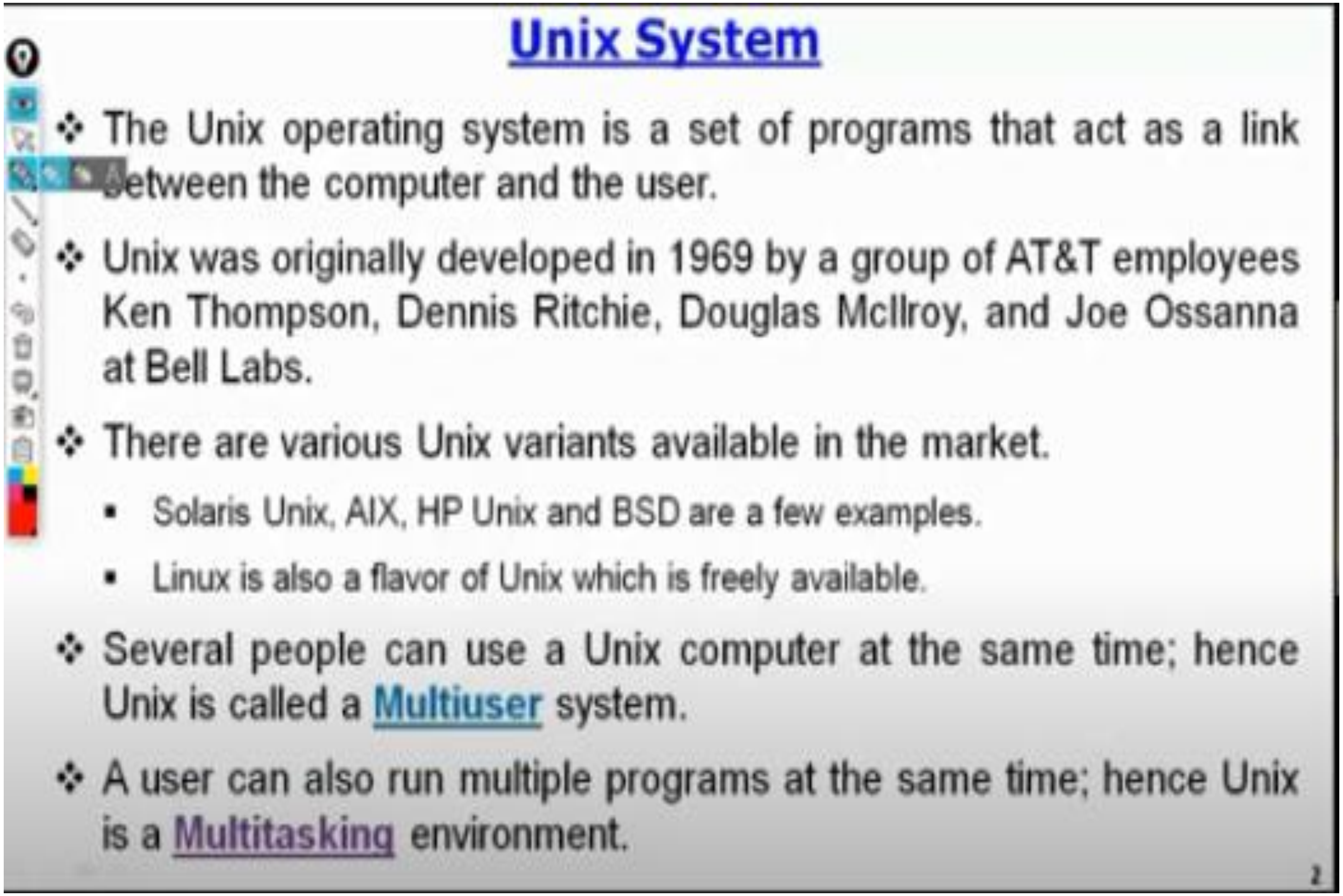
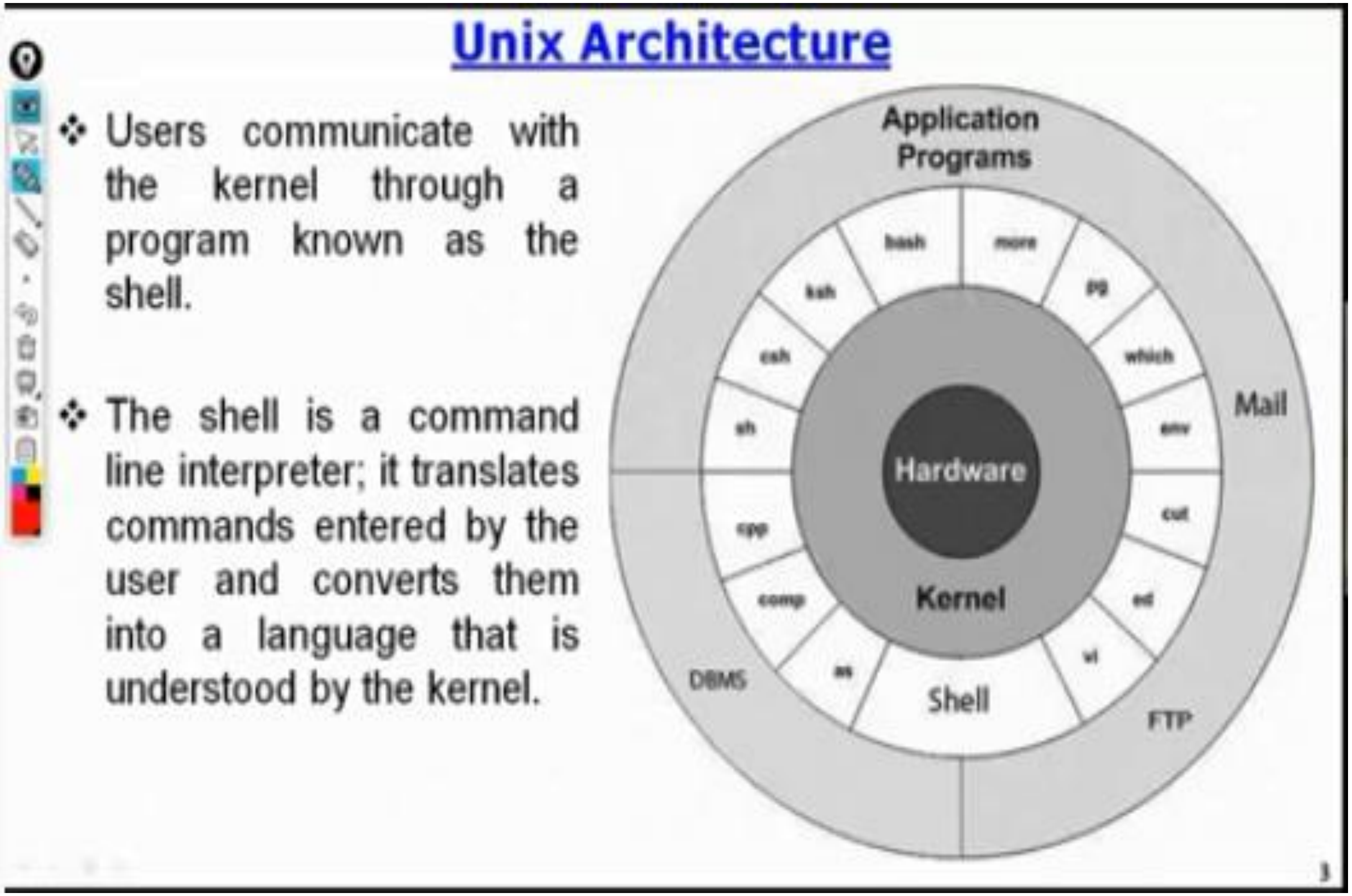


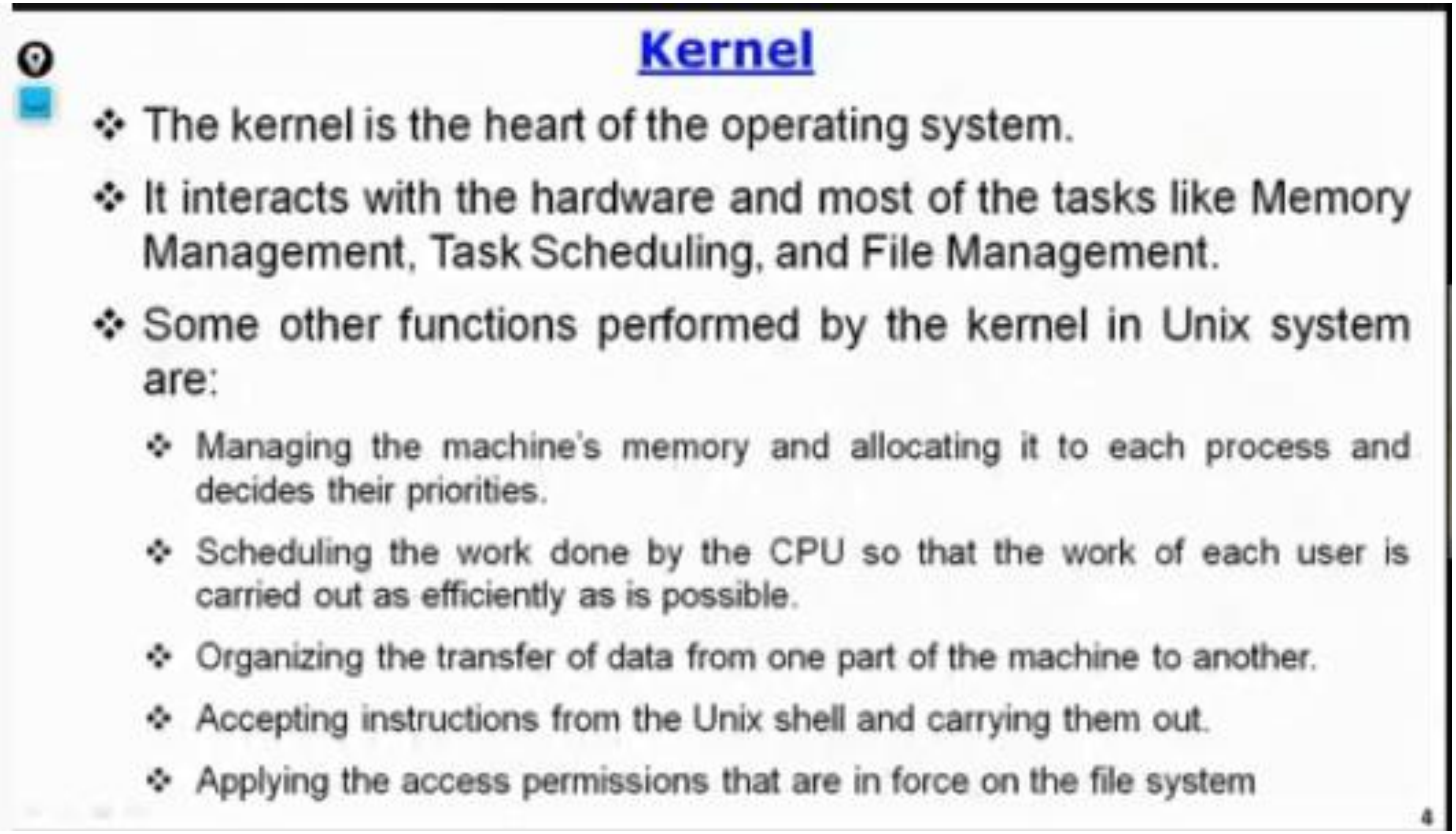
Unix System

- The Unix operating system is a set of programs that act as a link between the computer and the user.
- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
- There are various Unix variants available in the market.
  - Solaris Unix, AIX, HP Unix and BSD are a few examples.
  - Linux is also a flavor of Unix which is freely available.
- Several people can use a Unix computer at the same time; hence Unix is called a **Multiuser** system.
- A user can also run multiple programs at the same time; hence Unix is a **Multitasking** environment.



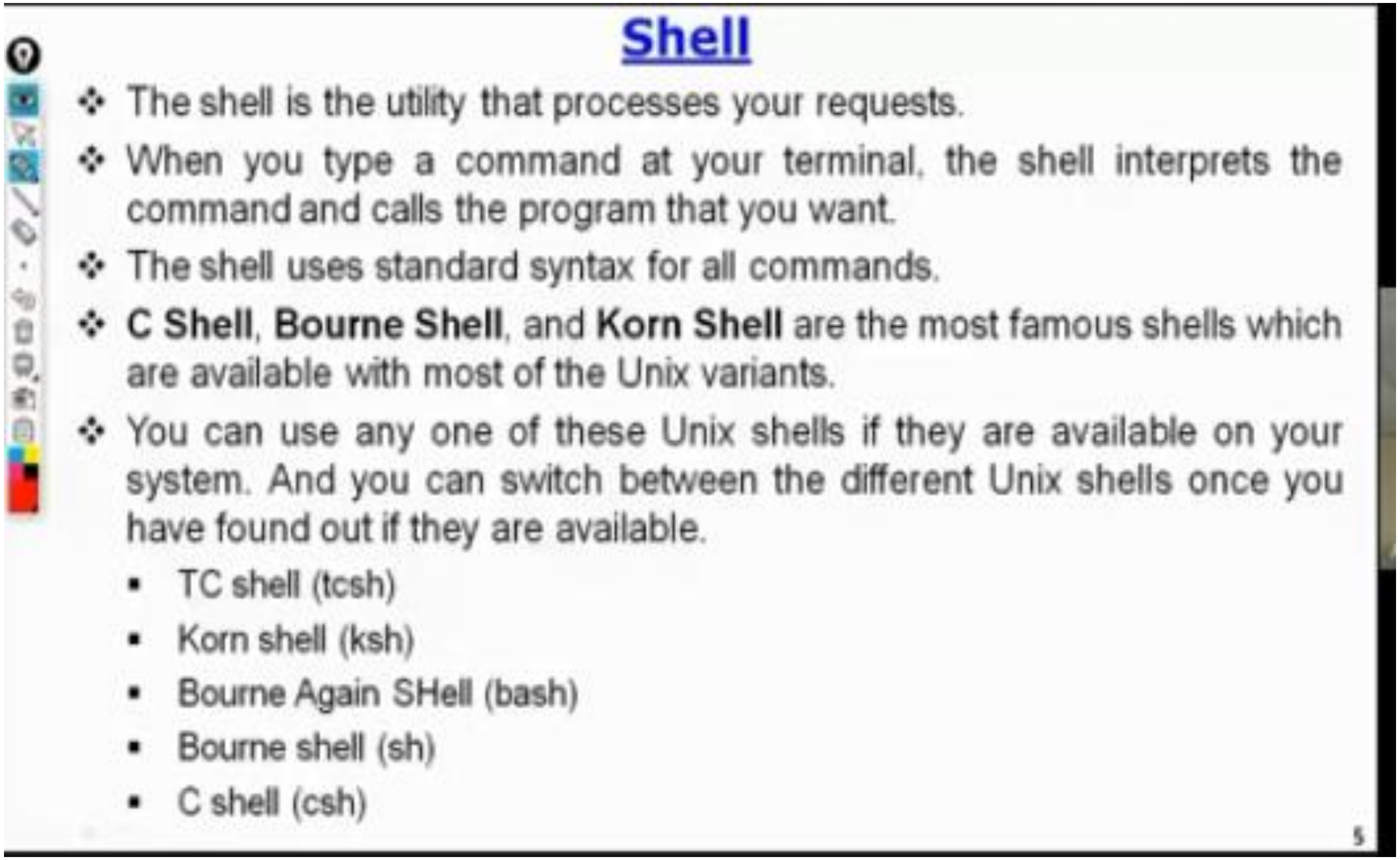
Unix Architecture

- Users communicate with the kernel through a program known as the shell.
- The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.



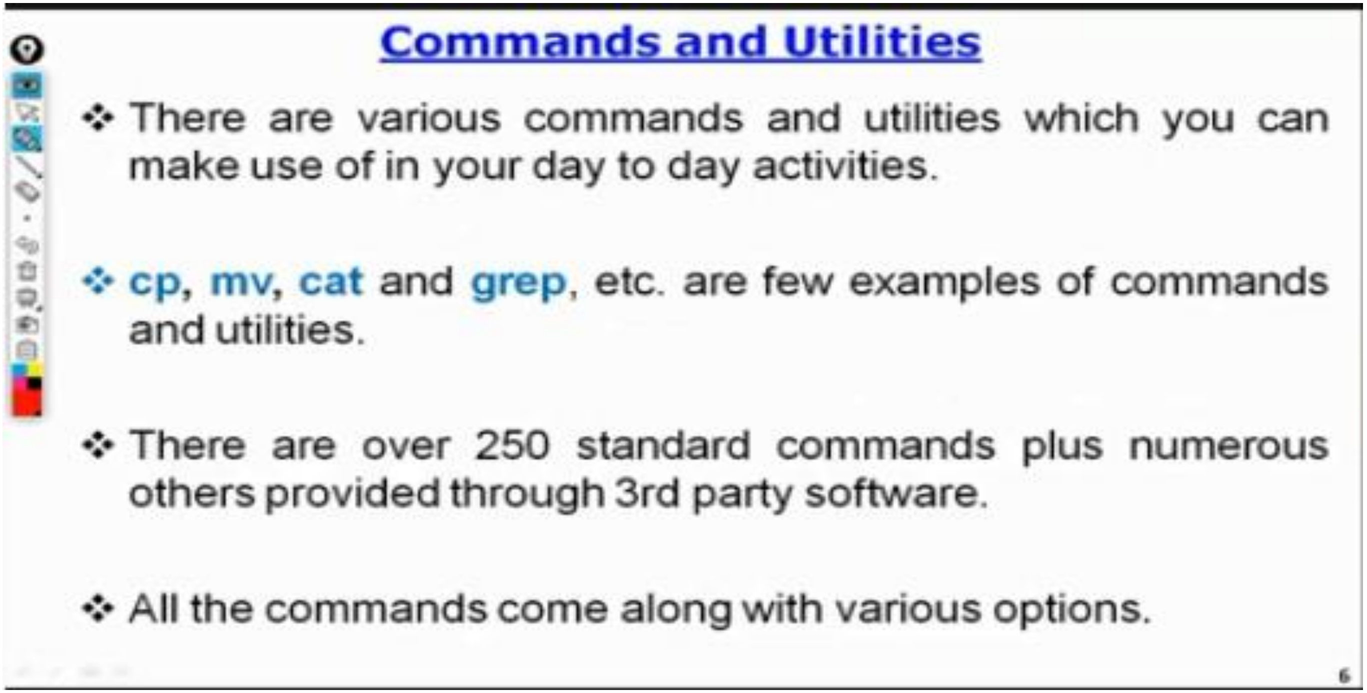
Kernel

- The kernel is the heart of the operating system.
- It interacts with the hardware and most of the tasks like Memory Management, Task Scheduling, and File Management.
- Some other functions performed by the kernel in Unix system are:
  - Managing the machine's memory and allocating it to each process and decides their priorities.
  - Scheduling the work done by the CPU so that the work of each user is carried out as efficiently as is possible.
  - Organizing the transfer of data from one part of the machine to another.
  - Accepting instructions from the Unix shell and carrying them out.
  - Applying the access permissions that are in force on the file system



Shell

- The shell is the utility that processes your requests.
- When you type a command at your terminal, the shell interprets the command and calls the program that you want.
- The shell uses standard syntax for all commands.
- **C Shell**, **Bourne Shell**, and **Korn Shell** are the most famous shells which are available with most of the Unix variants.
- You can use any one of these Unix shells if they are available on your system. And you can switch between the different Unix shells once you have found out if they are available.
  - TC shell (tcsh)
  - Korn shell (ksh)
  - Bourne Again SHell (bash)
  - Bourne shell (sh)
  - C shell (csh)



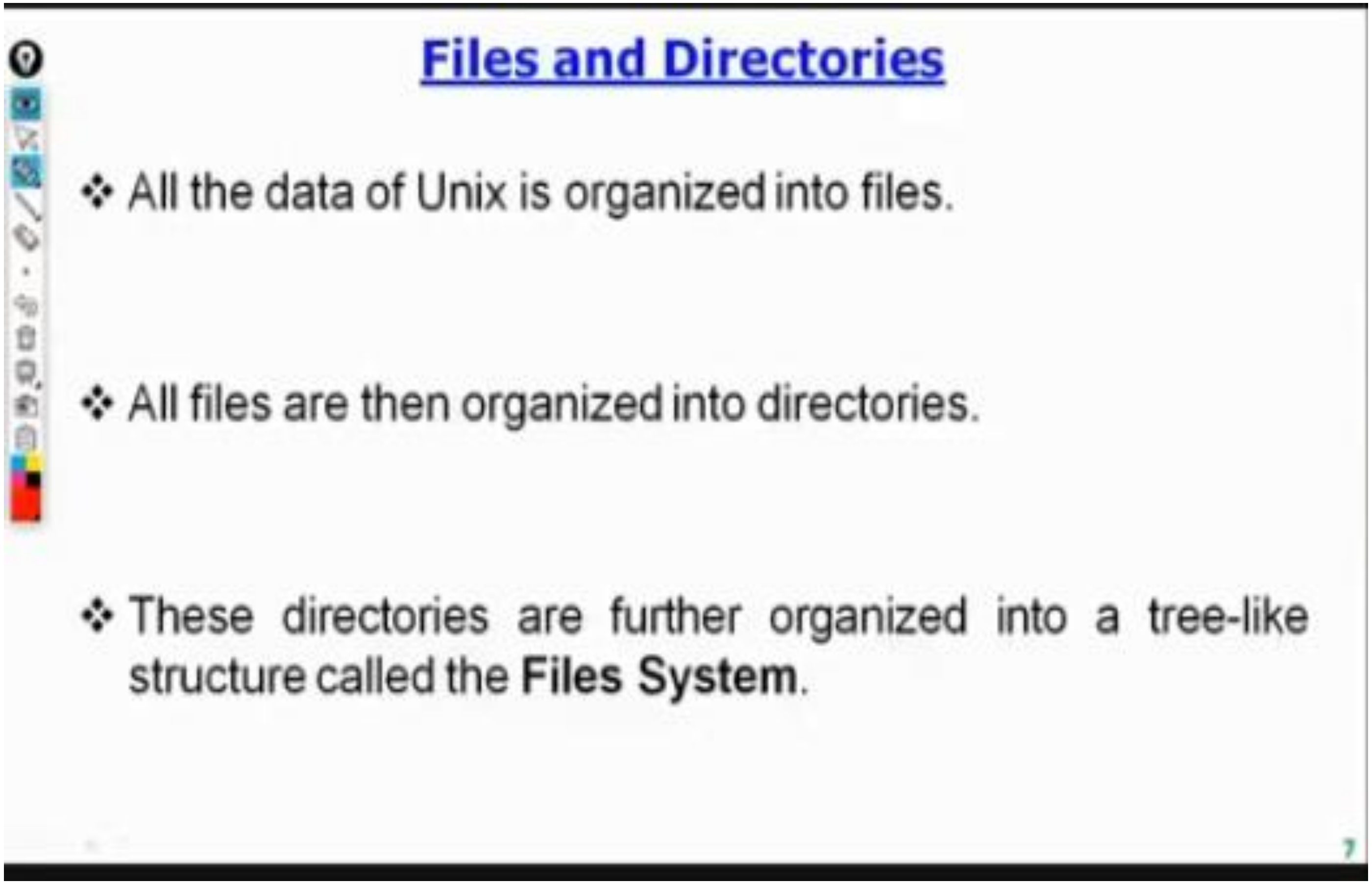
Commands and Utilities

- There are various commands and utilities which you can make use of in your day to day activities.

- **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities.

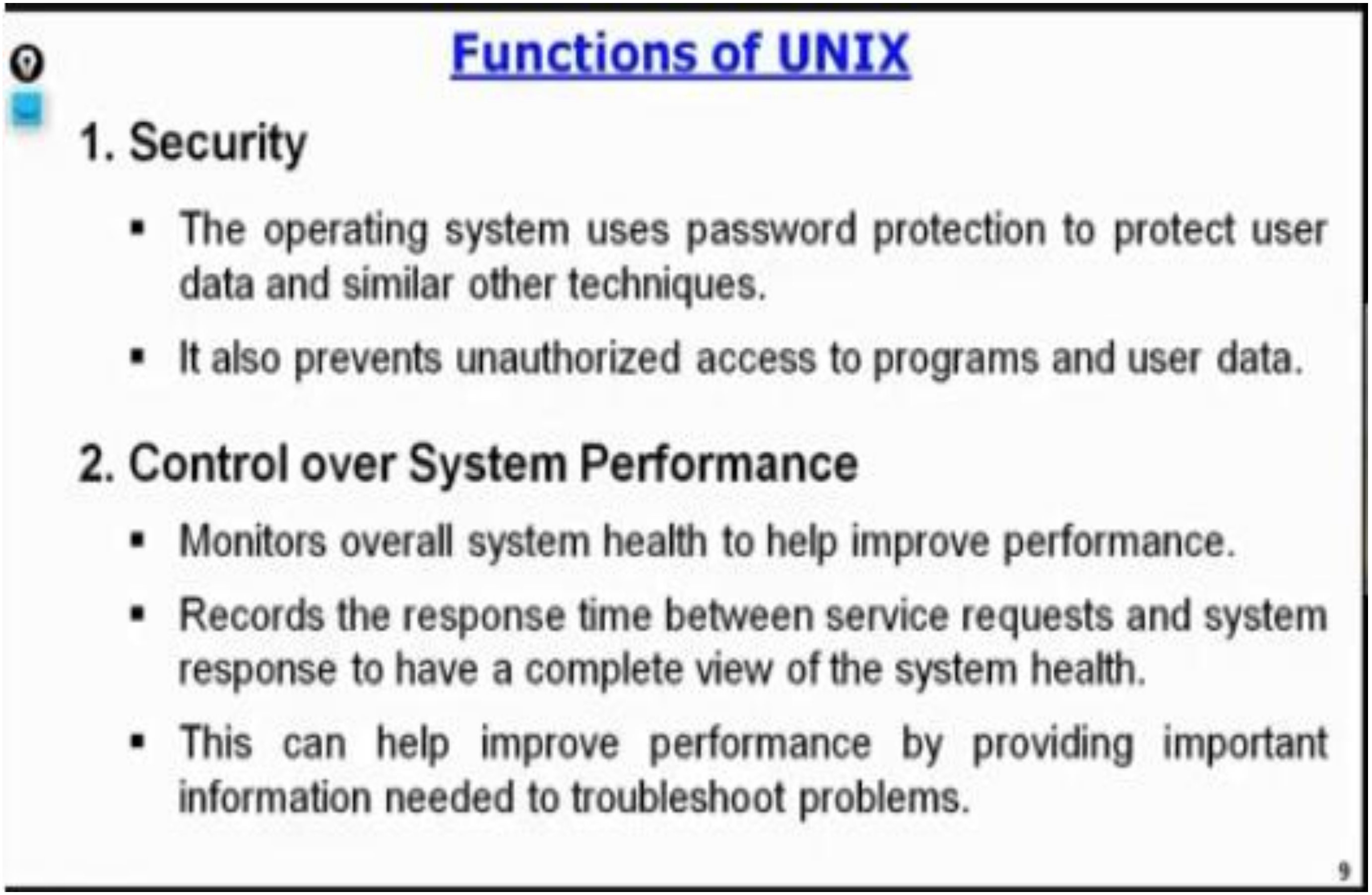
- There are over 250 standard commands plus numerous others provided through 3rd party software.

- All the commands come along with various options.



Files and Directories

- ❖ All the data of Unix is organized into files.
- ❖ All files are then organized into directories.
- ❖ These directories are further organized into a tree-like structure called the **Files System**.



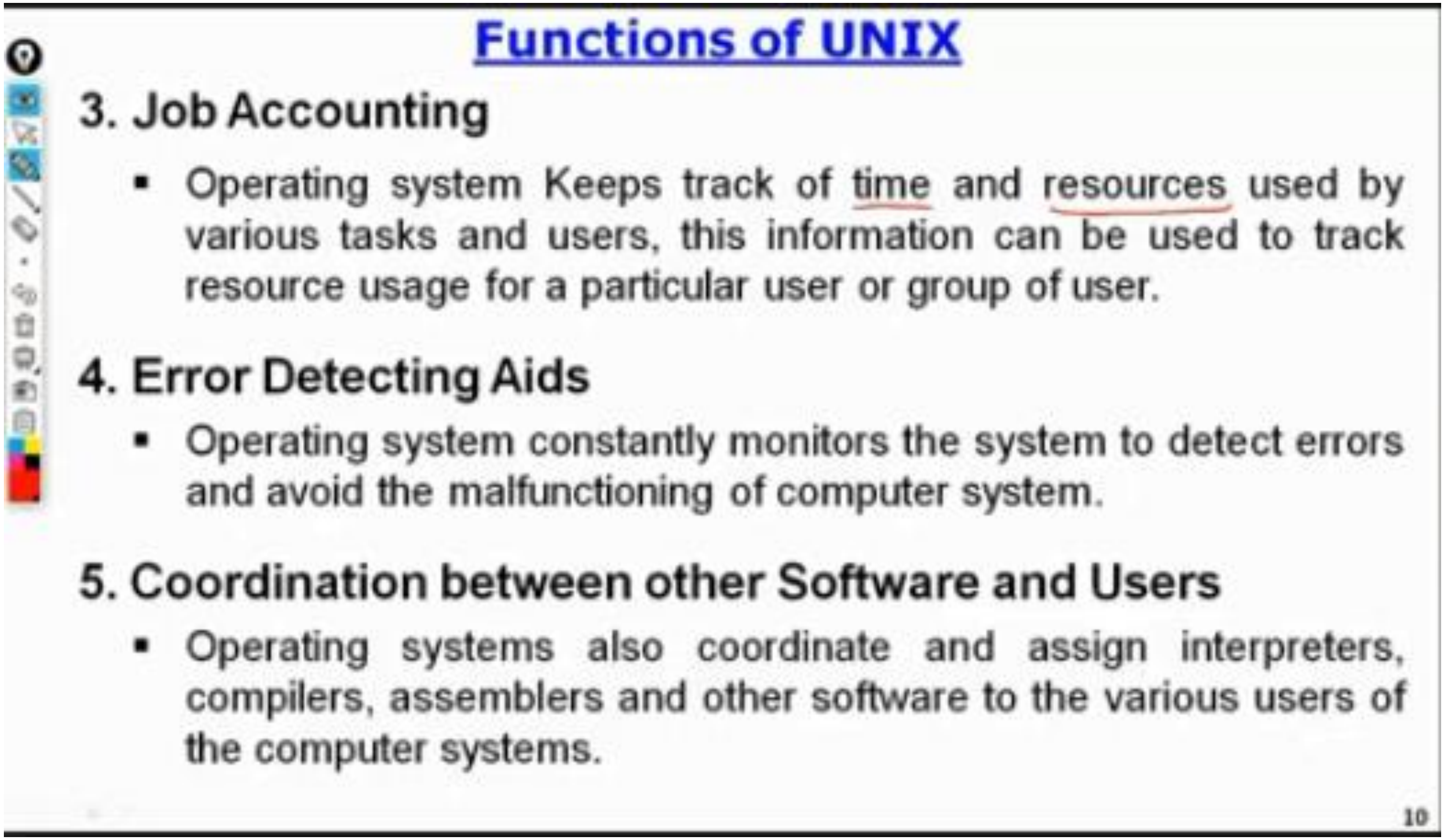
Functions of UNIX

1. Security

- The operating system uses password protection to protect user data and similar other techniques.
- It also prevents unauthorized access to programs and user data.

2. Control over System Performance

- Monitors overall system health to help improve performance.
- Records the response time between service requests and system response to have a complete view of the system health.
- This can help improve performance by providing important information needed to troubleshoot problems.



Functions of UNIX

3. Job Accounting

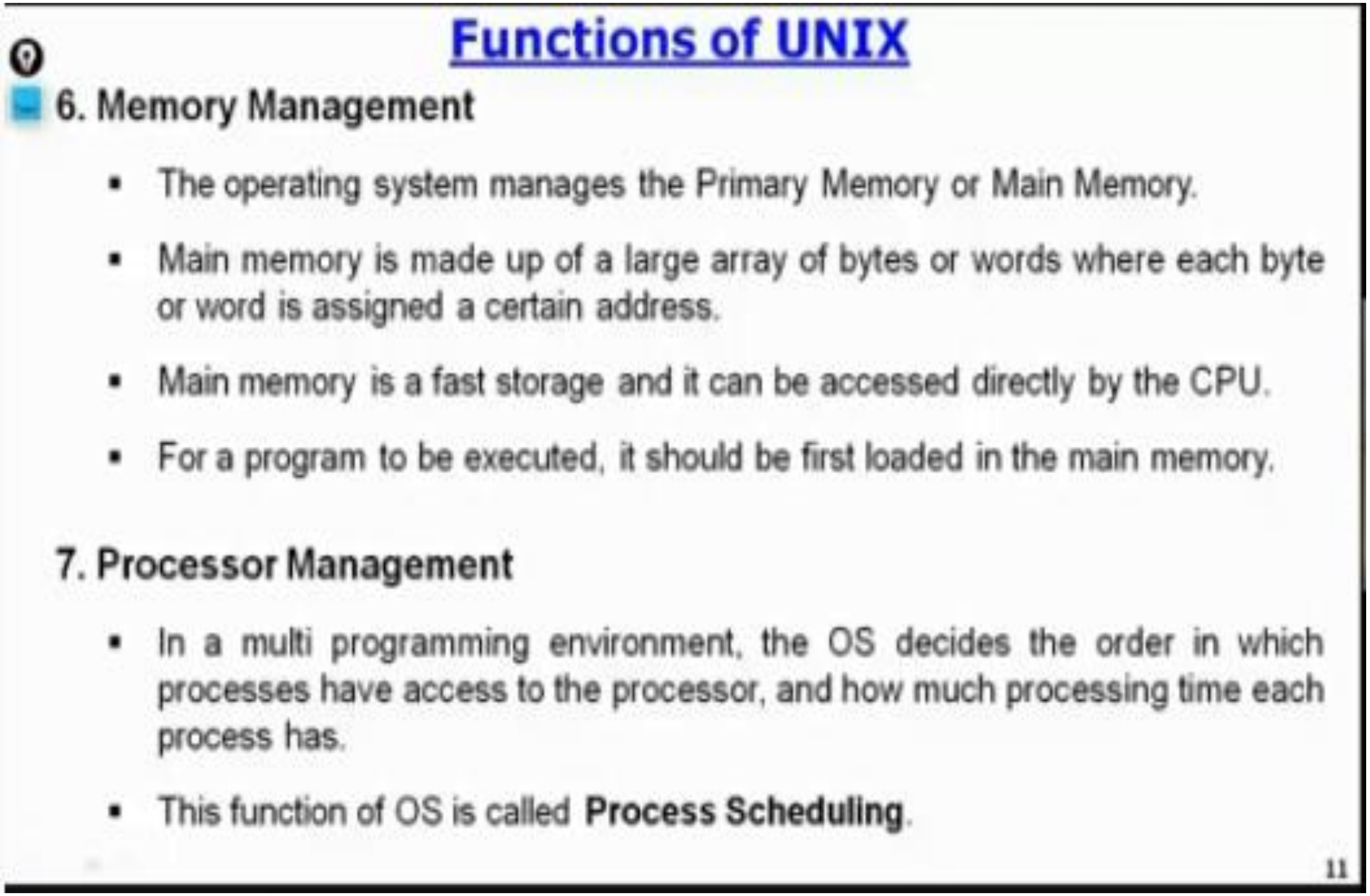
- Operating system Keeps track of time and resources used by various tasks and users, this information can be used to track resource usage for a particular user or group of user.

4. Error Detecting Aids

- Operating system constantly monitors the system to detect errors and avoid the malfunctioning of computer system.

5. Coordination between other Software and Users

- Operating systems also coordinate and assign interpreters, compilers, assemblers and other software to the various users of the computer systems.



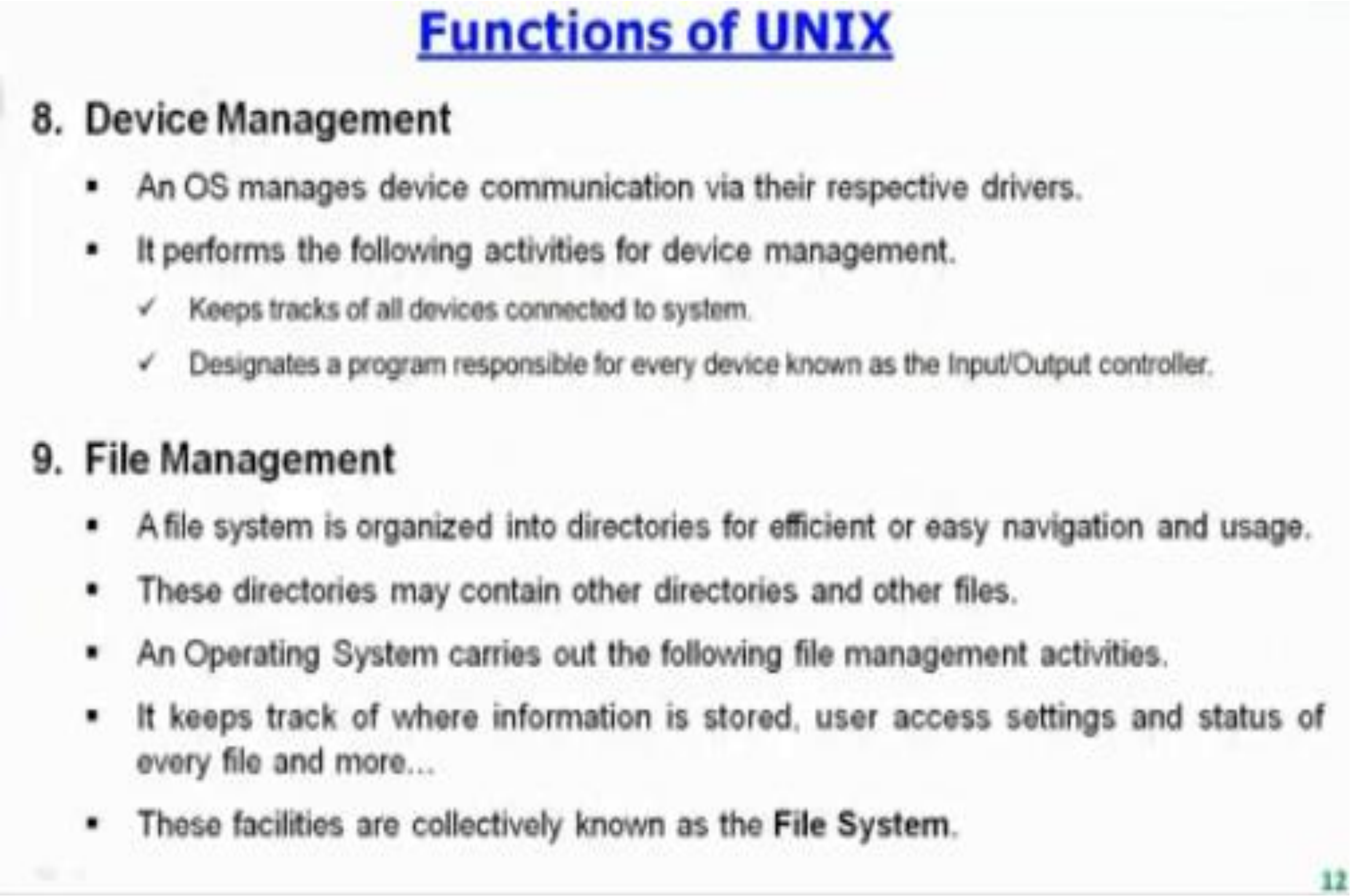
Functions of UNIX

6. Memory Management

- The operating system manages the Primary Memory or Main Memory.
- Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address.
- Main memory is a fast storage and it can be accessed directly by the CPU.
- For a program to be executed, it should be first loaded in the main memory.

7. Processor Management

- In a multi programming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has.
- This function of OS is called **Process Scheduling**.



Functions of UNIX

8. Device Management

- An OS manages device communication via their respective drivers.
- It performs the following activities for device management.
  - ✓ Keeps tracks of all devices connected to system.
  - ✓ Designates a program responsible for every device known as the Input/Output controller.

9. File Management

- A file system is organized into directories for efficient or easy navigation and usage.
- These directories may contain other directories and other files.
- An Operating System carries out the following file management activities.
- It keeps track of where information is stored, user access settings and status of every file and more...
- These facilities are collectively known as the **File System**.

Definition of an Operating System (OS)

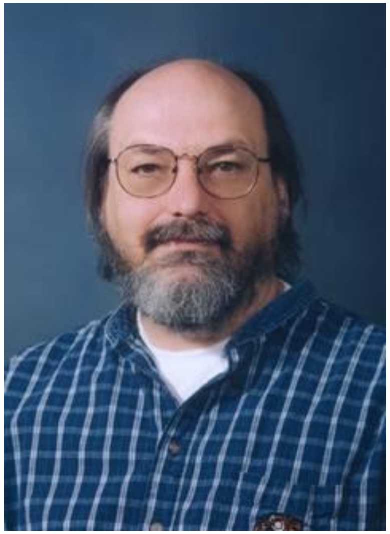
An operating system is a control program for a computer that performs the following operations:

- allocates computer resources
- schedules routine tasks
- provides a platform to run application software for users to accomplish tasks
- provides an interface between the user & the computer

History

- The Unix OS was developed by Ken Thompson at the AT&T Bell Laboratories in 1969.

He wanted to create a multi-user operating system to run “space wars” game.



History

- The system is divided into two parts
  - **Program and Services** : readily apparent to users, including such programs as the shell,mail,text processing packages and sccs(source code control system).
  - **Operating System** : that supports these programs and services.

Multics(Multiplexed Information and Computing Service :

- Ken’s philosophy was to create an operating system with commands or “utilities” that would do one thing well (i.e. **UNIX**). Pipes could be used combine commands.
- Aim to provide
  - simultaneous computer access to a large community of users,
  - to supply ample computation power and data storage,
  - to allow users to share their data easily, if required.

- Its primitive version was running on a GE645 computer by 1969.But it fails:
  - It did not provide the general service computing for which it was developed and goals were not satisfied.

Multics(Multiplexed Information and Computing Service :

- And Bell laboratories ended its participation in the project.

- To improve their programming environment ,Ken Thompson, Dennis Ritchie and others sketched a paper design of a file system that later evolved into an early version of the UNIX file system.
- Thompson wrote programs that simulated the behaviour of the proposed file system and of programs in a demand paging environment and encodes a simple kernel for the GE645 computer.

- Thompson wrote a game “space travel” in FORTRAN for a GECOS system.
- But it fails :
  - Program was unsatisfactory
    - It was difficult to control the “space ship” and expensive to run.

- He later used PDP-7 computer :
  - Good graphic display
  - Cheap executing power
  - And using space travel in PDP-7 helps them to learn about the m/c.
  - But its environment for program development required cross assembly of the program on the GECOS m/c and carrying paper tape for input to the PDP-7

- To create a better development environment, Thompson and Ritchie implemented their system design on the PDP-7 including :
  - Unix file system
  - Process subsystem
  - A small set of utility programs
  - ***Unics* (UNiplexed Information and Computing Service)**
    - **Now the new name evolved UNIX by Brian Kernighan.**

- Its success depends on implementation on real projects.
  - The unix system was moved to PDP-11 in 1971.
    - The system was characterized by its small size:
      - 16K bytes for the system
        - » 8k byets for user programs
        - » A disk of 512K bytes
        - » Limit of 64K bytes per file.

- After early success:
- Thompson set out a Fortran compiler for the new system,but instead came up with the lang B,influenced by BCPL.
- B was an interpretive language with the performance drawbacks ,so Ritchie developed it into one he called C,allowing generation of M/c code,declaration of data types and definition of data structures.

- In 1973, the OS was rewritten in C, the number of installations at Bell Laboratories grew to about 25 and UNIX System Group was formed to provide internal support.
- AT&T signed with a federal Govt. and allows to use UNIX system to universities who requested it for educational purposes.

- In 1974,Thompson and Ritchie published a paper describing the UNIX system in the communication of the ACM, giving further impetus to its acceptance.
- 1977,the no of Unix system sites had grown to about 500,of which 125 were in universities.
- Now it became popular in the operating telephone companies, providing a good environment for program development n/w transactions operation services and real time services.
- 1977,interactive systems corporation became the first value added reseller of a unix system ,enhancing it for use in office automation environments.

- It was ported into non-PDP m/c : interdata 8/32.
- Unix system III
- Unix system V
- Unix 4.3 BSD for VAX m/c
- In 1984:1,00,000 Unix system installation made in world wide web.

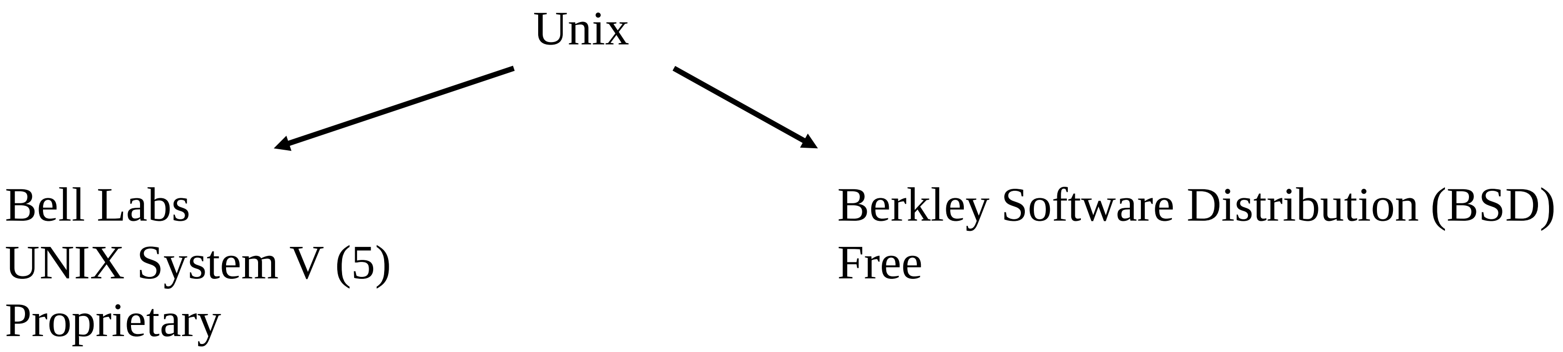
Development of Unix OS

Unix became a popular OS among institutions such as colleges & universities through a 4-year “try before you buy” deal.

- Efficient and inexpensive way of networking
- promotes Internet use and file-sharing
- Open system allows for source code to be shared among many programmers - allows for better coordination among programmers

Development of Unix OS

Students at University of California (in Berkley) further developed the UNIX operating system and introduced the BSD version of Unix



Unix

Bell Labs
UNIX System V (5)
Proprietary

Berkley Software Distribution (BSD)
Free

Development of Unix OS

There were versions of UNIX for the Personal Computer (PC), such as Microsoft announced its first Unix for 16-bit microcomputers called XENIX, etc., but they didn’t catch on in popularity until Linux was developed in the early 90’s.

History of Linux

- Linux operating system developed by programming student Linus Torvalds
- Linus wanted to develop Unix-like OS just to experiment with new 386 computer at the time...

Why Has Linux Become so Popular?

Linus decided to make Linux OS source-code for Linux Kernel open to all:

- Unlike traditional Operating Systems, anyone can modify and distribute Linux OS (as long as they distribute source code of Linux Kernel)
- "Competition among Hackers" allow code to be improved and distributed often
- Many users can spot bugs in the operating system or application if source code is "open"

Why Has Linux Become so Popular?

Other Factors:

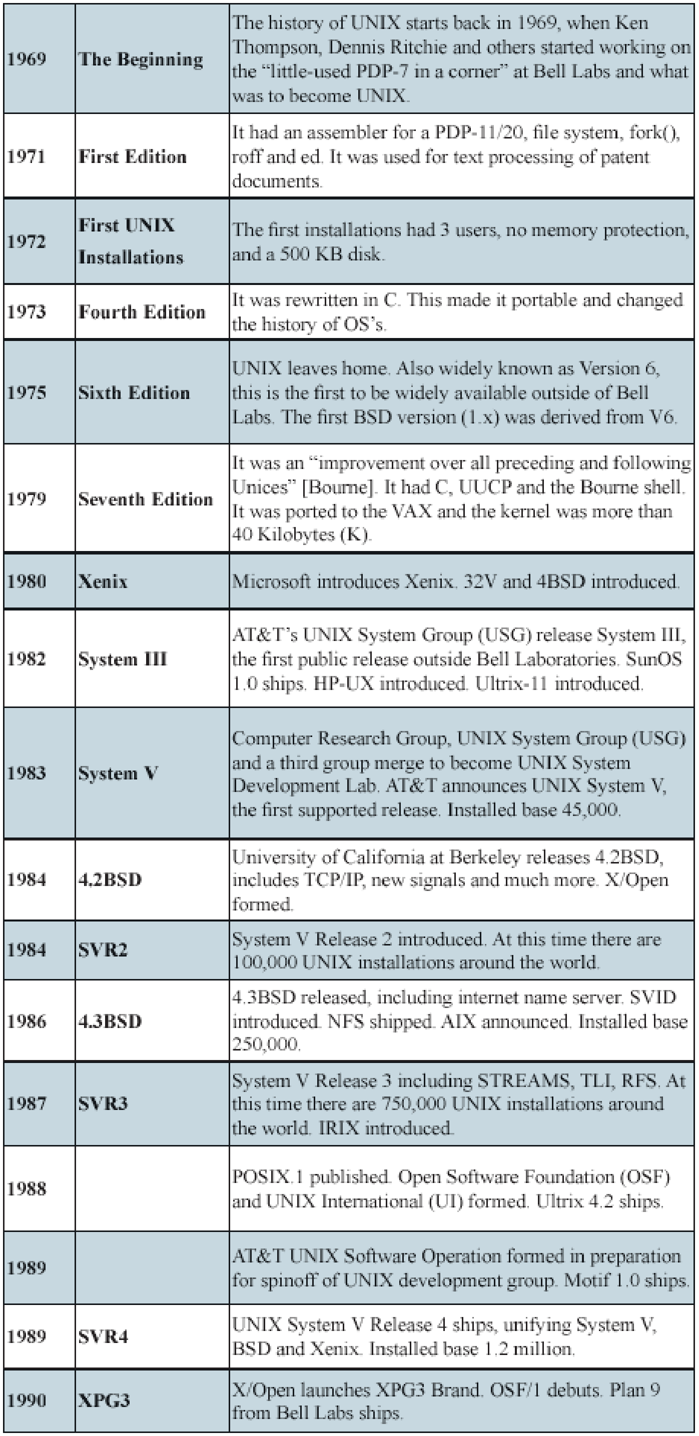
- PC's have increased processing power and a there has been a noted shift from mainframes and minicomputers to PCs.
- Since Linux is a "Unix Work-alike", this OS has a reputation to be a very stable platform for networking (creating at-home servers) and running / maintaining applications.
- Agencies such as Free Software Foundation created GNU project to provide free software.

Concerns

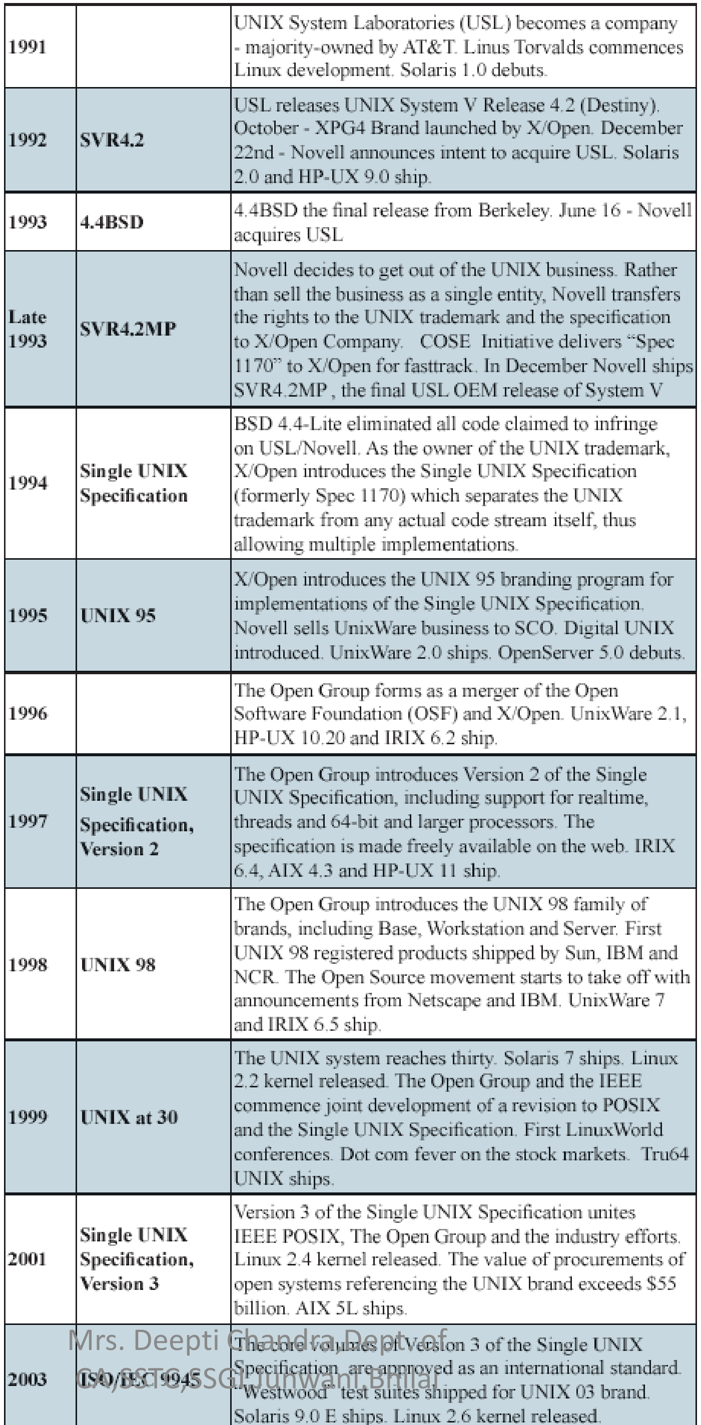
- Some people claim that “there are as many version of Linux as there are users…”
- POSIX (Portable Operating System Interface for Computer Environments) is a government standard to ensure consistency among different UNIX and Linux versions.
- Many versions of Linux are approaching POSIX standard.

Concerns

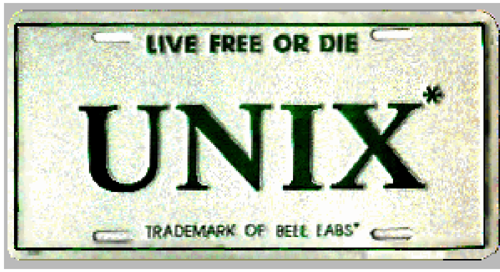
- Freedom of allowing Linux users to create “servers” connected up to Internet can lead to attacks from experienced hackers.
- Linux commands may be considered “user-unfriendly” although GUIs are now used.
- Prior reputation for difficult install process including the loss of data on other hard disk partitions.



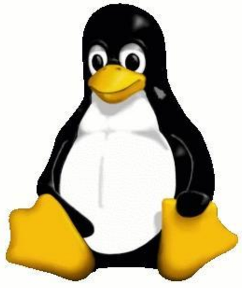
1969	**The Beginning**	The history of UNIX starts back in 1969, when Ken Thompson, Dennis Ritchie and others started working on the "little-used PDP-7 in a corner" at Bell Labs and what was to become UNIX.
1971	**First Edition**	It had an assembler for a PDP-11/20, file system, fork(), roff and ed. It was used for text processing of patent documents.
1972	**First UNIX Installations**	The first installations had 3 users, no memory protection, and a 500 KB disk.
1973	**Fourth Edition**	It was rewritten in C. This made it portable and changed the history of OS's.
1975	**Sixth Edition**	UNIX leaves home. Also widely known as Version 6, this is the first to be widely available outside of Bell Labs. The first BSD version (1.x) was derived from V6.
1979	**Seventh Edition**	It was an "improvement over all preceding and following Unices" [Bourne]. It had C, UUCP and the Bourne shell. It was ported to the VAX and the kernel was more than 40 Kilobytes (K).
1980	**Xenix**	Microsoft introduces Xenix. 32V and 4BSD introduced.
1982	**System III**	AT&T's UNIX System Group (USG) release System III, the first public release outside Bell Laboratories. SunOS 1.0 ships. HP-UX introduced. Ultrix-11 introduced.
1983	**System V**	Computer Research Group, UNIX System Group (USG) and a third group merge to become UNIX System Development Lab. AT&T announces UNIX System V, the first supported release. Installed base 45,000.
1984	**4.2BSD**	University of California at Berkeley releases 4.2BSD, includes TCP/IP, new signals and much more. X/Open formed.
1984	**SVR2**	System V Release 2 introduced. At this time there are 100,000 UNIX installations around the world.
1986	**4.3BSD**	4.3BSD released, including internet name server. SVID introduced. NFS shipped. AIX announced. Installed base 250,000.
1987	**SVR3**	System V Release 3 including STREAMS, TLI, RFS. At this time there are 750,000 UNIX installations around the world. IRIX introduced.
1988		POSIX.1 published. Open Software Foundation (OSF) and UNIX International (UI) formed. Ultrix 4.2 ships.
1989		AT&T UNIX Software Operation formed in preparation for spinoff of UNIX development group. Motif 1.0 ships.
1989	**SVR4**	UNIX System V Release 4 ships, unifying System V, BSD and Xenix. Installed base 1.2 million.
1990	**XPG3**	X/Open launches XPG3 Brand. OSF/1 debuts. Plan 9 from Bell Labs ships.



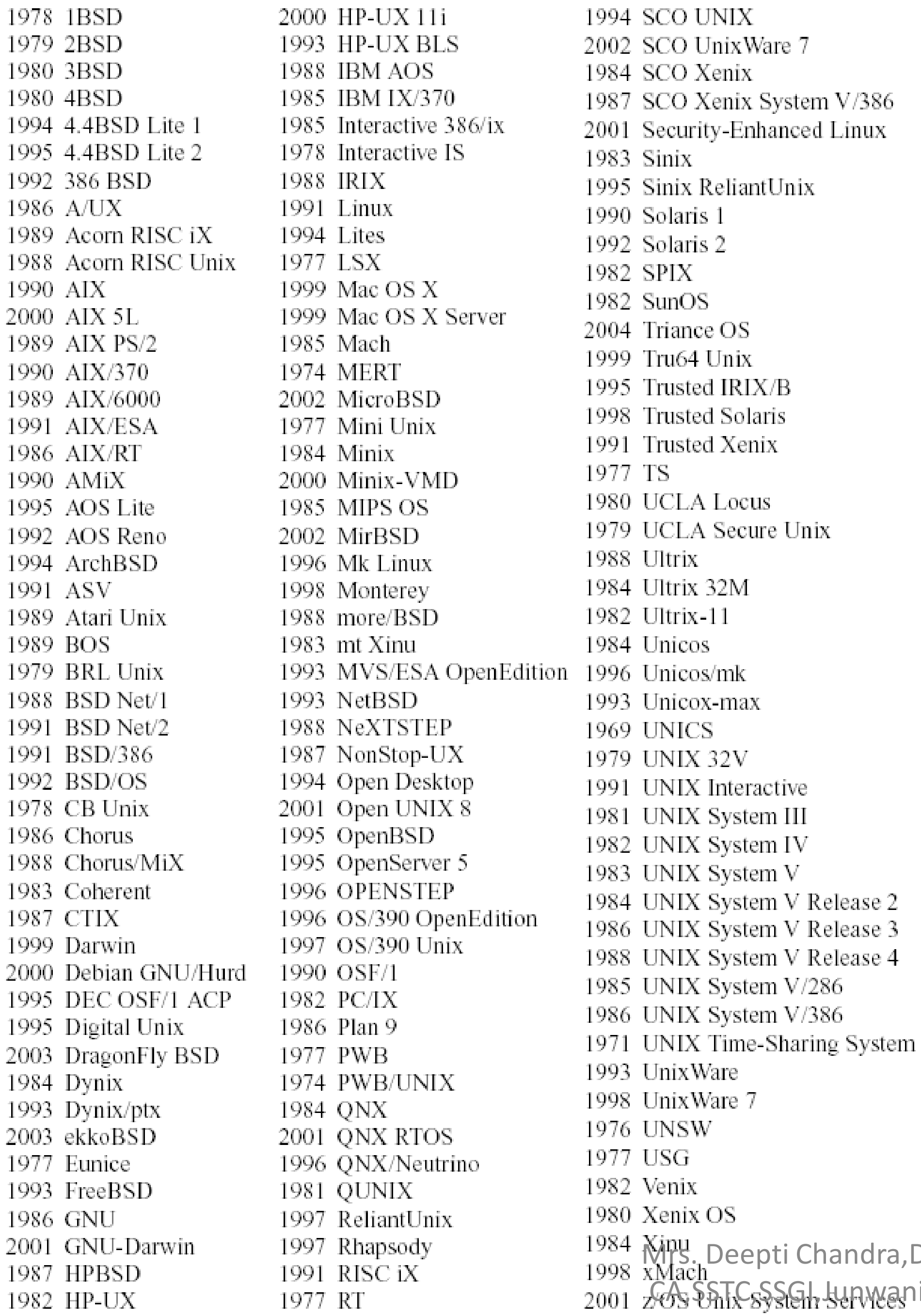
1991		UNIX System Laboratories (USL) becomes a company - majority-owned by AT&T. Linus Torvalds commences Linux development. Solaris 1.0 debuts.
1992	**SVR4.2**	USL releases UNIX System V Release 4.2 (Destiny). October - XPG4 Brand launched by X/Open. December 22nd - Novell announces intent to acquire USL. Solaris 2.0 and HP-UX 9.0 ship.
1993	**4.4BSD**	4.4BSD the final release from Berkeley. June 16 - Novell acquires USL
Late 1993	**SVR4.2MP**	Novell decides to get out of the UNIX business. Rather than sell the business as a single entity, Novell transfers the rights to the UNIX trademark and the specification to X/Open Company. COSE Initiative delivers "Spec 1170" to X/Open for fasttrack. In December Novell ships SVR4.2MP , the final USL OEM release of System V
1994	**Single UNIX Specification**	BSD 4.4-Lite eliminated all code claimed to infringe on USL/Novell. As the owner of the UNIX trademark, X/Open introduces the Single UNIX Specification (formerly Spec 1170) which separates the UNIX trademark from any actual code stream itself, thus allowing multiple implementations.
1995	**UNIX 95**	X/Open introduces the UNIX 95 branding program for implementations of the Single UNIX Specification. Novell sells UnixWare business to SCO. Digital UNIX introduced. UnixWare 2.0 ships. OpenServer 5.0 debuts.
1996		The Open Group forms as a merger of the Open Software Foundation (OSF) and X/Open. UnixWare 2.1, HP-UX 10.20 and IRIX 6.2 ship.
1997	**Single UNIX Specification, Version 2**	The Open Group introduces Version 2 of the Single UNIX Specification, including support for realtime, threads and 64-bit and larger processors. The specification is made freely available on the web. IRIX 6.4, AIX 4.3 and HP-UX 11 ship.
1998	**UNIX 98**	The Open Group introduces the UNIX 98 family of brands, including Base, Workstation and Server. First UNIX 98 registered products shipped by Sun, IBM and NCR. The Open Source movement starts to take off with announcements from Netscape and IBM. UnixWare 7 and IRIX 6.5 ship.
1999	**UNIX at 30**	The UNIX system reaches thirty. Solaris 7 ships. Linux 2.2 kernel released. The Open Group and the IEEE commence joint development of a revision to POSIX and the Single UNIX Specification. First LinuxWorld conferences. Dot com fever on the stock markets. Tru64 UNIX ships.
2001	**Single UNIX Specification, Version 3**	Version 3 of the Single UNIX Specification unites IEEE POSIX, The Open Group and the industry efforts. Linux 2.4 kernel released. The value of procurements of open systems referencing the UNIX brand exceeds $55 billion. AIX 5L ships.
2003	**ISO/IEC 9945**	The core volumes of Version 3 of the Single UNIX Specification, are approved as an international standard. "Westwood" test suites shipped for UNIX 03 brand. Solaris 9.0 E ships. Linux 2.6 kernel released.



LIVE FREE OR DIE
http://www.UNIX-systems.org
UNIX®
UNIX is a Registered Trademark of The Open Group



Unix versions



1978	1BSD
1979	2BSD
1980	3BSD
1980	4BSD
1994	4.4BSD Lite 1
1995	4.4BSD Lite 2
1992	386 BSD
1986	A/UX
1989	Acorn RISC iX
1988	Acorn RISC Unix
1990	AIX
2000	AIX 5L
1989	AIX PS/2
1990	AIX/370
1989	AIX/6000
1991	AIX/ESA
1986	AIX/RT
1990	AMiX
1995	AOS Lite
1992	AOS Reno
1994	ArchBSD
1991	ASV
1989	Atari Unix
1989	BOS
1979	BRL Unix
1988	BSD Net/1
1991	BSD Net/2
1991	BSD/386
1992	BSD/OS
1978	CB Unix
1986	Chorus
1988	Chorus/MiX
1983	Coherent
1987	CTIX
1999	Darwin
2000	Debian GNU/Hurd
1995	DEC OSF/1 ACP
1995	Digital Unix
2003	DragonFly BSD
1984	Dynix
1993	Dynix/ptx
2003	ekkoBSD
1977	Eunice
1993	FreeBSD
1986	GNU
2001	GNU-Darwin
1987	HPBSD
1982	HP-UX
2000	HP-UX 11i
1993	HP-UX BLS
1988	IBM AOS
1985	IBM IX/370
1985	Interactive 386/ix
1978	Interactive IS
1988	IRIX
1991	Linux
1994	Lites
1977	LSX
1999	Mac OS X
1999	Mac OS X Server
1985	Mach
1974	MERT
2002	MicroBSD
1977	Mini Unix
1984	Minix
2000	Minix-VMD
1985	MIPS OS
2002	MirBSD
1996	Mk Linux
1998	Monterey
1988	more/BSD
1983	mt Xinu
1993	MVS/ESA OpenEdition
1993	NetBSD
1988	NeXTSTEP
1987	NonStop-UX
1994	Open Desktop
2001	Open UNIX 8
1995	OpenBSD
1995	OpenServer 5
1996	OPENSTEP
1996	OS/390 OpenEdition
1997	OS/390 Unix
1990	OSF/1
1982	PC/IX
1986	Plan 9
1977	PWB
1974	PWB/UNIX
1984	QNX
2001	QNX RTOS
1996	QNX/Neutrino
1981	QUNIX
1997	ReliantUnix
1997	Rhapsody
1991	RISC iX
1977	RT
1994	SCO UNIX
2002	SCO UnixWare 7
1984	SCO Xenix
1987	SCO Xenix System V/386
2001	Security-Enhanced Linux
1983	Sinix
1995	Sinix ReliantUnix
1990	Solaris 1
1992	Solaris 2
1982	SPIX
1982	SunOS
2004	Triance OS
1999	Tru64 Unix
1995	Trusted IRIX/B
1998	Trusted Solaris
1991	Trusted Xenix
1977	TS
1980	UCLA Locus
1979	UCLA Secure Unix
1988	Ultrix
1984	Ultrix 32M
1982	Ultrix-11
1984	Unicos
1996	Unicos/mk
1993	Unicox-max
1969	UNICS
1979	UNIX 32V
1991	UNIX Interactive
1981	UNIX System III
1982	UNIX System IV
1983	UNIX System V
1984	UNIX System V Release 2
1986	UNIX System V Release 3
1988	UNIX System V Release 4
1985	UNIX System V/286
1986	UNIX System V/386
1971	UNIX Time-Sharing System
1993	UnixWare
1998	UnixWare 7
1976	UNSW
1977	USG
1982	Venix
1980	Xenix OS
1984	Xinu
1998	xMach
2001	z/OS Unix System Services

Popular GNU/Linux distributions

- Debian (Ubuntu,...)
- RedHat / Fedora
- Suse
- Gentoo
- Knopix (loads from CD)
- Yellow Dog (PowerPC)
- Slackware
- etc.

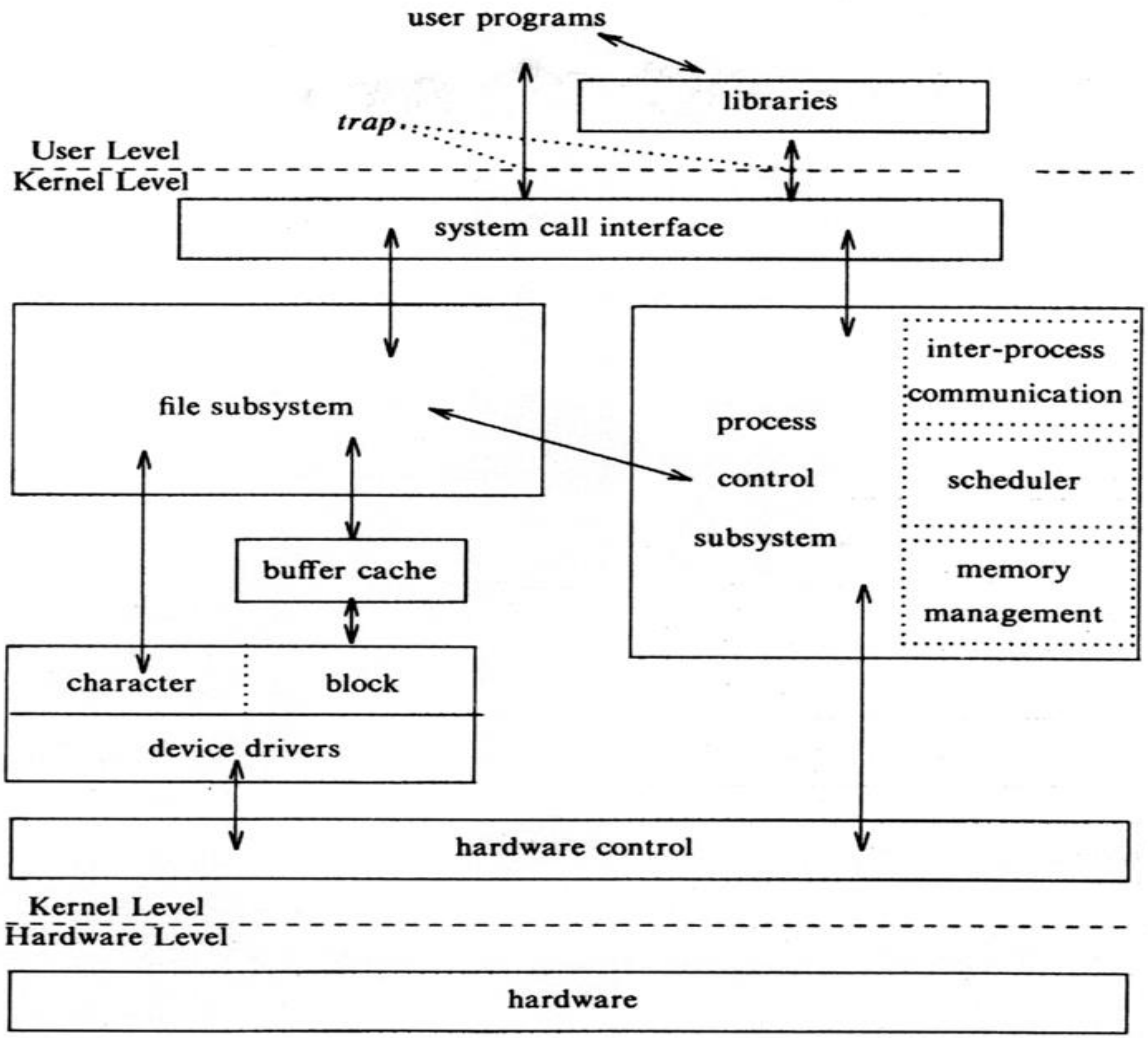
Popular Commercial Unix versions

- Solaris (Sun Microsystems)
- MacOS X (Apple)
- AIX (IBM)
- HP-UX (Hewlett Packard)

Architecture of the Unix o/s

(system kernel)

Block Diagram of System Kernel



figure

- Figure shows the file subsystem on the left and the process control subsystem on the right, the two major components of the kernel.

  figure shows three levels:

  User,kernel, and h/w.

  The System call and library interface represent the border between user programs and the kernel.

System call

System calls instruct the kernel to do various operations for the calling program (such as ls, who, date etc) and exchange data between kernel and the program

- The figure partition the set of system calls

into those that interact with the file subsystem and those that interact with the process control subsystem.

the file subsystem manages files,allocating file space, administering free space, controlling access to files, and retrieving data for users.

File subsystem

- Process interact with the file subsystem via a specific set of system calls, such as open, close, read, write, stat, chown, and chmod.

The file subsystem access file data using a buffering mechanism that regulates data flow between the kernel and secondary storage device.

The buffering mechanism interacts with block i/o device driver to initiate data transfer to and from the kernel.

Character device

- The file subsystem also interacts directly with “raw” i/o device driver without the intervention of a buffering mechanism.

Raw devices, sometimes called character devices, include all device that are not block devices.

Process control subsystem

- It is responsible for process synchronization, interprocess communication, memory management, and process scheduling.

some of the system calls for controlling process are fork, exec, exit, wait, brk, and signal.

Memory management

- This module controls the allocation of memory

If at any time the system does not have enough physical memory for all processes, the kernel moves them between main memory and secondary memory so that all processes get a fair chance to execute.

There are two policies for managing memory

1) Swapping 2) Demand paging

scheduler

- The swapper process is sometimes called the scheduler.

This module allocates

the CPU to process. it schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel preempts them when their recent run time exceeds a time quantum.

The scheduler then choose the highest priority eligible process to run.

Interprocess communication

- There are several forms of interprocess communication, ranging from asynchronous signaling of events to synchronous transmission of messages between process.

Hardware control

- It is responsible for handling interrupts and for communicating with the m/c.

Devices such as disks or terminals may interrupt the CPU while a process is executing. If so, the kernel may resume the execution of interrupted process after servicing the interrupt.

Overview of file subsystem

- The internal representation of a file is given by an inode, which contains a description of the disk layout of the file data and other information such as the file owner, access permissions, and access times.

  The term inode is a contraction term of the **index node**.

  Every file has one inode, but it may have several names, all of which map into the inode. Each name is called a link.

File Subsystem

- A file system is a collection of files and directories on a disk or tape in standard UNIX file system format.
- Inode: when a file is newly created an unused inode assigned to that file. it contains info for that file.

Kernel Data Structures :

1. File Table
2. UDFT
3. Inode Table

File table

- The kernel have a global data structure, called file table, to store information of file access.
- Each entry in file table contains:
  A. a pointer to in-core inode table
  B. the offset of next read or write in the file
  C. access rights (r/w) allowed to the opening process.
  D. count.

User File Descriptor table

- Each process has a user file descriptor table to identify all opened files.
- An entry in user file descriptor table points to an entry of kernel's global file table.
- Entry 0: standard input
- Entry 1: standard output
- Entry 2: error output

System Call: open

- open: A process may open an existing file to read or write
- syntax:

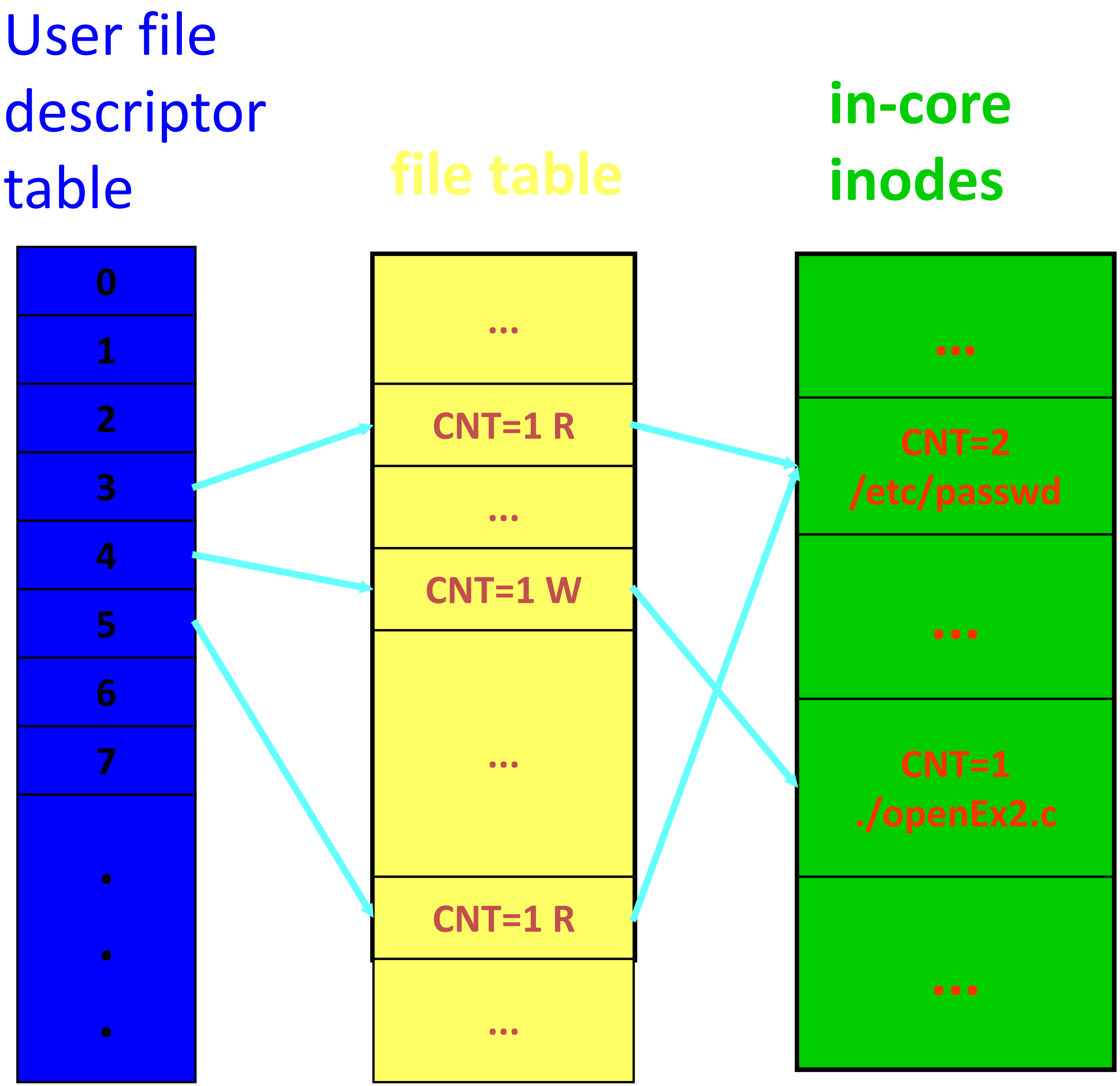
  fd = open(pathname, mode);

  A. pathname is the filename to be opened

  B. mode: read/write
- Example

```
$ cc openEx1.c -o openEx1
$ openEx1
Before open ...
fd1=3  fd2=4  fd3=5
$
```

```
main()
{
  int fd1, fd2, fd3;
  printf("Before open ...\n");
  fd1 = open("/etc/passwd", O_RDONLY);
  fd2 = open("./openEx1.c", O_WRONLY);
  fd3 = open("/etc/passwd", O_RDONLY);
  printf("fd1=%d  fd2=%d  fd3=%d \n", fd1, fd2, fd3);
}
```



User file descriptor table

file table

in-core inodes

0

1

2

3

4

5

6

7

.

.

.

…

CNT=1 R

…

CNT=1 W

…

CNT=1 R

…

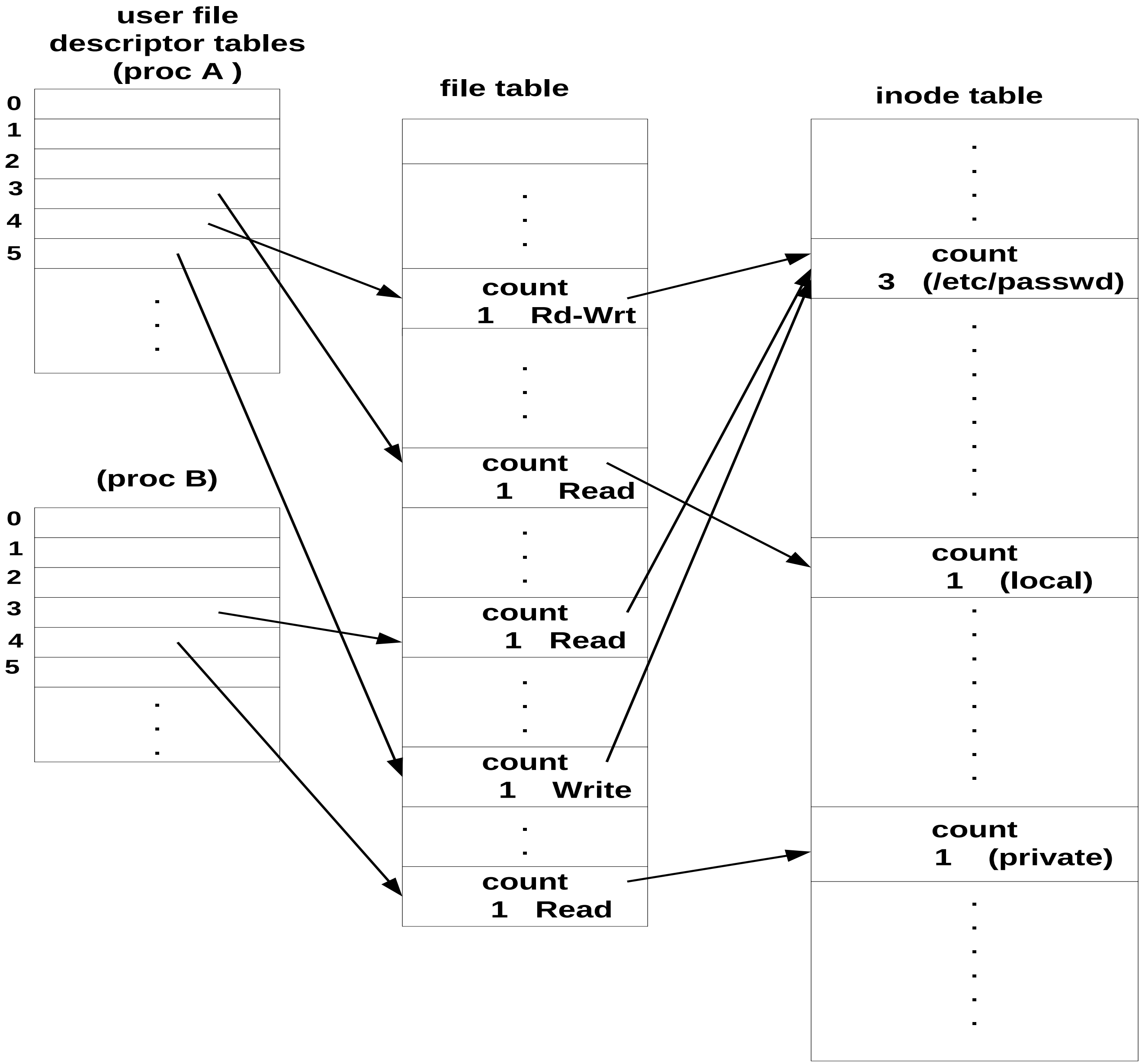
…

CNT=2 /etc/passwd

…

CNT=1 ./openEx2.c

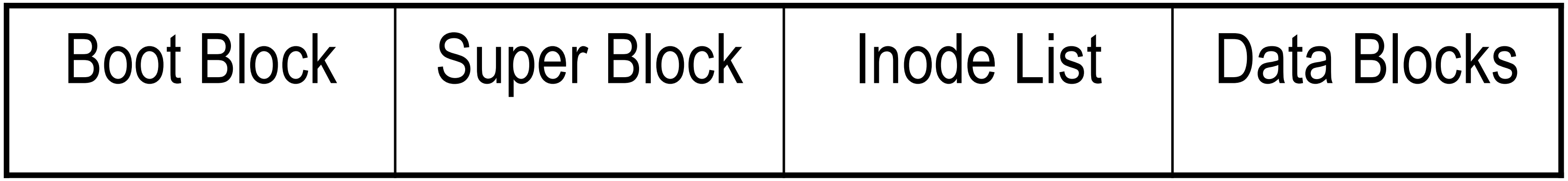
…


user file descriptor tables (proc A)
0
1
2
3
4
5
(proc B)
0
1
2
3
4
5
file table
count 1 Rd-Wrt
count 1 Read
count 1 Read
count 1 Write
count 1 Read
inode table
count 3 (/etc/passwd)
count 1 (local)
count 1 (private)

File System Layout

- A file system is consists of a sequence of logical blocks (512/1024 byte etc.)

- A file system has the following structure:



Boot Block | Super Block | Inode List | Data Blocks

File System: Boot Block

- The beginning of the file system
- Contains bootstrap code to load the operating system
- Initialize the operating system
- Typically occupies the first sector of the disk

File System: Super Block

- Describes the state of a file system
- Describes the size of the file system
  - How many files it can store
- Where to find free space on the file system
- Other information such as no of free inodes, free blocks ,lock field,flag etc.

File System: Inode List

- Inodes are used to access disk files.
- Inodes maps the disk files
- For each file there is an inode entry in the inode list block
- Inode list also keeps track of directory structure

I-node structure

- Type: file, directory, pipe, symbolic link
- Access: read/write/execute (owner, group,)
- owner: who own this I-node (file, directory, ...)
- timestamp: creation, modification, access time
- size: the number of bytes
- block count: the number of data blocks
- direct blocks: pointers to the data

In-core inode table

- An in-core inode contains
  - all the information of inode in disks.
  - status of in-core inode

    inode is locked,

    inode data changed

    file data changed.
  - the logic device number of the file system.
  - inode number
  - reference count

File System: Data Block

- Starts at the end of the inode list
- Contains disk files
- An allocated data block can belong to one and only one file in the file system
- A data block has 512 bytes.
- A data block may contains data of files or Administrative Data.

Process Subsystem

- Process subsystem describes the structure of a process and some process data structures used for memory management.
- Process state diagram and state transitions.

Processes(1)

- A process is the execution of a program
- A process is consists of text (machine code), data and stack
- Many process can run simultaneously as kernel schedules them for execution
- Several processes may be instances of one program
- A process reads and writes its data and stack sections, but it cannot read or write the data and stack of other processes
- A process communicates with other processes and the rest of the world via system calls

Processes(2)

- All processes in UNIX system, except the very first process (process 0) which is created by the system boot code, are created by the fork system call

System Call

- A process accesses system resources through system call.
- System call for
  - Process Control:
    - fork: create a new process
    - wait: allow a parent process to synchronize its execution with the exit of a child process.
    - exec: invoke a new program.
    - exit: terminate process execution
  - File system:

File: open, read, write, lseek, close

inode: chdir, chown chmod, stat fstat

others: pipe dup, mount, unmount, link, unlink

Processes

- A process has
  - ✵ text: machine instructions
    (may be shared by other processes)
  - ✵ data
  - ✵ stack
- Process may execute either in user mode and in kernel mode.

Data structure for processes

- Process information are stored in two places:
  - Process table
  - User table (U area)
- Kernel has a process table that keeps track of all active processes
- Each entry in the process table contains pointers to the text, data, stack and the U Area of a process.

Region

- A region is a contiguous area of a process’s address space such as text ,data and stack.

Process: Region Table

- Region table entries describes the attributes of the region, such as whether it contains text or data, whether it is shared or private
- The extra level from the per process region table to kernel region table allows independent processes to share regions.

Process Table

- Process table: an entry in process table has the following information:
  1. **process state:**
     A. running in user mode or kernel mode
     B. Ready in memory or Ready but swapped
     C. Sleep in memory or sleep and swapped
  2. **PID:** process id
  3. **UID:** user id (who owns the process)
  4. scheduling information
  5. signals that is sent to the process but not yet handled
  6. a pointer to per-process-region table
- There is a single process table for the entire system

User Table (u area)

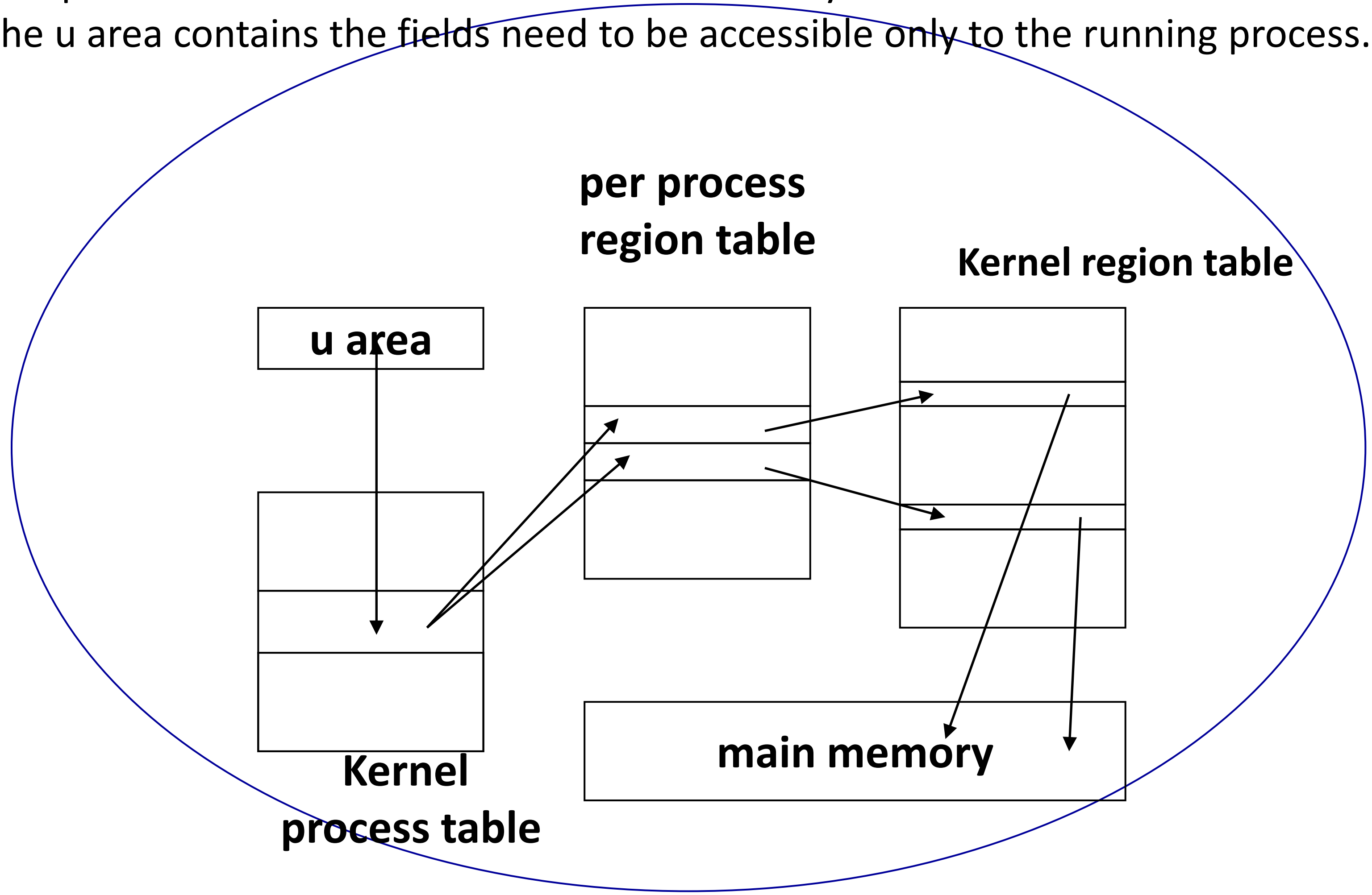
- Each process has only one private user table.
- **User table contains information that must be accessible while the process is in execution.**
- Fields of U Area
  1. A pointer to the process table slot
  2. parameters of the current system call, return values error codes
  3. file descriptors for all open files
  4. current directory and current root
  5. process and file size limits.
  6. I/O parameters

Process: U Area

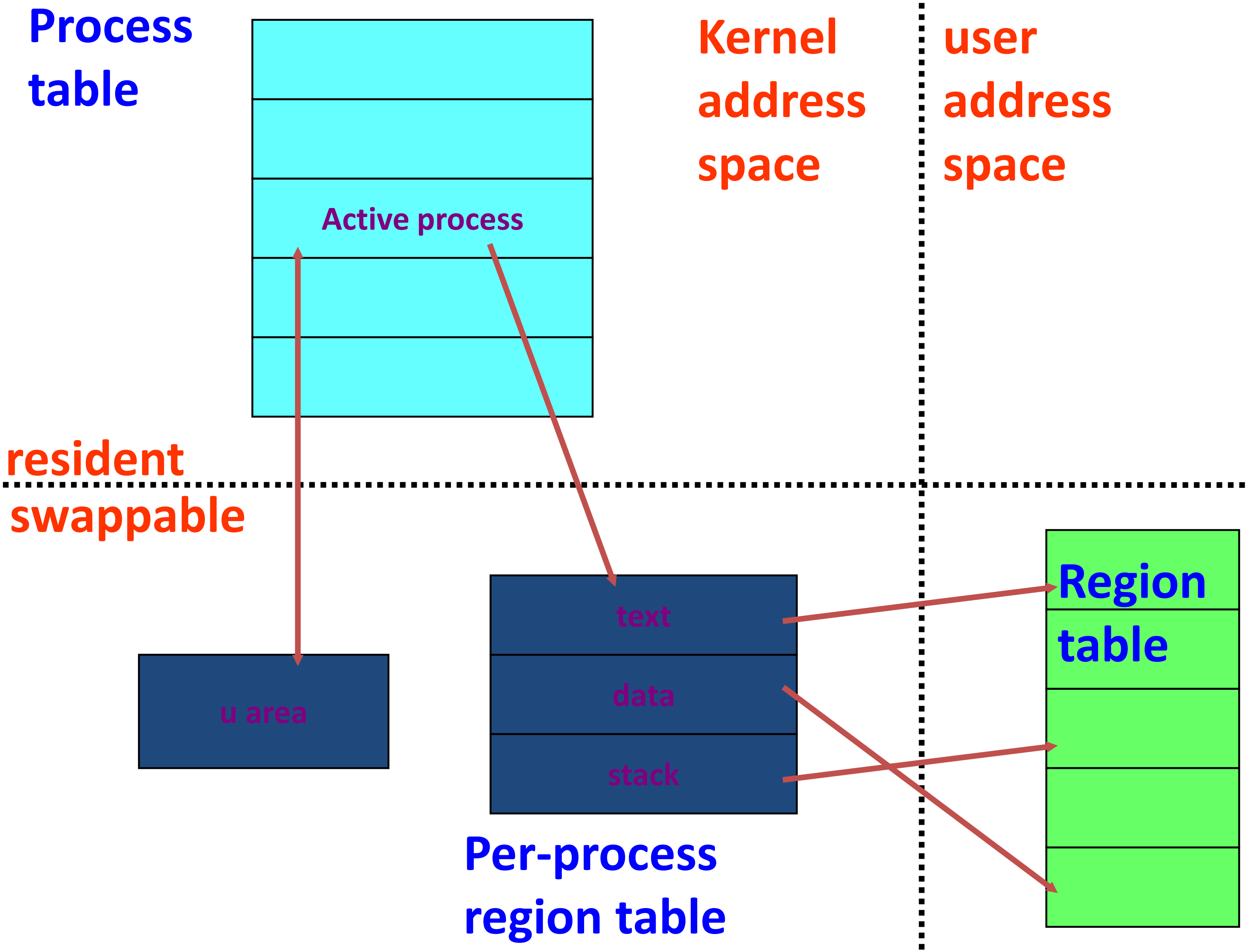
- Kernel can directly access fields of the U Area of the executing process but not of the U Area of other processes.
- The process table entry and the u area contain control and status information about the process.
- The u area contains information describing the process that needs to be accessible only when the process is executing.

Data Structure for Process

The process table contains fields that must always be accessible to the kernel and the u area contains the fields need to be accessible only to the running process.



Therefore ,the kernel allocates space for the u area only when creating a process



Process table

Active process

Kernel address space

user address space

resident

swappable

u area

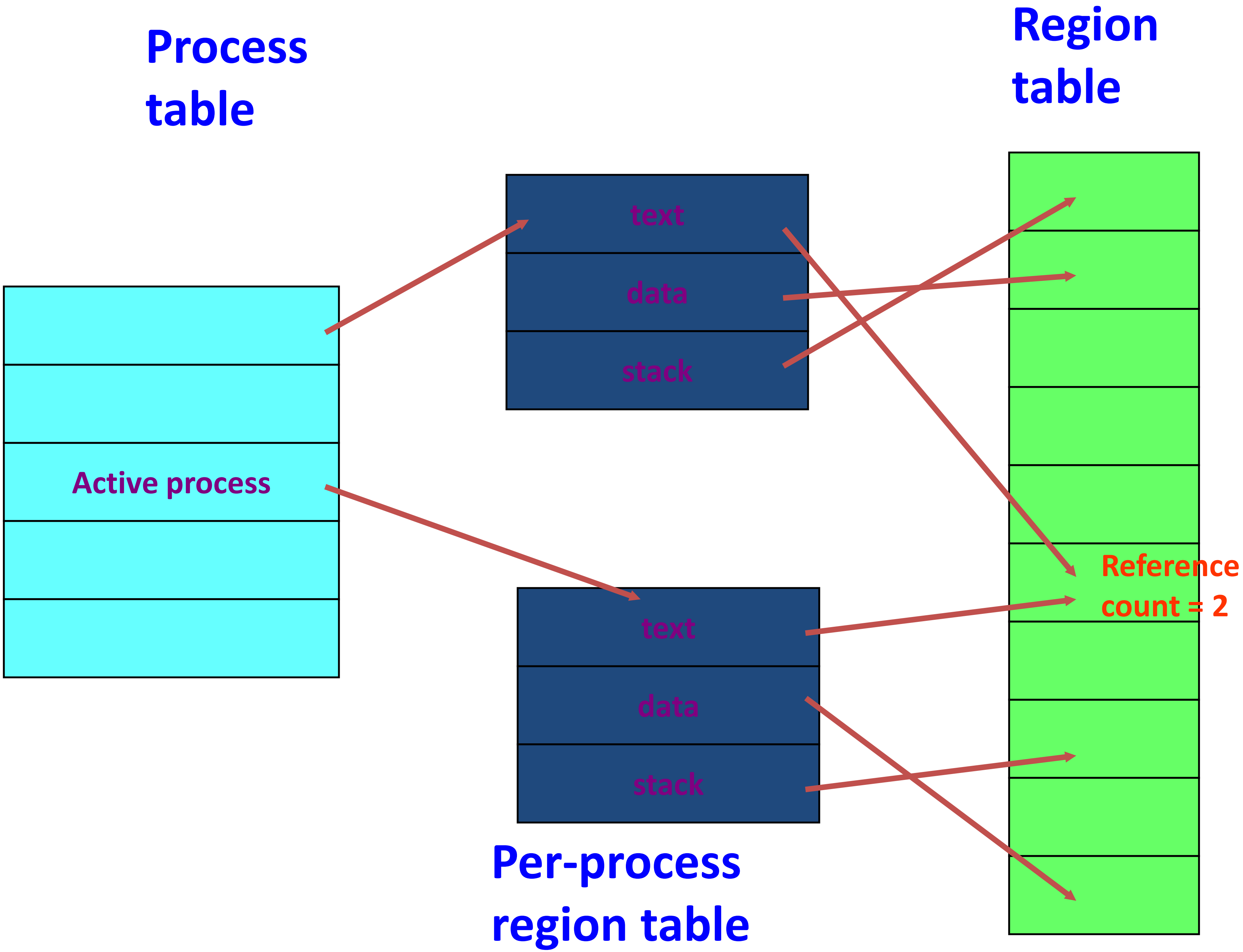
text

data

stack

Per-process region table

Region table



Process table

Region table

Active process

text

data

stack

text

data

stack

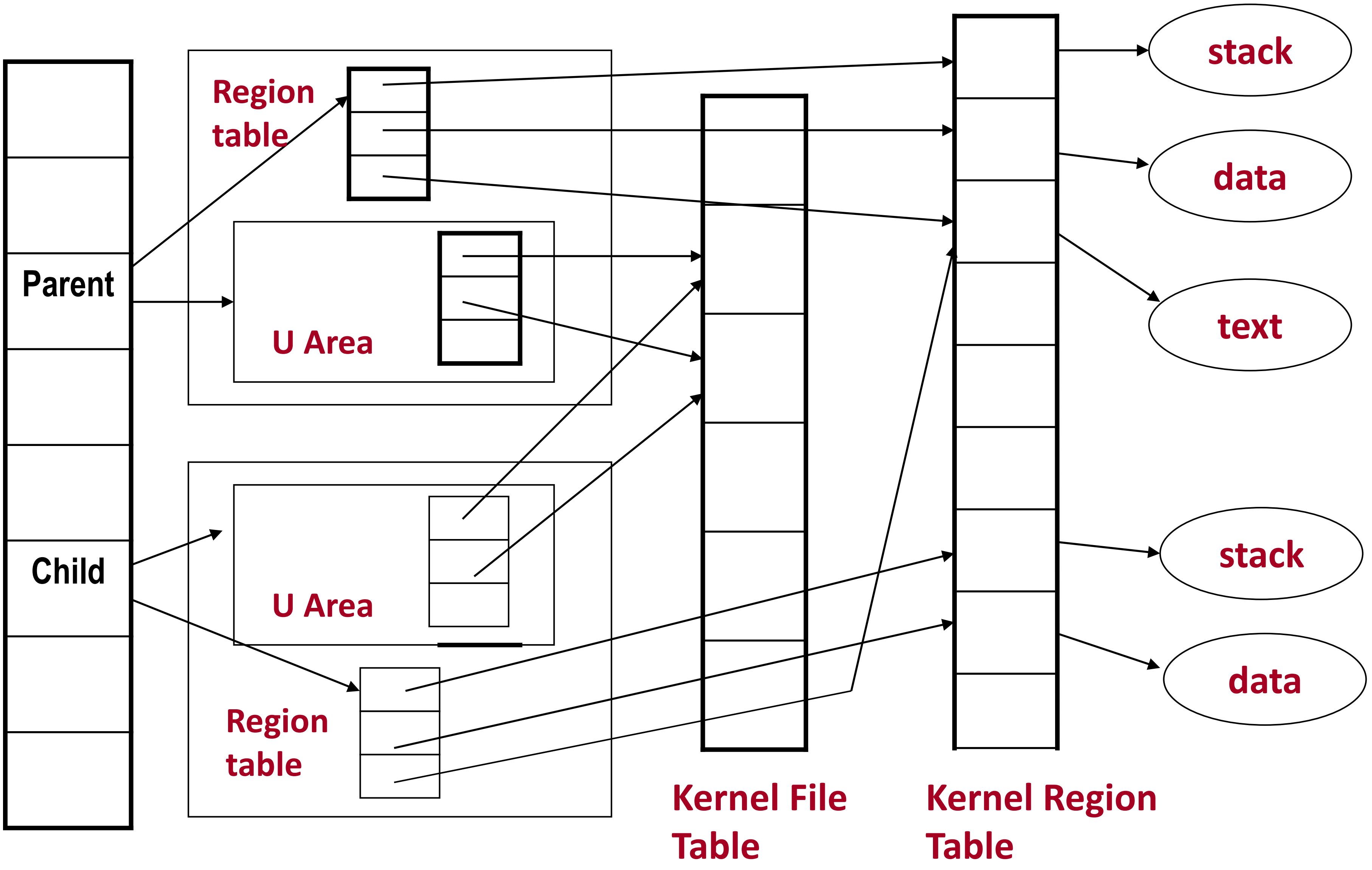
Reference count = 2

Per-process region table

Fork System Call(1)

- When a process is created by fork, it contains duplicate copies of the text, data and stack segments of its parent
- Also it has a File Descriptor Table (FDT) that contains references to the same opened files as its parent, such that they both share the same file pointer to each opened file.

Fork System Call(2)



Process Context

- The context of a process is its state:
  - Text, data(variable), register
  - Process region table, U Area,
  - User stack and kernel stack
- When executing a process, the system is said to be executing in the context of the process.

Context Switch

- When the kernel decides that it should execute another process, it does a context switch, so that the system executes in the context of the other process.
- When doing a context switch, the kernel saves enough information so that it can later switch back to the first process and resume its execution.

Mode of Process Execution(1)

- The UNIX process runs in two modes:
  - **User mode**
    - Can access its own instructions and data, but not kernel instruction and data
  - **Kernel mode**
    - Can access kernel and user instructions and data
- When a process executes a system call, the execution mode of the process changes from user mode to kernel mode

Mode of Process Execution(2)

- When moving from user to kernel mode, the kernel saves enough information so that it can later return to user mode and continue execution from where it left off.
- Mode change is not a context switch, just change in mode.

Process States

Process states are:

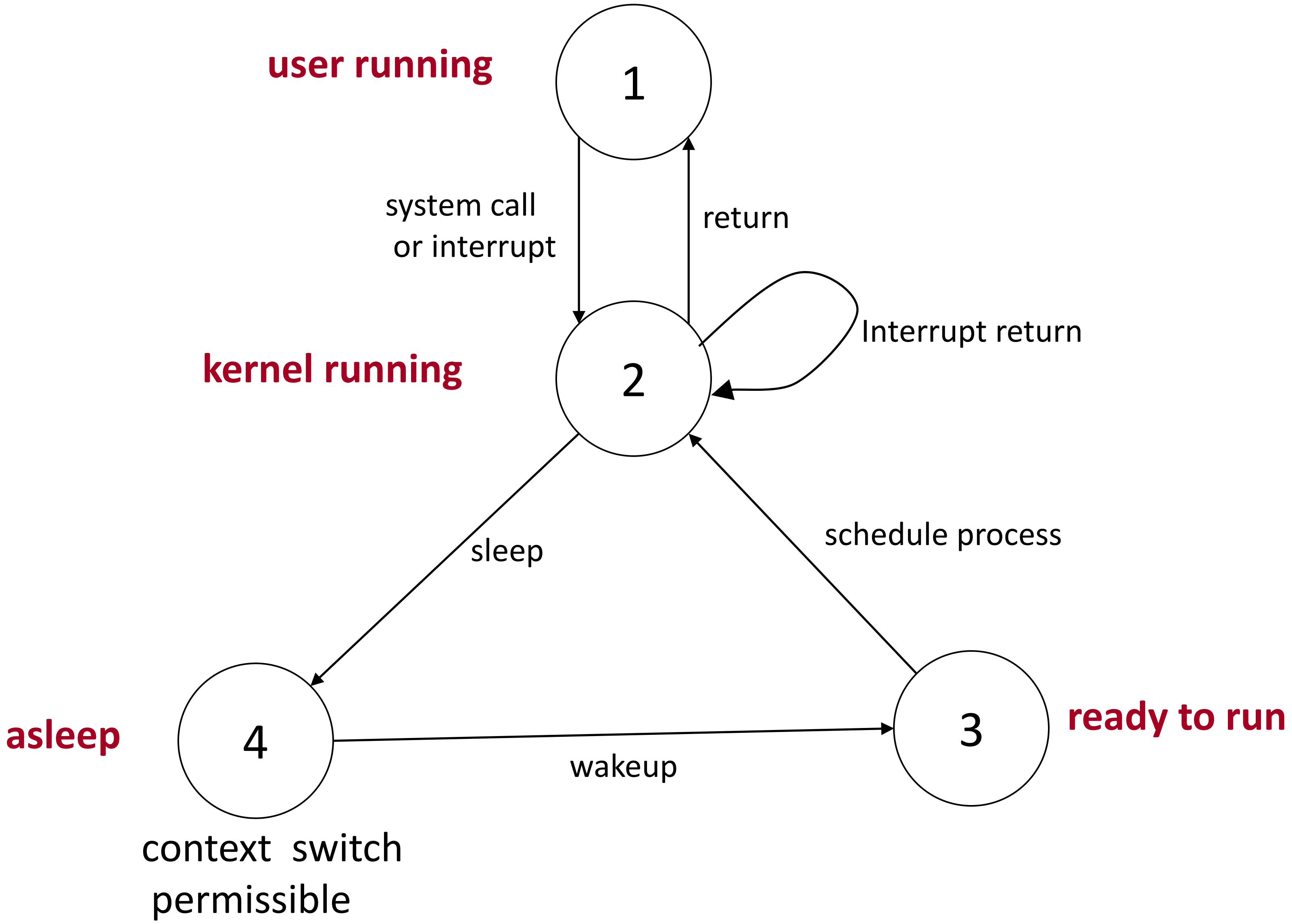
- The process is running in user mode
- The process is running in kernel mode
- The process is not executing, but it is ready to run as soon as the scheduler chooses it
- The process is sleeping
  - Such as waiting for I/O to complete

State transition

- A state transition diagram is a directed graph whose nodes represent the states a process can enter and whose edge represent the events that cause a process to move from one state to another.

- State transitions are legal b/w two states if there exits an edge from the first state to second.

Process State Transition(1)



Process State Transition(2)

- The kernel allows a context switch only when a process moves from the state kernel running to the state asleep.
- Process running in kernel mode cannot be preempted by other processes.

Sleep and wakeup

- When a process must temporarily suspend its execution (“go to sleep”), it does so of its own free will. Consequently, an interrupt handler cannot go to sleep, because if it could, the interrupted process would be put to sleep by default.

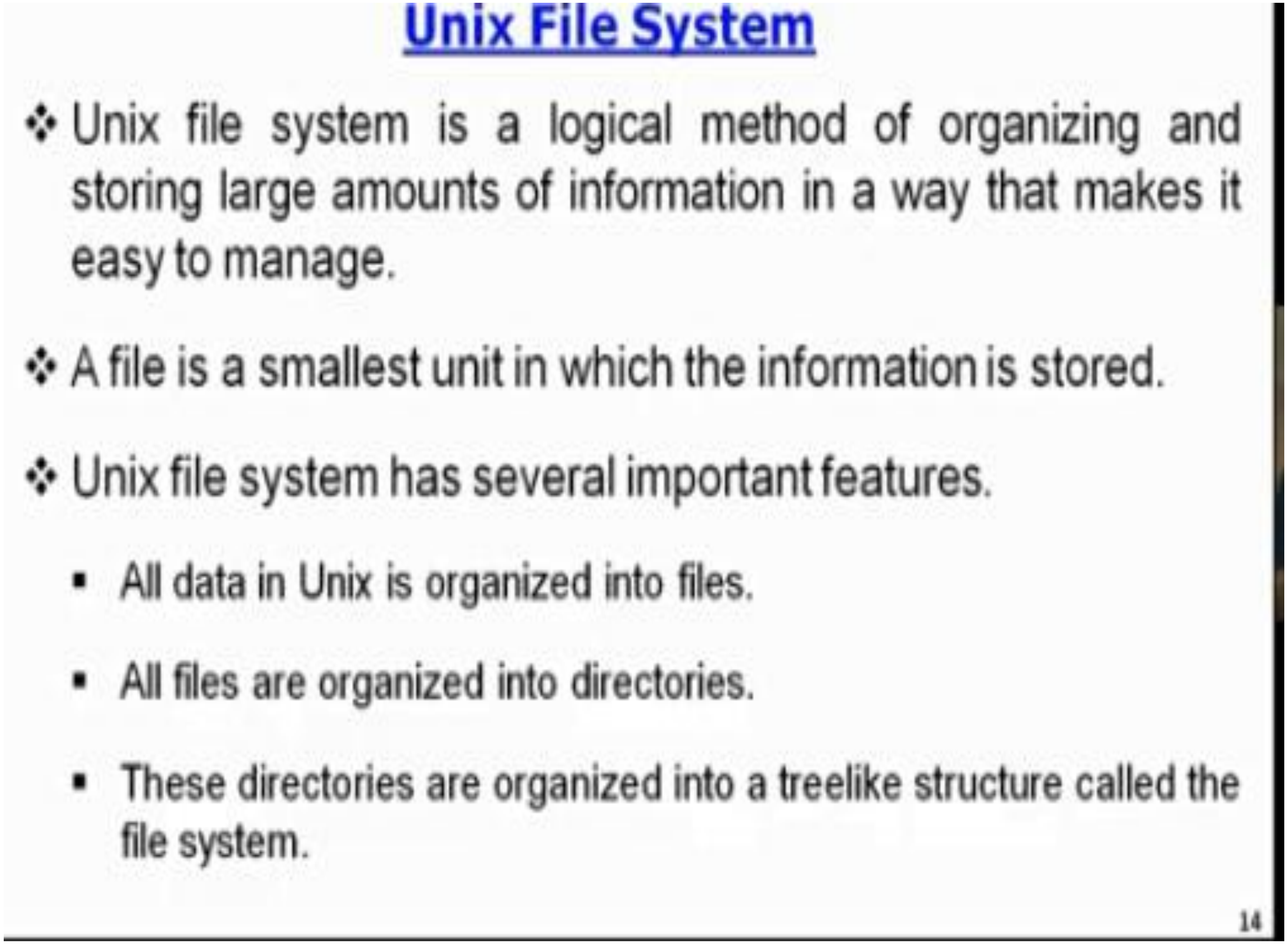
  Process go to sleep because they are awaiting the
  occurrence of some event, such waiting for i/o completion
  from a peripheral device, waiting for a process to exit, as
  waiting for system
  resources to become available, and so on.

Ready state

- Many process can simultaneously sleep on an event; when an event occurs, all process sleeping on the event wake up because the event condition is no longer true.

when a process wakes up, it follows the state transition from the "sleep" state to the "ready-to-run" state, where it is eligible for later scheduling; it does not execute immediately.

Sleeping process do not consume CPU resources.



Unix File System

- Unix file system is a logical method of organizing and storing large amounts of information in a way that makes it easy to manage.

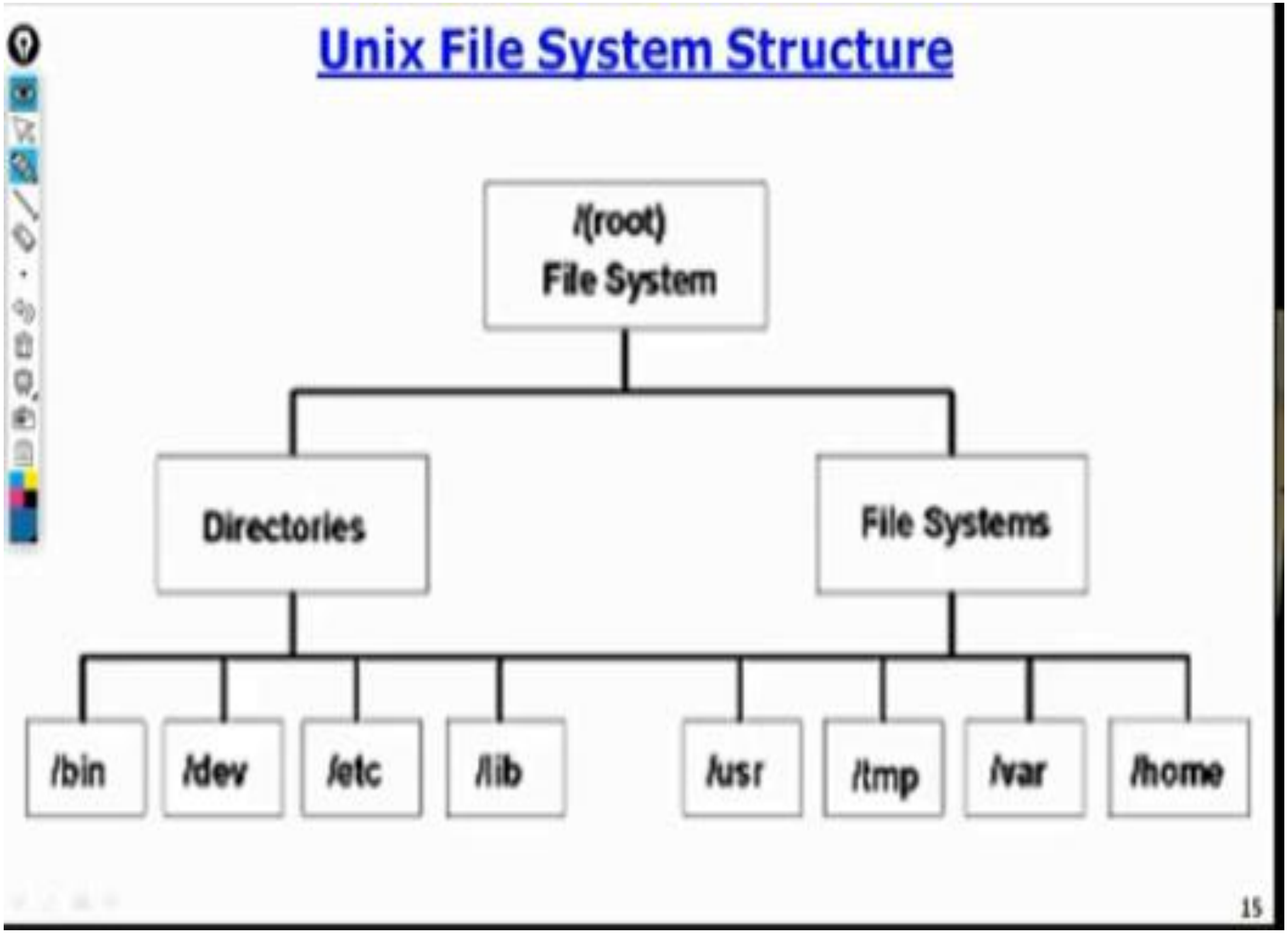
- A file is a smallest unit in which the information is stored.

- Unix file system has several important features.

  - All data in Unix is organized into files.

  - All files are organized into directories.

  - These directories are organized into a treelike structure called the file system.



Unix File System Structure

/(root)
File System

Directories

File Systems

/bin

/dev

/etc

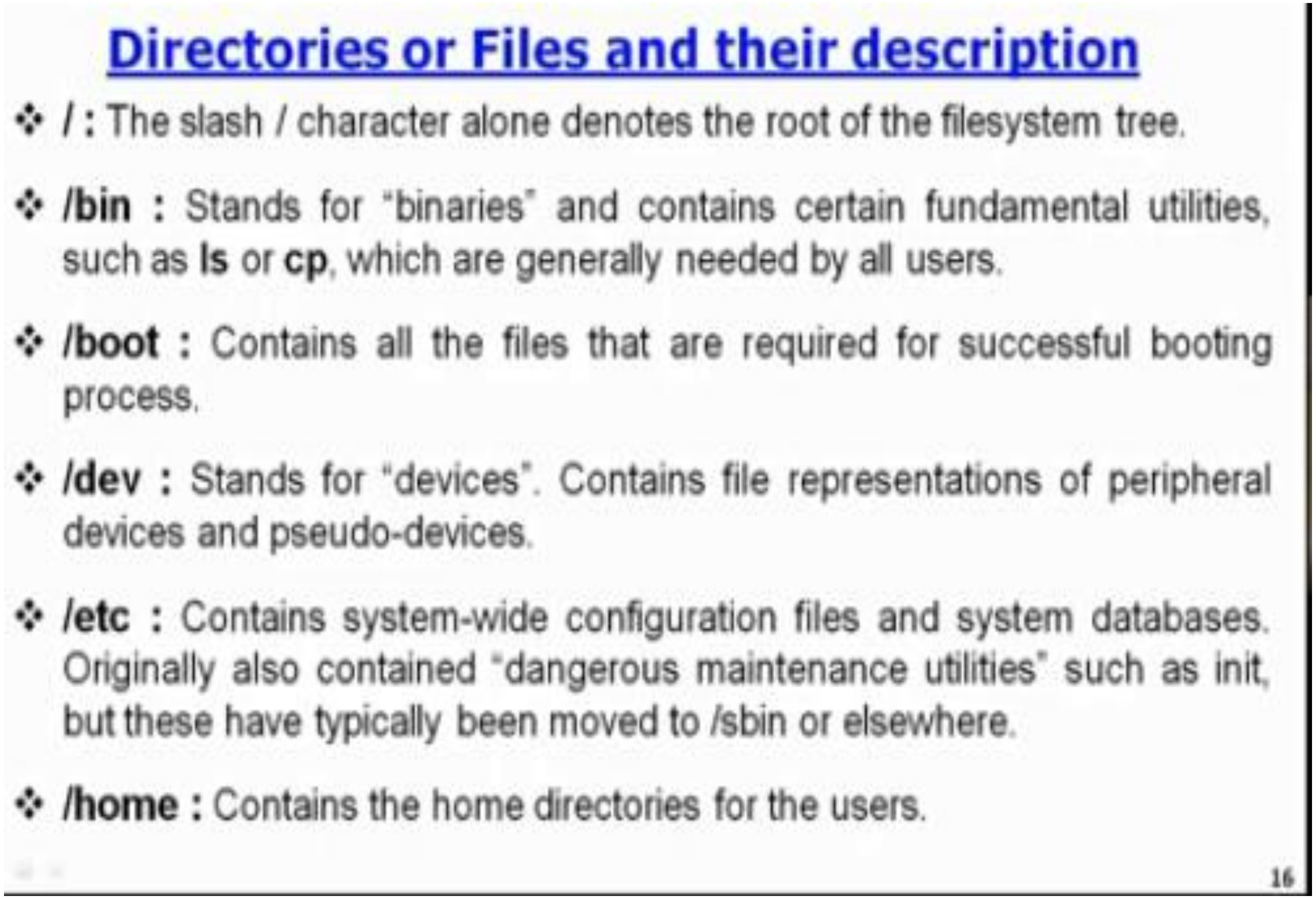
/lib

/usr

/tmp

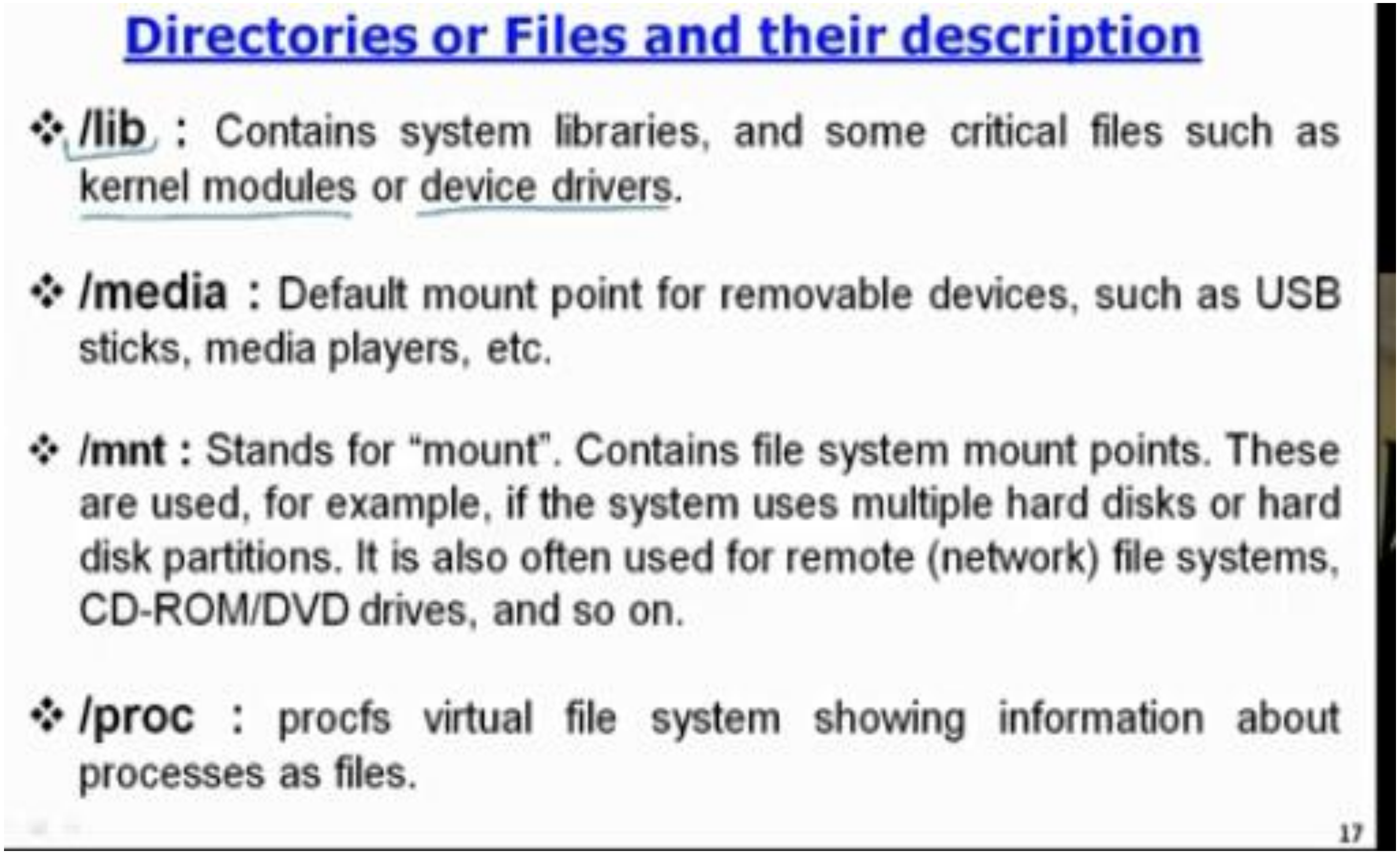
/var

/home



Directories or Files and their description

- **/ :** The slash / character alone denotes the root of the filesystem tree.
- **/bin :** Stands for "binaries" and contains certain fundamental utilities, such as **ls** or **cp**, which are generally needed by all users.
- **/boot :** Contains all the files that are required for successful booting process.
- **/dev :** Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.
- **/etc :** Contains system-wide configuration files and system databases. Originally also contained "dangerous maintenance utilities" such as init, but these have typically been moved to /sbin or elsewhere.
- **/home :** Contains the home directories for the users.



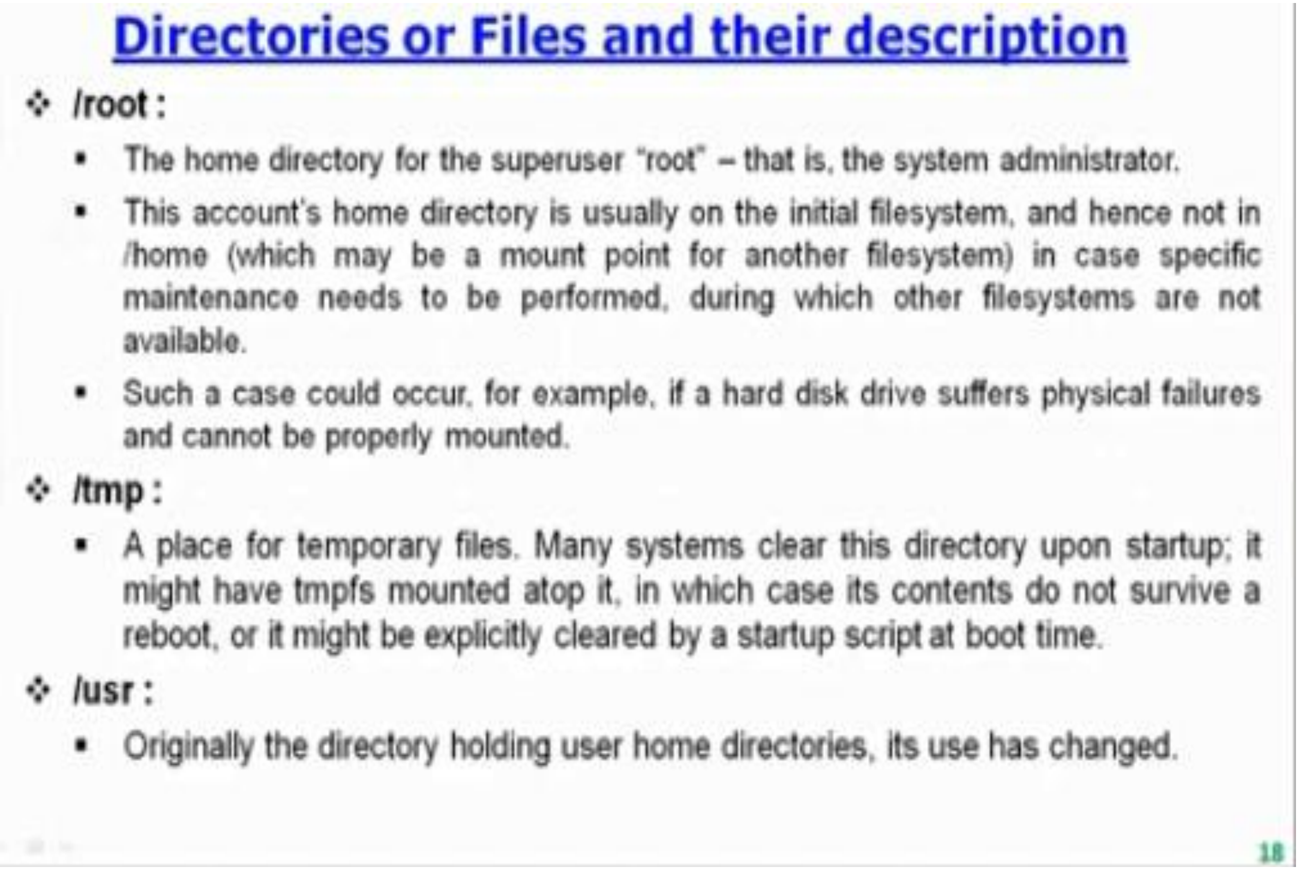
Directories or Files and their description

- **/lib** : Contains system libraries, and some critical files such as kernel modules or device drivers.

- **/media** : Default mount point for removable devices, such as USB sticks, media players, etc.

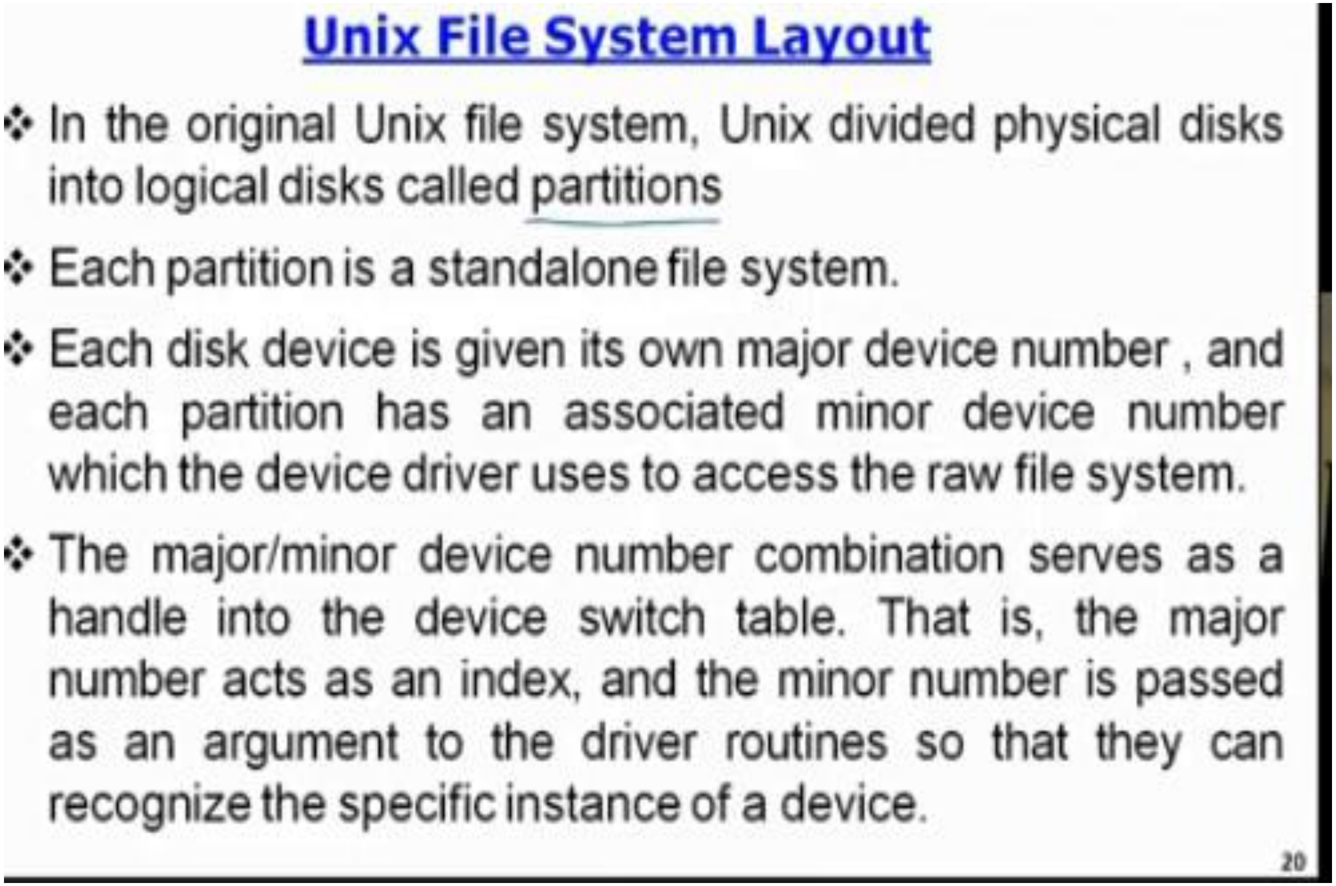
- **/mnt** : Stands for “mount”. Contains file system mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) file systems, CD-ROM/DVD drives, and so on.

- **/proc** : procfs virtual file system showing information about processes as files.



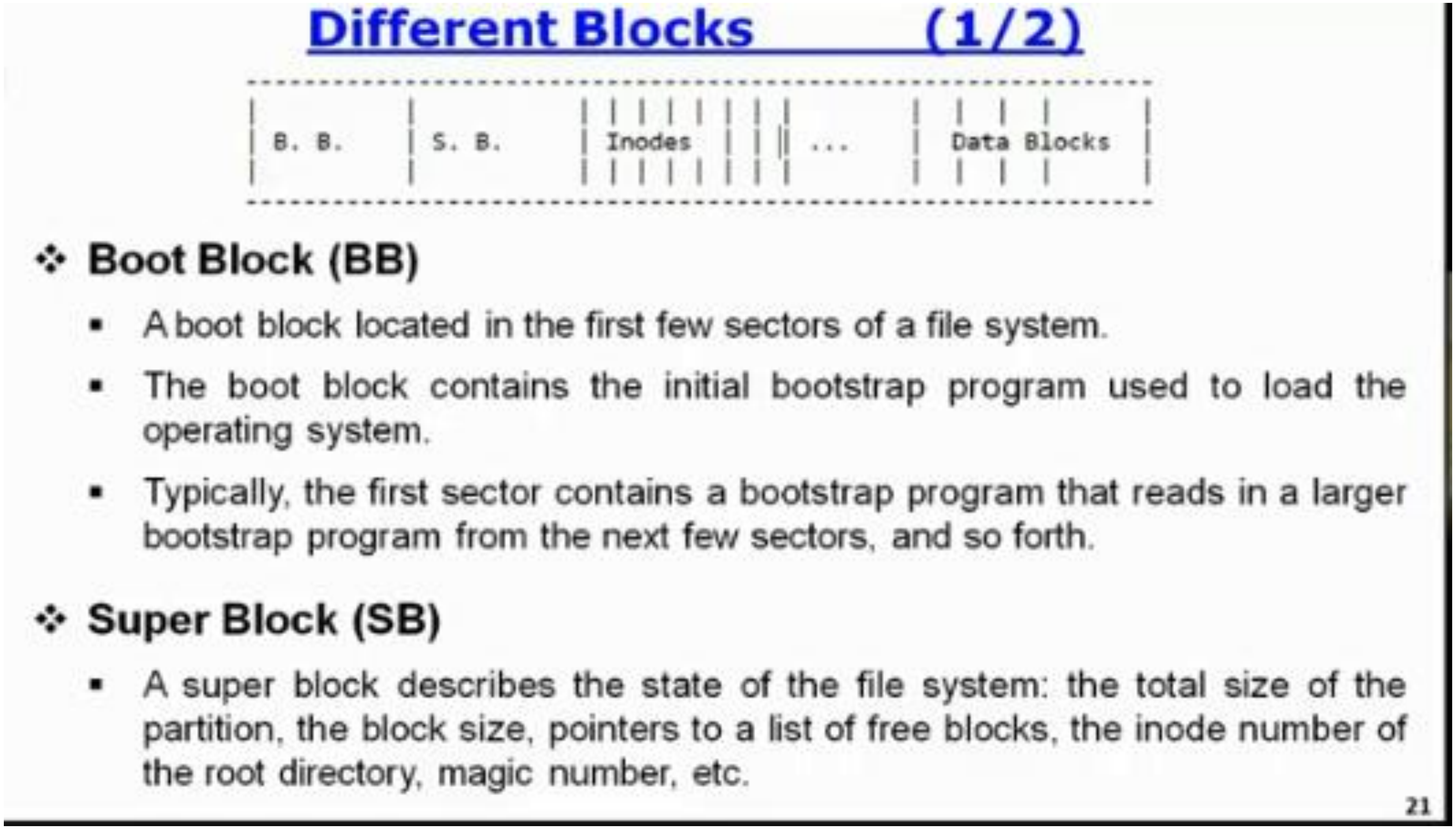
Directories or Files and their description

- **/root :**
  - The home directory for the superuser "root" – that is, the system administrator.
  - This account's home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available.
  - Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- **/tmp :**
  - A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- **/usr :**
  - Originally the directory holding user home directories, its use has changed.



Unix File System Layout

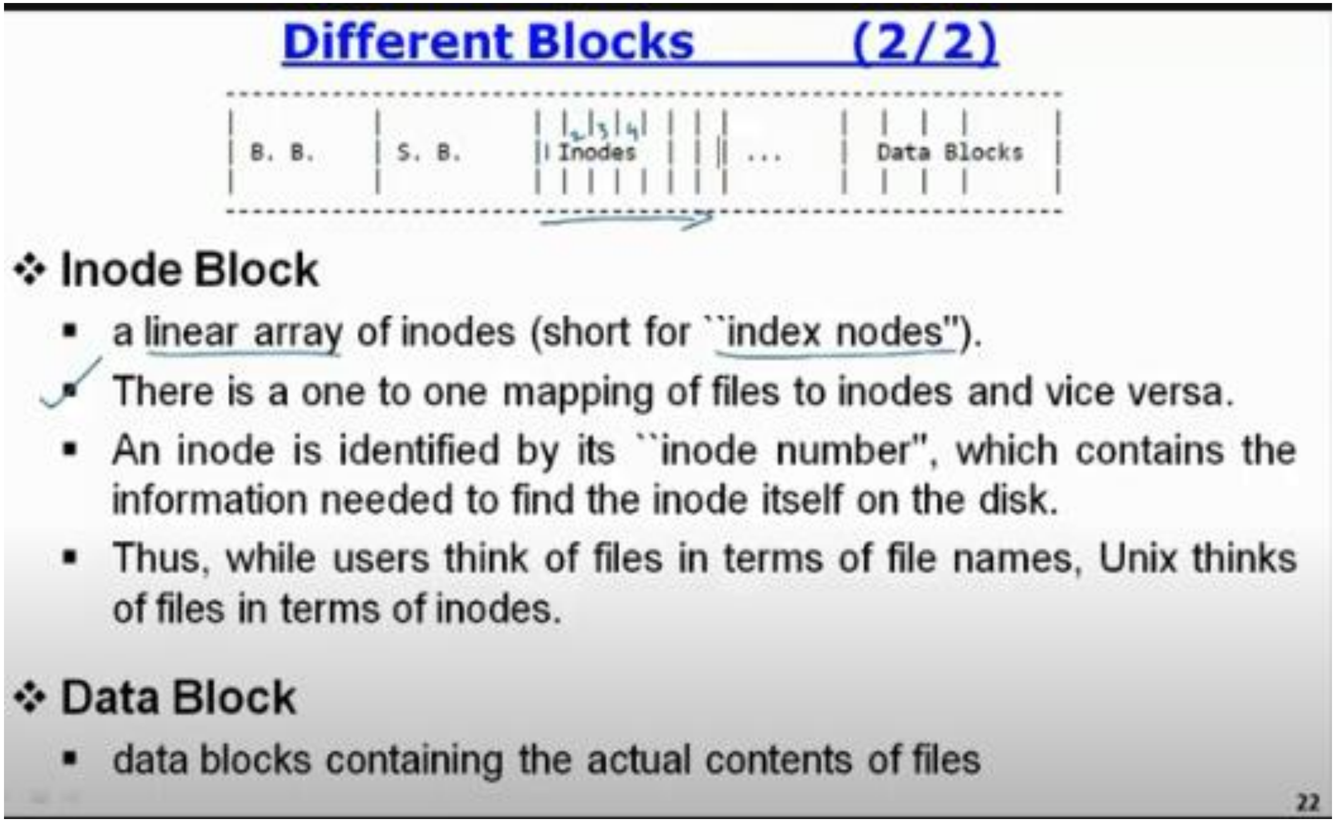
- In the original Unix file system, Unix divided physical disks into logical disks called partitions
- Each partition is a standalone file system.
- Each disk device is given its own major device number , and each partition has an associated minor device number which the device driver uses to access the raw file system.
- The major/minor device number combination serves as a handle into the device switch table. That is, the major number acts as an index, and the minor number is passed as an argument to the driver routines so that they can recognize the specific instance of a device.



Different Blocks (1/2)

```
---------------------------------------------------------------
|          |          | | | | | | | |       | | | |           |
| B. B.    | S. B.    | Inodes  | | || ...  | Data Blocks      |
|          |          | | | | | | | |       | | | |           |
---------------------------------------------------------------
```

- **Boot Block (BB)**
  - A boot block located in the first few sectors of a file system.
  - The boot block contains the initial bootstrap program used to load the operating system.
  - Typically, the first sector contains a bootstrap program that reads in a larger bootstrap program from the next few sectors, and so forth.
- **Super Block (SB)**
  - A super block describes the state of the file system: the total size of the partition, the block size, pointers to a list of free blocks, the inode number of the root directory, magic number, etc.



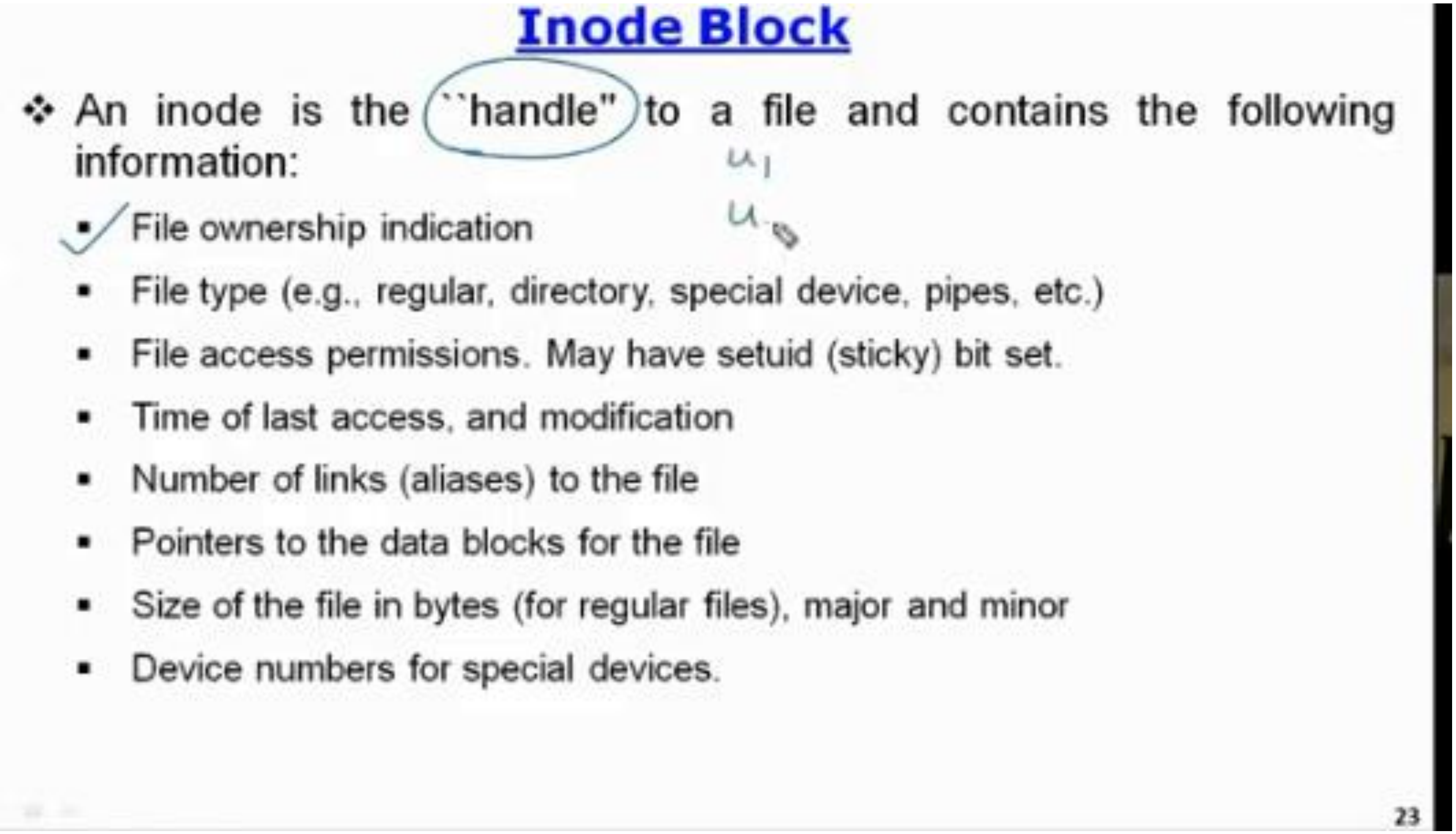
Different Blocks (2/2)

❖ Inode Block

- a linear array of inodes (short for ``index nodes'').
- There is a one to one mapping of files to inodes and vice versa.
- An inode is identified by its ``inode number'', which contains the information needed to find the inode itself on the disk.
- Thus, while users think of files in terms of file names, Unix thinks of files in terms of inodes.

❖ Data Block

- data blocks containing the actual contents of files



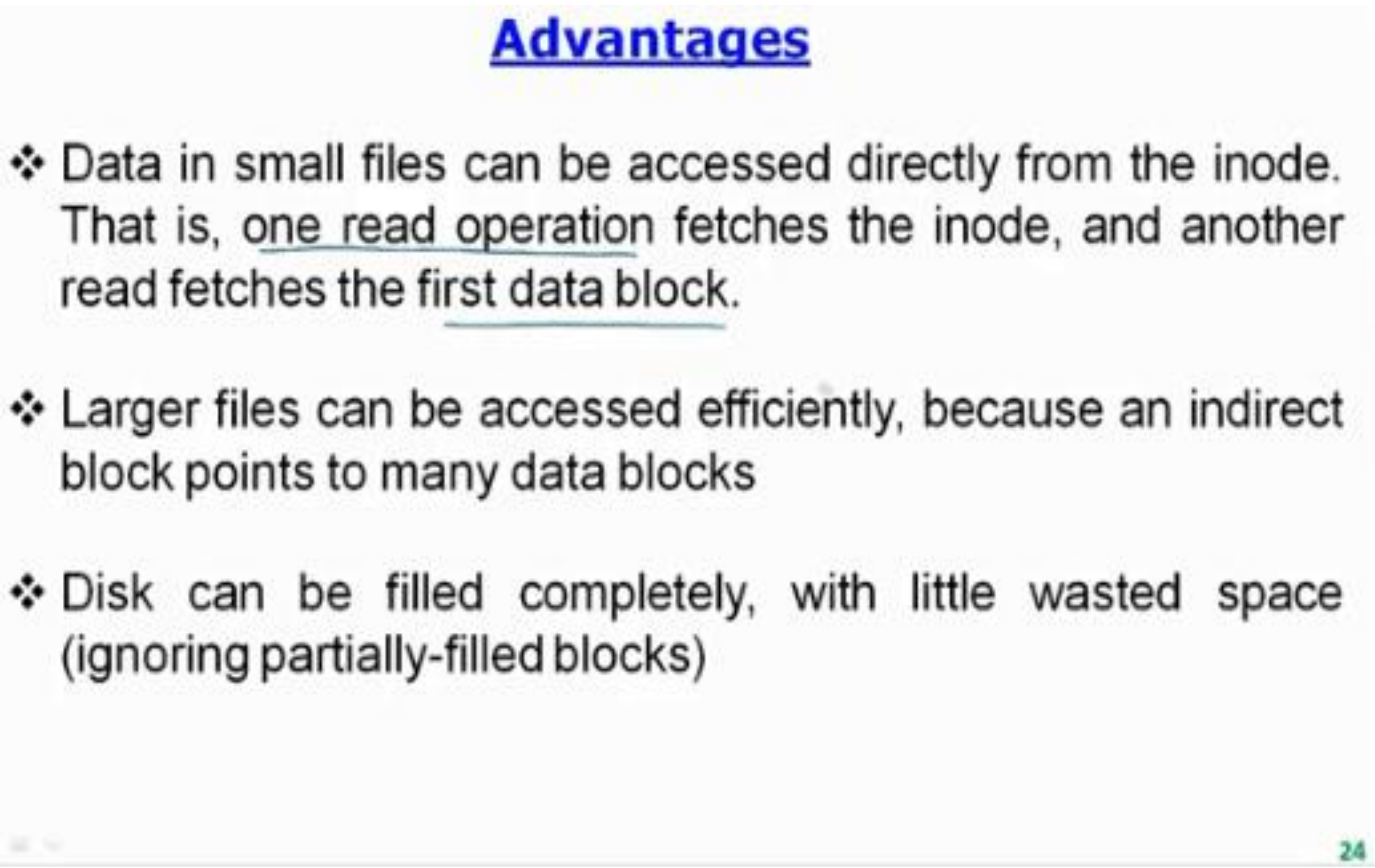
Inode Block

- An inode is the ``handle" to a file and contains the following information:

u_1

u

  - File ownership indication
  - File type (e.g., regular, directory, special device, pipes, etc.)
  - File access permissions. May have setuid (sticky) bit set.
  - Time of last access, and modification
  - Number of links (aliases) to the file
  - Pointers to the data blocks for the file
  - Size of the file in bytes (for regular files), major and minor
  - Device numbers for special devices.



Advantages

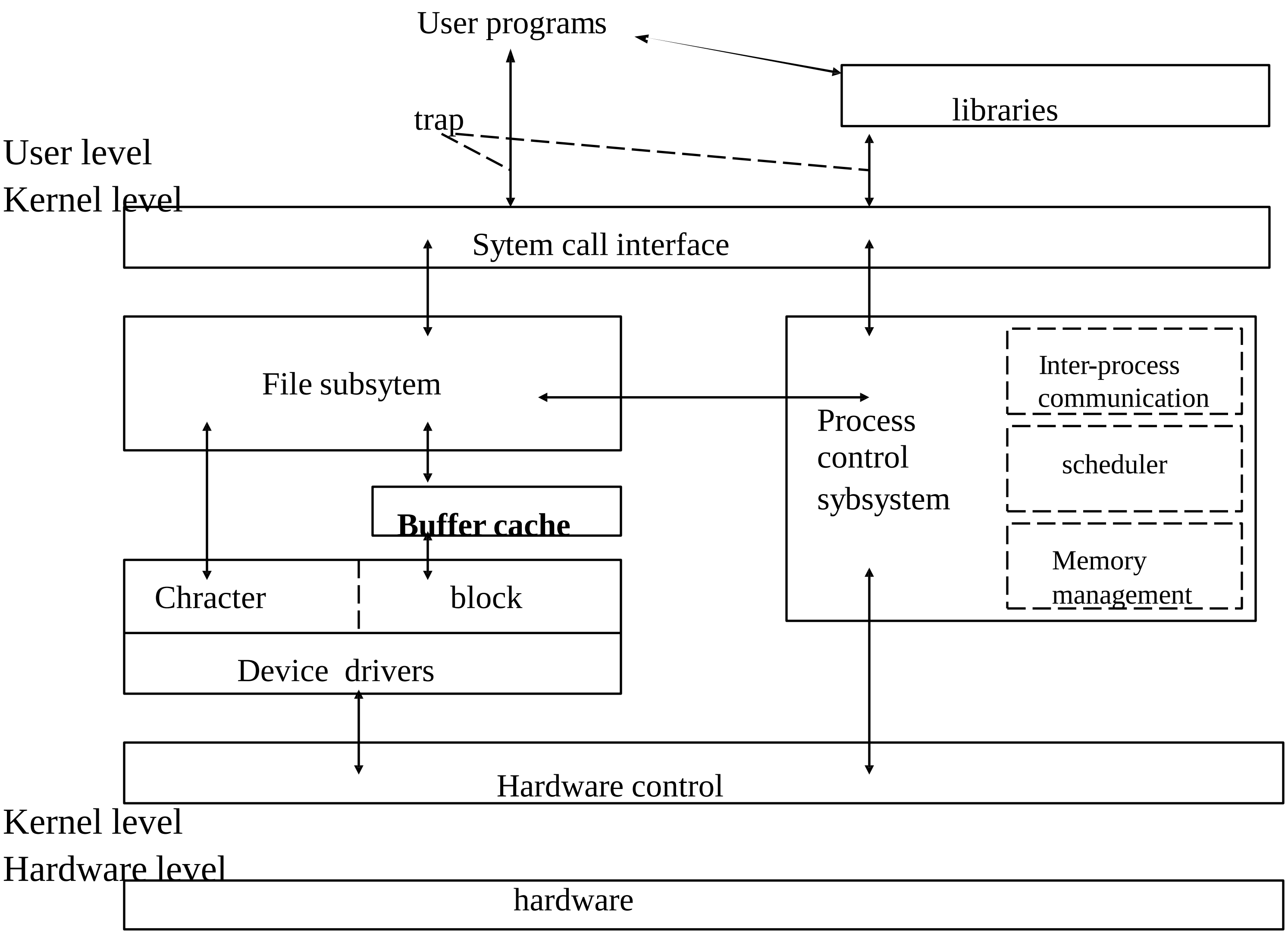
- Data in small files can be accessed directly from the inode. That is, one read operation fetches the inode, and another read fetches the first data block.
- Larger files can be accessed efficiently, because an indirect block points to many data blocks
- Disk can be filled completely, with little wasted space (ignoring partially-filled blocks)

Buffer Cache

TOPICS

UNIX system Architecture
Buffer Cache
Buffer Pool Structure
Retrieval of Buffer
Release Buffer
Reading and Writing Disk Blocks

UNIX Kernel Architecture



Buffer Cache

- Kernel maintains files on mass storage devices such as disks and allow processes to store new information or recall previously stored information.

- When a process wants to access data from a file, the kernel brings the data into main memory, alters it and then request to save in the file system.

- **Example: copy one.c two.c**

Buffer Cache

- The superblock of a file system describes the free space available on the file system.
- The kernel read the super block into memory to access its data and writes it back to the file system when it wishes to save its data.
- The inode describes the layout of a file.
- So the kernel reads the inode into the memory when any process wants to access the data in a file and writes the inode back to the file system when it wants to update the file layout.

Buffer Cache

- The kernel could read and write the file directly from the hard disk and put it in memory and vice versa but the response time and throughput will be very low in this case because of disks slow data transfer speed.
- To increase the response time and throughput, the kernel minimizes the frequency of disk access by keeping a pool of internal data buffer called buffer cache.

Buffer Cache

- Buffer cache contains the data in recently used disk blocks
- When reading data from disk, the kernel attempts to read from buffer cache.
  - If data is already in the buffer cache, the kernel does not need to read from disk.
  - If data is not in the buffer cache, the kernel reads the data from disk and cache it.

Buffer Cache

- Similarly, data being written to disk is cached so that it will be there if the kernel later tries to read it.

- Higher-level kernel algorithms instruct the buffer cache modules to pre-cache data or to delay-write data to maximize caching effect

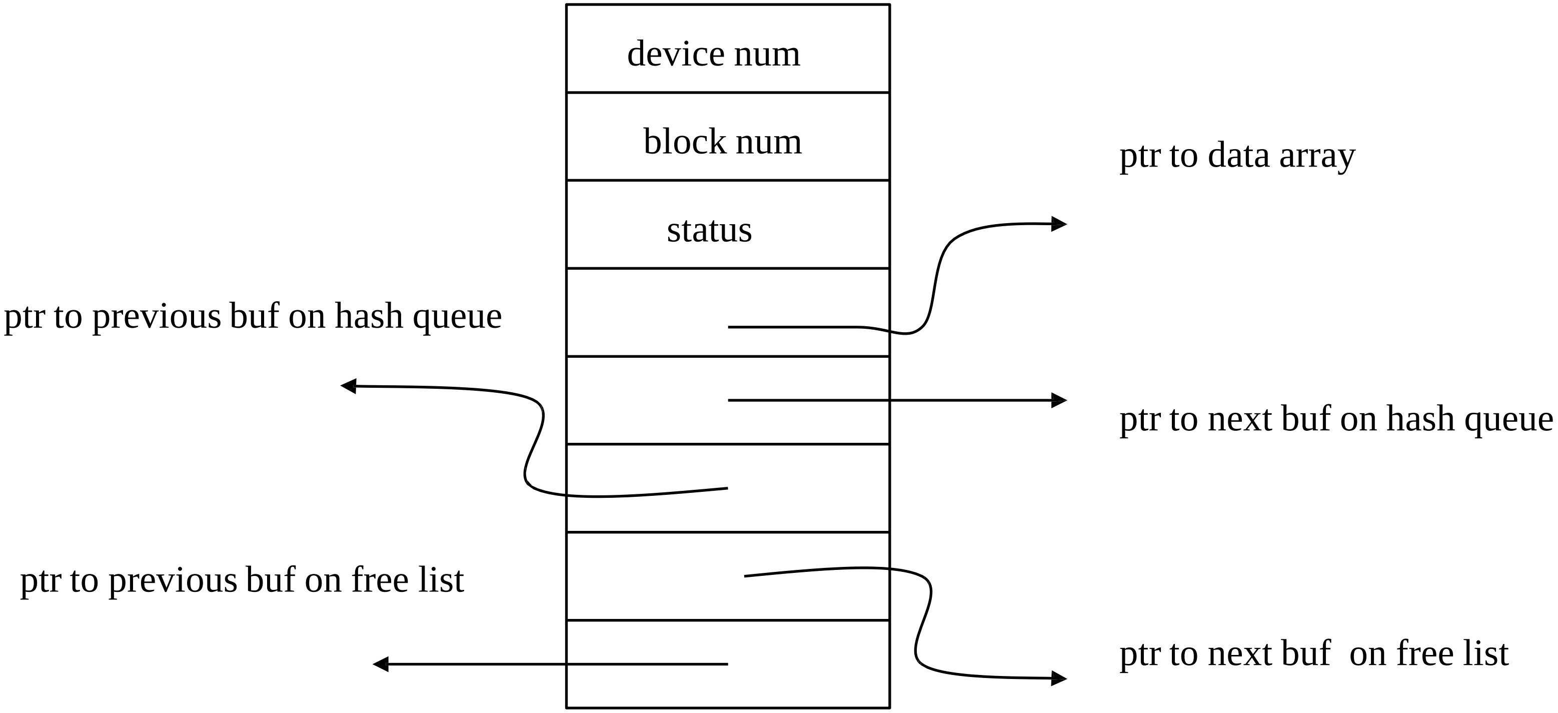
buffer

- The data in a buffer corresponds to the data in a logical disk block on a file system, and the kernel identifies the buffer contents by examining identifier fields in the buffer header
  The contents of the disk block map into the buffer. A disk block can never map into more than one buffer at a time.
  If two buffers were to contain data for one disk block, the kernel would not know which buffer containes the current data and could write incorrect data back to disk.

Buffer Headers

- A buffer consists of two parts
  - a memory array : that contains data from the disk
  - buffer header : that identifies the buffer
- disk block : buffer = 1 : 1

Buffer Headers



device num

block num

status

ptr to data array

ptr to previous buf on hash queue

ptr to next buf on hash queue

ptr to previous buf on free list

ptr to next buf on free list

Figure 3.1 Buffer Header

Buffer Headers

- device num
  - logical file system number
- block num
  - block number of the data on disk
- status
  - The buffer is currently locked.
  - The buffer contains valid data.
  - delayed-write
  - The kernel is currently reading or writing the contents of the disk.
  - A process is currently waiting for the buffer to become free.
- kernel identifies the buffer content by examing device num and block num.

Buffer header

- The buffer header contains a device number field and a block number field that specify the file system and block number of the data on disk and uniquely identify the buffer.
- The buffer header also contain a pointer to a data array for the buffer, and a status field that summarizes the current status of the buffer.
  - The buffer header also contain two sets of pointer, used by the buffer allocation algo. to maintain the overall structure of the buffer pool.

Status of a buffer

- The status of a buffer is a combination of the following condition:

1. The buffer is currently locked (busy)
2. The buffer contains valid data
3. The kernel must write the buffer contents to disk before reassigning the buffer, the condition is known as “delayed-write”.
4. The kernel is currently reading or writing the contents of the buffer to disk
5. A process is currently waiting for the buffer to become free

Structures of the buffer pool

- Buffer pool according to LRU
- The kernel maintains a free list of buffer
  - doubly linked circular list
  - take a buffer from the head of the free list.
  - When returning a buffer, attaches the buffer to the tail.

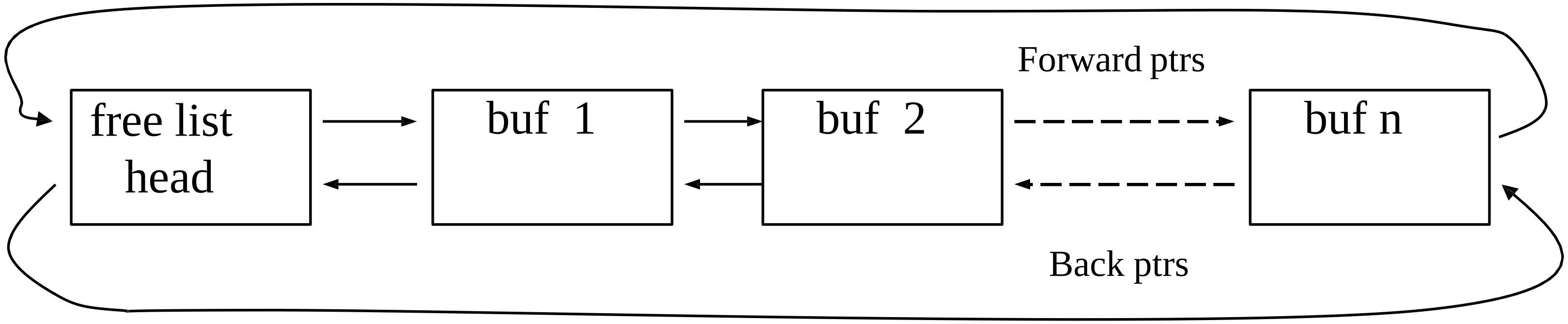


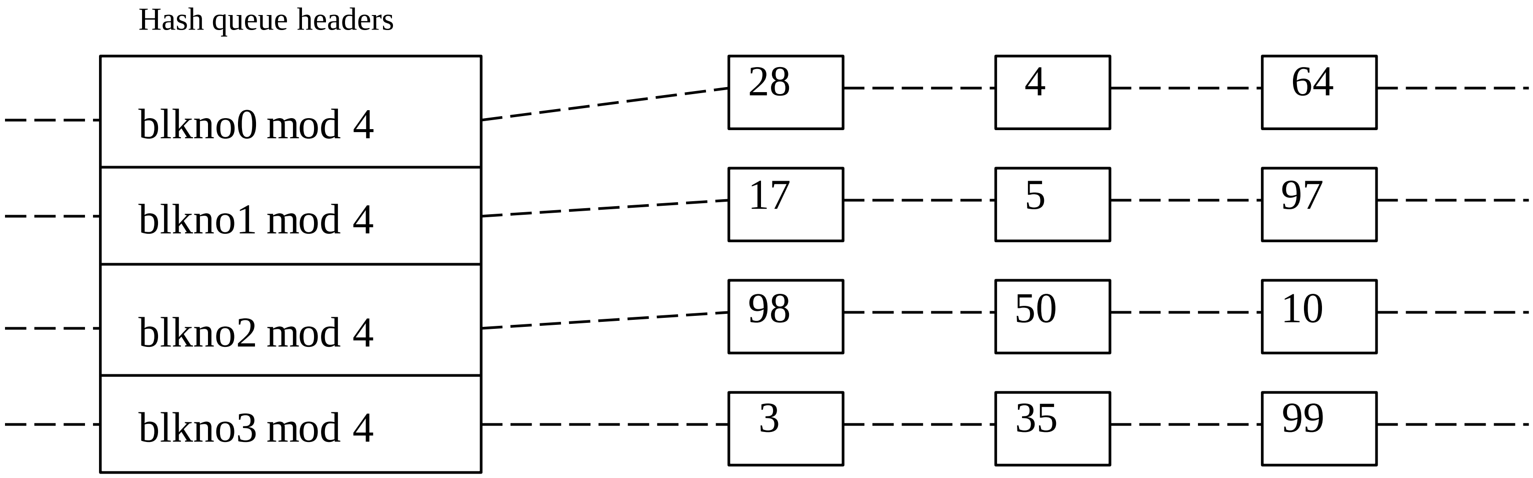
Figure 3.2 Free list of Buffers

Structures of the buffer pool

- When kernel want to access the disk it searches the buffer pool for a particular device number-block number combination (which is maintained in the buffer header).
- The entire buffer pool is organized as queues, hashed as a function of device number-block number combination.

Structures of the buffer pool

- When the kernel accesses a disk block
  - separate queue (doubly linked circular list)
  - hashed as a function of the device and block num
  - Every disk block exists on one and only one hash queue and only once on the queue



Hash queue headers

blkno0 mod 4 — 28 — 4 — 64

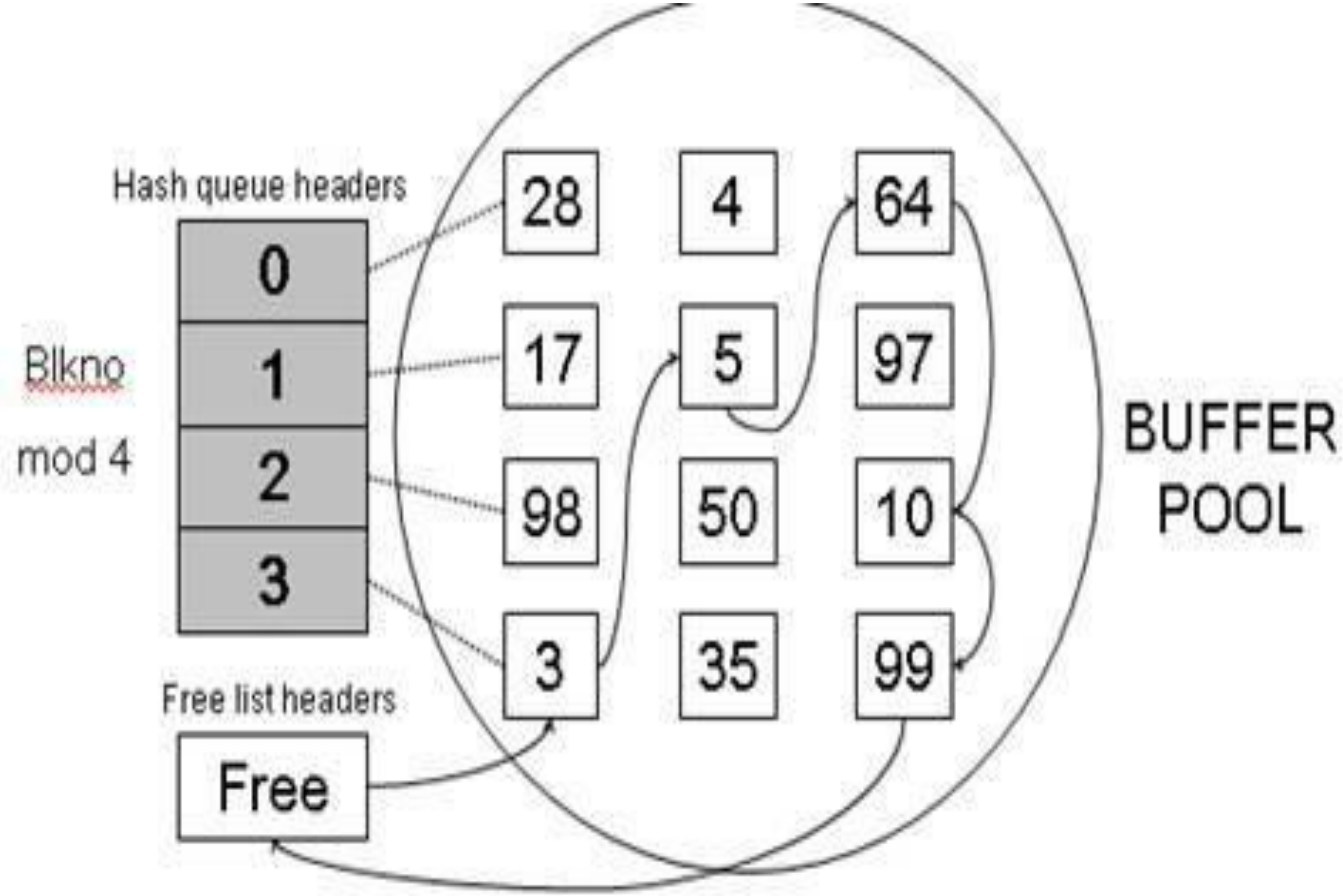
blkno1 mod 4 — 17 — 5 — 97

blkno2 mod 4 — 98 — 50 — 10

blkno3 mod 4 — 3 — 35 — 99

Figure 3.3 Buffers on the Hash Queues

Scenarios for retrieval of a buffer



Scenarios for Retrieval of a Buffer

□ The kernel finds the block on its hash queue, and its buffer is free.

□ The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list.

□ The kernel cannot find the block on the hash queue and, in attempting to allocate a buffer from the free list (as in scenario 2) , finds a buffer on the free list that has been marked "delayed write." The kernel must write the "delayed write" buffer to disk and allocate another buffer.

□ The kernel cannot find the block on the hash queue, and the free list of buffers is empty.

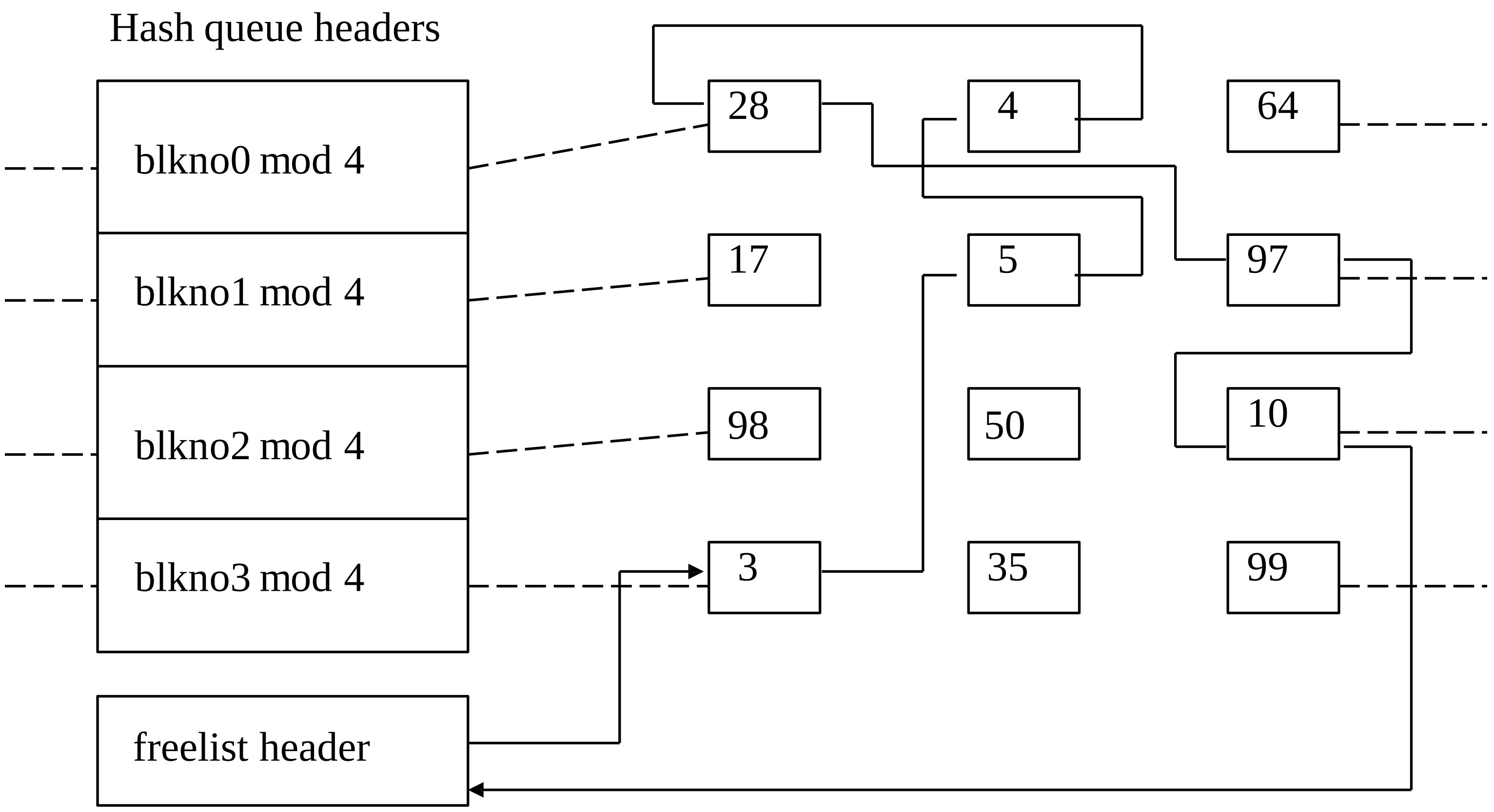
□ The kernel finds the block on the hash queue, but its buffer is currently busy.

Scenarios for retrieval of a buffer

- Determine the logical device num and block num
- The algorithms for reading and writing disk blocks use the algorithm *getblk*
  - The kernel finds the block on its hash queue
    - The buffer is free.
    - The buffer is currently busy.
  - The kernel cannot find the block on the hash queue
    - The kernel allocates a buffer from the free list.
    - In attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked "delayed write".
    - The free list of buffers is empty.

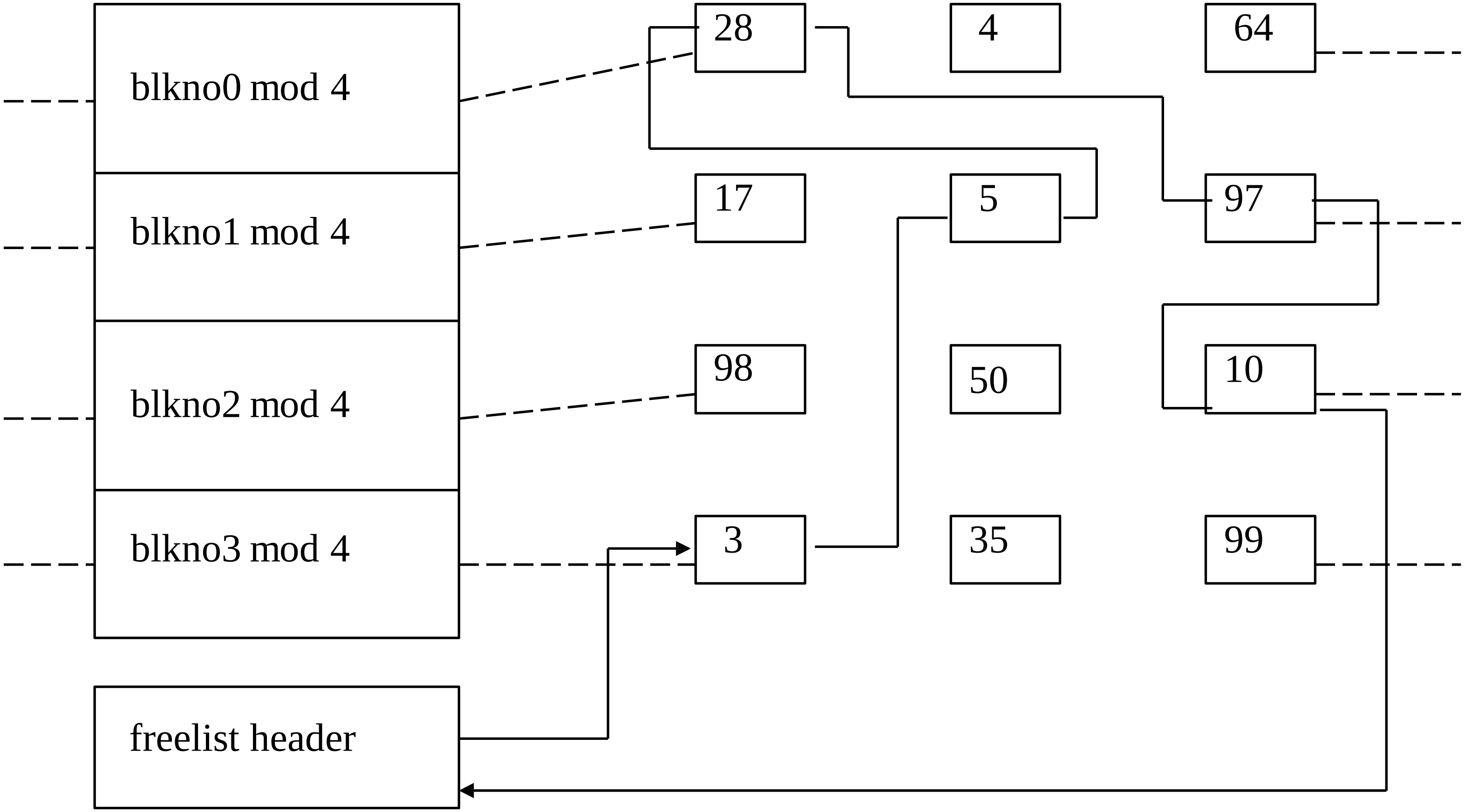
Retrieval of a Buffer:1st Scenario (a)

- The kernel finds the block on the hash queue and its buffer is free



Search for block 4

Retrieval of a Buffer:1st Scenario (b)



Remove block 4 from free list

Algorithm: ReleaseBlock

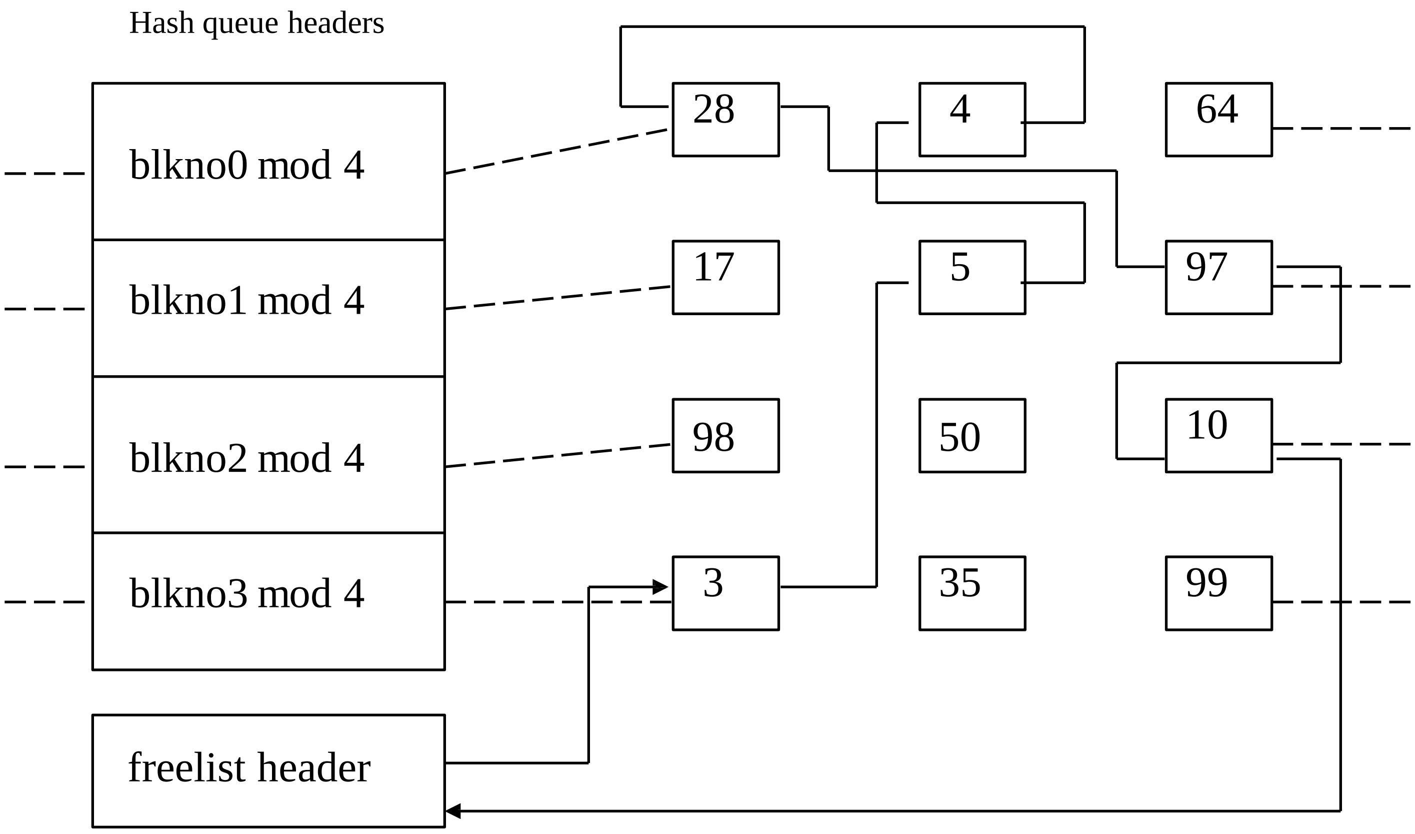
- ReleaseBlock (locked buffer)
  - wakeup all process event, waiting for any buffer to become free
  - wakeup all process event, waiting for this buffer to become free
  - raise processor execution level to block interrupt
  - if (buffer content valid and buffer not old)
    - enqueue buffer at the end of free list
  - else
    - enqueue buffer at the beginning of free list
  - lower processor execution level to allow interrupt
  - unlock (buffer)

Algorithm for Releasing a Buffer

```
algorithm brelse
input:    locked buffer
output:   none
{
    wakeup all procs: event, waiting for any buffer to become free;
    wakeup all procs: event, waiting for this buffer to become free;
    raise processor execution level to block interrupts;
    if(buffer contents valid and buffer not old)
            enqueue buffer at end of free list
    else
            enqueue buffer at beginning of free list
    lower processor execution level to allow interrupts;
    unlock(buffer);
}
```

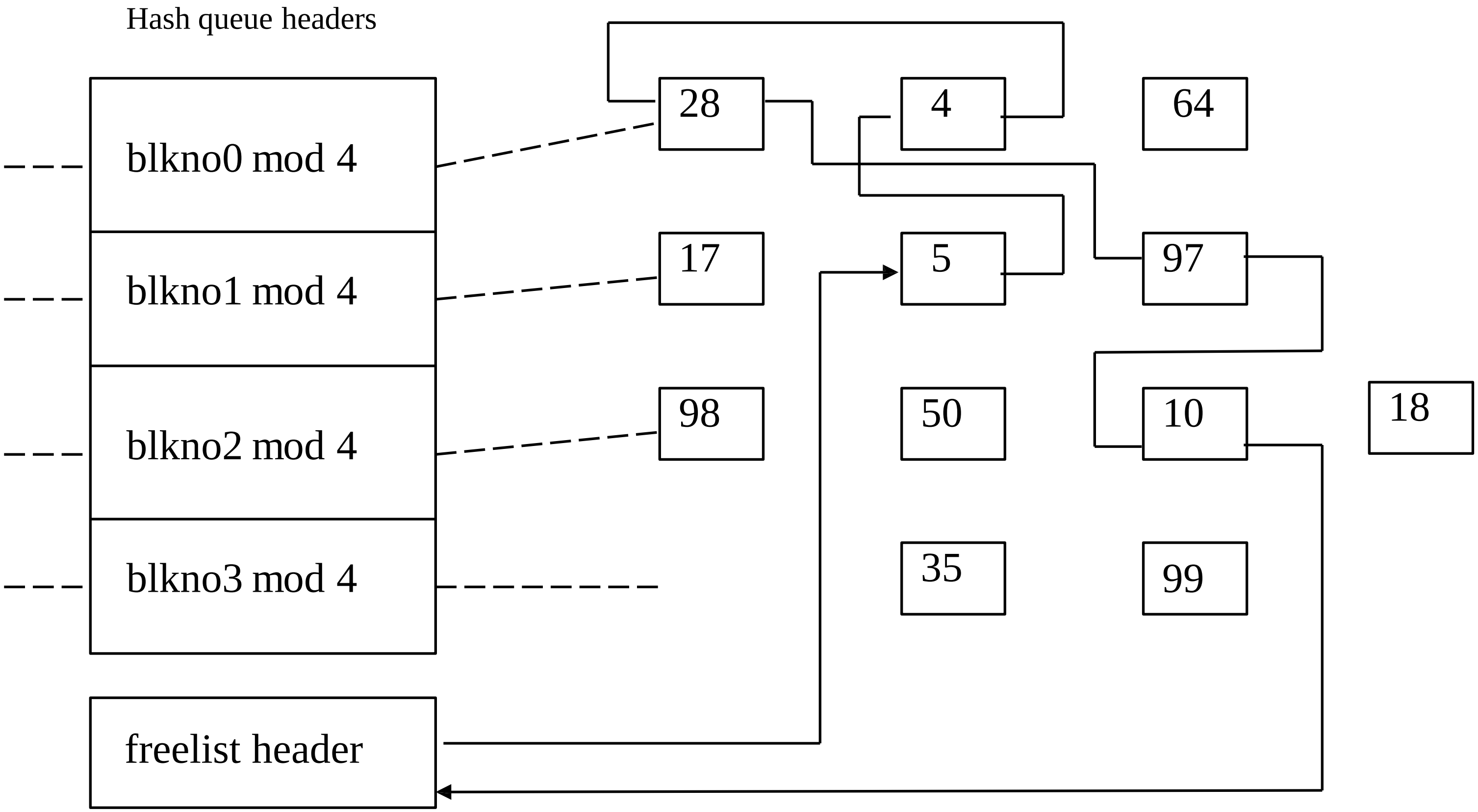
Retrieval of a Buffer: 2nd Scenario (a)

- The kernel cannot find the block on the hash queue, so it allocates a buffer from free list



Search for block 18: Not in cache

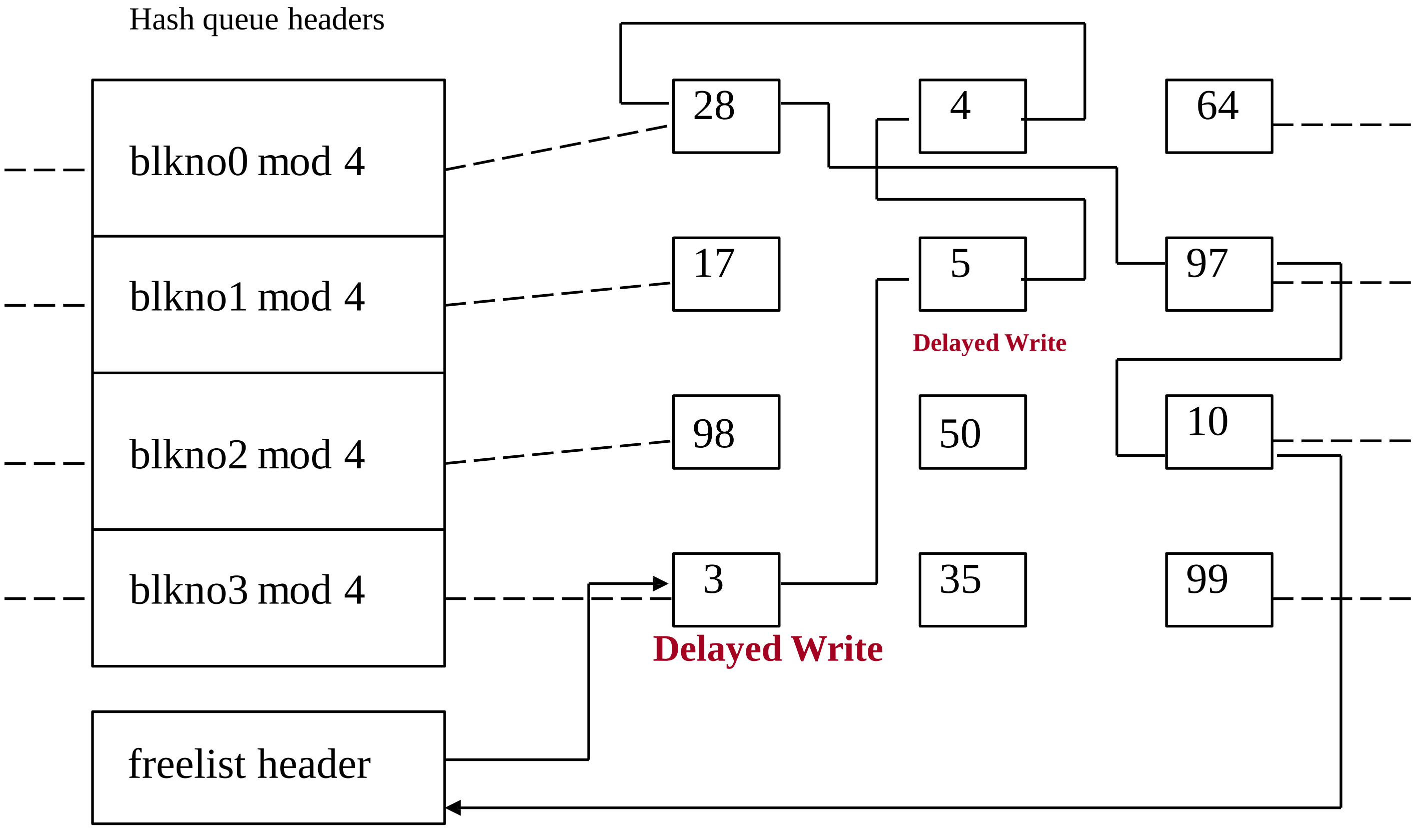
Retrieval of a Buffer: 2nd Scenario (b)



Remove 1st block from free list: Assign to 18

Retrieval of a Buffer: 3rd Scenario (a)

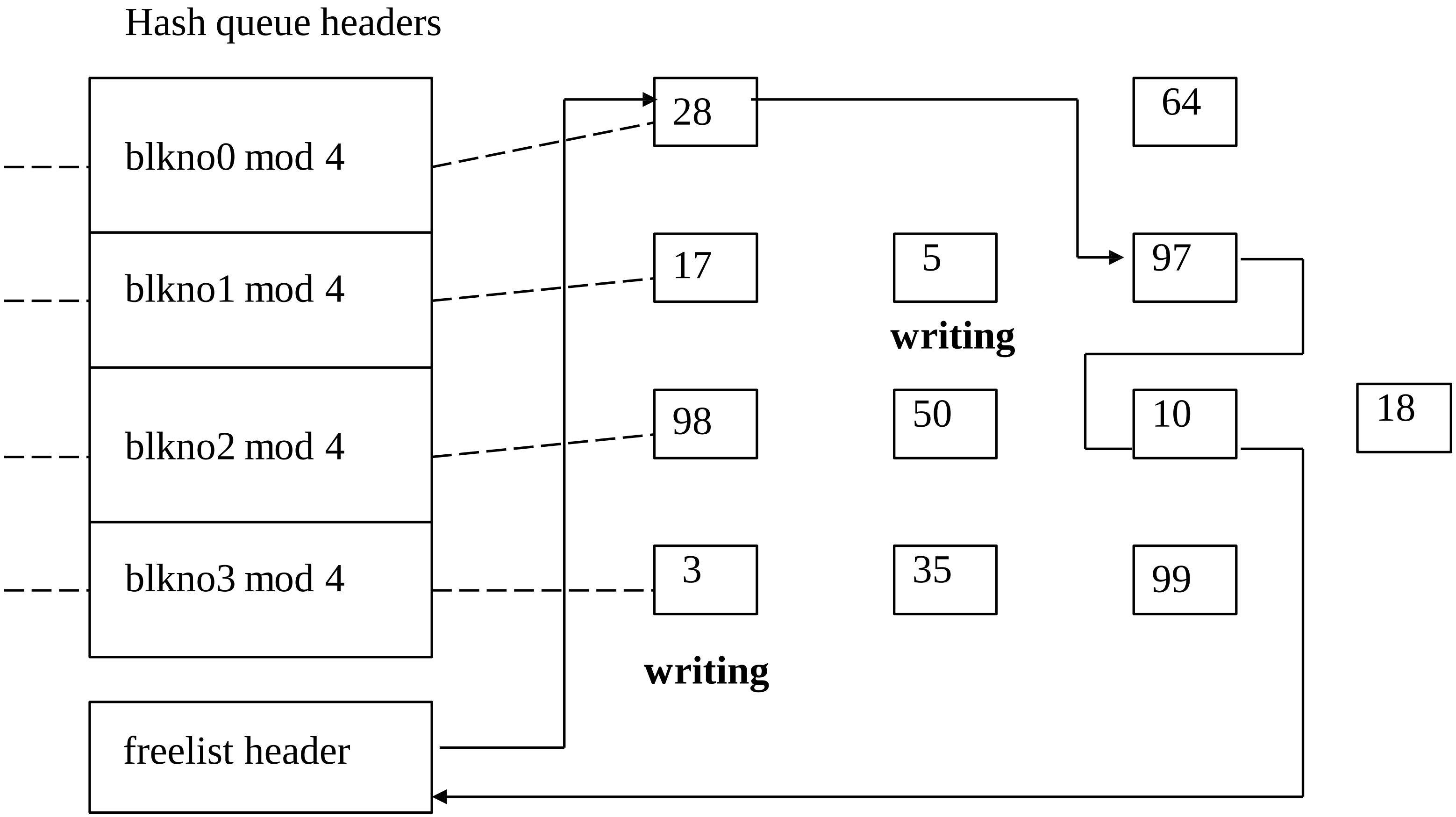
- The kernel cannot find the block on the hash queue, and finds delayed write buffers on hash queue



Search for block 18, Delayed write blocks on free list

Retrieval of a Buffer: 3rd Scenario (b)

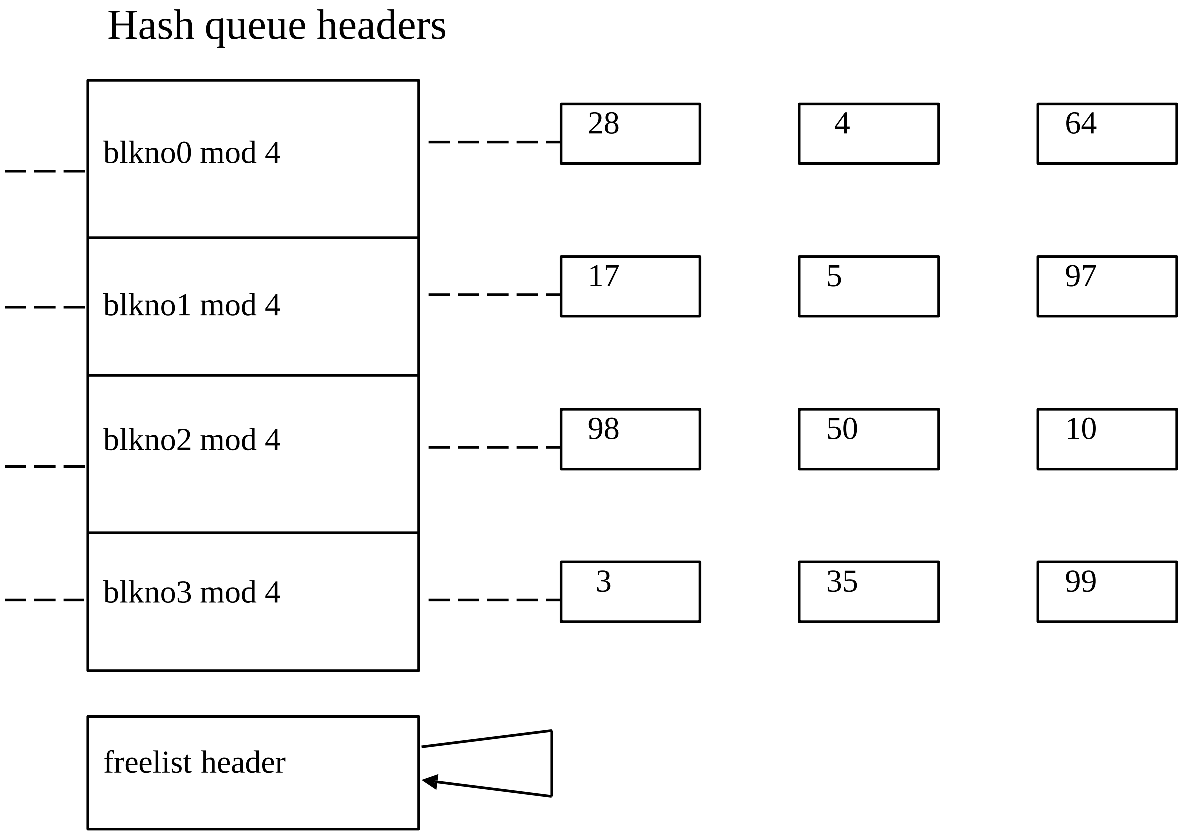
The kernel starts an asynchronous write and places it at the head of the free list



(b) Writing Blocks 3, 5, Reassign 4 to 18
Figure 3.8

Retrieval of a Buffer: 4th Scenario

- The kernel cannot find the buffer on the hash queue, and the free list is empty

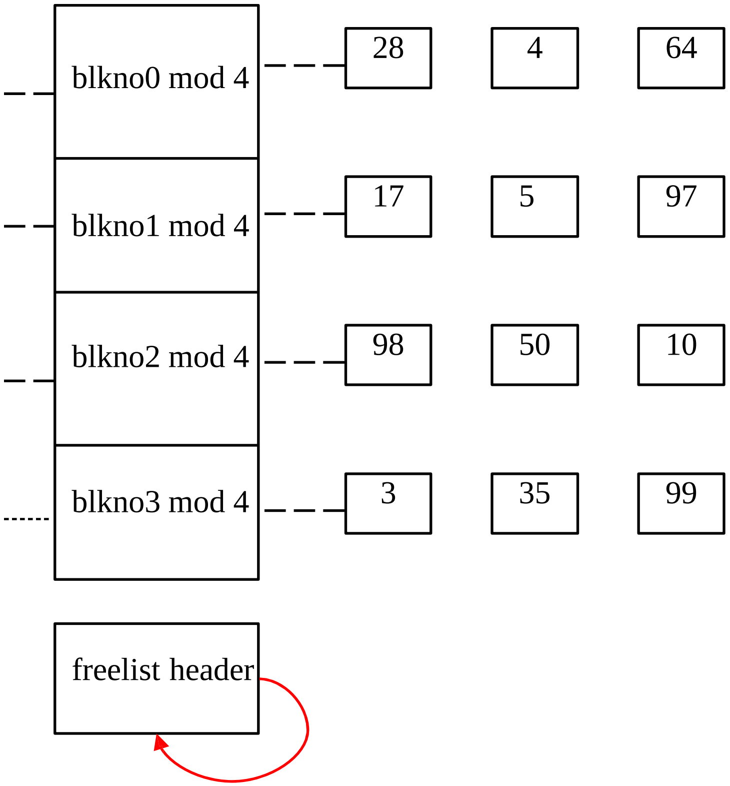


Search for block 18, free list empty

Scenarios for retrieval of a buffer

- The fourth scenario

Hash queue headers



blkno0 mod 4 — 28, 4, 64

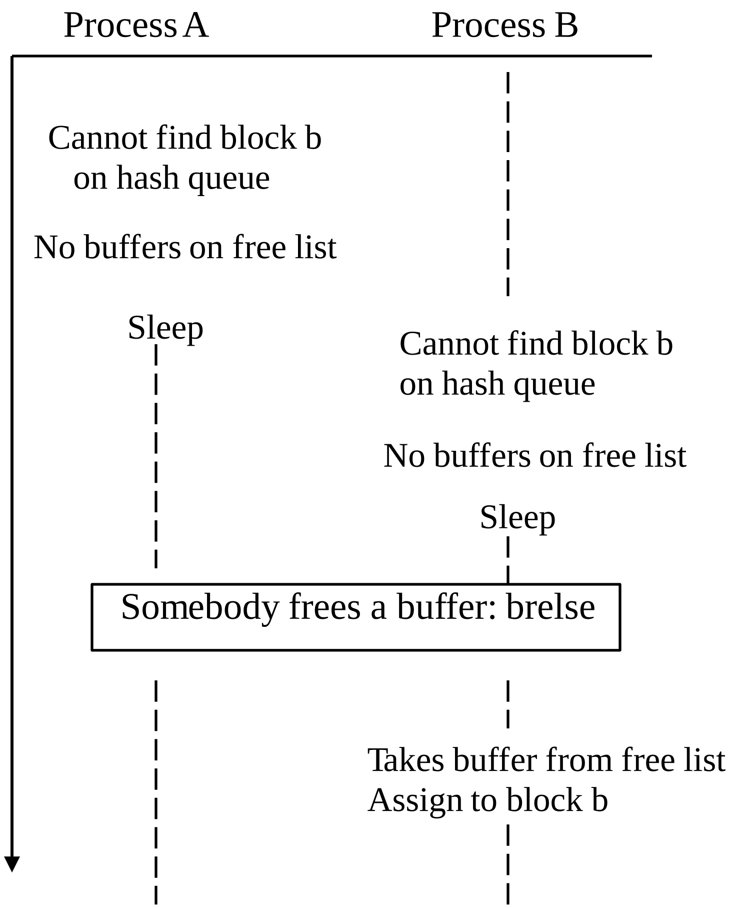
blkno1 mod 4 — 17, 5, 97

blkno2 mod 4 — 98, 50, 10

blkno3 mod 4 — 3, 35, 99

freelist header

Figure 3.9 Fourth Scenario for Allocating Buffer



Process A

Process B

Cannot find block b
on hash queue

No buffers on free list

Sleep

Cannot find block b
on hash queue

No buffers on free list

Sleep

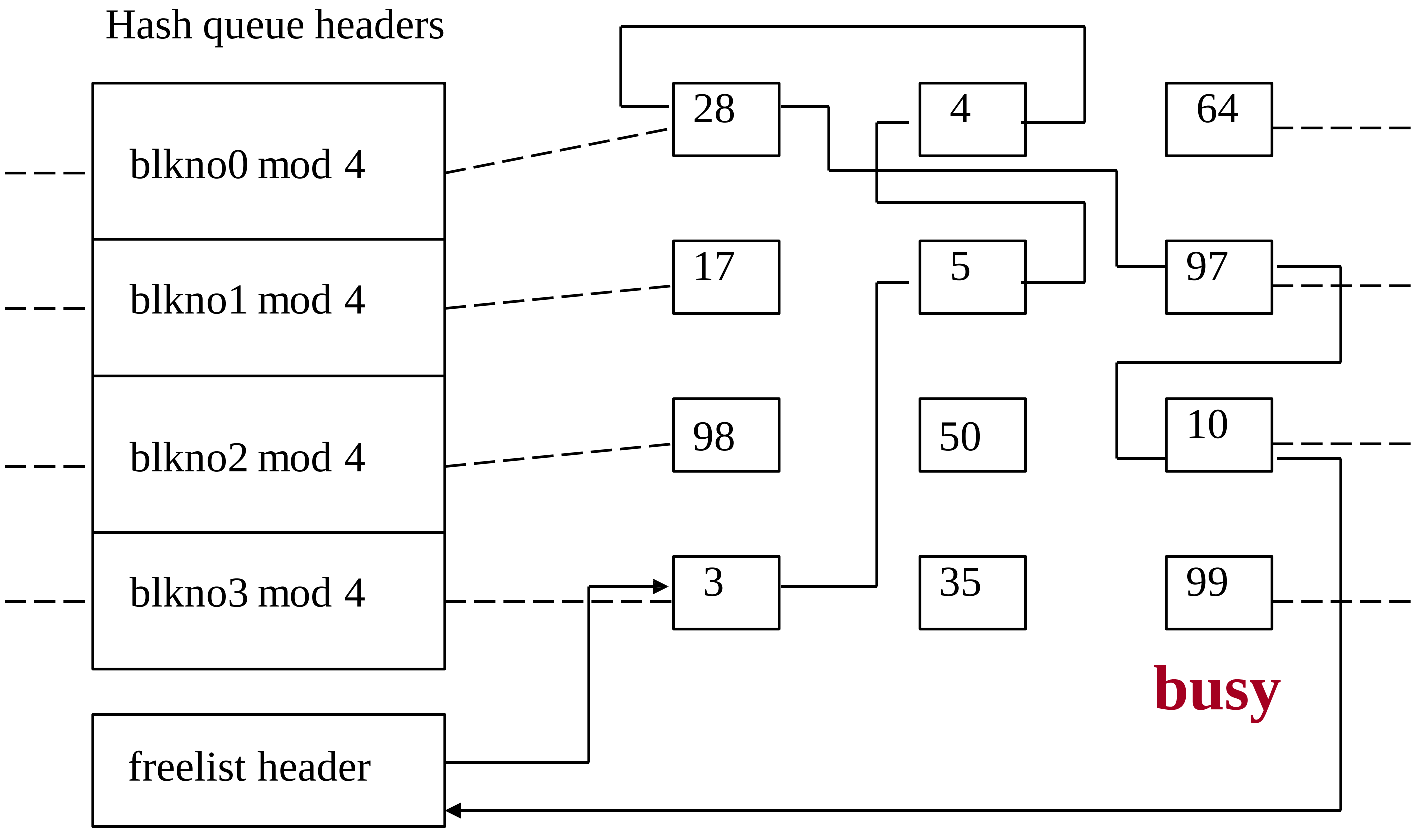
Somebody frees a buffer: brelse

Takes buffer from free list
Assign to block b

Figure 3.10 Race for Free Buffer

Retrieval of a Buffer: 5th Scenario

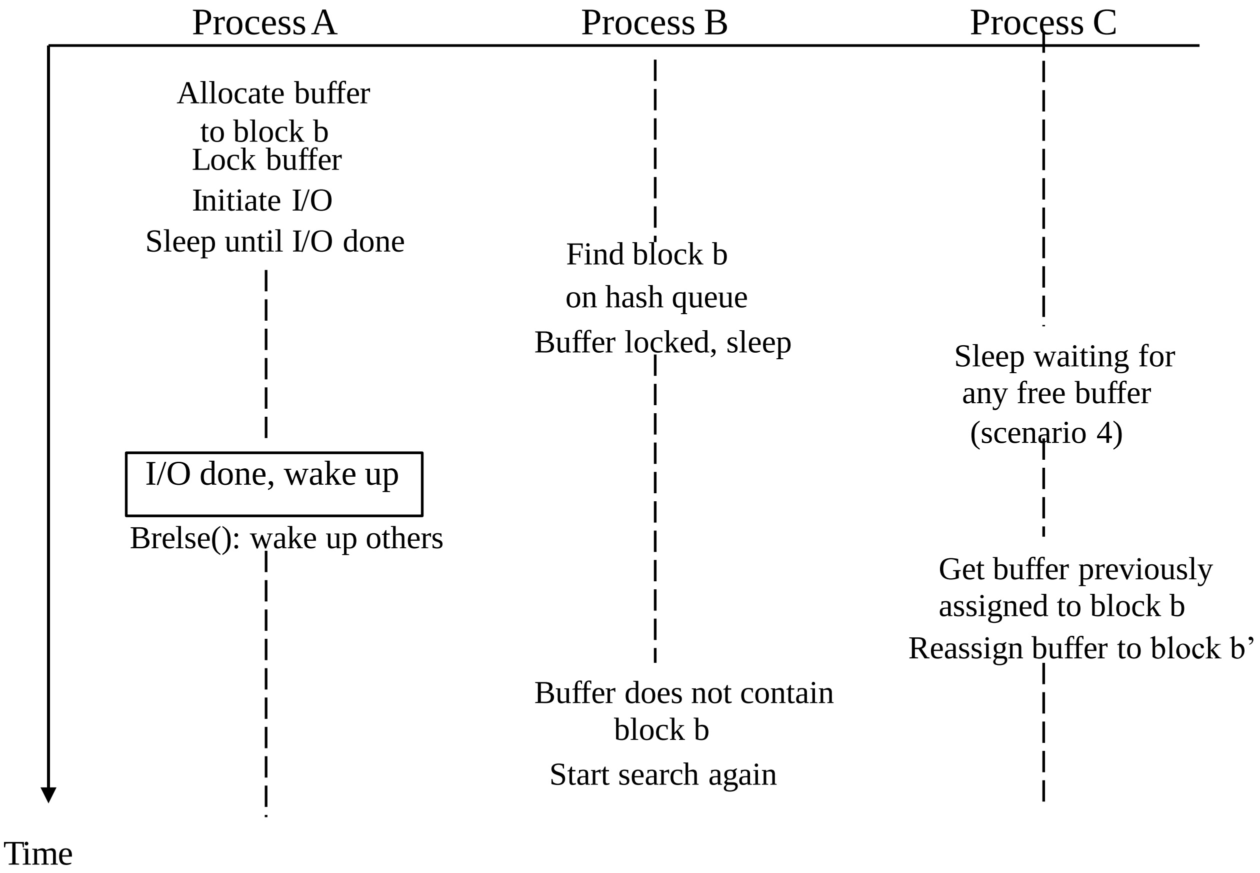
- Kernel finds the buffer on hash queue, but it is currently busy



Search for block 99, block busy

Scenarios for retrieval of a buffer

- The fifth scenario



Process A

Process B

Process C

Allocate buffer to block b

Lock buffer

Initiate I/O

Sleep until I/O done

Find block b on hash queue

Buffer locked, sleep

Sleep waiting for any free buffer (scenario 4)

I/O done, wake up

Brelse(): wake up others

Get buffer previously assigned to block b

Reassign buffer to block b’

Buffer does not contain block b

Start search again

Time

Figure 3.12 Race for a Locked Buffer

The algorithm for buffer allocation is safe:

Process must not sleep forever and kernel guarantees that all process waiting for buffers will wakeup ,because it allocates buffers during the execution of system call and frees them before returning.

Process in user mode do not control the allocation of kernel buffers directly.

The kernel looses the control over buffer only when it waits for the completion of I/O between the buffer and the disk.

If several processes sleep while waiting for a buffer to become free, the kernel doesn't guarantee that they get a buffer in the order that they requested one.

Algorithm: GetBlock

- GetBlock (file_system_no,block_no)
  - while (buffer not found)
    - if (block / buffer in hash queue)
      - if (buffer busy)
        - sleep (event buffer becomes free)
        - continue mark
      - buffer busy
      - remove buffer from free list
      - return buffer
    - else
      - if (there is no buffer on free list)
        - sleep (event any buffer becomes free)
        - continue
      - remove buffer from free list
      - if (buffer marked as delayed write)
        - asyschronous write buffer to disk
        - continue
      - remove buffer from hash queue
      - put buffer onto hash queue
      - return buffer

Algorithm for buffer allocation

Input : file system number
        Block number

Output : locked buffer that can now be used for block

```
while(buffer not found)
{
    if(block in hash queue)
    {                                   /*scenario 5*/
        if(buffer busy)
        {
                sleep(event buffer becomes free);
                continues;              /*scenario1 */
                                        /* back to while */
        }
        mark buffer busy;
        remove buffer from free list;
        return buffer;
    }
```

Algorithm for buffer allocation

```
else {
        if(there are no buffer on free list)
        {                                                       /*scenario 4*/
                sleep(event any buffer becomes free);
                continue;                               /*back to while loop*/
        }
        remove buffer from free list;
        if(buffer marked for delayed write)
            {                                                   /*scenario 3*/
                asynchronous write buffer to disk;
                continue;                               /*back to while loop*/
        }                                       /*scenario  2- found a free buffer */
        remove buffer from old hash queue;
        put buffer onto new hash queue;
        return buffer;
}
```

Reading and Writing disk blocks

- To read block ahead
  - The kernel checks if the block is in the cache or not.
  - If the block in not in the cache, it invokes the disk driver to read the block.
  - Then the process goes to sleep awaiting the event that the I/O is complete.
  - The disk controller interrupts the processor when the I/O is complete
  - The disk interrupt handler awakens the sleeping processes
  - The content of disk blocks are now in the buffer
  - When the process no longer need the buffer, it releases it so that other processes can access it

Reading and Writing disk blocks

- To write a disk block
  - The kernel informs the disk driver that it has a buffer whose contents should be output.
  - The disk driver schedules the block for I/O.
  - If the write is synchronous, the calling process goes to sleep awaiting I/O completion and releases the buffer when awakens.
  - If the write is asynchronous, the kernel starts the disk write but doesnot wait for the write to complete.
  - The kernel release the buffer when the I/O completes.

Reading a disk block

```
algorithm bread      /*block bread*/
input: file system block number
output: buffer containing data
{
    get buffer for block(algorithm getblk);
    if(buffer data valid)
           return buffer;
    initiate disk read;
    sleep(event disk read complete);
    return(buffer);
}
```

Read Ahead

- Read Ahead
  - Improving performance
    - Read additional block before request
  - Use breada()

Reading Disk Blocks

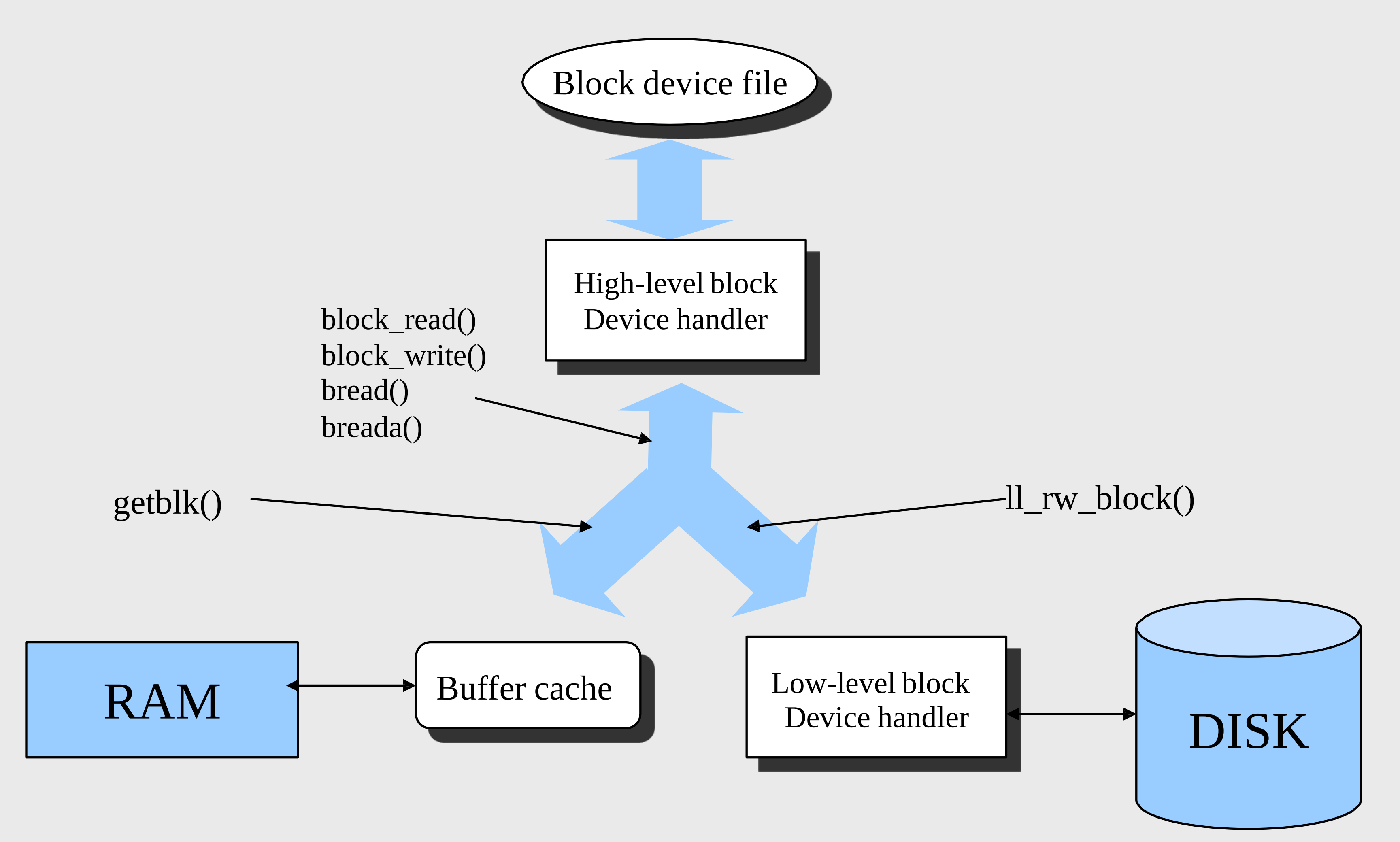


Figure : block device handler architecture for buffer I/O operation in Understanding the Linux Kernel

Read Ahead

```
Algorithm breada
I/P : 1) file system block number for immediate read
      2) file system block number for asynchronous read
O/P : buffer containing data for immediate read
{
    if(first block not in cache)
    {
        get buffer for first block(algorithm getblk);
        if(buffer data not valid)
                initiate disk read;
    }
```

Read Ahead(…)

```
if(second block not in cache)
  {
      get buffer for second block(algorithm getblk);
      if(buffer data valid)
          release buffer(algorithm brelse);
      else
          initiate disk read;
  }  // end of main()
```

Read Ahead(…)

```
    if(first block was originally in cache)
    {
        read first block(algorithm bread);
        return buffer;
    }
    sleep(event first buffer contains valid data);
    return buffer;
}
```

Writing disk Block

- Synchronous write
  - the calling process goes the sleep awaiting I/O completion and releases the buffer when awakens.
- Asynchronous write
  - the kernel starts the disk write but does not wait for write to complete. The kernel release the buffer when the I/O completes
- Delayed write
  - The kernel put off the physical write to disk until buffer reallocated
  - Look Scenario 3
- Relese : Use brelse()

Writing a disk block

```
algorithm bwrite /*block write*/
Input: buffer                    Output: none
{
    initiate disk write;
    if(I/O synchronous)
    {
        sleep(event I/O complete);
        release buffer(algorithm brelse);
    }
    else if(buffer marked for delayed write)
        mark buffer to put at head of free list;
}
```

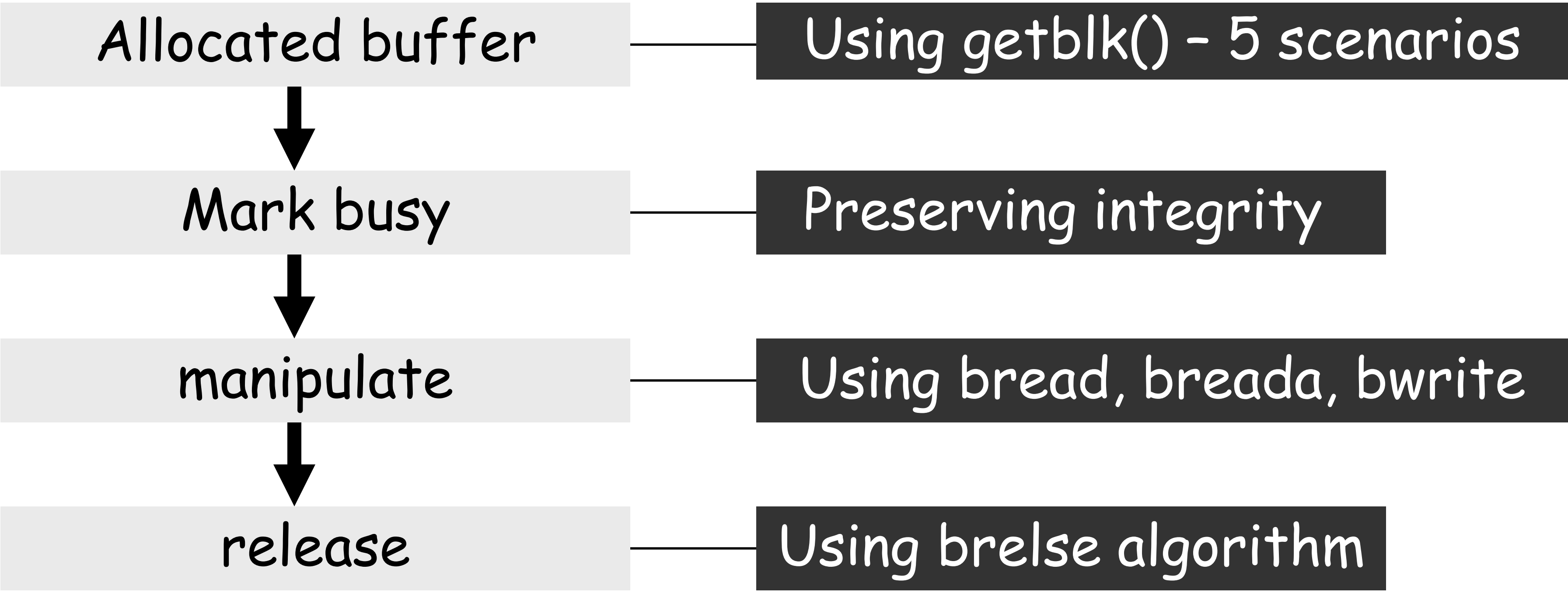
Advantages of the buffer cache

- Eliminates the need for special alignment of user buffers
  - by copying data from user buffers to system buffers,
- Reduce the amount of disk traffic, increasing throughput and decreasing response time.
- Allows uniform disk access using buffers
  - Kernel does not need to know the reason for the I/O : system design is simpler.
- Insure file system integrity
  - one disk block is in only one buffer

Disadvantages of the buffer cache

- Can be vulnerable to crashes
  - When delayed write
- requires an extra data copy
  - When reading and writing to and from user processes

What happen to buffer until now



Allocated buffer — Using getblk() – 5 scenarios

Mark busy — Preserving integrity

manipulate — Using bread, breada, bwrite

release — Using brelse algorithm