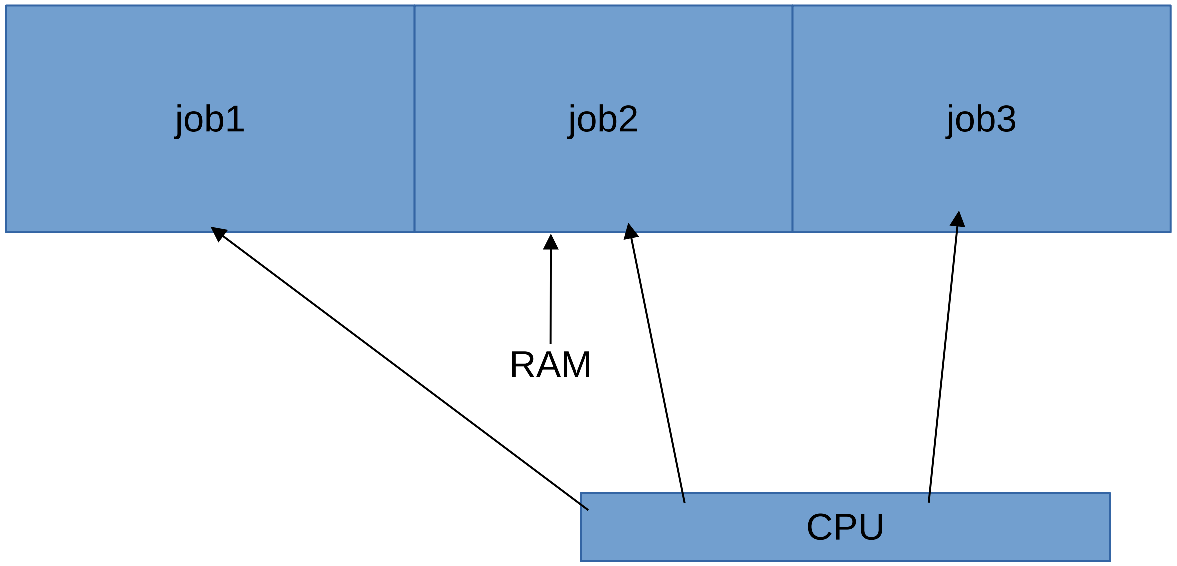
Still problem

CPU still not utilized fully....

E.g:

When a job stopped the operator would have to notice that by observing the console , Determine why it stopped(normal / abnormal) and then take a necessary actions.

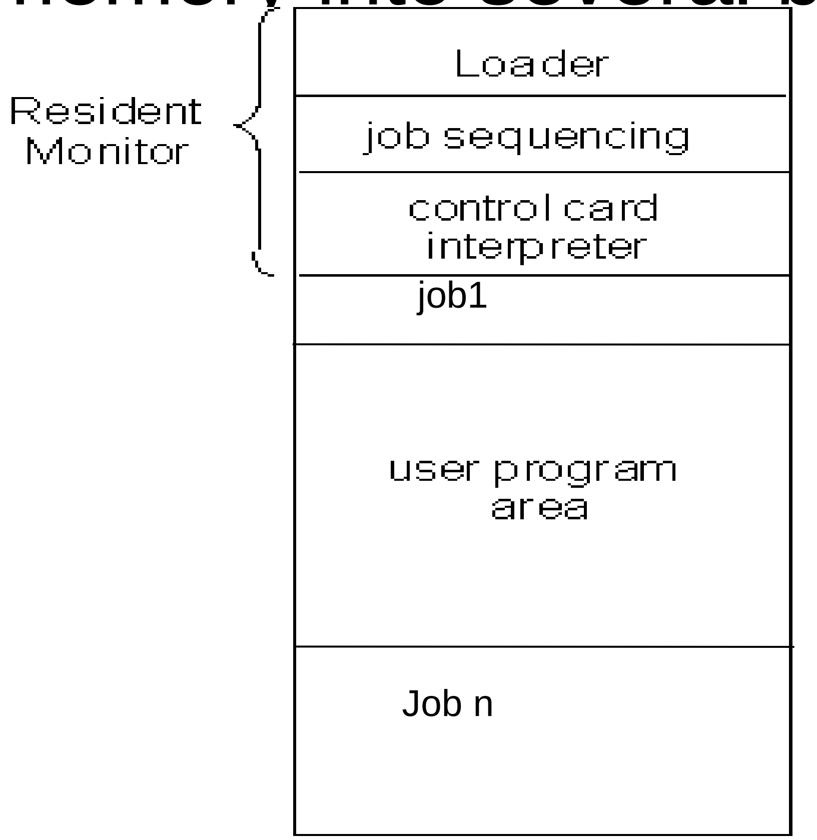
CPU Switching from one job to another takes time


job1
job2
job3
RAM
CPU

Solution

Resident monitor...... ?

The first operating systems were designed to improve the utilization. The solution was simple: break up the memory into several blocks, as shown



One program was always loaded into memory, and continuously running, the resident monitor.

If a user program was loaded into memory, the resident monitor would transfer control to the program.

When the program halted, it would return control to the resident monitor, which could then transfer control to the next program.

Even at this stage, interaction with the computer was not via terminals.

The user programs were read from punched cards, and utilities like the FORTRAN compiler from tape.

To improve efficiency, jobs were run in batches - programs that required the same utility (compiler, e.g.) would be run together, to avoid multiple loading/unloading of the compiler.

In modern computers, I/O is much faster.

Usually, the computer reads the executable program from peripheral memory devices such as floppies, or hard disks.

Output, if it is to be printed, is transmitted to printers and displayed on the screen. Still, compared to the CPU speed, I/O is many magnitudes slower.

Another bottleneck arises due to interactive programs.

If the execution requires input from the user, the CPU has to wait a long time while a user enters the input via keyboard (for instance).

Solution?????

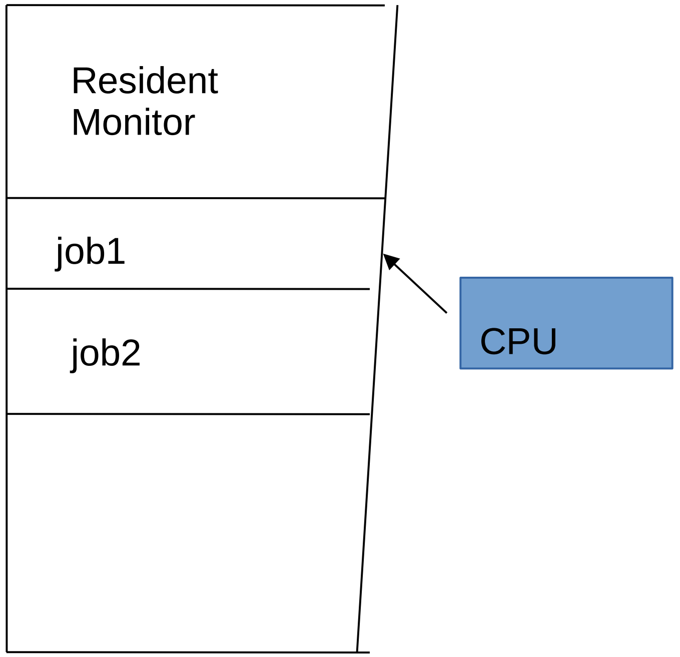
most operating systems schedule many different processes simultaneously.

This is called multiprogramming.

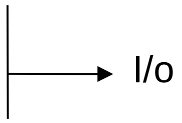
Here, many programs are loaded into different parts of the memory. The OS starts executing the first one.

If/when the execution requires I/O or some such delay, the CPU immediately switches to the next job and starts it; and so on.

The concept is similar to how human managers operates.

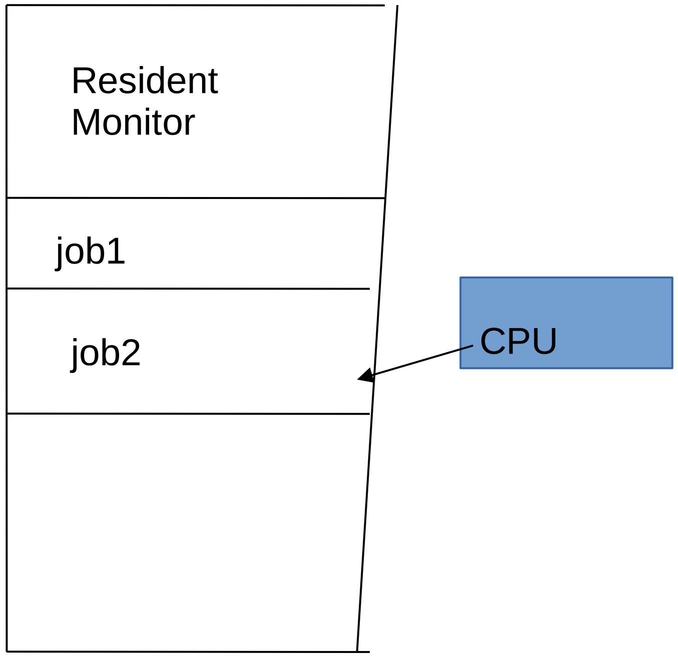


```
#include<stdio.h>
Int main()
{
        Int I;
        printf("\n enter the number");
        Scanf("%d",&i);
        printf("\n i=%d",i);
        Return 0;
}
```

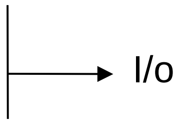


job1

When job1 is doing I/o cpu sit idle



```
#include<stdio.h>
Int main()
{
        Int I;
        printf("\n enter the number");
        Scanf("%d",&i);
        printf("\n i=%d",i);
        Return 0;
}
```



job1

When job1 is doing I/o cpu sit idle

```
#include<stdio.h>
Int main()
{
        Int p;
        P=90+20;
        Return 0;
}
```

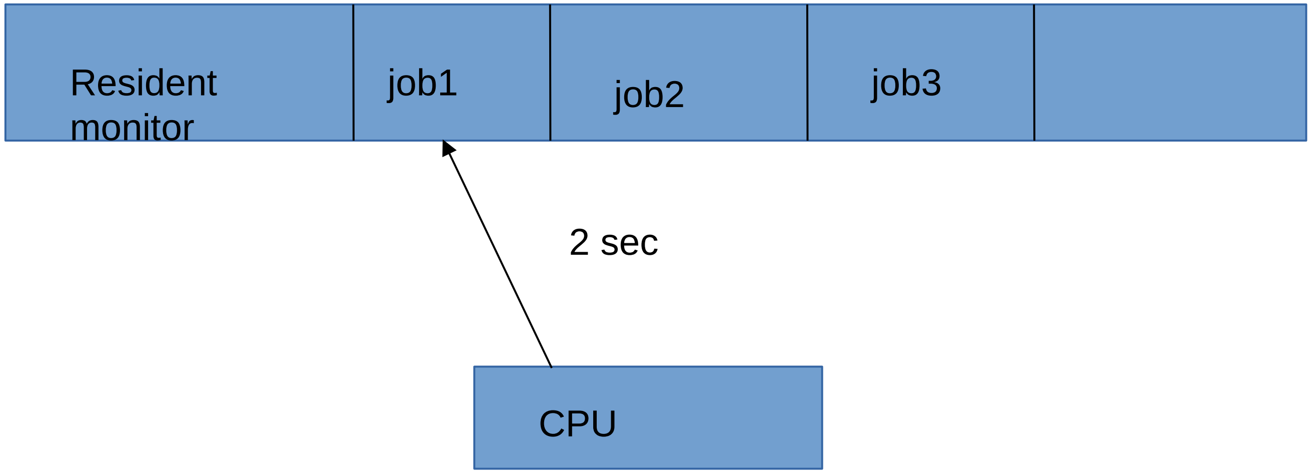
job2

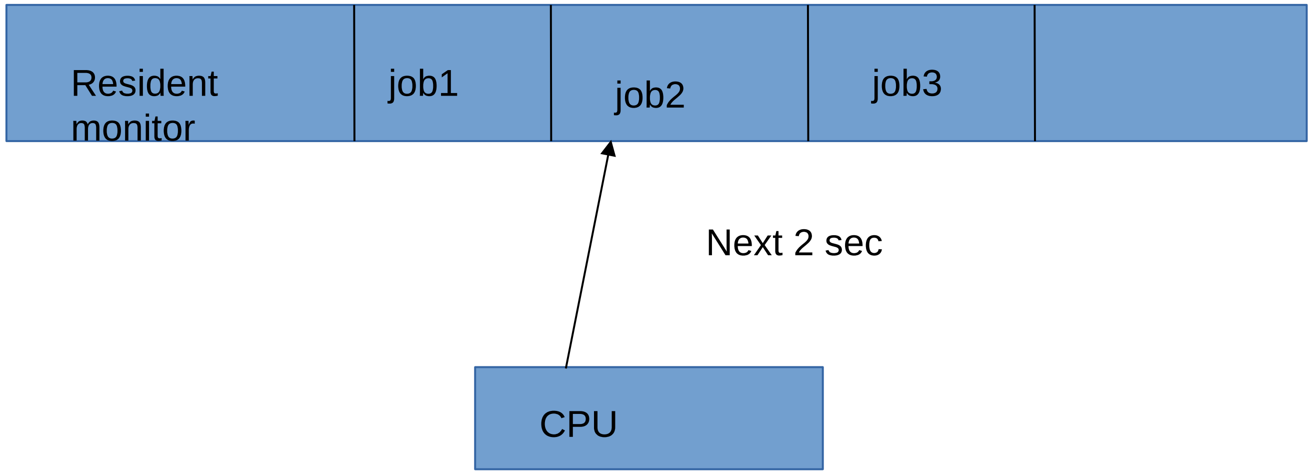
THE OTHER WAY IS TIME SHARING(or Multi tasking)

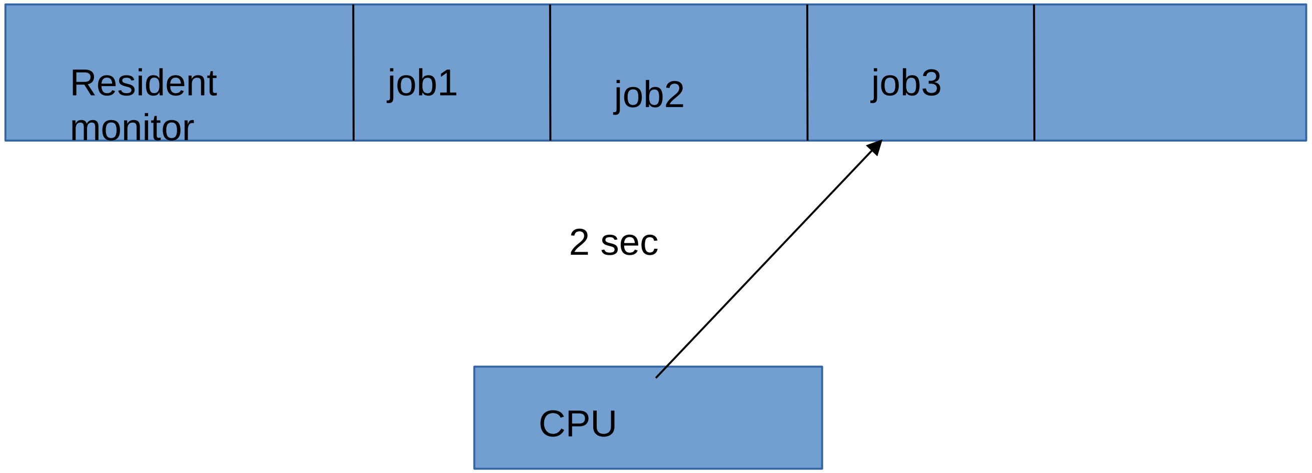
. Is an extension of multiprogramming.

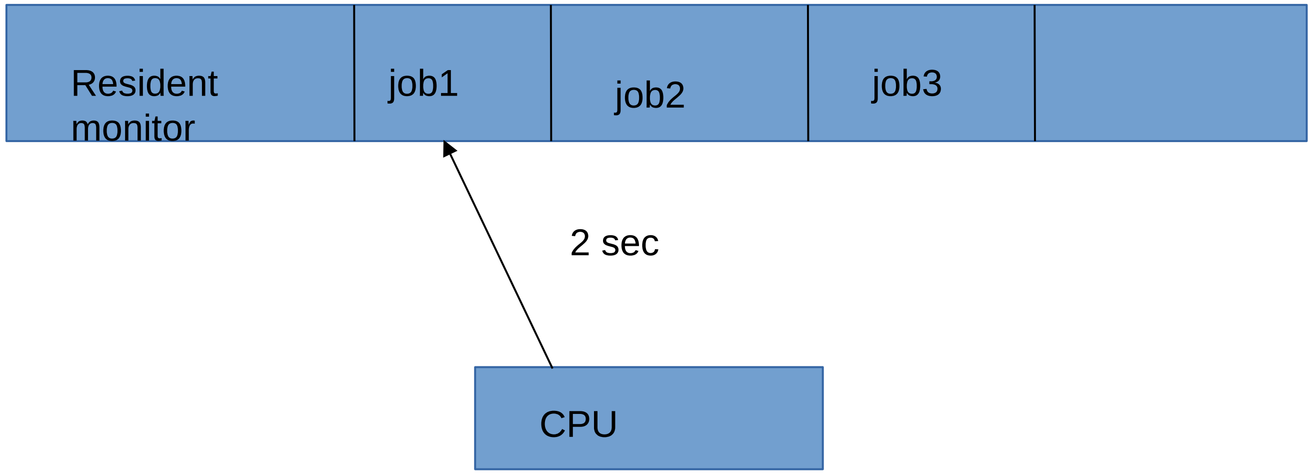
. Like in multiprogramming multiple jobs are executed by the cpu switching between them but the switches occur so frequently that the users may interact with each program while it is running.

. It gives an impression that s/he has his or her own computer. Whereas actually one computer is being shared among many users.


Resident monitor
job1
job2
job3
2 sec
CPU


Resident monitor
job1
job2
job3
Next 2 sec
CPU


Resident monitor
job1
job2
job3
2 sec
CPU


Resident monitor
job1
job2
job3
2 sec
CPU

Multitasking and Multiprogramming

An interactive computer system provides online communication between the user and the system.

The user gives instruction to the operating system or to the program directly and receives an immediate response.

Time sharing os is more complex than the Multiprogramming.

1. several jobs must be kept simultaneously in memory, which requires some form of memory management and protection.

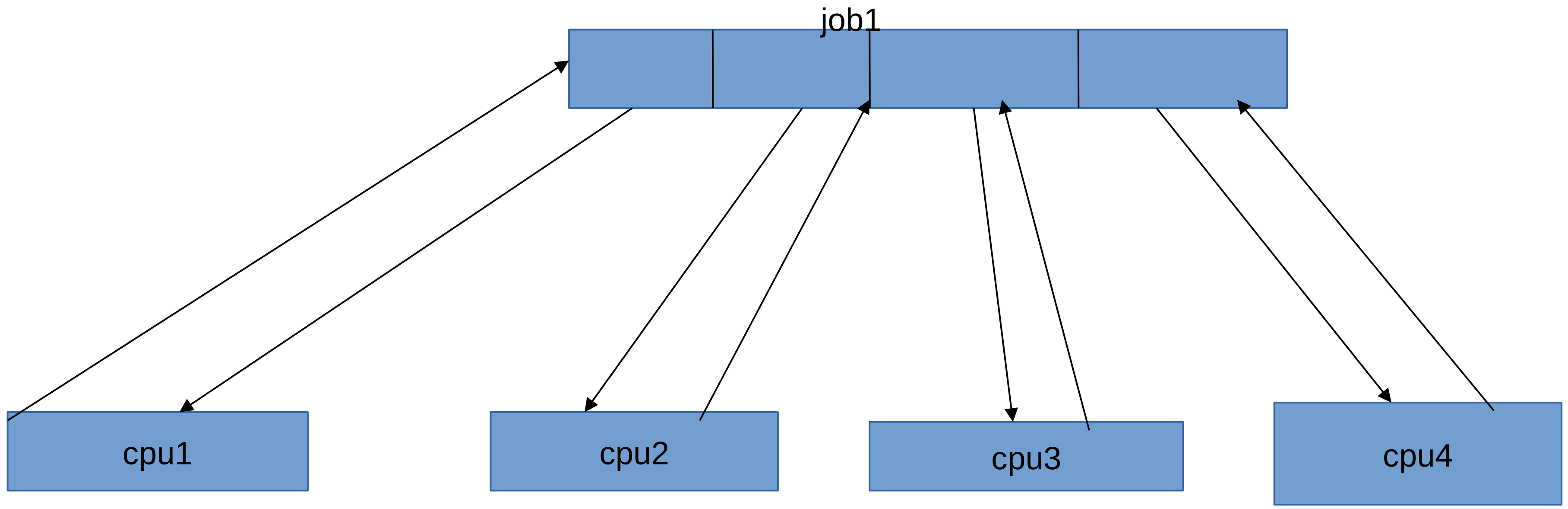
2.Time sharing system must also provide an online file system. The file system resides on a collection of disk hence , disk management is required.

3. They also provide a mechanism for concurrent execution, which required sophisticated CPU scheduling policies.

4. They also must provide mechanism for job synchronization and communication and must ensure that dead lock does not occure.

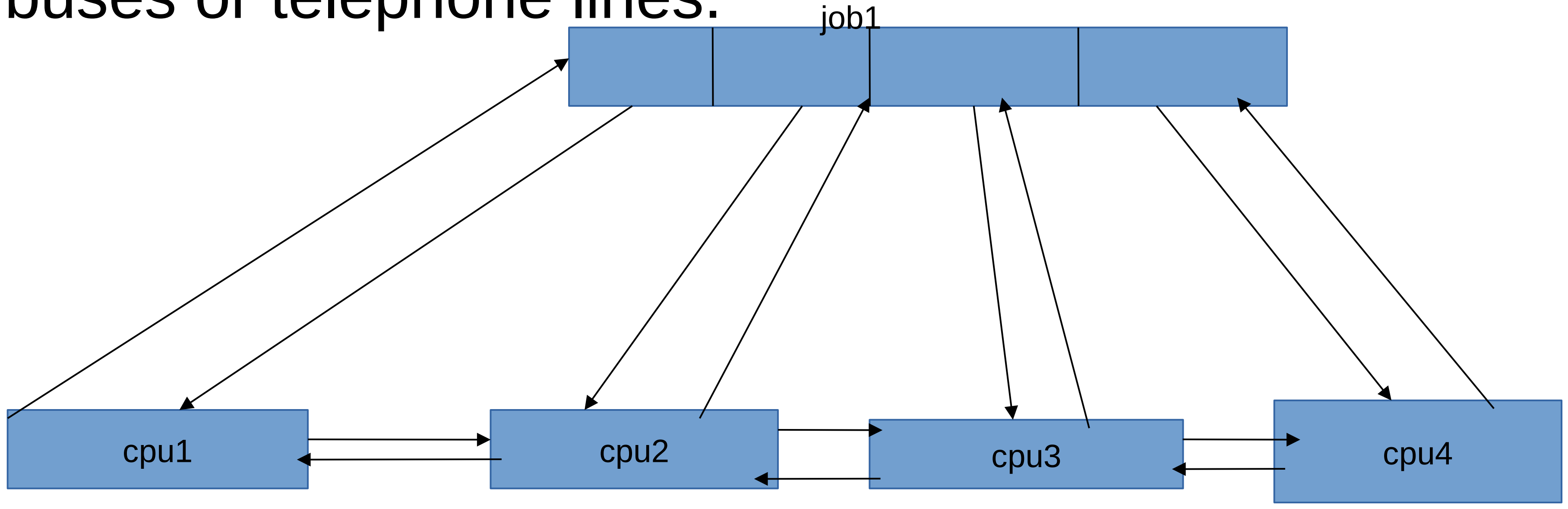
Distributed OS

Distribute the jobs among many processors.



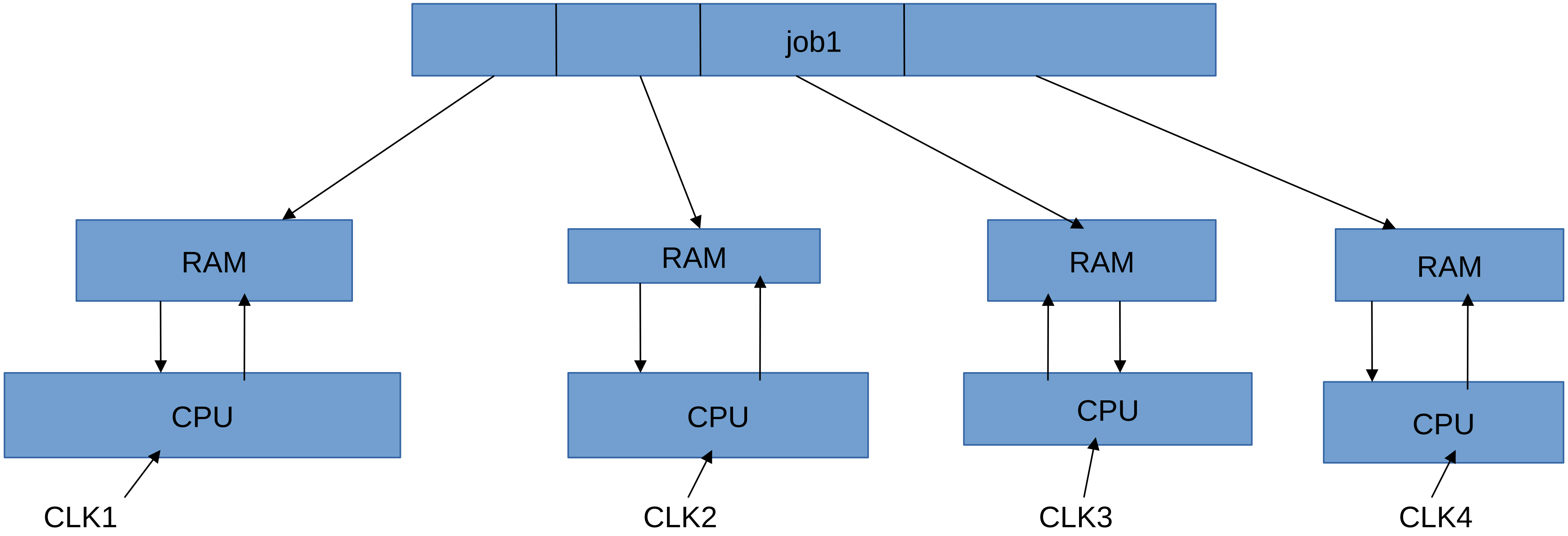
Distributed OS

The processors communicate with each other through various communication lines, such as high speed buses or telephone lines.



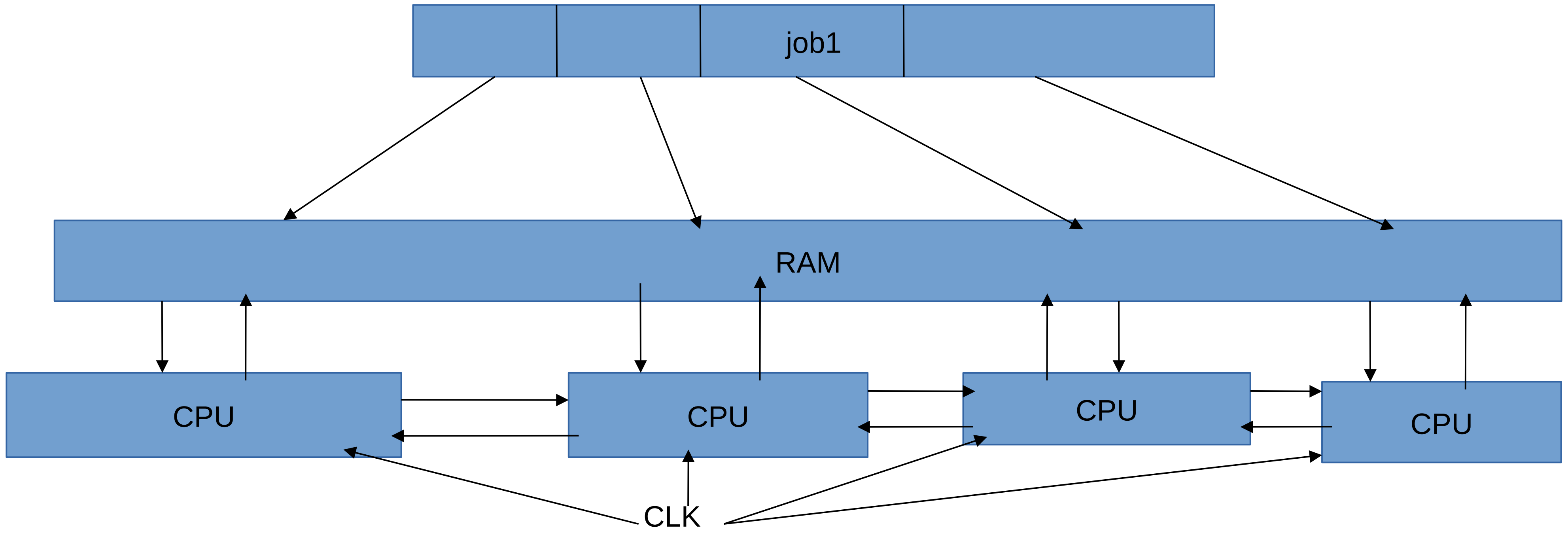
Distributed OS
Loosely coupled system

It does not share memory or a clock.



Distributed OS
Tightly coupled system

One in which there are multiple processors which communicate with each other by sharing computer bus,the clock,memory etc.


job1
RAM
CPU
CPU
CPU
CPU
CLK