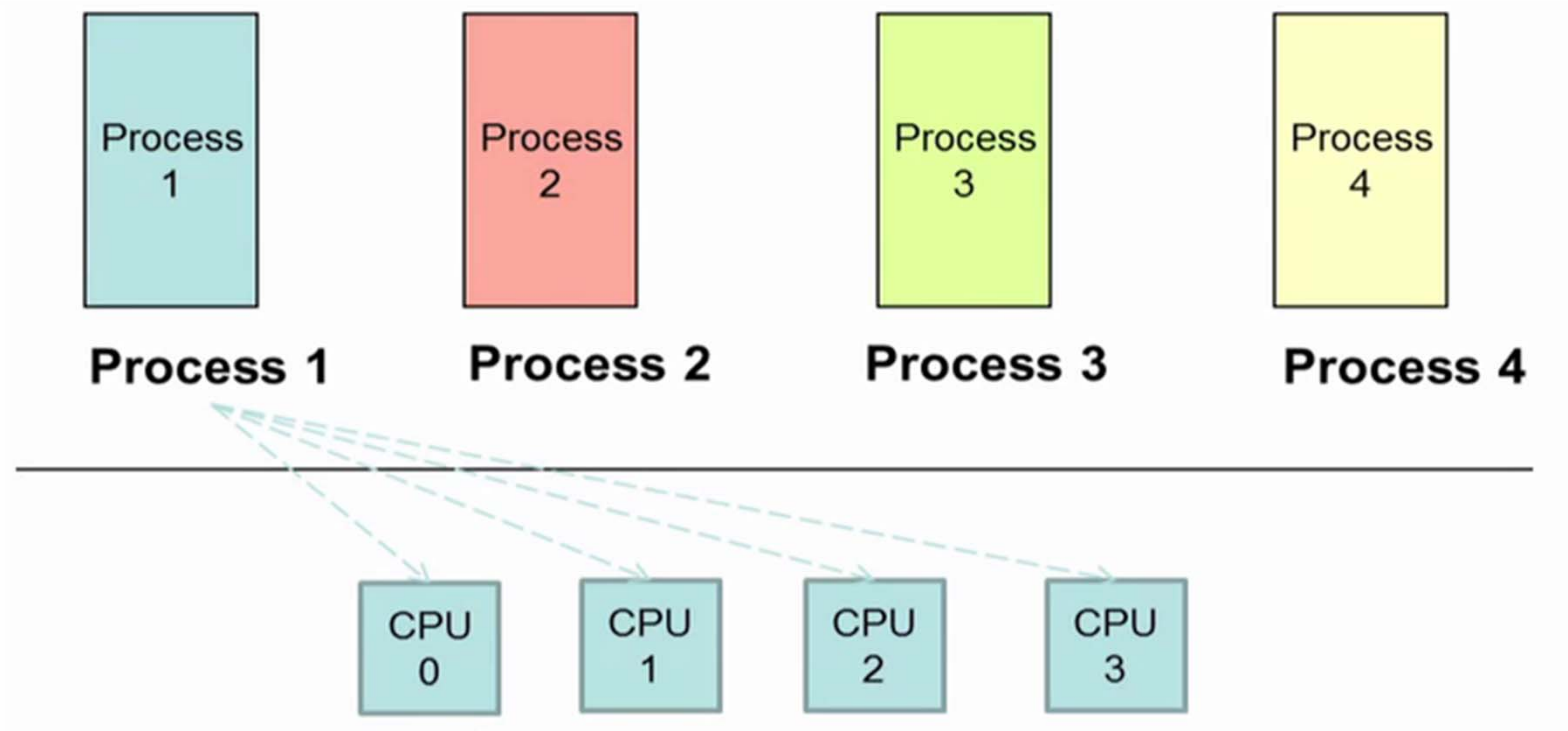
Multi-Processor Scheduling

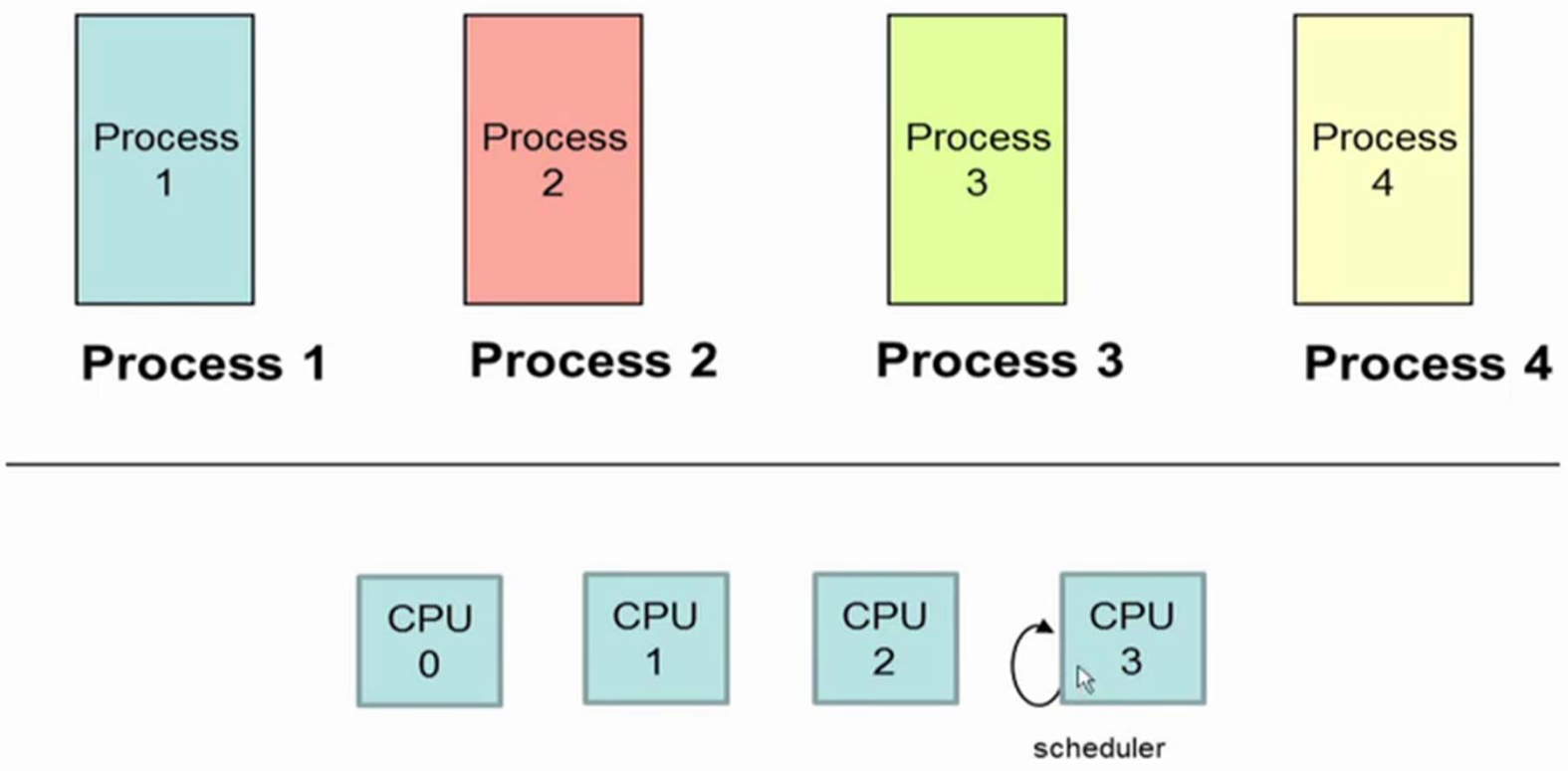
Introduction

- **Multiple CPUs** are available.
- More **complex** as compared to single processor scheduling
- Processors are identical in terms of their functionality i.e. **HOMOGENEOUS**.



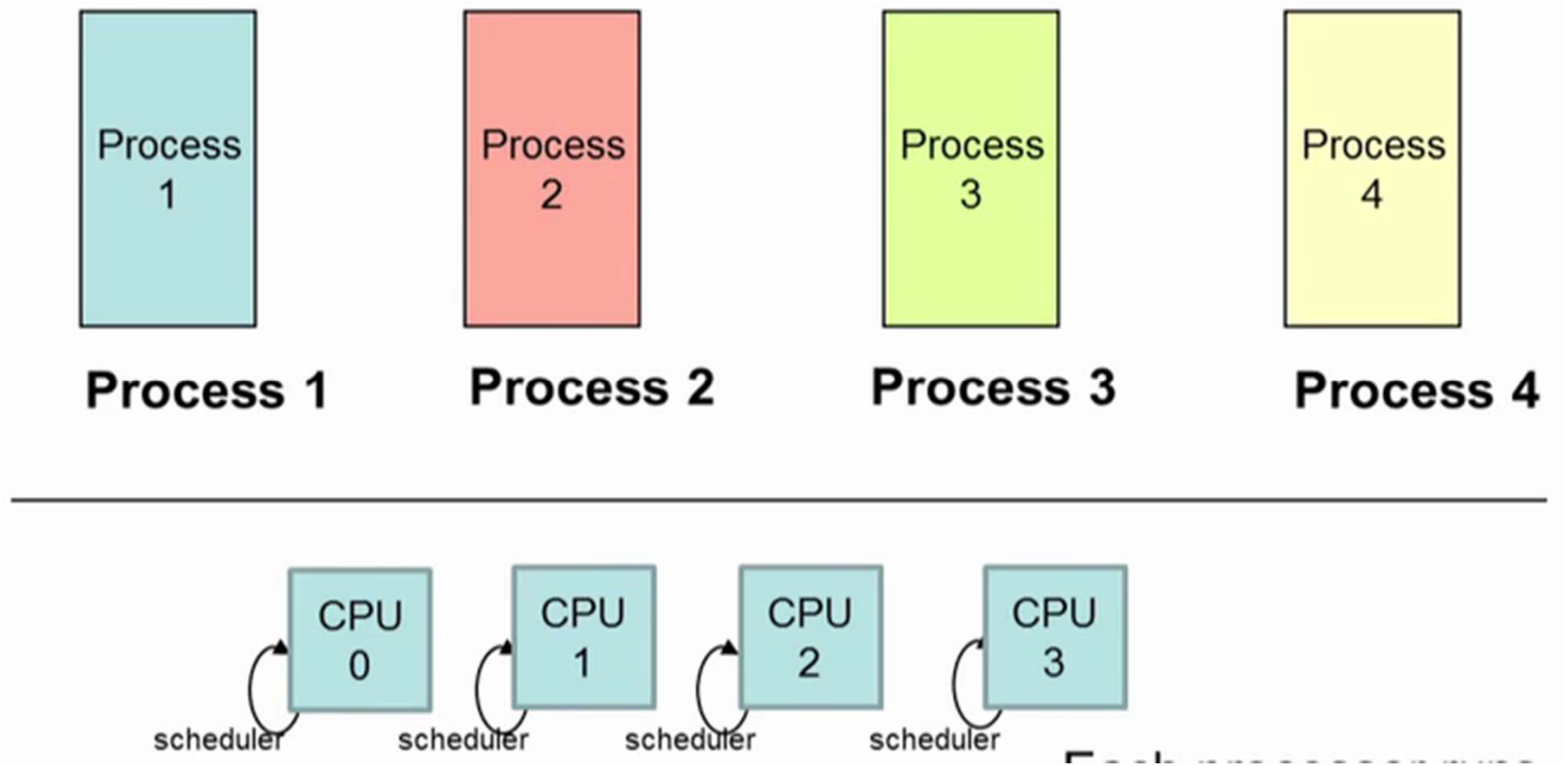
Single Scheduler (Asymmetric)

- All the scheduling decisions handled by a single processor.
- Master-slave architecture.
- Pros : Simple, reduces the need of data sharing.
- Cons: Performance degradation due to waiting for master.



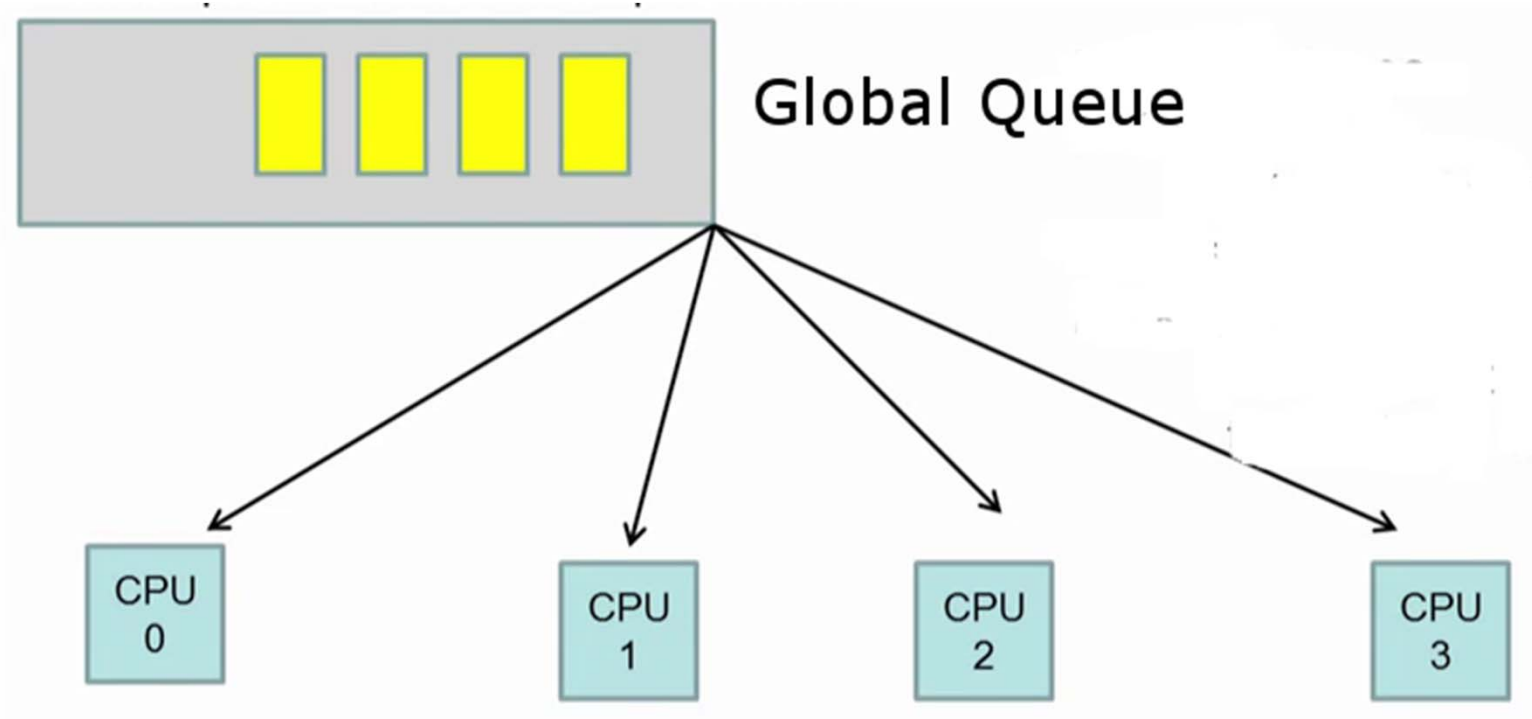
Multiple Schedulers (Symmetric)

- Each processor runs a scheduler independently for selecting which process to execute.
- Two variants: Global queue and Per CPU queues



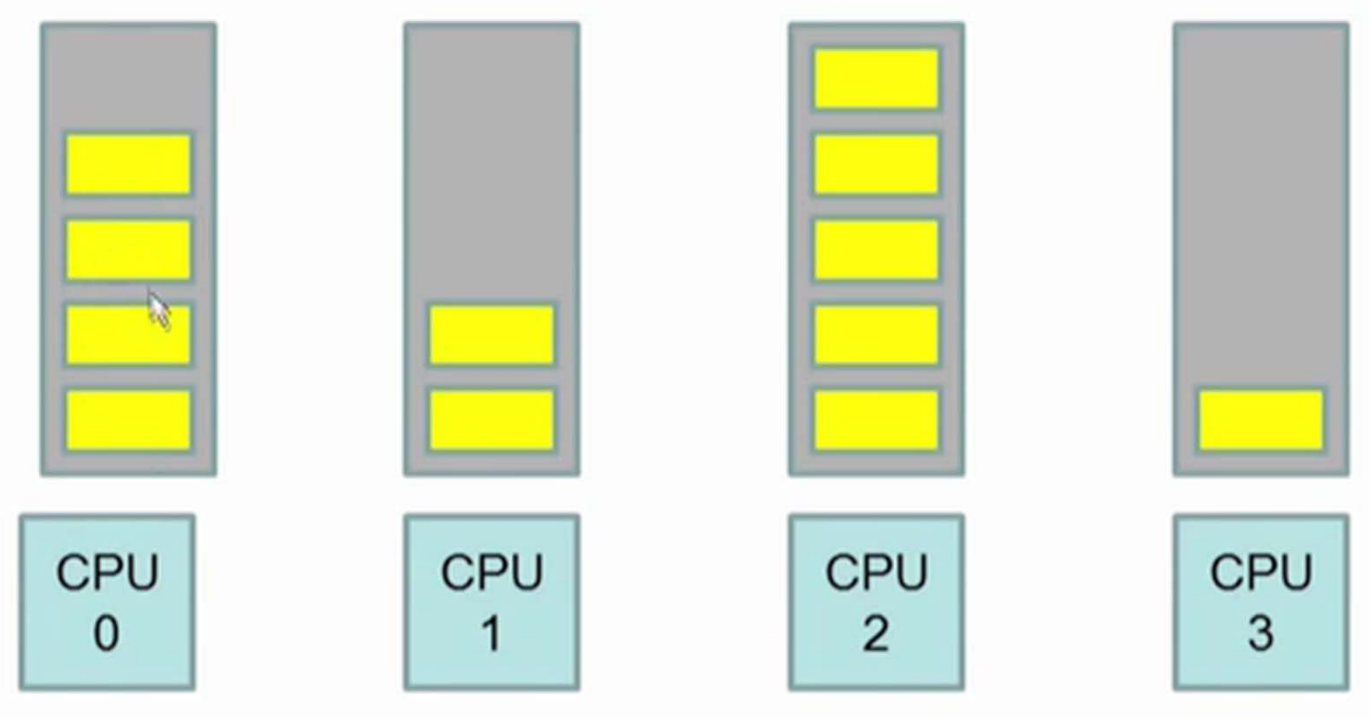
Global Queue

- One queue of ready processes is shared among all the processors.
- Individuals schedulers look up in the global queue for deciding which process to run next.
- Pros: CPU utilization, fairness.
- Cons: Not scalable, locking required.



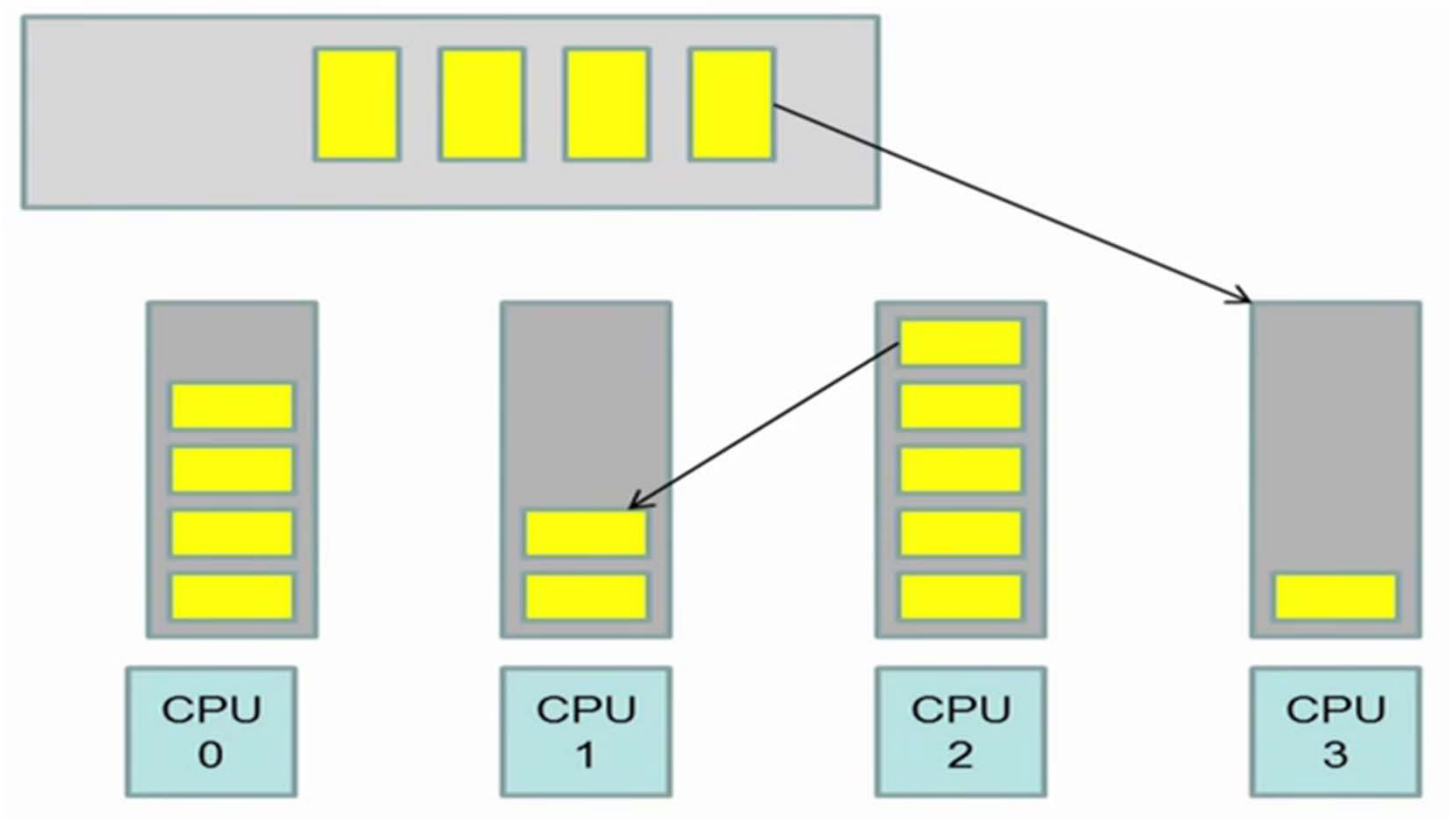
Per CPU Queues

- Each CPU has its own queue of ready processes.
- Static partitioning of processes for CPUs (by user or OS).
- Pros: Easy to implement, no locking mechanism.
- Cons: Load imbalance.



Hybrid Approach

- Use of local and global queues.
- Load balancing across queues feasible.
- Similar approach followed in Linux 2.6



Load Balancing

- Two Techniques : Push Migration and Pull Migration
- Push Migration : A special task periodically monitors load of all processors, and redistributes work when it finds an imbalance.
- Pull Migration: Idle processors pull a waiting task from a busy processor.

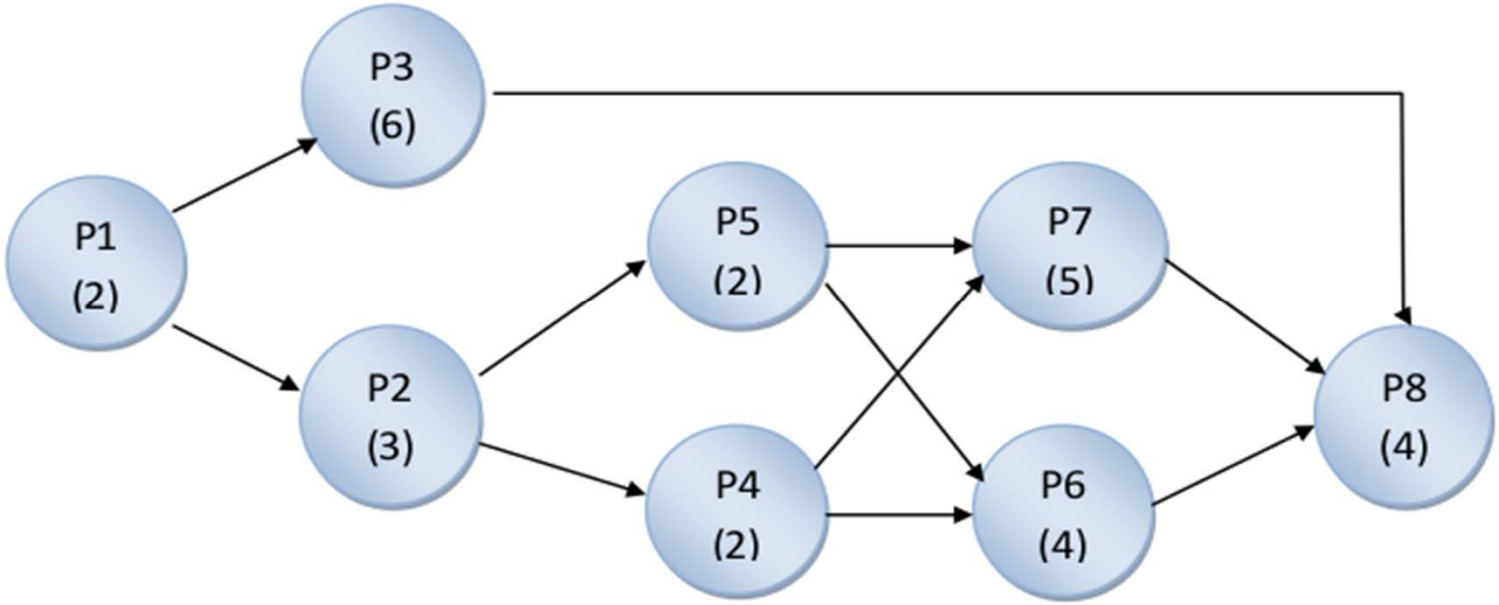
Note: Processor migration is expensive and should be avoided. Use of **Processor Affinity.**

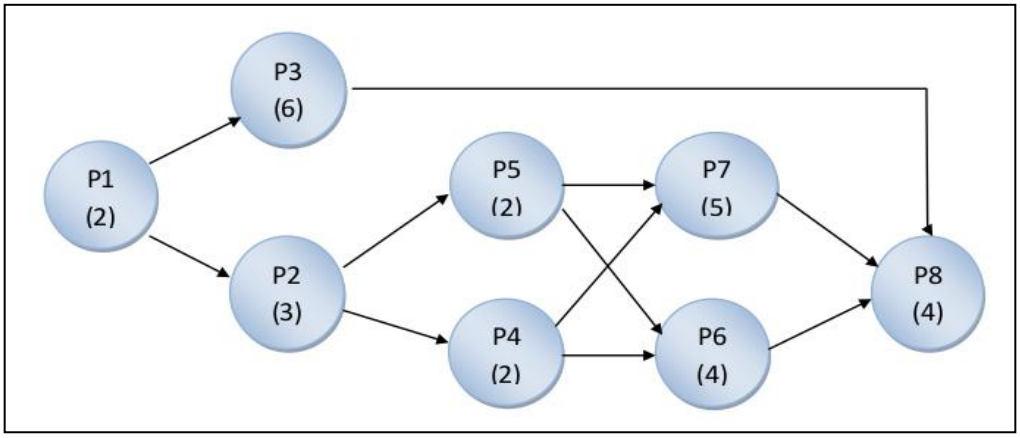
Numerical

Consider the dependency graph between the processes. At what time all the processes complete their execution using 2 CPUs (i.e. 2 processors) ?

Note: a) Use Non-preemptive mode

b) One process cannot share 2 CPUs at the same time





There are two CPUs, so we make a Gnatt chart of two parallel processors CPU-1 and CPU-2

Step 1: We can assign the first process P1 to either CPU-1 or CPU-2. Let's assign it to CPU-2.

CPU-1	Idle	
CPU-2	P1	

0 2

Step 2: P1 ends in 2s. So, let's assign P3 to CPU-2 and P2 to CPU-1. Also, CPU-1 becomes idle at t=5, so we can assign P4 to it at t=5.

			5	7	
CPU-1	Idle	P2	P4		
CPU-2	P1	P3			

0 2 8

Step 3: We can assign P5 to CPU-1 at t=7.

			5	7	9	
CPU-1	Idle	P2	P4	P5		
CPU-2	P1	P3		Idle		

0 2 8 9

Step 4: CPU-2 remains idle from t=8 to t=9 since P6 depends on the completion of (P4,P5), and P5 completes at t=9. Next we can assign P6 to CPU-1 and P7 to CPU-2.

			5	7	9	13	14	
CPU-1	Idle	P2	P4	P5	P6	Idle		
CPU-2	P1	P3		Idle	P7			

0 2 8 9 14

Step 5: CPU-1 remains idle from t=13 to t=14 since P8 needs (P3,P6,P7) to be completed, but P7 completes at t=14. Finally, we can assign P8 to either CPU-1 or CPU-2.

			5	7	9	13	14		18
CPU-1	Idle	P2	P4	P5	P6	Idle	P8		
CPU-2	P1	P3		Idle	P7		Idle		

0 2 8 9 14

Final answer: Thus all the processes complete their execution at **t = 18s**.

Sources

- NPTEL lecture by Prof. Chester Rebeiro, IIT Madras.
- https://www.geeksforgeeks.org/operating-system-multiple-processor-scheduling/