Operating Systems!

Scheduling:
Uniprocessor Scheduling
(Part 2)

Prof. Travis Peters

Montana State University

CS 460 - Operating Systems

Fall 2020

https://www.cs.montana.edu/cs460

Some diagrams and notes used in this slide deck have been adapted from Sean Smith's OS courses @ Dartmouth. Thanks, Sean!

Today

- Announcements
  - ~~Yalnix~~ How are projects/proposals going?! ☺

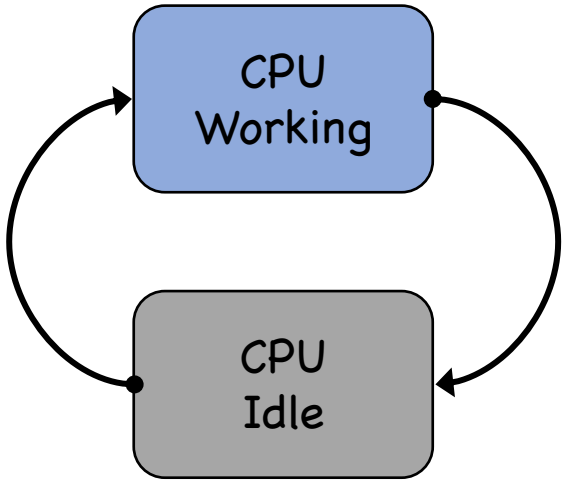
- Upcoming Deadlines
  - **Project Proposal (HARD DEADLINE) Sunday [10/25/2020] @ 11:59 PM (MST)**

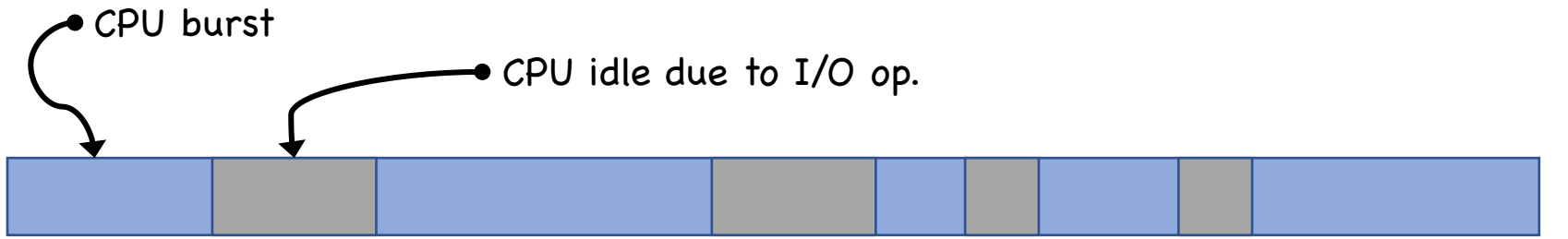
Yalnix Teams:
You should plan to be near completing checkpoint 3 at this point!

Non-Yalnix Teams:
Let me know if you want to discuss projects before submitting your proposal!

Today *(cont.)*

- Agenda
  - ~~Types of Processes~~
  - ~~Long-, medium-, and~~ ***short-term*** scheduling
  - Performance of different scheduling policies



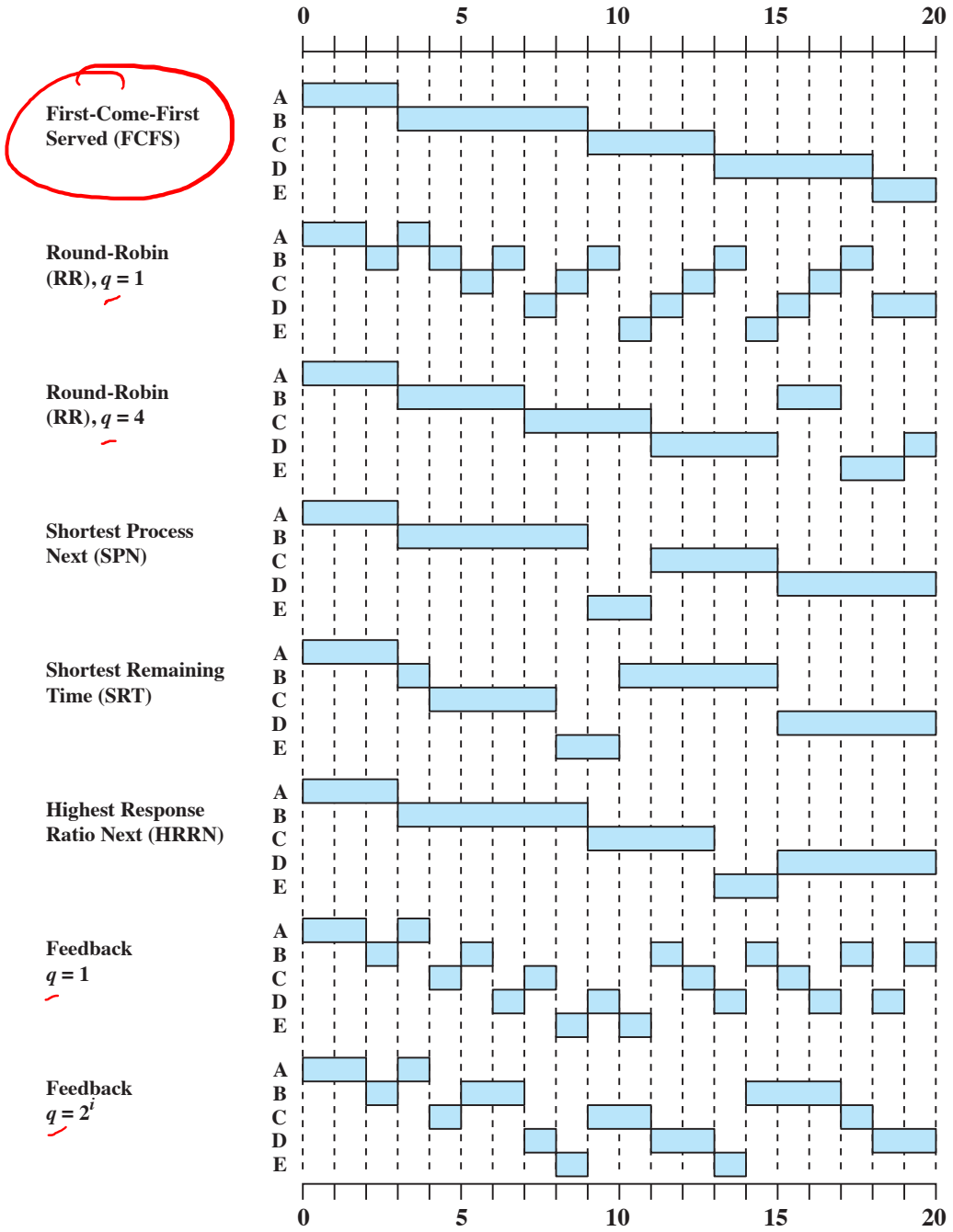


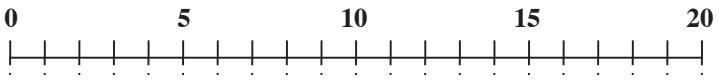
Adapted from https://www.cse.iitm.ac.in/~chester/courses/15o_os/slides/8_Scheduling.pdf

Comparing Scheduling Algorithms

An Example

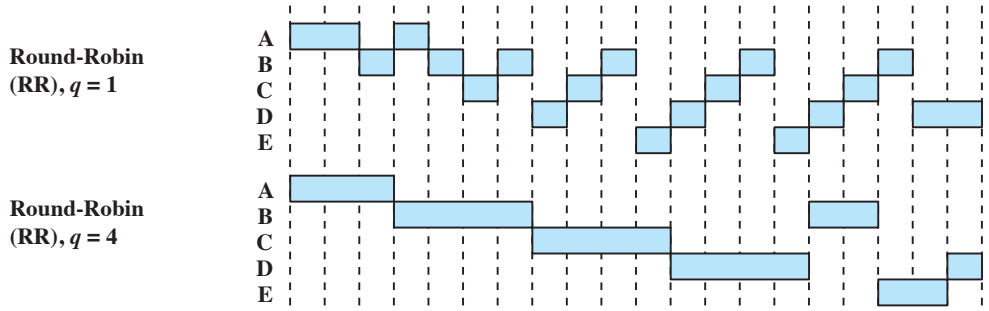
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2





Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (*Ts*)	3	6	4	5	2	Mean

Round Robin (RR)

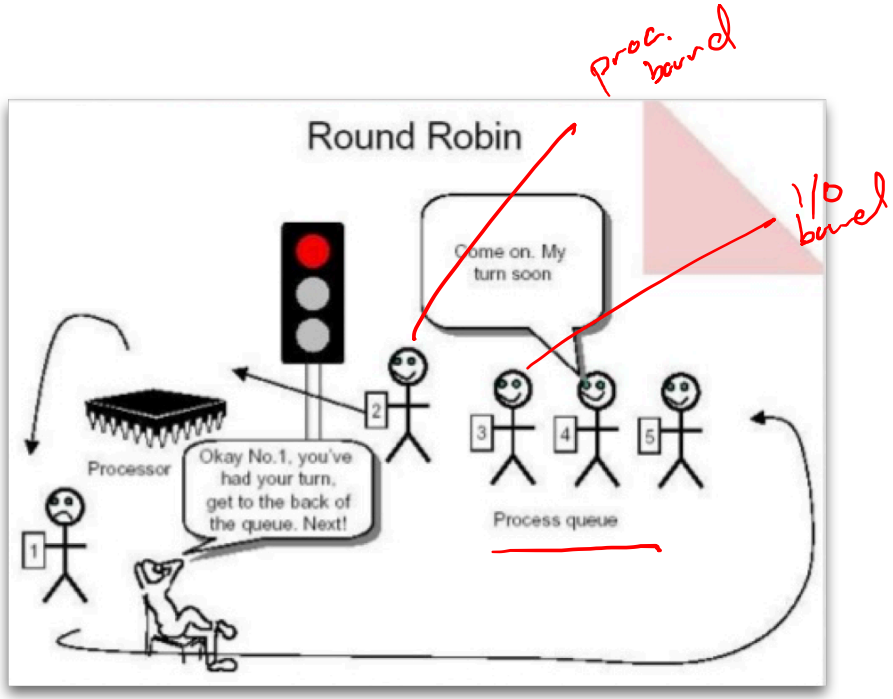


RR $q = 1$						
Finish Time	4	18	17	20	15	
Turnaround Time (*Tr*)	4	16	13	14	7	10.80
Tr/*Ts*	1.33	2.67	3.25	2.80	3.50	2.71
RR $q = 4$						
Finish Time	3	17	11	20	19	
Turnaround Time (*Tr*)	3	15	7	14	11	10.00
Tr/*Ts*	1.00	2.5	1.75	2.80	5.50	2.71

- Use preemption based on a clock!
  - clock-based interrupt (periodic intervals of execution)
  - preempted process goes back on the ready queue
  - "Time Slicing"

- Principle design choice: length of **time quanta (slice)**
  -> **Q:** Should we prefer **small slices** or **big slices**?

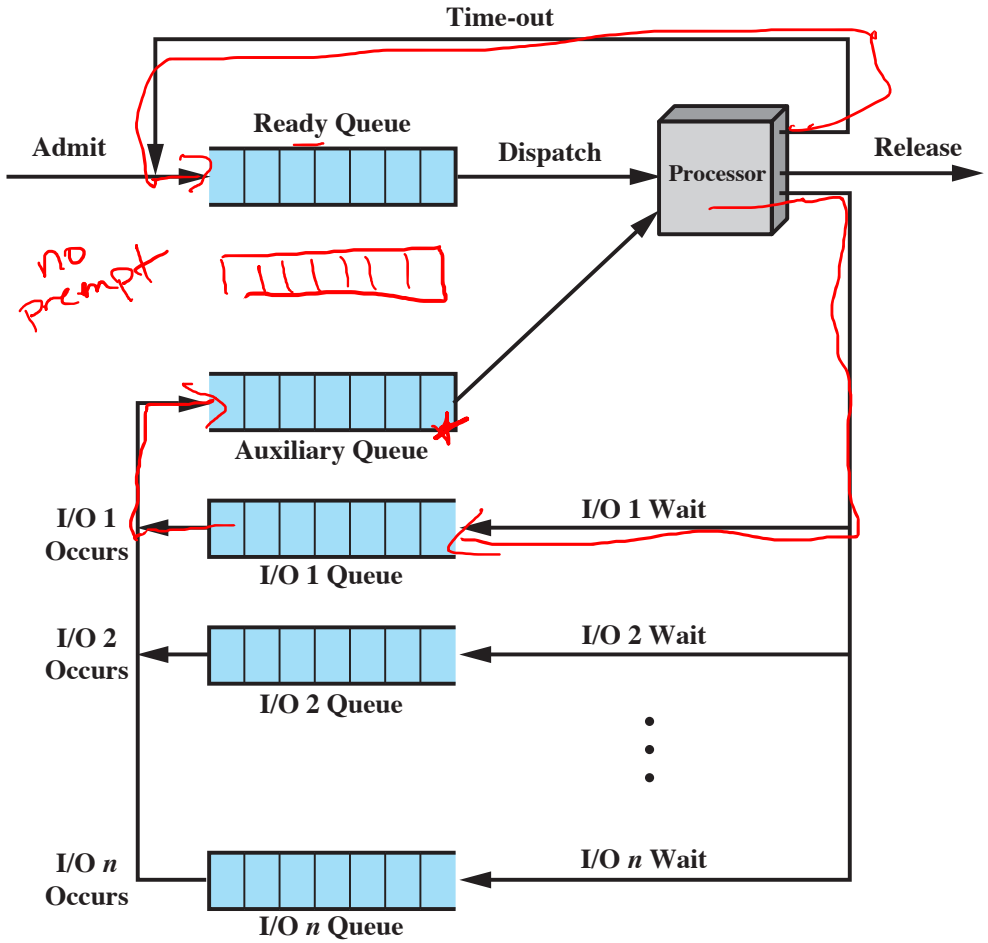
- **Q:** Any drawbacks?

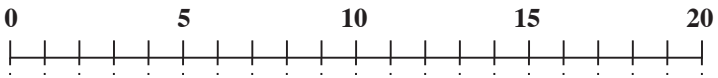


Refinement: *Virtual Round Robin (VRR)*

Same as RR… plus…

- Each process gets the opportunity to complete its time quanta
- Use **Auxiliary Queue (AQ)** to hold recently unblocked I/O-bound processes
- Processes in AQ get preference over processes in normal **Ready Queue (RQ)**

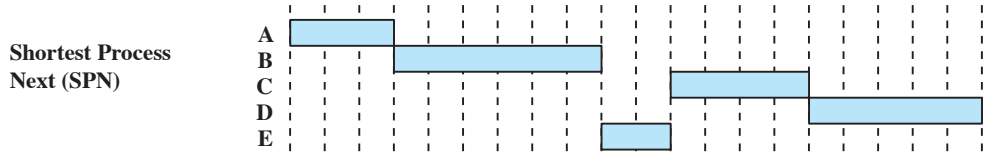




Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (*Ts*)	3	6	4	5	2	Mean

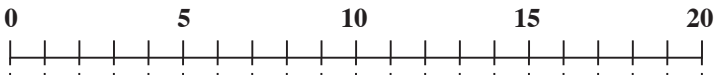
Shortest Process Next (SPN)

- No preemption allowed *-> processes run to completion*
- Process w/ shortest (expected) processing time is selected next *-> short processes can "cut the line"*
- Need to know (or estimate) the required processing time *-> How?*



	SPN					
Finish Time	3	9	15	20	11	
Turnaround Time (*Tr*)	3	7	11	14	3	7.60
Tr/*Ts*	1.00	1.17	2.75	2.80	1.50	1.84

- **Q:** Any issues with this approach? Longer procs could starve
- **Q:** Any undesirable characteristics? Escalate priority of procs that have been waiting a long time

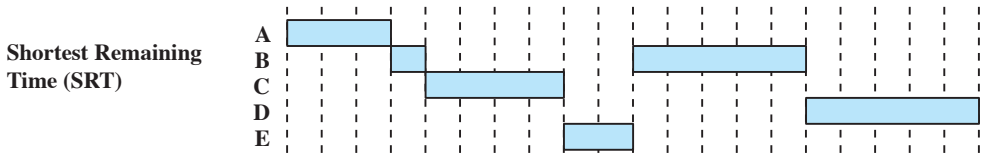


Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (*Ts*)	3	6	4	5	2	Mean

Shortest Remaining Time (SRT)

- Preemption allowed!
  -> processes can be interrupted

- Process w/ shortest (expected) ***remaining*** processing time is selected next
  -> processes w/ less remaining time can "cut the line" and preempt those w/ longer remaining times

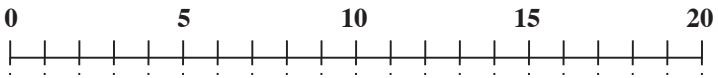


SRT						
Finish Time	3	15	8	20	10	
Turnaround Time (*Tr*)	3	13	4	14	2	7.20
Tr/*Ts*	1.00	2.17	1.00	2.80	1.00	1.59

long procs could starve

- **Q:** Advantages? Disadvantages?

↳ switch procs if proc a shorter arrives



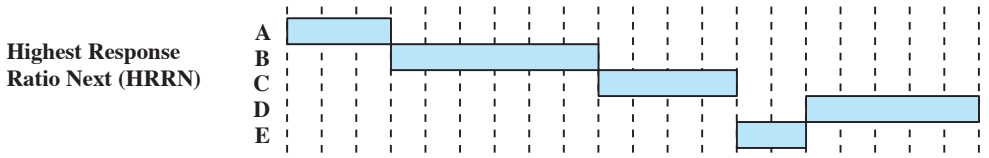
Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (*Ts*)	3	6	4	5	2	Mean

Highest Response Ratio Next (HRRN)

- When the current process blocks
  -> choose process w/ the greatest **normalized turnaround time (R)**

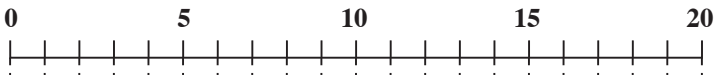
$$R = \frac{w + s}{s}$$

R = response ratio, w = time spent waiting, s = expected service time



HRRN						
Finish Time	3	9	13	20	15	
Turnaround Time (*Tr*)	3	7	9	14	7	8.00
Tr/*Ts*	1.00	1.17	2.25	2.80	3.5	2.14

- Accounts for the age of the process!
  -> shorter jobs are favored...
  -> ... but aging process without service increases the ratio,
  so a longer process will eventually get past competing shorter jobs.



Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (*Ts*)	3	6	4	5	2	Mean

Multilevel Feedback

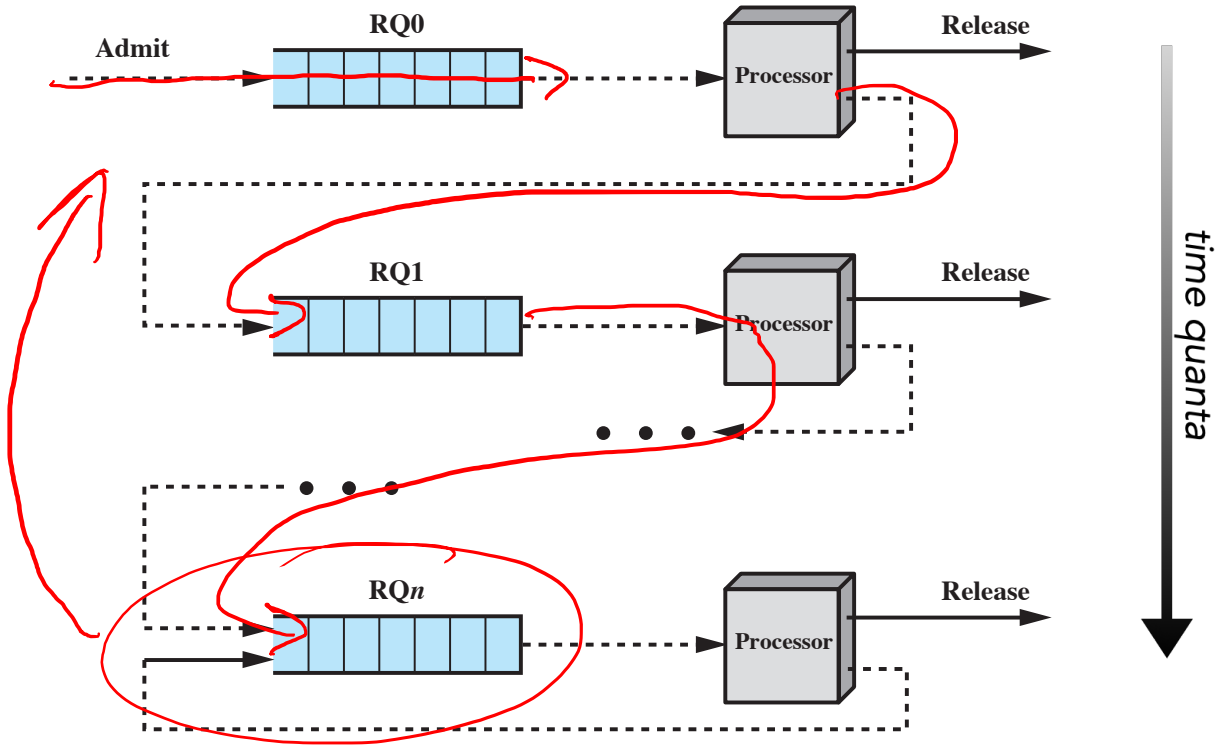
- SPN, SRT, HRRN are tricky
  -> need reasonable estimates for execution time

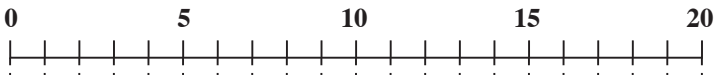
- What if we instead **"penalize"** jobs that have been running longer?

- If we can't get easy, reliable measures of
  time remaining to execute
  let us focus on
  time spent executing so far!

Multilevel Feedback

- Use some time quanta (re: RR/VRR) to periodically interrupt a running process
- "Demote" process to lower PQs each time it returns to the RQ
- Shortest processes still finish relatively quickly
- Use FCFS within each Q
  (Except lowest PQ, which uses RR)

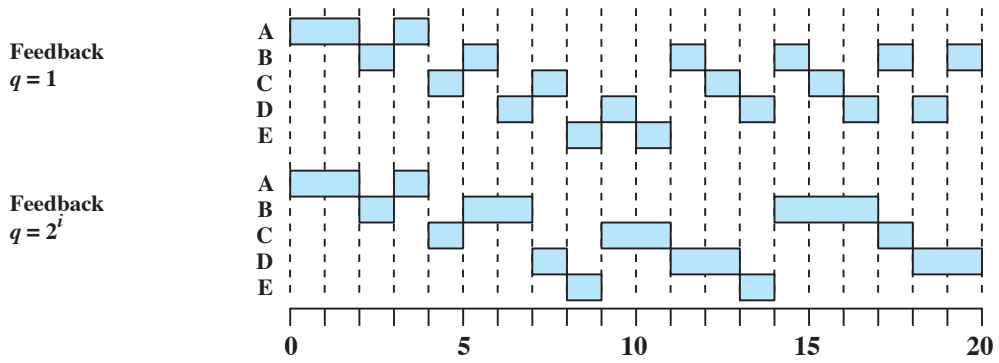




Process	A	B	C	D	E	
Arrival Time	0	2	4	6	8	
Service Time (*Ts*)	3	6	4	5	2	Mean

Multilevel Feedback

- SPN, SRT, HRRN are tricky
  -> need reasonable estimates for execution time
- What if we instead "**penalize**" jobs that have been running longer?
- If we can't get easy, reliable measures of
  time remaining to execute
  let us focus on
  time spent executing so far!



FB $q = 1$						
Finish Time	4	20	16	19	11	
Turnaround Time (*Tr*)	4	18	12	13	3	10.00
Tr/*Ts*	1.33	3.00	3.00	2.60	1.5	2.29
FB $q = 2i$						
Finish Time	4	17	18	20	14	
Turnaround Time (*Tr*)	4	15	14	14	6	10.60
Tr/*Ts*	1.33	2.50	3.50	2.80	3.00	2.63

Fair-Share Scheduling

- Approaches so far all treat processes as a fairly homogeneous collection of ready-to-execute items
  - Many independent processes
  - Can be sub-divided by priority
  - Otherwise, treated the same

- Q: Can we see a potential issue with this?
  - What if there are 2 users, Alice and Bob, where each run 1 "app"
  - … and Alice's app is made up of 1 process
  - … and Bob's app is made up of 100 processes

- Fair-Share Scheduling
  - assign each user/group a "share" of the processor
  - monitor resource usage…
    - give ***fewer resources*** to user/group that has had ***more than their fair share***
    - give ***more resources*** to user/group that has had ***less than their fair share***

NOTE:
"fair-share"
doesn't necessarily mean
"equal share"

So...
which is best?

Performance Comparisons & General Take-Aways

- Difficult to develop closed analytic models for comparisons*
  All approaches (except RR and FCFS) depend on expected service times, which really complicates things

- Relative performance depends on a variety of factors
  - ***probability distribution*** of service times of processes
  - ***efficiency*** of scheduling mechanisms
  - ***efficiency*** of context switching mechanisms
  - ***demand on***, and efficiency of, I/O subsystem(s)

-> See textbook and slides on "Queueing Analysis" and "Simulation Modeling"

Summary

Big Ideas

- Scheduling granularity
  -> long-, medium-, and *short-term**
- In scheduling, there are many objectives to balance
  -> e.g., process type, priority, (normalized) response time, no starvation.
- Various algorithms have been developed to address short-term scheduling
  -> FCFS, RR/VRR, SPN, SRT, HRRN, MLFQ, ...

What next?

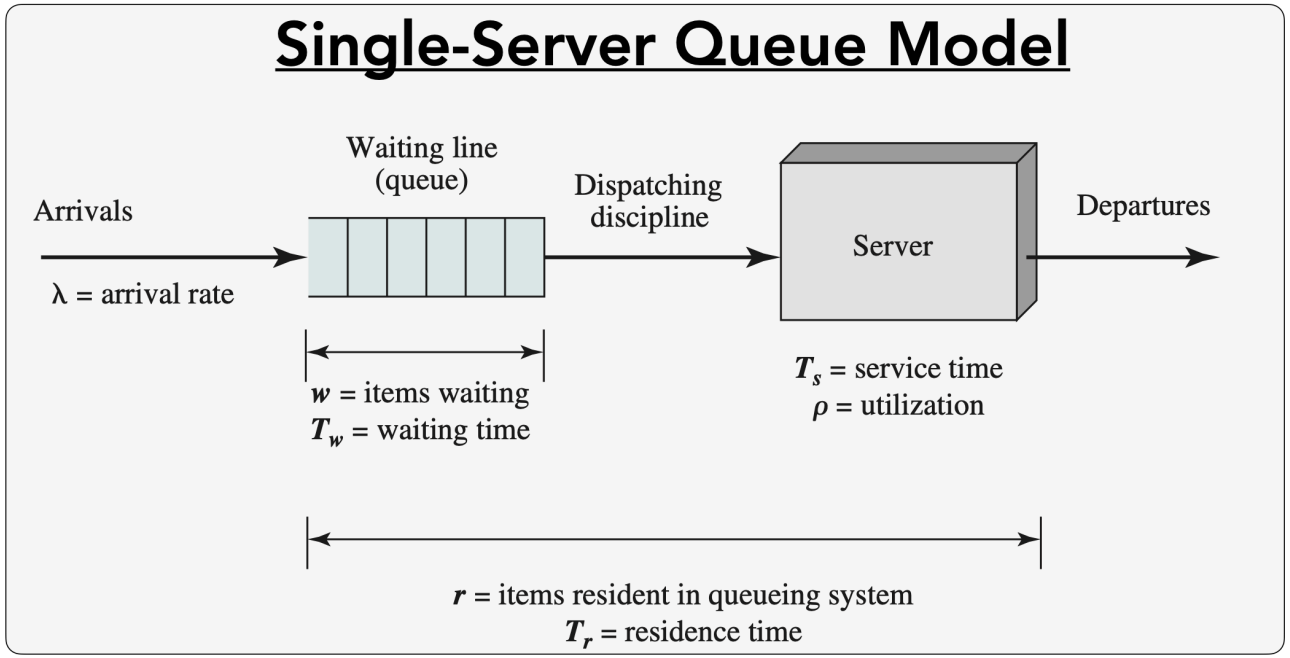
- Approaches to performance analysis and comparisons
- Extensions into multiprocessor and multicore scheduling
- Real-time scheduling
- Linux scheduling & the Completely Fair Scheduler (CFS)
- *and so much more!*

What Next?

Topics to explore when going beyond the basics of scheduling

Approach: Queuing Analysis

- Idea: Use analytic models
- Assume: Poisson arrivals
  -> random arrival times
- Assume: exponential service times
  -> more demand == longer service times
- Assume: priority-based scheduling…
  -> higher priority items served first
  -> FCFS for items of equal priority



Extensions also exist for
multiple servers and **multiple queues**

Approach: Simulation Modeling

- **Pros:** discrete event simulations allow a variety of models to be simulated & compared
  - Able to visualize direct comparisons between policies
  - Compare wait times, normalized turnaround times, …

- **Cons:** results only really hold for a given "run"
  - a particular collection of processes…
  - a particular set of assumptions & constraints…

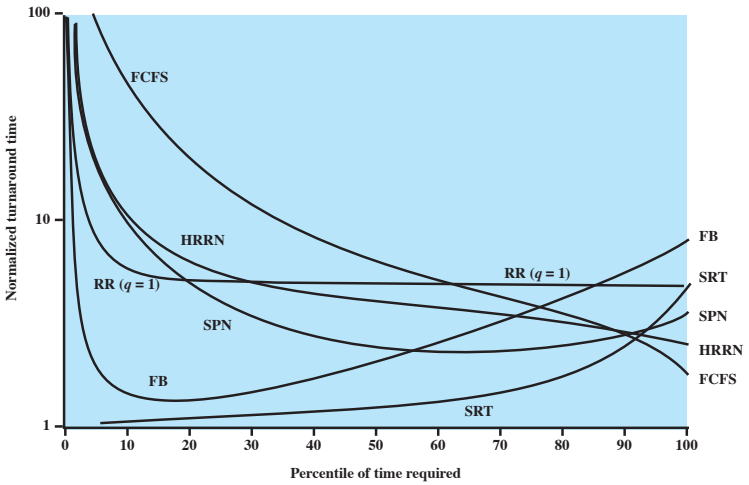


Figure 9.14 Simulation Results for Normalized Turnaround Time

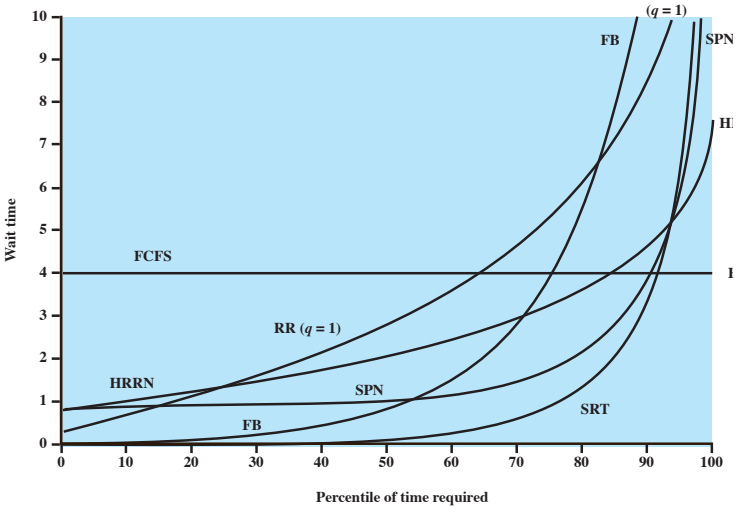
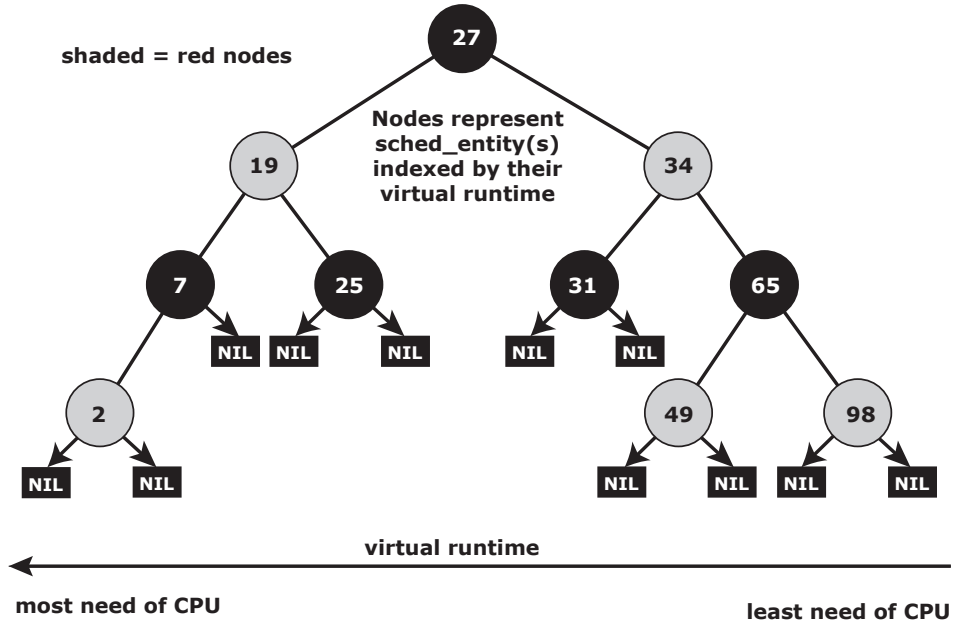


Figure 9.15 Simulation Results for Waiting Time

Linux Scheduling

- Scheduling Classes
  - **SCHED_FIFO** (limited preemption) & **SCHED_RR** (time-sliced) — (0-99)
  - **SCHED_NORMAL** (100-139)
    -> lower value == higher priority
    -> non-RT thread only executes if no RT threads are ready to execute

- Linux 2.6+ -> **O(1) Scheduler**
  - Named so because time to select & run a process takes constant time.
  - Complex and not good to run in the kernel….



- Linux 2.6.23+ -> **Completely Fair Scheduler (CFS)**
  - Maintain ***virtual runtime*** value for each task (normalized amt. of time spent executing so far)
  - ***Sleeper Fairness*** (i.e., ensure that processes that wait on I/O get fair access to processor)
  - Use Red-Black Tree to order runnable tasks (efficient inserts/deletes, and searches)