CPU Scheduling

ECE469, Feb 7

8. Using timer interrupt to do CPU management
27. Scheduling policies

Yiying Zhang

Readings

- Dinosaur Chapter 5
- Comet Chapter 7

Roadmap So Far

- Processes, creation
- Inter-process comm by sharing data → process synchronization
  - OS-provided sync. Primitives
    - Mutual exclusion & Critical section
    - Semaphore (binary semaphore)
    - Lock / condition variable
  - Classic sync. Problems
    - Producer-consumer problem
    - Reads-writers problem
    - Dining Philosophers problem (deadlock)
  - Semaphore implementation in OS
  - Wait-free synchronization
- Inter-process comm by messaging (mailboxes)
- Deadlocks

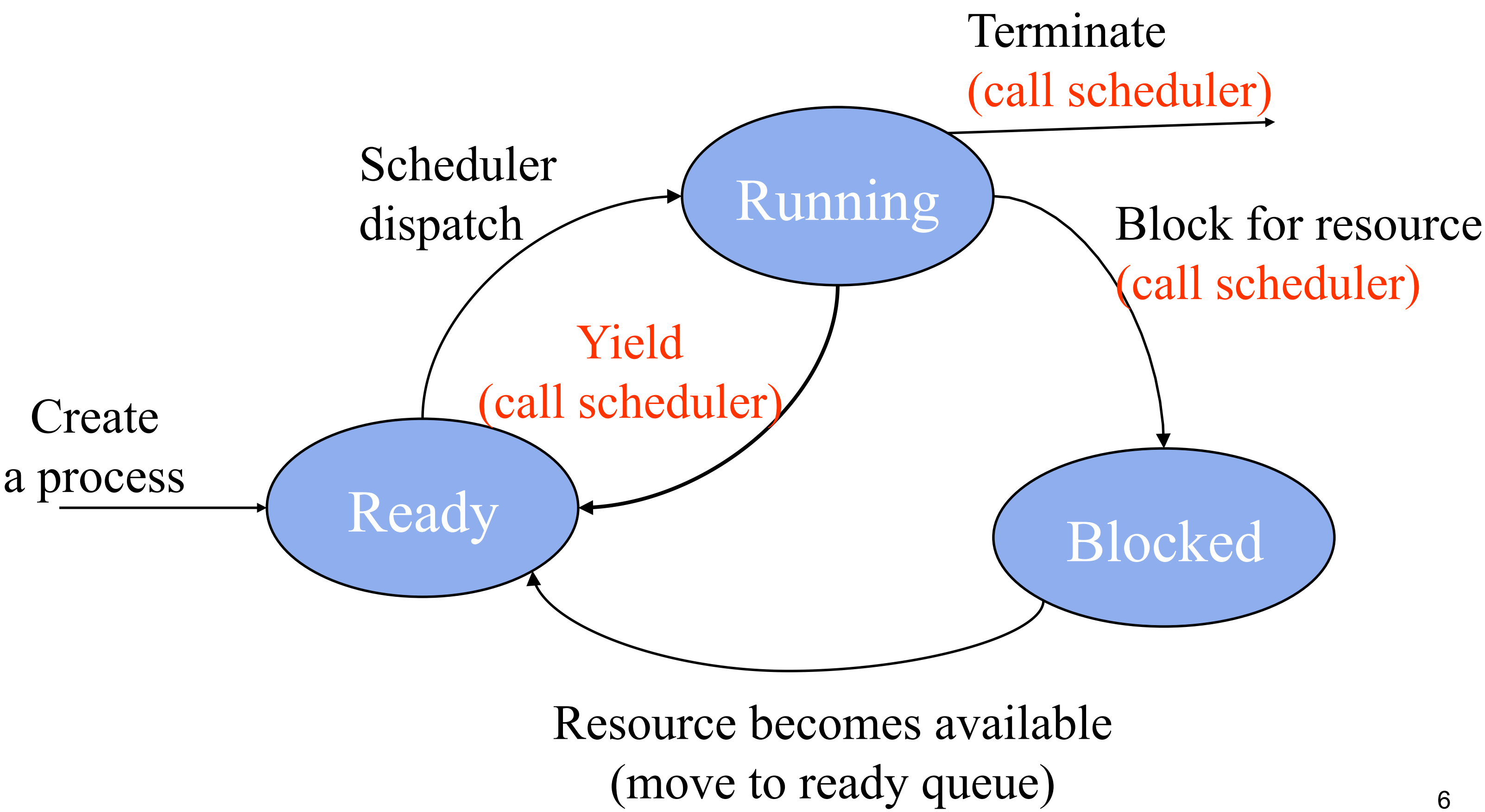
CPU Scheduling

- CPU scheduling is the basis of multiprogrammed operating systems
- By switching the CPU among processes, the OS can make the CPU/computer maximally utilized

Hardware Support

- Without hardware support, can we do anything other than non-preemptive scheduling?

[lec3] Process State Transition of Non-Preemptive Scheduling



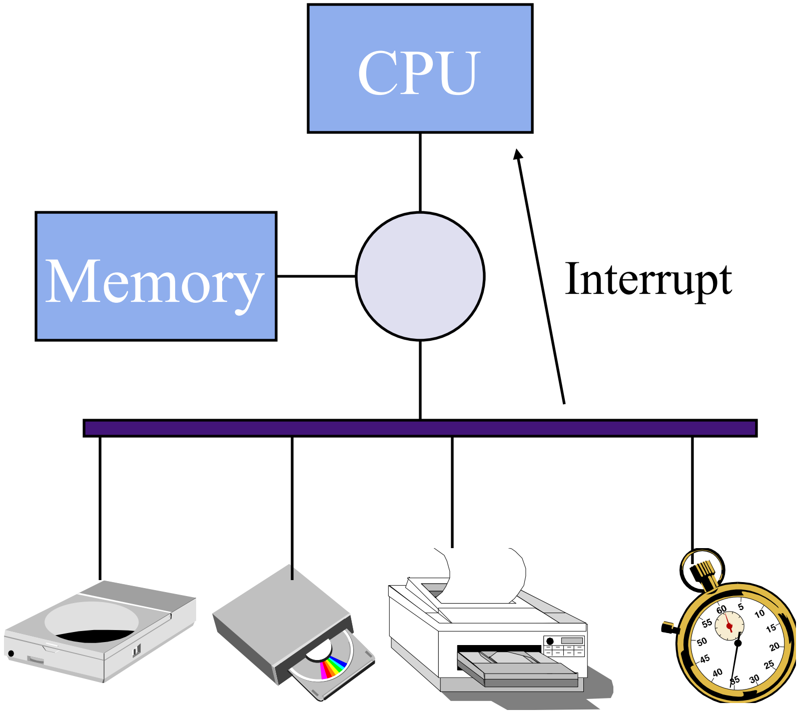
Timesharing Systems

- Timesharing systems support interactive use
  - each user feels he/she has the entire machine
- How?
  - optimize response time
  - based on time-slicing

Timer Interrupts

- Using timer interrupt to do CPU management

- Timer interrupt
    - generated by hardware
    - setting requires privilege
    - delivered to the OS



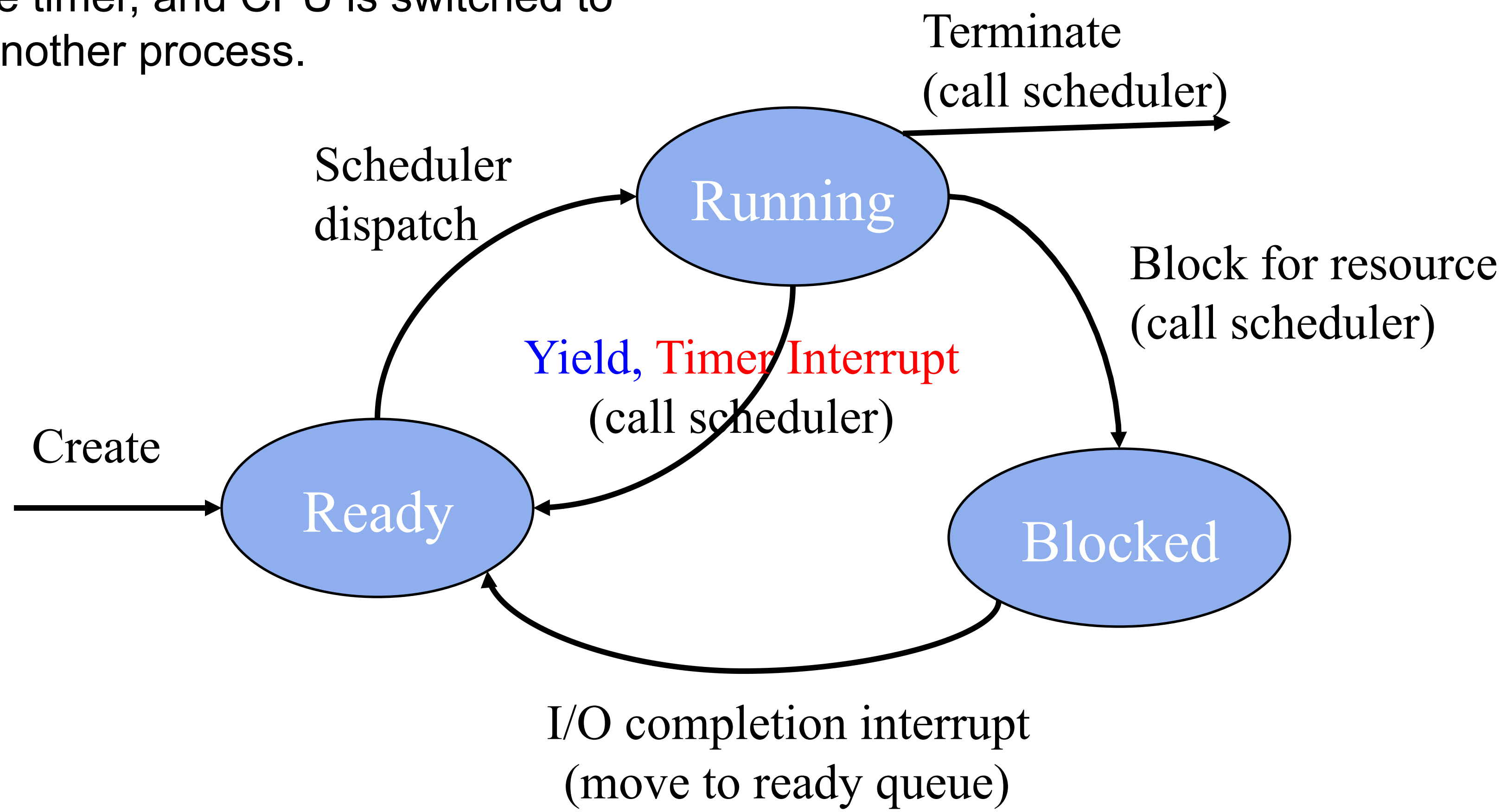
Using Interrupts For Scheduling

Basic idea

- before moving process to running, OS sets timer
- if process yields/blocks, clear timer, go to scheduler
- If timer expires, go to scheduler

Preemptive Scheduling

A running process is interrupted by the timer, and CPU is switched to run another process.



[lec3] *Context Switch*

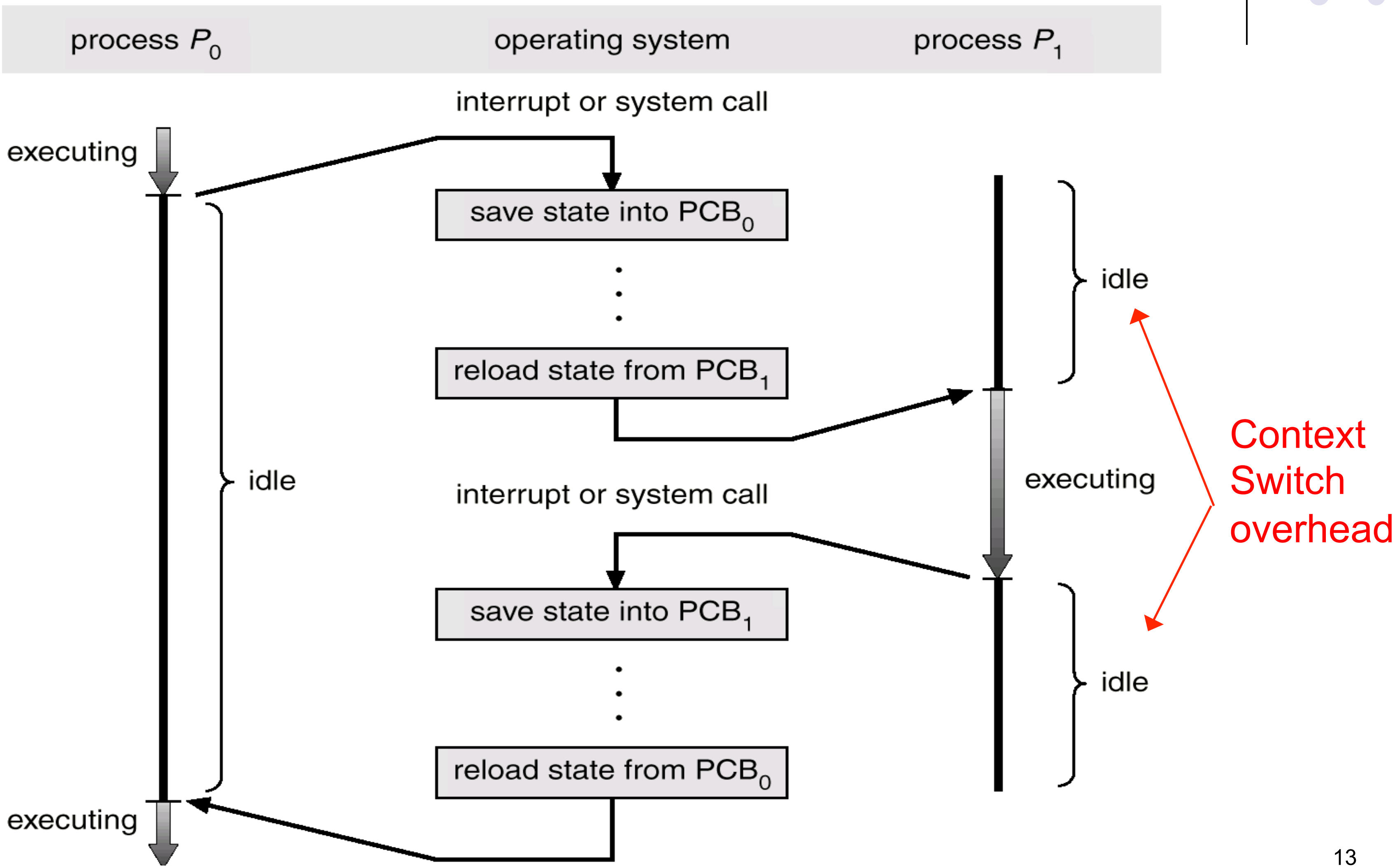
- Definition:
  switching the CPU to another process, which involves saving the state of the old process and loading the state of the new process

- What state?
- Where to store them?

Process Control Block (Process Table)

- Process management info
  - State (ready, running, blocked)
  - PC & Registers, parents, etc
  - CPU scheduling info (priorities, etc.)
- Memory management info
  - Segments, page table, stats, etc
- I/O and file management
  - Communication ports, directories, file descriptors, etc.

Context Switch



Preemptive Scheduling Considerations

- Timer granularity
  - Finer timers = more responsive
  - Coarser timers = more efficient

- CPU Accounting (CPU running stats)
  - Used by the scheduler
  - Useful for the programmer

OS as a Resource Manager: Allocation vs. Scheduling

- Allocation (spatial)
  - Who gets what. Given a set of requests for resources (e.g. memory), which processes should be given which resources (e.g. how much memory & where) for best utilization
- Scheduling (temporal)
  - How long can they keep it. When more resources (e.g. 10 CPUs) are requested than can be granted (e.g. 1 CPU), in what order can they be serviced?

[lec1] Separating Policy from Mechanism

Mechanism – tool to achieve some effect

Policy – decisions on how to use tool like algorithm

examples:

- All users treated equally
- All program instances treated equally
- Preferred users treated better

Separation leads to flexibility

Preemptive CPU Scheduling

- What is in it?
  - Mechanism + policy
  - Mechanisms fairly simple
  - Policy choices harder

[lec1] Brief History of Computer Systems (1)

- In the beginning, 1 user/program at a time
- Simple batch systems were 1st real OS
  - Spooling and buffering allowed jobs to be read ahead of time

- Multiprogramming systems provided increased utilization (throughput)
  - multiple runable jobs loaded in memory
  - overlap I/O with computation
  - benefit from asynchronous I/O devices
  - 1st instance where the OS must allocate and schedule resources
    - CPU scheduling
    - Memory management
    - Protection

[lec1] Brief History of Cmputer Systems (2)

- Timesharing systems support interactive use
  - Logical extension of multiprogramming
  - optimize response time by frequent time-slicing multiple jobs
  - each user feels he/she has the entire machine
  - permits interactive work

  - Most systems today are timesharing (focus of this class)

[lec1] Is there a perfect OS?
(resource manager, abstract machine)

Fairness

Efficiency

Portability

Interfaces

Security

Robustness

- Conflicting goals
  - Fairness vs efficiency
  - Efficiency vs portablity
  - …
- Furthermore, …

Challenges in Policy

- Flexibility - variability in job types
  - Long vs. short
  - Interactive vs. non-interactive
  - I/O-bound vs. compute-bound

- Issues
  - Short jobs shouldn't suffer go first, total waiting time low
  - (Interactive) Users shouldn't be annoyed

Challenges in Policy (cont)

- Fairness
  - All users should get access to CPU
  - Amount of CPU should be roughly even?

- Issue
  - Short-term vs. long-term fairness

Goals and Assumptions

- Goals (Performance metrics)
  - Minimize turnaround time
    - avg time to complete a job
    - $T_{turnaround} = T_{completion} - T_{arrival}$
  - Maximize throughput
    - operations (jobs) per second
    - Minimize overhead of context switches: large quanta
    - Efficient utilization (CPU, memory, disk etc)
  - Short response time waiting time
    - $T_{response} = T_{firstrun} - T_{arrival}$
    - type on a keyboard
    - Small quanta
  - Fairness
    - fair, no stavaton, no deadlock

Goals and Assumptions

- Goals often conflict
  - Response time vs. throughput
  - fairness vs. avg turnaround time?

- Assumptions
  - One process/program per user
  - Programs are independent

Scheduling policies

- Is there an optimal scheduling policy?
- Even if we narrow down to one goal?

- But we don’t know about future
  - Offline vs. online

Queuing Theory:
the mathematical study of waiting lines, or *queues*.

- An entire discipline to itself
- Mathematically oriented
- Some neat results

- Assumptions may be too restrictive to be able to model real-world situations exactly
  - E.g. assume infinite number of customers, infinite queue capacity, or no bounds on inter-arrival or service times
- Systems have grown more complex these days
- (Workload-driven) Simulation used instead now

Scheduling policies

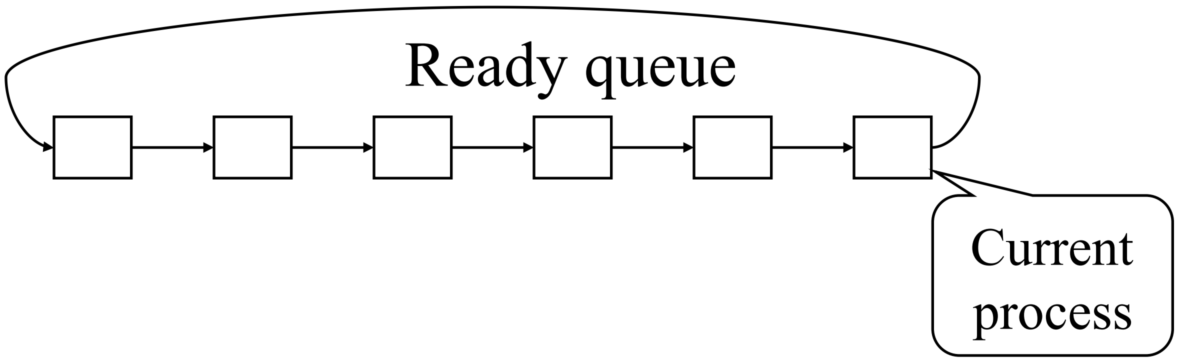
- FIFO
- Round Robin
- SJCF
- SRTCF

(Non-Preemptive scheduling) FIFO (FCFS) Policy

first come first serve

- What does it mean?
  - Run to completion (old days)
  - Run until blocked or yield
- Advantages
  - Simple
- Disadvantage?

Round Robin



- Each runs a time slice or ***quantum***
- How do you choose time slice?
  - Overhead vs. response time
  - Overhead is typically about 1% or less
  - Quantum typically between 10 ~ 100 millisec

Is Fairness Always Good?

- Assume 10 jobs waiting to be scheduled, each takes 100 seconds
- Assume no other overhead
- Total CPU time? 1000 seconds, always

- Implications?
  - Last job always finishes at 1000 seconds
  - So what’s the point of scheduling?

FIFO Example

- Job 1 – start 0, end 100
- Job 2 – start 100, end 200
- …
- Job 10 – start 900, end 1000

- Average turnaround time =100 + 200 +… / N = 550 sec

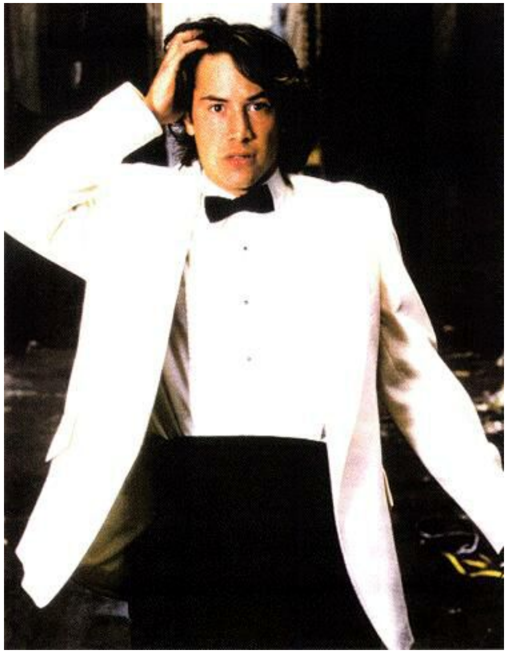
Round Robin Example

- Assume each quantum is 1 second
- Job 0 – 0, 10, 20, 30, 40,…, 990
- Job 1 – 1, 11, 21, 31,…, 991
- Job 2 – 2, 12, 22, 32,…, 992
- …

average turnaround 巨长， 因为所有人都在等， 都没有完成

- Avg turnaround time = 990+991+…/N = 995

Like, Whoa! Dude!



- Unfair policy was faster!
- Job 10 always ended at the same time
- Round-Robin just hurt jobs 1-9 with no gain

So Why Use Round-Robin?

- Imagine 10 jobs
- Jobs 1-9 are 100 seconds
- Job 10 is 10 seconds

- Which policy is better now?

FIFO again

- Jobs 1-9 are 100 seconds
- Job 10 is 10 seconds

- Job 0 – start 0, end 100
- Job 1 – start 100, end 200
- Job 10 – start 900, end 910

- Avg turnaround time = 100+200+…910/N = 541

Round-robin again

- Jobs 1-9 are 100 seconds
- Job 10 is 10 seconds

9% work drop →
2% avg turnaround drop for FIFO
17% avg turnaround drop for RR

- Job 0 – 0, 10, 20, …, 900
- Job 1 – 1, 11, 21, …, 901
- Job 10 – 9, 19, 29, …, 99

- Avg turnaround time = 900 + 901 + 908 + 99 / 10 = 824

So Why Use Round-Robin?

- Imagine 10 jobs
- Jobs 1 is 100 seconds
- Job 2-10 is 10 seconds

- Which policy is better now?
  - FIFO: average turnaround 145
  - RR: average turnaround 105