Real-Time Systems

Time-Driven and Partitioned Systems

Hermann Härtig

TECHNISCHE UNIVERSITÄT DRESDEN

WS 2018/19

Time-Driven vs. Event-Driven Scheduling

Time-Driven System

- at design time, a feasible schedule is computed
- the schedule is stored in a table
- *at certain points in time,* the scheduler dispatches tasks

Event-Driven System

- at design time, the feasibility of a set of tasks is determined depending on the scheduling algorithm
- *at certain events,* the scheduler computes a schedule and dispatches tasks

Outline

- time-driven in general
  (mostly following Jane Liu, Real-Time Systems)
  - cyclic schedules
  - tick-driven cyclic schedules
  - critical sections and precedence
- time and space partitioned systems
- time-driven communication
  (in separate lecture on communication)

Time-Driven Scheduling

Properties

- decisions, which job to execute next at specific time instants
- these are chosen a priori (before system begins execution)
- schedule is computed off-line

Typically restrictive assumptions: deterministic systems

- fixed number of tasks in systems
- with a priori fairly well known parameters (fixed inter-release times)
- tasks must be ready at their release times
- often used for safety-critical, hard real-time systems

Partitioned Systems

Usage scenario

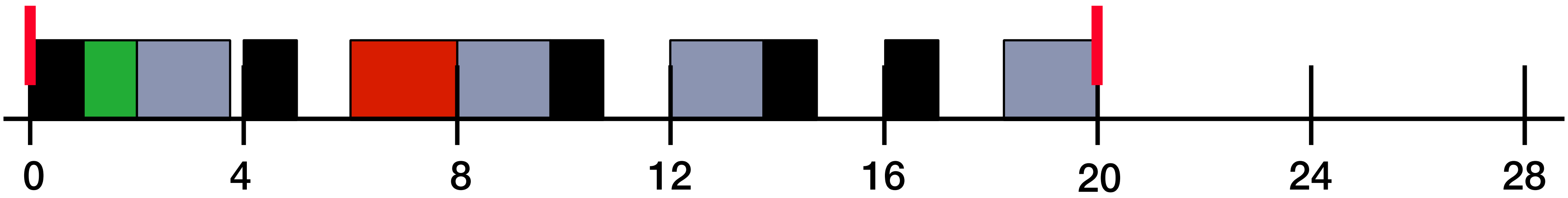
- separation of subsystems required for safety and/or security
- Subsystems are potentially very complex
- space partitioning:
  resources are allocated to one partition only
- time partitioning:
  timeline is partitioned into slots
  each slot belongs to one partition exclusively

Derive a Time-Driven Schedule

sufficient to find schedule for hyperperiod,
hyper period schedule is called a cyclic schedule

Example

- Tasks: (P_i, e_i):
  (4,1) (5,1.8) (20,1) (20,2)
- hyperperiod: 20
- arbitrary possible schedule for one hyperperiod:



Unused parts can be used for aperiodic jobs

Executing a Cyclic Schedule

store all scheduling points $(t_i, T(t_i))$ in table

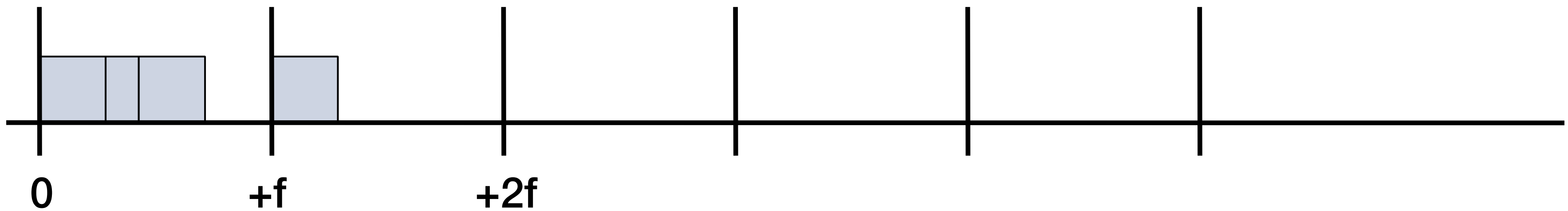
```
Do
   set timer to next decision point
   run current job in table
   wait for timer
Done
```

Cyclic Schedule

- note:
  scheduling actions at instants in time (not events!)
- contrast:
  priority driven systems scheduling decisions occur at events

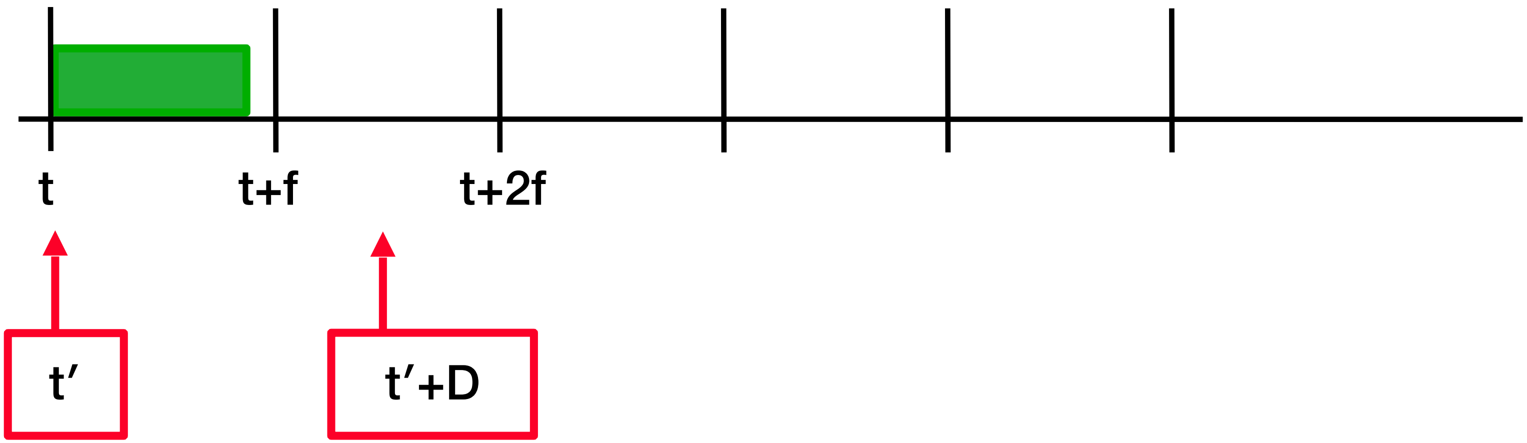
Tick-Driven Systems (Synchronous Systems)

- scheduling actions only at periodic instants of time
- time line divided into *frames* (Liu's terminology)
- structured variant of cyclic schedules
- no preemption within frames (in the normal case)
- at frame borders
  - scheduling decisions
  - check for violations
- question: What frame size?



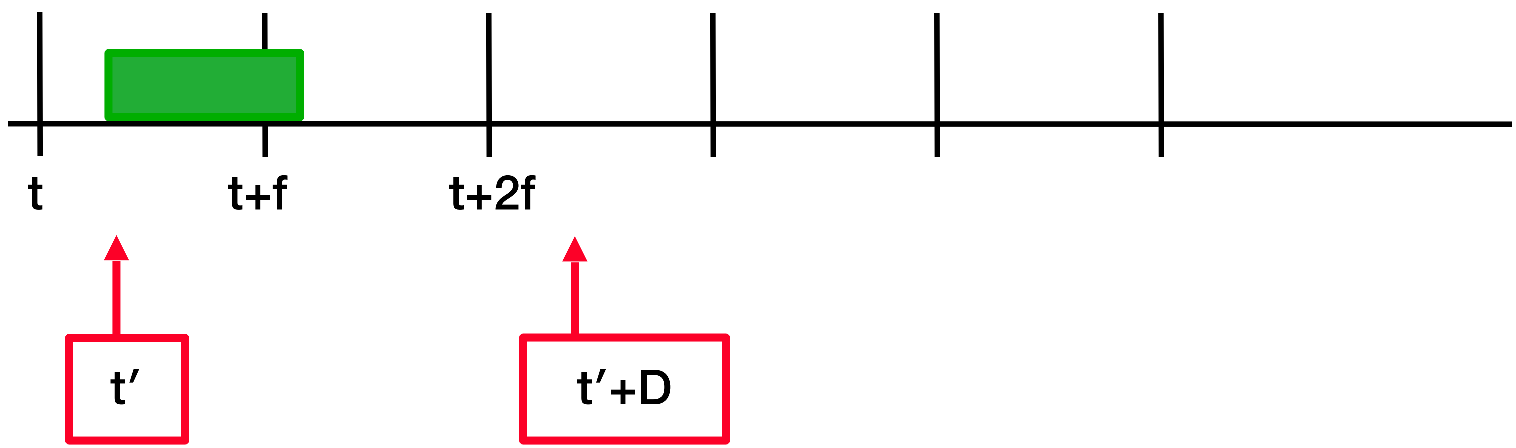
Frame Size f

1. at least one period should be multiple of f
   ensures an integer number of frames per hyperperiod
2. f ≥ max(ei) (avoids preemption)
3. one full frame (two boundaries)
   between release time t′ and deadline D
   for each job in all periods
   to enable policing by the scheduler before deadline



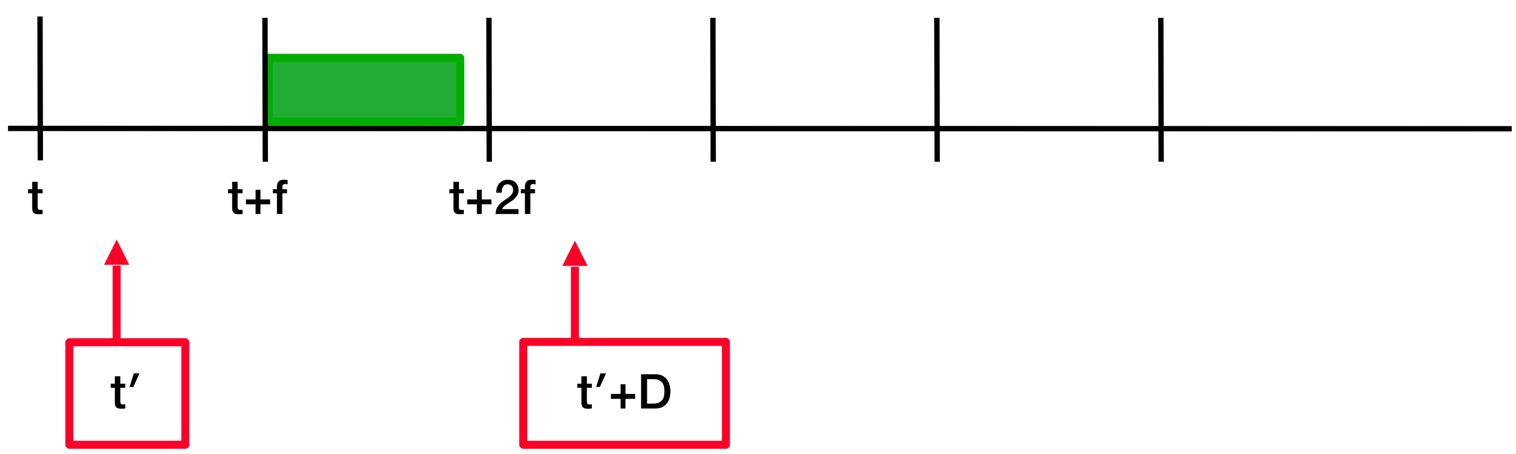
Frame Size f

1. at least one period should be multiple of f
   ensures an integer number of frames per hyperperiod
2. $f \geq \max(e_i)$ (avoids preemption)
3. one full frame (two boundaries)
   between release time t′ and deadline D
   for each job in all periods
   to enable policing by the scheduler before deadline



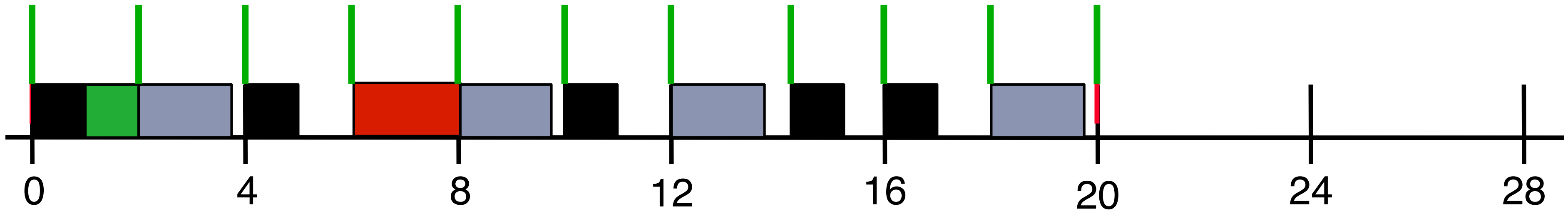
Frame Size f

1. at least one period should be multiple of f
   ensures an integer number of frames per hyperperiod
2. f ≥ max(ei) (avoids preemption)
3. one full frame (two boundaries)
   between release time t′ and deadline D
   for each job in all periods
   to enable policing by the scheduler before deadline

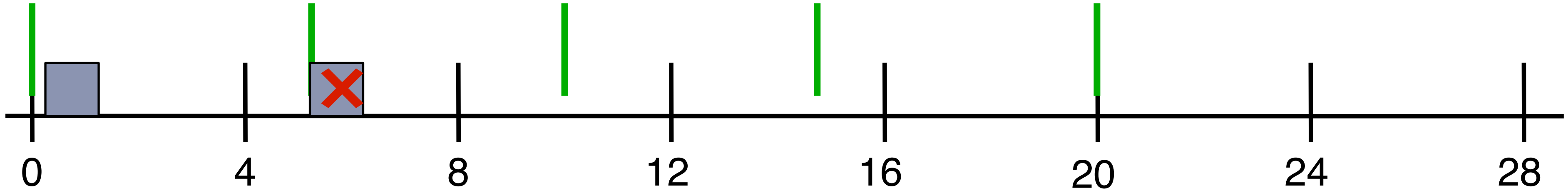


Examples

(4,1) (5,1.8) (20,1) (20,2)



(4,1) (5,2) (20,5)



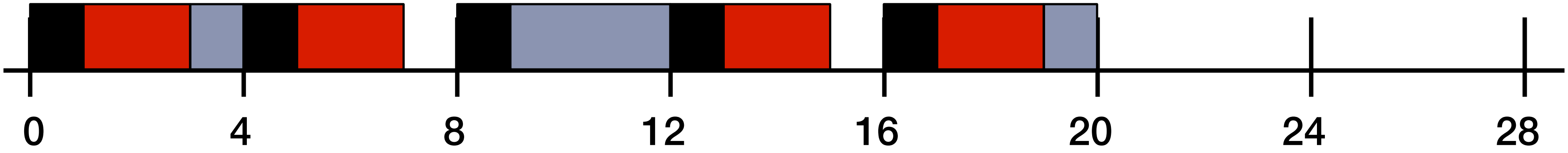
Slices

Decompose jobs in slices

- subroutines, bus scheduling: message segments

Example (4,1) (5,2) (20,5):

- cut (20,5) in (20,1) (20,3) (20,1)
- frame size: 4



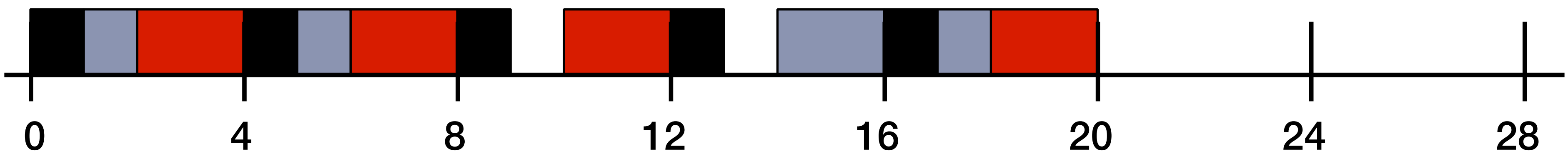
Problems

- If T1 in job 2 does not fully use its wcet, T2 runs early
- If T2 (job 3, in 13,15) overruns, scheduler detects at 16

Alternative

Better solution

- (4,1) (5,2) (20,5)
- cut (20,5) in (20,1) (20,1) (20,2) (20,1)
- frame: 2



A Cyclic Executive

current time t := 0; current frame k := 0;

```
at every f time units DO
   get jobs, slices from cyclic schedule
   t:=t+f; k:= t mod hyperperiod;
   react if last jobs/slices have not completed
   execute jobs
   take care of aperiodic jobs
DONE
```

Accommodating Aperiodic Jobs

- use time not allocated to slices
- objective: improve response time of aperiodic jobs
- slack stealing: execute aperiodic jobs before periodic

Accommodating Sporadic and Aperiodic Jobs

Assumptions

- known deadline, wcet: S(D,e)
- jobs preemptable

Example

- remove defective part from conveyer belt, if possible
- otherwise stop the belt

At Execution Time

- acceptance test: sum(slack times in all frames before d) $\geq$ e
- generate “slices” that fit in frames
- static: put slices in frames
- dynamic: queue according to EDF (after positive acceptance test)

Practicalities

Problems to Consider

- frame overruns
- incomplete test
- transient faults

What to Do

- terminate overrunning job
  (may be OK for robust controllers)
- suspend overrunning job/slice and resume it in next frame where it has allocation
- continue overrunning job into next frame

Mode Changes

- Task system static per operational mode
- Pre-computation of all involved schedules
- Reconfiguration when mode changes
- Cyclic schedule must be exchanged
- Code and data of new tasks must be brought in
- Use old schedule during reconfiguration, then switch
- Hard/Soft mode changes

Critical Sections

Task 0

```
while (true) {
   Work
   lock(l)
      Critical_Section
   unlock(l)
}
```

Task 1

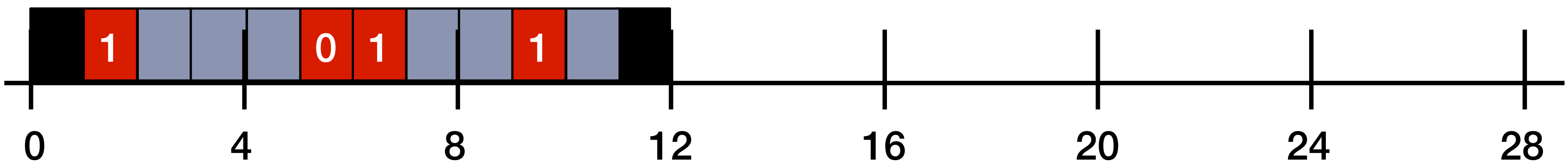
```
while (true) {
   Work
   lock(l)
      Critical_Section
   unlock(l)
}
```

Critical Sections

T_0: (12,1) (12,1) (12,1)

T_1: (4,1) (4,1) (4,1)

Red: critical section



- Split task, schedule critical section as separate slice
- no explicit lock/unlock operations needed
- Complicated in event driven systems (priority inversion)

Additional Topics

- conceptually simple:
  - precedence constraints
  - no concurrency control mechanisms
    e.g. mutexes (no priority inversion problem)
  - known cache interference (context switching)
  - several processors (if global time available)
    not so simple, but feasible
- replica determinism
- reintegration of nodes after faults
- deriving a schedule in the general case is NP-Hard

Motivation for Partitioned Systems

- No interference between subsystems
  - prevents misbehaving subsystems to damage one another
  - no timing anomalies
- Separate, systematic test of subsystems, deterministic behavior
- Prevents some timing covert channels

Space Partitioned Systems

Space Partitioning

allocate each resource to 1 partition

Examples

- disk partitioning
- address spaces (for example Unix processes)
- main memory
- IO devices
- caches
- SMP partitioning

Time Partitioned Systems

Time Partitioning

- divide time into slots
- allocate slot to 1 partition

Examples

- CPU
- busses

Implementation of Time Partitioning

… can be hard, because:

- Interaction of resources
  for example bus DMA and CPU-speed
- Multi-Processor
  all CPUs or partition CPUs?
  Synchronizing all participating CPUs
  Gang scheduling
- External events

Summary

- static, potentially with mode changes
- conceptually simple
- easy to test, validate, certify
- requires fixed inter-release times