Introduction

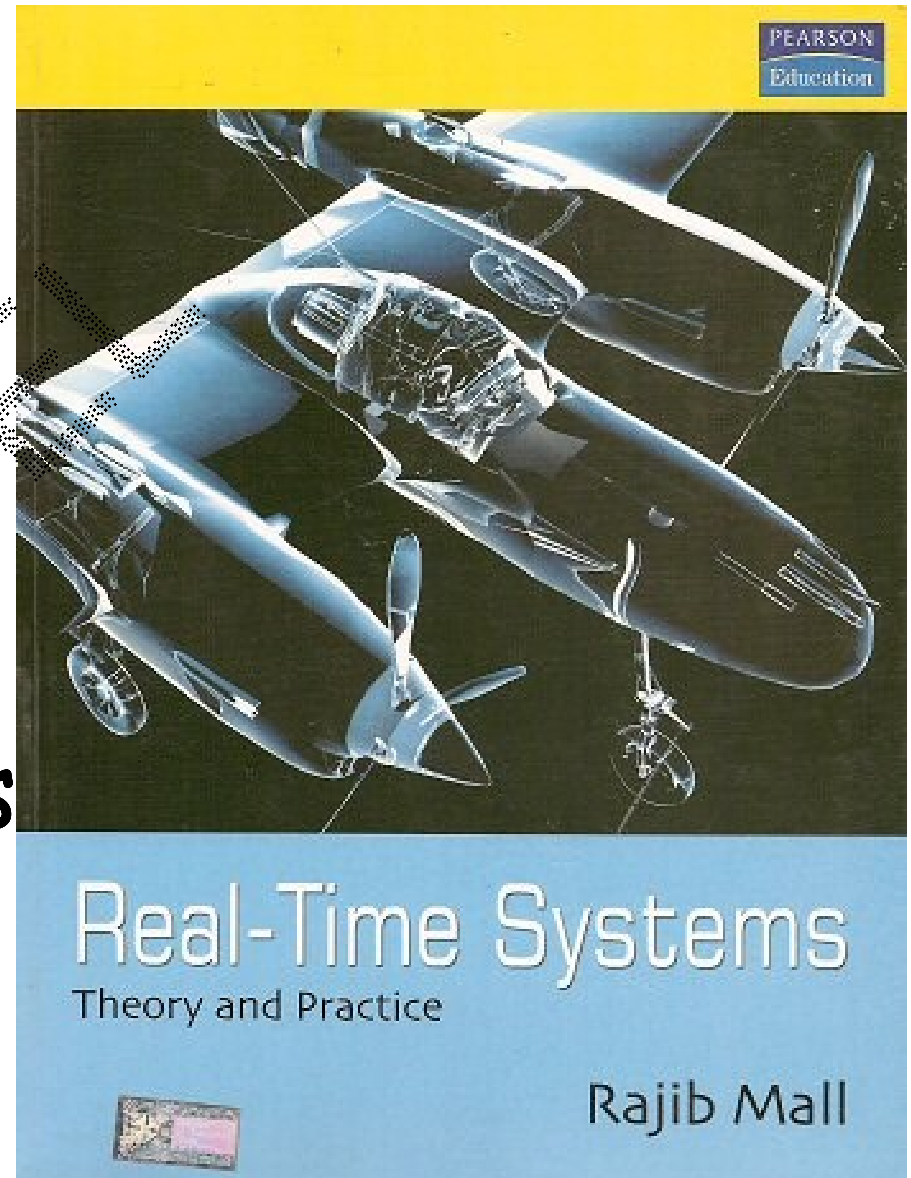
Dr. RAJIB MALL
Professor
Department Of Computer Science & Engineering
IIT Kharagpur.

Course Plan

- Introduction
- Basics of task scheduling
  - Cyclic executives
  - RMA and EDF
- Resource Sharing
- Scheduling on Multiprocessors
- POSIX-RT
- Commercial RTOS

Text Book

- R. Mall, Real-Time Systems, Pearson, 2008.
- To be supplemented with other materials



Reference Books

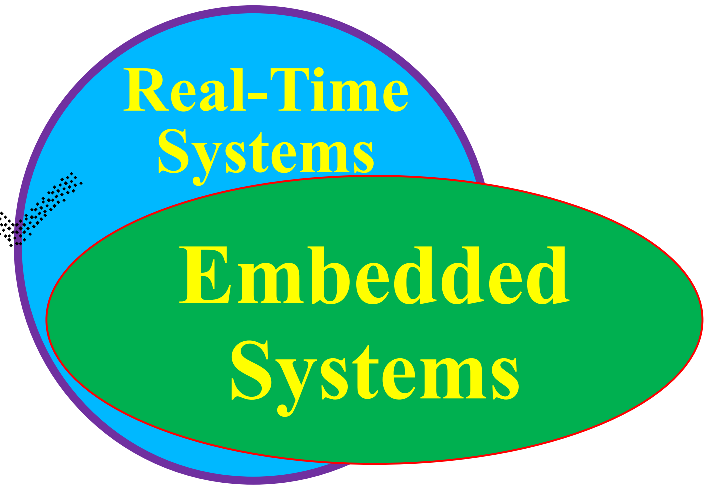
- Jane Liu, Real-Time Systems, Pearson, 2000
- C. Krishna and K. Shin, Real-Time Systems, McGraw-Hill, 2000

What is Real-Time?

- Real-time is a quantitative notion of time measured using a physical clock.
  - **Example:** After a certain event occurs (temperature exceeds 500 degrees) the corresponding action (coolant shower) must complete within 100mSec.
- This is in contrast to the qualitative notion of time:
  - Expressed using notions such as **before, after, sometime, eventually**, etc.

Real-Time Systems

- Characterized by time-constrained response to events.
- Often are:
  - Embedded
  - Safety-critical
- RTOS helps applications to meet their deadlines:
  - **Task scheduling is the primary mechanism for making applications meet their respective deadlines.**

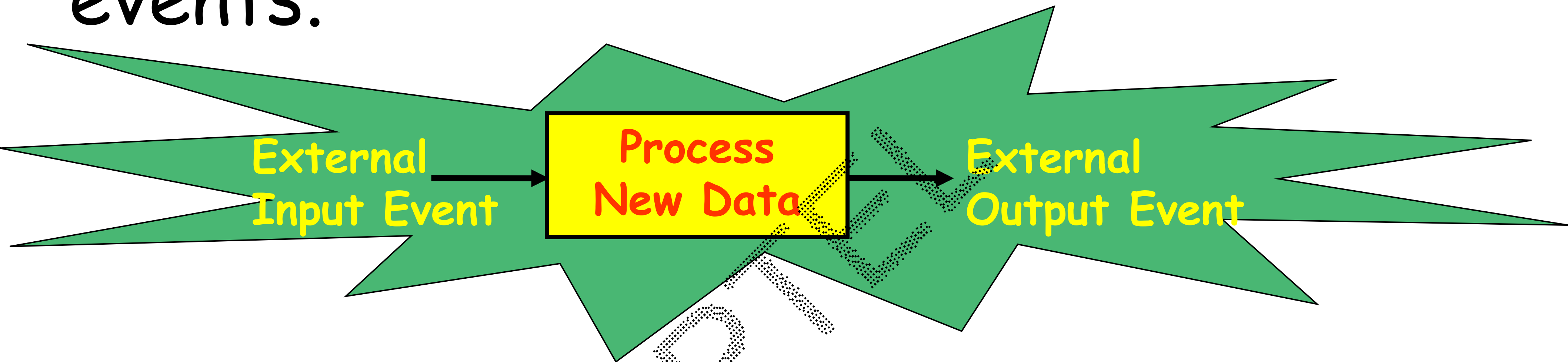


Why Surge in Embedded Applications?

- Trend of reducing cost of computers:
  - Processors
  - Memory
- Flexibility due to Internet
- Reducing power consumption
- Reducing size
- Increasing Processing power and hardware and software reliability

Important Characteristics

- An embedded system responds to events.



Example: An Automobile airbag system.

When the airbag's motion sensors detect a collision, the system needs to respond by deploying the airbag within 10ms or less.

- **or the system fails!**

How are RTOS Different from other OS?

- An embedded system responds to external inputs:
  - If response is late, the system fails.
- General purpose OS:
  - Minimizes response time and ensures fairness
- Real-time OS:
  - **Helps tasks meet their deadline.**

Characteristics of A Real-Time System

- **Real-time:**
  - At least tasks have real-time constraints, e.g a Deadline.
- **Correctness Criterion:**
  - Results should be logically correct,
  - **And also within the stipulated time.**

Why Have an OS in an Embedded Device?

- Support for:
  - Multitasking, scheduling, and synchronization
  - Timing aspects.
  - Memory management
  - File systems
  - Networking
  - Graphics displays
  - Interfacing wide range of I/O devices
  - Scheduling and buffering of I/O operations
  - Security and power Management

Embedded OS

- **Example:** *A smartphone operating system contains over five million lines of code!*
- Projects will hardly have the time and funding:
  - To develop all of this code on their own!
- Typical Embedded OS license fees are less than even Rs100 per device --- lower than a desktop OS
- Some very simple low-end devices might not need an OS:
  - But devices are getting more complex.

Types of Real-Time Systems

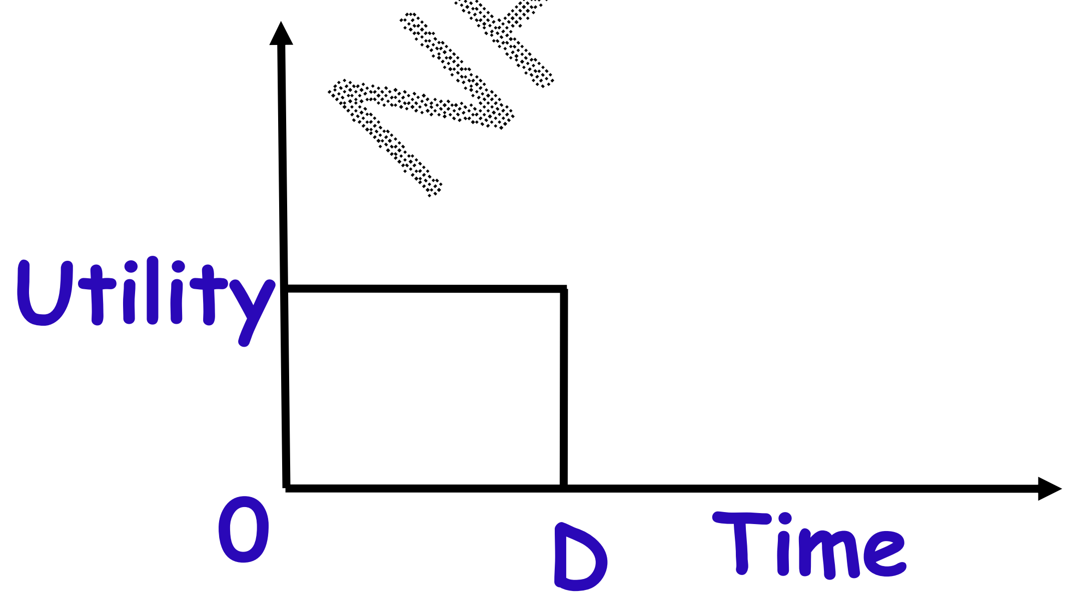
- Real-time systems are different from traditional systems:
  - Tasks have deadlines associated with them.
- Classified largely based on the consequence of not meeting deadline:
  - Hard real-time systems
  - Soft real-time systems
  - Firm real-time systems

Hard Real-Time Systems

- **If a deadline is not met:**
  - **The system is said to have failed.**
- The task deadlines are of the order of micro or milliseconds.
- Many hard real-time systems are safety-critical.
- **Examples:**
  - Industrial control applications
  - On-board computers
  - Robots

Firm Real-Time Systems

- If a deadline is missed occasionally, the system does not fail:
  - The results produced by a task after the deadline are ignored.

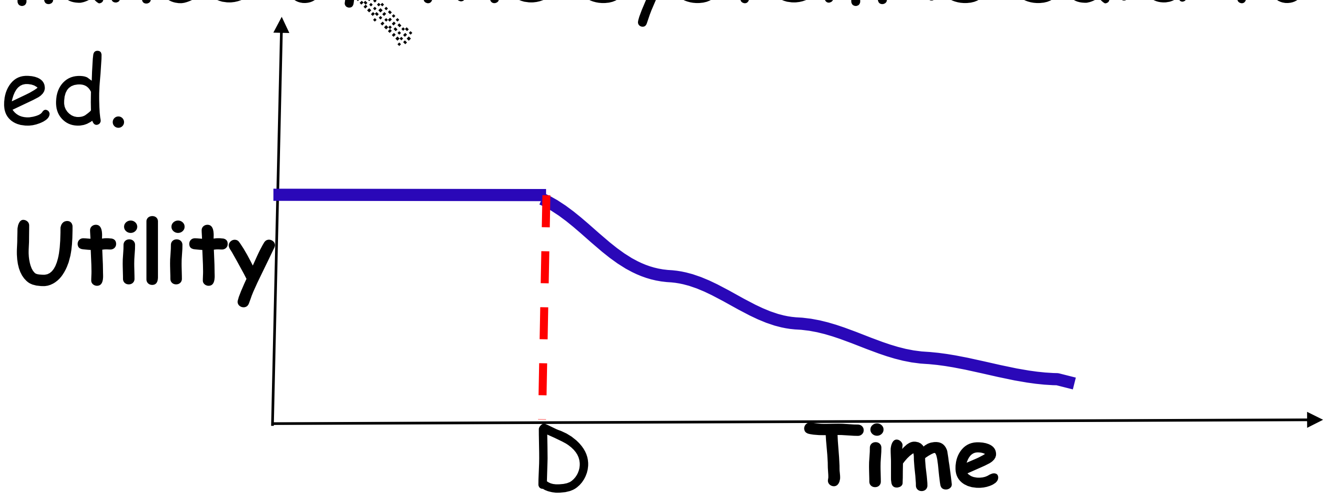


Firm Real-Time Systems

- Examples:
  - A video conferencing application
  - A telemetry application
  - Satellite-based surveillance applications

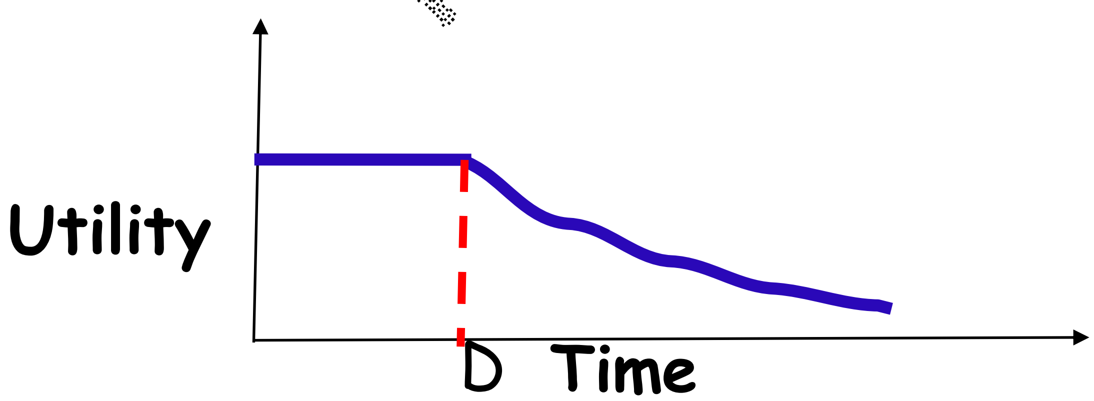
Soft Real-Time Systems

- If a deadline is missed, the system does not fail:
  - The utility of a result decreases with time after the deadline.
  - If several tasks miss deadline, then the performance of the system is said to have degraded.



Soft Real-Time Systems

- Use probabilistic requirements on deadline.
- For example, 99% of time deadlines will be met.



Soft Real-Time Systems

- Examples:
  - Railway reservation system
  - Web browsing
  - In fact, all interactive applications

Types of Tasks

- **Periodic:**
  - Recur according to a timer
  - A vast majority all real-time tasks are periodic
- **Aperiodic:**
  - Recur randomly and are soft real-time tasks
- **Sporadic:**
  - Recur randomly, but hard real-time tasks

Timing Constraints

- A timing constraint:
  - Defined with respect to some event.
- An event:
  - Occurs at an instant of time
  - Generated either by the system or its environment

Real-Time Tasks

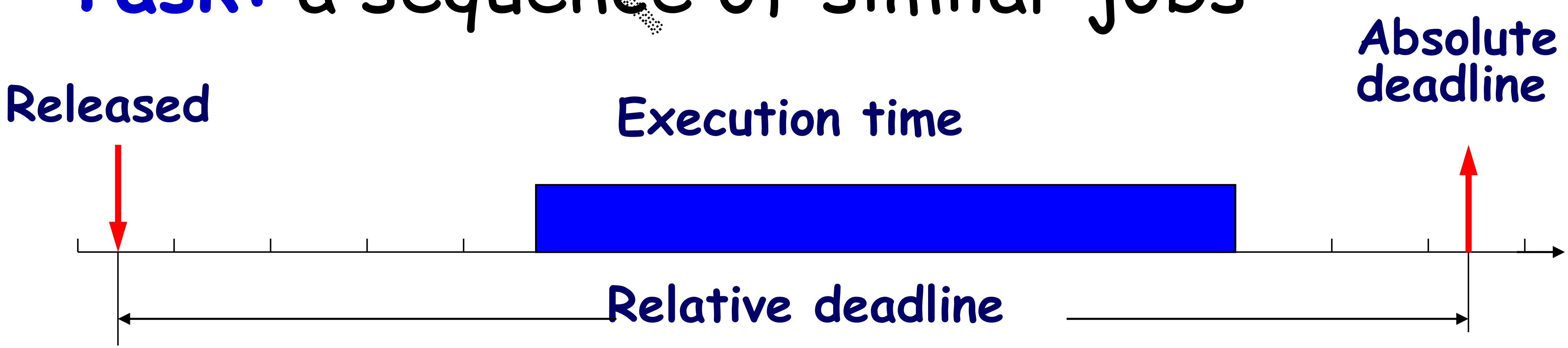
- Real-time tasks get generated due to certain event occurrences:
  - Either internal or external events.
- Example:
  - A task may get generated due to a temperature sensor sensing high-level.
- When a task gets generated:
  - It is said to be released or arrived .

Real-Time Task Scheduling

- Essentially refers to the order in which the various tasks are to be executed.
- It is the primary means adopted by an operating system to meet task deadlines.
- Obviously, scheduler is a very important component of every RTOS.

Real-Time Workload

- **Job:**
  - A unit of work
  - A computation, a file read, a message transmission, etc
  - **A task instance**
- **Task:** a sequence of similar jobs

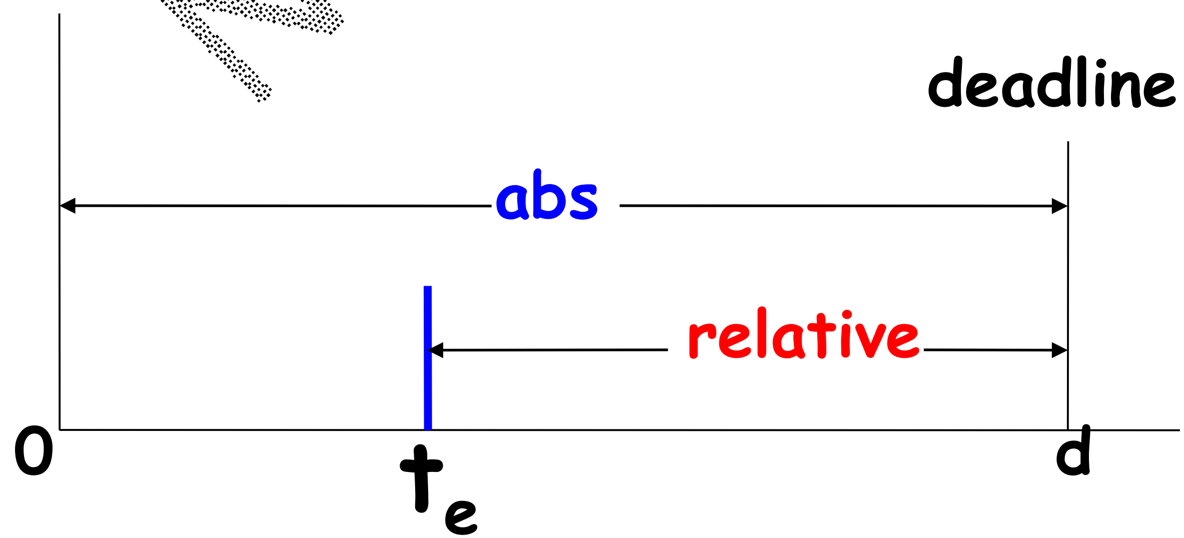


Task Instance (Job)

- A task typically recurs a large number of times:
  - Each time triggered by an event
  - Each time a task recurs, an instance of the task is said to have been generated or released.
- The ith time a task T recurs:
  - Job or Task instance Ti is said to have arrived.

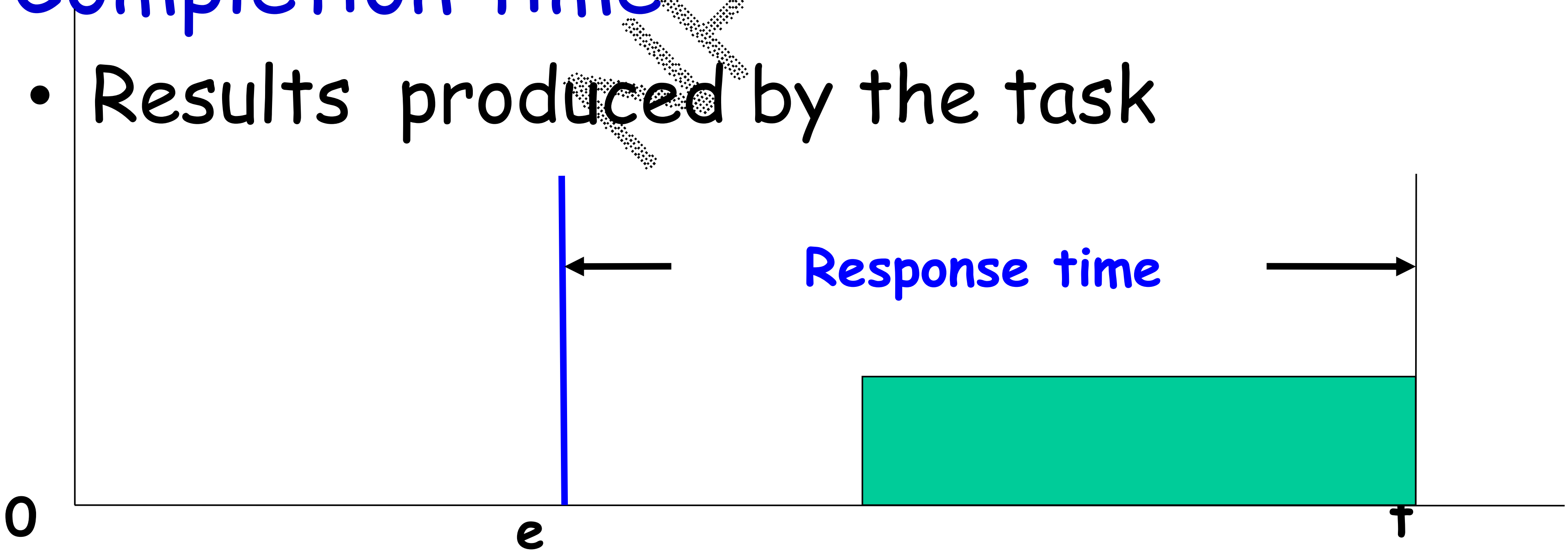
Relative and Absolute Deadlines

- **Absolute deadline:**
  - Counted from time 0.
- **Relative deadline:**
  - Counted from time of occurrence of task.



Response Time

- Duration between task release time and task completion time.
- Release time
  - The time of occurrence of the event generating the task.
- Completion time
  - Results produced by the task

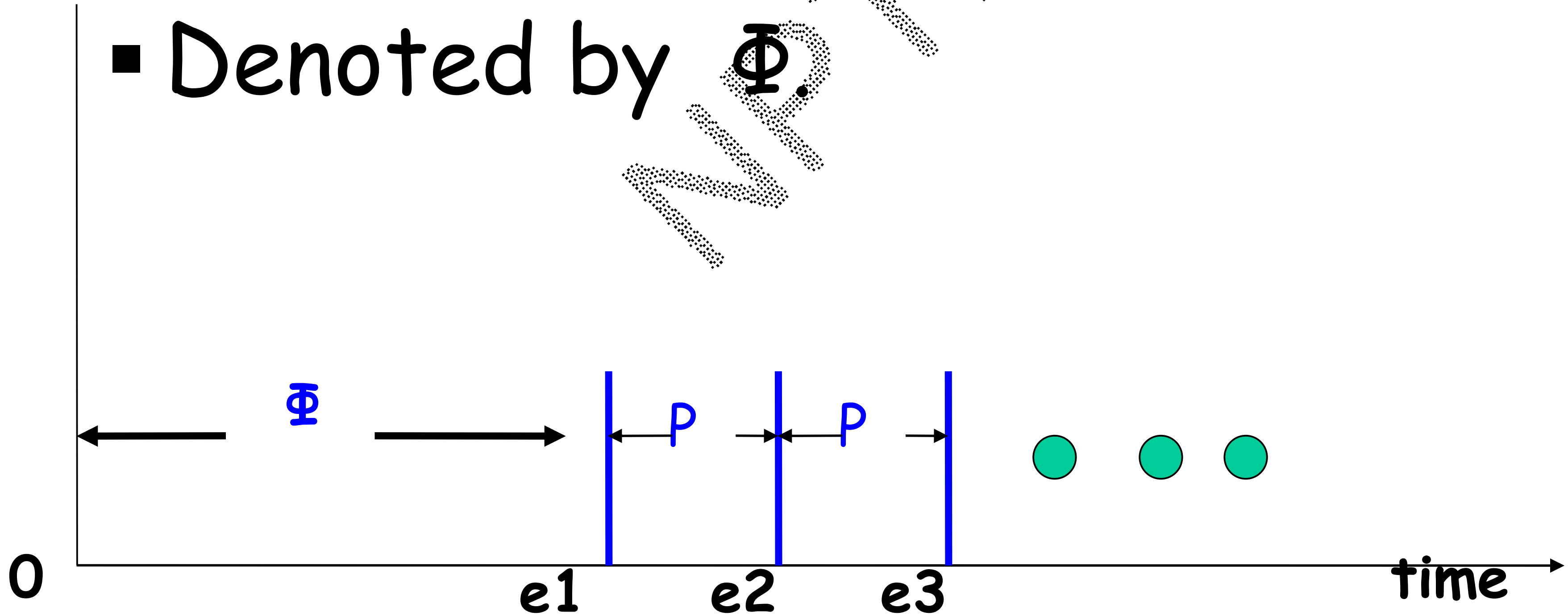


Response Time

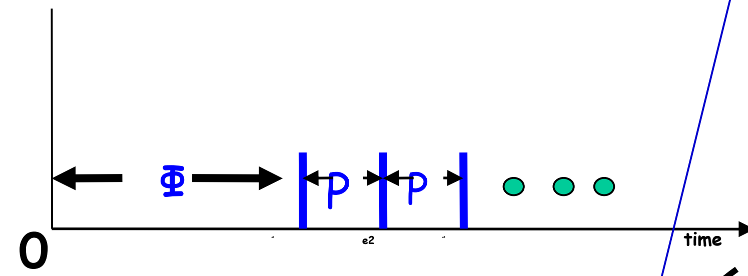
- For soft real-time tasks:
  - The response time needs to be minimized.
- For hard real-time tasks:
  - As long as the task completes within its deadline,
    - No advantage of completing it any early.

Phase of a Periodic Task

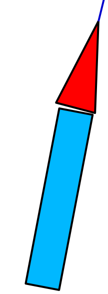
- Phase for a periodic task:
  - The time from 0 till the occurrence of the first instance of the task.
  - Denoted by Φ.



Phase Example



- The track correction task starts 2000 mSecs after the launch of the rocket:
  - Periodically recurs every 50 milli Seconds then on.
  - Each instance of the task requires a processing time of 8 mSecs and its relative deadline is 50 mSecs.



Few Task Scheduling Terminologies

- **Valid Schedule:**
  - At most one task is assigned to a processor at a time.
  - No task is scheduled before it is ready.
  - Precedence and resource constraints of all tasks are satisfied.
- **Feasible Schedule:**
  - Valid schedule is one in which all tasks meet their respective time constraints.

Scheduling Terminologies

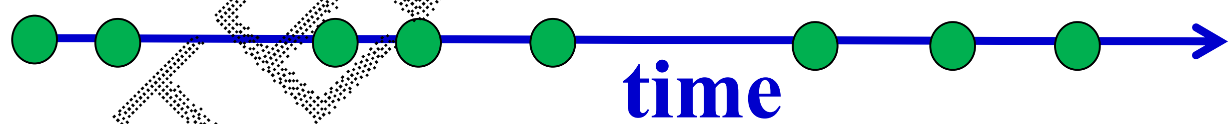
- **Proficient Scheduler:**
  - A scheduler S1 is more proficient compared to another Scheduler S2:
    - If whichever tasks that S2 can feasibly schedule so can S1, but not vice versa.
- **Equally proficient schedulers:**
  - If a task set scheduled by one can also be scheduled by the other and vice versa.

Scheduling Terminologies

- Optimal Scheduler:
  - An optimal scheduler can feasibly schedule any task set that can be scheduled by any other scheduler.

Scheduling Points

- **At these points on time line:**
  - Scheduler makes decision regarding which task to be run next.
- **Clock-driven:**



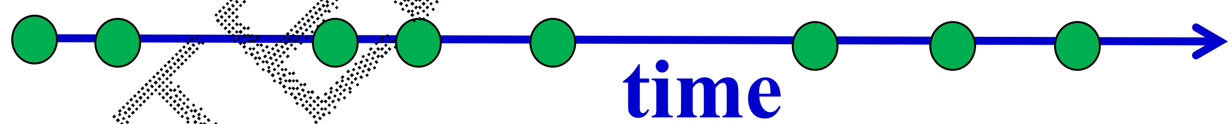
  - Scheduling points are defined by interrupts from a periodic timer.
- **Event-driven:**
  - Scheduling points defined by task completion and generation events.

Real-Time Task Scheduling

- Significant amount of research has been carried out to develop schedulers for real-time tasks:
  - Schedulers for uniprocessors
  - Schedulers for multiprocessors and distributed systems.

Scheduling Points

- **At these points on time line:**
  - Scheduler makes decision regarding which task to be run next.
- **Clock-driven:**



  - Scheduling points are defined by interrupts from a periodic timer.
- **Event-driven:**
  - Scheduling points defined by task completion and generation events.

Real-Time Task Scheduling

- Significant amount of research has been carried out to develop schedulers for real-time tasks:
  - Schedulers for uniprocessors
  - Schedulers for multiprocessors and distributed systems.

Task Scheduling on Uniprocessors

- Focus of much research during the 1970s and 80s.
- Real-time task schedulers can be broadly classified into:
  - Clock-driven
  - Event-driven

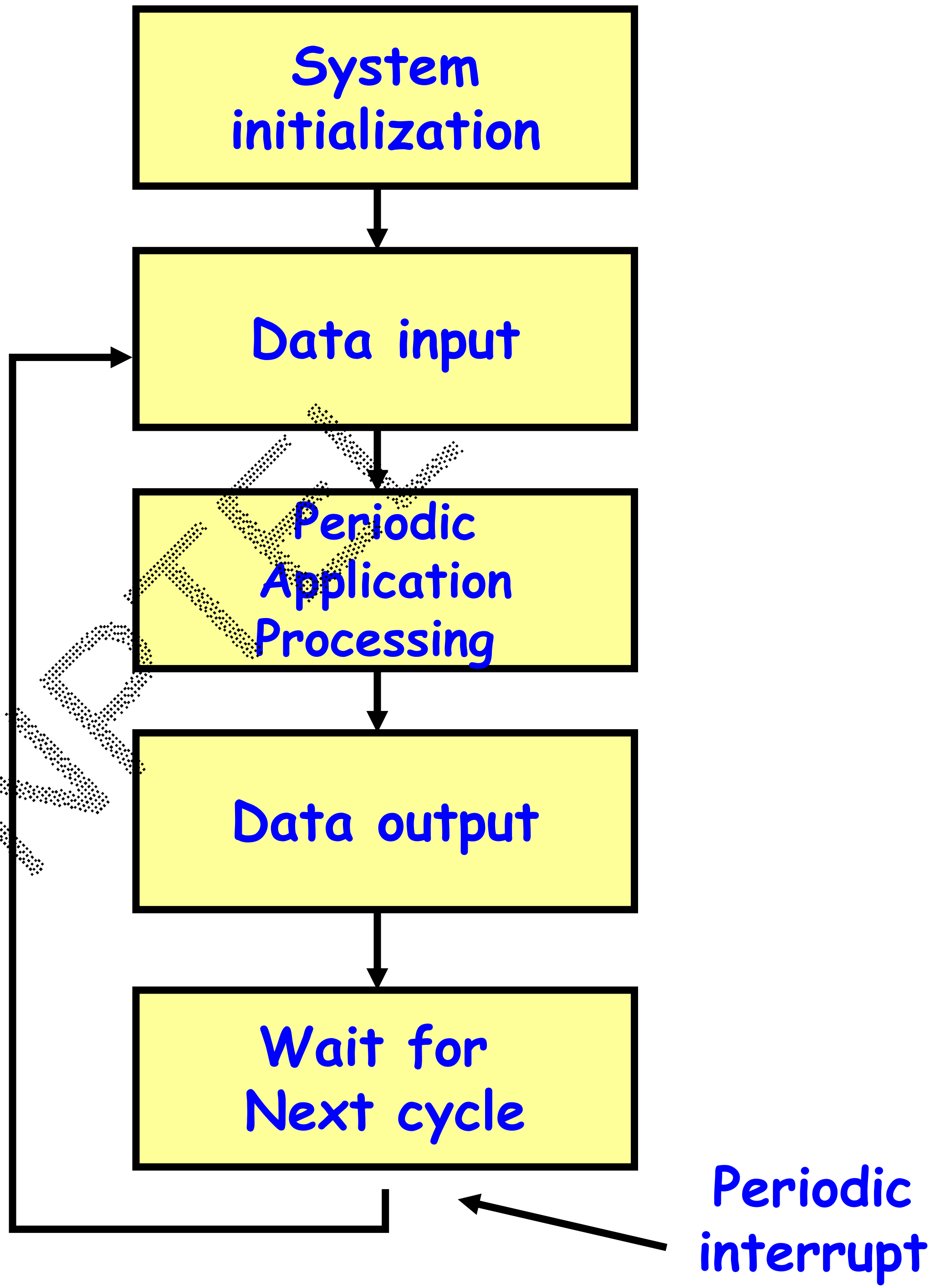
Summary of Schedulers

• Endless Loop	• No Tasks, Polled
• Simple Cyclic Executive	• Single frequency
• Multi-rate Cyclic Executive	• Multiple frequencies
• Priority-based Preemptive Scheduler	• Interrupt driven

•

Cyclic Executives

Simple Cyclic Executive

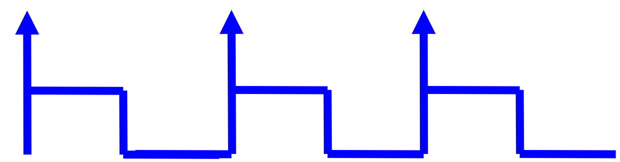


Cyclic Executive

- No actual processes exist:
  - Each cycle only involves a sequence of procedure calls.
- The procedures share a common address space and share data.
  - This data need not be protected (via a semaphore, for example) because "processes" are not preempted.
- All "process" periods are multiples of the cycle time.

Clock-Driven Scheduling: Basics

- Decision regarding which job to run next is made only at clock interrupt instants:



  - Interval timers determine the scheduling points.

- Which task to be run when and for how long is stored in a table.

Clock-Driven Scheduling

- Round robin scheduling:
- Popularly used:
  - Basic Timer-Driven Scheduler (Table-driven)
  - Cyclic Scheduler

Clock-Driven Schedulers

- Also called:
  - Offline schedulers
  - Static schedulers
- Used extensively in embedded applications:
  - Table driven schedulers
  - Cyclic schedulers

Clock-Driven Scheduling

- Used in low cost applications:
  - **Pro:**
    - Compact: Require very little storage space
    - Efficient: Incur very little runtime overhead.
  - **Con:**
    - Inflexible: Very difficult to accommodate aperiodic or sporadic tasks.
- The simplest is table-driven scheduler

Basic Table-Driven Scheduler

(ScheduleTable)

T1	$t(T_1)$
..	...
Tn	$t(T_n)$

```
const int SchedTableSize= 10;
timer_handler () {
  int next_time; task current;
  current  = SchedTable[entry].tsk;
  entry = (entry+1) % SchedTableSize;
  next_time = Table[entry].time + gettime();
 set_timer(next_time);
  execute_task(current);
  return;
}
```

Schedule Table

Task	Time
T1	90
T2	120
T3	75
T4	225
T5	50

Disadvantage of Table-Driven Schedulers

- When the number of tasks are large:
  - Requires setting the timer large number of times.
  - The overhead is significant:
    - Remember that a task instance runs only for a few milli or microseconds.

Cyclic Schedulers

- Cyclic schedulers are very popular:
  - Extensively being used.
- Many tiny embedded applications have severe constraints on memory and processing power:
  - Cannot even host a microkernel RTOS
  - Use cyclic schedulers.

Cyclic Schedulers

- For scheduling n periodic tasks:
  - The schedule is stored in a table.
    - Repeated forever.
  - The designer needs to develop a schedule for what period?
    - **LCM(P1,P2,...,Pn)**

Cyclic Schedulers

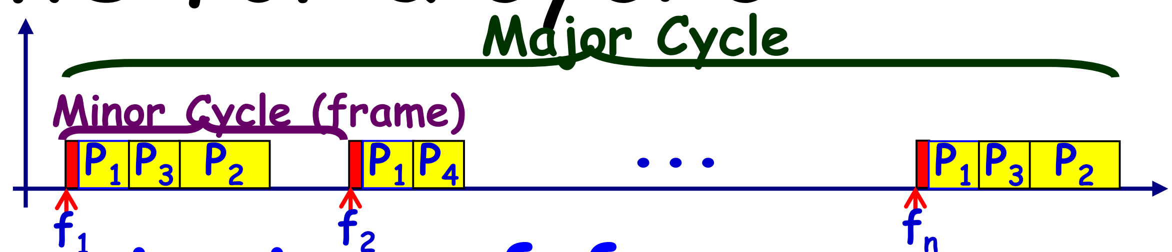
- Cyclic schedulers are very popular:
  - Extensively being used.
- Many tiny embedded applications have severe constraints on memory and processing power:
  - Cannot even host a microkernel RTOS
  - Use cyclic schedulers.

Cyclic Schedulers

- For scheduling n periodic tasks:
  - The schedule is stored in a table.
    - Repeated forever.
  - The designer needs to develop a schedule for what period?
    - **LCM(P1,P2,...,Pn)**

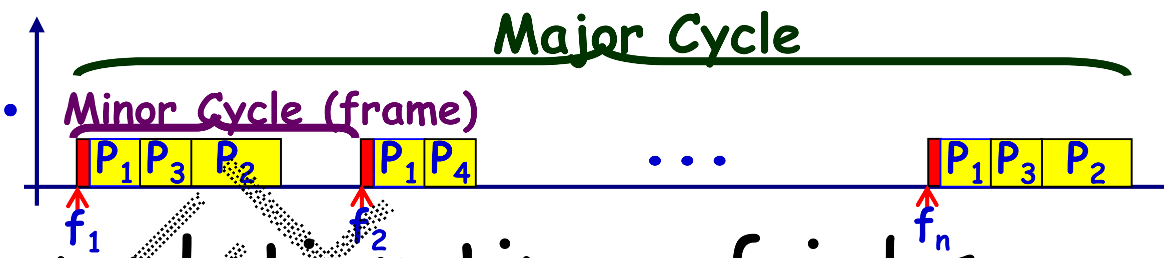
Cyclic Schedulers

- The schedule is Precomputed and stored for one major cycle:
  - This schedule is repeated.
- A major cycle is divided into:
  - One or more minor cycles (frames).
- Scheduling points for a cyclic scheduler:
  - Occur at the beginning of frames.



Cyclic Scheduler Basics

- Scheduling decisions are only made at frame boundaries.



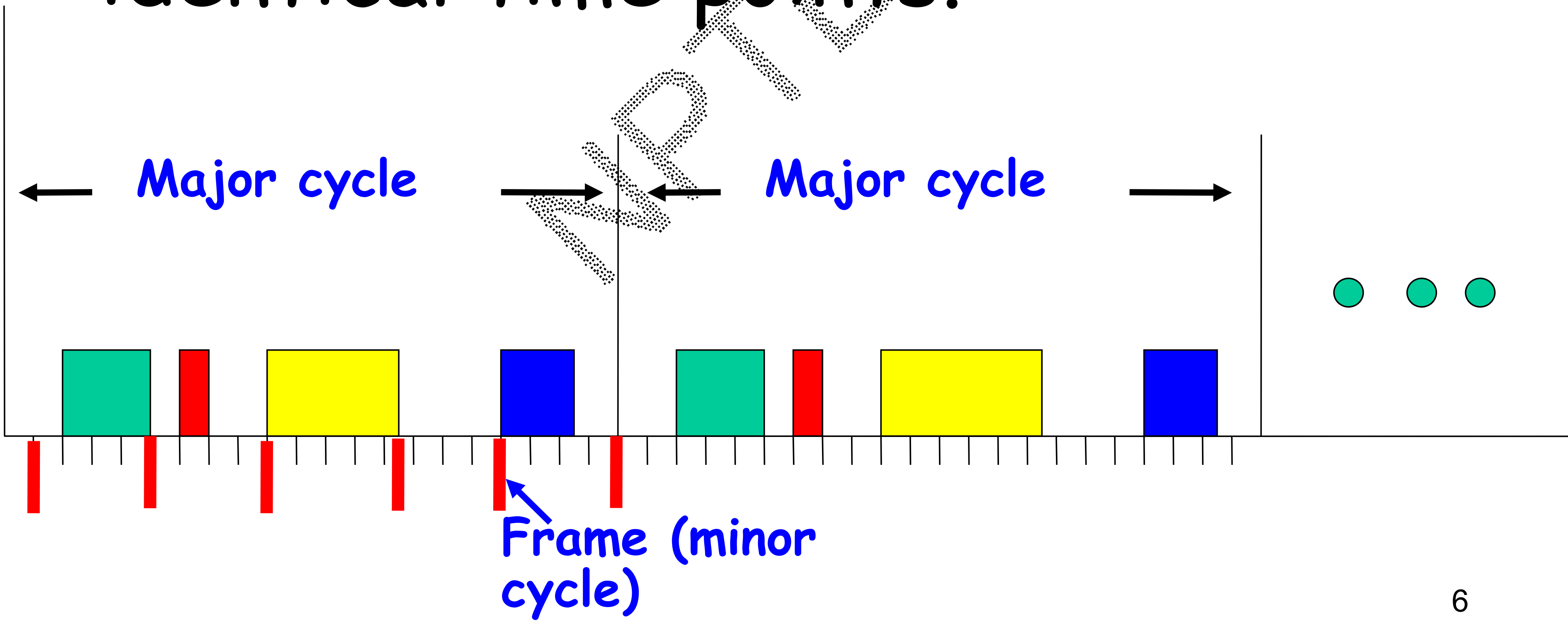
  - Exact start and completion time of jobs within frame is not known
  - But must know the max computation time
- Jobs are allocated to specific frames
- Major cycle is also called a Hyperperiod.

Cyclic Scheduler Basics

- If a schedule can not be found for the set of predefined jobs:
  - Then these are divided into job slices.
  - Essentially, divide a job into a sequence of smaller jobs.

Major Cycle

- In each major cycle:
  - The different jobs recur at identical time points.



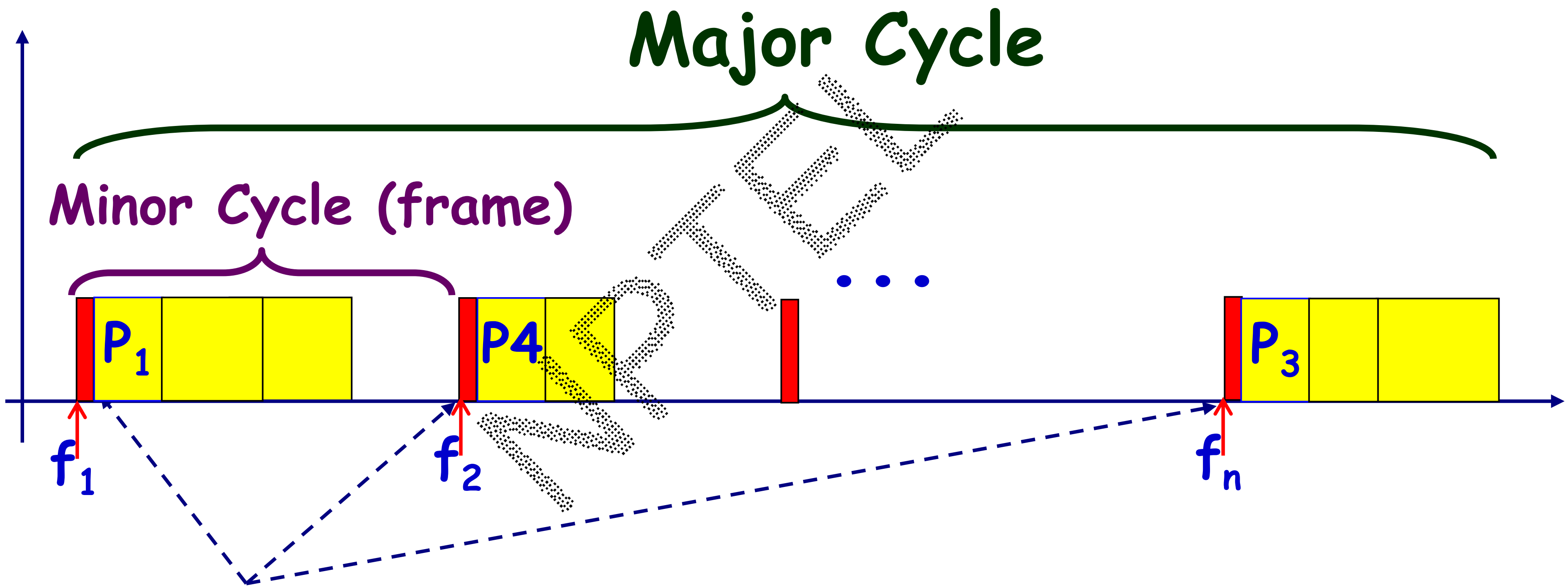
Major Cycle

- The major cycle of a set of tasks ST={T1,T2,...,Tn} is at least:
  - LCM(p1,p2,...,pn)
  - Holds even when tasks have arbitrary phasings.
  - Can be greater than LCM when F does not divide major cycle.

Minor Cycle (Frame)

- Each major cycle:
  - Usually contains an integral number of minor cycles (frames).
- Frame boundaries are marked:
  - Through interrupts generated from a periodic timer.

Major and Minor Cycle



- Cyclic scheduler runs in response to a tick event
- Red bar shows time to execute scheduler

A Typical Schedule

Frame	Task
F1	T2
F2	T3
F3	T2

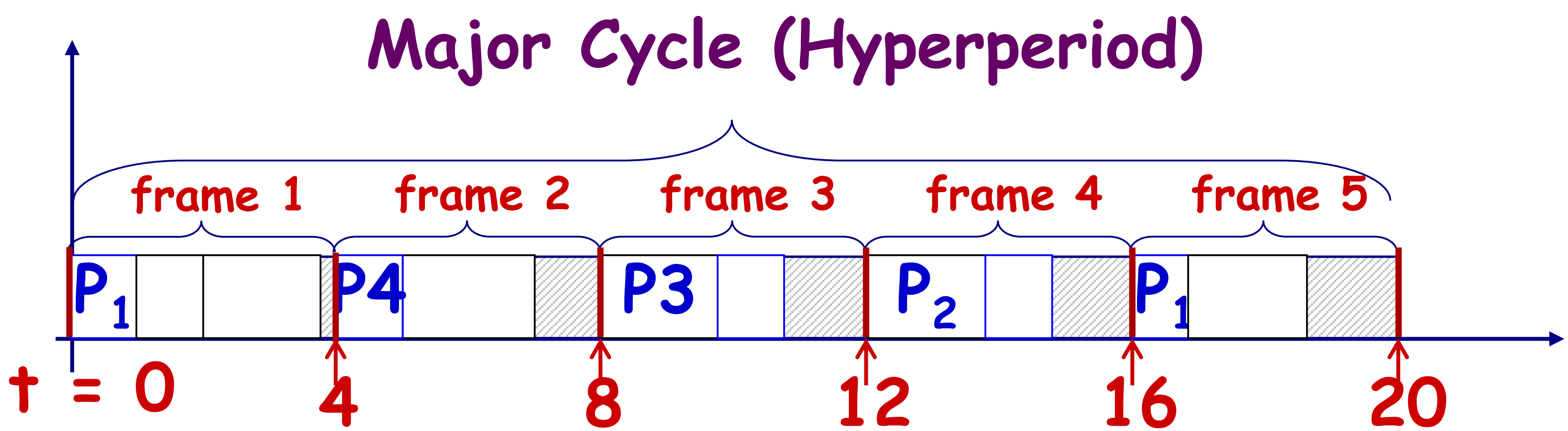
Constructing a Schedule

- Construct static schedule for a Major Cycle
- Cyclic Executive repeats this schedule
- There may be resulting idle intervals
  - if so attempt to arrange so they occur periodically

Cyclic Schedule

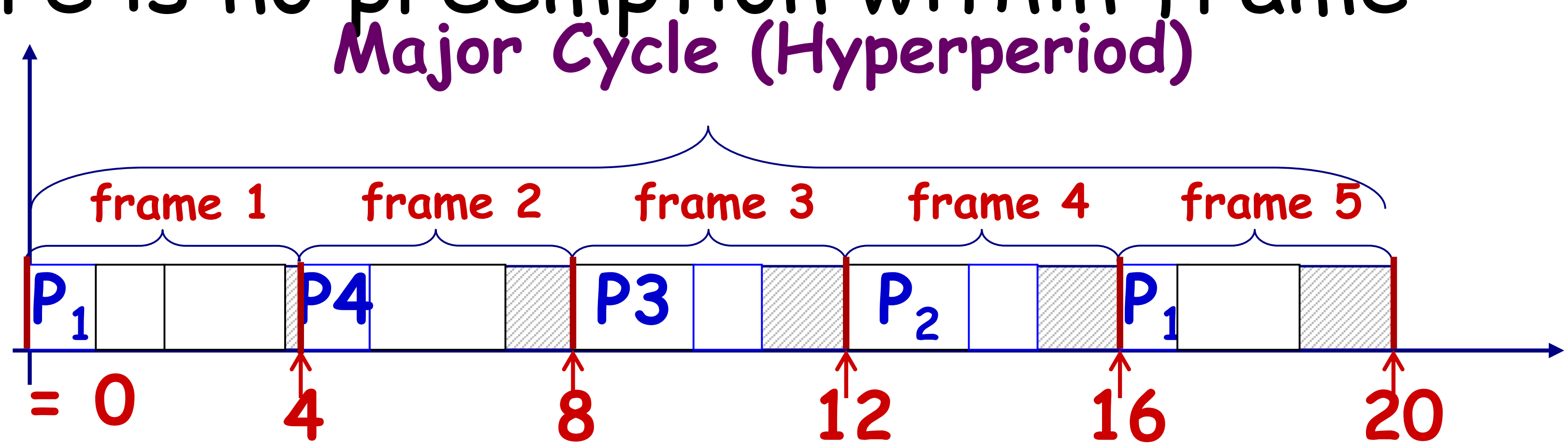
f_1	$tasks_1$
...	...
f_5	$tasks_5$

Task	=	(r, e)
T_1	=	(4, 1)
T_2	=	(5, 1.5)
T_3	=	(20, 1)
T_4	=	(20, 2)
H	=	20



Partitioning A Major Cycle into Frames

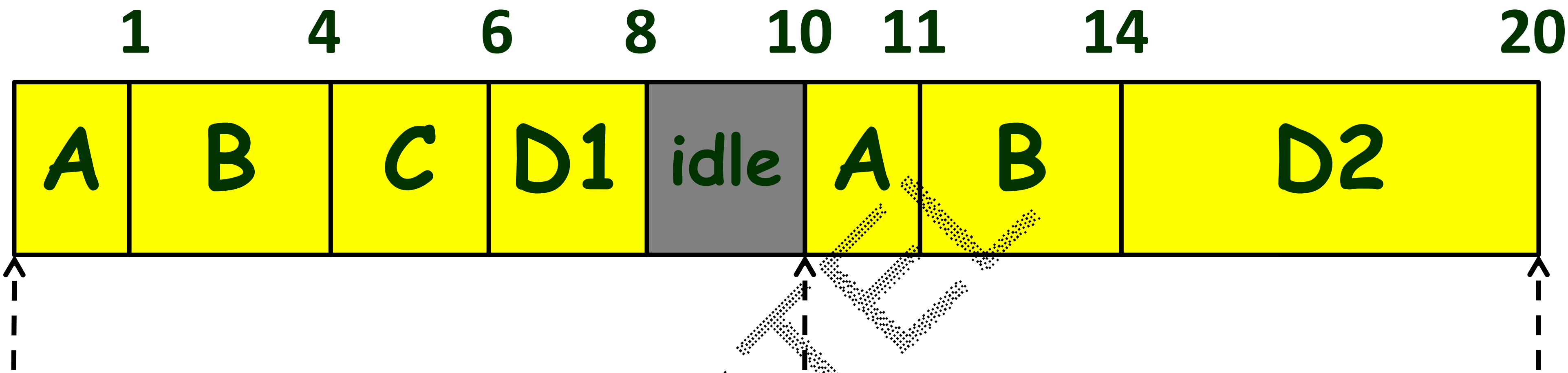
- Design steps:
  1. choose frame size,
  2. partition jobs into slices (if needed),
  3. place jobs/slices into frames.
- At Frames boundaries :
  - Cyclic executive performs scheduling
- There is no preemption within frame



Cyclic Executive Example

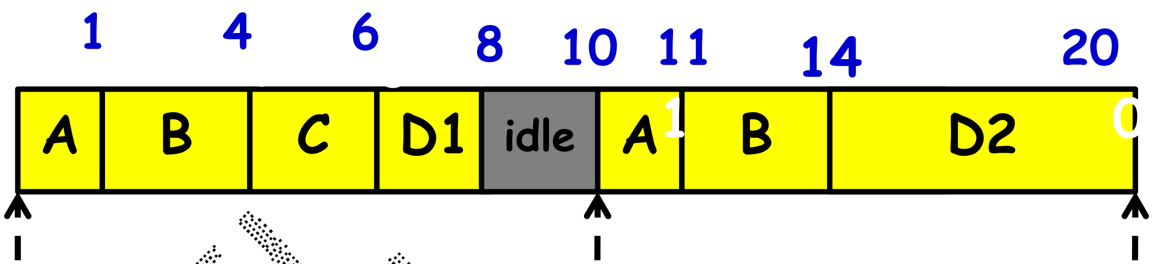
- Job=(computation time, period, relative deadline)
- A = (1, 10, 10)
- B = (3, 10, 10)
- C = (2, 20, 20)
- D = (8, 20, 20)
  - Two slices c(D1) = 2, c(D2) = 6
- Major Cycle = 20msec
- We select frame size = 10 msec

Schedule Example



- Could we create a different schedule?
- Is it better to distribute the idle time?

Schedule Example



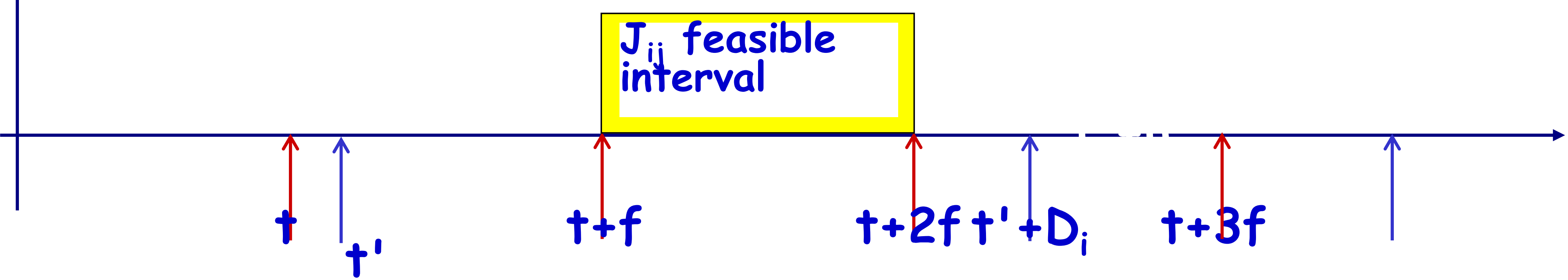
if Frame_no = 0 → run A ; B ; C ; D1;

else → run A ; B ; D2;

Frame_no = (Frame_no+1) mod 2;

Frame Size Constraints

1. Every job needs to start and complete within a frame
   - $f \geq \max(T_i)$, $1 \leq i \leq n$
2. Frame size f divides H (the Hyperperiod)
3. Between the release time and deadline of every job there is at least one frame



Minor Cycle (Frame)

- Each task is assigned to run in one or more frames.
- Frame size (F) is an important design parameter while using cyclic scheduler.
  - A selected frame size has to satisfy a few constraints.

Selecting an Appropriate Frame Size (F)

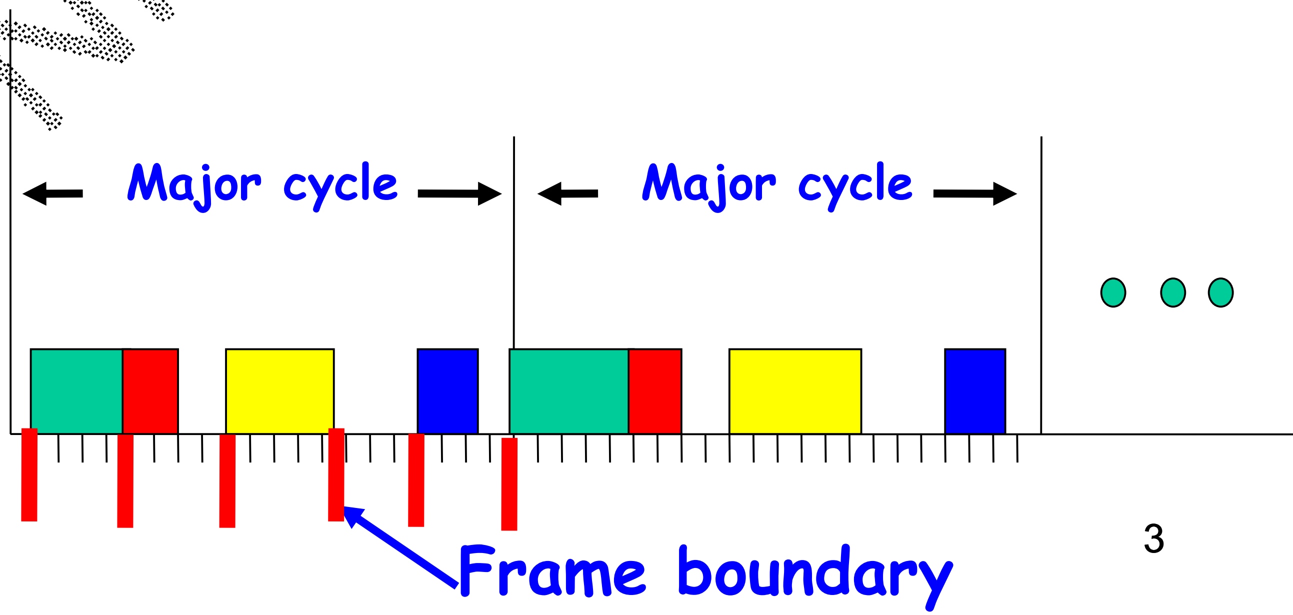
- Minimum scheduling overhead and chances of inconsistency:
  - F should be larger than each task size.
  - **Sets a lower bound.**
- Minimization of table size:
  - F should squarely divide major cycle.
  - **Allows only a few discrete frame size.**
- Satisfaction of task deadline:
  - Between the arrival of a task and its deadline:
    - At least one full frame must exist.
  - **Sets an upper bound**

Minimize Inconsistency

- Unless a job runs to completion:
  - Its partial results might be used by other jobs, leading to inconsistency
- To avoid scheduler overhead:
  - Selected frame size should be larger than execution time of each task.
  - Sets a lower bound for frame size.

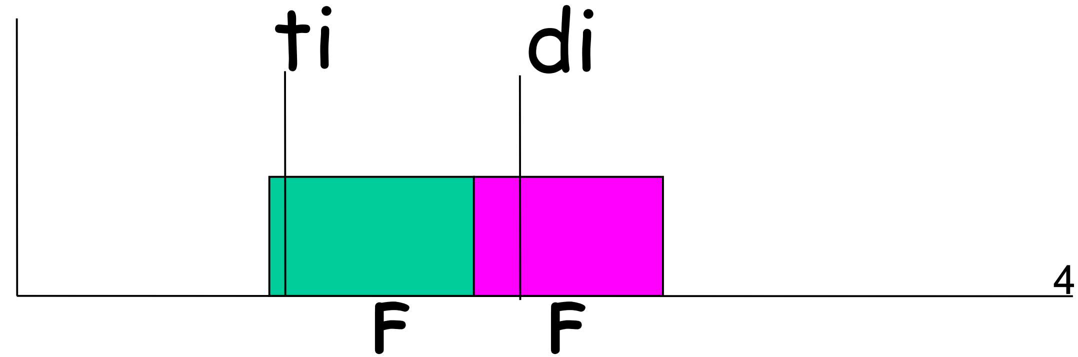
Minimization of Table Size

- Unless the minor cycle squarely divides the major cycle:
  - Storing schedule for one major cycle would not be sufficient.
  - Schedules in the major cycle would not repeat:
    - This would make the size of the table large.



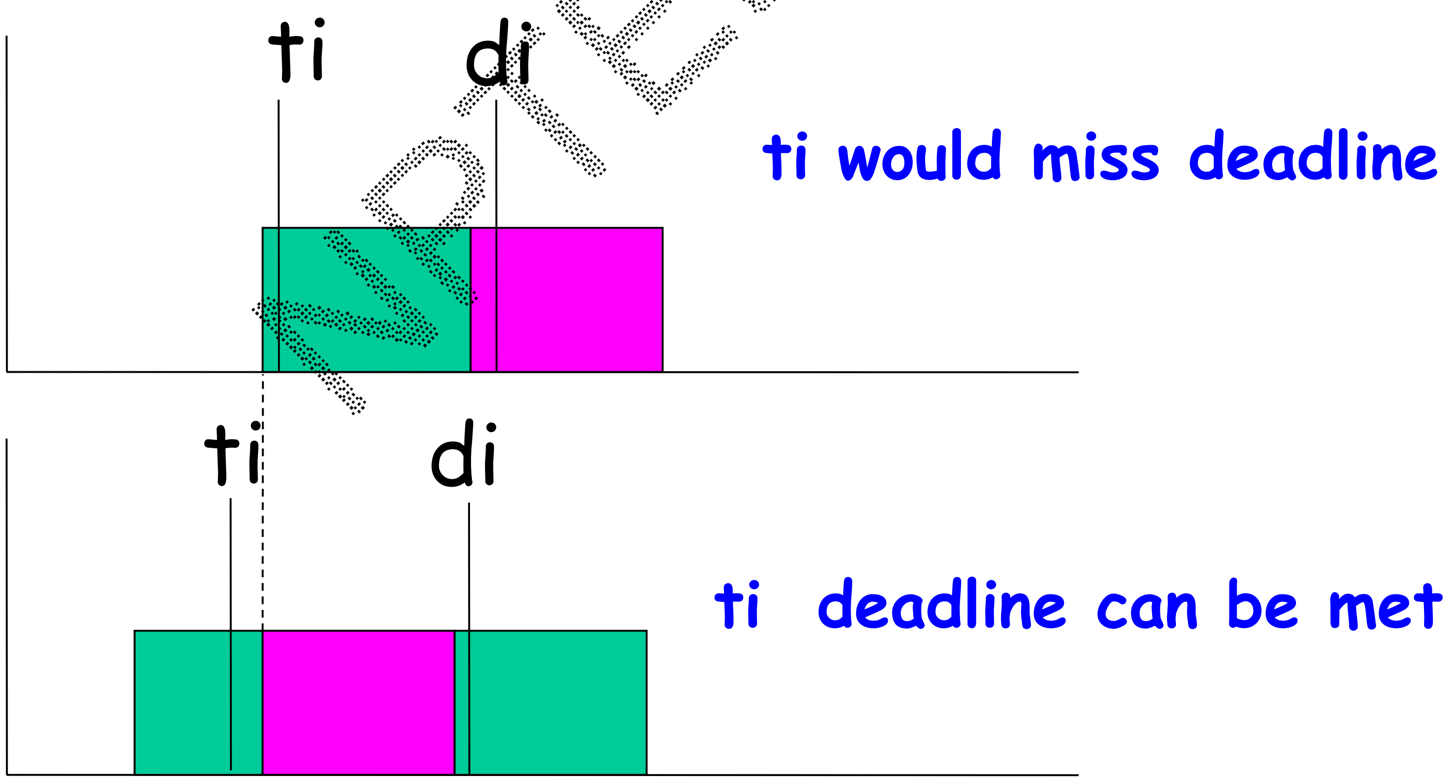
Satisfaction of Task Deadline

- Between the arrival of a task and its deadline:
  - At least one full frame must exist.
- If there is not even a single frame:
  - The task would miss its deadline,
    - By the time it could be taken up for scheduling, the deadline could be imminent.



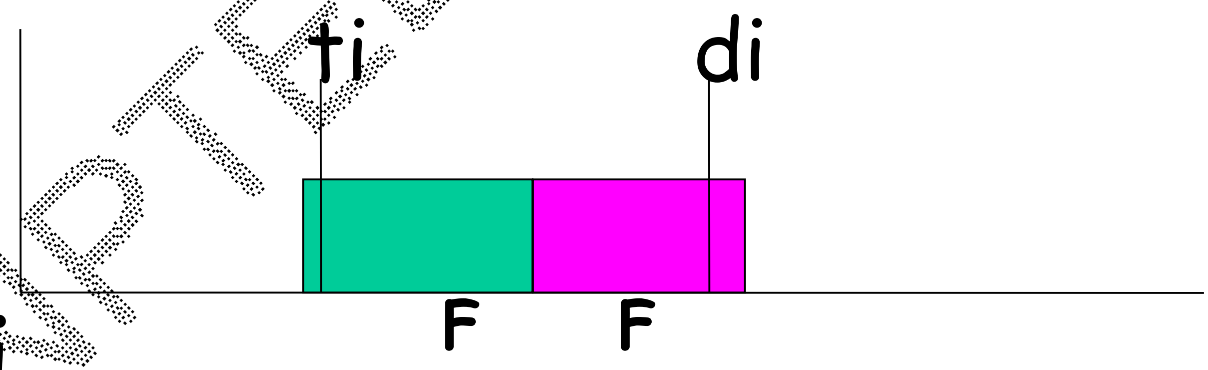
Satisfaction of Task Deadline

- The worst case for a task occurs when the task arrives just after a frame has started.



Satisfaction of Task Deadline

- The minimum separation of arrival time of ti from a frame start:
  - GCD(F,pi)



- Thus, for all ti
  - 2F-GCD(F,pi) ≤ di must be satisfied

Frame Size Constraints

(i) $f \geq \max(T_i)$ for all i

(ii) $2f - \gcd(p_i, f) \leq D_i$

(iii) $f*(\ LCM(p_i)/f) = LCM(pi)$

Selection of a Suitable Frame Size

- Several frame sizes may satisfy the constraints:
  - **Plausible frames.**
- A plausible frame size has been found:
  - Does not mean that the task set is schedulable.
- Also, the largest plausible frame size needs to be chosen:
  - **Scheduler overhead would be lower.**

Cyclic Scheduling Example 1

- Job=(computation time, period, relative deadline)
- A = (1, 6, 6)
- B = (2, 8, 8)

- Major cycle = 24
- Frame sizes: 2,3,4,6,12,24
- For F=2
  - 2*F-gcd(f,PA)= 2*2-2<=6 Acceptable
  - 2*F-gcd(f,PB) =2*2-2<=8 Acceptable

Example 1

- For F=3
  - 2*F – gcd(F,PA) = 6-3 <=6 Acceptable
  - 2*F- gcd(F,PB) = 6 – 1<=8 Acceptable
- For F=4
  - 2*F – gcd(F,PA) = 8-2 <=6 Acceptable
  - 2*F- gcd(F,PB) = 8 –4<=8 Acceptable

But not enough frames available!

- We can choose F=3

A = (1, 6, 6)
B = (2, 8, 8)

Example 2

- A = (1, 10, 10)
- B = (3, 10, 10)
- C = (2, 20, 20)
- D = (8, 20, 20)
- Major Cycle
  - Lcm (10, 10, 20, 20) = 20
- Frame Size
  - 1, 2, 4, 5, 10, 20 (must divide Major Cycle)
  - f >= max {1, 2, 3, 8} (geq longest computation time)
  - f can be 10 or 20
- Not enough frames in a major cycle to run all the jobs