

# Priority-driven Scheduling of Periodic Tasks (1)

Real-Time and Embedded Systems (M)

Lecture 5

UNIVERSITY  
*of*  
GLASGOW



# Lecture Outline

- Assumptions
- Fixed-priority algorithms
  - Rate monotonic
  - Deadline monotonic
- Dynamic-priority algorithms
  - Earliest deadline first
  - Least slack time
- Relative merits of fixed- and dynamic-priority scheduling
- Schedulable utilization and proof of schedulability

The material in lectures 5 & 6 corresponds to chapter 6 of Liu's book

# Assumptions

- Focus on well-known priority-driven algorithms for scheduling periodic tasks on a single processor
- Assume a restricted periodic task model:
  - A fixed number of independent periodic tasks exist
    - Jobs comprising those tasks:
      - Are ready for execution as soon as they are released
      - Can be pre-empted at any time
      - Never suspend themselves
    - New tasks only admitted after an acceptance test; may be rejected
    - The period of a task defined as minimum inter-release time of jobs in task
  - There are no aperiodic or sporadic tasks
  - Scheduling decisions made immediately upon job release and completion
    - Algorithms are event driven, never intentionally leave a resource idle
  - Context switch overhead negligibly small; unlimited priority levels

# Dynamic versus Static Systems

- Recall from lecture 3:
    - If jobs are scheduled on multiple processors, and a job can be dispatched to any of the processors, the system is *dynamic*
    - If jobs are partitioned into subsystems, each subsystem bound statically to a processor, we have a *static* system
    - Difficult to determine the best- and worst-case performance of dynamic systems, so most hard real-time systems built are static
  - In static systems, the scheduler for each processor schedules the jobs in its subsystem independent of the schedulers for the other processors
- ⇒ Results demonstrated for priority-driven uniprocessor systems are applicable to each subsystem of a static multiprocessor system
- They are *not* applicable to dynamic multiprocessor systems

# Fixed- and Dynamic-Priority Algorithms

- A priority-driven scheduler is an on-line scheduler
  - It does *not* pre-compute a schedule of tasks/jobs: instead assigns priorities to jobs when released, places them on a run queue in priority order
  - When pre-emption is allowed, a scheduling decision is made whenever a job is released or completed
  - At each scheduling decision time, the scheduler updates the run queues and executes the job at the head of the queue
- Jobs in a task may be assigned the same priority (*task level fixed-priority*) or different priorities (*task level dynamic-priority*)
- The priority of each job is usually fixed (*job level fixed-priority*); but some systems can vary the priority of a job after it has started (*job level dynamic-priority*)
  - Job level dynamic-priority usually very inefficient

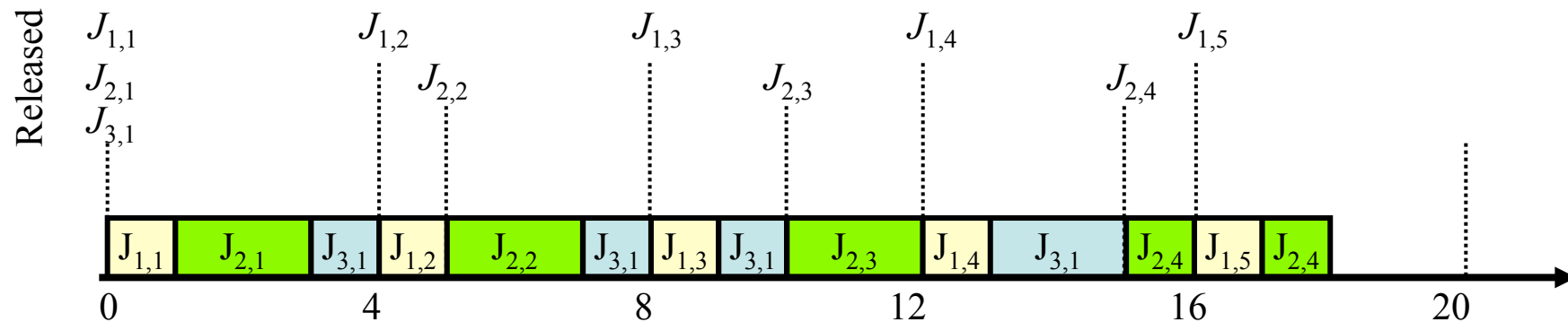
# Fixed-Priority Scheduling: Rate-Monotonic

- Best known fixed-priority algorithm is *rate-monotonic* scheduling
- Assigns priorities to tasks based on their periods
  - The shorter the period, the higher the priority
  - The *rate* (of job releases) is the inverse of the period, so jobs with higher rate have higher priority
- Widely studied and used
- For example, consider a system of 3 tasks:
  - $T_1 = (4, 1) \Rightarrow \text{rate} = 1/4$
  - $T_2 = (5, 2) \Rightarrow \text{rate} = 1/5$
  - $T_3 = (20, 5) \Rightarrow \text{rate} = 1/20$
  - Relative priorities:  $T_1 > T_2 > T_3$

# Example: Rate-Monotonic Scheduling

Time	Ready to run	Scheduled	Time	Ready to run	Scheduled
0	$J_{2,1}$ $J_{3,1}$	$J_{1,1}$	10	$J_{3,1}$	$J_{2,3}$
1	$J_{3,1}$	$J_{2,1}$	11	$J_{3,1}$	$J_{2,3}$
2	$J_{3,1}$	$J_{2,1}$	12	$J_{3,1}$	$J_{1,4}$
3		$J_{3,1}$	13		$J_{3,1}$
4	$J_{3,1}$	$J_{1,2}$	14		$J_{3,1}$
5	$J_{3,1}$	$J_{2,2}$	15		$J_{2,4}$
6	$J_{3,1}$	$J_{2,2}$	16	$J_{2,4}$	$J_{1,5}$
7		$J_{3,1}$	17		$J_{2,4}$
8	$J_{3,1}$	$J_{1,3}$	18		
9		$J_{3,1}$	19		

Low priority tasks (e.g.  $T_3$ ) are frequently preempted



# Fixed-Priority Scheduling: Deadline-Monotonic

- The *deadline-monotonic* algorithm assigns task priority according to relative deadlines – the shorter the relative deadline, the higher the priority
- When relative deadline of every task matches its period, then rate-monotonic and deadline-monotonic give identical results
- When the relative deadlines are arbitrary:
  - Deadline-monotonic can sometimes produce a feasible schedule in cases where rate-monotonic cannot
  - But, rate-monotonic always fails when deadline-monotonic fails

⇒ Deadline-monotonic preferred to rate-monotonic

# Dynamic-Priority Algorithms

- Earliest deadline first (EDF)
  - The job queue is ordered by earliest deadline
- Least slack time first (LST)
  - The job queue is ordered by least slack time
  - Two variations:
    - Strict LST – scheduling decisions are made also whenever a queued job's slack time becomes smaller than the executing job's slack time – *huge* overheads, not used
    - Non-strict LST – scheduling decisions made only when jobs release or complete
- First in, first out (FIFO)
  - Job queue is first-in-first-out by release time
- Last in, first out (LIFO)
  - Job queue is last-in-first-out by release time

[For details, see lecture 3]

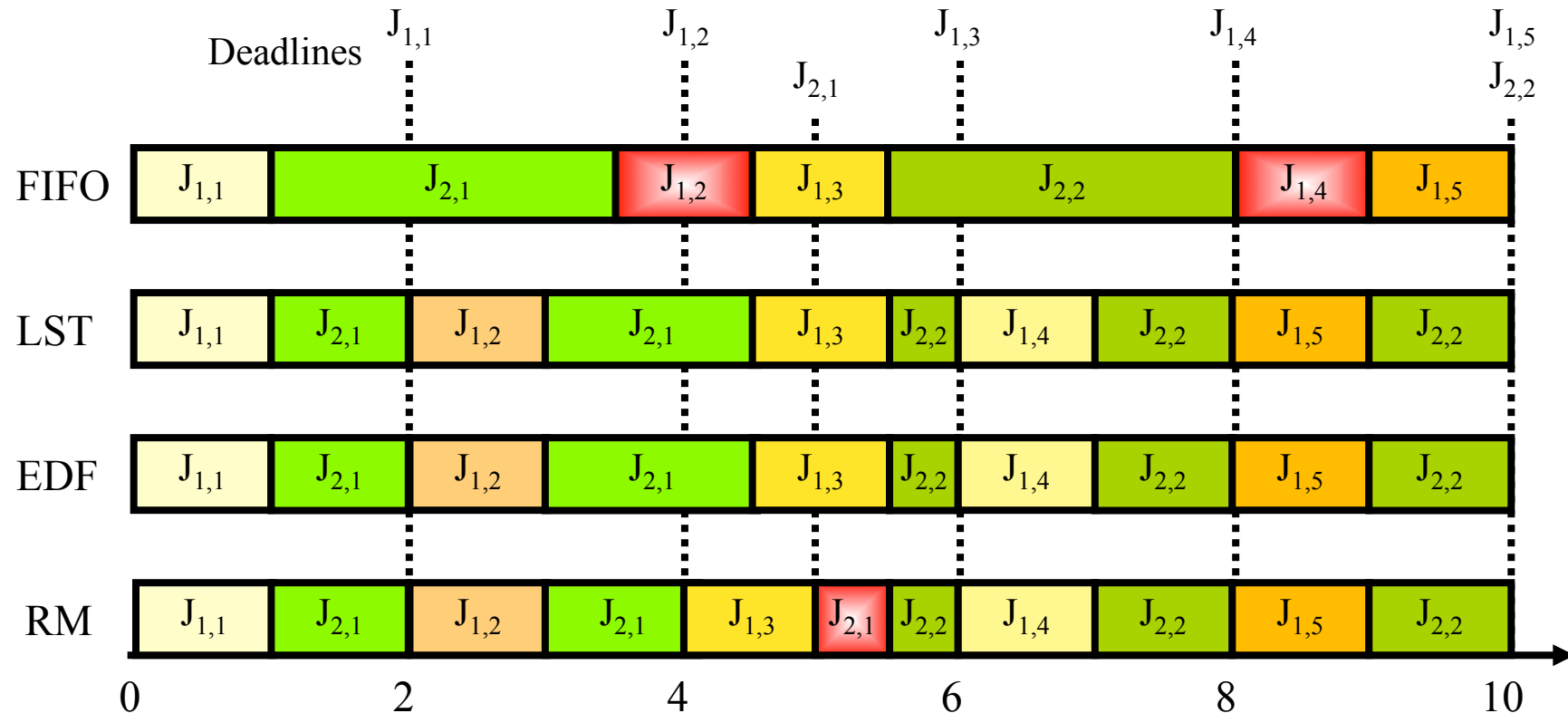
# Relative Merits

- Fixed- and dynamic-priority scheduling algorithms have different properties; neither appropriate for all scenarios
- Algorithms that do not take into account the urgencies of jobs in priority assignment usually perform poorly
  - E.g FIFO, LIFO
- The EDF algorithm gives higher priority to jobs that have missed their deadlines than to jobs whose deadline is still in the future
  - Not necessarily suited to systems where occasional overload unavoidable
- Dynamic algorithms like EDF can produce feasible schedules in cases where RM and DM cannot
  - But fixed priority algorithms often more predictable

# Example: Comparing Different Algorithms

- Compare the performance of RM, EDF, LST and FIFO scheduling
- Assume a single processor system with 2 tasks:
  - $T_1 = (2, 1)$
  - $T_2 = (5, 2.5)$   $H = 10$
- The total utilization is 1.0  $\Rightarrow$  no slack time
  - Expect some of these algorithms to lead to missed deadlines!
  - This is one of the cases where EDF works better than RM/DM

# Example: RM, EDF, LST and FIFO



- Demonstrate by exhaustive simulation that LST and EDF meet deadlines, but FIFO and RM don't

# Schedulability Tests

- Simulating schedules is both tedious and error-prone... can we demonstrate correctness without working through the schedule?
- Yes, in some cases! This is a *schedulability test*
  - A test to demonstrate that all deadlines are met, when scheduled using a particular algorithm
  - An efficient schedulability test can be used as an on-line acceptance test; clearly exhaustive simulation is too expensive for this!

# Schedulable Utilization

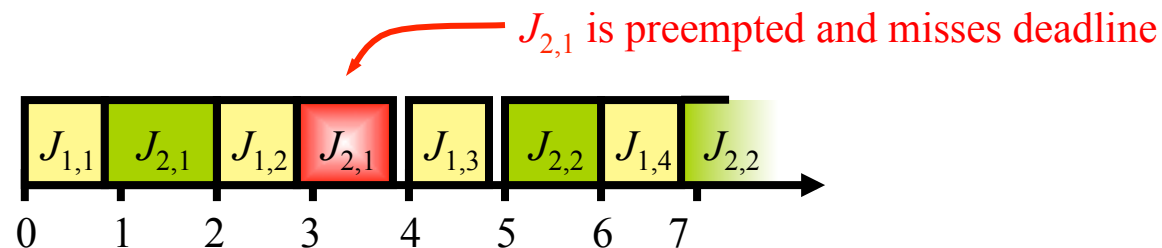
- Recall: a periodic task  $T_i$  is defined by the 4-tuple  $(\phi_i, p_i, e_i, D_i)$  with utilization  $u_i = e_i / p_i$
- Total utilization of the system  $U = \sum_{i=1}^n u_i$  where  $0 \leq U \leq 1$
- A scheduling algorithm can feasibly schedule any system of periodic tasks on a processor if  $U$  is equal to or less than the maximum schedulable utilization of the algorithm,  $U_{ALG}$ 
  - If  $U_{ALG} = 1$ , the algorithm is optimal
- Why is knowing of  $U_{ALG}$  important? It gives a schedulability test, where a system can be validated by showing that  $U \leq U_{ALG}$

# Schedulable Utilization: EDF

- Theorem: a system of independent preemptable periodic tasks with  $D_i = p_i$  can be feasibly scheduled on one processor using EDF if and only if  $U \leq 1$ 
  - $U_{EDF} = 1$  for independent, preemptable periodic tasks with  $D_i = p_i$   
[Expected since EDF proved optimal in lecture 3 – see the book for proof]
  - Corollary: result also holds if deadline longer than period:  $U_{EDF} = 1$  for independent preemptable periodic tasks with  $D_i \geq p_i$
- Notes:
  - Result is independent of  $\phi_i$
  - Result can also be shown to apply for LST

# Schedulable Utilization: EDF

- What happens if  $D_i < p_i$  for some  $i$ ? The test doesn't work...
  - E.g.  $T_1 = (2, 0.8)$ ,  $T_2 = (5, 2.3, 3)$



- However, there is an alternative test:
  - The density of the task,  $T_i$ , is  $\delta_i = e_i / \min(D_i, p_i)$
  - The density of the system is  $\Delta = \delta_1 + \delta_2 + \dots + \delta_n$
  - Theorem: A system  $T$  of independent, preemptable periodic tasks can be feasibly scheduled on one processor using EDT if  $\Delta \leq 1$ .
- Note:
  - This is a sufficient condition, but not a necessary condition – i.e. a system is *guaranteed* to be feasible if  $\Delta \leq 1$ , but *might* still be feasible if  $\Delta > 1$  (would have to run the exhaustive simulation to prove)

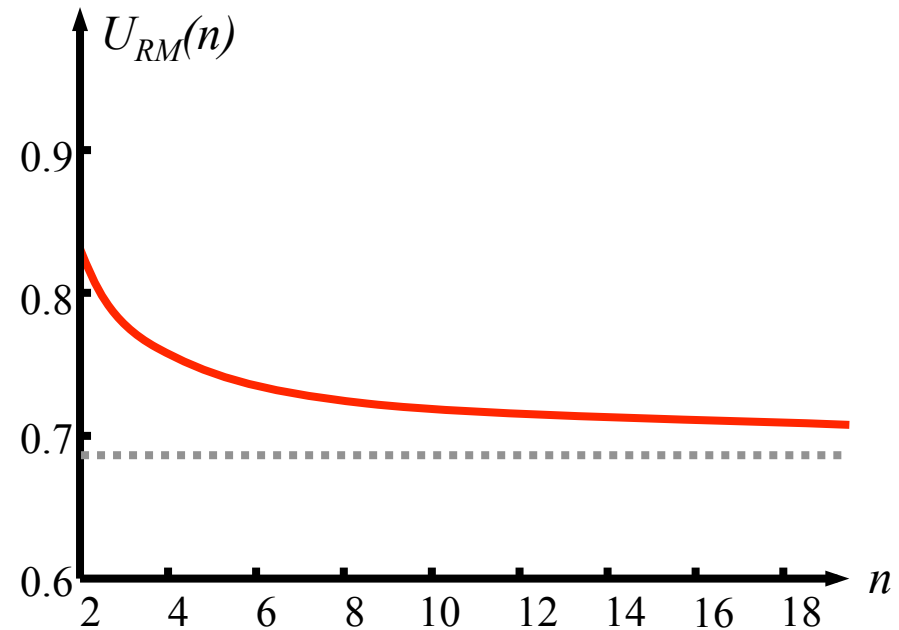
# Schedulable Utilization: EDF

- How can you use this in practice?
  - Assume using EDF to schedule multiple periodic tasks, known execution time for all jobs
  - ⇒ Choose the periods for the tasks such that the schedulability test is met
- Example: a simple digital controller, as discussed in lecture 1
  - Control-law computation task,  $T_1$ , takes  $e_1 = 8$  ms, sampling rate is 100 Hz (i.e.  $p_1 = 10$  ms)
    - ⇒  $u_1$  is 0.8
    - ⇒ the system is guaranteed to be schedulable
  - Want to add a built-in self test task,  $T_2$ , taking 50ms - will the system still work?
    - $U = u_1 + u_2 \leq 1.0$  where we know that  $u_1 = 0.8$ ,  $u_2 = 50\text{ms} / p_2$
    - To be schedulable  $\Rightarrow u_2 \leq 0.2 \Rightarrow p_2 \geq 250\text{ms}$
    - As long as the period for this task is 250 ms or more, the total utilization remains  $\leq 1$  and the system can be scheduled

# Schedulable Utilization of RM

- Theorem: a system of  $n$  independent preemptable periodic tasks with  $D_i = p_i$  can be feasibly scheduled on one processor using RM if and only if  $U \leq n \cdot (2^{1/n} - 1)$

- $U_{RM}(n) = n \cdot (2^{1/n} - 1)$
- For large  $n \rightarrow \ln 2$   
(i.e.  $n \rightarrow 0.69314718056\dots$ )
- [Proof in book - complicated!]



- $U \leq U_{RM}(n)$  is a sufficient, but not necessary, condition – i.e. a feasible rate monotonic schedule is *guaranteed* to exist if  $U \leq U_{RM}(n)$ , but *might* still be possible if  $U > U_{RM}(n)$

# Schedulable Utilization of RM

- What happens if the relative deadlines for tasks are not equal to their respective periods?
- Assume the deadline is some multiple  $v$  of the period:  $D_k = v \cdot p_k$
- It can be shown that:

$$U_{RM}(n, v) = \begin{cases} v & 0 \leq v \leq 0.5 \\ n((2v)^{1/n} - 1) + 1 - v & \text{for } 0.5 \leq v \leq 1 \\ v(n-1) \left[ \left( \frac{v+1}{v} \right)^{1/n-1} - 1 \right] & v = 2, 3, \dots \end{cases}$$

# Schedulable Utilization of RM

$n$	$v = 4.0$	$v = 3.0$	$v = 2.0$	$v = 1.0$	$v = 0.9$	$v = 0.8$	$v = 0.7$	$v = 0.6$	$v = 0.5$
2	0.944	0.928	0.898	0.828	0.783	0.729	0.666	0.590	0.500
3	0.926	0.906	0.868	0.779	0.749	0.708	0.656	0.588	0.500
4	0.917	0.894	0.853	0.756	0.733	0.698	0.651	0.586	0.500
5	0.912	0.888	0.844	0.743	0.723	0.692	0.648	0.585	0.500
6	0.909	0.884	0.838	0.734	0.717	0.688	0.646	0.585	0.500
7	0.906	0.881	0.834	0.728	0.713	0.686	0.644	0.584	0.500
8	0.905	0.878	0.831	0.724	0.709	0.684	0.643	0.584	0.500
9	0.903	0.876	0.829	0.720	0.707	0.682	0.642	0.584	0.500
$\infty$	0.892	0.863	0.810	0.693	0.687	0.670	0.636	0.582	0.500

$D_i > p_i \Rightarrow$  Schedulable  
utilization increases

$$D_i = p_i$$

$D_i < p_i \Rightarrow$  Schedulable  
utilization decreases

# Summary

Key points:

- Different priority scheduling algorithms
  - Earliest deadline first, least slack time, rate monotonic, deadline monotonic
  - Each has different properties, suited for different scenarios
- Scheduling tests, concept of maximum schedulable utilization
  - Examples for different algorithms

Tomorrow: practical factors, more schedulability tests...