REVIEW OF COMMONLY USED DATA STRUCTURES IN OS

NEEDS FOR EFFICIENT DATA STRUCTURE

Storage complexity & Computation complexity matter

Consider the problem of scheduling tasks according to their priority on CPU

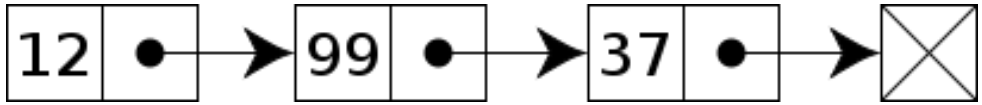
- **New tasks upon arrival, need to be inserted to the queue of pending tasks**
- **Every 100ms, find the task of the highest priority**
- **May need to update the priority of existing tasks at run time**
- **…**

LINKED LISTS

Used in process management, CPU scheduling, file systems

A sequence of nodes consisting of <data, pointer(s)>

Singly linked list




```
struct list_el {
   int val;
   struct list_el * next;
};
```

Singly linked list in C

```
public class LinkedList<E>
```

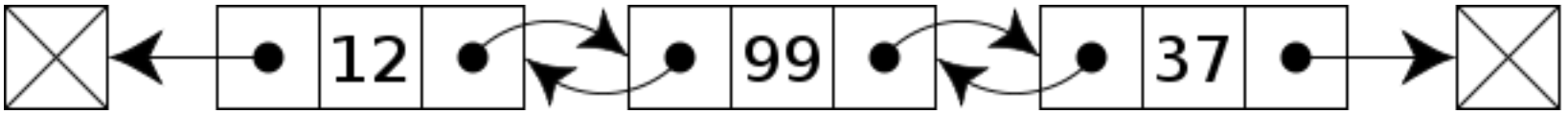
Linked list in Java

LINKED LISTS (CONT'D)

Consider a linked list of size n

- **Storage space?**
- **Cost of inserting a new element at a position x?**
- **Cost of deleting an existing element?**
- **Forward traversal?**
- **Reverse traversal?**

DOUBLY LINKED LIST




```
struct list_el {
   void *val;
   struct list_el * prev;
   struct list_el * next;
};
```

Singly linked list in C

```
public class LinkedList<E>
```

Linked list in Java

DOUBLY-LINKED LISTS (CONT'D)

Consider a doubly-linked list of size n

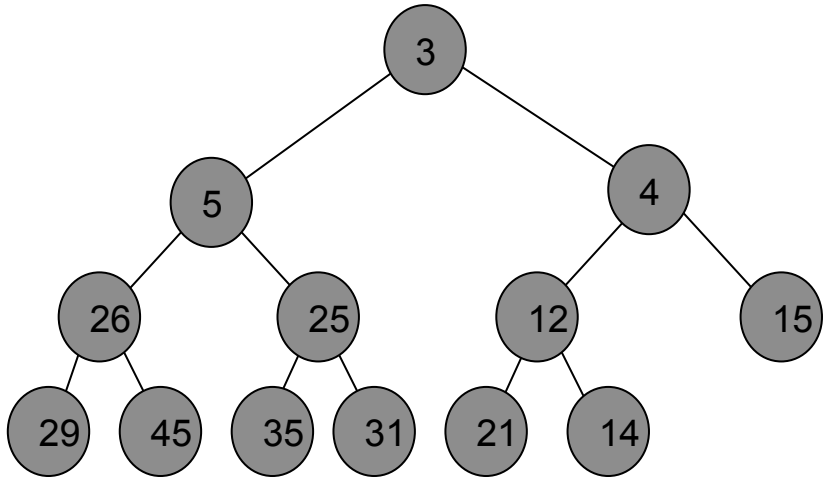
- **Storage space?**
- **Cost of inserting a new element at a position x?**
- **Cost of deleting an existing element?**
- **Forward traversal?**
- **Reverse traversal?**

(MIN-)HEAP

Useful in CPU scheduling (e.g., priority queues)

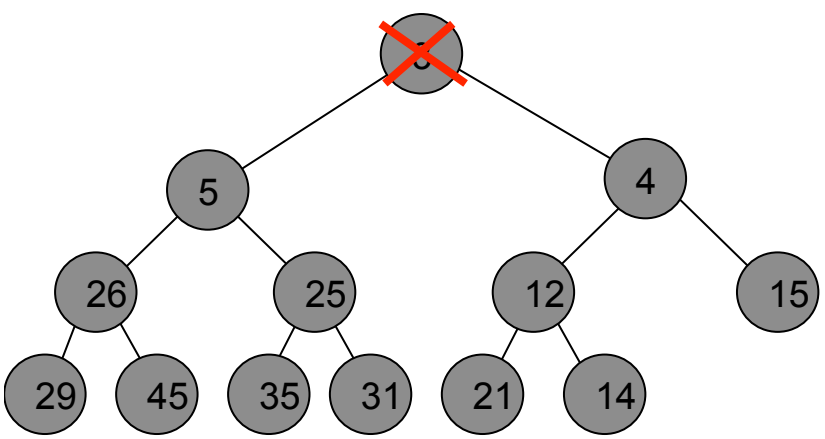
The key of parents always no greater than their children

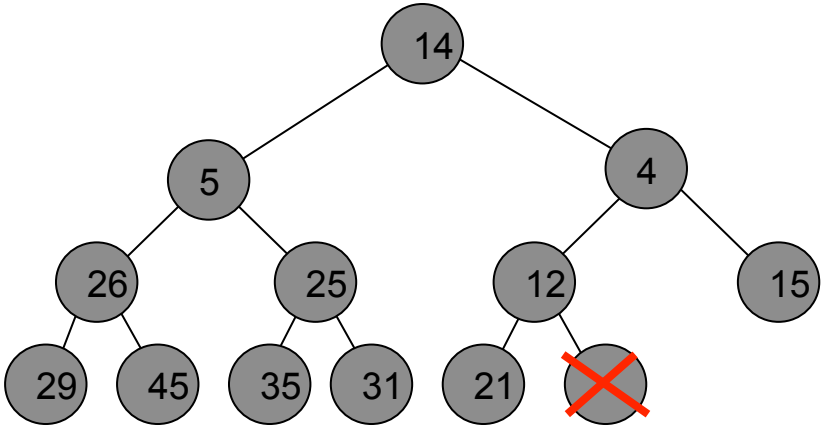
- **Storage space?**
- **Find-min**
- **Insertion**
- **Delete-min**



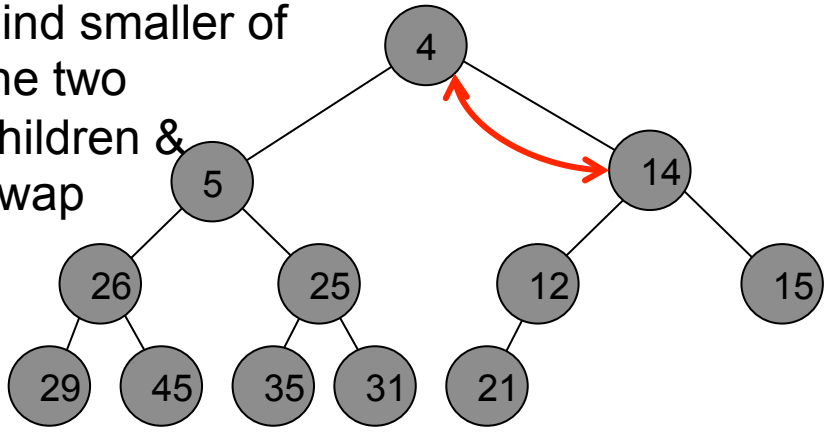
Binary minimum heap

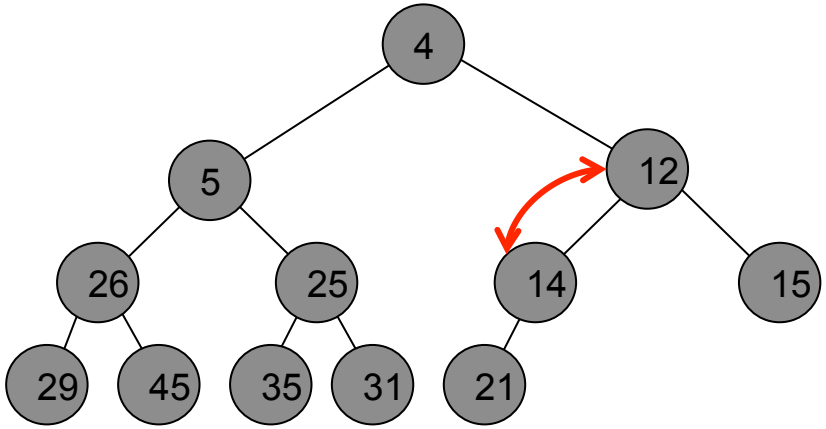
(MAX-)HEAP (CONT'D) – DELETE MIN





Find smaller of the two children & swap

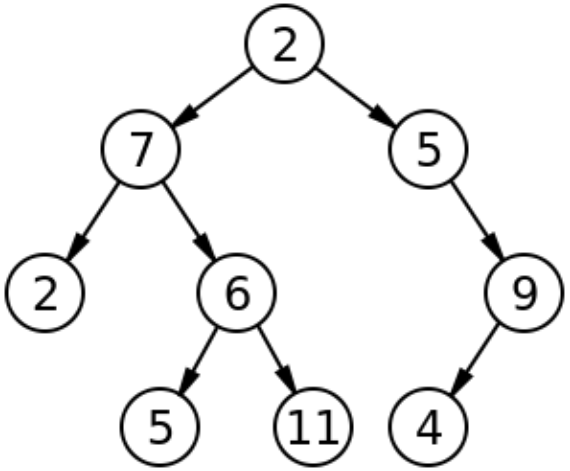




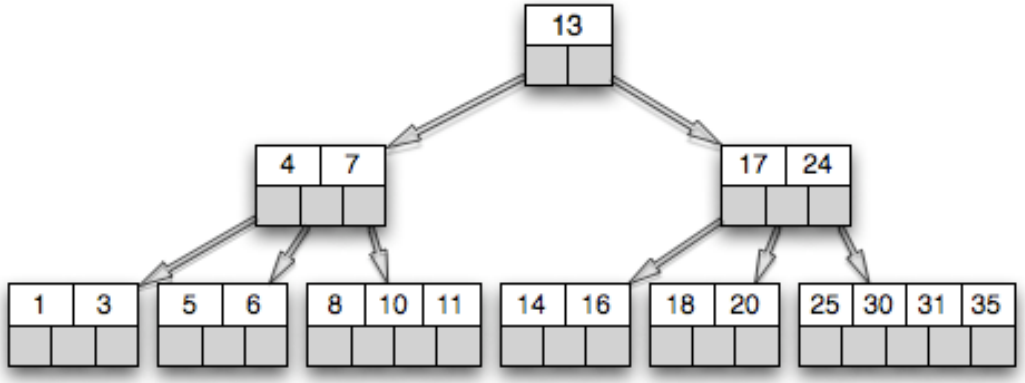
B-TREE

Used mostly in storage & file systems

internal (non-leaf) nodes can be have a variable # of child nodes in some pre-defined range



Binary tree



B-tree of order 5

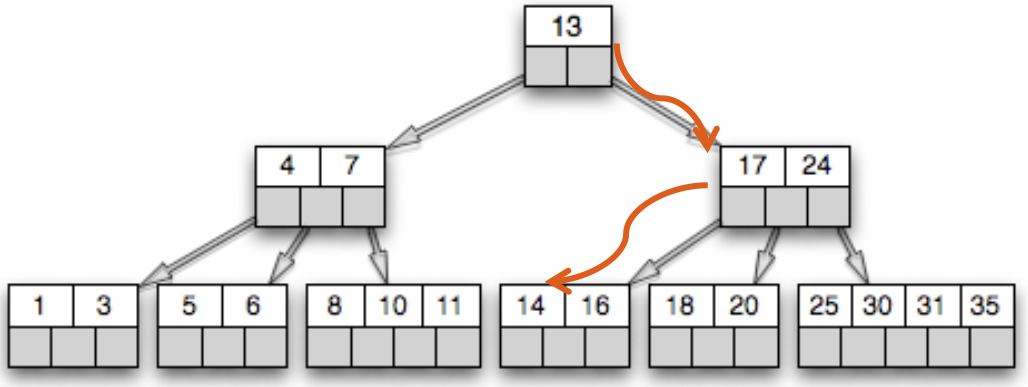
- Each node has at most 5 children
- (k-1) keys for k children
- All leaves at the same level

B-TREE (CONT'D)

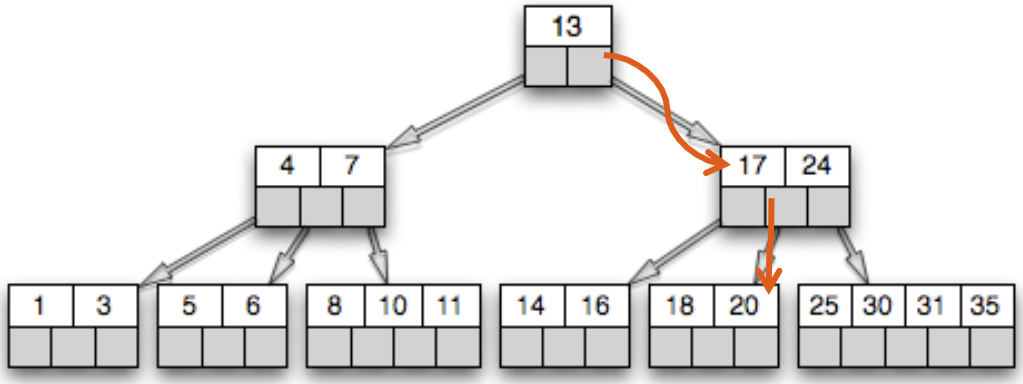
B-tree of n nodes

- **Space complexity?**
- **Search**
- **Insert**
- **Delete**

Finding 14



Insert 21



HASH TABLE

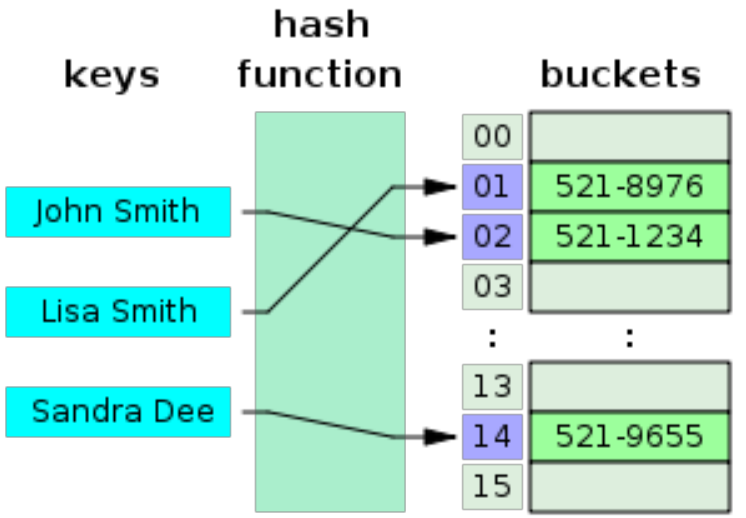
Used in memory management

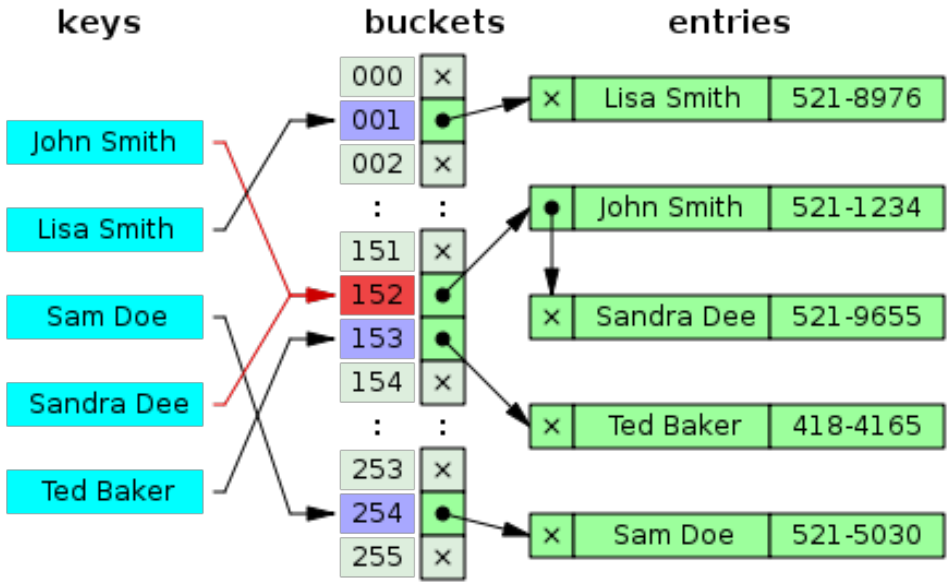
<key, value> pairs

Search & storage in O(1), except

when collision occurs

One solution to hash collision is to use separate chaining





PROCESS MANAGEMENT

READINGS: CHAPTER 3 - 4

OVERVIEW

Processes

States of a process

Operations on processes

- fork() , exec(), kill (), signal()

Cooperating Processes

- pipes

Threads and lightweight processes

PROCESSES

A process is a program executing a given sequential computation.

- An active entity unlike a program
- Think of the difference between a recipe in a cookbook and the activity of a cook preparing a dish according to the recipe!

There are many quasi-synonyms for process:

- Job (very old programmers still use it)
- Task
- ~~Program (strongly deprecated)~~

PROCESSES AND PROGRAMS (I)

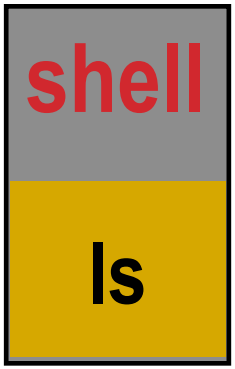
Can have one program and many processes

- When several users execute the same program (text editor, compiler, and so forth) at the same time, each execution of the program constitutes a separate process
- A program that forks another sequential computation gives birth to a new process.

PROCESSES AND PROGRAMS (II)

Can have one process and two—or more—programs

- A process that performs an exec() call replaces the program it was executing



IMPORTANCE OF PROCESSES

Processes are the basic entities managed by the operating system

- OS provides to each process the illusion it has the whole machine for itself
- Each process has a dedicated address space

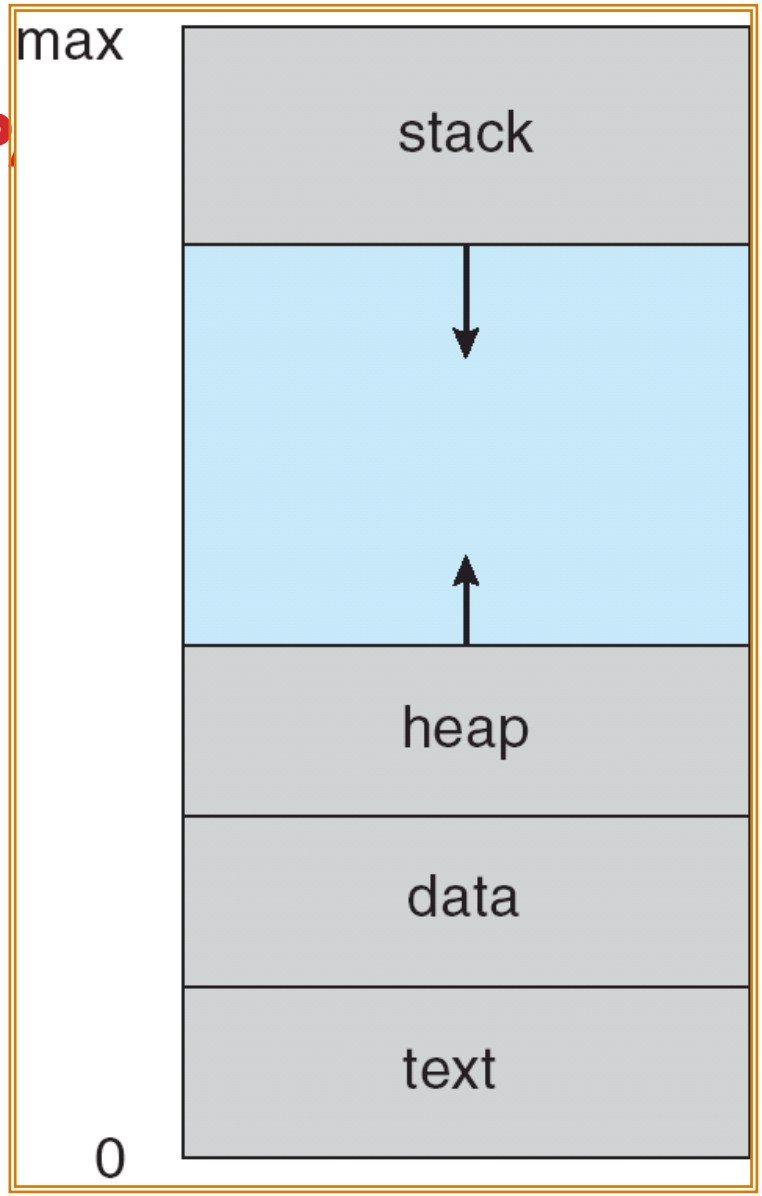
THE PROCESS ADDRESS SP

Set of main memory locations allocated to the process

- Other processes cannot access them
- Process cannot access address spaces of other processes

A process address space is the playpen or the sandbox of its owner

- Process address space is contiguous

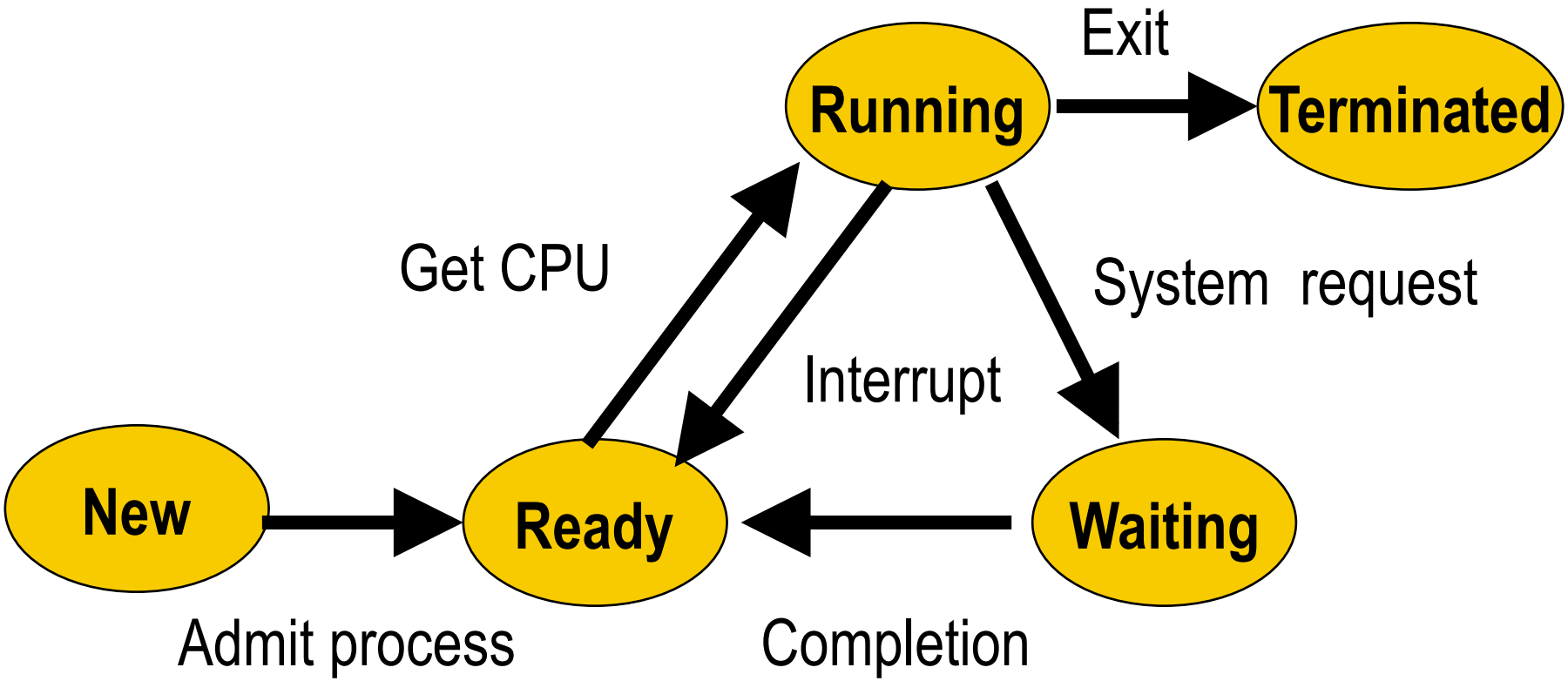


PROCESS STATES

Processes go repeatedly through several stages during their execution

- Waiting to get into main memory
- Waiting for the CPU
- Running
- Waiting for the completion of a system call

STATE DIAGRAM



PROCESS ARRIVAL

New process

- Starts in NEW state
- Gets allocated a Process Control Block (PCB) and main memory
- Is put in the READY state waiting for CPU time

THE READY STATE

AKA the ready queue

Contains all processes waiting for the CPU

Organized as a priority queue

Processes leave the priority queue when they get some CPU time

- Move then to the RUNNING state

THE RUNNING STATE (I)

A process in the running state has exclusive use of the CPU until

- It terminates and goes to the TERMINATED state
- It does a system call and goes to the WAITING state
- It is interrupted and returns to the READY state

THE RUNNING STATE (II)

Processes are forced to relinquish the CPU and return to the READY state when

- A higher-priority process arrives in the ready queue and preempts the running process
  - Get out, I'm more urgent than you!
- A timer interrupt indicates that the process has exceeded its time slice of CPU time

THE WAITING STATE (I)

Contains all processes waiting for the completion of a system request:

- I/O operation
- Any other system call

Process is said to be waiting, blocked or even sleeping (UNIX slang)

THE WAITING STATE (II)

A system call that does not require callers to wait until its completion is said to be non-blocking

- Calling processes are immediately returned to the READY state

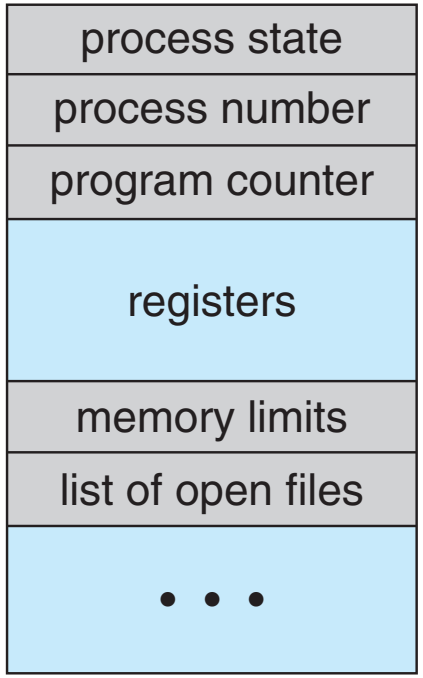
The waiting state is organized as a set of queues

- One queue per device, OS resource

THE PROCESS CONTROL BLOCK (I)

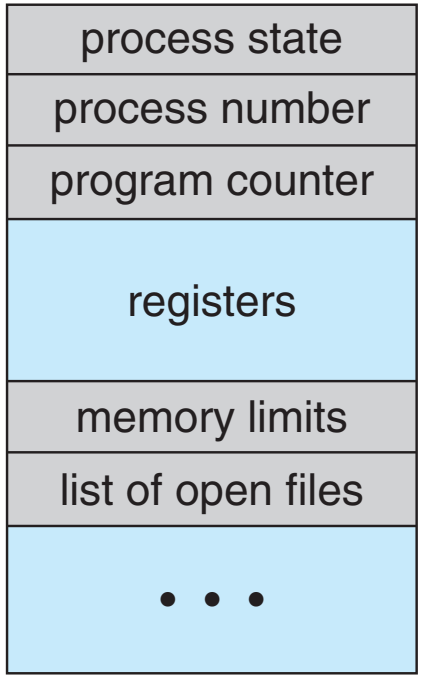
Contains all the information associated with a specific process:

- Process identification (pid), argument vector, ...
  - UNIX pids are unique integers
- Process state (new, ready, running, …),
- CPU scheduling information
  - Process priority, processors on which the process can run, ...,



THE PROCESS CONTROL BLOCK (II)

- Program counter and other CPU registers including the Program Status Word (PSW),
- Memory management information
  - Very system specific,
- Accounting information
  - CPU time used, system time used, ...
- I/O status information (list of opened files, allocated devices, ...)



THE PROCESS TABLE

System-wide table containing

- Process identification (pid), argument vector, ...
- Process current state
- Process parents
- Process priority and other CPU scheduling information
- A pointer to the executable machine code of a process

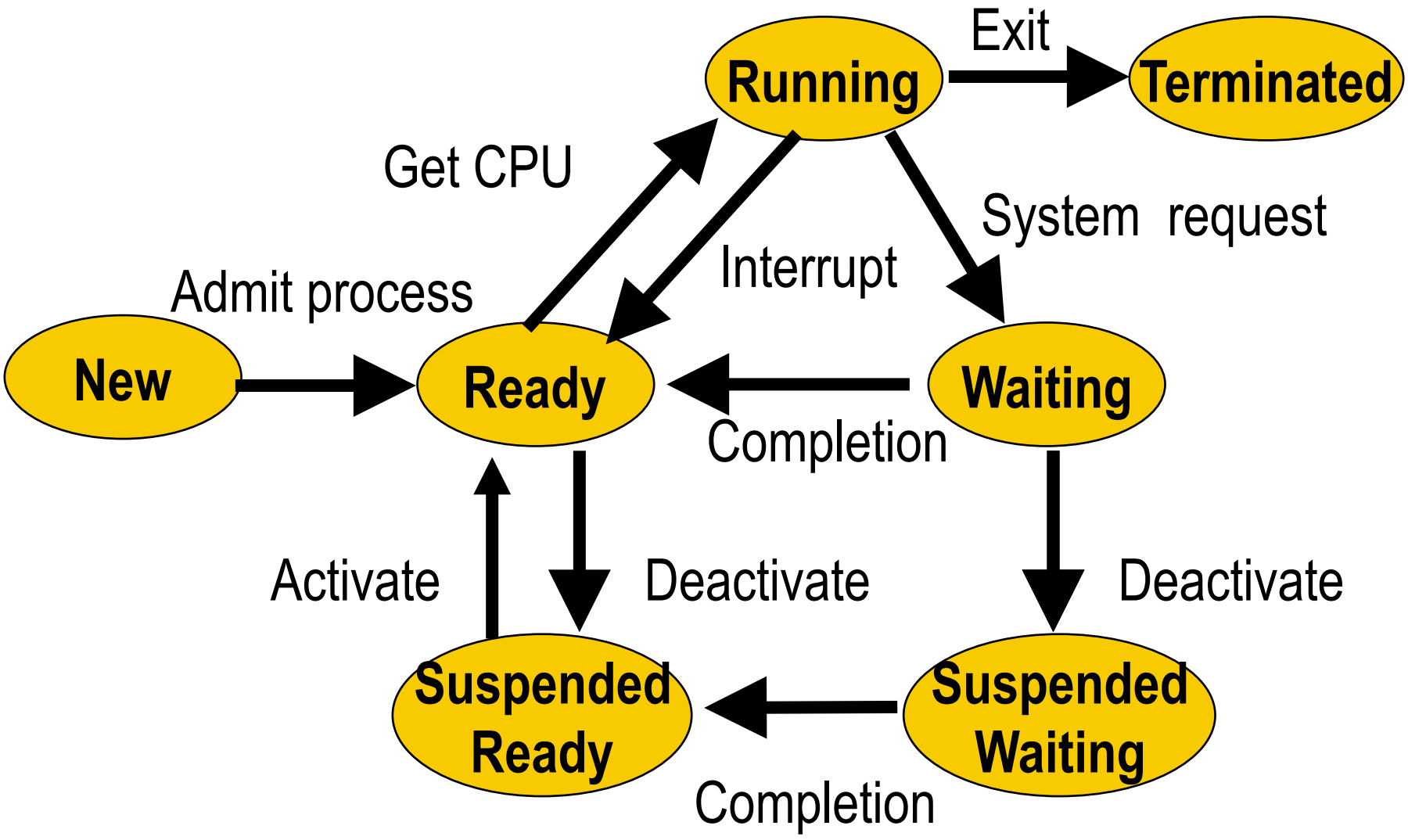
SWAPPING

Whenever the system is very loaded, we might want to expel from main memory or swap out

- Low priority processes
- Processes that have been waiting for a long time for an external event

These processes are said to be swapped out or suspended.

HOW IT WORKS


Running
Exit
Terminated
Get CPU
System request
Interrupt
Admit process
New
Ready
Waiting
Completion
Activate
Deactivate
Deactivate
Suspended Ready
Suspended Waiting
Completion

SUSPENDED PROCESSES

Suspended processes

- Do not reside in main memory
- Continue to be included in the process table

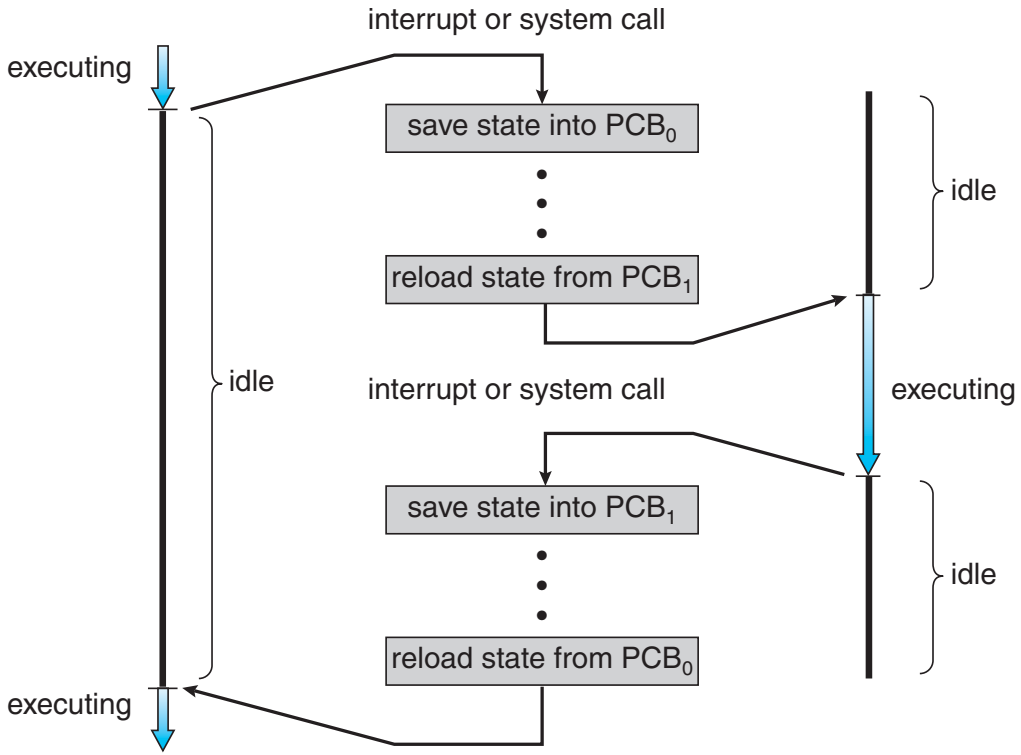
Can distinguish between two types of suspended processes:

- Waiting for the completion of some request (waiting_suspended)
- Ready to run (ready_suspended).

CONTEXT SWITCHING

When one process is running on a CPU and another needs to run on the same CPU, we need to switch between the processes

This is called a *context switch* (or a "state" save and "state" restore)



CONTEXT SWITCHING

The last step of a context switch is typically to load the program counter for the process that is about to run

Context switch has a time cost

- Dependent on hardware (e.g., # of machine registers)
- Not executing user processes during the time a context switch is occurring

OPERATIONS ON PROCESSES

Process creation

- fork()
- exec()
- The argument vector

Process deletion

- kill()
- signal()