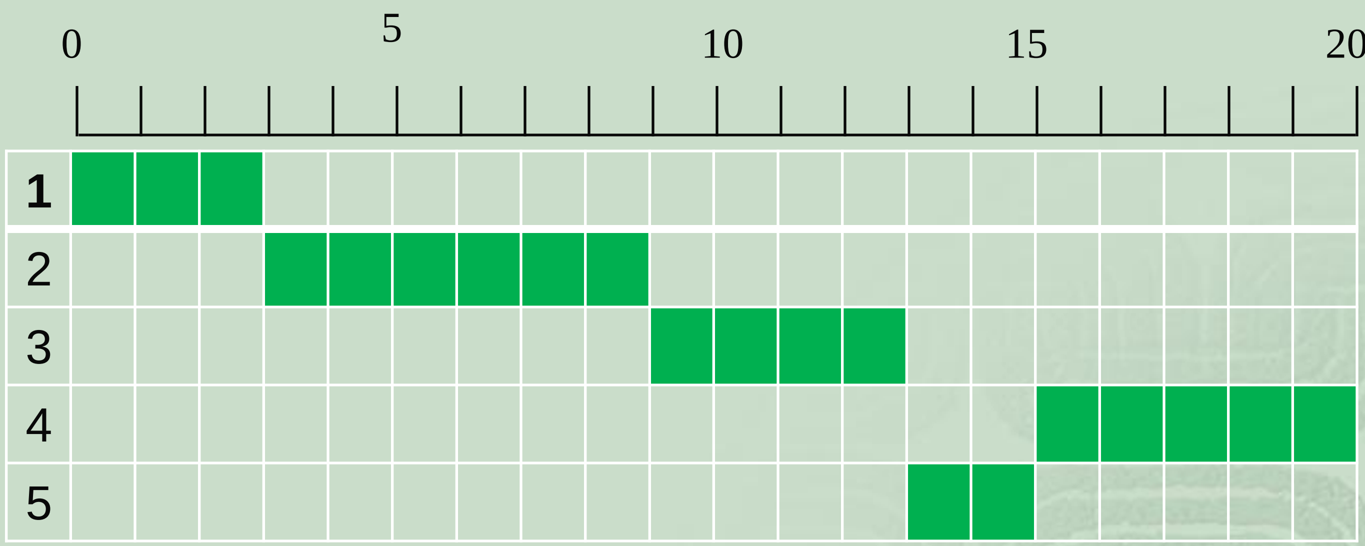
CPU Scheduling

Highest Response Ratio Next (HRRN)

Process	Arrival	Service
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



- Choose next process with the highest ratio

$$\frac{\textbf{time spent waiting + expected service time}}{\textbf{expected service time}}$$

Highest Response Ratio Next (continued…)

- Attractive approach to scheduling because it accounts for the age of a process.
- While shorter jobs are favored (a smaller denominator yields a larger ratio)

$$\frac{\textbf{time spent waiting + expected service time}}{\textbf{expected service time}}$$

Feedback Scheduling

- If we have no indication of the relative length of various processes, then SPN, SRT, and HRRN cannot be effectively used.
- Using feedback, we can give preference for shorter jobs by penalizing jobs that have been running longer.
- Feedback scheduling is done on a preemptive basis with a dynamic priority mechanism.
- A process is demoted to the next lower-priority queue each time it returns to the ready queue.

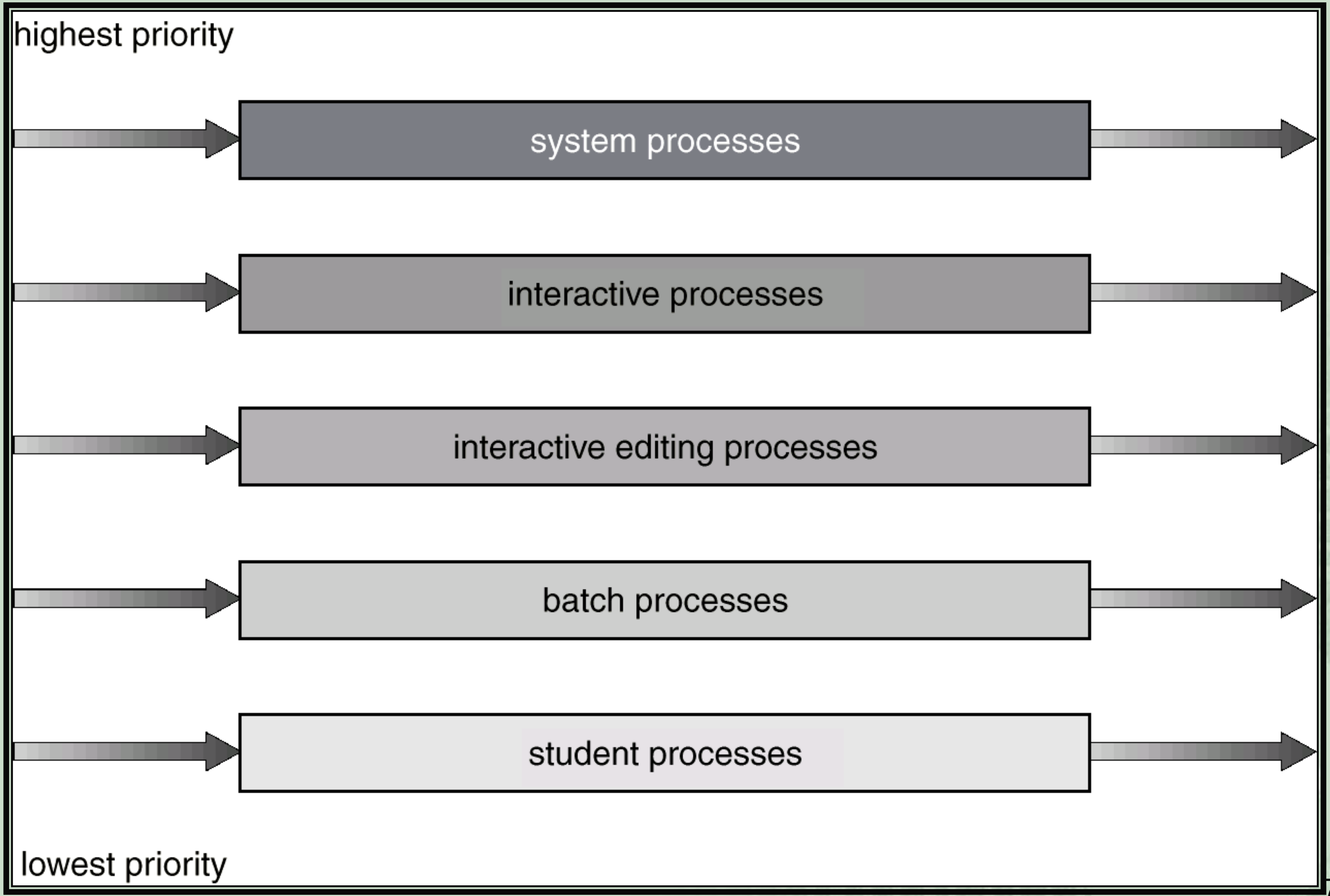
Feedback (continued…)

- Within each queue, a simple FCFS mechanism is used except once in the lowest-priority queue, a process cannot go lower and is treated in a RR fashion.
- Longer processes gradually drift downward.
- Newer, shorter processes are favored over older, longer processes.

Multilevel Queue

- Ready queue is partitioned into separate queues:
foreground (interactive)
background (batch)
- Each queue has its own scheduling algorithm,
foreground – RR
background – FCFS
- Scheduling must be done between the queues.
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

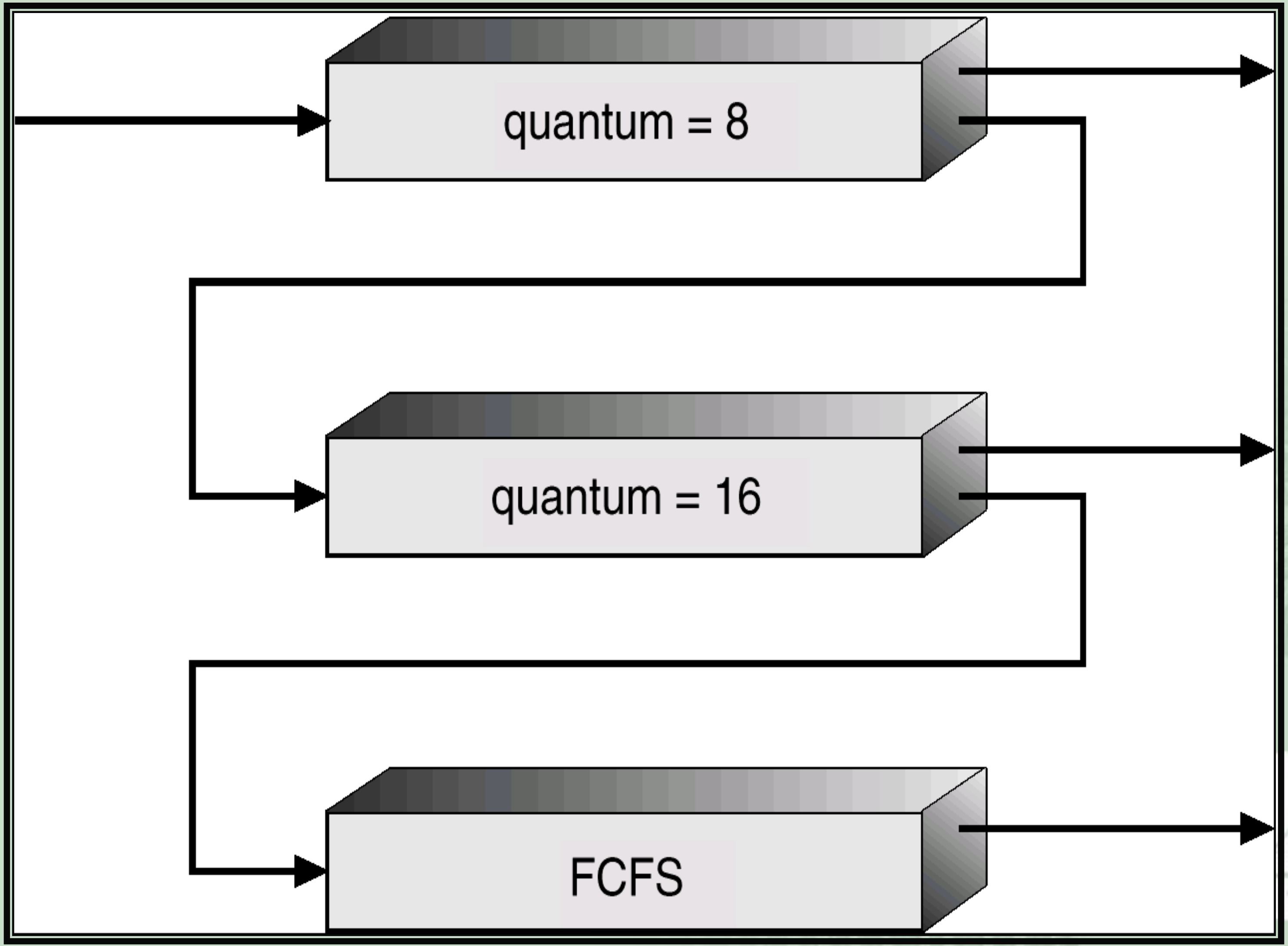
Multilevel Queue Scheduling



Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling:
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues



Scheduling Summary

- **Decide what processes should be executed by the processor**

- **Long-term scheduling**
  - Deals with creating a new process
  - Controls degree of multiprogramming
    - May be FCFS or priority based

- **Medium-term scheduling**
  - Deals with suspending processes

- **Short-term scheduling**
  - What process should we run next?
  - Invoked on:
    - clock or I/O interrupt
    - system call, signal

Short-term Criteria

- Criteria:
  - Response Time – Start to first output (interactive systems)
  - Turnaround Time – Start to finish
  - Predictability – Time doesn’t depend on what else is going on
  - Throughput – Jobs finished/unit time
  - Processor Utilization
  - Fairness – Processes treated equally
  - Enforcing Priorities
  - Balancing Resources – Try to keep all system resources busy

Scheduling

- Priorities
  - Are some processes more important than others?
  - Don't want to starve low-priority processes
- Decision Mode
  - Will we suspend the currently active process if it can continue?
    - No: Nonpreemptive
    - Yes: Preemptive
    - Yield: Some systems (Win 3.1, early Mac) used *cooperative multitasking* (processes voluntarily give up the CPU)
  - Preemption incurs more O.S. overhead, but prevents monopolizing the processor

Scheduling Algorithms

- First Come First Served (FCFS)
- Round Robin (RR) – time slicing.
- Shortest Process Next (SPN)
- Shortest Remaining Time (SRT)
- Highest Response Ratio Next (HRRH)
- Feedback (multi-queues)

Algorithms

- First Come First Served
  - Processes queued in order of arrival
  - Runs until finished or blocks on I/O
  - Tends to penalize short processes (have to wait for long processes)
  - Favors CPU-bound processes (I/O process quickly block)
- Round Robin
  - FCFS with preemption
  - Size of ticks affects performance
  - Favors CPU-bound processes

Algorithms (continued…)

- Shortest Process Next
  - Select process with shortest expected running time (non-preemptive)
  - Difficult to estimate required time (keep history)
  - Can starve long processes

- Shortest Remaining Time
  - Preemptive version of Shortest Process Next
  - May switch processes when a new process arrives
  - Still may starve long processes

Algorithms (continued…)

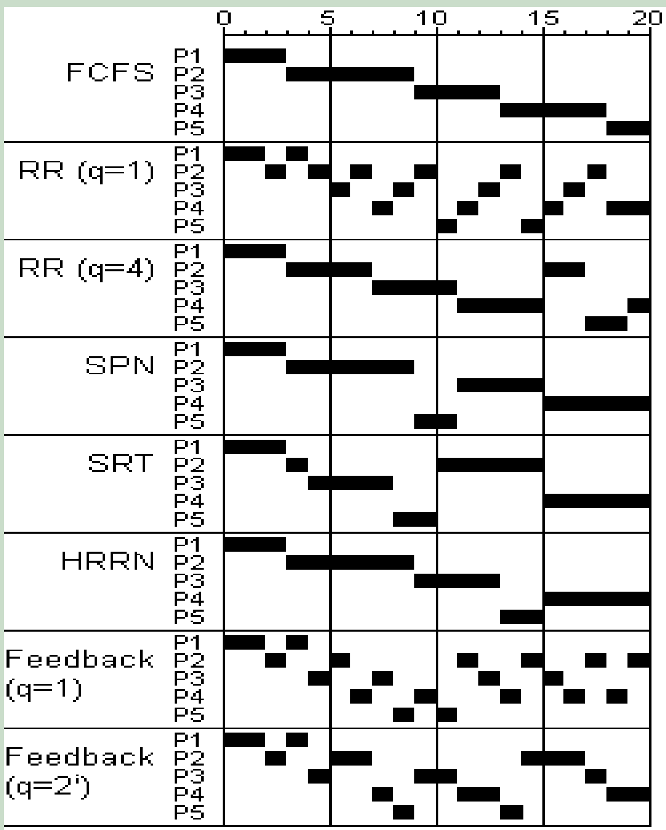
- Highest Response Ratio Next
  - Non-preemptive, tries to get best average normalized turnaround time
  - Depends on Response Ratio
    - W = time spent waiting
    - S = expected service time   RR = (W + S) / S
  - Select process with highest RR
- Feedback
  - Starts in high-priority queue, moves down in priority as it executes
  - Lower-priority queues often given longer time slices

Comparisons

	Selection Function	Decision Mode	Throughput	Response Time	Overhead	Effect on Processes	Starvation
FCFS	max[*w*]	Non-preemptive	Not emphasized	May be high, especially if there is a large variance in process execution times	Minimum	Penalizes short processes; penalizes I/O bound processes	NO
Round Robin (RR)	constant	Preemptive (at time quantum)	May be low if quantum is too small	Provides good response time for short processes	Minimum	Fair treatment; although it penalized I/O bound processes	NO
Shortest Process Next (SPN)	min[*s*]	Non-preemptive	High	Provides good response time for short processes	Can be high	Penalizes long processes	Possible
Shortest Remaining Time (SRT)	min[*s* – *e*]	Preemptive (at arrival)	High	Provides good response time	Can be high	Penalizes long processes	Possible
Highest Response Ratio Next (HRRN)	max((w + s) / s)	Non-preemptive	High	Provides good response time	Can be high	Good balance	NO
Feedback	Adjustable	Preemptive (at time quantum)	Not emphasized	Not emphasized	Can be high	May favor I/O bound processes	Possible

Comparisons

- P1: arrives at time 0, requires 3 units
- P2: arrives at time 2, requires 6 units
- P3: arrives at time 4, requires 4 units
- P4: arrives at time 6, requires 5 units
- P5: arrives at time 8, requires 2 units



	P1	P2	P3	P4	P5	Mean
Arrival	0	2	4	6	8	
Service Time	3	6	4	5	2	
FCFS						
Finish	3	9	13	18	20	
Turnaround	3	7	9	12	12	8.60
Tr / Ts	1.00	1.17	2.25	2.40	6.00	2.56
RR (q=1)						
Finish	4	18	17	20	15	
Turnaround	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	3	17	11	20	19	
Turnaround	3	15	7	14	11	10.00
Tr / Ts	1.00	2.50	1.75	2.80	5.50	2.71
SPN						
Finish	3	9	15	20	11	
Turnaround	3	7	11	14	3	7.60
Tr / Ts	1.00	1.17	2.75	2.80	1.50	1.84
SRT						
Finish	3	15	8	20	10	
Turnaround	3	13	4	14	2	7.20
Tr / Ts	1.00	2.17	1.00	2.80	1.00	1.59
HRRN						
Finish	3	9	13	20	15	
Turnaround	3	7	9	14	7	8.00
Tr / Ts	1.00	1.17	2.25	2.80	3.50	2.14
Feedback (q=1)						
Finish	4	20	16	19	11	
Turnaround	4	18	12	13	3	10.00
Tr / Ts	1.33	3.00	3.00	2.60	1.50	2.29
Feedback (q=2^(i-1))						
Finish	4	17	18	20	14	
Turnaround	4	15	14	14	6	10.60
Tr / Ts	1.33	2.50	3.50	2.80	3.00	2.63

Linux Scheduling

Numeric priority	Relative priority		Time quantum
0	highest	Real-time tasks	200ms
…			
99			…
100		Other tasks	
…			
…			
140	lowest		10ms

Linux assigns higher-priority tasks longer quanta

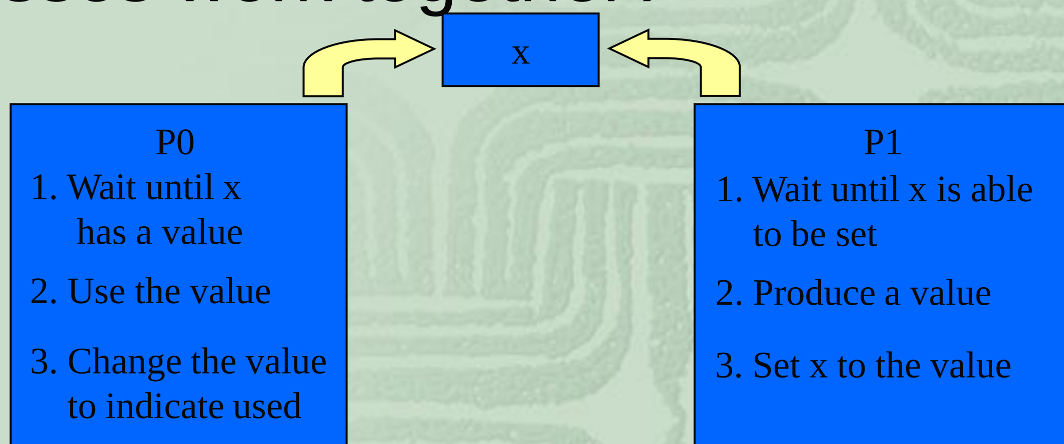
Windows 2000 Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Concurrent Processes

Cooperating Processes

- Operating systems allow for the creation and concurrent execution of multiple processes
  - concurrency can ease program complexity
  - concurrency can increase efficiency
- How can the processes work together?
  - Flags
  - Files
  - Messages
  - Shared memory



Cooperating Processes

- Concurrent processes (or threads) often need to share data (maintained either in shared memory or files) and resources
- If there is no controlled access to shared data, some processes will obtain an inconsistent view of this data
- The actions performed by concurrent processes may depend on the order in which their execution is interleaved
- With no synchronization, results are typically not deterministic nor reproducible.