Foundations of Cyber-Physical Systems

Real-Time Scheduling

Max Planck Institute for Software Systems

Rupak Majumdar & Björn Brandenburg

Agenda

1. High-Level Overview of Real-Time Systems
   → Approaches and classification.

2. Scheduling and Schedulability Analysis
   → How to guarantee that all deadlines will be met?

3. Response-Time Analysis (RTA)
   → Commonly employed in practice.
Readings

- Chapter 11 of textbook\textsuperscript{LeeSeshia11} (mainly overview)

- Two standard textbooks on real-time systems: Butazzo\textsuperscript{Buttazo11} (available as ebook via library) and Liu\textsuperscript{Liu00} (available in library, coverage of RTA)

- Chapter 2 of thesis\textsuperscript{Br11} provides intro to real-time + examples


High-Level Overview

Task Model & Classification
What does it mean to be “real-time”?

“The right answer at the right time.”

A real-time system is a system that has a dual notion of correctness:

- **Logical correctness**: it produces the correct outputs.
  → Just like any other software, it must meet its (logical) spec.

- **Temporal correctness**: it produces outputs at the correct times.
  → Unlike other software, it is also subject to a **temporal** spec.

The temporal specification mandates upper (and possibly lower) bounds on when an output should be produced in **units of physical time** (i.e., in real time).
Response Window

Real-time constraints typically specify a window during which a computation must be carried out.
Events vs. Jobs vs. Tasks vs. Recurring Activity

A real-time system **reacts** to events. ( Possibly the passage of time.)

There is a number of **event sources** / types of **recurring activities**.

There is one **task** for each event source / activity type.

Each time an event of a certain type occurs, the corresponding task releases a **job** that processes the event.

→ *Each task generates a possibly infinite sequence of jobs.*
Classification of Real-Time Workloads

1. **periodic** events / activity
   → “every <XXX> time units, do <something>” (e.g., sample sensor)

2. **sporadic** events
   → “whenever <some event> occurs, do <something>”
   → events are separated (at least) by a *minimum inter-arrival time*

3. **aperiodic** events
   → like sporadic, but no minimum inter-arrival time
   → system may have to **discard** some events in case of overload
“Hard” vs. “Soft” Real-Time Systems

There are many competing definitions. Some common examples:

• Based on **consequences**: in *hard* real-time systems, a timing violation can have *catastrophic consequences*. (Example: airbag)
  → *Everything else* is a *soft* real-time system.

• Based on **methods**: *formal methods* required for *hard* real-time systems.
  → Careful *engineering & testing* sufficient for *soft* real-time systems.

• Based on **correctness criteria**: in *hard* real-time systems, *all deadlines* must be met.
  → *Limited* deadline violations are ok in *soft* real-time systems.
  → But what does “limited” mean?
So#Real-Time#Systems$(1/2)$

No consensus on what “soft” means (except “not hard”).

Commonly found in practice:

- No analysis, but **observed deadline miss ratio** is “low.”
  → what “low” is depends on application (e.g., < 1%)
  → implicit assumption: deadline misses **not all at once**

- No analysis, but **observed tardiness** is “low” (e.g., < 1ms).
  → Intuition: there is some margin and “a little bit late” is acceptable
Soft Real-Time Systems (2/2)

No consensus on what “soft” means (except “not hard”).

- **Analytically derived deadline miss ratio bounds.**
  \[ m-k \text{-firm}: \text{out of any } k \text{ consecutive jobs, at least } m \text{ deadlines met} \text{ }^{HR97} \]

- **Analytically derived tardiness bounds.**\textsuperscript{Devi06}
  \[ \text{Bounded tardiness: every deadline may be missed.} \]
  \[ \text{But by at most constant amount that is bounded } a \text{ priori.} \]


Real-Time Computing is a Continuum

Both in terms of the **level of assurance** and the **strictness of the constraints**.

- **Hard**
  - static worst-case execution time analysis
  - & schedulability analysis
  - & strict guarantees

- **Soft**
  - no analysis, but observed deadline miss ratios and tardiness “low” enough

- Measured system parameters
  - & schedulability analysis
  - & strict or relaxed guarantees
Hard Real-Time Scheduling & Schedulability Analysis

How to guarantee all deadlines will be met?
Real-Time Scheduling

**Schedulability:** will all deadlines be met if scheduled with a scheduling policy \( P \)?

**Feasibility:** does there exist any way (i.e., any policy \( P \)) to meet all deadlines?

Note: always with respect to all possible event arrival sequences!

**Worst-case response time** (WCRT): how long will a job take at most to finish in the presence of interference from other jobs under scheduling policy \( P \)?

\[ \rightarrow \text{WCRT bound} \leq \text{deadline implies schedulability} \]
The Sporadic Task Model

The “Standard” Model of Real-Time Computation
Essential Workload Characteristics

To make real-time guarantees (= to rule out overload), we need to know three essential pieces of information:

• **How long** does it take to process a *single* event?
  → Called the *worst-case execution time* (WCET) of a task.

• **How frequent** are events?
  → Need bound on *arrival rate*, or its inverse, the *minimum inter-arrival time*.

• **How quickly** does the system need to react to each event type?
  → Called the *relative deadline* of a task.
The Sporadic Task Model

- **Task set** $\tau = \{T_1, T_2, \ldots, T_n\}$
  - number of tasks $n$

- **Task** $T_i = (e_i, p_i, d_i)$
  - worst-case execution time (WCET) $e_i$
  - period or minimum inter-arrival time $p_i$
  - relative deadline $d_i$

If jobs arrive exactly $p_i$ time units apart, then $T_i$ is a **periodic** task.
Properties of Jobs

A task $T_i$ releases an infinite sequence of jobs $J_{i,1}, J_{i,2}, J_{i,3}, \ldots$

- $a_{i,j}$ ... arrival time of job $J_{i,j}$
  - sporadic: $a_{i,j} + p_i \leq a_{i,j+1}$
  - periodic: $a_{i,j} + p_i = a_{i,j+1}$

- $f_{i,j}$ ... finish or completion time of job $J_{i,j}$

- $r_{i,j} = f_{i,j} - a_{i,j}$ ... response time of job $J_{i,j}$
  - slack = $d_i - r_{i,j}$, lateness = $r_{i,j} - d_i$, tardiness = $\max(0, \text{lateness})$
Sporadic Task Model Illustration

\[ p_i \]

\[ d_i \]

\[ r_{i,j} \]

\[ a_{i,j} \leq e_i \]

\[ f_{i,j} \]

\[ d_{i,j} \]

\[ a_{i,j}+1 \]

\[ \text{migration} \]

\[ \text{preempted} \]

\[ \text{scheduled on processor 1} \]

\[ \text{scheduled on processor 2} \]

\[ \text{release} \]

\[ \text{deadline} \]

\[ \text{completion} \]
Properties of Tasks

- $r_i = \max_j \{r_{i,j}\}$ ... maximum or worst-case response time of $T_i$

- $u_i = \frac{e_i}{p_i}$ ... the utilization of $T_i$

  → total utilization of the task set $u_{sum}(\tau) = \sum_{i=1}^{n} u_i$

- $\delta_i = \frac{e_i}{\min(p_i, d_i)}$ ... the density of $T_i$
Deadline Constraints

Three classes of deadline constraints:

- **implicit** deadline: $d_i = p_i$
  - “every $p_i$ time units, do <something>”
  - finish one event before the next arrives

- **constrained** deadline: $d_i \leq p_i$
  - “whenever <some event> occurs, do <something> within $d_i$ time units”

- **arbitrary** deadline: $d_i \leq p_i$
  - A bounded amount of backlog is permitted.
Additional Common Task Parameters

• $\phi_i = a_{i,1}$ ... the **phase** or **offset** of $T_i$
  → release time of the first job, interesting only for periodic tasks

• $j_i$ ... maximum **jitter**
  → delayed, irregular job release times (e.g., interrupt latency)

• $b_i$ ... maximum **blocking**
  → delays due to resource-sharing / mutex constraints (i.e., locks)

Assume $j_i = b_i = 0$ for now (→ independent tasks, zero jitter).
The sporadic task model\textsuperscript{Mok83} is an extension of the earlier Liu & Layland periodic task model\textsuperscript{LL73}.

- **LL restrictions:**
  - all tasks are periodic and have zero jitter
  - all tasks have implicit deadlines
  - all tasks are independent


How to schedule periodic and sporadic workloads?

How to ensure that $\forall T_i : r_i \leq d_i$?
Scheduling Approaches

1. static scheduling (= **offline** scheduling)
   → construct a repeating table offline

2. dynamic scheduling (= **online** scheduling)
   - FP: *rate-* or *deadline-monotonic*, ...
     → **assign priorities** to tasks at design time
   - JLFP: *earliest-deadline first* (EDF)
     → determine **job priority** on arrival
   - JLDP: *least-laxity first* (LLF)
     → **adjust** job priorities on the fly
Table-Driven Scheduling (1/2)

For periodic tasks, the schedule repeats after a while.

**Basic idea**: pre-determine a lookup table and store it in memory.

**How**: greedy approaches, branch-and-bound, max flow, ILP, SAT/SMT, CSP, ...

*Time-triggered architecture* \(^{KB03}\) (TTA) for distributed systems:
→ table-driven scheduling + synchronized clocks.

Table-Driven Scheduling (2/2)

Key benefits: very predictable, minimal runtime overheads, easy to validate, simple, easy to encode additional constraints.

Key limitations: inflexible, inefficient for sporadic tasks, table size.

Table size = hyperperiod $H(\tau)$:

$$H(\tau) = \text{lcm}\{p_i \mid T_i \in \tau\} = O(n!)$$

However, in practice, periods are typically not relatively prime.
Dynamic Scheduling

(= online, event-driven scheduling)
An FP Example Schedule

FP: schedule jobs in order of **decreasing task priority**.

Design-time problem: how should you assign priorities?
Classic Liu & Layland FP Results\textsuperscript{LL73} (1/2)

Assumptions: \textit{independent} (= no locks, no precedence constraints), implicit-deadline, periodic, fully preemptive tasks on a uniprocessor.

**Rate-monotonic** (RM): assign fixed priorities in order of increasing periods.

→ If $p_i < p_l$, then $T_i$ has higher priority than $T_l$.

**RM optimality**: if a task set (satisfying the stated assumptions) is schedulable with FP, then it is also schedulable with RM priorities.

Assumptions: independent (= no locks, no precedence constraints), implicit-deadline, periodic, fully preemptive tasks on a uniprocessor.

**RM utilization bound**: if \( u_{sum}(\tau) \leq n(2^{1/n} - 1) \), then \( \tau \) is schedulable under FP scheduling with RM priorities.

For \( n \to \infty, n(2^{1/n} - 1) = \ln 2 \approx 0.69 \).

→ To meet all deadlines with FP some idle time can be unavoidable.

---

An EDF Example Schedule

Earliest-Deadline First (EDF) run-time policy: schedule jobs in order of decreasing urgency. No design-time choices.
EDF is Optimal on a Uniprocessor (1/4)

Assumption: independent, fully preemptive jobs on a uniprocessor. → Applies to arbitrary job sets, not just periodic/sporadic tasks.

Claim: if there exists a schedule $S$ in which all deadlines are met, then no deadlines are missed under EDF, either.

Simple swapping argument...
EDF is Optimal on a Uniprocessor (2/4)

Step 1: Find a point in time $t_1$ in $S$ at which **EDF is violated**:  
→ some $J_{\text{later}}$ with deadline $d_{\text{later}}$ is scheduled or the CPU idles,  
→ while another job $J_{\text{earlier}}$ with deadline $d_{\text{earlier}}$ is pending, and  
→ if the CPU is not idle, then $d_{\text{earlier}} < d_{\text{later}}$ (EDF violation).
EDF is Optimal on a Uniprocessor (3/4)

**Step 2:** find $t_2$, the latest point in time that $J_{\text{earlier}}$ is scheduled in $S$.

$\rightarrow t_1 < t_2$ since $J_{\text{earlier}}$ is pending (= incomplete) at $t_1$.

$\rightarrow t_2 \leq d_{\text{earlier}}$ since no deadline is missed in $S$. 
EDF is Optimal on a Uniprocessor (4/4)

**Step 3:** swap the processor allocation at $t_1$ and $t_2$ in $S$.

**Invariant:** no deadline miss is introduced by the swap.

→ Trivially correct if CPU was idle at $t_1$. Otherwise:

→ If $J_{\text{later}}$ exists, the amount of processor service received by $J_{\text{later}}$ by time $d_{\text{later}}$ is unaffected since $t_2 \leq d_{\text{earlier}}$ and $d_{\text{earlier}} < d_{\text{later}}$.

**Finally:** repeat steps 1-3 until no more EDF violations are found.

→ $S$ is now an EDF schedule in which no deadlines are missed.
Classic Liu & Layland EDF Result\textsuperscript{LL73}

Assumptions: *independent* (= no locks, no precedence constraints), implicit-deadline, periodic, fully preemptive tasks on a uniprocessor.

**EDF utilization bound**: $\tau$ is schedulable under EDF scheduling if and only if $u_{sum}(\tau) \leq 1$.

Observation: if $u_{sum}(\tau) > 1$, then $\tau$ is infeasible.

$\rightarrow$ EDF is optimal on a uniprocessor.

Proof of L&L EDF Utilization Bound (1/6)

Suppose $u_{sum}(\tau) < 1$ and the first deadline is missed at time $t_2$.

**Busy window**: let $t_1$ denote the earliest time before $t_2$ such that the processor is **continuously busy** throughout $(t_1, t_2]$ executing jobs with deadlines on or before $t_2$. 
Observation 1: all jobs executing in $[t_1, t_2]$ must have:
→ a release time no earlier than $t_1$ and
→ a deadline no later than $t_2$.

How much processor service is required to finish all such jobs?
Proof of L&L EDF Utilization Bound (3/6)

How many jobs of one task $T_i$ can exist during $(t_1, t_2]$?

Bound: for each task $T_i$, there are at most $\left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor$ such jobs.
Proof of L&L EDF Utilization Bound (4/6)

Total amount of work required to complete all jobs during \((t_1, t_2]\):

\[
C(t_1, t_2) = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor \cdot e_i
\]

Where:

\[
\sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor \cdot e_i \leq \sum_{i=1}^{n} \left( \frac{t_2 - t_1}{p_i} \right) \cdot e_i = (t_2 - t_1) \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} = (t_2 - t_1) \cdot u_{sum}(\tau)
\]
Observation 2: to miss a deadline at $t_2$, the processor service required during $[t_1, t_2]$ must exceed the interval length.

$$(t_2 - t_1) < C(t_1, t_2)$$
Proof of L&L EDF Utilization Bound (6/6)

Putting Observation 1 and Observation 2 together:

\[(t_2 - t_1) < C(t_1, t_2) \leq (t_2 - t_1) \cdot u_{sum}(\tau)\]

Which is equivalent to:

\[1 < u_{sum}(\tau).\]
EDF vs. FP

• EDF is analytically superior, but FP is widely used in practice
  → inertia & standardization (POSIX, OSEK, AUTOSAR, ...)
  → “good enough” for closed systems (all tasks known a priori)
  → easier to reason about effects of overload under FP

• Component-based/contract-based design is easier with EDF
  → a priority by itself is meaningless (in contrast to deadlines)

See Butazzo\textsuperscript{Bu05} for further discussion...

Utilization Bounds: Discussion

• analytically elegant and classic, ground-breaking results
• efficient to compute (polynomial time)

BUT:

• RM bound is only sufficient, but not necessary in general
• utilization bounds provide only a yes/no answer
• **Constrained deadlines** are common
  → what about “density bounds”?
Feasible Task Set with Infinite Density

Consider a set of periodic tasks \( \tau = \{T_1, T_2, T_3, \ldots T_n\} \)
with \( \phi_i = 0, e_i = 1, p_i = n, \) and \( d_i = i \) for each \( T_i \in \tau. \)

The task set is feasible for any \( n: \) simply schedule by index.

**Total density:**

\[
\delta_{sum}(\tau) = \sum_{i=1}^{n} \frac{1}{\min(p_i, d_i)} = \sum_{i=1}^{n} \frac{1}{i}
\]

For \( n \to \infty, \) the harmonic series diverges.
Response-Time Analysis
Response-Time Analysis (RTA)

**Goal:** provide upper bound $R_i$ on worst-case response time.

- $r_i = \max_j \{r_i,j\}$ ... maximum response time in a concrete schedule

- $R_i \geq r_i$ ... response-time bound for any possible schedule

**Assumptions:**
→ FP scheduling: **task index = priority**, $T_1$ has the highest priority
→ independent, fully preemptive tasks
RTA for Finite Job Sets

a simpler warmup problem
A Simpler Problem: RTA for Finite Job Set (1/3)

Given: a finite job set \( \{J_1, \ldots, J_n\} \) with unknown release times (= sporadic tasks that each release a single job only), what is the worst-case response-time of a job \( J_i \)?
Key Concept: Busy Window

Level-$i$ busy window $(t_0, t_2]$:

- **Start** $t_0$: when a job of priority $i$ or higher is released and no jobs of priority $i$ or higher are incomplete.

- **End** $t_2$: when all jobs of priority $i$ or higher released during $(t_0, t_2]$ are complete.

**Properties:**
- Throughout level-$i$ busy window, only jobs of priority at least $i$ execute.
- Level-$i$ busy windows do not overlap (= can be analyzed independently).
A Simpler Problem: RTA for Finite Job Set (2/3)

Worst-case scenario = maximal level-$i$ busy window:
→ $J_i$ and all higher-priority jobs released together at time $t_0$.
→ $R_i = t_2 - t_0 = \text{sum("WCETs of } J_i \text{ and all higher-priority jobs")}$
A Simpler Problem: RTA for Finite Job Set (3/3)

\[ R_i = e_i + \sum_{J_h \in h_{p_j_i}} e_h \]

where \( h_{p_j_i} \) denotes the set of jobs with priority higher than \( J_i \).
RTA for Sporadic Tasks

with \textit{constrained} deadlines
How to determine the set of higher-priority jobs?

$T_3$ delayed by two jobs of $T_1$, but $T_4$ is delayed by five.

$\rightarrow$ the longer a job takes, the more often it is delayed!
Response Times are Non-Linear

Illustration: the longer a job takes, the more often it is delayed.

Bounding the Number of Preempting Jobs

How many jobs of $T_h$ “fit” into a level-i busy interval of length $\Delta$?

At most $\eta_h(\Delta) = \left\lfloor \frac{\Delta}{p_h} \right\rfloor$ jobs.
Bounding the Maximum Interference

How much **interference** (= delay) does $T_h$ cause for lower-priority jobs during a level-$i$ busy window ($i > h$) of length $\Delta$?

→ At most $\eta_h(\Delta) \cdot e_h = \left\lceil \frac{\Delta}{p_h} \right\rceil \cdot e_h$. 

![Diagram showing the busy window and the interference](image)
Response-Time Bound in Busy Window

assuming **constrained** deadlines

In a level-\(i\) busy window, only \(T_i\) and higher-priority tasks (\(h_{p_j}\)) execute.

\[ \rightarrow \text{Cumulative maximum interference bounds response time.} \]

If job \(J_{i,j}\) executes in a level-\(i\) busy window of length \(\Delta\):

\[
 r_{i,j} \leq e_i + \sum_{T_h \in h_{p_i}} \eta_h(\Delta) \cdot e_h = e_i + \sum_{T_h \in h_{p_i}} \left[ \frac{\Delta}{p_h} \right] \cdot e_h
\]

© MPI-SWS 2014
What is the Maximum Busy-Window Length?

assuming constrained deadlines

Given an upper bound $\Delta^\text{max}$, we can determine a bound $R_i \geq r_i$.

$$\forall J_{i,j}: r_{i,j} \leq e_i + \sum_{T_h \in h_p_i} \left[ \frac{\Delta^\text{max}}{p_h} \right] \cdot e_h = R_i$$

→ Is there a bound on the maximum busy-window length $\Delta^\text{max}$?
$T_i$ Job Completion = Level-$i$ Busy Window End

Observation: when $T_i$ releases a job $J_{i,j}$, then the corresponding busy interval $I$ ends when $J_{i,j}$ completes. (Why?)
Maximum Busy-Window Length = Maximum Response Time

- $\Delta = r_{i,j}$ if busy interval begins with arrival of $J_{i,j}$ (worst case).
- Therefore, w.r.t. any possible schedule: $\Delta^{\text{max}} = \max\{r_{i,j}\} = R_i$.  
  $\rightarrow$ Cyclic dependency!
RTA for Sporadic Tasks with Constrained Deadlines

The response time of task $T_i$ is bounded by the smallest $R_i$ in the range $e_i \leq R_i \leq d_i$ such that:

$$R_i = e_i + \sum_{T_h \in h_p} \left\lfloor \frac{R_i}{p_h} \right\rfloor \cdot e_h.$$ 

But how do we solve this equation?
Solving the RTA Recurrence

Cannot solve the recurrence symbolically due to the \([\ ]\) operator. Approximate: substitute \((R_i / p_h) + 1\) for \([R_i / p_h]\), then solve for \(R_i\).

**Exact solution:** find a fixed-point \(R_i^{(t+1)} = R_i^{(t)}\) of the recurrence

\[
R_i^{(t+1)} = e_i + \sum_{T_h \in h_{p_i}} \left[ \frac{R_i^{(t)}}{p_h} \right] \cdot e_h
\]

starting at \(R_i^{(t)} = e_i\); abort and fail when \(R_i^{(t+1)} > d_i\).
Table 2.4: Example task set.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>= $e_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_2$</td>
<td>= $e_2 + \left\lfloor \frac{R_2}{p_1} \right\rfloor \cdot e_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_3$</td>
<td>= $e_3 + \left\lfloor \frac{R_3}{p_1} \right\rfloor \cdot e_1 + \left\lfloor \frac{R_3}{p_2} \right\rfloor \cdot e_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_4$</td>
<td>= $e_4 + \left\lfloor \frac{R_4}{p_1} \right\rfloor \cdot e_1 + \left\lfloor \frac{R_4}{p_2} \right\rfloor \cdot e_2 + \left\lfloor \frac{R_4}{p_3} \right\rfloor \cdot e_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$u_{sum}(\tau) = \frac{171}{180} \approx 0.95$

$R_4 = e_4 + \left\lfloor \frac{R_4}{p_1} \right\rfloor \cdot e_1 + \left\lfloor \frac{R_4}{p_2} \right\rfloor \cdot e_2 + \left\lfloor \frac{R_4}{p_3} \right\rfloor \cdot e_3$

$= 3 + \left\lfloor \frac{18}{4} \right\rfloor \cdot 1 + \left\lfloor \frac{18}{5} \right\rfloor \cdot 1 + \left\lfloor \frac{18}{9} \right\rfloor \cdot 3 = 18$
Response-Time Analysis Notes

- Discovered independently at least three times \textsuperscript{JP86} \textsuperscript{LSD89} \textsuperscript{ABTR93}
  - Also known as \textit{time demand analysis} \textsuperscript{LSD89} (TDA)
  - Time complexity: actually NP-hard \textsuperscript{ER08} (but fast enough in practice)


RTA for Sporadic Tasks

with \textit{arbitrary} deadlines
What happens if $r_i > p_i$?

• Level-$i$ busy window can contain **more than one job** of $T_i$.

• Assumption: tasks are *sequential* ($\rightarrow$ jobs processed in FIFO order).

• *Incorrect “folk wisdom”: the first job (released with maximum higher-priority interference) incurs maximum response time.*

  $\rightarrow$ **No!** Earlier jobs of $T_i$ can delay later jobs of $T_i$. \(^{Le90}\)

---

Lehoczky’s Counter Example$^{Le90}$

- $T_1 : \{p_1 = 70, e_1 = 26\}$ and $T_2 : \{p_2 = 100, e_2 = 62\}$

![Diagram of task sets $T_1$ and $T_2$](based on an illustration by Jim Anderson)

Response-Time Analysis if $r_i > p_i$

- Can we predict which job of $T_i$ in a level-$i$ busy window incurs the maximum response time?
  → No, not in general: need to consider all jobs in a busy window.

- Let $w_i(q)$ denote the max. response time of the $q^{th}$ job of $T_i$ in busy window.

- Case $q = 1$ almost as before, but we permit solutions $w_i(1) > p_i$.

$$w_i(1) = e_i + \sum_{T_h \in h p_i} \left[ \frac{w_i(1)}{p_h} \right] \cdot e_h$$
Case $q > 1$: Considering Delay from Earlier Jobs

**Observation 1:** The $q^{\text{th}}$ job of $T_i$ in a level-$i$ busy window is delayed by $q - 1$ earlier jobs of $T_i$.

**Observation 2:** The $q^{\text{th}}$ job of $T_i$ in a level-$i$ busy window is released no earlier than $(q - 1) \cdot p_i$ time units after beginning of the interval.

$$w_i(q) = q \cdot e_i + \sum_{T_h \in hp_i} \left[ \frac{w_i(q) + (q - 1) \cdot p_i}{p_h} \right] \cdot e_h - (q - 1) \cdot p_i$$
General RTA Recurrence

\[ w_i(q) = q \cdot e_i + \sum_{T_h \in \text{hp}_i} \left[ \frac{w_i(q) + (q - 1) \cdot p_i}{p_h} \right] \cdot e_h - (q - 1) \cdot p_i \]

**Worst-case scenario:** the first job of \( T_i \) and each \( T_h \) released at beginning of level-\( i \) busy interval; subsequent jobs released as quickly as possible thereafter. (Proof omitted here.)
General RTA Example (1/2)

• $T_1: \{p_1 = 70, e_1 = 26\}$ and $T_2: \{p_2 = 100, e_2 = 62\}$

Let’s have a look at $J_{2,5}$, the fifth job in the level-2 busy window.
General RTA Example (2/2)

\[ T_1 : \{p_1 = 70, e_1 = 26\}, \quad T_2 : \{p_2 = 100, e_2 = 62\}, \quad r_{2,5} = 118 \]

- \( w_i(q) = q \cdot e_i + \sum_{T_h \in h_p_i} \left[ \frac{w_i(q) + (q - 1) \cdot p_i}{p_h} \right] \cdot e_h - (q - 1) \cdot p_i \)

- \( w_2(5) = 5 \cdot 62 + \left[ \frac{118 + (5 - 1) \cdot 100}{70} \right] \cdot 26 - (5 - 1) \cdot 100 \)

- \( w_2(5) = 310 + \left[ \frac{518}{70} \right] \cdot 26 - 400 = 310 + 8 \cdot 26 - 400 \)

- \( w_2(5) = 310 + 208 - 400 = 118 \checkmark \)
Bounding $q$: How many jobs to consider?

Recall that:

• Level-$i$ busy intervals do not overlap.

• Level-$i$ busy interval ends with a completion of a job of $T_i$.

→ If $(q - 1) \cdot p_i + w_i(q) < q \cdot p_i$, then level-$i$ busy interval necessarily ends before $(q + 1)^{th}$ job can be released.
General RTA for Sporadic Tasks with Arbitrary Deadlines

The response-time of task $T_i$ is bounded by:

$$R_i = \max\{w_i(q) \mid q = 1, 2, \ldots\},$$

where $w_i(q)$ is the smallest positive solution to the recurrence

$$w_i(q) = q \cdot e_i + \sum_{T_h \in hpi} \left[ \frac{w_i(q) + (q - 1) \cdot p_i}{p_h} \right] \cdot e_h - (q - 1) \cdot p_i.$$
RTA for Sporadic Tasks

with jitter and constrained deadlines
What is Jitter?

**Release jitter**: a job becomes *available for execution* only some time after the triggering event occurred.

- Example: sensor has some *latency* in detecting events
- Example: task triggered by periodic network transmissions, but data packet is *delayed*
- Example: in a **distributed real-time system**, the time at which jobs of consumer tasks become available for execution depend on the response time of producer tasks.
Release Jitter Illustration & Terminology

Jitter reduces the effective minimum separation between job releases

apparent minimum separation: \( p_i - j_i = t_2 - t_1 \)

\( p_i = t_2 - t_0 \)

release jitter

job available for execution

\( j_i \)

job arrival (event occurred)

job release (available for execution)

earliest-possible release of next job
Why is Jitter Problematic? (1/3)

\[ T_1 : \{ p_1 = 5, e_1 = 2 \} \text{ and } T_2 : \{ p_2 = 10, e_2 = 6 \} \]

Worst case without jitter: all jobs arrive together.
Why is Jitter Problematic? (2/3)

\[ T_1 : \{ p_1 = 5, e_1 = 2 \} \text{ and } T_2 : \{ p_2 = 10, e_2 = 6 \} \]

Not a worst case without jitter: shifted job arrival.
Why is Jitter Problematic? (3/3)

$T_1 : \{p_1 = 5, e_1 = 2, j_1 = 2\}$ and $T_2 : \{p_2 = 10, e_2 = 6, j_2 = 0\}$

Jitter moves additional job into level-2 busy window (new worst case).
Only Jitter of First Job in Busy Window is Relevant

Shifting interference within busy window does not increase response time. (First job jitter shifts interference into busy window.)
Accounting for Jitter (1/2)

Max. number of jobs in busy window of length $\Delta$:
Accounting for Jitter (2/2)

Max. number of jobs in busy window of length $\Delta$:

- **without** jitter: $\eta_h(\Delta) = \left\lfloor \frac{\Delta}{p_h} \right\rfloor$

- **with** jitter: $\eta'_h(\Delta) = \left\lfloor \frac{j_h + \Delta}{p_h} \right\rfloor$
Own Jitter also Increases Response Time

Without jitter: $R_1 = e_1 = 2$. With jitter: $R_1 = j_1 + e_1 = 4$. 
RTA for Tasks with **Jitter & Constrained** Deadlines

The response time of task $T_i$ is bounded by the smallest

$$R_i = j_i + R_i'$$

where $e_i \leq R_i' \leq d_i - j_i$ and

$$R_i' = e_i + \sum_{T_h \in h_{p_i}} \left[ \frac{j_h + R_i'}{p_h} \right] \cdot e_h.$$
Coming up next: RTA for...

• jitter + arbitrary deadlines
• blocking + jitter + arbitrary deadlines
• bursts + blocking + jitter + arbitrary deadlines
• activation offsets + bursts + blocking + jitter + arbitrary deadlines
• ...

...can’t we do something more general?
An Event Stream Model\textsuperscript{RRE03}
Compositional Performance Analysis (CPA)

The model underlying the tool \textit{SymTA/S by Symtavision}.

\textsuperscript{RRE03} K. Richter, R. Racu, and R. Ernst (2003). Scheduling Analysis Integration for Heterogenous Multiprocessor SoC. In Proc. RTSS'03.
Bursty Event Streams

**Challenge**: *short-term* high demand vs. *long-term* low utilization.

Hypothetical example: sensor data delivered by UDP

- Packet flow: at most 600 UDP packets per minute.
  - → **Average** rate of one packet every 100ms \((p_i \approx 100)\)

- **Bursty** arrivals: can see up to 8 packets in 100ms
  - → **Worst-case** minimum separation is 12.5 \((p_i \approx 12.5)\)
  - → If modeled as a sporadic task → 4800 packets per minute!

**Goal**: express both in one model.
Generalizing Event Arrival Bounds

So far we have assumed *specific event models* (periodic, sporadic, sporadic + jitter) to derive **bounding functions**.

\[ R'_i = e_i + \sum_{T_h \in h \ell} \left[ \frac{j_h + R'_i}{p_h} \right] \cdot e_h \]

**What if we make the bounding functions "first-class citizens"?**
Event Stream Model

- $T_i$ ... **event stream** with possibly **complex/bursty arrival patterns**
  $\rightarrow e_i$ ... per-event / per-job WCET (just as before)
  $\rightarrow d_i$ ... relative deadline (for each event/job, just as before)

- $\eta_i^+ (\Delta)$ ... **upper event arrival function** that upper-bounds the maximum number of events in any interval of length $\Delta$.

- $\eta_i^- (\Delta)$ ... **lower event arrival function** that lower-bounds the minimum number of events in any interval of length $\Delta$. 
Standard *Periodic* Event Streams

For period $P$ and Jitter $J$; $\eta_i^+ (\Delta)$ denoted by solid curve.\textsuperscript{RE12}

Standard Event Streams

Periodic with period $p_i$:

$$\eta_i^+(\Delta) = \left\lfloor \frac{\Delta}{p_i} \right\rfloor \quad \text{and} \quad \eta_i^-(\Delta) = \left\lceil \frac{\Delta}{p_i} \right\rceil$$

Periodic with maximum jitter $j_i$ and minimum jitter 0:

$$\eta_i^+(\Delta) = \left\lfloor \frac{\Delta + j_i}{p_i} \right\rfloor \quad \text{and} \quad \eta_i^-(\Delta) = \max\left(\left\lfloor \frac{\Delta - j_i}{p_i} \right\rfloor, 0\right)$$
**Standard Burst Model**

**Sporadically periodic**\(^{ABTR93}\) (= sporadic, limited-duration bursts) with

→ maximum burst length of at most \(n^b_i\) jobs

→ minimum event separation \(p^b_i\) during bursts

→ minimum burst separation \(p_i\), where \(n^b_i \cdot p^b_i \leq p_i\)

\[
\eta^+_i (\Delta) = \left\lfloor \frac{\Delta}{p_i} \right\rfloor \cdot n^b_i + \min \left( n^b_i, \left\lfloor \frac{\Delta}{p_i} \right\rfloor \cdot p^b_i \right)
\]

---

Standard Sporadic Event Streams

*sporadic* = arbitrary long times of inactivity

• $\eta_i^+ (\Delta)$ just like in the periodic case.

• But $\eta_i^- (\Delta) = 0$ for any $\Delta$.

**Irregular arrivals**: can extract $\eta_i^+ (\Delta)$ from traces of system activity.

→ Useful for "black box" components / event sources.
RTA for event streams with \textit{constrained} deadlines

The response time of events in stream $T_i$ is bounded by the smallest

$$R_i = j_i + R'_i$$

where $e_i \leq R'_i \leq d_i - j_i$ and

$$R'_i = e_i + \sum_{T_h \in h_p_i} \eta^+_h (R'_i) \cdot e_h.$$
How to Apply RTA for \textit{Arbitrary} Deadlines?

Recall RTA for $q^{th}$ job in a busy interval:

$$w_i(q) = q \cdot e_i + \sum_{T_h \in h p_i} \left[ \frac{w_i(q) + (q - 1) \cdot p_i}{p_h} \right] \cdot e_h - (q - 1) \cdot p_i.$$  

→ Need to generalize "minimum time to release $q$ events."
The Pseudo-Inverse of $\eta_h^+ (\Delta)$

- $\delta_i^- (n)$ ... lower bound on the minimum distance between any $n$ events.
- $\delta_i^+ (n)$ ... upper bound on the maximum distance between any $n$ events.

→ Can be automatically derived from $\eta_h^+ (\Delta)$ and $\eta_h^- (\Delta)$. 
RTA for event streams with arbitrary deadlines

The response time of events in stream $T_i$ is bounded by

$$R_i = j_i + \max\{w_i(q) \mid q = 1, 2, \ldots\},$$

where $w_i(q)$ is the smallest positive solution to the recurrence

$$w_i(q) = q \cdot e_i + \sum_{T_h \in hp_i} \eta_h^+ (w_i(q) + \delta_i^-(q)) \cdot e_h - \delta_i^-(q).$$
How to choose feasible priorities?

*RTA assumes priorities are given*...
Classic Result #1: Rate-Monotonic Priorities

Rate Monotonic (RM): $p_i < p_j \rightarrow i < j$

Rate-monotic priorities are optimal (w.r.t. FP scheduling) for independent, implicit-deadline, sporadic or periodic, fully preemptive tasks with zero jitter on a uniprocessor.\textsuperscript{LL73}

*No longer true if any of the assumptions are removed.*

Classic Result #2: Deadline-Monotonic Priorities

Deadline Monotonic (DM): $d_i < d_j \rightarrow i < j$

Deadline-monotonic priorities are **optimal** (w.r.t. FP scheduling) for independent, constrained-deadline, sporadic or periodic, fully preemptive tasks with **zero jitter** on a uniprocessor.\textsuperscript{LW82}

No longer true if any of the assumptions are removed.

What about arbitrary deadlines, jitter, and (later) blocking?

No simple heuristic is known for these cases, but optimal priorities can still be found using Lawler's technique\textsuperscript{La73} / Audsley's algorithm\textsuperscript{Au91}.

**Observation:** relative priority order of tasks in $h_{p_i}$ is irrelevant.

$$w_i(q) = q \cdot e_i + \sum_{T_h \in h_{p_i}} \eta_h^+(w_i(q) + \delta_i^-(q)) \cdot e_h - \delta_i^-(q).$$


\textsuperscript{Au91} N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.
Lawler's technique / Audsley's algorithm

Idea: assign lowest remaining priority to some task until all priorities are assigned or no more feasible assignment is possible.

Given: a set of $n$ tasks $S$
set $\text{REM} := S$ // remaining tasks without an assigned priority
for $P := n$ downto 1:
    find a task $T_i$ in $\text{REM}$ such that $\text{RTA}(T_i, hp=\text{REM}\{T_i\}) \leq T_i.\text{deadline}$
    if no such $T_i$: $\Rightarrow$ task set INFEASIBLE, abort
    $T_i.\text{prio} := P$
    remove $T_i$ from $\text{REM}$

$\text{RTA}(T_i, hp=\text{REM}\{T_i\}) \Leftarrow \text{bound } R_i \text{ assuming } h_{p_i} = \text{REM}\{T_i\}$. 
Summary FP Scheduling

• FP scheduling is **easy to implement efficiently** (w/ bitmaps).
  → The *de facto* standard supported by most RTOSs.

• Can analyze jittery, bursty, and irregular activation patterns.
  → Especially useful for system models extracted from traces.

• Can find **feasible priority assignment** if one exists.
  → Can minimize maximum tardiness if none exists.

• Can incorporate **blocking due to locks** (later in the course).

→ A mature, practical scheduling theory supported by commercial tools.
Schedulability Analysis of EDF
Recap: Exact Utilization Test\textsuperscript{LL73}

A set $\tau$ of independent, \textit{implicit-deadline}, sporadic, fully preemptive tasks with \textit{zero jitter} on a uniprocessor is schedulable \textbf{if and only if}

$$\sum_{T_i \in \tau} \frac{e_i}{p_i} \leq 1.$$  \textsuperscript{LL73} Liu, C. and Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 30:46–61.
Trivial Extension: Sufficient Density Test

A set $\tau$ of independent, **arbitrary-deadline**, sporadic, fully preemptive tasks with **zero jitter** on a uniprocessor is schedulable if

$$\sum_{T_i \in \tau} \delta_i = \sum_{T_i \in \tau} \frac{e_i}{\min(p_i, d_i)} \leq 1.$$
Exact EDF Schedulability = Uniprocessor Feasibility

Recall: **EDF is optimal** on a uniprocessor for independent, fully preemptive jobs.

Therefore, an exact **schedulability** test for EDF is also a general **feasibility** test for preemptive uniprocessor real-time scheduling.

→ Baruah's *processor demand analysis*\(^{\text{BRH90}}\) is the first such test.

Recap: EDF Busy Window

A deadline miss is always preceded by a busy window.

![Diagram showing CPU idle or executing jobs with deadlines past $t_2$ during the busy window from $t_1$ to $t_2$.]

**Key property:** only jobs released on or after $t_1$ with deadlines no later than $t_2$ execute during the busy window.
Processor Demand Analysis: Basic Idea

Recall: to miss a deadline, the total processor demand $C(t_1, t_2)$ during a busy window $(t_1, t_2]$ must exceed $t_2 - t_1$.

**Basic idea:** if $t_2 - t_1 \geq C(t_1, t_2)$ for all possible busy windows $(t_1, t_2]$, then no deadline will be missed.
Demand Bound Function $dbf(T_i, \Delta)$

The demand bound function $dbf(T_i, \Delta)$ of a task $T_i$ is

- an **upper bound** on the cumulative execution requirement of the jobs of $T_i$
- released during any interval $(t, t + \Delta]$ with deadlines no later than $t + \Delta$.

For a sporadic task $T_i$ with jitter $j_i$:

$$dbf(T_i, \Delta) = \max\left(0, \left[\frac{j_i + \Delta + p_i - d_i}{p_i}\right] \cdot e_i\right)$$
Bounding the Total Demand

By definition of $dbf(T_i, \Delta)$, we have for any busy window $(t_1, t_2]$:

$$C(t_1, t_2) \leq \sum_{T_i \in \tau} dbf(T_i, t_2 - t_1).$$
Processor Demand Analysis$^{BRH90}$

Let $L$ denote a bound on the maximum busy-window length.

A task set $\tau$ is schedulable under EDF if and only if $u_{\text{sum}}(\tau) \leq 1$ and

$$\forall t, \ 0 \leq t \leq L : \sum_{T_i \in \tau} dbf(T_i, t) \leq t.$$ 

$\rightarrow$ How to determine $L$?

Bounding the Maximum Relevant Busy-Window Length $L$

We seek an $L$ such that,

- if $\sum_{T_i \in \tau} dbf(T_i, t) > t$ for any $t$, then $\exists t' \leq L : \sum_{T_i \in \tau} dbf(T_i, t') > t'$.

- For sporadic task sets

  1. $L \leq H(\tau)$, where $H(\tau) = \text{lcm}\{p_i \mid T_i \in \tau\}$, and

  2. $L \leq L^* \text{ if } 1 - u_{\text{sum}}(\tau) > 0$, where $L^* = \frac{\sum_{i=1}^{n} (p_i - \min(d_i, p_i)) \cdot u_i}{1 - u_{\text{sum}}(\tau)}$. 
Processor Demand Analysis Notes

• $dbf(T_i, \Delta)$ is a step function $\rightarrow$ a point $t$ must be checked only if some $dbf(T_i, t)$ "steps," which it does when $\eta_i^+(t - d_i)$ "steps".

• In general, processor demand analysis is coNP-hard.\(^{\text{ER10}}\)

• Quick Processor demand Analysis\(^{\text{ZB09}}\) (QPA) implements the same check, but is faster since it manages to skip many test points and uses tighter bounds on $L$.


Bounding Response Times under EDF
Bounding Slack instead of Response Times

**Observation:** the maximum response time $r_i$ and the minimum slack $\text{slack}_i$ are duals of each other.

$$r_i = d_i - \text{slack}_i$$

Let $S_i \leq \text{slack}_i$. Then a response-time bound is given by $R_i = d_i - S_i$.

Turns out it's easier to bound the minimum slack.$^{\text{GY14}}$

---

A Slack Bound under EDF$^{GY14}$

$$S_i = \min \left\{ \Delta - \sum_{k=1}^{n} dbf(T_k, \Delta) \left| d_i \leq \Delta \leq L + \max_k \{d_k\} \right. \right\}$$

**Intuition:** suppose $T_i$ has a deadline at the end of an interval $(t, t + \Delta]$. Then at most $\sum_{k=1}^{n} dbf(T_k, \Delta)$ processor service is required, where as $\Delta$ time units of service are available.

$\rightarrow$ The "gap" between $\Delta$ and the busy-window length lower-bounds slack.

---

Properties of the Slack Bound $S_i$

- $S_i$ may be **negative**, in which case jobs can be **tardy**.

- If $S_i < 0$, then the bound is **tight** (i.e., there exists a valid schedule in which $S_i$ tardiness is incurred by some $J_i$).\(^{GY14}\)

- Otherwise, if $S_i \geq 0$, then the bound may be **pessimistic** (but the deadline is met anyway, so the pessimism may be acceptable).

Goal: Tightening the Slack Bound if $S_i \geq 0$

In the following, it is assumed that $\forall \Delta > d_i : dbf(T_k, \Delta) \leq \Delta$.

→ If not, then the simpler bound is already tight.

Definition: the request bound function $rbf(T_i, \Delta)$ denotes the maximum cumulative processor demand of jobs of $T_i$ released during any interval of length $\Delta$ and is given by:

$$rbf(T_i, \Delta) = \eta_i^+(\Delta) \cdot e_i.$$
Relation of $dbf(T_i, \Delta)$, $\eta_i^+(\Delta)$, and $rbf(T_i, \Delta)$

Recall: $\eta_i^+(\Delta) = \left[ \frac{j_i + \Delta}{p_i} \right] \cdot e_i$.

$rbf(T_i, \Delta) = \eta_i^+(\Delta) \cdot e_i$.

$dbf(T_i, \Delta) = \max \left( 0, \left[ \frac{j_i + \Delta + p_i - d_i}{p_i} \right] \cdot e_i \right) = \max \left( 0, \left[ \frac{j_i + (\Delta - d_i)}{p_i} \right] \cdot e_i \right)$

$\rightarrow$

$dbf(T_i, \Delta) = \max(0, \eta_i^+(\Delta - d_i) \cdot e_i) = \begin{cases} 0 & \text{if } \Delta < d_i \\ rbf(T_i, \Delta - d_i) & \text{otherwise} \end{cases}$
Definition: Mixed Bound function

For any $\Delta \geq \gamma \geq 0$, let

$$mbf(T_i, \Delta, \gamma) = \min(dbf(T_i, \Delta), rbf(T_i, \gamma)).$$

**Intuition:** within a busy window of length at most $\Delta$, consider only jobs released during the first $\gamma$ time units.
A Tight Slack Bound$^{GY14}$

\[ S_i = \min_{d_i \leq \Delta \leq L + \max_k \{d_k\}} \max \left\{ \Delta - \gamma \ \middle| \ \gamma \leq \Delta \ \land \ \sum_{k=1}^{n} mbf(T_k, \Delta, \gamma) \leq \sum_{k=1}^{n} sbf(T_k, \gamma) \right\} \]

For a proof see Guan and Yi$^{GY14}$.