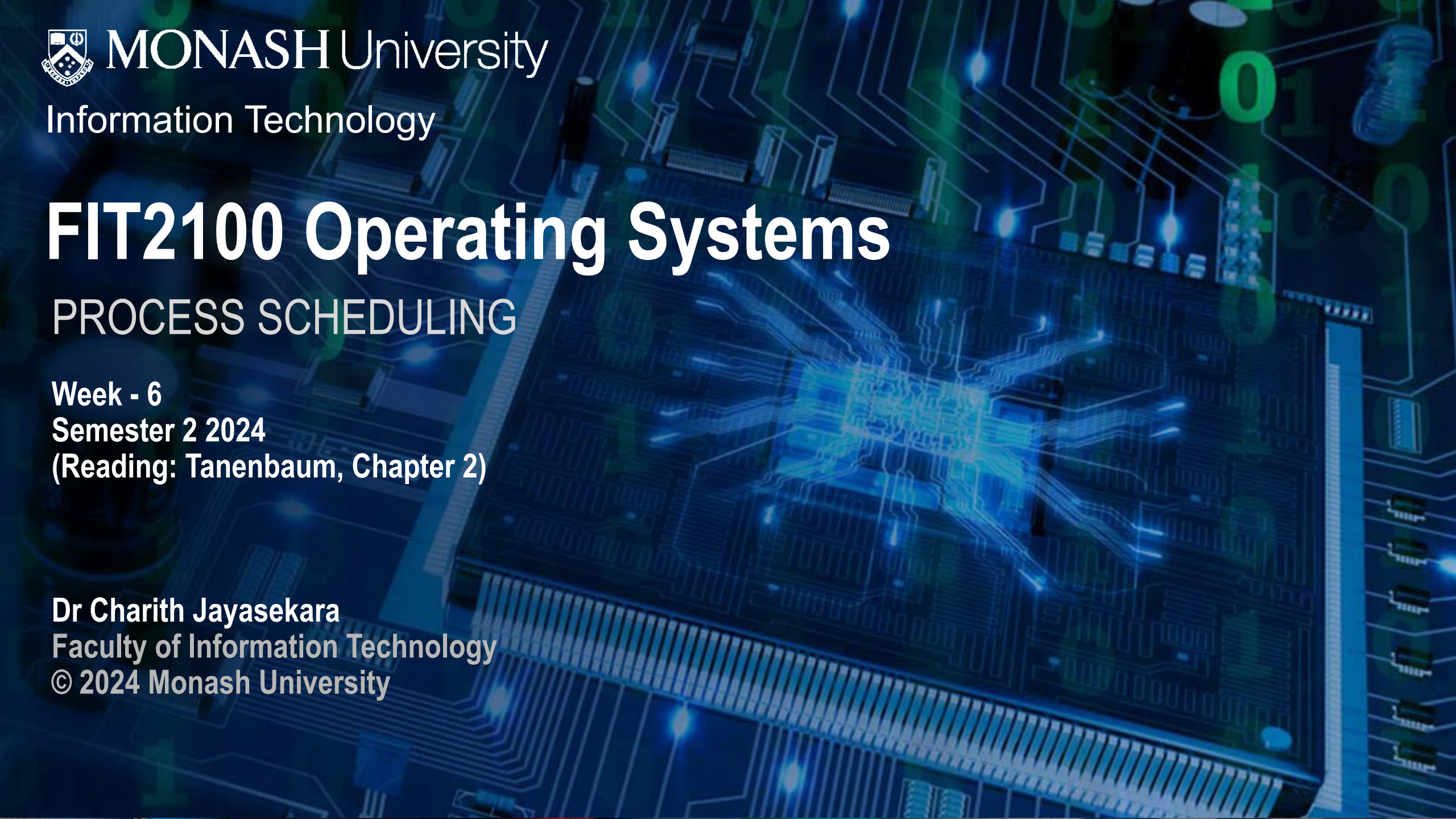


MONASH University

Information Technology

FIT2100 Operating Systems

PROCESS SCHEDULING

Week - 6
Semester 2 2024
(Reading: Tanenbaum, Chapter 2)

Dr Charith Jayasekara
Faculty of Information Technology

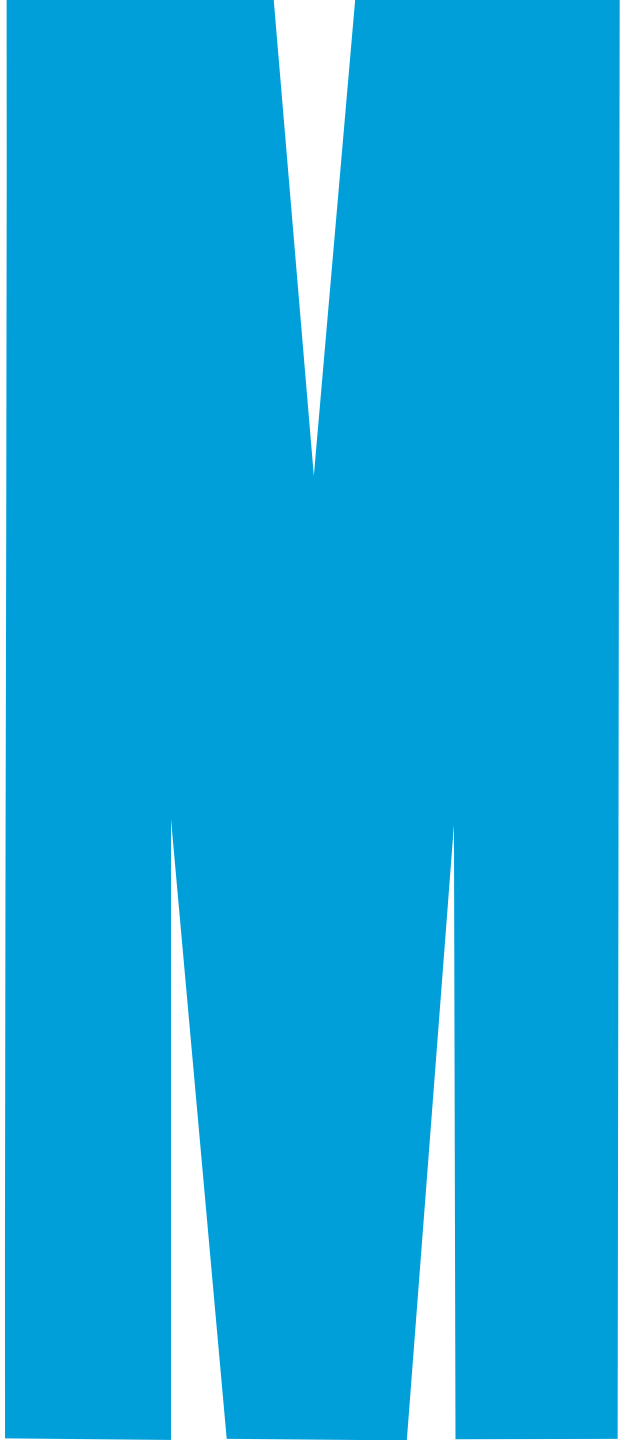
OUTLINE

- Introduction to scheduling
- Scheduling in Batch Systems
- Scheduling in Interactive Systems
- Scheduling in Real-time Systems

LEARNING OUTCOMES

- Upon the completion, you should be able to:
  - Understand the basic concepts in different **scheduling systems**.
  - Understand fundamentals of process scheduling algorithms.
  - Assess performance of different process **scheduling algorithms**.

INTRODUCTION



INTRODUCTION TO SCHEDULING

- When a computer is multiprogrammed, it frequently has multiple processes or threads competing for the CPU at the same time.
- This situation occurs whenever two or more of them are simultaneously in the ready state.
- If only one CPU is available, a choice has to be made which process to run next.
- The part of the operating system that makes the choice is called the **scheduler**, and the algorithm it uses is called the **scheduling algorithm**

INTRODUCTION TO SCHEDULING

HISTORY

- In older batch systems, the scheduling algorithm was simple as it only needed to run the next job in the punchcard stack, or on the magnetic tape.
- In a multiprogramming system, the scheduling algorithm is complicated because typically, multiple users compete for CPU time.
- Historically, as CPU time was a scarce resource, a good scheduler could make a huge difference in performance and user satisfaction.

INTRODUCTION TO SCHEDULING

ADVENT OF PERSONAL COMPUTERS

- Most of the time there is only one active process.
- Computers have become so much faster over the last 50 years that the CPU is seldom now a scarce resource.
- In a multiprogramming system, the scheduling algorithm is complicated because typically, multiple users compete for CPU time.
- Even when two programs are actually running at once, such as a word processor and a spreadsheet, it hardly mattered which ran first since the user was probably waiting for both of them to finish. As a consequence, scheduling did not matter much on early PCs.
- Many embedded products, e.g. cameras, continue to use this style of (non-)scheduling

INTRODUCTION TO SCHEDULING

WHY SCHEDULING IS STILL IMPORTANT

- In most processing platforms, multiple processes usually compete for CPU time, so scheduling remains a critical function for operating systems.
    - For example, when the CPU has to choose between running a process that gathers daily statistics or indexes disk data and one that services user requests, the users will be a lot happier if their processes get more CPU time and earlier.
- Mobile devices, such as notebooks, tablets and smartphones, are now competing with desktop systems in performance, and running operating systems that are derivatives of desktop operating systems – Android derived from Linux, and iOS/iPadOS derived from BSD/Mac OS X.
- The need to optimise power consumption is another requirement, especially with portable devices such as notebooks, tablets and smartphones.

INTRODUCTION TO SCHEDULING

WHY SCHEDULING IS STILL IMPORTANT

- The scheduler also has to worry about making efficient use of the CPU because process *context switching* is expensive.

    1. a switch from user mode to kernel mode must occur.
    2. the state of the current process must be saved, including storing its registers in the process table so they can be reloaded later. In some systems, the memory map (e.g., memory reference bits in the page table) must be saved as well.
    3. Next a new process must be selected by running the scheduling algorithm.
    4. After that, the memory management unit (MMU) must be reloaded with the memory map of the new process. This gets more complicated if process is not memory resident
    5. Finally, the new process must be started.

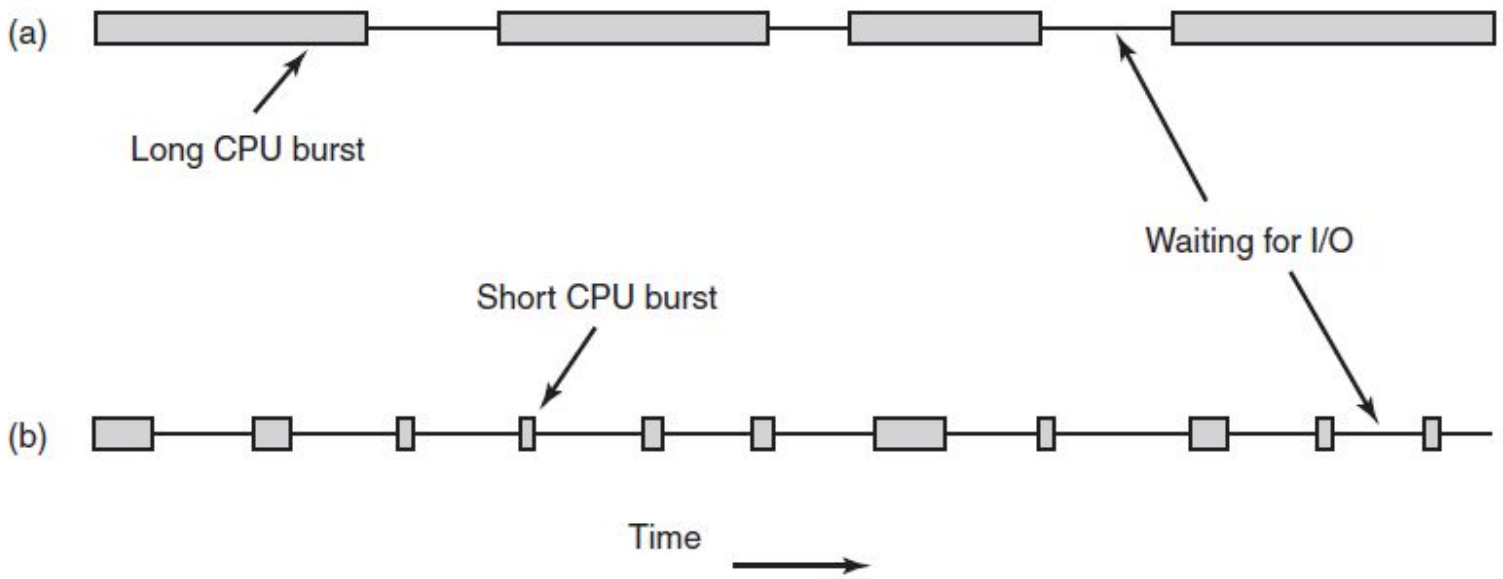
INTRODUCTION TO SCHEDULING

WHY SCHEDULING IS STILL IMPORTANT

- In addition to all that, the process context switch may invalidate the memory cache and related tables, forcing it to be dynamically reloaded from the main memory twice (upon entering the kernel and upon leaving it).
- All in all, doing too many process switches per second can chew up a substantial amount of CPU time, so caution is advised.
- *A case study: an experiment in industry running a 40 MHz clocked single CPU RISC architecture desktop machine (SunOS 4.3BSD) involved the machine servicing a Z85C30 serial communications chip that generated ~9,500 interrupts per second. System monitoring tools showed that more than 95% of CPU time was consumed performing context switching, and executing the very short ISR code!*
- **Punchline? Context switching is expensive in CPU time and should be treated as an unavoidable but expensive overhead**

INTRODUCTION TO SCHEDULING

PROCESS BEHAVIOUR - BURSTS OF CPU USAGE – CPU VS. I/O BOUND PROCESSES



(a) A CPU-bound process - spend most of their time computing .
(b) An I/O-bound process - spend most of their time waiting for I/O.

INTRODUCTION TO SCHEDULING

WHEN TO SCHEDULE

A. A new process is created

B. When a process exits

C. When a process blocks on I/O

D. When an I/O interrupt occurs

E. When a timer interrupt occurs

INTRODUCTION TO SCHEDULING

NON-PRE-EMPTIVE AND PRE-EMPTIVE

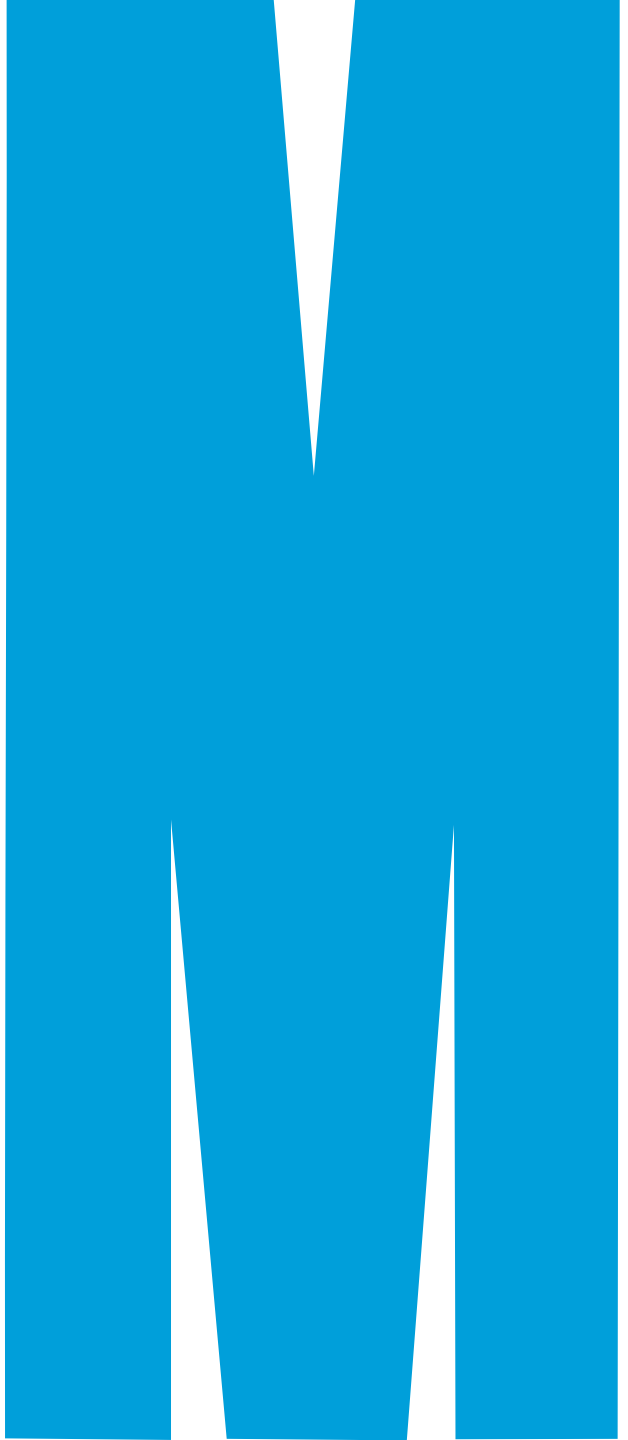
- Non-Pre-emptive
  - The scheduling algorithm picks a process to run and then just lets it run until it blocks (either on I/O or waiting for another process) or voluntarily releases the CPU.
  - Even if it runs for many hours, it will not be forcibly suspended.
  - In effect, no scheduling decisions are made during clock interrupts.
  - After clock-interrupt processing has been finished, the process that was running before the interrupt is resumed, unless a higher-priority process was waiting for a now-satisfied timeout.

INTRODUCTION TO SCHEDULING

NON-PRE-EMPTIVE AND PRE-EMPTIVE

- **Pre-emptive:**
  - The scheduling algorithm picks a process and lets it run for a maximum of some fixed time. If it is still running at the end of the time interval, it is suspended, and the scheduler picks another process to run (if one is available).
  - Doing pre-emptive scheduling requires having a clock interrupt occur at the end of the time interval to give control of the CPU back to the scheduler.
  - If no clock is available, non-preemptive scheduling is the only option.

CATEGORIES OF SCHEDULING ALGORITHMS



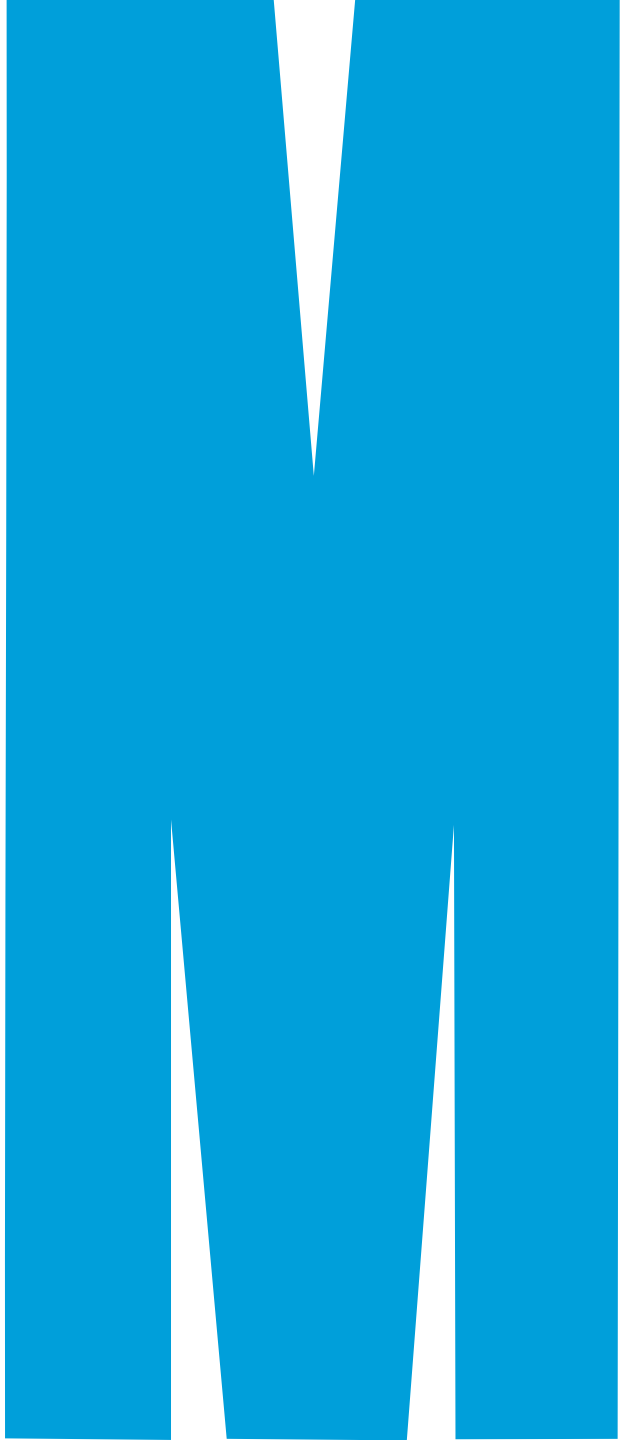
CATEGORIES OF SCHEDULING ALGORITHMS

1. Batch.
    a. non-pre-emptive algorithms, or pre-emptive algorithms with long time periods for each process, are often acceptable, e.g. payroll, inventory, accounts receivable, interest calculations (at banks), long running scientific or engineering simulation jobs.
2. Interactive.
    a. Pre-emption is essential to keep one process from hogging the CPU and denying service to the others.
    b. when multiple users must be serviced, all of whom are in a big hurry, e.g. servers
3. Real-time.
    a. Pre-emption is essential as processes have typically ranked priorities depending on the importance of the task performed, and timing of the task has critical constraints.
    b. The difference against interactive systems is that real-time systems have critical time constraints on the execution of a task that cannot be exceeded without causing a problem (e.g. flight control systems in aircraft where an accident might happen if the computation runs late)

GOALS OF SCHEDULING ALGORITHMS

- All systems
  - Policy enforcement - seeing that stated policy is carried out
  - Load Balancing - 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
  - Fairness - giving each process a fair share of the CPU
  - Response time - respond to requests quickly
  - Proportionality - meet users' expectations
- Real-time systems
  - Meeting execution time deadlines - avoid failed deadlines and resulting system failures
  - Predictability - avoid variable system behaviour over time, performance must be repeatable

SCHEDULING BATCH SYSTEMS



SCHEDULING BATCH SYSTEM

- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next

FIRST-COME FIRST-SERVED (FCFS)

- With this algorithm, processes are assigned the CPU in the order they request it.
- Basically, there is a single queue of ready processes.
- Adding a new job or unblocked process just requires attaching it to the end of the queue.
- Simpler to understand and easy to implement.
- Some processes can dominate the CPU time compared to other processes.

SHORTEST JOB FIRST (SJF)

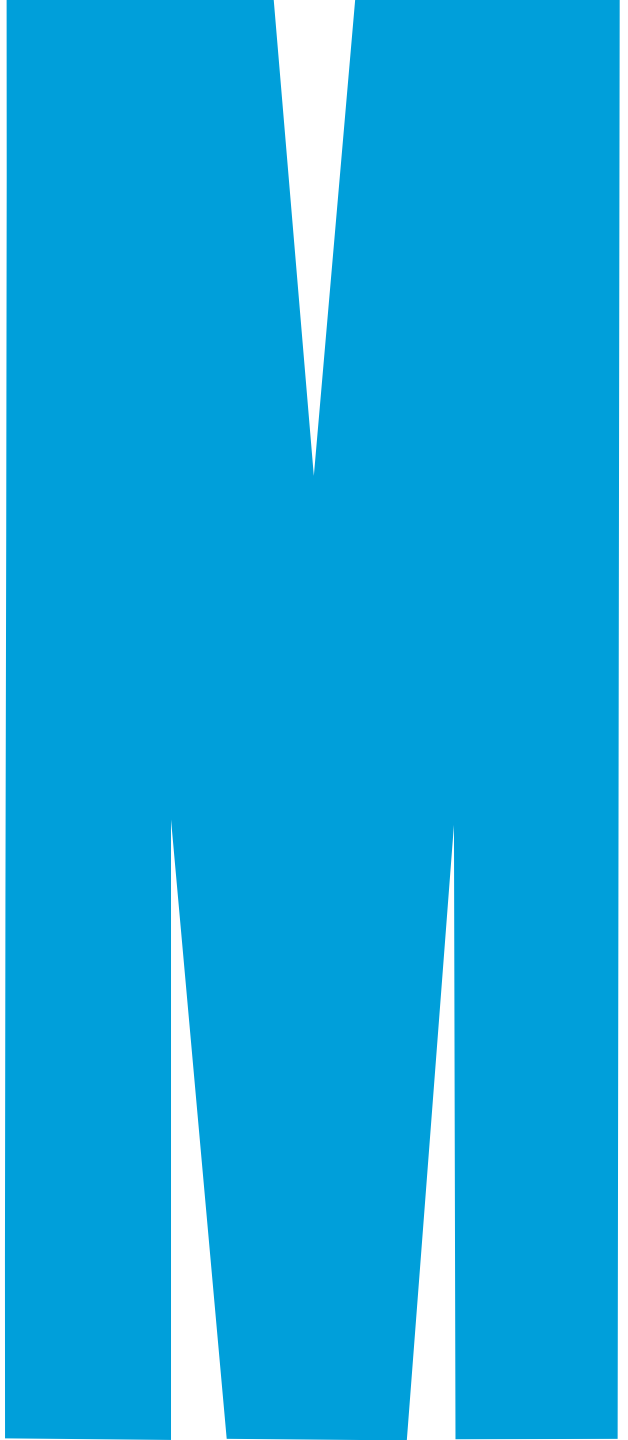
- Scheduler picks the shortest job first.

An example of shortest job first scheduling.
(a) Running four jobs in the original order (FCFS), vs.
(b) Running them in shortest job first order.

SHORTEST REMAINING TIME NEXT (SRTN)

- Preemptive version of the Shortest Job First.
- The scheduler always chooses the process whose remaining run time is the shortest.
- This scheme allows new short jobs to get good service.

SCHEDULING IN INTERACTIVE SYSTEMS



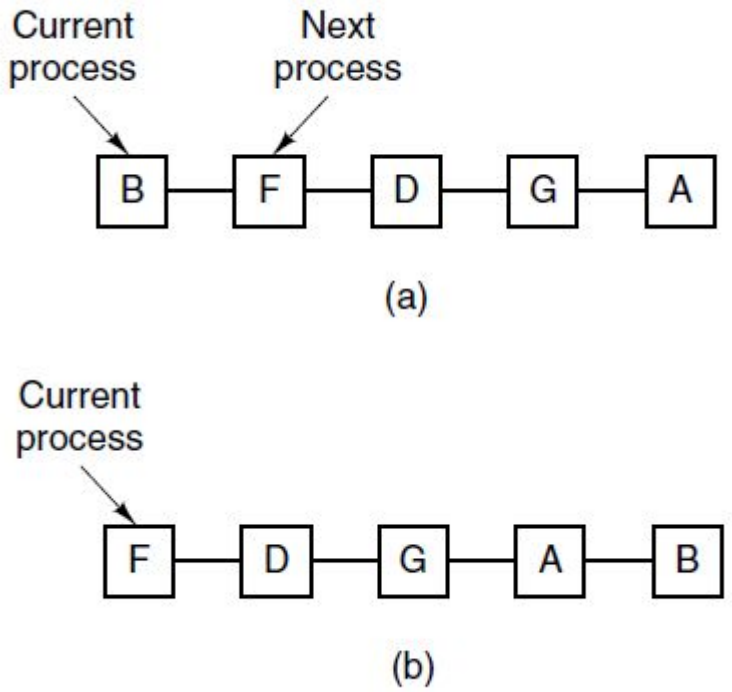
SCHEDULING IN INTERACTIVE SYSTEM

- Round-Robin Scheduling
- Priority Scheduling
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling

ROUND- ROBIN SCHEDULING (RR)

- One of the oldest, simplest, fairest, and most widely used algorithms is “round robin”.
- Each process is assigned a time interval, called its quantum, during which it is allowed to run.
- If the process is still running at the end of the quantum, the process is pre-empted and the CPU is given to another process.
- If the process has blocked or finished before the quantum has elapsed, the CPU switching is done when the process blocks, of course.
- Round robin is easy to implement.
- All the scheduler needs to do is maintain a list of runnable processes.

ROUND- ROBIN SCHEDULING (RR)



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

ROUND- ROBIN SCHEDULING (RR)

- Critical parameter is the length of the quantum.
- Switching from one process to another requires a certain amount of time for doing all the administration— saving and loading registers and memory maps, updating various tables and lists, flushing and reloading the memory cache, and so on.
- Suppose that this process switch or context switch, as it is sometimes called, takes 1 msec, including switching memory maps, flushing and reloading the cache, etc. Also suppose that the quantum is set at 4 msec.
- With these parameters, after doing 4 msec of useful work, the CPU will have to spend (i.e., waste) 1 msec on process switching. Thus 20% of the CPU time will be thrown away on administrative overhead.

PRIORITY SCHEDULING

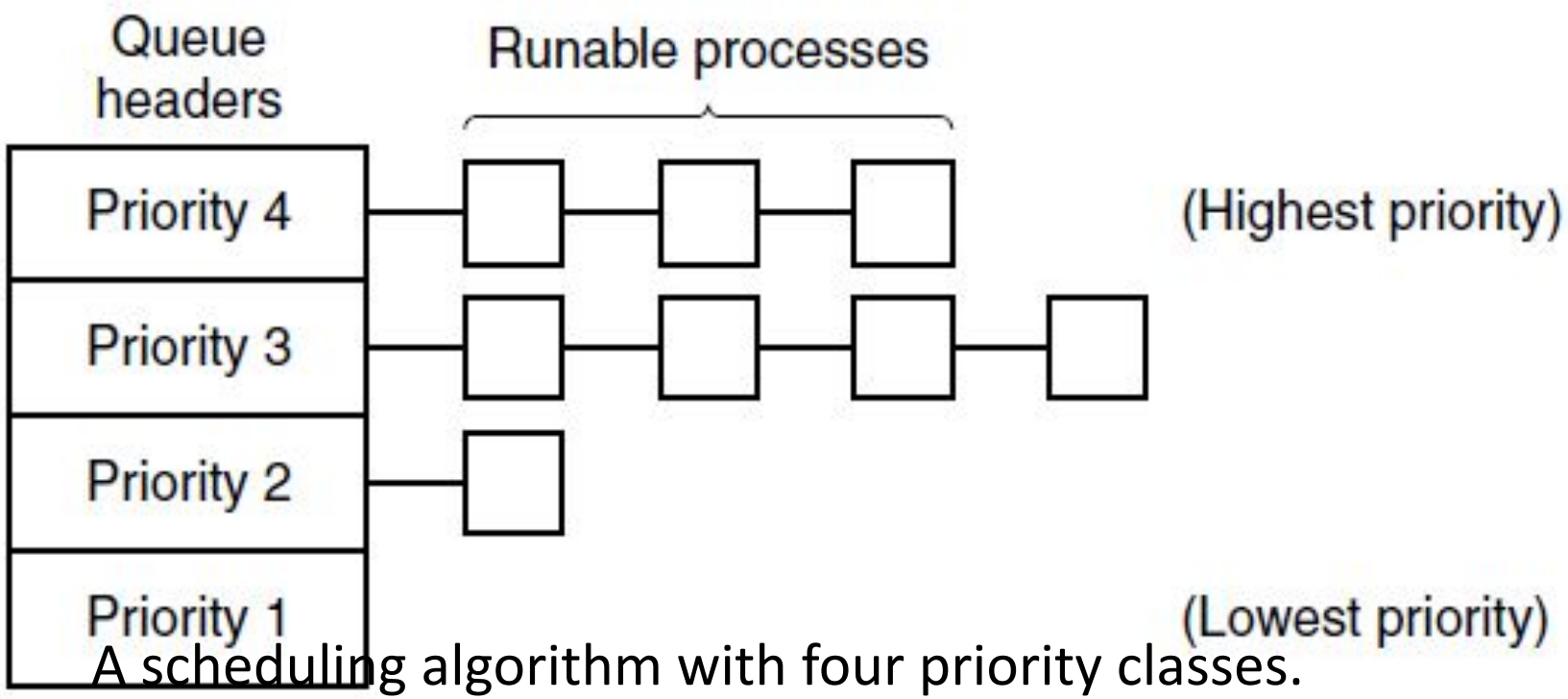
- Round-robin scheduling makes the implicit assumption that all processes are equally important
- The basic idea is straightforward: each process is assigned a priority, and the runnable process with the highest priority is allowed to run
- To prevent high-priority processes from running indefinitely, the scheduler may decrease the priority of the currently running process at each clock tick (i.e., at each clock interrupt).
- Priorities can be assigned to processes statically or dynamically

PRIORITY SCHEDULING

- Priorities can also be assigned dynamically by the system to achieve certain system goals.
- For example, some processes are highly I/O bound and spend most of their time waiting for I/O to complete. Whenever such a process wants the CPU, it should be given the CPU immediately, to let it start its next I/O request, which can then proceed in parallel with another process actually computing.
- It is often convenient to group processes into priority classes and use priority scheduling among the classes but round-robin scheduling within each class

PRIORITY SCHEDULING

- Round-robin scheduling makes the implicit assumption that all processes are equally important



A scheduling algorithm with four priority classes.

GUARANTEED SCHEDULING

- A completely different approach to scheduling is to make real promises to the users about performance and then live up to those promises.
- One promise that is realistic to make and easy to live up to is this: If n users are logged in while you are working, you will receive about 1/n of the CPU power. Similarly, on a single-user system with n processes running, all things being equal, each one should get 1/n of the CPU cycles.
- To make good on this promise, the system must keep track of how much CPU each process has had since its creation. It then computes the amount of CPU each one is entitled to, namely the time since creation divided by n.

GUARANTEED SCHEDULING

- Since the amount of CPU time each process has actually had is also known, it is fairly straightforward to compute the ratio of actual CPU time consumed to CPU time entitled.
- A ratio of 0.5 means that a process has only had half of what it should have had, and a ratio of 2.0 means that a process has had twice as much as it was entitled to. The algorithm is then to run the process with the lowest ratio until its ratio has moved above that of its closest competitor. Then that one is chosen to run next.

LOTTERY SCHEDULING

- The basic idea is to give processes lottery tickets for various system resources, such as CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the resource.
- When applied to CPU scheduling, the system might hold a lottery 50 times a second, with each winner getting 20 msec of CPU time as a prize.
- Lottery scheduling has several interesting properties.
- For example, if a new process shows up and is granted some tickets, at the very next lottery it will have a chance of winning in proportion to the number of tickets it holds. In other words, lottery scheduling is highly responsive.

LOTTERY SCHEDULING

- Cooperating processes may exchange tickets if they wish. For example, when a client process sends a message to a server process and then blocks, it may give all of its tickets to the server, to increase the chance of the server running next.
- When the server is finished, it returns the tickets so that the client can run again. In fact, in the absence of clients, servers need no tickets at all.
- Lottery scheduling can be used to solve problems that are difficult to handle with other methods. One example is a video server in which several processes are feeding video streams to their clients, but at different frame rates.
- Suppose that the processes need frames at 10, 20, and 25 frames/sec. By allocating these processes 10, 20, and 25 tickets, respectively, they will automatically divide the CPU in approximately the correct proportion, that is, 10 : 20 : 25

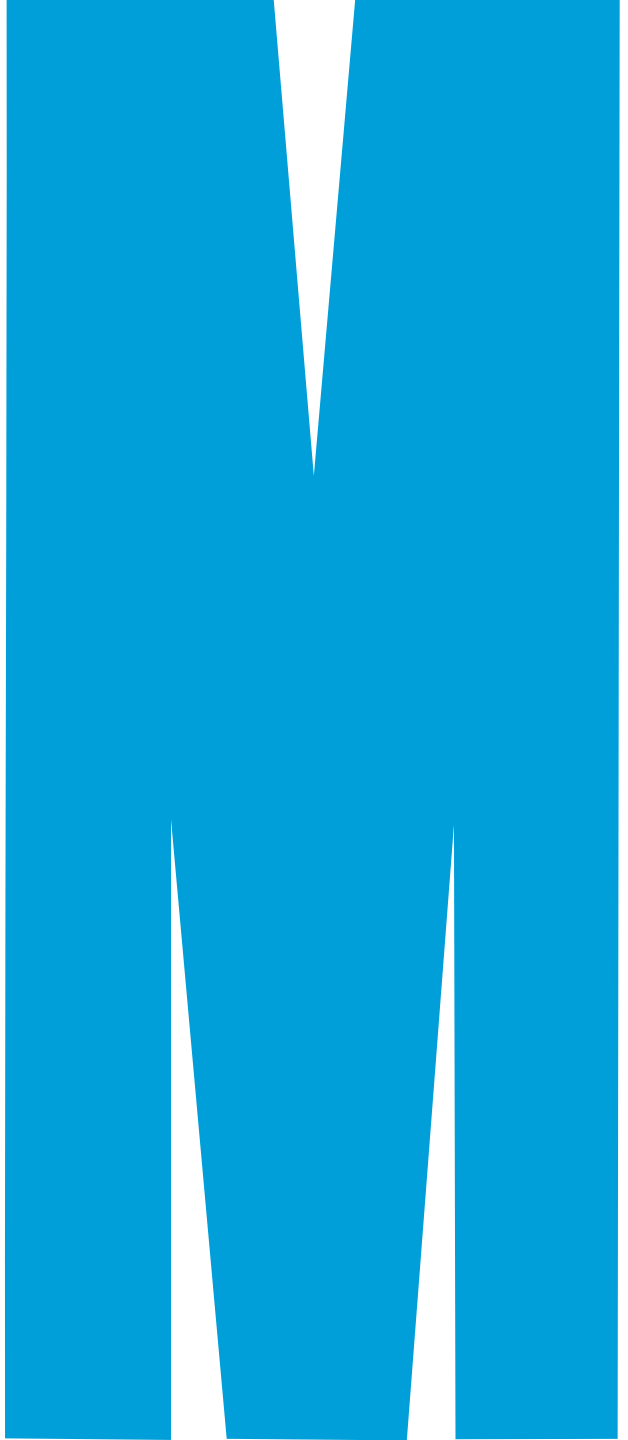
FAIR-SHARE SCHEDULING

- So far we have assumed that each process is scheduled on its own, without regard to who its owner is. As a result, if user 1 starts up nine processes and user 2 starts up one process, with round robin or equal priorities, user 1 will get 90% of the CPU and user 2 only 10% of it.
- To prevent this situation, some systems take into account which user owns a process before scheduling it. In this model, each user is allocated some fraction of the CPU and the scheduler picks processes in such a way as to enforce it.
- Thus if two users have each been promised 50% of the CPU, they will each get that, no matter how many processes they have in existence.

FAIR-SHARE SCHEDULING

- As an example, consider a system with two users, each of which has been promised 50% of the CPU. User 1 has four processes, A, B, C, and D, and user 2 has only one process, E. If round-robin scheduling is used, a possible scheduling sequence that meets all the constraints is this one:
- A E B E C E D E A E B E C E D E ...
- On the other hand, if user 1 is entitled to twice as much CPU time as user 2, we might get
- A B E C D E A B E C D E ...
- Numerous other possibilities exist, of course, and can be exploited, depending on what the notion of fairness is.

SCHEDULING ALGORITHM EXAMPLES



PROCESS SCHEDULING - EXAMPLE

METRICS

PROCESS	ARRIVAL TIME	SERVICE TIME
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

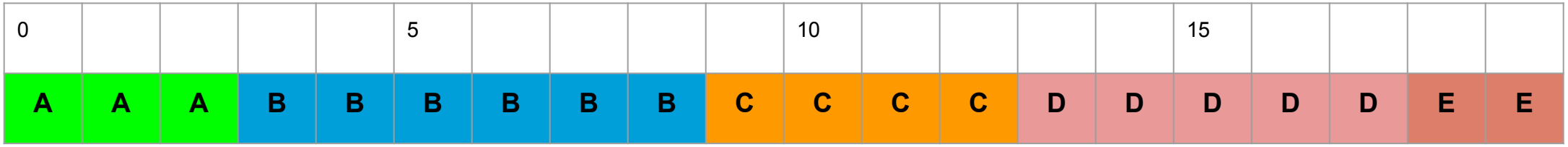
PROCESS SCHEDULING - EXAMPLE

FIRST COME FIRST SERVED (FCFS)

PROCESS	ARRIVAL TIME	SERVICE TIME
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

PROCESS SCHEDULING - EXAMPLE

FIRST COME FIRST SERVED (FCFS)



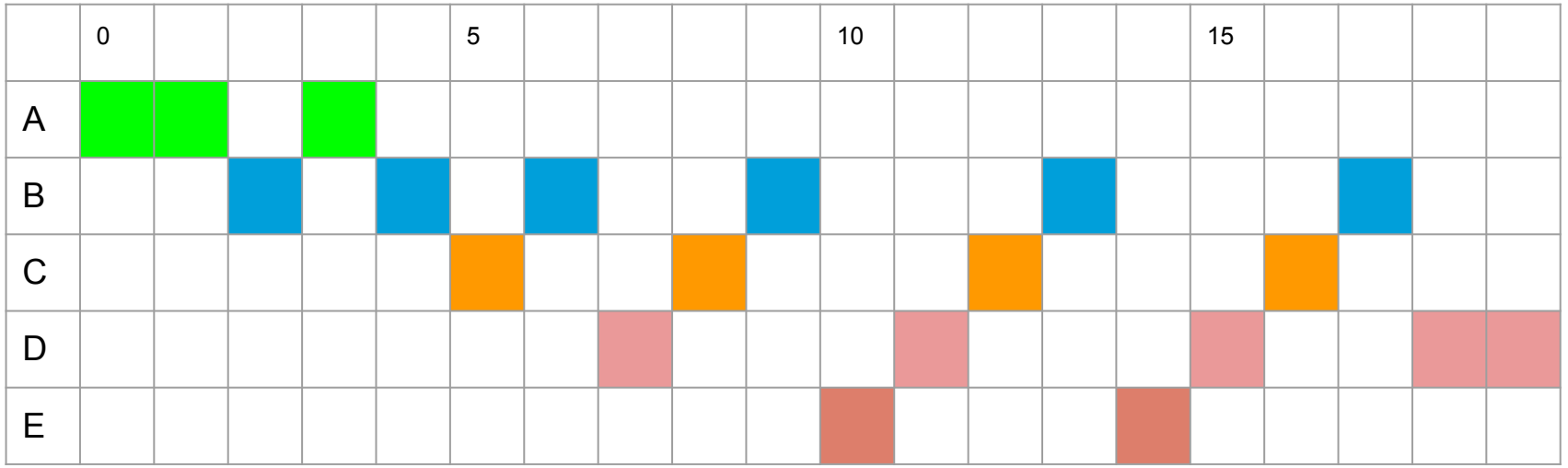
PROCESS	ARRIVAL TIME (Ta)	SERVICE TIME (Ts)	COMPLETION TIME (Tc)	TURNAROUND TIME (Tr)	Tr/Ts
A	0	3	3	3	1.00
B	2	6	9	7	1.17
C	4	4	13	9	2.25
D	6	5	18	12	2.40
E	8	2	20	12	6.00
Mean		4		8.6	2.56

PROCESS SCHEDULING - EXAMPLE

ROUND ROBIN (RR) - Q=1

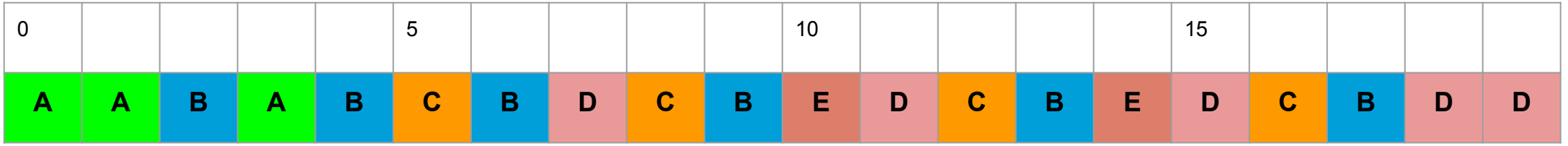
Time Quantum
q = 1

PROCESS	ARRIVAL TIME	SERVICE TIME
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



PROCESS SCHEDULING - EXAMPLE

ROUND ROBIN (RR) - Q=1



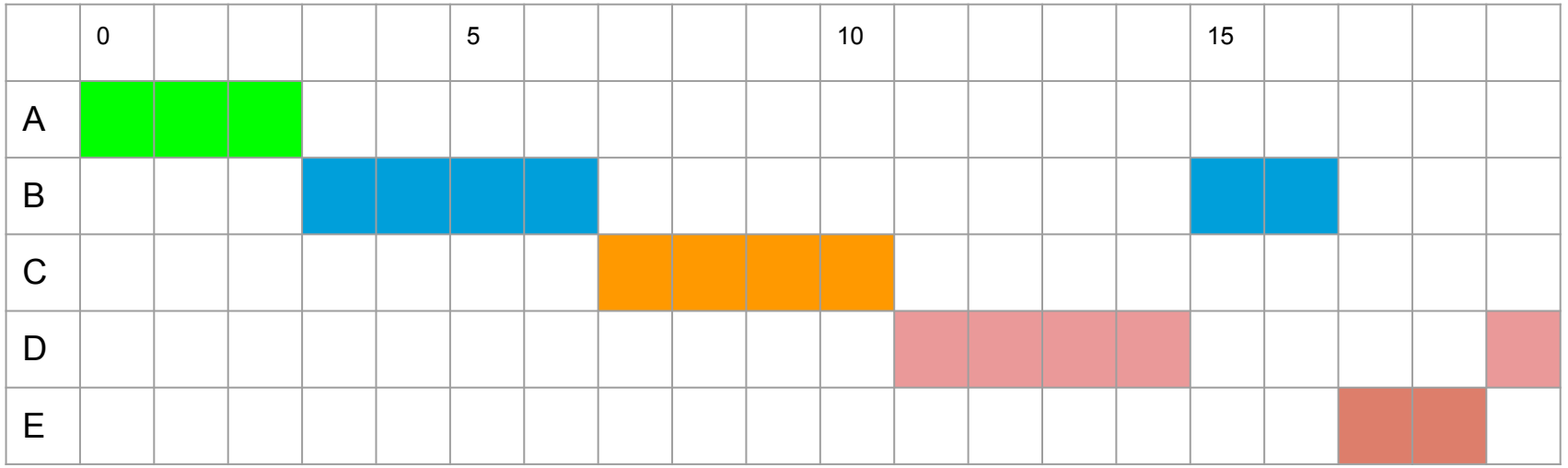
PROCESS	ARRIVAL TIME (Ta)	SERVICE TIME (Ts)	COMPLETION TIME (Tc)	TURNAROUND TIME (Tr)	Tr/Ts
A	0	3	4	4	1.33
B	2	6	18	12	2.67
C	4	4	17	13	3.25
D	6	5	20	14	2.80
E	8	2	15	7	3.50
Mean		4		10.80	2.71

PROCESS SCHEDULING - EXAMPLE

ROUND ROBIN (RR) - Q=4

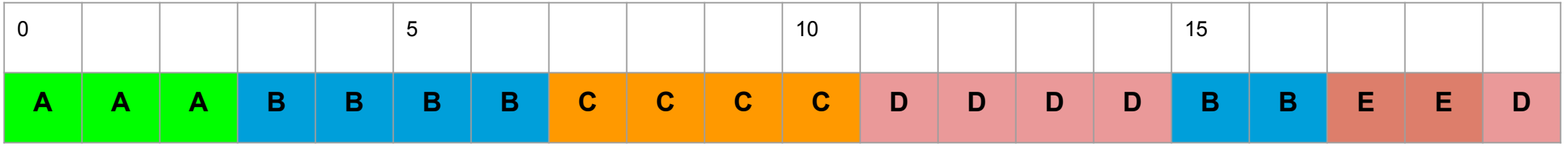
Time Quantum
q = 4

PROCESS	ARRIVAL TIME	SERVICE TIME
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



PROCESS SCHEDULING - EXAMPLE

ROUND ROBIN (RR) - Q=4

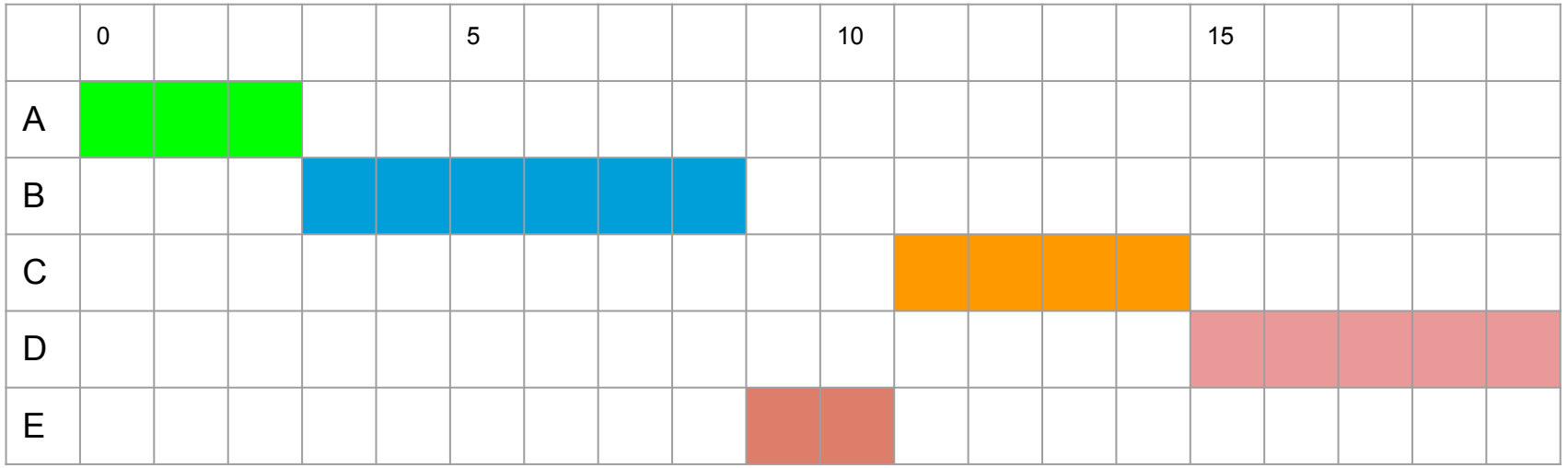


PROCESS	ARRIVAL TIME (Ta)	SERVICE TIME (Ts)	COMPLETION TIME (Tc)	TURNAROUND TIME (Tr)	Tr/Ts
A	0	3	3	3	1.00
B	2	6	17	15	2.50
C	4	4	11	7	1.75
D	6	5	20	14	2.80
E	8	2	19	11	5.50
Mean		4		10.00	2.71

PROCESS SCHEDULING - EXAMPLE

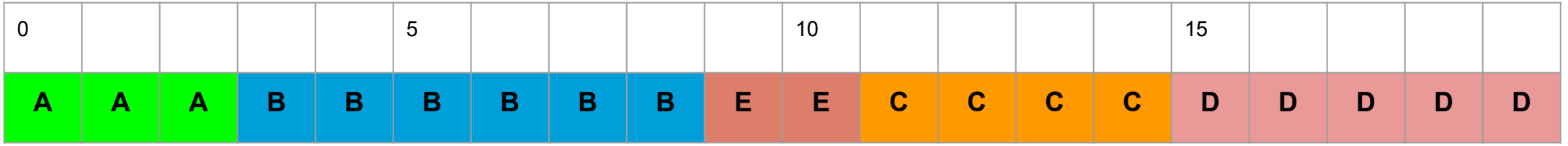
SHORTEST PROCESS NEXT (SPN)

PROCESS	ARRIVAL TIME	SERVICE TIME
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



PROCESS SCHEDULING - EXAMPLE

SHORTEST PROCESS NEXT (SPN)



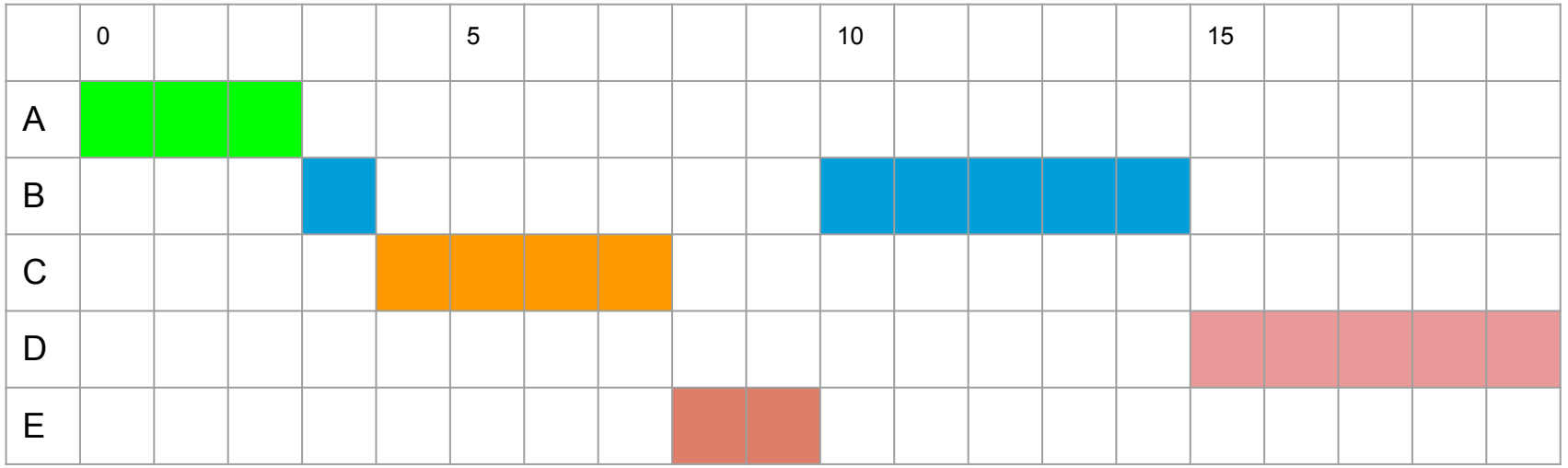
PROCESS	ARRIVAL TIME (Ta)	SERVICE TIME (Ts)	COMPLETION TIME (Tc)	TURNAROUND TIME (Tr)	Tr/Ts
A	0	3	3	3	1.00
B	2	6	9	7	1.17
C	4	4	15	11	2.75
D	6	5	20	14	2.80
E	8	2	11	3	1.50
Mean		4		7.6	1.84

PROCESS SCHEDULING - EXAMPLE

SHORTEST REMAINING TIME (SRT) - Q=1

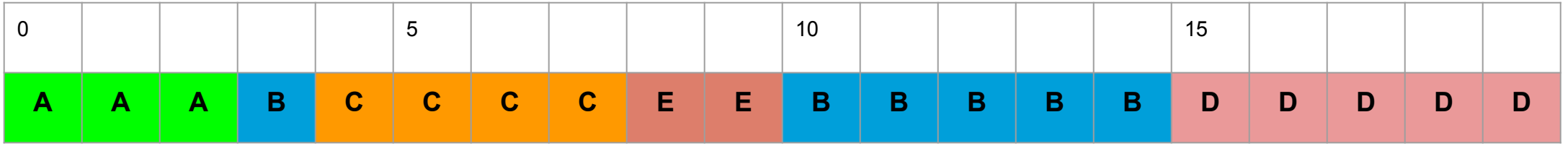
Time Quantum
q = 1

PROCESS	ARRIVAL TIME	SERVICE TIME
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



PROCESS SCHEDULING - EXAMPLE

SHORTEST REMAINING TIME (SRT) - Q=1



PROCESS	ARRIVAL TIME (Ta)	SERVICE TIME (Ts)	COMPLETION TIME (Tc)	TURNAROUND TIME (Tr)	Tr/Ts
A	0	3	3	3	1.00
B	2	6	15	13	2.17
C	4	4	8	4	1.00
D	6	5	20	14	2.80
E	8	2	10	2	1.00
Mean		4		7.2	1.59

PROCESS SCHEDULING - EXAMPLE

COMPARISON

ALGORITHM	MEAN TURNAROUND TIME	MEAN Tr/Ts
FCFS	8.6	2.56
RR q=1	10.80	2.71
RR q=4	10.00	2.71
SPN	7.6	1.84
SRT q=1	7.2	1.59

PROCESS SCHEDULING POLICIES

CHARACTERISTICS

ALGORITHM	FCFS	RR	SPN	SRT
Decision Mode	Non-preemptive	Preemptive	Non-preemptive	preemptive
Throughput	Not emphasized	May be low if q is too small	High	High
Response time	May be high	Good for short processes	Good for short processes	Good
Overhead	Less	Less	Can be high	Can be high
Fairness	Fair	Fair	Penalises long processes	Penalises long processes
Starvation	No	No	Possible	Possible

Summary

- So far we have discussed
  - Introduction to scheduling
  - Scheduling in
    - Batch Systems
    - Interactive Systems
    - Real-time Systems

- Reading
  - Tanenbaum, Chapter 2 (4th Edition)