NATIONAL ICT AUSTRALIA

THE UNIVERSITY OF NEW SOUTH WALES

Real-Time Systems

Stefan M. Petters

NICTA Ltd. and CSE UNSW

smp@cse.unsw.edu.au

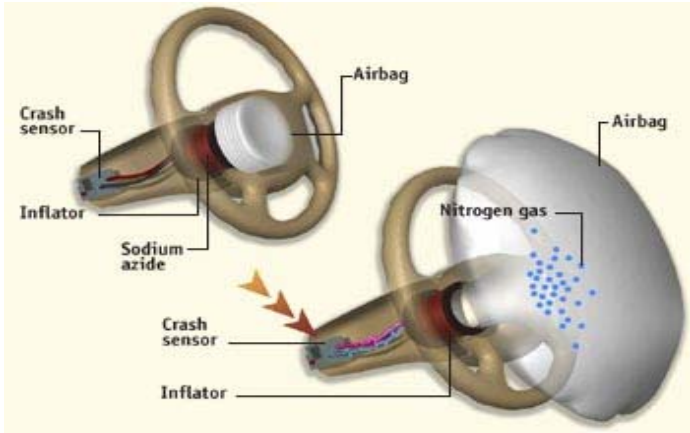
Lecture Content

- Definition of Real-Time Systems (RTS)
- Scheduling in RTS
- Schedulability Analysis
- Worst Case Execution Time Analysis
- Time and Distributed RTS

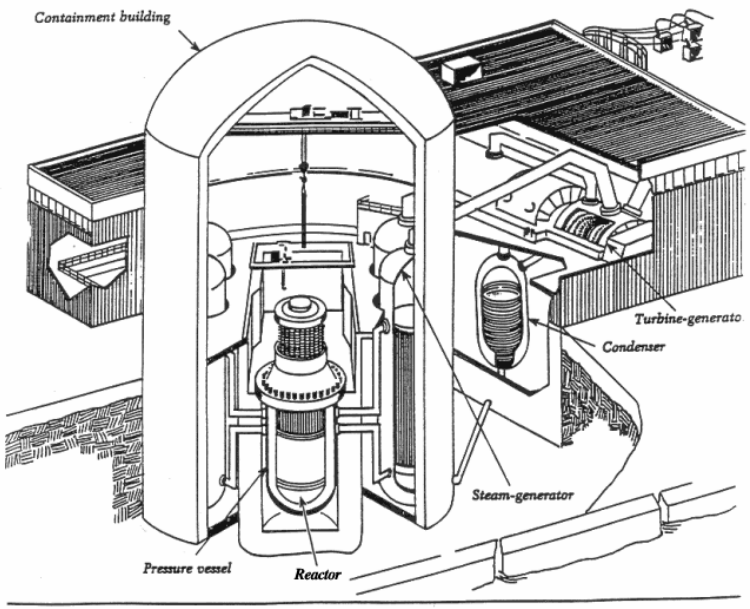
Definition

- A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period
  - the correctness depends not only on the logical result but also the time it was delivered
  - failure to respond is as bad as the wrong response!

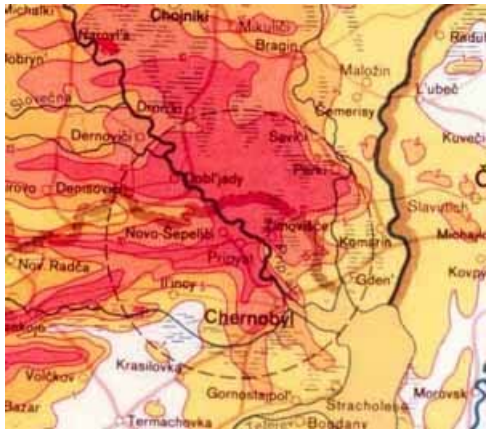
Real-Time Systems



Real-Time Systems

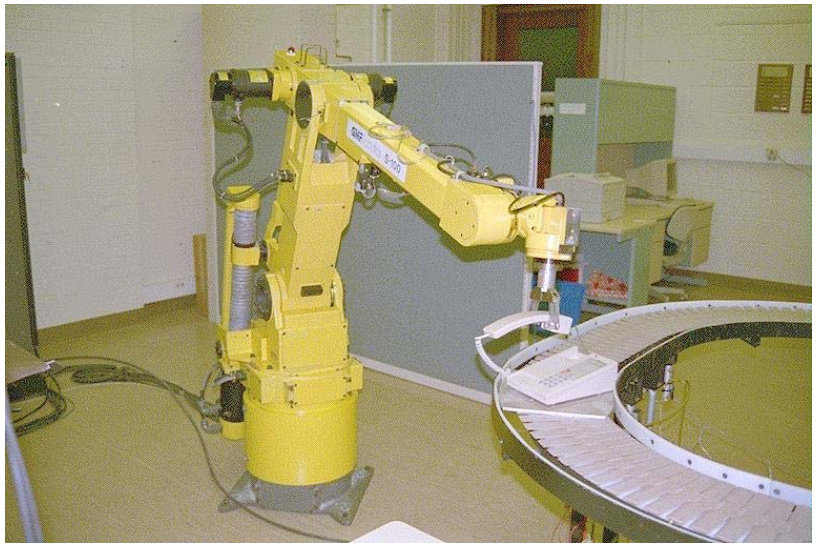


2 major parts of a nuclear powerplant work together to produce electricity.



Real-Time Systems

Real-Time Systems



Real-Time Systems

Is there a pattern?

- Hard real-time systems
- Soft real-time systems
- Firm teal-time systems
- Weakly hard real-time

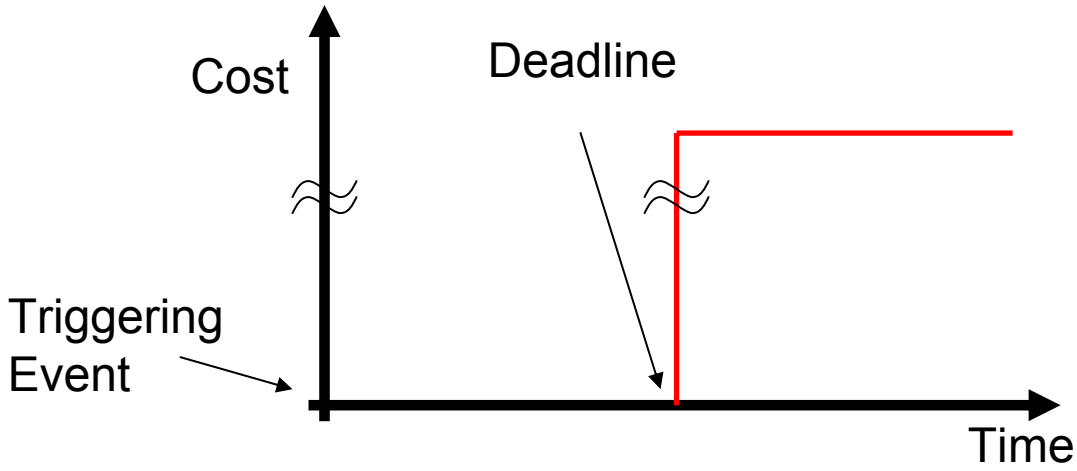
- A deadline is a given time after a triggering event, by which a response has to be completed.
- Therac 25 example

What's needed of an RTOS

- Fast context switches?
  - should be fast anyway
- Small size?
  - should be small anyway
- Quick response to external triggers?
  - not necessarily quick but predictable
- Multitasking?
  - often used, but not necessarily
- "Low Level" programming interfaces?
  - might be needed as with other embedded systems
- High processor utilisation?
  - desirable in any system (avoid oversized system)

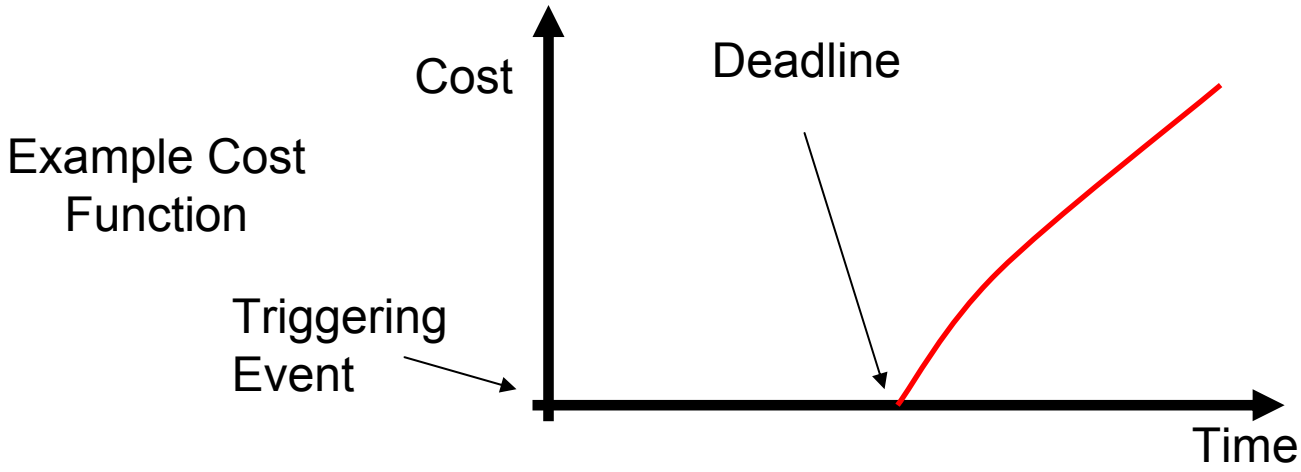
Hard Real-Time Systems

- An overrun in response time leads to potential loss of life and/or big financial damage
- Many of these systems are considered to be safety critical.
- Sometimes they are “only” mission critical, with the mission being very expensive.
- In general there is a cost function associated with the system.



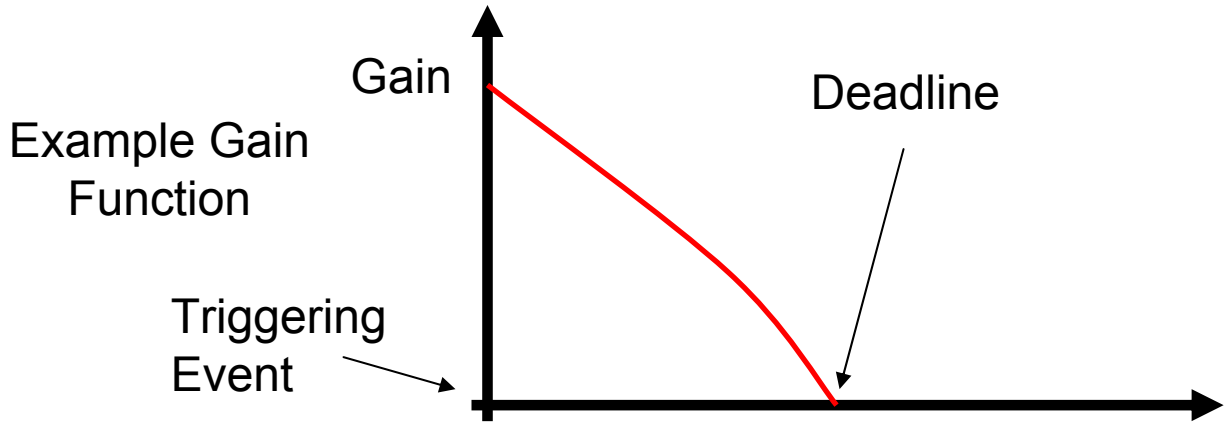
Soft Real-Time

- Deadline overruns are tolerable, but not desired.
- There are no catastrophic consequences of missing one or more deadlines.
- There is a cost associated to overrunning, but this cost may be abstract.
- Often connected to Quality-of-Service (QoS)



Firm Real-Time Systems

- The computation is obsolete if the job is not finished on time.
- Cost may be interpreted as loss of revenue.
- Typical example are forecast systems.



Weakly Hard Real-Time Systems

- Systems where *m* out of *k* deadlines have to be met.
- In most cases feedback control systems, in which the control becomes unstable with too many missed control cycles.
- Best suited if system has to deal with other failures as well (e.g. Electro Magnetic Interference EMI).
- Likely probabilistic guarantees sufficient.

Non Real-Time Systems?

- Yes, those exist!
- However, in most cases the (soft) real-time aspect may be constructed (e.g. acceptable response time to user input).
- Computer system is backed up by hardware (e.g. end position switches)
- Quite often simply oversized computers.

Requirement, Specification, Verification

- **Functional requirements :** Operation of the system and their effects.
- **Non-Functional requirements :** e.g., timing constraints.
  - F & NF requirements must be precisely defined and together used to construct the specification of the system.
- A **specification** is a mathematical statement of the properties to be exhibited by a system. It is abstracted such that
  - it can be checked for conformity against the requirement.
  - its properties can be examined independently of the way in which it will be implemented.
- The usual approaches for specifying computing system behavior entail enumerating events or actions that the system participates in and describing orders in which they can occur. It is not well understood how to extend such approaches for real-time constraints.
- F18 example

Scheduling in Real-Time Systems

Overview

- Specification and religious believes
- Preemptive vs. non preemptive scheduling
- Scheduling algorithms
- Message based synchronisation and communication
- Issues of mixed RT and non RT applications
- Overload situations
- Semaphores and blocking

Requirements

- Temporal requirements of the embedded system
  - Event driven
    - Reactive sensor/actuator systems
    - No fixed temporal relation between events (apart from minimum inter arrival times)
  - Cyclic
    - Feedback control type applications
    - Fixed cycles of external triggers with minimal jitter
  - Mixed
    - Anything in between

Specification

- Event triggered systems:
  - Passage of a certain amount of time
  - Asynchronous events
- Time triggered systems:
  - Predefined temporal relation of events
  - Events may be ignored until it's their turn to be served
- Matlab/Simulink type multi rate, single base rate systems:
  - All rates are multiples of the base rate
- Cyclic
  - feedback control loop

Task Model

- Periodic tasks
  - Time-driven. Characteristics are known a priori
  - Task τ_i is characterized by (T_i, C_i)
  - E.g.: Task monitoring temperature of a patient in an ICU.
- Aperiodic tasks
  - Event-driven. Characteristics are not known a priori
  - Task τ_i is characterized by (C_i, D_i) and some probabilistic profile for arrival patterns (e.g. Poisson model)
  - E.g.: Task activated upon detecting change in patient's condition.
- Sporadic Tasks
  - Aperiodic tasks with known minimum inter-arrival time (T_i, C_i)

Task Model

C_i = Computation time (usually Worst-Case Execution Time, WCET)

D_i = Deadline

T_i = Period or minimum interarrival time

J_i = Release jitter

P_i = Priority

B_i = Worst case blocking time

R_i = Worst case response time

Task Constraints

- Deadline constraint
- Resource constraints
  - Shared access (read-read), Exclusive access (write-x)
  - Energy
- Precedence constraints
  - $\tau_1 \Rightarrow \tau_2$: Task τ_2 can start executing only after τ_1 finishes its execution
- Fault-tolerant requirements
  - To achieve higher reliability for task execution
  - Redundancy in execution

Preemption

- Why preemptive scheduling is good:
  - It allows for shorter response time of high priority tasks
  - As a result it is likely to allow for a higher utilisation of the processor before the system starts missing deadlines
- Why preemptive scheduling is bad:
  - It leads to more task switches then necessary
  - The overheads of task switches are non-trivial
  - The system becomes harder to analyse whether it is able to meet all its deadlines
  - Preemption delay (cache refill etc.) becomes more expensive with modern processors
- Cooperative preemption?
  - Applications allow preemption at given points
  - Reduction of preemptions
  - Increase of latency for high priority tasks

Event Triggered Systems

"... The asynchronous design of the [AFTI-F16] DFCS introduced a random, unpredictable characteristic into the system. The system became untestable in that testing for each of the possible time relationships between the computers was impossible. This random time relationship was a major contributor to the flight test anomalies. Adversely affecting testability and having only postulated benefits, asynchronous operation of the DFCS demonstrated the need to avoid random, unpredictable, and uncompensated design characteristics."

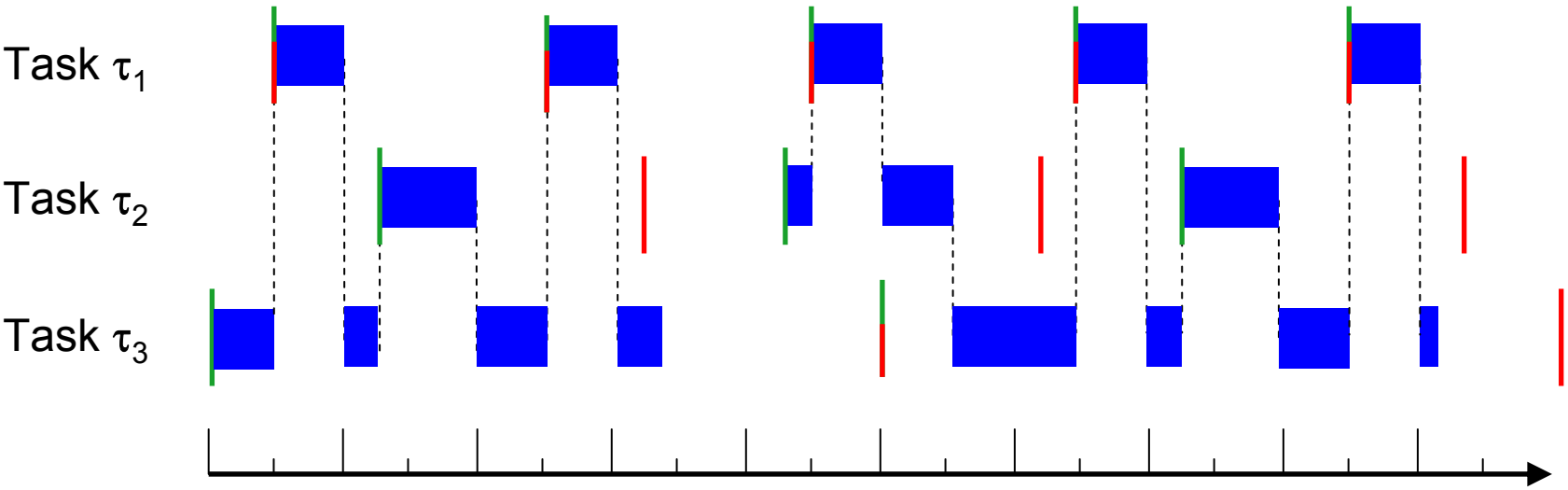
D. Mackall, flight-test engineer AFTI-F16 flight tests

Nasa Example: DEOS

Fixed Priority Scheduling

- Priorities may assigned by
  - Deadline: shortest deadline $\Rightarrow$ highest priority
  - Period: shortest period $\Rightarrow$ highest priority
  - "Importance"
- Scheduler picks from all ready tasks the one with the highest priority to be dispatched.
- Benefits:
  - Simple to implement
  - Not much overhead
  - Minimal latency for high priority tasks
- Drawbacks
  - Inflexible
  - Suboptimal (from analysis point of view)

Fixed Priority Scheduling(FPS)

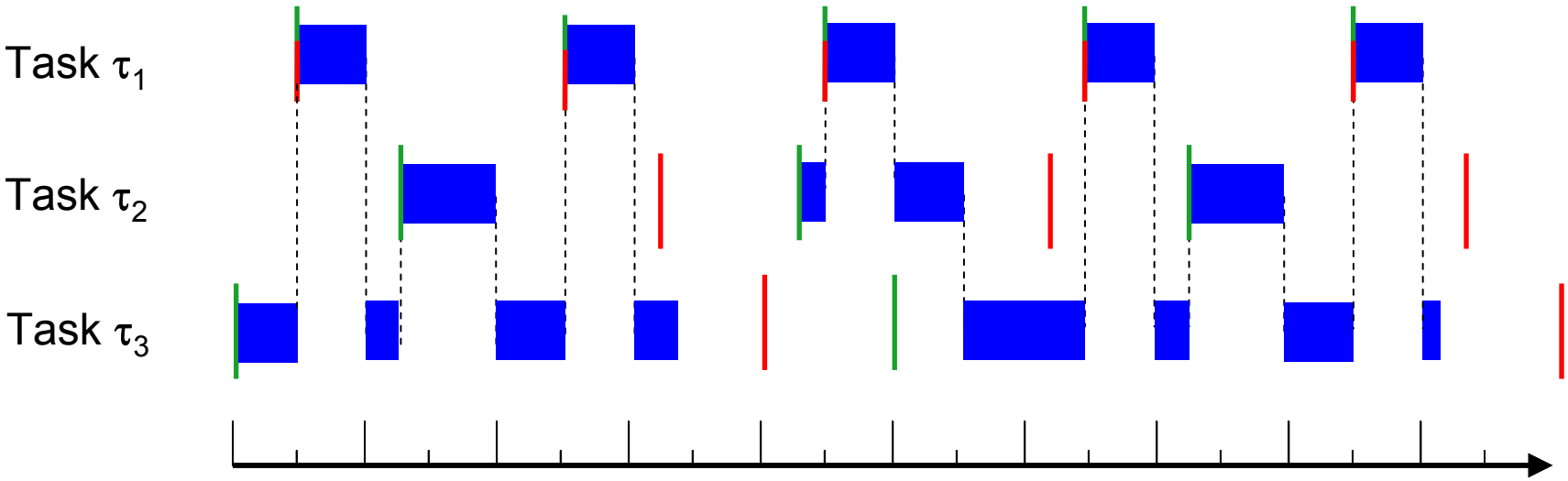


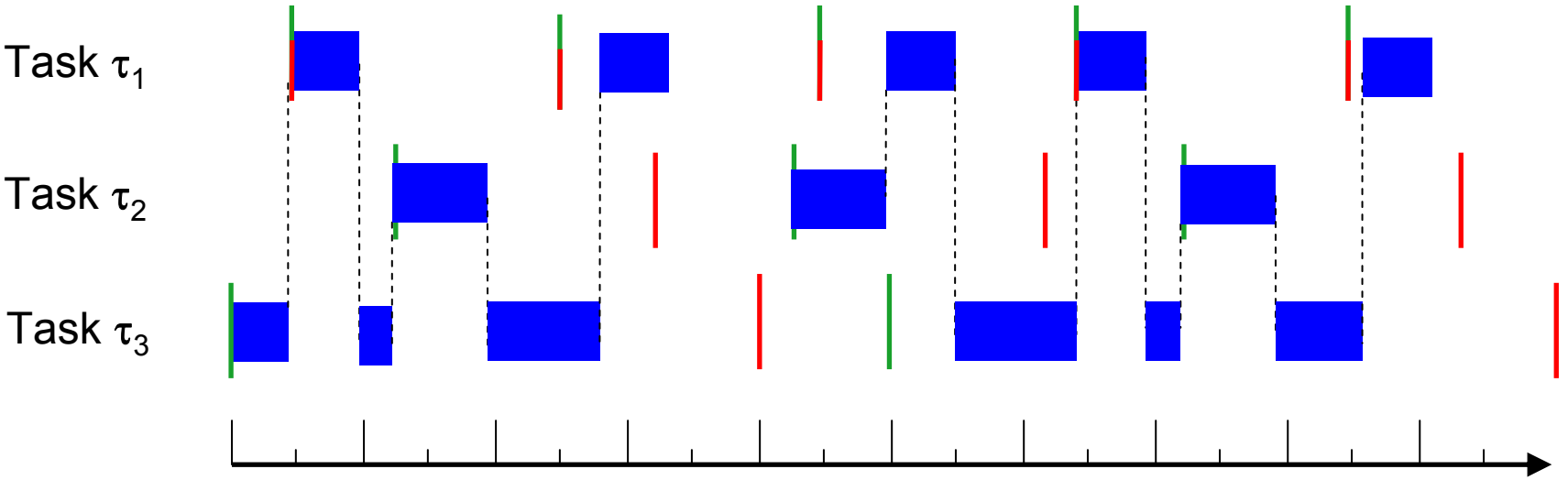
	Priority	C	T	D
Task τ_1	1	5	20	20
Task τ_2	2	8	30	20
Task τ_3	3	15	50	50

Earliest Deadline First (EDF)

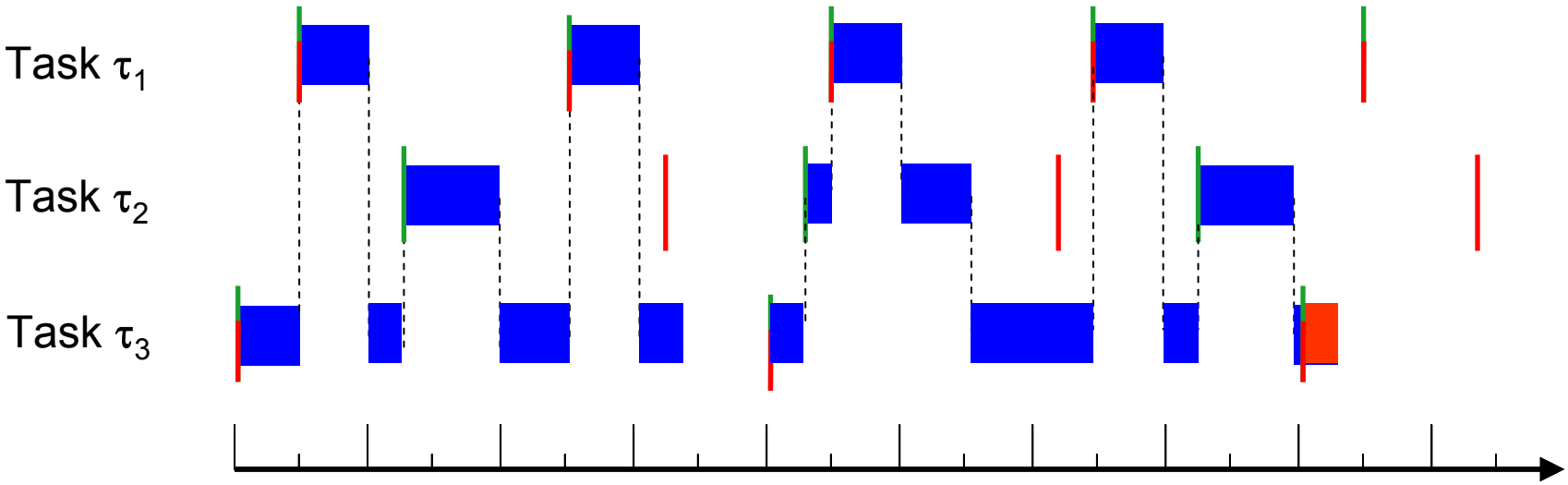
- Dynamic priorities
- Scheduler picks task, whose deadline is due next
- Advantages:
  - Optimality
  - Reduces number of task switches
  - Optimal if system is not overloaded
- Drawbacks:
  - Deteriorates badly under overload
  - Needs smarter scheduler
  - Scheduling is more expensive

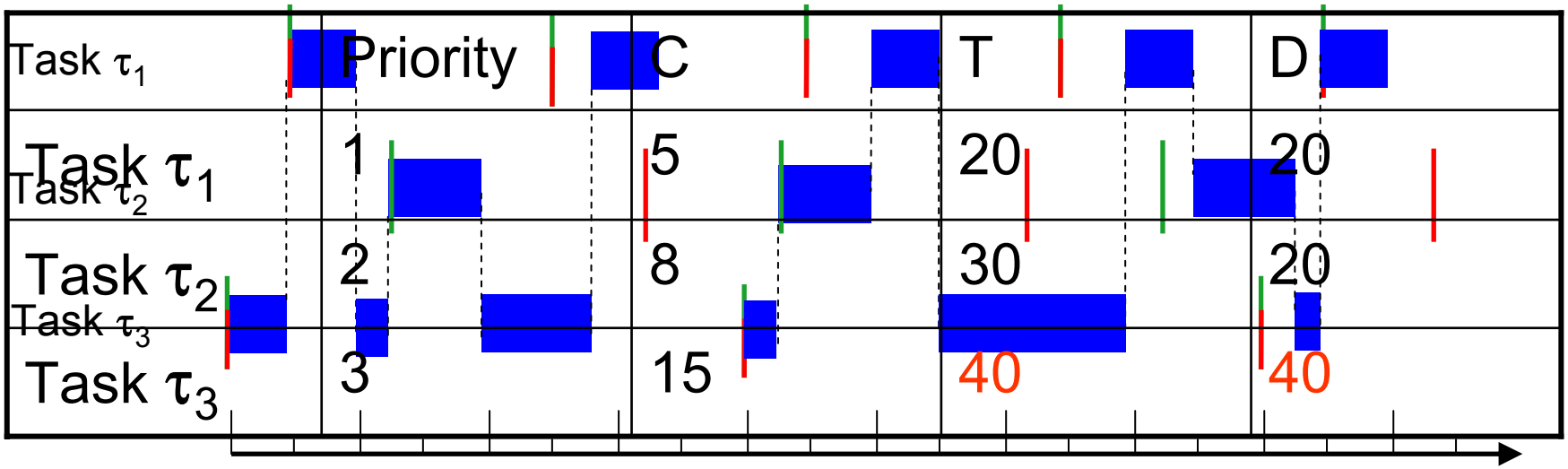
FPS vs. EDF





FPS vs. EDF



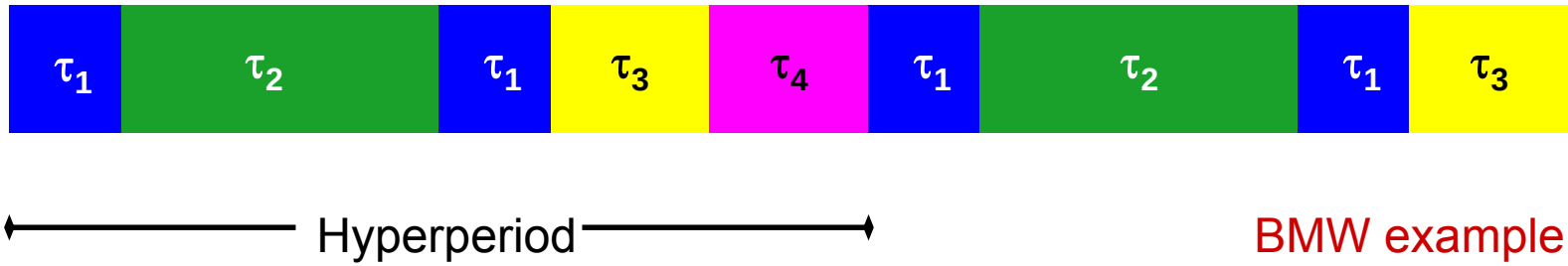


Time Triggered/Driven Scheduling

- Mostly static scheduling
- Time triggered scheduling allows easier reasoning and monitoring of response times
- Can be used to avoid preemption
- Can be used in event triggered systems, but increases greatly the latency
- Most often build around a base rate
- Can be implemented in big executive, using simple function calls

Time Triggered Scheduling

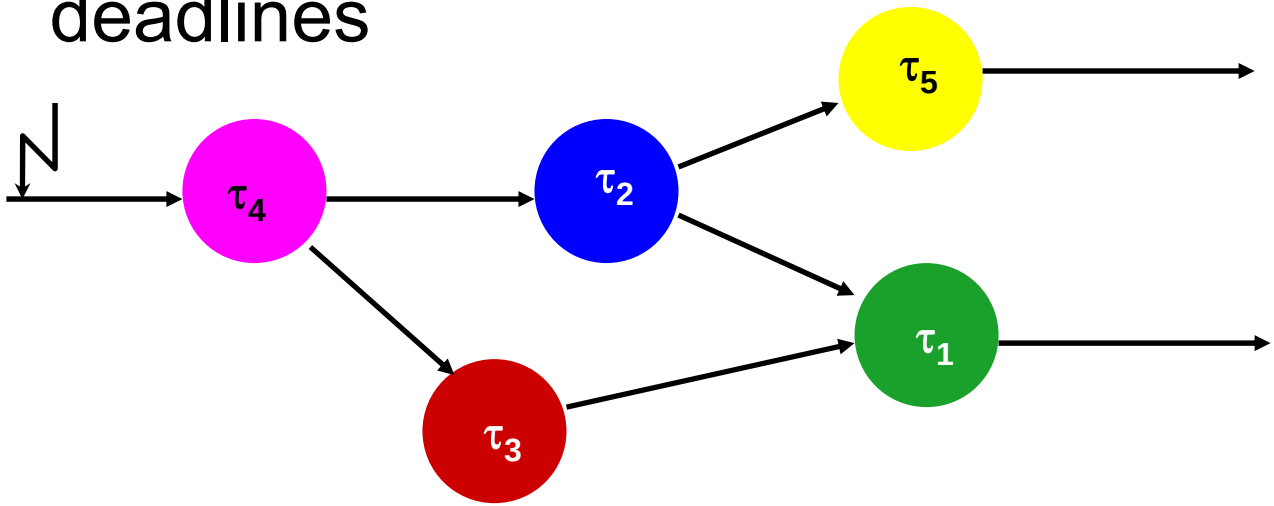
- Advantages:
  - Very simple to implement
  - Very efficient / little overhead (in suitable case)
- Disadvantages:
  - Big latency if event rate does not match base rate
  - Inflexible
  - Potentially big base rate (many scheduling decisions) or hyperperiod



BMW example

Message Based Synchronisation

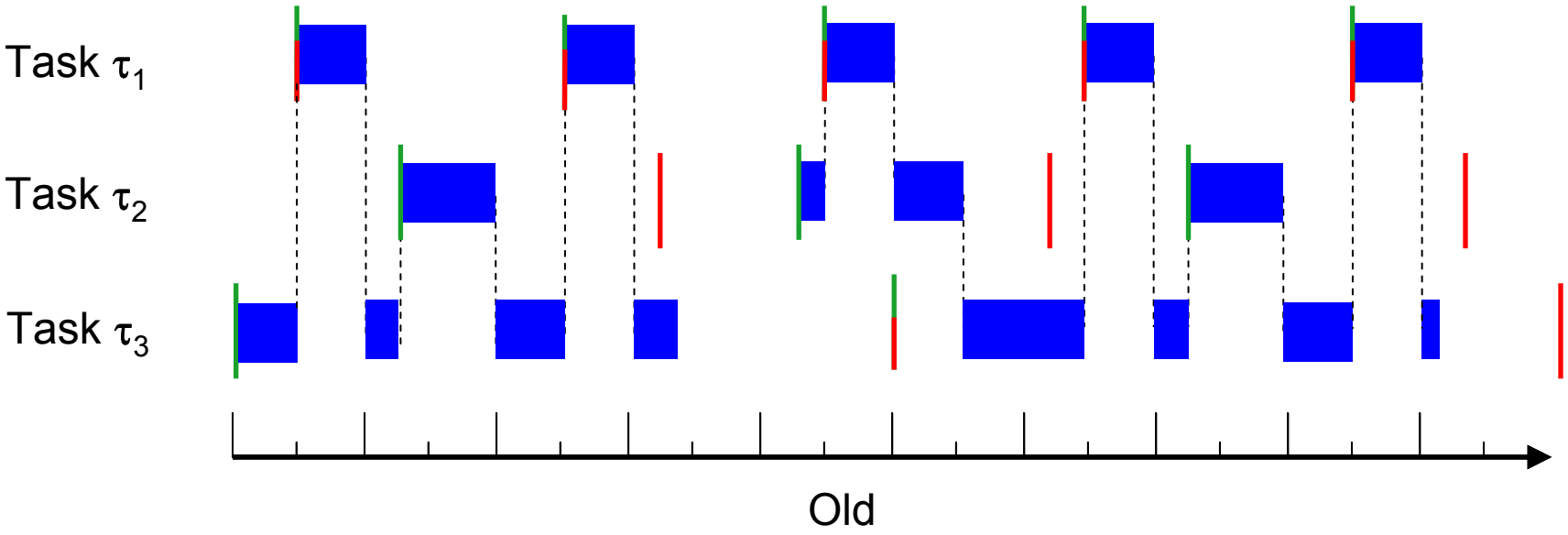
- Tasks communicate via messages
- Task wait for messages (blocked until message arrives)
- Suitable to enforce precedence relations
- Enables messages to be used to transport deadlines



Overload Situations

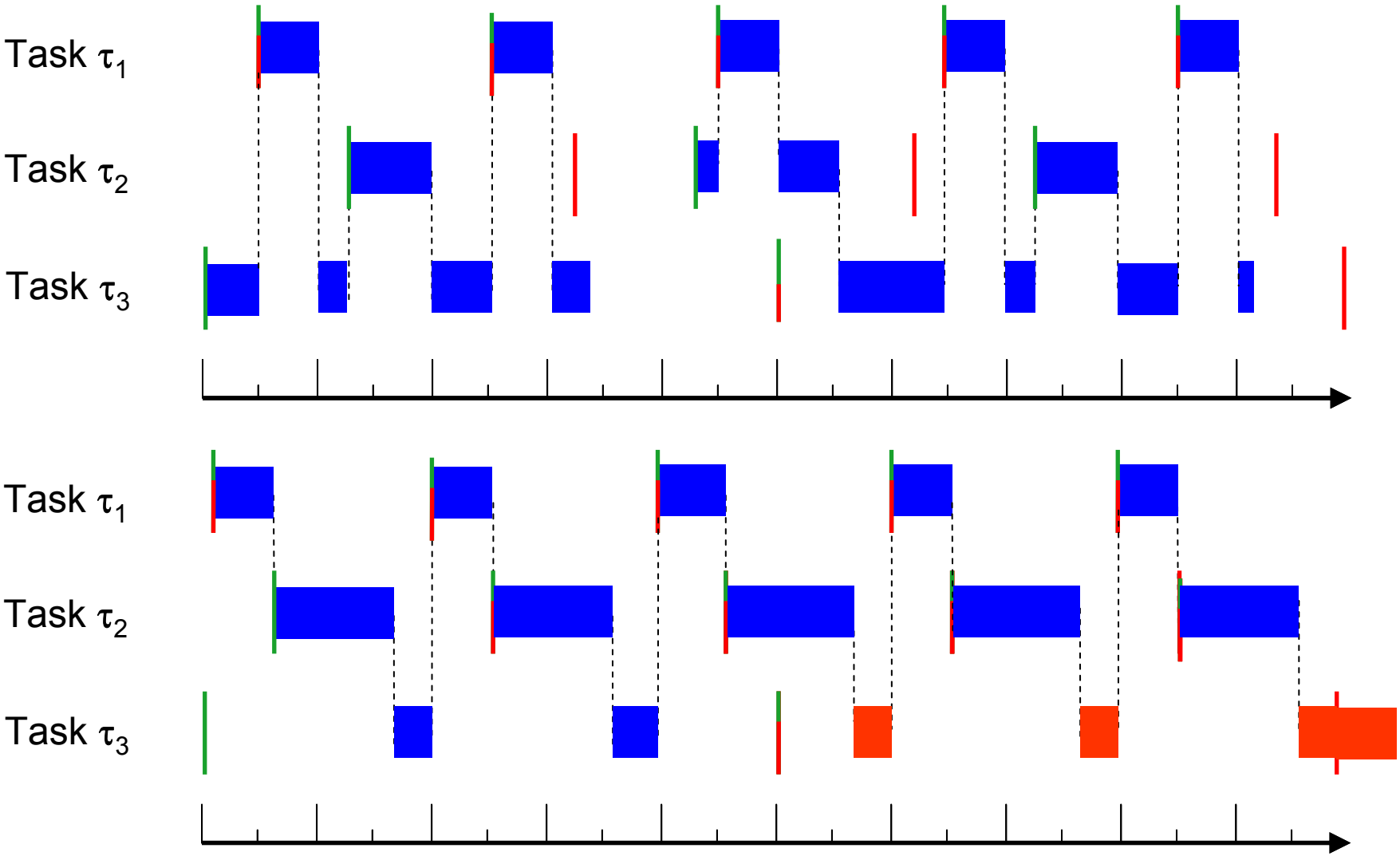
- Caused by faulty components of the system
  - Babbeling idiot or
  - A receiver part erroneously “receiving input”
  - EMI
- Or caused by wrong assumptions regarding the embedding environment
  - Basically wrong event rates or event correlation

Overload Situations in FPS

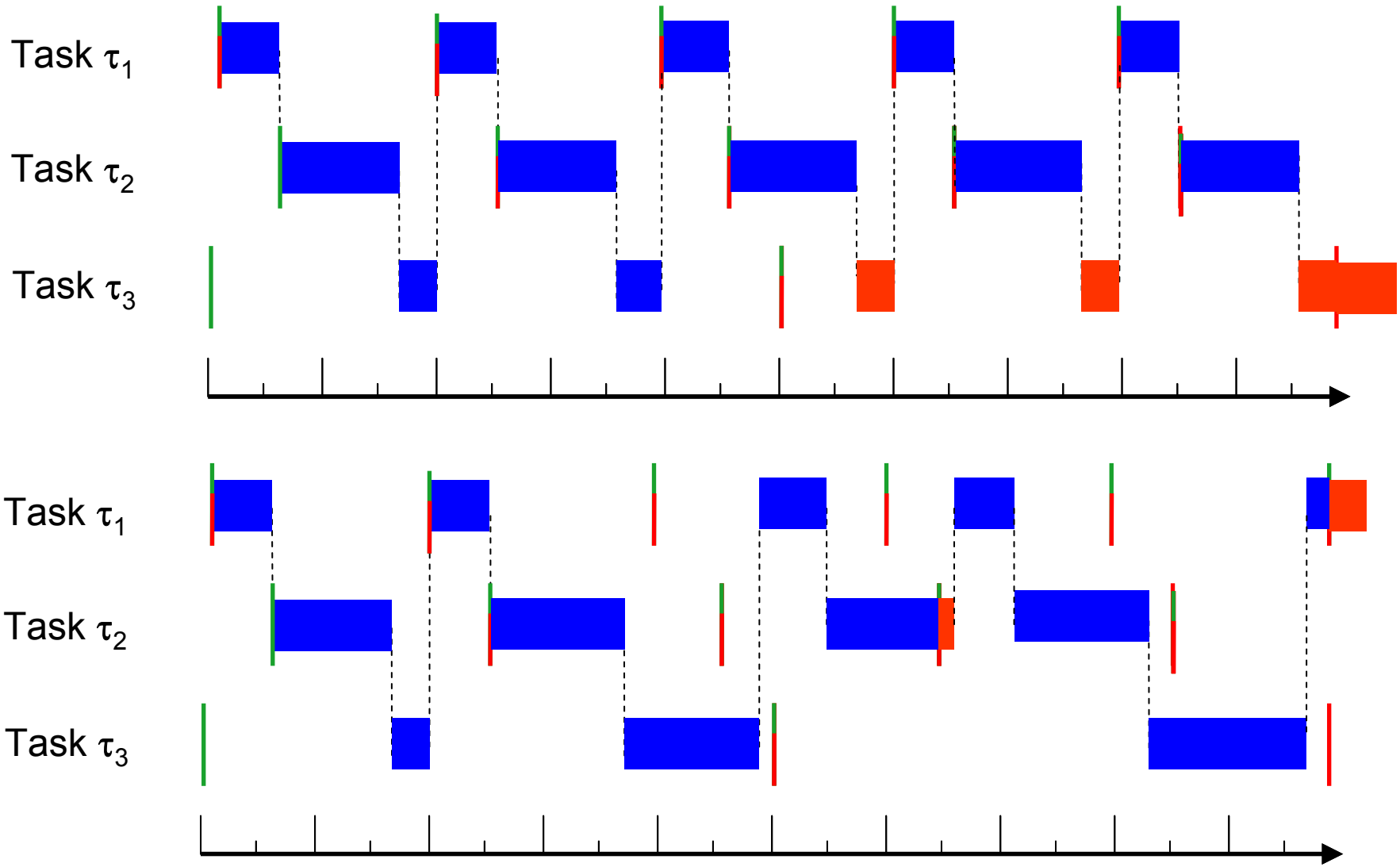


	Priority	C	T	D
Task τ_1	1	5	20	20
Task τ_2	2	12	20	20
Task τ_3	3	15	50	50

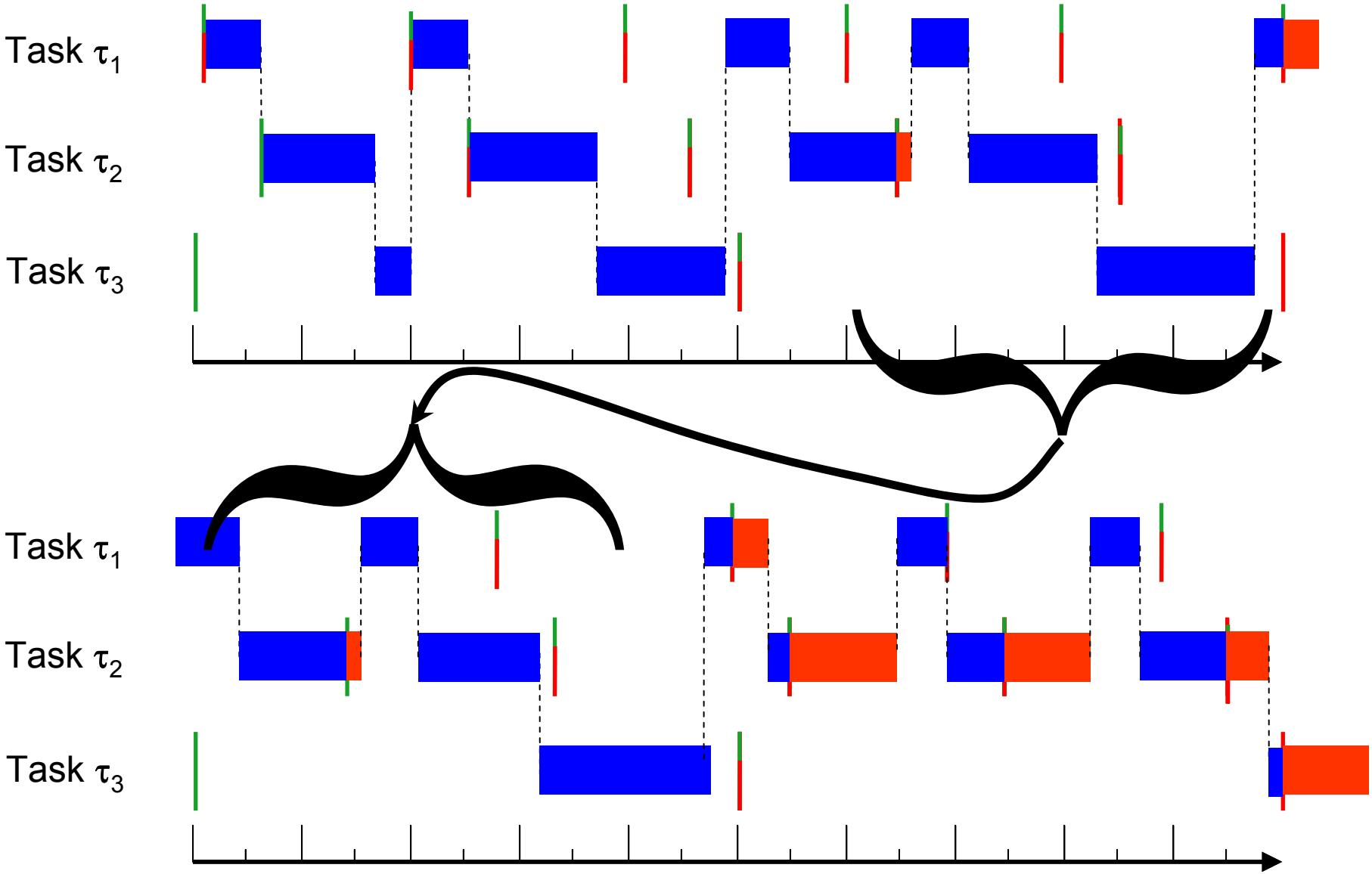
Overload Situations in FPS



Overload Situations in EDF



Overload Situations in EDF



Priority Inversion

- Happens when task is blocked in acquiring semaphore from held by lower priority task which is preempted by medium priority task.
- Similar case for server tasks.
- Cure
  - Priority Ceiling or
  - Priority inheritance
- Pathfinder example

Schedulability Analysis of Real-Time Systems

Schedulability Analysis

- Tries to establish, whether the task system described is actually schedulable
  - In the classical sense this is, whether all the deadlines are met under all circumstances;
  - Recent move to satisfaction of Quality-of-Service constraints;
- Relies on availability of computation time of tasks
  - WCET;
  - Execution time profiles.

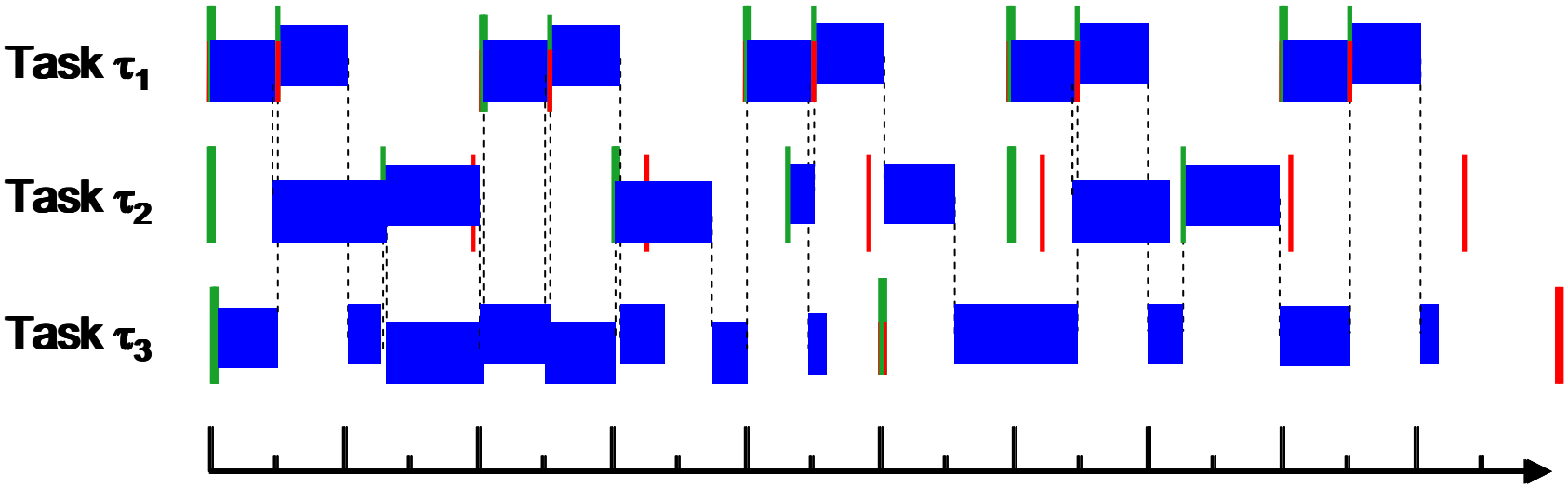
Critical Instant

- Trivial for independent tasks
  - All events happen at the same time;
  - However, implicitly consider all possible phases (take nothing for granted).
- However, get's more tricky (but tighter) having dependencies
  - What phasing of other activities produces the biggest load.
  - An activity is a string of tasks triggered by a single event.

Response Time Analysis

- Does not directly consider deadlines
- Makes the assumption of jobs being executed in order
- Usually used in fixed priority systems

	Priority	C	T	D
Task τ_1	1	5	20	20
Task τ_2	2	8	30	20
Task τ_3	3	15	50	50



Formal RTA

- Assumptions j<i $\Rightarrow$priority j is lower than priority i
- Critical instant
- Iterative process

$$w_i^0 = C_i$$

$$w_i^{n+1} = C_i + \sum_{\forall j<i} \left\lceil \frac{w_i^n}{T_j} \right\rceil * C_j$$

Blocking Time and Other Nasties

- Blocking time
- Jitter
- Pre-emption delay

$$w_i^{n+1} = C_i + B_i + \sum_{\forall j<i} \left\lceil \frac{J_j + w_i^n}{T_j} \right\rceil * (C_j + \delta_{i,j})$$

Rate Monotonic Analysis

- Looks at *utilisation* do determine whether a task is schedulable
- Initial work had following requirements:
  - All tasks with deadlines are periodic
  - All tasks are independent of each other (there exists no precedence relation, nor mutual exclusion)
  - $T_i = D_i$
  - C_i is known and constant
  - Time required for context switching is known
- Bound is given by:

$$\mu = \sum_{\forall i} \left\lceil \frac{C_i}{T_i} \right\rceil \leq n * (2^{1/n} - 1)$$

- Has been relaxed in various ways, but still it is only an approximate technique.
- Further info can be found here:
  http://www.heidmann.com/paul/rma/PAPER.htm

Worst Case Execution Time Analysis

WCET Analysis in General

- Aims to provide upper bounds on the execution time of tasks/threads/syscalls
- WCET's are input parameters to schedulability analysis
- Can be extended to provide execution time profiles or parametric WCET instead of single number.
- Can be done
  - Statically i.e. without running the program, analysing SW and HW
  - Dynamically by using measurements

Common Problems

- Structural analysis of the program
  - Basic blocks:
    - Single entry point
    - Single exit point
    - No alternative paths (sequential piece of code)
  - Control flow graph
    - Loop detection
    - Alternatives detection
- Computation
  - Path based
  - Tree based
  - Implicit path enumeration based (IPET)

Static Analysis

- Looking at basic blocks in isolation (tree based, IPET based)
  - Problem of caching effects
- Path based analysis popular but very expensive
- Problem of conservative assumptions
- Hardware analysis is very expensive
  - Data caches and modern branch prediction are very hard to model right.
  - Call for simpler hardware, e.g. scratchpad memory instead of caches

Measurement Based Analysis

- End-to-end measurements + safety factor used for industrial soft-real time system development
  - Failed for modern processors as WC could hardly be expressed as function of AC
- Measurement on basic block level
  - Safer than end-to-end measurements but potentially very pessimistic
- What is the worst-case on HW?
- Can it be reliably produced?
- What about preemption delay?

Distributed Real-Time Systems

Distributed Time?

Fundamentally, it is impossible to have a perfect, common time base

- So, we hope relativistic effects don't matter

- We put in hacks for network delay time
  - Measure typical propagation delays
  - Measure typical time variations (drift, jitter)
  - Assume that they don't change a lot, and add in fudge factors to account for them

- But, ***EVEN THEN***, "closely spaced" events are always a problem
- Partiot example

Ordering of Events

- Causal order
  - How did it happen?
  - Explains how several chain of events contributed to a certain situation to surface.
- Temporal order
  - When did things happen?
  - Describes the sequence of events.
  - Assumes observer with zero latency access to all events.
- Message arrival order
  - When did we learn of it?
  - Not necessarily in temporal order.
  - Can we relate when it actually happened?
  - Delays can be of various sources: e.g. conversions, priority blocking, multi hop routing
  - Obvious solution: Time Stamping

Time Measurement Inaccuracies

- **Quantization effects**
  - Micro-tick size limitation on a single node
  - Across-network Tick size limitation on a system

- **Variations**
  - Synchronization difference (impossible to sync all clocks *exactly*)
  - Clock drift (too fast, too slow, maybe time-varying)
  - *Maybe*, round-off error on time keeping
    - Gulf War/Patriot missile system story… floating point roundoff, not really clock drift, but similar issue

- **Measuring time errors**
  - **Offset**: difference in time at a particular micro-tick per an omniscient observer
  - **Precision**: (Π) maximum offset between any two clocks within system
  - **Accuracy**: (***A***) offset between system time and the "real" time

Clock Synchronisation

- **Every once in a while, clocks must be reset to the "correct" time**
  - Consensus among nodes (improving precision)
  - Consensus with notional reference clock (improving accuracy)
- **State correction**
  - Agree on the time and fast-forward/rewind to that time
  - Simple, but introduces discontinuities in time base
- **Rate correction**
  - Speed up/slow down tick rate to converge to better time
  - More difficult to implement, less chance of a problem
  - GPS time is "rate steered"
    - GPS time is typically accurate within 200 ns to 1 microsecond

What is out there?

- Time keeping is an essential problem of distributed RT systems
- Hence most distributed RT has special consideration for time keeping
- Example TTP/C where time is part of the state information of the system, which is checked to be consistent across the nodes

Books and other Info

Burns, Alan & Wellings, Andrew: Real-Time Systems and Programming Languages (3rd ed), Addison Wesley, 2001

Kopetz, Hermann: Real-time Systems : Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997

Joseph, Mathai: Real-time Systems: Specification, Verification and Analysis, 2001

http://www.tcs.com/techbytes/htdocs/RTSbook.zip

Dale A. Mackall: *Development and flight test experiences with a flight-crucial digital control system*. NASA Technical Paper 2857, NASA Ames Research Center, Dryden Flight Research Facility, Edwards, CA, 1988.