Common Approaches to Real-Time Scheduling

- **Clock-driven** (time-driven) schedulers
- **Priority-driven** schedulers
- Examples of priority driven schedulers
- Effective timing constraints
- How to reason about Schedulers: The Earliest-Deadline-First (EDF) Scheduler and its optimality

---

Clock-driven (time-driven) schedulers
- Scheduling decisions are made at specific time instants, which are typically chosen *a priori*.

Priority-driven schedulers
- Scheduling decisions are made when particular events in the system occur, *e.g.*
  - a job becomes available
  - processor becomes idle
- **Work-conserving**: processor is busy whenever there is work to be done.
Clock-Driven (Time-Driven) -- Overview

- **Scheduling decision time**: point in time when scheduler decides which job to execute next.
- Scheduling decision time in clock-driven schedulers is defined *a priori*.
- For example: Scheduler periodically wakes up and generates a portion of the schedule.
- **Special case**: When job parameters are known *a priori*, schedule can be pre-computed off-line, and stored as a table (*table-driven schedulers*)

![Diagram of scheduler job]

Priority-Driven -- Overview

- **Basic rule**: Never leave processor idle when there is work to be done. (Such schedulers are also called *work conserving*).
- Based on list-driven, greedy scheduling. Examples: FIFO, LIFO, SET, LET, EDF.
- Possible implementation of *preemptive* priority-driven scheduling:
  1. Assign priorities to jobs.
  2. Scheduling decisions are made when
     - Job becomes ready
     - Processor becomes idle
     - Priorities of jobs change
  3. At each scheduling decision time, choose ready task with highest priority.
- In non-preemptive case, scheduling decisions are made only when processor becomes idle.
Scheduling Decisions

- Scheduling decision points:
  1. The running process blocks, i.e. changes from running to waiting (current CPU burst of that thread is over).
  2. The running thread terminates.
  3. A waiting thread becomes ready (new CPU burst of that thread begins).
  4. The current thread is preempted, i.e. switches from running to ready.

Example: Priority-Driven Non-Preemptive Schedules

<table>
<thead>
<tr>
<th>Proc1</th>
<th>J₅</th>
<th>J₁</th>
<th>J₃</th>
<th>J₆</th>
<th>J₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc2</td>
<td>J₁</td>
<td>J₆</td>
<td>J₅</td>
<td>J₂</td>
<td>J₃</td>
</tr>
</tbody>
</table>

L = (J₁, J₂, J₃, J₄, J₅, J₆, J₇, J₈)

<table>
<thead>
<tr>
<th>Proc1</th>
<th>J₅</th>
<th>J₁</th>
<th>J₃</th>
<th>J₆</th>
<th>J₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc2</td>
<td>J₆</td>
<td>J₁</td>
<td>J₂</td>
<td>J₃</td>
<td>J₄</td>
</tr>
</tbody>
</table>

LET = (J₅, J₆, J₇, J₈, J₁, J₃, J₄, J₅, J₆, J₇)

<table>
<thead>
<tr>
<th>Proc1</th>
<th>J₅</th>
<th>J₁</th>
<th>J₃</th>
<th>J₆</th>
<th>J₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc2</td>
<td>J₁</td>
<td>J₆</td>
<td>J₅</td>
<td>J₂</td>
<td>J₃</td>
</tr>
</tbody>
</table>

Lₜ = (J₅, J₁, J₂, J₃, J₄, J₅, J₆, J₇)
Example: Priority-Driven Non-Preemptive Schedules

Proc1: 
\[ J_1 \quad J_2 \quad J_3 \quad J_4 \quad J_5 \quad J_6 \quad J_7 \quad J_8 \]

Proc2: 
\[ J_3 \quad J_4 \quad J_5 \quad J_6 \]

\[ L = (J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8) \]

Example: Priority-Driven Non-Preemptive Schedules

Proc1: 
\[ J_5 \quad J_8 \quad J_2 \quad J_6 \quad J_1 \quad J_3 \quad J_4 \quad J_7 \]

Proc2: 
\[ J_3 \quad J_4 \quad J_5 \quad J_6 \]

\[ \text{LET} = (J_5, J_8, J_2, J_6, J_1, J_3, J_4, J_7) \]
Example: Priority-Driven Non-Preemptive Schedules

### Example:

<table>
<thead>
<tr>
<th>Proc1</th>
<th>J5</th>
<th>J1</th>
<th>J3</th>
<th>J4</th>
<th>J6</th>
<th>J7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc2</td>
<td>J5</td>
<td>J2</td>
<td>J6</td>
<td>J7</td>
<td>J8</td>
<td>J1</td>
</tr>
</tbody>
</table>

$L = (J_8, J_1, J_2, J_3, J_4, J_5, J_6, J_7)$

---

Example: Priority-Driven Non-Preemptive Schedules

### Example:

<table>
<thead>
<tr>
<th>Proc1</th>
<th>J5</th>
<th>J1</th>
<th>J3</th>
<th>J4</th>
<th>J6</th>
<th>J7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc2</td>
<td>J5</td>
<td>J2</td>
<td>J6</td>
<td>J7</td>
<td>J8</td>
<td>J1</td>
</tr>
</tbody>
</table>

$L = (J_8, J_1, J_2, J_3, J_4, J_5, J_6, J_7)$

---

Example: Priority-Driven Non-Preemptive Schedules

### Example:

<table>
<thead>
<tr>
<th>Proc1</th>
<th>J5</th>
<th>J1</th>
<th>J3</th>
<th>J4</th>
<th>J6</th>
<th>J7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc2</td>
<td>J5</td>
<td>J2</td>
<td>J6</td>
<td>J7</td>
<td>J8</td>
<td>J1</td>
</tr>
</tbody>
</table>

$L = (J_8, J_1, J_2, J_3, J_4, J_5, J_6, J_7)$

---

© R. Bettati
Interlude 1: The EDF Algorithm

The EDF (Earliest-Deadline-First) Algorithm:
At any time, execute that available job with the earliest deadline.

Theorem: (Optimality of EDF)
In a system one processor and with preemptions allowed, EDF can produce a feasible schedule of a job set J with arbitrary release times and deadlines iff such a schedule exists.

Proof: By schedule transformation.

Proof of Optimality of EDF

- Assume that arbitrary schedule $S$ meets timing constraints.
- For $S$ to not be an EDF schedule, we must have the following situation:
Proof of Optimality of EDF (2)

- We now have two cases.

Case 1: \( L(A) > L(B) \)

Proof of Optimality of EDF (3)

- We now have two cases.

Case 2: \( L(A) \leq L(B) \)
EDF Not Always Optimal

- Case 1: When preemption is not allowed:
  
  \[
  \begin{align*}
  J_1 &= (0, 10, 3) \\
  J_2 &= (2, 14, 6) \\
  J_3 &= (4, 12, 4)
  \end{align*}
  \]

- Case 2: On a multiprocessor:
  
  \[
  \begin{align*}
  J_1 &= (0, 4, 1) \\
  J_2 &= (0, 4, 1) \\
  J_3 &= (0, 5, 5)
  \end{align*}
  \]

Interlude 2: Preemptive Scheduling of Jobs with Arbitrary Release Times, Deadlines, Execution Times

- Determine schedule over a hyperperiod (if necessary).
- Formulate scheduling problem as network flow problem.
Interlude 3: NP Completeness of Non-Preemptive Deadline Scheduling

**Theorem:** The problem of scheduling a non-preemptable set of jobs \( J_1, \ldots, J_n \) each with release time \( r_i \), deadline \( d_i \), and execution time \( C_i \), is NP-complete.

**Proof:** Transformation from PARTITION [Garey/Johnson, 1979]

Given: Finite set \( A = \{A_1, \ldots, A_i, \ldots, A_m\} \), each element of size \( a_i \).

Let \( B = \sum_{i=1}^{m} a_i \)

Partition \( A \) into two sets, each of same size.

Define a job set \( J_1, \ldots, J_{m+1} \) as follows:

- for \( 1 \leq i \leq m \), define \( J_i = \begin{cases} r_i & = 0 \\ d_i & = B + 1 \\ e_i & = a_i \end{cases} \)
- \( J_{m+1} = \begin{cases} r_{m+1} & = \left\lceil \frac{B}{2} \right\rceil \\ d_{m+1} & = \left\lceil \frac{(B+1)}{2} \right\rceil \\ e_{m+1} & = 1 \end{cases} \)

0 \hspace{1cm} \frac{B}{2} \hspace{1cm} \frac{B}{2}+1 \hspace{1cm} B+1