Virtualization

G54CCS – Lecture 4

Richard Mortier

http://www.cs.nott.ac.uk/~rmm/teaching/2011-g54ccs/

Don’t forget!

- Extra lab session **today** 4pm, C11
- Appears that B52 also free at that time
- Please split yourselves between the two

Overview

- Why Virtualization?
- Abstraction & Layering
- The Operating System
- Virtualization
- Benefits

Overview

- Why Virtualization?
- Abstraction & Layering
- The Operating System
- Virtualization
- Benefits

Why Virtualization?

- Perhaps the key enabling cloud technology
  - How?
  - Why?
- To answer, we must dive into the technology
  - (a little)
- Three important concepts:
  - Abstraction/Layering
  - Isolation
  - Multiplexing

Overview

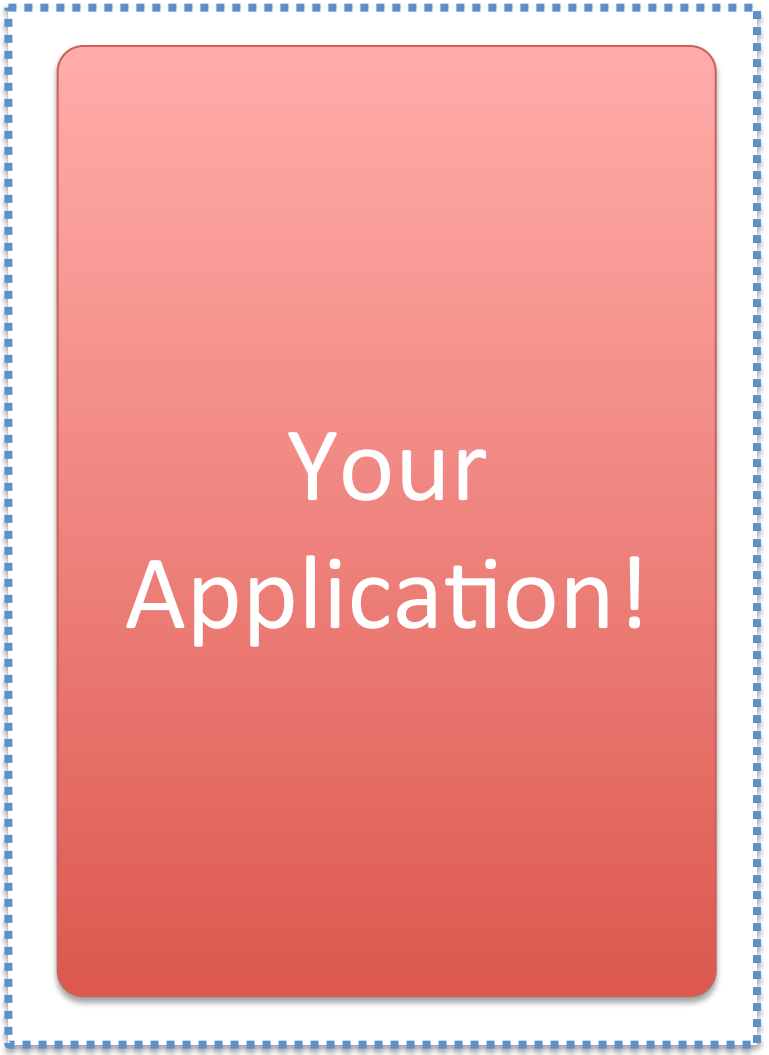
- Why Virtualization?
- Abstraction & Layering
  - What's Under Your Application?
  - Hardware
- The Operating System
- Virtualization
- Benefits

Abstraction and Layering

- Computers are complex – need to control this
- *Abstraction*
  - Deliberately hiding details
  - Presenting a simplified *interface*
- *Layering*
  - Control interactions
  - Each layer depends only on interface below
  - …and services layer above via own interface

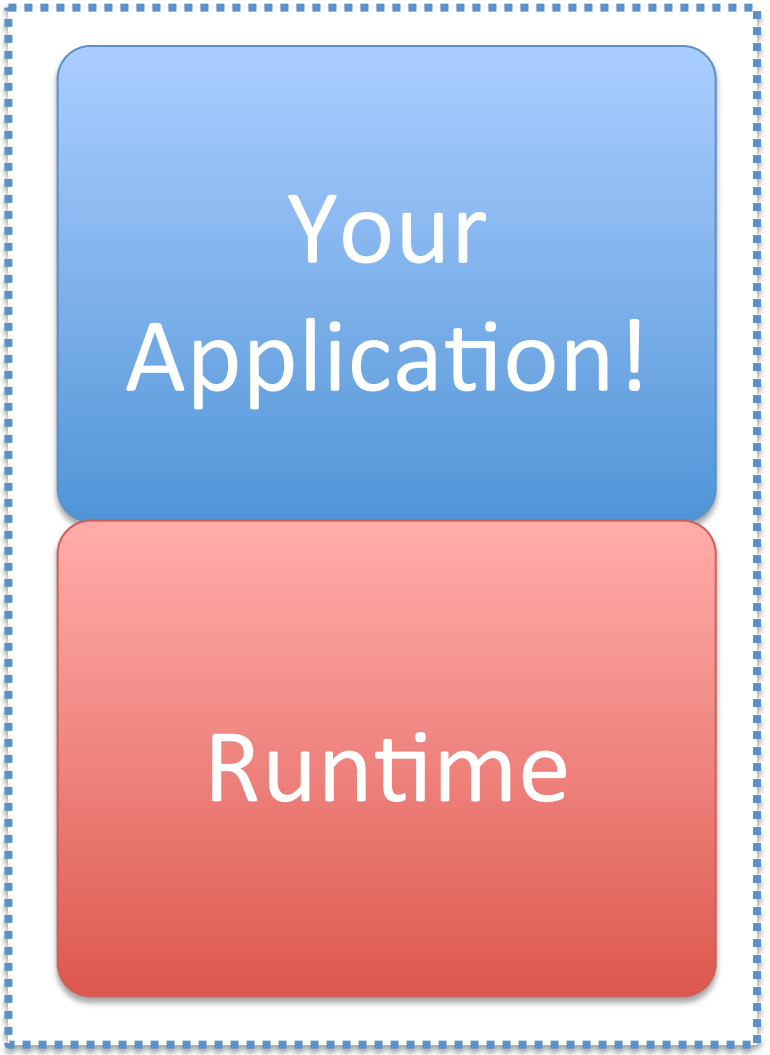
What's Under Your App?

- Application code
  - E.g., in your labs
  - What you want to run
  - What handles your users' requests
  - What makes you money



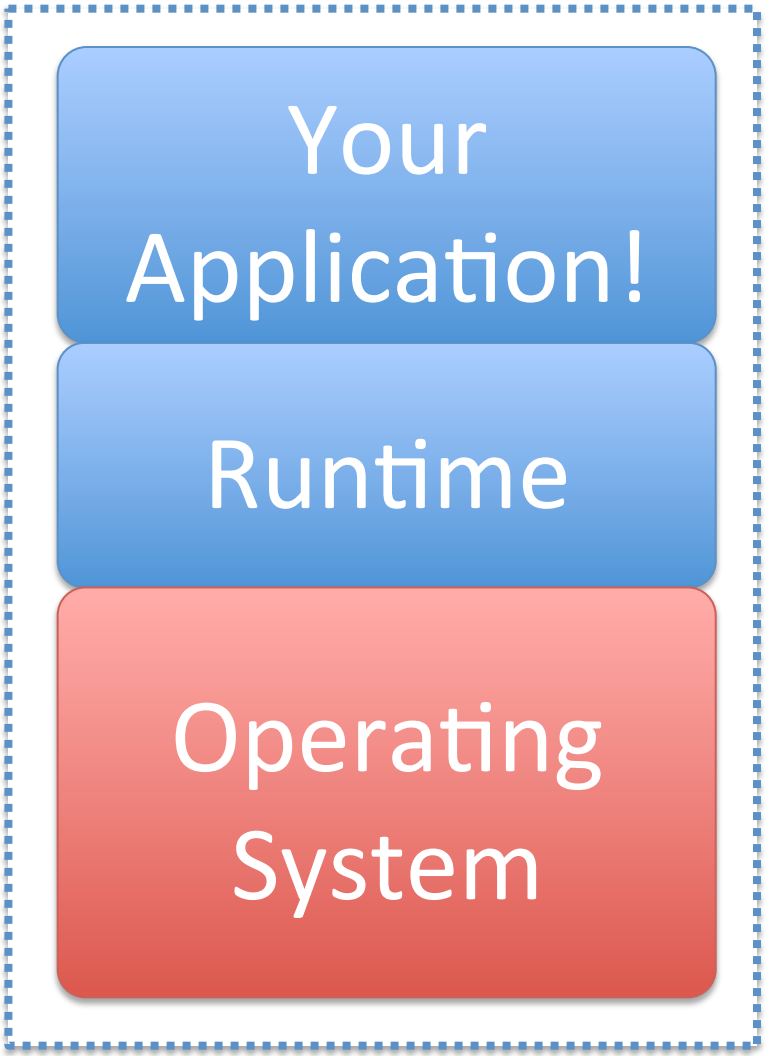
What's Under Your App?

- Application code
  - E.g., in your labs
  - What you want to run
  - What handles your users' requests
  - What makes you money
- Sits on top of a *runtime*
  - python.exe; JVM; libc
  - Provides environment and support functions



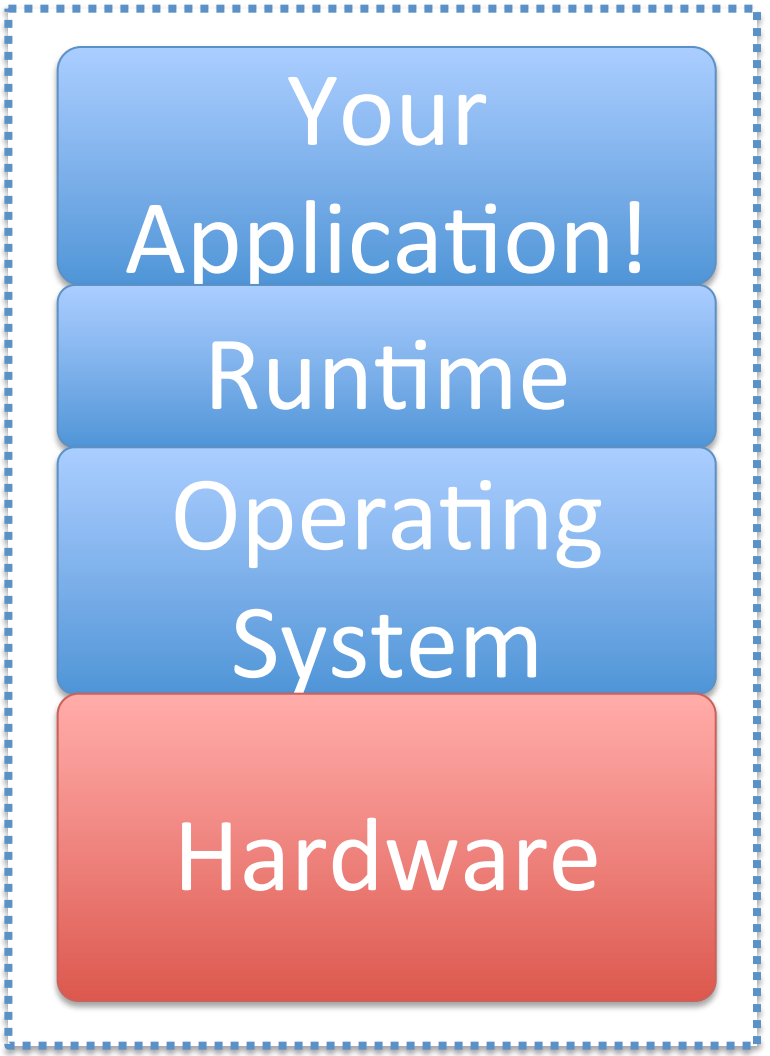
What's Under Your App?

- Operating System
  - Abstracts hardware
  - Isolates code
  - Manages resources



What's Under Your App?

- Operating System
  - Abstracts hardware
  - Isolates code
  - Manages resources
- Hardware
  - CPU with *N* cores
  - Memory, storage
  - Communications



Hardware

- The physical components of the computer
- Processing
  - CPU, motherboard, GPU
- Memory
  - Registers, cache, main memory
- Peripherals
  - Communications – network cards
  - Storage – Hard disks, SSD
  - Screens, cameras, printers, …

Overview

- Why Virtualization?
- Abstraction & Layering
- The Operating System
  - The Boot Process
  - Resource Management
  - Isolation, Buffering, Multiplexing
- Virtualization
- Benefits

The Boot Process (simplified!)

- Power on and stable
- CPU starts executing code
- Goes first to the BIOS
  - Initiates POST
  - Finds boot device, loads MBR code, jumps
- MBR code knows how to read disk
  - Find, load, execute operating system
- Can now run application code
  - E.g., a shell, a window manager

The Operating System (simplified!)

- Minimally,
  - Manages resources
  - Simplifies hardware
  - Provides application programming interface (API)
- As part of *resource management*,
  - Isolation
  - Buffering
  - Multiplexing

Resources

- CPU
  - What should run now?
- Memory
  - Hide memory hierarchy from programmer
  - Virtual address space
- Storage
  - OS provides high level interface to block storage
  - E.g., file system, key-value store, object serialization
- Network
  - Range of communications protocols and primitives
  - Access to hardware to send and receive traffic

Isolation

- Hardware isolation – one machine, one app
  - But typically even then, many pieces of software
  - E.g., OS, standard services, application
- Software isolation
  - Use *processes* to protect memory between apps
    - Other concurrency models are available
  - Limits scope for interference and corruption
  - Application may have many processes

Interacting with Hardware

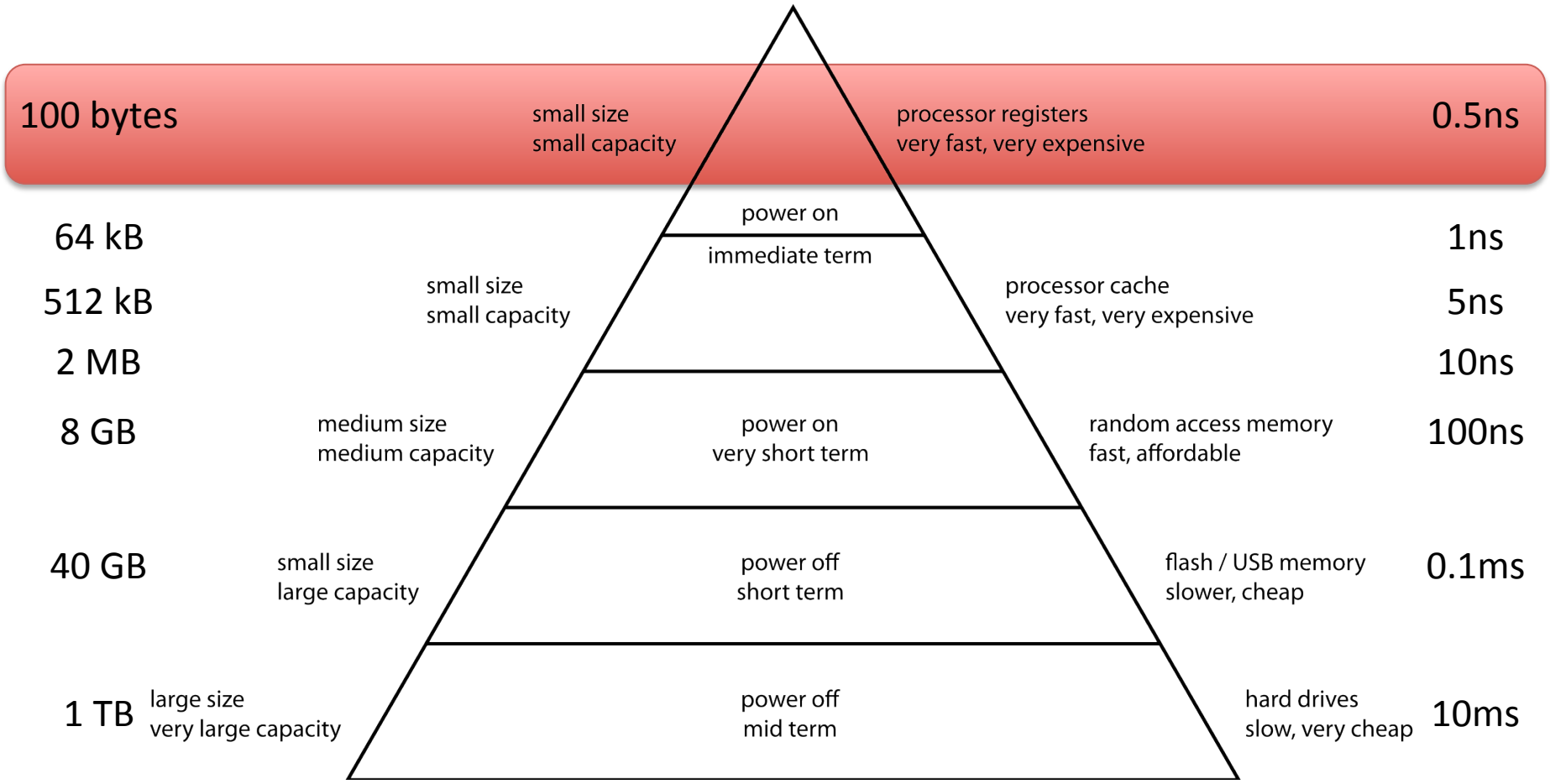
- How to protect interactions with hardware?
  - Prevent untrusted user code having direct access
- Kernel *vs*. User mode
  - Use hardware support – x86 *rings*
  - *syscalls* provide interface to OS
  - E.g., read(), write(), recv(), send(), … [POSIX]
- Manage passing of buffers across boundary
  - Must ensure that app can't corrupt memory

Multiplexing

- Multiple applications running simultaneously
  - …or the illusion of that if only 1 CPU
- Makes more efficient use of the hardware
  - While one app is waiting, another is getting on
  - If none are waiting, then switch between apps
  - Wide range of policies for who goes next
- Waiting is quite common due to latency mismatch
- Example – Web server waiting on database
  - Request comes in, gets processed
  - Response needs several DB records plus some files

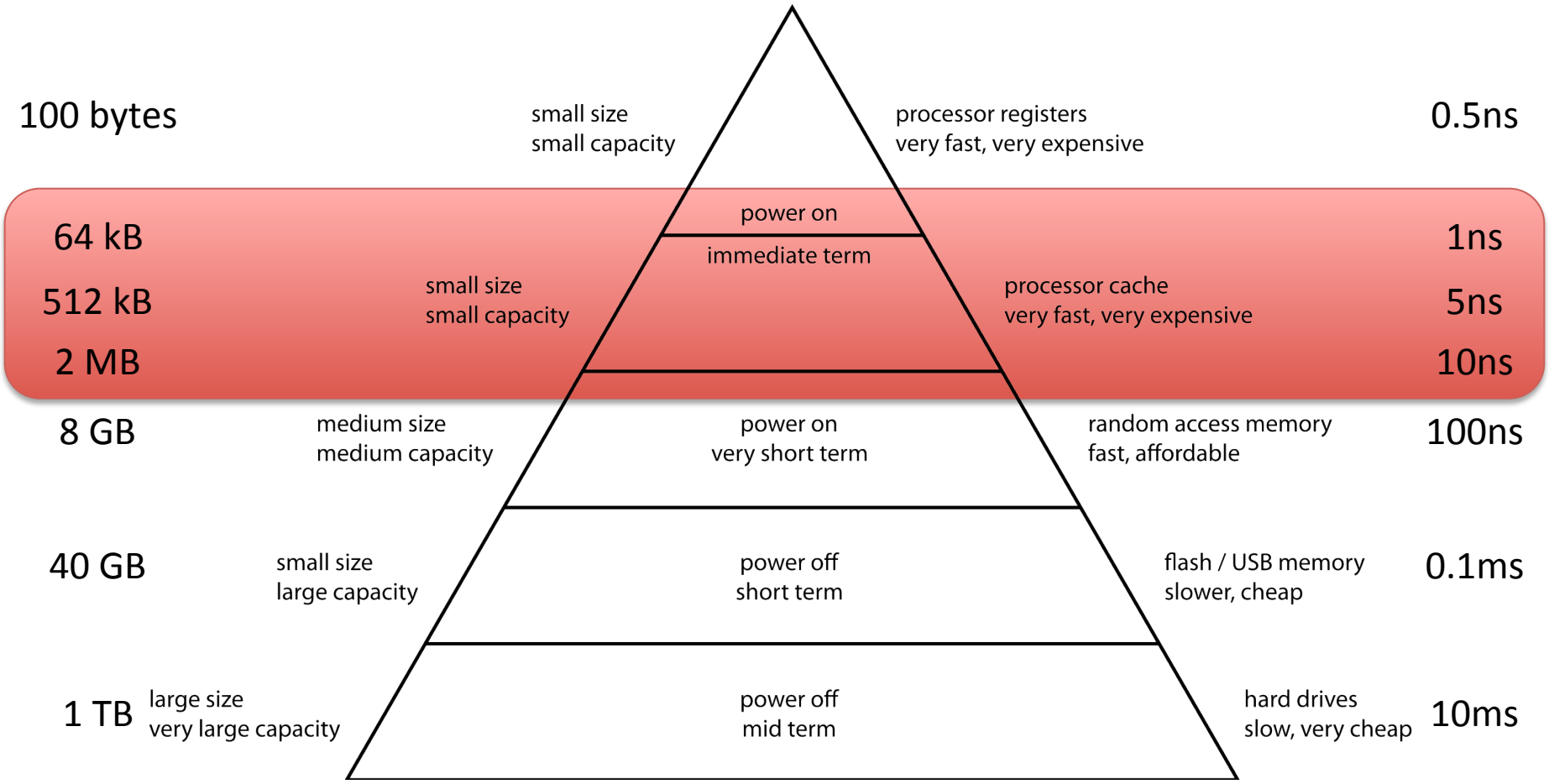
Capacity *vs*. Latency

approximate!



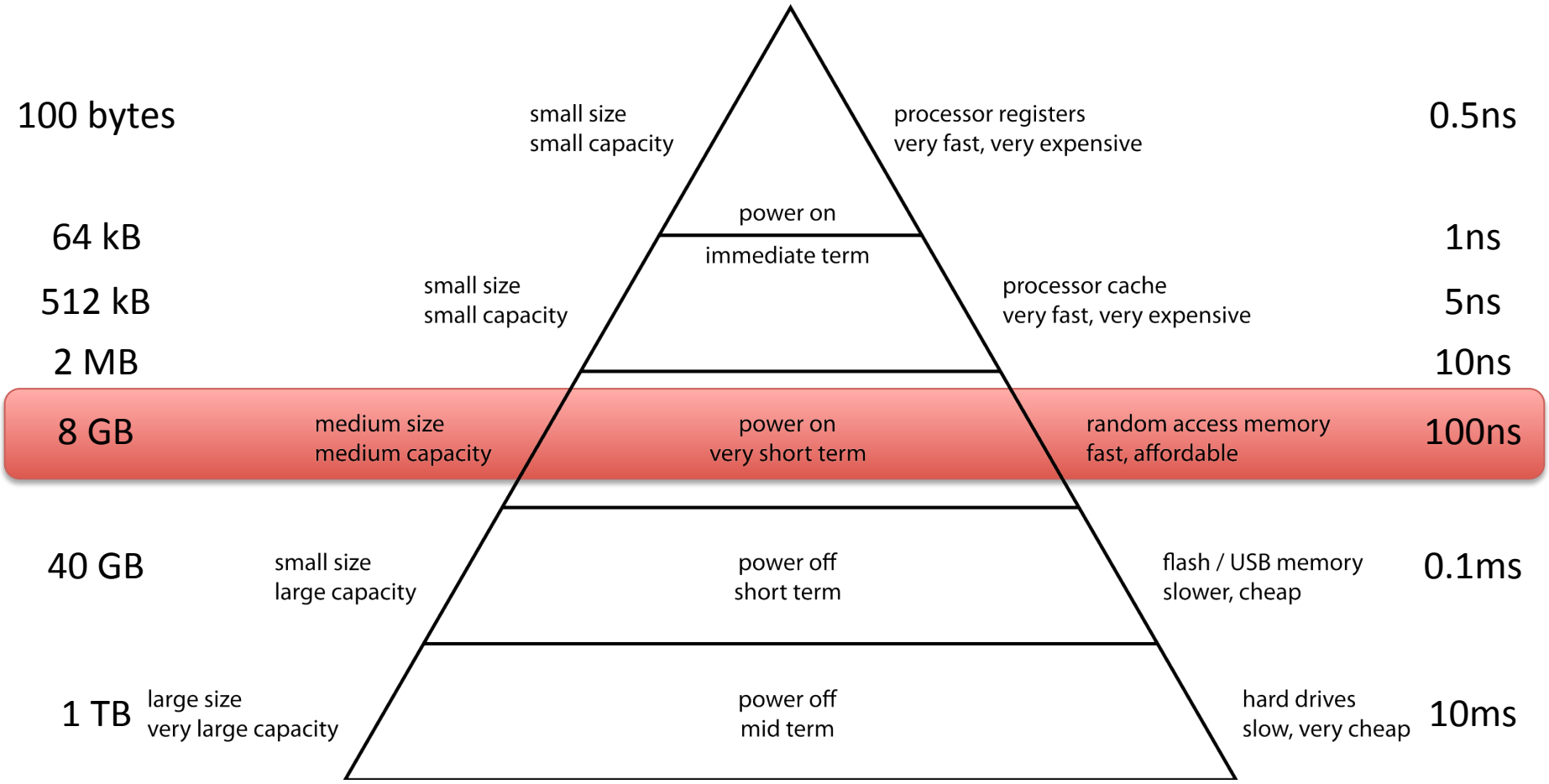
Capacity *vs*. Latency

approximate!



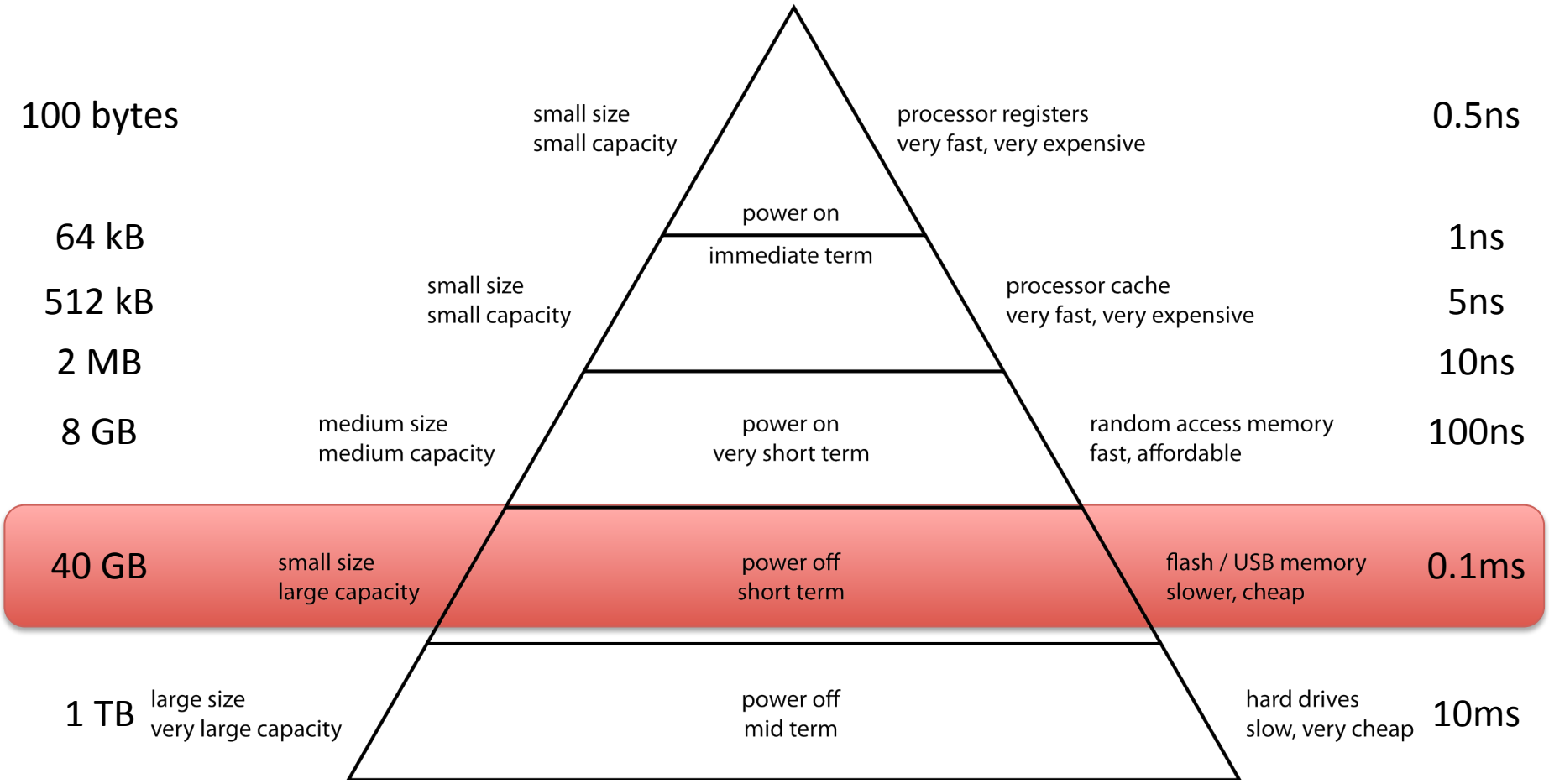
Capacity *vs*. Latency

approximate!



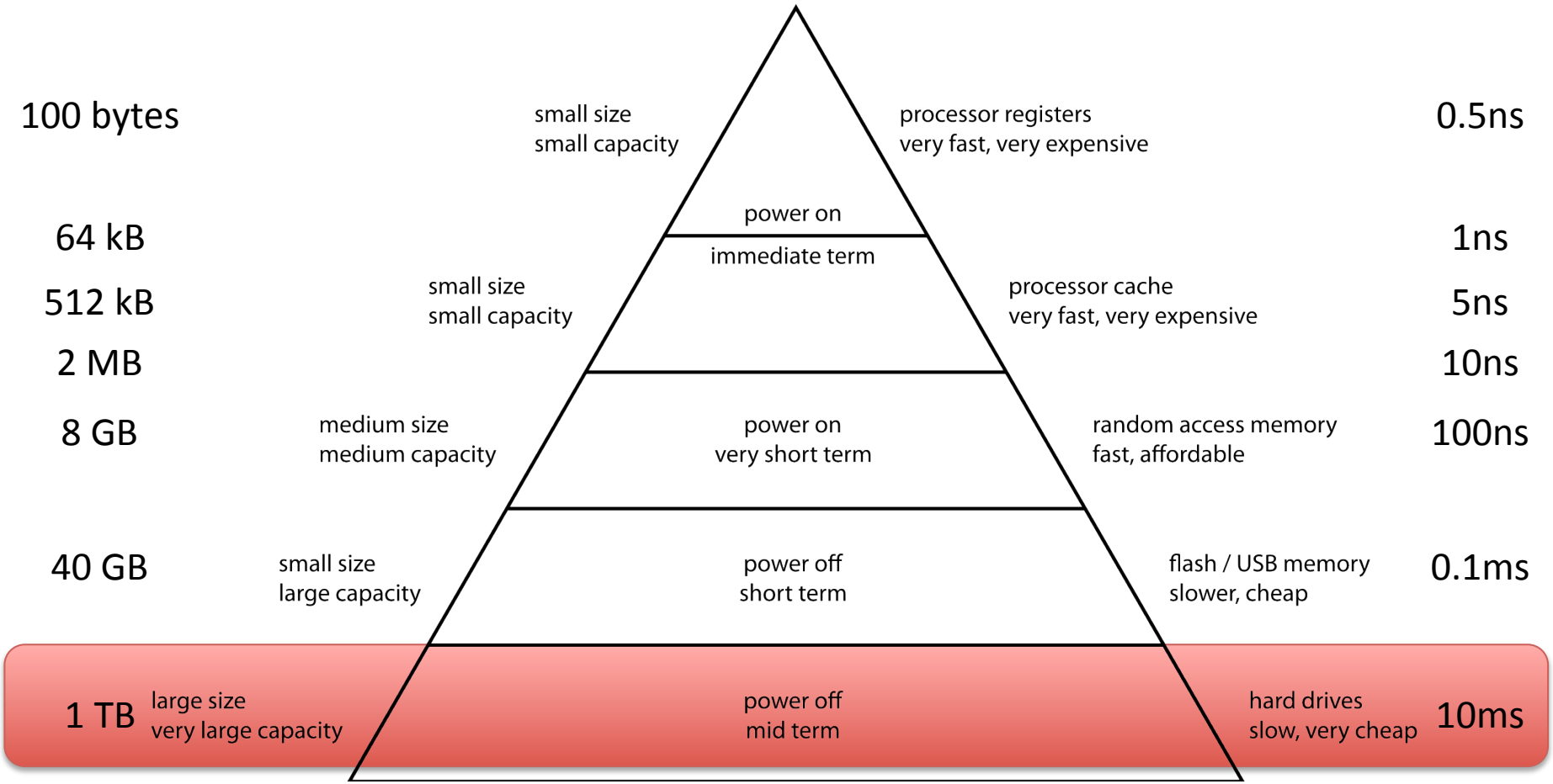
Capacity *vs*. Latency

approximate!



Capacity *vs*. Latency

approximate!



Overview

- Why Virtualization?
- Abstraction & Layering
- The Operating System
- Virtualization
  - Virtualization Types
  - Compatibility Options
  - In the Wild
- Benefits

Virtualization

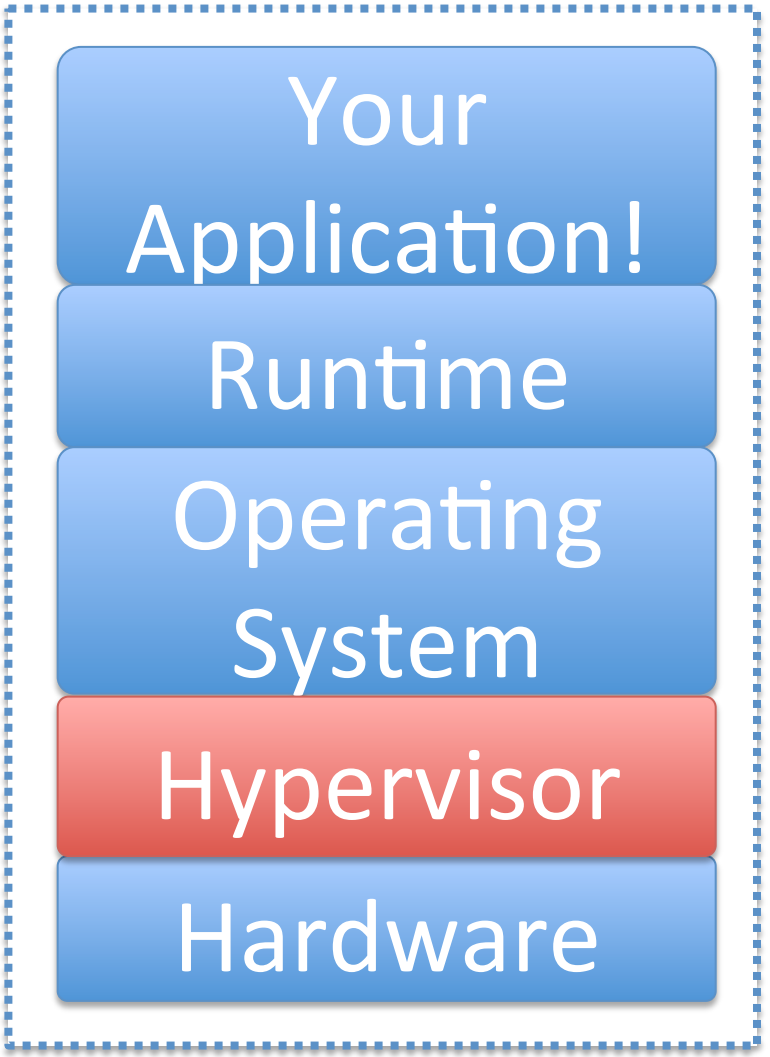
- Software deployment
  - More than just the application
  - Applications rely on many services
  - E.g., DNS, firewall, filtering, backup, storage
- Can we do better?

Virtualization

- Software deployment
  - More than just the application
  - Applications rely on many services
  - E.g., DNS, firewall, filtering, backup, storage
- Can we do better?
  - Encapsulate OS and associated state
  - Configuration, data, filesystem, code
- But how?

Virtualization

- Apply *indirection*
  - A traditional computer science solution
  - Insert extra layer of code to support the OS
  - OS itself still supports application processes
  - *Virtual Machine* (*VM*)



Virtualization Types

- Type I – *Hypervisor Architecture*
  - So-called "bare metal"
  - A *hypervisor* or *VMM* sits beneath the OS
  - On x86, OS must be modified to target this
  - E.g., Xen, VMWare ESX Server
- Type II – *Hosted Architecture*
  - Sits above a *host OS*
  - E.g., VirtualBox, VMWare Workstation

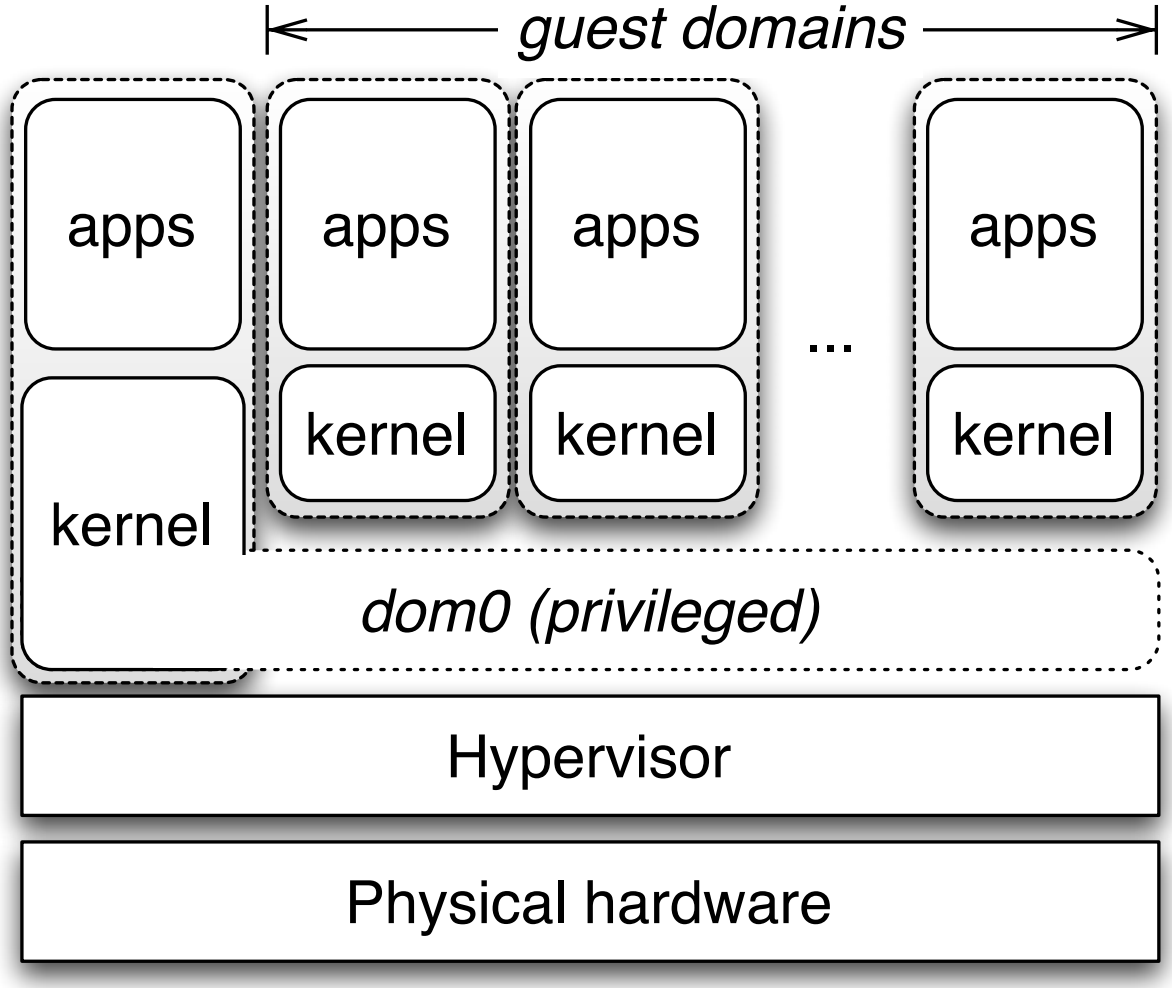
Compatibility Options

- Full virtualization
  - Completely transparent to guest OS
  - Binary level compatibility, no changes required
  - Potential performance issues
- Paravirtualization
  - Transparent to applications
  - Requires modification of guest OS
    - Xen required ~3k lines of Linux
  - Improves performance

Aside – *x86* Virtualization

- Instruction set difficult to virtualize
  - Non-privileged instructions expose VMM
  - Three approaches
- Emulation (VirtualPC)
  - Emulate all instructions in kernel mode
- Binary translation (VMWare)
  - Convert kernel code to call into VMM
- Paravirtualization (Xen)
  - Remove privileged instructions from Guest OS

Example – Xen


guest domains
apps
apps
apps
apps
...
kernel
kernel
kernel
kernel
dom0 (privileged)
Hypervisor
Physical hardware

In the Wild

- Amazon's EC2 infrastructure
  - Uses a (patched) Xen hypervisor
  - Supports Linux, Windows, other VM images
  - Amazon Machine Image (AMI) as self-contained VM
  - Can provide own kernels too nowadays
- Microsoft's Azure platform
  - Customized Hyper-V
  - Provides wide range of Windows VM images, plus selected Linux distributions
  - Had to submit code for drivers for Linux

Overview

- Why Virtualization?
- Abstraction & Layering
- The Operating System
- Virtualization
- Benefits

Benefits

- Natural provisioning primitive
  - Easier to deploy and manage
  - Take advantage of features like live migration
- Datacenter scaledown
  - Coalesce machine roles onto less hardware
  - Reduces costs in all directions
- Security
  - Better isolation can improve security, resilience
- Charging
  - Separate resource consumption, separate billing

Summary

- Why Virtualization?
- Abstraction & Layering
  - What's Under Your Application?
  - Hardware
- The Operating System
  - The Boot Process
  - Resource Management
  - Isolation, Buffering, Multiplexing
- Virtualization
  - Virtualization Types
  - Compatibility Options
  - In the Wild
- Benefits