SES

Chapter 10: Real-Time Embedded Systems

Prof. Dr.-Ing. Bernd-Christian Renner

Institute smartPORT
Hamburg University of Technology
TUHH

Contents

1

Definitions and Examples

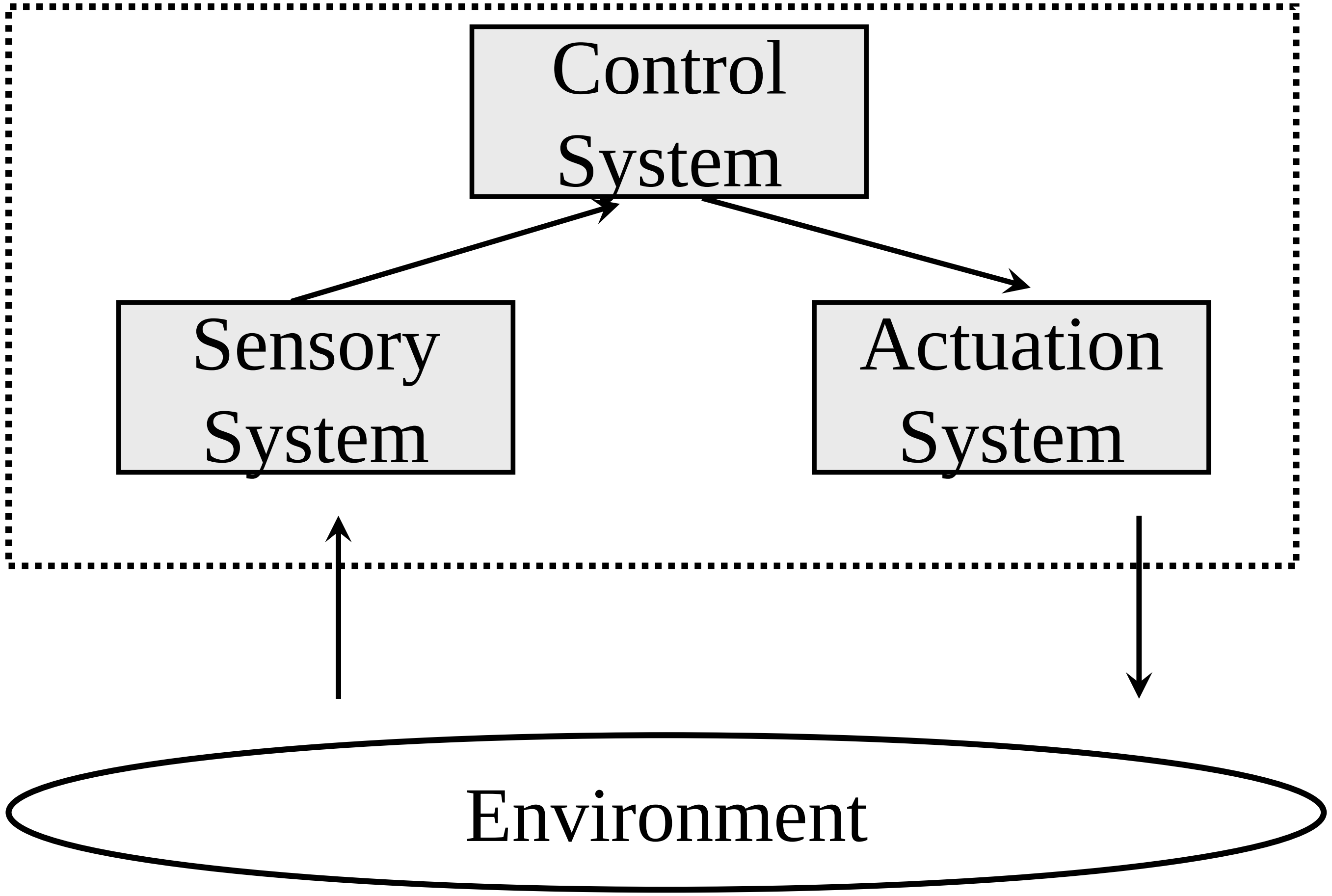
What is a Real-Time System?

- Real-time system
  - Information processing activity which has to respond to externally generated input stimuli within a specified period otherwise risks severe consequences, including failure
- Logical correctness of real-time system is based on
  - correctness of outputs and
  - their timelines

Related Notions

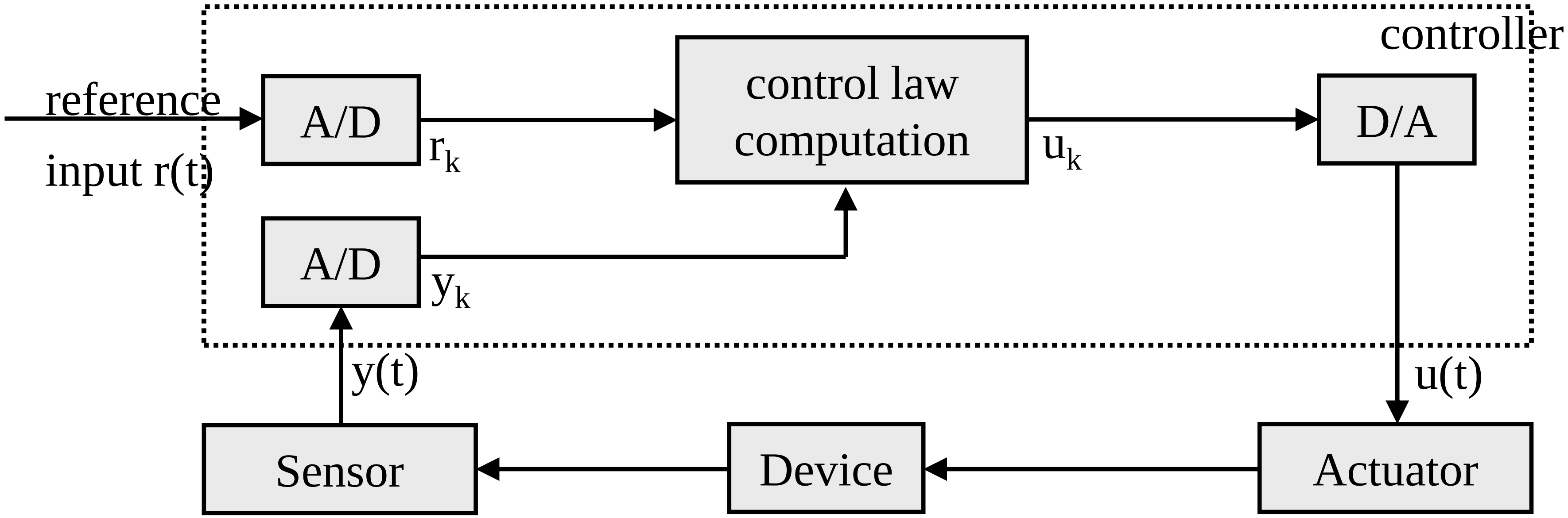
- Reactive system
  - Continuous interaction with the environment (as opposed to information processing)
- Embedded system
  - Computer system encapsulated in its environment (device it controls), combination of computer hardware and software, dedicated to specific purpose
- Safety-critical system
  - A failure may cause injury, loss of lives, significant financial loss

Block Diagram of generic Real-Time System



Example 1: Digital Process Control

- Controlling device with actuator, based on sampled sensor data
  - y(t): measured state of device
  - r(t): desired state of the device
  - Calculate control output u(t) as a function of y(t), r(t)



Example 1: Digital Process Control

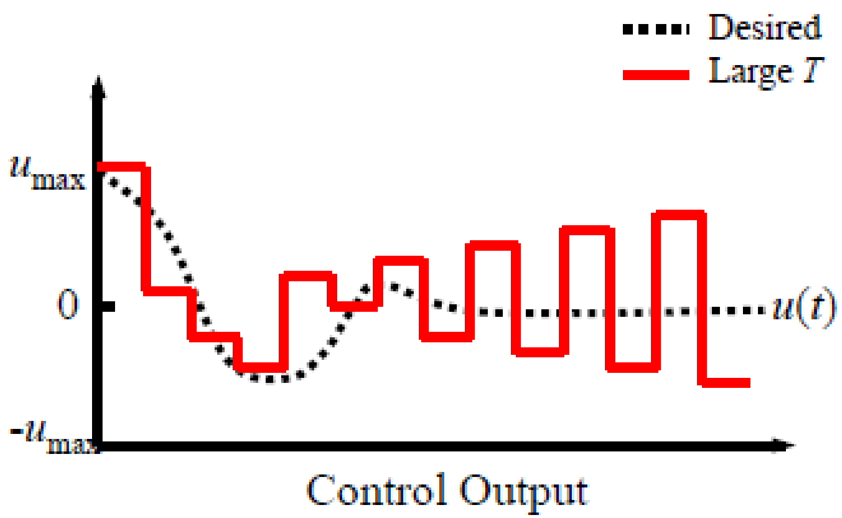
- Pseudo-code for controller:

```
set timer to interrupt periodically with period T;
at timer interrupt, do
   analogue-to-digital conversion of y(t) to get y_k;
   analogue-to-digital conversion of r(t) to get r_k;
   compute control output u_k based on reference r_k and y_k;
   digital-to-analogue conversion of u_k to get u(t);
end do;
```

- Sampling period T
  - time between consecutive measurements of $y(t)$, $r(t)$
- Hardware must **guarantee** that all conversions and processing is possible in time T
- How can this be guaranteed?

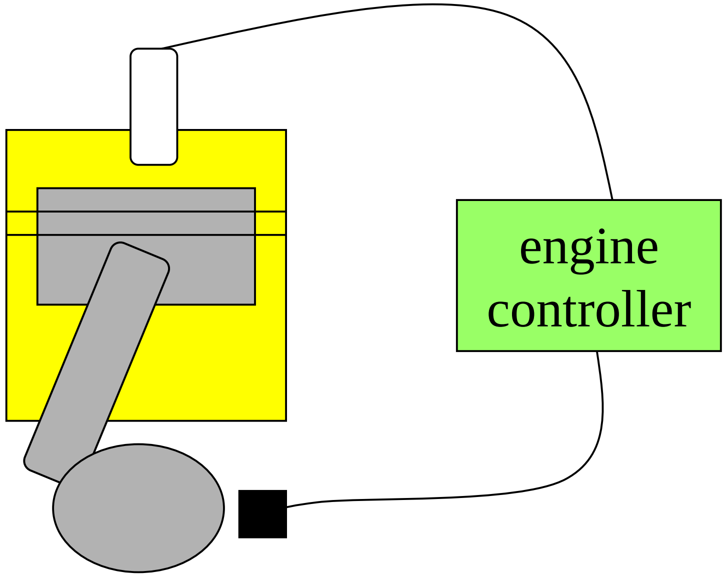
Example 1: Digital Process Control

- Effective control of device depends on
  - Correct control law computation and reference input
  - Accuracy of the sensor measurements:
    - Resolution of sampled data (i.e. bits per sample)
  - Length of T (i.e. samples per second, 1/*T*)
- Small *T* better approximates analogue behavior
  - Small *T* requires
    - more processor-time
    - better ADC hardware
  - Downside:
    - possibly lower resolution
- Large T results in oscillation



Multi-rate systems

- Tasks may be synchronous or asynchronous
- Synchronous tasks may recur at different rates
- Processes run at different rates based on computational needs of the tasks
- Example 2: Engine control
  - spark control
  - crankshaft & oxygen sensing
  - fuel/air mixture
  - multi mode operation: warm-up, cruise, climbing steep hills



Typical Rates in Engine Controllers

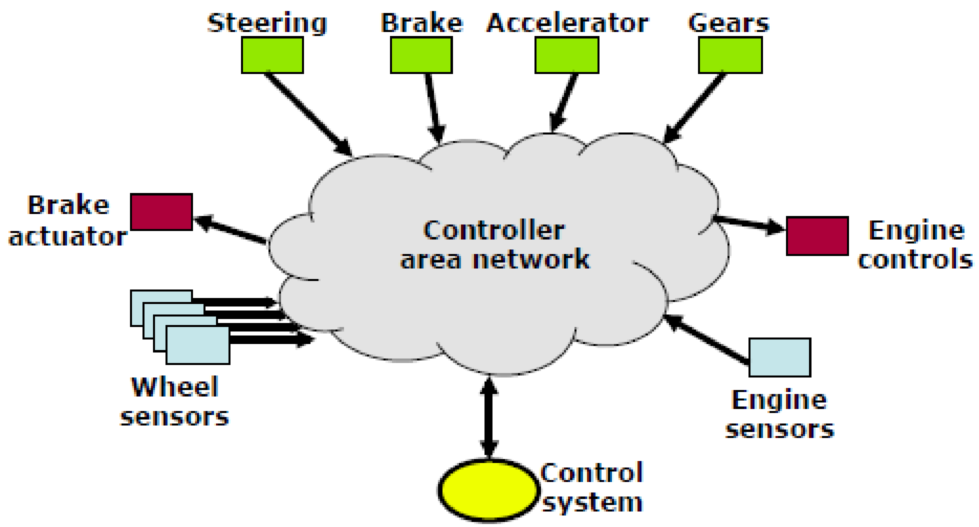
Variable	Update period (ms)
Engine spark timing	2
Throttle (Gaspedal)	2
Air flow	4
Battery voltage	4
Fuel flow	10
Recycled exhaust gas	25
Status switches	20
Air temperature	400
Barometric pressure	1000
Spark (dwell)	1
Fuel adjustment	8
Carburetor (Vergaser)	25

Real-Time Communications

- Real-time systems are increasingly distributed, including communication networks
  - Control loop may include a communication step
  - System may depend on network stimuli
- Not only does a system need to run a control law with time constraints, it must also
  - schedule communications
  - send and receive messages according to deadlines

Example 3: Drive by Wire

- All data must be delivered reliably
  - Bad if you turn steering wheel, and nothing happens
- Systems are prioritized
  - Anti-lock brakes have a faster response time than driver, so prioritize to ensure car doesn’t skid
  - Commands from control system have highest priority, then sensors and actuators
- Network must schedule and prioritize communications



2

Soft and Hard Real Time

Soft and Hard Real Time

- Hard RT System
  - missing a deadline may cause failure of system, e.g., aircraft control, nuclear plant control
    - Side airbag in car, reaction in <10 ms
    - Wing vibration of airplane, sensing every 5 ms
- Soft RT System
  - meeting a deadline is highly desirable for performance reasons, e.g., multimedia application, booking system, displaying status information

Real time vs best effort:
- best effort = low average time
- real time = predictability, bounded worst case time

Soft and Hard Real Time

- Most systems contain both hard and soft deadlines
- Essential for hard real time: Upper bounds of execution times of all tasks must be known at compile time
  - Commonly called **Worst-Case Execution Time** (WCET)
- Further notion: Firm deadline
  - Missing a deadline makes task useless (similar to hard deadline), however deadline may be missed occasionally (similar to soft deadline)
- Tasks may have cost functions associated with them for missing their deadline
  - System tries to minimize costs

Predictability

- Predictability: Correctness of prediction of system’s state
- Predictability
  - is one of the most important requirements
  - but also one of the most difficult requirements to achieve, in particular in modern processors:
    - cache, pipelines, branch prediction, interrupt handling
    - memory management
    - priority inversion
    - difficult to calculate WCET
    - ...

Types of Real-Time Applications

- Purely periodic
  - Every task executes periodically
  - Demands in (computing, communication, and storage) resources do not vary significantly from period to period
  - Example
    - Most digital controllers and real-time monitors
- Mostly periodic
  - Most tasks execute periodically
  - System must also respond to some external events (fault recovery and external commands) asynchronously
  - Example
    - Modern avionics and process control systems

Types of Real-Time Applications

- Asynchronous: Mostly predictable
  - Most tasks are not periodic
  - Time between consecutive executions of a task may vary, but variance is bounded
- Asynchronous: Unpredictable
  - Applications that react to asynchronous events and have tasks with unpredictable high variance in run-time
  - Variations have neither bounded ranges nor known statistics

3

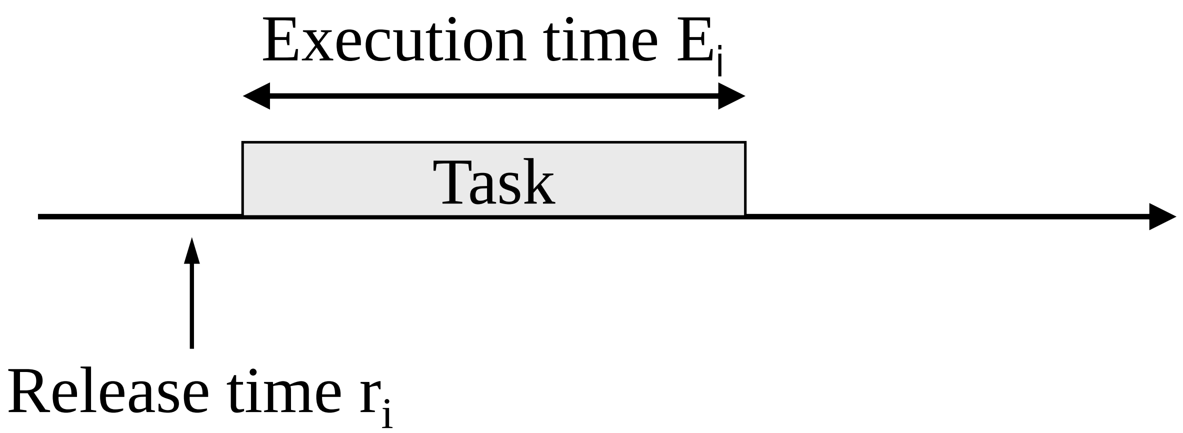
Basic Notions

Basic Notions

- **Task** (a.k.a. process, job)
  - Something that needs to be done (sequentially)
- Given a set of tasks $J = \{J_1, J_2, \ldots, J_n\}$
- **Schedule**
  - Assignment of execution times for tasks such that all tasks are executed until completion
- Formal definition of schedule σ:
  - $\sigma: T \rightarrow J$ where $\sigma(t)$ denotes task which is executed at time t
  - If $\sigma(t) = 0$ then processor is called **idle**
- **Context switch**
  - Times when σ changes its value
- **Preemptive schedule**
  - Schedule in which running task can be suspended at any time

Basic Notions

- A schedule is said to be **feasible**, if all tasks can be completed according to a set of specified constraints, e.g., deadlines
- A set of tasks is said to be **schedulable**, if there exists at least one feasible schedule
- **Release time** (a.k.a. arrival time) r_i
  - time at which a task becomes ready for execution
- **Execution time** E_i
  - time necessary for executing task without interruption

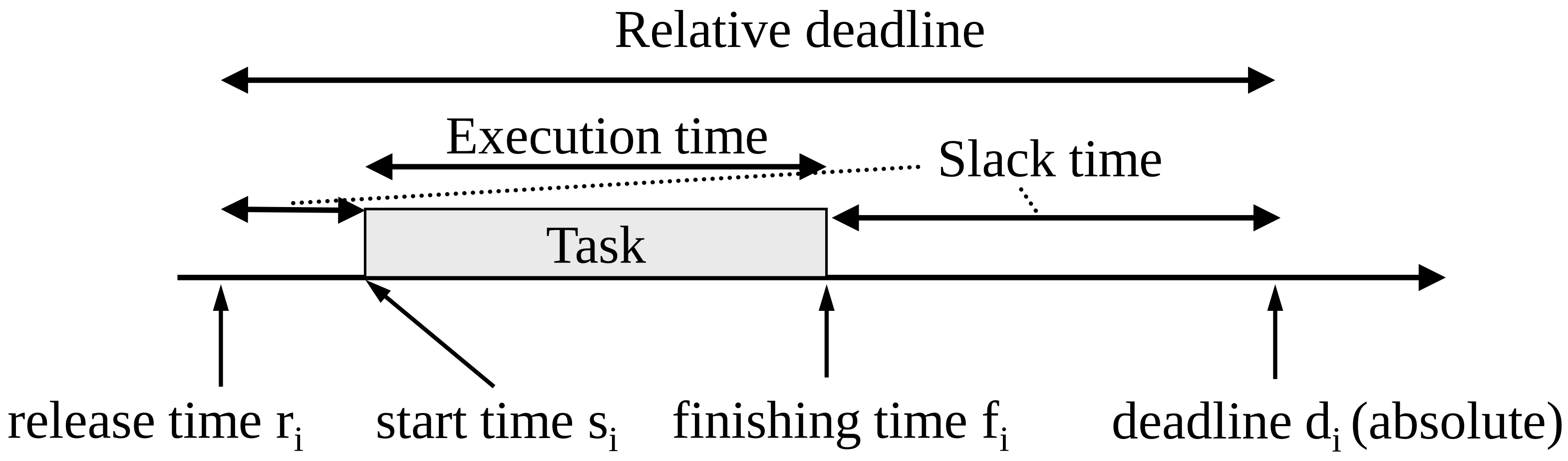


Task Execution Characteristics

- Execution time E_i
  - Execution time in absence of preemption (seconds or clock cycles)
  - Problem: Execution time varies (caching, pipelining,..)
  - Real-time systems: Only WCET matters!

- General schedulability analysis is NP-hard problem
  - For aperiodic tasks EDF (explained later) runs in polynomial time

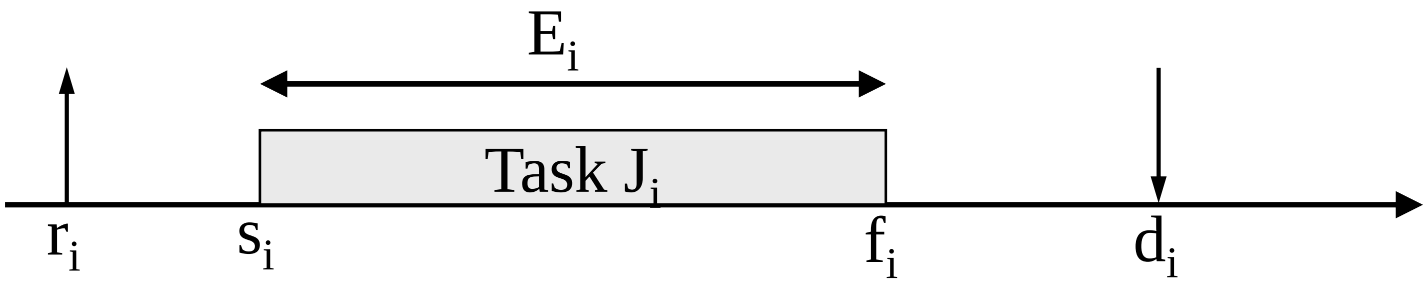
Basic Notions



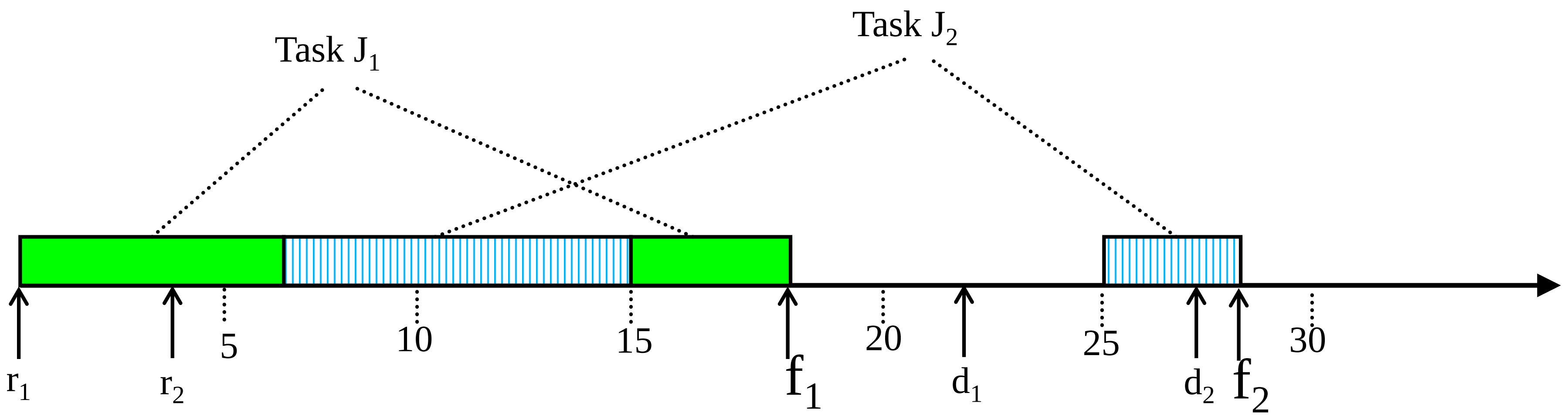
- **Deadline** d_i
  - time at which a task must be completed
- **Start time** s_i
  - time at which a task starts its execution
- **Finishing time** f_i
  - time at which a task finishes its execution
- **Slack time** $X_i = d_i - r_i - E_i$
  - Maximum time a task can be delayed on its activation to complete within its deadline

Basic Notions

- $d_i \geq r_i + E_i$
- **Lateness**: $L_i = f_i - d_i$
  - represents delay of a task completion with respect to its deadline
  - if task completes before deadline, lateness is negative



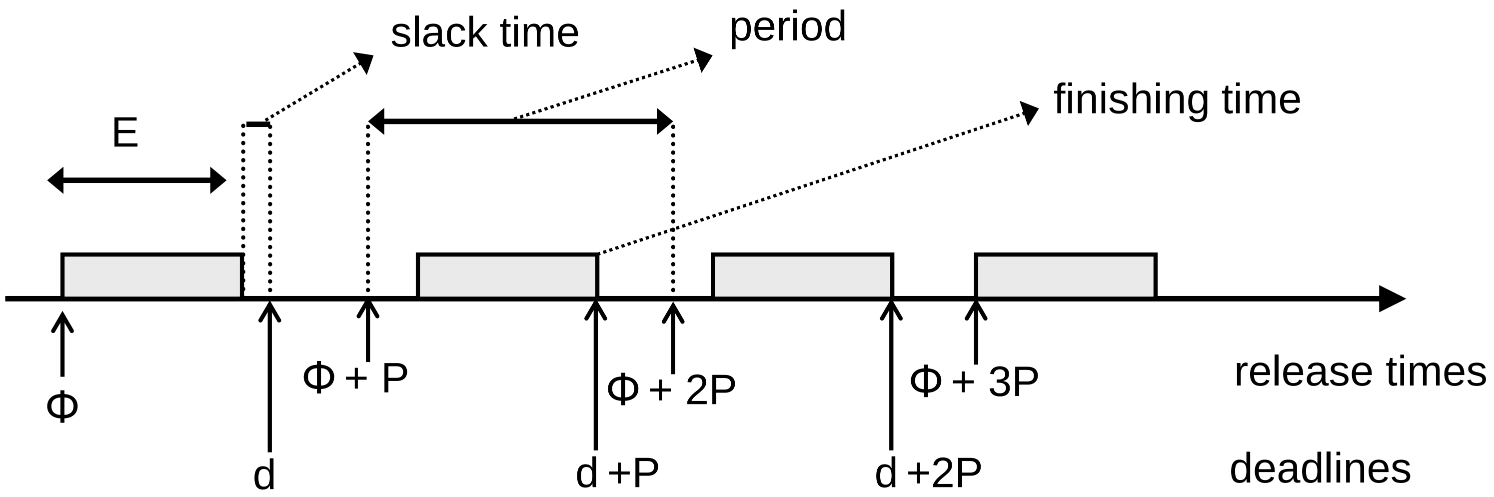
Example



- Execution times: $E_1 = 9$, $E_2 = 12$
- Release times: $r_1 = 0$, $r_2 = 4$
- Deadlines: $d_1 = 22$, $d_2 = 27$ ► Slack time: $X_1 = 13$, $X_2 = 11$
- Start times: $s_1 = 0$, $s_2 = 6$
- Finishing times: $f_1 = 18$, $f_2 = 28$ ► Lateness: $L_1 = -4$, $L_2 = 1$

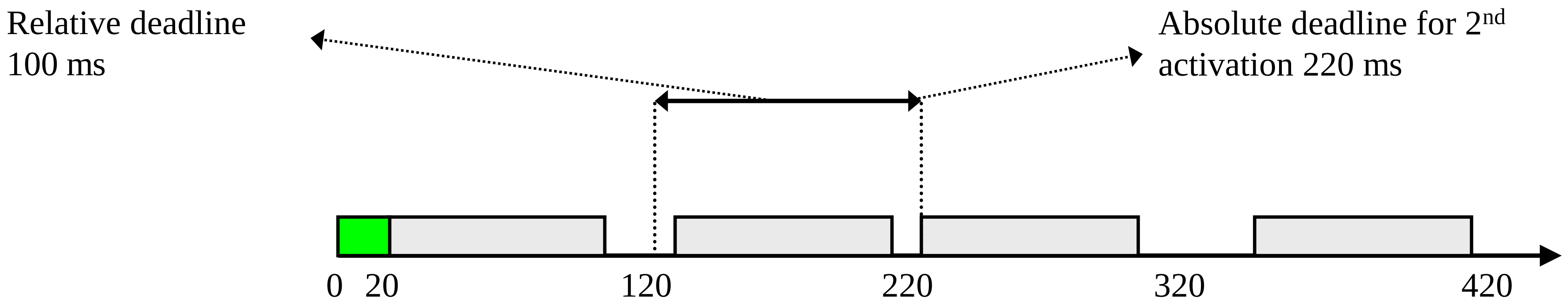
Periodic tasks

- Periodic task:
  - Sequence of identical activities (same execution time) that are regularly activated at a constant rate with period P
  - Release time of first instance is called **phase** Φ
    - Release time of kth activation is $\Phi + (k-1)P$
    - Deadline of kth activation is $d + (k-1)P$ (often $d = P$)



Example: Heating Furnace

- System takes 20 ms to initialize after turned on
- After initialization, every 100 ms, the system:
  - samples and reads temperature sensor
  - computes control-law for furnace to process temperature readings
  - determines correct flow rates of fuel, air and coolant
  - adjusts flow rates to match computed values
- Release time of kth activation is $20 + (k-1) \times 100$ ms



4

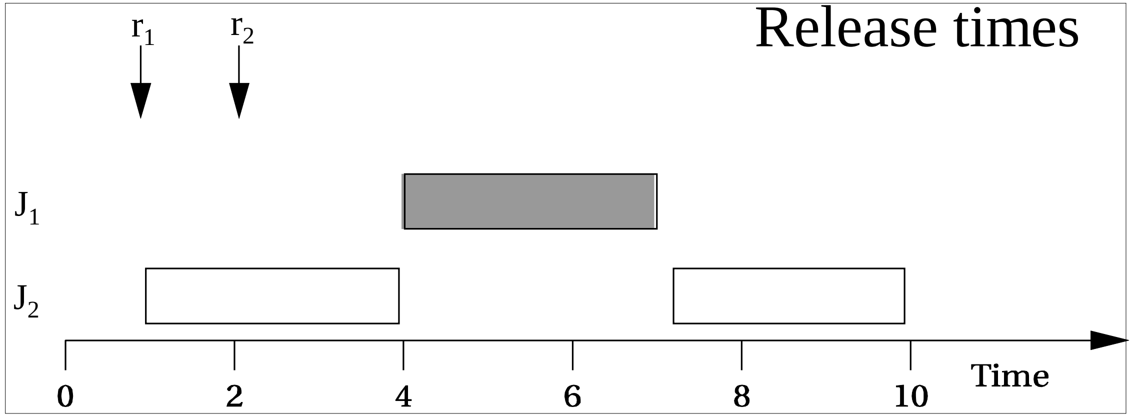
Scheduling

Scheduling Goals

- Produce feasible schedules
- Minimize given cost function
- Metrics
  - Average response time
  - Total completion time
  - For soft RT systems
    - Minimize maximum lateness & number of late tasks
- Scheduling process consumes CPU time
  - Thus, not all CPU time is available for tasks
  - Scheduling overhead must be taken into account for exact schedule
  - Assumption in the following:
    - All overheads caused by scheduler are assumed to be zero

Scheduling and Context Switch Overhead

Task	Execution time E_i	Period = Deadline d_i
J_1	3	10
J_2	3	5



- No feasible schedule exists if context switch overhead is 1
  - $E_1 + 2E_2 + 2 = 11$, hence earliest finishing time is 12
  - Second invocation of J_2 must be finished before 11

Scheduling Algorithms

- Static algorithms
  - Scheduling decisions are based on fixed parameters, assigned to tasks before their activation
- Dynamic algorithms
  - Scheduling decisions are based on dynamic parameters that may change during execution

Scheduling Based on Priorities

- **Principle of priority scheduling**
  - Upon each scheduling event (task finishes, task released) task with highest priority in task set is scheduled, if currently executing task has lower priority then it is interrupted
  - Ties are broken arbitrarily, e.g. by FIFO
- Task set = released but not completed tasks
- Task set is dynamically maintained, implemented as priority queue
- Priorities are determined
  - statically during creation or
  - dynamically during execution
- All scheduling algorithms are based on priorities

5

Specific Scheduling Algorithms

Specific Scheduling Algorithms

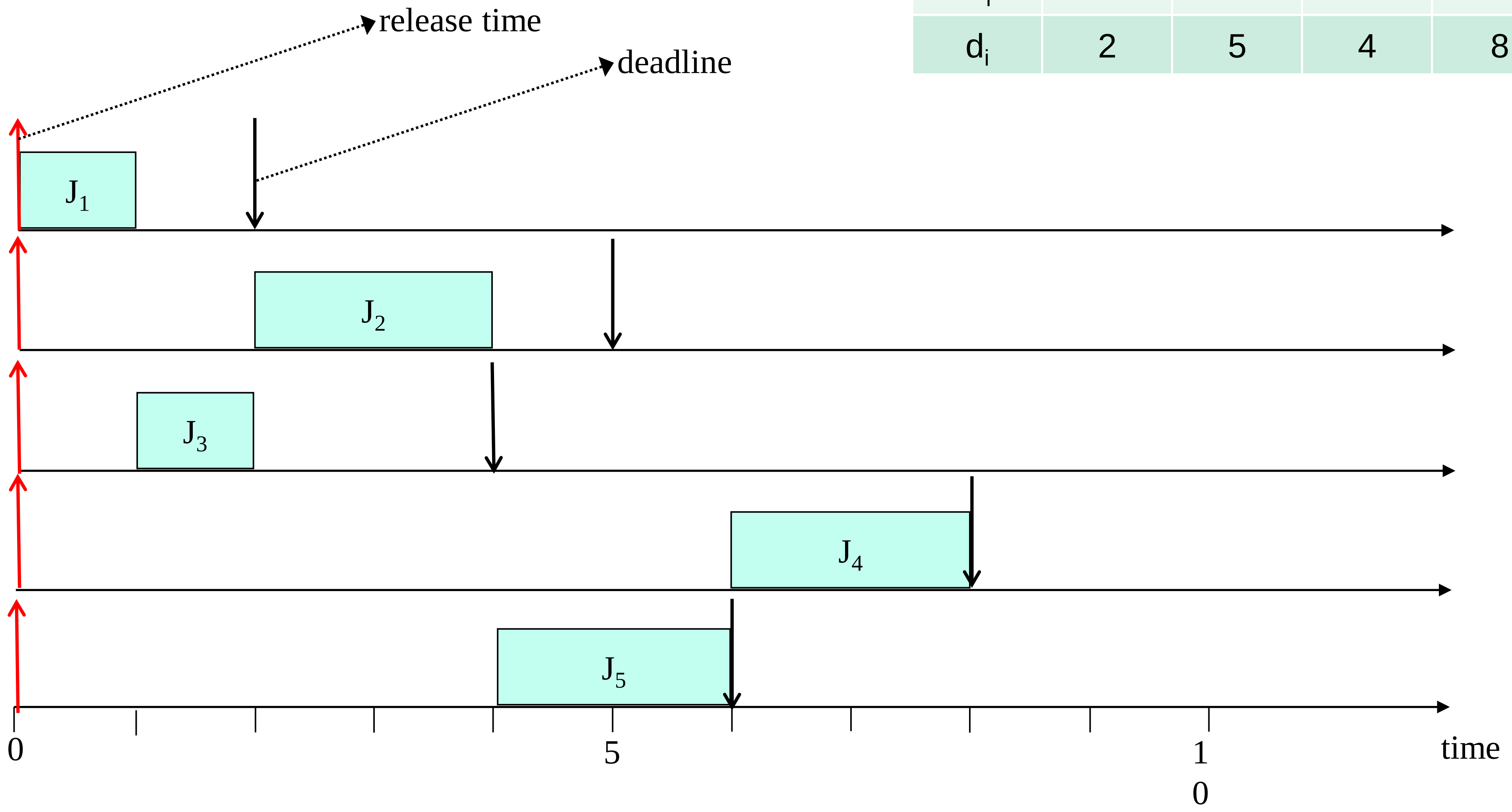
- Aperiodic tasks
  - Earliest Deadline First (EDF)
    - preemptive schedule
    - arbitrary arrival times
    - independent tasks (i.e. no precedence constraints)
  - Earliest Deadline First (EDF*)
    - With precedence constraints
- Periodic tasks
  - Rate Monotonic Scheduling (RM)
    - no precedence constraints
    - deadlines equal periods
  - Deadline Monotonic Scheduling (DM)
    - Same as RM but deadlines can be different from periods

Earliest Deadline First (EDF)

- Priorities are based on deadlines
  - Tasks with earlier deadlines have higher priorities
- Guarantees
  - If a set of tasks has a feasible schedule, then EDF will schedule these tasks, so they all complete by their deadline
  - Schedule is optimal with respect to minimizing maximum lateness
- EDF can also be used for periodic tasks

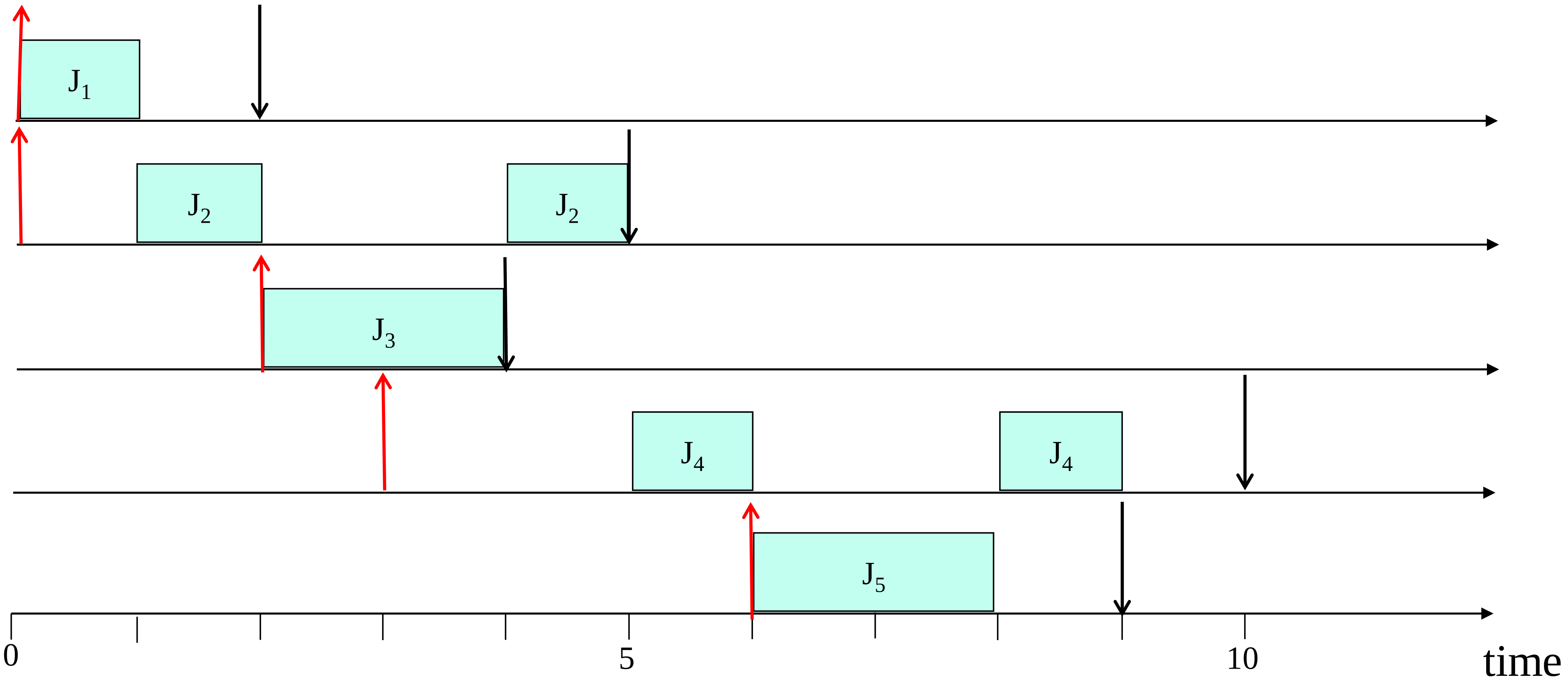
EDF: Example 1

	J_1	J_2	J_3	J_4	J_5
r_i	0	0	0	0	0
E_i	1	2	1	2	2
d_i	2	5	4	8	6



EDF: Example 2

	J_1	J_2	J_3	J_4	J_5
r_i	0	0	2	3	6
E_i	1	2	2	2	2
d_i	2	5	4	10	9



Concept of proof for guarantee of EDF

- For each time interval [t,t+1) it is verified, whether actual running task is the one with earliest absolute deadline
- If this is not the case, task with earliest absolute deadline is executed in this interval instead
- This operation cannot increase the maximum lateness

Schedulability test for EDF

- Assume there exist a feasible schedule for a set of n tasks
- Consider a new task arriving at time t (i.e. now n+1 tasks)
- Order all tasks by deadline J = $\{J_1, J_2, \ldots, J_{n+1}\}$
- Denote by $c_i(t)$ the remaining worst-case execution time of task J_i at time t for this schedule
-
```
f0 := t
for i := 1,…,n+1
   fi := fi-1 + ci(t)
   if (fi > di)
      return impossible to schedule
return possible to schedule
```

Pros and Cons of EDF

- Pros
  - Simple and works nicely in theory
  - Simple schedulability test
  - Optimal
  - Best CPU utilization
- Cons
  - Difficult to implement in practice. Not very often adopted due to dynamic priority-assignment (maintenance of ready queue is expensive)
  - Non stable: if any task instance fails to meet its deadline, the system is not predictable, any instance of any task may fail

EDF*: EDF With Precedence Constraints

- In some applications tasks have precedence constraints
  - A task must be completed before start of some other tasks
- Requirements
  - In a valid schedule a task starts execution
    - not earlier than its release time and
    - not earlier than the finishing times of its predecessors
  - All tasks finish their execution within their deadlines
- EDF* algorithm
  - Determines a feasible schedule for tasks with precedence constraints if there exists one

Principle of EDF*

1. Transform set J of dependent tasks into set J* of independent tasks with new release times r* and new deadlines d*
2. Apply EDF to set J*

- Transformation is done by modification of
  - Release times
    - Task must not start execution earlier than minimum finishing time of its predecessors and its own release time
  - Deadlines
    - Task must finish execution time within its deadline
    - Task must not finish execution later than maximum start time of its successor

EDF*

Algorithm for modification of release times:

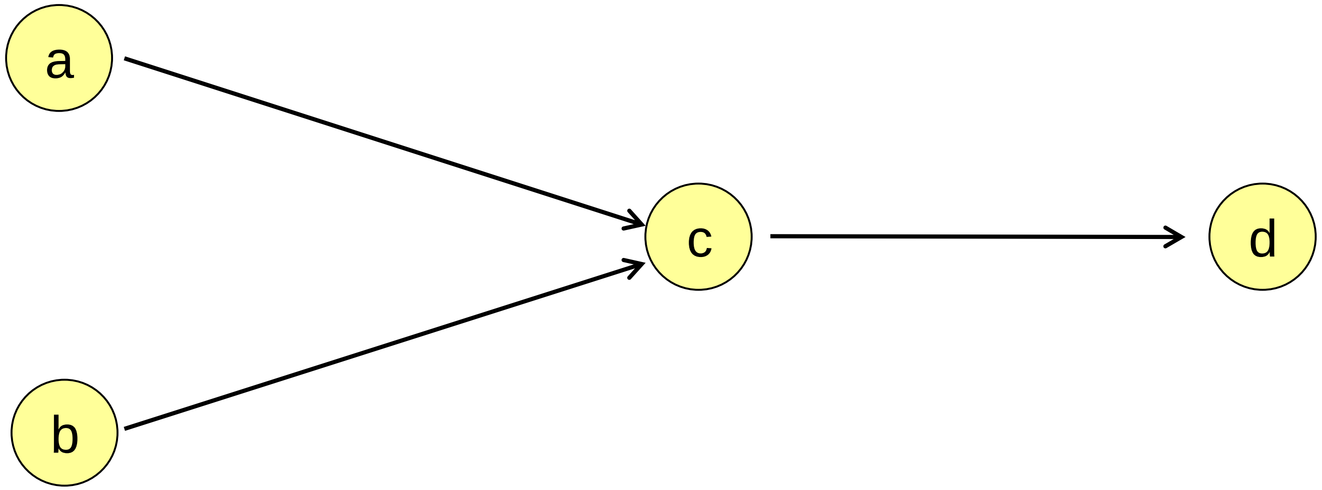
1. For any initial node of precedence graph set $r_i^* = r_i$
2. Select a task J_k such that its release time has not been modified, but release times of all immediate predecessors J_i have been modified
3. If no such task exists, halt
   else
   set $r_k^* = \max \{r_k, \max\{r_i^* + E_i : J_i \rightarrow J_k\}\}$
4. Return to second step

EDF*

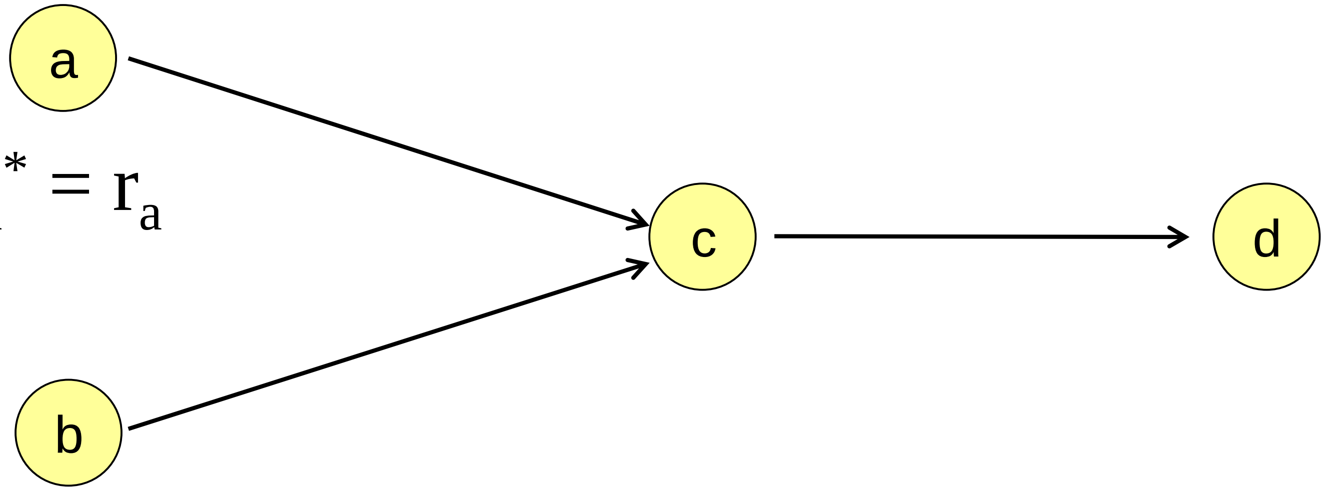
Algorithm for modification of deadlines:

1. For all terminal nodes of precedence graph set $d_i^* = d_i$
2. Select a task J_k such that its deadline has not been modified but deadline of all immediate successors J_i have been modified
3. If no such task exists, halt
   else
   set $d_k^* = \min\{d_k, \min\{d_i^* - E_i : J_k \rightarrow J_i\}\}$
4. Return to second step

Example: EDF*



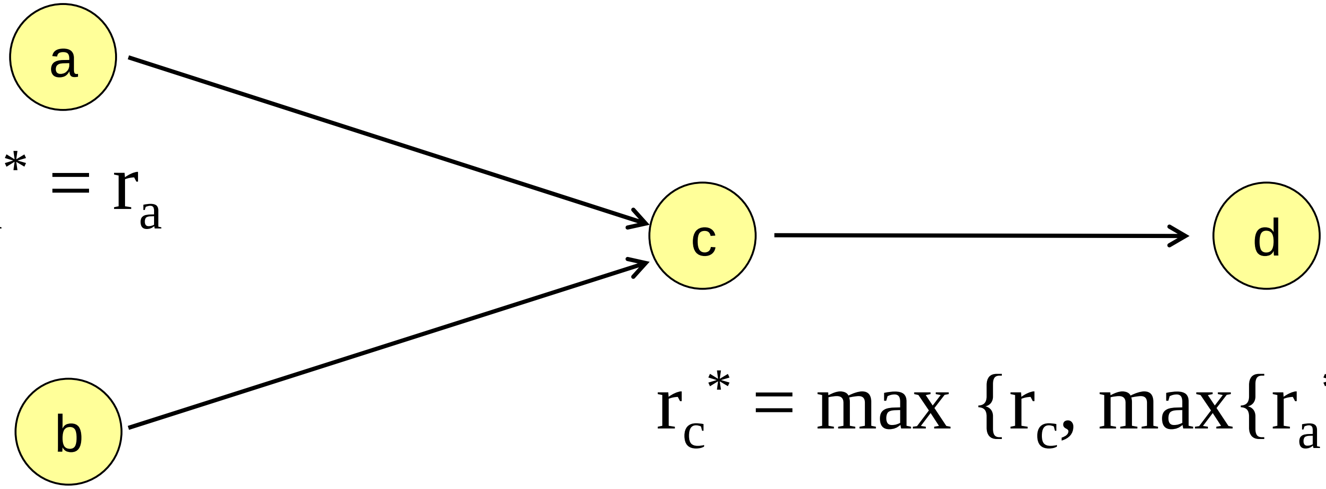
Example: EDF*



$r_a^* = r_a$

$r_b^* = r_b$

Example: EDF*



$$r_a^* = r_a$$

$$r_b^* = r_b$$

$$r_c^* = \max\{r_c, \max\{r_a^* + E_a, r_b^* + E_b\}\}$$

Example: EDF*

$$r_d^* = \max\{r_d, \max\{r_c^* + E_c\}\}$$

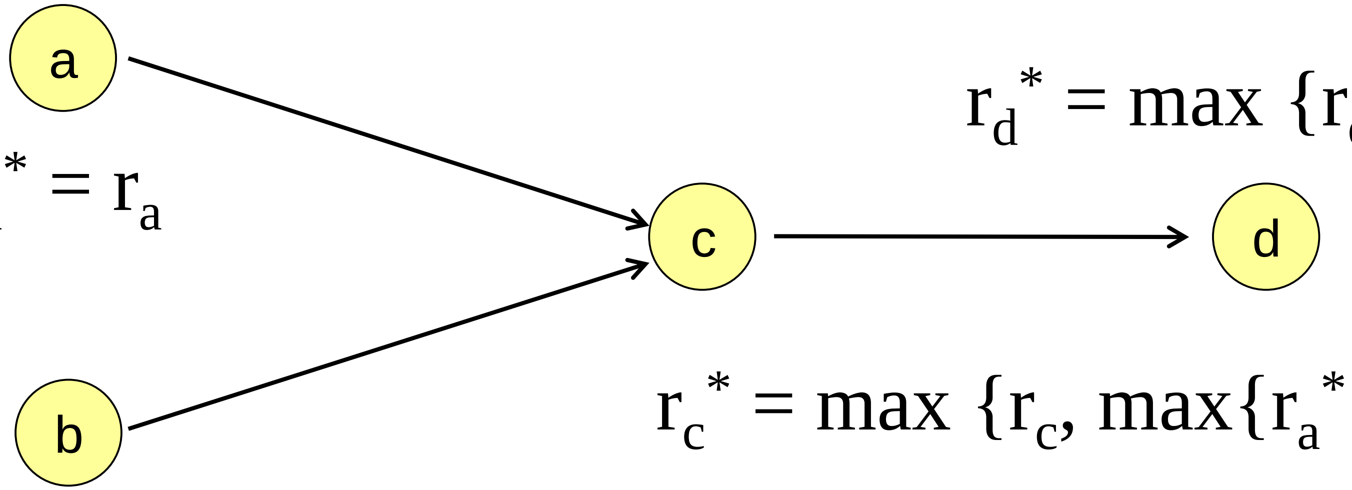
$$\begin{tikzcd}
a \arrow[dr] & & \\
 & c \arrow[r] & d \\
b \arrow[ur] & &
\end{tikzcd}$$

$$r_a^* = r_a$$

$$r_c^* = \max\{r_c, \max\{r_a^* + E_a, r_b^* + E_b\}\}$$

$$r_b^* = r_b$$

Example: EDF*



$$r_d^* = \max\{r_d, \max\{r_c^* + E_c\}\}$$

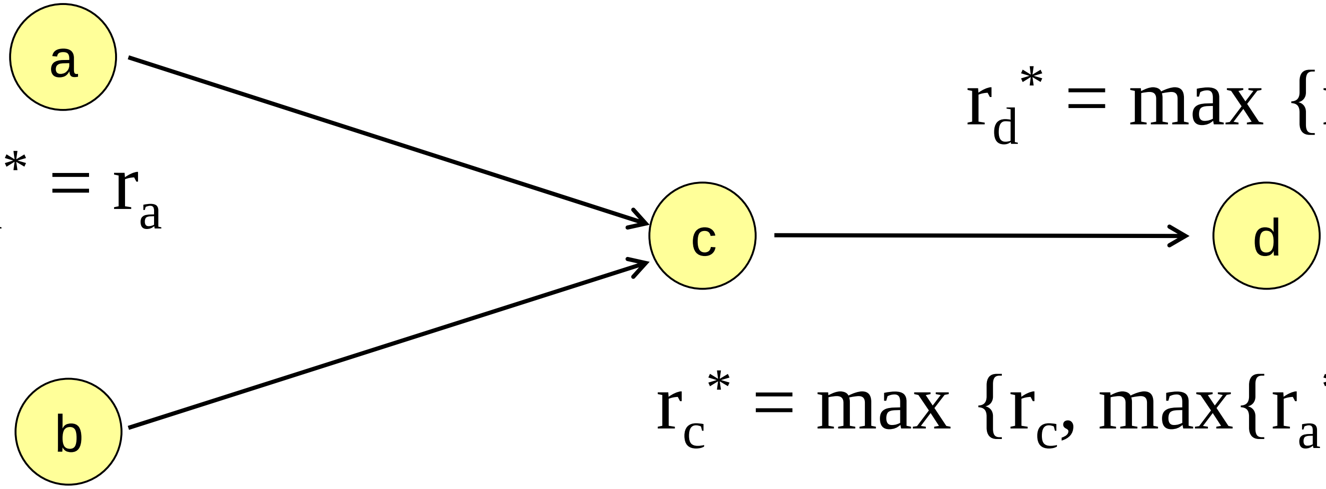
$$r_a^* = r_a$$

$$d_d^* = d_d$$

$$r_c^* = \max\{r_c, \max\{r_a^* + E_a, r_b^* + E_b\}\}$$

$$r_b^* = r_b$$

Example: EDF*



$$r_a^* = r_a$$

$$r_b^* = r_b$$

$$r_d^* = \max\{r_d, \max\{r_c^* + E_c\}\}$$

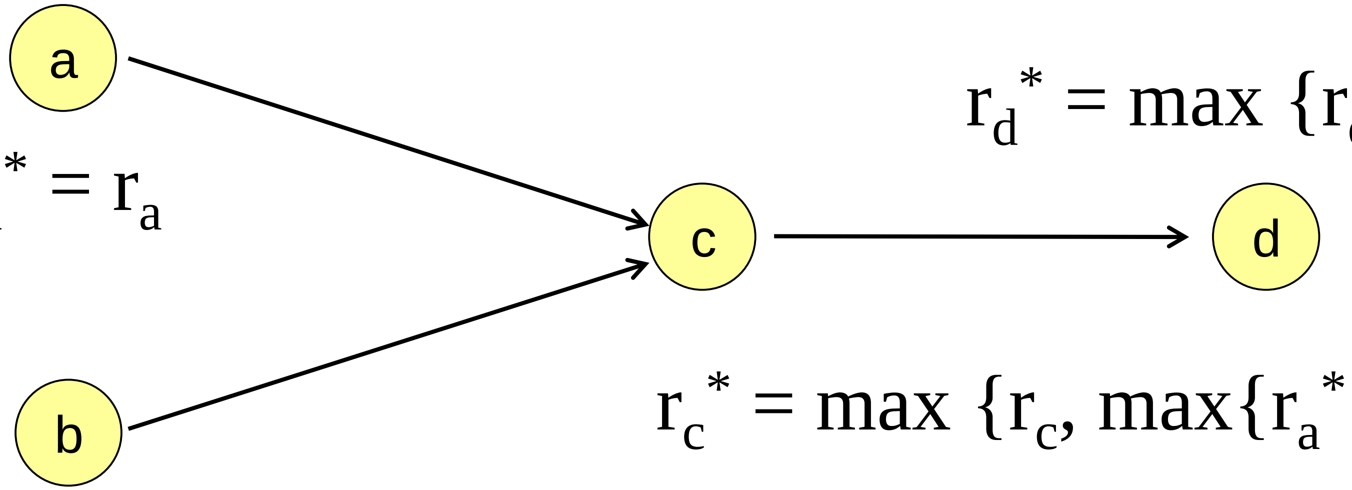
$$d_d^* = d_d$$

$$r_c^* = \max\{r_c, \max\{r_a^* + E_a, r_b^* + E_b\}\}$$

$$d_c^* = \min\{d_c, \min\{d_d^* - E_d\}\}$$

Example: EDF*

$d_a^* = \min\{d_a, d_c^* - E_c\}$



$r_a^* = r_a$

$r_d^* = \max\{r_d, \max\{r_c^* + E_c\}\}$

$d_d^* = d_d$

$r_c^* = \max\{r_c, \max\{r_a^* + E_a, r_b^* + E_b\}\}$

$d_c^* = \min\{d_c, d_d^* - E_d\}$

$r_b^* = r_b$

$d_b^* = \min\{d_b, d_c^* - E_c\}$

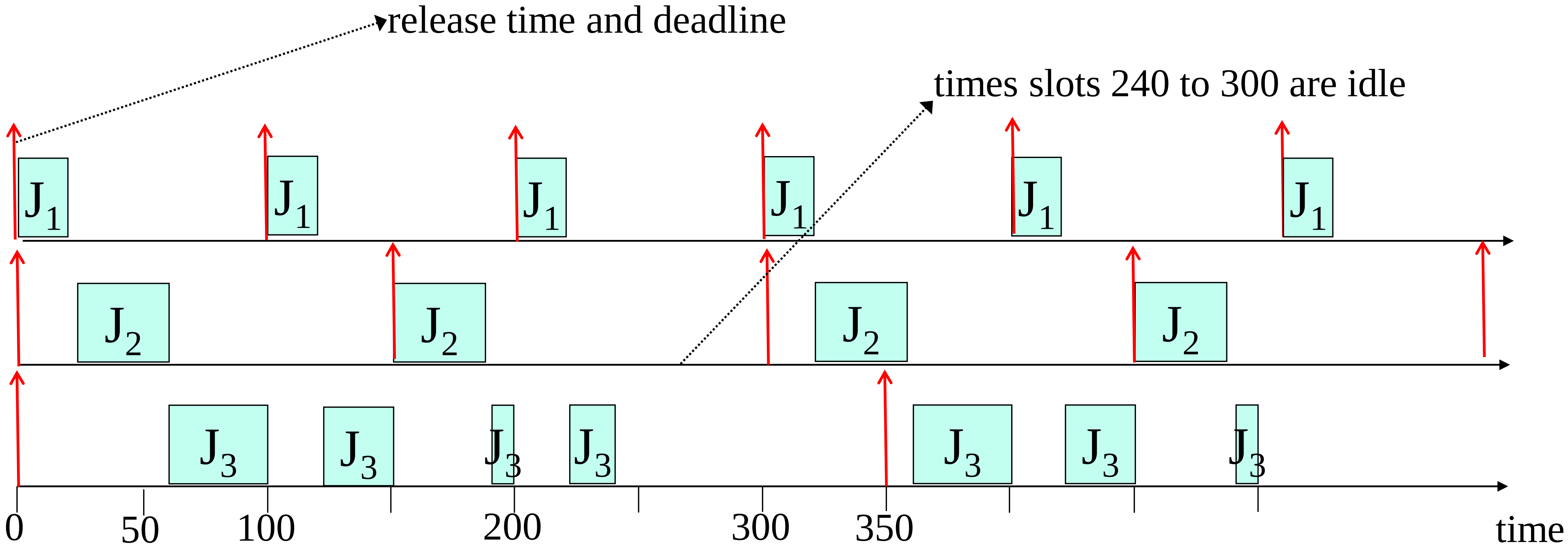
Rate Monotonic Scheduling (RM)

- For independent periodic tasks only
- Preemptive schedule
- Definition of priority:
  Tasks with shorter periods have higher priorities
  - priority = 1/period (this is called the rate)
- Priorities are static (they never change)
- Assumptions
  - deadline = period
  - tasks are always released at start of period

RM: Example 1

	J_1	J_2	J_3
P_i	100	150	350
E_i	20	40	100

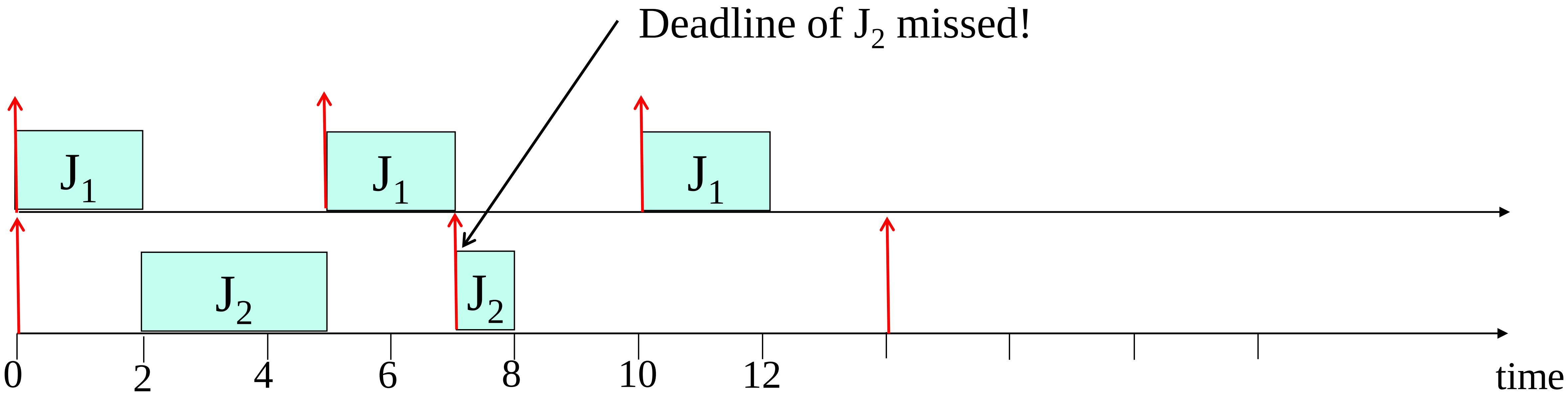
RM does produce feasible schedule!



RM: Example 2

	J_1	J_2
P_i	5	7
E_i	2	4

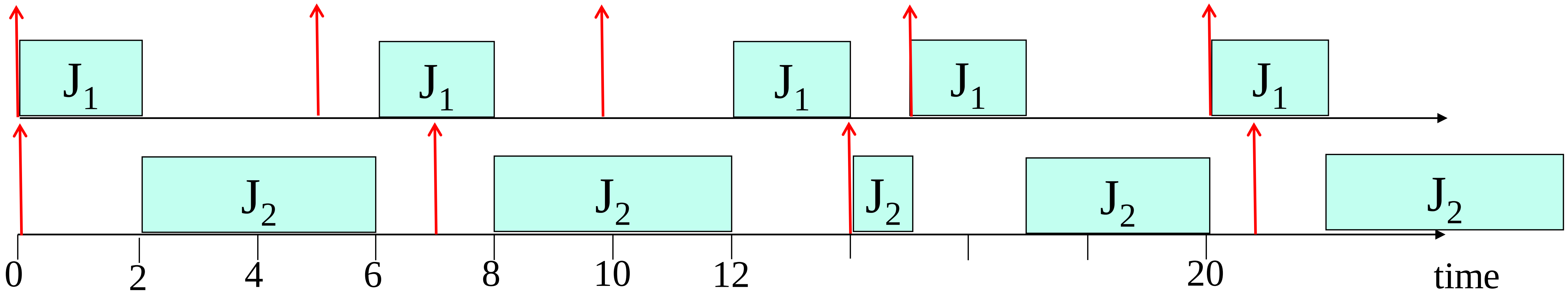
RM does not always produce feasible schedule!



Example 2 with EDF

	J_1	J_2
P_i	5	7
E_i	2	4

EDF does produce a feasible schedule for this task set!



CPU Utilization

- CPU utilization U of a task is E/P
  - ratio of period P and execution time E
- CPU utilization a set of tasks is $U = \sum_i \frac{E_i}{P_i}$
  - $U = 0.752$ for first example
  - $U = 0.971$ for second example
- CPU utilization is a measure on how busy processor could be during shortest repeating cycle
  - $U > 1$ (overload)
    - some task will fail to meet deadline no matter what algorithm is used!
  - $U \leq 1$
    - doesn’t imply schedulable, it depends on scheduling algorithm
    - If U = 1 and CPU is kept busy, all deadlines will be met
- **Theorem**
  EDF produces a feasible schedule if and only if $U \leq 1$

Schedulability test for RM

- U < 1 does not imply schedulability for RM
  - see previous example
- The schedulability test for RMS is: $U \leq n(2^{1/n}-1)$
  - For $n = 2$ this requires $U \leq 0.828$
  - For $n = 3$ this requires $U \leq 0.7797$
  - As number of tasks approaches grows, U converges to 0.69
    - i.e. if U ≤ 69% then RMS guarantees to meet all deadlines
- RM is optimal in the following sense:
  - If a task set is schedulable with any fixed-priority scheduling algorithm, it is also schedulable with RM
  - EDF has no fixed priorities, priority = 1/(time until deadline)

RM: Example 3

$U = 1/3 + 1/5 + 1/6 + 2/10 = 0.899$

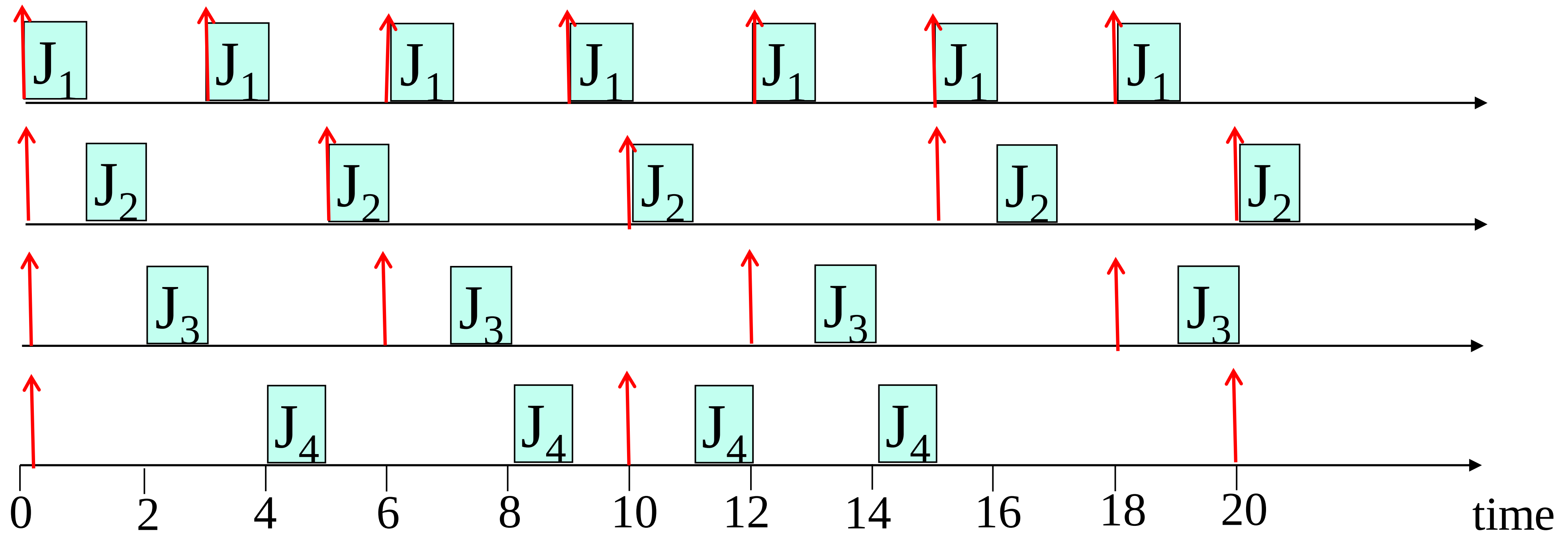
Schedulability bound: 0.756

RM produces a valid schedule

Hence, schedulabilty test is only sufficient, not necessary!

Precise/exact schedulability test exists

	J_1	J_2	J_3	J_4
P_i	3	5	6	10
E_i	1	1	1	2



Pros and Cons of RM

- Pros
  - Simple to understand
  - Easy to implement, static priority assignment
  - Stable: though some of the lower priority tasks fail to meet deadlines, others may meet deadlines
- Cons
  - lower CPU utilization than EDF
  - Requires D=P
  - Only deals with independent tasks

- The cons except lower CPU utilization can be fixed
- RM is no longer optimal if deadlines are shorter than period

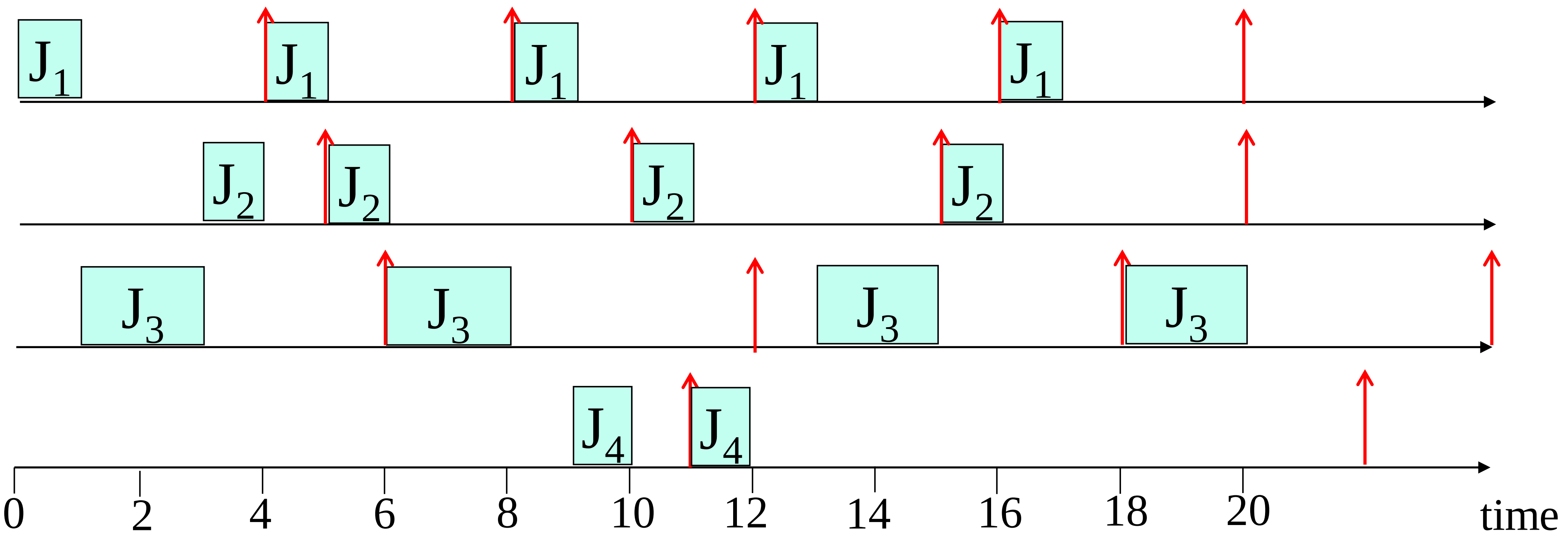
Deadline Monotonic Scheduling (DM)

- Task model: same as for RM but deadlines can be shorter than period
- Definition of priority:
  Tasks with shorter deadlines have higher priorities
- If deadlines are equal to periods then DM = RM

DM: Example 1

Time slot 17 is idle

	J_1	J_2	J_3	J_4
P_i	4	5	6	11
E_i	1	1	2	1
d_i	3	5	4	10



Deadline Monotonic Scheduling (DM)

- Principle
  - Scheduler keeps a list of released tasks, sorted by priorities
  - DM schedules first task on list
  - If a new task has higher priority than current task, then current task is preempted
- Schedulability test for DM
  - If $U = \sum_i \frac{E_i}{D_i} \leq n\,(2^{\frac{1}{n}} - 1)$ then task set schedule by DM
- Test is only sufficient
- $U = 0.874$ for previous example
- Bound for $n = 4$ is 0.756
- Is it schedulable?

6

Real-time Operating Systems

Real-time Operating Systems (RTOS)

- RTOS
  Operating system designed to meet strict deadlines
- Methods employed
  - Drop or reduce certain tasks when they cannot be executed within time constraints (load shedding)
  - Monitor input consistently and in a timely manner
  - Keep track of how much of each resource (CPU time, RAM, communications bandwidth, etc.) might possibly be used in worst-case by currently-running task, and refuse to accept a new task unless it fits in remaining unallocated resources

Simple Variant of RTOS

- Periodic execution
  - Provides a minimal time service: Scheduled clock pulse with fixed period
  - No preemption
  - Allows implementation of a static cyclic schedule, provided:
    - All tasks can be scheduled in a frame-based manner according to their WCET
    - All interactions with hardware to be done on a polled basis, i.e. no blocking

Simple Variant of RTOS

- Common approach:

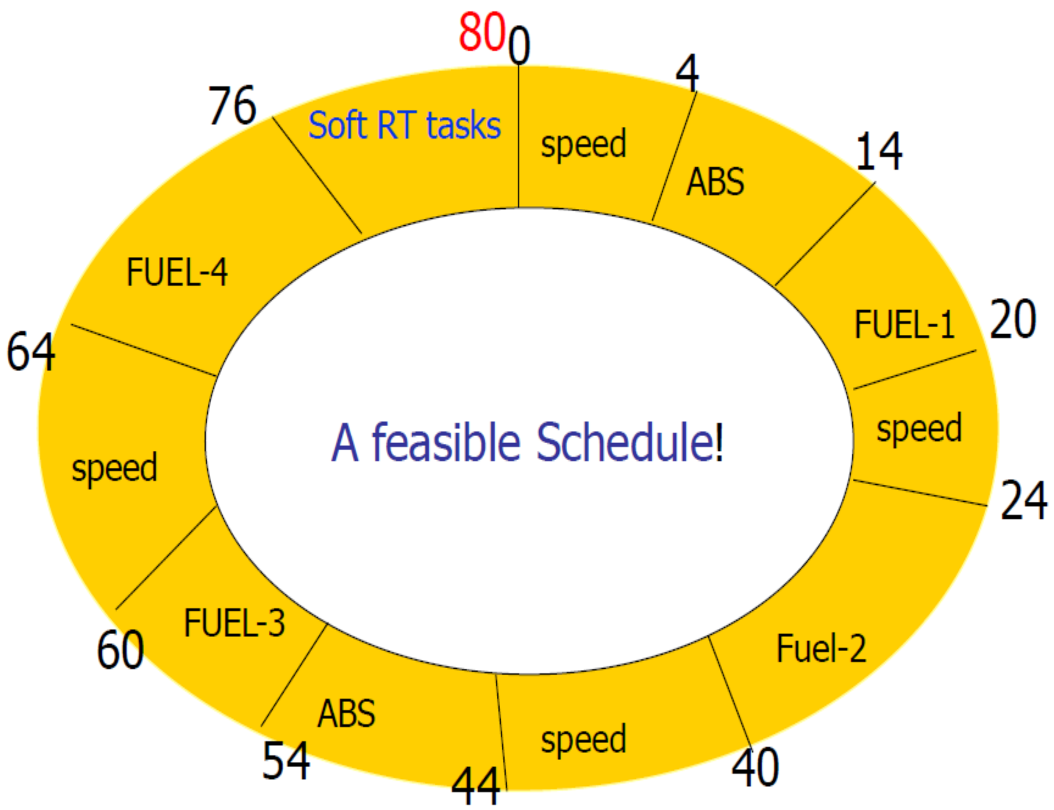
```
setup timer
c = 0;
while (1) {
   suspend until timer expires
   c++;
   do tasks due every cycle
   if ((c % 2) == 0)
      do tasks due every 2nd cycle
   if ((c % 3) == 0)
      do tasks due every 3rd cycle
   ...
}
```

Advanced Task scheduling

- Many real-time embedded systems are more complex, they need a sophisticated OS with priority scheduling
- Compiler generates execution plan (schedule) based on information about WCET
  - Non-preemptive schedules
    - Only determines order of tasks, tasks always finish execution
  - Preemptive schedules
    - Contains information about order and how long a task executes before preemption to allow next task to execute

Example 2: Car Control

- Activities of a car control system
  - Speed measurement: E = 4 ms, P = 20ms
  - ABS control: E = 10 ms, P = 40 ms
  - Fuel injection: E = 40 ms, P = 80ms
  - Other soft deadlines
    - air condition etc.
  - Schedule produced by EDF



Existing RTOS: Windows CE

- Built specifically for embedded systems and appliance market
- Scalable real-time 32-bit platform, supports Windows API
- Preemptive priority scheduling with 256 priority levels per process/task
- Kernel is 400 Kbytes
- Development tool: Visual Studio
- Supports Intel x86 and compatibles, MIPS, and ARM processors

Existing RTOS: QNX

- Real-time microkernel surrounded by resource managers
  - Provides POSIX/UNIX compatibility
  - Microkernels typically support only most basic services
  - Optional resource managers allow scalability from small ROM-based systems to huge multiprocessor systems
- Preemptive task scheduling using FIFO, round-robin, adaptive, or priorities
- Microkernel < 10 Kbytes and complies with POSIX real-time standard
- Supports PowerPC, x86 family, MIPS, ARM, StrongARM and XScale CPUs

- Alternatives: RTLinux, LynxOS, VxWorks, eCos,…

Fotolia

SES

Chapter 10: Real-Time Embedded Systems

Prof. Dr.-Ing. Bernd-Christian Renner

Institute smartPORT
Hamburg University of Technology

TUHH