

Real-Time Systems

Academic Year 2019/20
Master Degree in Computer Science
Department of Mathematics
University of Padua
Tullio Vardanega

Lecture topics

- | | |
|-----------------------------------|--|
| 1. Introduction | 5. Distributed systems |
| 2. Scheduling basics | 6. Timing analysis |
| 3. Fixed-priority scheduling | 7. Multicore systems |
| a. Task interactions and blocking | 8. Predictable parallel programming |
| b. Exercises and extensions | |
| 4. Implementation issues | Bibliography |
| a. Programming real-time systems | • J. Liu, "Real-Time Systems", Prentice Hall, 2000 |
| b. Under the hood | • A. Burns and A. Wellings, "Analysable Real-Time Systems - Programmed in Ada", Amazon Books, 2016 |
| | • State-of-the-art literature |

Welcome words

- **Learning outcomes**
 - Realize the existence and the needs of software systems whose response time is critical to their use
 - Understand the principles, methods, techniques and technology required to develop them to guaranteed predictability
- **Instructional methods**
 - Audio complement to slide desks, uploaded in sync
 - The latter are in the public domain, the former in UNIPD's Moodle space
 - Video-conferences to elaborate on specifics
 - On request, and on regular schedule
 - Reciprocal feedback
 - Instructor to student, about incremental (self-)assignments
 - Student to instructor, about the progression of learning
 - Interaction
 - Posts in Moodle's billboard for this class, for all that must be public
 - Email otherwise

1. Introduction

Where we make some initial acquaintance with what real-time systems are, and why they came about, and then take a first look at their abstract concept

Initial intuition /1

■ Real-time system /1

- An aggregate of computers, I/O devices and *application-specific software*, characterized by
 - Intensive interaction with external environment
 - Time-dependent variations in the state of the external environment
 - Need to keep control over all individual parts of the external environment and to react to changes
- System activities subject to timing constraints
 - Reactivity, accuracy, duration, completion, responsiveness: all dimensions of *timeliness*
- System activities inherently *concurrent* and increasingly *parallel*
- The satisfaction of all system constraints must be proved

2019/2020 UniPD - T. Vardanega

Real-Time Systems

5 of 537

Concurrency vs. parallelism

- Concurrent programming allows using multiple logical threads of control to reflect *cohesively* the collaborative structure of the solution
 - I do “my” bit, you do “yours”; our respective waiting is not wasteful
 - Threads form the architecture: they are long-lived
- Parallel programming promotes a divide-and-conquer logic to solve a problem, with multiple threads that work *independently* on the problem space
 - I work as fast as I can and know nothing about you
 - Threads are mindful of throughput: they are short-lived

2019/2020 UniPD - T. Vardanega

Real-Time Systems

6 of 537

Initial intuition /2

■ Real-time system /2

- Operational correctness does not solely depend on the logical result but also on the time at which the result is produced
 - The computed response has an application-specific utility
 - Correctness is defined in the value domain and in the time domain
 - A logically-correct response produced later than due may be bad

■ Embedded system

- The computer and its software are fully immersed in an *engineering system* comprised of the external environment subject to its control
 - Defined in terms of physical attributes that can be interacted with

2019/2020 UniPD - T. Vardanega

Real-Time Systems

7 of 537

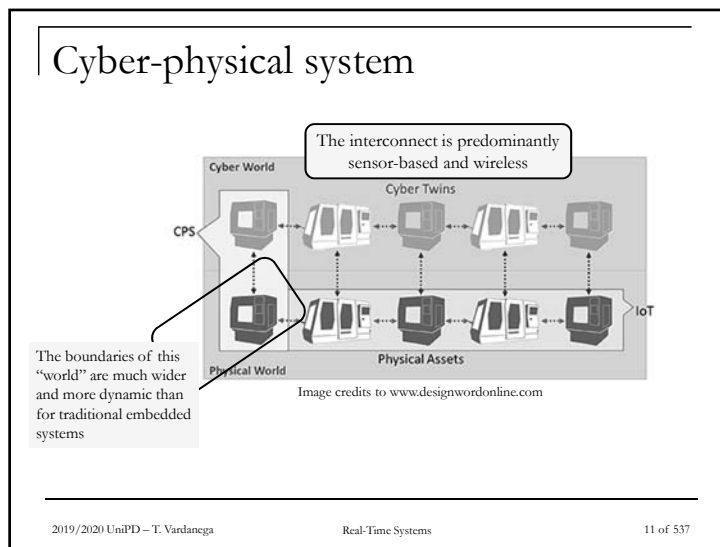
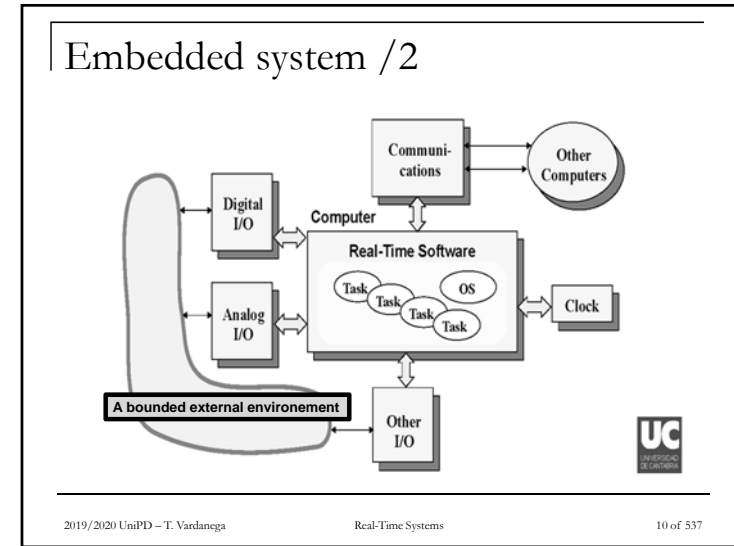
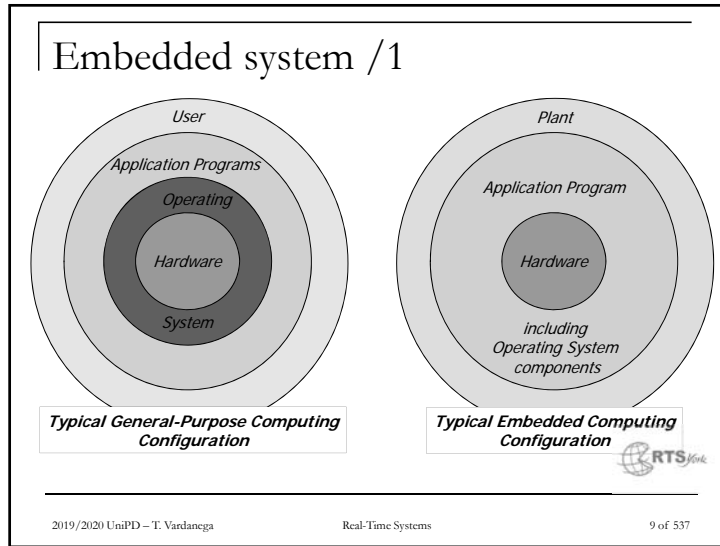
Embedded systems vs. CPS

- **Cyber-physical systems (CPS)** are the new frontier in real-time systems research: how do they differ from embedded systems?
 - Traditional embedded systems are *closed* systems
 - The interaction with the environment is bounded to selected parameters only, and the system operation varies solely within a fixed set of modes toward a given mission
 - Cyber-physical systems are intrinsically *open*
 - The environment forms part of the system, and it is highly dynamic, hence not fully statically known
 - The functional needs of the system may vary rapidly over time

2019/2020 UniPD - T. Vardanega

Real-Time Systems

8 of 537



- ### Cybernetics: now and then
- Born in 1948 as *the science of control systems*, prerequisite to the automation of control
 - From Greek's κυβερνητης (Latin's "gubernator"), steersman
 - Sensing the external (physical) environment
 - Computing the distance from the expected status
 - Actuating devices to effect the system or the environment, so as to reduce that distance
 - Every control action performed on the external environment causes (positive or negative) *feedback*
 - The goal of cybernetics is to calibrate control actions so that the system objective is reached with bounded feedback
- 2019/2020 UniPD - T. Vardanega Real-Time Systems 12 of 537

Automation of control /1

- A digital system comprised of sensors and actuators

$a_k = a_{k-2} + \left[\alpha(r_k - s_k) + \beta(r_{k-1} - s_{k-1}) + \gamma(r_{k-2} - s_{k-2}) \right]$

2019/2020 UniPD - T. Vardanega Real-Time Systems 13 of 537

Automation of control /2

- Factors of influence
 - Quality of response (*responsiveness*)
 - Sensor sampling is typically periodic with period T
 - For the convenience of control theory
 - Actuator commanding is produced at the time of the next sampling
 - As part of feedback control mathematics
 - System stability degrades with the width of the sampling period
 - Plant *capacity*
 - Good-quality control reduces oscillations
 - A system that needs to react rapidly to environmental changes and is capable of it within *rise time* R requires higher frequency of actuation and thus faster sampling \rightarrow hence shorter T
 - A rule-of-thumb R/T ratio normally ranges [10 .. 20]

2019/2020 UniPD - T. Vardanega Real-Time Systems 14 of 537

Automation of control /3

- Complex systems must support multiple distinct periods T_i
 - Easier to set a **harmonic** relation between all T_i
 - This removes the need for concurrency of execution in the relevant computations
 - But it causes coupling between possibly unrelated control actions which is a poor architectural choice
 - There may be diverse components of speed
 - *Forward, side slip, altitude*
 - As well as diverse components of rotation
 - *Roll, pitch, yaw*
 - Each of them requires separate control activities each performed at a specific rate

Any three-dimensional rotation can be described as a sequence of *roll* (x), *pitch* (y), *yaw* (z) rotations (Euler angles)

2019/2020 UniPD - T. Vardanega Real-Time Systems 15 of 537

Automation of control /4

(Artificially) harmonic multi-rate system

- 180 Hz cycle
 - Check all sensor data and select sources to sample
 - Reconfigure system in case of read error
- 90 Hz cycle
 - Perform control law for pitch, roll, yaw (internal loop)
 - Command actuators
 - Perform sanity check
- 30 Hz cycle
 - Perform control law for pitch, roll, yaw (external loop) and integration
- 30 Hz cycle
 - Capture operator keyboard input and choice of operation model
 - Normalize sensor data and transform coordinates; update reference data
- *Can you figure how those activities can progress together?*

2019/2020 UniPD - T. Vardanega Real-Time Systems 16 of 537

Automation of control /5

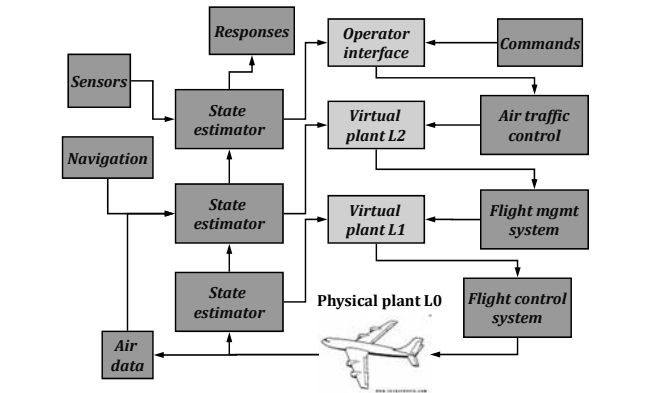
- Command and control systems are often organized in a hierarchical fashion
 - At the lowest level we place the digital control systems that operate on the physical environment
 - At the highest level we place the interface with the human operator
 - The output of higher-level controllers becomes a reference value $r(t)$ for lower-level controllers
 - The more composite the hierarchy the more complex the interdependence in the logic and timing of operation

2019/2020 UniPD - T. Vardanega

Real-Time Systems

17 of 537

Example: hierarchical control system



2019/2020 UniPD - T. Vardanega

Real-Time Systems

18 of 537

Application requirements

- A control system consists of (distributed) resources governed by a *real-time operating system*, RTOS
- The system design overall must meet stringent **reliability** requirements
 - Measured in terms of *maximum acceptable probability of failure*
 - For example: 10^{-9} /hour of flight for the Airbus A-3X0 control system
 - One failure in 10^9 hours of flight (> 114k years!)

2019/2020 UniPD - T. Vardanega

Real-Time Systems

19 of 537

RTS key characteristics /1

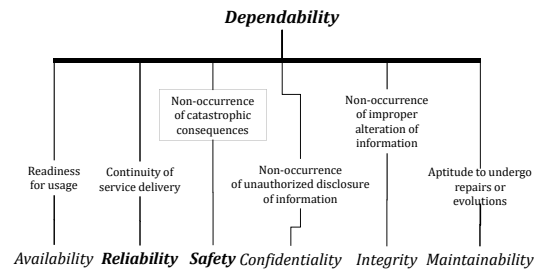
- **Complexity**
 - In algorithms, mostly because of the need to apply discrete control over analog and continuous physical phenomena
 - In development, mostly owing to more demanding verification and validation processes
- **Heterogeneity** of components and of processing activities
 - Multi-disciplinary engineering (spanning control, SW, and system)
- Extreme **variability** in size and scope
 - From tiny and pervasive (nano-devices) to very large (aircraft, plant)
 - In all cases, finite in computational resources
- Proven **dependability**

2019/2020 UniPD - T. Vardanega

Real-Time Systems

20 of 537

Dependability attributes



2019/2020 UniPD - T. Vardanega

Real-Time Systems

21 of 537

RTS key characteristics /2

- Must respond to events triggered by the external environment as well as by the passing of time
 - Double nature: *event-driven* and *time-driven*
- Continuity of operation
 - A real-time embedded system must be capable of operating without (constant) human supervision
 - Nearly no keyboard-based interaction!
- Software architecture inherently concurrent and increasingly parallel
- Must be temporally **predictable**
 - Need for static (off-line) verification of correct temporal behavior
 - How does that relate to **determinism**?

2019/2020 UniPD - T. Vardanega

Real-Time Systems

22 of 537

Predictability vs. determinism

- Predictability (what can be established a priori) may be regarded as a continuum
 - Its highest end is full a-priori knowledge
 - Which allows deterministic reasoning, hence absolute certainty
 - Its lowest end is total absence of a-priori knowledge
 - See what happens ...
- Seeking predictability implies reasoning about kinds and degrees of uncertainty, pursuing an acceptable balance
 - Very rarely we have full a-priori knowledge

2019/2020 UniPD - T. Vardanega

Real-Time Systems

23 of 537

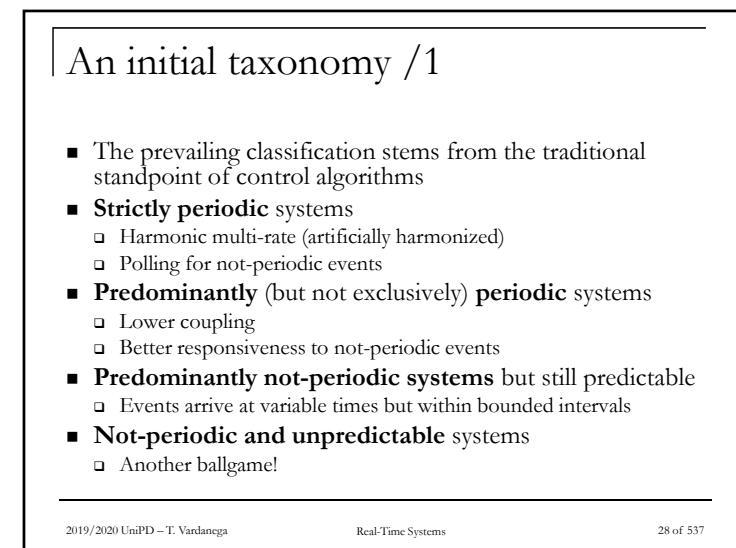
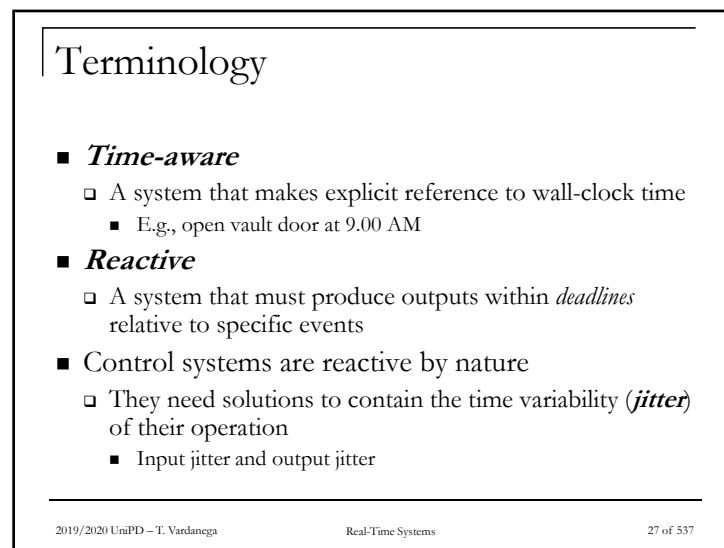
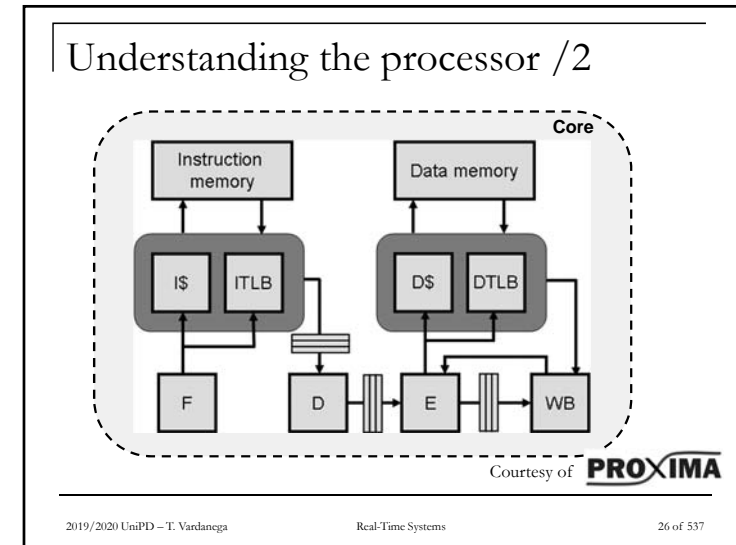
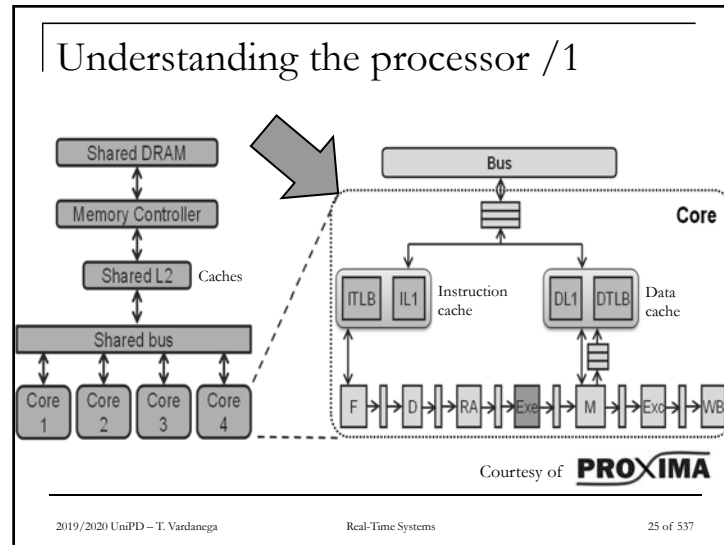
Meeting real-time requirements

- Minimizing the application tasks' average response time may matter to general-purpose computing; it does **not** to RTS!
 - Real-time computing is **not** equivalent to fast computing
- Given real-time requirements and a HW/SW implementation, how can one show that those requirements are met?
 - Testing and simulation are **not** sufficient (obviously!)
 - Maiden flight of space shuttle, 12 April 1981: there was a 1/67 probability of a *transient overload* occurring at system initialization
 - It never showed up in testing; it did at launch
- The answer to that requires seeking system-level **predictability**
 - This requires understanding the *worst case*
 - Which in turn requires understanding how execution works ...

2019/2020 UniPD - T. Vardanega

Real-Time Systems

24 of 537



Definitions /1

- **Job**
 - Unit of work selected for execution by the scheduler
 - Needs physical and logical *resources* to execute
 - Each job has an entry point where it awaits activation
- **Task**
 - Unit of functional and architectural composition
 - Issues one job at a time, until completion, to perform actual work
 - One such task is said to be *recurrent*

2019/2020 UniPD - T. Vardanega

Real-Time Systems

29 of 537

An initial taxonomy /2

- **Periodic** tasks
 - Their jobs become ready at regular intervals of time, T
 - Their arrival is synchronous to some time reference
- **Aperiodic** tasks
 - Recurrent but irregular
 - Their arrival cannot be anticipated (asynchronous)
- **Sporadic** tasks
 - Their jobs become ready at variable times but at bounded minimum distance from one another

2019/2020 UniPD - T. Vardanega

Real-Time Systems

30 of 537

Definitions /2

- **Release time**
 - When a job should become eligible for execution
 - The corresponding trigger is called **release event**
 - There may be some temporal delay between the arrival of the release event and when the scheduler actually recognizes the job as ready
 - May be set at some *offset* from system start time
 - The offset of the first job of task τ is named **phase**, ϕ , and it is one of the attributes of τ

2019/2020 UniPD - T. Vardanega

Real-Time Systems

31 of 537

Definitions /3

- **Deadline**
 - The time by which a job must complete its execution
 - May be $<$ (*constrained*), $=$ (*implicit*), $>$ (*arbitrary*) than the next job's release time
- **Response time**
 - The time span between the job's release and its actual completion
- The longest admissible response time for a job j_i is termed the job's **relative deadline**, D_i
- The algebraic summation of release time and relative deadline is termed **absolute deadline**, d_i

2019/2020 UniPD - T. Vardanega

Real-Time Systems

32 of 537

A timeline

Example

Without loss of generality, timelines have events occur at integral points in time

↑ = job release
↓ = job deadline

Job is released at time 3.
It's (absolute) deadline is at time 10.
It's relative deadline is 7.
It's response time is 6.

Jim Anderson Real-Time Systems Introduction - 18

2019/2020 UniPD - T. Vardanega Real-Time Systems 33 of 537

Definitions /4

- **Hard deadline**
 - If the consequences of a job completing past the deadline are serious and possibly intolerable
 - Satisfaction must be proven off line
- **Soft deadline**
 - If the consequences of a job occasionally completing past the assigned deadline are tolerable
 - The quantitative interpretation of “occasional” may be established in probabilistic terms or in terms of *utility*
- **Firm deadline**
 - A soft deadline with utility ≤ 0 past the deadline point

2019/2020 UniPD - T. Vardanega Real-Time Systems 34 of 537

Definitions /5

- **Laxity** (aka *slack*)
 - $s(t) = (d - t) - r$ is the *slack* at time t of job J with deadline d and remaining time of execution r
 - A job with non-negative laxity meets its deadline
- **Tardiness**
 - The distance between a job's response time and its deadline
 - A job with negative laxity has tardiness
- **Usefulness**
 - Value of (residual) utility of the job's computational product as a function of its tardiness

2019/2020 UniPD - T. Vardanega Real-Time Systems 35 of 537

Utility function

Difficult to quantify

Usefulness

A firm deadline a one for which the value of the functional product drops to 0 at the expiry of the deadline

Can be computed

Laxity ≥ 0 Deadline Tardiness > 0 Time

This is an interesting notion but difficult to apply and verify

2019/2020 UniPD - T. Vardanega Real-Time Systems 36 of 537

Abstract models /1

- **Active resources** (processor, server)
 - They “do” what they have to
 - Execute machine instructions, move data, process queries, etc.
 - Jobs must acquire them to make progress toward completion
 - This need is the cause of contention
- Active resources have a type
 - Those of the same type can be used interchangeably by a job
 - Those of different types cannot
 - For example, processors may have different speed, which affects the rate of progress for the jobs that run on them

2019/2020 UniPD – T. Vardanega

Real-Time Systems

37 of 537

Abstract models /2

- **Passive resources** (memory, shared data, semaphores, ...)
- A passive resource doesn't do anything per se, but jobs *may need* it to make progress
- They may be reused if use does not exhaust them
 - If always available in sufficient quantity to satisfy all needs, they are said to be *plentiful* and can be ignored
- Passive resources that matter to real-time systems are those that may cause *bottlenecks*
 - Access to memory may matter more (owing to *arbitration*) than memory itself (which may be considered plentiful)

2019/2020 UniPD – T. Vardanega

Real-Time Systems

38 of 537

Abstract models /3

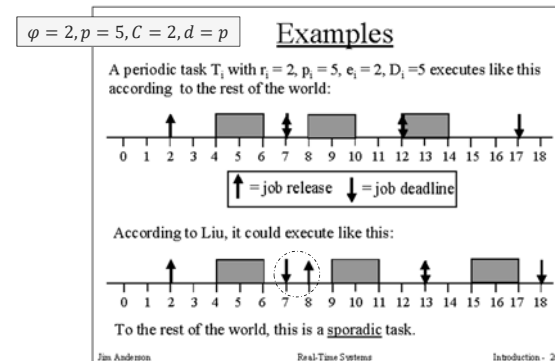
- Temporal parameters
 - **Jitter**
 - Variability in the release time or in the time of input (data freshness) or output (stability of control)
 - **Inter-arrival time**
 - Separation between the release time of successive jobs which are not strictly periodic
 - Job is *sporadic* if a guaranteed minimum such value exists
 - Job is *aperiodic* otherwise
 - **Execution time, C**
 - For any job J_i , C_i may vary between a *best-case* (BCET) C_i^b and a *worst-case* (WCET) C_i^w

2019/2020 UniPD – T. Vardanega

Real-Time Systems

39 of 537

Periodic task and sporadic task



2019/2020 UniPD – T. Vardanega

Real-Time Systems

40 of 537

Abstract models /4

■ **Periodic model**

- Comprises periodic and sporadic jobs
- Accuracy of representation decreases with increasing jitter and variability of execution time
- **Hyperperiod** H_S of task set $S = \{\tau_i\}, i = 1, \dots, N$
 - Defined as LCM (least common multiple) of task periods $\{p_i\}$
- **Utilization**
 - For every task τ_i : defined as the ratio between execution time and period: $U_i = \frac{c_i}{p_i} \leq 1$
 - For the system (*total utilization*): $U = \sum_i U_i \leq m$, where m is the number of CPUs ($m = 1$, for now)

2019/2020 UniPD - T. Vardanega

Real-Time Systems

41 of 537

Abstract models /5

■ Fixing execution parameters

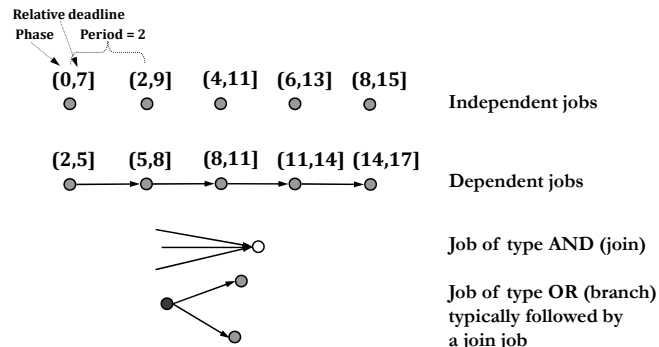
- The time that elapses between when a periodic job becomes ready and the next period p is certainly $\leq p$
- Setting phase $\varphi > 0$ and deadline $d < p$ for a job may help limit its output jitter (**why?**)
- The jobs of a system may be independent of one another
 - Hence they can execute in any order
- Or they may be subject to *precedence constraints*
 - As it is typically the case in collaborative architectural styles (e.g., producer - consumer)

2019/2020 UniPD - T. Vardanega

Real-Time Systems

42 of 537

Task precedence graphs



2019/2020 UniPD - T. Vardanega

Real-Time Systems

43 of 537

Types of precedence constraints

- One job's release time cannot follow that of a successor job
- **Effective release time (ERT)**
 - For a job J_i with predecessors $\{J_{k=1, \dots, i-1}\}$, ERT_i is the *latest* value between its own release time and the maximum effective release time of its predecessors, ERT_k , plus C_k
- One job's deadline cannot precede that of a predecessor job
- **Effective deadline (ED)**
 - For a job J_i with successors $\{J_{k=i+1, \dots, n}\}$, ED_i is the *earliest* value between D_i and the minimum effective deadline of its successors, ED_k , less C_k
- For single processors with preemptive scheduling, ERT and ED are the only precedence constraints of consequence

2019/2020 UniPD - T. Vardanega

Real-Time Systems

44 of 537

Abstract models /6

- Fixing design parameters
 - Permissibility of job preemption
 - May depend on the capabilities of the execution environment (e.g., *non-reentrancy*) but also on the programming style
 - Preemption causes time and space overhead
 - Job *criticality*
 - May be assimilated to a priority of execution eligibility
 - In general, it indicates which activities must be guaranteed, perhaps even to the detriment of others
 - Permissibility of resource preemption
 - Some resources are intrinsically preemptable
 - Others do not permit it

Which ones?

2019/2020 UniPD - T. Vardanega

Real-Time Systems

45 of 537

Abstract models /7

Recall
 BCET: best-case execution time
 WCET: worst-case execution

- Selecting jobs for execution
 - The scheduler assigns a job to the processor resource
 - The resulting assignment is termed *schedule*
- A schedule is *valid* if
 - Each processor is assigned to at most 1 job at a time
 - Each job is assigned to at most 1 processor at a time
 - No job is scheduled before its release time
 - The scheduling algorithm ensures that the amount of processor time assigned to a job is \geq than its BCET and \leq than its WCET
 - All precedence constraints in place among tasks as well as among resources are satisfied

2019/2020 UniPD - T. Vardanega

Real-Time Systems

46 of 537

Abstract models /8

- A *valid schedule* is said to be *feasible* if it satisfies the temporal constraints of every job
- A *job set* is said to be *schedulable* by a scheduling algorithm if that algorithm always produces a *valid* schedule for that problem
- A *scheduling algorithm* is *optimal* if it always produces a *feasible* schedule when one exists
- Actual systems may include multiple schedulers that operate in some hierarchical fashion
 - E.g., some scheduler governs access to logical resources; some other schedulers govern access to physical resources

2019/2020 UniPD - T. Vardanega

Real-Time Systems

47 of 537

Abstract models /9

- Two algorithms are of prime interests for real-time systems
 - The *scheduling algorithm*, which we should like to be *optimal*
 - Comparatively easy problem
 - The *analysis algorithm* that tests the *feasibility* of applying a scheduling algorithm to a given job set
 - Much harder problem
- The scientific community, but not always in full consistency, divides the analysis algorithms in
 - *Feasibility tests*, which are exact
 - Necessary and sufficient
 - *Schedulability tests*, which are only sufficient


2019/2020 UniPD - T. Vardanega

Real-Time Systems

48 of 537

Further characterization /1


	Time-Share Systems	Real-Time Systems
Capacity	High throughput	Ability to meet timing requirements: Schedulability
Responsiveness	Fast average response	Ensured worst-case latency
Overload	Fairness	Stability of critical part



2019/2020 UniPD - T. Vardanega Real-Time Systems 49 of 537

Further characterization /2

- The design and development of a RTS mind the worst case before considering the average case (if at all)
 - Improving the average case is of no use and it may even be counterproductive
 - The cache addresses the average case and therefore operates *adversarially* to the needs of real-time systems
- Stability of control prevails over fairness
 - The former concern is selective the other general
- When feasibility is proven, starvation is of no consequence
 - The non-critical part of the system may even experience starvation



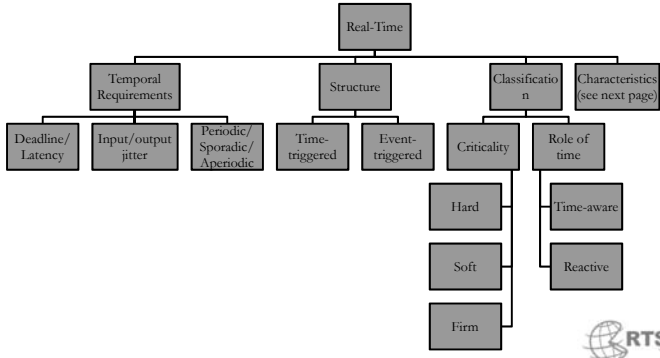
2019/2020 UniPD - T. Vardanega Real-Time Systems 50 of 537

Summary /1

- From initial intuition to more solid definition of real-time embedded system
- Survey of application requirements and key characteristics
- Taxonomy of tasks
- Dispelling false myths
- Introduced abstract models to reason in general about real-time systems

2019/2020 UniPD - T. Vardanega Real-Time Systems 51 of 537

Summary /2



```

    graph TD
      RT[Real-Time] --> TR[Temporal Requirements]
      RT --> S[Structure]
      RT --> C[Classification]
      RT --> CH[Characteristics see next page]
      TR --> DL[Deadline/Latency]
      TR --> IO[Input/output jitter]
      TR --> PS[Periodic/Sporadic/Aperiodic]
      S --> TT[Time-triggered]
      S --> ET[Event-triggered]
      C --> CR[Criticality]
      C --> RTIME[Role of time]
      CR --> H[Hard]
      CR --> SO[Soft]
      CR --> FI[Firm]
      RTIME --> TA[Time-aware]
      RTIME --> RE[Reactive]
    
```

2019/2020 UniPD - T. Vardanega Real-Time Systems 52 of 537

