Uniprocessor Real-Time Scheduling

Robert Davis

Real-Time Systems Research Group, University of York

rob.davis@york.ac.uk
http://www-users.cs.york.ac.uk/~robdavis/

Overview

- Background
  - What is real-time?, Task models, Scheduling policies and Analysis
- Fixed Priority Scheduling
  - Fundamentals, Resource Sharing, Response Time Analysis, Extensions: Arbitrary Deadlines, Non-pre-emptive scheduling
  - Modelling RTOS behaviour and overheads
  - Priority Assignment
- EDF Scheduling
  - Fundamentals, Resource Sharing, Processor Demand Analysis, QPA,
  - EDF v FP in theory and practice
- Building on the fundamentals
  - Limited Pre-emption, Cache Related Pre-emption Delays
- Wrap up
  - Success stories, hot topics, open problems

What is a Real-Time System?

Real-Time System is any system which has to respond to externally generated input stimuli within a specified time

- Functionally correct – the right computations
- Temporally correct – completed within predefined time constraints
- Time constraints typically expressed in terms of deadlines on the elapsed time between the stimuli and the corresponding response

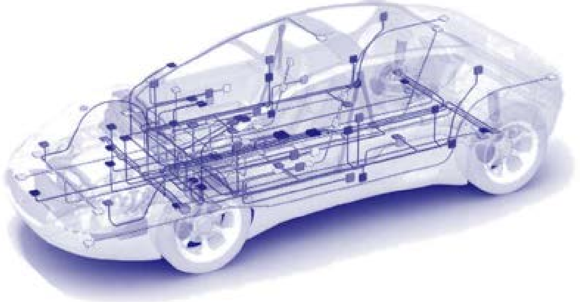
- Hard Real-Time
  - Failure to meet a deadline constitutes a failure of the application (e.g. flight control system)
- Soft Real-Time
  - Latency in excess of the deadline leads to degraded quality of service (e.g. data acquisition, video playback)

Examples of Real-Time Systems

Robotics and Factory Automation

Instrumentation

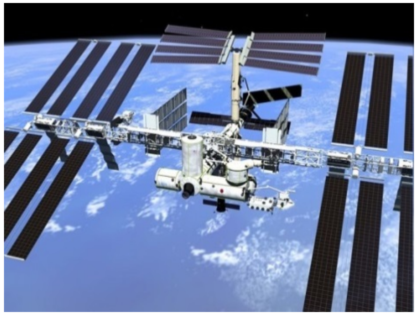
Avionics



Automotive Electronics

Telecommunications

Medical Systems



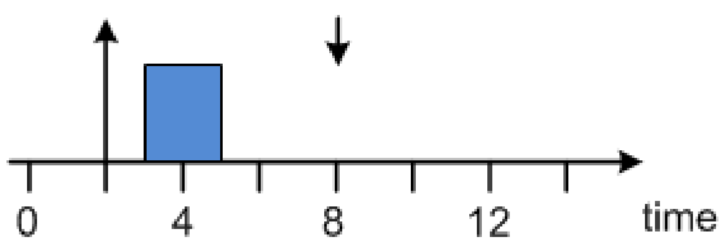
Space

Real-Time Applications

- Time-triggered
  - Monitoring and data acquisition
  - Control loops
  - Typically periodic behaviours e.g. every 20ms
- Event-triggered (interrupt-driven)
  - Simple sensors (switch closes)
  - Engine rotation
  - Peripheral devices (e.g. comms – message received)
  - Often generate non-periodic behaviours

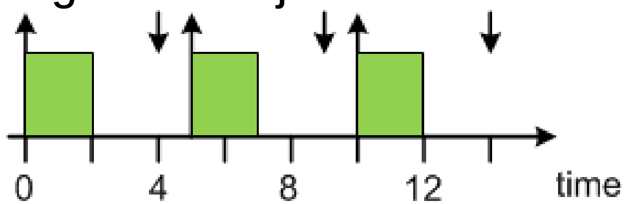
Real-Time Applications

- Applications de-composed into tasks
  - Tasks implement the functionality of the system
- Tasking model
  - Static set of n tasks τ_i (i = 1 to n)
  - Each task gives rise to a potentially infinite sequence of jobs
  - Job of task τ_i
    - Arrives at some time t and is released (ready to execute)
    - Execute for a time no greater than Worst-Case Execution Time (WCET) C_i
    - Before an Absolute Deadline which is a Relative Deadline D_i after its arrival

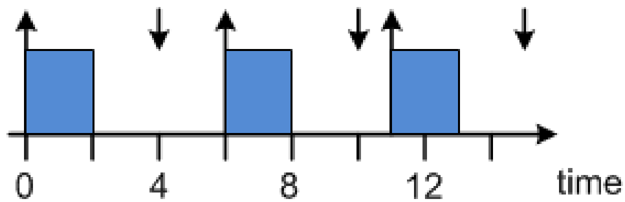


Task Timing Behaviour

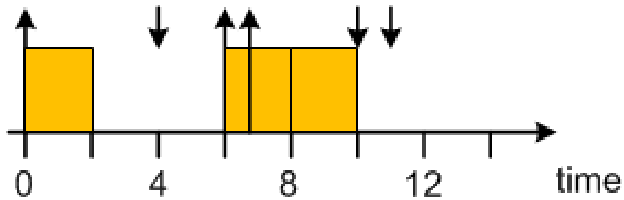
- Types of task (time-triggered and event-triggered) based on pattern of arrivals
  - Periodic: generates jobs with a strict period of T_i between them



  - Sporadic: minimum inter-arrival time T_i between jobs

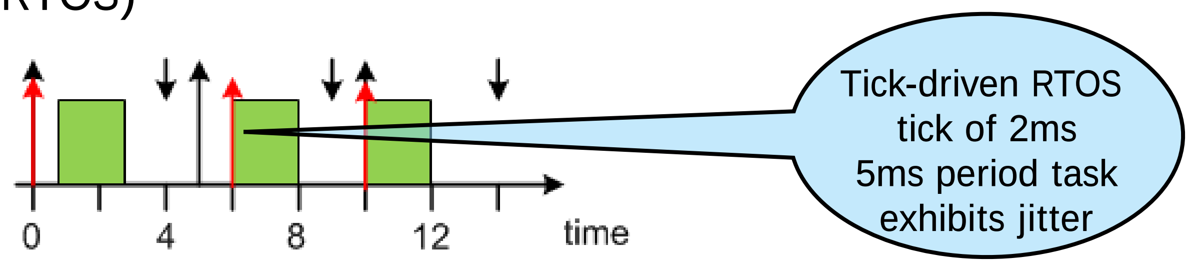


  - Aperiodic: no minimum inter-arrival time, so jobs can arrive arbitrarily close together



Task Timing Behaviour

- Execution times
  - Assume a bounded WCET C_i for Hard Real-Time task τ_i
- Types of Deadline
  - Implicit: Same as period / minimum inter-arrival time $D_i = T_i$ between them
  - Constrained: $D_i \leq T_i$
  - Arbitrary: not related to T_i (but needs to be $\geq C_i$)
- Release jitter
  - Job may arrive but may not be released ready for processing immediately
  - Variability in time from arrival to release is release jitter J_i (e.g. tick-driven RTOS)



Task Timing Behaviour

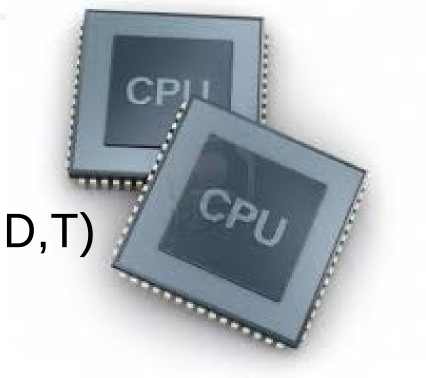
- Shared Resources
  - Jobs may need mutually exclusive access to shared resources (e.g. shared data structures, on-chip peripherals e.g. communications)

- Non-pre-emptable behaviour
  - Jobs may have critical sections with interrupts disabled
  - RTOS API calls may internally disable/enable interrupts or scheduling
  - Jobs may need to execute non-pre-emptably to reduce output jitter (from reading a sensor value to outputting a response to an actuator)

THE UNIVERSITY of York

RTS York

Uniprocessor Real-Time Scheduling

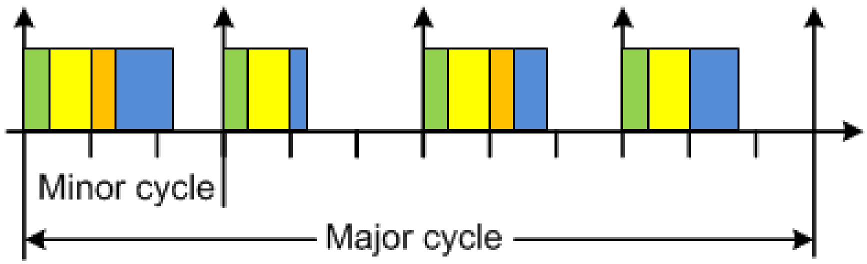


- Why do we need scheduling at all?
  - Single processor can only execute one job at a time
  - Tasks can have very different timing characteristics (C,D,T)
  - Multiple tasks and each task can potentially generate an infinite sequence of jobs

- Terminology
  - Scheduler: part of a RTOS which decides at run-time which job to execute
  - Scheduling policy: rules used by the scheduler to choose between jobs
  - Schedulability analysis: some maths used offline to determine if jobs can always be guaranteed to meet their deadlines according to a system and task model
    - To be useful need to upper bound OS and other overheads

Scheduling policies: Offline

- Static Cyclic Scheduling:
  - Table driven or Cyclic Executive – no scheduler as such, just a hard-coded cycle of procedures to execute

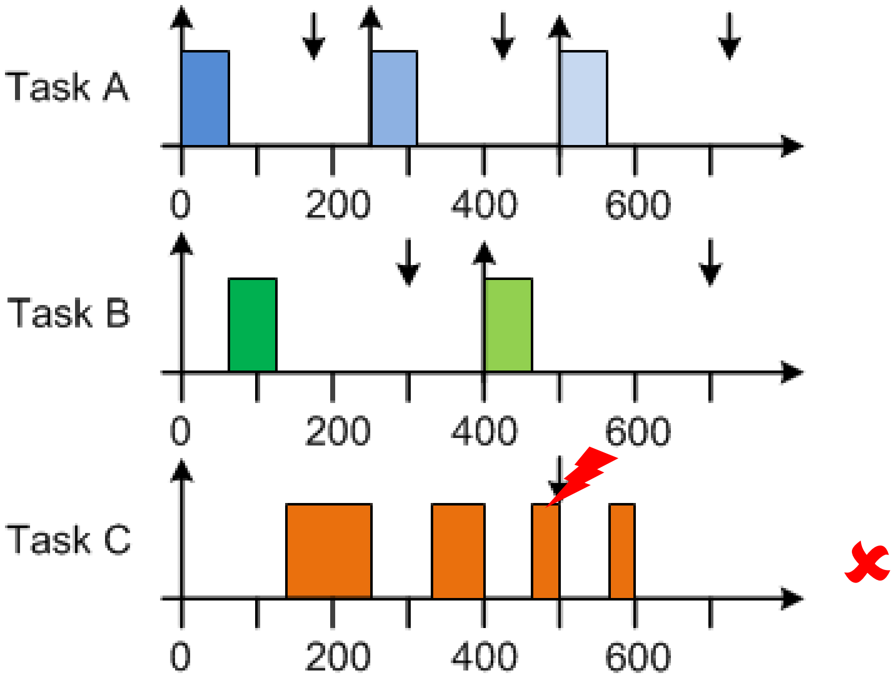


- Advantages:
  - Apparently low overhead (simple cyclic code), Deterministic
- Disadvantages:
  - Hard to support sporadic behaviour (need to reserve time in each cycle) for what may be one off jobs
  - Need to split large jobs (long C,T) into fragments
  - Hard to maintain
  - Lacks flexibility

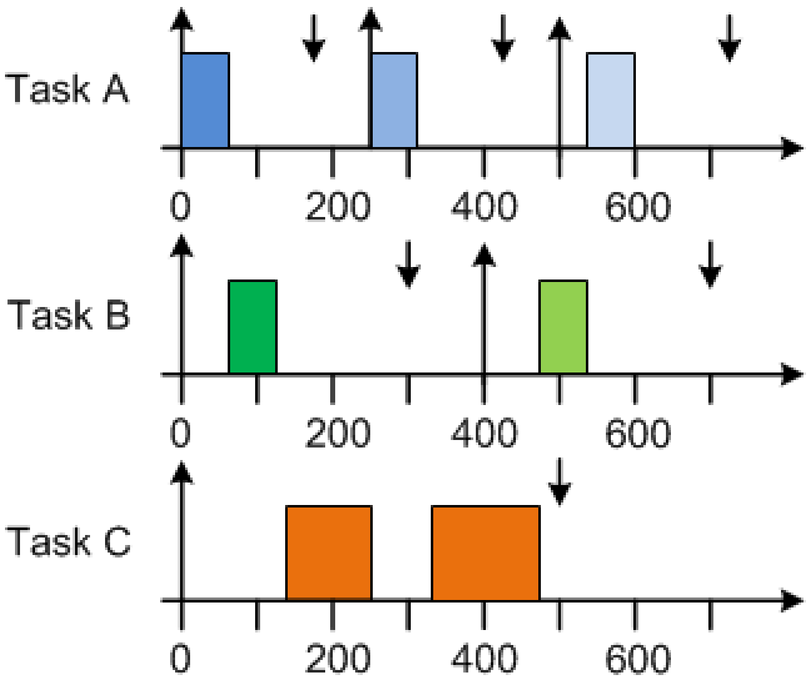
Scheduling policies: Online

- Fixed Task Priority
  - Referred to as Fixed Priority (FP) in uniprocessor scheduling
  - Each task has a fixed priority, each job of a task has the same priority as the task
  - At run-time the scheduler executes the ready job with the highest priority
- Fixed Job Priority
  - Each job has a fixed priority (jobs of the same task can have different priorities)
  - Earliest Deadline First (EDF) – scheduler executes the ready job with the earliest absolute deadline
- Dynamic Priority
  - The priority of a job can vary at run-time
  - Least Laxity First (LLF) – scheduler executes the ready job with least laxity (absolute deadline less remaining execution time)

Scheduling policies
Fixed Priority (Pre-emptive)



Scheduling policies
Earliest Deadline First (Pre-emptive)



✓

Terminology: scheduling policies

- Pre-emptive / non-pre-emptive
  - A scheduling policy is pre-emptive if it will chose to suspend a job that has started executing, but not yet finished in order to run another job
  - A scheduling policy is non-pre-emptive if it always allows any job that has started executing to complete before starting another job
- Work-conserving
  - A scheduling policy is work-conserving if it never idles the processor when there are jobs ready

Terminology: Schedulability

- Schedulable
  - A taskset is schedulable using a specific scheduling algorithm (policy) if all valid sequences of jobs that may be generated by the taskset can be scheduled without a deadline being missed
- Feasible
  - A taskset is feasible if there exists some scheduling algorithm (policy) under which it is schedulable
- Optimality
  - A scheduling policy is optimal if it can schedule any feasible taskset

EDF is an optimal pre-emptive scheduling policy for single processors

Terminology: Schedulability tests

- Aim to build predictable systems
  - Want to ensure before a system runs that its deadlines will be met [or in the case of probabilistic real-time systems that the chance of a deadline being missed is below a specified threshold e.g. 10^{-9} per hour]
  - Use Schedulability Analysis to calculate offline based on models of the application tasks and the system if deadlines will be met online

- Schedulability tests:
  - Sufficient: All tasksets deemed schedulable by the test are in fact schedulable
  - Necessary: All tasksets deemed unschedulable by the test are in fact unschedulable
  - Exact: Sufficient and necessary

Fixed Priority Scheduling

Fixed Priority Scheduling Early Schedulability Tests

- Utilisation based tests for independent periodic tasks with implicit deadlines
  - Processor utilisation of a taskset $U = \sum_{\forall j} \frac{C_j}{T_j}$
  - 1967 Fineberg & Serlin
    - Two periodic tasks with implicit deadlines (D = T)
    - Better to assign the higher priority to the task with the shorter period
    - Schedulable provided that $U \leq 2(\sqrt{2} - 1) \approx 0.83$
  - 1973 Liu & Layland (also Serlin 1972)
    - *n* periodic tasks with implicit deadlines (D=T)
    - Rate Monotonic is the optimal priority assignment policy
    - Schedulable provided that

$$U \leq n(\sqrt[n]{2} - 1) \quad \rightarrow \ln(2) \approx 0.693$$

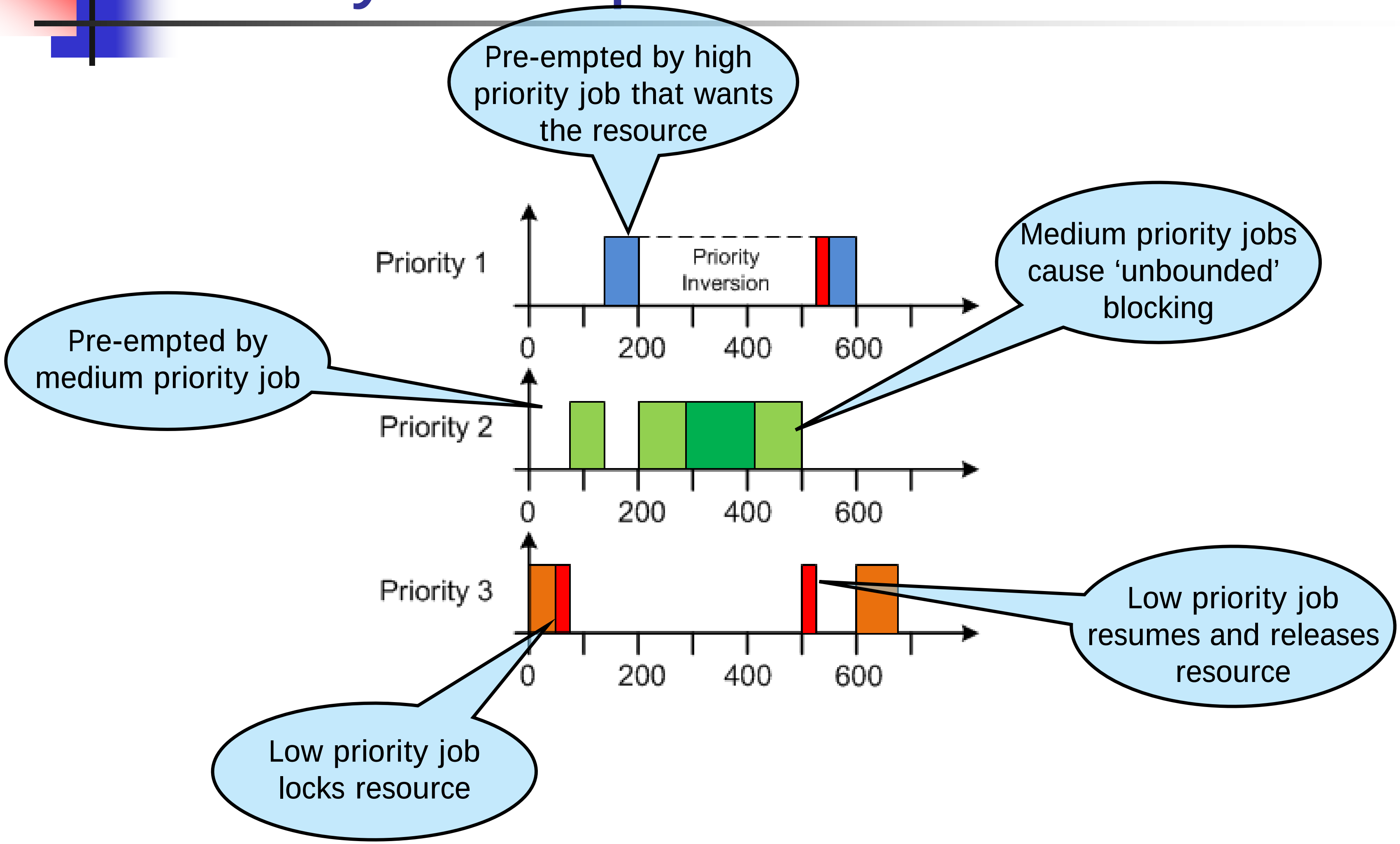
Test are sufficient but not exact: many tasksets fail the tests but are schedulable

Resource sharing

- Mutually exclusive access to shared resources
  - Peripheral devices and communications
  - Shared data structures requiring atomic update to preserve data consistency

  - Non-pre-emptive scheduling solves this but at a high cost to schedulability
- Simple approach using semaphores or mutexes
  - Impacts schedulability
  - Problems of Deadlock

Resource sharing: Binary Semaphores



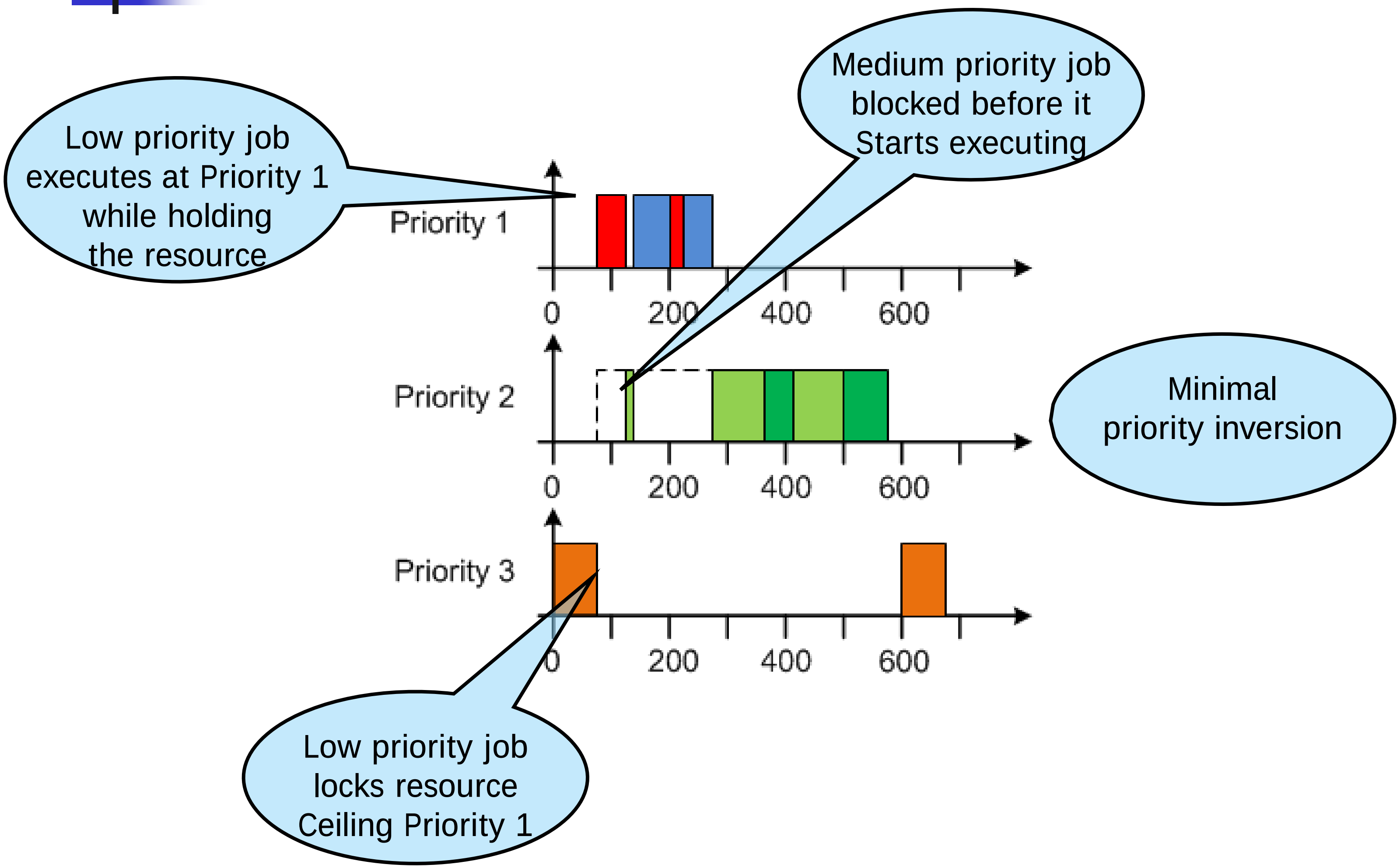
Stack Resource Policy (SRP)

- Stack Resource Policy for FP scheduling
  - SRP assigns a Ceiling Priority to each Resource equal to the highest priority of any task that accesses the resource

- On locking a resource
  - Job's priority is saved (on the stack) and its priority is raised to the higher of its current priority and the Ceiling Priority of the resource
- On unlocking a resource
  - Job's previous priority is restored (from the stack)
- Fixed priority scheduling
  - Takes care of mutual exclusion as no job can access a resource that is already locked as its priority will not be high enough to pre-empt the job that locked the resource

[T.P. Baker "Stack-based Scheduling of Real-Time Processes." Real-Time Systems Journal (3)1, pages 67-100. 1991]

Resource sharing: Stack Resource Policy



Stack Resource Policy (SRP)

- Schedulability analysis with SRP: Blocking factor B_i
  - B_i is the time for which a job of task τ_i may be blocked from executing by jobs of lower priority tasks
  - B_i is limited to the (single) longest time that a job of a lower priority task executes with a resource locked that has a Ceiling Priority of i or higher (i.e. a resource shared with task τ_i or a task of higher priority)
- Properties of SRP
  - Resource access is serialised, once a job starts to execute it never has to wait for a lower priority job to unlock a resource, so no additional context switches due to resource locking
  - Deadlock free
  - Permits and requires properly nested resource access
  - Enables single stack execution (Important in RTOS: OSEK, Autosar)

[T.P. Baker "Stack-based Scheduling of Real-Time Processes." Real-Time Systems Journal (3)1, pages 67-100. 1991]

Stack Resource Policy (SRP)

- Name confusion
  - Often referred to as the Priority Ceiling Protocol, but that is actually a more complex policy without some of the nice properties of SRP - check out Baker's original paper for a comparison
  - SRP sometimes also referred to as the Immediate Priority Ceiling Protocol
  - Don't believe what it says on Wikipedia!

- Utilisation based tests with blocking

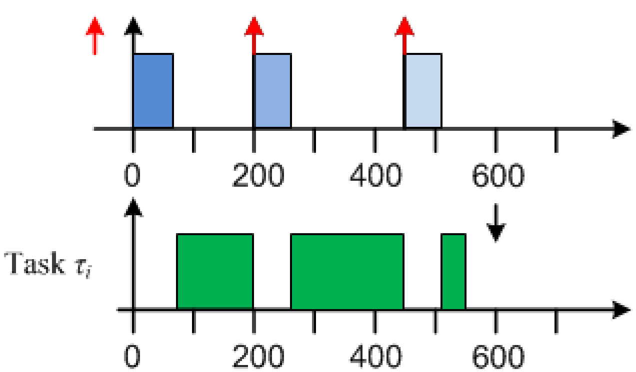
$$\sum_{\forall j}\left(\frac{C_j}{T_j}+\frac{B_j}{T_j}\right)\leq n(\sqrt[n]{2}-1)$$

Fixed Priority Scheduling: Response Time Tests

- Response Time Test
  - Worst-Case Response Time R_i is the longest time that a job of task τ_i can take from arrival to completion (= release jitter + longest time from release to completion)
  - Response time compared to the task's deadline to determine schedulability ($R_i \leq D_i$)
  - Precise calculation of R_i gives an exact test

Key concepts
Critical Instant

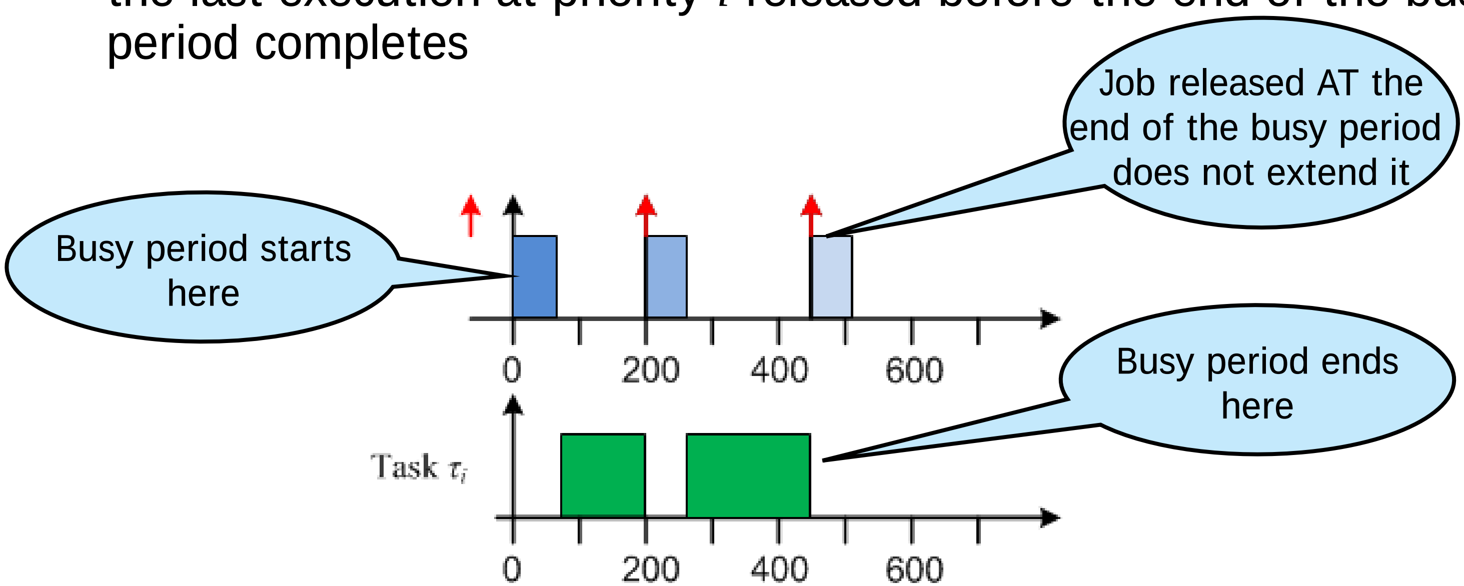
- Critical Instant
  - Defines a scenario or pattern of job releases such that a job of task τ_i experiences its worst-case response time
    - Synchronous release of a job of task τ_i and jobs of all higher priority tasks, which are then released again as soon as possible
    - First job of each higher priority task has maximum release jitter, subsequent jobs have zero jitter (maximises interference)
    - First job of task τ_i has maximum release jitter (maximises release delay)
    - Low priority task locks a resource creating the maximum blocking just before this synchronous release (maximises blocking)
  - Proof this is the worst-case?



THE UNIVERSITY *of York* — RTS *York*

Key concepts
Priority level-i Busy Period

- Priority level-*i* Busy period
  - Period of time $[t_1, t_2)$ during which tasks, of priority i or higher $hp(i)$, that were released at the start of the busy period at t_1, or during the busy period but strictly before its end at t_2, are either executing or *ready to execute*.
  - With pre-emptive scheduling, the end of the busy period is when the last execution at priority i released before the end of the busy period completes



Response Time Analysis Constrained Deadlines

- Sporadic Tasks with Constrained Deadlines
  - Worst-case response time R_i of task τ_i corresponds to its release jitter + longest time from release to completion

$$R_i = w_i + J_i$$

  - Longest time from release to completion equates to the length w_i of the longest priority level-i busy period including one job of task τ_i
    - Only need to include one job of task τ_i because if the busy period is not finished by the next release then the job is unschedulable (as $D_i \leq T_i$)
  - Critical instant:
    - Defines how we determine the length of the busy period

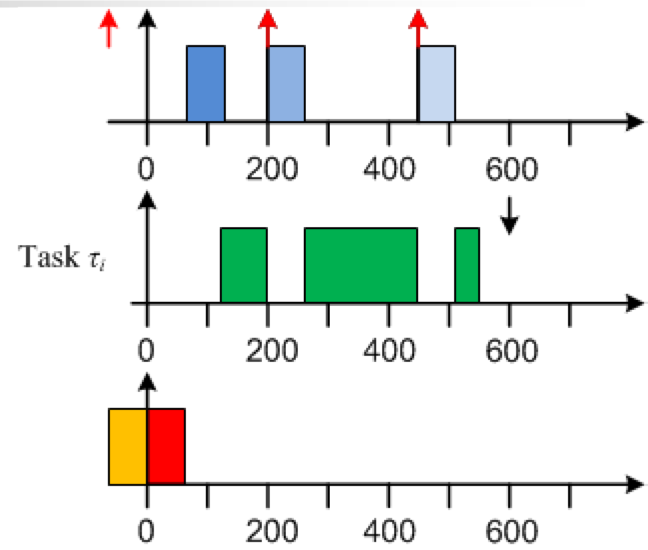
Busy-period calculation

Blocking from a job of a lower Priority task

Interference from multiple jobs of higher priority tasks

$$w_i = B_i + C_i + \sum_{\forall j \in hp(i)} I_j$$

Execution time of task τ_i



Number of releases of higher priority tasks

$$\left\lceil \frac{w_i + J_j}{T_j} \right\rceil$$

Increased with release jitter of $hp(i)$ tasks

Interference

$$I_j = \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

Response Time Analysis Constrained Deadlines

$$w_i^{m+1} = B_i + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^m + J_j}{T_j} \right\rceil C_j$$

- Solution?
  - Busy period length w_i on both LHS and RHS of equation
  - RHS is a monotonic non-decreasing function of w_i
  - Solve using a fixed point iteration starting with $w_i = B_i + C_i$
  - Ends when $w_i^{m+1} + J_i > D_i$ in which case the task is unschedulable
  - Or on convergence $w_i^{m+1} = w_i^m$ in which case the task is schedulable

$$R_i = w_i + J_i$$

- Need to check all tasks to show taskset is schedulable
- Exact test – pseudo-polynomial complexity
  - Can speed up convergence by starting with better initial values

[N.C. Audsley, A. Burns, M. Richardson, A.J. Wellings, “Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling”. Software Engineering Journal, 8(5), pages 284-292, 1993.]

[R.I. Davis, A. Zabos, A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems”. IEEE Transactions on Computers, Vol. 57, No. 9, pages 1261-1276, 2008]

Response Time Analysis

Example:

Task	Execution Time	Deadline	Period
A	3	7	7
B	2	12	12
C	5	20	20

$$w_c^0 = 5$$

$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 2 = 10$$

$$w_c^2 = 5 + \left\lceil \frac{10}{7} \right\rceil 3 + \left\lceil \frac{10}{12} \right\rceil 2 = 13$$

$$w_c^3 = 5 + \left\lceil \frac{13}{7} \right\rceil 3 + \left\lceil \frac{13}{12} \right\rceil 2 = 15$$

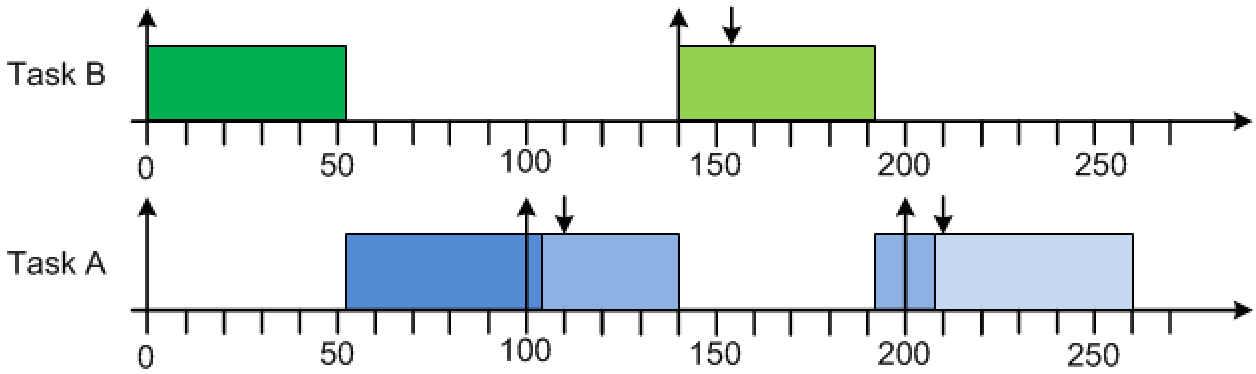
$$w_c^4 = 5 + \left\lceil \frac{15}{7} \right\rceil 3 + \left\lceil \frac{15}{12} \right\rceil 2 = 18$$

$$w_c^5 = 5 + \left\lceil \frac{18}{7} \right\rceil 3 + \left\lceil \frac{18}{12} \right\rceil 2 = 18$$

$$R_c = 18$$

Response Time Analysis Arbitrary Deadlines

Task	Execution Time	Deadline	Period
A	52	110	100
B	52	154	140



Response time of first job of task A is 104

Response time of second job of task A is 106

[Lehoczky J., "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In proceedings Real-Time Systems Symposium, pages 201–209, 1990]

Response Time Analysis Arbitrary Deadlines

- Busy period and response times
  - Worst-case response time occurs for some job of task τ_i in the priority level-i busy period following a critical instant
    - Characterised as before: Simultaneous release of jobs of task τ_i and all higher priority tasks. All jobs re-arrive as soon as possible. Blocking due to a lower priority task at the start of the busy period
  - However, busy period does not end with completion of the first job because by then another job of the same task may have been released
  - Don't know which job of task τ_i will have the worst-case response time, so need to check all of them in the priority level-i busy period
- Length of Busy Period $L_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{L_i^m + J_j}{T_j} \right\rceil C_j$
  - Solve via fixed point iteration
  - Number of jobs of task τ_i in busy period $Q_i = \left\lceil \frac{L_i + J_i}{T_i} \right\rceil$

FP: Response Time Analysis: Arbitrary Deadlines

- Find **completion** time of q^{th} job of task τ_i from start of busy period (q = 0 is the first):

$$w_{i,q}^{m+1} = B_i + (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_{i,q}^{m} + J_j}{T_j} \right\rceil C_j$$

  - Starts with initial value $w_{i,q}^{0} = B_i + (q+1)C_i$
  - End when $w_{i,q}^{m+1} - qT_i + J_i > D_i$ (job and task is unschedulable)
  - or when $w_{i,q}^{m+1} = w_{i,q}^{m}$ completion time is then $W_{i,q} = w_{i,q}^{m+1}$
  - Do this for $q = 0,1,2,3,...Q_i - 1$ checking all jobs in the busy period
  - Then WCRT $R_i = \max_{\forall q=0,1,2...Q_i-1}(W_{i,q}^{p} - qT_i + J_i)$
  - Task schedulable provided that $R_i \le D_i$

Exact test – pseudo-polynomial complexity

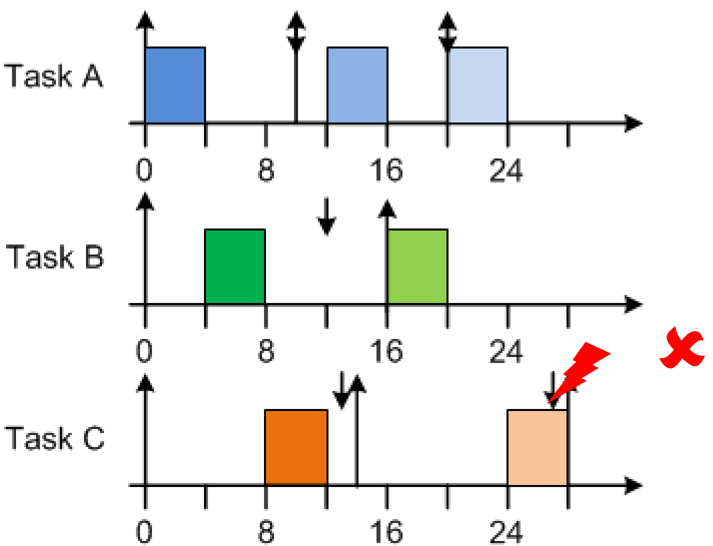
[K.W. Tindell, A. Burns, A.J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks". Real-Time Systems. Volume 6, Number 2, pages 133-151, 1994]

Response Time Analysis Non-pre-emptive Scheduling

- Similar approach as for arbitrary deadlines
  - Need to look at the whole priority level-i busy period as WCRT may not occur for first job of task τ_i following a critical instant **even with constrained deadlines**
  - Due to push-through blocking from the previous job of the same task

Example

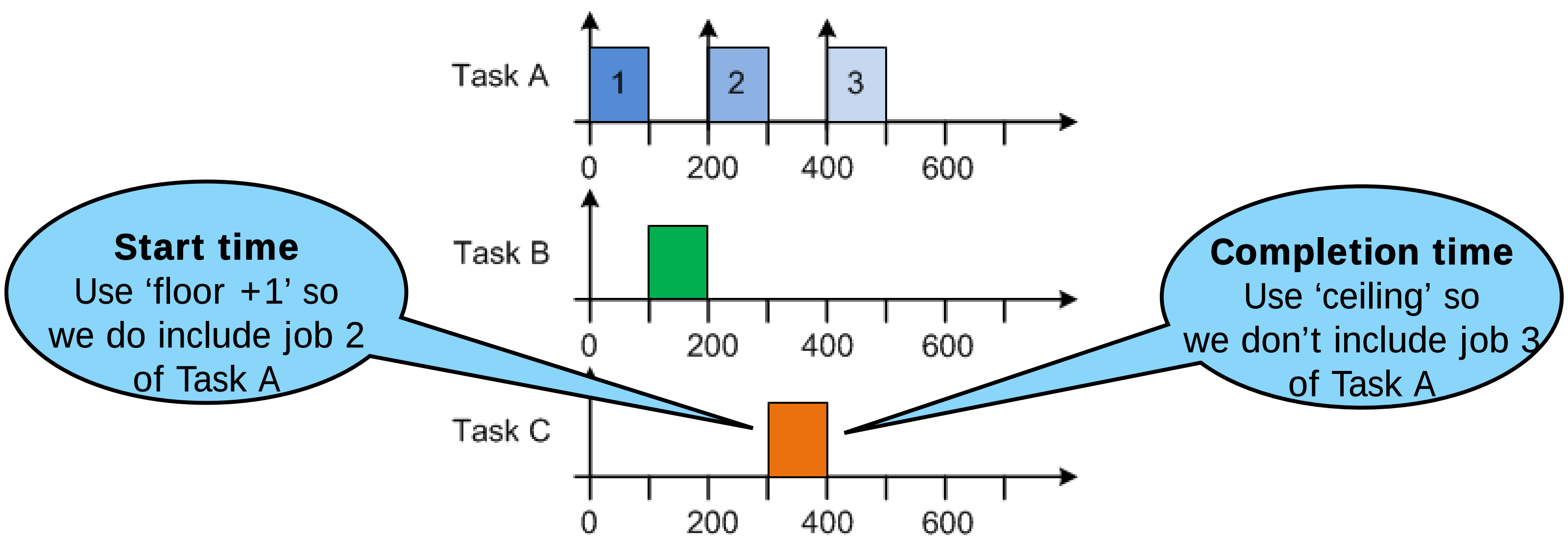
Task	Execution Time	Deadline	Period
A	4	10	10
B	4	12	16
C	4	13	14



- First job of task C, R = 12
- Second job, R = 14

Response Time Analysis Non-pre-emptive Scheduling

- Interested in start time of jobs
  - Non-pre-emptive scheduling so once a job starts it finishes C_i later
  - Need to find first time w_i when on the next time unit processor would be idle and so the next job of task τ_i could start
  - Use $\lfloor \quad \rfloor + 1$ rather than $\lceil \quad \rceil$



Response Time Analysis Non-pre-emptive Scheduling

- Blocking $$B_i = \max_{\forall k \in lp(i)} (C_k - 1)$$
- Find **start** time of q^{th} job of task τ_i from start of busy period:
  - Once it starts, it will finish C_i later (as its non-pre-emptable)

$$w_{i,q}^{m+1} = B_i + qC_i + \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{w_{i,q}^m + J_j}{T_j} \right\rfloor + 1 \right) C_j$$

  - Iteration starts with initial value $w_{i,q}^0 = B_i + qC_i$
  - Ends when $w_{i,q}^{m+1} + C_i - qT_i + J_i > D_i$ (job and task is unschedulable)
  - or when $w_{i,q}^{m+1} = w_{i,q}^m$ start time is then $W_{i,q} = w_{i,q}^{m+1}$
  - Do this for $q = 0,1,2,3,\ldots Q_i - 1$ checking all jobs in the busy period
  - Then WCRT $R_i = \max_{\forall q = 0,1,2\ldots Q_i - 1} (W_{i,q} + C_i - qT_i + J_i)$
  - Task schedulable provided that $R_i \le D_i$

Exact test – pseudo-polynomial complexity

Response Time Analysis Non-pre-emptive Scheduling

- Simpler test for constrained deadlines
  - Idea of push-through or self-blocking from the previous job of the same task

$$B_i = \max_{\forall k \in lp(i)} (C_k - 1)$$

$$w_i^{m+1} = \max(\underline{B_i, C_i}) + \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{w_i^m + J_j}{T_j} \right\rfloor + 1 \right) C_j$$

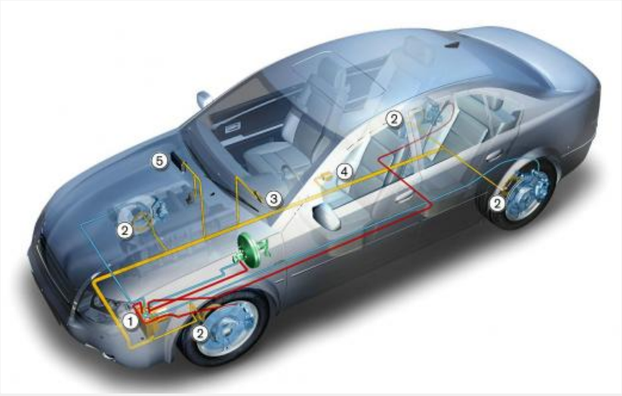
$$R_i = w_i + C_i + J_i$$

$$R_i \le D_i$$

  - Now only need to check one job
  - Test is only sufficient, it may deem some tasksets unschedulable that are in fact schedulable

Controller Area Network

- CAN is a simple broadcast network used in nearly all cars sold today
  - Approx. 1 billion CAN enabled microcontrollers sold each year
  - Typical cars today have 20 – 30 ECUs inter-connected via 2 or more CAN buses
  - CAN messages scheduled non-pre-emptively with message arbitration making the network similar in terms of analysis to a single processor



Controller Area Network

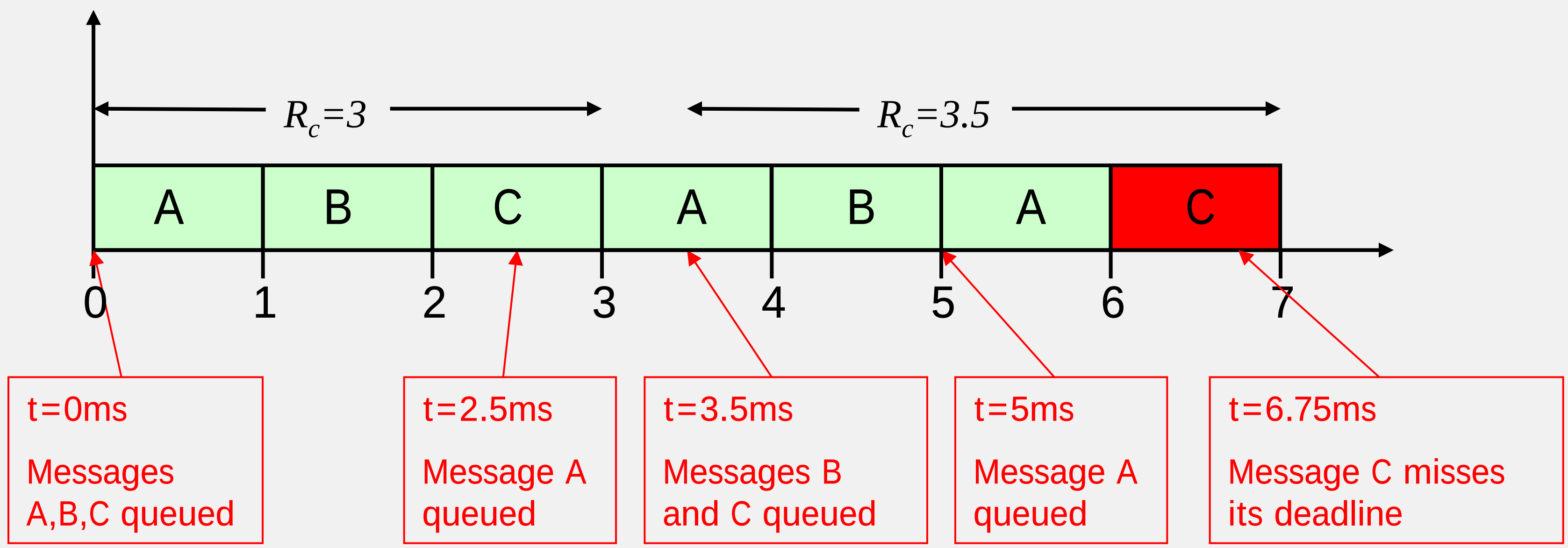
- Schedulability analysis for CAN developed in 1993-95
  The original analysis was:
  - Used in teaching
  - Referenced in over 500 subsequent research papers
  - Lead to at least two PhD Theses
  - In 1995 recognised by Volvo Car Corporation used in the development of the Volvo S80 (P23)
  - Formed basis of commercial CAN analysis tools now owned by Mentor Graphics
  - Used by many Automotive manufacturers who built millions of cars with networks analysed using these techniques
  - Enabled increases in network utilisation from 30-40% to typically 70-80%

But, it was flawed...

[K.W. Tindell, A. Burns, A.J. Wellings, "Calculating Controller Area Network (CAN) Message Response Times", Control Engineering Practice, Vol 3, No 8, pp1163-1169, 1995. DOI:10.1016/0967-0661(95)00112-8]

Example

Msg	Pri	Period	Deadline	TX Time	R
A	1	2.5ms	2.5ms	1ms	2ms
B	2	3.5ms	3.25ms	1ms	3ms
C	3	3.5ms	3.25ms	1ms	3ms

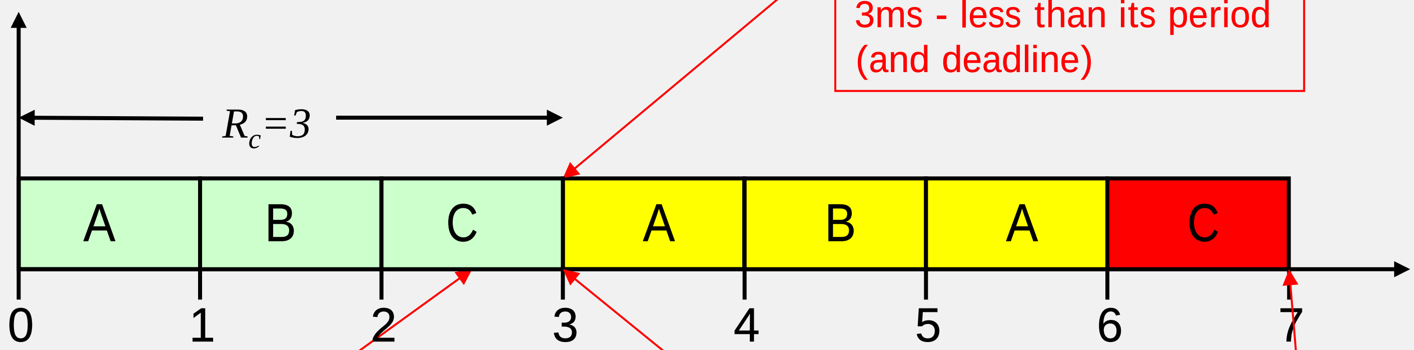


The original CAN schedulability analysis gave an optimistic response time for message C: 3ms v. 3.5ms

But 2nd instance of message C misses its deadline

What is the flaw in the analysis?

Response time of 1st instance of message C is 3ms - less than its period (and deadline)



BUT transmission of message C is non-pre-emptive and blocks message A, pushing extra interference into next period of C

Busy period at priority of message C does NOT end with transmission of message C

Busy period ends here. Must examine all instances of message C in the busy period to find WCRT

Revised Analysis for CAN

[R.I.Davis, A. Burns, R.J. Bril, and J.J. Lukkien, "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised". *Real-Time Systems*, Volume 35, Number 3, pp. 239-272, April 2007. DOI: 10.1007/s11241-007-9012-7]

Volcano Network Architect

Mentor Graphics

- Commercial CAN schedulability analysis tool
- Used a sufficient schedulability test, assuming maximum possible blocking factor irrespective of message priorities
- Equates to the simple sufficient test and is therefore slightly pessimistic but correct
- Used to analyse CAN systems for
  Volvo S80, S/V/XC 70, S40, V50, XC90
  and many other cars from other manufacturers

Response Time Analysis for Controller Area Network

- Great example of applying schedulability analysis
  - No WCET problem (TX times)
  - No awkward overheads
- Does everything match the classic schedulability analysis?

  No!
  - CAN peripherals and SW Device Drivers behaviours
  - Mix of priority and FIFO queues at different levels
    - So not quite 'Fixed Priority Scheduling'
  - Non-abortable TX buffers
    - Cause unbounded priority inversion while low priority message waits to be sent (blocking all those behind it)

Still work to be done 20 years after the first papers were published

[R.I. Davis, S. Kollmann, V. Pollex, F. Slomka, "Schedulability Analysis for Controller Area Network (CAN) with FIFO Queues Priority Queues and Gateways". Real-Time Systems, Volume 49, Issue 1, pages 73-116, Jan 2013]

[D.A. Khan, R.I. Davis, N. Navet "Schedulability Analysis of CAN with Non-abortable Transmission Requests". In proceedings IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2011]

Analysing Real Systems: Uniprocessors

- What else do we need to account for?
  - RTOS behaviour and overheads
    - Context switch times
    - Blocking due to API calls (critical sections) and context switches
    - Tick-driven or Event-driven scheduler
  - Interference from Interrupt Handlers
  - More complex task behaviours

- It can be done
  - ETAS RTA-OSEK (and RTA-OS Autosar) RTOS Schedulability analysis tools exist taking into account transactions, offsets, RTOS behaviour and overheads
- But its not easy
  - Needs careful RTOS design (compliant with scheduling theory) and determination of worst-case overheads
  - Additions to schedulability analysis for simple theoretical models

Response Time Analysis for Real Systems

- Example: analysis for an event-driven RTOS
  - Non-pre-emptive RTOS execution at the start and end of each job

$$C_i^{pre} + C_i + C_i^{post}$$

$$B_i^{CS} = \max(B_i, \max_{\forall k \in lp(i)} (C_k^{pre}, C_k^{post}))$$

$$w_i^{m+1} = \max(B_i^{CS}, C_i^{post}) + C_i^{pre} + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^m + J_j}{T_j} \right\rceil (C_i^{pre} + C_i + C_i^{post})$$

$$R_i = w_i + J_i$$

  - Also typically need more sophisticated modelling of task timing behaviour e.g. offsets, transactions, more complex arrival patterns, etc. Interference from Interrupt Handlers (bursty arrivals) etc.

THE UNIVERSITY of York

RTS York

Fixed Priority Scheduling

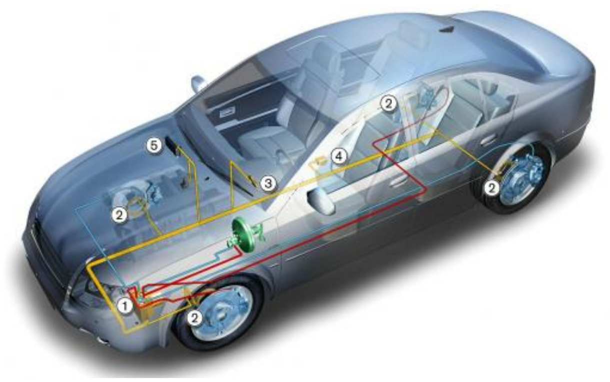
- Scheduling and schedulability analysis are only half the story...

- What about **Priority** Assignment?
  - Why is it important?
  - What is an optimal assignment?
  - How do we find it?
  - Is Optimal Priority Assignment enough? Can we optimise other things as well?

Priority assignment

- Why is priority assignment important
  - Achieve a schedulable system when it otherwise wouldn't be
  - Provide a schedulable system avoiding hardware overprovision / maximising use of hardware resources
  - Provide headroom for unforeseen interference or overruns

- Example
  - Controller Area Network (CAN)
  - Used for in-vehicle networks
  - Message IDs are the priorities



When priority assignment goes bad!

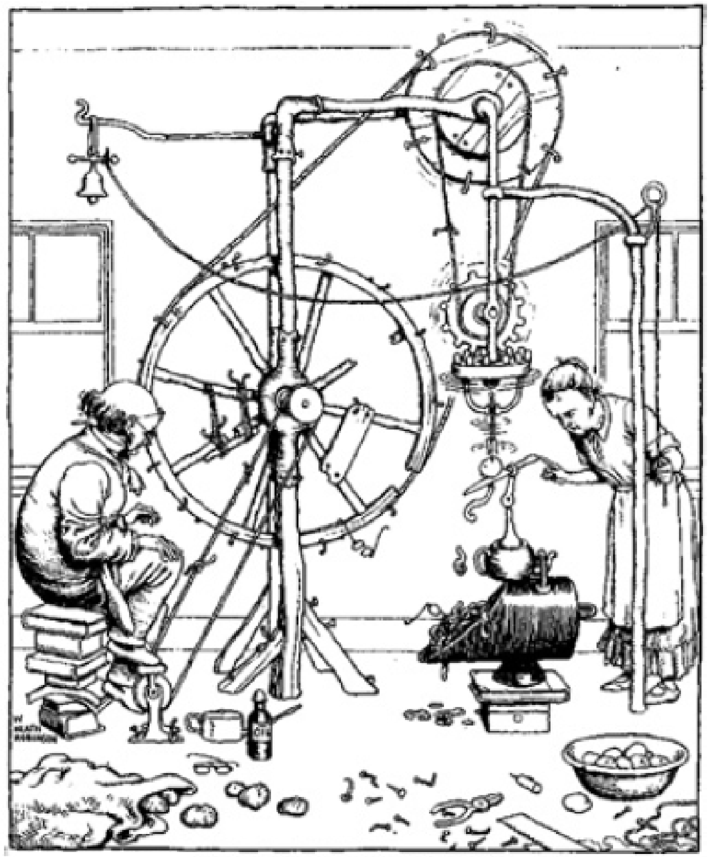
- From Darren Buttle's Keynote talk at ECRTS 2012

  The myth of CAN bus Utilisation –

  "You cannot run CAN reliably at more than 30% utilisation[1]"

  [1] Figures may vary but not significantly

- Why?
  - Message IDs i.e. priorities assigned in an ad-hoc way reflecting data and ECU supplier (legacy issues)
  - ...as well as many other issues, including device driver implementation



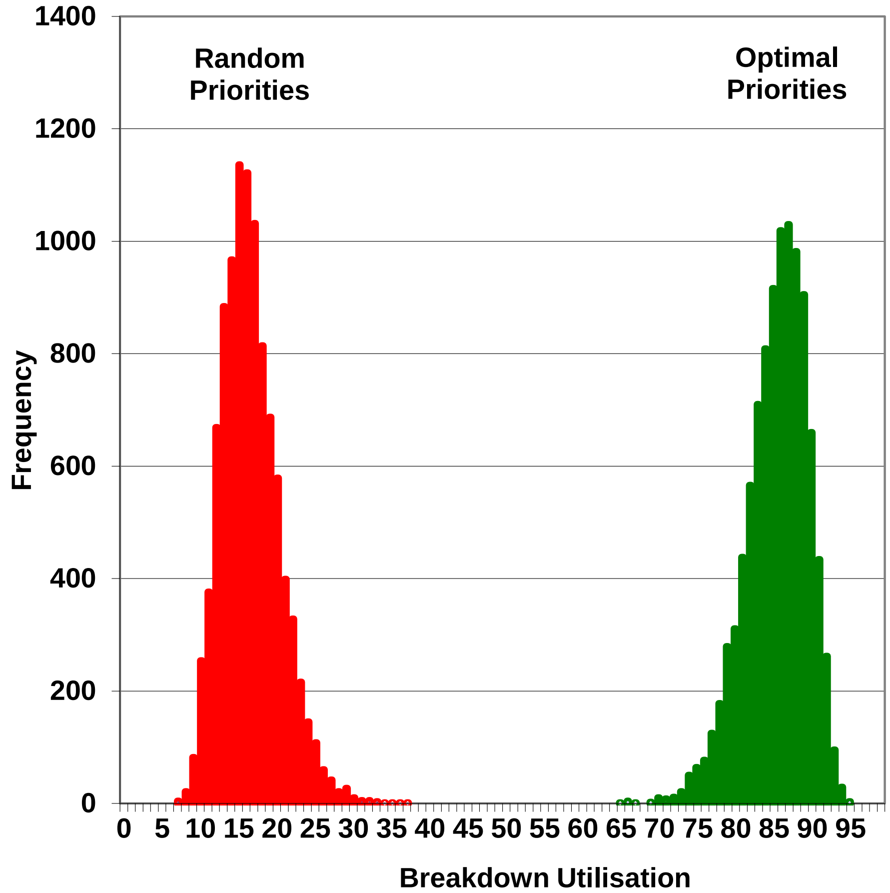
The Professor's invention for peeling potatoes.

When priority assignment goes bad!

- Example: CAN
  - Typical automotive config:
    - 80 messages
    - 10ms -1s periods
    - All priority queues
  - x10,000 message sets

- Breakdown utilisation
  - Scale bus speed to find util. at which deadlines are missed

80% v 30% or less



[R.I. Davis, S. Kollmann, V. Pollex, F. Slomka, "Schedulability Analysis for Controller Area Network (CAN) with FIFO Queues Priority Queues and Gateways". *Real-Time Systems,* 2012]

Optimal priority assignment

- Definition Optimal priority assignment

For a given system model, a priority assignment policy P is referred to as optimal if there are no systems, compliant with the model, that are schedulable using fixed priority scheduling with another priority assignment policy that are not also schedulable using policy P.

For fixed priority scheduling, by using an optimal priority assignment policy we can schedule any system that can be scheduled under using any other priority assignment policy

[N.C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Dept. Computer Science, University of York, UK, 1991.]
[N.C. Audsley, “On priority assignment in fixed priority scheduling”, Information Processing Letters, 79(1): 39-44, May 2001.]

Early work on priority assignment

- 1967 Fineberg & Serlin
  - Two periodic tasks with implicit deadlines, better to assign the higher priority to the task with the shorter period
- 1973 Liu & Layland
  - Rate-Monotonic priority ordering is optimal for implicit deadline periodic tasksets (synchronous arrivals)
- 1982 Leung & Whitehead
  - Deadline-Monotonic priority ordering is optimal for constrained deadline tasksets (synchronous arrivals)
  - Deadline Monotonic not optimal for the asynchronous case (offsets)
- 1990 Lehoczky
  - Deadline Monotonic not optimal for arbitrary deadline tasksets
- 1994 Burns et al.
  - Deadline Monotonic not optimal for deadlines prior to completion
- 1996 George
  - Deadline Monotonic not optimal for non-pre-emptive scheduling

Deadline Monotonic optimality

- Deadline Monotonic Priority Ordering (DMPO) Optimal for synchronous constrained deadline tasksets
  - Response time analysis

$$R_i^{m+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^m}{T_j} \right\rceil C_j$$

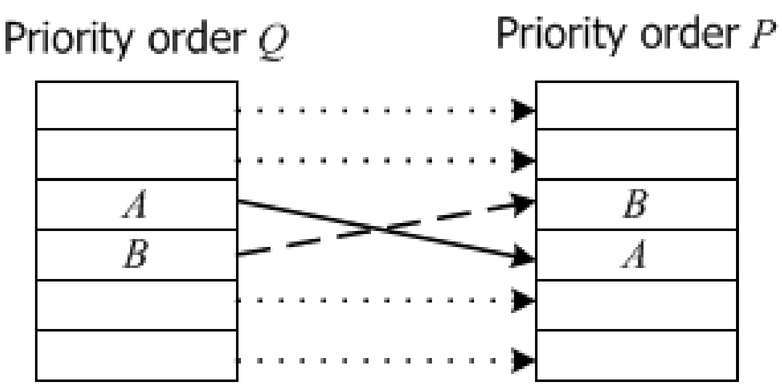
Proof sketch

Assume some other priority ordering Q is schedulable

Swap two tasks A and B at adjacent priorities where $D_A > D_B$ and A is at a higher priority and the taskset will remain schedulable

Priority order Q : let $y = R_B \le D_B \le T_B$

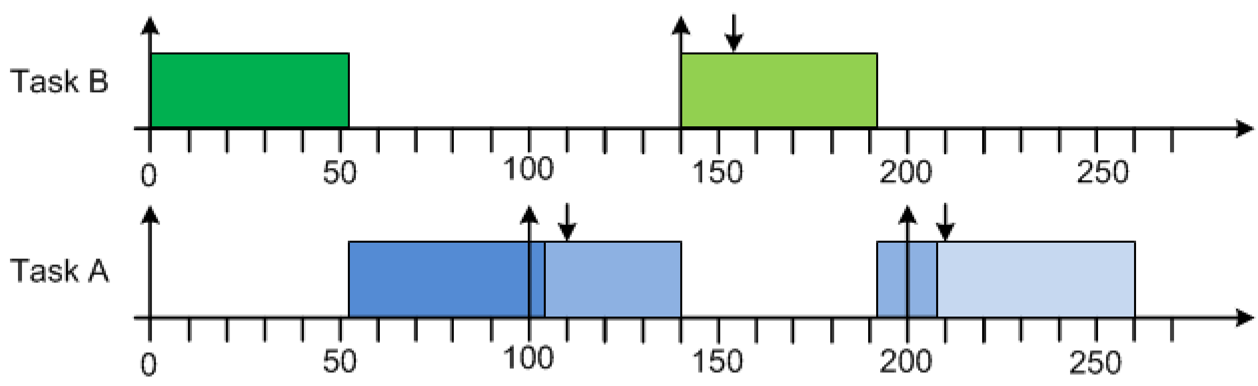
Priority order P : $R_A = y$ (as $y \le T_B$) and so there is interference from only one job of task B, hence as $D_A > D_B$ task A is schedulable



Deadline Monotonic: non-optimality

Tasks with arbitrary deadlines

Task	Execution Time	Deadline	Period
A	52	110	100
B	52	154	140

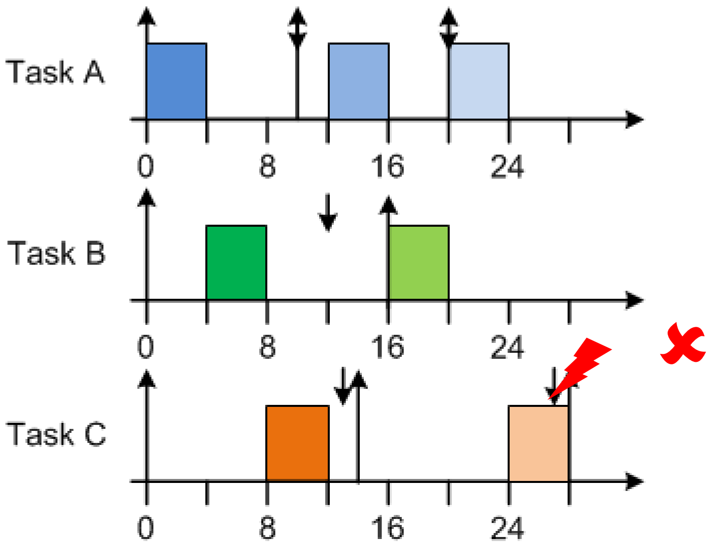


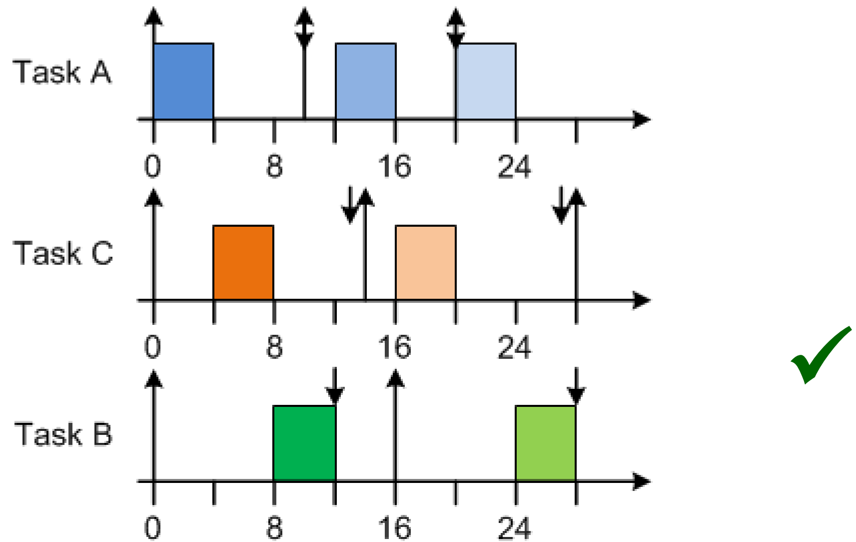
[Lehoczky J., “Fixed priority scheduling of periodic task sets with arbitrary deadlines”. In proceedings Real-Time Systems Symposium, pages 201–209, 1990]

Deadline Monotonic: non-optimality

- Non-pre-emptive scheduling

Task	Execution Time	Deadline	Period
A	4	10	10
B	4	12	16
C	4	13	14



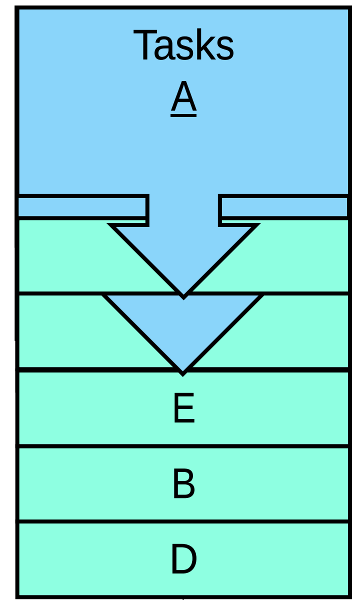


[L. George, N. Rivierre, M. Spuri, "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", INRIA Research Report, No. 2966, September 1996]

Example derived from: [R.I. Davis and A. Burns "Robust priority assignment for messages on Controller Area Network (CAN)". Real-Time Systems, Volume 41, Issue 2, pages 152-180, February 2009]

Optimal Priority Assignment

```
for each priority level i, lowest first {
    for each unassigned task τ {
         if τ is schedulable at priority i
         assuming that all unassigned tasks are
         at higher priorities {
              assign task τ to priority level i
              break (exit for loop)
         }
    }
    if no tasks are schedulable at priority i {
         return unschedulable
    }
}
return schedulable
```



$n(n+1)/2$ schedulability tests rather than $n!$
by exploring all possible orderings
$n = 25$, that is 325 tests rather than 15511210043330985984000000

[N.C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Dept. Computer Science, University of York, UK, 1991.]

[N.C. Audsley, "On priority assignment in fixed priority scheduling", Information Processing Letters, 79(1): 39-44, May 2001.]

[K. Bletsas, and N.C. Audsley, "Optimal priority assignment in the presence of blocking". *Information Processing Letters* Vol. 99, No. 3, pp83-86, August. 2006]

OPA applicability

Powerful idea as we have said very little about the actual schedulability test hence broad applicability

OPA algorithm provides optimal priority assignment w.r.t. any schedulability test *S* for fixed priority scheduling provided that three conditions are met...

Condition 1: Schedulability of a task may, according to the test, be dependent on the set of higher priority tasks, but not on their relative priority ordering

Condition 2: Schedulability of a task may, according to the test, be dependent on the set of lower priority tasks, but not on their relative priority ordering

Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority

- Tests meeting these conditions referred to as **OPA-compatible**

Applicability

- Resource sharing, offsets, arbitrary deadlines, deadlines before completion, non-pre-emptive, CAN, multiframe tasks, mixed criticality, some global FP scheduling on multiprocessors

[R.I. Davis, A. Burns "Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". In proceedings Real-Time Systems Symposium pp 398-409, 2009.]

Minimising the number of Priority Levels with OPA

Important for practical systems that may support only a limited number of priorities

```
for each priority level i, lowest first {
    Z = empty set
    for each unassigned task τ {
        if τ is schedulable at priority i  assuming that
        all unassigned tasks are  at higher priorities {
            add τ to Z
        }
    }
    if no tasks are schedulable at priority i {
        return unschedulable
    }
    else {
        assign all tasks in Z to priority i
    }
    if no unassigned tasks remain {
        break
    }
}
return schedulable
```

[N.C. Audsley, "On priority assignment in fixed priority scheduling", Information Processing Letters, 79(1): 39-44, May 2001.]

Robust Priority Assignment

- Drawback of OPA algorithm
  - Arbitrary choice of schedulable tasks at each priority
  - May leave the system only just schedulable – i.e fragile not robust to minor changes

- In practice tasks may be subject to additional interference
  - Execution time budget overruns; interrupts occurring in bursts or at ill-defined rates; ill-defined RTOS overheads; ill-defined critical sections; cycle stealing by peripheral devices (DMA) etc. etc.

- Want a robust priority ordering
  - As well as being optimal, able to tolerate the maximum amount of additional interference
  - General model of additional interference $E(\alpha, w, i)$ (= α)

[R.I. Davis, A. Burns. "Robust Priority Assignment for Fixed Priority Real-Time Systems”. In proceedings IEEE Real-Time Systems Symposium pp. 3-14. Tucson, Arizona, USA. December 2007.]

RPA Algorithm

```
for each priority level i, lowest first
{
    for each unassigned task τ
    {
         determine the largest value of α for which task τ
         is schedulable at priority i assuming that all
         unassigned tasks have higher priorities
    }
    if no tasks are schedulable at priority i
    {
         return unschedulable
    }
    else
    {
         assign the schedulable task that tolerates the
         max α at priority i to priority i
    }
}
return schedulable
```

Ordering achieved in optimal and robust (tolerates the most additional interference (largest α) of any priority ordering)

Robust Priority Assignment

- Example: Non-pre-emptive scheduling
  - Additional interference from single invocation of an interrupt handler with unknown execution time
  - Additional interference $E(\alpha, w, i) = \alpha$

Task	C	D	T
τ_A	125	450	450
τ_B	125	550	550
τ_C	65	600	600
τ_D	125	1000	1000
τ_E	125	2000	2000

THE UNIVERSITY of York

RTS York

Robust Priority Assignment

- Computed values of α

Priority	Task				
	τ_A	τ_B	τ_C	τ_D	τ_E
5	NS	NS	NS	120	**354**
4	NS	NS	NS	**120**	-
3	10	**110**	74	-	-
2	135	-	**199**	-	-
1	**200**	-	-	-	-

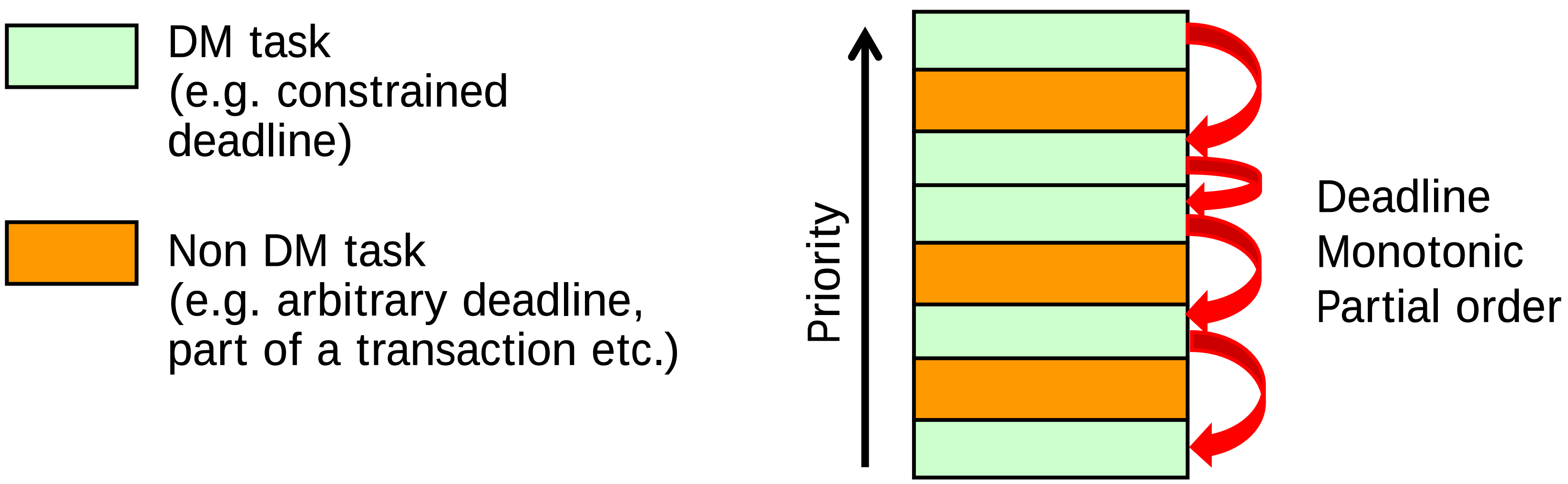
- Robust priority ordering
  - Tolerates additional interference of up to **110** time units
- Deadline monotonic: neither optimal nor robust
  - Tolerates additional interference of up to **74** time units
- OPA: may be worse still
  - Might tolerate additional interference of only **10** time units

Robust Priority Assignment

- Mixed systems: two subsets of tasks
  - “DM tasks”
    - Satisfy the restrictions where Deadline Monotonic priority ordering is known to be optimal
    - Pre-emptable, D≤T, resource access according to SRP, no transactions or offsets
  - “Non DM tasks”
    - Don’t satisfy the restrictions where Deadline Monotonic priority ordering is known to be optimal
    - Pre-emptable with D>T, non-pre-emptable, co-operative scheduling with non-pre-emtable final sections, transactions, non-zero offset

[R.I. Davis, A. Burns. "Robust Priority Assignment for Fixed Priority Real-Time Systems”. In proceedings IEEE Real-Time Systems Symposium pp. 3-14. Tucson, Arizona, USA. December 2007.]

Robust Priority Assignment



For mixed systems containing both DM and non DM tasks, then there exists a robust priority order with the DM tasks in Deadline Monotonic partial order

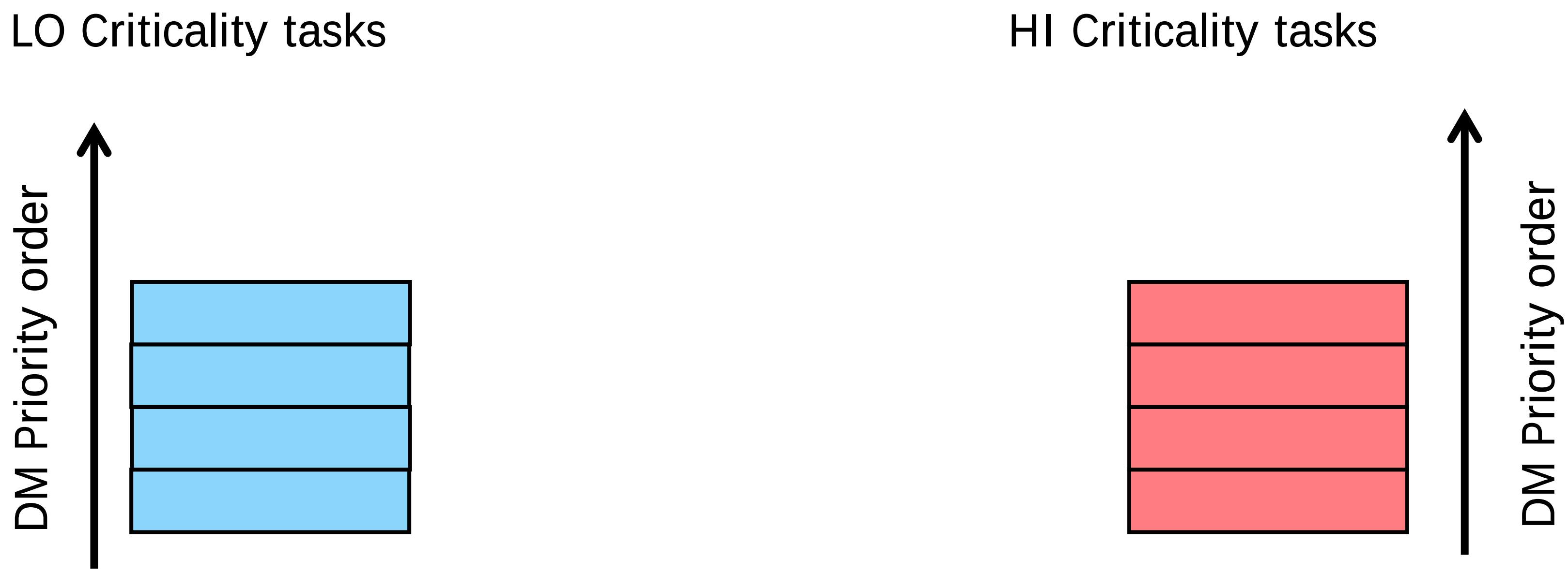
Robust Priority Assignment

- Can improve efficiency of OPA and RPA algorithms
  - Of all the DM tasks, the one with the largest deadline is the one that can tolerate the most additional interference at a given priority level
  - Only one DM task need be checked at each priority level – the one with the largest deadline of all unassigned DM tasks
  - For n tasks, k of which are DM tasks:

    $(n(n+1)-k(k-1))/2$ task schedulability tests instead of $n(n+1)/2$
  - Example: 4 tasks with D > T, 46 constrained deadline tasks
    max. of 240 schedulability tests instead of 1275

[R.I. Davis, A. Burns. "Robust Priority Assignment for Fixed Priority Real-Time Systems". In proceedings IEEE Real-Time Systems Symposium pp. 3-14. Tucson, Arizona, USA. December 2007.]

Fixed Priority Scheduling of Mixed Criticality Systems



2n-1 schedulability tests rather than $n(n+1)/2$

[S.K. Baruah, A. Burns, R.I. Davis "Response Time Analysis for Mixed Criticality Systems" . In proceedings 32nd IEEE Real-Time Systems Symposium (RTSS'11) , pages 34-43, Nov 29th - Dec 2nd, 2011]

Earliest Deadline First

Earliest Deadline First

- Pre-emptive EDF executes the job with the earliest absolute deadline

- Early results
  - Liu and Layland 1973 – showed that any independent, periodic taskset with implicit deadlines is schedulable under EDF if $U \leq 1$
  - Dertouzos 1974 proved that pre-emptive EDF is an optimal scheduling policy for single processors[1]

[1]At least when there are no overheads or complicated things like Cache Related Pre-emption Delays (CRPD)

Stack Resource Polciy(SRP) for EDF

- Uses concept of Pre-emption Levels
  - Each resource has a pre-emption level equal to the minimum Relative Deadline (or D-J) of any task that locks the resource
  - Each job has an initial pre-emption level = Relative Deadline (or D-J) of its task

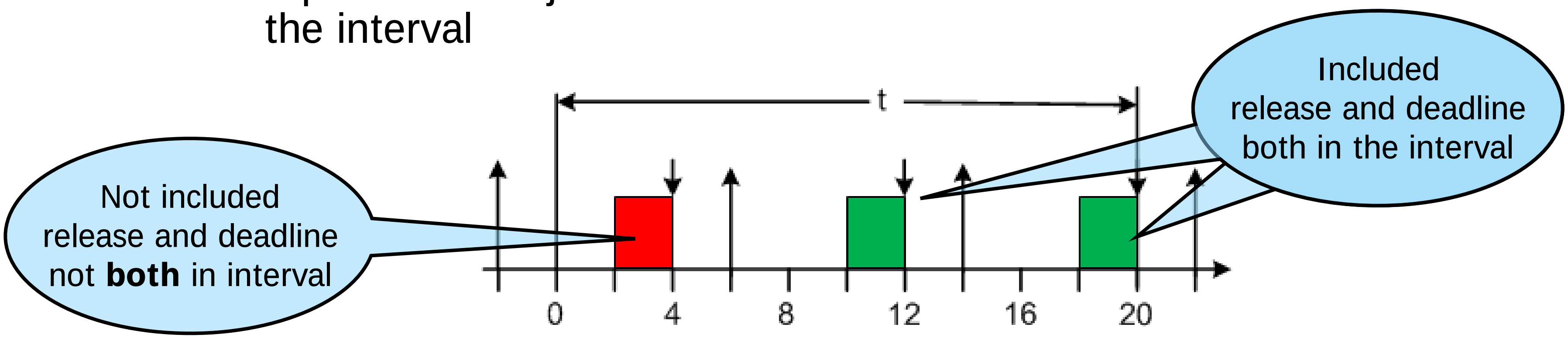
- Run-time operation
  - On locking a resource
    - Save the current pre-emption level of the job on the stack, set its pre-emption level to higher of current level and pre-emption level of the resource (i.e. smaller value)
  - On unlocking a resource
    - Restore the job's pre-emption level from the stack
  - A job may only pre-empt another job if it has an earlier absolute deadline <u>and</u> a strictly higher pre-emption level (smaller value)

Stack Resource Policy (SRP) for EDF

- Properties
  - Deadlock free
  - All blocking before a job starts to actually execute
  - No additional context switches due to resource locking
  - Permits single stack execution
  - Blocking limited to a single resource access from a single job with a larger value of D-J (longer relative deadline minus Jitter)

Exact Schedulability Analysis for Pre-emptive EDF

- Key concept: Processor Demand Bound
  - The maximum processor demand in an interval of length t
  - Equates to the jobs that are released and have their deadlines in the interval



With release jitter we get $\max\left(0, \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor + 1\right)$ jobs of task τ_i

$$h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor + 1\right) C_i$$

Exact Schedulability Analysis for Pre-emptive EDF

- Processor Demand Bound: $h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t + J_i - D_i}{T_i} \right\rfloor + 1\right) C_i$
- Worst-case additional effect of blocking under SRP: $b(t)$

$$b(t) = \max\left(A_{a,k} \mid D_a - J_a > t, D_k - J_k \le t\right)$$

$A_{a,k}$ is the time a job of task τ_a executes for with a resource locked that is also accessed by task τ_k

- Exact schedulability test:
  - $U \le 1$ and overall Processor Demand in any interval t must not exceed the length of the interval:

$$\forall t \quad h(t) + b(t) \le t$$

  - Can limit values of t that we check as $h(t) + b(t)$ only changes at

$$\forall i \quad t = kT_i + D_i - J_i$$

but still infinitely many values to check...

[S.K. Baruah, A.K. Mok, L.E. Rosier, "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In proceedings IEEE Real-Time Systems Symposium (RTSS), pages182-190, 1990]

Exact Schedulability Analysis for EDF

- If taskset is unschedulable test will show this by $t = L$
  - L_a : formula derived from $h(t) + b(t) \le t$ (works for U < 1)

$$L_a = \max\left((D_1 - T_1 - J_1), \ldots (D_n, - T_n, - J_n,), \frac{\max_{d_i < D_{\max}} (b(d_i)) + \sum_{\forall i} (T_i + J_i - D_i) U_i}{1 - U} \right)$$

What happens if D=T, J=0, b(t)=0?

$L_a = 0$ which means we don't need to test any values!

Derived by substituting x for $\lfloor x \rfloor$

  - L_b : length of longest busy period

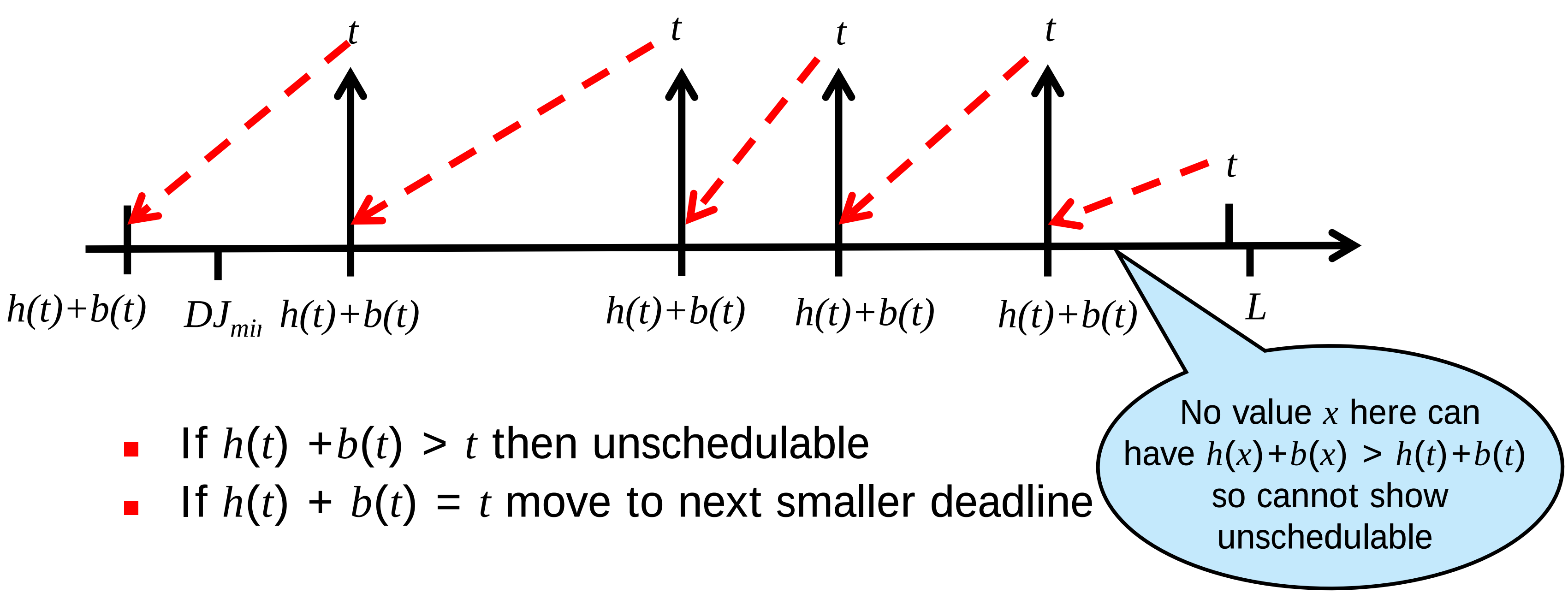
$$L_b^{m+1} = \sum_{\forall j} \left\lceil \frac{L_b^m + J_j}{T_j} \right\rceil C_j$$

  - $L = \min(L_a, L_b)$

Still there can be a very large number of points to check...

Quick Processor demand Analysis: QPA for EDF

- QPA
  - Key observation: $h(t) + b(t)$ is monotonically non-decreasing in t



  - If $h(t) + b(t) > t$ then unschedulable
  - If $h(t) + b(t) = t$ move to next smaller deadline

[F. Zhang, A. Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling," IEEE Transactions on Computers, pages 1250-1258, September, 2009]

F. Zhang, A. Burns, A. “Schedulability Analysis of EDF Scheduled Embedded Real-Time Systems with Resource Sharing”. ACM Transactions on Embedded Computing Systems 9, 4, Article 39, March 2011]

Quick Processor demand Analysis: QPA for EDF

- QPA Algorithm

$$
\begin{array}{ll}
1 & t = \max\{d_i \mid d_i \le L\} \\
2 & \text{while}(\ h(t)+b(t) \le t \wedge h(t)+b(t) > DJ_{\min}\ )\ \{ \\
3 & \qquad \text{if}(\ h(t)+b(t) < t\ ) \\
4 & \qquad\qquad \{\ t = h(t)+b(t)\ \ \} \\
5 & \qquad \text{else} \\
6 & \qquad\qquad \{\ t = \max\{d_i \mid d_i < t\}\ \ \} \\
7 & \} \\
8 & \text{if}(\ h(t)+b(t) \le DJ_{\min}\ ) \Rightarrow \text{ task set is schedulable} \\
9 & \text{else} \Rightarrow \text{ task set is unschedulable}
\end{array}
$$

  - Simple & effective
  - Complexity of Processor Demand Bound test is exponential for U =1 otherwise pseudo-polynomial
  - QPA gives a very large reduction in the number of points evaluated (exponentially so in practice) easily copes with 100s or 1000s of tasks

[F. Zhang, A. Burns, A. “Schedulability Analysis of EDF Scheduled Embedded Real-Time Systems with

Schedulability Analysis for Non-pre-emptive EDF

- Same approach as for pre-emptive EDF
  - Account for non-pre-emptive execution via different blocking term

$$b(t) = \max_{\forall i: D_i - J_i > t} (C_i - 1)$$

  - Same fundamental equations

$$\forall t \quad h(t) + b(t) \leq t$$

  - Can again use QPA to provide a quick test

[L. George, N. Rivierre, M. Spuri, “Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling”, INRIA Research Report, No. 2966, September 1996]

Theorectical Comparison between EDF and FP

- Optimality
  - Pre-emptive EDF is an optimal uniprocessor scheduling policy
  - Non-pre-emptive EDF is weakly optimal (optimal among all work-conserving non-pre-emptive policies)

- How much better is EDF than FP?
  - Utilisation bound for implicit deadline tasksets

    FP: $U \leq n(\sqrt[n]{2} - 1) \rightarrow \ln(2) \approx 0.693$
    EDF: $U \leq 1$

  - So processor would need to be $1/\ln(2) \approx 1.44$ times faster to guarantee that any implicit deadline taskset schedulable by EDF could be scheduled using FP

[C.L. Liu, J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1) pages 46-61, 1973]

FP v EDF Speedup factors

Speedup factor: increase in processing speed required so that any feasible taskset (schedulable by an optimal algorithm) can be scheduled using Fixed Priority scheduling

Taskset Constraints [Priority ordering]	**FP-P Speedup factor**		**FP-NP Speedup factor**	
	Lower bound	Upper bound	Lower bound	Upper bound
Implicit-deadline [RM] [OPA]	**$1/\ln(2) \approx 1.44269$**		**$1/\Omega \approx 1.76322$**	**2**
Constrained-deadline [DM] [OPA]	**$1/\Omega \approx 1.76322$**		**$1/\Omega \approx 1.76322$**	**2**
Arbitrary-deadline [OPA] [OPA]	**$1/\Omega \approx 1.76322$**	**2**	**$1/\Omega \approx 1.76322$**	**2**

[R.I. Davis, T. Rothvoß, S.K. Baruah, A. Burns, “Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling.” Real-Time Systems, Volume 43, Number 3, pages 211-258, November 2009]

In practice FP vs EDF

- FP used in most commercial RTOS
- Why if EDF is better?
  - FP simpler & faster implementation
    - Bit masks for priorities, in practice don't need that many priority levels to get good schedulability: FP + FIFO works
  - FP vs EDF schedulability is the same for harmonic task periods
  - FP More predictable under overload
    - EDF have a cascade of deadline misses (all tasks), FP only task of lower priority than the over-running job tend to miss deadlines
  - Standards
    - E.g. OSEK and Autosar OS standards – specify FP scheduling
  - Supporting theory
    - FP scheduling theory more mature than EDF, but EDF catching up
  - Inertia
    - Continuing with what has previously been done

Summing up

- Learnt about uniprocessor scheduling
  - Fixed Priority and EDF scheduling

  - Resource locking protocols (Stack Resource Policy)

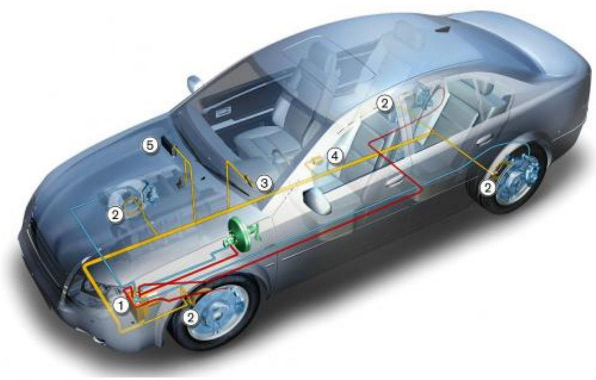
  - Fundamentals of Schedulability analysis
    - Pre-emptive and non-pre-emptive cases
    - Some extensions: blocking, release jitter, arbitrary deadlines

  - Priority assignment
    - Deadline Monotonic
    - Optimal Priority Assignment
    - Robust Priority Assignment

Success Stories
Fixed Priority Scheduling Theory

- Controller Area Network (CAN)
  - Analysis enables bus utilisation of up to ~70-80% compared to ~30% before
  - Involved in a start-up company NRTA that developed **Volcano** for Volvo in mid-1990s
  - Technology now owned and marketed by Mentor Graphics
  - Influenced CAN device driver HW design (MSCAN)
  - Used in millions of cars: Volvo, LandRover, Jaguar, Aston-Martin, Mazda, SAIC (China)

- Classical theory still needs adapting to HW behaviours and SW engineering practice
  - Non-abortable TX buffers
  - FIFO queues
  - Multiple levels of FIFO and priority queues



Success Stories
Fixed Priority Scheduling Theory

- RTOS
  - Involved in a start-up company LiveDevices that developed an OSEK RTOS (1997-2003) RTA-OSEK
  - RTOS was designed to comply with scheduling theory
  - Took advantage of FP+SRP scheduling to permit single stack operation saving memory (v. important for small microcontrollers)
  - RTOS analysable with minimal overheads
  - Supported by schedulability analysis tools
  - Company sold in 2003 to ETAS (part of Bosch)

ETAS

- Since then
  - RTA-OS (Autosar) extension, and RTA-OSEK
  - RTOS deployments running at approx. 50 million ECUs per year...

Hot research topics

- Integration of WCET and Schedulability analysis
  - CRPD and Limited Pre-emption
- Mixed Criticality Systems
  - Scheduling techniques and analysis for systems with applications at different criticality levels (different views of WCET)
- Probabilistic Real-Time Systems
  - Providing assurance that the probability of a deadline being missed is below some required threshold e.g. 10^{-9} failures per hour
  - Randomised architectures

Finally... a 20 year old Open Problem

- Dual Priority scheduling
  - Each task has two priorities
  - A fixed time S_i after task τ_i is released, its priority is promoted to the higher of its two priorities

- Hypothesis
  - Utilisation bound for Dual Priority scheduling of implicit deadline sporadic tasks is 100%
  - Proved for $n = 2$
  - No counter examples found so far for n > 2

- Problem: Choosing priorities and promotion times

[A. Burns, "Dual Priority Scheduling: Is the Processor Utilisation bound 100%" In proceedings RTSOPS, 2010.]

Questions?