Introduction to Real-Time Systems

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br
http://www.lisha.ufsc.br/Giovani

Objectives

- Introduction to Real-Time Systems
  - Motivation for studying
  - Formal definition
  - Soft x hard real-time systems
  - Predictability

What you need to know to follow

- Experience with operating systems
  - Scheduling
  - Memory management
  - Resource management

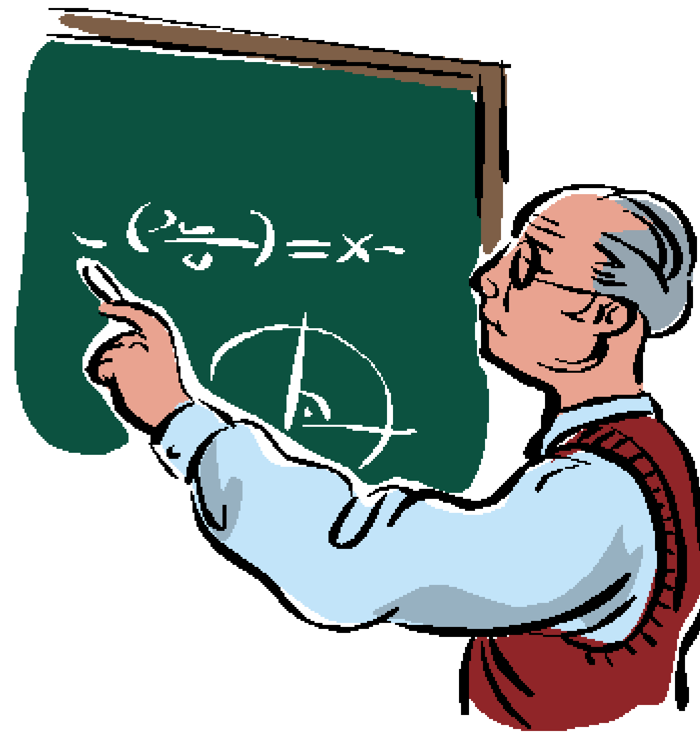
Books

- J. Liu: Real-Time Systems. Prentice Hall, 2000.
  - Best as an intro book, less good as a reference
- Farines, Silva, Oliveira. Sistemas de Tempo Real. DAS/UFSC. Julho 2000
  - Good introduction to single-core real-time systems
- G.C. Buttazzo: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and applications. Springer, 2004
  - Good reference on real-time scheduling
- Hermann Kopetz. Real-Time Systems: design principles for Distributed Embedded Applications

What you will learn

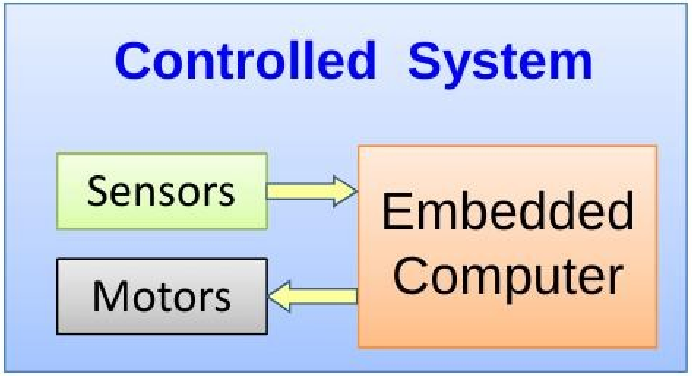
- Definition of real-time systems
- Motivation for studying real-time systems
- Soft x hard real-time systems
- The most important concept of a real-time system
  - Predictability

Let's get started



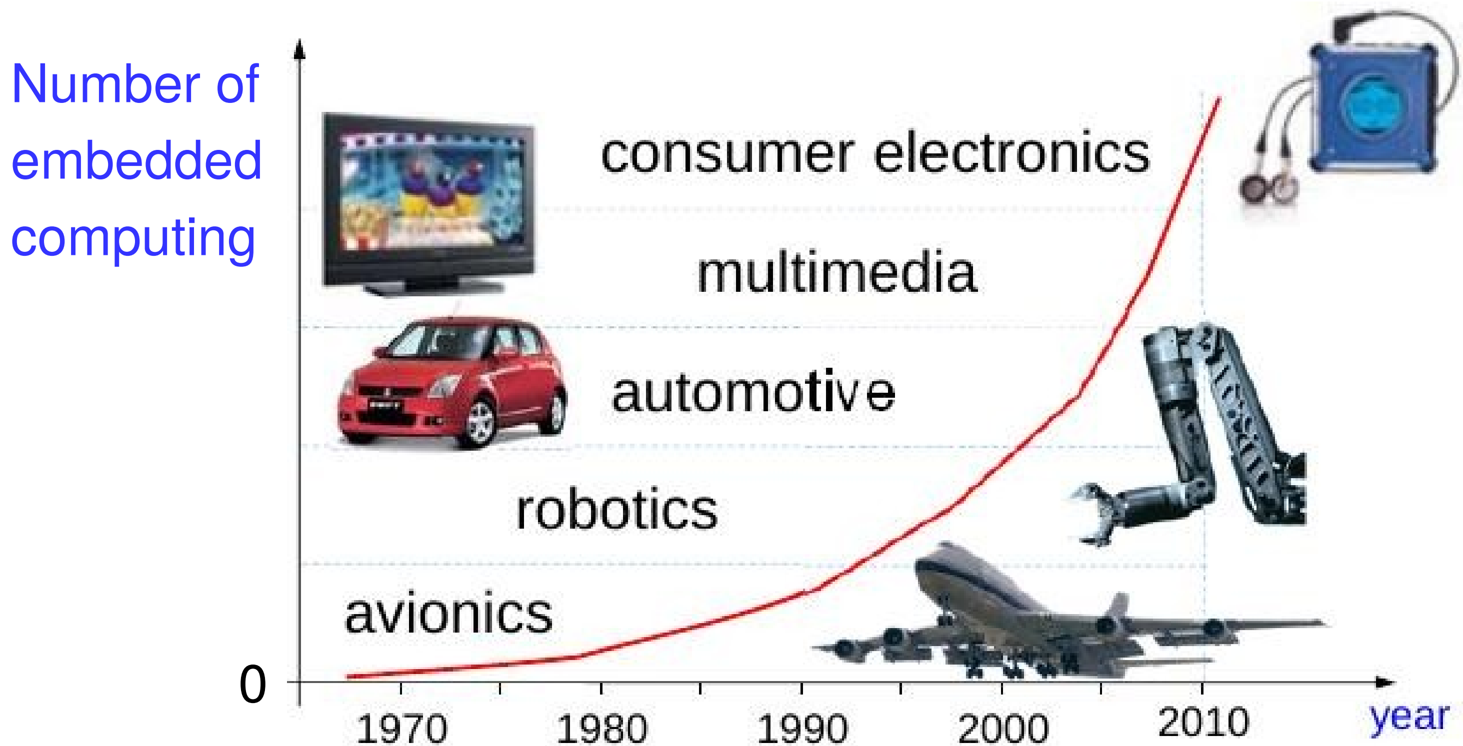
Formal Definition

- The correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced
  - A correct value at the wrong time is a fault
- They are typically embedded in a larger system to control its functions:
  - Real-Time Embedded Systems



Evolution of Embedded Systems

- Embedded computing systems have grown exponentially in several application domains:



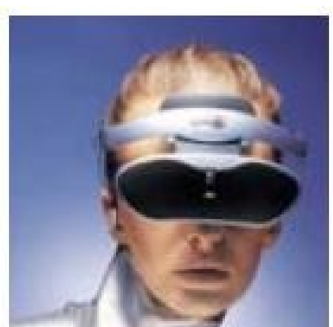
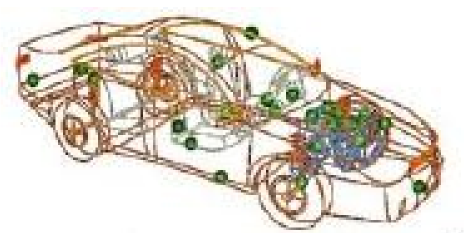
Computers Everywhere

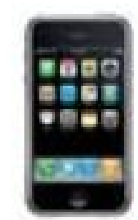
- Today, 98% of all processors in the planet are embedded in other objects

Typical Applications

- avionics
- automotive
- robotics
- industrial automation
- telecommunications
- multimedia systems
- consumer electronics





From Hardware to Software

- We are experiencing a dematerialization process in which many functions are converted into software
- Examples

  Money

  Documents

  Books

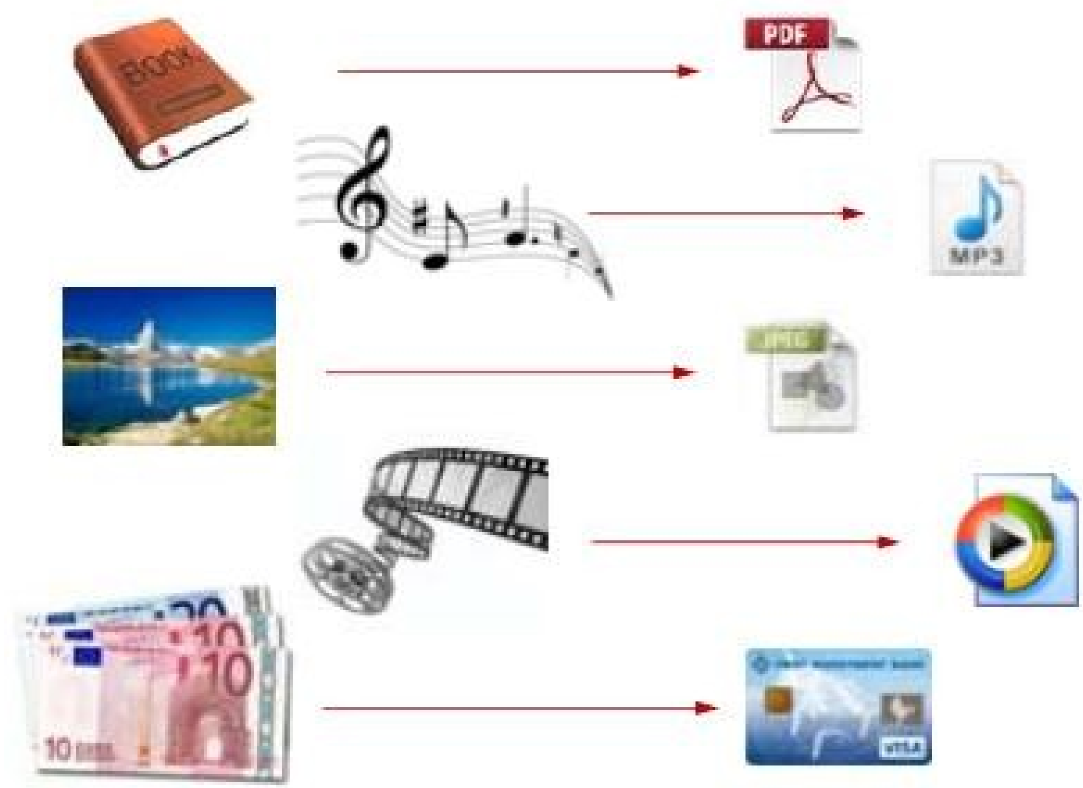
  Music

  Pictures

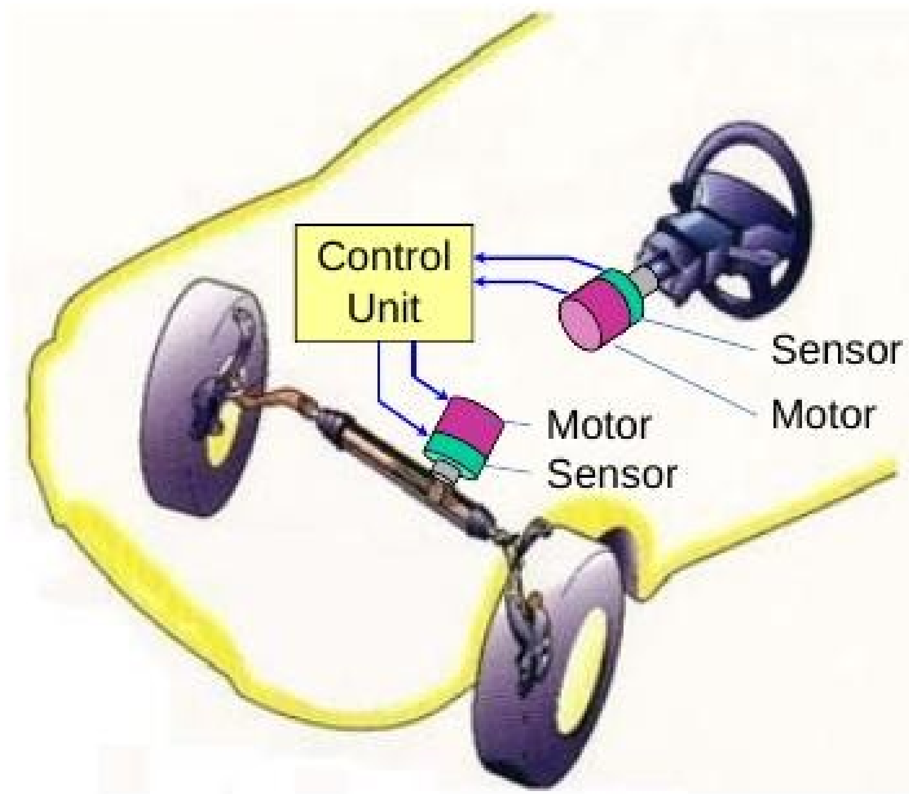
  Movies

  Tickets

  Education



Steer by Wire



Why?

- There are many advantages
  - Software is more flexible than hardware
  - It can be quickly changed/adapted/updated
  - It can be upgraded remotely
  - It can evolve into intelligent control algorithms
  - It has no mass, so it can “travel” at the speed of light

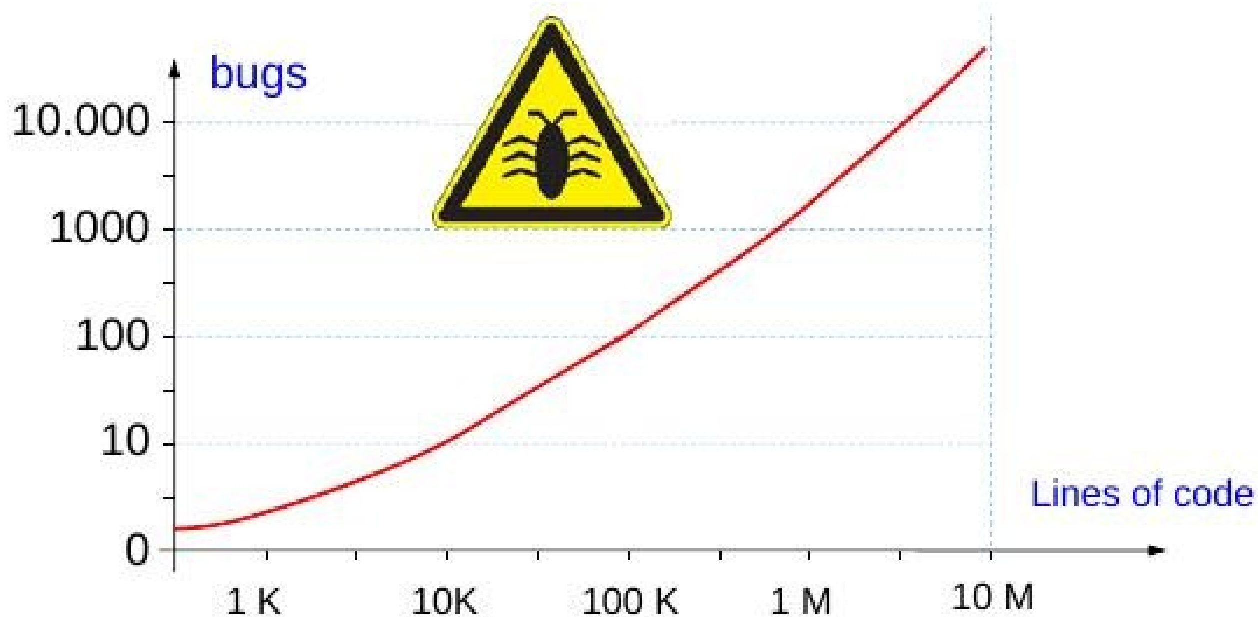
Increasing complexity

- The price to be paid is a higher software complexity
- Related problems
  - Difficult design
  - Less predictability
  - Less reliability
- Novel solutions for
  - Component-based software design
  - Analysis for guaranteeing predictability and safety
  - Testing

Federal University of Santa Catarina
Real-Time Operating Systems

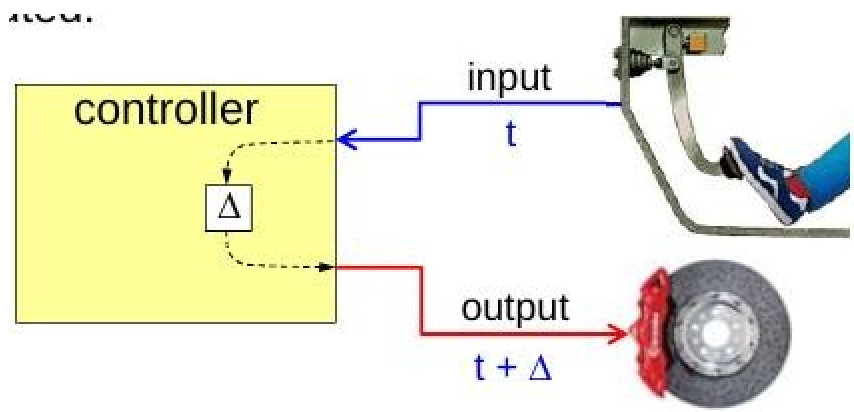
Complexity and bugs

- Software bugs increase with complexity



Software reliability

- Reliability does not only depend on the correctness of single instructions, but also on when they are executed



A correct action executed <u>too late</u> can be useless or even dangerous

Real-Time Systems

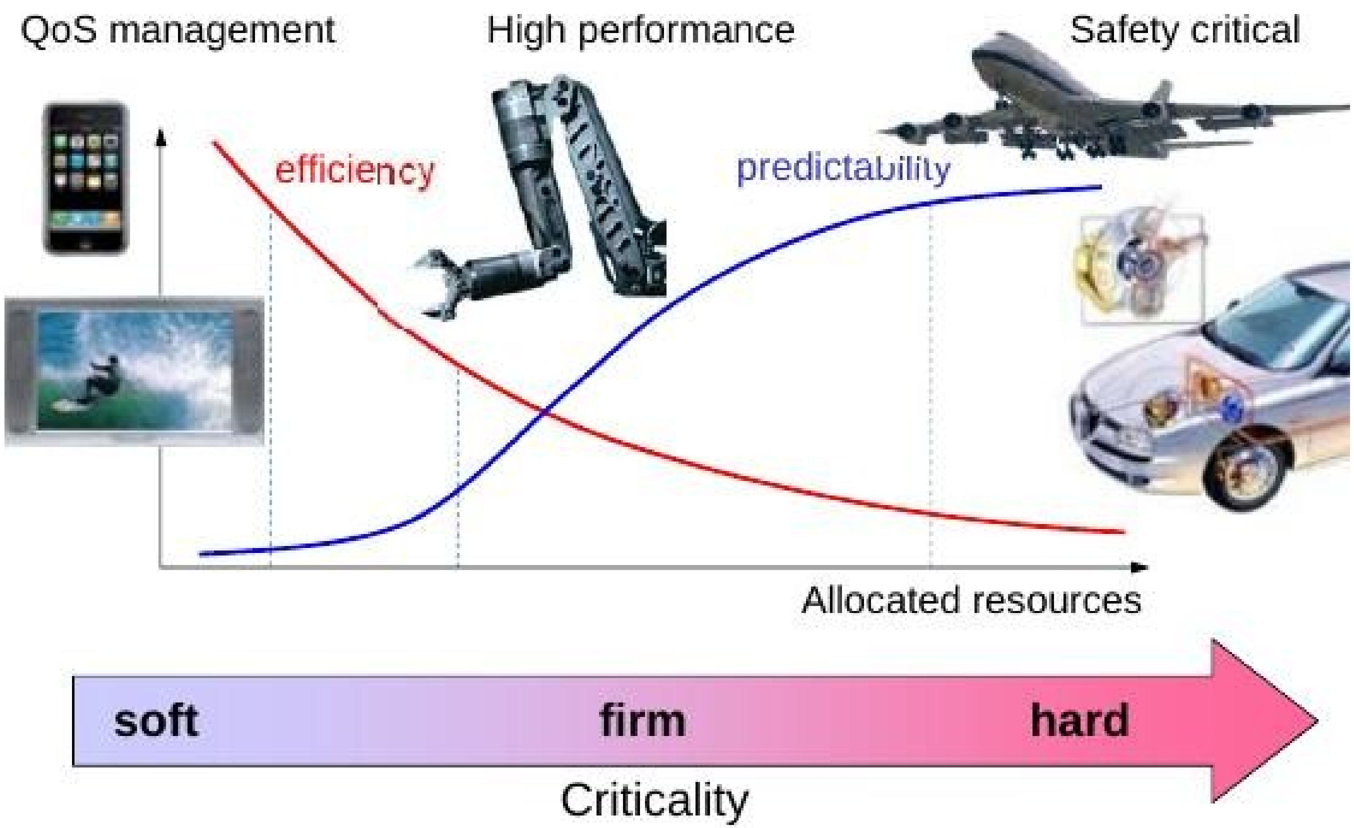
Computing systems that must guarantee bounded and predictable response times are called real-time systems

- Activities are associated with timing constraints (deadlines)
- Predictability of response times must be guaranteed
  - for each critical activity
  - for all possible combination of events

Soft x Hard Real-Time

- Soft Real-Time (SRT): missing deadlines is undesirable, but will not lead to catastrophic consequences
  - Related to the concept of "Quality of Service"
  - Typically interested in average-case response time
  - Ex: reservation systems, media players, phones
- Hard Real-Time (HRT): missing deadlines is not an option
  - Interested in worst-case response times
  - Ex: airplanes, nuclear plants, military systems

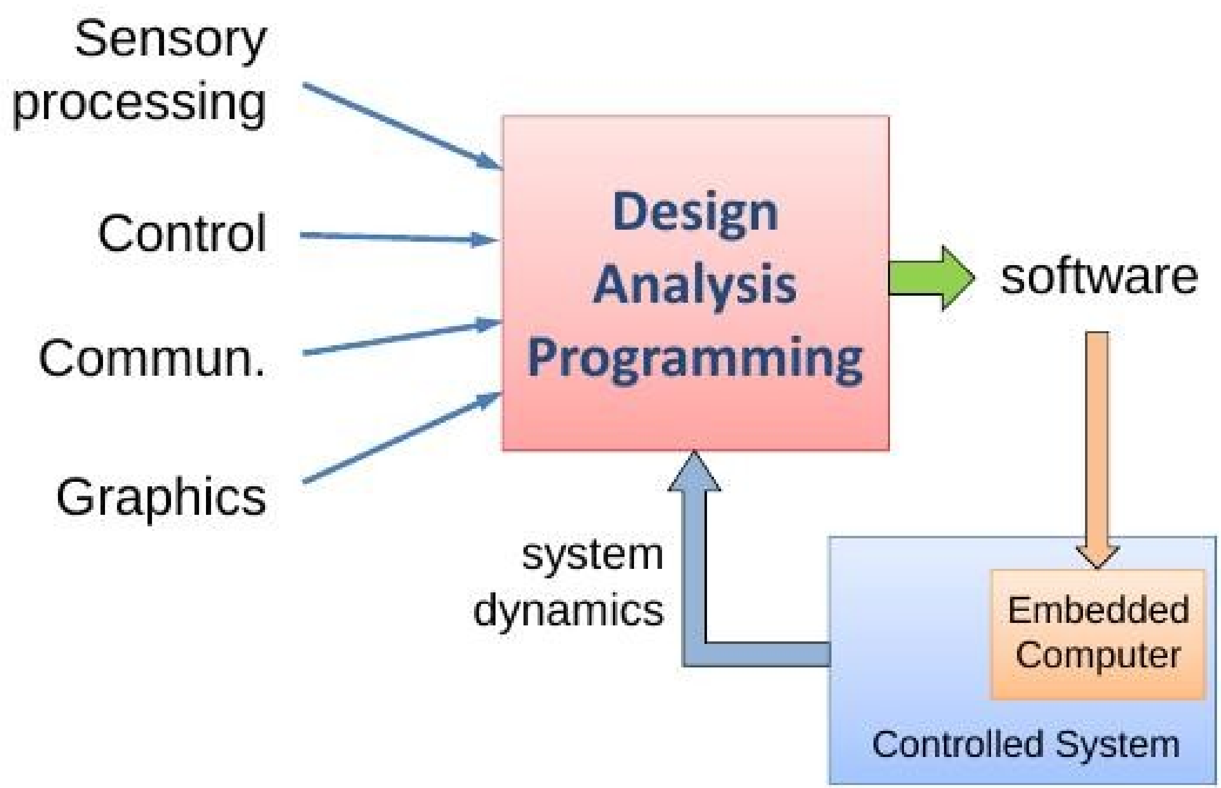
Real-Time Spectrum



What's special in Embedded Systems?

FEATURES		REQUIREMENTS
Scarce resources (space, weight, time, memory, energy)	→	**High efficiency** in resource management
High concurrency and resource sharing (high task interference)	→	**Temporal isolation** to limit the interference
Interaction with the environment (causing timing constraints)	→	**High predictability** in the response time
High variability on workload and resource demand	→	**Adaptivity** to handle overload situations

Our focus: predictable software



Control and implementation

- Often, control and implementation are done by different people that do not talk to each other

- Control guys typically assume a computer with infinite resources and computational power. In some cases, computation is modeled by a fixed delay

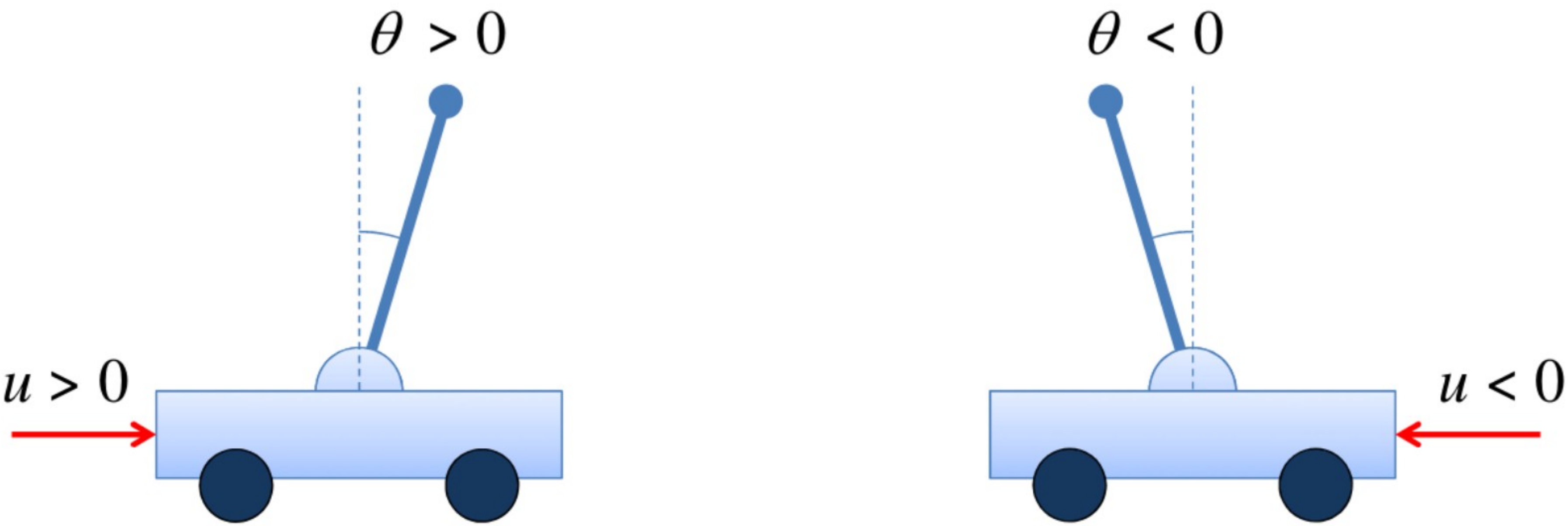
Control and implementation

- In reality, a computer:
  - has limited resources
  - finite computational power (non null execution times)
  - executes several concurrent activities
  - introduces variable delays (often unpredictable)

Modeling such factors and taking them into account in the design phase allows a significant improvement in performance and realibility

A control example

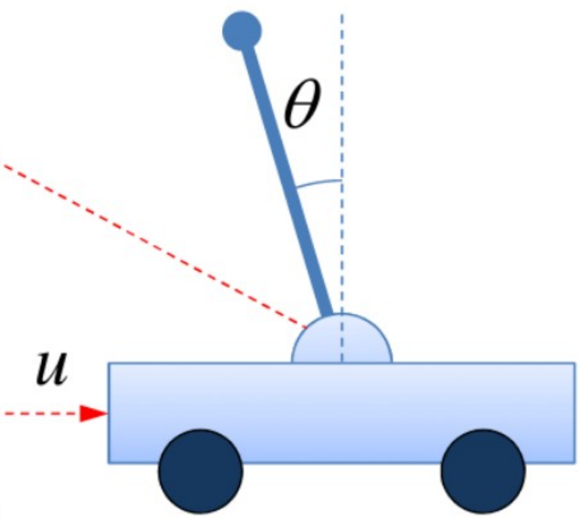
A positive angle θ requires a positive control action μ.



$\theta > 0 \Rightarrow u > 0$ $\theta < 0 \Rightarrow u < 0$

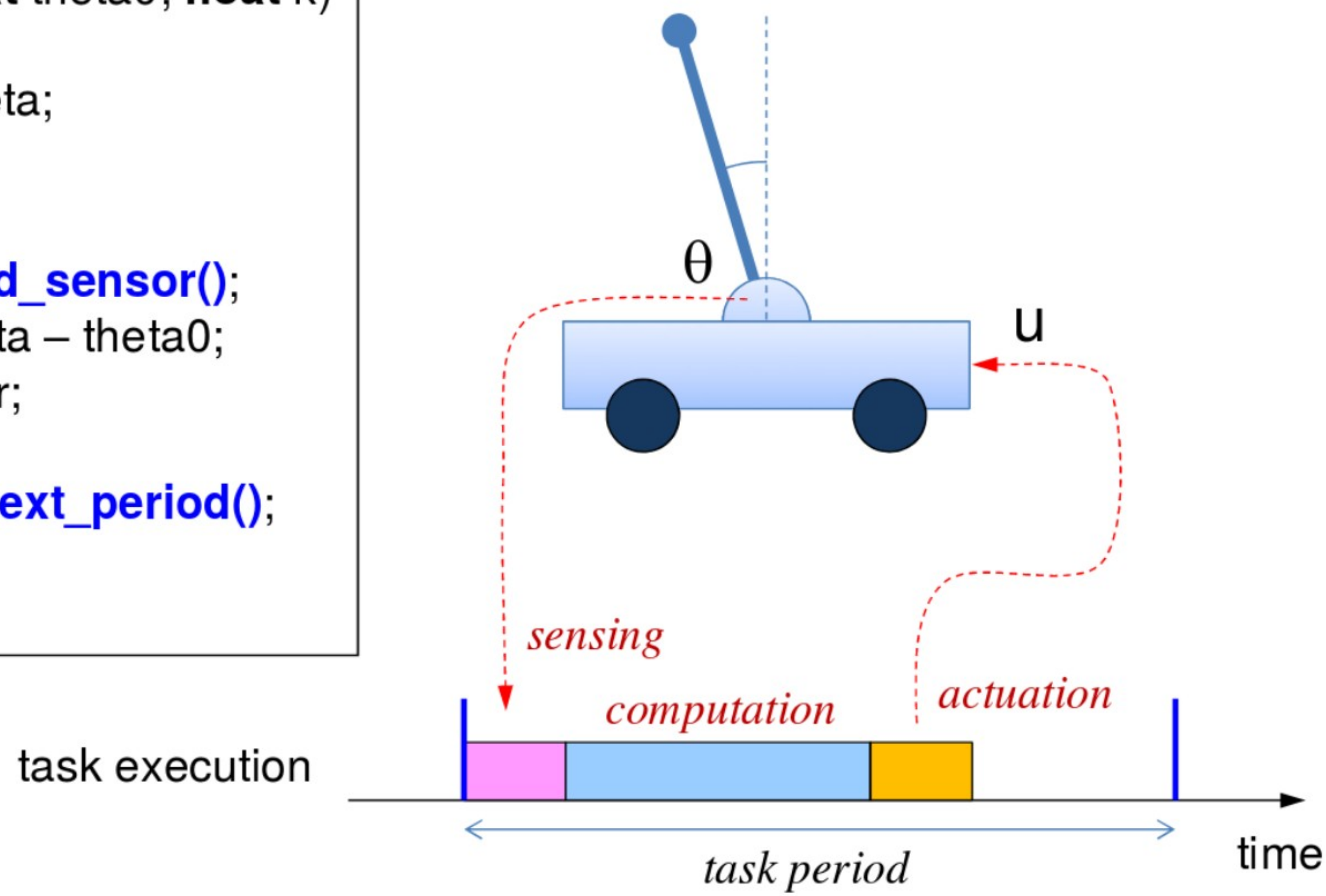
A control task

```
task    control(float theta0, float k)                 control gain
{                                                      reference angle
float   error;
float   u;
float   theta;

    while (1) {
                                                       sensing
        theta = read_sensor();

        error  = theta – theta0;   }                   computation
        u = k * error;             }
                                                       actuation
        output(u);

        wait_for_next_period();                        synchronization
    }
}
```



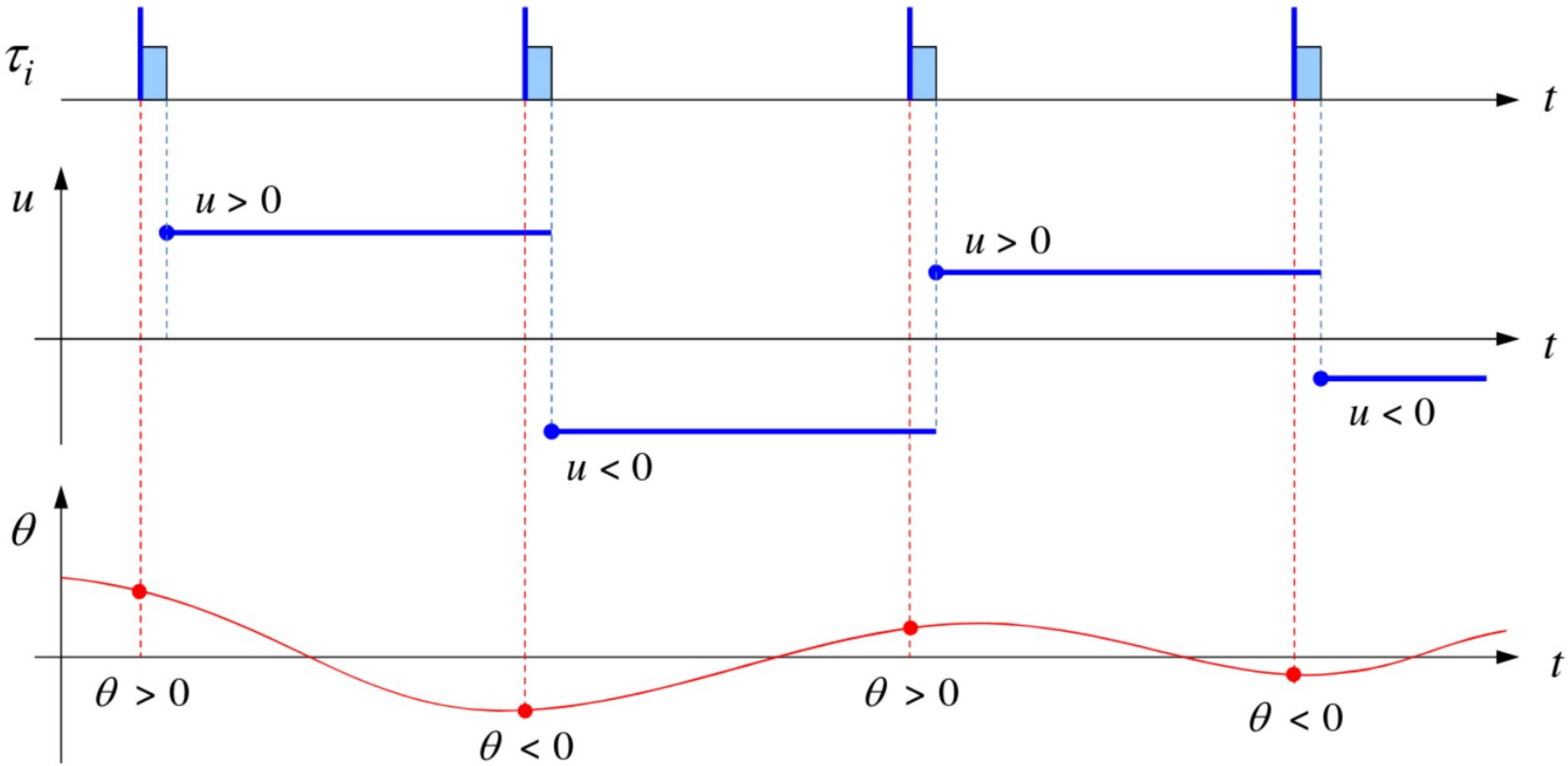
A control task

```
task    control(float theta0, float k)
{
float   error, u, theta;

    while (1) {
        theta = read_sensor();
        error  = theta – theta0;
        u = k * error;
        output(u);
        wait_for_next_period();
    }
}
```



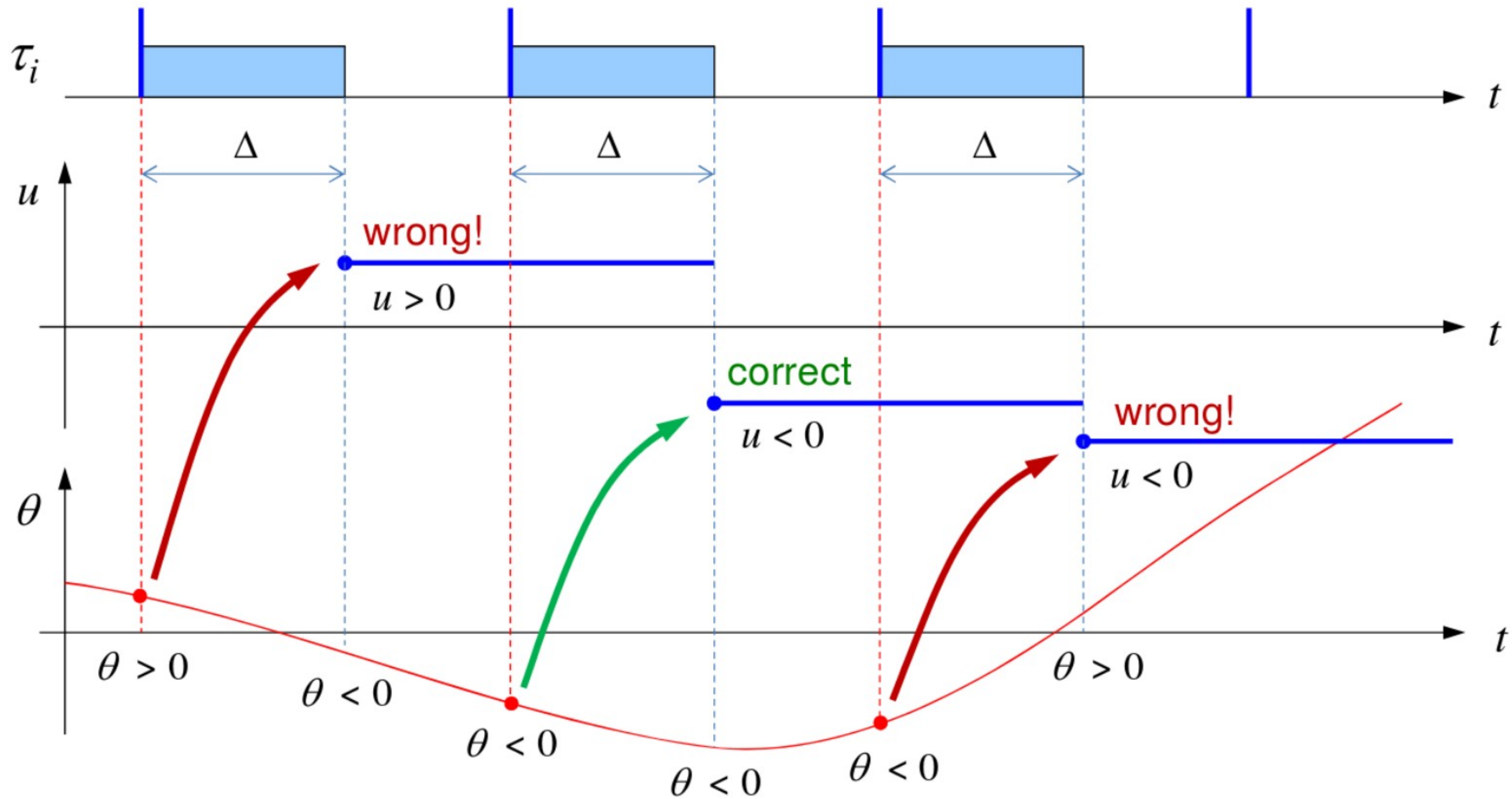
Tradition control view

Negligible delay and jitter



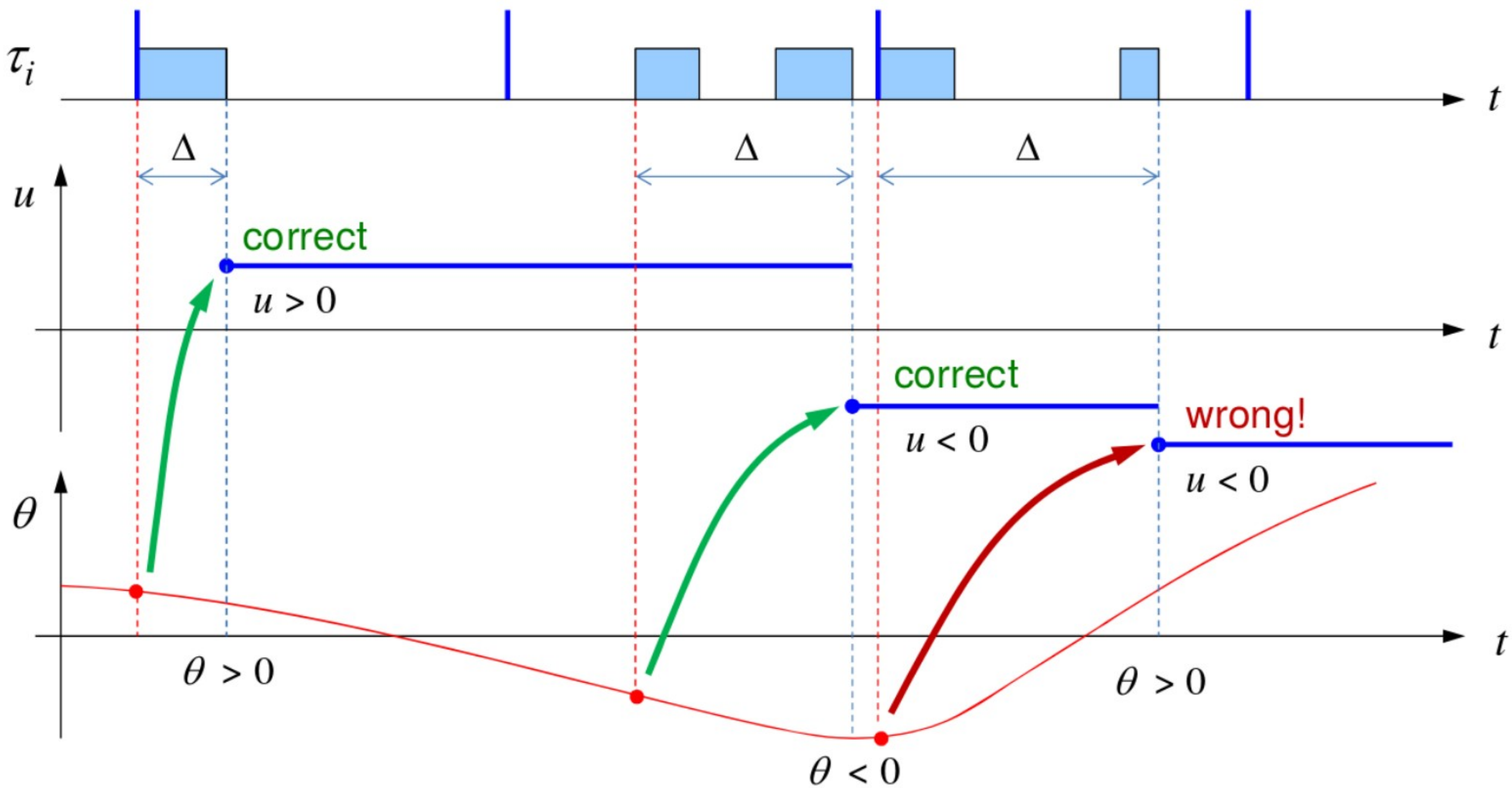
Effect of computation times

Computation times introduce a non negligible delay:



Actual Situation

Actual situation: variable delay and jitter:



RTOS responsabilities

- A real-time operating system (RTOS) is responsible for
  - managing concurrency
  - activating periodic tasks at the beginning of each period (time management)
  - deciding the execution order of tasks (scheduling)
  - solving possible timing conflicts during the access of shared resources (mutual exclusion)
  - manage the timely execution of asynchronous events (interrupt handling)

Typical objection

It is not worth to invest in RT theory, because computer speed is increasing exponentially, and all timing constraints can eventually be handled

- Answer
  - Given an arbitrary computer speed, we must always guarantee that timing constraints can be met. Testing is NOT sufficient.

Real-Time != Fast

- A real-time system is not a fast system
- Speed is always relative to a specific environment
- Running fast is good, but does not guarantee a correct behavior

Speed vs. Predictability

- The objective of a real-time system is to guarantee the timing behavior of each individual task
- The objective of a fast system is to minimize the average response time of a task set. However…

Do not trust the average when you have to guarantee individual performance

Sources of non determinism

- Architecture
  - cache, pipelining, interrupts, DMA
- Operating System
  - scheduling, synchronization, communication
- Language
  - lack of explicit support for time
- Design methodologies
  - lack of analysis and verification techniques

Traditional (wrong) approach

- Most RT application are designed using empirical techniques
  - assembly programming
  - timing through dedicated timers
  - control through driver programming
  - priority manipulation

Disadvantages

1) Tedious programming which heavily depends on programmer's ability
2) Difficult code understanding
3) Difficult software maintainability
   a) millions lines of code
   b) code understanding takes more time than re-writing
   c) re-writing is very expensive and bug prone
4) Difficult to verify timing constraints without explicit support from OS and the language

Implications

- Such a way of programming RT applications is very dangerous
- It may work in most situations, but the risk of a failure is high
- When the system fails is very difficult to understand why
- Conclusion: low reliability

Accidents due to SW

- Task overrun during LEM lunar landing
  (http://njnnetwork.com/2009/07/1202-computer-error-almost-aborted-lunar-landing/)
- First flight of the Space Shuttle (synch)
- Ariane 5 (overflow)
- Airbus 320 (cart task)
- Airbus 320 (holding task)
- Pathfinder (reset for timeout, priority inversion)

Lessons learned

- Tests, although necessary, allow only a partial verification of system's behavior
- Predictability must be improved at the level of the operating system
- The system must be designed to be fault-tolerant and handle overload conditions
- Critical systems must be designed under pessimistic assumptions

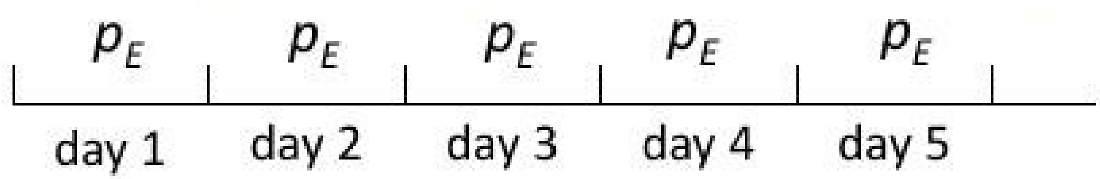
and always remember of the Murphy's Law

- If something can go wrong, it will go wrong
- If a system stops working, it will do it at the worst possible time
- Sooner or later, the worst possible combination of circumstances will occur

Proving Murphy's law

Let p_E be the probability for event E to occur in a day

What is the probability for E to occur in *n* days?

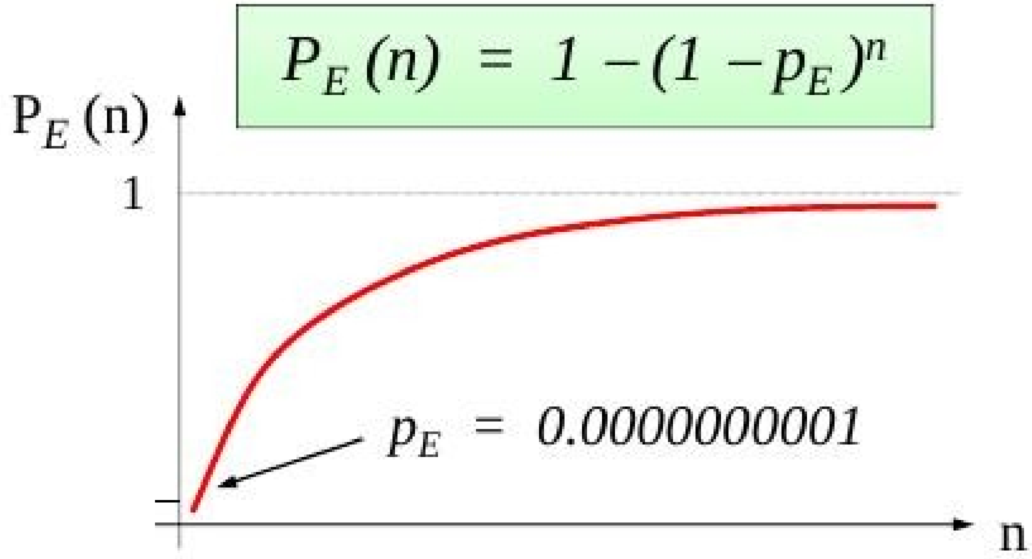


prob. of E not occurring in 1 day $\qquad q_E = 1 - p_E$

prob. of E not occurring in n days $\qquad Q_E(n) = (1 - p_E)^n$

prob. of E occurring in n day $\qquad P_E(n) = 1 - Q_E(n)$

Understanding Murphy's law



If something can go wrong (no matter how small Pe is), it will go wrong (that is, the probability for E to occur in long time intervals tends to 1).

Review

- Formal definition of Real-Time Systems
- Soft x Hard real-time systems
- Predictability

References

- Giorgio Buttazo. Real-time systems course
- Farines, Silva, Oliveira. Sistemas de Tempo Real. DAS/UFSC. Julho 2000
- G.C. Buttazzo: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and applications. Springer, 2004