Real-Time Scheduling

slides: P. Puschner

Scheduling

Task Model
Assumptions about task timing, interaction

Scheduling Algorithm
Scheduling mode and selection function

Schedulability Test
Prediction of worst-case behavior

Real-Time Scheduling Requirements

- Precedence constraints
- Mutual exclusion
- Rate requirements
- Deadline and response-time requirements

Classification of Scheduling Algorithms

- Guaranteed versus best-effort
- Static versus dynamic
- Preemptive versus non-preemptive
- Single-processor versus multi-processor
- Central versus distributed

Terminology

Periodic task

- hard deadline
- executed repeatedly at (semi)regular time intervals
- Parameters: T_i … period (min.), D_i … deadline, C_i … WCET

Aperiodic task

- soft or no deadline
- goal: optimize responsiveness

Sporadic task

- hard deadline
- Executed sporadically
- Parameters: $mint_i$ … minimum inter-arrival time, D_i, C_i

Clairvoyance

A scheduler is clairvoyant if it knows everything about the future.

A scheduler is optimal if it can find a schedule whenever the best clairvoyant scheduler can find a schedule.

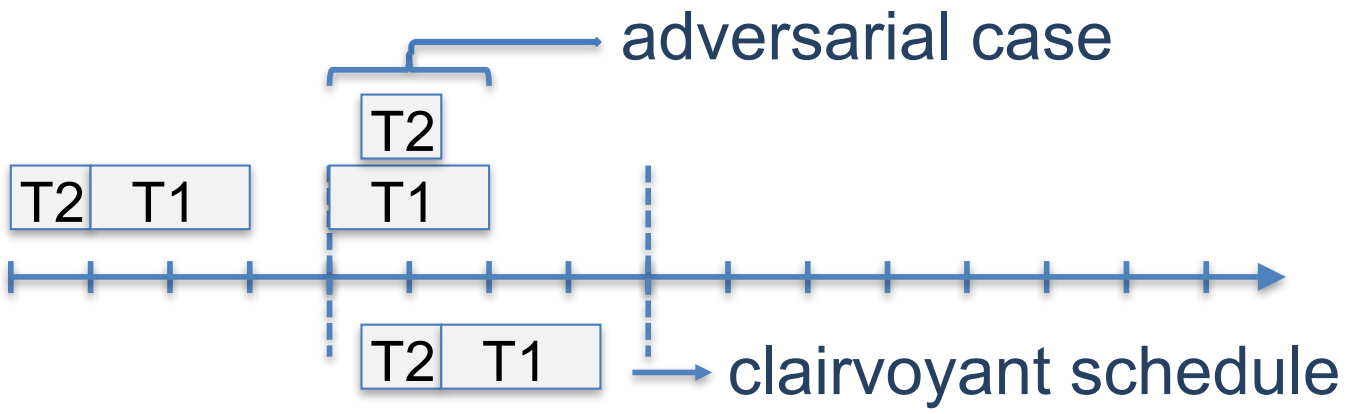
In the general case, a dynamic scheduler cannot be optimal – proof: adversary argument.

Under restricting assumptions, optimal dynamic schedulers exist.

Adversary Argument

Task set: T_1, T_2, mutually exclusive

Task	Period / Deadl.	WCET	Type
T1	4 / 4	2	periodic
T2	4 / 1	1	sporadic



Although there is a solution, an online scheduler cannot find it.

A Simple Model for Application Tasks

- Application consists of a fixed set of *n* tasks
- All tasks are periodic, periods are known
- Task deadlines are equal to task periods
- Worst-case execution times of all tasks are known
- Tasks are independent
- Overheads, context-switch times are ignored

Cyclic Executives

- Concurrent or pseudo-concurrent applications are mapped to collections of procedures/procedure calls
- Procedure calls are grouped into calls for each minor cycle
- All minor cycles together form the major cycle of the schedule
- The minor cycle determines the minimum cycle time of a task
- The major cycle determines the maximum cycle time of a task
- Statically planned
- Fully deterministic behavior and timing

Cyclic Executive – Example

Task	Period	Exec. Time
a	10	2
b	10	3
c	20	4
d	20	2
e	40	2
f	40	1

minor cycle: 10 time units

major cycle: 40 time units

```
while (1) {
        wait_for_timer_interrupt();
        task_a(); task_b(); task_c(); task_f();
        wait_for_timer_interrupt();
        task_a(); task_b(); task_d(); task_e();
        wait_for_timer_interrupt();
        task_a(); task_b(); task_c();
        wait_for_timer_interrupt();
        task_a(); task_b(); task_d();
}
```

Cyclic Executives – Properties

- Procedure calls instead of tasks at runtime
- Procedures share common address space
  - Can share common data structures
  - Mutual exclusion is guaranteed by construction
- All task periods must be a multiple of the minor cycle
- Long periods are difficult to accommodate – major cycle
- Inflexible – no sporadic or aperiodic tasks
- Large (long) tasks need to be split
  - Invalidates designed task structure
  - May invalidate mutex assumption between tasks

Fixed-Priority Scheduling (FPS)

- Each task has a static priority
- Task priorities are computed before runtime
- Priorities of ready tasks determine the execution order of tasks
- Priorities are derived from temporal requirements

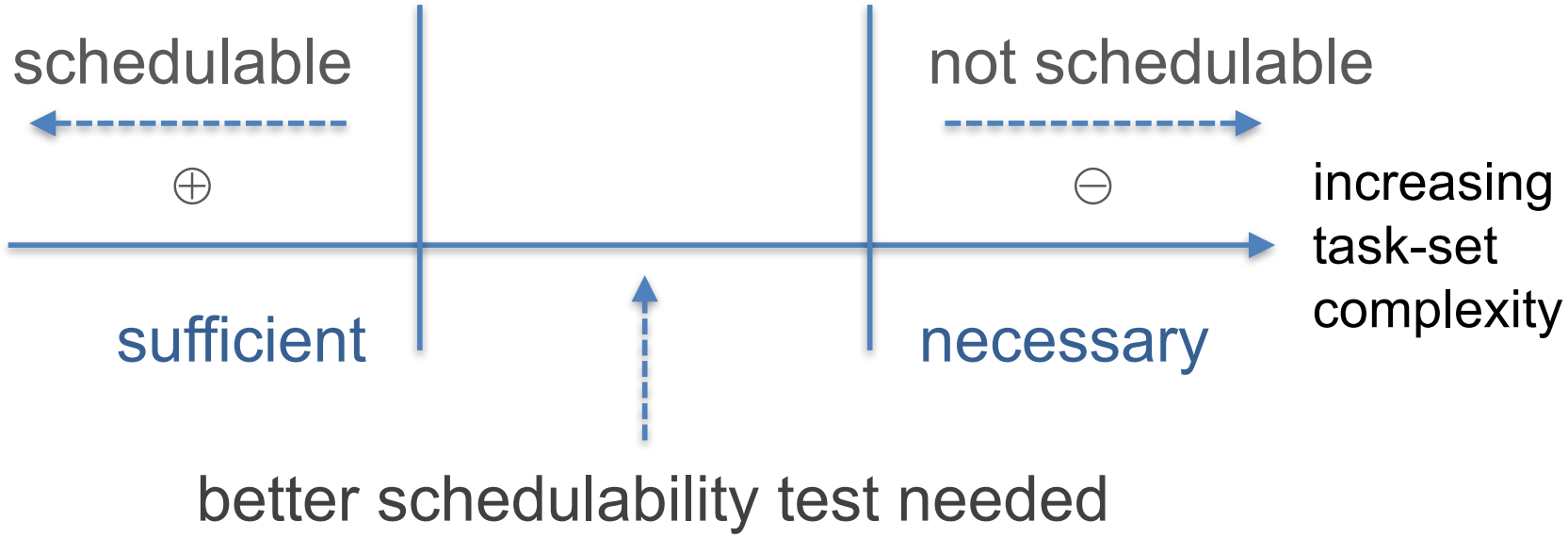
Rate-Monotonic Scheduling (RMS)

- Fixed priority scheduling, preemptive
- Rate-monotonic priority assignment
  - The shorter the period (= the higher the rate) of a task, the higher its priority P_i
  - For all $Task_i$, $Task_j$: $T_i < T_j \Leftrightarrow P_i > P_j$
- Selection function: Among the ready tasks the task with highest priority is selected to execute next.
- The rate-monotonic priority assignment is optimal for FPS
  - If a task set can be scheduled with a preemptive fixed-priority scheduler then the task set can also be scheduled with RMS

Schedulability Tests

If a sufficient schedulability test is positive, the tasks are definitely schedulable

If a necessary schedulability test is negative, the tasks are definitely not schedulable



Utilization-Based Schedulability Test

Utilization $U := \sum C_i / T_i$

Necessary schedulability test for RMS

$$U \leq 1$$

Sufficient schedulability test for RMS

$$U \leq n(2^{1/n} - 1)$$

Utilization-Based Schedulability Test

Theorem of Liu and Layland: A system of n independent, preemptable periodic tasks with $D_i = T_i$ can be feasibly scheduled on a processor according to the RM algorithm if its total utilization U (U_{RM}) is at most

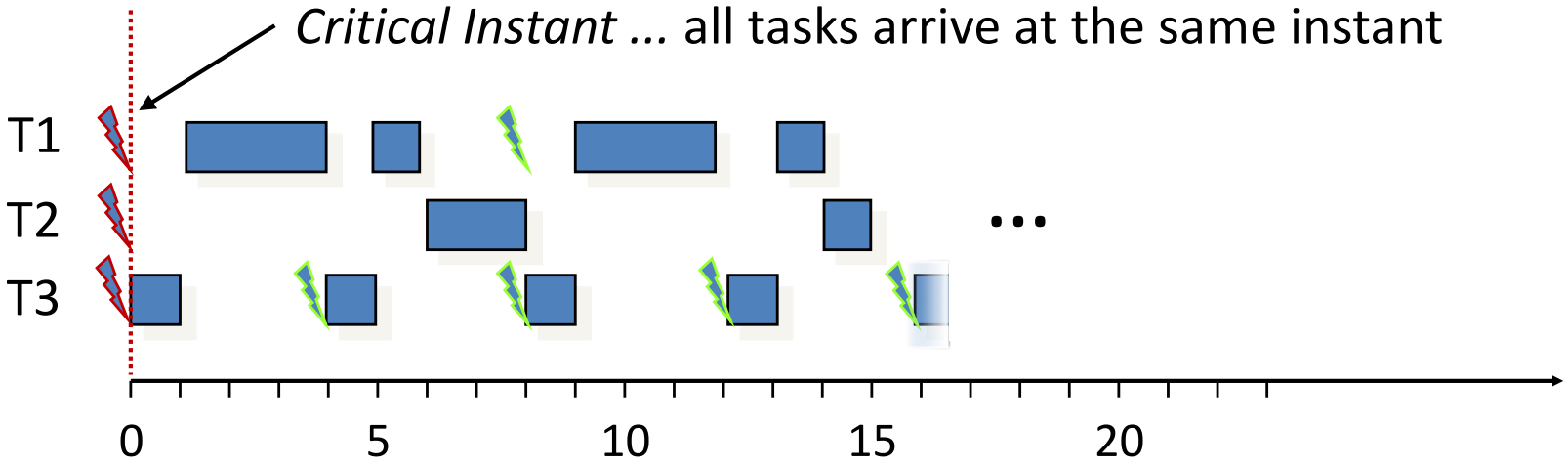
$$U_{RM}(n) = n(2^{1/n} - 1)$$

Examples: $U_{RM}(2) = 0.83$, $U_{RM}(3) = 0.78$, $U_{RM}(5) = 0.74$, $U_{RM}(10) = 0.72$

For big n: $U_{RM}(n) \approx \ln 2 \; (\approx 0.69)$

RMS Scheduling – Example

Task	C_i	T_i
T1	4	8
T2	3	16
T3	1	4



Earliest Deadline First Scheduling (EDF)

- Absolute deadlines determine the execution order of the tasks
- Selection function: the task with the earliest absolute deadline is selected to execute next
- Utilization-based schedulability test for EDF – necessary and sufficient condition:

$$\sum C_i / T_i \leq 1$$

- In general EDF can provide higher utilization than RMS

FPS versus EDF

- Implementation of static priorities (FPS) is easier
- EDF: ready queue sorted by deadlines; tasks that become ready need to be inserted at the right place
- FPS: tasks without deadlines can be added more easily, e.g., by assigning a low priority to these tasks;
  in EDF: assignment of “artificial” deadlines
- Overload
  FPS: Low-priority tasks miss their deadlines
  EDF: unpredictable; potential of domino effect

Response-Time Analysis for FPS

- Utilization-based tests are
  - Simple
  - Not exact
  - Not applicable to more general task models

⇨ Response-time analysis

- Compute worst-case response time, R_i, for each task thereby considering interference I_i from tasks of higher priority

$$R_i = C_i + I_i$$

- Check whether the task meets its deadline, i.e., $R_i \leq D_i$

Response-Time Analysis for FPS (2)

- To bound interference I_i, we need to know how often each task $Task_j$ of higher priority preempts $Task_i$
- Assuming that all tasks start at the same time, e.g., time 0, the maximum number of task preemptions of $Task_i$ by $Task_j$ is:

$$\left\lceil \frac{R_i}{T_j} \right\rceil$$

- For each preemption, the maximum interference is C_j. Therefore the interference of $Task_j$ is:

$$\left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Response-Time Analysis for FPS (3)

- Let hp_i be the set of tasks with priority higher than $Task_i$
- R_i can be calculated by considering the interference of hp_i

$$R_i = C_i + \sum_{j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- The formula can be solved by solving the following set of recurrence relations:

$$w_i^{n+1} = C_i + \sum_{j \in hp_i} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

starting with: $w_i^0 = C_i$

Response-Time Analysis for FPS (4)

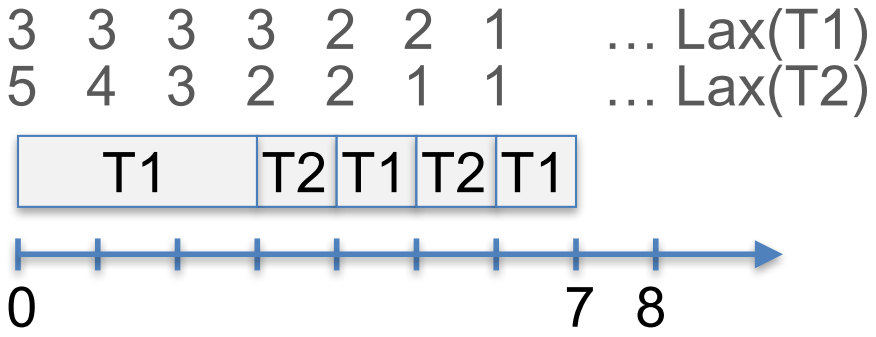
The response-time analysis is a necessary and sufficient schedulability test

- If a set of tasks passes the test then all tasks will meet their deadlines
- If the task set fails the test, then a deadline miss will occur at runtime (unless WCET estimates are pessimistic)

Least-Laxity First Scheduling (LLF)

- Laxity: Difference between deadline and remaining computation time
- Selection function: The task with the smallest laxity gets the highest (dynamic) priority and is therefore selected for executing next
- In uniprocessor systems LLF scheduling is optimal

Task	Deadline	WCET
T1	8	5
T2	7	2

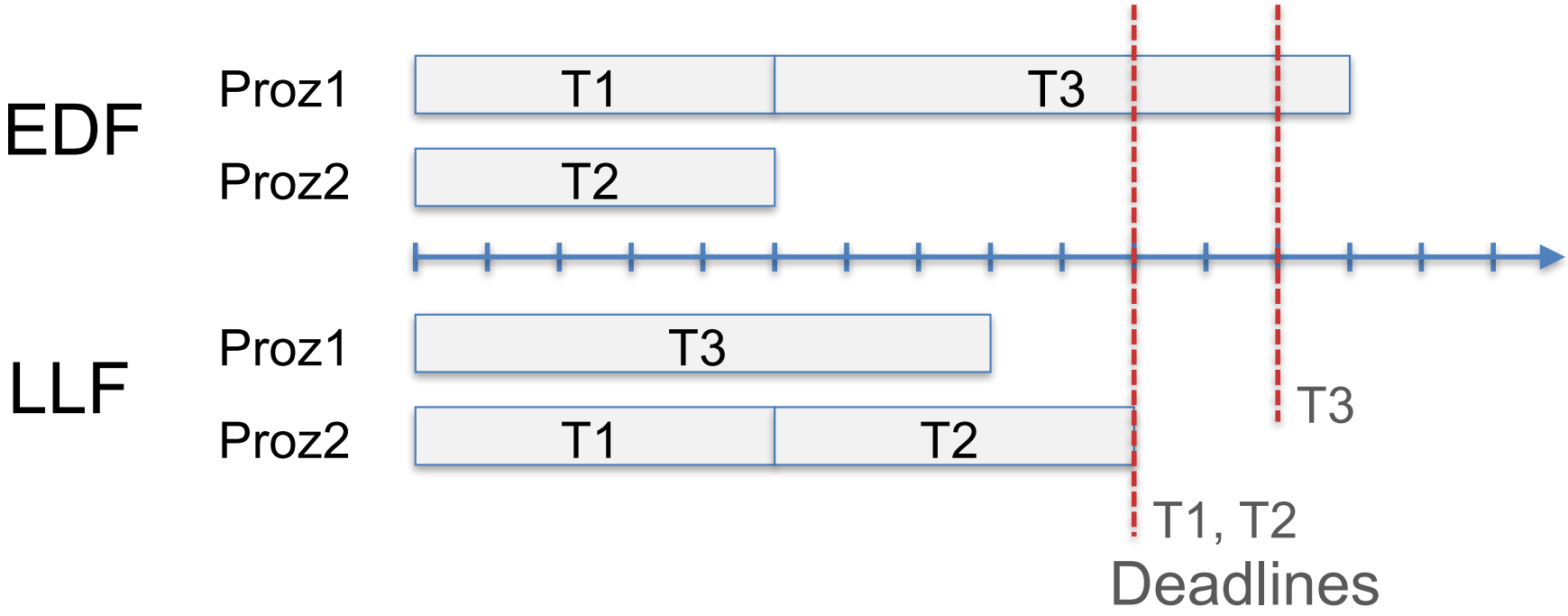


- Modified LLF (MLLF) reduces number of task switches

Multiprocessor Scheduling

Task set

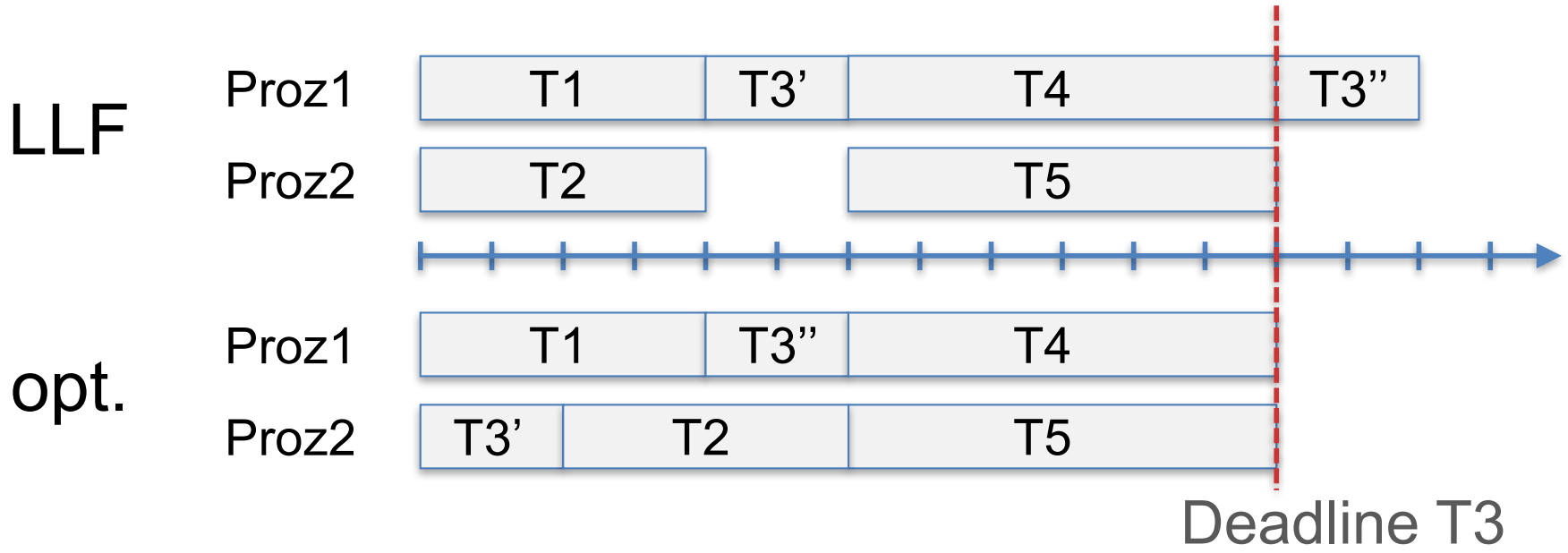
Task	Deadline	WCET
T1	10	5
T2	10	5
T3	12	8



Non-Optimality of LLF in Multiprozessor Sys.

Task set

Task	Arrival	Deadline	WCET
T1	0	4	4
T2	0	8	4
T3	0	12	4
T4	6	12	6
T5	6	12	6



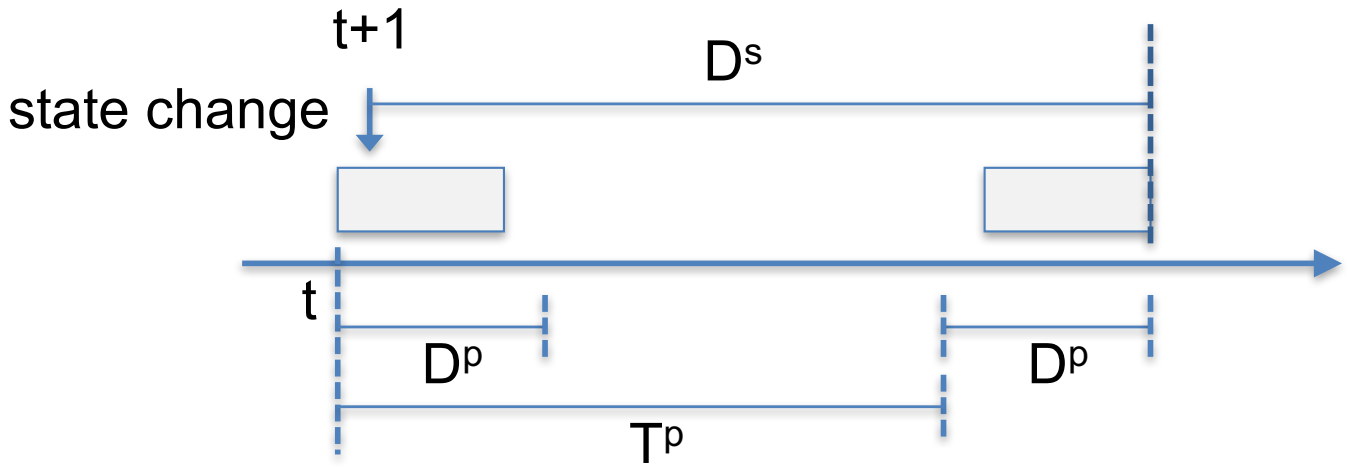
Sporadic Task Scheduling

- Transformation of the sporadic task to a quasi-periodic task
- Sporadic task parameters: $mint^s$, D^s, C^s
- Quasi-periodic task parameters

$$C^p = C^s$$

$$D^p \leq D^s, \text{ e.g., } D^p = C^p$$

$$T^p = \min(mint^s, D^s - D^p + 1)$$



Sporadic Server Task

- Sporadic-task transformation may yield poor processor utilization, especially if D^s is small compared to $mint^s$.
- We can define a server task for the sporadic request that has a short latency
- The server is scheduled in every period, but is only executed if the sporadic request actually appears. Otherwise the other tasks are scheduled
- This will require a task set in which all the other tasks have a laxity of at least the execution time of the server task.

Priority Inversion

- Consider tasks with mutual exclusion constraints.
- Priority inversion is a phenomenon that occurs when a higher-priority task is blocked by a lower-priority task.
- Direct blocking: a high-priority task must not preempt the exclusive resource use by a low-priority task
- Indirect blocking of a high-priority task by a medium-priority task – the medium priority task preempts a low-priority task that holds a shared resource – has to be avoided.

Priority Inversion (2)

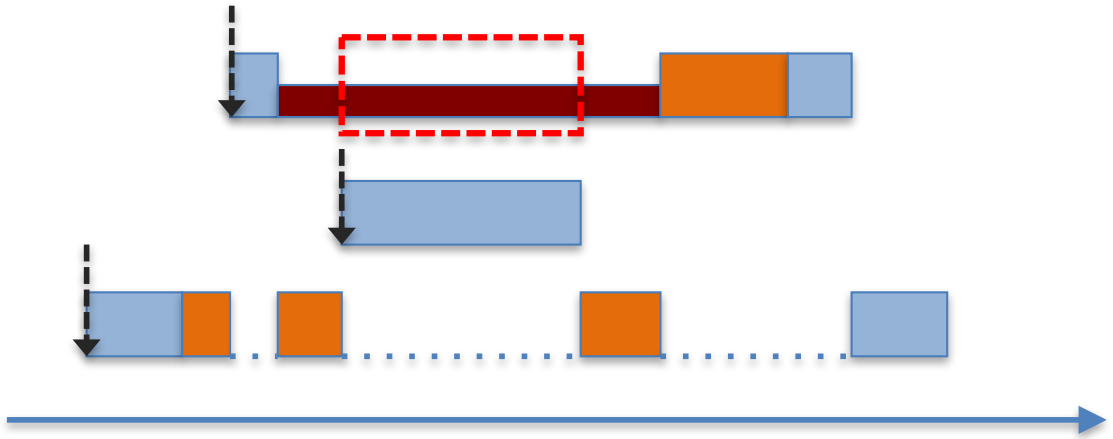
- In the shown example the high-priority task is indirectly blocked by the medium-priority task (dashed box).

Task 1 and Task 3 use the same resource

Task 3, highest priority

Task 2, medium priority

Task 1, lowest priority



… task executes

… mutex resource use

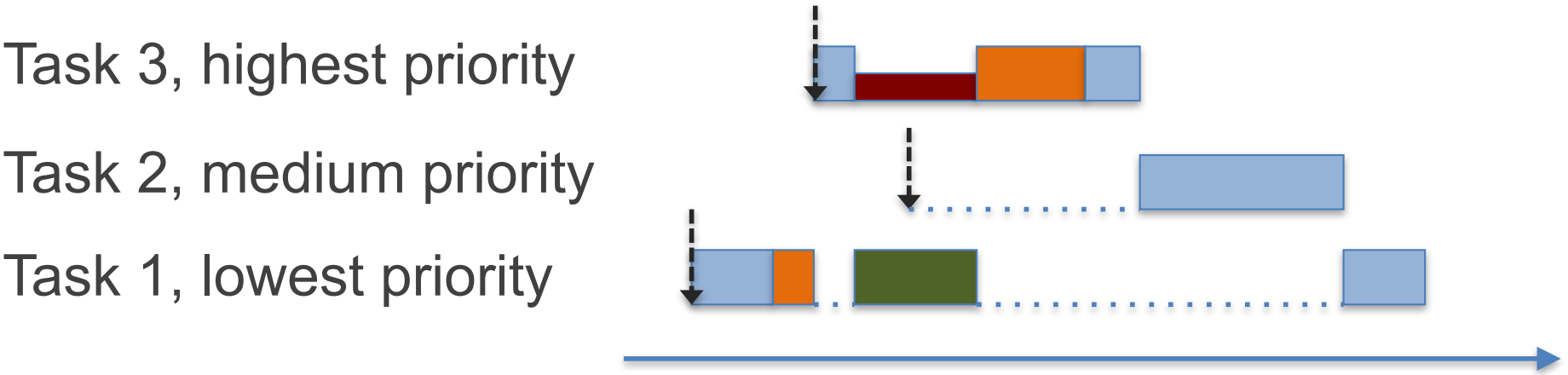
… task is blocked, priority inversion

… task is preempted

… task indirectly blocked

Priority Inheritance

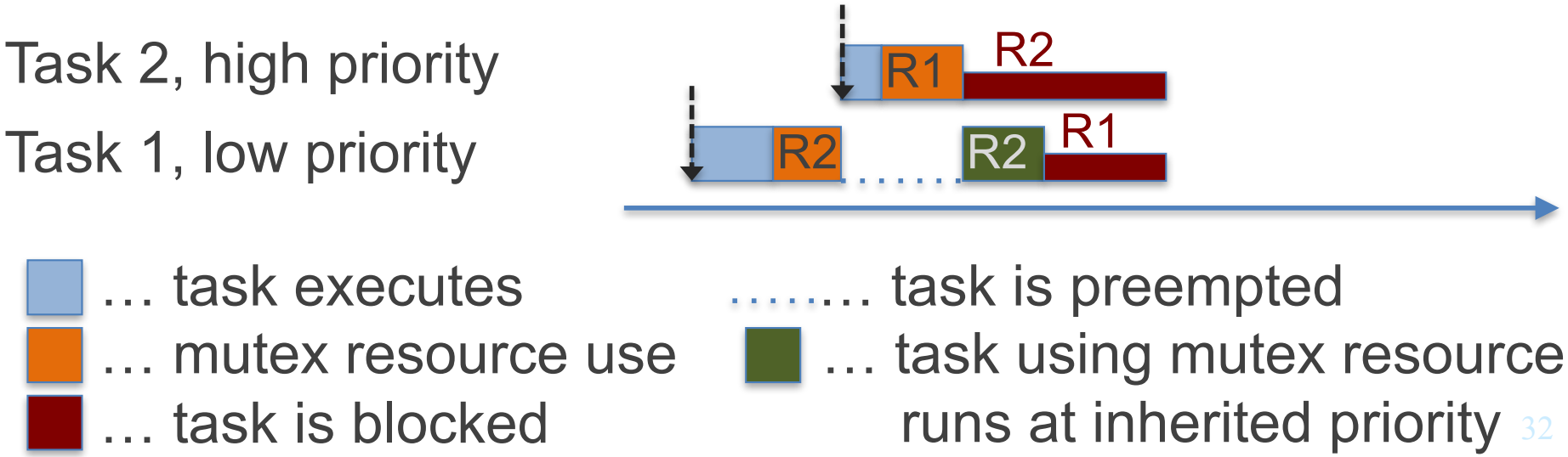
- When a low-priority task blocks one or more tasks of higher priority, it temporarily assumes the highest priority of a task it blocks



… task executes
… mutex resource use
… task is blocked
…… task is preempted
… task using mutex resource runs at inherited priority

Priority Inheritance (2)

- The priority-inheritance protocol does not prevent deadlocks
- Example
  1. Task 1 locks R2
  2. Task 2 preempts Task 1 and locks R1
  3. Task 2 tries to lock R2 but fails
  4. Task 1 inherits priority from Task 2 but blocks when trying to lock R1



Priority Ceiling Protocol

- Each process has a default priority.
- Assign a priority ceiling to each resource:
  The priority ceiling equals the priority of the highest-priority task that uses the resource.
- At each time instant a task executes at a dynamic priority that is the maximum of its own static priority and the ceiling values of all recources that it has locked.

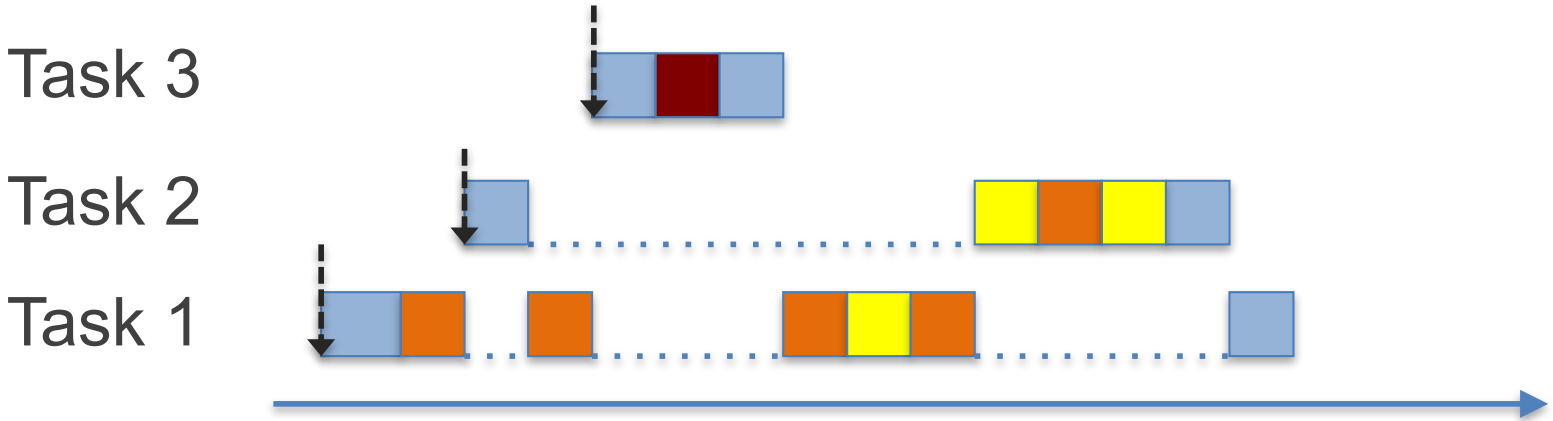
⇨ A task can only assume a new resource if the task's priority is higher than the priority ceilings of all the resources locked by other tasks.

Priority Ceiling Protocol – Example

Task 3: … P(S1) … V(S1) … highest priority

Task 2: … P(S2) … P(S3) … V(S3) … V(S2) … medium priority

Task 1: … P(S3) … P(S2) … V(S2) … V(S3) … lowest priority



Critical section guarded by Sx (priority ceiling):

… S1 (high) … S2 (medium) … S3 (medium)

Calculating the Maximal Blocking Time

- Let us assume a process has *K* critical sections, i.e., it can be blocked at most *K* times
- Define: *usage(k, i)* is 1 if the resource used in critical section *k* is used by at least one task with lower and one task with higher or equal priority than $Task_i$, otherwise it is 0.
- *C(k)* is the WCET of critical section *k.*
- The maximum blocking time B_i of $Task_i$ is:

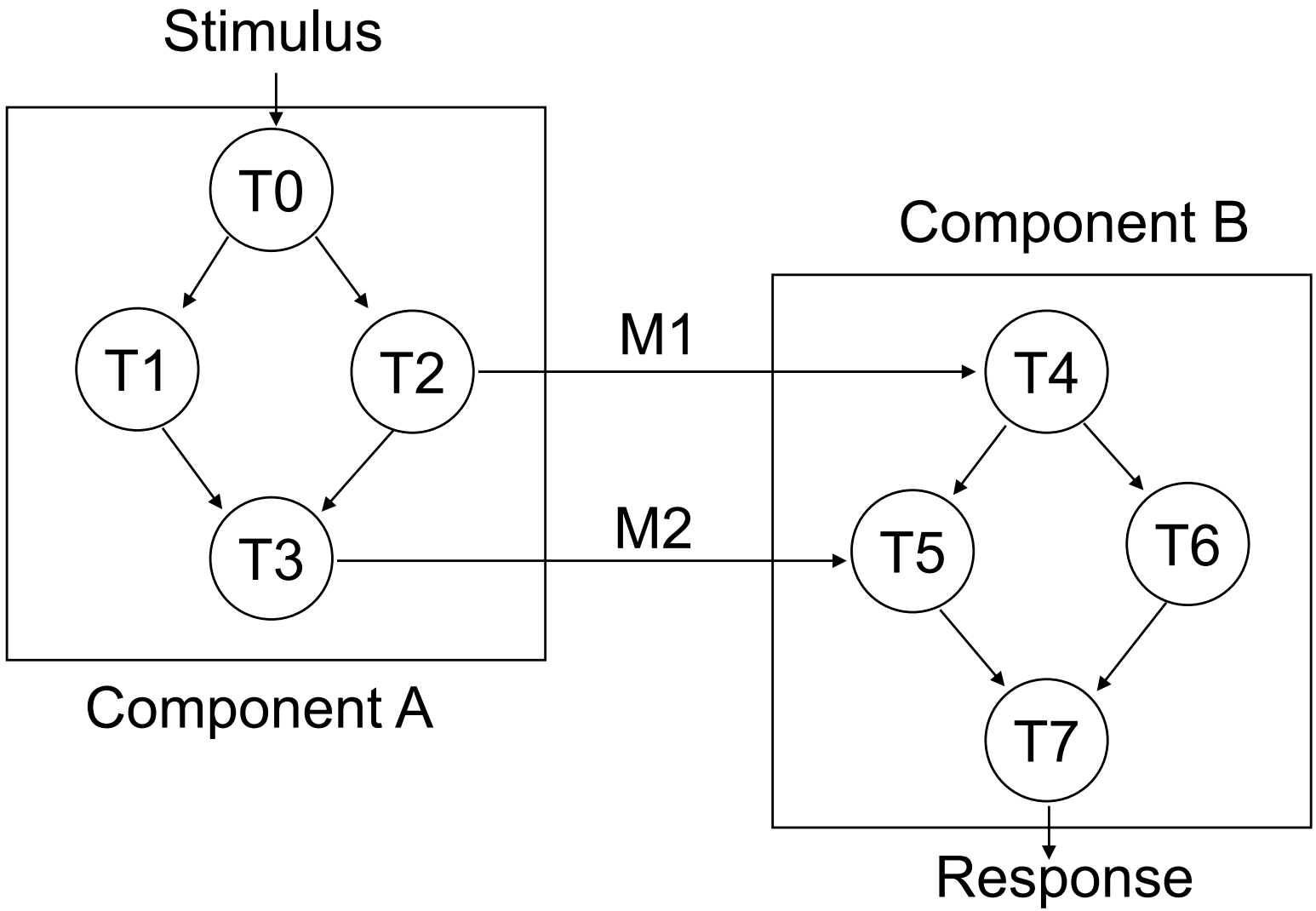
$$B_i = \sum_{k=1}^{K} usage(k, i)\ C(k)$$

Response Time with Blocking

- Using the calculated worst-case blocking times, the maximum response time of $Task_i$ can be described by the following recurrence equation:

$$R_i = C_i + B_i + \sum_{j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

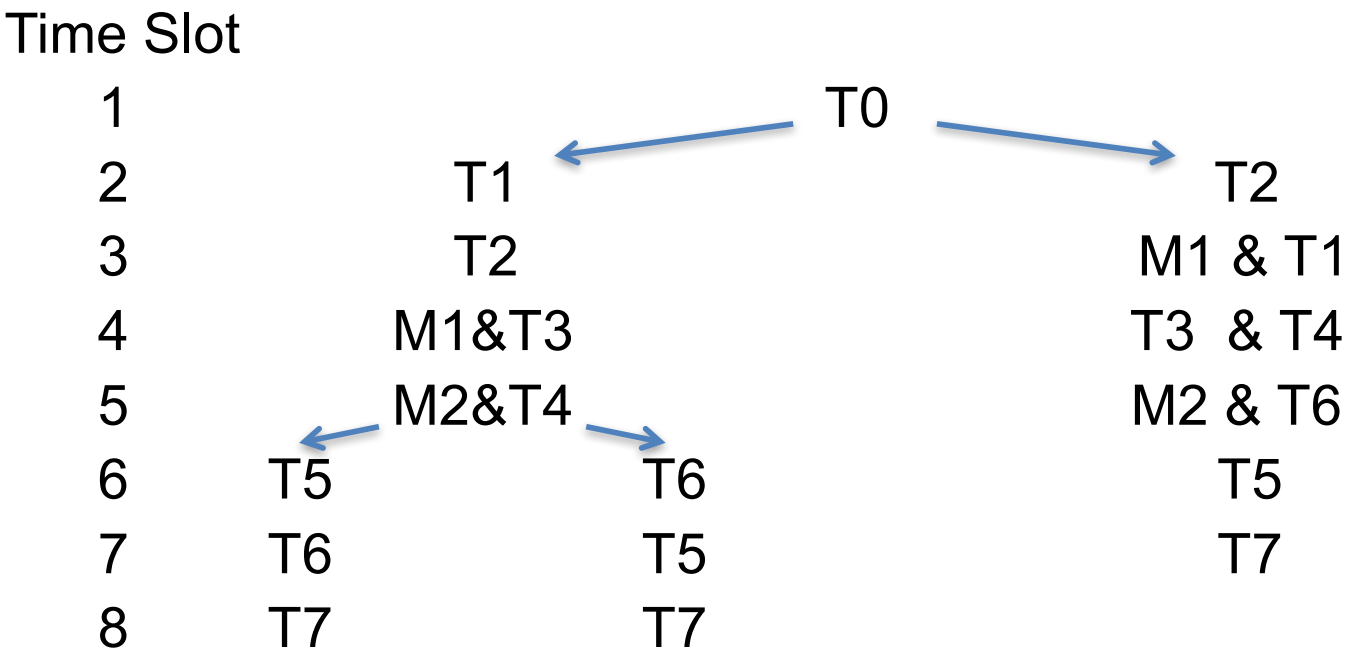
Static Scheduling – Precedence Graph



Static Scheduling – A Search Problem

- The goal of the static (pre runtime) scheduler is to find a path through a search tree that
  - Meets all deadlines
  - Observes all constraints (mutual exclusion, precendence, etc.)
- The scheduler generates a table (task description list … TaDL) that the dispatcher of a time-triggered operating system interprets at runtime.
- Schedule construction by heuristic search
  - heuristic function estimates expected response time of partial solutions.
  - If the expected response time is larger than the allowed response time, the respective branch of the search tree is pruned.

Static Scheduling – Search-Tree Example



- Schedulability Test: by construction of the schedule.
  If the task set is not schedulable then the scheduler will not find a schedule.

Points to Remember

- To provide deadline guarantees the task model must be defined, the set of tasks, the task timing parameters and interferences (mutex) must be known at analysis time
- Schedulability tests are a tool to judge task sets
- Schedulers: RMS (FPS), EDF, LLF
- Mutex: Priority inversion → priority ceiling protocol
- Run-time versus static scheduling
  - Run-time scheduler: flexibility (?), might cope with temporary overload – hope; scheduling decisions are taken at runtime; schedulability test has to cover all possible scenarios
  - Static scheduler: rigid interpretation of dispatch table, little run-time overhead (lookup); has to find one feasible schedule = successful completion of schedulability test