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.

Real-Time Task Scheduling in Uniprocessors

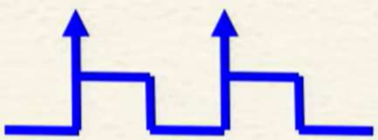
- Focus of much research during the 1970s and 80s.
- Real-time task schedulers can be broadly classified into:
  - **Clock-driven**
  - **Event-driven**
- This basic classification is done based on whether the scheduling points are defined by clock or other events

An Overview of Uniprocessor Real-Time Schedulers

Scheduler	Characteristics
Endless Loop	No Tasks, Polled
Timer-Driven Cyclic Executive	Single frequency, Polled
Multi-rate Cyclic Executive	Multiple frequencies, Polled
Priority-based Preemptive Scheduler	Polled + Interrupt driven

Clock-Driven Scheduling: Basics

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



- Scheduling points determined by a timer
- Timer can be one-shot or periodic
- Which task to be run when and for how long is stored in a table.

Task	Time
1	5
3	7
2	10
7	5
6	2

Assumptions

No actual processes exist:

- Each minor cycle is just 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.

Assumptions

- Clock-driven schedulers used in deterministic systems
- Tasks are assumed to be periodic:
  - Also the parameters of all periodic tasks are assumed to be known a priori
- Aperiodic jobs may exist
- There are no sporadic jobs
  - Recall: sporadic jobs have hard deadlines, aperiodic jobs do not

Clock-Driven Schedulers

- Also called:
  - Offline schedulers
  - Static schedulers

- Used extensively in embedded applications.

Pros and Cons of Clock-Driven Schedulers

- **Pro:**
  - **Compact:** Require very little storage space
  - **Efficient:** Incur very little runtime overhead.
  - **Small code:** Can be proven to work correctly
- **Con:**
  - **Inflexible:** Very difficult to accommodate sporadic tasks.
- Used in low cost applications

Popular Clock-Driven Schedulers

- **Round robin schedulers**
  - Not used in real-timer applications
- **Table-driven Schedulers**
- **Cyclic Schedulers**

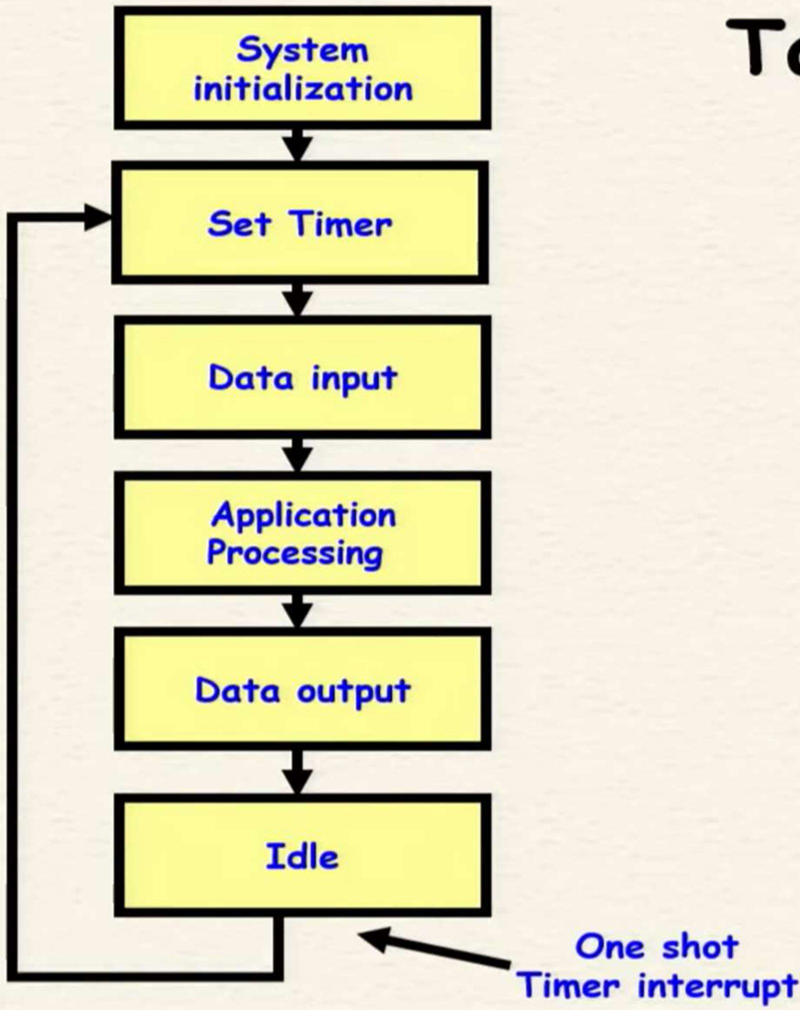
Round Robin Scheduler

- Periodically releases the CPU from long-running jobs based on timer interrupts:
  - So that short jobs can get a fair share of CPU time
- **Preemptive:** A process is forced to leave its running state and replaced by another running process
- **Time slice:** Interval between timer interrupts

Round Robin Scheduler: Some Thoughts

- Time slice is a critical parameter:
  - If time slice is too long, scheduler degrades to FIFO
  - If time slice is too short, throughput suffers as context switching cost dominates

Table-Driven Scheduler



Task	Timer
T1	50
T5	75
T2	30
T4	85
T3	70

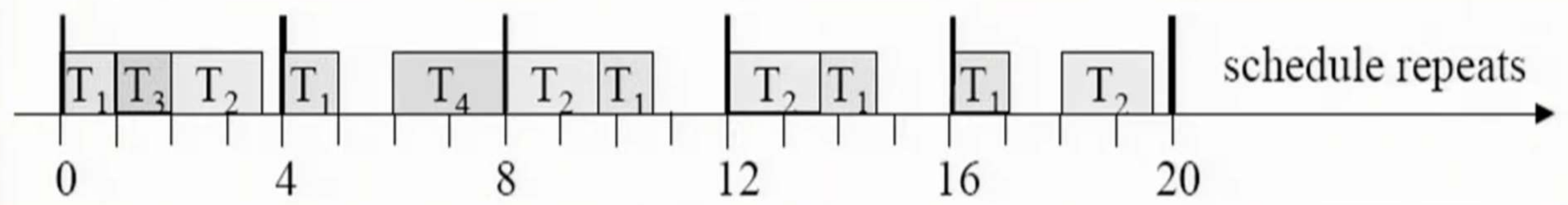
Basic Table-Driven Scheduler

```
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;
}
```

Task	Timer
T1	50
T5	75
T2	30
T4	85
T3	70

Table-Driven Schedule: Example

- Consider a system of four tasks, T1 = (4,1), T2 = (5,1), T3 = (20,1), T4 = (20,2).
- Static schedule:



A 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

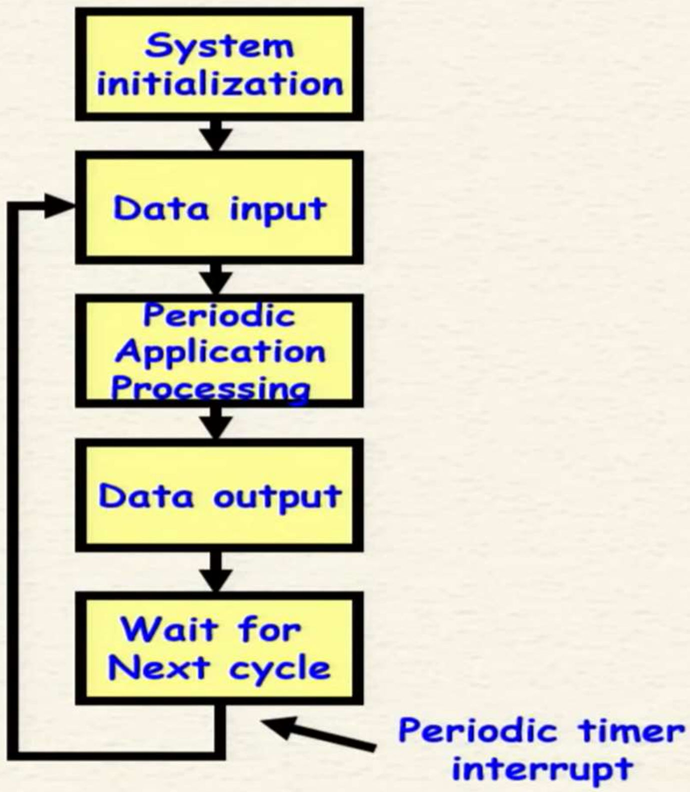
- Cyclic schedulers are very popular:
  - Extensively being used.
- Many tiny embedded applications have severe constraints on memory and processing power:
  - Cannot even host a microkernel RTOS, use cyclic schedulers.
- Also used by many safety-critical application

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)

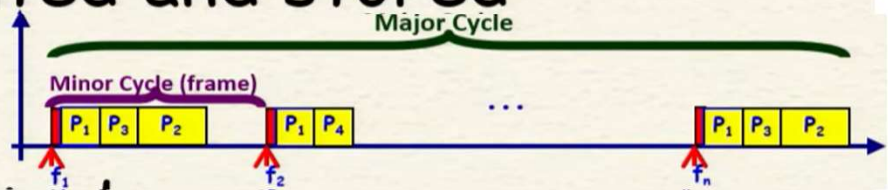
Task	Timer
T1	50
T5	75
T2	30
T4	85
T3	70

Cyclic Schedulers



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.

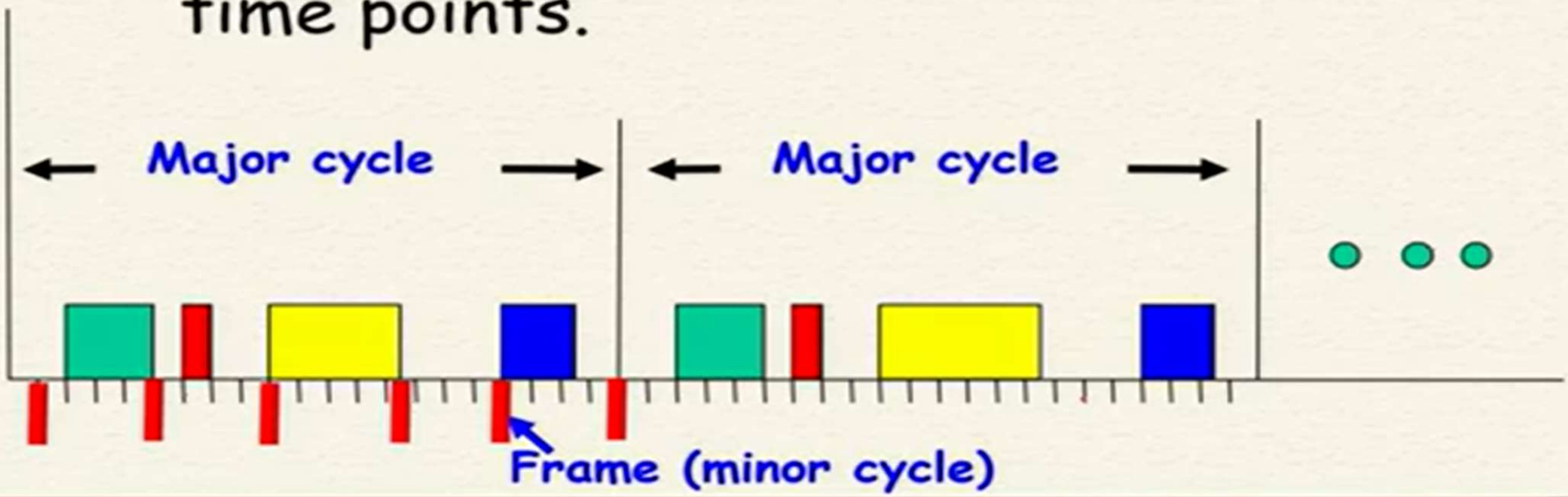
- Exact start and completion time of jobs within frame is not known
- Max computation time cannot exceed frame siz
- Jobs are allocated to specific frames
- Major cycle is also called a **Hyperperiod.**

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.



Event-Driven Schedulers

Event-Driven Schedulers

- Compared to clock-driven schedulers:
  - More proficient
- Can handle sporadic and aperiodic tasks.
  - Used in relatively complex applications.

Event-Driven Schedulers

- **Scheduling points:**
  - Defined by task completion and arrival events.
- **Preemptive schedulers:**
  - On arrival of a higher priority task, the running task may be preempted.
- **Simplest event-driven scheduler:**
  - Foreground-Background Scheduler

Event-Driven Schedulers: Two Characteristics

- **Preemptive schedulers:**
  - When a higher priority task becomes ready any executing lower priority task is preempted.
- **Greedy schedulers:**
  - These never keep the processor idle if a task is ready.