Time-Driven Systems

(following Jane Liu, Real-Time Systems)

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

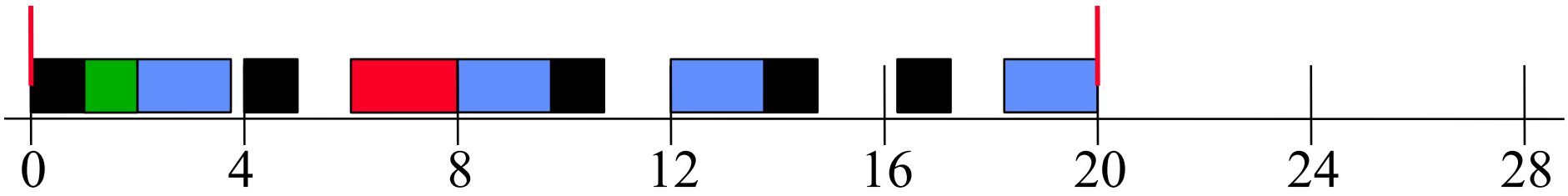
- fixed number of tasks in systems
- with a priori fairly well known parameters (fixed inter-release times)
- tasks must be ready at their release times
- usually used for safety-critical, hard real-time systems !!!

Deriving a Schedule

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 Schedule

store all decision 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"

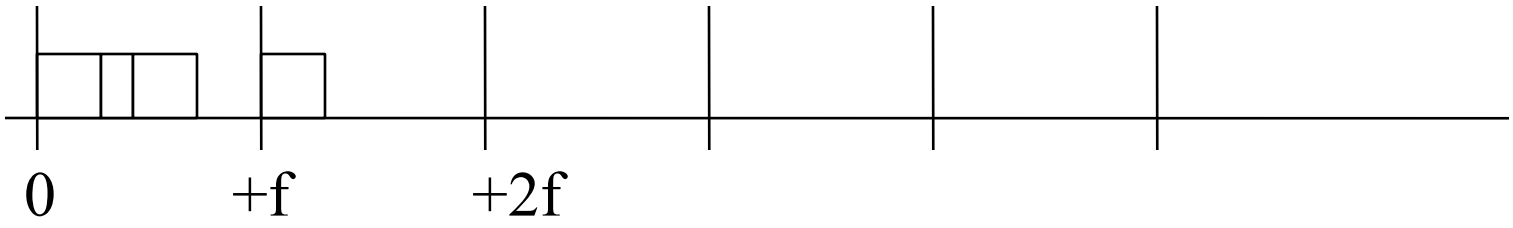
note: scheduling decisions at instants in time (not events!)

contrast:

  priority driven systems scheduling decisions at events

Frames

Scheduling decisions only at periodic instants of time
no preemption within frames (in the normal case)
check for violations etc. at frame borders
frame size ?

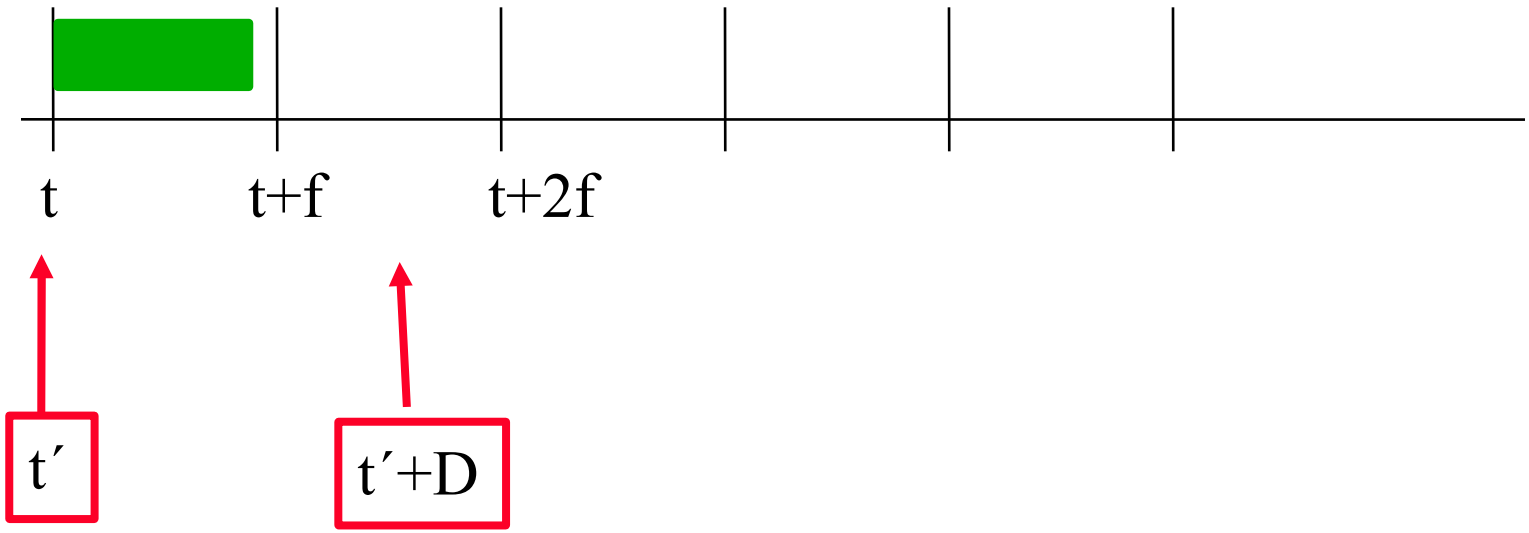


Frame Size f

1: At least one period should be multiple of f

2: f □ 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 the scheduler checks before deadline

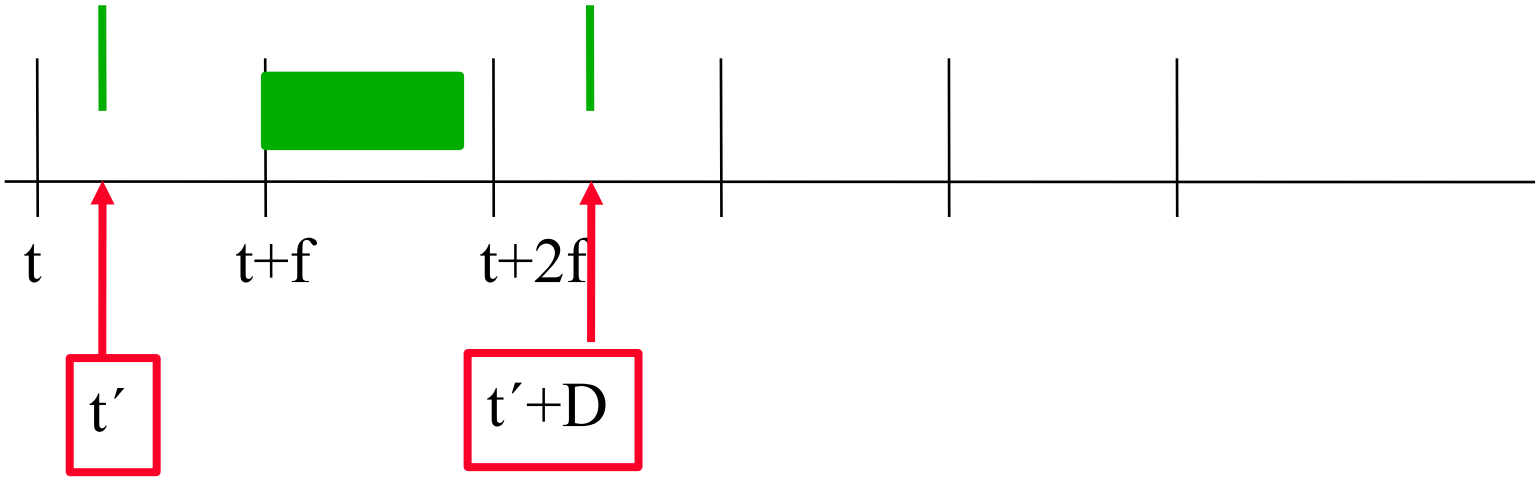


Frame Size f

1: At least one period should be multiple of f

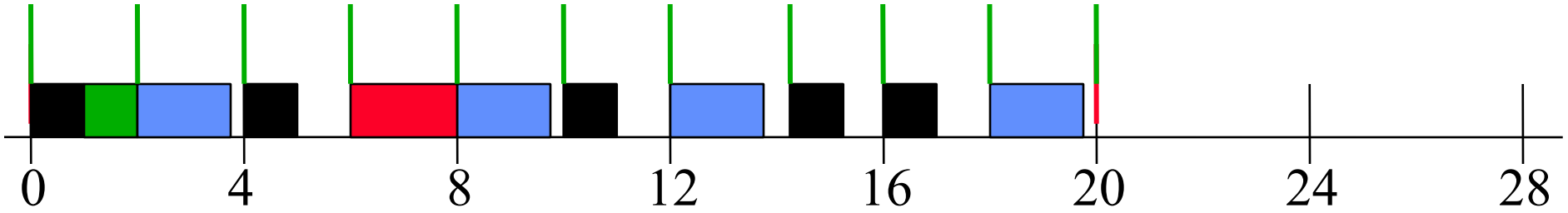
2: f □ max(e_i) (avoids preemption)

3: one full frame(two boundaries)
between release time(t´) and deadline(D)
for each job in all periods
to enable the scheduler checks before deadline
more critical case: t´ > t: t + 2f □ t´ + D

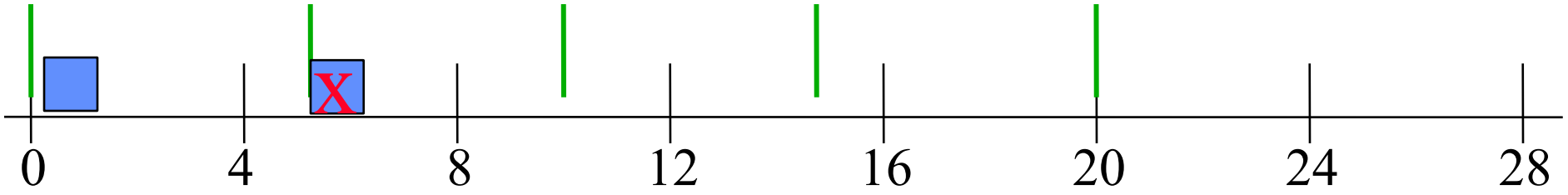


Examples

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



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



Slices

decompose jobs in slices

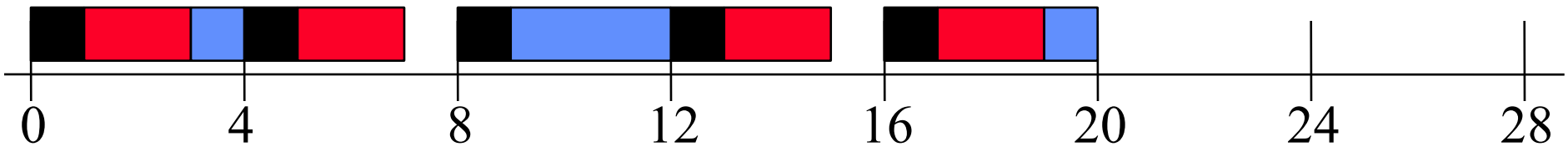
subroutines

cut messages into segments ...

(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

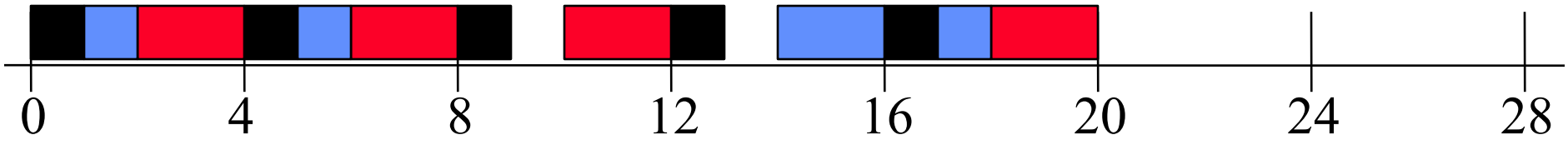
Alternative

better:

(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;
    if last jobs, slices have not completed properly,
      do something
    execute things ...

    Take care about 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: deadline, wcet: S(d,e) known,
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) >= e

Generate "slices" that fit in frames

Static: put slices in frames

Dynamic: queue according to EDF (after succ. acceptance test!)

Practicalities

Frame overruns ...

incomplete test ...

transient faults ...

what to do:

terminate overruning job
(may be ok for robust controllers)

suspend overunning 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”.
example: aircraft control: taxi, start, fly, land
Precomputation of all envolved 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 ...

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 (later in this lecture):

communication on shared bus
  time division multiple access (TDMA)

replica determinism

reintegration of nodes after faults

deriving a schedule in the general case is NP-Hard

Important Variant: Time Partitioned Systems

Summary

Static ... Mode changes
conceptually simple
easy to test, validate, certify.

Fixed inter-release times