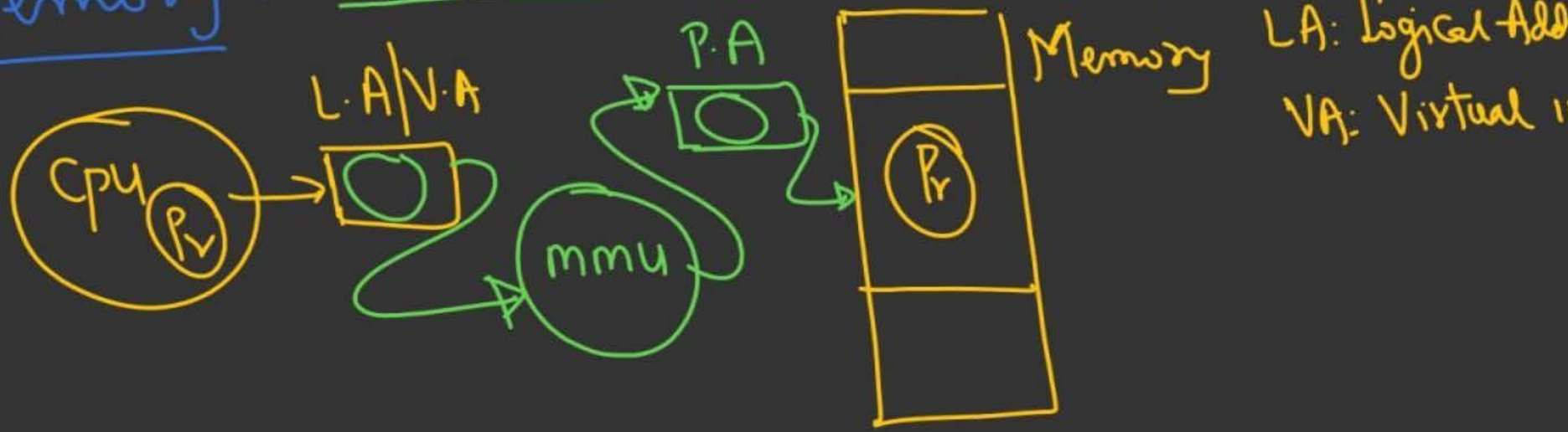
* Architectural Requirement for Implementation of a PreEmptive based M.Pr.O.S

H/W Support

1) IO (Secondary Storage devices) : {DMA} (Direct Memory access)

mmu: Mem Mgmnt unit

2) Memory : Address Translation Support



LA: Logical Addrs

VA: Virtual "

Imp

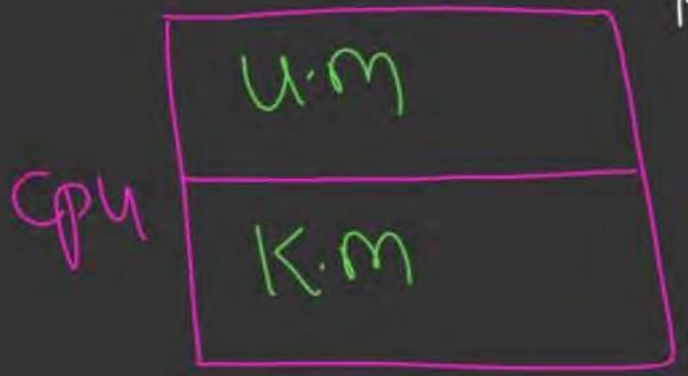
3) cpu : Dual Mode operation

cpu: central processing unit

(**)

: Every Processor in a M.Pr O.S must Support Imp Modes (atleast) for M.Pr as

- (UM) User Mode — Non Privileged
- Kernel Mode (KM) — Privileged — System



User Mode

1. User Programs/appl's run in User Mode;
2. User Mode execution is PreEmptive (Non-atomic)
3.

Kernel Mode

1) O.S programs/routines run in Kernel Mode;
2) Kernel Mode execution of OS programs is Non-PreEmptive (atomic)

1. Road Transport

- Ambulance (km) OS
- Common Man Vehicle (cm)

(user appl.)

2. Convoy of P.m/c.m

OS
Progs

Res

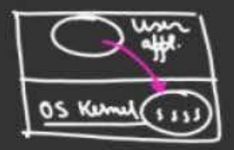
P/A

⑧ Mode Shifting (MSB)

→ O.S is a Service Provider

→ Users & Programmers are Service users

→ Many a times during the execution of user Programs, it is need to access/avail OS Service



→ Mode Shifting from UM → KM, whenever an User appl needs an OS Service;

```
main() UM
{
  int a, b, c;
1. a = 1;
2. b = 2;
3. c = a+b;
4. f(c);
}
```

User-level instrns

UM

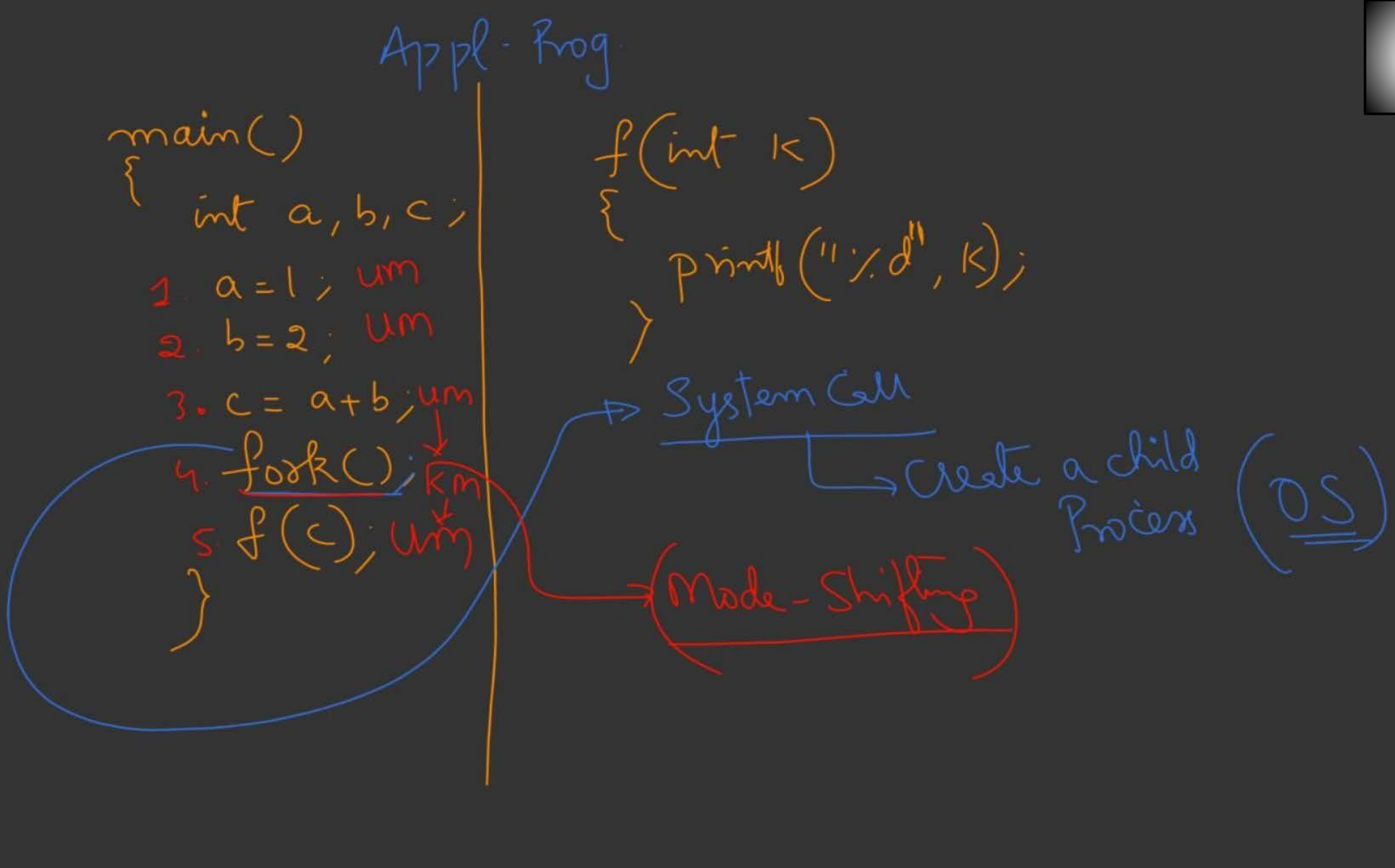
↓

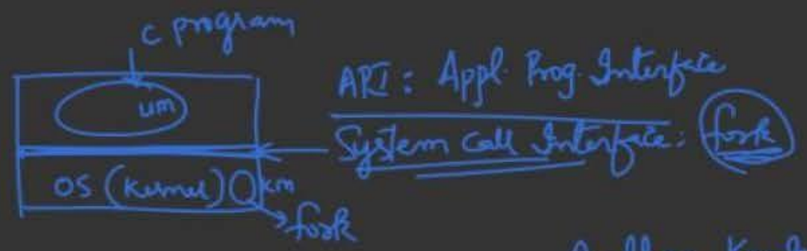
Predefined In-built Library

User Defined

```
f(int k)
{
UM printf("%d", k);
```

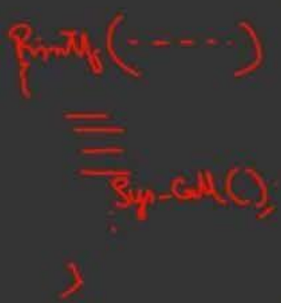
→ All user-defined & Predefined fns execute (Start) in UM.


Appl. Prog
main()
{
int a, b, c;
1. a=1; um
2. b=2; um
3. c= a+b; um
4. fork(); km
5 f(c); um
}
f(int k)
{
printf("%d", k);
}
System Call
Create a child Process
(OS)
Mode-Shifting



API : Appl. Prog. Interface

System call Interface : fork



" User-Defined & Pre-Defined functions may also activate/use System Calls "