OS-level Virtualization, Memory & Data Centers

S. M. Ahsan Kazmi

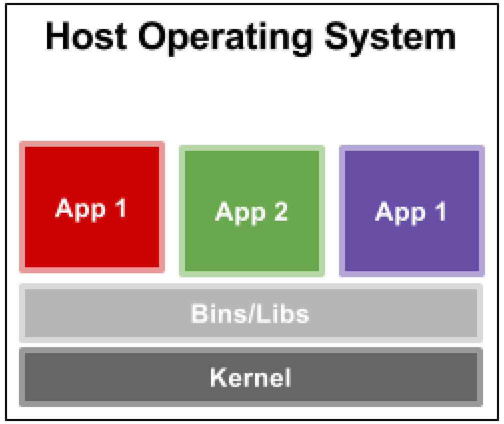
Outline

- OS-Level Virtualization
  - Comparison with different types of virtualization
  - Containers Building Blocks
  - Containers Orchestration
- Memory Technologies and Hierarchies
  - Virtualization- Memory Basics and Challenges
  - Memory reclamation approaches
- Data Center and Cloud

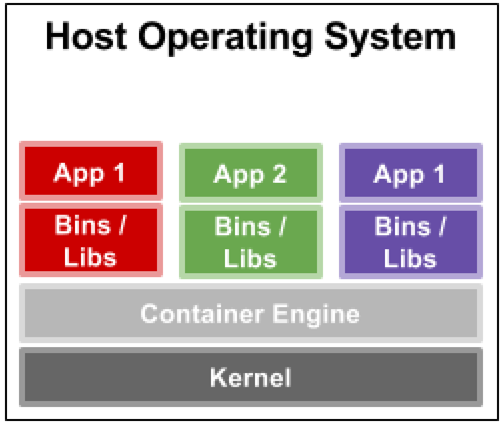
Introduction-OS Virtualization

- Containers, otherwise known as **operating-system-level virtualization**, are a lightweight approach to virtualization that only provides the bare minimum that an application requires to run

- Typically, a container includes
  - Application
  - Dependencies
  - Libraries
  - Binaries
  - Configuration files



Native Running Applications



Containerized Applications

Why do we need container?

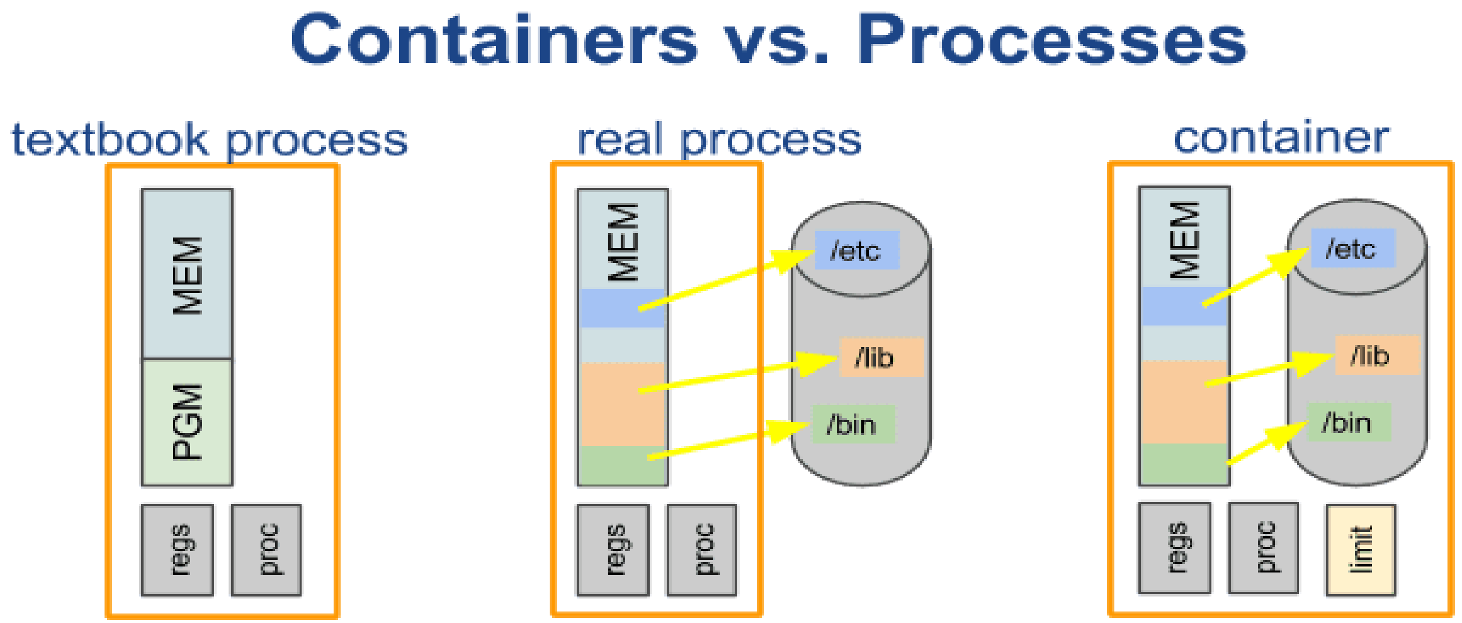
- Isolation:
  - Containers allow the deployment of one or more applications on the same physical machine, each requiring exclusive access to its respective resources.
- Security:
  - Network services can be run in a container, which limits the damage caused by a security breach or violation.
    - For instance, an intruder who successfully exploits a security hole in one of the applications running in that container is restricted to the set of actions possible within that container.

Why do we need container?

- Virtualization and transparency:
  - Containers provide the system with a virtualized environment that can hide or limit the visibility of the physical devices or system's configuration underneath it.
  - The general principle behind a container is to avoid changing the environment in which applications are running except for addressing security or isolation issues.

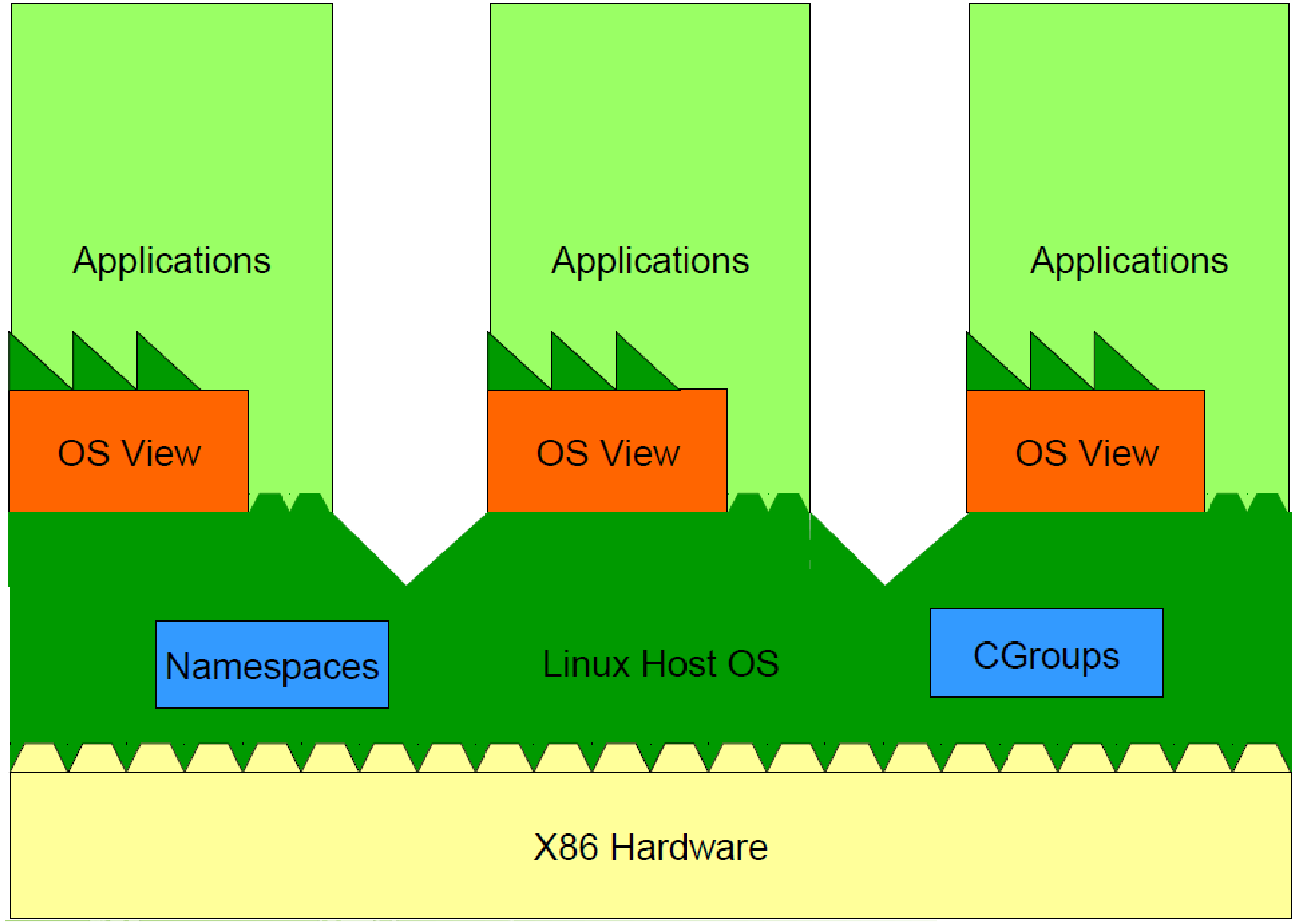
Containers vs. Processes

- Containers really are processes with their full environment.
- Textbook states that a process has its own address space, program, CPU state, and process table entry.
  - Actually, the modern real process has memory mapped from the files system into the process address space and often consists of dozens of shared libraries



Lightweight process virtualization

- A process that gives the user the illusion that he runs a Linux operating system.
- You can run many such processes on a machine, and all such processes in fact share a single Linux kernel that runs on the machine.



Lightweight process virtualization

- Opposed to hypervisor solutions (Xen) where you run another instance of the kernel

- The idea is not really a new paradigm - we have Solaris Zones and BSD jails already several years ago

- Advantages of Hypervisor-based VMs:
  - You can create VMs of other operating systems (Windows, BSDs).
  - Security – Though there were cases of security vulnerabilities that were found and required installing patches to handle them (like VENOM)

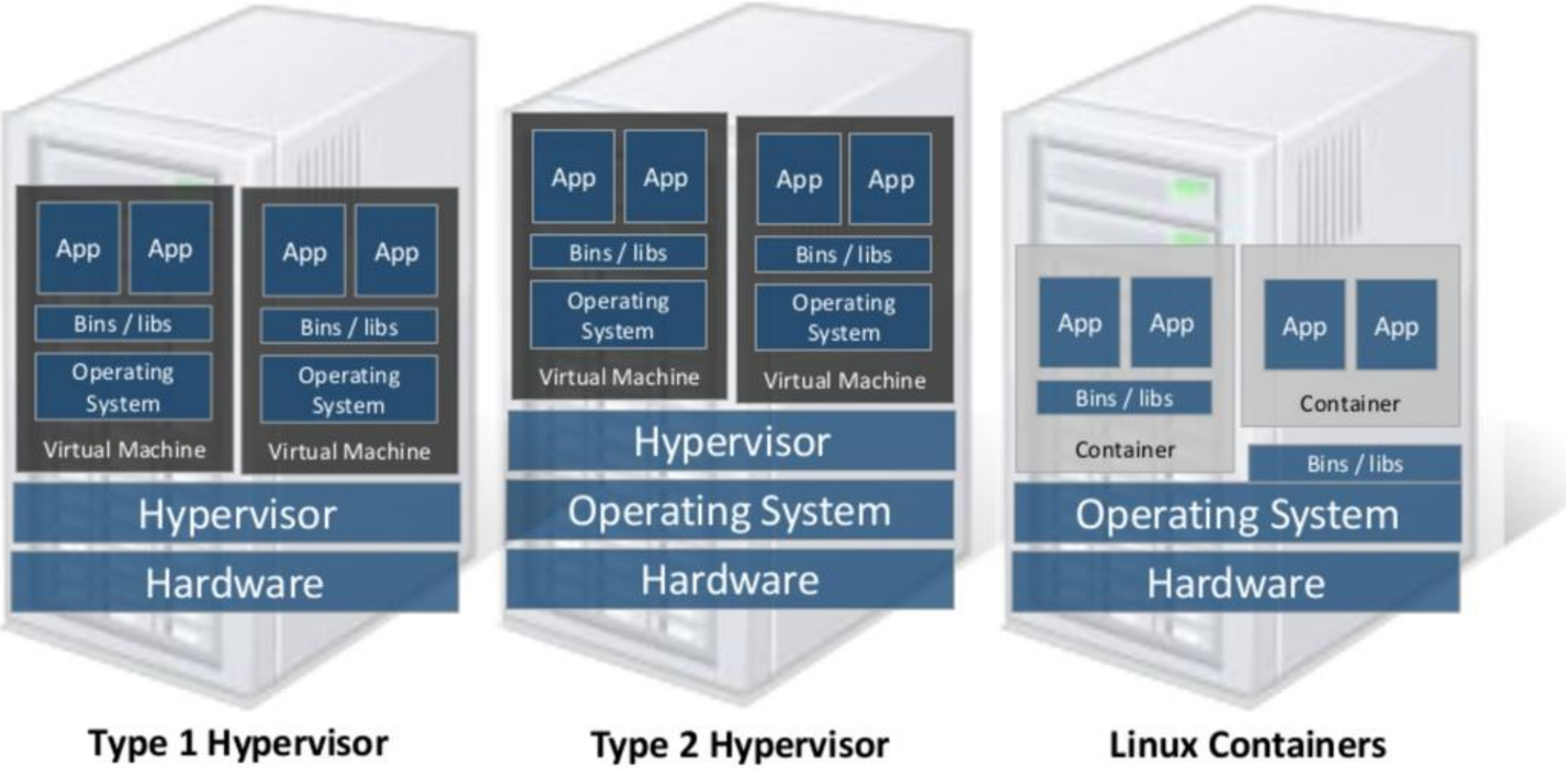
Containers vs. Hypervisor-based VMs

Containers – Advantages:

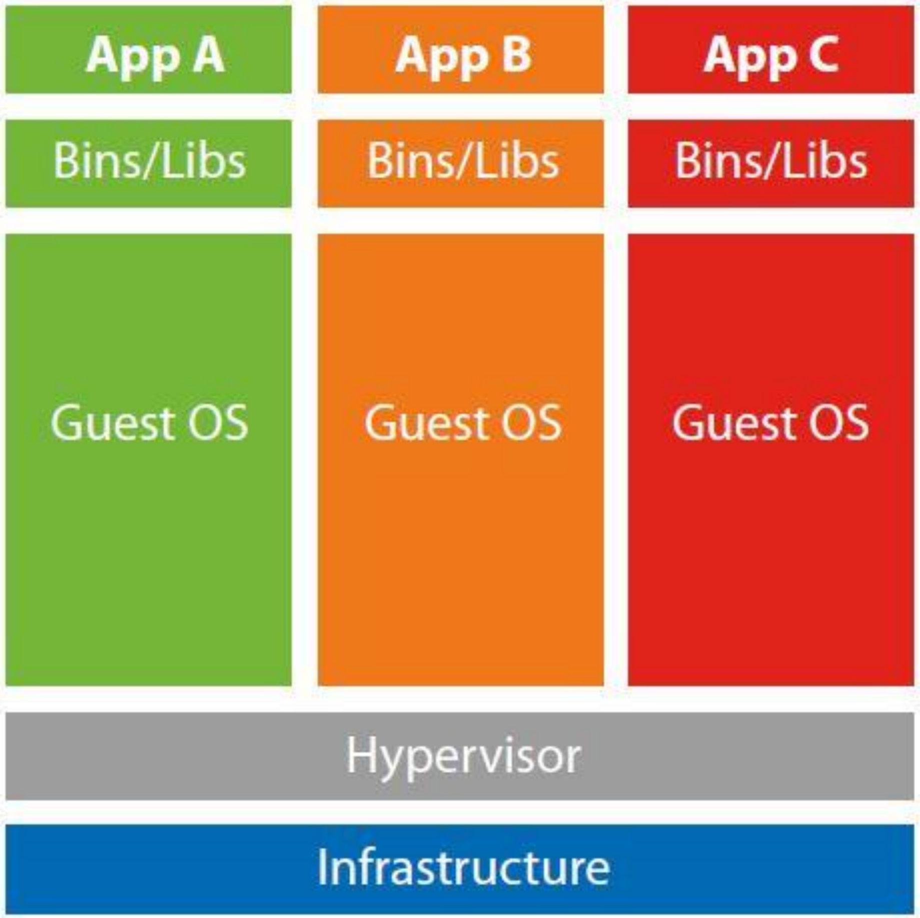
- **Lightweight**: occupies fewer resources (like memory) significantly than a hypervisor
- **Density**: you can install many more containers on a given host than VMs.
- **Elasticity**: startup time and shutdown time are much lesser, almost instantaneous. Creation of a container has the overhead of creating a Linux process, which can be of the order of milliseconds while creating a VM based on XEN/KVM can take seconds.

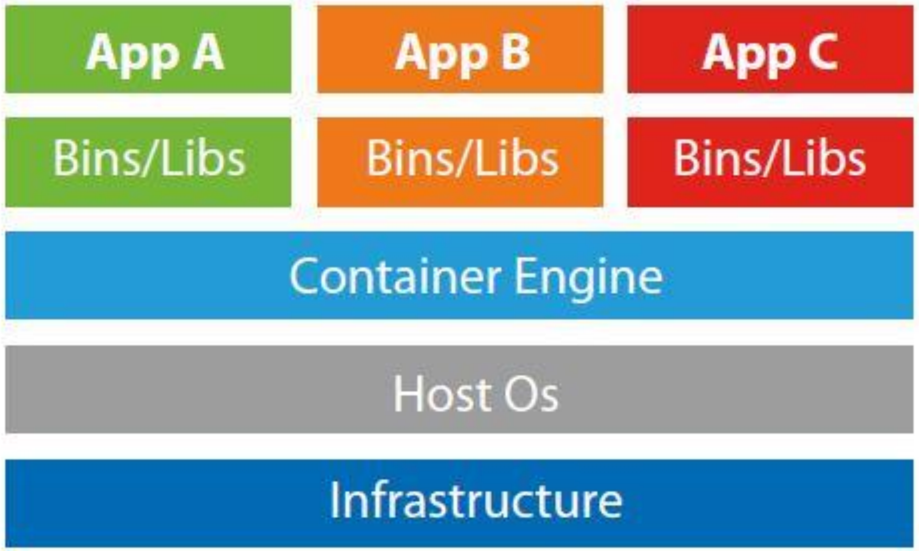
The lightness of the containers in fact provides both their density and their elasticity.

VMs vs. Containers



VMs vs. Containers





Operating-System Level Virtualization

- In between system/hypervisor-based VM and Process

- Not System VM:
  - Cannot choose OS

- Not Process VM:
  - Multiple processes, not isolated

- As if multiple instances of the same OS are running on the same machine
  - Example: Docker, LXC, rkt, runC, systemd-nspawn OpenVZ, Jails, Zones

Low level approach: enhanced chroot

- It's not quite like a VM:
  - uses the host kernel
  - can't boot a different OS

- It's just normal processes on the host machine
  - contrast with VMs which are opaque

Building Blocks of Containers

- The namespace subsystem and the cgroup subsystem are the basis of lightweight process virtualization.

- Can be used also for setting a testing environment or as a resource management/resource isolation setup and for accounting.

The cgroup subsystem: background

The cgroup (control groups) subsystem is a Resource Management and Resource Accounting/Tracking solution, providing a generic process-grouping framework.

- It handles resources such as ***processor time, number of processes per group, etc.*** for a process or set of processes.
- CPU, memory, and IO are the most important resources that cgroup deals with

Control groups

- Resource metering and limiting: – what and how much can a container use?
  - memory
  - CPU
  - block I/O
  - network

- Examples
  - CPU: weighted proportional share of CPU for a group
  - cpuset: cores that a group can access
  - block IO : weighted proportional block IO access
  - memory: max memory limit for a group

How to use cgroups?

- Control groups can be used in multiple ways:
  - Directly by accessing the cgroup virtual file system manually.
  - Indirectly through other software that uses cgroups, such as Docker, Linux Containers (LXC) virtualization, libvirt, etc.

Namespaces

- Provide processes with their own system view
- **Cgroups** = limits how much you can use;
- **namespaces** = limits what you can see
  - you can’t use/affect what you can’t see

Linux Containers: Namespaces

- A namespace wraps a **global system resource** in an **abstraction** that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource.

- Changes to the global resource are visible to other processes that are members of the namespace but are invisible to other processes.

- Currently, Linux provides 6 types of namespaces

Namespaces – cont.

- mnt (mount points, filesystems) (what files, dir are visible)
- pid (processes)
- net (network stack)
- ipc (System V IPC)
- uts (hostname)
- user (UIDs)
- (plans to add more)

Which one is the best container?

- They all use the same kernel features
- Performance will be almost the same
- Look at:
  - Features
  - Design
  - Ecosystem

Container Orchestration

- Orchestration is the automated configuration, coordination, and management of computer systems

- Container orchestration platforms empower users to easily deploy, manage, and scale multi-container-based applications in large clusters without having to worry about which server will host a particular container.

Container Orchestration

- Why?
  - Cannot pile up containers on one machine (capacity problem).
  - Require multiple machines (also called nodes) to be running in a cluster (i.e., networked and talking to each other)
  - Where do I deploy containers?

- Examples: Kubernetes, Docker Swarm, etc.

Memory Technologies and Hierarchies

Memory Technologies

	Capacity	Latency	Cost/GB
Register	100s of bits	20 ps	$$$$
SRAM	~10 KB-10 MB	1-10 ns	~$1000
DRAM	~10 GB	80 ns	~$10
Flash	~100 GB	100 us	~$1
Hard disk	~1 TB	10 ms	~$0.1

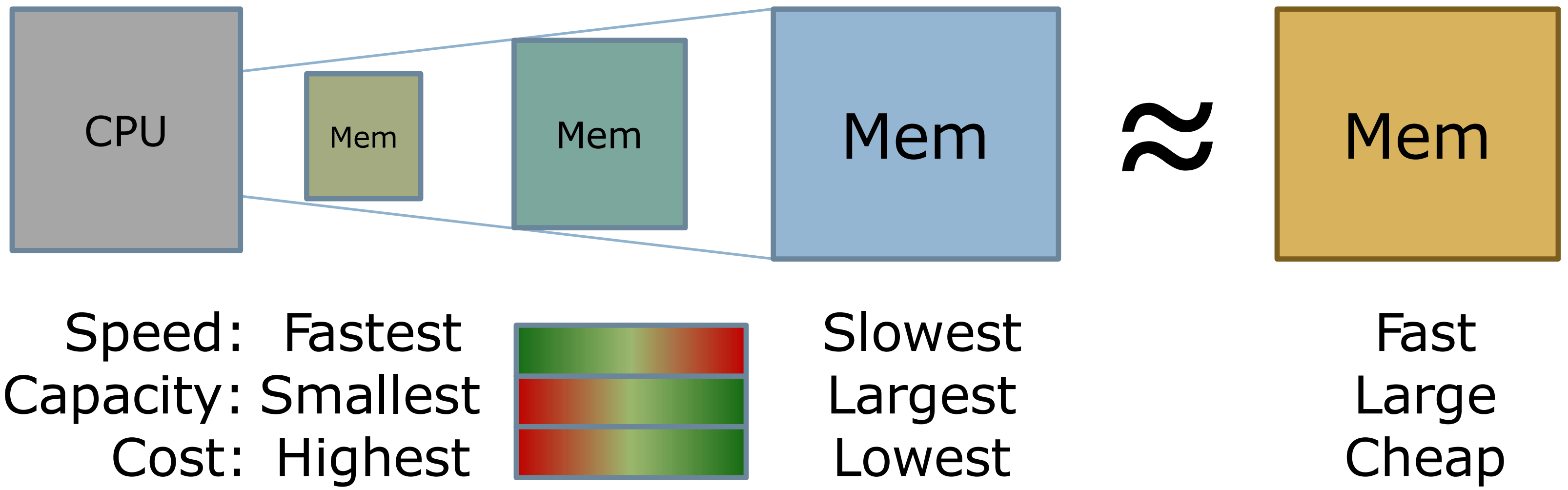
- Different technologies have vastly different tradeoffs
- Size is a fundamental limit, even setting cost aside:
  - Small + low latency, low energy, or
  - Large + high-latency, high energy
- Can we get best of both worlds? (large, fast, cheap)

The Memory Hierarchy

Want large, fast, and cheap memory, but... Large memories are slow (even if built with fast components)

Fast memories are expensive

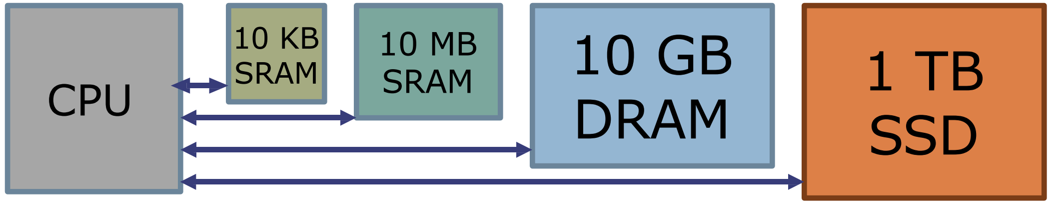
Solution: Use a hierarchy of memories with different tradeoffs to fake a large, fast, cheap memory



Memory Hierarchy Interface

Approach 1: Expose Hierarchy

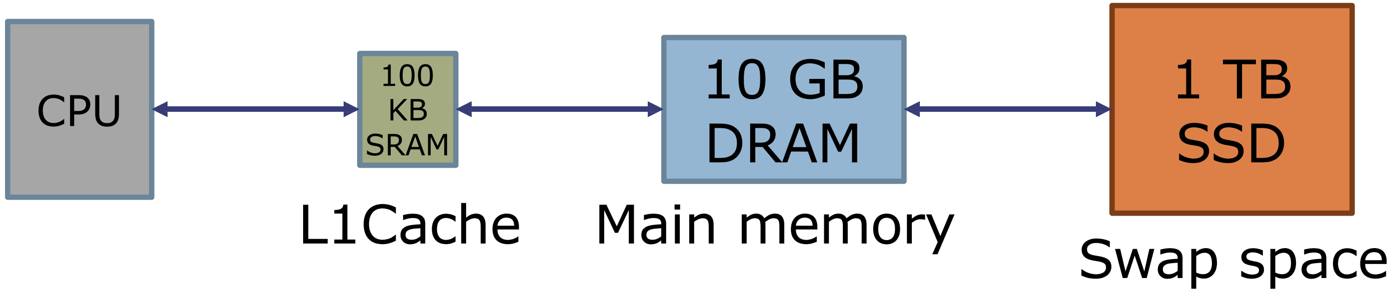
- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives



- Tell programmers: "Use them cleverly"

Approach 2: Hide Hierarchy

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns

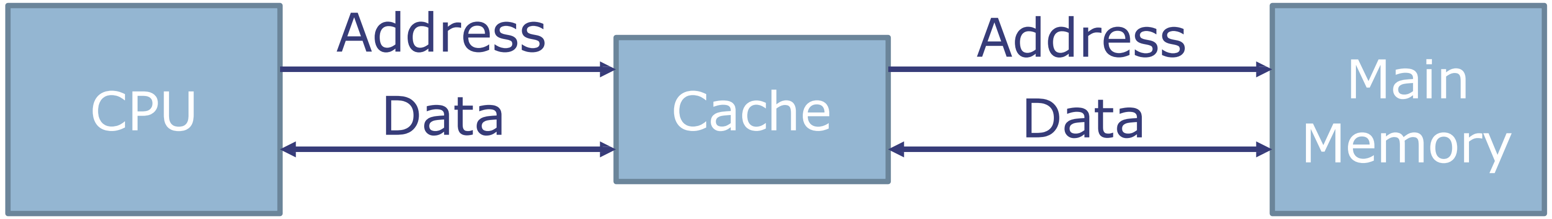


Common Predictable Patterns

- Two predictable properties of memory accesses:
  - **Temporal locality**: If a location has been accessed recently, it is likely to be accessed (reused) soon
  - **Spatial locality**: If a location has been accessed recently, it is likely that nearby locations will be accessed soon

Caches

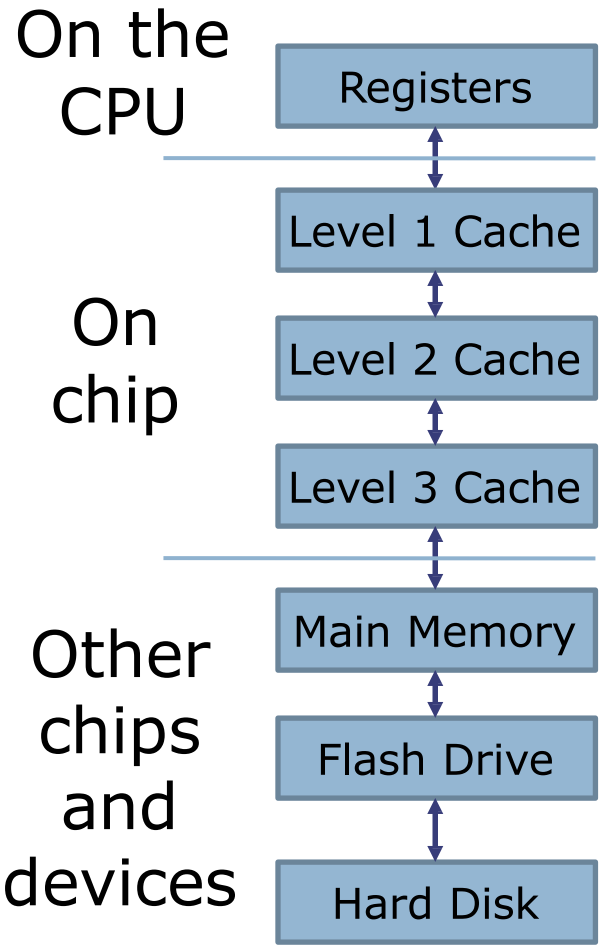
- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations



- Processor sends accesses to cache. Two options:
  - Cache hit: Data for this address in cache returned quickly
  - Cache miss: Data not in the cache
    - Fetch data from memory, and send it back to the processor
    - Retain this data in the cache (replacing some other data)

A Typical Memory Hierarchy

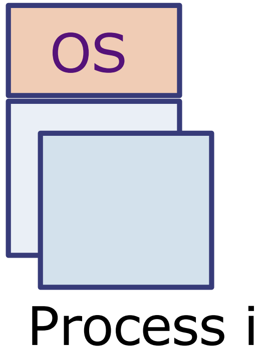
Computers use many levels of caches:



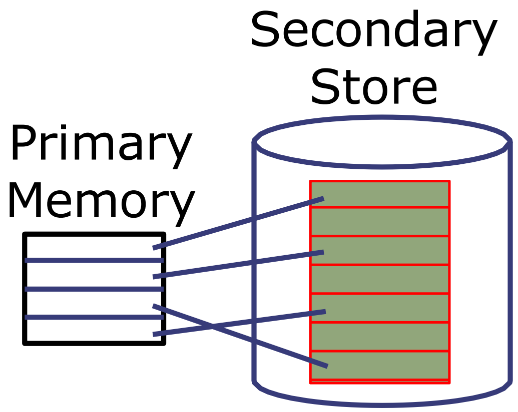
Access time	Capacity	Managed By
1 cycle	1 KB	Software/Compiler
2-4 cycles	32 KB	Hardware
10 cycles	256 KB	Hardware
40 cycles	10 MB	Hardware
200 cycles	10 GB	Software/OS
10-100us	100 GB	Software/OS
10ms	1 TB	Software/OS

Virtual Memory (VM) Systems: Illusion of a large, private, uniform store

- Protection & Privacy
  - Each process has a private address space



- Demand Paging
  - Provides the ability to run programs larger than the main memory

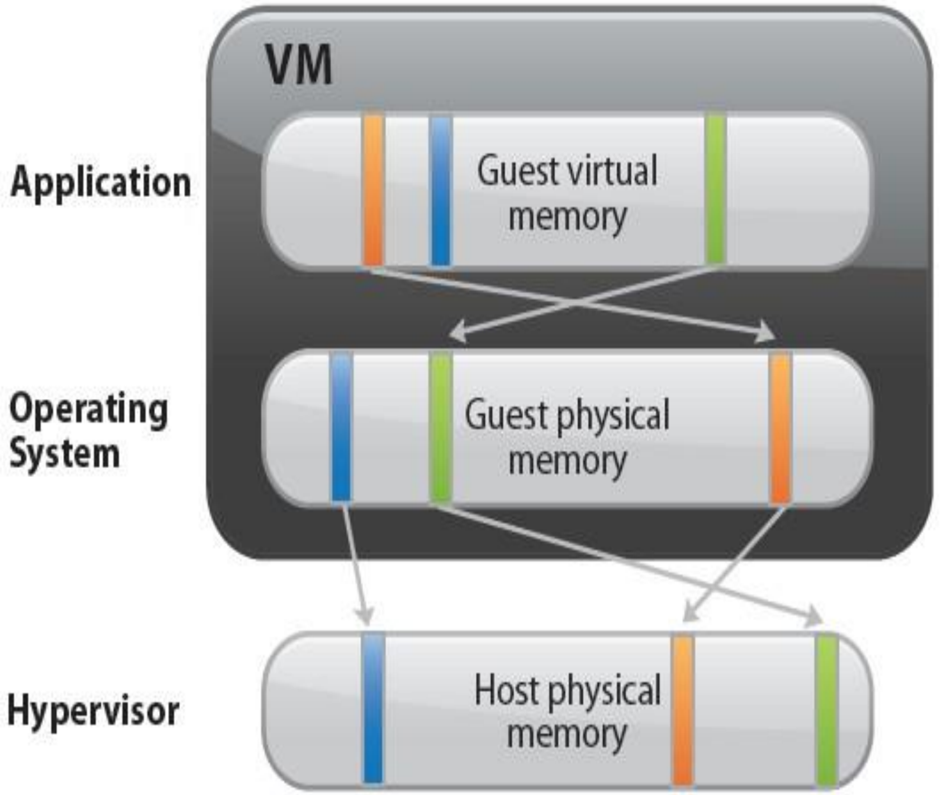


The price of VM is address translation on each memory reference

Virtualization- Memory Basics

- The virtual memory space, that is the guest's memory space, is divided into blocks, typically 4KB, called pages.
- The physical memory, that is the host's memory, is also divided into blocks, also typically 4KB (provides support for large pages of 2 MB)
- When the host's physical memory is full, the data for virtual pages that are not present in the host's physical memory are stored on disk.
- From the view of the application running inside the virtual machine, the hypervisor adds an extra level of address translation that maps the guest's physical address to the host's physical address. As a result, there are three virtual memory layers:
  - guest virtual memory,
  - guest physical memory,
  - host physical memory.

Memory Virtualization Basics: VMs

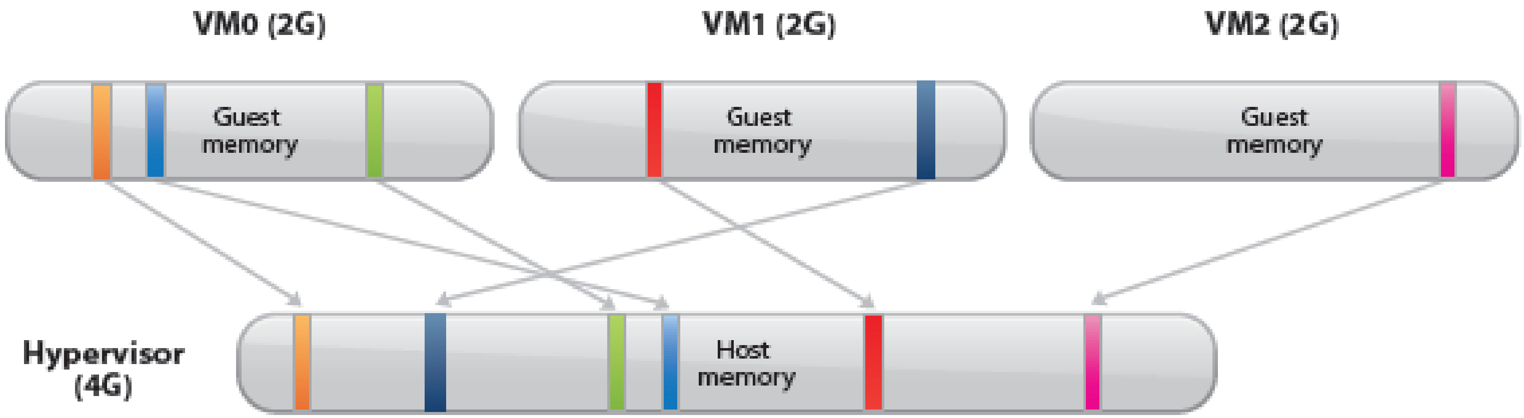


(a)

Memory Reclamation: Motivation

- For example, you can enable a host with 4G host physical memory to run three virtual machines with 2G guest physical memory each. How?

- Without memory overcommitment, only one virtual machine can be run



In order to effectively support memory overcommitment, the hypervisor must provide efficient host memory reclamation techniques.

Memory Reclamation Approaches

Several techniques to reclaim virtual machine memory:

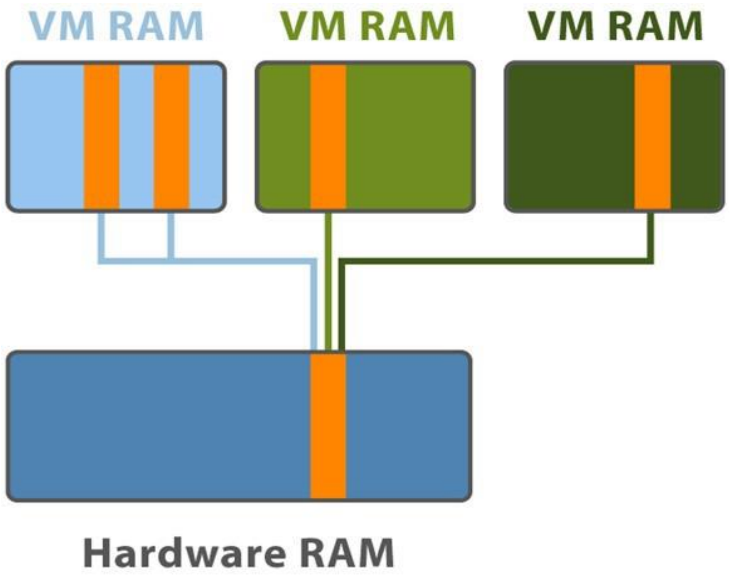
1. **Transparent page sharing (TPS)**—reclaims memory by removing redundant pages with identical content

2. **Ballooning**—reclaims memory by artificially increasing the memory pressure inside the guest

3. **Memory compression**—reclaims memory by compressing the pages that need to be swapped

Transparent Page Sharing (TPS)

- When multiple virtual machines are running, some of them may have identical sets of memory content.
  - For example, several virtual machines may be running the same guest operating system and have the same applications.
- With page sharing, the hypervisor can reclaim the redundant copies and keep only one copy.
- As a result, the total virtual machine host memory consumption is reduced and a higher level of memory overcommitment is possible.



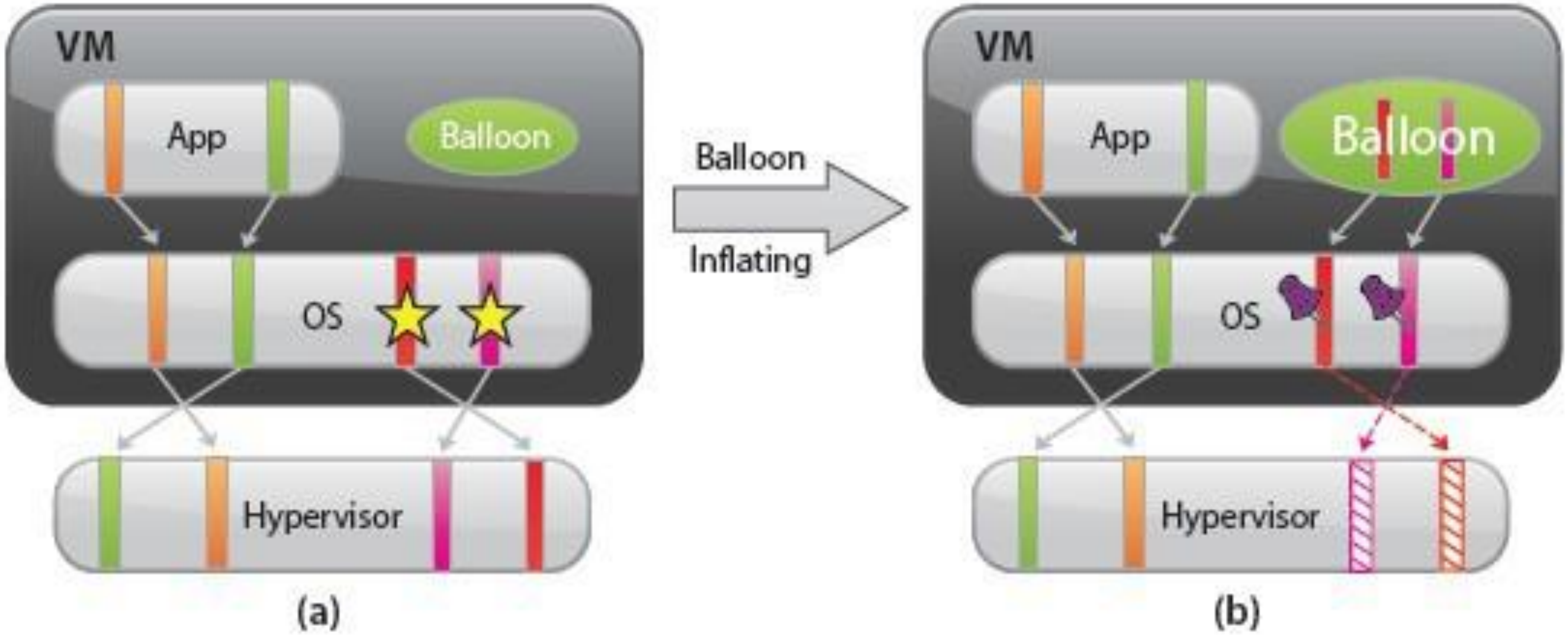
Ballooning

- When the hypervisor runs multiple guests and the total amount of the free host memory becomes low, none of the guests will free guest physical memory
    - guest OS cannot detect the host's memory shortage.
    - Guests run isolated from each other and don't even know they are virtual machines.

- Ballooning makes the guest operating system aware of the low memory status of the host.

- A balloon driver is loaded into the guest operating system and this driver, communicates with the hypervisor through a private channel.
    - If the hypervisor needs to reclaim guest memory, it sets a proper target balloon size for the balloon driver.

Ballooning: Inflating the balloon in a virtual machine



Memory Compression

If the swapped-out pages can be compressed and stored in a compression cache located in the main memory, the next access to the page only causes a page decompression which can be an order of magnitude faster than the disk access.

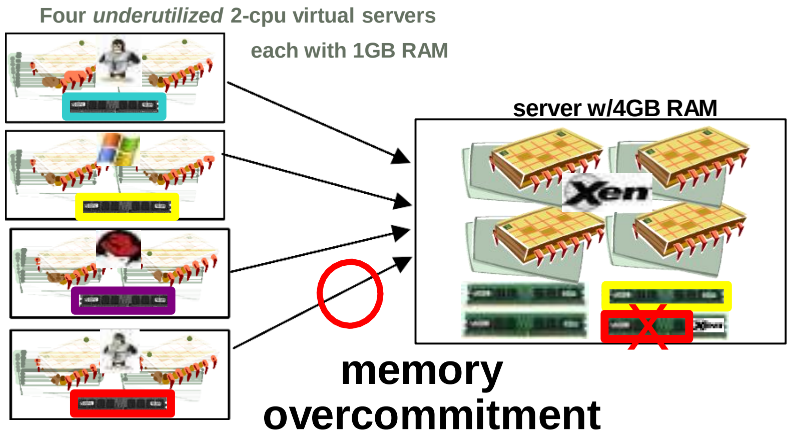
Reclaiming Memory: Combining Ballooning with compression

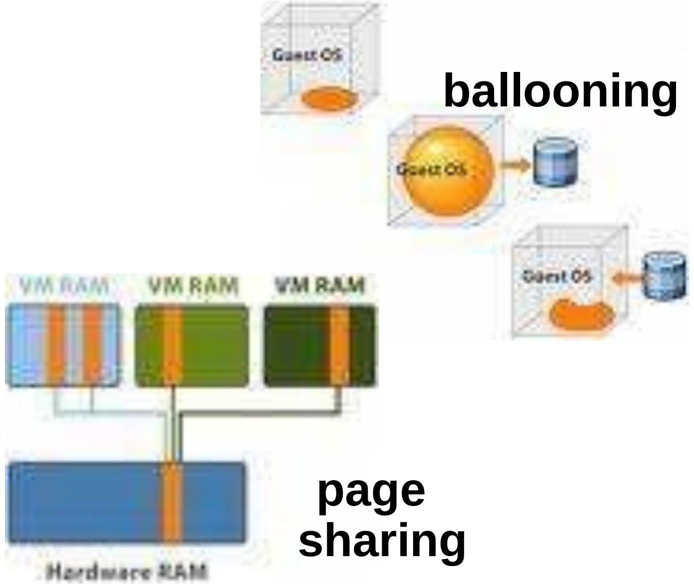
- If ballooning is not sufficient to reclaim memory or the host-free memory drops towards the hard threshold, the hypervisor starts to use swapping in addition to ballooning.

- During swapping, memory compression is activated as well.

Motivation for an efficient approach

- Memory is increasingly becoming a bottleneck in virtualized system
- Existing mechanisms have major holes





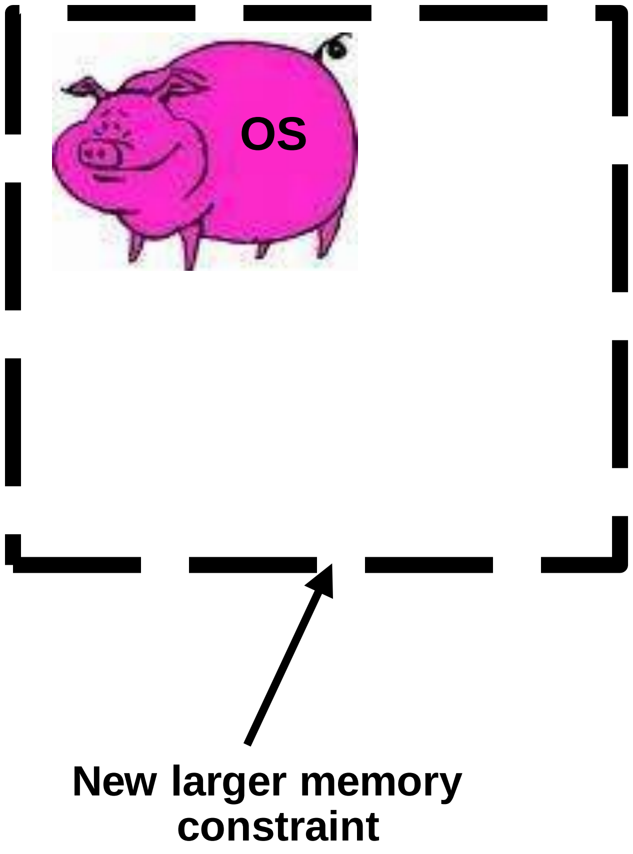
The Virtualized Physical Memory Resource Optimization Challenge

Optimize, across time, the distribution of machine memory among a maximal set of virtual machines by:

- measuring the current and future memory needs of each running VM
- reclaiming memory from those VMs that have an excess of memory and either:
  - providing it to VMs that need more memory or
  - using it to provide additional new VMs.

- *Goal:* *without* suffering a significant performance penalty

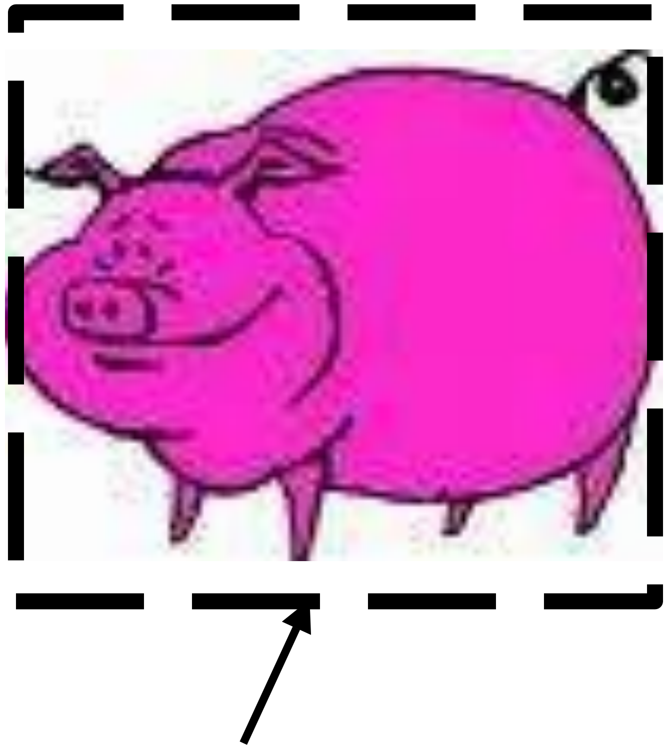
OS Physical Memory Management



- Operating systems are memory hogs!

If you give an operating system more memory…..

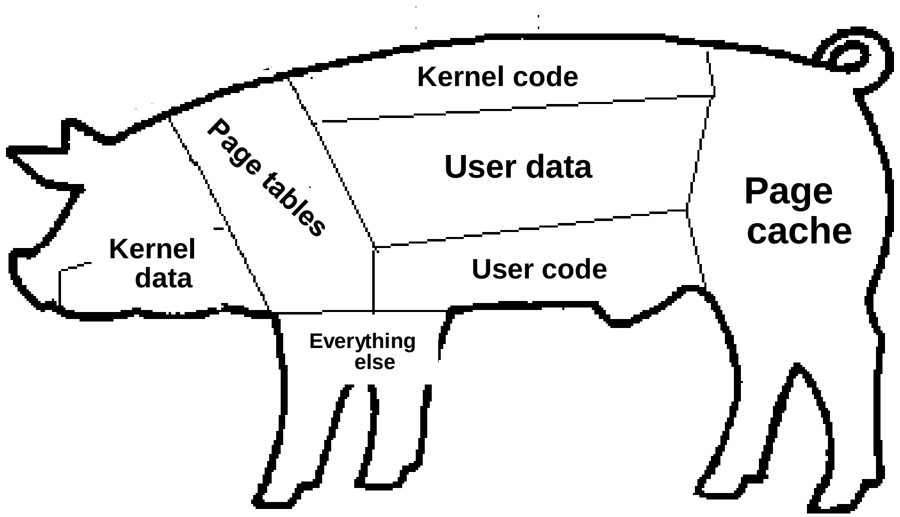
OS Physical Memory Management



- Operating systems are memory hogs!

…it uses up any memory you give it!

OS Physical Memory Management



- What does an OS do with all that memory?

OS Physical Memory Management

- What does an OS do with all that memory?

…much of the time mostly page cache

… some of which will be useful in the future

… and some of which is ***wasted***

VMM Physical Memory Management

in the presence of ballooning

Using ballooning to take memory away:

- not instantaneous
- guest can't predict future needs
  - good pages are evicted along with the bad
- don't know how much/fast to balloon
  - Too much or too fast
    � thrashing or the Out-Of-Memory(OOM) killer

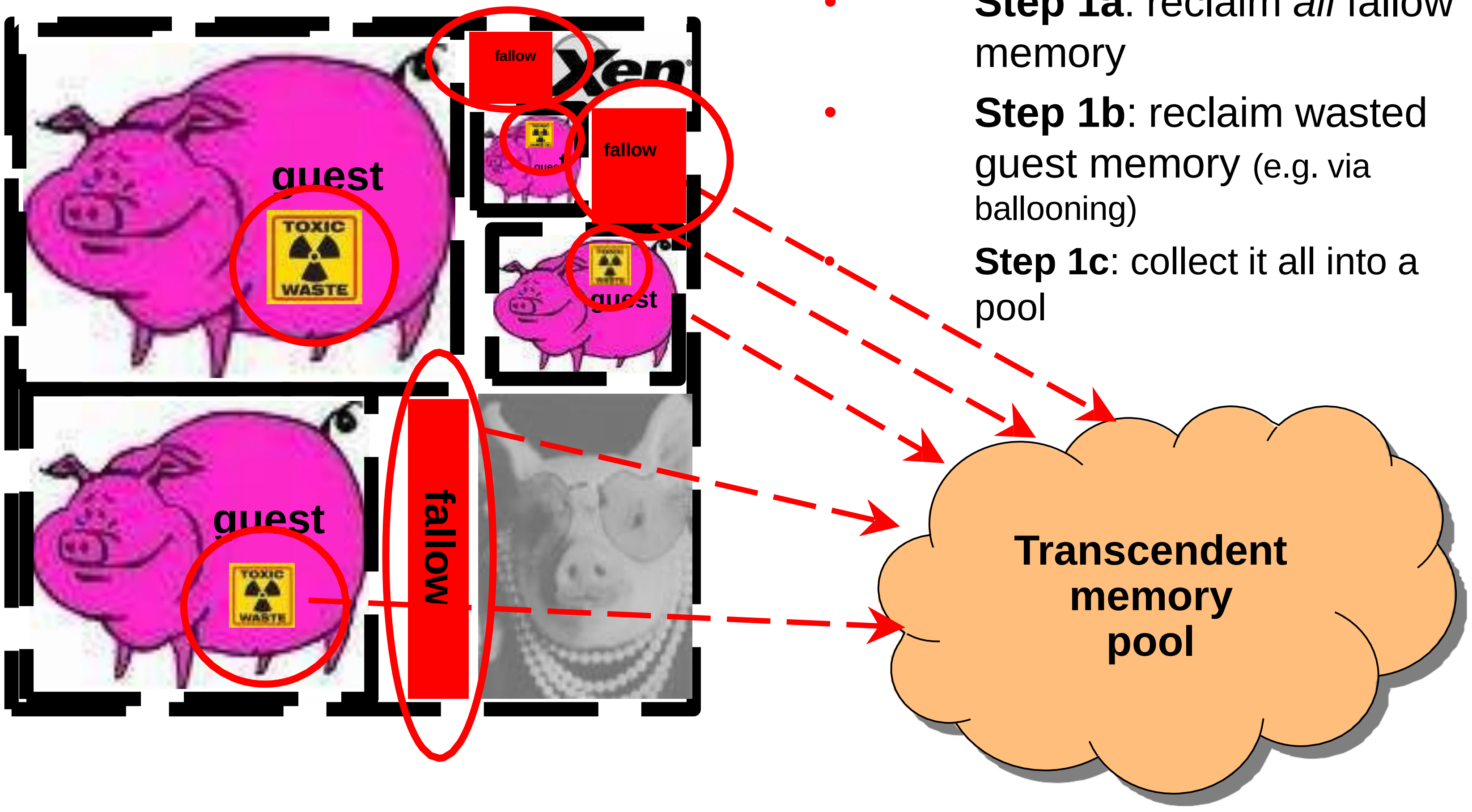
Why this IS a hard problem: Summary

- OS's use as much memory as they are given
  - but cannot predict the future so often guess wrong
  - often much memory owned by an OS is wasted

- Ballooning helps but:
  - can't predict future memory needs of guests
  - the price of incorrect guesses

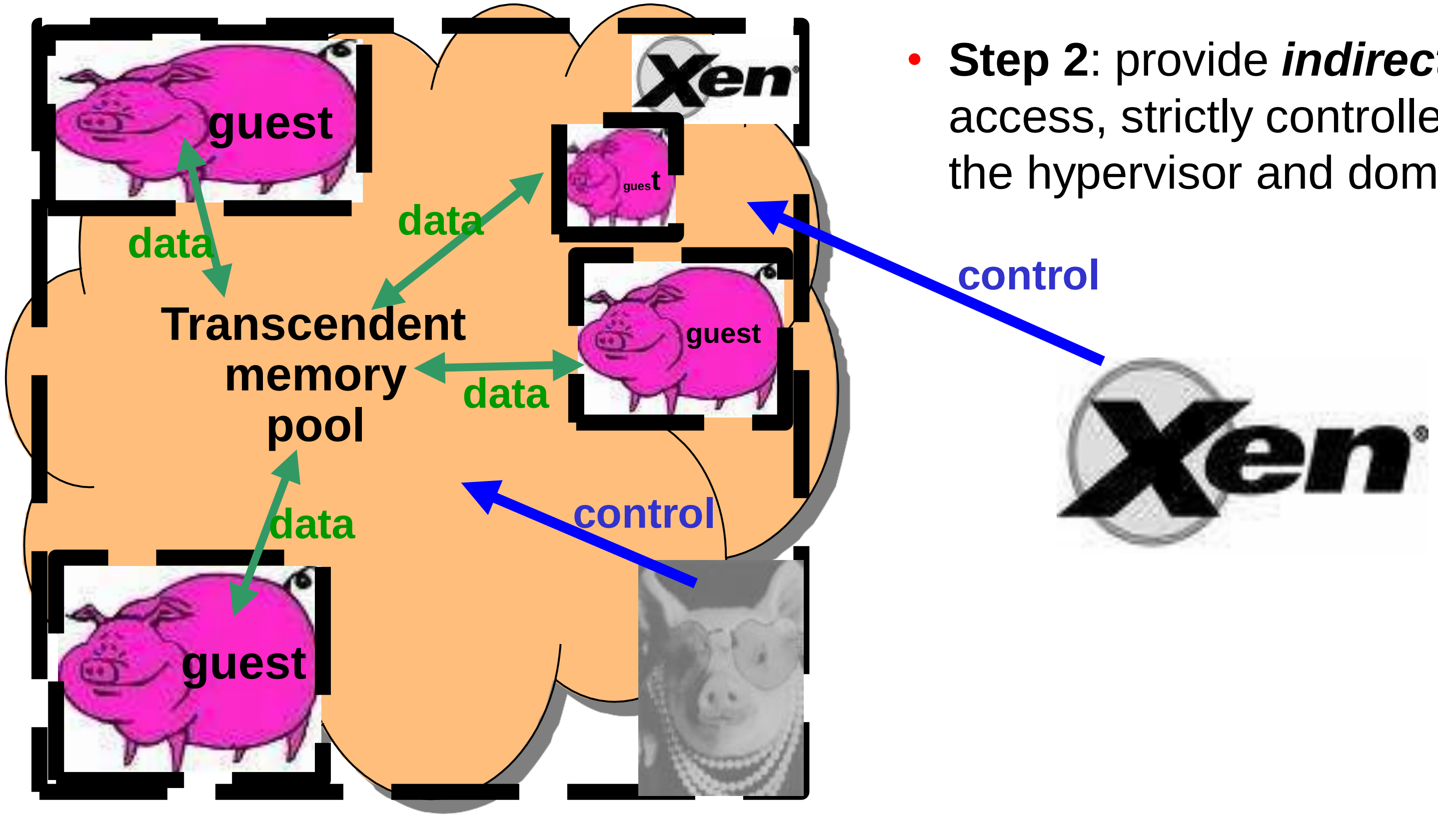
◆ NEED A NEW APPROACH TO VIRTUALIZE PHYSICAL MEMORY MANAGEMENT!!

Transcendent memory: creating the transcendent memory pool



- **Step 1a**: reclaim *all* fallow memory
- **Step 1b**: reclaim wasted guest memory (e.g. via ballooning)
- **Step 1c**: collect it all into a pool

Transcendent memory: creating the transcendent memory pool



- **Step 2**: provide ***indirect*** access, strictly controlled by the hypervisor and dom0

Data Center and Cloud

Data Centers and Cloud

- Large server and storage farms
  - 1000s of servers
  - Many TBs or PBs of data
- Used by
  - Enterprises for server applications
  - Internet companies
  - Some of the biggest DCs are owned by Google, Facebook, etc.
- Used for
  - Data processing
  - Web sites
  - Business apps

Traditional vs “Modern”

- Data Center architecture and its usages have been changing over the past decade

- Traditional - static
  - Applications run on physical servers
  - System administrators monitor and manually manage servers
  - Use Storage Array Networks (SAN) or Network Attached Storage (NAS) to hold data

- Modern - dynamic, larger scale
  - Run applications inside virtual machines
  - Flexible mapping from virtual to physical resources
  - Increased automation allows the larger scale

Inside a Data Center

- Giant warehouse filled with:
  - Racks of servers
  - Storage arrays
  - Cooling infrastructure
  - Power converters
  - Backup generators

Modular Data Center

- Use shipping containers
- Each container is filled with thousands of servers
- Can easily add new containers
  - "Plug and play"
  - Just add electricity
- Pre-assembled, cheaper
- Allows data center to be easily expanded

Economy of Scale

- Larger data centers can be cheaper to buy and run than smaller ones
  - Lower prices for buying equipment in bulk
  - Cheaper energy rates
- Automation allows a small number of sys admins to manage thousands of servers
- General trend is towards larger mega data centers
  - 100,000s of servers

Data Center Challenges

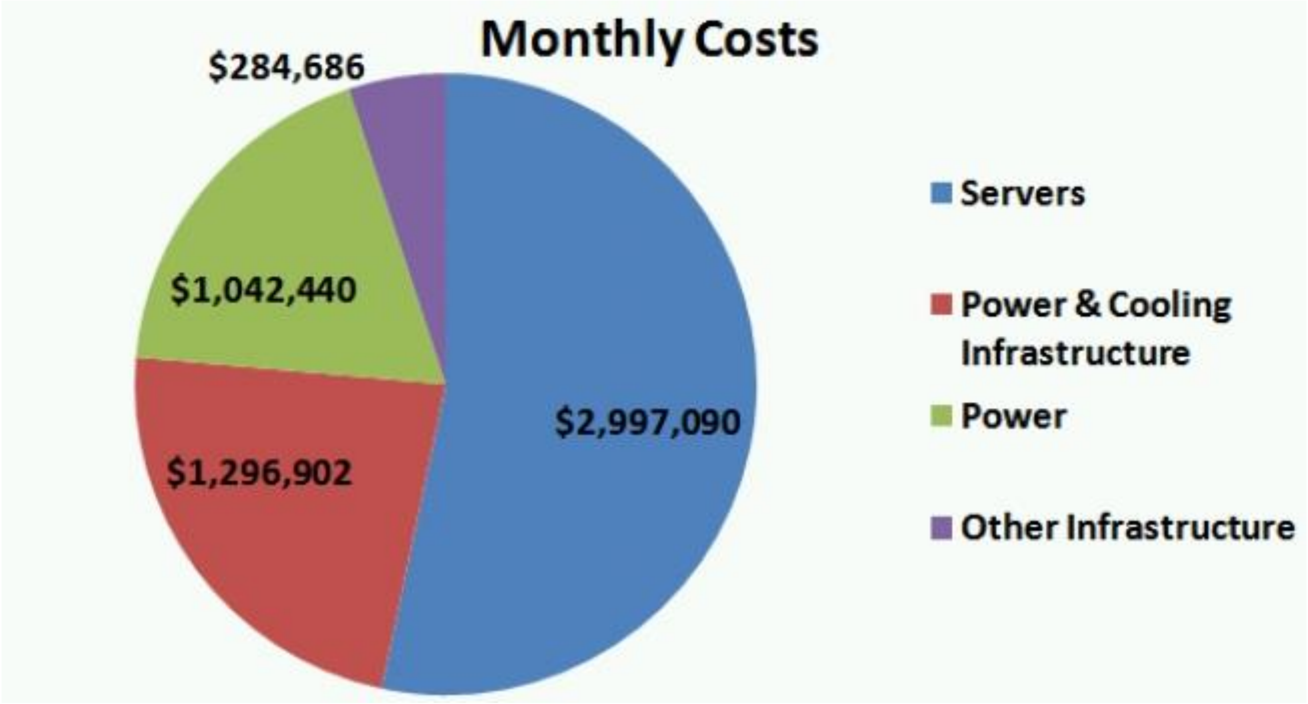
- Resource management
  - How to efficiently use server and storage resources?
  - Many apps have variable, unpredictable workloads
  - Want high-performance **and** low cost
  - Automated resource management
  - Performance profiling and prediction

- Energy Efficiency
  - Servers consume huge amounts of energy
  - Want to be “green”
  - Want to save money

- Other factors
  - Design & Construction of a DC

Data Center Cost

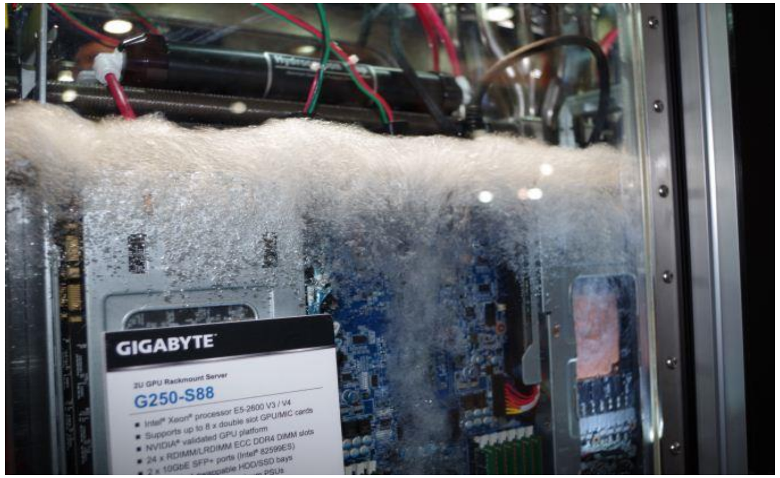
- Running a data center is expensive
- Efficiency captured as PUE (Power Usage Effectiveness)
  - Ratio of IT Power / Total Power

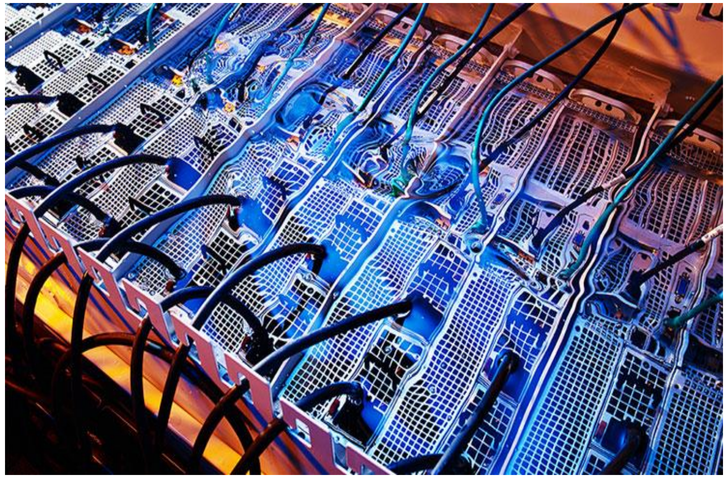


https://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/

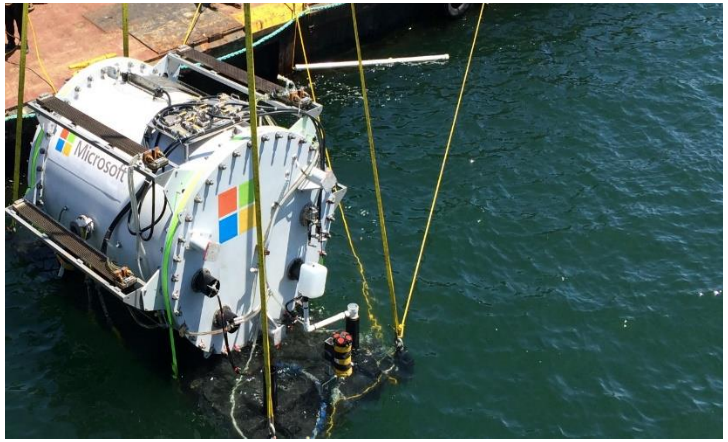
Some Solutions

- Immersion cooling





- DC in the Ocean



What is the cloud?

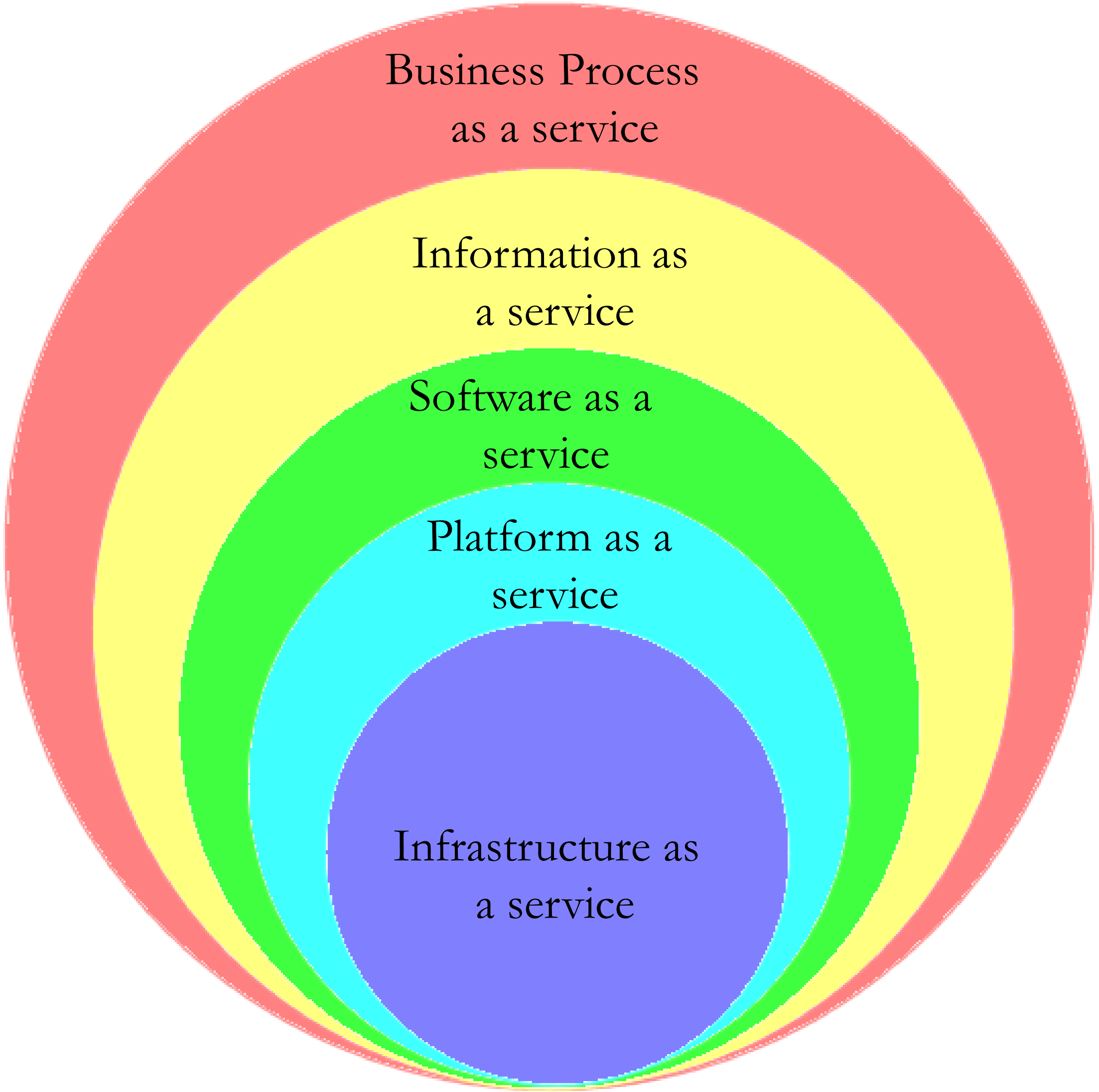
- The cloud is not a physical entity, it's a network of remote servers around the globe that are hooked together and operate as a single ecosystem. These servers are designed to
  - store and manage data
  - run applications
  - deliver content
- From a local or personal computer to online access
  - the information/data will be ubiquitous.
- Summary
  - Remotely available
  - Pay-as-you-go
  - High scalability
  - Shared infrastructure

The Cloud Services

- **Software as a service (SaaS)**
  - Hosted applications, managed by provider
  - SaaS applications are designed for end-users, delivered over the web
- **Platform as a Service (PaaS)**
  - Platform to let you run your own apps
  - Provider handles scalability
  - PaaS is the set of tools and services designed to make coding and deploying those applications quick and efficient
- **Infrastructure as a Service (IaaS)**
  - IaaS is the hardware and software that powers it all – servers, storage, networks, and operating systems

5 levels of abstraction in cloud computing services

NIST definition +
BPaaS &
IaaS

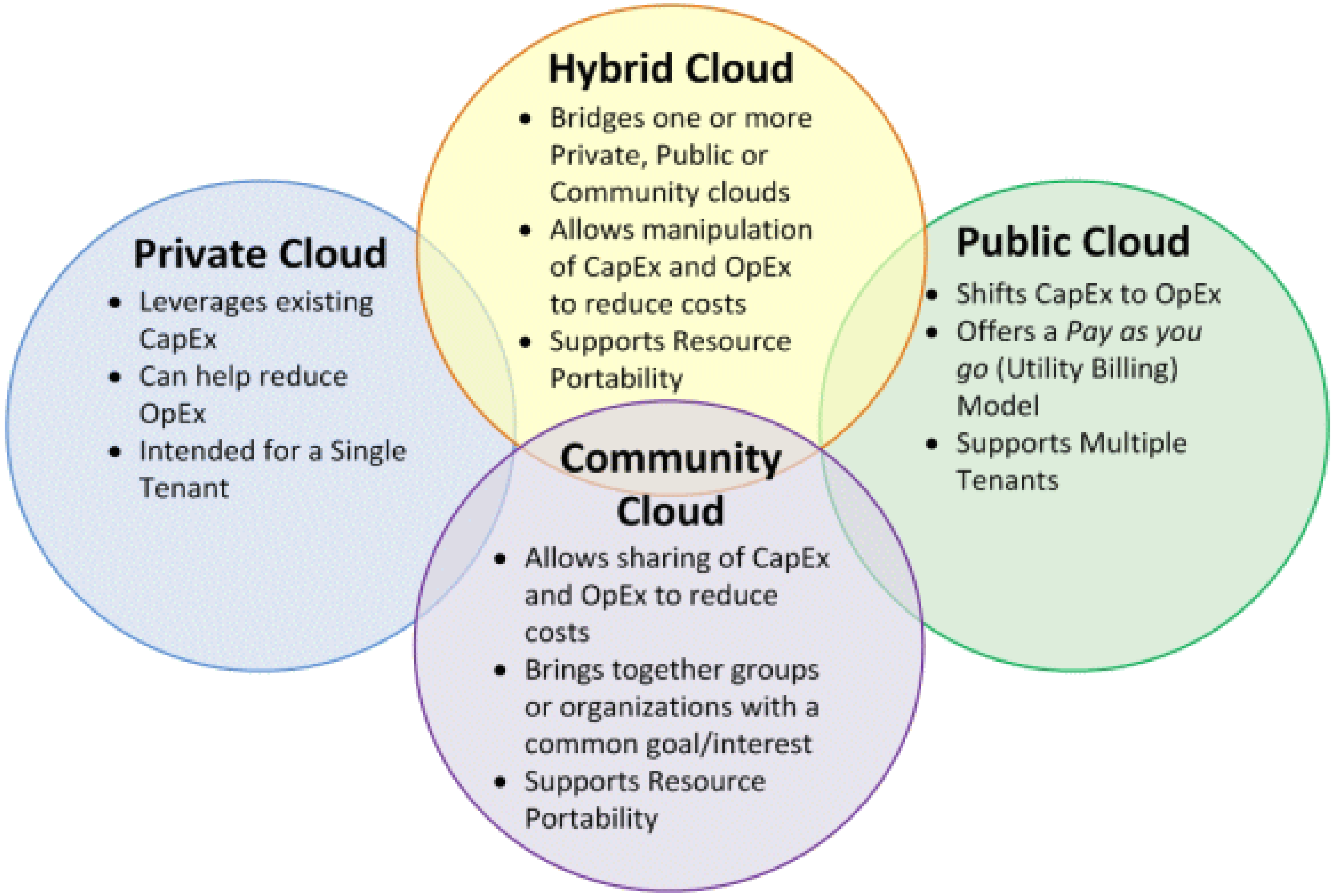


Public or Private

- Not all enterprises are comfortable with using public cloud services
  - Don't want to share CPU cycles or disks with competitors
  - Privacy and regulatory concerns
- Private Cloud
  - Use cloud computing concepts in a private data center
  - Automate VM management and deployment
  - Provides the same convenience as public cloud
  - May have a higher cost
- Hybrid Model
  - Move resources between private and public depending on load

Four Cloud Deployment Models:

Key functions & cost impacts



Cloud Challenges

- Privacy / Security
  - How to guarantee isolation between client resources?
- Extreme Scalability
  - How to efficiently manage 1,000,000 servers?
- Programming models
  - How to effectively use 1,000,000 servers?

Cloud services and products (examples)

References

- Above the Clouds: A Berkeley View of Cloud Computing, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf
- Andrew Birrell and Bruce Nelson, Implementing RPCs, ACM Transactions on Computer Systems, Vol. 2, No. 1, Pages 39-59, February 1984.
- B. Bershad, T. Anderson, E. Lazowska, and H. Levy, Lightweight Remote Procedure Call, Proceedings of the 12th ACM Symposium on Operating Systems Principles, Operating Systems Review, Vol. 23, No. 5, Pages 12-113, December 1989
- Distributed Systems: Principles and Paradigms by Tanenbaum and van Steen, Chapter 4.1~4.2.