COMPUTER SCIENCE & IT

OPERATING SYSTEM

Process Management

Lecture-01

Dr. KHALEEL KHAN

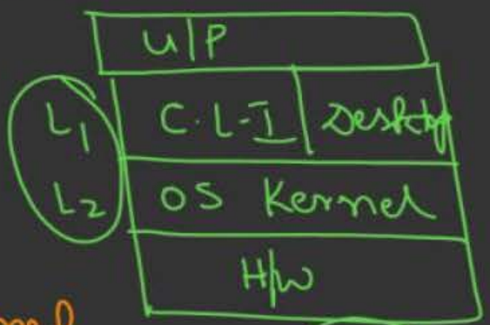
Today's Goal

"Implementation of system call" Process Concept, Program v/s Concept"

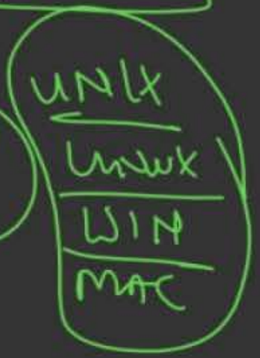
Recap:

→ what is O.S (Interface)

→ Functions & Goals
- Convenience
- L_1
- L_2



→ Types of O.S:
- Uniprog.
  - DOS
- Multiprogrammed
  - N.Pr
  - Pr



Principles & Concepts Pr. based M.Pr O.S

Architectural Requirement

UNIX | WIN
MAC

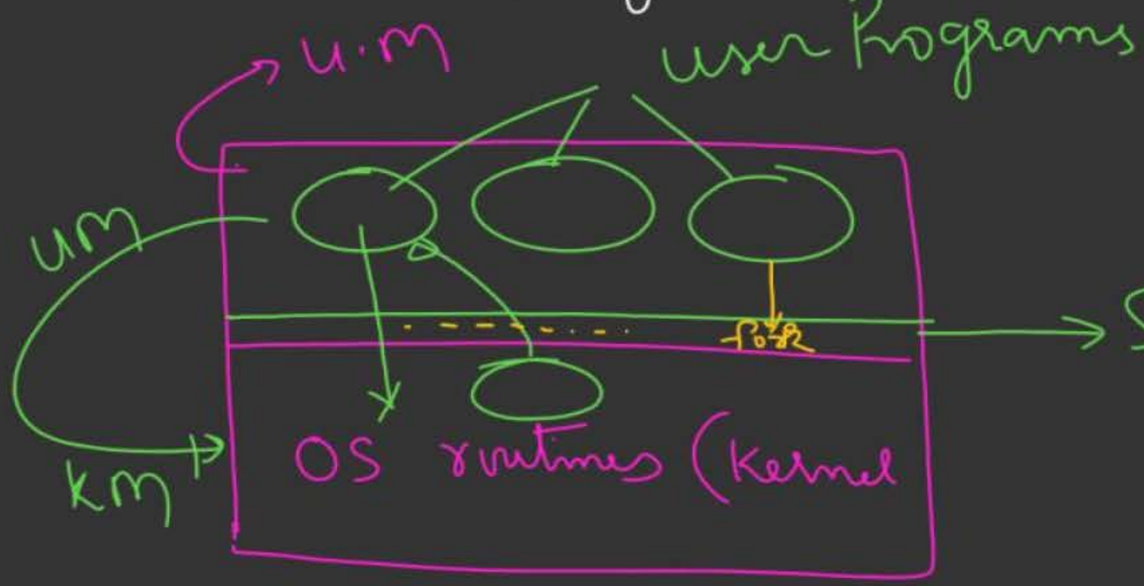
(i) I/O (Sec. Storage) : DMA

(ii) Mem : Address Translation
MMU

(iii) CPU : Dual Mode :

- Mode bit
- (Pr. Mode)
- KM
- OS programs
- Atomic
- UM
- user programs
- Pre Emptive
- Non-Atomic

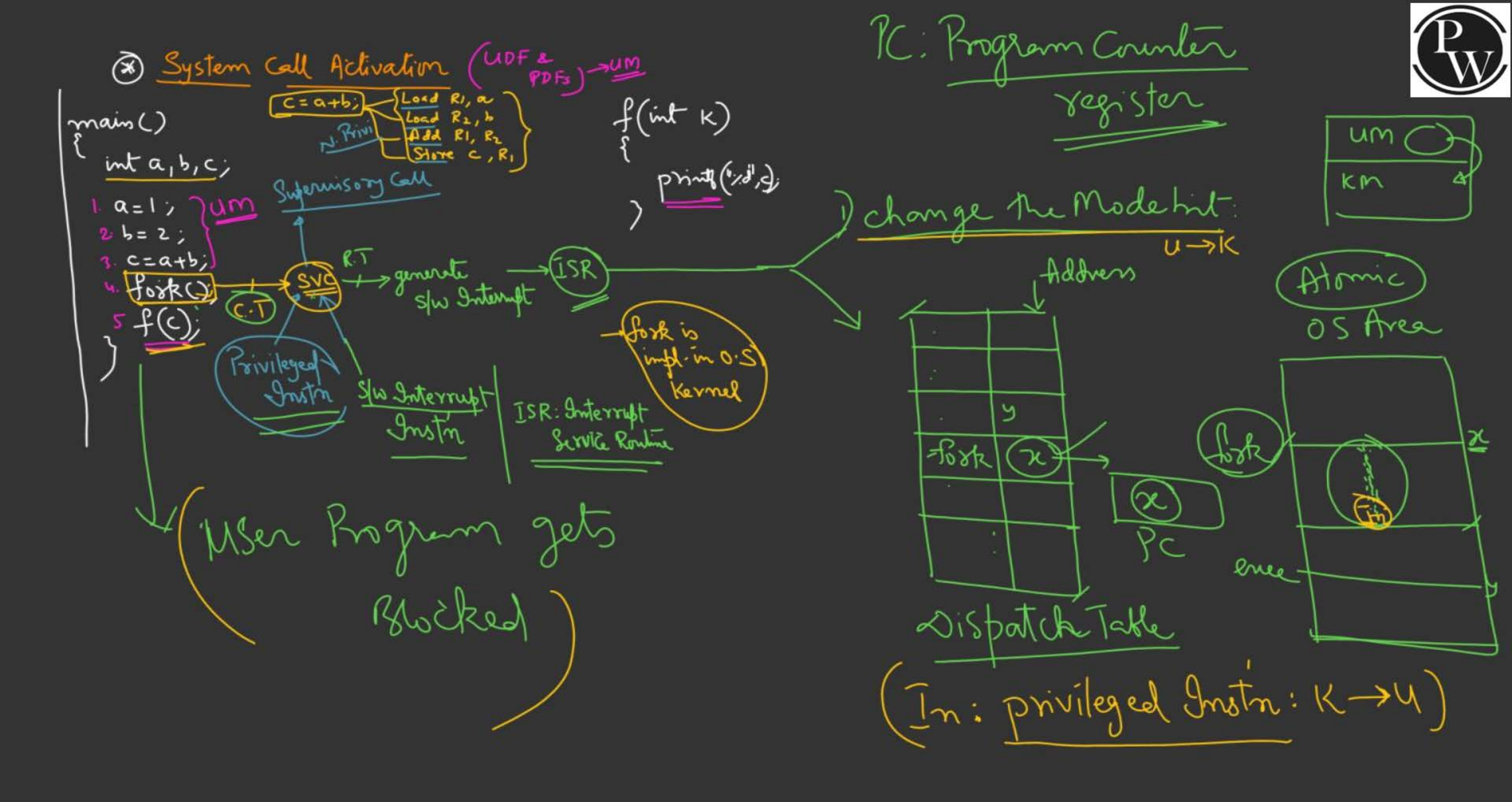
⊛ Mode Shifting :



fork() : create a Process

System call

is an OS Service


System Call Activation (UDF & PDFs) → UM
main()
{
int a, b, c;
1. a=1;
2. b=2;
3. c=a+b;
4. fork();
5. f(c);
}
UM
C.T
SVC
Supervisory Call
Privileged Instn
S/w Interrupt Instn
c=a+b;
Load R1, a
Load R2, b
Add R1, R2
Store c, R1
N. Privi
f(int k)
{
printf("%d", c);
}
R.T
generate s/w Interrupt
ISR
ISR: Interrupt Service Routine
fork is impl. in O.S kernel
(User Program gets Blocked)
PC: Program Counter register
1) change the Mode bit: u→k
Address
y
fork
x
PC
Dispatch Table
um
km
Atomic
OS Area
fork
x
In
exec
(In: privileged Instn: K→U)

Important points

1. Sys. Calls are activated thru a privilegeed Instn (S/w Int. Instn) that generate S/w Interrupt
2. (ISR) changes the mode bit (U → K) & locate the Sys. Call thru dispatch table.
3. User Process gets Blocked
4. Sys. Call routines executes atomically
5. Last Instn of Sys. Call, changes the mode (K → U) & unblock the user program that activated the Sys. Call

Interrupts
- H/W : generated by H/w devices
  CPU ; IOD
- S/W : execution of Instn's

c = a / b ;

b = 0

divide overflow

S/w Int.

II. PROCESS MANAGEMENT (Mgmnt of CPU)

1. Process Concepts:

→ Program vs Process?

Program
- Machine language Program
- executable
- (.exe)

Process
- (Code) Instns
- Data

int a, b, c;

Data

C = a+b → HLL →
- Load R1, a
- load R2, b
- Add R1, R2
- Store c, R1

(Instns)

STATIC: fixed size & Known

DYNAMIC: Not known size & Alloc takes place @ R.T

Mem alloc takes place b/f R.T (load time)

Program
Job
Application
Task

Synonymous

c/c++

```
int n, A[n];
scanf("%d", &n);
```

X

Dynamic array

Can we Implement a dynamic array thru other means in c/c++

↳ ptrs & DMA

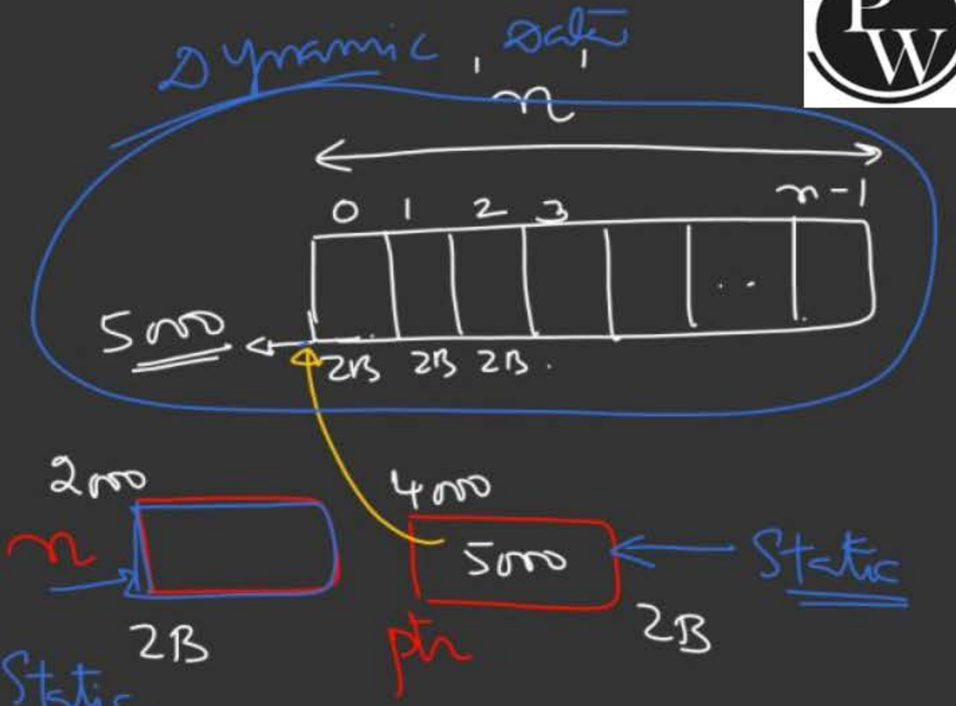
malloc

```
int n, *ptr;
scanf("%d", &n);
ptr = (int *) malloc(sizeof(int) * n);
```

int 2B

n = 5

Dynamic (R.T)



int = 2B

Assumption

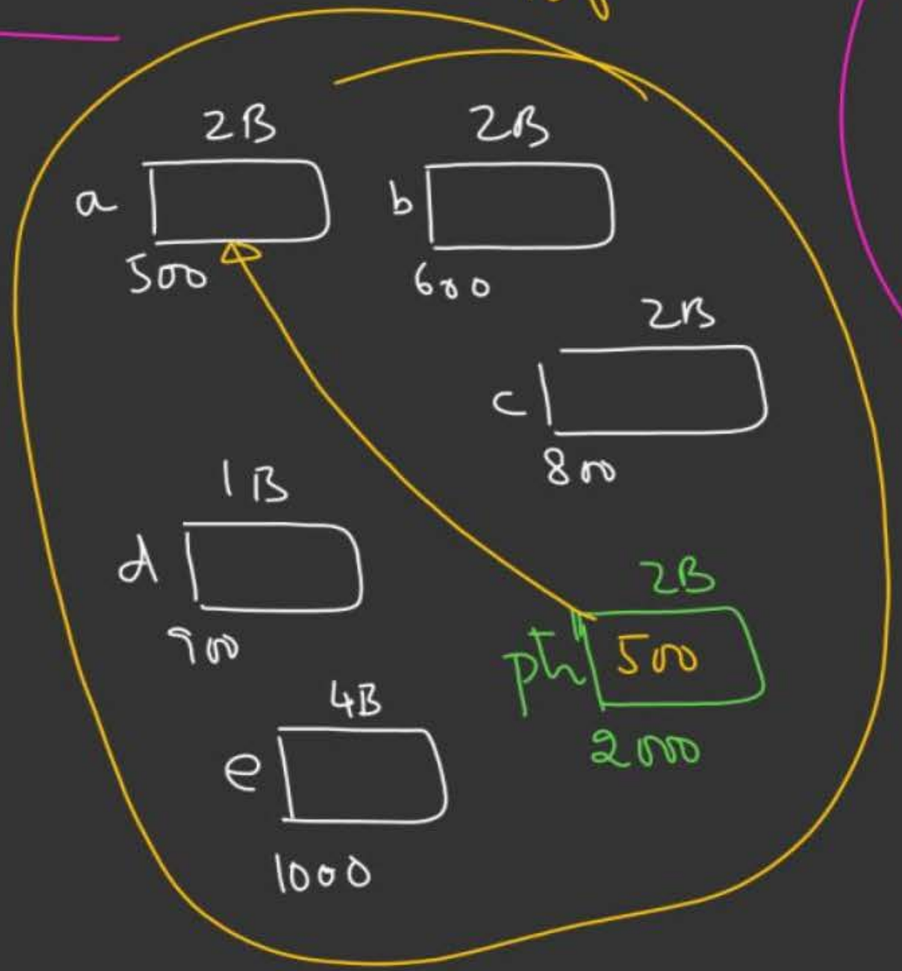
pointer is 2B

Total No of Bytes = 13 Bytes

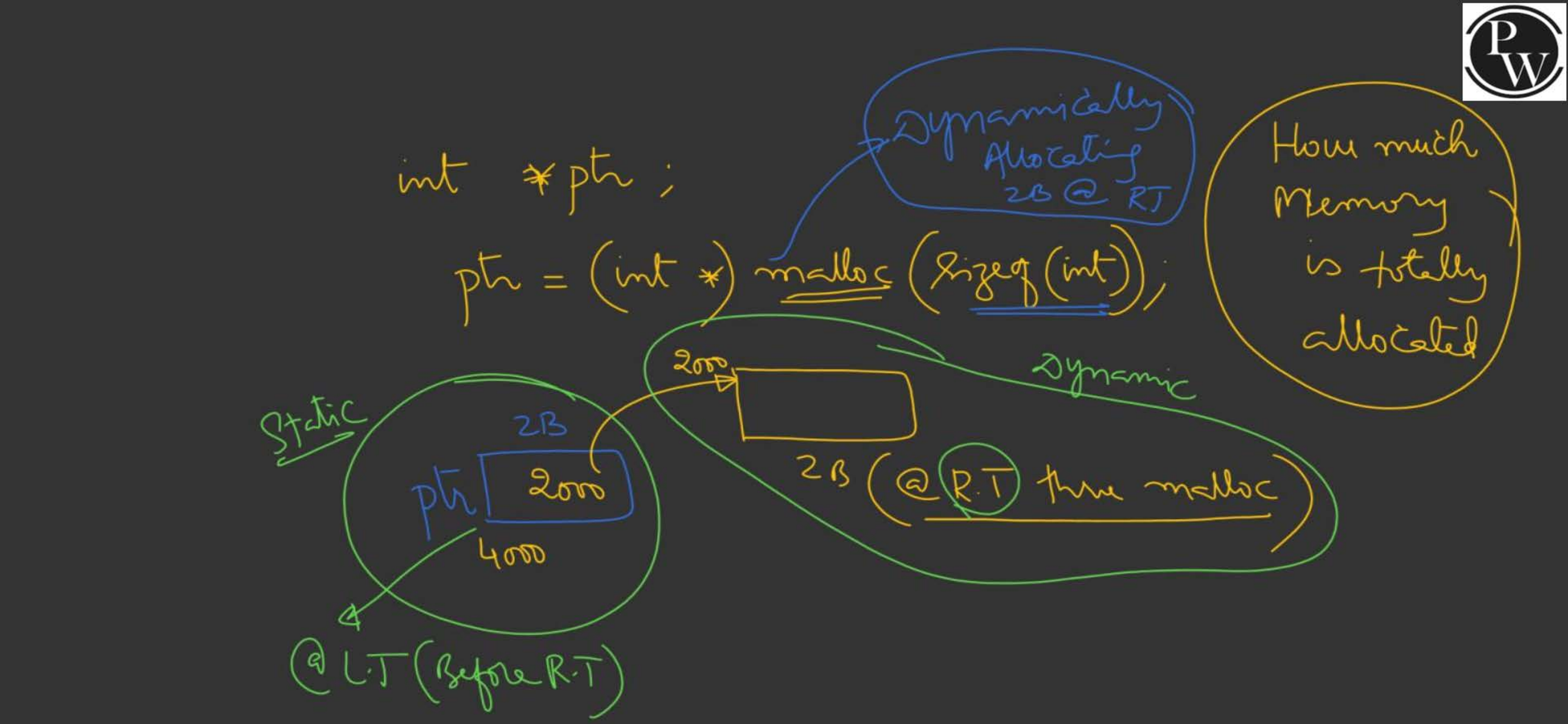
```
int a, b, c;
char d;
float e;
int *pth;
pth = &a;
```

hold address of an Integer

Before R.T



int = 2B
char = 1B
float = 4B
every pointer = 2B


int *ptr;
ptr = (int *) malloc (sizeof (int));
Dynamically Allocating 2B @ RT
How much Memory is totally allocated
Static
2B
ptr
2000
4000
@ L.T (Before R.T)
2000
Dynamic
2B (@ R.T thru malloc

1) Known Size + S.A

2) Known Size + D.A

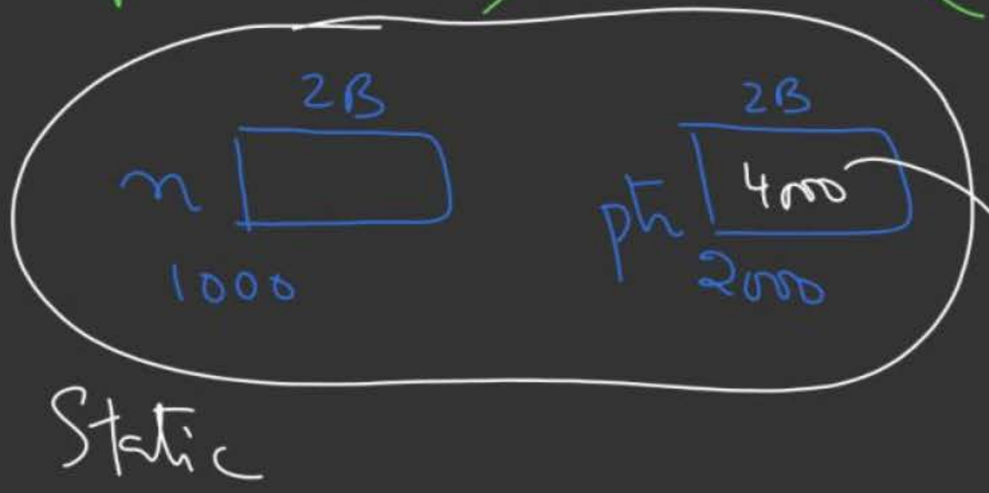
3) Unknown Size + DA

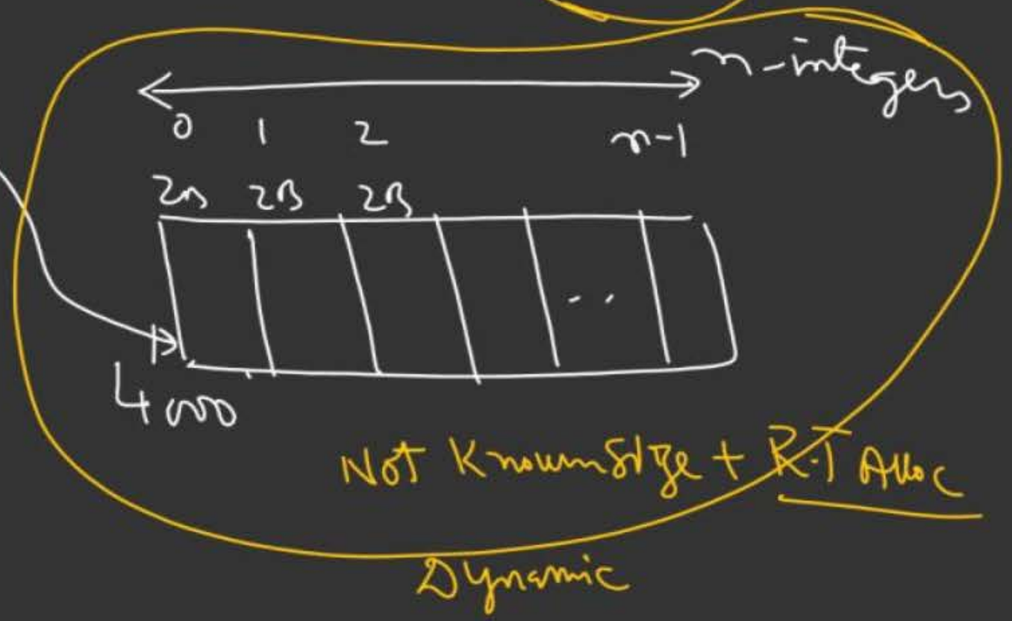
```
int n, *ptr;
scanf("%d", &n);
ptr = (int *) malloc (sizeof(int) * n);
```

dynamic

sizeof(int) = 2B

2B * n = 2nB





THANK YOU!