UNIVERSIDADE DE ÉVORA

# Virtualização e Contentores (*Containers*)

# Motivação

O funcionamento de sistemas assenta em abstrações que organizam o funcionamento de diversas componenentes:

- Interpretadores
- Memória
- Comunicação

Virtualização é um pilar da computação em nuvem que simplifica a gestão de recursos físicos relacionados com aquelas três componentes

O estado de uma Máquina Virtual (***V****irtual* ***M****achine* VM) em execução num *virtual machine monitor* (VMM) pode ser gravado e migrado para outro servidor, para distribuição de carga

Virtualização permite ao utilizador o uso de ambiente que lhe é familiar, em vez de obrigar ao uso de outros idiossincráticos

# Motivação

A virtualização de recursos em nuvem é importante para:

- Segurança, por permitir compartimentar/isolar serviços que executam na mesma plataforma/hardware
- Desempenho e fiabilidade, permitindo a migração de aplicações de uma plataforma para outra
- Desenvolvimento e gestão de recursos oferecidos pelos fornecedores (CSPs)
- Isolamento ou compartimentação do desempenho

UNIVERSIDADE DE ÉVORA

# Virtualização

Simula a interface com um recurso físico mediante:

- **Multiplexing**
  - criar múltiplos objetos a partir de uma só instância de objeto físico. Exemplo: um processador é multiplexado por vários threads/processos.
- **Agregação**
  - criar um recurso virtual a partir de vários objetos físicos/reais. Exemplo: vários discos em RAID vistos como um só disco
- **Emulação**
  - formar um objeto virtual, do tipo A, a partir de um objeto físico de tipo diferente B; Exemplo: um disco emula a memória RAM.
- **Multiplexing e emulação**
  - como exemplo da memória virtual com paginação para memória real ou disco; um endereço virtual emula um endereço real

# Organização em camadas *(Layering)*

*Layering* – organização por camadas como abordagem para lidar com a complexidade.

Minimiza interações entre subsistemas de um sistema complexo

- Simplifica a descrição dos subsistemas
- cada subsistema é abstraído através de interfaces para outros subsistemas
- É possível desenhar, implementar, e modificar cada subsistema de forma independente.

*Layering* num sistema computacional:

- Hardware
- Software
  - Sistema operativo
  - Bibliotecas
  - Aplicações

# Interfaces

Instruction Set Architecture (ISA)

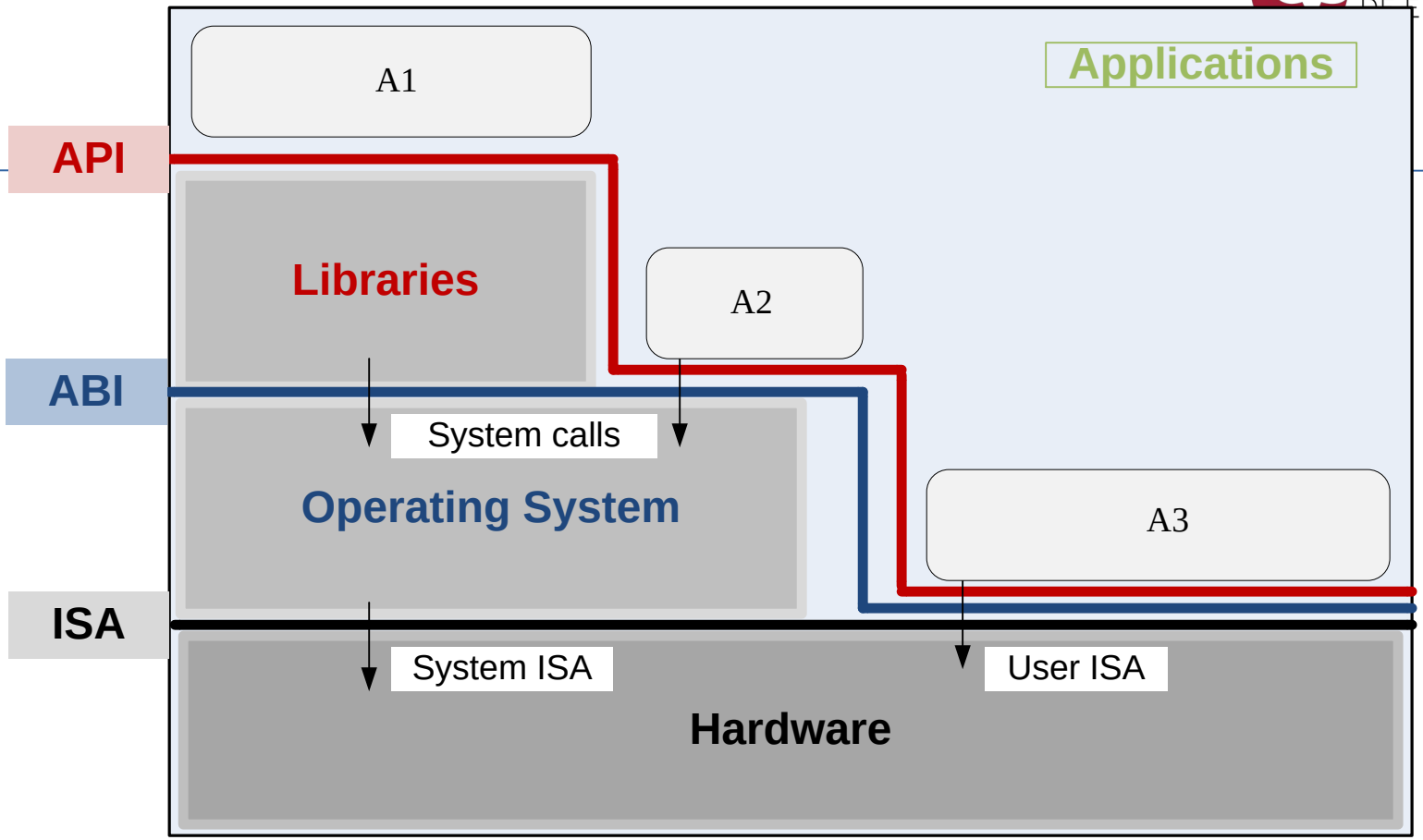
- na fronteira entre hardware e software

Application Binary Interface (ABI)

- Permite que aplicações e biblioteca acedam ao hardware
- Não inclui instruções previlegiadas, invocando system calls em alternativa

Application Program Interface (API)

- Conjunto de instruções disponibilizadas como interface para as aplicações
- Permite que aplicações acedam à ISA
- Inclui chamadas a bibliotecas de HLL, que poderão invocar system calls




Application Programming Interface, Application Binary Interface, and Instruction Set Architecture . An application uses library functions (A1), makes system calls (A2), and executes machine instructions (A3).

# Portabilidade do código

Binários criados por um compilador para um SO e ISA específicos não são portáveis

Ao compilar um programa escrito numa HLL (LP de alto nível) é possível conseguir que seja executado numa VM, e a portabilidade pode ser conferida pelo próprio ambiente da VM, com tradução dinâmica entre o binário original (para a arquitetura do *guest OS*) e a ISA do sistema anfitrião.

# Virtual Machine Monitor (VMM / hypervisor)

Particiona/divide os recursos de um computador para uma ou mais Vms. Permite que vários SOs executem em simultâneo numa só plataforma de hardware.

VMM permite

- A partilha da plataforma por vários serviços
- Migração *live* – transferência de uma aplicação em funcionamento de um servidor para outro
- Alteração de um sistema mantendo retrocompatibilidade
- Facilita separação do ambiente de execução de vários sistemas, contribuindo para a segurança

# Hypervisor

Classificados num de dois tipos:

**- Type 1 hypervisor** (VMM de tipo 1) executa diretamente numa certa plataforma (como programa de controlo de SOs). Um "guest" OS executa no segundo nível acima do hardware.

- Exemplos: CP/CMS desenvolvido pela IBM nos 1960s, antecessor do z/VM, também da IBM; Xen VMware's ESX Server; Sun's Hypervisor

**- Type 2 hypervisor** (VMM de tipo 2) executa sobre um OS. Um "guest" OS executa no terceiro nível acima do hardware.

- Exemplos: VMware server, Microsoft Virtual Server, Oracle VirtualBox

# VMM virtualiza o CPU e a memória

VMM

•Intecepta *privileged instructions* solicitadas por um SO convidado e assegura a correção e segurança da operação/execução

•Intecepta *interrupts* e encaminha-os para o respetivo sistema operativo convidado

•Controla a gestão da memória virtual

•Mantém uma tabela *shadow page* para cada SO convidado, replicando alterações feitas por estes na sua *shadow page*. A tabela aponta para um bloco real, e é usado pelo *Memory Management Unit* (MMU) para tradução dinâmica de endereços.

•Monitoriza a carga e desempenho do sistema, tomando medidas corretivas para evitar degradação do desempenho.

# *Virtual Machines* (VMs)

VM - isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources.

Process VM - a virtual platform created for an individual process and destroyed once the process terminates.

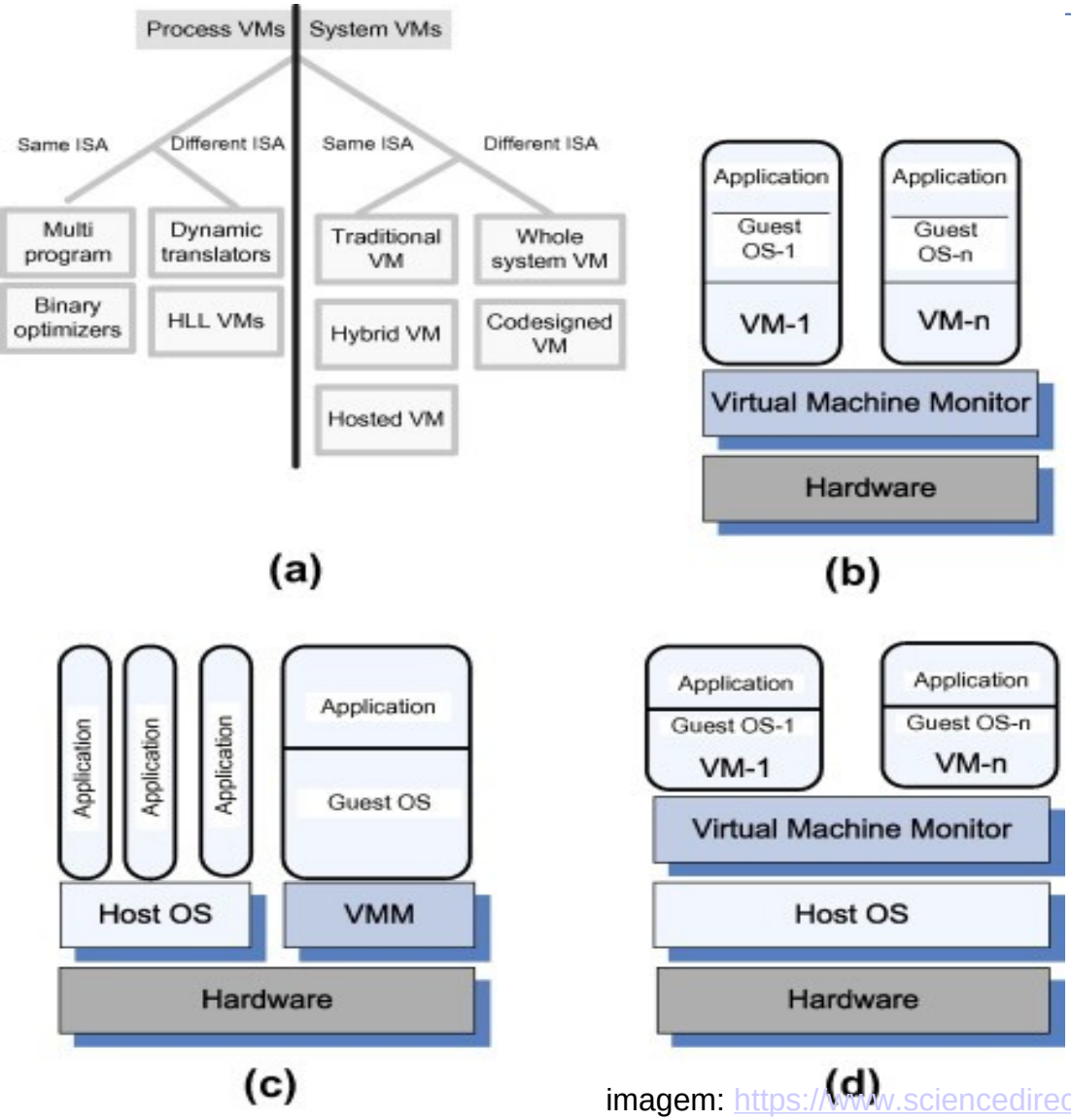
System VM - supports an operating system together with many user processes.

Traditional VM - supports multiple virtual machines and runs directly on the hardware.

Hybrid VM - shares the hardware with a host operating system and supports multiple virtual machines.

Hosted VM - runs under a host operating system.

# *Virtual Machines* (VMs)




(a) A taxonomy of process and system VMs for the same and for different ISAs. Traditional, hybrid, and hosted are three classes of VM for systems with the same ISA.

(b) Traditional VMs. The VMM supports multiple VMs and runs directly on the hardware.

(c) A hybrid VM. The VMM shares the hardware with a host operating system and supports multiple virtual machines.

(d) A hosted VM. The VMM runs under a host operating system.

imagem: https://www.sciencedirect.com/topics/computer-science/virtual-machine-monitor

# Compartimentação para desempenho e segurança

- O comportamento em tempo real de uma aplicação é afetado por outras aplicações que executam de forma concorrente na mesma plataforma, e competem pelo CPU, cache, memória, disco, e rede.
  - É difícil estimar com rigor o tempo para completar um processamento.

Isolar/compartimentar/desagregar para desempenho (*Performance isolation*)

- Uma condição crítica para garantias de QoS em ambiente de computação partilhada.

Um VMM é mais simples e mais focado que um SO tradicional. Exemplo - Xen tem aproximadamente 60,000 linhas de código; Denali tem perto de metade, 30,000.

A vulnerabilidade de VMMs é consideravelmente reduzida, pelo facto dos sistemas exporem menor número de operações privilegiadas.

UNIVERSIDADE DE ÉVORA

# Arquitetura de computadores e virtualização

Condições para virtualização eficiente

- Um programa em execução numa VM deve ter um comportamento idêntico ao demonstrado quando corre diretamente numa máquina equivalente
- VMM deve estar em pleno controlo dos recursos virtualizados
- Uma fração significativa das instruções máquina deve ser executada sem a intervenção do VMM.

Duas classes de instrução máquina:

- Sensível – requer precauções quando executada
- Inócua – não sensível

# Virtualização plena e paravirtualização

***Full virtualization*** – a guest OS can run unchanged under the VMM as if it was running directly on the hardware platform.

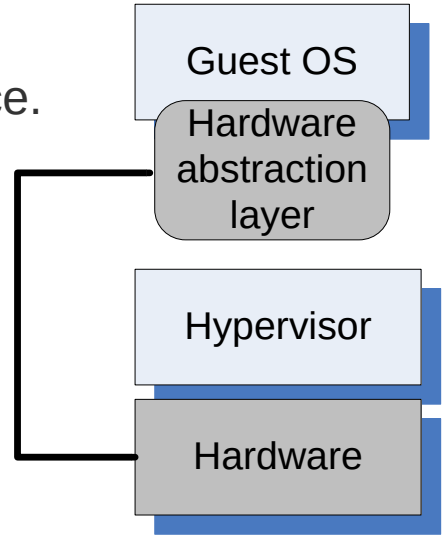
- Requires a virtualizable architecture.

Examples: Vmware.

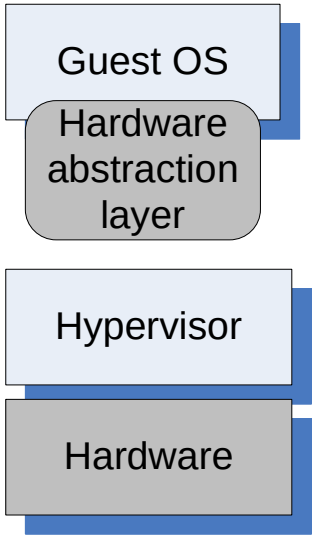
***Paravirtualization*** - a guest operating system is modified to use only instructions that can be virtualized. Reasons for paravirtualization:

- Some aspects of the hardware cannot be virtualized.
- Improved performance.
- Present a simpler interface.

Examples: Xen, Denaly




(a) Full virtualization




(b) Paravirtualization

UNIVERSIDADE DE ÉVORA

# Protection Levels

protection levels also known as rings

0 has the highest level privilege and it is in this ring that the operating system kernel normally runs.

- Code executing in ring 0 is said to be running in system space, kernel mode or supervisor mode.
- All other code such as applications running on the operating system operates in less privileged rings, typically ring 3.

UNIVERSIDADE DE ÉVORA

# Virtualization of x86 architecture

**Ring de-privileging** - a VMMs forces the operating system and the applications to run at a privilege level greater than 0.

**Ring aliasing** - a guest OS is forced to run at a privilege level other than that it was originally designed for.

**Address space compression** - a VMM uses parts of the guest address space to store several system data structures.

**Non-faulting access to privileged state** - several store instructions can only be executed at privileged level 0 because they operate on data structures that control the CPU operation. They fail silently when executed at a privilege level other than 0.

Guest system calls which cause transitions to/from privilege level 0 must be emulated by the VMM.

**Interrupt virtualization** - in response to a physical interrupt, the VMM generates a ``virtual interrupt'' and delivers it later to the target guest OS which can mask interrupts.

# Virtualization of x86 architecture (cont'd)

Access to hidden state - elements of the system state, e.g., descriptor caches for segment registers, are hidden; there is no mechanism for saving and restoring the hidden components when there is a context switch from one VM to another.

Ring compression - paging and segmentation protect VMM code from being overwritten by guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish between privilege levels 0, 1, and 2, thus the guest OS must run at privilege level 3, the so called (0/3/3) mode. Privilege levels 1 and 2 cannot be used thus, the name ring compression.

The task-priority register is frequently used by a guest OS; the VMM must protect the access to this register and trap all attempts to access it. This can cause a significant performance degradation.

UNIVERSIDADE DE ÉVORA

# Linux KVM (Kernel Virtual Machine)

Linux incorporates KVM into the Linux kernel (from v 2.6.20)

KVM is a full virtualization solution that is unique in that it turns a Linux kernel into a hypervisor using a kernel module

This module allows other guest operating systems to then run in user-space of the host Linux kernel

The KVM module in the kernel exposes the virtualized hardware through the /dev/kvm character device

The guest operating system interfaces to the KVM module using a modified QEMU process for PC hardware emulation

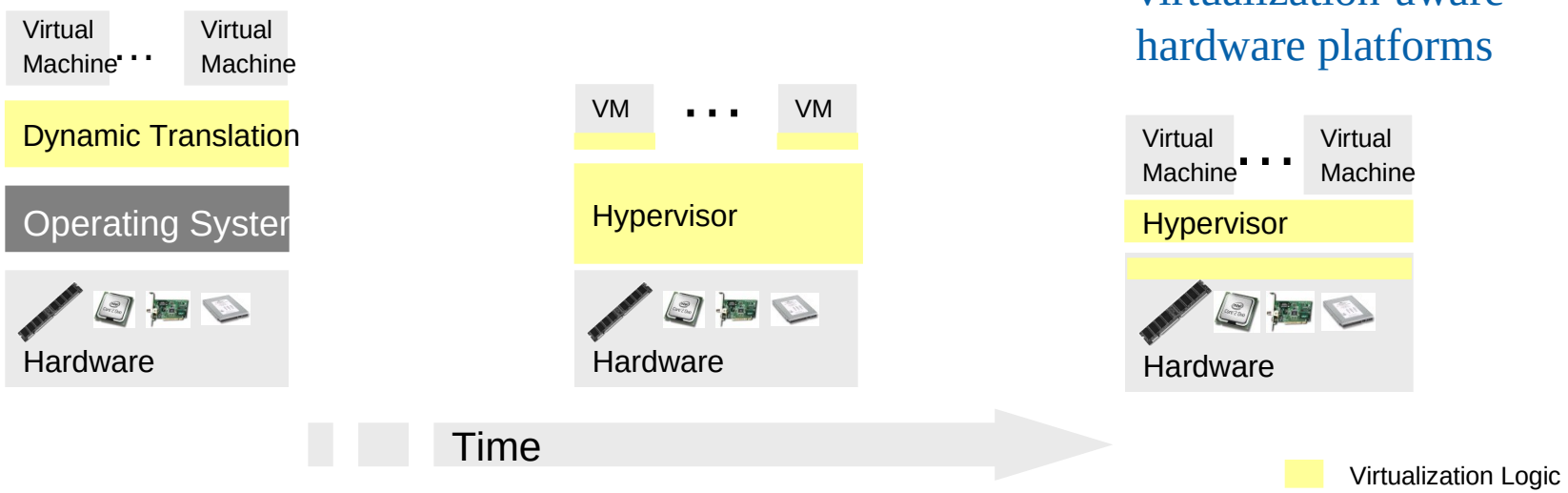
UNIVERSIDADE DE ÉVORA

# Virtualization Examples

- Disaster recovery
  - Virtual machines can be used as "hot standby" environments for physical production servers. This changes the classical "backup-and-restore" philosophy, by providing backup images that can "boot" into live virtual machines, capable of taking over workload for a production server experiencing an outage.
- Testing and training
  - Hardware virtualization can give root access to a virtual machine. This can be very useful such as in kernel development and operating system courses.
- Portable workspaces
  - Recent technologies have used virtualization to create portable workspaces on devices like USB memory sticks. Examples:
    - Thinstall
    - MojoPac, Ceedo
    - moka5 and LivePC

# Evolution of Software solutions*

- 1st Generation: Full virtualization (Binary rewriting)
  - Software Based
  - VMware and Microsoft
- 2nd Generation: Paravirtualization
  - Cooperative virtualization
  - Modified guest
  - VMware, Xen
- 3rd Generation: Silicon-based (Hardware-assisted) virtualization
  - Unmodified guest
  - VMware and Xen on virtualization-aware hardware platforms




*This slide is from Intel® Corporation

# Desvantagens/riscos por vezes associados a virtualização

Numa organização em camadas, as proteções num determinado nível/camada podem ser ultrapassadas por malware em camada inferior.

Pode existir um VMM Rootkit entre o hardware e o sistema operativo

- Rootkit - malware com acesso privilegiado ao sistema
- O Rootkit pode permitir que um SO malicioso separado seja executado clandestinamente, e de forma invisível/indetetável para o SO convidado e aplicativos sobre ele executados.
  - Este SO malicioso poderia:
    - observar dados, eventos ou o estado do SO alvo
    - executar serviços, retransmissões de spam ou ataques distribuídos DDoS
    - interferir com o funcionamento de aplicações

# Contentores (*Containers*)

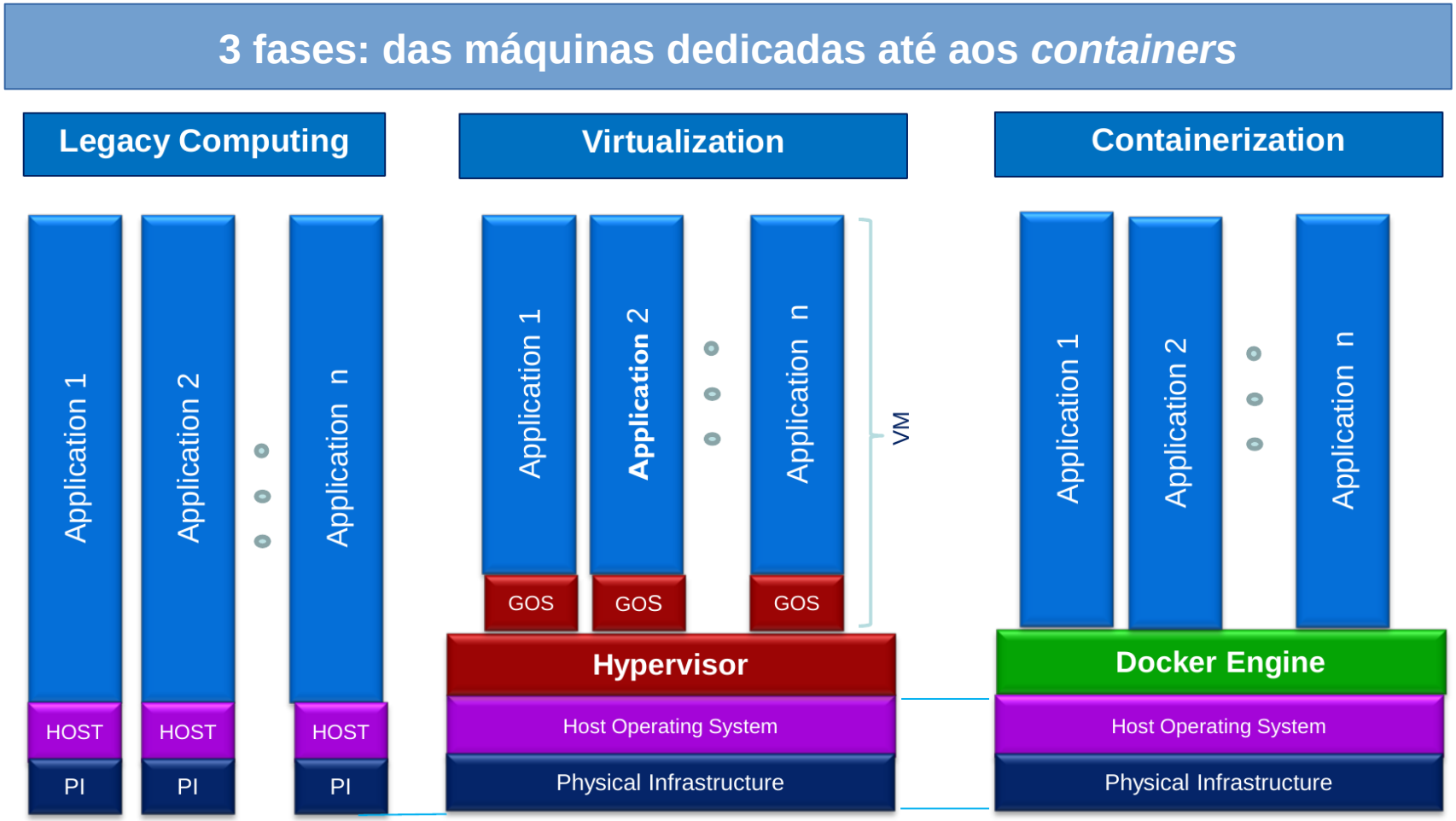
***Containers***: recursos para alojar aplicações

- forma mais leve de virtualização orientada para aplicações
- isolamento: entre aplicações, e entre aplicações e SO
- portabilidade: software, dados, configurações e logs/registos de atividade

***Container orchestration*** (orquestração)

- agilização da implantação e gestão de serviços baseados em *containers*
  - (kubernetes...)

# *Containers*




*Legend: DE=Docker Engine; G-OS=Guest Operating System; HOST=Host Operating System; PI=Physical Infrastructure*

imagem: Anil Vasudeva e SNIA Tutorial © 2015 Storage Networking Industry Association

UNIVERSIDADE DE ÉVORA

# VMs vs Containers

- Pontos favoráveis aos *Containers*
  - permitem concentrar mais carga computacional na mesma plataforma
  - requerem menos tempo para arranque de uma solução
  - Flexibilidade
  - Maior facilidade para especificar um ambiente de execução
- Desafios
  - A partilha da infraestrutura subjacente pode trazer desafios
    - imprevisibilidade do desempenho
    - esgotar recursos partilhados (rede? processador?)
- Mas... não têm de ser alternativa. VMs e *Containers* podem usar-se de modo combinado, ou ser escolhidos em função de cada caso.

UNIVERSIDADE DE ÉVORA

# Soluções/opções para *containers*

Alguns tipos de *container*

- LXC - API e ferramentas para containers /Linux kernel
  - https://linuxcontainers.org/lxc/
  - https://linuxcontainers.org/lxd/introduction/
- Windows Server Containers
  - https://docs.microsoft.com/en-us/virtualization/windowscontainers/
- Docker
  - https://www.docker.com/why-docker
  - https://www.docker.com/101-tutorial

UNIVERSIDADE DE ÉVORA

# Créditos, referências e leituras

- *Cloud Computing: Theory and Practice*

  Dan C. Marinescu

  Chapter 5

- *Hypervisor, Virtualization Stack, And Device Virtualization Architectures*

  Mike Neil, Microsoft Corporation

- *Distributed and Cloud Computing*

  K. Hwang, G. Fox and J. Dongarra