# Priority assignment

## Question

How do we set thread/message priority for the purpose of meeting deadlines?

## Static priorities

Assign a fixed priority to each thread and keep it constant until termination.

## Dynamic priorities

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.
Priority assignment

**Question**
How do we set thread/message priority for the purpose of meeting deadlines?

**Static priorities**
Assign a **fixed priority** to each thread and keep it constant until termination.

**Dynamic priorities**
Determine the priority at run-time from factors such as the time remaining until deadline.

:-(
In neither case a method exists that is both predictable and generally applicable to all programs!

:-)
It is possible to get by if we concentrate on programs of a restricted form.
Priority assignment

**Question**

How do we set thread/message priority for the purpose of meeting deadlines?

**Static priorities**

Assign a fixed priority to each thread and keep it constant until termination.

**Dynamic priorities**

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both predictable and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.
Priority assignment

**Question**

How do we set thread/message priority for the purpose of meeting deadlines?

**Static priorities**

Assign a **fixed priority** to each thread and keep it constant until termination.

**Dynamic priorities**

Determine the priority at run-time from factors such as the time remaining until deadline.

:-(

In neither case a method exists that is both **predictable** and generally applicable to all programs!

:-)

It is possible to get by if we concentrate on programs of a restricted form.
Priority assignment

Question
How do we set thread/message priority for the purpose of meeting deadlines?

Static priorities
Assign a fixed priority to each thread and keep it constant until termination.

Dynamic priorities
Determine the priority at run-time from factors such as the time remaining until deadline.

In neither case a method exists that is both predictable and generally applicable to all programs!

It is possible to get by if we concentrate on programs of a restricted form.
Initial restricted model

- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Assigning priorities

Analysis

Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Initial restricted model

- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Initial restricted model

- Only periodic reactions
- Fixed periods
- **No internal communication**
- Known, fixed WCETs
- Deadlines = periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Assigning priorities

**Analysis**

Initial restricted model

- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Initial restricted model

- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines $=$ periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Assigning priorities

Analysis

Initial restricted model

- Only periodic reactions
- Fixed periods
- No internal communication
- Known, fixed WCETs
- Deadlines = periods

More advanced courses on real-time systems discuss how to remove some of these restrictions.
Assigning priorities

Notation

Each reactive object obj\textsubscript{i} executes a message (thread/task/job) m\textsubscript{i} in a periodic fashion.

For each message m\textsubscript{i}:

- We know its period T\textsubscript{i} (given, determines the AFTER offset)
- We know its WCET C\textsubscript{i} (measured or analyzed)
- We know its relative deadline D\textsubscript{i} (given, equal to T\textsubscript{i} for now)

We want to determine its priority P\textsubscript{i}!
Each reactive object $\text{obj}_i$ executes a message (thread/task/job) $m_i$ in a periodic fashion.

For each message $m_i$:
- We know its period $T_i$ (given, determines the AFTER offset)
- We know its WCET $C_i$ (measured or analyzed)
- We know its relative deadline $D_i$ (given, equal to $T_i$ for now)

We want to determine its priority $P_i$!
Assigning priorities

Analysis

Notation

Each reactive object $\text{obj}_i$ executes a message (thread/task/job) $m_i$ in a periodic fashion.

For each message $m_i$

- We know its period $T_i$ (given, determines the AFTER offset)
- We know its WCET $C_i$ (measured or analyzed)
- We know its relative deadline $D_i$ (given, equal to $T_i$ for now)

We want to determine its priority $P_i$!
Each reactive object $\text{obj}_i$ executes a message (thread/task/job) $m_i$ in a periodic fashion.

For each message $m_i$:

- We know its period $T_i$ (given, determines the AFTER offset)
- We know its WCET $C_i$ (measured or analyzed)
- We know its relative deadline $D_i$ (given, equal to $T_i$ for now)

We want to determine its priority $P_i$!
Each reactive object $\text{obj}_i$ executes a message (thread/task/job) $m_i$ in a periodic fashion.

For each message $m_i$:
- We know its period $T_i$ (given, determines the AFTER offset)
- We know its WCET $C_i$ (measured or analyzed)
- We know its relative deadline $D_i$ (given, equal to $T_i$ for now)

We want to determine its priority $P_i$!
In concrete code

The application

```c
void ignite()
{
    BEFORE(D_1, &obj_1, m_1, arg_1);
    BEFORE(D_2, &obj_2, m_2, arg_2);
    ...
    BEFORE(D_n, &obj_n, m_n, arg_n);
}

STARTUP(ignite());
```
In concrete code

Class\textsubscript{i} obj\textsubscript{i} = initClass\textsubscript{i}();

int m\textsubscript{i}(Class\textsubscript{i} *self, int arg)\{
    // read ports
    // compute
    // update self state
    // write ports
    AFTERBEFORE(T\textsubscript{i}, D\textsubscript{i}, self, m\textsubscript{i},arg);
\}

Each D\textsubscript{i} = T\textsubscript{i}
Assigning priorities

Analysis

In concrete code

The objects

Class_i obj_i = initClass_i();

int m_i(Class_i *self, int arg){
    // read ports
    // compute
    // update self state
    // write ports
    AFTERBEFORE(T_i, D_i, self, m_i, arg);
}

Each D_i = T_i
Schematically (again)
Static priorities – method

Rate monotonic (RM)

Under the given assumptions, there exists a static priority assignment rule that is really simple

The shorter the period, the higher the priority

For RM, the actual priority values do not matter, only their relative order.

Because of our inverse priority scale, we can simply implement RM by letting $P_i = D_i (=T_i)$
Static priorities – method

Rate monotonic (RM)

Under the given assumptions, there exists a static priority assignment rule that is really simple

The shorter the period, the higher the priority

For RM, the actual priority values do not matter, only their relative order.

Because of our inverse priority scale, we can simply implement RM by letting $P_i = D_i (=T_i)$
Assigning priorities

Analysis

Static priorities – method

Rate monotonic (RM)
Under the given assumptions, there exists a static priority assignment rule that is really simple

The shorter the period, the higher the priority

For RM, the actual priority values do not matter, only their relative order.

Because of our inverse priority scale, we can simply implement RM by letting \( P_i = D_i (=T_i) \)
Given a set of periodic tasks with periods:

<table>
<thead>
<tr>
<th>T1</th>
<th>=</th>
<th>25ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>=</td>
<td>60ms</td>
</tr>
<tr>
<td>T3</td>
<td>=</td>
<td>45ms</td>
</tr>
</tbody>
</table>

Valid priority assignments:

<table>
<thead>
<tr>
<th>P1</th>
<th>=</th>
<th>10</th>
<th>P1</th>
<th>=</th>
<th>1</th>
<th>P1</th>
<th>=</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>=</td>
<td>19</td>
<td>P2</td>
<td>=</td>
<td>3</td>
<td>P2</td>
<td>=</td>
<td>60</td>
</tr>
<tr>
<td>P3</td>
<td>=</td>
<td>12</td>
<td>P3</td>
<td>=</td>
<td>2</td>
<td>P2</td>
<td>=</td>
<td>45</td>
</tr>
</tbody>
</table>
Given a set of periodic tasks with periods

<table>
<thead>
<tr>
<th>T1</th>
<th>=</th>
<th>25ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>=</td>
<td>60ms</td>
</tr>
<tr>
<td>T3</td>
<td>=</td>
<td>45ms</td>
</tr>
</tbody>
</table>

Valid priority assignments

<table>
<thead>
<tr>
<th>P1</th>
<th>=</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>=</td>
<td>19</td>
</tr>
<tr>
<td>P3</td>
<td>=</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>=</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>=</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>=</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P1</th>
<th>=</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>=</td>
<td>60</td>
</tr>
<tr>
<td>P3</td>
<td>=</td>
<td>45</td>
</tr>
</tbody>
</table>
For each task, the period is equal to its deadline. Arrows mark start of period. Blue when they get to execute. Gray when they have to wait for higher prio tasks to complete.
RM example

For each task, the period is equal to its deadline. Arrows mark start of period. Blue when they get to execute. Gray when they have to wait for higher prio tasks to complete.
Dynamic priorities – method

- Earliest Deadline First – EDF

Under the given assumptions, there exists a dynamic priority assignment rule that is really simple:

The shorter the time remaining until deadline, the higher the priority

Because EDF will want to distinguish between messages on basis of their *absolute* deadlines, priority values must use the same units as the system clock.

Under EDF, each activation $n$ of periodic task $i$ will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$
Dynamic priorities – method

Earliest Deadline First – EDF

Under the given assumptions, there exists a dynamic priority assignment rule that is really simple:

The shorter the time remaining until deadline, the higher the priority

Because EDF will want to distinguish between messages on basis of their **absolute** deadlines, priority values must use the same units as the system clock.

Under EDF, each activation $n$ of periodic task $i$ will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$
Assigning priorities

Analysis

Dynamic priorities – method

Earliest Deadline First – EDF

Under the given assumptions, there exists a dynamic priority assignment rule that is really simple:

The shorter the time remaining until deadline, the higher the priority

Because EDF will want to distinguish between messages on basis of their **absolute** deadlines, priority values must use the same units as the system clock.

Under EDF, each activation $n$ of periodic task $i$ will receive a new priority: $P_{i(n)} = \text{baseline}_{i(n)} + D_i$
EDF example

T1 arrives later, but its deadline is earlier than both T2’s and T3’s absolute deadlines!
T1 arrives later, but its deadline is earlier than both T2’s and T3’s absolute deadlines!
EDF example

Deadline of T1 < Deadline of T2
EDF example

(absolute) Deadline of T1 > (absolute) Deadline of T2
Under some given assumptions, there might be several ways of assigning priorities so that deadlines are met.

Clearly, a method that only fails if every other method also fails is preferred — such a method is called {	extbf{optimal}}.

- \text{RM} is optimal among static assignment methods.
- \text{EDF} is optimal among dynamic methods.
Under some given assumptions, there might be several ways of assigning priorities so that deadlines are met.

Clearly, a method that only fails if every other method also fails is preferred — such a method is called **optimal**

- RM is optimal among static assignment methods
- EDF is optimal among dynamic methods
Under some given assumptions, there might be several ways of assigning priorities so that deadlines are met.

Clearly, a method that only fails if every other method also fails is preferred — such a method is called optimal.

- RM is optimal among static assignment methods
- EDF is optimal among dynamic methods
Assigning priorities

Analysis

Schedulability

However, knowing that a priority assignment is the best one possible is not the same thing as knowing that it is good enough, i.e. knowing that deadlines actually will be met!

Assume all we know is that our priority assignment method is optimal. This is like knowing where the shortest path from A to B lies, but still not knowing if the path is short enough so that B can be reached in time.
However, knowing that a priority assignment is the best one possible is not the same thing as knowing that it is good enough, i.e. knowing that deadlines actually will be met!

Assume all we know is that our priority assignment method is optimal. This is like knowing where the shortest path from A to B lies, but still not knowing if the path is short enough so that B can be reached in time.
To answer whether our tasks will actually meet their deadlines at run-time, we need to determine if our task set is at all schedulable (recall that an optimal priority assignment method will produce a successful schedule if such a schedule exists).

Clearly, the question of schedulability must take the WCETs of tasks into account!
To answer whether our tasks will actually meet their deadlines at run-time, we need to determine if our task set is at all **schedulable** (recall that an optimal priority assignment method will produce a successful schedule if such a schedule exists)

Clearly, the question of schedulability must take the WCETs of tasks into account!
Utilization-based analysis

For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

That is, the CPU utilization of a periodic task $i$ is the ratio $\frac{C_i}{T_i}$, where $C_i$ is the WCET and $T_i$ is the period.

Note

Any task for which $C_i = T_i$ will effectively need exclusive access to the CPU!
For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

That is, the **CPU utilization** of a periodic task $i$ is the ratio $\frac{C_i}{T_i}$, where $C_i$ is the WCET and $T_i$ is the period.

**Note**

Any task for which $C_i = T_i$ will effectively need exclusive access to the CPU!
For a periodic task set, an important measure is how big a fraction of each turn a task is actually using the CPU.

That is, the CPU utilization of a periodic task $i$ is the ratio $\frac{C_i}{T_i}$, where $C_i$ is the WCET and $T_i$ is the period.

Note

Any task for which $C_i = T_i$ will effectively need exclusive access to the CPU!
Utilization-based analysis (RM)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

\[ U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1) \]

where \( N \) is the number of threads.

That is, the sum of all CPU utilizations must be less than a certain bound that depends on \( N \).
Utilization-based analysis (RM)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

\[ U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{1/N} - 1) \]

where \( N \) is the number of threads.

That is, the sum of all CPU utilizations must be less than a certain bound that depends on \( N \).
## Utilization bounds

<table>
<thead>
<tr>
<th>N</th>
<th>Utilization bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0 %</td>
</tr>
<tr>
<td>2</td>
<td>82.8 %</td>
</tr>
<tr>
<td>3</td>
<td>78.0 %</td>
</tr>
<tr>
<td>4</td>
<td>75.7 %</td>
</tr>
<tr>
<td>5</td>
<td>74.3 %</td>
</tr>
<tr>
<td>10</td>
<td>71.8 %</td>
</tr>
</tbody>
</table>

Approaches 69.3% asymptotically
Example A

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>WCET</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$T_i$</td>
<td>$C_i$</td>
<td>$U_i$</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>12</td>
<td>24%</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>10</td>
<td>25%</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>10</td>
<td>33%</td>
</tr>
</tbody>
</table>

The combined utilization $U$ is 82%, which is above the bound for 3 threads (78%).

The task set **fails** the utilization test.
Assigning priorities

Time-line for example A

Missed deadline
### Example B

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>WCET</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>$T_i$</td>
<td>$C_i$</td>
<td>$U_i$</td>
</tr>
<tr>
<td>1</td>
<td>80</td>
<td>32</td>
<td>40%</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>5</td>
<td>12.5%</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>4</td>
<td>25%</td>
</tr>
</tbody>
</table>

The combined utilization $U$ is 77.5%, which is below the bound for 3 threads (78%).

The task set **will meet** all its deadlines!
Time-line for example B
Example C

<table>
<thead>
<tr>
<th>Task</th>
<th>Period</th>
<th>WCET</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$T_i$</td>
<td>$C_i$</td>
<td>$U_i$</td>
</tr>
<tr>
<td>1</td>
<td>80</td>
<td>40</td>
<td>50%</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>10</td>
<td>25%</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>5</td>
<td>25%</td>
</tr>
</tbody>
</table>

The combined utilization $U$ is 100%, which is well above the bound for 3 threads (78%).

However, this task set still meets all its deadlines!

How can this be??
Time-line for example C
Characteristics

The utilization-based test

- Is **sufficient** (pass the test and you are OK)
- Is **not necessary** (fail, and you might still have a chance)

Why bother with such a test?

- Because it is so simple!
- Because only very specific sets of tasks fail the test and still meet their deadlines!
Characteristics

The utilization-based test

- Is **sufficient** (pass the test and you are OK)
- Is **not necessary** (fail, and you might still have a chance)

Why bother with such a test?

- Because it is so simple!
- Because only very specific sets of tasks fail the test and still meet their deadlines!
Utilization-based analysis (EDF)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

\[ U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \]

That is, the sum of all CPU utilizations must be less than 100%, independent of the number of tasks.

Unlike the case for RM, the utilization-based test for EDF is both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)
Utilization-based analysis (EDF)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

That is, the sum of all CPU utilizations must be less than 100%, independent of the number of tasks.

Unlike the case for RM, the utilization-based test for EDF is both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)
Utilization-based analysis (EDF)

Given a set of simple periodic tasks, scheduling with priorities according to RM will succeed if

\[ U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \]

That is, the sum of all CPU utilizations must be less than 100%, independent of the number of tasks.

Unlike the case for RM, the utilization-based test for EDF is both sufficient and necessary (demand more than 100% of the CPU and you are bound to fail!)
EDF vs RM

Similarities
- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF
- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
EDF vs RM

Similarities

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
Similarities

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
EDF vs RM

Similarities
- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF
- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
EDF vs RM

Similarities

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
EDF vs RM

Similarities

- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF

- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
Similarities
- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF
- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
  - Superior CPU utilization
EDF vs RM

Similarities
- Both algorithms are optimal within their class
- Both are easy to implement in terms of priority queues
- Both have simple utilization-based schedulability tests
- Both can be extended in similar ways

Advantages of EDF
- Close relation to terminology of real-time specifications
- Directly applicable to sporadic, interrupt-driven tasks
- Superior CPU utilization
Assigning priorities

Analysis

EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
EDF vs RM

**Drawbacks of EDF**

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when \( D_i \leq T_i \) (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
EDF vs RM

Drawbacks of EDF

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
EDF vs RM

**Drawbacks of EDF**

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
EDF vs RM

**Drawbacks of EDF**

- It exhibits random behaviour under transient overload (but so does RM, in fact, in a different way)
- RM predictably skips low priority tasks under constant overload (but EDF rescales task periods instead)
- Utilization-based test becomes more elaborate for EDF when $D_i \leq T_i$ (but is still feasible)
- Operating systems generally don’t support it (priority scales lack granularity, no automatic time-stamping)
- Few languages allow for natural deadline constraints

However, for reactive objects, EDF fits nice as an alternative to RM
In TinyTimber.c

```c
struct msg_block{
    ...
    Time baseline;
    Time priority;
    ...
};

void async(Time offset, Time prio, OBJECT *to, METHOD meth, int arg){
    ...
    m->baseline = MAX(TIMERGET(),
                      current->baseline + offset);
    m->priority = prio;
    ...
}
```
In TinyTimber.c

```c
struct msg_block{
    ...
    Time baseline;
    Time deadline;
    ...
};

void async(Time BL, Time DL,
        OBJECT *to, METHOD meth, int arg){
    ...
    m->baseline=MAX(TIMERGET(),
                   current->baseline+BL);
    m->deadline = m->baseline+DL;
    ...
}
```
More on real-time

Loosening the restrictions

What can be said if we allow tasks/threads/messages that are not periodic or for which deadline is not equal to period or that can block?

Other analysis

Response-time analysis, a more powerful technique than utilization based, is needed.

We leave this for more specialized courses on real-time (such as distributed real time systems)
More on real-time

Loosening the restrictions

What can be said if we allow tasks/threads/messages that are not periodic or for which deadline is not equal to period or that can block?

Other analysis

Response-time analysis, a more powerful technique than utilization based, is needed.

We leave this for more specialized courses on real-time (such as distributed real time systems)
More on real-time

Loosening the restrictions
What can be said if we allow tasks/threads/messages that are not periodic or for which deadline is not equal to period or that can block?

Other analysis
Response-time analysis, a more powerful technique than utilization based, is needed.

We leave this for more specialized courses on real-time (such as distributed real time systems)