CSCI-GA.2250-001

Operating Systems

Process/Thread Scheduling

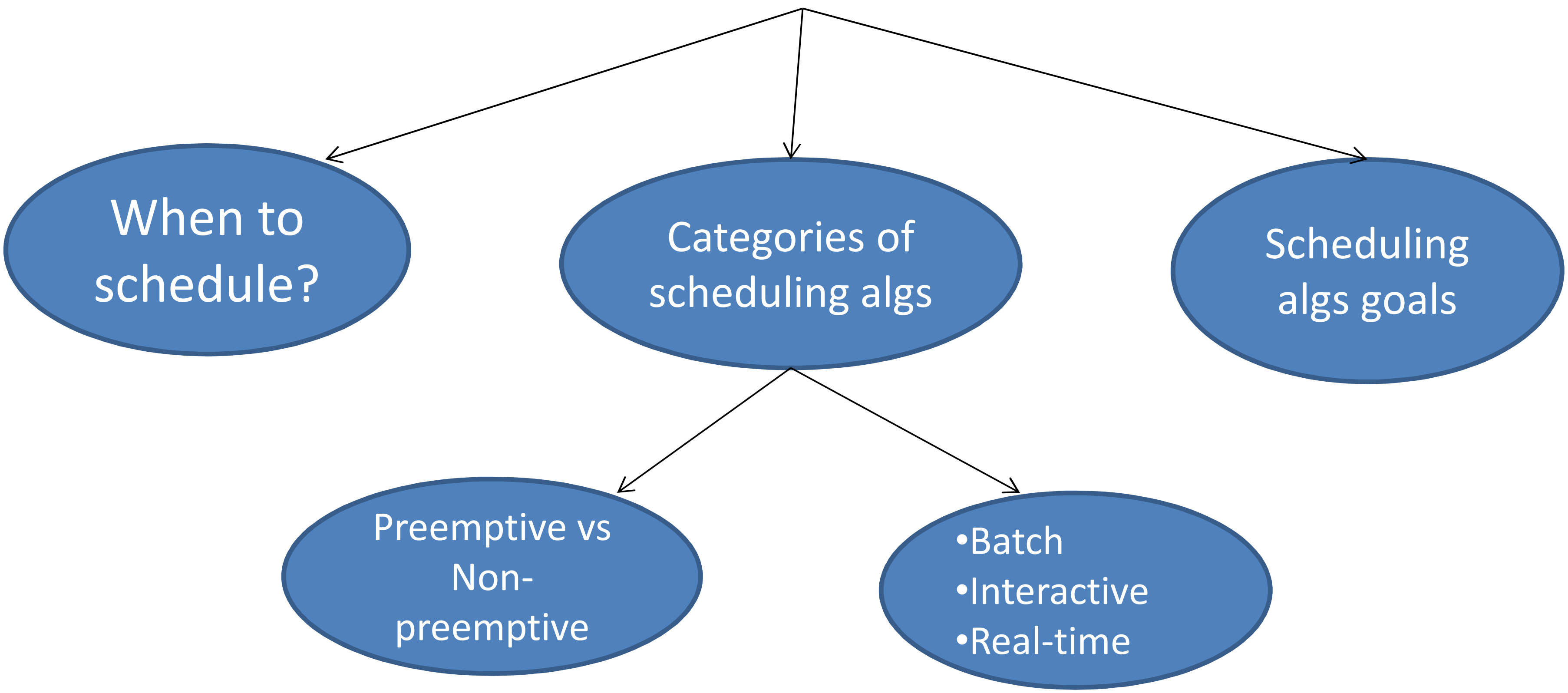
Hubertus Franke
frankeh@cs.nyu.edu

Scheduling

- Whether scheduling is based on processes or threads depends on whether the OS in multi-threading capable.
- In this deck/lab2 we use process as the assumption, just be aware that it applies to threads in a multi-threading capable OS.
- Given a group of ready processes or threads, which process/thread to run?

Scheduling

Given a group of ready processes, which process to run?



When to Schedule?

- When a process is created
- When a process exits
- When a process blocks
- When an I/O interrupt occurs

Categories of Scheduling Algorithms

- Batch
  - No users impatiently waiting
  - mostly non-preemptive, or preemptive with long period for each process
- Interactive
  - preemption is essential
- Real-time
  - deadlines

How/what to measure "Scheduling"

- Turn Around Time (Batch)
- Throughput (e.g. jobs per second)
- Response Time (Interactive)
- Average wait times

Scheduling – Process Behavior

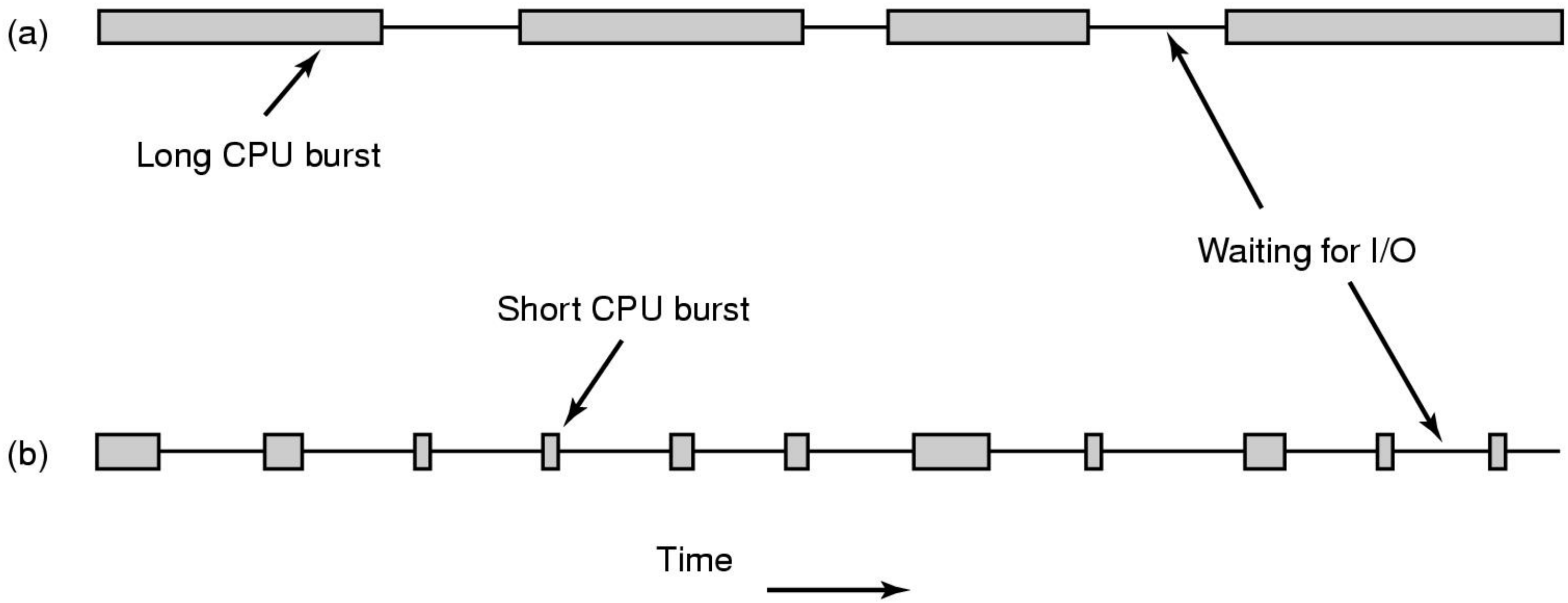
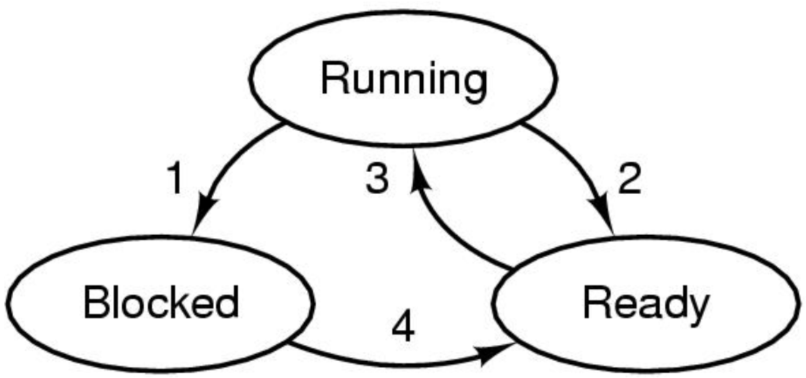


Figure 2-38. Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process.

Almost ALL scheduling Algorithms can be described by the following state diagram



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

Scheduling Algorithms Goals

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

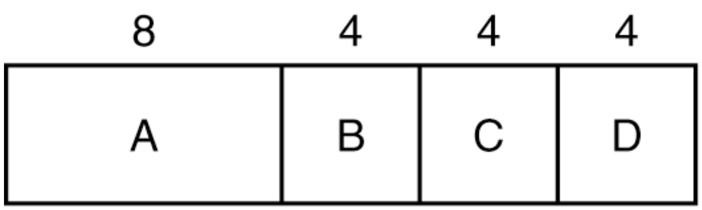
Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

Scheduling in Batch Systems: First-Come First-Served

- Non-preemptive
- Processes ordered as queue
- A new process added to the end of the queue
- A blocked process that becomes ready added to the end of the queue
- Main disadv: Can hurt I/O bound processes

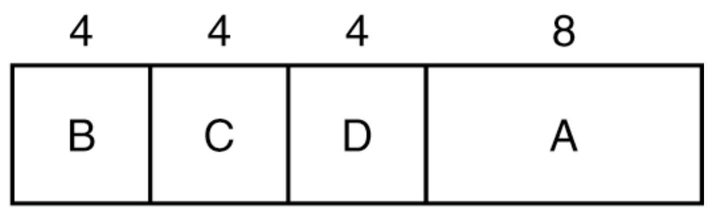
Scheduling in Batch Systems: Shortest Job First

- Non-preemtive
- Assumes runtime is known in advance
- Is only optimal when all the jobs are available simultaneously



(a)

Run in original order



(b)

Run in shortest job first

Scheduling in Batch Systems: Shortest Remaining Time Next

- Preemptive
- Scheduler always chooses the process whose remaining time is the shortest
- Runtime has to be known in advance

Scheduling in Interactive Systems: Round-Robin

- Each process is assigned a time interval: quantum
- After this quantum, the CPU is given to another process
- What is the length of this quantum?
  - too short -> too many context switches -> lower CPU efficiency
  - too long -> poor response to short interactive
  - quantum longer than CPU burst is good (why?)

Round-Robin Scheduling (cont)

- Promotes Fairness

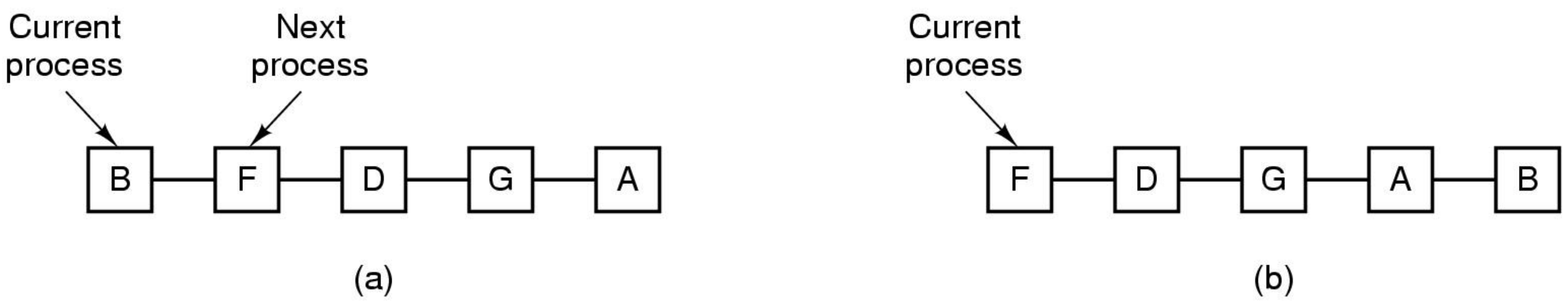
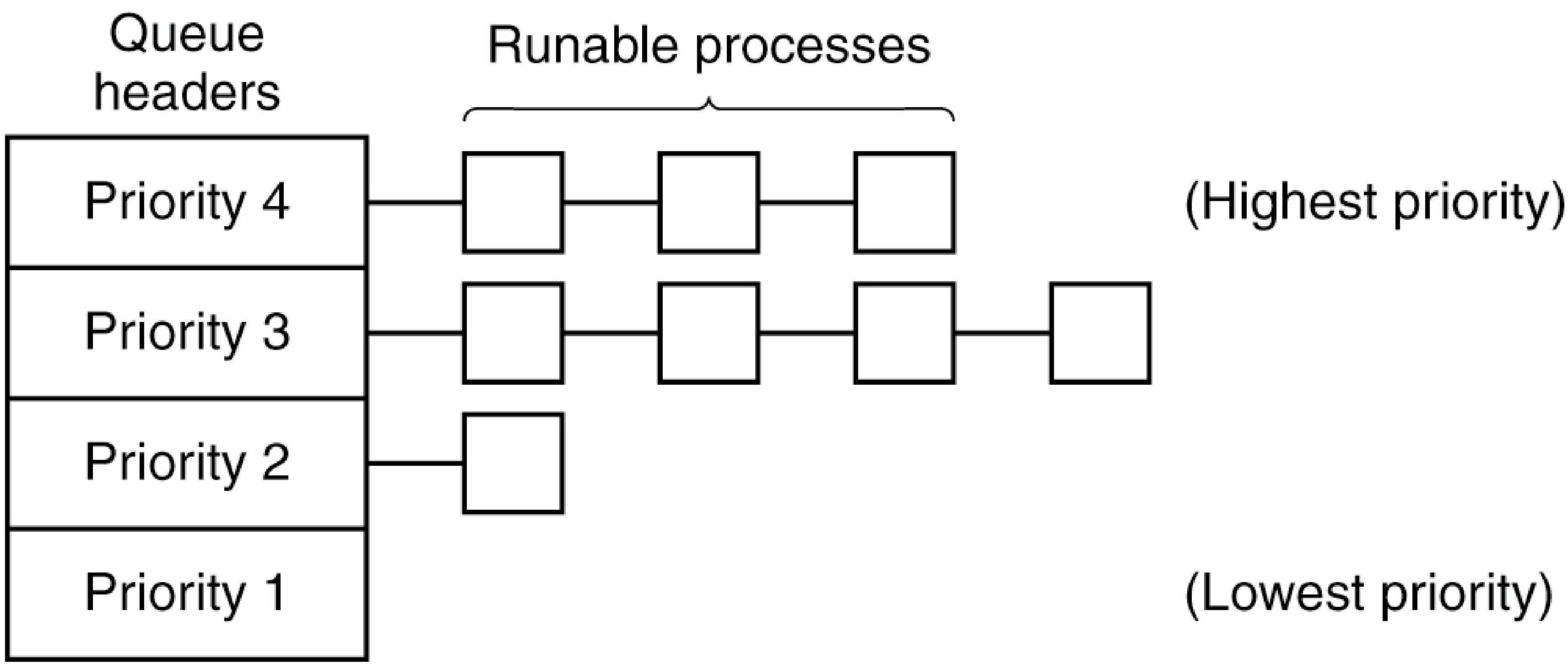


Figure 2-41. Round-robin scheduling.
(a) The list of runnable processes. (b) The list of runnable processes after B uses up its quantum.

Scheduling in Interactive Systems: Priority Scheduling

- Each process is assigned a priority
- runnable process with the highest priority is allowed to run
- Priorities are assigned statically or dynamically
- Must not allow a process to run forever
  - Can decrease the priority of the currently running process
  - Use time quantum for each process

Scheduling in Interactive Systems: Multiple Queues



Multi-Level FeedBack Queues

- Multiple levels of priority
- Each level is run round-robin
- If process has to be preempted, moves to worse priority.
- What kind of process should be in bottom queue?

Lottery Scheduling

- Each runnable entity is given a certain number of tickets.
- The more tickets you have, the higher your odds of winning.
- Trade tickets?
- Problems?

Fair Share Scheduler

- Schedule not only based on individual process, but process's owner.
- N users, each user may have different # of processes.
- Does this make sense on a PC?

Policy versus Mechanism

- Separate what is allowed to be done with how it is done
  - a process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
  - mechanism in the kernel
- Parameters filled in by user processes
  - policy set by user process

Scheduling in Real-Time

- Process must respond to an event within a deadline
- Hard real-time vs soft real-time
- Periodic vs aperiodic events
- Processes must be schedulable
- Scheduling algorithms can be static or dynamic

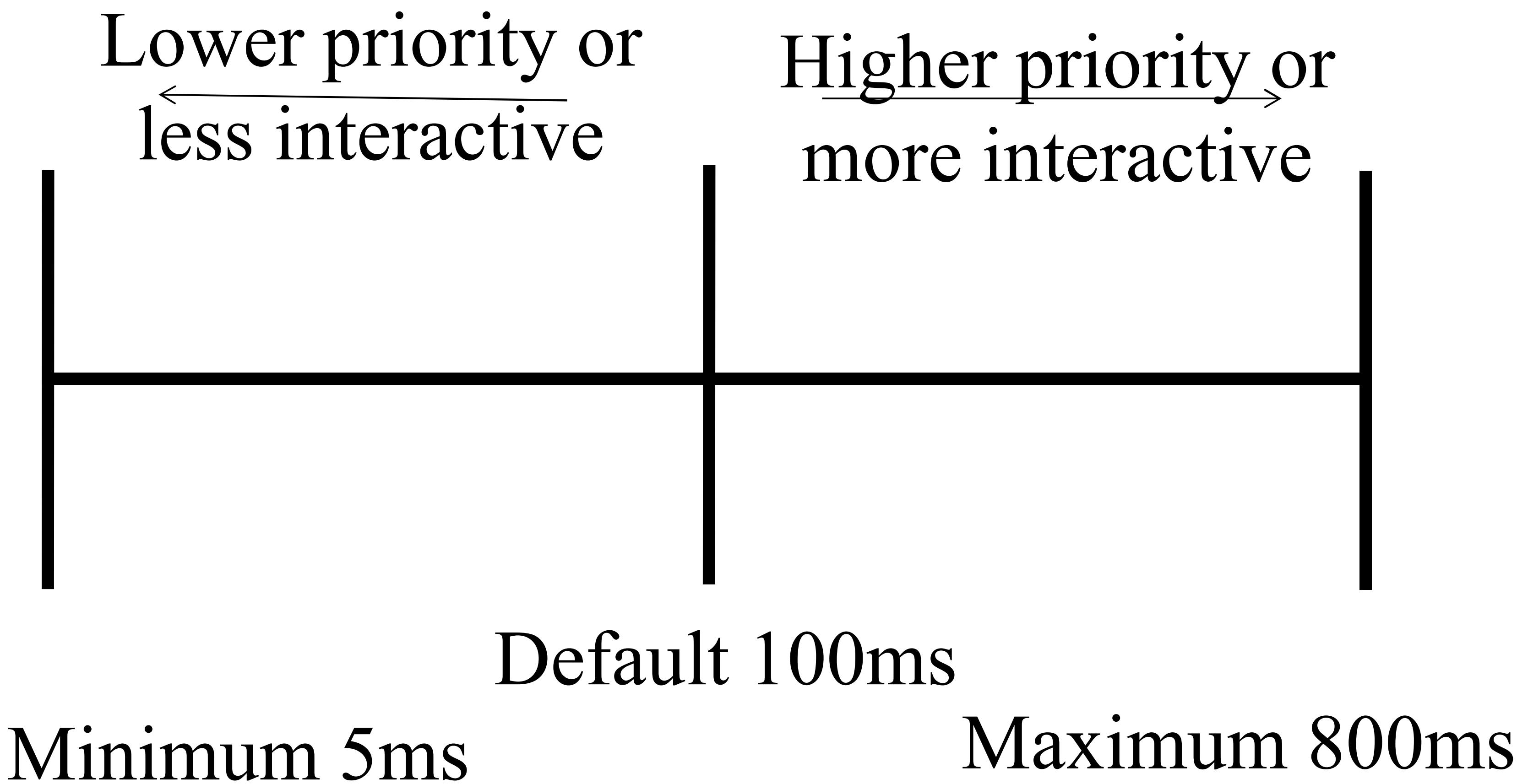
Thread Scheduling

- Two levels of parallelism: processes and threads within processes
- Kernel-bases vs user-space

Example Linux Scheduling

- Implementation has changed multiple times
- Dynamic Priority-Based Scheduling
- Two Priority Ranges
  - Nice value -20 to +19, default 0. Larger values are lower priority. Nice value of -20 receives maximum timeslice, +19 minimum.
  - Real-time priority. By default values range 0 to 99. Real-time processes have a higher priority than normal processes.

Linux Timeslice



Scheduler Goals

- O(1) scheduling – constant time
- SMP – each processor has its own locking and individual runqueue
- SMP Affinity. Only migrate process from one CPU to another if imbalanced runqueues.
- Good interactive performance
- Fairness
- Optimized for one or two processes but scales

Runqueues
<kernel/sched.c> struct runqueue

*active – active priority array

*expired – expired priority array

arrays[2] –priority arrays

*migration_thread

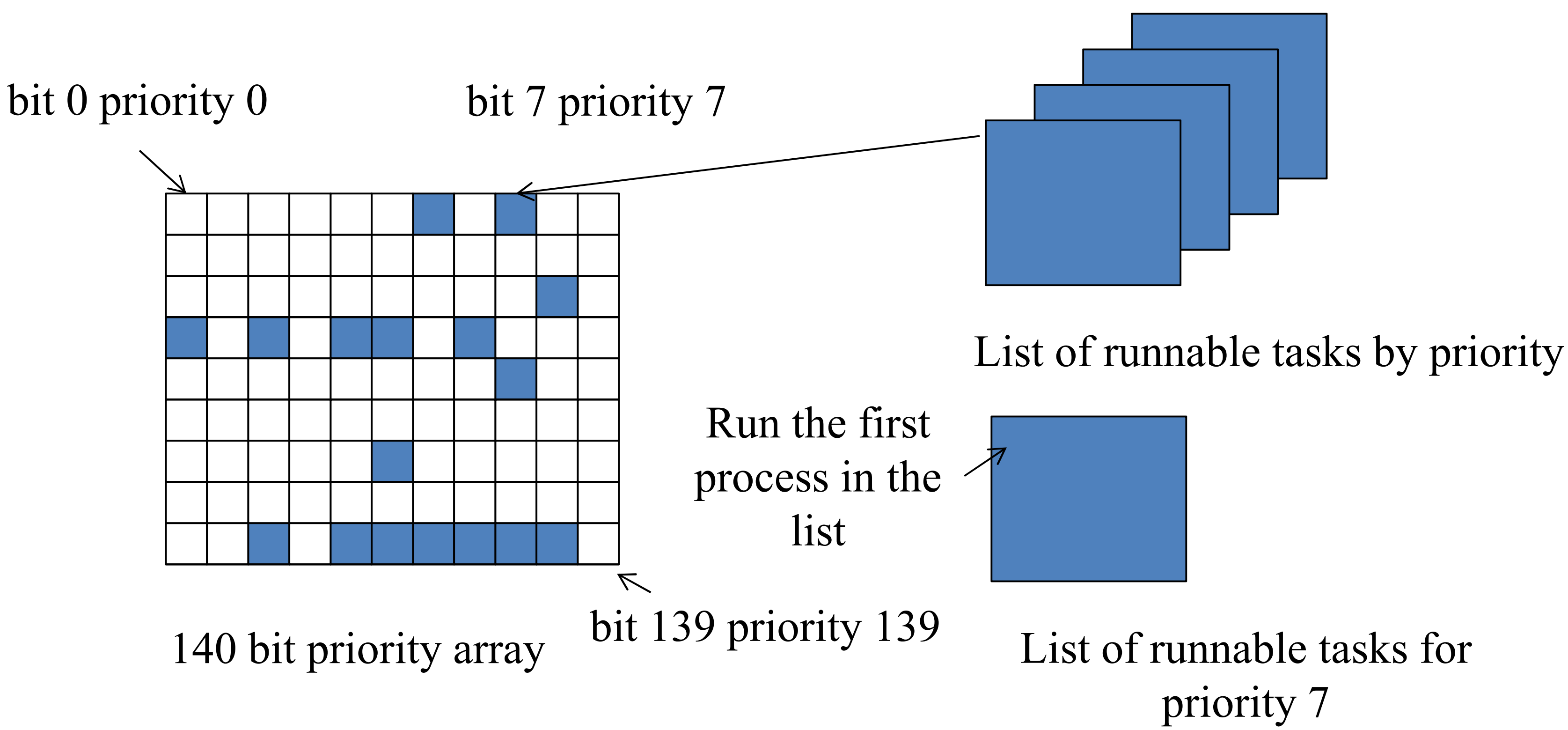
migration_queue

nr_iowait – number of tasks waiting on I/O

Priority Arrays

- Each runqueue has two priority arrays – active expired
- Each priority array contains a bitmap
  - If bit is set in bitmap, it indicates there are processes with a given priority. (There is also a count.)
- Allows constant retrieval algorithm to find highest set bit

Scheduler Algorithm



Calculating Priority and Timeslice

effective_prio() returns task's dynamic priority.

nice value + or – bonus in range -5 to +5

Interactivity measure by how much time a process sleeps. Indicates I/O activity.

sleep_avg incremented to max_sleep_avg (10 millisecs) every time does I/O. If no I/O, decremented.

Load Balancing

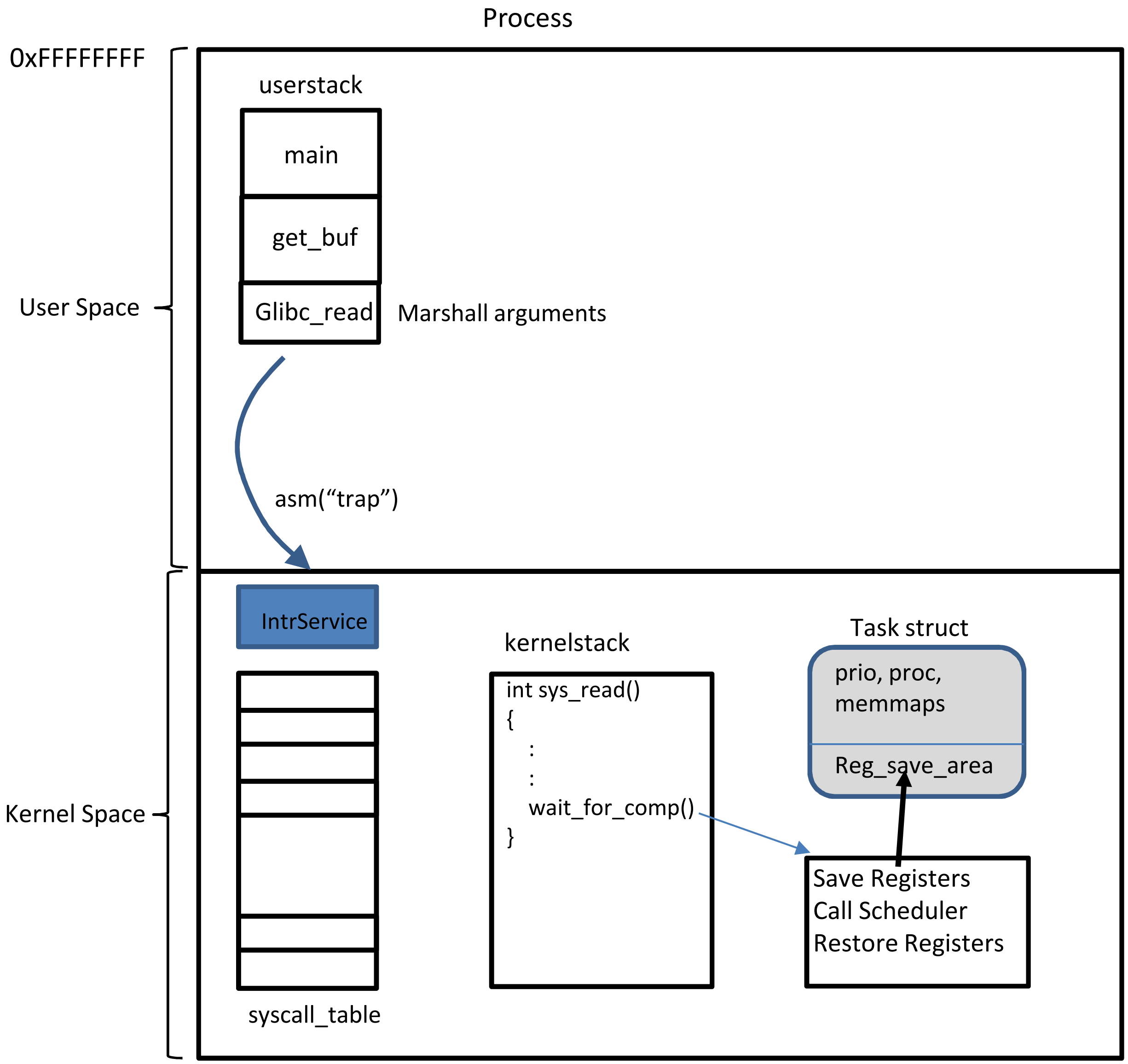
- New processes are typically located on local CPU
- In multi processor environment each CPU runs a scheduler instance (see data structures) and schedules in that domain
- Occasionally or when no process is runnable, scheduler_i looks to steal work elsewhere
- Each scheduler maintains a load average and history to determine stability of its load
- Typically done in a hierarchy

- Let's discuss:

Context Switch

- Scenarios:
  1) Current process blocks <u>OR</u>
  2) Preemption.

CtxSwitch: Process Blocks




```
int call_io()
{
   char buffer[128];
   return read(0,buffer,128);
}

main() {
   call_io();
}
```

CtxSwitch: Preemption