

# Real-time and VMM

-Real-time Xen for Embedded Devices

Chuck Yoo and team

Operating Systems Laboratory

Korea University

Feb. 25, 2009

# C o n t e n t s

- M o t i v a t i o n
- B a c k g r o u n d
  - T w o k e y p r o p e r t i e s i n r e a l - t i m e s y s t e m s
- C h a l l e n g e s t o X e n
  - d e t e r m i n i s m
  - p r e d i c t a b i l i t y
- D e v e l o p m e n t s t a t u s

# M o t i v a t i o n s

w h y r e a l - t i m e s u p p o r t i s i m p o r t a n t

- V a r i o u s f o r m s o f n e w e m b e d d e d s y s t e m s
  - R e a l - t i m e r e q u i r e m e n t
- M o b i l e p h o n e t o r u n b o t h :
  - C o m m u n i c a t i o n s t a c k l i k e G S M , C D M A
  - A p p l i c a t i o n s l i k e U I , g a m e s
- A u t o m o b i l e s , R o b o t s , U A V , a n d m o r e

# Determinism in Real-time System

- **Schedulability**
  - Theoretical foundation
  - Task  $T_i = (p_i, c_i, d_i)$ ,  
execution  $c_i$  time within a deadline  $d_i$ , during each period  $p_i$
  - Schedulability can be decided for a given scheduling policy and a real-time task set
- Real-time operating system performs schedulability test whenever the task set or the scheduling policy is changed
  - Performs admission control so that all the task in the system can meet the execution condition
  - In order to simplify the schedulability analysis at run-time, it fixes the task set and utilization a priori.

# Schedulability Test

- For a given Task set  $T = \{T_i(p_i, c_i, d_i)\}$  and Algorithm A, scheduler S checks whether all the tasks in the system meet the execution condition

- T is RM-schedulable if and only if

$$w_i(t) \leq t$$

$$, \text{ where } t = kp_j, j = 1, 2, \dots, i, k = 1, \dots, \frac{p_i}{p_j} ,$$

$$w_i(t) = \sum_{k=1}^i c_k \frac{t}{p_k}, \quad 0 < t < p_i .$$

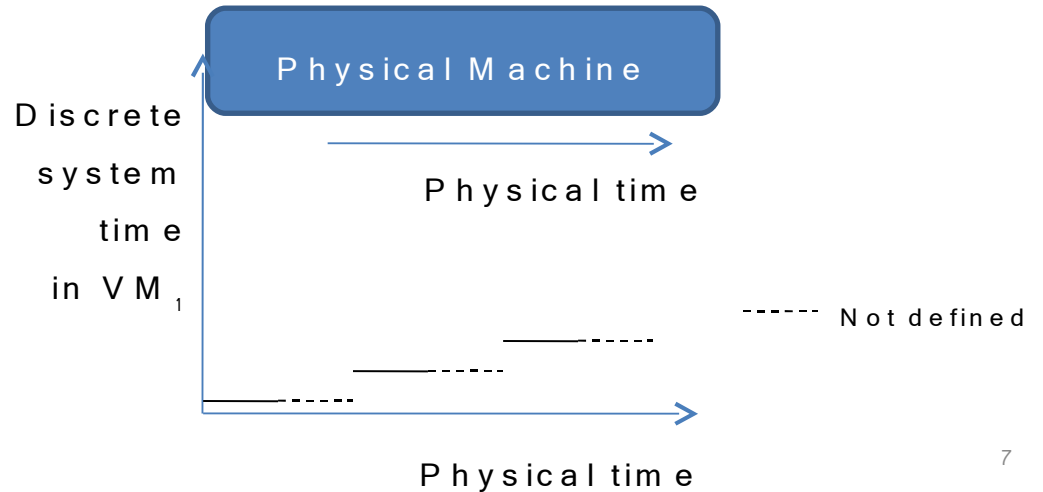
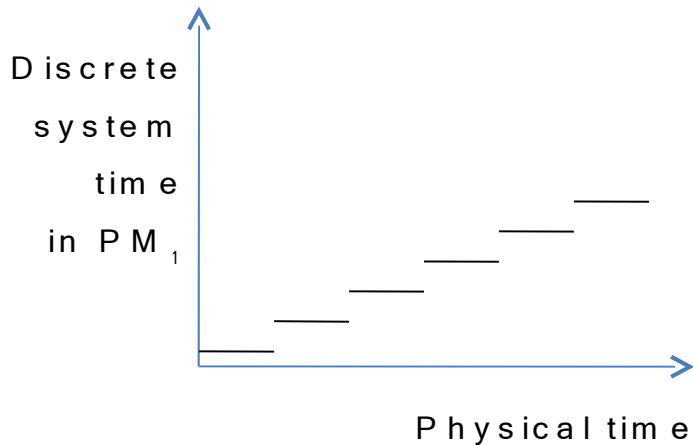
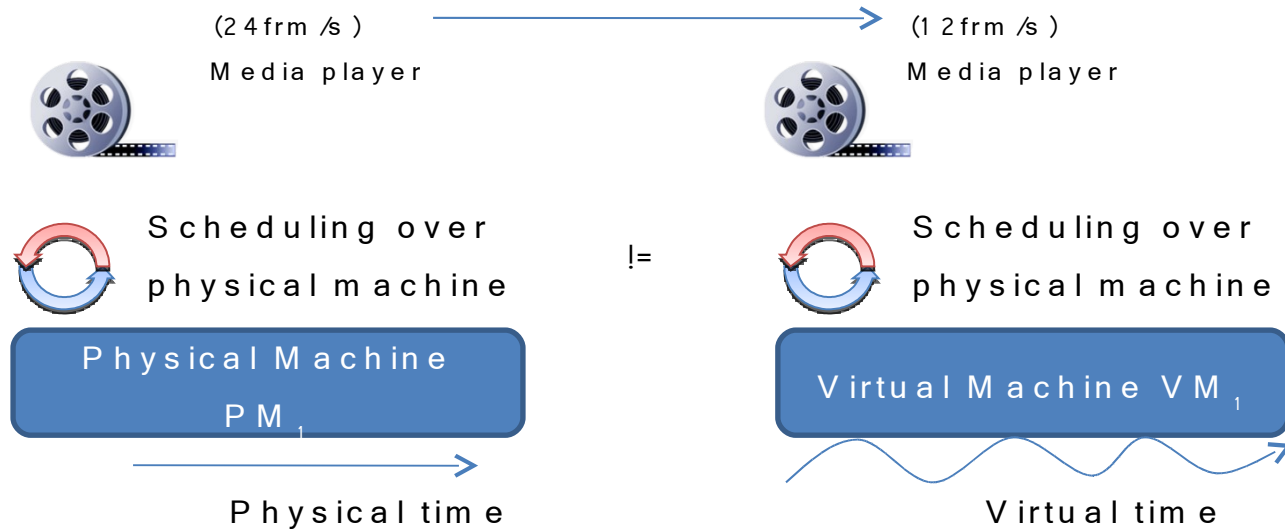
- T is EDF-schedulable if and only if

$$U = \sum_{i=1}^n \frac{c_i}{p_i} \leq 1 .$$

# Predictability in Real-time System

- Interrupt is a key obstacle to predictability
  - Interrupt is **asynchronous and un-deterministic**
  - For predictability, need to make interrupt processing predictable
- Real-time operating system adopts mechanisms to make it **time-bounded**
  - Disable another interrupt
  - Nested /prioritize interrupt
  - Polling I/O
  - Deferred (Delayed) processing
    - e.g.) bottom half, softirq, etc.

# Xen: Physical time $\neq$ Virtual time



# Xen: I/O model

- Does not provide predictable time-bound
- Split driver model is not suitable for real-time I/O

# VMM for Embedded Systems

- Xen-ARM from Samsung Elec.  
(first ARM porting)
- L4 from Open Kernel Labs  
(microkernel-based virtualization)
- VLX from VirtualLogix  
(real-time support RTOS runs inside the VMM)

Similar issues inherited !

C h a l l e n g e t o X e n

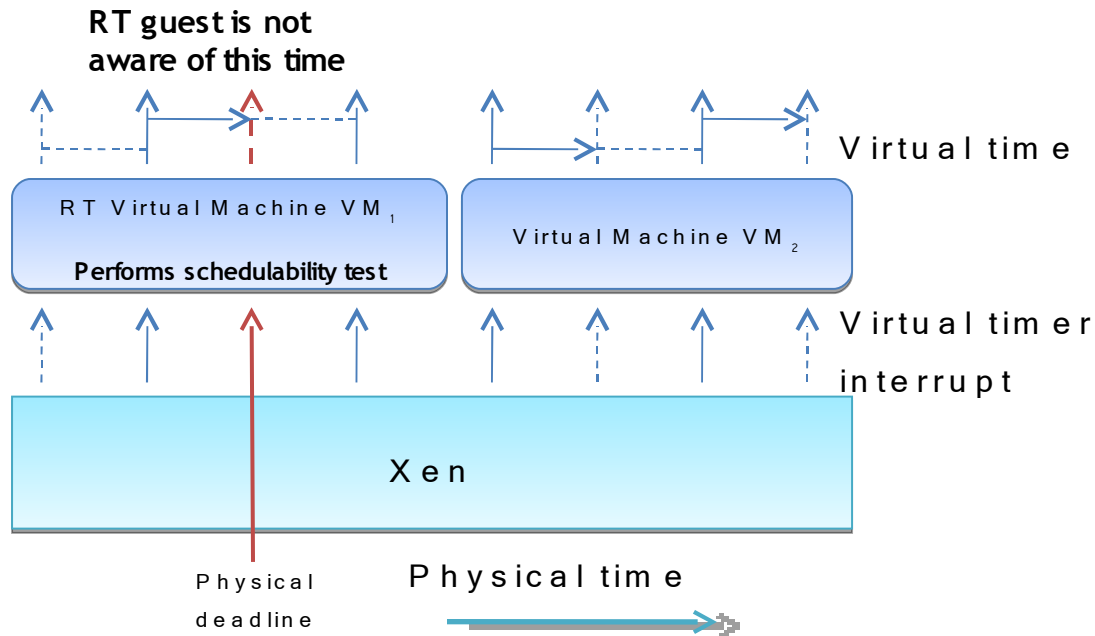
**HOW TO PROVIDE  
DETERMINISM?**

# Scheduling in Xen

- SEDF, Credit schedulers
  - Simple EDF implements real-time EDF scheduler
    - User should define period, execution slice
  - Credit scheduler implements credit based fair scheduling with I/O boost
    - Supports multi-processor, load balancing
    - I/O BOOST priority to keep responsiveness
- Time keeping in Xen
  - Timer interrupts are distributed over VMs
    - Hardware timer interrupts are handled by the Xen, and delivered by via event channel
    - VM can be aware of time passage by virtual timer interrupt which has been sent by the Xen
      - Time flows only when the guest OS is scheduled by the VMM

# How to Perform Schedulability Analysis?

- Problem
  - Task scheduling should be performed by RTOS
    - It performs schedulability test, RT scheduling policy
    - Xen does not know the tasks in the RTOS
  - However, RT guest is not aware of the physical time
    - RTOS is aware of the virtual time provided by the Xen
    - Deadline might be missed if the Xen does not timely schedule RT guest

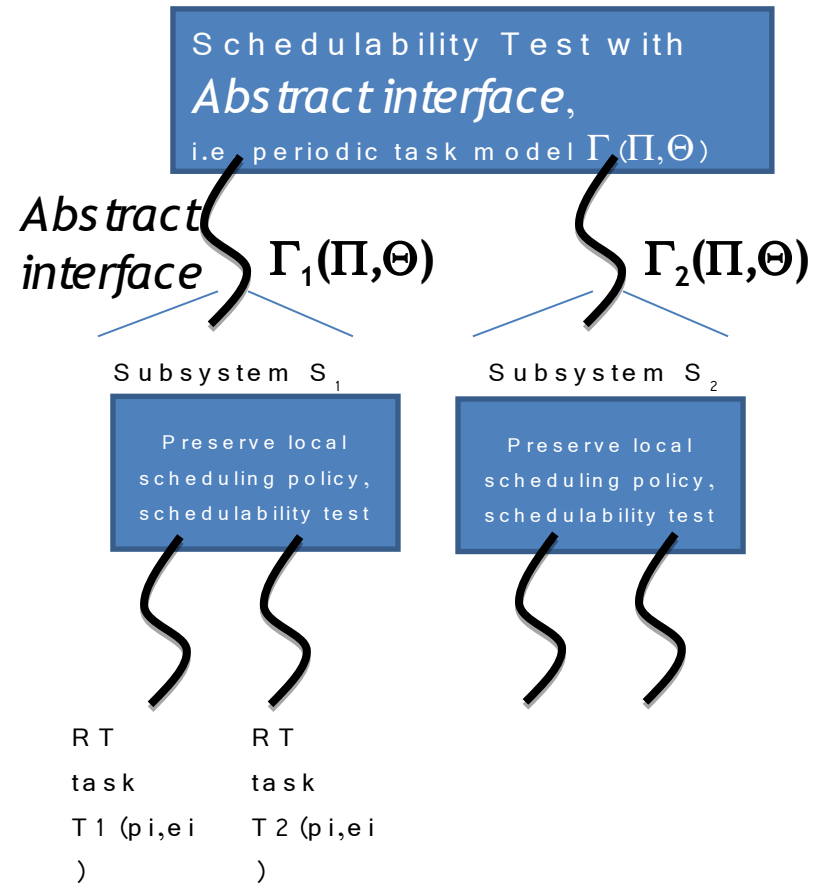


# Solution 1 - Integrated scheduling at VMM

- Scheduling at the lowest layer
  - At VMM ?
    - VMM should be aware of all the real-time tasks in RT guest OS
      - e.g.) L4 task scheduling
    - What if multiple RT guest OS
  - In addition,
    - How to support GPOS
      - Scheduling analysis whenever GPOS creates a user process
      - Sorting all the tasks in all the guests for SEDF at run time should have a considerable overhead
    - Interfere GPOS scheduler
      - Local scheduler knows a best way to utilize the resources

# Solution 2 – Workload aggregation

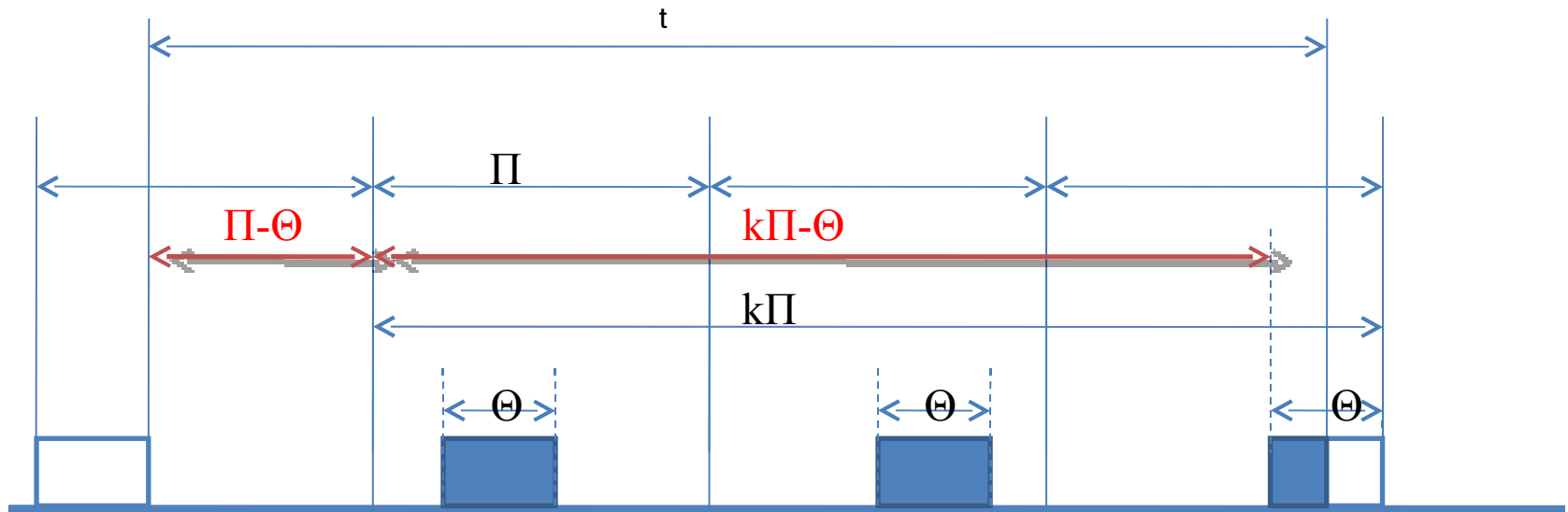
- Hierarchical scheduling framework
  - RT guest OS can **preserve its own scheduling policy and schedulability**
    - No global schedulability test for local task set change
  - Aggregate RT guest OS (its tasks) as one real-time task
    - Need abstract interface to translate all the RT tasks to a single RT task
    - **slightly more overhead**



# Aggregation

## Periodic resource model

- $k = \max(\lfloor (t - (\Pi - \Theta)) / \Pi \rfloor, 1)$ 
  - number of periods within  $t$
  - i.e.  $k = 3$  for the figure below



# Example of Aggregation

- For RT guest R whose task set T is  $\{t_1(50, 7, 50), t_2(75, 9, 75)\}$  and algorithm A is EDF.

- Utilization of workload = 0.26
- Let period  $\Pi$ , resource capacity  $U = \Theta / \Pi$
- Supply bound function (sbf): minimum resource supply from VMM

$$sbf_{\Gamma}(t) = \begin{cases} t - k\Pi - \Theta & \text{if } t \in [k\Pi - 2\Theta, k\Pi - \Theta] \\ k - 1 & \text{otherwise,} \end{cases}$$

, where  $k = \max\left\{\frac{t - \Pi - \Theta}{\Pi}, 1\right\}$

- Demand bound function (dbf): total resource demand for the workload

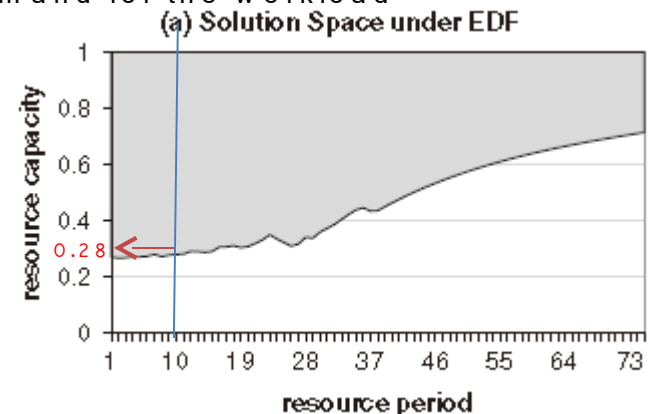
$$dbf_{EDF}(W, t) = \sum_{T_i \in W} \frac{t}{p_i} \cdot e_i.$$

- Gray region in the graph presents feasible resource allocation

$$\forall 0 < t \leq LCM_W, dbf_{EDF}(W, t) \leq sbf_{\Gamma}(t)$$

, where  $LCM_W$  is the least common multiple of  $p_i$  for all  $T_i \in W$ .

- Let's assume that  $\Pi = 10$ ,
  - We need to allocate at least 2.8 (as  $\Theta$ ) to R
  - Rounding it and we get  $\Gamma(10, 3)$



I. Shin, I. Lee "Compositional real-time scheduling framework with periodic model," ACM Trans. on Embedded Computing Systems Vol.7 (3), 2008

# To make aggregation work

- To incorporate the hierarchical scheduling framework
  - Need a tool to calculate with the given task set
- Resource supply bound  $\geq$  Resource demand bound
  - Xen should provide enough resources to RTOS
    - RTOS calculates demand bound (total resource demand to meet the schedulability)
    - Xen provides CPU resource to RTOS at least to resource supply bound (minimum resource supply for a corresponding scheduling policy)

C h a l l e n g e t o X e n

# **HOW TO DEAL WITH PREDICTABILITY?**

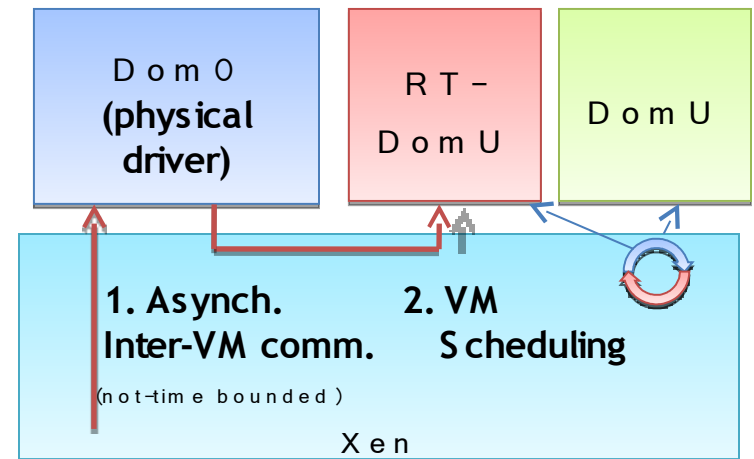
# How to Handle Interrupts in Time-bounded fashion?

- Problem

- Xen's domain model splits physical I/O processing from virtual machine

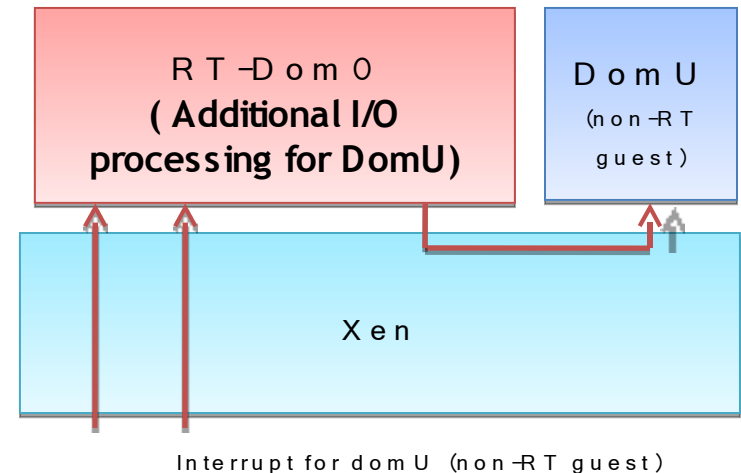
- Additional latency
    - Unbounded-processing time

- Dom 0 may be kept busy i.e. target guest may not be scheduled immediately



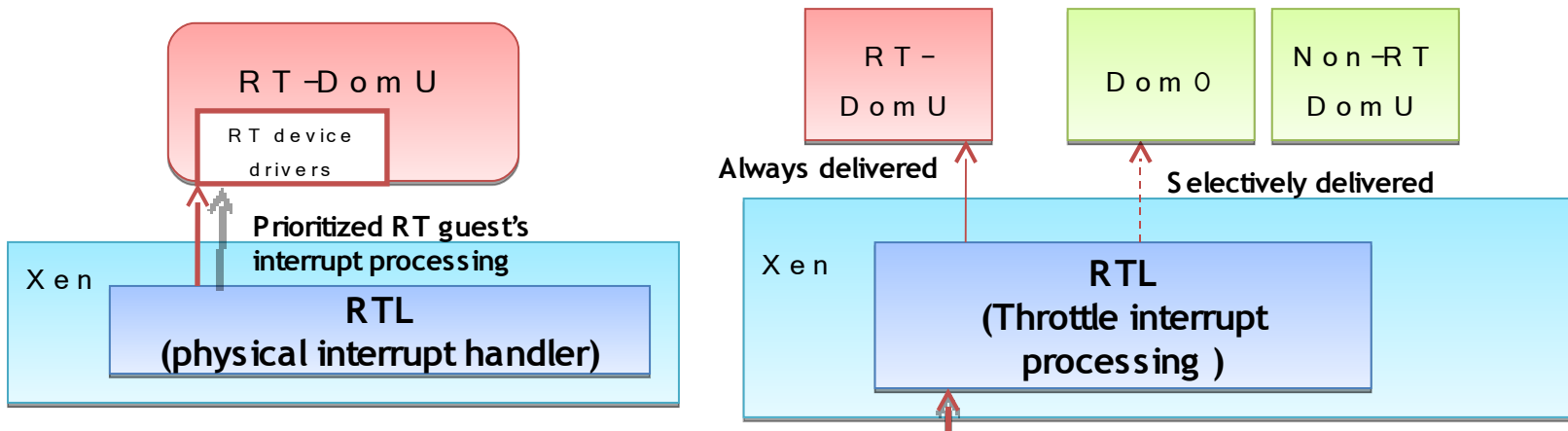
# Solution 1 – Putting RT guest at Dom 0

- At Dom 0
  - Special control/driver domain
  - For faster I/O response
    - Good for timely interrupt processing
- Cons
  - Another non-RT VM's I/O processing is affected
    - Hard to support schedulability analysis (non-RT guest processing, which is not time-bounded)
  - RT guest can be overloaded with too many interrupts



# Solution 2 - Adopting Real-time Sublayer (RTL)

- Let RT guest access physical devices directly
  - RT devices are dedicated to RT guests
  - Interrupt service routine (ISR) of RT guest should be minimal
- Add real-time sublayer to provide time-bounded interrupts for RT guest
  - Handle RT guest's interrupt first
  - Throttle interrupt processing for RT guest and non-RT guest
    - Selectively deliver interrupts to dom0 by monitoring incoming interrupts
    - To avoid unbounded interrupt processing for non-RT guest  
e.g.) network flooding



# M o r e o n R T L

- R T L controls the incoming rate of the aggressive interrupts
  - discard interrupts when too many
    - if current interrupt incoming rate > maximum interrupt rate threshold
      - Max. interrupt rate could be derived from available cpu bandwidth from schedulability analysis

# Development status

- Ported uC /O S -II over XenARM platform
  - Freescale iM X 21
- Being implemented
  - Hierarchical scheduler
  - Tools for schedulability analysis
  - RTL
- Preview at  
<http://www.youtube.com/watch?v=Vli9zb62>

# C o n c l u s i o n

- R e a l - t i m e s u p p o r t i s i m p o r t a n t
- X e n c a n d o i t

**THANKS !**

# Real-time and VMM

-Real-time Xen for Embedded Devices

Chuck Yoo and team  
Operating Systems Laboratory  
Korea University  
Feb. 25, 2009

# C o n t e n t s

- M o t i v a t i o n
- B a c k g r o u n d
  - T w o k e y p r o p e r t i e s i n r e a l - t i m e s y s t e m s
- C h a l l e n g e s t o X e n
  - d e t e r m i n i s m
  - p r e d i c t a b i l i t y
- D e v e l o p m e n t s t a t u s

# Motivations

why real-time support is important

- Various forms of new embedded systems
  - Real-time requirement
- Mobile phone to run both:
  - Communication stack like GSM, CDMA
  - Applications like UI, games
- Automobiles, Robots, UAV, and more

# Determinism in Real-time System

- **Schedulability**
  - Theoretical foundation
  - Task  $T_i = (p_i, c_i, d_i)$ ,  
execution  $c_i$  time within a deadline  $d_i$ , during each period  $p_i$
  - Schedulability can be decided for a given scheduling policy and a real-time task set
- Real-time operating system performs schedulability test whenever the task set or the scheduling policy is changed
  - Performs admission control so that all the task in the system can meet the execution condition
  - In order to simplify the schedulability analysis at run-time, it fixes the task set and utilization a priori.

# Schedulability Test

- For a given Task set  $T = \{T_i(p_i, c_i, d_i)\}$  and Algorithm A, scheduler S checks whether all the tasks in the system meet the execution condition

- T is RM-schedulable if and only if

$$w_i(t) \leq t$$

, where  $t = kp_j, j = 1, 2, \dots, i, k = 1, \dots, \frac{p_i}{p_j}$ ,

$$w_i(t) = \sum_{k=1}^i c_k \frac{t}{p_k}, \quad 0 < t < p_i.$$

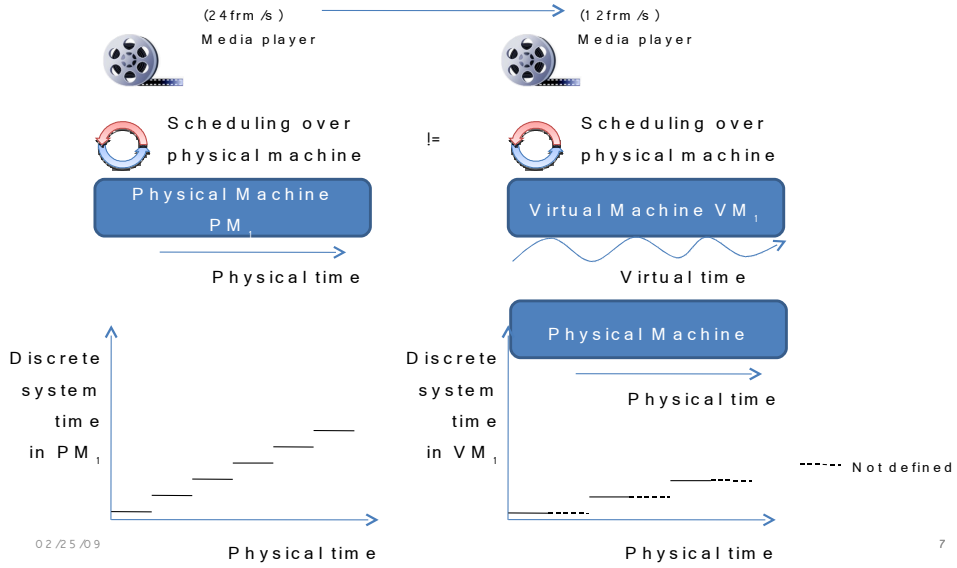
- T is EDF-schedulable if and only if

$$U = \sum_{i=1}^n \frac{c_i}{p_i} \leq 1.$$

# Predictability in Real-time System

- Interrupt is a key obstacle to predictability
  - Interrupt is **asynchronous and un-deterministic**
  - For predictability, need to make interrupt processing predictable
- Real-time operating system adopts mechanisms to make it **time-bounded**
  - Disable another interrupt
  - Nested/prioritize interrupt
  - Polling I/O
  - Deferred (Delayed) processing
    - e.g.) bottom half, softirq, etc.

# Xen: Physical time $\neq$ Virtual time



## Xen: I/O model

- Does not provide predictable time-bound
- Split driver model is not suitable for real-time I/O

## VMM for Embedded Systems

- Xen-ARM from Samsung Elec.  
(first ARM porting)
- L4 from Open Kernel Labs  
(microkernel-based virtualization)
- VLX from VirtualLogix  
(real-time support RTOS runs inside the VMM)

Similar issues inherited !

Challenge to Xen

# **HOW TO PROVIDE DETERMINISM?**

# Scheduling in Xen

- **SEDF, Credit schedulers**
  - Simple EDF implements real-time EDF scheduler
    - User should define period, execution slice
  - Credit scheduler implements credit based fair scheduling with I/O boost
    - Supports multi-processor, load balancing
    - I/O BOOST priority to keep responsiveness
- **Time keeping in Xen**
  - Timer interrupts are distributed over VMs
    - Hardware timer interrupts are handled by the Xen, and delivered by via event channel
    - VM can be aware of time passage by virtual timer interrupt which has been sent by the Xen
      - Time flows only when the guest OS is scheduled by the VMM

# How to Perform Schedulability Analysis?

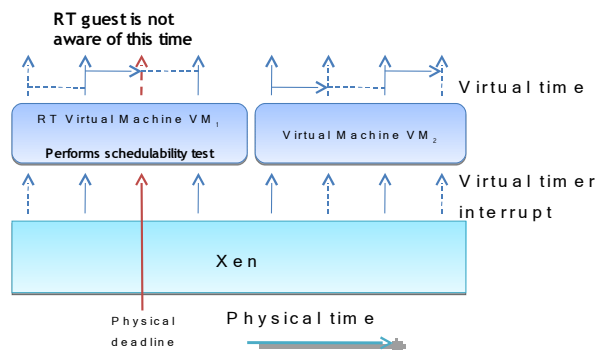
- Problem

- Task scheduling should be performed by RTOS

- It performs schedulability test, RT scheduling policy
    - Xen does not know the tasks in the RTOS

- However, RT guest is not aware of the physical time

- RTOS is aware of the virtual time provided by the Xen
    - Deadline might be missed if the Xen does not timely schedule RT guest

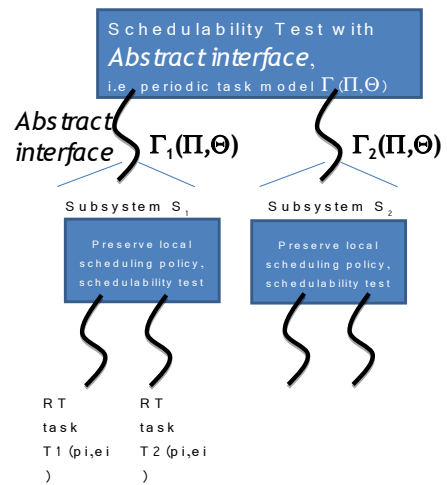


# Solution 1 - Integrated scheduling at VMM

- Scheduling at the lowest layer
  - At VMM ?
    - VMM should be aware of all the real-time tasks in RT guest OS
      - e.g.) L4 task scheduling
    - What if multiple RT guest OS
  - In addition ,
    - How to support GPOS
      - Scheduling analysis whenever GPOS creates a user process
      - Sorting all the tasks in all the guests for SEDF at run time should have a considerable overhead
    - Interfere GPOS scheduler
      - Local scheduler knows a best way to utilize the resources

## Solution 2 – Workload aggregation

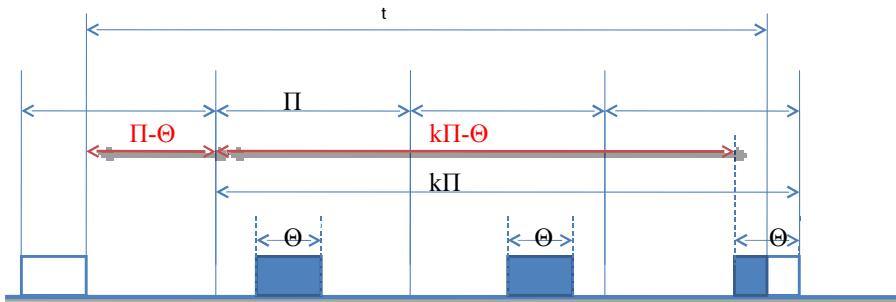
- Hierarchical scheduling framework
  - RT guest OS can **preserve its own scheduling policy and schedulability**
    - No global schedulability test for local task set change
  - Aggregate RT guest OS (its tasks) as one real-time task
  - Implication to Xen
    - Need abstract interface to translate all the RT tasks to a single RT task
    - **slightly more overhead**



# Aggregation

## Periodic resource model

- $k = \max(\lfloor (t - (\Pi - \Theta)) / \Pi \rfloor, 1)$ 
  - number of periods within  $t$
  - i.e.  $k=3$  for the figure below



# Example of Aggregation

- For RT guest R whose task set T is  $\{t_1(50, 7, 50), t_2(75, 9, 75)\}$  and algorithm A is EDF.

- Utilization of workload = 0.26
- Let period  $\Pi$ , resource capacity  $U = \Theta / \Pi$
- Supply bound function (sbf): minimum resource supply from VMM

$$sbf_R(t) = \begin{cases} t - \lfloor k + 1 \rfloor \Pi - \Theta & \text{if } t \in \lfloor k + 1 \rfloor \Pi - 2\Theta, \lfloor k + 1 \rfloor \Pi - \Theta \\ \lfloor k + 1 \rfloor \Pi & \text{otherwise,} \end{cases}$$

, where  $k = \max\left\{\frac{t - \Pi - \Theta}{\Pi}, 1\right\}$

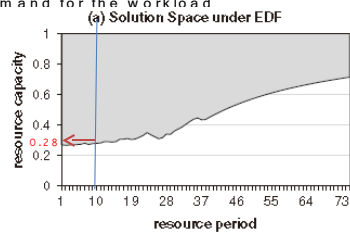
- Demand bound function (dbf): total resource demand for the workload

$$dbf_{EDF}(W, t) = \sum_{T_i \in W} \frac{t}{p_i} \cdot e_i.$$

- Gray region in the graph presents feasible resource allocation

$$\forall 0 < t \leq LCM_W, dbf_{EDF}(W, t) \leq sbf_R(t)$$

, where  $LCM_W$  is the least common multiple of  $p_i$  for all  $T_i \in W$ .



I. Shin, J. Lee "Compositional real-time scheduling framework with periodic model," ACM Trans. on Embedded Computing Systems Vol.7 (3), 2008

## To make aggregation work

- To incorporate the hierarchical scheduling framework
  - Need a tool to calculate with the given task set
- Resource supply bound  $\geq$  Resource demand bound
  - Xen should provide enough resources to RTOS
    - RTOS calculates demand bound (total resource demand to meet the schedulability)
    - Xen provides CPU resource to RTOS at least to resource supply bound (minimum resource supply for a corresponding scheduling policy)

C h a l l e n g e t o X e n

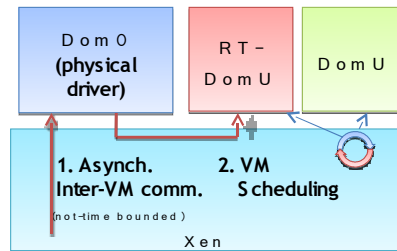
# **HOW TO DEAL WITH PREDICTABILITY?**

# How to Handle Interrupts in Time-bounded fashion?

- Problem

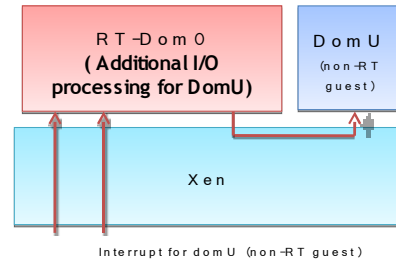
- Xen's domain model splits physical I/O processing from virtual machine

- Additional latency
    - Unbounded-processing time
      - Dom0 may be kept busy i.e. target guest may not be scheduled immediately



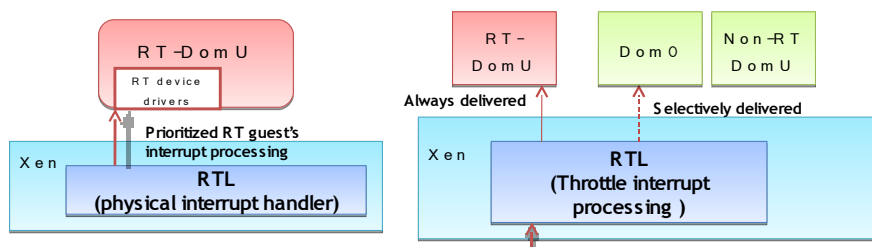
# Solution 1 – Putting RT guest at Dom 0

- At Dom 0
  - Special control/driver domain
  - For faster I/O response
    - Good for timely interrupt processing
- Cons
  - Another non-RT VM's I/O processing is affected
    - Hard to support schedulability analysis (non-RT guest processing, which is not time-bounded)
  - RT guest can be overloaded with too many interrupts



## Solution 2 – Adopting Real-time Sublayer (RTL)

- Let RT guest access physical devices directly
  - RT devices are dedicated to RT guests
  - Interrupt service routine (ISR) of RT guest should be minimal
- Add real-time sublayer to provide time-bounded interrupts for RT guest
  - Handle RT guest's interrupt first
  - Throttle interrupt processing for RT guest and non-RT guest
    - Selectively deliver interrupts to dom 0 by monitoring incoming interrupts
    - To avoid unbounded interrupt processing for non-RT guest  
e.g.) network flooding



## More on RTL

- RTL controls the incoming rate of the aggressive interrupts
  - discard interrupts when too many
    - if current interrupt incoming rate > maximum interrupt rate threshold
      - Max. interrupt rate could be derived from available cpu bandwidth from schedulability analysis

## Development status

- Ported uC /OS -II over XenARM platform
  - Freescale iMX21
- Being implemented
  - Hierarchical scheduler
  - Tools for schedulability analysis
  - RTL
- Preview at  
<http://www.youtube.com/watch?v=Vli9zb62>

## Conclusion

- Real-time support is important
- Xen can do it

**THANKS!**