Timing Analysis of Parallel Real-Time Systems

Luis Miguel Pinho & Tiago Carvalho

CERCIRAS Training School 2023: Advanced Topics in Resource-Aware Computing

Riga, Latvia, September 2023

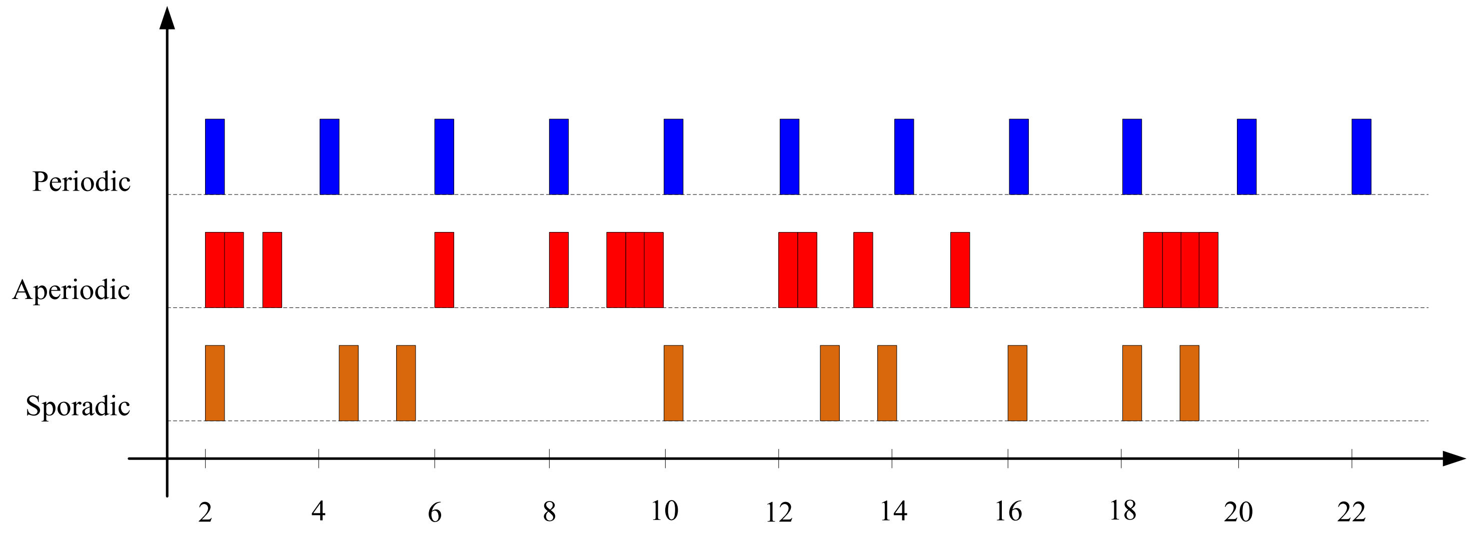
Brief review of real-time systems

Review of Real-Time Systems

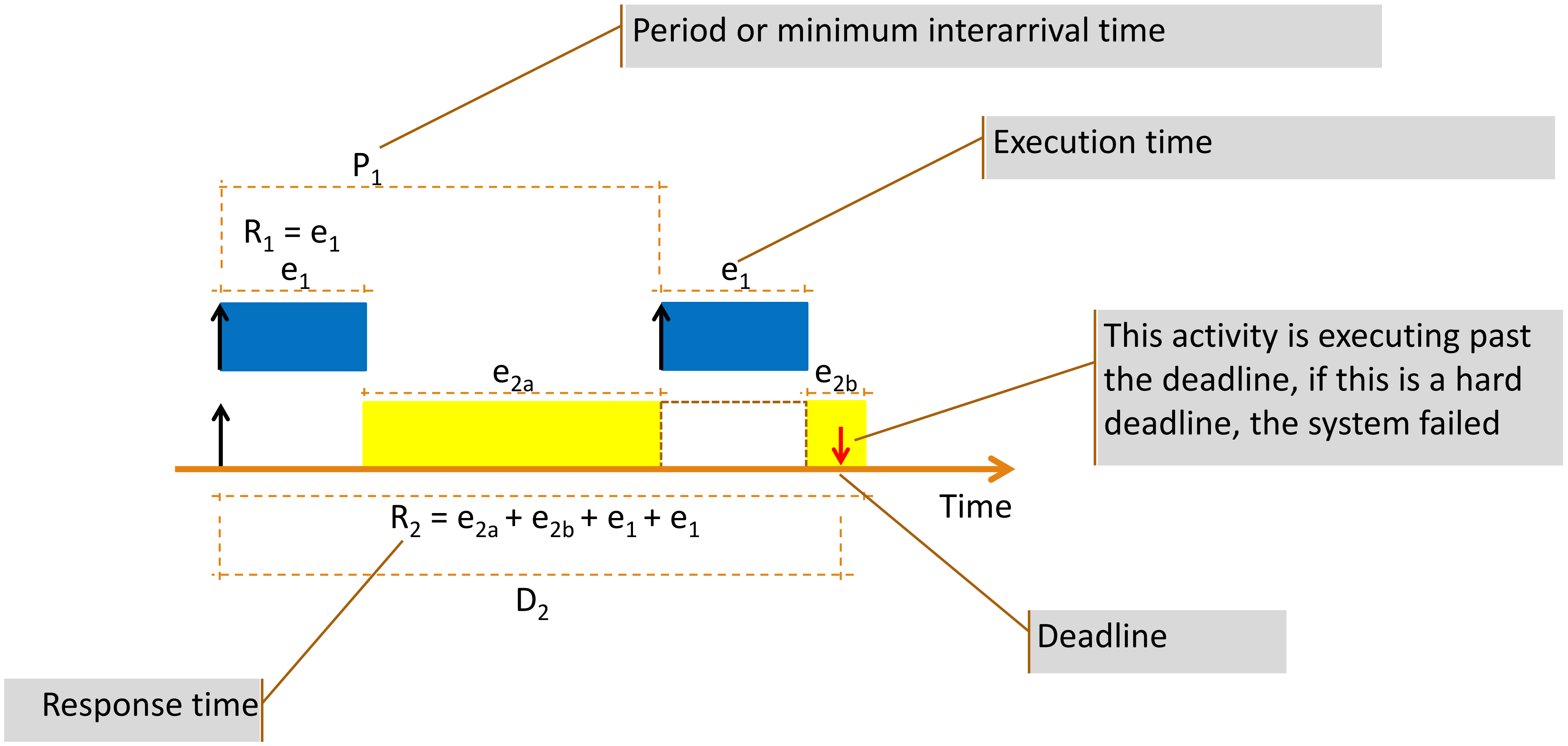
- Distinguishing feature of real-time systems: handling time is fundamental
  - Results need to be provided within specific time intervals (deadlines)
    - Deadlines are requirements (imposed) from applications and systems
  - Real-time is many times an important factor in cyber-physical systems
    - Correction of the computing system, an already important aspect is even more challenged due to the time dimension
  - If deadlines are not met, the system may fail (hard deadlines), which can lead to severe consequences
    - Vehicle not stopping before collision
    - X-ray machine causing too much radiation
    - Trade order after stock price change
    - …

Review of Real-Time Systems

- Managing the interaction of multiple concurrent activities (which may also communicate/synchronize), competing for the same hardware resources
  - Activities can be periodic or non-periodic (i.e., event-driven)
    - Periodic activities repeat regularly after a fixed time interval
    - Sporadic activities have a maximum frequency of repetition
    - Aperiodic activities have no constraints

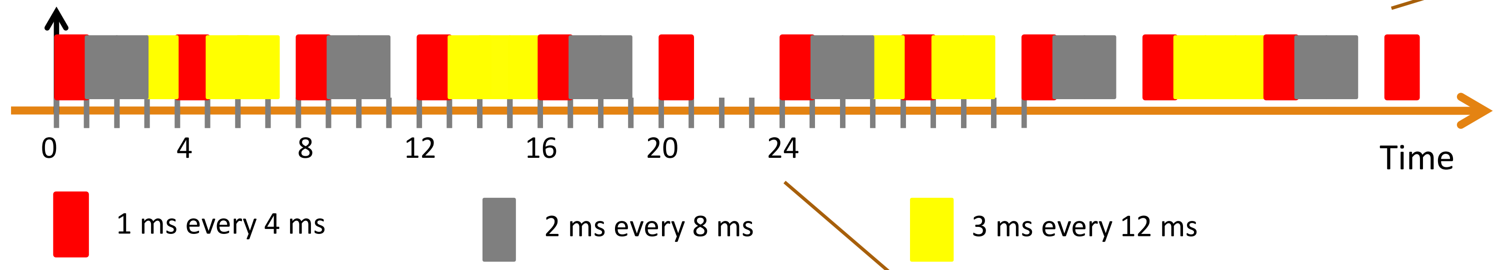


Review of Real-Time Systems


Period or minimum interarrival time
Execution time
P_1
$R_1 = e_1$
e_1
e_1
e_{2a}
e_{2b}
This activity is executing past the deadline, if this is a hard deadline, the system failed
Time
$R_2 = e_{2a} + e_{2b} + e_1 + e_1$
D_2
Deadline
Response time

Review of Real-Time Systems

- The approaches to schedule the concurrent activities in the processors are fundamental to guarantee the correct behaviour of the application
  - Approaches range from fully static cyclic executives, to dynamic scheduling decided during execution

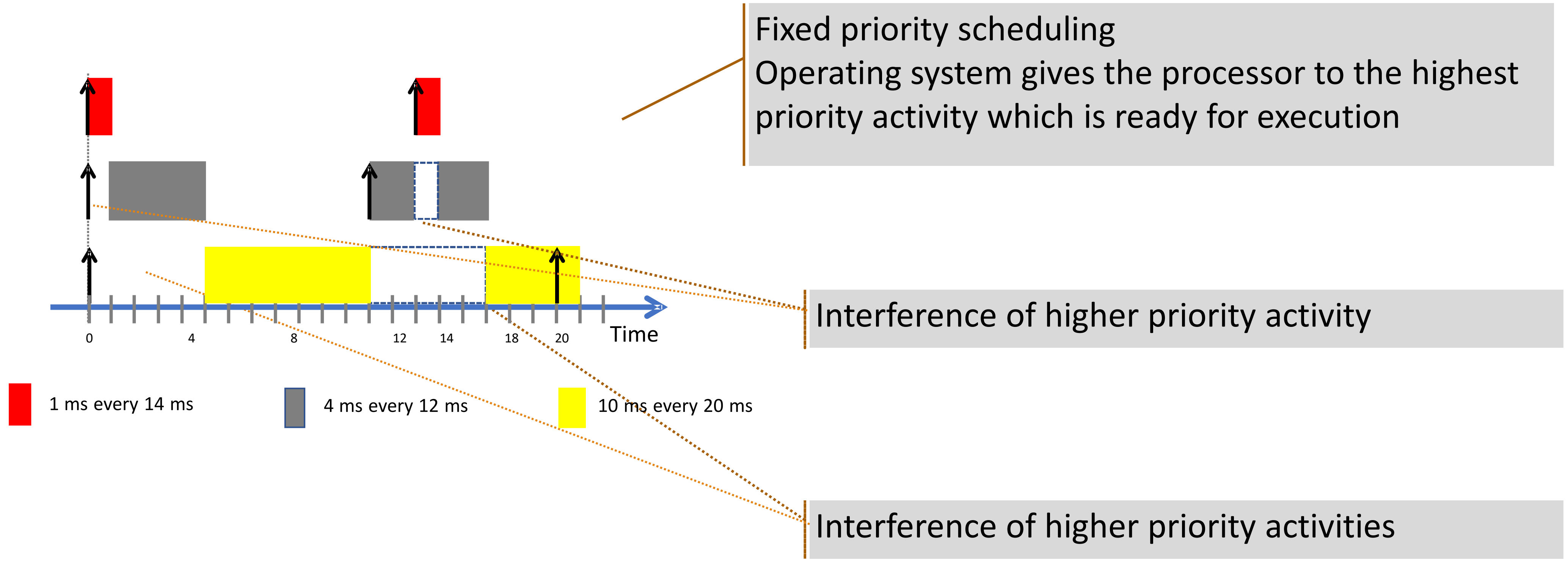


Static cyclic schedule
All activities are executed within pre-allocated slots pre-allocated slots considering their maximum frequency

Repetition starts in the minimum common multiple of activity periods

Review of Real-Time Systems

- The approaches to schedule the concurrent activities in the processors are fundamental to guarantee the correct behaviour of the application
  - Approaches range from fully static cyclic executives, to dynamic scheduling decided during execution



0
4
8
12
14
18
20
Time
1 ms every 14 ms
4 ms every 12 ms
10 ms every 20 ms
Fixed priority scheduling
Operating system gives the processor to the highest priority activity which is ready for execution
Interference of higher priority activity
Interference of higher priority activities

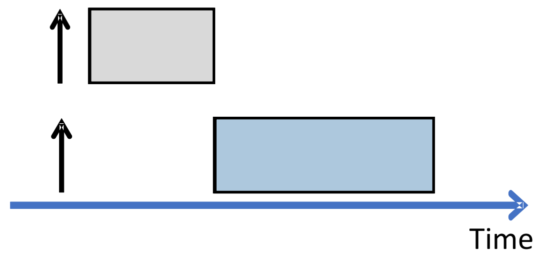
Review of Real-Time Systems

- An important and complementary area of activity is related to the techniques for analysing the timing behaviour of applications
  - Timing analysis – the objective is to compute bounds (or probabilistic profiles) on the worst-case execution time (WCET) of activities
  - Schedulability analyses – the objective is to check analytically, at design or run time, whether all the deadlines will be met
  - Usually necessary to give guarantees of correct time behaviour before deployment
    - Offline analysis –will always meet (hard) deadlines (under a certain confidence)
- Other areas also address scheduling theory
  - Real-time focuses on recurrent activities characterised by period, deadline and WCET
  - The goal is in proving that the timing requirements of the activities are always met
- These analysis are not addressed in this lesson, except on the programming models' support

Real-time and Parallelism

Real-Time Parallel Models

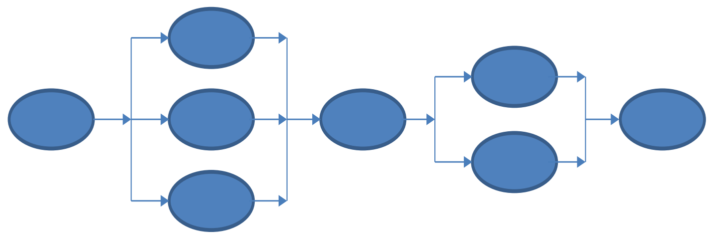
- The trend of parallel computing is also affecting the development of real-time systems
  - Concurrent activities can actually execute in parallel, increasing the interference effects
  - And one activity may also execute with internal parallelism
- One activity no longer specifies a flow of control, but instead, it executes a graph of computation
  - Connected code segments, which may fork or join parallel segments



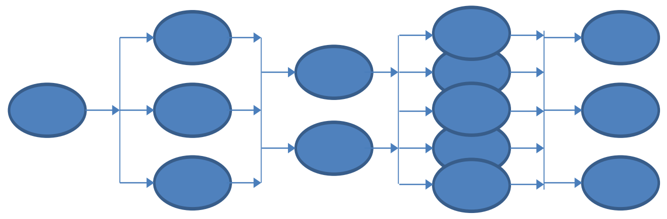
- This means that it is necessary to consider both the concurrency of different activities as well as the parallel execution within each activity
  - Adding to the constraints introduced by the timing requirements and the interaction with the physical world

Real-Time Parallel Models

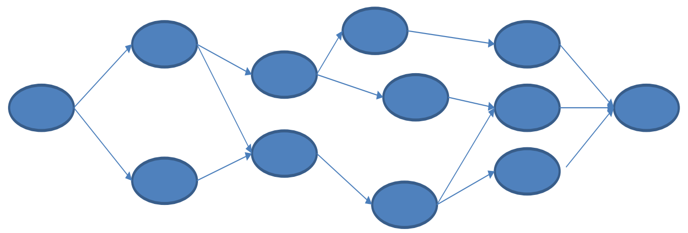
- Significant number of models have been proposed for parallelism inside an activity
  - The Fork/Join model, where parallel segments need to join before a new parallel segment can execute



  - The Synchronous parallel model, which allows different parallel segments without an intermediate sequential one



  - The DAG (directed acyclic graph) model, which allows code segments to spawn and join parallelism in a graph

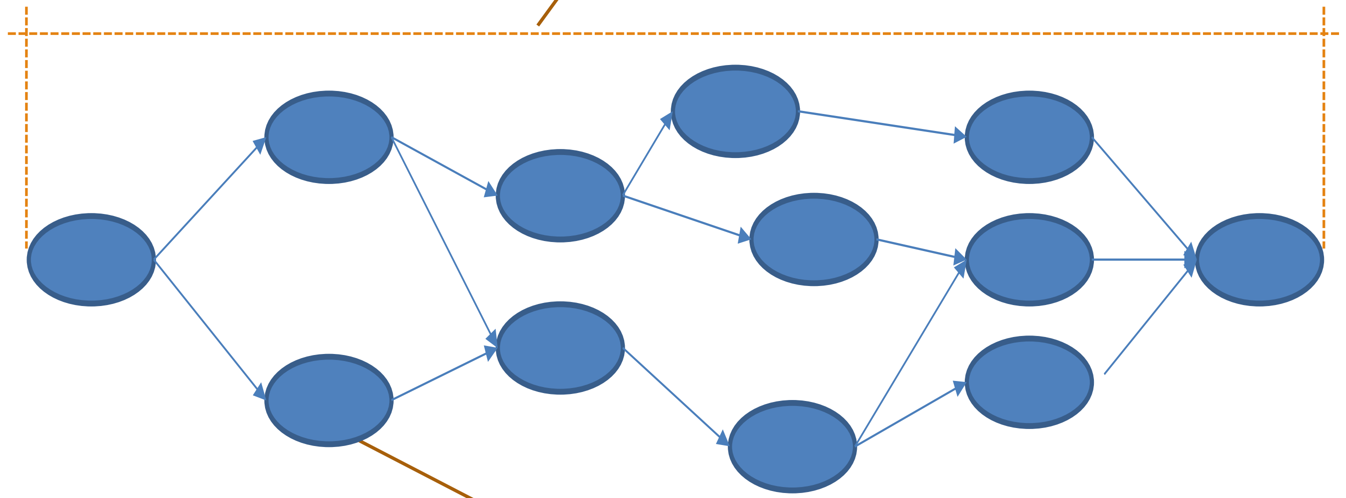


  - Other variants exist, but mostly as specializations of these three models

Real-Time Parallel Models

- Change on activity characteristics

The response time needs to consider the end-to-end interval between the start of the first segment, and the finishing of the last segment

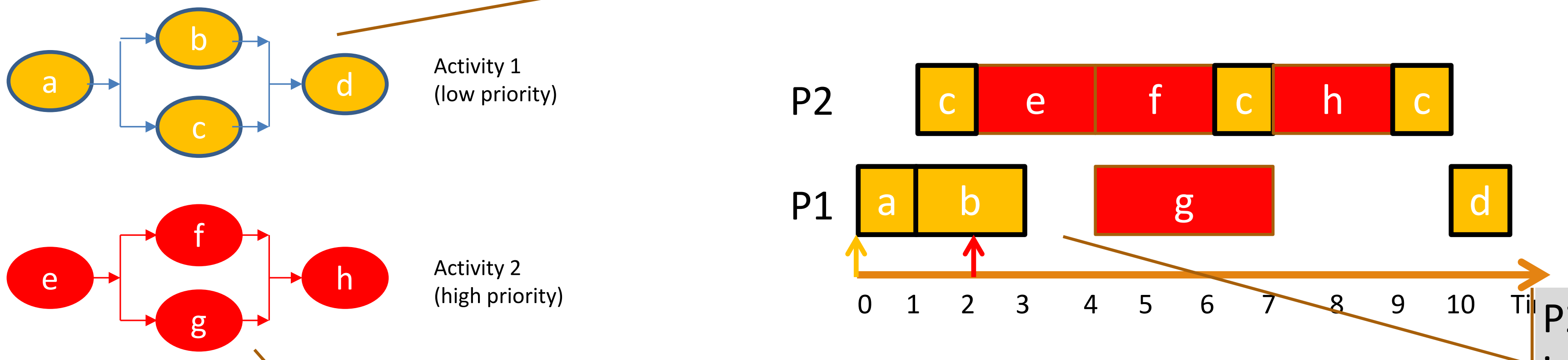


Each code segment has an execution time, the execution time of the activity is the sum of all segments' execution times

This is in a single activity - when considering multiple concurrent real-time activities, the response time will be also impacted with interference and blocking.

Real-Time Parallel Models

- Multiple real-time activities



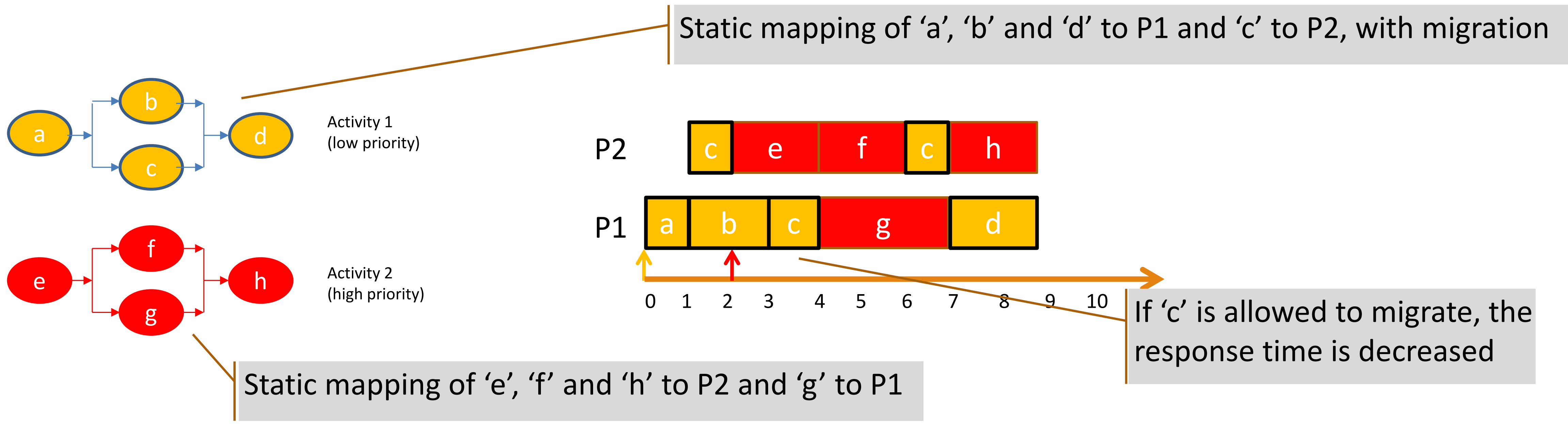
Static mapping of ‘a’, ‘b’ and ‘d’ to P1 and ‘c’ to P2

Static mapping of ‘e’, ‘f’ and ‘h’ to P2 and ‘g’ to P1

P2 cannot execute ‘d’, as ‘c’ is not finished, and cannot execute ‘g’, as ‘e’ is not finished

Real-Time Parallel Models

- Multiple real-time activities



Static mapping of ‘a’, ‘b’ and ‘d’ to P1 and ‘c’ to P2, with migration

If ‘c’ is allowed to migrate, the response time is decreased

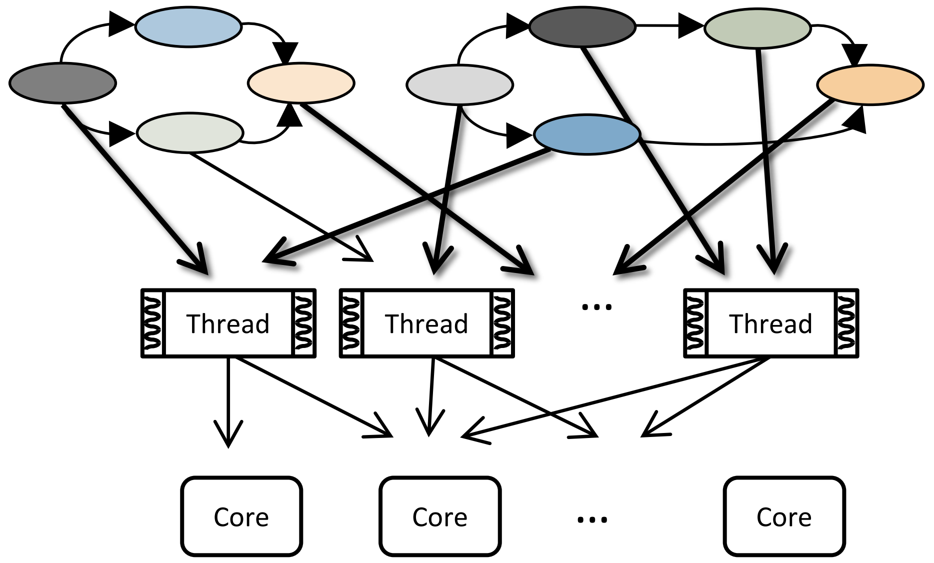
Static mapping of ‘e’, ‘f’ and ‘h’ to P2 and ‘g’ to P1

Mapping and Scheduling

- Scheduling in parallel platforms has been widely studied, and many different approaches exist
  - Fully partitioned – parallel segments are statically mapped to cores, and scheduled in the cores using traditional single processor scheduling
    - Simple to use and higher predictability, but may lead to fragmentation
  - Fully global –segments are globally scheduled, cores execute the next ready segment
    - In principle fully utilizes the processor, but much more difficult to analyse and predict
  - Hybrid approaches
    - Only a subset of segments migrates
    - Migration is restricted within clusters of cores
    - Migration is not allowed after segment starts execution
      - But segments may be dynamically mapped (e.g., work-stealing)

Mapping and Scheduling

- Nevertheless, when considering parallel execution in a platform, there are usually two levels of execution management
  - Mapping the parallel segments to underlying operating system executable units (e.g. threads)
  - Dynamically schedule the execution of the threads to achieve both predictability and high-performance



In practice there are two levels of "scheduling"

Timing and Schedulability Analysis

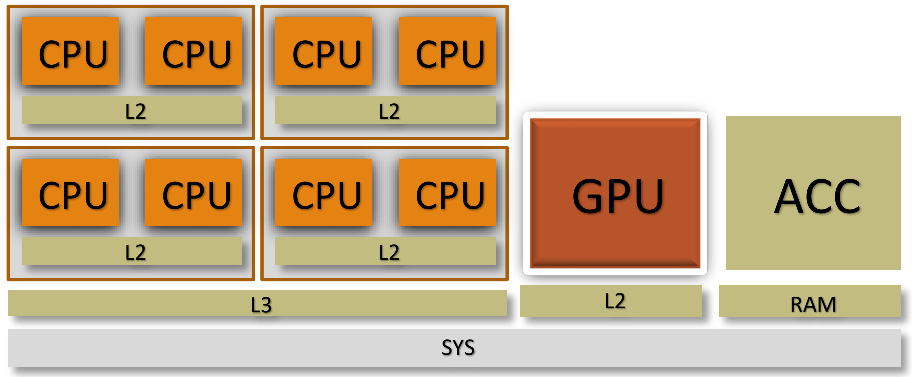
- Parallel platforms and parallel applications introduce significant challenges to the analyses that guarantee correct execution
  - Finding worst-case execution time is extremely difficult
    - Contention of parallel execution when accessing hardware shared resources (e.g., memory)
    - Even when executing in different processors
  - Finding worst-case response time is increasingly complex (and pessimistic)
    - Needs to cope not only with interference from other activities
    - As well as interference from parallel segments of the same activity

Timing and Schedulability Analysis

- Worst-case scenarios take into account corner cases, and need to consider all potential interactions
  - This may cause a significant inflation of the execution and response times
  - Or be intractable in practice
    - Several analyses give response times larger than the sum of all execution times, even on a single activity
  - Predictability is higher if some level of "*staticness*" is provided to the system
    - Static mapping of segments to cores
    - Partitioned scheduling

Heterogenous platforms

- Accelerator devices introduce further complexity
  - Mapping has additional dimensions
    - Even more if variants are allowed
    - Optimizing only for small cases
  - For scheduling, data transfers more important than processor time
    - Parallel transfer and processing
- Analysis is also more complex
  - Timing analysis
    - Static timing analysis bounds are substantially increased
      - Due to complex architecture and black-box nature of many components
    - Measurement based timing analysis becomes much more complex
  - Schedulability analysis
    - Interaction of different models, such as preemptive in CPU and limited preemptive in the accelerator
    - Analysing from bus and memory accesses point of view, instead of processor
  - Easier if some level of “staticness” is applied to the system



Programming Parallel Real-Time Systems

Real-Time Systems Programming

- Programming models and paradigms for real-time systems are built on models of concurrency
  - Dealing with the inherent concurrent nature of the physical systems
- Additional requirements emanate to support the real-time computing model
  - Representation and accuracy of time base
  - Specifying recurrent activities (periodic, aperiodic, servers, …)
  - Specifying activity properties (priority, deadline, execution time, …)
  - Communication and synchronization (share data and synchronize between concurrent activities)
  - Mapping the computation to the underlying processors (static, clustering, migration, …)
  - Controlling the scheduling of computation in the platform processors (global or partitioned, fixed or dynamic, …)

Real-Time Systems Programming

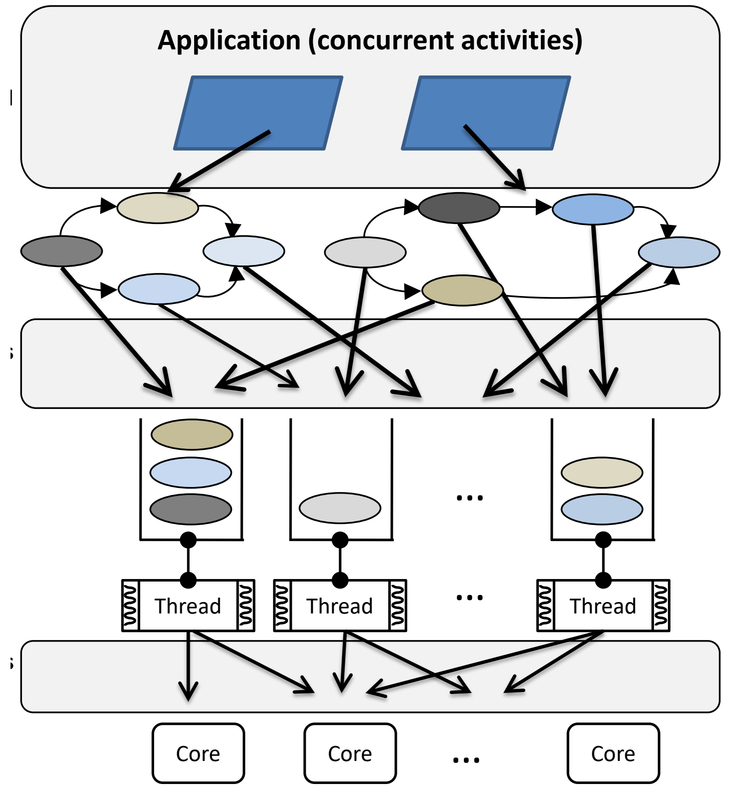
- Two main approaches for the programming paradigm
  - Use of a sequential language together with libraries and operating system implementation the concurrency and real-time features
    - An example is a C program, that uses operating system calls (with the POSIX specification) to handle concurrency and real-time behaviour
  - Use of a language with explicit support for concurrency and real-time
    - An example is the Ada language, which has the notion of activities at the language level (task type) and features in the language for defining real-time behaviour

Real-Time Systems Programming

- Sequential language (combined with library and real-time support) issues
  - The compiler and the operating system are not aware of the concurrency between the different threads, therefore most of the concurrency errors cannot be easily detected
  - Readability of the code is highly impaired, since the concurrent nature of the application, and eventual communication/synchronization, is not directly visible
  - Programs are many times not portable across different operating systems
- A concurrent/real-time language improves readability and correctness
  - Very important to cyber-physical systems and even more in critical applications
  - However, many times different languages are used in different components of the application which is easier if an operating system specification is used
  - And implementing the language runtime on top of different platforms and operating systems may cause a time lag between the availability of a platform, and the required language support
- **If available, the latter is much better!**
  - **Ada rules** ☺

Programming Parallel Real-Time Systems

- When moving from a concurrent model to both concurrency and parallelism, programming real-time systems needs to consider
  - The support to real-time parallel models
  - Time-safe communication and synchronization between sequential and parallel code
  - Different types and levels of mapping and scheduling
- Runtimes and operating systems deal most of the times with the mapping and scheduling of parallel computation
  - But it is important that the programming model allows to specify and control the underlying execution



Programming Parallel Real-Time Systems

- The programmability of parallel real-time systems is still a topic of research and development
  - A few approaches exist, but there is no complete model
  - Two ongoing approaches
    - The use of a subset of the OpenMP tasking model
    - The Ada 2022 parallel programming model
- Both approached are based on a fine-grained parallel model
  - Different perspectives, which reflect in the design differences and availability of mechanisms
    - OpenMP is from the high-performance community, with a very complete parallel programming model, to which real-time needs to be added
    - Ada is mostly the critical and real-time systems communities, providing support for programming real-time systems, to which a parallel model is being added

Timing Analysis of Parallel Real-Time Programs

Challenges

Safety-Critical systems

“Simple” functions $\longrightarrow$ More complex
Programming and design guidelines $\longrightarrow$ No guidelines
Well written and structured code
Simple and predictable hardware $\longrightarrow$ More powerful

$\Rightarrow$ strong and reliable V&V processes and tools **?**

Challenges

Safety-Critical systems

“Simple” functions → More complex

Programming and design guidelines → No guidelines

Well written and structured code

Simple and predictable hardware → More powerful

⇨ strong and reliable V&V processes and tools ?

Business-critical systems

Performance-critical systems

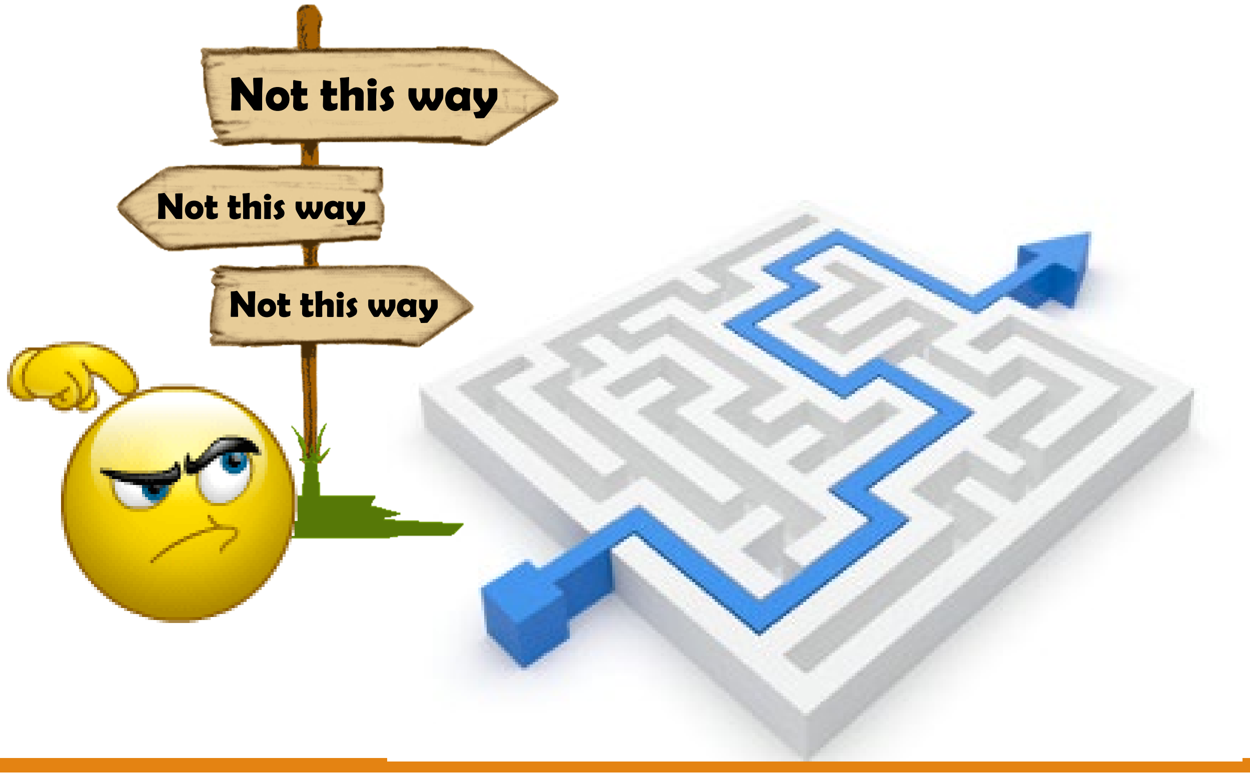
Timing Analysis

- Determining Worst-Case Execution Time
  - Static WCET techniques
  - Measurement-based techniques
  - Hybrid techniques
  - Probabilistic WCET

Static WCET techniques

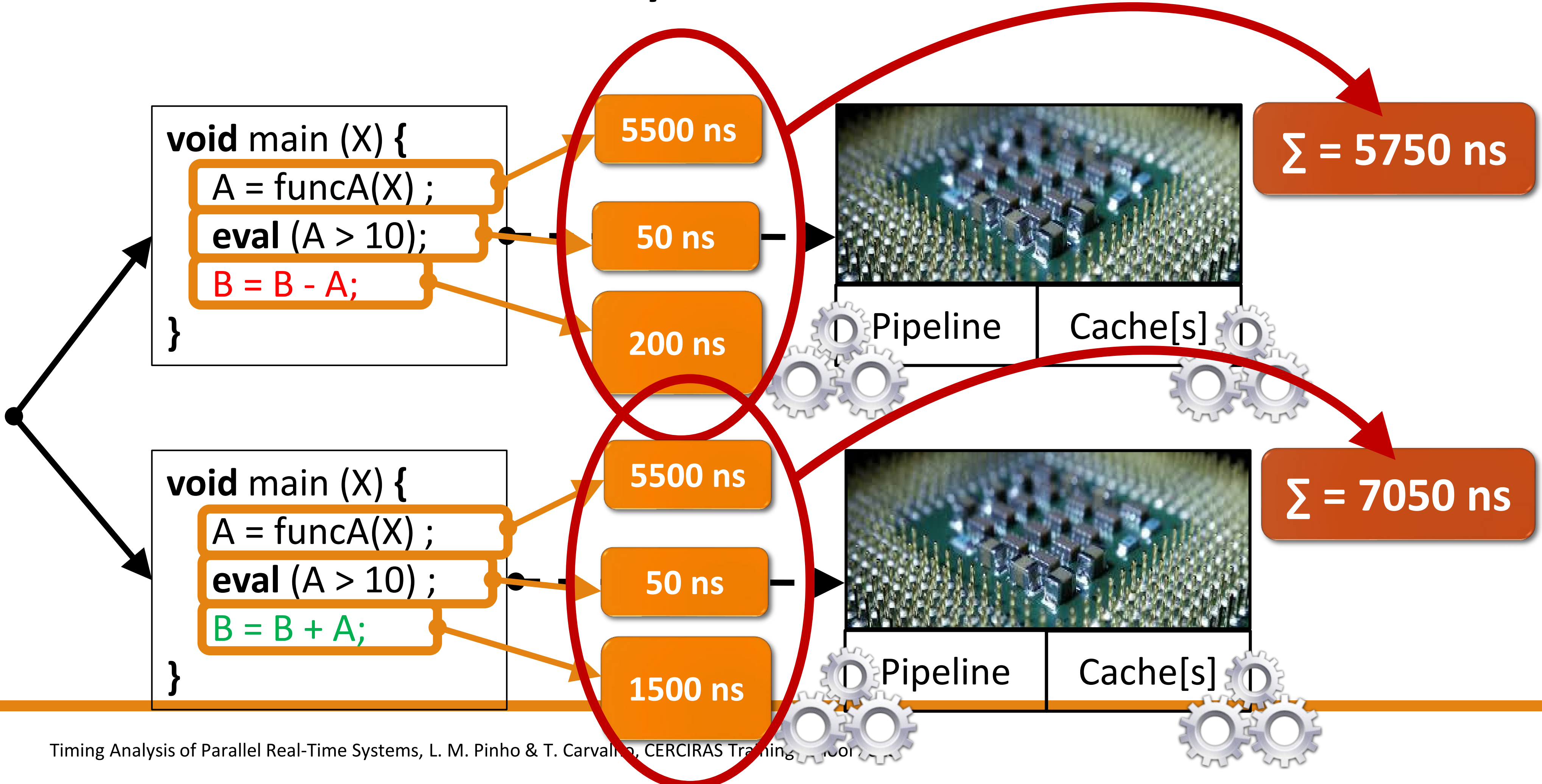
Phase 1: Flow analysis

Identify the feasible execution path in a program



Static WCET techniques

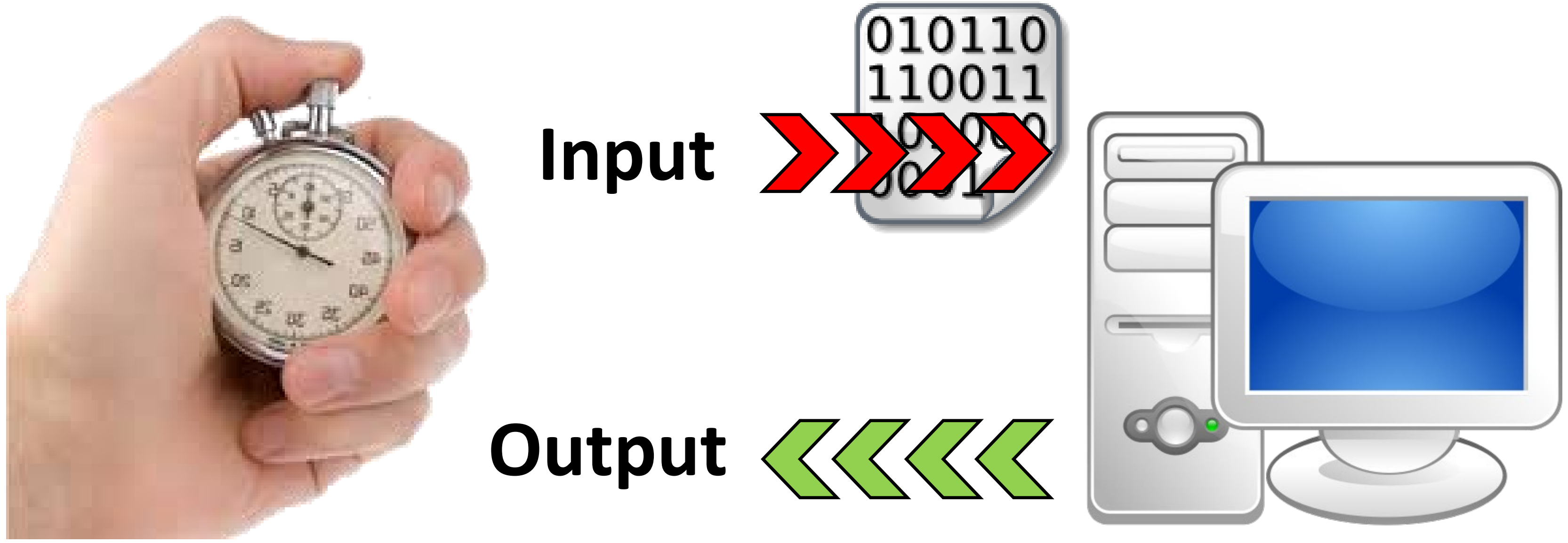
Phase 2: Low-level analysis



Pros & cons

- ➕ No need to have the actual hardware available
- ➕ Years of experience, reliability proven for simple embedded processors -> very efficient for SC applications
- ➖ Little support for multicores
- ➖ Long time-to-market due to the inherent complexity
- ➖ V&V issues and associated cost
- ➖ Accuracy for more complex platforms? (how to deal with IPs in COTS?)

Measurement-based techniques

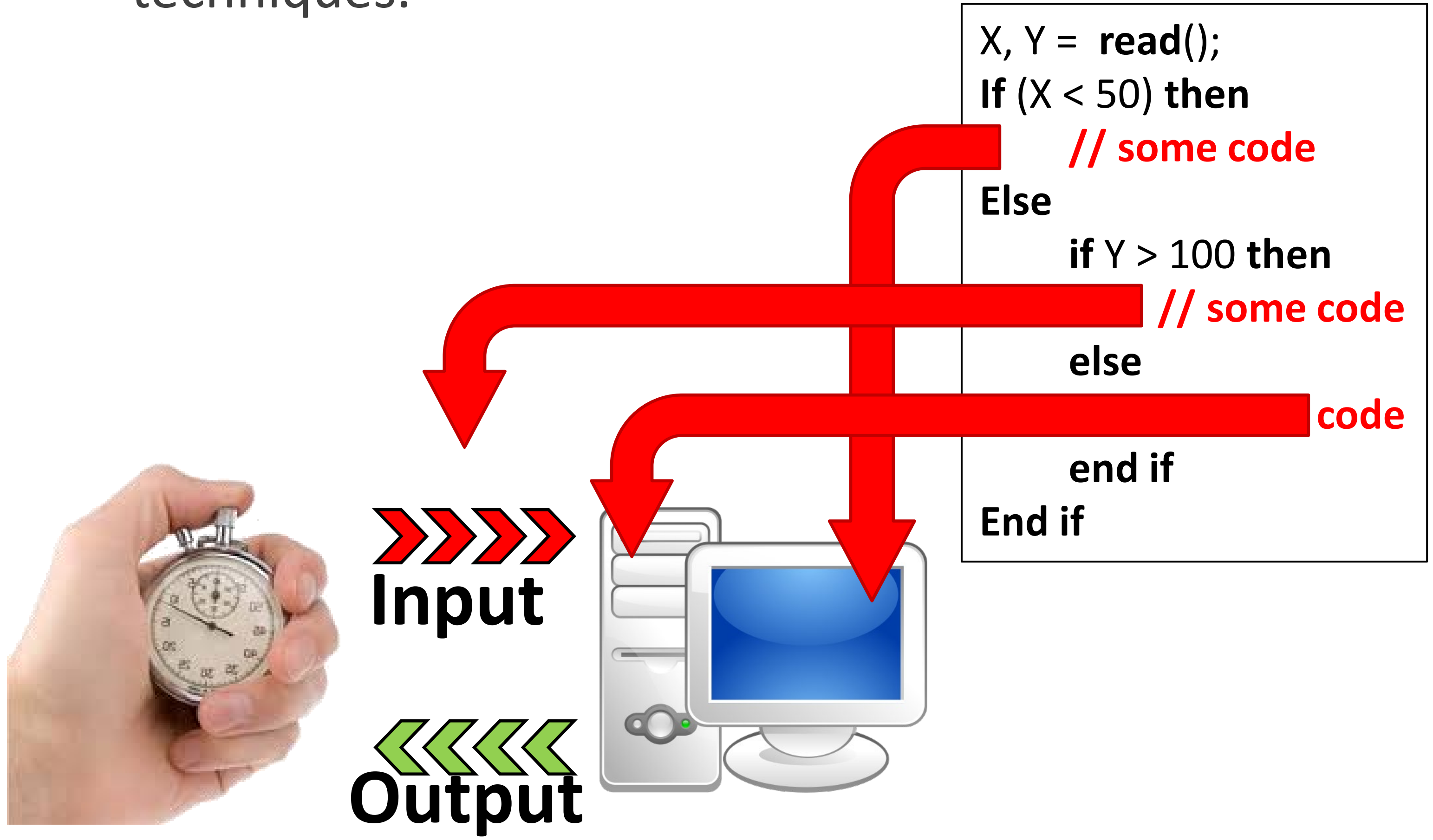


Pros & cons

- ➕ Estimations available immediately (also, average, etc.)
- ➕ No need to design accurate model -> reduced effort and cost
- ➖ Requires the hardware to be available, which may not be the case if the HW is developed in parallel with the SW
- ➖ Difficult to set up an environment which acts like the final system
- ➖ Intrusive instrumentation code
- ➖ Exhaustive testing is impossible

Hybrid techniques

- Combine the merits of static and measurement-based analysis techniques.



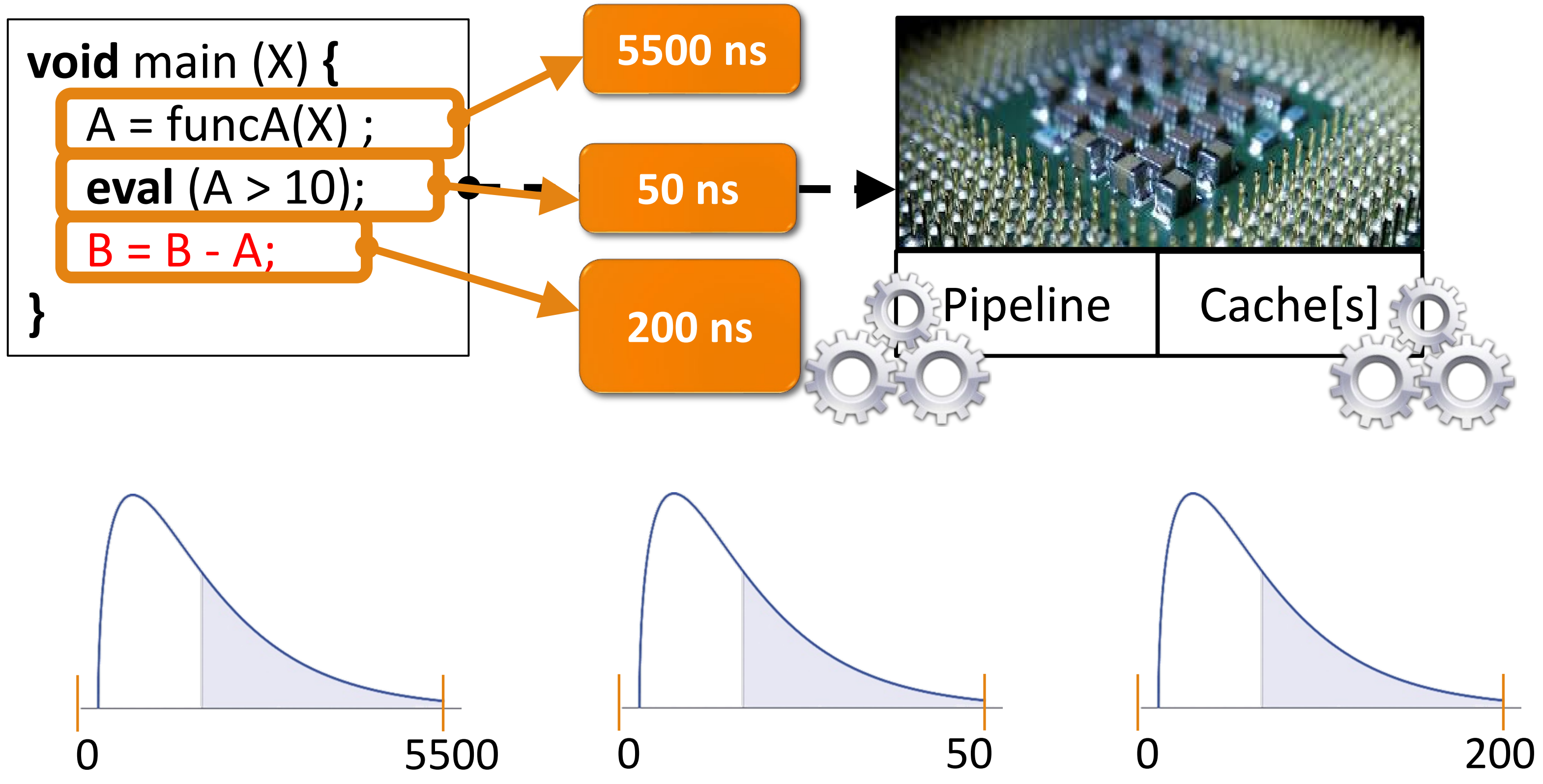
```
X, Y =  read();
If (X < 50) then
        // some code
Else
        if Y > 100 then
                // some code
        else
                code
        end if
End if
```

Pros & cons

- ➕ Do not rely on complex models
- ➕ Provide accurate estimates
- ➖ Intrusive instrumentation code
- ➖ Exhaustive testing is impossible

Probabilistic WCET techniques

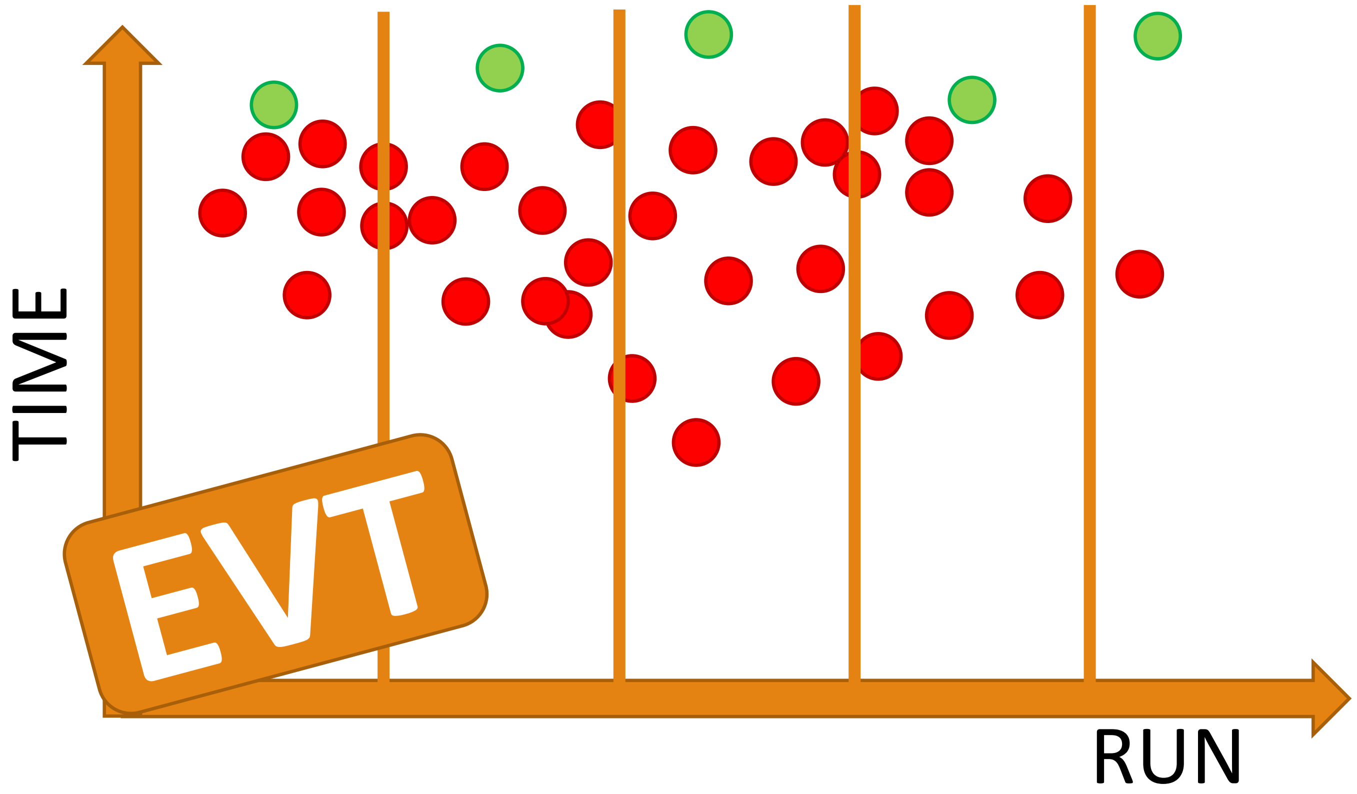
- Static pWCET



```
void main (X) {
    A = funcA(X) ;
    eval (A > 10);
    B = B - A;
}
```

Probabilistic WCET techniques

- Measurement-based



Pros & cons

- ? Allow to derive estimates with confidence level
- ⛔ Require specific conditions of independence
- ⛔ Slowly reaching maturity

Challenges

- Existing tools and methodologies against these new settings and requirements

Static analysis is out the window

Not because of the complexity of the architecture!
Because of the architecture, the programming model, the OS and runtime, the man-power, the rapid evolution of the hardware…

Challenges

⛔ Measurements taken in isolation are not safe as the execution time is subject to variation due to the shared resources

⛔ Measurements taken in a totally congested system are not meaningful

$\Rightarrow$ **Design process** that creates interference on different mappings

$\Rightarrow$ **Investigate** different configurations of the software mapping on hardware

Methodology

- Exploring measurement-based approaches
  - Allowing to analyze and reason about an application timing behavior
- Knowing the parallel graph of execution
  - Deriving timing information for each node
- Collecting execution traces is a tedious process
  - Involves many steps, in different languages
- Developed measurement-based analysis tool
  - Collecting runtime execution traces is (almost) automatic

Open research problems

- From the traces obtained, we can further analyze how sensitive to concurrent activity the analyzed tasks are
  - Define safety margin accordingly
  - Make recommendations to set up the environment in an appropriate way: dynamic vs. static mapping, PREM, ...
  - Restrict the runtime, capture the maximum activity and map it to a pre-defined level of interference intensity
  - Etc…
- In this course we will focus on the methodology and tools to trace and configure
  - Limited analysis

Summary

Summary

- Real-time systems require programming paradigms that can describe and manage concurrency and time properties, as first-class entities
  - The addition of parallel programming introduces a new dimension of challenges and advanced concepts
- There are still open issues to address to enable predictable parallel programming
  - Particularly the specification of real-time properties and the control of execution in the vertical stack
- Timing analysis of parallel computation is also a challenge
  - We will be addressing a hybrid methodology
  - Measurement based approach to derive execution times
  - Using the knowledge of the task parallel graph to determine the end-to-end execution time
    - And for schedulability analysis (not addressed in the course)