Machinevirtualisatie

Raphael 'kena' Poss
Universiteit van Amsterdam

Besturingsystemen

VANDAAG...

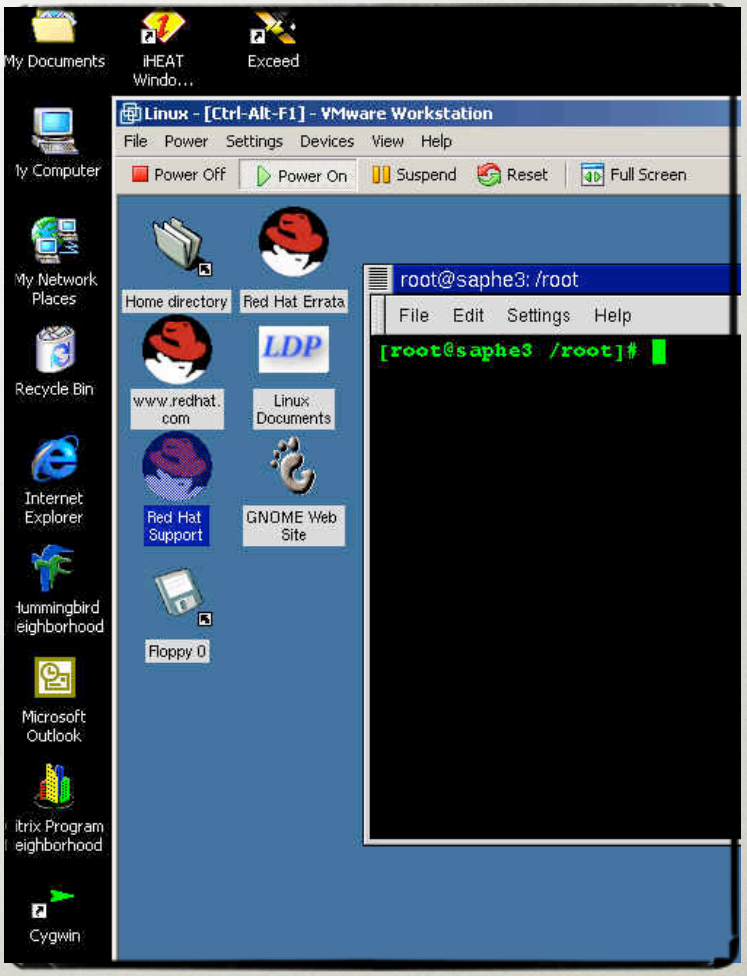
- zal je leren:
  - wat machinevirtualisatie is
  - waarvoor het wordt gebruikt
  - hoe het werkt
  - het verschil tussen VMWare, QEMU, KVM, Xen en anderen.

Kortom

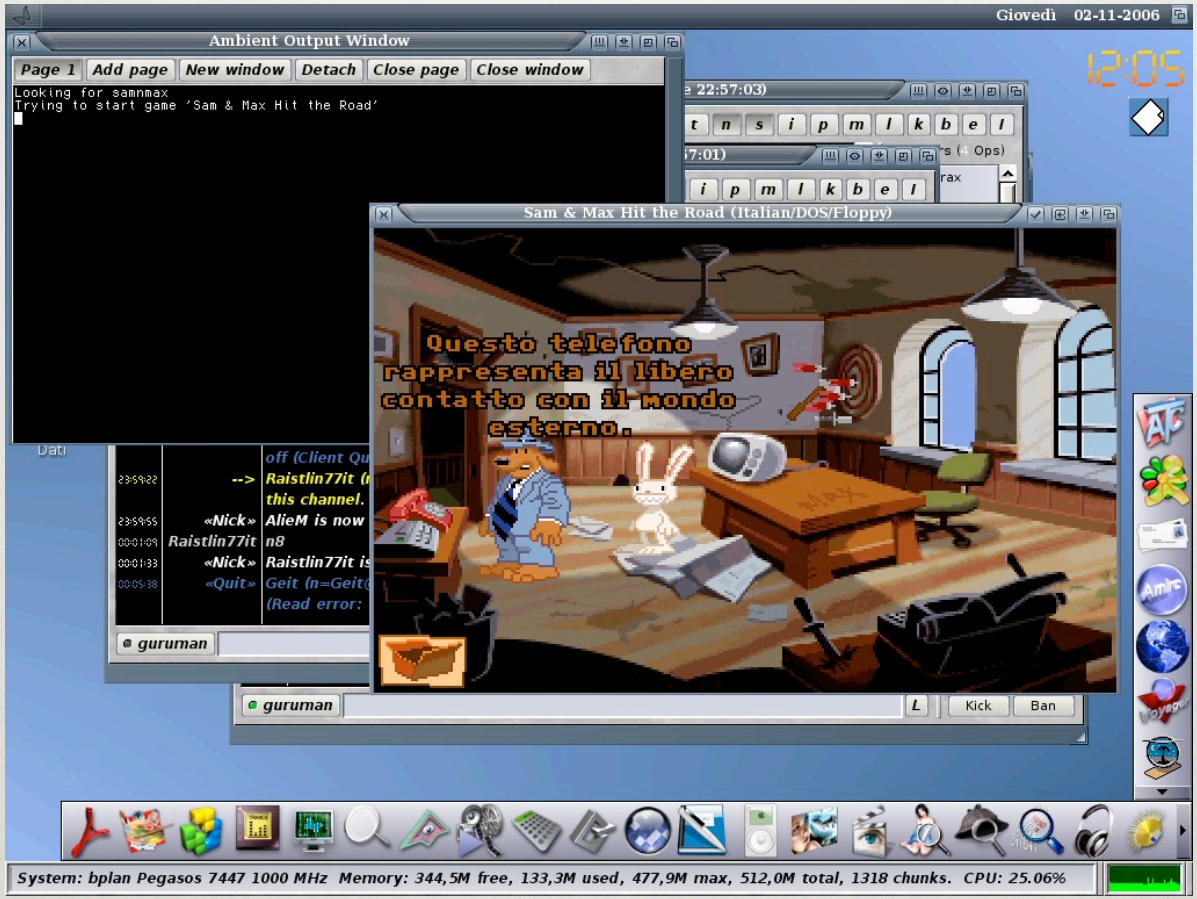
Machinevirtualisatie vindt plaats als *de toegang tot echte hardware door een "guest" OS wordt beheerd door een "host" OS*.

Oftewel:

als **een OS "binnen" een andere** draait.



Virtualisatie vs. Emulatie



Voorbeeld: SCUMMVM om "oude spelletjes" te spelen op moderne hardware: is dit virtualisatie?

Machinevirtualisatie vs. Emulatie

- In dit college:
  - Emulatie: een simulatieprogramma interpreteert de hele code van de guest software - vaak 100% trager of meer
  - Virtualisatie: deel van de code van de guest software draait direct op de host processor - vaak 50% trager of minder
- **Pas op: het woord "virtualisatie" wordt soms gebruikt voor beide in de literatuur.** Dit college gaat alleen over het 2de begrip

Machinevirtualisatie vs. Emulatie Voorbeelden:

- Virtualisatie:
  - **VMWare**
  - **Xen**
  - **KVM**
  - z/VM
  - KQEMU
- Emulatie:
  - VirtualPC
  - SCUMMVM
  - UAE
  - DOSBox
  - **QEMU**

Machinevirtualisatie: Waarom?

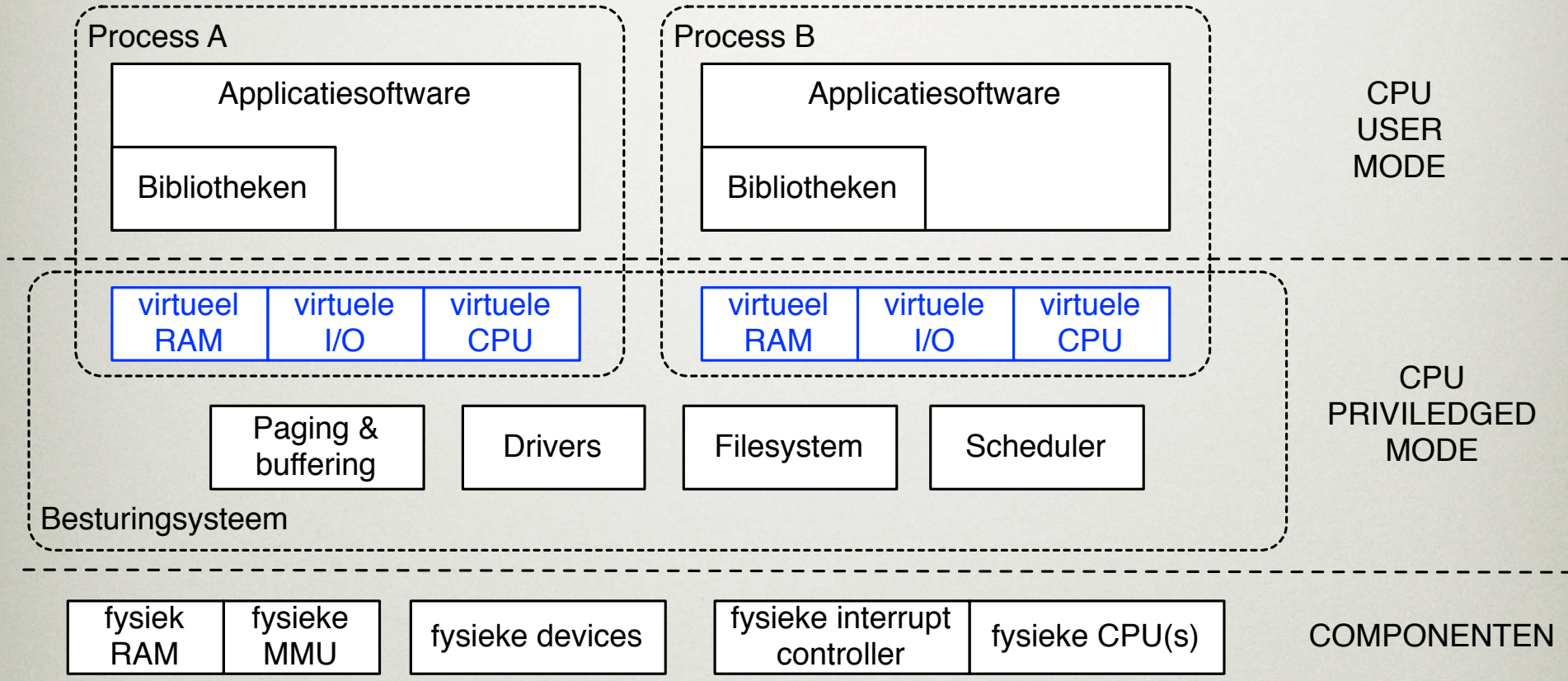
- Oorspronkelijk (IBM, jaren '60): als alternatieve **architectuur** voor besturingsystemen
- Heden:
  - **Serverbeheer** in datacenters / clouds
  - **Hardware delen** tussen meerdere OS
  - **Debug & testen** van systeemsoftware

Geschiedenis

- De “IBM jaren” - **alleen IBM werkt serieus op virtualisatie**
  - 1964: IBM, CP40 dan CP67
  - 1972: IBM, VM370
  - 1974: Popek&Goldberg, formalisatie “virtualiseerbaarheid”
- De “lange niks” **- IBM steeds bezig maar minder, PC markt groeit**
  - 1987: Intel, virt. i8086 op i80386
- De “datacenter jaren” **- Internet + goedkope CPUs + RAM**
  - 1999-2001: VMware
  - 2000: IBM, z/VM
  - 2003: Xen
  - 2005: HP Integrity
  - 2007: VirtualBox, Linux KVM

Architectuur virtuele machines

ARCHITECTUUR
"GEWONE OS"



Voorbeeld: Linux

Virtualisatie vs. Machinevirtualisatie

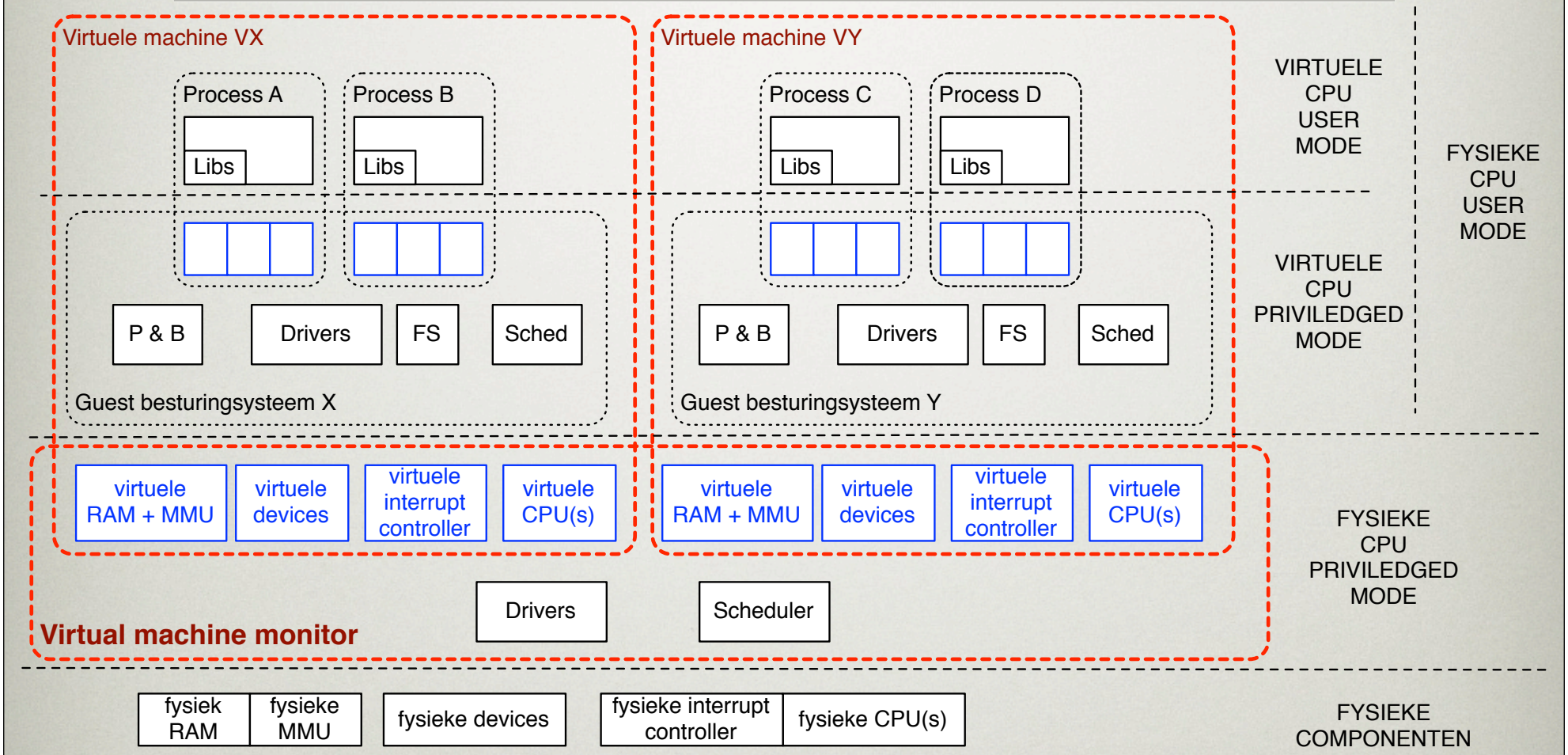
- time sharing
  = virt. van “eenvoudige” processor
- paging
  = virt. van RAM adressen
- processen (bvb Unix)
  = virt. simpele CPU
  + RAM adressen
  + eenvoudige I/O

- Machinevirtualisatie
  = virt. volle CPU(s)
  + volle RAM
  + volle I/O devices

Omdat CPU, RAM and I/O tegelijk worden virtualiseerd, kan een geheel OS draaien als guest software.

Architectuur

Virtual machine monitor

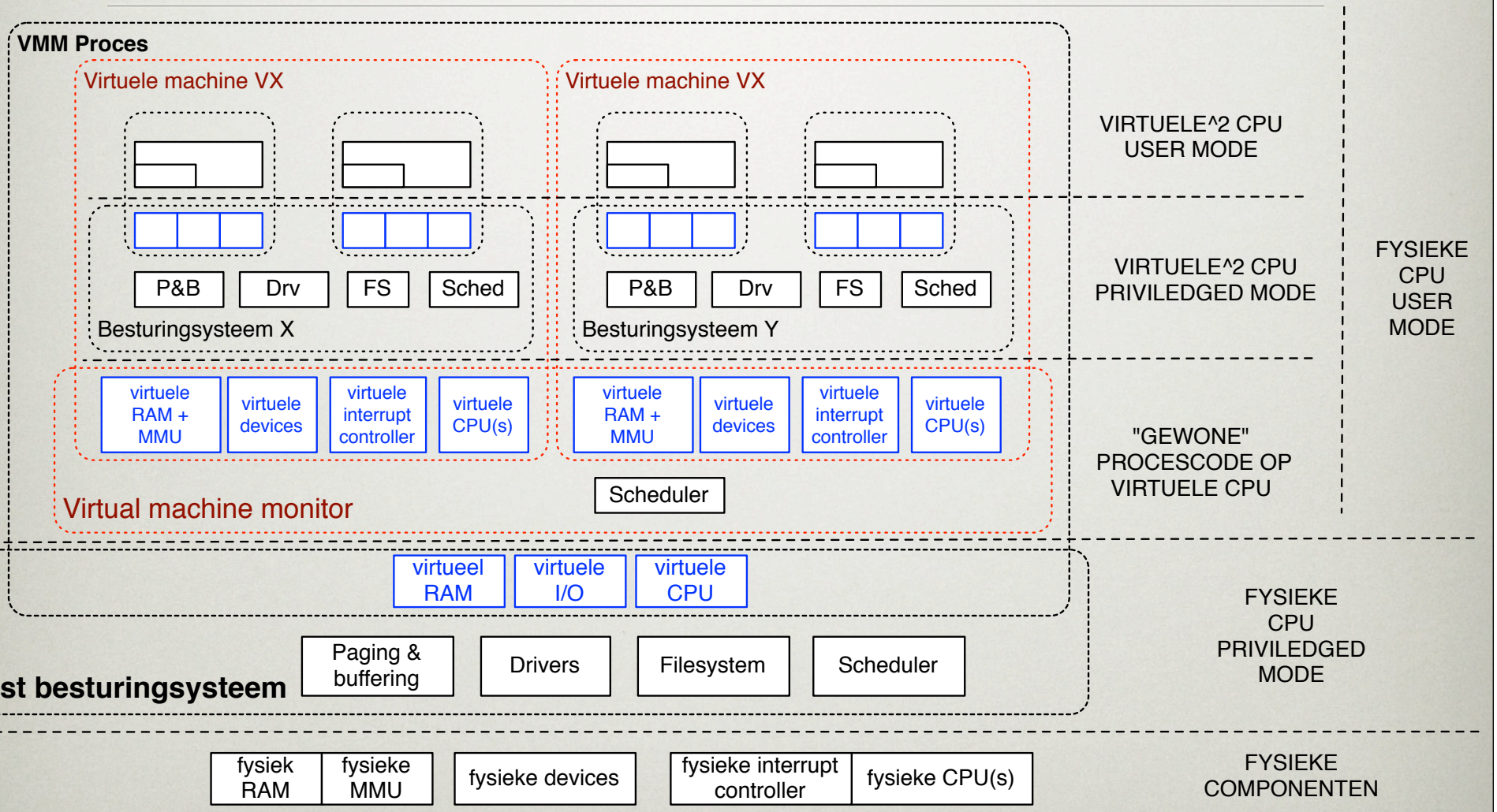


Voorbeeld: VMWare server

Virtual machine monitors, of "hypervisor"

- Rol van VMM / hypervisor:
  - deelt CPU tijd tussen VMs
  - verdeelt fysieke RAM
  - emuleert MMU, interrupt controller, I/O device controllers
- "Native virtualisatie":
  Fysieke hardware beheerd direct door VMM
  - Onhandig te installeren in bestaande omgeving
- **Embedded virtualisatie**: VMM als gastproces

Architectuur Embedded VMM



Voorbeeld: VirtualBox, VMWare Workstation

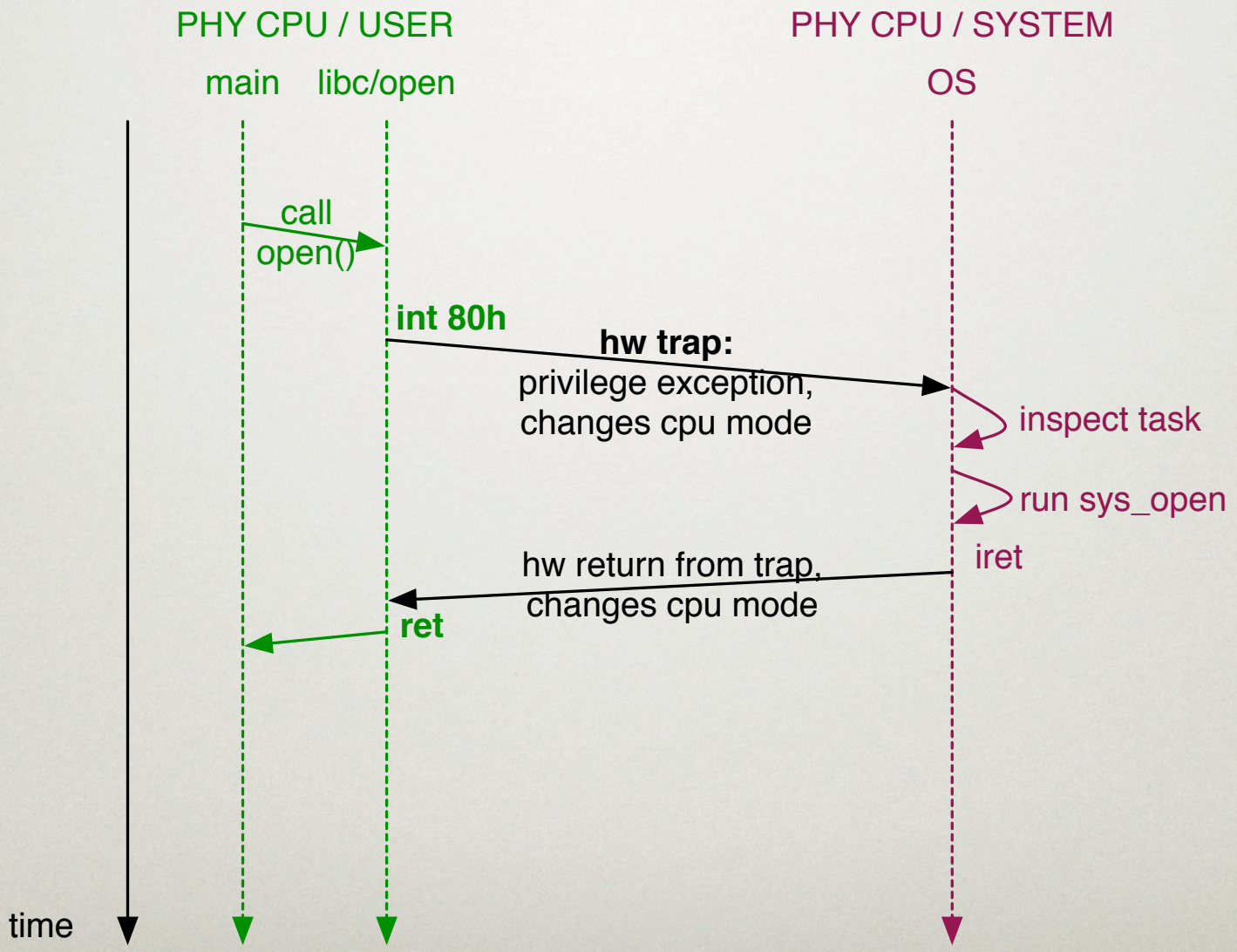
Hoe het werkt: processor virtualisatie

Achtergrondkennis

CPU executie niveau	"**Unprivileged**" instructies bvb. add, call	"**Privileged**" instructies bvb. cli, lldt	**Fouten** bvb. puts(NULL)
User	gewoon uitgevoerd	trap naar "privilege fout" in systeem modus, eventueel vertaald naar SIGILL	trap naar fouthandler in systeem modus, eventueel vertaald naar SIGBUS/SIGSEGV
Systeem	gewoon uitgevoerd	gewoon uitgevoerd	Hardware stop

Unix en moderne besturingsystemen:
"User" voor taken / processen, "Systeem" voor OS

Voorbeeld: Linux / x86

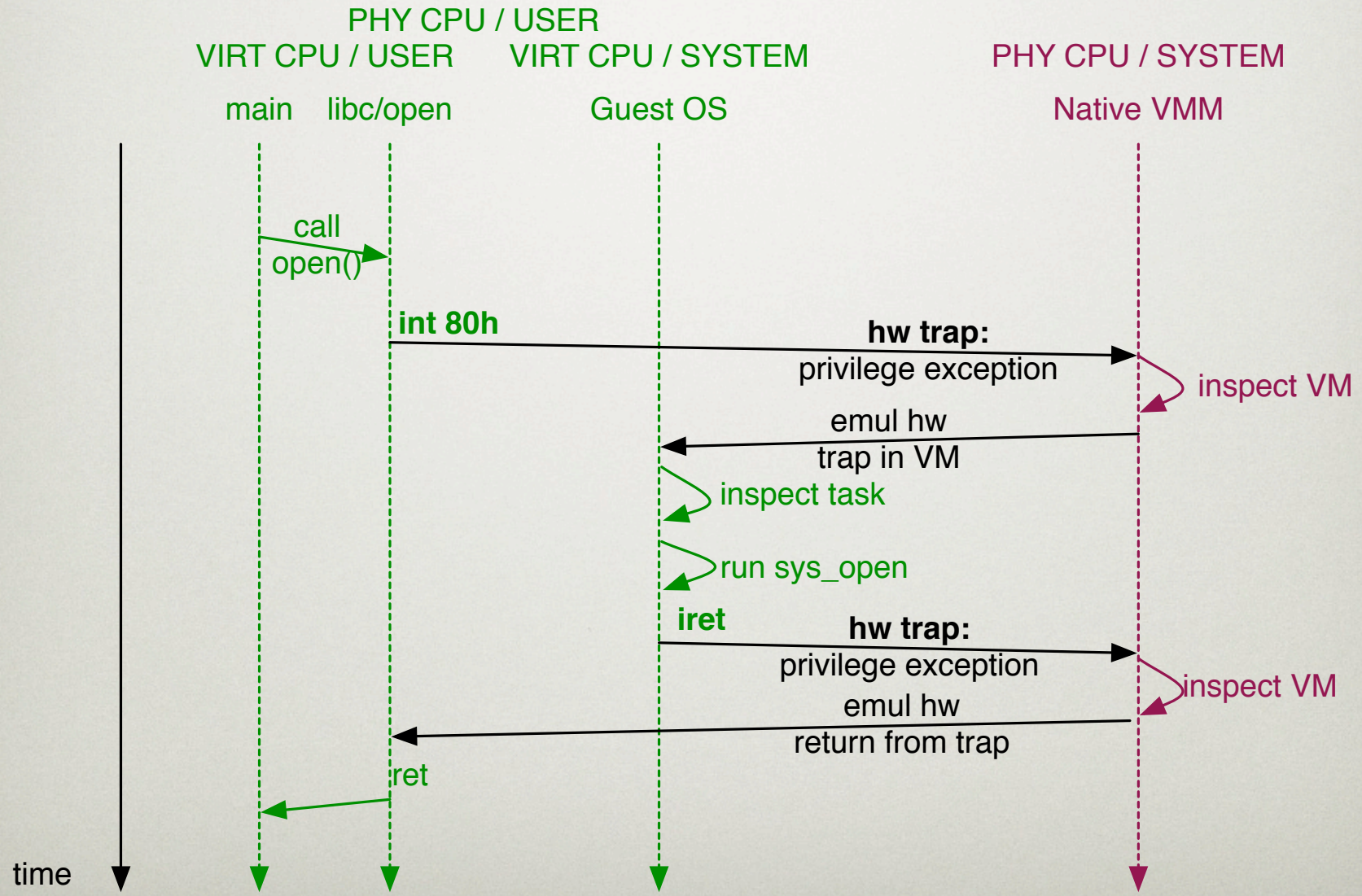


Met native VMM

Host CPU executieniveau	Virt. CPU executieniveau	"**Unprivileged**" instructies bvb. add, call	"**Privileged**" instructies bvb. cli, lldt	**Fouten** bvb. puts(NULL)
User	**VM/User**	gewoon uitgevoerd	trap naar Systeem, dan emulatie trap naar VM/Sys modus door VMM	trap naar Systeem, dan emulatie trap naar fouthandler in VM/Sys modus door VMM
User	**VM/Systeem**	gewoon uitgevoerd	trap naar Systeem, dan emulatie privileged instructie door VMM	trap naar Systeem, dan emulatie trap naar fouthandler in VM/Sys modus door VMM
Systeem (VMM)		gewoon uitgevoerd	gewoon uitgevoerd	Hardware stop

Zo werkt het met IBM CP-67, z/OS, VMWare server, etc.

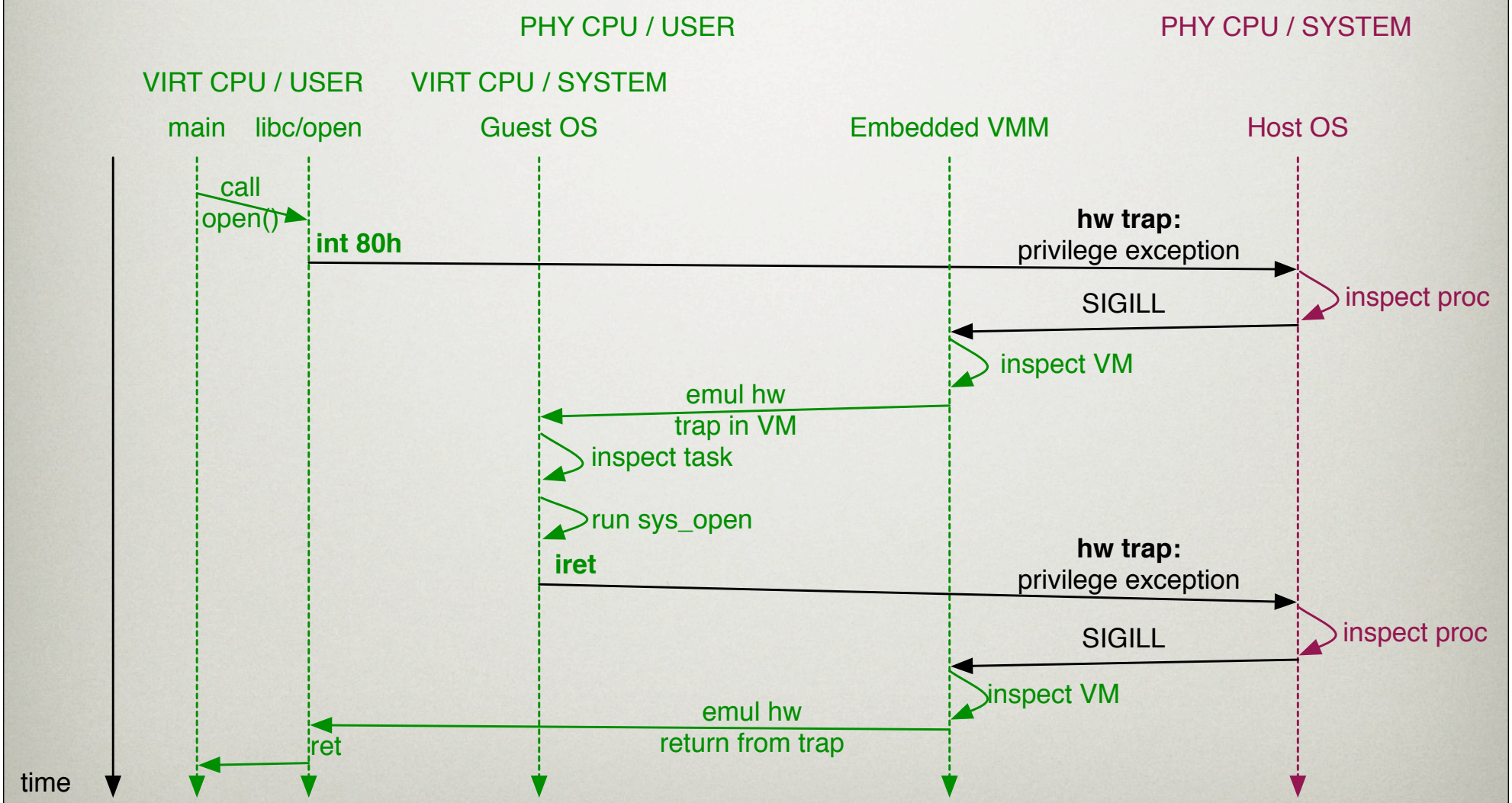
Voorbeeld: Linux / x86



Met embedded VMM + "Goede" CPU hardware

Host CPU executieniveau	Virt. CPU executieniveau	"Unprivileged" instructies bvb. add, call	"Privileged" instructies bvb. cli, lldt	**Fouten** bvb. puts(NULL)
User	**VM/User**	gewoon uitgevoerd	trap naar Host OS, VMM krijgt SIGILL, dan emulatie trap naar fouthandler VM/Sys	trap naar Host OS, VMM krijgt SIGxxx, dan emulatie trap naar fouthandler VM/Sys
User	**VM/Systeem**	gewoon uitgevoerd	trap naar Host OS, VMM krijgt SIGILL, dan emulatie instructie door VMM	trap naar Host OS, VMM krijgt SIGxxx, dan emulatie trap naar fouthandler VM/Sys
User (VMM)		gewoon uitgevoerd	(NvT)	trap naar Host OS, VMM proces stopt
Systeem (Host OS)		gewoon uitgevoerd	gewoon uitgevoerd	Hardware stop

Voorbeeld: Linux / x86



"Goede" CPU hardware Wat is dat?

- Trap-gebaseerde virtualisatie werkt alleen als alle "gevoelige" instructies een trap naar systeemmodus veroorzaken
  - Dit werd geformaliseerd door Popek & Goldberg in 1974: http://dx.doi.org/10.1145/361011.361073
  - Wél het geval voor alle IBM en Sun processoren
- Fun fact: **tot 2005** (AMD-V, Intel VT-x), **was dit niet mogelijk voor x86 processoren!**
  - Eerste versie VMWare workstation in 2001: Hoe kan dat?

VMWare octrooi 6397242, 1998

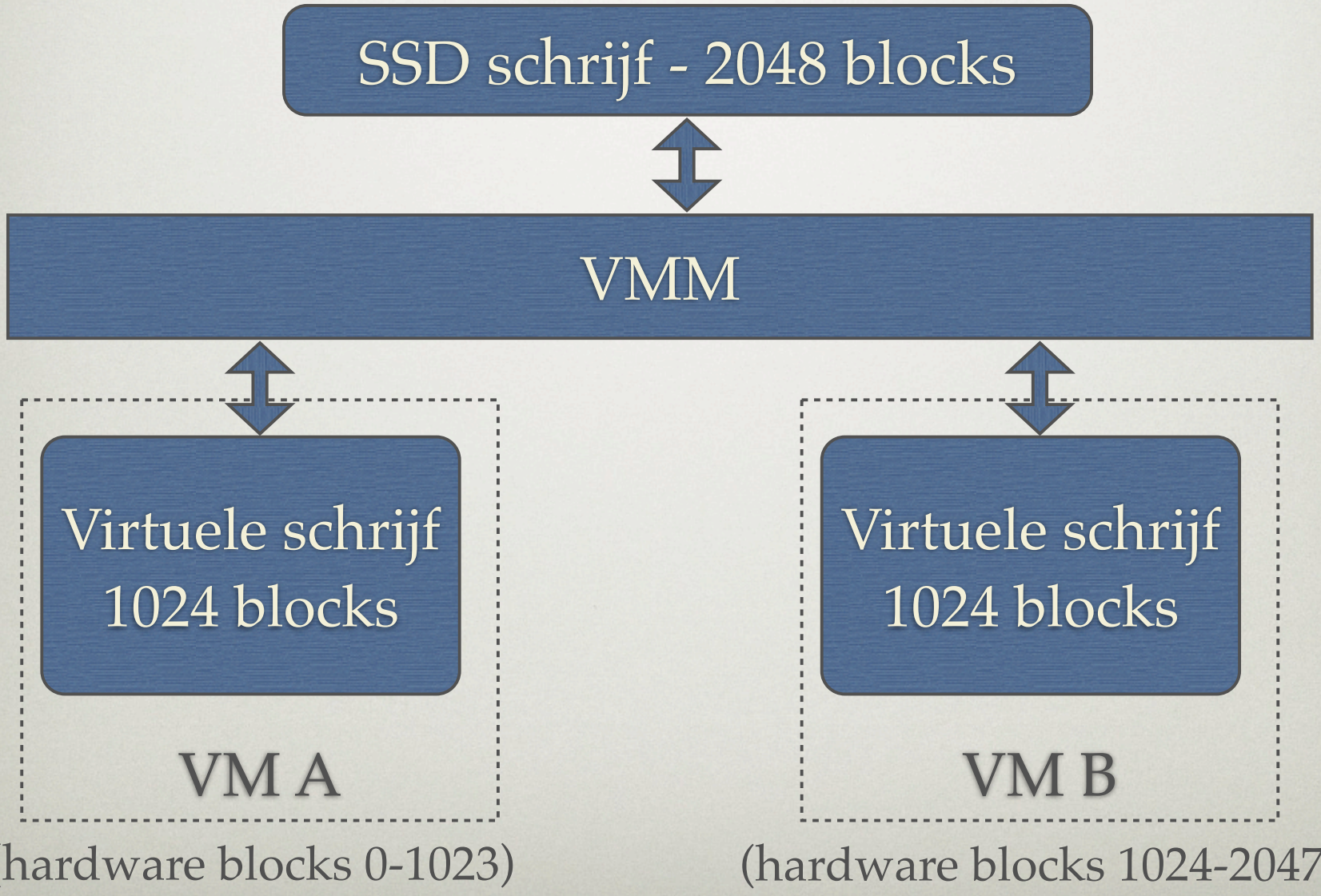
Host CPU executieniveau	Virt. CPU executieniveau	"Unprivileged" instructies bvb. add, call	"Privileged" instructies bvb. cli, lldt	Fouten bvb. puts(NULL)
User	**VM/User**	gewoon uitgevoerd	gedetecteed vóór executie en vertaald naar emulatiecode door JIT	trap naar Systeem, VMM krijgt signaal, dan emulatie trap in VM/Sys modus
User	**VM/Systeem**	gewoon uitgevoerd	gedetecteed vóór executie en vertaald naar emulatiecode door JIT	trap naar Systeem, VMM krijgt signaal, dan emulatie trap in VM/Sys modus
User (VMM)		gewoon uitgevoerd	(NvT)	trap naar fouthandler in Host OS
Systeem (Host OS)		gewoon uitgevoerd	gewoon uitgevoerd	Hardware stop

VMWare 1998-2005 analyseert de x86 machinecode voordat het wordt uitgevoerd, en vertaalt het gedeeltelijk: **privileged instructies worden geëmuleerd**

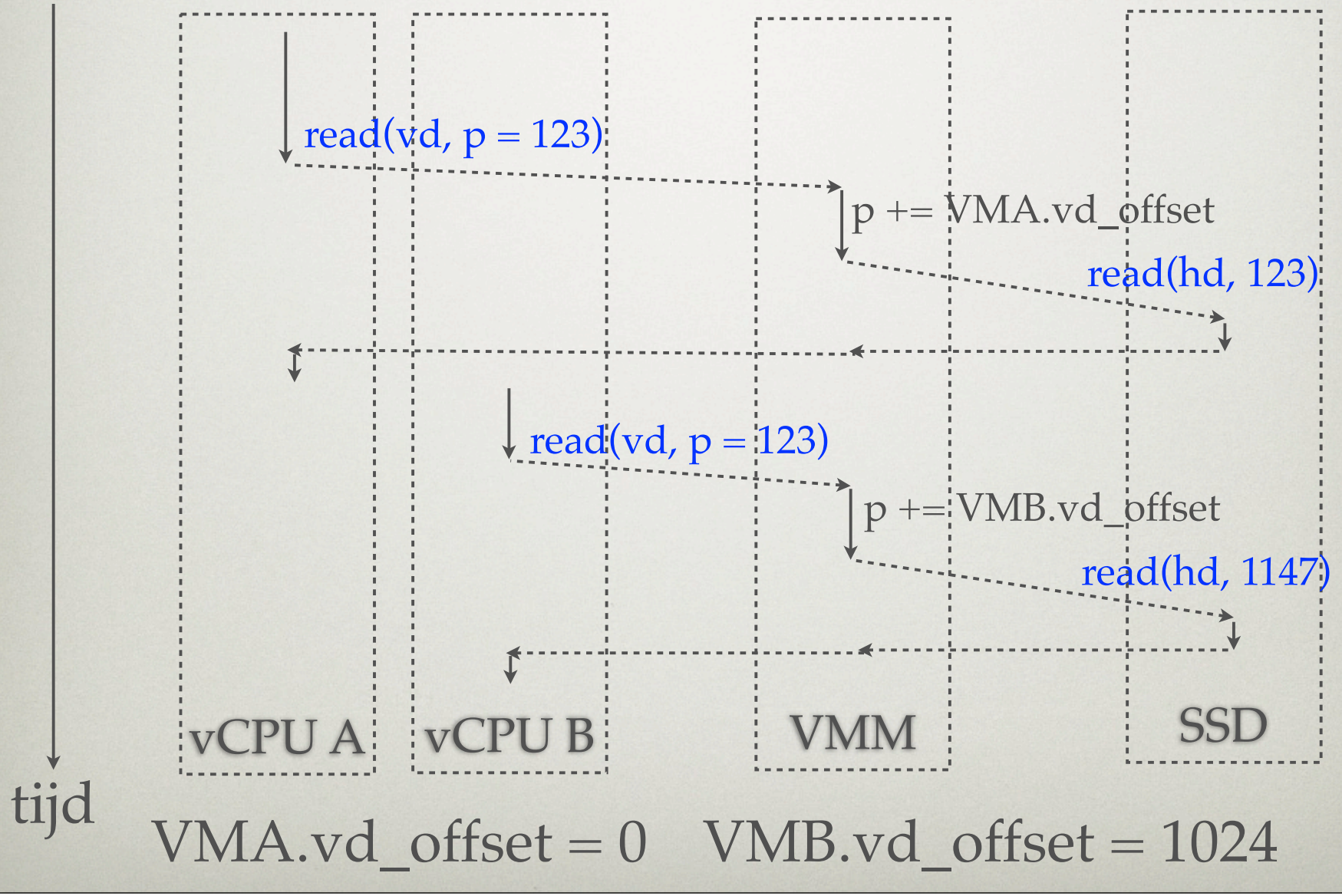
Dit is niet meer nodig met AMD-V en VT-x sinds 2005-2006!

Hoe het werkt: I/O virtualisatie

Voorbeeld


SSD schrijf - 2048 blocks
VMM
Virtuele schrijf
1024 blocks
Virtuele schrijf
1024 blocks
VM A
VM B
(hardware blocks 0-1023)
(hardware blocks 1024-2047)

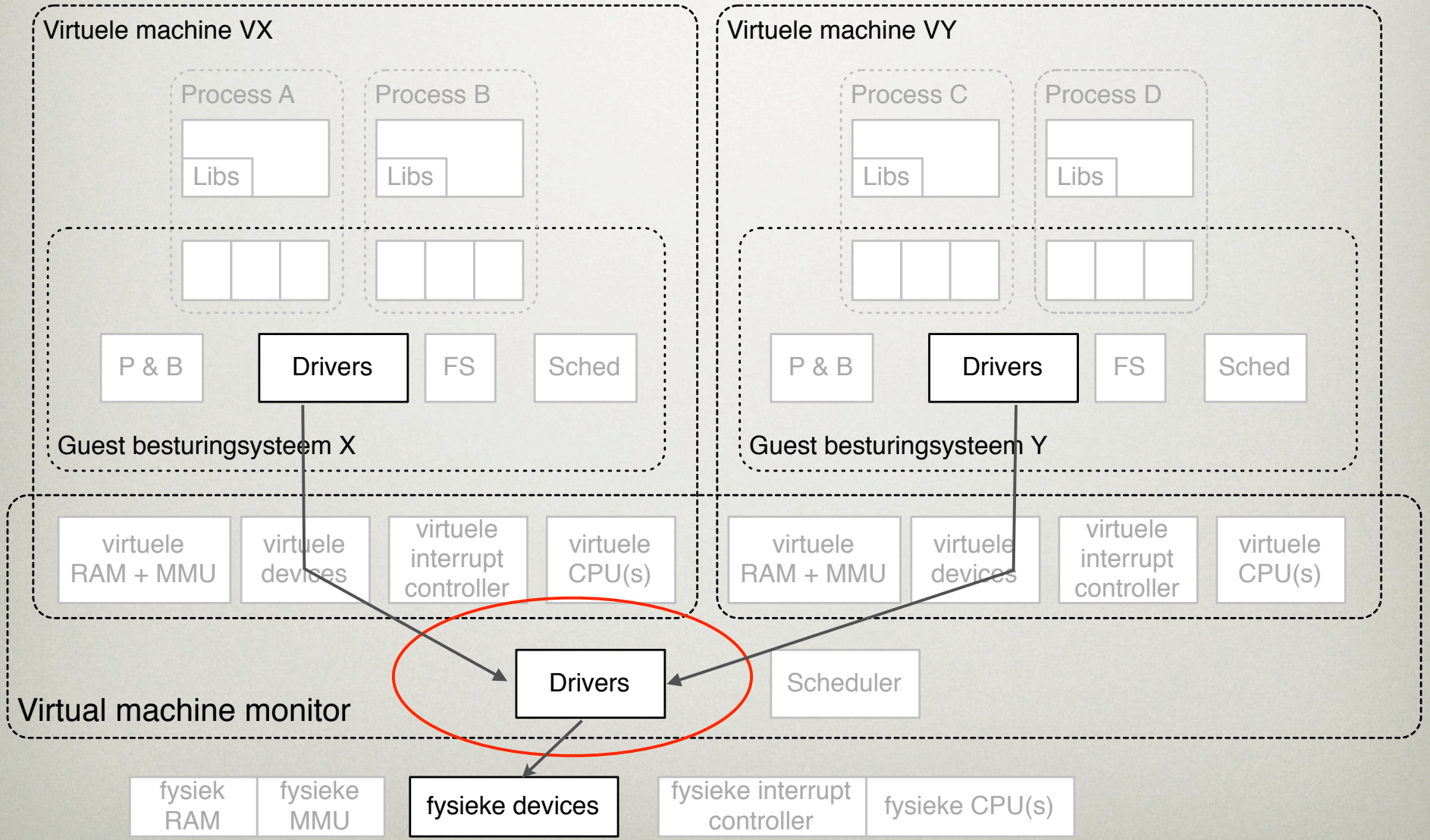
Voorbeeld


read(vd, p = 123)
p += VMA.vd_offset
read(hd, 123)
read(vd, p = 123)
p += VMB.vd_offset
read(hd, 1147)
vCPU A
vCPU B
VMM
SSD
tijd
VMA.vd_offset = 0
VMB.vd_offset = 1024

Algemeen principe: I/O emulatie

- Guest OS plaatst HW opdracht naar virtuele device
  - Kan door MMIO of I/O instructies
  - Privileged, dus trap naar VMM
- VMM ontvangt opdracht, dan vertaalt offsets / adressen
- Opdracht na vertaling wordt verstuurd naar host device
- Foutmeldingen/resultaten vanuit host device ook vertaald richting guest OS

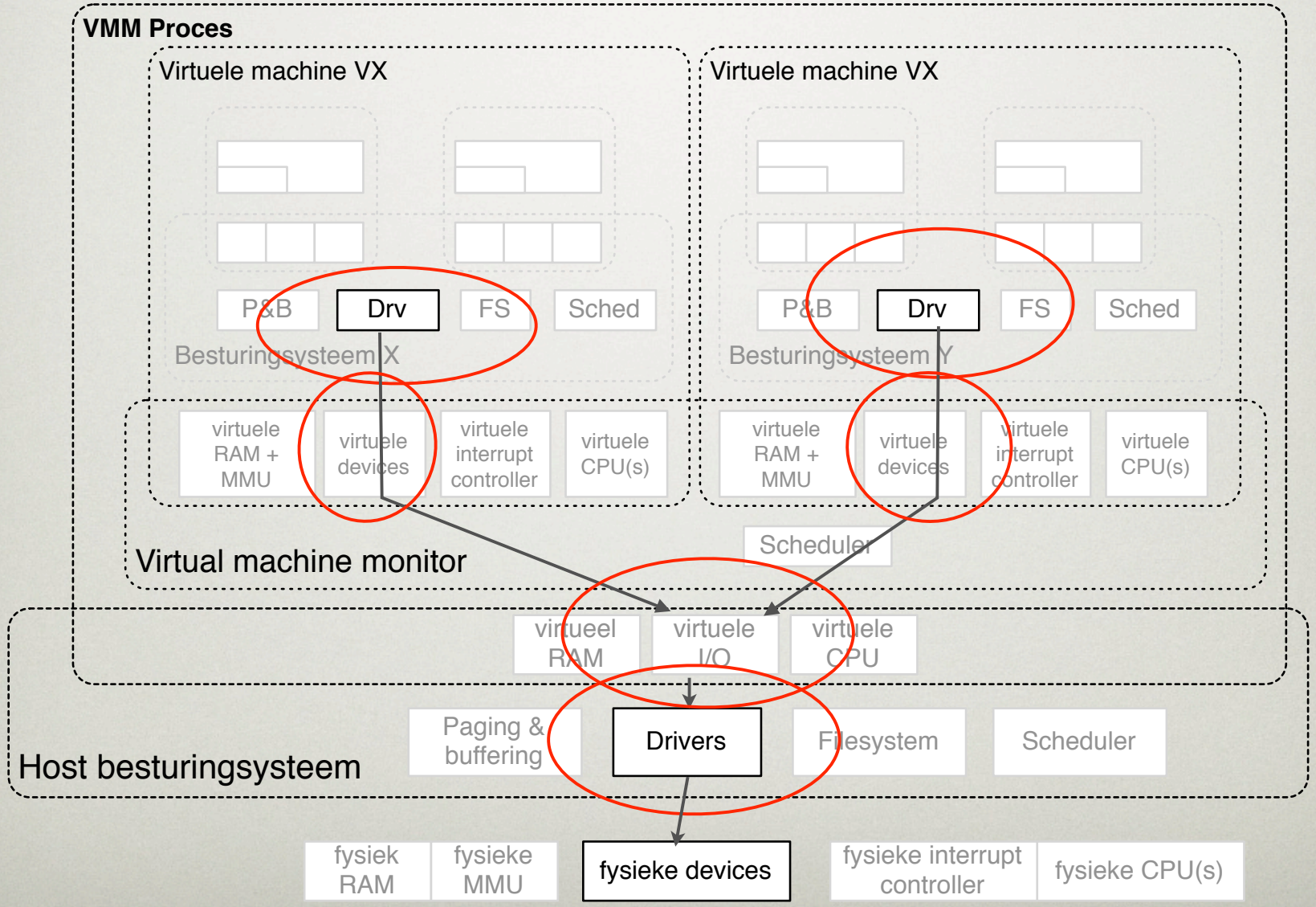
Indirectie door abstractielagen



Probleem: waar komen de drivers vandaan?

- HW Fabrikanten leveren drivers voor “gewone OS”, niet VMM
- VMM dus beperkt tot hardware waarvoor VMM ontwikkelaar drivers al heeft
- Minder flexibiliteit op devices dus hogere kosten - meestal zichtbaar in datacenters
- Dé reden waarom embedded VMMs populairder dan native zijn geworden

Kost: drivers, drivers en meer drivers (emb.)



Probleem: Abstractiekosten

- I/O virtualisatie betekent emulatie
  - opdrachten geïnterpreteerd: trager
  - meer virtualisatielagen: trager-er
- Datacenters zijn gevoelig voor I/O prestatie!
- Kan dit beter?
  - Ja, door intermediaire emulatielagen te verwijderen
- Dit leidt tot **para-virtualisatie**

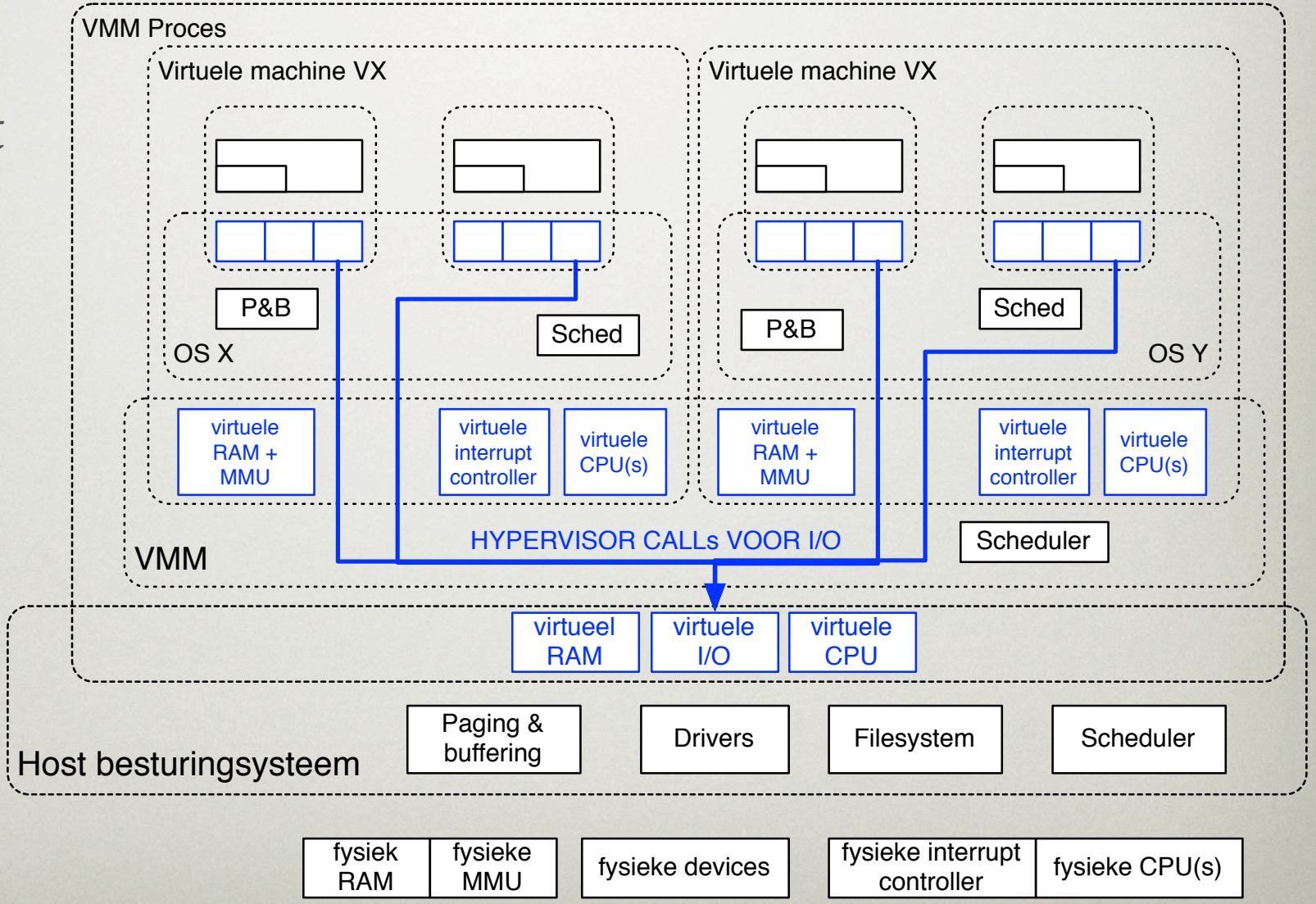
Hoe het werkt: para-virtualisatie

Para-virtualisatie, Kortom

- Gemotiveerd door betere I/O prestaties
- Breekt het concept van virtualisatie: guest OS “weet” dat er een host OS is
- Host OS biedt diensten (bvb HW drivers) direct aan guest OS
- Gevirtualiseerde I/O in guest processen “praten” direct tegen host driver

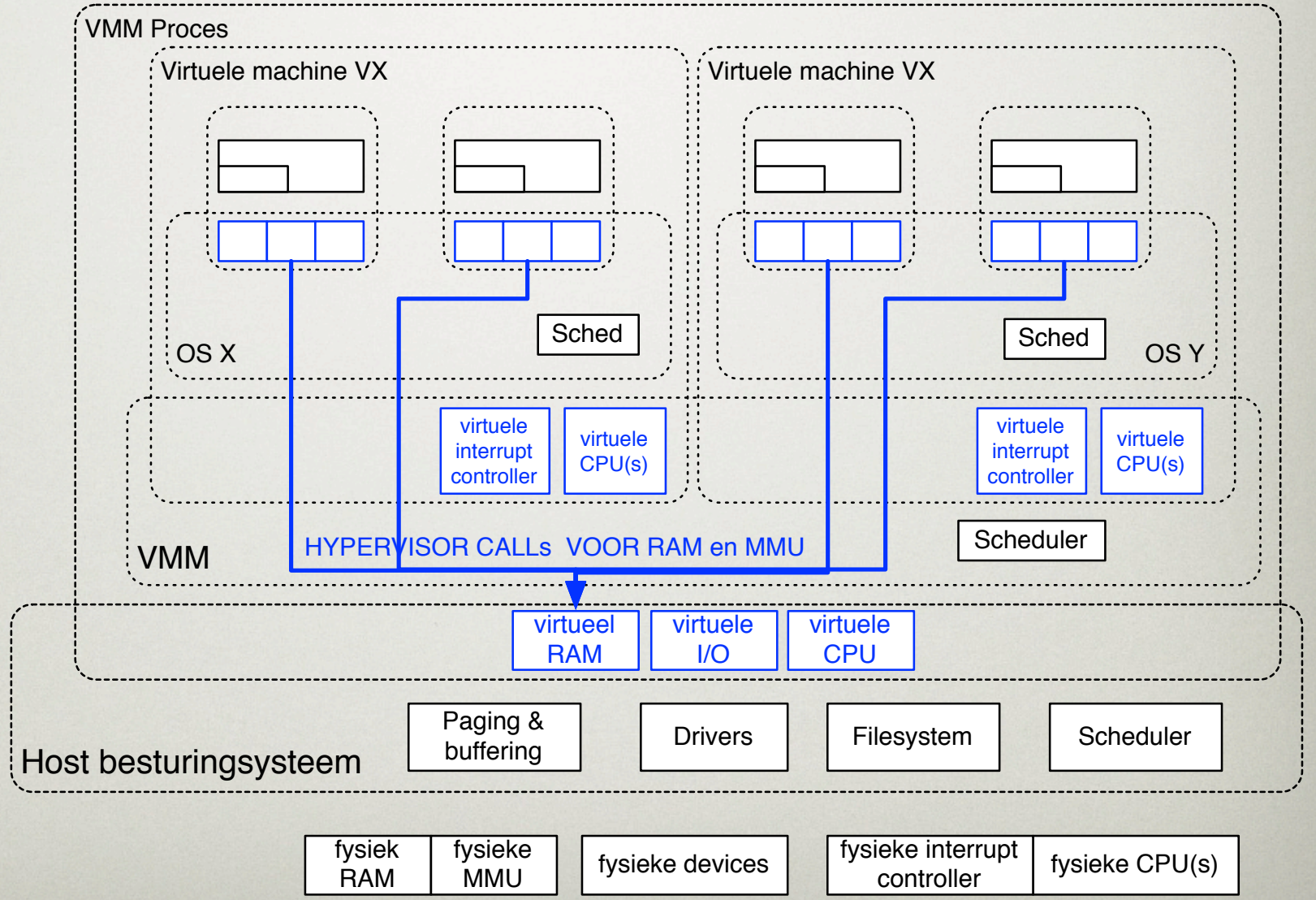
Architectuur para-virtualisatie (1)

Bespaart guest FS & drivers



Architectuur para-virtualisatie (2)

Bespaart guest paging & buffering ook!



Virtuele omgeving: voorbeeld I/O devices

- Volle virtualisatie:
  - Harde schrijf: PIIX3 SATA, geëmuleerd
  - Netwerk: RealTek 8139, geëmuleerd
- Para-virtualisatie:
  - Harde schrijf: “virtio”, interface via hypercall
  - Netwerk: “virtio”, interface via hypercall

Wat hebben we vandaag geleerd

Deze tabel begrijpen: Huidige Technologie

	Native "full" VMM	Native para-virtualisatie	Embedded "full" VMM	Embedded para-virtualisatie
VMWare	ja	ja	ja	ja
z/VM	ja	ja	nee*	nee*
VirtualBox	nee	nee	ja	ja (VBox exts)
Linux KVM	nee	nee	ja (QEMU I/O)	ja
Xen	nee	nee	nee	ja
Linux containers	nee	nee	nee	ja
KQEMU	nee	nee	ja	nee

De volgende vragen beantwoorden:

- Verschil virtualisatie / emulatie?
- Vooreis “virtualiseerbaarheid”?
- Rol VMM / hypervisor?
- Verschil “native” vs “embedded” VMM?
- Hoe werkt CPU virtualisatie?
- Hoe werkt I/O virtualisatie?
- Waarom para-virtualisatie? Hoe?
- Voordelen / nadelen “hypervisor call”?
- Voordelen / nadelen “native” VMM vs. “embedded” VMM?

Om verder te gaan

- T.W. Doeppner. *Operating systems in depth*, Sectie 4.2. Wiley publishing. 2011
- R. J. Creasy, *The origin of the VM/370 time-sharing system*, IBM Journal of Research & Development, Vol. 25, No. 5 (September 1981), pp. 483–90 http://pages.cs.wisc.edu/~stjones/proj/vm_reading/ibmrd2505M.pdf
- G.J. Popek and P. Goldberg. F*ormal requirements for virtualizable third generation architectures*. Commun. ACM, 17(7):412–421, 1974. http://dl.acm.org/citation.cfm?id=361073