Real-Time Systems

Marcus Völp Hermann Härtig

Time-Driven and

Partitioned Systems

14/11/2012

TECHNISCHE UNIVERSITÄT DRESDEN

Time Driven vs. Event Driven Scheduling

Time driven:

- compute feasible schedule at design time
- store schedule in table
- scheduler dispatches tasks at *certain points* in time

Event driven:

- determine feasibility of set of task at design time depending on the scheduling algorithm to be used
- scheduler computes schedule and dispatches task at certain events

Time-Driven and Partitioned Systems

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 (later)

Time-Driven Scheduling

Jane Liu

Properties:

- scheduling decisions are made at specific time instants
- time instants are chosen a priori (before the system begins execution) and stored in table
- off-line computed schedule determines which job to run at which time instants.

Typically "restrictive" assumptions: "deterministic" systems

- fixed number of tasks in systems
- task parameters are fairly well known before the system starts (fixed inter-release times)
- tasks must be ready at their release times

- usually 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

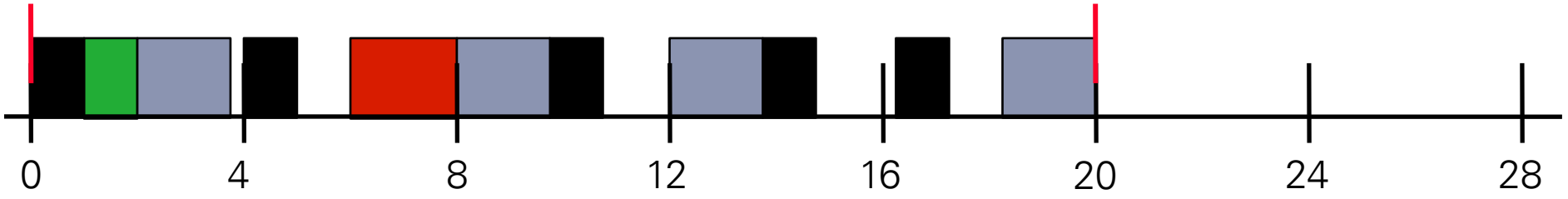
Jane Liu

sufficient to find schedule for hyperperiod ...
which 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

Jane Liu

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!)

in contrast: scheduling actions at events (e.g., blocking, release of job) in priority driven systems

Tick-Driven Systems ("Synchronous" Systems)

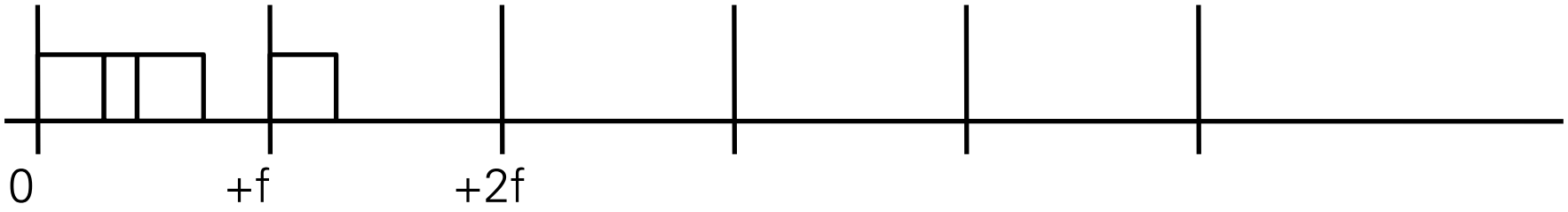
Jane Liu

scheduling actions only at periodic instants of time

no preemption within frames (in the normal case)

check for violations etc. at frame borders

frame size ?



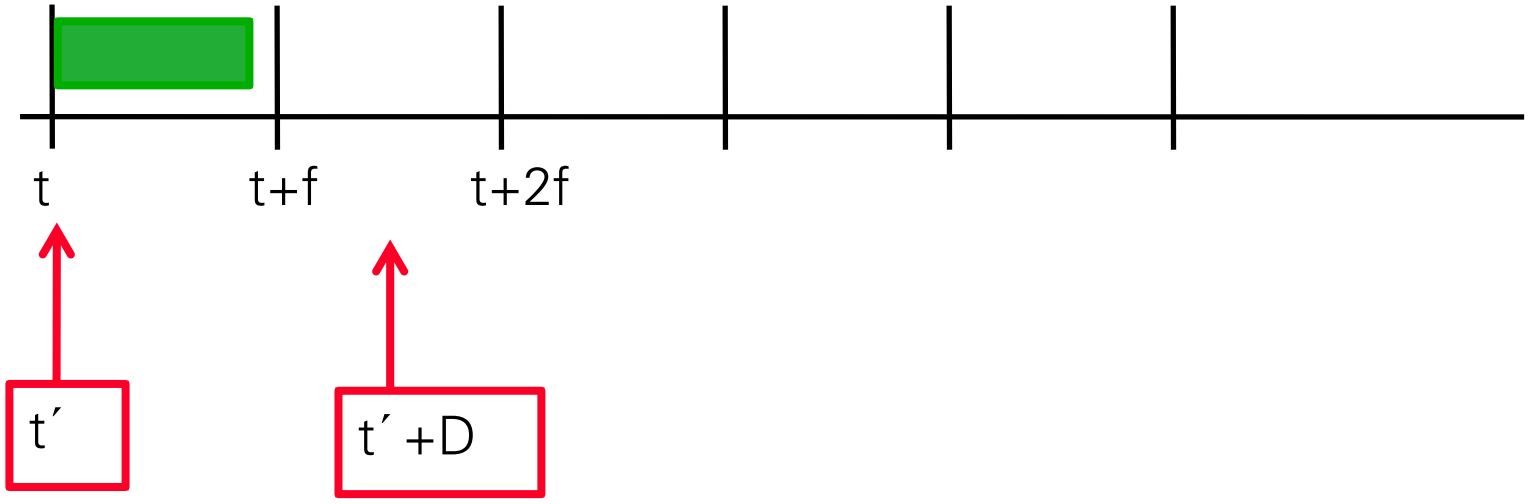
Frame Size f

Jane Liu

1: at least one period should be multiple of f

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
(enables scheduler checks before deadline)



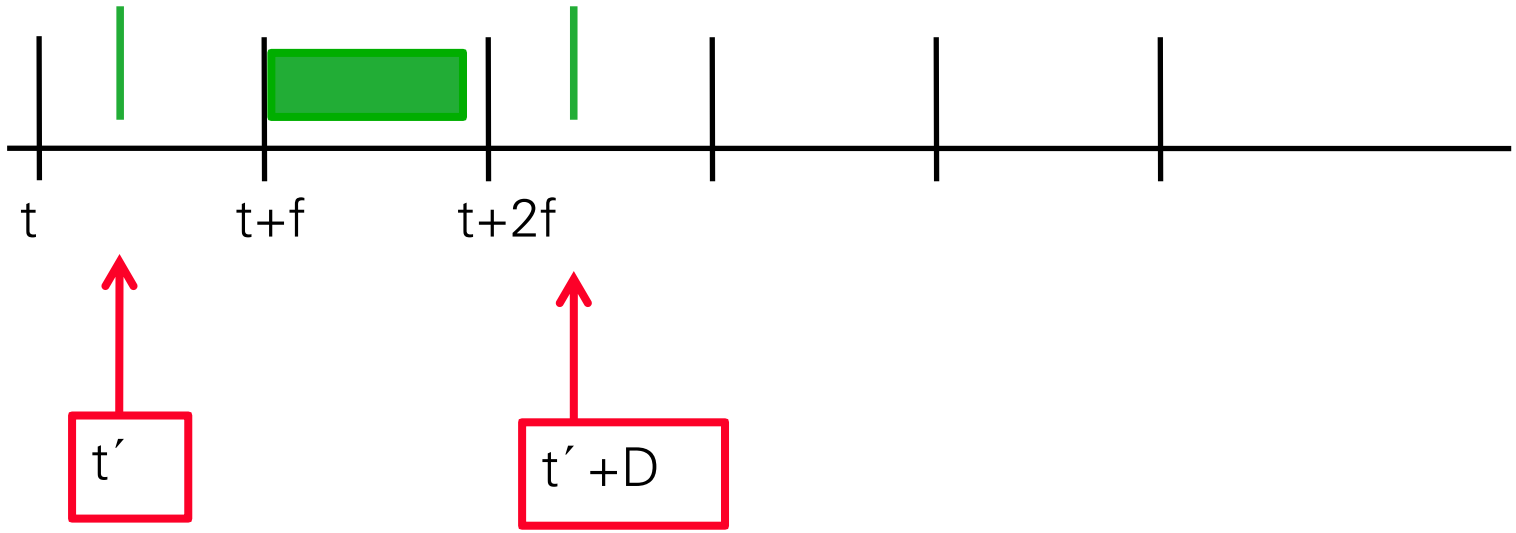
Frame Size f

Jane Liu

1: at least one period should be multiple of f

2: $f \geq \max(e_i)$ (avoids preemption)

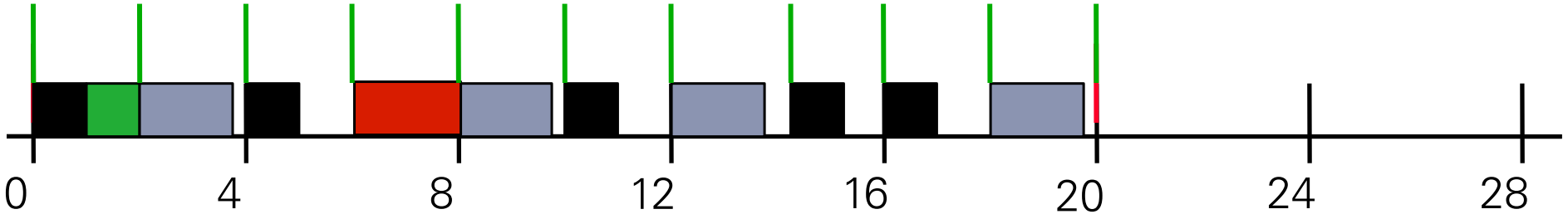
3: one full frame (two boundaries)
between release time (t') and deadline (D)
for each job in all periods
(enables scheduler checks before deadline)
more critical case: $t' > t$, $t + 2f \leq t' + D$



Examples

Jane Liu

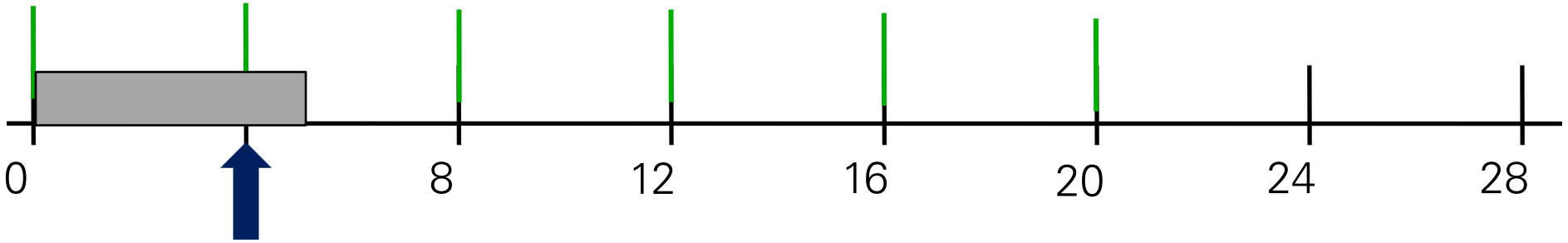
example 1: (4,1) (5,1.8) (20,1) (20,2) f = 2



Examples

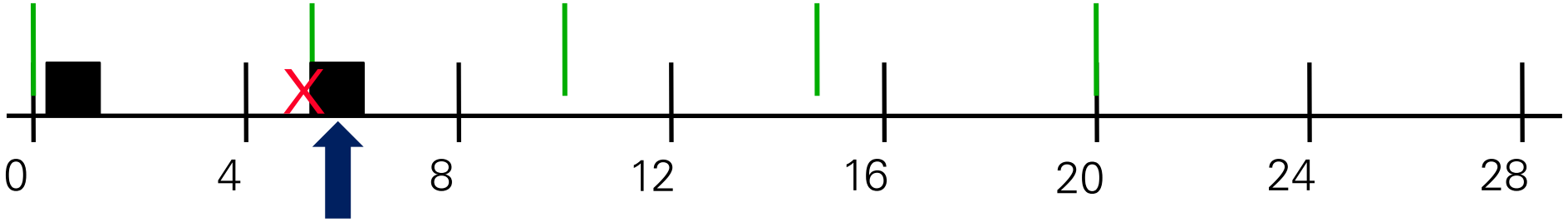
Jane Liu

example 2: (4,1) (5,2) (20,5) f = 4



preemption unavoidable (f < max (e_i))

f = 5



cannot check "black" deadline
(only one boundary between t' and D)

Slices

Jane Liu

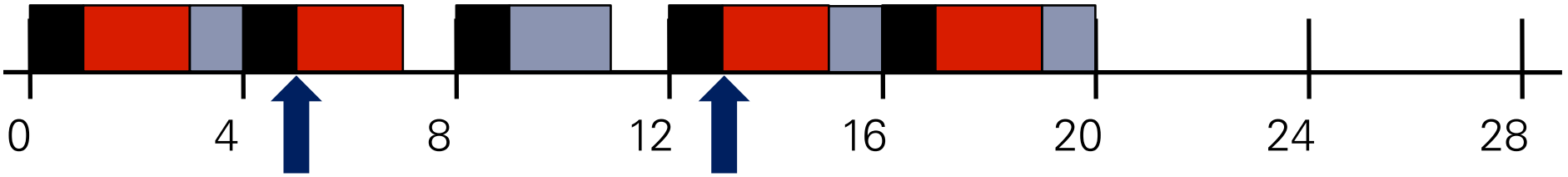
decompose jobs in slices:

- subroutines
- cut messages into segments ...

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

frame size: 4

cut (20,5) in (20,1) (20,-) (20,2) (20,1) (20,1)



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

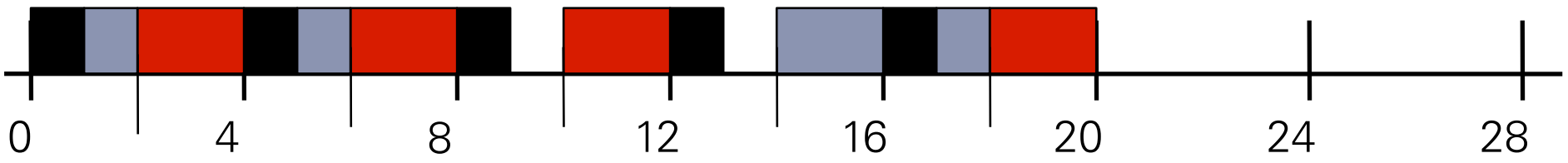
Jane Liu

better:

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

frame size: 2

cut (20,5) in (20,1) (20,1) (20,-) (20,2) (20,1)



A Cyclic Executive

Jane Liu

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;

if last jobs, slices have not completed properly,
do something

execute things ...

take care about aperiodic jobs

DONE

Accommodating Aperiodic Jobs

Jane Liu

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

Jane Liu

Assumptions:

known deadline and WCET: S(D,e); jobs are 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) >= e

generate "slices" that fit in frames

static: put slices in frames

dynamic: queue after successful acceptance test
according to event driven schedule (e.g., EDF)

Practicalities

Jane Liu

- 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

schedule different set of jobs in different “operation mode”

examples:

aircraft control: taxi, start, fly, land, ...

mobile phone: stand by, speak, video, ...

task system per “operational mode” is static

precomputation of all involved schedules

reconfiguration when mode changes:

- bring in code and data of new task
- exchange cyclic schedule hard / soft deadline for mode change (use old schedule during reconfiguration)

Critical Sections

- **Task 0**

```
Do {
  Work
  lock(l)
    Critical section
  Unlock(l)
} forever
```

- **Task 1**

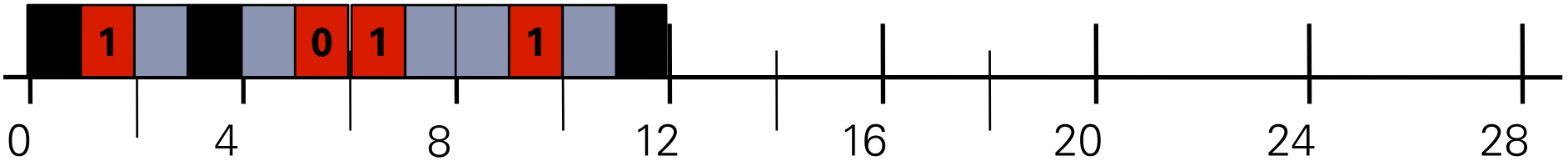
```
Do {
  Work
  lock(l)
    Critical section
  Unlock(l)
} forever
```

Critical Sections(2)

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 (→ Blackboard)
  - no need for concurrency control mechanisms
    e.g. mutexes (no priority inversion problem)
  - known cache interference (context switching)
  - several processors
    (need global time and small clock drifts for precedence)
- replica determinism
- reintegration of nodes after faults
- deriving a schedule in the general case is NP-Hard

Time-Driven and Partitioned Systems

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 (later)

Motivation for Partitioned Systems

no interference between subsystems !

- prevents misbehaving subsystems to damage other
- no timing anomalies

deterministic behavior

test and validate subsystems in isolation; run as tested

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 (→ later lecture)
- SMP partitioning (→ later lecture)

Time Partitioned Systems

time partitioning:

divide time into slots

allocate slot to 1 partition

examples

- CPU
- busses
- mobile communication

Implementation of time partitioning

… is hard, because:

- interaction of resources

  for example bus DMA and CPU-speed (→ lecture HW-RT)

- multi-processor

  all CPUs, some CPUs, or partition CPUs ?

  synchronizing all participating CPUs

  "gang scheduling"

- external events

Time Driven Communication

- divide network-time into slots
- allocate slots to communication partners
- if sparse time is used, each message can be identified by its (sparse) time stamp
- detecting a missing message becomes simple
- example: TT-Ethernet (→ Paper Reading)

→ more in later lecture on real communication

Forward pointers

later in this course:

- time-driven communication → TT-Ethernet
- a high-level language for tick-driven systems → Esterel
- cache partitioning
- partitioning operating systems

Summary

- static …, except mode changes
- conceptually simple
- easy to test, validate, certify.

- but: fixed inter-release times
- next: event driven systems