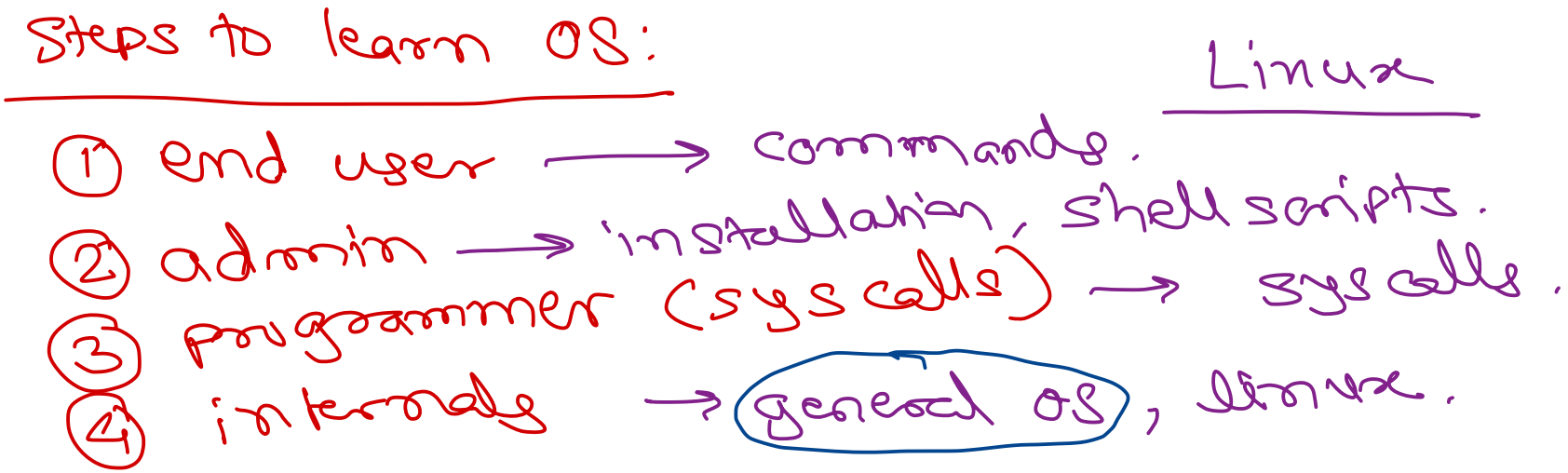
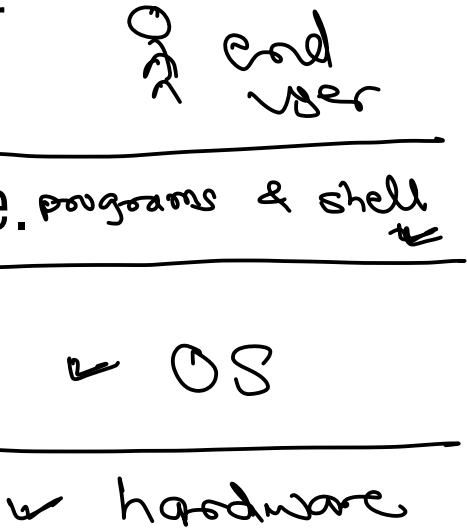
Operating System Introduction

Sunbeam Infotech



What is an Operating System?

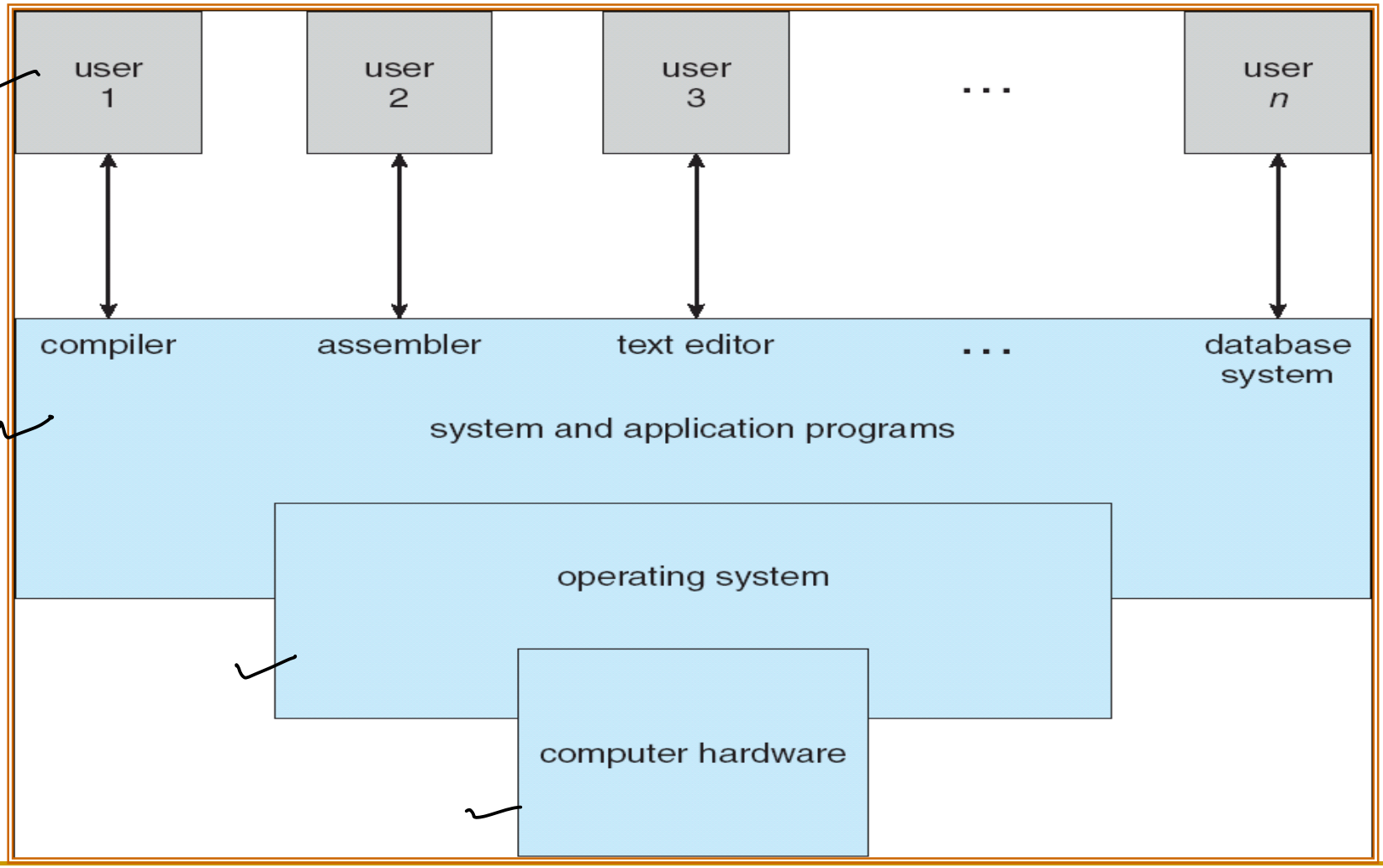
- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.



Computer System Structure

- Hardware – provides basic computing resources
  - CPU, memory, I/O devices
- Operating system
  - Controls and coordinates use of hardware among various applications and users
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
  - Word processors, compilers, web browsers, database systems, video games, shell
- Users
  - People, machines, other computers

Four Components of Computer System



Operating System Definition

windows : ntoskrnl.exe
linux: vmlinuz

- OS is a resource allocator
  - Manages all resources and
  - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer
- Everything a vendor ships when you order an operating system

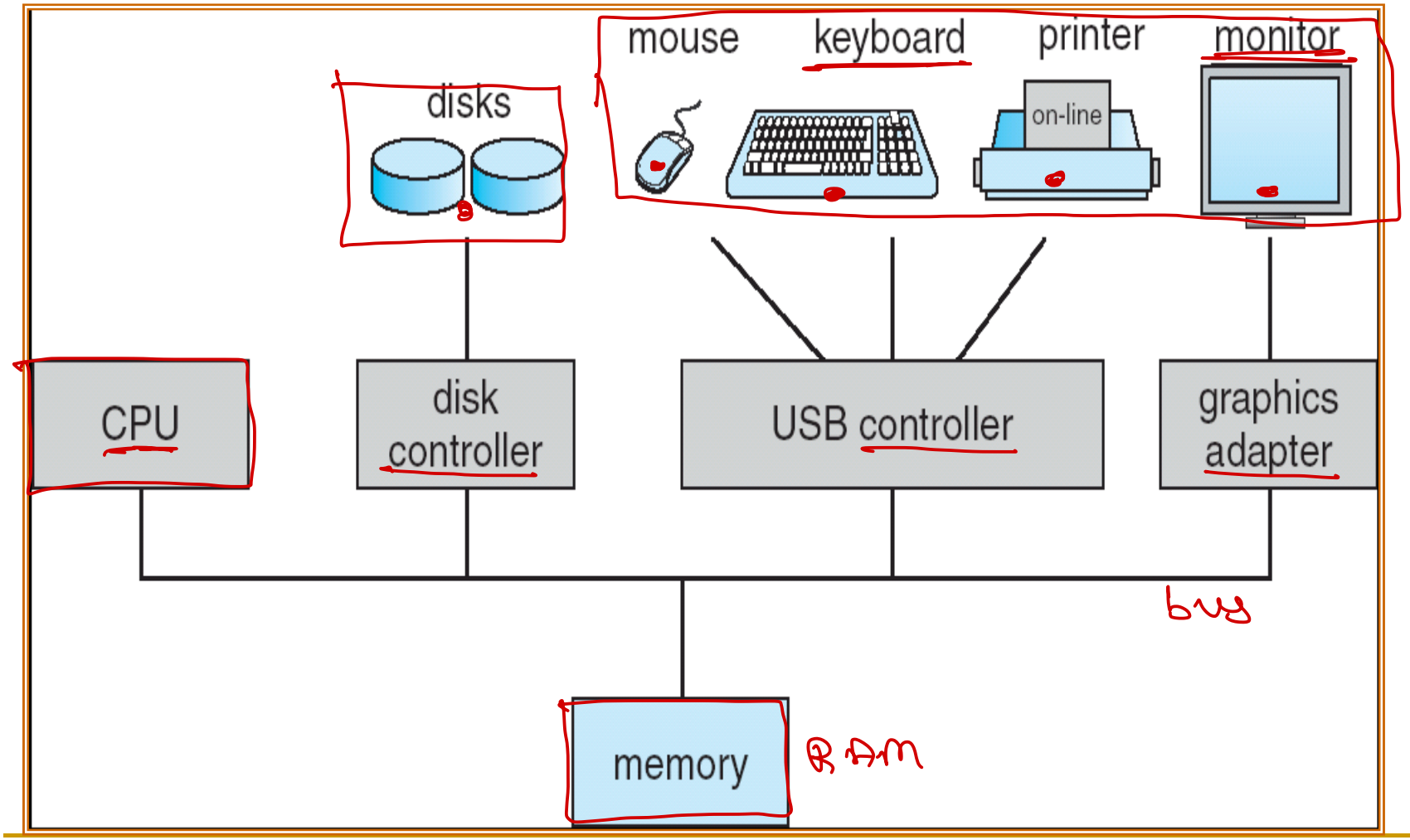
CD/DVD = Core OS + programs + utilities

- "The one program running at all times on the computer" is the kernel. Everything else is either a system program (ships with the operating system) or an application program

Computer System

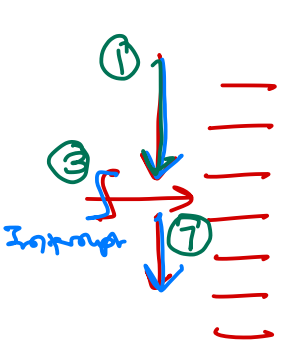
Sunbeam Infotech

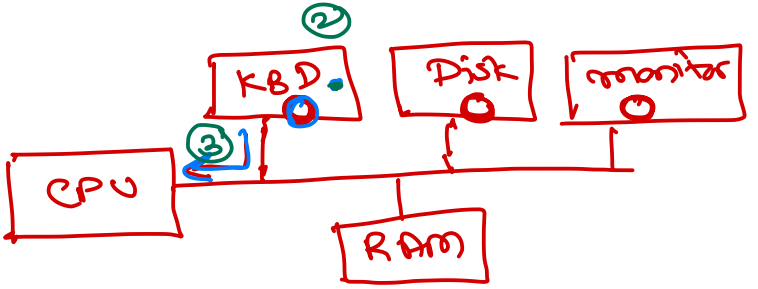
Computer System Structure



Computer-System Operation

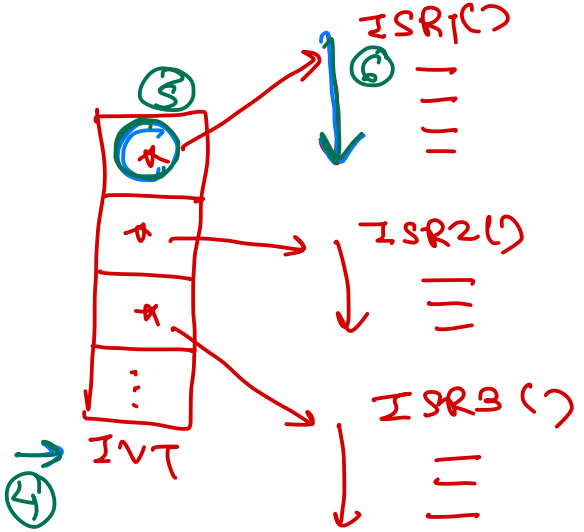
- One or more CPUs, device controllers connect through common bus providing access to shared memory.
- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type and has a local buffer.
- CPU moves data from/to main memory to/from local buffers.
- Actual input and output occurs between the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an interrupt.



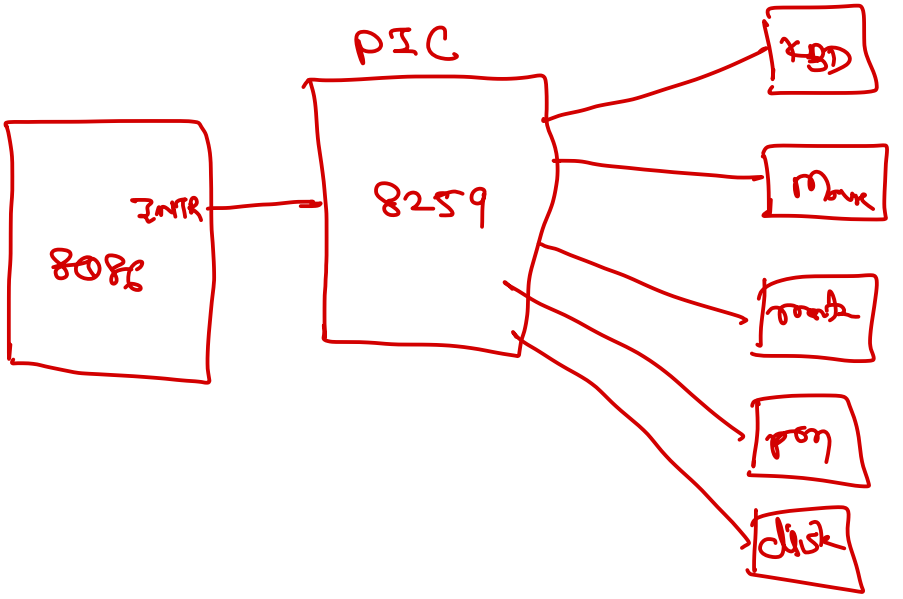


① polling (sync IO)

② interrupt (async IO)



8086 arch



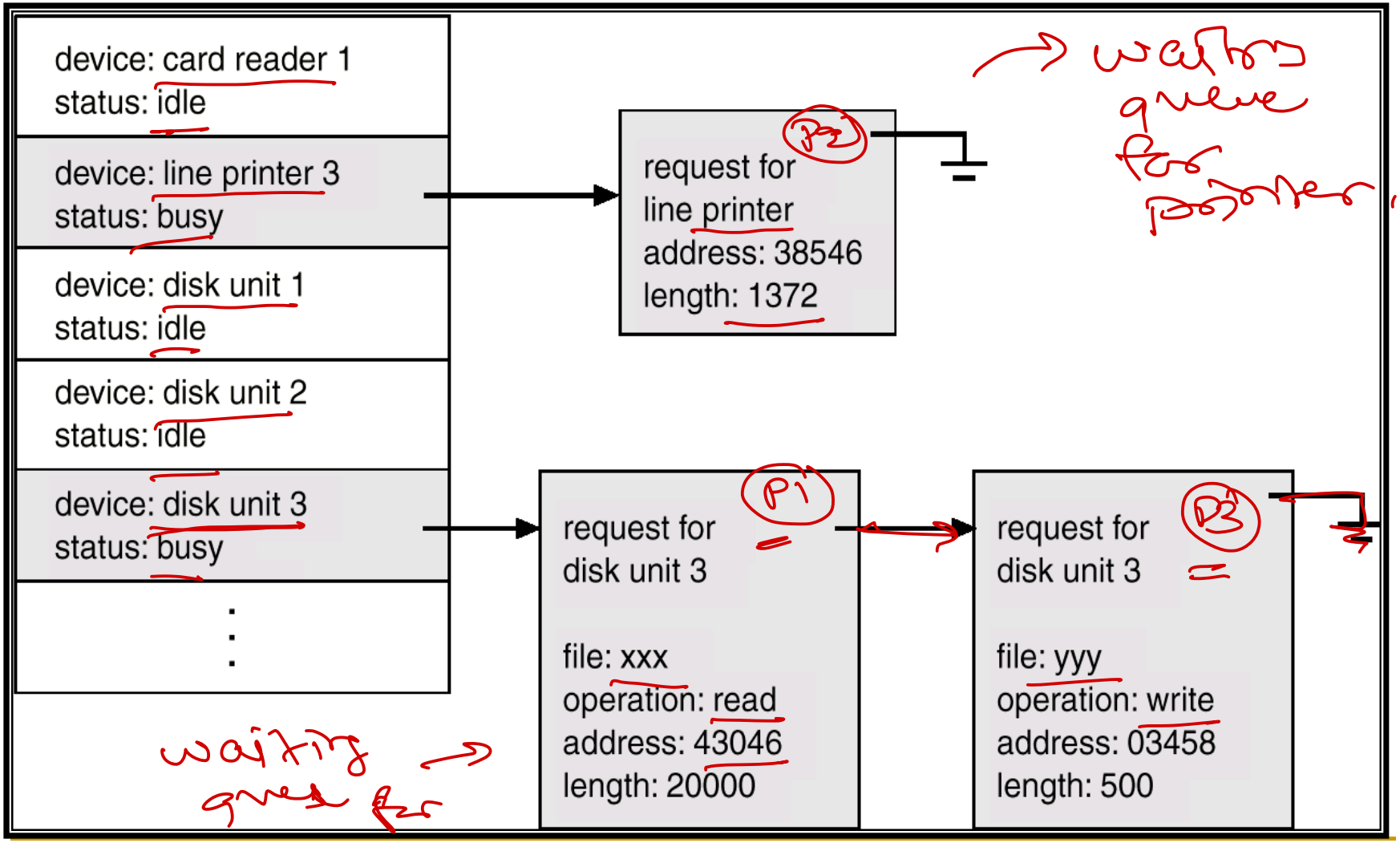
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all service routines.
- Interrupt architecture must save the address of the interrupted instruction. The OS preserves the state of the CPU by storing registers and the program counter.
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
- A trap is a software-generated interrupt caused either by an error or a user request.
- An operating system is interrupt driven.

I/O Structure

- In synchronous I/O method, after I/O starts, control returns to user program only upon completing I/O
  - wait instruction or loop idles the CPU until the next interrupt
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- In asynchronous method, after I/O starts, control return to user program without waiting for I/O completion
  - System call – request to the operating system to allow user to wait for I/O completion.
  - Device-status table contains entry for each I/O device indicating its type, address, and state.
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Device-Status Table



Operating System Concepts

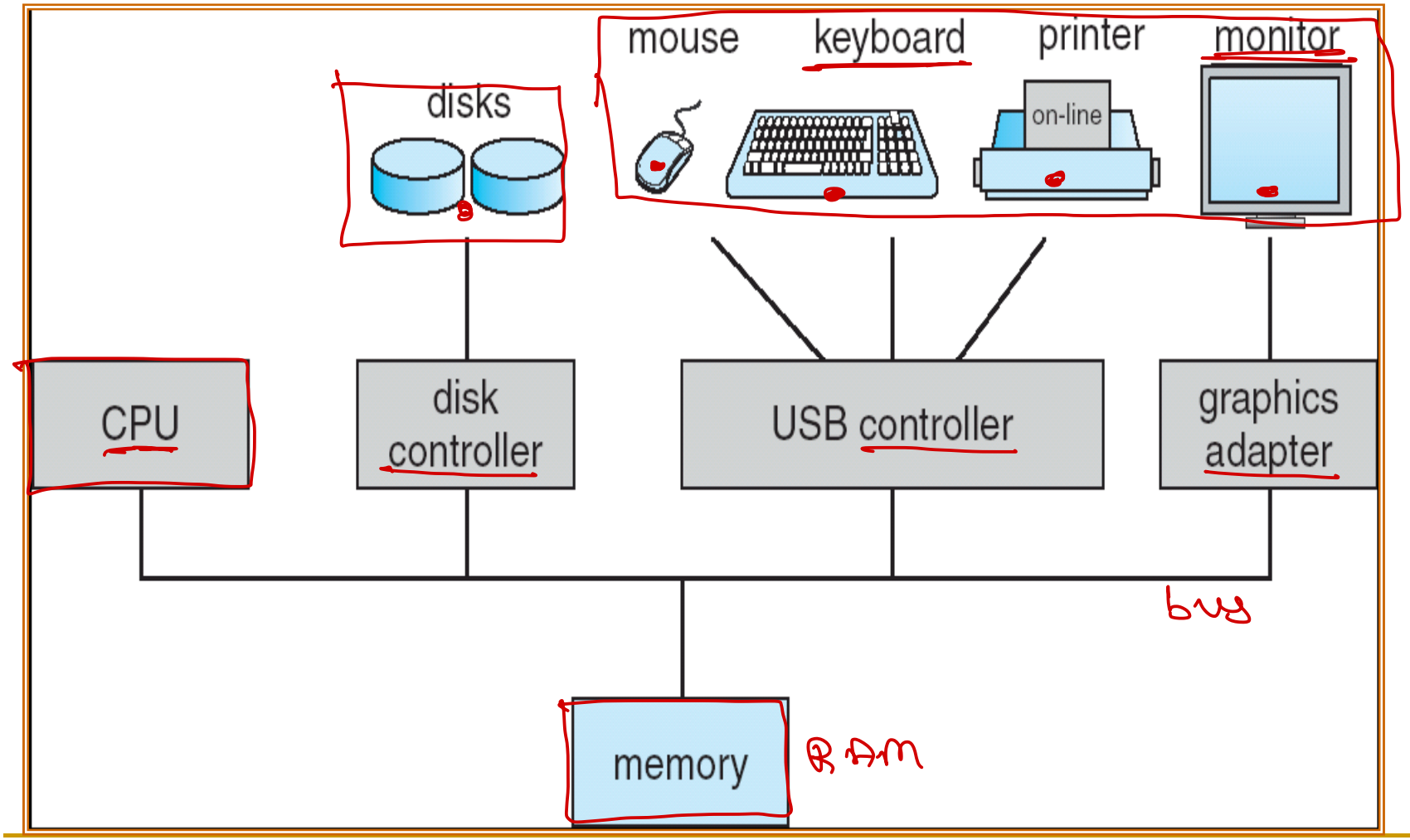
Source: Galvin OS books/slides

Edited by: Nilesh Ghule

Computer System

Sunbeam Infotech

Computer System Structure



Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.

Memories

Primary
(directly accessible by CPU)

1. CPU registers
2. Cache
3. RAM (main memory)

(Memory)

Secondary
(accessible by CPU via primary memory)

1. hard disk
2. optical disk
3. magnetic tapes.
4. ROM

(Storage)

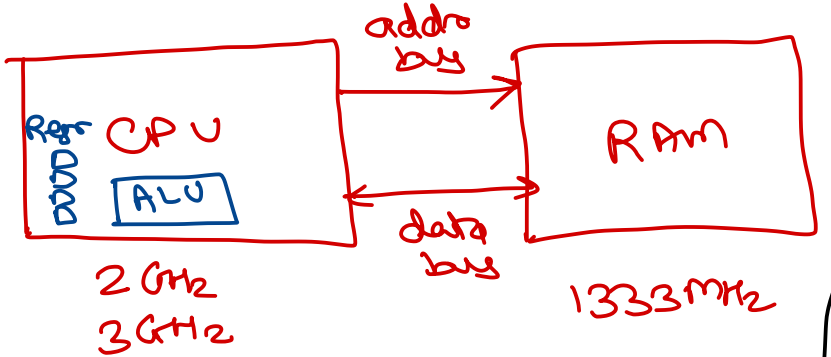
Storage Structure

- Main memory – only large storage media that the CPU can access directly.
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into tracks, which are subdivided into sectors.
  - The disk controller determines the logical interaction between the device and the computer.
- Storage systems organized in hierarchy.
  - Speed, Cost, Volatility

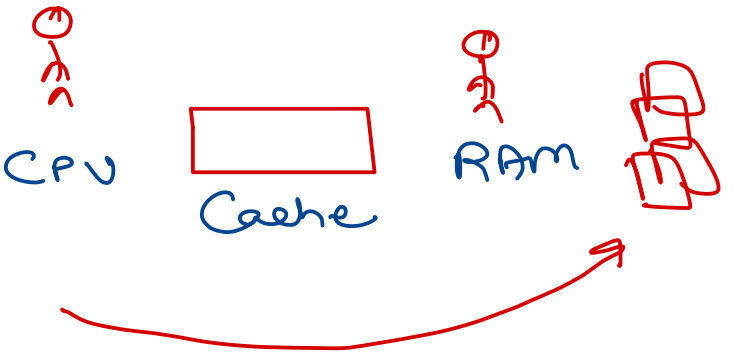
Main Memory

- The main memory and processor registers are directly accessed by the CPU.
- The machine instructions take the addresses from main memory or registers, not the disk.
- In concept of memory mapped I/O, a set of memory addresses are reserved to tie with device registers.
- Even I/O ports are mapped with some addresses.
- The user program or OS writes data on these addresses and then set control register to send the data.
- Access to main memory is slower than the registers.

RAM



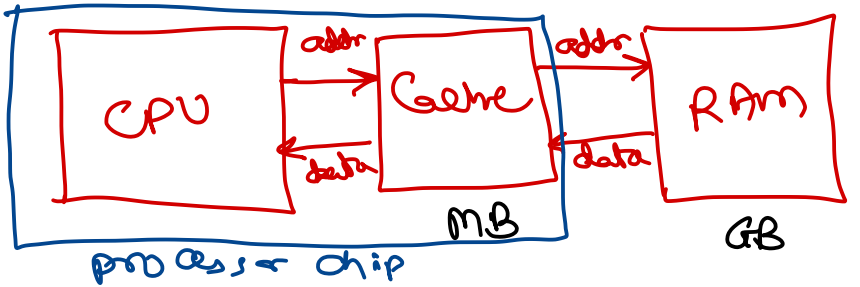
Cache



Cache is high speed associative memory

* search by addr is very fast.
* address → data.
* Least recently accessed data is overwritten.
* Cache always contains recent data only.

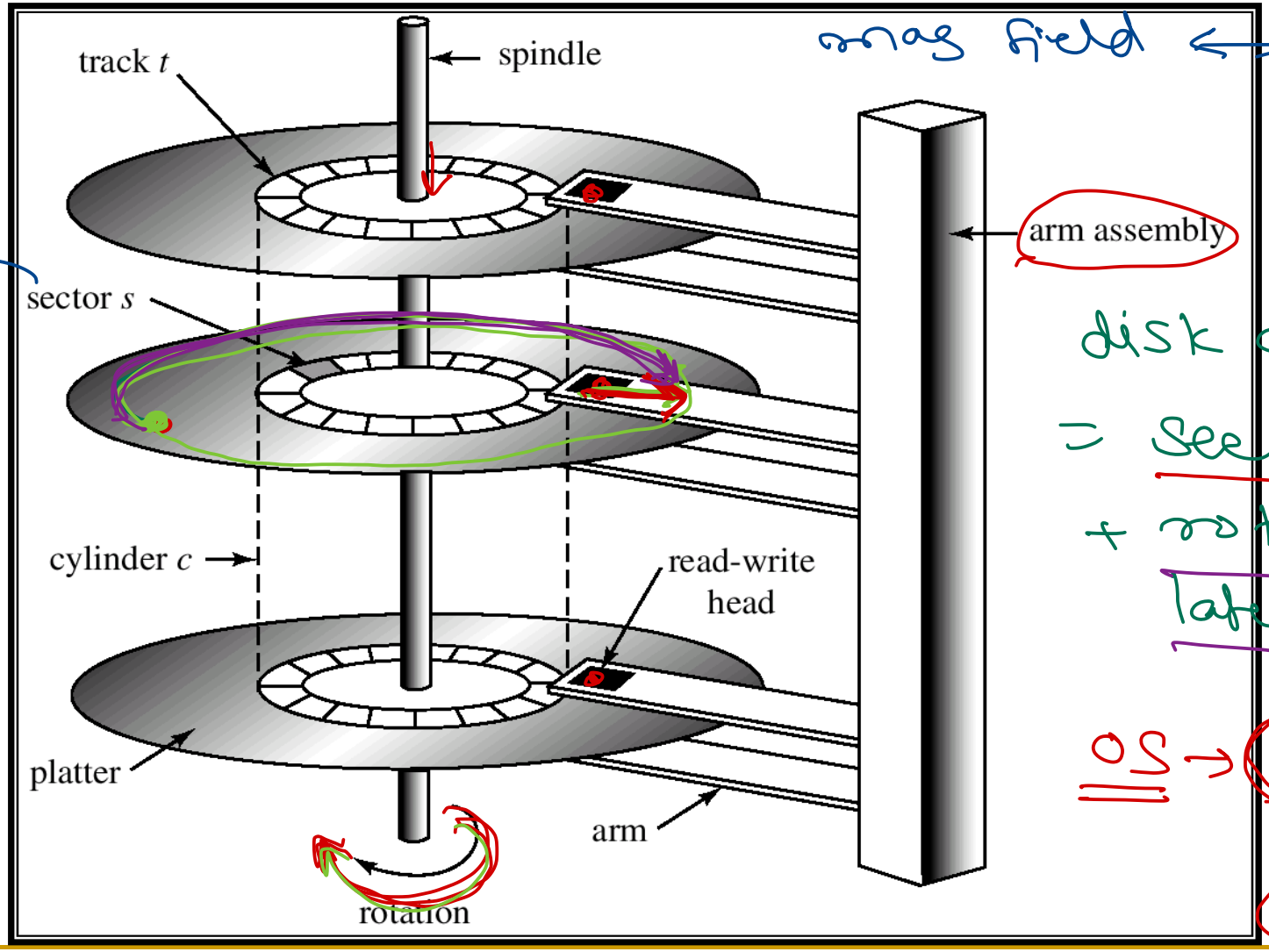
Cache hit
Cache miss



* magnetic dipole
N → S → 1
S → N → 0
* faraday's laws

Magnetic Disks

mag field ↔ current



512 bytes

disk access time
= seek time
+ rotational latency

CHS

OS → LBA → CHS

disk capacity = H * T per head * S per track * 512

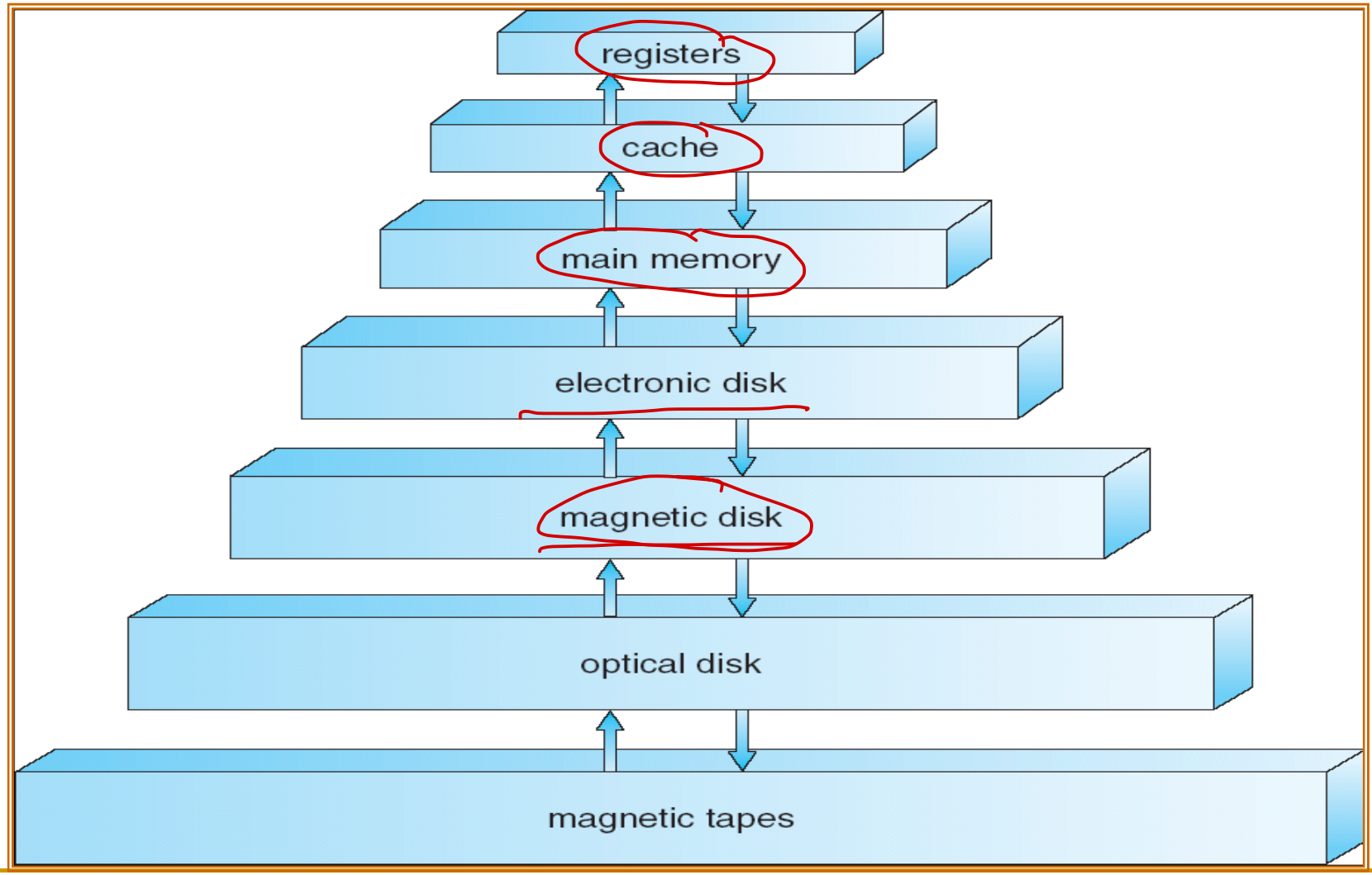
Magnetic Disks

- Magnetic disk speed is based on two factors
  - Transfer rate: data flow rate betⁿ drive and computer
  - Positioning time or Random access time
    - Seek time: move disk arm to desired cylinder
    - Rotation latency: rotate disk head to desired sector
- Drive is attached to computer through I/O bus
- Different types of buses are
  - Enhanced integrated drive electronics (EIDE)
  - Advanced technology attachment (ATA) : (PATA/SATA)
  - Small computer systems interface (SCSI)
- Host controller is connected at computer end of bus connecting to drive and do the I/O.

Performance of Various Levels of Storage

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

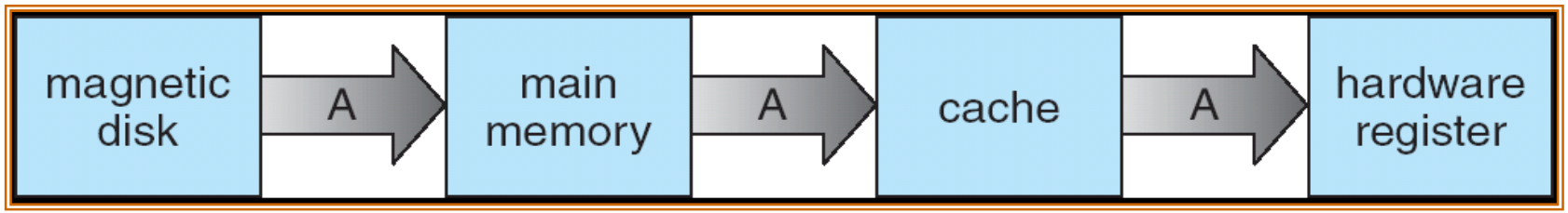
Storage-Device Hierarchy



Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Migration of int A from disk to register



- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy
- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist

Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule

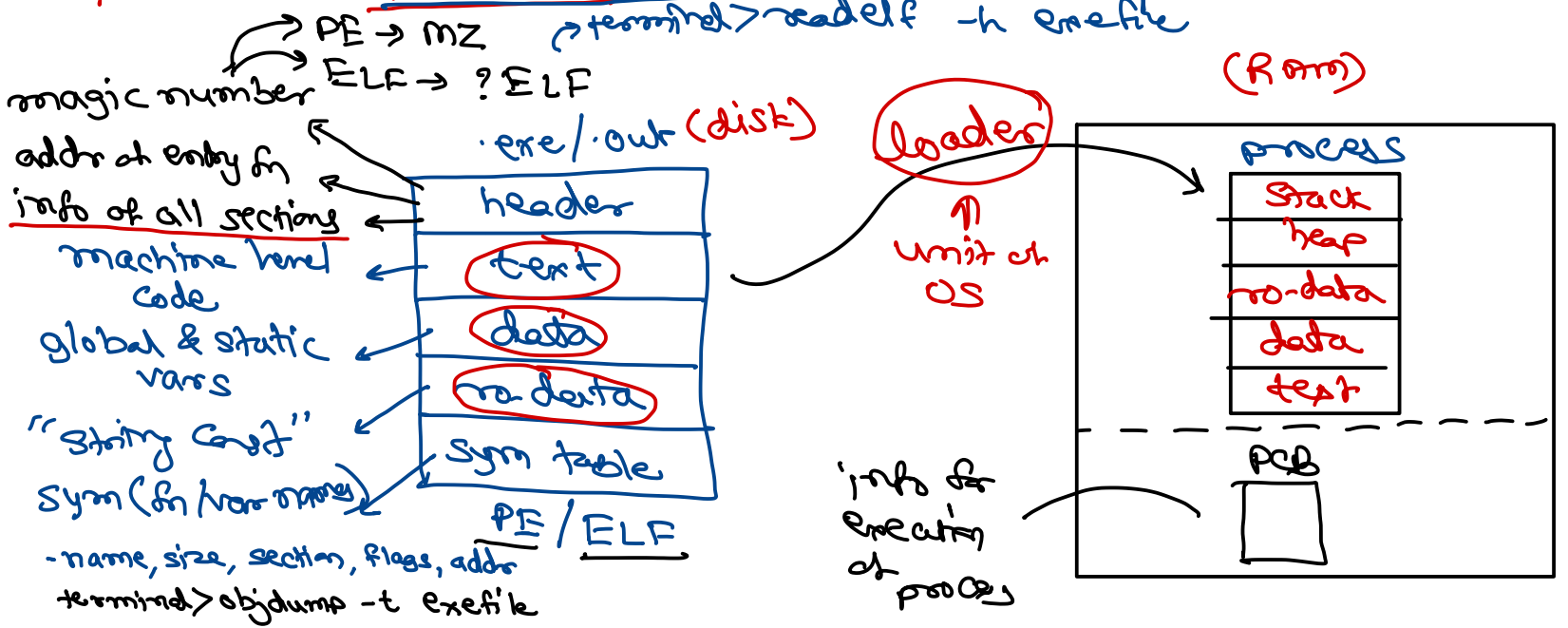
processes have lives and files have spaces. – UNIX.

UNIX OS = Process subsystem + File subsystem.

Process

Program: set of instructions given to the computer ↔ executable file

process: program under execution



File

file is collection of data/info on storage device

file = contents (data) + info (metadata)

disk → data blocks

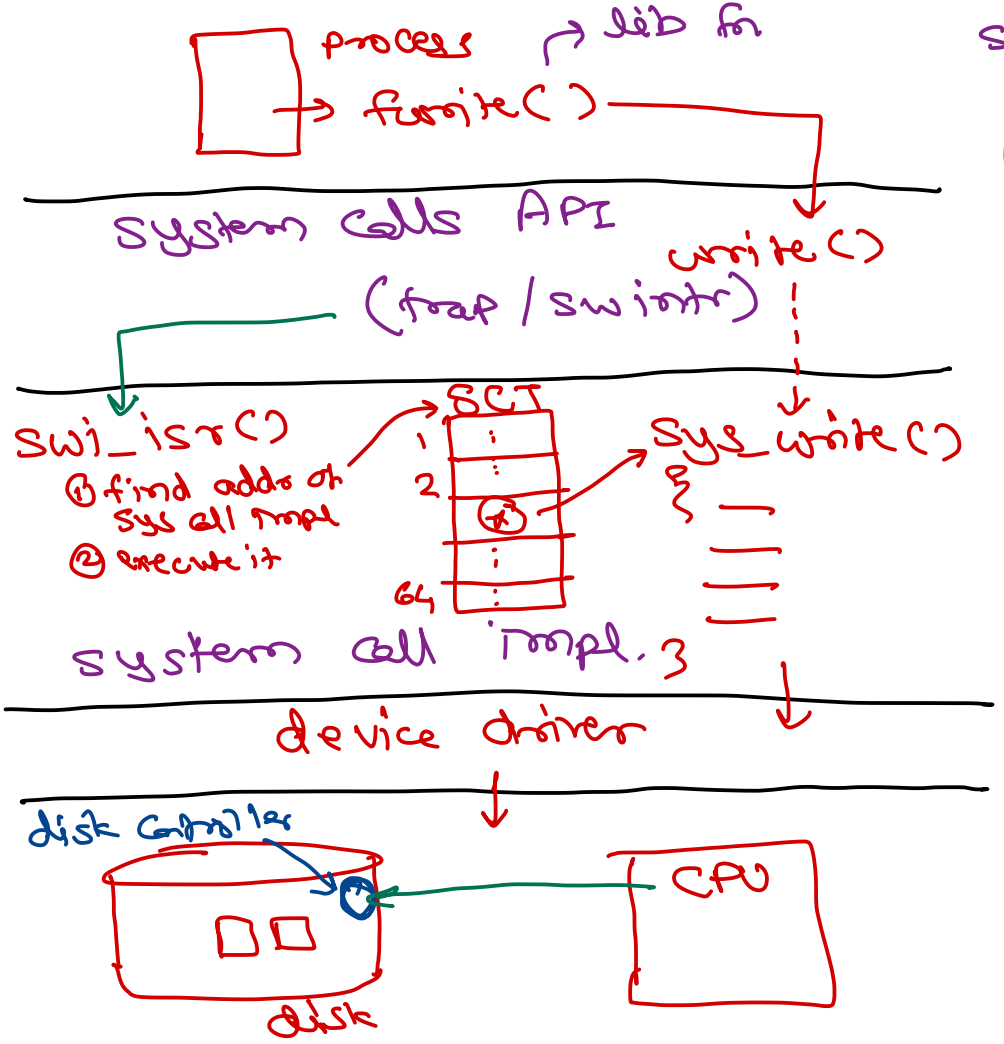
FCB (inode)

1. type
2. size
3. permissions (mode)
4. user & group
5. timestamps

...

Hardware Protection

- Early operating systems work as resident monitors.
- Then OS start doing additional jobs like I/O, resource allocator, etc.
- In multiprogramming environment, one program could disturb other program in memory by corrupting its data.
- The programming errors are detected by hardware and conveyed to operating system via interrupt. OS should take appropriate action like terminating victim program.
- The following protection mechanisms are available:
  - Dual-Mode Operation
  - I/O Protection
  - Memory Protection
  - CPU Protection



Syscalls are fns exposed by the kernel, so that user programs can access kernel functionality

UNIX : 64 sys calls.

Linux : >300 sys calls.

Software interrupt

A special assembly instruction (arch dependent) → Cause execution of ISR.

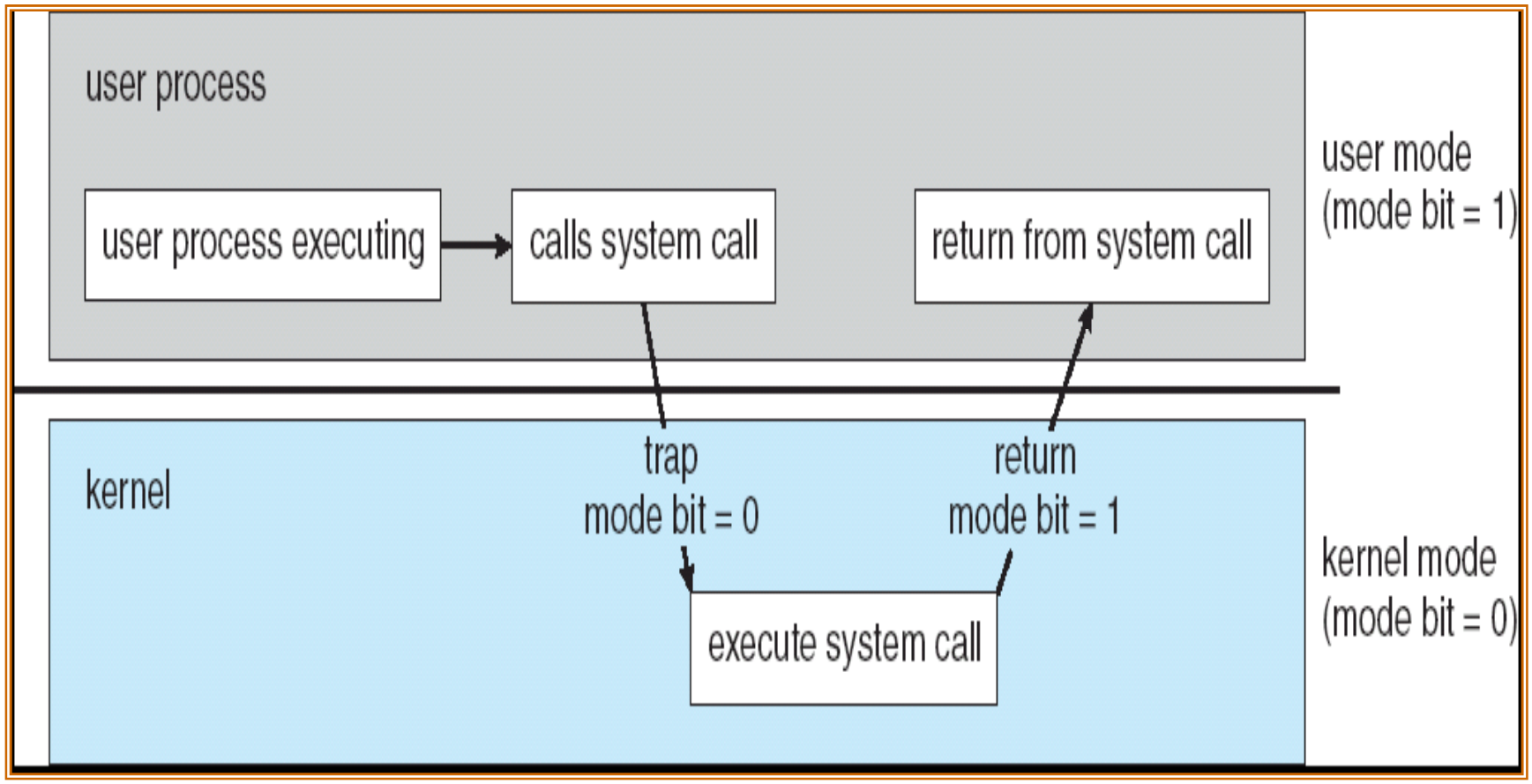
x86 : INT

ARM : SWI / SVC

Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
  - User mode – execution done on behalf of a user.
  - Monitor mode (also kernel mode or system mode) – execution done on behalf of operating system.
- Mode bit added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.

User mode and Kernel mode



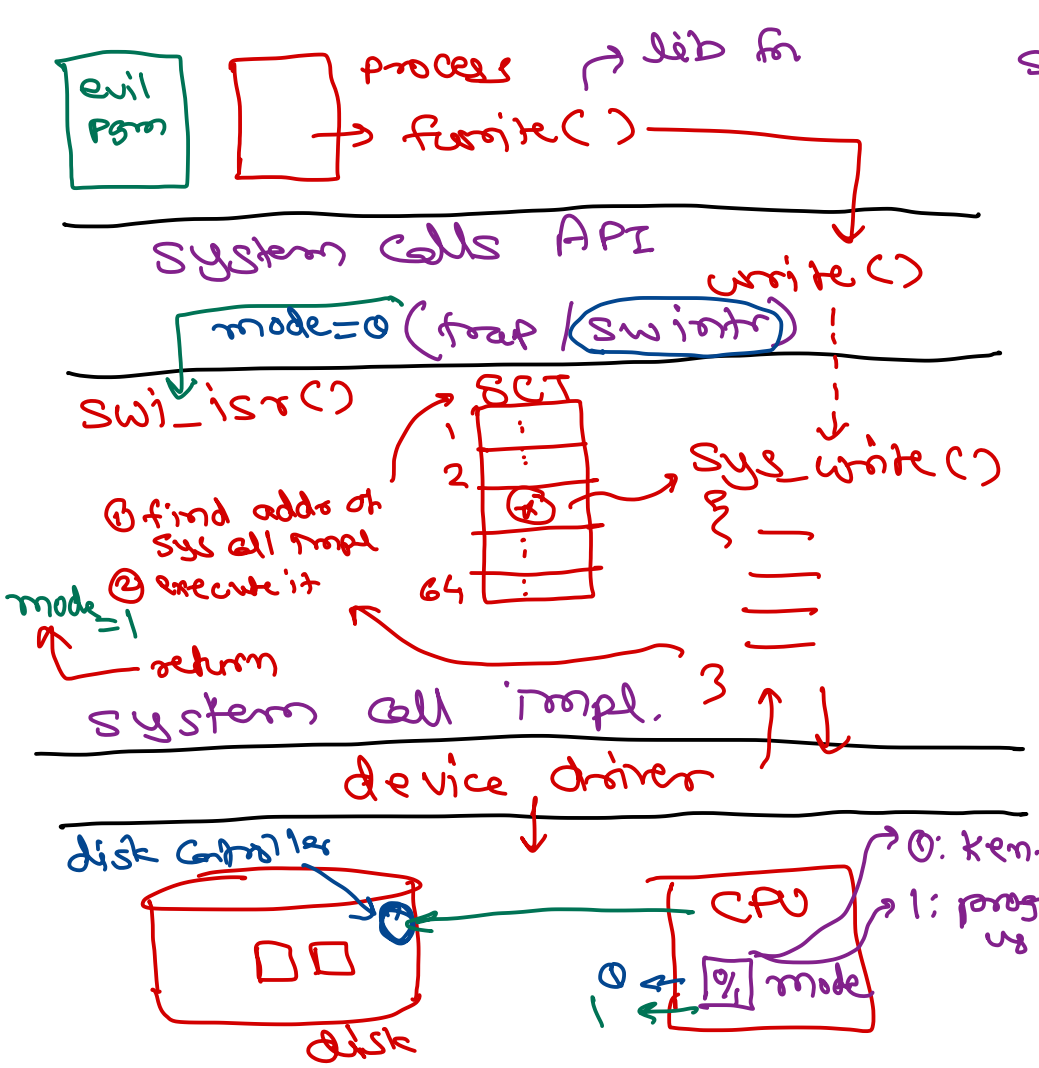
Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule

Hardware Protection

- Early operating systems work as resident monitors.
- Then OS start doing additional jobs like I/O, resource allocator, etc.
- In multiprogramming environment, one program could disturb other program in memory by corrupting its data.
- The programming errors are detected by hardware and conveyed to operating system via interrupt. OS should take appropriate action like terminating victim program.
- The following protection mechanisms are available:
  - Dual-Mode Operation
  - I/O Protection
  - Memory Protection
  - CPU Protection



Syscalls are fns exposed by the kernel, so that user programs can access kernel functionality

UNIX : 64 sys calls.

Linux : >300 sys calls.

Software interrupt

a special assembly instruction (arch dependent) → cause execution of ISR.

x86 : INT

ARM : SWI / SVC

(*) mode bit is in (modern) CPU.

when mode bit is 0 → kernel code is running

" " " is 1 → user code is running

(*) mode = 0 → kernel / system / monitor / privileged

mode = 1 → user / non-privileged.

(*) privileged mode special instns are also allowed

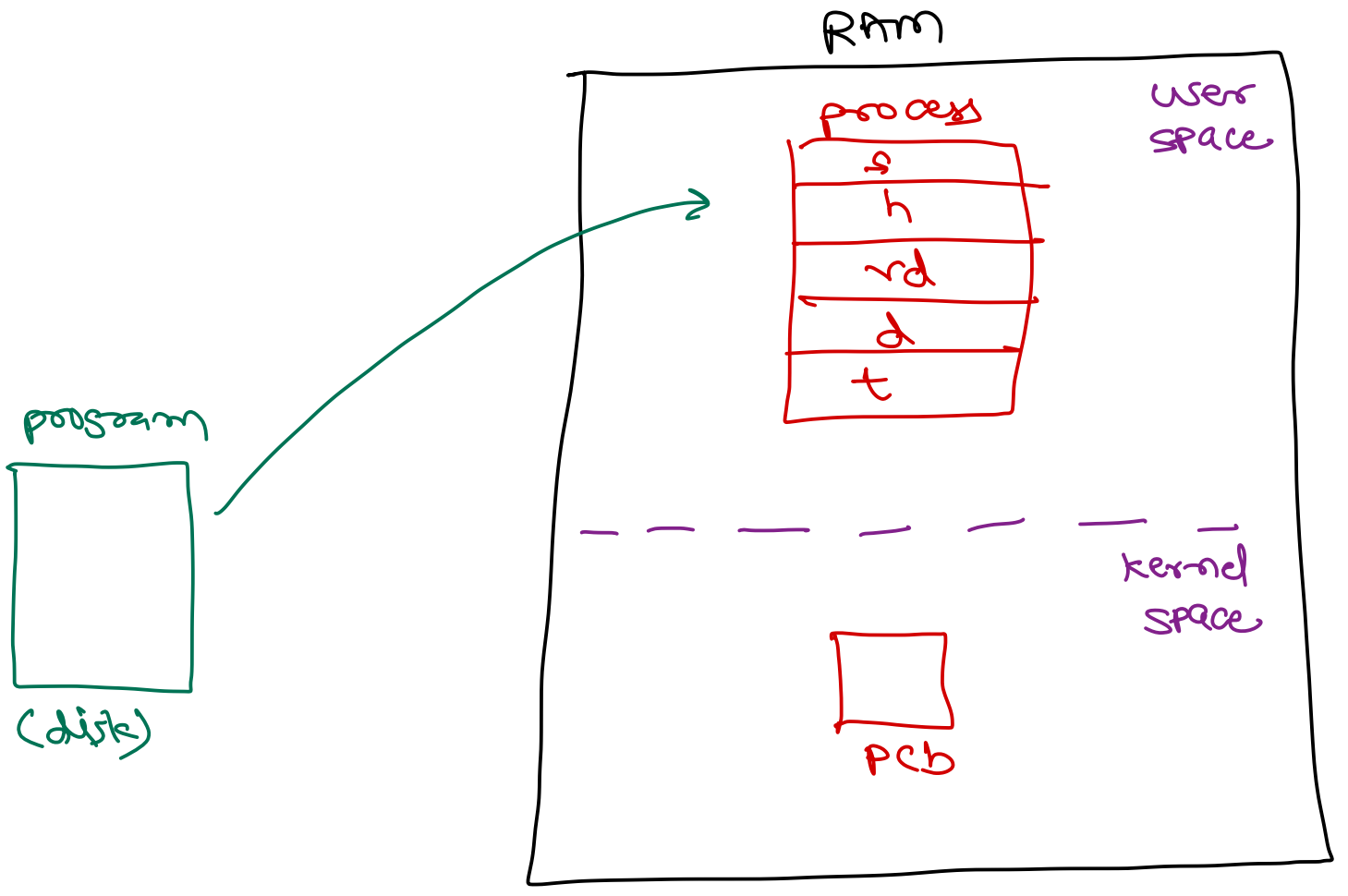
e.g. IO.

non-privileged mode allows only general purpose instructions.

(*) In hw, when interrupt arrives, mode = 0.

when ISR is completed, mode = 1.

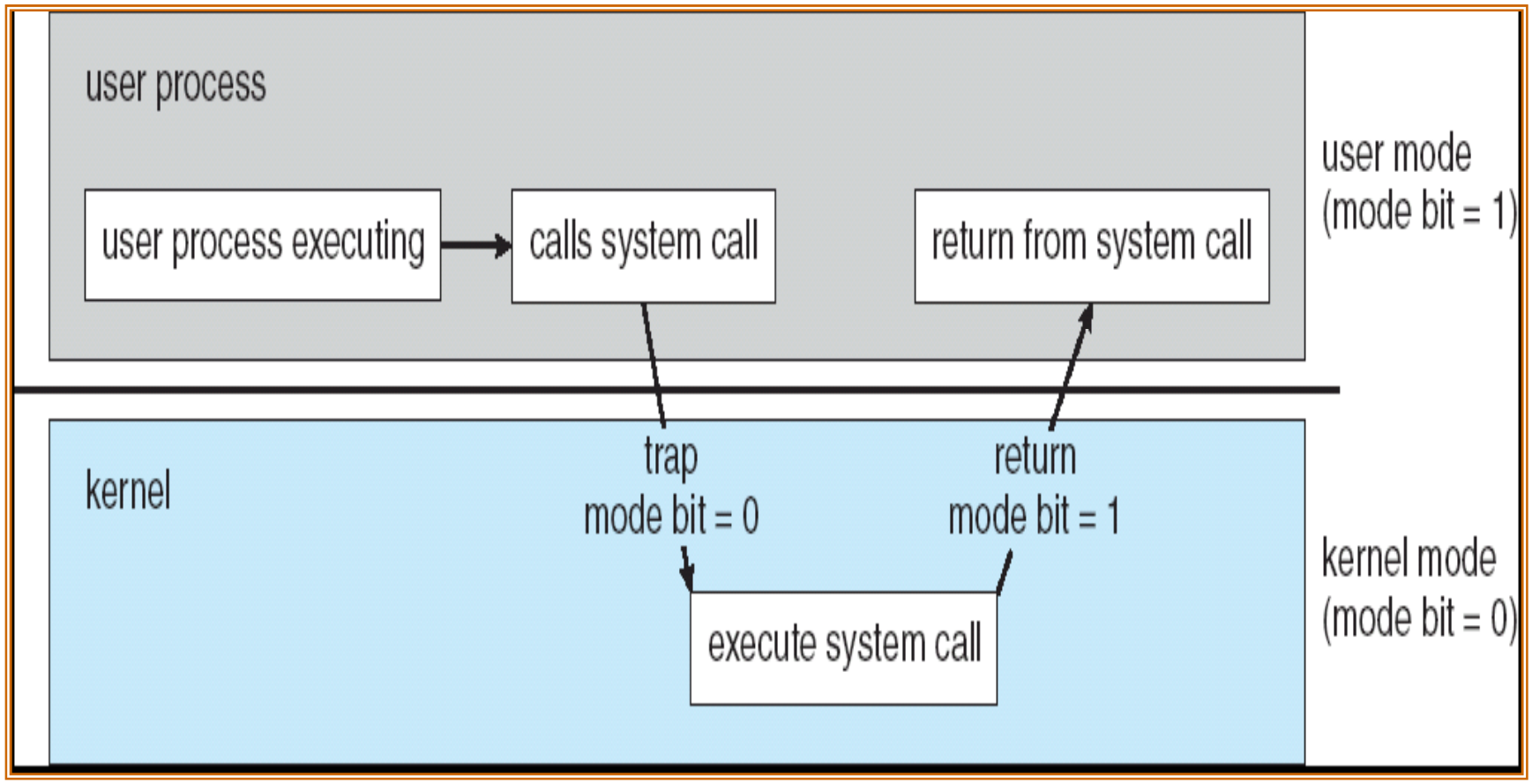
(*) Logical part of RAM in which OS code/data is kept, is called as kernel space. Rest all area in which user programs are executed, is called as user space.


RAM
process
user space
s
h
rd
d
t
program
(disk)
kernel space
PCB

Dual-Mode Operation

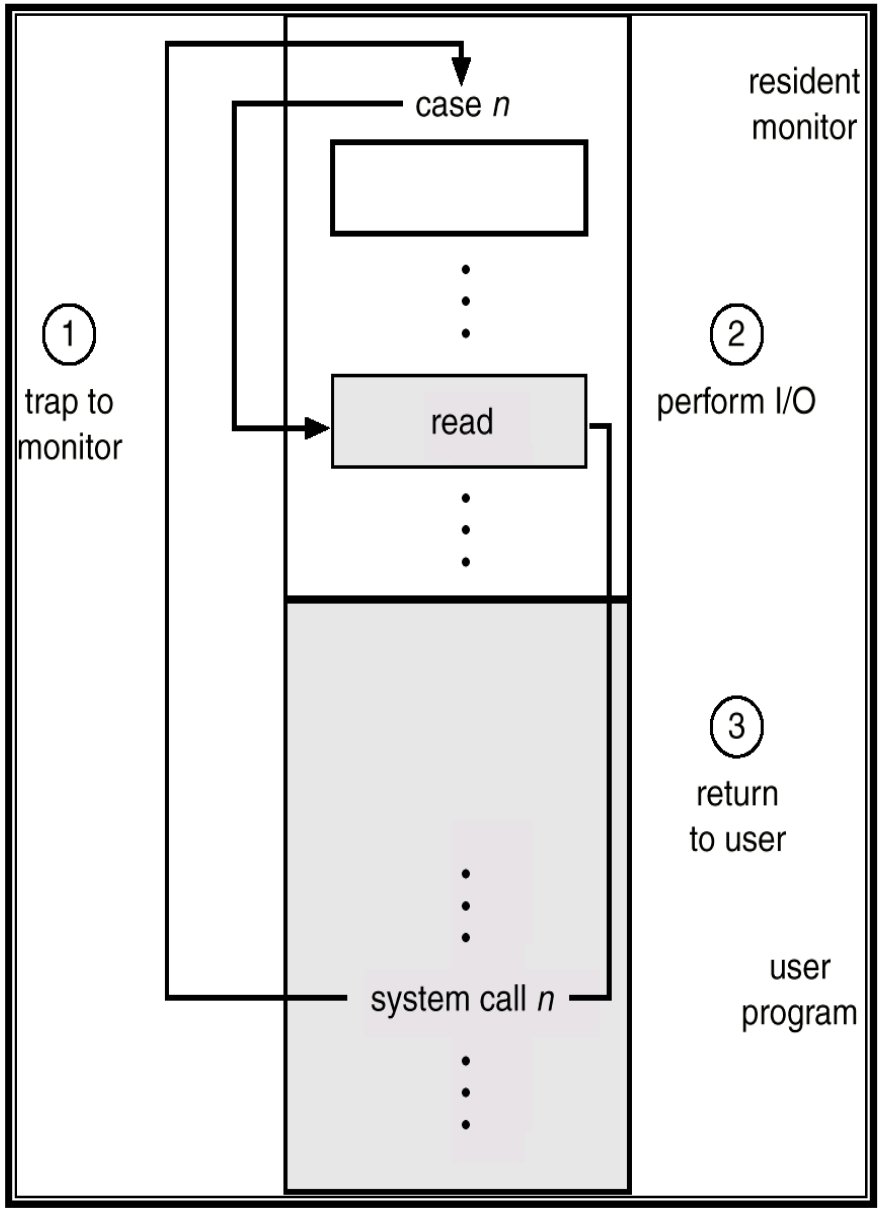
- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
  - User mode – execution done on behalf of a user.
  - Monitor mode (also kernel mode or system mode) – execution done on behalf of operating system.
- Mode bit added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt or fault occurs hardware switches to monitor mode.

User mode and Kernel mode



I/O Protection

- All I/O instructions are privileged instructions.
- Since all I/O operations are done through IVT, it must be protected from user programs.
- I/O must be done via system calls.
- Must ensure that a user program could never gain control of the computer in monitor mode.



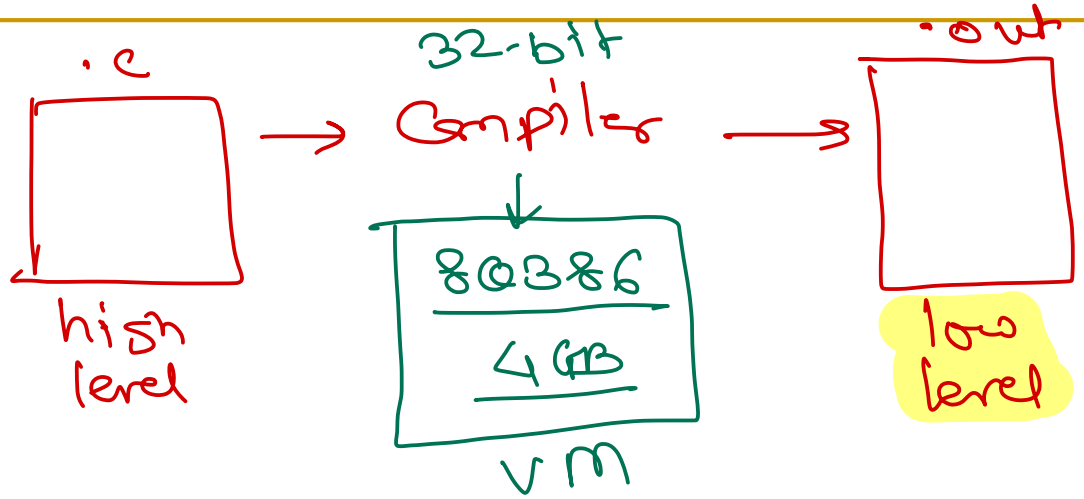
Thank you!

Source: Galvin OS books/slides

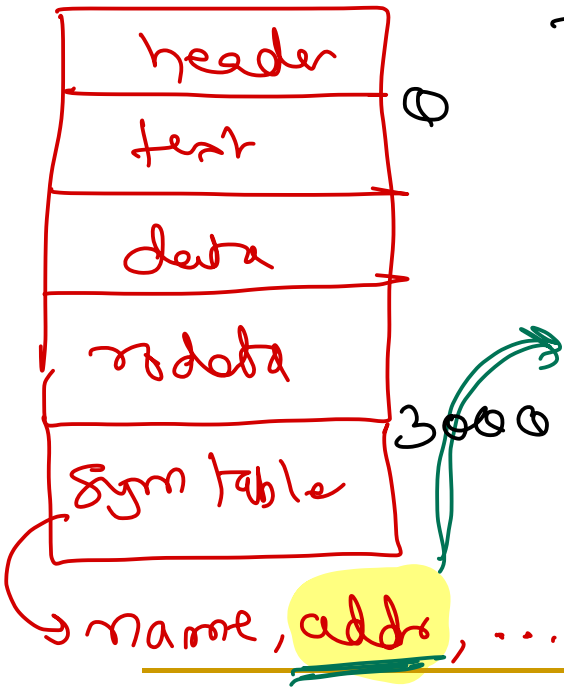
Edited by: Nilesh Ghule

Computer System

Sunbeam Infotech



exe file (disk)

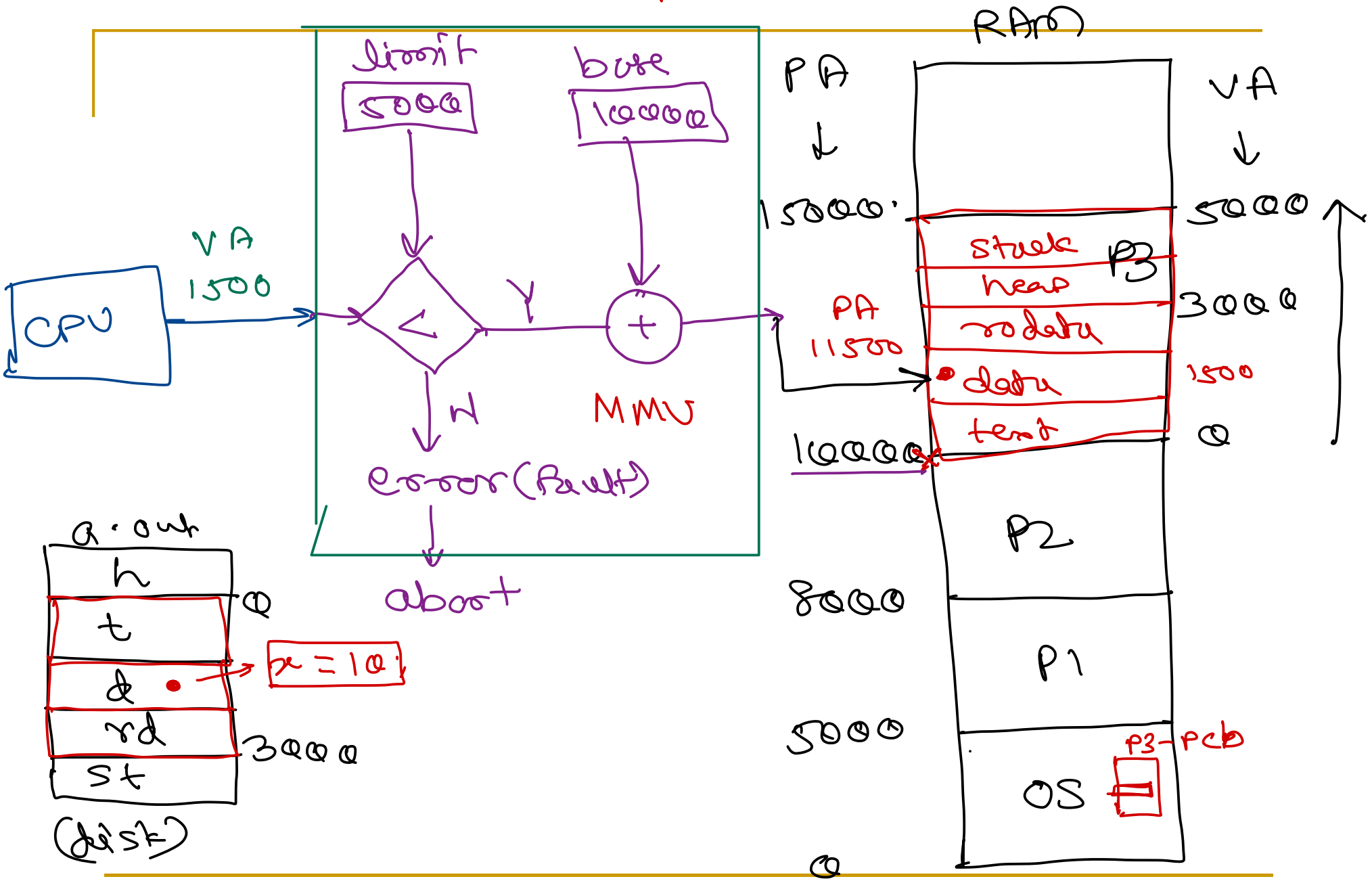


terminal> file exepath → ELF 80386

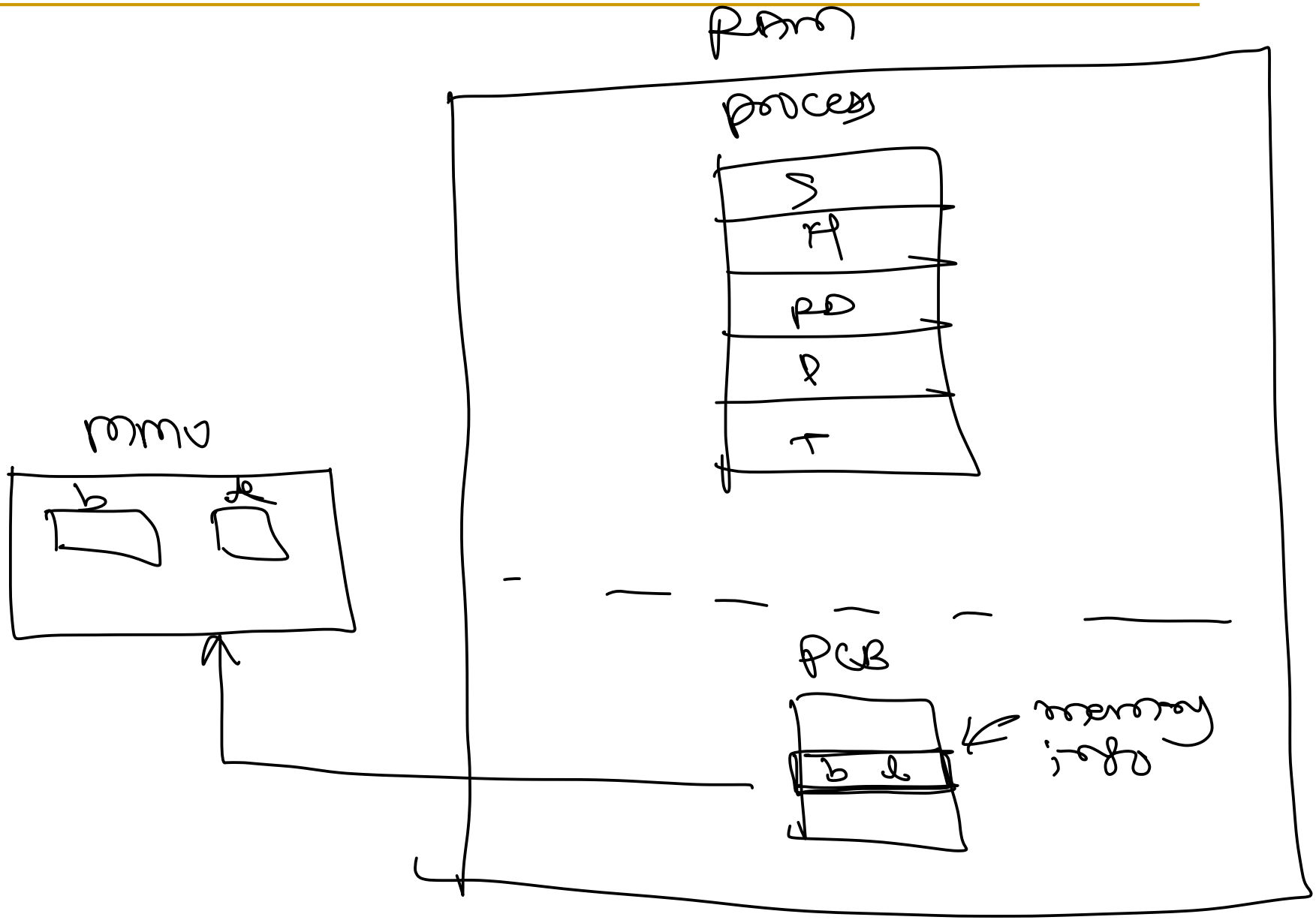
terminal> readelf -a exepath

logical addr / virtual addresses

CPU always execute a process in its virtual address space.



virtual addr space = set of virtual addresses.
physical addr space = set of physical addresses.


RAM
process
S
PD
D
T
MMU
PCB

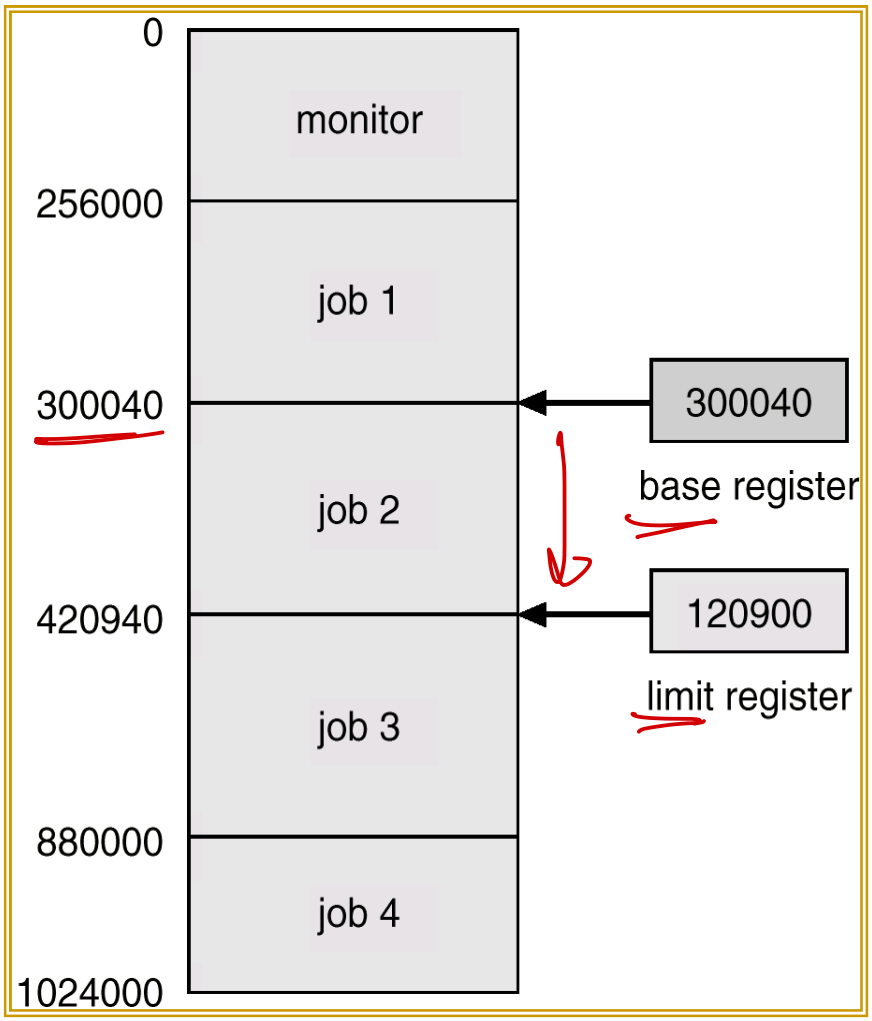
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- A user program must be protected from the other user program.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - Base register – holds the smallest legal physical memory address.
  - Limit register – contains the size of the range
- Memory outside the defined range is protected.

simple mmu

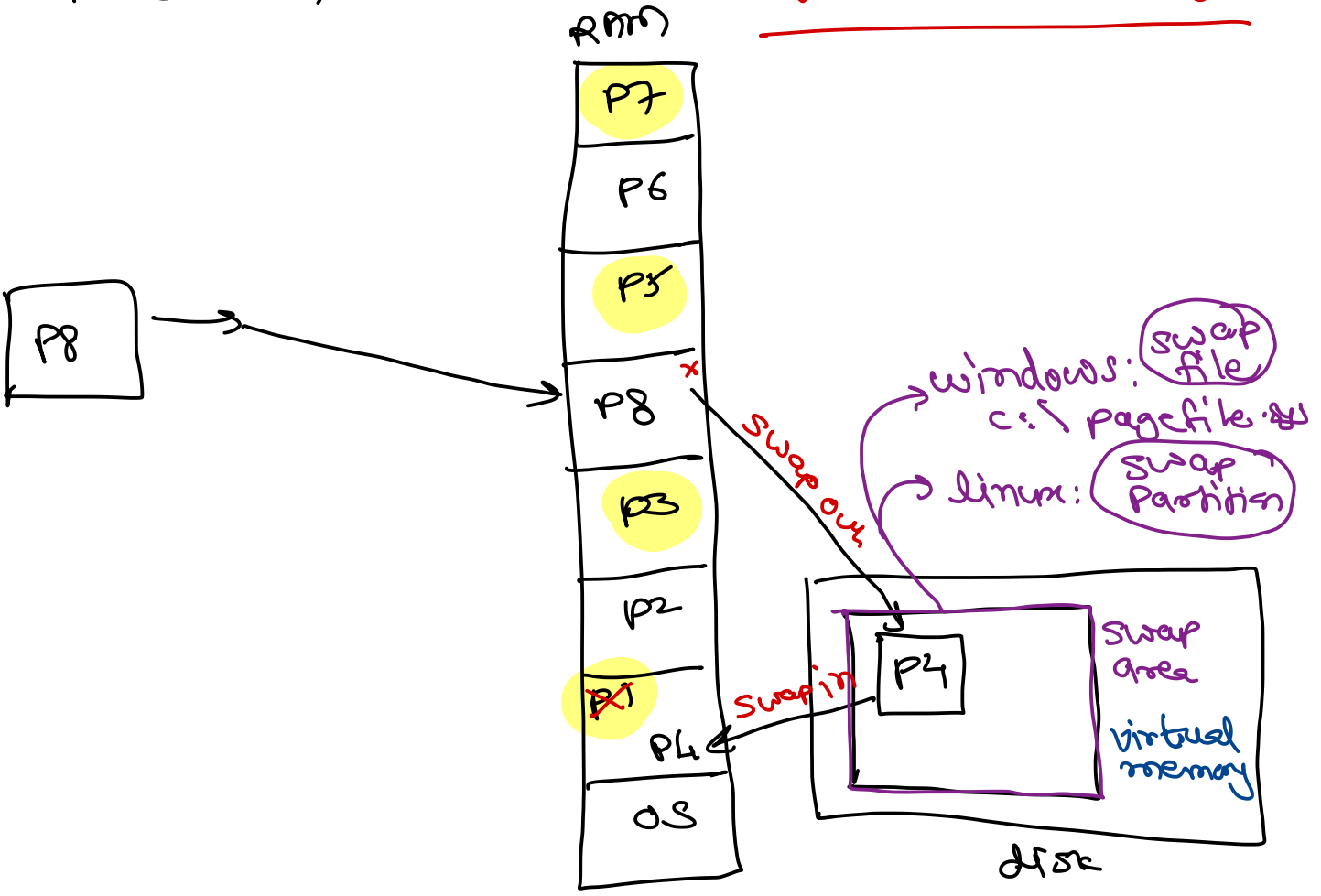
Use of Base and Limit Register

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.
- The load instructions for the base and limit registers are privileged instructions.



terminal> free -m

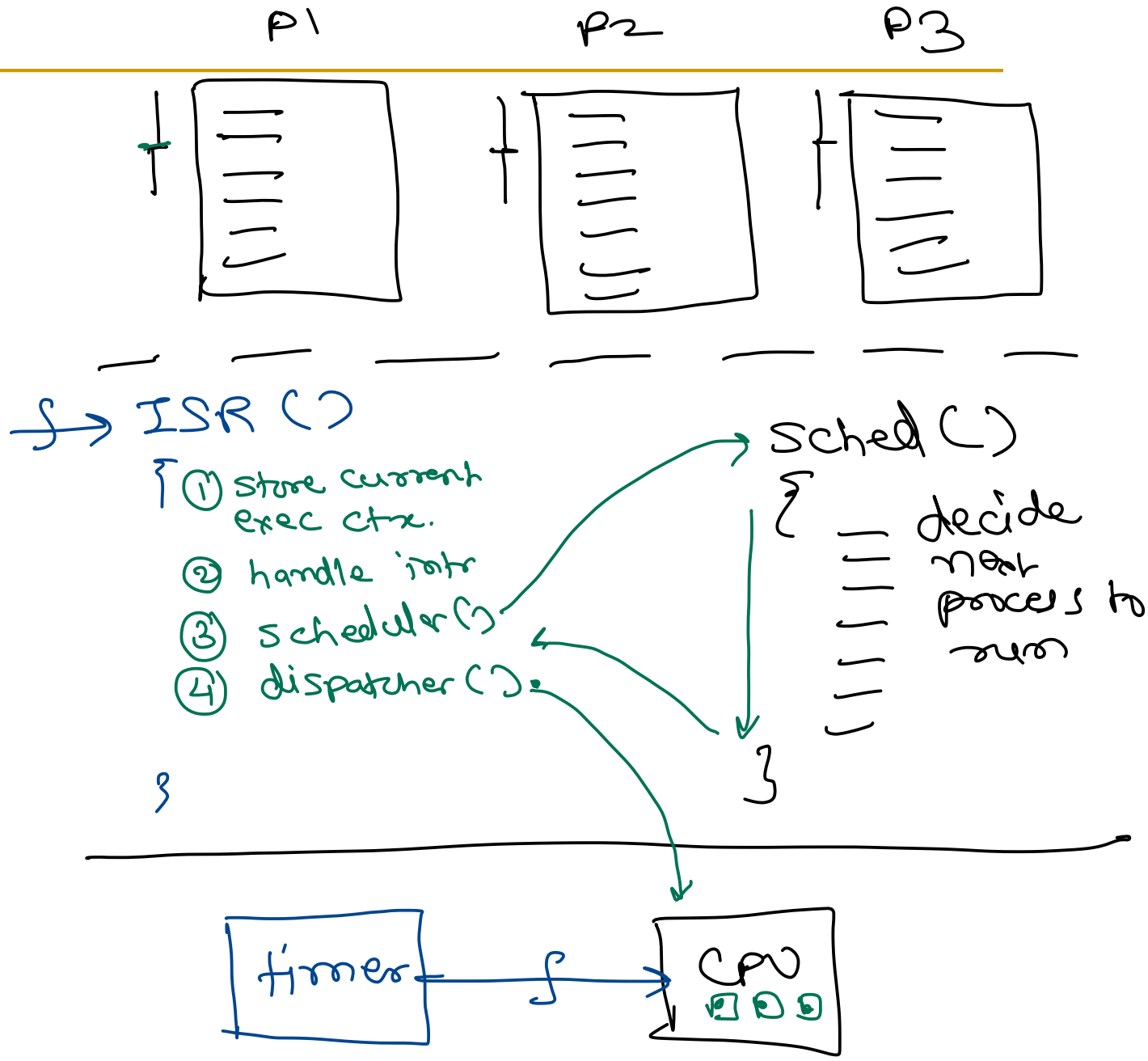
Virtual memory



CPU Protection

- User program may stuck up in infinite loop and may not return control to the operating system.
- Timer interrupts computer after specified period to ensure operating system maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches the value 0, an interrupt occurs.
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time
- Timer is commonly used to implement time sharing.
- Changing timer values is a privileged instruction.
- Computers may have time-of-day clock that is independent of operating system.

win 2.x


P1
P2
P3
ISR()
{
① store current exec ctx.
② handle intr
③ scheduler()
④ dispatcher();
}
sched()
{
decide next process to run
}
timer
CPU

Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule

Operating System

Sunbeam Infotech

Multiprogramming

- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via job scheduling
- When any job has to wait (for I/O for example), OS switches to another job
- Multiprogramming needed to use the system in efficient way.

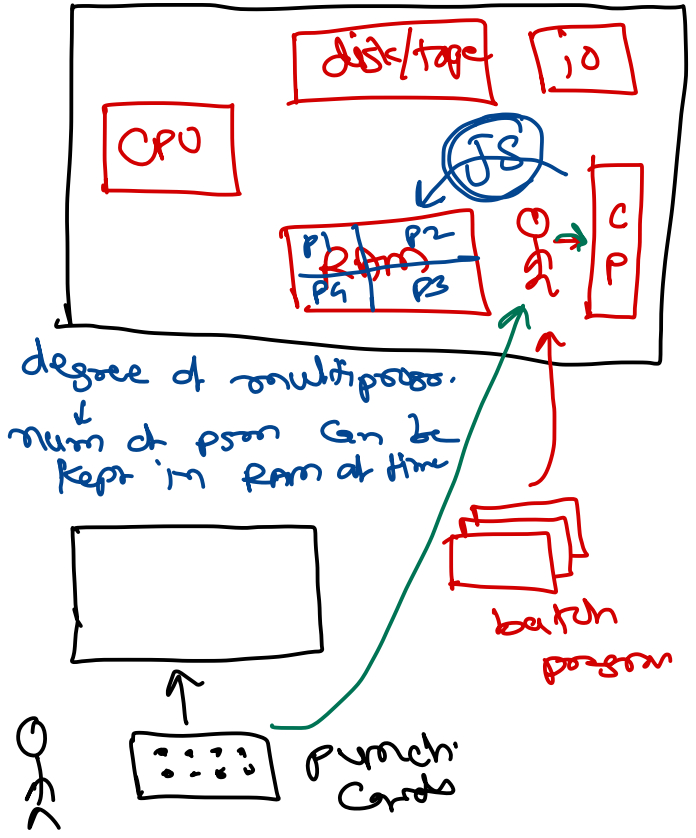
Multitasking

- Multitasking (Time sharing) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running
- Response time should be < 1 second
- If several jobs ready to run at the same time CPU scheduling is necessary to schedule them all.
- Multitasking can be achieved in one of the ways:
  - multi-processing
  - multi-threading
- Virtual memory allows execution of processes not completely in memory. If processes don't fit in memory, they are swapped in and out to run.

Types of operating systems

- Few operating systems are designed to be efficient while other are designed to be convenient.
- Depending on design goals and hardware constraints different types of operating systems exists.
  - Mainframe Systems
  - Desktop Systems
  - Multiprocessor Systems
  - Distributed Systems
  - Real -Time Systems
  - Handheld Systems

Mainframe Computer → PDP-7 & PDP-11 → IBM



① Resident Monitor

② Batch System

③ multiprogramming
- ↳ loading multiple pgms in RAM
- ↳ better cpu utilization
- ↳ job scheduler
  - ↳ loads a mix of cpu bound and io bound programs.

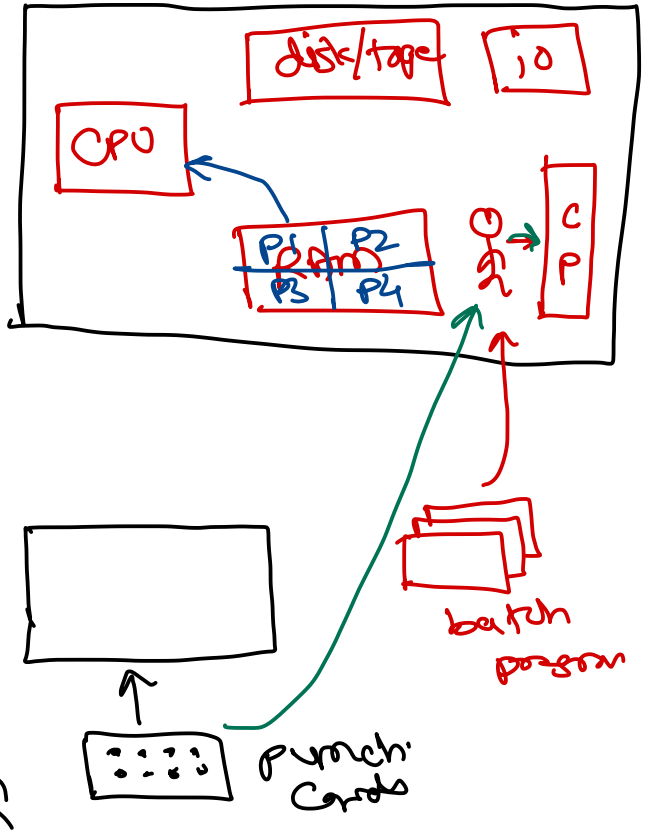
time = cpu burst + io burst

→ gt a&b

→ c = a+b

→ pr c =

Mainframe Computer → PDP-7 & PDP-11 → IBM



① Resident Monitor

② Batch System

③ multi programming

④ Multi-Tasking

↳ time Sharing.

time = CPU burst + IO burst

→ If a&b

→ C = a+b

→ for c =

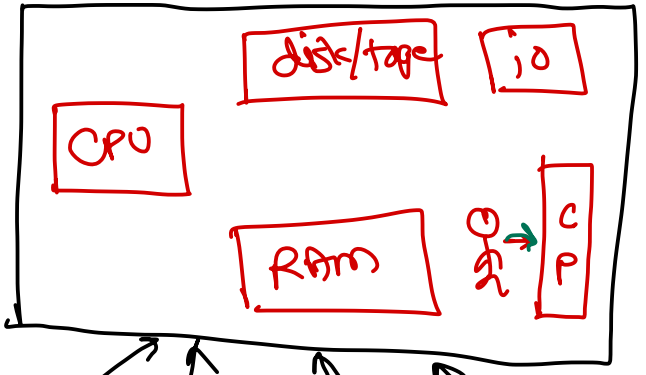
sharing cpu time among multiple tasks present in main memory & ready for execution.

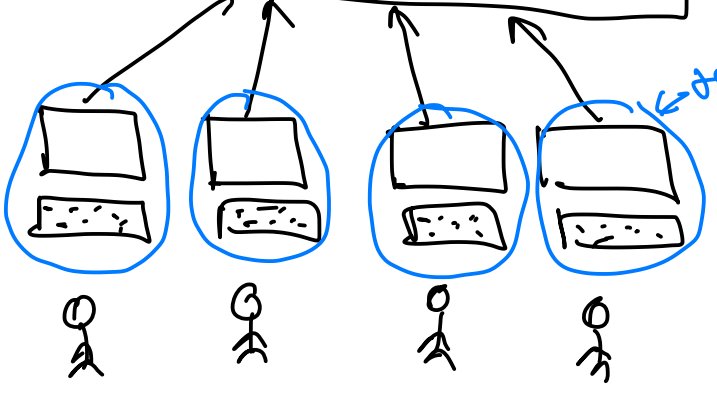
→ process based multitasking. ✓

→ thread based multi tasking ✓

↳ running multiple tasks concurrently in a prog

Mainframe Computer → PDP-7 & PDP-11 → IBM





① Resident Monitor

② Batch System

③ Multi programming

④ Multi-tasking

⑤ Multi-User

time
= CPU burst
+ io burst

→ if a&b
→ c=a+b
→ pf c

In modern system → terminal ← input → output

↓

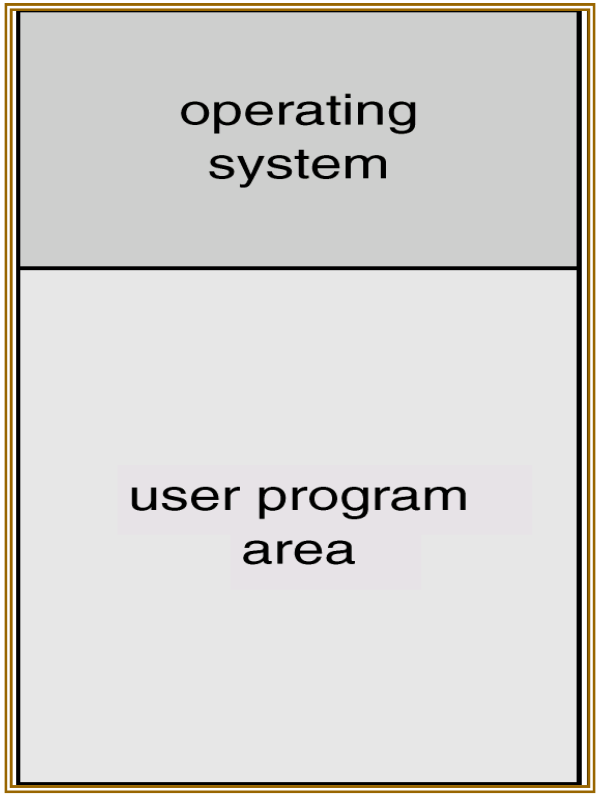
tty

Mainframe Systems

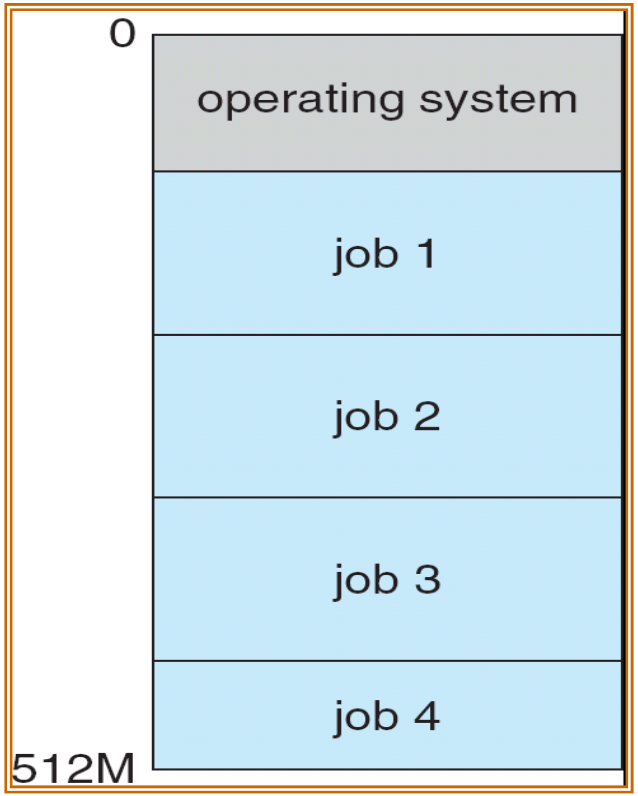
- Batch systems
  - Reduce setup time by batching similar jobs
  - Transfers control from one job to another.
- Resident monitor
  - initial control in monitor
  - control transfers to job
  - when job completes control transfers back to monitor
- Multiprogrammed systems
  - Multiple jobs in memory at a time
- Time sharing systems
  - Executes multiple jobs using time sharing concept

Mainframe Systems

e.g. UNIX, IBM360, ...



- Simple Batch System

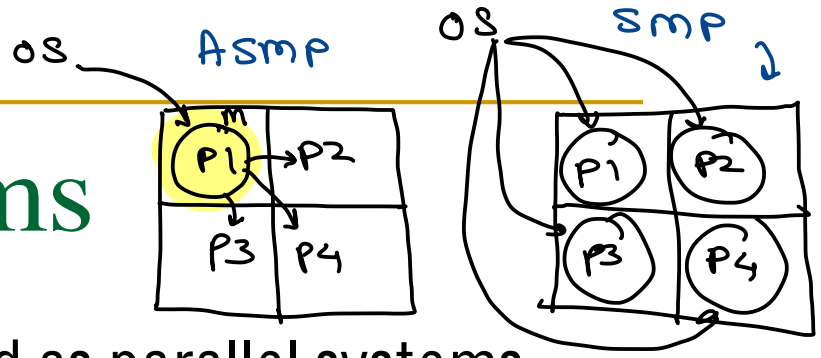


- Multiprogramming System

Desktop Systems

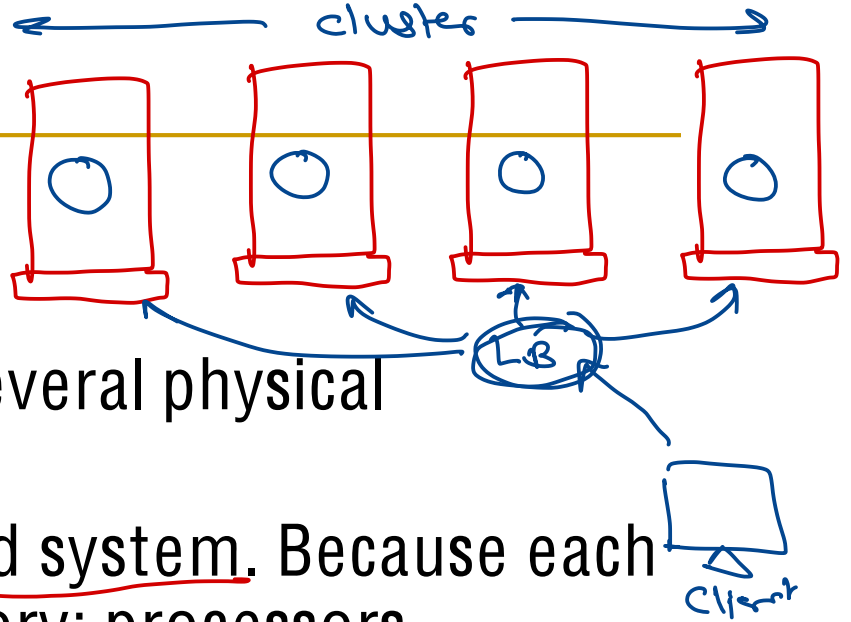
- Personal computers – computer system dedicated to a single user.
- I/O devices – keyboards, mice, display screens, small printers.
- User convenience and responsiveness.
- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

Multiprocessor Systems



- Multiprocessor systems are also called as parallel systems.
- Multiprocessor systems with more than one CPU in close communication.
- Advantages of parallel system: → increased computation ability
  - Increased throughput, Economical and Increased reliability
- Symmetric multiprocessing (SMP)
  - Each processor runs and identical copy of the operating system.
  - Most modern operating systems support SMP
- Asymmetric multiprocessing
  - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
  - More common in extremely large systems

Distributed Systems

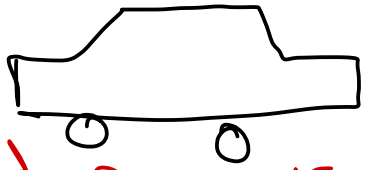


- Distribute computation among several physical processors.
- It is also called as Loosely coupled system. Because each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
- Advantages of distributed systems.
  - Resources Sharing, Load balancing, Reliability
- Requires networking infrastructure.
- Local area networks (LAN) or Wide area networks (WAN)
- May be either client-server or peer-to-peer systems.

1. scalability
2. availability
3. fault tolerance

accuracy = correctness + time

Real -Time Systems



minimal interrupt latency, priority based execution.

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
- Well-defined fixed-time constraints.
  - missing deadline - catastrophic
- These systems may be either hard or soft real-time.
  - media player.
- Hard real-time
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
- Soft real-time → consumer electronics
  - More flexible and hence widely used.

RTS

MS

RTOS, pSOS, Symbian

FreeRTOS, uC/OS, VRTX, Vxworks, MiTron, WinCE,

Handheld Systems

- These systems are used for mobile hardware.
  - Personal Digital Assistants (PDAs)
  - Cellular phones
  - Portable multimedia systems
- Issues:
  - Limited memory
  - Slow processors
  - Small display screens

- Linux
- Android
- iOS
- Symbian
- window phone

Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule

Operating System

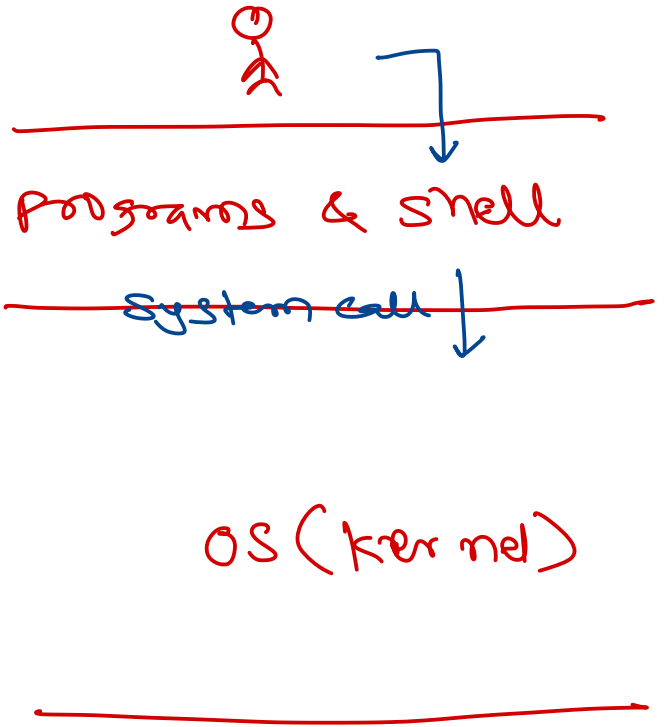
Sunbeam Infotech

What Operating System does?

- There are different types of operating systems, however all these systems have some core services
  - Process Management & cpu scheduling
  - Memory Management
  - Storage Management
  - File System Management
  - I/O System Management & hardware abstraction
  - Protection and Security → anti-virus
  - User interfacing → shell
  - Networking

kernel

extra



shell → Cmnd interpreter.

↳ GUI & CLI

GUI → Windows: explorer.exe

Linux: GNOME or KDE

CLI → windows: cmd.exe

DOS: command.com

Linux: bash, csh, ksh, ...

terminal> echo $SHELL

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is passive, process is active.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Reusable resources are released as process terminates
- Process management activities done by OS:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication

Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a quickly accessible by the CPU and I/O devices.
- All data is in memory before and after processing.
- All instructions are in memory for execution.
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - WORM (write-once, read-many-times) and RW (read-write)

File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- Files usually organized into directories
- Access control on most systems to determine who can access what
- OS activities include
  - Creating and deleting files and directories
  - Primitives to manipulate files and directories
  - Mapping files onto secondary storage
  - Backup files onto stable (non-volatile) storage media

I/O Management

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

Protection and Security

- Protection is mechanism for controlling access of processes or users to resources defined by the OS.
  - Systems generally first distinguish among users, to determine who can do what
    - User identities (user IDs, security IDs) include name and associated number, one per user
    - User ID then associated with all files, processes of that user to determine access control
    - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
    - Privilege escalation allows user to change to effective ID with more rights
- Security is defense of system against internal/external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

Operating System Interface - CLI

- User communicates with the core OS graphical or command line interface.
- CLI allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
  - Primarily fetches a command from user and execute it
    - Sometimes commands built-in, sometimes just names of programs
    - If the latter, adding new features doesn't require shell modification

Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI (Java desktop, KDE)

Operating System Design

Sunbeam Infotech

OS Design and Implementation

- Design and Implementation of OS not "solvable", but some approaches have proven successful
- Internals of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
  - User goals and System goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

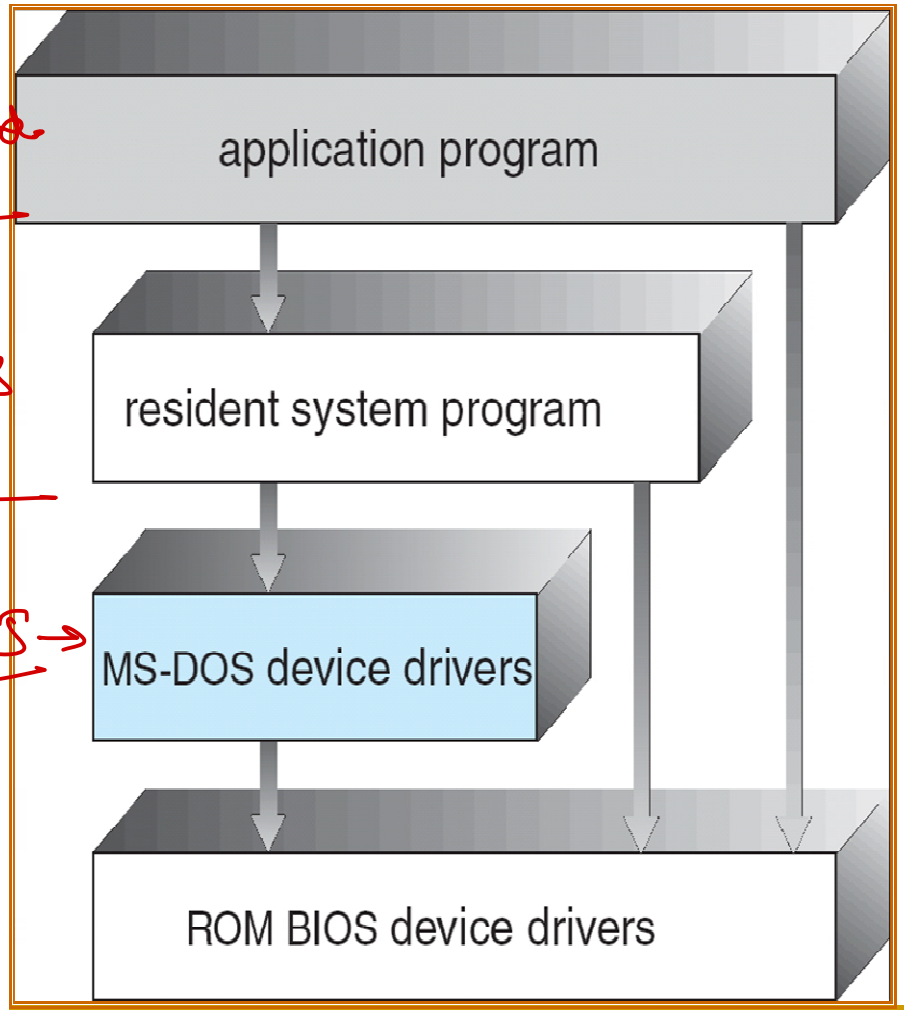
OS Design and Implementation

MS: User Convenience

UNIX: efficiency

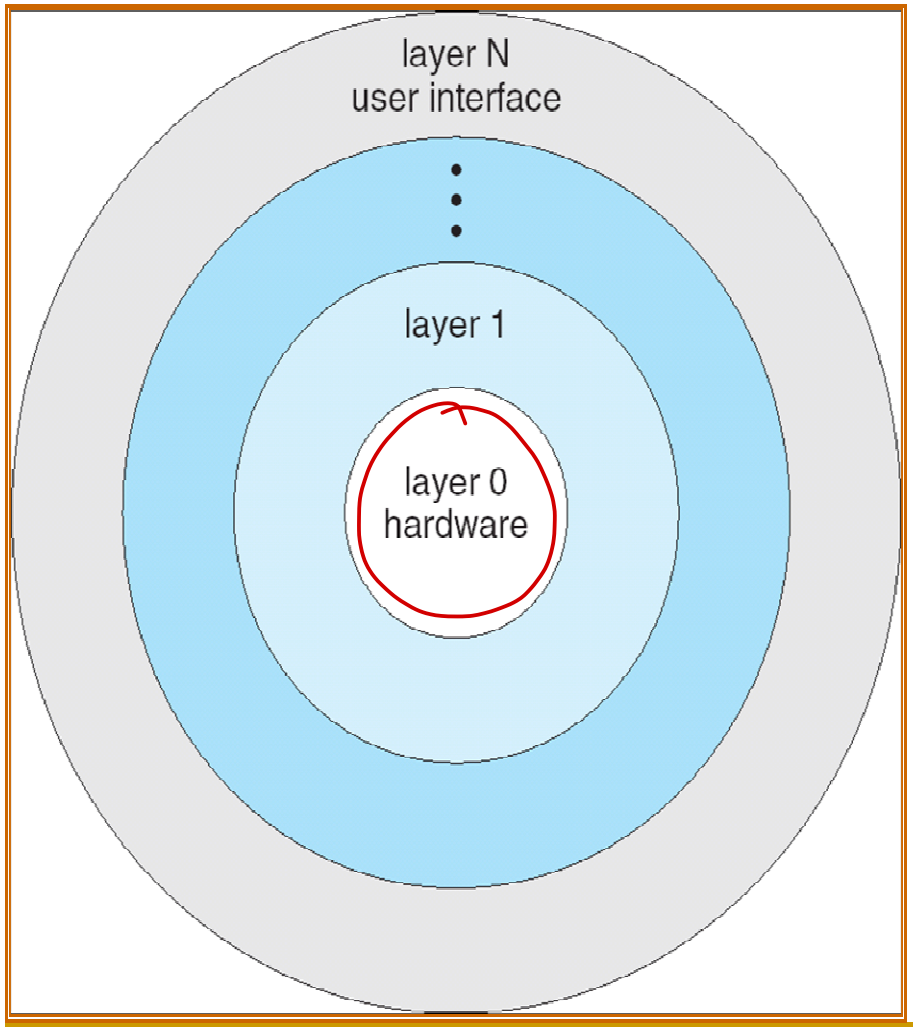
- Important principle to separate
  - Policy: What will be done?
  - Mechanism: How to do it?
- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
- There are different examples of operating systems depending on the goals of designs and the policies and mechanisms.

Simple Structure : MS-DOS



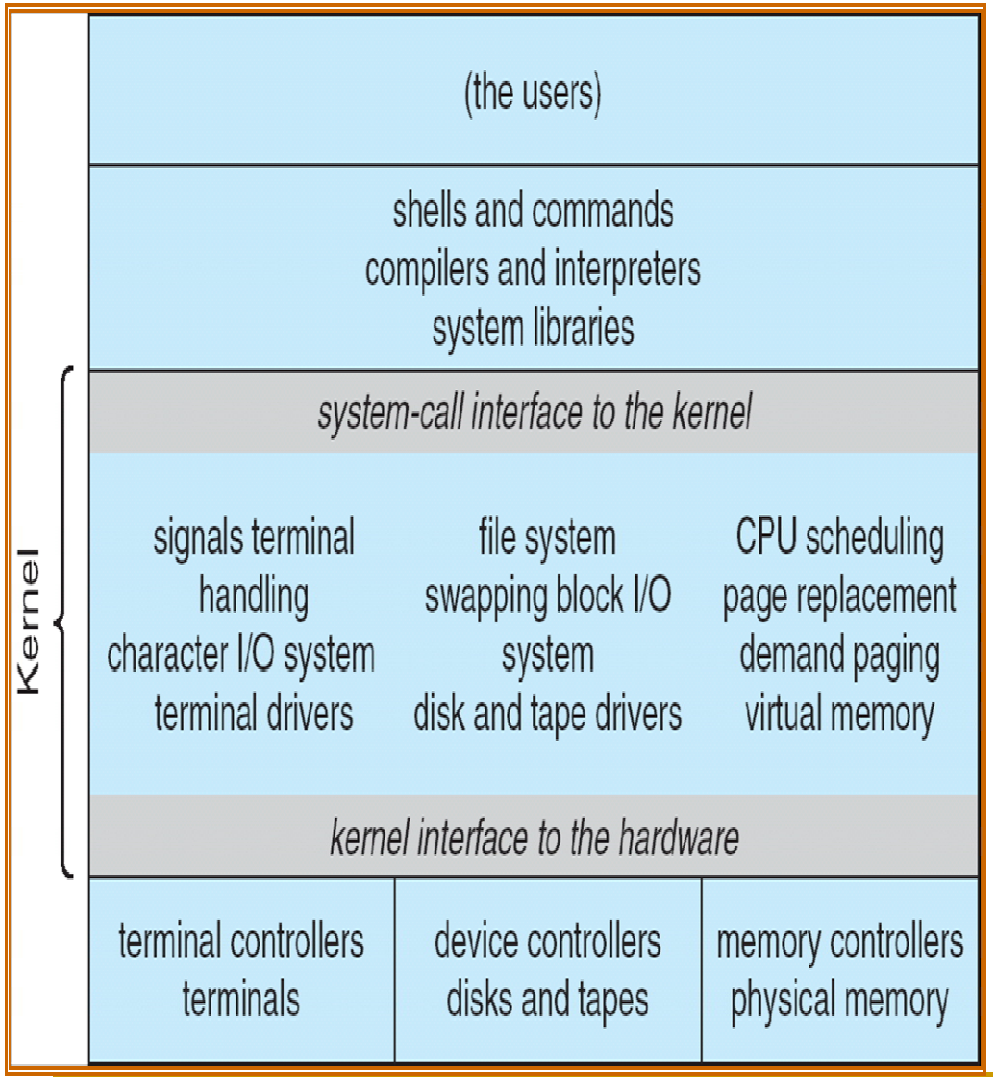
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

Layered Structure



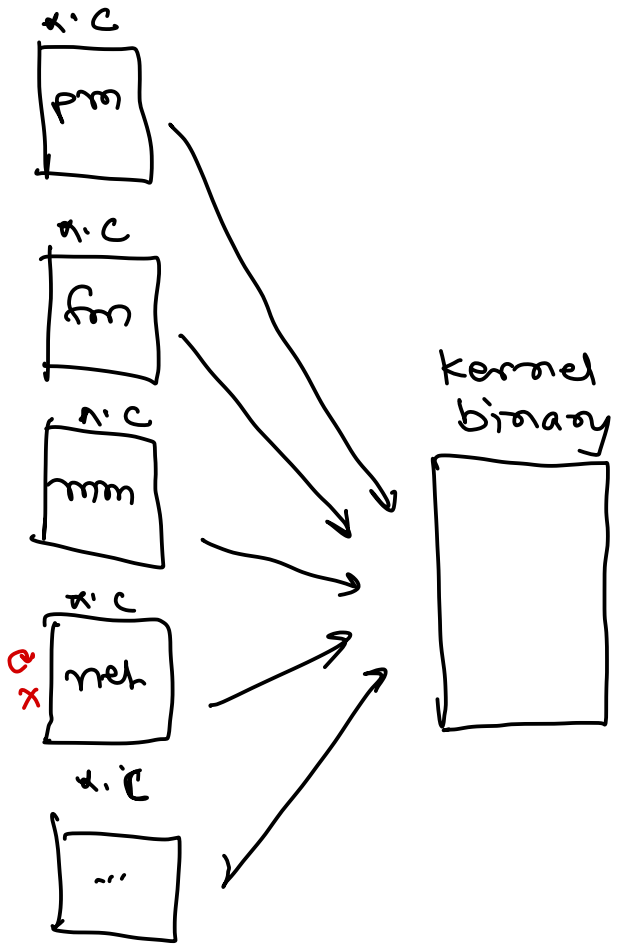
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

UNIX System



- UNIX is limited by hardware functionality, the original UNIX had limited structuring.
- UNIX consists two separate parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Monolithic



✓ ① fast (all modules are in same binary).

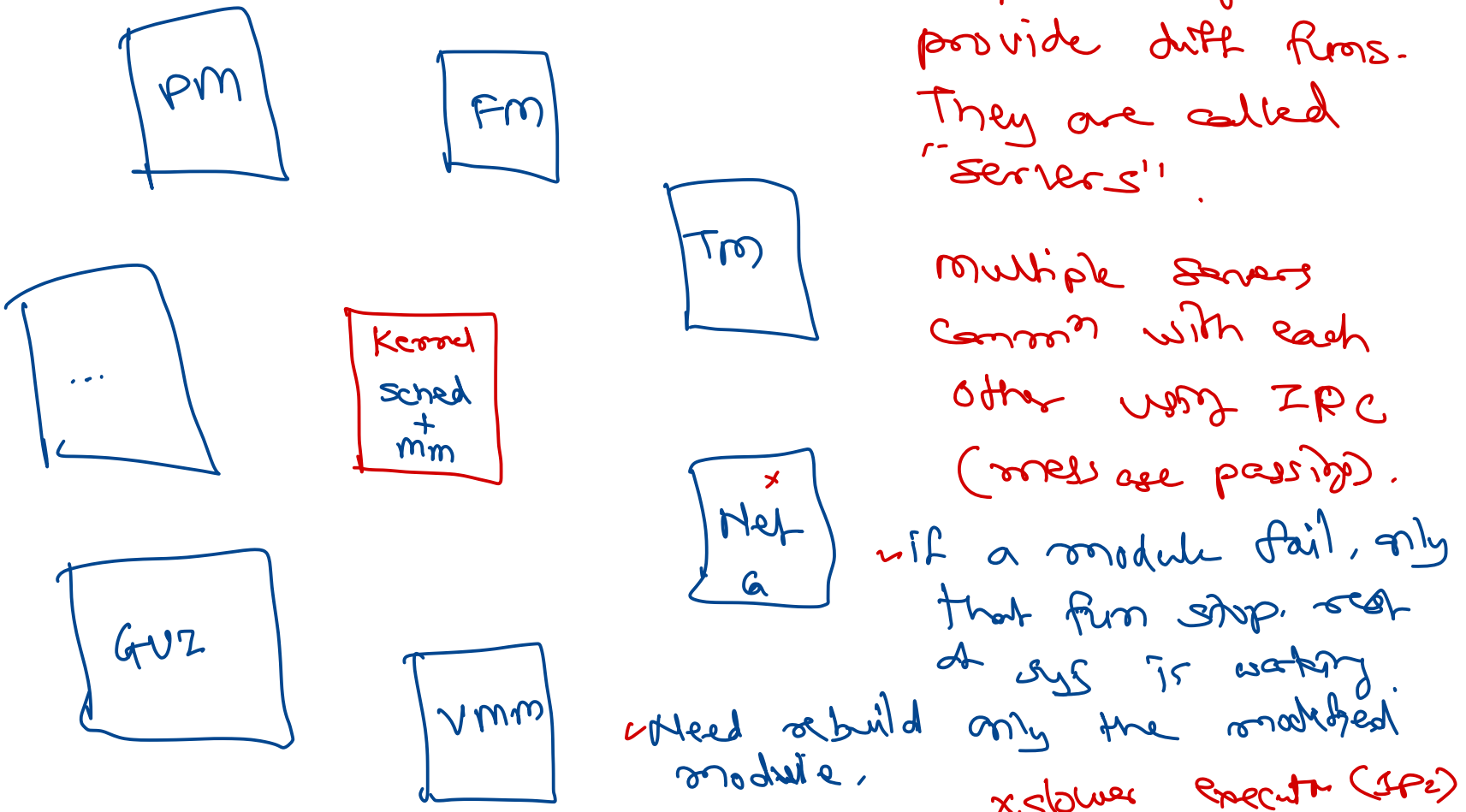
✗ ② if any module fails, kernel crash.

✗ ③ any change in any src code, force to re build & re deploy whole kernel.

DOS, UNIX, Linux(?)

↳ /boot/vmlinuz

Micro-kernel

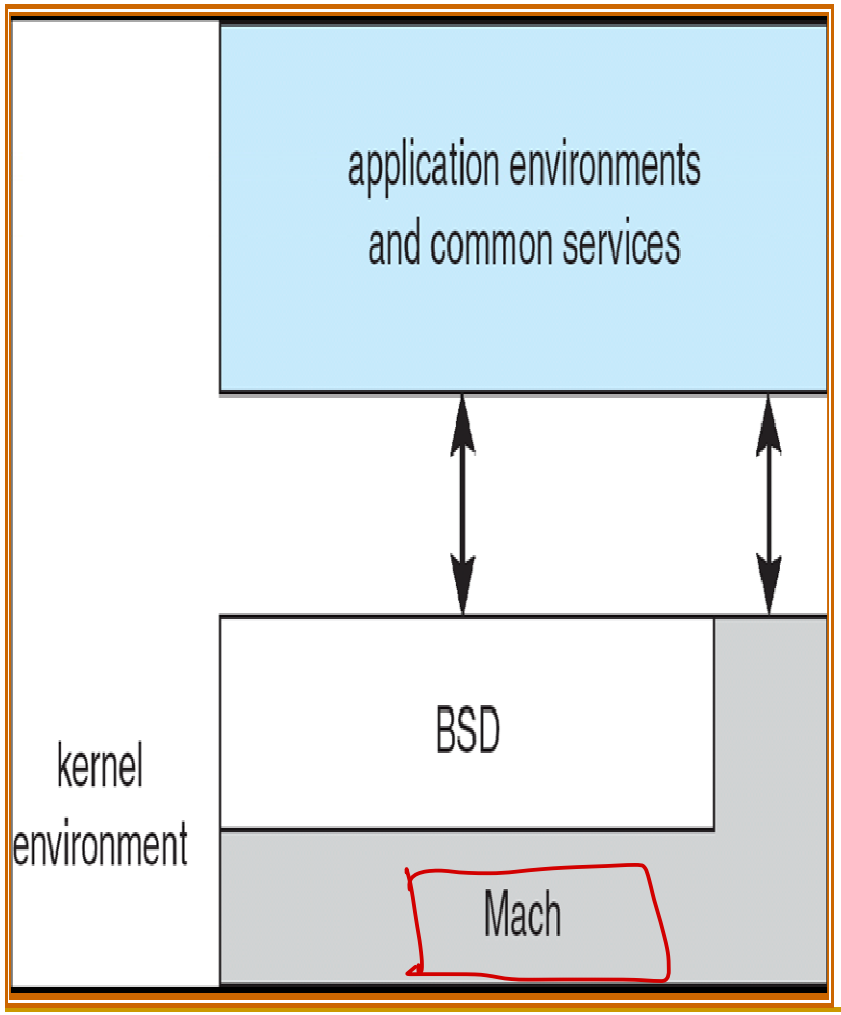


'independent processes provide diff fcns. They are called "servers".

Multiple servers communicate with each other using IPC (msgs are passing).

- if a module fail, only that fcn stop. eg. A sys is waiting.
- Need rebuild only the modified module.
- slower execution (IPC)

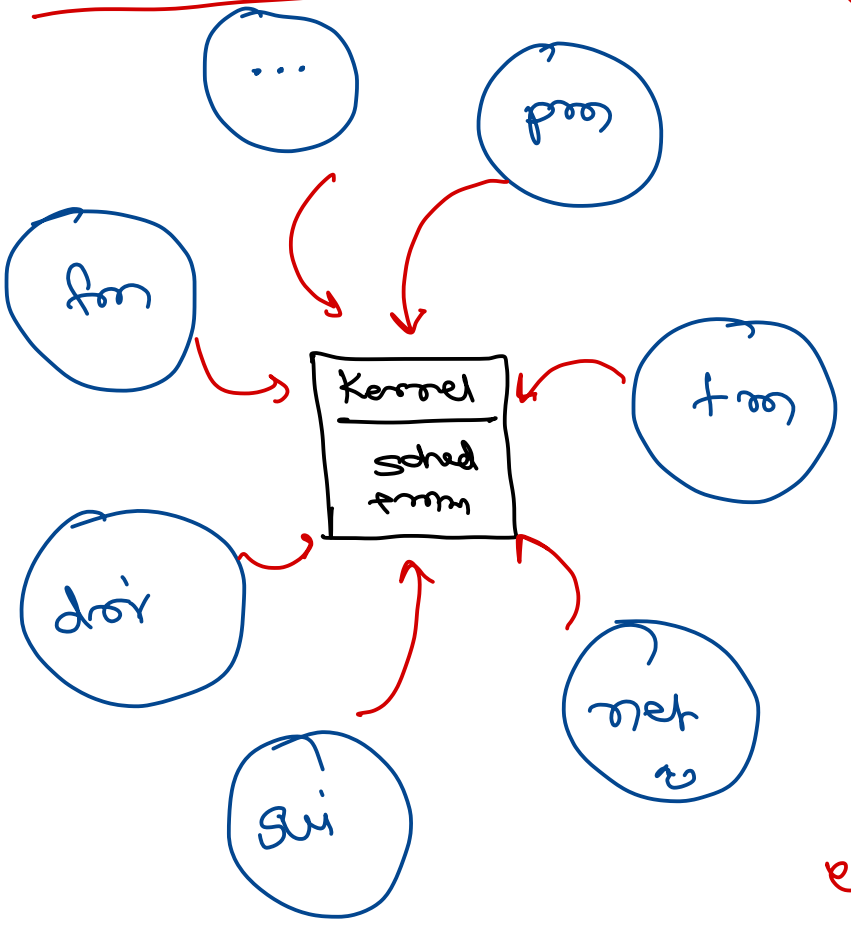
Microkernel Structure: MAC OS X



- Moves as much from the kernel into "user" space
- Communication takes place between user modules using message passing
  - Easier to extend a microkernel
  - Easier to port on new architecture
  - More reliable (less code running in kernel mode)
  - More secure
  - Performance overhead of user space to kernel space comm^{n}

Symbian, QNX,

Modular Kernel



(win) .dll or (linux) .so

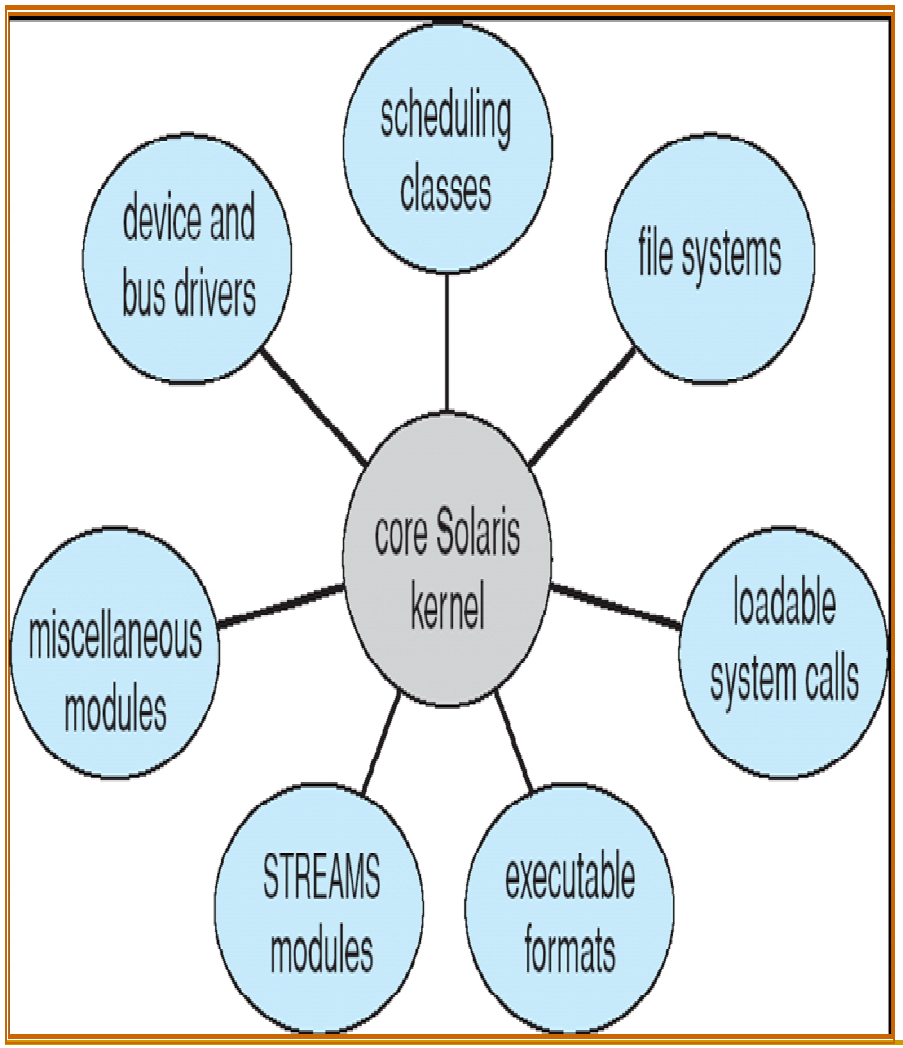
+ fast (all modules are loaded in same process)

+ any change in component need to rebuild that component only.

x if module fails, kernel crash.

e.g. windows: kernel = ntoskrnl.exe

Modular Structure: Solaris



- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Hybrid Kernel → made up of multiple kernels.

Mac OS X = BSD UNIX + Mach

BSD UNIX → Monolithic

Mach → Microkernel

1. monolithic
2. microkernel
3. modular
4. hybrid

Linux Kernel

1. **Static Components**
   - Process, Thread, Scheduler
   - Syscalls, Memory, ...

   } vmlinuz ↑ Kernel

2. **Dynamic Components**
   - Device Drivers
   - file systems

   } .ko → modules

UNIX

1965: AT&T, MIT, GE
↓
MULTICS

Dennis Ritchie
+ Ken Thompson → UCB → BSD UNIX

1968: Project stopped

1969: AT&T: office editorial system

1970: UNIX → PDP-7
PL/1 + Assembly

1971: C programming lang.

1972: UNIX – PDP-11
C + assembly

Bill Joy
1. vi editor
2. c shell
3. n-curses lib
4. virtual memory
5. demand paging
6. TCP protocol
7. Socket

UNIX flavours

AT&T → UNIX

USB → BSD UNIX

Sun → Solaris

HP → HP-UX

DG → DG-UX

SCO → SCO UNIX

SGI → IRIX (OpenGL)

MS → Xenix (→ PC)

Apple → Mac OS X

IBM → AIX

Comer → XINU

Tenanbaum → minix

Linus Torvalds → Linux

Richard Stallman → OSF

www.kernel.org

Linux Distributions

www.kernel.org

Redhat → Kernel + GUI + BL + Utils + apps → RHEL / Fedora → CentOS

Novell → SuSE

Debian → Ubuntu → Mint

Slackware

Knoppix

Backtrack

www.distrowatch.com

Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule

System Booting

Sunbeam Infotech

Bootable device

- The procedure of starting a computer system by loading the kernel is known as booting the system.
- The first sector of any storage device is known as boot sector.
- If boot sector of any storage device contains a special program called bootstrap program, then that device is said to be a bootable device.
- Similarly if boot sector of the partition of any disk contains this program, partition is said to be a boot partition.

bootstrap program

- Bootstrap program locates the kernel, loads it into main memory and starts its execution.
- Sometimes two-step process where boot block at fixed location loads complex bootstrap program.
- There is a separate bootstrap program for each operating system.
- If multiple operating systems are installed on the computer, the boot loader encompasses the bootstrap/boot loader for other operating systems.
- NTLDR, GRUB, LILO, etc are few examples of boot loaders.

Boot loaders

windows (before vista): ntldr

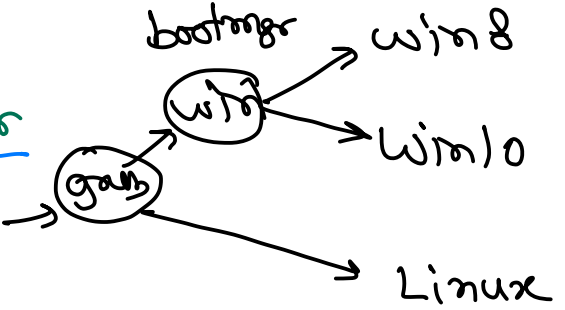
windows (vista+): bootmgr

Linux: LiLo, GrUB

BSD UNIX: BTX

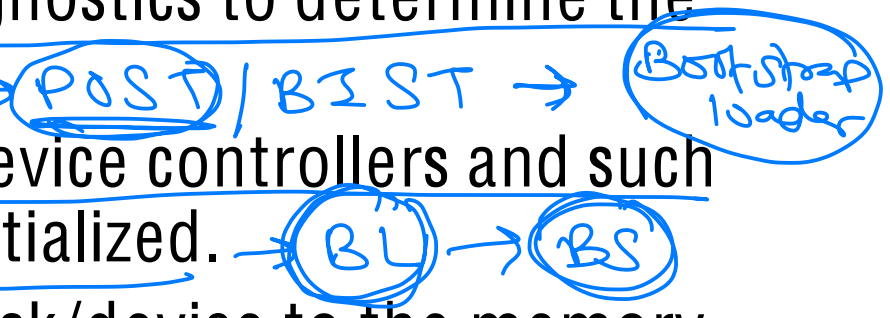
Solaris: SILo

Mac OSX: Boot Camp (Darwin)



Booting process

- When computer is started, the instruction pointer (program counter) is loaded with a predefined memory location and execution starts there.
- The initial boot program is kept here in ROM, as RAM is in an unknown state at system startup.
- The bootstrap program run diagnostics to determine the state of machine (hardware).
- It also initialize CPU registers, device controllers and such things so that basic system is initialized.
- It loads kernel of OS from the disk/device to the memory and starts the first process of that kernel.
- Finally one or more system processes are created which functions as an operating system.



Power on

↓

BaseRom programs

↓

RAM

firmware

BIOS → Set of programs

Base Rom (motherboard)

① POST ② Bootstrap loader - to find bootable device.

Bootloader - to show options of OS to load.

/ user select

Bootstrap ⟶ Load the kernel.

hardware → PCB, wires, electronic ccts, ...

firmware → programs fixed/burned in ROM.
Cannot easily add/remove.

PC → IBM + MS → firmware = BIOS

Intel + HP → firmware = EFI

Software → programs installed in OS.
they can be easily added/removed.

Virtual Machine

OS virtualization

applns of virtualization

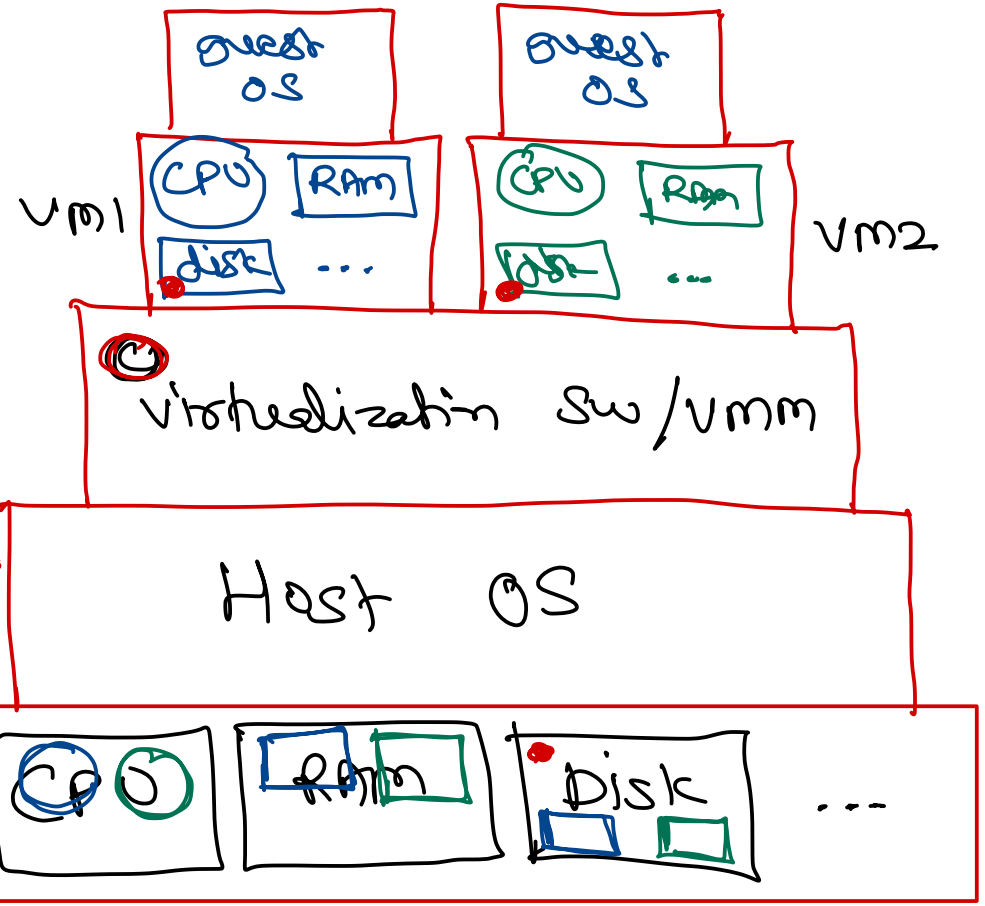
1. testing / dev sw on multiple platform.
2. learning networking.

limitations

1. no CPU intensive task.

virtualization sw

1. vmware
2. Virtual box
3. Hyper-V
4. Parallels



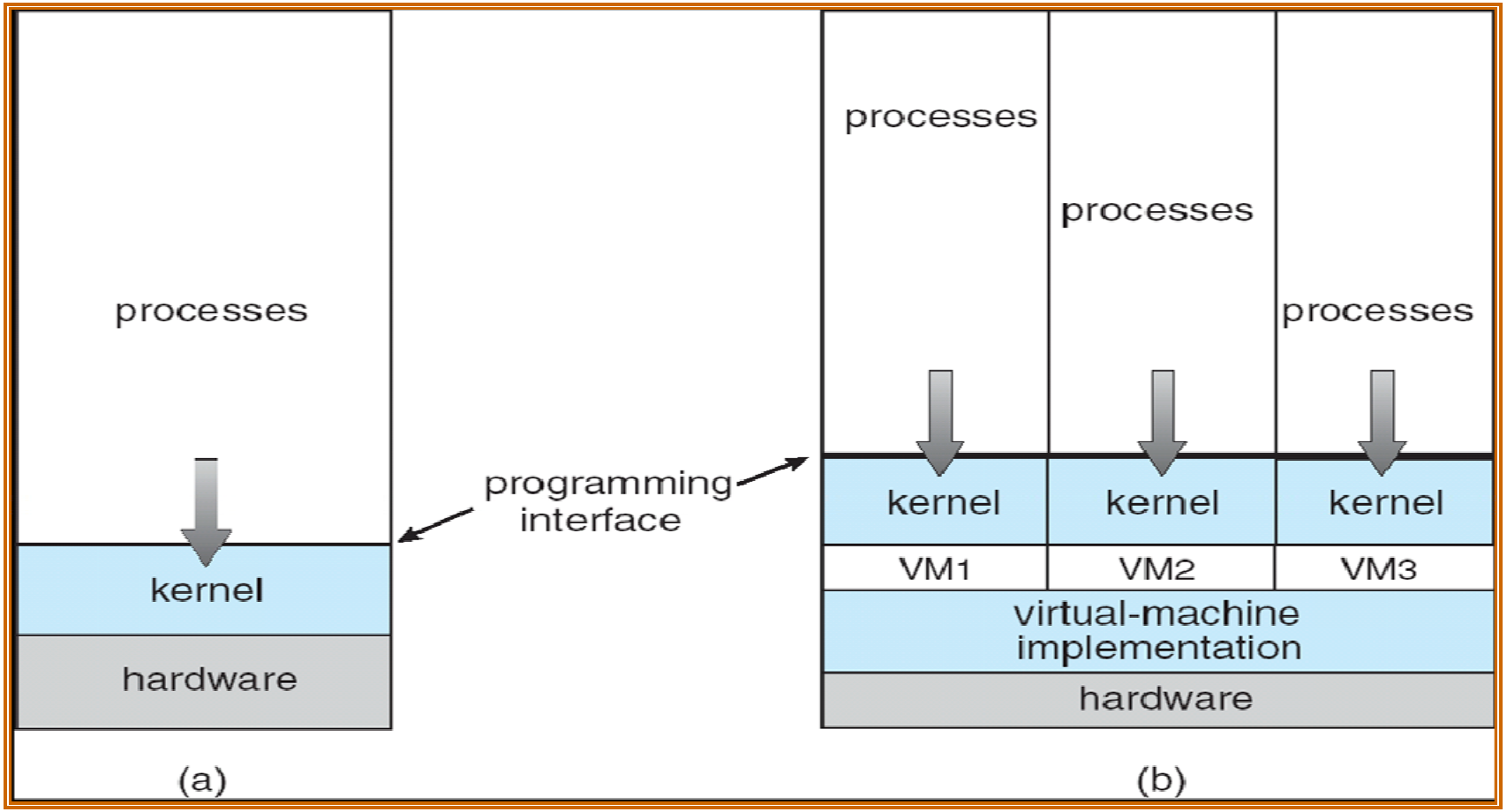
lscpu

* emulator : qemu

Virtual Machines

- Virtual machine takes layered approach for logical conclusion
- It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface identical to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Spooling and a file system can provide virtual card readers and virtual line printers
  - A normal user time-sharing terminal serves as the virtual machine operator's console

Virtual Machine (cont'd)

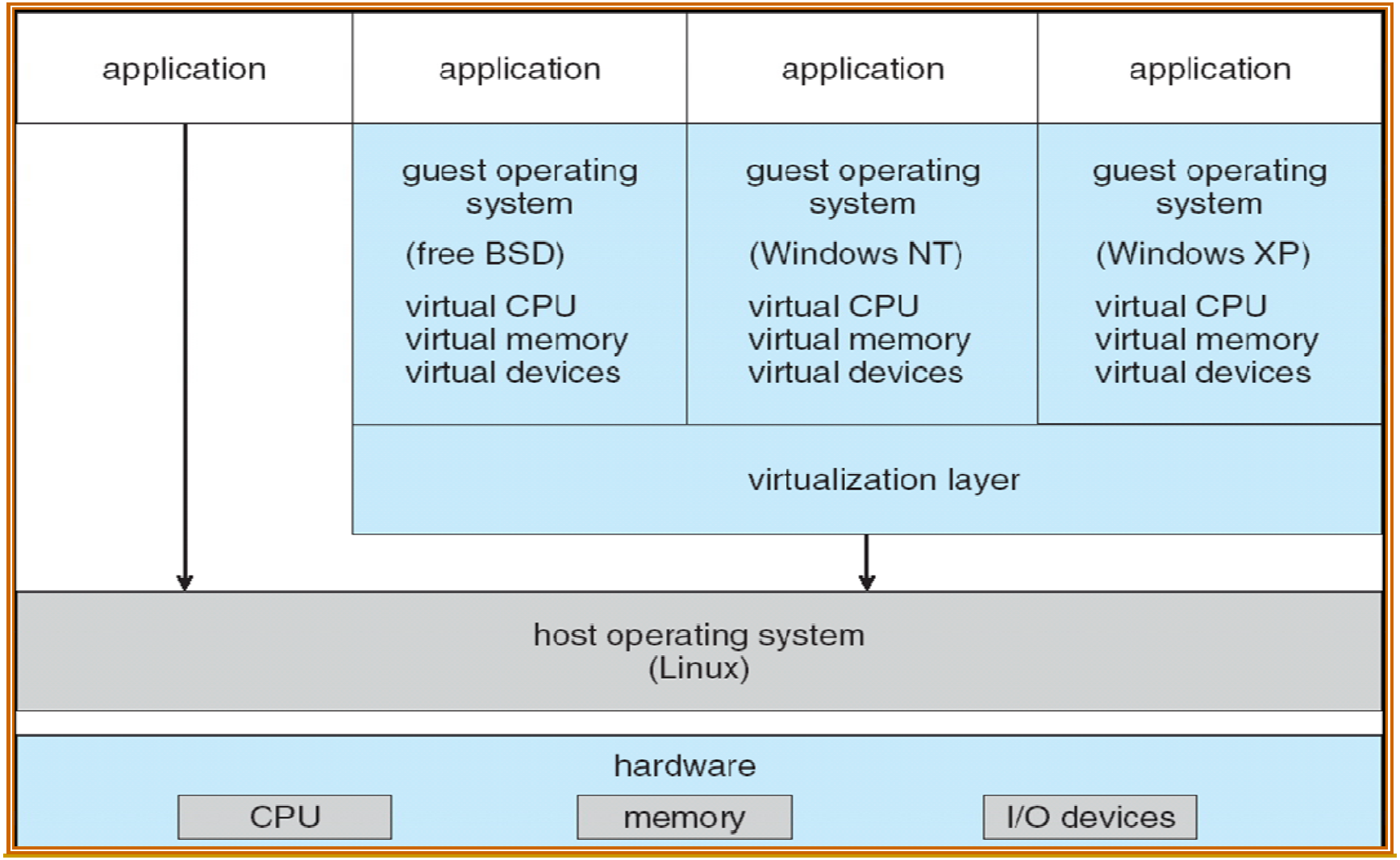


(a) Non-Virtual Machine AND (b) Virtual Machine

Virtual Machine (cont'd)

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine

VMware Architecture



Thank you!

Source: Galvin OS books/slides

Edited by: Nilesh Ghule

 # Linux Filesystem Structure:
- While installation of Linux system we need to create two partitions:
1. "/" partition also called as **data partition**
2. swap partition
- Linux follows FHS i.e. **FileSystem Heirarchy Standards.**
- data can be store into the **"data partition"** in an organized manner, so linux filesystem structure is having **heirachical structure**, which is a like an inverted tree.
- when any user loggedin into the system/opens a terminal, system takes the user bydefualt into the user's home dir.
- linux filesystem structure starts with **root dir ("/")**, and all the data can be stored/kept inside it under sub-dir's in an organized manner.
- root dir i.e. "/" dir contains sub-dir's like "/boot", "/bin", "/sbin", "/etc", "/lib", "/usr", "/home", "/root" "/mnt", "/dev" etc....

1. "/boot": contains static files of the boot loader, kernel, initrd (initial ramdisk).

2. "/bin": contains user commands in binary format, like ls, cat, cp etc...

3. "/sbin": contains admin/system commands in binary format, like lscpu, adduser, deluser, etc...

4. "/home": contains home dir's of all user for multi-user system.
- for any user on its creation of an account bydefault subdir by the name of the user got created by the operating system, in which that user can store data, user can have read, write as well execute perms in that dir.

5. "/root": root is a home dir for root user

6. "/etc": contains host specific system configuration files - it is like a "control panel" in windows.

- in computing configuration files (or config files) are files used to configure the parameters and initial settings for some computer programs. They are used for user applications, server processes and operating system settings.
- e.g. **YaST: Yet another Setup Tool** (Linux operating system setup and configuration tool) or **debconf** (for performing system-wide configuration taskson UNIX-like operating systems).
"/etc/opt": contains configuration files for **add-on-packages** that are stored in /opt.

7. "/dev": contains device files

8. "/lib": contains shared libraries required for /bin and /sbin.

9. "/mnt": contains temporarily mounted filesystems

10. "/media": mount point for removable media such as CDROMs.

11. **"/opt":** contains optional application software packages.

12. "/proc": virtual filesystem provides process and kernel information as a files. In Linux , corresponds to a **procfs** mount. Generally automatically generated and populated by the system, on the fly.

 # path - it is a unique location of any file in a filesystem structure that can be represent in a string format i.e. in sequence of chars format seperated filenames/sub-dir names by **delimiter char** or **delimiter.**
e.g. "/home/sachin"
- there are three delimiter chars:
1. slash ("/") -- in linux
2. backslash ("\") -- in windows
3. colon (":") -- in linux as well as in windows
- there are two ways by which we can mention path of any file/dir:

1. absolute path: path of any file/dir with respect to root directory ("/") i.e. it is a full path of any file/dir which starts with root dir ("/").

2. relative path: path of any file/dir with respect to the current working dir.

- **UNIX consideres/treats everything as a file,** i.e. from UNIX point of veiw (as Linux is UNIX like/based OS) there are 7 types of files in UNIX/Linux:

1. regular file (-): linux treats all text files, source files, audio files vedio files, image files etc....

2. directory file (d): directory special file whose contents are name of files and sub dir's.

3. character special device file (c): devices from which data gets transferred character by character are called as character special devices e.g. KBD, Monitor, Printer, serial ports & parellel ports etc...

- information of device file gets loaded into "/dev" while booting.

4. block special device file (b): devices from which data gets transefered block by block are called as block devices, e.g. all storage devices are block devices

- OS treats all block devices as "block special device file".

5. socket file (s): this is special type of file can be used for an IPC.

6. named pipe file (p): this file can be used for an IPC

7. linkable file (l): this is special type of file which contains info about another file.

+ Command Name: "pwd" - print/present working directory

- this command displays absolute path (full path) of the present working directory

- pwd comamand internally refers the value of shell variable by the name "PWD".

+ Command Name: "cd" - to change current working directory
$cd <dirpath> -- change dir to the dirpath
$cd / -- goto the root directory -- "/" slash -- is user short hand variable
$cd ~ -- goto the user's home dir -- "~" tild symbol -- is user short hand variable
$cd -- goto the user's home dir
$cd - -- goto the prev directory
$cd . -- remains in current dir
$cd .. -- goto the parent dir of the current working dir

+ Command Name: "ls" - to display contents of the directory
- contents of the dir are nothing but name of files and sub-dir's.
$ls <dirpath> -- it display contents of the dir by the name "dirpath".
$ls -- bydefault it displays contents of the current directory
$ls -l -- it display contents of the dir in a listing format.

total n
n = total no. of data blocks allocated for the files and sub dir's in a given dir
feild-1: first char of feild 1 denotes type the file next 3 chars of field 1 denotes access perms for user/owner
next 3 chars of field 1 denotes access perms for group members last 3 chars of field 1 denotes access perms for group others.
Feild-2: denotes no. of links exists for that file for regular file bydefualt no. of links i.e. link count = 1
for directory file bydefualt no. of links i.e. link count = 2.
feild-3 : denotes name of an user/owner
feild-4 : denotes name of group
feild-5 : denotes name of size of the file
feild-6 : denotes time stamps - date of creation and last modified date & time in ISO format.

feild-7 : name of the file
- In UNIX/Linux filename starts with (".") are considered as a hidden file
- the period (".")

options/flags/args description:
-h : displays size of the files in human redable format
-l : displays contents of the dir in a listing format
-a : display all contents of the dir (including hidden files)
-A : display all contents of the dir (including hidden files but excluding entries for "." & "..").
-i : display inode number of the files inode number is an unique idenfier of the file.
etc..... explore more options from man pages

When we create any new file, first inode for that file gets created into the filesystem.
- when we create a new file it has 3 things:
1. inode
2. directory entry
3. data blocks (optional)

+ directory file (d):
- directory is a special file, whose contents are names of files and sub dir's.
- **directory is a container** which contains directory entries of files and sub dir's inside it.
- directory entry of a file is nothing but the link of that file.
- one file may has multiple dir entries but one file can have only one "inode".

- In a single filesystem we can have more than one files having same name, but we cannot have more than one files having same "inode number" i.e.

inode number is an unique identier of a file in that filesystem.
- We can have multiple directory entries (i.e. no. of links) for a same file at different locations.
- Directory file is a like a container whose contents are name of its files and subdirs/ directory file is a container whose contents are "dir entires".

- Directory entry of any file contains min two fields:

1. **inode number**
2. **name of the file**

- When we create a new file, filesystem assigns/creates/allocates three things for that file:

1. **inode** (compulsory)
2. **directory entry** - gets created in its parent dir location => link of that file (compulsory)
3. **data blocks** (optional)

- Directory entry is also reffered as "hard link" of that file.
- In Linux filename starts with (.) dot is reffered as **hidden file.**

- When we create a new dir, bydefault every dir contains two hidden entries:

. **--> current dir**
.. **--> parent dir**

- To maintain parent-child relationship between files **single dot (.) & two dots (..)** can be maintained in a filesystem.

- **"link count"** = no. of links exists for that file in a filesystem i.e. no. of ways which we can access the file.
- bydefualt link count any regular file = 1
- bydefualt link count any directory file = 2
(For a dir file -> 1 dir entry gets created inside its parent dir location & one hidden dir

entry gets created inside that dir itself by the name single dot (.)).

- Command to execute a program w.r.t. its parent dir -> **$./program.out**

 # OS DAY-10

- Directory structure:
- from user point of view contents of the directory file are name of its files and sub dir's, whereas from filesystem point of view contents of directory files are directory entries of its files & sub dir's.
- **"directory entry" = inode number + filename**

- fileio program: (CRUD operations):
1. write record into a file:
step1 - open a file/create a new file: "rb+"/read as well as write
fopen() -> open()

- fopen() is a library function which internally makes call to open() system call api.

open() open a file or create a new file

0644 - access perms can be assigned for a file while creation in octal format

- first digit : leading 0 indicates it is an octal constant
- second digit : indicates access perms for owner/ user.
- third digit : indicates access perms for group members
- fourth digit : indicates access perms for others

4 - read
2 - write
1 - execute

0544 ->
0 : octal constant
5 : user can read as well execute
4 : grp members can read only
4 : others can read only

0744:
0 : octal constant
7 : user/owner - r+w+x
4 : grp members can only read
4 : other can only read

- "chmod" is used to change mode bits i.e. to change access perms

- **perror()** is a C library function which prints user message as well as actual system error message.

step2 - write a record into a file:
fwrite() -> write()

step3 - close a file:
fclose() -> close()

2. read records from a file:
step1 - open a file (file must exists already)
step2 - read all the records from a file one by one and display
step3 - close a file

+ fileio system calls/system call api : open(), write(), read(), close()

3. update record into a file
4. delete record from a file

+ for a text file
1. "w" - write only
2. "w+" - write as well read
3. "a" - append only
4. "a+" - append as well as read
5. "r" - read only
6. "r+" - read as well write

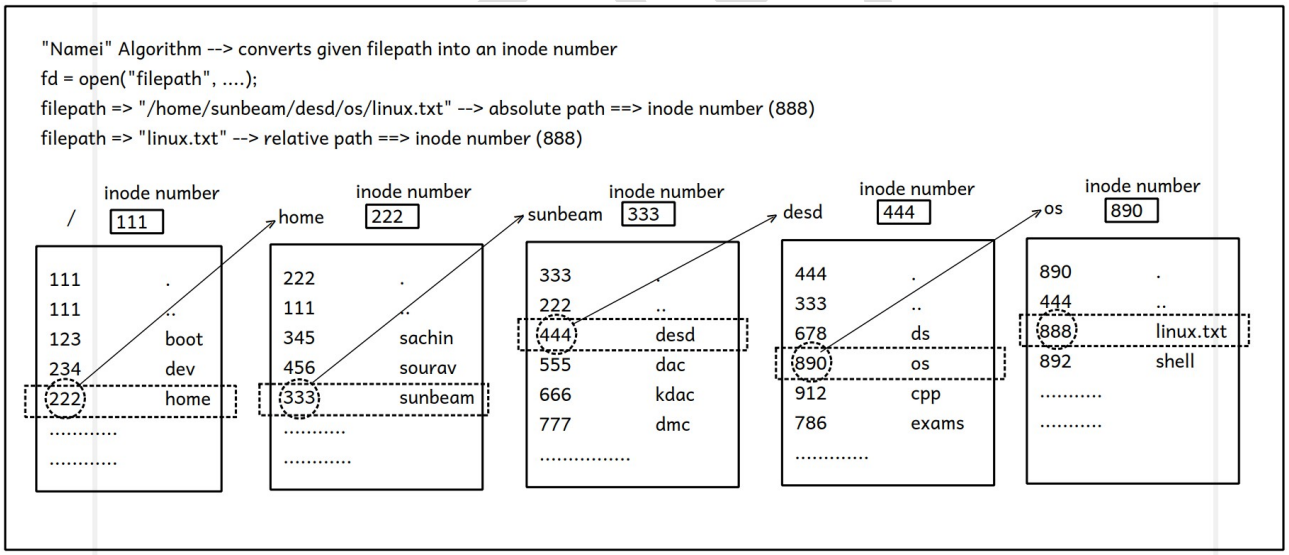
+ for a binary file:

1. "wb" - write only
2. "wb+" - write as well read
3. "ab" - append only
4. "ab+" - append as well as read
5. "rb" - read only
6. "rb+" - read as well write

 # open() system call internals:
int fd = open("filepath", O_RDONLY, 0644);

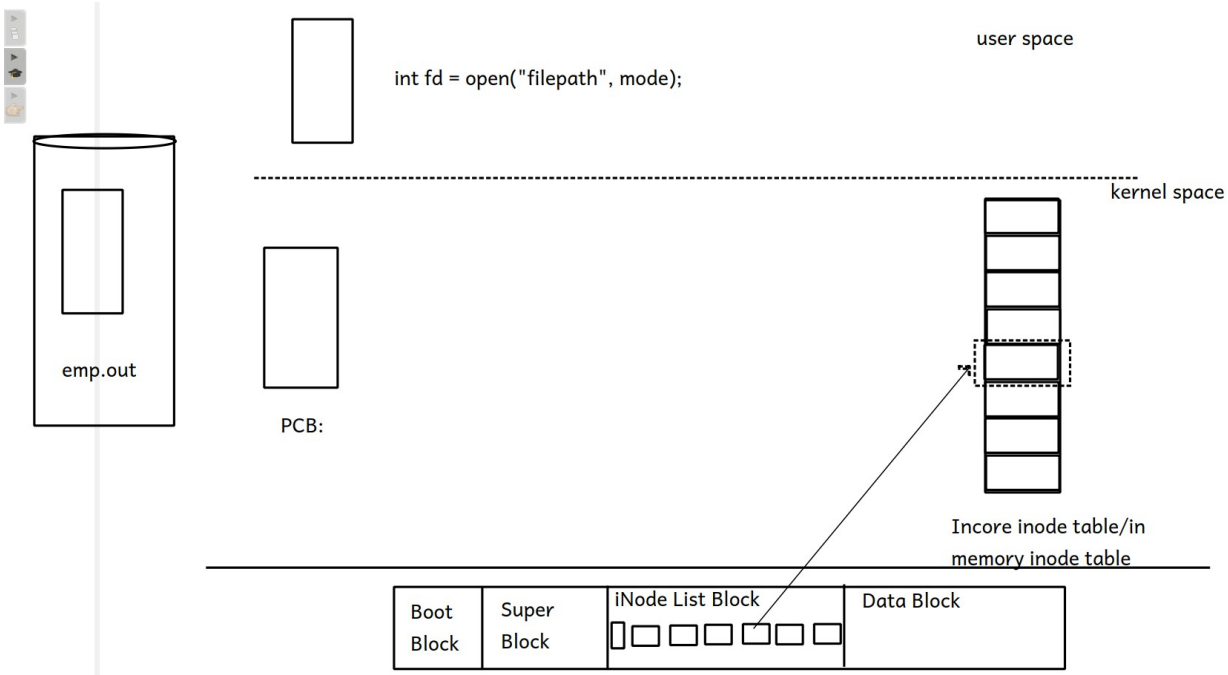
step1: it converts given filepath into its inode number.
- filepath may be absolute path or it may relative path
- algo which converts given filepath into its inode number is reffered as **"namei" algorithm**, it is called as **pathname translation.**

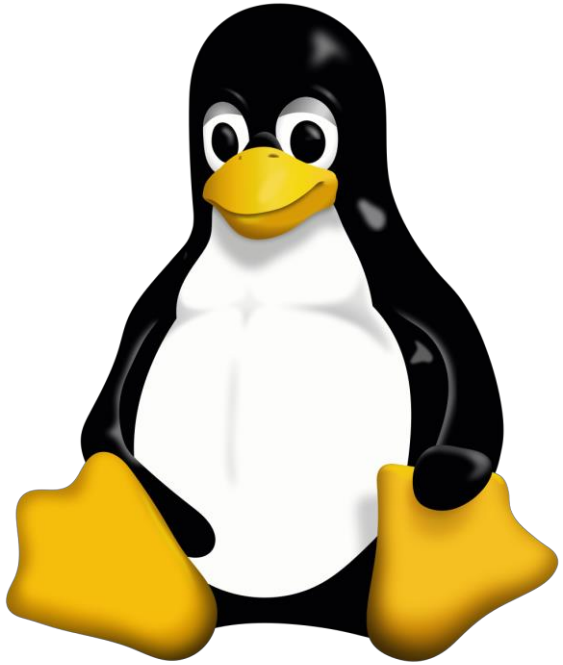
For Example:



step2: once open() syscall got the inode number of that file, its corresponding inode gets loaded into the main memory by searching it from inode list block, and inode entry of that file got added into one global table (kernel data structure) reffered as **"in-memory inode table".**

- **In-Memory/In-Core Inode Table:** contains list of inodes of all recently opened files.

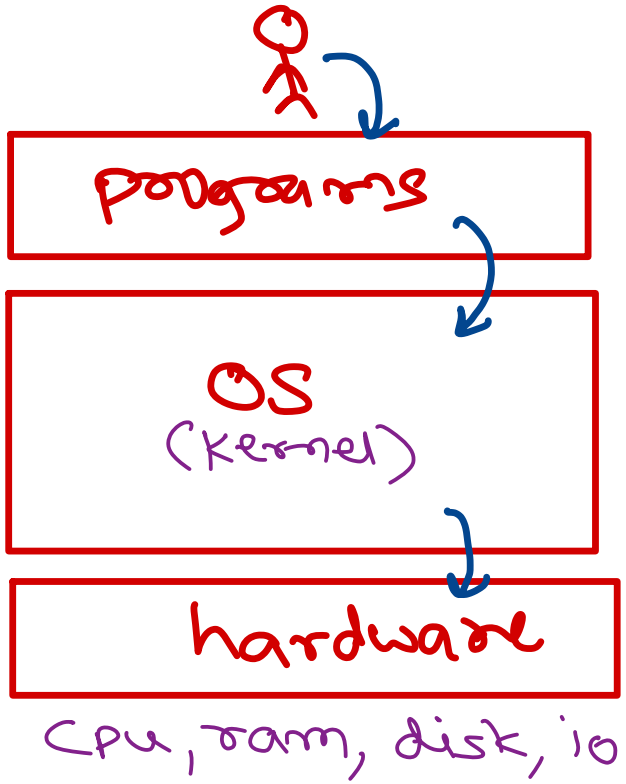

user space
int fd = open("filepath", mode);
kernel space
emp.out
PCB:
Incore inode table/in
memory inode table
Boot
Block
Super
Block
iNode List Block
Data Block



Operating System – Linux Programming

Sunbeam Infotech

Operating Systems



OS learning

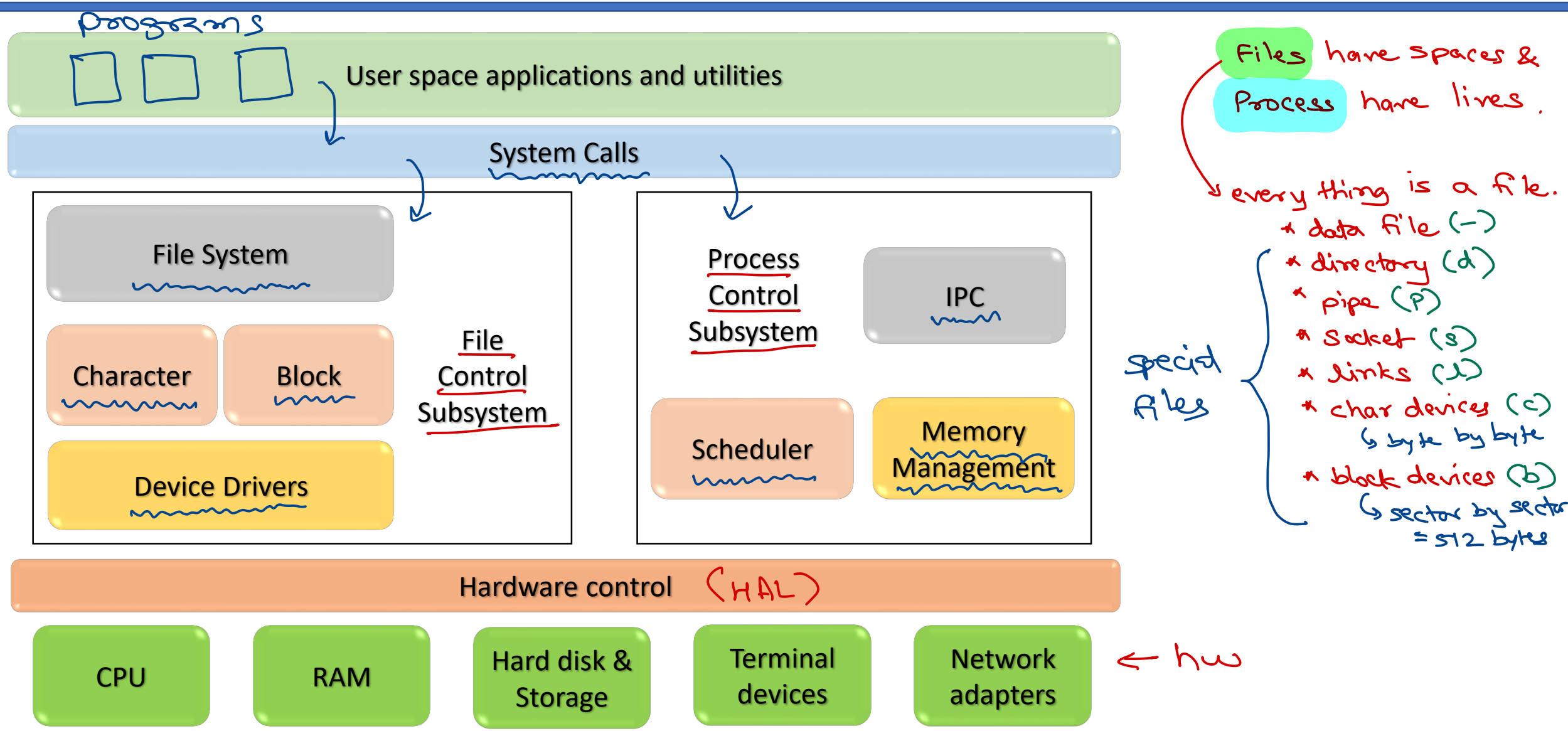
1. end user
   - commands
2. administrator
   - installation
   - shell scripts
3. programmer
   - system calls
4. designer
   - OS internals

OS functions

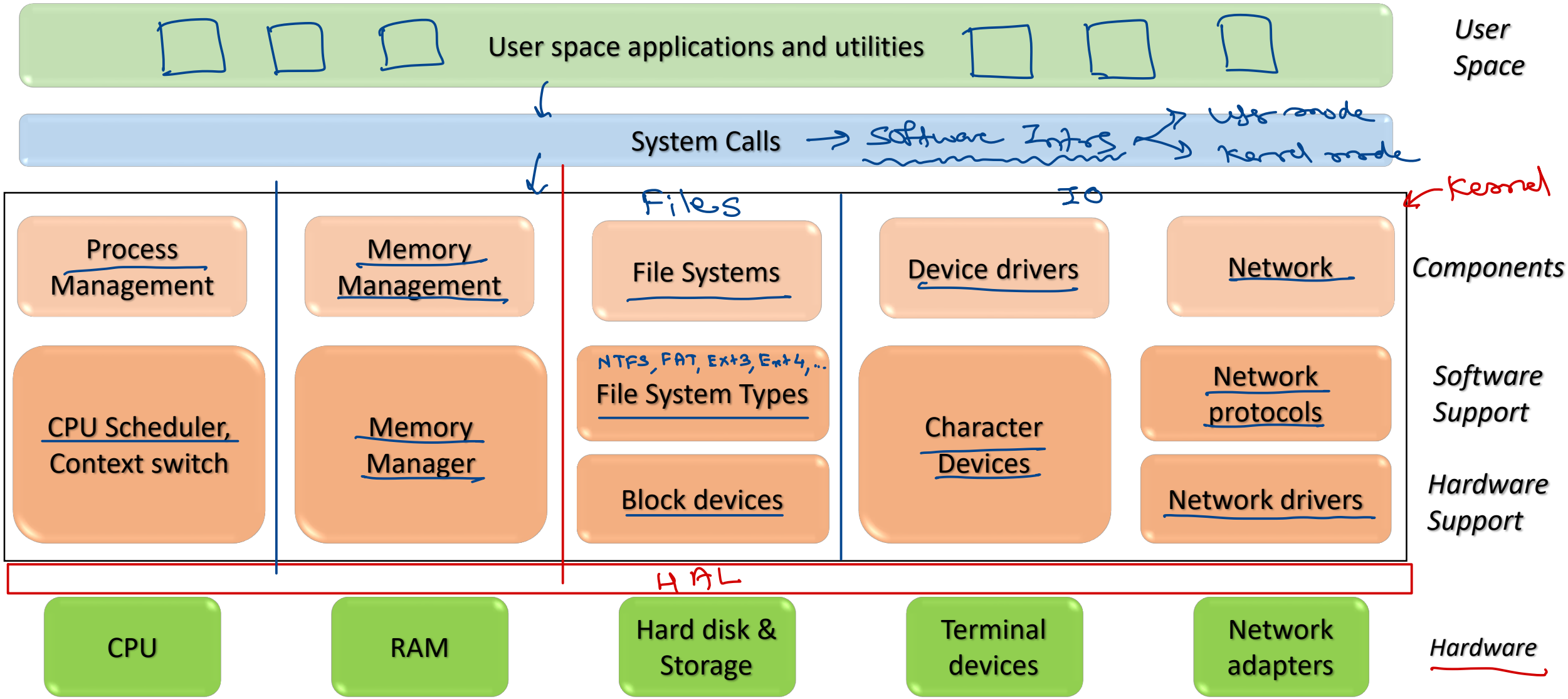
Core OS / Kernel
1. process mgmt
2. file & io mgmt
3. memory mgmt
4. cpu scheduling
5. hardware abstraction

6. user interfacing
7. networking
8. protection & security

Unix architecture

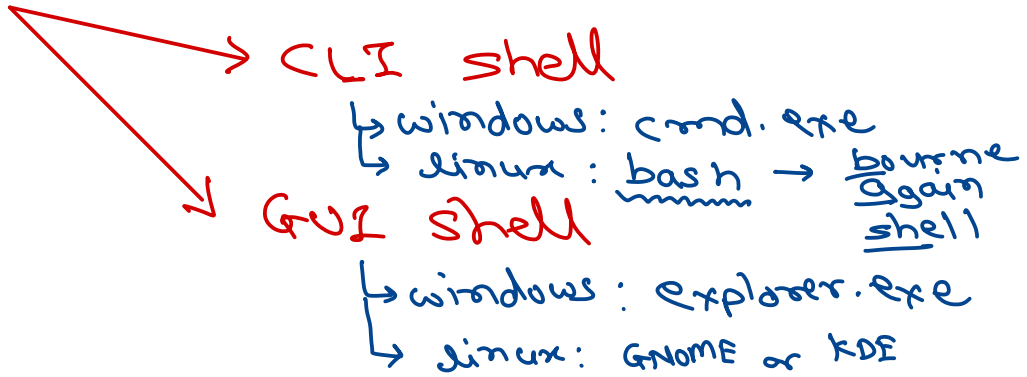


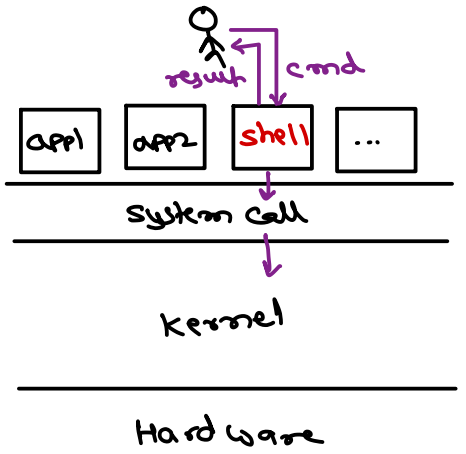
Linux kernel architecture



Linux shell

- Shell is a program that inputs commands from end user and get them executed from the kernel. It is also referred as command interpreter.
- Linux have variety of shells
  - bsh/bash
  - csh/tcsh → C shell
  - zsh
  - ksh → korn shell (unix)
- BASH shell
  - Compatible with bsh/sh.





Linux File-system hierarchy

- / → root of FS
  - boot → kernel (vmlinuz), bootloader (grub)
  - bin → executables / commands
  - sbin → system commands (admin)
  - lib → libraries (.so) and device drivers (.ko)
  - usr → installed programs / softwares
  - etc → config files (app, hw, sys config).
  - dev → device files (char & block)
  - proc → monitoring / dynamic config
  - sys → device mangement
  - tmp → temp file system (auto lost when shutdown).
  - mnt → mount point (to see other FS)
    - sudo mount -t vfat /dev/sdb1 /mnt → mounting pen drive.
    - ls /mnt
    - sudo umount /mnt
  - root → directory for admin(root) user.
  - home
    - user1
    - user2 → Documents, Downloads, Desktop, ...

If username is "sunbeam", its home dir is /home/sunbeam.

File system commands

- Absolute path
  - Complete path of file or directory.
  - Always starts from root (/).
  - Same path (irrespective of current directory)
- Relative path
  - Path of file or directory w.r.t. current directory.
  - Doesn't starts from root (/).
  - Depends on current directory.
- Special directories
  - . : current directory → ./a.out
  - .. : parent directory
  - ~ : home directory → e.g. /home/sunbeam
- File system commands
  - pwd
  - cd dirpath → absolute or relative

File system commands

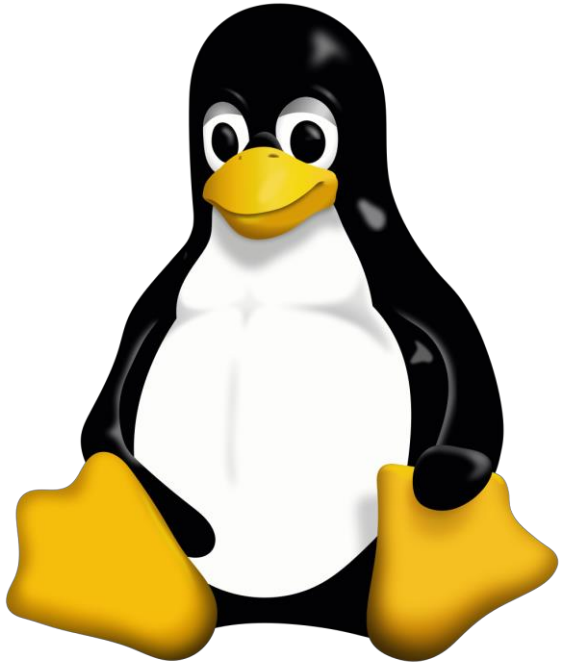
- ls dirpath
  - ls -l dirpath
- mkdir dirpath and rmdir dirpath
- cat filepath
  - cat > filepath
- cp filepath dirpath
  - cp -r srcdirpath destdirpath
- mv filepath dirpath
  - mv oldfile newfile
  - hidden files
- rm filepath
  - rm -r dirpath
- Shell wildcards
  - *
  - .

Shell advanced topics

- Redirection
  - Input redirection (<)
    - command < in_file
  - Output redirection (>)
    - command > out_file
    - command >> out_file → append output in file
  - Error redirection (2>)
    - command 2> err_file
- Pipe
  - command1 | command2 →
- Command execution
  - $?
  - command1 && command2
  - command1 || command2
  - command &

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>



Operating System – Linux Programming

Sunbeam Infotech

File system commands

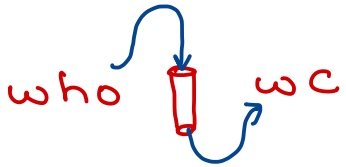
- ls dirpath
  - ls -l dirpath
- mkdir dirpath and rmdir dirpath
- cat filepath
  - cat > filepath
- cp filepath dirpath
  - cp -r srcdirpath destdirpath
- mv filepath dirpath
  - mv oldfile newfile
  - hidden files
- rm filepath
  - rm -r dirpath
- Shell wildcards
  - *
  - ?

Shell advanced topics

- Redirection
  - Input redirection (<)
    - command < in_file
  - Output redirection (>)
    - command > out_file
    - command >> out_file → append output in file
  - Error redirection (2>)
    - command 2> err_file

sort < in.txt > out.txt 2> err.txt

- Pipe
  - command1 | command2 →



- Command execution
  - $? → exit code of previous command/program → 0: success; non-zero: failure.
  - command1 && command2 → if first cmd is success, then run second cmd.
  - command1 || command2 → if first cmd is failed, then run second cmd.
  - command & → async cmd execution. → shell doesn't wait for cmd to complete.

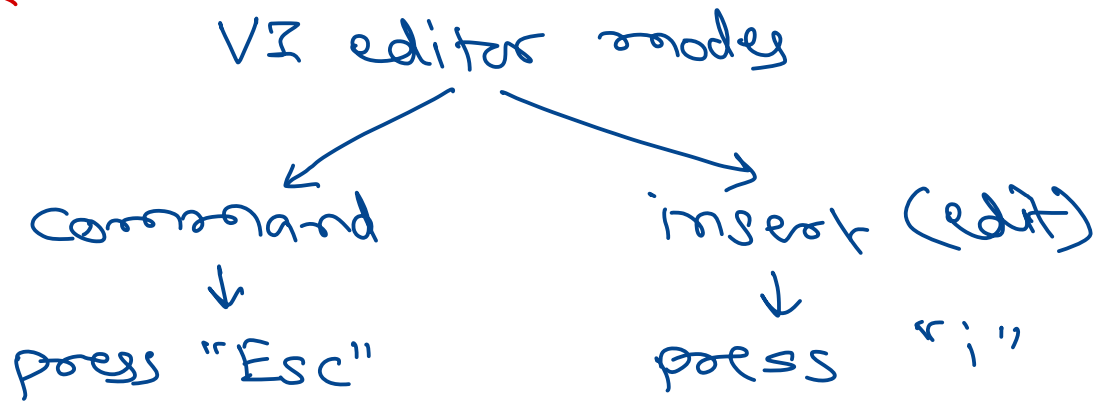
By default, shell waits for cmd to be completed.

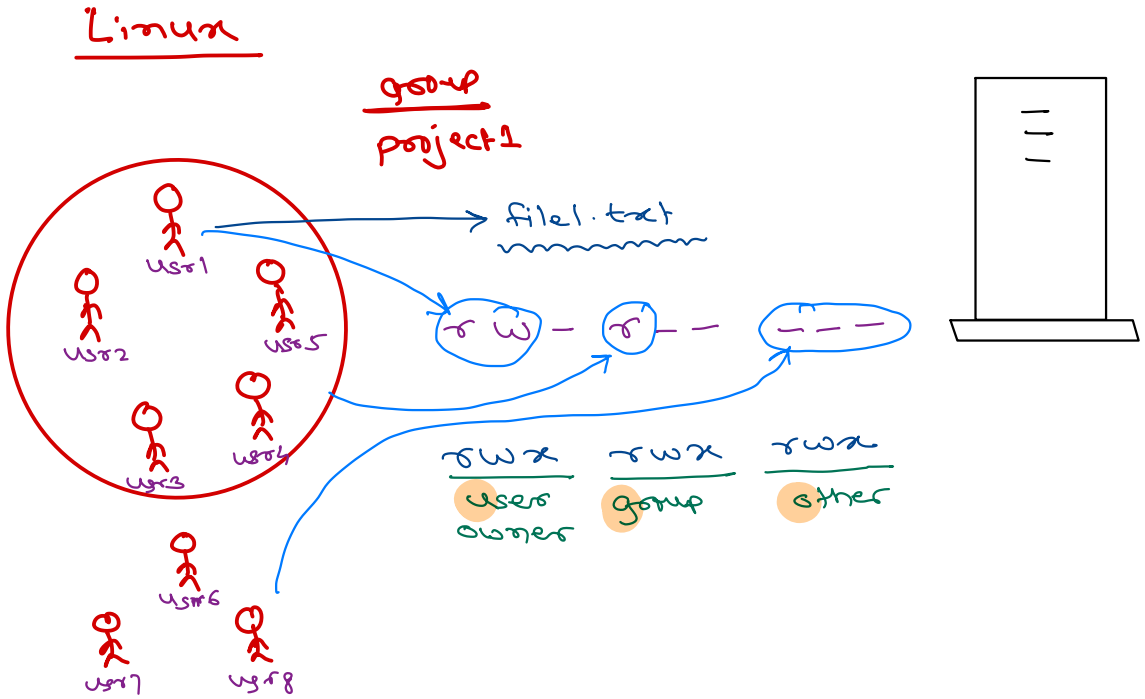
Regular expression

- Regex commands
  - grep → GNU Regular Expression Parser
  - egrep → Extended GNU Regular Expression Parser → grep -E
  - fgrep → Fixed GNU Regular Expression Parser → grep -F
- Regex wildcard characters
  - $ → ends with
  - ^ → starts with
  - . → any single char
  - [scanset] → any single char in given scan set.
    [^scanset] → any single char not in scan set.
  - * → 0 or more occurrency of prev char/group
  - + → 1 or more occurrency of prev char/group
  - ? → 0 or 1 occurrency of prev char/group
  - {n}, {m,n}, {m,}, {,n} → num of occurrency of prev char/group (>=m, <=n)
  - (w1|w2|w3) → find a word from w1, w2 or w3.
  - \ → remove special meaning of wild card char.
- Building regex → "^[0-9]{10}$" → 10 digit mobile

VI editor → World's best editor → vim → VI improved.

- Developed by Bill Joy. → UCB → BSD UNIX
- Operations
  - create/open file → vim filepath
  - write/save → :w
  - quit → :q
  - write and quit → :wq
  - quit without saving → :q!
- .vimrc file
- Copy/Cut
  - line
  - c lines
  - m to n lines
  - word
  - c words
- Paste
- Find word




Linux
group
project1
file1.txt
user1
user2
user5
user3
user4
user6
user7
user8
rwx
rwx
rwx
user
owner
group
other

BASH shell script

- Shell script is collection of shell commands along with programming constructs.
- Shebang line
  - #!/bin/bash
- echo “Hello World”
  - -n
  - -e
- Shell variables
  - var=value
  - echo “$var”
- expr command
- Command substitution
  - var=`command`
  - var=$(command)
- User input
  - read var

BASH shell script

- if-else
  - if [condition]
    then
        …
    fi
  - if [condition]
    then
        …
    elif [condition]
    then
        …
    else
        …
    fi

- test command
  - -eq, -ne, -gt, -lt, -ge, -le
  - -a, -o, !
  - -f, -d, -w, -r, -x

- loops
  - while [condition]
  - do
  - …
  - done

  - until [condition]
  - do
  - …
  - done

BASH shell scripts

- case
  - case expr in
    c1)
        …
        ;;
    c2)
        …
        ;;
    c3)
        …
        ;;
    *)
        …
    esac

- for loop
  - C like for loop
    - for ((initialization; condition; modification))
      do
          …
      done
  - for-each loop
    - for var in collection
      do
          …
      done

BASH shell script

- Positional parameters
  - terminal> ./script.sh arg1 arg2 arg3
  - Special variables
    - $0
    - $1, $2, …, $9
    - $#
    - $*
    - shift command
- BASH functions

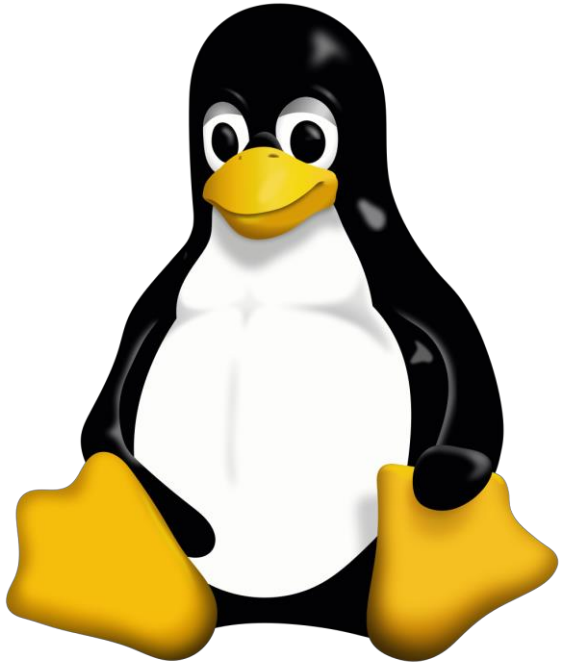
```
function my_func() {
        …
}

result=$(my_func arg1 arg2 …)
```

BASH shell script

- Arrays
  - arr=(val1 val2 val3 …)
  - ${arr[0]}, ${arr[1]}, …
  - ${arr[*]} – collection of values
  - ${#arr[*]} – count of values
- Strings
  - str='string value'
  - ${#str} – string length
  - ${str:start_index} – substring
  - ${str:start_index:count} – substring
  - if [[$str =~ regex]]; then echo "true"; fi
  - ${str/find/replace}
- Directory operations
  - pushd dirpath
  - popd
  - dirs -v

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>



Operating System – Linux Programming

Sunbeam Infotech

BASH shell script

- Shell script is collection of shell commands along with programming constructs.
- Shebang line
  - #!/bin/bash
- echo “Hello World”
  - -n
  - -e
- Shell variables
  - var=value
  - echo “$var”
- expr command
- Command substitution
  - var=`command`
  - var=$(command)
- User input
  - read var

BASH shell script

- if-else
  - if [condition]
    then
        …
    fi
  - if [condition]
    then
        …
    elif [condition]
    then
        …
    else
        …
    fi

- test command
  - -eq, -ne, -gt, -lt, -ge, -le
  - -a, -o, !
  - -f, -d, -w, -r, -x

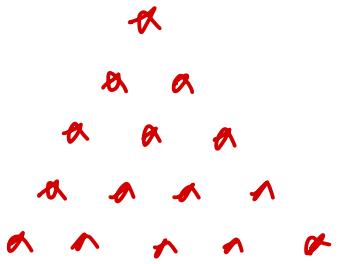
- loops
  - while [condition]
  - do
  - …
  - done

  - until [condition]
  - do
  - …
  - done

BASH shell scripts

- case
  - case expr in
    c1)
    ...
    ;;
    c2)
    ...
    ;;
    c3)
    ...
    ;;
    *)
    ...
    esac

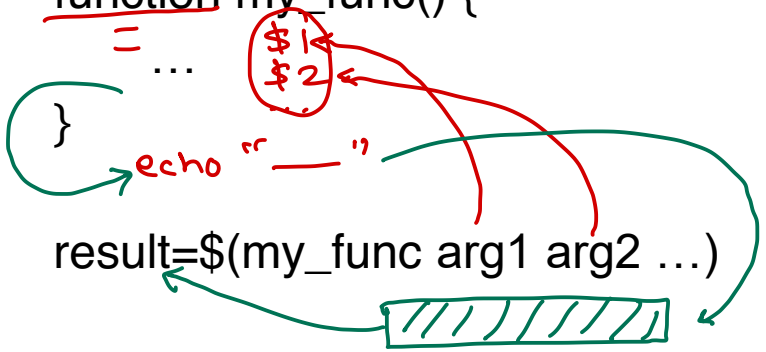
- for loop
  - C like for loop
    - for ((initialization; condition; modification))
      do
      …
      done
  - for-each loop
    - for var in collection
      do
      …
      done



BASH shell script

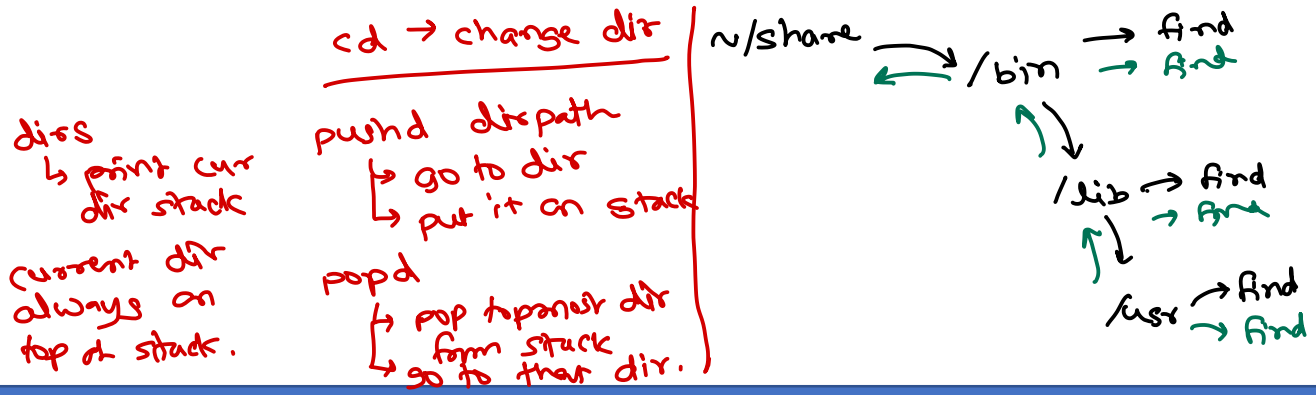
- Positional parameters → cmd line args
  - terminal> ./script.sh arg1 arg2 arg3
  - Special variables
    - $0 – script name
    - $1, $2, …, $9 → arg1, arg2, …, arg9
    - $# – cnt of args
    - $* → list of all args – for-each
    - shift command → shift/scrap first n args. further args are renumbered as $1, $2, …
- BASH functions

```
function my_func() {
    …
}

result=$(my_func arg1 arg2 …)
```



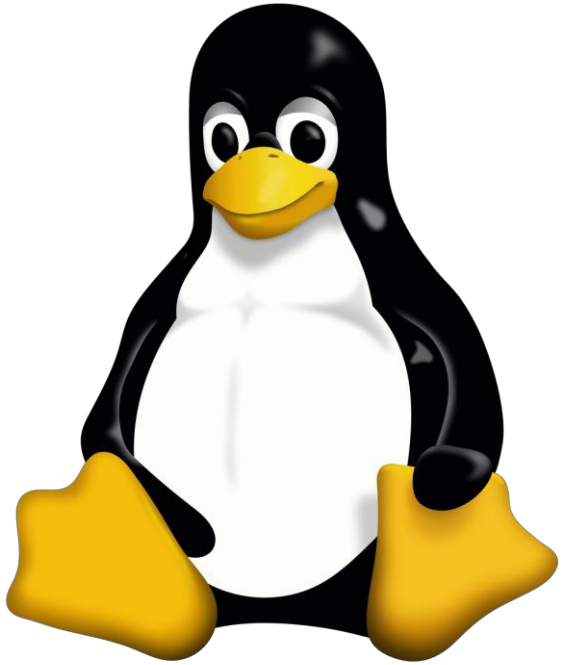
BASH shell script

- Arrays
  - arr=(val1 val2 val3 …)
  - ${arr[0]}, ${arr[1]}, …
  - ${arr[*]} – collection of values
  - ${#arr[*]} – count of values
- Strings
  - str='string value'
  - ${#str} – string length
  - ${str:start_index} – substring
  - ${str:start_index:count} – substring
  - if [[$str =~ regex]]; then echo "true"; fi
  - ${str/find/replace}
- Directory operations
  - pushd dirpath
  - popd
  - dirs -v



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>



Operating System – Linux Programming

Sunbeam Infotech

BASH shell script

- BASH special files
  1. ~/.bash_profile (login shell)
  2. ~/.bash_login (login shell)
  3. ~/. profile (login shell)
  - ~/.bashrc (interactive non-login shell)
  - ~/.bash_logout (login shell)
- Running script in current shell
  - source ./script.sh
  - . ./script.sh

vim ~/.profile
vim ~/.bashrc

1 → new bash shell
3 → bsh/ bash shell.

new terminal in GUI.

shell
- interactive
  - bash prompt
  - terminal ✓
  - ssh ✓
- non-interactive
  - terminal> bash demo.sh
  - ✓ internally bash cmd /pgm is executed.
  - ✓ To execute given script.

login

* ssh
  - secure shell
    - bash
    - Comm^n in encrypted
  - Server: sshd
    - port = 22
  - client: ssh/scp

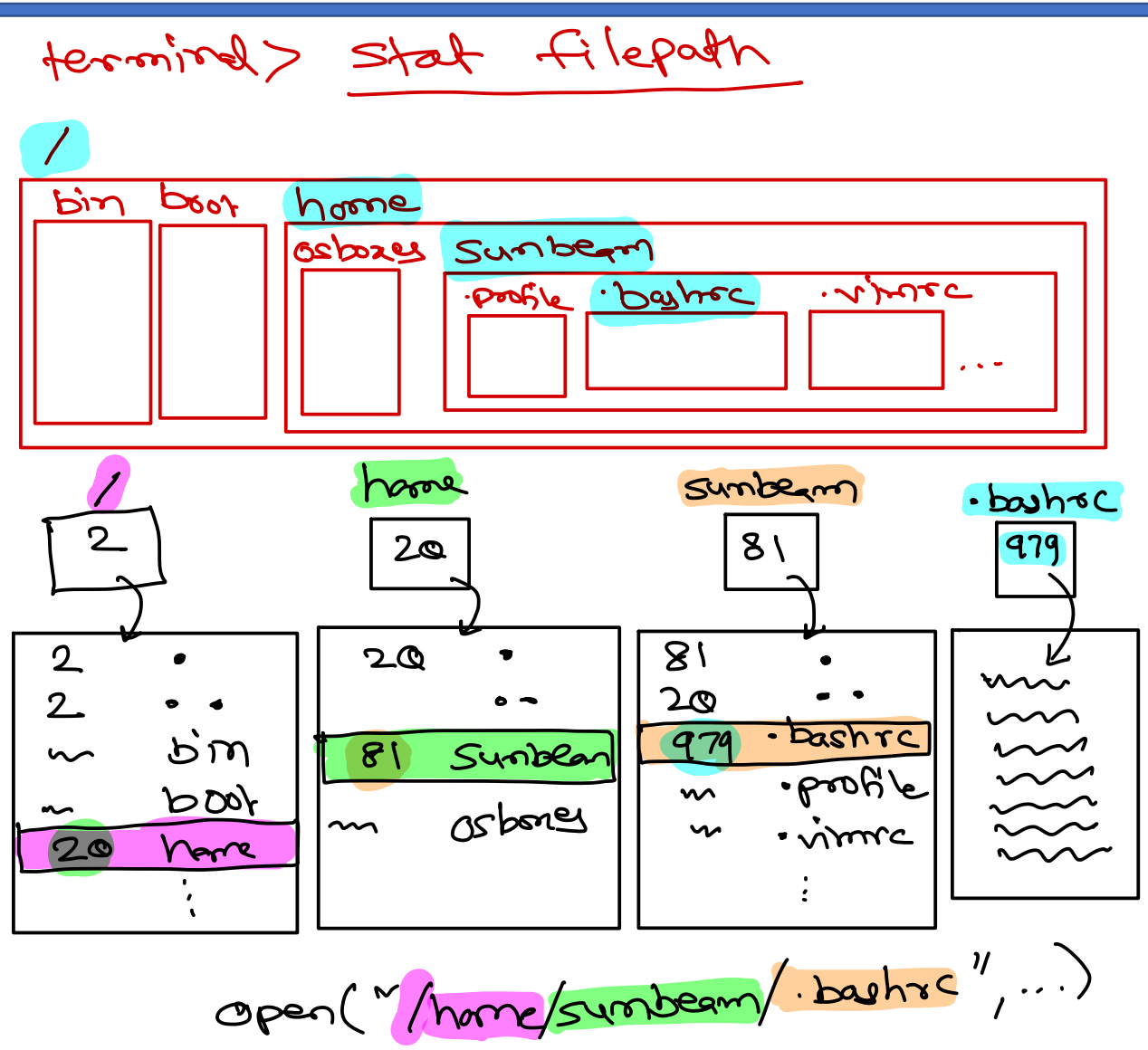
* telnet
  - shell
    - bash
    - Comm^n
  - not enc-
    - port = 23

env command
- environment variables.
  - imp info about system.
    - $PATH
    - $USER, $HOME
    - $SHELL
    - $PS1 → shell prompt
  - export var=value

```
int main(int argc,
         char *argv[],
         char *envp[])
{
  // C program
}
```

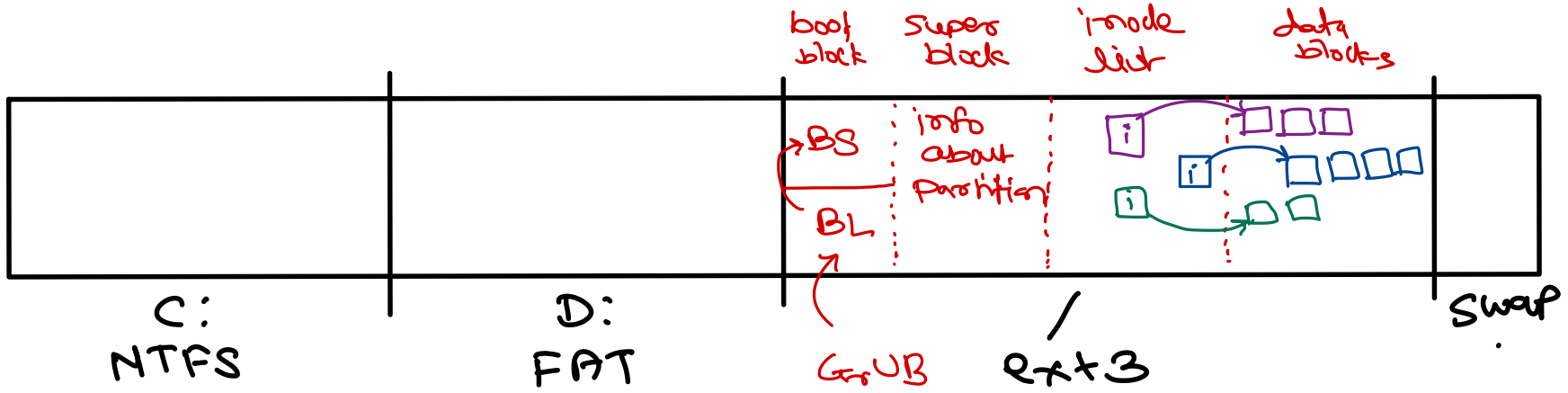
File

- File is collection of data and information on storage device.
- File = Data (Contents) + Metadata (Information)
- File metadata is stored in inode (FCB). → struct
  - Type, Mode, Size, User & Group, Links.
  - Timestamps, Info about data blocks.
- File data is stored in data block(s).
- File types: regular, directory, link, pipe, socket, character and block.
- Directory file contains directory entries for each sub-directory and file in it. Each dentry contains inode number & data block.



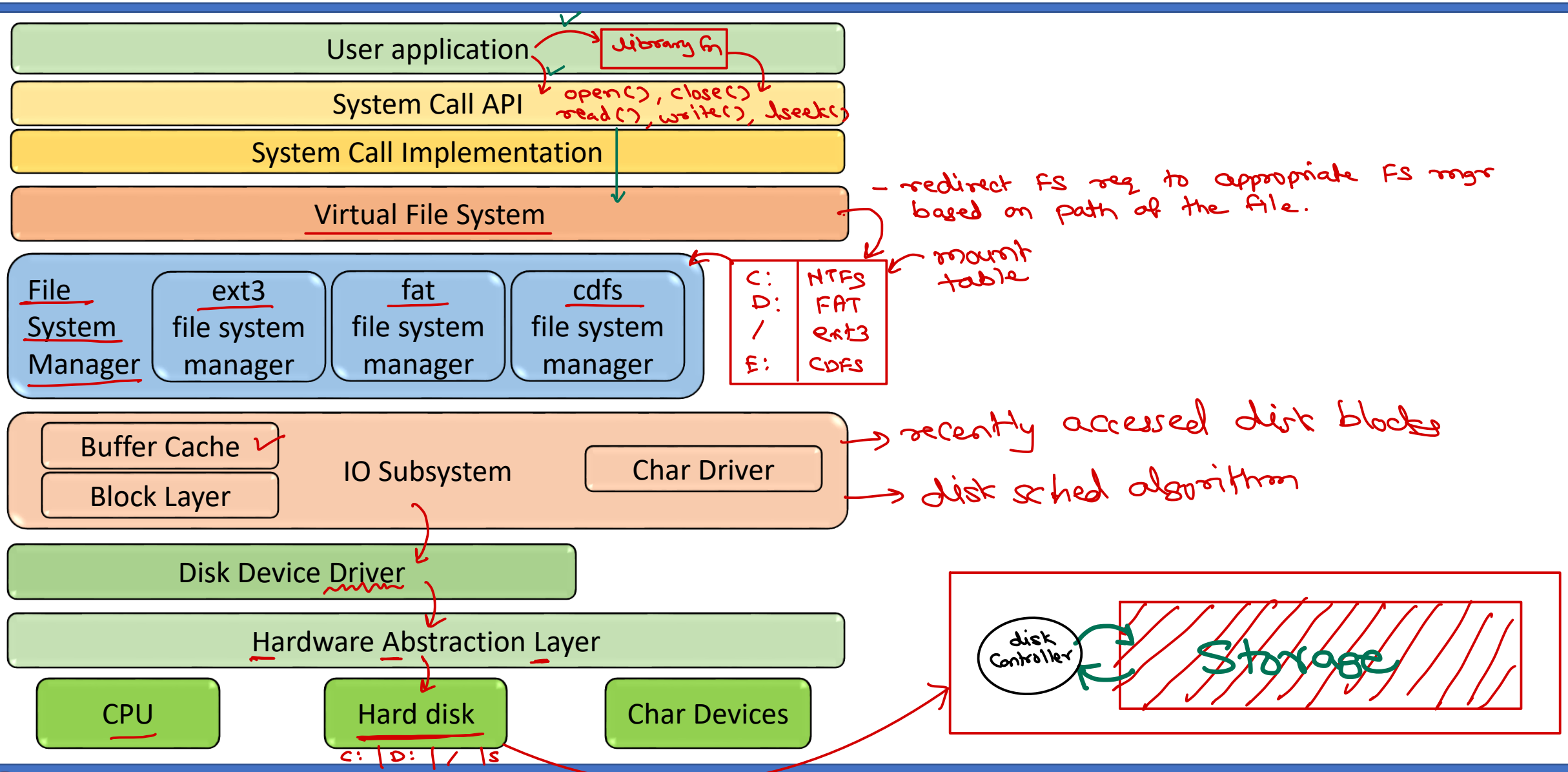
File system

- File system is way of organizing files on the disk.
- File systems have
  - Boot block
  - Super block
  - Inode list
  - Data blocks



- Content/arrangement of these components will change FS to FS.
- Different FS use different algorithms/ data structures to allocate disk.
- File system layout & operations are handled by file system manager using IO subsystem and device driver.

File System architecture

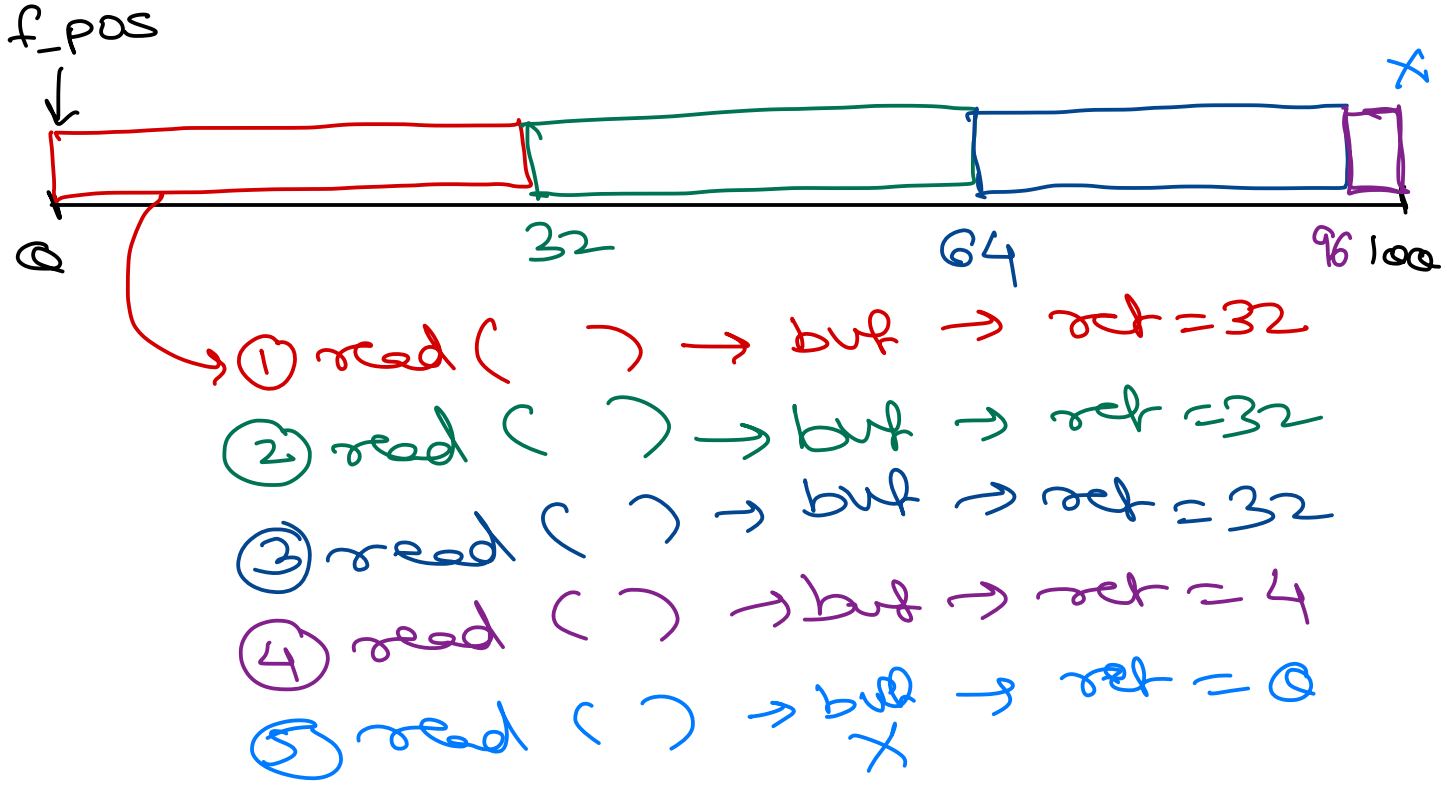


File IO System Calls

- File system operations deals with metadata (inode and dentry).
  - stat(), link(), symlink(), unlink(), mkdir(), …
- File IO operations deals with data/contents.
  - open(), close(), read(), write(), lseek(), ioctl(), …
- fd = open("filepath", flags, mode);
  - Open or create a file.
- close(fd);
  - Close opened file.
- ret = read(fd, buf, nbytes);
  - Read from file into user buffer.
- ret = write(fd, buf, nbytes);
  - Write from file into user buffer.
- newpos = lseek(fd, offset, whence);
  - Change file current position.

File copy program

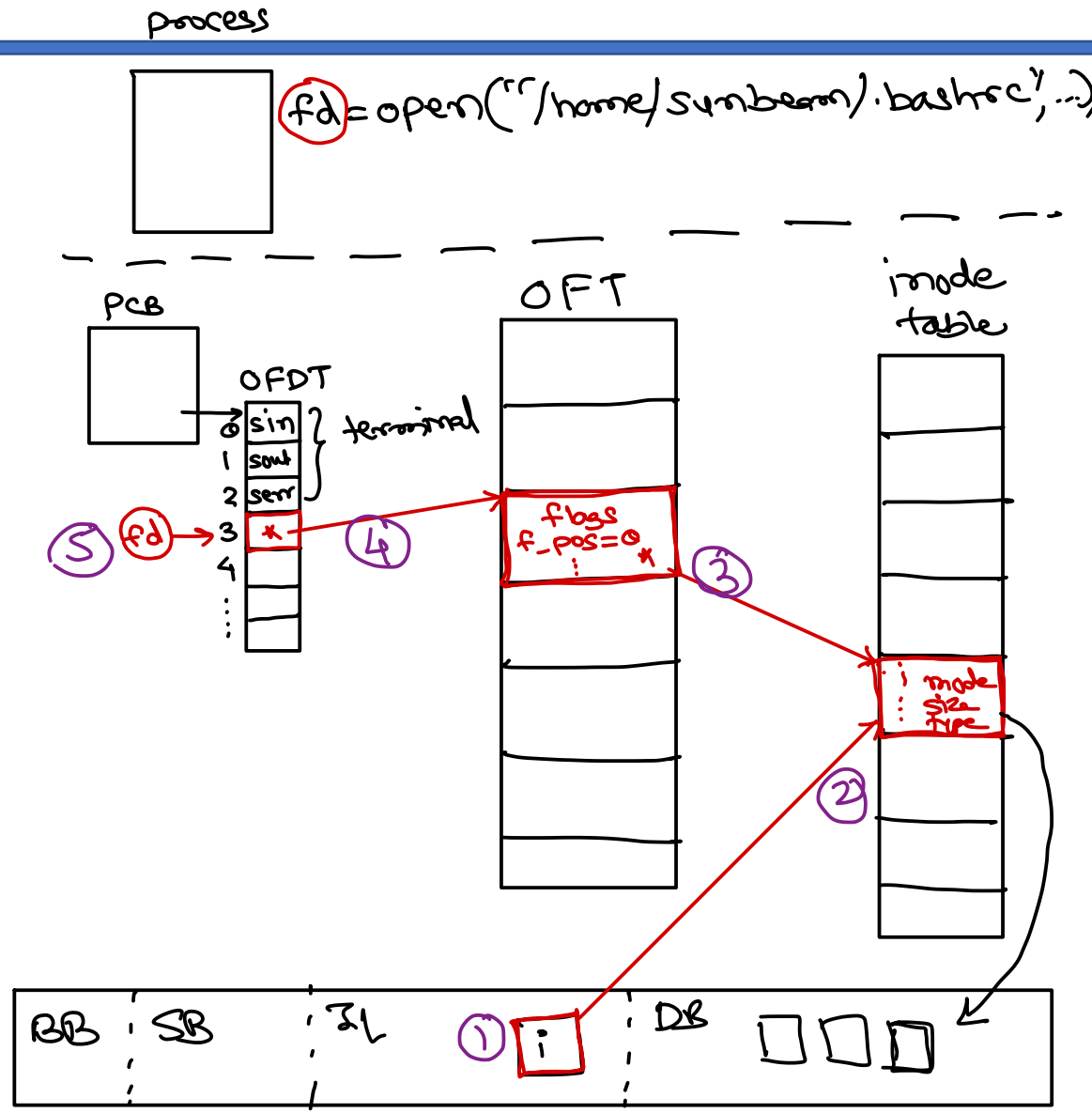
- 1. Open source file for reading.
- 2. Get file metadata (mode).
- 3. Open destination for writing.
  - Truncate if exists.
  - Create if doesn't exists.
- 4. Read n bytes from source file.
- 5. Write them into destination file.
- 6. Repeat 4-5 until file ends.
- 7. Close destination file.
- 8. Close source file.



fd = open("filepath", flags, mode);

1. Convert path name into inode number. (namei())
2. Load inode into in-memory inode table.
3. Make an entry into open file table (OFT).
4. Add pointer to OFT into open file descriptor table (OFDT).
5. Return index of OFDT to the program, known as file descriptor (FD).

- struct inode → inode table entry
- struct file → OFT entry



ret = read(fd, buf, nbytes);

- Get current file position from OFT entry.
- Calculate file logical block number and block byte offset.
- Convert file logical block to disk block using data blocks information in inode.
- Check if disk block is available in buffer cache.
- If not available, instruct disk driver to load the block from disk.
- Disk driver initiate disk IO and block the current process.
- Read desired part of block from buffer cache into user buffer. Update file position into OFT.
- Repeat for subsequent file blocks, until desired number of bytes are read.
- Return number of bytes allocated.

ret = write(fd, buf, nbytes);

- Get current file position from OFT entry.
- Calculate file logical block number and block byte offset.
- Convert file logical block to disk block using data blocks information in inode.
- Check if disk block is available in buffer cache.
- In not available, instruct disk driver to load the block from disk.
- Disk driver initiate disk IO and block the current process.
- Overwrite on intended part of block in buffer cache (from user buffer). Update file position into OFT. Mark the block in buffer cache as dirty, so that it will be flushed on the disk.
- Repeat for subsequent file blocks, until desired number of bytes are written.
- Return number of bytes allocated.

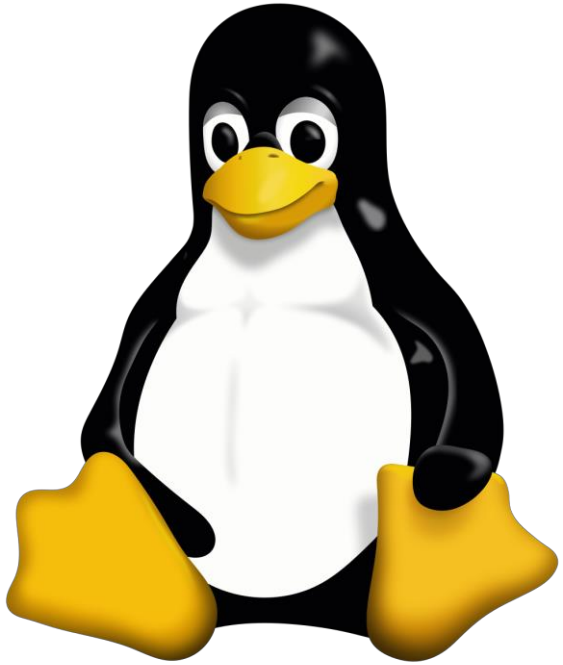
close(fd) and newpos = lseek(fd, offset, whence);

- close(fd);
  - Decrement reference count in OFT entry.
  - If count is zero, release the file/resources.

- newpos = lseek(fd, offset, whence);
  - Get current file position from OFT entry.
  - Calculate new file position.
  - Update file position in OFT entry.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

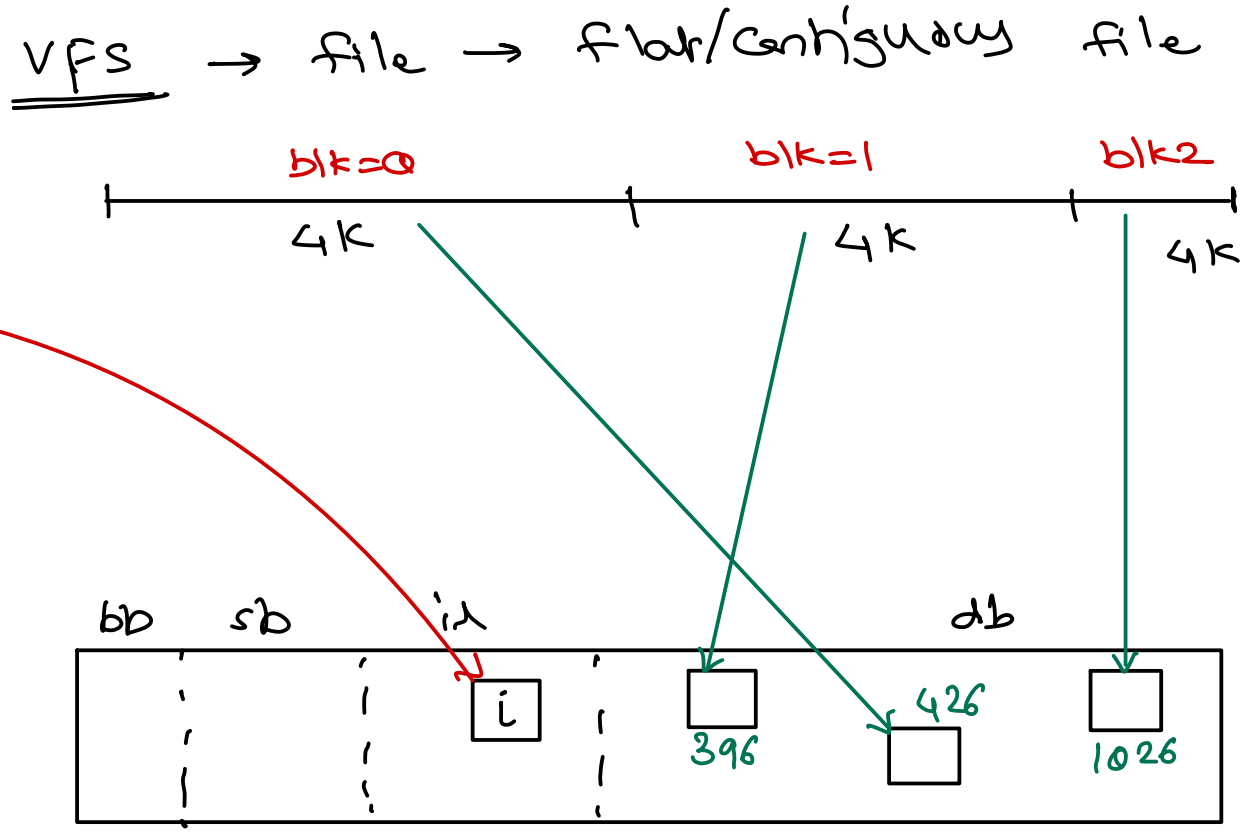


Operating System – Linux Programming

Sunbeam Infotech

ret = read(fd, buf, nbytes);

- Get current file position from OFT entry.
- Calculate file logical block number and block byte offset.
- Convert file logical block to disk block using data blocks information in inode.
- Check if disk block is available in buffer cache.
- If not available, instruct disk driver to load the block from disk.
- Disk driver initiate disk IO and block the current process.
- Read desired part of block from buffer cache into user buffer. Update file position into OFT.
- Repeat for subsequent file blocks, until desired number of bytes are read.
- Return number of bytes allocated.



ret = write(fd, buf, nbytes);

- Get current file position from OFT entry.
- Calculate file logical block number and block byte offset.
- Convert file logical block to disk block using data blocks information in inode.
- Check if disk block is available in buffer cache.
- In not available, instruct disk driver to load the block from disk.
- Disk driver initiate disk IO and block the current process.
- Overwrite on intended part of block in buffer cache (from user buffer). Update file position into OFT. Mark the block in buffer cache as dirty, so that it will be flushed on the disk.
- Repeat for subsequent file blocks, until desired number of bytes are written.
- Return number of bytes allocated.

close(fd) and newpos = lseek(fd, offset, whence);

- close(fd);
  - Decrement reference count in OFT entry.
  - If count is zero, release the file/resources.

C → fseek(fp, offset, whence);
↳ 0 - SEEK_SET
1 - CUR
2 - END

- newpos = lseek(fd, offset, whence);
  - Get current file position from OFT entry.
  - Calculate new file position.
  - Update file position in OFT entry.

ret = lseek(fd, 0, SEEK_SET);
↳ take f_pos at start of file.
↳ C - rewind(fp);

ret = lseek(fd, 0, SEEK_END);
↳ take f_pos to end of file.
↳ returns - file size

ret = lseek(fd, 0, SEEK_CUR);
↳ doesn't change file pos.
↳ return current file pos.
↳ C → ftell()

File syscalls

File IO	C
① open()	fopen() - FILE*
② close()	fclose()
③ read()	fread()
④ write()	fwrite()
⑤ lseek()	fseek()

Directory Read	C
① openat()	opendir() - DIR*
② getdents()	readdir()
③ close()	closedir()

Filesystem syscalls

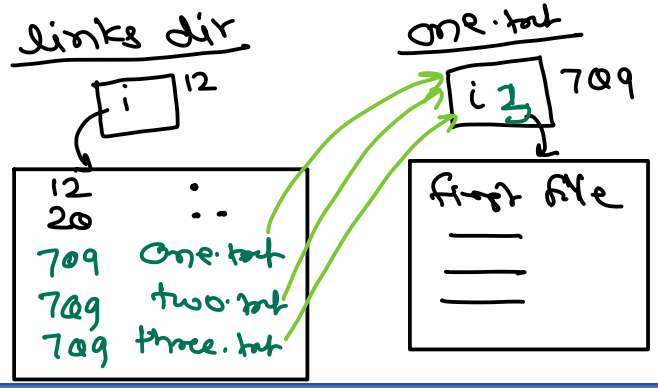
① mkdir()
② stat()
③ rm cmd → unlink()
④ mv cmd → link() & unlink()

hard link

↳ another direnty refering to same inode.

↳ terminal) ln filepath linkpath

link(,);



rm cmd

↓

unlink(filepath)

① del dentry
② decr link cnt
③ if link cnt become zero, then release inode & data blocks of file.

symbolic links

* windows shortcut
  - ↳ create a new special file (1) to keep addr/path of target file.
* ln -s filepath linkpath
  - ↳ symlink(,);
* good practice to use absolute path while creating sym link

Disk allocation mechanisms

① contiguous alloc

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40

f1.txt → 18 3

f2.txt → 26 4

f3.txt → 3 8

② linked alloc

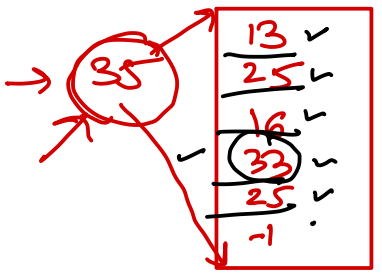
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40

f.txt → start 13 end 37

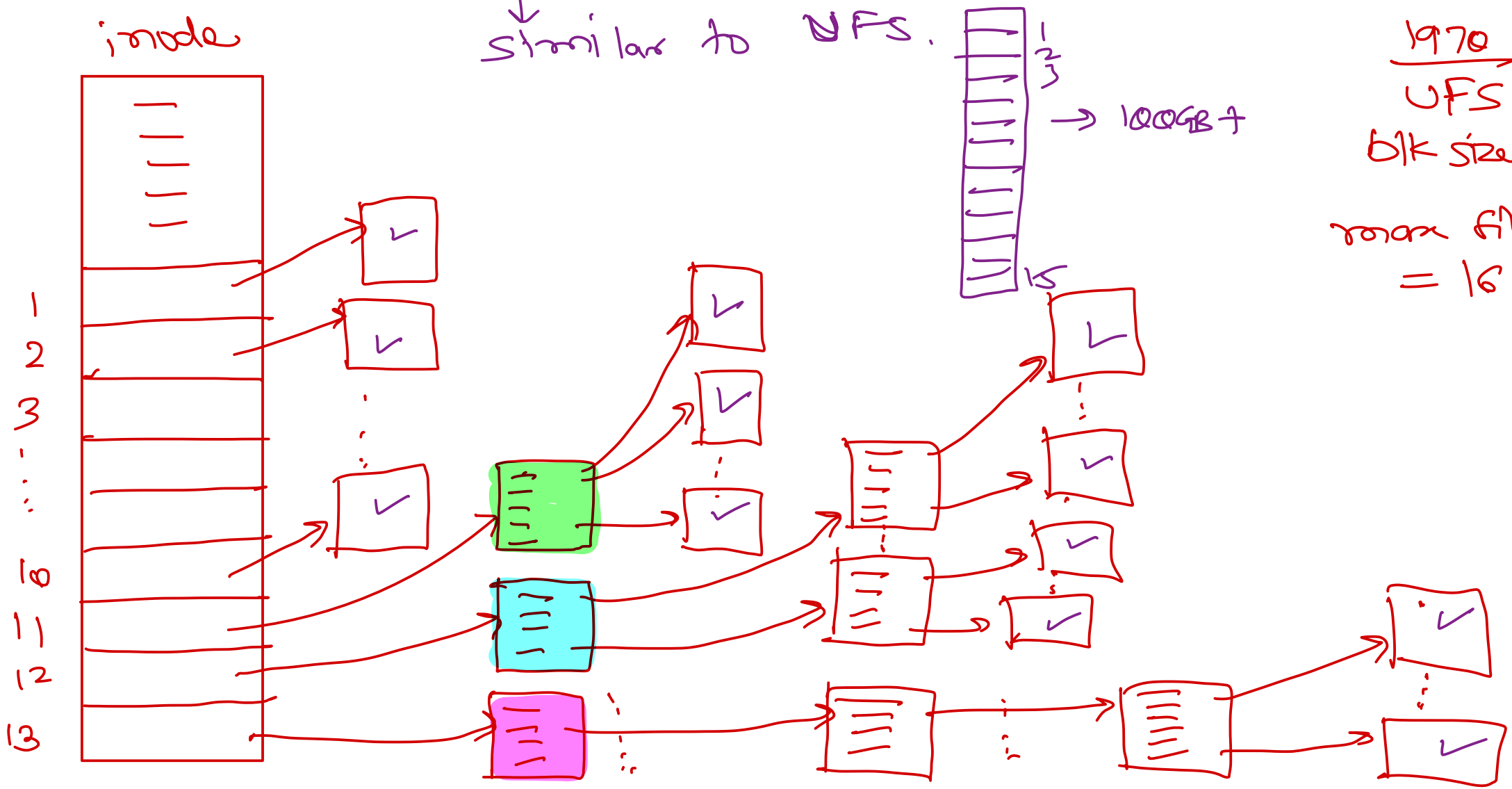
③ indexed alloc.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40

f.txt →



UFS - Ritchie → Ext2/3

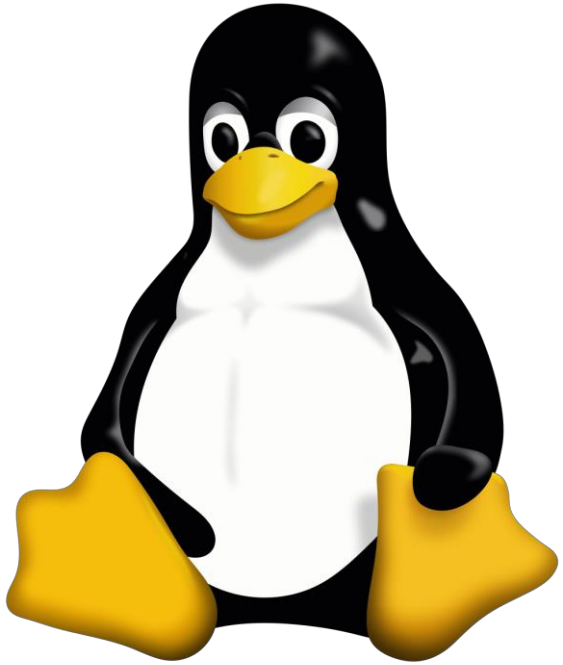


1970
UFS
blk size = 4K

max file size
= 16 GB

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>



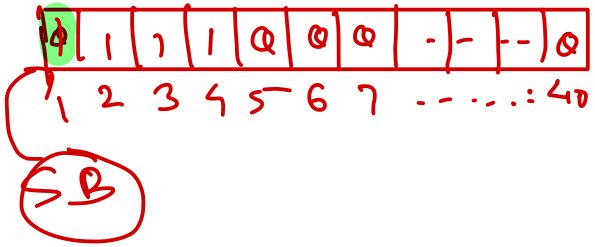
Operating System – Linux Programming

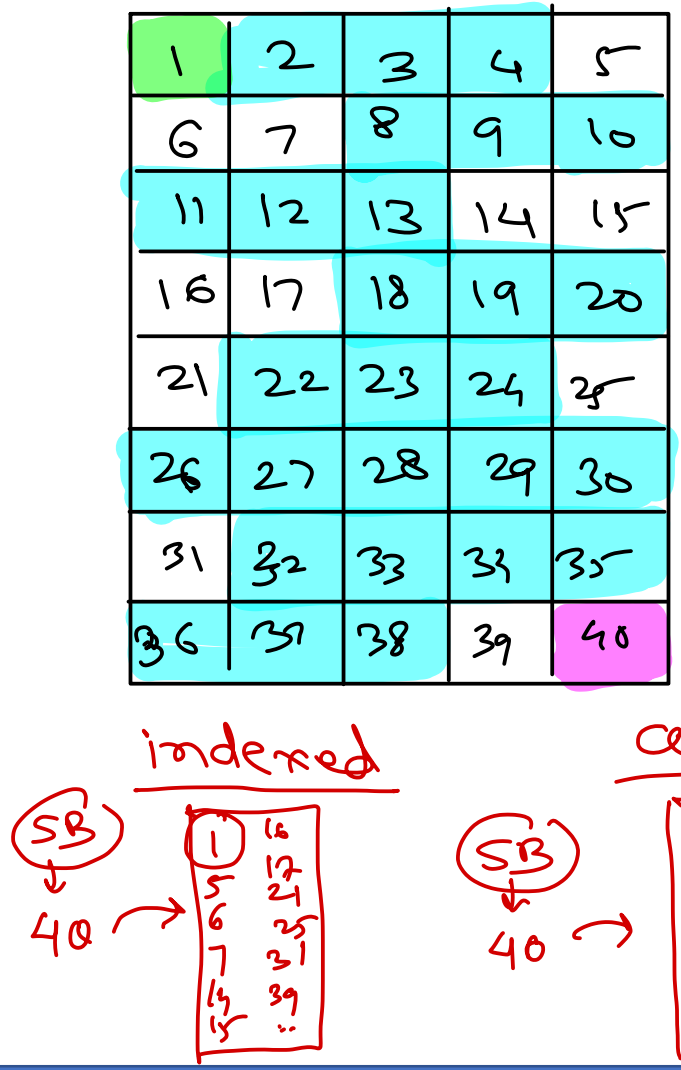
Sunbeam Infotech

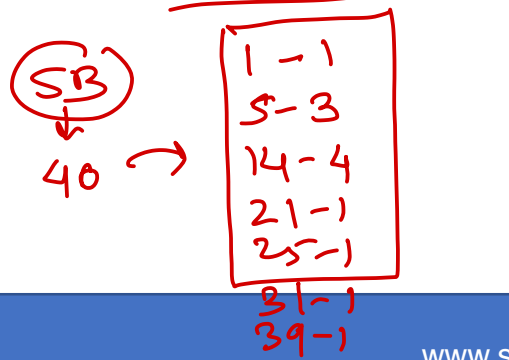
Free Space Mgmt Algo → super block maintain info about free inode / data blocks.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40

bit vector/map





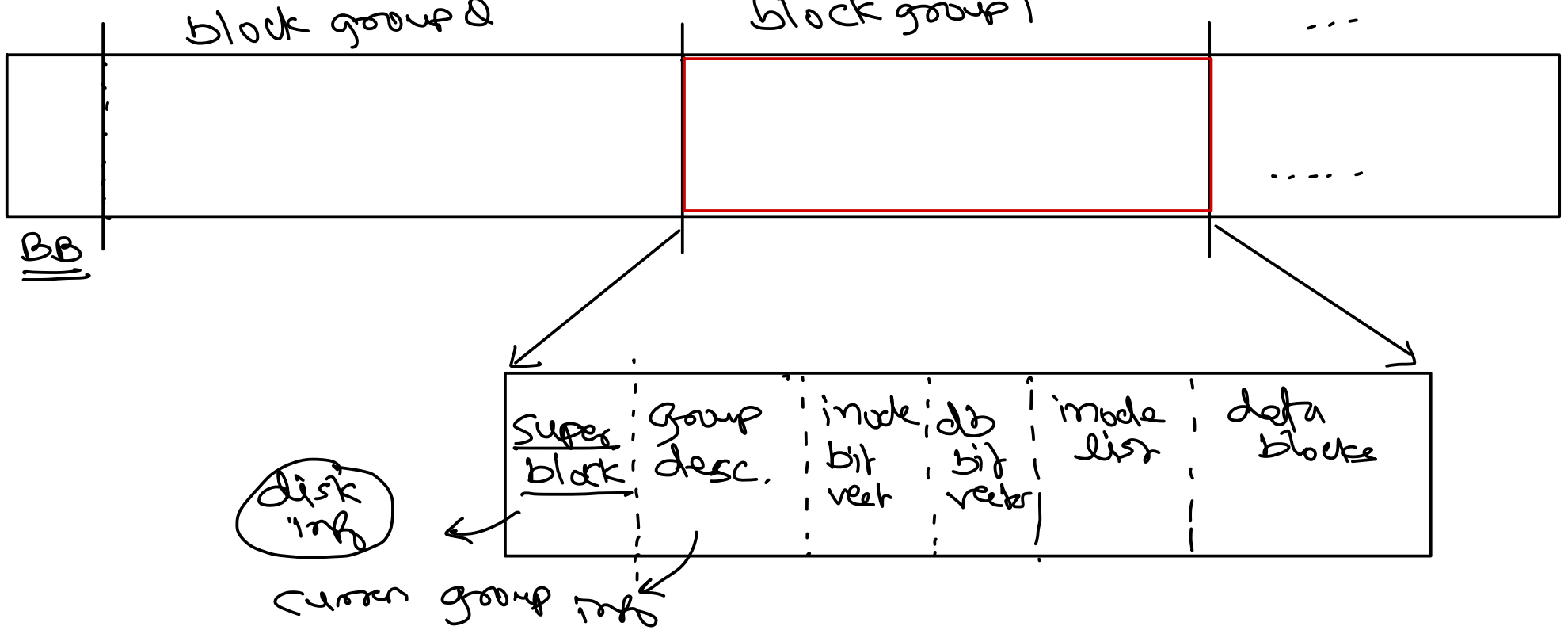


Ext 2/3

FS is created by formatting tool.

Linux: mkfs.

terminal> sudo mkfs -t ext3 /dev/sdb1



ext3
= ext2
+ journaling

* File system checks
  → fsck /dev/sdb1
  check all dentries, inodes & datablocks.
  This slow process
* Journaling speed up checks by limiting check to currently opened file & dirs.

Disk scheduling → IO Subsystem.

Disk access time
= rotational latency
+ seek time.

(5) LOOK → implementation policy of SCAN/C-SCAN to conserve power.

pending :
Cyl rw req 80 20 90 10 40 65 5 35

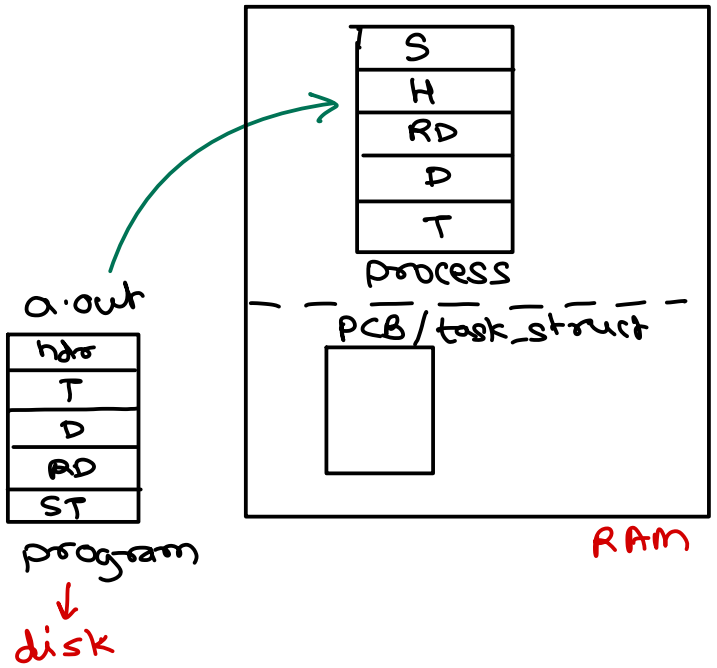
(1) FCFS → 30 → 80 → 20 → 90 → 10 → 40 → 65 → 5 → 35
total seek distance = 50 + 60 + 70 + 80 + 30 + 25 + 60 + 30 = 395

(2) SSTF → 30 → 35 → 40 → 20 → 10 → 5 → 65 → 80 → 90
T.S.D. = 5 + 5 + 20 + 10 + 5 + 60 + 15 + 10 = 130

✓ (3) SCAN / Elevator → 30, 35, 40, 65, 80, 90, 100, 20, 10, 5

(4) Circular SCAN → 30, 35, 40, 65, 80, 90, 100/0, 5, 10, 20.

Process Mangement



PCB contains
1. pid
2. exit status
3. sched info
   - a. priority
   - b. time quantum
   - c. time left
4. memory info
   - a. segments
   - b. page table
5. files info
   - a. open file desc table
   - b. cur directory
6. execution ctx
7. Kernel stack

OS - data structures for process execution.
1. job queue / process table
   - all processes in systems
2. ready queue / run queue
   - all processes ready to execute on cpu.
3. waiting queue(s)
   - wait queue per device / sync obj.

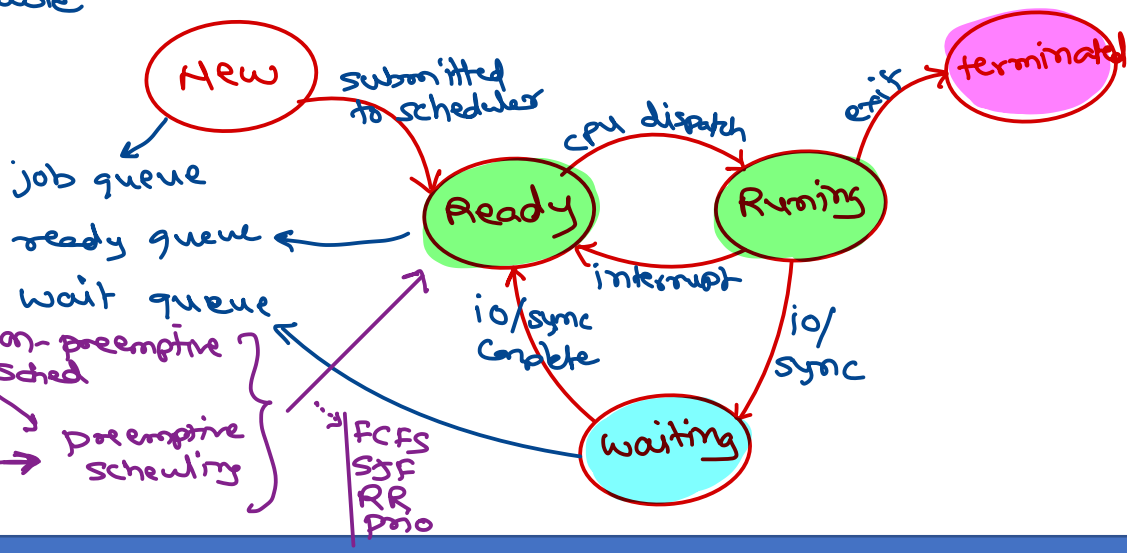
CPU Scheduler - decides next process to be executed on cpu.

CPU dispatcher - dispatch the process on CPU i.e. load its execution context (CPU regr values) into CPU.

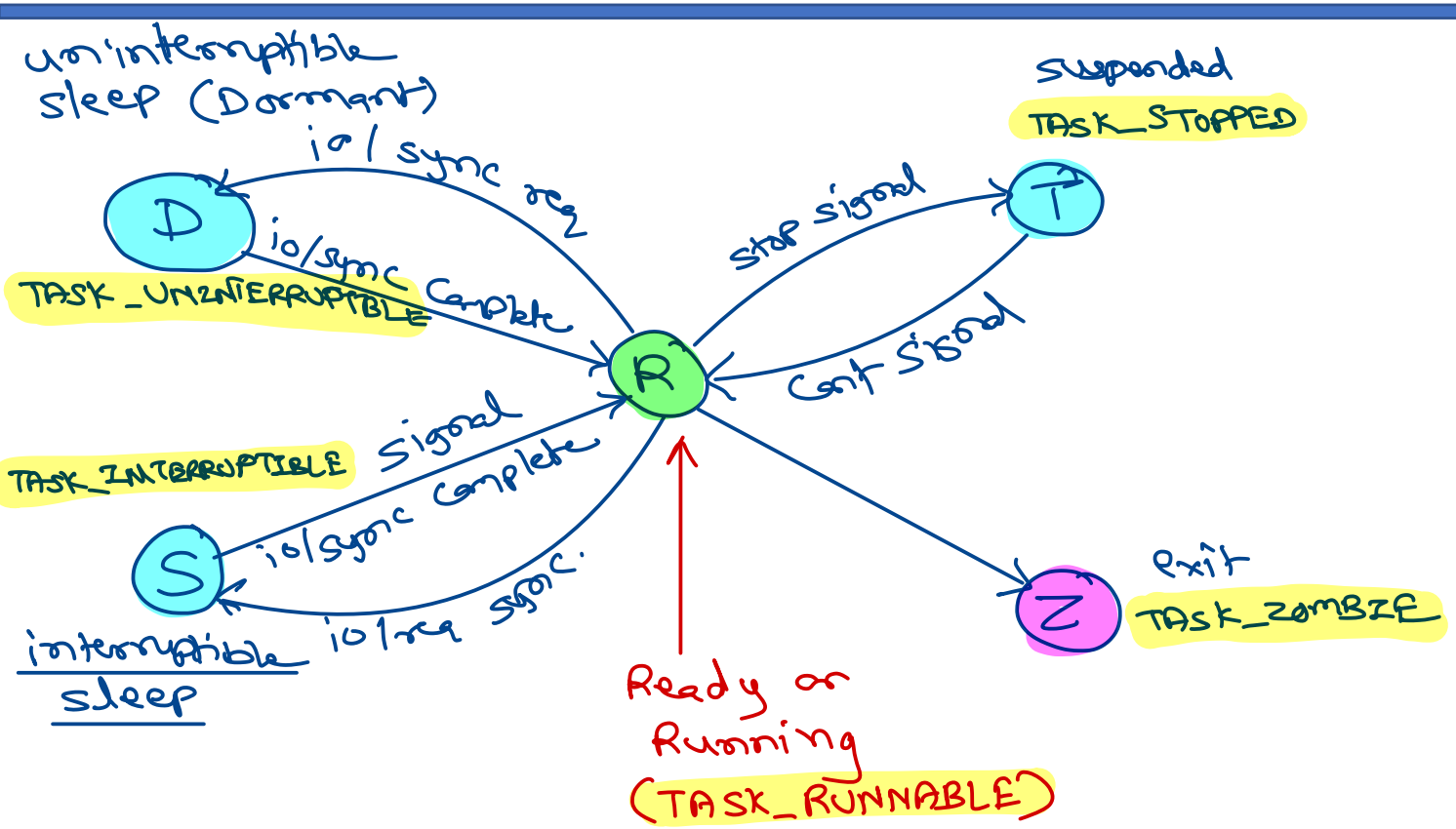
Scheduler
1. Run → T
2. Run → W
3. Run → Rdy
4. W → Rdy

(1), (2) → non-preemptive sched
(3), (4) → preemptive scheduling
FCFS
SJF
RR
Prio

Process Life Cycle



Linux Process States

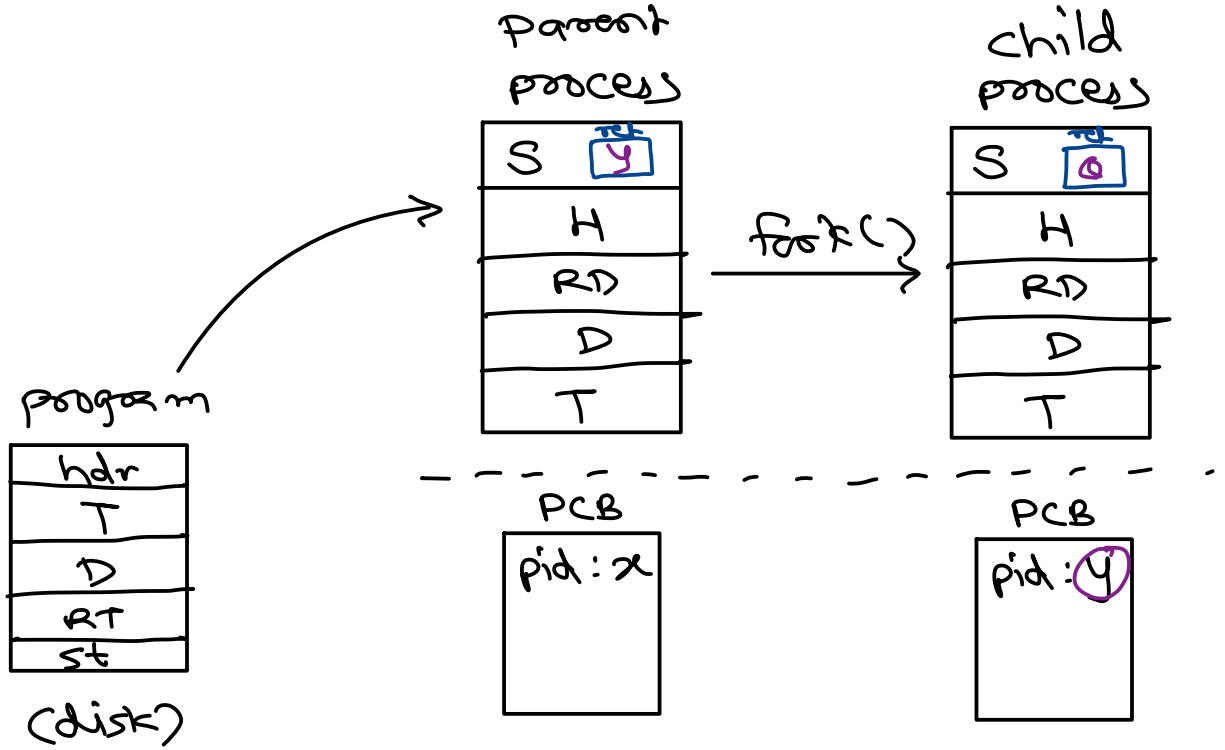


terminal> ps -e -o pid, state, cmd

Process Creation

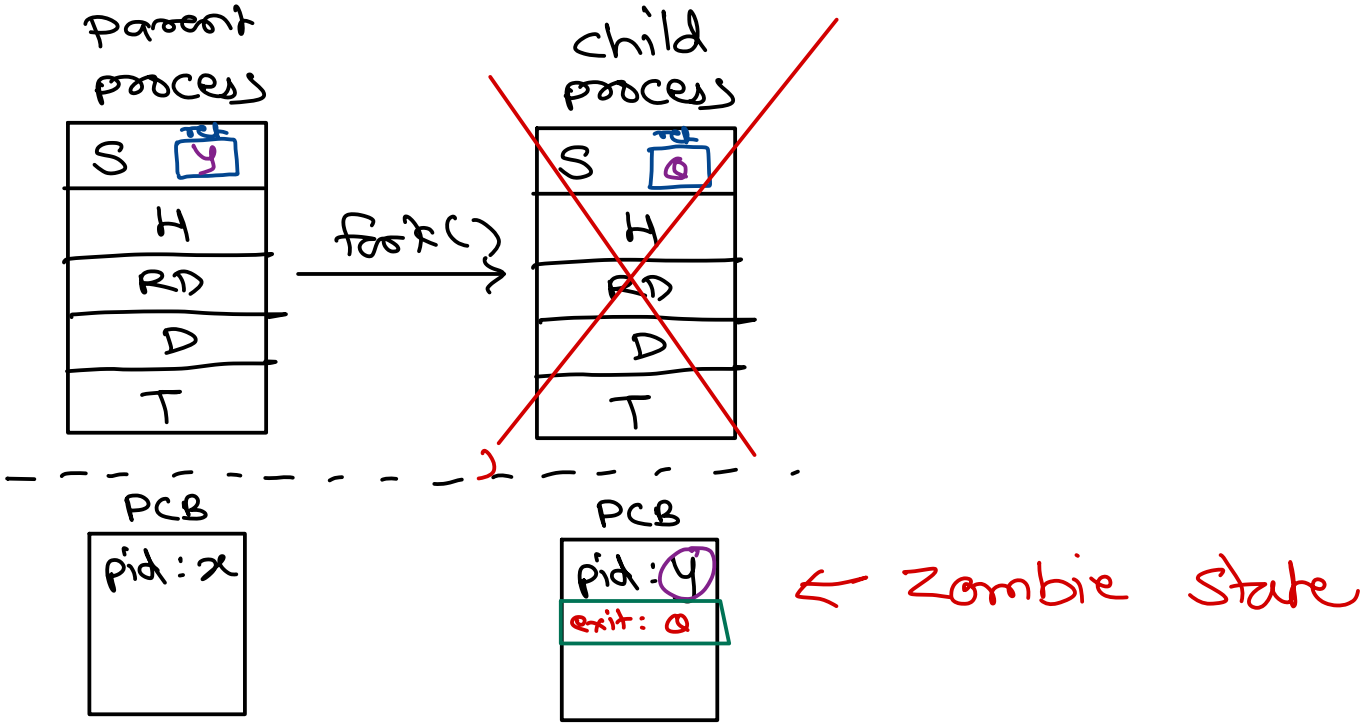
OS - syscall

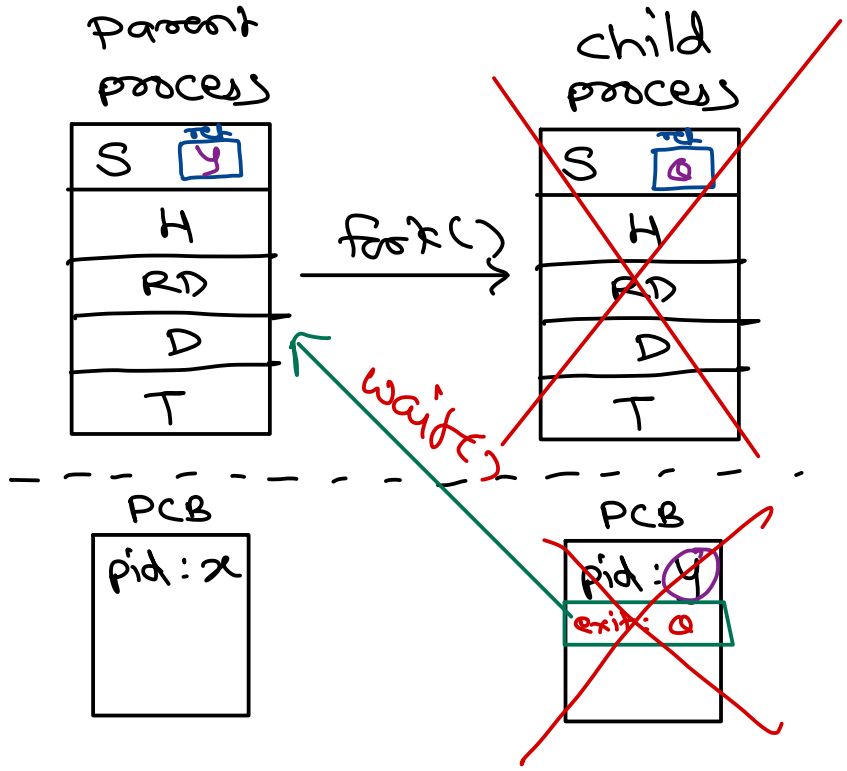
windows → CreateProcess()
UNIX → fork()
BSD UNIX → fork(), vfork()
Linux → fork(), vfork(), clone()




```
main() {
    int ret;
    ret = fork();
    pf("%d\n", ret);
    if(ret == 0) {
        //child proces
    }
    else {
        //parent proces
    }
}
```

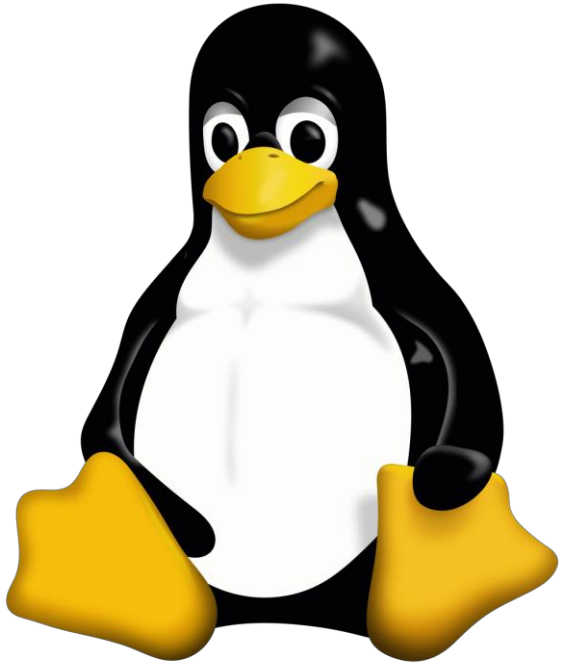
child & Parent


parent process
child process
S
H
RD
D
T
fork()
pid : x
pid : y
exit : 0
PCB
PCB
← Zombie State


parent process
child process
S
H
RD
D
T
fork()
wait()
pid : x
pid : y
exit : 0
PCB
PCB

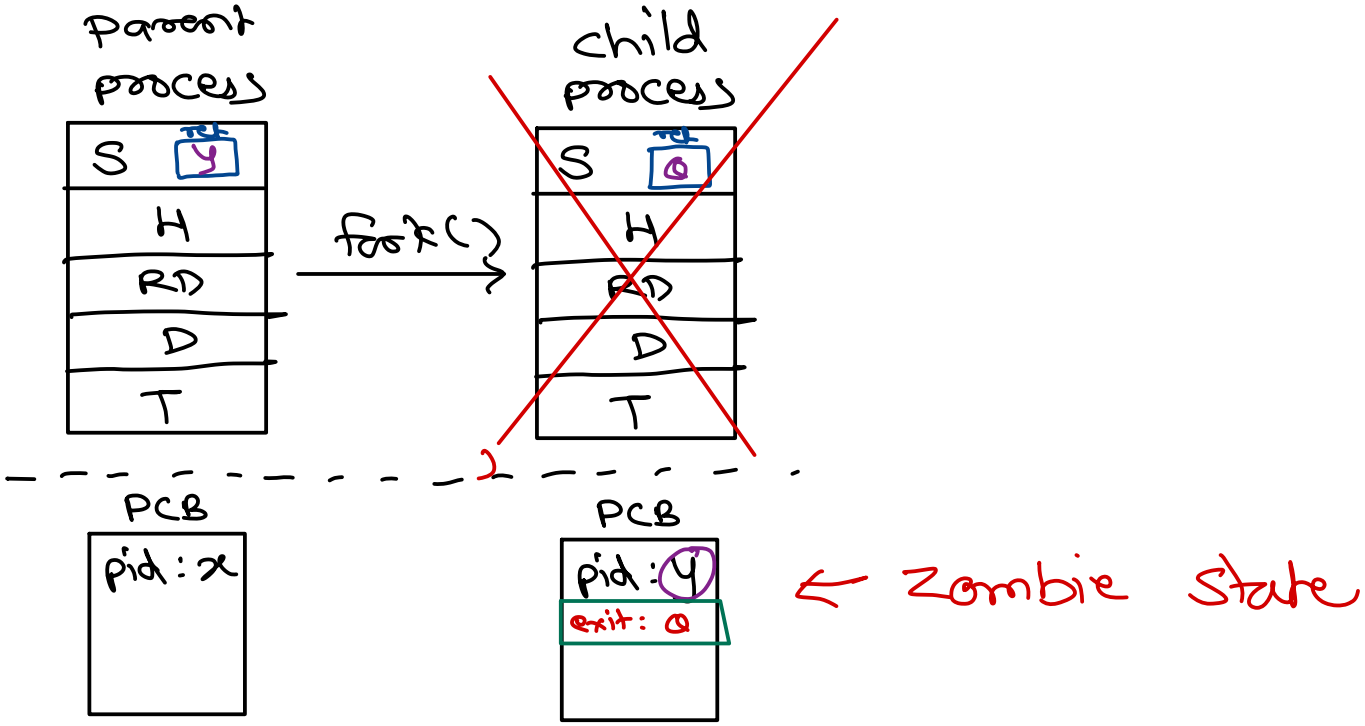
Thank you!

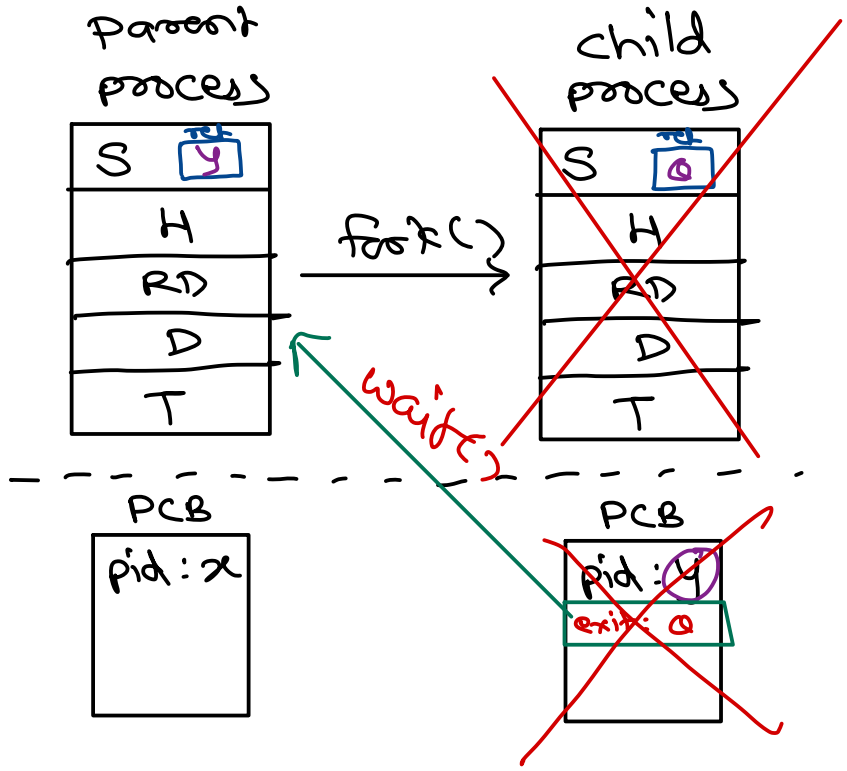
Nilesh Ghule <nilesh@sunbeaminfo.com>



Operating System – Linux Programming

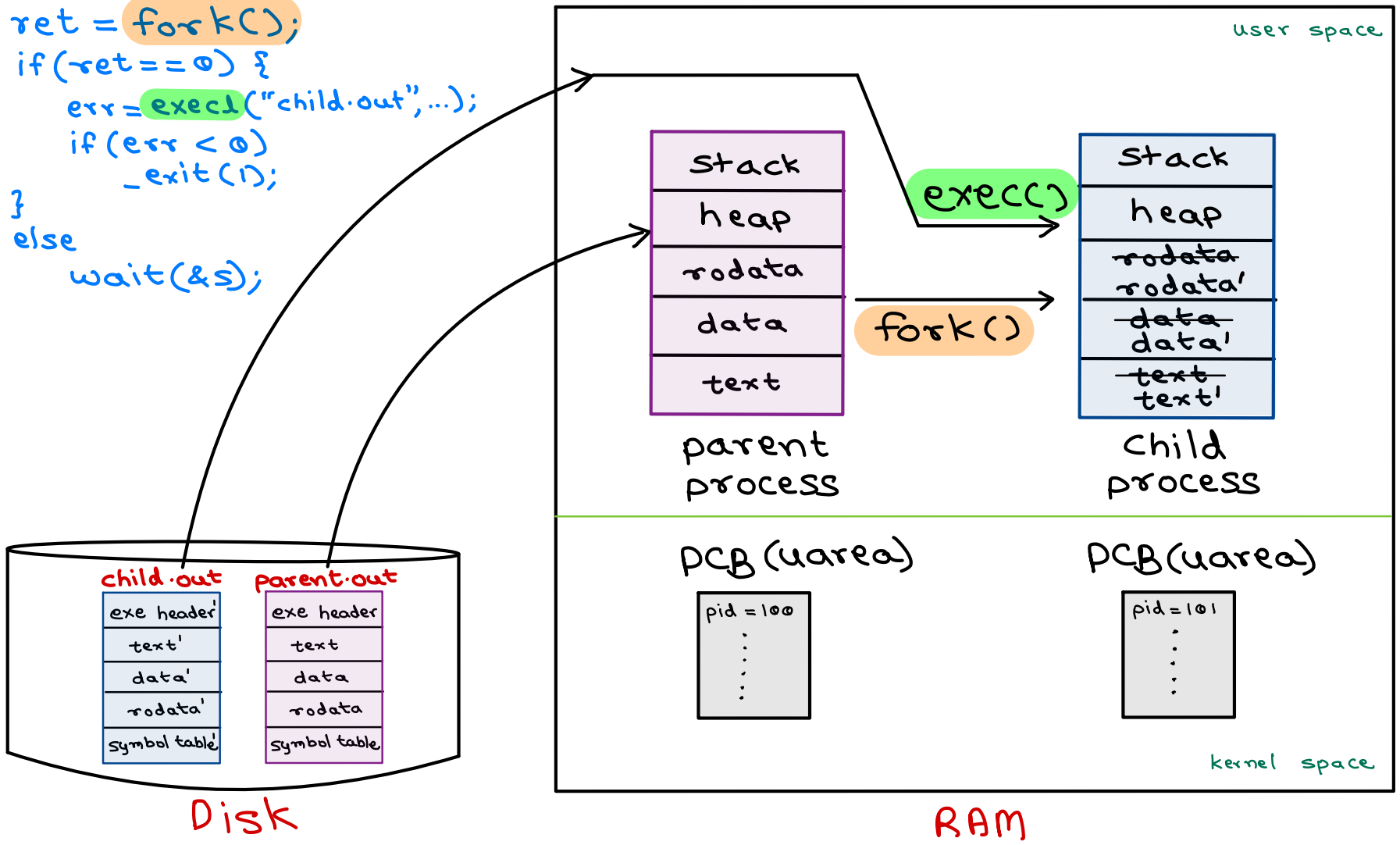
Sunbeam Infotech


parent process
child process
S
H
RD
D
T
fork()
pid : x
pid : y
exit : 0
PCB
PCB
← Zombie State


parent process
child process
S
H
RD
D
T
fork()
wait()
pid : x
pid : y
exit : 0
PCB
PCB

exec() syscall.

```
ret = fork();
if (ret == 0) {
    err = execl("child.out",...);
    if (err < 0)
        _exit(1);
}
else
    wait(&s);
```



exec() family

- execl()
- execlp()
- execle()
- execv()
- execvp()
- execvpe()

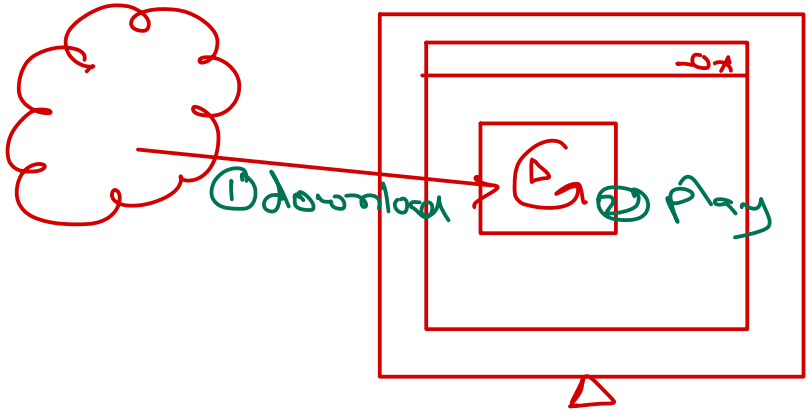
(library fn (3))

execve()
↳ syscall (2)

- l → var arg list → cmdline args.
- v → arg vector/arrays → cmdline args.
- p → child program will be searched in all dirs mentioned in PATH.
- e → env vars to be given to child.

```
main(argc, argv, envp){
}
```

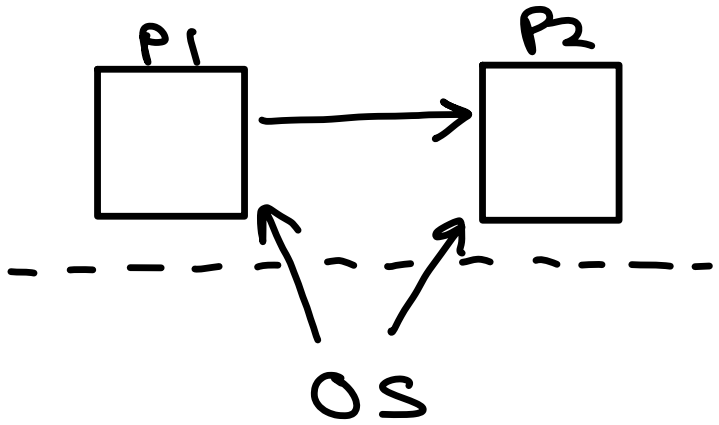
IPC



Linux IPC

1. signals
2. shared memory
3. message queue
4. pipe
5. socket

signals



if signal is not handled, default action take place.

1. TERM
2. CORE
3. STOP
4. CONT
5. IGN

} terminal> man 7 signal

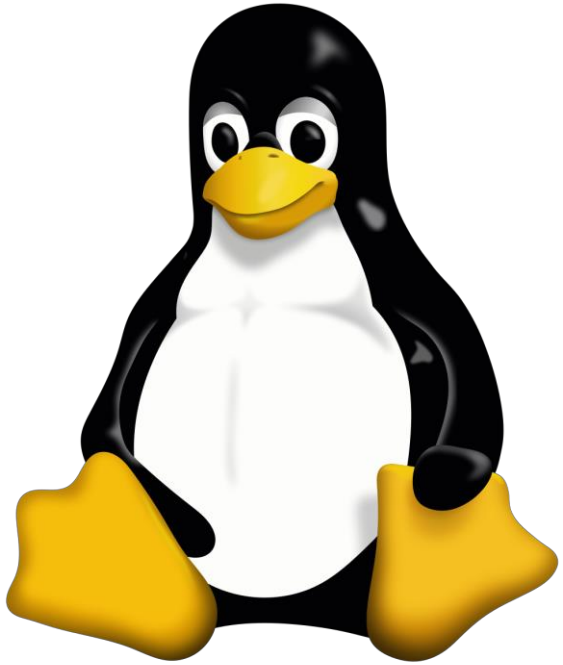
terminal> kill -l

Imp Signals

2. SIGINT → ctrl+C (term).
15. SIGTERM → } OS shutdown (term).
9. SIGKILL → } OS shutdown (term).
19. SIGSTOP → ctrl+S (stop)
18. SIGCONT → ctrl+q (cont)
11. SIGSEGV → seg violation (core) (dangling ptr)
17. SIGCHLD → child send signal to parent while terminating. (ign)

Thank you!

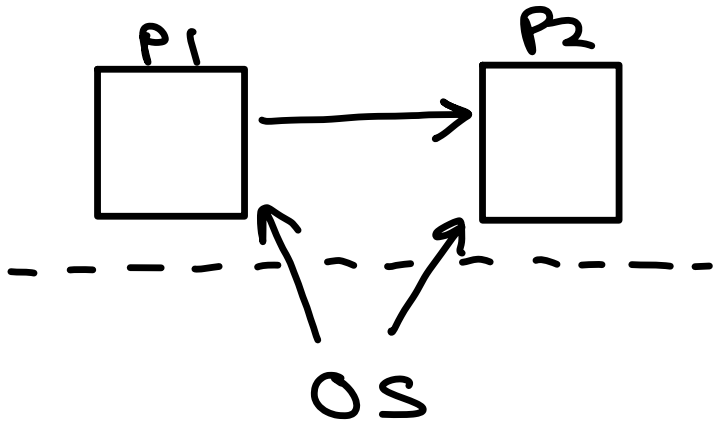
Nilesh Ghule <nilesh@sunbeaminfo.com>



Operating System – Linux Programming

Sunbeam Infotech

signals



if signal is not handled, default action take place:

1. TERM
2. CORE
3. STOP
4. CONT
5. IGN

} terminal > man 7 signal

terminal > kill -l

Imp Signals

2. SIGINT → ctrl+C (term).
15. SIGTERM → } OS shutdown (term).
9. SIGKILL → } (term).

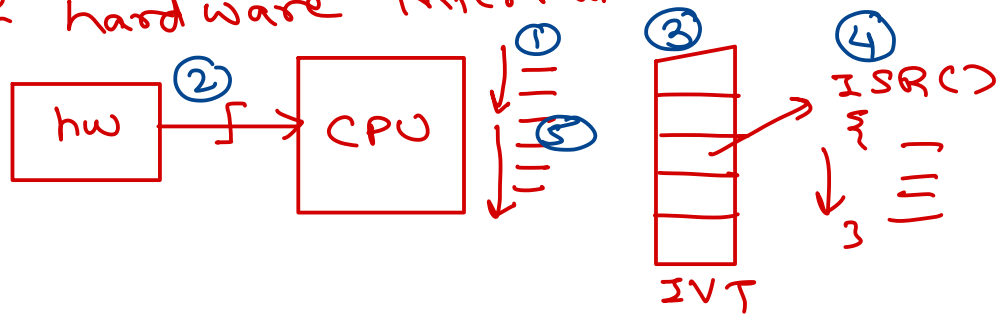
cannot be handled

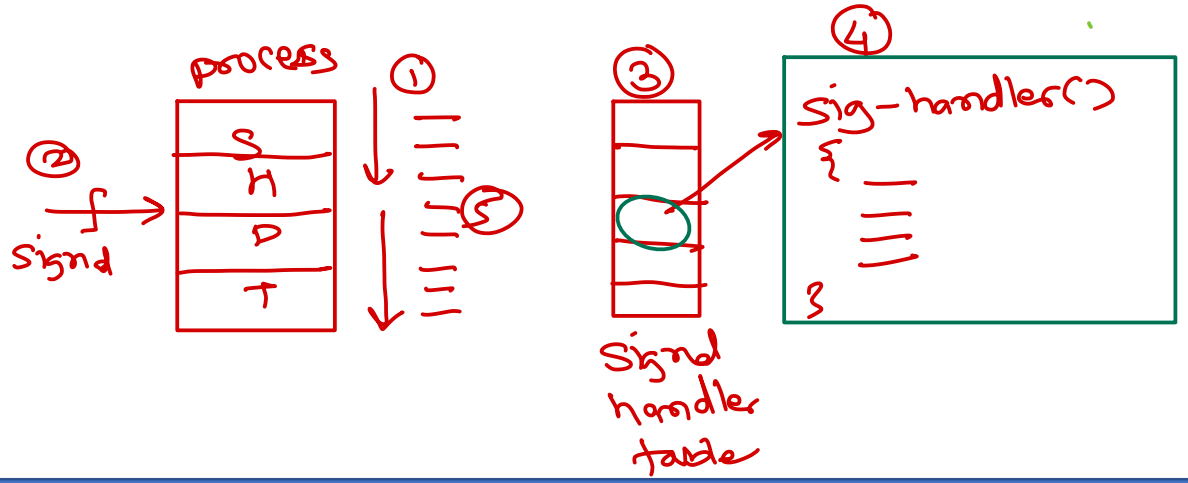
19. SIGSTOP → ctrl+S (stop)
18. SIGCONT → ctrl+q (cont)
11. SIGSEGV → seg violation (core) (dangling ptr)
17. SIGCHLD → child send signal to parent while terminating. (ign)
1. SIGHUP → hang up - parent terminal is closed. (term)

terminal >
kill -signum pid
kill -signame pid
pkill -signum program name
pkill -signame program name
killall -

* kill -signum pid ← syscall
  ↳ kill(pid, signum);
    ↳ man 2 kill

* signal is software counter part of hardware interrupt.

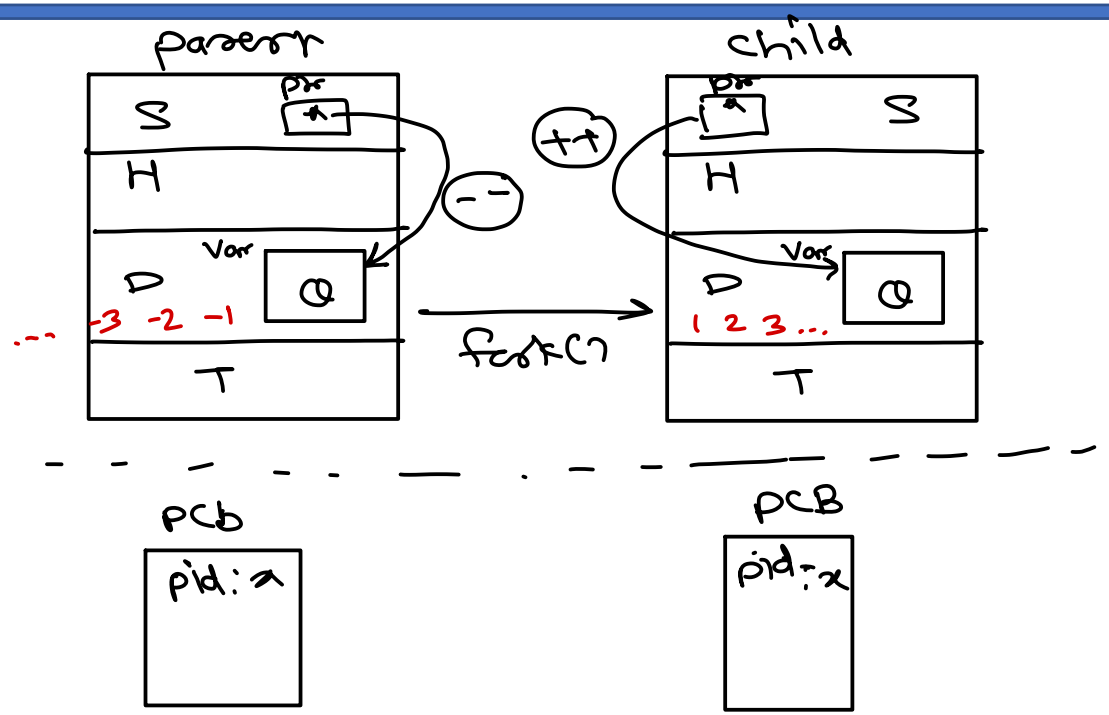


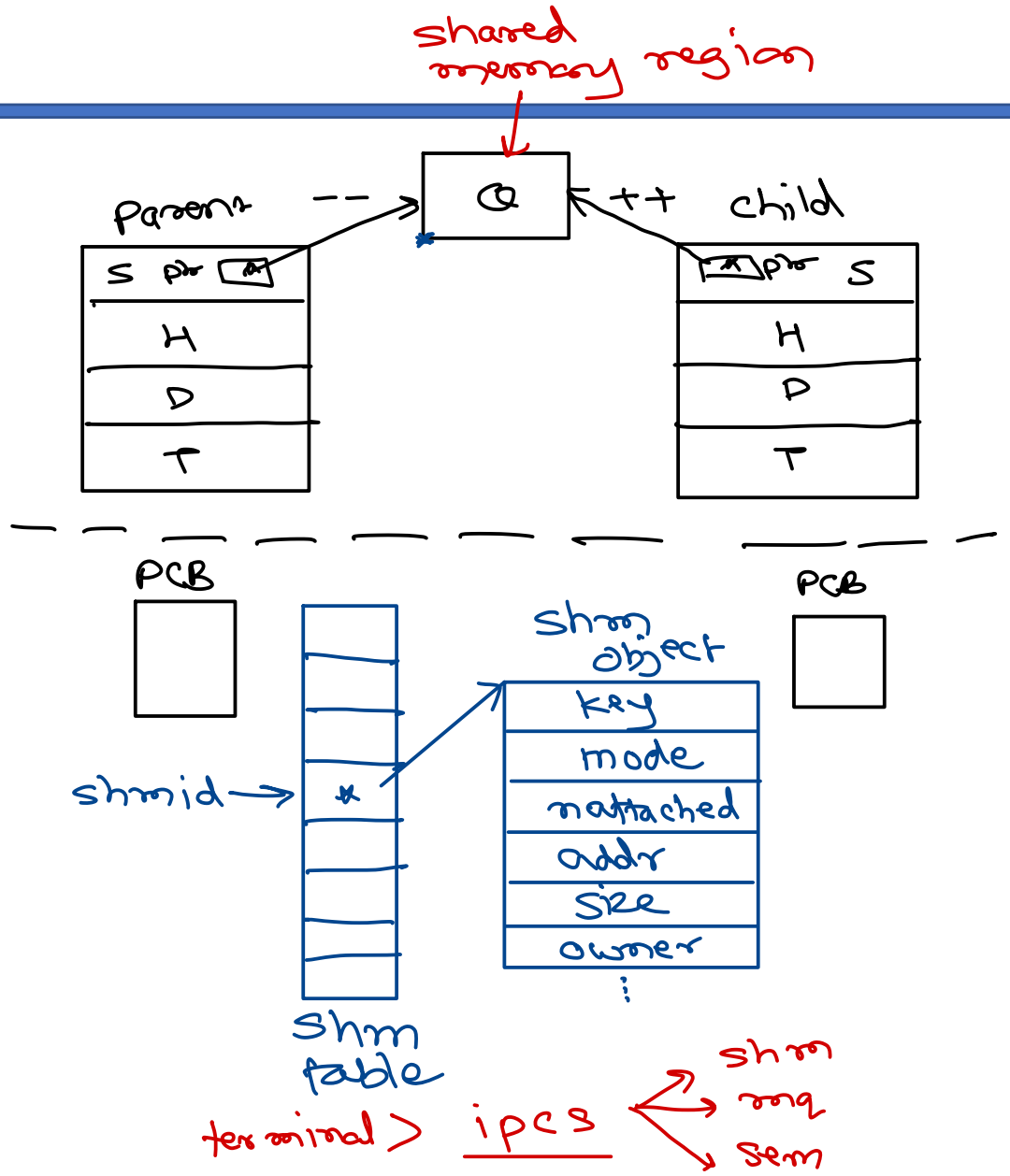


UNIX
- signal() → register signal handler for given signal. Internally it make its entry into process's (PCB) signal handler table.
  It returns old sig handler addr.

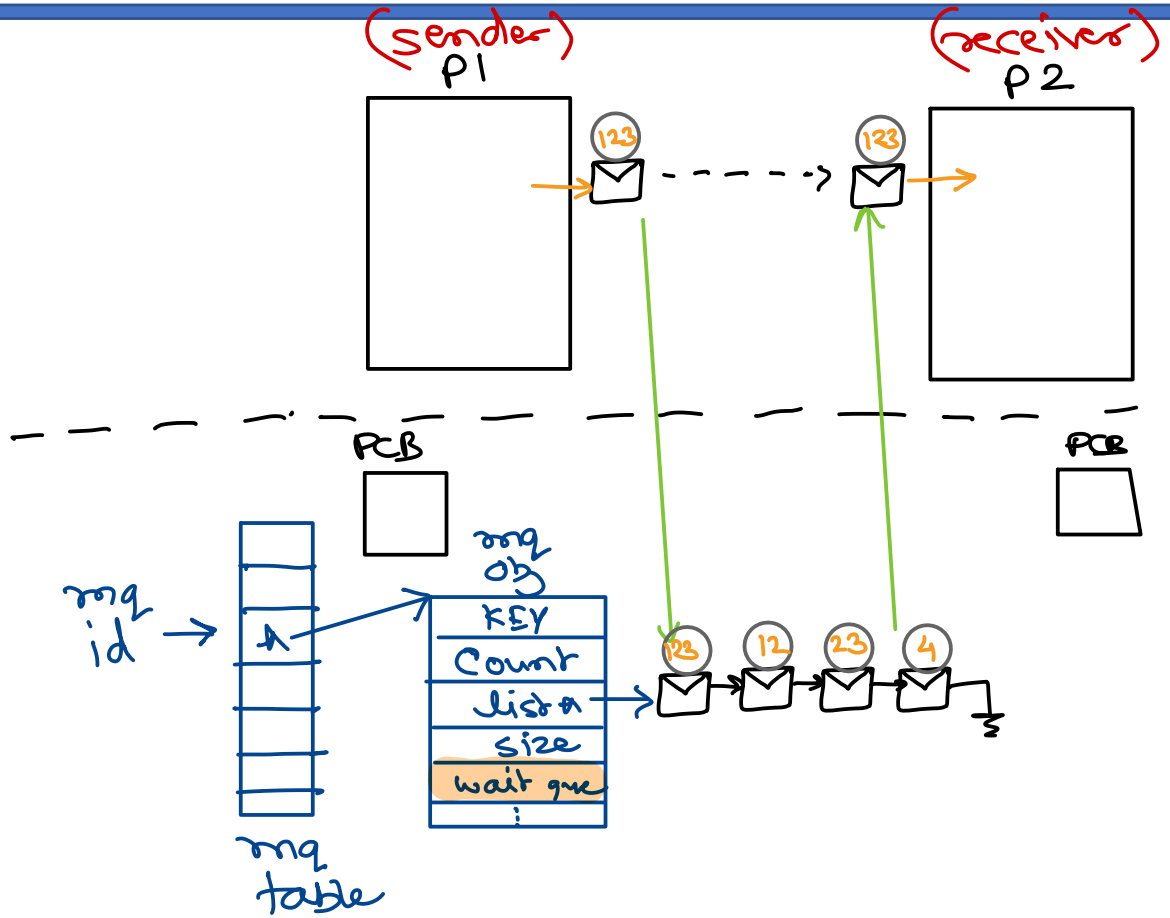
Linux
- sigaction() → register sig handler. enhanced signal() sys call.
- signal handlers are typically registered at the start of appln.

Shared memory

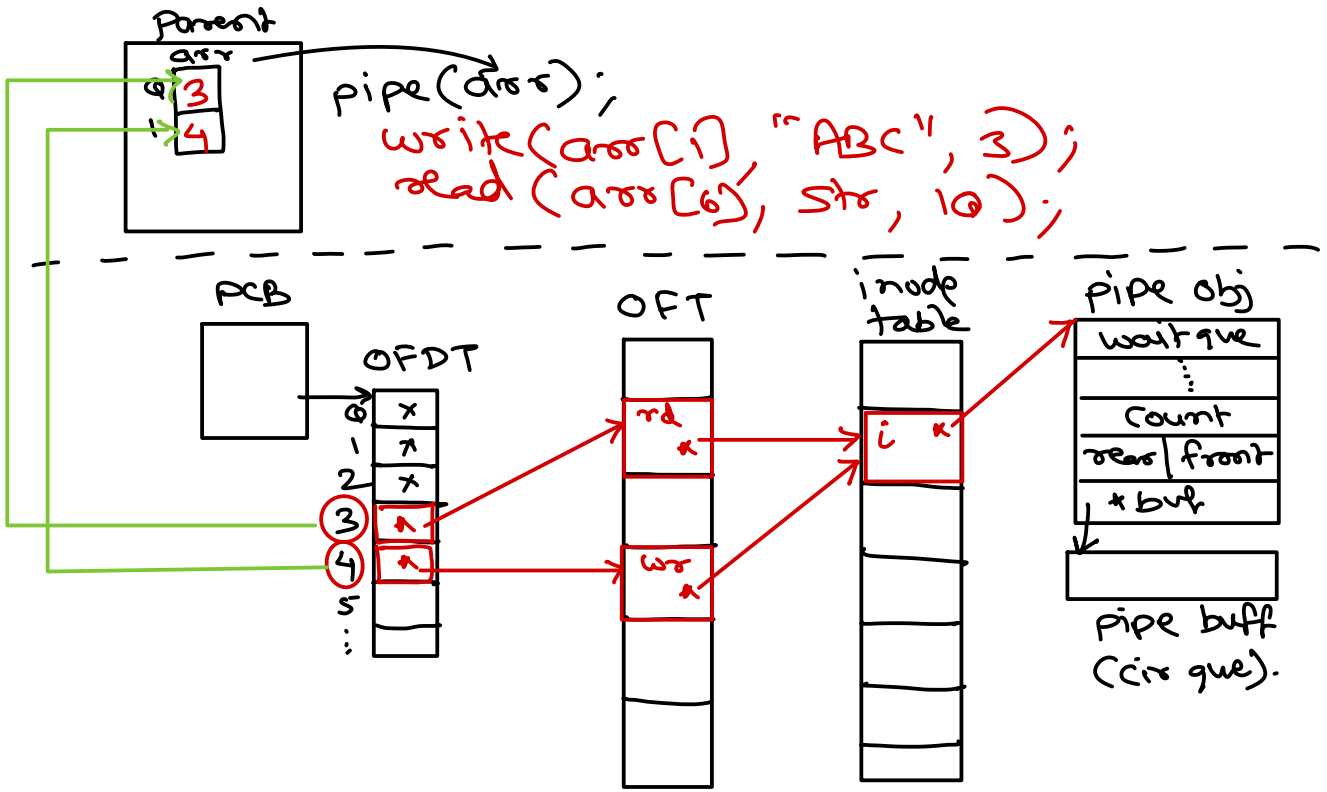


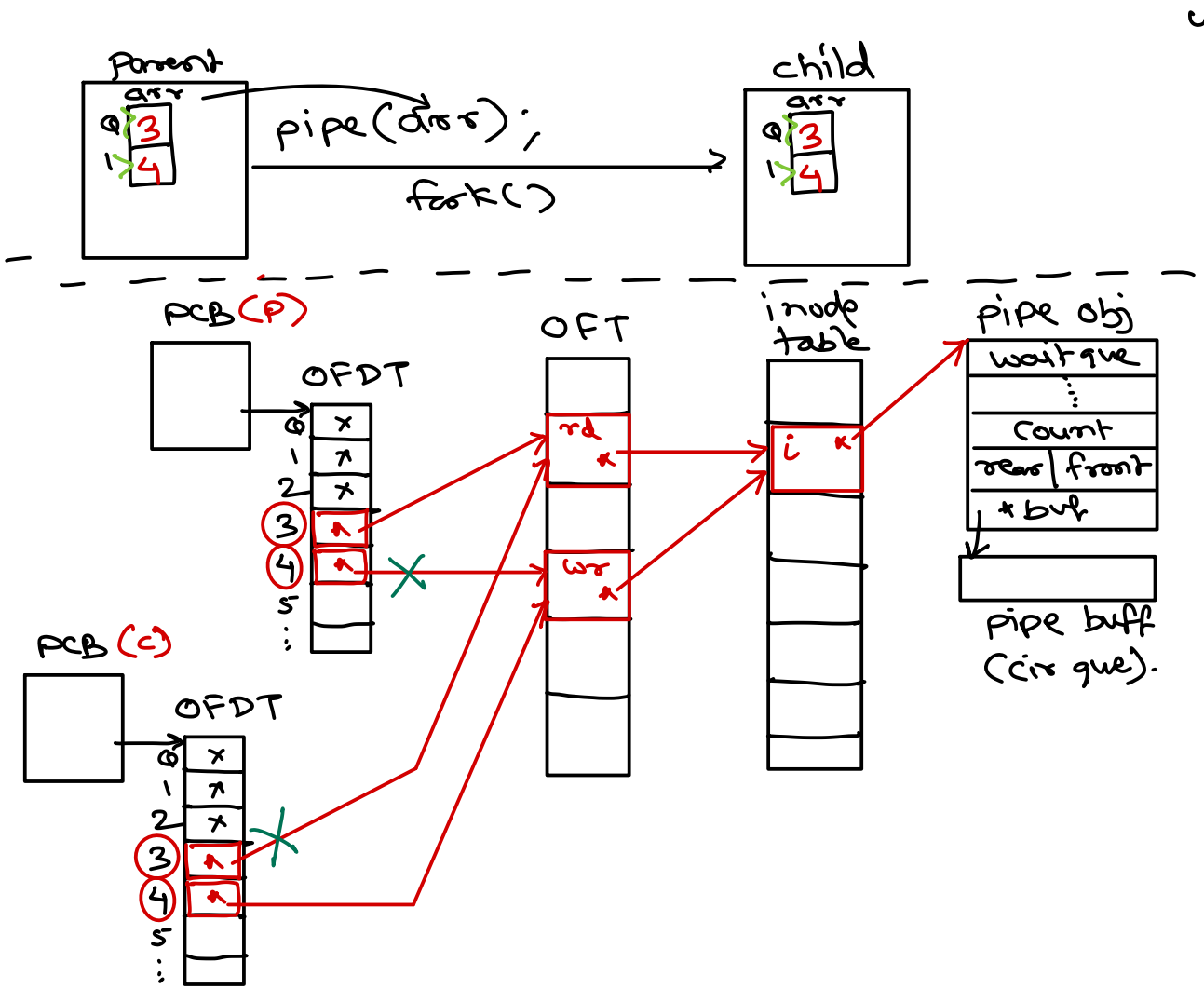


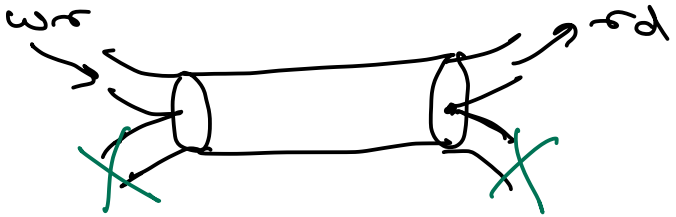
message queue

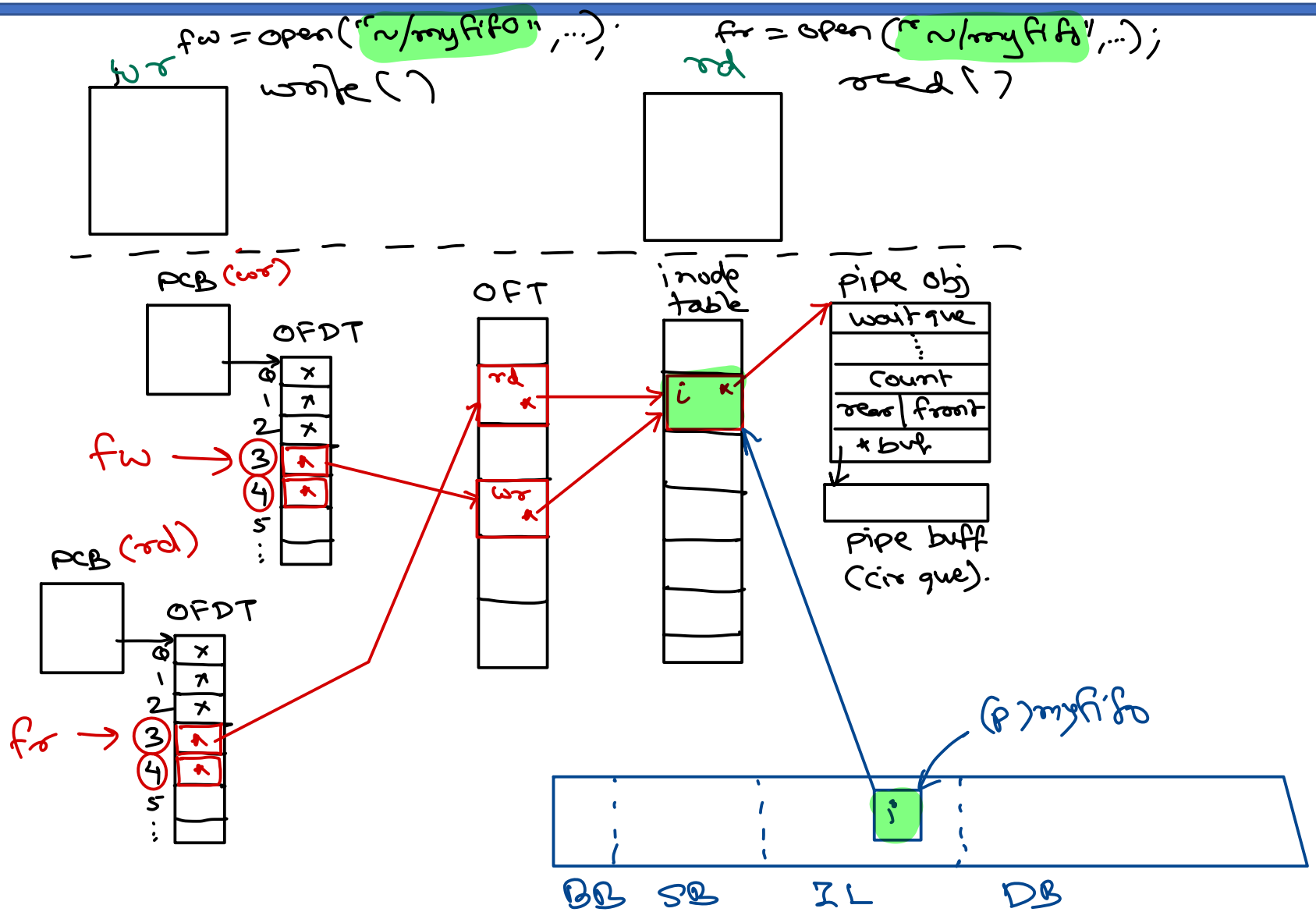


pipe()


Parent
arr
0
1
3
4
pipe(arr);
write(arr[1], "ABC", 3);
read(arr[0], str, 10);
PCB
OFDT
OFT
inode table
pipe obj
waitque
count
rear
front
*buf
pipe buff
(cir que).
0
1
2
3
4
5
x
x
x
rd
wr
i


parent
arr
0
1
3
4
pipe(arr);
fork()
child
arr
0
1
3
4
PCB(P)
OFDT
OFT
inode table
pipe obj
wait que
count
rear
front
*buf
rd
wr
i
0
1
2
3
4
5
x
PCB (C)
OFDT
pipe buff
(cir que).

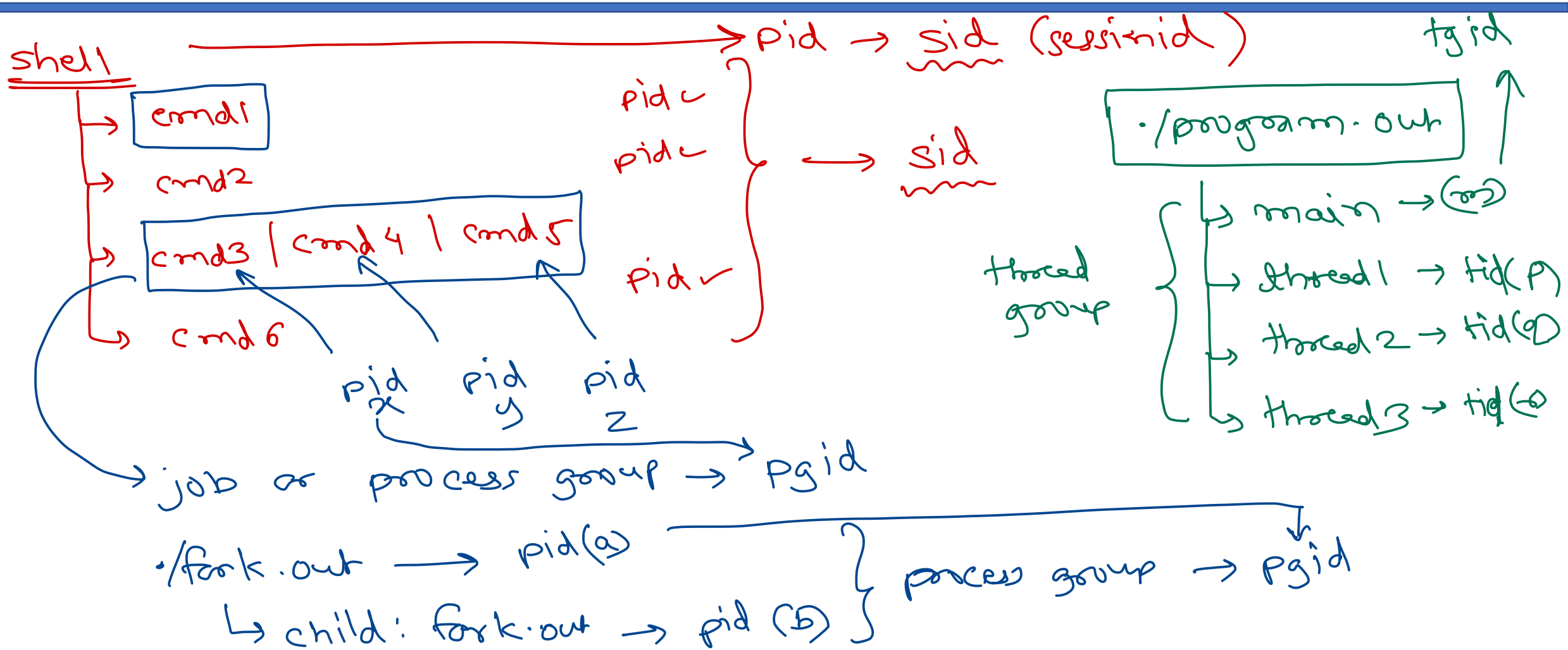

wr
rd


fw = open("n/myfifo",...);
wr
write()
fr = open("n/myfifo",...);
rd
read()
PCB (wr)
OFDT
OFT
inode table
pipe obj
wait que
count
rear
front
*buf
fw
rd
wr
i
pipe buff (cir que).
PCB (rd)
OFDT
fr
(p)myfifo
BB
SB
IL
DB

Sockets

* Socket is communication endpoint.
* Socket developed by BSD UNIX.
* Socket is extension of pipe concept.
    * socket is bi-directional.
* Socket has many types (address families).
    * AF_UNIX
    * AF_INET
    * AF_BLUETOOTH
    * ...

* AF_UNIX
    * Comm betn two processes on same computer.
    * special file (s)
* AF_INET (internet socket)
    * Comm betn two processes on same/diff computers.
    * inet socket = ip address + port number.
        * TCP or UDP


shell
cmd1
cmd2
cmd3 | cmd4 | cmd5
cmd6
pid → sid (sessionid)
pid c
pid c
pid v
sid
pid x
pid y
pid z
job or process group → pgid
./fork.out → pid(a)
child: fork.out → pid (b)
process group → pgid
tgid
./program.out
main → (a)
thread group
thread1 → tid(p)
thread2 → tid(q)
thread3 → tid(r)

Linux runlevels

0 → shutdown

1 → single user mode (min services).

2 → multi user mode

3 → multi user mode + network.

4 → reserved

5 → multi user + network + GUI

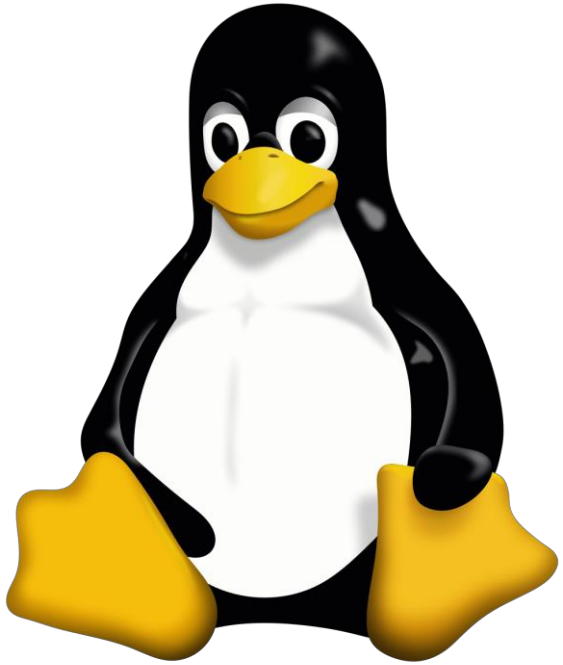
6 → reboot

terminal> sudo init n

terminal> runlevel

Thank you!

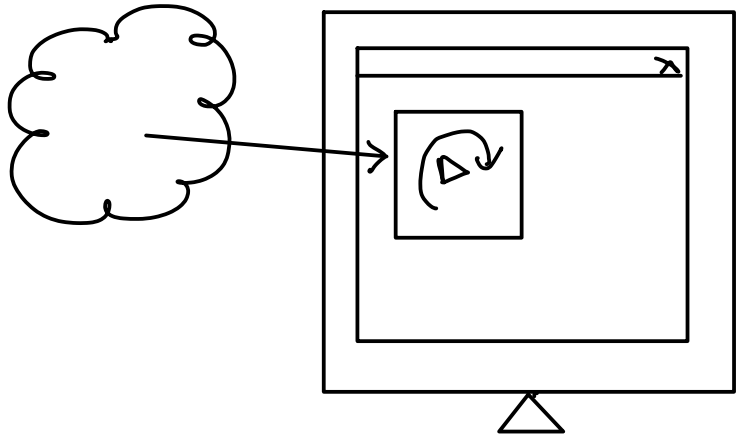
Nilesh Ghule <nilesh@sunbeaminfo.com>

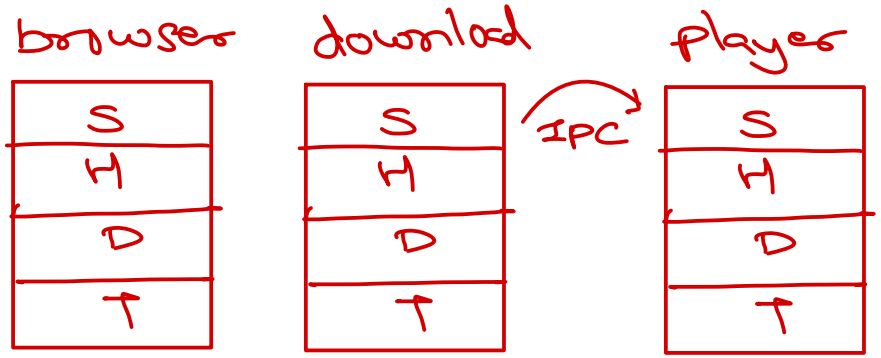


Operating System – Linux Programming

Sunbeam Infotech

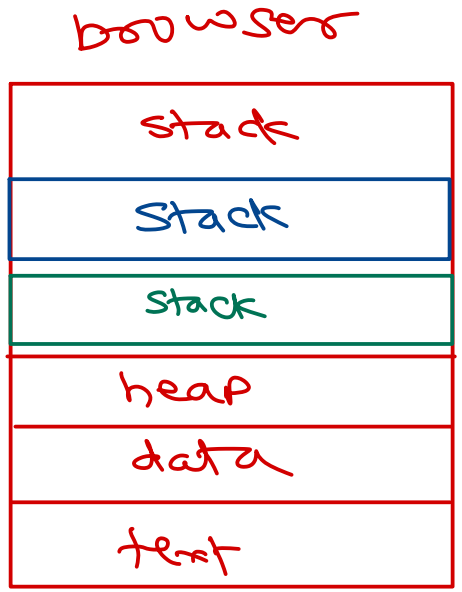
Multi-Threading





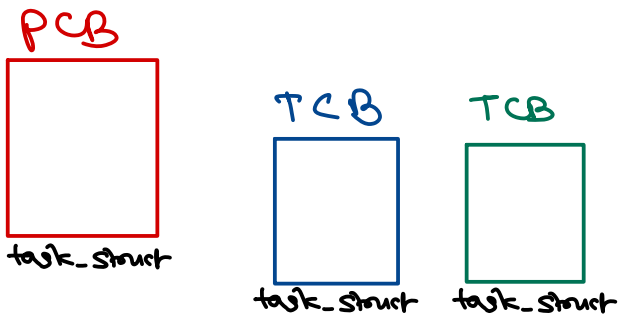
* process based multitasking

* thread based multi-tasking

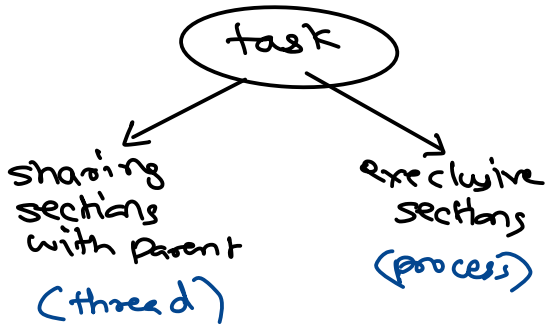


* Thread is light-weight process.
* Thread is unit of execution/scheduling.
* process have a default thread - main thread.

terminal> ps -e -m -o pid,tid,nlwp,cmd



* Linux:

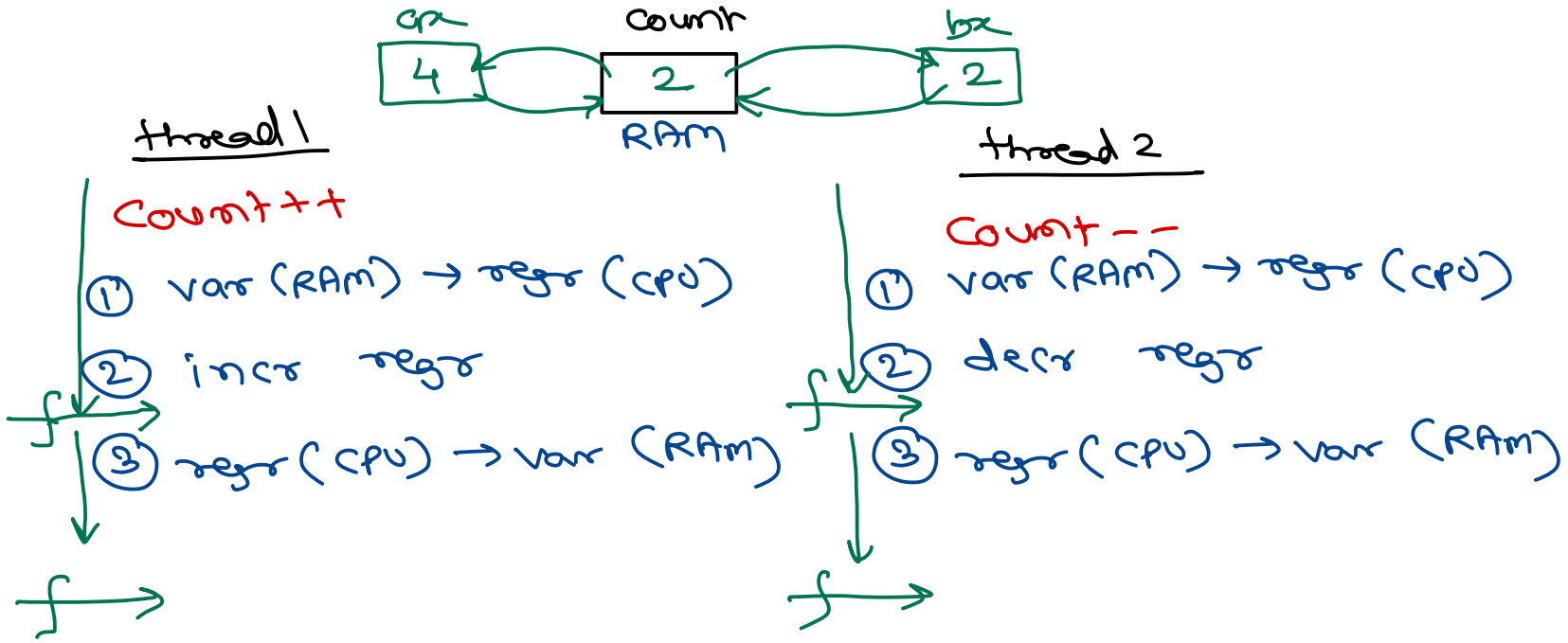


Linux syscall
* clone() → thread
* clone() → process

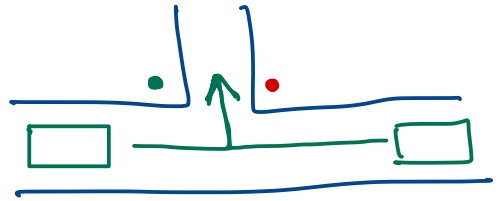
library funs
* win32 : CreateThread()
* POSIX thread library
UNIX - pthread_create()

thread creation
* step1 : implement thread function/procedure
* step2 : call thread creation fn/api

Race condition – peterson's problem



result : corrupted / unexpected.



Synchronization

Allow / block processes / threads so that resource can be accessed safely.

Linux Sync mechanisms

1. Semaphore
2. Mutex
3. Condition variable

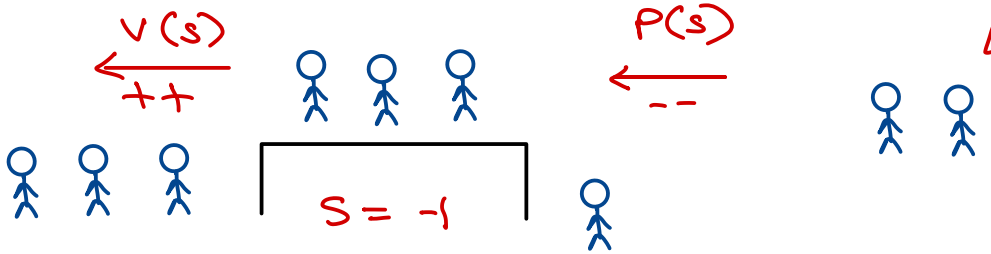
Synchronization

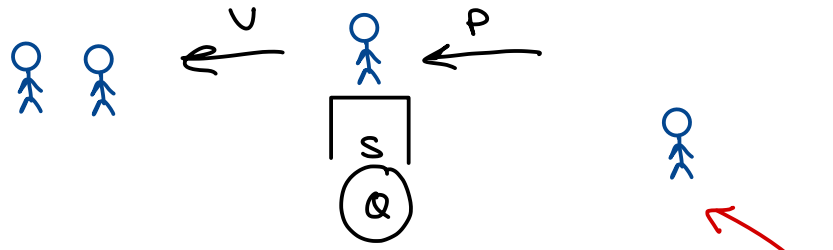
Semaphore — semaphore as counter

- Dijkstra →

V(s) incr op
1. incr sem count.
2. if one/more process-es are blocked on sem, wake up one of the poress.

decr op - P(s)
1. decr sem count.
2. if cnt < 0, block calling procer.



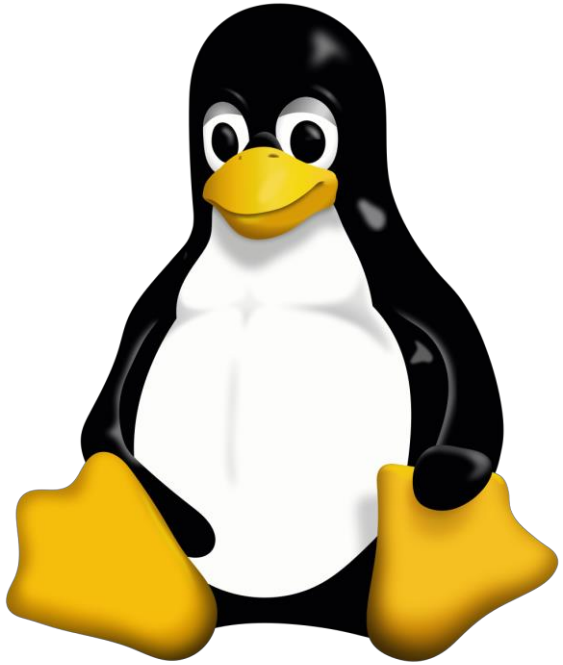


Semaphore types
* counting semaphore
* binary semaphore

Mutex - mutual exclusion
- mutex is light weight than bin sem.
- mutex → lock
  unlock
- locking process is owner of mutex & only owner can unlock.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

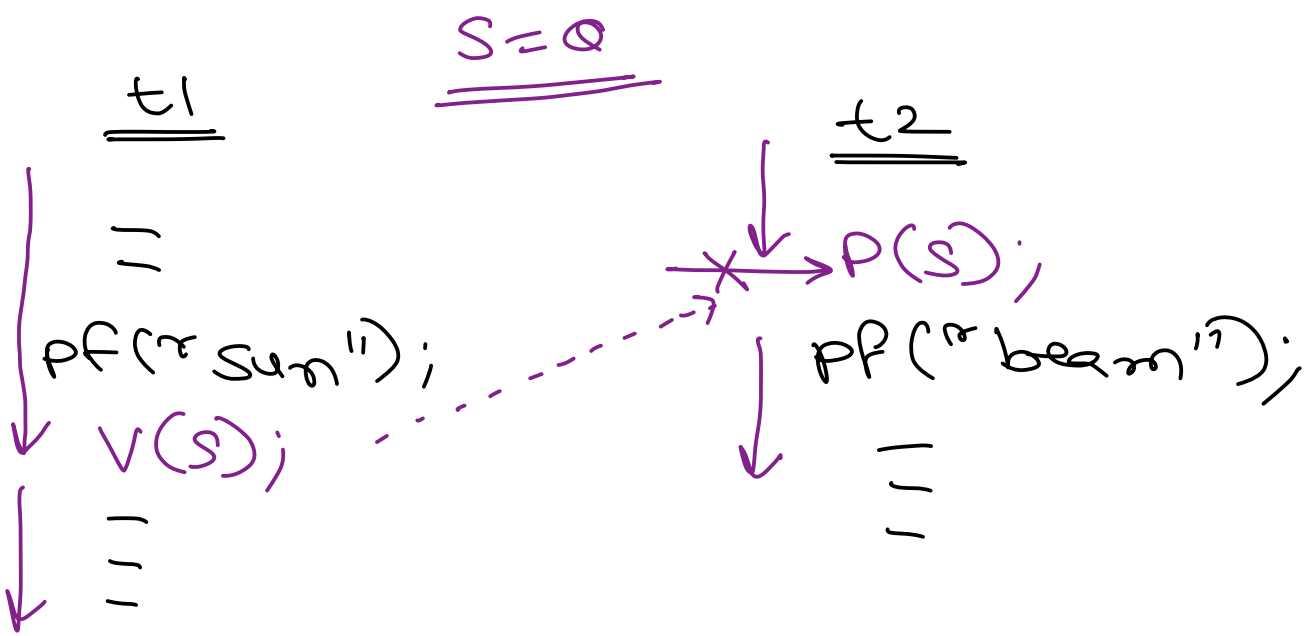


Operating System – Linux Programming

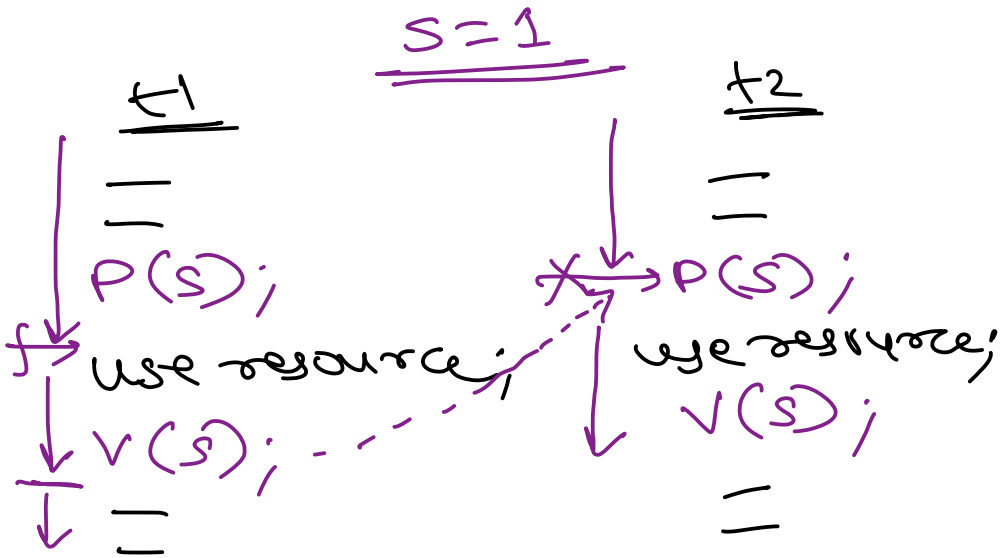
Sunbeam Infotech

Semaphore / Mutex

① semaphore as a flag/event



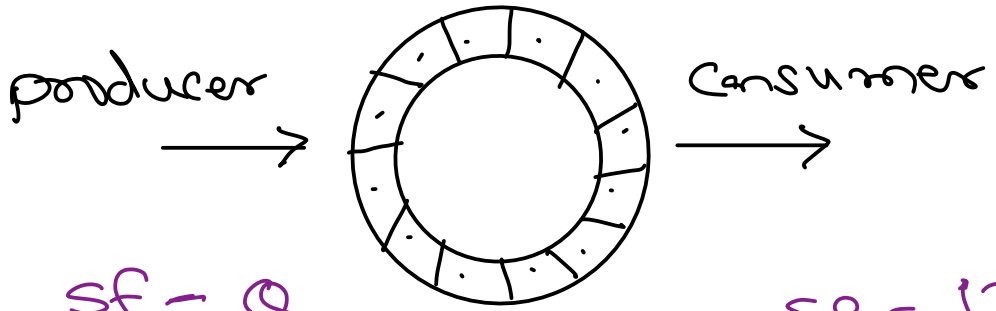
② semaphore for mutual exech



① condition variable
② binary semaphore

① mutex
② binary semaphore

③ Sem for counting.



sf = 0

se = 12

P(se);

push into buf;

V(sf);

P(sf);

pop from buf;

V(se);

all eles empty

① counting semaphore

all eles filled

sf = 12

se = 0

P(se);

push into buf;

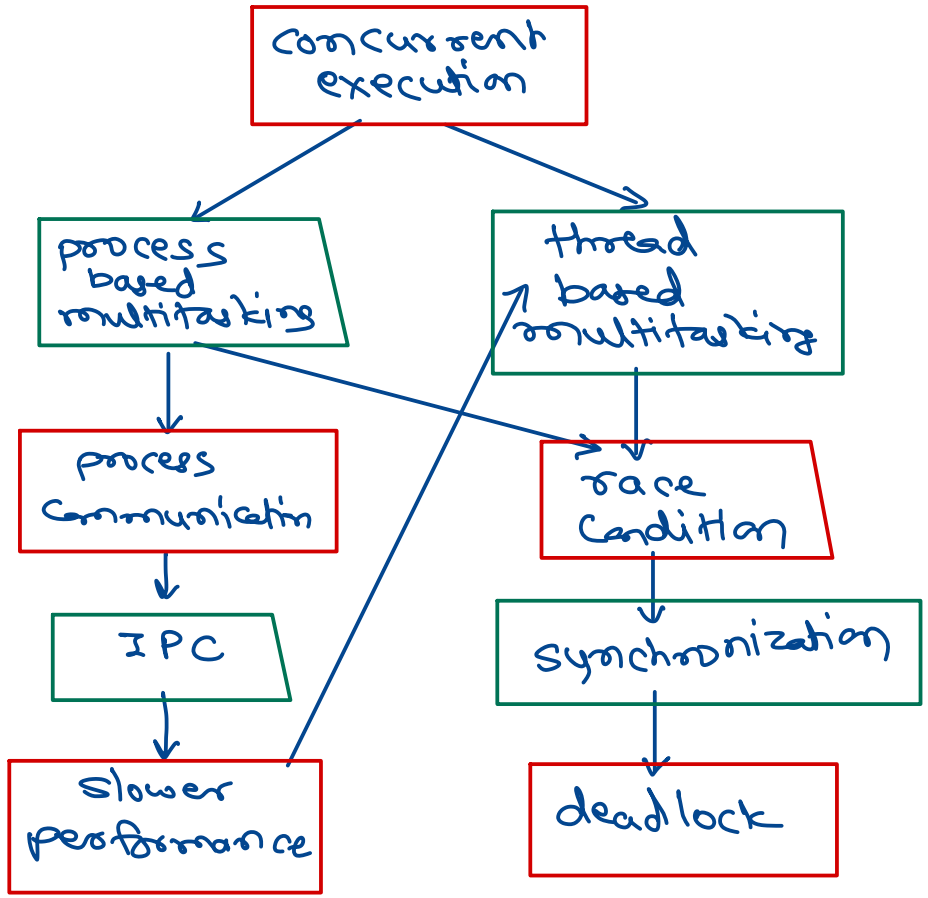
V(sf);

P(sf);

pop from buf;

V(se);

deadlock



program1

P(sf)

1. open file.

P(sp)

2. connect pointer.

3. print file

V(sp)

4. disconnect pointer

V(sf)

5. close file

program2

P(sp)

1. connect pointer

P(sf)

2. open file

3. print file

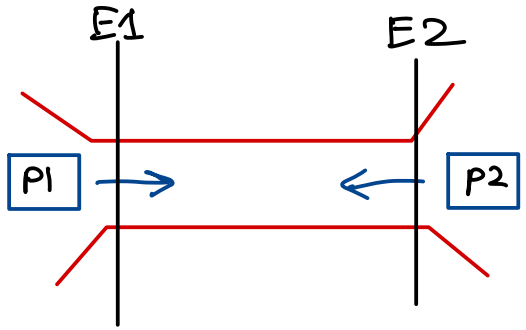
V(sf)

4. close file

V(sp)

5. disconnect pointer.

Both processes involved in deadlock are in blocked state.



Deadlock characteristics

1. No preemption
2. Mutual exclusion
3. Hold & wait
4. Circular wait

Deadlock vs Starvation

1. Starvation
   - process not getting CPU/IO due to low priority.
   - process is in ready state.
2. deadlock
   - process not getting resource because blocked by another process (deadlock chars).
   - process is in blocked state.

Deadlock solution

prevention is better than cure.

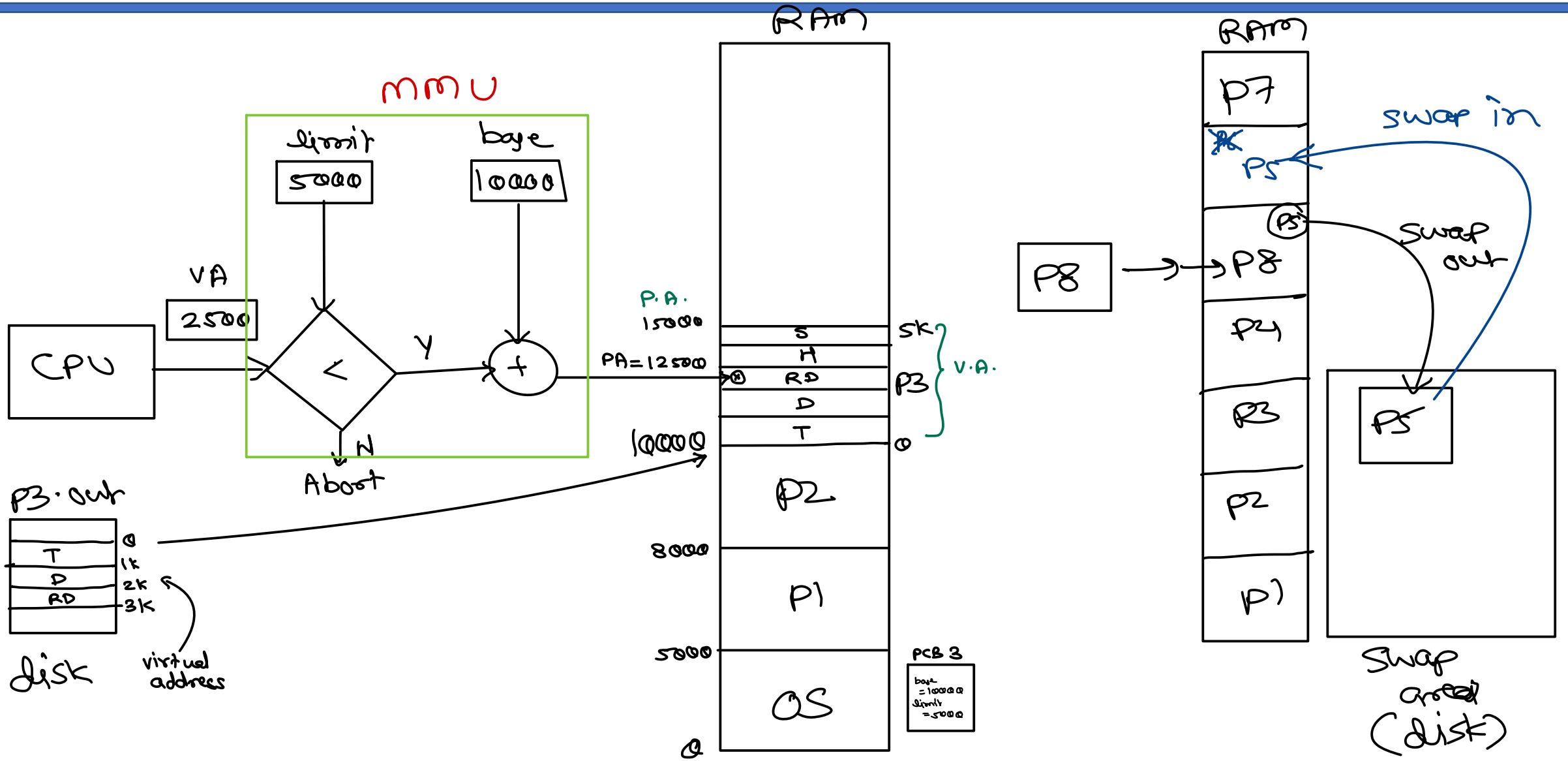
Deadlock prevention

Deadlock Detection

Deadlock avoidance

1. safe state
2. resource alloc graph
3. banker's algo

Memory Management



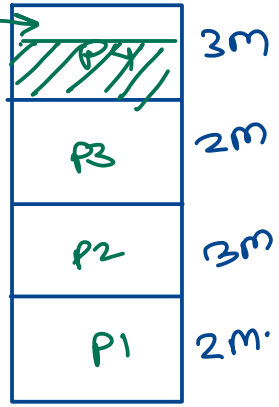
MM Schemes — MMU

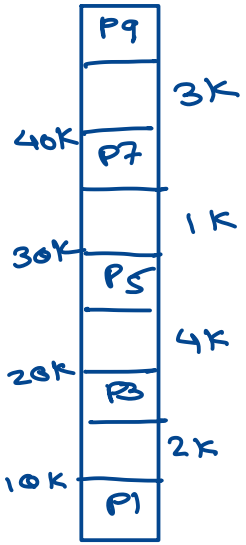
1. Contiguous alloc ← simple
2. segmentation ← segmentation
3. paging ← paging

Contiguous alloc

- fixed Partition method
- variable / dynamic method

internal fragmentation





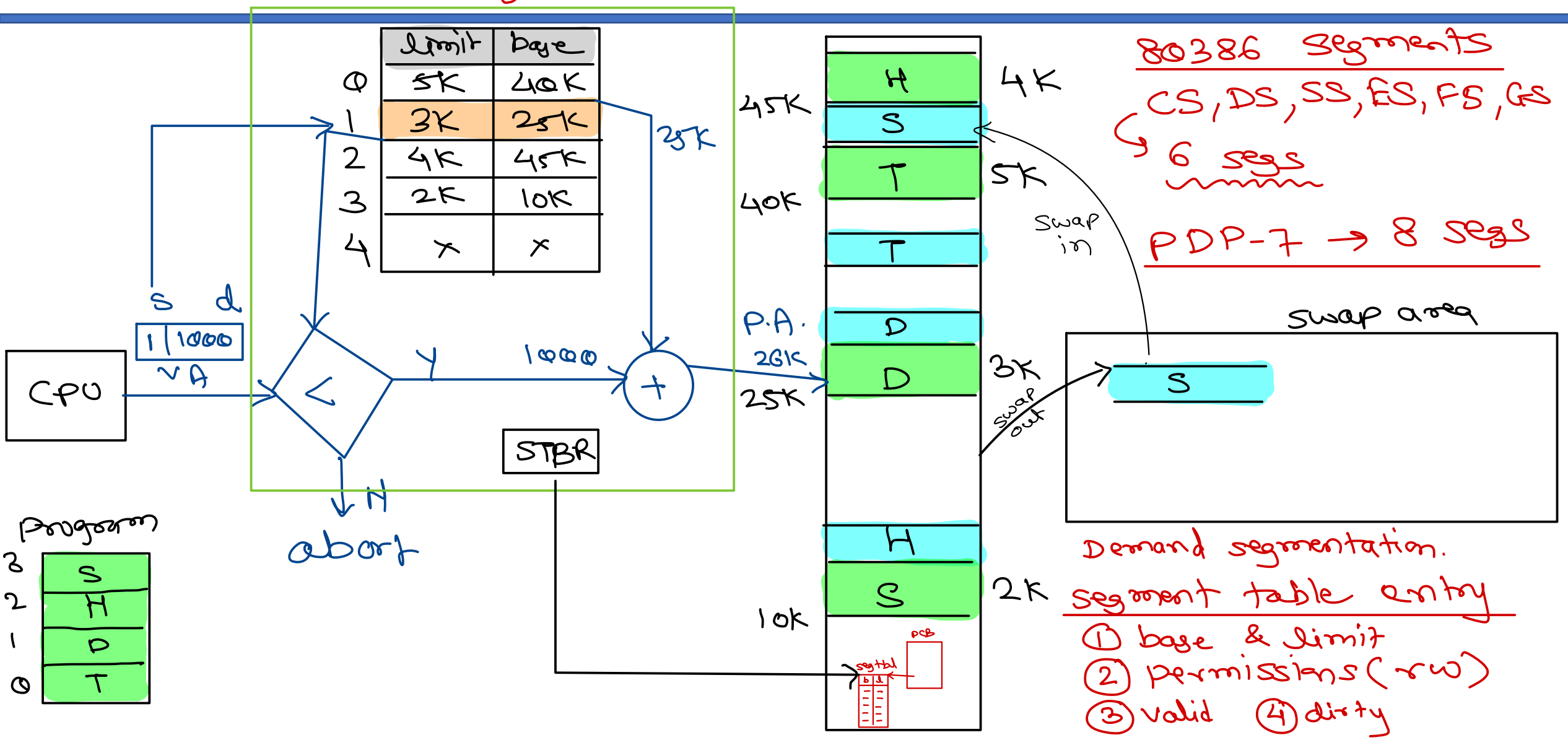
Compaction ← external fragmentation ← P11 (5K)

base	limit
10K	2K
20K	4K
30K	1K
40K	3K

first fit, worst fit, best fit

P10 (0.5K)

Segmentation



80386 segments

CS, DS, SS, ES, FS, GS

↳ 6 segs

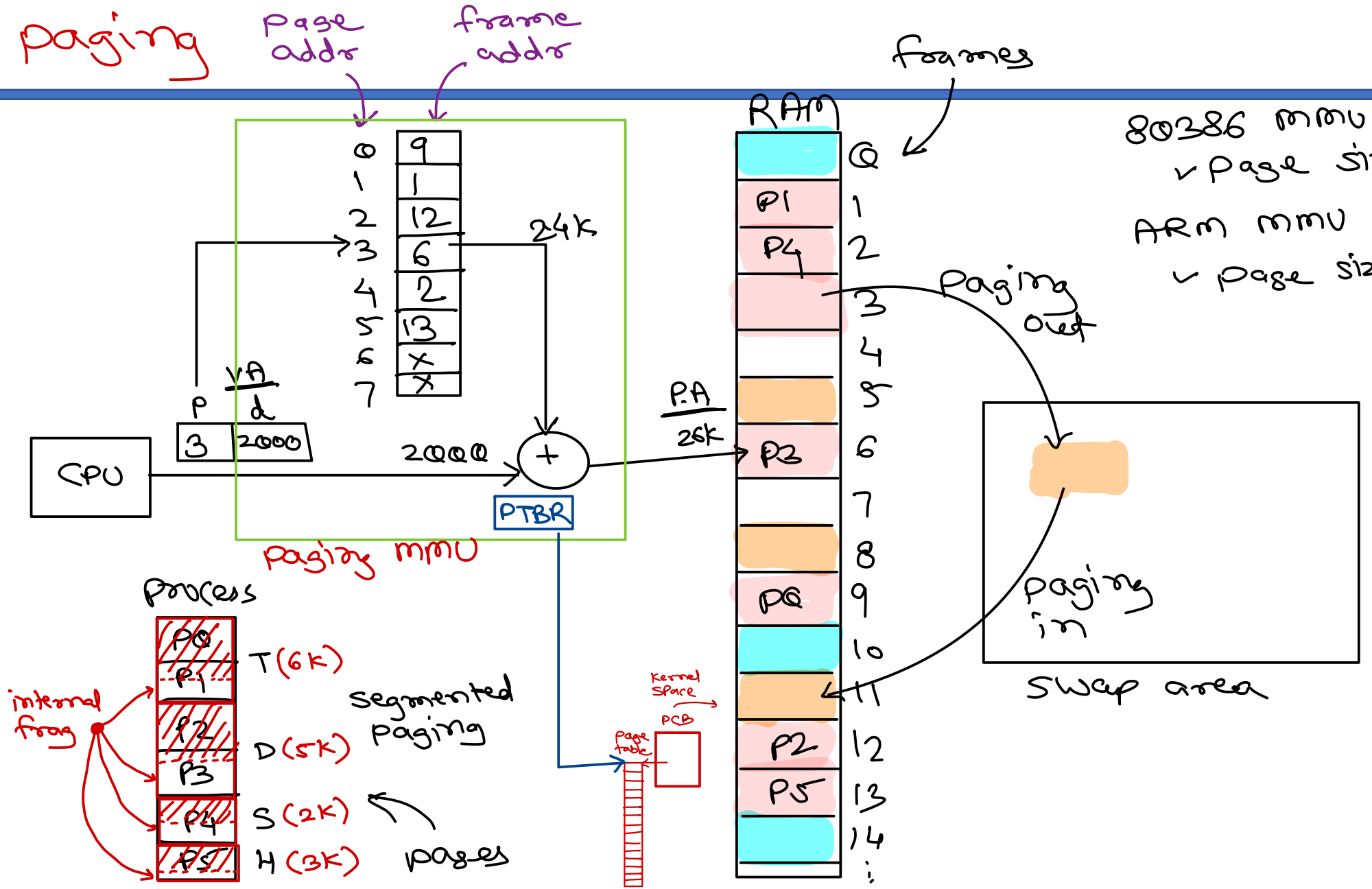
PDP-7 → 8 segs

Demand segmentation.

segment table entry

① base & limit
② permissions (rw)
③ valid ④ dirty

paging



80386 mmu
- page size = 4K or 4M

ARM mmu
- page size = 1K, 2K, 4K or 64K

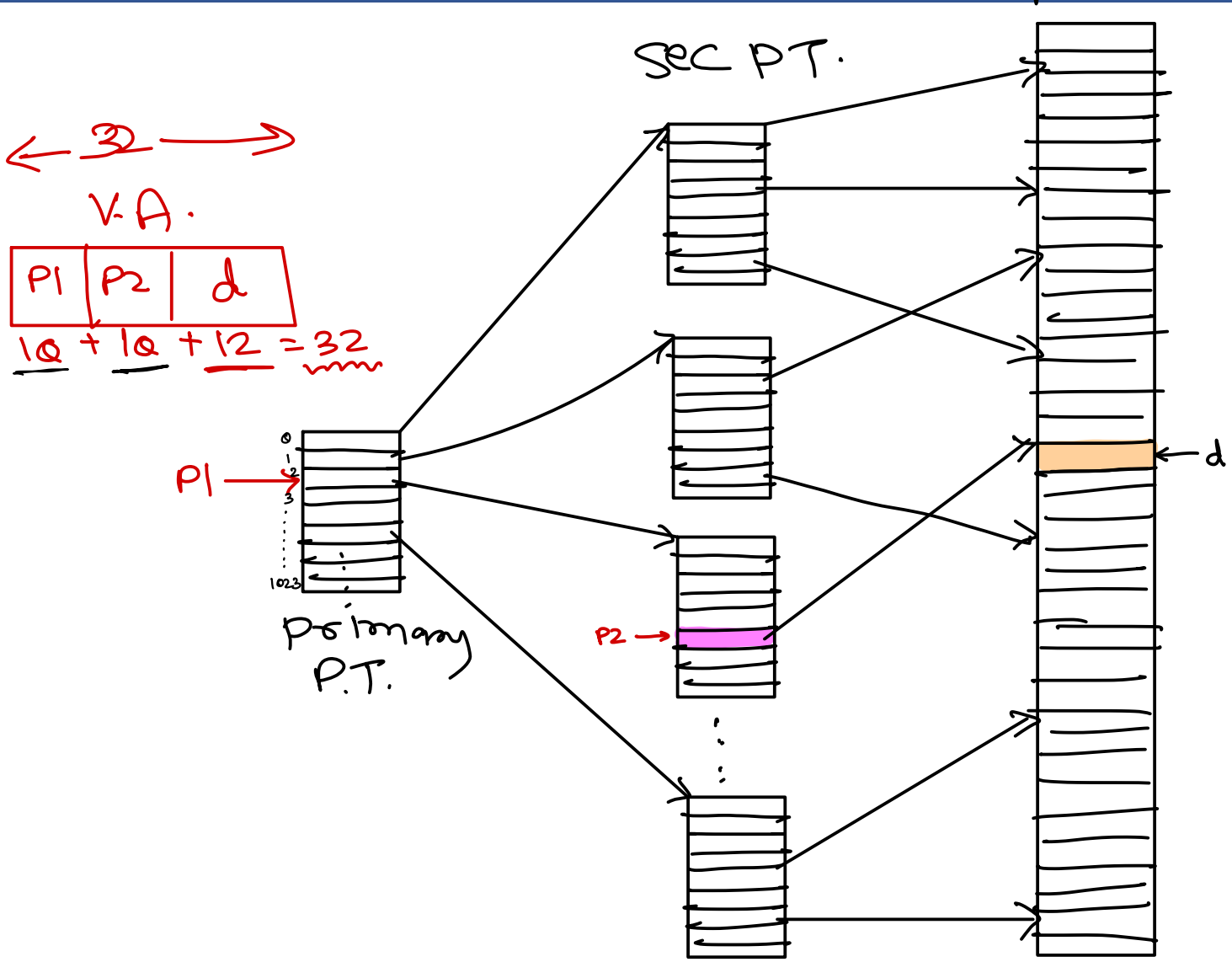
Demand Paging

page table entry

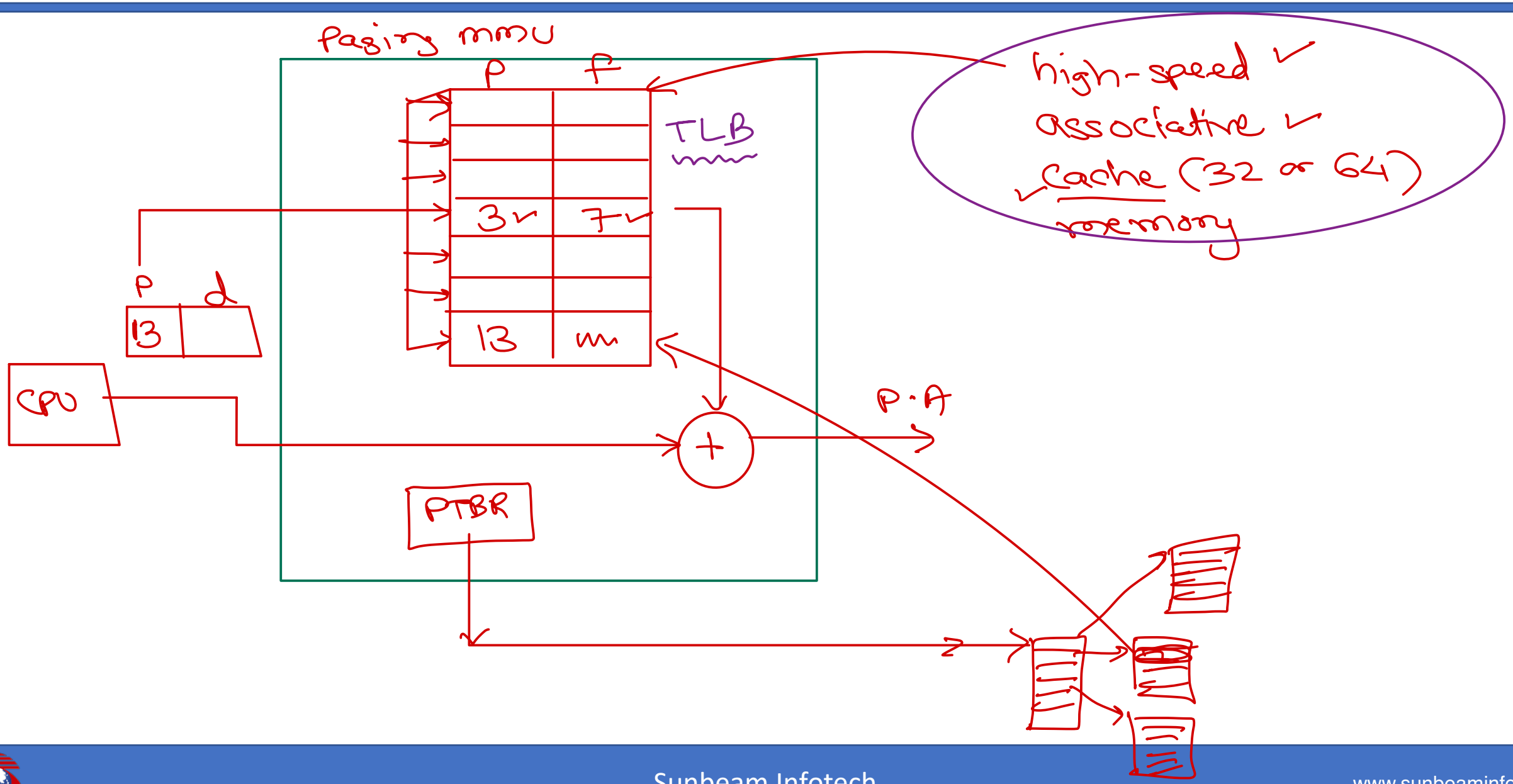
1. frame addr
2. permission (rw)
3. validity bit
4. dirty bit

2-level paging

32-bit OS

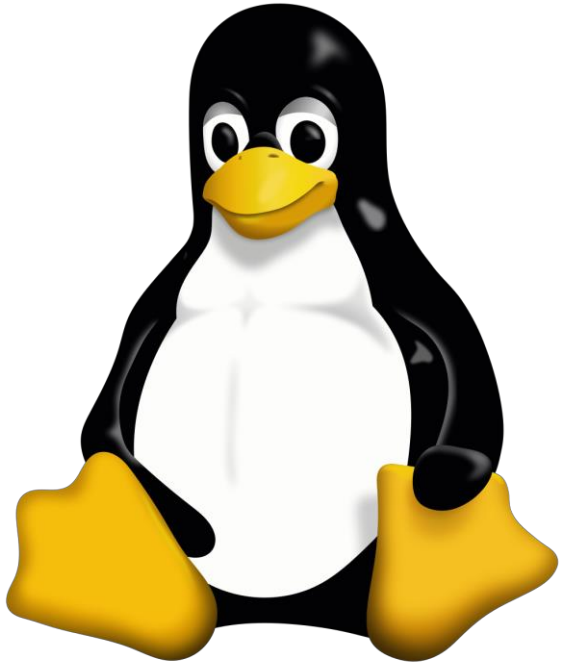


max mem
$= 2^{32} = 4$ GB.


Paging mmu
p
f
TLB
high-speed
associative
cache (32 or 64)
memory
3
7
13
m
p
d
13
CPU
+
P.A
PTBR

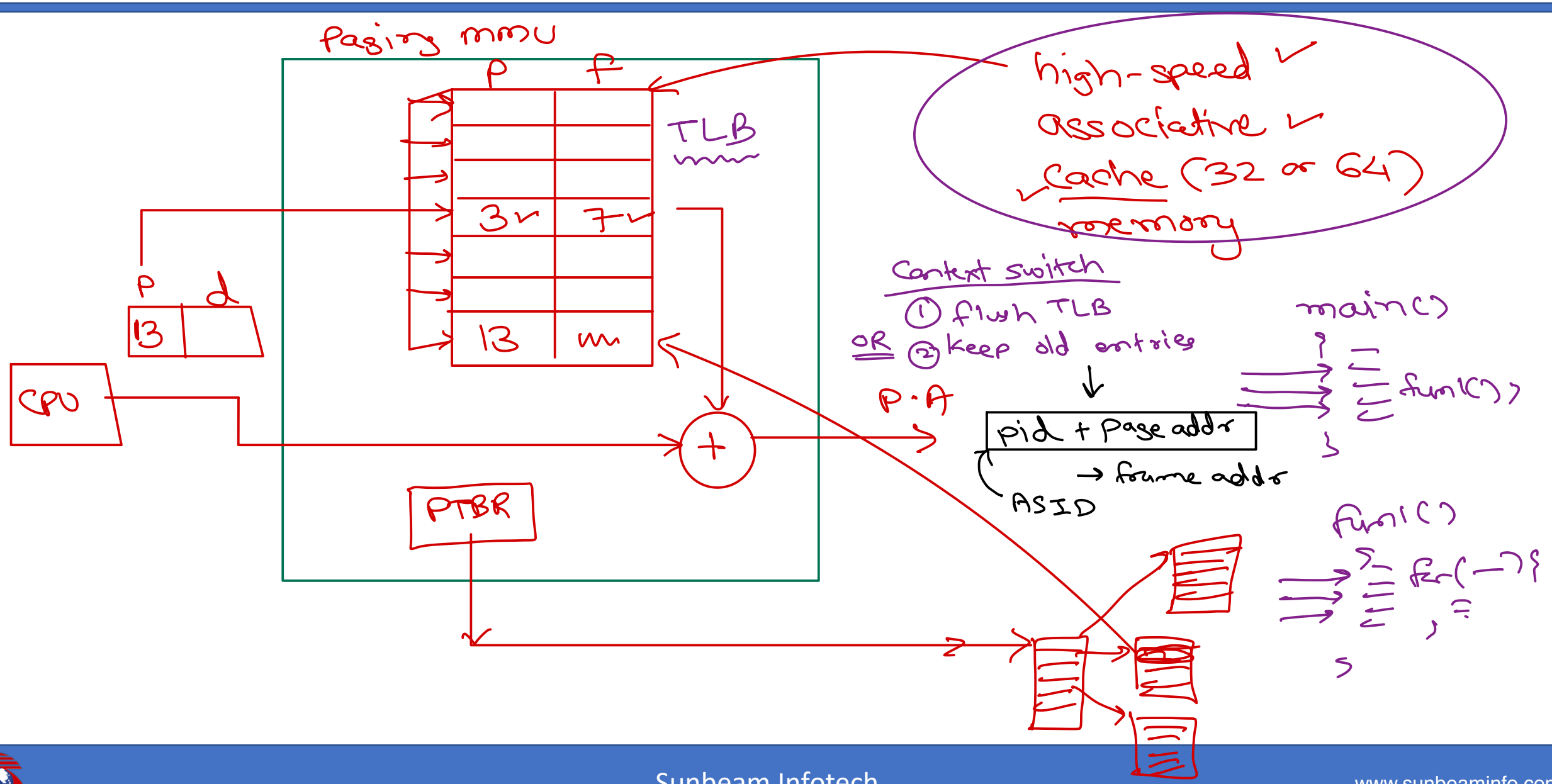
Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>



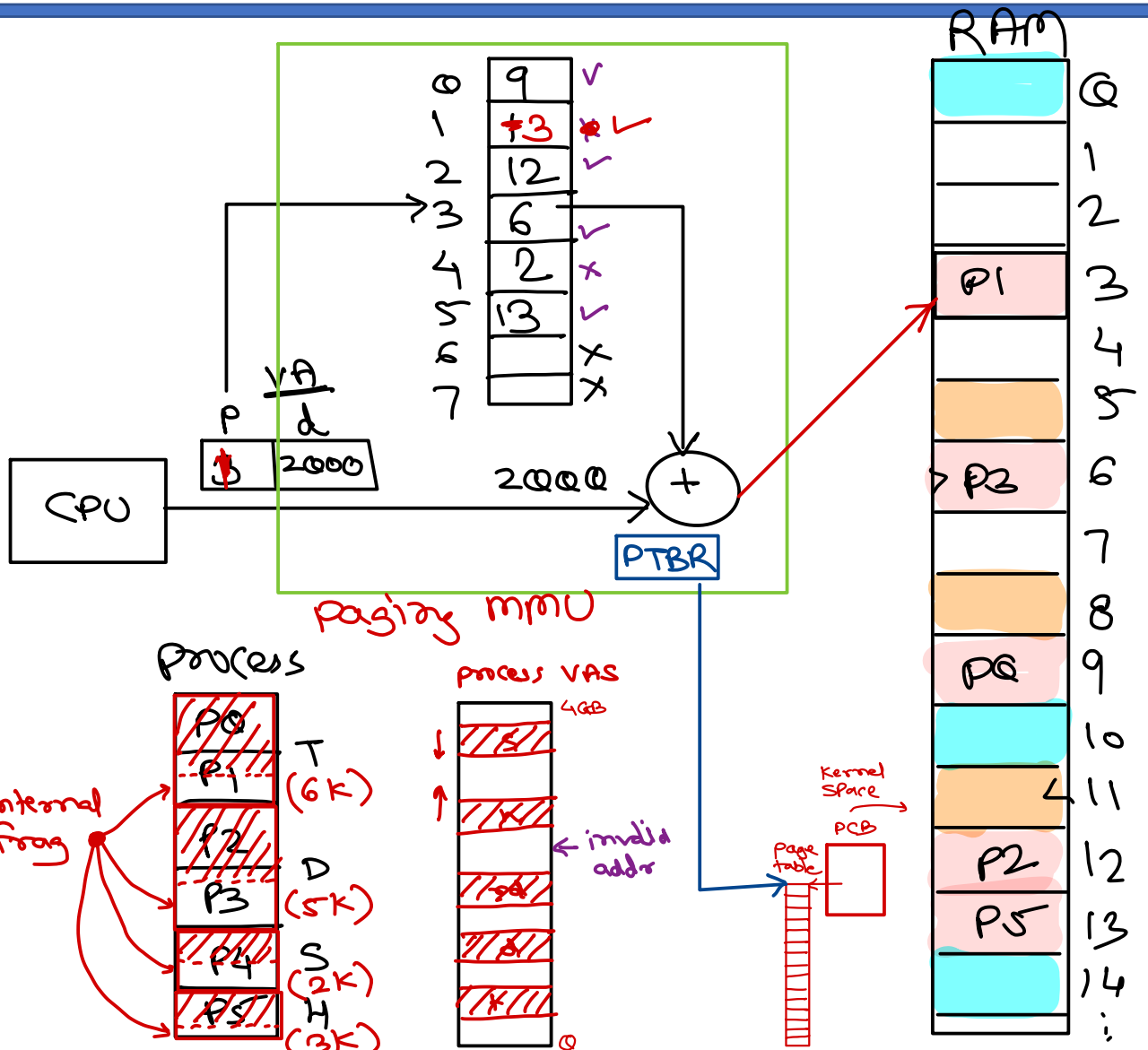
Operating System – Linux Programming

Sunbeam Infotech


Paging mmu
p
f
TLB
3
7
13
m
p
d
13
CPU
PTBR
high-speed
associative
cache (32 or 64)
memory
Context switch
① flush TLB
OR ② keep old entries
P.A
pid + Page addr
→ frame addr
ASID
main()
fun()
fun()
fn()

paging

malloc()



PTE - validity bit

1 → page in RAM

0 → page not in RAM

① page is swapped out → P4

② page is not yet loaded → P1

③ page is not yet allocated → dyn mem alloc.

④ page is not part of process

physical

CPU ⟶ req page → PTE (invalid) → page fault
↓
page not in RAM

page fault → page fault exception handler (OS) is executed.

ⓐ check if VA is in process VAS (in some seg).
if not send SIGSEGV (abort). → validity fault

ⓑ check if have appropriate perm on the requested VA.
if not send SIGSEGV (abort). → protection fault

ⓒ allocate an empty frame for the page.
if no frame is empty, swap out some page (victim page) → page replacement algorithm.
① FIFO
② Optimal
③ LRU
& make that frame available for current page.

ⓓ if page is on disk/swap, load it in the allocated frame.

ⓔ update PTE → frame addr, valid bit = 1

ⓕ restart the instn at which page fault occured.

virtual memory advantages:

① process size > RAM

② can run more processes than RAM size.

Thrashing

disk IO (swapping) > process execution

overall system is slower.

linux swap file → mkswap

---

② vfork()

① dirty bit

PTE → d=0 when page is loaded from disk/swap.

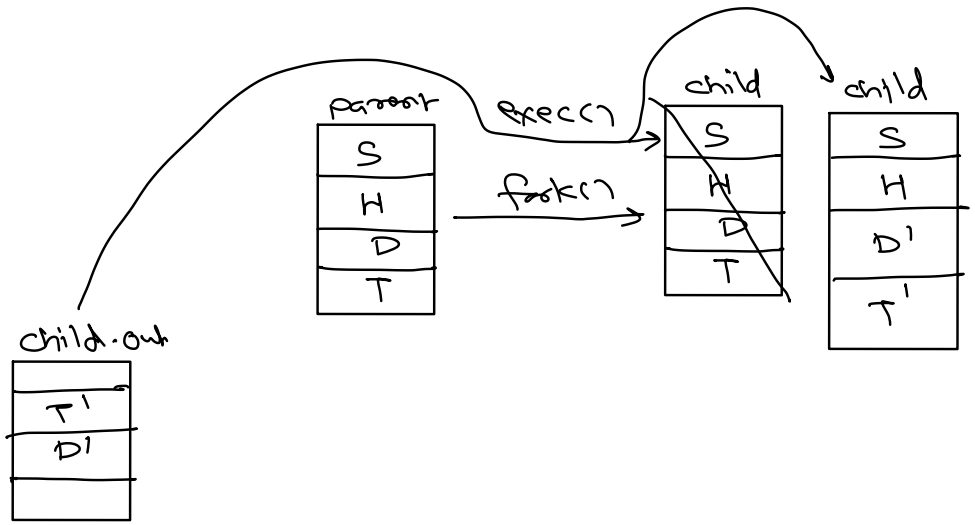
↳ copy of page in RAM is same as copy in disk. (ie- page is not modified).

dirty bit is set to 1, when write-op is done on page. Next time is page is to be swapped out, it must be written on disk.

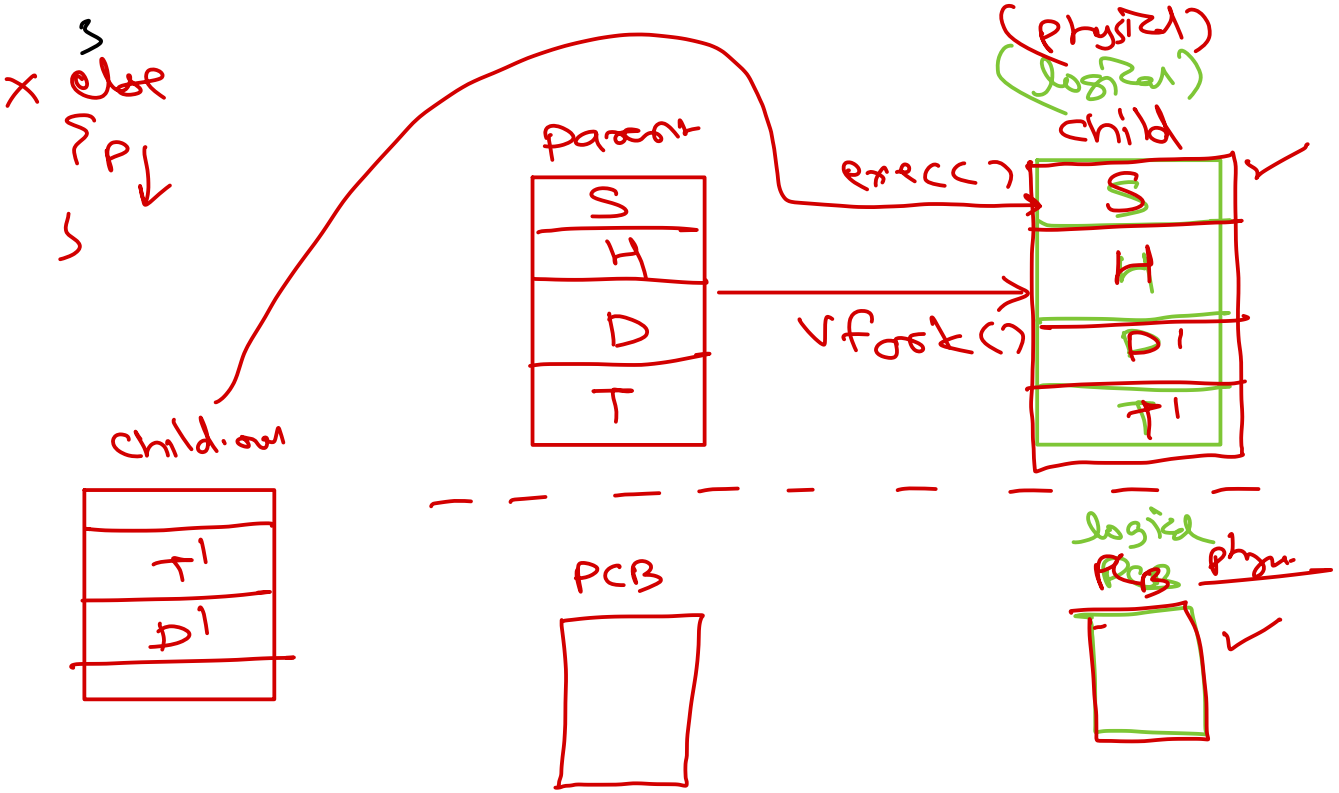
③ copy-on-write

vfork() → BSD UNIX

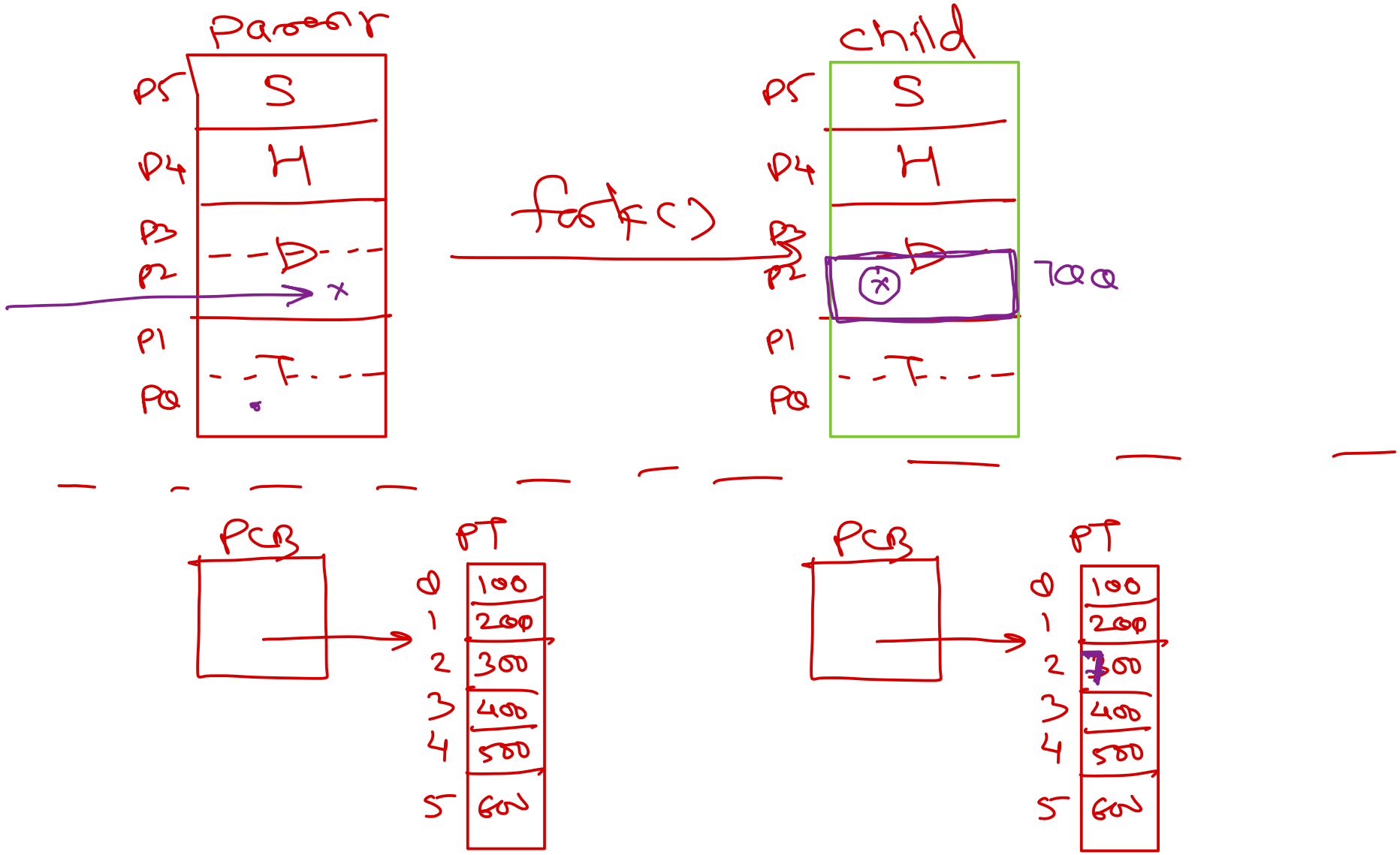
```
ret = fork();
if (ret == 0){
    exec("child.out");

}
```




```
Par
ret = vfork();
if (ret == 0){
→ P
    exec("child.out");

X else
{ P
}
```



Copy-on-write



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>