

Embedded Virtualization for Mobile Devices

Jim Huang (黃敬群) <jserv@0xlab.org>

Developer, 0xlab

March 29, 2012 / 中國科學技術大學軟件學院（蘇州）

Rights to copy

© Copyright 2012 **0xlab**

<http://0xlab.org/>

contact@0xlab.org



Attribution – ShareAlike 3.0



You are free

- ✓ to copy, distribute, display, and perform the work
- ✓ to make derivative works
- ✓ to make commercial use of the work

Corrections, suggestions, contributions and translations are welcome!

Latest update: Mar 29, 2012

Under the following conditions

-  **Attribution.** You must give the original author credit.
-  **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>



Goals of This Presentation

- Give you an overview about
 - virtualization in general
 - device virtualization on ARM
 - doing virtualization in several approaches
- We will not discuss
 - language runtimes
 - how to use VMware/Xen/KVM/...



Agenda

(1) Motivations

(2) Hypervisor Design

(3) Embedded Hypervisors for
ARM



Motivations of enabling virtualization for embedded devices



Definition

"**virtualization** is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. "

– Wikipedia

Future Computing Trends

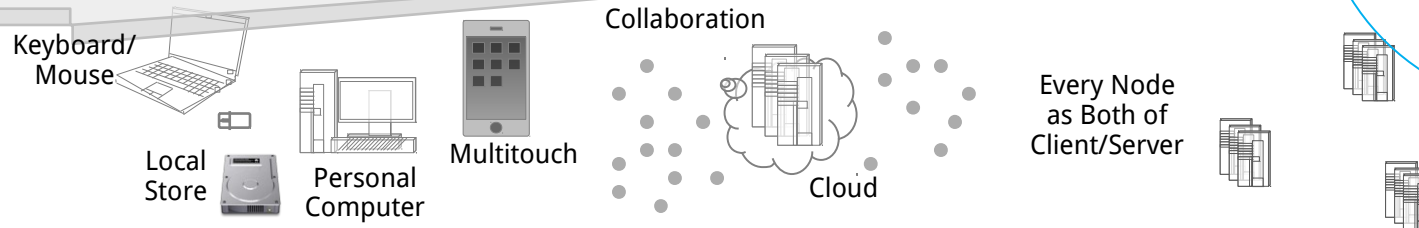
Changes in Computing

Closed
Centralized
Correct Info.
Stationary

- Keyboard/Mouse
- Voice Call, SMS
- Centralized/Concentrated
- Known Comm. Entities
- Multitouch
- Video Call, MMS
- Augmented Reality
- Eye-Tracking
- Distributed/Scattered
- Unknown/Utrusted Comm. Entities
- Gesture
- Manytouch
- Interactive 3D UI
- Realtime Web

Open
Distributed
Correct+Timely
Info.
Mobile

Sensor Network



Embedded

Single-core

Multi-core

Many-core

IT

Single-core

Multi-core

Many-core

UC Berkeley
SensorNet Chip
(TI MSP430 8MHz
core, 10KB RAM)

[2009]

- Tiger 1GHz Single-Core
- Dunnington 3GHz 6-core

[2012]

- ARM 2GHz 4-core
- Intel 4GHz 32-core

[2017]

- ARM 3GHz 8-core
- Intel 6GHz 128-core
- SensorNet Chip
(128MHz core, 160KB RAM)

Privacy

Realtime



Source: **Xen ARM Virtualization**, Xen Summit Asia 2011
by Dr. Sang-bum Suh, Samsung

Server Virtualization::Benefits

- Workload consolidation
 - Increase server utilization
 - Reduce capital, hardware, power, space, heat costs
- Legacy OS support
 - Especially with large 3rd-party software products
- Instant provisioning
 - Easily create new virtual machines
 - Easily reallocate resources (memory, processor, IO) between running virtual machines
- Migration
 - Predicted hardware downtime
 - Workload balancing



Embedded Virtualization::Benefits

- Workload consolidation
- Flexible resource provisioning
- License barrier
- Legacy software support
 - Especially important with dozens or hundreds of embedded operating systems, commercial and even home-brew
- Reliability
- Security



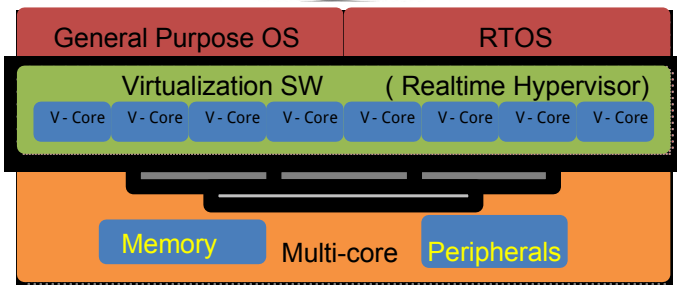
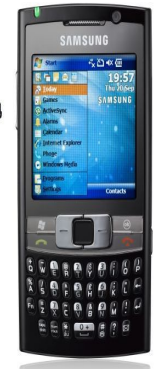
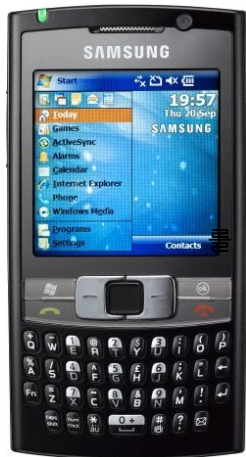
Why?

(1) Hardware Consolidation

- Application Processor and Baseband Processor can share multicore ARM CPU SoC to run both Linux and RTOS efficiently.

(2) OS Isolation

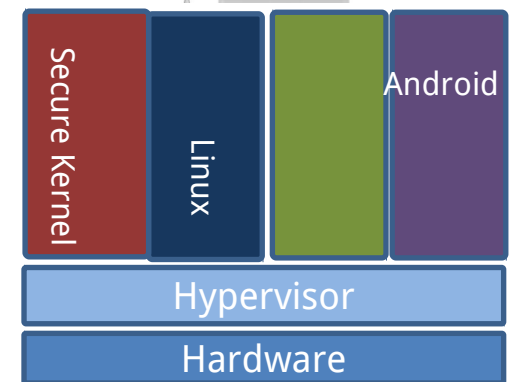
- important call services can be effectively separated from downloaded third party applications by virtualized ARM combined with access control.



AP SoC + BP SoC → Consolidated Multicore SoC



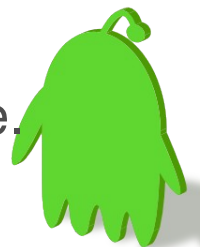
Secure Smartphone



Rich Applications from Multiple OS

(3) Rich User Experience

- multiple OS domains can run concurrently on a single smartphone.



Source: **Xen ARM Virtualization**, Xen Summit Asia 2011
by Dr. Sang-bum Suh, Samsung

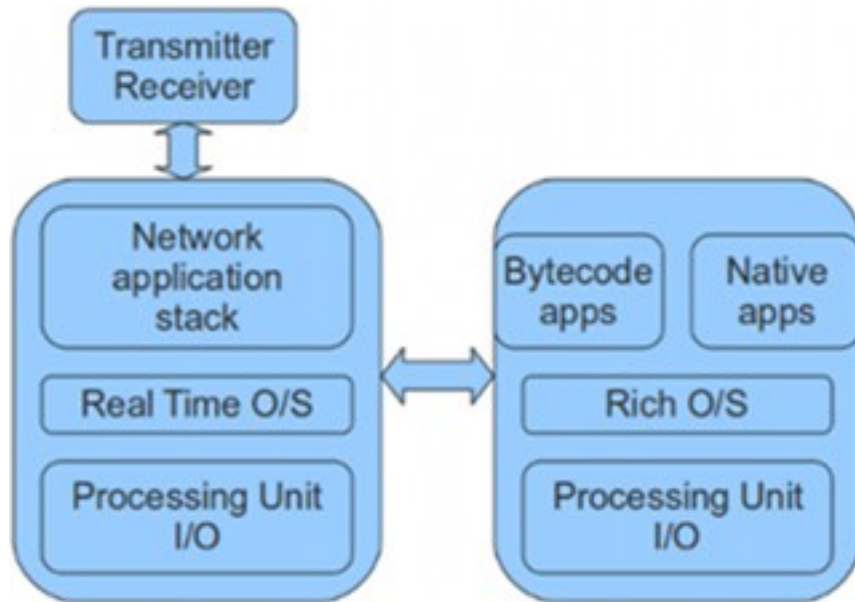
Use Case: Low-cost 3G Handset

- Mobile Handsets
 - Major applications runs on Linux
 - 3G Modem software stack runs on RTOS domain
- Virtualization in multimedia Devices
 - Reduces BOM (bill of materials)
 - Enables the Reusability of legacy code/applications
 - Reduces the system development time
- Instrumentation, Automation
 - Run RTOS for Measurement and analysis
 - Run a GPOS for Graphical Interface
- Real cases: Motorola Evoke QA4

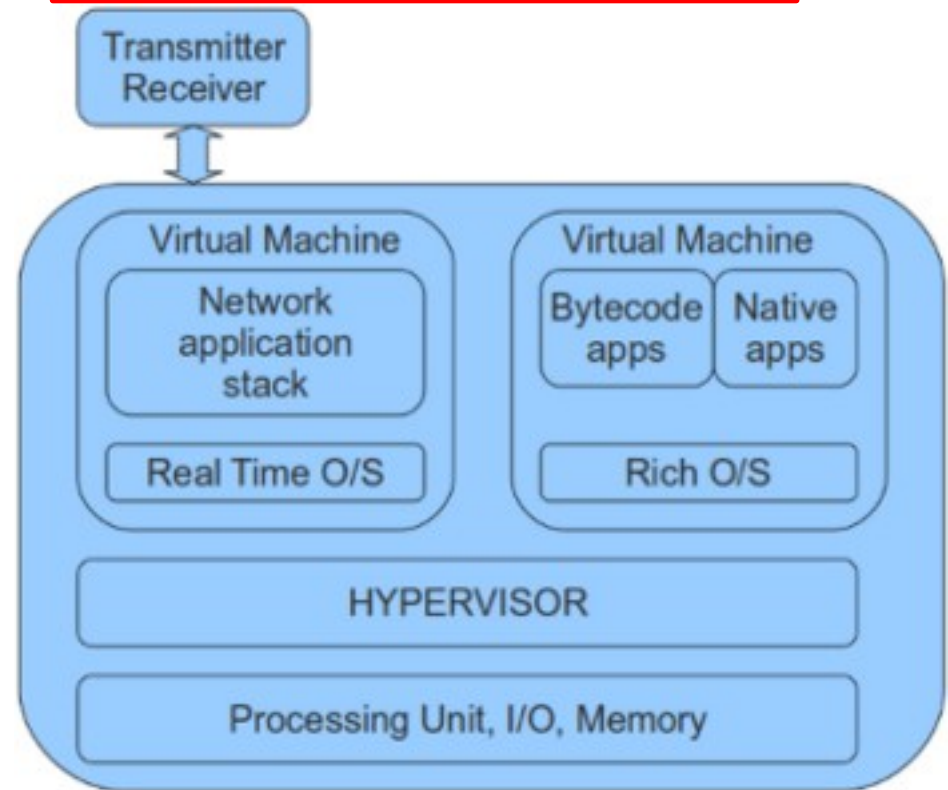
Hypervisor



original mobile phone:
two CPUs required



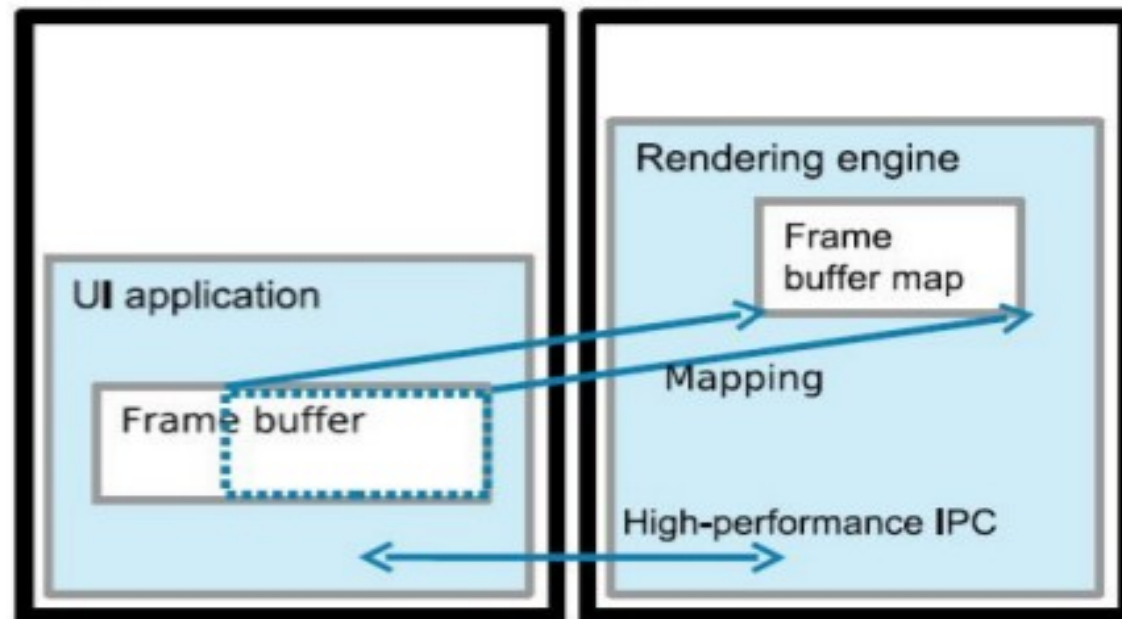
with Virtualization: single chip



- Evoke's UI functionalities including the touch screen is owned by the Linux apps while video rendering uses a rendering engine running on BREW.
- When a user requests a BREW app, Linux communicates with BREW in the other VM to start up the app. The BREW obtains access to the screen by using a frame buffer from a shared-memory mapping.

Linux VM

BREW VM



Example: Ubuntu for Android

Mobile and Desktop Virtualization in One Device
by VMware and Canonical

<http://www.ubuntu.com/devices/android>

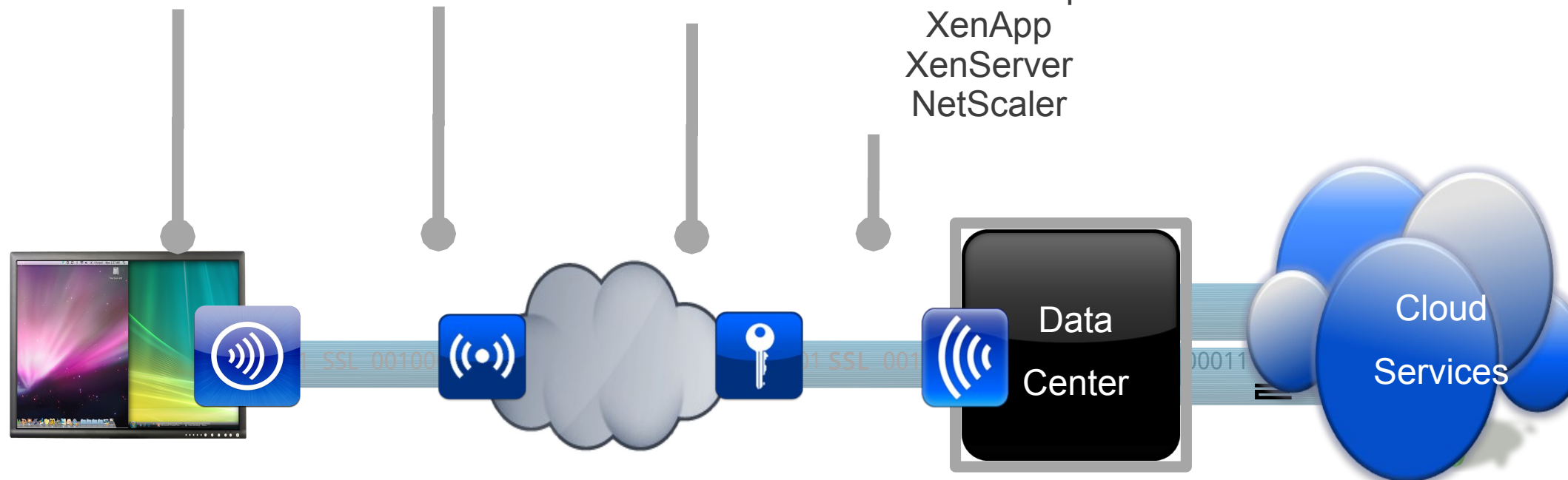


Use Case: Nirvana:

The Convergence of Mobile and Desktop
Virtualization in One Device
by OKLabs + Citrix



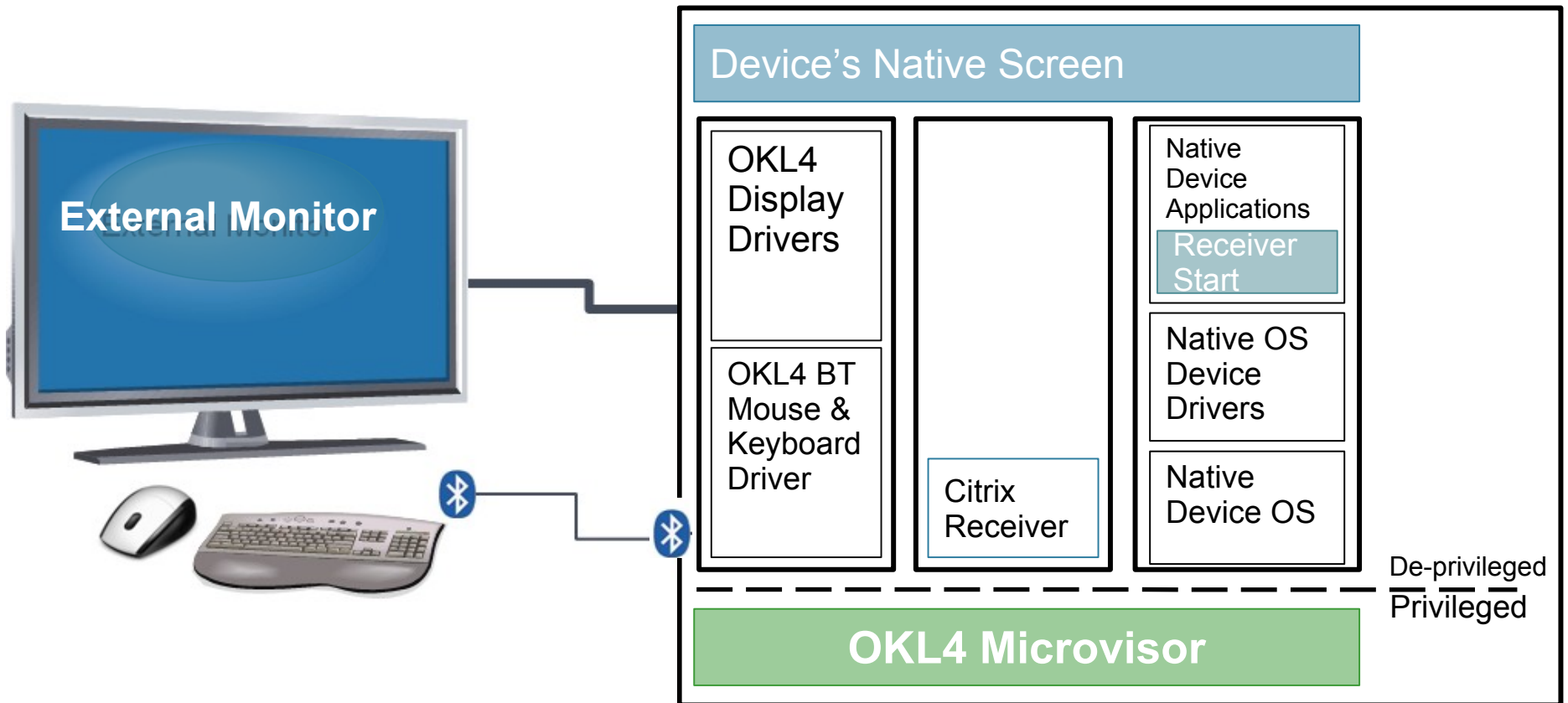
XenDesktop
XenApp
XenServer
NetScaler



- Nirvana phone = Smartphone
- + Full-sized display
 - + Keyboard & mouse
 - + Virtual desktop
 - + OKL4 mobile virtualization

Nirvana Phone

Mobile Device



Demo video:

<http://www.youtube.com/user/OpenKernelLabs>



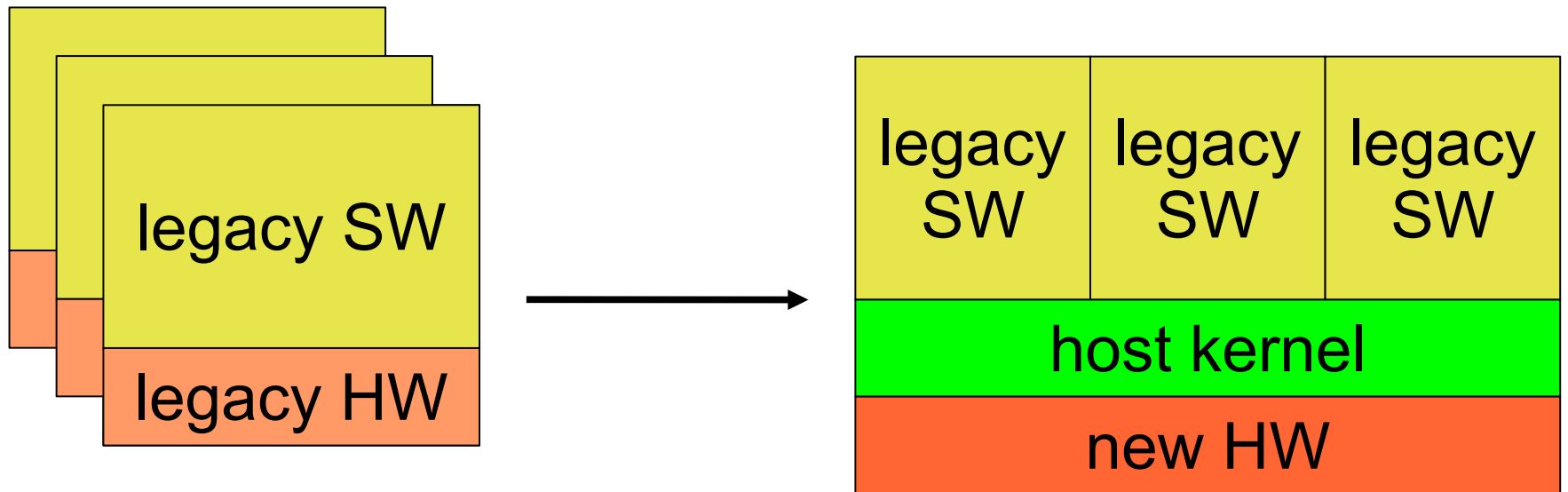
Embedded Virtualization Use Case

- Workload consolidation
- Legacy software
- Multicore enablement
- Improve reliability
- Secure monitoring



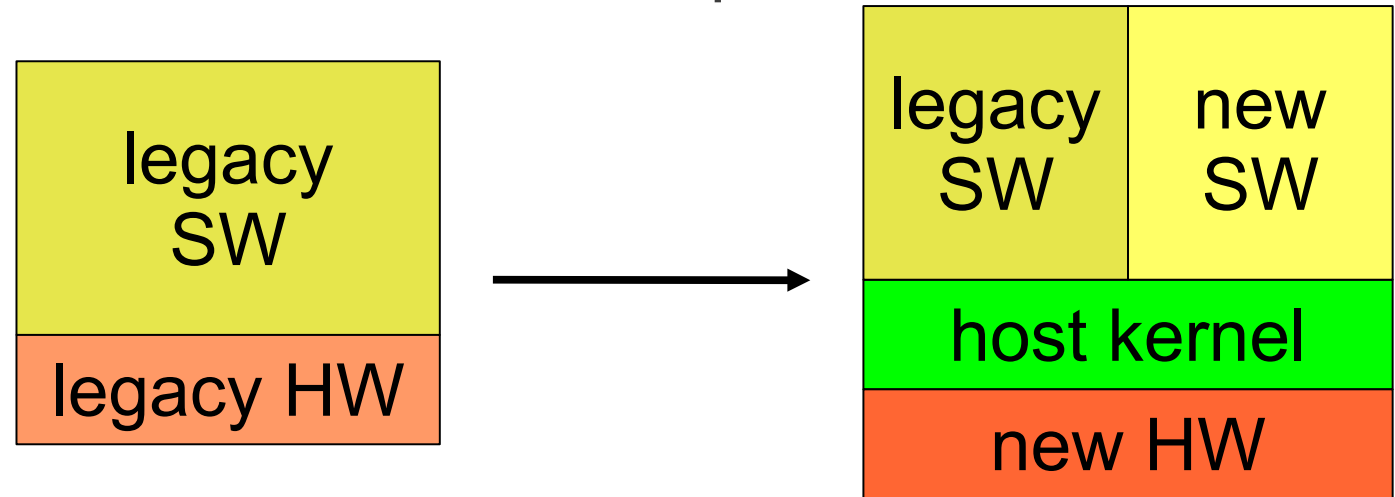
Use Case: Workload Consolidation

- Consolidate legacy systems

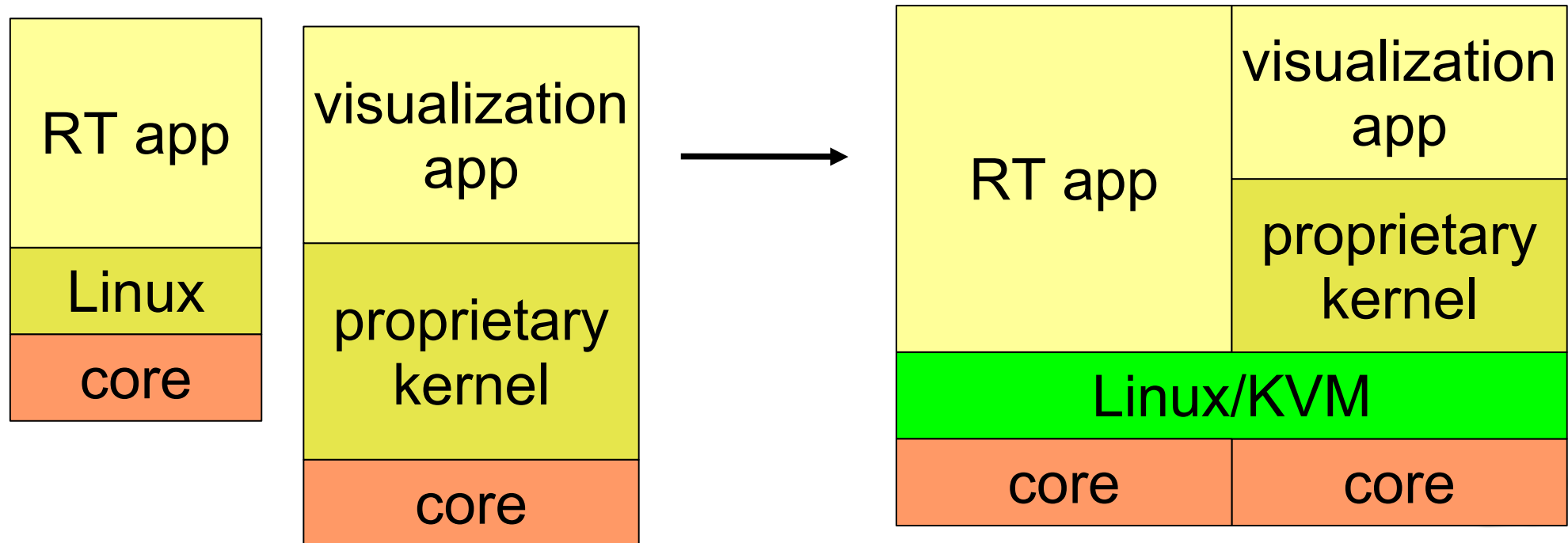


Use Case: Legacy Software

- Run legacy software on new core/chip/board with full virtualization

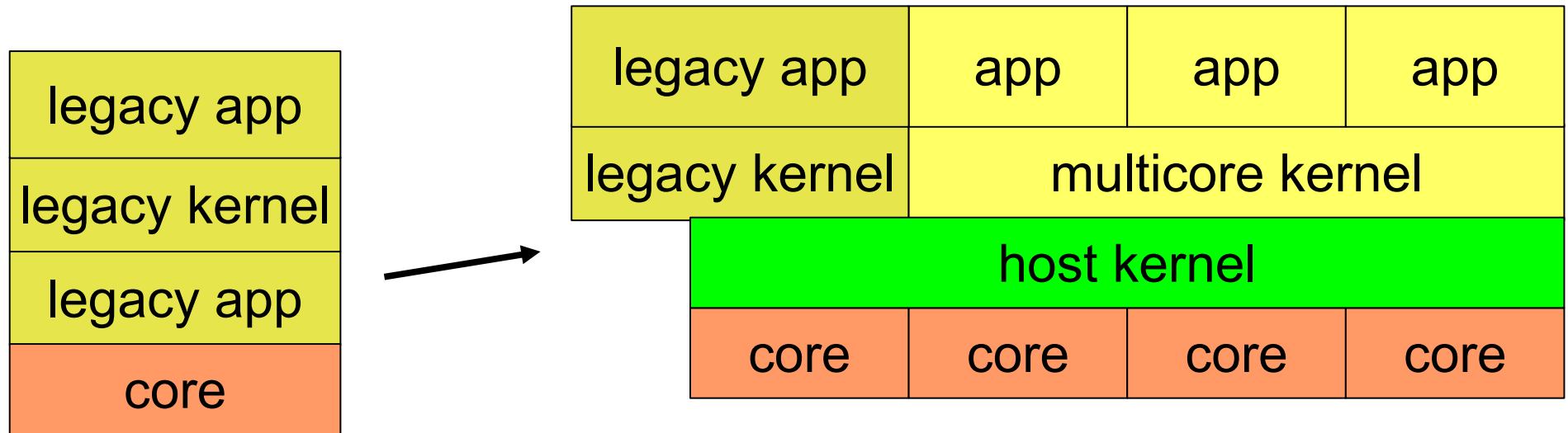


- Consolidate legacy software

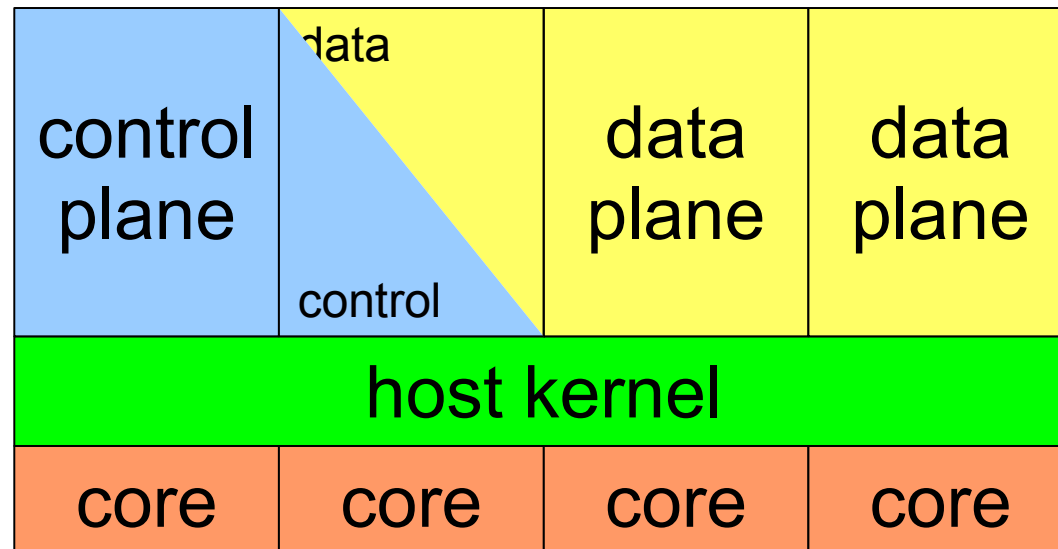


Use Case: Multicore Enablement

- Legacy uniprocessor applications

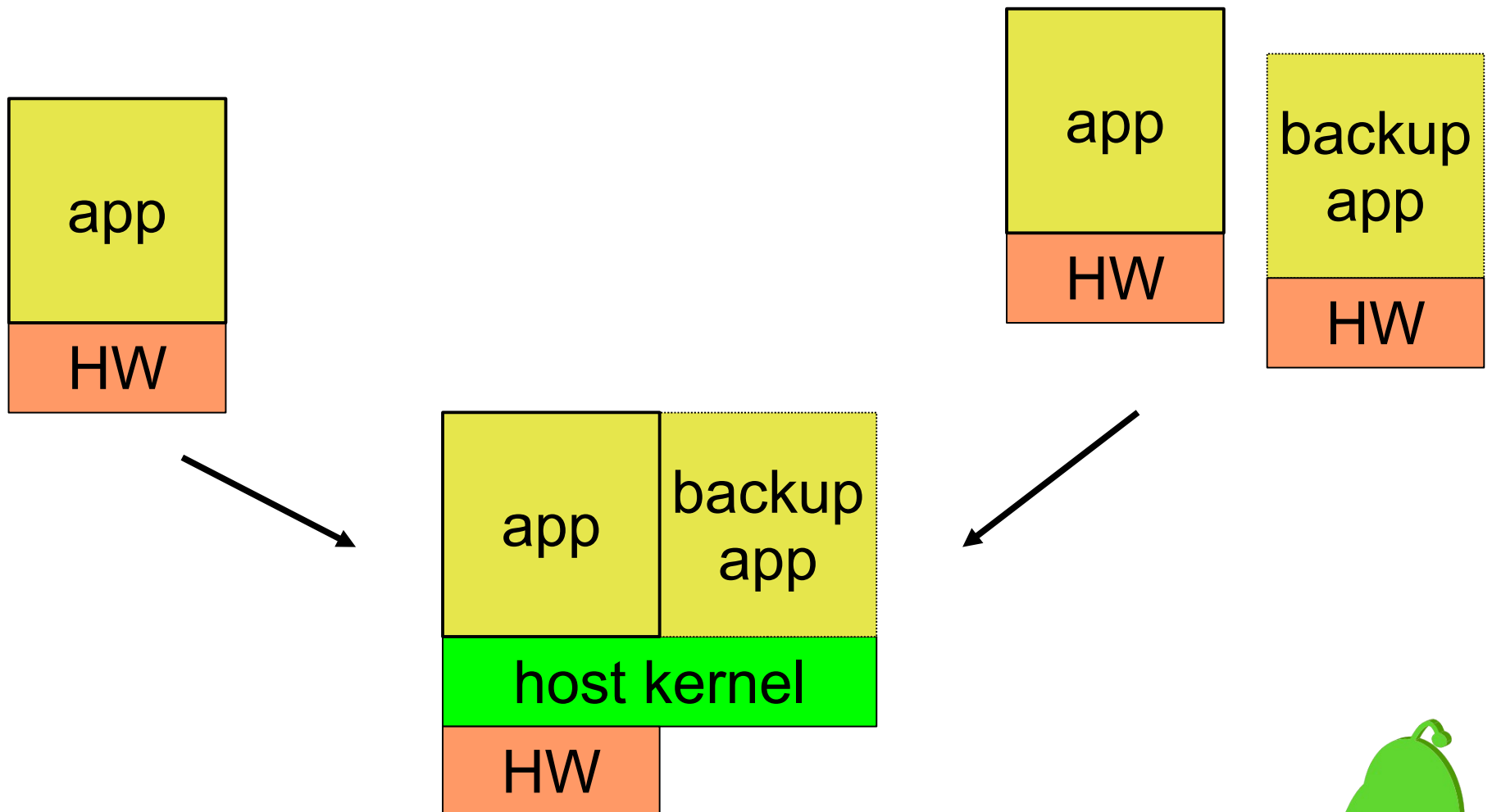


- Flexible resource management



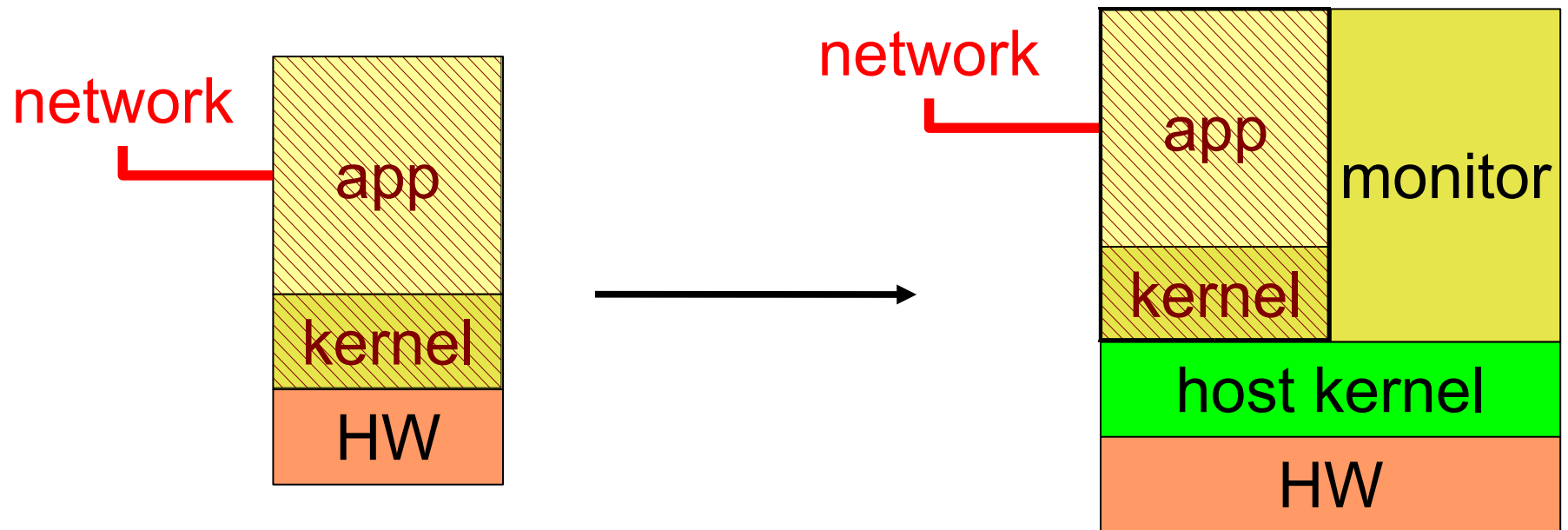
Use Case: Improved Reliability

- Hot standby without additional hardware



Use Case: Secure Monitoring

- Protect monitoring software



Hypervisor Design



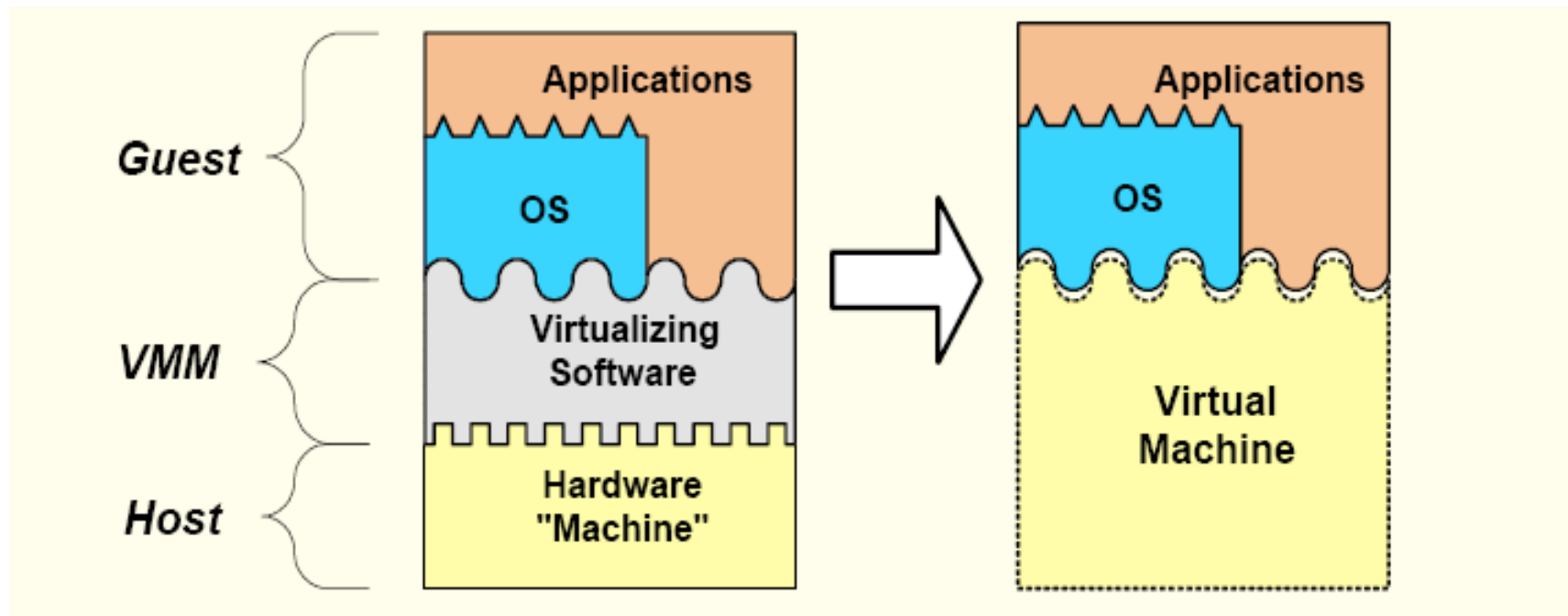
"All problems in computer science
can be solved by another level of
indirection."

-- David Wheeler --



Virtual Machine

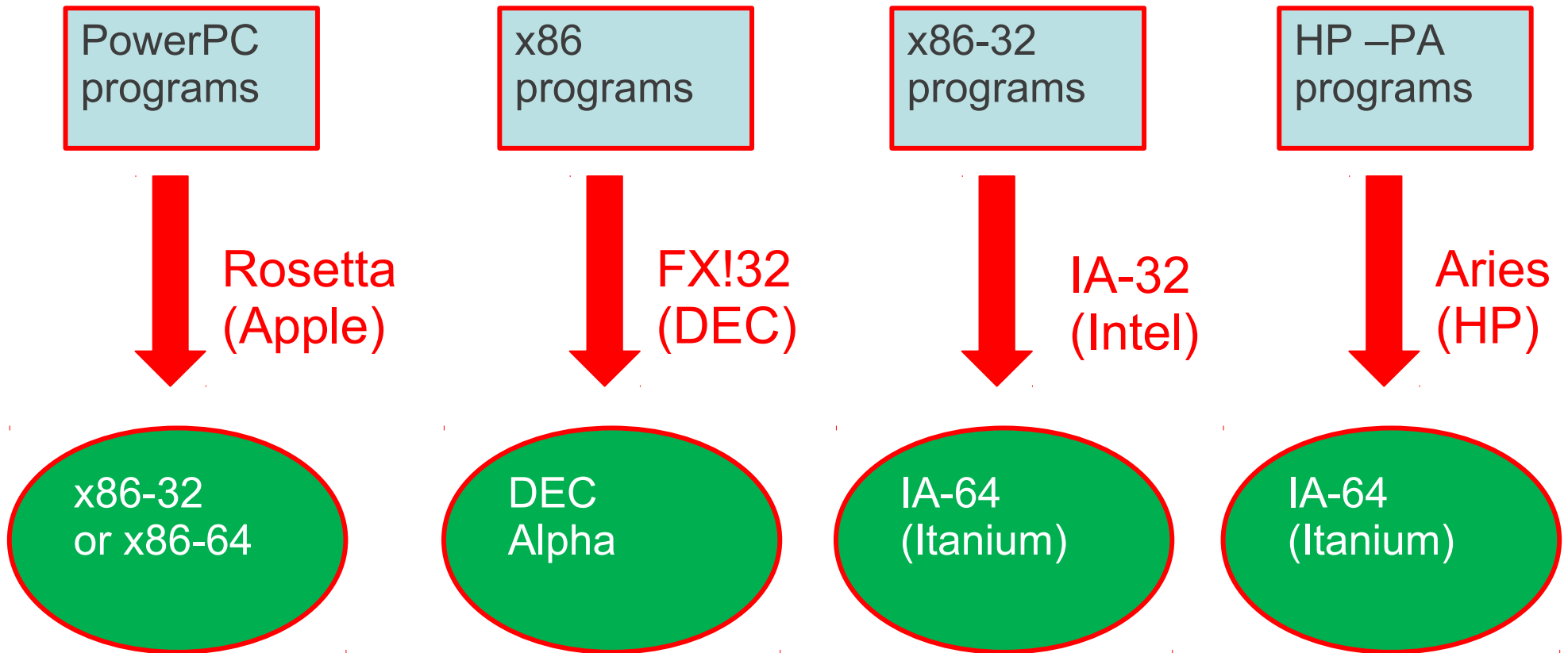
- Gerald Popek and Robert Goldberg defined it as **“efficient, isolated duplicate of a real machine”**
 - Add Virtualizing Software to a Host platform and support Guest process or system on a Virtual Machine (VM)



The software that provides this illusion is the Virtual Machine Monitor (VMM, mostly used synonymous with Hypervisor)

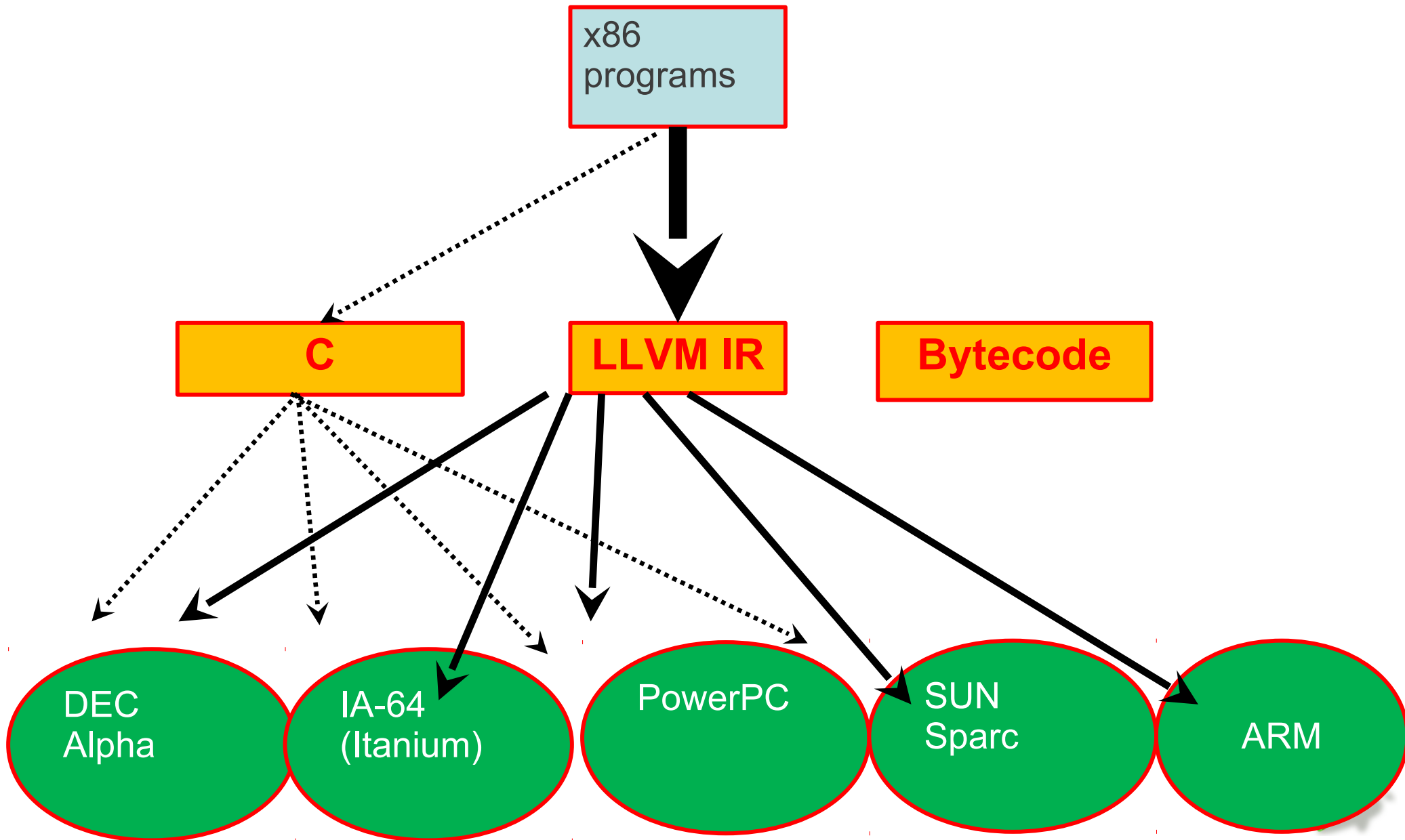


Virtual Machine for Portability (in the past)



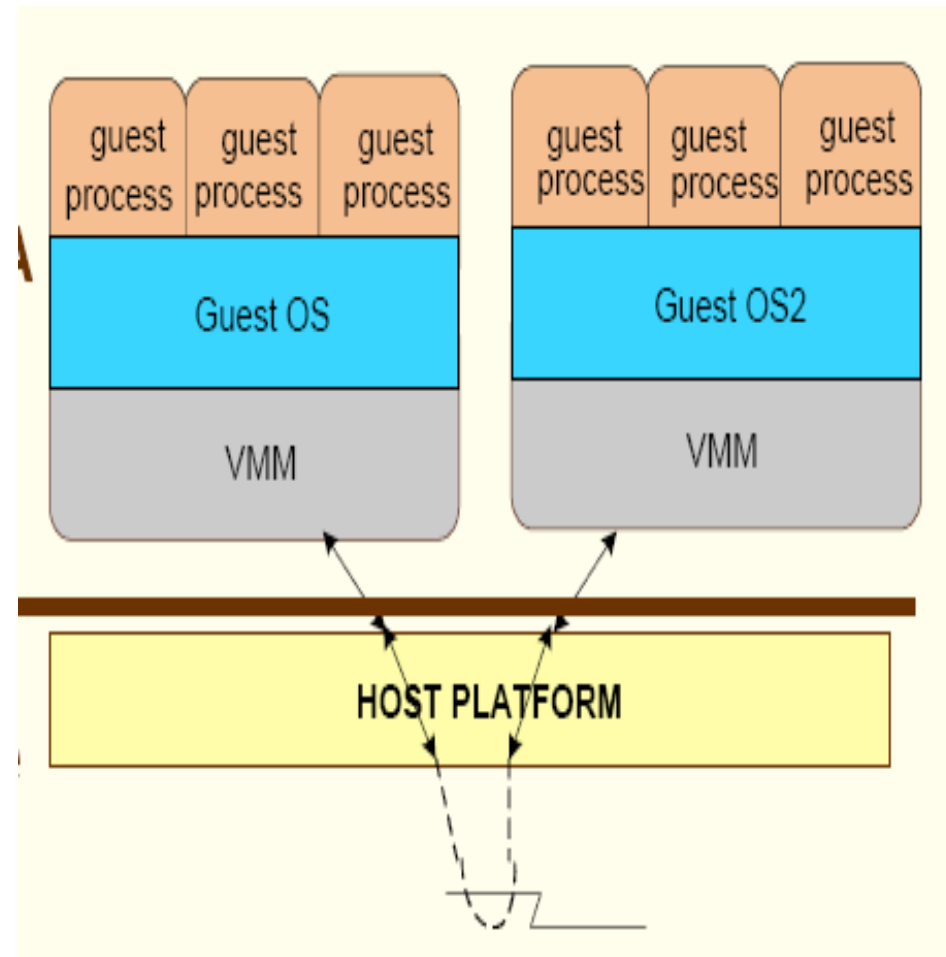
Virtual Machine for Portability

NOTE: we won't discuss such topic in this presentation



System Virtual Machine

- Provide a system environment
- Constructed at ISA level
- Allow multiple OS environments, or support time sharing.
- virtualizing software that implements system VM is called as VMM (virtual machine monitor)
- Examples:
 - IBM VM/360, VMware, VLX, WindRiver Hypervisor, ENEA Hypervisor
 - Xtratum, Lguest, BhyVe (BSD Hypervisor)
 - **Xen, KVM, OKL4, Xvisor, Codezero**



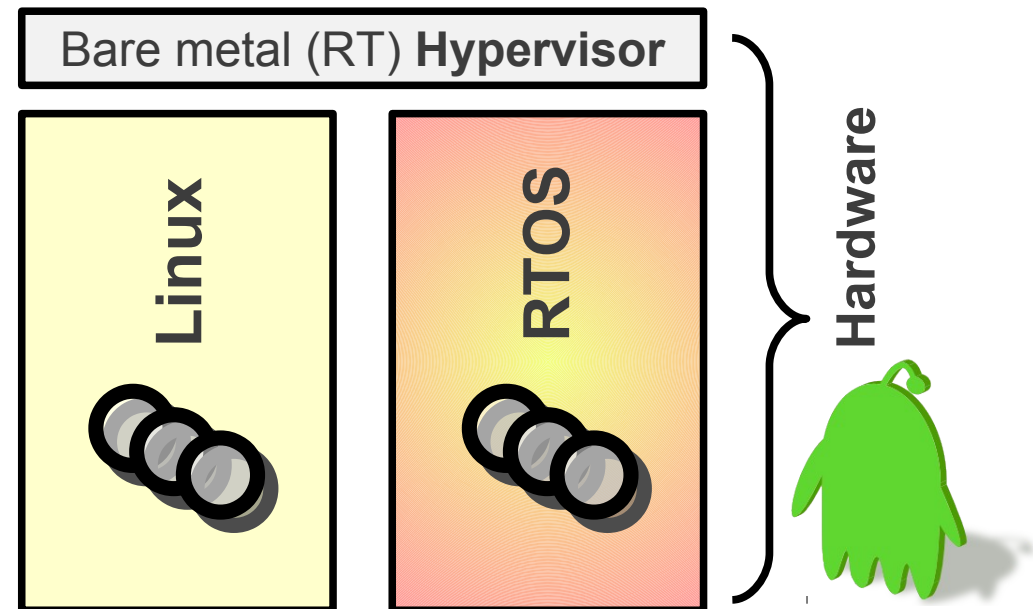
Virtual network communication



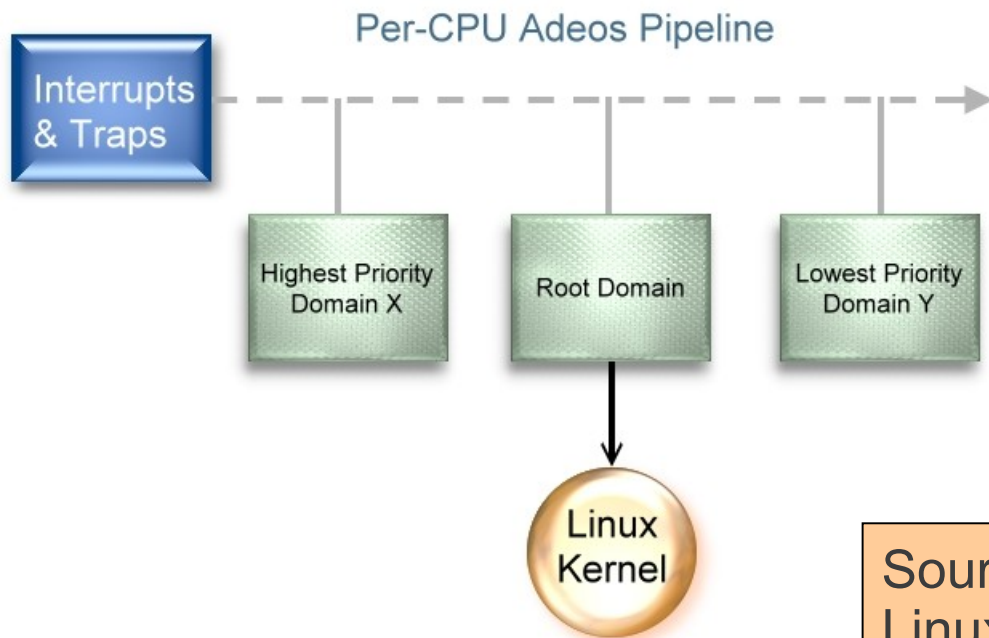
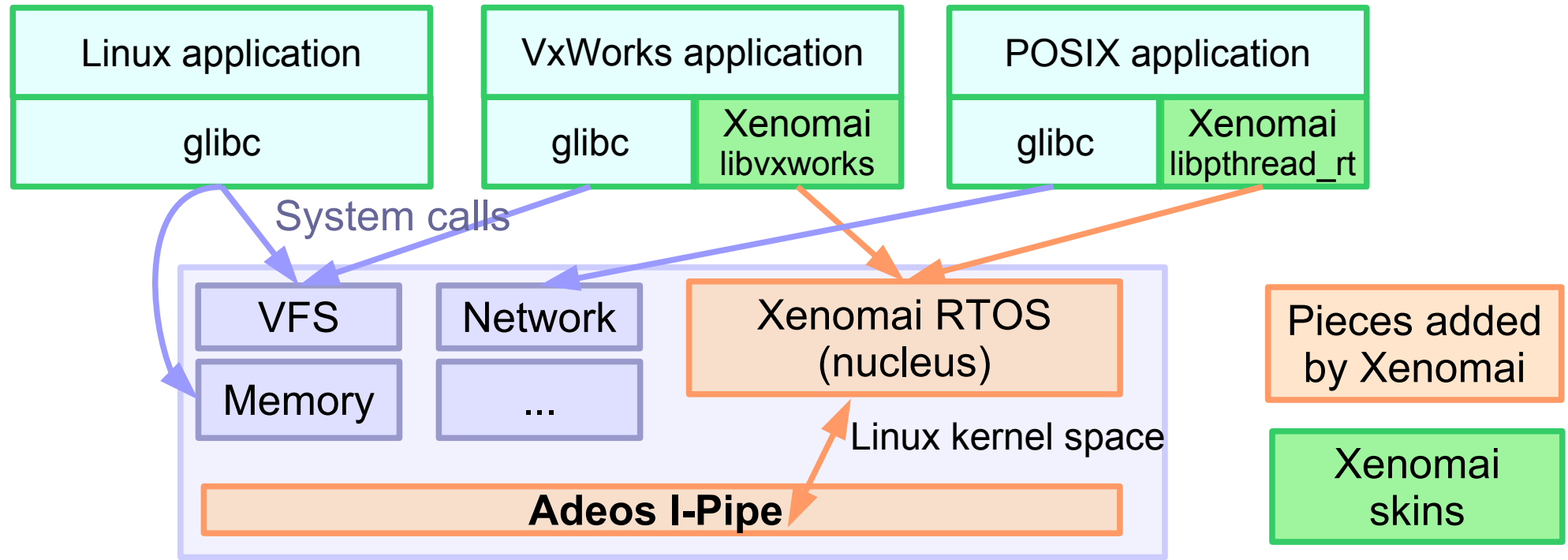
NOTE: We only focus on system virtual machine here.
Therefore, this presentation ignores Linux vserver, FreeBSD jail, etc.

Virtualization is Common Technique

- Example: In the past, Linux is far from being real-time, but RTLinux/RTAI/Xenomai/Xtratum attempted to "improve" Linux by introducing new virtualization layer.
- real-time capable virtualization
- Dual kernel approach



Example: Xenomai (Linux Realtime Extension)

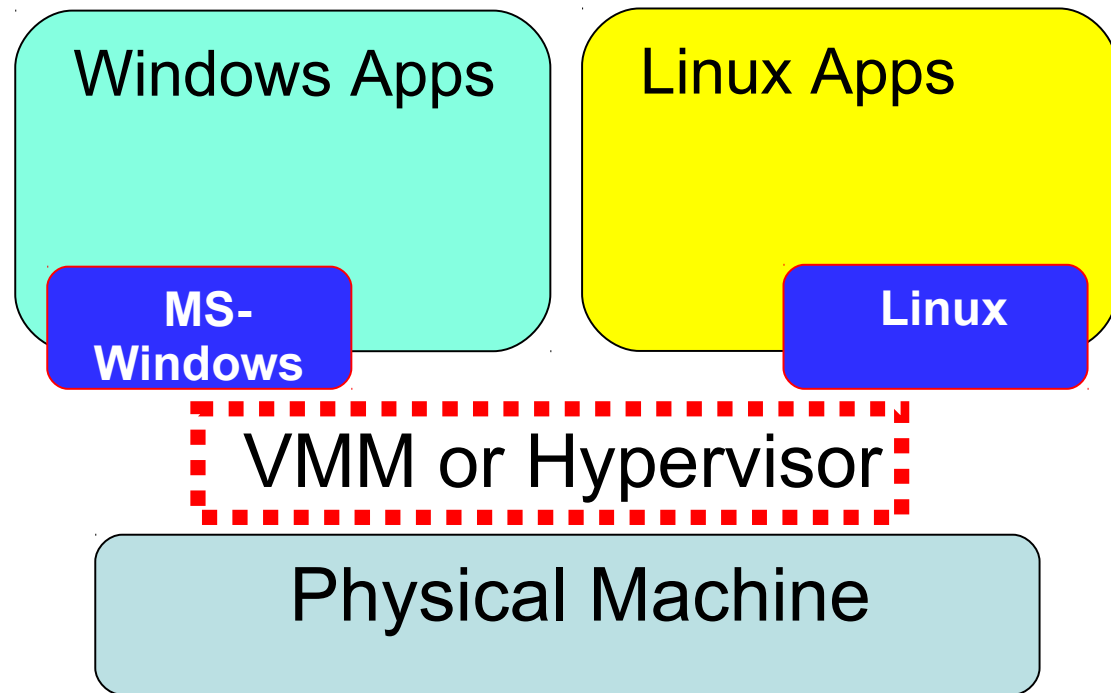


- From Adeos point of view, guest OSeS are prioritized domains.
- For each event (interrupts, exceptions, syscalls, etc...), the various domains may handle the event or pass it down the pipeline.

Source: Real-time in embedded Linux systems, Free Electrons (2011)

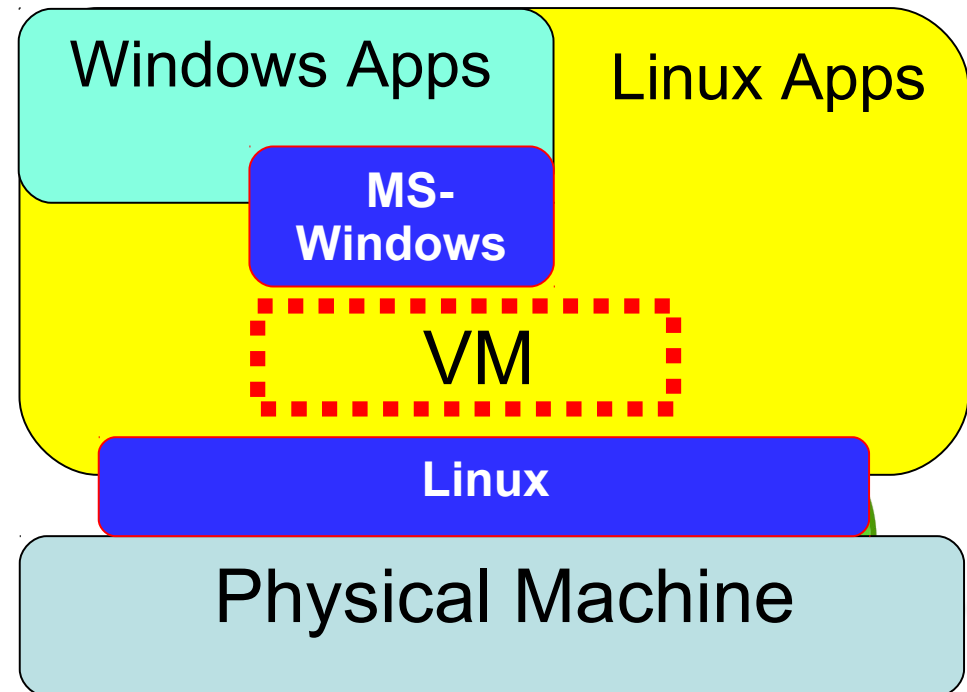


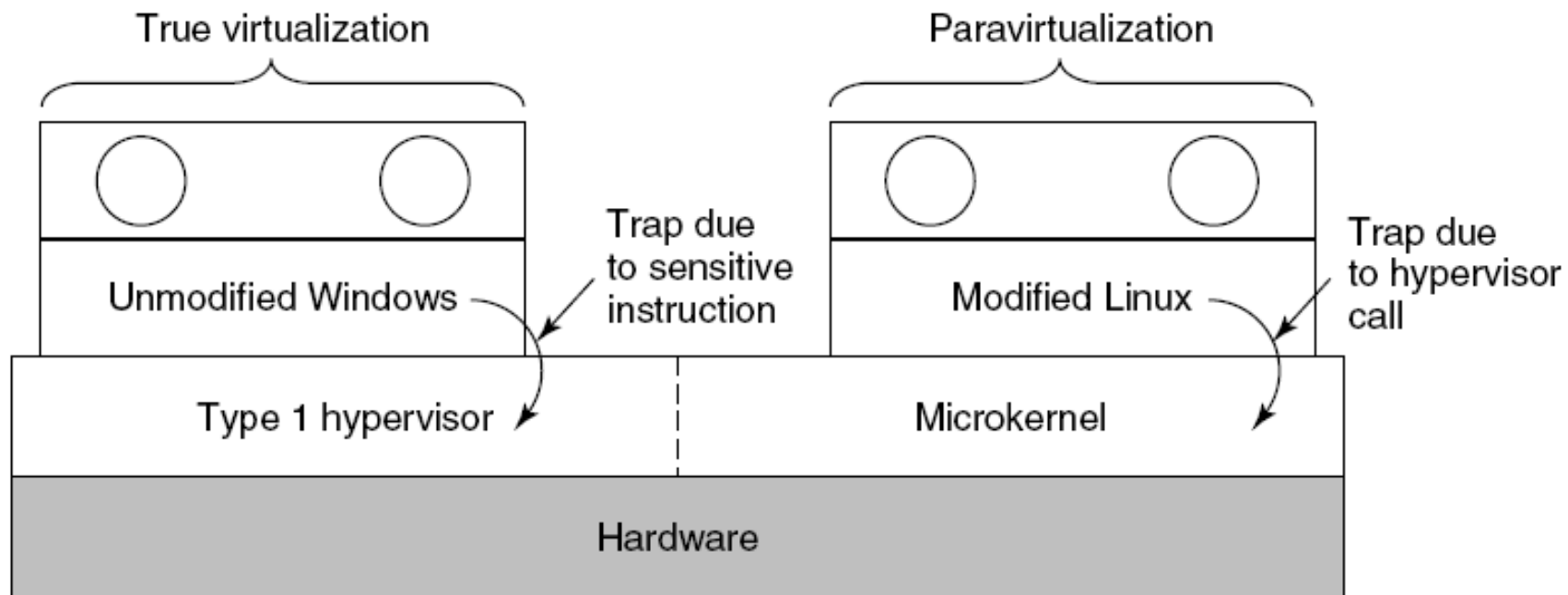
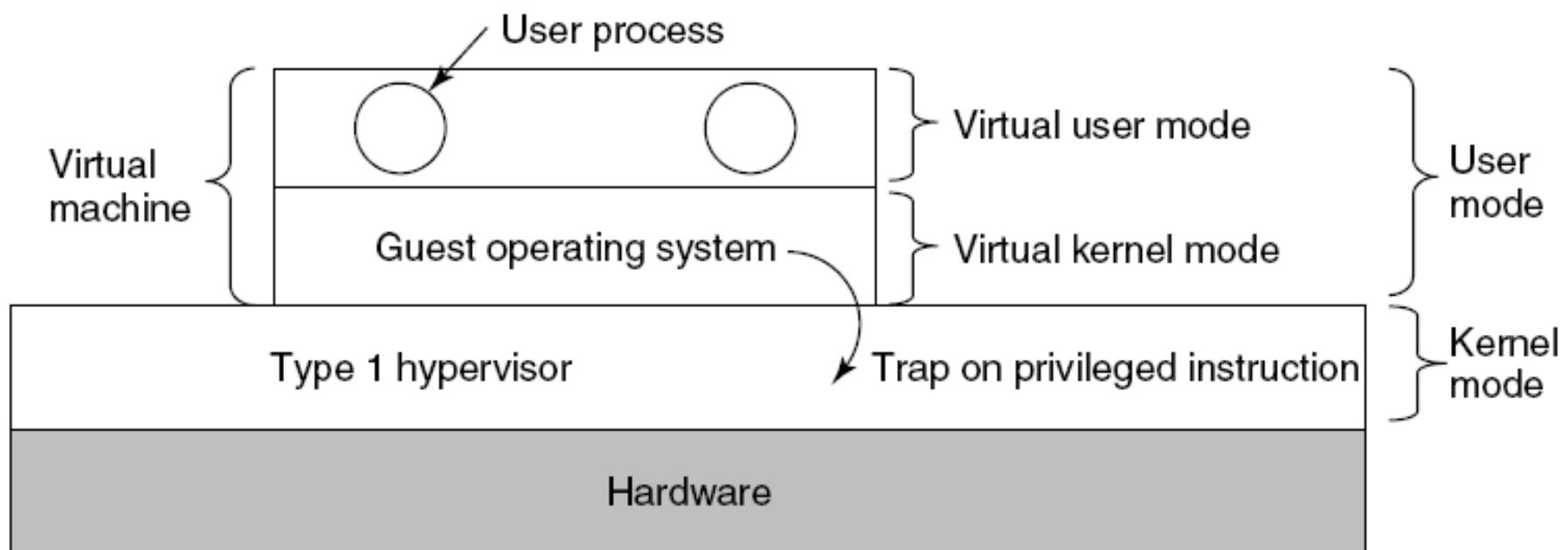
- Type I
- Bare metal system VM



General Classification of Virtualization technologies

- Type 2
- Hosted System VM





Myth of Type I and Type II Hypervisor

source: <http://gala4th.blogspot.com/2011/10/myth-of-type-i-and-type-ii-hypervisors.html>

- Myth: *Type-2 is "lesser" than a true "type-1" hypervisor.*
- Virtualization theory really started with a paper from Gerald Popek and Robert Goldberg called *Formal Requirements for Virtualizable Third Generation Architectures*. (1974)
 - **Sensitive Instructions**
 - **Privileged Instructions**
- The terms "type-1" and "type-2" originate from a paper by John Robin called *Analyzing the Intel Pentium's Capability to Support a Secure Virtual Machine Monitor*. (USENIX 2000)
 - Popek/Goldberg proof does not eliminate the possibility of using dynamic translation



Virtualizable

is a property of the Instruction Set Architecture (ISA)

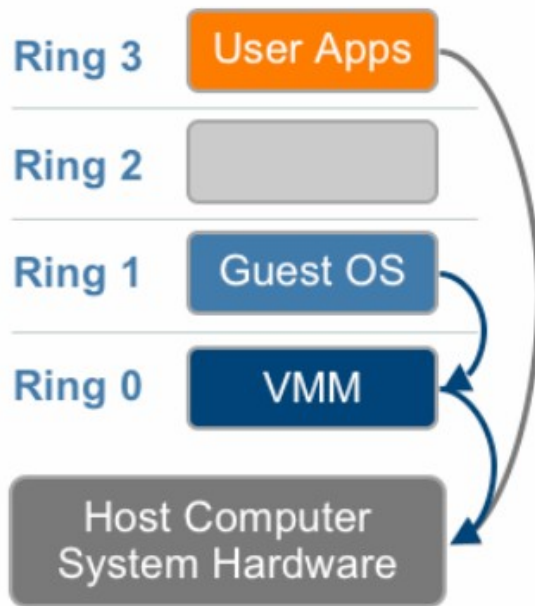
- **A sensitive instruction**
 - changes the configuration or mode of the processor,or
 - depends in its behavior on the processor's state
- **A privileged instruction**
 - must be executed with sufficient privilege
 - causes a trap in user mode

If all sensitive instructions are privileged, a VMM can be written.

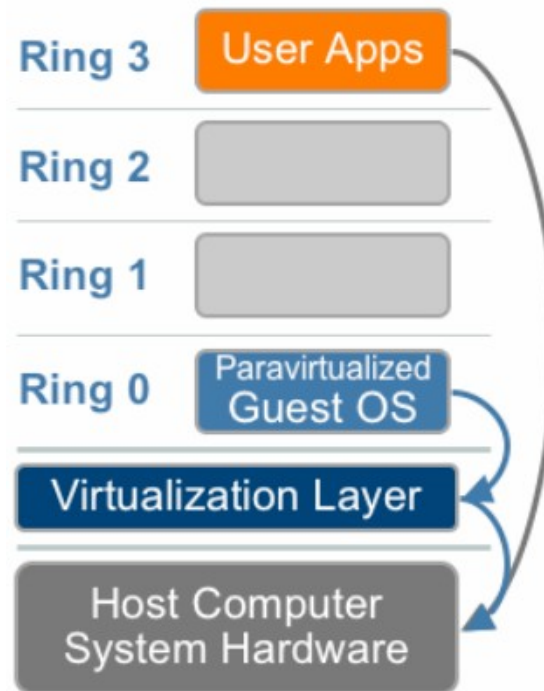


System Virtualization Implementations

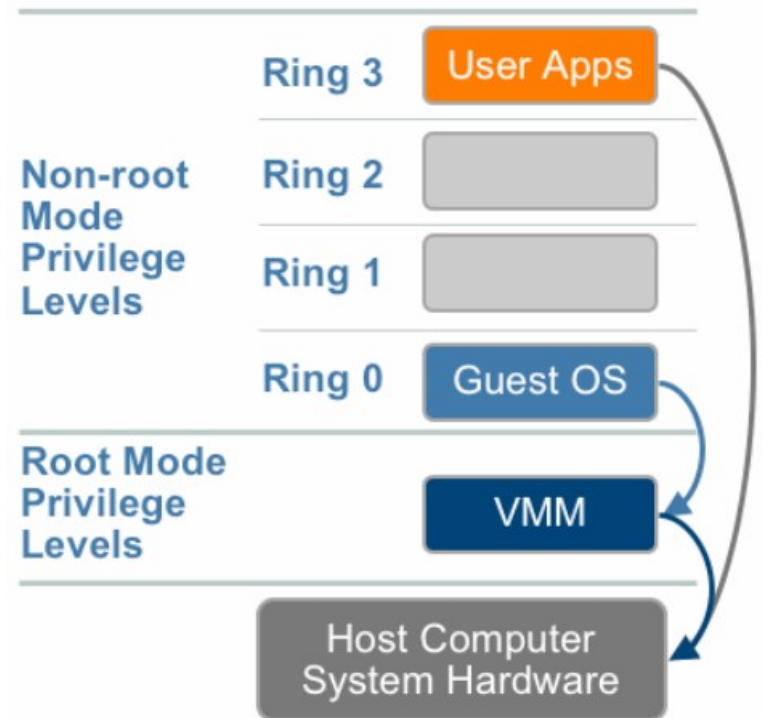
Full Virtualization



Para Virtualization

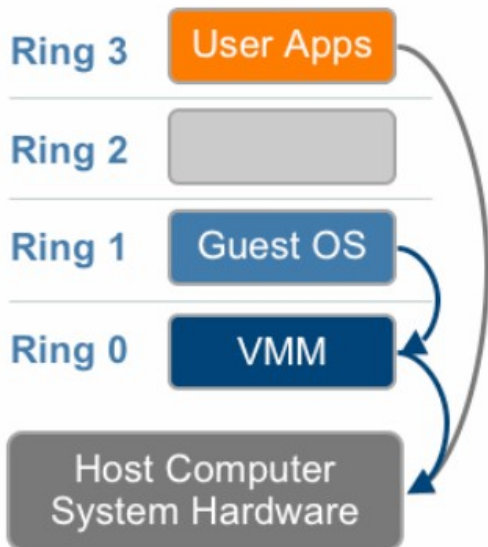
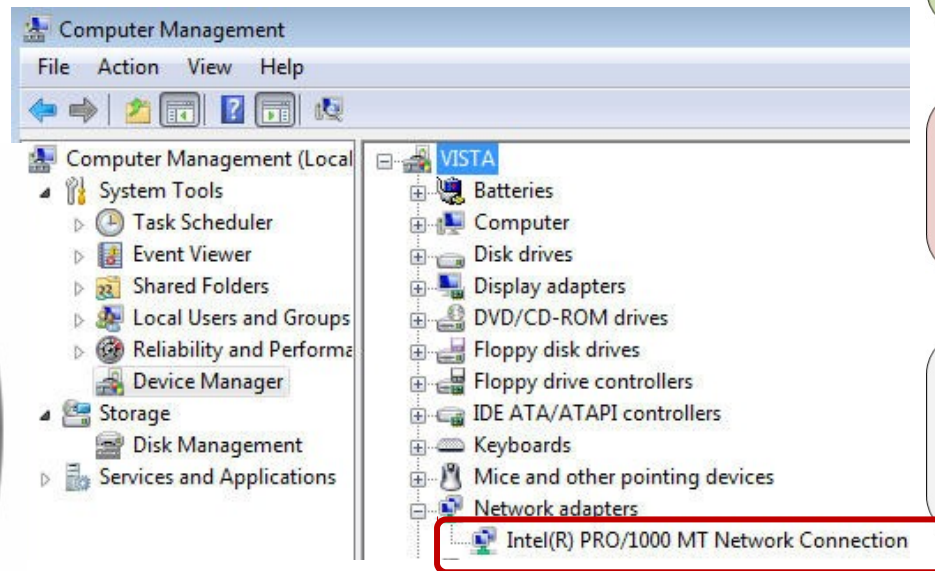
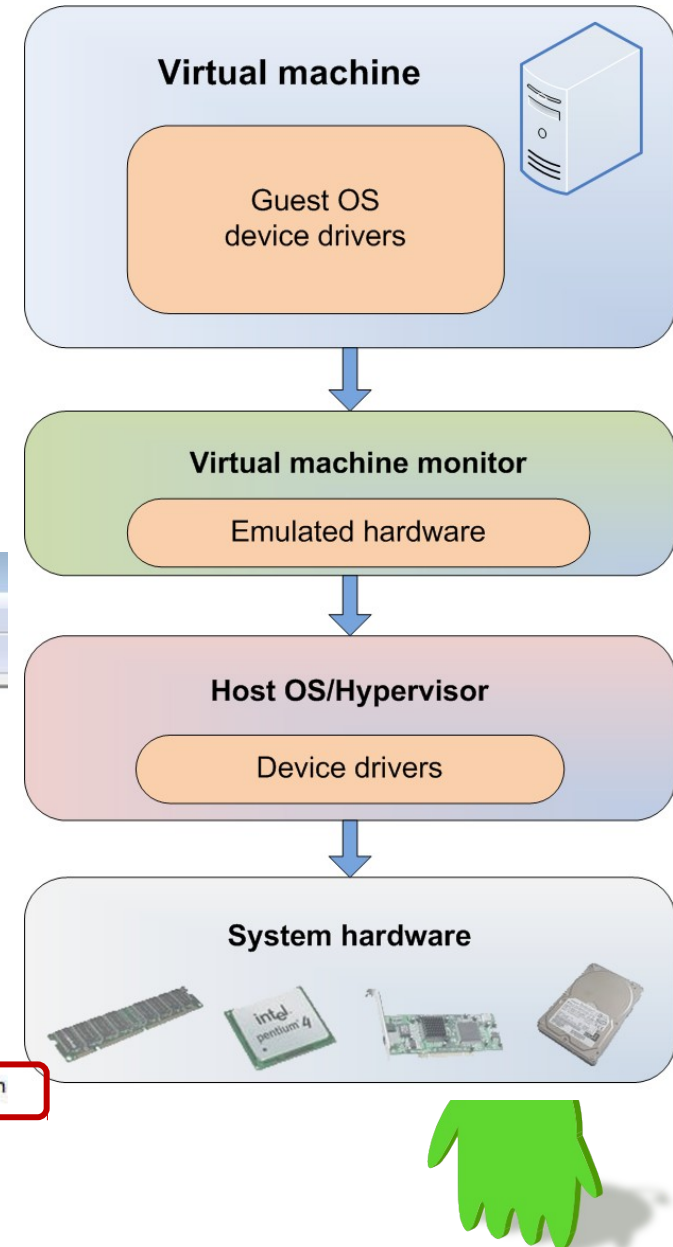


Hardware Assisted Virtualization



Full Virtualization

- Everything is virtualized
- Full hardware emulation
- Emulation = latency



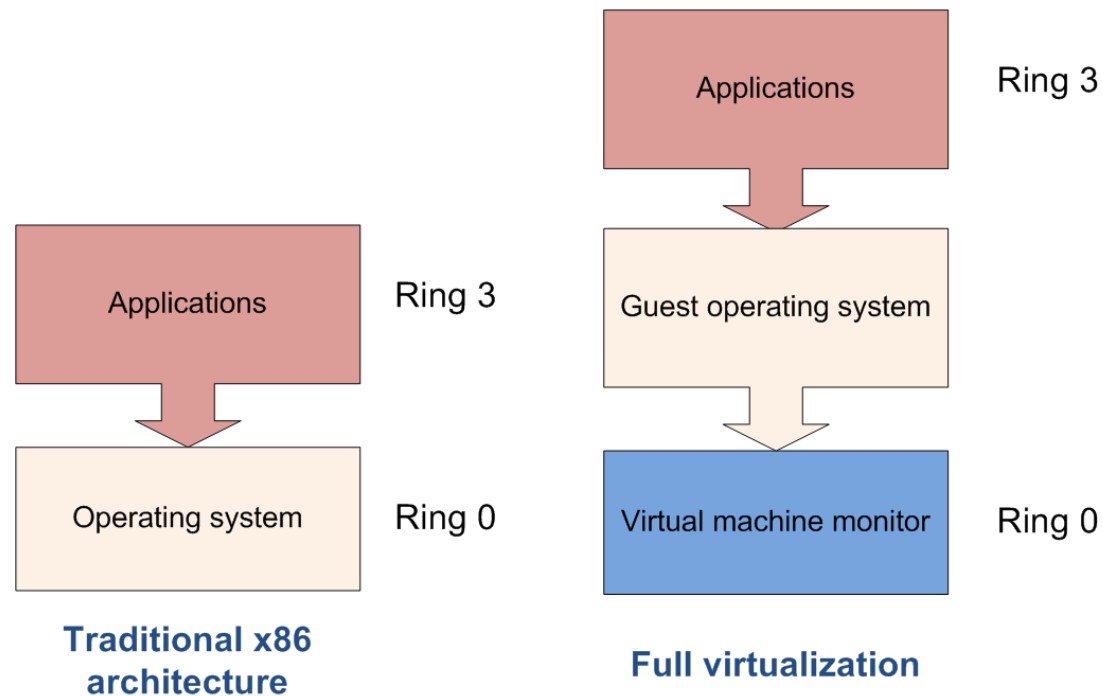
Recall the definitions

- **Sensitive Instructions** as defined by "*Formal Requirements for Virtualizable Third Generation Architectures*" (1974):
 - Mode Referencing Instructions
 - Sensitive Register/Memory Access Instructions
 - Storage Protection System Referencing Instructions
 - All I/O Instructions
- Theorem about strict **virtualizability** by "*Analyzing the Intel Pentium's Capability to Support a Secure Virtual Machine Monitor*" (2000):
 - For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.



Privileged Instructions

- Privileged instructions: OS kernel and device driver access to system hardware
- **Trapped and Emulated** by VMM
 - execute guest in separate address space in unprivileged mode
 - emulate all instructions that cause traps



Current Visible Registers

Abort Mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
spsr

0x1C

FIQ

0x18

IRQ

0x14

(Reserved)

0x10

Data Abort

0x0C

Prefetch Abort

0x08

Software Interrupt

0x04

Undefined Instruction

0x00

Reset

Vector Table

User

FIQ

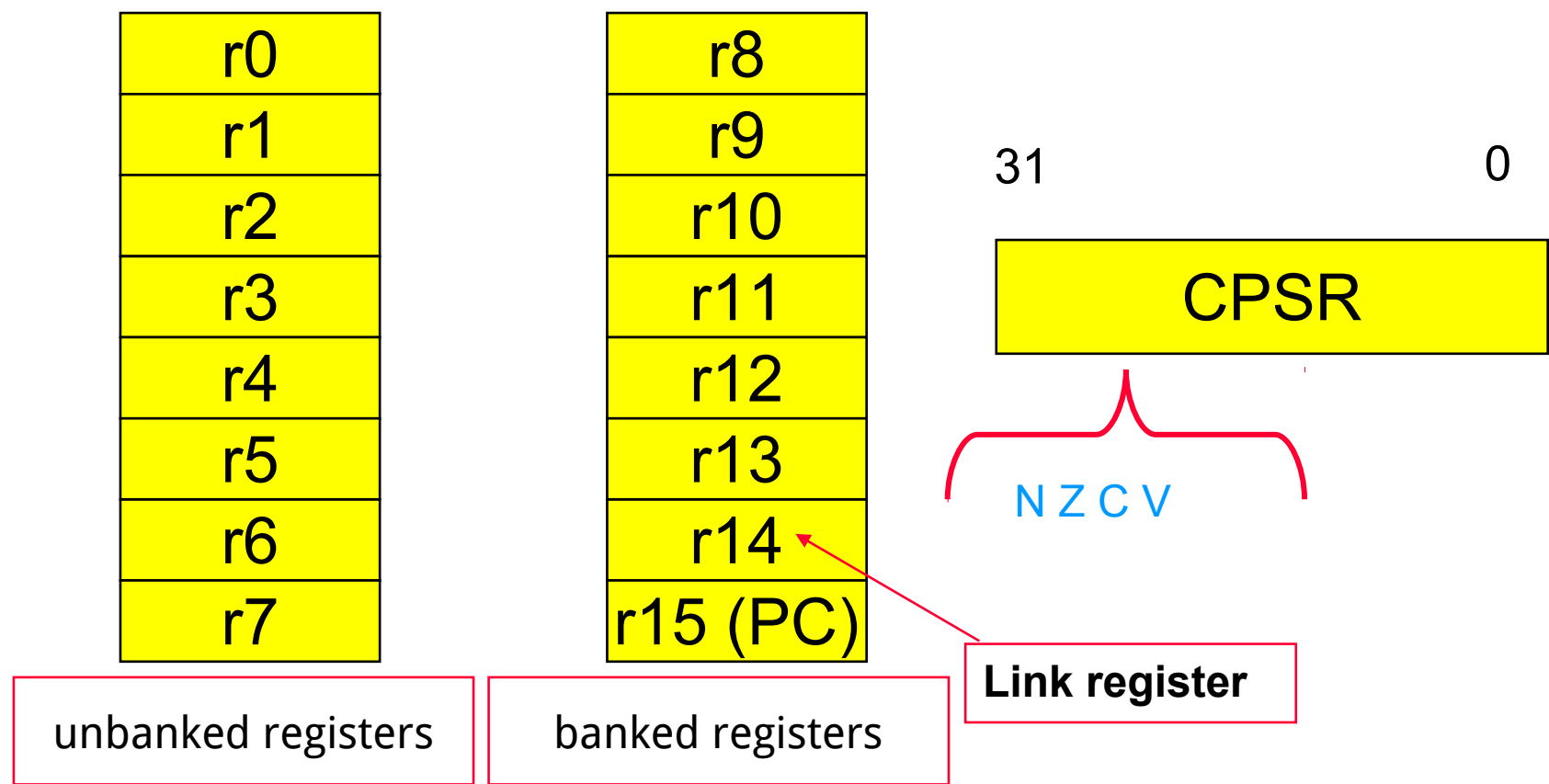
IRQ

SVC

Undef

Banked out Registers

	r8			
	r9			
	r10			
	r11			
	r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
	spsr	spsr	spsr	spsr

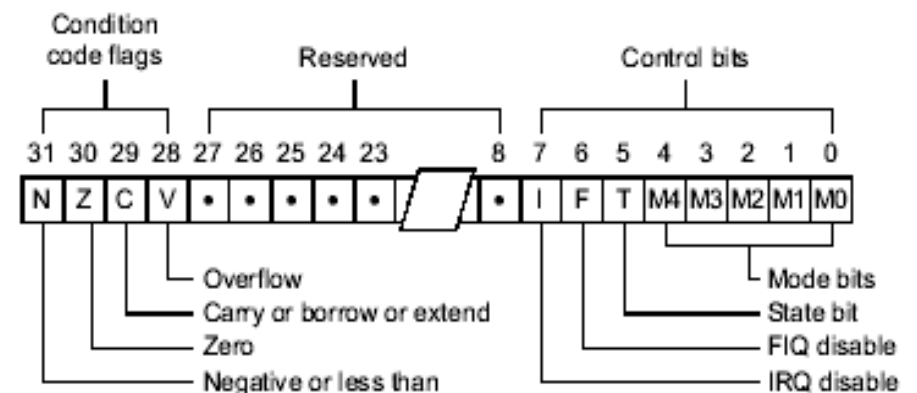


- Every arithmetic, logical, or shifting operation may set CPSR (*current program statues register*) bits:

- N (negative), Z (zero), C (carry), V (overflow).

- Examples:

- $-1 + 1 = 0$: NZCV = 0110.
- $2^{31}-1+1 = -2^{31}$: NZCV = 0101.



ARM Architecture (armv4)

- 6 basic operating modes (1 user, 5 privileged)
- 37 registers, all 32 bits wide
 - 1 program counter
 - 5 dedicated saved program status registers
 - 1 Current program status register (PSR)
 - 30 general purpose registers
- Special usage
 - r13 (stack pointer)
 - r14 (link register)
 - r15 (program counter, PC)



Typical ARM instructions (armv4)

- branch and branch with Link (**B**, **BL**)
- data processing instructions (**AND**, **TST**, **MOV**, ...)
- shifts: logical (**LSR**), arithmetic (**ASR**), rotate (**ROR**)
- test (**TEQ**, **TST**, **CMP**, **CMN**)
- processor status register transfer (**MSR**, **MRS**)
- memory load/store words (**LDR**, **STR**)
- push/pop Stack Operations (**STM**, **LDM**)
- software Interrupt (**SWI**; operating mode switch)
- co-processor (**CDP**, **LDC**, **STC**, **MRC**, **MCR**)



Problematic Instructions (1)

- Type 1

Instructions which executed in user mode will cause **undefined instruction** exception

- Example

MCR p15, 0, r0, c2, c0, 0

Move r0 to c2 and c0 in coprocessor specified by p15 (co-processor) for operation according to option 0 and 0

- MRC: from coproc to register

- MCR: from register to coproc

- Problem:

- Operand-dependent operation



Problematic Instructions (2)

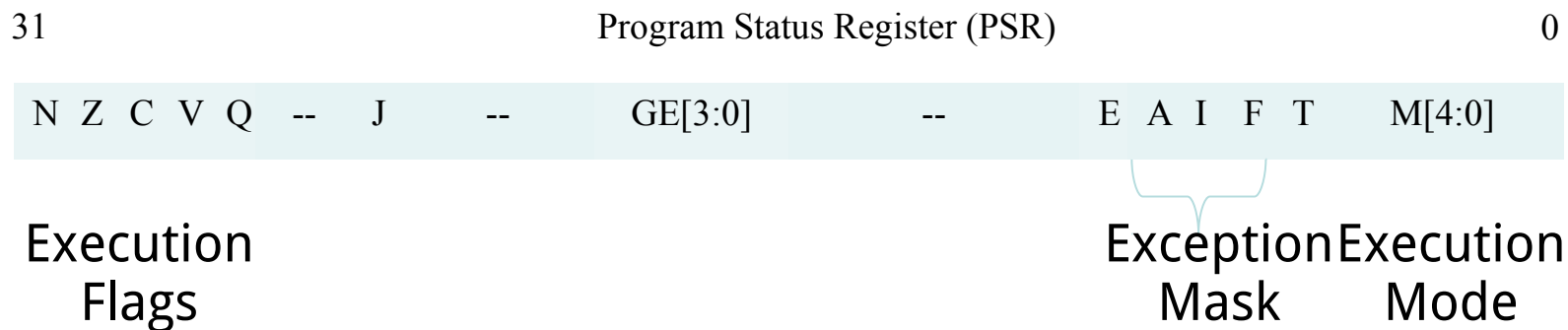
- Type 2

Instructions which executed in user mode will have **no effect**

- Example

MSR cpsr c, #0xD3

Switch to privileged mode and disable interrupt



Problematic Instructions (3)

- Type 3

Instructions which executed in user mode will cause **unpredictable behaviors**.

- Example

MOVS PC, LR

The return instruction changes the **program counter** and switches to **user mode**.

- This instruction causes unpredictable behavior when executed in user mode.



ARM Sensitive Instructions

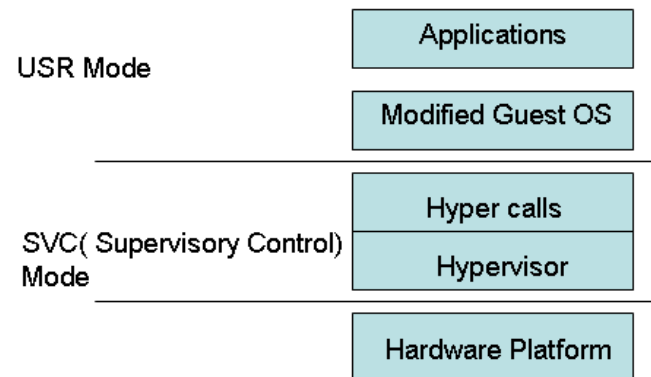
- Coprocessor Access Instructions
MRC / MCR / CDP / LDC / STC
- SIMD/VFP System Register Access Instructions
VMRS / VMSR
- TrustZone Secure State Entry Instructions
SMC
- Memory-Mapped I/O Access Instructions
Load/Store instructions from/into memory-mapped I/O locations
- Direct (Explicit/Implicit) CPSR Access Instructions
MRS / MSR / CPS / SRS / RFE / LDM (conditional execution) / **DPSPC**
- Indirect CPSR Access Instructions
LDRT / STRT – Load/Store Unprivileged (“As User”)
- Banked Register Access Instructions
LDM / STM (User mode registers)



Solutions to Problematic Instructions

[Hardware Techniques]

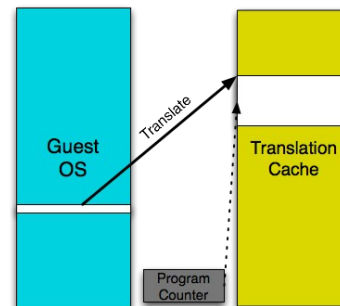
- Privileged Instruction Semantics dictated/translated by instruction set architecture
- MMU-enforced traps
 - Example: page fault
- Tracing/debug support
 - Example: **bkpt** (breakpoint)
- Hardware-assisted Virtualization
 - Example: extra privileged mode, HYP, in ARM Cortex-A15



Solutions to Problematic Instructions

[Software Techniques]

Complexity	Binary translation	Hypercall
Design	High	Low
Implementation	Medium	High
Runtime	High	Medium
Mapped to programming languages	Virtual function	Normal function

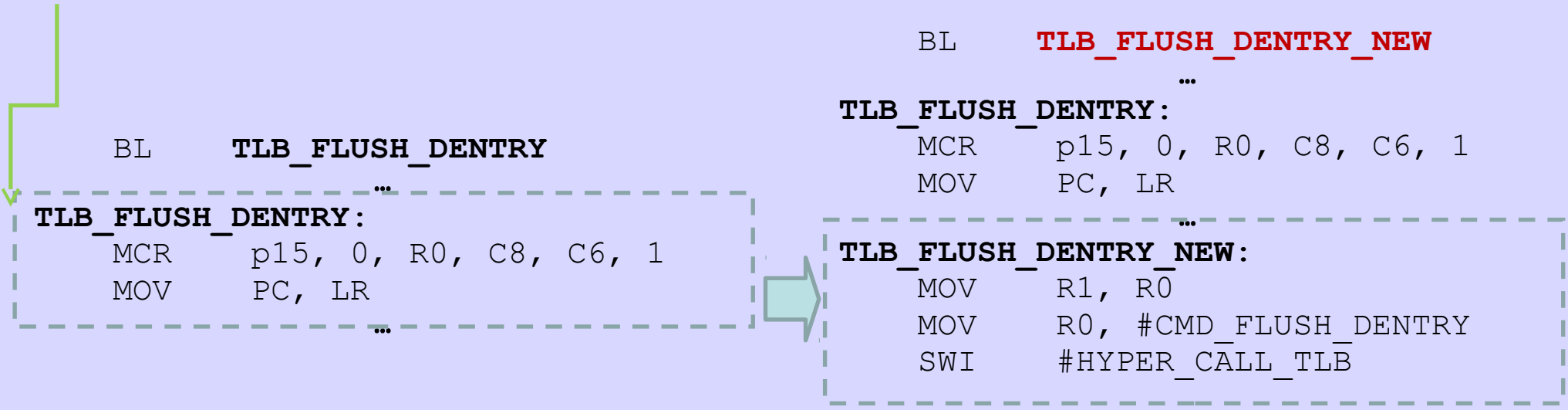


Method: trap and emulate



Dynamic Binary Translation

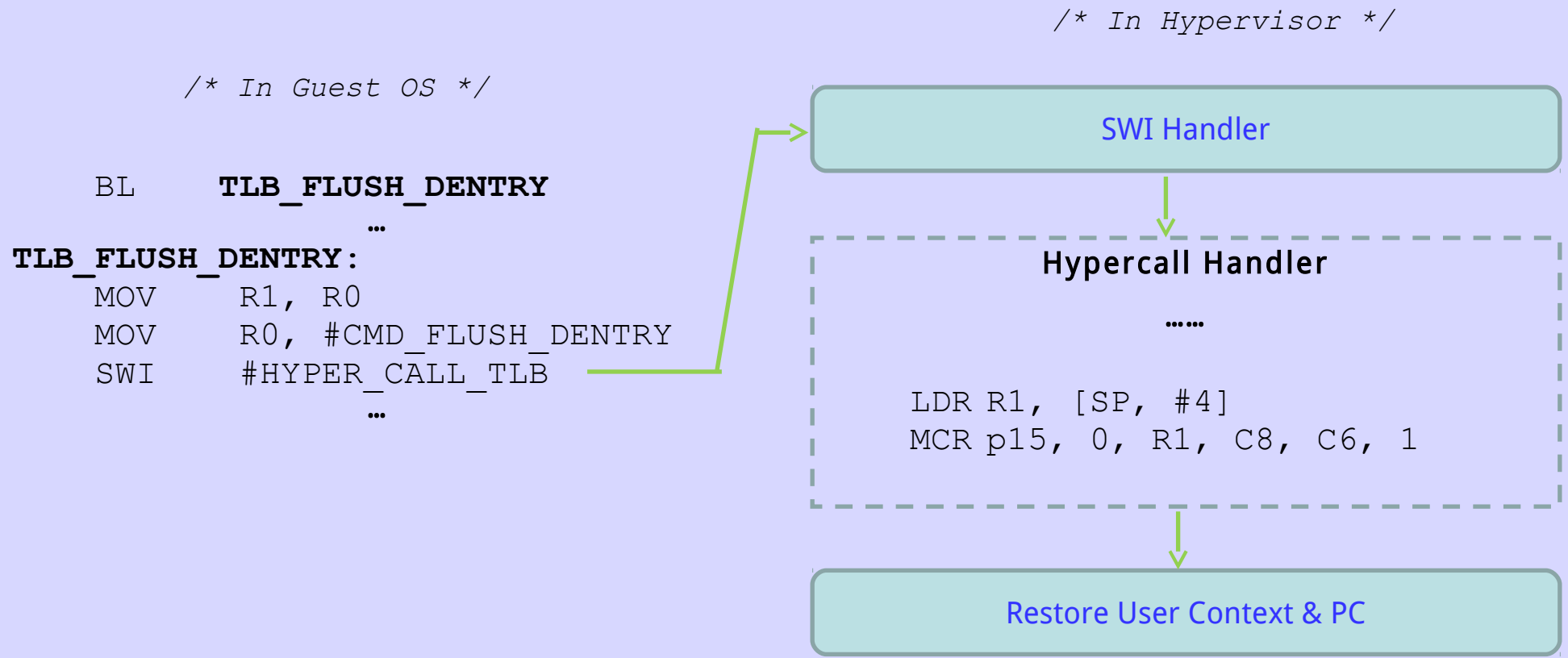
Translation Basic Block



- ARM has a fixed instruction size
 - 32-bit in ARM mode and 16-bit in Thumb mode
- Perform binary translation
 - Follow control-flow
 - Translate basic block (if not already translated) at the current PC
 - Ensure interposition at end of translated sequence
 - All writes (but not reads) to PC now become problematic instructions
 - Replace problematic instructions 1-1 with hypercalls to trap and emulate → self-modifying code



Virtualization APIs – hypercalls



- Use trap instruction to issue hypercall
- Encode hypercall type and original instruction bits in hypercall hint
- Upon trapping into the VMM, decode the hypercall type and the original instruction bits, and emulate instruction semantics

```
mrs Rd, R <cpsr/spsr>
```

cond	0001	OR00	SBO.	-Rd-	SBZ.	0000	SBZ.
------	------	------	------	------	------	------	------

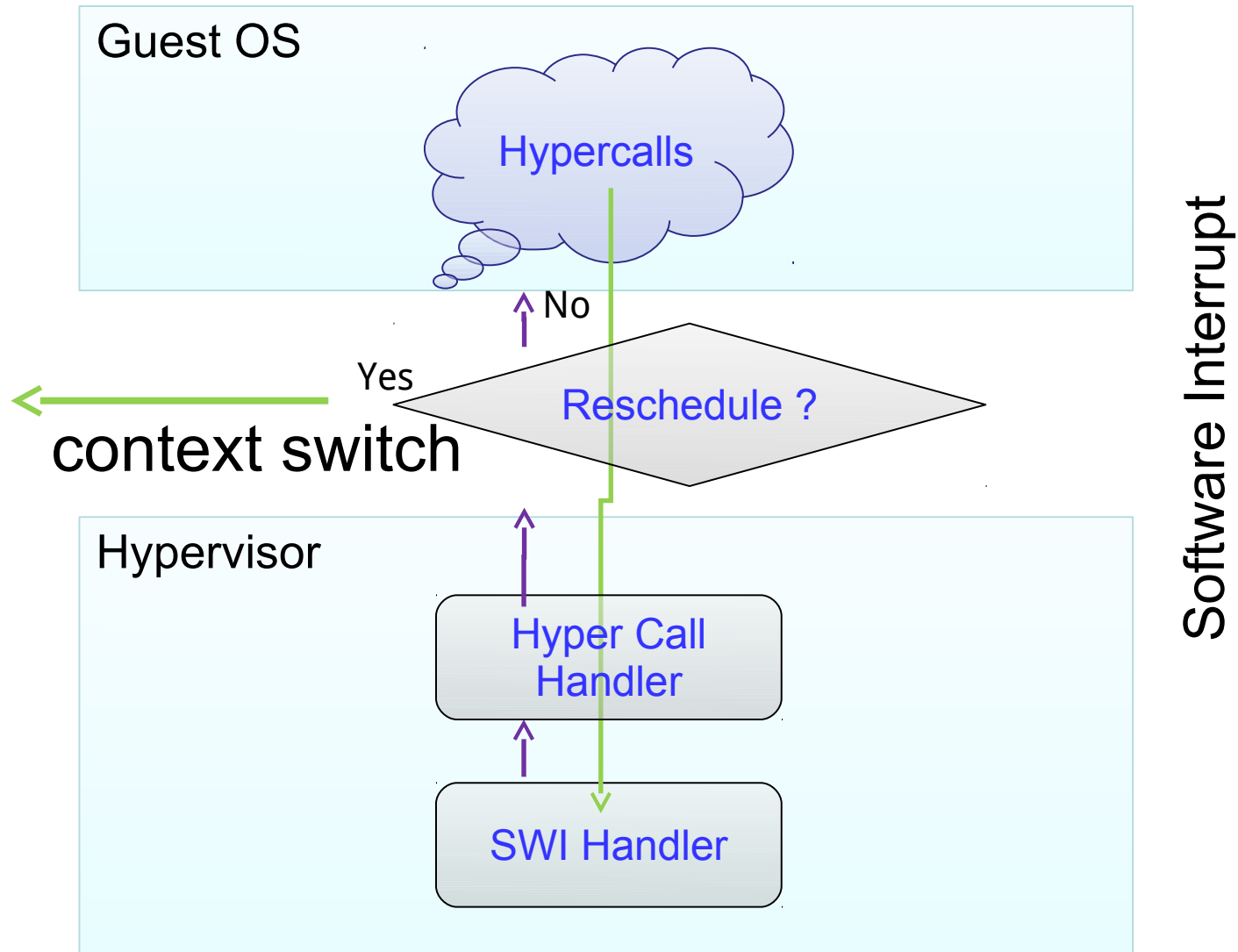
cond	1111	000010	OR	-Rd-	0000	0000	0000
------	------	--------	----	------	------	------	------

```
mrs r8, cpsr
```

```
swi 0x088000
```



Hypercall



Hypercall in xen-i386

System Call

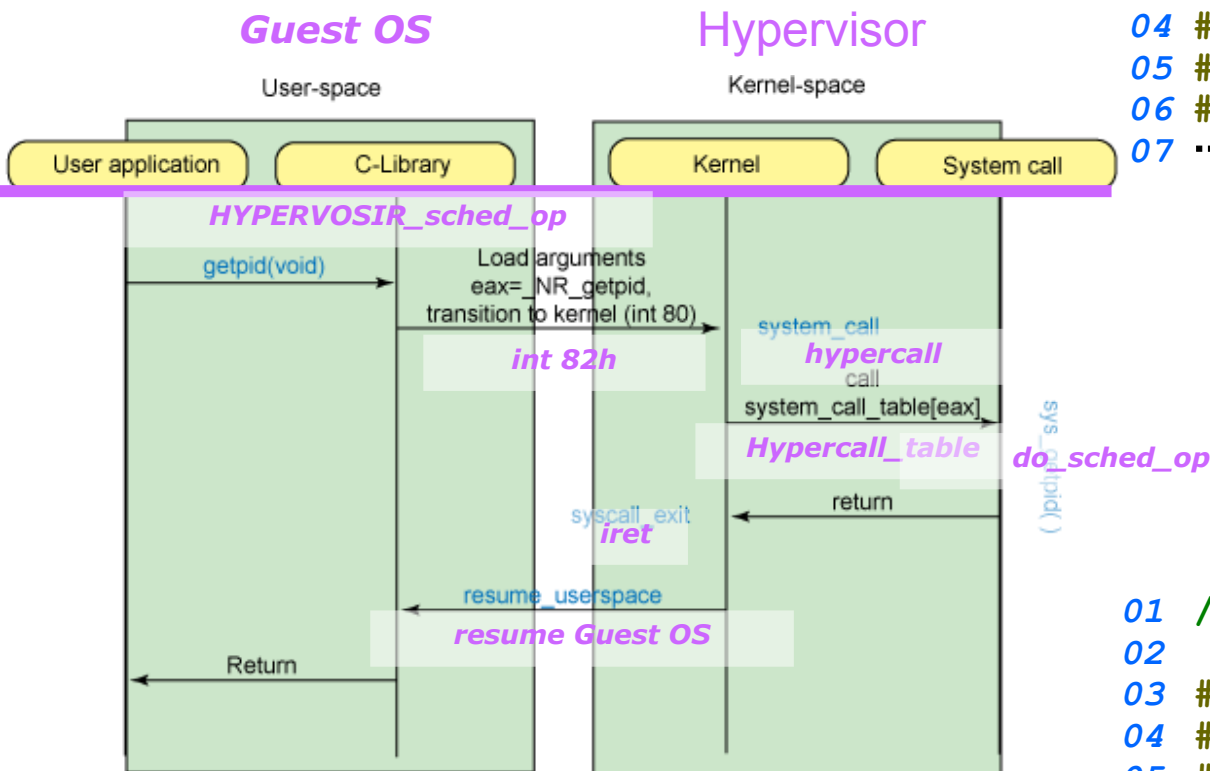
int 0x80

```
01 // linux/include/asm/unistd.h
02
03 #define __NR_restart_syscall    0
04 #define __NR_exit               1
05 #define __NR_fork               2
06 #define __NR_read               3
07 ...
```

Hyper Call

int 0x82

```
01 // xen/include/public/xen.h
02
03 #define __HYPERVISOR_set_trap_table 0
04 #define __HYPERVISOR_mmu_update    1
05 #define __HYPERVISOR_set_gdt       2
06 #define __HYPERVISOR_stack_switch  3
07 ...
```



Case study: Xvisor-ARM

<https://github.com/xvisor>

- File: `arch/arm/cpu/arm32/elf2cpatch.py`
 - Script to generate `cpatch` script from guest OS ELF
- Functionality before generating the final ELF image
 - Each sensitive non-privileged ARM instruction is converted to a hypercall.
 - Hypercall in ARM instruction set is `SVC <imm24>` instruction.
 - Encode sensitive non-privileged instructions in `<imm24>` operand of `SVC` instruction. (software interrupt)
 - Each encoded instruction will have its own unique `inst_id`.
 - The `inst_field` for each encoded sensitive non-privileged instruction will be different.



How does Xvisor handle problematic instructions like MSR?

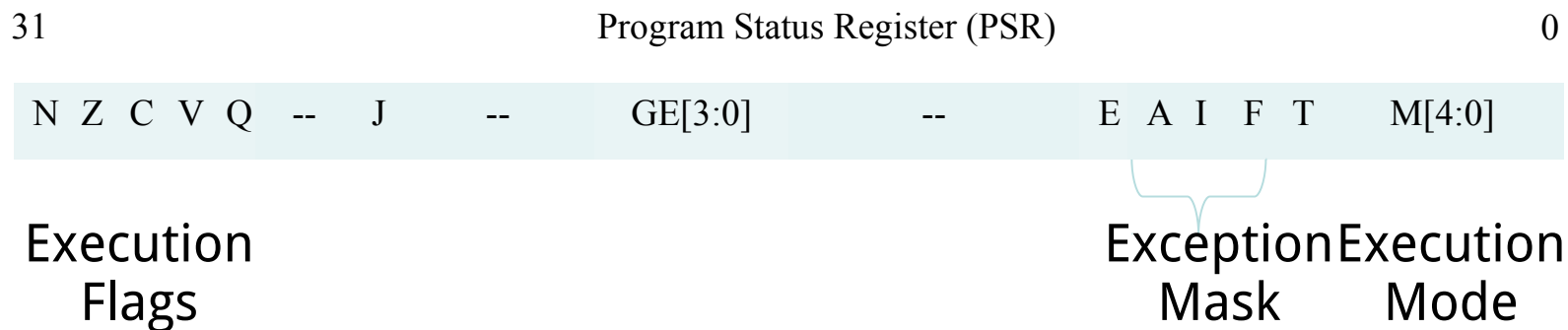
- Type 2

Instructions which executed in user mode will have **no effect**

- Example

MSR cpsr c, #0xD3

Switch to privileged mode and disable interrupt



First, cpatch (ELF patching tool) looks
up the instructions...

MSR cpsr_c, #0xD3

Switch to privileged mode and disable interrupt

```
# MSR (immediate)
#     Syntax:
#         msr<c> <spec_reg>, #<const>
#     Fields:
#         cond = bits[31:28]
#         R = bits[22:22]
#         mask = bits[19:16]
#         imm12 = bits[11:0]
#     Hypercall Fields:
#         inst_cond[31:28] = cond
#         inst_op[27:24] = 0xf
#         inst_id[23:20] = 0
#         inst_subid[19:17] = 2
#         inst_fields[16:13] = mask
#         inst_fields[12:1] = imm12
#         inst_fields[0:0] = R
```



```

def convert_msr_i_inst(hxstr):
    hx = int(hxstr, 16)
    inst_id = 0
    inst_subid = 2
    cond = (hx >> 28) & 0xF
    R = (hx >> 22) & 0x1
    mask = (hx >> 16) & 0xF
    imm12 = (hx >> 0) & 0xFFF
    rethx = 0x0F000000
    rethx = rethx | (cond << 28)
    rethx = rethx | (inst_id << 20)
    rethx = rethx | (inst_subid << 17)
    rethx = rethx | (mask << 13)
    rethx = rethx | (imm12 << 1)
    rethx = rethx | (R << 0)
    return rethx

```

```

# MSR (immediate)
# Syntax:
# msr<c> <spec_reg>, #<const>
# Fields:
# cond = bits[31:28]
# R = bits[22:22]
# mask = bits[19:16]
# imm12 = bits[11:0]
# Hypercall Fields:
# inst_cond[31:28] = cond
# inst_op[27:24] = 0xf
# inst_id[23:20] = 0
# inst_subid[19:17] = 2
# inst_fields[16:13] = mask
# inst_fields[12:1] = imm12
# inst_fields[0:0] = R

```

Xvisor utilizes cpatch to convert all problematic instructions for OS image files (ELF format).



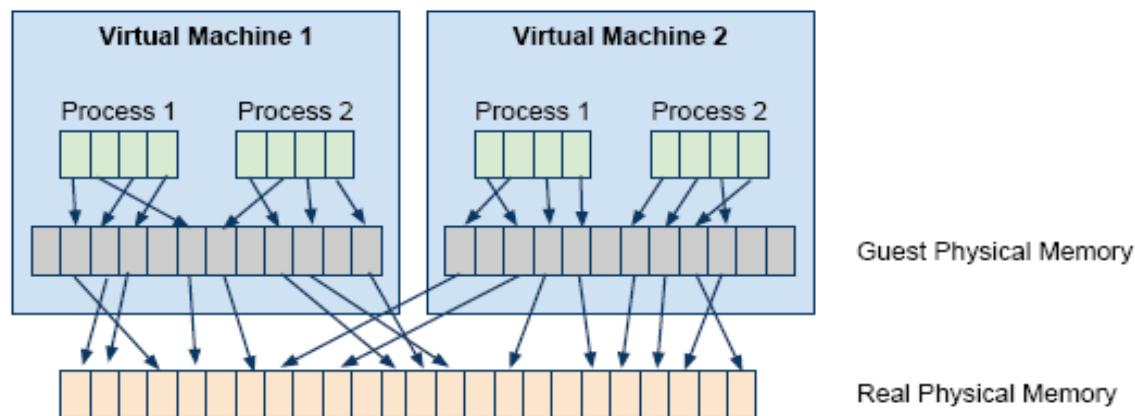
Requirements of real Hypervisor

- VMM at higher privilege level than VMs
 - CPU Virtualization
 - Memory Virtualization
 - Device & I/O Virtualization
- User and System modes
- Privileged instructions only available in system mode
 - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
 - Including page tables, interrupt controls, I/O registers



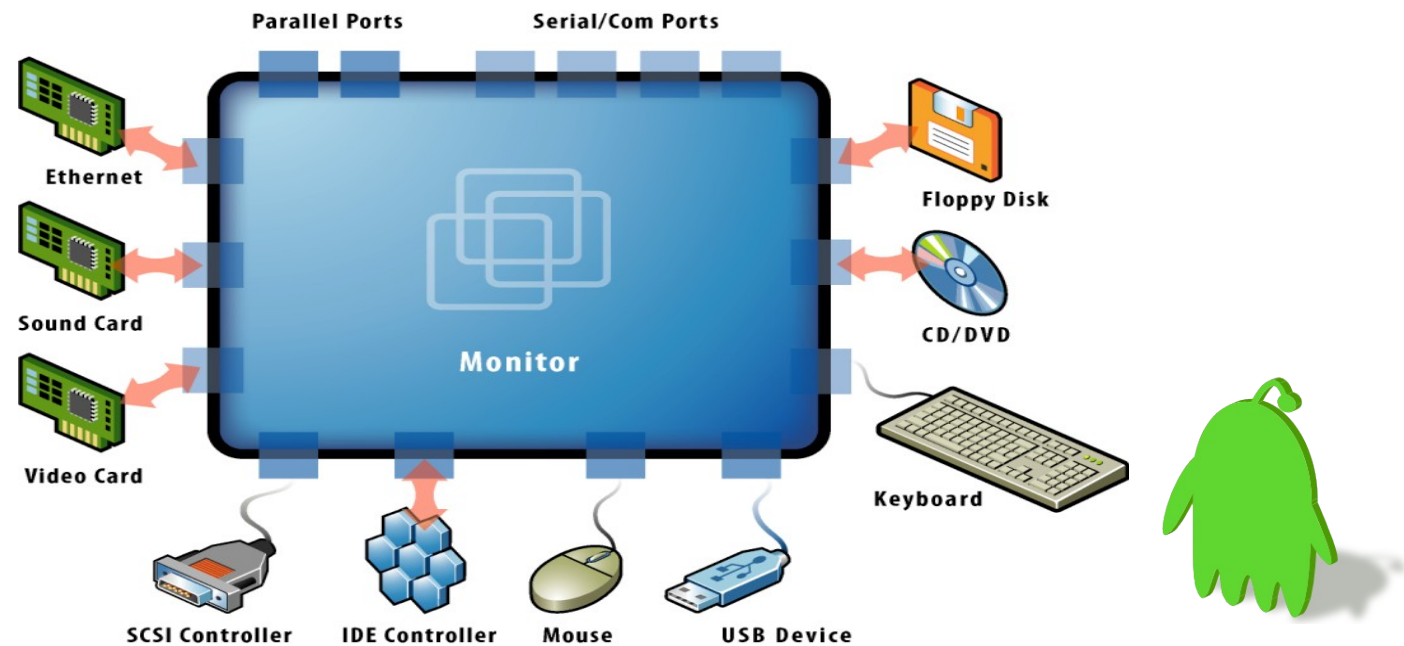
Memory Virtualization

- Deal with allocation of Physical memory among Guest OS
- RAM space shares among Guest OS
- Processors with memory virtualization support is expecting in 2nd generation processors (Intel VT and ARM Cortex-A15)



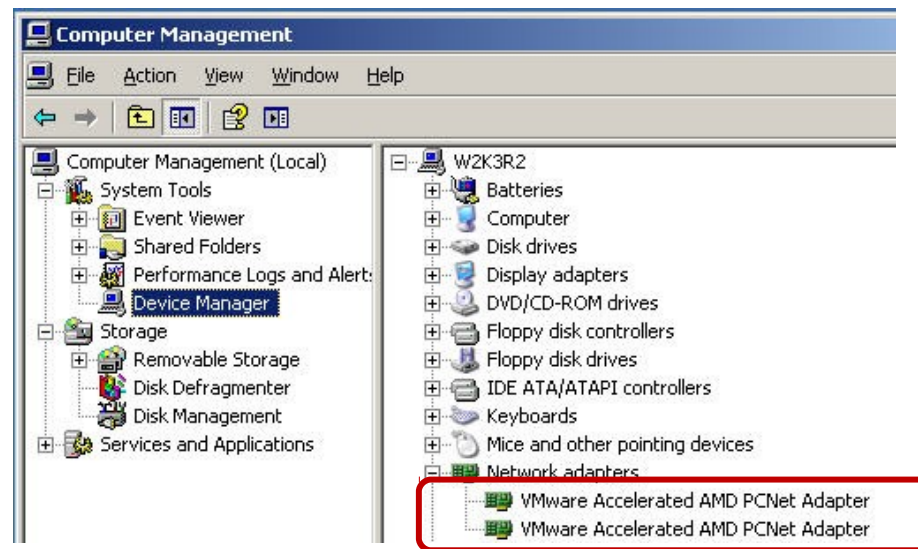
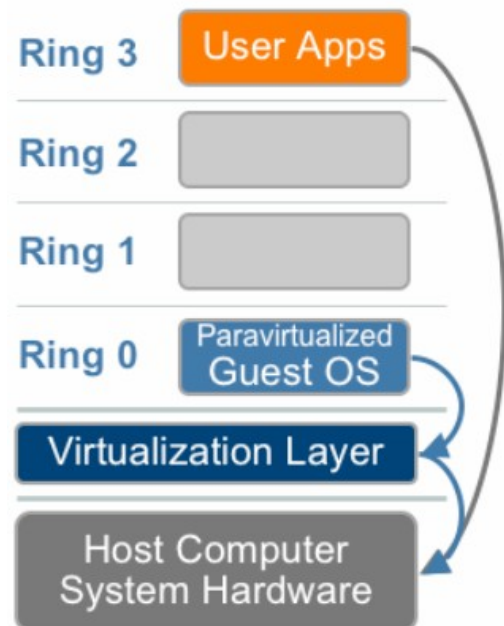
Device and I/O Virtualization

- Deal with routing of I/O requests between virtual devices and the shared physical hardware
- Similar to the single I/O device shared concurrently among different applications.
- Hypervisor virtualizes the physical hardware and present each virtual machine with a standard set of virtual devices



Paravirtualization

- OS or system devices are virtualization aware
- Requirements:
 - OS level – translated/modified kernel
 - Device level – paravirtualized or “enlightened” device drivers



Paravirtualization

- Why all the trouble? Just “port” a guest operating system to the interface of your choice.
- Paravirtualization can
 - provide better performance
 - simplify VMM
- but at a maintenance cost and you need the source code
 - Compromise: Use paravirtualized drivers for I/O performance (KVM virtio, VMware).
- Examples: MkLinux, L4Linux, Xen, . . .



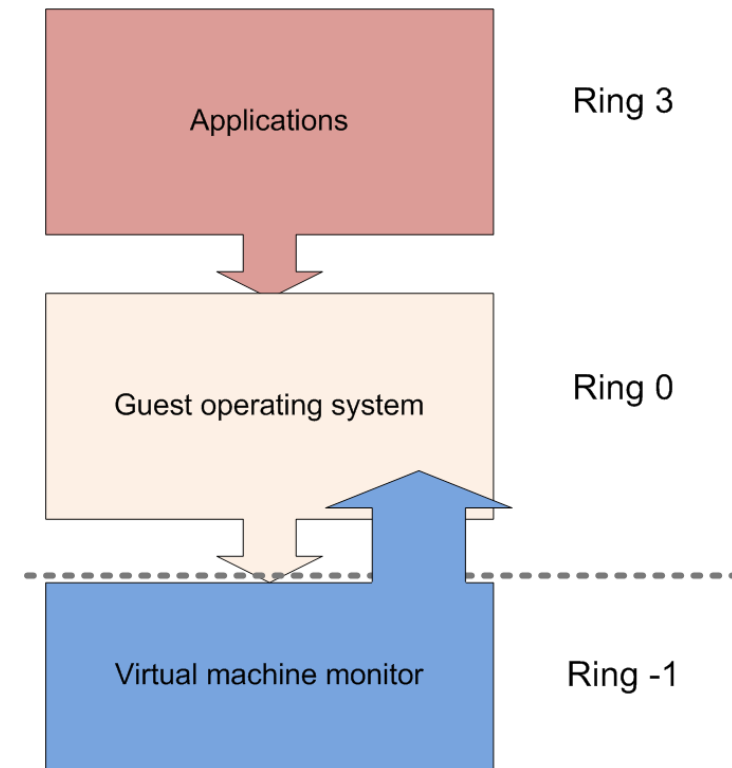
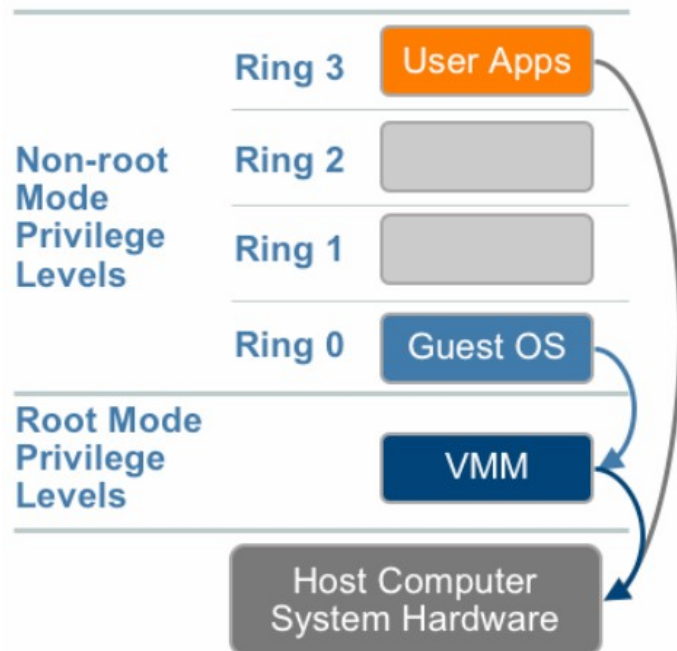
Paravirtualization

- Paravirtualization can also be semi-automated:
 - Sensitive instructions are automatically identified (in compiler output).
 - Sensitive memory access needs to manually identified.
 - Leave markers in binary.
 - On VM load-time, VMM replaces instructions with emulation code.
 - In-Place VMM translates to hypervisor calls.
- Benefits:
 - less effort than plain paravirtualization
 - comparable speed



Hardware-assisted Virtualization

- Hardware is virtualization aware
- Hypervisor and VMM load at Ring -1
- Remove CPU emulation bottleneck
- Provides address bus isolation

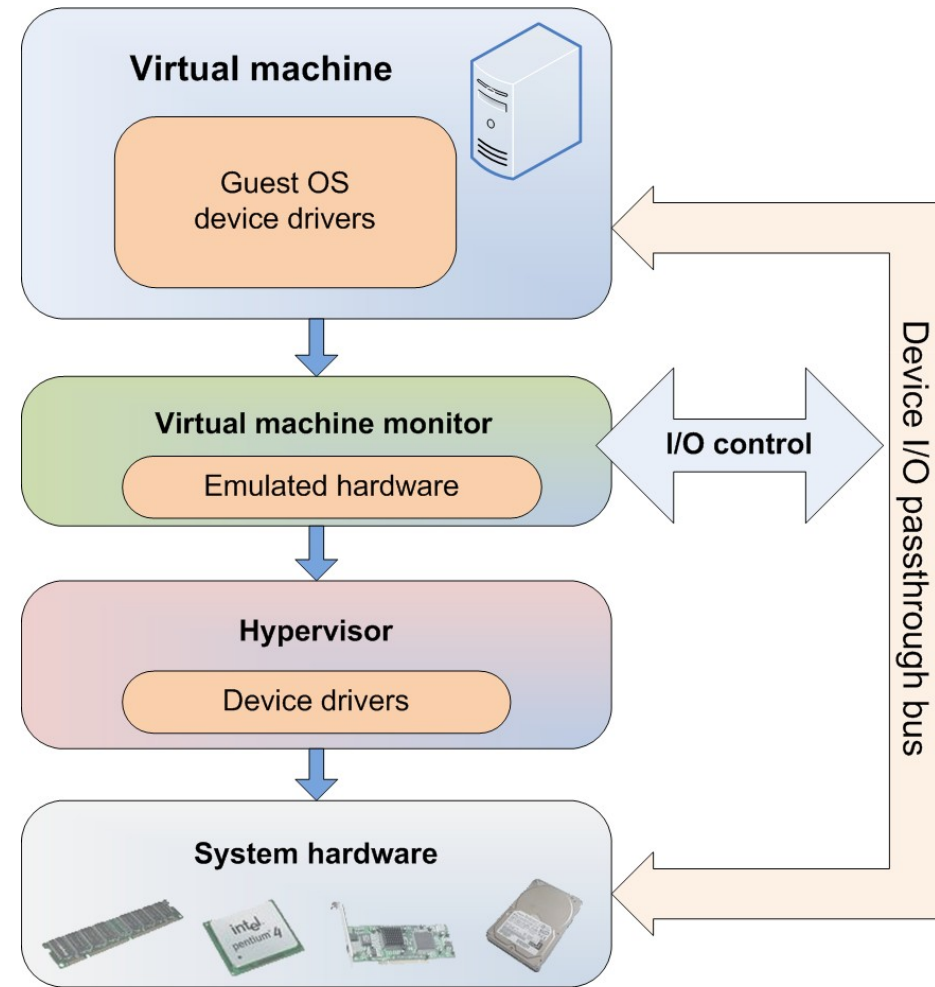


Hardware-assisted virtualization



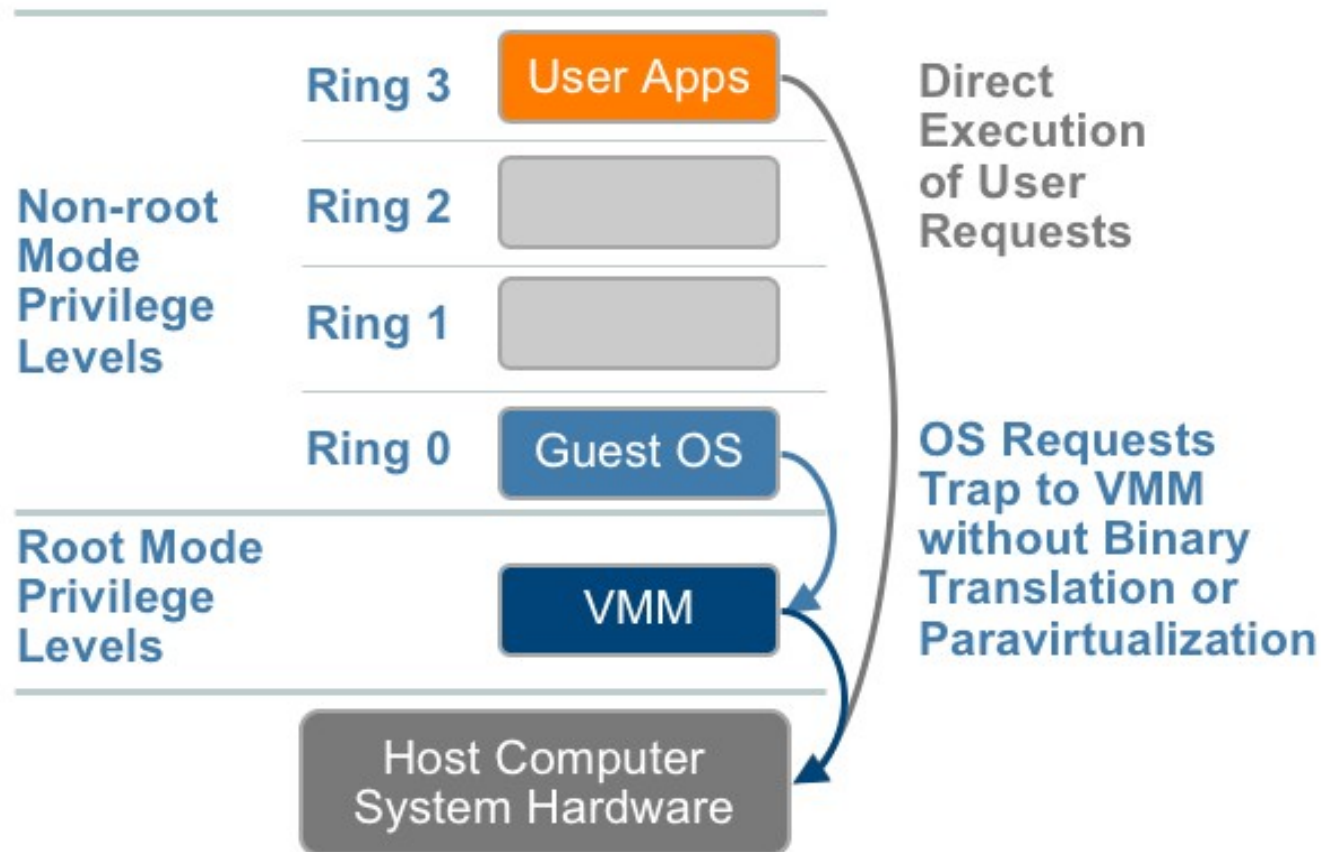
Hardware-assisted Virtualization

- VMM coordinates direct hardware access
- Memory virtualization solved in 2nd generation hardware assisted platforms
- Passthrough I/O has limited use cases without IOV (I/O Virtualization)
<http://www.pcisig.com/specifications/iov/>



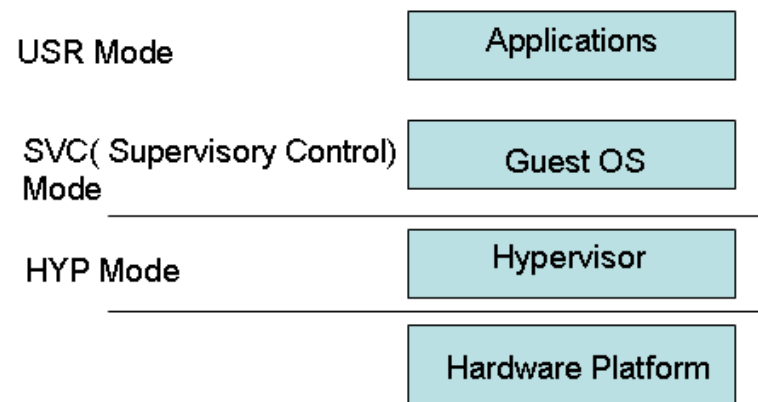
Hardware-assisted Virtualization in x86

- VT technology enables new execution mode (VMX-Root Mode in x86 by Intel) in the processors to support virtualization
- Hypervisor runs in a root mode below Ring0
- OS requests trap VMM without binary translation or PV
- Specialized Hardware support is required
- A special CPU privileged mode is to be selected to support



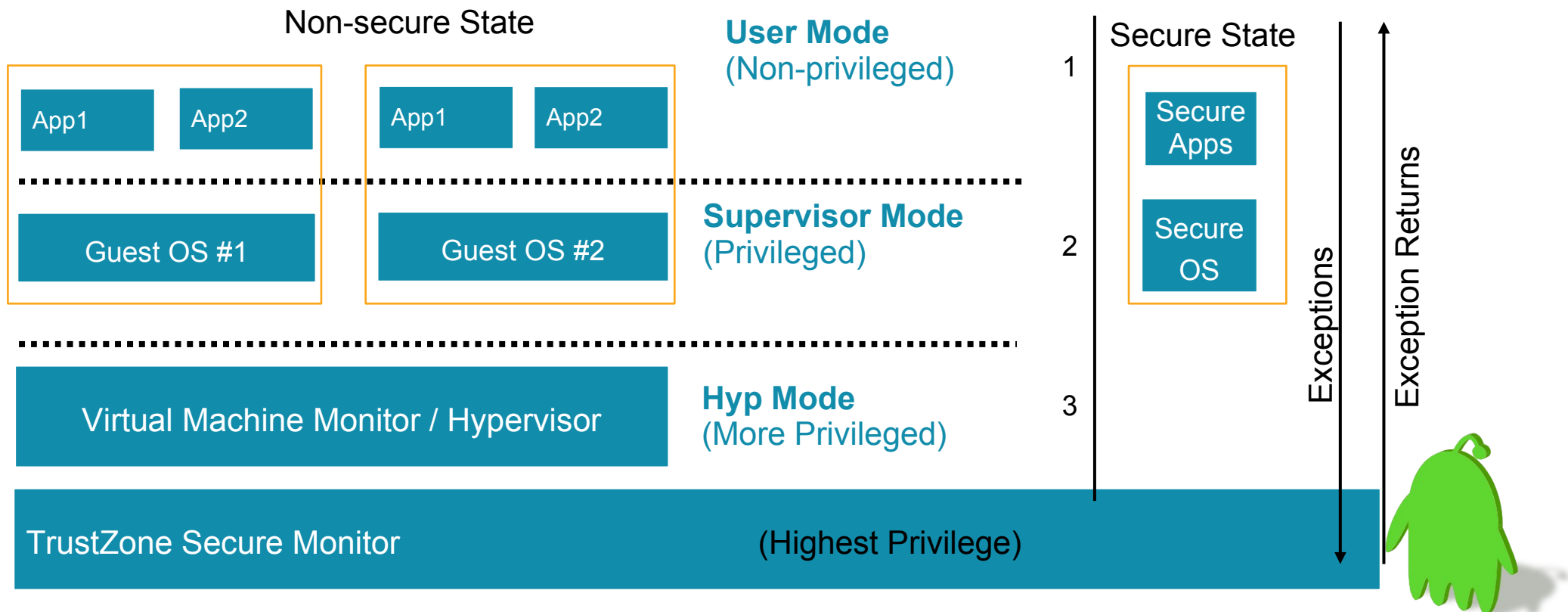
Hardware-assisted Virtualization in ARM

- Enable new execution mode Hypervisor (HYP)
- Hypervisor runs in a Hypervisor (HYP) mode
- Guest OS Runs in Supervisory Control (SVC) mode
- Applications runs in User (USR) mode



Virtualization: Third Privilege

- Guest OS same kernel/user privilege structure
- HYP mode higher privilege than OS kernel level
- VMM controls wide range of OS accesses
- Hardware maintains TZ security (4th privilege)



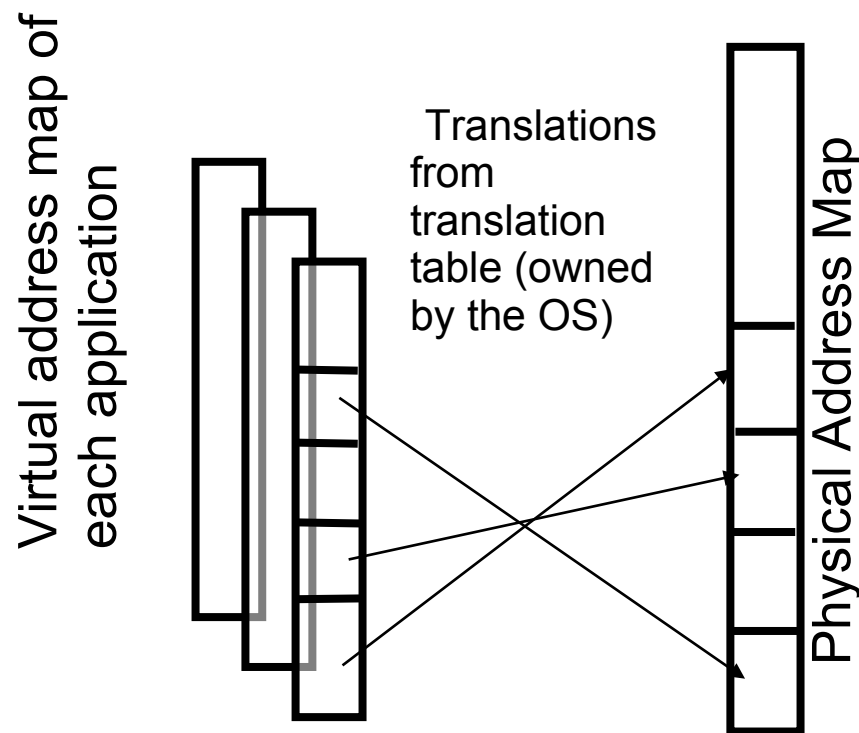
Virtualization Extensions: The Basics

- New Non-secure level of privilege to hold Hypervisor
 - Hyp mode
- New mechanisms avoid the need Hypervisor intervention for:
 - Guest OS Interrupt masking bits
 - Guest OS page table management
 - Guest OS Device Drivers due to Hypervisor memory relocation
 - Guest OS communication with the interrupt controller (GIC)
- New traps into Hyp mode for:
 - ID register accesses and idling (WFI/WFE)
 - Miscellaneous “difficult” System Control Register cases
- New mechanisms to improve:
 - Guest OS Load/Store emulation by the Hypervisor
 - Emulation of trapped instructions through syndromes

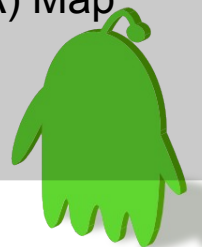
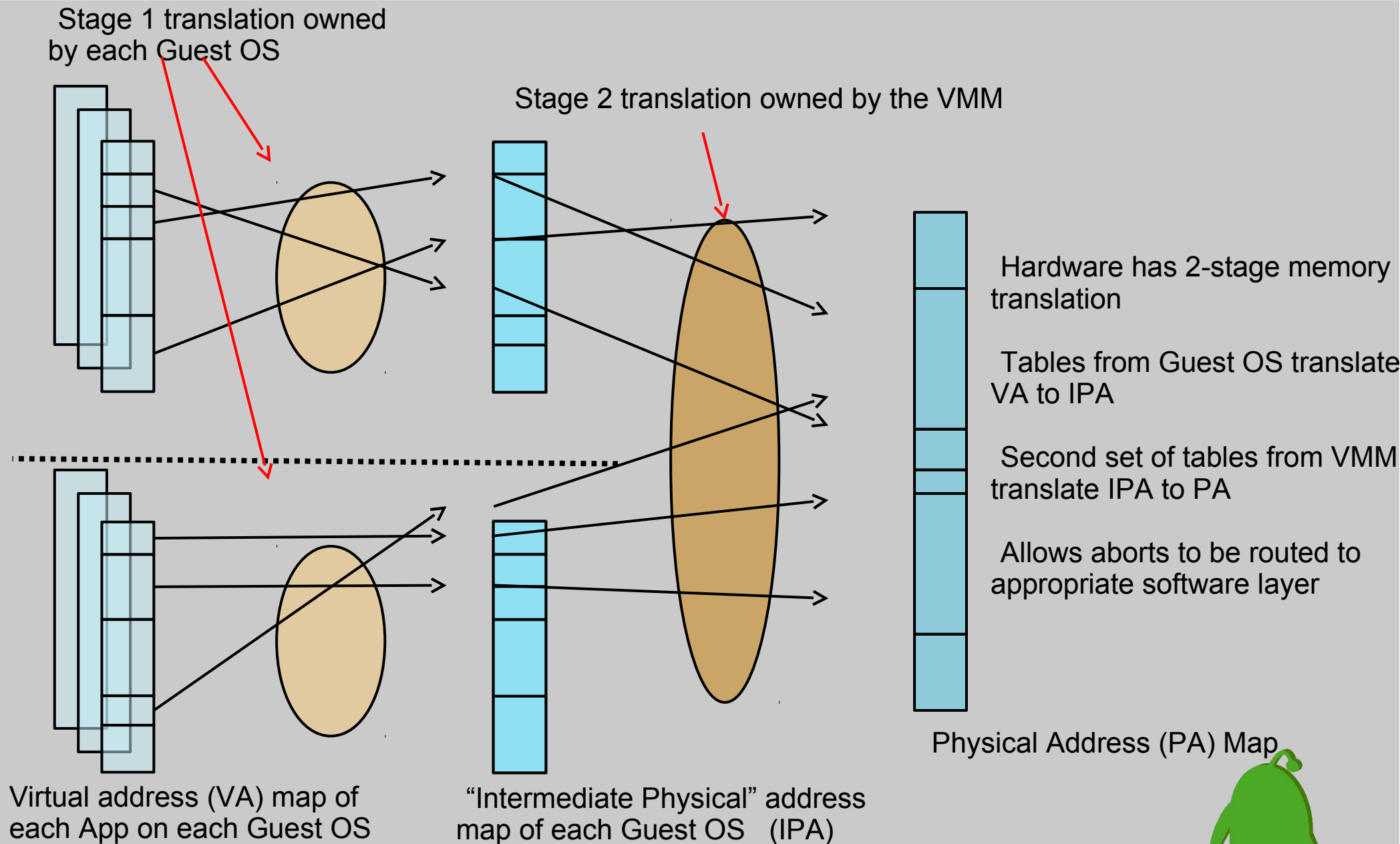


Memory - the Classic Resource

- Before virtualization: the OS owns the memory
 - Allocates areas of memory to the different applications
 - Virtual Memory commonly used in “rich” operating systems

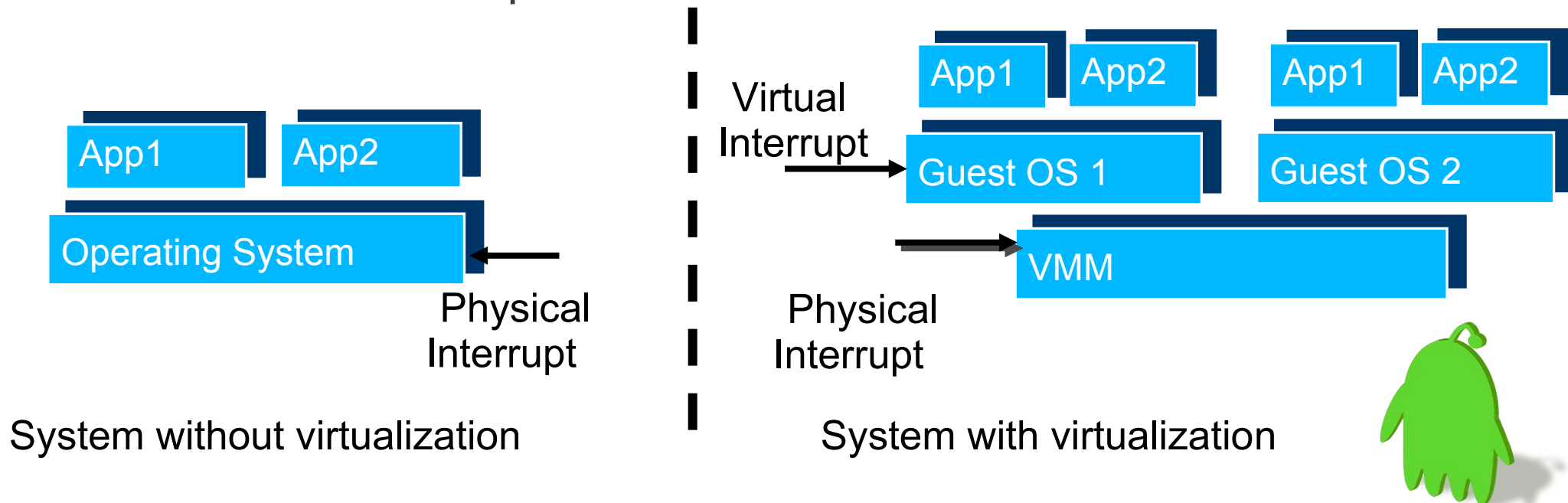


Virtual Memory in Two Stages



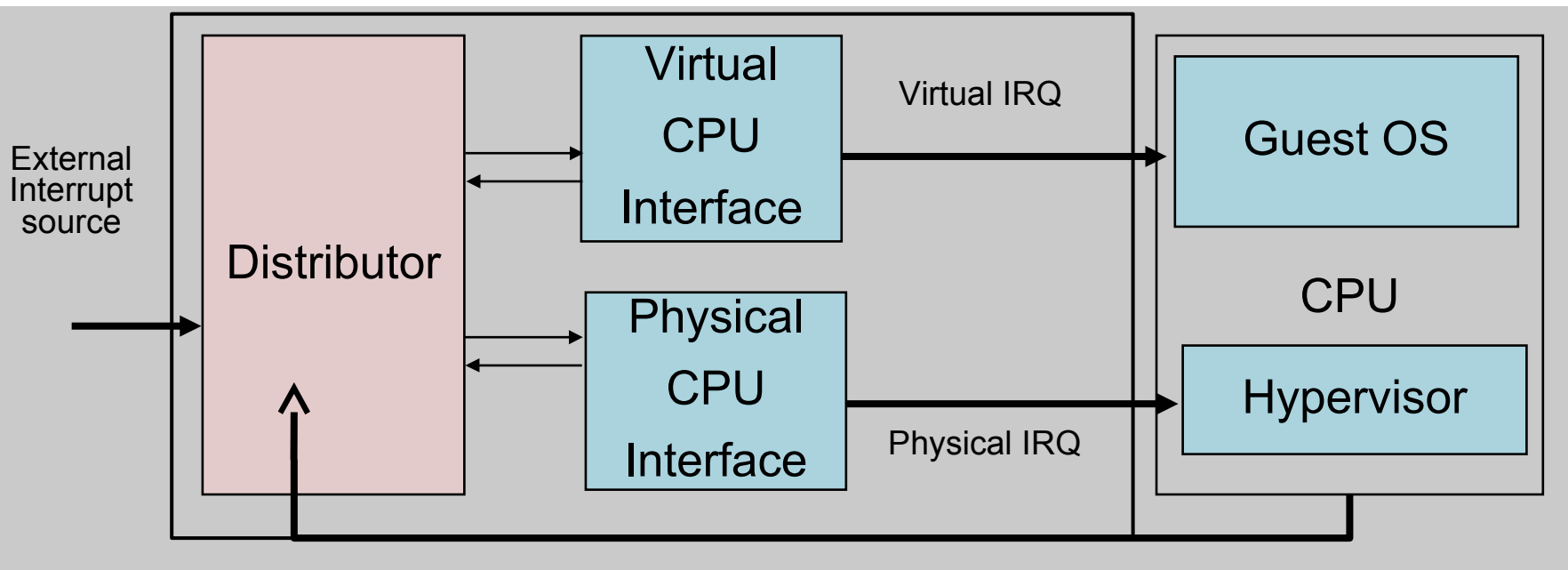
Classic Issue: Interrupts

- An Interrupt might need to be routed to one of
 - Current or different Guest OS
 - Hypervisor
 - OS/RTOS running in the secure TrustZone environment
- Basic model of the ARM virtualization extensions
 - Physical interrupts are taken initially in the Hypervisor
 - If the Interrupt should go to a Guest OS, Hypervisor maps a “virtual” interrupt for that Guest OS



Virtual interrupt example

- External IRQ (configured as virtual by the hypervisor) arrives at the GIC
- GIC Distributor signals a Physical IRQ to the CPU
- CPU takes HYP trap, and Hypervisor reads the interrupt status from the Physical CPU Interface
- Hypervisor makes an entry in register list in the GIC
- GIC Distributor signals a Virtual IRQ to the CPU
- CPU takes an IRQ exception, and Guest OS running on the virtual machine reads the interrupt status from the Virtual CPU Interface

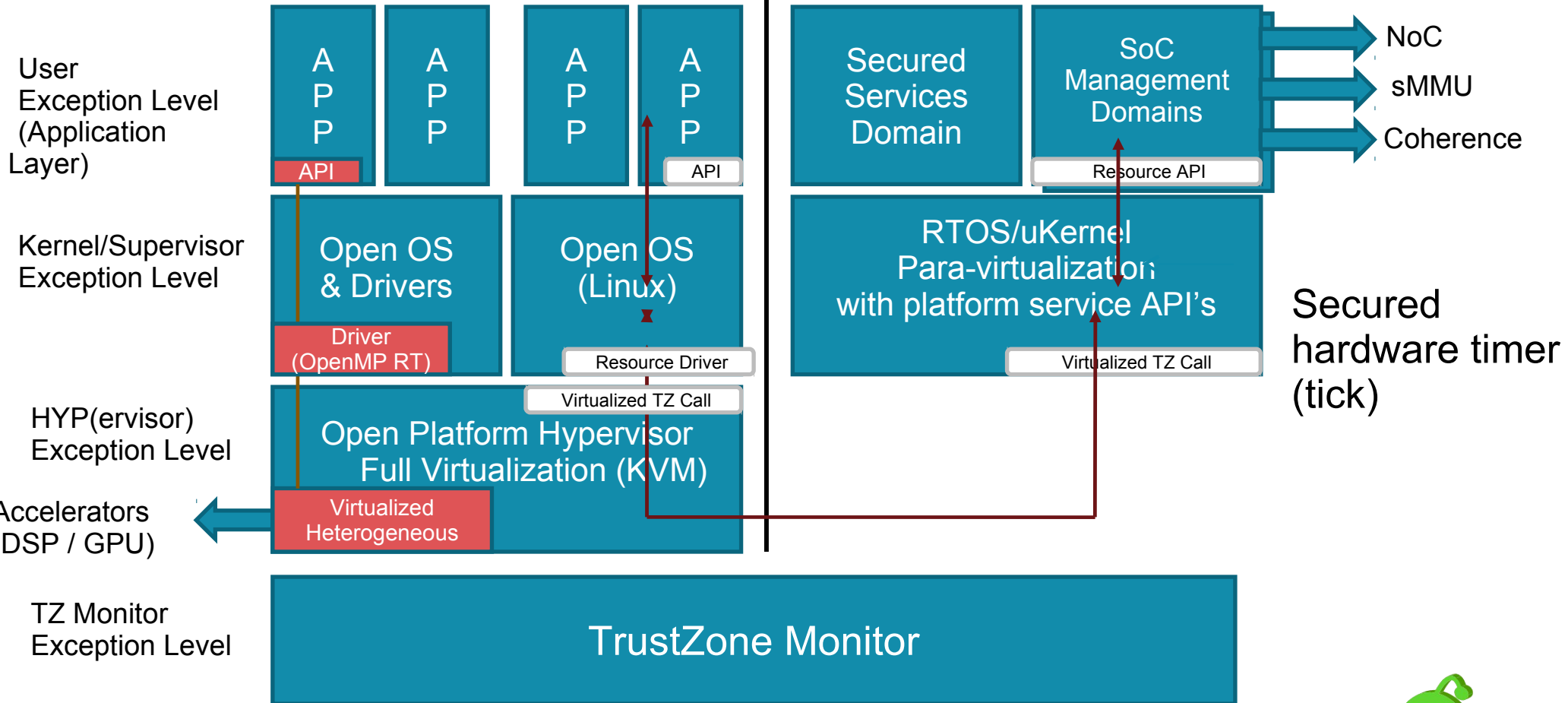


Spanning Hypervisor Framework

ARM Non-Secure Execution Environment

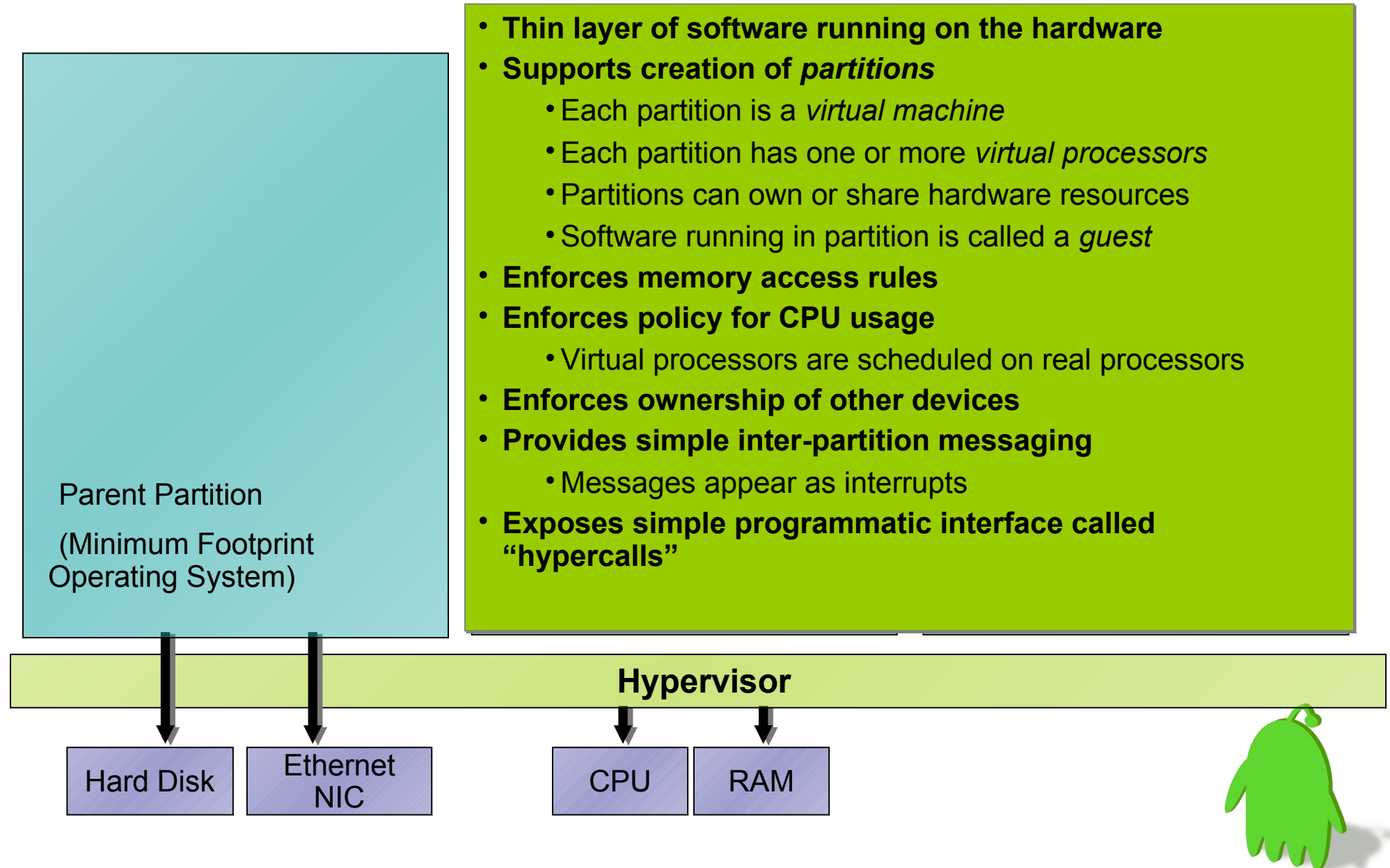
ARM Secure Execution Environment

Platform Resources



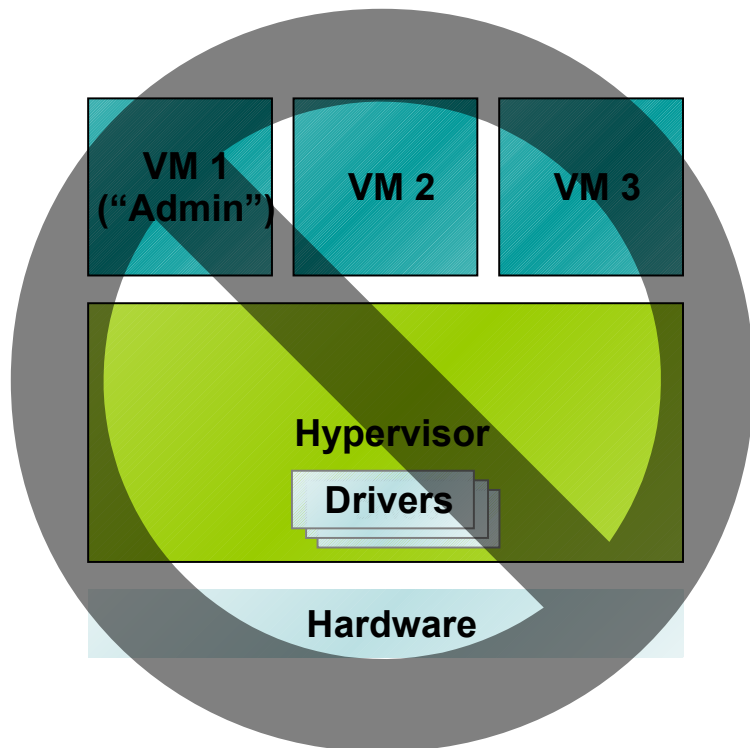
Source: Hardware accelerated Virtualization in the ARM Cortex™ Processors, John Goodacre, ARM Ltd. (2011)

What does Hypervisor looks like

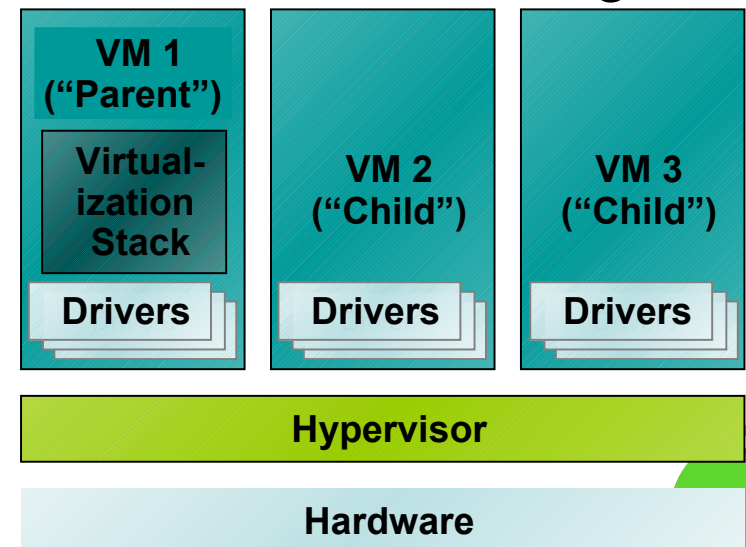


Monolithic vs. Microkernel

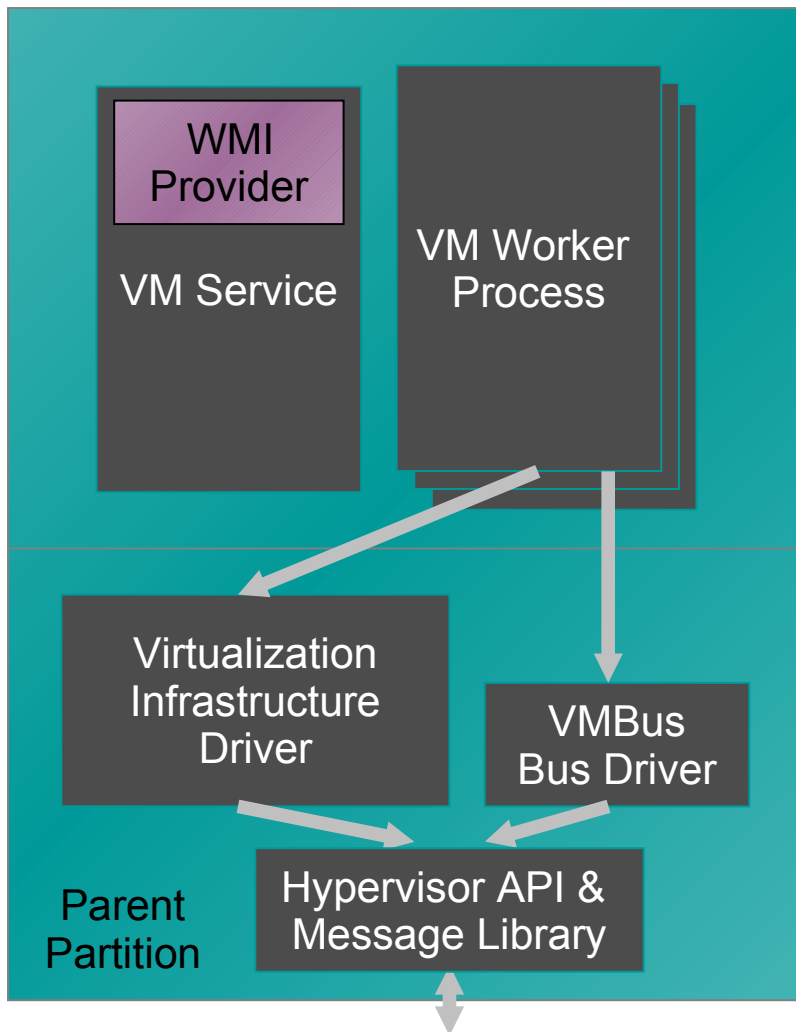
- Monolithic hypervisor
 - Simpler than a modern kernel, but still complex
 - Contains its own drivers model



- Microkernel based hypervisor
 - Simple partitioning functionality
 - Increase reliability and minimize TCB
 - No third-party code
 - Drivers run within guests



Virtualization Stack



- **Collection of user-mode and kernel-mode components**
 - Runs within a partition on top of a (minimal) OS
 - Contains all VM support not in the hypervisor
- **Interacts with hypervisor**
 - Calls the hypervisor to perform certain actions
 - Responds to messages from the hypervisor or from other partitions
- **Creates and manages a group of “child partitions”**
 - Manages memory for child partitions
 - Virtualizes devices for child partitions
- **Exposes a management interface**

Child Partition 1

Child Partition 2

Hypervisor

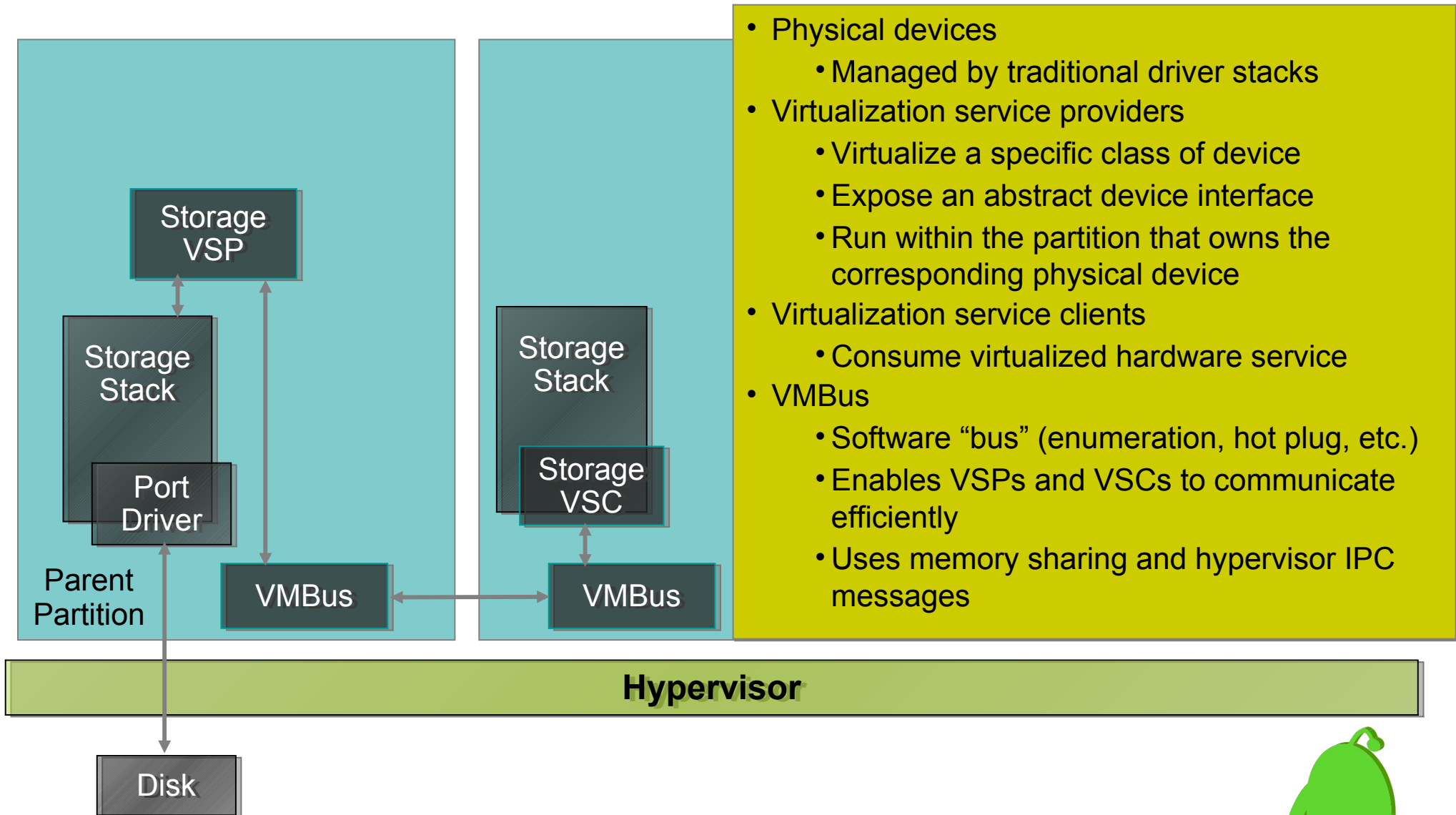


Device Virtualization

- Standard VSP (Virtualization service providers)
 - Storage: support difference drive chains
 - Network: provide virtualized network mechanism
 - Video: 2D/3D graphics w/ or w/o HW acceleration
 - USB: allow a USB device to be assigned to a partition
 - Input: keyboard, mouse, touchscreen
 - Time: virtualization for RTC hardware



Device Virtualization



Embedded Virtualization Issues

- Memory footprint
- Security
 - Increases size of Trusted Computing Base
- Direct IO Access
- Emulate IO
- Virtual IO
- Real-time support



Direct I/O Access

- Guest can directly access physical IO without host involvement
 - Native speed
- IOMMU provides isolation and physical address translation (DMA)
 - Translation could be done with guest modifications
- Issues:
 - IOMMU **required** for DMA isolation
 - Limited by number of physical IO devices
 - Guests must have device drivers
 - What about legacy guests on new hardware?
 - Breaks migration
 - IRQ delivery and routing



Emulated I/O

- Host software emulates guest IO accesses
- Issues:
 - Must write software to (perfectly?) emulate hardware
 - Dramatic increase in IO latency
 - Host OS must have physical device drivers
 - Device driver availability, licensing concerns



Virtual I/O

- No hardware at all, just inter-guest data transfer
- New guest device drivers co-operate with host
- Issues:
 - Requires guest modification (at least new device drivers)
 - Host OS still needs physical IO drivers



Embedded Hypervisors for ARM



Embedded Hypervisors for ARM

(open source part)

- Xen
 - Xen-arm, contributed by **Samsung**
ARM9, ARM11, ARM Cortex-A9 MP
 - Xen-arm-cortex-a15, contributed by **Citrix** -
<https://lkml.org/lkml/2011/11/29/265>
ARM Cortex-A15
- OKL4 (from open to close source), OKLabs
- L4Linux, TU Desden
- KVM ARM porting
 - Columbia University
 - NTHU, Taiwan
- Xvisor: supports ARMv5, ARMv7, ARMv7+VE
- Codezero



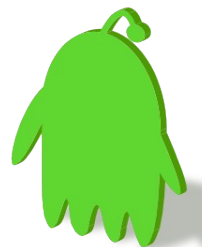
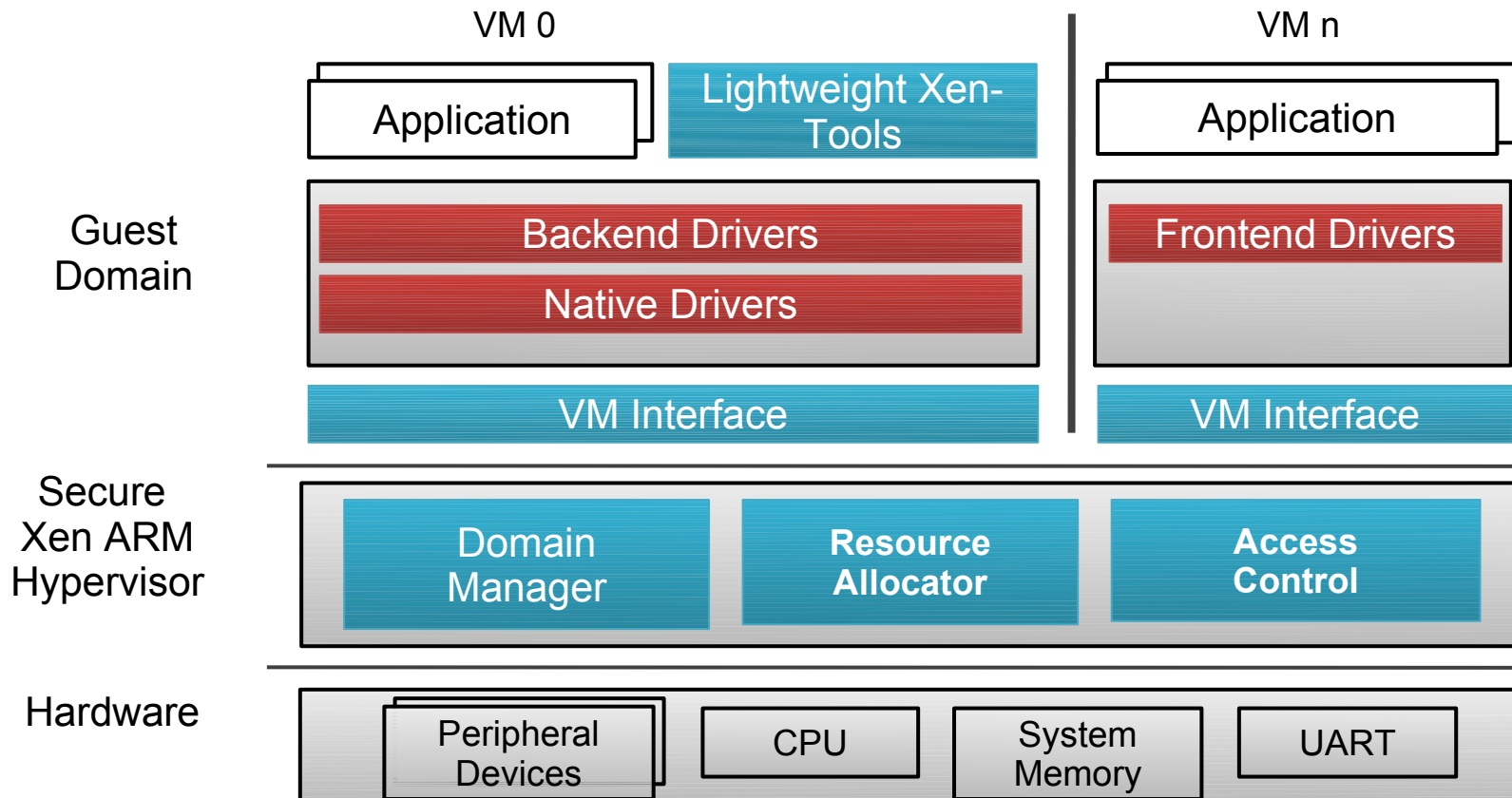
Xen-ARM (Samsung)

Goals

Lightweight virtualization for secure 3G/4G mobile devices

- High performance hypervisor based on ARM processor
- Fine-grained access control fitted to mobile devices

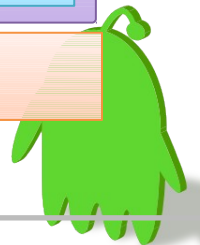
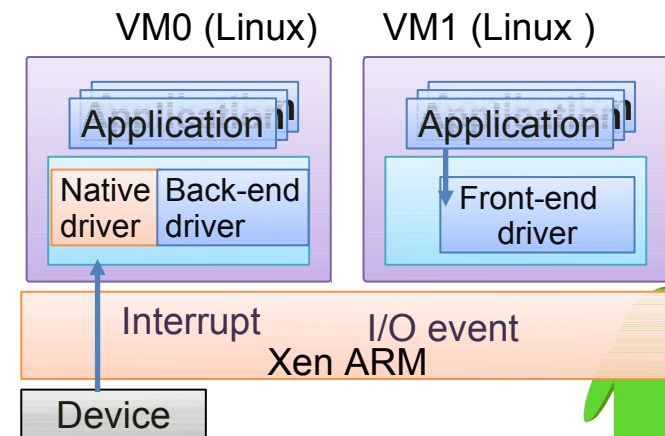
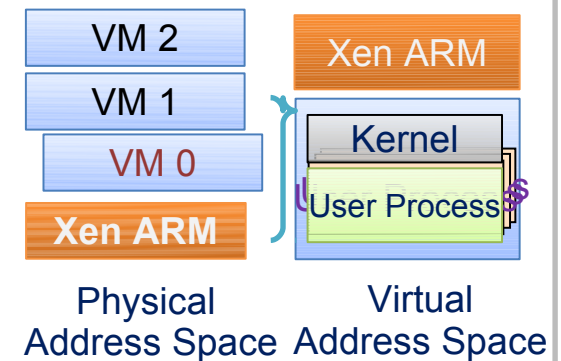
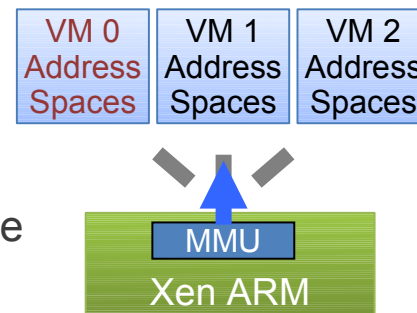
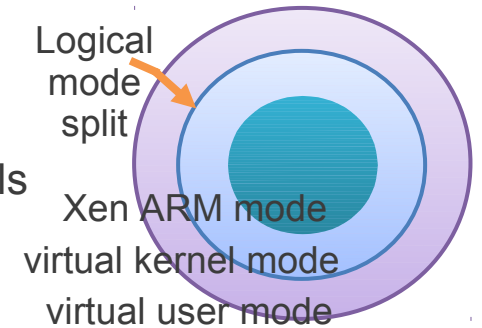
Architecture of Xen ARM

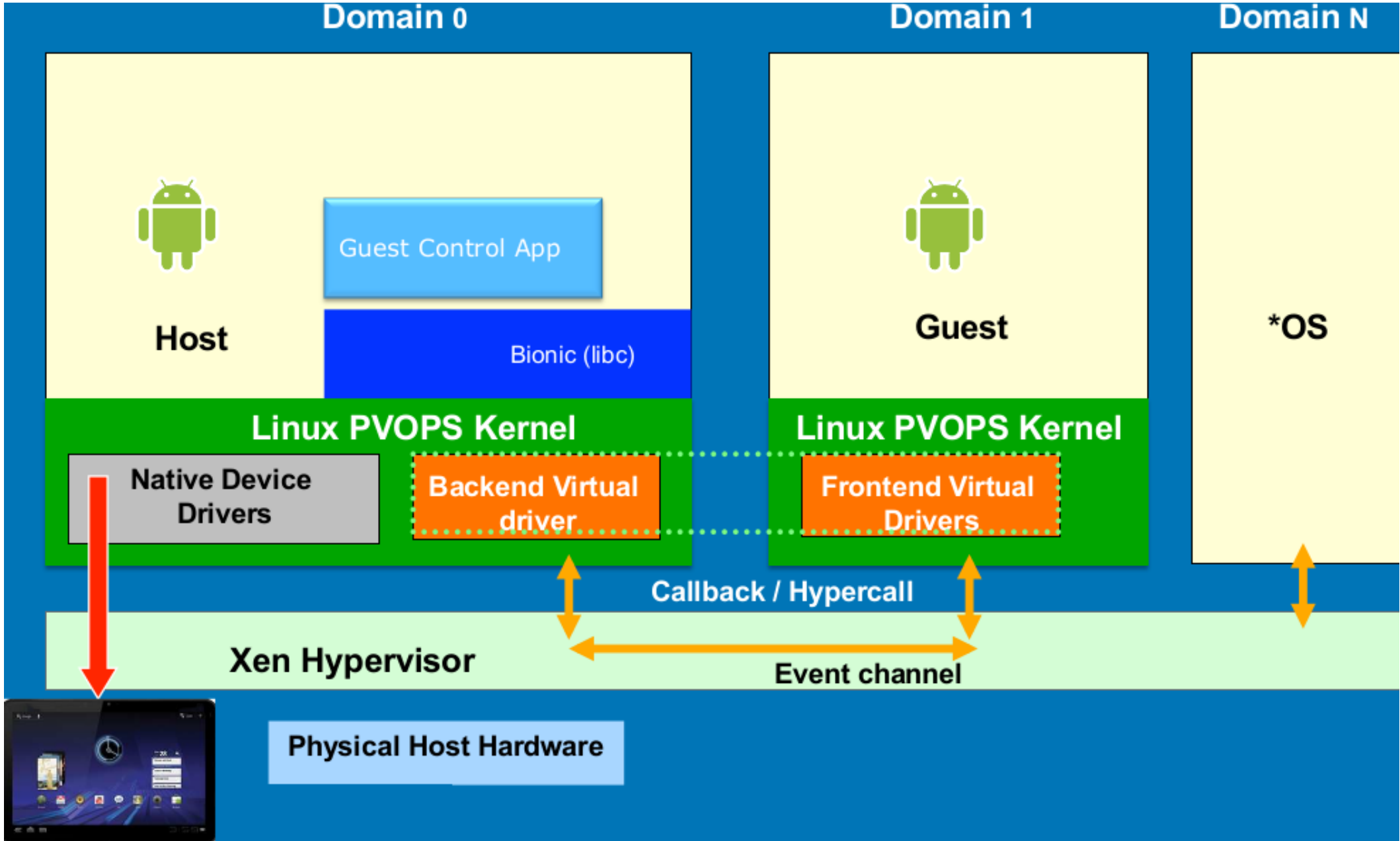


Xen-ARM (Samsung)

Overview

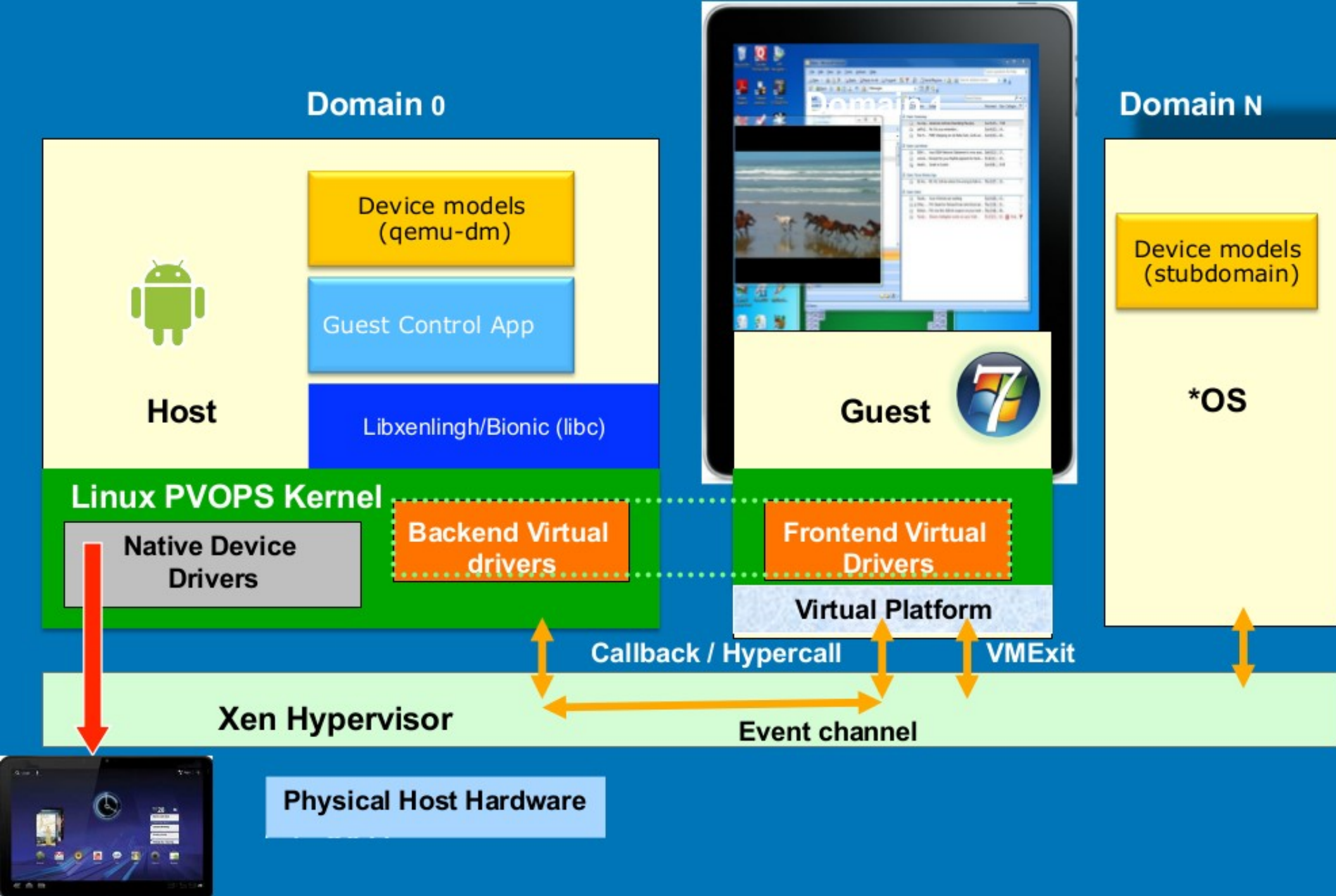
- CPU virtualization
- Virtualization requires 3 privilege CPU levels, but ARM supports 2 levels
 - Xen ARM mode: supervisor mode (most privileged level)
 - Virtual kernel mode: User mode (least privileged level)
 - Virtual user mode: User mode (least privileged level)
- Memory virtualization
- VM's local memory should be
 - protected from other VMs
 - Xen ARM switches VM's virtual address space
 - using MMU
 - VM is not allowed to manipulate MMU directly
- I/O virtualization
- Split driver model of Xen ARM
 - Client & Server architecture for shared I/O devices
 - Client: frontend driver
 - Server: native/backend driver





- Xen without assisted hardware VM





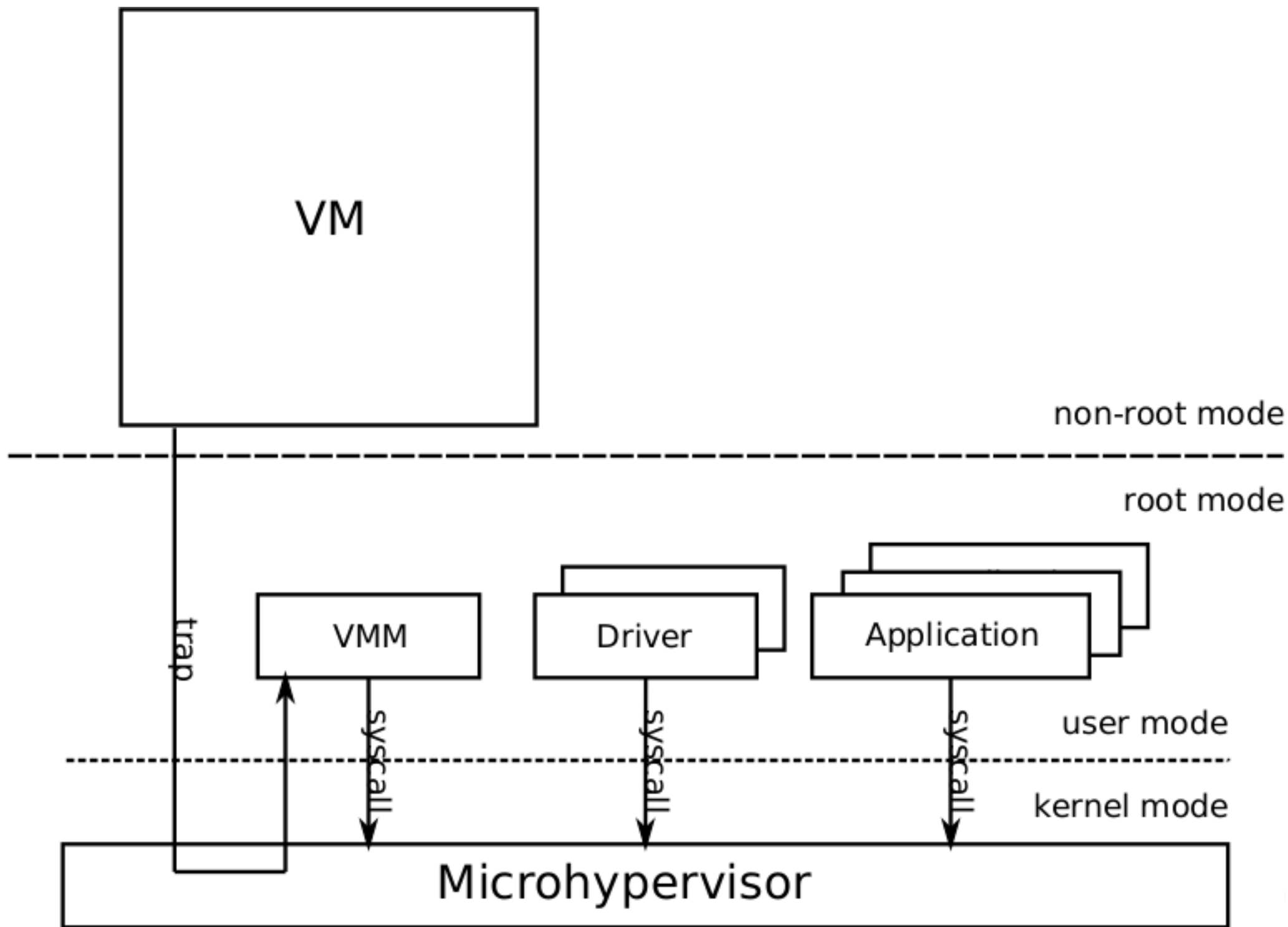
- Xen with assisted hardware VM

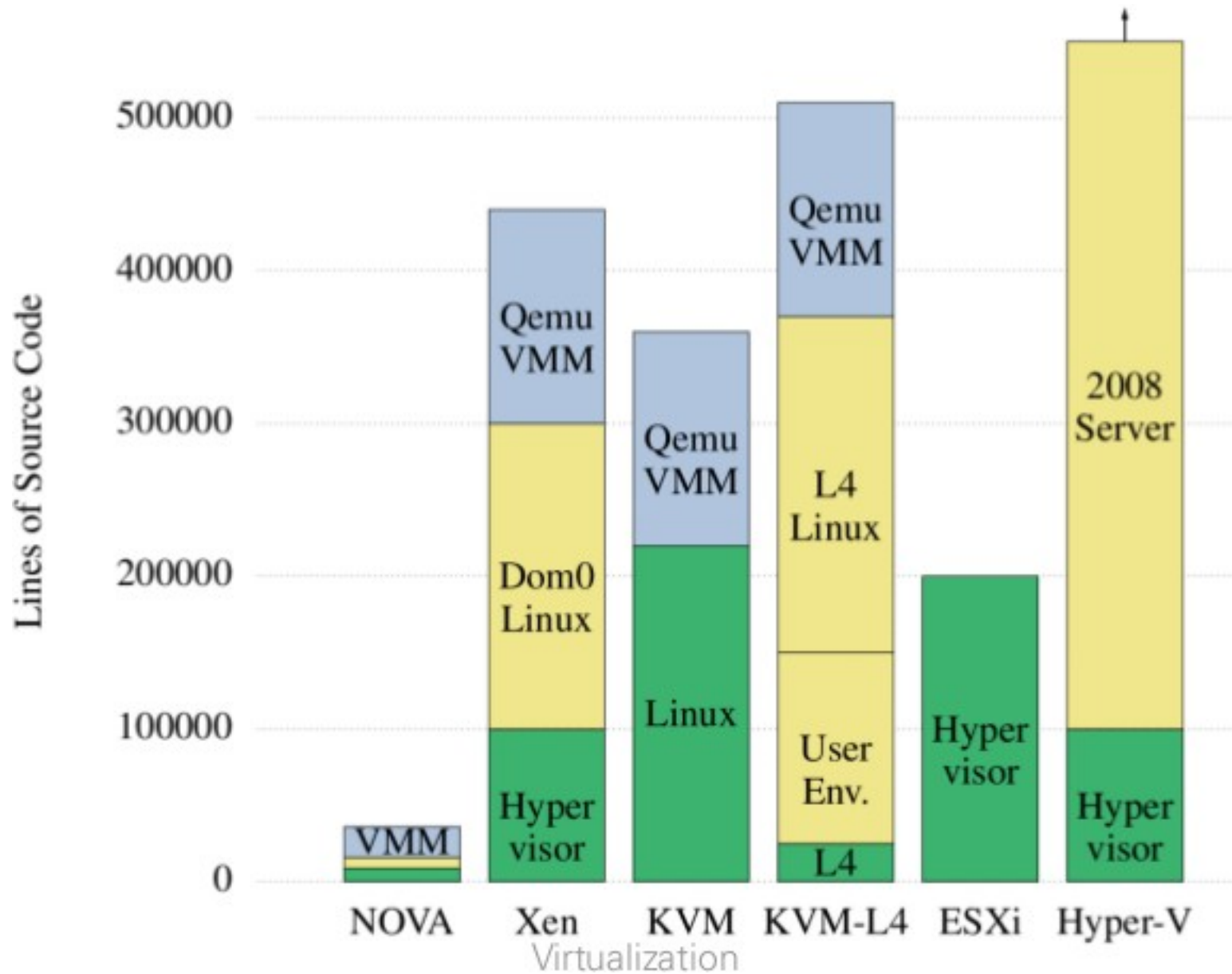


Micro-hypervisor

- Microvisor – OKL4 4.0
- Research projects such as NOVA, Coyotos, and seL4
- Microhypervisor
 - “the kernel part”
 - provides isolation
 - mechanisms, no policies
 - enables safe access to
 - virtualization features to userspace
- VMM
 - “the userland part”
 - CPU emulation
 - device emulation







Source: **VIRTUALIZATION**, Julian Stecklina, TU Dresden



Advantage of NOA architecture:

Reduce TCB of each VM

- Micro-hypervisor provides low-level protection domains
 - address spaces
 - virtual machines
- VM exits are relayed to VMM as IPC with selective guest state
- one VMM per guest in (root mode) userspace:
 - possibly specialized VMMs to reduce attack surface
 - only one generic VMM implemented



Reference

- 前瞻資訊科技—虛擬化，薛智文，台大資訊所 (2011)
- ARM Virtualization: CPU & MMU Issues, Prashanth Bungale, vmware
- Hardware accelerated Virtualization in the ARM Cortex™ Processors, John Goodacre, ARM Ltd. (2011)
- Hypervisors and the Power Architecture
<http://www.techdesignforums.com/embedded/embedded-topics/embedded-development-platforms/hypervisors-and-the-power-architecture/>
- Philippe Gerum, State of Real-Time Linux: Don't Stop Until History Follows, ELC Europe 2009





<http://0xlab.org>