Les 9: Virtualisatie

All problems in computer science can be solved
by another level of indirection...
— David Wheeler

… but that usually will create another problem.
— David Wheeler

best9-1

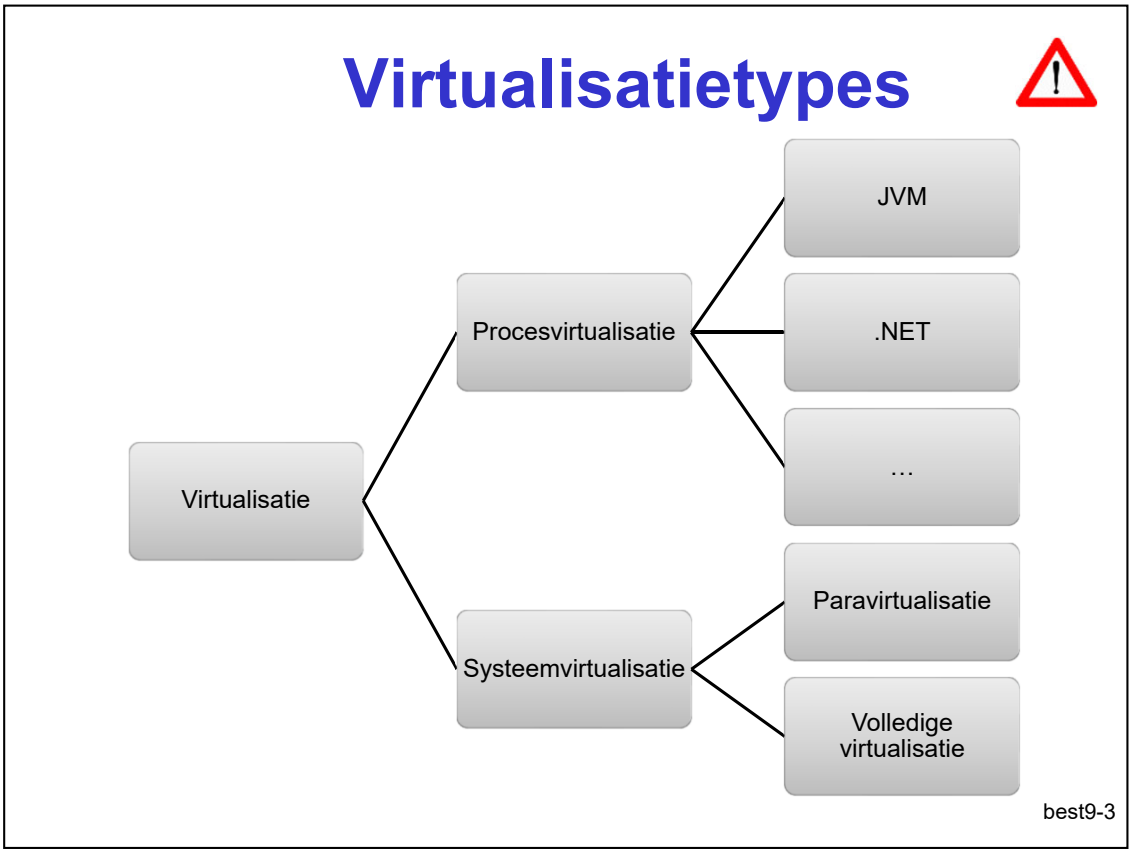
In deze cursus zijn er al tal van abstracties en virtualisaties aan bod gekomen. In dit hoofdstuk gaan we de ultieme virtualisatie bespreken: de virtualisatie van een compleet computersysteem. Virtualisatie van een volledig systeem bestond reeds in de jaren 60 op mainframecomputers. Sinds de opkomst van de personal computers en de werkstations in de jaren tachtig van de vorige eeuw was virtualisatie wat in de vergetelheid geraakt. Eén van de redenen was wellicht dat de systemen te weinig systeemmiddelen aan boord hadden om meer dan één virtuele machine op een aanvaardbare manier te laten werken. Een tweede reden was ongetwijfeld dat velen dachten dat de Intel-architectuur niet virtualiseerbaar was. In 1998 toonde VMware aan dat dit wel mogelijk was.

Overzicht

- Definities
- Klassieke virtualiseerbaarheid
- Virtualisatie
  - CVE
  - Geheugen
  - IO
- Migratie

best9-2

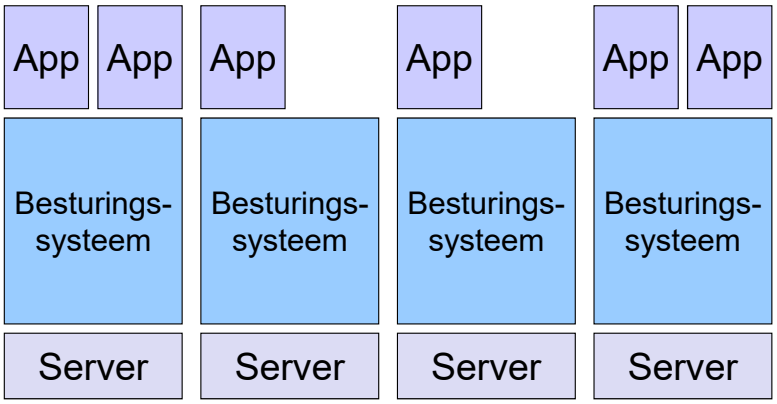
Bij het virtualiseren van een computersysteem zijn er drie componenten die gevirtualiseerd moeten worden: de CVE, het geheugen en het IO-systeem.



Vooraleer ons in de details van virtualisatie te verdiepen, is het goed om duidelijk te maken dat er twee soorten van virtualisatie zijn: procesvirtualisatie en systeemvirtualisatie. Bij procesvirtualisatie gaat het over het uitvoeren van een virtuele machine zoals JVM of .NET. Het is dus de implementatie van een virtuele computerarchitectuur. Procesvirtualisatie biedt vaak ook een API aan om andere diensten van het onderliggende besturingssysteem te gebruiken. De bytecode die uitgevoerd wordt, wordt hierdoor gemakkelijker porteerbaar.

Bij systeemvirtualisatie virtualiseert men het computersysteem en biedt men een softwareabstractie van de hardware aan. Men simuleert als het ware het computersysteem. Als de gesimuleerde architectuur dezelfde is als de architectuur waarop ze uitgevoerd wordt, dan zijn er veel mogelijkheden tot optimalisatie. Zo kan men in dat geval de gewone instructies uitvoeren op de processor in plaats van ze te simuleren. Dit maakt systeemvirtualisatie efficiënter dan procesvirtualisatie. Het aanbieden van een efficiënte 100% getrouwe softwareabstractie is echter niet eenvoudig zonder hardwareondersteuning (zoals dit het geval was in de beginperiode van VMware). Daarom zijn er alternatieven in de vorm van paravirtualisatie ontstaan. Bij paravirtualisatie biedt men geen 100% getrouwe softwareabstractie aan. Dat impliceert wel dat de software die uitgevoerd wordt moet aangepast worden om hiermee rekening te houden. In dit hoofdstuk zullen we het verder enkel hebben over systeemvirtualisatie.

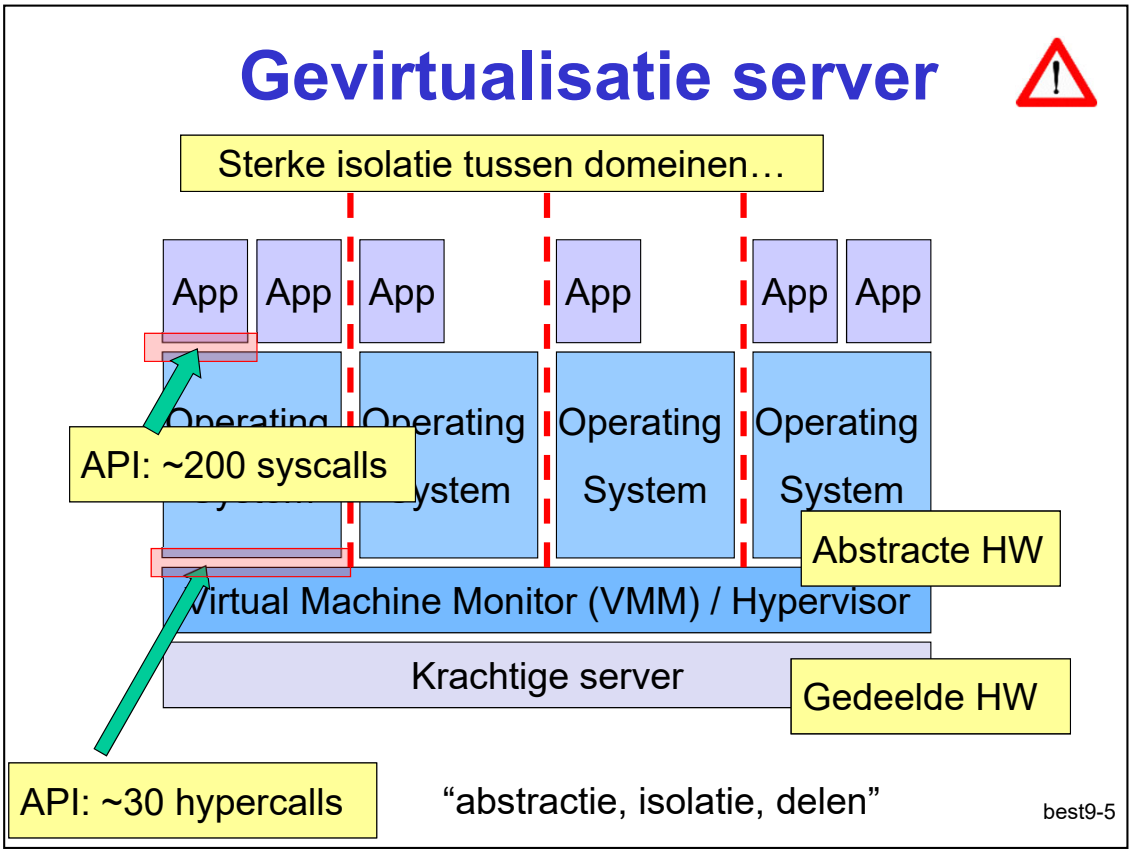
Klassiek serverpark



best9-4

In een klassiek serverpark staan er verschillende servers die elk hun eigen besturingssysteem uitvoeren, met daarboven de applicaties (webserver, databankserver, mailserver, …).

De applicaties zijn vast geïnstalleerd op bepaalde servers die gedimensioneerd zijn om allerhande belastingscenario's aan te kunnen, inclusief de piekbelasting die maar een paar keer per jaar voorkomt. De gemiddelde belasting van de individuele servers is soms maar 5-10%.

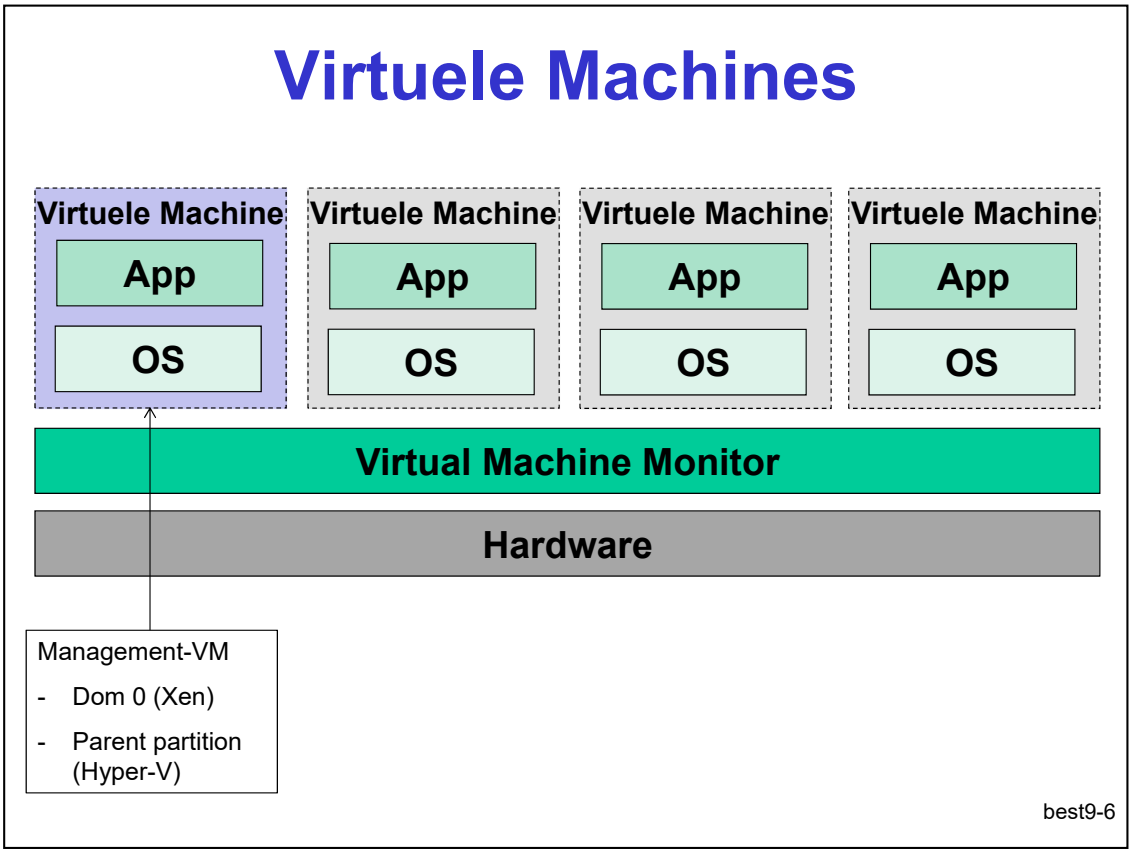


Bij een gevirtualiseerde server is de situatie enigszins anders. Eerst wordt de hardware geabstraheerd door middel van een softwarelaag die we de Virtual Machine Monitor (VMM) of Hypervisor heten. De VMM biedt een softwareabstractie van de onderliggende hardware aan, en deze softwareabstractie kan gedupliceerd worden. De VMM kan met andere woorden verschillende virtuele machines aanbieden. Op elk van die virtuele machines kan een besturingssysteem geïnstalleerd worden (deze hoeven niet allemaal dezelfde te zijn). Bovenop die besturingssystemen worden dan de applicaties geïnstalleerd. De VMM zorgt ervoor dat de verschillende virtuele machines sterk geïsoleerd zijn van elkaar (vergelijkbaar met de isolatie tussen adresruimten bij virtueel geheugen).

Een VMM (voor paravirtualisatie) heeft een relatief eenvoudige API met een paar tientallen oproepen die men hypercalls noemt. Een besturingssysteem heeft een rijkere API.

De bedoeling van virtualisatie is te abstraheren, te isoleren, en te delen (van de hardware). Het aspect delen wordt soms ook omschreven als consolideren als men vertrekt van het klassieke serverpark. De verschillende servers worden geconsolideerd op 1 krachtige server. Het voordeel van de consolidatie is dat i.p.v. vier servers met een wisselende belasting van 5-10%, we nu een misschien iets zwaardere server (maar misschien ook niet) zullen hebben met een belasting van misschien 30-40%. Doordat er minder hardwareservers zijn, zal de

investeringskost kleiner zijn, en ook het stroomverbruik zal lager liggen omdat het aantal servers kleiner is, en omdat het stroomverbruik groter is dan de proportionele belasting (een server die 10% belast wordt verbruikt meer dan 10% van de stroom van een volledig belast systeem).



Zoals reeds uitgelegd, biedt de VMM een aantal virtuele machines aan die elk hun eigen softwarestapel zullen uitvoeren. Vaak is er ook een management-VM die afhankelijk van het product verschillende namen kan dragen. Die management-VM wordt gebruikt om het gevirtualiseerde systeem te beheren (aanmaken van bijkomende VM's, instellen van beleidsparameters, monitoring van het systeem, …). Het is ook de eerste VM die gestart wordt bij het opstarten van het systeem, en vaak heeft deze VM rechtstreekse toegang tot de hardware, dit in tegenstelling met de andere VM's.

De software die draait in een VM is totaal onafhankelijk van de andere VM's. Dit levert een aantal voordelen op: virtuele machines kunnen gemakkelijk overgezet worden van één machine naar een andere, zoals we later zullen zien is het zelfs mogelijk om een virtuele machine te verhuizen terwijl applicaties aan het uitvoeren zijn. Daarnaast is het ook mogelijk om een virtuele machine te dupliceren. Als een webserver overbelast dreigt te geraken, dan is het zeer eenvoudig om er een tweede op te starten (op dezelfde server of op een andere server). Op die manier kan men aan belastingspreiding doen. Dit maakt het beheer van een serverpark uiteraard veel flexibeler.

Waarom virtualiseren?

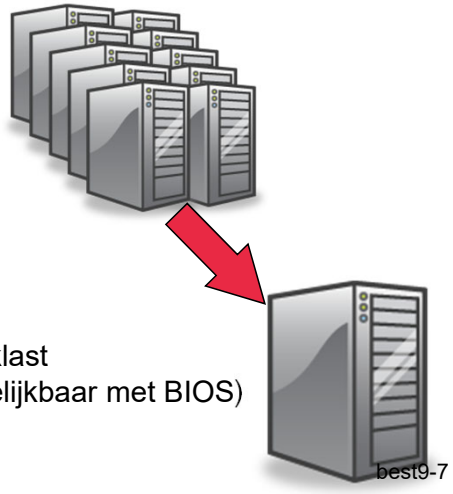
Reduceer Total Cost of Ownership (TCO)

- Toename systeembelasting (gemiddelde systeembelasting per server is vaak minder dan 10% gemiddeld en minder dan 50% bij piekbelasting)
- Hardwarebesparing (minder servers) (25% of the TCO)
- Besparing op ruimte, stroom, koeling (50% van de operationele kost van een data center)

Eenvoudiger beheer

- Dynamische provisionering
- Werklastbeheer/isolatie
- Eenvoudige migratie
- Herconfiguratie

Betere beveiliging
Compatibiliteit met legacysystemen
Bescherming van IT investeringen
Virtualisatie is een schaalbare multi-core werklast
Virtualisatie wordt een platformelement (vergelijkbaar met BIOS)



best9-7

Er zijn veel redenen waarom men virtualisatie gebruikt in het beheer van grote computerparken. Ze staan hierboven opgesomd.

Provisionering = toekennen van systeemmiddelen aan bepaalde taken (b.v. een groter of kleiner aantal webservers).

De compatibiliteit met legacysystemen moet als volgt begrepen worden: (i) de VMM kan in principe ook een abstractie van een andere (=oudere) machine aanbieden dan de onderliggende machine. Als dit gaat over een andere ISA, dan zal dit uiteraard gepaard gaan met een grote inefficiëntie. Als het gaat over het al dan niet voorkomen bepaalde instructies, is dit wel haalbaar met een beperkte prestatie-impact. (ii) op één server kunnen er verschillende VM's draaien. Indien er nog een legacyapplicatie is die een oudere versie van een besturingssysteem nodig heeft, dan kan deze perfect draaien naast alle andere VM's met hun state-of-the-art versie van het besturingssysteem. Men hoeft voor deze legacyapplicatie geen afzonderlijke server meer te reserveren.

Multicores worden in klassieke besturingssystemen onderbenut omdat veel toepassingen ééndradig zijn, en dus geen extra cores gebruiken. Door op een multicore een VMM te draaien die verschillende VM's ondersteunt, kunnen de individuele cores beter gebruikt worden.

Definitie van virtuele machine

- G. Popek and R. Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures”, Comm. ACM 17 (7), 1974

- Een virtuele machine is een softwareabstractie van een fysieke machine, die voldoet aan drie voorwaarden
  - **Equivalent**: de uitvoeringsomgeving moet identiek zijn aan die van de fysieke machine (met uitzondering van timing, totaal #systeemmiddelen, etc.)
  - **Veilig**: VMM controleert alle systeemmiddelen en zorgt ervoor dat de diverse VM’s van elkaar geïsoleerd zijn
  - **Efficiënt**: de waarneembare vertraging moet minimaal zijn

best9-8

Het begrip virtuele machine werd reeds in 1974 gedefinieerd door Popek en Goldberg.

Overzicht

- Definities
- Klassieke virtualiseerbaarheid
- Virtualisatie
  - CVE
  - Geheugen
  - IO
- Migratie

best9-9

Een hardwareplatform dat naar de geest van Popek en Goldberg (equivalent, veilig en efficiënt) kan gevirtualiseerd worden omschrijft men als ‘klassiek virtualiseerbaar’.

Formele Vereisten voor klassieke virtualiseerbaarheid

- Een machine M wordt gekarakteriseerd door drie klassen van instructies:
- P(M) zijn de **geprivilegieerde instructies**
  - Een instructie is geprivilegieerd indien ze een trap genereert in gebruikersmode, maar niet in systeemmode
- S(M) zijn de **sensitieve instructies**, bestaande uit
  - Controlesensitieve instructies die b.v. de processormode of the MMU-instellingen veranderen
  - Locatiesensitieve instructies waarvan het gedrag afhangt van de processormode of van de plaats in het geheugen
- O(M) zijn de **onschadelijke instructies** (=niet-sensitieve instructies)

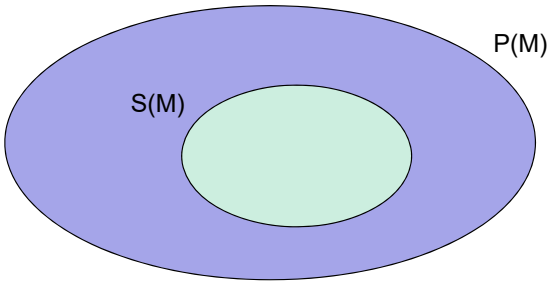
best9-10

Op het niveau van de ISA maakt men een onderscheid tussen drie klassen van instructies.

Klassieke virtualiseerbaarheid

M is klassiek virtualiseerbaar indien

$S(M) \subseteq P(M)$



dan kunnen we het besturingssysteem deprivilegiëren,
O(M) rechtstreeks uitvoeren, S(M) traps laten genereren,
en de VMM de systeemcode laten emuleren

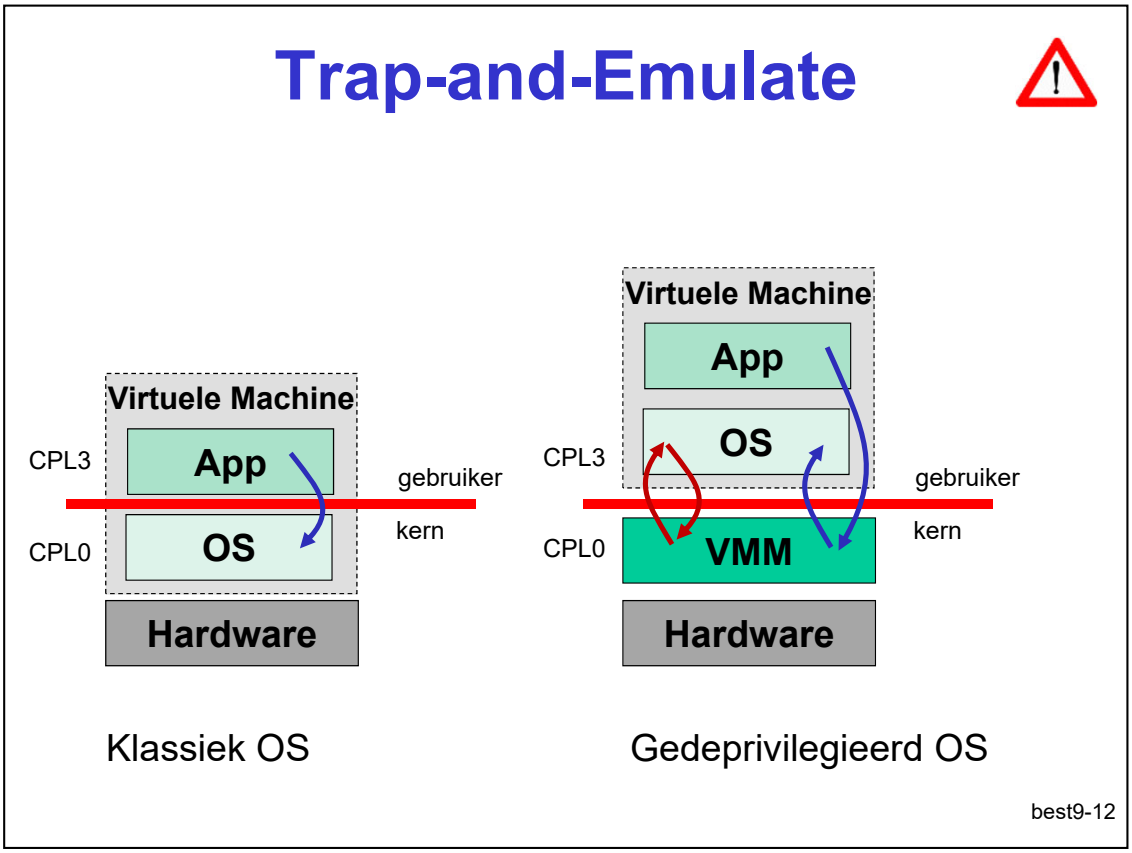
“Trap-and-Emulate”

best9-11

Popek en Goldberg stellen dat een architectuur klassiek virtualiseerbaar is als alle sensitieve instructies ook geprivilegieerde instructies zijn.

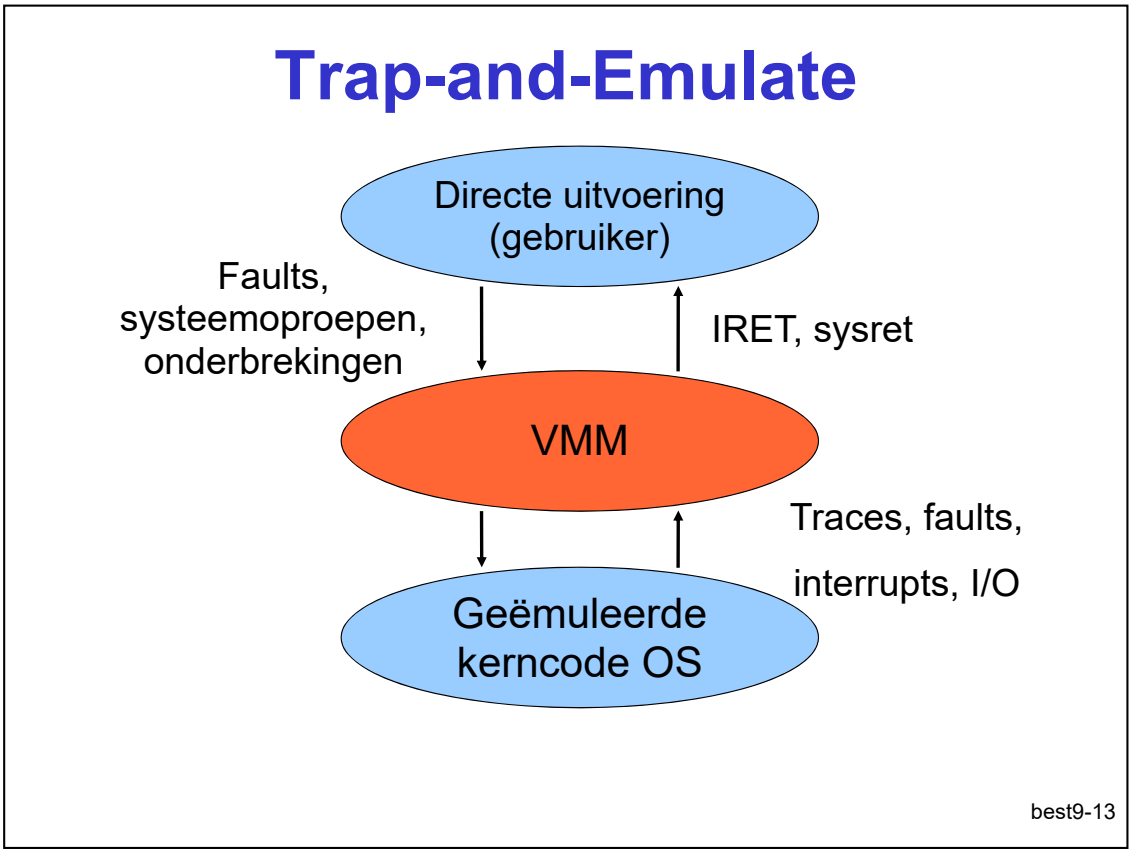
Dat wil zeggen dat de instructies een trap zullen genereren van zodra ze in gebruikersmode uitgevoerd worden. Dit laat ons toe om het volgende probleem op te lossen: de VMM moet in een hoger protectieniveau uitgevoerd worden dan het besturingssysteem. Aangezien het systeemniveau het hoogste protectieniveau is, moet de VMM op dat niveau uitgevoerd wordt. Dit impliceert ook dat het besturingssysteem in gebruikersmode moet uitgevoerd worden, samen met de applicaties.

Men noemt dit het deprivilegiëren van het besturingssysteem. Alle geprivilegieerde instructies uit het besturingssysteem zullen dan automatisch een trap genereren. De VMM kan dan nagaan wat de oorzaak van de trap was, de geprivilegieerde instructie lokaal in systeemmode uitvoeren en de resultaten terug naar de VM sturen.

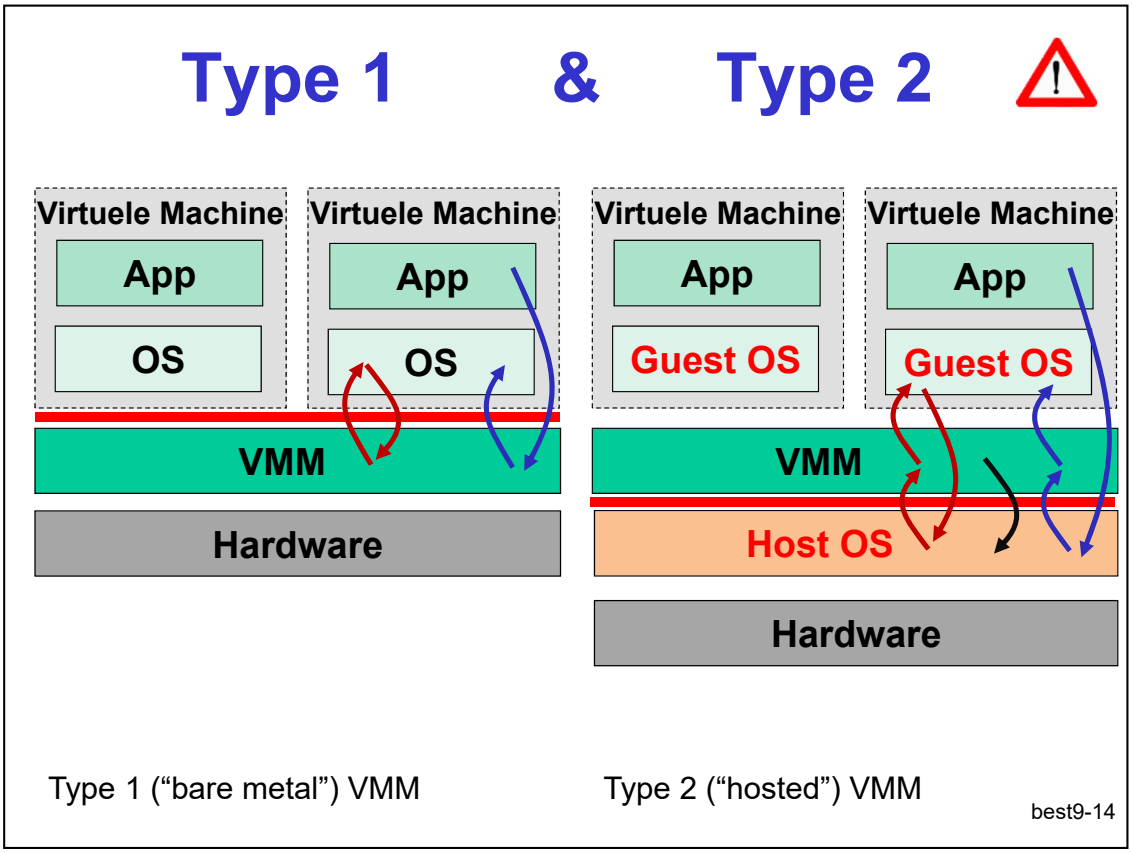


Rechts staat het beeld van een gedeprivilegieerd besturingssysteem. Alle geprivilegieerde instructies die uitgevoerd worden in het besturingssysteem veroorzaken een trap naar de VMM waar de instructie dan uitgevoerd wordt. In tegenstelling met het klassieke besturingssysteem waar de geprivilegieerde instructies gewoon uitgevoerd worden, moet de VMM nu wel degelijk op zoek naar de oorzaak van de trap door te gaan kijken door welke instructie hij veroorzaakt werd en deze instructie dan te emuleren in de VMM.

Eventuele geprivilegieerde instructies uit de applicaties of instructies die van nature een trap uitvoeren (zoals systeemoproepen), komen ook terecht in de VMM. Deze gaat op zijn beurt na wat er aan de hand is, en emuleert dan de code die het besturingssysteem in een dergelijke situatie zou uitvoeren (als er bv. een paginafout optreedt, dan moet de VMM uiteraard beroep doen op het besturingssysteem om te weten wat het besturingssysteem in een dergelijke situatie zou doen). Het mag duidelijk zijn dat geprivilegieerde instructies een stuk trager zullen uitgevoerd worden. Gelukkig zijn het niet de meeste voorkomende instructies in een werklast.



Dit is een meer realistisch beeld van wat er gebeurt bij deprivilegiëring. De VMM nestelt zich eigenlijk tussen de applicatie en het besturingssysteem en observeert vanuit die positie alles wat er gebeurt, en leidt alles in goede banen.



De VMM die we tot nog toe besproken hebben wordt door Popek en Goldberg een Type 1 VMM genoemd. Dit is een VMM die rechtstreeks op de hardware uitgevoerd wordt.

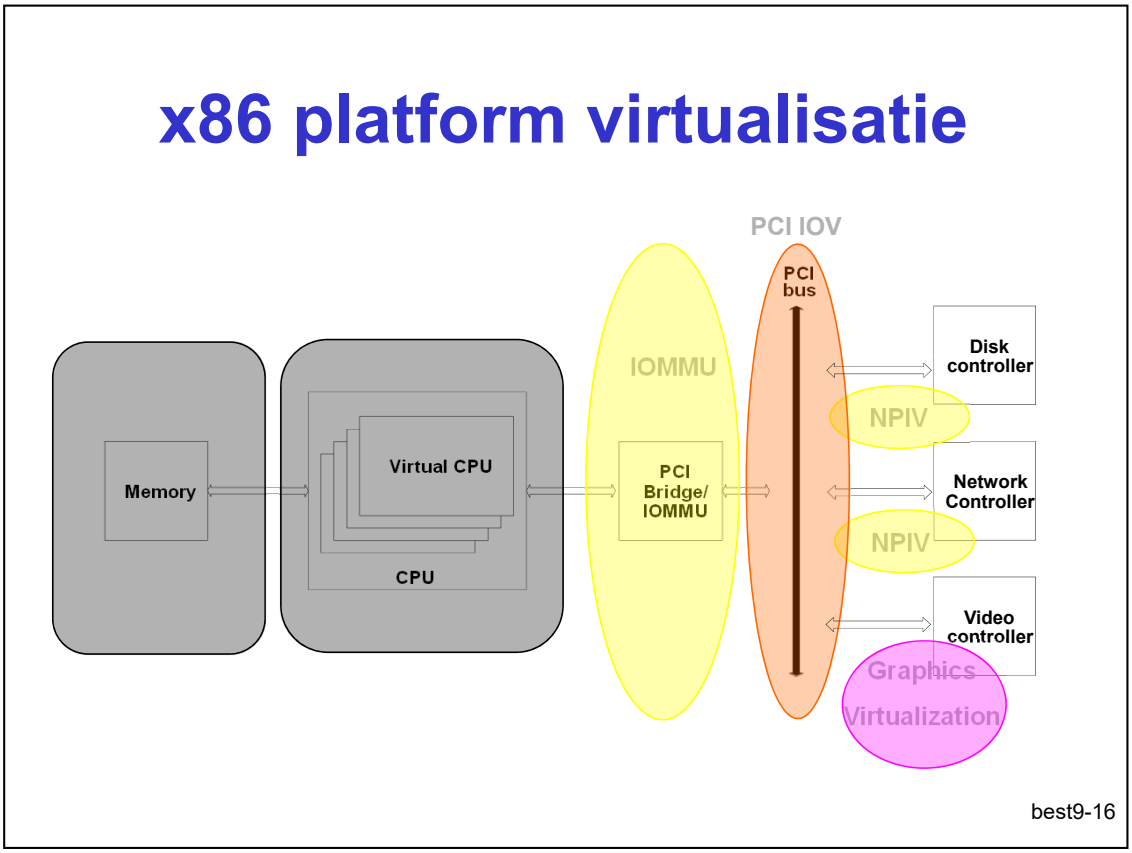
Men kan een VMM echter ook als traditionele gebruikersapplicatie laten uitvoeren. Dat wordt een Type 2 VMM genoemd (zie rechts, men maakt nu een onderscheid tussen het Host OS en het Guest OS – beide hoeven niet hetzelfde te zijn). Een type 2 VMM kent wel een ander uitvoeringsverloop. Naast het feit dat de VMM nu een gebruikersapplicatie is die zich tevreden zal moeten stellen met de rekentijd die ter beschikking gesteld wordt door het Host OS (men kan de VMM wel een hogere prioriteit geven, maar het blijft een gebruikersapplicatie die in competitie zal moeten treden met de andere gebruikersprocessen), verloopt de uitvoering van de geprivilegieerde instructies nu ook anders. Alle traps gaan naar systeemmode, en dit betekent in dit geval het Host OS. Het Host OS zal moeten uitzoeken wat de VMM bij een dergelijke trap van plan was om te doen, en deze VMM moet dan op zijn beurt in het Guest OS op zoek naar de betekenis van de trap. Deze extra indirectie veroorzaakt uiteraard bijkomende inefficiënties.

Overzicht

- Definities
- Klassieke virtualiseerbaarheid
- Virtualisatie
  - CVE
  - Geheugen
  - IO
- Migratie

best9-15

We gaan nu over naar de virtualisatie van de drie basiscomponenten van een computersysteem: de CVE, het geheugen en de IO.



Dit is het model van het platform dat we voor de rest van dit hoofdstuk zullen gebruiken. De CVE is verbonden met het geheugen en met het IO-systeem. De gekleurde vlakken zijn virtualisatieoplossingen.

Problemen Trap-and-Emulate

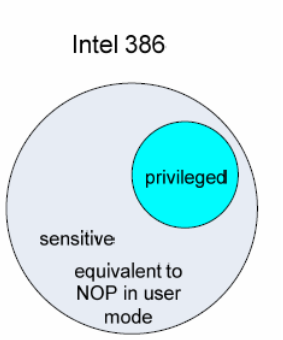
- Duur (~3000 cycli per trap)
- Er treden zeer veel traps op
  - B.v., paginafouten, I/O instructies, …
- De oude x86 ondersteunt geen trap-and-emulate
  - Sleutelprobleem: 16 dual-purpose instructies
  - Klassiek voorbeeld: popf-instructie: gedrag afhankelijk van de mode
    - Gebruikersmode: verandert de ALU-vlaggen
    - Systeemmode: verandert ALU en systeemvlaggen
    - + Genereert geen trap in gebruikersmode
- Privilegeniveau guest observeerbaar in x86 (via %cs)

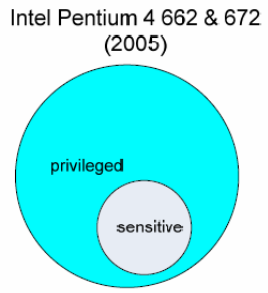
best9-17

Trap-and-Emulate werkt niet bij de originele Intelarchitectuur van de jaren 90 omdat er sensitieve instructies (zoals popf) zijn die niet geprivilegieerd zijn. De popf-instructie gedraagt zich anders in systeemmode en in gebruikersmode. Dat wil zeggen dat het gedrag van het besturingssysteem zal veranderen als het gedeprivilegieerd uitgevoerd wordt. Daarnaast zijn er ook nog een aantal andere problemen zoals het feit dat het besturingssysteem kan achterhalen dat het gedeprivilegieerd uitgevoerd wordt door naar de inhoud van %cs te gaan kijken.

Virtualisatie CVE

- In software
  - Simuleer de processor (traag)
  - Dynamisch binair herschrijven
  - Paravirtualisatie





- In hardware
  - Trap-and-emulate

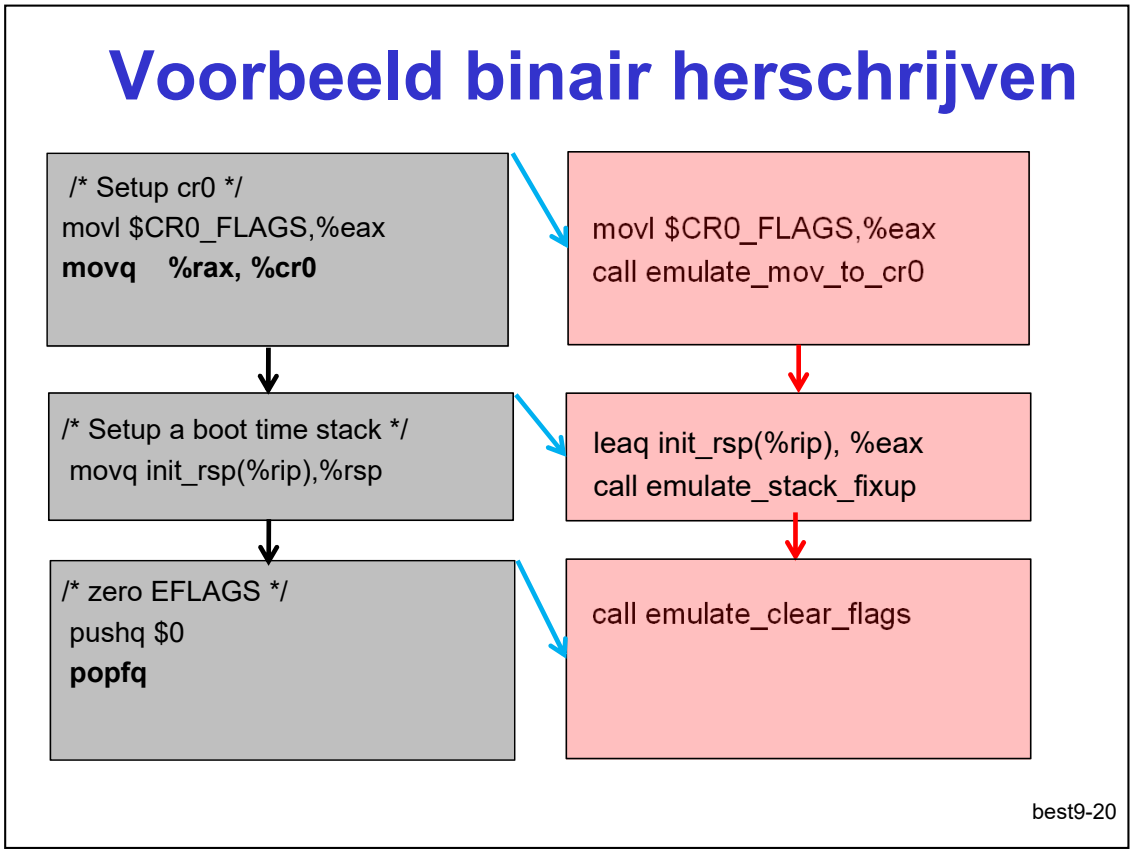
best9-18

Pas in 2005 zijn de sensitieve instructies een deelverzameling van de geprivilegieerde instructies geworden en kon de architectuur klassiek gevirtualiseerd worden met trap-and-emulate. Tot die tijd was er geen andere mogelijkheid dan de processor in software te virtualiseren door hem ofwel te simuleren, de code binair te herschrijven of te paravirtualiseren.

Binair herschrijven

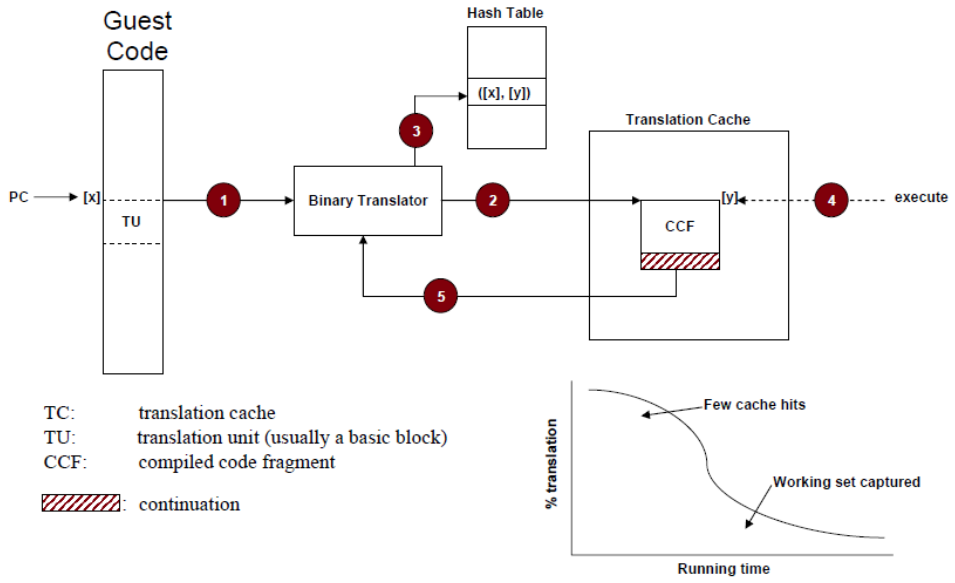
- Om te virtualiseren moet de VMM de S(M) instructies kunnen detecteren en vervangen door de gepaste emulatieroutine.
- Dit kan b.v. door x86 instructies te interpreteren en systeemtoestand van processor op de juiste manier aan te passen. Dit is echter zeer traag.
- VMware oplossing: **binair herschrijven = just-in-time compilatie van x86 naar x86**
  - O(M) wordt op zichzelf afgebeeld
  - S(M) wordt geëmuleerd
  - P(M) wordt gebruikt om de overgangen tussen gebruikersmode en systeemmode te detecteren
- Hoeft enkel maar voor kerncode te gebeuren – applicaties blijven onveranderd in gebruikersmode draaien

best9-19



Bij binair herschrijven wordt de te emuleren code (dit is in de praktijk de code van het besturingssysteem) afgelopen en alle probleeminstructies worden vervangen door aangepaste oproepen naar de VMM. Eenmaal de code herschreven is en zich in de het geheugen van de VMM bevindt, kan ze zonder bijkomende traps uitgevoerd worden. VMware was het eerste bedrijf dat men deze oplossing op de markt kwam.

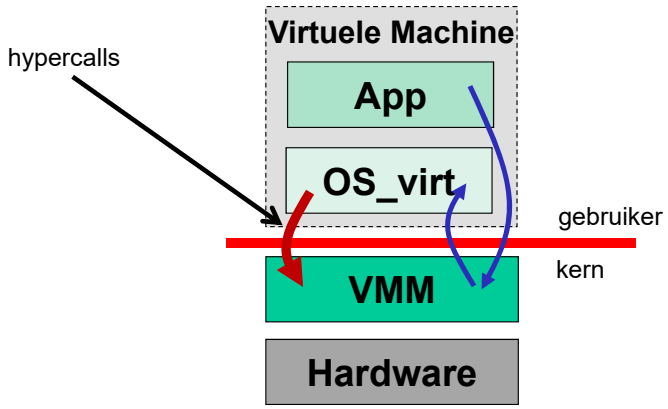
VMware binaire herschrijver



best9-21

Het geniale van de VMware-oplossing was dat de VMM de vertaalde stukken code bijhield in een cache waardoor ze maar eenmaal moesten vertaald worden. Van zodra de hete stukken code van het besturingssysteem vertaald waren, was er geen tussenkomst van de binaire vertaler meer nodig, en werd de code bovendien sneller dan klassieke trap-and-emulate omdat in de oorspronkelijk code verschillende geprivilegieerde instructies konden voorkomen die na vertaling allemaal vervangen werden door veel efficiëntere functieoproepen in de VMM. Dit was een belangrijke doorbraak in de VMM's voor de Intel architectuur.

Paravirtualisatie = statisch binair herschrijven



best9-22

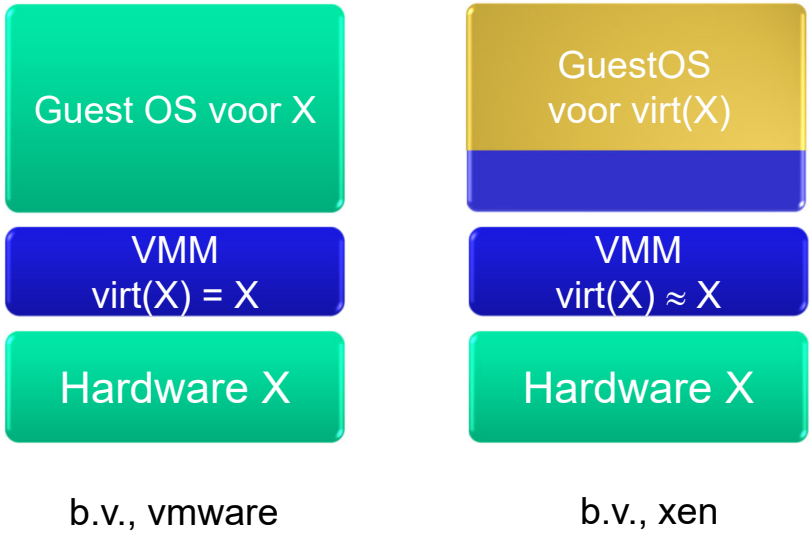
Daarnaast werd er ook gewerkt aan paravirtualisatie (vooral door Xen). Voor deze oplossing is het nodig om een speciale versie van het besturingssysteem te maken voor een bepaalde VMM.

Dit geparavirtualiseerd besturingssysteem maakt dan gebruik van hypercalls om de systeeminstructies uit te voeren. Traps vanuit de applicaties blijven uiteraard op dezelfde manier afgehandeld worden als voorheen.

Paravirtualisatie is zeer efficiënt, maar heeft als belangrijkste nadeel dat het besturingssysteem moet aangepast worden. Voor open source besturingssystemen is dit mogelijk, voor gesloten systemen zoals Microsoft Windows is dit geen optie.

Bijkomende nadelen zijn dat het paravirtualiseren van een besturingssysteem geen eenmalige operatie is, maar bij elke nieuwe versie van het besturingssysteem opnieuw moet gebeuren, dat het niet noodzakelijk gebeurt door de ontwikkelaars van het besturingssysteem, en dat men door het paravirtualiseren ook fouten kan introduceren in het besturingssysteem.

Volledige virtualisatie vs paravirtualisatie

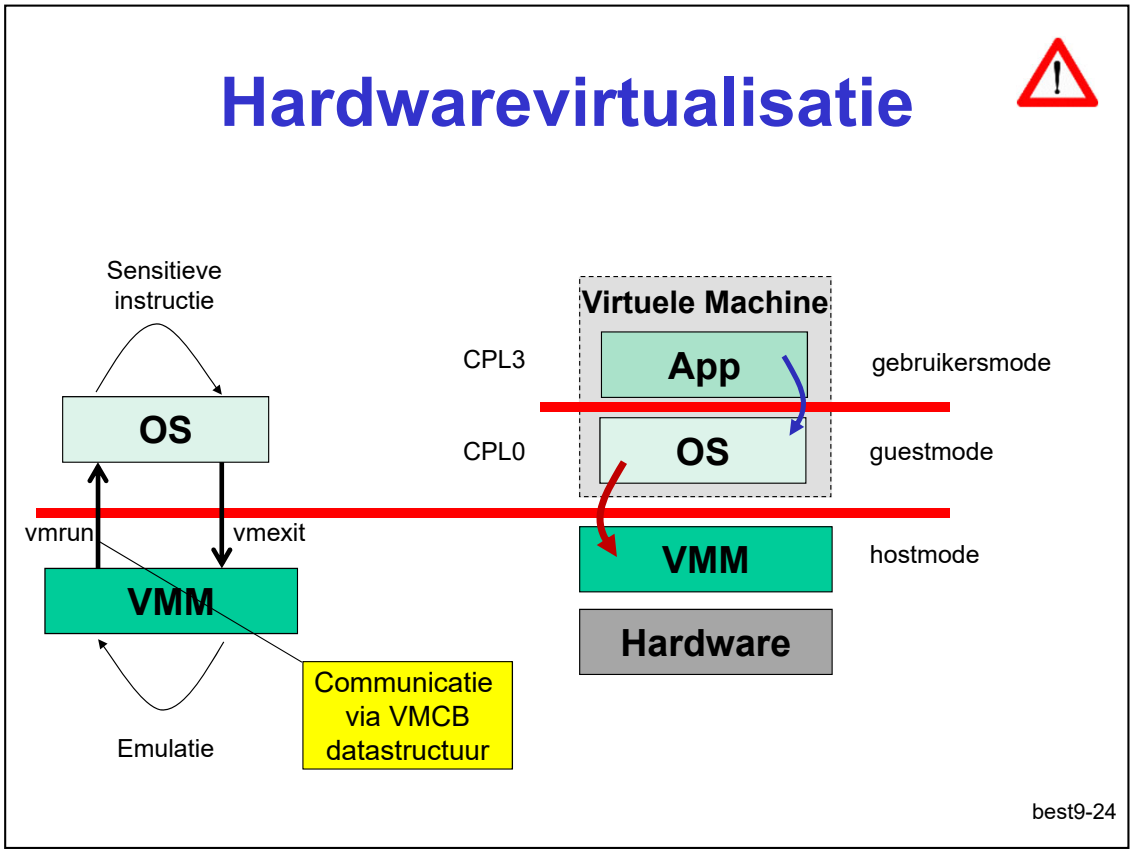


b.v., vmware

b.v., xen

best9-23

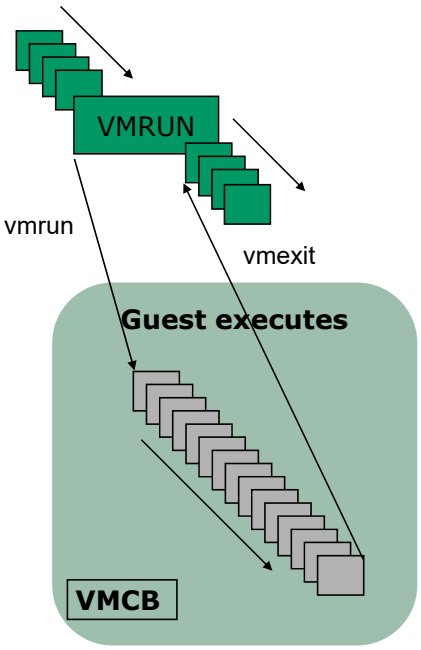
Bij paravirtualisatie strekt de VMM zich als het ware tot in het besturingssysteem uit.



De meeste hardwarefabrikanten bieden tegenwoordig hardwareondersteuning voor virtualisatie aan. Deze komt er in grote lijnen op neer dat ze ervoor zorgen dat alle sensitieve instructies geprivilegieerd zijn, en dat ze een bijkomend privilegeniveau voor de VMM introduceren. Dat betekent concreet dat het besturingssysteem in systeemmode (nu guestmode genoemd) blijft uitvoeren, en dat de applicaties en het besturingssysteem zich dus niet bewust hoeven te zijn van het feit ze in een virtuele machine uitgevoerd worden. Alle traps van de applicaties gaan automatisch naar de guestmode, zonder via de VMM om te gaan. Alle traps die gegenereerd worden in de guestmode worden afgehandeld in de hostmode waarin de VMM draait. Als het besturingssysteem dus een in- of een out-instructie wil uitvoeren dan zal dit steeds onder de controle van de VMM gebeuren, en dat is ook nodig voor de isolatie want we willen niet dat de ene VM het IO-gedrag van een andere VM kan beïnvloeden.

De overgang tussen hostmode en guestmode gebeurt via de vmrun-instructie. Van zodra het besturingssysteem een geprivilegieerde instructie probeert uit te voeren, treedt er een vmexit op en keert de controle terug naar de VMM waar de geprivilegieerde instructie dan geëmuleerd wordt. De communicatie tussen de beide niveaus gebeurt via een VM controleblok (VMCB). Sensitieve instructies worden nu in guestmode uitgevoerd waardoor ze hun oorspronkelijk semantiek bewaren, en dus geen tussenkomst van de VMM nodig hebben.

Hardwarevirtualisatie

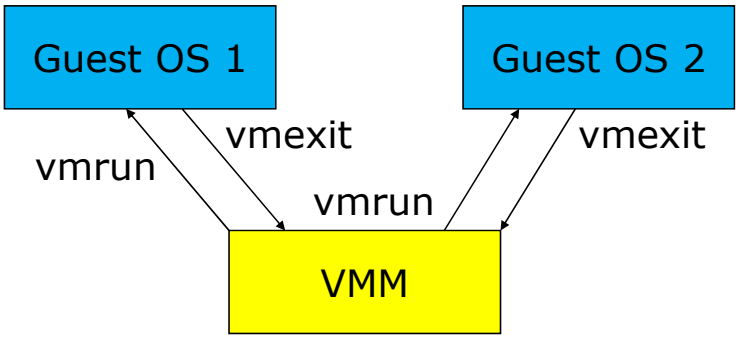


- De guest wordt opgeroepen door de VMM via vmrun
- De guest voert uit totdat
  - Hij een hypercall uitvoert
  - Hij een actie uitvoert die een exit veroorzaakt
- Per guest worden de exitvoorwaarden bepaald in de VMCB (VM control block)
  - Welke excepties en onderbrekingen veroorzaken een exit
  - Welke instructies veroorzaken een exit

best9-25

Deze slide schetst het verloop van een vmrun-vmexit.

World switch



“world switch” is heel duur

Oplossingen:
- Versnellen van world switch (cache, tlb, …)
- Verminderen van de frequentie ervan

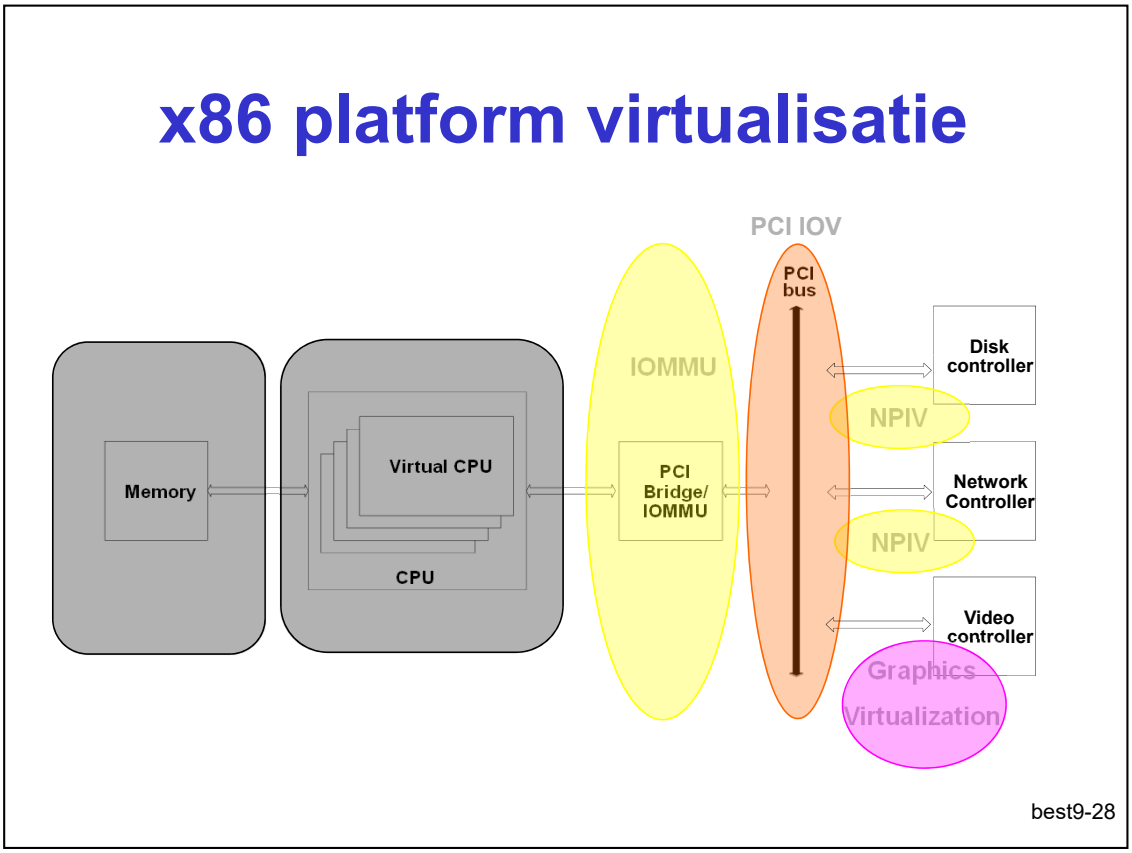
best9-26

De VMM zal doorgaans meer dan één VM ondersteunen hetgeen wil zeggen dat de VMM ervoor moet zorgen dat de systeemmiddelen van het platform verdeeld worden tussen de verschillende VM’s. Dit vereist contextwisselingen op VM-niveau. Omdat de toestand van een VM natuurlijk veel groter is dan die van een proces, noemt men dit een **world switch**, en dit is een zeer dure operatie. Alle technieken om deze te versnellen worden in de praktijk gebruikt (zie ook verder). Door VM’s toe te wijzen aan verschillende cores kan men in de praktijk het aantal world switches aanzienlijk reduceren omdat ze dan elk hun eigen core hebben en ze niet in competitie komen voor de processortijd op die core.

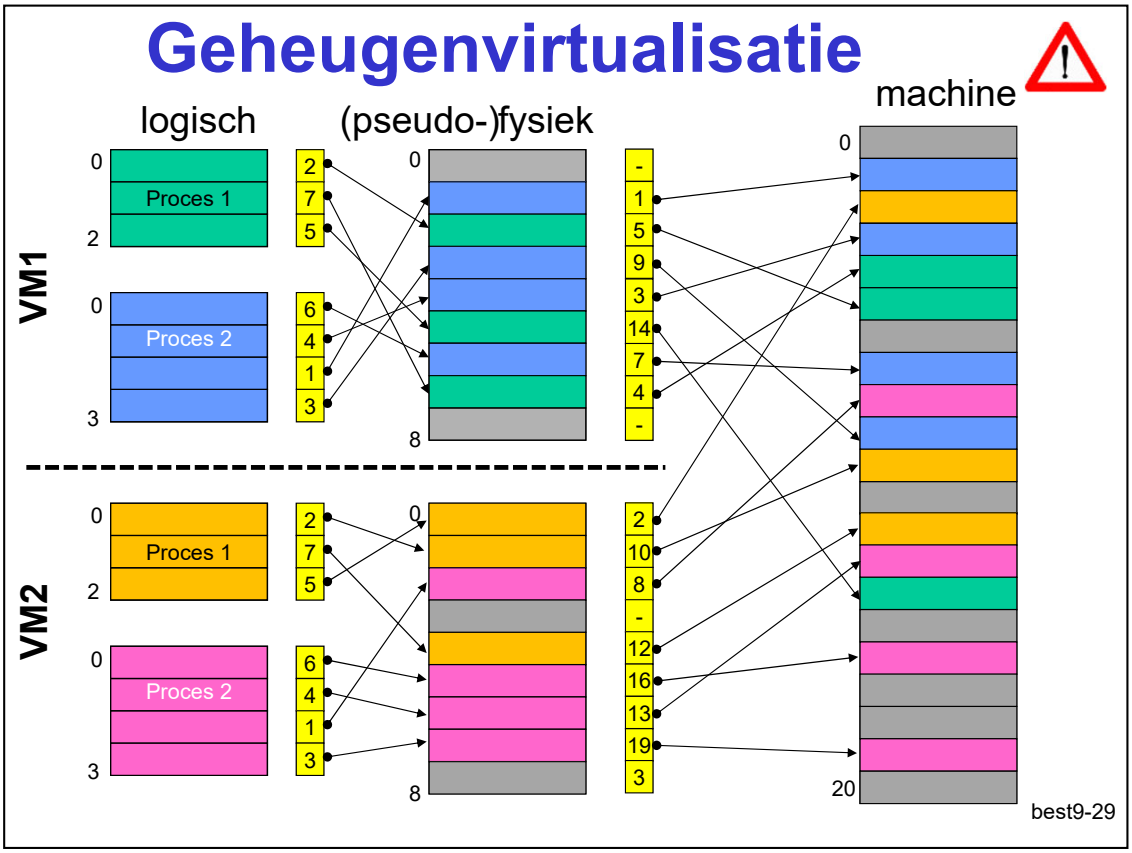
Overzicht

- Definities
- Klassieke virtualiseerbaarheid
- Virtualisatie
  - CVE
  - Geheugen
  - IO
- Migratie

best9-27



Het geheugen in de tweede component die moet gevirtualiseerd worden. Hierbij doet zich de moeilijkheid voor dat het geheugen binnenin het besturingssysteem reeds gevirtualiseerd is, en dat het dus een tweede maal zal moeten gevirtualiseerd worden op het niveau van de VMM.



Net zoals bij de processor zullen we dit realiseren door een bijkomend niveau toe te voegen (voor de processor werd de systeemmode de guestmode, en werd de hostmode toegevoegd als protectieniveau voor de VMM). Hier zal het fysieke geheugen nu pseudo-fysiek geheugen worden dat via adresvertaling zal afgebeeld worden op zgn. machinegeheugen. Het besturingssysteem zal dus blijvend een vertaling uitvoeren van logische adressen naar (pseudo-)fysieke adressen. De VMM voert dan een bijkomende vertaling uit van (pseudo-)fysieke adressen naar machineadressen.

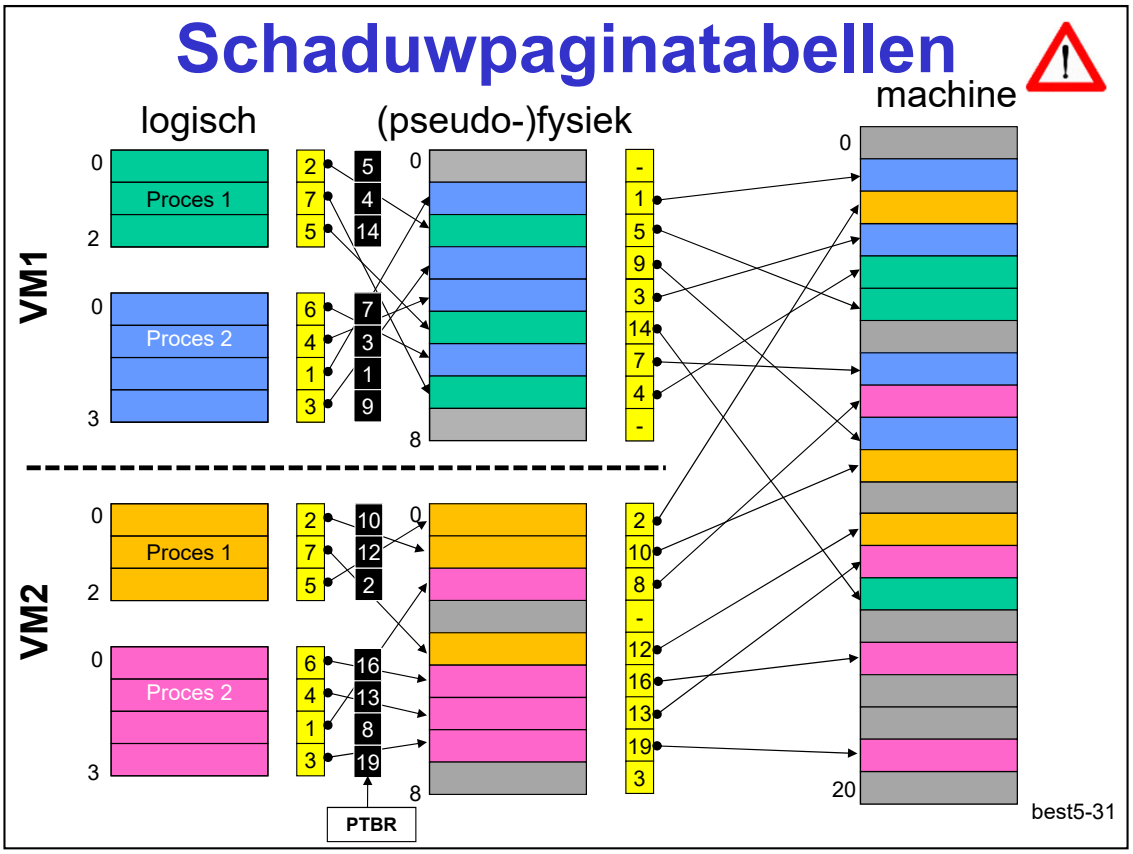
Virtualisatie geheugen

- Software: schaduwpaginatabellen
- Hardware: Second level address translation
  - Intel: Extended Page Tables
  - AMD: Nested Page Tables

- Compressie van de geheugenvoetafdruk

best9-30

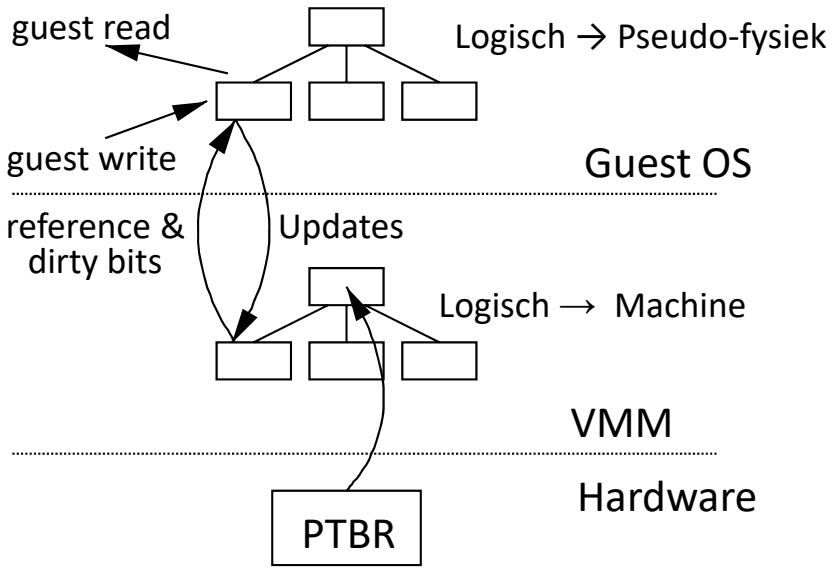
Er zijn opnieuw twee manieren om het geheugen te virtualiseren: met of zonder hardwareondersteuning. Eens te meer zal blijken dat er veel aan efficiëntie kan gewonnen worden door hardwareondersteuning.



De oplossing zonder hardwareondersteuning staat voor de uitdaging om de klassieke MMU (die logische adressen omzet naar fysieke) te gebruiken om de omzetting van logische adressen naar machineadressen uit te voeren. In principe zou de MMU hiervoor tweemaal na elkaar moeten gebruikt worden, eenmaal van logische naar (pseudo-)fysiek, en nogmaals voor de omzetting van (pseudo-)fysiek naar machineadresssen. Dit is echter niet mogelijk.

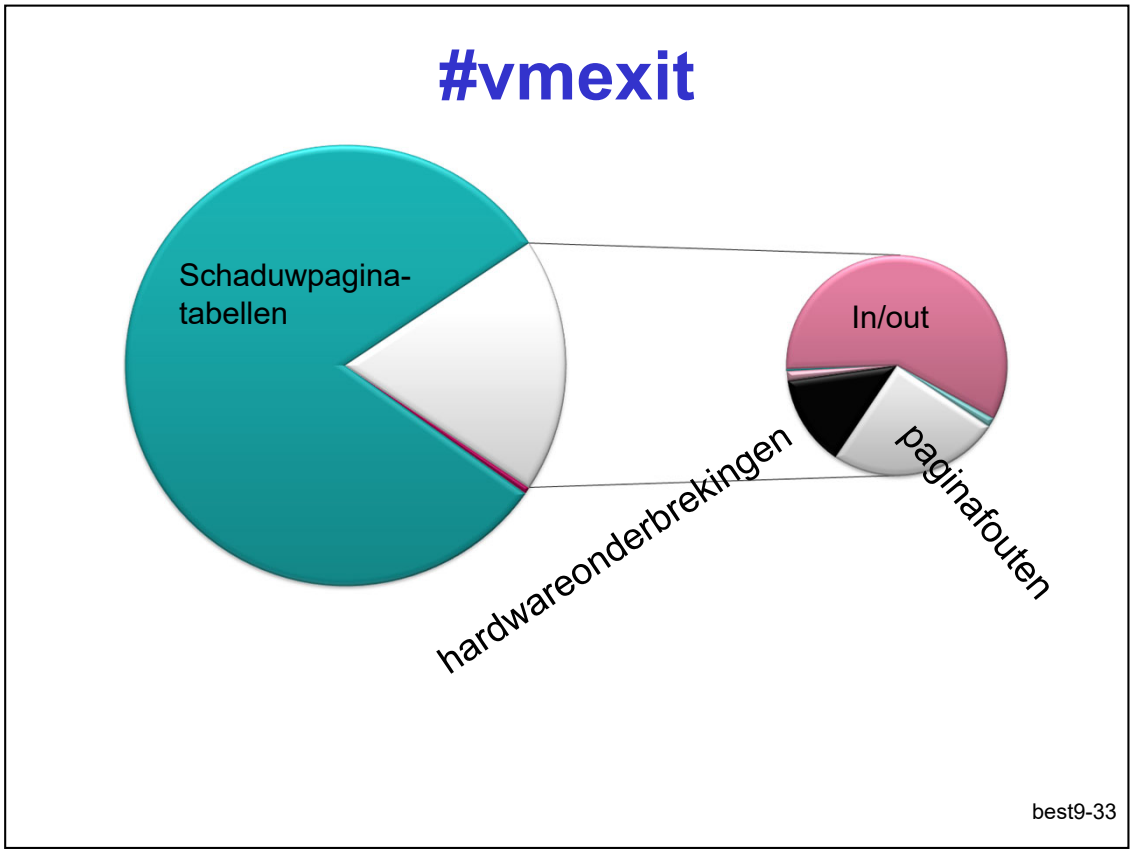
De softwareoplossing houdt per proces een extra set paginatabellen bij (in het zwart, de zgn. schaduwpaginatabellen) die de omzetting van logische naar machineadressen in één keer uitvoeren. Dit is op het eerste zicht een elegante oplossing, maar ze is minder triviaal dan ze lijkt. De grootste uitdaging is om de schaduwpaginatabellen gesynchroniseerd te houden met de paginatabellen per proces zoals die door het besturingssysteem bijgehouden worden. Het besturingssysteem mag immers niet kunnen merken dat het in een virtuele machine uitgevoerd wordt. Dit is een complexe aangelegenheid.

Schaduwpaginatabellen

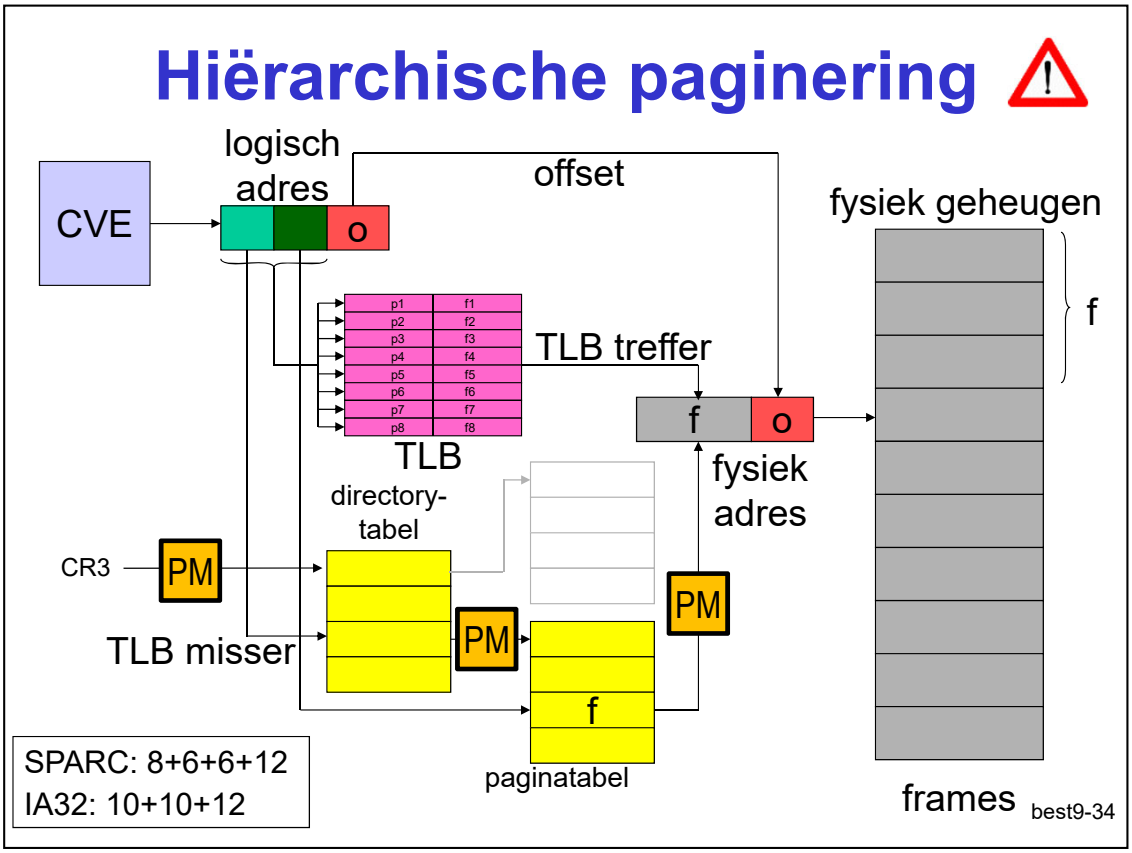


best9-32

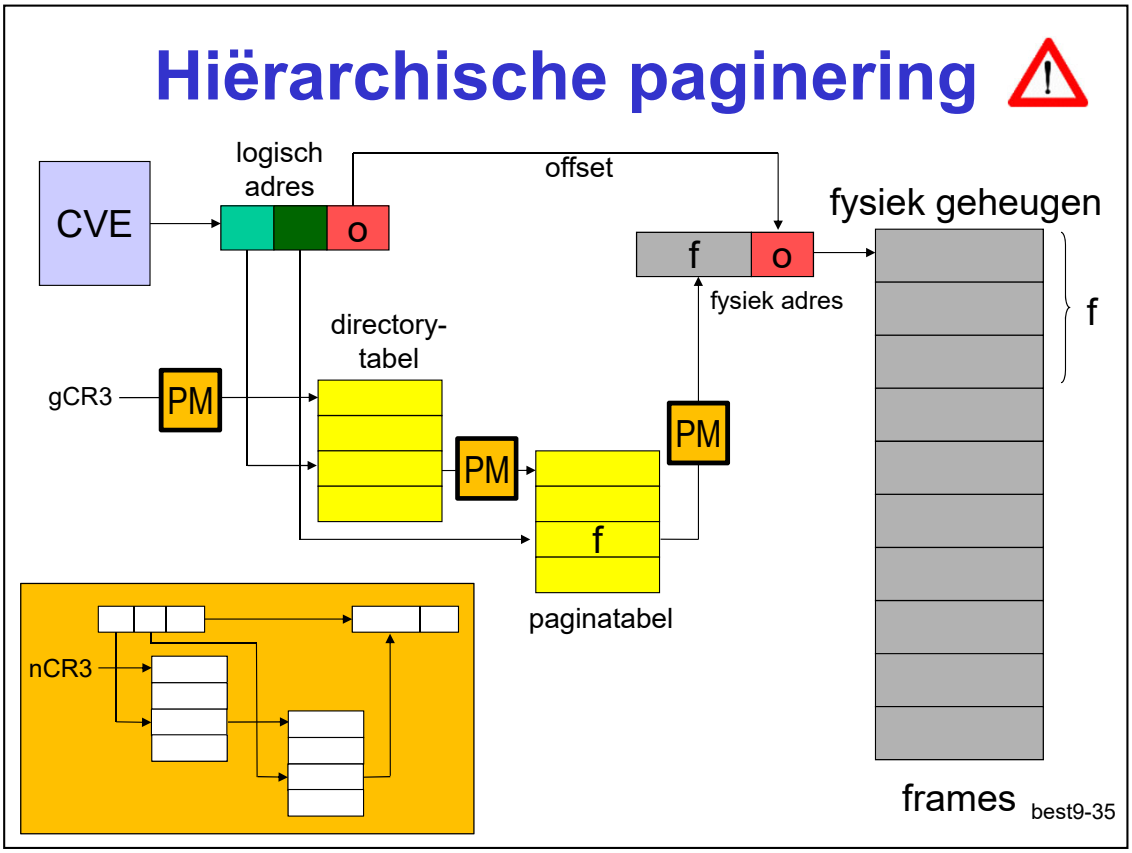
Het bijhouden van de schaduwpaginatabellen in de VMM is complex
- De VMM onderschept alle (lezen+schrijven) toegangen naar PTBR (register CR3)
- De VMM monitort alle veranderingen in de paginatabellen van het Guest OS (deze worden readonly gemaakt zodat elke wijziging opgemerkt wordt)
- Alle wijzigingen in de schaduwpaginatabellen moeten ook teruggepropageerd worden naar de paginatabellen van het Guest OS – bij een world switch moeten die immers correct bewaard worden
- Het Guest OS krijg de echte paginatabellen en de echte PTBR nooit te zien.
- Address Space ID's (ASID) worden gebruikt om de TLB-prestatie te verhogen (TLB houdt dan vertalingen bij voor verschillende Guest OS's zodat de TLB bij een world switch niet gewist hoeft te worden)



Het gesynchroniseerd bijhouden van de schaduwpaginatabellen veroorzaakt 75% van alle vmexits in een virtuele machine. Het is dus een voor de hand liggende kandidaat voor hardwareondersteuning. De overblijvende vmexits gaan meer dan de helft over IO-operaties. De rest is verdeeld over hardwareonderbrekingen en paginafouten.



We vertrekken van tweeniveaupaginering zoals die gebruikt wordt in de IA32. Alle oorspronkelijk fysieke adresssen (CR3, maar ook alle inhouden van de tabellen omdat we bij het aflopen van de tabellen geen gebruik kunnen maken van adresvertalingen; we kunnen geen adresvertaling implementeren op basis van diezelfde adresvertaling) worden nu pseudofysiek. Dat wil zeggen dat de adressen in de tabellen nu niet langer naar fysieke geheugenframes verwijzen en dat de MMU de inhoud van de tabellen niet meer kan terugvinden zonder dat die pseudofysieke adressen eerst omgezet worden in machineadressen. Hiervoor wordt een extra blokje ingelast dat de pseudofysieke adressen omzet naar machineadressen (PM).

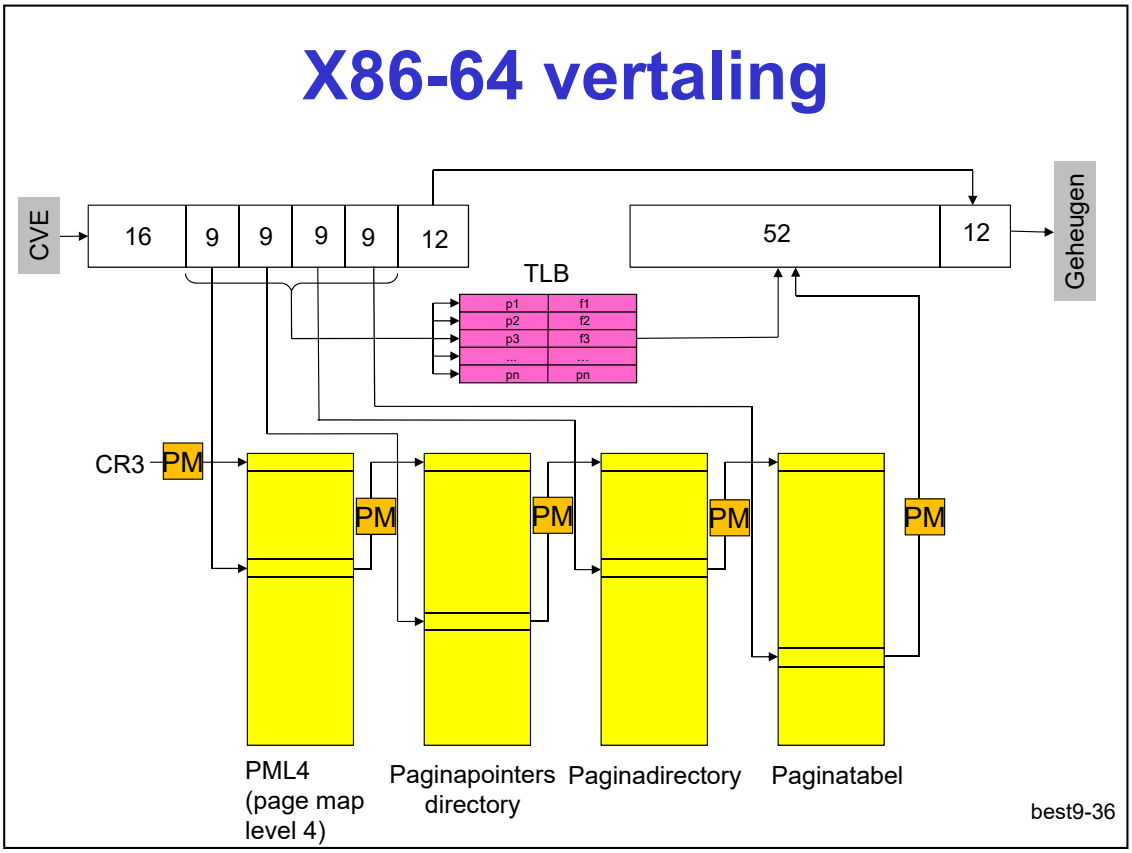


De implementatie van PM is ook een gewone adresvertaling, in dit geval ook in twee niveaus. Deze tabellen zullen bijgehouden worden door de VMM. Het guest-OS is totaal onbewust dat deze vertaling automatisch toegevoegd wordt bij het aflopen van de tabellen.

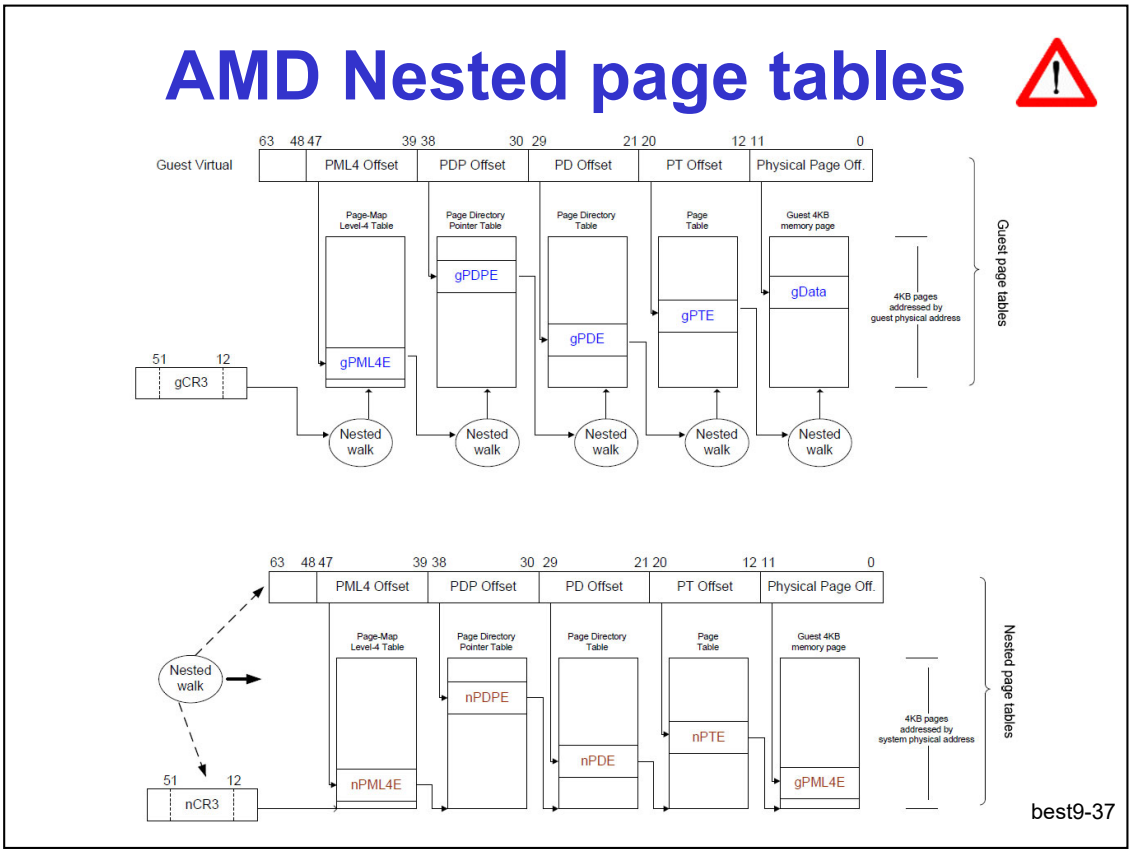
gCR3 = CR3 voor de guest (beheerd door het guest-OS)

nCR3 = CR3 voor het 'nested' niveau (beheerd door de VMM)

Zonder nested niveau wordt elke geheugentoegang vervangen door drie geheugentoegangen (2 tabellen + geheugen) als er geen TLB gebruikt wordt. Bij het gebruik van een nested niveau wordt een geheugentoegang vervangen door 8 geheugentoegangen (3 van zonet + 3 keer 2 toegangen voor het PM-blokje). Dit is uiteraard enkel aanvaardbaar op voorwaarde dat de TLB de meeste vertalingen overbodig maakt.



Bij 64-bit-vertaling gebeurt precies hetzelfde, maar dan 5 keer i.p.v. 3 keer. Het PM-blokje zal ook gebruik maken van 4 niveaus.



Het volledige schema voor de AMD-architectuur.

Volledige vertaling

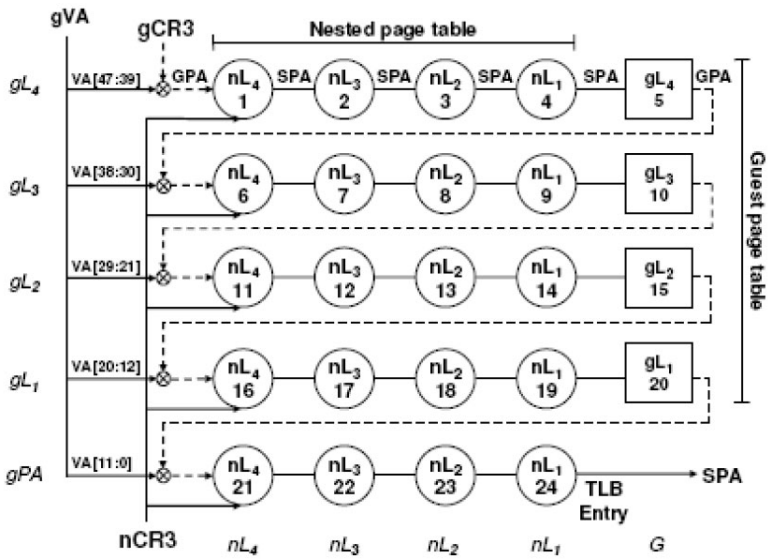


Figure 5: Address translation with nested paging. GPA is guest physical address; SPA is system physical address; nL is nested level; gL is guest level

best9-38

Concreet betekent dit dat een adresvertaling die oorspronkelijk uit vier tabeltoegangen bestond, er nu 5x4 toegangen bijkrijgt, of 24 in totaal.
Dit is een niet-verwaarloosbare toename, maar dankzij de goede werking van de TLB is dit aanvaardbaar. Het maakt het bijhouden van schaduwpaginatabellen volledig overbodig, en dat betekent een grote reductie in de complexiteit van de VMM.

Virtualisatie geheugen

- Software: schaduwpagina's
- Hardware: Second level address translation
  - Intel: Extended Page Tables
  - AMD: Nested Page Tables

- Compressie van de geheugenvoetafdruk

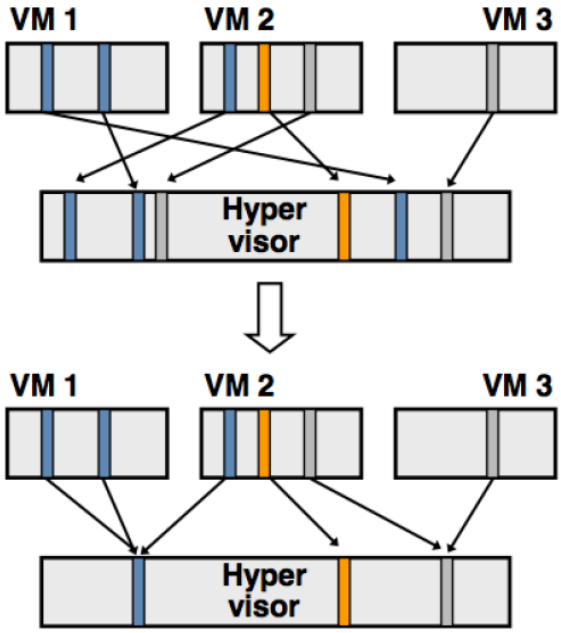
Geheugenvereisten

- Hoeveel geheugen?
  - 2 VM's per core
  - 8 cores per socket
  - 4 socket machine
  - 2 GB of memory per VM
  - 2 x 8 x 4 x 2 = 128 GiB per node!
- Hoeveel identieke pagina's?
  - Guest OS zit 64 keer in het machinegeheugen
  - Nogal wat pagina's bevatten enkel nullen
  - Dezelfde applicaties lopen honderden keren

best9-40

Als we even nadenken over de werkelijke inhoud van het geheugen van een gevirtualiseerde server, dan merken we ten eerste dat de geheugenvereisten zeer groot zijn, en tegelijk dat er niet zeer efficiënt met het intern geheugen omgesprongen wordt doordat dezelfde binaire gegevens verschillende keren opgeslagen worden.

Transparent Page Sharing



best9-41

In principe kunnen de machineframes die precies dezelfde informatie bevatten gemakkelijk gedeeld worden, op voorwaarde dat ze readonly zijn (of gemaakt worden). Mocht er op een bepaald ogenblik toch geschreven worden naar een machineframe, dan kan het altijd ontdubbeld worden (copy-on-write).

Compactie

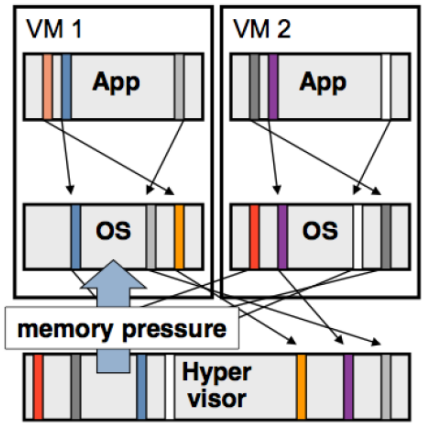
1. Bereken een secure hash (SHA1) voor elk machine-frame
2. indien SHA1(M_1) == SHA1(M_2):
   - Zoek alle pagina's die afbeelden op M_1 en M_2 en beeldt ze af op M1
   - Markeer alle pagina's readonly
   - Geef M2 vrij

- Compactie kan gebeuren door een systeemdraad in de VMM.
- Afhankelijk van de werklast kan dit leiden tot een besparing tot 33% van het fysiek geheugen
- Met copy-on-write kan de compactie ongedaan gemaakt worden.

best9-42

Ballooning

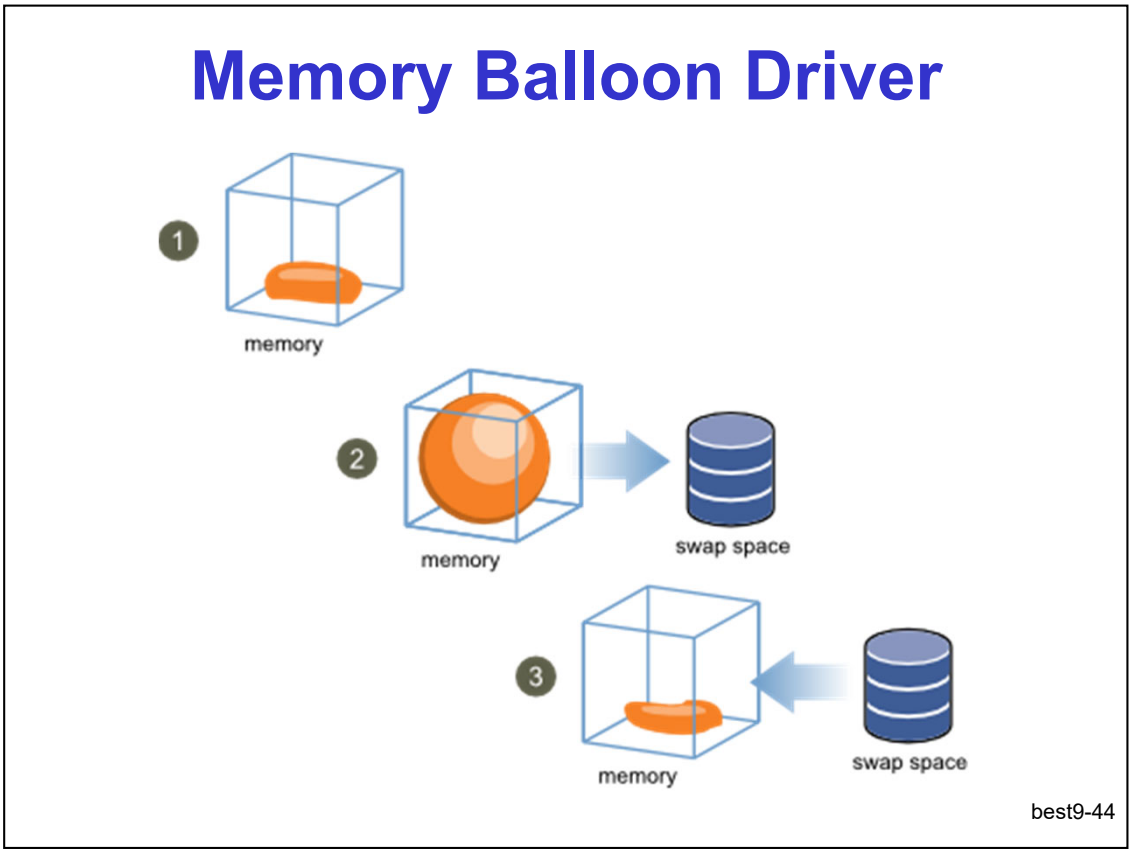
- **Hypervisor wants to reclaim memory**
- **Guest OS is not aware of this**
  - Thinks it owns all physical memory
  - Sits inside its own "box", unaware it's running in a VM or that other VMs are running
- **Goal: make the guest aware so it frees up some of its memory**
- **Solution: artificially create memory pressure inside the VM**
  - "Push" memory pressure from the hypervisor into the VM
  - Use "balloon" driver inside the VM to create memory pressure



best9-43

vmware

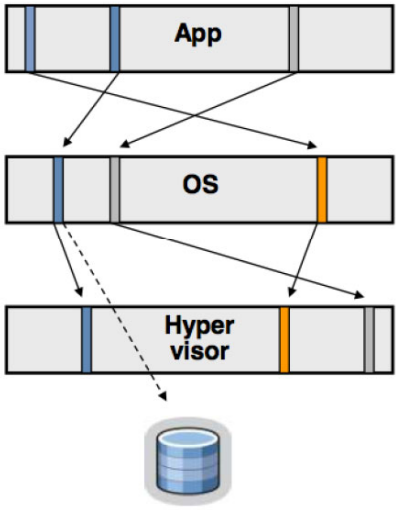
De VMM is verantwoordelijk voor het verdelen van de totale hoeveelheid machinegeheugen over de verschillende virtuele machines. Het besturingssysteem dat in een VM draait zal dit geheugen dan op zijn beurt verdelen over de verschillende processen en het besturingssysteem zelf. Het besturingssysteem gaat er echter vanuit dat het toegang heeft tot alle geheugen van de onderliggende machine en zal zoveel mogelijk van dat geheugen proberen gebruiken om de processen zo efficiënt mogelijk te laten uitvoeren, om bestandscaches te ondersteunen, enz. Zo zal het voorkomen dat de gezamenlijke aanvragen voor machinegeheugen door de verschillende virtuele machines de totale hoeveelheid machinegeheugen overtreffen. De VMM kan uiteraard secundair geheugen gebruiken om op het VMM-niveau virtueel geheugen aan te bieden, maar dat is bijzonder inefficiënt. Beter is het om het Guest OS te verplichten om zijn werklast uit te voeren met minder machinegeheugen. Probleem is echter dat het Guest OS niet weet dat het in een VM draait, en dat we het Guest OS dus ook niet kunnen vragen om zijn geheugenconsumptie te beperken. Een elegante oplossing om toch dit effect te bereiken is om een zgn balloondriver te installeren in het Guest OS. Deze balloondriver wordt aangestuurd door de VMM en alloceert/dealloceert op commando van de VMM blokken geheugen.



Als de balloondriver veel geheugen alloceert, dan zal het Guest OS op zoek gaan naar geheugen, en die pagina's die niet behoren tot de kernwerkverzameling van de processen uitpagineren naar de swapruimte. Op die manier kan het Guest OS zelf beslissen op welke manier het geheugen vrijmaakt. De VMM zou uiteraard ook pagina's kunnen uitpagineren, maar hij heeft geen toegang tot de kennis van het geheugenmanagement van het Guest OS. Daarom is het beter om het Guest OS die taak zelf te laten uitvoeren. De balloondriver zal de gealloceerde machineframes uiteraard afstaan aan de VMM die ze dan ter beschikking kan stellen aan een andere VM.

Swapping

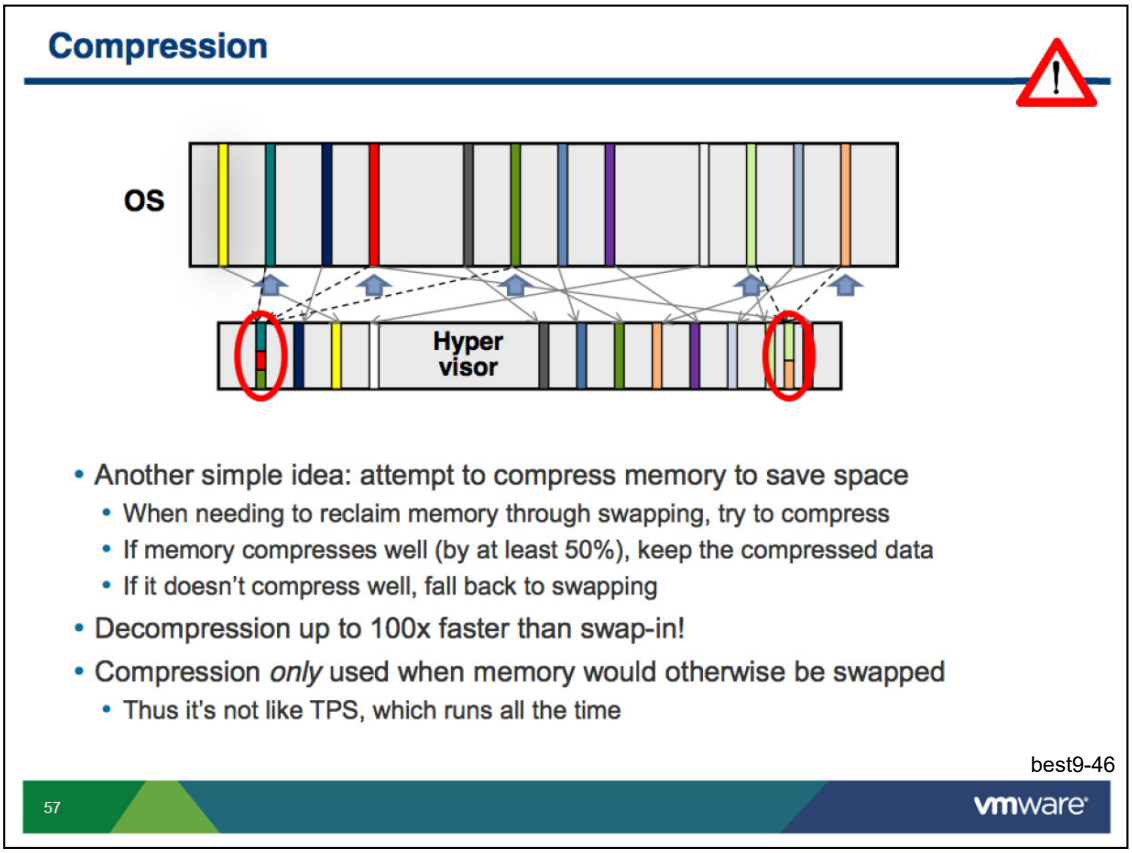
- **Hypervisor can swap VM memory**
  - Swaps out to a per-VM swap file
  - Transparent to the VM
- **Hypervisor swapping is a last resort**
  - Both TPS and ballooning preferred
    - Low overhead for TPS
    - Guest swapping due to ballooning performs better than hypervisor swapping
    - But both take time
  - Hypervisor swapping quickly reclaims memory, but is more expensive overall



best9-45

vmware

De VMM kan uiteraard ook swappen, maar dit is zeer duur, en wordt best zoveel mogelijk vermeden.



Nog een andere techniek is om machineframes niet uit te pagineren, maar om ze te comprimeren. Comprimeren en decomprimeren gaat sneller dan uit- en inpagineren.

Uiteraard is het maar beperkt bruikbaar omdat ook de gecomprimeerde frames nog steeds plaats zullen innemen. Uitgepagineerde frames komen uiteraard 100% vrij.

When To Reclaim Memory *

- **Not memory overcommitted**
  - Never reclaim memory from guest through ballooning or swapping
    - No need to!
    - (Transparent page sharing is always running)
- **Memory overcommitted**
  - Only reclaim once free physical memory drops below threshold
    - Start ballooning when memory starts to fall toward threshold
    - Start swapping as memory nears/passes threshold

* Assumes no VM or resource pool memory limits are set!

Understanding Virtualization Memory Management Concepts, K. Colbert
http://www.vmworld.com/docs/DOC-4643

best9-47

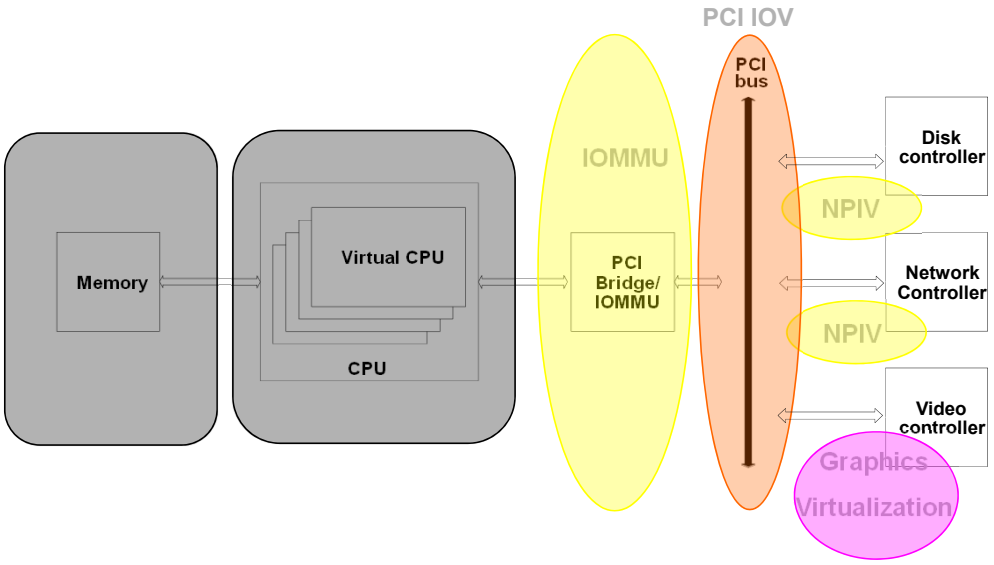
Overzicht

- Definities
- Klassieke virtualiseerbaarheid
- Virtualisatie
  - CVE
  - Geheugen
  - IO
    - Software
    - Hardware
- Migratie

best9-48

Het laatste te virtualiseren onderdeel is het IO-systeem.

x86 platform virtualisatie



best9-49

Bij IO-virtualisatie zijn er drie manieren om tewerk te gaan:

- Er is de volledige software-emulatie van de IO-instructies
- Er is paravirtualisatie van de drivers
- Er is hardwareondersteuning voor IO-virtualisatie

Software-emulatie

- Elke in/out-instructie veroorzaakt een vmexit die geëmuleerd moet worden. Een device driver bevat vaak een opeenstapeling van in/out instructies. Dit leidt tot een complexe, kwetsbare en trage virtualisatieoplossing. Binair herschrijven kan het aantal world switches doen dalen.
- DMA maakt gebruik van fysieke adressen (maakt geen gebruik van de MMU). In een VM zijn de fysieke adressen pseudo-fysiek. Zelfs na vertaling is er een probleem omdat DMA naar alle fysieke adressen kan lezen/schrijven (ook van een andere VM)

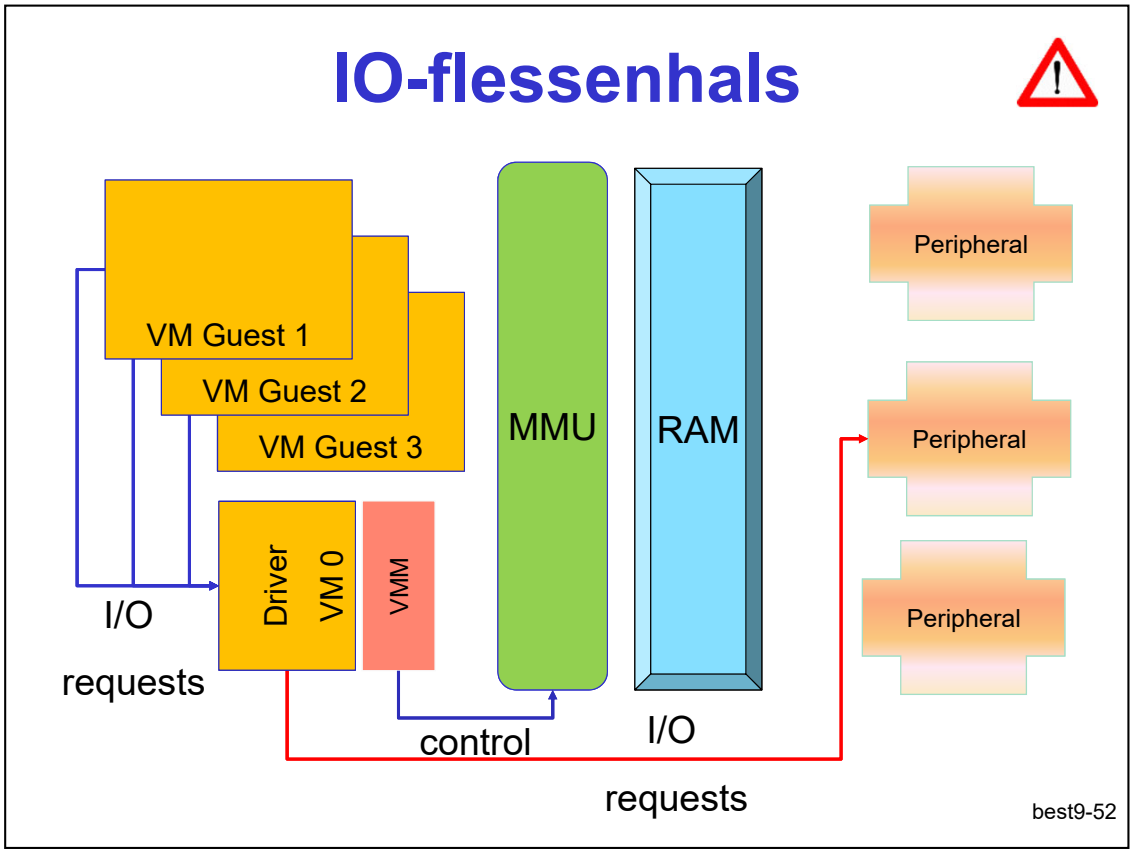
IO is het inefficiëntste onderdeel van virtualisatie

best9-50

Paravirtualisatie voor drivers

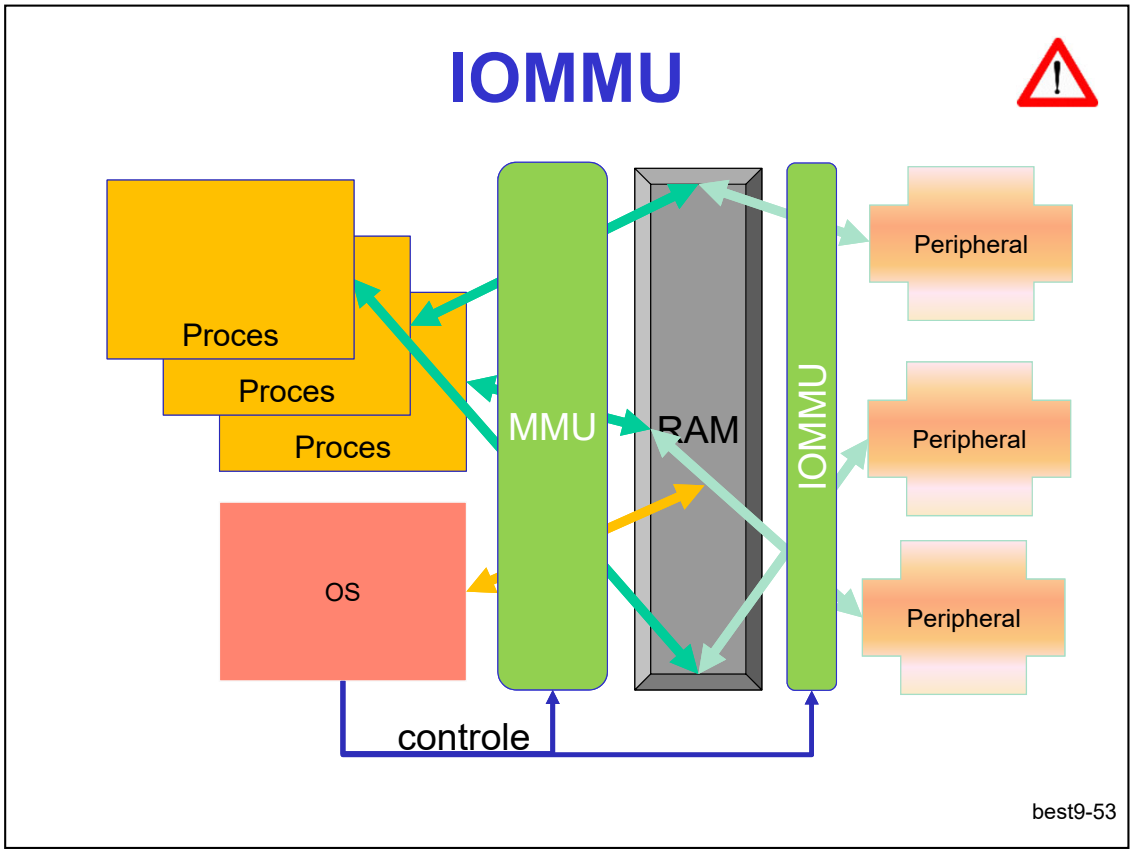
- De oorspronkelijke device drivers kunnen vervangen worden door gespecialiseerde device drivers die horen bij de VMM en bestaan uit een aantal hypercalls
- De echte toegang tot de fysieke hardware kan door de VMM zelf gebeuren, of gedelegeerd worden naar een speciale VM die alle IO voor haar rekening neemt (management VM of een speciale driver-VM)

best9-51

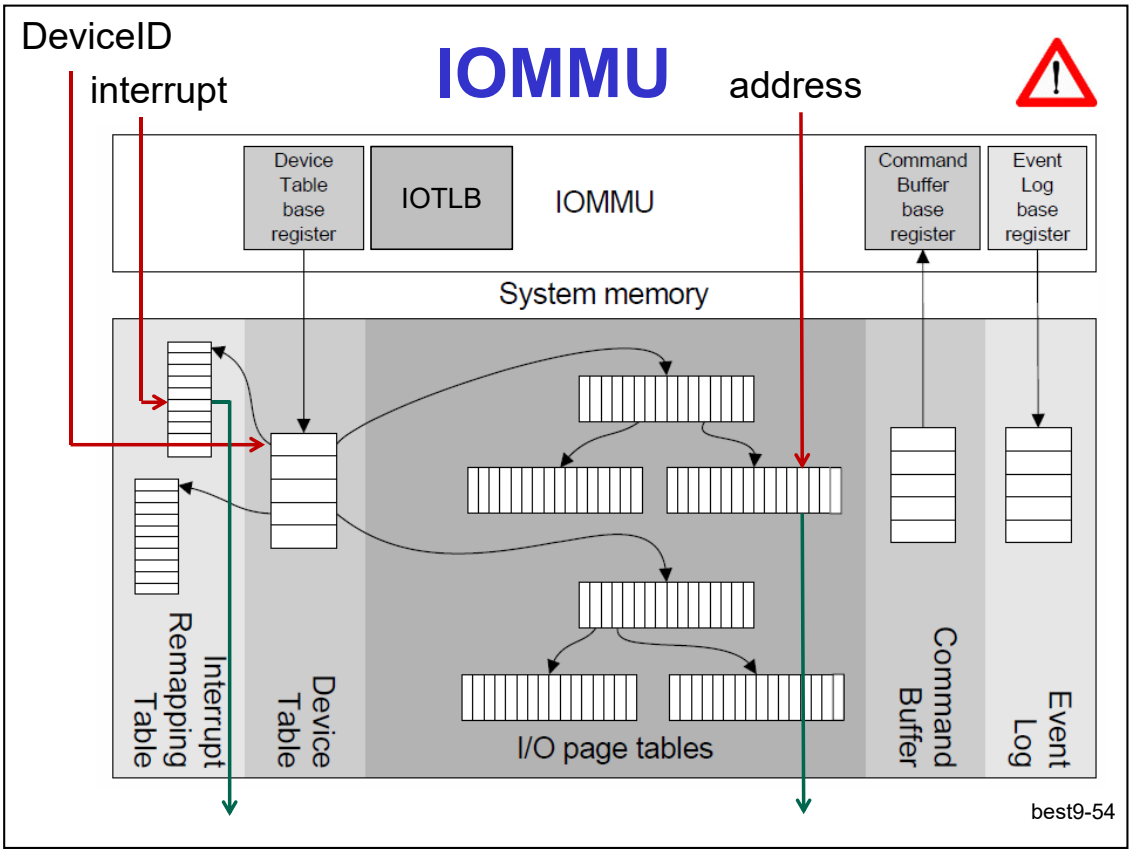


In het geval dat de IO gebeurt via een driver-VM die alle IO voor zijn rekening neemt ontstaat er snel een flessenhals, in het bijzonder voor die randapparaten die snel moeten zijn zoals secundaire opslag en netwerkcommunicatie.

Ideaal moeten deze randapparaten rechtstreeks door de VM kunnen worden aangestuurd, maar dan wel met de garantie dat ze nog steeds kunnen worden gedeeld, en dat ze 100% geïsoleerd zijn van de andere VM's. Dit kan enkel via hardwareondersteuning: IOMMU



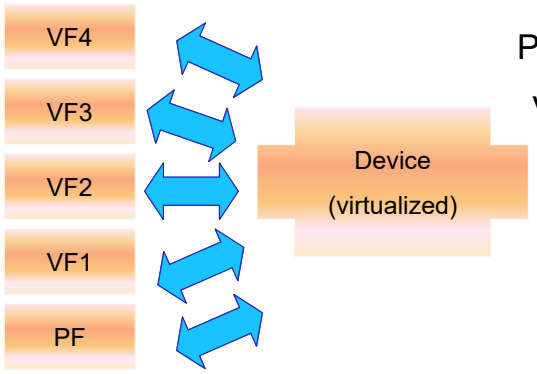
Naast de MMU die zorgt voor adresvertaling tussen de CVE en de RAM en de isolatie tussen de processen garandeert, bestaat er ook een IOMMU die op een gelijkaardige manier kan zorgen voor de vertaling van adressen tussen de randapparaten en de RAM. Concreet kan dit betekenen dat een proces een randapparaat configureert om de gegevens via DMA op te halen of achter te laten op een gegeven logisch adres. Telkens wanneer het randapparaat wil lezen/schrijven naar de RAM zal dat logisch adres vertaald worden naar het corresponderende fysieke adres. De paginatabellen voor de IOMMU kunnen, maar hoeven niet, gemeenschappelijk zijn met die van de MMU. Naast de vertaling van de adressen, kan de IOMMU ook de onderbrekingen die gegenereerd worden door de randapparaten vertalen naar een set van onderbrekingen die dan bij de processor afgeleverd wordt. De IOMMU staat, net zoals de MMU onder de controle van het besturingssysteem.



Intern bestaat de IOMMU uit een devicetabel met 64 Ki elementen van 256 bit. Elk element bevat naast een aantal controlebits een pointer naar een set van paginatabellen voor de adresvertaling, en naar een zgn Interrupt Remapping Table. Deze laatste geeft per onderbreking aan op welke manier ze moet aangeboden worden aan de processor. Verder bevat de IOMMU ook een IOTLB waar recente vertalingen gecachet worden, ontvangt hij commando's van het besturingssysteem via het commandobuffer, en houdt hij een log bij.

Elk adres/onderbreking is afkomstig van een randapparaat en krijgt een 16 bit DeviceID mee. Bij PCI is dat een bus-device-function (BDF) identificatie. Bij andere protocols kan dit een andere identificatie zijn. De vertaling begint met het ophalen van het devicetabelelement aangewezen door het DeviceID.

HW Device Virtualization



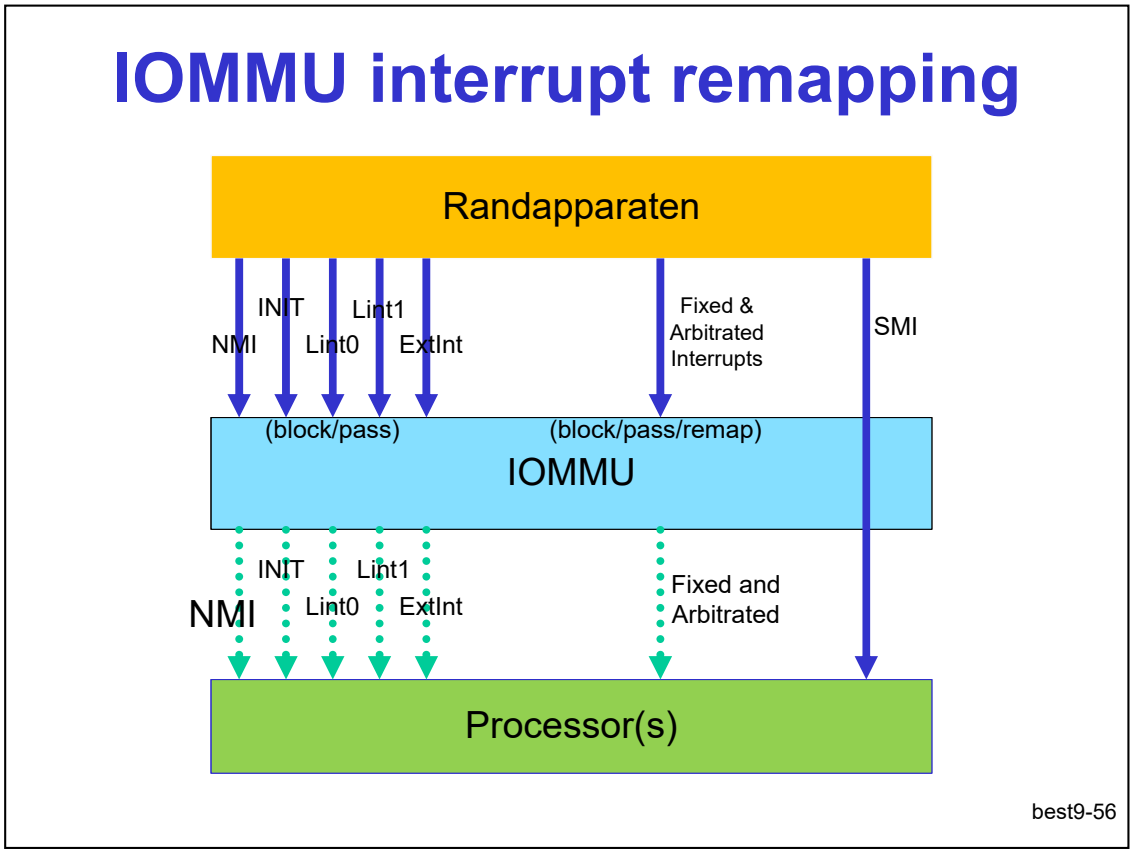
PF: Physical Function

VF: Virtual Function

- Device implements many virtual functions
- Each function assigned a unique Bus-Device-Function tuple (BDF)
- Each function can be assigned to a separate guest VM
- Device tags DMA and interrupt transactions with BDF
- Each function can be isolated and access only the assigned guest VM

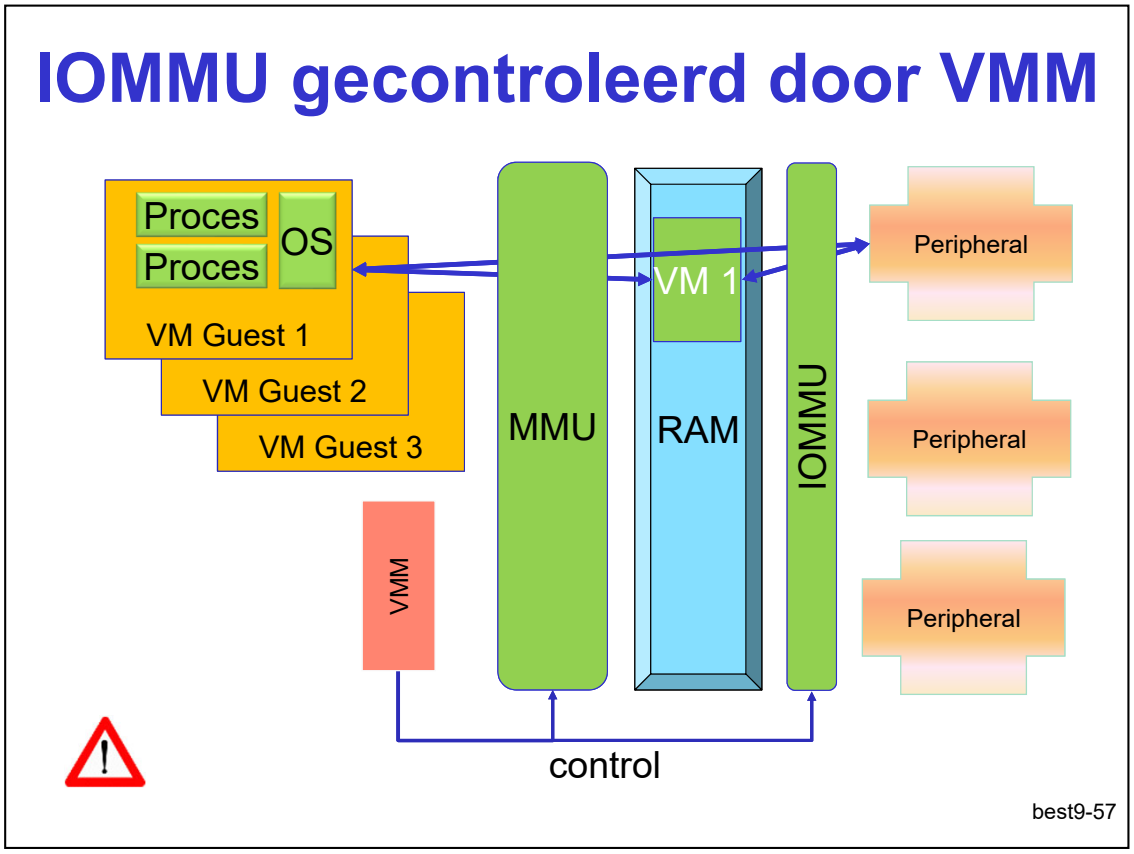
best9-55

Randapparaten kunnen ook intern gevirtualiseerd worden. Dat betekent dat het randapparaat zich kan voordoen als verschillende randapparaten ofschoon er maar 1 fysiek exemplaar is. Het fysieke exemplaar wordt de Physical Function genoemd. De virtuele kopieën heten de Virtual Functions. Alle functies krijgen een eigen DeviceID. Via een IOMMU kunnen bepaalde functies toegewezen worden aan bepaalde VM's zonder dat het risico bestaat dat ze interfereren met de werking van een andere VM.



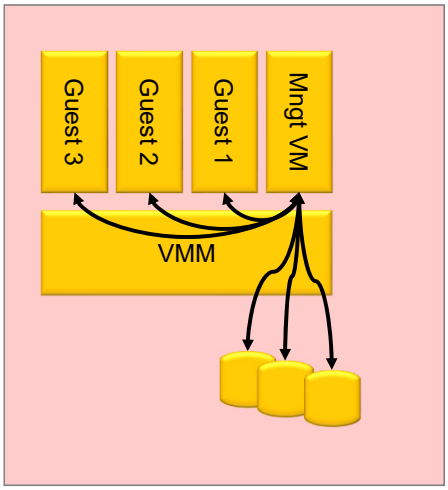
Ook de onderbrekingen die van de randapparaten komen kunnen via de IOMMU vertaald worden alvorens ze door de processor te laten afhandelen. Bijzondere onderbrekingen zoals NMI worden al dan niet doorgelaten. De reguliere onderbrekingen kunnen tegengehouden worden, doorgelaten worden, of afgebeeld worden op een andere onderbreking. De enige uitzondering is de SMI (system management interrupt) die de processor in het hoogste privilegeniveau moet brengen. Deze is zo belangrijk voor het systeem dat hij altijd doorgelaten wordt.

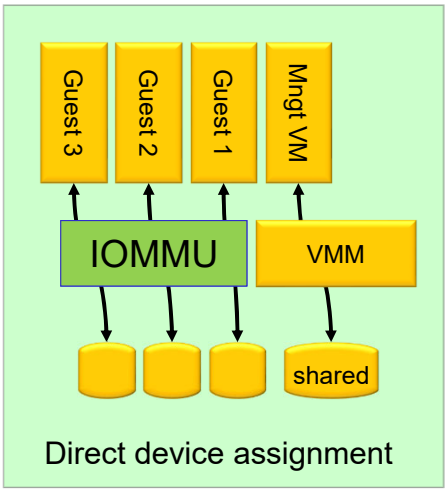
De Interrupt Remapping Table bestaat uit 8ki elementen van 32 bit groot. Ze vertaalt de binnenkomende onderbrekingen naar uitgaande onderbrekingen.



Als we de IOMMU nu onder de controle van de VMM plaatsen, dan kan deze er ook voor zorgen dat de communicatie tussen de randapparaten en de diverse VM's op een gecontroleerde manier gebeurt.

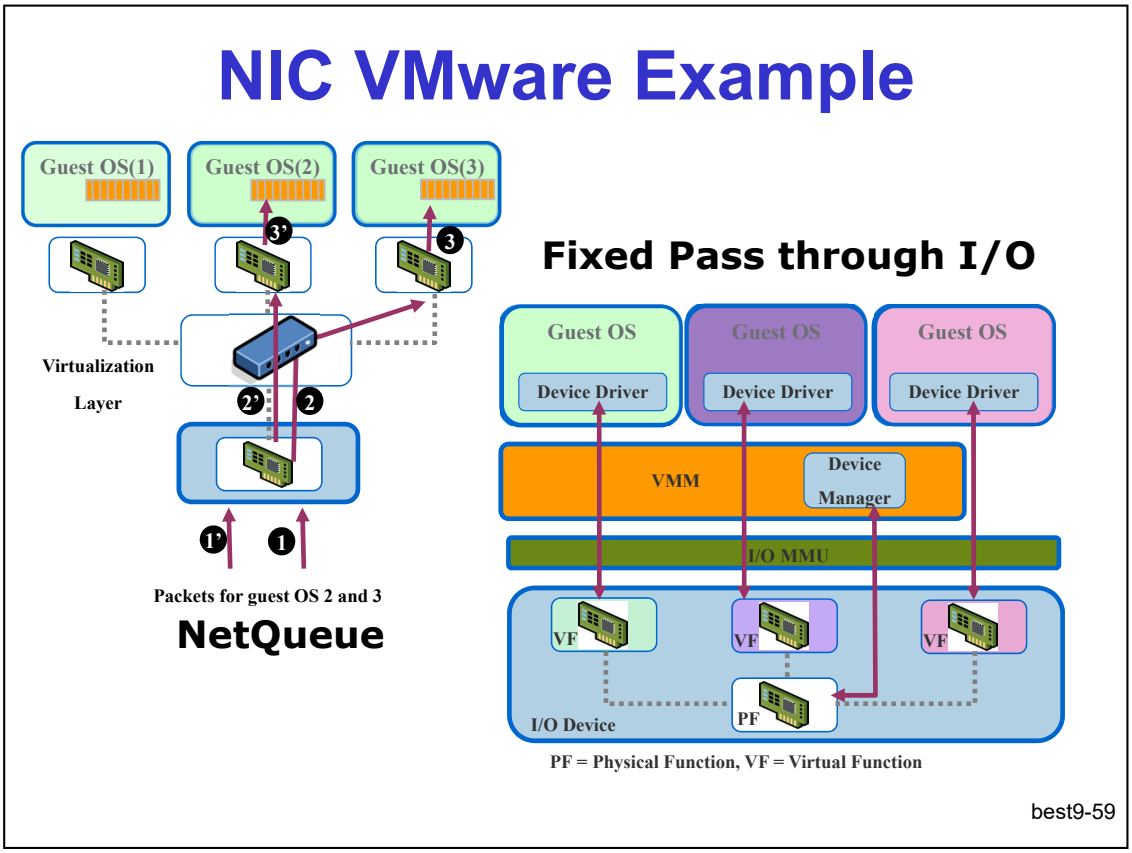
Hardware IO : IOMMU





best9-58

Indien we optimaal gebruik maken van de mogelijkheden van de IOMMU, dan kan de VMM de IOMMU zodanig configureren dat de meeste randapparaten rechtstreeks gekoppeld kunnen worden aan een bepaalde VM en het besturingssysteem dat in die VM draait zonder veel aanpassingen gebruik kan blijven maken van de randapparaten. Hiermee is de laatste prestatieflessenhals van de VMM eigenlijk weggewerkt.



Een voorbeeld van het gebruik van virtual functions in VMware.

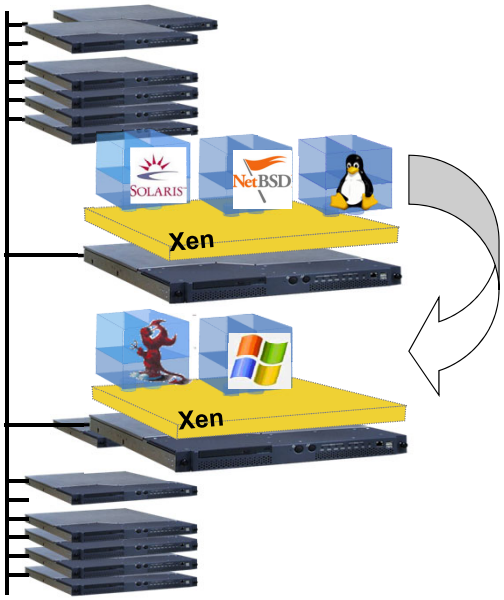
Overzicht

- Definities
- Klassieke virtualiseerbaarheid
- Virtualisatie
  - CVE
  - Geheugen
  - IO
    - Software
    - Hardware
- Migratie

best9-60

Een aspect dat nog niet aan bod gekomen is, is het migreren van virtuele machines. Virtuele machines zijn volledig geïsoleerd van de rest van de activiteiten van een computersysteem. Dit laat toe om ze te migreren van het ene computersysteem naar het andere.

Rationale voor VM migratie

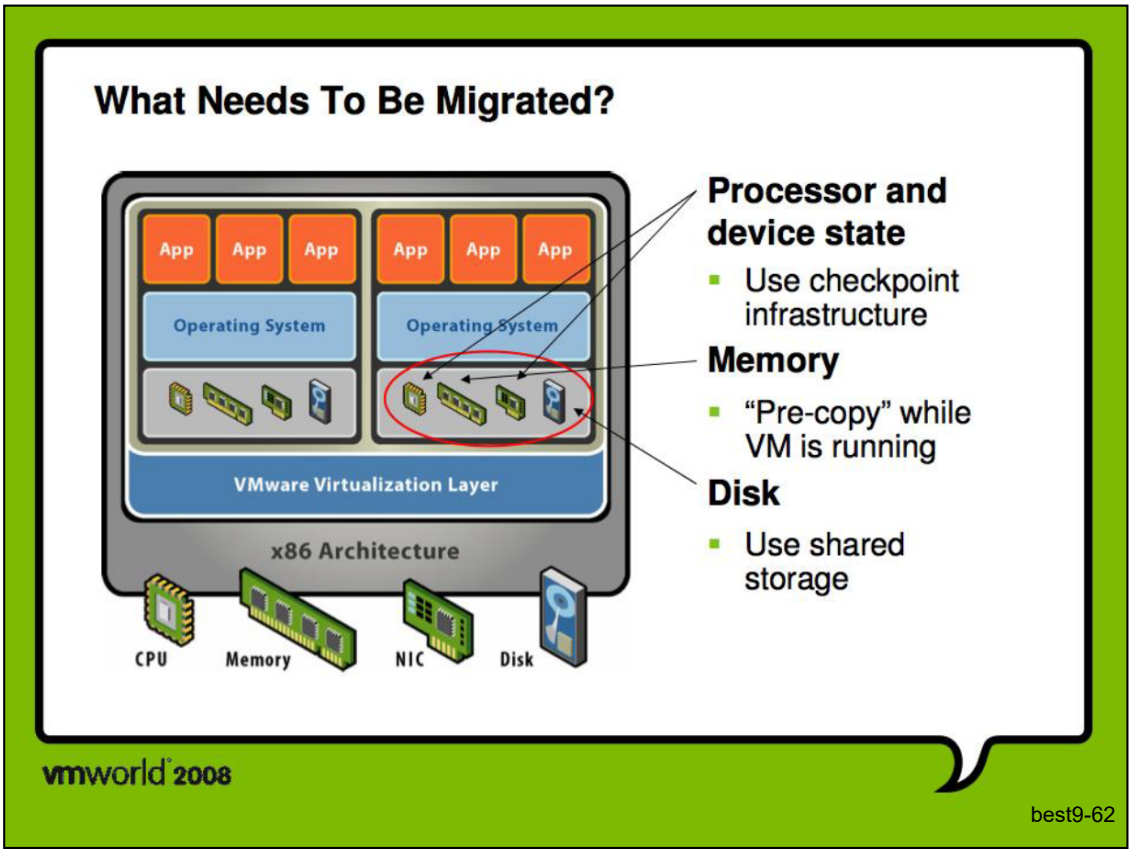


VM-migratie faciliteert:

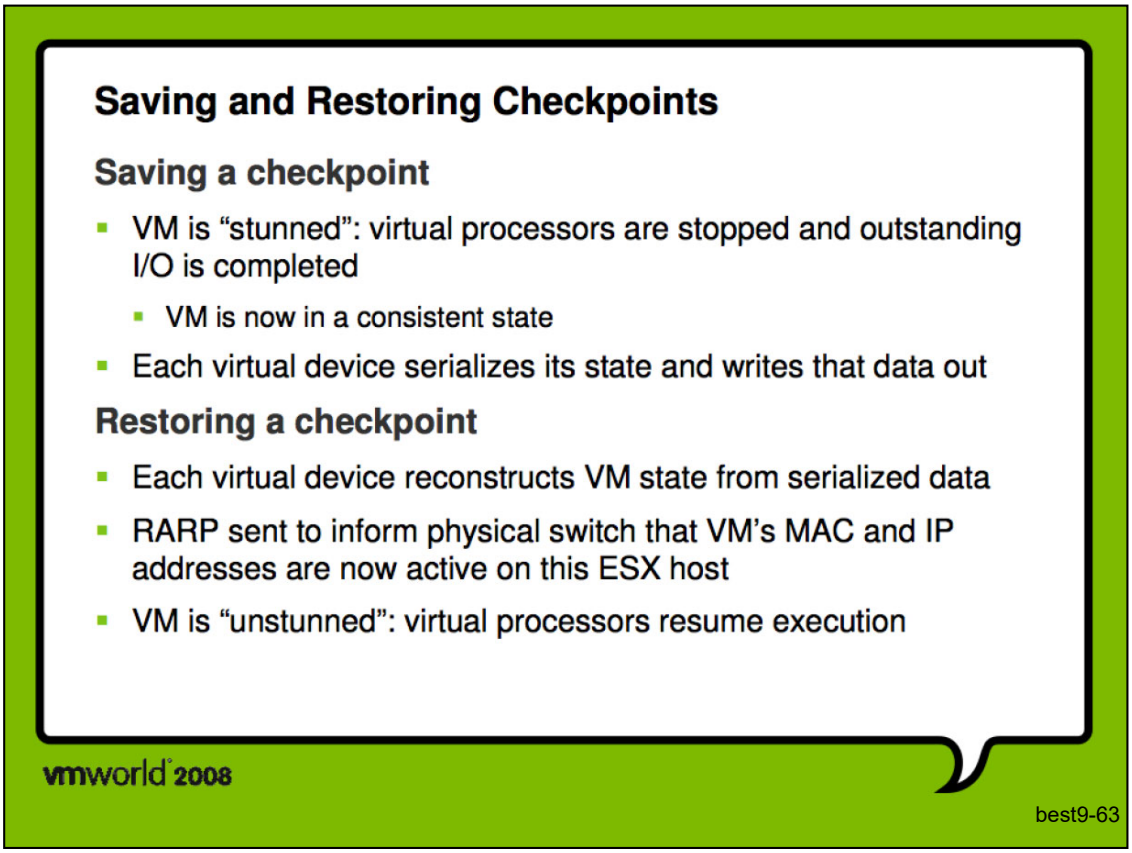
- Hoge beschikbaarheid
  B.v. Machineonderhoud
- Belastingsspreiding

best9-61

Op deze figuur staan verschillende servers getekend die VM's uitvoeren met verschillende besturingssystemen. VM-migratie maakt het mogelijk dat een werkende VM tijdens de uitvoering van het ene naar het ander computersysteem verplaatst wordt. De uitdaging hierbij is om ervoor te zorgen dat de dienstverlening die door de VM aangeboden wordt zo kort mogelijk onderbroken wordt en dat de migratie zelf niet te veel interfereert met de werking van de andere VM's.



Om een VM te migreren, moeten we haar toestand vastleggen en overbrengen naar een andere machine. Die toestand bestaat uit de toestand van de processor en van de randapparaten, de toestand van het geheugen, en de toestand van de secundaire opslag. Voor de secundaire opslag veronderstellen we dat deze gedeeld wordt door de twee computersystemen zodat deze niet moet gekopieerd worden. Het kopiëren van een schijf zou sowieso heel lang duren. De grootste overblijvende opdracht is het migreren van de toestand van het geheugen.



Een naïeve manier om een VM te migreren is het maken van een zgn checkpoint, en dit checkpoint te herstellen op een andere machine. Het maken van een checkpoint veronderstelt dat de VM eerst gestopt wordt in een consistente toestand en dat deze toestand bewaard wordt. Op de bestemming worden alle componenten terug in de juiste toestand gebracht, en kan de werking verdergezet worden.

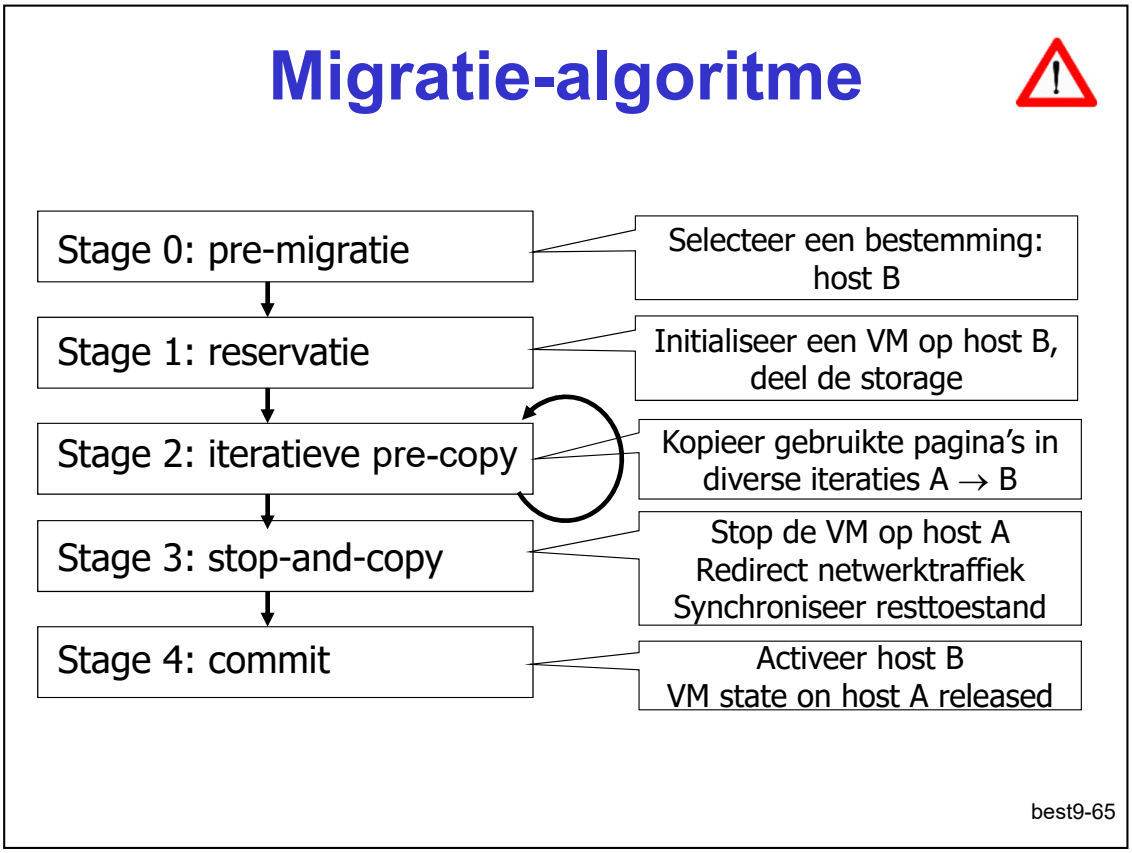
Uitdagingen bij migratie

- VM's hebben een grote interne toestand
- Sommige VM's hebben soft real-time vereisten
  - b.v. webservers, databanken, gameservers

  ➔ **Minimaliseer onbeschikbaarheid**
- Migratie vereist extra systeemmiddelen (geheugen, bandbreedte)

  ➔ **Beperk de extra belasting veroorzaakt door de migratie**

best9-64

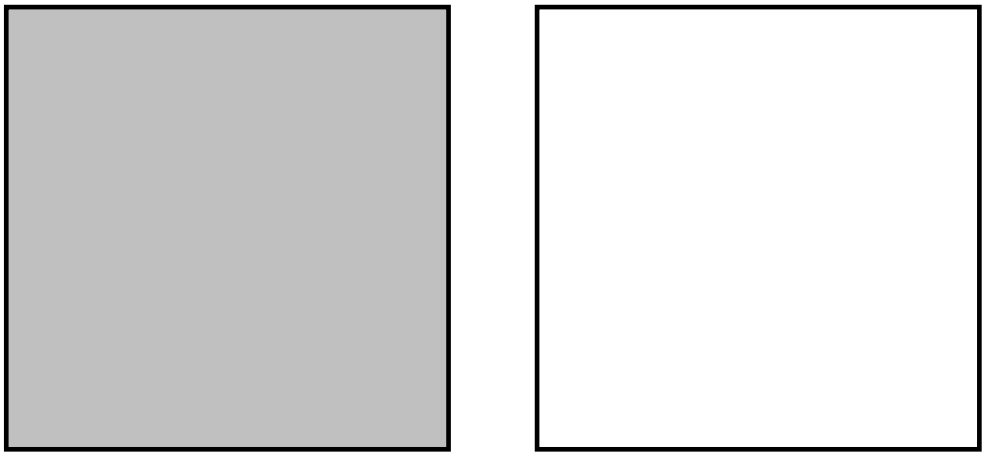


Het migratiealgoritme heeft twee doelstellingen:

- De verstoring die veroorzaakt wordt door de migratie beperken. Dit kan gebeuren door de toestand geleidelijk aan door te sturen, en daarbij geen overmatig beslag te leggen op de bandbreedte van het netwerk, de schijf, enz.
- De onbeschikbaarheid van de dienstverlening zo kort mogelijk te houden.

Uiteraard zijn deze twee doelstellingen niet compatibel. Het geleidelijk aan kopiëren, zal ervoor zorgen dat dit langer duurt. Daarom heeft men ervoor gekozen om tijdens het kopiëren de bron-VM gewoon verder te laten werken, en in verschillende iteraties de inmiddels terug aangepaste toestand terug te kopiëren. Bij elke iteratie wordt de te versturen toestand kleiner zodat de tijd om deze te versturen ook korter wordt, en dus ook het tijdsvenster waarin die toestand opnieuw kan wijzigen. Hierna wordt het algoritme in meer detail uitgelegd.

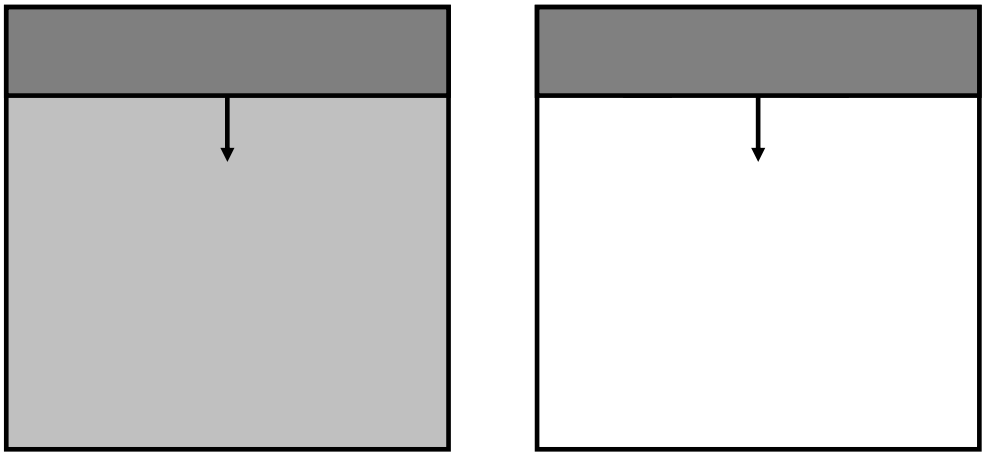
Pre-Copy Migratie: Ronde 1



best9-66

De bron-VM staat links, de doel-VM staat rechts.

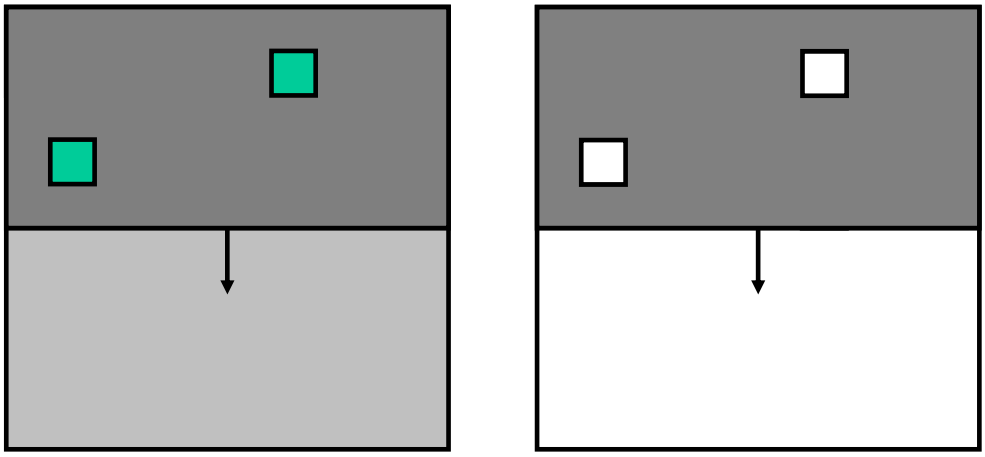
Pre-Copy Migratie: Ronde 1



best9-67

De pre-copy begint pagina per pagina te kopiëren naar de bestemming. De gekopieerde pagina's worden in de bron-VM readonly gemaakt.

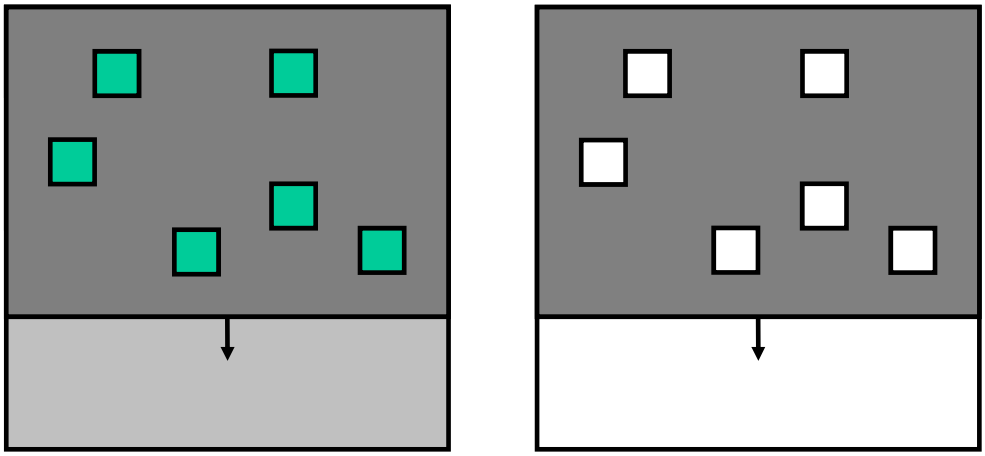
Pre-Copy Migratie: Ronde 1



best9-68

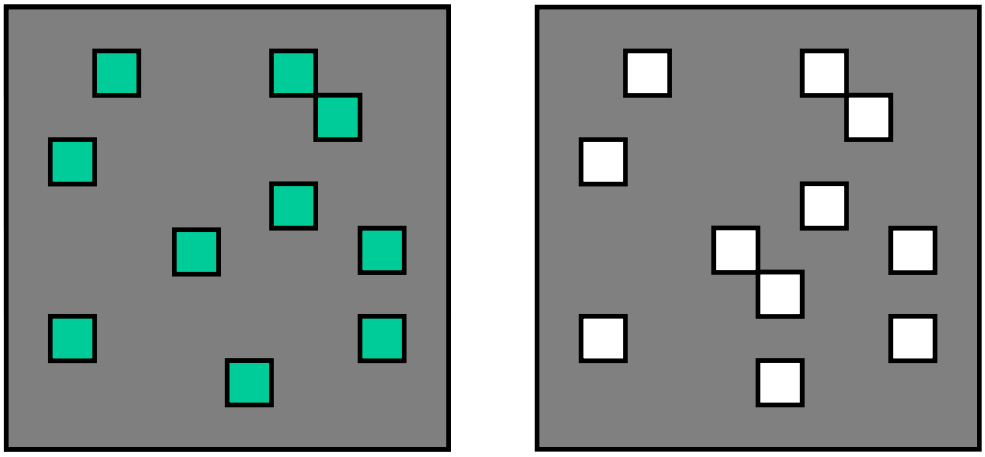
Van zodra de nog werkende bron-VM probeert te schrijven op een reeds gekopieerde pagina, wordt dit genoteerd. Deze pagina zal opnieuw gekopieerd moeten worden.

Pre-Copy Migratie: Ronde 1



best9-69

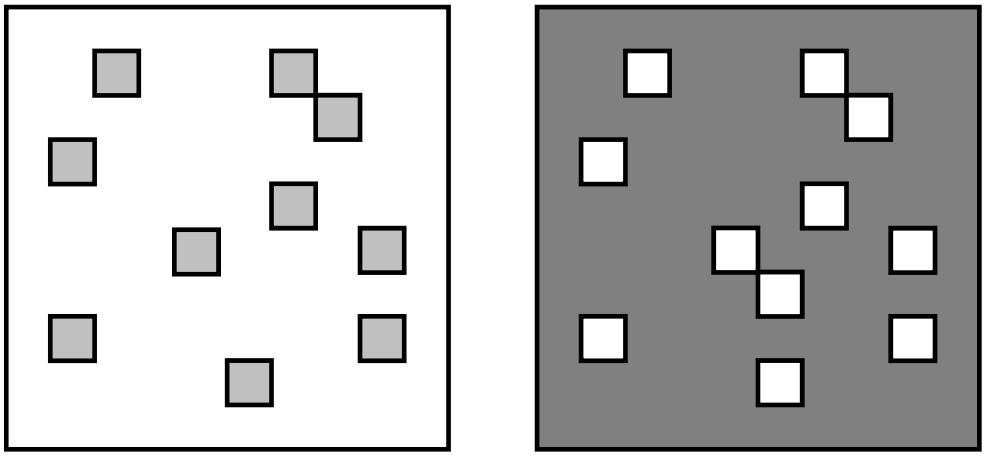
Pre-Copy Migratie: Ronde 1



best9-70

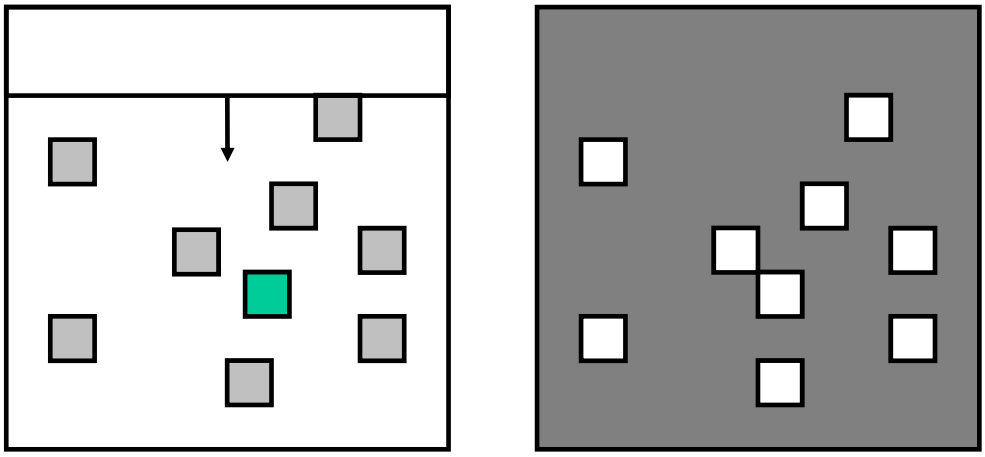
Op het einde van de eerste ronde hebben we het meeste van de toestand al gekopieerd, maar er moeten nog een aantal pagina's nagestuurd worden.

Pre-Copy Migratie: Ronde 2



best9-71

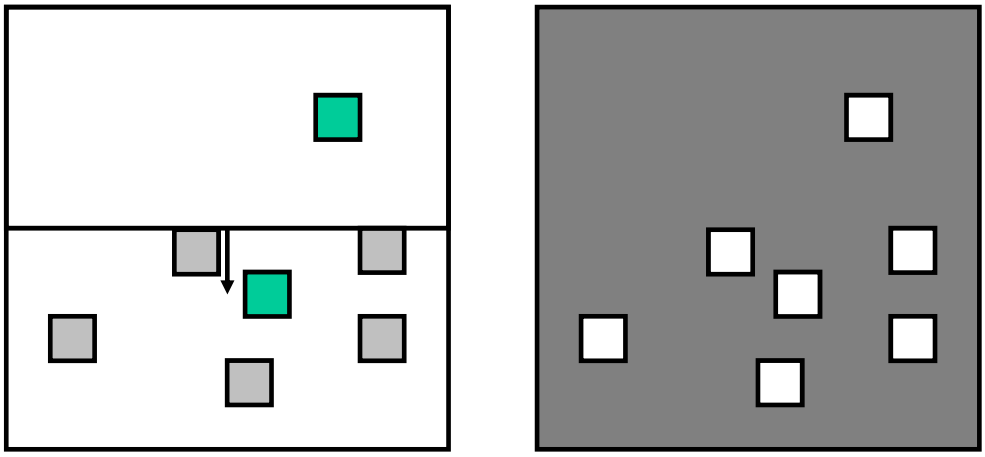
Pre-Copy Migratie: Ronde 2



best9-72

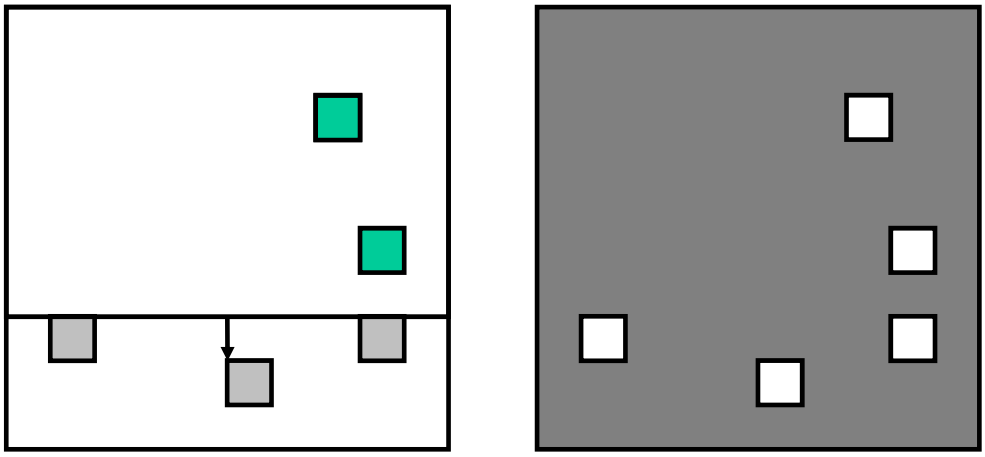
Tweede ronde start – maar gaat deze keer sneller vooruit. Verstuurde pagina's worden opnieuw readonly.

Pre-Copy Migratie: Ronde 2



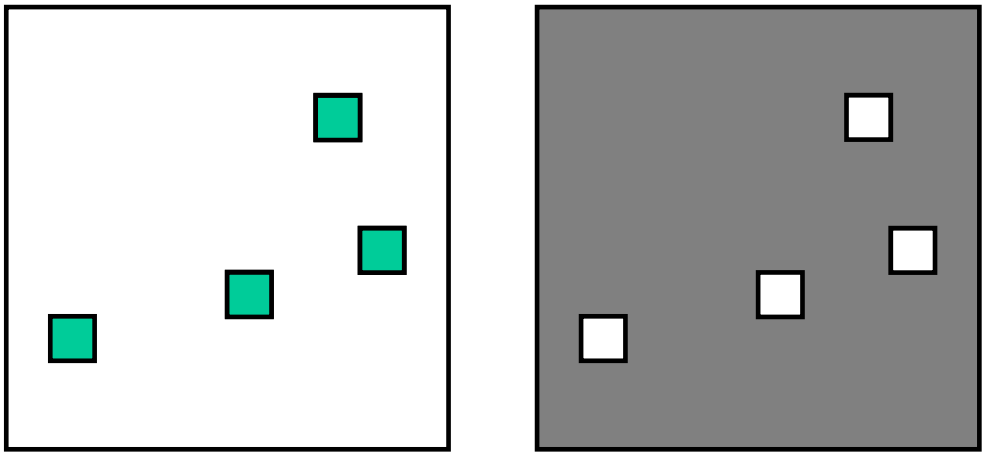
best9-73

Pre-Copy Migratie: Round 2



best9-74

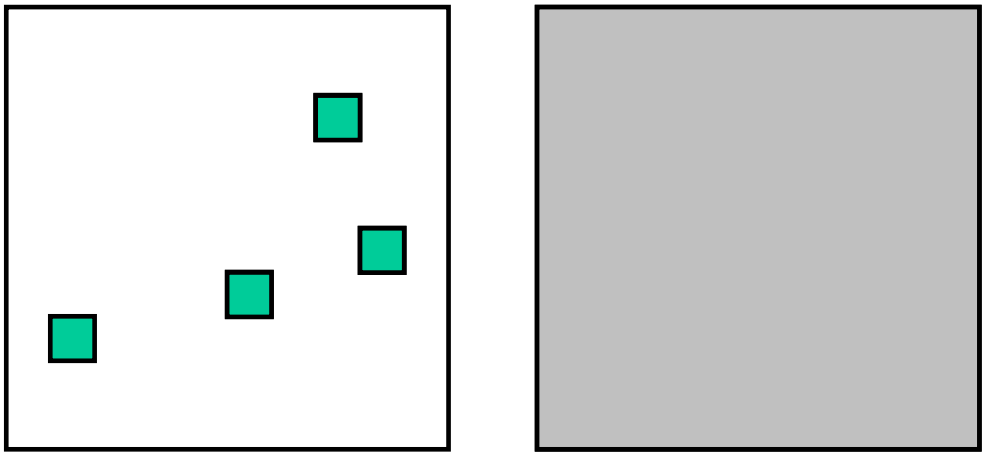
Pre-Copy Migratie: Ronde 2



best9-75

Op het einde van de tweede ronde blijven er nog vier veranderde pagina's over.

Pre-Copy Migratie: checkpoint copy



best9-76

Nu wordt een checkpoint van de processor, randapparaten en veranderd geheugen genomen en doorgestuurd. De VM op de bestemming is nu klaar om starten.

Impact van migratie

- Systeemmiddelen die ingezet worden voor migratie moeten beperkt blijven:
  - “Dirty logging” kost ongeveer 2-3%
- B.v. eerste kopieeriteratie aan 100Mb/s, langzaam verhogend voor de volgende iteraties
  - Kortere kopieeroperaties
  - Ondertussen minder aangepaste pagina’s
  - Volgende iteratie wordt kleiner

best9-77

Simple Pre-copy Example *

Migrating a 2 GB VM:

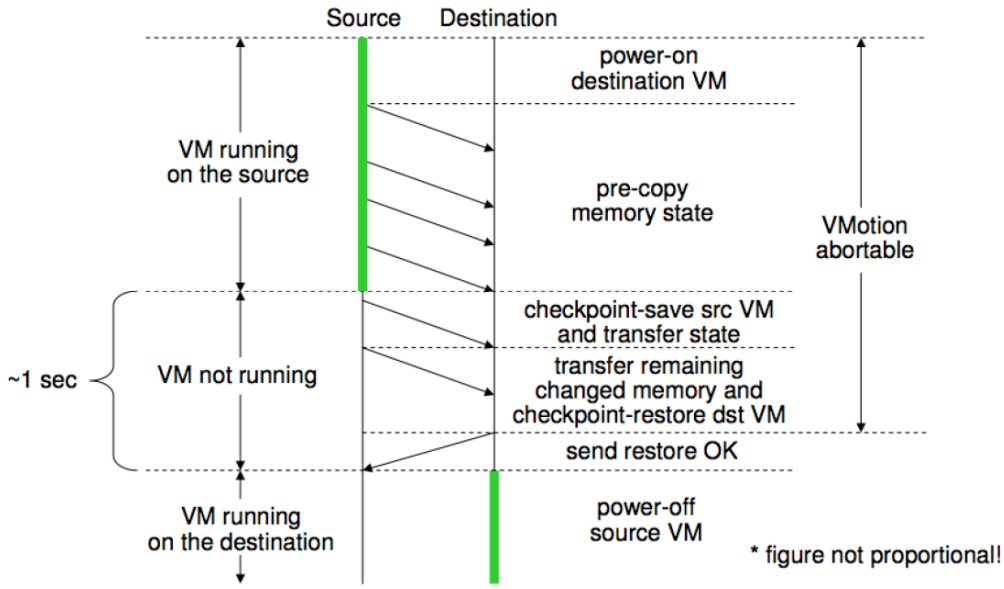
Pre-copy iteration	Memory to send	Time to send	Changed memory
1	2048 MB	16 s	512 MB
2	512 MB	4 s	128 MB
3	128 MB	1 s	32 MB
4	32 MB	0.25 s	8 MB

8 MB of changed memory can be sent in ~0.06 seconds, so OK to stop pre-copy

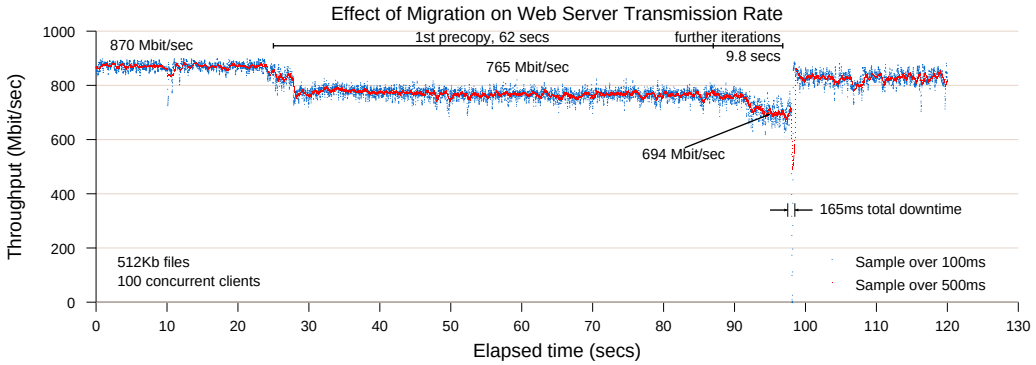
* Assumes constant memory workload and network bandwidth

best9-78

Putting It All Together

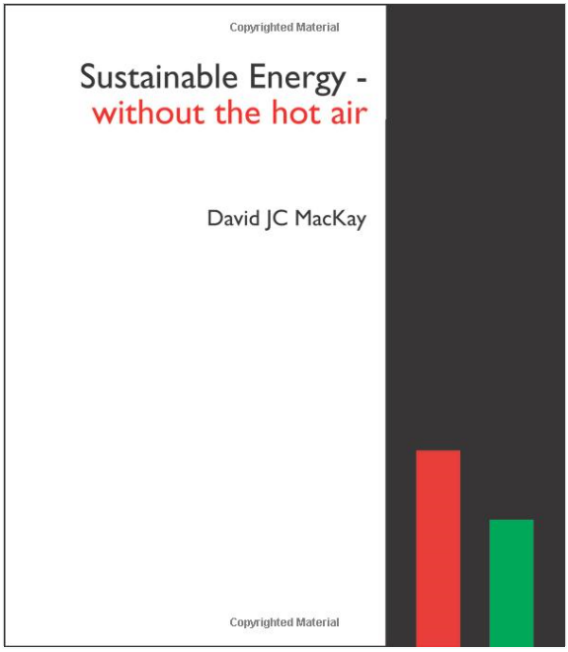


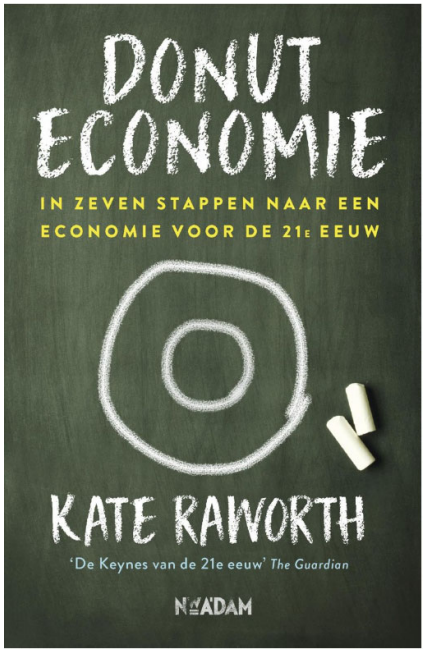
Webserver Migratie



best9-80

Het effect van de migratie van een webserver wordt hier gevisualiseerd. Het totale proces neemt iets meer dan 1 minuut in beslag. Gedurende de migratieperiode is er beperkte daling van de throughput merkbaar. De eigenlijke transitie neemt slechts 165 ms in beslag en is in de praktijk niet merkbaar voor de gebruiker. Deze technologie is bruikbaar om VM's in data centers te migreren van de ene naar de andere server om de belasting te spreiden, om machines tijdelijk vrij te maken, om aan onderhoud te doen, om de werklast tijdens de rustige periodes te consolideren op een kleiner aantal servers en op die manier energie uit te sparen, enz.


Copyrighted Material
Sustainable Energy -
without the hot air
David JC MacKay
Copyrighted Material


DONUT
ECONOMIE
IN ZEVEN STAPPEN NAAR EEN
ECONOMIE VOOR DE 21E EEUW
KATE RAWORTH
'De Keynes van de 21e eeuw' The Guardian
NIEUWEADAM

best9-81