CS 303: Operating Systems
Lecture Set 1

30 July 2018

Instructor

Gourinath B.

POD#1D Room#307

IIT Indore

Is OS mandatory?

- NO, not necessarily, then too much efforts be put…..
- But if we use an OS, advantages:
  - Abstract interfaces to HW access
  - Reusability (no need to write code to implement common behaviour)

What Operating Systems Do?

Several perspectives -

- OS is program most involved with the hardware
  - hardware abstraction
- OS is a resource allocator
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the computer

Operating System

- The low-level software on a computer which provides user-programs (i.e. applications) with necessary execution environment by:

  (a) defining a framework for program execution and

  (b) defining a set of services required

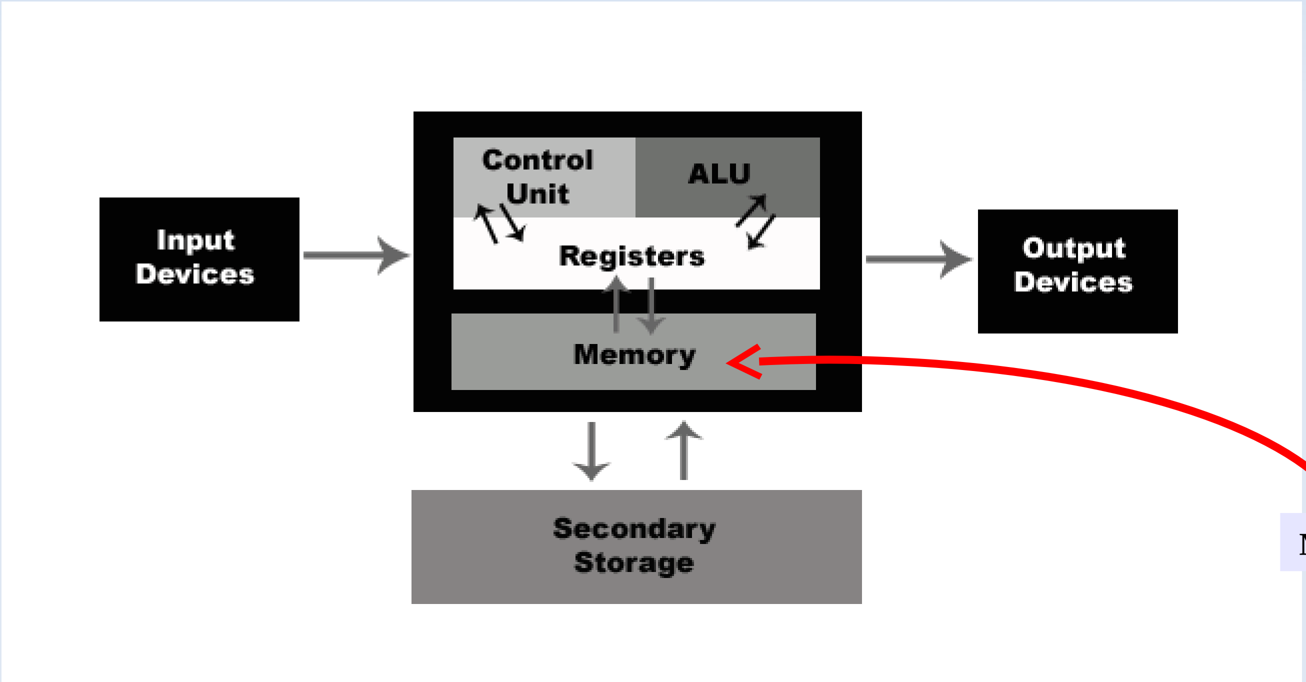
  - When no application program is running, it gives default interface to the user

  - OS: kernel together with
    - set of system programs which use facilities provided by the kernel to perform higher-level functionalities (house-keeping) tasks

- KERNEL: is the core (irreducible core from which entire OS func. Gets delivered)

Computer System Organization
(Abstract view)

Typically, a common bus for all!

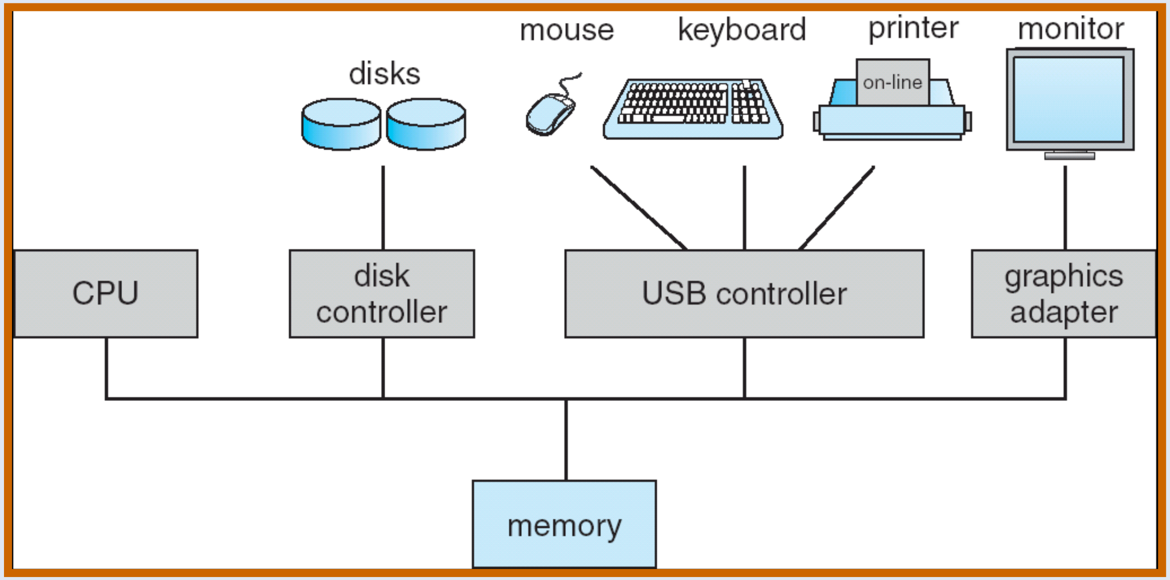


Main Memory

Requires: Dev. Controller HW for delegation → Dev. Driver (SW) is required

Computer System Organization (refined)

- One or more CPUs, device controllers connected through common bus providing access to shared memory
- Concurrent execution (CPUs and devices) competing for memory access (cycles)



Storage Structure

Primary storage

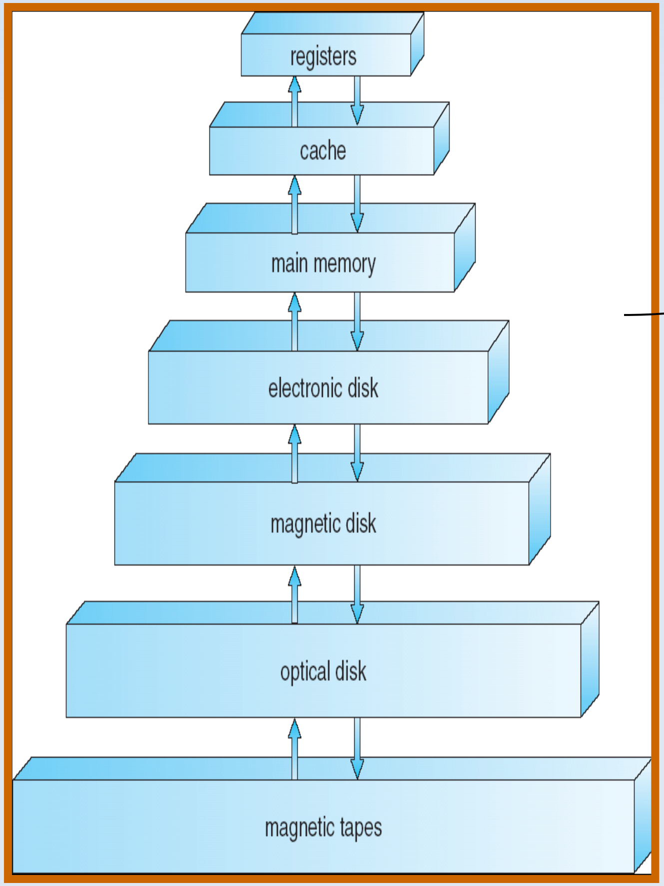
- Main memory, the only mem. directly accessible by CPU
  - Program must be in main memory in order to be executed
  - Main memory usually not large enough to hold all programs and data (paging)
  - Main memory is *volatile*

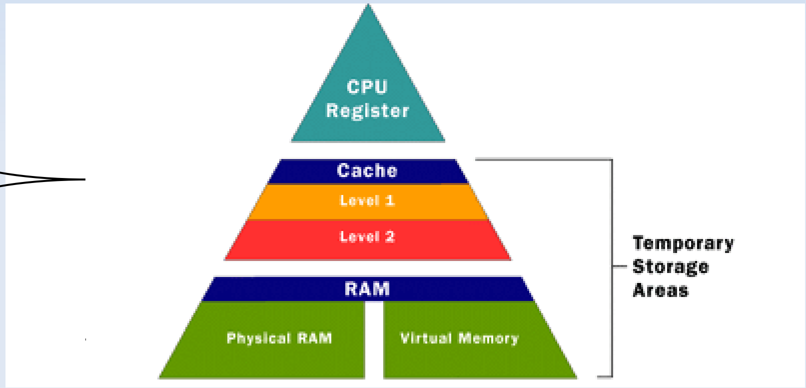
Secondary storage:

- large quantities of data, permanently

In general, we have a hierarchy of storage devices varying by speed, cost, size and volatility

Storage-Device Hierarchy





Architecture Variants

- Single-processor system
  - From PDAs to mainframes
- Multi-processor systems
  - Increase throughput
  - Economy of scale
  - Increased reliability
  - Asymmetric multiprocessing
    - Each processor assigned a specific task (master-slave)
  - Clusters, distributed systems
- Multiple cores, blade servers, etc.

Operating System Services

Services provided to user programs:

- I/O operations
  - User program cannot directly access I/O hardware, OS does the low level part for them
- Communications
  - Both inter-process on the same computer, and between computers over a network
  - via shared memory or through message passing
- Error detection
  - Errors do occur: in the CPU and memory hardware, in I/O devices, in user programs
  -
  - Low level debugging tools really help: DUMA, GDB, DDD, etc.

Memory Management

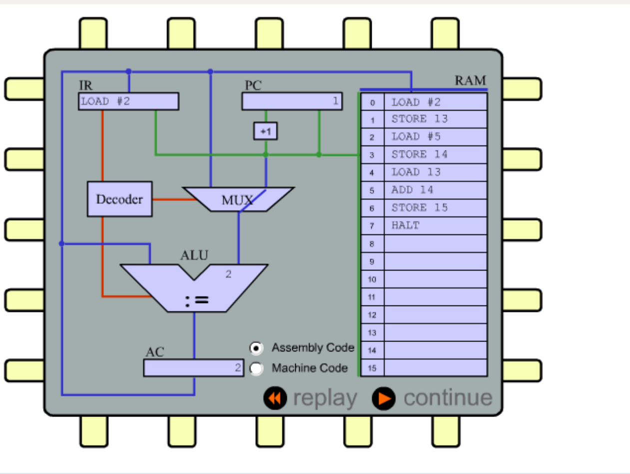
- OS keeps track of which part of memory is currently being used
- Deciding which process to move in or out of the memory
- Allocating and deallocating memory

Storage Management

- Creating and deleting files/directories
- File/directory organization
- Mapping files into secondary storage
- Making file back-ups

Process

- This concept provides a systematic way of monitoring and controlling program execution
- addTwoNums.c → addTwoNums
  - Both are just files residing on secondary memory
  - They are NOT processes
- An (executable) program when loaded into MM



Process

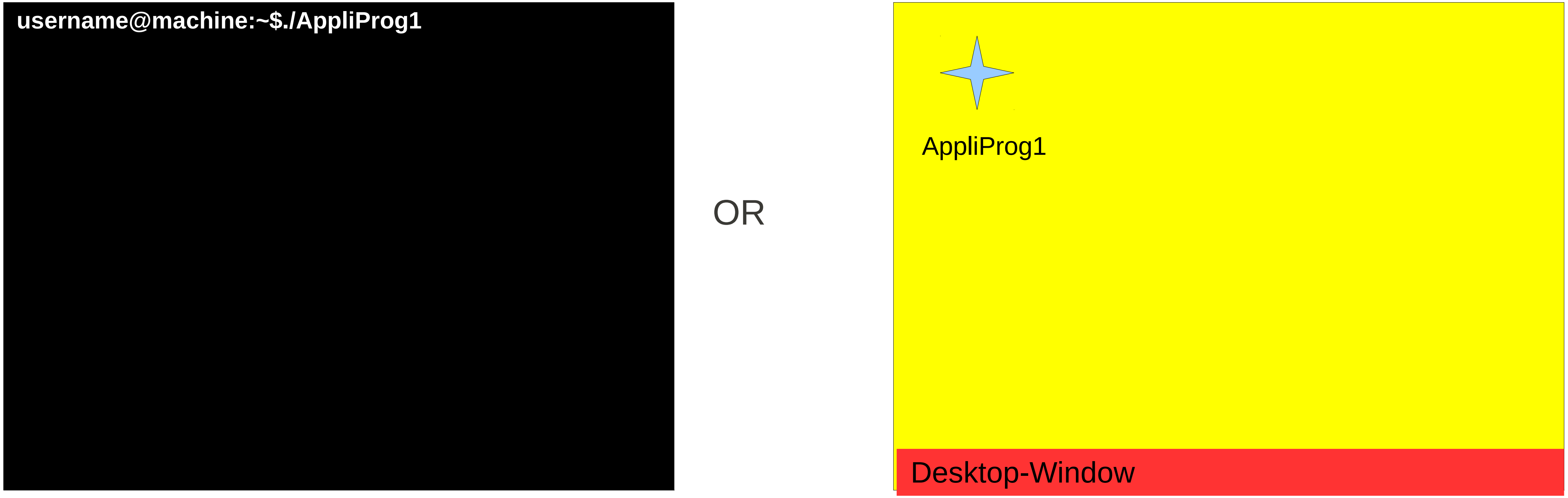
- PROCESS is a program in execution with:
    - associated data (variables, buffers…)
    - execution context: i.e. all the information that (the CPU needs to execute the process + content of the processor registers)
- the OS manages:
    - Creation and deletion of user and system processes
    - Suspending and resuming processes
    - Process synchronization
    - Process communication
    - Deadlocks (priorities)

OS User Interfaces

- Command Line Interface (CLI)/ Character User Interface (CUI)
  - (CLI) Shells: bash, sh, csh, bash, etc.
  - Means of interacting with computer by keying in text
  - Such programs/shells are called CL interpreters
- Graphic User Interface (GUI)
  - interacting via graphical icons and visual indicators
  - GNOME/metacity, KDE/kwin, etc.
- Natural Lang. User Interface (NLUI)
  - Voice based NL UI controls
  - Briana (for win), GNOME Do (for linux), etc.

User Interface Interpreters

- Any interface in essence would have an appropriate interpreter
- On identifying a well formed input, a set of default locations for a matching executable program is looked for
- The matching program is launched
- These interpreters are themselves processes (i.e. programs in execution)

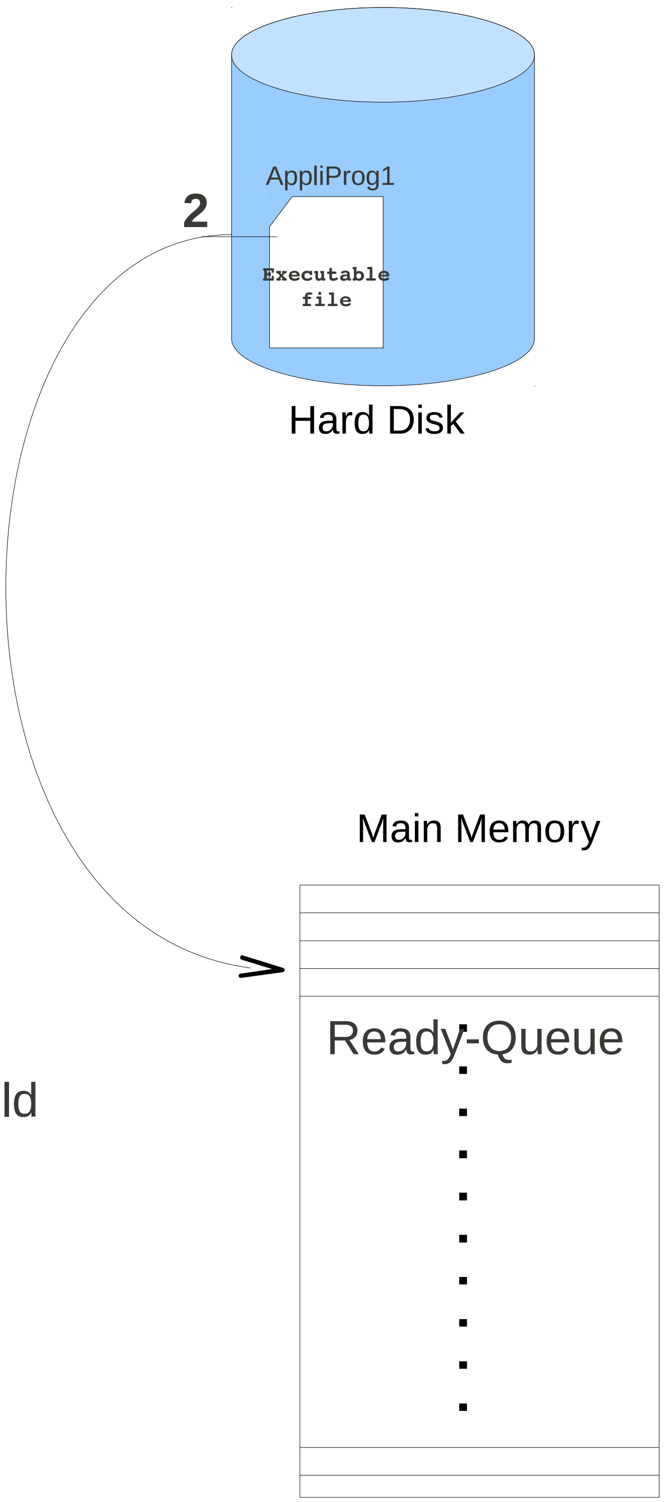


CLI Process
Ex: bash

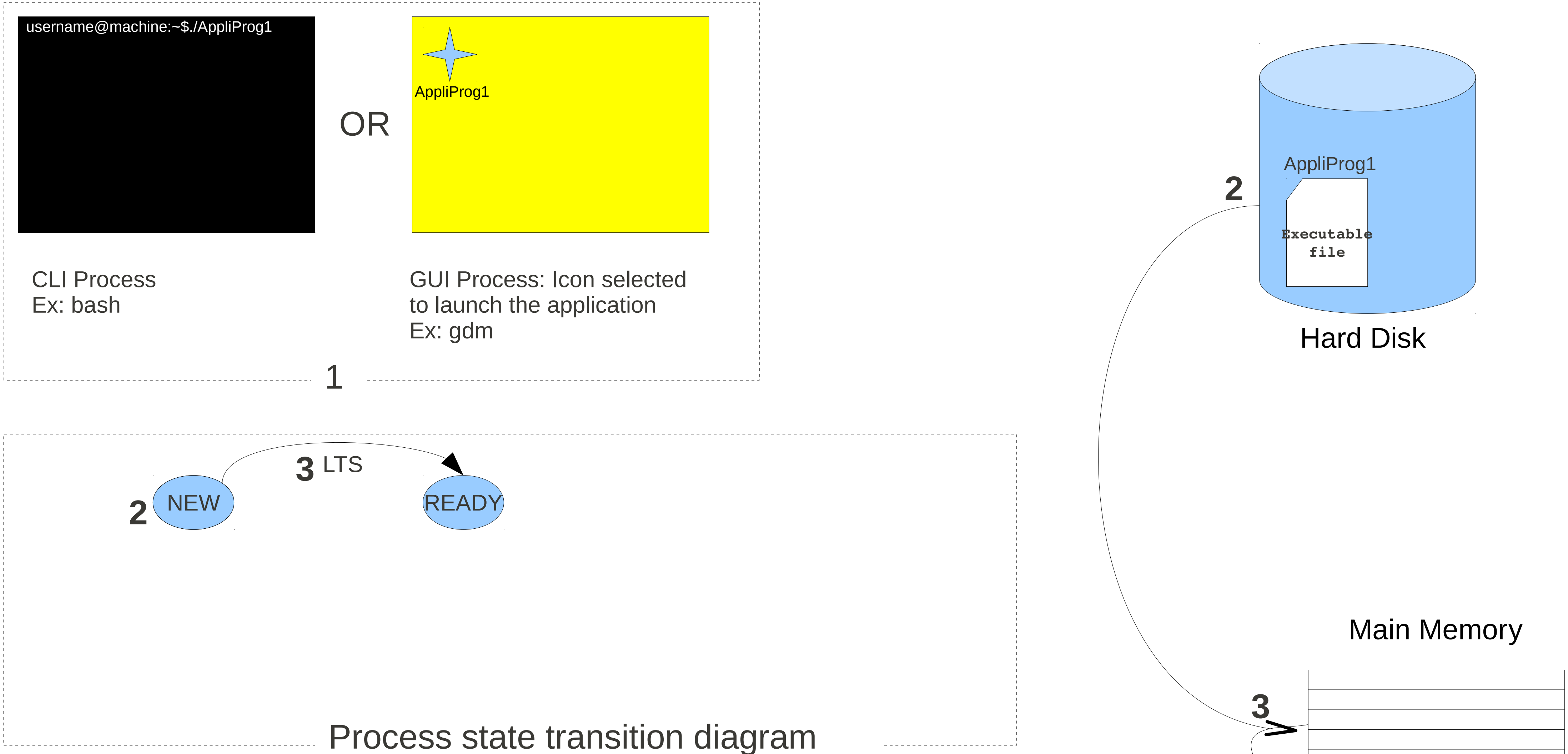
GUI Process: Icon selected to launch the application
Ex: gdm

1

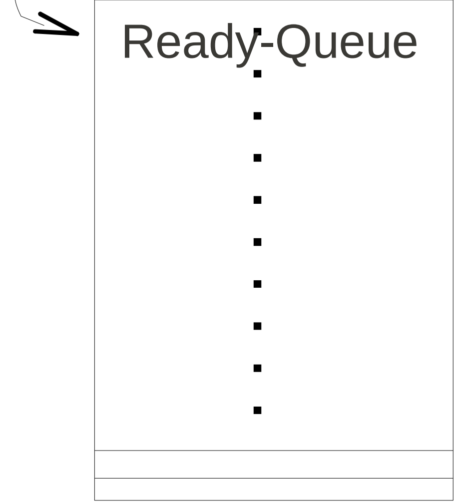
1. User launches “AppliProg1” via CLI or GUI

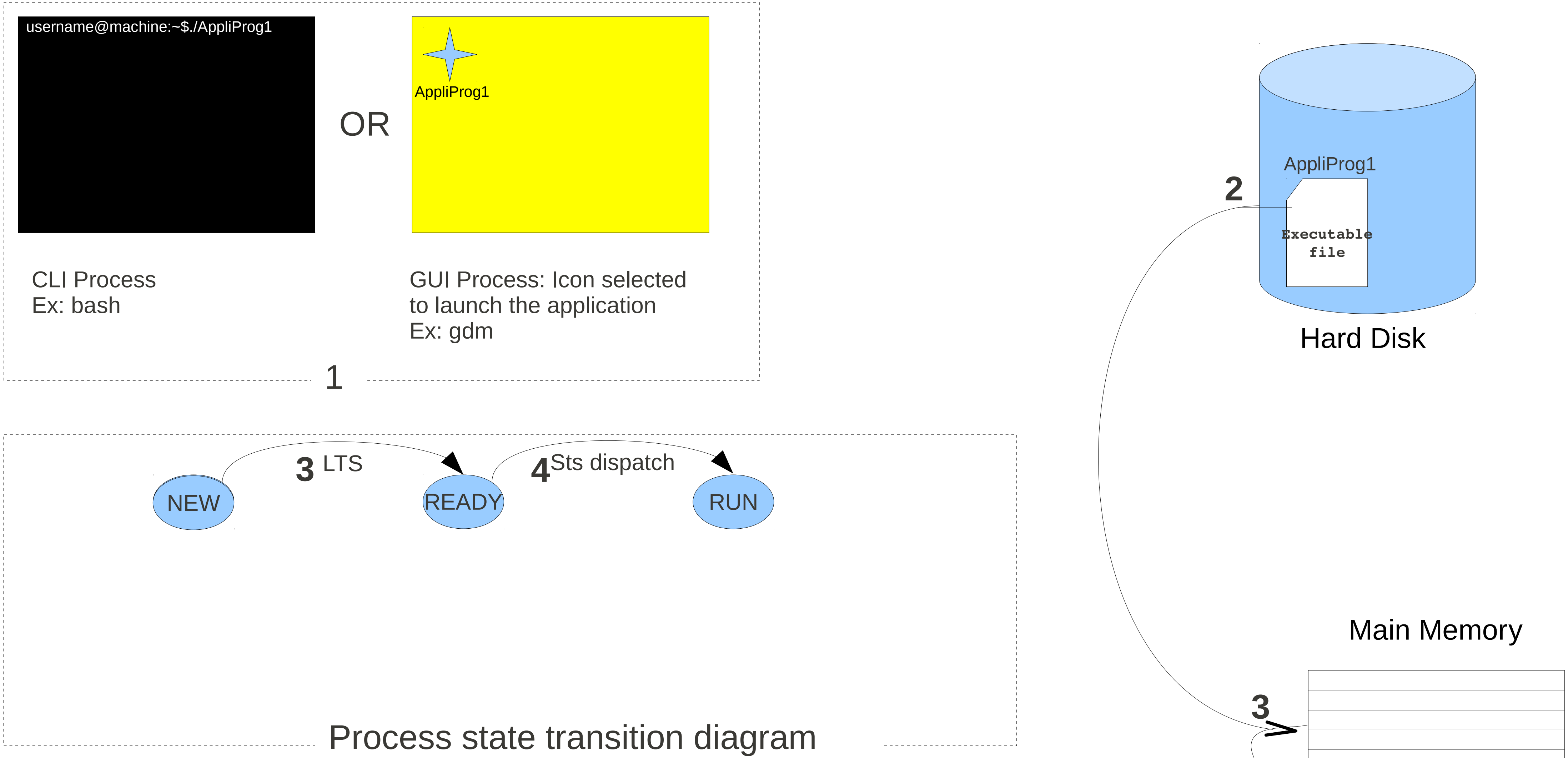


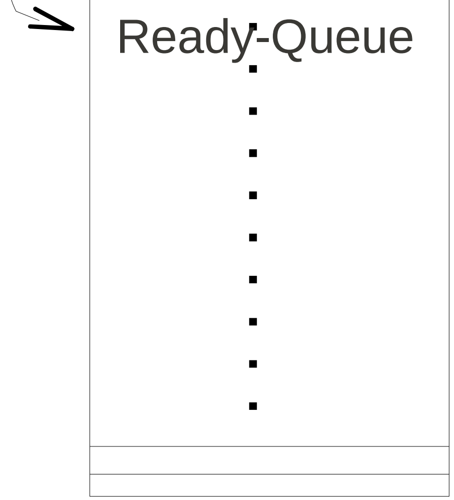
1. User launches “AppliProg1” either by CLI/GUI/...
2. The CLI/GUI process **fork/clones** to execute AppliProg1 as its child
Now this process is in **NEW state.** Here prog loads into memory and process gets its PCB



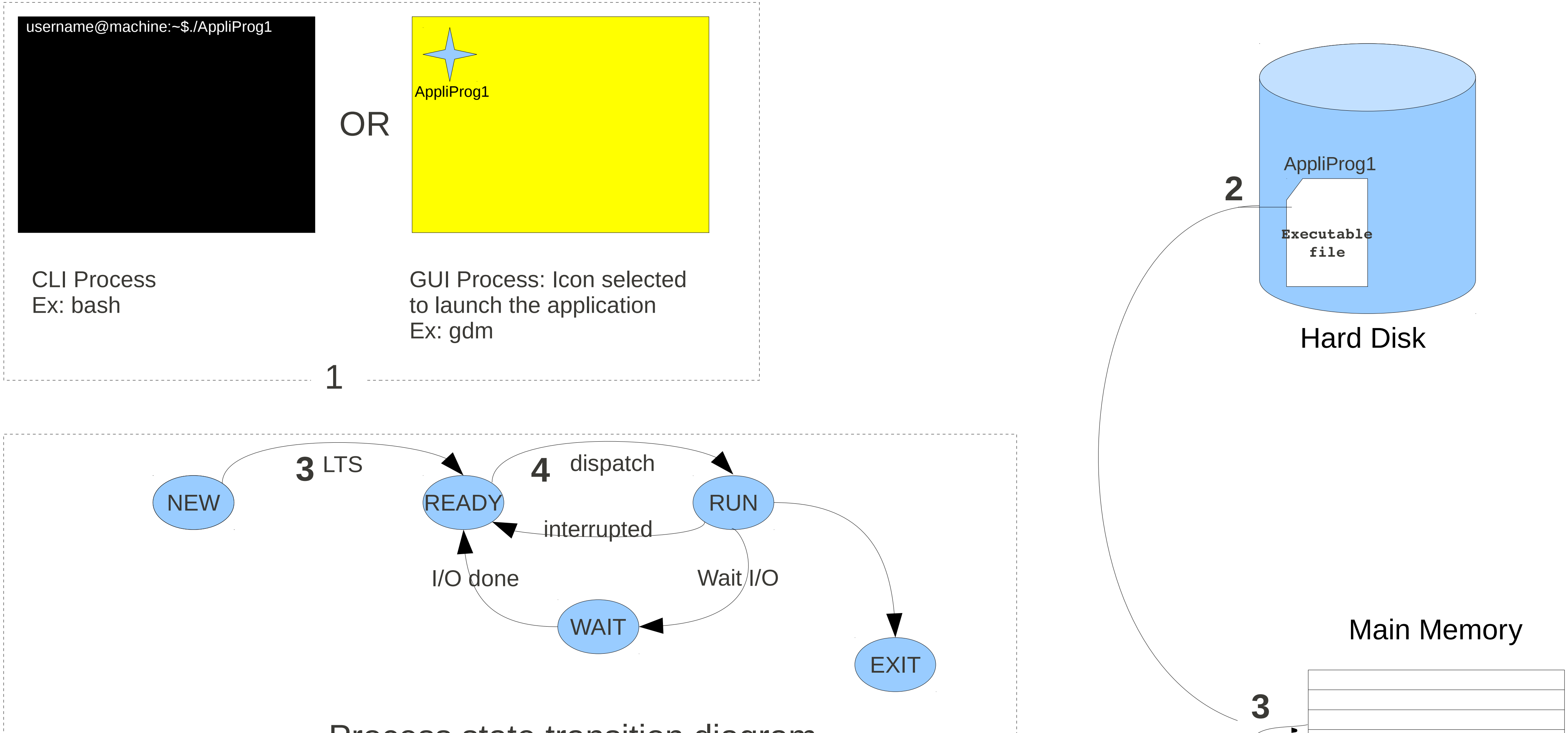
1. User launches “AppliProg1” either by CLI/GUI/...
2. The CLI/GUI process **fork/clones** to execute AppliProg1 as its child
Now this process is in **NEW state.**
3. LTS moves this proc into **Ready state** into Ready-queue



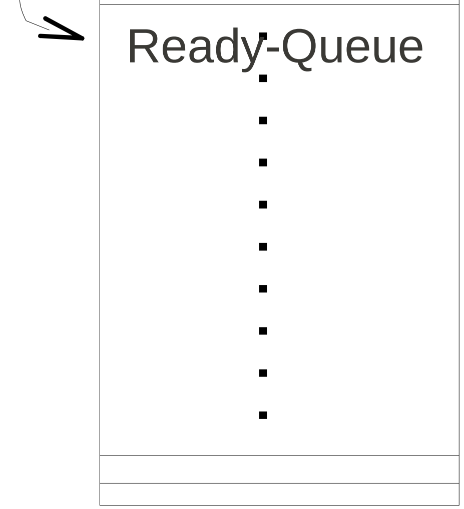




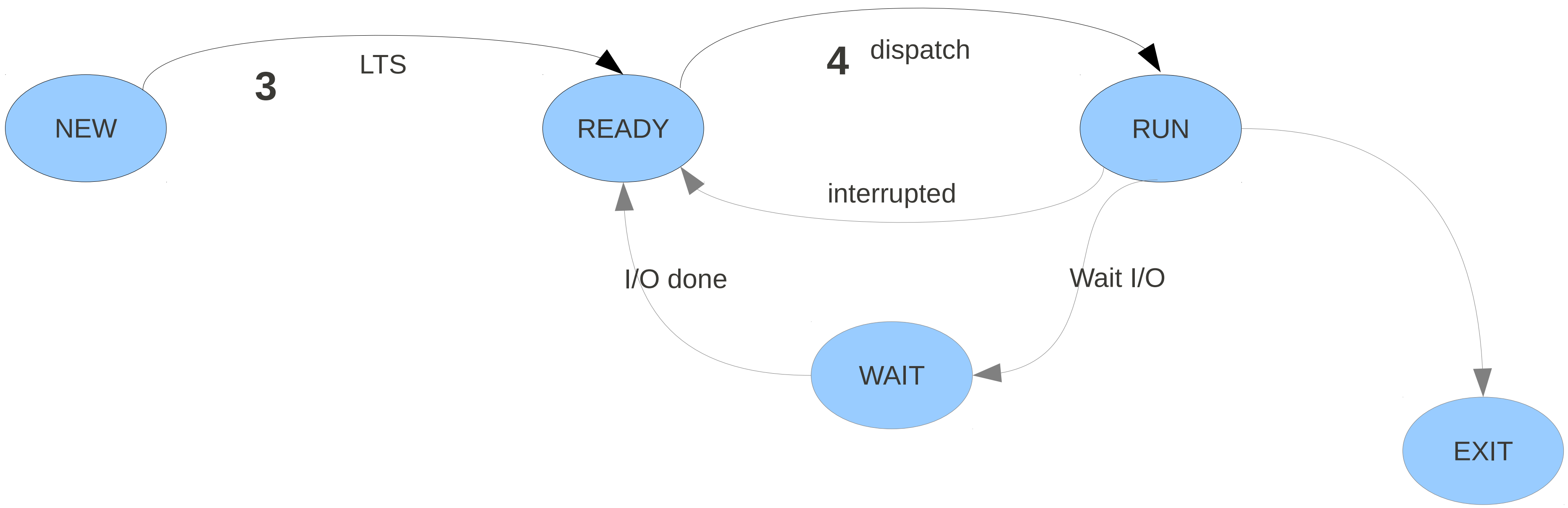
1. User launches “AppliProg1” either by CLI/GUI/...
2. The CLI/GUI process **fork/clones** to execute AppliProg1 as its child
Now this process is in **NEW state.** Here prog loads into memory and process gets its PCB
3. LTS moves into **Ready state** into Ready-queue
4. STS selects this “AppliProg1” for execution marking it as “**Running State**” based on the scheduling policy (ex: FCFS, PBS, SJF, etc.)



Process state transition diagram



1. User launches “AppliProg1” either by CLI/GUI/...
2. The CLI/GUI process **fork/clones** to execute AppliProg1 as its child
Now this process is in **NEW state.** Here prog loads into memory and process gets its PCB
3. LTS moves into **Ready state** into Ready-queue
4. STS selects this “AppliProg1” for execution marking it as “**Running State**” based on the scheduling policy (ex: FCFS, PBS, SJF, etc.)
5. The rest of process state diagram corresponds to I/O waitng and IRQ activities as discussed earlier in class.

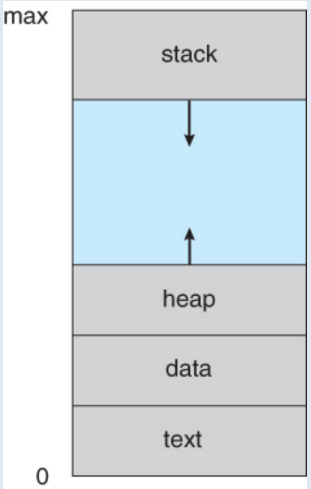


1. User launches “AppliProg1” either by CLI/GUI/...
2. The CLI/GUI process **fork/clones** to execute AppliProg1 as its child
Now this process is in **NEW state.** Here prog loads into memory and process gets its PCB
3. LTS moves into **Ready state** into Ready-queue
4. STS selects this “AppliProg1” for execution marking it as “**Running State**” based on the scheduling policy (ex: FCFS, PBS, SJF, etc.)
5. The rest of process state diagram corresponds to I/O waitng and IRQ activities as discussed earlier in class.

Program Vs. Process MMAP

- Program has
  – text/code
  – data: global/static/constant (init + uninit) and local (to methods)

  – These take different locations in process memory map



Stack: local variables + arguments [rw]

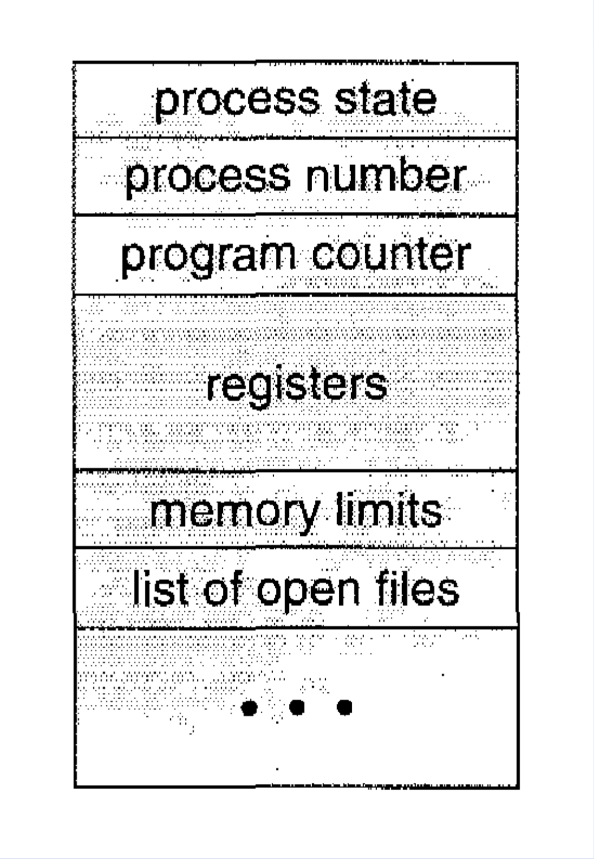
Heap: dynamic variables (get created at R/T) [rw]

Data: init + uninit [rw]

Text: machine code [wo]

Process Control Block

- Process is represented in the OS as Process Control Block (PCB)



Process State: state from PSTD

Process number: ID

Program Counter: PC value

Registers: CPU's SPR, GPR, etc

Memory Limits: base and limit register values

Open files: open files during the course at a PP