Virtualisation for Embedded Real-Time Systems

Robert Kaiser

robert.kaiser@sysgo.com

SYSGO AG

Wiesbaden University of Applied Sciences

Germany

Overview

- Motivation
- Existing virtualisation systems
- Workload classes
- Impact of virtualisation on timing behaviour
- Requirements
- Approach
- Summary/outlook

Motivation

- Increasing performance of embedded systems
- $\Rightarrow$ Increasing software complexity
- Classical embedded OSes not up to the challenge:
  - Single address/name space
  - Single (often proprietary) OS interfaces
- $\Rightarrow$ Future embedded systems need:
  - fault containment
  - multiple OS interfaces
- Virtualisation: successful response in server market
- **But:** current VM implementations not suitable for embedded systems, especially wrt. real-time issues

Existing virtualisation systems

- Virtualisation: invented in the mid 1960's by IBM
- Current main protagonists: VMware and Xen
- Both are clearly not designed for real-time use:
  - Proportional share assumption
  - No way to establish a strictly time-driven schedule
  - No real-time OS interfaces available (Xen)
- Xen: possibility to exchange VM scheduler

Current virtual machine implementations are of limited use for real-time purposes.

Workload classes

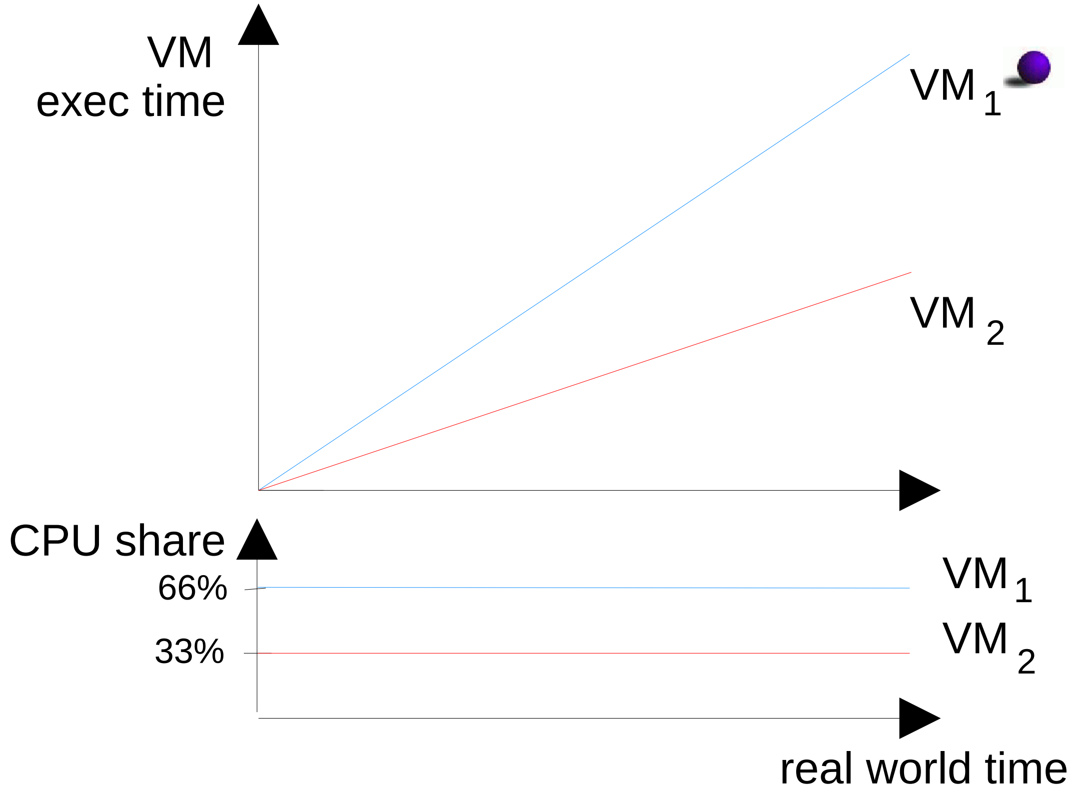
Complex embedded system: must be prepared to handle a mixture of applications with diverse timing requirements:

- Real-time: Must [a] or should [b] meet deadlines. Two subclasses:
  - Time triggered: static schedule, typically periodic
  - Event triggered: processes arrive in response to (unpredictable) events. Assumed to be sporadic
- Non-real-time: No need to meet deadlines.
  Instead, try to utilise all available resources.

[a] = ”hard” real-time

[b] = ”soft” real-time

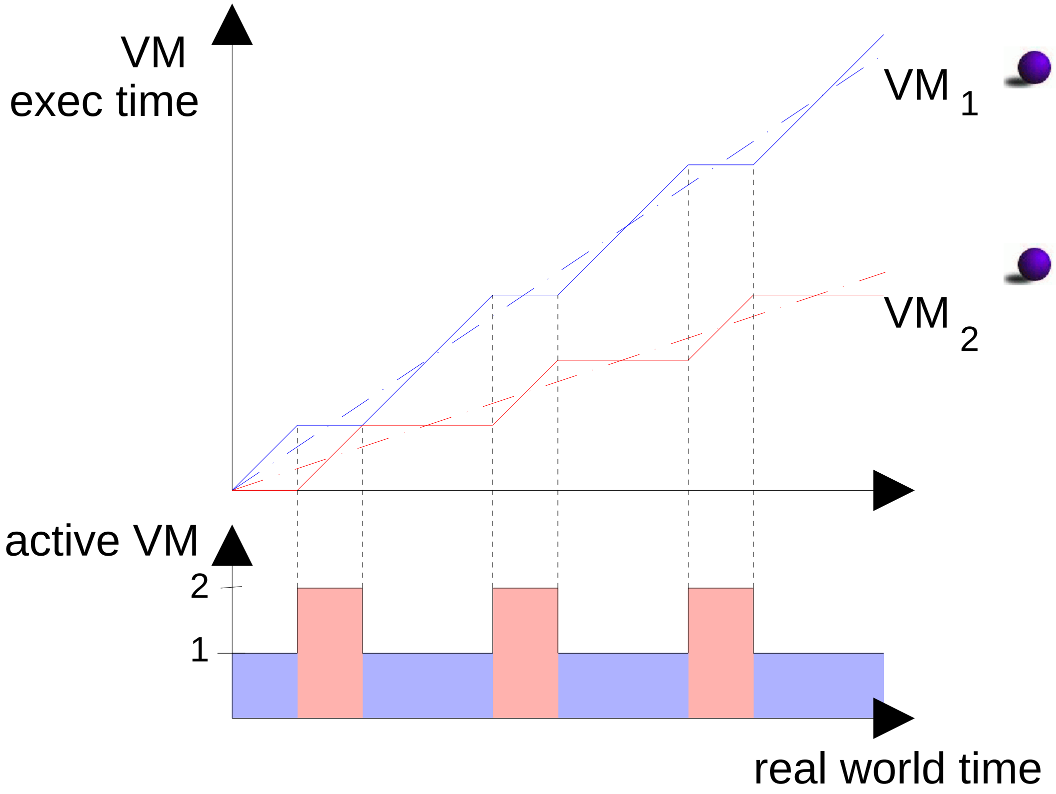
Impact on timing behaviour (1)



- Idealised behaviour: proportional share

Impact on timing behaviour (1)

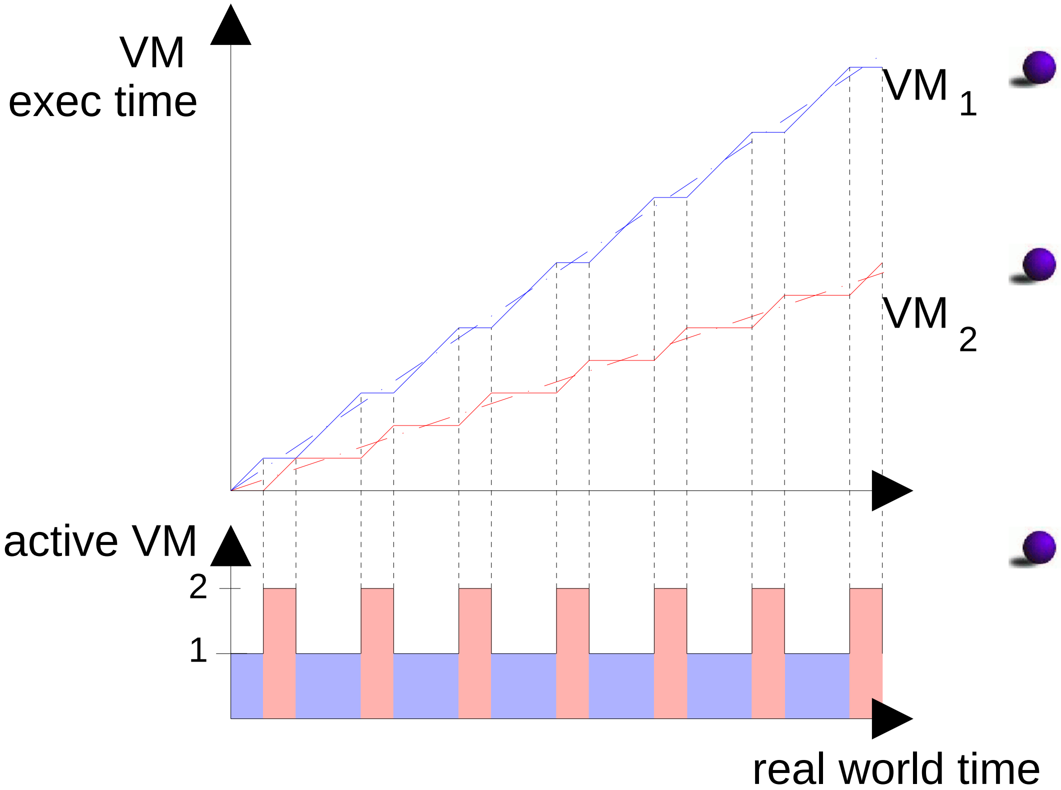
SYSGO REAL–TIME SOLUTIONS



- Idealised behaviour: proportional share
- Reality (uniprocessor): approximation by time-slicing

Impact on timing behaviour (1)

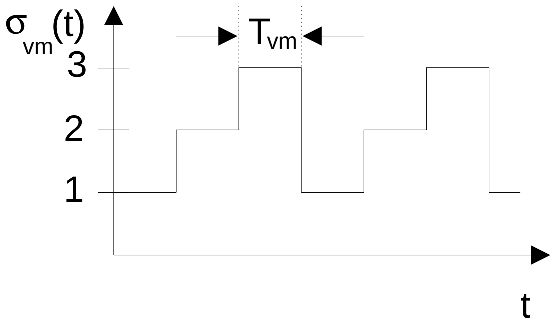
SYSGO REAL–TIME SOLUTIONS



- Idealised behaviour: proportional share
- Reality (uniprocessor): approximation by time-slicing
- Approximation improves as time slices are made smaller

Impact on timing behaviour (2)

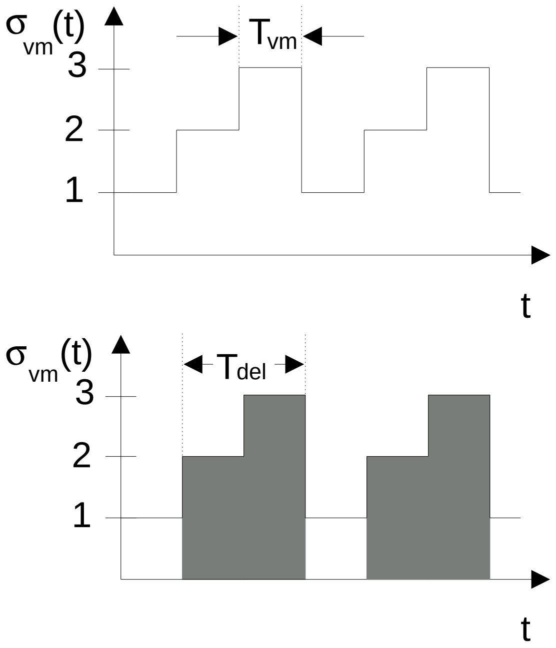
Estimate impact based on simple example:



- N virtual machines, time slice: T_{vm}

Impact on timing behaviour (2)

Estimate impact based on simple example:

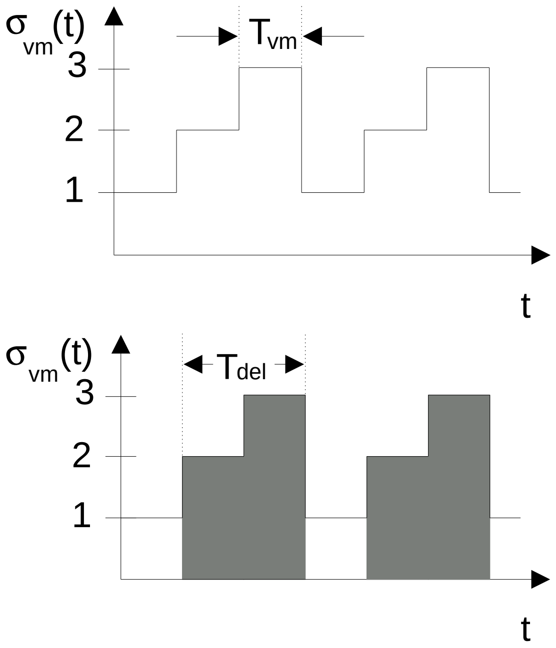


- N virtual machines, time slice: T_{vm}
- Each VM experiences a delay ("blackout") of: $T_{del} = T_{vm} \cdot (N - 1)$

Impact on timing behaviour (2)

SYSGO REAL−TIME SOLUTIONS

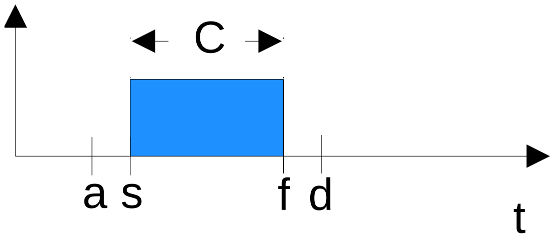
Estimate impact based on simple example:



- N virtual machines, time slice: T_{vm}
- Each VM experiences a delay ("blackout") of:
  $T_{del} = T_{vm} \cdot (N - 1)$

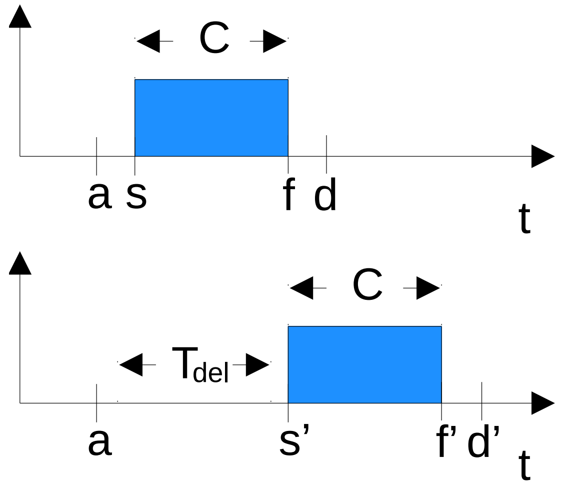
Delay may hit a real-time process running in a VM at any time.

Impact on timing behaviour (3)



Delay may hit a real-time process running in a VM at any time, e.g:

Impact on timing behaviour (3)

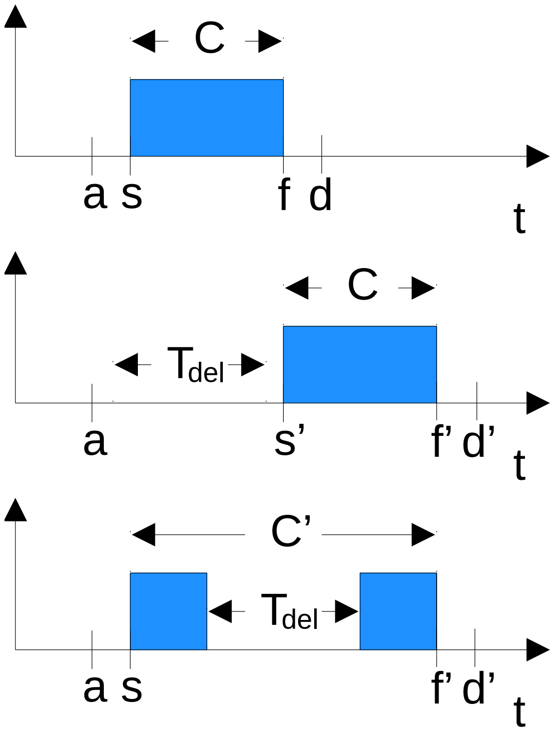


Delay may hit a real-time process running in a VM at any time, e.g:

- between arrival and start time

  affects: response time, jitter and deadline

Impact on timing behaviour (3)



Delay may hit a real-time process running in a VM at any time, e.g:

- between arrival and start time

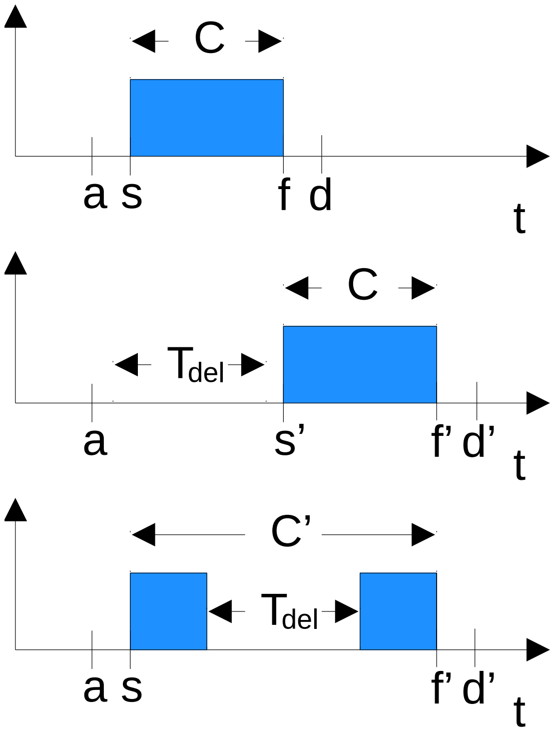
  affects: response time, jitter and deadline

- when process is active

  affects: computation time and deadline

Impact on timing behaviour (3)

SYSGO REAL-TIME SOLUTIONS



Delay may hit a real-time process running in a VM at any time, e.g:

- between arrival and start time

  affects: response time, jitter and deadline

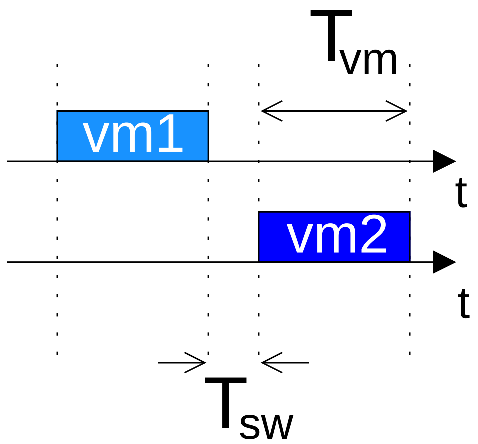
- when process is active

  affects: computation time and deadline

⇒ **Delay affects <u>all</u> parameters of a process that are critical for its real-time performance.**

Impact on timing behaviour (4)

Delay: $T_{del} = T_{vm} \cdot (N-1)$



- To reduce delay: reduce VM time slice (T_{vm}) as far as possible
- **Limit:** switching overhead: $U_{vm} = \frac{T_{sw}}{T_{vm}+T_{sw}}$
- $\Rightarrow T_{del} = \frac{T_{sw}\cdot(1-U_{vm})\cdot(N-1)}{U_{vm}}$
- I.e. impact depends on:
  - Worst case context switch time T_{sw} (Hardware constant)
  - Acceptable switching overhead U_{vm} (Design decision)

Impact on timing behaviour (5)

SYSGO REAL-TIME SOLUTIONS

Comparison with non-virtualised system:

- Response time/jitter limited directly by switch time: T_{sw}
- Relative impact: $\frac{T_{del}}{T_{sw}} = \frac{(1-U_{vm})\cdot(N-1)}{U_{vm}}$

Some realistic numbers:

- 3 virtual machines
- 5% overhead accepted

$\Rightarrow$ **response time and jitter are roughly 38 times(!) higher.**

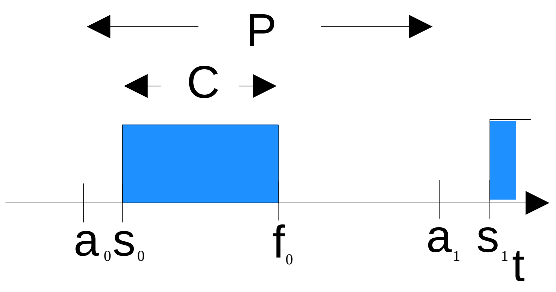
Impact on timing behaviour (6)

Impact of virtualisation on real-time performance is extensive, but bounded.

- Real-time programs can in principle work inside virtual machines, but will show significantly worse real-time performance (e.g. response time, jitter, computation time).
- Reason: assumption of proportionally shared processor.
- To achieve better real-time performance:
  - Abandon proportional share assumption
  - Adapt VM scheduling to workload classes

Experiment: Xen (1)

Periodic thread in DomU, Linux in Dom0

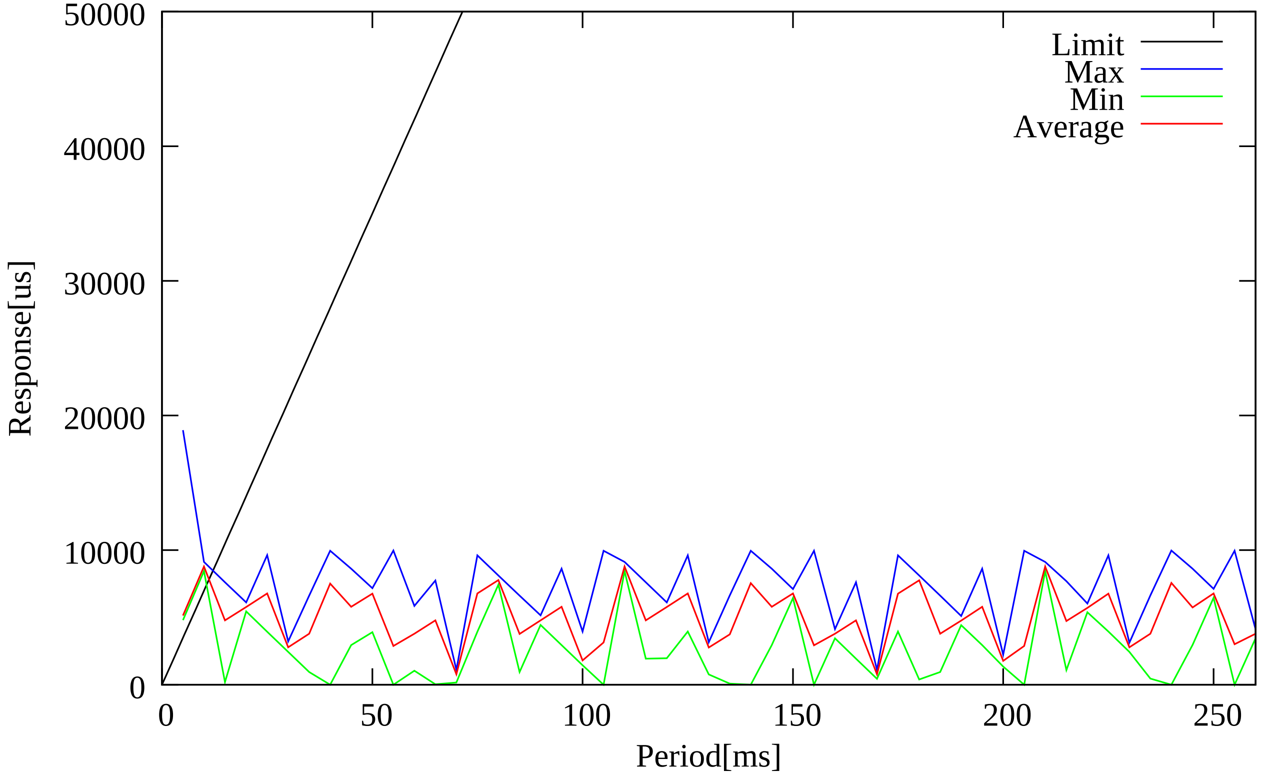


- $Load = \frac{C}{P}$
- $Response = s_i - a_i$

- Change: Period P
- Change: Load percentage
- Change: Dom0 (Linux) either idle or fully loaded
- Measure: Response (min/max/average)

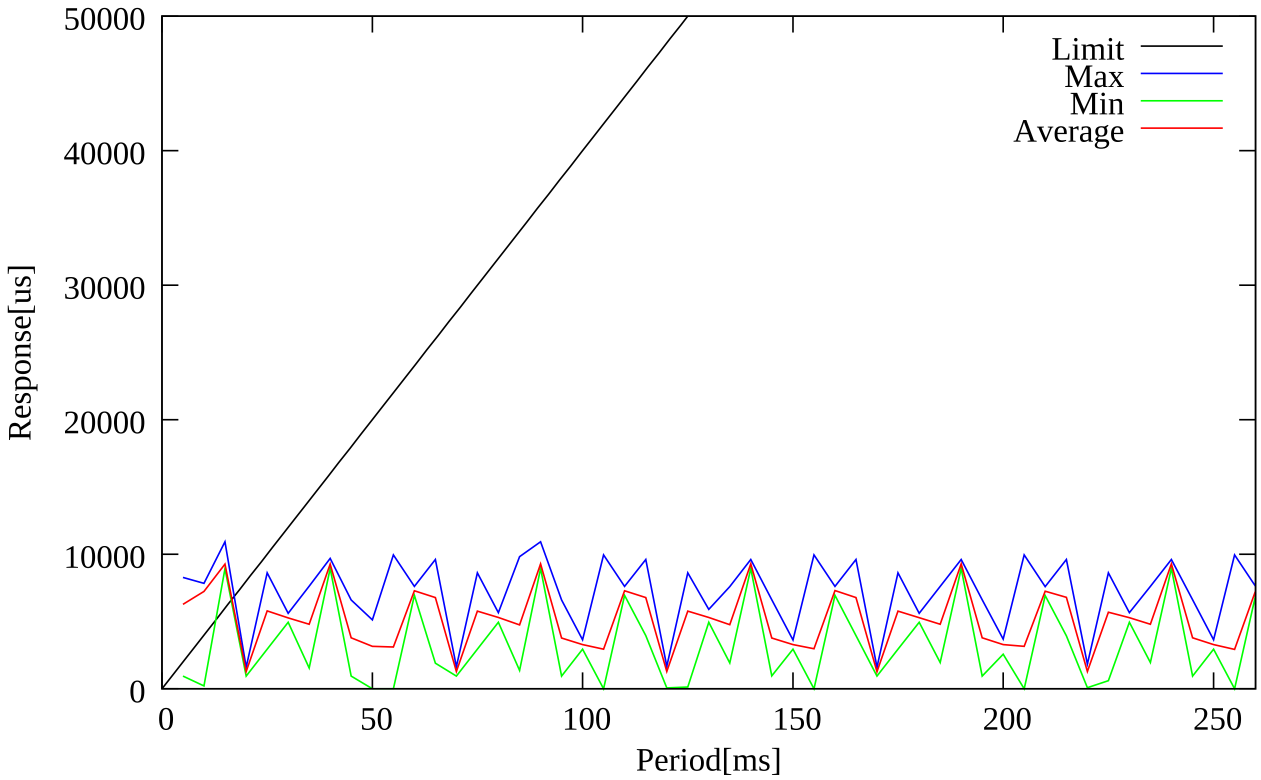
Experiment: Xen (2)

Dom0 (Linux) idle, Load = 30%.



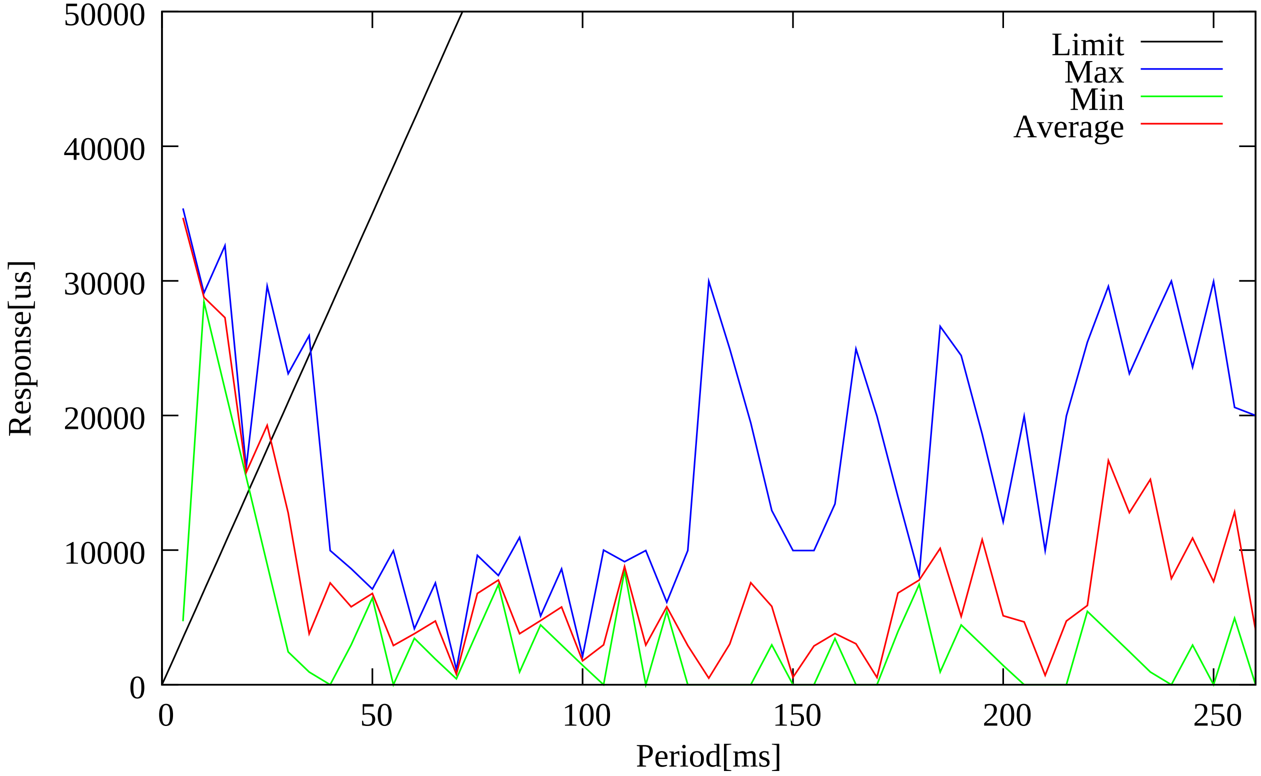
Experiment: Xen (2)

Dom0 (Linux) idle, Load = 60%.



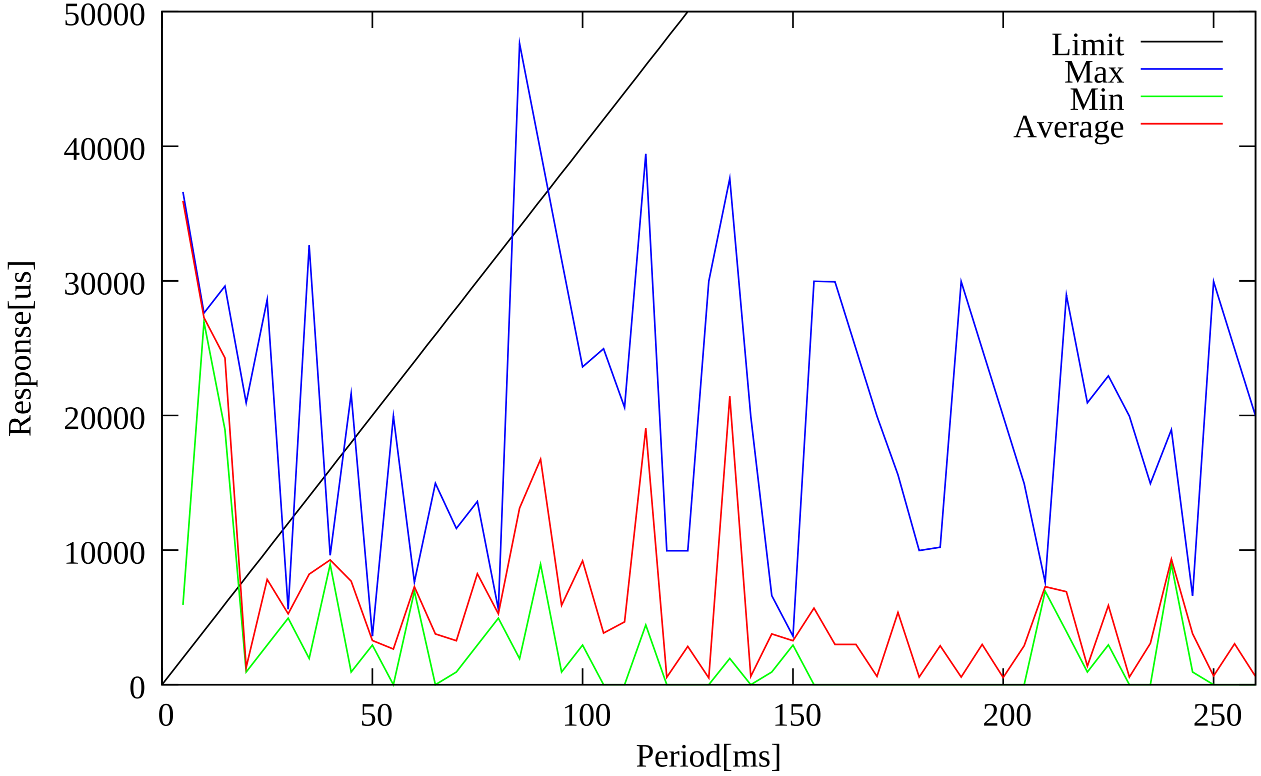
Experiment: Xen (2)

Dom0 (Linux) busy, Load = 30%.



Experiment: Xen (2)

Dom0 (Linux) busy, Load = 60%.



Experiment: Xen (3)

Conclusions:

- Response time (appears to be) bounded

  (no proof, just measurements)

- Typically: ~10-20 **milli**seconds

  (contemporary RTOSes: ~10-20 **micro**seconds)

- Domains are **not** temporally decoupled

  (Strong impact on worst-case response)

Requirements(1)

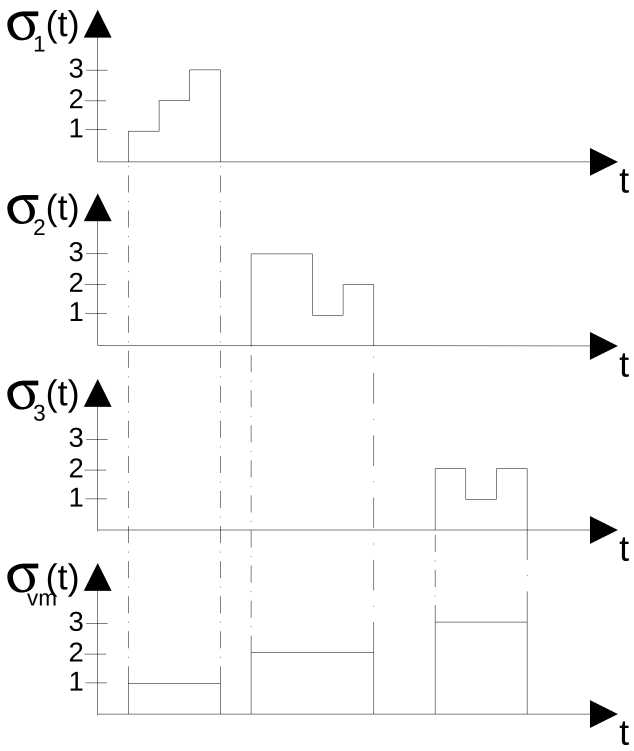
SYSGO REAL-TIME SOLUTIONS

Operating systems reflect application requirements.

- OS functionalities for different workload classes are (more or less) orthogonal.
- Covering all requirements with a single OS interface is possible, but not advisable, esp. in a VM environment.
- Assumption: each class uses its own OS
- $\Rightarrow$ Every class runs in a separate VM
- $\Rightarrow$ There are 3 distinct classes of VMs:
  1. Real-time, time-triggered
  2. Real-time, event-triggered
  3. Non-real-time

Requirements (2): Determinism

Time-triggered VMs:



- Define VM schedule to be a ”super” schedule of all time-triggered subsystem schedules.
- Only possible if time-triggered schedules ...
  - .. do not overlap
  - .. have the same period

Resulting VM ”super” schedule is strictly a function of time

Requirements (3): Responsiveness

SYSGO REAL-TIME SOLUTIONS

Event-triggered VMs:

- Reserving a time slot for event handling is possible, but may be too slow.
- Need a way for VMs to respond to events immediately, i.e. preempt the currently running VM.
- Contradictive to previous requirement
- Safety/security risk
- Must allow this only for trusted programs.
- Problem cannot be solved in a generic way, so the system must offer sufficient flexibiltiy to be configurable as needed.

Requirements (4): Re-allocation

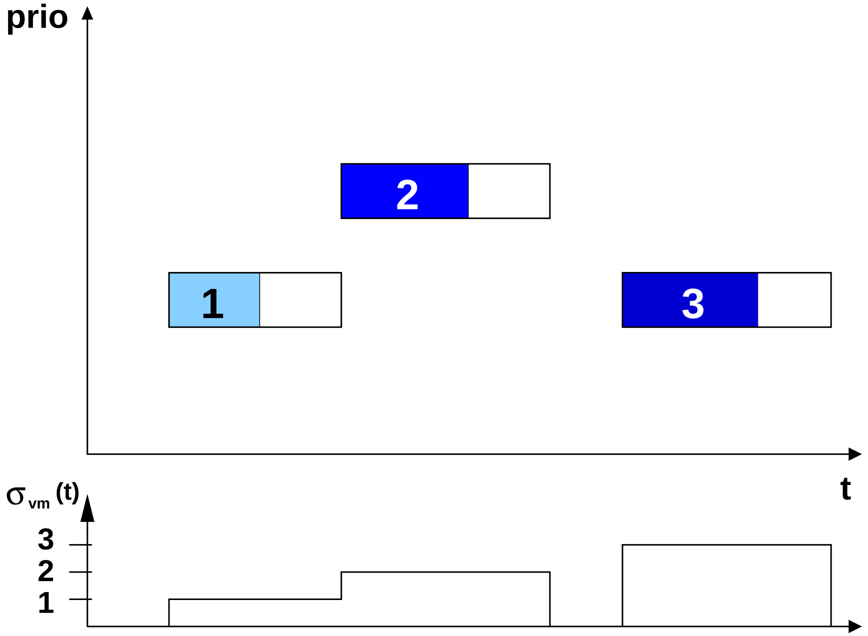
SYSGO REAL–TIME SOLUTIONS

Non-real-time VMs:

- Allocation of time to real-time VMs is done according to worst-case assumptions.
- In most cases, real-time VMs will not need all of their allocated time.
- Dynamically re-assign unused time to non-real-time VMs.
- **Also:** Must also be able to avoid starvation.
- Non-real-time VMs must share their resources evenly.

Approach(1)

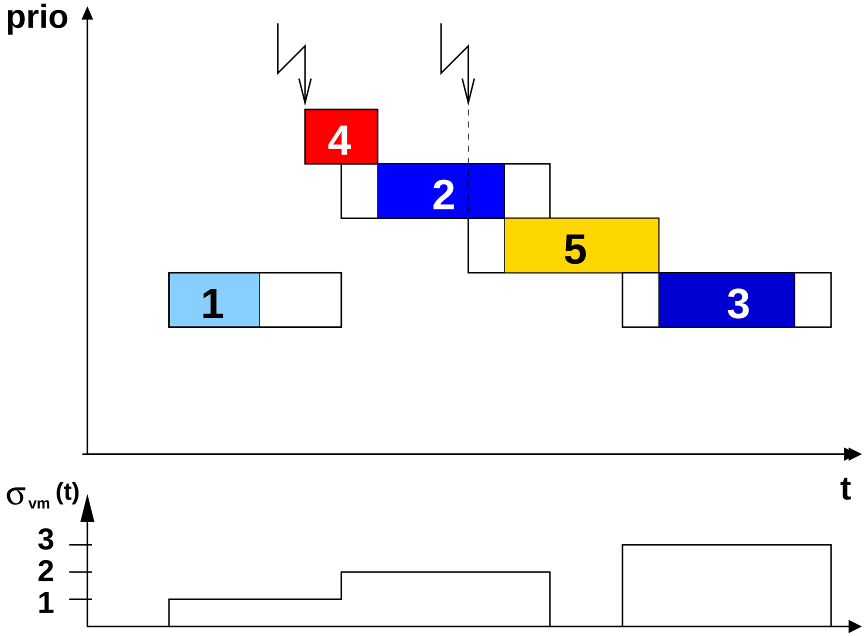
Basic idea: combine time-driven and priority-based scheduling:



- Establish a strictly time-driven scheduler for time-triggered VMs.

Approach(1)

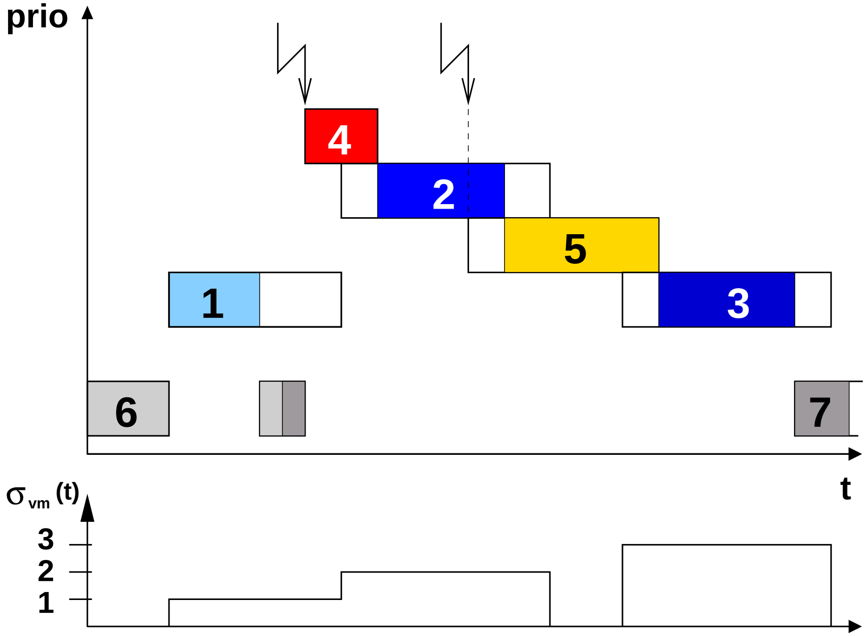
Basic idea: combine time-driven and priority-based scheduling:



- Establish a strictly time-driven scheduler for time-triggered VMs.
- Other VMs compete, based on priority:
  - Higher $\Rightarrow$ can preempt time-triggered VMs
  - Lower $\Rightarrow$ consume time not used by higher priority VMs

Approach(1)

Basic idea: combine time-driven and priority-based scheduling:



- Establish a strictly time-driven scheduler for time-triggered VMs.
- Other VMs compete, based on priority:
  - Higher ⇒ can preempt time-triggered VMs
  - Lower ⇒ consume time not used by higher priority VMs
  - Same ⇒ share time evenly

Approach(2)

First implementation: PikeOS microkernel

- VMs are represented as groups of processes:

  synchronuous: "VM container"

  asynchronous: Event/interrupt handlers

- Processes are assigned priorities and *time domains*.
- Processes can only execute if their time domain is active, regardless of priority.
- Time domains are represented by arrays of (FIFO) ready queues, with fixed priority levels.
- $\Rightarrow$ Classical, priority-driven FIFO scheduling within each time domain

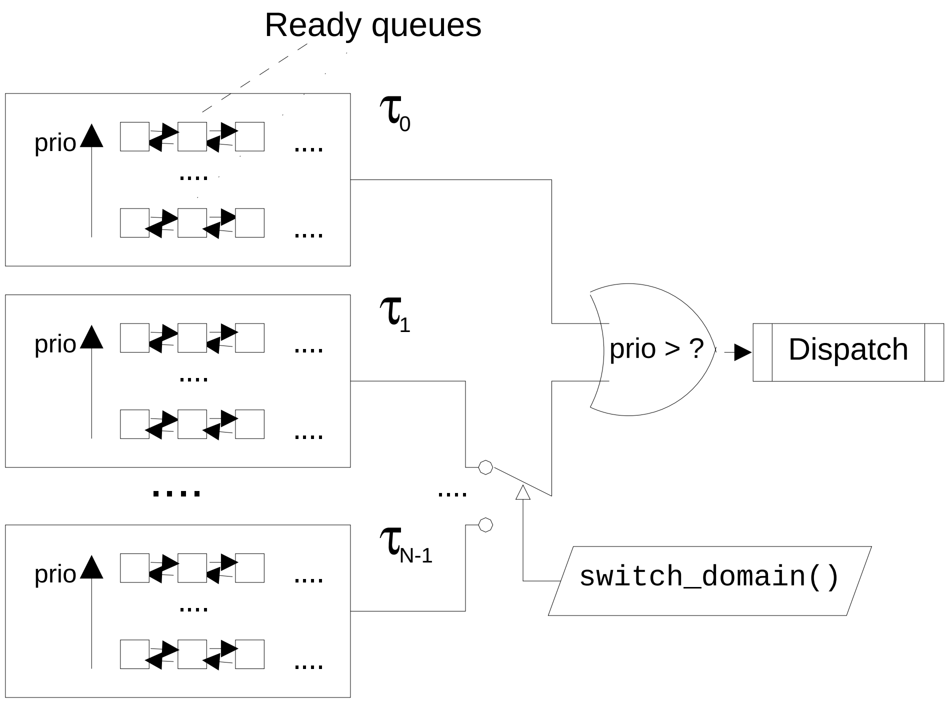
Approach(3)

- Similar to ARINC 653 (”partition scheduling”), time domains are cyclically activated.
- Unlike ARINC 653, two time domains can be active at the same time:

  τ_0: *background* domain: always active

  τ_i: *foreground* domain: one of $N-1$ time domains, cyclically switched
- Processes from τ_0 and the currently active τ_i compete by priority.

Approach(4)



- The microkernel only implements the mechanism to switch between domains.
- Switching policy is to be implemented at (trusted) user level.
  $\Rightarrow$ possibility to implement arbitrary policies

First results

PikeOS microkernel

- Conceptually based on ”L4” (Liedtke 1995)
- Currently: Implementations for PowerPC, ia-32 and MIPS
- OSes to run inside VMs: Linux, POSIX threads (PSE51), OSEK OS, ...
- Worst case context switch time: T_{sw} = 25 μs (PowerPC MPC5200@400 MHz)
- $\Rightarrow$ Impact (T_{del}) can be as low as 500 μs

Next steps

Current research

- Multiprocessor (Multicore) Support
  - Separation of time-triggered and event-triggered systems
  - Coscheduling of parallel real-time applications
- Use Xen as testbed
- Distributed Systems
  - Coscheduling of distributed real-time applications

Summary

- Current virtualisation systems are not designed for real-time applications.
- Impact of virtualisation on real-time performance is severe.
- Must adjust VM scheduling to (in part: conflicting) timing requirements of VMs.
- The outlined approach & implementation allows VMs with different timing requirements to coexist.
- Coexistence of time-triggered and event-triggered systems remains problematic
- The outlined approach allows VMs to choose precedence for either time-triggered or event-triggered processes.

The End

Thank you for your attention!