Component-Based Software Design

Giorgio Buttazzo
g.buttazzo@sssup.it

Real-Time Systems Laboratory

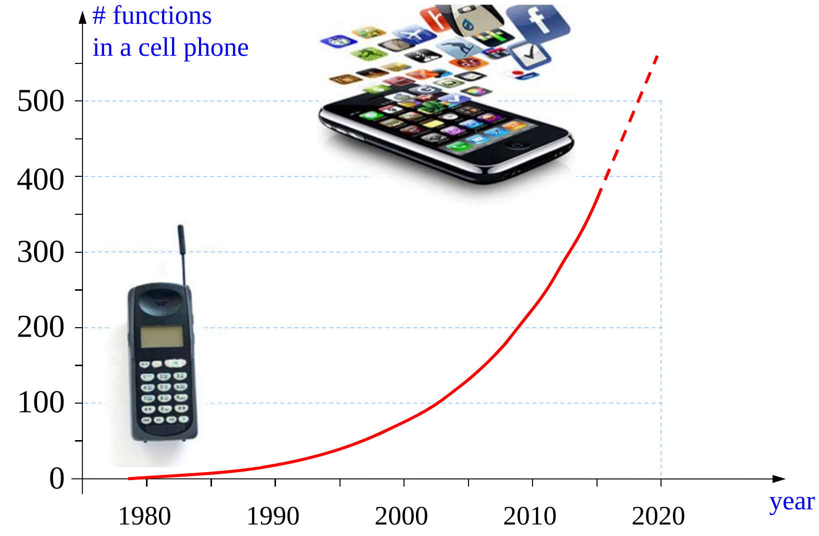
Scuola Superiore Sant'Anna

Context

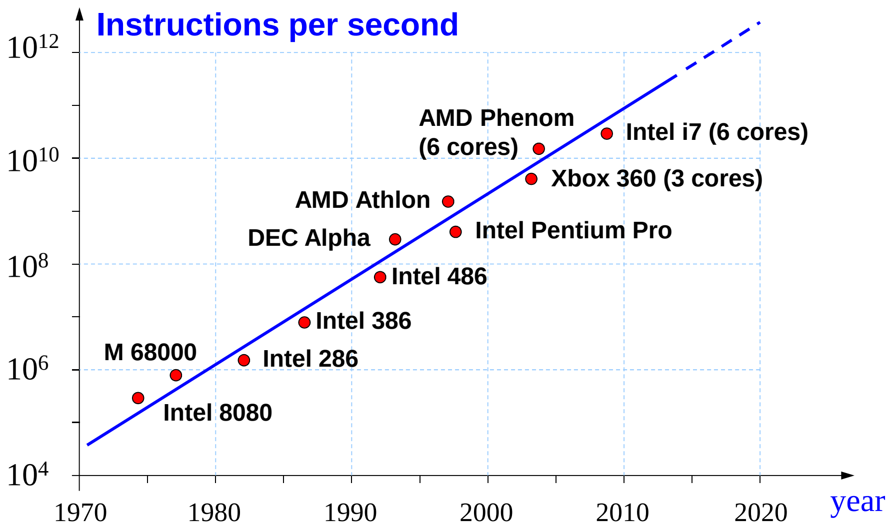
Embedded systems are becoming more complex every day:

- more functions
- higher performance
- higher efficiency
- new hardware platforms

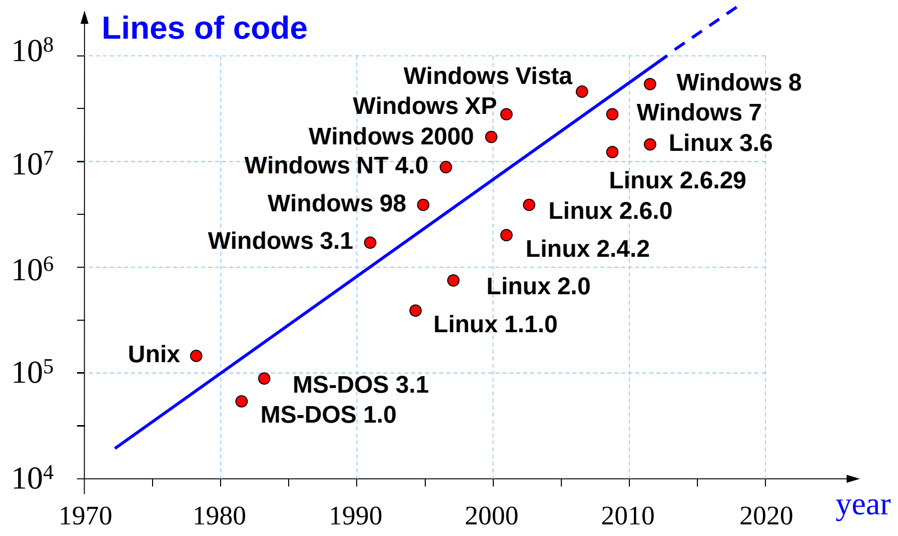
Increasing complexity



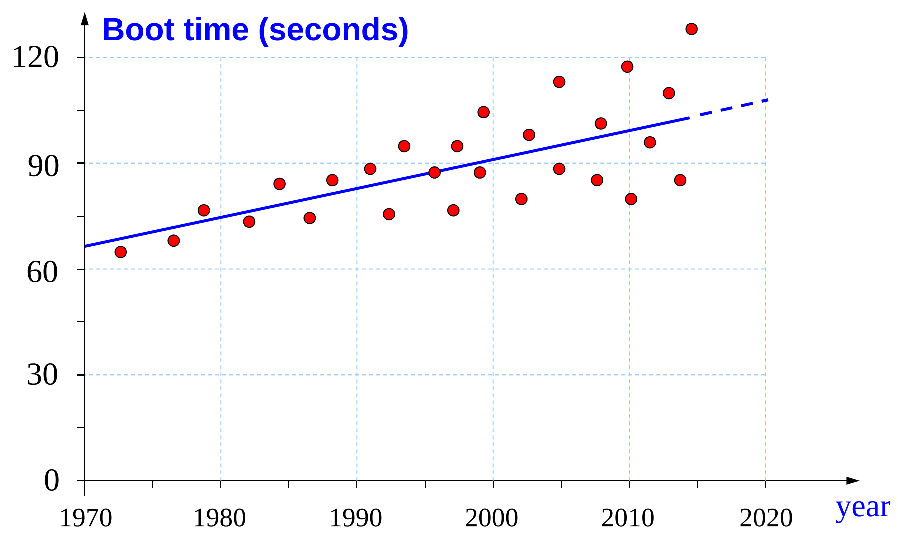
Hardware Performance



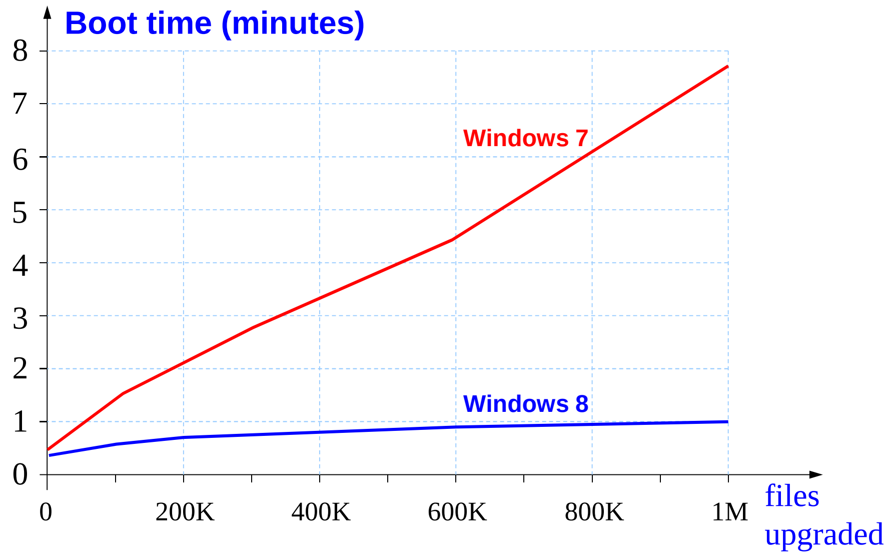
Software Complexity



And the Result is …

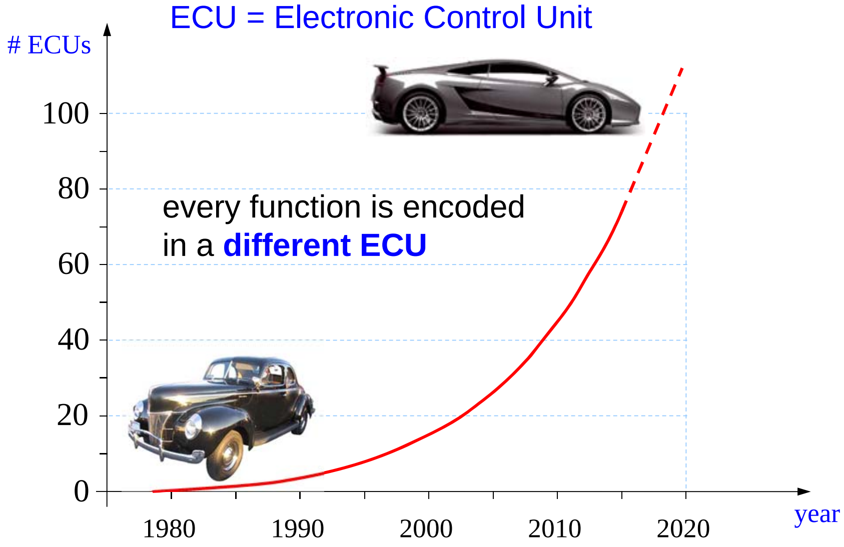


It increases with upgrades



ECU growth in a car

ECU = Electronic Control Unit



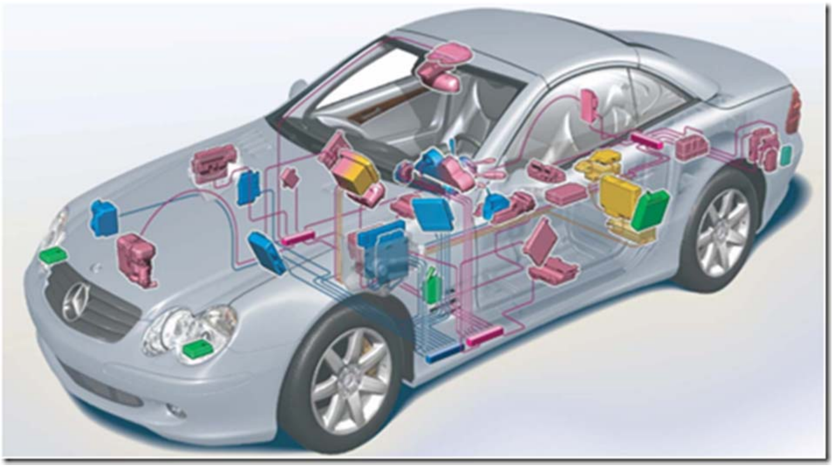
Advantages of separation

Separating functions in dedicated ECUs allows:

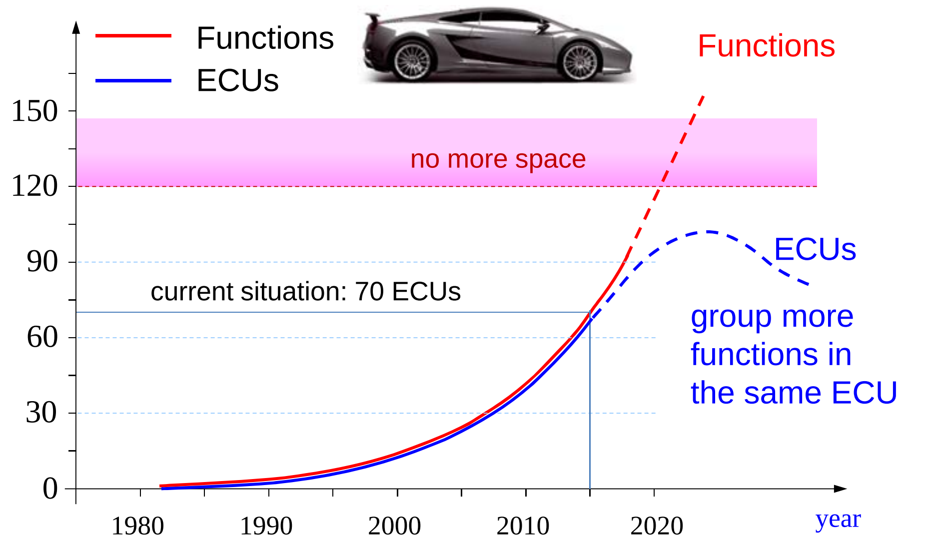
- easier development
- easier testing
- easier certification
- easier maintenance

Problems of separation

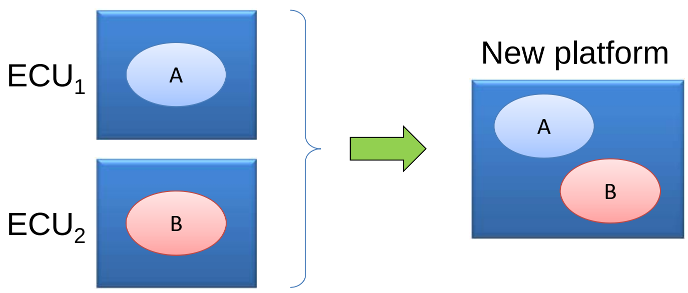
With the increasing number of ECUs, there are problems of space, weight, energy.



How to add more functions?

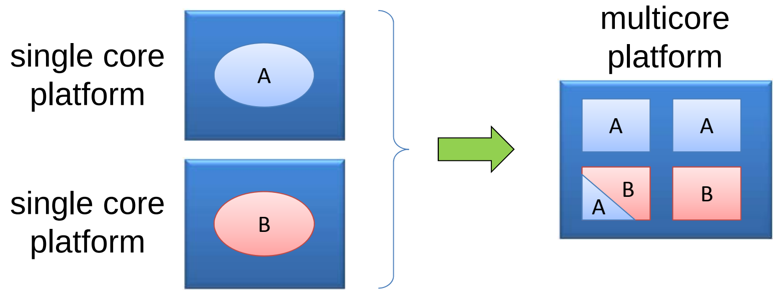


That's nice, but



- How can we **test** and **certify** a function in the presence of other applications?
- How can we **guarantee** behavior and performance to get certification?

Additional problems



- How do we **partition** the applications on the available cores?
- How does the Worst-Case Execution Time (**WCET**) scale on multicore architecture?

The problem

When multiple applications run on the same platform, they interfere with each other due to the use of shared resources.

Interference:	phenomenon for which the execution of a task affects the one of other tasks.

In the following, we will

- identify the causes of interference
- present possible solutions

Interference mechanisms

Tasks may interfere for different reasons:

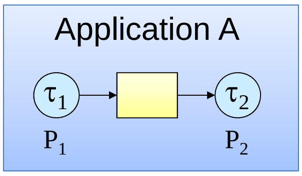
- **Time:** concurrent access to shared resources, as processing units and communication channels.
- **Space:** due to sharing the same memory space (Cache, DRAM, Hard Disk).
- **Energy:** sharing the energy source (battery).
- **Temperature:** eating up each other.

Why do we care?

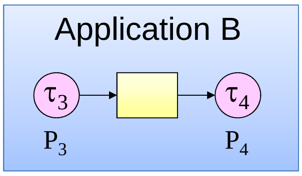
Because interference has different negative effects:

- It decreases efficiency and schedulability
- It reduces predictability
- It jeopardizes safety
- It complicates the analysis

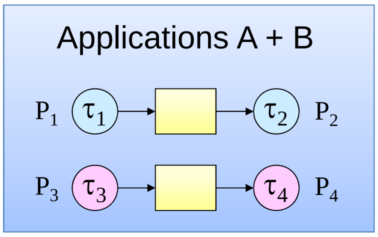
A simple example



CPU 1: speed = 1



CPU 2: speed = 1



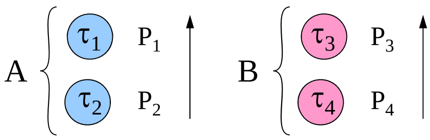
Platform: speed = 2

- Priorities must be assigned
- Task interference can jeopardize predictability

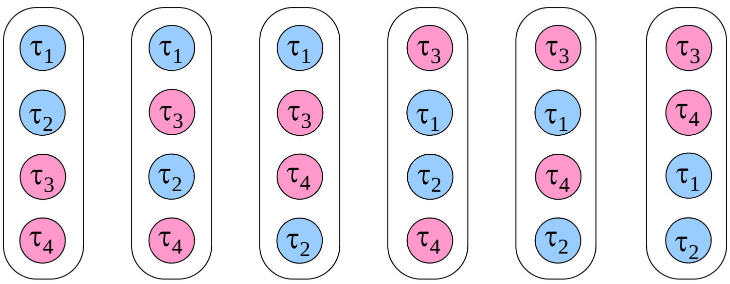
17

Priority explosion!

How many priority assignments satisfy both priority orders?



There are 6 priority assignments that satisfy both priority orders:

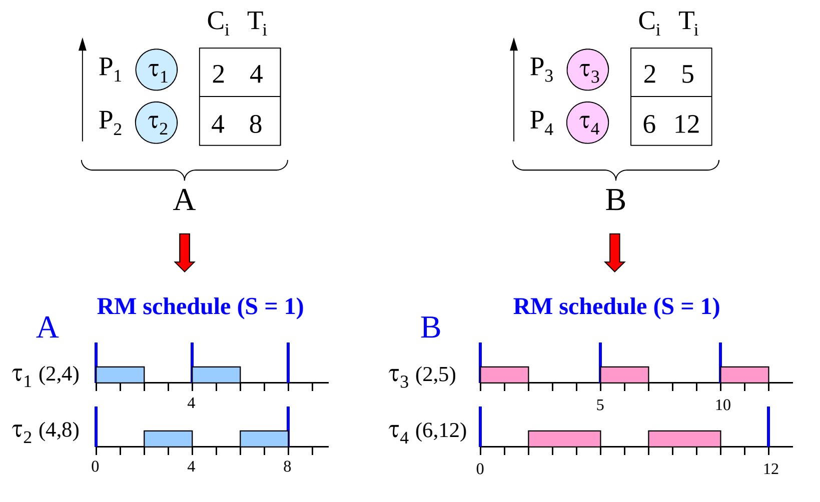


18

Non trivial questions

- How do computation times scale in the new platform?
- Which priority order do we choose?
- Do they all lead to a feasible schedule?
- Are they different in terms of performance?
- How can we reduce the reciprocal interference?

Let's go into details



Now let's groups them

How computation times scale in the new platform?

A (speed = 1)

	C_i	T_i
τ_1	2	4
τ_2	4	8

B (speed = 1)

	C_i	T_i
τ_3	2	5
τ_4	6	12

Assume for simplicity

$$C_i' = \frac{C_i}{s}$$

speed = 2

		C_i	T_i
P_1	τ_1	1	4
P_2	τ_2	2	8
P_3	τ_3	1	5
P_4	τ_4	3	12

A+B

Now let's groups them

If the new platform has a fixed priority scheduler, what is the best priority order?

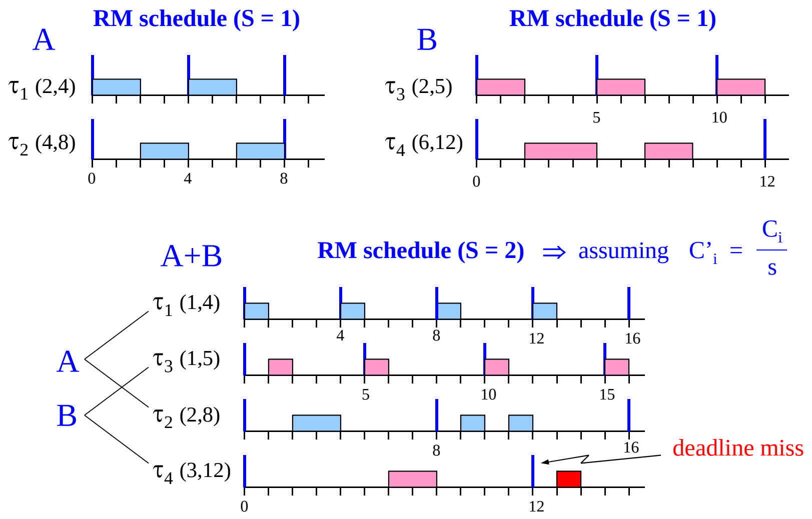
		C_i	T_i
P_1	τ_1	1	4
P_2	τ_2	2	8
P_3	τ_3	1	5
P_4	τ_4	3	12

A+B

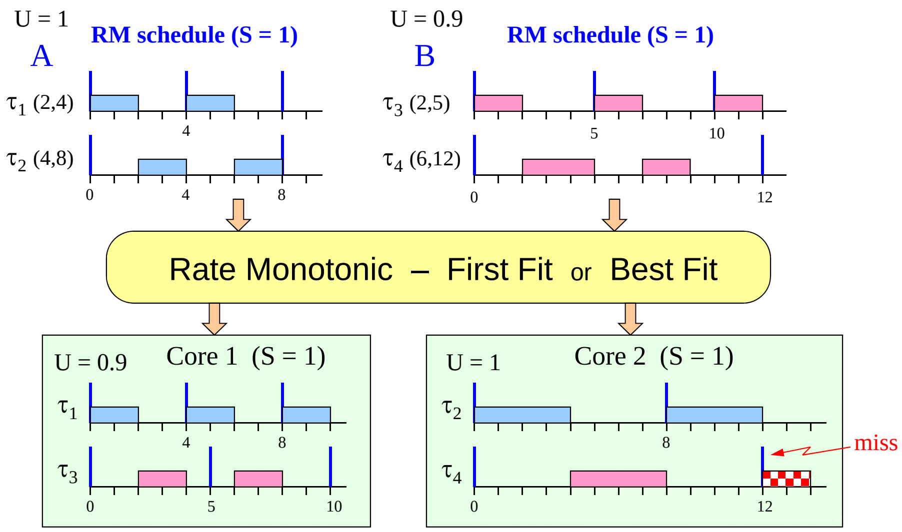
τ_1 4	τ_1 4	τ_1 4	τ_3 5	τ_3 5	τ_3 5
τ_2 8	τ_3 5	τ_3 5	τ_1 4	τ_1 4	τ_4 12
τ_3 5	τ_2 8	τ_4 12	τ_2 8	τ_4 12	τ_1 4
τ_4 12	τ_4 12	τ_2 8	τ_4 12	τ_2 8	τ_2 8

RM ordering (optimal)

All together are not feasible!



Example on 2 cores



Course outline - 1

1. Motivation and examples
2. Brief summary of uniprocessor analysis
3. Interference analysis and techniques to reduce it
   - Temporal isolation
   - Resource reservations servers
   - Hierarchical component-based systems
   - Schedulability analysis of single components
   - Resource sharing protocols for hierarchical systems
4. Energy-aware scheduling

Course outline - 2

5. Multiprocessor scheduling
   - Architecture issues and modeling
   - Performance analysis
   - Scheduling paradigms
   - Task allocation and feasibility bounds
6. Processor abstraction and interface
   - Efficient algorithms for the interface design.
   - Multiprocessor abstractions.
   - Applications models.
   - Application partitioning and resource allocation

Course outline - 3

7. Standards for component-based development
   - ARINC: a standard for avionic systems.
   - AUTOSAR a standard for automotive systems
8. Component-oriented programming and models
   - introduction to C++ patterns
   - UML models of components
   - code generation using patterns under Eclipse-EMF
9. Hypervisors
   - The Xen project
   - Guaranteeing real-time constraints on hypervisor-based systems