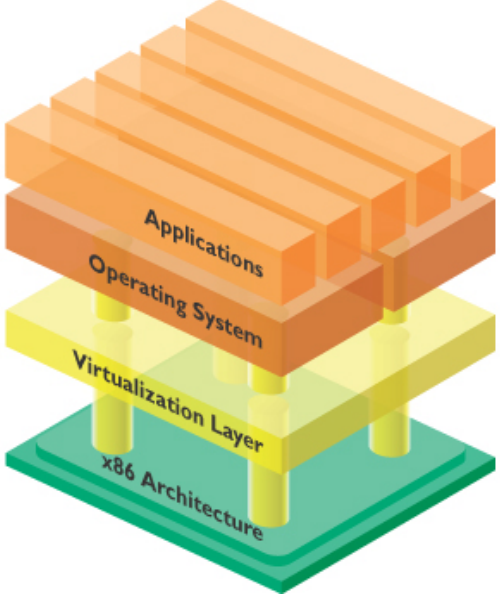


Virtualization

André Przywara, AMD OSRC

June 27th, 2008
Chaos Computer Club Dresden

Agenda

- What is Virtualization?
- Virtualization technologies
- Hardware virtualization: AMD-V™
- Virtualization in use
- Virtualization demo
- Virtualization outlook

What is Virtualization

What is virtualization?

- precise definition difficult
- easy way: running several operating systems at once
- more general: sharing computer resources
- abstraction layer to partition a computer
- differing from
  - Emulation
  - Containers
  - Simulation

Virtualization dictionary

- **Host:** physical machine
- **Guest:** emulated or virtualized machine
- **Hypervisor** (**HV**) or **V**irtual **M**achine **M**onitor (**VMM**)**:** System software that manages virtualization
- **Paravirtualization:** guest is modified to ease or allow virtualization
- **Full virtualization:** allows to run unmodified OS kernels
- **Hardware virtualized machine** (**HVM**)**:** using hardware virtualization
- **Domain:** XEN term for a guest, could be privileged (Dom0) or ordinary (DomU)

The Virtualization Gap

The Idea of Virtualization dates back into the 1970s

Popek and *Goldberg* described the requirements for efficient virtual machines (1974):

- **Equivalence** (VM run the same as native machines)
- **Resource Control** (Hypervisor has control over all resources)
- **Efficiency** (most instruction run without hypervisor intervention)

Supposed principle: Execute all instructions in user mode, system commands will trap and can be emulated

The x86 architecture does (did) not fulfill these requirements

Example: pushf; pop eax; and ax, 0FEFFh; push eax; **popf**

Several (17) other commands exists

This gap can be bridged by software or by hardware

Software solution is not trivial in development and testing

One hardware implementation is AMD-V

Why do we need Virtualization ?

- Computers grow bigger and bigger
- Not every application can benefit from this
- Maintenance of large systems can be problematic
- Splitting them up can be a solution
- Computers are capable of running several OS at once
- Possible usage scenarios:

Server consolidation

Legacy applications

System isolation

Development

Server consolidation

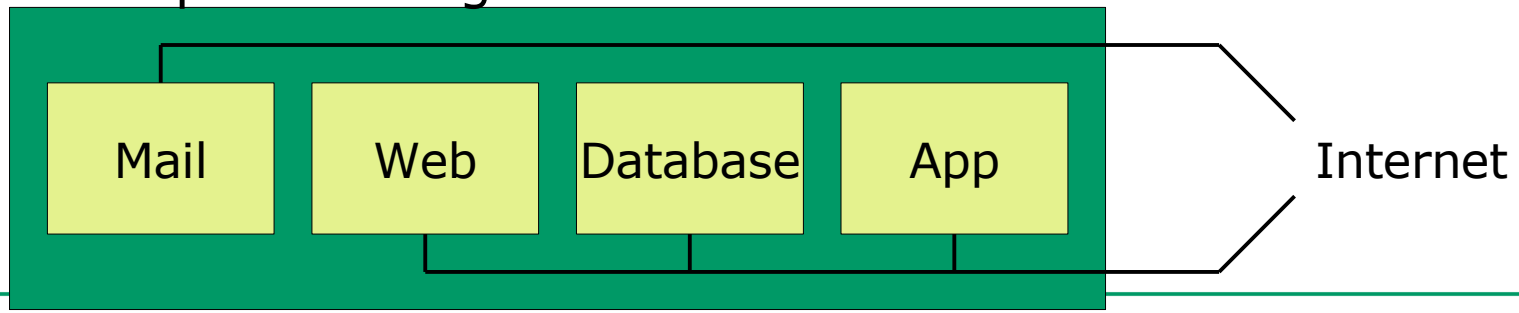
- Imagine important services running for years now on some ageing hardware
- Old hardware is capable enough, but likely to fail
- Moving to a new computer can cause pain
- => The service can become a **virtualized service**
- Old operating system continues to run inside a *container*
- Containers can be moved around
- Very short downtimes
- Almost no migration issues
- Several services can be merged into one piece of hardware
- Virtualization guarantees separation between services

Legacy applications

- Some applications require certain environments
  - Certain OS versions (RHEL3, Windows 2000, etc.)
  - Old incompatible libraries (thread libs, old Apache, ...)
  - 32bit only
  - Strange set-up compromising security
  - Processor numbers limited due to licensing issues
- Those applications can be trapped inside a VM
- Each VM contains one service
- VMs can be tailored to fit application's demand
- Several 32bit applications together can use more than 4 GB of RAM

System isolation

- Enterprise critical services should be isolated
- Don't run a mail and a web server on one machine!
- Compromising one service probably compromises the other
- But: not every services requires a whole machine
- Putting each service on one machine isn't the best choice (hardware faults, administration efforts, cost, energy)
- *Solution:* Put each service in one ***virtual*** machine
- Virtualization ensures isolation
- Each system can be secured to a higher level (no modules, only very limited services, stricter access policy)
- Hardware pass through even allows firewall virtualization



Development

- Testing of applications in different environments
  - Different browsers, different operating systems
- Easy rollback of changes
- Using different development suites
- Tracing allows low-level debugging
  - Watch low-level things without expensive hardware debuggers
- Hardware can be changed quickly
  - More or less RAM
  - Different hard-disk sizes and numbers
  - Using ISO images directly for CD-ROMs
  - Switching between SMP and UP
- Testing Client/Server networking systems
  - Virtual nets can be arranged quickly and easily
  - Emulation of multiple clients or servers

Virtualization Technologies

Virtualization Technologies

Different techniques to achieve similar goals:

Emulation (QEMU)

Full Virtualization (VMware, VirtualBox, VirtualPC)

Para-Virtualization (Xen)

HW-Virtualization (Xen/KVM)

Emulation: QEMU

- QEMU is an open source system emulator
- Consists of two parts:
  - A portable, cross-platform CPU emulator
  - Various emulated hardware devices
- Emulates whole systems on several architectures
- Pure userspace (but kernel module option)
- Quite slow, but capable
- Both XEN and KVM use the hardware devices from QEMU
- Although QEMU is an emulator, x86 machines can be semi-virtualized on x86 machines with an optional kernel module
- Performance ratio is about 10% with emulation, better than 50% with kernel module

QEMU emulator: CPU emulation

- QEMU runs under various systems: Linux, BSD, Windows, Solaris, MacOS X
- Code is portable: supports x86, AMD64, PowerPC, Alpha, Sparc, ARM, MIPS, etc.
- Emulated CPUs can be x86, AMD64, ARM, Sparc, PowerPC, MIPS, m68k
- Each guest instruction is split up into µOPS
- µOPS have one C function assigned
- Host compiler compiles and optimizes this C function
- QEMU glues those functions together (at runtime)
- Result is fairly optimized code (compare: JIT)
- Translated code is cached for further loop iterations

QEMU emulator: Hardware emulation

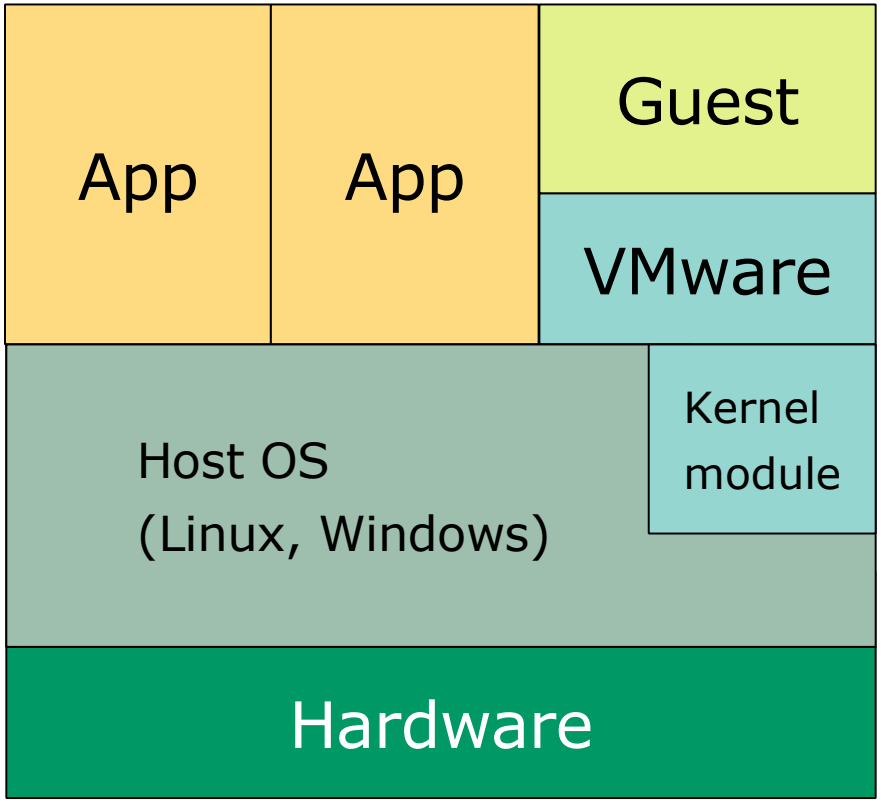
- QEMU comes with a bunch of emulated hardware devices
- Although often describes as “drivers”, these are actually their counterparts
- Drivers give I/O and MMI/O as input, devices connect to real host drivers and emulate the desired behavior
- Devices include networks cards (NE2K_PCI), IDE controller (PIIX4), graphics card (Cirrus Logic 54xx), etc.
- Those hardware is quite common and supported out-of-the-box by most operating systems

The Virtualization Pioneer: VMware

- Informations are rare and shallow
- Based on 1999's paper and some later benchmark papers
- Technique they use is called: **Binary translation**
- Bridging the virtualization gap *by in-place patching*
- Involves the following steps
  - Decode guest instructions (non-trivial for x86)
  - Scan for critical instructions
  - Replace those with some kind of breakpoint
  - Stop at a branch instruction
  - Let this code block run within the guest's context
  - On exit emulate trapping instruction
  - Redo with next code block
- Translated blocks will be cached for later reuse
- Blocks can be chained or linked with jumps
- Works on top of existing OS (Linux or Windows)

VMware technology

- VMware is an ordinary user space application
- uses kernel module to handle memory management and low level issues



- VMware ESX merges host kernel and hyper-visor

VMware's I/O

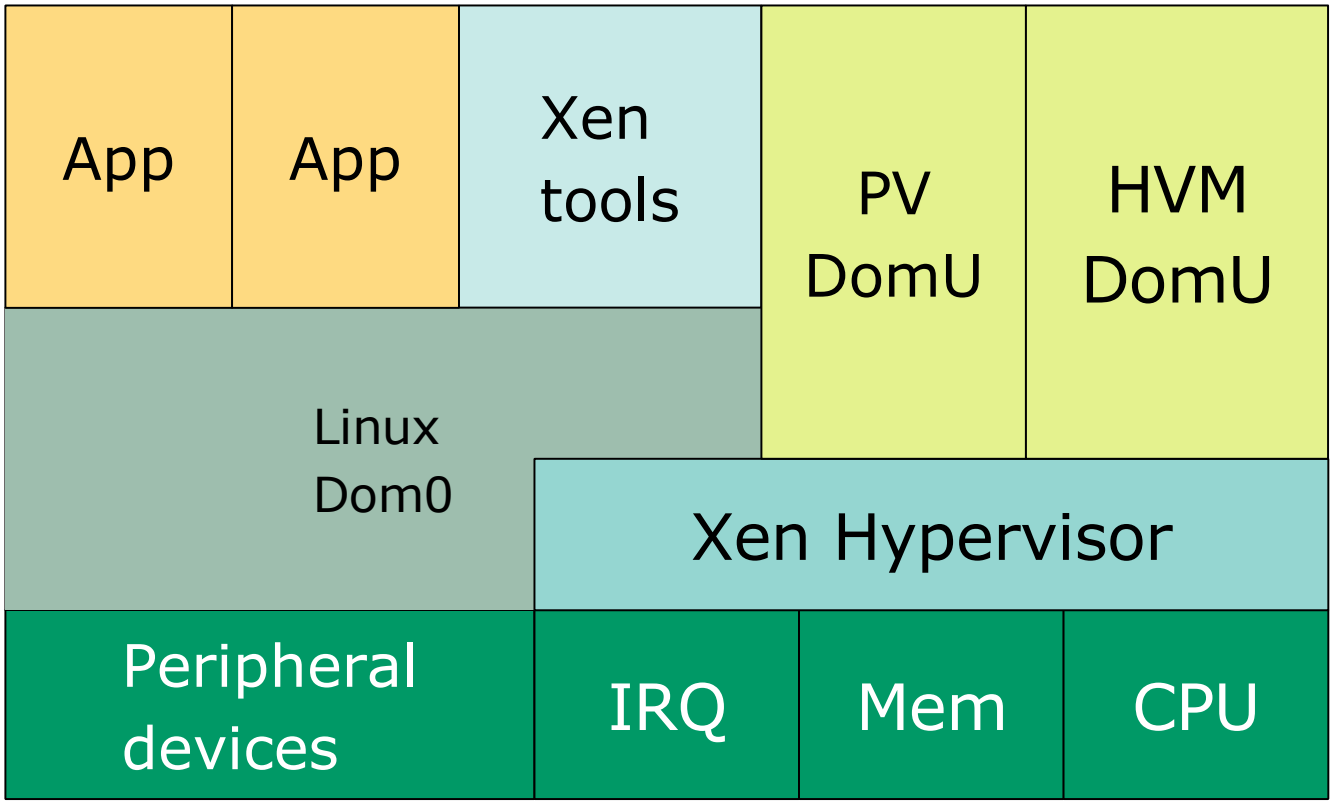
- Emulates simple standard hardware (like QEMU)
- Provides special *guest* drivers to increase performance and usability
- "VMware tools" provide the following features:
  - Virtual Graphics Driver
  - Virtual Disk Driver (SCSI)
  - Virtual Network Controller
  - Seamless mouse integration
  - Shared host file exchange
  - Hot-plugging of virtual devices
  - Time synchronization
- Improves overall performance much
- Available for Windows, Linux, Solaris, FreeBSD, Netware

Xen or the Art of Para-Virtualization

- Xen is an open source project
- Commercial support and packaging available
- Introduced para-virtualization into x86 virtualization
- Virtualization gap is bridged *by patching the source* of the guest operating system to avoid critical instructions
- Xen itself (often called Xen HV) is a tiny layer below a host OS
- Drivers from a privileged guest (Dom0) are used
- Dom0 is used to control hypervisor operation
- Xen HV needs to be booted first
- loads privileged guest domain as a module
- supports Linux, *BSD, Solaris (for Dom0 and PV)

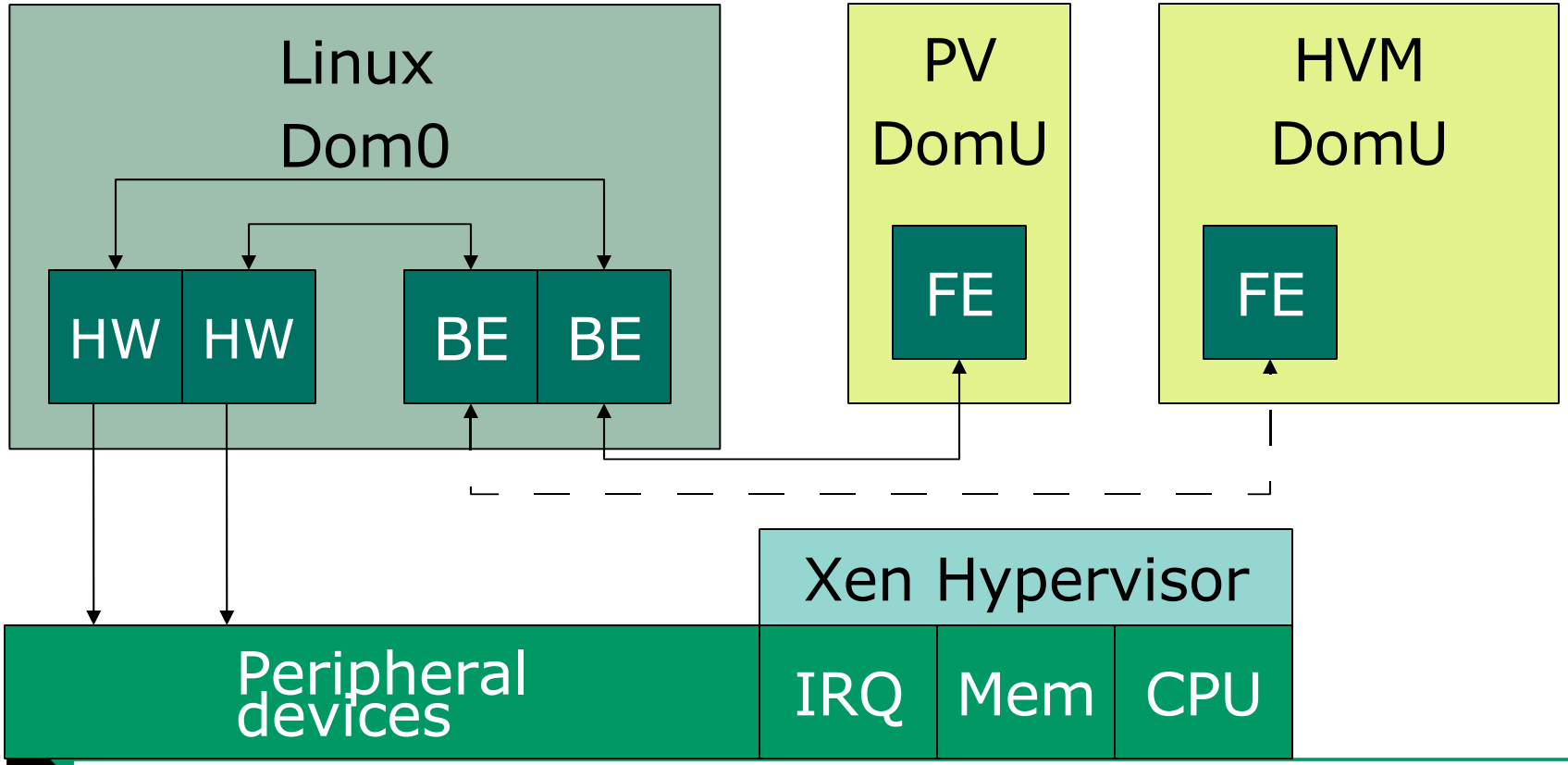
Xen Virtualization Architecture

- Xen uses existing Operating System (Dom0) to provide drivers
- HV management tools are Dom0 userspace applications



Paravirtualized drivers in Xen

- Xen uses a backend/frontend architecture
- Driver stub in guest (frontend) forwards I/O request to Dom0
- Dom0 queues the request and redirects them to the hardware



Hardware assisted virtualization: Xen & KVM

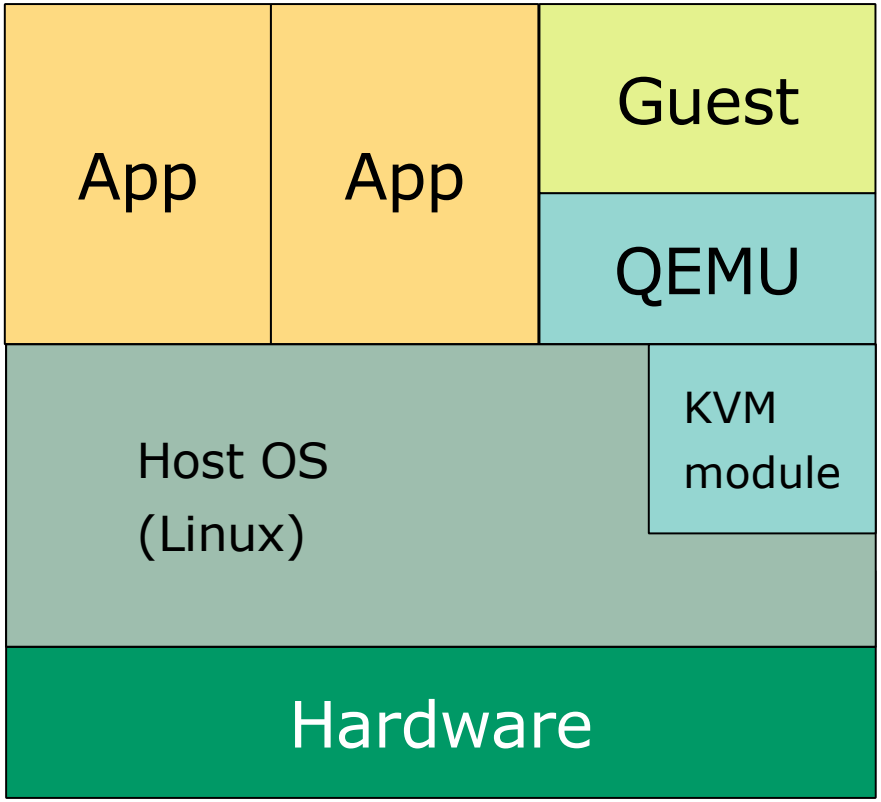
- Xen is performing, but requires modified Operating Systems
- Recent hardware features allow **unmodified** systems to run
- Virtualization gap is bridged *by hardware*
- Hypervisor sets up the stage, installs handlers and lets the system run
- Guests run natively until specific conditions occur
- Processor interrupts the guest and hands control to the HV
- Exception condition include (details later):
  - Critical instructions (popf)
  - I/O accesses
  - Hardware interrupts or exceptions
- Hypervisor can be very simple => less error prone

HVM domains in Xen

- Hardware virtualized domains since Xen 3.0
- Only DomUs supported
- Integrated in Xen, differences in config file
- Using QEMU driver architecture (called ioemu in Xen)
- Xen provides BIOS code (taken from the Bochs project)
- Guest boots up in real mode and loads it's bootloader
- Host CPU is passed through (although modified)
- Rest of hardware is emulated (QEMU)
- Screen output can be displayed directly or via VNC

The Kernel Virtual Machine (KVM)

- Quite recent addition to the virtualization arena
- Open Source project sponsored by Qumranet
- Included in Linux since 2.6.20
- Implemented as a "Virtualization Driver"
- Requires hardware supported virtualization
- easy structure and small footprint
- uses Linux services (scheduler, memory)
- not very stable yet, but advancing rapidly



Comparison of Virtualization technologies

- VMware Server presents a mature implementation of binary translation
- Xen presents a usable paravirtualization solution
- KVM as an example of a recent implementation of hardware virtualization

	VMware Server	Xen PV	KVM
Software complexity	High	Moderate	**Low**
Age	9 years	5 years	2 years
Installation effort	Fair	High	**Low**
I/O speed	Fast	**Very fast**	Slow
Maturity state	**Mature**	Stable	Unstable
Requires HW virtualization	Sometimes	**No**	Yes
Supports HW virtualization	Not really	**(Yes)**	**Yes**

* not on Intel hardware for 64bit guests

** only on Intel hardware for 64bit guests

AMD-V™ Technology

AMD-V: Hardware virtualization

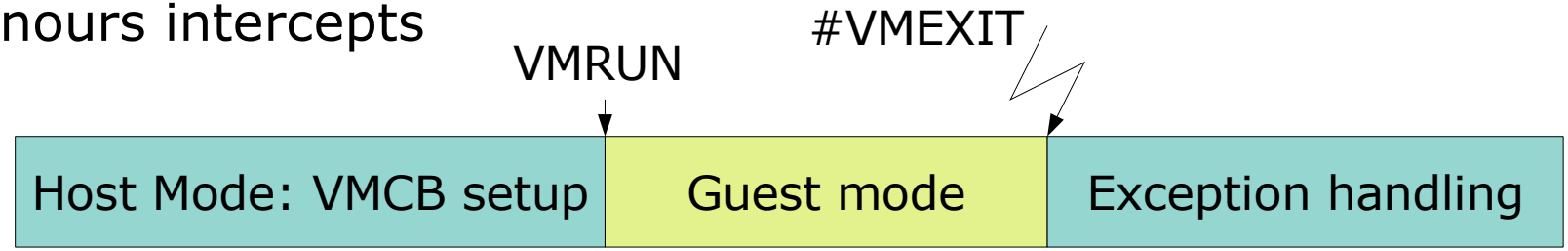
- Solves the remaining issues in x86 architecture
- Is implemented in AMD's K8 Rev. F silicon and higher:
  - AMD Opteron's and AMD Athlon64's with DDR2 memory interface (AM2 or L1 sockets)
  - All AMD Turion 64 X2 notebook chips
  - Disabled in AMD Sempron chips

Introduces new processor mode: **Guest mode**

- All instructions in guest are executed conditionally
- Illegal instructions or events causing an exit to **Host mode**

AMD-V scheme of work

- More implementation details in AMD manual vol 2. sect. 15
- Central structure is the **V**irtual **M**achine **C**ontrol **B**lock
- VMCB contains
  - interception vector bitmask describing events
  - state save area holding the guest's state
- VMCB is placed somewhere in memory
- Hypervisor fills the VMCB state save area with initial values
- Hypervisor sets bits for critical events
- Calls new command with physical address of the VMCB
- Processor switches now to „managed" guest mode and honours intercepts



The Virtual Machine Control Block

- Bits in Control Area control interception of:
  - read/writes of Control Registers (CR0-CR15)
  - read/writes of Debug Registers (DR0-DR15)
  - exceptions 0-31
  - various system instructions (read/writes to gdtr/ldtr/idtr)
  - critical user instructions (pushf, popf, cpuid, rdtsc, ...)
  - virtualization instructions
  - accesses to I/O ports and MSRs
  - address of I/O and MSR permission bitmap
  - page table handling
  - interrupts handling
  - exit codes
  - nested paging
- State Save Area contains:
  - segment and descriptor table registers
  - control registers
  - flags, stack pointer, instruction pointer
  - additional system registers

New AMD-V instructions

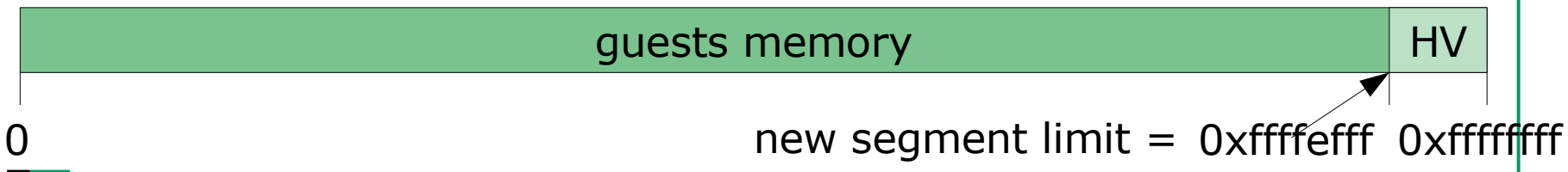
- **VMRUN** instruction takes physical address of the VMCB
- Processor restores state from the VMCB and runs in guest mode
- intercept event causes the processor to exit guest mode
- this is called a „**#VMEXIT**“ (implicit instruction)
- instruction flow continues behind the VMRUN instruction
- only some processor state is saved in the VMCB:
- some other data can be saved with **VMSAVE**
- restoring is done with **VMLOAD**
- guest mode can be left explicitly with **VMMCALL**

Additional Hardware Tricks

- Segmentation Limits Support (Rev. D/E)
- Paged real mode (Rev. F / AMD-V)
- Device Exclusion Vector (DEV) (Rev. F / AMD-V)
- Tagged TLBs / Address Space Identifiers (ASIDs) (Rev. F / AMD-V)
- Nested paging (Barcelona & Co.)
- IOMMU (upcoming)

Segmentation Limits

- Segmentation not effective anymore in 64bit mode
- Although almost no operating system used it, VMware did
- Hypervisor code must be in the guest address space
- Popek/Goldberg requires HV code not to be visible
- VMware used segmentation to hide its code
- Allows executing without reading (not possible with MMU)
- Segments are tagged with a privilege level (rings)
- HV code is on top of virtual memory (4GB)
- segment limit is put just below this code
- AMD re-introduced segment *limits* in Rev. D/E of AMD64
- allows fast virtualization of 64bit guests in VMware



Paged Real Mode

- Problem: No access restrictions in real mode
- Hypervisor cannot intercept critical memory accesses (VGA memory)
- Modern operating systems don't use real mode, but:
  - all of them come up in real mode (bootloaders!)
- One could ignore real mode, but:
- AMD chose to introduce *paging* in real mode
- Hypervisor sets up page tables
- Processor will issue page faults when accessing critical memory
- Hypervisor can trap and emulate access
- Saves a lot of emulation code
- Xen: Emulation is only partial on Intel chips

Device Exclusion Vector (DEV)

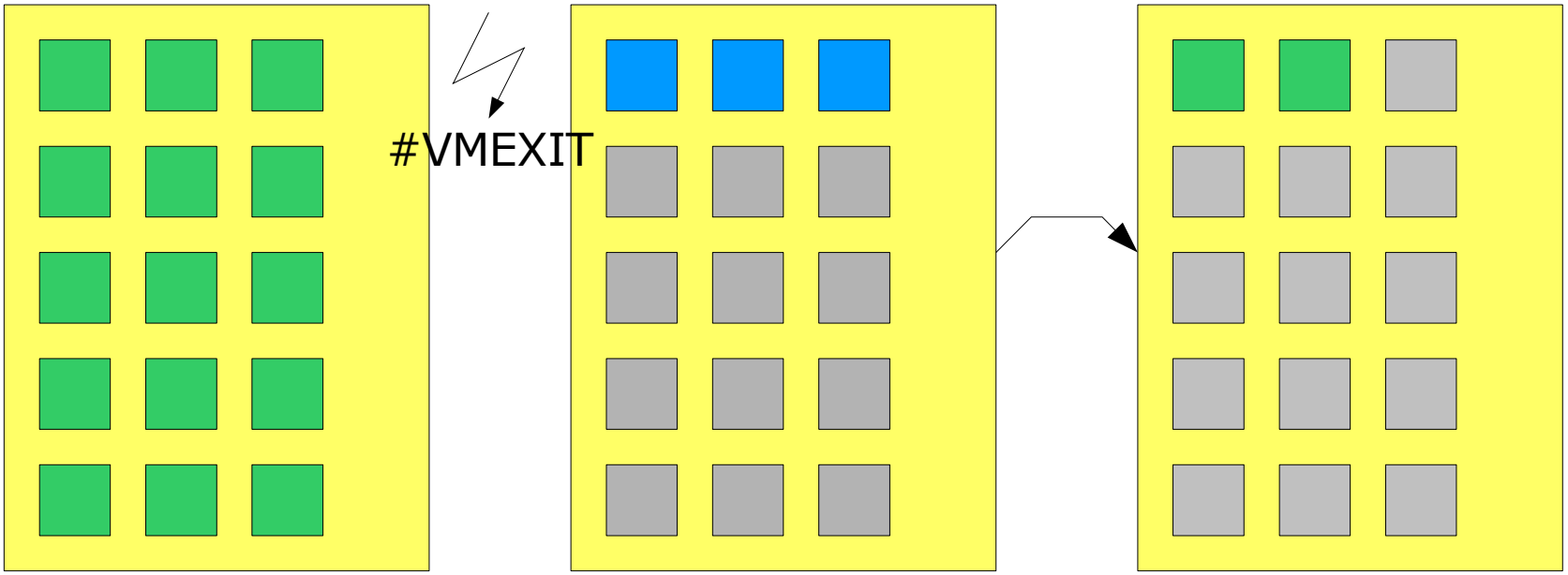
- Passing real hardware to guests is desirable
- I/O and MMIO can be trapped or redirected, but:
- DMA accesses impose a challenge:
  - guest knows only about *it's own* pseudo-physical addresses
  - Hardware devices can access *any* host-physical addresses
- "enlighted guests" can translate pseudo to host, but errors or malicious code breaks isolation
- DEV solves this:
  - protection domains are introduced
  - each domain is assigned one bit vector defining access to memory on a per-page base
  - protection domains are assigned to Hypertransport devices
  - results are cached in the northbridge

Details in chapter 15.23 (External access protection) of AMD manual vol. 2

Tagged TLB

- **T**ranslation **L**ookaside **B**uffer accelerates translation of virtual memory
- Page table walks are cached in the TLBs
- changing of virtual memory arrangement (task switches) requires flushing the TLB
- Re-walking the page table can be costly, TLB hit is crucial for performance
- tagging each TLB entry with an Address Space Identifier (ASID) can avoid unnecessary flushes
- TLB entries stay with their guests
- no need to flush all ASIDs on guest switches
- Current implementation in Xen gives 11% improvement
- future processors will introduce “Flush-by-ASID”

TLB usage illustrated

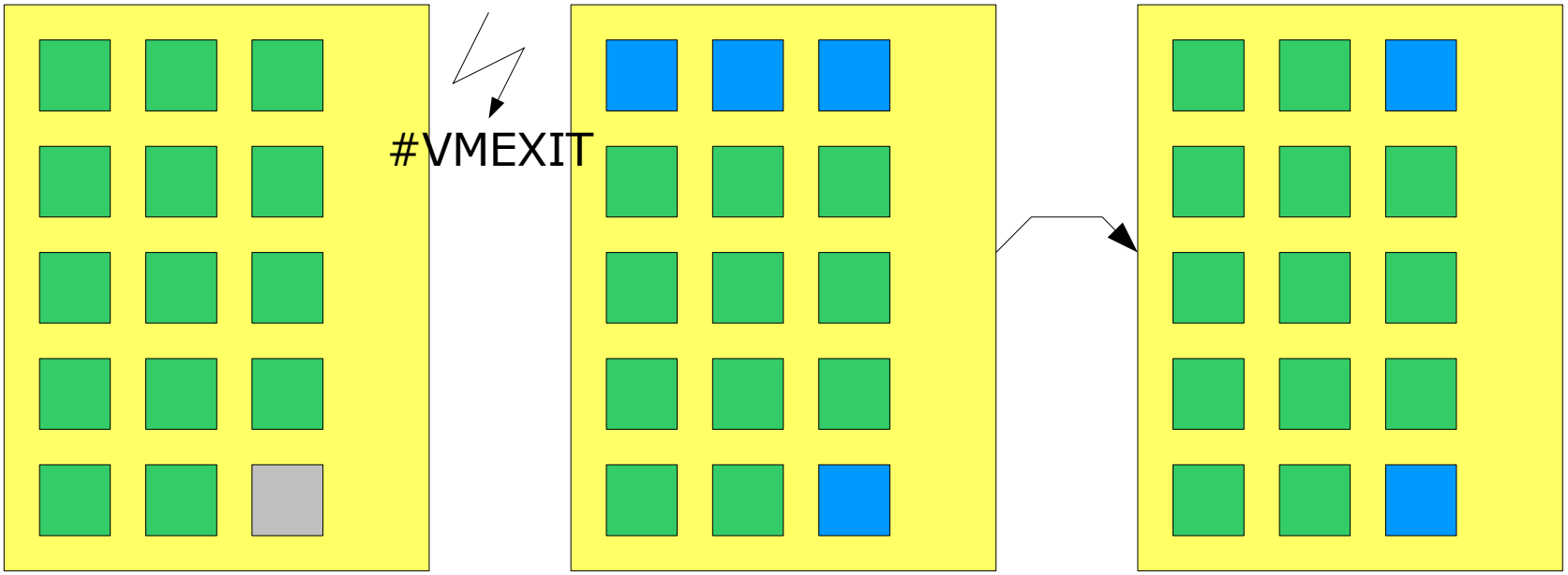


Used by guest

Used by hypervisor

unused

Tagged TLB operation illustrated



Used by guest

Used by hypervisor

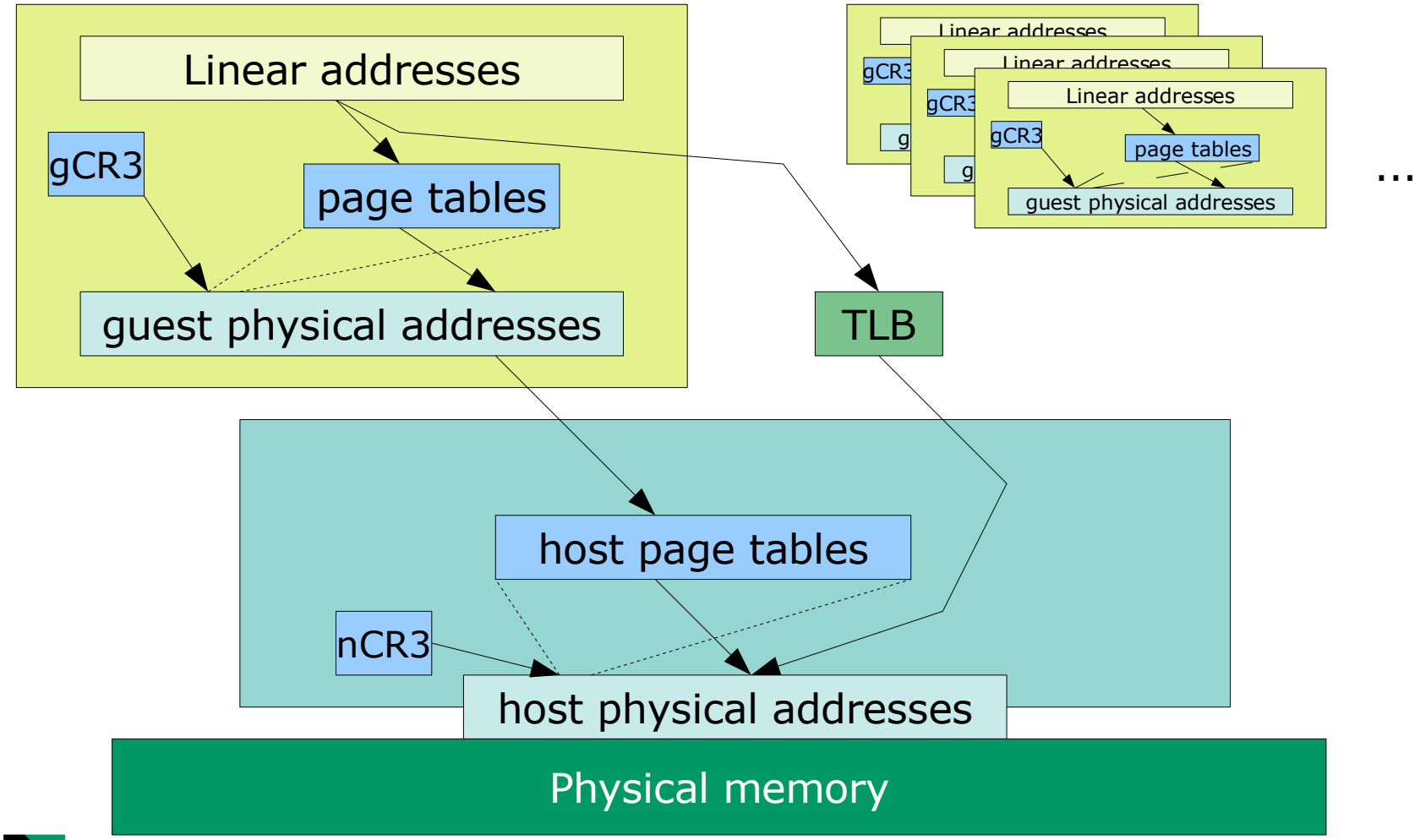
unused

Nested Paging (RVI)

- Hardware did not virtualize Memory management
- Virtual memory management must be emulated by the hypervisor
- Shadow page tables hold real guest entries
- Any access to the page tables must be trapped and emulated
- Barcelona introduces ***R****apid* ***V****irtualization* ***I****ndexing*
- each guest has it's own CR3 register (gCR3)
- Processor does guest page table walk **and …**
- appends host page table walk automatically (nCR3)
- result is cached in TLB
- guest can manage its own page tables undisturbed
- already implemented in Xen: 15 – 20% improvement

Nested Paging operation

two level address translation: GV -> GP -> HP



IOMMU

- Access to hardware is usually done by the hypervisor
- Direct access from the guest would improve performance
- DEV requires the guest driver to know the real addresses
- **IOMMU** allows *remapping* of DMA addresses
- comparable to virtual memory for devices
- Allows safe access from within guests
- guest drivers can use their own view of memory
- will be implemented in host/bus bridges (chipset)
- will allow virtualizable devices
- Xen patches already upstream

Virtualization in use

QEMU in real life: Installation

- qemu (still) requires GCC version 3, so use distribution packages (or install gcc3)
- ordinary user space application, not even root-only
- contains binaries for different emulation targets:
  - x86_32, x86_64, arm, sparc(64), mips(64), powerpc(64)...
- qemu-system-<arch> for full system emulation
- qemu-<arch> for Linux user land emulation
- installs various BIOS binaries
- contains qemu-img for handling image files
- use (optional) kernel module: kqemu
- **`$ modprobe kqemu`**
- Make sure /dev/kqemu is existent and writable

QEMU in real life: Running guest

- qemu-system-x86_64

  -hda /path/to/hdd/image

  -m 192 (Megabytes of memory)

  -k de (for German keyboards)

  -cdrom /path/to/image.iso

  -boot d (boot from CD-ROM)

  -smp 2 (emulate dual processor machine)

  -usb (enable USB emulation)

QEMU in real life: tips

- Remote display via VNC
  `-vnc :0`
- Qemu monitor: Alt+1 in guest window
  - Check guest
  - Save, restore, suspend
  - inserting/removing of CD-ROMs (.iso files)
- Direct Linux kernel loading:
  `-kernel vmlinuz-x.y.z -append 'ro root=/dev/hda1'`
- USB tablet emulation
  - Absolute reporting of position helps avoid lags
  - “Grabless” mouse feature

  `-usb -usbdevice tablet`

Xen in real life: Installation

- Avoid building Xen, but use distribution packages
- Usually packages for
  - Xen hypervisor (single ELF file, looks like a kernel)
  - PV kernel (for Dom0, vmlinuz-2.6.18.8-xen{,0,U})
  - Xen tools (mostly Python scripts for controlling guests)
  - sometimes: ioemu (support files for hardware virtualization)
- enter Xen into grub (which is required):

```
kernel /boot/xen.gz
module /boot/vmlinuz-2.6.18.8-xen0 root=/dev/sda5...
initrd /boot/initrd.gz
```

- reboot! ;-(

Xen in real life: Configuration files

- Xen uses Python configuration files
- cp /etc/xen/xmexample . (for PV DomUs)
- adjust following options:

```
kernel=/some/path/vmlinuz-2.6.18.8-xen{,U}
memory=<Megabytes>
name=<Unique name>
disk=['phy:hda1,hda1,w', ',hdc:cdrom,r']      or
disk=['file:/data/slamd64-12.hdd.img,hda,w'] or
disk=['tap:qcow:/data/slamd64_'+str(num)+'.qcow
  ,hda,w']
```

Xen in real life: Starting guests

- Using xm tool (Python script)
  `xm create` <config.file>
  `xm list` (showing all domains)
  `xm vcpu-list`, **`xm vcpu-pin`** (changing CPU affinity)
  `xm pause` <domid>, **`xm unpause`**
  `xm migrate` (to another host)
  `xm destroy`
  `xm top` (for monitoring guests)
  `xm console` (for PV guests or serial console on HVM)
  `xm info` (for hardware and hypervisor info)

Xen in real life: neat tricks

- Ballooning: Change memory during runtime

  `$ xm mem-set` <domid> <memory>
- Live Migration: Move guest to another host

  `$ xm migrate -l` <domid> <hostname>

KVM in real life: Installation

- KVM is already in Linux, but better use KVM packaged modules (or distribution versions)
- KVM consists of:
  - kernel module(s): virtualization drivers (for AMD and Intel)
  - library for accessing these drivers (libkvm)
  - patched qemu (efforts to merge are underway)
- no reboot necessary: **`modprobe kvm-amd`**

KVM in real life: Running guests

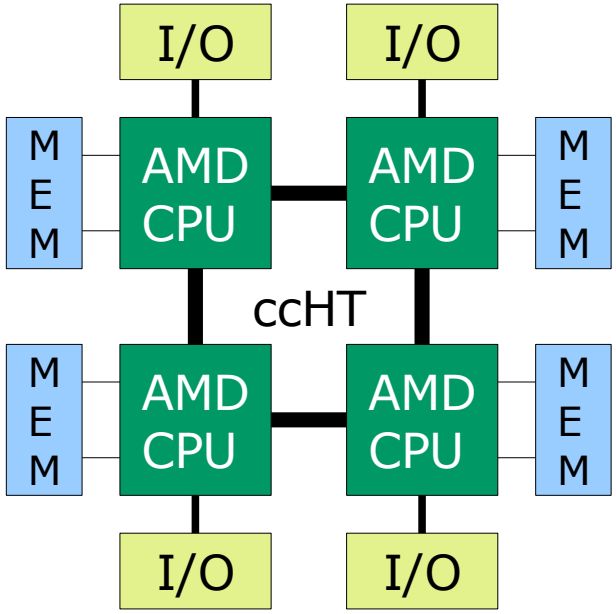
- launch patched qemu with usual options:

```
$ qemu-system-x86_64 -hda disk.img -m 512
```

- every VCPU is a thread
- memory is allocated from userland (can be swapped)
- Make sure /dev/kvm is there and is accessible
  - Watch out for a message: uses qemu emulation otherwise!

Multi processor considerations

- Multi-core machines are perfect for virtualization
- Virtual machines can be load balanced across the cores
  - multiple guests can be scheduled on one core
  - guests can be restricted to certain cores
- Exclusive assignment of guests to cores is possible
- SMP guests are possible
- more cores (or more processors) offer more capacity for virtual machines => scales naturally
- I/O and RAM bandwidth gets important
- AMD Opteron architecture's DCA is superior in scaling



Migration

- Virtual machines consist of configuration options and disk images
- Both can be simply copied between machines
- Even easier with network storage (SAN or NAS)
- Virtual hardware is identical in the same hypervisor
- Migration scenario:
  - shutdown virtual machine on old server
  - copy image over to new machine
  - start up copied image on new server
- Guest network configuration should be flexible (DHCP)

Live migration

- Downtimes for migration not always desirable
- Xen (and others) offer **live migration**
- Image will be copied while the guest is running:
  - Source guest image will be marked as read-only
  - image copying starts
  - any writes at the source will be catched and transmitted
  - this is done with hard disk content and RAM content as well
  - if all data is transmitted, source guest will be suspended
  - at the same time target guest will be resumed
  - downtime is in millisecond range
  - TCP connections stay alive!
- Hardware upgrade or repair scenario:
  - live-migrate all machines to a temporary host
  - shut down original host, repair or upgrade, then restart
  - migrate virtual machines back

Real time challenges

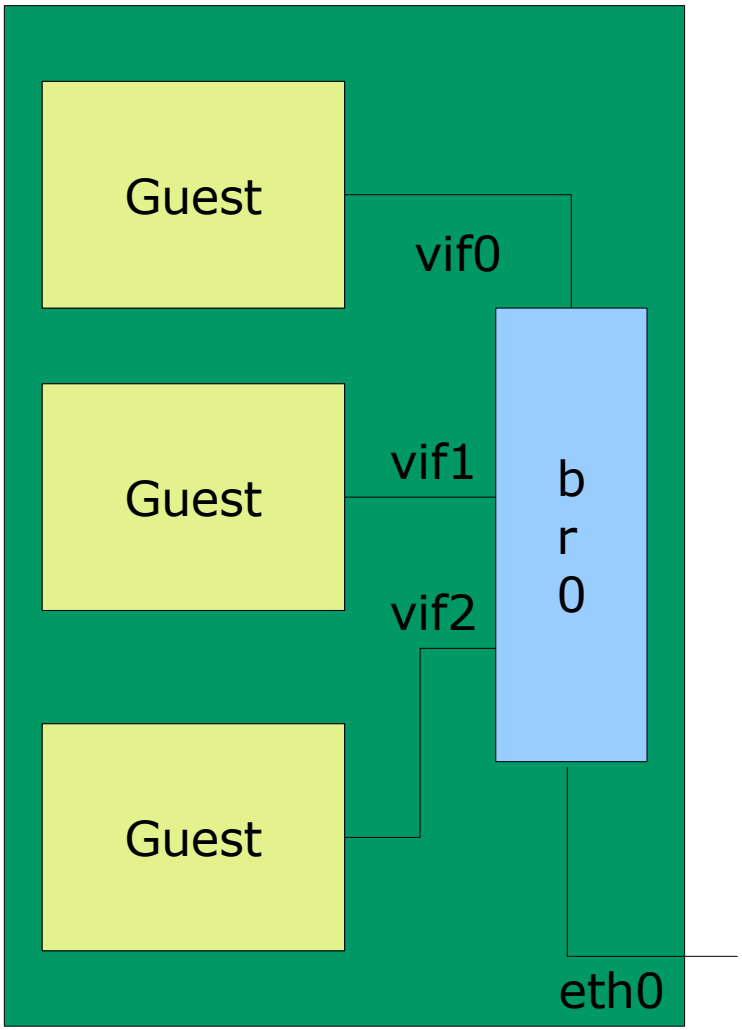
- Virtualization imposes challenges on timing
- Hypervisor catches at least timer interrupts and cares about scheduling
- guest time would be halted if HV takes over
- TSC is also virtualized!
- VMCB contains TSC offset field to compensate loss of time
- trade-off between cycle-correctness and wall clock time deviation
- Real time kernels do not simply work within hardware virtualized domains
- Real time qualities have to be integrated into the *hypervisor*
- SYSGO and Virtual Logix work on real time virtualization solutions
- Issues are known and will be solved!

High availability

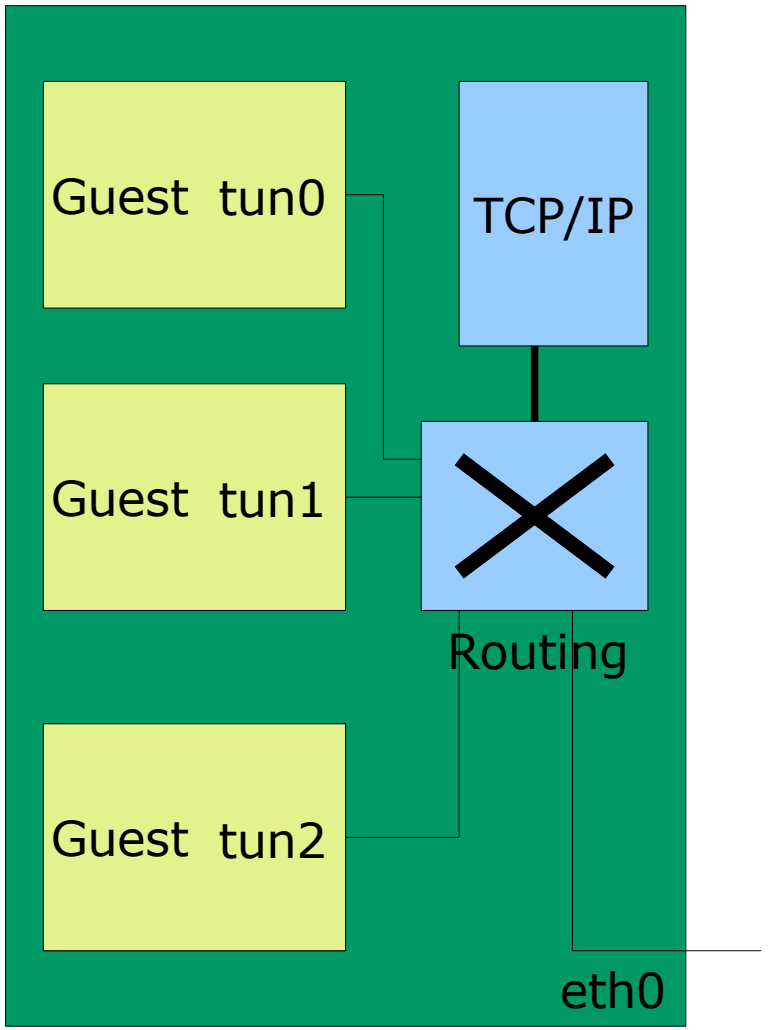
- Live migration allows fast switching between hosts
- Feature can be used to ensure high availability:
  - constantly syncing status from one host to another
  - guest on fail-over machine is suspended
  - will be resumed when needed

- **M**achine **C**heck **A**rchitecture (MCA) reports (potential) hardware failures on x86
- Correctable errors get reported => flaky machines can be detected
- Recent efforts by AMD implement MCA in Xen hypervisor
- Adjustable policy reports exceptions to either hypervisor, managing Dom0 or DomU guest directly

Possible Xen network topologies



Virtual switch (xen-br0)



Virtual NICs (routing)

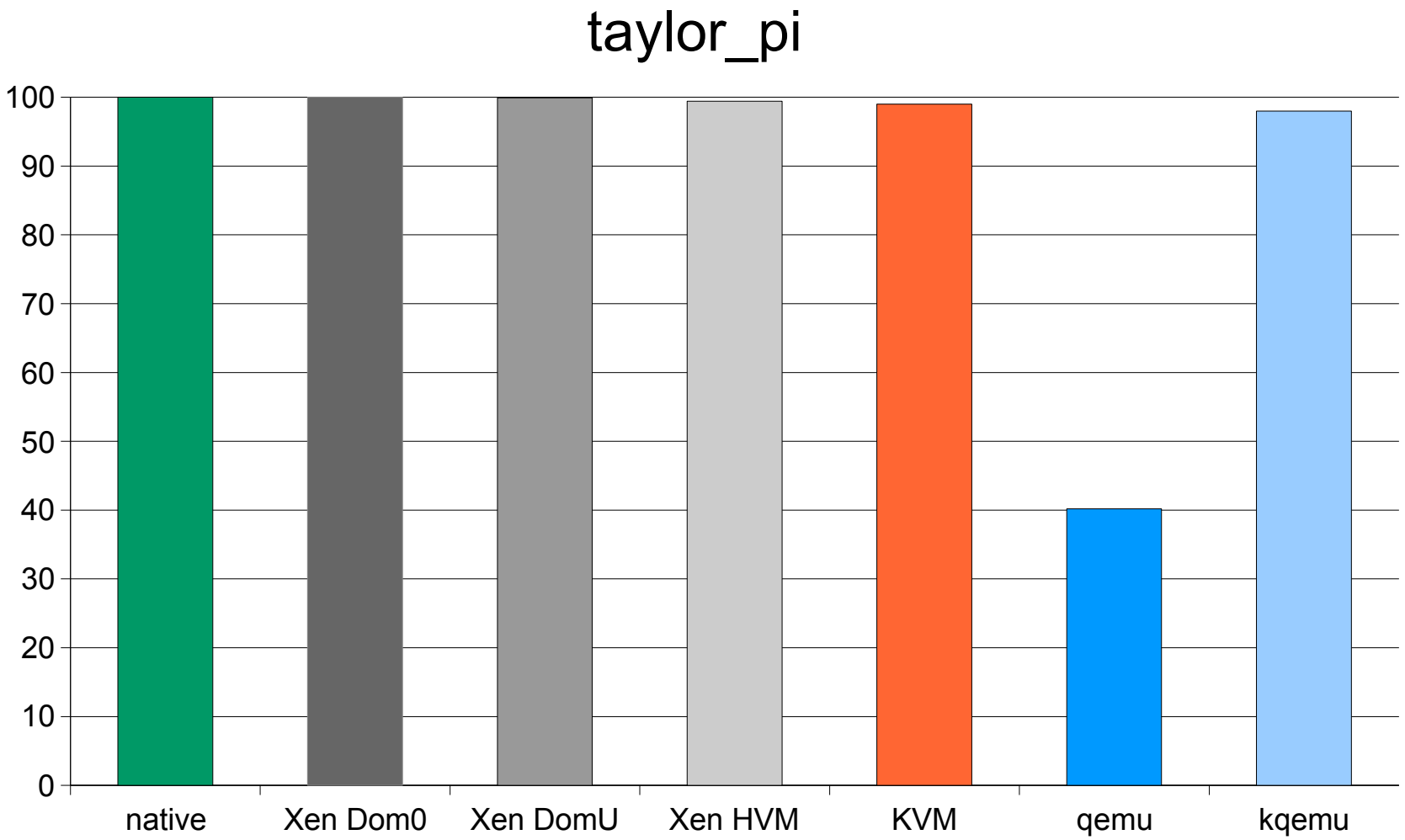
Virtualization demo

Benchmarks

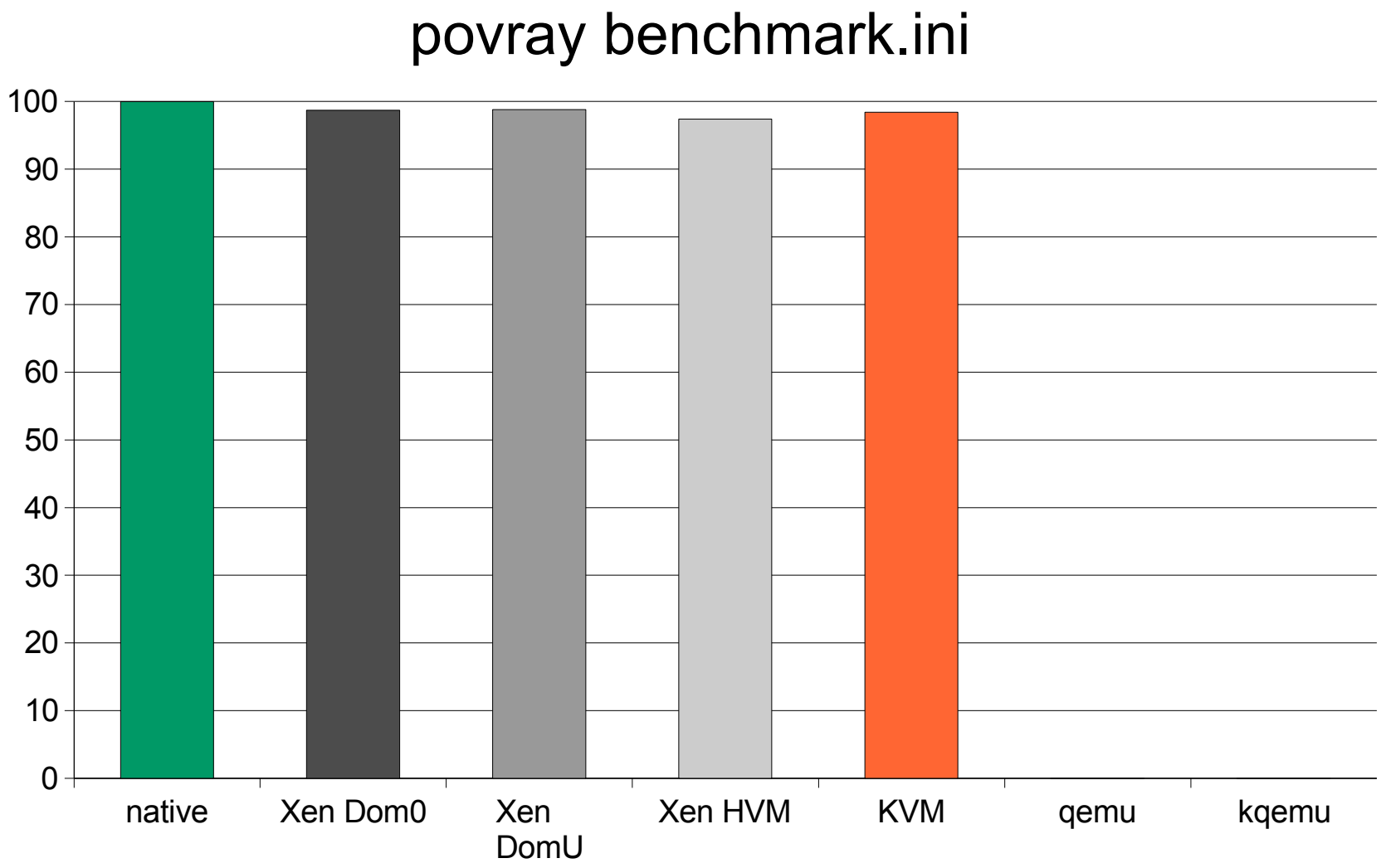
Benchmarking virtual machines

- Be careful:
  - Time in guests is mostly unstable
  - I/O or VMEXITs can make things a lot slower
  - Keep all conditions the same (HV, host & guest kernel, ...)
- Kind of benchmarks:
  - CPU: taylor_pi
  - CPU+mem: povray benchmark.ini
  - CPU+mem+I/O: kernbench (-s -H -O -M)
- Benchmarks relative to native
- Used machine: Tyan server, 4*AMD Opteron 8220, 16GB RAM
- SW: SLAMD64 12.0, KVM-69, qemu 0.9.0, Xen unstable, Linux 2.6.25.3 (2.6.18.8 in Xen)

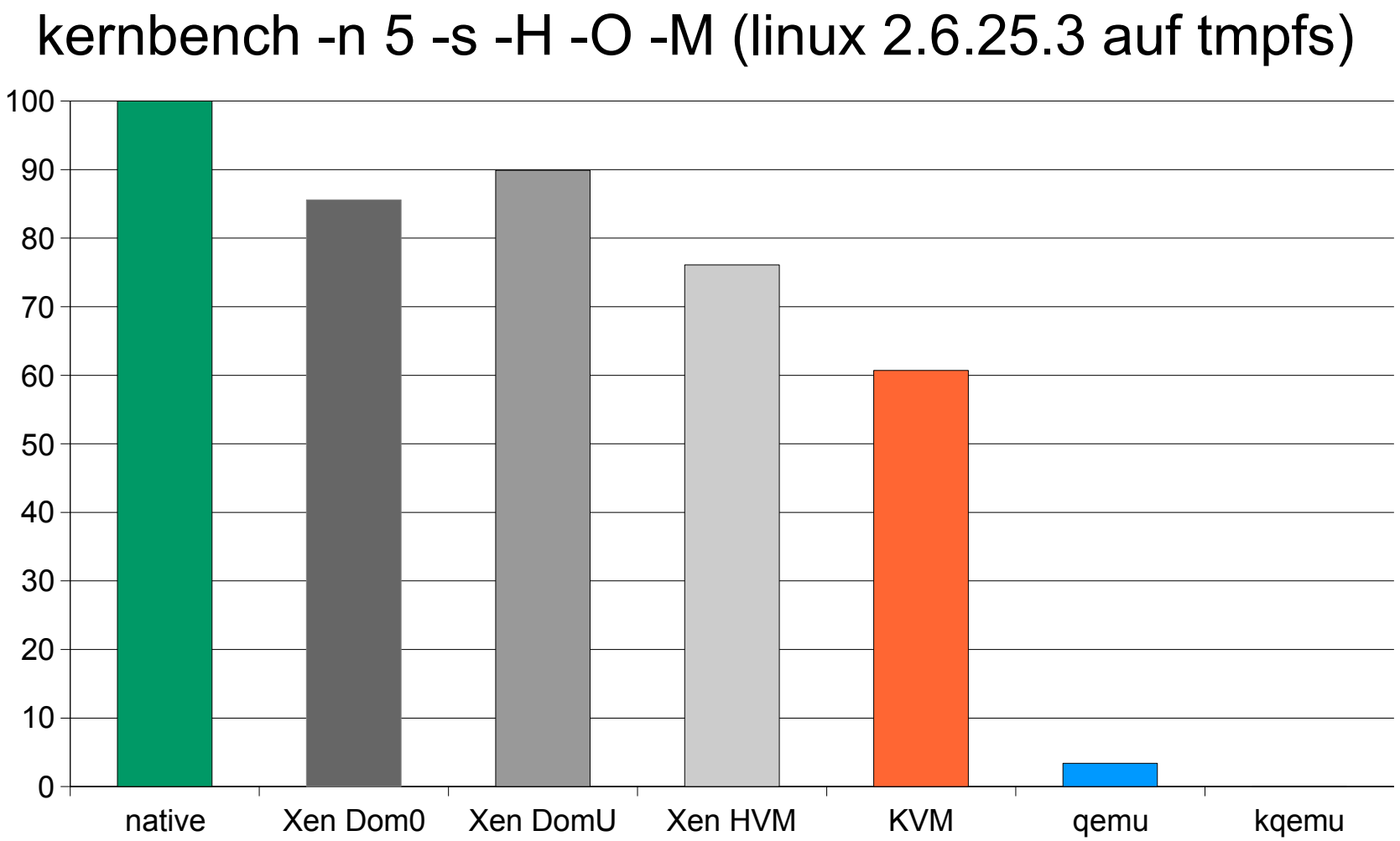
Pure CPU load: taylor_pi



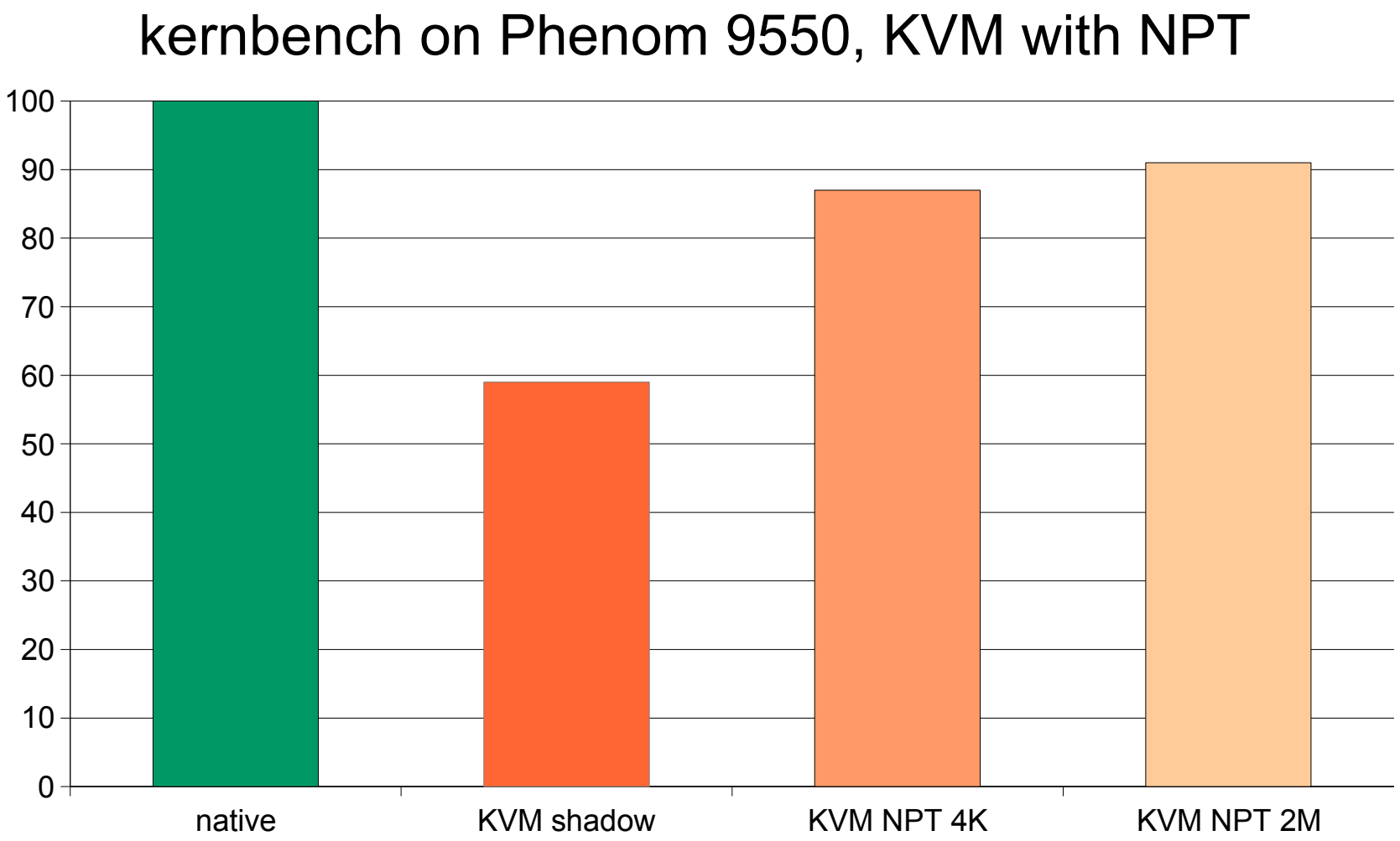
CPU + memory: POVRay



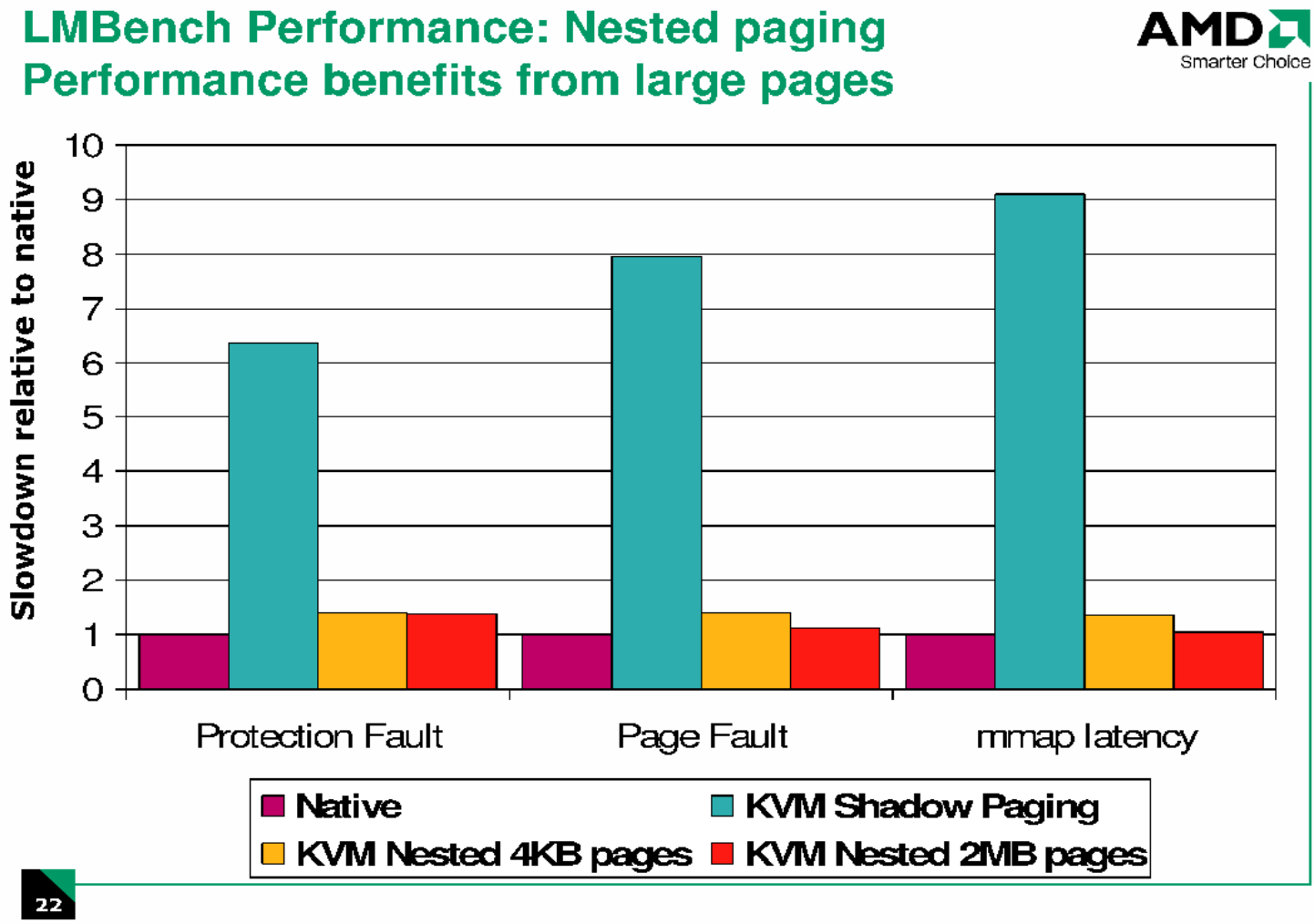
CPU + memory + I/O: kernbench



Nested paging performance: kernbench



Nested Paging performance



Virtualization outlook

Future Direction of Virtualization

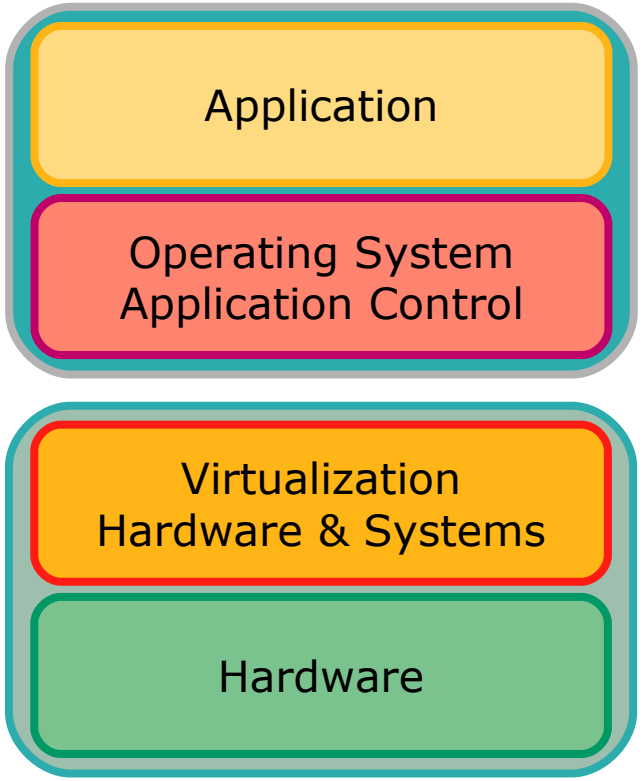
- New HW technologies provide explicit support
  - Increase CPU assistance
  - I/O MMU's and virtualization-aware I/O devices
  - reduce *world switch times* (by improved chip design)

- Operating systems become virtualization-aware
  - OSes provide information to the hypervisor (PV)
  - Skip expensive-to-virtualize operations
  - reduce *number of world switches*
  - paravirtualized drivers become available (e.g. from SuSE)
  - Hypervisors get more and more integrated (RHEL5, SLES10, Windows Server 2008 or KVM approach)

Hybrid hypervisor

- Idea: combine advantages of para-virtualization with unmodified guests
- Way to go: avoid costly world switches
- Solution: insert hypervisor code into guests memory
- Injected code can check exit conditions within the guest
- Example: old Windows versions utilize TPR very often (600,000 times per second!)
- TPR is accessed via MMIO, will cause #VMEXIT
- round trip can be up to 2500 cycles
- catching access without exit is crucial for performance

Virtualized Appliances



Operating system will be coupled with Application.

Virtualization layer will be coupled with HW.

Pre-configured, purpose-built virtual device
Pre-installed and pre-configured OS & application
Limited configuration/customization exposed to user
Simple installation and setup
Doesn't require dedicated machine

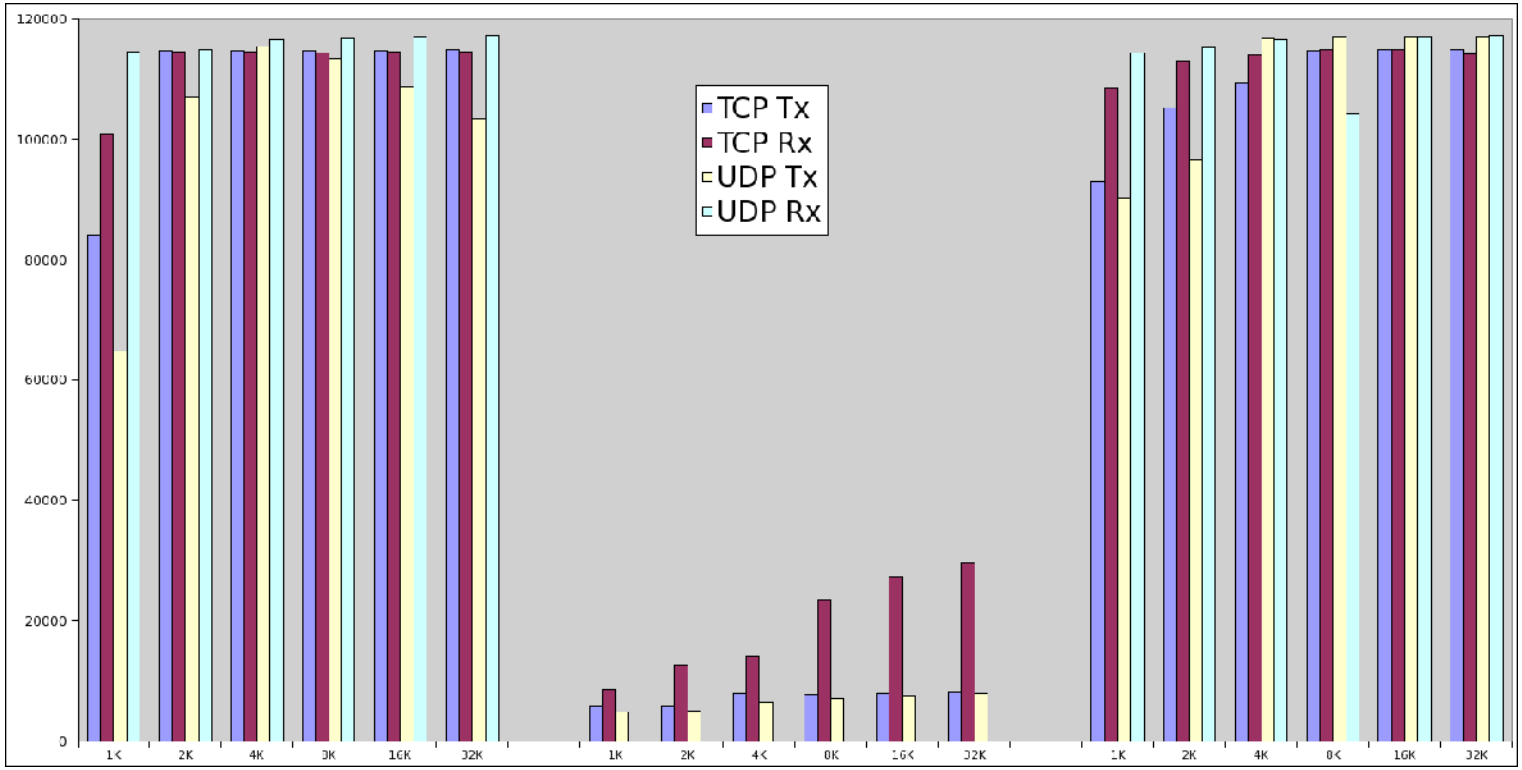
Virtualization goes mainstream

- Major Linux vendors include virtualization in their products
- Example: RHEL 5 and SLES 10 include Xen
- Use is easy:
  - use software package to install Xen support
  - reboot machine
  - choose Xen from the boot loader
  - fire up virtualization management GUI to control virtual machines
- Even community distributions include Xen
- Microsoft/Novell deal includes Xen support for Windows
- Novell ships paravirtualized drivers for Windows

Paravirtualized drivers

- Trap and emulation of I/O in hardware virtualized guests is very costly
- I/O performance (network, disk) is quite poor
- PV drivers avoid unnecessary VMEXITs
- optional package is used after installation
- PV drivers redirect high level request (read a sector, send an Ethernet frame) directly to the hypervisor
- Xen uses shared memory pages to avoid copying
- Pages with guest data are remapped to the hypervisor
- Performance is very good
- Vendors provide PV drivers even for legacy systems (old Windows versions, RHEL 3&4, SLES 9, ...)

Paravirtualized Drivers Benchmark



NetIO benchmark running under Dom0 or RHEL3 HVM-DomU (using a 2.4.21-47.EL UP kernel)

System: IBM x3455 server with 2P2C Opteron 2210, 8GB RAM, 2 Broadcom TG3 NICs, running OpenSuse 10.2 64 bit with Xen unstable (Linux 2.6.18)

Server side was Barcelona prototype (4P4C) with 8GB RAM, 2 Broadcom TG3 NICs, running Fedora 7 with 2.6.21

QEMU disk images

- Virtual harddisks are contained in files:
  - Raw images: can be sparse (UNIX-way), easy to loop-mount, copying usually blows them up
  - QCOW images: ordinary file, includes tables of used sectors, grows with usage
    - Supported by Xen, KVM, QEMU
    - Supports backing files
  - VMDK: same as QCOW, but from VMWare
- QEMU (and Xen and KVM) provide qemu-img
  - Can create image file in various formats
  - Can convert between all of them
  - Joining and splitting backing files
  - Supported formats: parallels qcow2 vvfat vpc bochs dmg cloop vmdk qcow cow host_device raw