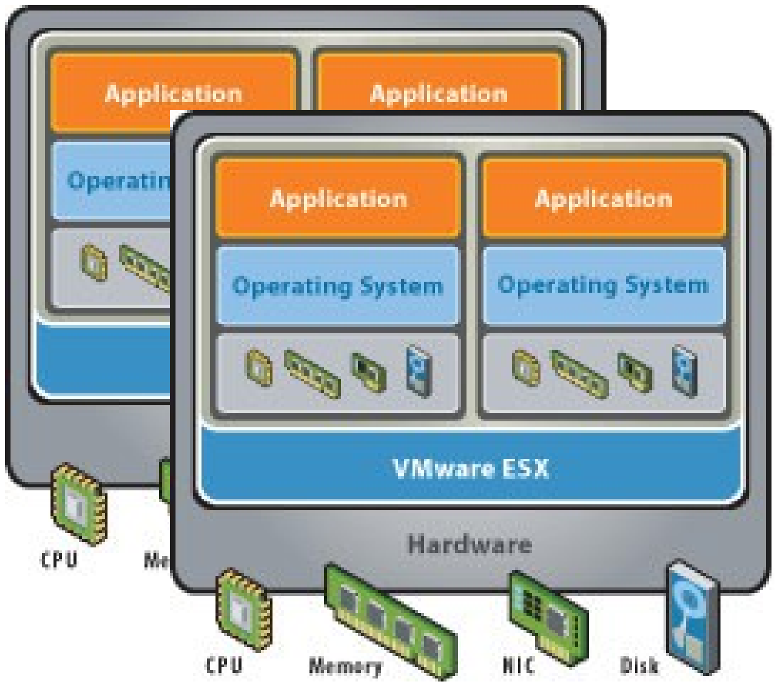
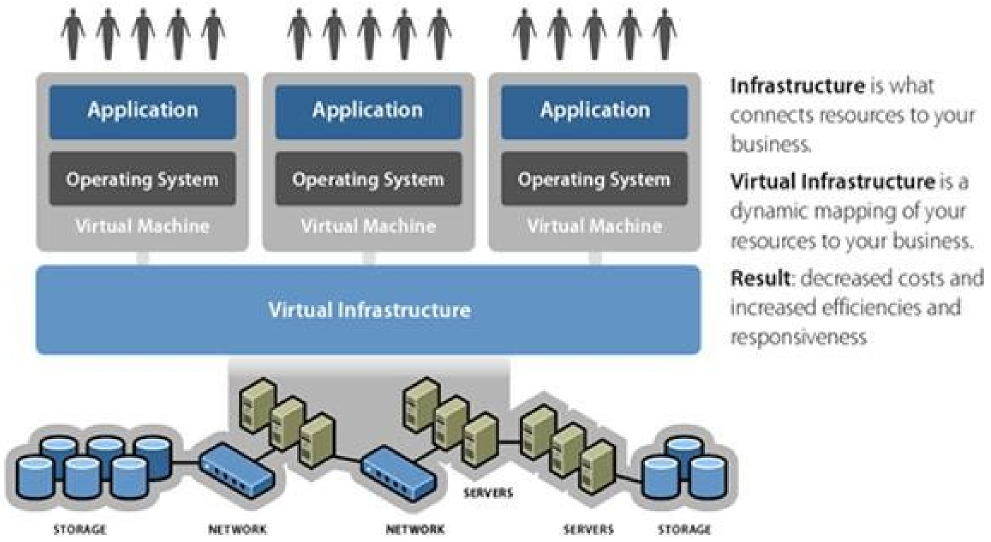
# Virtualizacija operacijskih sistemov

# Kaj je virtualizacija




Application
Operating System
Virtual Machine
Application
Operating System
Virtual Machine
Application
Operating System
Virtual Machine
Virtual Infrastructure
STORAGE
NETWORK
NETWORK
SERVERS
SERVERS
STORAGE
**Infrastructure** is what connects resources to your business.
**Virtual Infrastructure** is a dynamic mapping of your resources to your business.
**Result**: decreased costs and increased efficiencies and responsiveness

- Single OS image per machine
- Software and hardware tightly coupled
- Running multiple applications on same machine
- Underutilized resources
- Inflexible infrastructure
- Hardware-independence of operating system and applications
- Virtual machines can be provisioned to any system
- Can manage OS and applications as a single unit by encapsulating them into virtual machine

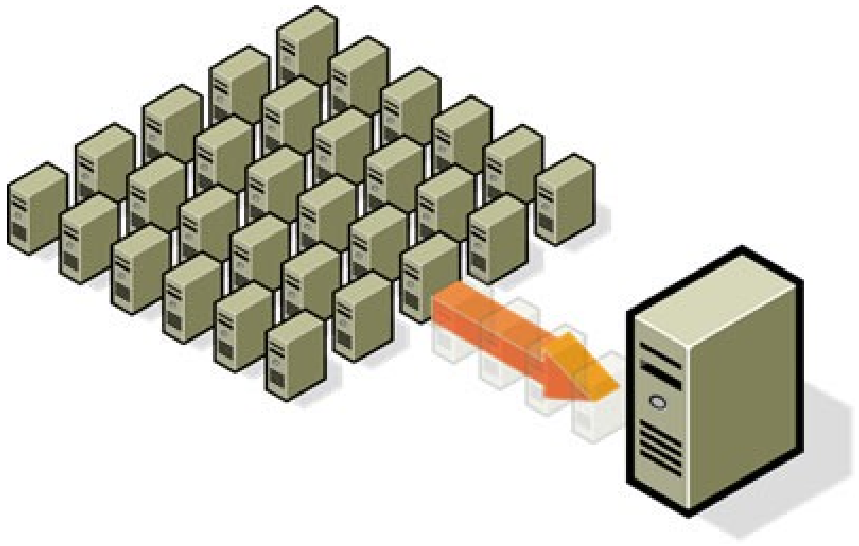
# Visoki stroški infrastrukture

- – Maintenance
- – Leases
- – Networking
- – Floor space
- – Cooling
- – Power
- – Disaster Recovery

## Heterogena okolja

## Migracija strežnikov

- Too many servers for too little work
- Aging hardware reaching end of usable life
- High infrastructure requirements
- Limited flexibility in shared environments



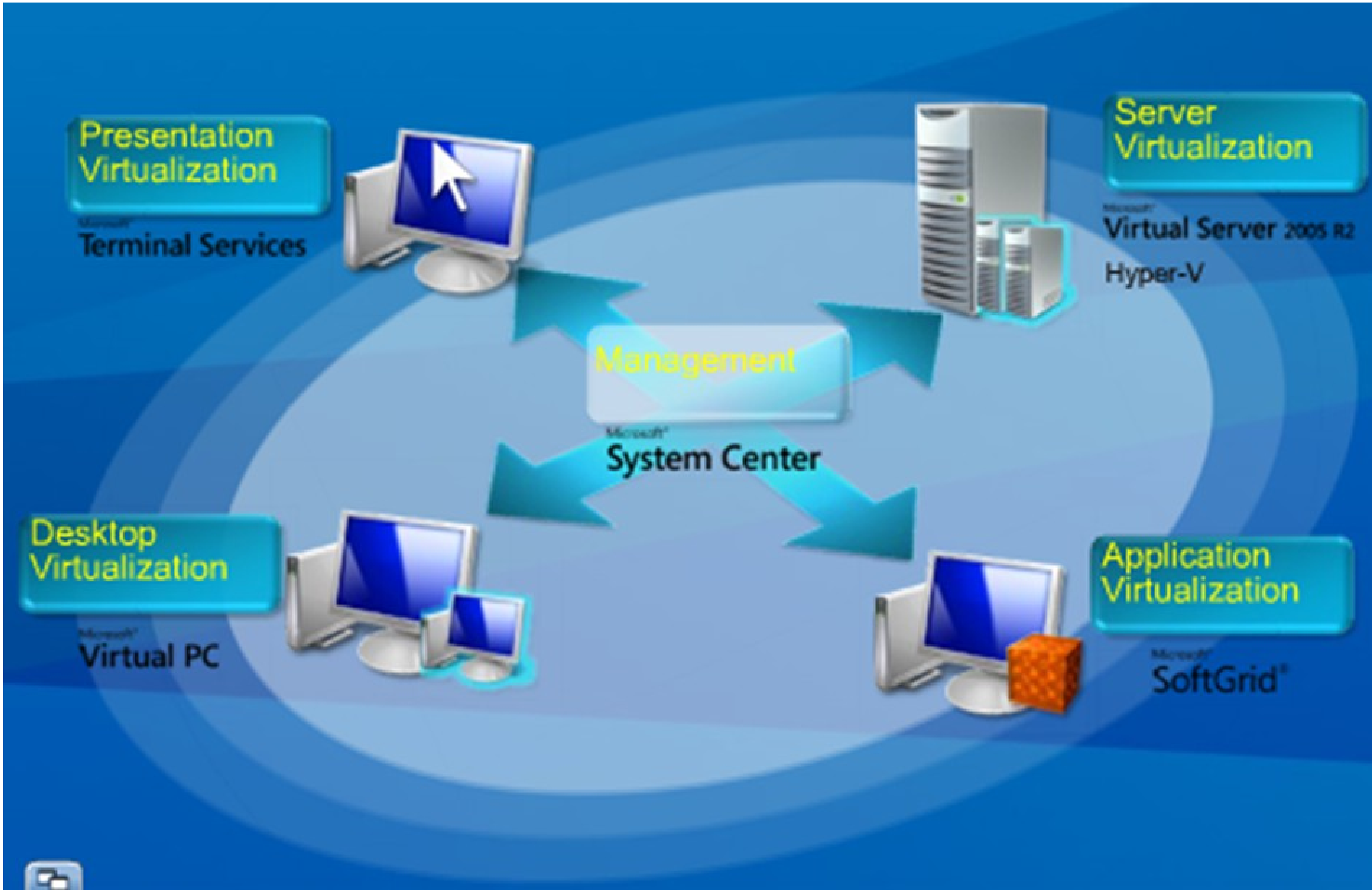
Reduce costs by consolidating services onto the fewest number of physical machines

# Virtualization is “HOT”

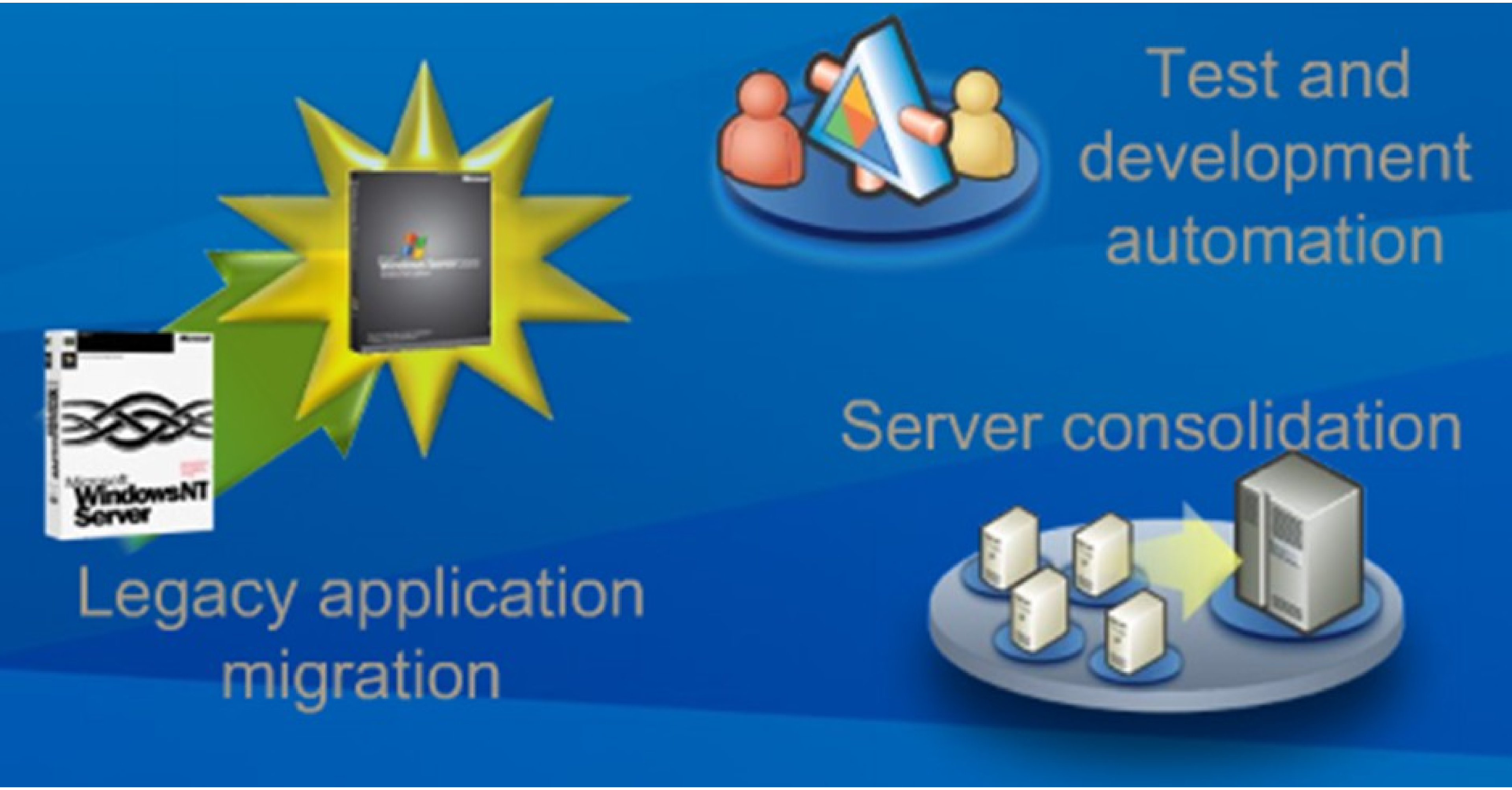
- Microsoft acquires Connectix Corp.
- EMC acquires VMware
- Veritas acquires Ejascent
- IBM, already a pioneer
- Sun working hard on it
- HP picking up

➔Virtualization is HOT!!!

# Server 2008 in tehnologije virtualizacije



# Virtualizacija in izzivi arhitekture



## Zakaj torej virtualizacija?

- Server consolidation
- Application Consolidation
- Sandboxing
- Multiple execution environments
- Virtual hardware
- Debugging
- Software migration (Mobility)
- Appliance (software)
- Testing/Quality Assurance

# Kaj je v resnici virtualizacija?

- Real vs. Virtual
  - Similar essence, effect
  - *”Formally”* different
- A framework that combines or divides [computing] resources to present a *transparent* view of one or more environments
  - Hardware/software partitioning (or aggregation)
  - Partial or complete machine simulation
  - Emulation (again, can be partial or complete)
  - Time-sharing (in fact, sharing in general)
  - In general, can be M-to-N mapping (M “real” resources, N “virtual” resources)
  - Examples: VM (M-N), Grid Computing (M-1) , Multitasking (1-N)

# Problemi implementacije virtualnega stroja

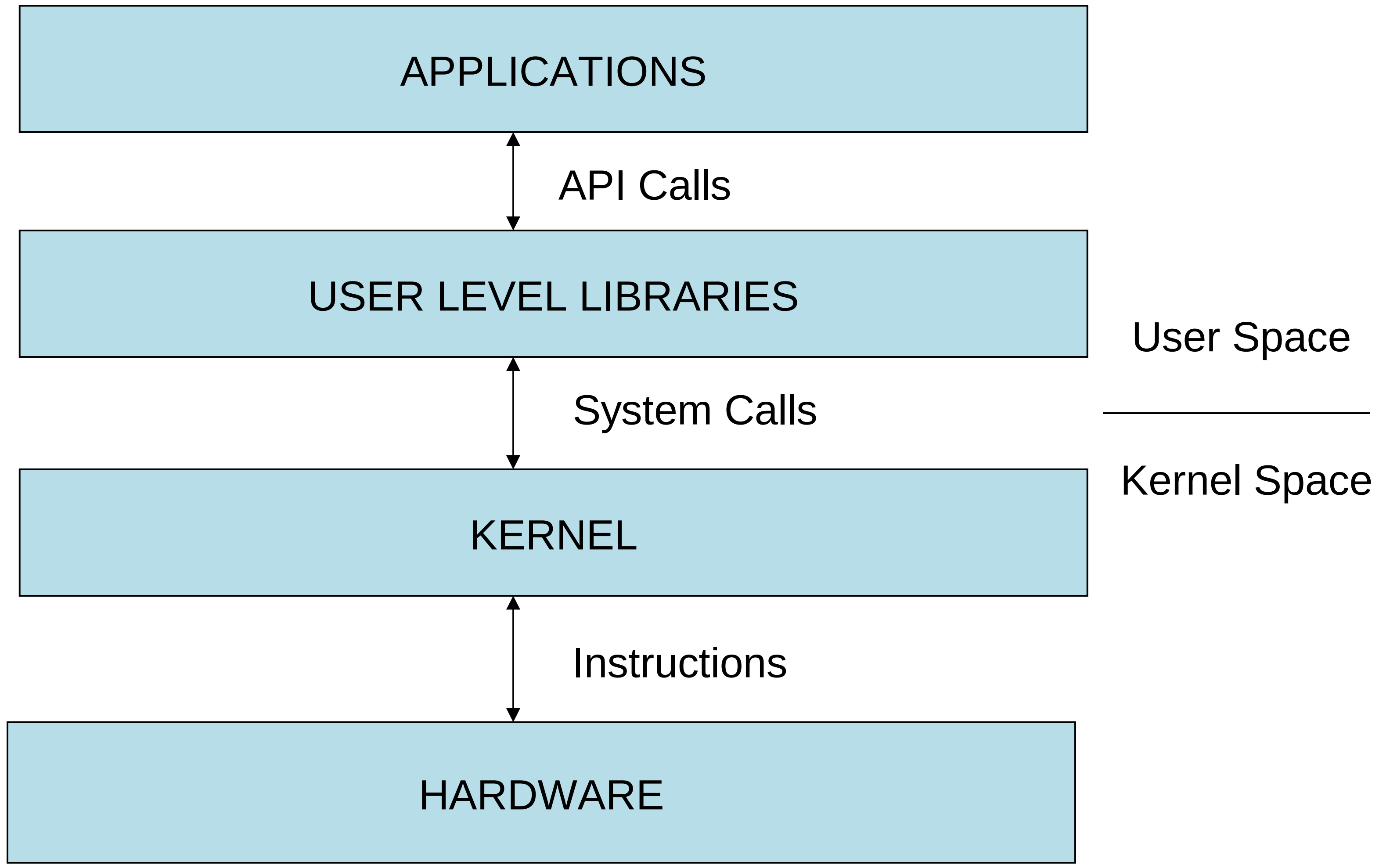
- Only one “bare” machine interface
- Virtualizable Architecture

  “A virtualizable architecture allows any instruction inspecting/modifying machine state to be trapped when executed in any but the most privileged mode”

  - Popek & Goldberg (1974)
- X86 is not virtualizable (Vanderpool??)

- Hard to optimize [from below]
  - Unused memory pages
  - Idle CPU
- Difficult to know what NOT to do
  - Example: Page faults (VMM), System Calls (OS level)

# Stroji: večplastna arhitektura



## Možni nivoji abstrakcije

- Instruction Set Architecture (ISA)
  - Emulate the ISA in software
    - Interprets, translates to host ISA (if required)
    - Device abstractions implemented in software
    - Inefficient
  - Optimizations: Caching?  Code reorganization?
  - Applications: Debugging, Teaching, multiple OS
- Hardware Abstraction Layer (HAL)
  - Between “real machine” and “emulator” (maps to real hardware)
  - Handling non-virtualizable architectures (scan, insert code?)
  - Applications: Fast and usable, virtual hardware (in above too), consolidation, migration

# Še o možnih nivojih abstrakcije

- Operating System Level
  - Virtualized SysCall Interface (may be same)
  - May or may not provide all the device abstractions
  - Easy to manipulate (create, configure, destroy)
- Library (user-level API) Level
  - Presents a different subsystem API to application
  - Complex implementation, if kernel API is limited
  - User-level device drivers
- Application (Programming Language) Level
  - Virtual architecture (ISA, registers, memory, …)
  - Platform-independence (➔ highly portable)
  - Less control on the system (extremely high-level)

# Značilnosti možnih abstrakcij

| | ISA | HAL | OS | Library | PL |
|---|---|---|---|---|---|
| Performance | * | **** | **** | *** | ** |
| Flexibility | **** | *** | ** | ** | ** |
| Ease of Impl | ** | * | *** | ** | ** |
| Degree of Isolation | *** | **** | ** | ** | *** |

**(več zvezdic je bolje)**

## Virtualizacija na nivoju operacijskega sistema

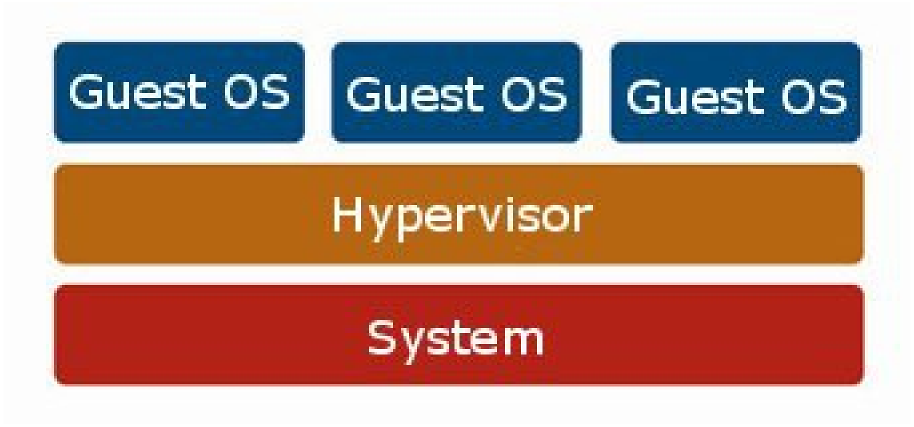
**Operating system-level virtualization** is a server virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. Such instances (often called containers, VEs, VPSs or jails) may look and feel like a real server, from the point of view of its owner.

On Unix systems, this technology can be thought of as an advanced implementation of the standard chroot mechanism. In addition to isolation mechanisms, the kernel often provides resource management features to limit the impact of one container's activities on the other containers.

WEB

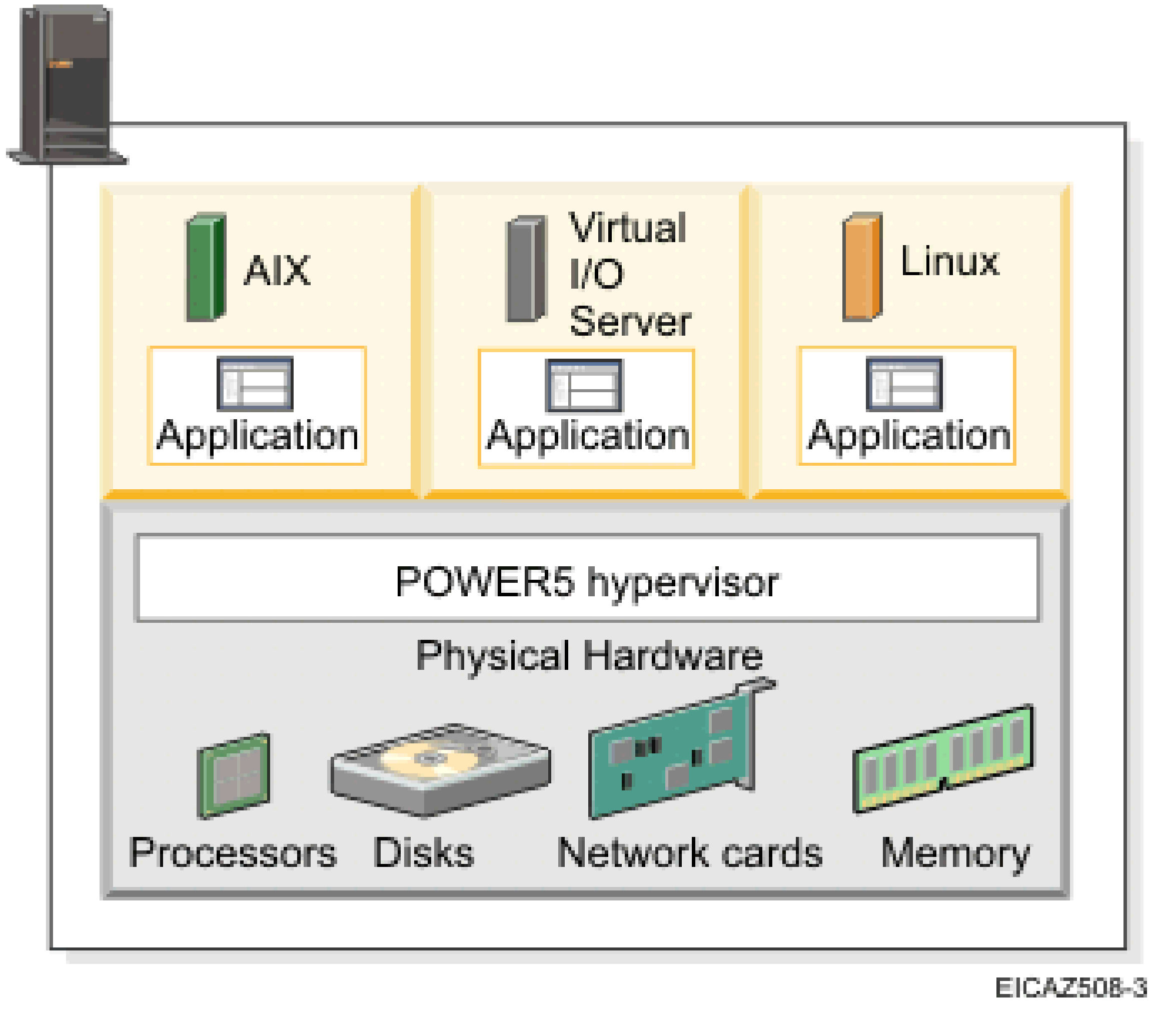
# Ključna tehnologija: hipervizor

- A computing layer which allows multiple operating systems to run on a host computer at the same time.
- Originally developed in the 1970s as part of the IBM S/360
- Many modern day variants from different developers



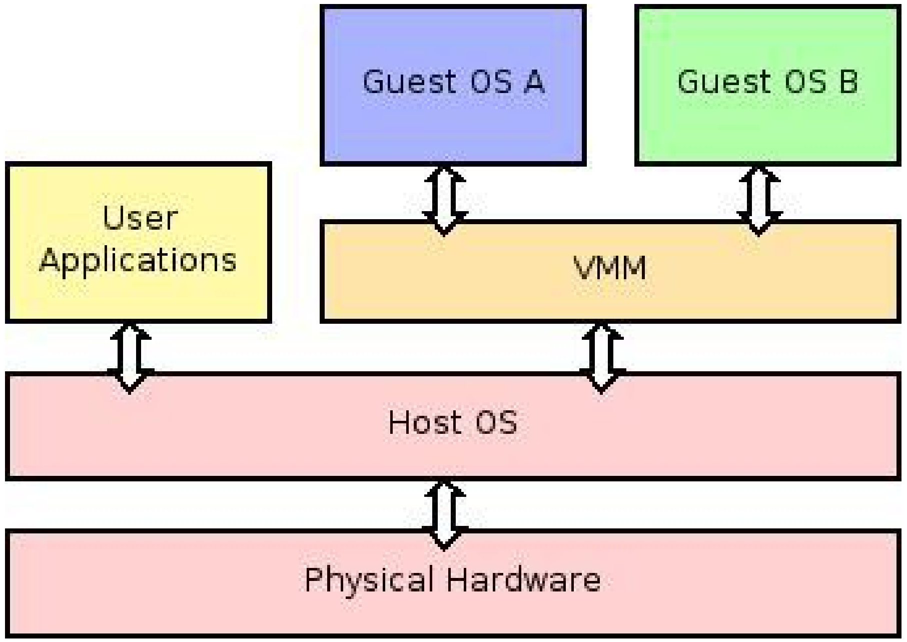
# Primer aparaturne virtualizacije

- IBM pSeries Servers



# Primer programske virtualizacije

- VMware Server (GSX)



## Virtualizacija na nivoju ISA (Instruction set architecture)

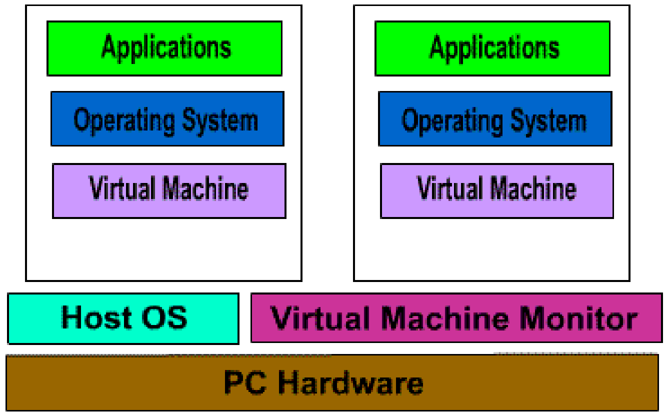
- Technologies
  - Emulation: Translates guest ISA to native ISA
  - Emulates h/w specific IN/OUT instructions to mimic a device
  - Translation Cache: Optimizes emulation by making use of similar recent instructions
  - Code rearrangement
  - Speculative scheduling (alias hardware)
- Issues
  - Efficient Exception handling
  - Self-modifying code

# Primeri virtualizacije na nivoju ISA

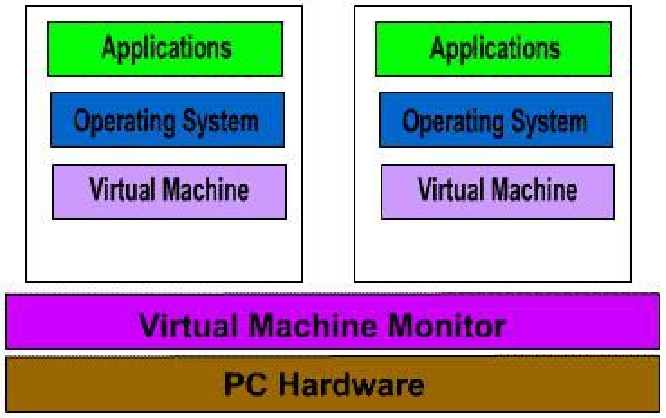
- **Bochs**: Open source x86 emulator
  - Emulates whole PC environment
    - x86 processor and most of the hardware (VGA, disk, keyboard, mouse, …)
    - Custom BIOS, emulation of power-up, reboot
    - Host ISAs: x86, PowerPC, Alpha, Sun, and MIPS
- **Crusoe** (Transmeta)
  - "Code morphing engine" – dynamic x86 emulator on VLIW processor
  - 16 MB "translation cache"
  - Shadow registers: Enables easy exception handling
- **QEMU**:
  - Full Implementation
    - Multiple target ISAs: x86, ARM, PowerPC, Sparc
    - Supports self-modifying code
    - Full-software and simulated (using mmap()) MMU
  - User-space only: Useful for Cross-compilation and cross-debugging

# Tehnike virtualizacije HAL (Hardware Abstraction Layer)

- Standalone vs. Hosted
  - Drivers
  - Host and VMM worlds
  - I/O
- Protection Rings
  - Multilevel privilege domains
- Handling “silent” fails
  - Scan code and insert/replace artificial traps
  - Cache results to optimize

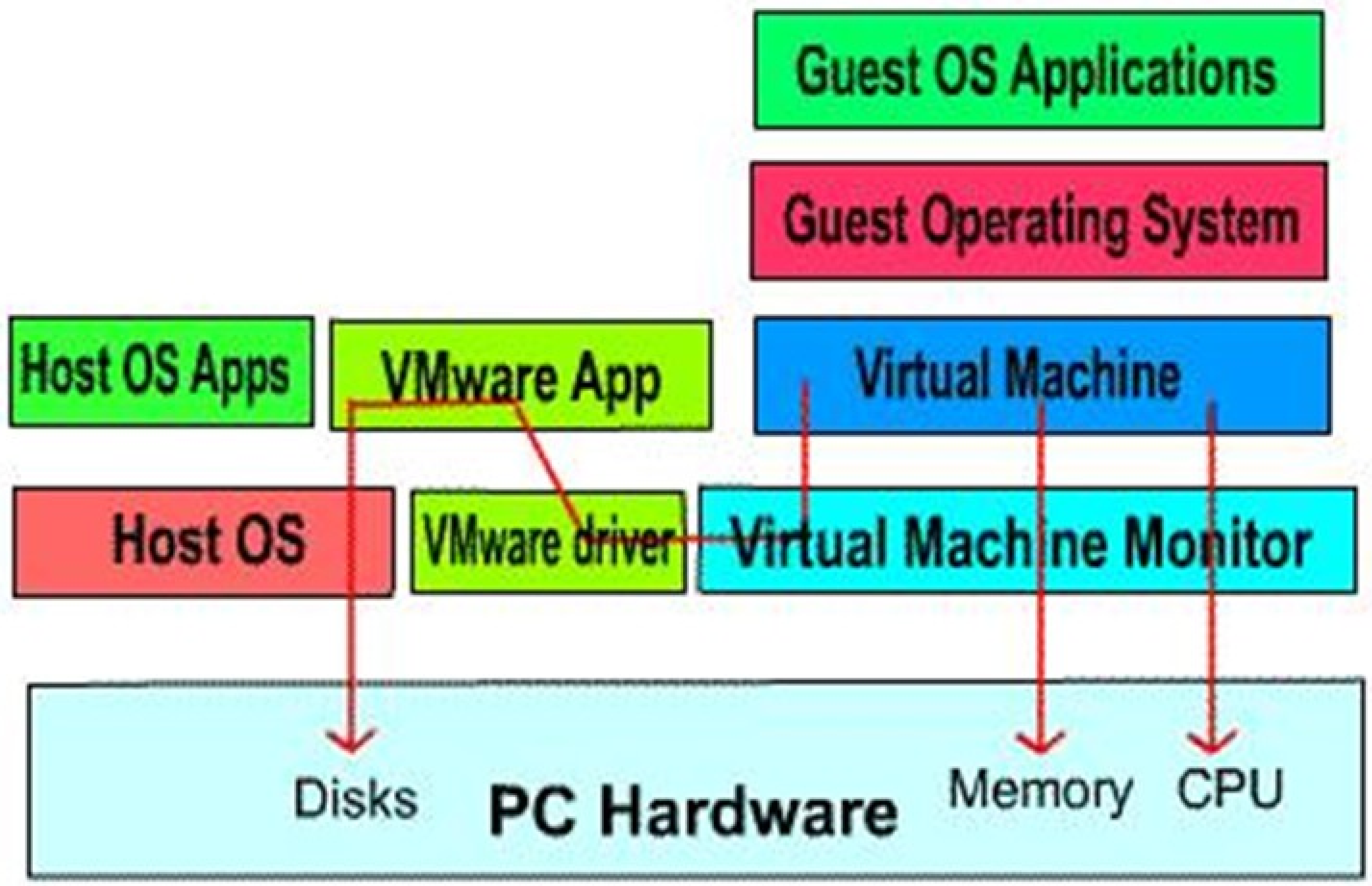


Hosted Virtual Machine



Stand Alone Virtual Machine

# Arhitektura Vmware Workstation



## VMware: I/O Virtualizacija

- VMM does not have access to I/O
- I/O in “host world”
  - Low level I/O instructions (issued by guest OS) are merged to high-level I/O system calls
  - VM Application executes I/O SysCalls
- VM Driver works as the communication link between VMM and VM Application
- World switch needs to “save” and “restore” machine state
- Additional techniques to increase efficiency

# Primerjava aparaturne in programske virtualizacije

**S/W based (x86)**

- Requires ‘emulation’ of guest’s privileged code
  - can be implemented very efficiently: Binary Translation (BT)
- Does not allow full virtualization
  - sensitive unprivileged instructions (SxDT)
- Widely used today
  - VMWare, VirtualPC

**H/W virtualization**

- VT-x (Intel IA32)
- SVM/Pacifica (AMD64)
- Does not require guest’s priv code emulation
- Should allow for full virtualization of x86/x64 guests
- Still not popular in commercial VMMs

# Full VMMs vs. “Thin” hypervisors

**<u>Full VMMs</u>**

- Create full system abstraction and isolation for guest,
- Emulation of I/O devices
  - Disks, network cards, graphics cards, BIOS…
- Trivial to detect,
- Usage:
  - server virtualization,
  - malware analysis,
  - Development systems

**<u>“Thins hypervisors”</u>**

- Transparently control the target machine
- Based on hardware virtualization (SVM, VT-x)
- Isolation not a goal!
  - native I/O access
  - Shared address space with guest (sometimes)
- Very hard to detect
- Usage:
  - stealth malware,
  - Anti-DRM

# Virtualizacija- druga plat medalje

WEB

Blue Pill:  virtualization based malware

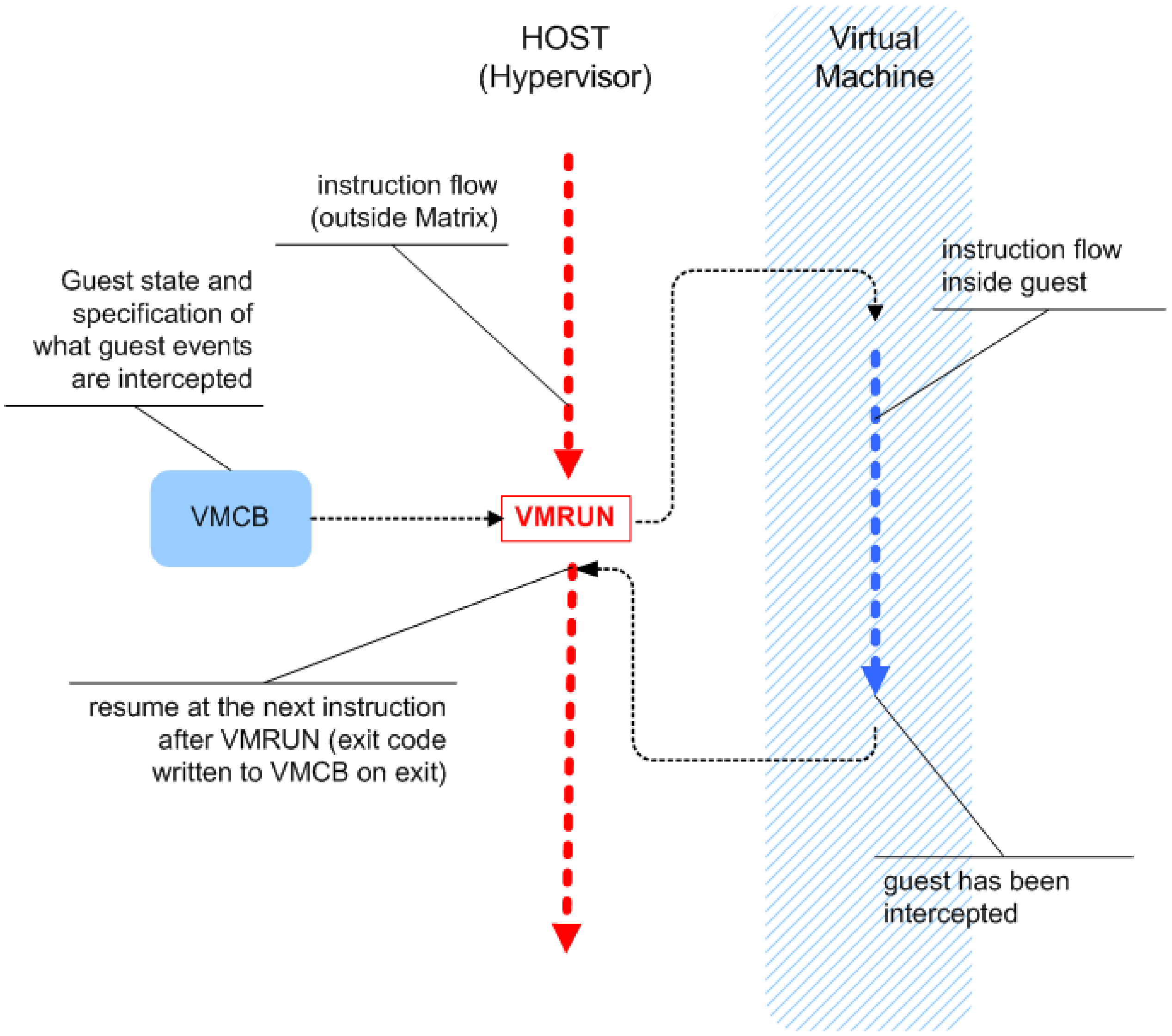
# Ideja modre tabletke

- Exploit AMD64 SVM extensions to move the operating system into the virtual machine (do it ‘on-the-fly’)
- Provide thin hypervisor to control the OS
- Hypervisor is responsible for controlling “interesting” events inside gust OS

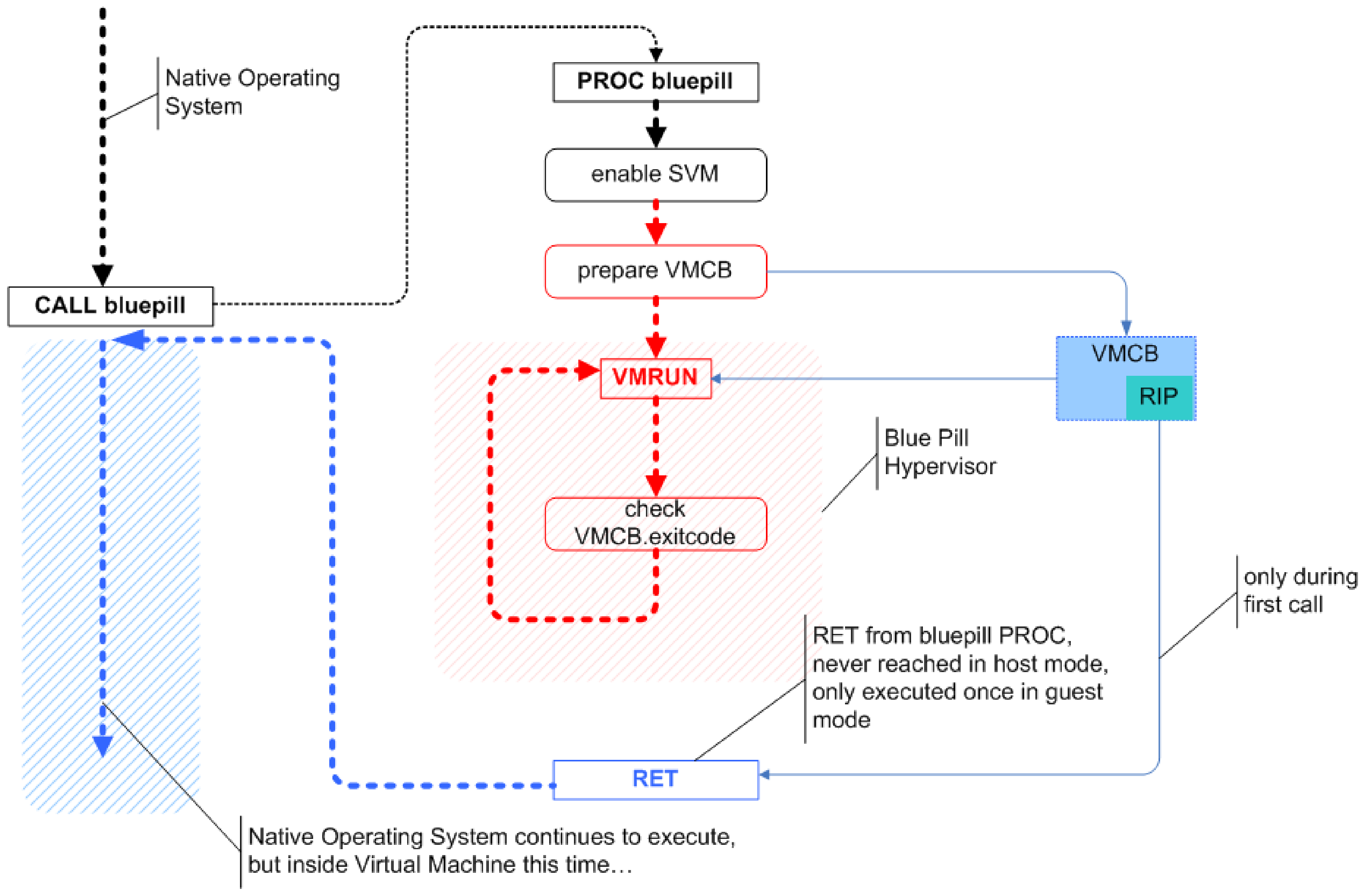
## SVM

- SVM is a set of instructions which can be used to implement Secure Virtual Machines on AMD64
- MSR EFER register: bit 12 (SVME) controls weather SVM mode is enabled or not
- EFER.SVME must be set to 1 before execution of any SVM instruction.
- Reference:
  - AMD64 Architecture Programmer’s Manual Vol. 2: System Programming Rev 3.11
  - http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf

# Srce SVM: Instrukcija VMRUN



# Ideja modre tabletke (poenostavljeno)


Native Operating System
PROC bluepill
enable SVM
prepare VMCB
CALL bluepill
VMRUN
VMCB
RIP
Blue Pill Hypervisor
check VMCB.exitcode
only during first call
RET from bluepill PROC, never reached in host mode, only executed once in guest mode
RET
Native Operating System continues to execute, but inside Virtual Machine this time…

## BP installs itself ON THE FLY!

- The main idea behind BP is that it installs itself on the fly
- Thus, no modifications to BIOS, boot sector or system files are necessary
- BP, by default, does not survive system reboot

## SubVirt Rootkit

- SubVirt has been created a few months before BP by researches at MS Research and University of Michigan
- SubVirt uses commercial (full) VMM (Virtual PC or VMWare) to run the original OS inside a VM…

WEB

WEB

## Primerjava SubVirt in Blue Pill

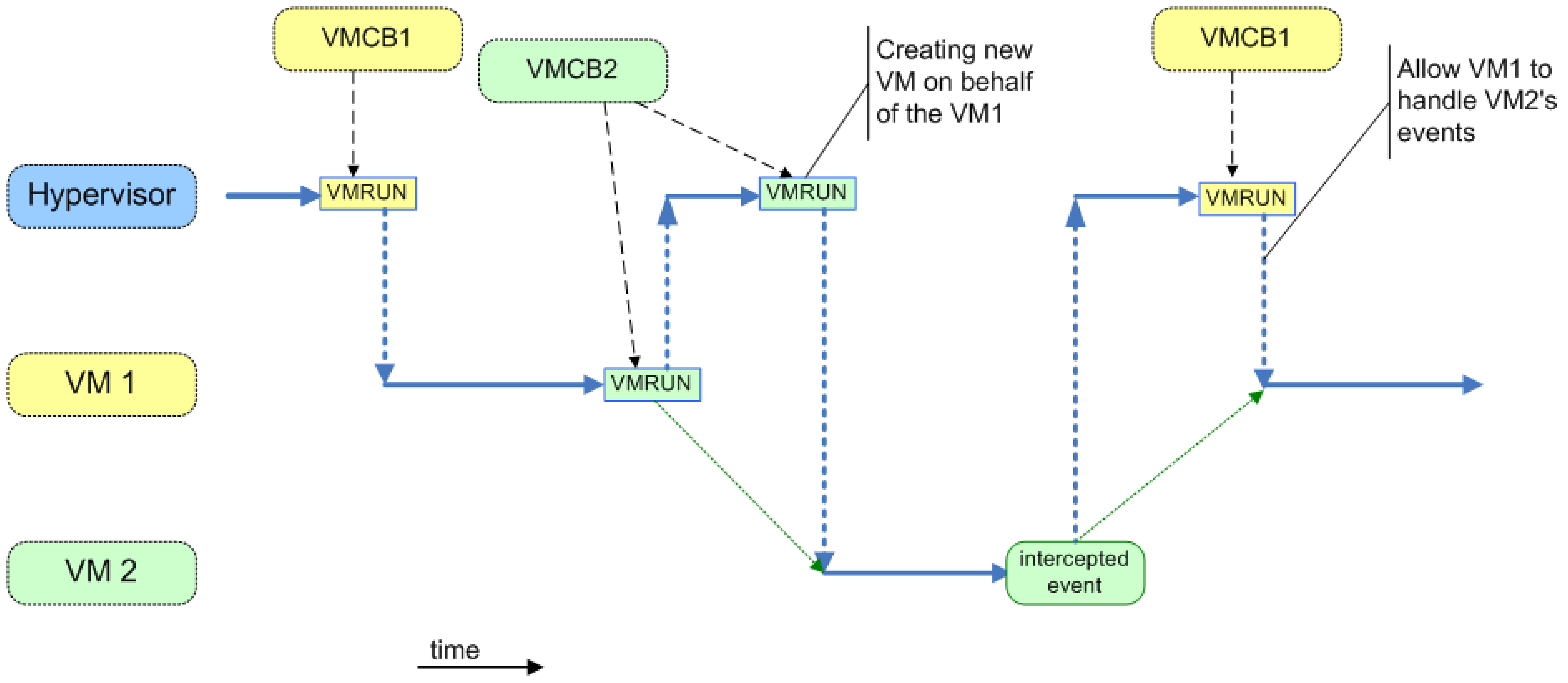
- SV is permanent! SV has to take control before the original OS during the boot phase. SV can be detected off line.
- SV runs on x86, which does not allow for full virtualization (e.g. SxDT attack)
- SV is based on a commercial VMM, which creates and emulates virtual hardware. This allows for easy detection

- Blue Pill can be installed on the fly – no reboot nor any modifications in BIOS or boot sectors are necessary. BP can not be detected off line.
- BP relies on AMD SVM technology which promises full virtualization
- BP uses ultra thin hypervisor and all the hardware is natively accessible without performance penalty

# Matrix inside another Matrix

- What happens when you install Blue Pill inside a system which is already “bluepilled”?
- If nested virtualization is not handled correctly this will allow for trivial detection – all the detector would have to do was to try creating a test VM using a VMRUN instruction
- Of course we can cheat the guest OS that the processor does not support SVM (because we control MSR registers from hypervisor), but this wouldn’t cheat more inquisitive users ;)
- So, we need to handle nested VMs…

# Gnezdeni virtualni stroji



## Blue Pill based malware

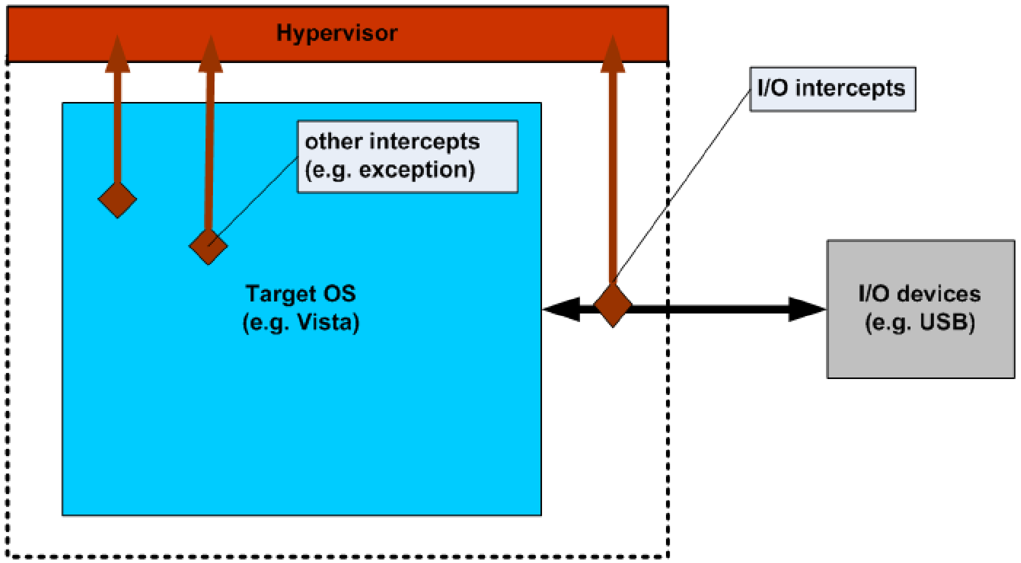
- Blue Pill is just a way of silently moving the running OS into Matrix on the fly
- BP technology can be exploited in many various ways in order to create stealth malware
- Basically ‘sky is the limit’ here :)
- On the next slides we present some simple example:

# “Delusion Backdoor”

- Simple Blue Pill based network backdoor
- Uses two DB registers to hook:
  - `ReceiveNetBufferListsHandler`
  - `SendNetBufferListsComplete`
- Blue Pill takes care of:
  - handling #DB exception (no need for IDT[1] hooking inside guest)
  - protecting debug registers, so that guest can not realize they are used for hooking
- Not even a single byte is modified in the NDIS data structures nor code!
- Delusion comes with its own TCP/IP stack based on lwIP

# Lastništvo ciljnega OS

- If we manage to run the target OS inside a hardware hypervisor controlled by ourselves…
- …we can bypass all the OS-provided prevention technologies
  - kernel prevention
  - anti-debugging
- It does not matter what OS is running inside
  - we control the guest’s memory
  - we control the guest’s I/O
  - we can set hardware breakpoints



## Odkrivanje modre tabletke in drugih hipervizorjev

- Timing analysis
- Exploiting implementation bugs (processor/system bugs)
- Using dedicated CPU instruction (not available now)

## Časovna analiza

- Direct Timing Analysis
  - measure execution time of an instruction which you know (or suspect) that is intercepted by the hypervisor…
  - e.g. RDMSR EFER
- Indirect Timing Analysis
  - measure execution time of some operation which is not intercepted by the hypervisor, but its timing is affected by the presence of a hypervisor
  - e.g. page access time
    - should work on VT-x, which does not use tagged TLB

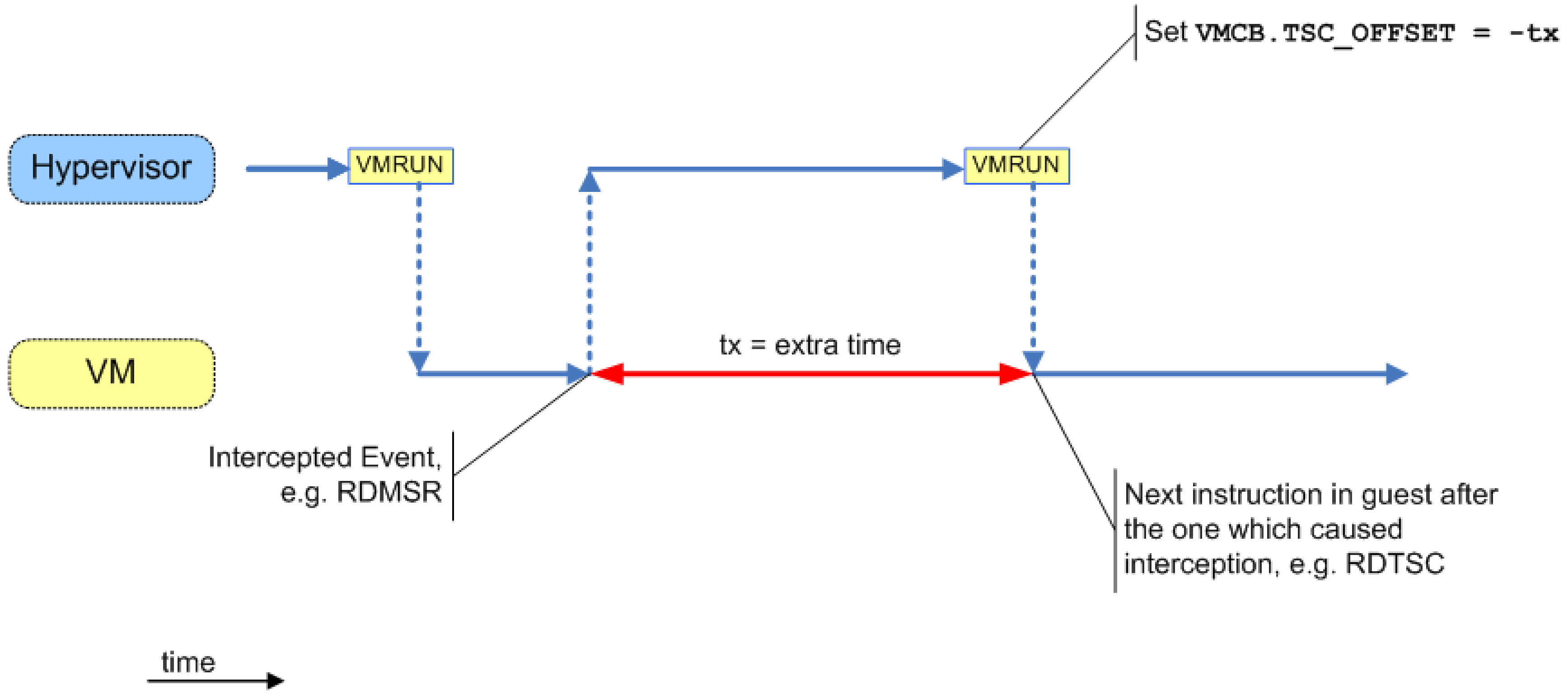
# Merjenje časa

- CPU Tic Counter
  - RDTSC instruction
  - resolution: number of processor cycles (super high!)
  - very accurate, but trivial to cheat!
- Real Time Clock
  - I/O with RTC device
  - resolution: milliseconds (poor)
  - relatively easy to cheat (I/O interceptions)
- External clock
  - e.g. NTP protocol
  - resolution: seconds (very poor)
  - can not be cheated using generic approach – only attacks against specific implementation

## Zanesljivost merjenja časa

- RDTSC
  - RDTSC can be cheated via VMCB.TSC_OFFSET
  - Also, RDTSC is “interceptable”
- Real Time Clock
  - Can be cheated via I/O interception
- Network Time Protocol (NTP)
  - Can be cheated via incoming packet manipulation
- Encrypted NTP
  - Can not be generically cheated
  - Implementation-Specific Attacks only

# Dilatacija časa za gostujoči OS


Set VMCB.TSC_OFFSET = -tx
Hypervisor
VMRUN
VMRUN
VM
tx = extra time
Intercepted Event, e.g. RDMSR
Next instruction in guest after the one which caused interception, e.g. RDTSC
time

## Izkoriščanje hrošče implementacije

- We can expect that some (complex) CPU instructions to behave (sometimes) in some way differently when executed inside a hardware VM…
- In other words: CPU implementation bugs
- On the one hand they are nearly ideal solution…
- …but on the other hand: they are just bugs!
  - means they will be fixed sometime
  - might not affect all processors (e.g. only some revisions)
  - should legitimate products rely on bugs?

## Aparaturna rdeča tabletka?

- How about creating a new instruction – **`SVMCHECK:`**
```
mov rax, <password>
svmcheck
cmp rax, 0
jnz inside_vm
```
- Password should be different for every processor
- Password is necessary so that it would be impossible to write a *generic* program which would behave differently inside VM and on a native machine.
- Users would get the passwords on certificates when they buy a new processor or computer
- Password would have to be entered to the AV program during its installation.
- This could only by done by AMD/Intel! Let's push on them!

# Zaščita: blokiranje namestitve hipervizorjev

- Disable virtualization!
  - “Do not plug your computer to the Internet” ;)
- Install “preventive” hypervisor first (right after the system boot)
  - What policy used to allow/block installation of other (nested) hypervisor?

# “Preventivni” hipervizor

- As a defense we can install preventive hypervisor that will not allow any other to be installed later, e.g. as a result of kernel bug exploitation…
  - requires that we ensure secure boot process
  - otherwise we cannot know whether our hypervisor is the “real” one or “nested”! (see earlier)
- <u>Should be very lightweight!</u>
- Policy?
  - disallow any other hypervsiors to load later?
    - Effectively disabling the virtualization technology (which has some legitimate usages after all)
  - allow only “trusted” hypervsiors?
    - What is “trusted”?
    - How can we ensure there are no bugs in “trusted” hypervisors?

## Kako to implementirati

- Preventive Hypervisor can not share the address space with the guest
- Shadow Paging/Nested Paging is required to prevent tampering with hypervisor memory
- IOMMU should be used to prevent from accessing hypervisor’s physical memory via DMA
  - currently the “poor man’s alternative” to IOMMU on AMD processors is External Access Protection (EAP) technology
    - there does not seem to be any similar technology today on Intel VT-x processors (i.e. similar to EAP)
  - IOMMU is expected to be available in 2008 on most Intel and AMD processors though

## Ne dovolimo zagona OS znotraj VM?

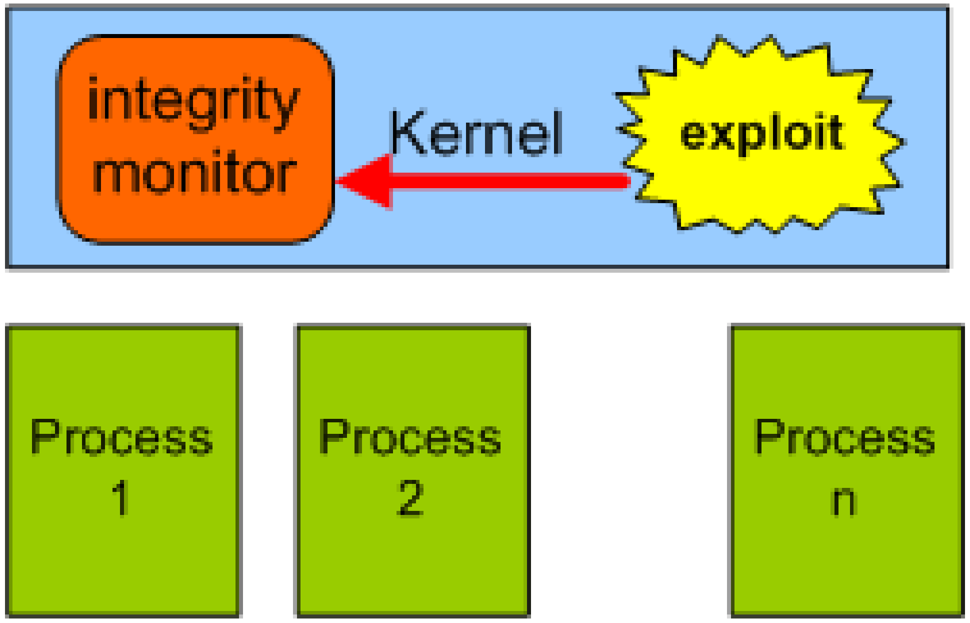
- Unclear whether it is possible
- If possible, then for sure with the help of TPM/Bitlocker
- Subject of further research…

- Note that this is complementary to “preventive hypervisor” idea (or some detection) – as it does not protect against hypervisors installations via kernel exploit (Blue Pill)
  - This is required only to prevent anti-DRM attacks

## Hipervizorji kot varnostniki

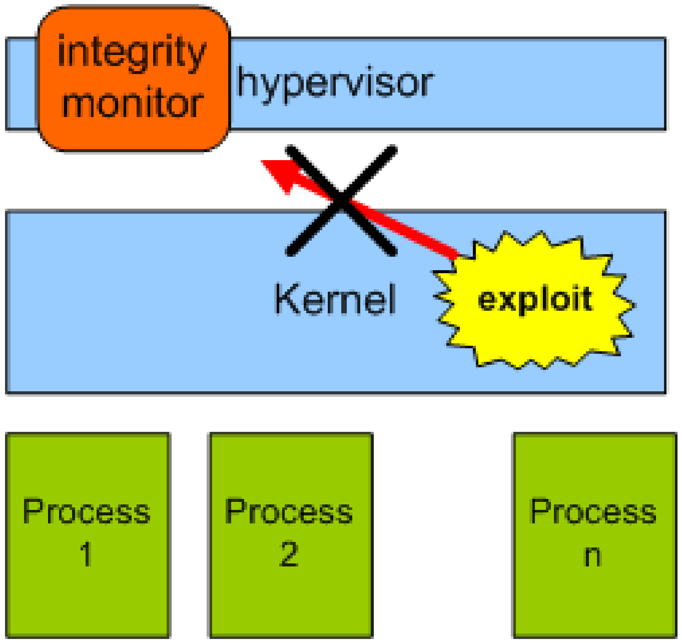
# Nadzor integritete na nivoju jedra

- Prone to all attacks from within kernel
- Kernel protection is very hard (impossible) in most OS!



## Nadzor integritete znotraj hipervizorja

- Hypervisors can be much better controlled then kernel
  - hypervisor can be very “thin” (easy verifiable)
- An integrity monitor inside hypervisor is not prone to direct attacks from kernel mode!



## Legalna vprašanja?

- Such hypervisor would have to be installed during secured boot process
  - otherwise we can not ensure that it’s not “nested”!
- Is it acceptable to install such a hypervisor by
  - An A/V program
  - DRM application?

# Končni razmislek

- We cannot implement <u>effective </u>preventive hypervisors today, because of:
  - the lack of IOMMU technology (DMA attacks)
  - the lack of Nested Paging (Shadow Paging is too slow)
- Detection is very hard:
  - Should not be based on CPU bugs!!!
  - Need to use encrypted NTP for reliable timing
- In other words, we can not effectively fight virtualization based malware today!

# Končno sporočilo

- Virtualization technology on Intel and AMD processors seems to be very immature as of today –
  - It allows for effective malware implementation
  - But does not allow for effective fighting with such a malware!
- Because of the lack of IOMMU and Nested Paging, current hardware virtualization does not seem to offer any significant benefit over traditional, software based virtualization, to create full VMMs (e.g. VMWare)