

In Search of Butterflies: Exceedance Analysis for Real-Time Systems under Transient Overload

M. Zini¹, F. Marković², D. Casini¹, A. Biondi¹, and B. Brandenburg²

¹*Scuola Superiore Sant'Anna, Pisa, Italy*

²*Max Planck Institute for Software Systems, Kaiserslautern, Germany*

45th IEEE Real-Time Systems Symposium, York, UK - 12/12/2024



MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS





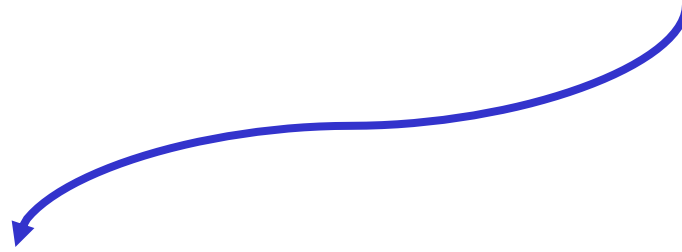
- The theoretical analysis of real-time systems often relies on the concept of **worst-case execution time** (WCET).

WCET can be bounded with:



- The theoretical analysis of real-time systems often relies on the concept of **worst-case execution time** (WCET).

WCET can be bounded with:



Analytical methods





- The theoretical analysis of real-time systems often relies on the concept of **worst-case execution time** (WCET).

WCET can be bounded with:

Analytical methods



Empirical methods





Analytical methods





Analytical methods



PROBLEM

High complexity of modern hardware and software stacks.

Example:

Linux-based solutions in critical systems:

- Unmanned aerial vehicles
- Autonomous driving (e.g., Tesla)
- Spacecrafts (e.g., SpaceX)





Empirical methods





WCET is estimated with an experimental approach

Empirical methods





Empirical methods



WCET is estimated with an experimental approach



The bound is not provably safe!!!



Empirical methods



WCET is estimated with an experimental approach



The bound is not provably safe!!!

In practice, we are not dealing with a **worst-case execution time**, but rather with a

Nominal Execution Time (NET)

Nominal Execution Time (NET)



In practice, we are not dealing with a **worst-case execution time**, but rather with a

Nominal Execution Time (NET)



In practice, we are not dealing with a **worst-case execution time**, but rather with a

Nominal Execution Time (NET)

NETs can be **exceeded at run-time** due to many factors:

- Unaccounted interference / microarchitectural effects
- Intentionally under-provisioned systems
 - The WATERS 2017 challenge's task set is unschedulable with WCETs
 - E.g., NET = 99th percentile of observed execution times
- ...

Nominal Execution Time (NET)



NETs cannot be trusted!

QUESTION

What happens if jobs **exceed** their NET at runtime?



NETs cannot be trusted!

QUESTION

What happens if jobs **exceed** their NET at runtime?



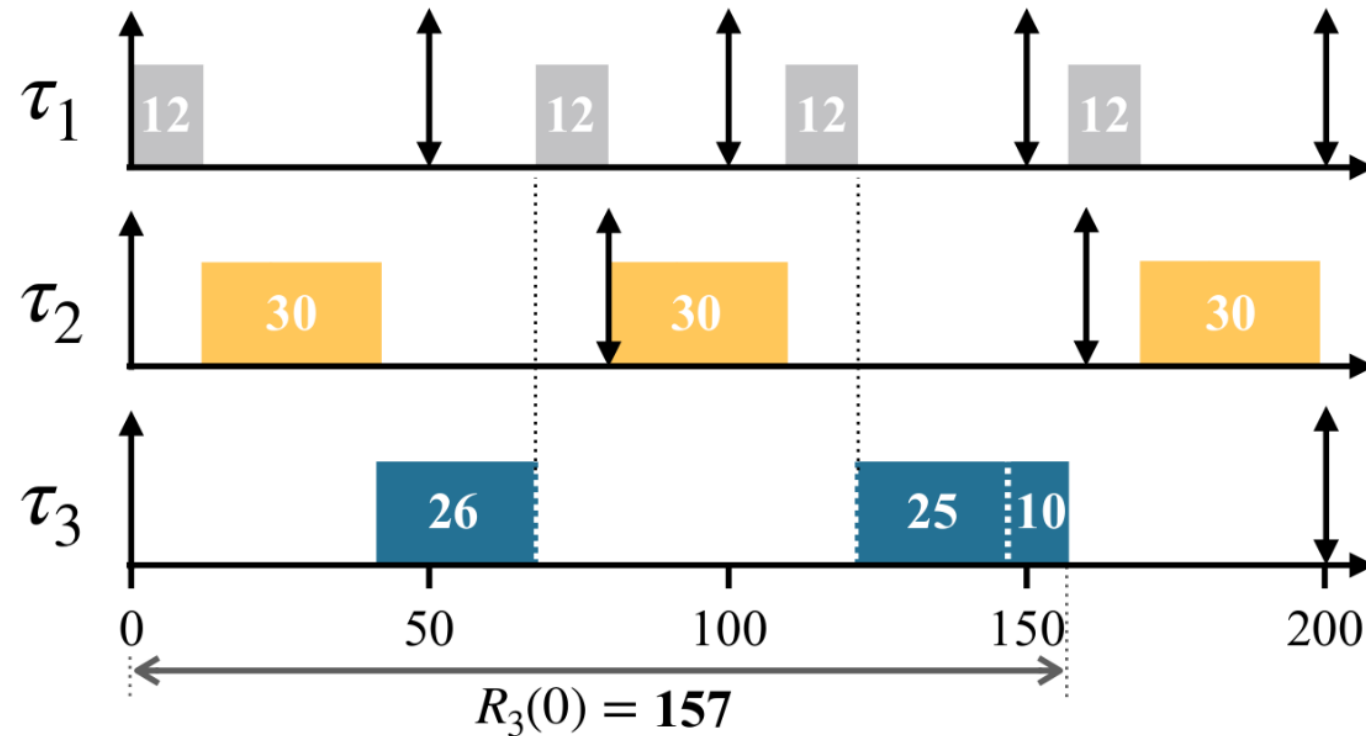
We refer to the extra execution time as **exceedance**

Motivating Example



For example, consider this simple limited-preemptive taskset:

Task	Period	NET
τ_1	50	<12>
τ_2	80	<30>
τ_3	200	<26,25,10>

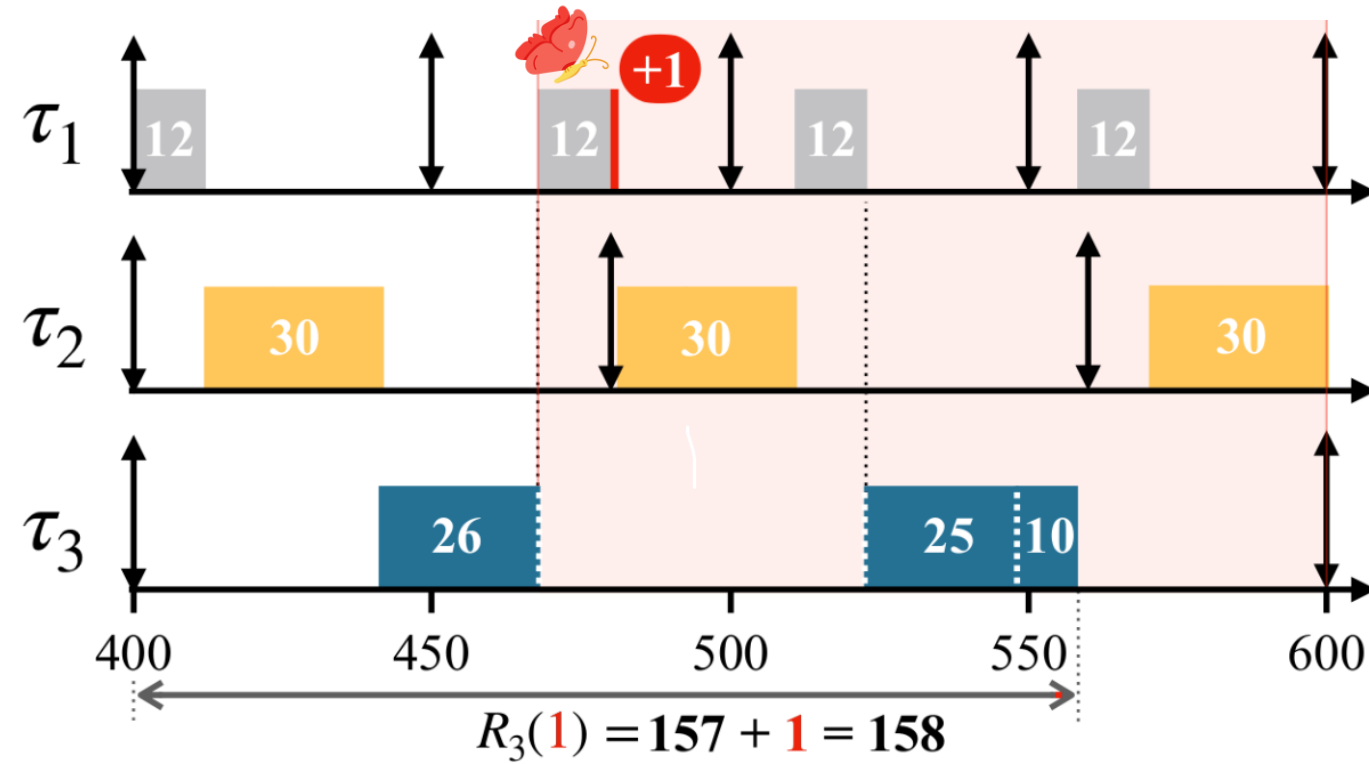


Motivating Example



We **add 1 unit of exceedance** to the second job of task τ_1

Task	Period	NET
τ_1	50	<12>
τ_2	80	<30>
τ_3	200	<26,25,10>

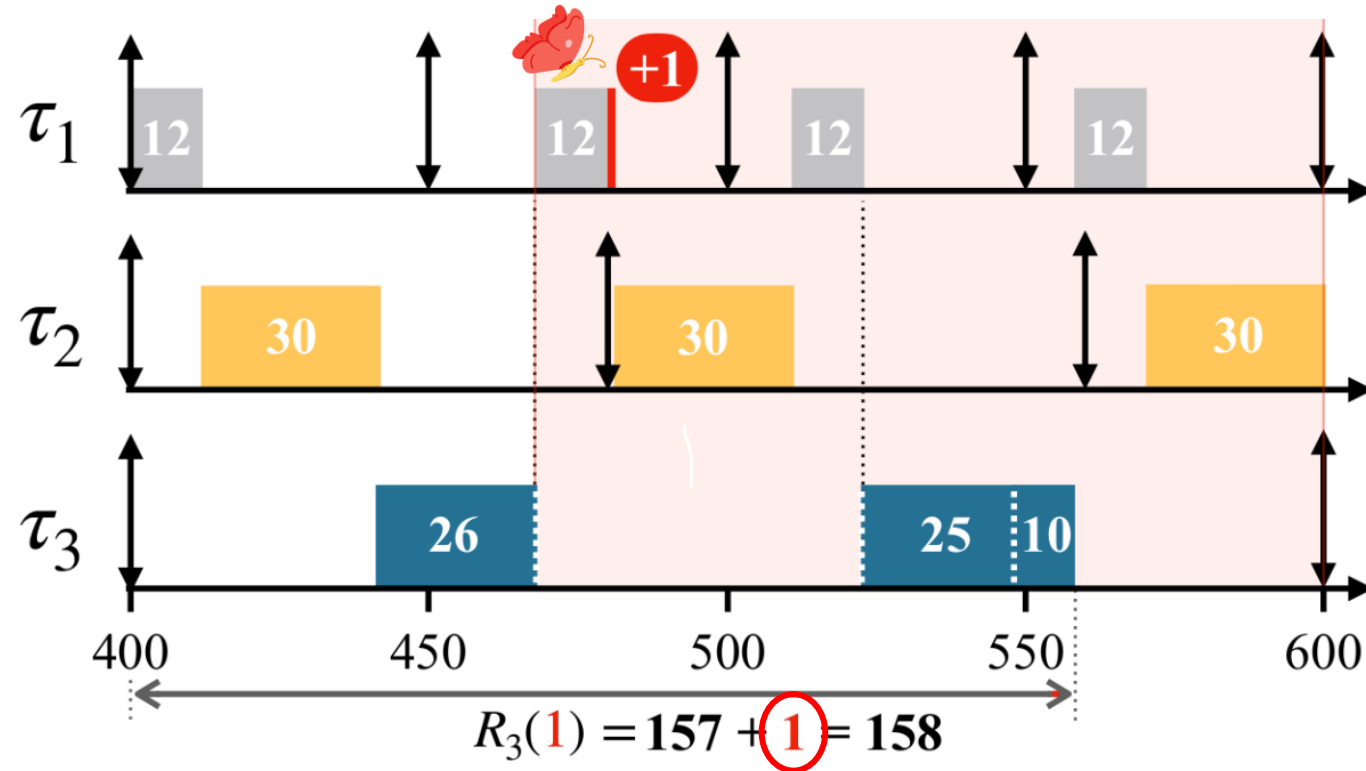


Motivating Example



We **add 1 unit of exceedance** to the second job of task τ_1

Task	Period	NET
τ_1	50	<12>
τ_2	80	<30>
τ_3	200	<26,25,10>



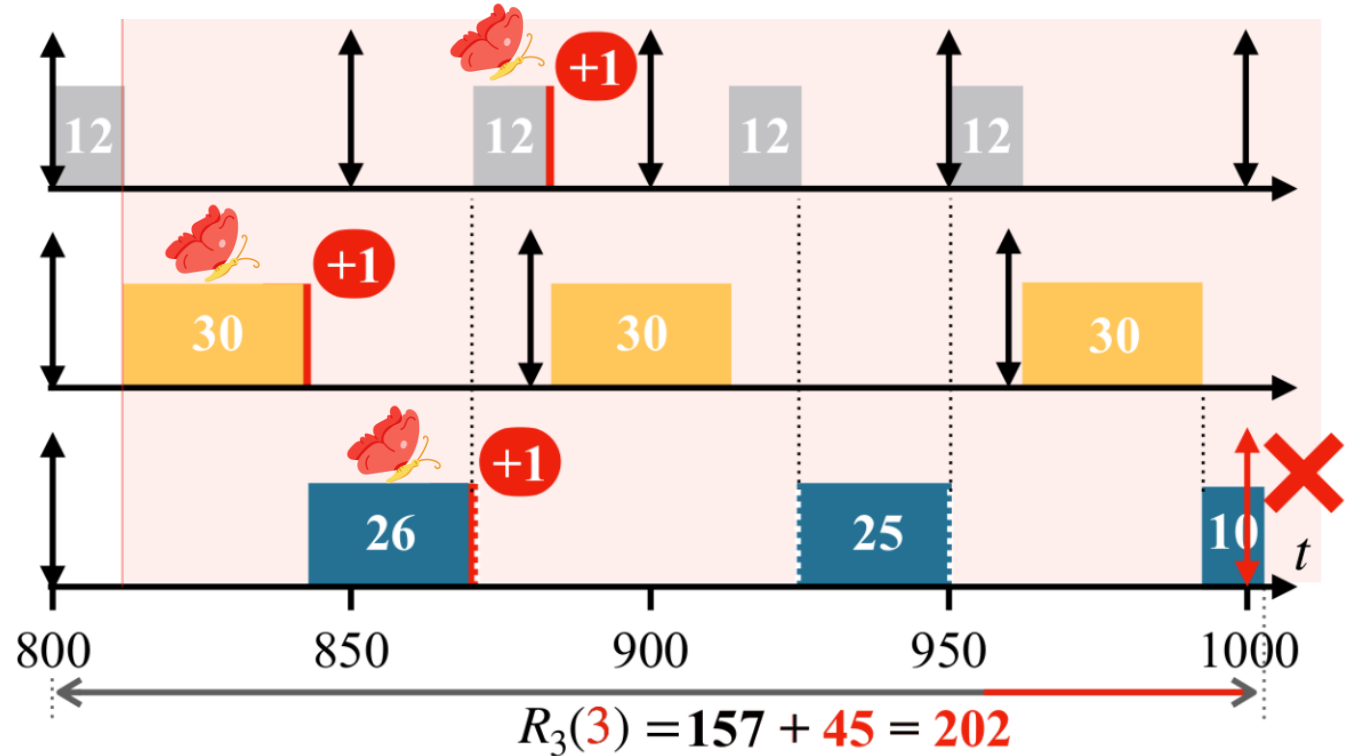
τ_3 's response time increased by 1 time unit

Motivating Example



We add 1 unit of exceedance to the first job of task τ_2 and τ_3

Task	Period	NET
τ_1	50	<12>
τ_2	80	<30>
τ_3	200	<26,25,10>

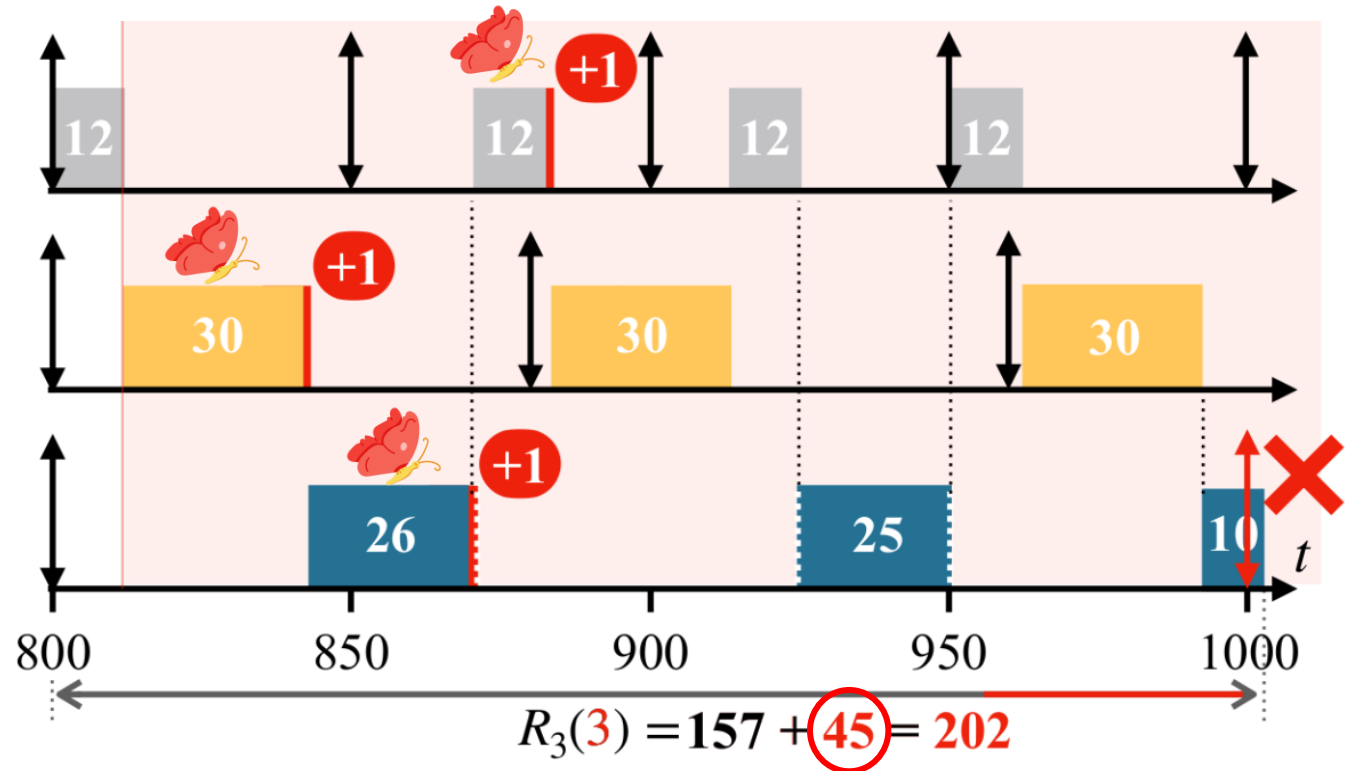


Motivating Example



We add 1 unit of exceedance to the first job of task τ_2 and τ_3

Task	Period	NET
τ_1	50	<12>
τ_2	80	<30>
τ_3	200	<26,25,10>



τ_3 's response time increased by 45 time units!!!



The consequences of **NET exceedance** are not easy to predict:

- NET + 1 \implies Response time + 1
- NET + 2 \implies Response time + 2
- NET + 3 \implies Response time + **45**
- ...



The consequences of **NET exceedance** are not easy to predict:

- NET + 1 \implies Response time + 1

- NET + 2 \implies Response time + 2

- NET + 3 \implies Response time + 45



Nonlinear increase!

- ...



The consequences of **NET exceedance** are not easy to predict:

■ NET + 1 \implies Response time + 1

■ NET + 2 \implies Response time + 2

■ NET + 3 \implies Response time + 45

■ ...

Nonlinear increase!

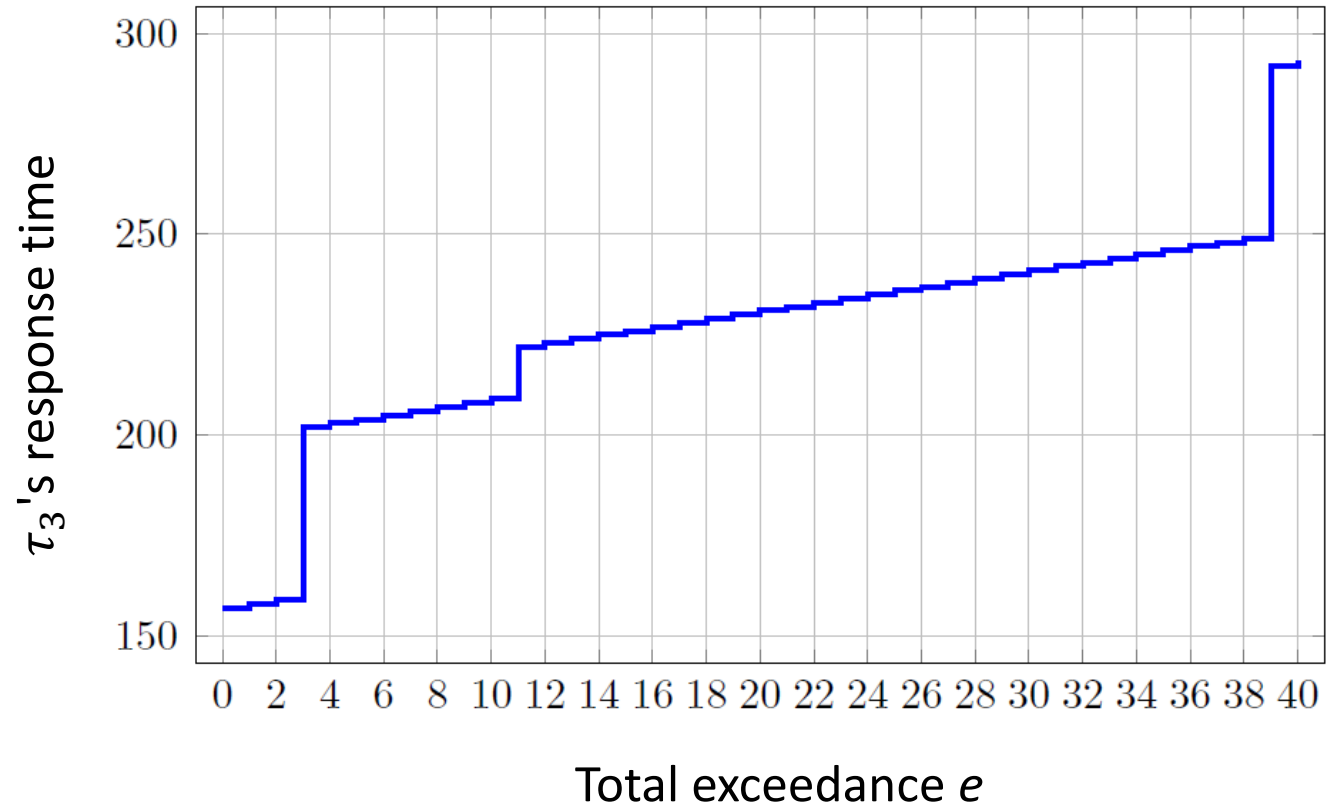


If we neglect this phenomenon, we might **over-estimate the system's temporal safety margin**



But response-time nonlinearities are difficult to predict

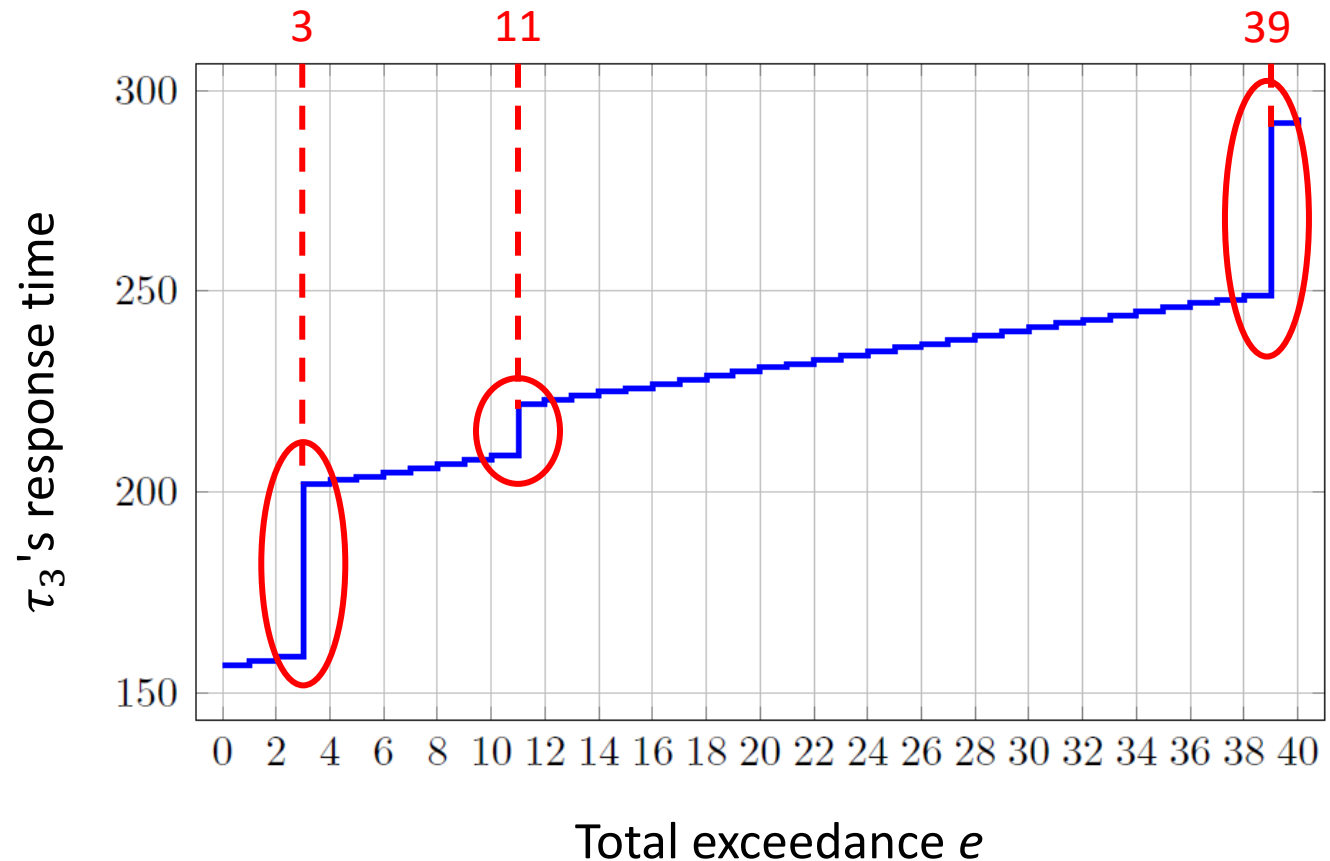
Task	Period	NET
τ_1	50	<12>
τ_2	80	<10, 20>
τ_3	200	<26,25,10>





But response-time nonlinearities are difficult to predict

Task	Period	NET
τ_1	50	<12>
τ_2	80	<10, 20>
τ_3	200	<26,25,10>





We understood that **nonlinearities** are:

- Dangerous for schedulability
- Difficult to predict



We understand that **nonlinearities** are:

- Dangerous for schedulability
- Difficult to predict



We need a strategy to understand
how much the tasks can exceed the NETs
before generating a nonlinearity



Is It Sensitivity Analysis?



A similar problem has already been faced in literature with **sensitivity analysis**.

But sensitivity analysis:

Is It Sensitivity Analysis?



A similar problem has already been faced in literature with **sensitivity analysis**.

But sensitivity analysis:

Lacks support for:

- limited-preemptive
- fully non-preemptive

Is It Sensitivity Analysis?



A similar problem has already been faced in literature with **sensitivity analysis**.

But sensitivity analysis:

Lacks support for:

- limited-preemptive
- fully non-preemptive

Does not consider:

- Arbitrary arrival curves
- Multiple scheduling policies
- Arbitrary deadlines for FP

Is It Sensitivity Analysis?



A similar problem has already been faced in literature with **sensitivity analysis**.

But sensitivity analysis:

Lacks support for:

- limited-preemptive
- fully non-preemptive

Does not consider:

- Arbitrary arrival curves
- Multiple scheduling policies
- Arbitrary deadlines for FP

Yields a scaling factor /
additive constant



Considers permanent
overload

Is It Sensitivity Analysis?



A similar problem has already been faced in literature with **sensitivity analysis**.

But sensitivity analysis:

Lacks support for:

- limited-preemptive
- fully non-preemptive

Does not consider:

- Arbitrary arrival curves
- Multiple scheduling policies
- Arbitrary deadlines for FP

Yields a scaling factor /
additive constant



Considers permanent
overload

Gives no information
on nonlinearities
before or after the
deadlines



A similar problem has already been faced in literature with **sensitivity analysis**.

But sensitivity analysis:

Lacks support for:

- limited-preemptive
- fully non-preemptive

Does not consider:

- Arbitrary arrival curves
- Multiple scheduling policies
- Arbitrary deadlines for FP

Yields a scaling factor /
additive constant



Considers permanent
overload

Gives no information
on nonlinearities
before or after the
deadlines

We also aim to provide **explainability**



Our solution is based on the **abstract Response-Time Analysis** framework^[1] by Bozhko et al.

[1] S. Bozhko and B. B. Brandenburg. Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle, *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*



Our solution is based on the **abstract Response-Time Analysis** framework^[1] by Bozhko et al.



Provides a unified and general *abstract response-time analysis*, independent of specific scheduling policies, workload models, and preemption policies

[1] S. Bozhko and B. B. Brandenburg. Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle, *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*



Our solution is based on the **abstract Response-Time Analysis** framework^[1] by Bozhko et al.



Provides a unified and general *abstract response-time analysis*, independent of specific scheduling policies, workload models, and preemption policies

We extended the framework to support the **presence of exceedance** and provided concrete instantiations for **3 scheduling policies and 4 preemption models**:

[1] S. Bozhko and B. B. Brandenburg. Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle, *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*



Our solution is based on the **abstract Response-Time Analysis** framework^[1] by Bozhko et al.



Provides a unified and general *abstract response-time analysis*, independent of specific scheduling policies, workload models, and preemption policies

We extended the framework to support the **presence of exceedance** and provided concrete instantiations for **3 scheduling policies and 4 preemption models**:

- Fixed priority
- EDF
- FIFO
- Fully preemptive
- Fully non-preemptive
- Segmented non-preemptive
- Floating non-preemptive

[1] S. Bozhko and B. B. Brandenburg. Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle, *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*

Search for Nonlinearities



The function $R_i(e)$ yields the response time of task τ_i with exceedance e .



The function $R_i(e)$ yields the response time of task τ_i with exceedance e .



The "jumps" of this function are the **nonlinearities** we are looking for!



The function $R_i(e)$ yields the response time of task τ_i with exceedance e .



The **"jumps"** of this function are the **nonlinearities** we are looking for!

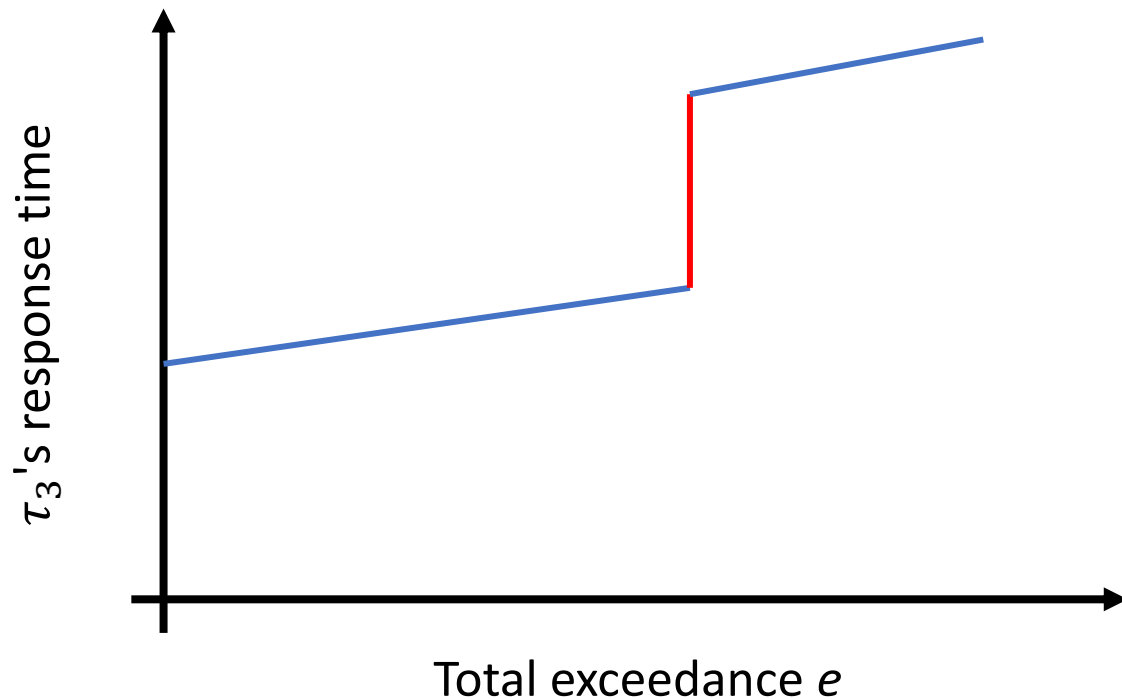


We developed an algorithm to
find such "jumps" efficiently



The search algorithm is based on:

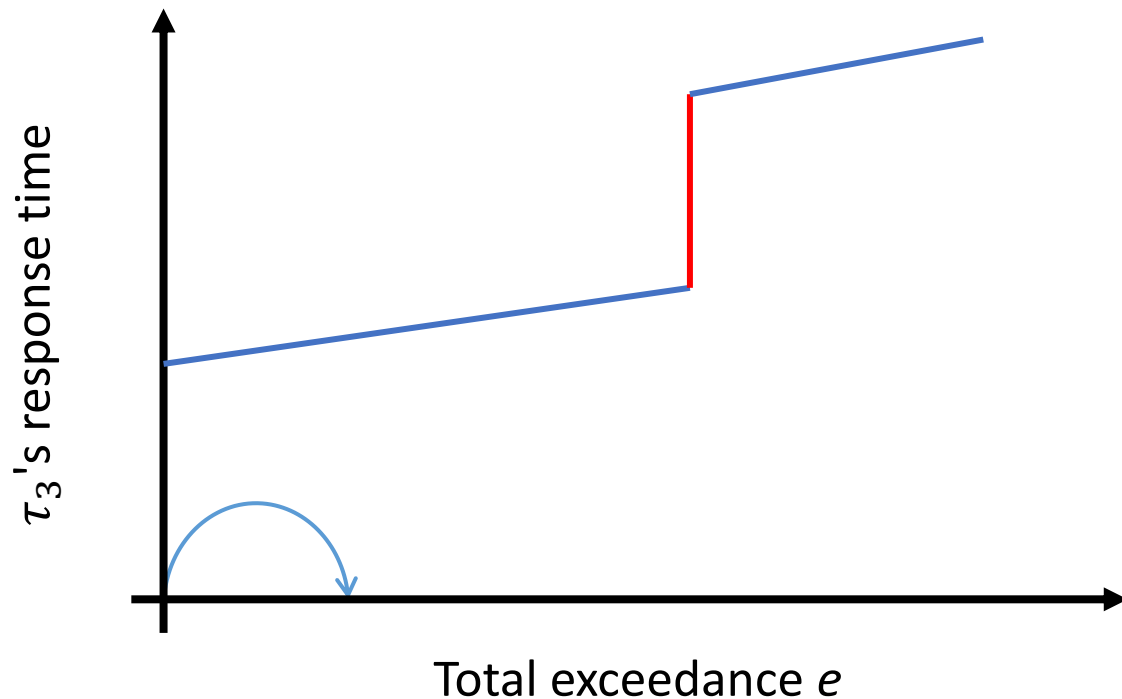
1. An exponential search of the **nonlinearity interval**





The search algorithm is based on:

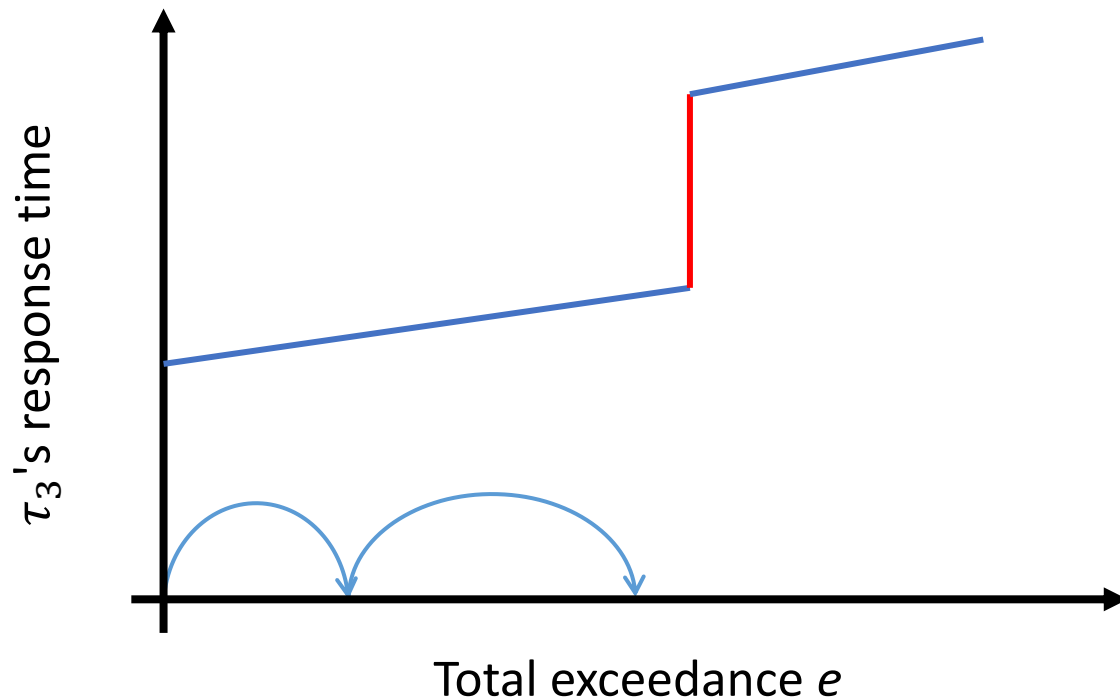
1. An exponential search of the **nonlinearity interval**





The search algorithm is based on:

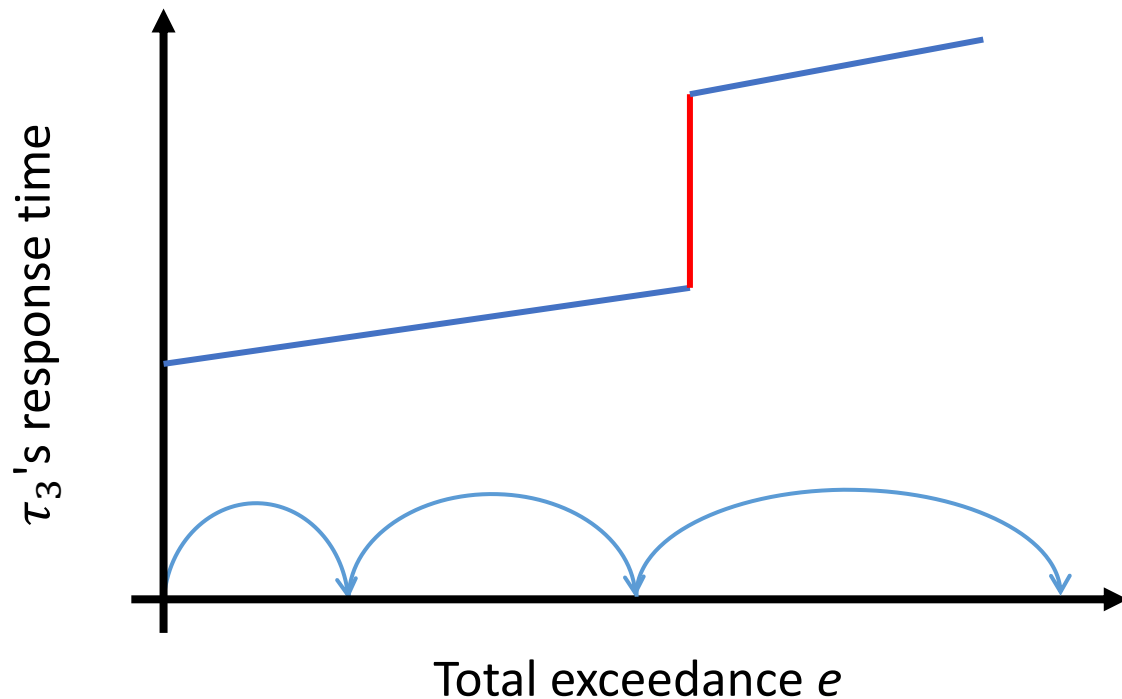
1. An exponential search of the **nonlinearity interval**





The search algorithm is based on:

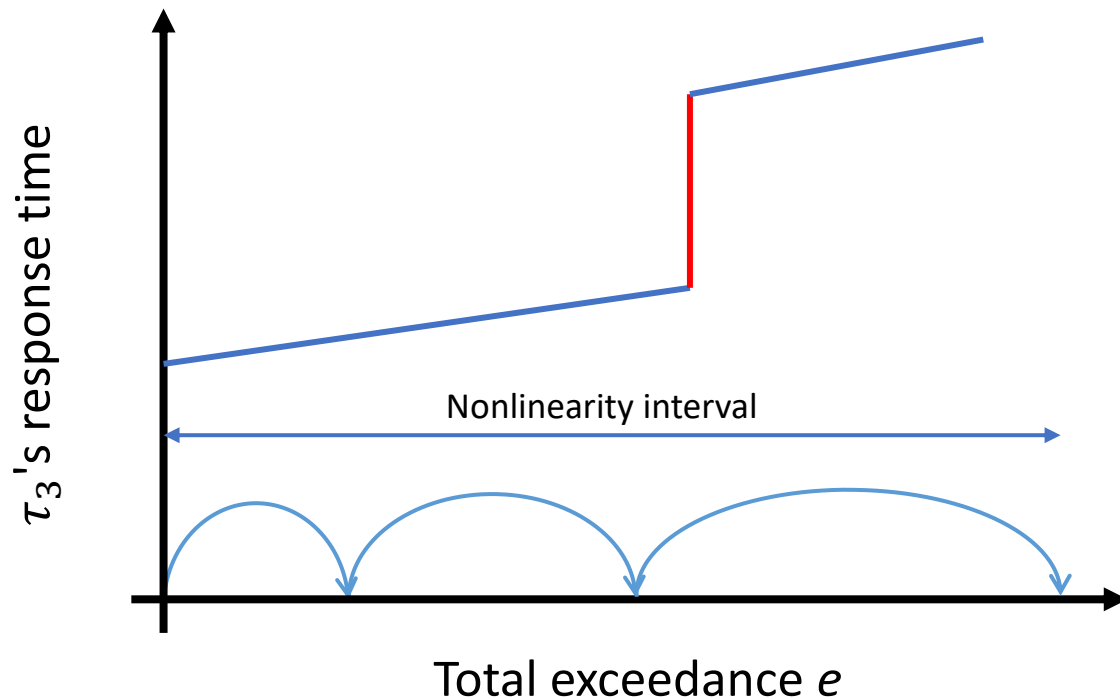
1. An exponential search of the **nonlinearity interval**





The search algorithm is based on:

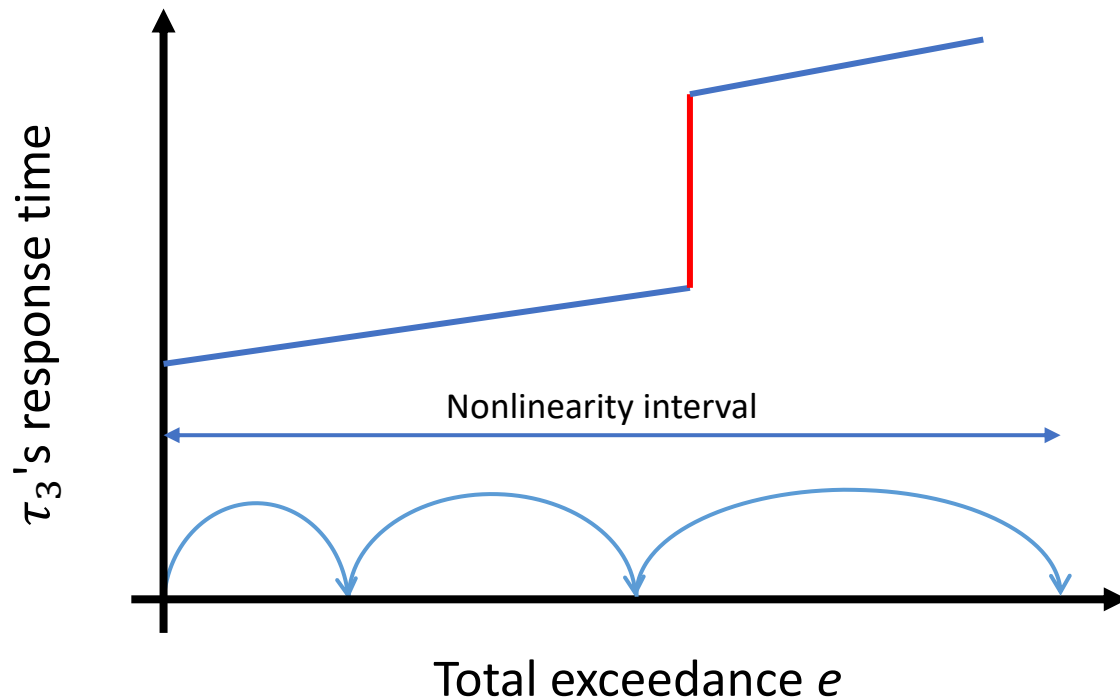
1. An exponential search of the **nonlinearity interval**



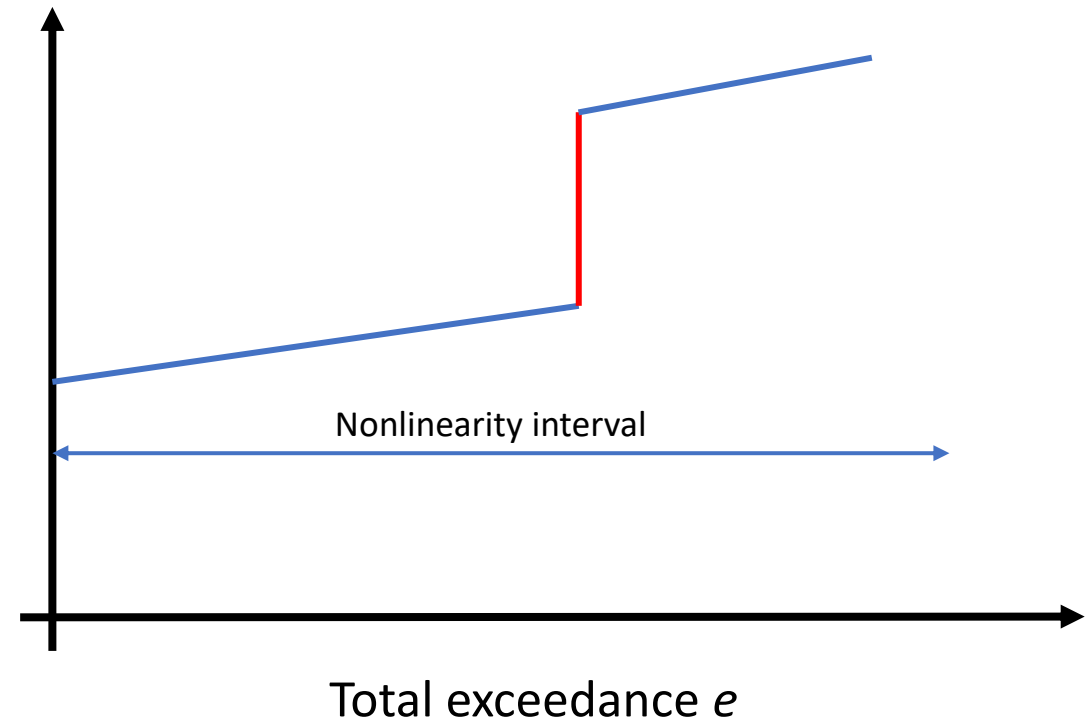


The search algorithm is based on:

1. An exponential search of the **nonlinearity interval**



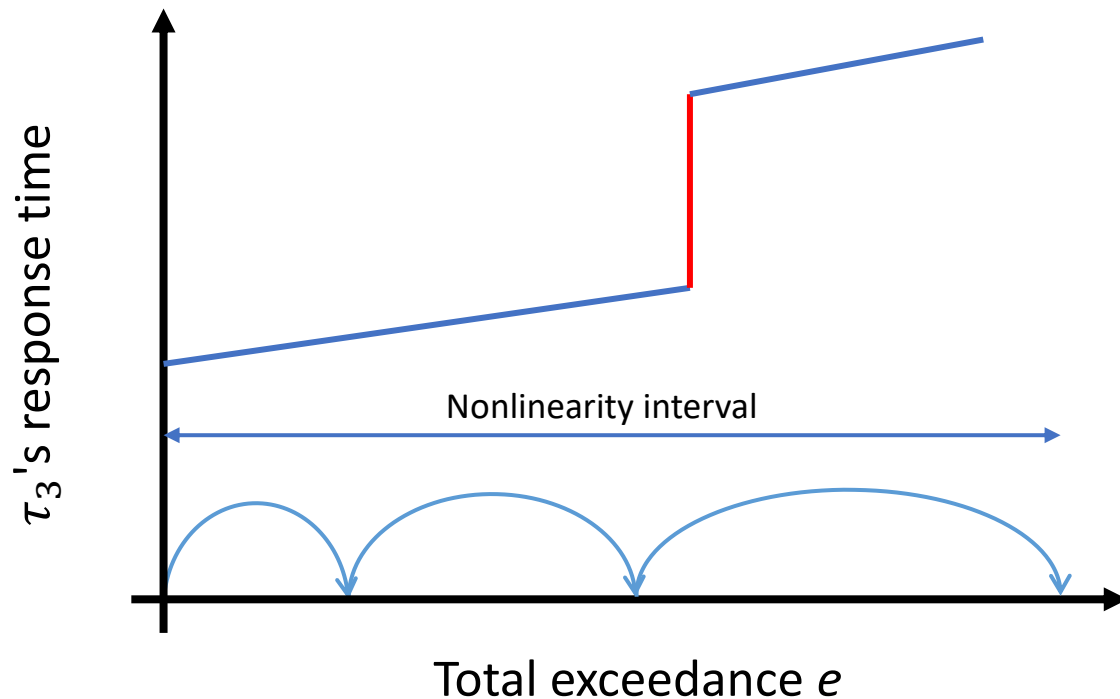
2. A binary search on the nonlinearity interval



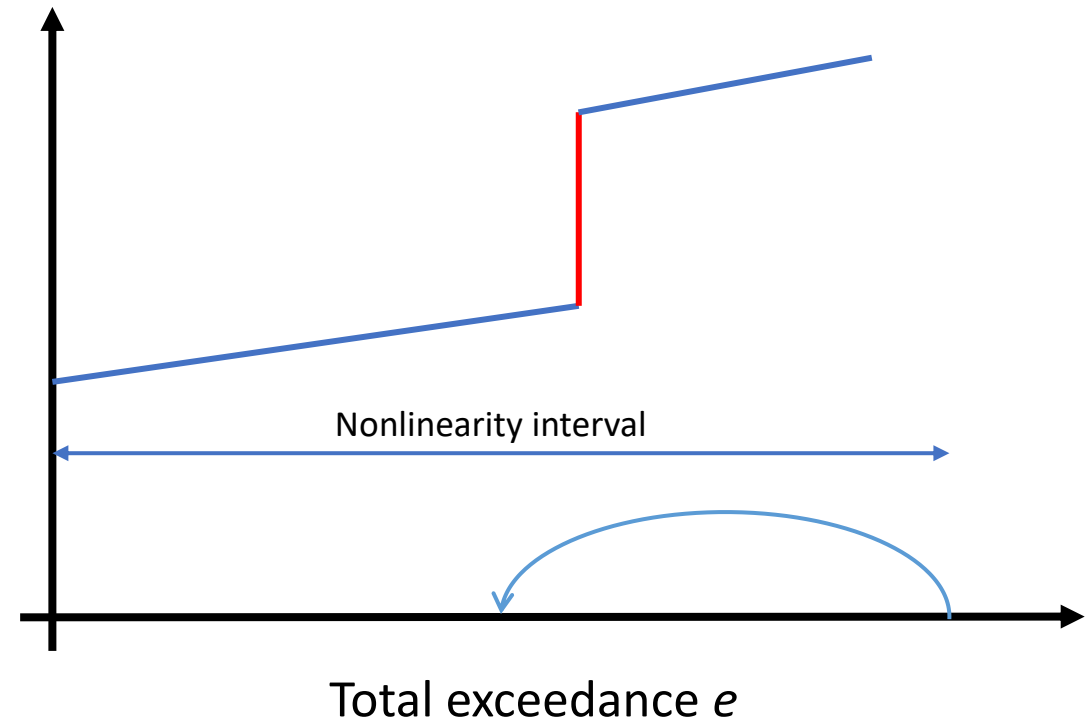


The search algorithm is based on:

1. An exponential search of the **nonlinearity interval**



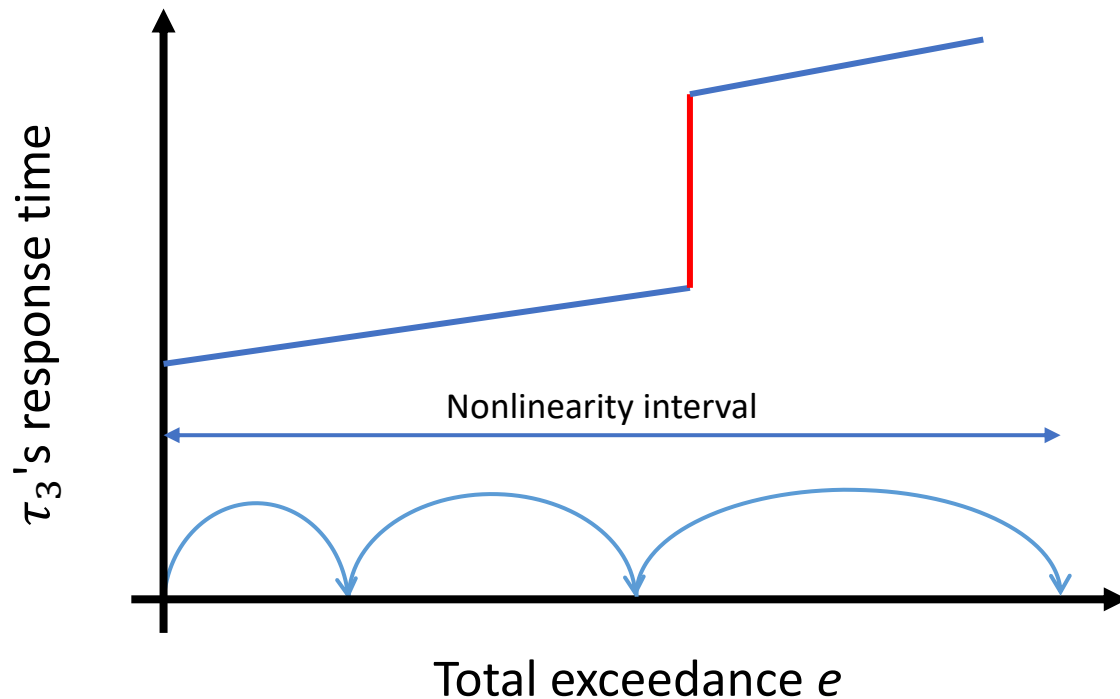
2. A binary search on the nonlinearity interval



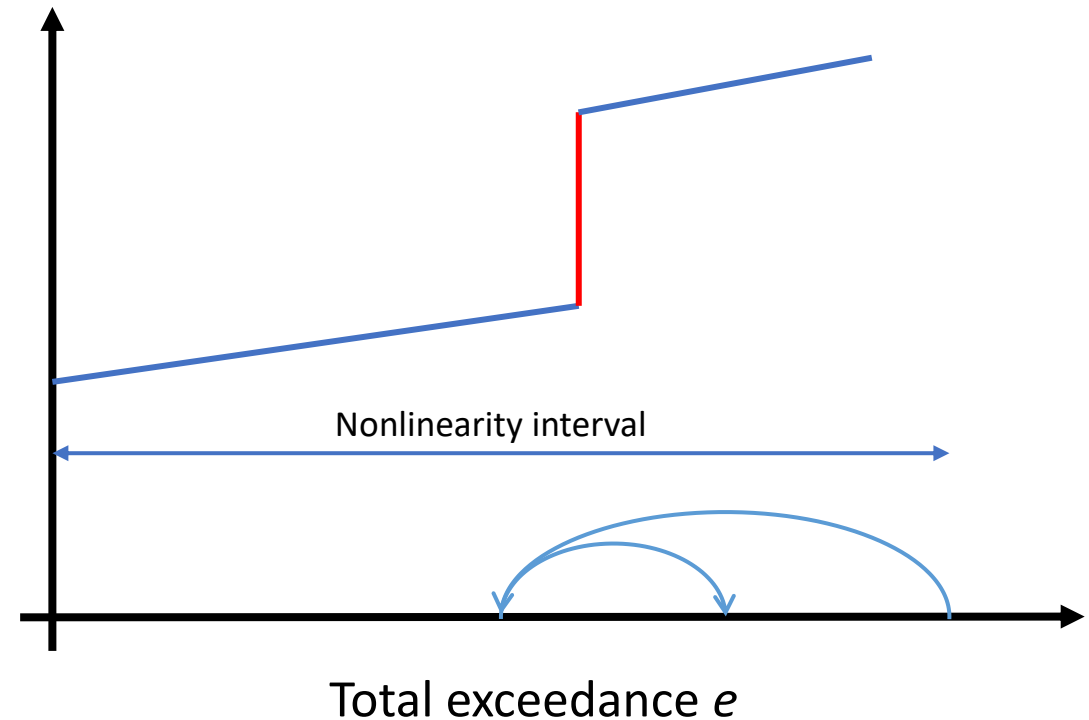


The search algorithm is based on:

1. An exponential search of the **nonlinearity interval**



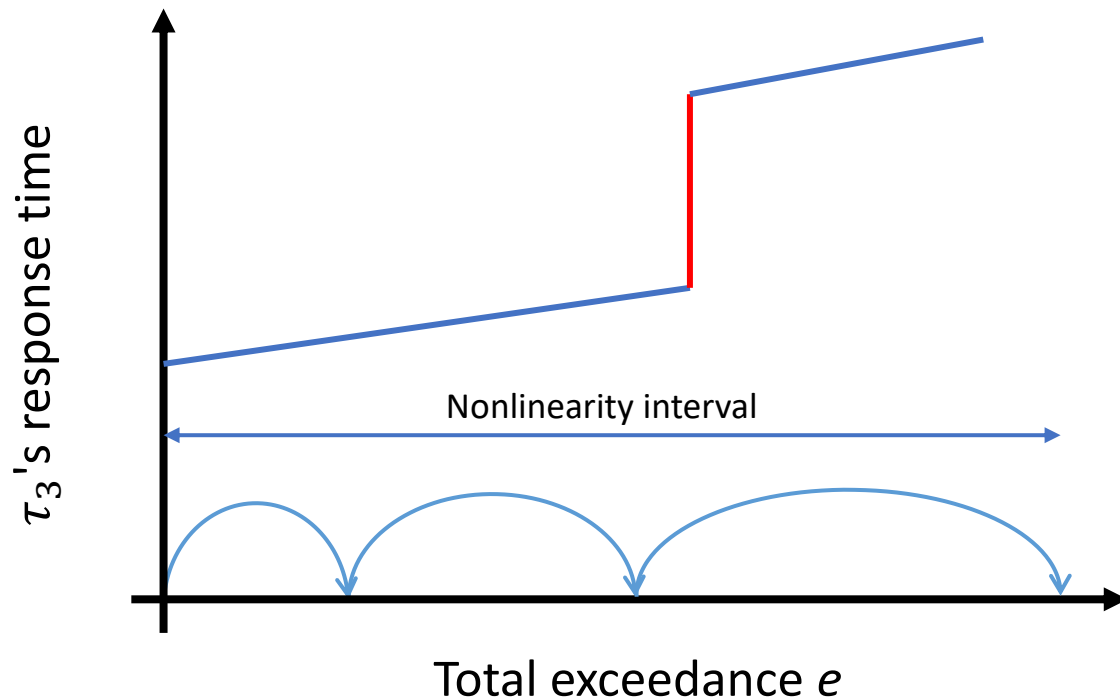
2. A binary search on the nonlinearity interval



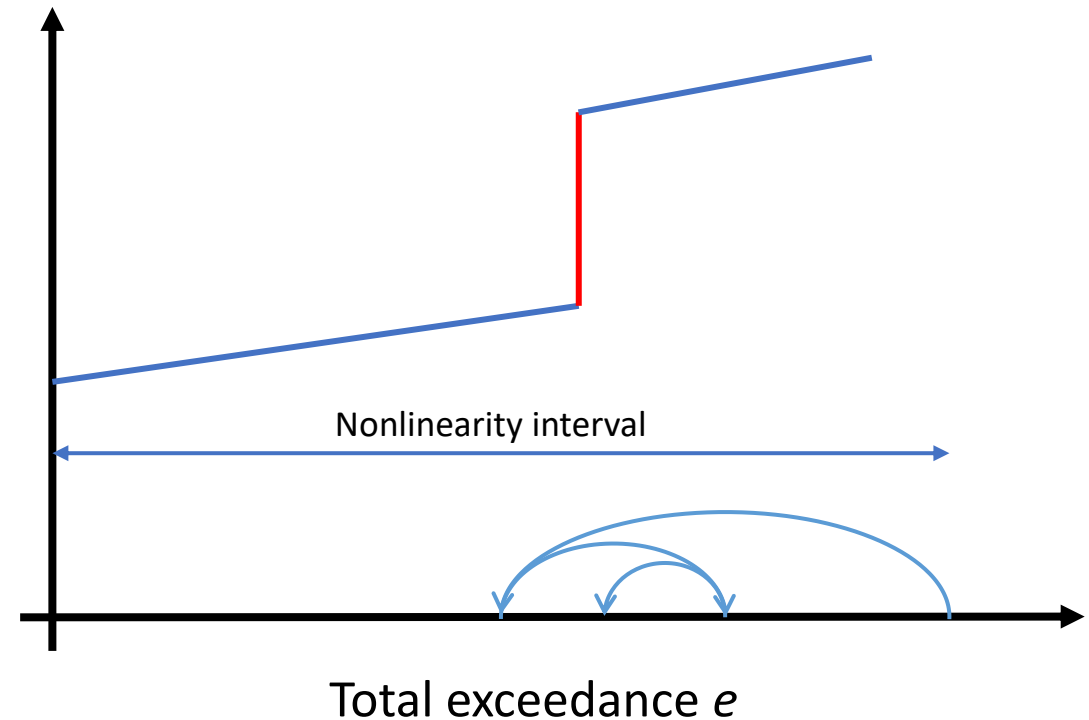


The search algorithm is based on:

1. An exponential search of the **nonlinearity interval**



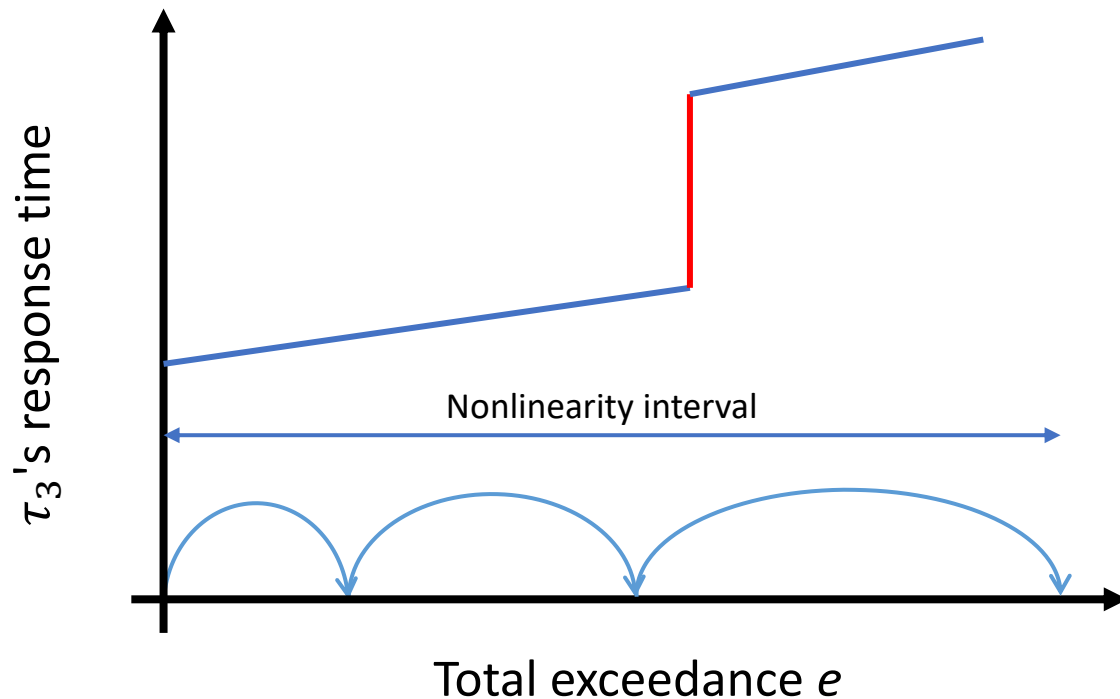
2. A binary search on the nonlinearity interval



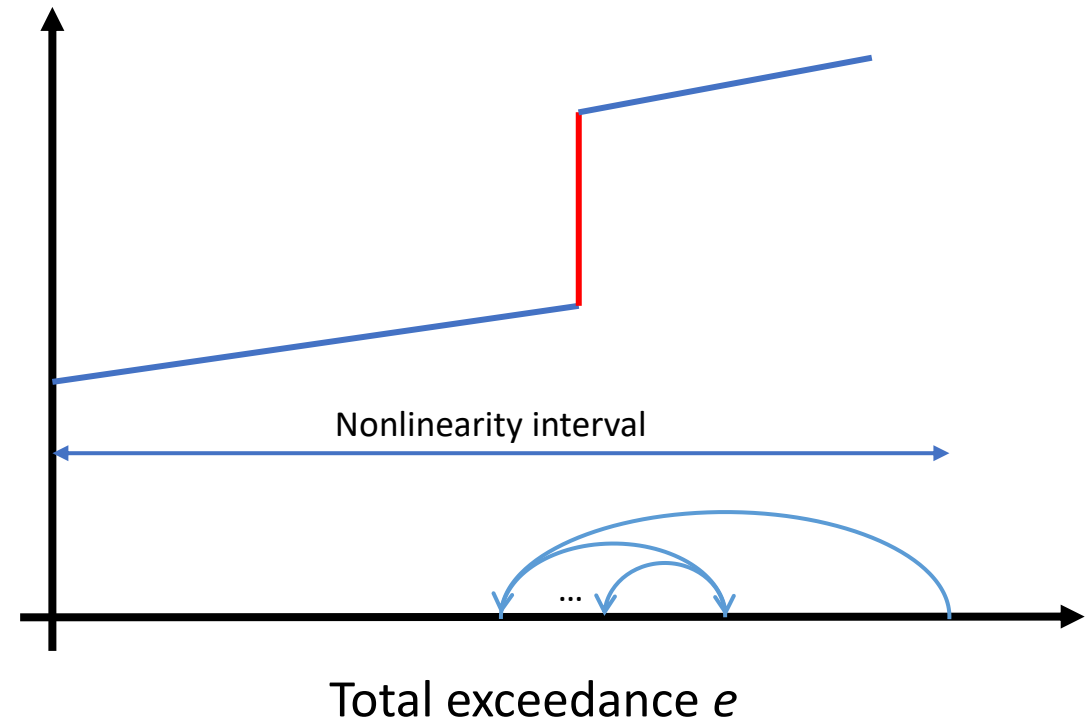


The search algorithm is based on:

1. An exponential search of the **nonlinearity interval**



2. A binary search on the nonlinearity interval



Who Generates the Exceedance?



Once we know the amount of exceedance that produces a nonlinearity, we still have **many scenarios** that can produce such an amount of exceedance:



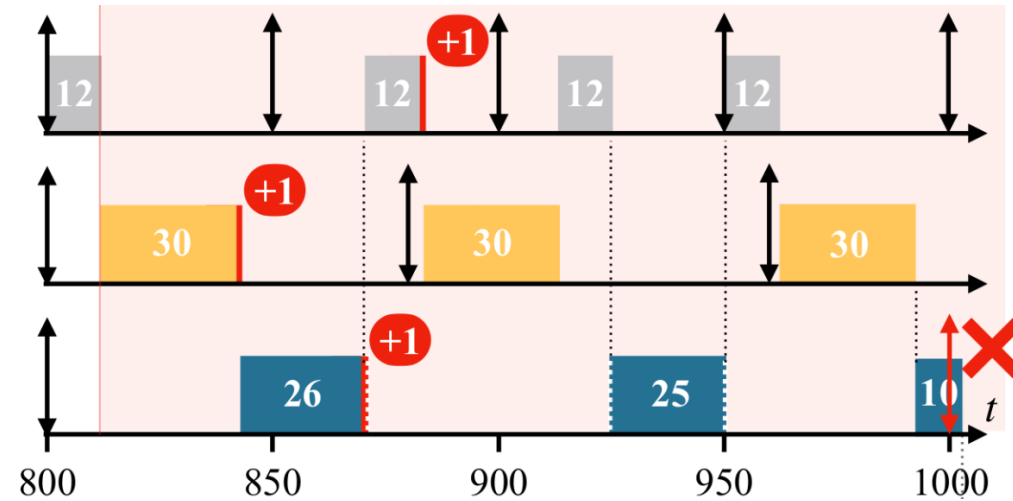
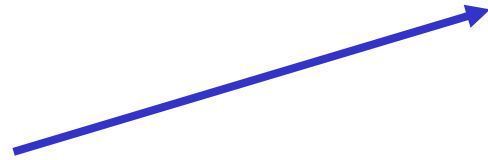
Once we know the amount of exceedance that produces a nonlinearity, we still have **many scenarios** that can produce such an amount of exceedance:

Exceedance:
3 time units



Once we know the amount of exceedance that produces a nonlinearity, we still have **many scenarios** that can produce such an amount of exceedance:

Exceedance:
3 time units

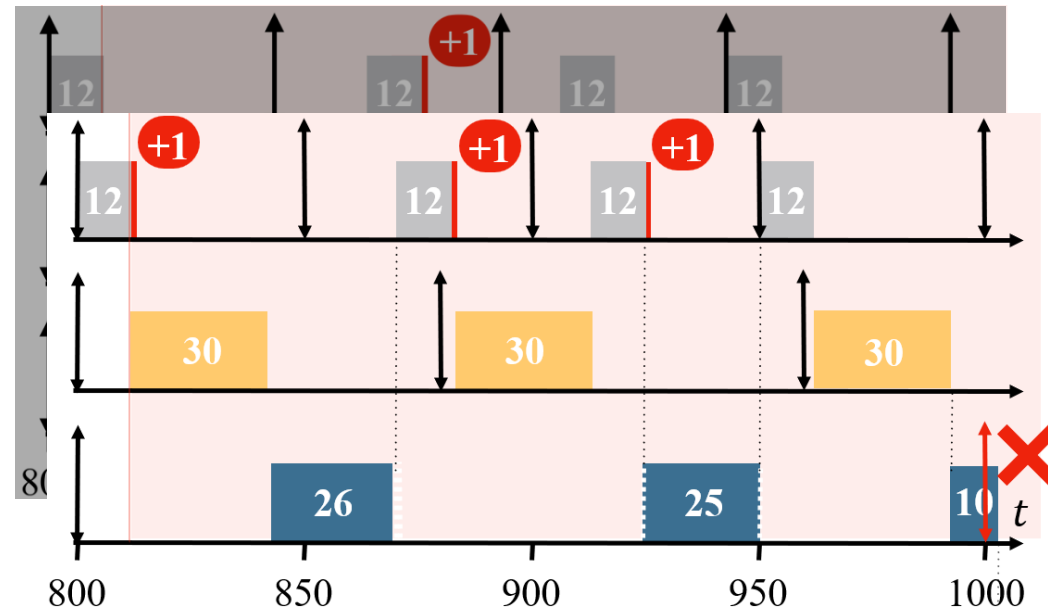


Who Generates the Exceedance?



Once we know the amount of exceedance that produces a nonlinearity, we still have **many scenarios** that can produce such an amount of exceedance:

Exceedance:
3 time units

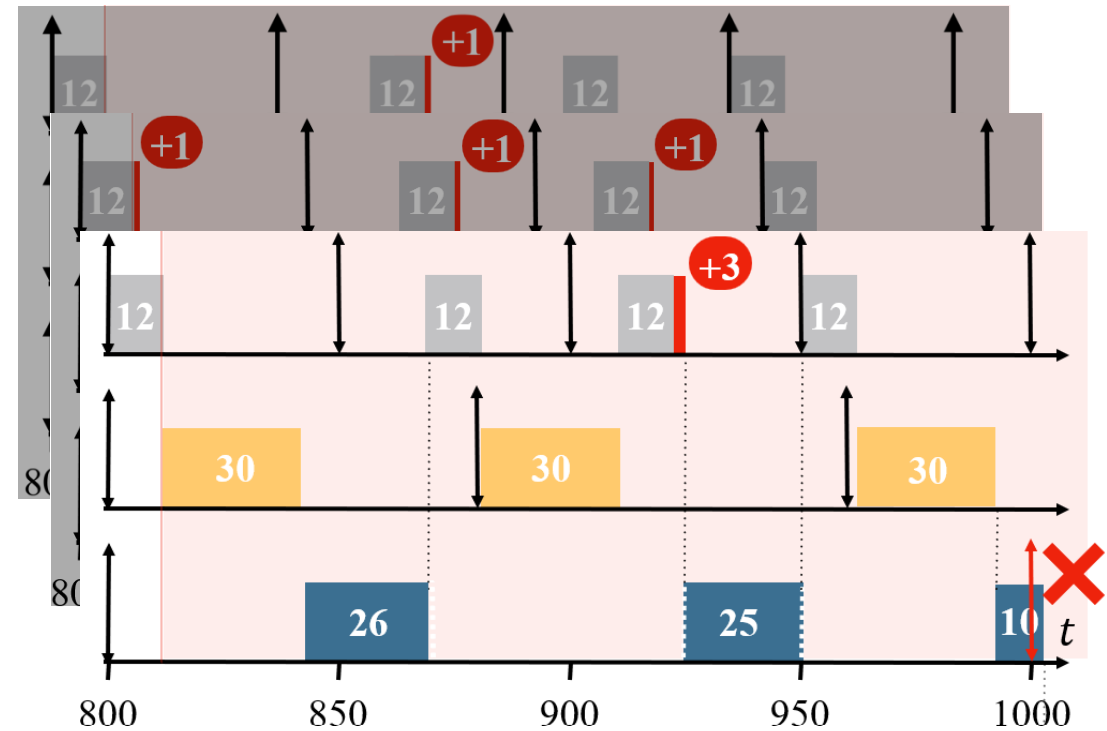


Who Generates the Exceedance?



Once we know the amount of exceedance that produces a nonlinearity, we still have **many scenarios** that can produce such an amount of exceedance:

Exceedance:
3 time units





We need an instrument to **explore the space of interesting scenarios**



We need an instrument to **explore the space of interesting scenarios**

QUESTION

What is an **interesting scenario**?



We need an instrument to **explore the space of interesting scenarios**

QUESTION

What is an **interesting scenario**?



This may vary based on:

- The specific system
- The workload
- The designer's preferences
- ...



We developed a **configurable MILP-based tool** that produces execution traces that trigger the nonlinearities.



We developed a **configurable MILP-based tool** that produces execution traces that trigger the nonlinearities.

The tool has **three main tuning knobs**:



We developed a **configurable MILP-based tool** that produces execution traces that trigger the nonlinearities.

The tool has **three main tuning knobs**:

NET trustworthiness

Is the task recently developed or well-known?





We developed a **configurable MILP-based tool** that produces execution traces that trigger the nonlinearities.

The tool has **three main tuning knobs**:

NET trustworthiness

Is the task recently developed or well-known?



Min / Max exceedance per job

Support for budget enforcement or self-aborting jobs.
Exploration of specific scenarios.



Solution Configurability



We developed a **configurable MILP-based tool** that produces execution traces that trigger the nonlinearities.

The tool has **three main tuning knobs**:

NET trustworthiness

Is the task recently developed or well-known?



Min / Max exceedance per job

Support for budget enforcement or self-aborting jobs.
Exploration of specific scenarios.



Exeedance balancer

A few jobs exceed a lot
vs
many jobs exceed a little





The search algorithm and the exceedance-distribution tool were **evaluated** with a set of experiments:



The search algorithm and the exceedance-distribution tool were **evaluated** with a set of experiments:

- Evaluation on synthetic tasksets
 - Dirichlet-Rescale-based (DRS) workload
 - Automotive-based workload



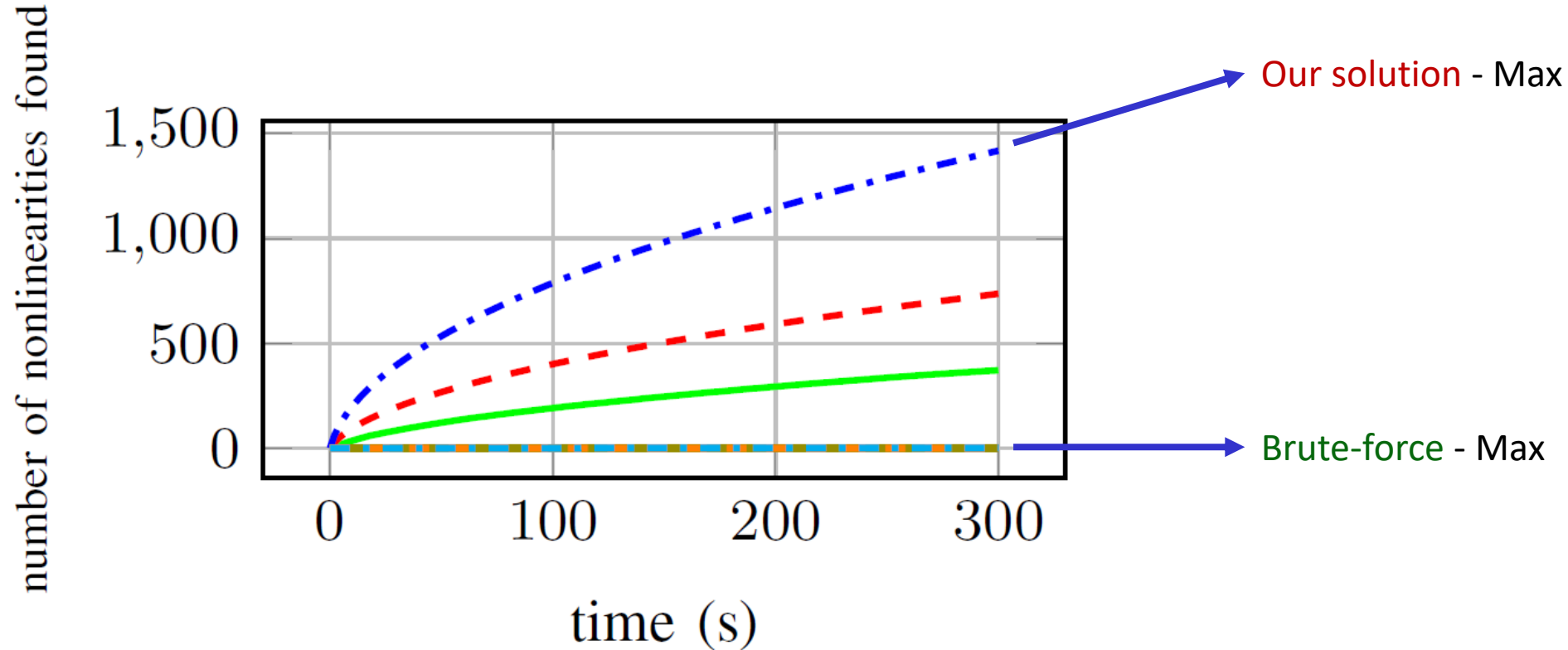
The search algorithm and the exceedance-distribution tool were **evaluated** with a set of experiments:

- Evaluation on synthetic tasksets
 - Dirichlet-Rescale-based (DRS) workload
 - Automotive-based workload
- Case study
 - WATERS 2017 industrial challenge



The **nonlinearities search algorithm** was compared with a brute-force solution

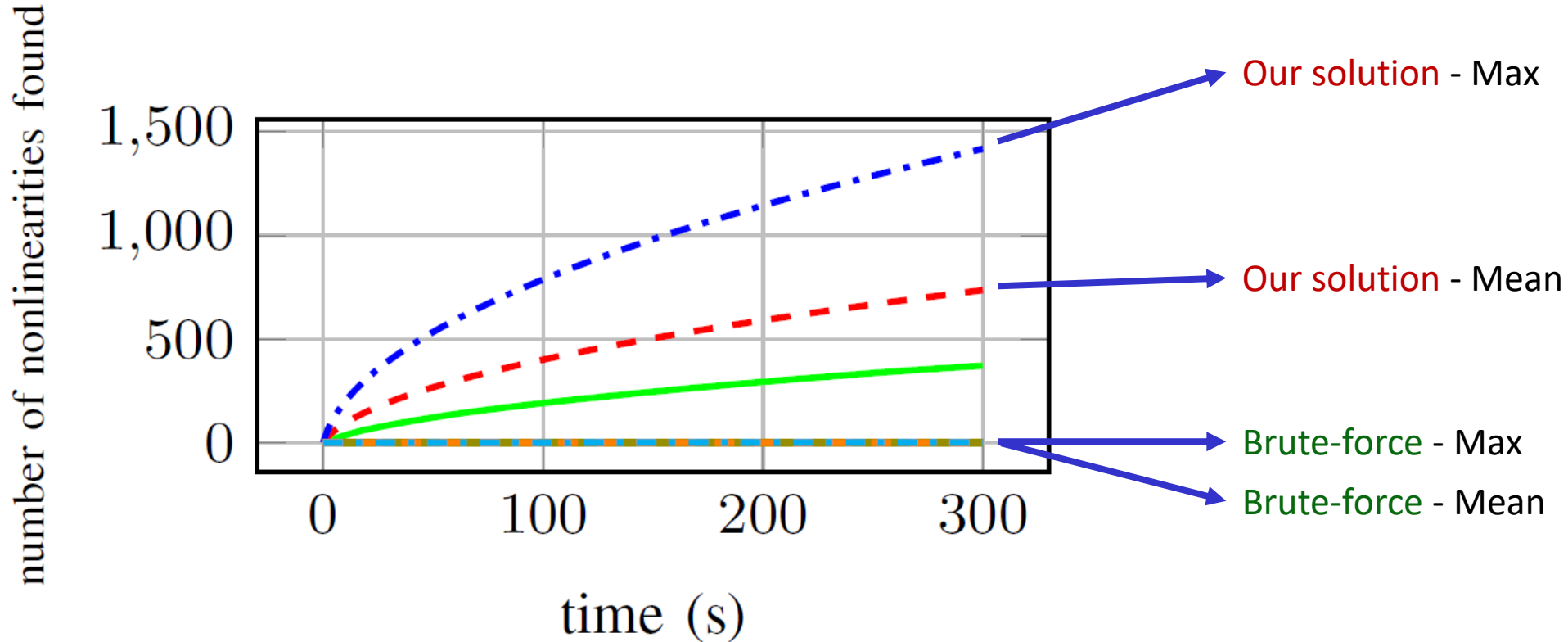
DRS-based, fully non-preemptive workload – 3600 tasksets





The **nonlinearities search algorithm** was compared with a brute-force solution

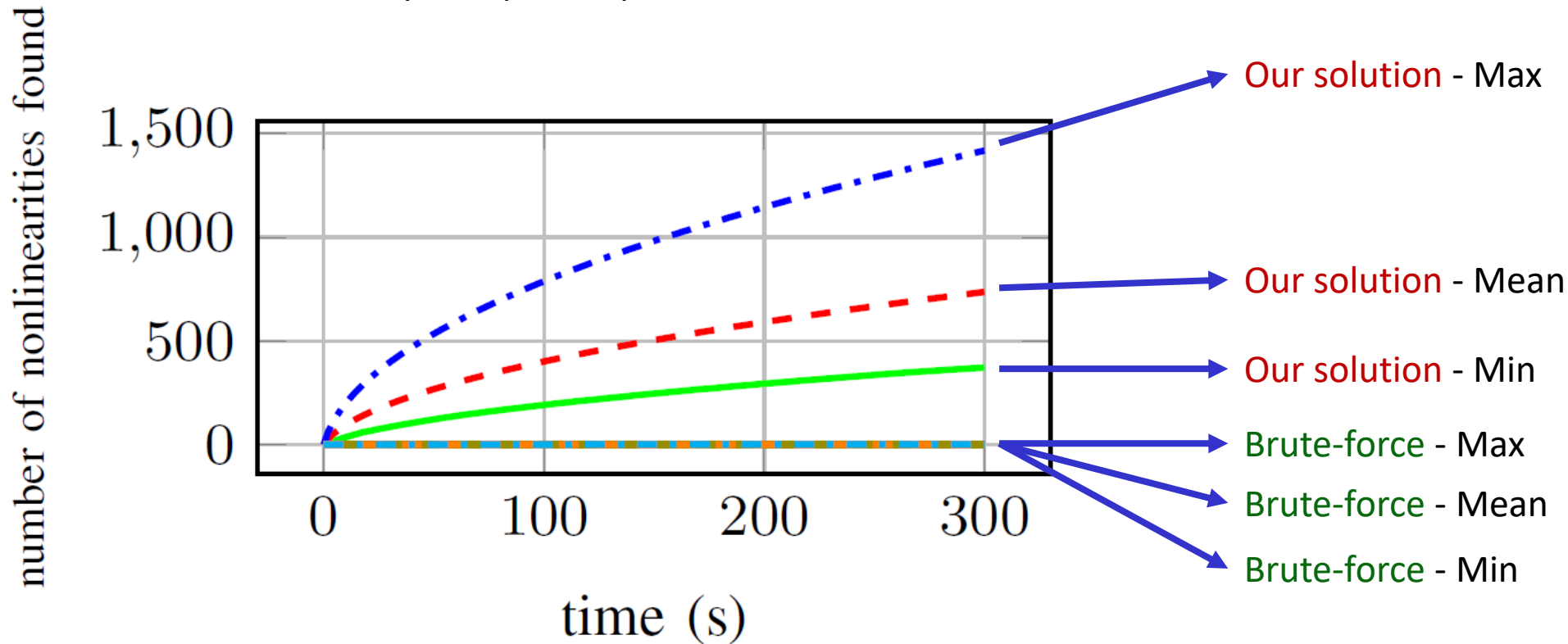
DRS-based, fully non-preemptive workload – 3600 tasksets





The **nonlinearities search algorithm** was compared with a brute-force solution

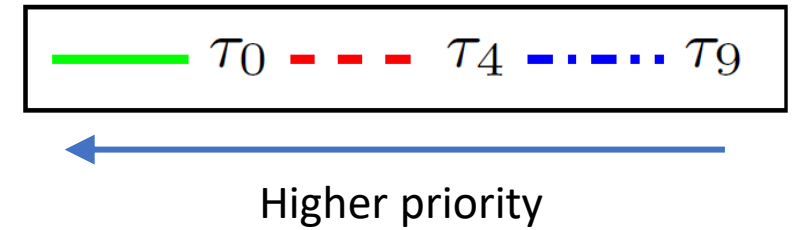
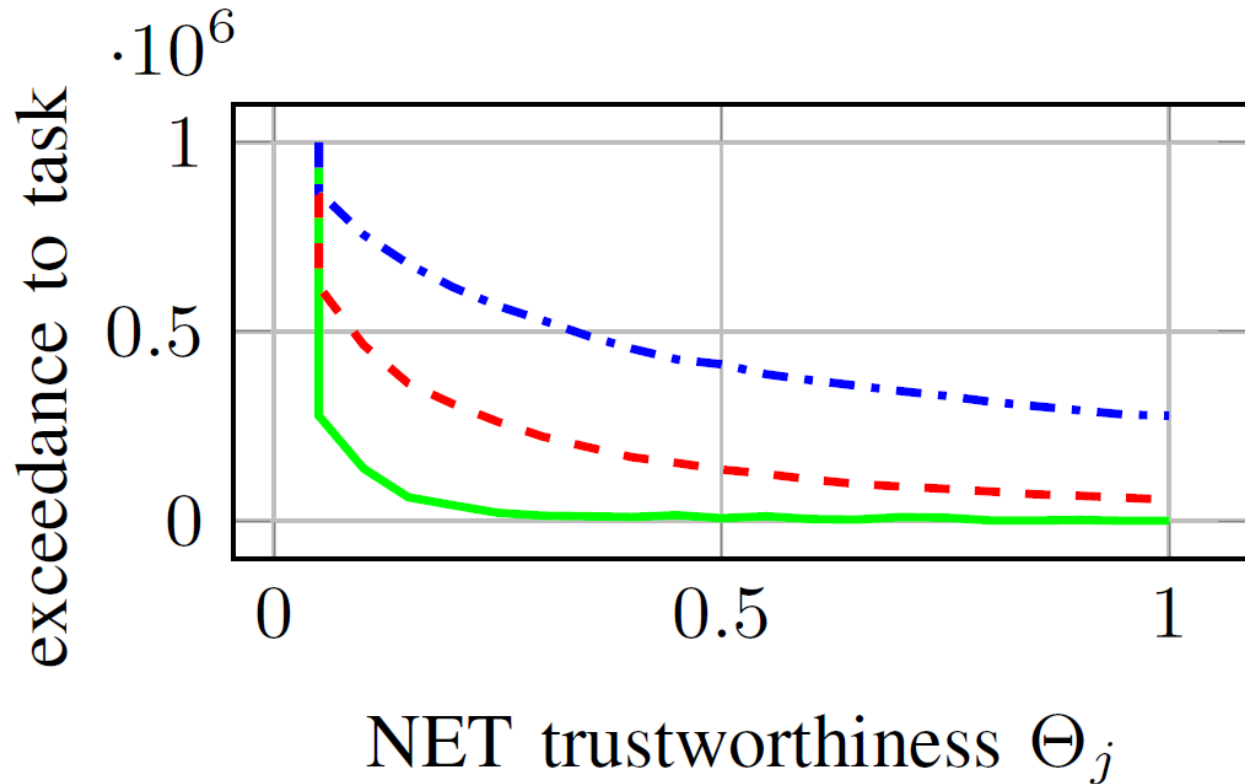
DRS-based, fully non-preemptive workload – 3600 tasksets





The **NET trustworthiness** parameter affects the amount of exceedance assigned to a task

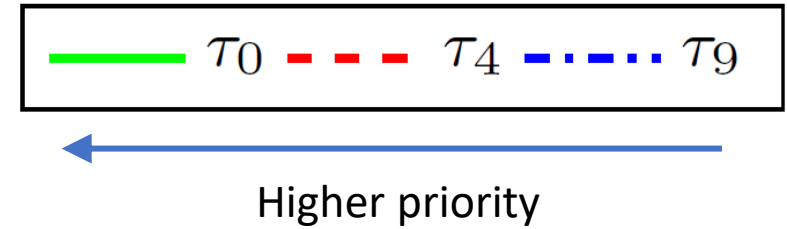
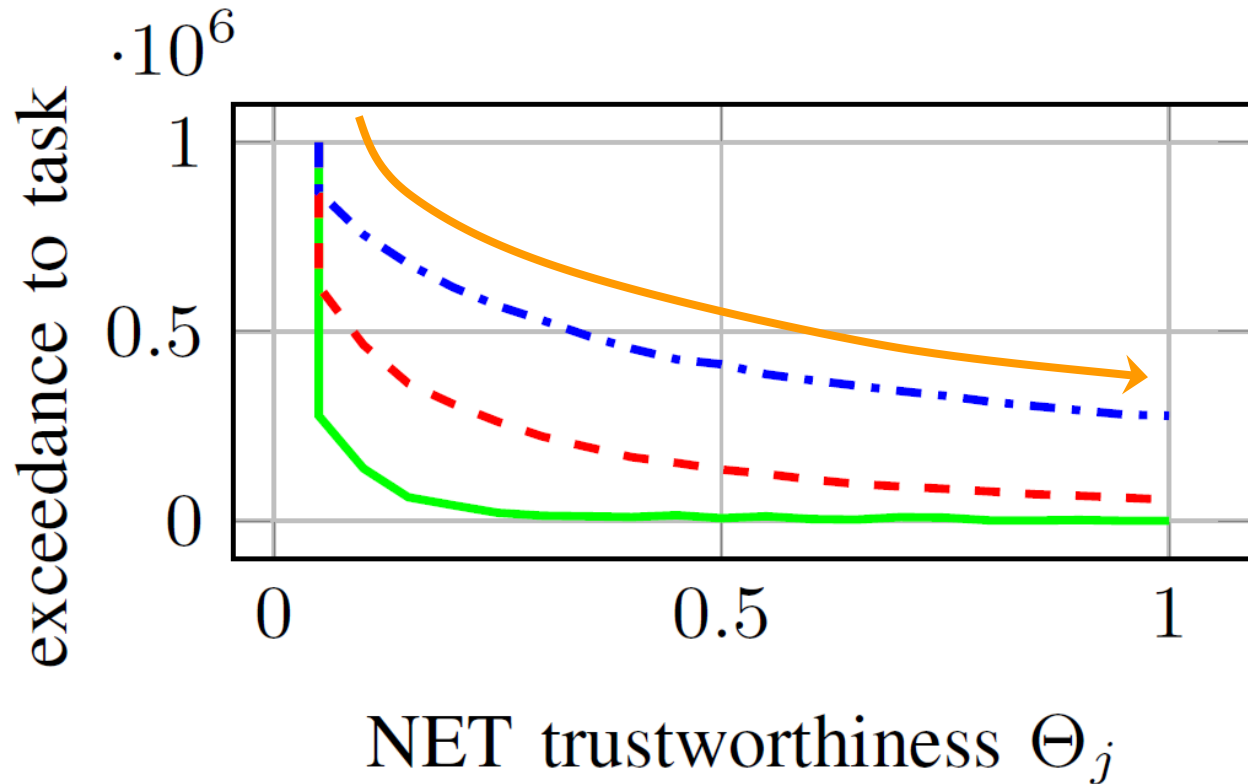
DRS-based, fully preemptive workload – 100 tasksets w/ 10 tasks





The **NET trustworthiness** parameter affects the amount of exceedance assigned to a task

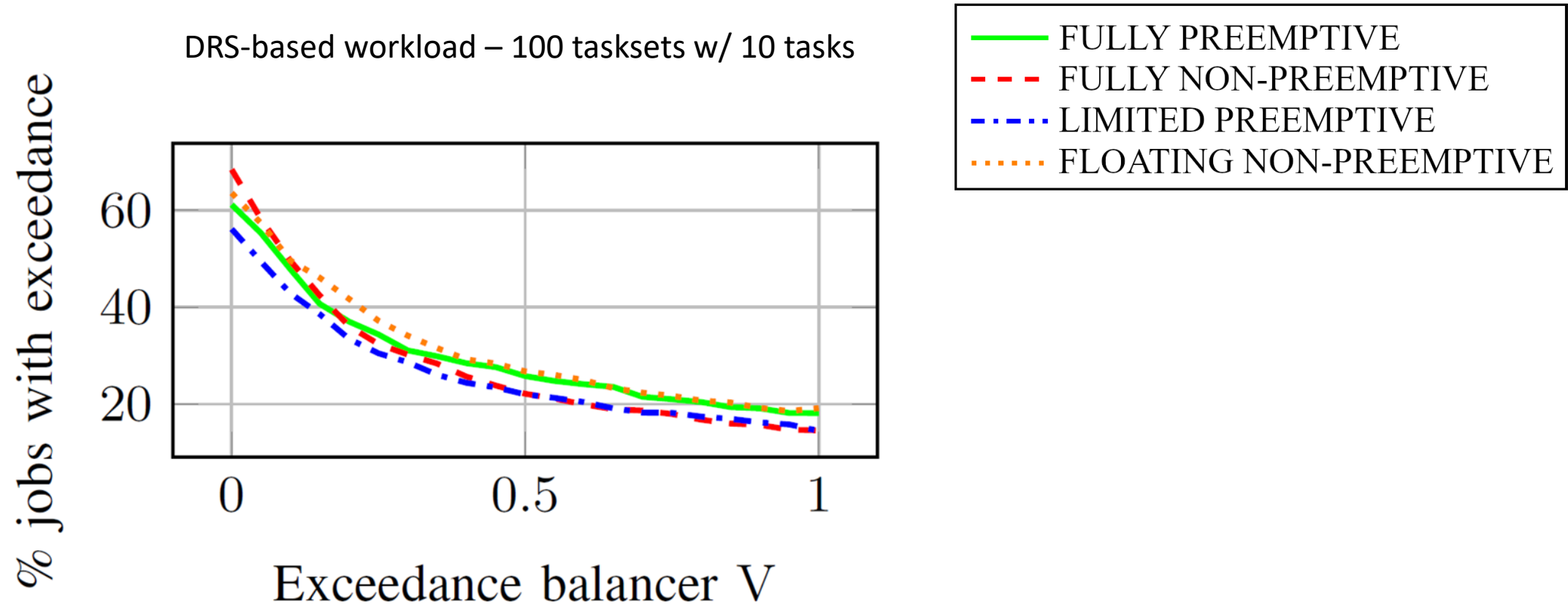
DRS-based, fully preemptive workload – 100 tasksets w/ 10 tasks



If the task is **trusted more**,
it is assigned **less exceedance**

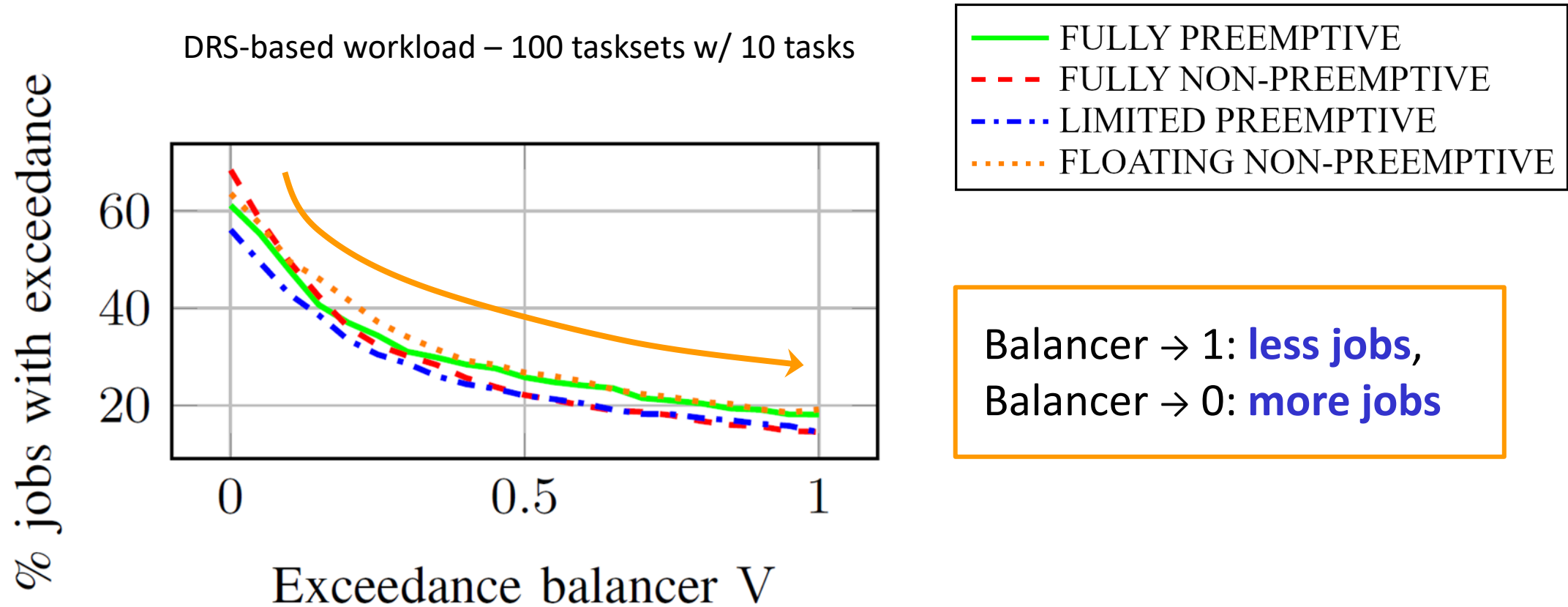


The **exceedance balancer** parameter affects the number of jobs that are assigned exceedance





The **exceedance balancer** parameter affects the number of jobs that are assigned exceedance



Case Study



The optimization problem can be **easily extended** to consider additional metrics.



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:

Original taskset not schedulable
with WCETs!



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

Slowdown = $exceedance / NET$

NOTE:

Original taskset not schedulable
with WCETs!



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:
Original taskset not schedulable
with WCETs!

Slowdown = exceedance / NET

The MILP was modified to look for the slowdown that makes the taskset unschedulable.

Task	Period	NET	e	min max x_h^s	$L_7(e)$
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:
Original taskset not schedulable
with WCETs!

Slowdown = exceedance / NET

The MILP was modified to look for the slowdown that makes the taskset unschedulable.

Exceedance to deadline miss

Task	Period	NET	e	min max x_h^s	$L_7(e)$
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:
Original taskset not schedulable with WCETs!

Slowdown = exceedance / NET

The MILP was modified to look for the slowdown that makes the taskset unschedulable.

Task	Period	NET	Slowdown necessary for a deadline miss to occur		$L_7(e)$
			Exceedance to deadline miss e	$\min \max x_h^s$	
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:
Original taskset not schedulable with WCETs!

Slowdown = exceedance / NET

The MILP was modified to look for the slowdown that makes the taskset unschedulable.

Task	Period	NET	Exceedance to deadline miss e	Slowdown necessary for a deadline miss to occur $\min \max x_h^s$	Bound on the recovery time $L_7(e)$
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:
Original taskset not schedulable with WCETs!

Slowdown = exceedance / NET

The MILP was modified to look for the slowdown that makes the taskset unschedulable.

Task	Period	NET	Exceedance to deadline miss e	Slowdown necessary for a deadline miss to occur $\min \max x_h^s$	Bound on the recovery time $L_7(e)$
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms

High-priority tasks have a large safety margin

Case Study



The optimization problem can be **easily extended** to consider additional metrics.

Case study: WATERS 2017 challenge - $NET = 0.9 \cdot WCET$

NOTE:
Original taskset not schedulable with WCETs!

Slowdown = exceedance / NET

The MILP was modified to look for the slowdown that makes the taskset unschedulable.

Task	Period	NET	Exceedance to deadline miss e	Slowdown necessary for a deadline miss to occur $\min \max x_h^s$	Bound on the recovery time $L_7(e)$
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms

High-priority tasks have a large safety margin

Low-priority tasks have a very small safety margin!

Conclusions





- When using NETs, exceedance effects must be analyzed carefully due to the presence of **hard-to-predict nonlinearities**.



- When using NETs, exceedance effects must be analyzed carefully due to the presence of **hard-to-predict nonlinearities**.
- Exceedance analysis helps to **efficiently explore** the space of exceedance effects in order to:
 - Explore hypothetical scenarios;
 - Assess the system's safety margin w.r.t. transient execution-time fluctuations.



- When using NETs, exceedance effects must be analyzed carefully due to the presence of **hard-to-predict nonlinearities**.
- Exceedance analysis helps to **efficiently explore** the space of exceedance effects in order to:
 - Explore hypothetical scenarios;
 - Assess the system's safety margin w.r.t. transient execution-time fluctuations.
- The results of the analysis can be presented visually and **easily understood** by the system's designer.

Future work





- The approach can be extended to **other scheduling policies**
 - Support for locking protocols
 - Self-suspending tasks
 - ...



- The approach can be extended to **other scheduling policies**
 - Support for locking protocols
 - Self-suspending tasks
 - ...

- The approach can be extended to **other task parameters:**
 - Release jitter
 - Critical-section length
 - Supply-bound functions
 - ...

Thank you!