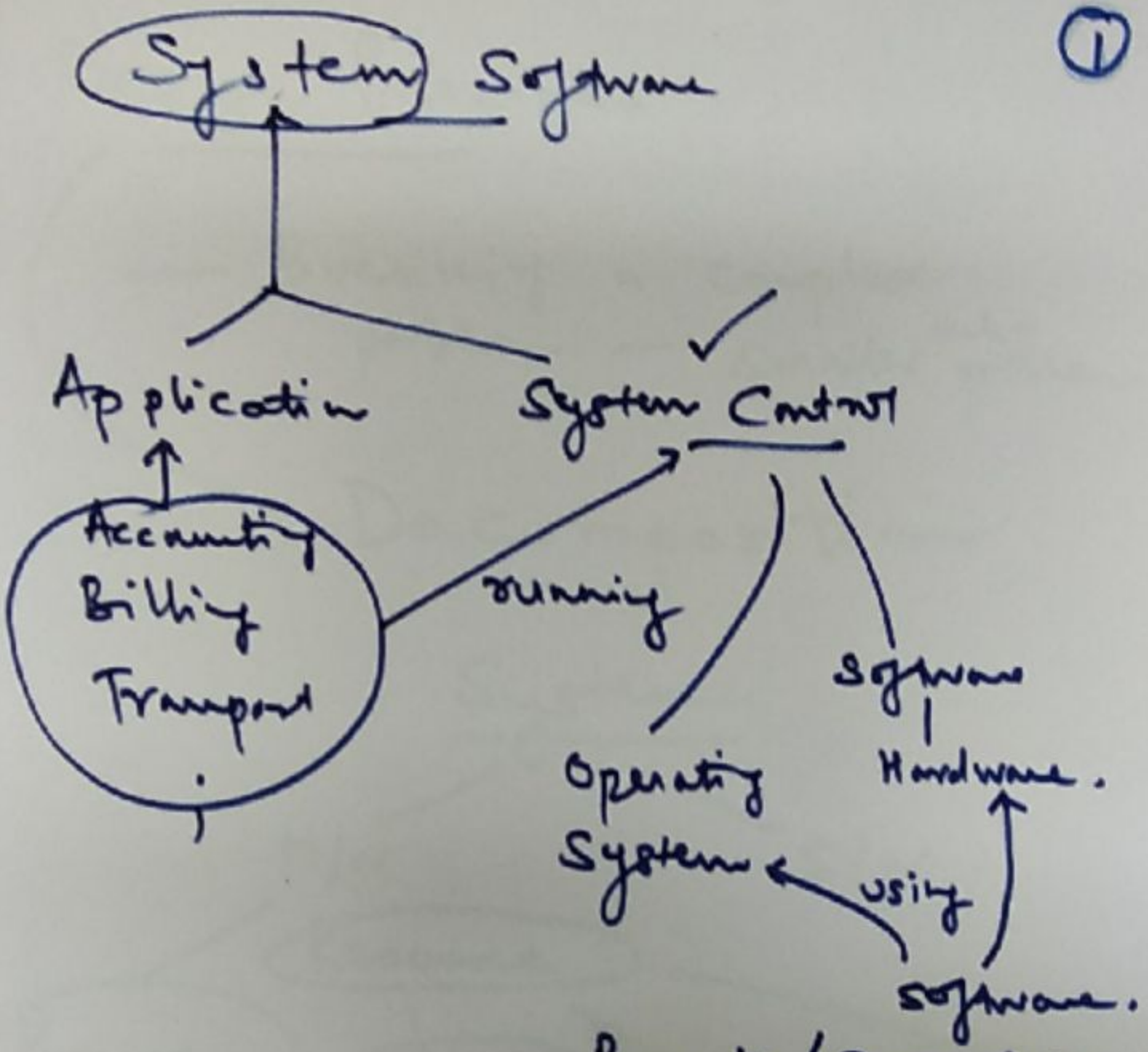


Projects / Exercises

~~Scope~~
Scope
(Define)

- Own Shell.
- Assembler

Reasonably demonstrate Concepts

Design Principles

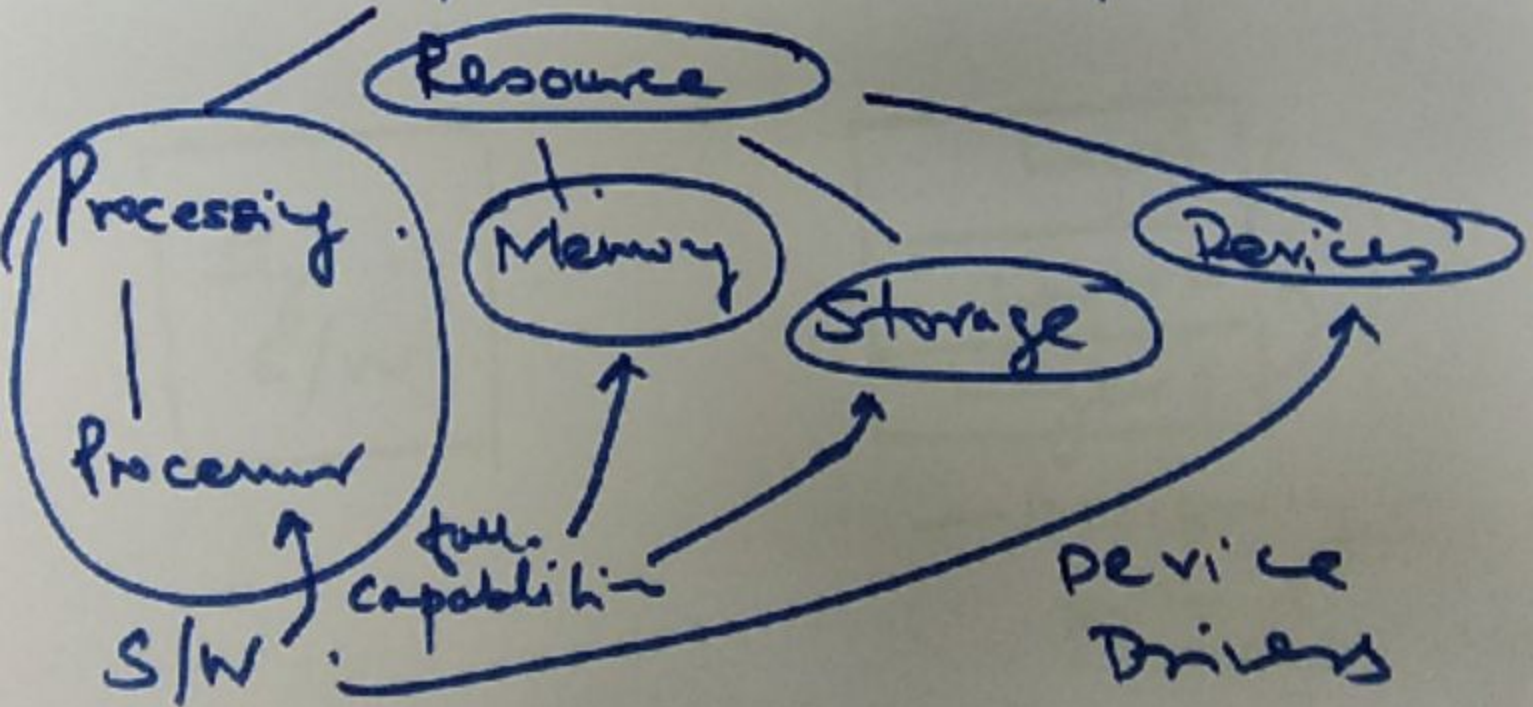
— Breaking a complex problem — Smaller sub-problem

Decomposition

System

H/W

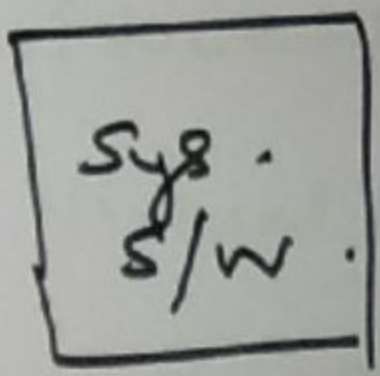
S/W.

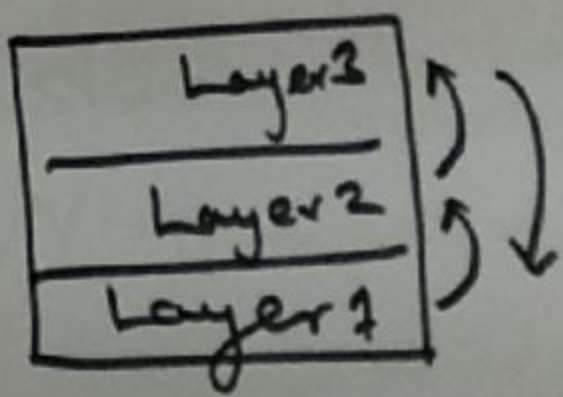


Layering

Build System.

- Monolithic
  - one
  - old
  - useful.
- Layered.
  - system functionality across layers





- Designing interface

Resource — CPU, Memory... —

Manage

— Resource Manager

- Fairness — Impartial behavior of system
- Efficiency

Control Program

Resources.

- Process → Shield the inside details
- Virtual Machine / Virtualization

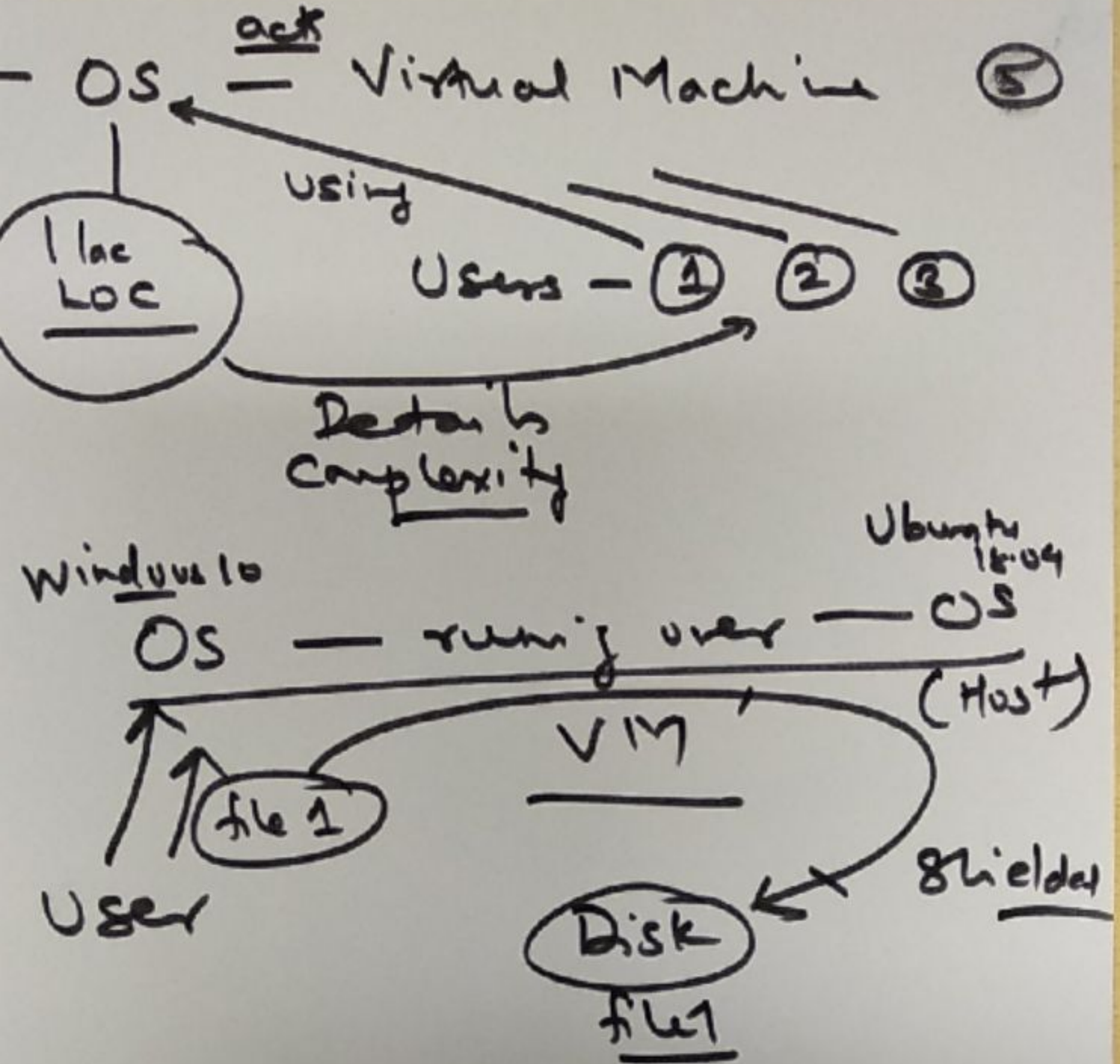
Laptop — OS

Base / (Host)

Windows 10

Ubuntu 18.04 LTS (64-bit)

Ubuntu 12.04 LTS (32-bit)



Distributed System

Cloud Computing

— Uniprogramming

System
CPU

Single
Uniprogramming
one
single

Multiple
Multiprocessor → Multi core
>1 CPUs >1 core
CPU
-cost
-size . -cost ↓
-size ↓

Multiprogramming
- execute multiple programs

Process — relation — Program

- System View
- requires resources
- Active becomes.

User write
- Language Specific.
- Human readable.

useful. — No resources used

Run. — Passive

Program
- C program
- Memory. (printing)
- Files (Basic)

gcc — Object code.

a.out — Run

Run
exe

Input-Output Devices (I/O)

CPU — doing everything

- CPU utilization factor ↑ high
- CPU Idle ↓ minimum

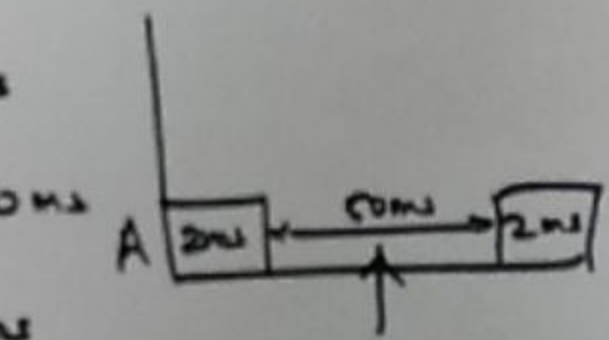
System — Operation
- CPU operation — short, fast
- I/O operation — long, slow

Program A

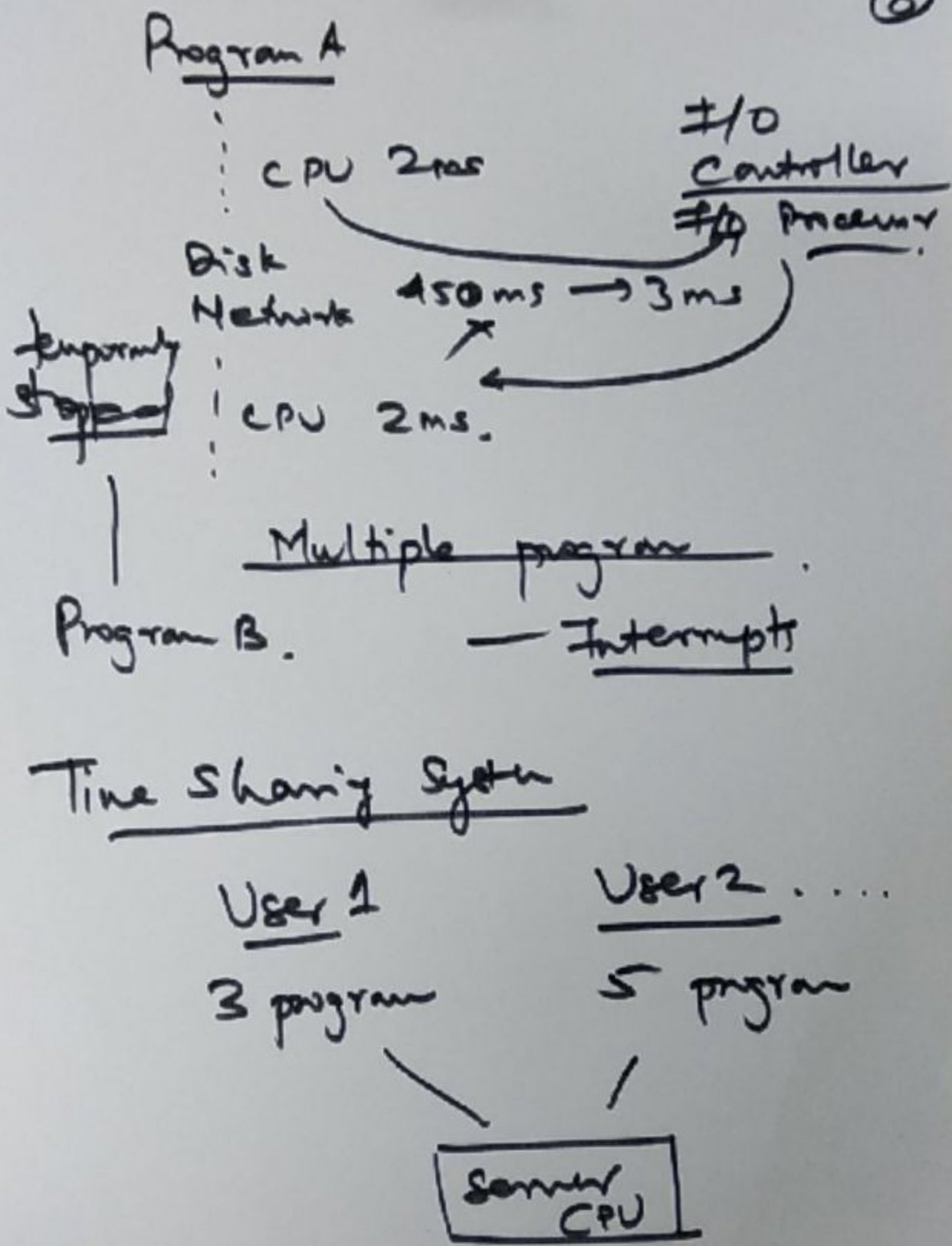
CPU 2 ms
- Disk
- Network

50 ms

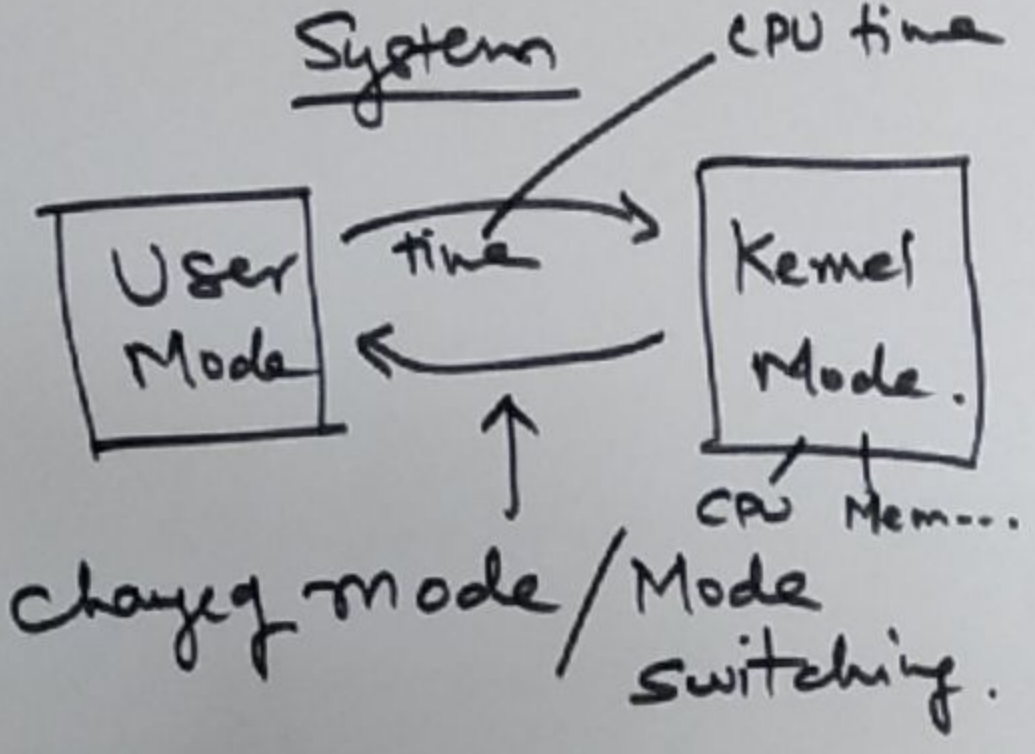
CPU 2 ms



I/O Processing


Program A
CPU 2ms
I/O Controller
I/O Process
Disk Network
450ms → 3ms
temporarily stopped
CPU 2ms.
Multiple program
Program B.
Interrupt
Time Sharing System
User 1
User 2 ....
3 program
5 program
Server CPU

- System Modes
  - User Mode — limited.
  - System Mode
    - Kernel Mode — full control
- Security.



Process

|

Classification — Types of Processes

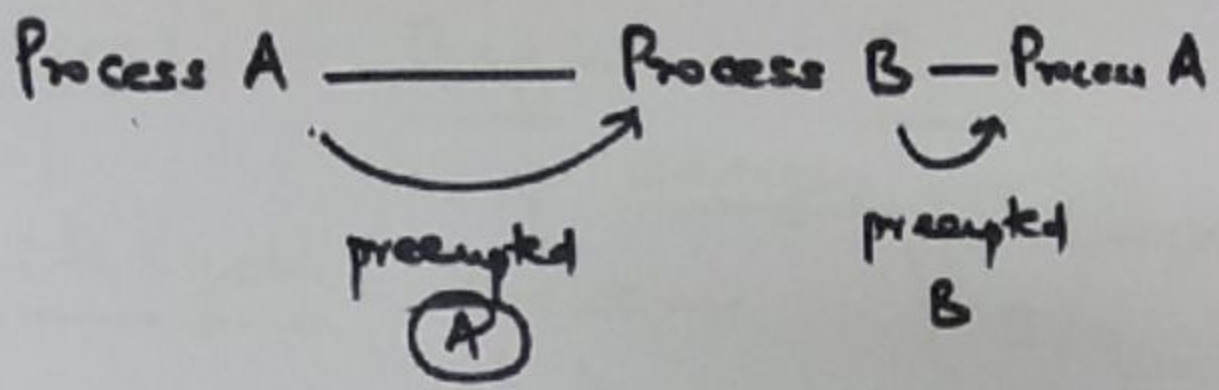
could be used to gather

1. User level process (low)
   vs.
   Kernel level process (high)

2. CPU-bound process
   CPU-intensive process
   vs.
   I/O-bound process
   I/O-intensive process
   | Type of activity.

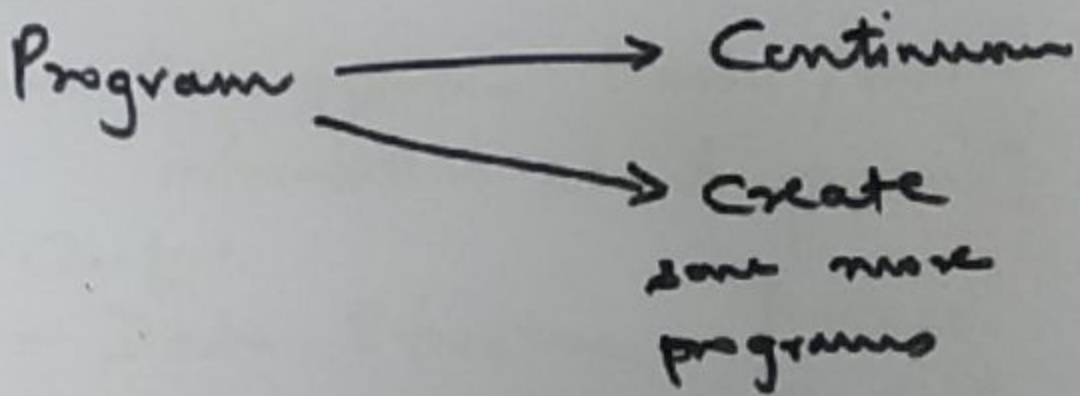
3. Cooperative process
   vs.
   Non-Cooperative process (Independent)
   | Relation both processes.

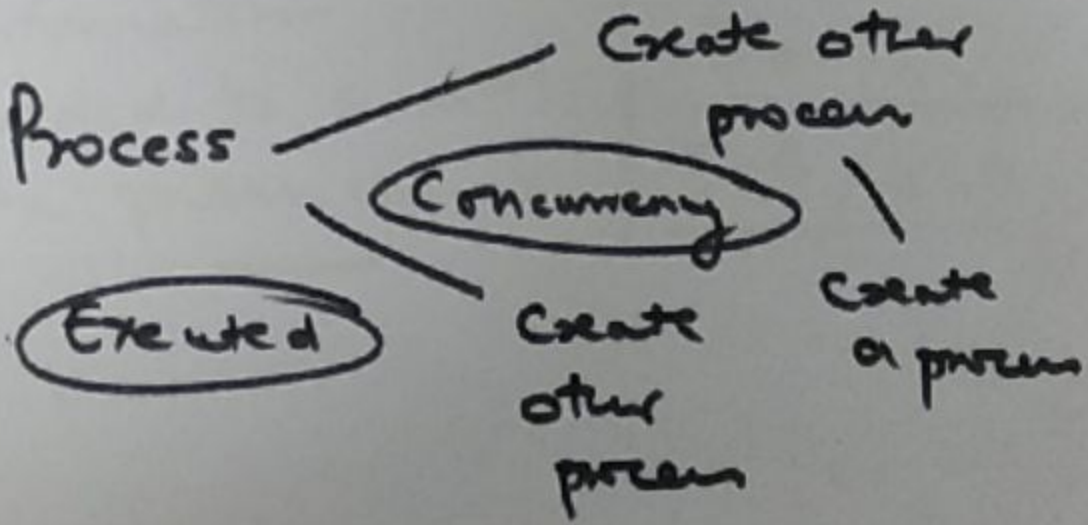
4. Preemptive vs Non-preemptive

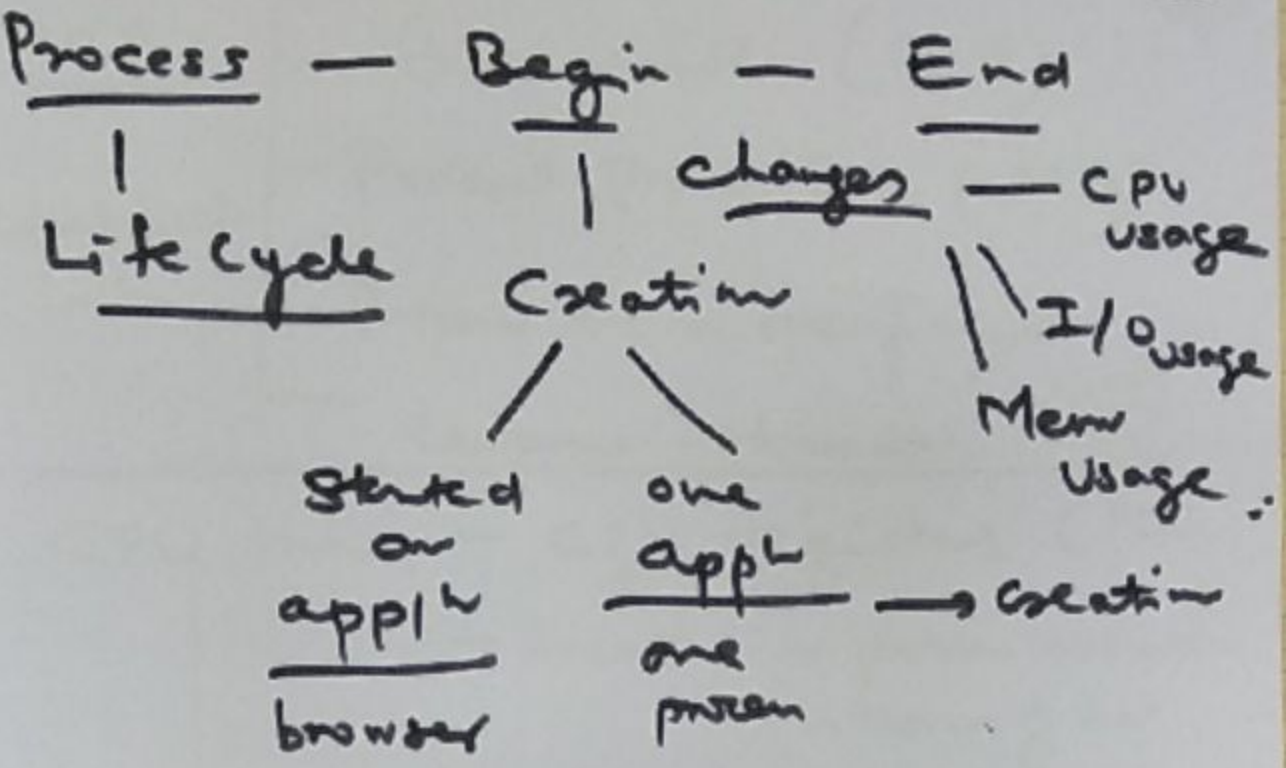


Process ——— Concurrency

Two or more processes could together







How the system tracks / maintains information about prosen?

Context
— Data Structure — Process Control Block (PCB)

Information needed to control the prosen execution

TCB / JCB

PCB

Identification info.
- Process ID (PID)
- Parent process ID (PPID)
- Process for a specific user (UID)

Resource information

CPU info.
- CPU registers (PC)
- location - process execution in terms of inst.

Scheduling — order of process execution

Mem usage — location of process

Secondary storage usage ..

I/O devices ...

Files — used/opened ...

For every process — PCB

Process Creation — how?

System call — privileged operation (Not in user mode)

fork()

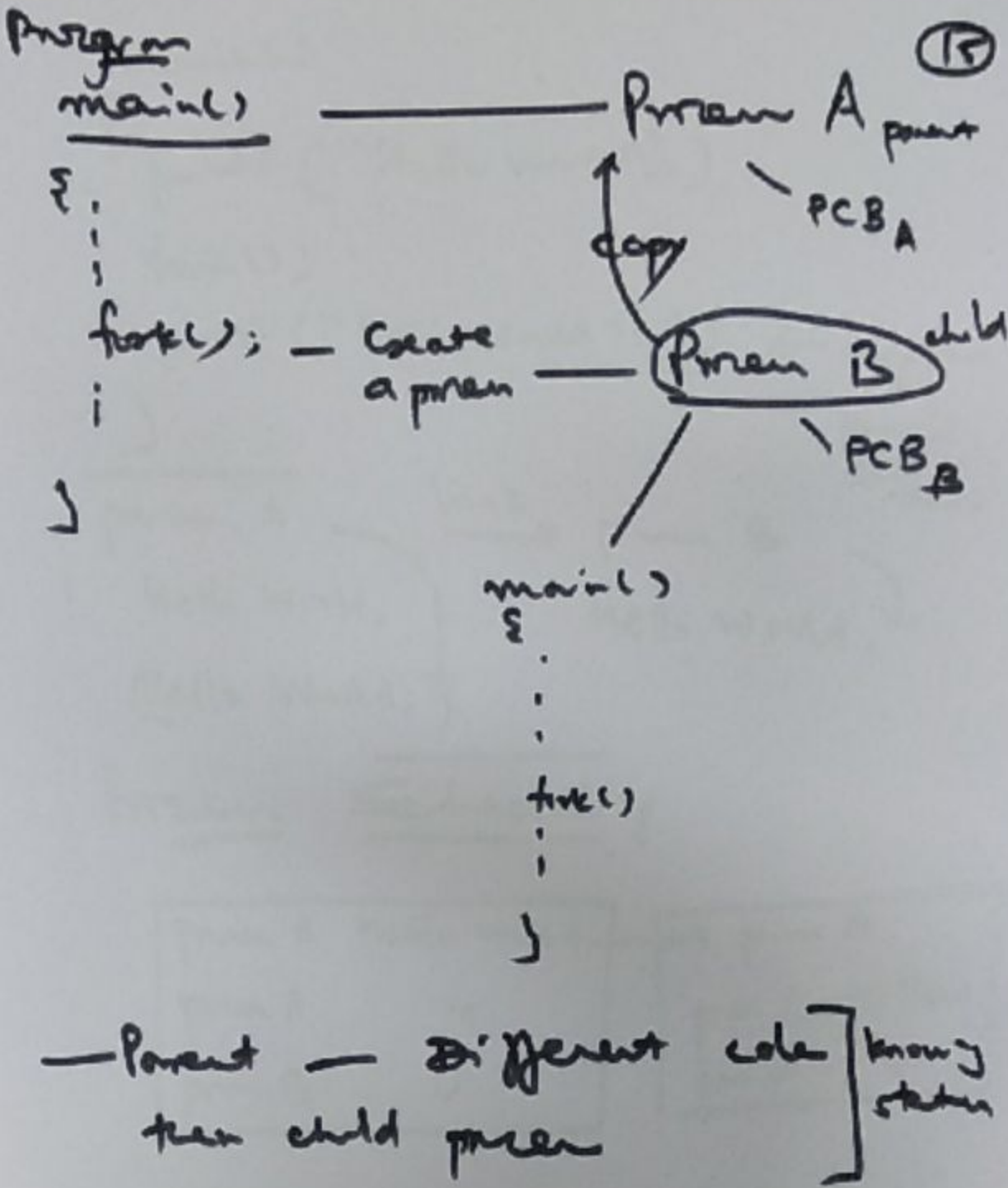
initiated / called in user mode

after only — kernel mode

```
main()
{
    ....
    User mode
    System call (user mode)
    ....
}
```

kernel

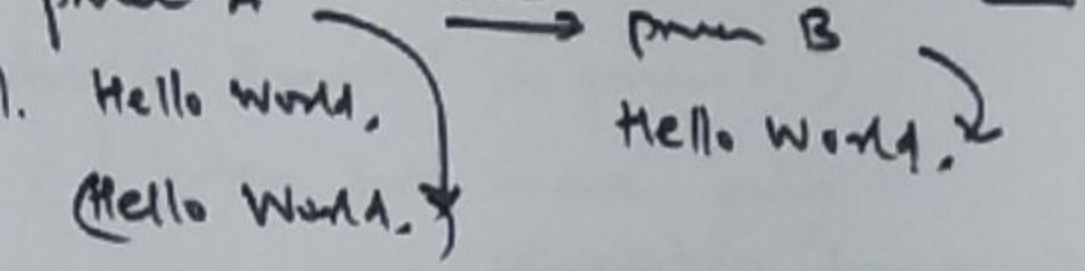
fork() — Process Creation



— Parent — Different code — know status
than child proc

```
main()
{
1.  printf("Hello world\n");
    fork();
    printf("HelloWorld\n");
}
```

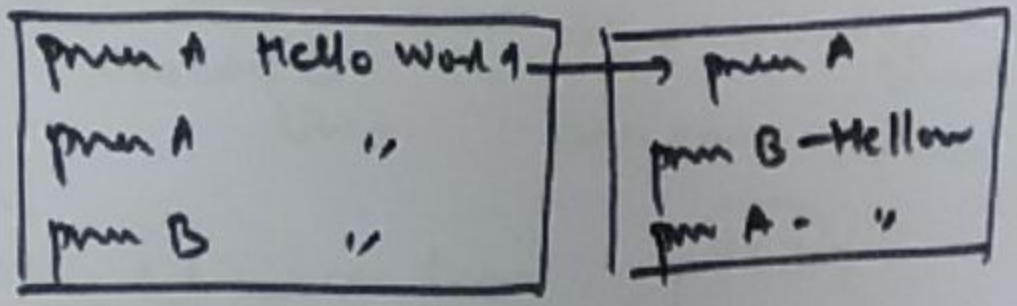
← executed by parent child

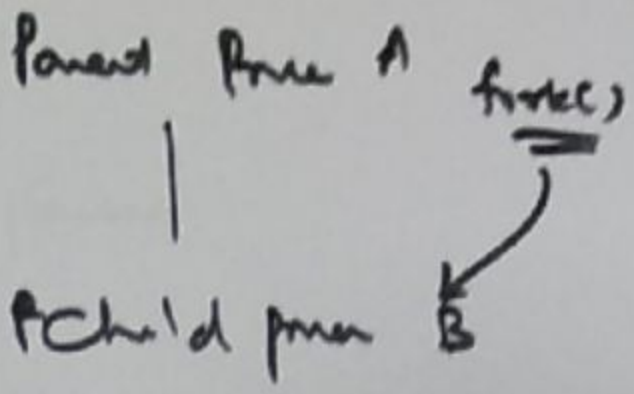


1. Hello World.
   Hello World.

Hello World.

Process Scheduling



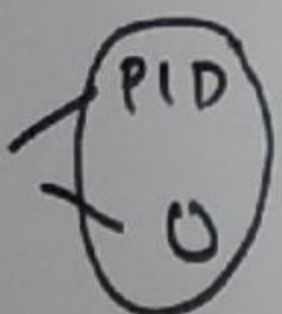



```
main()
{
    5 LOC
```

6. fork() (executed in parent A)
7 Line : ↓ below fork — one only remaining line executed)
}

Parent A	Parent B
execute 13 lines	copy — has 13 lines execute — 7 lines

fork() — return status



— code.
if ... else

```
if (PID)
    : parent
else
    : child.
)
```

if (PID)

8.
9. | parent
10.
else

11.
12. | child.
13

Process A

1- 6, 8- 10

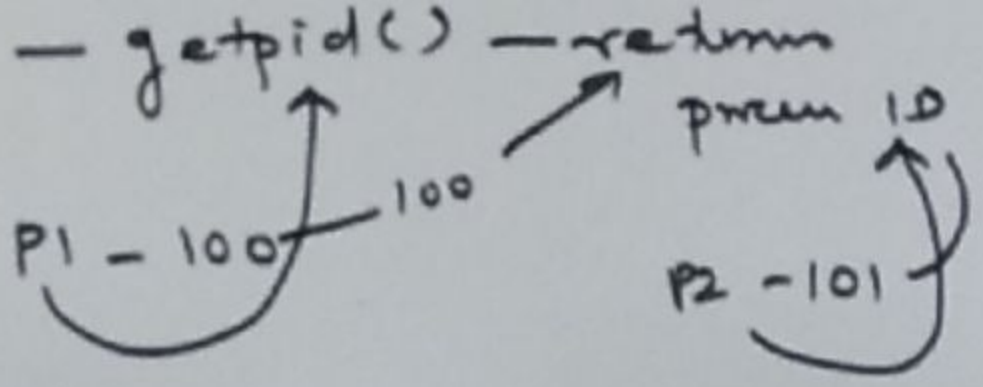
Process B

11 — 13

— Invoke one appln from another
system call

- Concurrency

calls



printf("%d Hello World\n", getpid());

fork() — Process Creation

↓ Lifecycle

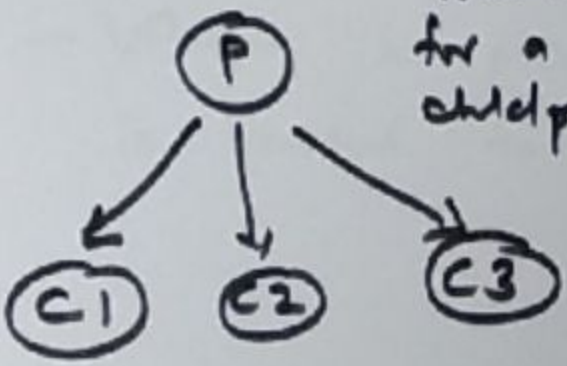
Process Termination

Details

Introduction

Stage 2

Wait() — wait for a child process



order of termination

C3 —

C1 —

C2 —

waitpid()

Specific process

Parent forks() — Wait() / Creation

Child wake() ~ Termination

— State Models (Conceptual model to show process lifecycle)

— When a process is created, it is meant to do some activity

↓

Execution of instruction

↓

— RUN (using CPU)

— It Not using the CPU

Resource.

CPU

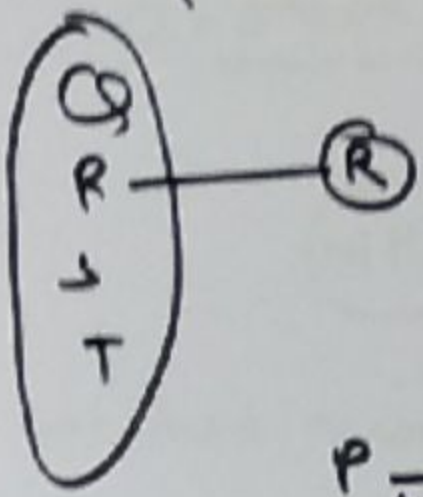
Mem

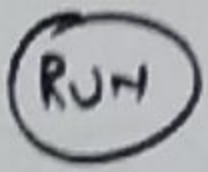
Storage

:

;

other processes.



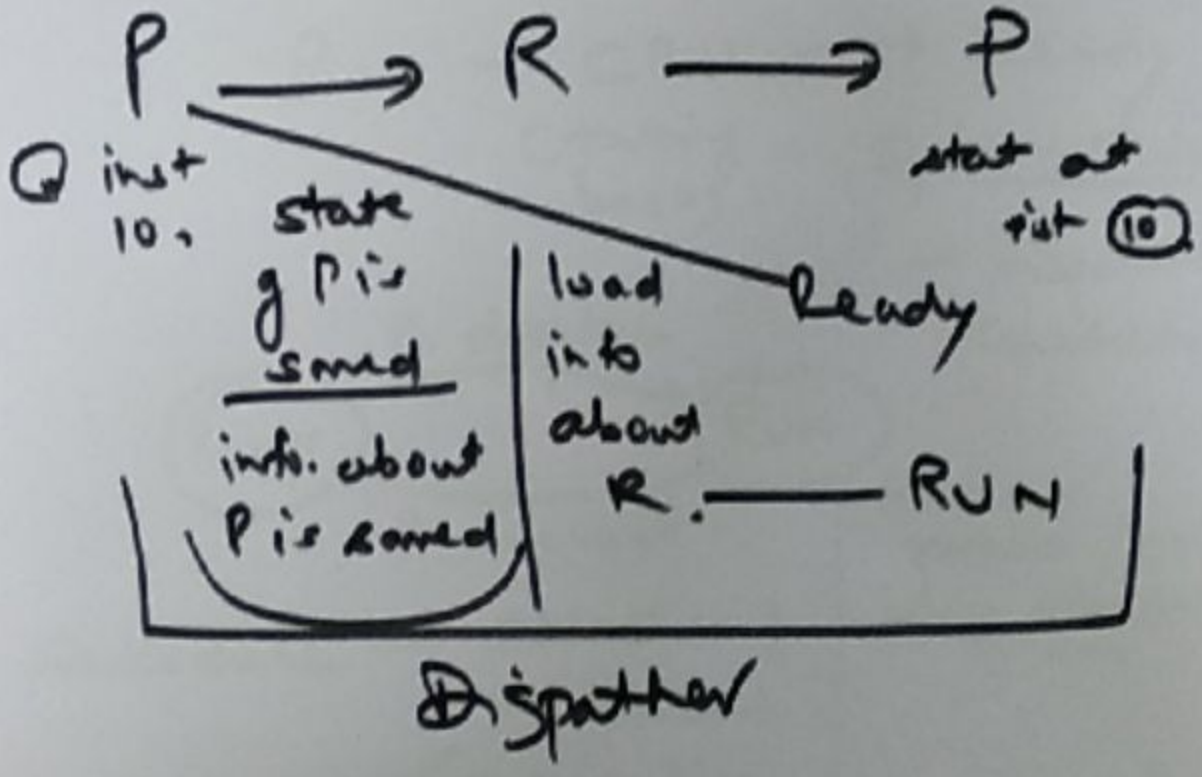


P

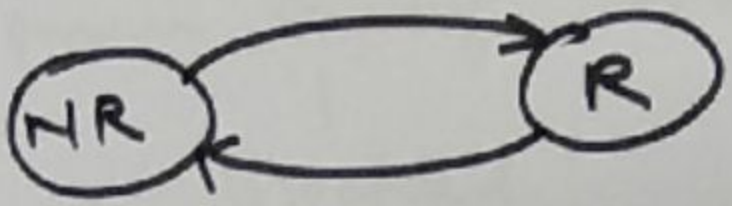
P is using CPU

P → R → P
(process switching)

— Indicate that a process can use CPU for some time



2-state model



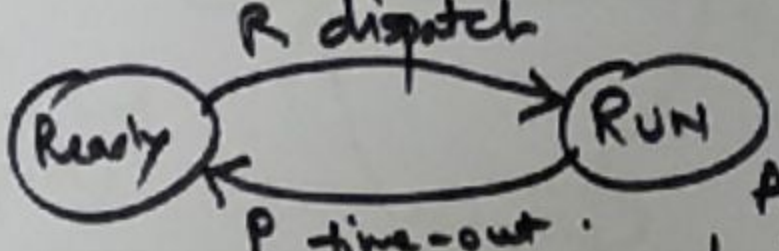
— Add more states

— Ready state. — Process has all resources except CPU

Process P is Ready

RUN — CPU → Ready
(taking back)

System Modules
— Dispatcher
— Scheduler



Schedule — Selecting a process (Scheduling Algo)

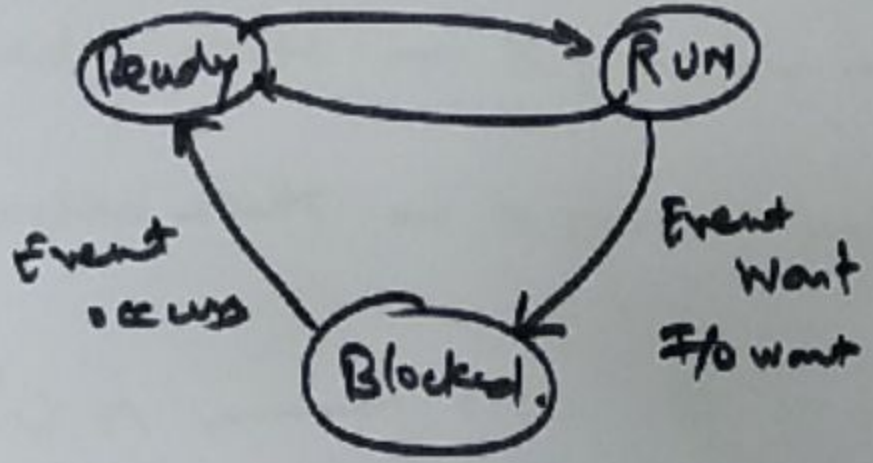
After a process has been selected

I/O Resources

— process is using I/O

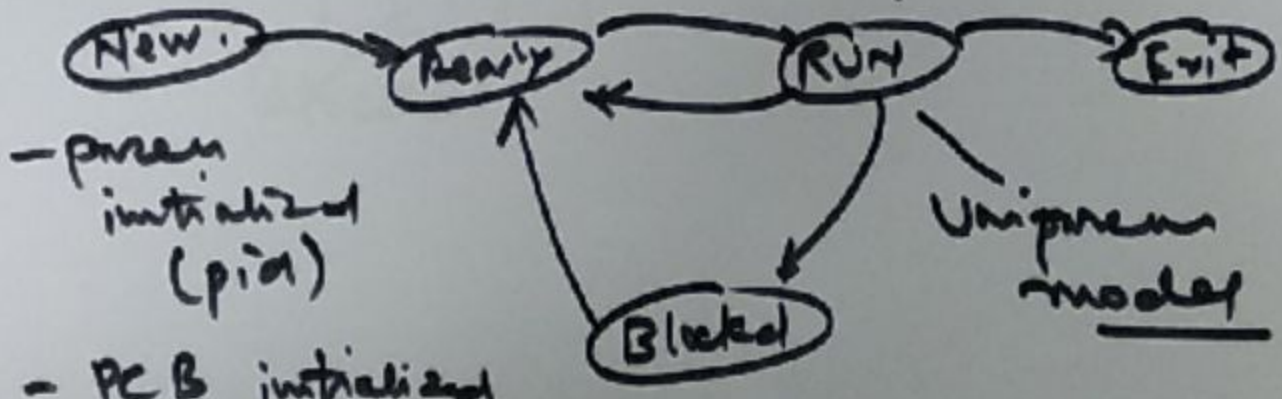
↓

Blocked.



3-state Model

Exception



- process initialized (pid)
- PCB initialized.
- Available resources. (if any)

5-state model

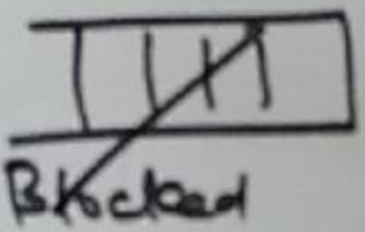
Quad-core

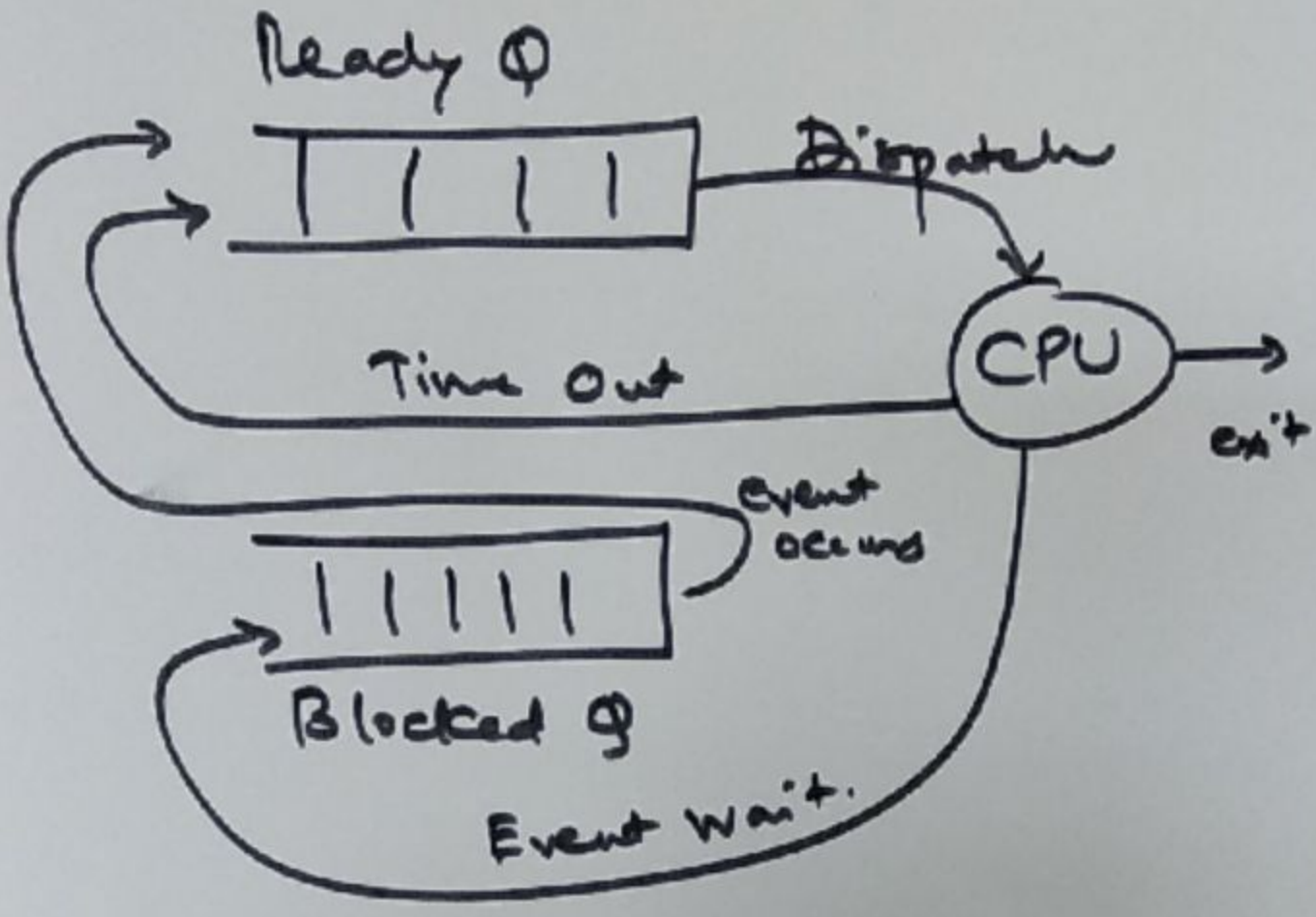
Implementation model — Queue model.

Ready state — k process — Ready Q

Blocked state — k process — Blocked Q

Run — No Q.





State model = states + transition

Queuing model = state model replacing states with specific Q