1

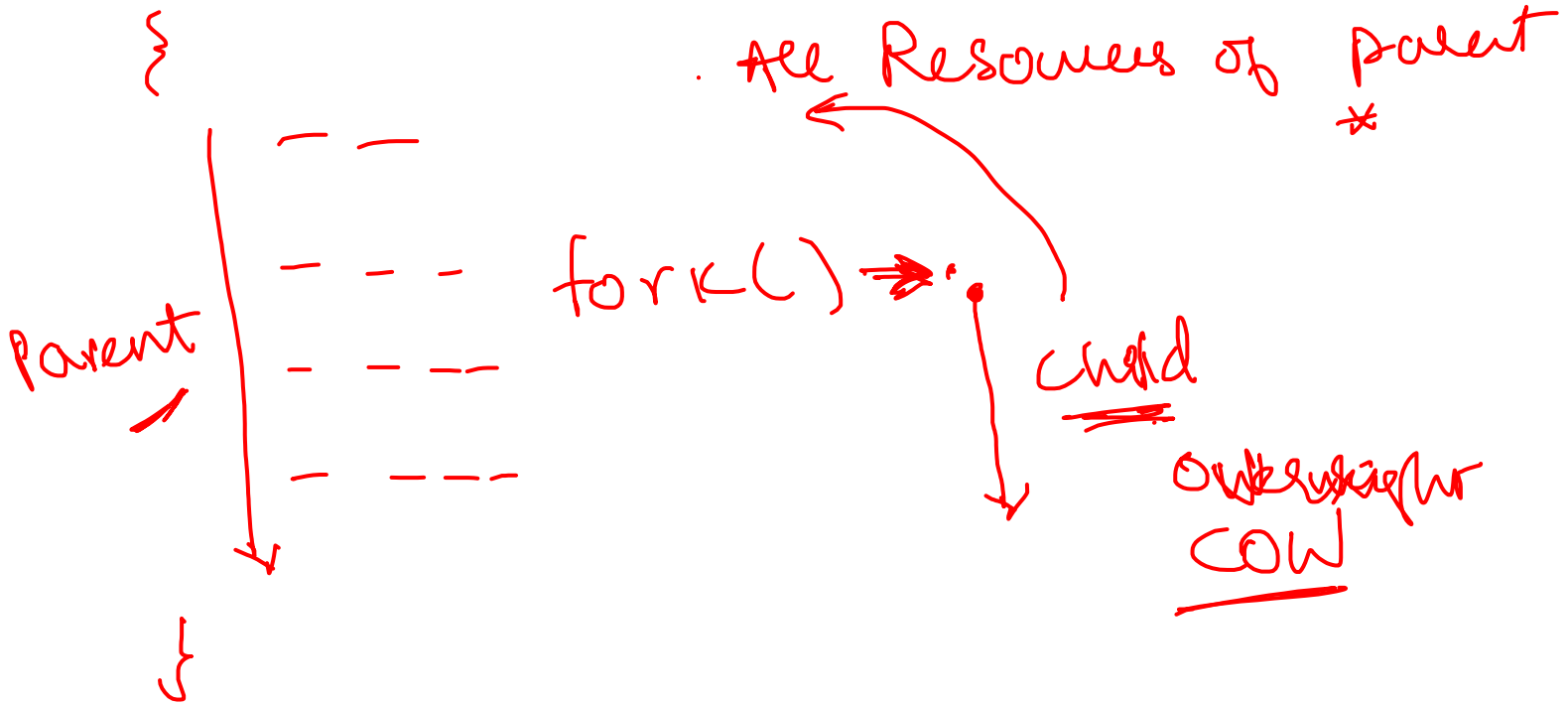
main.c → Program

./main ↔ process

→ Parent process

Process mgt

↳ fork()



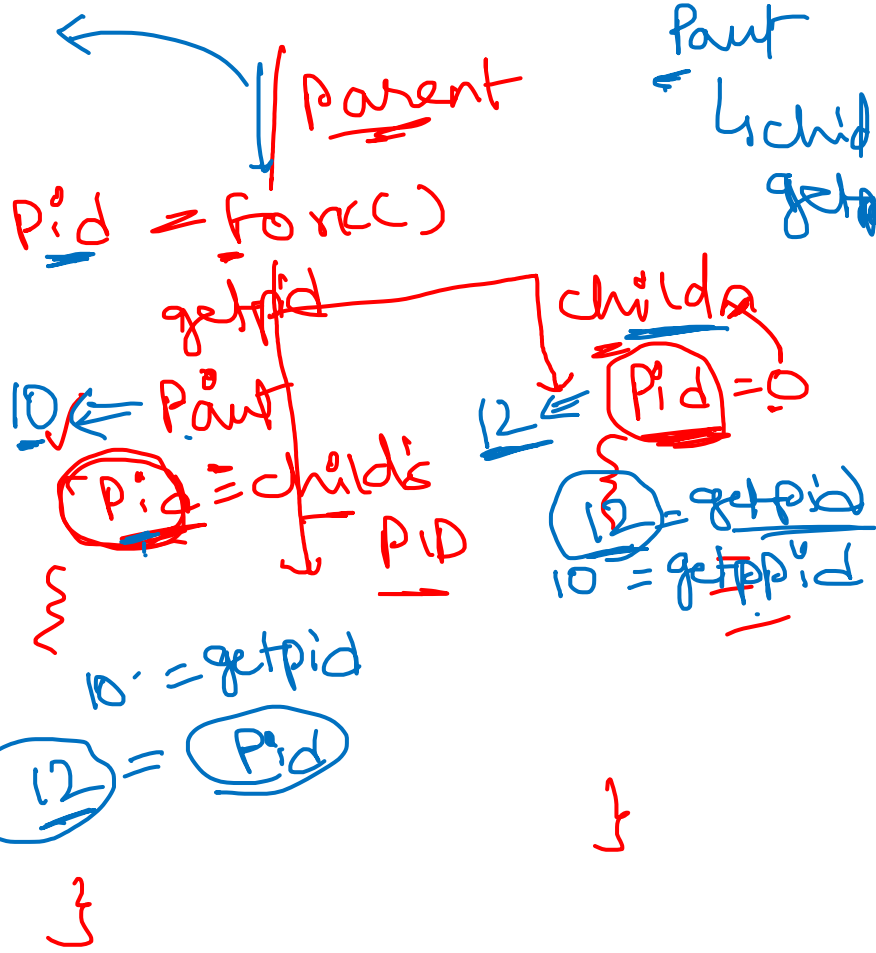
1.

{

Parent : Hello world "

fork

} printf(" Hello world "), Pid = fork()



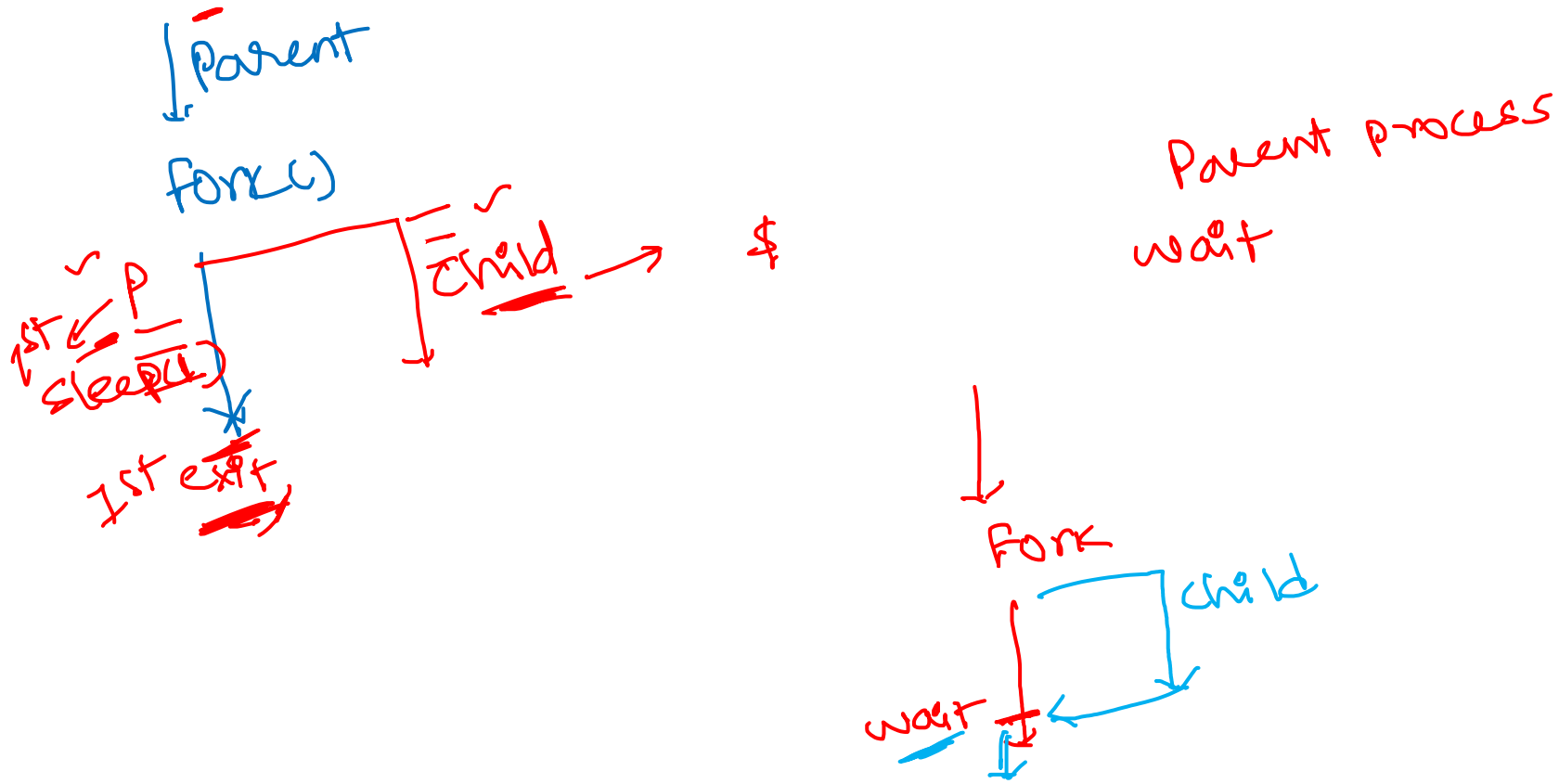
example1

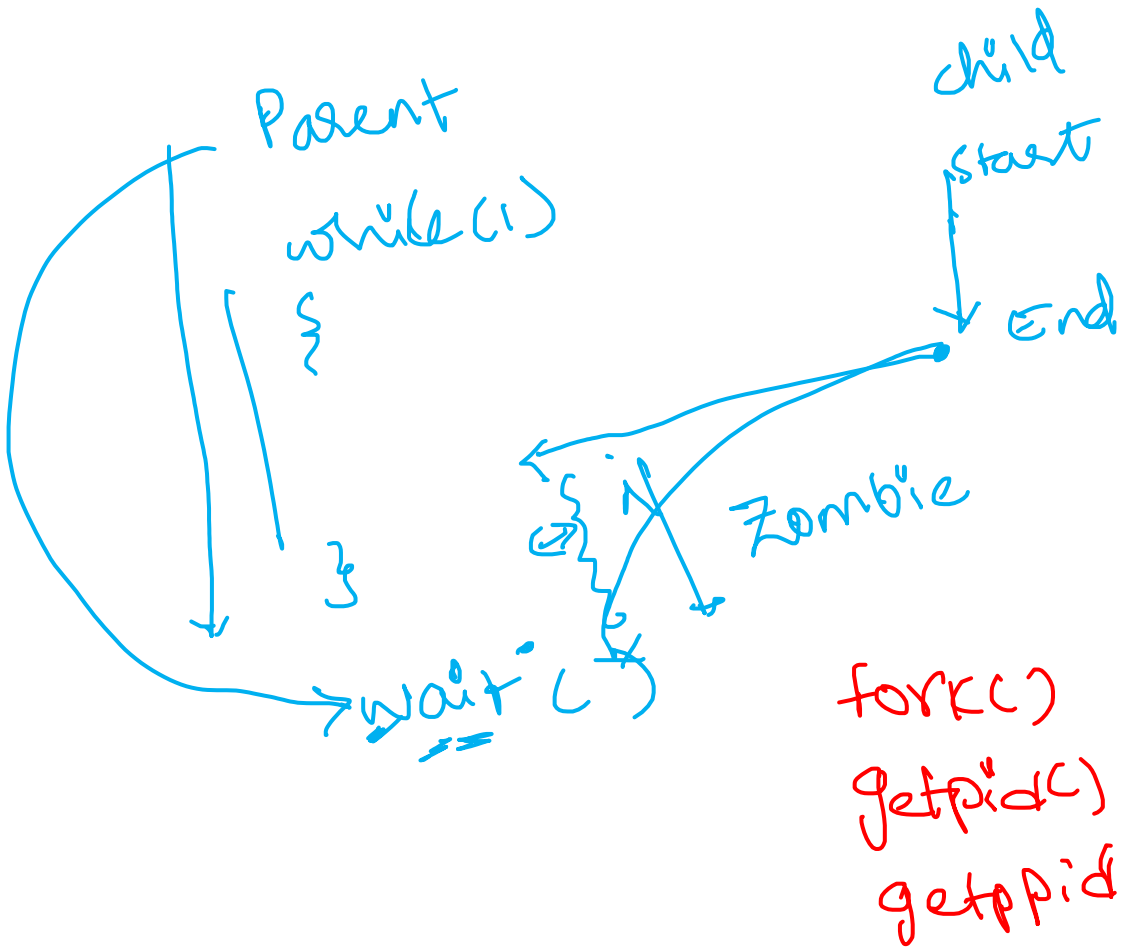
./example1

Pid

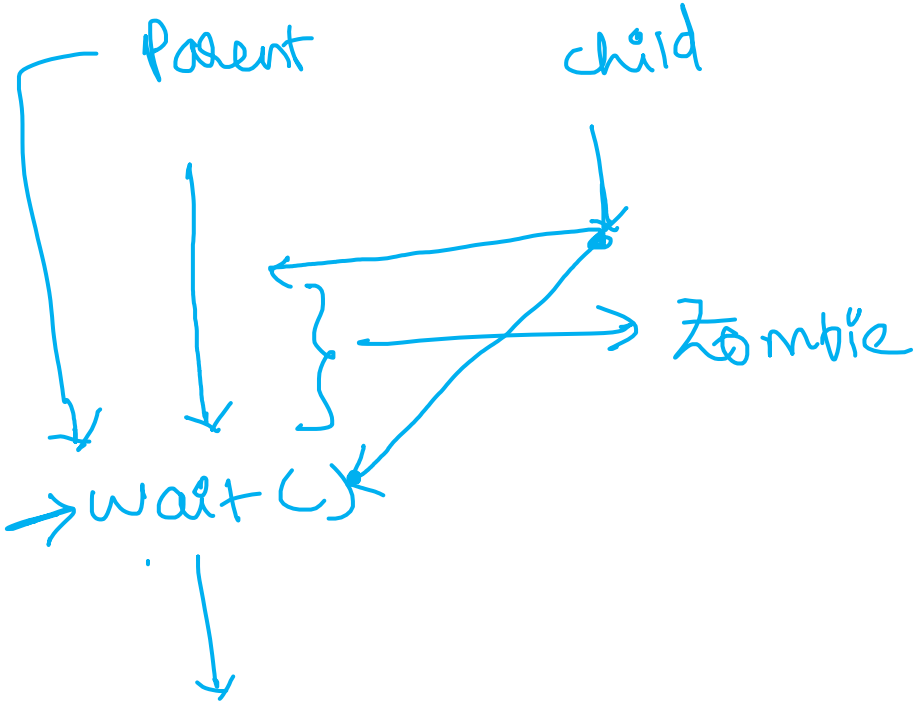
./example1

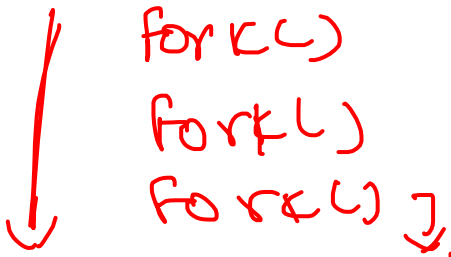
Pid

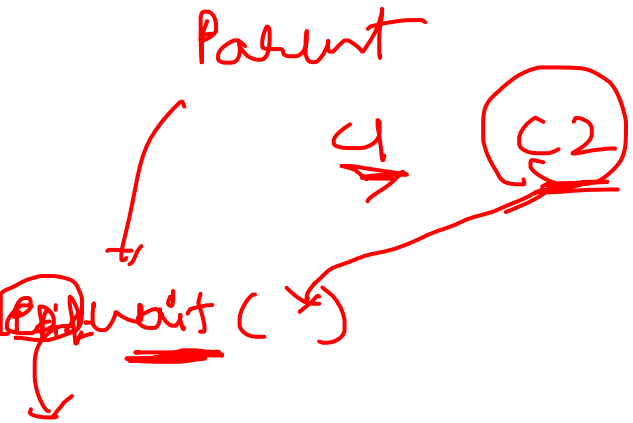

Parent
fork()
child
P
1st
sleep()
1st exit
$
Parent process
wait
fork
child
wait

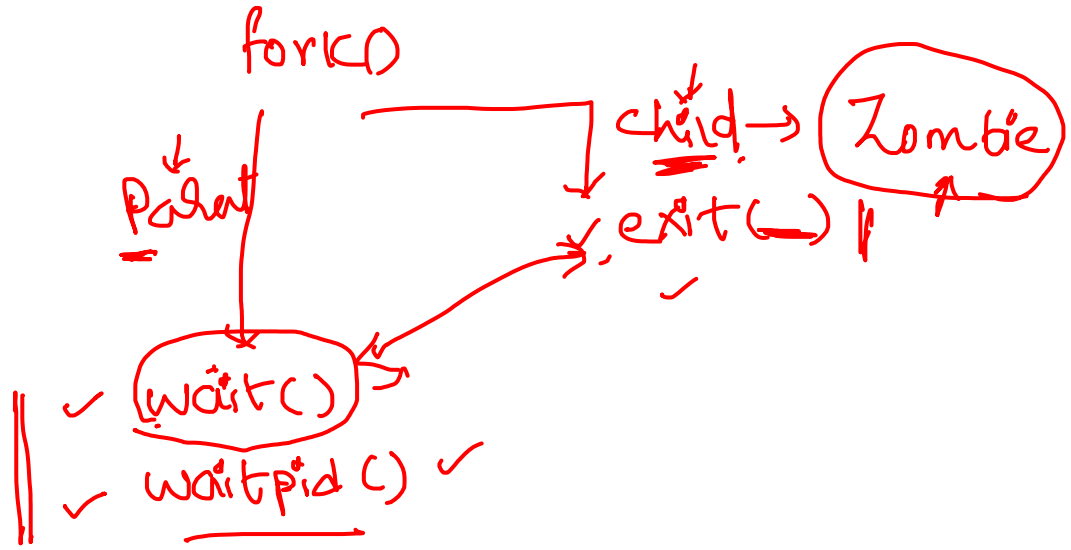


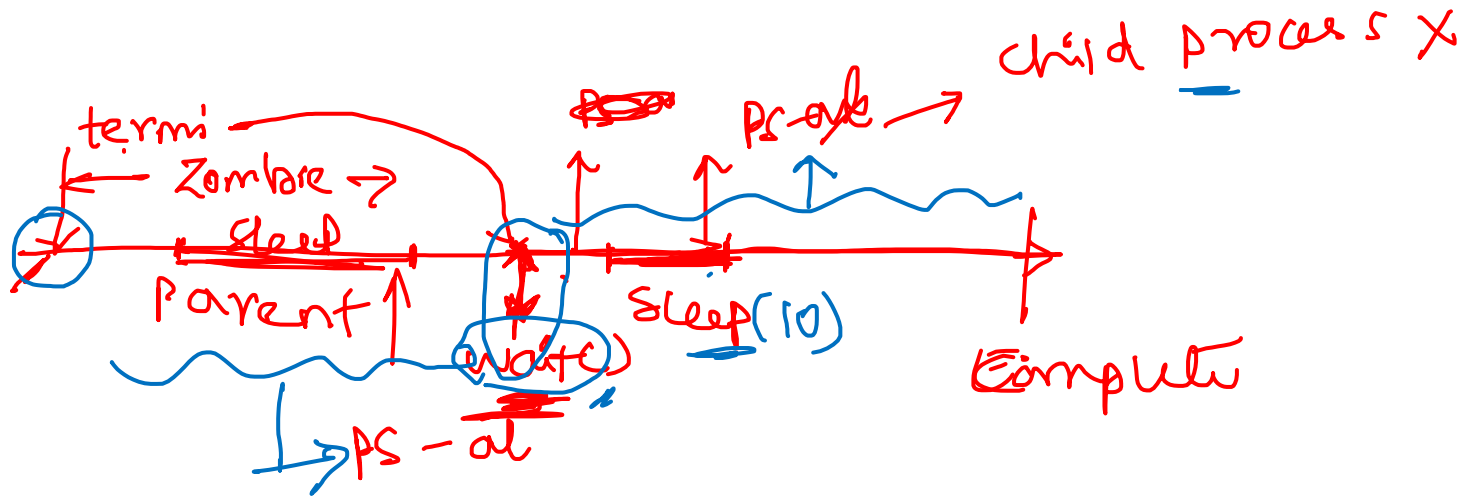
fork()
getpid()
getppid()




fork()
fork()
fork()
printf("Hello world");


Parent
C2
wait()


fork()
Parent
child
Zombie
exit()
wait()
waitpid()


child process X
termi
Zombie
sleep
parent
wait()
ps -al
sleep(10)
Computer

32 bit / 64 bit arch

Databus
32 bits

Databus
64 bit

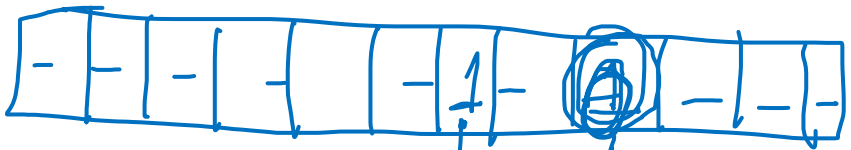
Registers
32 bits

8085 ⇒ 8 bit arch
A B - - -
8 bits

86 ⇒ 16 bit
AX

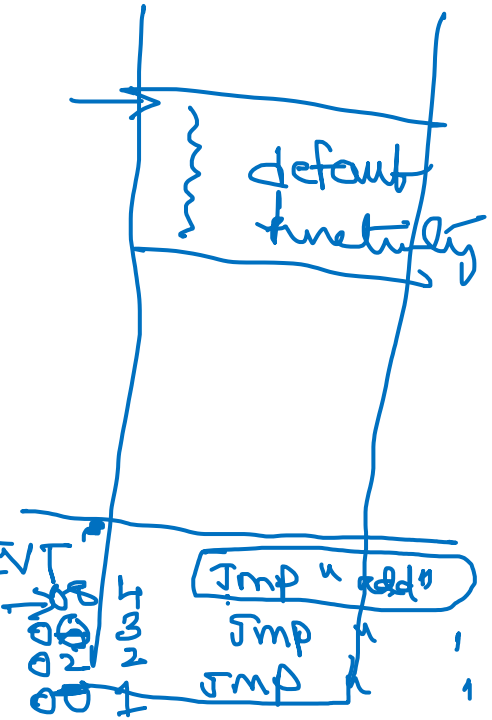
80386
32 bit EAX

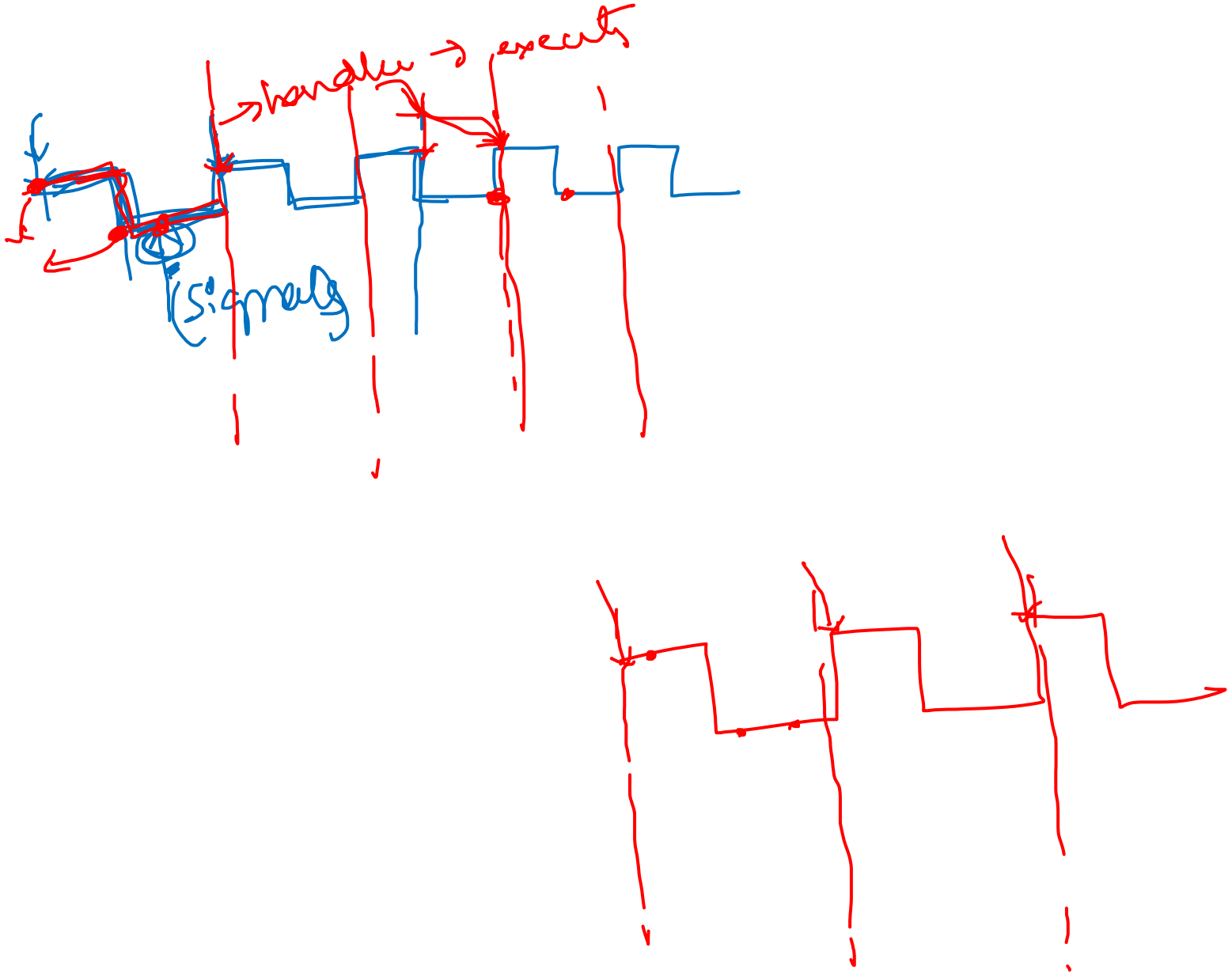
Interrupts
64 Interrupts

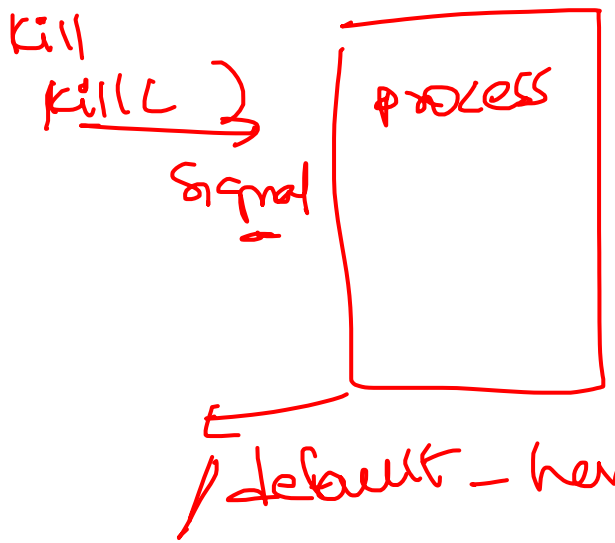


4 x 2 = 8

default functionality → executed




handle
execute
signal

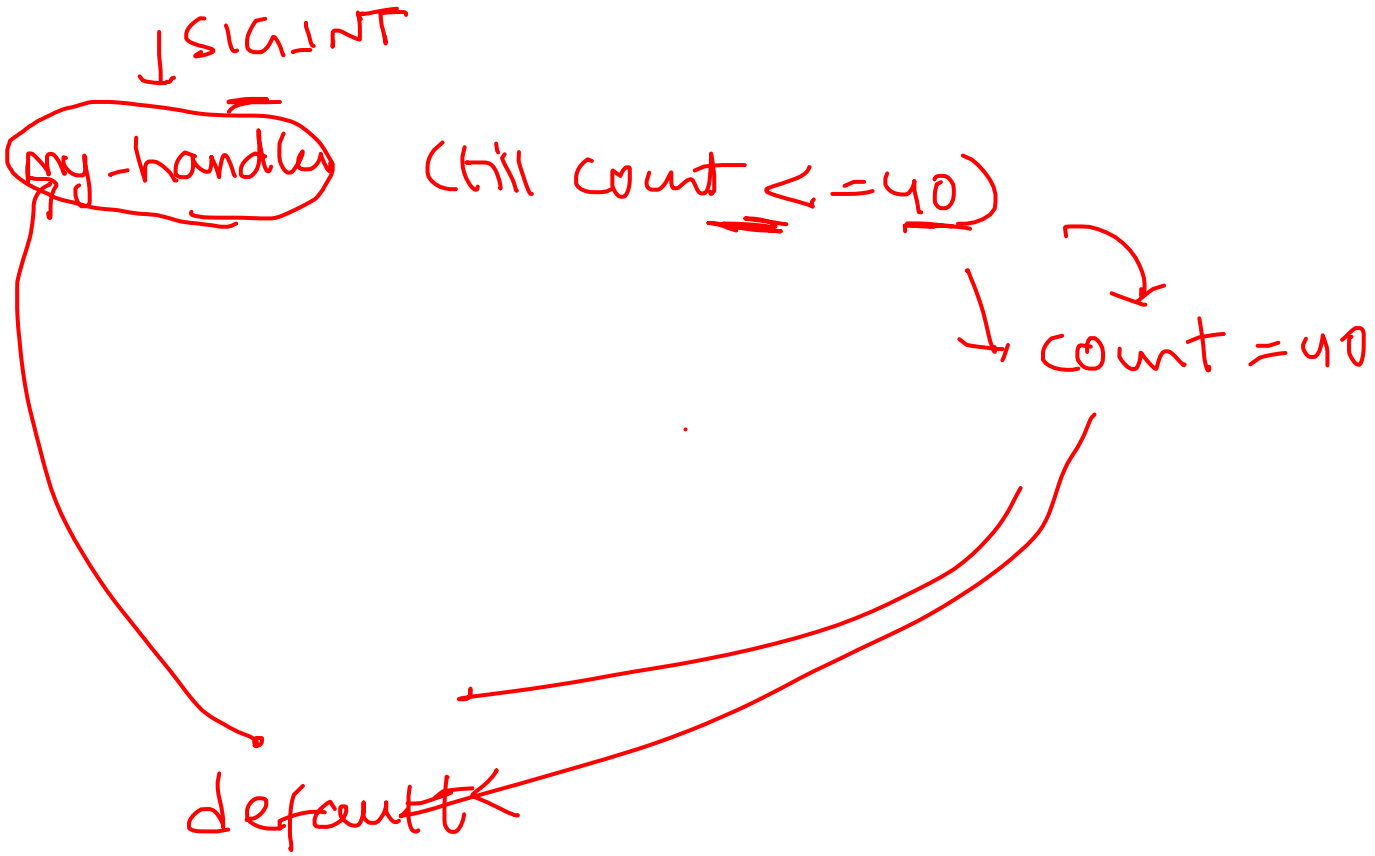


↳ change this default behaviour

signal() ——— step 1: (Register for which you are going to behaviour

- SIG-IGN
- SIG-DFL
- handler

——— step 2: : How to change it


SIGINT
my-handler
(till count <=40)
count =40
default

Exec

→ signal(SIGSTOP, SIG_IGN) ✗

```
while(1)
{
    p ("Hw");
}
```

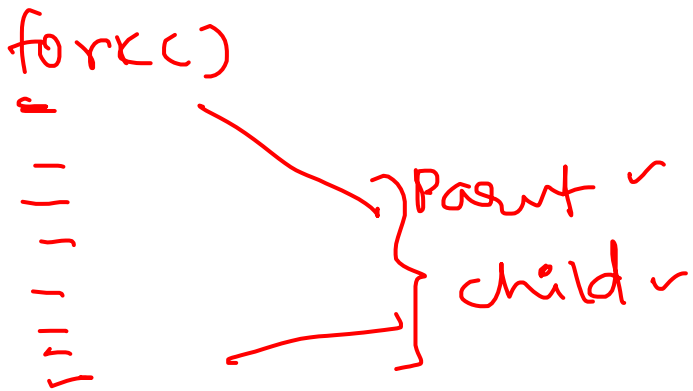
fork(), wait(), exit()

system - - - - kill, kill(), signal()

kill, kill() sends the signal

signal() change the default behaviour of a signal

exec

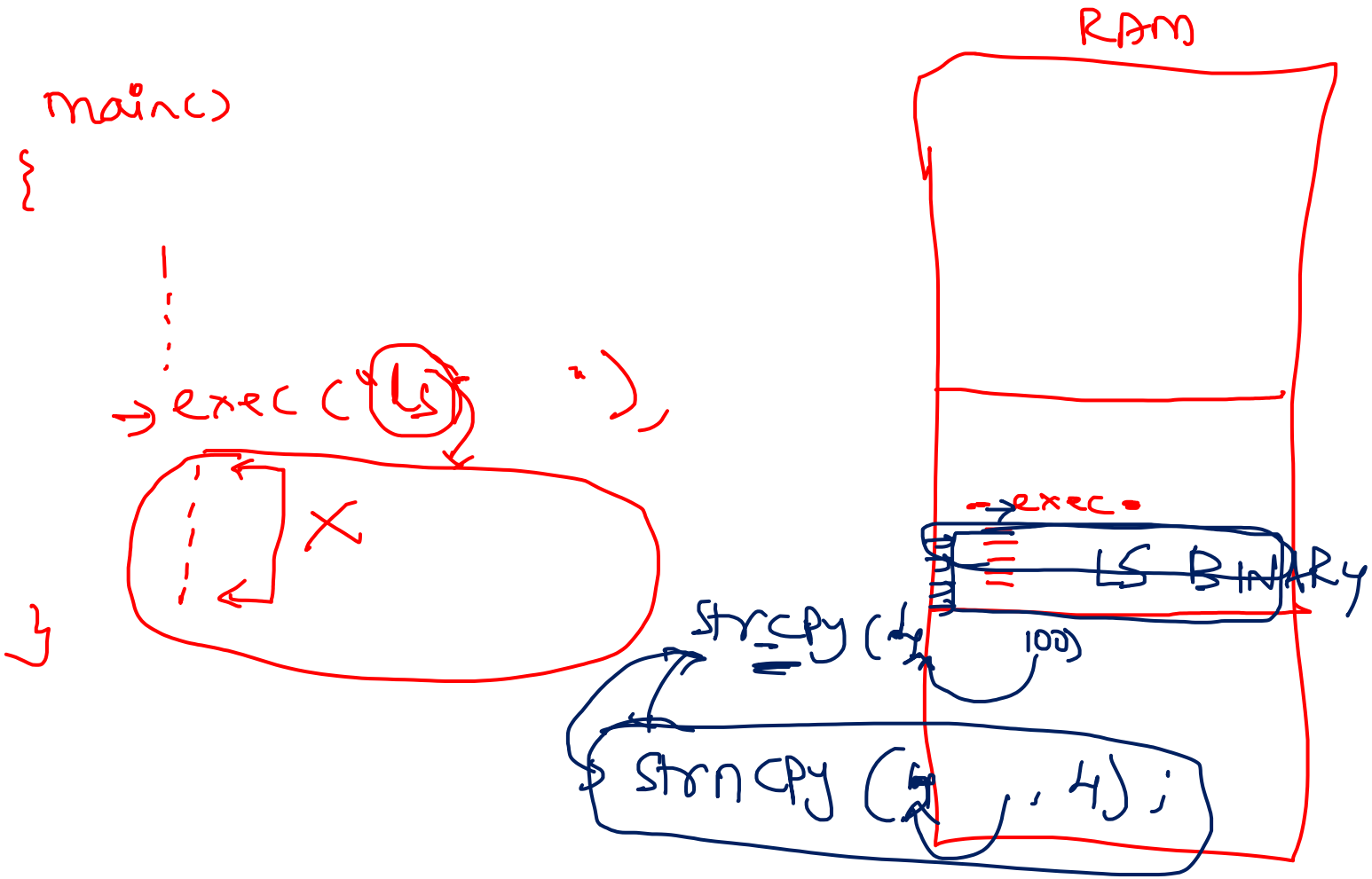



```
if(pid -> )
{

}
else
{

}
```

exec



[ ]

execv

execvp

execve

execl

execlp

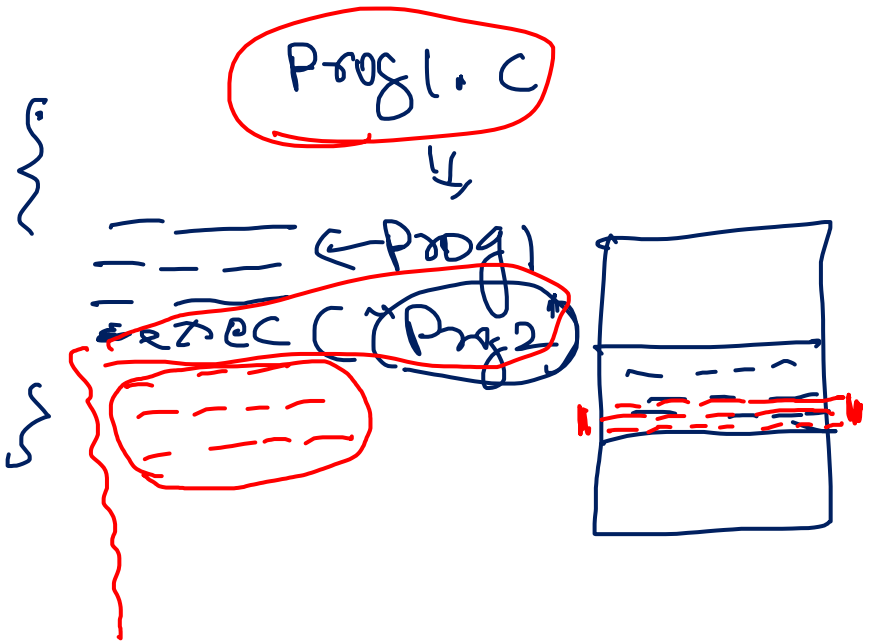
execle

prog2

ps -ax ; ...

exec ( ) ps -ax

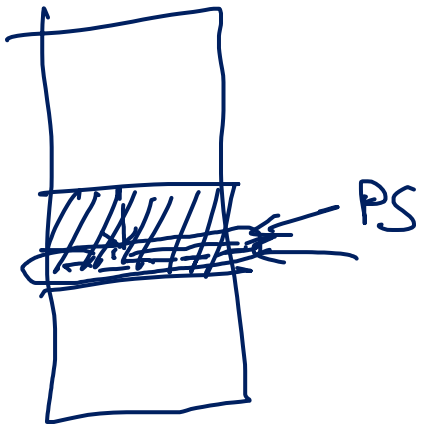
→ execl ("/bin/ps", "ps", "ax", 0);

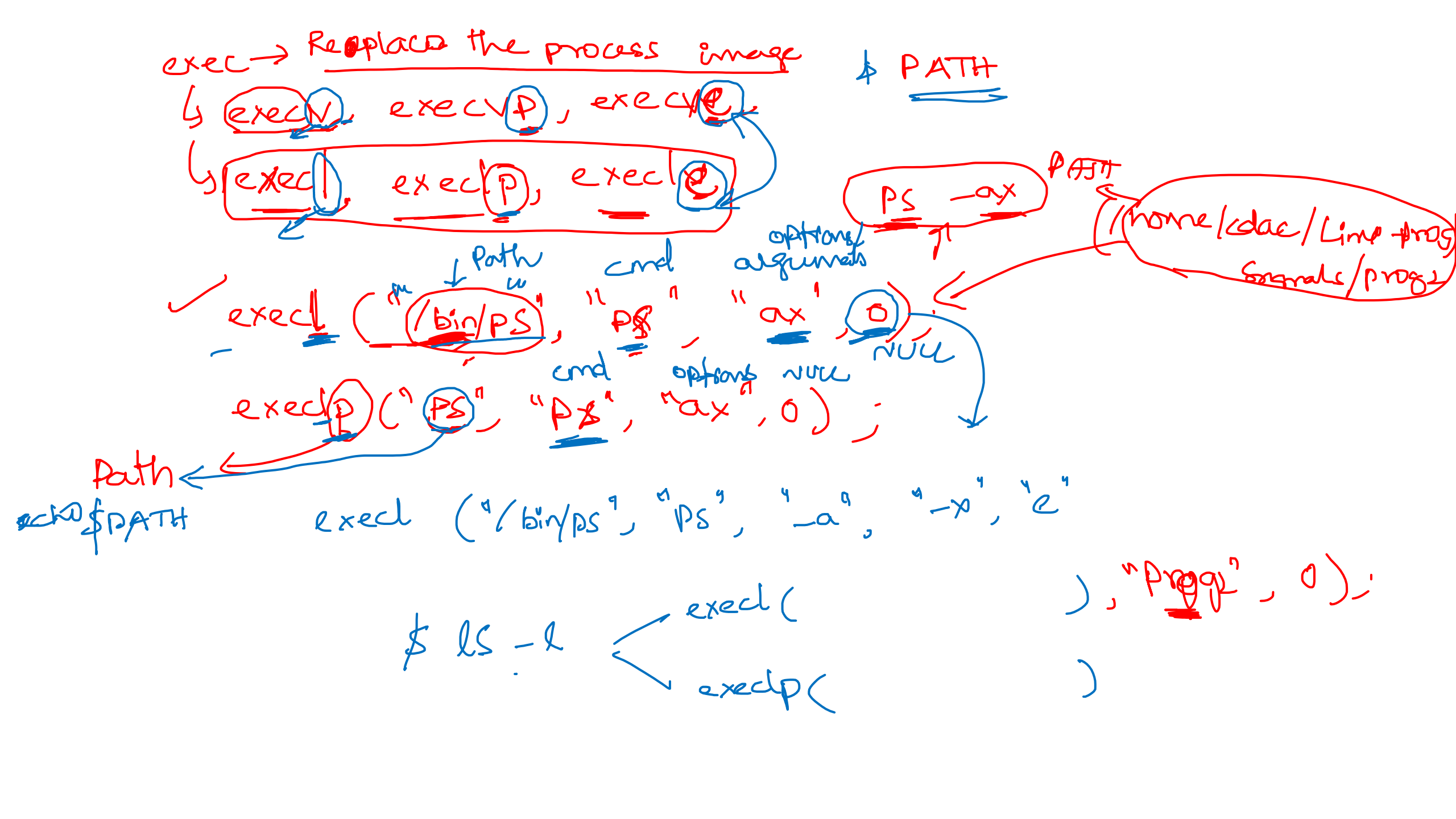


Prog2.c

↓

prog2




exec → Replaces the process image
execv
execvp
execve
execl
execlp
execle
PATH
PATH
ps -ax
/home/cdac/Linux prog
Sourcecode/prog1
options/arguments
Path
cmd
execl ("/bin/PS", "PS", "ax", 0);
NULL
cmd
options
NULL
execlp ("PS", "PS", "ax", 0);
Path
echo $PATH
execl ("/bin/ps", "ps", "-a", "-x", "e"
), "Progp", 0);
$ ls -l
execl(
)
execlp(
)

ls -l ...

execl (<Path of bin>, <binary>, <option arg list ..., 0>)

executable

✗ execl ("/bin/ls", "ls", "-l", 0);

execl ("/bin/ls" ls_argv);

✓ execlp ("ls", "ls", "-l", 0);

execvp ("ls", ls_argv);

echo $PATH

var name

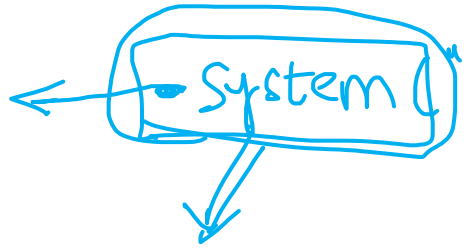
what u want to run

→ char *const ls_argv[] = {"ls", "-l", 0};

char *const ls_env[] = {"PATH=/bin:/usr/bin", "TERM=console", 0};

execle ("/bin/ls", "ls", "-l", 0, ls_env);

execve ("/bin/ls", ls_argv, ls_env



"ps -ax"); → spawning a separate process → pid

fork() ✓

← Parent

child() + pid

{ execl("/bin/ps", "ps", "ax", 0);

COW

Child

int a = 10

fork()

→ Parent
→ a = 20

child → sleep();
value
second of a ??

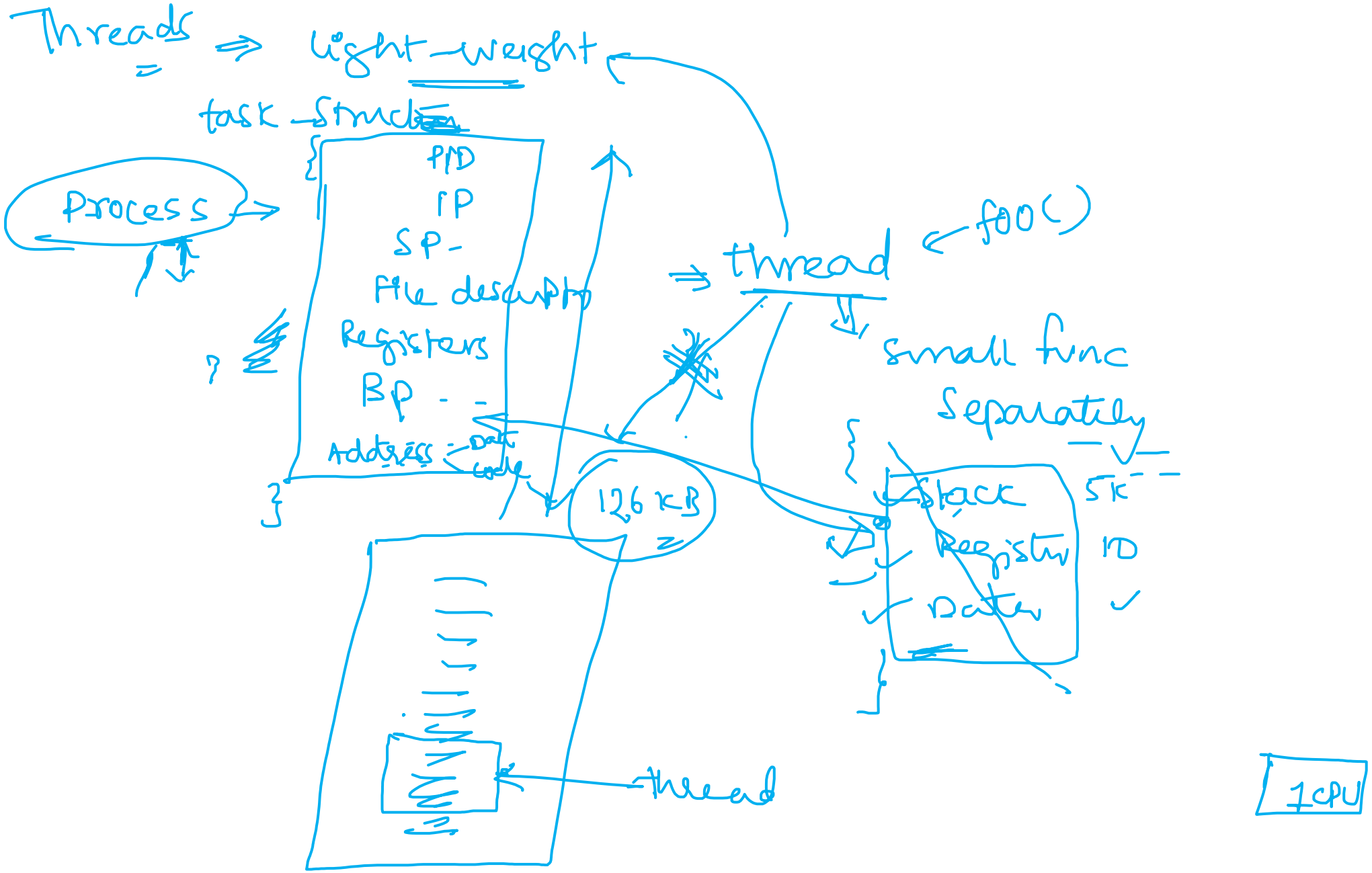
fork()

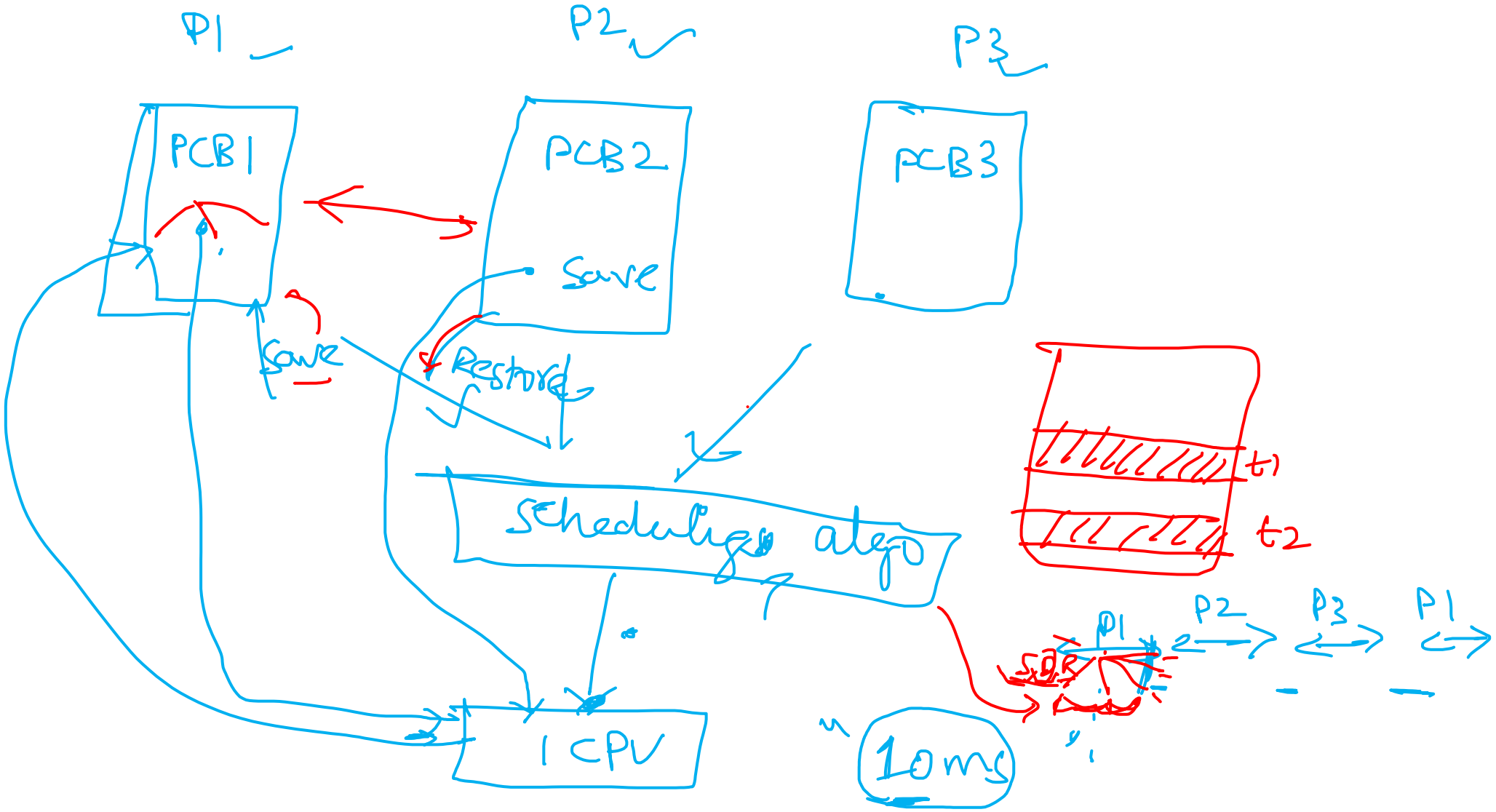
exec() — 6 variants

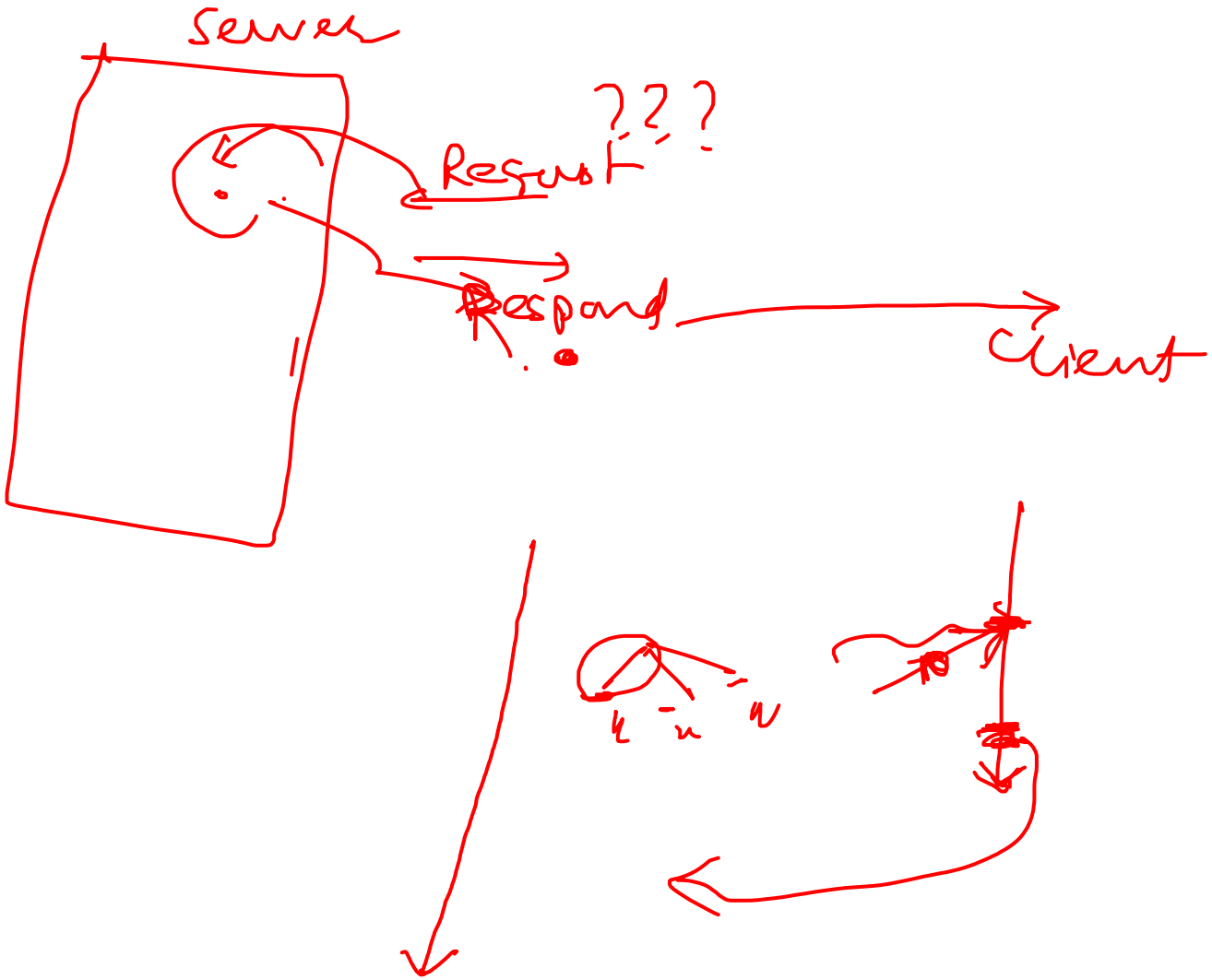
system()

Signals < Register handle

wait, exit


Threads ⇒ light-weight
task_struct
PID
IP
SP
File descriptor
Registers
BP
Address
Data
Code
Process
thread
foo()
small func
Separately
Stack
5K
Registers
ID
Data
126 KB
thread
1CPU


P1
P2
P3
PCB1
PCB2
PCB3
Save
Save
Restore
Scheduler algo
1 CPU
10ms
t1
t2
P1
P2
P3
P1
SQR


Server
???
Request
Respond
Client