CDAC MUMBAI

COS : Day 2

Processes

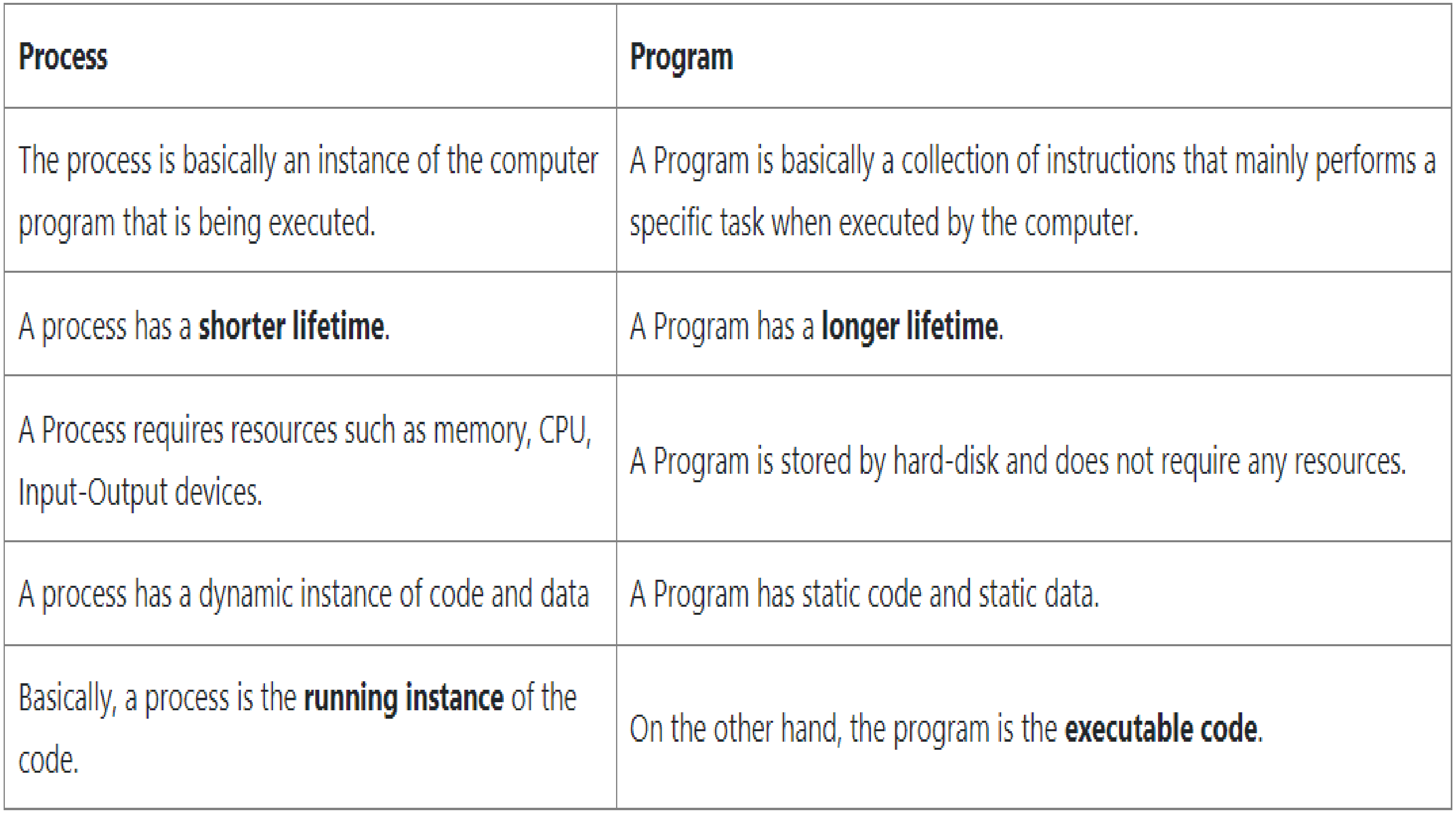
Dr Kiran Waghmare
CDAC Mumbai

Start Slide

Agenda: Processes

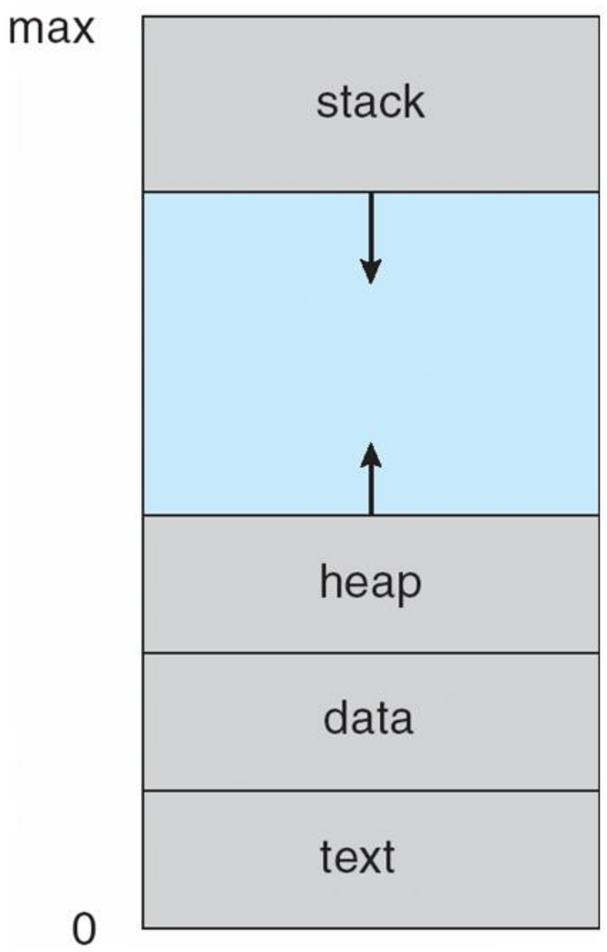
- Preemptive and non preemptive
- Process mgmt
- Process life cycle
- Schedulers
- Scheduling algorithms
- Creation of fork, waitpid, exec system calls
- Orphan and zombie

Process vs Program



Process	Program
The process is basically an instance of the computer program that is being executed.	A Program is basically a collection of instructions that mainly performs a specific task when executed by the computer.
A process has a **shorter lifetime**.	A Program has a **longer lifetime**.
A Process requires resources such as memory, CPU, Input-Output devices.	A Program is stored by hard-disk and does not require any resources.
A process has a dynamic instance of code and data	A Program has static code and static data.
Basically, a process is the **running instance** of the code.	On the other hand, the program is the **executable code**.

Process in Memory



Process in Operating System

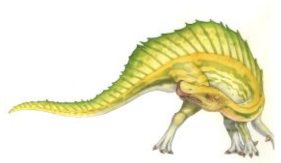
- A process is a **program in execution** which then forms the basis of all computation.
- The process is not as same as program code but a lot more than it.
- A process is an 'active' entity as opposed to the program which is considered to be a 'passive' entity.
- Attributes held by the process include hardware state, memory, CPU, etc.

- **Process memory** is divided into four sections for efficient working :

- The **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The **Data section** is made up of the global and static variables, allocated and initialized prior to executing the main.
- The **Heap** is used for the dynamic memory allocation and is managed via calls to new, delete, malloc, free, etc.
- The **Stack** is used for local variables. Space on the stack is reserved for local variables when they are declared.

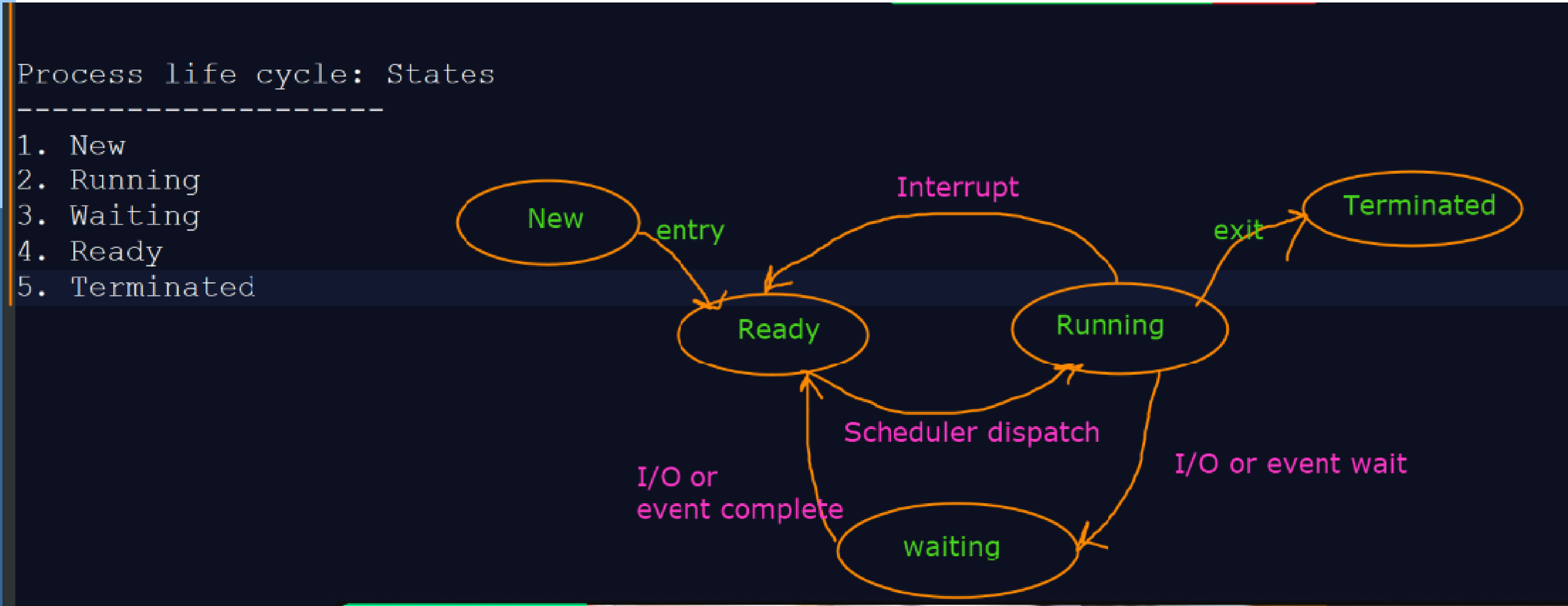
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- **Process – a program in execution; process execution must progress in sequential fashion**
- A process includes:
  - program counter
  - stack
  - data section



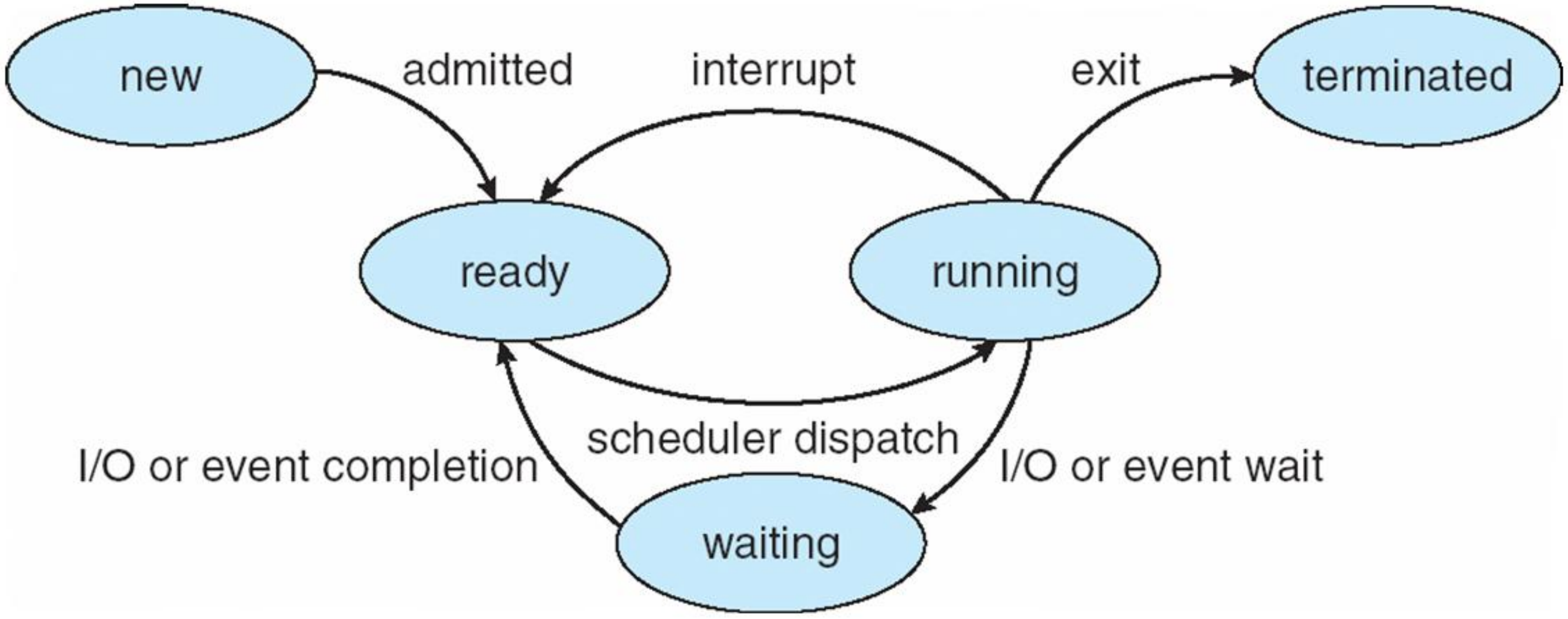
Process State

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution



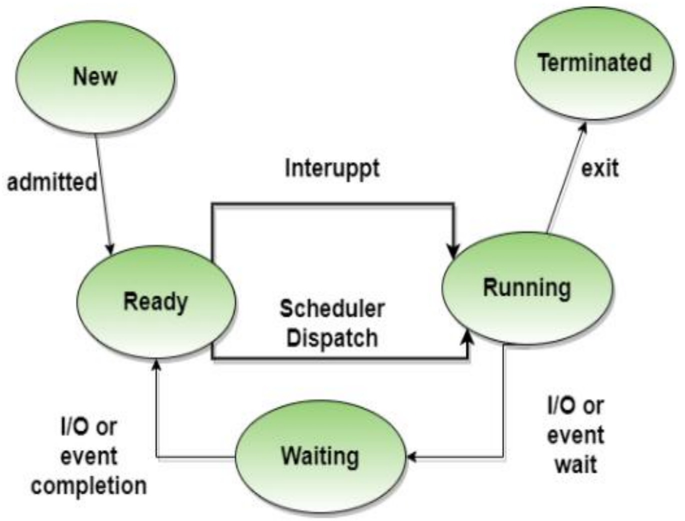
```
Process life cycle: States
--------------------
1. New
2. Running
3. Waiting
4. Ready
5. Terminated
```

Diagram of Process State



The different Process States

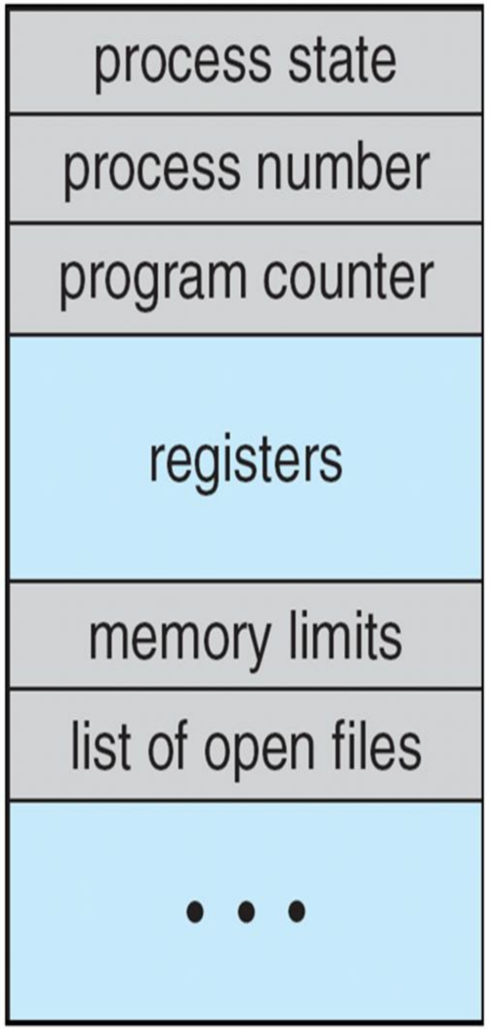
- Processes in the operating system can be in any of the following states:
- NEW- The process is being created.
- READY- The process is waiting to be assigned to a processor.
- RUNNING- Instructions are being executed.
- WAITING- The process is waiting for some event to occur(such as an I/O completion or reception of a signal).
- TERMINATED- The process has finishe



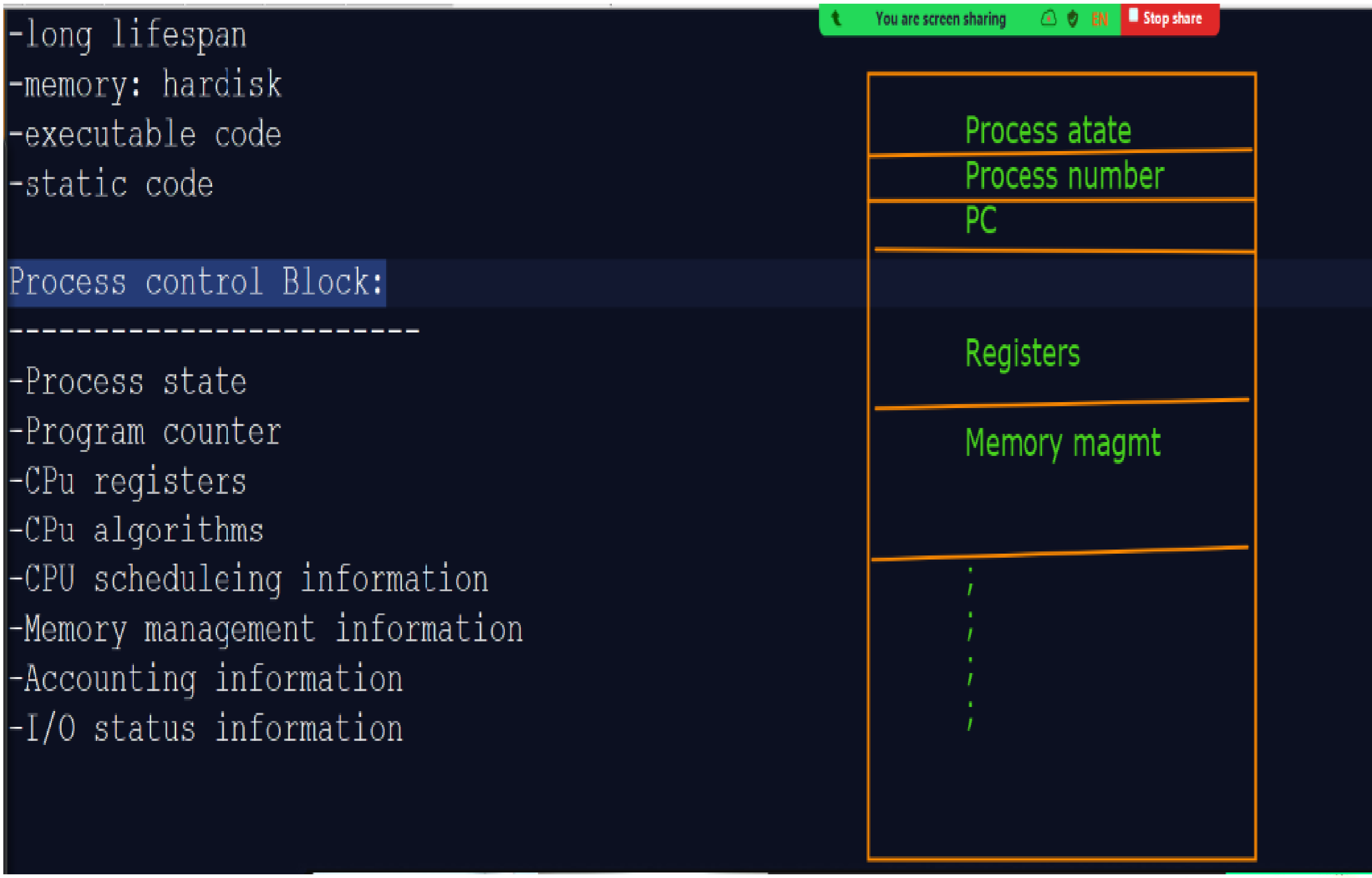
Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information



Process Control Block (PCB)



```
-long lifespan
-memory: hardisk
-executable code
-static code

Process control Block:
-------------------------
-Process state
-Program counter
-CPu registers
-CPu algorithms
-CPU scheduleing information
-Memory management information
-Accounting information
-I/O status information
```

Process Control Block

- There is a Process Control Block for each process, enclosing all the information about the process. It is also known as the task control block. It is a data structure, which contains the following:

- **Process State**: It can be running, waiting, etc.
- **Process ID** and the **parent process ID**.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- **CPU Scheduling** information: Such as priority information and pointers to scheduling queues.
- **Memory Management information**: For example, page tables or segment tables.
- **Accounting information**: The User and kernel CPU time consumed, account numbers, limits, etc.
- **I/O Status information:** Devices allocated, open file tables, etc.

What is Process Scheduling?

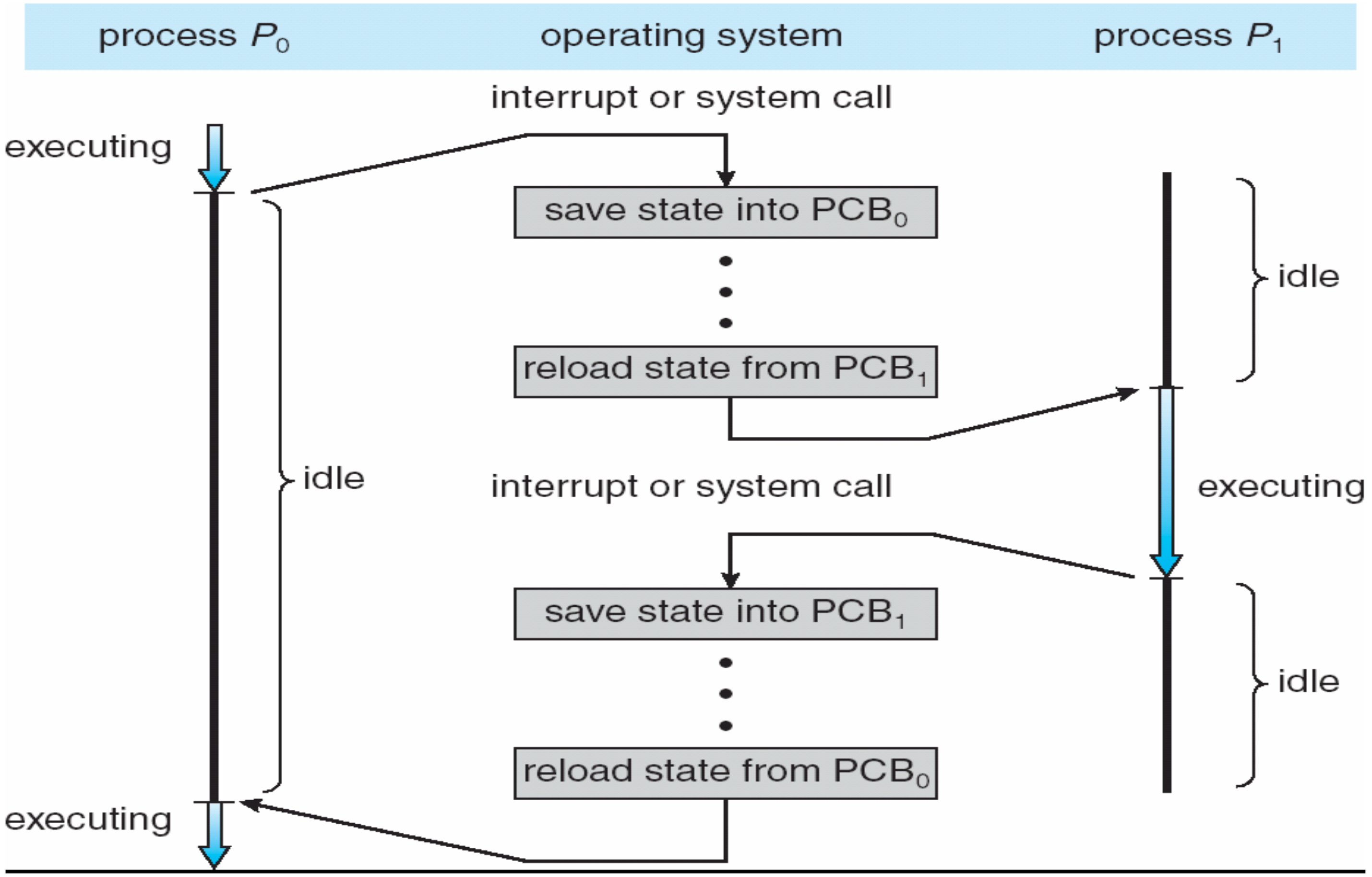
- The act of determining which process is in the ready state, and should be moved to the running state is known as **Process Scheduling**.

- The prime aim of the process scheduling system is to **keep the CPU busy all the time and to deliver minimum response time for all programs**. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Scheduling fell into one of the two general categories:

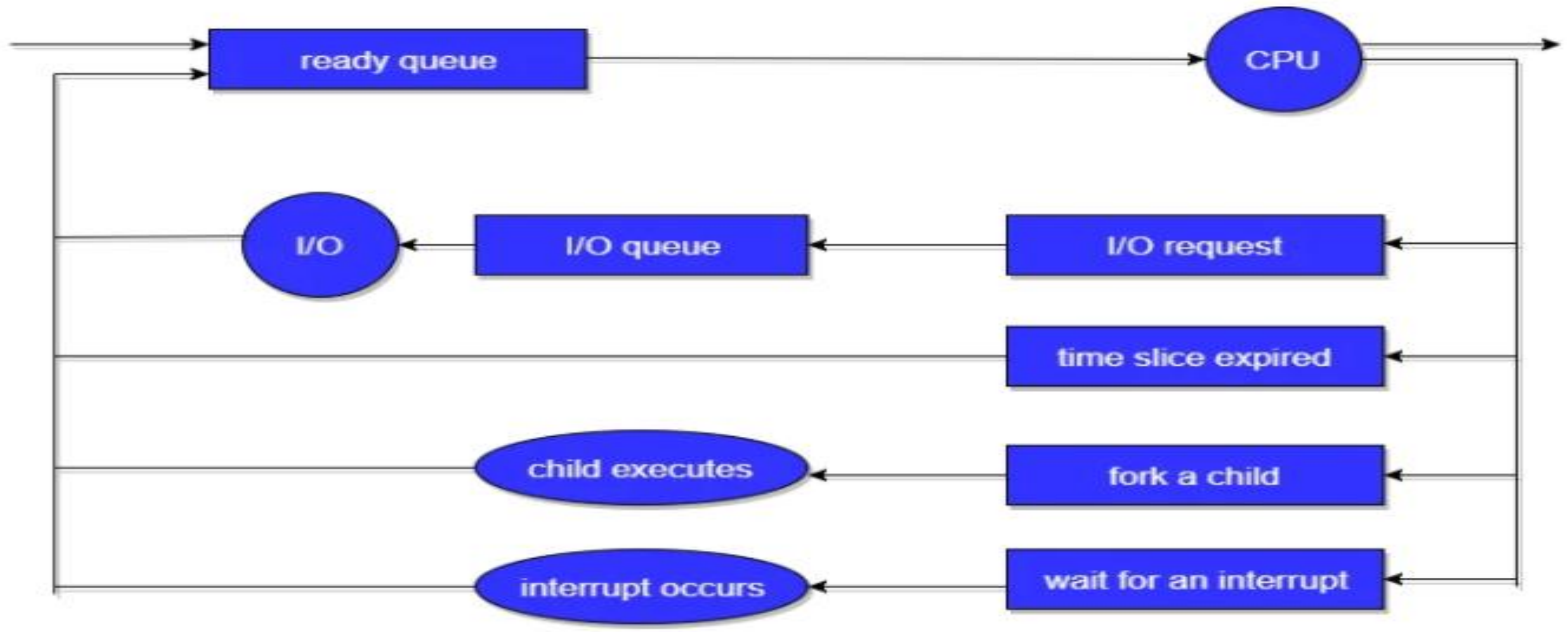
- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

CPU Switch From Process to Process



Process Scheduling

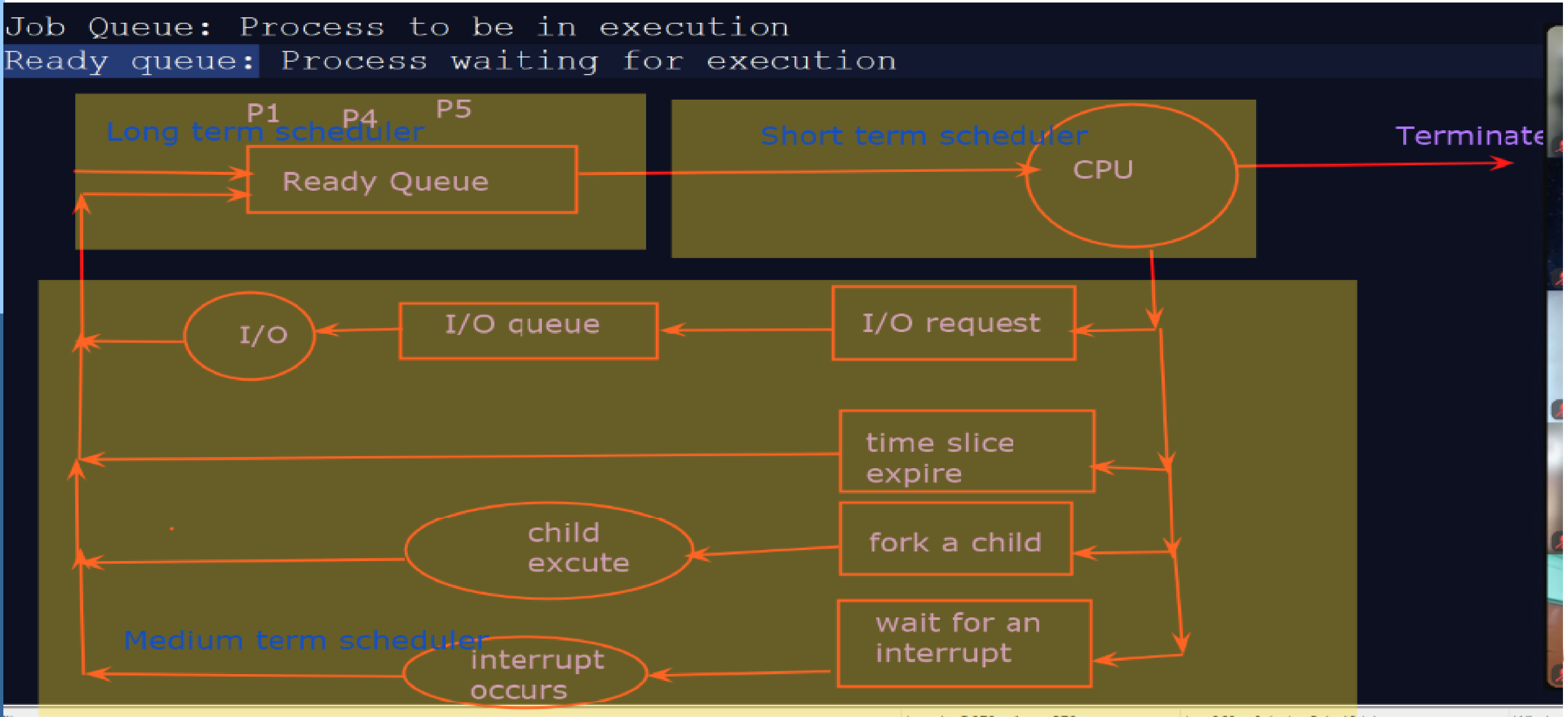
- When there are two or more runnable processes then it is decided by the Operating system which one to run first then it is referred to as Process Scheduling.
- A scheduler is used to make decisions by using some scheduling algorithm.
- Given below are the properties of a **Good Scheduling Algorithm**:
- Response time should be **minimum** for the users.
- The **number of jobs processed per hour should be maximum** i.e Good scheduling algorithm should give maximum throughput.
- The utilization of the CPU should be 100%.
- Each process should get a fair share of the CPU.



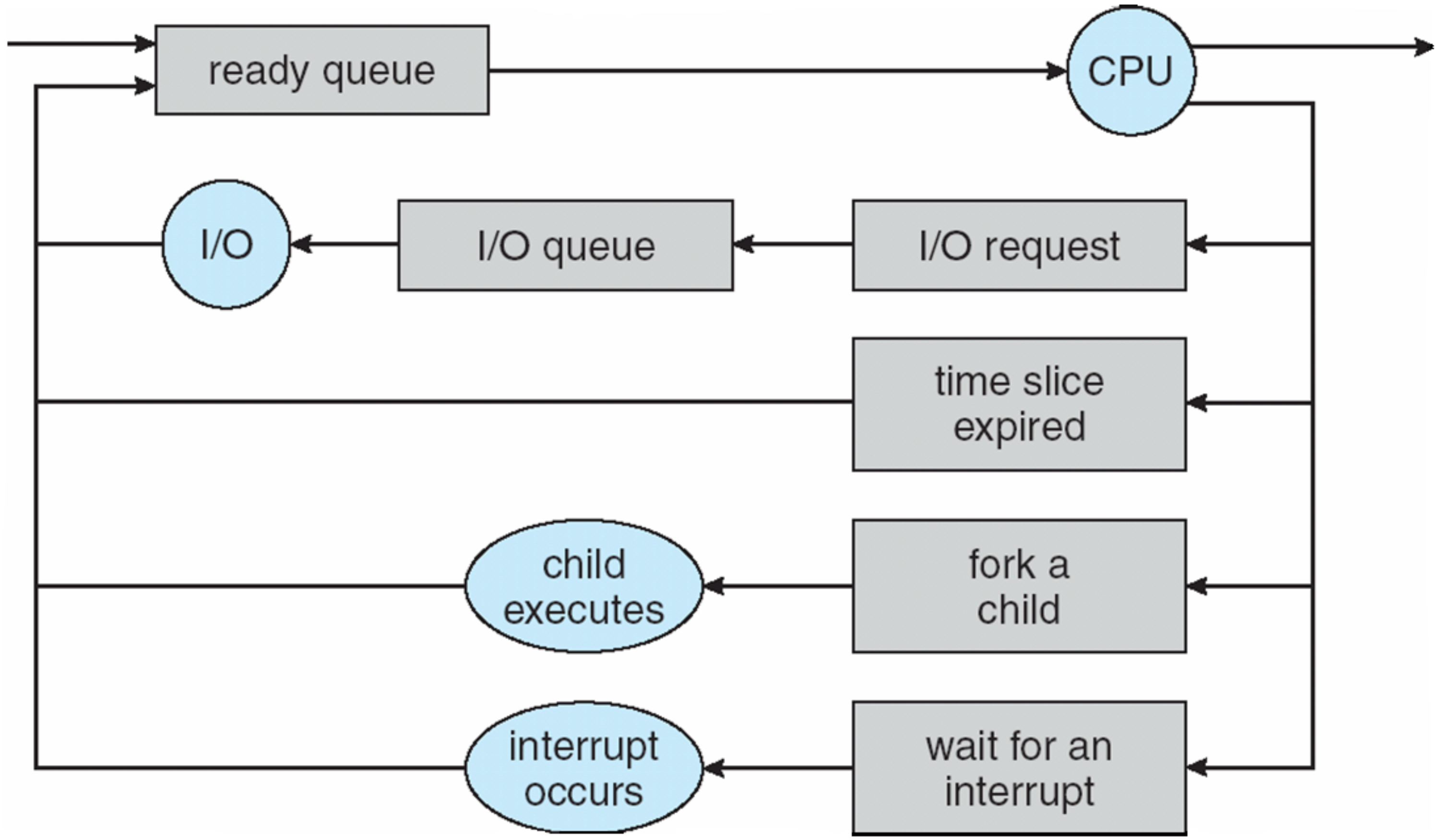
In the first two cases, the process eventually switches from the waiting state to the ready state, and is then put back in the ready queue.
A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

Process Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues


Job Queue: Process to be in execution
Ready queue: Process waiting for execution

Representation of Process Scheduling



What are Scheduling Queues?

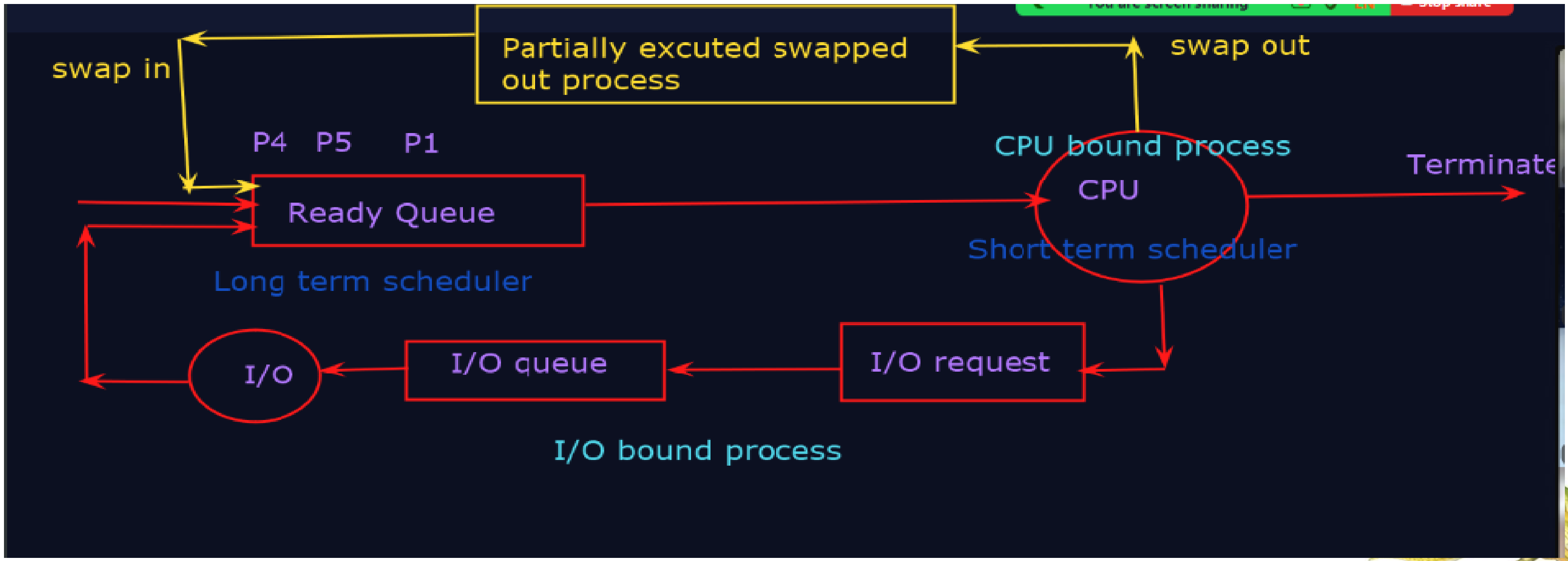
- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues.** There are unique device queues available for each I/O device.
- A new process is initially put in the Ready queue. It waits in the ready queue until it is selected for execution(or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the I/O queue.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

Types of Schedulers

- There are three types of schedulers available:

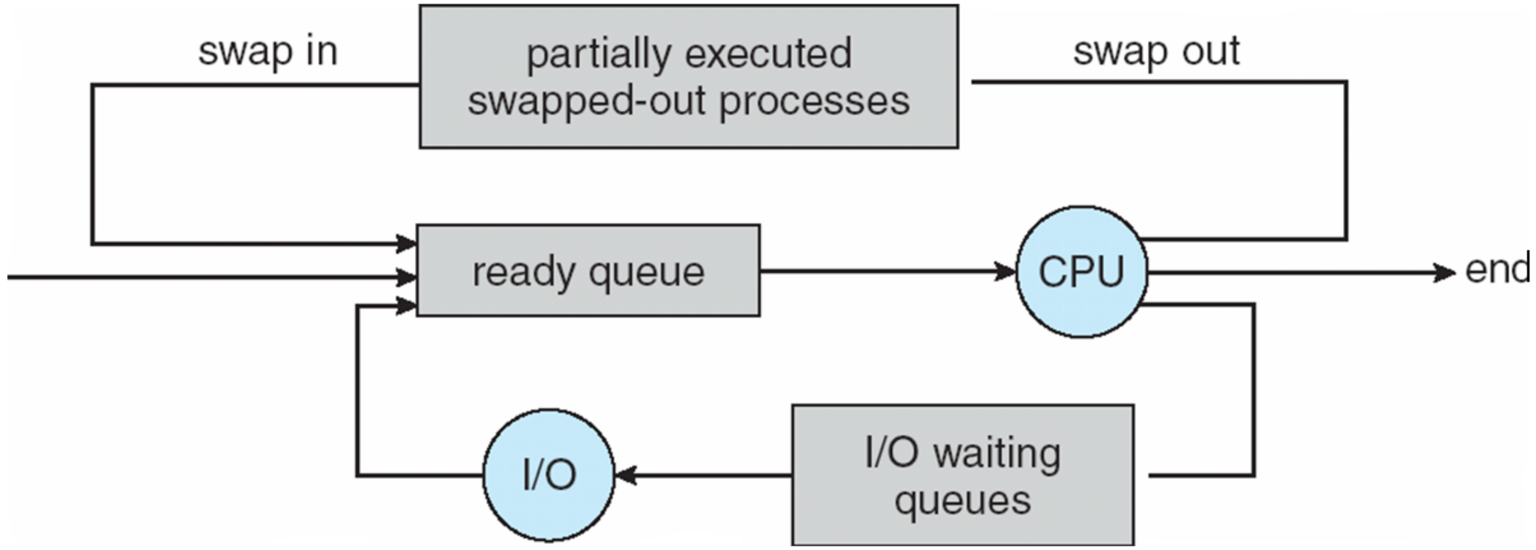
- Long Term Scheduler
- Short Term Scheduler
- Medium Term Scheduler



Schedulers

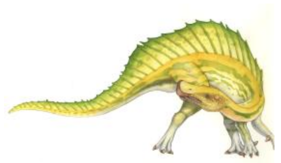
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

Addition of Medium Term Scheduling



Schedulers (Cont)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts



- **Long Term Scheduler**
- Long term scheduler **runs less frequently**. Long Term Schedulers decide which program must get into the job queue. From the job queue, the **Job Processor**, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to **maintain a good degree of Multiprogramming**. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.
- **Short Term Scheduler**
- This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is **to enhance CPU performance and increase process execution rate**.
- **Medium Term Scheduler**
- This scheduler removes the processes from memory (and from active contention for the CPU), and thus **reduces the degree of multiprogramming**. At some later time, the process can be **reintroduced into memory and its execution van be continued where it left off.** This scheme is called **swapping**. The process is swapped out, and is later swapped in, by the medium term scheduler.
- Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up. This complete process is descripted in the below diagram:

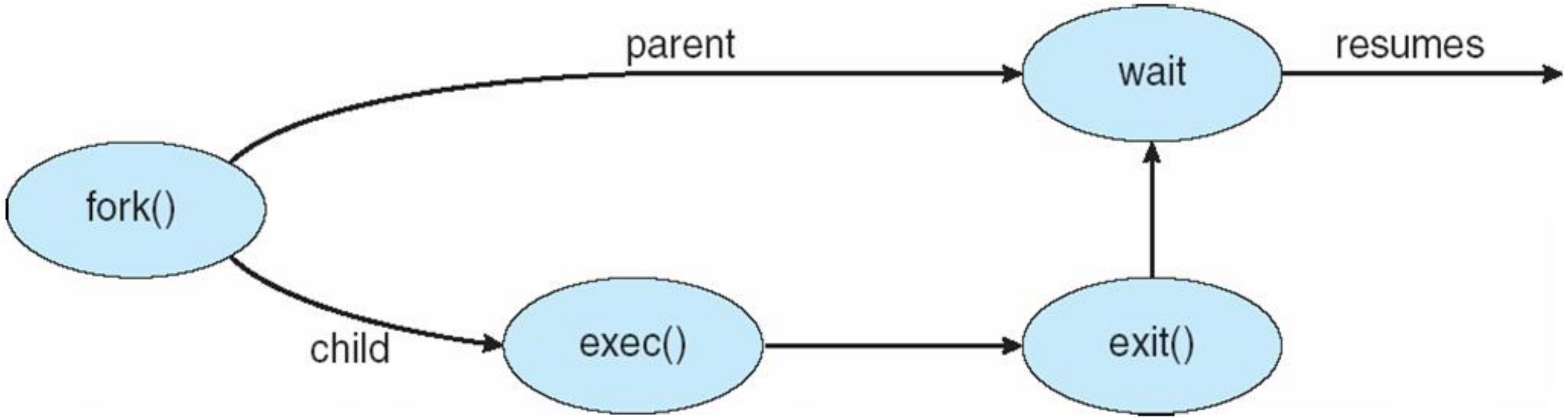
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

What is Context Switch?

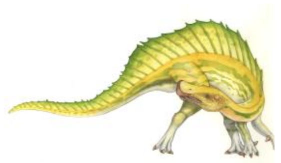
- Switching the CPU to another process requires **saving** the state of the old process and **loading** the saved state for the new process. This task is known as a **Context Switch.**
- The **context** of a process is represented in the **Process Control Block(PCB)** of a process; **it includes the value of the CPU registers**, the process state and memory-management information. When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions(such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.
- Context Switching has become such a **performance bottleneck** that programmers are using new structures(threads) to avoid it whenever and wherever possible.

Process Creation

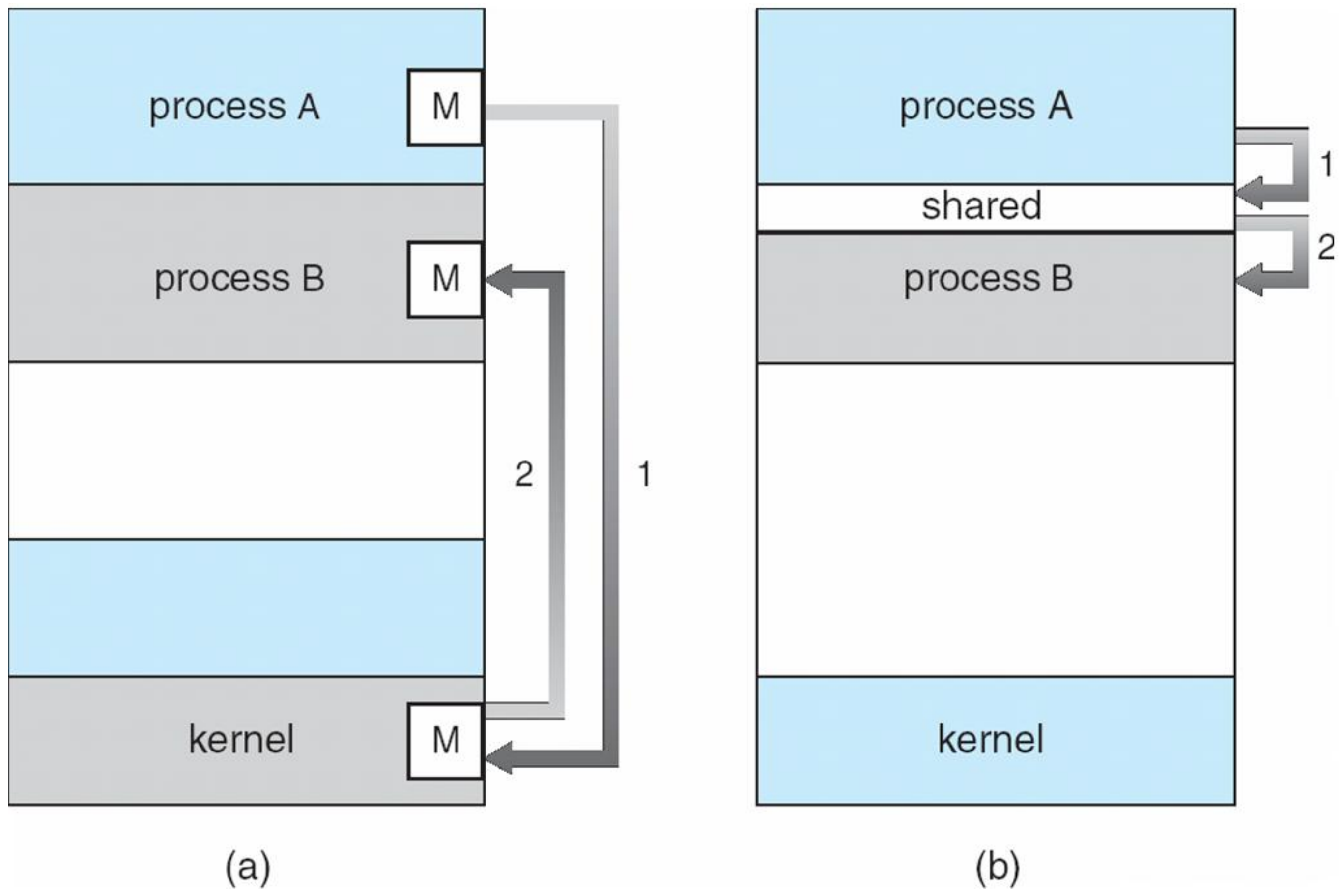


Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing



Communications Models

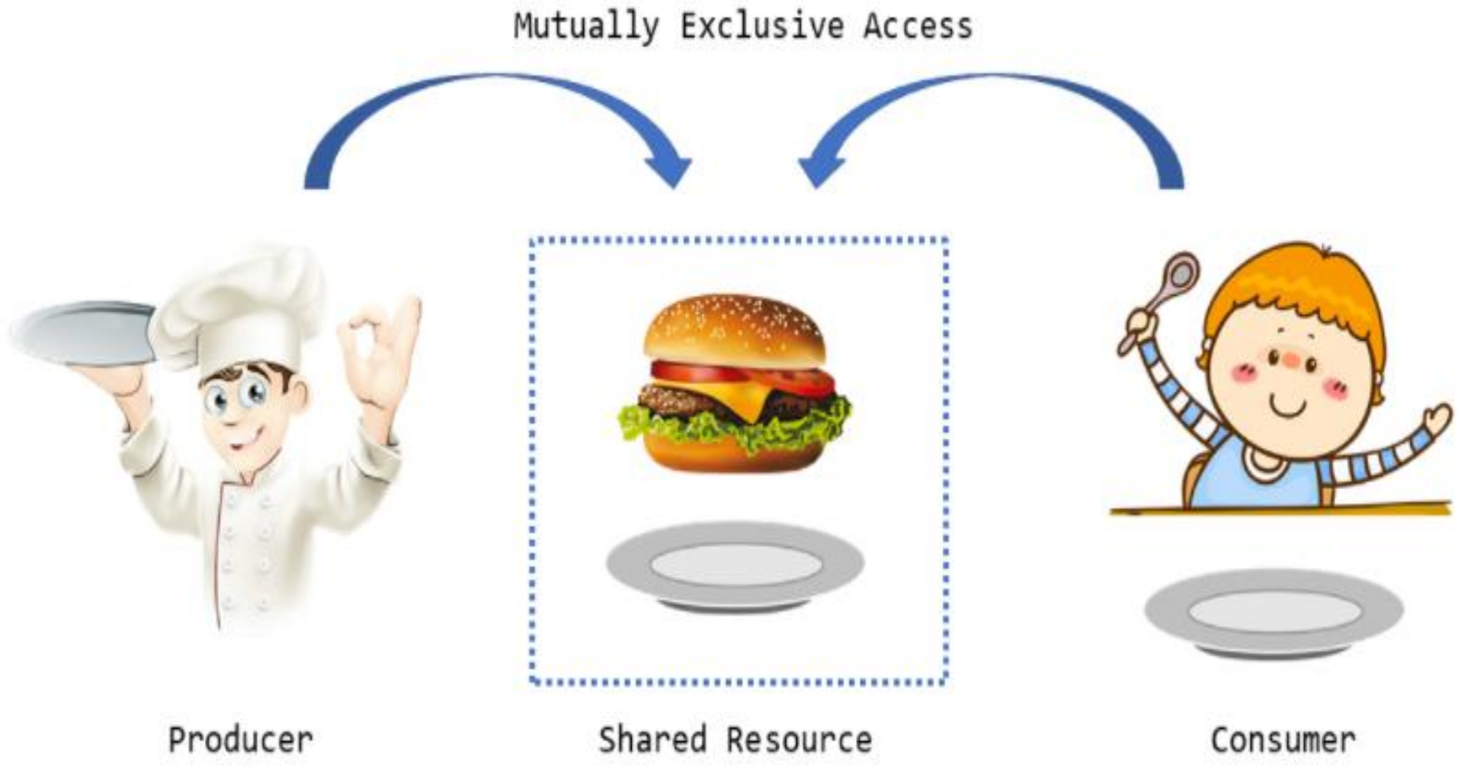


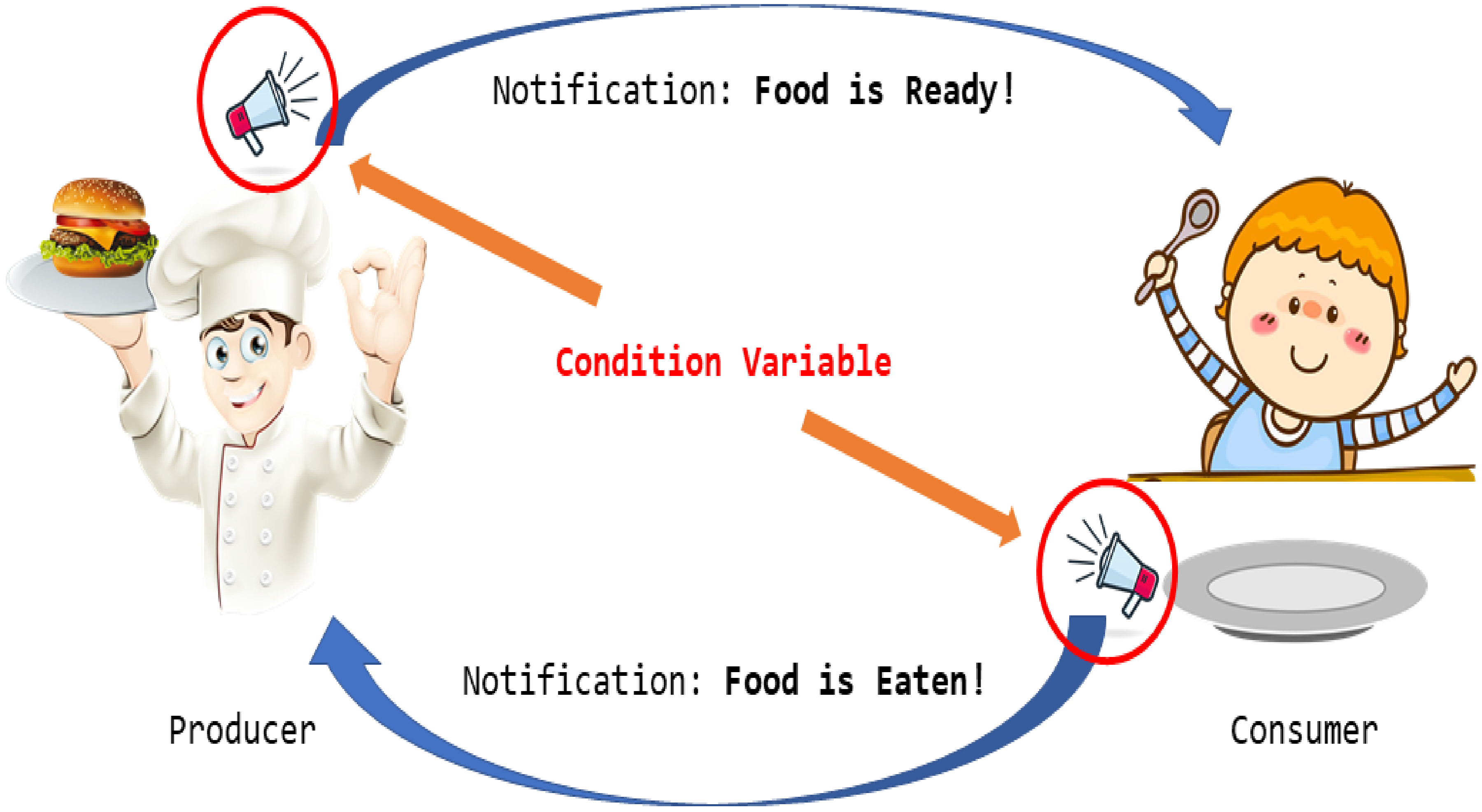
Cooperating Processes

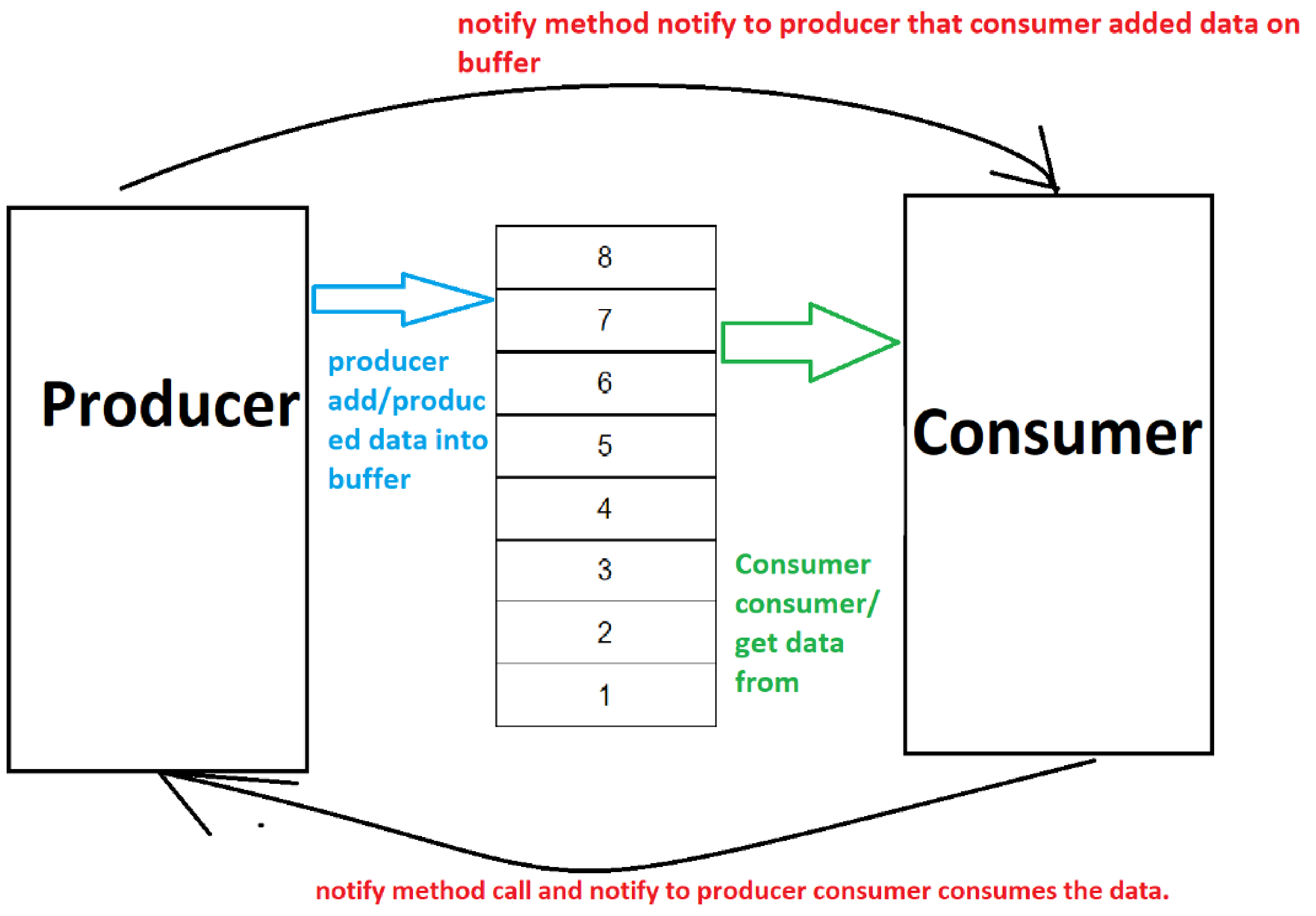
- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

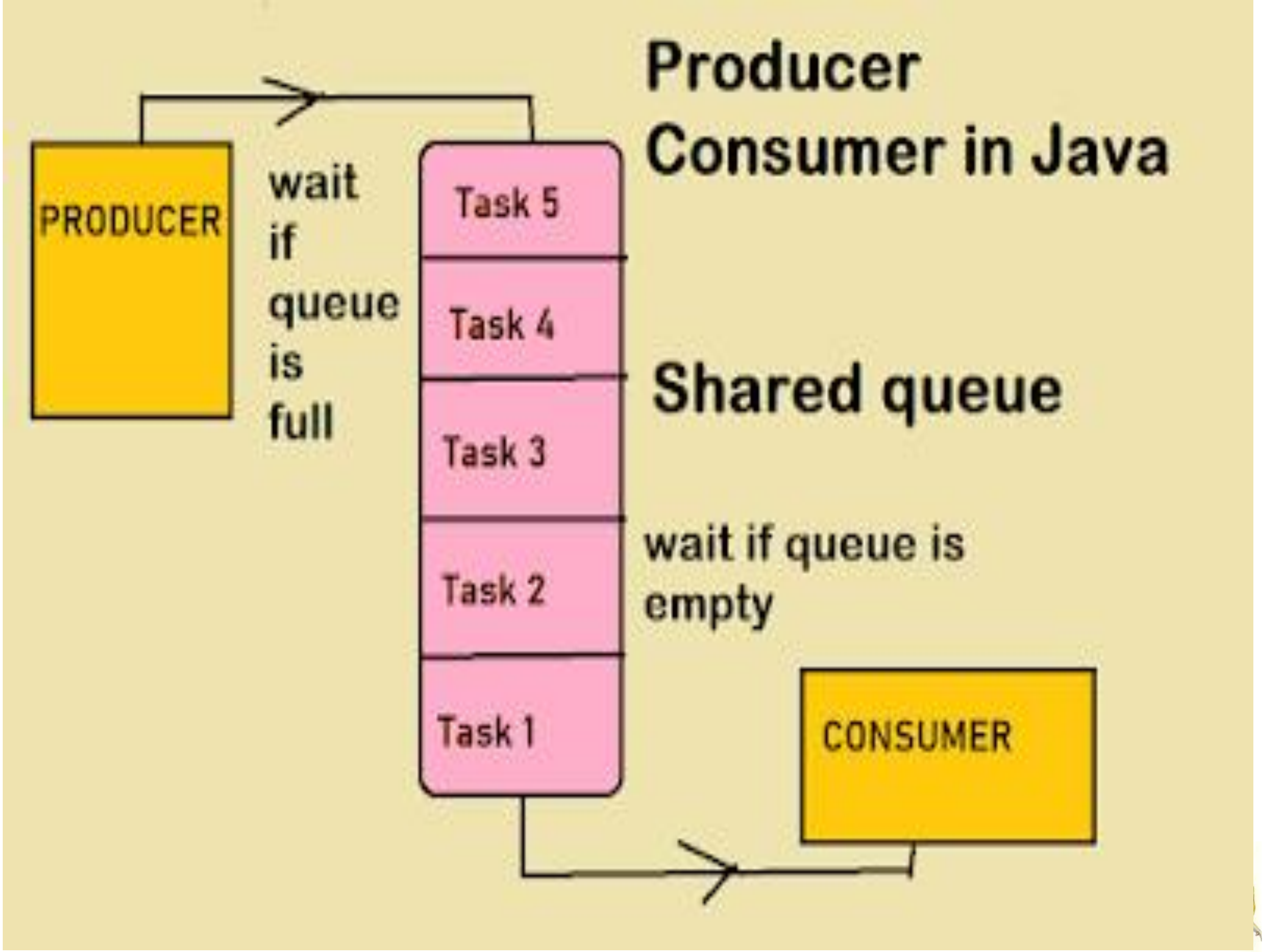
Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer
  - *bounded-buffer* assumes that there is a fixed buffer size


Mutually Exclusive Access
Producer
Shared Resource
Consumer


Notification: **Food is Ready!**
Condition Variable
Notification: **Food is Eaten!**
Producer
Consumer


notify method notify to producer that consumer added data on buffer
Producer
producer add/produced data into buffer
8
7
6
5
4
3
2
1
Consumer consumer/ get data from
Consumer
notify method call and notify to producer consumer consumes the data.


Producer
Consumer in Java
PRODUCER
wait if queue is full
Task 5
Task 4
Task 3
Task 2
Task 1
Shared queue
wait if queue is empty
CONSUMER