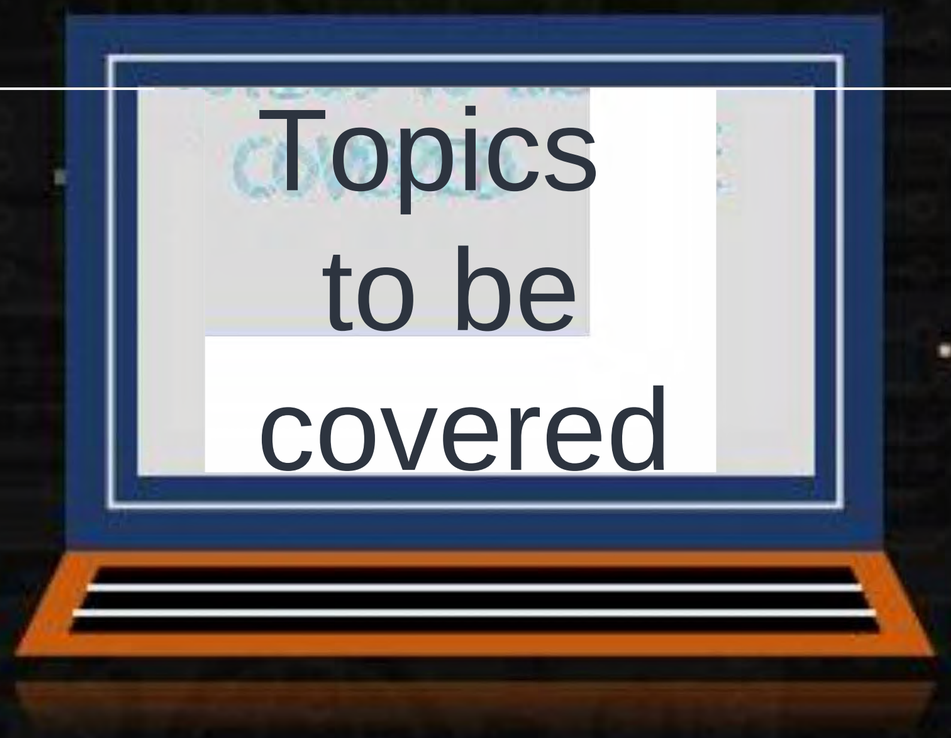
CS & IT ENGINEERING

Operating System

System Calls and Threads Part-2

Lecture no:02

By- Dr. Khaleel Khan sir



System Calls and Threads Part-2

Common and well known system calls are:

- execl, execv, execle, execve, execlp, execvp: executes a file
- exit: exits a process
- fcntl: controls open files
- fork: creates a new process
- getpid, getpgrp, getppid: gets group and process IDs
- getuid, geteuid, getgid, getegid: gets user and group IDs
- ioctl: controls character devices
- kill: sends a signal to one or more processes
- link: links a new file name to an existing file
- lseek: moves read/write file pointer

System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Programs

- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Q.1

```
main ()
{
    int i, n;
    for (i = 1; i <= n; ++i)
    if (fork () == 0)
        print ("*");
}
```

child

"*" gets Printed

$= (2^n - 1)$

$(2^n - 1)$: "*"

```
for (i=1; i<=n; ++i)
    if (fork()==0);
```

printf("*"); || P+child

outside for loop

Total Processes = 2^n

Q.2

```
main ()
{
    int i, n;
    for (i = 1; i< = n; ++i)
    {
      fork ();
      print ("*");
    }
}
```

2

= ( )

H/w

Q3)

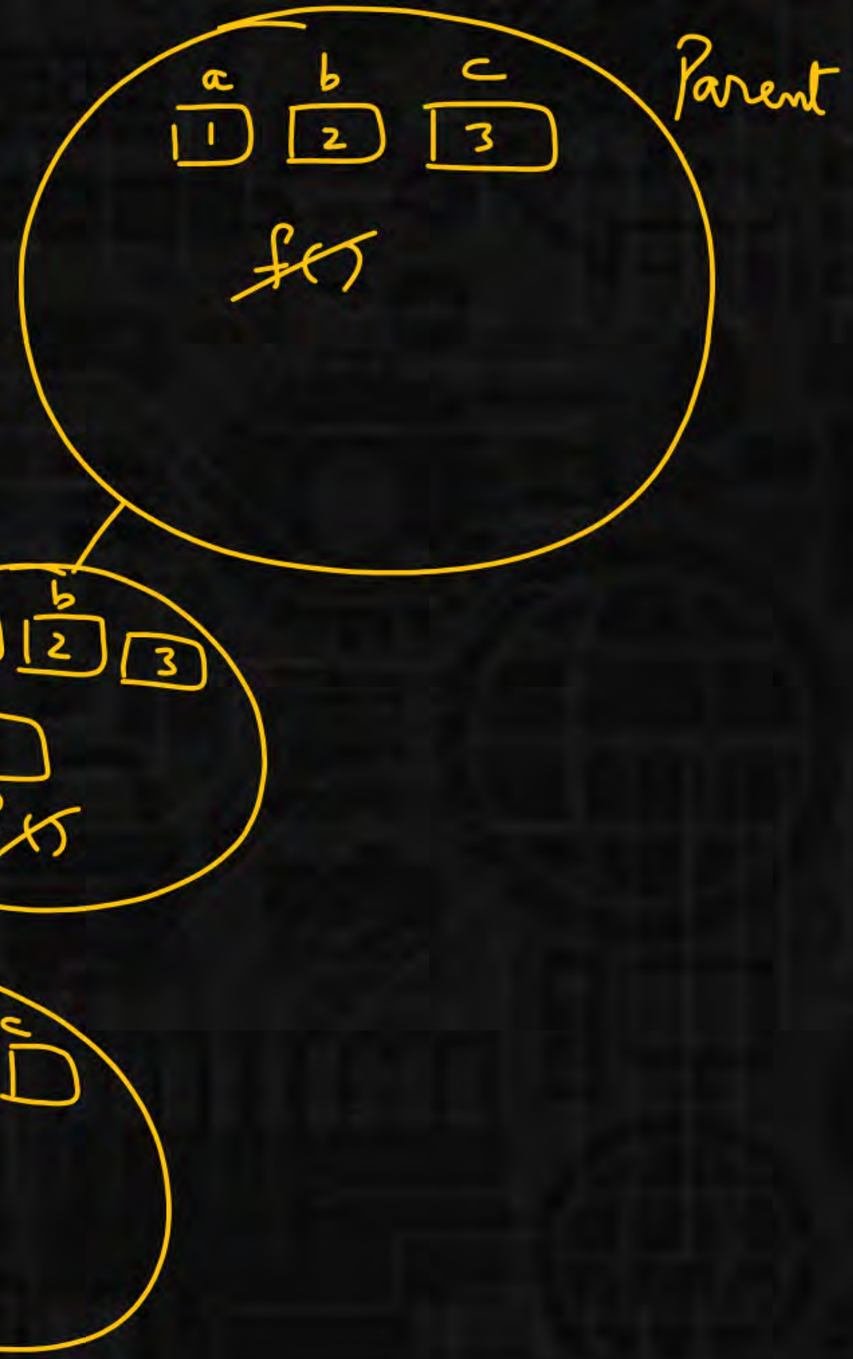
```
for(i=1; i<=n; ++i)
{
    printf("*");
    fork();
}
```

1

= ( )

Q.4

```
main ()
{
    int a = 1, b = 2, c = 3; d = 5;
    a + = b+ = + + c;
    print (a, b, c);
    if (fork () = = 0)
    {
    int d;
First + + a; + + b; – – c;
    print (a, b, c);
    if (fork () = = 0)
    {
See.
        d = a + b + c;
        print (a, b, c, d);
    }
```



```
        else
Parent  {
            - - a; - - b;
            c = a + b; d = a + b + c;
            print (a, b, c, d);
        }
        }
        else
Parent  {
        c + = b + = + + a;
        print (a, b, c) ;
        }
        print (d);
    }
```

Compilation-Error

Q.5 The following C program is executed on a Unix/Linux system:

```
#include <unistd. h>
int main( )
{        int i;
         for (i = 0; i < 10; i + +)
             if (i % 2 = = 0) fork( );
}
```

10 times (n=10)

5

The total number of child processes created is $(2^5 - 1) = 31$

Inverted Paging

→ Paging is a NCG Technique; → Ext. Fragmentation;

→ Paging Involves

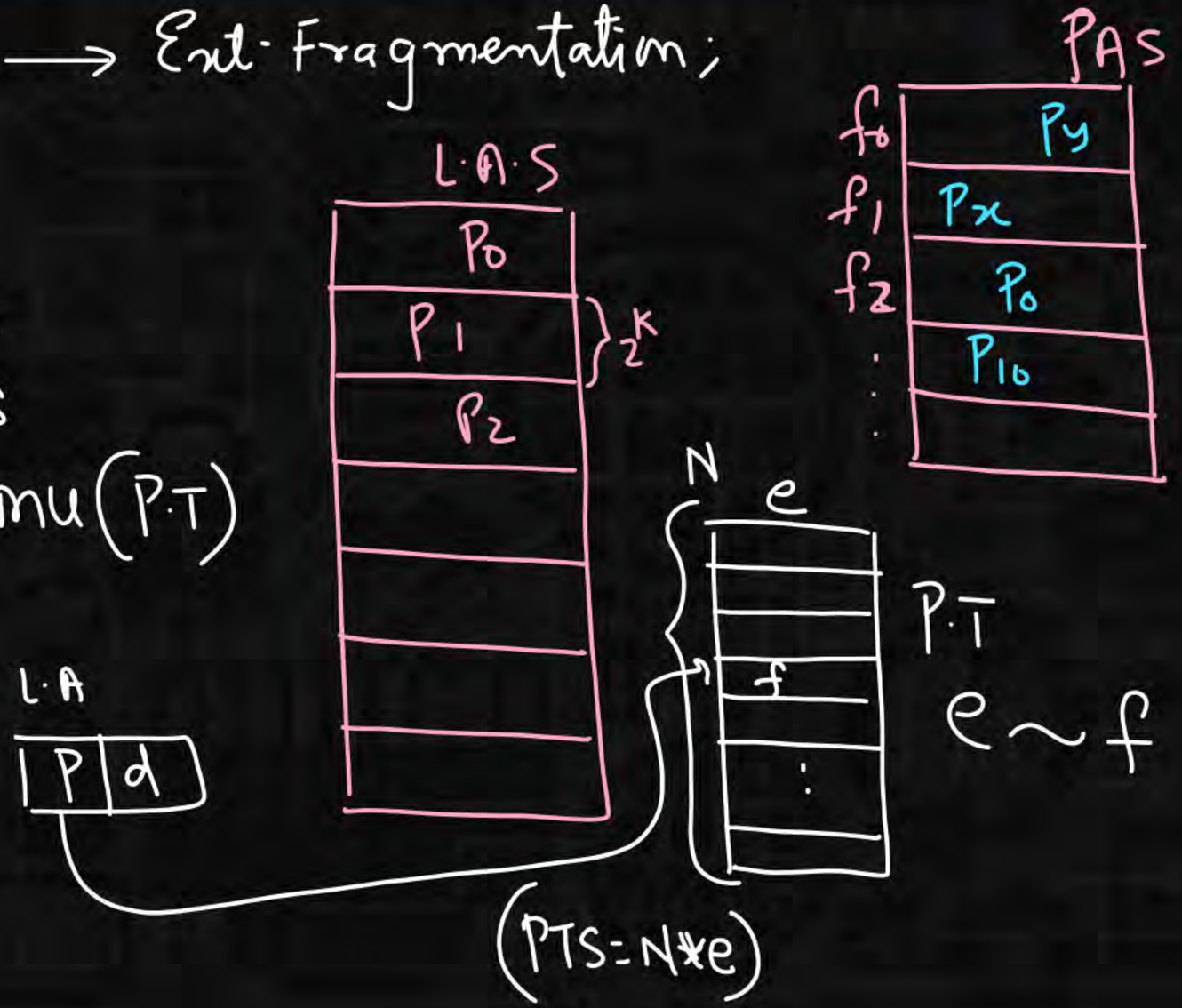
(i) orgnz. of LAS

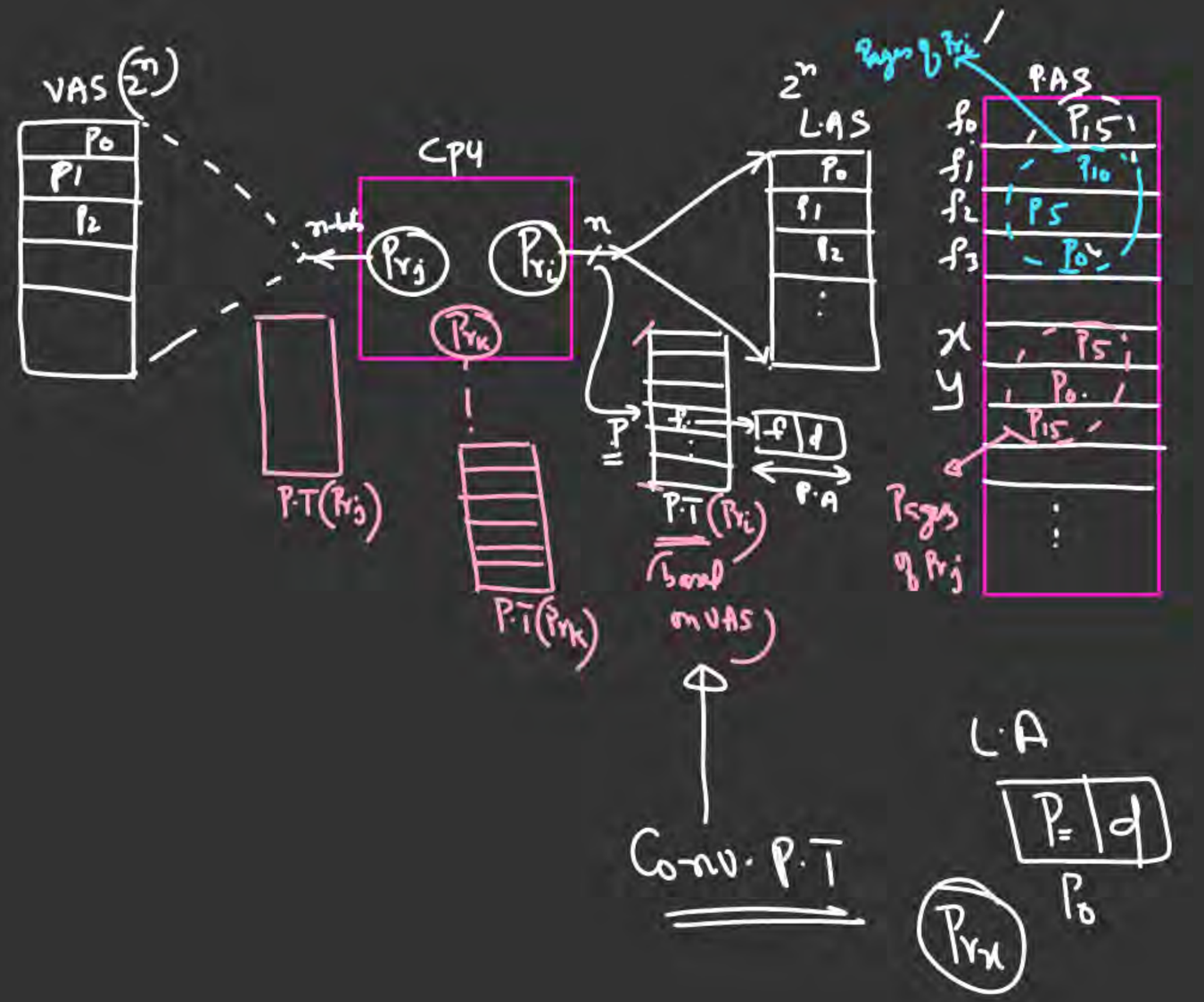
(ii) " " PAS

(iii) " " MMU (P.T)

→ Each Process has its own P.T

→ P.T's are Stored in m.m (PAS)





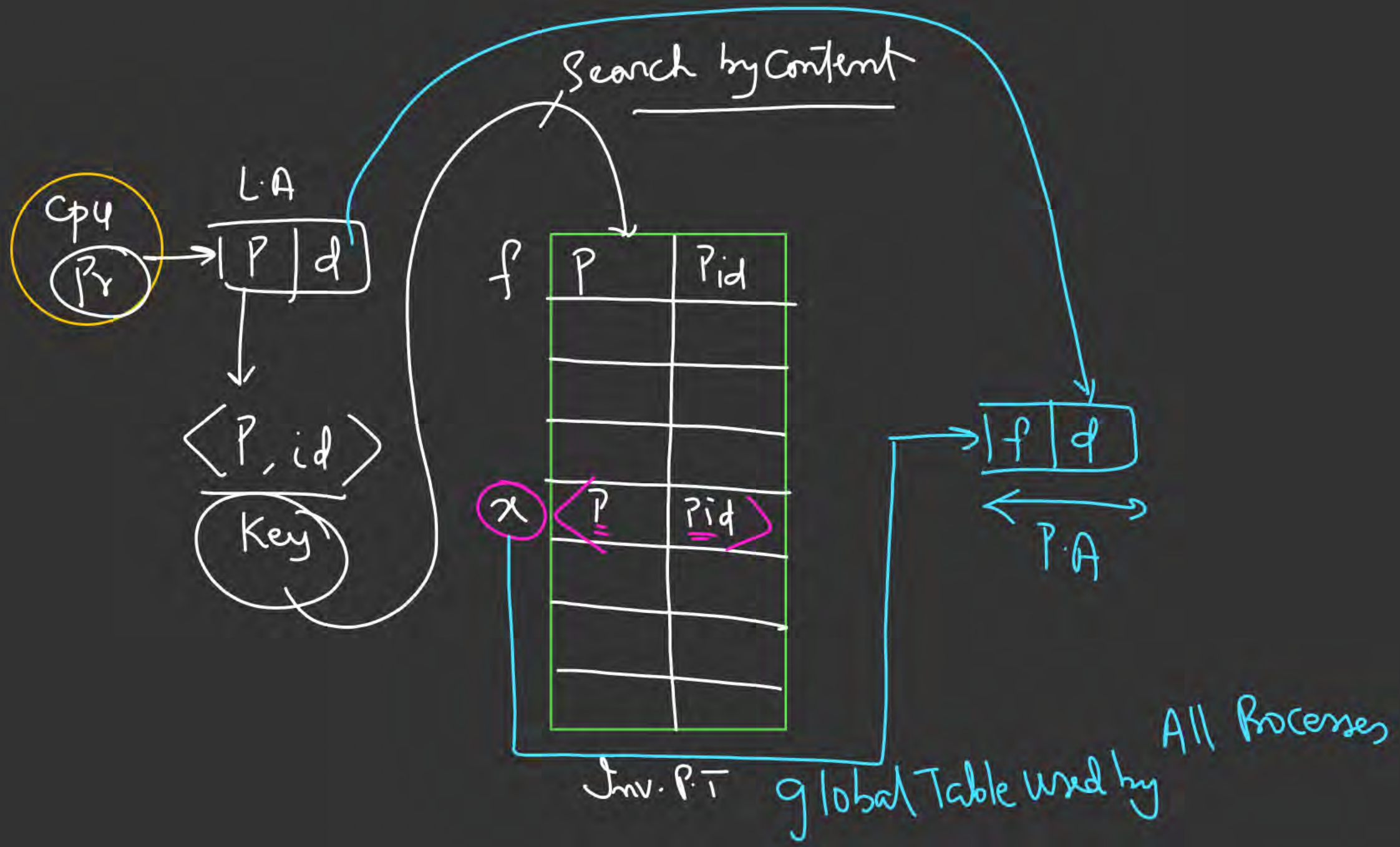
f | P | Pid

No. of Entries (F.T) = No. of frames

(Search by Content)

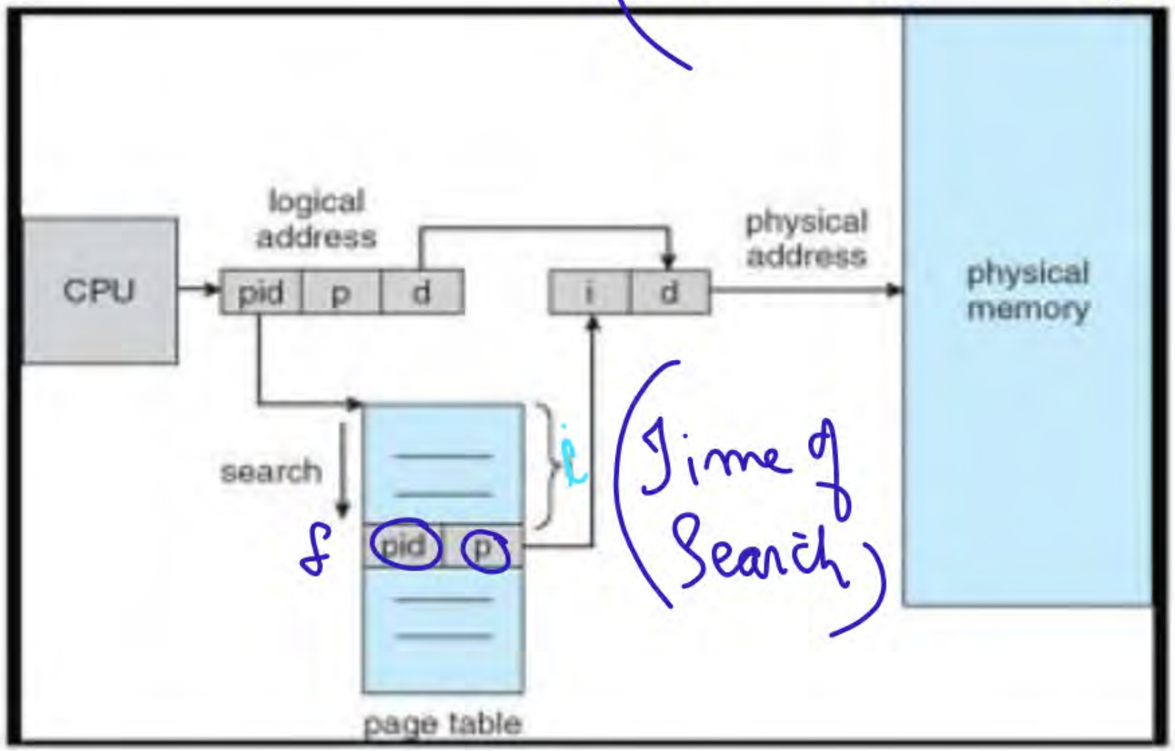
Frame Table

Inverted P.T


Search by Content
CPU
Pr
L.A
P
d
<P, id>
Key
f
P
Pid
x
P
Pid
f
d
P.A
Inv. P.T
global Table used by All Processes

Inverted Page Table

(Space efficiency)



(Time of Search)

L.A = 32 bits;
P.A = 26 bits; P.S = 4KB;
e = 4B;

Q1) Conv. P.T.S

400MB

$= N * e$

$= \frac{2^{32}}{2^{12}} = (2^{20}) \times 4B$

$= 4MB$

Q2) Inv. P.T.S

$= M * e = \left(\frac{2^{26}}{2^{12}}\right) = 2^{14} * 4B$

$= 16K * 4B$

$= 64KB$


THANK YOU
GW
SOLDIERS !