Virtualizációs Technológiák

Bevezetés

Kovács Ákos

Forrás, BME-VIK

Virtualizációs technológiák

https://www.vik.bme.hu/kepzes/targyak/VIMIAV89/

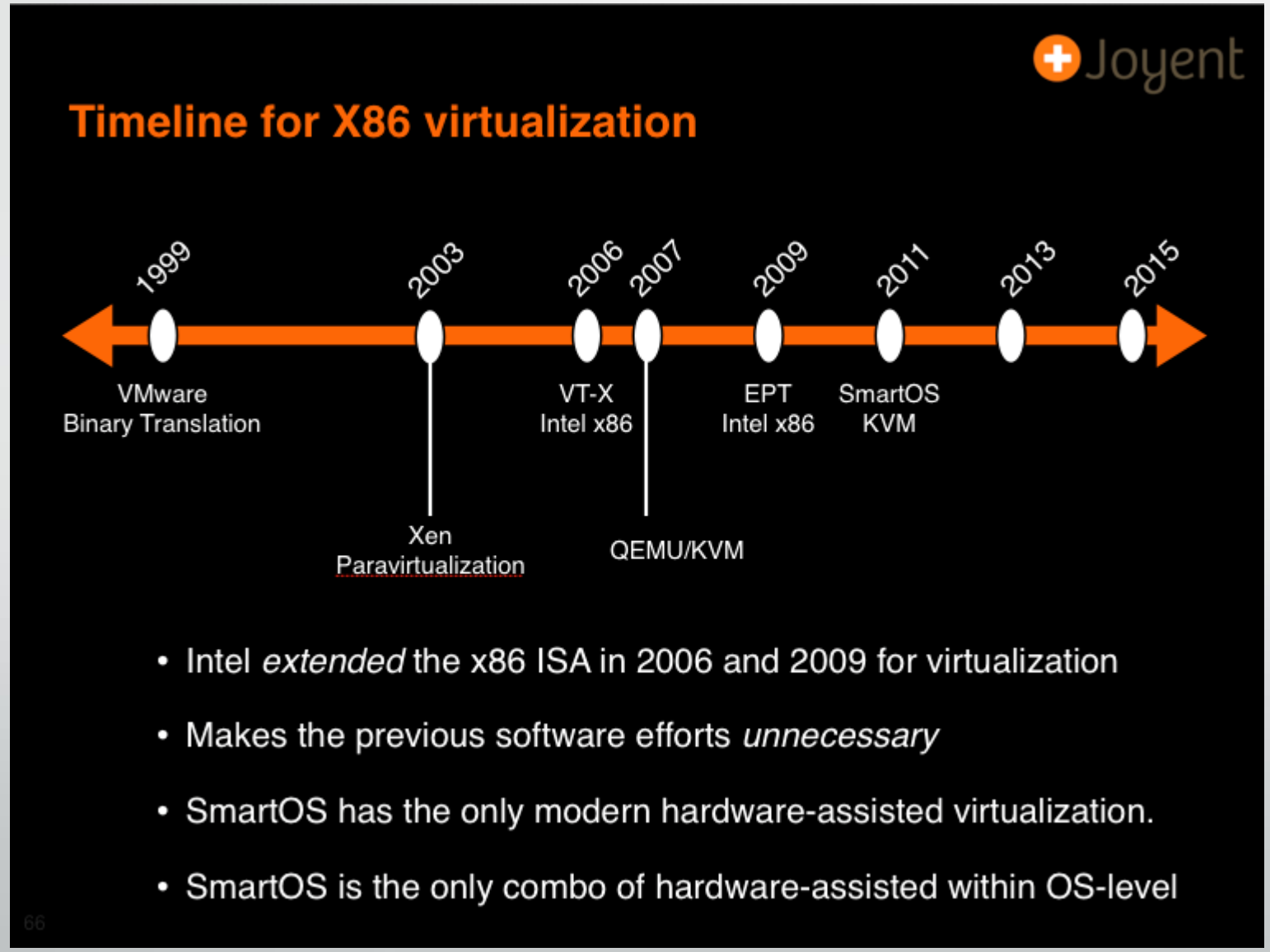
Mi is az a Virtualizáció?

- „Az erőforrások elvonatkoztatása az erőforrást nyújtó elemektől”

  - kellemesen sejtelmes általános definíció

- Jellemzően:
  - fizikai erőforrásokból logikai erőforrások képzése, amik függetlenek a tényleges fizikai elemektől
  - korlátos erőforrások szétosztása több részre
- Ez egy új ötlet?
  - Korántsem – az oprendszerek is ezt csinálják…
  - Mobilplatform (Android, Java)

Történeti áttekintés



- http://jason.files.wordpress.com/2011/12/screen-shot-2011-12-22-at-9-33-28-am.png

Mi kell egy virtuális CPU-hoz

- *VMM – virtual machine monitor*:
  olyan komponens, ami a virtuális gépek számára az absztrakciót biztosítja

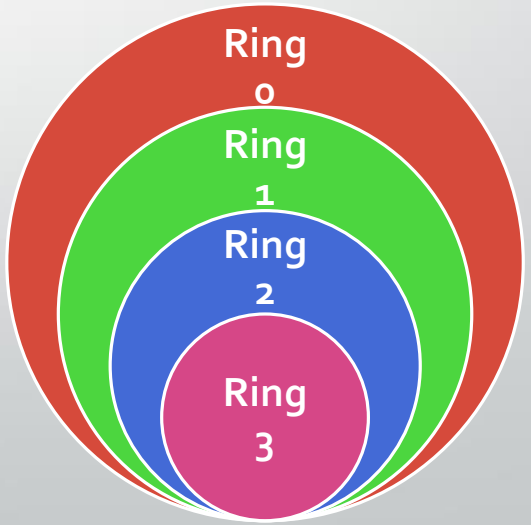
Popek és Goldberg szerint egy VMM alapvető jellemzői:

- **Ekvivalencia**
  - A VMM felett futó program mindig pontosan ugyanazt az eredményt adja futás közben, mintha fizikai CPU-n közvetlenül futna
- **Erőforrás kezelés**
  - A VMM minden virtualizált erőforrást teljes egészében maga felügyel
- **Hatékonyság**
  - A virtuális gépben futó program utasításainak nagy része változtatás és VMM beavatkozás nélkül fut a fizikai CPU-n

Mit tud a processzor – üzemmódok

- A CPU üzemmódok célja:
  - Visszamenőleges kompatibilitás (x86 másból sem áll…)
    - Pl.: valós mód (16 bit), védett mód (16 és 32 bit), long mód (64 bit)…
  - Egy program ne tudjon bizonyos műveleteket végezni
  - Operációs rendszer el tudja szigetelni a programokat egymástól
    - „védett” módok
    - Futási **privilégium szintek**, al-üzemmódok (rings)

- Példa: 4 privilégium szint az x86 védett módjában
  - Ring 0, *supervisor* mód: legbővebb utasításhalmaz
  - Ring 1
  - Ring 2
  - Ring 3, *user* mód: legszűkebb részhalmaz



VMM jellemzői

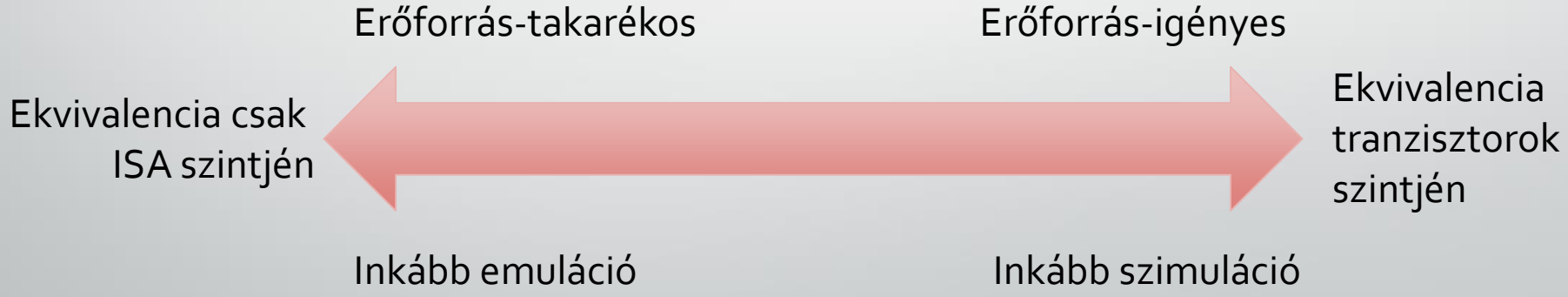
Mik a problémák ezzel?

- Az ekvivalencia és az erőforrás kezelési kritérium ellentmondó követelményeket támaszt
- Ha a VMM kezeli az erőforrásokat, mit csinál a virtuális gépben futó OS?

Szimuláció és emuláció

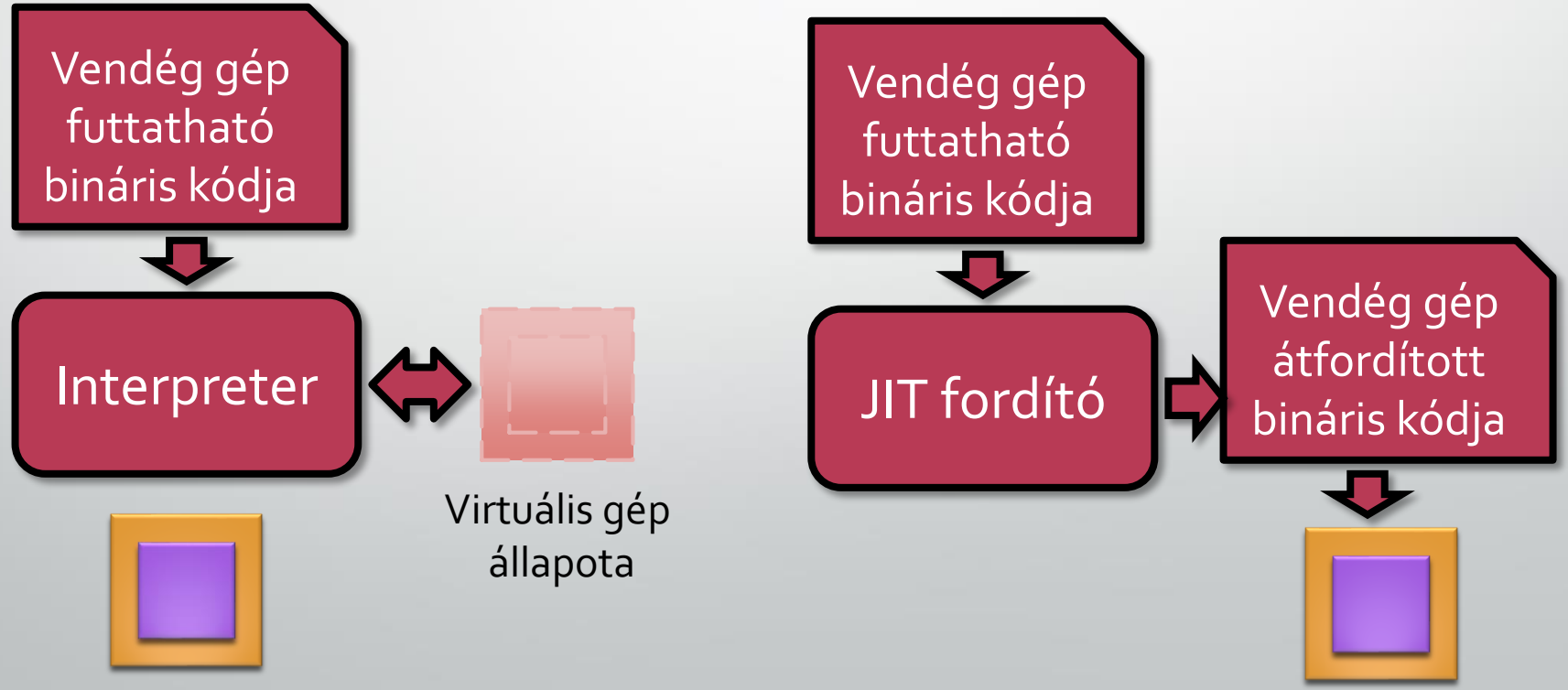
Ekvivalencia biztosítása

- **Szimuláció**: szoftverrel modellezzük a processzor belső működését (különböző mélységig megtartva a valósághűséget). Fizikailag sohasem fut a virtuális gép kódja a CPU-n.
- **Emuláció**: helyettesítjük a VMM felett futtatandó programot/annak egy részletét egy másikkal, ami végül pontosan ugyanazt az eredményt adja (de akár teljesen más futási utat bejárva, elkerülve a privilegizált utasítások használatát)



Tiszta emuláció

- A vendég virtuális gép kódját a processzor nem futtatja közvetlenül, hanem adatként feldolgozza
  - Eltérhet a virtuális gép CPU architektúrája a futtató CPU-tól
  - Virtualizációhoz (P&G értelemben vett) képest lassú



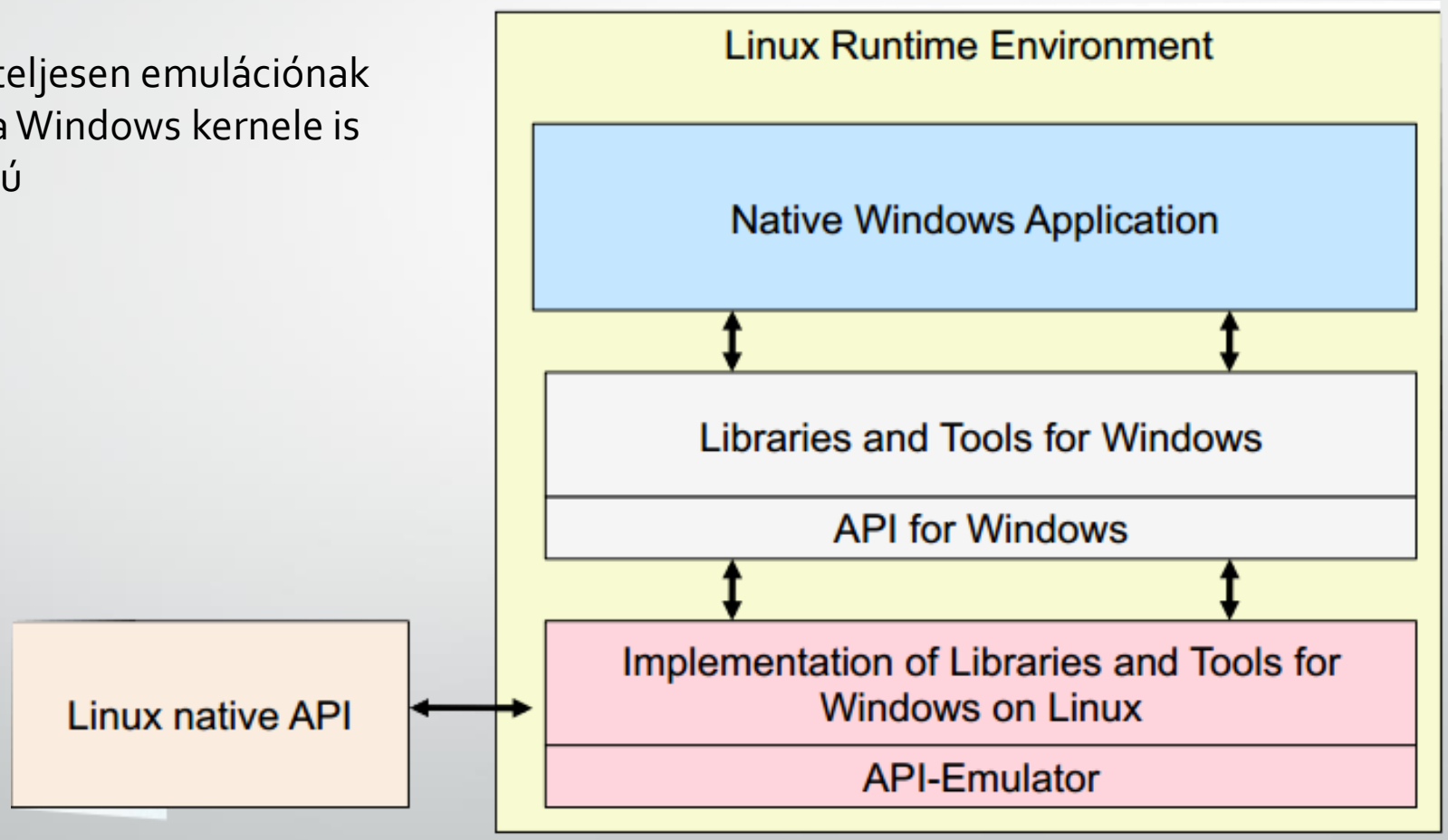
Emuláció megvalósításának lehetőségei

Emuláció (és szimuláció) kétféleképpen

- **Futási idejű értelmező (Interpreter)** – adatként kezeli és lépésenként hajtja végre egy szoftveres modellen a virtuális gép utasításfolyamát
  - Lassú, a CPU közvetlenül csak az értelmező kódját futtatja
  - Hordozható futtatókörnyezet
  - Izoláció természetesen adódik
- **„Éppen időben" fordító (JIT compiler „just in time")** – végrehajtás előtt egy fordító feldolgozza a virtuális gép soron következő utasításait, és kódot generál belőle, ami az eredetivel ekvivalens viselkedést mutat
  - Közvetlenül futtatható kódot generál, cache-elési eljárásokkal gyors lehet
  - Nehéz implementálni, nem hordozható
  - Izolációt nem automatikusan biztosítja
  - Példa: QEMU x86 emulátor *(és Java VM, .NET CLR is ilyen)*

WINE „Emuláció”

Nem nevezhető teljesen emulációnak mivel a Linux és a Windows kernele is x86 architektúrájú

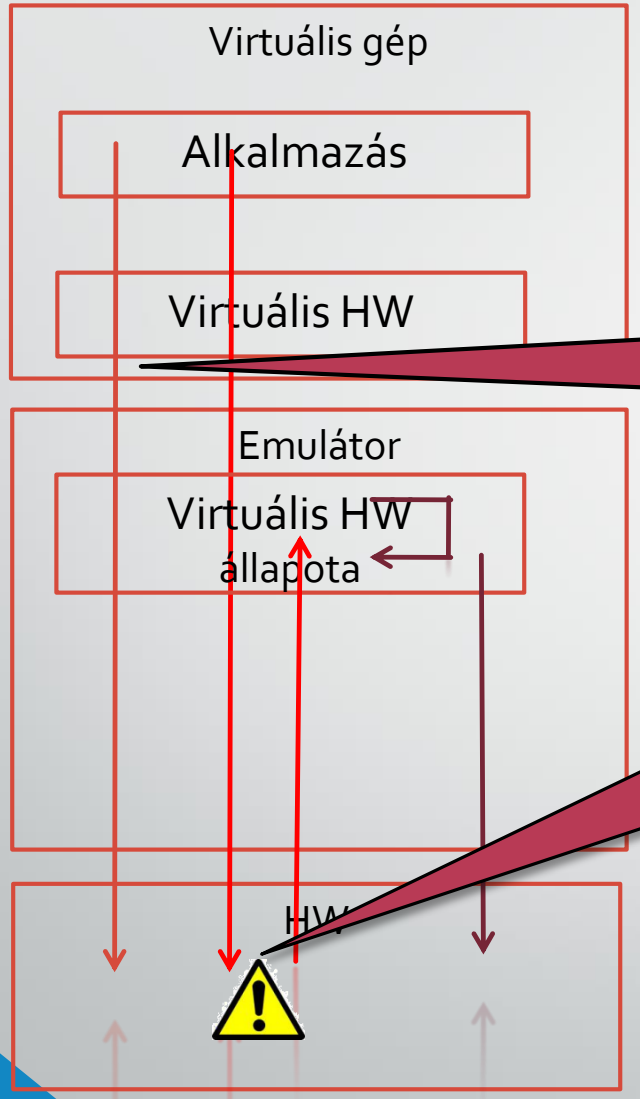


Virtualizáció

- **Virtualizáció**:
  - A vendég gép utasításainak egy részét beavatkozás nélkül végrehajtja a fizikai CPU-n
  - A privilegizált utasításokat kell külön kezelni
  - Klasszikus módszer: *trap & emulate*

- Következmény:
  - Gyorsabb, mint az emuláció
  - A vendég és a fizikai gép ISA-ja megegyezik
    - ISA=utasításkészlet

Trap & emulate módszer



(**Trap**: hardveres kivételkezelő rutin ami után a végrehajtás folytatódhat)

A nem privilegizált utasítások közvetlenül a valós CPU-n hajtódnak végre

A privilegizált vagy érzékeny műveletek trap-et váltanak ki, és a VMM veszi át a végrehajtást

HW támogatás szükséges:
- védelmi szintek
- virtuális gép alacsony védelmi szinten fut
- privilegizált utasítások nem megfelelő szinten kiadva trap-et okoznak

Natív és virtualizált rendszer összehasonlítása

A VMM-ben tett „kör” költséges, a CPU üzemmód váltás során teljes állapotmentés majd a végén visszaállítás kell!

- Mikor éppen virtualizált operációs rendszer fut
- SYSCALL és INT továbbra is a Ring3-ból Ringo-ba hív át!
- A VMM megkapja a vezérlést, feladata hogy továbbítsa a hívást a vendég kernelnek
- Amikor a vendég kernel Ring1-ben nem engedélyezett utasítást hajtana végre a VMM elkapja, lekezeli, és úgy tesz, mintha megtörtént volna.
- A VMM kapja a timer interruptot is, így ütemezheti a vendég gépeket. Ezt is továbbítja az aktív vendég kernel felé is.
- A vendég OS – teljesítménytől eltekintve – nem tud róla, hogy virtualizált

Három lehetőség a virtualizációra (x86)

- **Szoftveres virtualizáció**
  - (Trap and Emulate + bináris fordítás)
- **Paravirtualizáció**
  - (vendég OS kódját módosítjuk, hogy tudjon a virtualizációról, és hívhassa a VMM-et)
- **Hardveres virtualizáció**
  - (Trap and Emulate, teljesen hardveres támogatással)

Bináris fordítás (binary translation)

- Mit tegyünk a problémás utasításokkal?
- Szoftveres virtualizáció **bináris fordítás**sal (VMware megoldása)
  - Egy JIT fordító a végrehajtás előtt végignézi a kód szegmenst és kicseréli a problémás utasításokat pl. SYSCALL-ra, vagy valami egyéb kódrészletre
  - Kicserélhet egyéb, amúgy elfogható utasításokat is rögtön a kezelő kódrészlettel, hogy elkerülje a felesleges hívást a VMM-be (inline translation)
  - Mivel a kód hossza megváltozhat ezért a kódra mutató pointereket (jump, branch utasítások) is módosítani kell
  - A végrehajtás menete: elő-fordítás + virtualizált végrehajtás elfogással + elfogott utasítások emulációja
- Teljesítmény?
  - Nem teljes fordítást végez, az utasítások többségét változatlanul hagyja
  - A JIT fordító minden kódrészletet csak az első futáskor jár be, ismételt futtatáskor már cache-eli a már módosított kódrészeket
  - Optimalizálással csökkenthető a VMM-be történő hívások száma

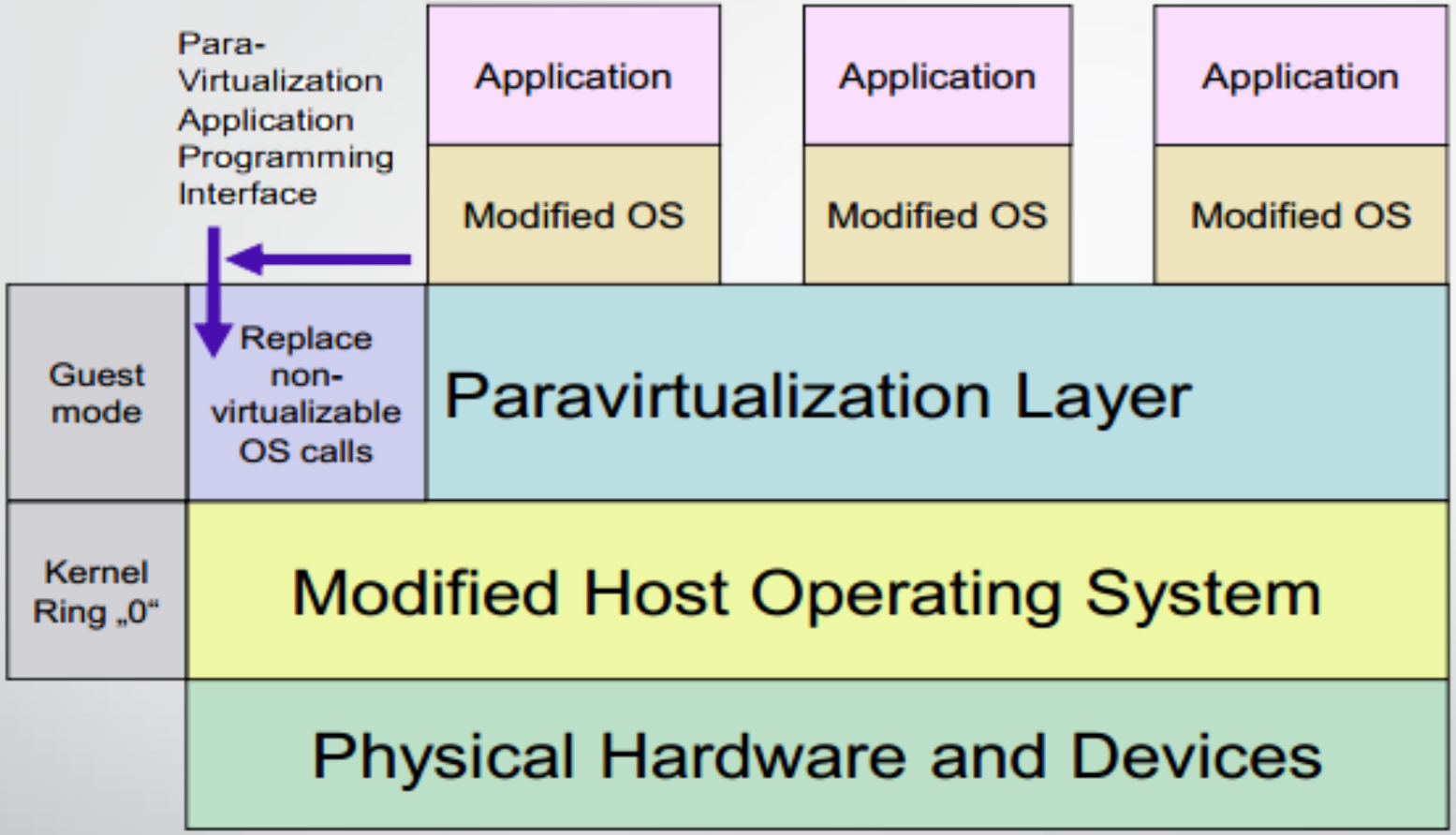
Három lehetőség a virtualizációra (x86)

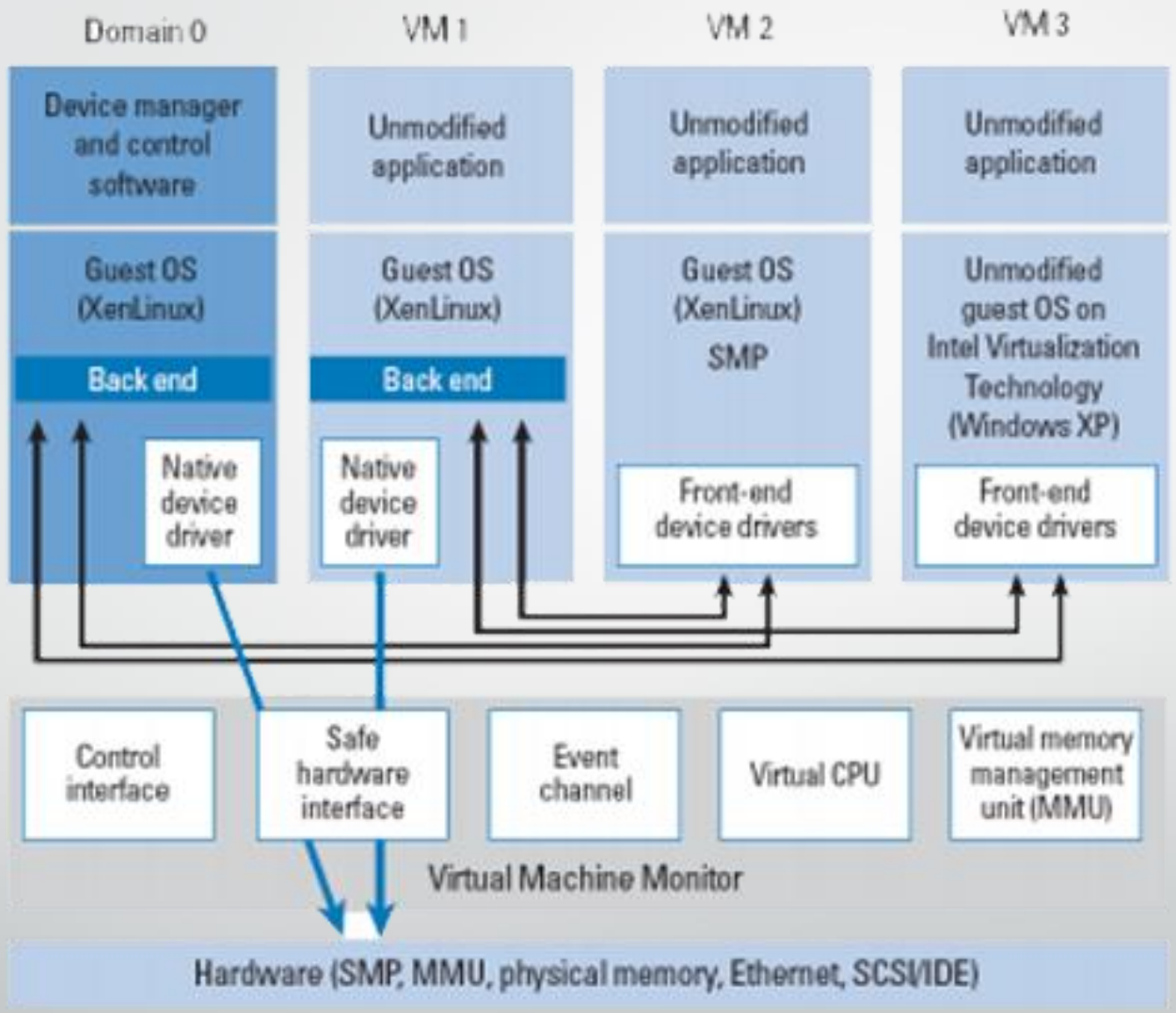
- **Szoftveres virtualizáció**
  - (Trap and Emulate + bináris fordítás)
- **Paravirtualizáció**
  - (vendég OS kódját módosítjuk, hogy tudjon a virtualizációról, és hívhassa a VMM-et)
- **Hardveres virtualizáció**
  - (Trap and Emulate, teljesen hardveres támogatással)

Paravirtualizáció

- Módosítsuk a vendég OS kernelt, hogy ne használjon elfoghatatlan utasításokat!
- Nem kell semmiféle előfordító
- A vendég OS tehát kifejezetten tud róla, hogy virtualizált

- Ne csak az elfoghatatlan, de minden „szükségtelen” vagy „elkerülhető” privilegizált utasítást irtsunk ki a vendég kernelből –> Kevesebb váltás kell a VMM-be.
- Vezessünk be saját rendszerhívásokat a vendég kernel-VMM kommunikációra, és amit csak lehet, ezzel oldjuk meg (perifériák kezelése)

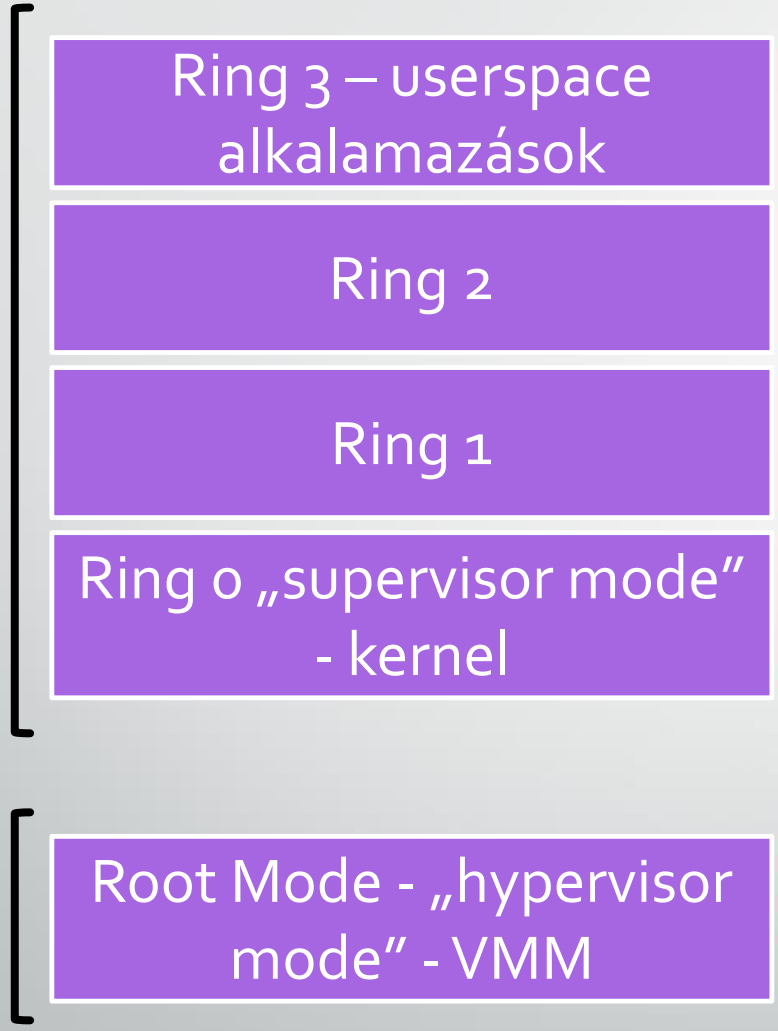

Para-
Virtualization
Application
Programming
Interface
Application
Application
Application
Modified OS
Modified OS
Modified OS
Guest
mode
Replace
non-
virtualizable
OS calls
Paravirtualization Layer
Kernel
Ring „0“
Modified Host Operating System
Physical Hardware and Devices


Domain 0
VM 1
VM 2
VM 3
Device manager and control software
Unmodified application
Unmodified application
Unmodified application
Guest OS (XenLinux)
Guest OS (XenLinux)
Guest OS (XenLinux) SMP
Unmodified guest OS on Intel Virtualization Technology (Windows XP)
Back end
Back end
Native device driver
Native device driver
Front-end device drivers
Front-end device drivers
Control interface
Safe hardware interface
Event channel
Virtual CPU
Virtual memory management unit (MMU)
Virtual Machine Monitor
Hardware (SMP, MMU, physical memory, Ethernet, SCSI/IDE)

Három lehetőség a virtualizációra (x86)

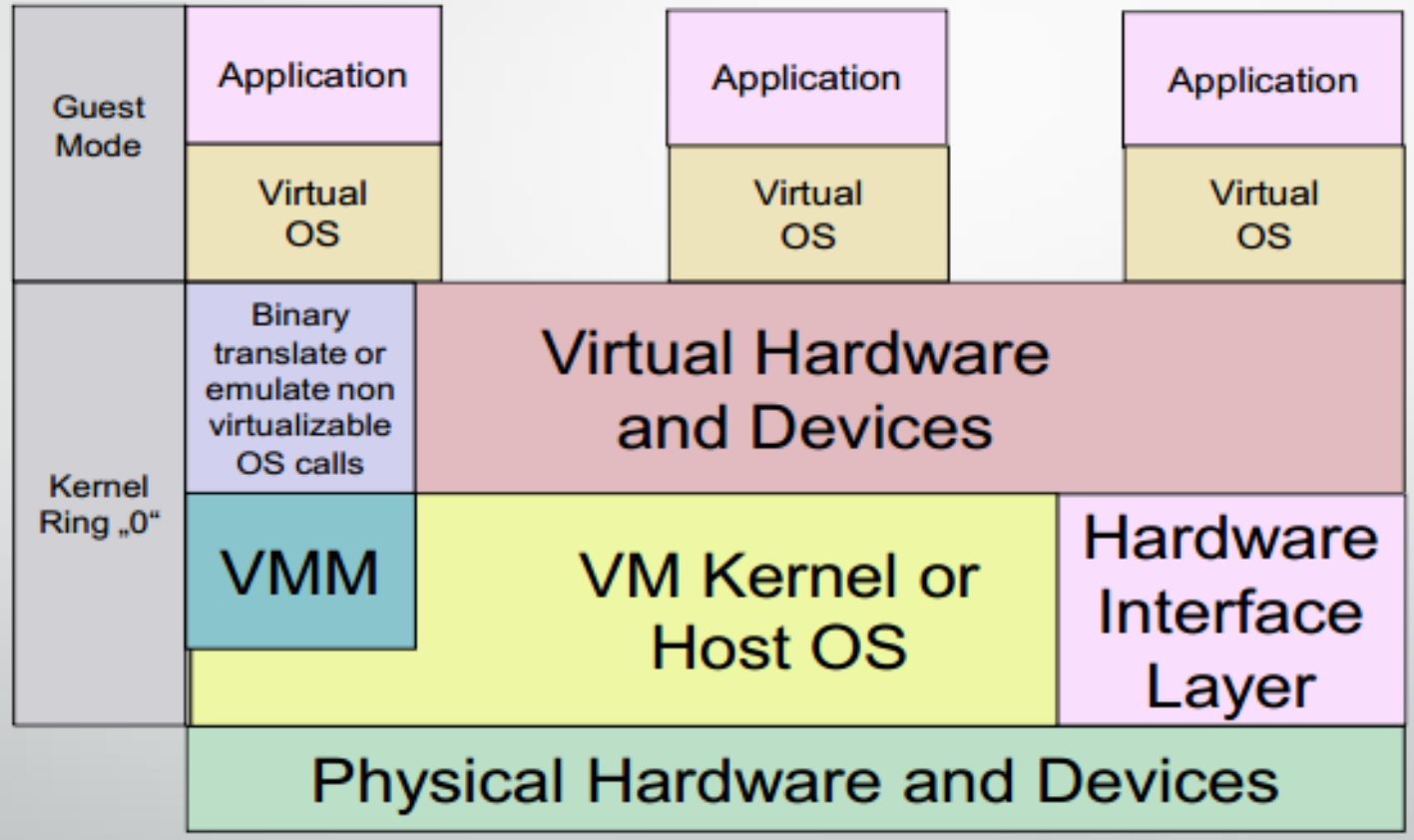
- **Szoftveres virtualizáció**
  - (Trap and Emulate + bináris fordítás)
- **Paravirtualizáció**
  - (vendég OS kódját módosítjuk, hogy tudjon a virtualizációról, és hívhassa a VMM-et)
- **Hardveres virtualizáció**
  - (Trap and Emulate, teljesen hardveres támogatással)

Hardveres Virtualizáció

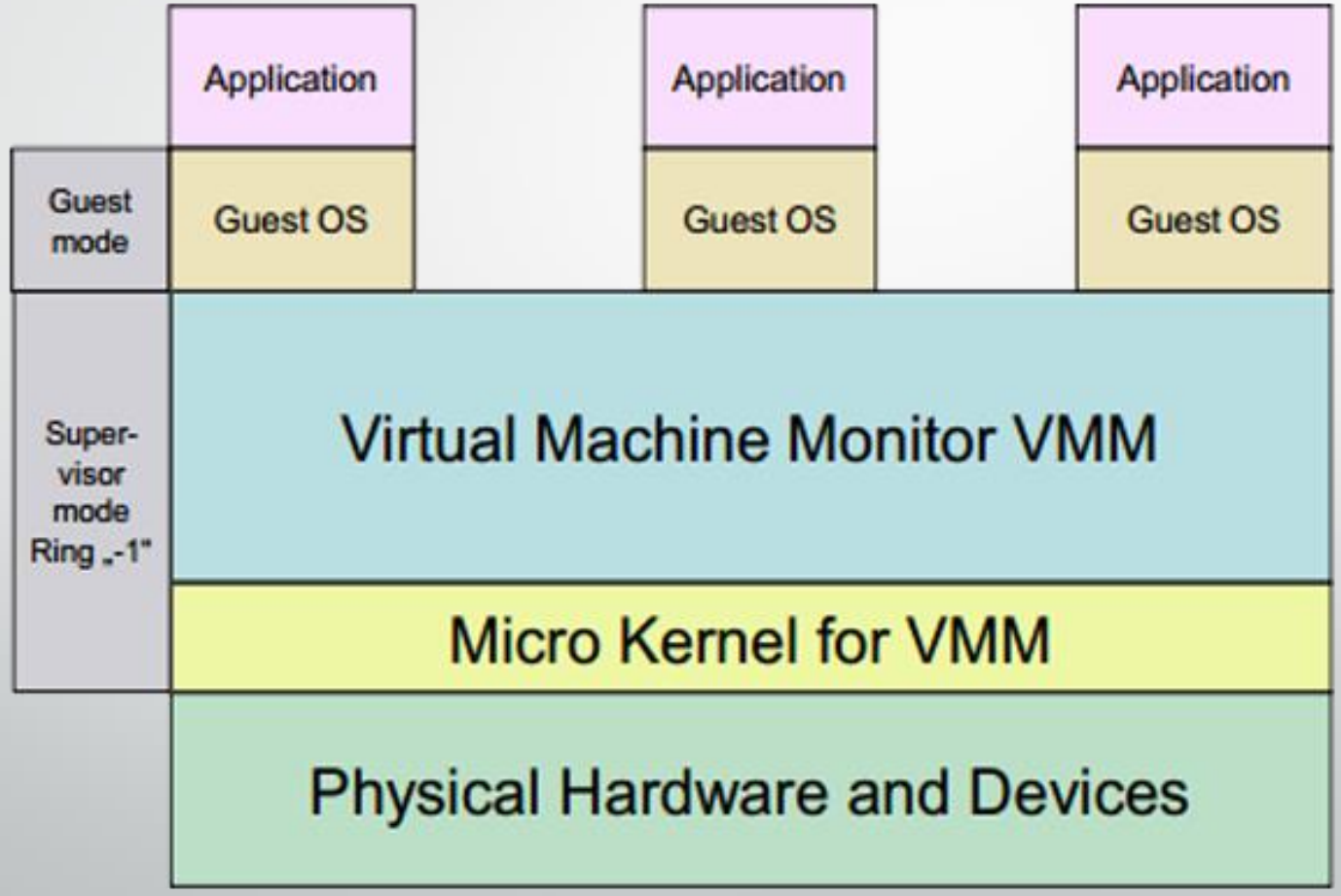


- Intel VT-x és AMD-V kiegészítésekkel új mód jött be (root/non-root mode)
- SYSCALL és INT Ring3-ból Ringo-ba hív át – nem kell felesleges kört futnia a VMM-ben
- Van külön VMCALL utasítás is, amivel ki lehet hívni a Root Mode-ba
- Minden szükséges privilegizált utasítást elkap
- A binary translation sok optimalizációra adott lehetőséget, ami itt hiányzik
- Lassabb volt a szoftveres virtualizációnál (de javul, a VMCALL-VMRESUME körülfordulási időt minden CPU generációval csökkentik)

Hardveres virtualizáció bináris fordítással



Hardveres virtualizáció CPU támogatással



Melyik a legjobb/leggyorsabb módszer?

- Folyamatosan változik a válasz
  - környezettől, terheléstől is függ
  - HW virtualizáció kezdetben kiforratlanabb, mint a BT
- Megoldások több módszert használnak vegyesen

	VMware ESX/ESXi	Microsoft Hyper-V	Xen
Szoftveres (BT)	+	-	-
Paravirtualizáció	- (már nem)	+ (részben)	+
Hardveres (Intel VT-x, AMD-V)	+	+	+