R

بسم الله الرحمن الرحیم

سیستم عامل

جلسه سوم – پردازه

Original slides by: Jonathan Walpole & Hadi Foroughmand Arabi – Some slides/images by Silberschatz, Galvin and Gagne

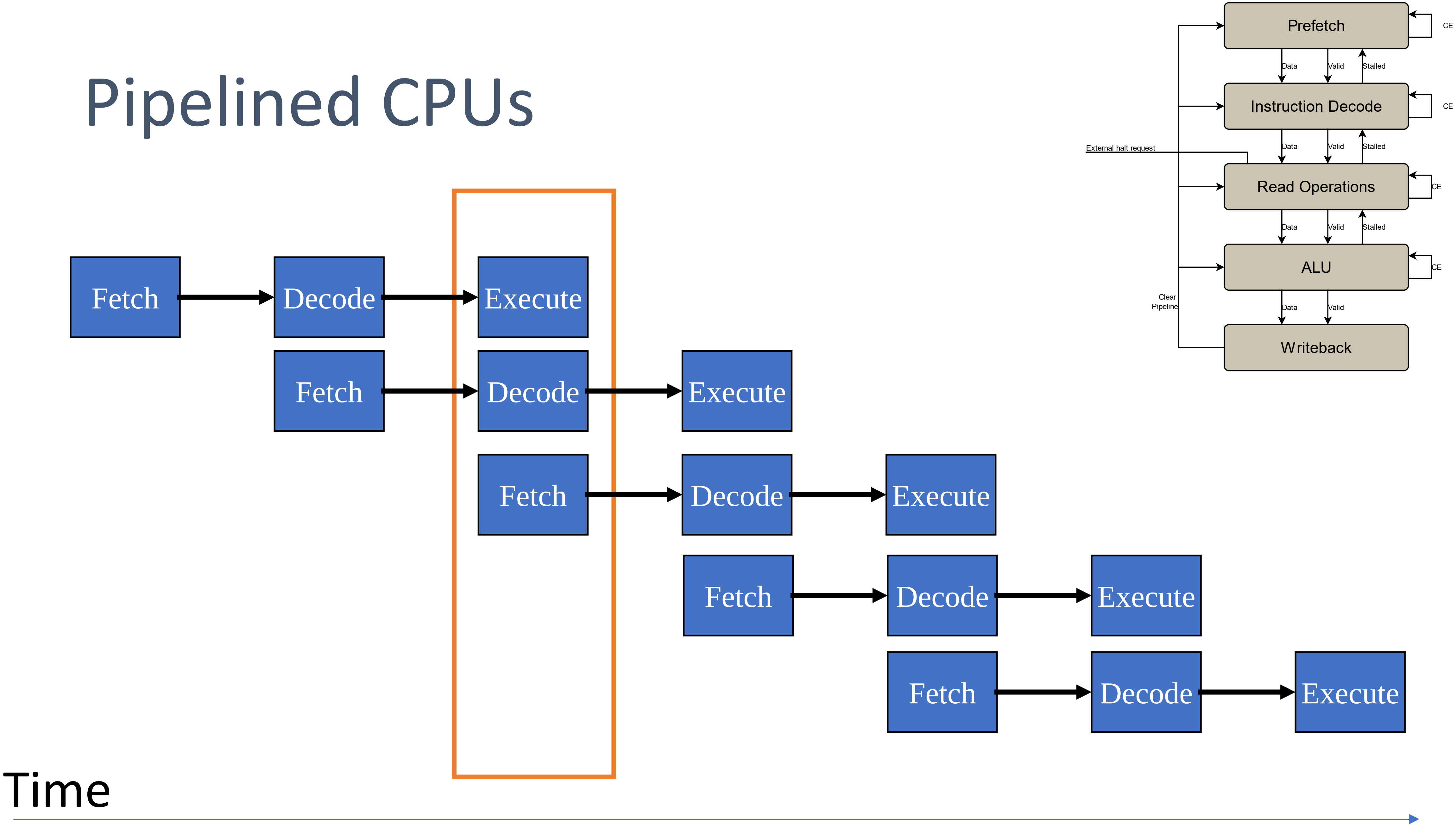
آنچه گذشت

جلسه‌ی قبل، معماری کامپیوتر

پردازنده

پردازنده‌های Pipelined، Superscalar و Multiprocessing

Pipelined CPUs



حافظه

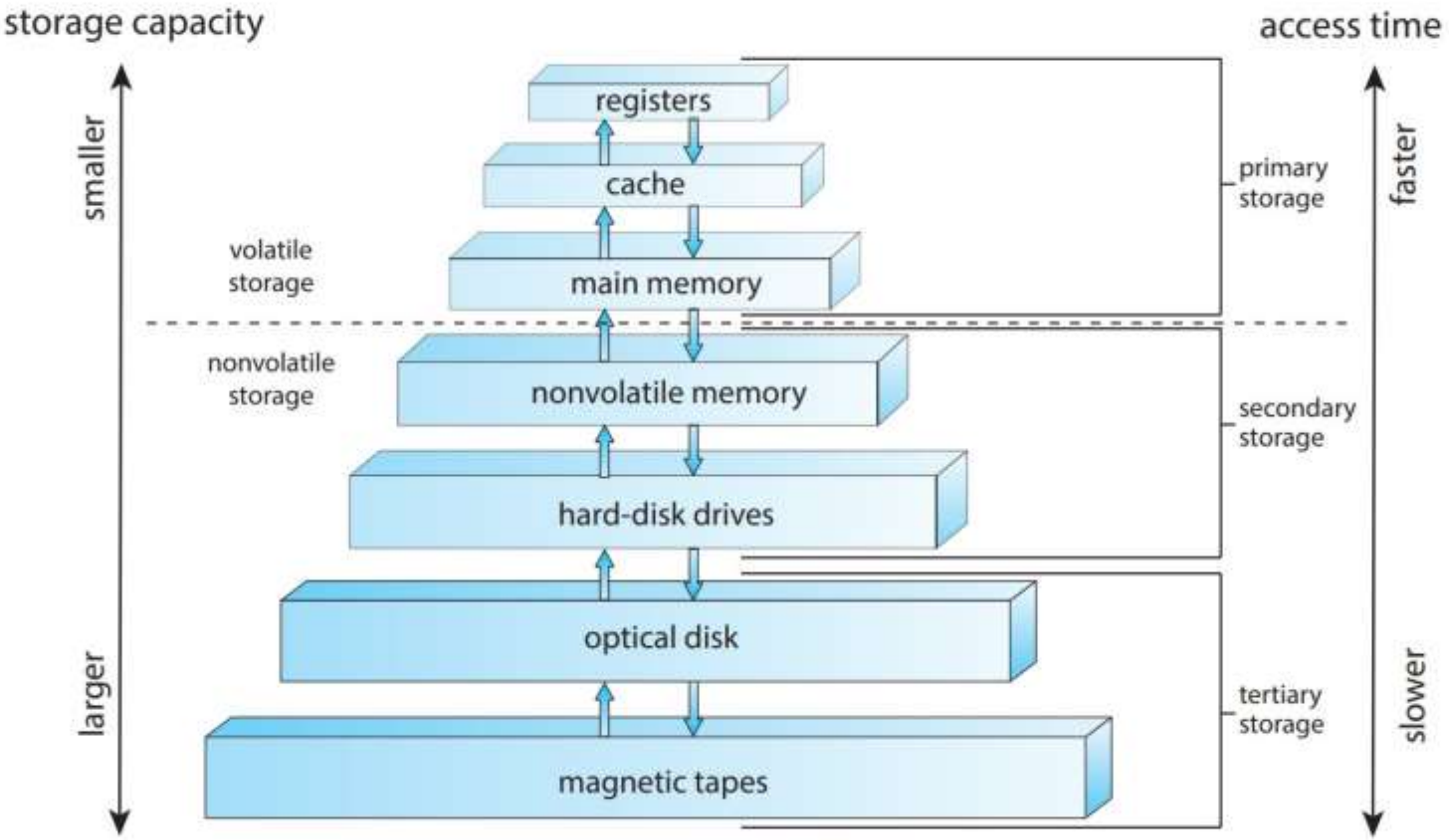


Figure 1.6 Storage-device hierarchy.

Virtual Address

- Base register
- Page Table

دستگاه‌های ورودی و خروجی

Device, Device Controller, Driver

Linkers and Loaders

- Source code compiled into object files designed to be loaded into any physical memory location – **relocatable object file**
- **Linker** combines these into single binary **executable** file
  - *Also brings in libraries*
- Program resides on secondary storage as binary executable
- Must be brought into memory by **loader** to be executed
  - ***Relocation*** *assigns final addresses to program parts and adjusts code and data in program to match those addresses*
- Modern general purpose systems don’t link libraries into executables
  - *Rather,* ***dynamically linked libraries*** *(in Windows,* ***DLLs****) are loaded as needed, shared by all that use the same version of that same library (loaded once)*
- Object, executable files have standard formats, so operating system knows how to load and start them

Questions?

این جلسه

جا مانده از جلسه‌ی قبل

- فایل سیستم
- شبکه

مفهوم پردازه

Definition of “Program”

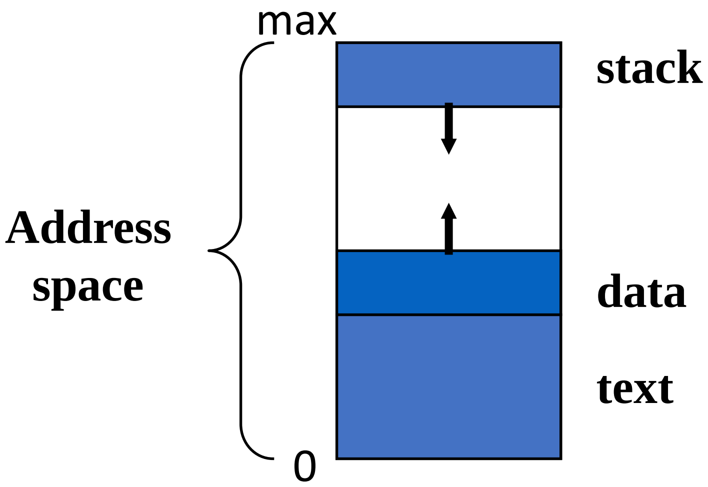
- A **program** is a **set of instructions**
- **passive entity** stored on disk

The Process Concept (Vs. Program)

- Process – a program in execution
- Program
  - description of how to perform an activity
  - instructions and static data values
- Process
  - a snapshot of a program in execution
  - memory (program instructions, static and dynamic data values)
  - CPU state (registers, PC, SP, etc)
  - operating system state (open files, accounting statistics etc)

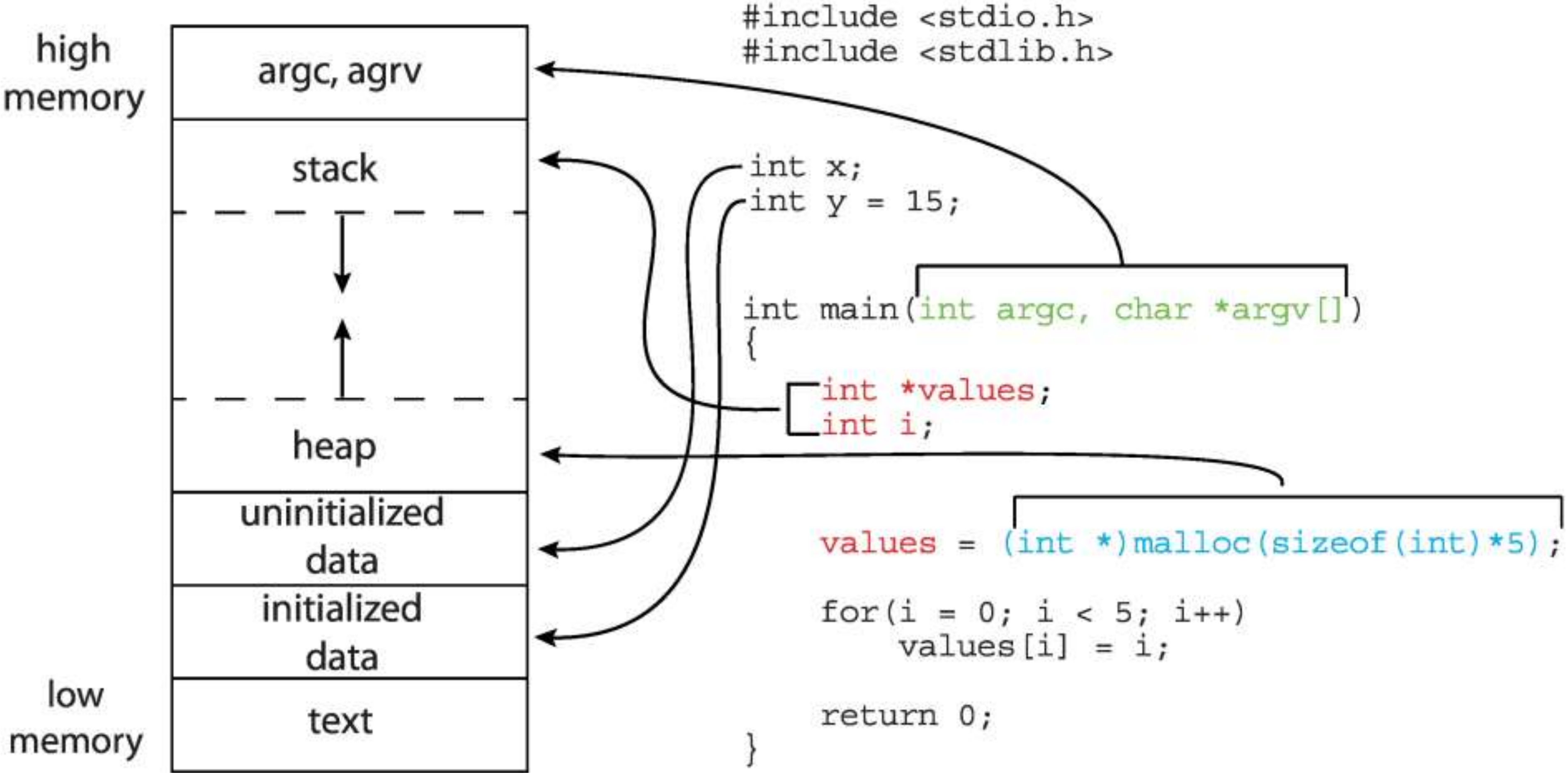
Process Address Space

- Each process runs in its own virtual memory *address space* that consists of:
  - ***Stack space*** *– used for function and system calls*
  - ***Data space*** *– variables (both static and dynamic allocation)*
  - ***Text*** *– the program code (usually read only)*



- Invoking the same program multiple times results in the creation of multiple distinct address spaces

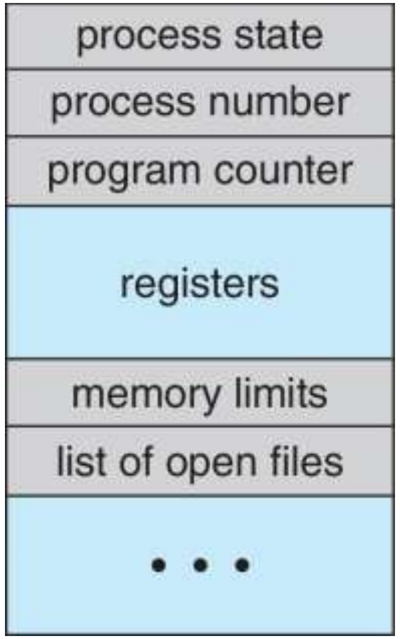
Memory Layout of a C Program



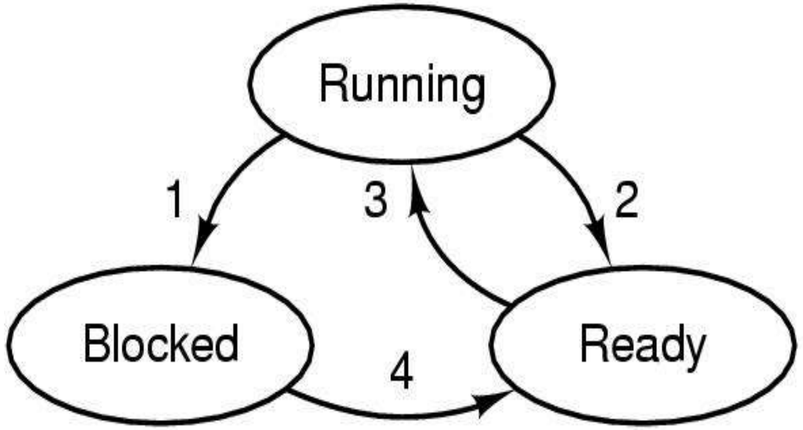
```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```

Process Control Block (PCB)

- **Process state** – running, waiting, etc.
- **Program counter** – location of instruction to next execute
- **CPU registers** – contents of all process-centric registers
- **CPU scheduling information-** priorities, scheduling queue pointers
- **Memory-management information** – memory allocated to the process
- **Accounting information** – CPU used, clock time elapsed since start, time limits
- **I/O status information** – I/O devices allocated to process, list of open files

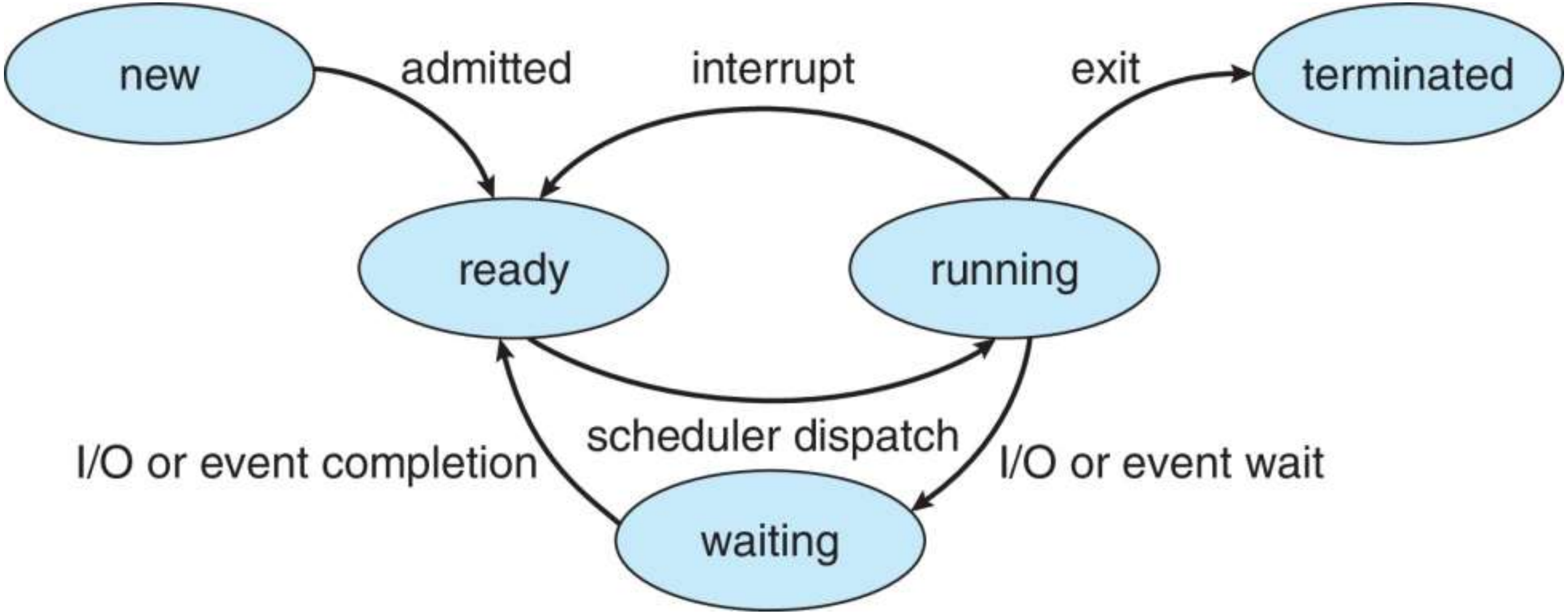


Process States



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Process State in More Detail



Signals

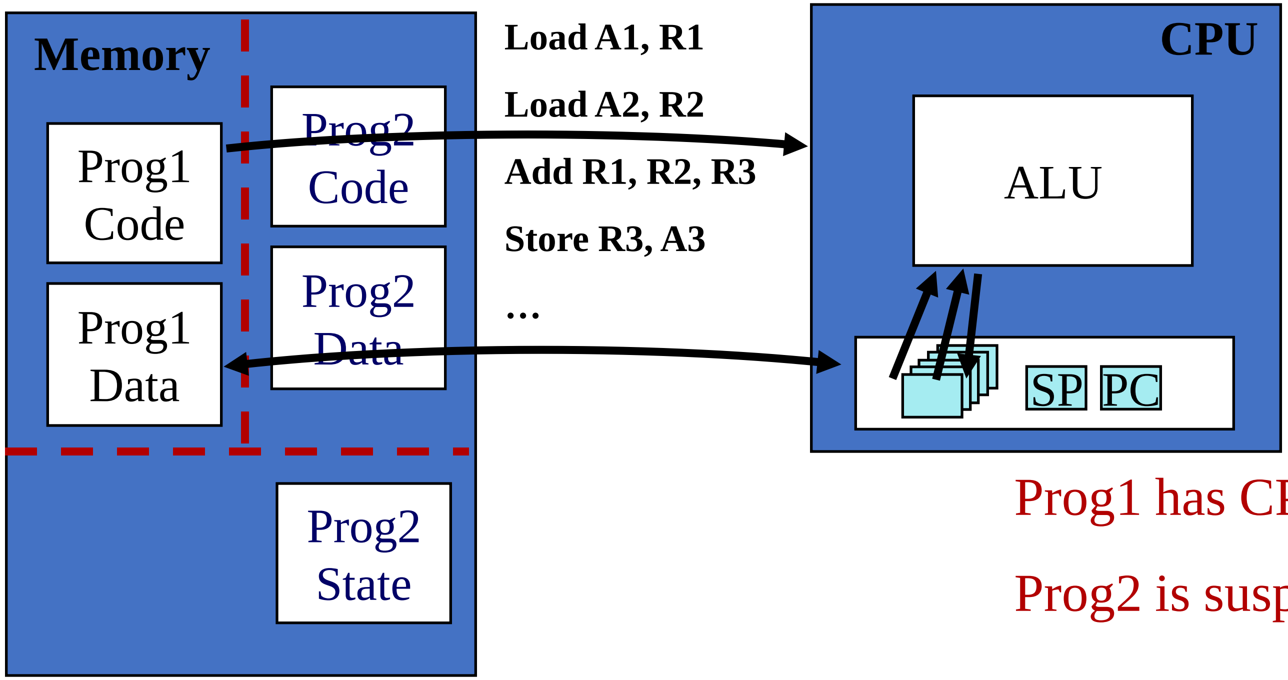
- A signal is an **asynchronous notification** sent to a process to inform it of an event, like a termination request or an interrupt from the user (e.g., pressing Ctrl+C).
- Signals allow the OS or other processes to communicate with and control processes.

CONTEXT SWITCH

جا به جا شدن بین دو پردازه

Switching Among Processes

- Program instructions operate on operands in memory and (temporarily) in registers

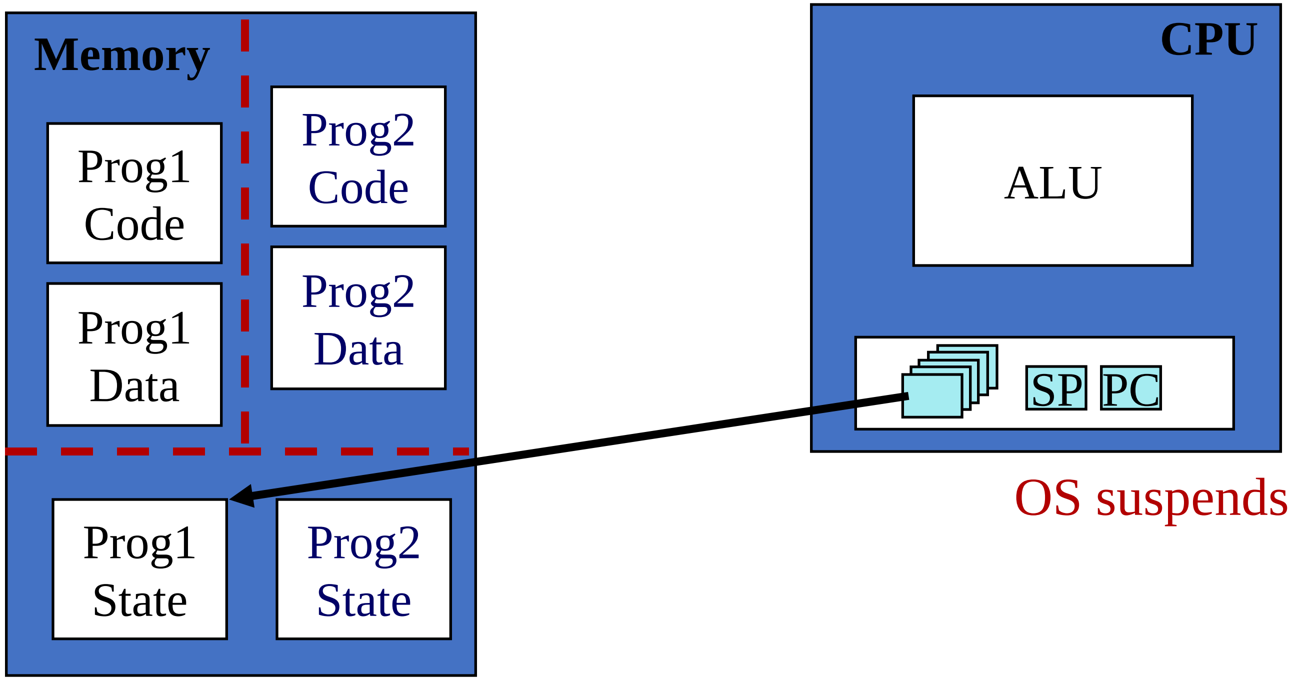


Prog1 has CPU

Prog2 is suspended

Switching Among Processes

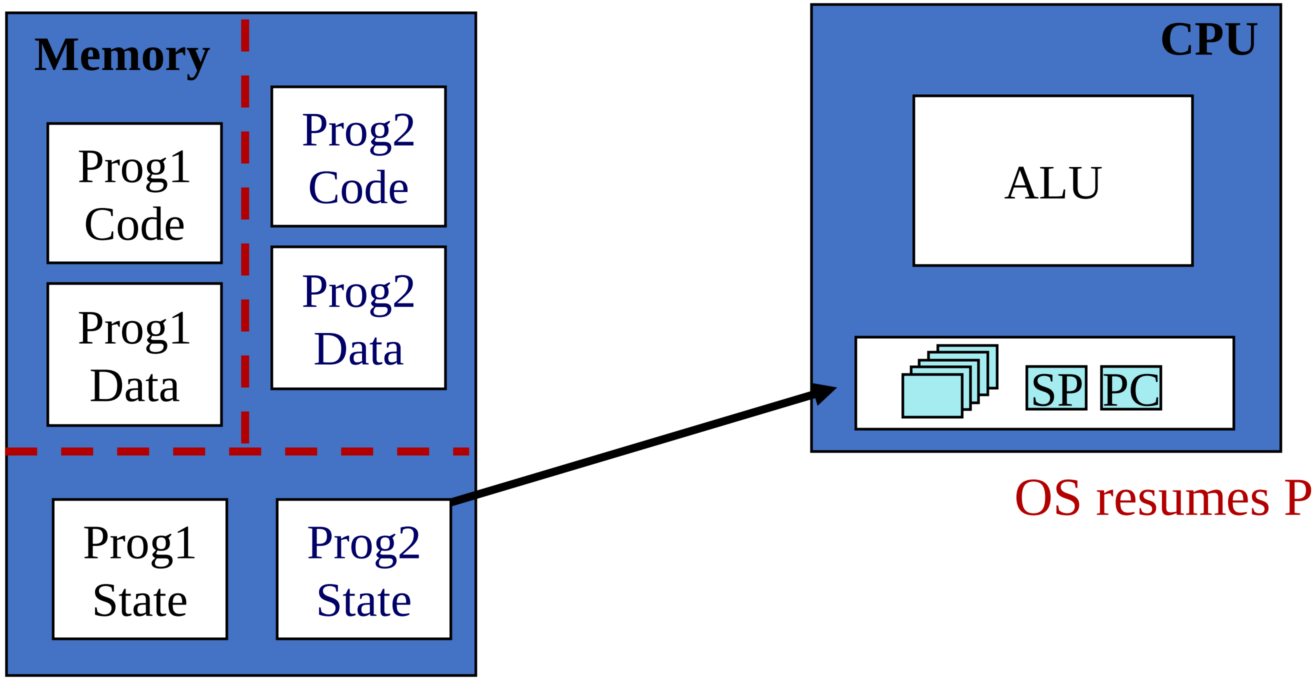
- Saving all the information about a process allows a process to be *temporarily suspended* and later *resumed*



OS suspends Prog1

Switching Among Processes

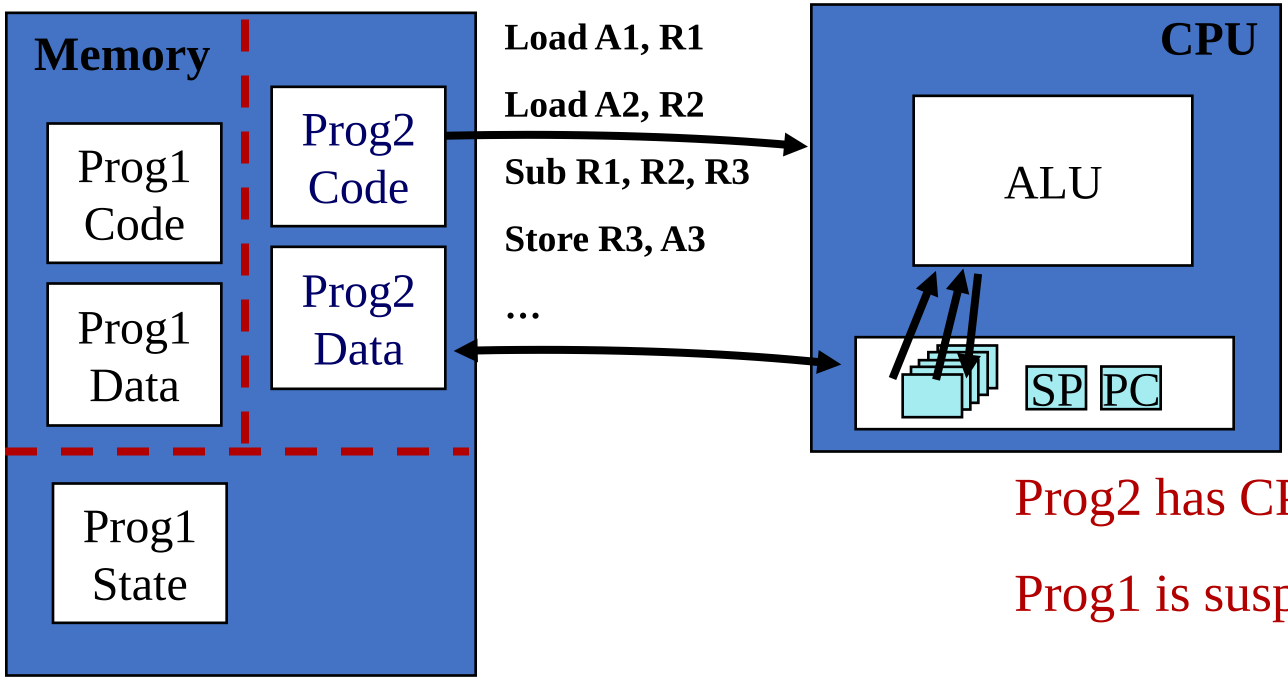
- Saving all the information about a process allows a process to be *temporarily suspended* and later *resumed*



OS resumes Prog2

Switching Among Processes

- Program instructions operate on operands in memory and in registers

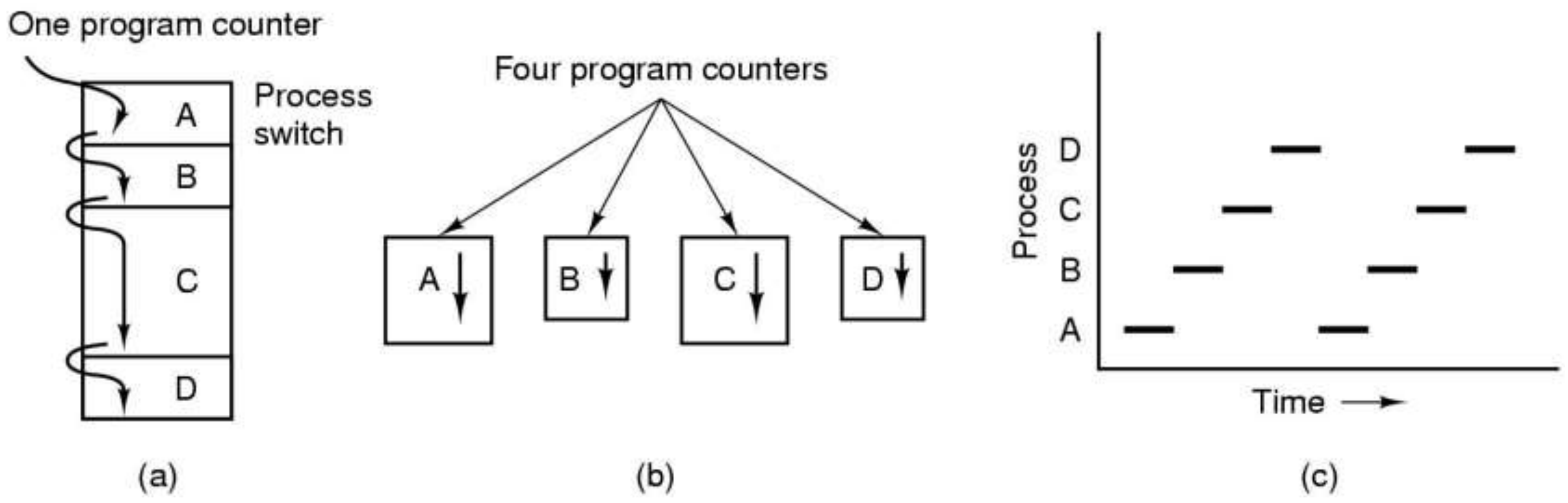


Prog2 has CPU

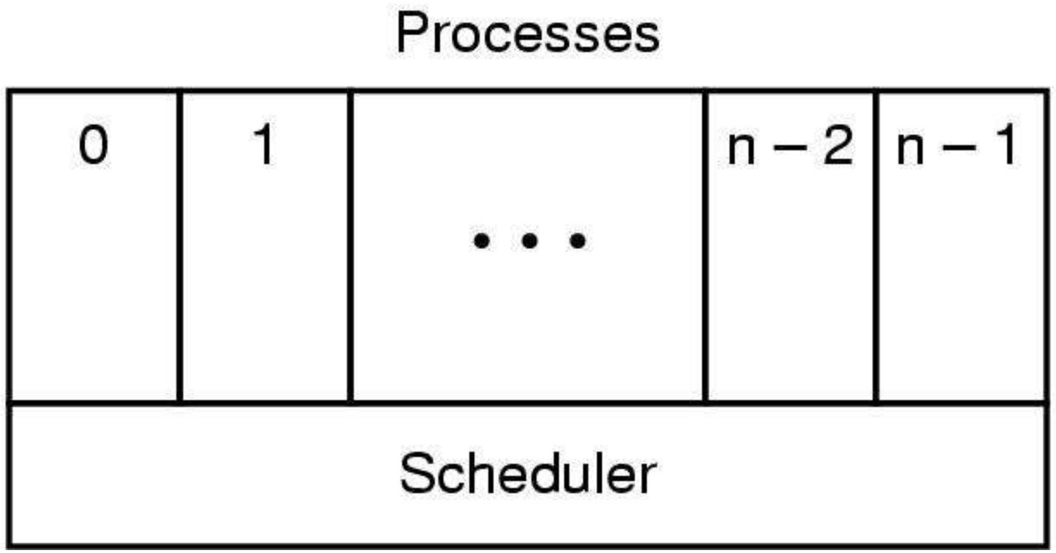
Prog1 is suspended

Why use the process abstraction?

- Multiprogramming of four programs in the same address space
- Conceptual model of 4 independent, sequential processes
- Only one program active at any instant



The Scheduler



- Lowest layer of process-structured OS
  - *handles interrupts & scheduling of processes*
- Sequential processes only exist above that layer

ساختن پردازه‌ی جدید

How do processes get created?

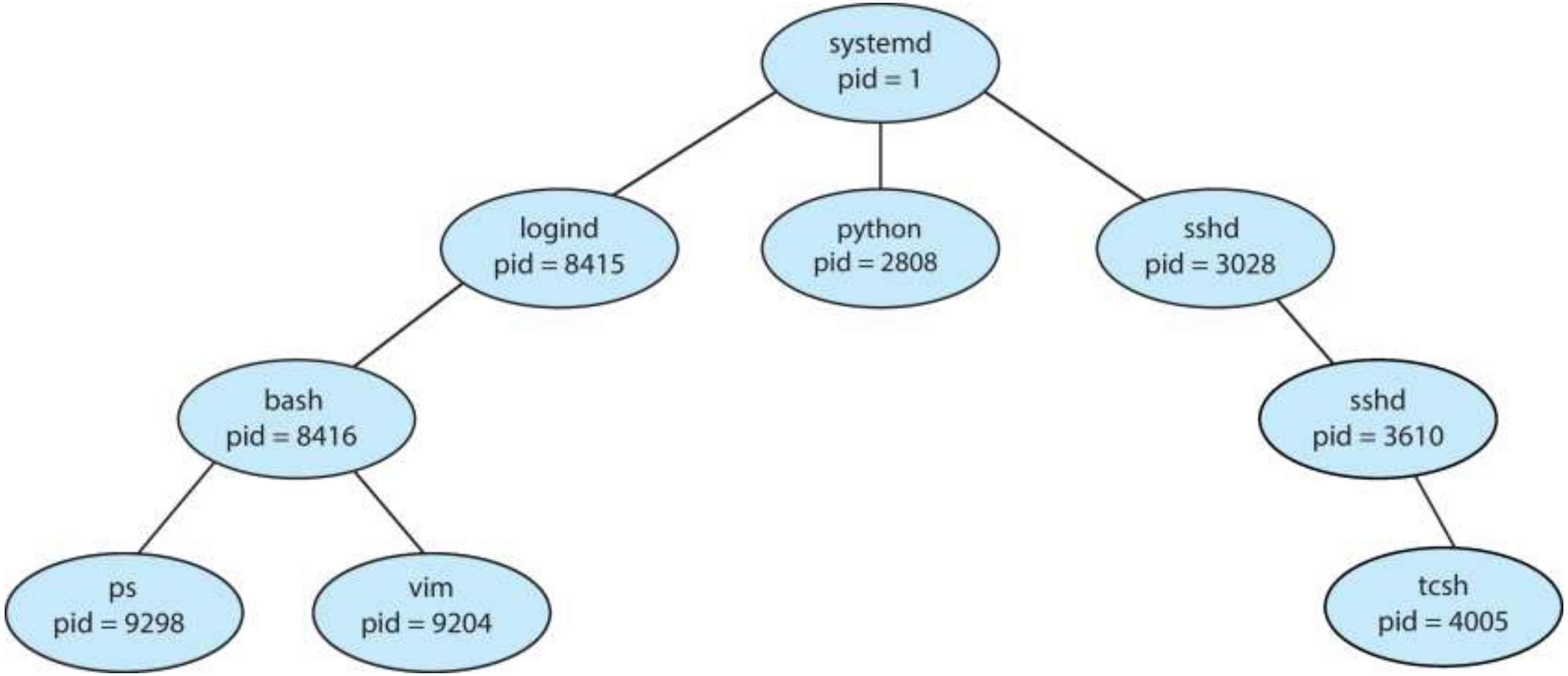
Principal events that cause process creation:

- System initialization
- Initiation of a batch job
- User request to create a new process
- Execution of a process creation system call from another process

Process Hierarchies

- Parent process creates child process
  - *Special system calls for communicating with and waiting for child processes*
  - *each process is assigned a unique identifying number or process ID (PID)*
- Child processes can create their own child processes
  - *Forms a hierarchy*
  - *UNIX calls this hierarchy a "process group“*

A Tree of Processes in Linux



Process Creation in UNIX

- All processes have a unique process id
  - ***getpid()***, ***getppid()*** *system calls allow processes to get their information*
- Process creation
  - ***fork()*** *system call creates a copy of a process and returns in both processes (parent and child), but with a different return value*
  - ***exec()*** *replaces an address space with a new program*
- Process termination, signaling
  - ***signal()***, ***kill()*** *system calls allow a process to be terminated or have specific signals sent to it*

Process Creation Example

csh (pid = 22)

```
…

pid = fork()
if (pid == 0) {
  // child…
  …
exec(“/bin/ls”);
  }
else {
  // parent
  wait();
  }
…
```

Process Creation Example

csh (pid = 22)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

csh (pid = 24)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

Process Creation Example

csh (pid = 22)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

csh (pid = 24)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

Process Creation Example

csh (pid = 22)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

csh (pid = 24)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

Process Creation Example

csh (pid = 22)

```
…

pid = fork()
if (pid == 0) {
   // child…
   …
   exec(“/bin/ls”);
   }
else {
   // parent
   wait();
   }
…
```

ls (pid = 24)

```
//ls program

main(){

   //look up dir

   …

}
```

Process Creation (fork)

- Fork creates a new process by *copying* the calling process
- The new process has its own
  - *Memory address space (copied from parent)*
    - Instructions (same program as parent!)
    - Data
    - Stack
  - *Register set (copied from parent)*
  - *Process table entry in the OS*

تمام شدن یک پردازه

How Do Processes Terminate?

Conditions that terminate processes:

- *Normal exit (voluntary)*
- *Error exit (voluntary)*
- *Fatal error (involuntary)*
- *Killed by another process (involuntary)*

Killing a process

- Sending kill signal to kernel
- Killing a process does not kill its descendants

wait()

- Waits until:
  - *A child is killed/terminated, or*
  - *A signal is received from OS*

Some important signals in Linux

- **SIGINT** (Interrupt): Sent when the user interrupts a process (usually with Ctrl+C).
- **SIGKILL** (Kill): Immediately terminates the process. Cannot be ignored or handled by the process.
- **SIGTERM** (Terminate): Requests the process to gracefully terminate. Can be caught to allow cleanup before exiting.
- **SIGSTOP** (Stop): Stops a process execution. Can be resumed later with **SIGCONT**.

Fork Challenge!

What is the output of the program?

```
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("Parent process started with PID: %d\n", getpid());

    fork();
    fork();
    printf("Process with PID: %d, Parent PID: %d\n", getpid(), getppid());

    return 0;
}
```