



POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA



EMBEDDED SYSTEMS
Prof. William Fornaciari

Worst-Case Execution Time (WCET) analyses

William Fornaciari
<william.fornaciari@polimi.it>

Summary

Real-Time systems

- Introduction and Classification

Worst-Case Execution Time

- Definition and motivation
- Sources of unpredictability in modern architectures

Traditional Analyses

- Structure of a WCET analyzer
- Architecture model and related unpredictability
- Approximated analyses and timing anomalies

Ongoing research

- Evolution of traditional approaches
- Probabilistic approaches

Real-Time systems

Recap

- In **real-time systems** the correctness of a program depends not only on the logical correctness of the results, but also on the **satisfaction of temporal constraints**
- *Real-time computing* != “Computing as fast as possible”
- **General-purpose systems** have usually the goal of maximizing the *throughput*, while **Real-time systems** are focused on guarantee that the tasks finish to execute before their *deadline*.

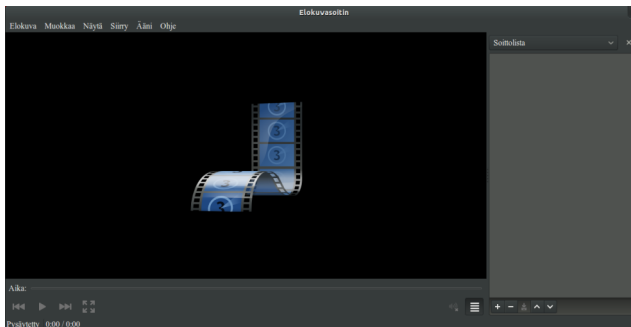
Classification recall

We can classify real-time systems in three main categories:

- 1) **Soft** Real-Time systems: *some* deadline violations are allowed provided that the user experience remains acceptable.

Note: this is a very general definition, the meaning of “acceptable” depends on the specific scenario.

Examples: video players (a lost of a frame is acceptable), audio applications, ...

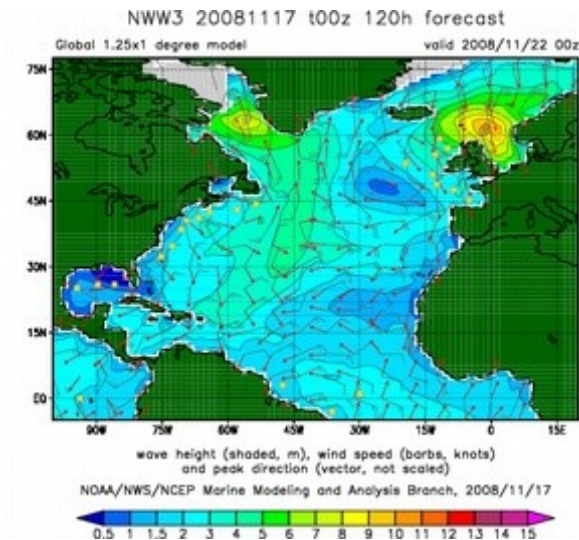
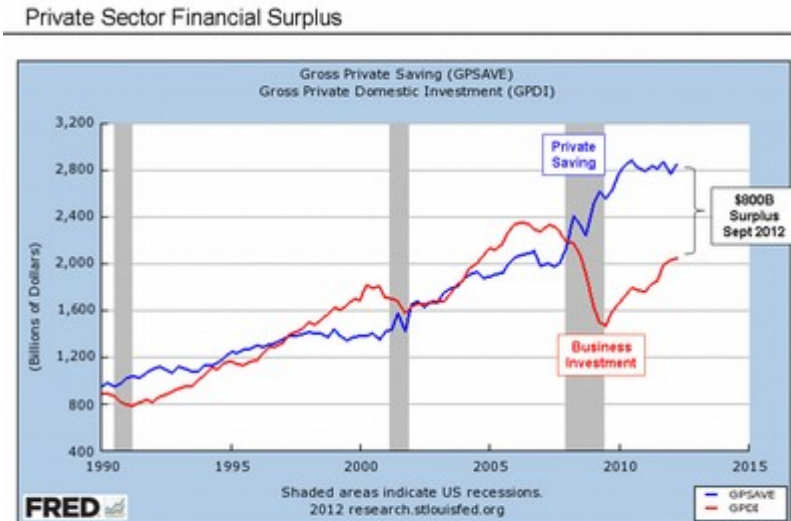


Classification recall

We can classify real-time systems in three main categories:

- 2) **Firm** Real-Time systems: like soft real-time systems few deadline misses are acceptable, however the result has no value and must be discarded.

Examples: financial applications, weather forecast



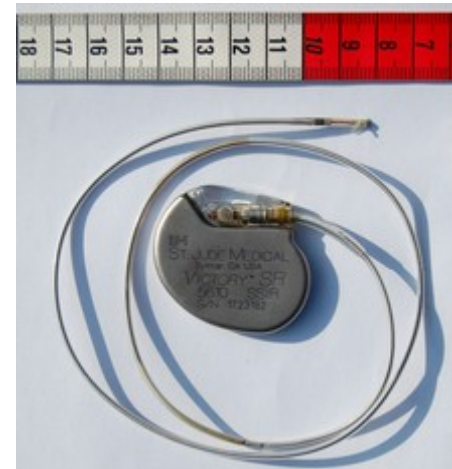
Classification recall

We can classify real-time systems in three main categories:

3) **Hard** Real-Time systems: the tasks must meet all the deadlines.

Note: this is often related to the concepts of mission- and safety-critical systems

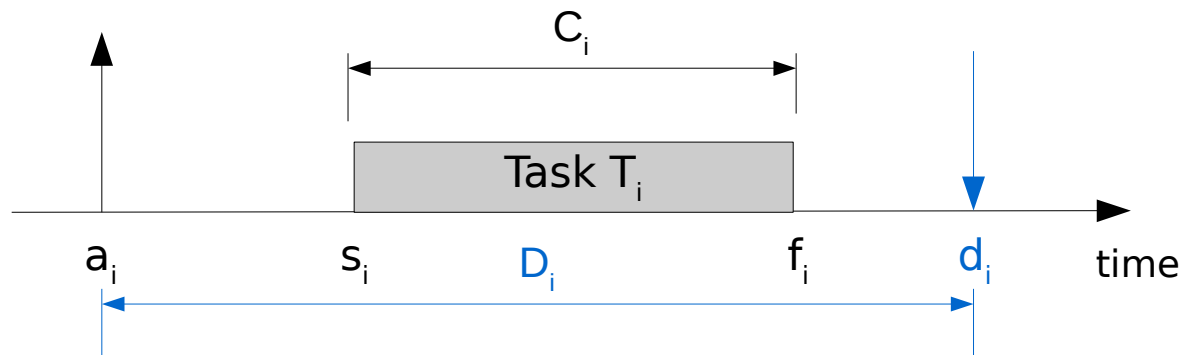
Examples: avionics, medical devices, ...



Hard real-time systems

Introduction to WCET

- Focus on **hard real-time systems** that require strong guarantees on the timing behavior of tasks.
- In order to provide guarantees on the constraints satisfaction we have to compute the **Worst-Case Execution Time (WCET)** for the i -th task:



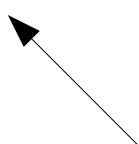
The scheduler must guarantee that f_i is *always* lower than d_i , but it needs to know the upper-bound of C_i (WCET).

WCET Formal definition

A formal definition of Worst-Case Execution Time is:
Given...

- a task τ_i ,
- a set of possible inputs I
- a set of possible machine states S

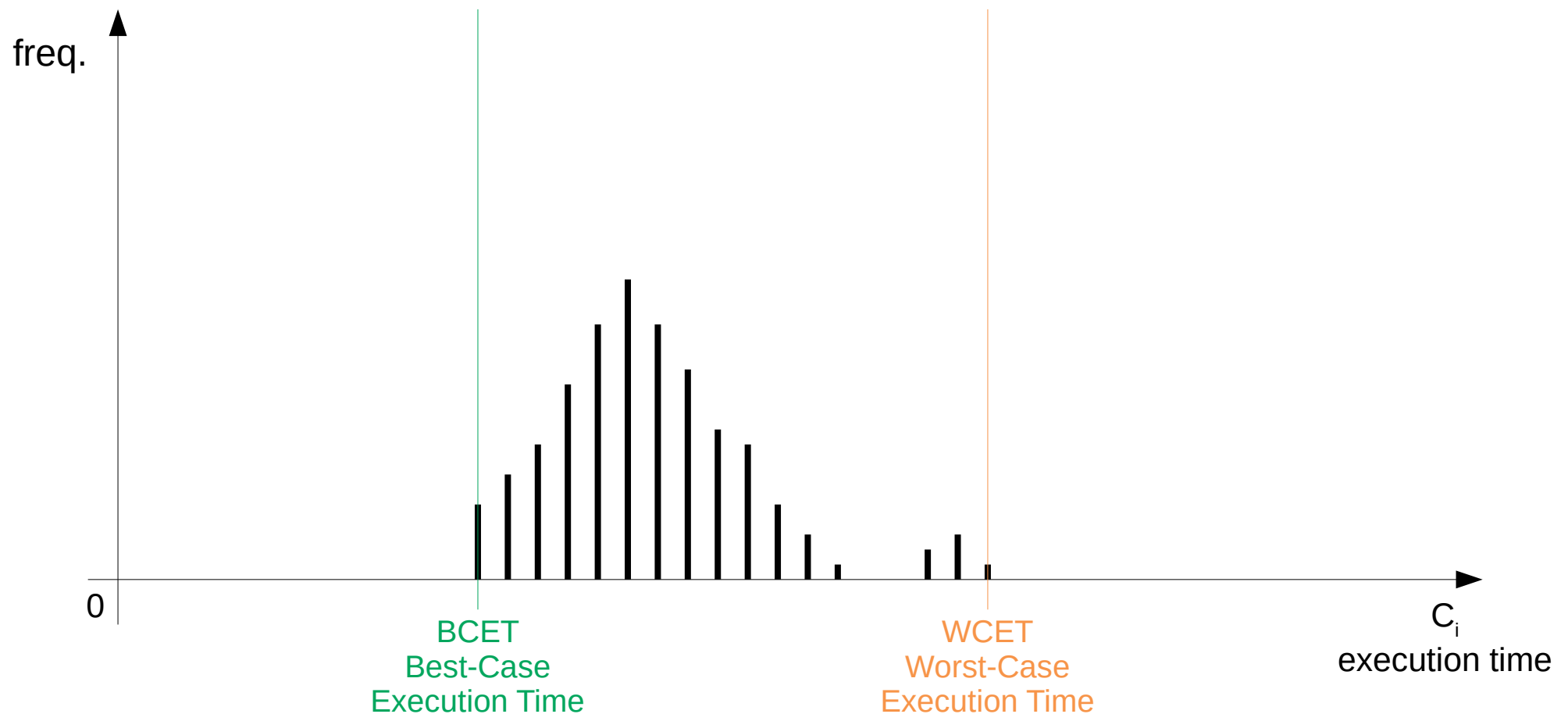
$$WCET_i := \max_{j \in I} \max_{s \in S} C_i(j, s)$$



Execution time of
the task τ_i for input j
and machine state s

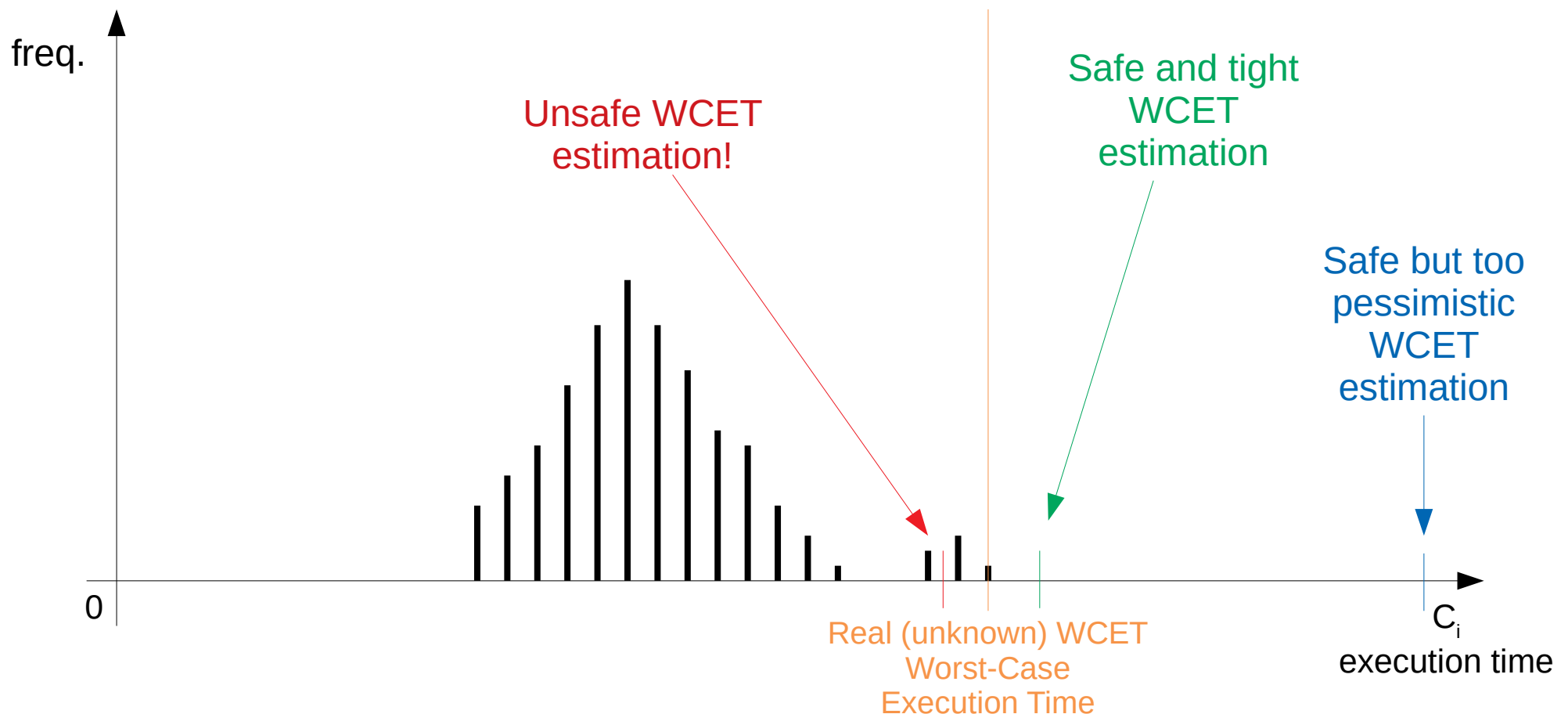
WCET Representation

The Worst-Case Execution Time can be also depicted with a histogram representing the execution time frequencies:



WCET Approximations

Often, computing the real WCET is unfeasible due to large input space (I) and/or large machine states space (S). Thus, approximations are used to estimate the WCET.



WCET Approximations

Why we need approximations?

The input space is often too large

- A complex program contains several branches, jumps, etc. that depend on input values.
- Floating-point operations have different execution times depending on operands.

The machine state space is also too large

- Cache, pipelines, shared bus, shared memory, ...
- Status of peripherals, DMA, ...
- Multi-core and Multi-processors

Moreover, when multiple tasks are running concurrently or simultaneously, their interferences must be assessed.

WCET Computation: multi-tasking issues

Inter-task interferences

On multi-core processors, the simultaneous execution of multiple tasks typically introduces some interferences, with an impact on the turnaround time of the tasks:

- 1) intra-task** interference: self-inflicted delay, e.g. evicting a cache line needed in the future.
- 2) intra-core** interference: a task running concurrently to another task delays the latter, i.e. evicting common cache lines
- 3) inter-core** interference: a task running simultaneously in another core causes delay to other core(s), e.g. trashing a common level of cache or occupying the shared bus

Compute a tight and safe estimation of WCET taking into account the task interferences is hard, especially when multi-core or multi-level caches are present in the system.

WCET Computation: HW issues

Computing the WCET in modern processors include is extremely complex due to the following reasons:

- 1) Memory controller:** on some architecture it can, in theory, take an arbitrarily large amount of time to perform a memory operation.
- 2) NoC traffic:** routing algorithms can not always guarantee a worst-case latency time.
- 3) Access to the Front-Side Bus:** hard to control and analyze the contention to access to the shared bus
- 4) Cache state:** considering shared caches and multiple tasks, the number of possible cache states explode
- 5) Cache latencies:** the worst-case behavior of the cache is sometimes unknown or cannot be guaranteed by the underlying algorithm.

WCET Computation: HW issues

For COTS systems we also have to deal with :

- 1) Power and Thermal management:** techniques like DVFS impact on execution time
- 2) System Management Interrupts (SMIs):** special interrupts handled by the hardware directly (not easy to obtain a worst-case latency)
- 3) Instruction Prefetching:** prefetching improve the average-case execution time but makes the WCET computation harder.
- 4) Speculation:** hard to predict the result of an hardware speculation, it may create silly conditions that increase the WCET.

WCET Computation: HW issues

COTS hardware has several advanced features to improve the “Average-case execution time”, but it’s not easy to be analyzed for WCET.

In fact, most of hard real-time systems of today:

- is single-core
- is simple (no prefetching, no SMI, no speculation, ...)
- has no cache or 1 level of cache
- is composed of stable and old technology

This is especially true when the system requires some sort of certification (e.g. nuclear power plant computers, medical equipments, ...)

How to compute the WCET?

Traditional analyses compute the WCET using the program and architectural knowledge. The WCET clearly depends on the machine you are running your program!

What is it better to analyze:

the *source code* or the *binary code*?

How to compute the WCET?

Traditional analyses compute the WCET using the program and architectural knowledge. The WCET clearly depends on the machine you are running your program!

What is it better to analyze:

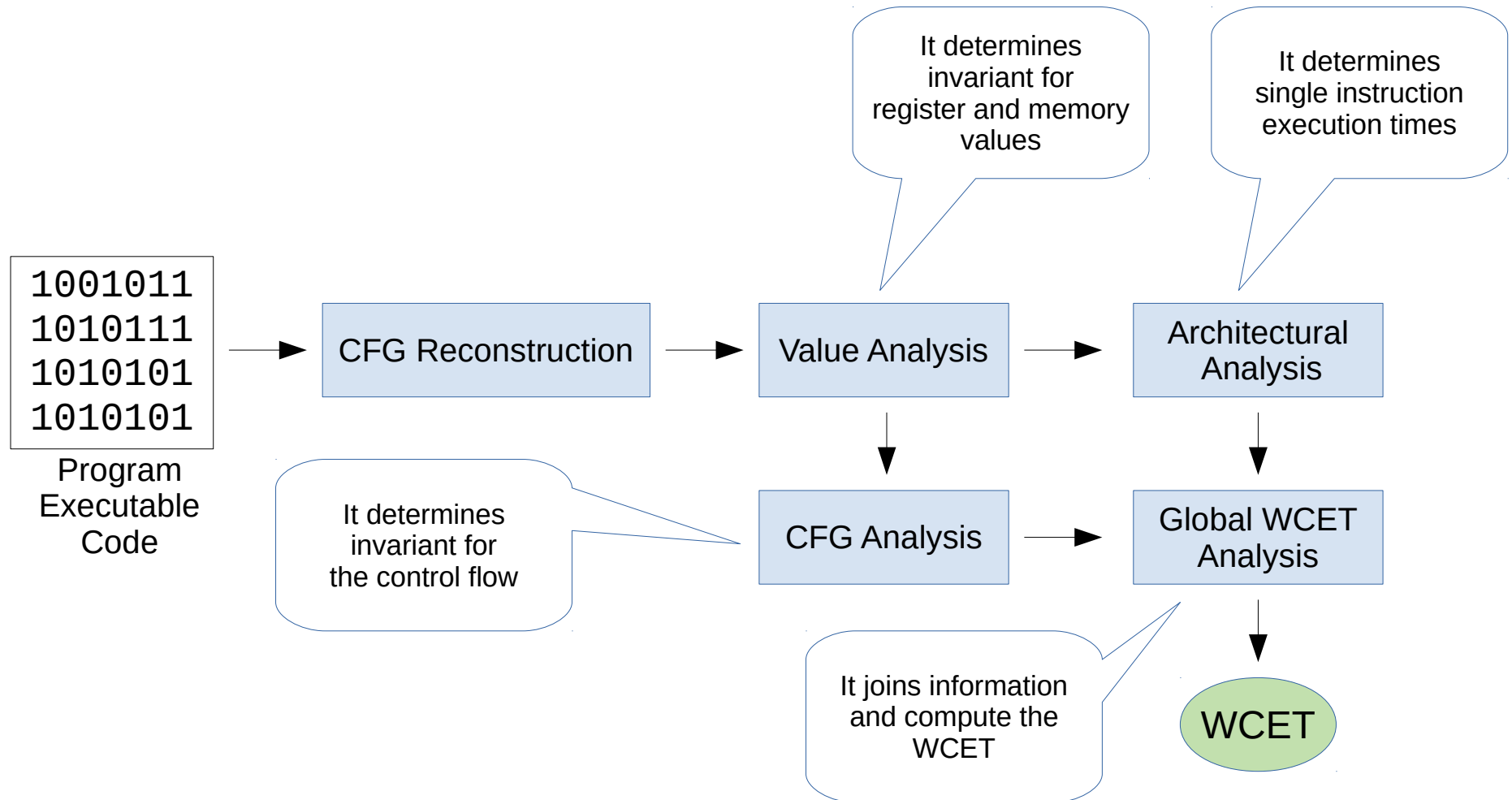
the *source code* or the *binary code*?

The **binary code**:

- It's easier because we already have a one-to-one match with machine instructions
- The compiler may introduce optimizations that can increase or decrease the WCET

WCET analyzer structure

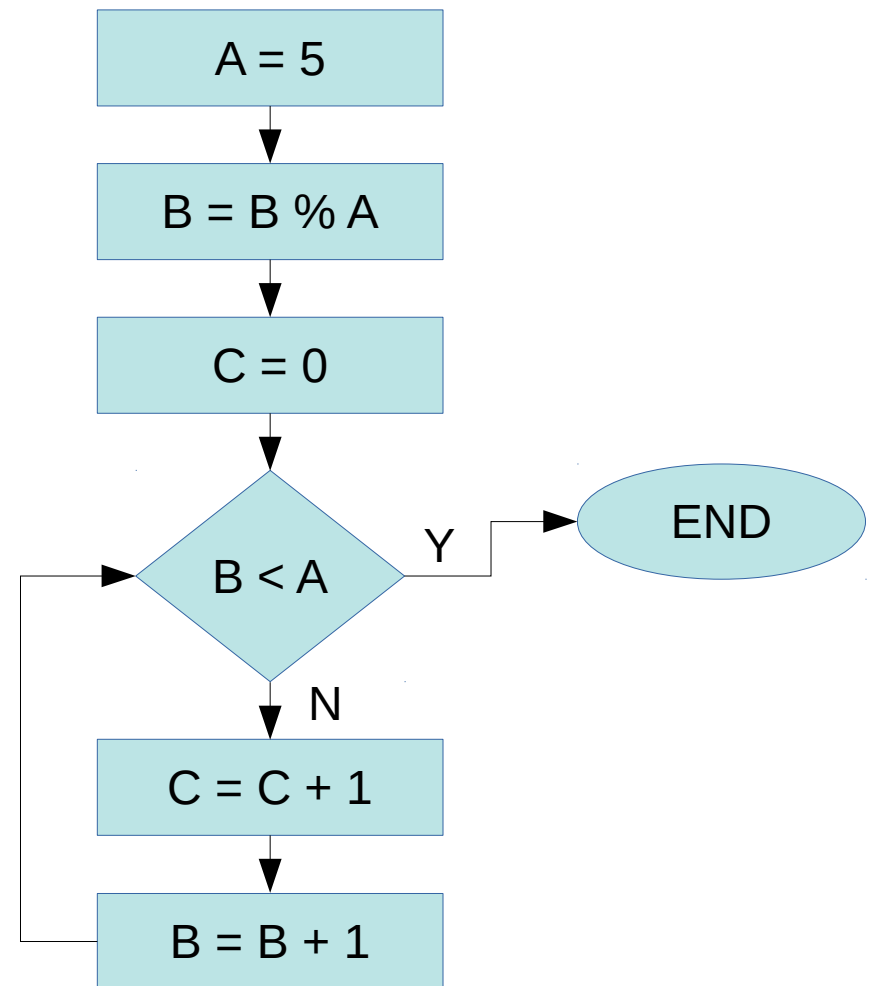
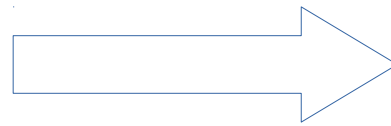
Traditional WCET analysis tools have the following structure:



CFG Reconstruction

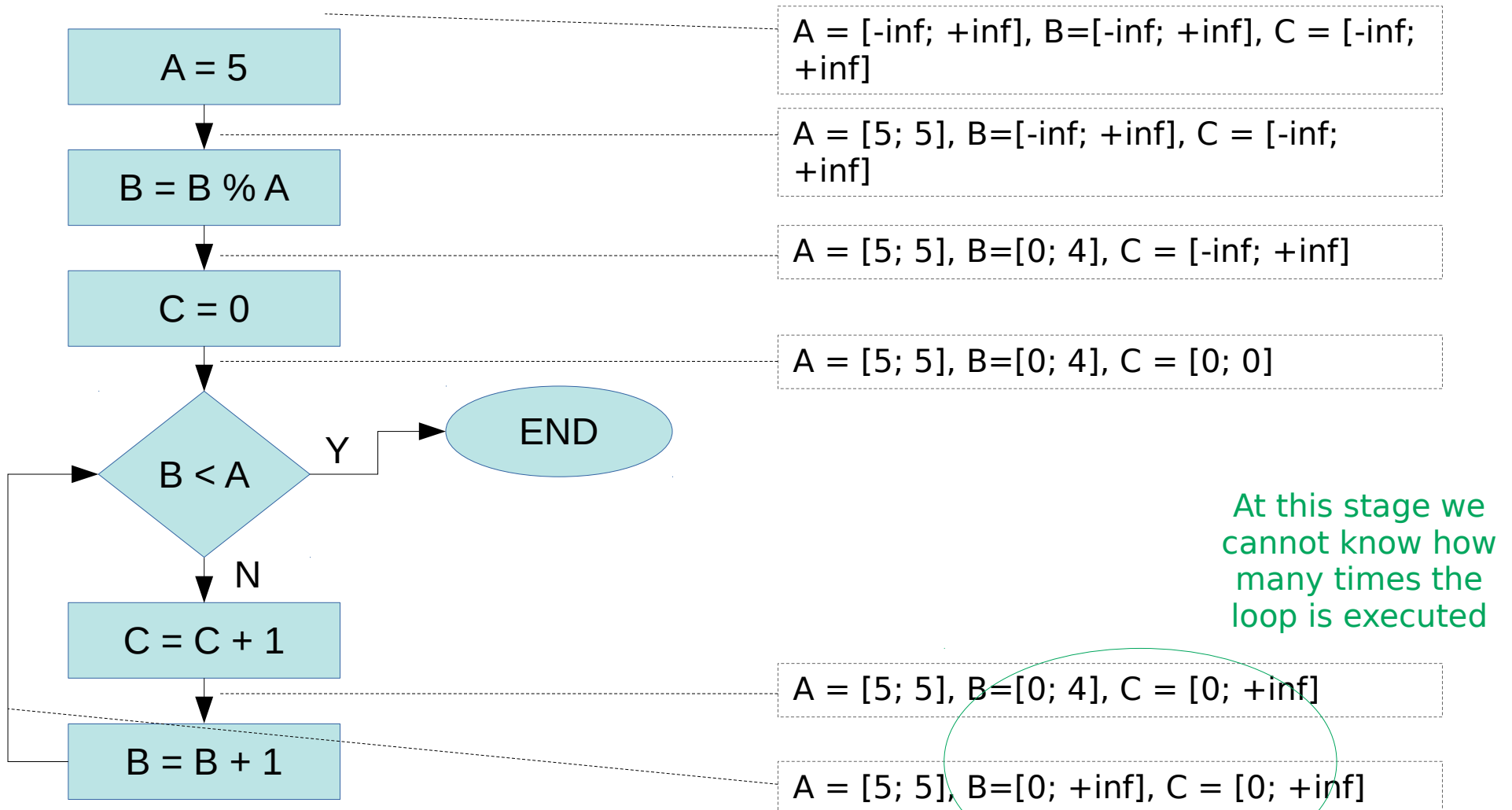
From the binary/assembly code the Control Flow Graph is reconstructed (trivial example):

```
MOV CX, 5
MOV BX, CX
DIV BX
MOV AX, DX
MOV BX, 0
CMP AX, CX
JGE end
INC BX
INC CX
end:
HLT
```



Value analysis

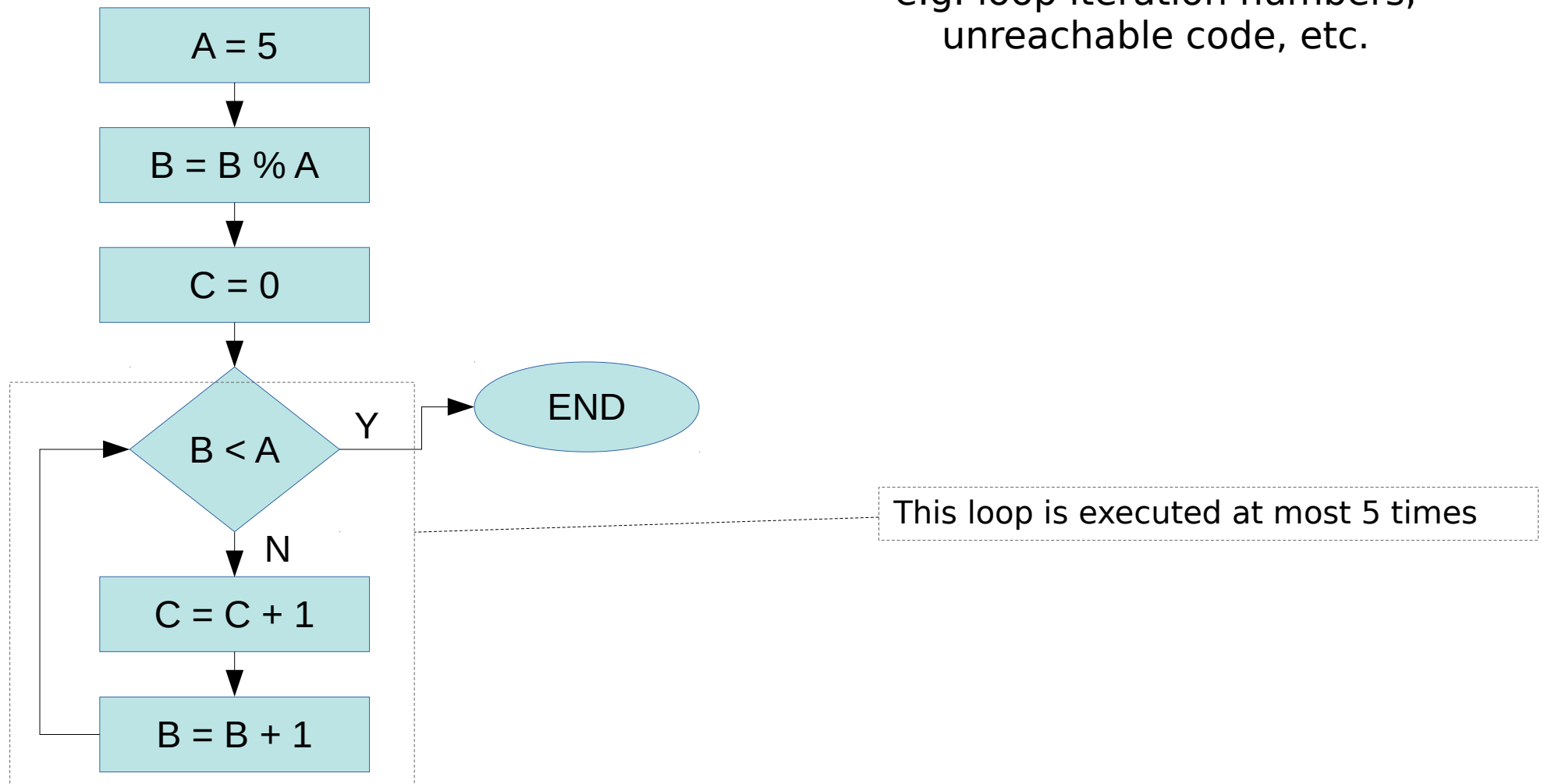
The value analysis aims at finding invariants for variables (both registers and memory locations)



CFG analysis

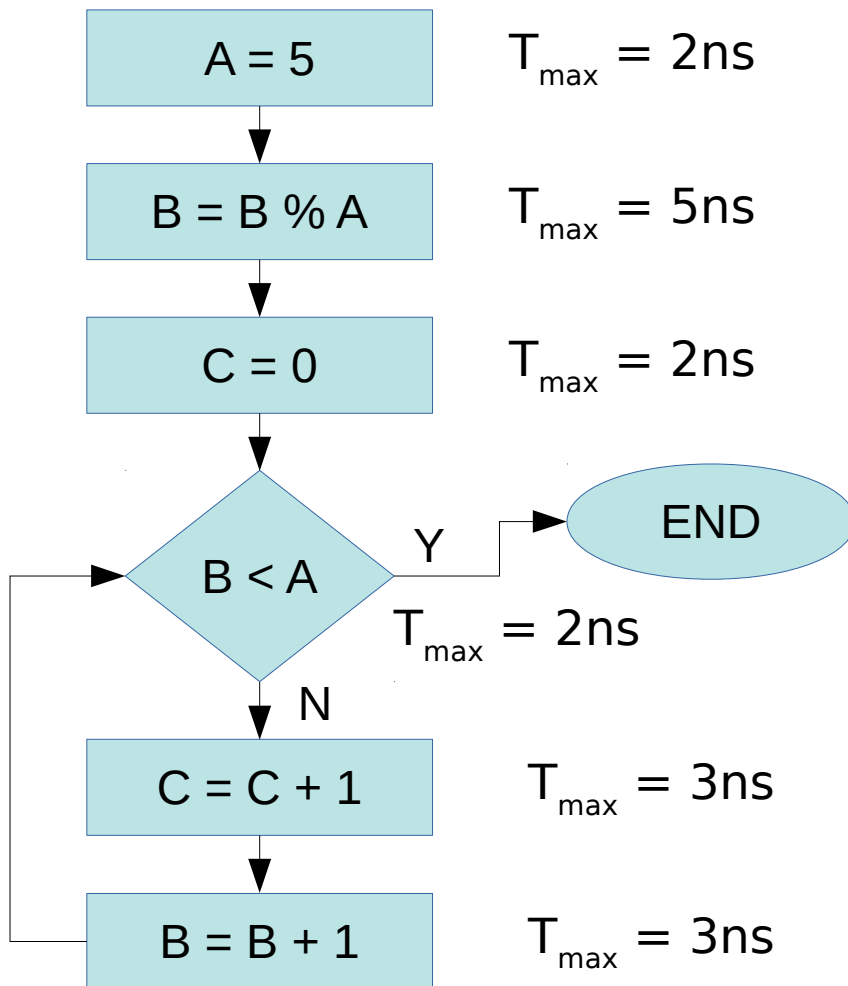
Using the information from value analysis the CFG finds invariants for control flow.

e.g. loop iteration numbers,
unreachable code, etc.



Architectural analysis

The architectural analysis determines the real execution time of the single instruction

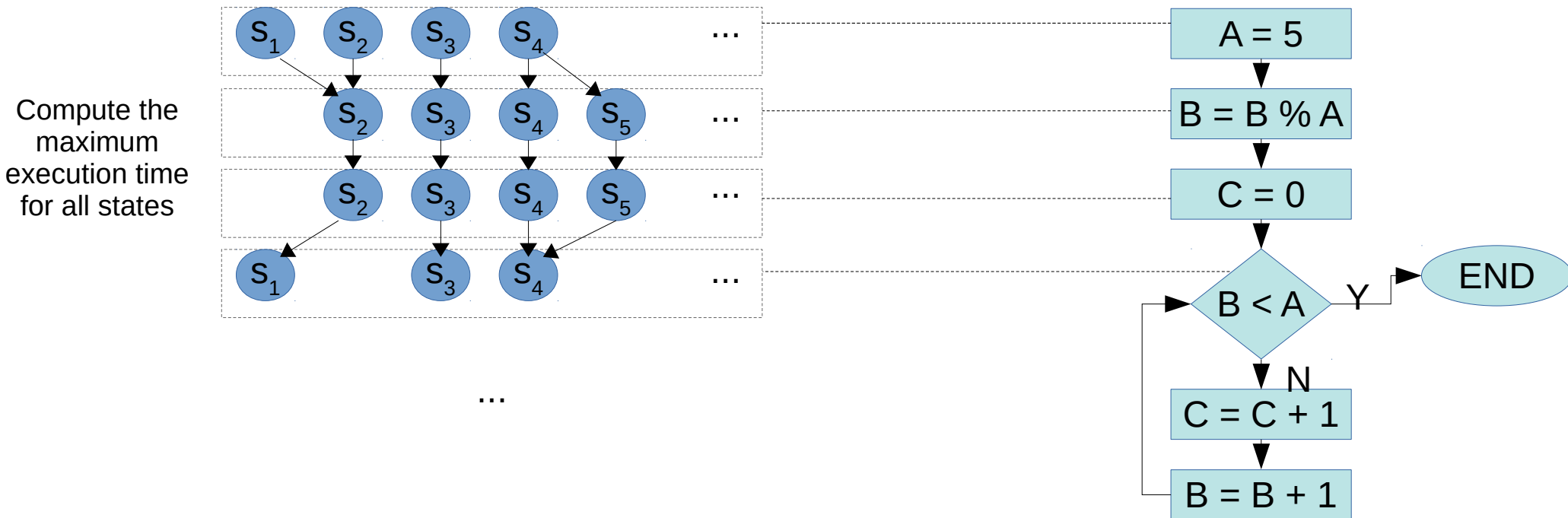


This is an over-simplification!

Architectural analysis

Real processors may have hundreds of possible states that depends on the pipeline, cache status, out-of-order execution, etc.

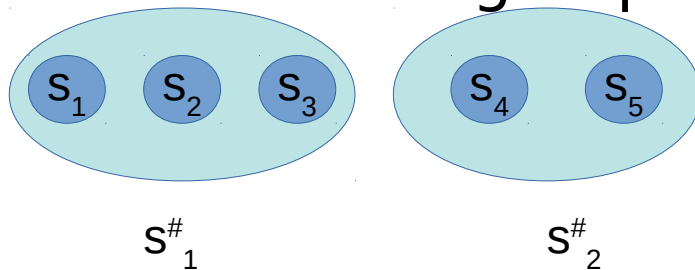
We need to represents the maximum time per instruction for each possible state:



Architectural analysis

However, exploring all the possible states is unfeasible in terms of computational power: **the state space is too large.**

Possible solution: grouping states in approximation sets



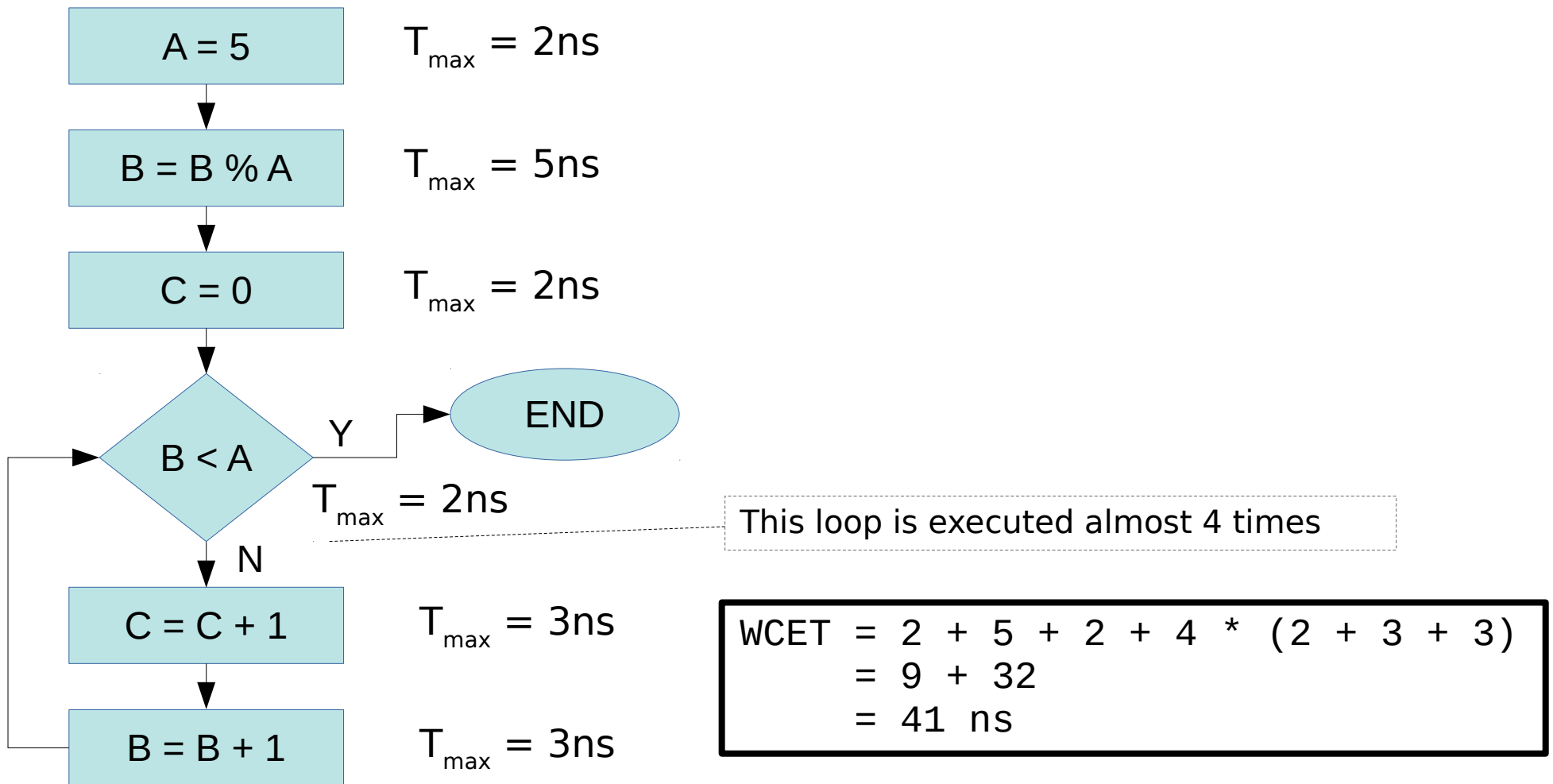
To be “safe” the approximation, we must guarantee that the execution time $T(I, s)$ for any feasible instruction I satisfy:

$$T(I, s^{\#}) \geq T(I, \bar{s}) \quad \forall I, \forall \bar{s} \in s^{\#}$$

In our first case: $T(I, s^{\#}_1) \geq \max(T(I, s_1), T(I, s_2), T(I, s_3))$

Global WCET analysis

At the end, we have to combine all the information of the previous analysis to get the WCET



Issues

The WCET analysis is in general affected by several issues:

- Computing the WCET is the same of solving the *halting problem*, thus it is **undecidable** in general
 - True, however, hard real-time programs are usually simple, short and easy to analyze. You cannot compute WCET for a general program.
- **Scheduler** decisions and task **interferences** clearly impact the WCET analysis and the architectural analysis
- The architectural analysis is computational unfeasible for modern architectures (multi/many-core, multi-level of caches, etc.)
- Architectural approximations are not easy due to **timing anomalies**

Timing anomalies (1/3)

Consider a system with one level of cache that performs an access to a memory location L .

- In first approximation we can consider two architectural states:
 - s_H : the value is in the cache (HIT)
 - s_M : the value is not in the cache (MISS)
- Clearly, the time to execute the load instruction is higher when the processor is in the s_M state
 - Thus, can we safely approximate the miss/hit states always considering a miss as a worst-case?

Timing anomalies (2/3)

Thus, can we safely approximate the miss/hit states always considering a miss as a worst-case?

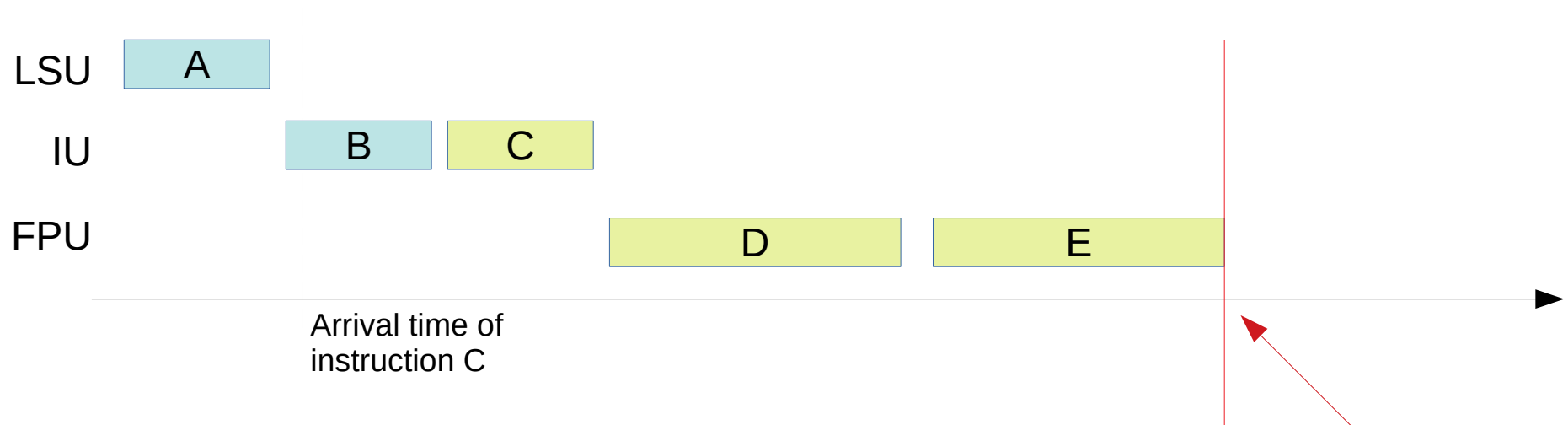
NO!

Why? **Timing anomalies** may happen:

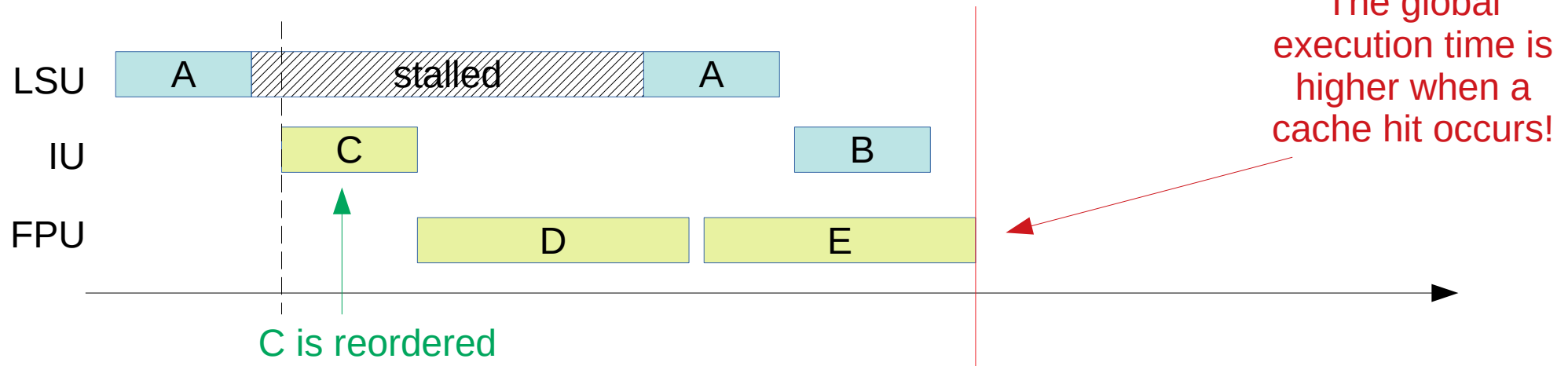
- Consider a processor with 3 functional units: store-load unit (SLU), integer unit (IU), floating-point unit (FPU). It executes 5 instructions A,B,C,D,E with the following precedence constraints:
 - A (load) → B (add)
 - C (add) → D (fmul), C → E (fmul)

Timing anomalies (3/3)

In case of cache hit:



In case of cache miss:



Multiple tasks problem

Obtaining WCET for co-running tasks is a very challenging problem.

We will see the following 4 possible solutions:

- The Murphy approach
- Integrated Analysis
- Partitioning and Isolation
- Time composability

The Murphy approach

Idea

- Compute the WCET of each tasks considering all the possible interferences from other tasks.
- Then, *compute the WCET from the tasks interaction.*

Pros

- Simple

Cons

- Results will be extremely pessimistic
 - Computed WCET may be more than 10x bigger than the real WCET

Integrated Analysis

Idea

- Takes into account all the possible interferences among all the tasks in the entire tasks set.

Pros

- Very high precision in the WCET estimation

Cons

- Computationally unfeasible even for 2 tasks

Isolation Approach

Idea

- Partitioning the resources to avoid the interferences.
- For example, isolating tasks on specific cores or time-division the access to the bus/NoC.

Pros

- Easy to analyze

Cons

- Partitioning is not simple and it requires hardware features
- Utilization of resources drops → resource waste

Time composability

Idea

- 1) Compute the WCET without any interference
- 2) Compute the generated interferences to resources
- 3) Recompute the WCET of each task taking into account the interferences associated to each resource

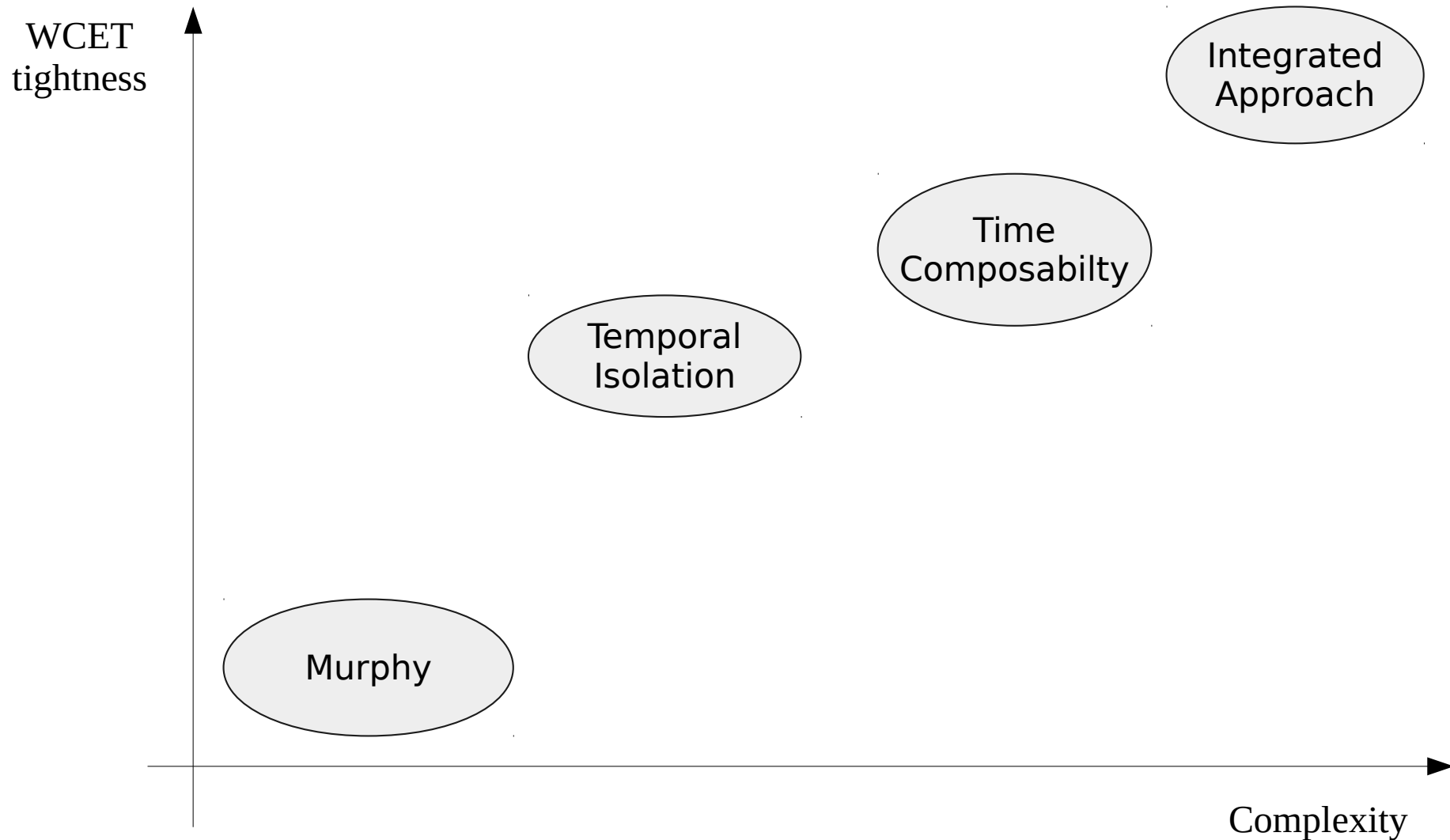
Pros

- Precise and efficient

Cons

- It is extremely hard to correctly perform step 2
- Often based on interference assumptions hard to verify

Multiple tasks problem – Summary



Evolution

WCET computation is hard and very difficult for modern processors having several complex features.

We can identify two trends in last years WCET research:

- The advance the traditional analyses and scheduling
- Using a probabilistic approach

Probabilistic approaches (1/2)

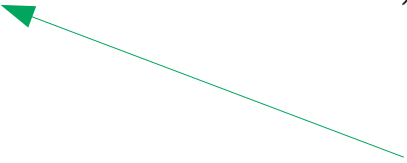
The introduction of probability approaches in WCET analysis is recent technique and still a research topic.

- The idea is to provide a so-called **probabilistic-WCET (pWCET)**, i.e. a statistical distribution that given a desired “probability of failure” **p** it provides the relative WCET:

$$p = P(X > \text{WCET})$$

- For example:

- $\text{pWCET}(p=10^{-6}) = 42.9422$
- $\text{pWCET}(p=10^{-9}) = 46.1358$



Random variable
representing the
execution time

Probabilistic approaches (2/2)

In this way, the WCET is **underestimated** and consequently **unsafe**. However, we have a bound on the probability of the “failing event”: execution time is greater than expected.

When pWCET is used in *mission-* or *safety-*critical systems*, this probability can be added to the the **system failure analysis** that usually considers hardware faults only.

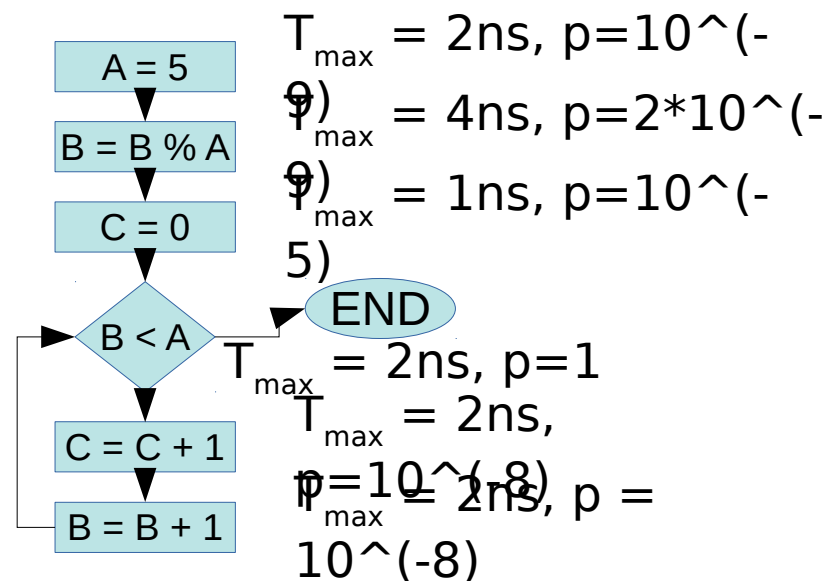
* Probabilistic approaches are still a research topic, they are not ready to be used in real safety-critical systems.

Types of probabilistic approaches (1/2)

There are two main categories of probabilistic approaches:

- **Static Probabilistic Timing Analysis (SPTA)**

- Similar to the previous analysis, but the probability is added to the CFG analysis and/or the Architectural analysis
- Specific statistical operators are used to combine the different execution time estimation

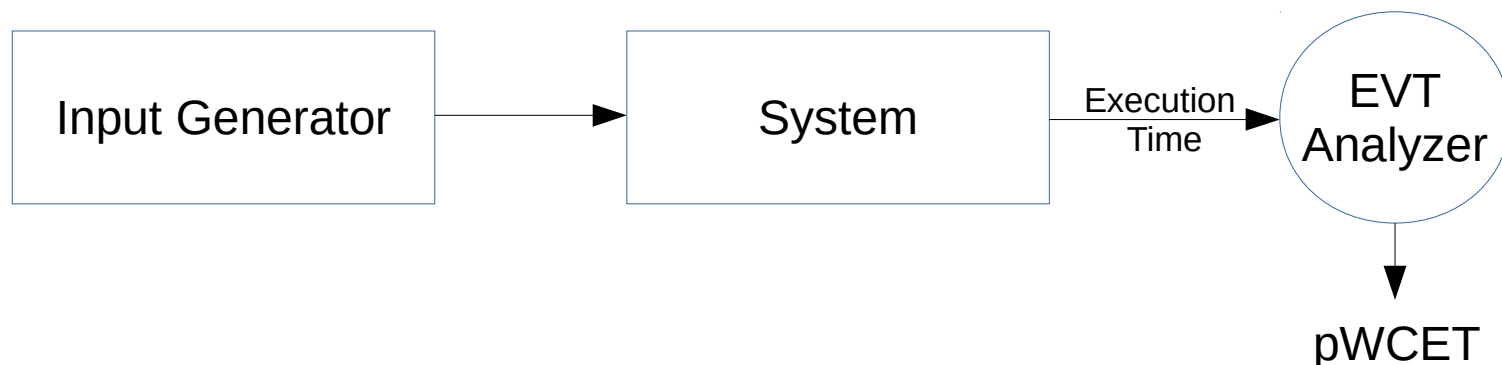


Types of probabilistic approaches (2/2)

There are two main categories of probabilistic approaches:

- **Measurement-Based Probabilistic Timing Analysis (MBPTA)**

- The system is considered as a black-box with respect to the processor states and the execution time is directly measured
- To obtain the pWCET a statistical theory called **Extreme Value Theory (EVT)** is used



(Dis)advantages

PROs:

- Statistical approaches produce a WCET value significantly lower than the one that will be produced by traditional analyses
- The system complexity is less problematic for SPTA and not a problem for MBPTA
 - The effort required to perform the MBPTA analysis is not dependent on the system complexity

CONs:

- Difficult to obtain probability in the execution time of single instructions
- Hard to prove the necessary EVT hypotheses for MBPTA analyses → still far from being possible to certify it for mission- and safety-critical systems

From WCET to WCEC

The WCET is strongly related to another metric:

Worst-Case Energy Consumption (WCEC)

Similar to the WCET, the WCEC is the maximum amount of energy a task can consume to carry out the result. The worst-case scenario usually corresponds to the WCET one.

However, WCEC is influenced by other factors, such as the system power/thermal model, and the power management techniques, such as DVFS.

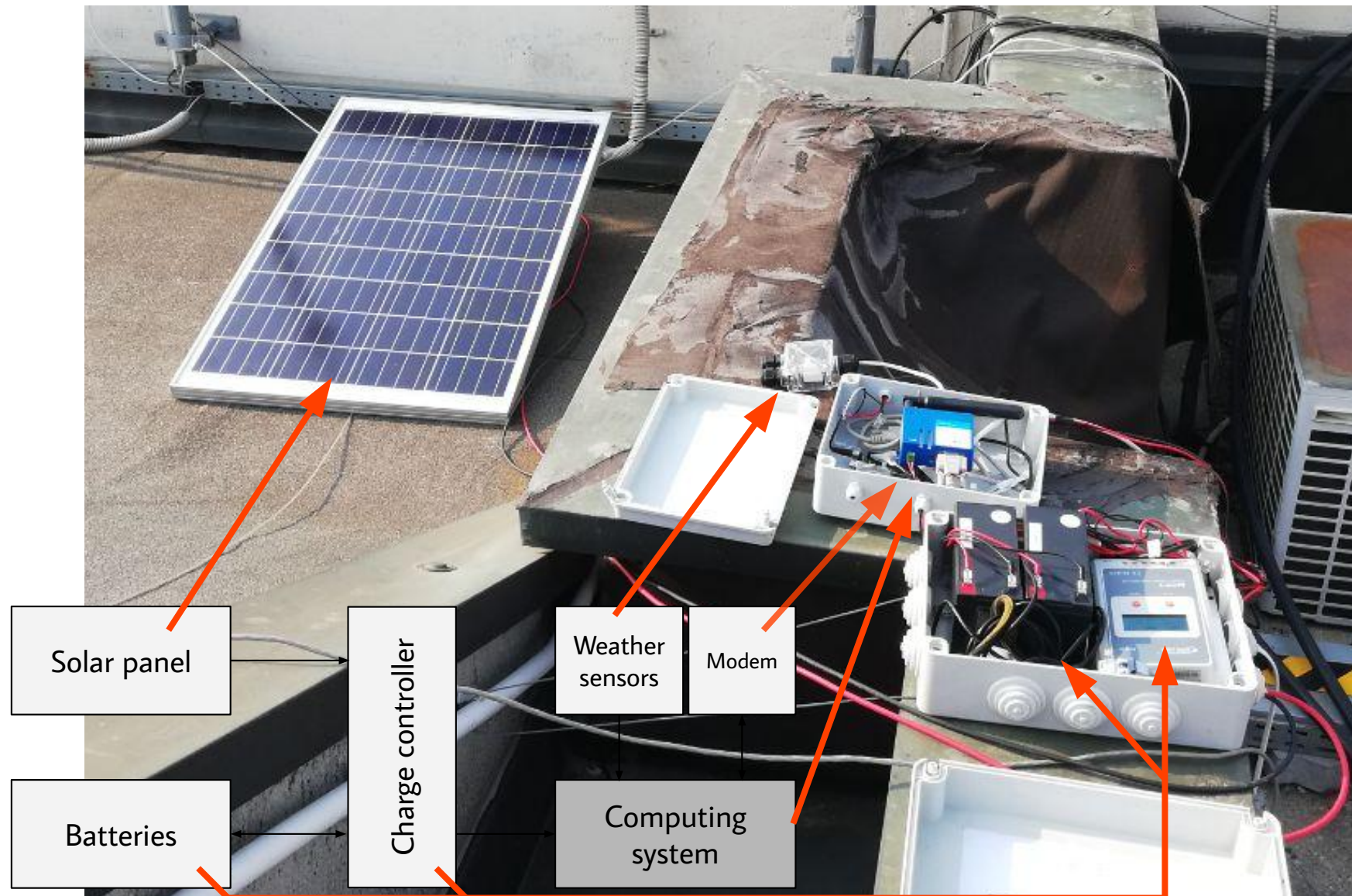
Energy-Constrained Systems

The WCEC is crucial in the analysis of **Energy-Constrained Systems**, that must guarantee to remain online under any condition. This is especially important when the system is harvesting energy from the environment.

Example:

- An embedded system harvesting energy from a solar panel and using a battery as energy storage.
 - A system requirement may be “to survive for a period of time of X hours (days) even in the absence of the solar input power”.

An example from a previous MS thesis



An example from a previous MS thesis

The research questions are:

1) How to estimate the future energy budget, given:

- Remote weather information coming from a weather service
 - Long-term prediction
- Local weather information based on sensors
 - Short-term and local prediction

2) Given the energy budget, how to allocate the task to let the system survive during the night or adverse meteorological conditions?

Interested in these topics?

In you are interested in these topics - and real-time systems in general - we can offer:

- Course projects
- Theses proposal

For further information:

Federico Reghenzani <federico.reghenzani@polimi.it>

Questions?

Contacts

- William Fornaciari
<https://home.deib.polimi.it/fornacia>
william.fornaciari@polimi.it
- **HEAPLab**
DEIB Politecnico di Milano
Building 21, Floor 1
Phone: 9613 / 9623
- **Slides credits:**
 - Federico Reghenzani <federico.reghenzani@polimi.it>

