Lecture 3

Process Scheduling

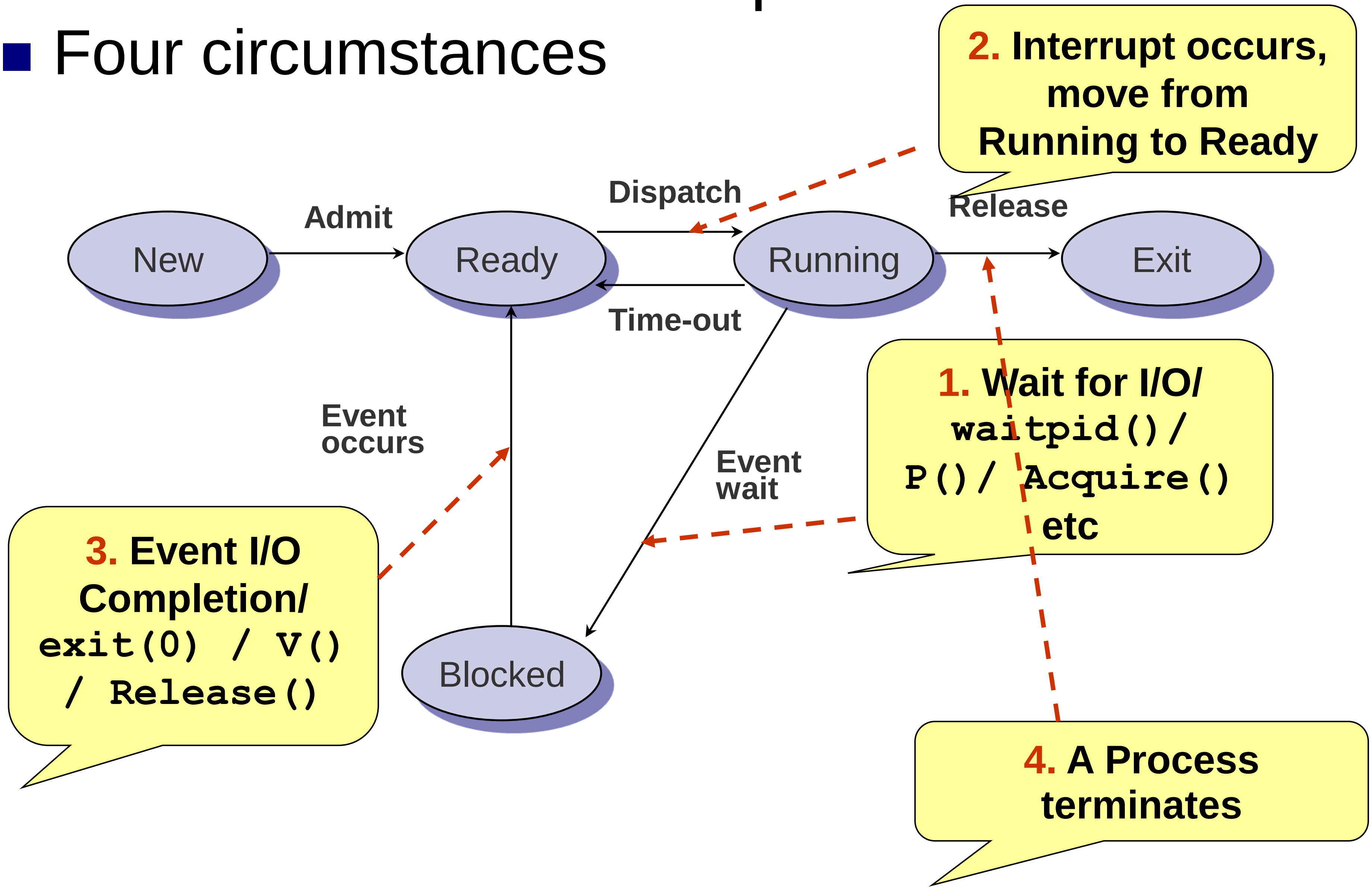
Operating Systems

Scheduling

- Short term scheduler (CPU Schedular)
  - Whenever the CPU becomes idle, a process must be selected for execution
  - The Process is selected from the Ready queue
- Ready queue is not necessarily a FIFO queue
- It can be
  - Priority based
  - A Tree
  - Unordered linked list etc

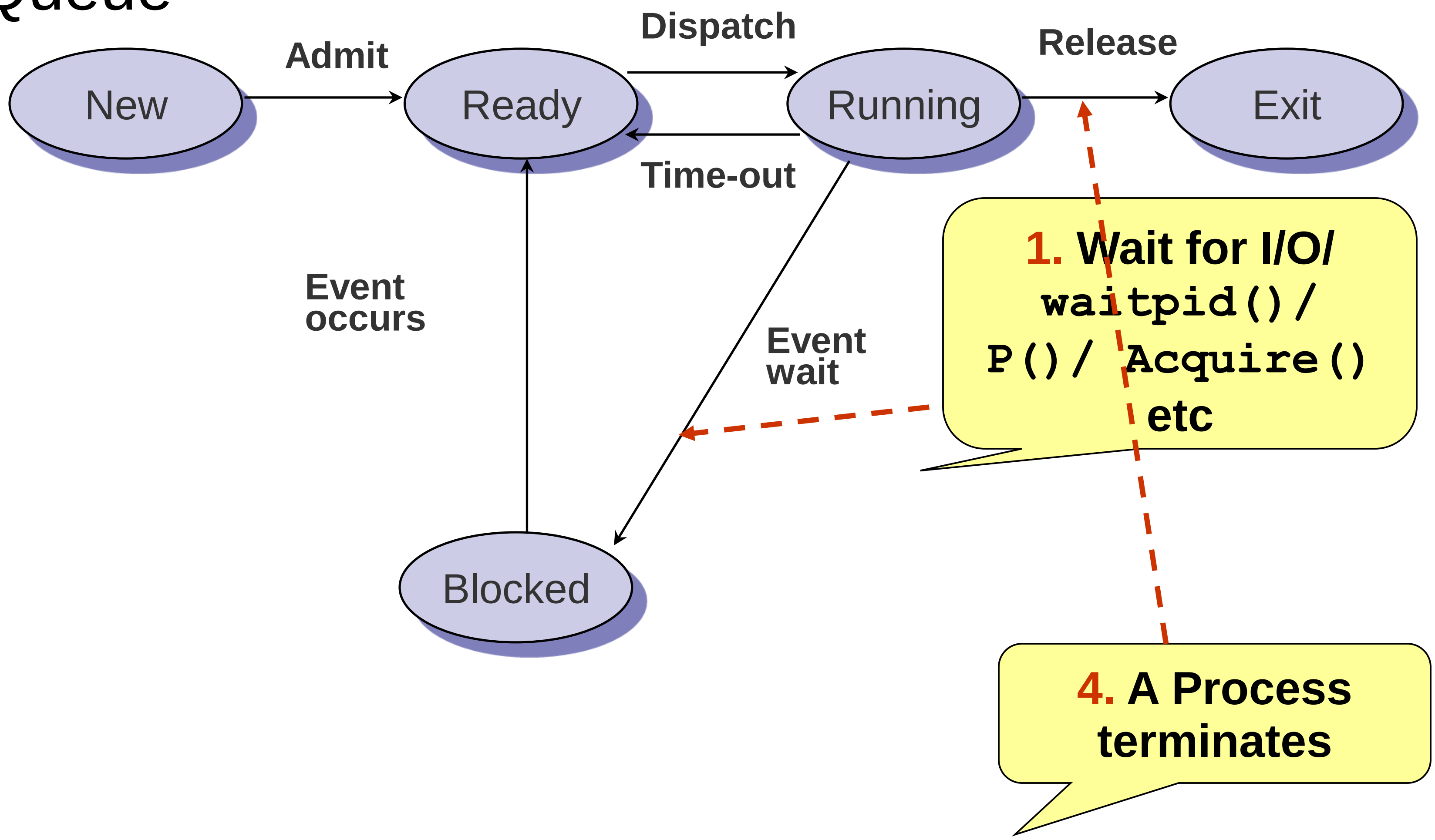
When to select a new process to Run

- Four circumstances



Non Preemptive Scheduling

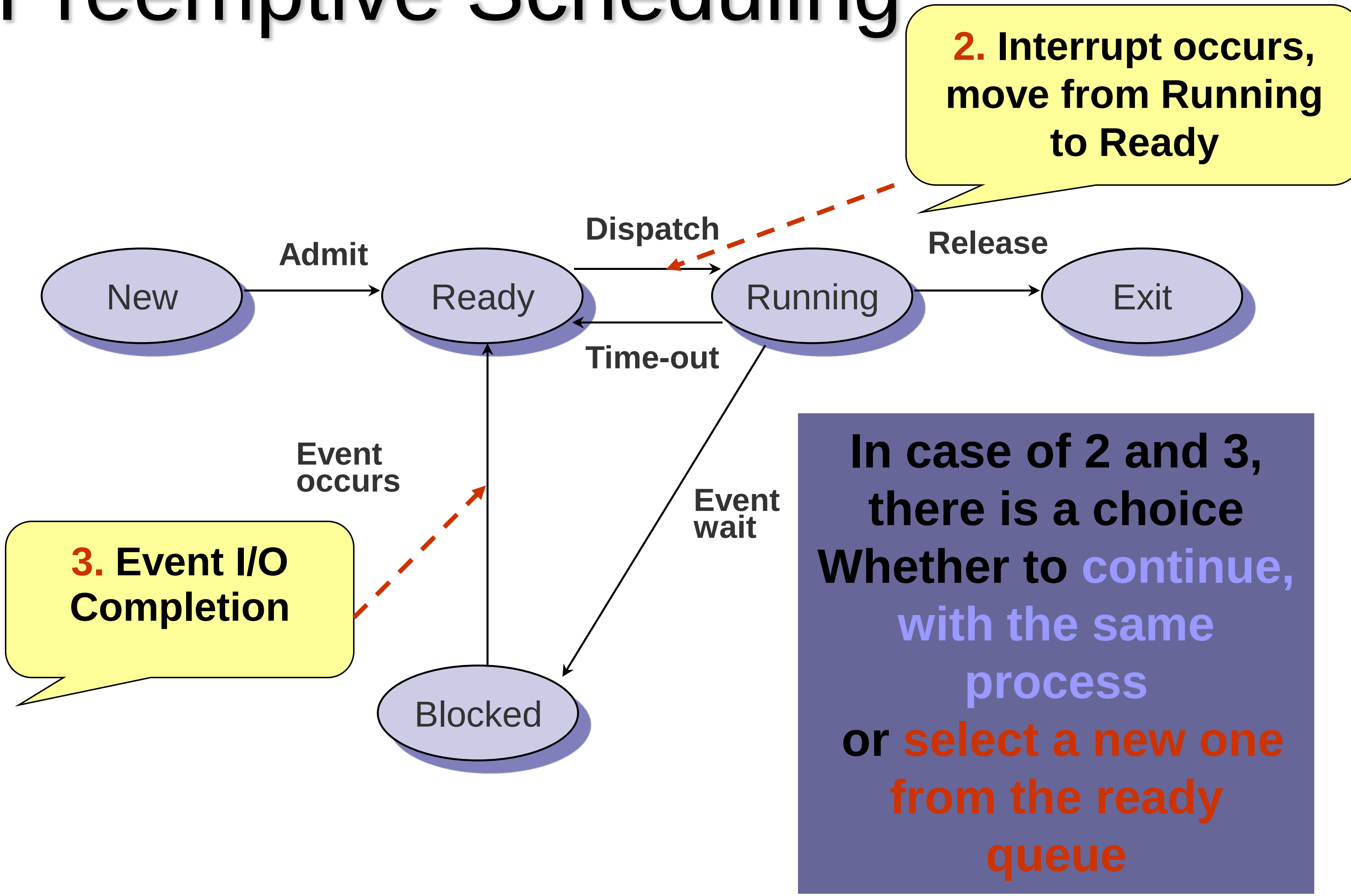
- Only the case 1 and 4
- Must select a new process, if any, from the Ready Queue



Non Preemptive Scheduling

- Once the CPU has been allocated to a process
- The process keeps it until
  - It Terminates
  - Or has to wait for:
    - I/O
    - Mutex
    - Child process
    - Semaphore
    - Conditional Variables etc
- There is no way, to get the CPU back, FORCEFULLY

Preemptive Scheduling



Scheduling Issues

- Fairness
  - Don’t starve process
- Priorities
  - Most important first
- Deadlines
  - Task X must be done by time *t*
- Optimization
  - Throughput, response time
- Reality - No universal scheduling policy
- Many models

Optimization Criteria

- CPU Utilization
  - Keep the CPU as busy as is possible
  - May range from 0% to 100%
- Throughput
  - Number of processes completed per unit time
  - E.g. long processes
    - 1 process / hr
  - Short processes
    - 10 processes / hr

Optimization Criteria

- Turnaround Time
  - How long it take to execute a Process
  - **`Turnaround = Completion_Time`**

    `− Submission_Time`
  - **`Turnaround =`** $\texttt{Wait\_Time}_{\texttt{GetIntoMemory}}$

    `+` $\texttt{Wait\_Time}_{\texttt{ReadyQueue}}$

    `+` $\texttt{Wait\_Time}_{\texttt{BlockQueue}}$

    `+ CPU_Execution_Time`

Optimization Criteria

- Scheduling Algorithm does not effect the waiting time in Block Queue
- It only effect the Waiting Time in the Ready Queue
- Waiting Time
  - Sum of the periods spent waiting in the Ready Queue

Optimization Criteria

- Turnaround Time is not a good criteria for Interactive Systems
- A process may
  - Produce “Some” output
  - Computes new results, while previous results are output to the user
- Response Time
- **`Response_Time = First_Response_Start_Time`**
  - **`Submission_Time`**

Optimization Criteria - Summary

- We would like to Maximize
  - CPU Utilization
  - Throughput
- And Minimize
  - Turnaround Time
  - Waiting Time
  - Response Time

Scheduling Algorithms

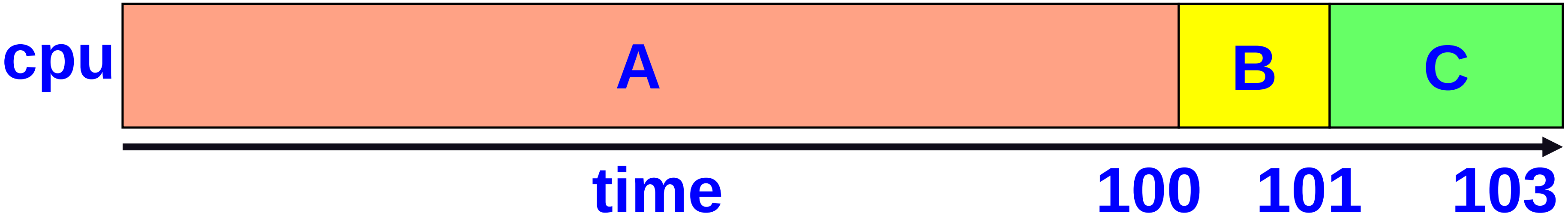
- First come, First serve
- Shortest Job First
- Priority Scheduling
- Round-Robin Scheduling
- Multi-level Queue Scheduling
- Multi-level Feed back queue Scheduling

First come, First serve

- Simplest scheduling algorithm:
  - Run jobs in order that they arrive
- Uni-programming:
  - Run until done
- Multi-programming:
  - Run until done or Blocks on I/O
- Non-preemptive
  - A Process keeps CPU until done or I/O
- Advantage:
  - Simplicity

First come, First serve

- Disadvantage
  - Wait time depends on arrival order
  - Unfair to later jobs
  - (worst case: long job arrives first)
- Three jobs (times: A=100, B=1, C=2) arrive in the order A, B, C

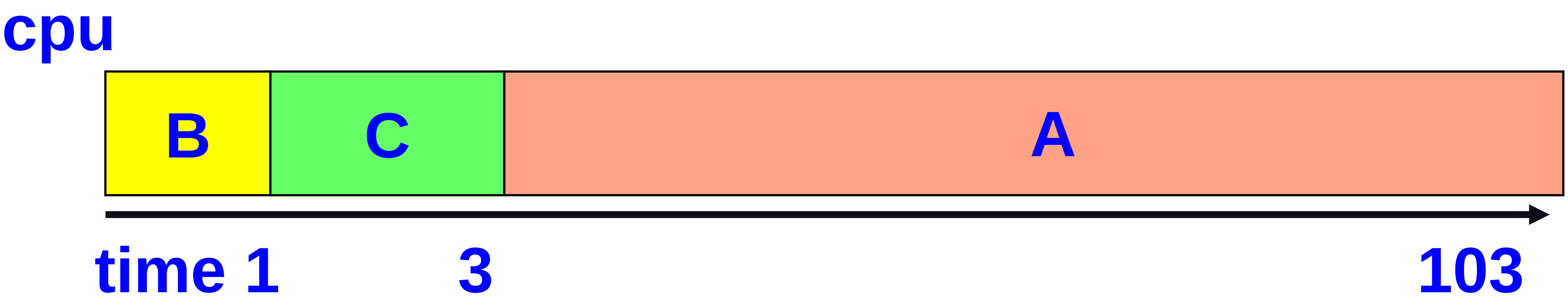


Average Waiting Time $= (0 + 100 + 101) / 3$

$= 67$

First come, First serve

- Now if they arrive in the order B, C, A



Average Waiting Time $= (0 + 1 + 3) / 3$

$= 1.33$

FCFS Convoy effect

- A CPU bound job will hold CPU until
  - Terminates
  - Or it causes an I/O burst
    - Rare occurrence, since the thread is CPU-bound
- Long periods where no I/O requests issued, and CPU held
- Result:
  - Poor I/O device utilization

FCFS Convoy effect : Example

- One CPU bound job, many I/O bound
- CPU bound runs
  - I/O jobs blocked in ready queue
  - I/O devices idle
- CPU bound blocks
  - I/O bound job(s) run, quickly block on I/O
- CPU bound runs again
- I/O of the I/O bound jobs completes
- CPU bound still runs while I/O devices idle (continues…)

Round robin (RR)

- Solution to job monopolizing CPU?
- Interrupt it.
  - Run job for some “time slice,”
  - When time is up, or it blocks
  - It moves to back of a FIFO queue
- Advantage:
  - Fair allocation of CPU across jobs
  - Low average waiting time when job lengths vary

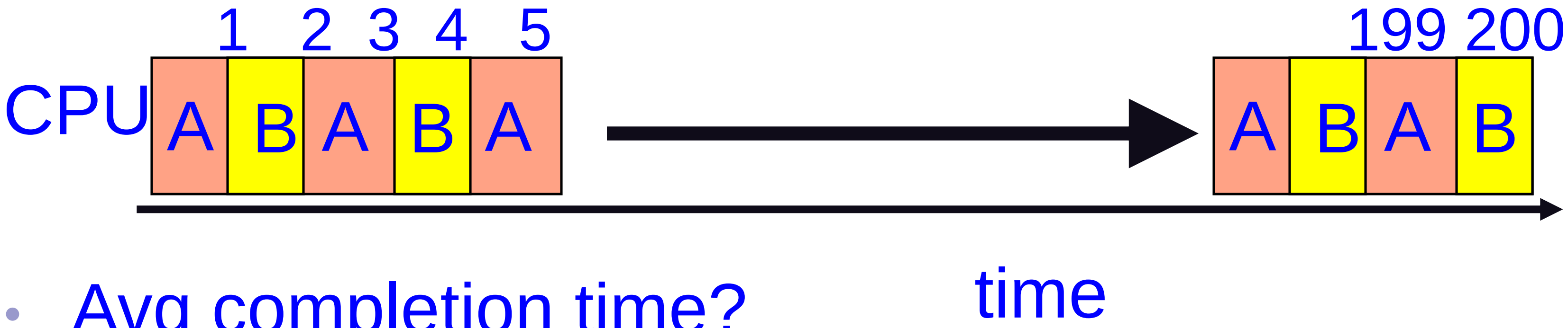
Round robin (RR)

= (103 + 2 + 5) / 3

What is avg completion time?

- Good for Varying sized jobs
- But what about same-sized jobs?
- Assume 2 jobs of time =100 each:

Round Robin's Disadvantage



- Avg completion time?
- (200 + 200) / 2 = 200
- How does this compare with FCFS for same two jobs?
- (100 + 200) / 2 = 150

RR Time slice tradeoffs

- Performance depends on length of the timeslice
- Context switching isn’t a free operation.
- If timeslice time is set too high (attempting to amortize context switch cost)
  - You get FCFS.
  - i.e. Processes will finish or block before their slice is up anyway
- If it’s set too low you’re spending all of your time context switching between threads.