ESSE 2012 - Embedded Tutorials

November 08, 2012 – 9:00am – 1:00pm - Room: TBD

Embedded System's Virtualization

Concepts, issues and challenges

Fabiano Hessel (speaker) and Alexandra Aguiar

Group of Embedded Systems – GSE,

Faculty of Informatics, PUCRS, Porto Alegre Brazil

Brazil....Porto Alegre...

without beach and cold?!!

Outline

- Introduction (what is virtualization?)
- Motivation (why to use virtualization?)
- Use Cases (how is virtualization used?)
- Challenges (what can go wrong?)
- Conclusion (Let’s talk about it!)

Embedded Systems, over the years...

of processors
# of features
SW complexity
Communication complexity

+

Time-to-market
User flexibility
Security needs
Software reuse

=

Real-time needs
Size Constraint
Energy Constraint
Cost Constraint

Embedded Systems

- More requirements
  - Same constraints
- More possible scenarios
  - Same constraints
- More devices
  - Same constraints

Typical Embedded Systems

- Long ago…

  Application Software Layer (Optional)

  System Software Layer (Optional)

  Hardware Layer (Required)

- Currently…

  Application Software Layer (Required)

  System Software Layer (Required)

  Hardware Layer (Required)

Decreasing GAP to GP computing

- Some embedded systems have many features once available only in general purpose computers
- Embedded Multiprocessing has become a reality and GP techniques are being revisited
- Convergence of systems, applications and devices
- Cloud computing
- Ubiquitous computing

GP techniques in ES

- Throughout the years, many techniques have been migrated from GP to ES
  - Communication and Network protocols (buses, NoCs)
  - Different Programming models
  - Task migration

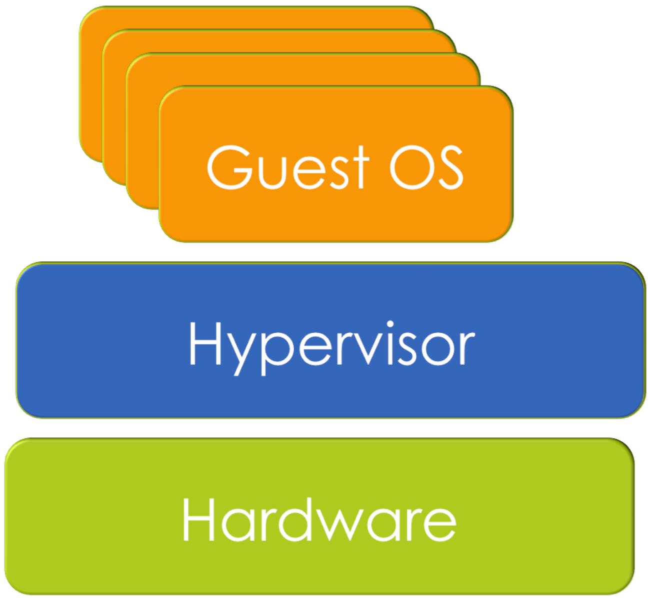
And…

Virtualization

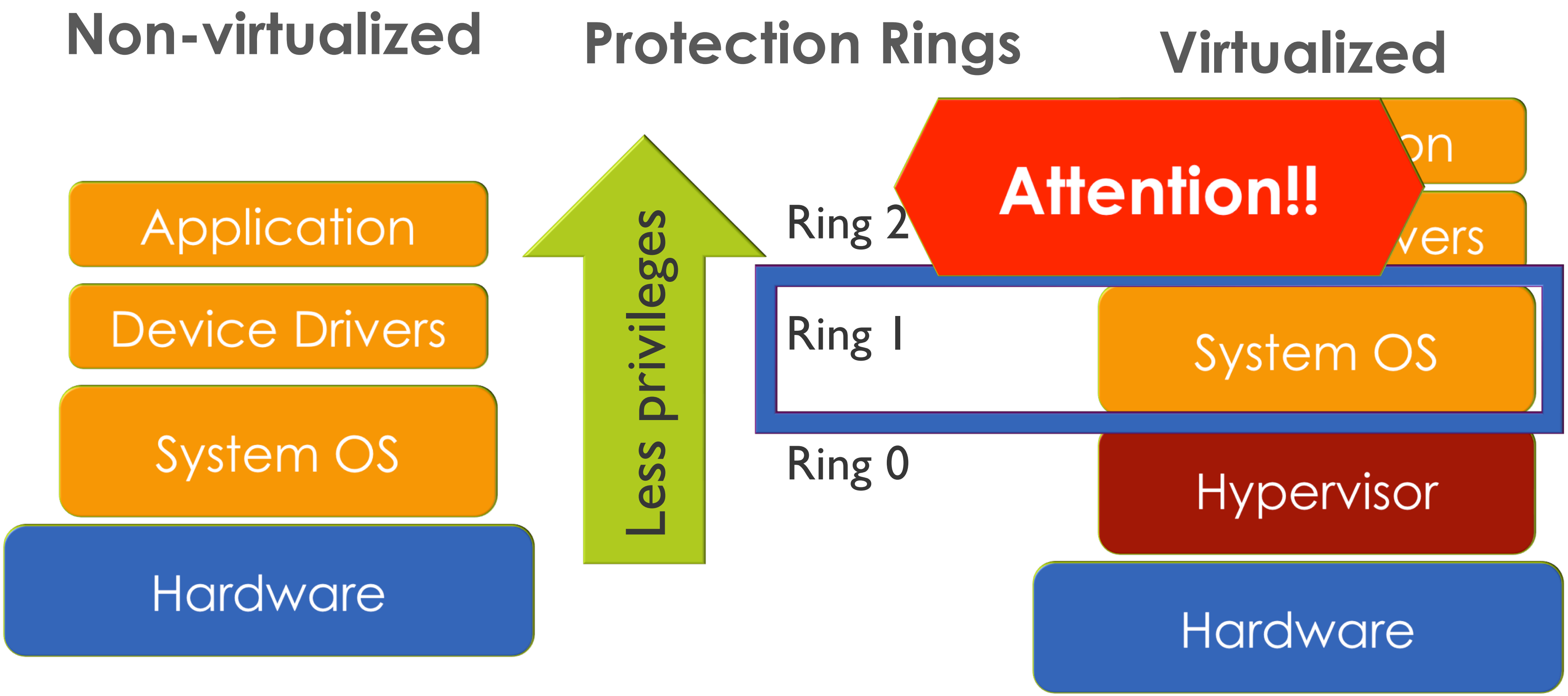
Virtualization, an overview

- Pros:
  - Several OS’s in the same machine
  - Security increase
  - Cost reduction
- Cons:
  - Too heavy
  - Too much memory consumption
  - Virtualization versus real time

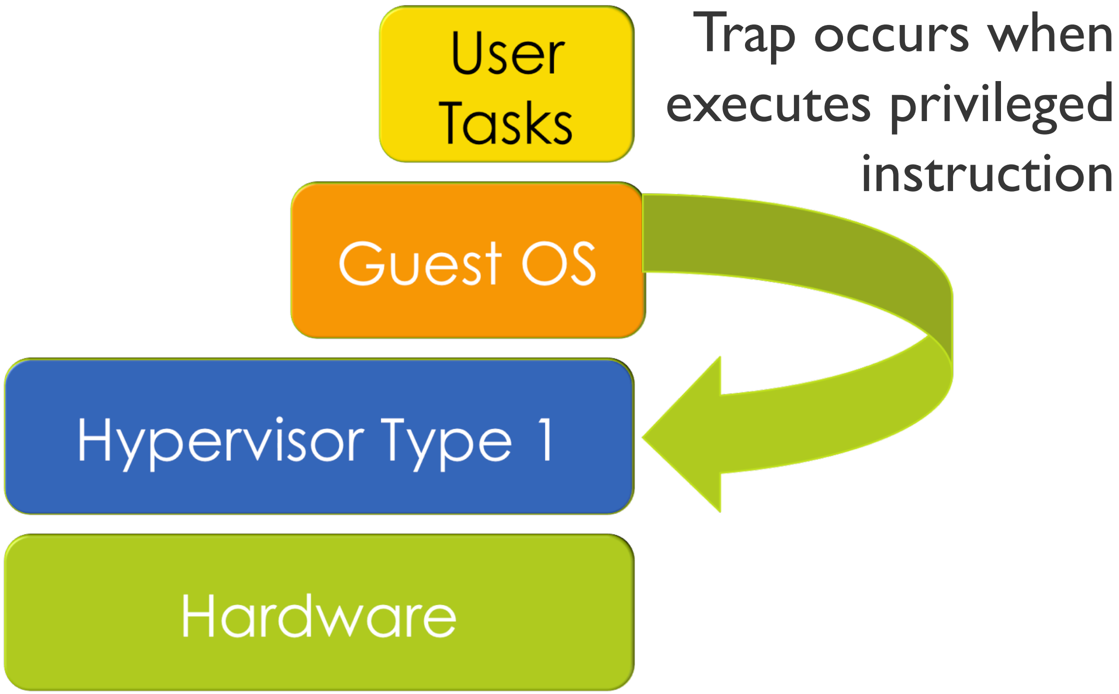
Hypervisor



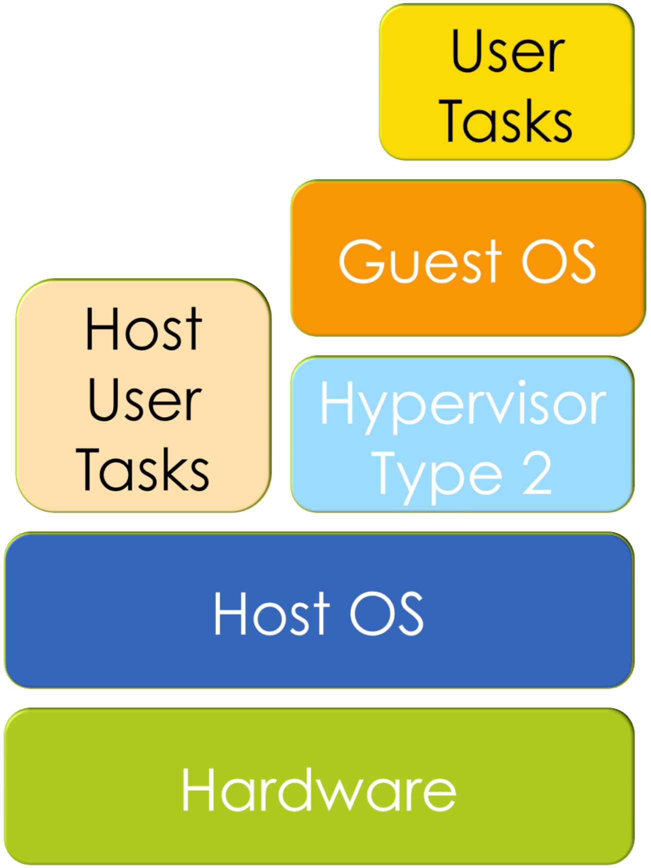
Hypervisor causes deprivileges



Hypervisor techniques to allow virtualization – Type 1

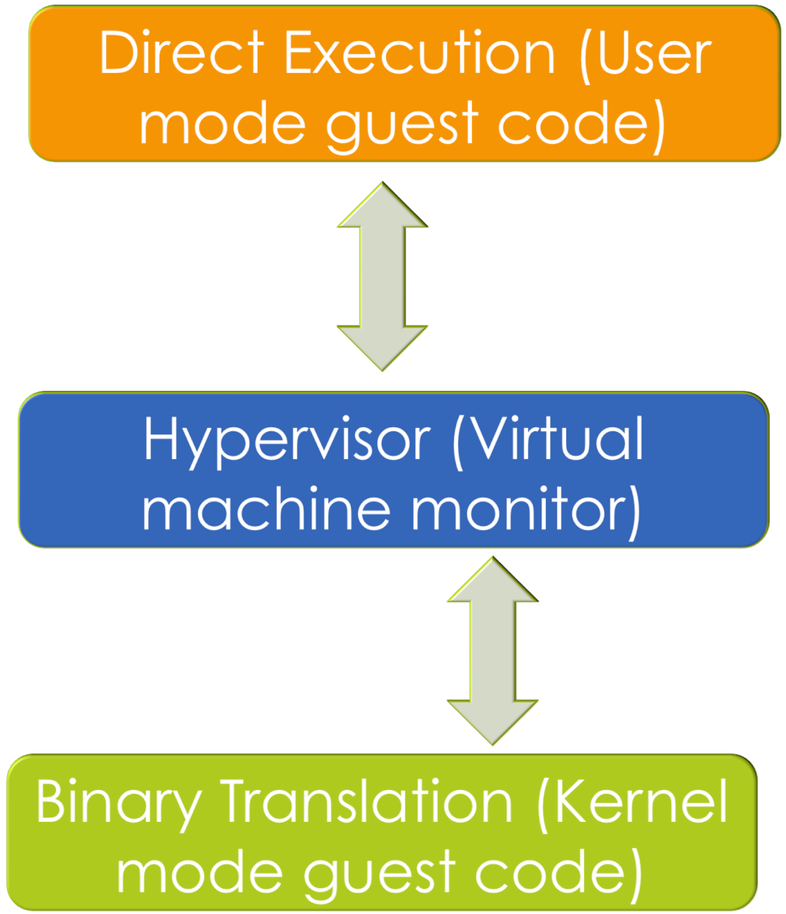


Hypervisor techniques to allow virtualization – Type 2



Some solutions: Binary Translation

- Mainly used by VMWare in the 90's
  - X86 to x86 translator
  - Translates, at run time, the guest OS binary code
  - Guest applications continue in its intended privilege ring



Binary Translation

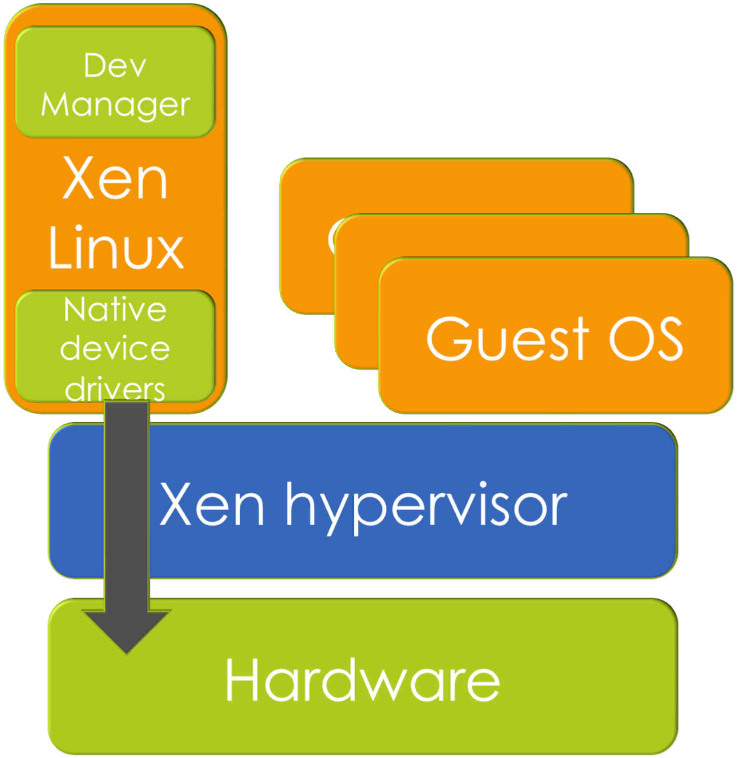
- Guest OS becomes and input to the translator
- Non-privileged instructions are simply copied (x86 to x86)
- Privileged instructions cause changes in the translated code
- Translated code can become way larger than the original

Some solutions: Paravirtualization

- Instead of translating the guest OS binary code it replaces sensitive instructions by explicit hypervisor calls (hypercalls)
- The hypervisor must offer an interface with possible system calls to be used by the guest OS
- However, the main idea is kept: avoid privileged instructions to run without proper permission

Paravirtualization

- Xen implementation:
  - Domain 0 for I/O operations
  - Appears to the other VMs as real native drivers requiring no emulation
  - Interesting concept, adopted by several other virtualization solutions

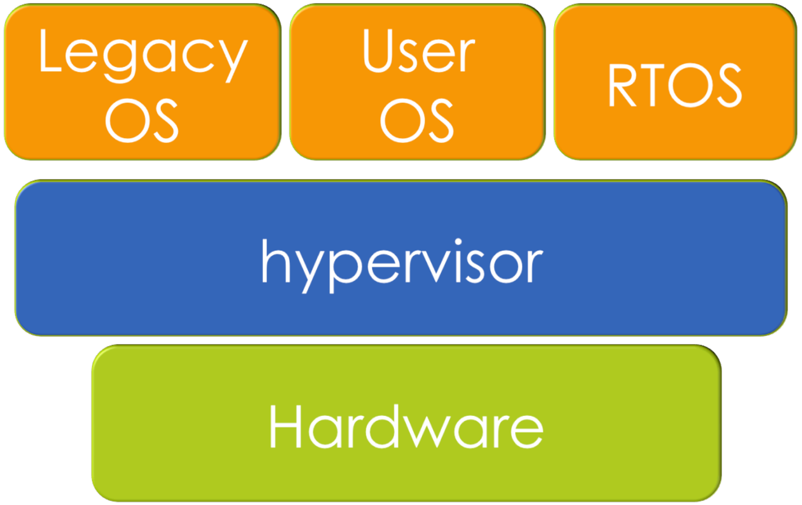


Hardware accelerated virtualization

- Main strategy: add a new execution mode
- Hypervisor can run safe and transparent during the execution of VMs
- Amount of traps is drastically reduced
- For real performance boost, improvements regarding memory management had to be made
  - TLB keeps track of different VM's mapping at the same time

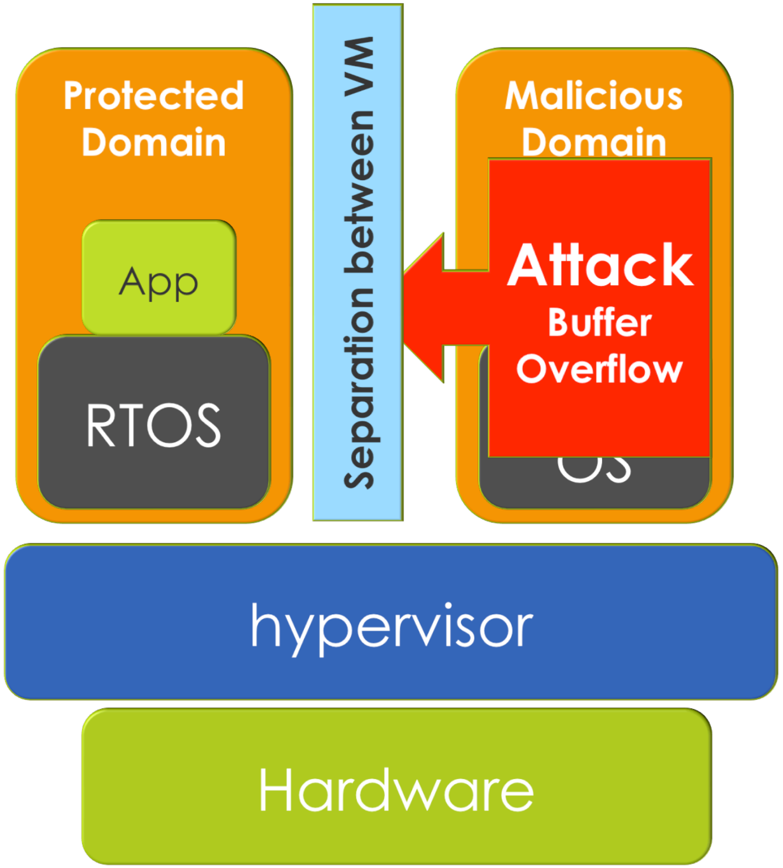
Motivational Examples for ES: SW quality

- First and direct advantage for virtualization: several OS's in the same physical machine
  - Legacy software must coexist with current and incompatible applications
  - Separation between RT and user interface application, by using different OS's



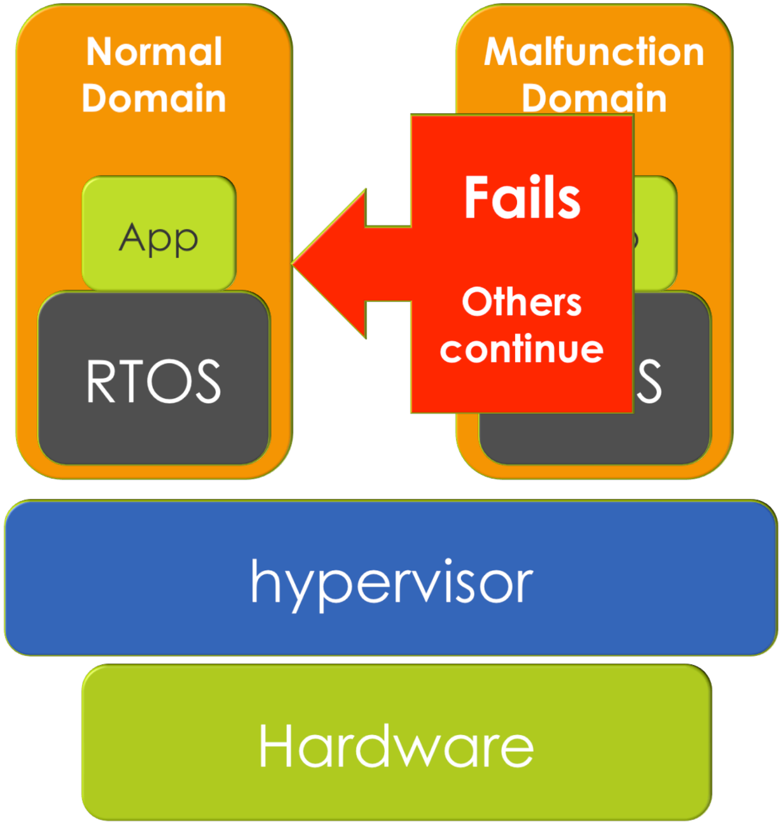
Motivational Examples for ES: Security

- Protects and encapsulates embedded OS's
- User attacks can have limited consequences by separated VM's
- Needs hypervisor secure implementation
  - One strategy: keep it as small as possible



Motivational Examples for ES: multicore

- Asymmetric Multiprocessing with different OS's in each VM
- Symmetric Multiprocessing with a single OS running onto multiple cores
- Independent restart of processors and VM's
- Improved reliability
  - in case of VM malfunction
  - In case of physical processor malfunction



In short…

- Software design quality
- Software reuse
- System safety
- Improved Reliability

Still…

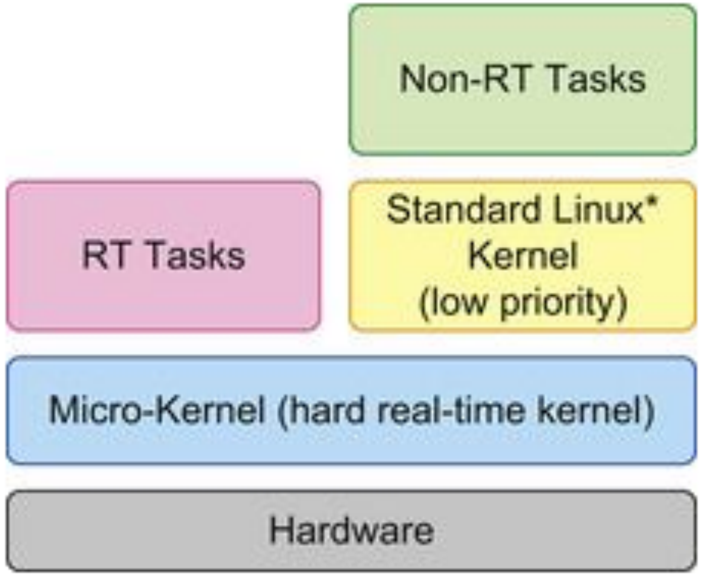
- Security-sensitive or mission-critical systems
  - As they need protected environment
  - Hypervisor separates trusted from untrusted OS's and applications
- Asymmetric arrangements can be used to allow some parts of the system to boot up faster than others
- License protection (proprietary and GPL isolation)

Still…

- Firmware over the air upgrades can be easier
- Separated reboots
- VM migration among devices (pervasive computing)

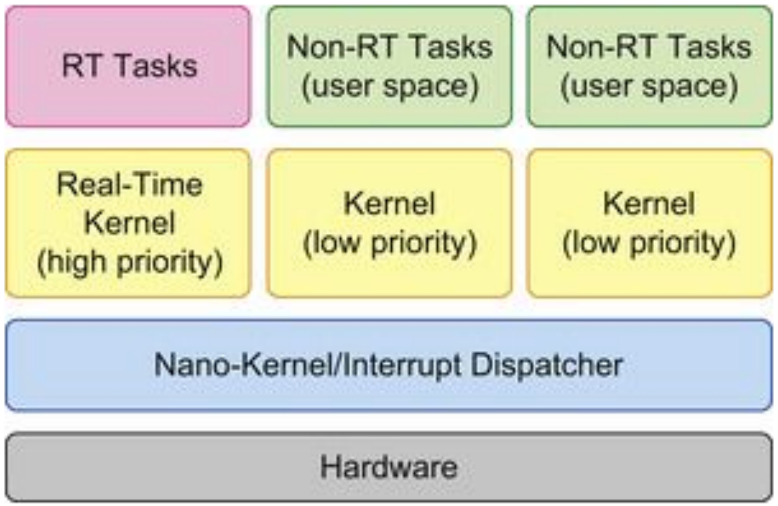
Some ES virtualization solutions

- Xenomai
  - Linux-based
  - Real-time (not hard)
  - Scheduler of real-time kernel treats standard linux as an idle task
  - linux processes can be preempted at any time



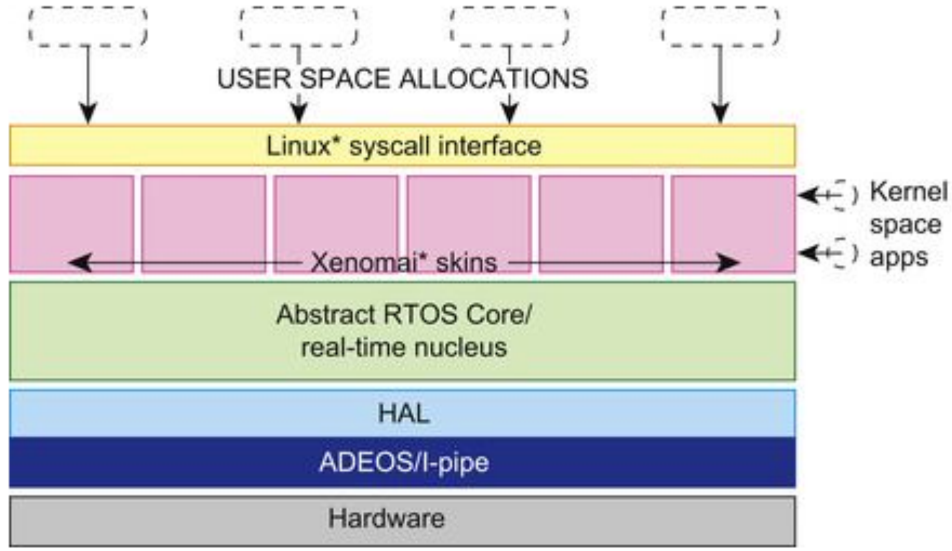
Some ES virtualization solutions

- Xenomai
  - Interrupt management handled by real-time kernel
    - If interrupt occurs in the middle of a real-time task, it is stored
    - Only when real-time kernel is done, the interrupt is handed over the standard linux kernel



Some ES virtualization solutions

- Xenomai
  - Provides API for RT tasks, timers and sync objects
  - Emulates popular RTOS APIs through *skins* abstraction to ease porting

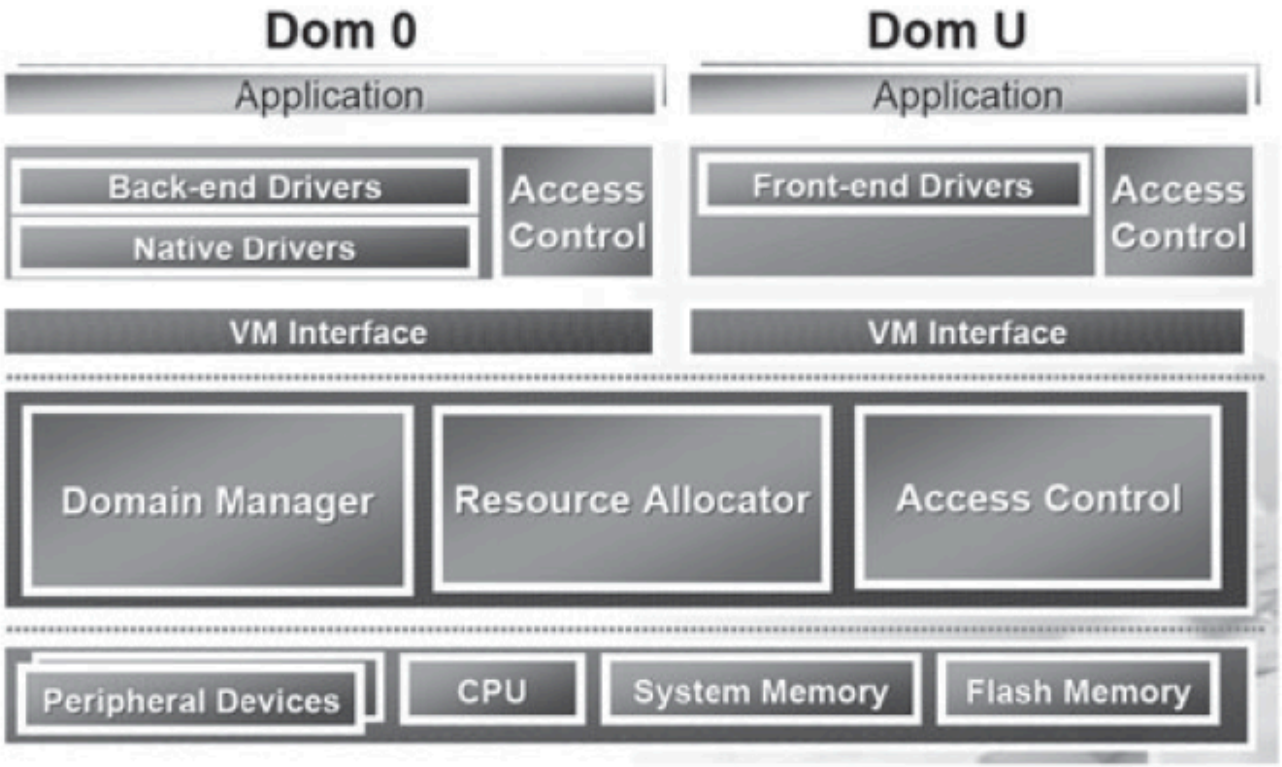


Some ES virtualization solutions

- EmbeddedXEN
  - Academic project for embedded RT application
  - Executed in ARM cores
  - Based in XEN hypervisor for general purpose computers
  - Creates a page table for each guest OS (when virtual memory support is available)
    - Aims to provide strong and straightforward isolation among VM's
  - Access control prevent users from accessing given processes in kernel space

Some ES virtualization solutions

- EmbeddedXEN approach

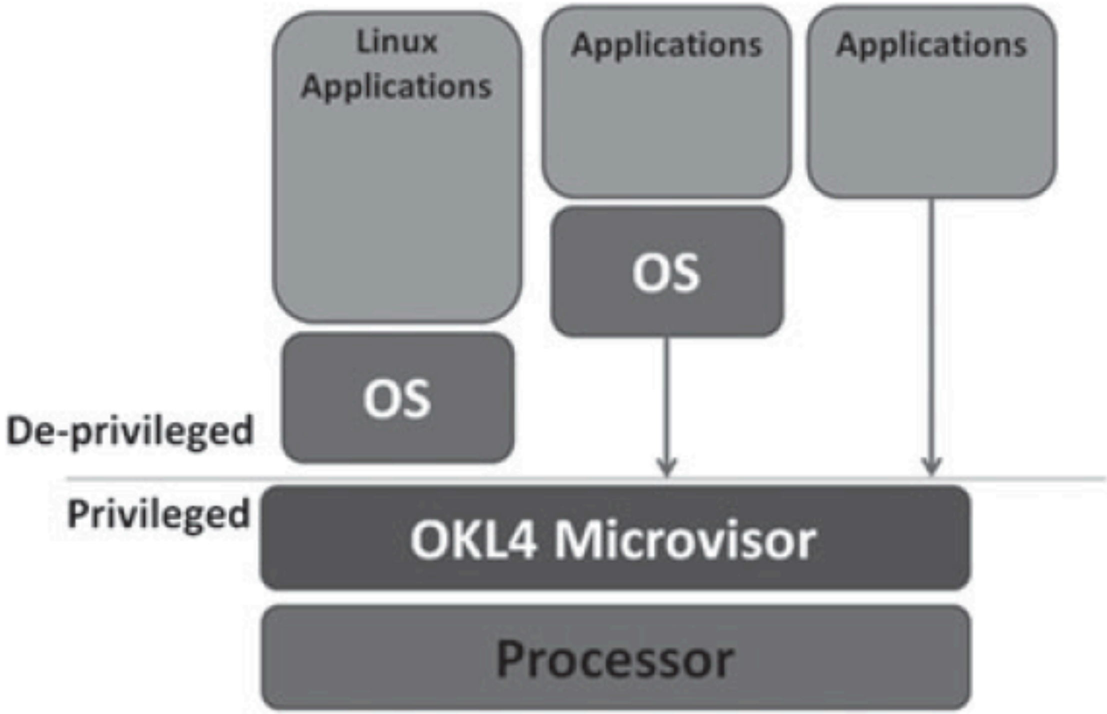


Some ES virtualization solutions

- OKL4
  - L4 family microkernel with low overhead rates
  - High-performance IPC message mechanism
  - Low overhead virtualization
  - Memory sharing strategy
    - Memory regions can be shared by different address spaces

Some ES virtualization solutions

- OKL4

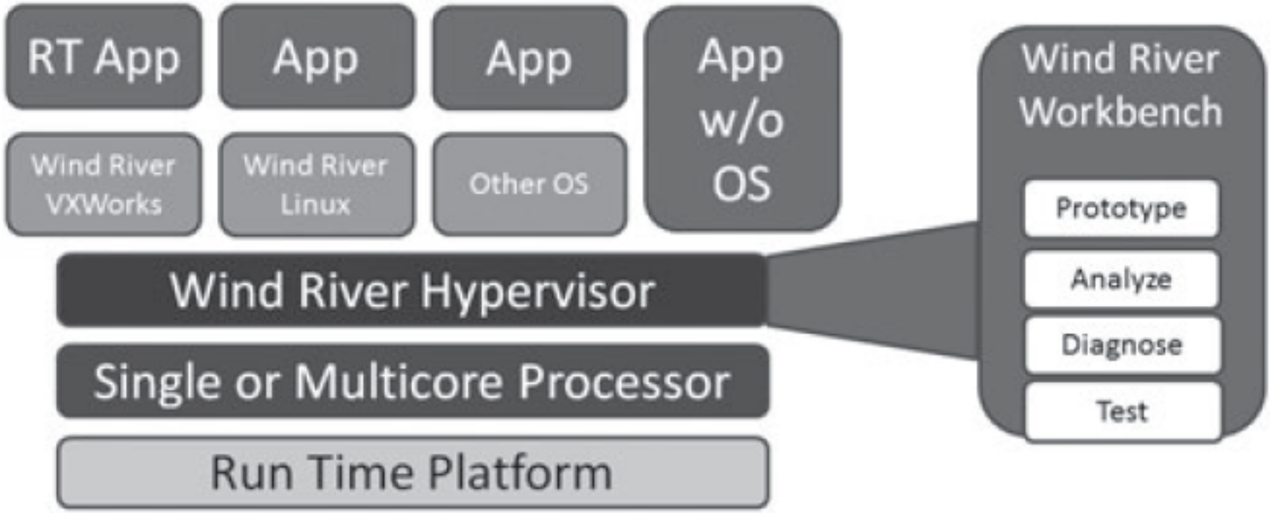


Some ES virtualization solutions

- WindRiver
  - Supports single and multicore processor
  - X86 and powerPC
  - Integrates with VxWorks and Wind River Linux
  - Enables devices to be assigned for VM's
  - Communication through message-passing
  - Allows VM restart

Some ES virtualization solutions

- WindRiver

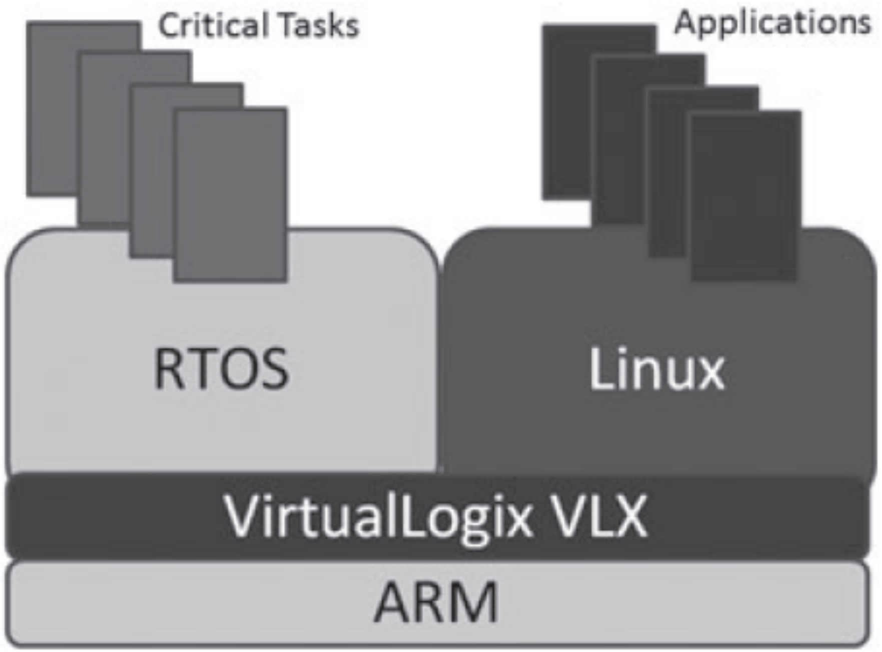


Some ES virtualization solutions

- VirtualLogic VLX
  - ARM and x86
  - Several SW: android, linux, proprietary, symbian
  - OS/device independence
  - Separation of design
  - Includes advanced system level policies for scheduling, memory, power and security management.

Some ES virtualization solutions

- VirtualLogic VLX

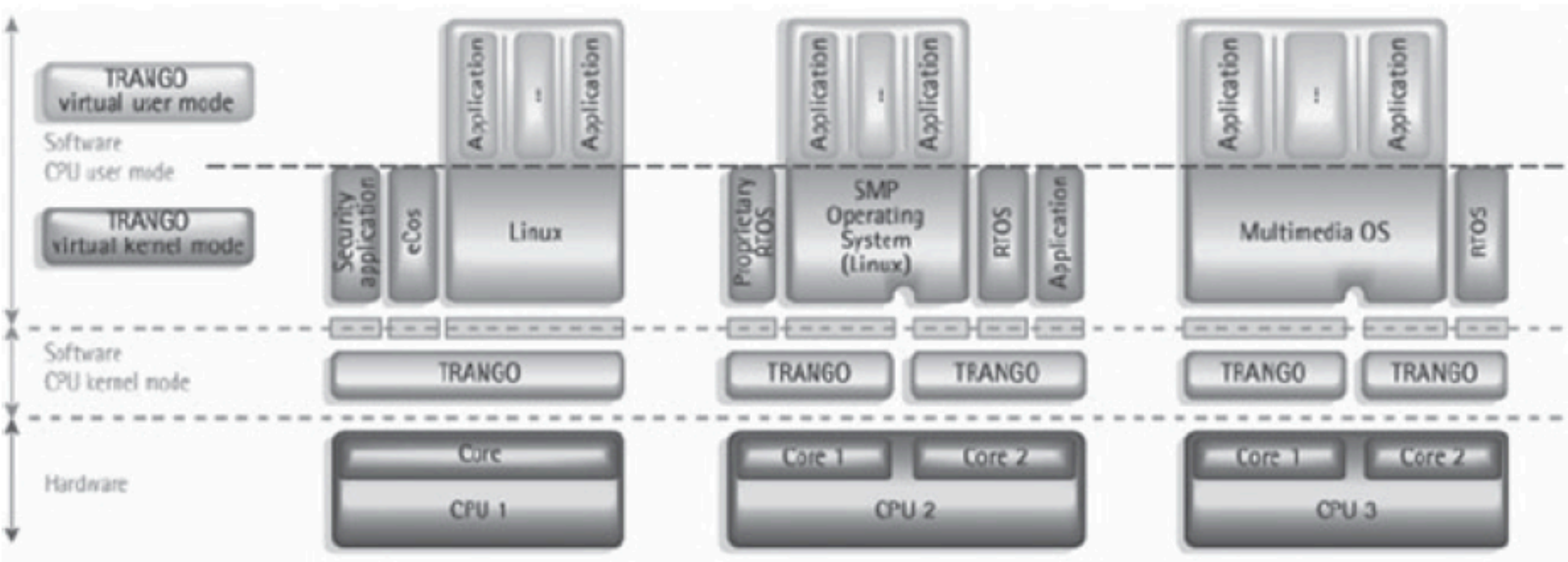


Some ES virtualization solutions

- Trango
  - Wide device application
    - Dvd players, printers, routers, set-top box
  - Allows integration of multimedia, real-time and trusted applications
  - Reduces production and development costs
  - Supports SMP OS's

Some ES virtualization solutions

- Trango

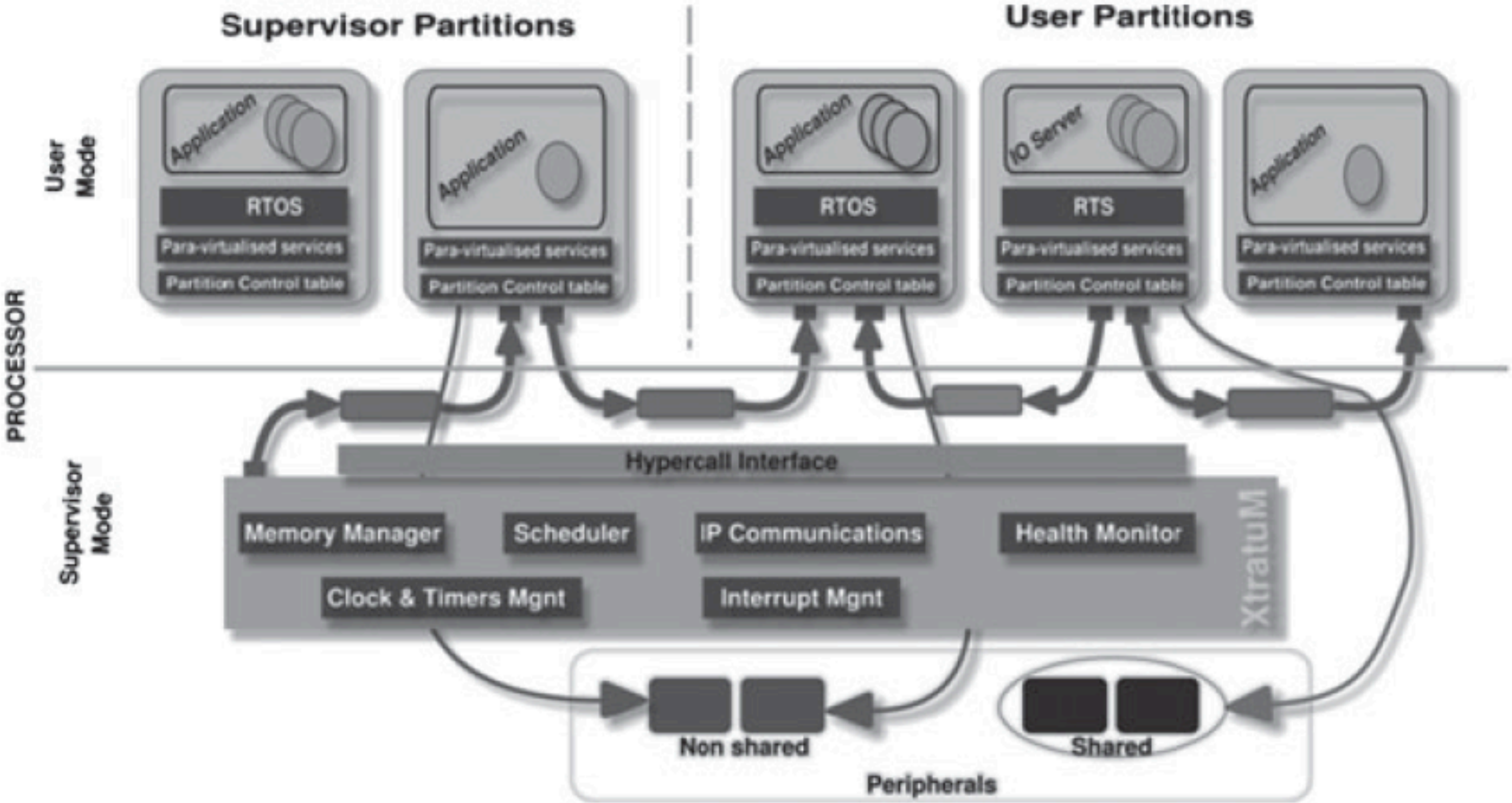


Some ES virtualization solutions

- XtratuM
  - Strong temporal isolation, implemented as a fixed cyclic scheduler
  - Strong spatial isolation, that is, all partitions are executed in processor user mode and do not share memory
  - Basic resource virtualization, such as clock and timers, interrupts, memory, CPU, and special devices
  - Real-time scheduling policy for partition scheduling
  - Efficient context switch for partitions
  - Deterministic hypercalls (hypervisor system calls)

Some ES virtualization solutions

- XtratuM

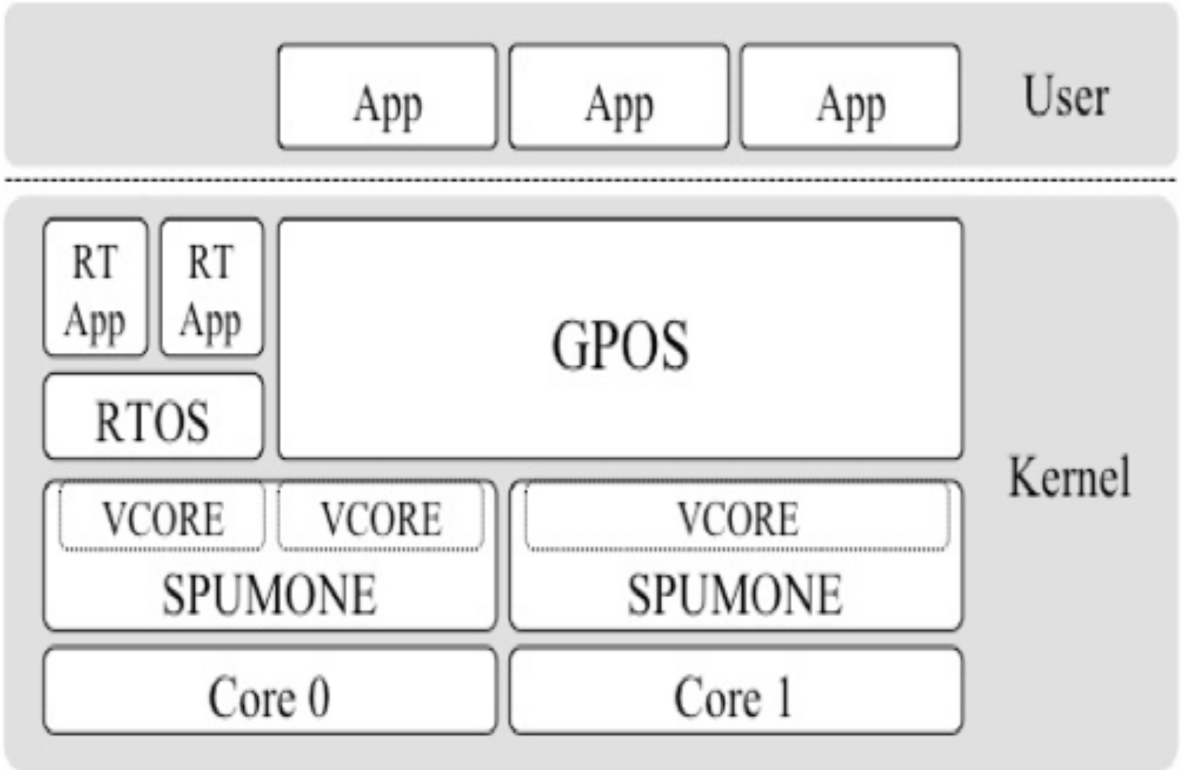


Some ES virtualization solutions

- SPUMONE
  - Lightweight virtualization layer
  - Uses a simpler inter-VM communication
  - Aims to reduce virtualization overhead
  - Only CPU is virtualized (not devices)
    - Devices are shared between guest OS's

Some ES virtualization solutions

- SPUMONE



Challenges

- Some limitations are present in GP computing and are inherit to virtualization
- Great challenge: hardware heterogeneity
  - ARM, MIPS, PowerPC…
  - Therefore, it may be possible to coexist different virtualization solution, proper for each architecture
- Application heterogeneity
  - Again, different techniques may be used

Challenges

- Limited resources
- Real-time workload
- Complexity and costs involved in developing a suitable virtualization layer for different embedded segments is too high
- Specific VM's are still the option to reduce overhead

Challenges

- Embedded systems are highly integrated and connected
- Sub-systems should communicate efficiently
- Virtualization isolates sub-systems
- Virtual network schemes can cause too heavy overhead

Challenges

- Software complexity
- Embedded systems have rich applications
  - Large
  - Probably with many bugs
  - Updates can introduce even more bugs
  - If application failure affects VM, it can cause system degradation

Challenges

- Scheduling matters
- How to deal with devices where real-time applications need to coexist with rich functionality applications?
- Guest OS has no idea whatsoever about the hypervisor scheduling
- Different non real-time OS's may need different priority levels

Challenges

- Summarizing the requirements discussed, we can highlight the following:
  - A small hypervisor with support for multiple VMs
  - High-bandwidth, low-latency communication between system components
  - Minimal impact on system resources as well as real-time performance
  - Scheduling policy between VMs and support for real-time system components

InterOS Communication

- Over the time, virtualization technology focuses mainly on isolating VMs
- VM's coexistence become safer
- This is bad for communication intensive distributed tasks
- Embedded applications with communication needs must be answered quickly
- Virtual Ethernet x Shared Memory

InterOS Communication

- Requirements
  - Performance:
    - Communication mechanism should yield high throughput, low latency and low CPU consumption
    - Needed metric for ES
  - Transparency:
    - Mechanism should present a transparent interface
      - Minimal or no change to the guest OS or application

InterOS Communication

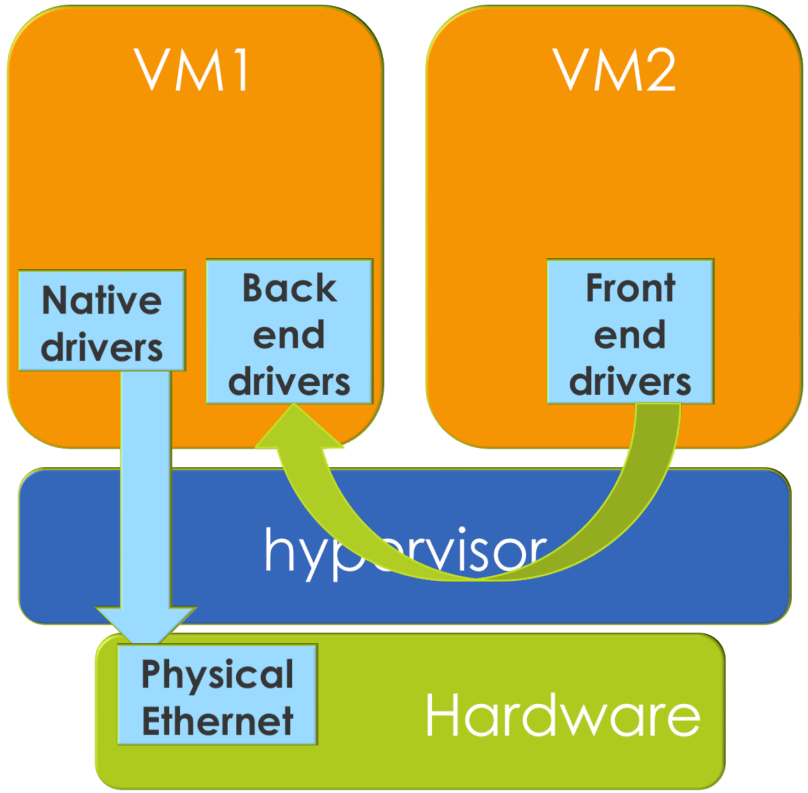
- Requirements
  - Dynamism:
    - The communication mechanism should be enabled or disabled without any changes to the functionality of the VM
  - Non-intrusiveness:
    - The mechanism should be designed to be deployed without requiring modification to the guest OS or the hypervisor

InterOS Communication

- Virtual Ethernet
  - Logical ethernet device provided by the hypervisor to a guest OS
  - Guest OS uses the virtual Ethernet interface to communicate to other guest OS's and to the rest of the system
  - Indirectly, virtual ethernet depends on physical device to perform its communication

InterOS Communication

- Virtual Ethernet
  - An example where one VM (VM1), more privileged owns the device
  - Split device driver architecture

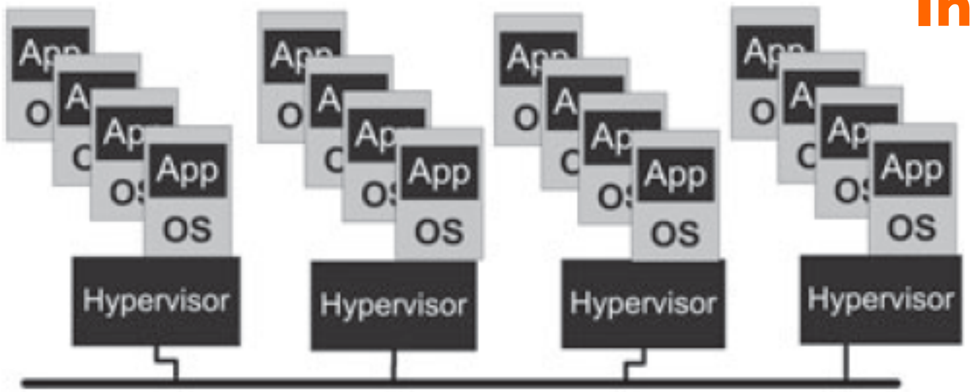


InterOS Communication

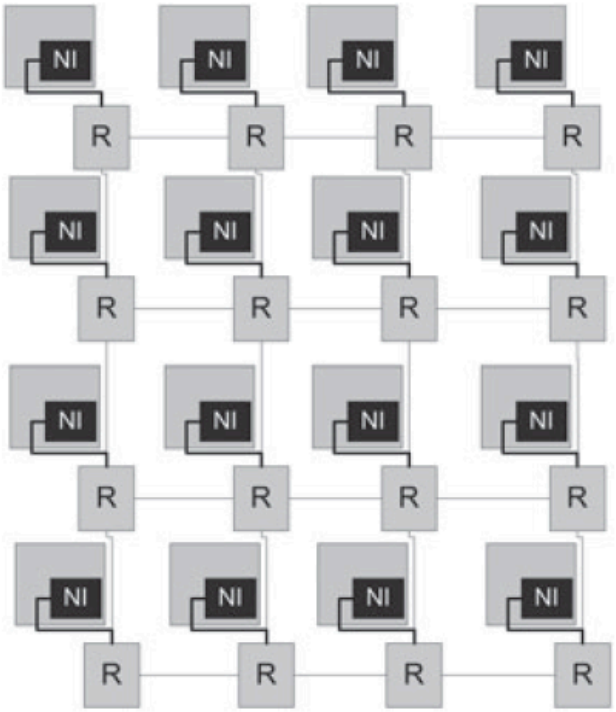
- Shared memory
  - Provides a block of physical memory for common concurrent access
  - Very fast
  - Well suited for virtualization
  - Limited scalability

Ideas

- Massively Parallel MPSoC

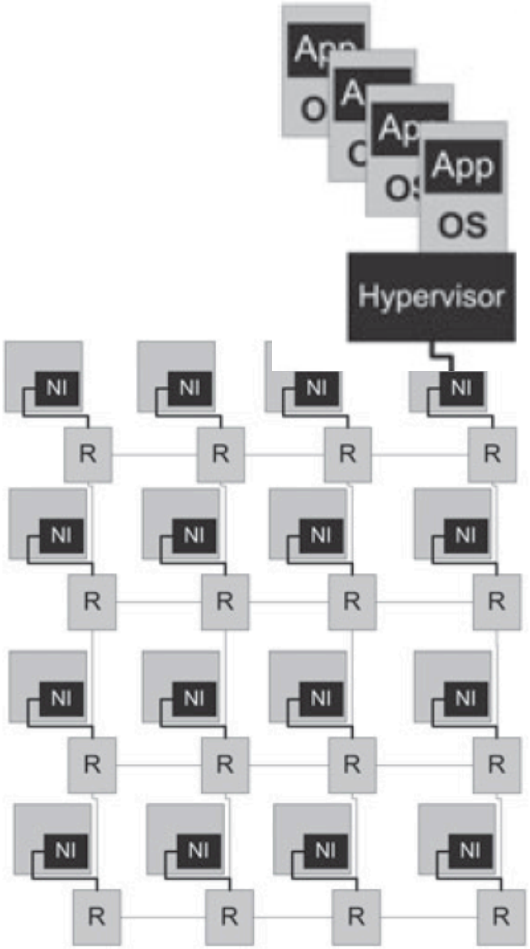


Instead of



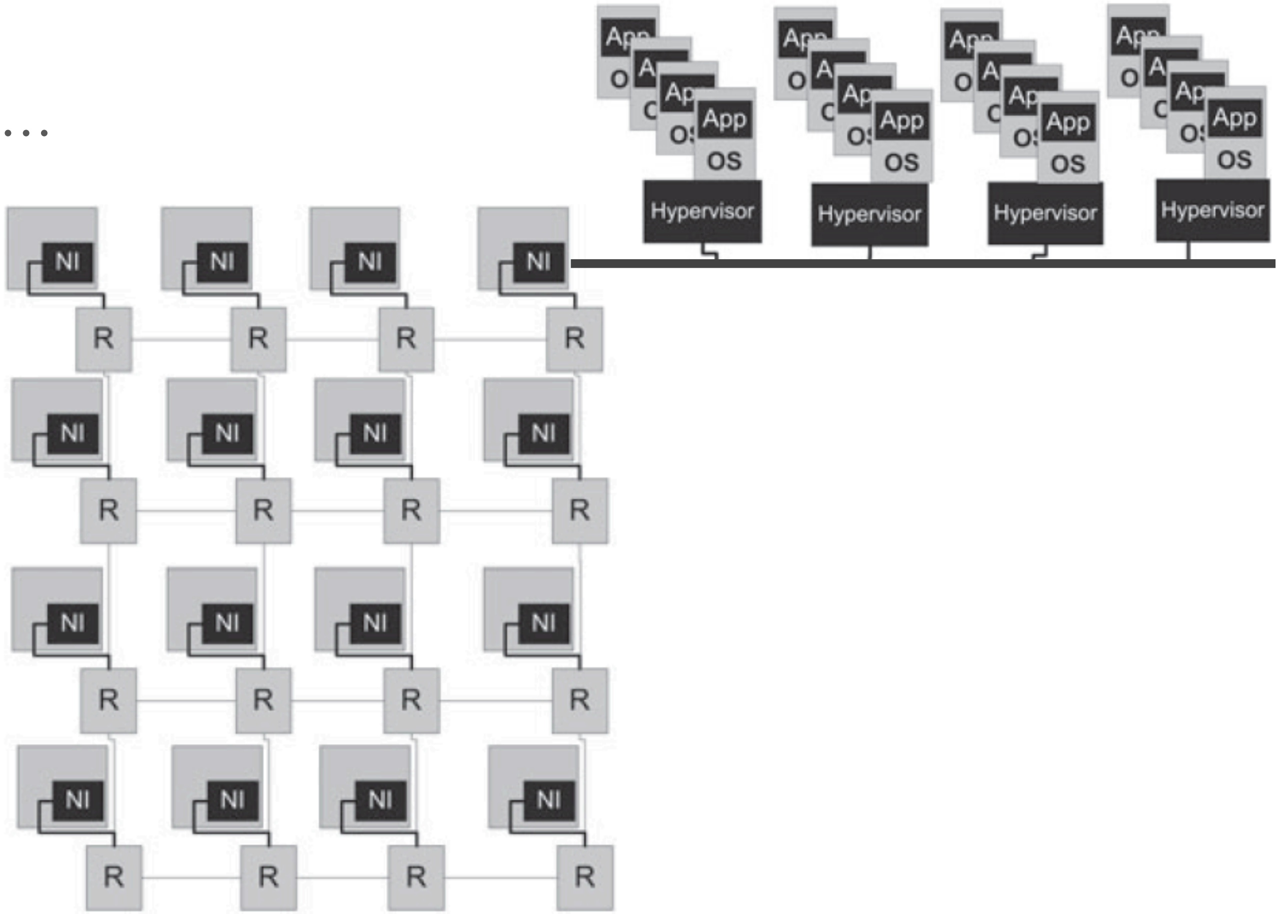
Ideas

- Or…



Ideas

- Or…



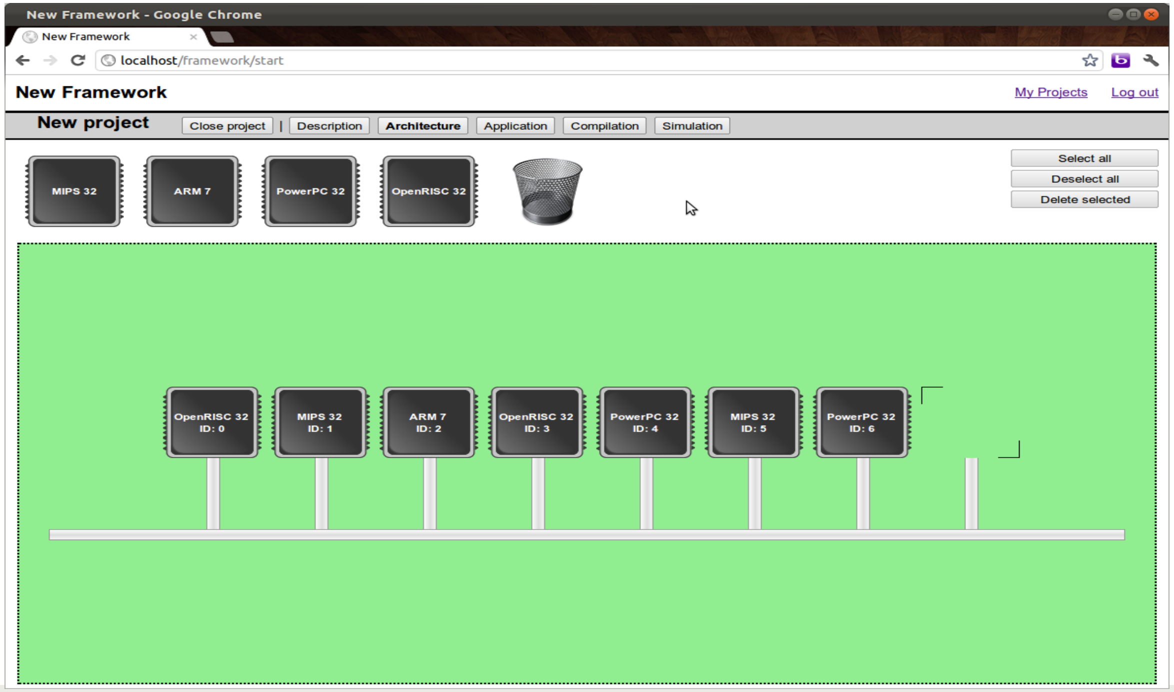
Challenges

- To run unmodified GuestOS and applications
- To provide strong spatial isolation to improve security
- To have low overhead components

- The implementation totally depends on the underlying hardware
- Different architecture options are available
- Very difficult to target all such constraints at once

HellFire Framework

- Different embedded processors
- Up to 128 processors can be used in the design, connected by a bus or NoC
- Run over virtual platform (eg, OVP)

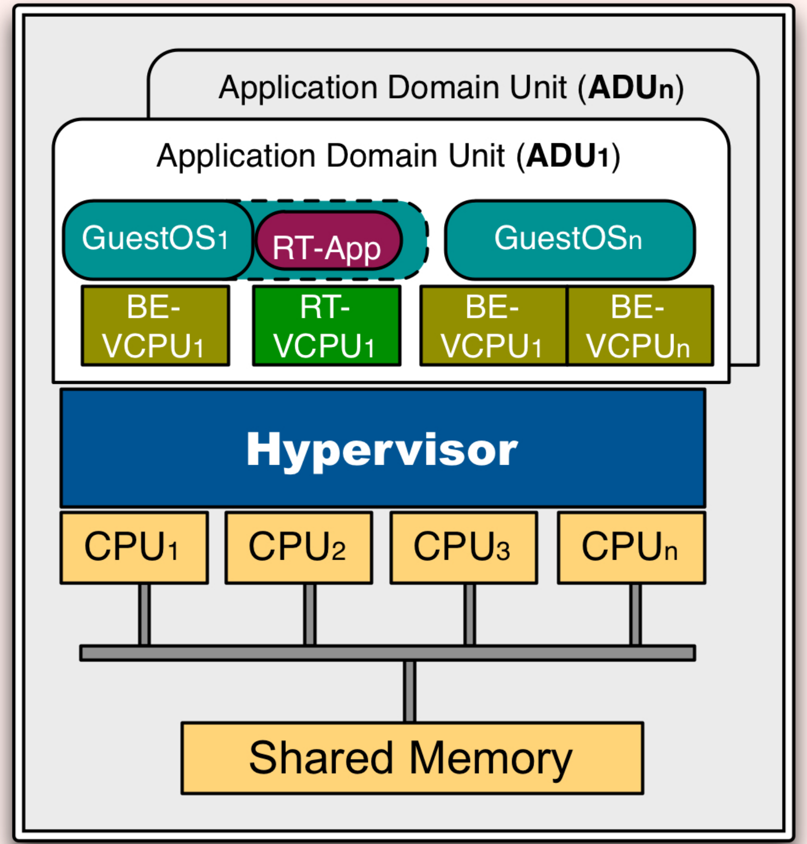


Objectives

- Add virtualization support to a MIPS-based architecture
  - Why MIPS? Because is widely adopted
  - Which MIPS? MIPS4k
- Memory protection among virtual machines is mandatory
- No changes in the GuestOSs are desired
- Full virtualization support
  - Already a reality for embedded systems, ARM announced in 2011 its own architecture support

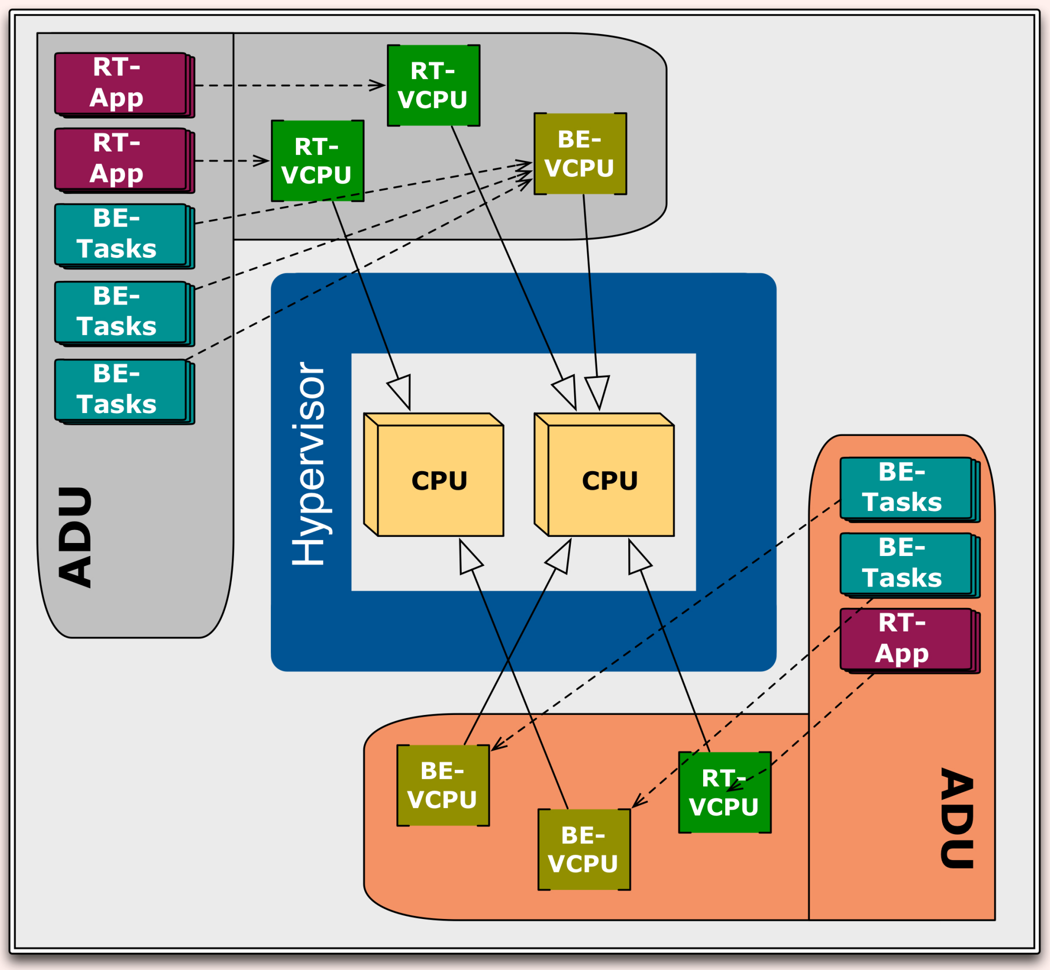
Virtualization Model

- Application Domain Unit (ADU)
  - Corresponds to a virtual machine
- GuestOS and Tasks
  - No modifications are need in the source code
- Hypervisor
  - Manages the creation and execution of VCPUs and ADUs
  - Efficient scheduling: Physical units are always aware of the next VCPU
- Virtual Processing Unit (VCPU)
  - Individually scheduled onto physical units
  - Several VCPUs can compose one ADU
- Physical Processing Unit (PPU)
  - MIPS-based platform



Dynamic Mapping

- Tasks among VCPUs
- VCPUs among PPUs



Conclusion

- Can bring innumerous advantages for ES:
  - Allow several OS's
  - Reduce manufacturing cost
  - Improve security and reliability
  - Decreases software development complexity
- Several challenges should be addressed
- Existing solutions and promising field

Discussion

Embedded System's Virtualization – Concepts, issues and challenges

ESSE 2012 - Embedded Tutorials

November 08, 2012 – 9:00am – 1:00pm - Room: TBD

Embedded System's Virtualization

Concepts, issues and challenges

Fabiano Hessel (speaker) and Alexandra Aguiar

{fabiano.hessel, alexandra.aguiar}@pucrs.br

Faculty of Informatics, PUCRS, Porto Alegre Brazil