ECSE 421 Lecture 15: Introduction to Scheduling

ESD Chapter 6

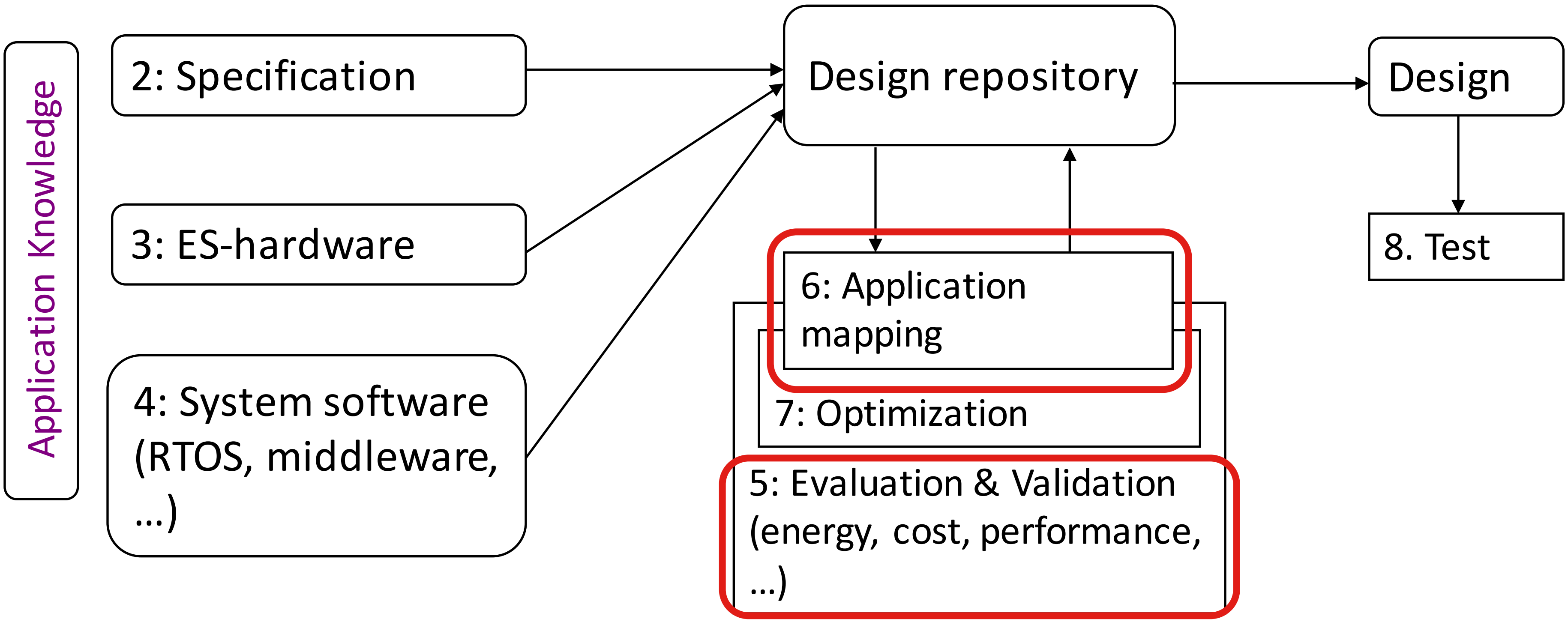
Last Time

- Design Objectives
  - Energy and Power
  - Temperature
  - Reliability
  - Electromagnetic Compatibility
- Validation
  - Simulation
  - Emulation
  - Formal Verification

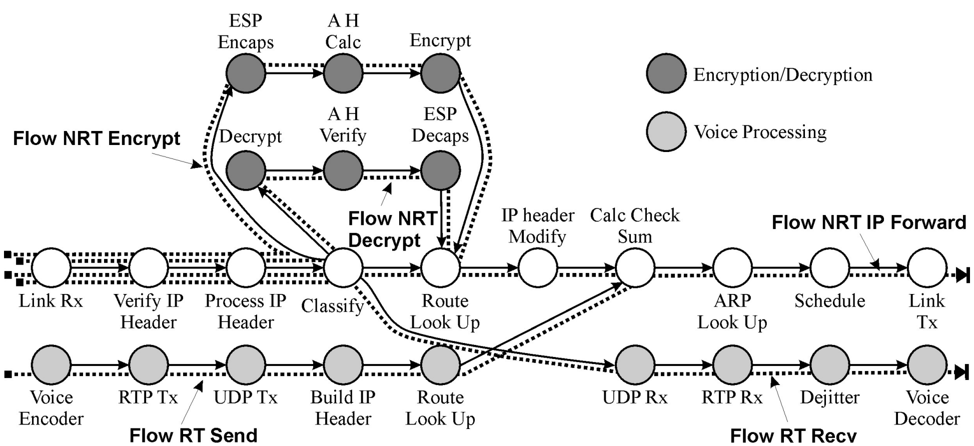
Where Are We?

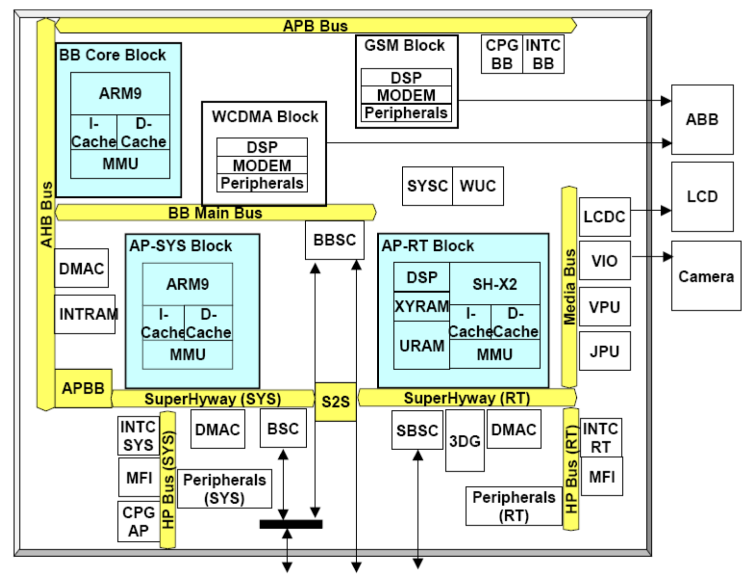
	R	4-Feb-2016	L07	DES / Von Neumann Model of Computation	2.8-2.10	5	LA2	LA1	
5	T	9-Feb-2016	L08	Sensors	3.1-3.2	7.3,12.1-6			
	R	11-Feb-2016	L09	Processing Elements	3.3	12.6-12			
6	T	16-Feb-2016		***No class***				LA2	
	R	18-Feb-2016	L10	More Processing Elements / FPGAs			LA3		
7	T	23-Feb-2016	L11	Memories, Communication, Output	3.4-3.6				
	R	25-Feb-2016		***Midterm exam: in-class, closed book***			P		Chapters 1-3
	T	1-Mar-2016		***No class***					Winter break
	R	3-Mar-2016		***No class***					Winter break
8	T	8-Mar-2016	L12	Embedded Operating Systems	4.1			LA3	
	R	10-Mar-2016	L13	Performance Evaluation	5.1-5.2				
9	T	15-Mar-2016	L14	More Evaluation and Validation	5.3-5.8		Project		
	R	17-Mar-2016		Catch-up Day					
10	T	22-Mar-2016	L15	Introduction to Scheduling	6.1-6.2.2	8			
	R	24-Mar-2016	L16	Scheduling Aperiodic Tasks	6.2.3-6.2.4				
11	T	29-Mar-2016	L17	Scheduling Periodic Tasks	6.2.5-6.2.6	10			
	R	31-Mar-2016	L18	HW/SW Partitioning	6.3				
12	T	5-Apr-2016	L19	Mapping Applications to Multiprocessors	6.4				
	R	7-Apr-2016	L20	Intro to Compile-time Optimization	7.1-7.2				
13	T	12-Apr-2016	L21	Energy/Memory-aware Compilation	7.3.1-7.3.3				Demo week
	R	14-Apr-2016	L22	Further Optimization	7.3.4-7.4				Demo week
	F	15-Apr-2016		***Last day of classes***				Project	Demo week
15	R	28-Apr-2016		***Final Exam: closed book, cumulative***					9:00 AM

Hypothetical Design Flow



Mapping of Applications to Platforms





Problem Description

- **Given**
  - A set of applications
  - Scenarios describing how these applications will be used
  - A set of candidate architectures comprising
    - (Possibly heterogeneous) processors
    - (Possibly heterogeneous) communication architectures
    - Possible scheduling policies
- **Find**
  - A mapping of applications to processors
  - Appropriate scheduling techniques (if not fixed)
  - A target architecture (if DSE is included)
- **Objectives**
  - Keeping deadlines and/or maximizing performance
  - Minimizing cost, energy consumption

Tools urgently needed!

Related Work

- Mapping to ECUs in automotive design
- Scheduling theory:
  Provides insight for the mapping *task* → *start times*
- Hardware/software partitioning:
  Can be applied if it supports multiple processors
- High performance computing (HPC)
  Automatic parallelization, but only for
  - single applications, and
  - fixed architectures, with
  - no support for scheduling, and generally
  - memory and communication model are different
- High-level synthesis
  Provides useful terms like scheduling, allocation, assignment
- Optimization theory

Scope of Mapping Algorithms

- Useful terms from hardware synthesis:
  - **Resource Allocation**
    Decision concerning type and number of available resources
  - **Resource Assignment**
    Mapping: Task → (Hardware) Resource
  - ***xx* to *yy* binding**
    Describes a mapping from behavioral to structural domain, e.g. task to processor binding, variable to memory binding
  - **Scheduling**
    Mapping: Tasks → Task start times
    Sometimes, resource assignment is considered being included in scheduling.

Classes of Algorithms In This Course

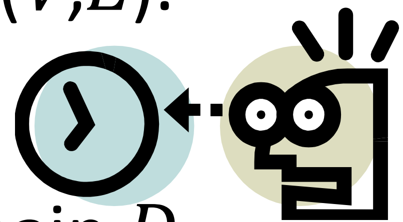
- **Classical scheduling algorithms**
  Mostly for independent tasks and ignoring communication, mostly for uniprocessors and homogeneous multiprocessors
- **Dependent tasks as in architectural synthesis**
  Initially designed in different context, but applicable
- **Hardware/software partitioning**
  Dependent tasks, heterogeneous systems, focus on resource assignment
- **Design space exploration using evolutionary algorithms**; Heterogeneous systems, incl. communication modeling

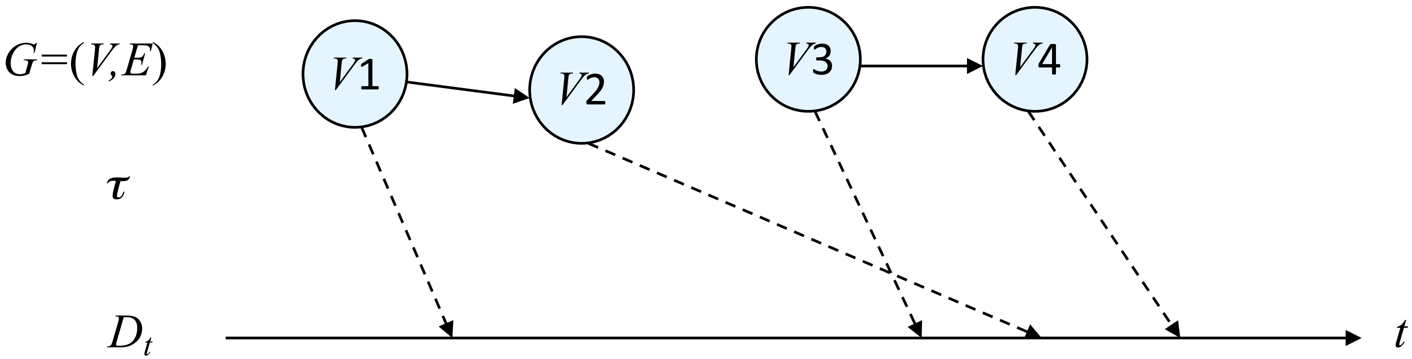
Real-time Scheduling

- Assume that we are given a task graph G=(V,E).
- **Def.**: A schedule τ of G is a mapping

  $$V \rightarrow D_t$$

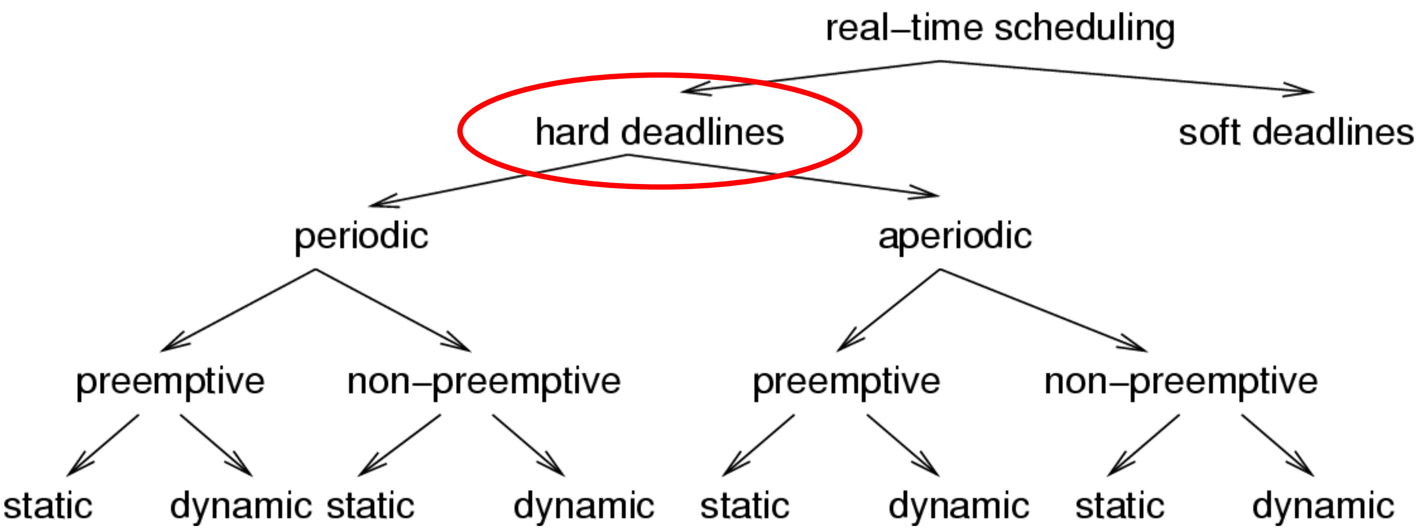
  of a set of tasks V to start times from domain D_t.

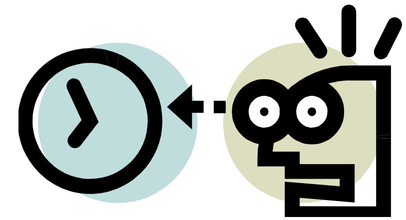




- Schedules have to respect a number of constraints
  - Resource constraints, dependency constraints, deadlines
- **Scheduling** = finding such a mapping

Hard and Soft Deadlines

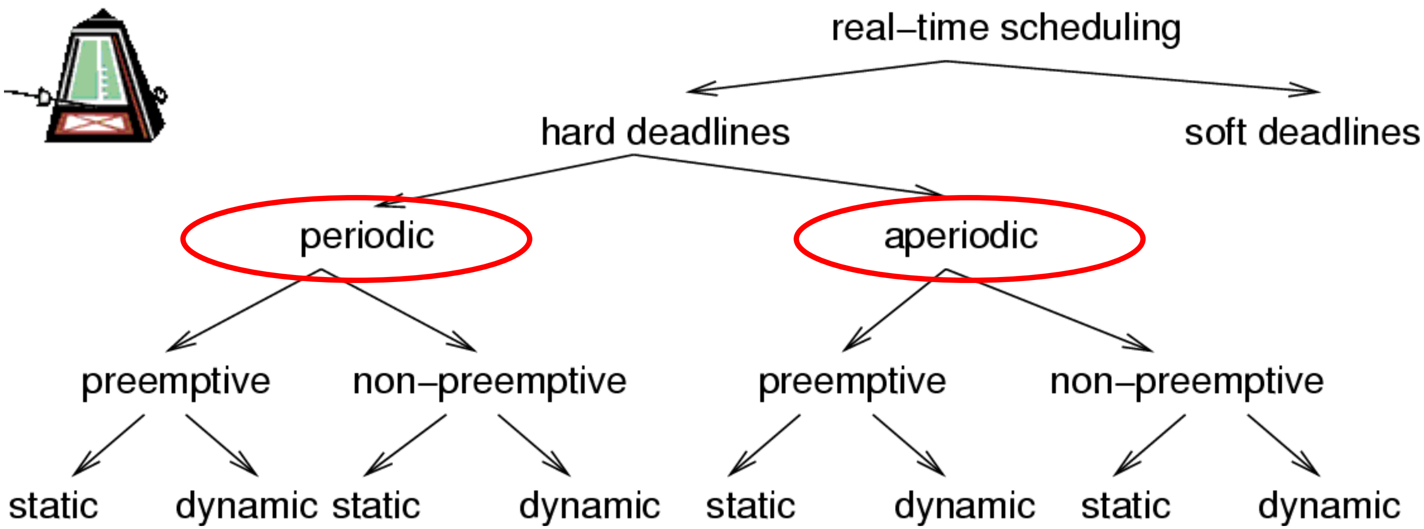




- **Def.**: A time-constraint (deadline) is **hard** if not meeting that constraint could result in a catastrophe [Kopetz, 1997]
- All other time constraints are called **soft**
- We will focus on hard deadlines

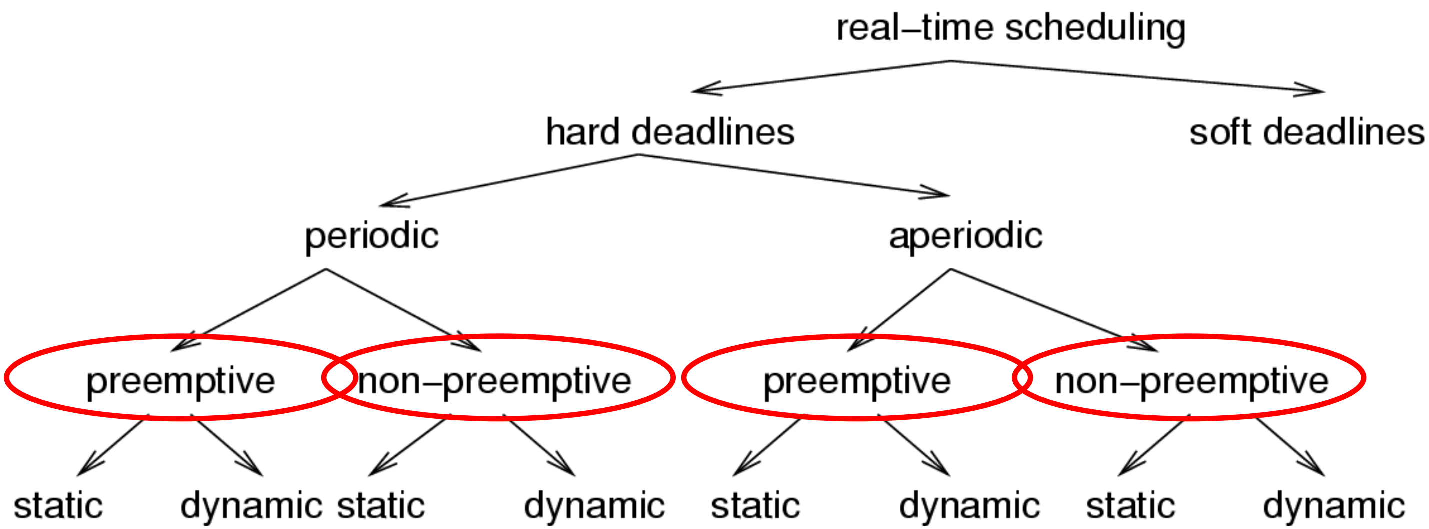
McGill

Periodic and Aperiodic Tasks



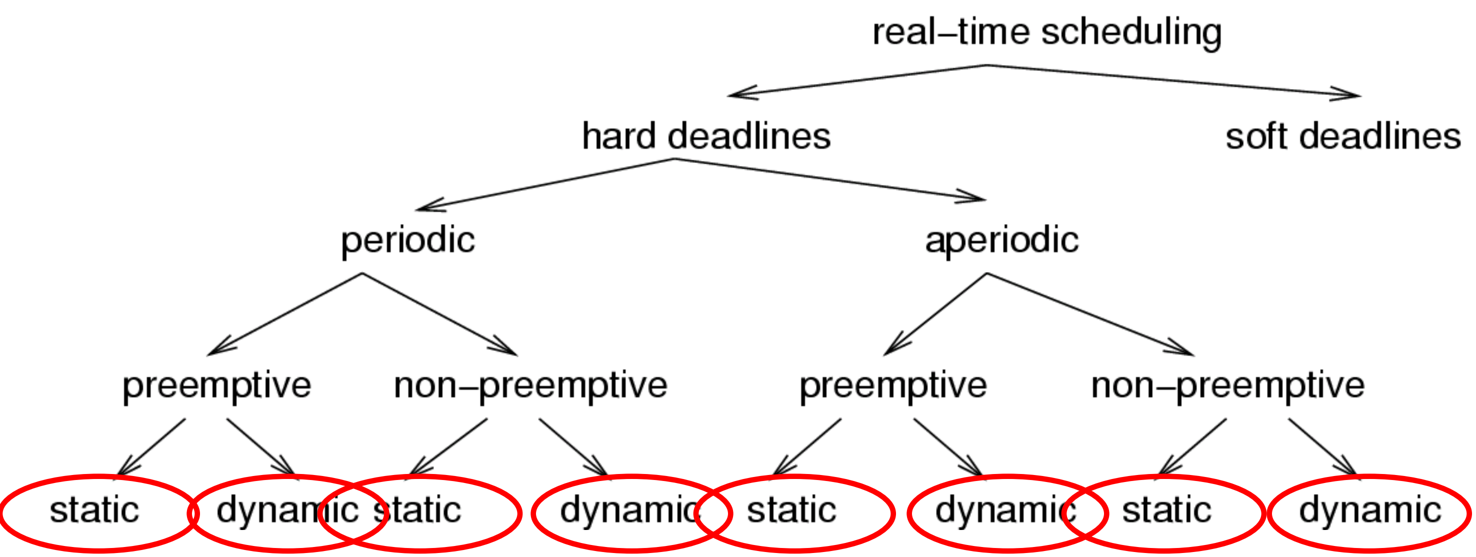
- **Def.**: Tasks that execute once every p units of time are called **periodic.** p is called their period. Each execution of a periodic task is called a **job**.
- All other tasks are called **aperiodic**.
- **Def.**: Tasks requesting the processor at unpredictable times are called **sporadic**, if there is a minimum separation time between requests.

Preemptive and Non-preemptive



- Non-preemptive schedulers:
  - Tasks are executed until they are done
  - Response time for external events may be quite long
- Preemptive schedulers: To be used if
  - some tasks have long execution times, or
  - if the response time for external events needs to be short

Dynamic/Online Scheduling

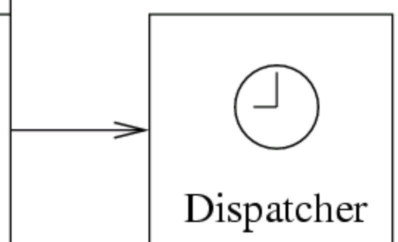


- Run-time processor allocation (scheduling)
  - Given tasks that have arrived so far,
  - How should a new task be assigned to a processor?

Static/Offline Scheduling

- Design-time processor allocation
  - Assumes *a priori* knowledge about
    - arrival times, (a_i)
    - execution times (c_i), and
    - deadlines (d_i)
  - Dispatcher allocates processor when interrupted by timer
  - Timer controlled by a table generated at design time

Time	Action	WCET
10	start T1	12
17	send M5	
22	stop T1	
38	start T2	20
47	send M3	



Dynamic vs. Static Scheduling

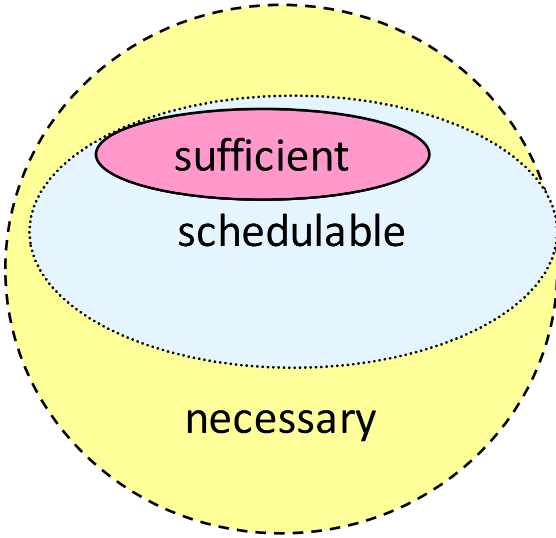
- With dynamic scheduling
  - Schedulability analysis is often difficult
  - High-priority external events are handled immediately
- With static scheduling
  - Scheduability analysis is easy: a schedule is obviously valid or not
  - Trade-off between utilization and the response time of sporadic tasks

Centralized and Distributed Scheduling

- **Mono- and multi-processor scheduling**
  - Simple algorithms handle single processors
  - More complex algorithms handle multiple processors
    - Algorithms for homogeneous multi-processor systems
    - Algorithms for heterogeneous multi-processor systems (includes HW accelerators as special case)
- **Centralized and distributed scheduling**
  - Multiprocessor scheduling on a single node
    - Global information, low communication cost
  - Coordinated scheduling across many nodes
    - High communication cost, limited global information

Schedulability

- **Def.**: A set of tasks is **schedulable** under a set of constraints if a schedule exists for that task set that satisfies all constraints.
- **Exact tests** are NP-hard in many situations
- **Sufficient tests**
  – Sufficient conditions for schedule checked
  – (Hopefully) small probability of not guaranteeing a schedule even though one exists
- **Necessary tests**
  – Checking necessary conditions
  – Used to show no schedule exists
  – There may be cases in which no schedule exists and we cannot prove it

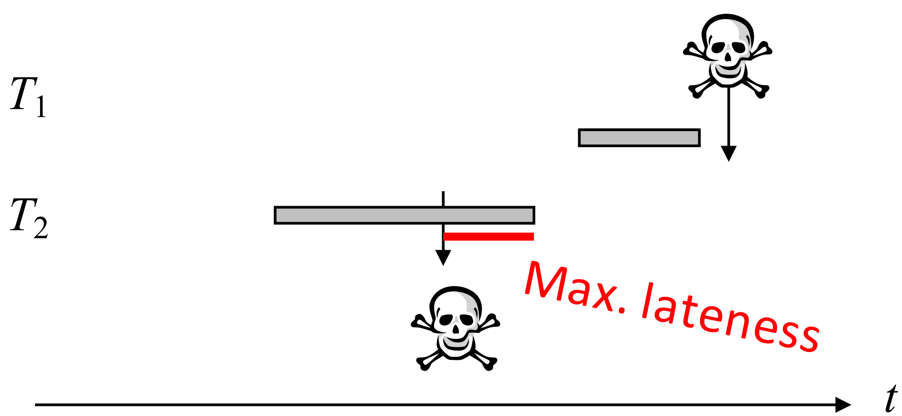


Cost Functions

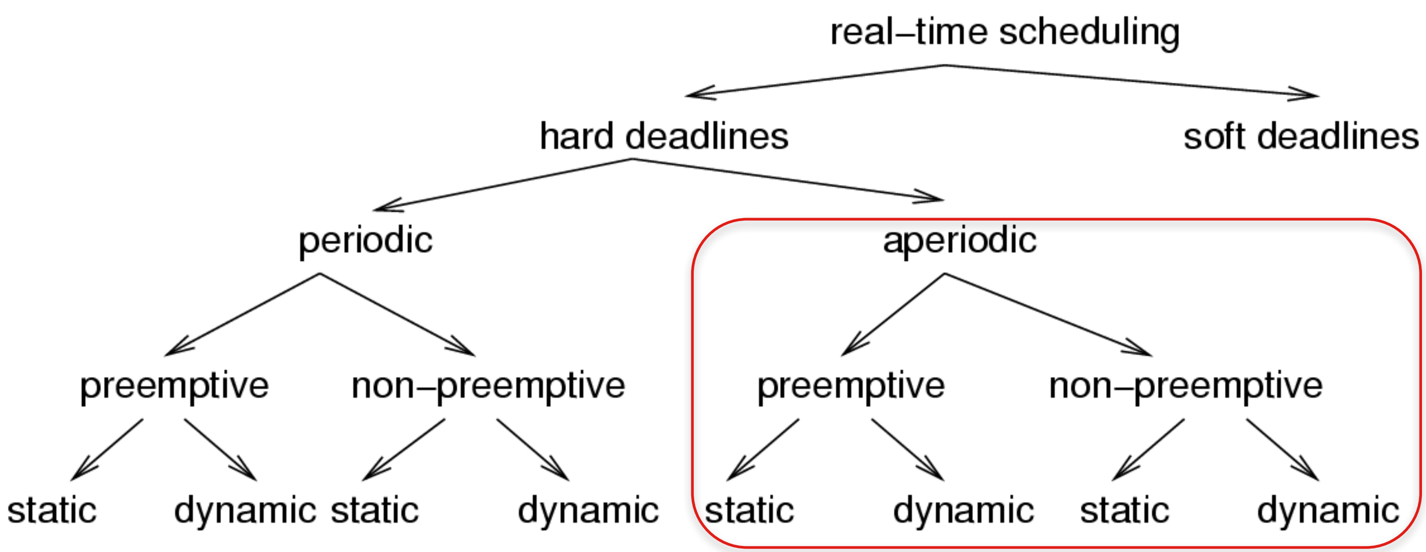
- **Cost function**: the quantitative objective being minimized by an algorithm
  - Different algorithms minimizing different functions
- **Def.**: **Maximum lateness** =

  $max_{all\ tasks}$ (*completion time – deadline*)

  Is < 0 if all tasks complete before deadline



Aperiodic Task Scheduling

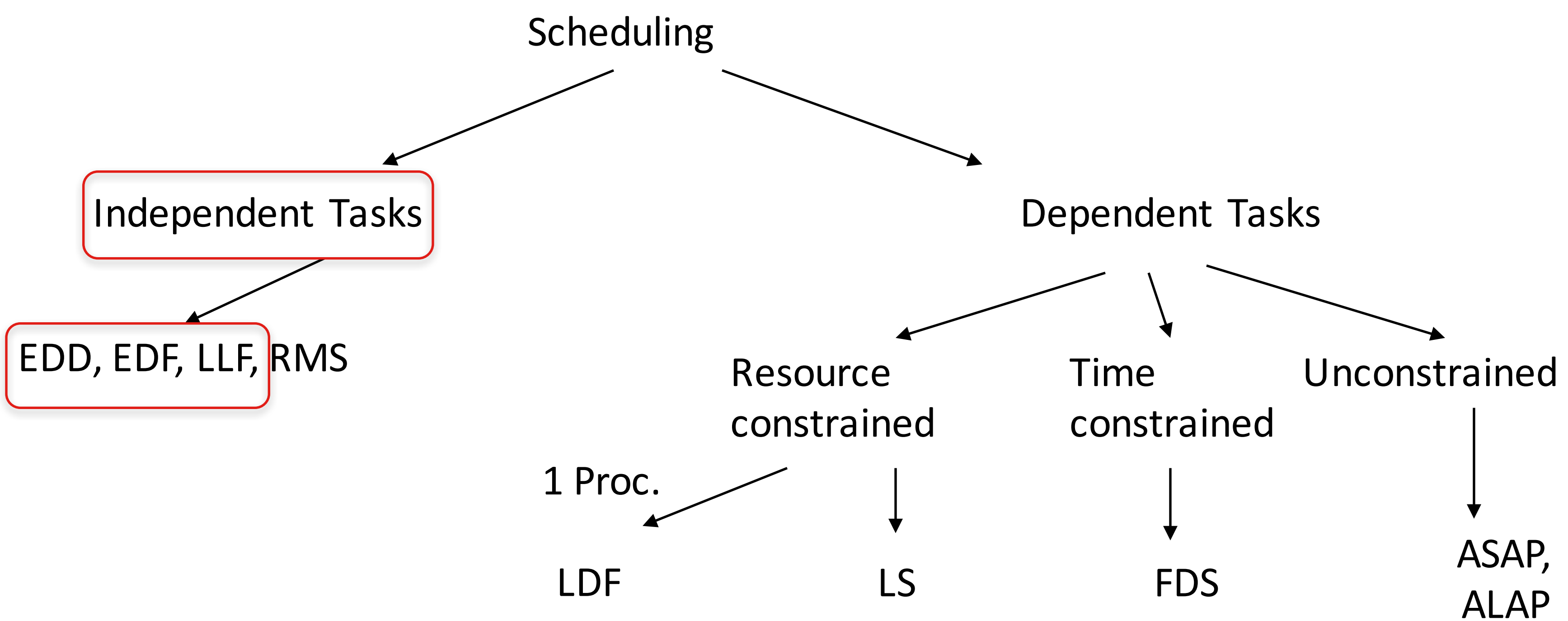


- Scheduling of aperiodic tasks
  - With and without preemption
  - Dynamic and static techniques

Classes of Algorithms In This Course

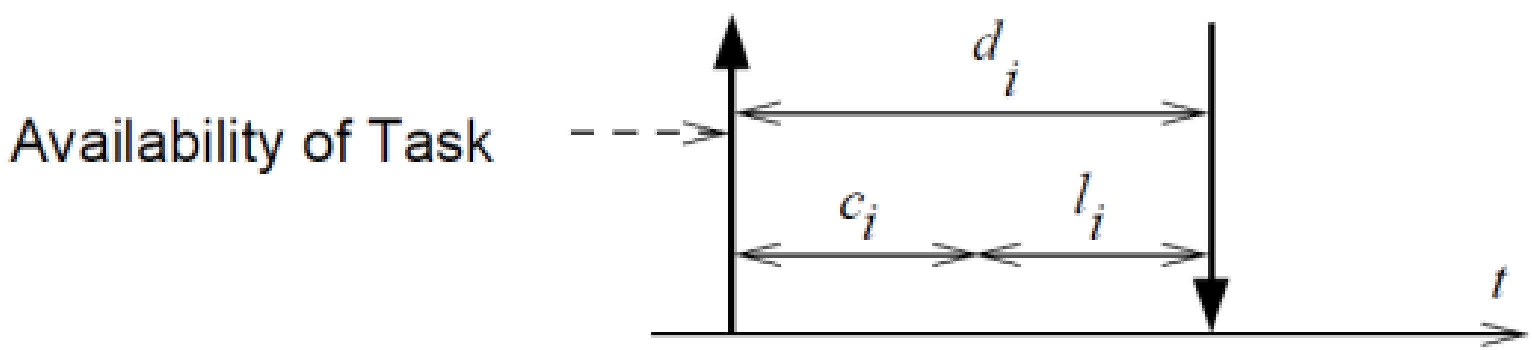
- **Classical scheduling algorithms**
  Mostly for independent tasks and ignoring communication, mostly for uniprocessors and homogeneous multiprocessors
- **Dependent tasks as in architectural synthesis**
  Initially designed in different context, but applicable
- **Hardware/software partitioning**
  Dependent tasks, heterogeneous systems, focus on resource assignment
- **Design space exploration using evolutionary algorithms**; Heterogeneous systems, incl. communication modeling

Classification of Scheduling Problems



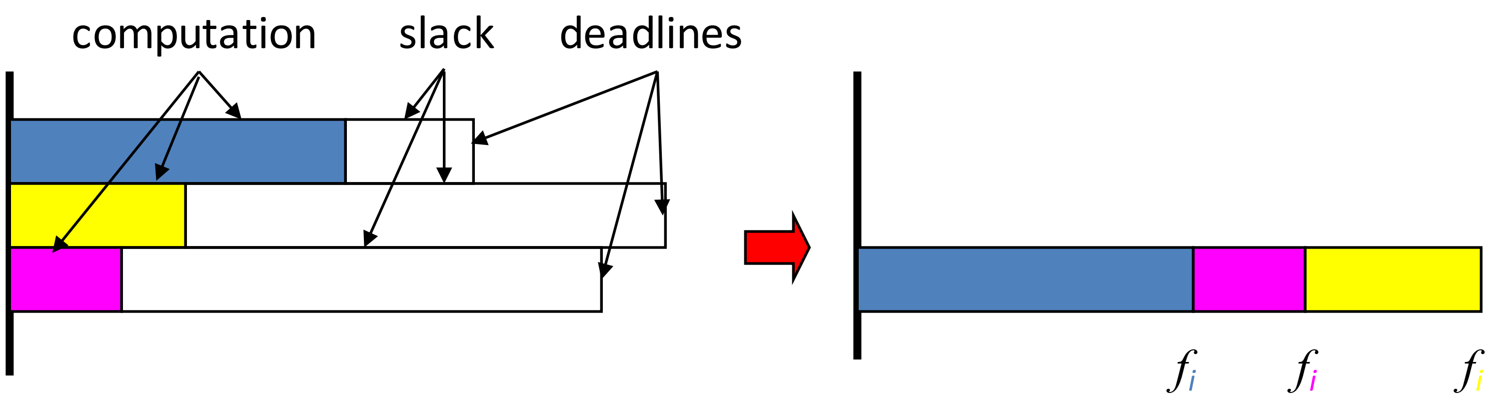
Aperiodic Scheduling: Independent Tasks

- Let $\{T_i\}$ be a set of tasks. Let:
  - c_i be the execution time of T_i,
  - d_i be the **deadline interval**
    - The time between T_i becoming available and when T_i has to finish execution
  - l_i be the **laxity** or **slack**, defined as $l_i = d_i - c_i$
  - f_i be the finishing time



Uniprocessor With Equal Arrival Times

- Preemption is useless
- **Earliest Due Date** (EDD): Execute task with earliest due date (deadline) first



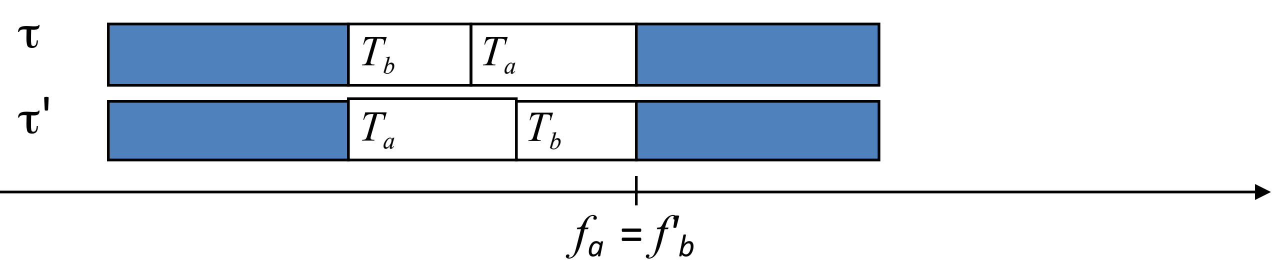
- EDD requires all tasks to be sorted by their (absolute) deadlines
  – Its complexity is therefore O(n log(n))

Optimality of EDD

- EDD is optimal since it follows Jackson's rule:
  - Given a set of n independent tasks, any algorithm that executes the tasks in order of non-decreasing (absolute) deadlines is optimal with respect to minimizing the maximum lateness.
- Proof (See Buttazzo, 2002):
  - Let τ be a schedule produced by any algorithm A
  - If $A \neq$ EDD $\rightarrow \exists\ T_a, T_b, d_a \leq d_b, T_b$ immediately precedes T_a in τ.
  - Let τ' be the schedule obtained by exchanging T_a *and* T_b.

Exchanging T_a and T_b

- Max. lateness for T_a and T_b in τ is $L_{max}(a,b) = f_a - d_a$
- Max. lateness for T_a and T_b in τ' is
  $$L'_{max}(a, b) = \max(L'_a, L'_b)$$
- Two possible cases
  - $L'_a > L'_b$: $\rightarrow L'_{max}(a,b) = f'_a - d_a < f_a - d_a = L_{max}(a,b)$
    since T_a starts earlier in schedule τ'
  - $L'_a \leq L'_b$: $\rightarrow L'_{max}(a,b) = f'_b - d_b = f_a - d_b \leq f_a - d_a = L_{max}(a,b)$
    since $f_a = f'_b$ and $d_a \leq d_b$
- Therefore, $L'_{max}(a,b) \leq L_{max}(a,b)$



EDD is Optimal

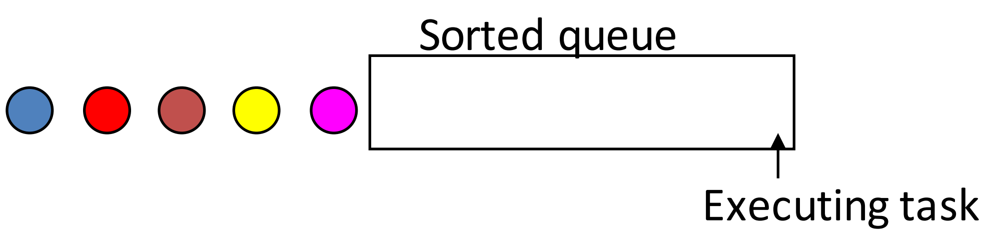
- Any schedule τ with lateness L can be transformed into an EDD schedule τ^n with lateness $L^n \leq L$, which is the minimum lateness
- EDD is optimal (*q.e.d.*)

Earliest Deadline First (EDF) (1)

- Horn’s Theorem
  - When tasks have different arrival times, preemption potentially reduces lateness
- **Theorem** [Horn74]: Given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

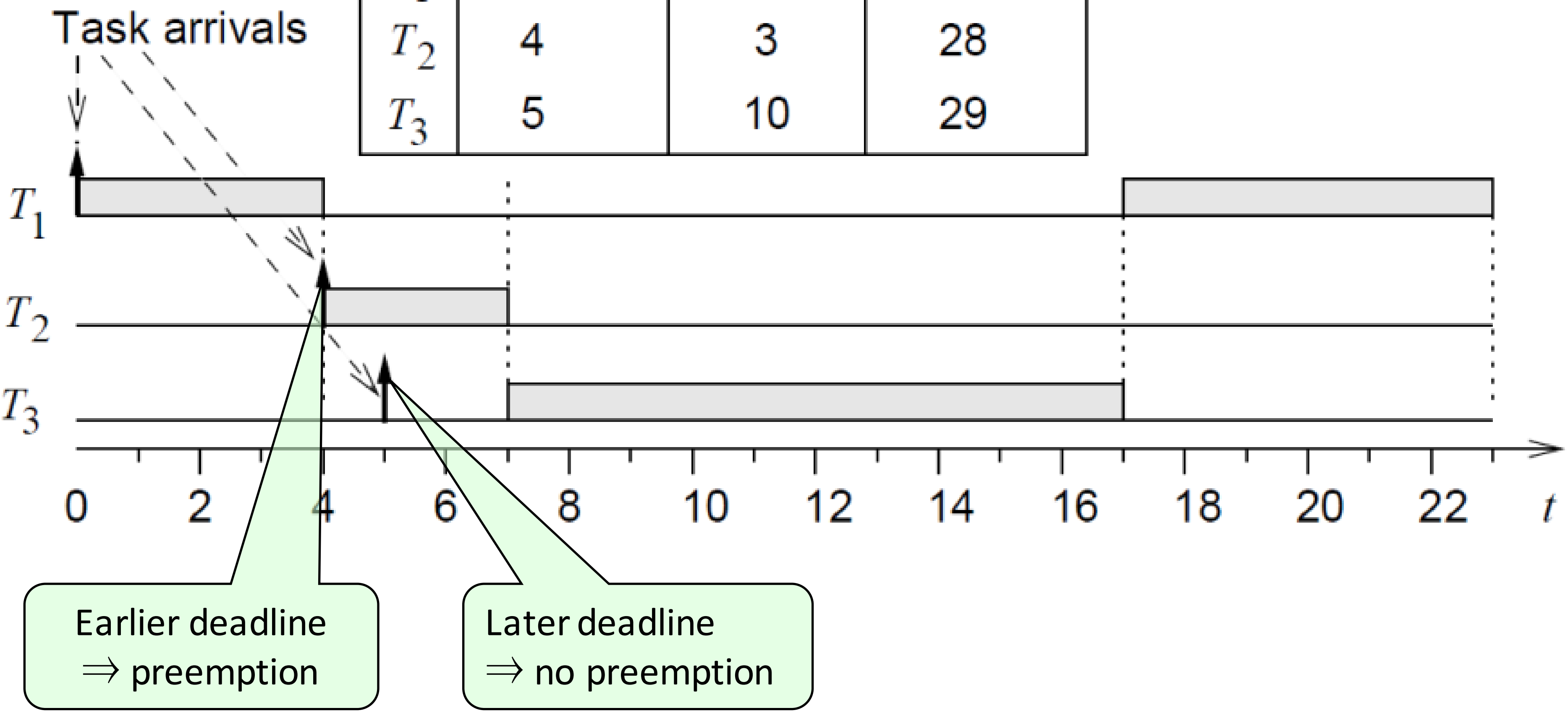
Earliest Deadline First (EDF) (2)

- Earliest deadline first (EDF) algorithm:
  - When a new ready task arrives, insert it into a queue of ready tasks, sorted by absolute deadline
  - The task at head of queue is executed
  - Preempt the currently executing task if a newly arrived task is inserted at the head of the queue
- Straightforward approach with sorted lists (full comparison with existing tasks for each arriving task)
  - Algorithm requires run-time $O(n^2)$
  - Complexity goes down with binary search, bucket arrays



Earliest Deadline First (EDF) (3)

	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29

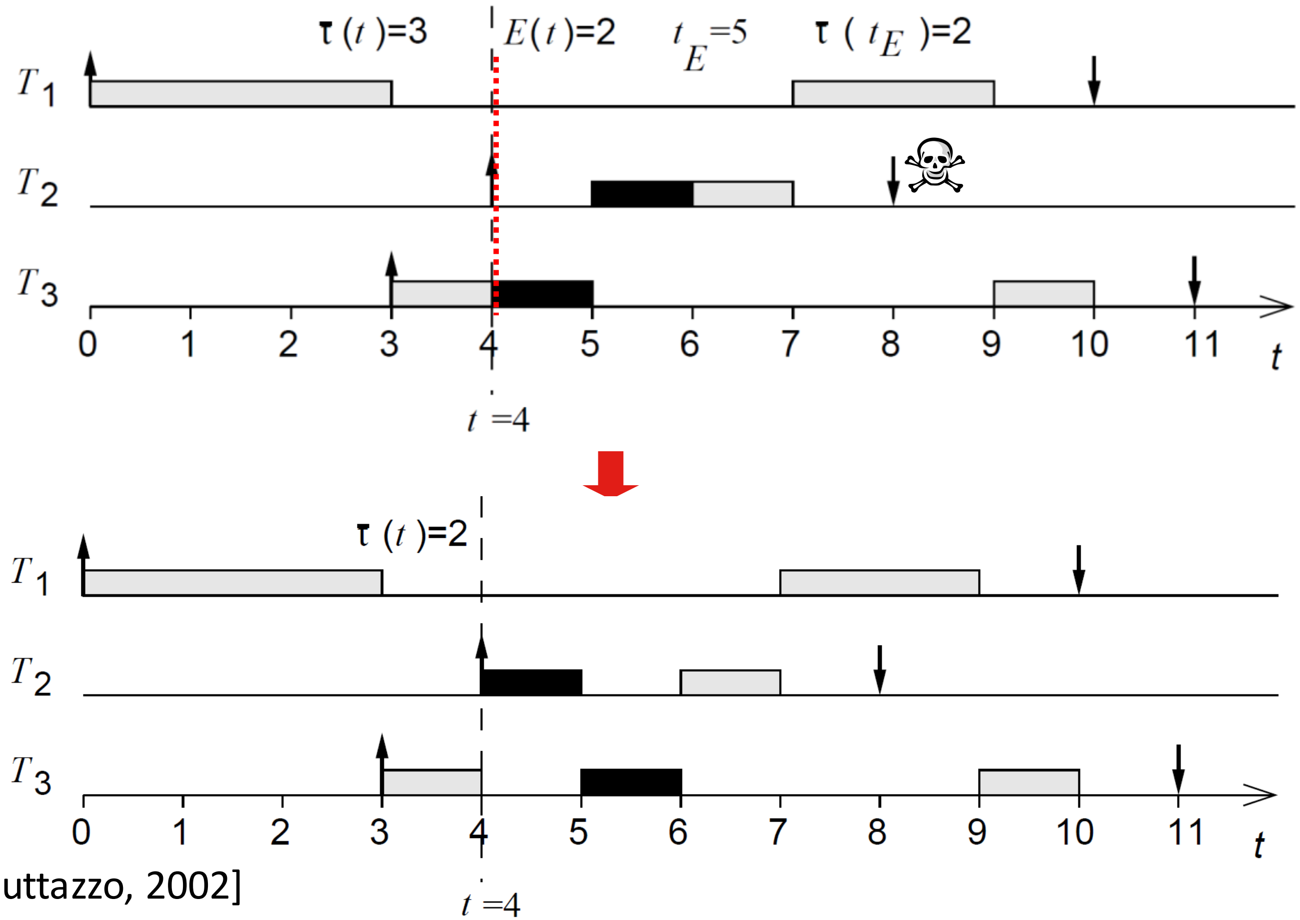


Optimality of EDF (1)

- To be shown: EDF minimizes maximum lateness.
- **Proof** (Buttazzo, 2002):
  - Let τ be a schedule produced by generic schedule A
  - Let τ_{EDF}: schedule produced by EDF
  - Preemption allowed: tasks executed in disjoint time intervals
  - τ divided into time slices of 1 time unit each
  - Time slices denoted by $[t, t+1)$
  - Let $\tau(t)$: task executing in $[t, t+1)$
  - Let $E(t)$: task which, at time t, has the earliest deadline
  - Let $t_E(t)$: time ($\geq t$) at which the next slice of task $E(t)$ begins its execution in the current schedule

Optimality of EDF (2)

- If $\tau \neq \tau_{EDF}$, then there exists time t: $\tau(t) \neq E(t)$
- Idea: swapping $\tau(t)$ and $E(t)$ cannot increase max. lateness.



If $\tau(t)$ starts at t=0 and $D=\max_i\{d_i\}$ then τ_{EDF} can be obtained from τ by at most D swaps

[Buttazzo, 2002]

Optimality of EDF (3)

Algorithm **`interchange`**:
{ **for** (t=0 **to** D-1) {
 if ($\tau(t) \neq E(t)$) {
 $\tau(t_E) = \tau(t)$;
 $\tau(t) = E(t)$; }}}

Using the same argument as in the proof of Jackson's algorithm, it can be shown that swapping cannot increase maximum lateness; hence EDF is optimal.

Does **`interchange`** preserve schedulability?

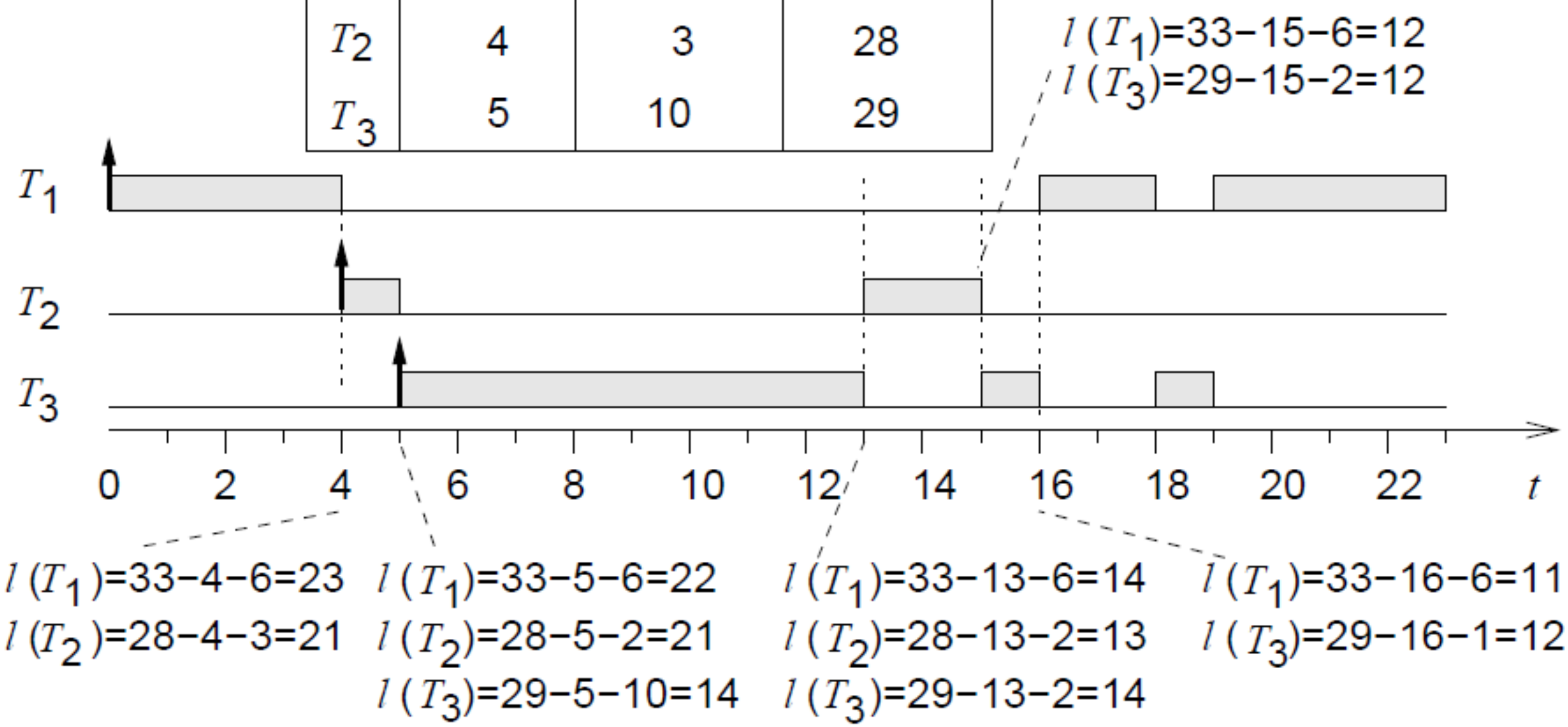
1. task $E(t)$ moved earlier
   - Deadline met in new schedule if met in τ
2. task $\tau(t)$ delayed
   - if $\tau(t)$ is feasible, then $(t_E+1) \leq d_E$, where d_E is the earliest deadline
   - Since $d_E \leq d_i$ for any i, we have $t_E+1 \leq d_i$, which guarantees schedulability of the delayed task

q.e.d.

Least Laxity First (LLF) (1)

- Priorities = decreasing function of the laxity
  - Lower laxity $\Rightarrow$ higher priority
  - Priority changes as relative laxity changes
  - Preemptive

	arrival	duration	deadline
T_1	0	10	33
T_2	4	3	28
T_3	5	10	29



Least Laxity First (LLF) (2)

- Pros
  - Detects missed deadlines early
  - Optimal for mono-processor systems
- Cons
  - Many context switches between tasks
  - Overhead for calls of the scheduler
    - *Note*: insufficient to call scheduler and re-compute laxity just at task arrival times
  - Dynamic priorities: cannot be used with a fixed prioity OS
  - Requires the knowledge of the execution time

Scheduling Without Preemption (1)

- **Lemma**: If preemption is not allowed, optimal schedules may have to leave the processor idle at certain times
- **Proof**: Suppose: optimal schedulers never leave processor idle

Scheduling Without Preemption (2)

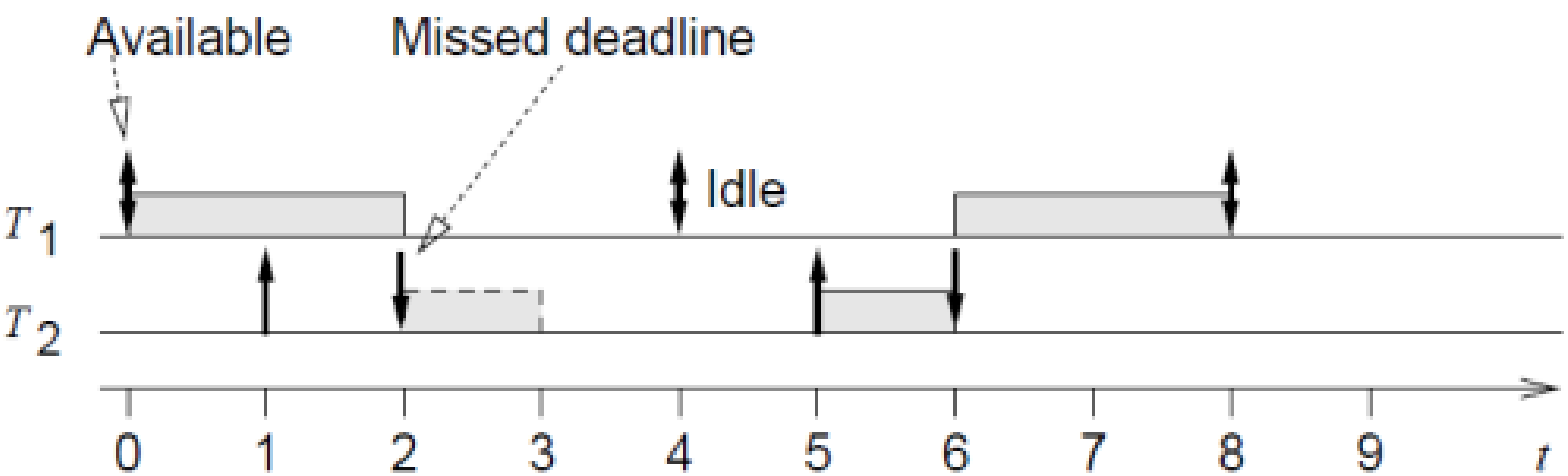
T_1: periodic, $c_1 = 2$, $p_1 = 4$, $d_1 = 4$

T_2: occasionally available at times $4*n+1$, $c_2 = 1$, $d_2 = 1$

T_1 has to start at $t = 0$

$\Rightarrow$ deadline missed, but schedule is possible (start T_2 first)

$\Rightarrow$ scheduler is not optimal $\Rightarrow$ contradiction! *q.e.d.*



Scheduling Without Preemption

- If preemption is not allowed
  - Optimal schedules may leave processor idle
  - Allows later arriving tasks to met early deadlines
- Knowledge about the future is needed for optimal scheduling algorithms
  - No online algorithm can decide whether or not to idle
- EDF is optimal among all scheduling algorithms not keeping the processor idle at certain times
- If arrival times are known a priori, the scheduling problem becomes NP-hard in general

Summary

- Application Mapping
  - The assignment of tasks to resources
  - Many formulations, depending on what parts of the design are fixed
- Scheduling
  - The assignment of tasks to start times
  - Many variations, depending on the nature of the application
- Independent, Aperiodic Task Scheduling
  - Earliest Due Date (EDD)
  - Earliest Deadline First (EDF)
  - Least Laxity First (LLF)

Next Time

- More aperiodic task scheduling
  - With precedence constraints (dependencies)
- Periodic scheduling
- Chapter 6.2