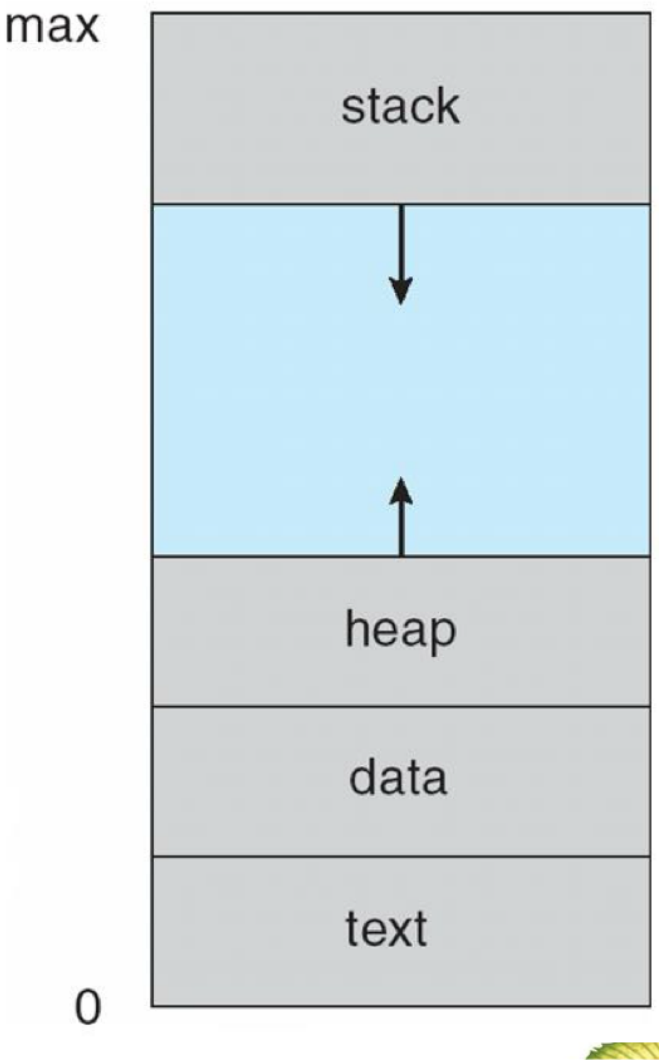
Processes

→ Process is program execution in a sequential fashion

- There are multiple parts present :-



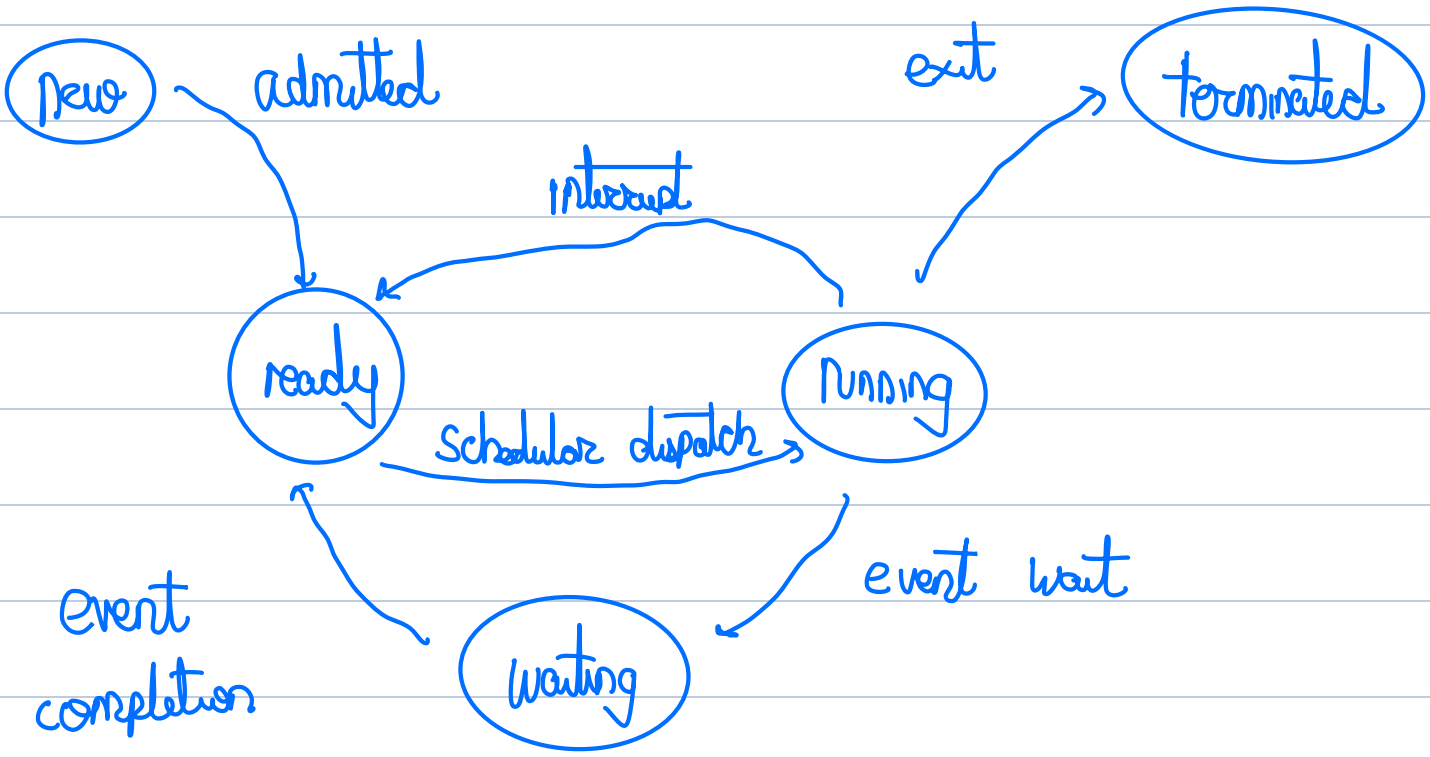
- Text section, includes current activity represented by program counter and processor registers

- Stack containing temporary data

- Data section containing global variables

- Heap containing memory dynamically allocated during run time

→ Process Management Activities -



New - 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

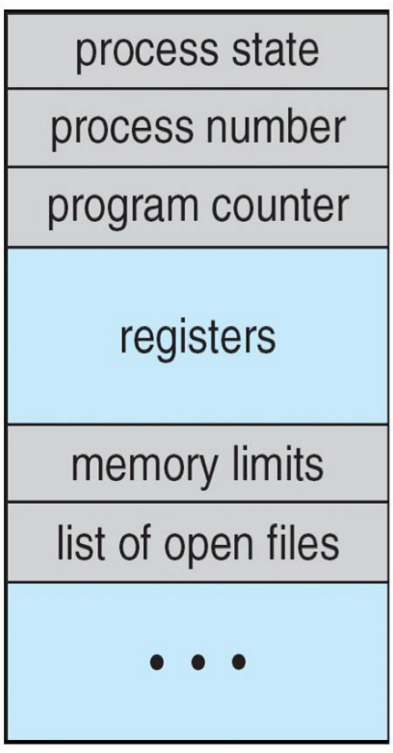
→ Process control Block -

- Data structure used for every process

- PCB is identified by an integer process id (PID)

- PCB keeps all the information needed to keep track of a process as listed below

- PCB is maintained for a process throughout its lifetime and is deleted once the process terminates



1) Process state - running, waiting etc

2) Program counter - location of instruction to next execute

3) CPU registers - contents of all process centric registers like accumulators, index registers etc

4.) CPU scheduling information - priorities, scheduling queue pointers, other scheduling parameters

5.) Memory management information - Such as value of base, and limit reg, page tables, segment table etc

6.) Accounting information - CPU used, clock time elapsed since start, time limits

7.) I/O status information - I/O devices

Threads - 1.) Process has a single thread of execution
2.) Multiple threads → Multiple Processes

→ Process Scheduling -

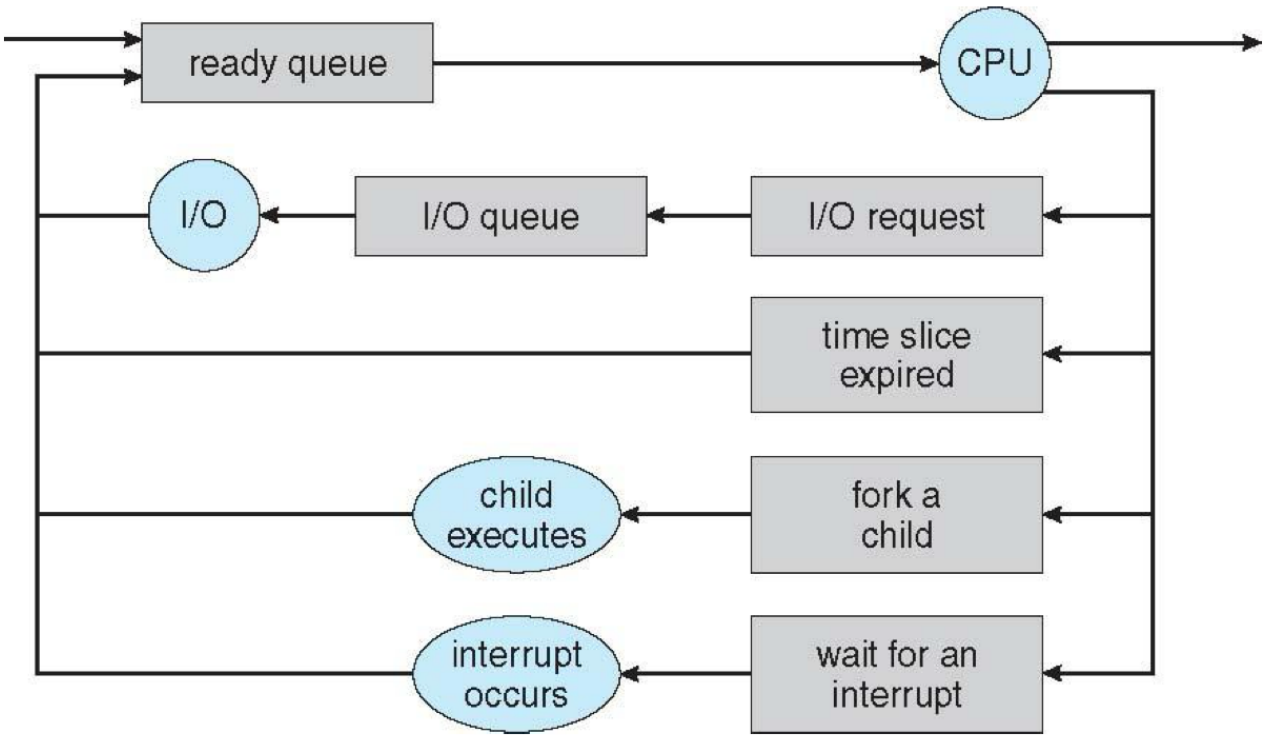
What does a process scheduler do? - Schedules available processes for the CPU to execute.

1.) Job queue - set of all processes in the system
2.) Ready queue - set of all processes residing in main memory,

ready and waiting to execute

3) Device queues - set of processes waiting for an I/O device

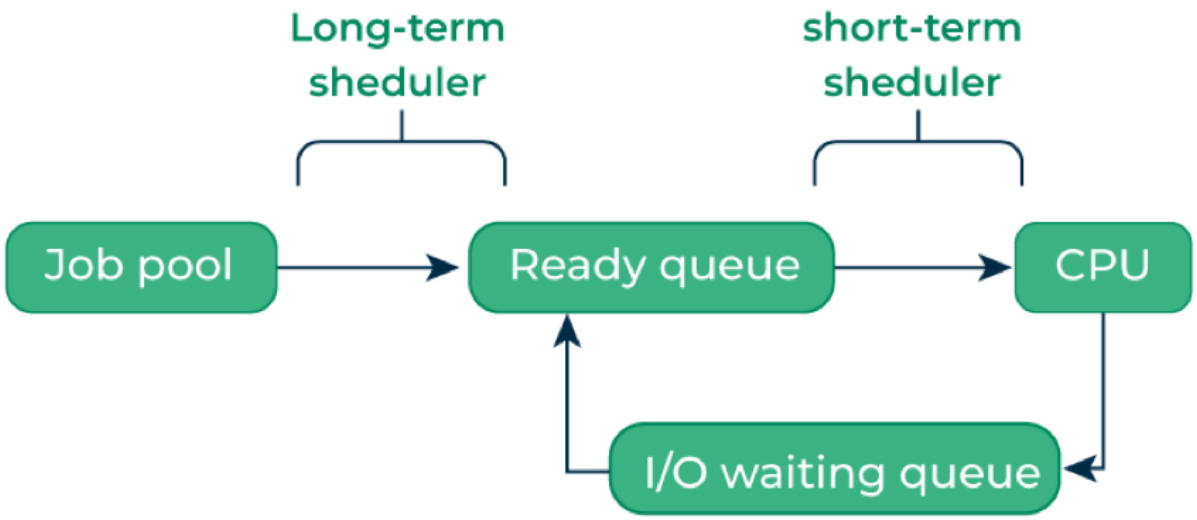
Representation -



→ Schedulers

1) Short term scheduler (CPU scheduler) - selects which process should be executed next and allocates CPU. Selects one process from ready state to running state

2) Long term scheduler- selects which processes should be brought into the ready queue



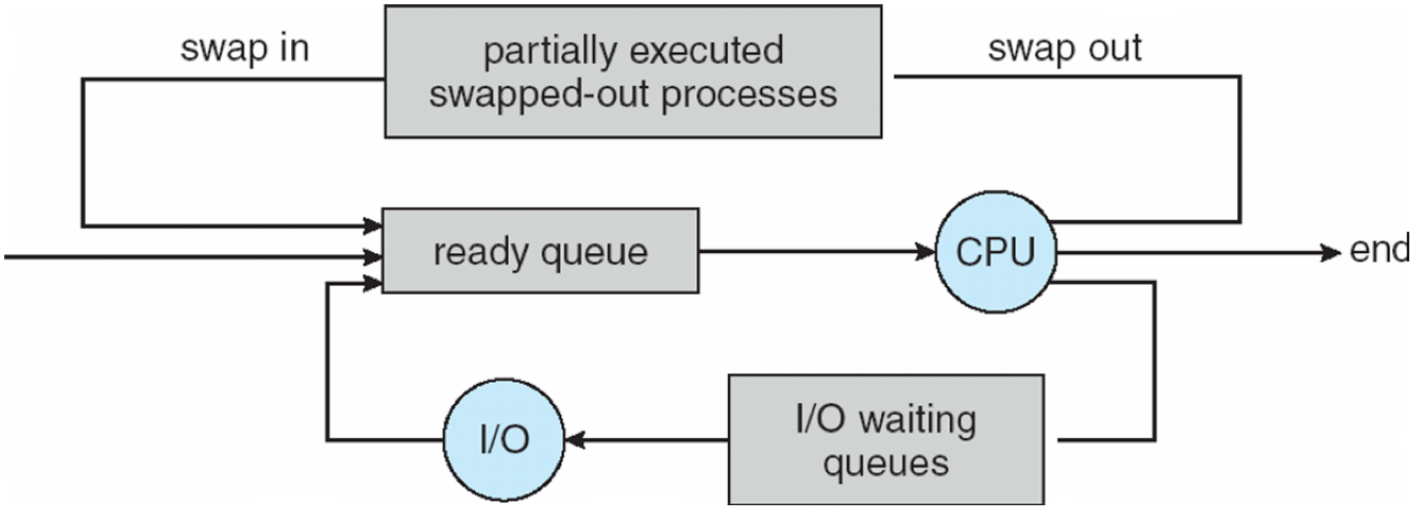
Processes can be described as either :-

(i) I/O bound processes - Spends more time doing I/O than computations, many short CPU bursts

(ii) CPU bound process - spends more time doing computations

→ Addition of Medium Term Scheduling-

- Can be added if degree of multiple programming needs to decrease mostly used in time sharing system



Swapping is reqwod to improve process mix or because of change in memory requirement

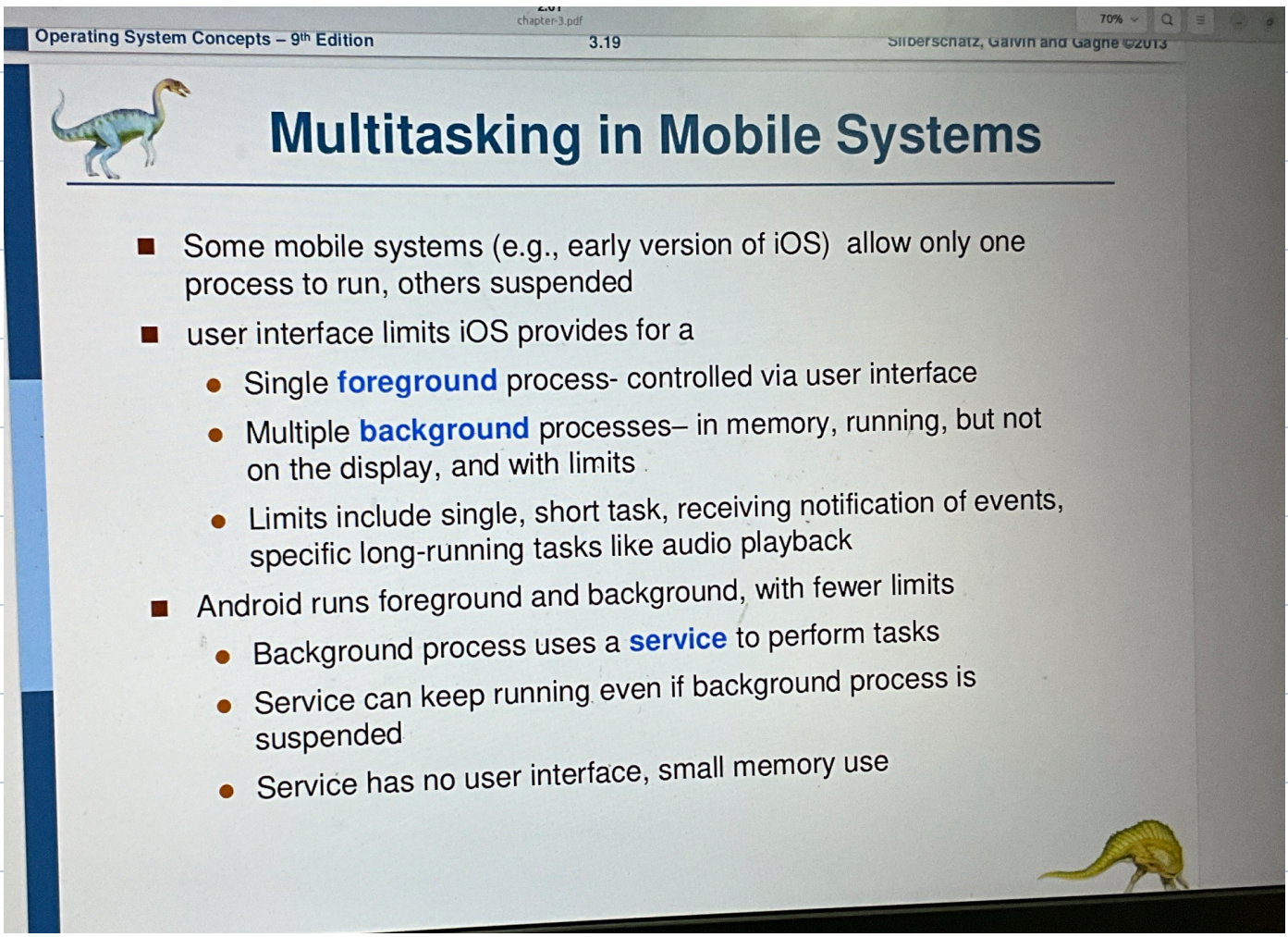
→ Context Switch -

What does a context switch do? - 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.

More complex OS and PCB ∝ longer the context switch

Multiple contexts loaded at once → hardware provides multiple set of registers per CPU

→ Multitasking in Mobile Systems -



→ Operation Processes-

1) Process creation-

- Parent processes create childeren processes which in turn create other processes forming a tree of processes.

- managed using PID (process identifier)

- Resource sharing options - (i) parent and childeren share all resources
(ii) Childeren share subset of parents resources
(iii) Parent and child share no resources

Execution options - (i) parent and childeren execute concurrently
(ii) Parent waits until childeren terminate

Address space - (i) child duplicate of parent
(ii) child has a program loaded into it

→ Process Termination -

- Process executes last statement and then asks the operating system to delete it using the exit() system call
(i) Returns status data from child to parent (via wait())
(ii) Process resources are deallocated by operating system

- Parent may terminate the execution of children processes using the abort() system call. Some reasons are
(i) Child has exceeded allocated resources
(ii) Task assigned to child is no longer required
(iii) The parent is exiting and operating systems does not allow a child to continue if its parent terminates

- Cascading termination - All children, grandchildren etc are terminate
- This termination is initiated by the operating system

Zombie - no parent waiting [did not invoke wait()]
Orphan - parent terminated without invoking wait()

→ Interprocess communication-

Processes within a system may be independent (or) cooperating

Reasons for cooperating processes -

(i) Information Sharing
(ii) Computation speedup
(iii) Modularity
(iv) Convenience

- Cooperating processes need interprocess communication (IPC)

- Two models of IPC :-
(i) Shared memory
(ii) Message passing

→ Cooperating Processes -

- Independent processes 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 :- (i) Information Sharing
  (ii) Computation speed-up
  (iii) Modularity
  (iv) Convenience

→ Interprocess communication - Shared Memory

- Memory shared among the processes to communicate

- Shared memory region resides in the address space of the process creating the shared memory segment. Other processes that wish to communicate using their shared memory segment must attach it to their address space.

- The communication is under the control of the user processes not the operating system

Example: Producer-Consumer Problem

- Paradigm for cooperating processes, ***producer*** process produces information that is consumed by a ***consumer*** process
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be **synchronized**, so that the consumer does not try to consume an item that has not yet been produced.

Types of buffer:

- **unbounded-buffer** places no practical limit on the size of the buffer
- **bounded-buffer** assumes that there is a fixed buffer size