Interlude: Quick Review

Interlude: Quick Review

Review: Task Scheduling

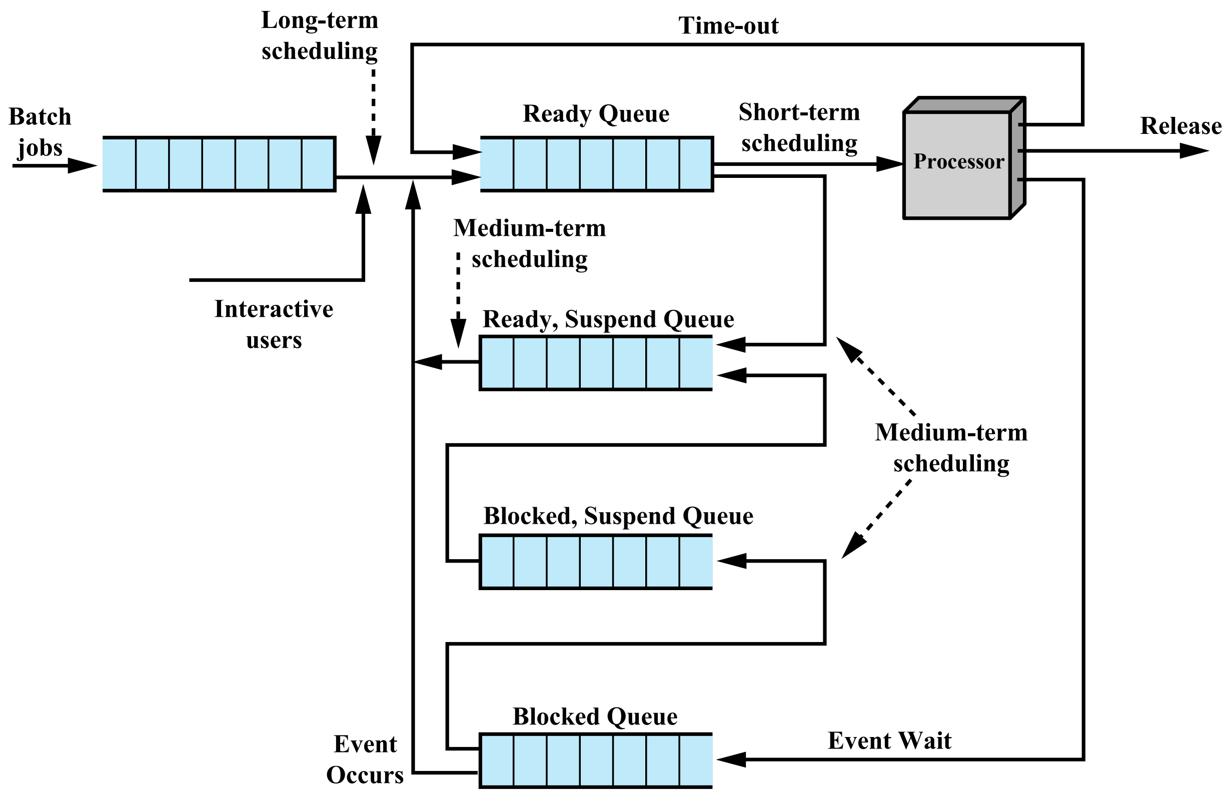


Figure 9.3 Queuing Diagram for Scheduling

Review: Bad Assumptions

All tasks:

1. ~~Run for the same amount of time~~ FCFS
2. ~~Arrive at the same time (roughly)~~ SPN shortest process next
3. ~~Once started, run to completion~~ SRT shortest remaining time
4. ~~Only use the CPU~~ I/O
5. ~~Have a known run-time.~~ Exp Averaging

Part 5: Feedback Queues

"Multi-Level Feedback Queue" (MLFQ) to be precise

lower → bigger #
higher → smaller #

The Algorithm (Part 1)

1. don't estimate runtime

* smaller is more important

2. schedule preemptively, using mutiple ready queues $RQ_0 \rightarrow RQ_N$

   a) Priority(task) = N − RQ # 0 → highest priority

   b) Priority(A) > Priority(B) ⟶ schedule A

   c) Priority(A) == Priority(B) ⟶ Schedule using RR time slice

   d) all tasks enter into RQ 0

   e) on preemption, move the running task down a RQ

   f)

   g)

Visual

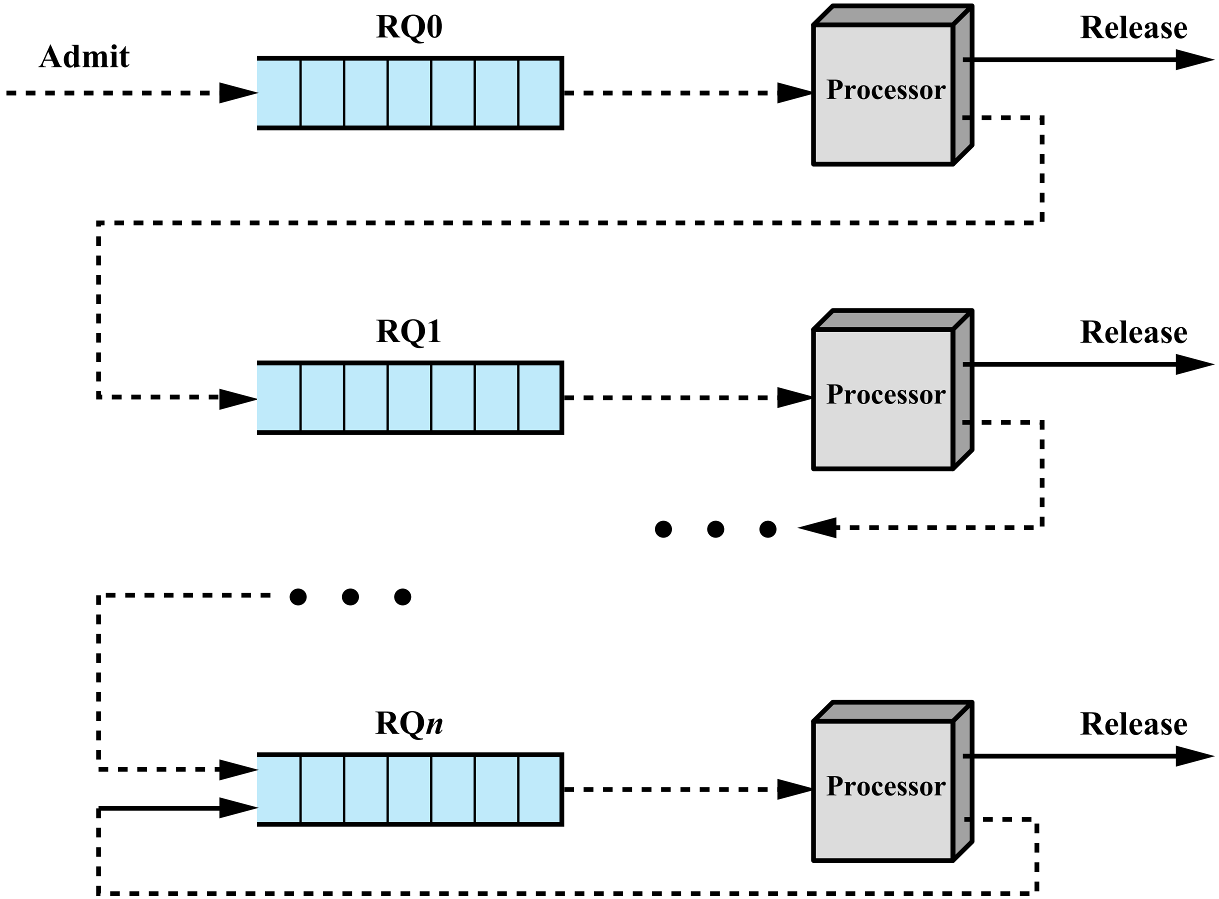


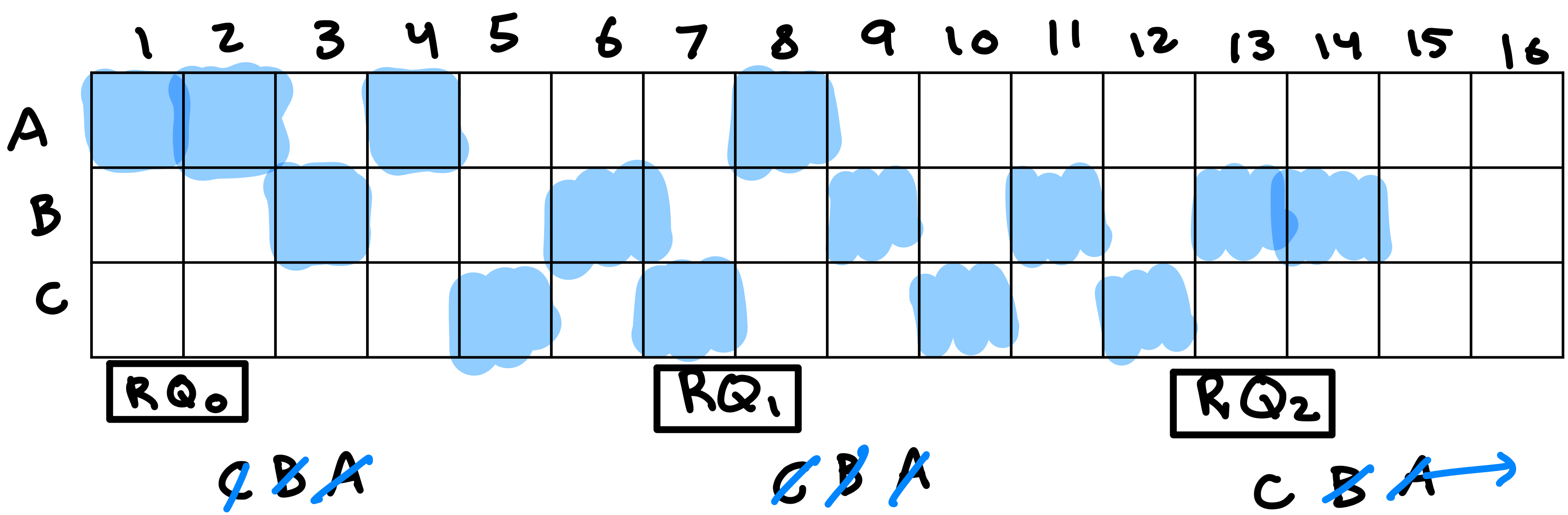
Figure 9.10 Feedback Scheduling

Simple Example

Task	Arrival Time	Service Time
A	$0 - \varepsilon$	4
B	$2 - \varepsilon$	6
C	$4 - \varepsilon$	4

$q = 1$

3 RQs

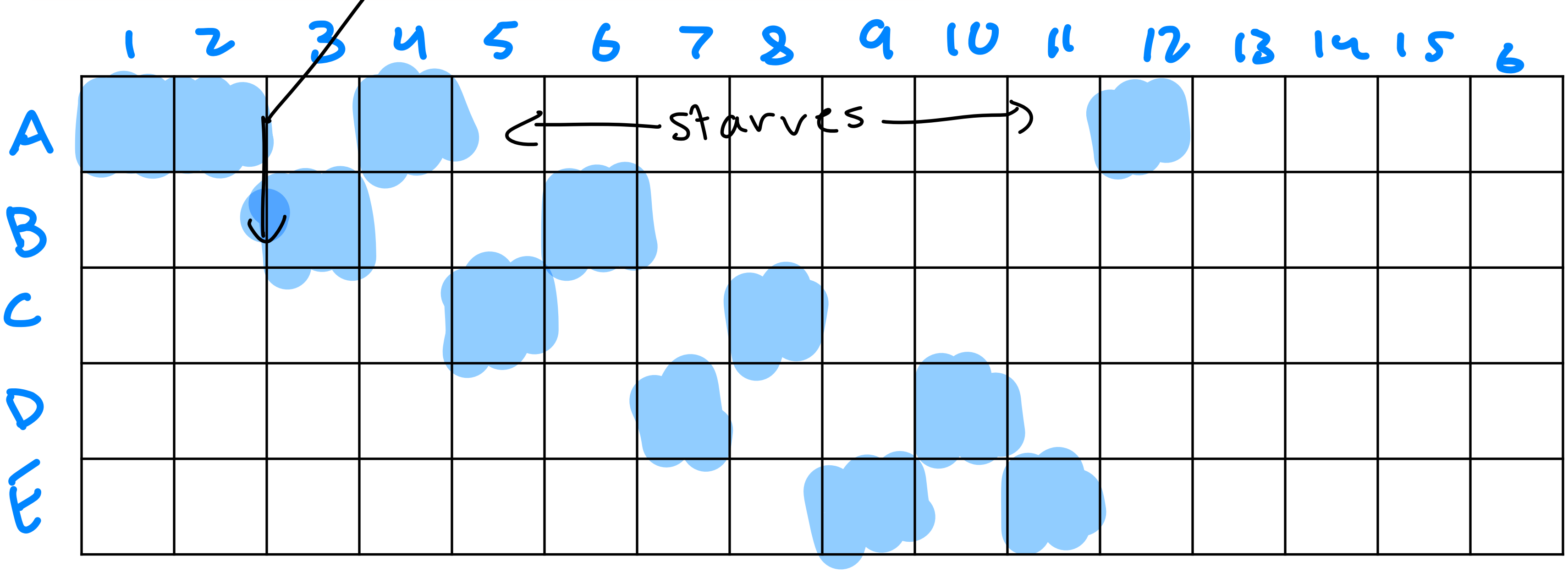


…what happens to long(er) tasks?

Task	Arrival Time	Service Time
A	$0 - \varepsilon$	4
B	$2 - \varepsilon$	2
C	$4 - \varepsilon$	2
D	$6 - \varepsilon$	2
E	$8 - \varepsilon$	2

a goes from 2 → 1

a starves and gets bad turnaround time



The Algorithm (Part 2)

1. Don't estimate runtime
2. Schedule *preemptively* using *multiple* ready queues
   a) Priority(task) = n – RQ #
   b) Priority(A) > Priority(B) → schedule A
   c) Priority(A) = Priority(B) → schedule in RR
   d) Tasks start in queue 0
   e) On pre-emption, tasks move *down* a queue
   f) increase the length of a time slice for lower queues
   g)

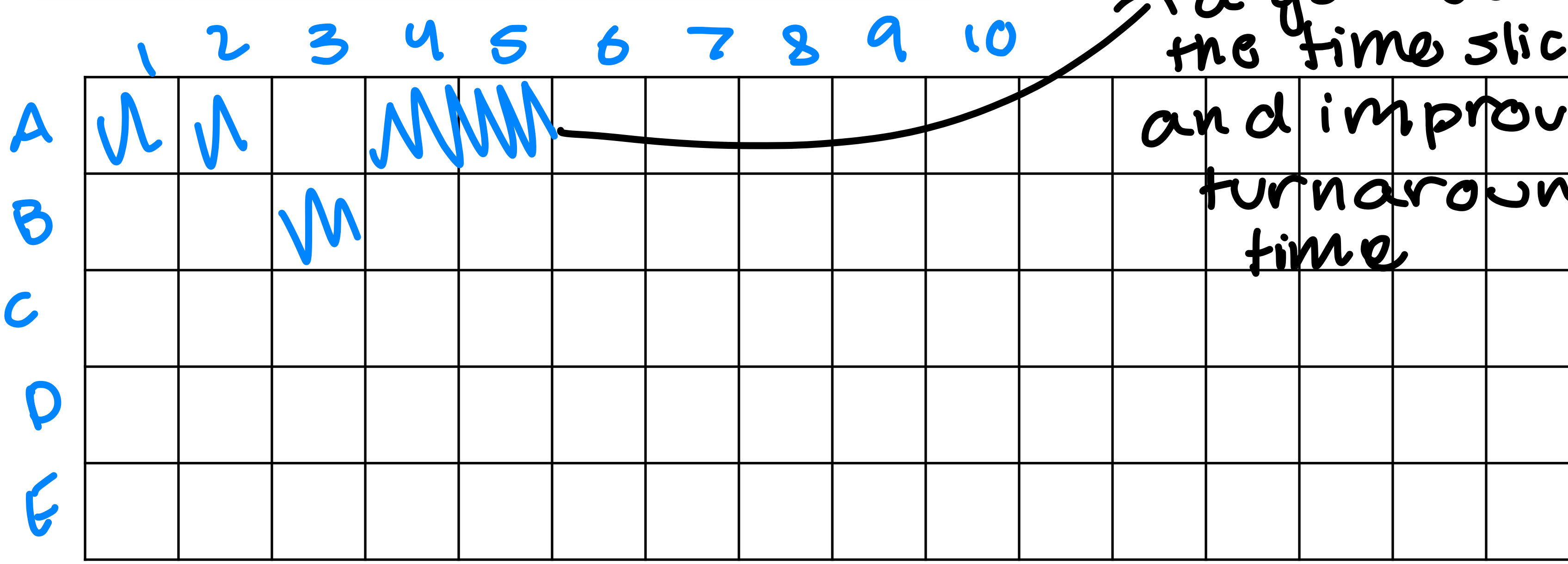
each lower q gets 2x the ts length

$q = 2^i$

Redo

Task	Arrival Time	Service Time
A	$0 - \varepsilon$	4
B	$2 - \varepsilon$	2
C	$4 - \varepsilon$	2
D	$6 - \varepsilon$	2
E	$8 - \varepsilon$	2

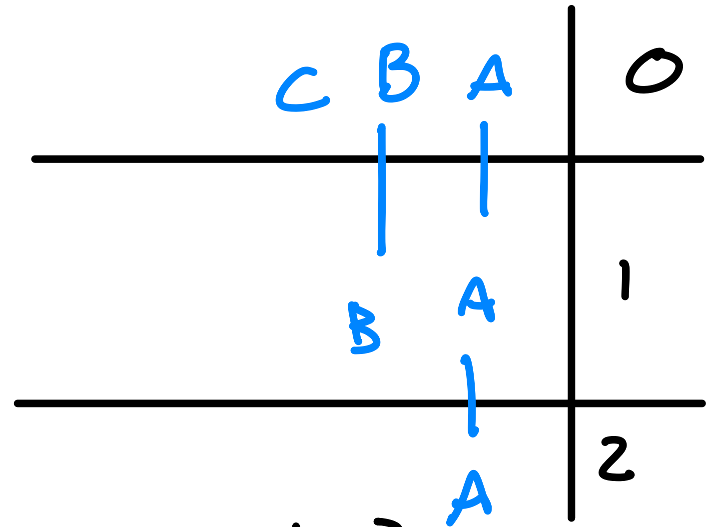
B A 0
B A 1
2

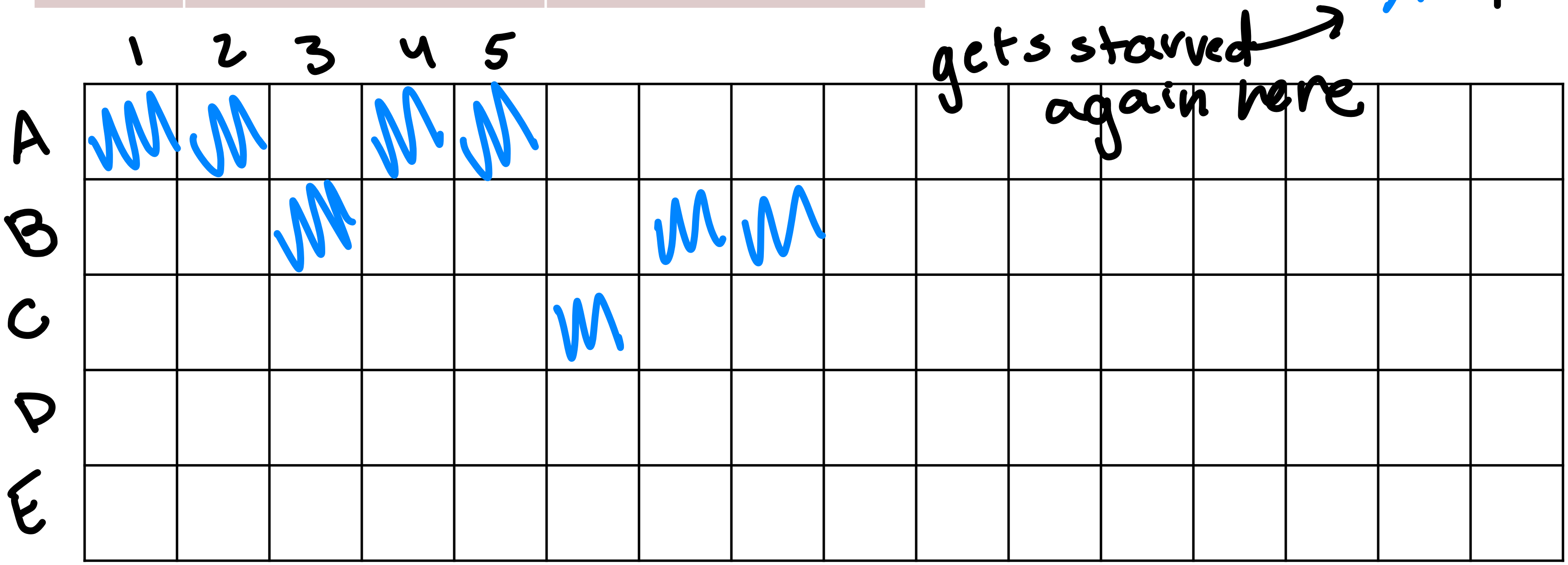


a gets double the time slice and improved turnaround time

What about even *longer* tasks?

Task	Arrival Time	Service Time
A	$0 - \varepsilon$	5
B	$2 - \varepsilon$	2
C	$4 - \varepsilon$	2
D	$6 - \varepsilon$	2
E	$8 - \varepsilon$	2





The Algorithm (Part 3)

1. Don't estimate runtime
2. Schedule *preemptively* using *multiple* ready queues
   a) Priority(task) = n – RQ #
   b) Priority(A) > Priority(B) → schedule A
   c) Priority(A) = Priority(B) → schedule in RR
   d) Tasks start in queue 0
   e) On pre-emption, tasks move *down* a queue
   f) Longer time-slice for lower queues, e.g., $q = 2^i$
   g) after a task has waited for S consecutive time units, promote to RQ_0

Analysis

- Does not require future knowledge
  → only based on the past
- "May favor I/O bound tasks"
  → since only get demoted on pre-emption, not willfully giving up CPU
  Fix: demote at the moment a proc has run for a total time slice
- A good balance between response time and turnaround time

⇒ MLFQs form the basis for most UNIX schedulers

Part 6: Wrapping Up

Adding Priority

Typically, four levels of priority:

1. System ← highest
2. Interactive
3. Normal
4. Batch

⇒ use priority queues

More Performance Metrics

Throughput: The # of tasks completed per unit of time

EXIT

Even more… (see book)

- Task deadlines
- Predictability
- Processor utilization
- Enforcing priorities
- Balancing resources
- Overhead

Formal Analysis

- **E** : total execution time

how much time has the task gotten?

- **W** : total waiting time
- **S** : total service time

either predict the future
or
Require future knowledge

Selection function:

which task is *selected* to be scheduled

Summary of Scheduling Algorithms

be able to fill this out for final → answer in ch 9

	FCFS	RR Round Robin	SPN	SRT	HRRN	MLFQ
Selection Function	max(w)					
Preemptive?	X					
Starvation Possible?	X					
Response Time	poor					
Overhead	low					

med: 77
mean: 74

Midterm