Real-Time (Paradigms) (1)

1. Temporal Specifications

RT systems are in essence *responsive (reactive),* i.e. responding to events from the environment (user).

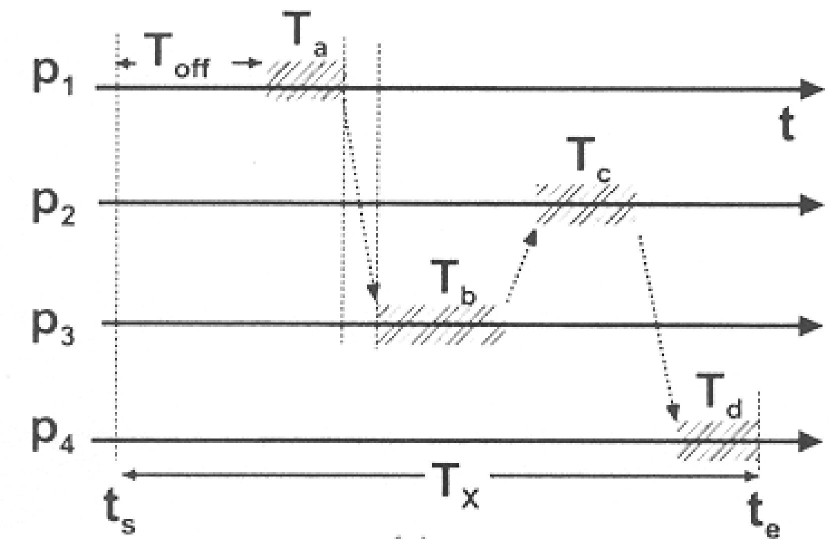
Response Time

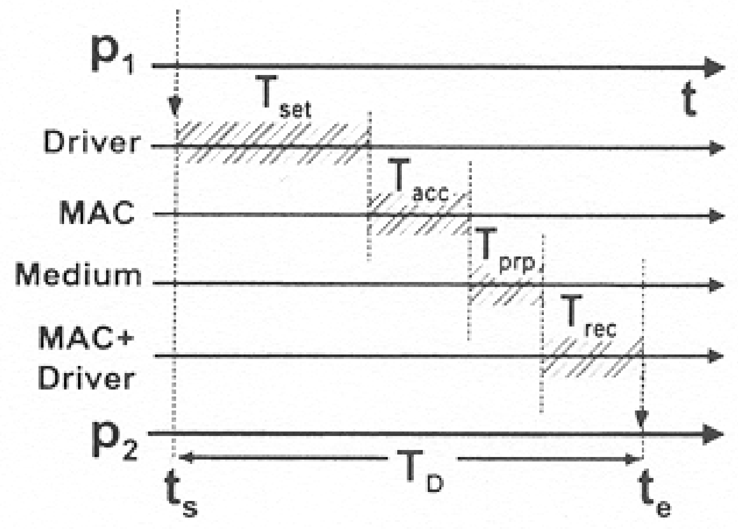
Interval between the occurrence of an input event and the first related output event

Timed Action

Execution of an action A such that its termination event happens within an interval T_A from a reference real time instant t_A.

Timing Analysis of an action: (a) Computation (b) Communication



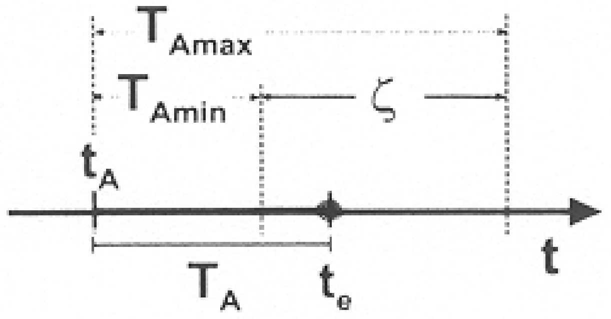


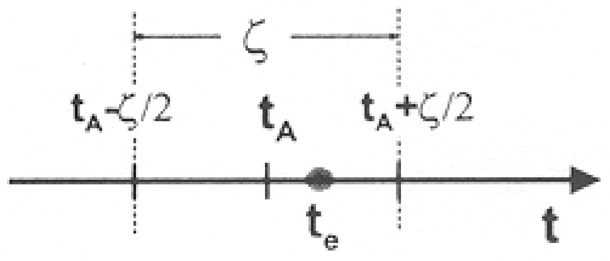
Real-Time (Paradigms) (2)

Jitter

variance in the duration of an action execution or imprecision in the positioning of its termination event.

Example

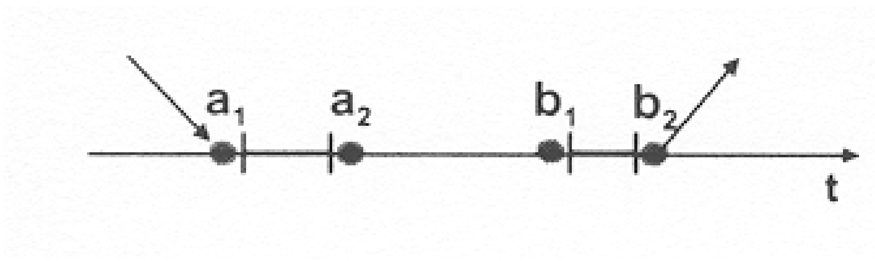


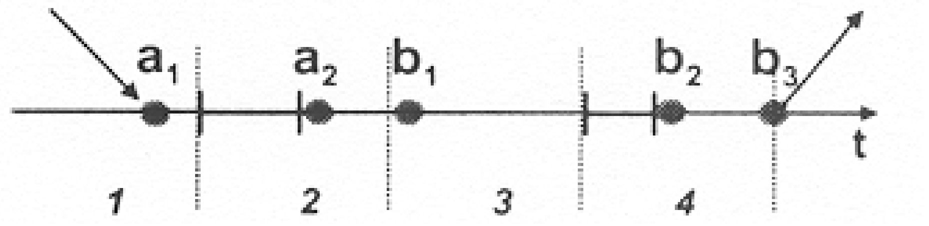


Mainly two approaches of triggering timed actions:

- *event-triggered:* system reacts upon the occurrence of an input event
- *time-triggered:* system reacts upon the command of a clock

Example





Real-Time (Paradigms) (3)

System predictability depends on the predictability of the inputs received from the environment which again depends on the class of application.

Trade-off:

Guaranteeing system predictability is simpler given a model assuming for regular (periodic) arrival patterns but: potential lack of coverage

Assuming a model accepting irregular (aperiodic) arrival patterns are closer to reality but: designing and proving that such systems are predictable is much more difficult

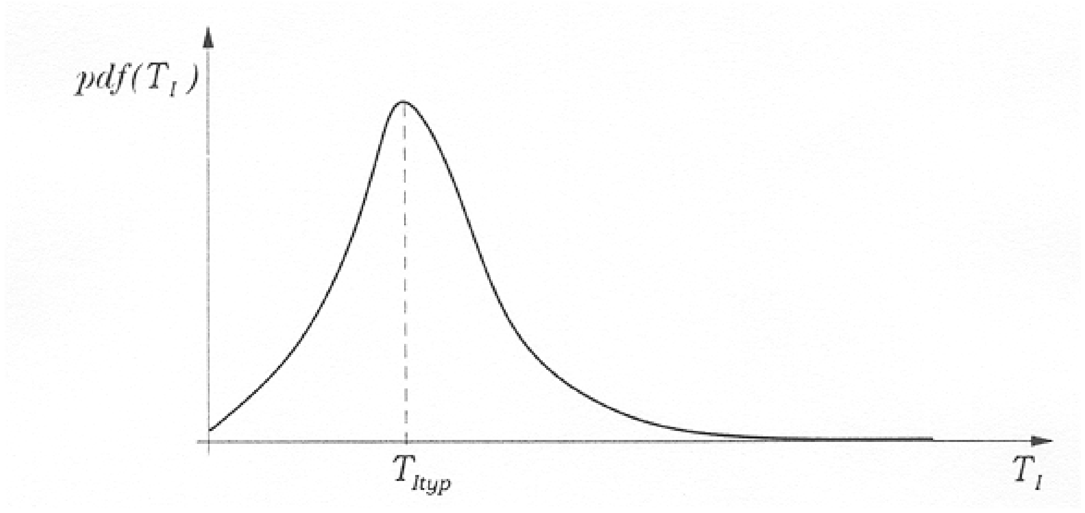
W.r.t the arrival of tasks, 3 types can be distinguished:

Periodic are such where tasks are released regularly at fixed rates (periods).

Aperiodic are such where tasks are released irregularly at some unknown and possibly unbounded rate.

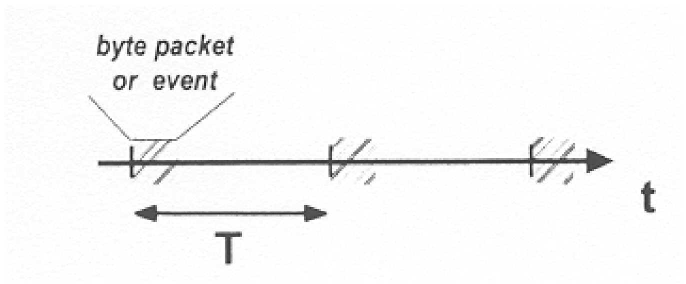
Sporadic tasks are such where tasks are released irregularly with some bounded rate. This rate is characterized by a minimum interarrival period.

Aperiodic Distribution

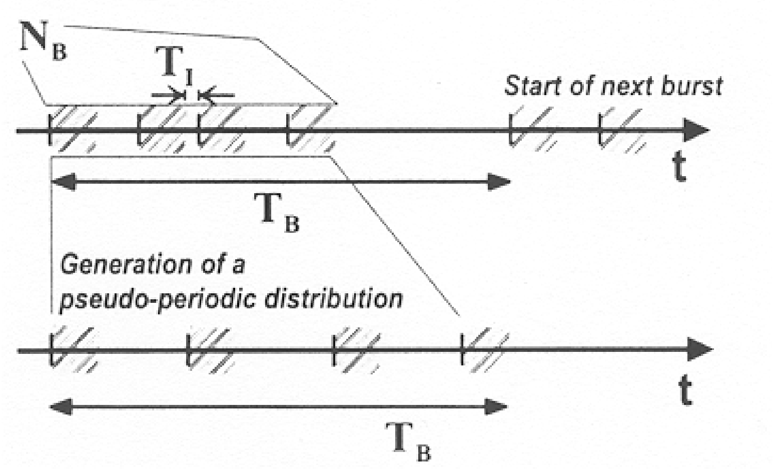


Real-Time (Paradigms) (4)

Periodic Distribution



Sporadic Distribution



burst period T_B: lower bound for the interval between the start of two consecutive bursts

burst length N_B: upper bound of number of events occurring in one burst

inter-arrival time T_I: lower bound for the interval between the occurrence of two consecutive events

Utilization Factor

measure of percentage a resource is used over a given time interval

Real-Time (Paradigms) (5)

2. Entities and Representatives

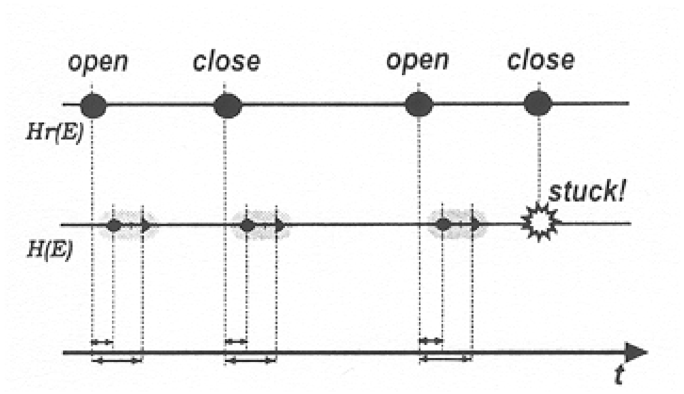
RT entity: element of the environment the state of which can be read or written, but not both

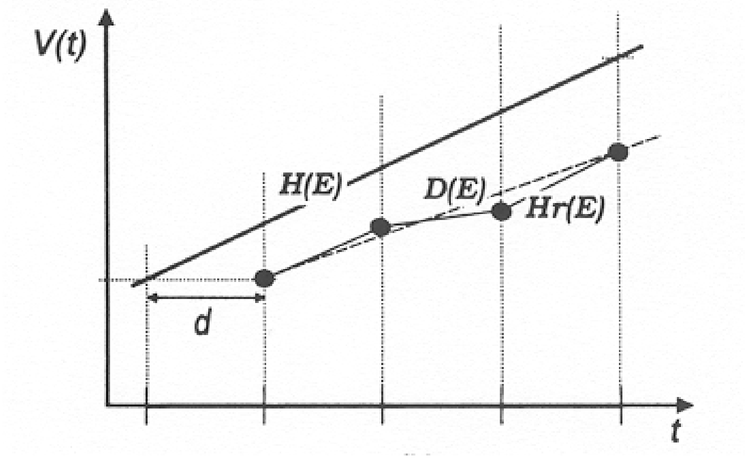
Representative: element of the (controlling) computer system which observes or acts on a RT entity´s state

The state of a RT entity is not accurately reflected in its representative at all times during system evolution!

---> A representative emulates its RT entity with an error in the value of state, or in the time of state changes, or both.

Examples for RT Entity - Representative Relationship





Real-Time (Paradigms) (6)

3. Time-Value Duality

Time-Value entity: RT entity E the value V of which depends on time, i.e. $V = E(t)$

For operations using time-value entities to be correct, two problems must be solved:

1. ensuring the correct observation of
   - the instantaneous value of the RT entity and
   - its positioning in the timeline, i.e. the corresponding time of the value
2. ensuring the correct use of the observation, i.e. using the observed value while it is still valid

ad 1)

Given a known V_0, observation $(r(E_i)(t_i), T_i)$ is *consistent in the value domain*, if and only if $v_i <= V_0$

Given a known Z_i, observation $(r(E_i)(t_i), T_i)$ is *consistent in the time domain*, if and only if $\zeta_i <= Z_0$

A set of observations is *mutually consistent,* if they are consistent and the timestamps of all observations fall within a given interval Z_m (also called *relative validity interval* in the context of databases)

ad 2)

Given a known V_a, observation $(r(E_i)(t_i), T_i)$ is *temporarily consistent at t_a* $>= T_i$,

if and only if $|E_i(t_a)-E_i(T_i)| <= V_a$ (also called *absolute validity interval* in the context of databases)

The first problem addresses the consistency property w.r.t. the observation instant, the second one deals with the evolution of consistency over time, a specific characteristics of time-value entities.

Real-Time (Paradigms) (7)

1. Time and Clocks

Time is a very useful artifact to represent the
- ordering
- sequencing
- synchronizing

of events in any system.

The passage of time is marked by an abstract monotonically increasing **continuous** function, called *real time*
Along history, time has been represented (measured) in different ways, mainly dependent on how the unit of time,called *second*, was measured.
timeline: graphical representation of time units as sequence of points over a straight line (digitized time)

The use of time in computer systems addresses two aspects:
- observing and recording the place of events in a timeline (ordering, sequencing)
- enforcing the future positioning of events in the timeline (synchronizing)

Real-Time Paradigms (8)

UT (AT, 1833)	Universal Time (UT) Mittlere Sonnenzeit, gemessen am Greenwich 0-Meridian (GMT). Basiert auf der mittleren Länge eines Sonnentags, d.h. auf der Erdrotation
Zeitzonen (1884):	Gebiete für die dieselbe Zeit festgelegt ist. 1884 wird die Welt in 24 Zeitzonen aufgeteilt. Die Zeitzonen unterscheiden sich von UT (GMT) ganzzahlig um jeweils 1 Stunde .
ET (AT, 1955)	Ephimeridenzeit (ET), basiert auf der Umlaufzeit der Erde um die Sonne. Harold Spencer Jones stellte 1939 fest, daß die Rotation der Erde variiert, die Umlaufzeit um die Sonne nicht. 1 Sekunde der ET wird festgelegt als der 1/31.566.925,9747 Teil des tropische Jahres, das am Mittag des 1. Januars 1900 begann. (Tropisches Jahr: Periode zwischen zwei aufeinanderfolgenden Umläufen der Sonne durch den Himmelsäquator in derselben Richtung.)
UT2 (AT, 1960)	Zeit, basierend auf und gemittelt über den lokalen Beobachtungen verschiedener über die Erde verteilter Observatorien und anschließend nochmals auf empirischer Basis korrigiert
TAI (PT, 1961)	Temps Atomique International (TAI) basiert auf mehreren koordinierten Cäsium-Uhren. Fortlaufende Zeitzählung, beginnend mit dem 1.Januar 1958 0 Uhr UT2-Zeit (daher konsistent mit UT2). 1 Sekunde der TAI ist 9 192 631 770 mal die Periode der Strahlung des Atoms Cäsium 133. Driftrate $\rho \approx 10^{-14}$, d.h. Abweichung ca. 1 Sek / 300000 Jahre
UTC (PT, 1972)	Universal Time Coordinated (UTC) basiert auf TAI, wird aber ständig an UT2 angepaßt. Immer wenn UTC und UT2 mehr als 800 ms auseinander gedrifted sind, wird eine “Schaltsekunde” eingefügt. UTC beginnt am 1. Januar 1972. Seit dieser Zeit sind bis 1992 15 Schaltsekunden eingefügt worden. UTC ist damit eine an AT angepaßte physikalische Zeit.

Real-Time (Paradigms) (9)

local physical clock:

implements in hardware the mapping of real time t into a clock time pc(t), which is a monotonically increasing **discrete** function. They are based typically on oscillators such as quartz. The timeline now becomes a sequence of discrete ticks.

They are mainly characterized by the parameters (also representing its imperfections)

- *granularity g:* time difference between two consecutive ticks t(i) and t(i+1): g:= pc(t(i+1)) - pc(t(i))
- *drift rate* ρ:□□□□□□□□ constant denoting the drift of a physical clock from real time

  $\rho \approx 10^{-5}$, i.e. several microseconds per second, e.g. ca. 36 ms after 1 hour, almost 1 s after 1 day
- *clock rate*: : $1-\rho \leq (pc(t(i+1) - pc(t(i))) / \Delta t \leq 1+\rho$ for $\Delta t = t(i+1) - t(i)$

local clocks can be used to

- represent a timer to set *timeouts*
- timestamp local events
- measure local durations

They cannot be used for timing analysis regarding global events in a distributed systems because of ρ

—> need to synchronize all local clocks by means of a *clock synchronization algorithm*

global clocks

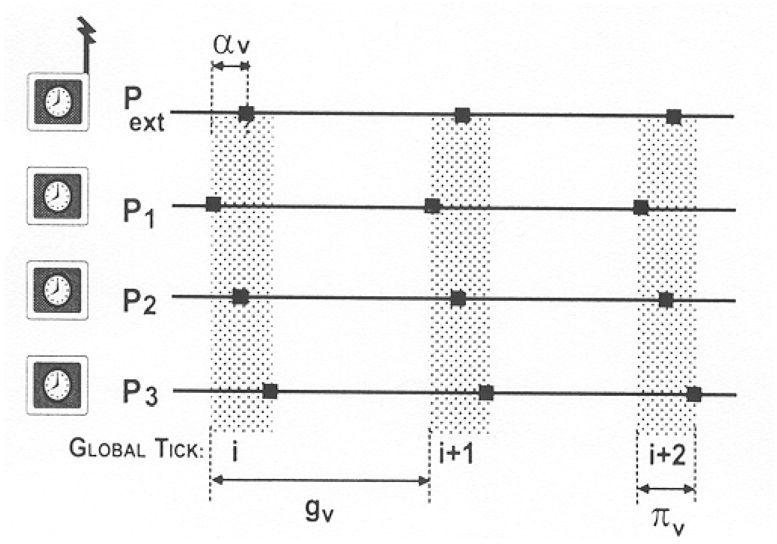
A global clock in a distributed system is built by synchronizing in periodic rounds all local clocks as close as possible to the same initial value.

virtual clock: the time vc(t) delivered by a synchronized physical clock

The set of virtual clocks under the control of the synch. algorithm. constitutes the global clock of the system

Real-Time Paradigms (10)

Properties of a Global Clock:



precision π_v denotes the maximum deviation between two corresponding ticks of any two virtual clocks, as seen by an outside observer, measured by the external reference clock representing the real time.

π_v := max{for all i,k,l : $|vc_k(t(i)) - vc_l(t(i))|$}

granularity g_v denotes the time interval between two consecutive global ticks

accuracy α_v□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□between a tick of any of the virtual clocks and the corresponding tick of the external reference clock P_{ext}.

α_v := max{for all i,k : $|vc_k(t(i)) - P_{ext}(t(i))|$}

convergence δ_v denotes the maximum deviation between any two ticks of the virtual clocks immediately after the termination of a synchronization round (minimal deviation:= maximal precision).

δ_v := max{for all k,l : $|vc_k(t(0)) - vc_l(t(0))|$}

convergence δ is a measure for the quality of the clock synch. algorithm (internal synchronization)

accuracy α is a measure for the external synchronization, e.g. by means of GPS

Real-Time Paradigms (11)

The definitions above imply the following relations: π≥δπ≤ □α , □> π
(precision cannot be better than convergence and at least twice the accuracy, it is senseless to select a granularity finer than the precision)

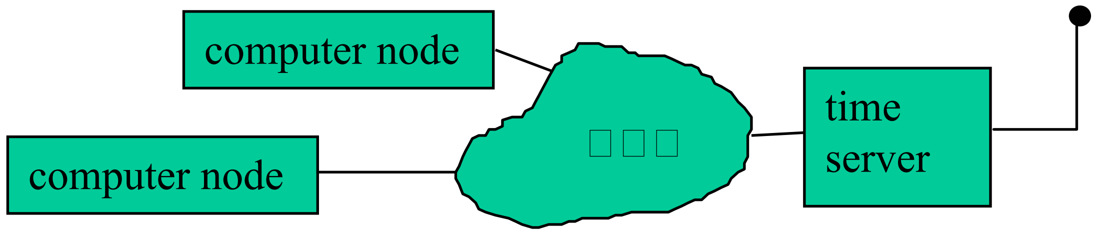
→ any globally visible event e is timestamped t(e) by different nodes of the system with at most one tick difference

→ let d := |t(e_1) - t(e_2)| (No. of ticks); if d < 2 --> no physical order of the events e_1 and e_2 can be deduced

→ granularity (which itself depends on precision) determines the resolution of the global time grid

Required components to define a global time basis:

- an external reference time, e.g. UTC-based
- local physical clocks
- a synchronization algorithm



GPS (Global Positioning System):

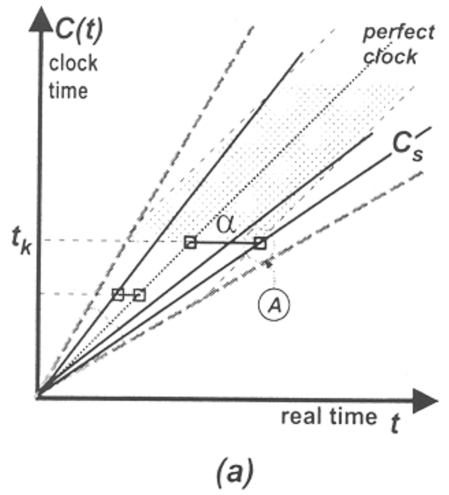
- network of 21 satellites covering earth surface
- equipped with cesium atomic clocks with high stability (ρ_g ca.10^{-14} , i.e. 1sec drift in 3 000 000 years)
- GPS-receiver clocks mostly provide UTC with an accuracy of $\alpha_g \leq$ 100ns

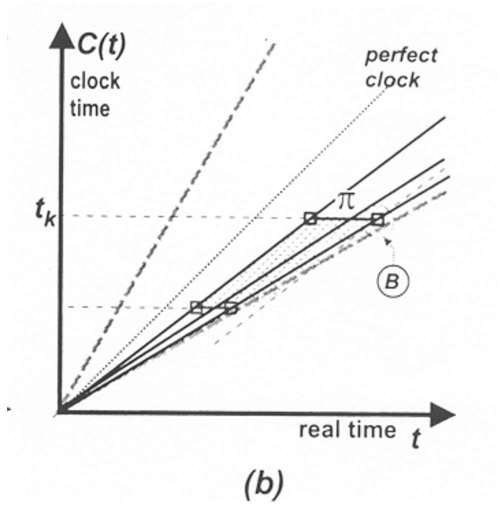
□□ (being under the light cone of

Real-Time (Paradigms) (12)

7. Clock Synchronization

Behavior of a Clock over Time:(a) Accuracy Drift; (b) Precision Drift





clock synchronization:

The process of maintaining the required properties of precision(*internal synch*.) and accuracy (*external* + *internal synch* ($\Pi = 2\,\alpha$) of a set of clocks

Assumption: the drift rate of each individual clock is bounded

---> this allows to predict the maximum deviation after a given time interval.

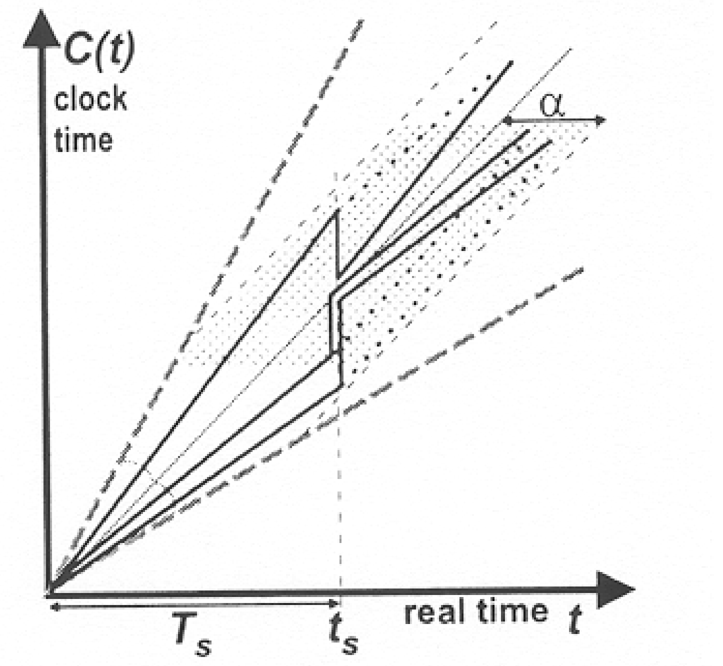
Real-Time (Paradigms) (13)

Basic result:

Convergence, i.e. the precision achieved immediately after the synchronization, cannot be made arbitrarily small due to a remote *clock reading error* caused by the variance in message delays.

resynchronization interval T_S: time interval between successive synchronizations

Clock Synchronization



amortization: rate correction factor applied when clock is read (instead of instantaneously changing the clock time)

state synchronization: adjusting clocks by changing their value (done by software, PC (hardware) clock remains unchanged)

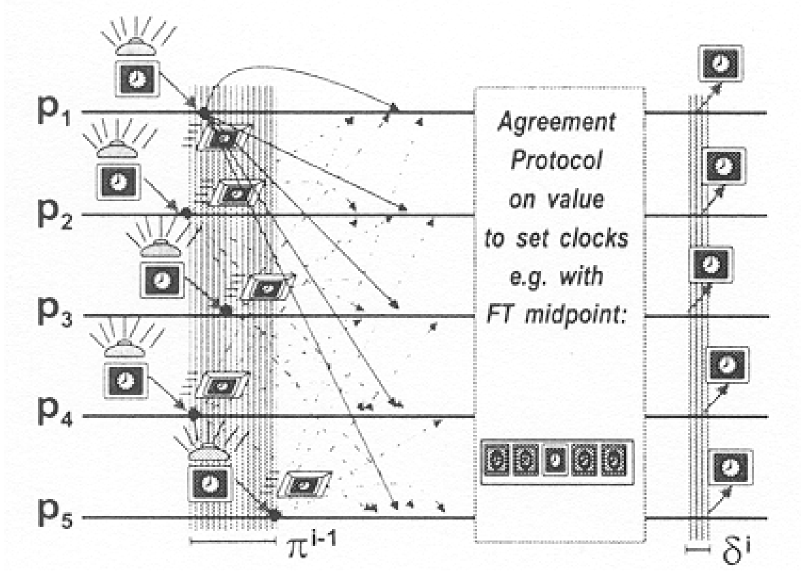
rate synchronization: adjusting the rate at which the hardware clock ticks

Real-Time (Paradigms) (14)

Internal Synchronization

Respective algorithms are normally cooperative, .i.e. each clock applies a *convergence function* to the values of each process.

Averaging Synchronization



p1
p2
p3
p4
p5
Agreement Protocol on value to set clocks e.g. with FT midpoint:
π^{i-1}
δ^i

convergence functions could be, e.g.:

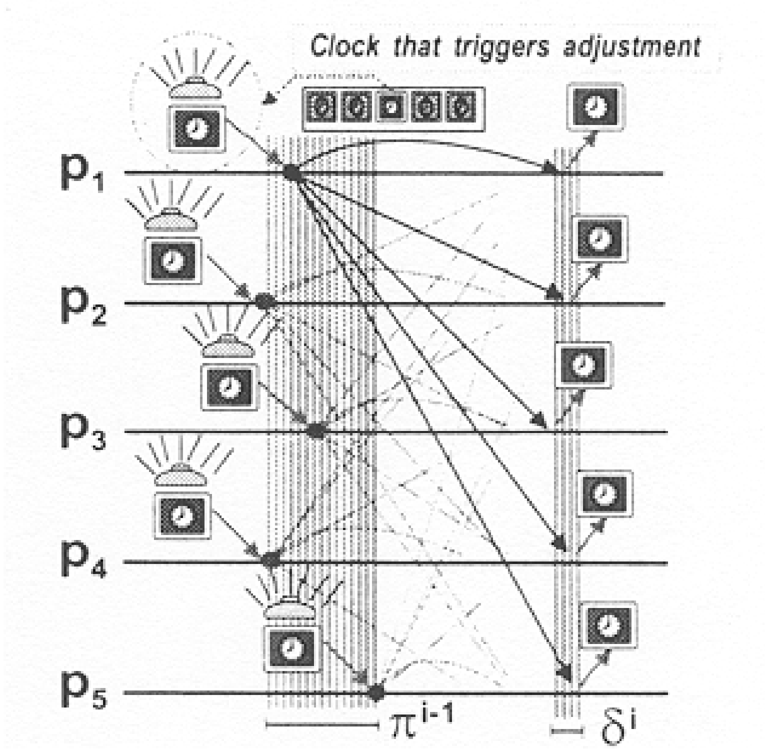
- average
- midpoint

Real-Time (Paradigms) (15)

Non-Averaging Synchronization

Instead of disseminating individual clock values and subsequently applying convergence functions agreed on, here, processes disseminate a control message to signal end of a synchronization interval.

Example



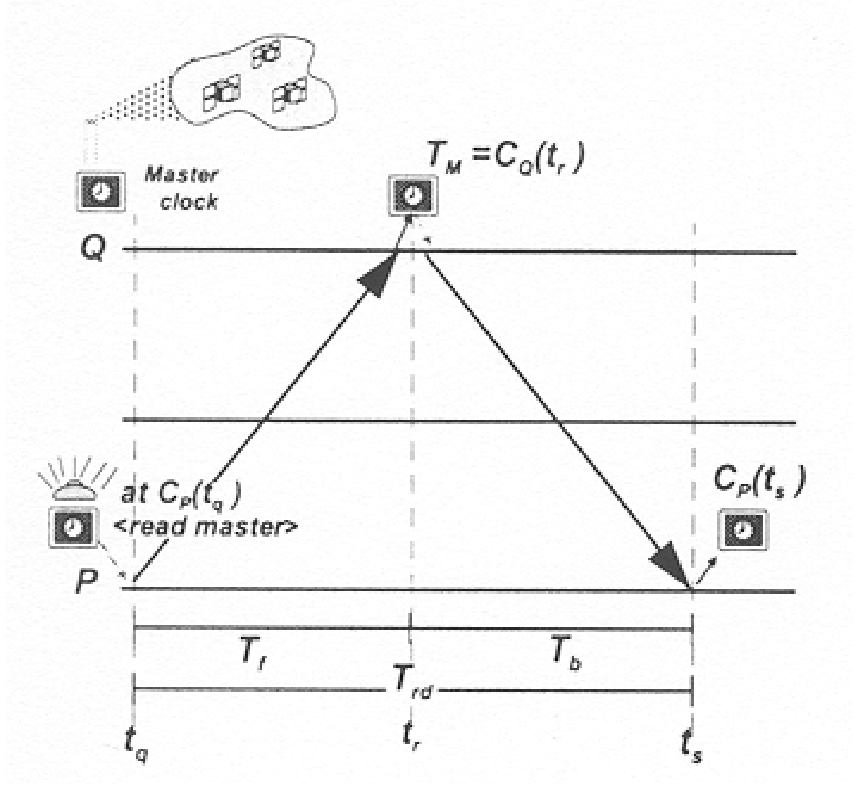
Real-Time (Paradigms) (16)

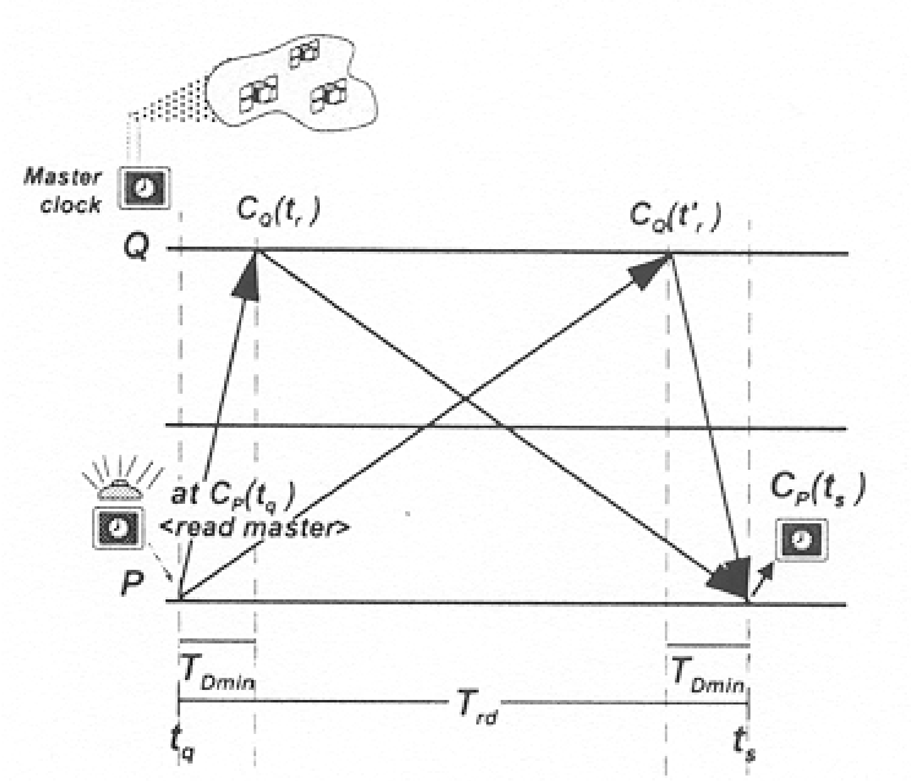
External Synchronization

Respective algorithms are not cooperative, but master-slave.

Simplest method: Multicasting of time by the master (used to synchronize GPS receiver units)

Round-Trip External Synchronization





Real-Time (Paradigms) (17)

4. Scheduling

Scheduling is concerned with assigning needed resources in order to execute tasks such that the system meets the timing requirements. Scheduling is the backbone of a RT system and, therefore, is the most widely researched topic within RT systems.

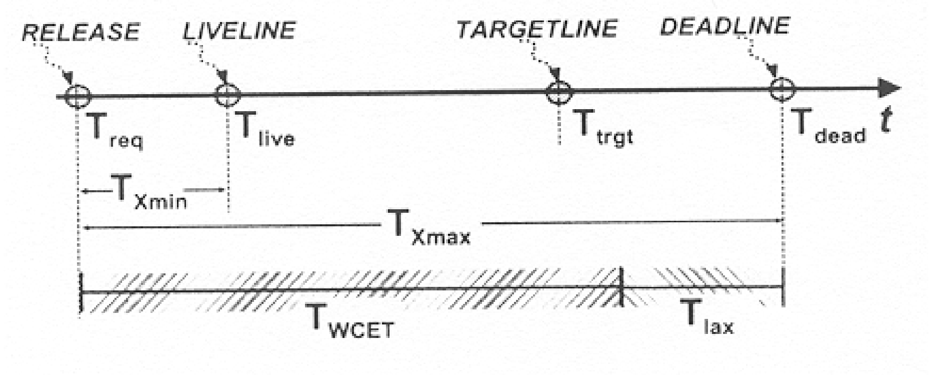
Policies of Non-RT (general purpose) systems aim at

- fairness
- high performance (throughput)
- high resource utilization

RT systems only aim at

- predictability, if necessary, in detriment of the other aims.

Important timing parameters of a task



Real-Time (Paradigms) (18)

W.r.t. the flexibility of tasks regarding their timing constraints and functionality, they can be classified as:

Hard tasks

All timing constraints must be met and optimal functionality is delivered.

Critical tasks

Their activation can be triggered later than the given release time.

Redundant tasks

All timing constraints are met and the delivered functionality(accuracy) is not optimal (gracefully degraded) but still acceptable (correct in the sense of in compliance with the overall specification).

Soft (best effort) tasks

Missing the deadlines of soft tasks can be tolerated.

Real-Time (Paradigms) (19)

Classification of scheduling algorithms:

Preemptive

The task being executed can be interrupted at any time in order to assign the processor to another task according to the used algorithm.

Non-preemptive

A task, once started, is executed by the processor until completion.

Static

Scheduling decisions are based on static (fixed) task parameters.

Dynamic

Scheduling decisions are based on dynamic (possibly changing at system run-time) task parameters

calendar-based

Tasks are executed according to a resulting calendar (time schedule).

Priority-based

Tasks are executing according to assigned (fixed or dynamically changing) priorities.

Independent

Release time of tasks does not depend on the termination time of other tasks

Real-Time (Paradigms) (20)

Static (off-line) scheduling

schedulability analysis is done off-line, i.e. before run-time

---> the used scheduling algorithm has complete a priori - knowledge about all relevant task parameters, i.e. a deterministic system and environment is assumed

Dynamic (on-line) scheduling

schedulability analysis is done on-line, i.e. at run-time

---> the used scheduling algorithm must not (cannot) have complete a priori - knowledge about all relevant task parameters

---> provides predictability w.r.t. individual task arrivals

(Timing) Fault-Tolerant scheduling

trading predictability and enhanced throughput for potentially degraded functionality of individual tasks

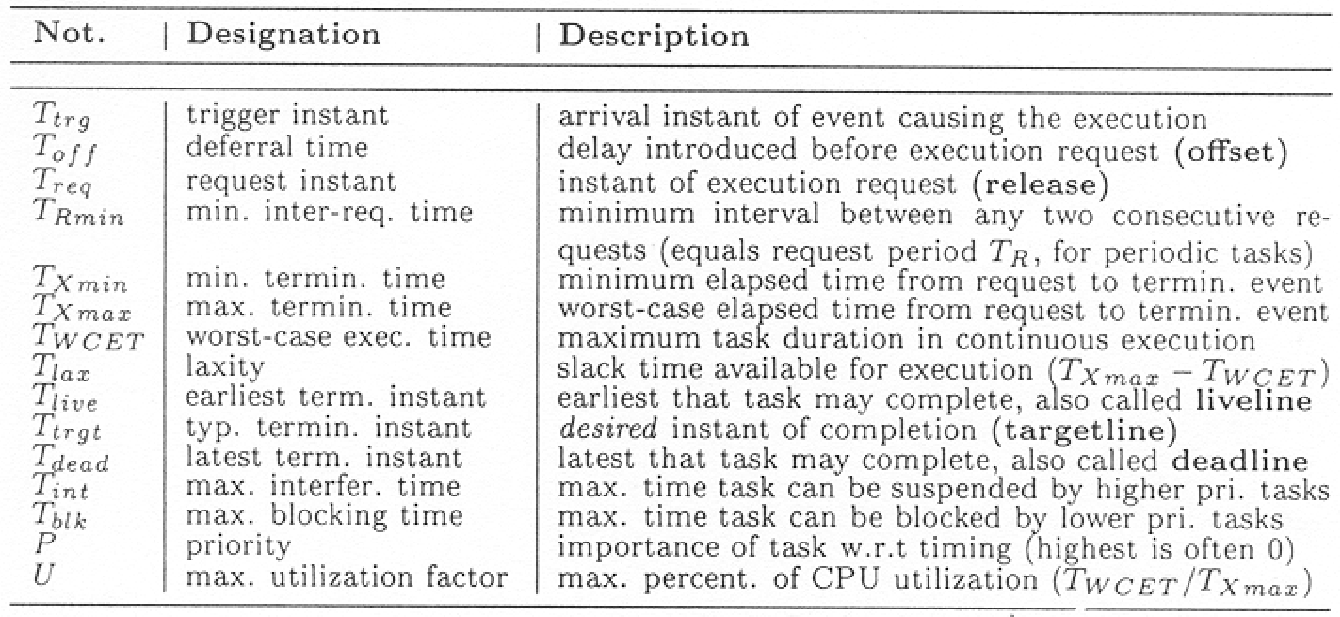
Schedulability

A set of tasks is *schedulable* or *feasible* if all deadlines are met by some algorithm.

An algorithm is *optimal* for a given task set if it fails to meet all deadlines only if no other algorithm can meet all deadlines, i.e. it always generates a feasible schedule if one exists.

Real-Time (Paradigms) (22)

Table of Task Execution Timing Parameters



Not.	Designation	Description
T_{trg}	trigger instant	arrival instant of event causing the execution
T_{off}	deferral time	delay introduced before execution request **(offset)**
T_{req}	request instant	instant of execution request **(release)**
T_{Rmin}	min. inter-req. time	minimum interval between any two consecutive requests (equals request period T_R, for periodic tasks)
T_{Xmin}	min. termin. time	minimum elapsed time from request to termin. event
T_{Xmax}	max. termin. time	worst-case elapsed time from request to termin. event
T_{WCET}	worst-case exec. time	maximum task duration in continuous execution
T_{lax}	laxity	slack time available for execution ($T_{Xmax} - T_{WCET}$)
T_{live}	earliest term. instant	earliest that task may complete, also called **liveline**
T_{trgt}	typ. termin. instant	*desired* instant of completion **(targetline)**
T_{dead}	latest term. instant	latest that task may complete, also called **deadline**
T_{int}	max. interfer. time	max. time task can be suspended by higher pri. tasks
T_{blk}	max. blocking time	max. time task can be blocked by lower pri. tasks
P	priority	importance of task w.r.t timing (highest is often 0)
U	max. utilization factor	max. percent. of CPU utilization (T_{WCET}/T_{Xmax})

Determining whether a given task set is feasible is called *schedulability testing.* The outcome can be

- *sufficient:* passing it indicates that it is feasible
- *necessary:* failing it indicates that it is not feasible
- *exact:* sufficient and necessary

Utilization-based Tests

- fail, if the generated schedule will use the CPU more than a given percentage
- are sufficient, but not necessary

□□□□

Real-Time (Paradigms) (23)

Processor utilization factor U:

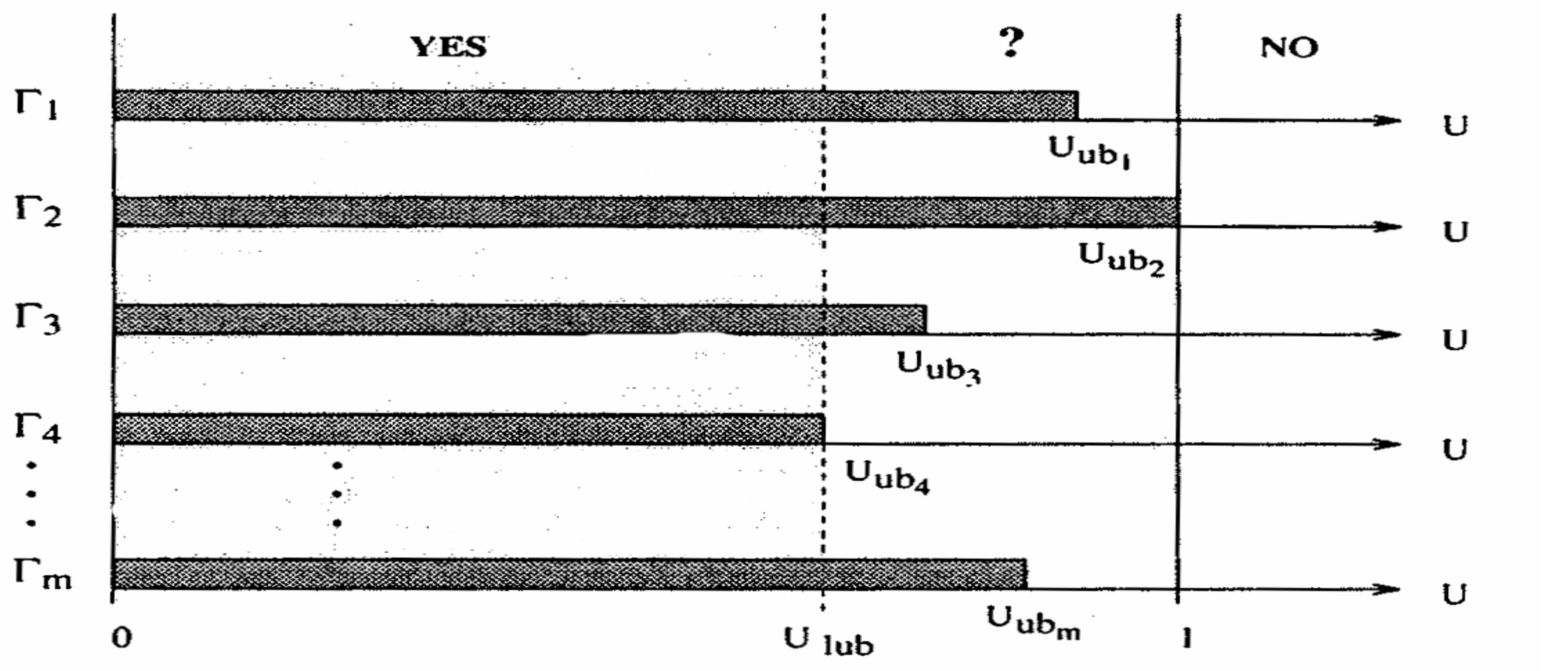
Given a finite set Γof n periodic tasks τ_ι, U is the fraction of processor time spent in the execution of Γ, i.e.

$U = \Sigma C_i/T_i$ (i = 1,...,n)

$U_{ub}(\Gamma,A)$ is the upper bound of U for Γ under a given algorithm A in order to be feasible

$U = U_{ub}(\Gamma,A)$ ––-> Γ is said to *fully utilize* the processor under A (full does not mean optimal utilization)

$U_{lub}(A) = \min U_{ub}(\Gamma,A)$ is the least upper bound for all Γ with $U = U_{ub}(\Gamma,A)$



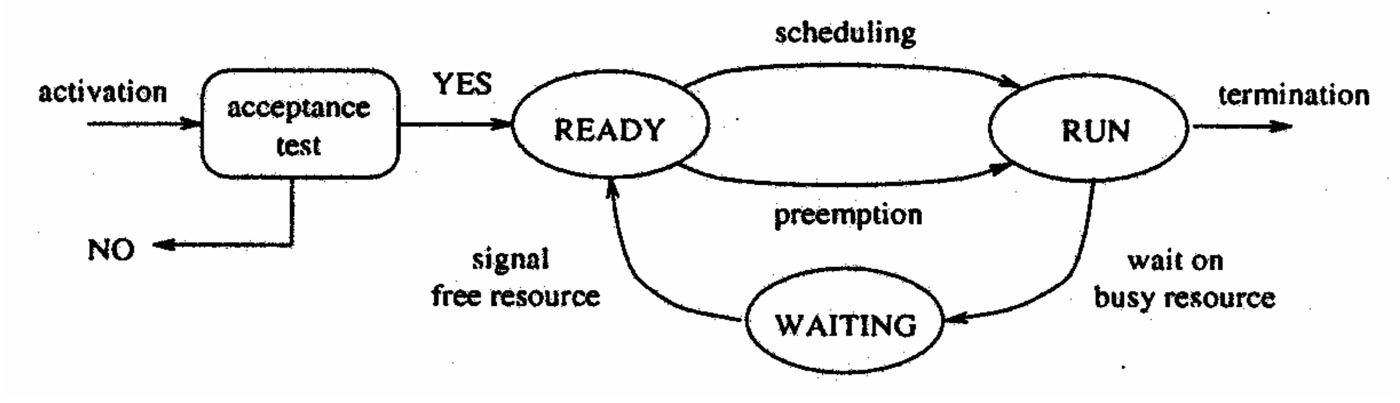
Real-Time (Paradigms) (24)

Response Time - based Tests

- determines for each task T_{max} by computing WCET + T_{int} and comparing it with $T_{dead.}$
- are exact

Acceptance Tests

- provide predictability w.r.t. individual task arrivals
- are sufficient

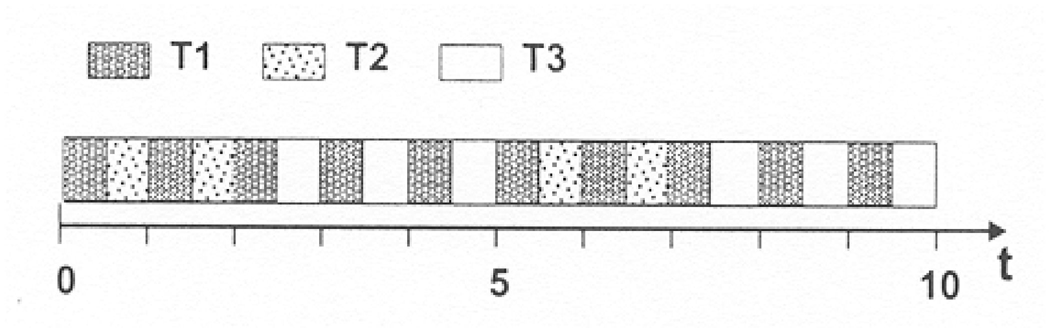


Rate-Monotonic Scheduling Algorithm (RM)

- designed for static scheduling of independent periodic tasks (all periods and WCET´s are known)
- the task´s priority is inversely related to its period ---> tasks with smaller periods have higher priorities
- it is preemptive and based on static priorities
- if for all tasks $T_{xmax} = T_R$, it is optimal among all fixed-priority algorithms
- $U_{lub} <= \ln 2$ is a sufficient condition for the schedulability test, $U_{lub} <= 1$, if the task set is *harmonic,* meaning that all periods are multiples of the smallest period

Real-Time (Paradigms) (28)

Example of a rate-monotic schedule



Heuristics for dealing with sporadic tasks:

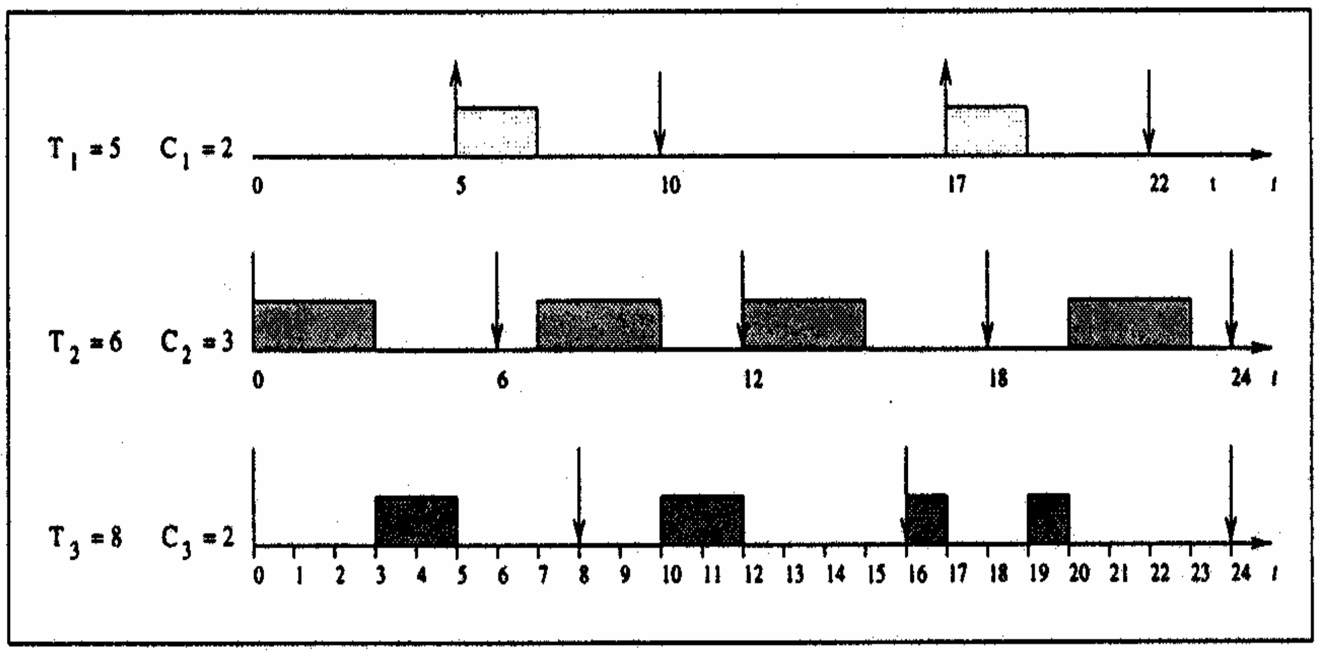
- modeling them as *pseudo-periodic* by defining $T_R = T_{Rmin}$
  Main drawback: most of the periods are empty --> very low processor utilization
- Adding a periodic server task with high priority to serve the pending sporadic requests (*sporadic server)*

Note:

Since RM is optimal among all static assignments, an improvement of the bound for U can be achieved only by using dynamic scheduling algorithms.

Real-Time (Paradigms) (29)

Example of an earliest deadline first - schedule

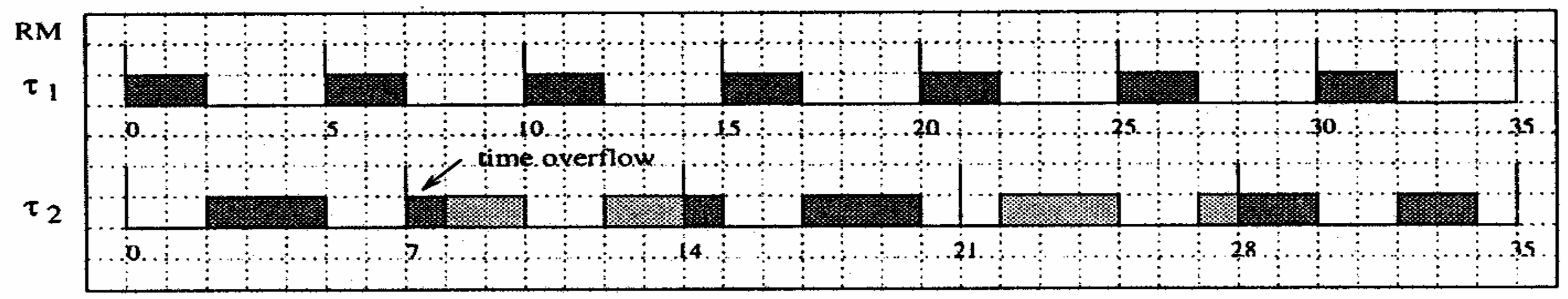


Earliest Deadline First Scheduling Algorithm (EDF)

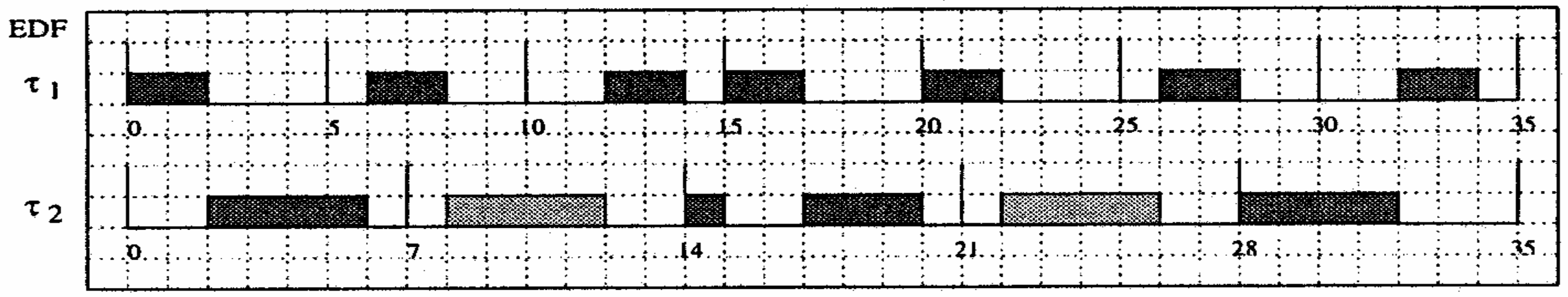
- designed for static and dynamic scheduling of independent periodic and sporadic tasks
- the task´s priority is inversely related to its absolute deadline ---> tasks with shorter deadlines have higher priorities
- it is preemptive and based on dynamic priorities
- It is optimal among all algorithms
- If used for static scheduling, U<= 1 is a sufficient condition for the schedulability test

Real-Time (Paradigms) (30)

Comparison EDF <---> RM by means of an example



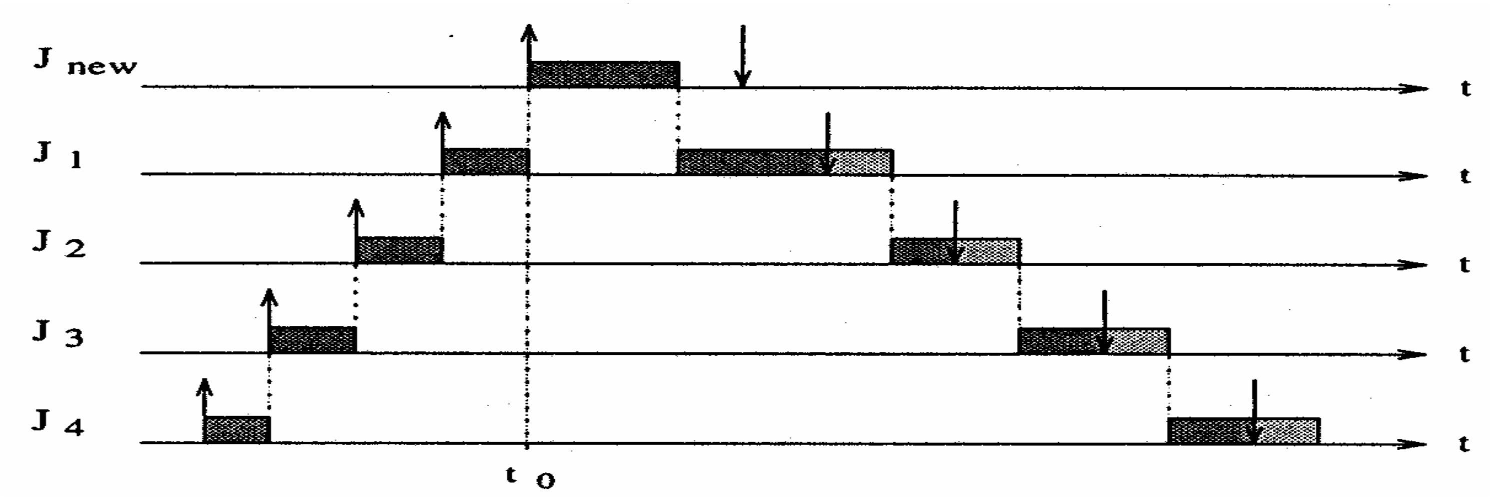
(a)



(b)

Real-Time (Paradigms) (31)

Example of the domino effect



The last example constitutes a best-effort approach
 ---> no feasibility checking is done
---> no individual task deadline can be guaranteed
---> provides no predictability

Classification of scheduling policies

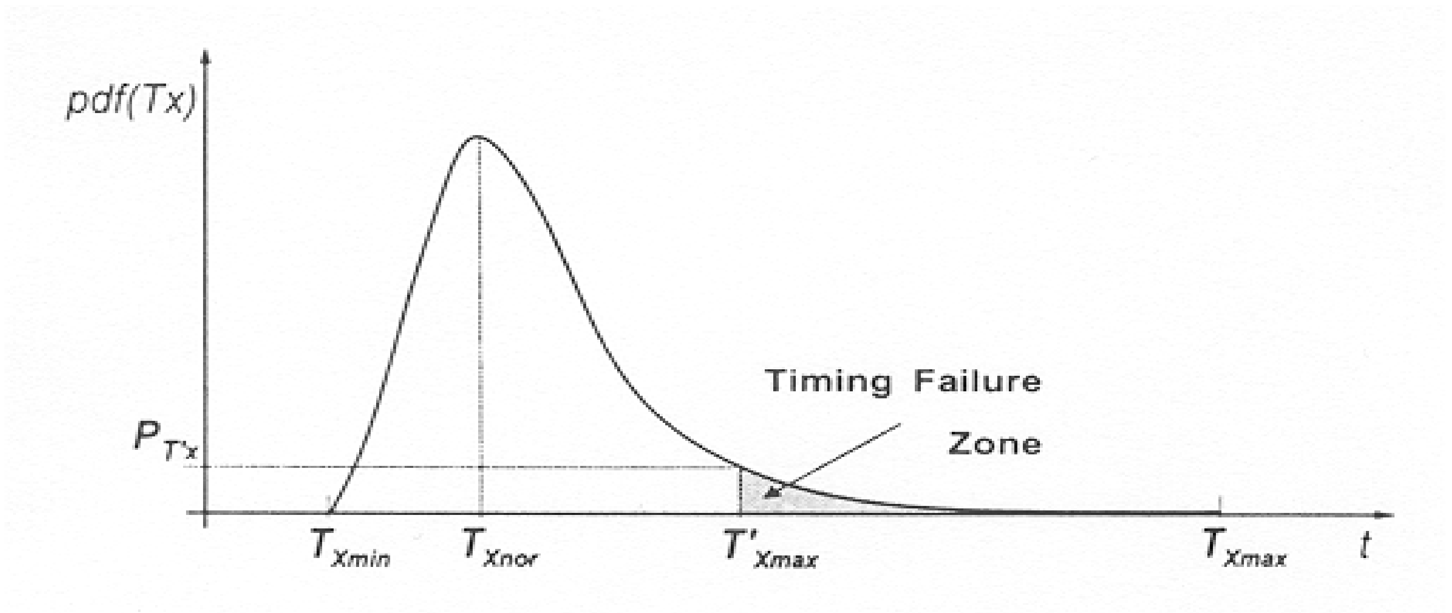
Several scheduling policies exist, depending on whether

- a system performs schedulability tests at all
- if so, when it is done
- what type of schedule is produced as a result of the analysis
- whether fault-tolerance is considered

Real-Time (Paradigms) (32)

Problem: What if applications with timing requirements can provide only uncertain timing parameters?

Distribution of Termination Times



Two important classes of guarantee-based dynamic scheduling for overload situations

- *robust*
  - different policies for task acceptance and guaranteed timely execution
  - often using a reclaiming mechanism for accepted but later rejected tasks
- *fault-tolerant*
  - trading functional redundancy for predictability

Real-Time (Paradigms) (33)

Problems with WCET's:

- dependent on hardware architecture,OS, compiler, PL ---> difficult to predict
- many features serve to improve average case behavior, n o t worst case behavior

  Examples:
  - caches, pipelining, virtual memory
  - interrupt handling, preemptions
  - optimizing compilers
  - recursions

- **Even more difficult if depending on the environment (embedded systems)**

Real-Time (Paradigms) (34)

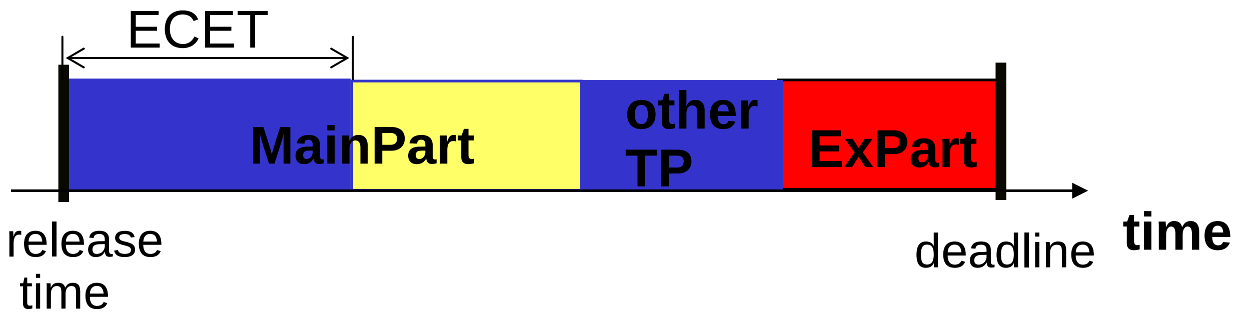
Goals of TAFT (Time - Aware Fault - Tolerant) Scheduling

- No Handling of tasks with unknown or too pessimistic WCETs
  - Introduction of Expected Case Execution Time (ECET)
- Still with Timing Guarantees
  - Scheduled exception handling **before** the deadline
- Fault-Tolerance with respect to timing errors
  - Graceful degradation in overload situations
  - Tradeoff between functionality and timing

Real-Time (Paradigms) (35)

TAFT Scheduling

- Each module is scheduled as a task pair consisting of a main part and an exception part
  - Main part: actual module functionality, ECET scheduled
  - Exception part: module specific exception handling, WCET scheduled
- Timing faults are confined to modules



Real-Time (Paradigms) (36)

Three-level Scheduling:

- Level One – ExceptionParts
    - Highest dispatching priorities
    - LRT (Latest Release Time - Reverse-EDF)
    - Tries to do everything as late as possible
- Level Two - MainParts executed within the reserved (and guaranteed) ECET
    - Medium dispatching priorities
    - EDF
    - Tries to do everything as soon as possible
- Level Three - MainParts executed beyond the reserved (and guaranteed) ECET
    - Lowest dispatching priorities
    - EDF
    - Tries to do everything as soon as possible

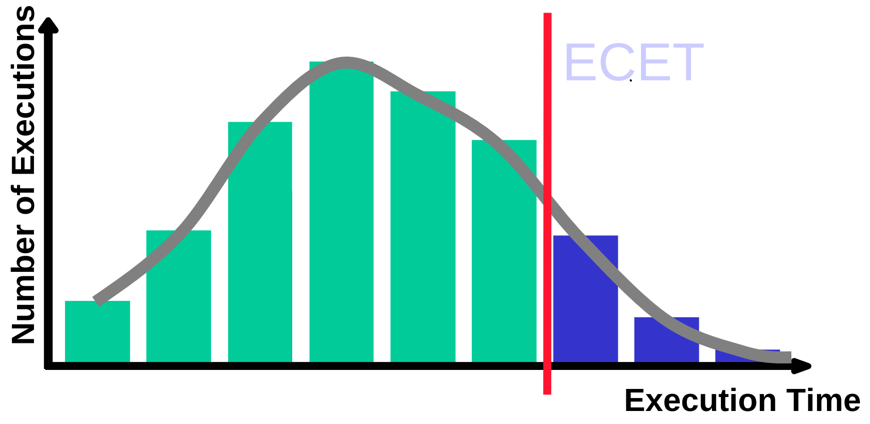
Real-Time (Paradigms) (37)

ECET$_{t,p}$ of task-instance *t* of task τ with probability *p*

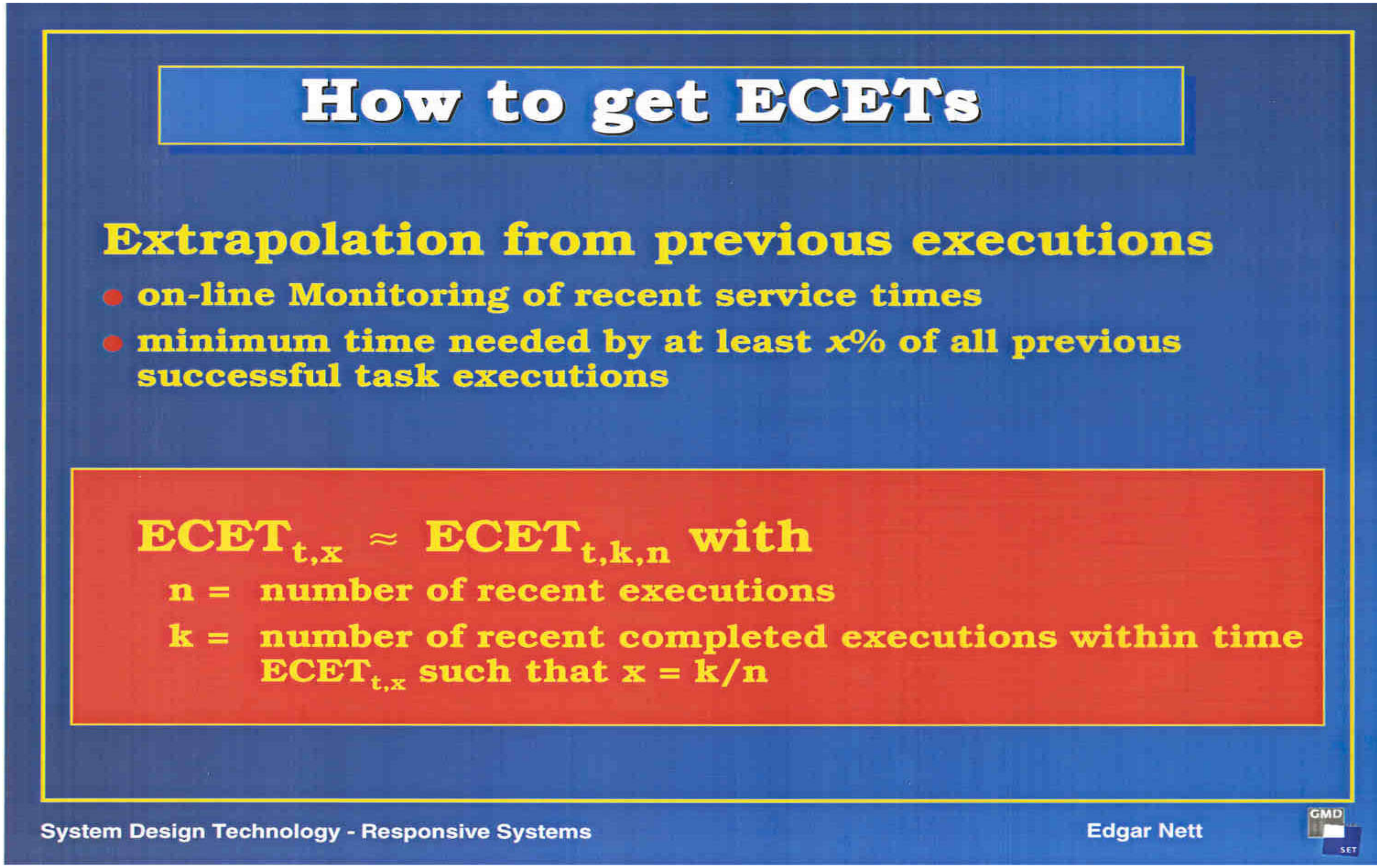
- CPU-time required to complete task-instance *t* with probability *p*
- *p*-quantile of the probabilistic density function of *T*'s execution time

ECET$_{t,k,n}$ - The minimal execution time that was needed to successfully complete at least *k* out of the last *n* most recent executions of τ before *t*.

- A statistic quantity



Real-Time (Paradigms) (38)



How to get ECETs

Extrapolation from previous executions

- on-line Monitoring of recent service times
- minimum time needed by at least *x*% of all previous successful task executions

$ECET_{t,x} \approx ECET_{t,k,n}$ with

n =	number of recent executions
k =	number of recent completed executions within time $ECET_{t,x}$ such that $x = k/n$

System Design Technology - Responsive Systems   Edgar Nett