ECE3140 / CS3420
Embedded Systems

Real-Time Scheduling Algorithms for Periodic Tasks

José F. Martínez

Cornell University

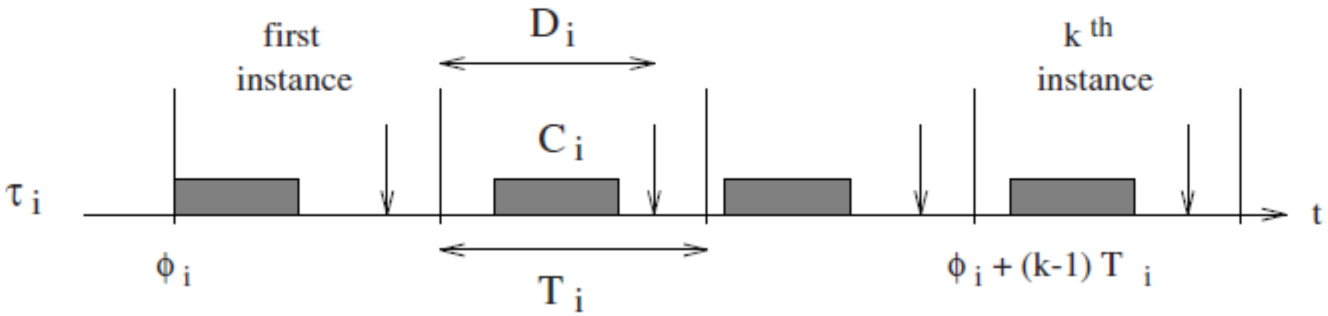
Outline: Scheduling for Periodic Tasks

Scheduling algorithms for periodic real-time tasks

- Review: periodic tasks
- Timeline scheduling
- Rate Monotonic (RM) scheduling
  - Algorithm
  - Schedulability analysis
- RM vs. EDF
- Reference
  - Chapter 4, “Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications” by Giorgio C. Buttazzo (Free electronic copy through Cornell library)

Periodic Tasks

- Periodic tasks are those that have jobs that repeat at a regular interval in time



Source: ‘Hard Real-Time Computing Systems’ by Buttazzo

- For each periodic task τ_i:
  - Each job $\tau_{i,k}$ is activated at $r_{i,k} = \Phi_i + (k-1)T_i$
    - Φ_i represents the phase of a task: $r_{i,k}$
  - Each job $\tau_{i,k}$ has a deadline $d_{i,k} = r_{i,k} + D_i$

Example: Video Streaming

- Multiple periodic tasks with different periods need to run
- Task 1: download a video stream from a server
- Task 2: decode and output video
  - H.264: typically, 30 frames per second
- Task 3: decode and output audio
  - AAC: sampling frequency from 8 to 96kHz

Assumptions

- **A1.** The instances of a periodic task τ_i are regularly activated at a constant rate. The interval T_i between two consecutive activations is the *period* of the task.
- **A2.** All instances of a periodic task τ_i have the same worst-case execution time C_i.
- **A3.** All instances of a periodic task τ_i have the same relative deadline D_i, which is equal to the period T_i.
- **A4.** All tasks in Γ are independent; that is, there are no precedence relations and no resource constraints.
- **A5.** No task can suspend itself, for example on I/O operations.
- **A6.** All overheads in the kernel are assumed to be zero.

Timeline Scheduling

- Classic technique, also called cyclic scheduling
- Used for decades in military systems, navigation, and monitoring
- Examples
    - Air traffic control
    - Boeing 777
    - Space shuttle

Timeline Scheduling: Approach

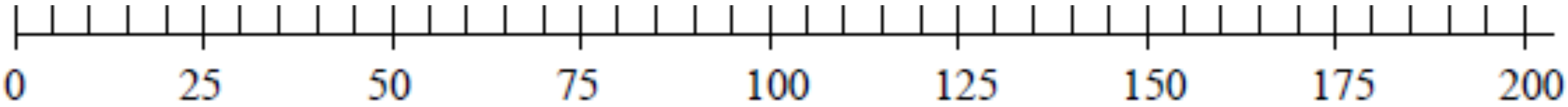
- The time axis is divided into intervals of equal length, called *time slots* or *minor cycles*

- Each task is *statically* allocated to a time slot in order to meet its desired request rate
  - Multiple tasks can be allocated to one time slot as long as their combined execution time is less than the time slot

- Timers are used to activate execution in each slot
  - The schedule is hardcoded in a program

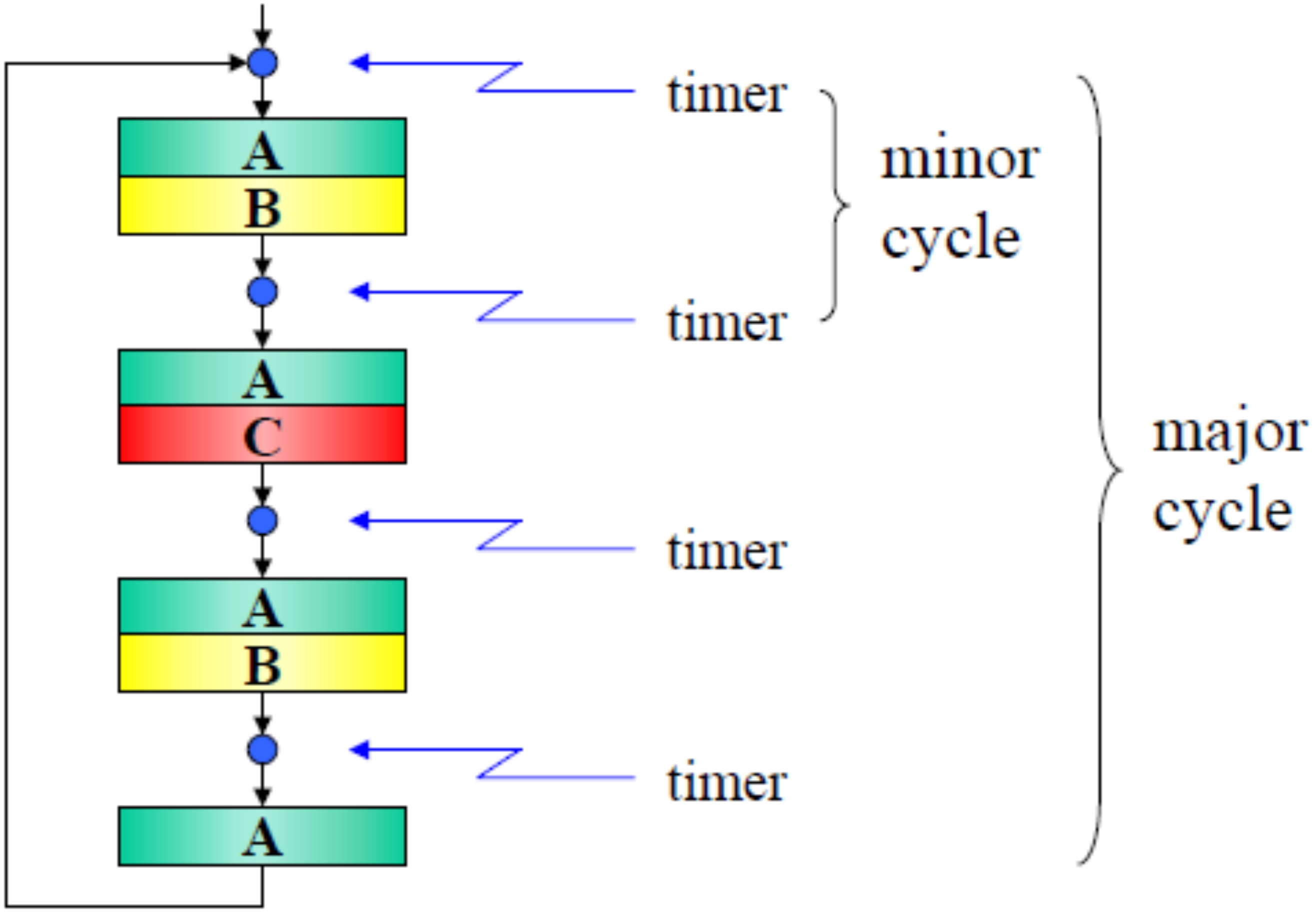
Timeline Scheduling: Example

Task	T_i	C_i
A	25ms	10ms
B	50ms	10ms
C	100ms	10ms



- Minor cycle: Δ = GCD of the periods = 25ms
- Major cycle: T = LCM of the periods = 100ms
  - The minimum interval after which the schedule repeats itself

Implementation



Source: Lecture slides by G. Buttazzo

Advantages and Disadvantages

- Advantages
  - Simple implementation (no real-time OS is required)
  - Low run-time overhead (no scheduler)
  - Some jitter can be tolerated

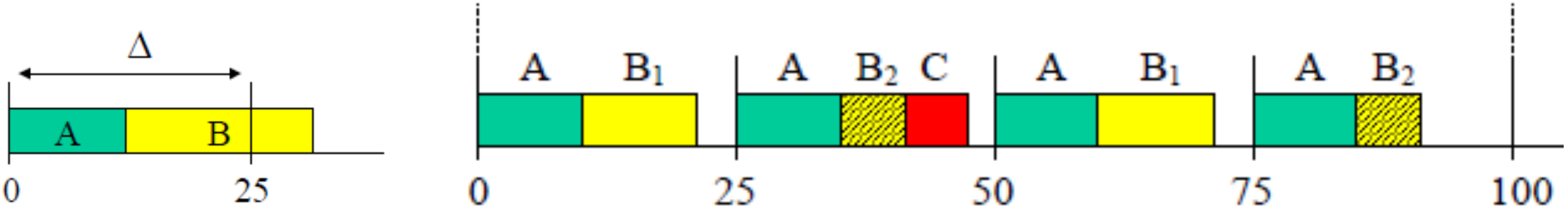
- Disadvantages
  - Not robust when a task overruns and does not finish by the end of a time slot
  - Difficult to expand the schedule
  - Not easy to handle aperiodic activities

Overrun and Expandibility

- What happens if we have an overrun?
  - If the task continues, there can be a domino effect
  - If the task is aborted, the system could be in an inconsistent state

- If one or more tasks need to be updated, we may need to re-design the whole schedule

- Example: B is updated to be longer
  - Common approach: split the task into two sub-tasks



Frequency Change Example

- If the frequency of some task is changed, the impact can be even more significant

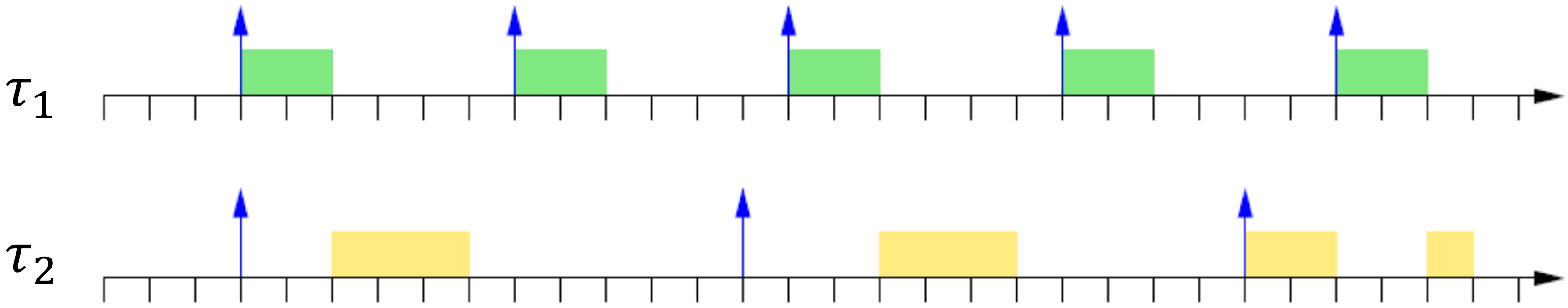
Task	T_i	New T_i	C_i
A	25ms	25ms	10ms
B	50ms	**40ms**	10ms
C	100ms	100ms	10ms

- Original
  - Minor cycle: Δ = 25ms
  - Major cycle: T = 100ms
- New
  - Minor cycle: Δ = 5ms
  - Major cycle: T = 200ms

Recap: Rate Monotonic Scheduling

- Execute tasks with priority scheduling
- Each task is assigned a *fixed* priority proportional to its rate

$$p_i \propto \frac{1}{T_i}$$



- Verify the feasibility of the schedule using analytical techniques

How to Determine Schedulability?

- Each task uses the processor for a fraction of time:

$$U_i = \frac{C_i}{T_i}$$

- The total processor utilization is given by:

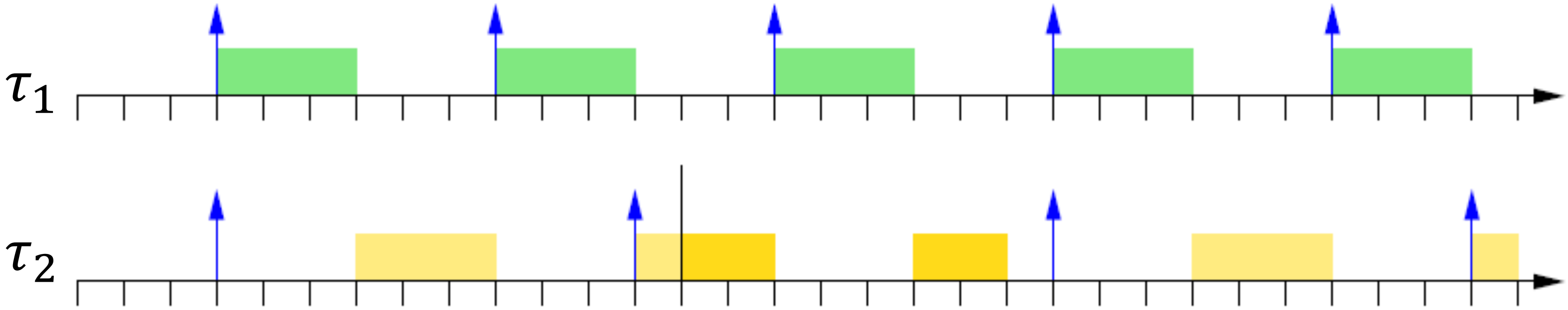
$$U_{cpu} = \sum_i U_i$$

- U_{cpu} measures the processor load

- If $U_{cpu} > 1$, the processor is overloaded and the tasks set cannot be all scheduled

Infeasible RM Schedule

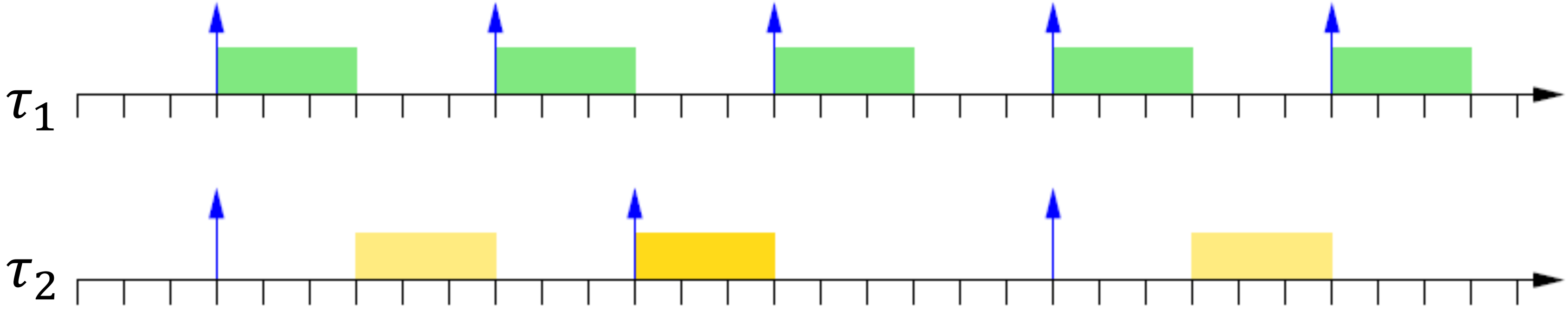
- $U_{cpu} < 1$ does not necessarily mean that there exists a feasible schedule



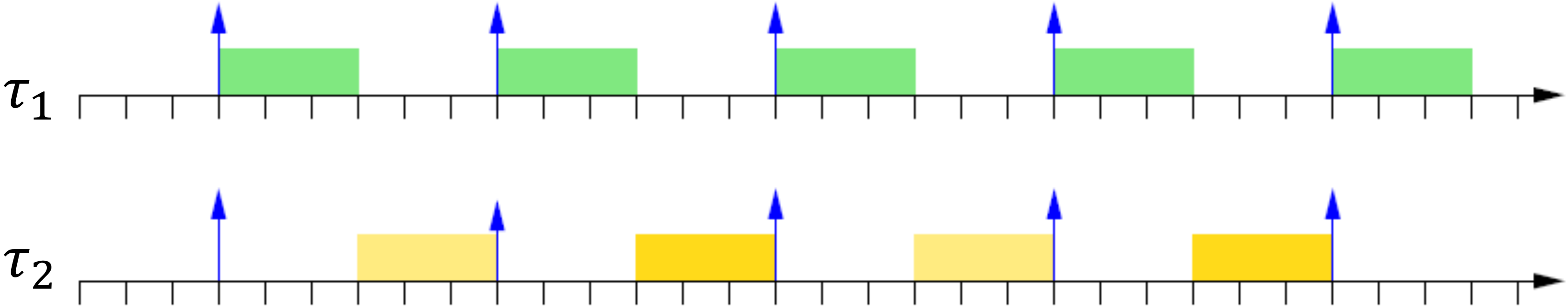
- Here, $U_{cpu} < 1$ but the RM schedule is infeasible
  - $U_1 = \frac{3}{6},\ U_2 = \frac{4}{9}$

Utilization Upper Bound?

- Increasing C_1 or C_2 causes missed deadlines ($U_{cpu} = 0.833$)



- Here $U_{cpu} = 1$ and the schedule is feasible.



A Sufficient Condition

Liu/Layland result (1973): for n periodic tasks, if

$$U_{cpu} \leq n(2^{\frac{1}{n}} - 1)$$

then RM will produce a feasible schedule

- In the limit $n \rightarrow \infty$, RHS is $\ln 2$ (~69%)
- RM may still be able to produce a feasible schedule for a set of periodic tasks with a higher utilization
  - Not a necessary condition

Optimality of RM

RM is ***optimal*** among ***all fixed priority*** algorithms

- If there exists a fixed priority assignment which leads to a feasible schedule, then RM produces a feasible schedule

- If a task set is not schedulable by RM, then it cannot be scheduled by any fixed priority assignment

Optimality of EDF

EDF is ***optimal*** among ***all*** algorithms

- If there exists a feasible schedule for a task set, then EDF will generate a feasible schedule

- A set of *n* periodic tasks is schedulable by the EDF algorithm if and only if

$$U_{cpu} = \sum_{i=1}^{n} U_i \leq 1$$

RM vs. EDF

- RM is easy to implement on a commercial kernel with a priority scheduler, but no support for periods or deadlines
  - RM: simply assign a fixed priority to each task
  - EDF: requires dynamically adjusting the priority. Mapping from a deadline to a priority also adds complexity
- If implemented in the kernel, both RM and EDF have similar implementation complexity
  - RM can be implemented with a smaller number of queues if a small number of priority levels are sufficient
- EDF often has lower run-time overhead than RM
  - EDF needs to re-assign the deadline on each job (higher scheduler overhead), but usually leads to less context switches
- If a system becomes overloaded, any task except for the highest priority one may miss a deadline in RM
  - In EDF, any task may miss a deadline