

# **Real-Time Systems**



# Lecture Content NICTA

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

© NICTA 2007/2008

#### 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!

© NICTA 2007/2008

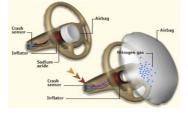
No: 3

## Real-Time Systems

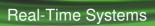




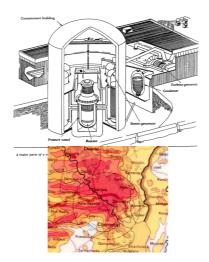




© NICTA 2007/2008











© NICTA 2007/2008

No: 5

# Real-Time Systems



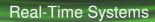








© NICTA 2007/2008













© NICTA 2007/2008

No. 7

# 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

© NICTA 2007/2008

#### 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)

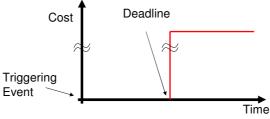
© NICTA 2007/2008

No. 9

### 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.

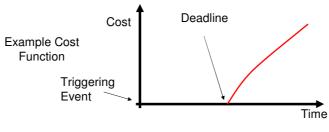


No: 10

© NICTA 2007/2008



- · 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)



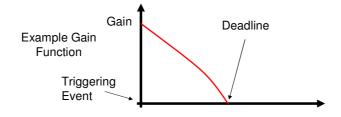
© NICTA 2007/2008

No: 11

## 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.



© NICTA 2007/2008

### 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.

© NICTA 2007/2008

No: 13

## 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.

© NICTA 2007/2008

## 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.

© NICTA 2007/2008

No: 15

#### Requirement, Specification, Verification



- 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, therac-25 example

© NICTA 2007/2008



## Scheduling in Real-Time Systems

© NICTA 2007/2008

No: 17

# Overview

- Specification and religious believes
- Preemptive vs. non preemptive scheduling
- · Scheduling algorithms
- Message based synchronisation and communication
- · Overload situations
- Blocking and Priority Inversion

© NICTA 2007/2008

#### 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

© NICTA 2007/2008

No: 19

## 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

© NICTA 2007/2008

# Task Model

#### · Periodic tasks

- Time-driven. Characteristics are known a priori
- Task τ<sub>i</sub> is characterized by (T<sub>i</sub>, C<sub>i</sub>)
- E.g.: Task monitoring temperature of a patient in an ICU.

#### · Aperiodic tasks

- Event-driven. Characteristics are not known a priori
- Task  $\tau_i$  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  $(T_{\scriptscriptstyle i},\,C_{\scriptscriptstyle i})$ 

© NICTA 2007/2008

No: 21

# Task Model

- C<sub>i</sub>= Computation time (usually Worst-Case Execution Time, WCET)
- D<sub>i</sub>= Deadline
- T<sub>i</sub> = Period or minimum interarrival time
- J<sub>i</sub> = Release jitter
- $P_i$  = Priority
- B<sub>i</sub> = Worst case blocking time
- R<sub>i</sub>= Worst case response time

© NICTA 2007/2008

#### **Task Constraints**



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

© NICTA 2007/2008

No: 23

#### 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

© NICTA 2007/2008

## Preemption



- Cooperative preemption?
  - Applications allow preemption at given points
  - Reduction of preemptions
  - Increase of latency for high priority tasks

© NICTA 2007/2008

No: 25

## **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 AFTI-F16 flight tests

© NICTA 2007/2008

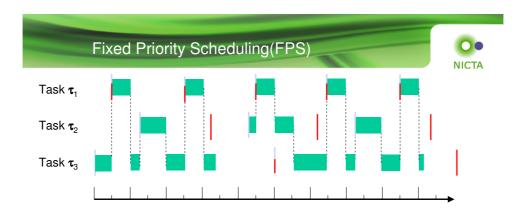
## Fixed Priority Scheduling



- · Priorities may assigned by
  - Deadline: shortest deadline ⇒ highest priority
  - Period: shortest period ⇒ 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)

© NICTA 2007/2008

No: 27



	Priority	С	Т	D
Task τ <sub>1</sub>	1	5	20	20
Task τ <sub>2</sub>	2	8	30	20
Task τ <sub>3</sub>	3	15	50	50

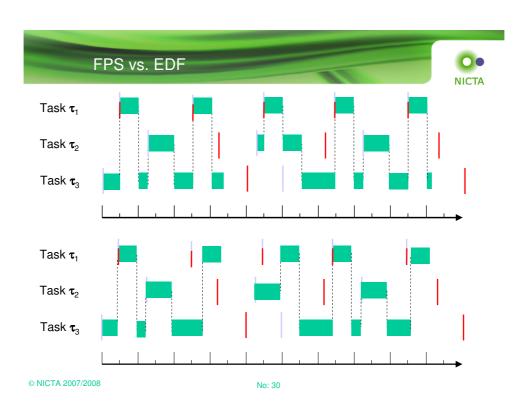
© NICTA 2007/2008

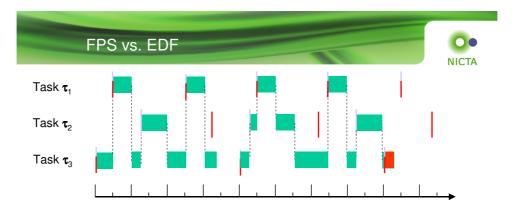
# Earliest Deadline First (EDF)



- · 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

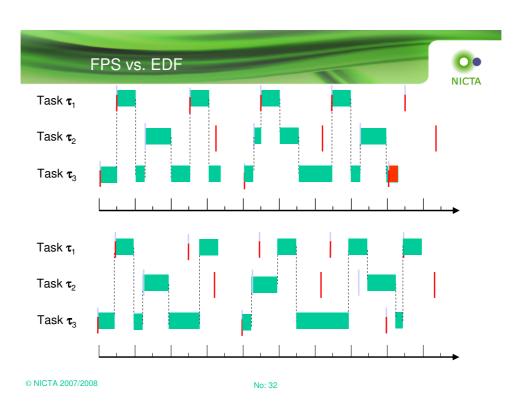
© NICTA 2007/2008





	Priority	С	Т	D
Task τ <sub>1</sub>	1	5	20	20
Task τ <sub>2</sub>	2	8	30	20
Task τ <sub>3</sub>	3	15	40	40

© NICTA 2007/2008



## 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

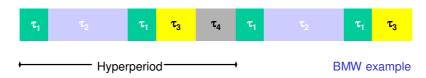
© NICTA 2007/2008

No: 33

## 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

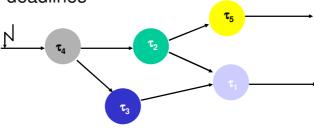


© NICTA 2007/2008

## 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



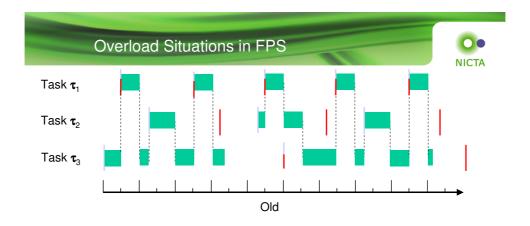
Caused by faulty components of the system

No: 35

- 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

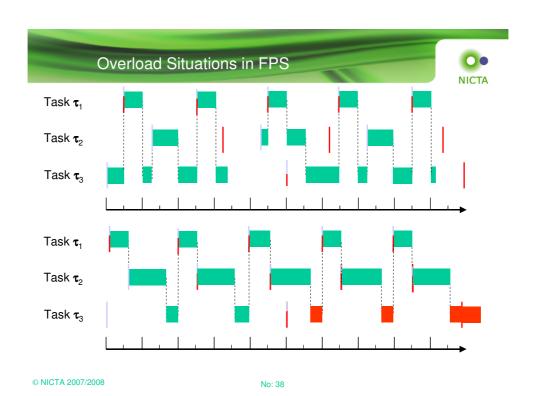
© NICTA 2007/2008

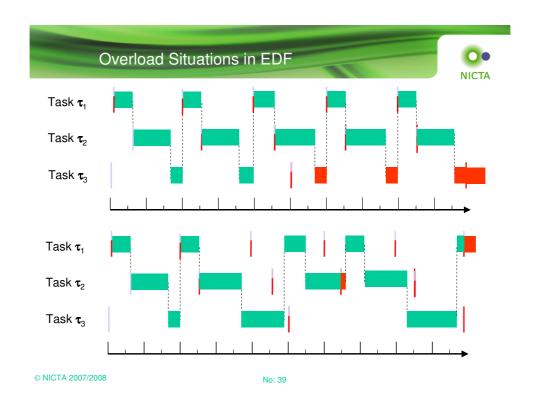
© NICTA 2007/2008

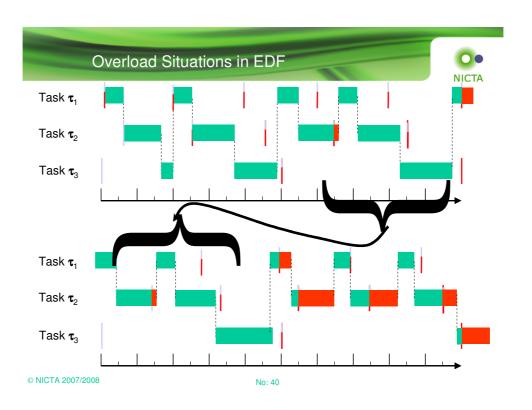


	Priority	С	Т	D
Task τ <sub>1</sub>	1	5	20	20
Task τ <sub>2</sub>	2	12	20	20
Task τ <sub>3</sub>	3	15	50	50

© NICTA 2007/2008 No: 37



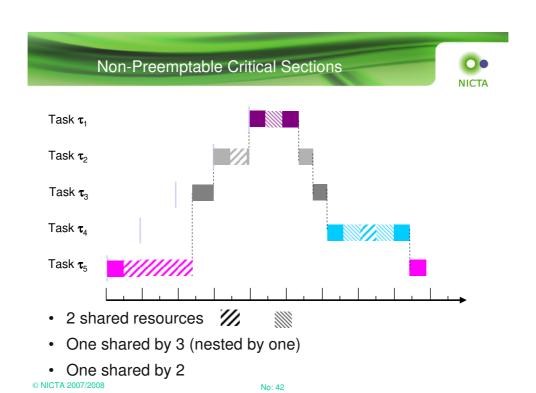




# Priority Inversion

- 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.
- Pathfinder example

© NICTA 2007/2008 No: 41



# Non-Preemptable Critical Section



#### GOOD

- Simple
- No deadlock.
- No unbounded priority inversion
- No prior knowledge about resources.
- Each task blocked by at most 1 task of lower priority
- Works with fixed and dynamic priorities. (especially good for short critical sections with high contention)

#### BAD

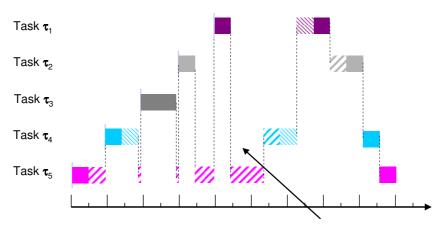
- Tasks blocked even when no contention exists.

© NICTA 2007/2008

No: 43

# Priority Inheritance





Note the indirect inheritance

© NICTA 2007/2008

## Priority Inheritance



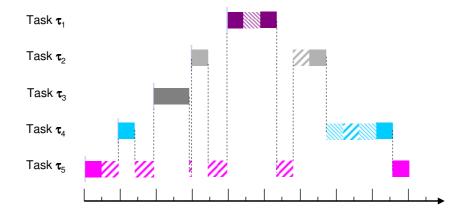
- When lower priority job blocks, it inherits priority of blocked job.
- GOOD
  - No unbounded priority inversion
  - Simple
  - No prior knowledge required
  - Works with fixed and dynamic priorities.
- BAD
  - Possible Deadlock.
  - Blocking of jobs not in resource contention.
  - Blocking time could be better
  - Indirection a pain in the neck

© NICTA 2007/2008

No: 45

# Basic Priority Ceiling Protocol





© NICTA 2007/2008

## Basic Priority Ceiling Protocol



- · Lower priority task inherits priority of blocked task.
- Task may be denied resource even when available.
- Also known as Original Priority Ceiling Protocoll (OPCP)
- GOOD
  - No deadlock.
  - No unbounded priority inversion.
  - Blocking time reduced.
- BAD
  - Task may be denied resource even when available.
  - Need a priori knowledge of use of resources.

$$B_i = \max_{k=1}^{K} usage(k,i)C(k)$$

 $B_i = \sum_{k=1}^{K} usage(k, i)C(k)$ 

**Basic Priority Ceiling** 

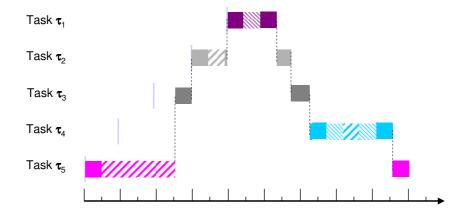
Priority Inheritance

© NICTA 2007/2008

No: 47

## Immediate Priority Ceiling Protocol





© NICTA 2007/2008

#### Immediate Priority Ceiling Protocol



- Lower priority task inherits priority of potentially blocked task. Task may be denied resource even when available.
- GOOD
  - Simple.
  - Shared run-time stack.
  - Reduced Context-Switching
  - No deadlock.
  - No unbounded priority inversion.
- BAD
  - Task may be denied resource even when available
  - Task may be affected by blocking effect without using any resources
  - Need a priori knowledge of use of resources.
  - No self suspension while holding a resource

© NICTA 2007/2008

No: 49

### Implementation Comparison



- · Non-preemptable critical sections
  - Easy to implement. Either blocking interrupts or syscall to have that implemented on behalf of task
- Priority Inheritance
  - Fairly straightforward, however requires various references (e.g. which thread is holding a resource)
- · Basic Priority Ceiling
  - Requires application designer to explicitly identify which resources will be requested later (when first resource request of nested requests is made) on top of references
- · Immediate priority ceiling
  - Very easy to implement: Only requires ceilings associated with each resource mutex (that's something which may be automated if all tasks known
  - Alternatively server task encapsulating the critical section

© NICTA 2007/2008

### Reflective/Feedback-based Scheduling



- · Adaptive systems
- By definition soft real time
- Adjusts scheduling based on information about change
- Capable of better coping with "the unknown"
- Connects quite well with adaptive applications

© NICTA 2007/2008

No: 51







Schedulability Analysis of Real-Time Systems

© NICTA 2007/2008

### 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.

© NICTA 2007/2008

No: 53

#### Critical Instant



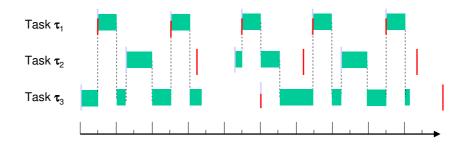
- Trivial for independent tasks
  - All events happen at the same time;
  - However, implicitly consider all possible phases (take nothing for granted).
- However, get's more tricky (but tighter) having dependencies
  - What phasing of other activities produces the biggest load.
  - An activity is a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

© NICTA 2007/2008

# Response Time Analysis



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



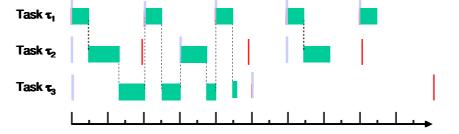
© NICTA 2007/2008

No: 55

## Response Time Analysis



	Priority	С	Т	D
Task τ <sub>1</sub>	1	5	20	20
Task τ <sub>2</sub>	2	8	30	20
Task τ <sub>3</sub>	3	15	50	50



© NICTA 2007/2008

#### Formal RTA



- Assumptions j<i ⇒priority j is lower than</li> priority i
- · Critical instant
- Iterative process  $w_i^0 = C_i$

$$w_i^0 = C_i$$

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

© NICTA 2007/2008

No: 57

## **Blocking Time and Other Nasties**



- · Blocking time
- Jitter
- · Pre-emption delay

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

© NICTA 2007/2008

### 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<sub>i</sub> is known and constant
  - Time required for context switching is known

© NICTA 2007/2008

No: 59

## Rate Monotonic Analysis contd

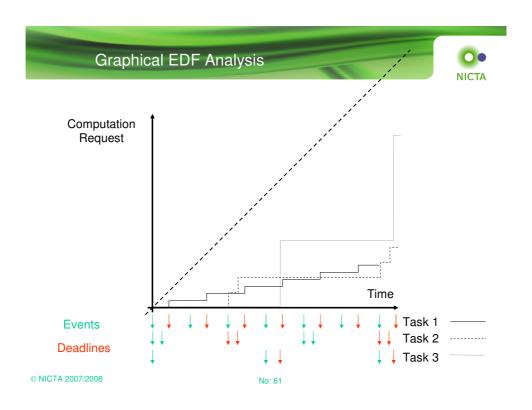


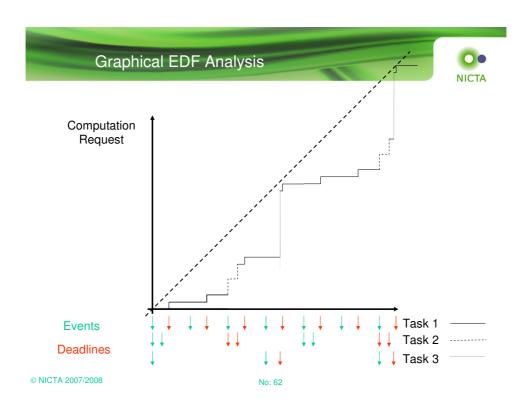
· Bound is given by:

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

- 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

© NICTA 2007/2008







### Worst Case Execution Time Analysis

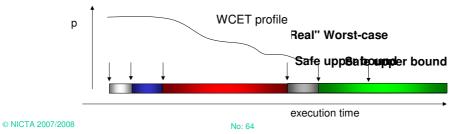
© NICTA 2007/2008

No: 63

# Problem Definition

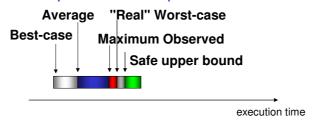


- All of the scheduling analysis presented previously requires the Worst-Case Execution time to be known
- · Target is to come up with
  - a safe upper bound
  - as close as possible to the "real" worst case.
  - Ideally with more than just single number (probabilistic analysis)

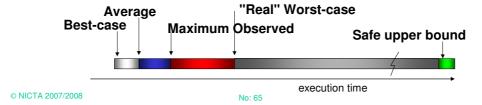




#### Simple Code + Simple Processors



#### Complex Code + Advanced processors



## Is it a Problem?



- Safety critical computer systems exist and are deployed
- Yes, but ...
  - Safety critical systems have been
    - highly restrictive in terms of HW/SW used
    - · highly restrictive in terms of complexity
    - used a lot of manual inspection and pen and paper work

© NICTA 2007/2008

# Is it a Problem? contd

- The stuff in the last slide doesn't scale!
- industry not in the safety critical arena have been using measurements with safety factors.
  - Worked fine with simple architectures, but doesn't work good with more advanced computers
  - Excessive overestimation and underestimation with same factor for different programs, parameterisation doesn't help too much
- Large body of academic work, but little industrial uptake: YET

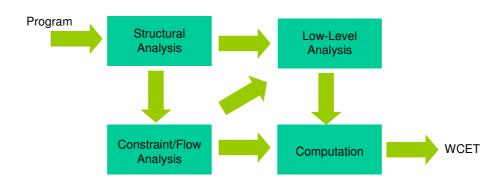
© NICTA 2007/2008

No: 67

## Generic Problem Partitioning



 Some analysis methods integrate some aspects of these, but the general requirements are the same



© NICTA 2007/2008

## Structural Analysis



- · Can work on:
  - Source code and/or
  - Object code or
  - Assembly Code

© NICTA 2007/2008

No: 69

## Structural Analysis contd



- Object Code
  - Pros:
  - All compiler optimisations are done
  - This is what really is running, no trouble with macros, preprocessors
  - Cons:
  - Needs to second guess what all the variables meant
  - A lot of the original information is lost
    - E.g. multiple conditions, indirect function calls, object member functions

© NICTA 2007/2008

## Structural Analysis contd



# Assembly Code

- Pros:
- All compiler optimisations done
- Cons:
- Same as Object Code +
- Potentially still some macros

© NICTA 2007/2008

No: 71

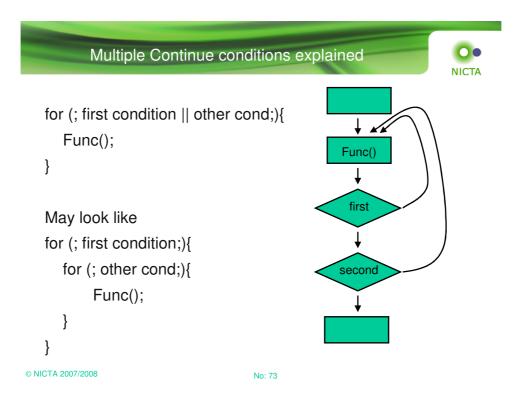
## Structural Analysis contd



#### Source Code

- Pros:
- All information the user has put there is there
- Structure in pure form (e.g. multiple loop continue conditions, object member functions, indirect calls)
- Cons:
- Trouble with macros, preprocessors etc.
- Needs to second guess what the compiler will do

© NICTA 2007/2008

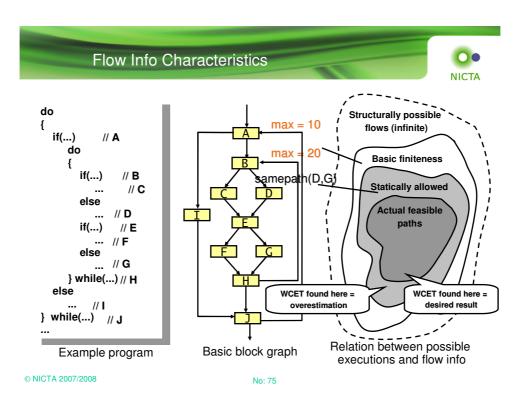


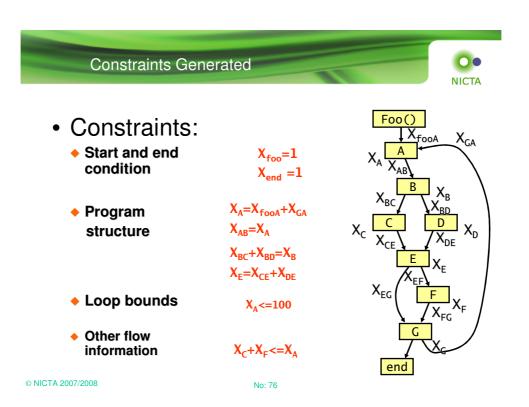
#### Flow information



- For low level analysis and computation we need to restrict flow to reasonable subset.
- This information can be gained:
  - By static analysis (most importantly abstract interpretation)
  - By observation (worst case?)
  - By user annotations

© NICTA 2007/2008







- WCET analysis requires a deep understanding of
  - hardware features of processors
  - Interaction of software and hardware

© NICTA 2007/2008

No: 77

# 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

© NICTA 2007/2008

#### 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?

© NICTA 2007/2008

No: 79

## Path Based Computation



- · Follows each individual paths
- Becomes quickly intractable for large applications
- Altenbernd and Co have tried a simplified approach:
  - Starting out from the entry point of a function follow a path in the CFG and annotate each node with the execution time up to this node
  - Do so with any other path, but whenever a join node is reached compare new value up to this point with annotated value

© NICTA 2007/2008

# Path Based Computation



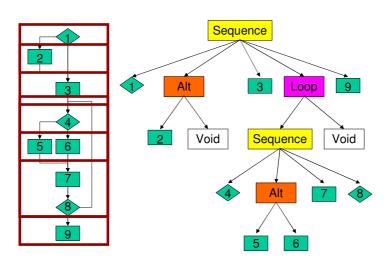
- Continue if new value is larger or not smaller than the the old value minus the smallest of the largest 5 overall execution times paths computed so far. (otherwise start next path)
- If overall path is larger than smallest of the largest 5 overall execution times, keep (remove the compared smallest of the largest 5 overall execution time paths.
- Check feasibility of 5 longest paths (path may actually happen)

© NICTA 2007/2008

No: 81

#### Tree Representation





© NICTA 2007/2008

#### **IPET Calculation**



# • WCET=

 $\max \Sigma(x_{\text{entity}} * t_{\text{entity}})$ 

Where each x<sub>entity</sub>
 satisfies all constraints

© NICTA 2007/2008

No: 83

#### Calculation methods



- Solution methods:
- Integer linear programming
  - Constraint satisfaction
- Solution:
  - Counts for each individual node and edge
  - The value of the WCET

ming  $X_B=100$   $X_C=100$   $X_D=0$   $X_C=100$   $X_C=100$   $X_C=100$   $X_C=100$   $X_C=100$   $X_C=100$   $X_C=100$   $X_C=100$ 

Foo()  $X_{foo}=1$ 

 $X_{\Delta}=100$ 

© NICTA 2007/2008



# Multiprocessor/Multithreaded Real-Time Systems

© NICTA 2007/2008

No: 85

# WHY

- Performance
  - Responsiveness in the presence of many external events
- Throughput
  - Managing continuous load
- Fault tolerance
  - Managing bugs, HW faults
- Reliability
  - Ensuring uptime, HW/SW upgrades ...

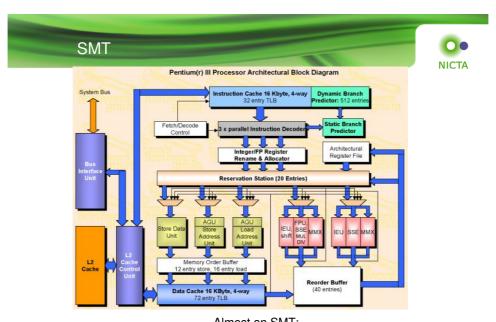
© NICTA 2007/2008



- Symmetric Multithreading (SMT)
  - Contention on execution units, caches, memory
- Symmetric Multiprocessor (SMP)
  - Contention on memory, cache coherency, eg NUMA
- Asymmetric Multiprocessor
  - Specialised units, coherency
- Distributed System
  - Latency in communication, loosely coupled

© NICTA 2007/2008

No: 87



Almost an SMT:

Image taken from http://www.tommesani.com/images/P3Architecture.jpg

© NICTA 2007/2008



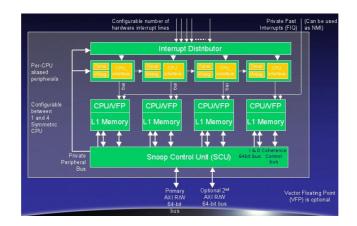
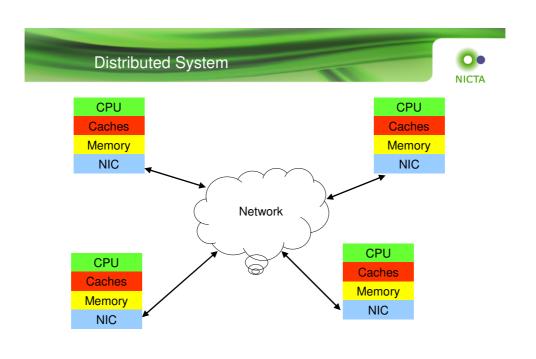


Image taken from http://linuxdevices.com/files/misc/arm-mpcore-architecture-big.jpg

© NICTA 2007/2008

No: 89



© NICTA 2007/2008



- Resource contention
  - Execution units
  - Caches
  - Memory
  - Network
- · Adding a CPU does not help
  - Example double the load, 2 instead of 1 CPU

© NICTA 2007/2008

No: 91

# Solutions??



- Resource contention still there!
- Assignment using heuristics
- Non partitioning
  - mostly theoretical so far
  - Assumptions:
    - · Zero preemption cost
    - · Zero migration cost
    - · Infinite time slicing
  - Don't translate into reality
  - Acceptance test and no task migration a way to make it work

© NICTA 2007/2008

# Solutions??

- Quite often non-preemptive
  - Fewer context switches
  - Reasoning is easy
    - IEEE Computer reference to insanity
  - Testing is easier??
  - Reduce need for blocking
- But!

© NICTA 2007/2008

No: 93

# Non-Preemptive



- But!!!
  - Less efficient processor use
  - Anomalies: response time can increase with
    - · Changing the priority list
    - · Increasing number of CPUs
    - · Reducing execution times
    - · Weakening the precedence constraints
  - Bin packing problem NP hard
  - Theoretically: time slicing into small quantums (PFAIR), but practically useless, as preemption and task migration overhead outweigh gains of Multiprocessors.

© NICTA 2007/2008



- · No global solution.
- Partitioning and reducing it to single CPU problem good, but still contention of resources.
- Next step: After figuring out how to do the scheduling, what about preemption delay?
- Industry works with SMP/SMT, but most often on a very ad hoc basis.
- Active and unsolved research area
- · Why does it work on non-RT?
  - Running the "wrong" task is not critical.

© NICTA 2007/2008

No: 95



# Integrating Real-Time and General-Purpose Computing

Many thanks to: Scott A. Brandt University of California, Santa Cruz

© NICTA 2007/2008

#### Real-Time vs. General-Purpose OS



- Real-time and general-purpose operating systems implement many of the same basic operations
  - Process mgmt., memory mgmt, I/O mgmt, etc.
- They aim for fundamentally different goals
  - Real-time: Guaranteed performance, timeliness, reliability
  - General-purpose: Responsiveness, fairness, flexibility, graceful degradation, rich feature set
- They have largely evolved separately
  - Real-time system design lags general-purpose system design by decades
- They need to merge

© NICTA 2007/2008

No: 97

# Why?



- We want both flexible general-purpose processing and robust realtime processing
  - Multimedia is ubiquitous in general-purpose systems
  - Real-time systems are growing in size and complexity
- Such systems are possible
  - Look at the popularity of RTLinux
  - GP hardware has grown powerful enough to support traditional hard realtime tasks (multimedia, soft modems, etc.)
  - Windows, MacOS, etc., are already headed in this direction
- Existing solutions are ad hoc
  - RTLinux, MacOS, Windows?
- The world is already headed that way
  - Microsoft, HP, Intel, Dell all want to develop integrated home systems
  - Complex distributed real-time systems do more than hard real-time
- We need to get out in front and lead the way

© NICTA 2007/2008

#### How?



- We need integrated solutions for each type of resource
  - CPU, storage, memory, network, ...
- They must be hard real-time at their core
  - This is the only way to guarantee the hardest constraints
- They must provide native hard real-time, soft real-time, and best-effort support
  - SRT and BE support cannot be added as an afterthought
  - Neither can HRT
- We need an overall model for managing the separate resources
  - Each process must be able to specify it's per-resource constraints
  - Defaults should be reasonable, and helpful

© NICTA 2007/2008

No: 99

## Kinds of Timeliness Requirements



# • Hard Real-Time (HRT) [e.g. flight control]



- Rate-Based (RB) [e.g. desktop audio]
  - Continuous processing requirements
- Soft Real-Time (SRT) [e.g. desktop video]
  - Non-critical deadlines and/or variable processing needs, worst-case, average-case, or no estimates
- Best Effort (BE) [e.g. editor or compiler]
  - Undefined timeliness requirements

- We want to run processes with different timeliness requirements in the same system
  - HRT, RB, SRT, and BE
- Existing schedulers largely provide point solutions:
  - HRT or RB or one flavor of SRT or BE
- Hierarchical scheduling is a partial solution
  - Allows apps with a variety of timeliness requirements, BUT
  - Static, inflexible hierarchies
- Goal: Uniform, fully dynamic integrated real-time scheduling
  - Same scheduler for all types of applications

© NICTA 2007/2008

Unconstrained

#### Separate Resource Allocation and Dispatching



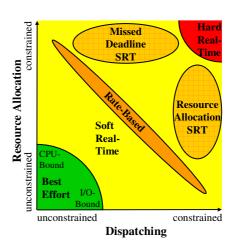
Observation: Scheduling consists of two distinct questions:

#### Resource allocation

How much resources to allocate to each process

#### Dispatching

- When to give each process the resources it has been allocated
- Existing schedulers integrate their management
  - Real-time schedulers implicitly separate them somewhat via job admission



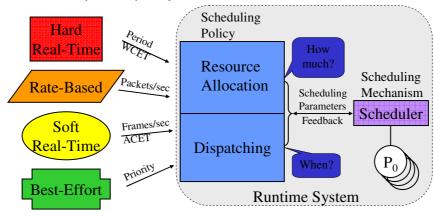
© NICTA 2007/2008

No: 101

#### The (RAD) Scheduling Model



- Separate management of Resource Allocation and Dispatching
  - and separate policy and mechanism



© NICTA 2007/2008 No: 102

#### Rate-Based Earliest Deadline Scheduler



#### · Basic Idea

- EDF provides hard guarantees
- Varying rates and periods provide flexibility
- Programmable timer interrupts guarantee isolation between processes

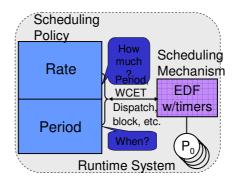
## RBED policy

- Resource allocation: Target rateof-progress for each process (S ≤ 100%)
- Dispatching: Period based on process timeliness needs

#### RBED mechanism

Rate-Enforcing EDF: EDF + programmable timer interrupts

RBED: RAD Scheduler using rate and period to control resource allocation and dispatching



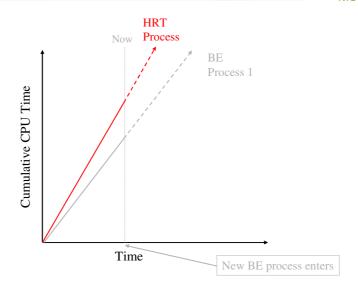
rate = utilization WCET = rate\*period

© NICTA 2007/2008

No: 103

# Adjusting Rates at Runtime

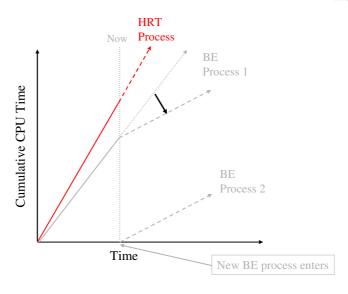




© NICTA 2007/2008

# Adjusting Rates at Runtime





© NICTA 2007/2008

No: 105

#### **RBED Periodic Task Model**



#### **EDF**

- Period and WCET are specified per task
  - $T_i$  has sequential jobs  $J_{i,k}$
  - $\begin{array}{ll} & J_{i,k} \text{ has release time } r_{i,k}, \\ & \text{period } \boldsymbol{p}_i, \text{ deadline } d_{i,k} \end{array}$
  - $r_{i,k} = d_{i,k-1}$ , and  $d_{i,k} = r_{i,k} + p_i$
  - $-\mathbf{u}_{i} = \mathbf{e}_{i}/\mathbf{p}_{i}$  and  $U = \Sigma \mathbf{u}_{i}$

#### **RBED**

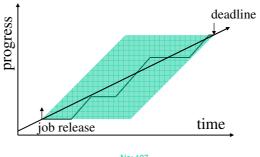
- Period and WCET are specified per job
  - T<sub>i</sub> has sequential jobs J<sub>i,k</sub>
  - $\ J_{i,k} \ has \ release \ time \ r_{i,k}, \\ period \ \textbf{p}_{i,k}, \ deadline \ d_{i,k}$
  - $r_{i,k} = d_{i,k-1}$ , and  $d_{i,k} = r_{i,k} + p_{i,k}$
  - $\mathbf{u}_{i,k} = \mathbf{e}_{i,k}/\mathbf{p}_{i,k}$  and  $U = \Sigma \mathbf{u}_{i,k}$
- Theorem 1: EDF is optimal under the new task model
  - Corollary: A new task may enter the system at any time, as long as resources are available for it

© NICTA 2007/2008

## Two Observations



- At deadlines, a task's actual resource allocation is equal to its target resource allocation
- · Actual resource allocation is bounded to the feasible region



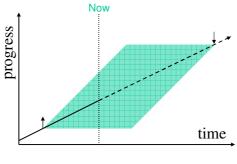
© NICTA 2007/2008

No: 107

#### Increasing Rate (= increasing WCET)



- Theorem 2: The resource usage of any task can be increased at any time, within the available resources
  - Given a feasible EDF schedule, at any time task  $T_i$  may increase utilization by any amount up to 1-U without causing any task to miss deadlines in the resulting EDF schedule

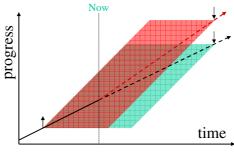


© NICTA 2007/2008

#### Increasing Rate (= increasing WCET)



- Theorem 2: The resource usage of any task can be increased at any time, within the available resources
  - Given a feasible EDF schedule, at any time task  $T_i$  may increase utilization by any amount up to 1-U without causing any task to miss deadlines in the resulting EDF schedule



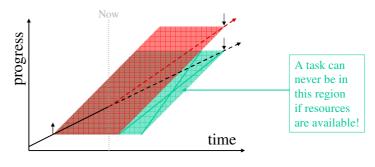
© NICTA 2007/2008

No: 109

#### Increasing Rate (= increasing WCET)



- Theorem 2: The resource usage of any task can be increased at any time, within the available resources
  - Given a feasible EDF schedule, at any time task  $T_i$  may increase utilization by any amount up to 1-U without causing any task to miss deadlines in the resulting EDF schedule

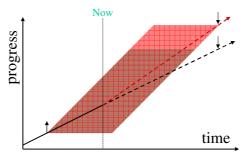


© NICTA 2007/2008

#### Increasing Rate (= increasing WCET)



- Theorem 2: The resource usage of any task can be increased at any time, within the available resources
  - Given a feasible EDF schedule, at any time task *Ti* may increase utilization by any amount up to 1–U without causing any task to miss deadlines in the resulting EDF schedule



© NICTA 2007/2008

No: 111

#### **RBED EDF Mode Change Theory**



- Theorem 1: EDF is optimal under this task model
- Corollary: A new task may enter at any time, within available resources
- Theorem 2: The rate of any task can be increased at any time, within available resources
- Theorem 3: The period of any task can be increased at any time
- Theorem 4: The rate of any task can be lowered at any time, down to what it has already used in the current period
- Theorem 5: The period of any task can be reduced at any time, down to the time corresponding to the current period's resource usage
- Corollary: The period of any task can be increased at any time (without changing WCET)
- Corollary: The period of a job which is ahead of its target allocation can be reduced at any time, down to the time corresponding to its current resource usage (without changing WCET) as long as the resources are available for the rate change

© NICTA 2007/2008

## **RBED Theory Summary**



- Rate and period can be changed without causing missed deadlines
  - At deadlines, rate and period changes are unconstrained (except by available resources)
  - In between, decreases are constrained by resource usage in the current period
  - The changes may be combined
- · Isolation between processes is guaranteed

© NICTA 2007/2008

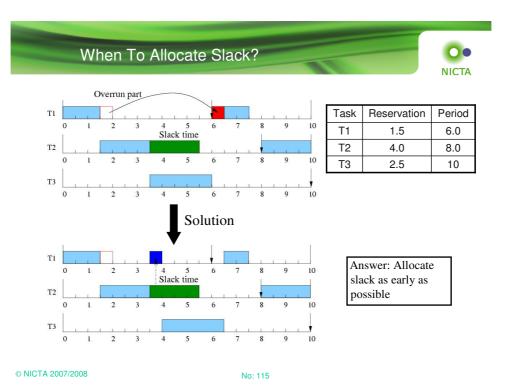
No: 113

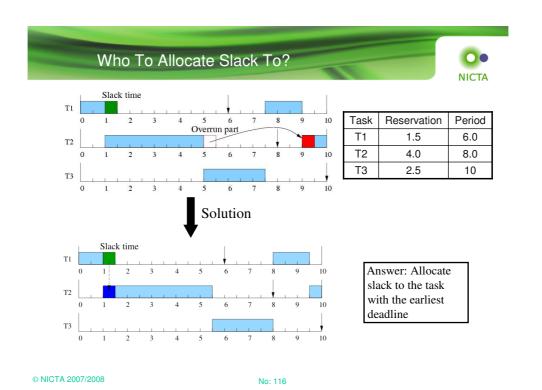
#### Better Slack Management: BACKSLASH

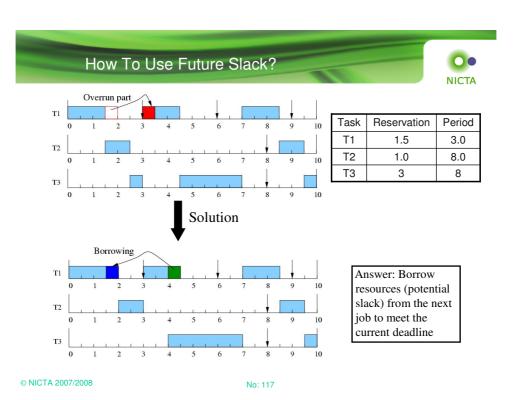


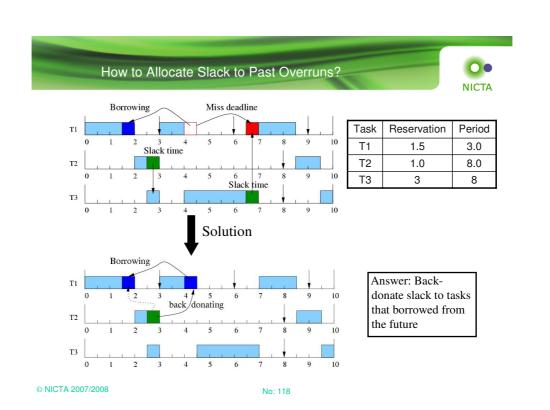
- Existing algorithms tend to ignore the needs of "background" tasks
  - Slack provided when everything else is idle
  - Aim for "fair" allocation and 100% utilization
- Slack reclamation is critical in an integrated real-time system
  - Utilization is important for best-effort systems
  - Soft real-time and best effort performance depends on the effective use of slack
- BACKSLASH improves performance via slack scheduling
  - Focuses on when slack is allocated, and to which process

© NICTA 2007/2008









#### **Principles**



- 1. Allocate slack as early as possible
  - With the priority of the donating task
- Allocate slack to the task with highest priority (earliest original deadline)
  - Task deadline, not server deadline
- Allow tasks to borrow against their own future resource reservations to complete their current job
  - With the priority of the donating job
- 4. Retroactively allocate slack to tasks that have borrowed from their current budget to complete a previous job

**SRAND** 



**SLAD** 



**SLASH** 



BACK SLASH

© NICTA 2007/2008

No: 119

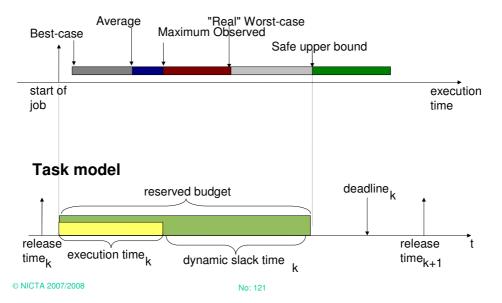
#### **BACKSLASH Conclusions**

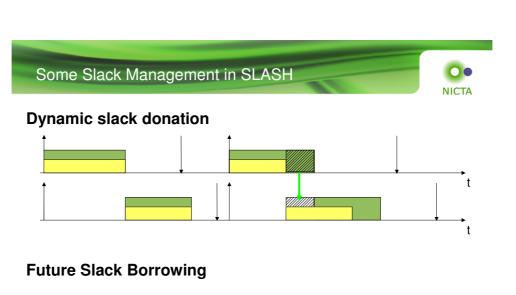


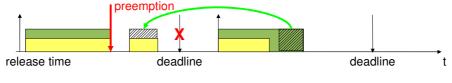
- In an integrated system supporting HRT, SRT and BE, the performance of SRT (and BE) depends on the effective reclamation and distribution of slack
- Four principles for effective slack reclamation and distribution:
  - 1. Distribute slack as early as possible
  - 2. Give slack to the ready task with the highest priority
  - 3. Allow tasks to borrow against future reservations
  - 4. Retroactively give slack to tasks that needed it
  - 5. SMASH: Conserve slack across idle times!
- Our results show that these principles are effective: BACKSLASH significantly outperforms the other algorithms and improves SRT (and/or BE) performance

© NICTA 2007/2008

# Bandwidth Enforcement in RBED and Power Management



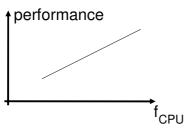




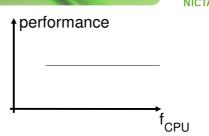
© NICTA 2007/2008

# **Modelling Time**





CPU bound application



Memory bound application

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{bus}}{f_{bus}} + \dots$$

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{bus}}{f_{bus}} + \dots$$

$$C_{mem} = \alpha_1 PMC_1 + \alpha_2 PMC_2 + \dots$$

$$C_{bus} = \beta_1 PMC_1 + \beta_2 PMC_2 + \dots$$

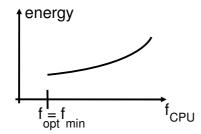
$$C_{cpu} = C_{tot} - C_{mem} - C_{bus} - \dots$$

© NICTA 2007/2008

No: 123

# Modelling Energy



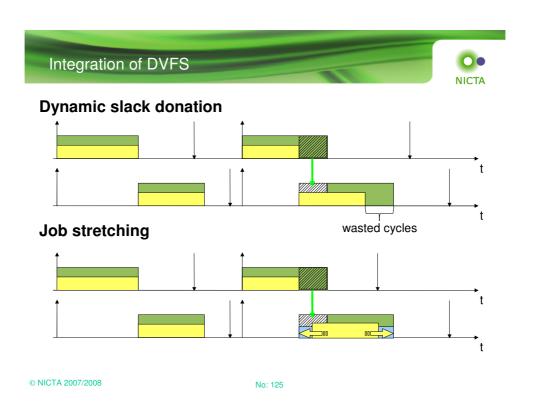


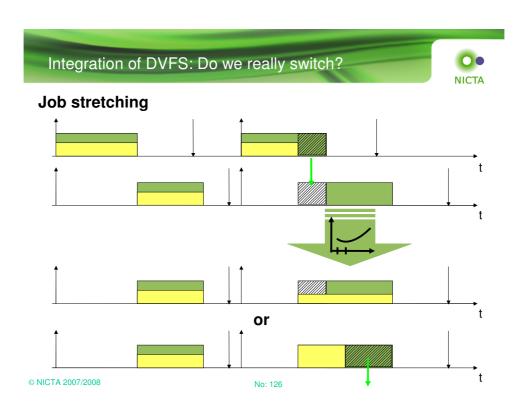
$$E_{tot} = E_{stat} + E_{dyn}$$
 
$$E_{tot} = P_{stat}T + \int_{0}^{T} P_{dyn}dt$$
 
$$E_{dyn} \propto fV^{2}\Delta t \propto cyclesV^{2}$$

energy f<sub>CPU</sub> f < f min opt

$$E_{dyn} = V^{2} (\chi_{cpu} f_{cpu} + \chi_{bus} f_{bus} + ...) \Delta t$$
$$+ \chi_{mem} f_{mem} \Delta t$$
$$+ \gamma_{1} PMC_{1} + \gamma_{2} PMC_{2} + ...$$
$$+ V^{2} (\phi_{1} PMC_{1} + \phi_{2} PMC_{2} + ...)$$

© NICTA 2007/2008



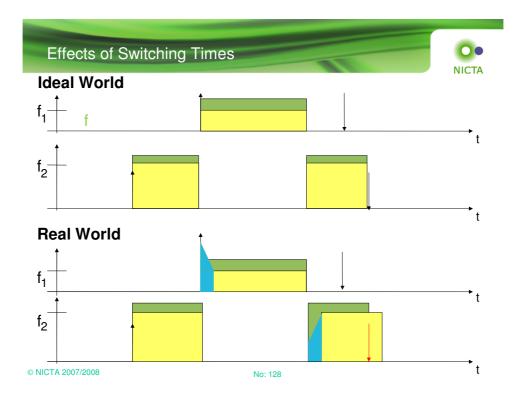




- · Switch to another frequency setting if
  - Job can finish on time in the frequency setting (inclusive switching cost)
  - System energy will be minimised

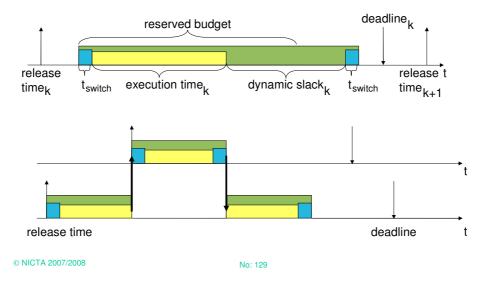
newEnergy = energyAtCurrentFrequency
newFrequency = currentFrequency
for frequency in frequencySetPoints
if WCETAtSwitchedFrequency + switching.WCET < remainingBudet
 && switching.Energy + energyAtSwitchedFrequency < newEnergy
 newEnergy = switchingCost.Energy + energyAtSwitchedFrequency;
 newFrequency = frequency;
if newFrequency != currentFrequency
 switchFrequency (newFrequency);</pre>

© NICTA 2007/2008 No:





#### New task model



# 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://ebook.ieeelab.com/Embedded/RTSbook.pdf

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.

© NICTA 2007/2008

#### Basic Priority Ceiling Protocol



- · Scheduling:
  - Highest priority task in ready queue gets scheduled. Priority exceptions as below
- Each resource has a ceiling priority equivalent of highest priority using task
- Allocation
  - If resource locked, block
  - If (potentially modified) priority of task higher than the ceiling of any resource not used by that task but used at the time, allocate
  - Else, block
- Priorities:
  - If task τ<sub>1</sub> blocked at resource held by task τ<sub>2</sub>:
    - $\tau_2$  is lifted in priority to task  $\tau_1$
- revert to original priority once all resources are released

  No: 131

## Basic Priority Ceiling and Deadlock



- At any time the priority of task τ<sub>i</sub> > ceiling priority of resource currently in use THEN
  - 1. task  $\tau_{\text{I}}$  will not require any resource currently in use
  - 2. Any task  $\tau_k$  with priority greater than task  $\tau_l$  will not require any resource currently in use
  - i.e.:
  - No task currently holding a resource can inherit a higher priority and preempt task  $\tau_{\text{I}}$  w

© NICTA 2007/2008