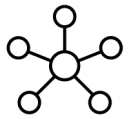
A map of all our failures

Why Reliable Systems Need More Deadline Misses

Silviu S. Craciunas

Senior Principal Scientist, NXP
Associate Professor, Technical University of Denmark



Real-time And intelliGent Edge computing workshop (RAGE 2026)

trustmotion
part of NXP

Project highlights

MotionWise middleware

Hardware development until the A-Sample grade

Real-time operating system for the Vestas 2-, 4-and 6-Megawatt turbines

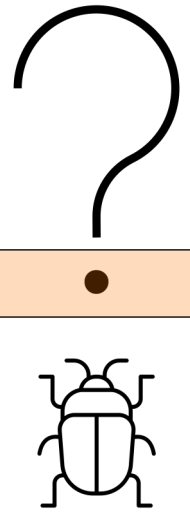
Collins Aerospace Power System and Air Management for the Boeing 787-8, 787-9, 787-10 family

TTEthernet Switch and End System IP products for the avionics system.

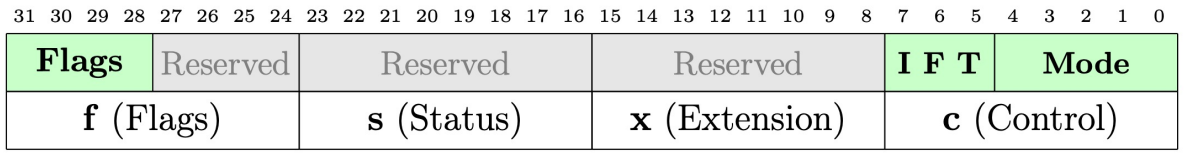
TTEthernet is the “nervous system“ i.e. avionics network platform of the space craft

My biggest failure

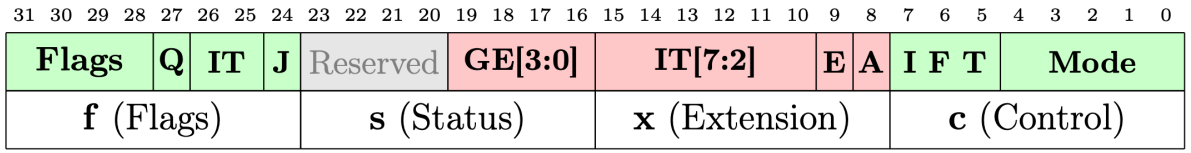
```
ldr         r0, curTask
ldr         r0, [r0]
ldr         r14, [r0]
ldmia       r14!, {r0-r1}
vmsr        fpscr, r1
msr         spsr_cf, r0
vldmia.64   r14!, {d0-d15}
ldmia       r14, {r0-lr}^
ldr         lr, [r14, #0x3C]
subs        pc, lr, #4
```



```
ldr         r0, curTask
ldr         r0, [r0]
ldr         r14, [r0]
ldmia       r14!, {r0-r1}
vmsr        fpscr, r1
msr         spsr_cxsf, r0
vldmia.64   r14!, {d0-d15}
ldmia       r14, {r0-lr}^
ldr         lr, [r14, #0x3C]
subs        pc, lr, #4
```



ARM5: Reserved for future use



ARM7: If-Then state bits, Endianness bit, Greater-Than-Equal flags for SIMD instructions.

Every Engineer's Nightmare: The Bug That Disappears When You Look for It

Bohrbug: a “boring”, deterministic failure. Like the Bohr atom, they are solid. You input X, it crashes. These are "**Good**" bugs because we can easily debug and fix them.

Heisenbugs: a non-deterministic failure. Like the uncertainty principle, they change or vanish when you inspect them. These are caused by race conditions, rare interrupt alignments, or cache coherence issues. These are the "**Bad**" bugs that are a nightmare to debug and fix.

J. Gray, "Why do Computers Stop and What can be done about it?," Proc. 5th Symp. on Reliability in Distributed Software and Database Systems, 1986

BohrMiss vs HeisenMiss

Time is the ultimate Heisenbug:

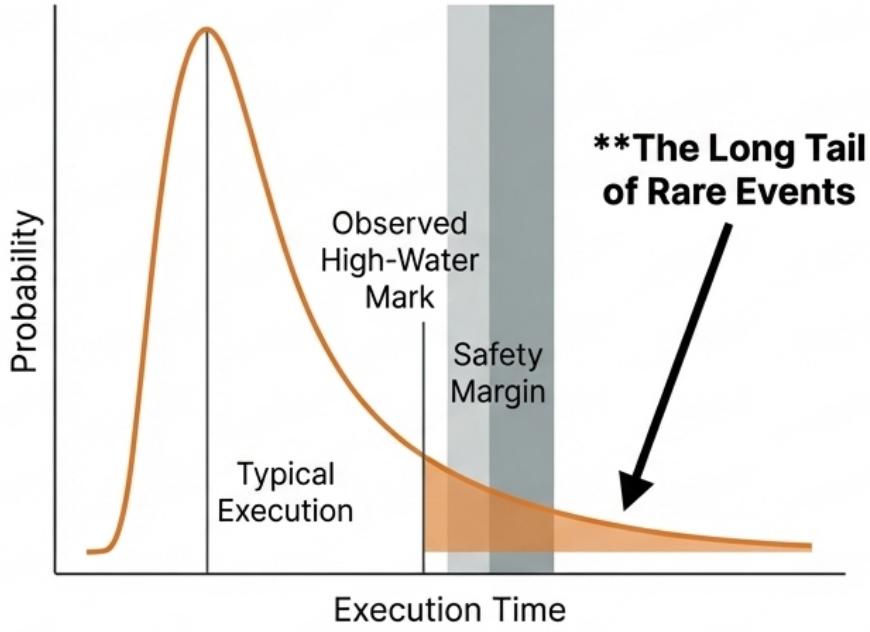
- In real-time systems, the logic may be correct, but time itself is the variable.
- A system can function perfectly for 1,000 hours & then fail for 1 millisecond.
- This introduces a new class of timing-related Heisenbugs.
- "**HeisenMiss**": latent model inaccuracies that manifest as probabilistic timing failures.
- Modern system complexity creates more opportunities for HeisenMisses to emerge.

Key insights:

- Design, development, and integration phases are very different from production and deployment stages.
- There will always be model inaccuracies!
- True system reliability isn't the absence of failure during development.
- Reliability in deployment comes from designing the system such that any potential failure will be deterministic, reproducible, and visible during development.

The WCET illusion

- Maximum execution time of a program on a micro-architecture for all possible inputs.
- Analysis-based: Pessimistic, expensive, and sometimes just not possible.
- Measurement-based:
  - Execute program for all inputs: often impractical
  - Execute program for selected inputs to get a lower bound on WCET: observed WCET can be far below true WCET



Reality check wrt. WCET:

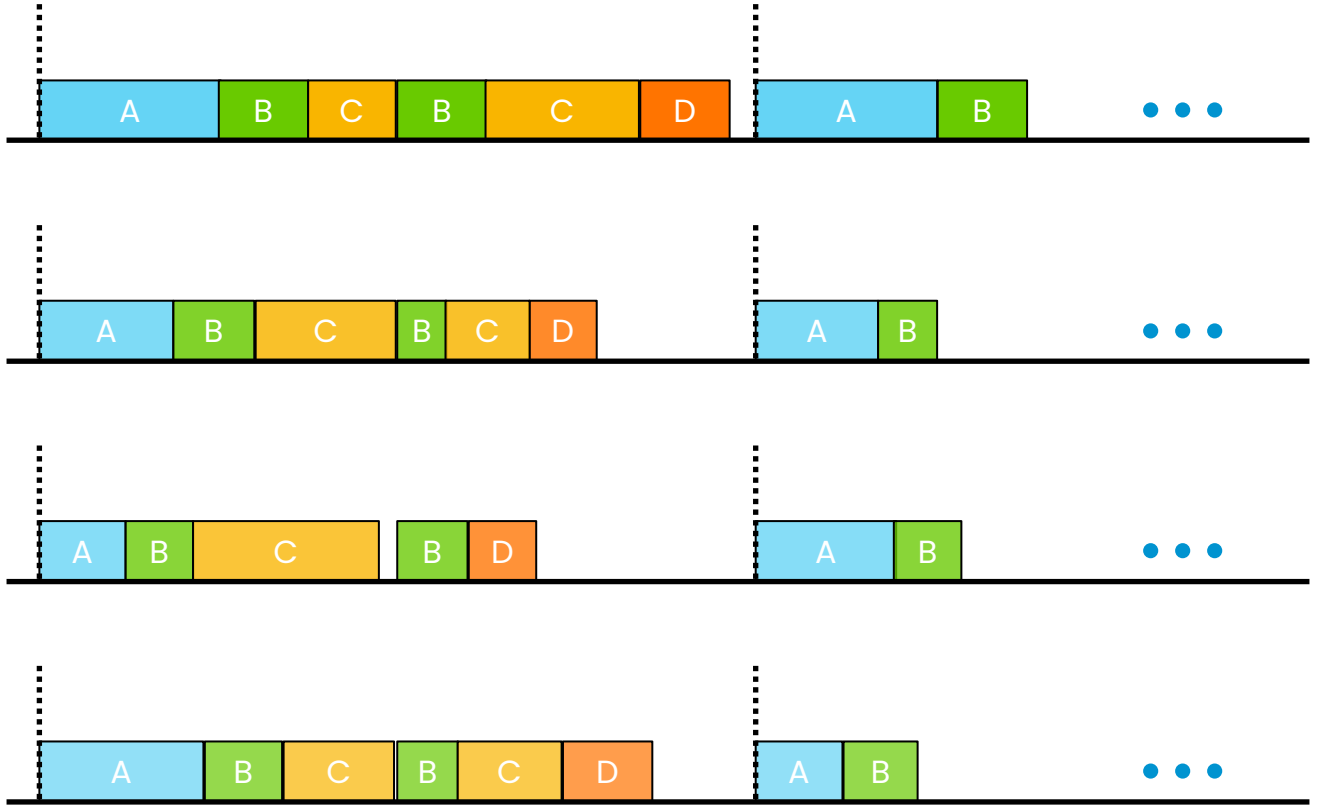
1. WCET is almost impossible to get, and even if we get it, it is very pessimistic (wasting resources).
2. Even if good, tight WCET numbers are possible, customers will not provide them (too expensive).
3. Mostly, we get some measurement data, but not an exhaustive CFG sweep.

We design systems based on the assumption that the WCET is a single, safe number.
In reality, it is an incomplete distribution.

Runtime variance of fixed-priority (FP) scheduling

simulated trace
with WCET for all tasks

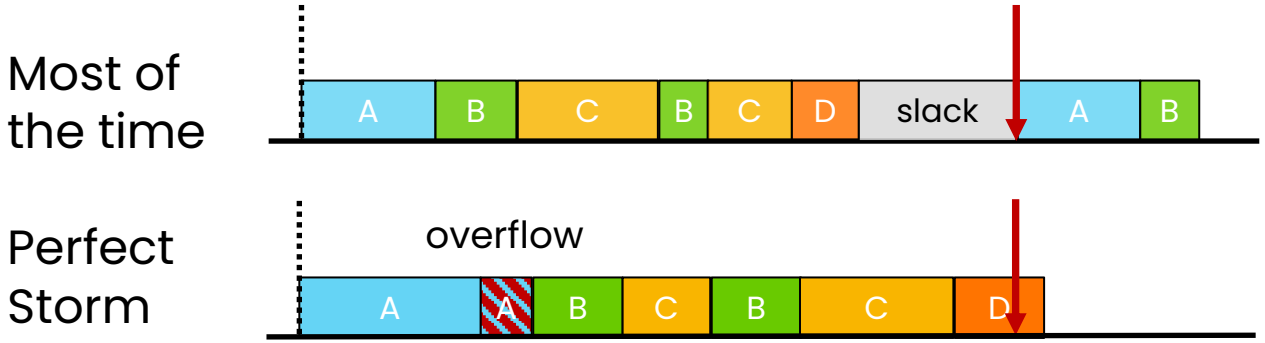
possible actual traces



The HeisenMiss Factory

What happens if a task executes for WCET + 1% in a classical FP system?
Most of the time: **Nothing**. The slack by other tasks absorbs the delay. **The bug is invisible.**

The "**Perfect Storm**": The system works fine for months until the day that Task A overflows, and Task B, C, & D ALL hit their execution peaks simultaneously. The delays stack up, the slack vanishes, and you miss a hard deadline.



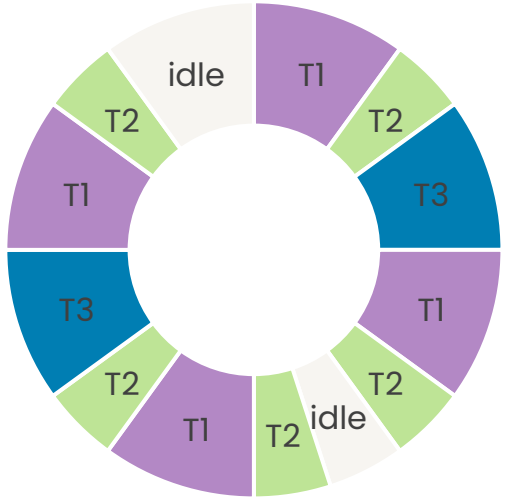
Task A overflowed, but task D missed its deadline.
How do you debug this?

The system is essentially a **HeisenMiss factory**.
The system's behaviour is **probabilistic**.

Time-triggered scheduling

- Tasks are executed based on a **static schedule table** that has been computed at design time
- All scheduling decisions are taken at **compile time**, WCET needs to be known.
- The complete scheduling timeline for TT tasks is **fixed**.
- Simple (e.g. deadlines) & complex (e.g. chains) **constraints** are satisfied by **schedule creation**.
- One solution is sufficient and any solution is a sufficient schedulability test.
- **Idle time** in the static schedule can be used more dynamically.

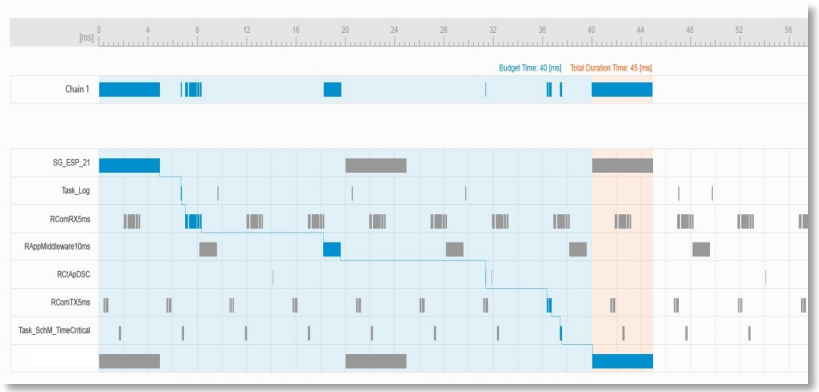
Static Schedule Table



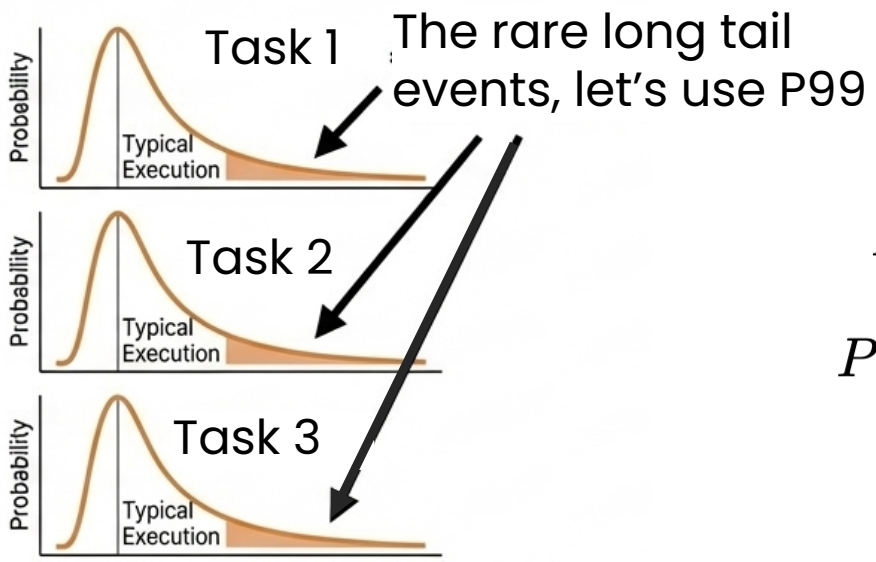
Time-triggered scheduling

Time-Triggered provides correct-by-design real-time guarantees.

- **Complex timing requirements**: cause-effect chains, different types of jitter
- **Temporal isolation**: no starvation possible, temporal isolation between all tasks
- **Determinism/Predictability**: increased stability and testability, no unwanted run-time effects, many system properties become predictable, e.g., locks, task pre-emption
- **Re-simulation:** Equivalent behaviour between cloud and embedded target
- **Synchronization to communication**: stable real-time behaviour of cause-effect chains
- **Compositionality/Integration**: adding or modifying tasks can be done incrementally, system integration of SWCs from different suppliers is easier
- **Schedulability**: more correct configurations can be realized
- "**What you see is what you get**" execution



Probability of Bohrmisses vs Heisenmisses



$$P(C_i > P_{99}) = 0.01 \quad (\text{or } 10^{-2})$$

$$P(\text{Failure}_{\text{FP}}) = P(C_1 > P_{99}) \times P(C_2 > P_{99}) \times P(C_3 > P_{99})$$

$$P(\text{Failure}_{\text{FP}}) = 0.01 \times 0.01 \times 0.01 = \mathbf{10^{-6}} \quad (\text{One in a Million})$$

$$\forall \tau_i, P(\text{Failure}_{\text{TT}}) = P(C_i > P_{99}) = \mathbf{10^{-2}} \quad (\text{One in a Hundred})$$

In a TT system, we can decide which tasks has slack and which has to finish strictly within the slot
The Failure Condition: If ANY task exceeds its P99 execution time, it hits the end of its slot.
The Scheduler acts as a guillotine. The failure of T3 is not masked by the good behavior of T1.
The Bohrtime Benefit: The failure rate is 10,000 times higher than in the FP system (10^{-2} vs 10^{-6}).

- This forces the failure to appear during development.
- It prevents the "latent inaccuracy" of a model from hiding behind system slack.
- It converts a probabilistic gamble into a deterministic constraint.

Why not just add deadline/budget monitoring to FP?

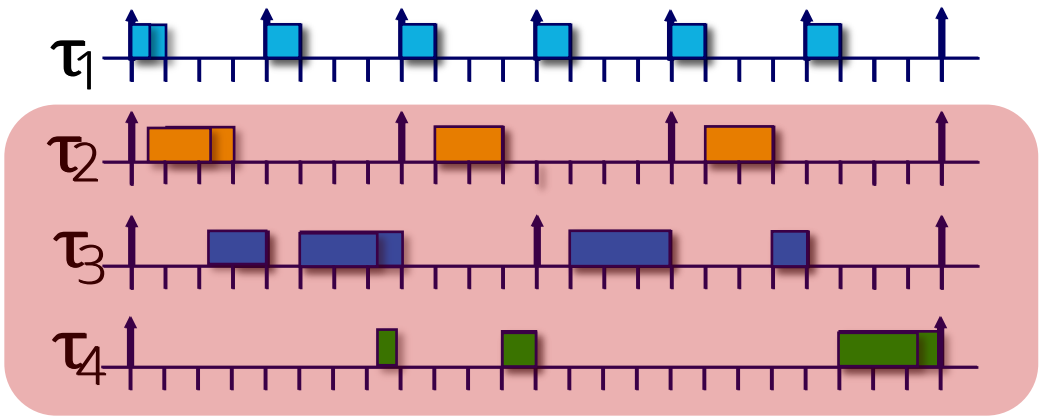
- Precise execution-time accounting is hard
  - Cycle-accurate accounting across preemption, interrupts, migration, DMA/accelerators, cache
  - Where do we account for system overhead?
- Most commercial FP RTOSes either don't offer this, or offer a coarse-grained approximation.
- TT systems avoid this entire class of problems:
  - Time is measured implicitly by the schedule.
  - No runtime accounting is required.
- Budget checks occur at: timer ticks, context switches, or instrumented code points.
  - Leads to additional interrupts altered preemption points.
  - Ironically: the act of monitoring can change the timing behavior it is meant to observe.
- This is far less of an issue in TT, where timing boundaries are architectural, and no fine-grained accounting is needed.

Even with budget monitoring, FP scheduling turns execution-time variance into an exponential number of states to test.

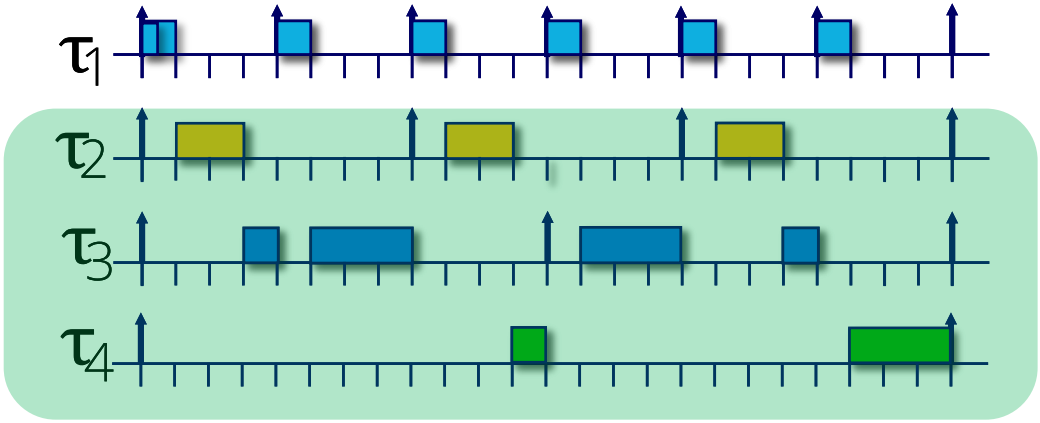
TT vs. FP/EDF

What happens if at some point in time, one task executes for less than the computed WCET?

Non-TT (FP/EDF)
- changes the execution of all other tasks
- state change for the entire system

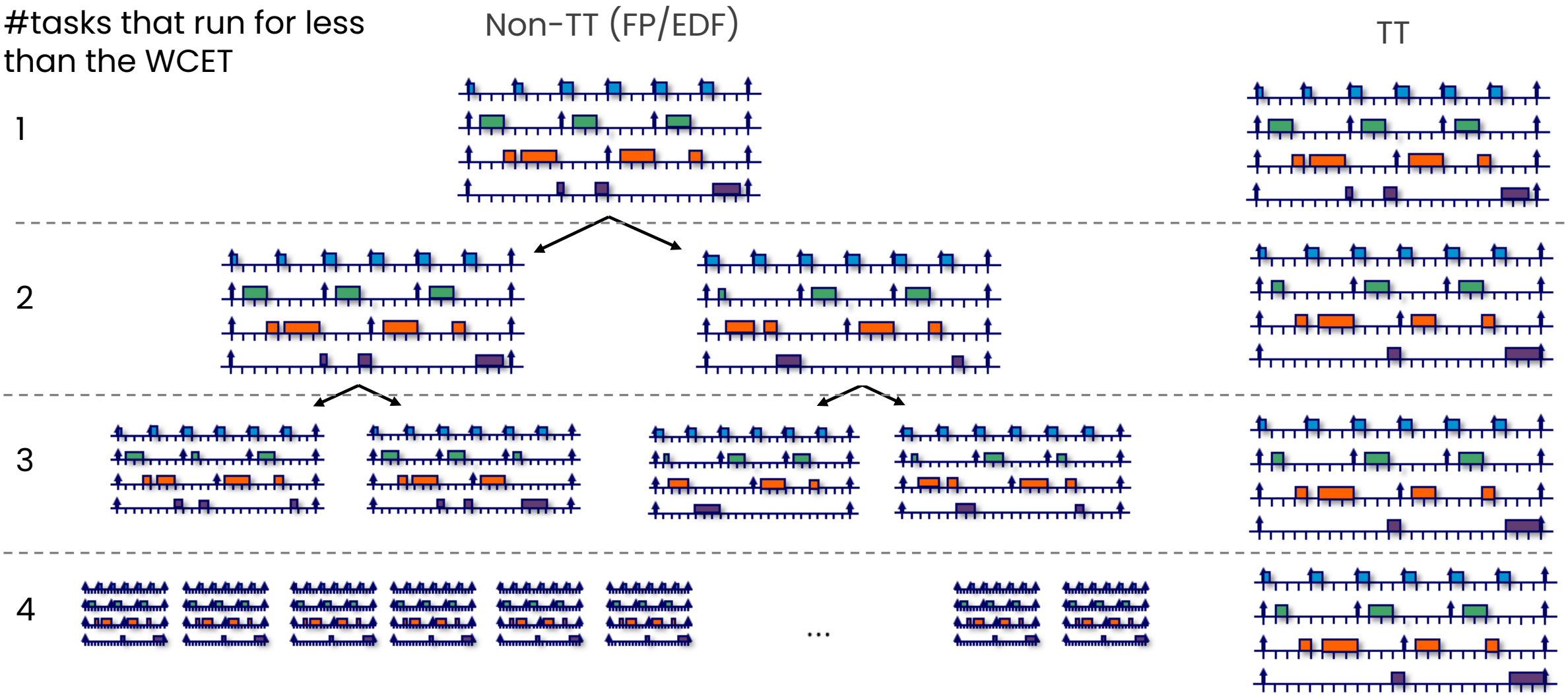


Time-Triggered
- execution of all other tasks remains the same
- no state change



State space explosion

#tasks that run for less than the WCET



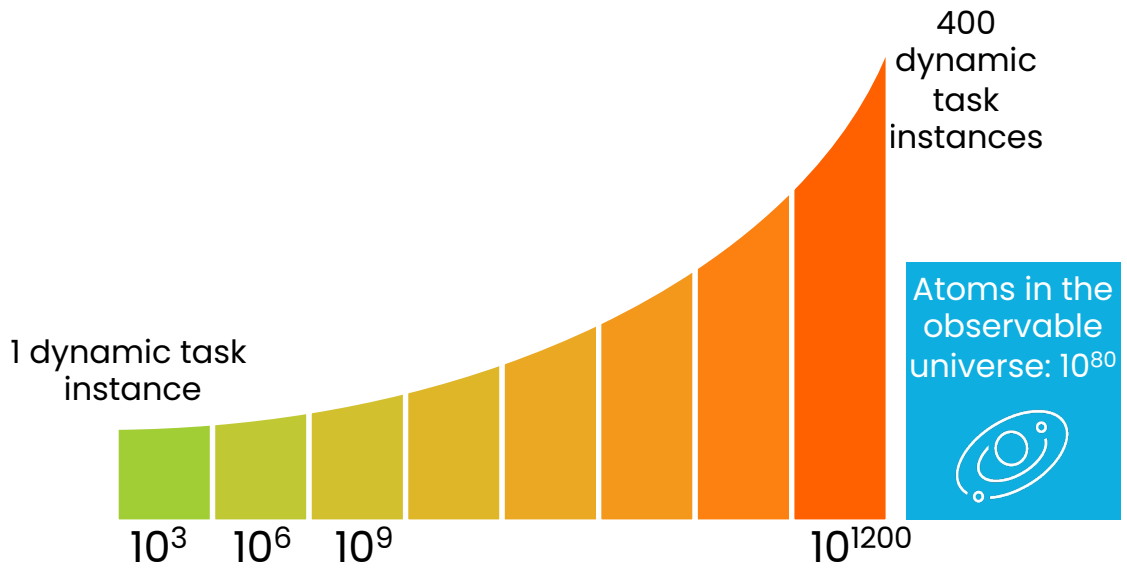
The State Space Explosion: Why You Can Not Test Your Way to Safety

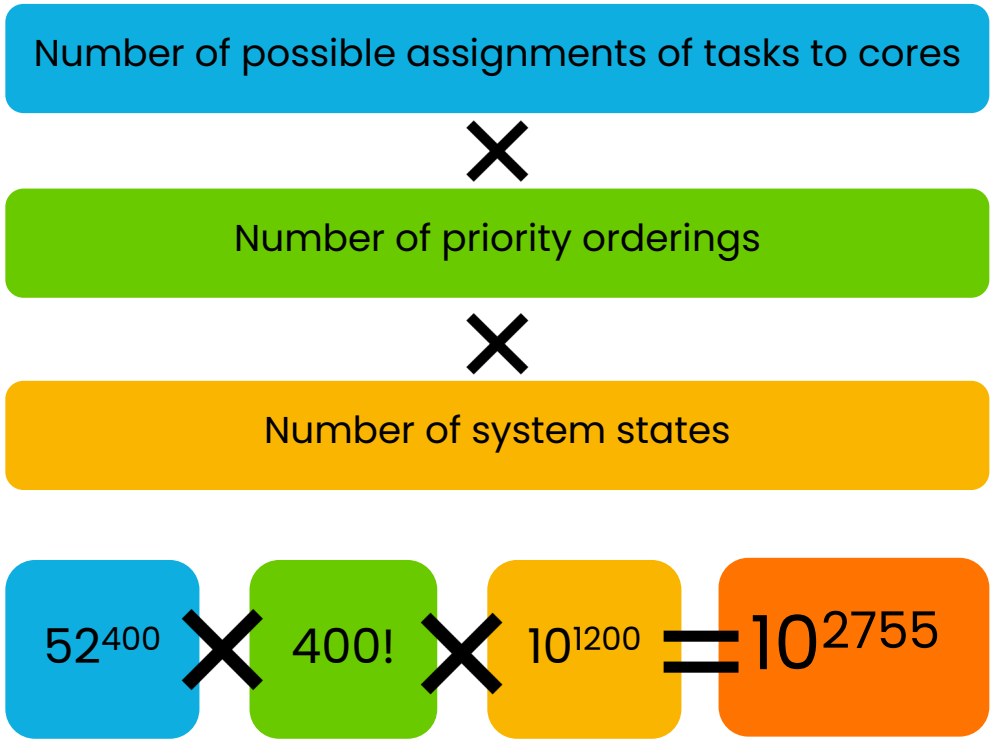
Safety-relevant projects need to address corner-cases not covered by testing. ISO 26262

How many system states do I have to potentially check/address to cover every corner case?

Non-time-triggered (FP/EDF/etc.):

- changes the execution of all other tasks
- state change for the entire system





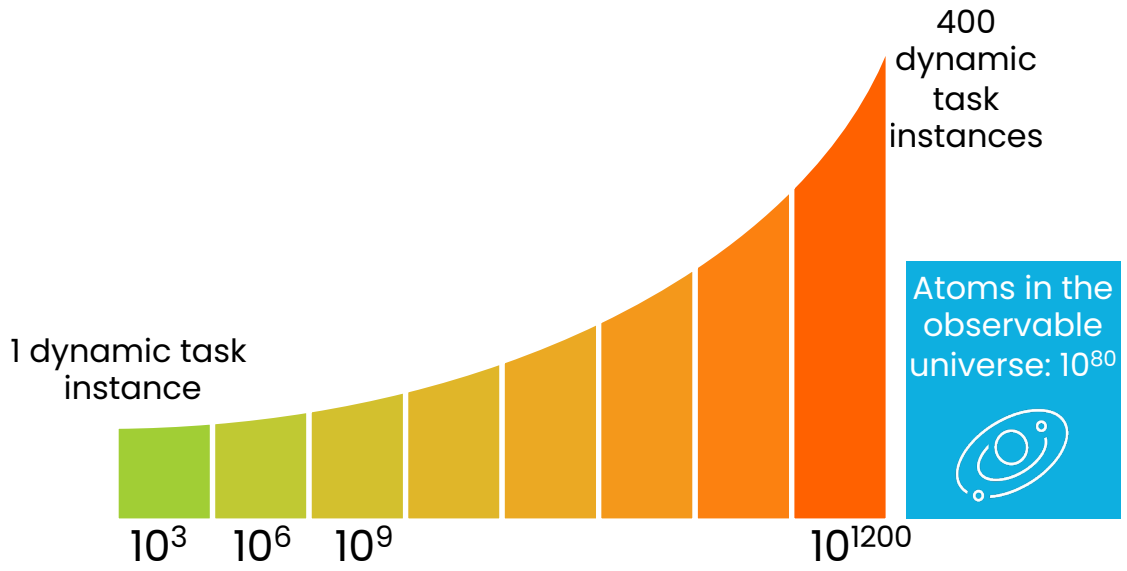
The State Space Explosion: Why You Can Not Test Your Way to Safety

Safety-relevant projects need to address corner-cases not covered by testing. ISO 26262

How many system states do I have to potentially check/address to cover every corner case?

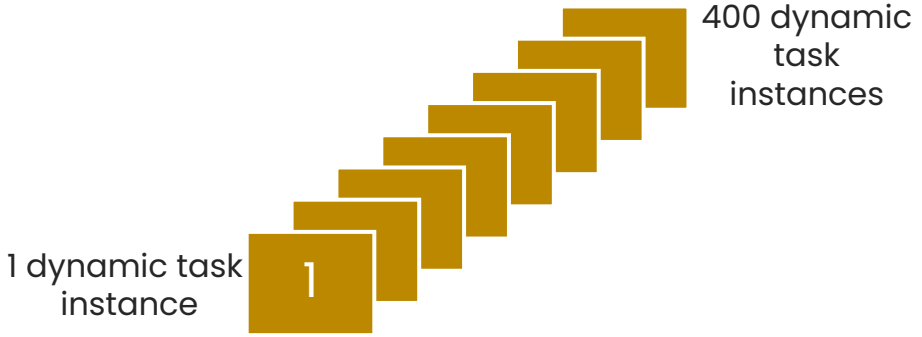
Non-time-triggered (FP/EDF/etc.):

- changes the execution of all other tasks
- state change for the entire system



Time-triggered:

- execution of all other tasks remains the same
- no state change



The State Space Explosion: Why You Can Not Test Your Way to Safety

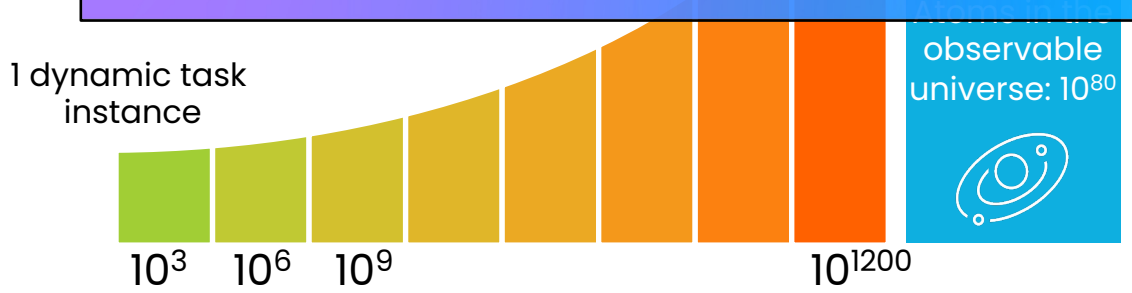
Safety-relevant projects need to address corner-cases not covered by testing. ISO 26262

How many system states do I have to potentially check/address to cover every corner case?

The complexity is moved from an intractable runtime analysis problem to a design-time synthesis problem.

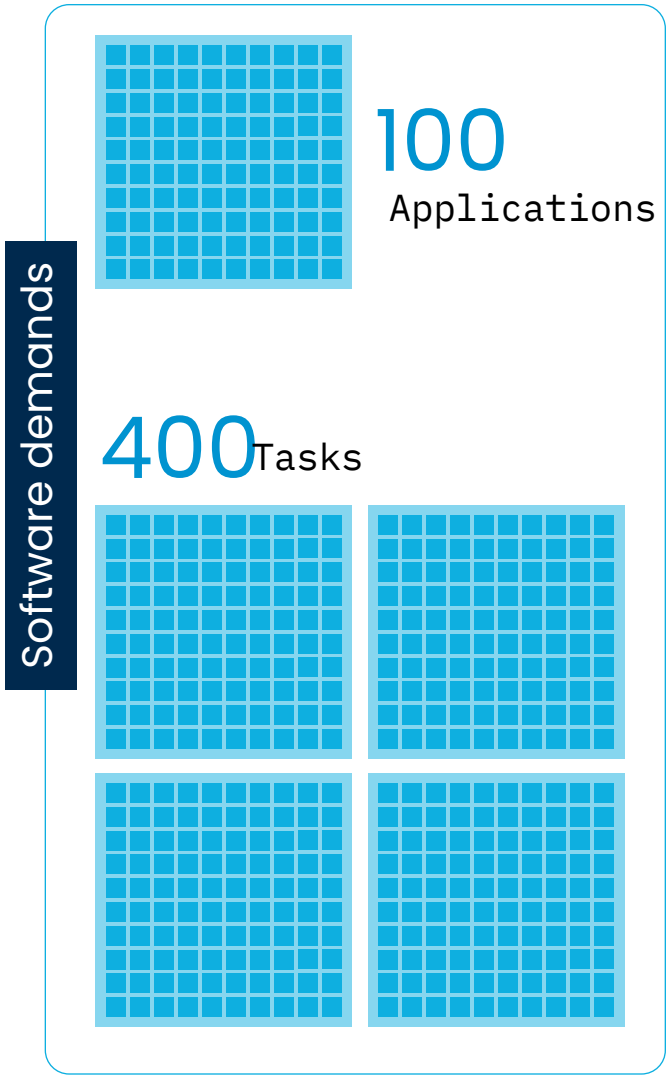
The choice between TT and other methods does not change the complexity of the problem!

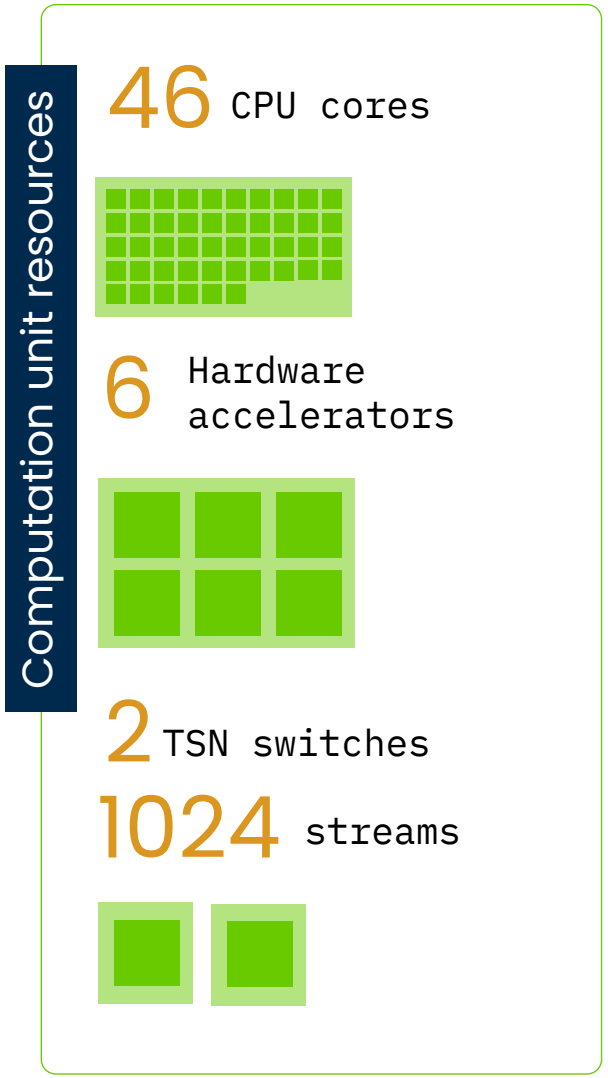
It rather affects the tools/methodology (=models/algorithms) available to tame this complexity!

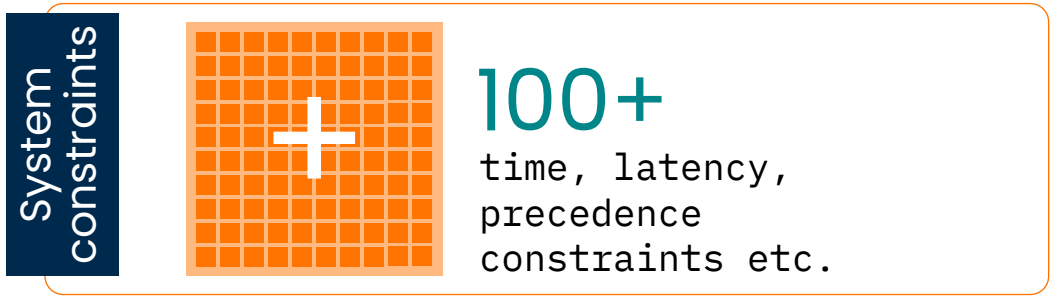


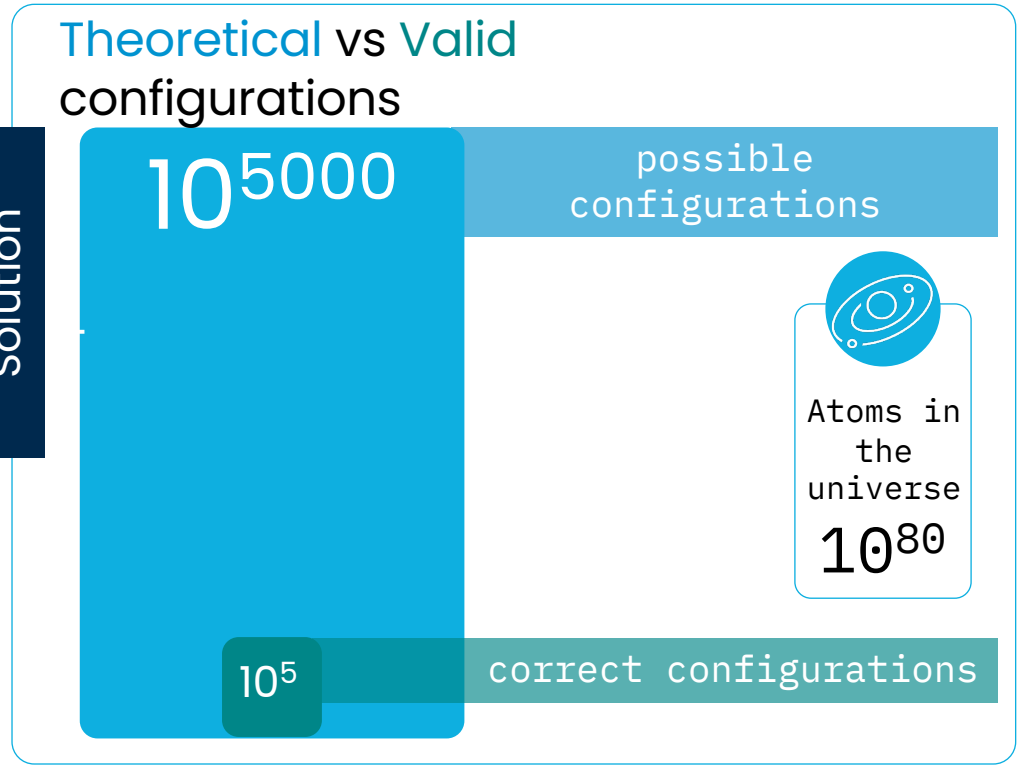
Real-world scheduling complexity

Real-world: thousands of tasks running on tens of cores/CPUs with hundreds of complex constraints









TT “debugability”

What Happens in Non-TT Systems

- Timing jitter and slack make many issues **rare or execution-dependent**.
- Some misses manifest only under specific load/timing patterns: hard to reproduce & hard to fix.

TT makes timing behavior more deterministic

- Some interactions that maybe extend WCET are removed
- Repeatable failures -> faster detection & root-cause analysis

Determinism Improves Debugging

- **Deviations from the formal model** show up consistently.
- These deviations are **Bohrmisses**: stable, repeatable, and analyzable.
- In development: **earlier detection** of modelling mistakes (e.g., wrong WCET).
- In operation: remaining bugs may be easier to diagnose because behavior is more **reproducible**.

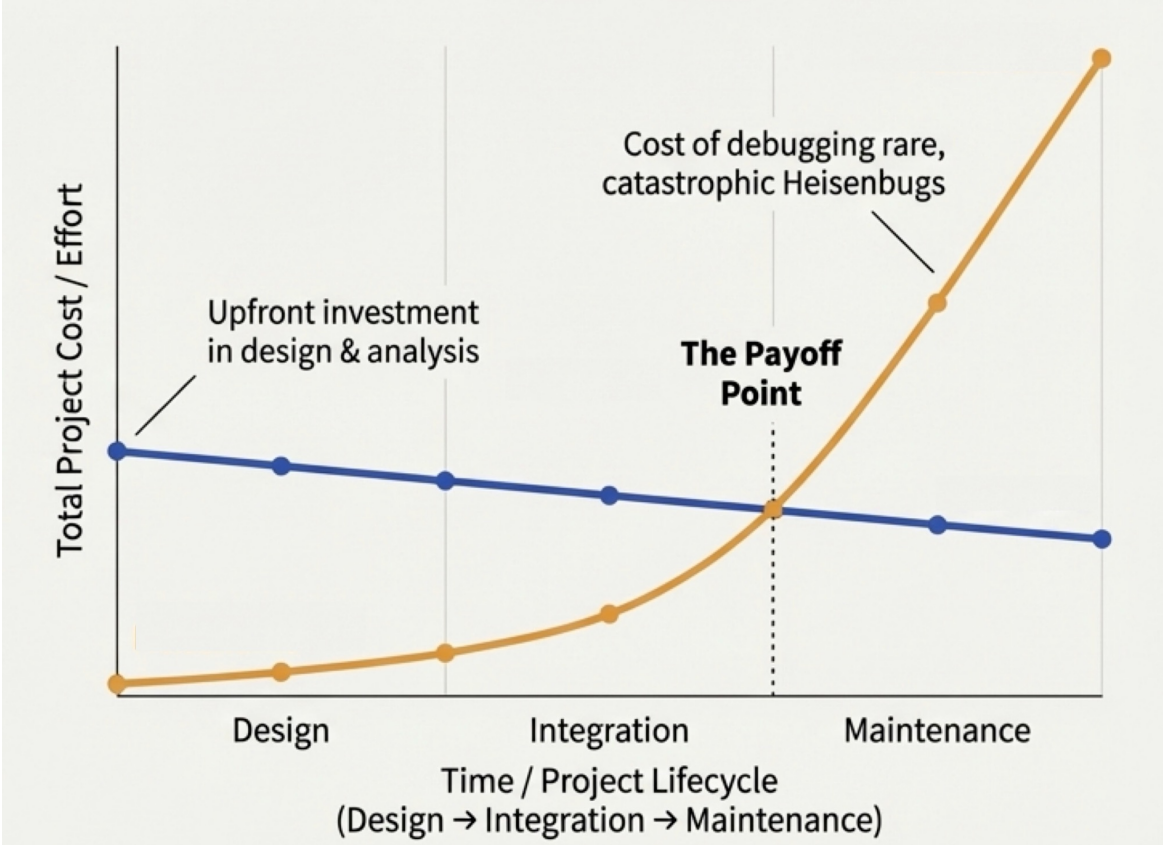
TT exposes existing defects more reliably and improves both correct-by-design (model adherence) and correct-by-testing (bug detection).

Trading Upfront Design Effort for Lifetime Predictability

- Yes, designing a TT system requires more upfront analysis, modelling, and tooling.
- The initial “boilerplate” can feel high.
- Flexible, priority-based systems are often easier to get running initially.

- **The Payoff**
  - **HeisenMiss Cost**: Small initial effort, but an exponentially growing cost of debugging, integration, and re-validation as rare Heisenmisses emerge late in the project or even after deployment.
  - **BohrMiss Payoff**: Large initial effort, but a flat, manageable cost of debugging and validation because failures are deterministic and found early.

- Invest in rigorous design once, or pay the price of unpredictable failures later?



Conclusion

- “Robust” FP systems often mask timing errors via slack.
- This creates **HeisenMisses**: Latent, probabilistic defects that only emerge during the “Perfect Storm” in production.
- **Fail Loudly, Fail Early**: 5% deterministic deadline misses (BohrMiss) in integration is better than a 0.01% chance of a catastrophic miss (HeisenMiss) in production.
- Reliability doesn’t come from hiding deadline misses; it comes from exposing them.
- **Determinism** is a design-time debugging tool, not just a runtime property.
- **The New Mandate**: Design systems that are more sensitive to failure during development.

Thank you

silviu.craciunas@nxp.com

https://scraciunas.github.io/

www.linkedin.com/in/scraciunas

nxp.com