Chapter 3: Processes

COMP362 Operating Systems
Prof. AJ Bieszczad

Outline: Processes

- Process Concept
- Process Lifecycle
- Process Control Block (PCB)
- Process Scheduling
- Context Switch
- Process Creation
  - cloning with fork()
  - morphing with exec()
- Processes Termination

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 memory for:
  - text
    - code
  - data
    - static data of the program
  - stack
    - for function calls; fixed size; grows "down"
  - heap
    - for dynamic allocation; OS can resize it as needed
- Some addresses randomized to increase security

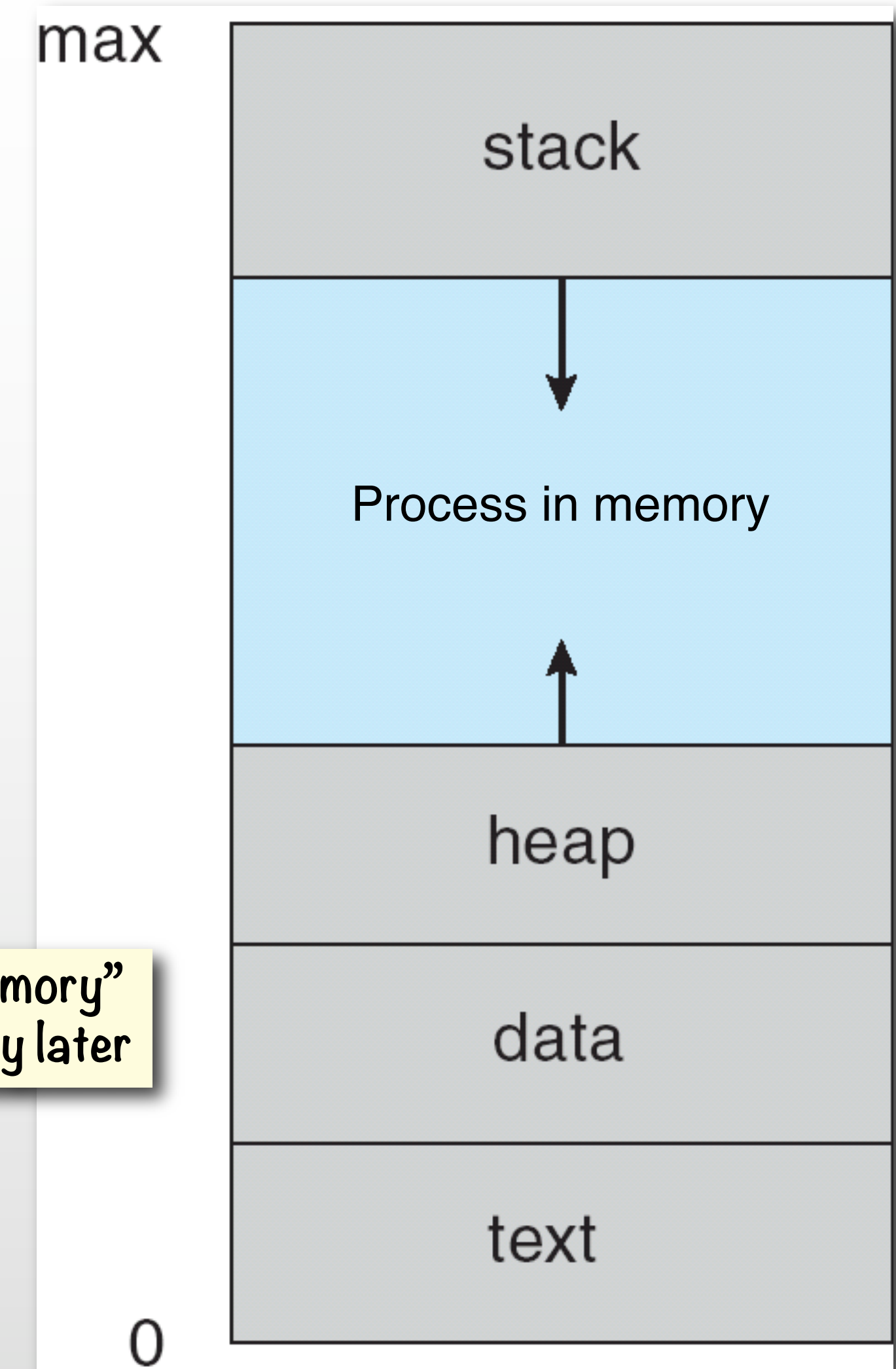- Resources can be controlled by system calls

> Linux provides procfs
> macOS does not

```
$ cat /proc/6514/maps
$ pmap 6514
```

> process ID

> "virtual memory"
> we will study later

```
$ man ulimit
$ man setrlimit
```

Process in memory diagram:
max
- stack
- heap
- data
- text
0

Data Placement in a Process
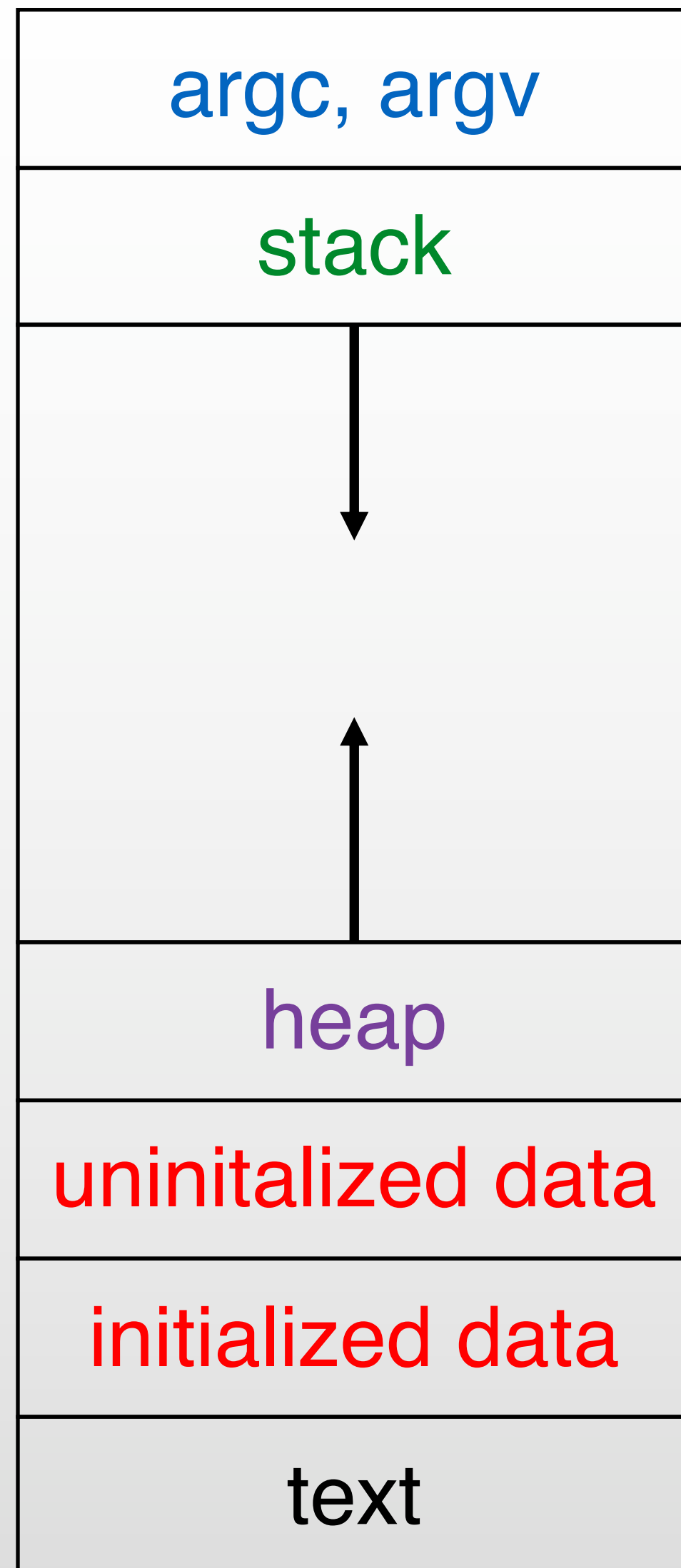
```
$ gcc memory_layout.c -o memory_layout
$ size memory_layout
```

argc, argv
stack
heap
uninitalized data
initialized data
text

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

int error_type;
char *error_msg = "ERROR";

int main(int argc, char **argv)
{
    int *val;


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


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

    return EXIT_SUCCESS;
}
```
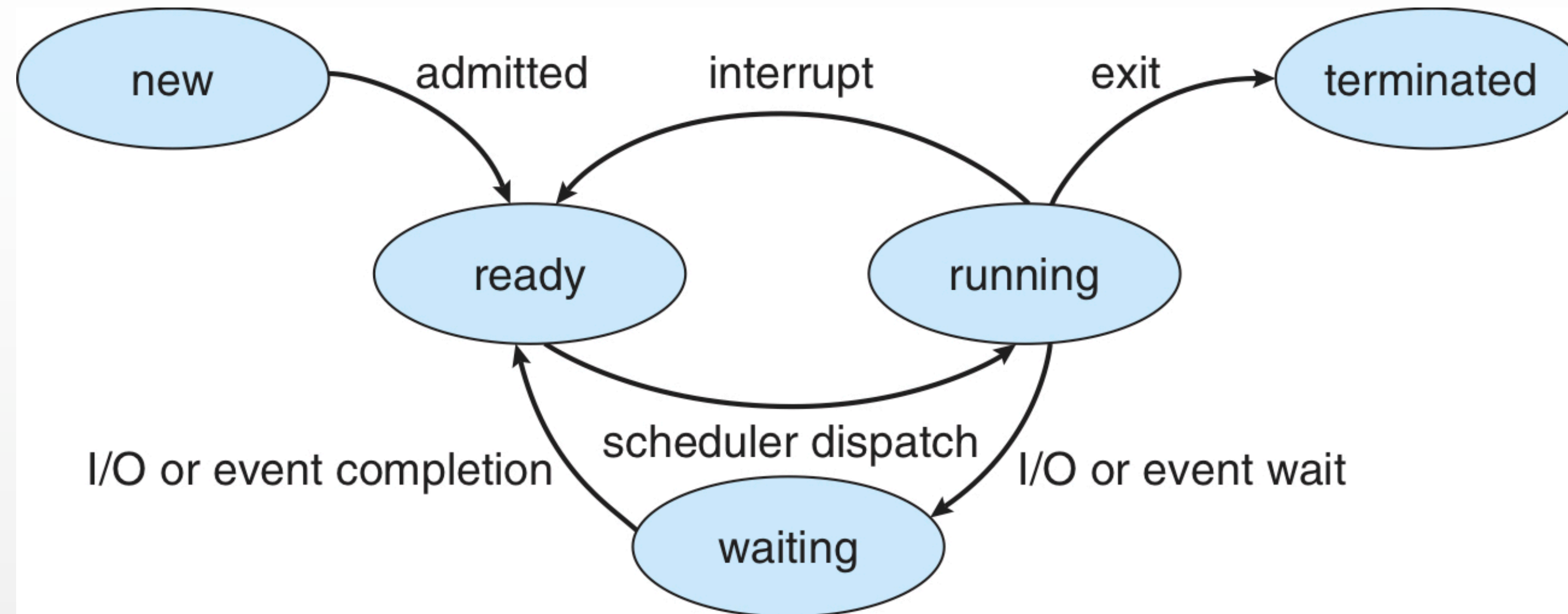
COMP362 Operating Systems, Prof. AJ Bieszczad

Process Lifecycle

- Throughout its life in the system, a process changes states as follows

- **new**: The process is being created
- **ready**: The process is waiting to be assigned to a processor (CPU)
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **terminated**: The process has finished execution

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
- OS maintains a list of PCBs for all processes; e.g.,
 - array
  - linked list
  - hybrid

process state
process number
program counter
registers
memory limits
list of open files
• • •
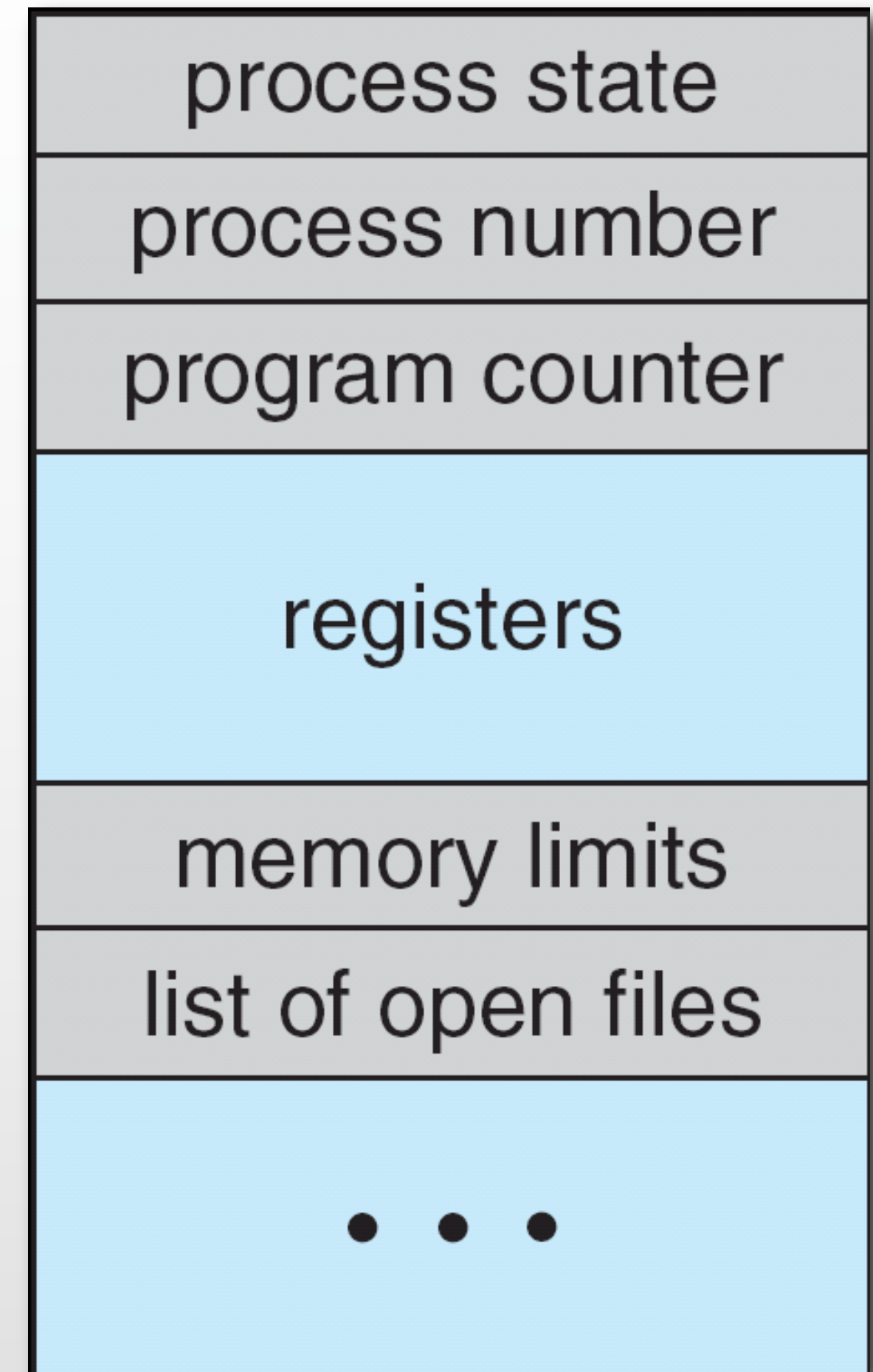
Process Scheduling Queues

- Processes migrate among various OS queues
  - actually: references to PCBs used
- **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
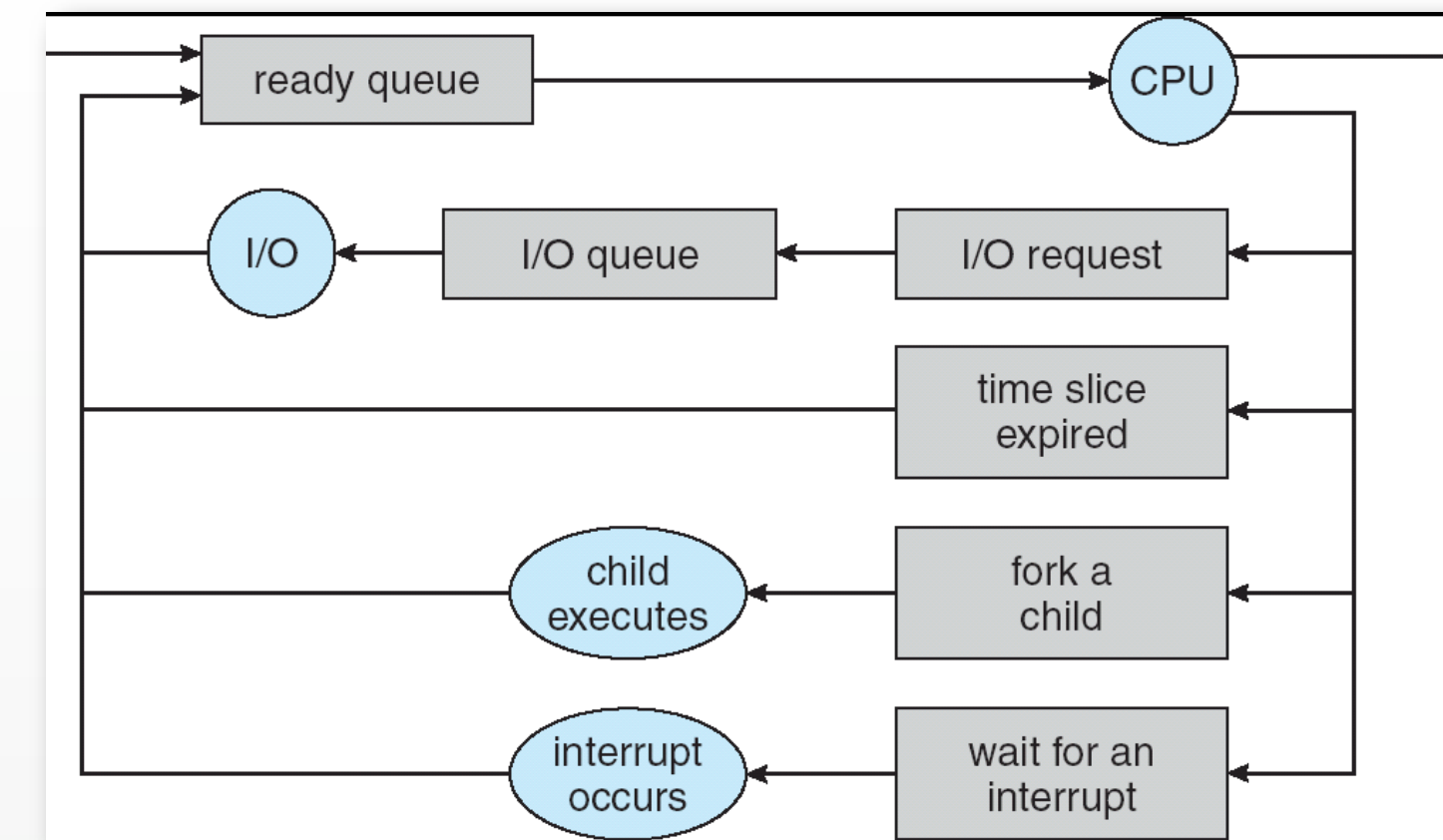
some reasons for suspending a process
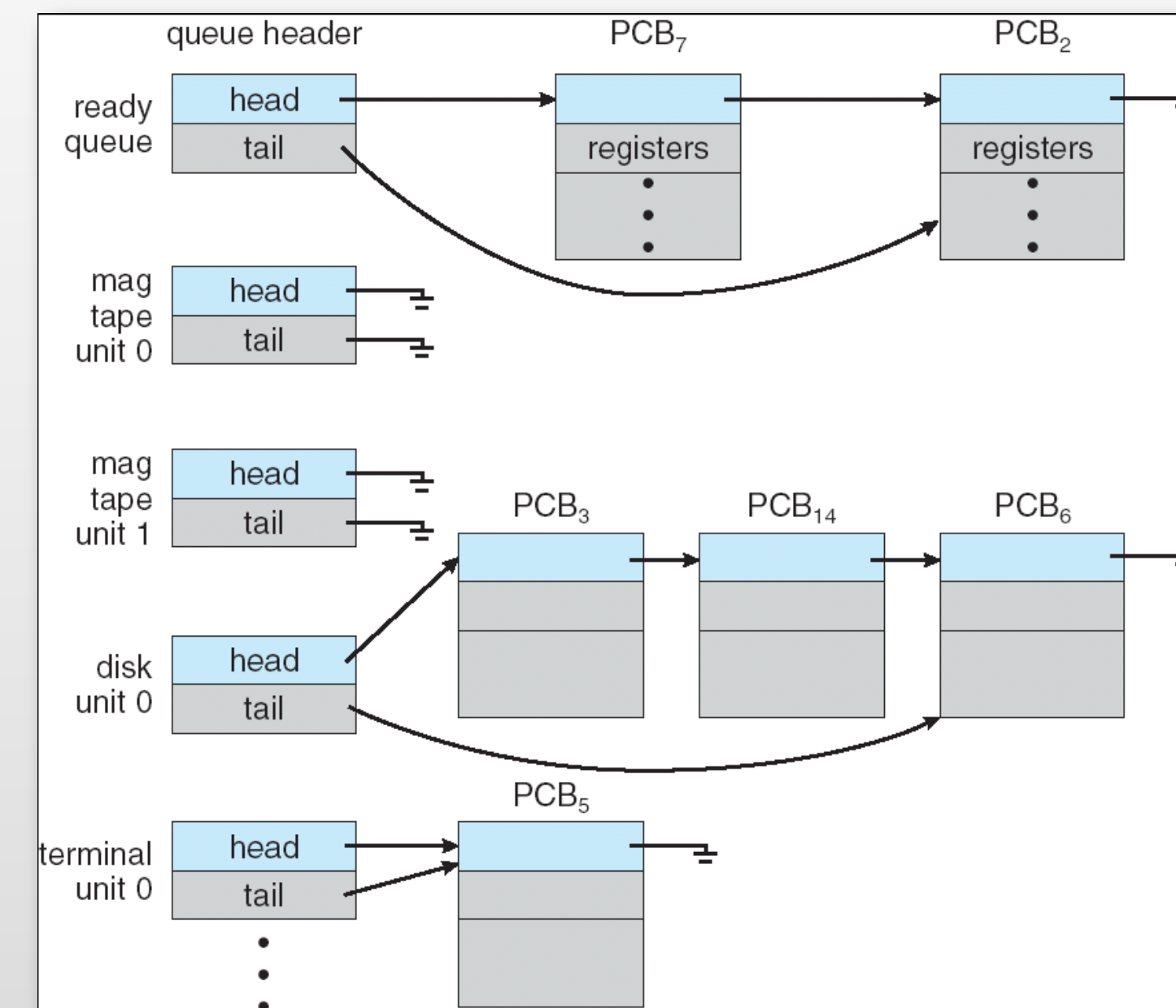

some device queues

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

- **Medium-term scheduler**
  - for swapping processes out and in
    - when inactive, consuming too much memory, etc.

Analogy: taking several courses

- ☑ how many can you take in a semester?
- ☑ which one should you attend now?
- ☑ which one can you put away for a week if you are swamped with work?
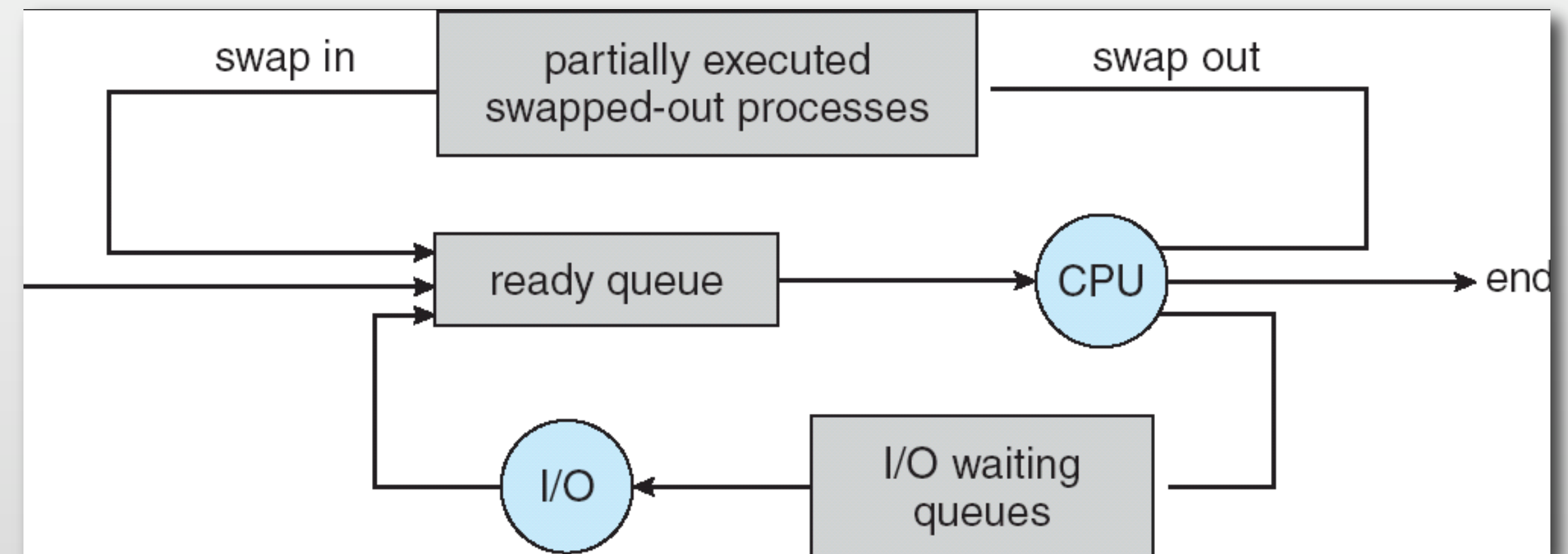
Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds)
  - 👉 must be fast

- Long-term scheduler is invoked very infrequently (seconds, minutes)
  - 👉 does not need to be very fast

- The long-term scheduler controls the degree of **multiprogramming**
  - long-term scheduler must take many things into account when admitting programs; that includes type of the process:
    - **I/O-bound process**
      - spends more time doing I/O than computations, many short CPU bursts
    - **CPU-bound process**
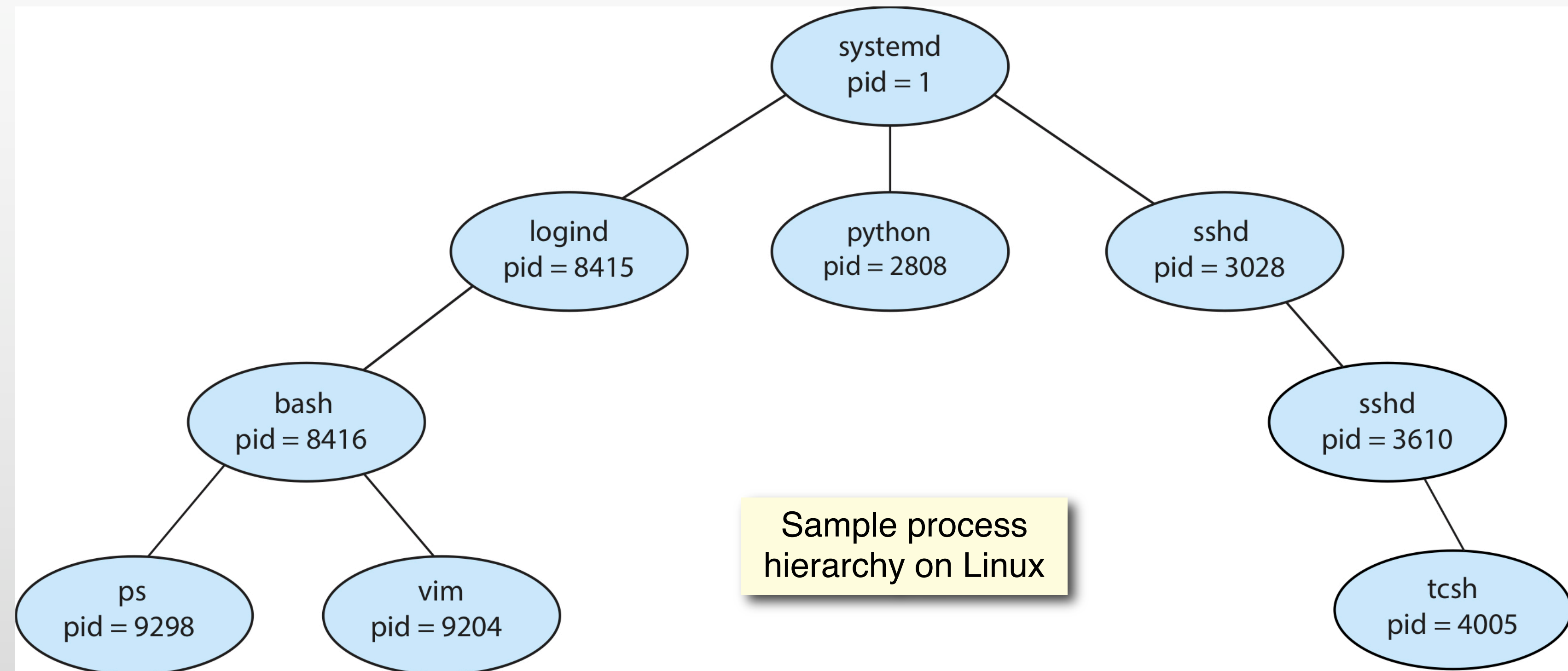   - spends more time doing computations; few very long CPU bursts

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
- **Context-switch** time is an overhead
  - the system is not executing programs
- Time depends on hardware support
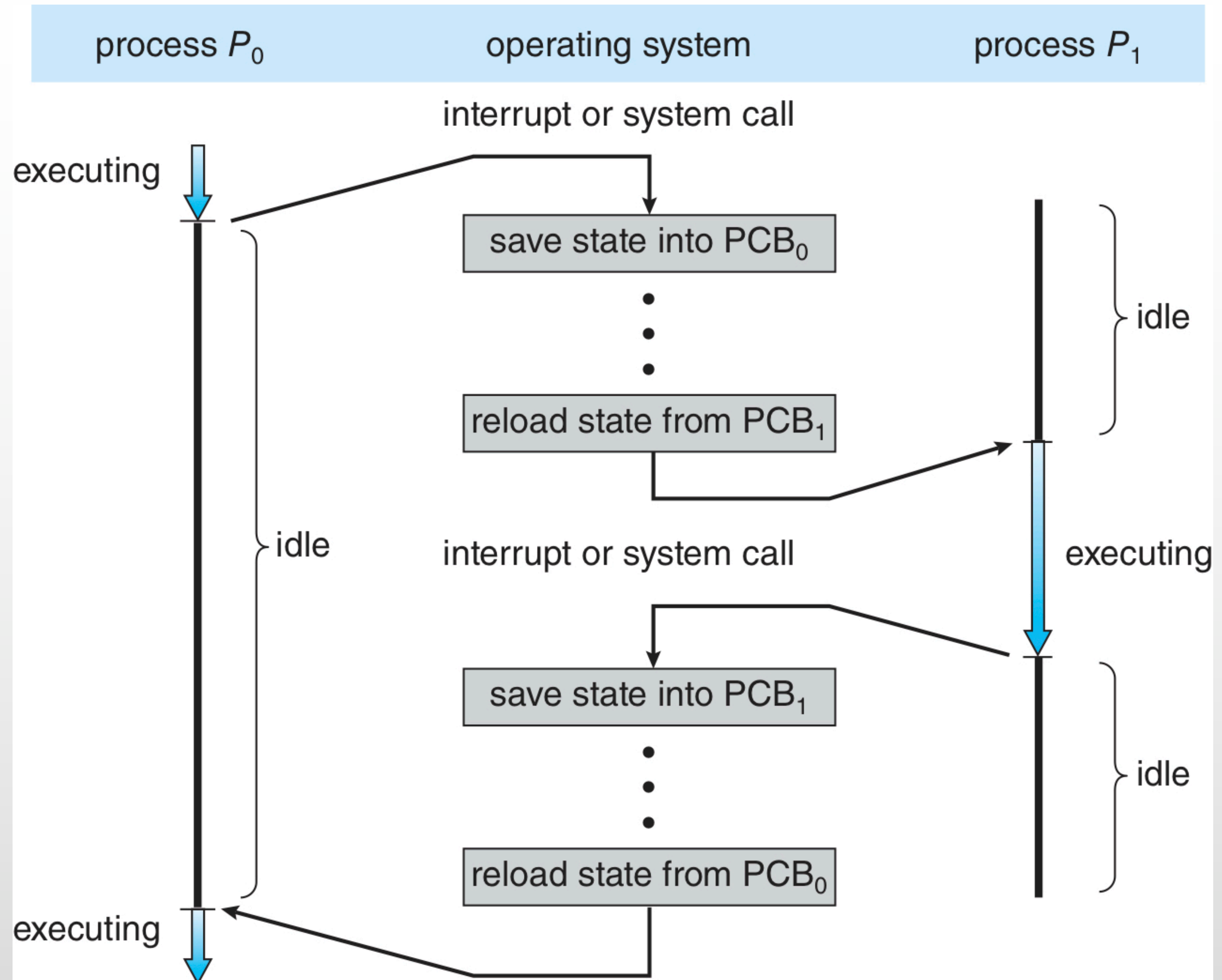
Process Creation

- **Parent process** creates **children processes**, which, in turn create other processes, forming a hierarchy of processes
- Resource sharing options
  - Parent and children may share all resources
  - Children may share subset of parent's resources
  - Parent and children may share no resources
- Execution options
  - Parent and children may execute concurrently
  - Parent may wait until children terminate

systemd
pid = 1

logind
pid = 8415

python
pid = 2808

sshd
pid = 3028

bash
pid = 8416

sshd
pid = 3610

ps
pid = 9298

vim
pid = 9204

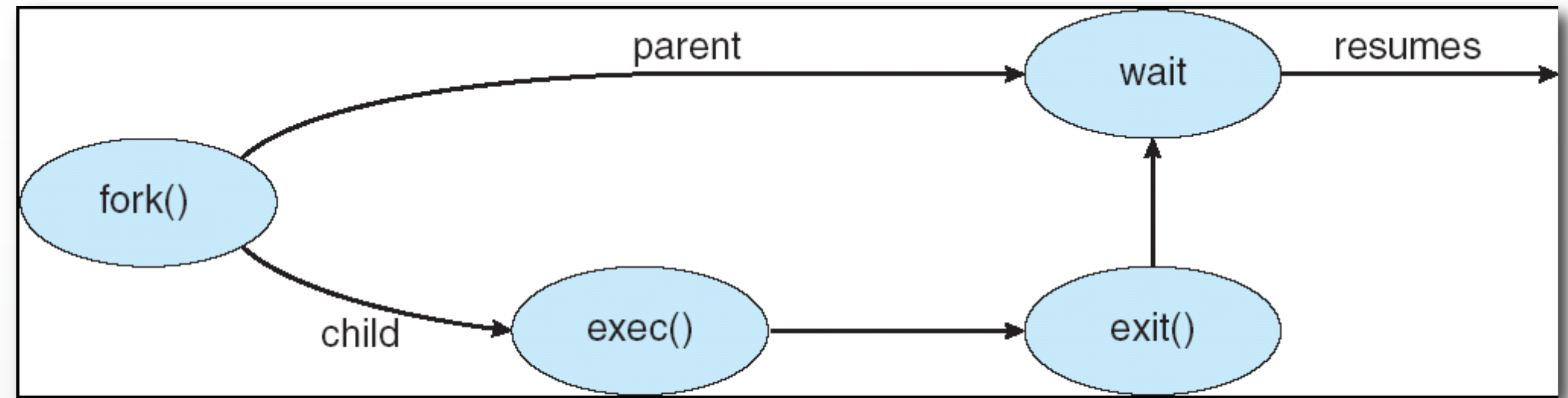Sample process hierarchy on Linux

tcsh
pid = 4005

Process Creation

- Address space choices
  - Child may be a duplicate of the parent
  - Child has another program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call is used after a fork to replace the process' memory space with a new program


```c
/* fork a child process */
pid = fork();
if (pid < 0)
{
    /* error occurred */
}
else if (pid == 0)
{
    /* child process */
}
else /* pid > 0 */
{
    /* parent process */
}
```

`create_process.c`

Process Creation in POSIX

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
    pid_t pid;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed\n");
        exit(EXIT_FAILURE);
    }
    else if (pid == 0) { /* child process */
        printf("I am the child %d\n", getpid());
        execlp("/bin/ls", "ls", NULL); // run /bin/ls in the child
        exit(EXIT_FAILURE); // executed only if execlp() fails
    }
    else { /* parent process; pid is the pid of the child */
        /* parent will wait for the child to complete */
        printf("I am the parent %d\n", getpid());
        waitpid(pid, NULL, 1); // wait for the child with the given pid
        printf("Child Complete\n");
        exit(EXIT_SUCCESS);
    }
}
```

POSIX standard API

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - output data from child to parent (via **wait**)
  - process' resources are **deallocated** by operating system
- Parent may terminate execution of children processes (**abort**)
  - for example, when:
    - child has exceeded allocated resources
    - task assigned to child is no longer required
    - if parent is exiting
      - some operating system do not allow a child to continue if its parent terminates
        - **cascading termination -** if all children along with their children, and so on, are being terminated
- Terminated process may enter so-called a "zombie" state
  - while the operating system is performing termination-related cleanup
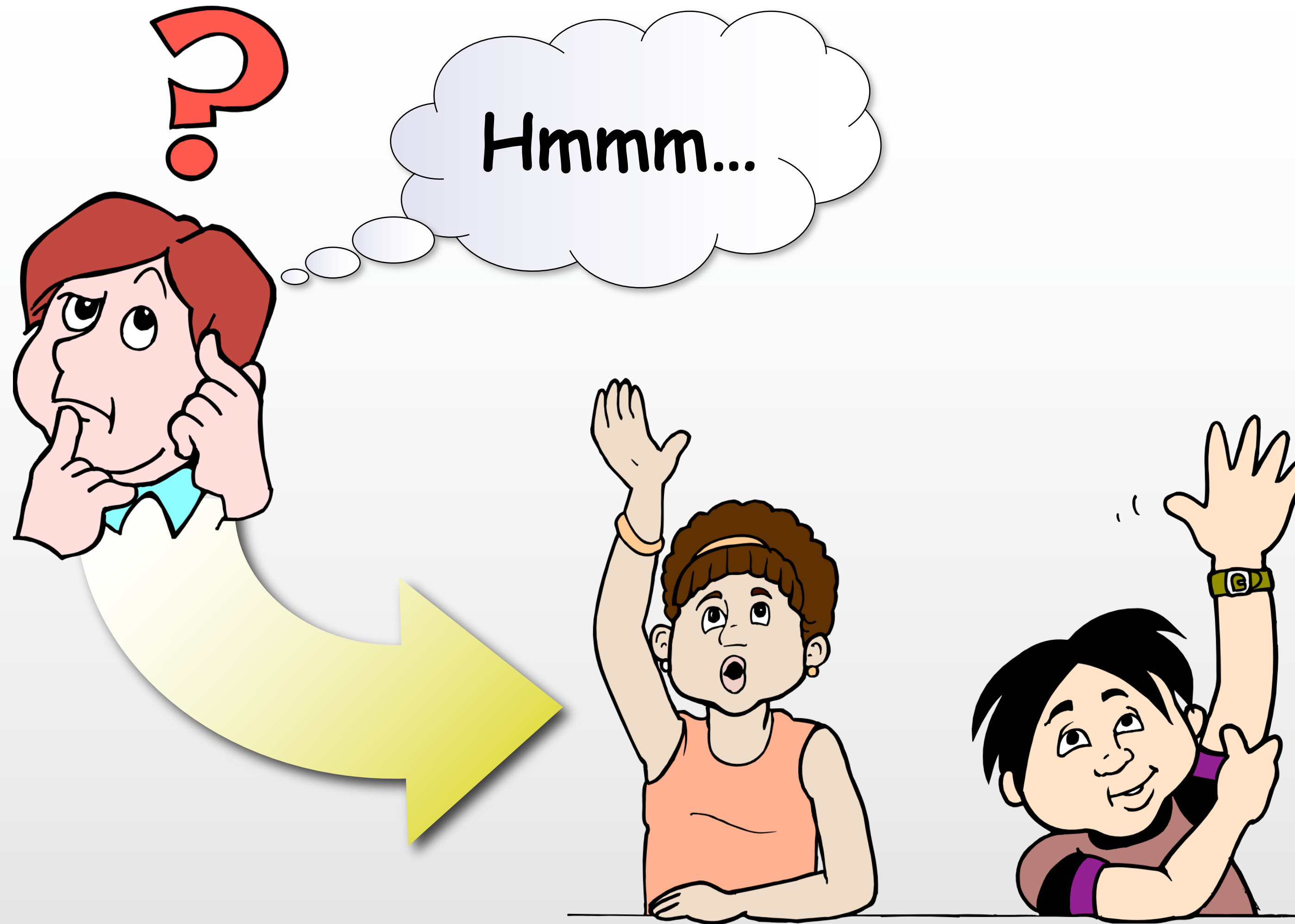
Process Management

```
$ man ps
<exit by pressing Q>
$
$ ps -elf | grep iam
0 R aj          9622  9621 31  80   0 -   613 -      14:28 pts/0   00:00:06 iam 1.1
0 R aj          9623  9621 31  80   0 -   613 -      14:28 pts/0   00:00:06 iam 1.2
0 R aj          9624  9622 30  80   0 -   613 -      14:28 pts/0   00:00:06 iam 1.1.1
0 R aj          9625  9623 30  80   0 -   613 -      14:28 pts/0   00:00:06 iam 1.2.1
0 R aj          9626  9622 31  80   0 -   613 -      14:28 pts/0   00:00:06 iam 1.1.2
0 R aj          9627  9623 30  80   0 -   613 -      14:28 pts/0   00:00:06 iam 1.2.2
0 S aj          9697  9648  0  80   0 -  4370 pipe_w 14:29 pts/3   00:00:00 grep --color=auto iam
$
$ ps -elf | grep proc_
0 S aj          9621  2447  0  80   0 -   613 do_wai 14:28 pts/0   00:00:00 ./proc_hier
0 S aj          9721  9648  0  80   0 -  4403 pipe_w 14:29 pts/3   00:00:00 grep --color=auto proc_
$ kill -KILL 9621
$
$ man top
$ top
<exit by pressing Q>
$
$ sudo apt install htop
$ man htop
$ htop
<exit by pressing Q>
$
$ man procfs
$ cat /proc/12345/maps
$ cat /proc/12345/sched
```

htop is a better version of top

accessing process information using process pseudo file system (content of **/proc**) — only on Linux!

COMP362 Operating Systems
Prof. AJ Bieszczad