CSE 3320
Operating Systems
CPU Scheduling

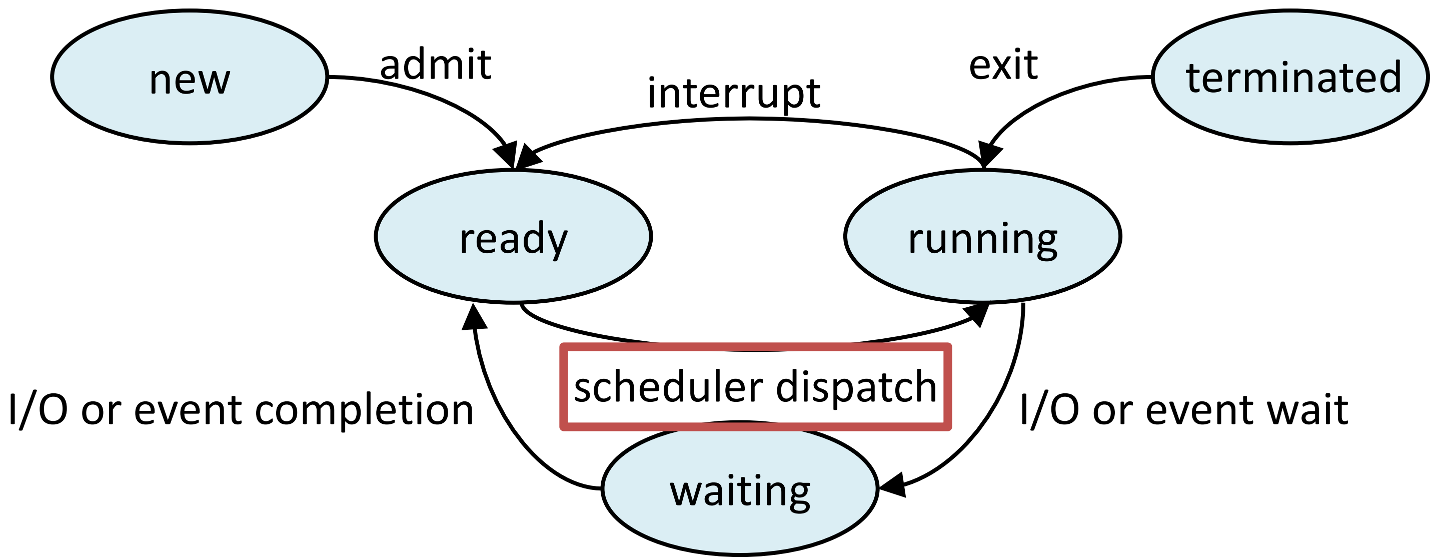
Jia Rao

Department of Computer Science and Engineering

http://ranger.uta.edu/~jrao

What is CPU Scheduling?

- The five-state process model



CPU scheduling

Selects from among the processes/threads that are ready to execute, and allocates the CPU to it

Why CPU Scheduling?

- In support of multiprogramming
  - uniprocessor systems
    - Time-sharing processor
  - multiprocessor systems
    - Efficiently distributing tasks
  - Real-time systems
    - Reliably guaranteeing deadlines
- It is (maybe) the most important part in a OS
  - Why some OS seems to be faster than others?
  - Why I do not see performance improvement when upgrading to a 16-core computer?

In this Lecture

- Outline
  - Basics of CPU scheduling
    - Scheduling policies
    - Evaluation criteria
    - Examples
    - A practical policy
  - Challenges on emerging hardware and applications
    - Many core, NUMA, asymmetric processors
    - Data center, accurate resource provisioning
  - A close look at the state-of-art
    - The Linux CFS scheduler

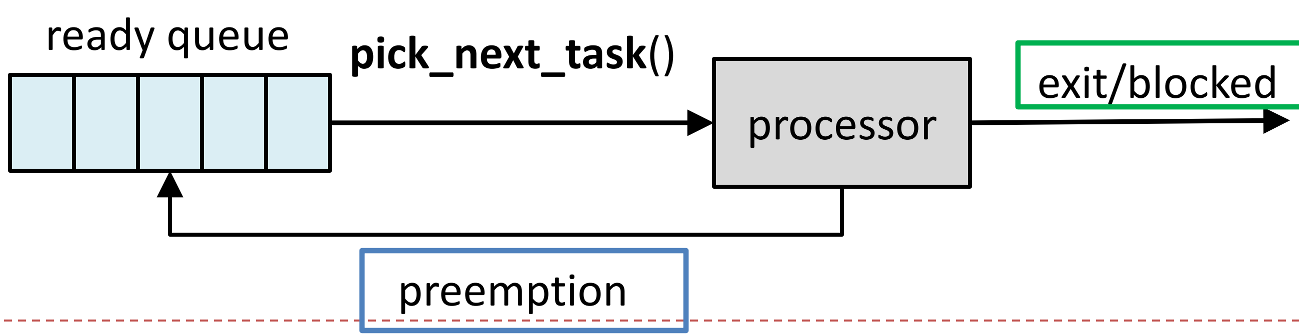
CPU Scheduling

- CPU scheduling may take place at
  - Clock interrupts
  - I/O completion
  - I/O interrupts
  - Termination

preemptive

nonpreemptive

- Nonpreemptive
  - Scheduling only when current process terminates or gives up control
- Preemptive
  - Processes can be forced to give up control



Scheduling Goals

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

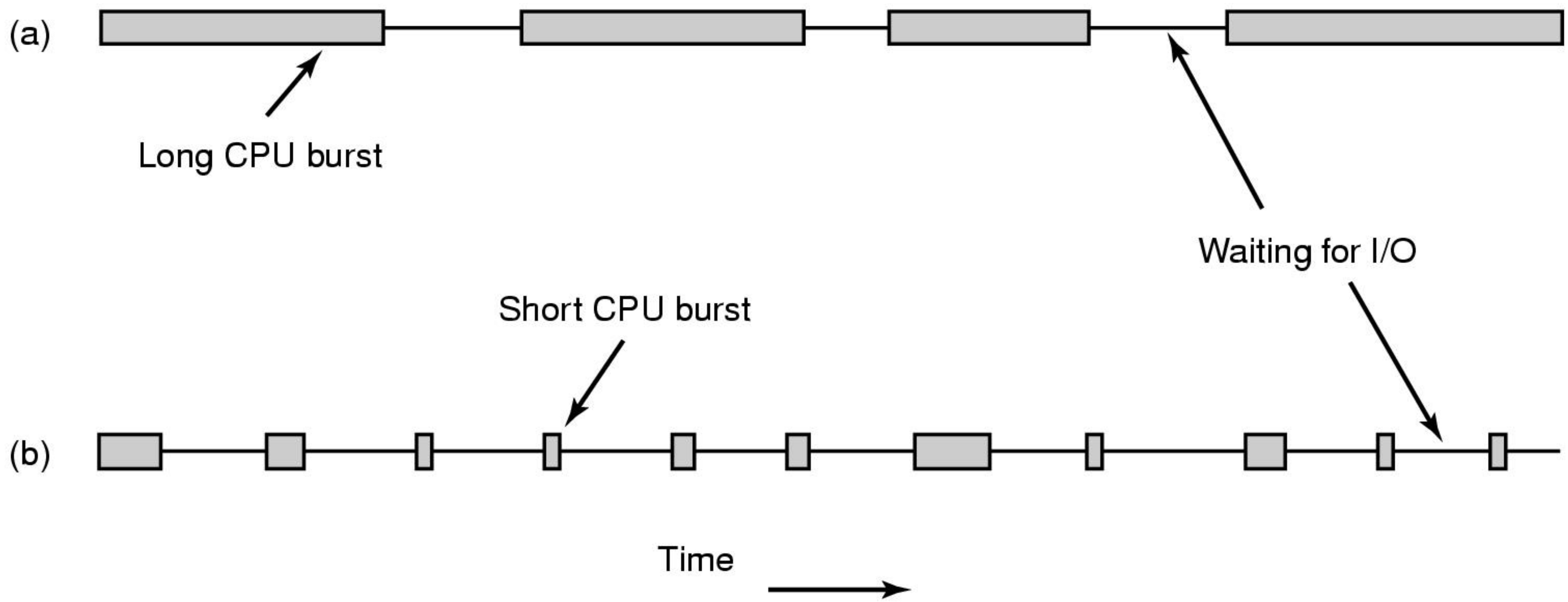
Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

Scheduling Goals: A Different Point of View

- User oriented → minimize
  - Response time (wait time): the time that the first response is received (interactivity)
  - Turnaround time: the time that the task finishes
  - Predictability: variations in different runs
- System oriented → maximize
  - Throughput: # of tasks that finish per time unit
  - Utilization: the percentage of time the CPU is busy
  - Fairness: avoid starvation

Process Behaviors



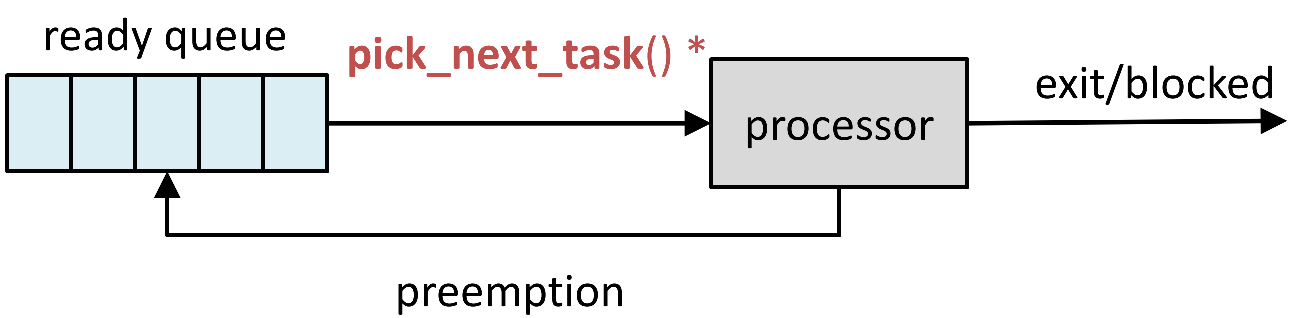
- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound/CPU-intensive process
  - an I/O bound / I/O intensive process
    - I/O is when a process enters the blocked state waiting for an external device to complete its work

Scheduling Policies

- Batch Systems
  - First-Come First-Serve
  - Shortest Job First
  - Shortest Remaining Time Next
- Interactive Systems
  - Round-Robin
  - Priority Scheduling
  - Multiple Queues
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
- Real-time Systems
  - Rate Monotonic Scheduling
  - Earliest Deadline First Scheduling

More on Scheduling Policy

- Determine the next ready task to run
  - How we design pick_next_task()
- Basic policies
  - First-Come, First -Served (FCFS)
  - Shortest-Job-First (SJF)
  - Round Robin (RR)
  - Priority scheduling

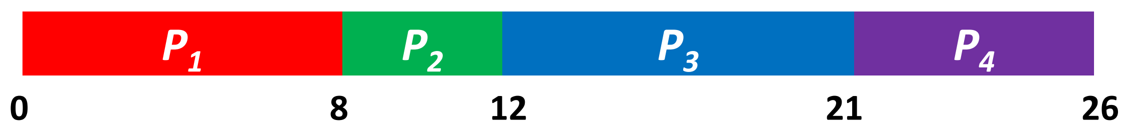


* The actual major schedule function in Linux

First-Come, First-Serve (FCFS)

- CPU schedules the task that arrived earliest, non-preemptive

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Average turnaround time = ((8-0)+(12-1)+(21-2)+(26-3)) / 4 = 15.25

Average response time = (0+(8-1)+(12-2)+(21-3)) /4 = 8.75

Average turnaround time = ((5-0)+(14-1)+(18-2)+(26-3)) / 4 = 14.25

Average response time = (0+(5-1)+(14-2)+(18-3)) /4 = 7.75

Shortest Job First (SJF)

- CPU schedules the task with the shortest remaining time

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

nonpreemptive

Average turnaround time = ((8-0)+(12-1)+(26-2)+(17-3)) / 4 = 14.25

Average response time = (0+(8-1)+(17-2)+(12-3)) /4 = 7.75

preemptive

Average turnaround time = ((17-0)+(5-1)+(10-3)+(26-2)) / 4 = 13

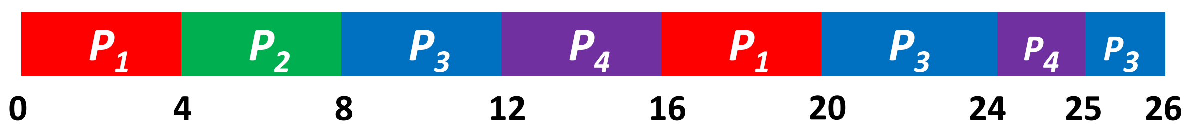
Average response time = (0+(1-1)+(5-3)+(17-2)) /4 = 4.25

Round Robin (RR)

- Like FCFS, but with limited time slices, preemptive

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

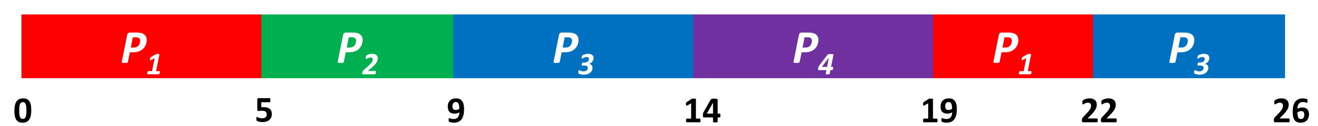
***q*=4**



Average turnaround time = ((20-0)+(8-1)+(26-2)+(25-3)) / 4 = 18.25

Average response time = (0+(4-1)+(8-2)+(12-3)) /4 = 4.5

***q*=5**



Average turnaround time = ((22-0)+(9-1)+(26-2)+(19-3)) / 4 = 17.5

Average response time = (0+(5-1)+(9-2)+(14-3)) /4 = 5.5

Priority Scheduling

- CPU schedules the highest priority first, FCFS within the same priority

Process	Priority	Burst Time
P_1	3	8
P_2	1	4
P_3	4	9
P_4	2	5

Process	Priority	Burst Time
P_1	2	8
P_2	4	4
P_3	1	9
P_4	3	5

Put it together

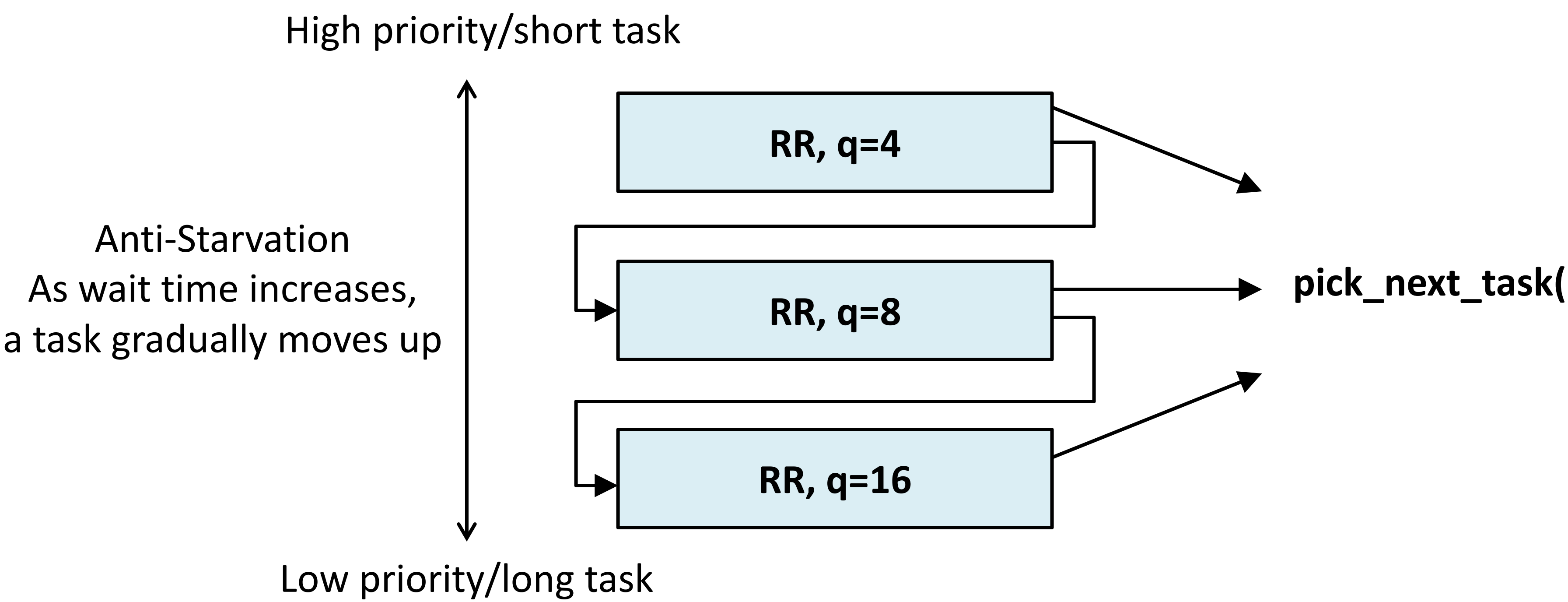
	Turnaround time	Response time
FCFS	15.25	8.75
SJF-preemptive	**13**	**4.25**
RR (q=5)	17.5	5.5
Priority scheduling	N/A	N/A

	Throughput	Response time	Starvation
FCFS	TBD	TBD	No
SJF-preemptive	High	Good	Yes
RR	Can be low	Good	No
Priority scheduling	Can be high	Can be good	Can remove

⇩

Multilevel Feedback Queue

Multilevel Feedback Queue



Windows XP, Mac OS X, Linux 2.6.22 and before

Real-time Scheduling

Schedulable real-time system

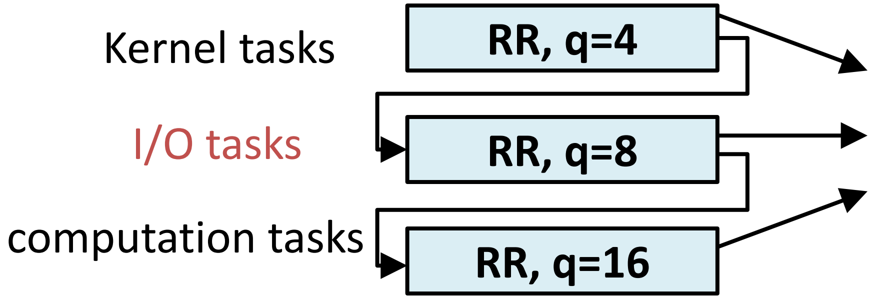
- Given
  - *m* periodic events
  - event *i* occurs within period P_i and requires C_i seconds
- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- Example: a soft real-time system with three periodic events, with periods of 100, 200, and 500 ms, respectively. If these events require 50, 30, and 100 ms of CPU time per event, respective, the system is schedulable
  - Process/context switching overhead is often an issue though!
  - Given the example, what would be the maximum CPU burst for a 4th job with a period of 500 ms ?

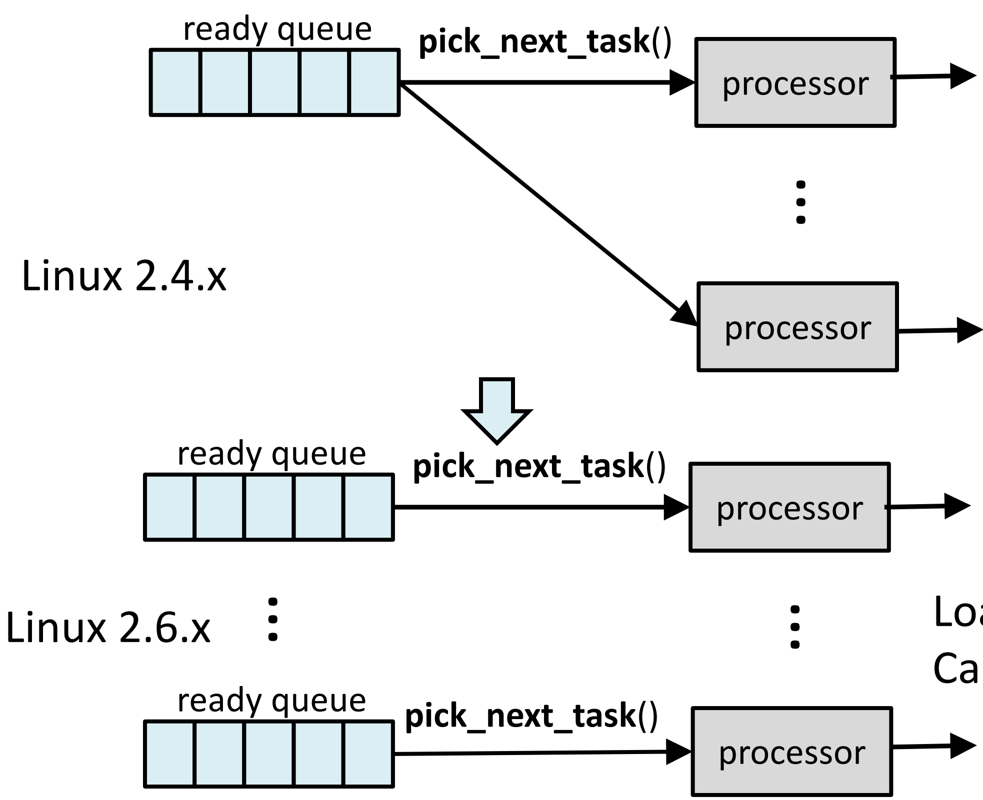
Misc.

- I/O tasks
  - Identify: runs few, sleeps a lot
  - Considered as short task
  - High priority
- Theoretical analysis
  - Assume task distribution
  - Queuing model



Challenges on Emerging Hardware and Applications

- Multiprocessor→Many core



Multiprocessor = more powerful processor

pick_next_task() will be the bottleneck

Load balancing
Cache affinity

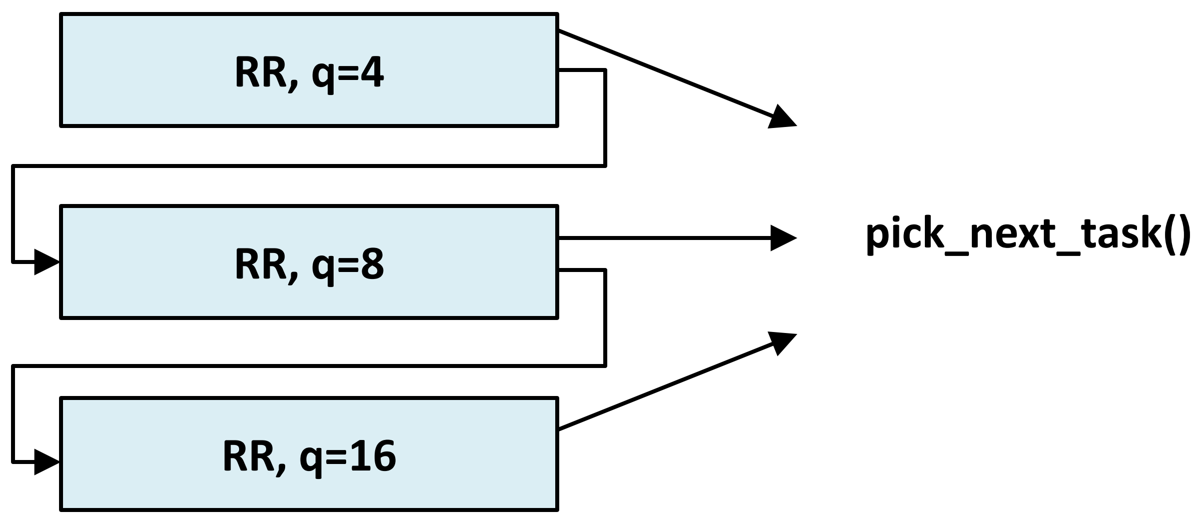
Problem:
Self-scheduling, global control is difficult

Challenges on Emerging Hardware and Applications (cont')

- NUMA, Asymmetric processors
  - OLD: CPU time → useful work
  - NEW: calibrated CPU time →useful work
- Data center, accurate resource provisioning
  - Proportional fair sharing → $P_1 : P_2 = 1 : 2$

q=?
When to move task?
Heuristic-based

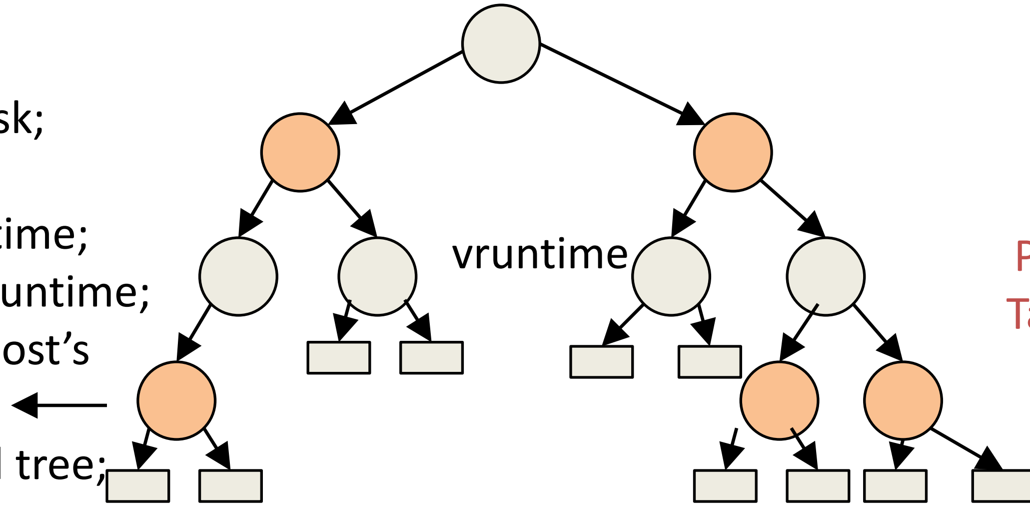
Fine-grained
Adaptive *q*



A Close Look at the State-of-Art

- Linux Completely Fair Scheduler (CFS)
  - Separate ready queue per processor
  - Red-black tree based ready queue
  - Proportional fair sharing

pick_next_task()
Remove the leftmost task;
Run the task;
Every ticks update vruntime;
vruntime = vruntime + runtime;
if vruntime > curr_leftmost's
Preempt curr task;
Put it back to the sorted tree;
Run curr_leftmost;



Fine-grain
Adaptive *q*
Priority: low priority
Task's vruntime runs
Faster
No starvation

Summary

- The basic scheduling policies are important
  - Multilevel Feedback Queue = RR + SJF + Priority
  - CFS = RR + SJF + Priority + smart data structure
- Additional readings
  - Go to http://lxr.linux.no/linux+v2.6.24/kernel/
  - Read /kernel/sched.c, /kernel/sched_fair.c, /include/linux/sched.h (starting from the schedule(void) function)
  - See how the vruntime is actually updated
  - Documentation: http://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
  - Another interesting scheduler: BFS
    - http://ck.kolivas.org/patches/bfs/bfs-faq.txt