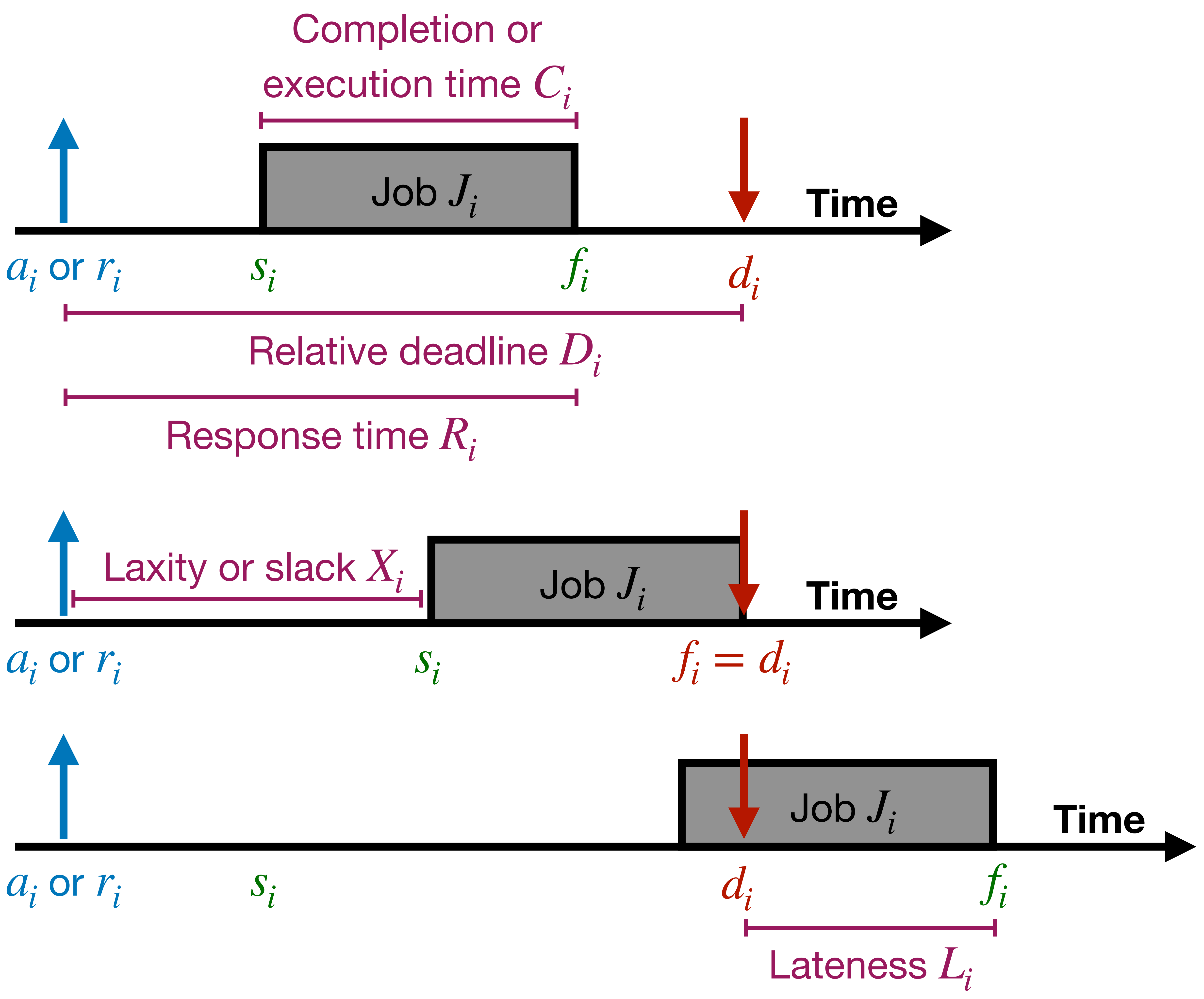
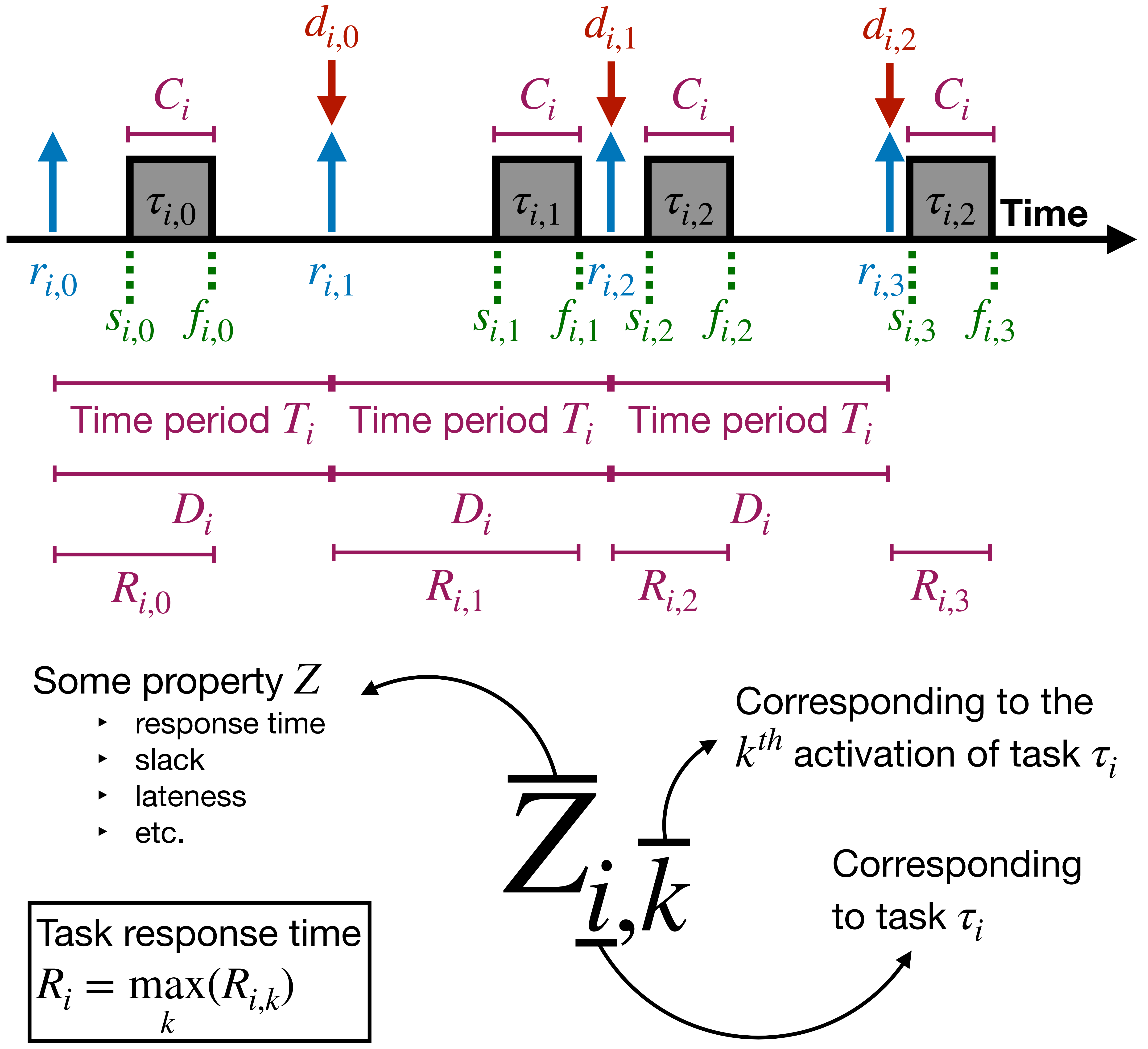
Periodic Task Scheduling

CPEN 432 Real-Time System Design

Arpan Gujarati
University of British Columbia

Aperiodic Job vs. Periodic Task





Scheduling Objectives

- Control applications consist of multiple concurrent periodic tasks
  - E.g., sensory data acquisition, low-level servoing, control loops, system monitoring
  - Each task may have unique characteristics (time period, execution time, etc.)
  - OS must guarantee each task is regularly activated at its proper rate
    - and completed within its deadline (could be different from its period)

- Given a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ consisting of n tasks
  - can we find a scheduling algorithm A?
    - such that when all tasks are integrated on a platform consisting of m processors
      - every job of every task is guaranteed to not miss its deadline!

- Given τ, can we find A such that $\forall \tau_i \in \tau : ? \leq ?$

Assumptions

A1: All jobs of τ_i are regularly activated at a constant frequency of $1/T_i$

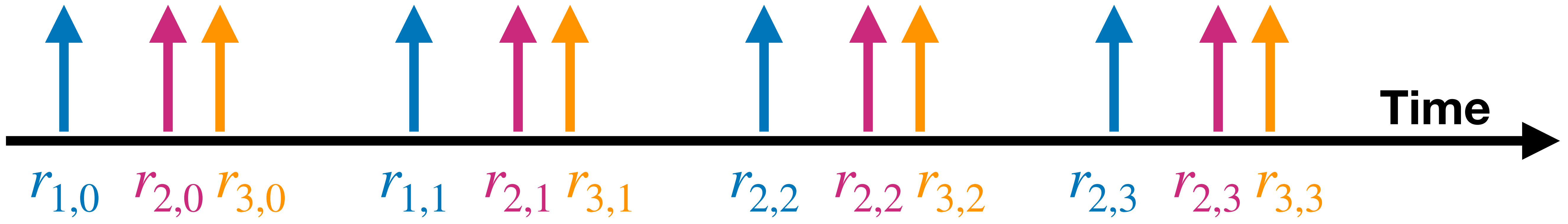
A2: All jobs of τ_i have the same worst-case execution time C_i

A3. All jobs of τ_i have the same relative deadline $D_i = T_i$

A4. All tasks in τ are independent (no dependencies, no shared resources)

Note

- The tasks **need not be released synchronously**
  - E.g., it is possible that $r_{1,0} \neq r_{2,0} \neq \ldots \neq r_{n,0}$



- The tasks can be **preempted** in between

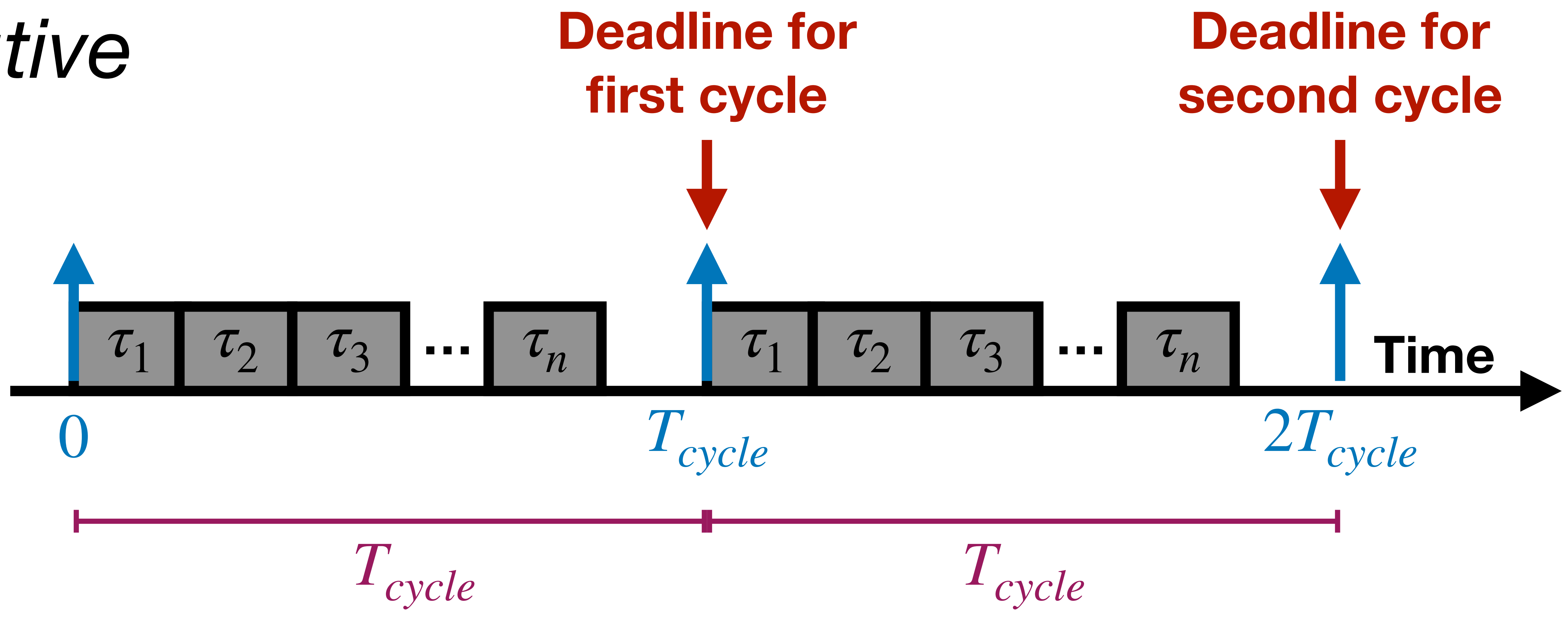
Next few lectures …

- Four scheduling algorithms
  - Timeline Scheduling (TS)
  - Rate Monotonic (RM)
  - Earliest Deadline First (EDF)
  - Deadline Monotonic (DM)
- Schedulability analyses (or guarantee tests)
- Optimality proofs (if any)

Timeline Scheduling

Overview

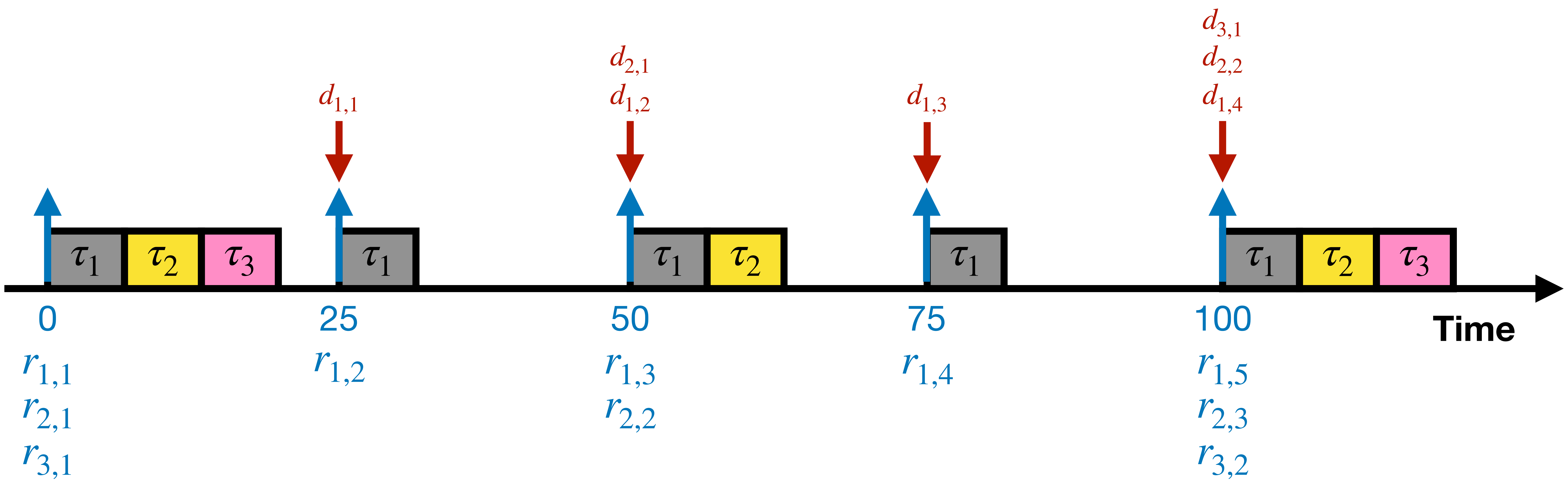
- OS runs one simple *cyclic executive*
  - Single large periodic task
  - Executes with time period T_{cycle}



- While (true)
  - $t_{start} =$ Now
  - If $condition_1$ holds: Execute a job of τ_1
  - If $condition_2$ holds: Execute a job of τ_2
  - …
  - If $condition_n$ holds: Execute a job of τ_n
  - Wait until $t_{start} + T_{cycle}$

Example #1

ID	T	C
1	25 ms	6 ms
2	50 ms	6 ms
3	100 ms	6 ms

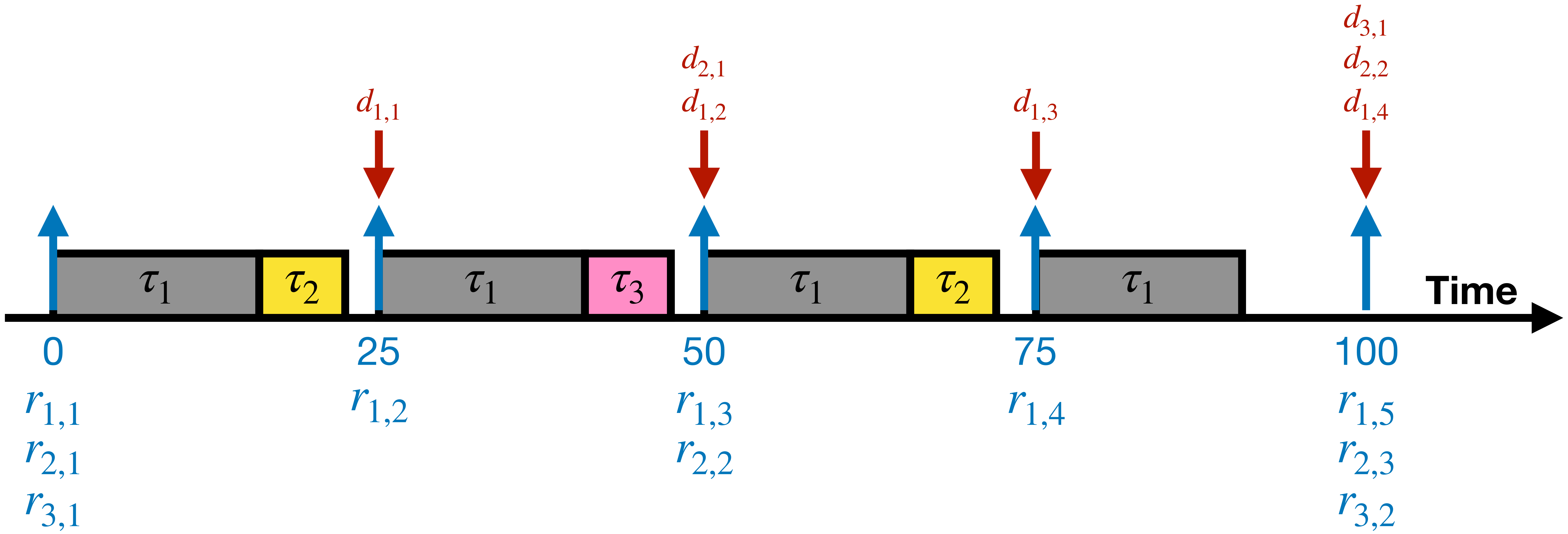


- Initialize
  - $T_{cycle} = gcd_i(T_i) = 25\,ms$
  - $counter = 0$
- While (true)
  - t_{start} = Now
  - Execute a job of τ_1
  - If $counter \,\%\, 2 == 0$: Execute a job of τ_2
  - If $counter \,\%\, 4 == 0$: Execute a job of τ_3
  - $counter++$
  - wait until $t_{start} + T_{cycle}$

Schedulability analysis: $C_1 + C_2 + C_3 \leq 25\ ms$

Example #2

ID	T	C
1	25 ms	15 ms
2	50 ms	6 ms
3	100 ms	6 ms



- $T_{cycle} = gcd_i(T_i) = 25\ ms$
- $counter = 0$
- While (true)
  - t_{start} = Now
  - Execute a job of τ_1
  - If $counter \% 2 == 0$: Execute a job of τ_2
  - If $counter \% 4 == 1$: Execute a job of τ_3
  - $counter++$
  - wait until $t_{start} + T_{cycle}$

Schedulability analysis:

- $C_1 + C_2 \leq 25\ ms$
- $C_1 + C_3 \leq 25\ ms$

Example #3

ID	T	C
1	25 ms	15 ms
2	40 ms	6 ms
3	100 ms	6 ms

Rate Monotonic Scheduling

Overview

- RM is a **fixed-priority** scheduling algorithm
  - Each task is assigned a priority beforehand

- RM assigns priorities based on **task frequency**
  - Higher frequency (smaller time period) $\Longrightarrow$ Higher priority

- Famous result by Liu and Layland [1973]
  - RM is **optimal** among all fixed-priority algorithms
    - i.e., no fixed-priority algorithm can schedule a task set that cannot be scheduled by RM
    - i.e., if any fixed-priority algorithm can schedule a task set, RM can also schedule the task set

RM Optimality Proof [1/n]

- **Critical instant** of a task
  - Arrival time that produce the **largest** task response time

- **Theorem:** The critical instant for any task occurs whenever the task is released simultaneously with all higher-priority tasks
  - **Corollary:** It suffices to check for a task's schedulability at its critical instant

RM Optimality Proof [2/n]

- For simplicity
  - Let $\tau = \{\tau_1, \tau_2\}$ such that $T_1 < T_2$
- Only two fixed-priority assignments possible
  - RM: τ_1 is assigned the higher priority
  - Algorithm A: τ_2 is assigned the higher priority
- Recall RM optimality statement
  - If any fixed-priority algorithm can schedule a task set, RM can also schedule the task set
  - In this case, if A can schedule $\tau = \{\tau_1, \tau_2\}$, RM can also schedule $\tau = \{\tau_1, \tau_2\}$
- Proof sketch
  - Step 1: Algorithm A can schedule $\tau \implies$ Predicate P_1
  - Step 2: For RM to schedule τ, we require another predicate P_2
    - i.e., Predicate $P_2 \implies$ RM can schedule τ
  - Step 3: Show that $P_1 \implies P_2$

RM Optimality Proof [3/n]

- Step 1: Algorithm A can schedule $\tau \implies$ Predicate P_1
  - As per A, τ_2 is assigned the higher priority, so it will trivially be schedulable
  - Let's consider the critical instant to see if τ_1 is also schedulable (despite its lower priority)

RM Optimality Proof [4/n]

- Step 2: Predicate $P_2 \implies$ RM can schedule τ
  - As per RM, τ_1 is assigned the higher priority, so it will trivially be schedulable
  - Let's consider the critical instant to see if τ_2 is also schedulable (despite its lower priority)

RM Optimality Proof [5/n]

- Step 3: Show that $P_1 \implies P_2$
- Here's P_1 …
- Here's P_2 …

RM Optimality Proof [5/n]

- We showed that if $\tau = \{\tau_1, \tau_2\}$ such that $T_1 < T_2$ is schedulable by an arbitrary priority assignment, it is also schedulable by RM
- What if τ consists of more than two tasks?
  - Textbook: *"This result can easily be extended to a set of n periodic tasks."* :-)
  - Expect a question in the homework assignment
    - See Liu and Layland's 1973 paper for reference
    - (soft copy available at https://cpen432.github.io/readings/)

RM Schedulability Test

- Processor utilization factor
  - Fraction of processor time spent executing tasks in $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

- By simply checking the utilization, can we say if RM can schedule it?
  - That is, if $U \leq U_{limit}$, irrespective of the task parameters, τ is schedulable by RM

- Examples
  - $U_{limit} = 1.0?$
  - $U_{limit} = 0.9?$