CS & IT ENGINEERING

Operating Systems

Introduction & Background

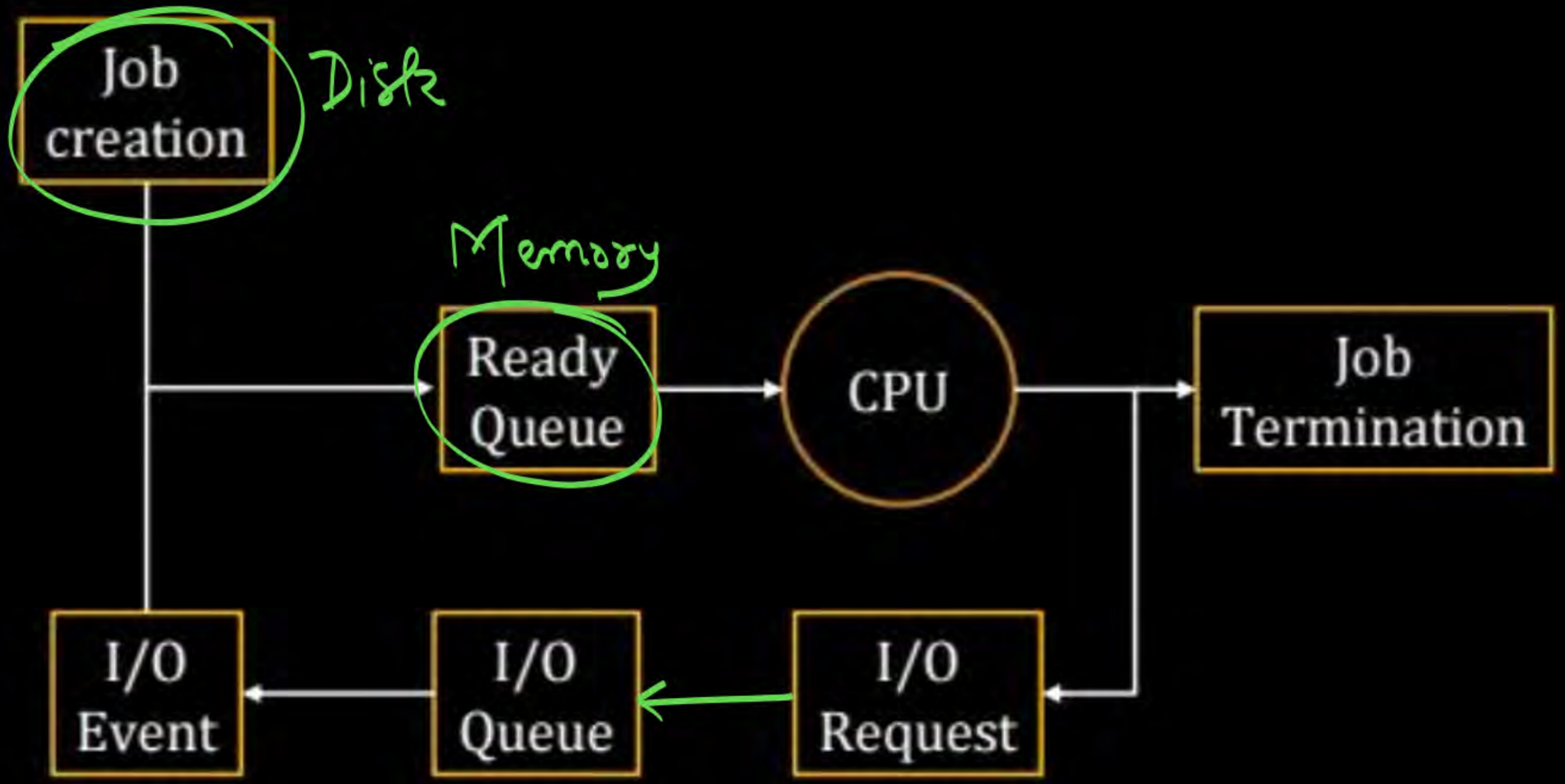
Lecture No. 3

By- Dr. Khaleel Khan sir

- Architectural Support for M.Pr
- Dual Mode Operation of CPU
- Mode Shifting

Program VS Process

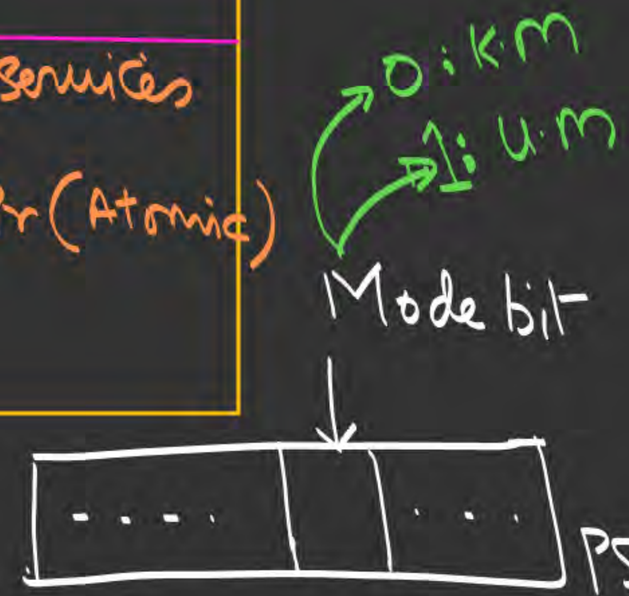
Multiprogramming
A Schematic view of Multiprogramming

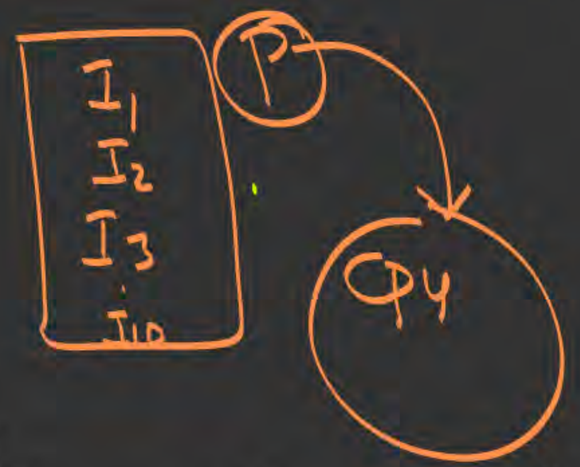


(iii) Dual Mode operation :

- User Mode (U.M) / Non-Privileged
- Kernel Mode (K.M)
- Privileged Mode
- Protected Mode
- System ||

U.M	1. user appl's runs in U.M 2. U.M is PreEmptive (Non-Atomic)
K.M	1. O.S Runtimes / Services runs in K.M 2. K.M is Non-Pr (Atomic)

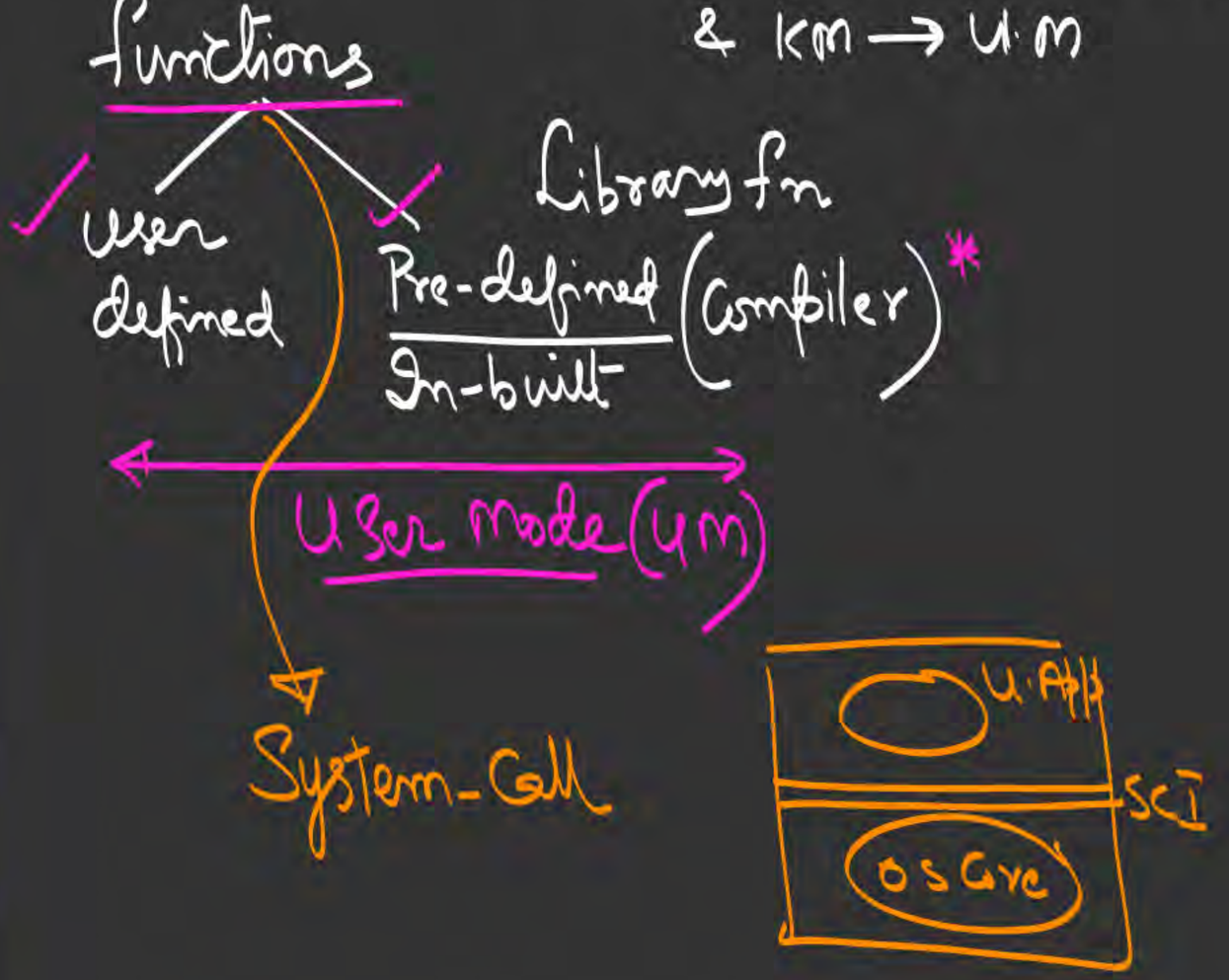


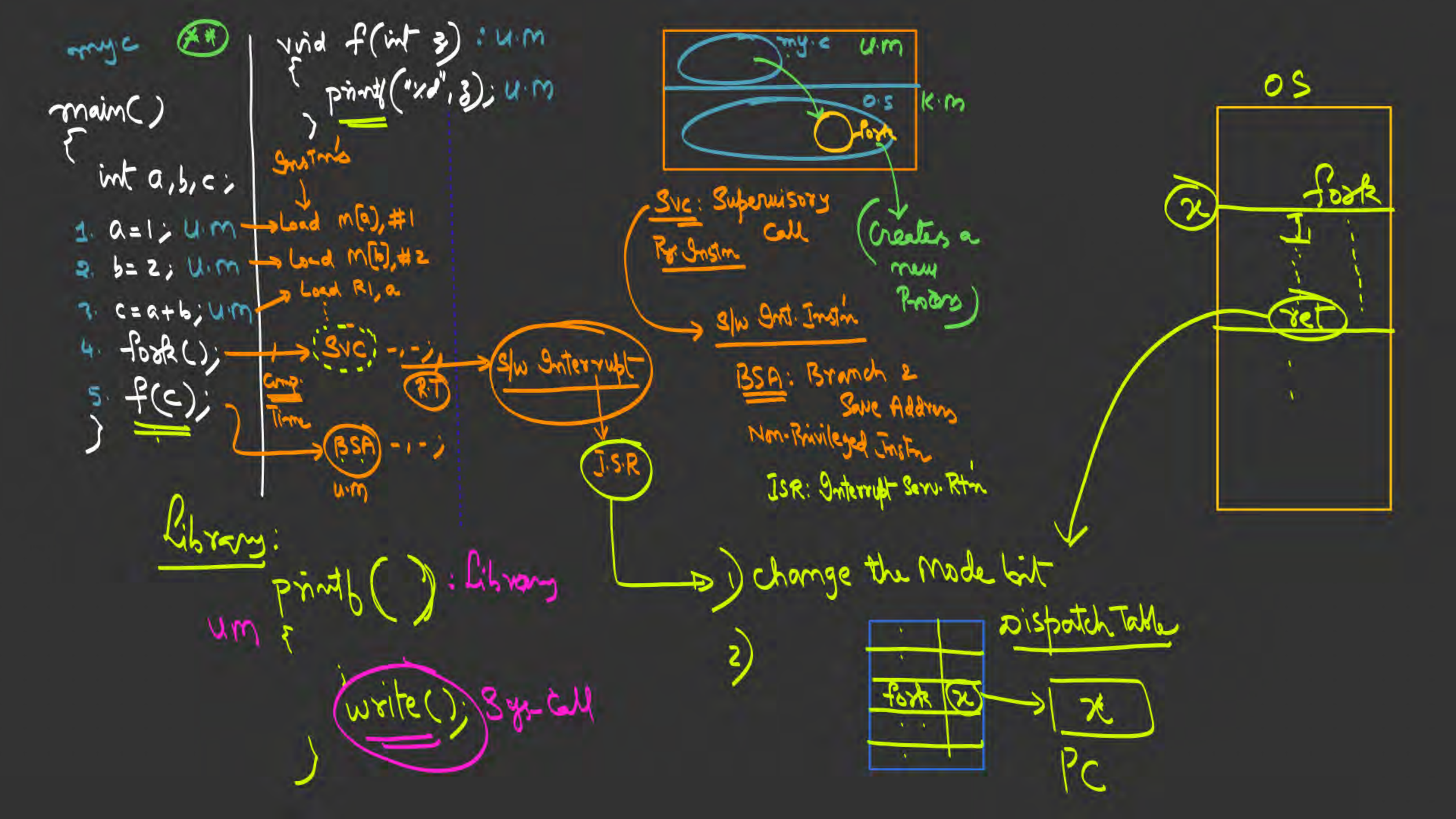


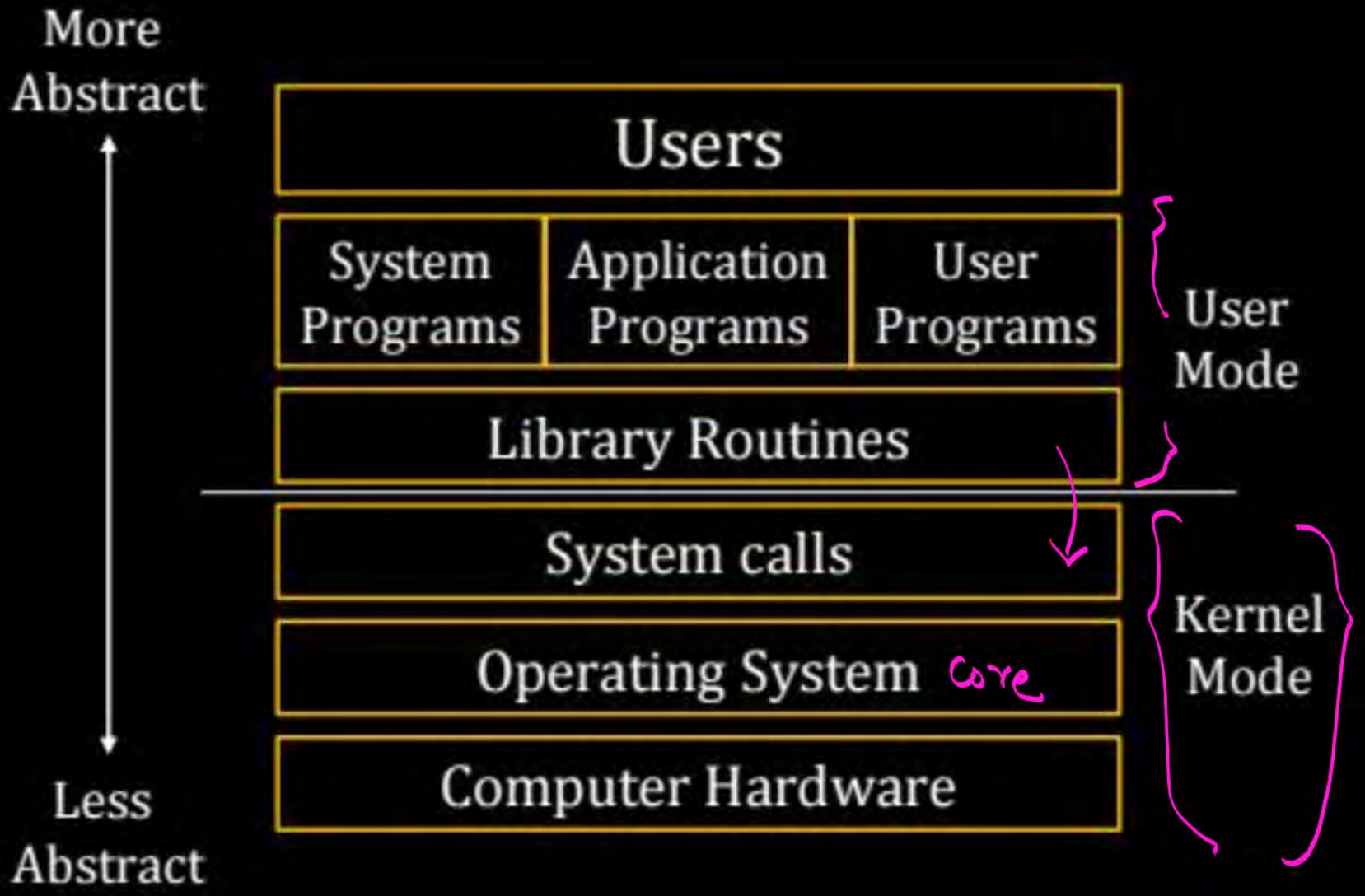
Mode Shifting: → O.S is a Service Provider & user appl's may need O S Services(K·m); ∴ Mode Shifting from um → km & km → um

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

```
f(int k)
{
  Sim(k);
  :
  printf(k);
}
```




my.c
main()
{
int a,b,c;
1. a=1; U.m
2. b=2; U.m
3. c=a+b; U.m
4. fork();
5. f(c);
}
void f(int z) : U.m
{
printf("%d",z); U.m
}
Instrns
Load m(a),#1
Load m(b),#2
Load R1,a
SVC
Comp. Time
BSA
U.m
R.I
S/w Interrupt
J.S.R
my.c U.m
O.S
K.m
fork
SVC: Supervisory Call
Pvg. Instrn.
(Creates a new Process)
S/w Instr. Instrn.
BSA: Branch & Save Address
Non-Privileged Instrn
ISR: Interrupt Serv. Rtn
OS
fork
ret
Library:
printf( ) : Library
U.m
write( ); Sys. Call
1) change the mode bit
2)
dispatch Table
fork
PC

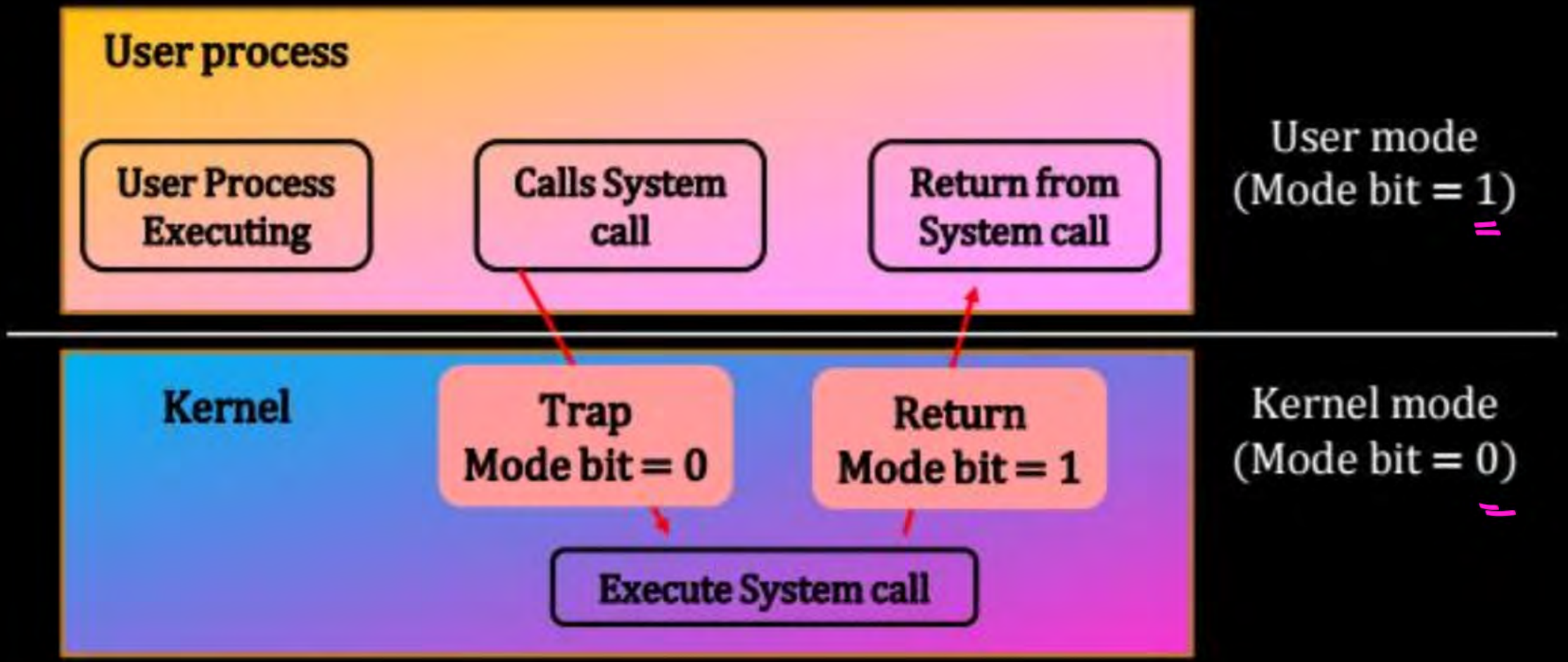

More Abstract
Users
System Programs
Application Programs
User Programs
User Mode
Library Routines
System calls
Operating System
Core
Kernel Mode
Computer Hardware
Less Abstract

1. Kernel Mode

   In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

2. User Mode

   In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.


User process
User Process Executing
Calls System call
Return from System call
User mode (Mode bit = 1)
Kernel
Trap Mode bit = 0
Return Mode bit = 1
Execute System call
Kernel mode (Mode bit = 0)

Q.1

A Processor needs Software Interrupt to

A. Test the Interrupt System of the Processor.

B. Implement Co-Routines.

C. Obtain system services which need execution of privileged instructions.

D. Return from subroutine.

Q.2

A CPU has two Modes-Privileged and Non-Privileged. In order to change the mode from Non- Privileged to Privileged.

A. A Hardware Interrupt is needed.

B. A Software Interrupt is needed.

C. A Privileged Instruction (which does not generate an interrupt) is needed.

D. A Non - Privileged Instruction (which does not generate an interrupt) is needed.

Q.3

System Calls are usually invoked by using:

A. A Software Interrupt

B. Polling

C. An Indirect jump

D. A Privileged Instruction.

Q.4

A computer handles several interrupt sources of which the following are relevant for this question:

- Interrupt from CPU temperature sensor (raises interrupt if CPU temperature is too high)
- Interrupt from Mouse (raises interrupt if the mouse is moved or a button is pressed)
- Interrupt from Keyboard (raises interrupt if a key is pressed or released)
- Interrupt from Hard Disk (raises interrupt when a disk read is completed)

Which one of these will be handled at the HIGHEST priority?

A Interrupt from Hard Disk

B Interrupt from Mouse

C Interrupt from Keyboard

D ✓ Interrupt from CPU temperature

Program vs Process

(.exe Program)

- Instructions (Code)
- Data
  - Static
    - → Fixed (Known) Size
    - → Alloc b/f Run Time
  - Dynamic
    - → Unknown Size (Variable Size)
    - → Alloc @ Run Time

my.c

```
main()
{
  int a, b, c;
  a = 1;
  b = 2;
  c = a + b;
}
```

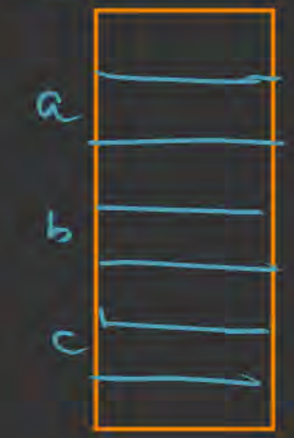
Data

Static Data

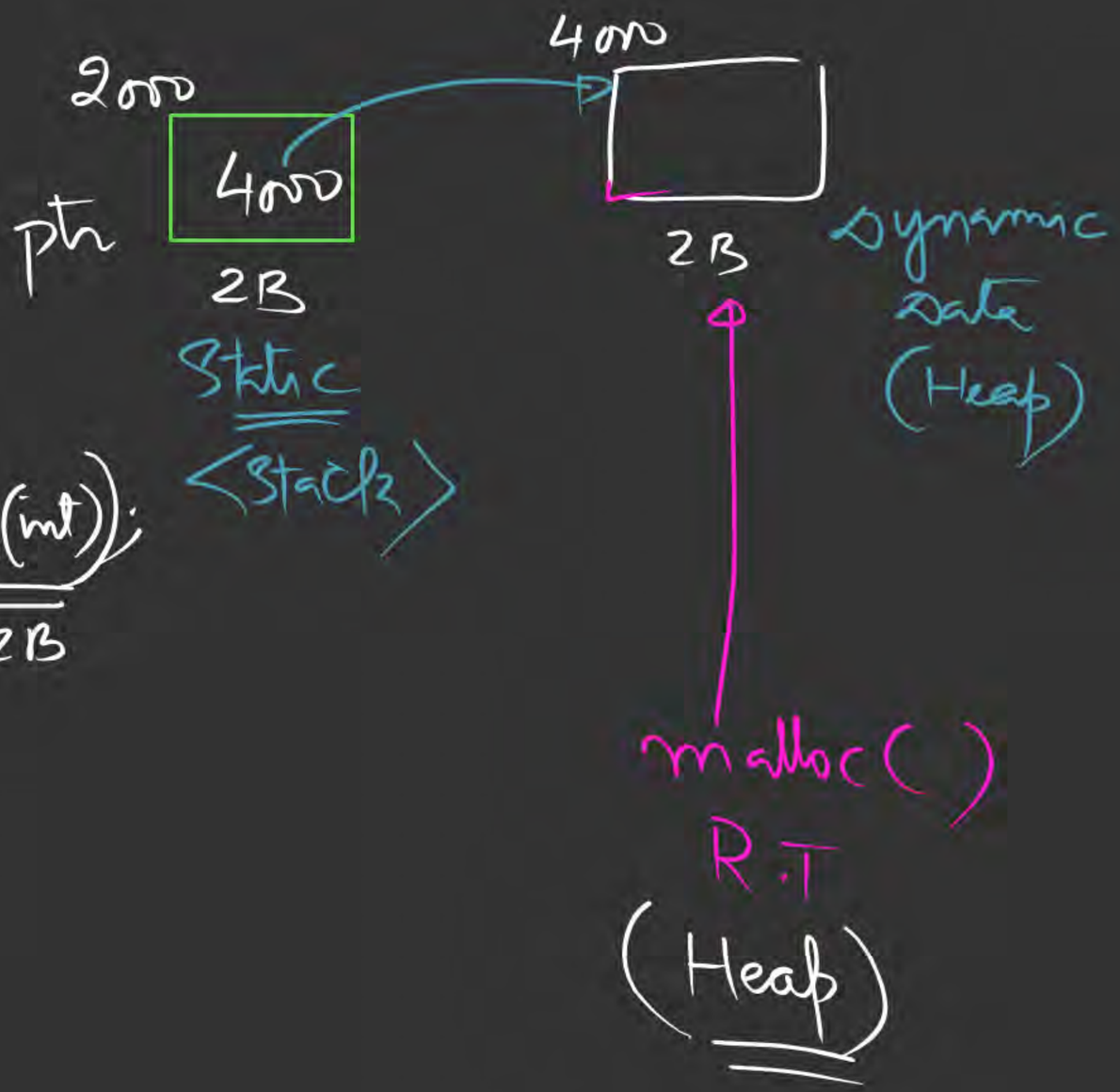
M/C lang. (.exe Prog.)

Code / Instns

```
Load  a, #1
Load  b, #2
Load  R1, a
Load  R2, b
Add   R1, R2
Store c, R1
```



```
main()
{
    int *pth;
    pth = (int*) malloc(sizeof(int));
}
```




THANK YOU
GW
SOLDIERS !