CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2021 L25

Virtualization

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

FAQ

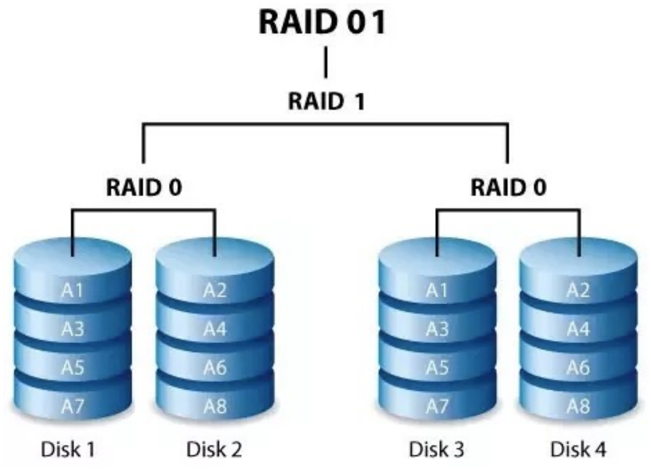
- **Parity bit(s):** Extra bits obtained using data bits. Used for error detection/correction.
- **Ex:** Parity $\text{bit}_i = \text{word}_0\ \text{bit}_i \oplus .. \oplus \text{word}_n\ \text{bit}_i$

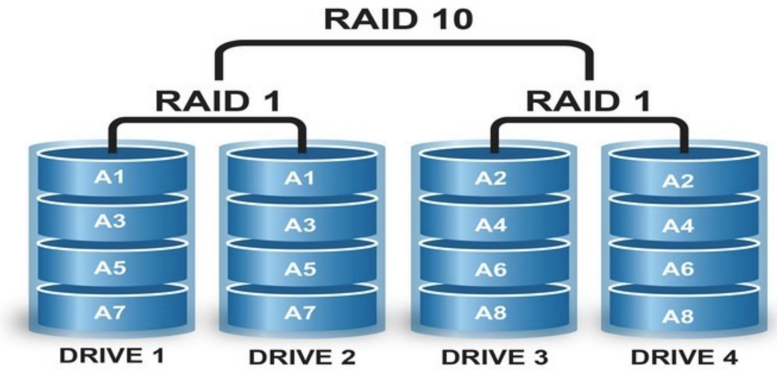
  = bit needed make 1's even
  - Block parity: bit-by-bit parity for all disks
  - RAID 5, 6: Parity blocks are distributed among the disks.
- RAID recovery:
  - RAID 5,6: rebuild using available data, parity info
  - RAID 1: Copy info from good mirror
- How do we know a disk is corrupted? Use of CRC redundancy at a lower level.
- There are many RAID configurations are possible: using combinations of striping, mirroring, parity.

RAIDs: Nested systems

Nested systems: combine striping with mirroring/parity

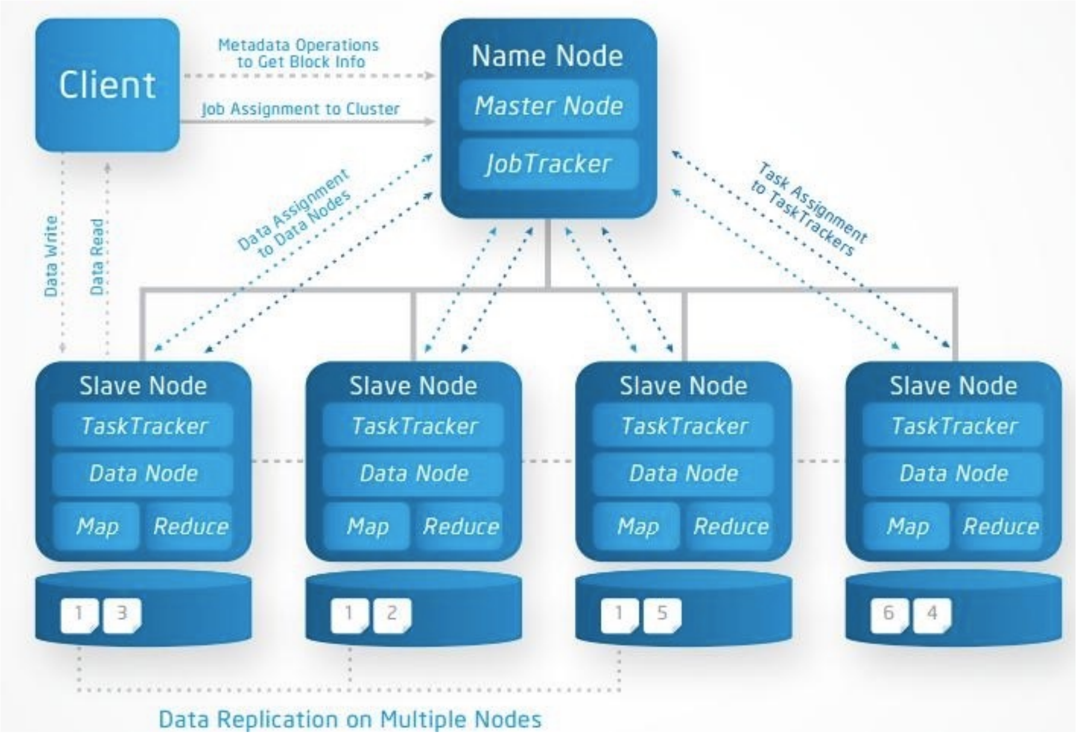
- RAID 01: Two RAID 0 systems (with striping) mirrored
- RAID 10: Multiple RAID 1 systems (with mirroring) striped.





Colorado State University

HDFS Architecture



http://a4academics.com/images/hadoop/Hadoop-Architecture-Read-Write.jpg

Secondary Name node
If primary fails

Name Node: metadata giving where blocks are physically located
Data Nodes: hold blocks of files (files are distributed

Q. What do I need to know? motivation, approaches, concepts

Colorado State University

HDFS

HDFS fault tolerance

- Three copies of each data block.
- The Name Node recreates a Data Node if no heartbeat is received from it.
- The Standby Name Node takes over if the Name Node goes down.

Not needed: move massive data

- Most computation is done locally in parallel and partial results are transmitted.

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Virtualization

Q. Do we really need to know about virtualization?
Yes, absolutely.

Slides based on

- Various sources

Virtualization

- A Virtual scheme provides a simpler perspective of a Physical scheme. Needs mapping.
  - Example: each process a separate virtual address space and resource allocation.
  - OS allocates physical memory and disk space and handles mapping.
- System ("machine") virtualization
  - A machine needs its own CPU, memory, storage, I/O to run its OS and apps. "Machine" = {CPU, memory, storage, I/O, OS, apps}
  - Needs to be isolated from other machines.
  - Can "virtual machines" be allocated resources from physical hardware, with allocation done by a Virtual Machine Monitor (VMM or hypervisor)?
  - Can a virtual machine be "migrated" from one physical system to another?

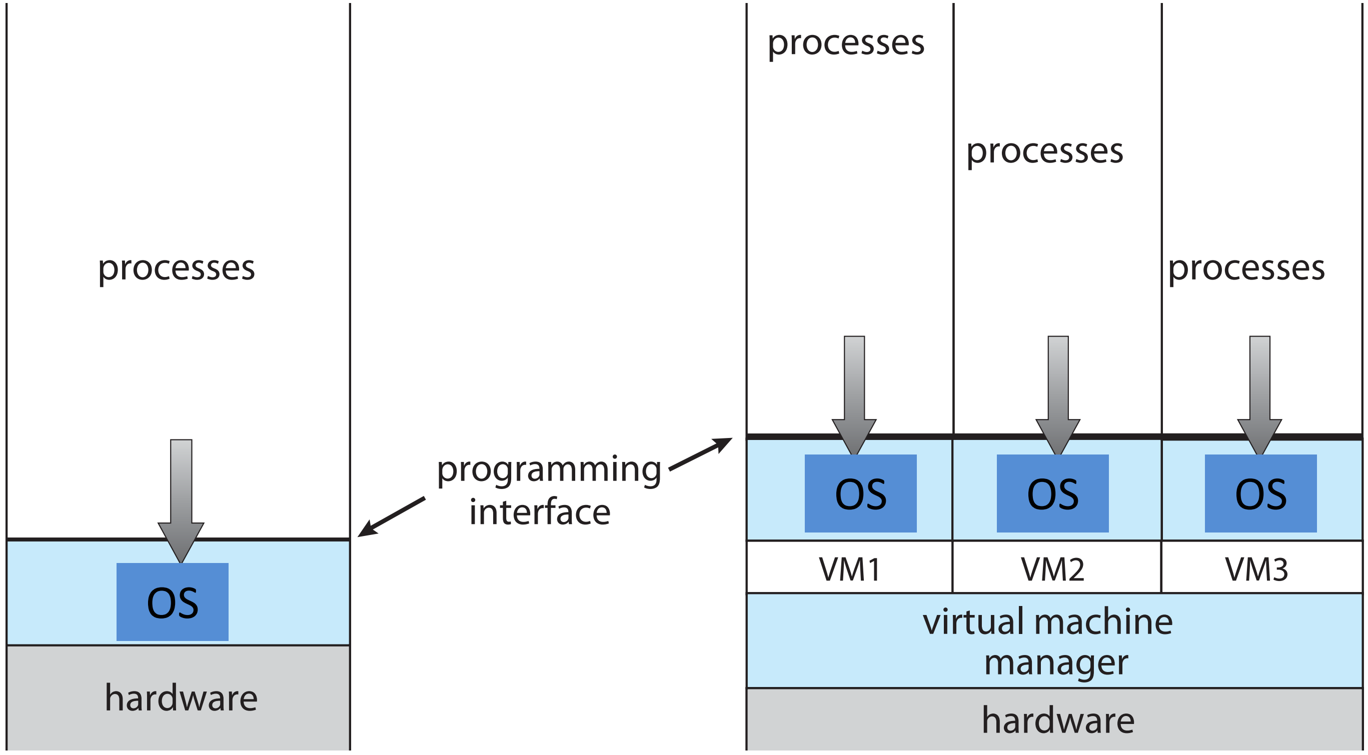
Virtualization: Why?

- Processors have gradually become very powerful
- Dedicated servers can be very underutilized (5-15%)
- Virtualization allow a single server to support several virtualized servers: typical consolidation ratio 6:1
- Power cost a major expense for data centers
  - Companies frequently locate their data centers in the middle of nowhere where power cost is low
- If a hardware server crashes, would be nice to migrate the load to another one.
- A key component of cloud computing

Virtual Machines (VM)

- Virtualization technology enables a single PC/server to simultaneously run multiple operating systems or multiple sessions of a single OS.
- A machine with virtualization can host many applications, including those that run on different operating systems, on a single platform.
- The host *operating system (or VMM)* can support a number of virtual machines, each of which has the characteristics of a particular OS.
- The software that enables virtualization is a **virtual machine monitor (VMM),** or **hypervisor.**

Virtual Machines (VM)



Traditional
physical machine

Hypervisor with
virtual machines

I have replaced the word kernel by OS.

Colorado State University

“Strictly Virtualizable”

A processor or mode of a processor is *strictly virtualizable* if, when executed in a lesser privileged mode:

- all instructions that access privileged state trap
- all instructions either trap or execute identically
- Intel, AMD processors have mode that enables such virtualization.
  - Intel CPUs: VT (Virtualization Technology)
  - AMD CPUs: SVM (Secure Virtual Machine)

OUR SERVERS ARE USING TOO MUCH ELECTRICITY. WE NEED TO VIRTUALIZE.
scottadams@aol.com
www.dilbert.com
I DID MY PART BY READING ABOUT VIRTUALIZATION IN A TRADE JOURNAL. NOW YOU DO THE SOFTWARE PART.
3-12-08 ©2008 Scott Adams, Inc./Dist. by UFS, Inc.
WHY IS YOUR PART TAKING SO LONG?

© Scott Adams, Inc./Dist. by UFS, Inc.

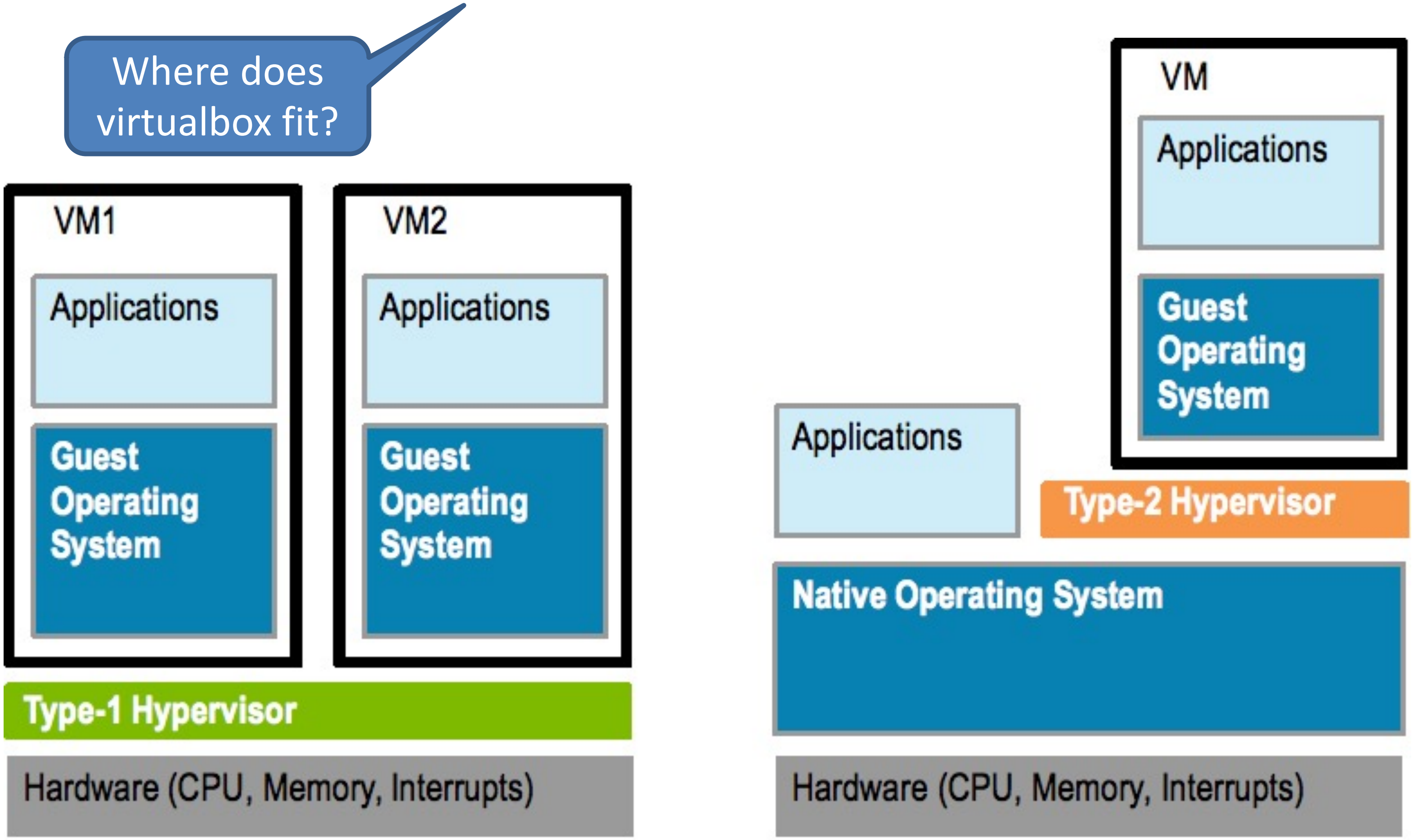
Implementation of VMMs

- **Type 1 hypervisors** - Operating-system-like software built to provide virtualization. Runs on ‘bare metal”.
  - Including **VMware ESX**, Joyent SmartOS, and Citrix XenServer
- Also includes general-purpose operating systems that provide standard functions as well as VMM functions
  - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
  - Including VMware Workstation and Fusion, Parallels Desktop, and Oracle **VirtualBox**

Implementation of VMMs

A higher layer uses services of the lower layers.

Where does virtualbox fit?



https://microkerneldude.files.wordpress.com/2012/01/type1-vs-2.png

OS vs Hypervisor (VMM)

- **Isolation**
  - OS isolates processes from each other
  - **Hypervisor isolate VMs from each other**
- **Resource allocation**
  - OS: allocates resources to processes, logical to physical mapping
  - **Hypervisor allocates resources to Virtual Machines, “physical” to hardware mapping.** Resources: CPU, Memory, Disk, mouse/keyboard, network address ..
- **Concurrency**
  - OS can run processes concurrently
  - **Hypervisor can run VMs concurrently**
- **Access to hardware resources**
  - OS provides as a service
  - **Hypervisor provides directly (type 1) or indirectly (type 2)**

Virtualization support

- Terminology:
  - Guest Operating System
    - The OS running on top of the hypervisor (both types)
  - Host Operating System
    - For a type 2 only: the OS that runs on the hardware ¨executions
- Create isolated environment in which VMs can be run
- When a guest OS is started in an isolated environment continues to run until it causes an exception and traps to the hypervisor
  - For e.g. by executing an I/O instruction
- Set of operations that trap is controlled by a hardware bit map set by hypervisor
  - trap-and-emulate approach becomes possible

Virtual Machine (VM) as a software construct

- Each VM is configured with
  - some number of processors,
  - some amount of RAM, storage resources,
  - and connectivity through the network ports
- Once the VM is created it can be
  - activated on like a physical server, loaded with an operating system and software solutions, and used just like a physical server
- But unlike a physical server,
  - VM only sees the resources it has been configured with, not all of the resources of the physical host itself
- The hypervisor facilitates the translation and I/O between the virtual machine and the physical server.

Half a CPU?

Virtual Machine (VM) as a set of files

- Configuration file describes the attributes of the virtual machine containing
  - server definition,
  - how many virtual processors (vCPUs)
  - how much RAM is allocated,
  - which I/O devices the VM has access to,
  - how many network interface cards (NICs) are in the virtual server
  - the storage that the VM can access
- When a virtual machine is instantiated, additional files are created for logging, for memory paging etc.
- Copying a VM produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself

Virtualization benefits

- Run multiple, OSes on a single machine
  - **Consolidation**, app dev, …
- Security: Host system protected from VMs, VMs protected from each other
  - Sharing though shared file system volume, network communication
- Freeze, **suspend**, running VM
  - Then can move or copy somewhere else and **resume**
    - **Live migration**
  - Snapshot of a given state, able to restore back to that state
  - **Clone** by creating copy and running both original and copy
- Hence – cloud computing

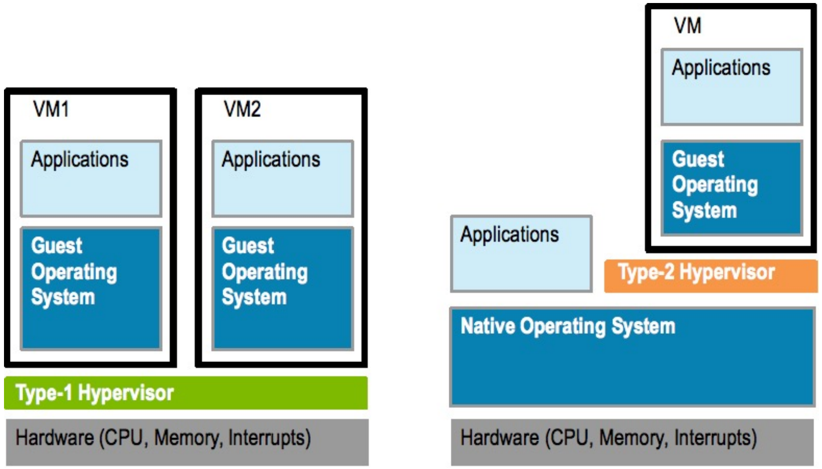
Building Block – Trap and Emulate

- When Guest OS attempts to execute a privileged instruction, what happens?
  - Causes a trap
  - VMM gains control, analyzes error, executes operation as attempted by guest directly if type 1, indirectly if type 2
  - Returns control to guest in user mode
  - Known as **trap-and-emulate**
- Trap-and-emulate was the technique used for implementing floating point instructions in CPUs without floating point coprocessor

Sensitive instructions

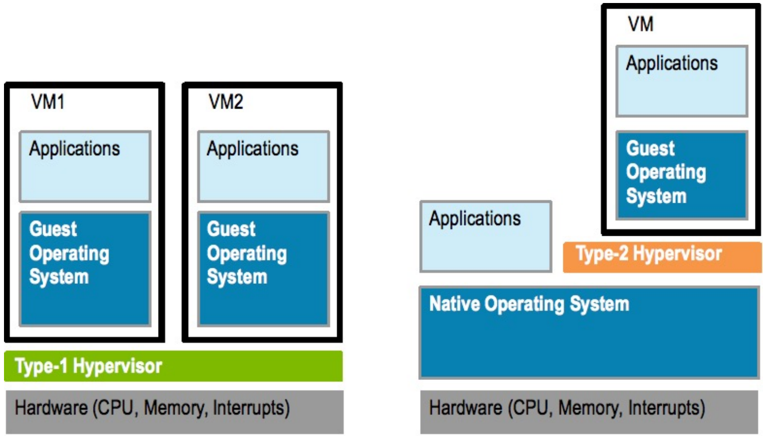
- Some CPUs didn't have clean separation between privileged and non-privileged instructions
  - Sensitive instructions
    - Consider Intel x86 `popf` instruction
    - If CPU in privileged mode -> all flags replaced
    - If CPU in user mode -> on some flags replaced
      - No trap is generated
- Binary translation (complex) solves the problem
  1. If guest VCPU is in user mode, guest can run instructions natively
  2. If guest VCPU in kernel mode (guest believes it is in kernel mode)
     1. VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
     2. Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)
  3. Cached translations can reduce overhead
- Not needed in newer (>2005) processors with virtualization support. Support may need to be enabled.
  - Intel CPUs: VT (Virtualization Technology)
  - AMD CPUs: SVM (Secure Virtual Machine)

Type 1 Hypervisors



- **Run on top of bare metal**
- **Guest OSs believe they are running on bare metal, are unaware of hypervisor**
  - are not modified
  - Better performance
- **Choice for data centers**
  - Consolidation of multiple OSes and apps onto less HW
  - Move guests between systems to balance performance
  - Snapshots and cloning
- **Hypervisor creates runs and manages guest OSes**
  - Run in kernel mode
  - Implement device drivers
  - provide traditional OS services like CPU and memory management
- Examples: VMWare esx (dedicated) , Windows with Hyper-V (includes OS)

Type 2 Hypervisors



- **Run on top of a host OS**
- VMM is simply a process, managed by host OS
  - host doesn't know they are a VMM running guests
- Poorer overall performance because can't take advantage of some HW features
- Host OS is just a regular one
  - Individuals could have Type 2 hypervisor (e.g. Virtualbox) on native host (perhaps windows), run one or more guests (perhaps Linux, MacOS)

Full vs Para-virtualization

- Full virtualization: Guest OS is unaware of the hypervisor. It thinks it is running on bare metal.
- Para-virtualization: Guest OS is modified and optimized. It sees underlying hypervisor.
  - Introduced and developed by Xen
    - Modifications needed: Linux 1.36%, XP: 0.04% of code base
  - Does not need as much hardware support
  - allowed virtualization of older x86 CPUs without binary translation
  - Not used by Xen on newer processors

CPU Scheduling

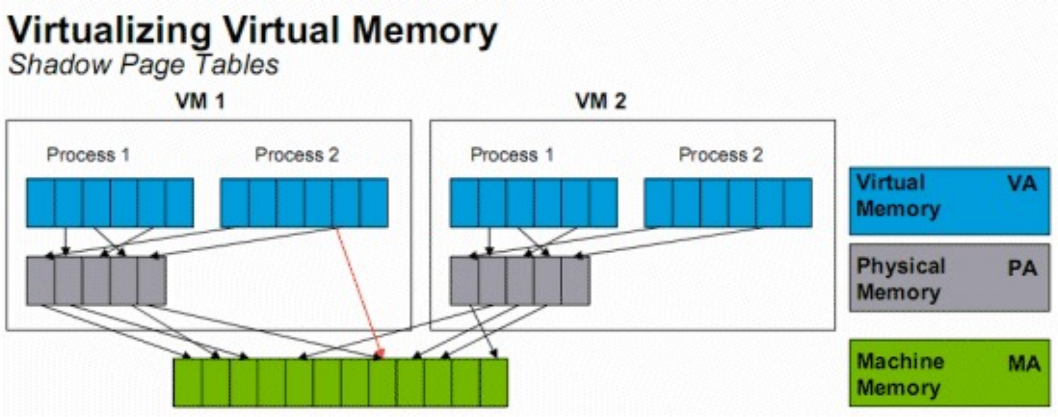
- One or more virtual CPUs (vCPUs) per guest
  - Can be adjusted throughout life of VM
- When enough CPUs for all guests
  - VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs (CPU overcommitment)
  - VMM can use scheduling algorithms to allocate vCPUs
  - Some add fairness aspect
- Oversubscription of CPUs means guests may get CPU cycles they expect
  - Time-of-day clocks may be incorrect
  - Some VMMs provide application to run in each guest to fix time-of-day

Memory Management

Memory mapping:

- On a bare metal machine: OS uses page table/TLB to map Virtual page number (VPN) to Physical page number (PPN) (physical memory is shared). Each process has its own page table/TLB.
  - VPN -> PPN (we have seen this already, we called PPN a frame number)
- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map "PPN" of each VM to MPN (Shadow page table)

  PPN ->MPN



Memory Management

- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

  PPN ->MPN

- Where is this done?
  - Has to be done by hypervisor type 1. Guest OS knows nothing about MPN.
  - Page Table/TLB updates are trapped to VMM.

    It needs to do VPN->PPN ->MPN.
  - It can do VPN->MPN directly (VMware ESX)

Colorado State University

Handling memory oversubscription

Oversubscription solutions:

- *Deduplication* by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests
- Double-paging, the guest page table indicates a page is in a physical frame but the VMM moves some of those to disk.
- Install a pseudo-device driver in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)
  - Balloon memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available.

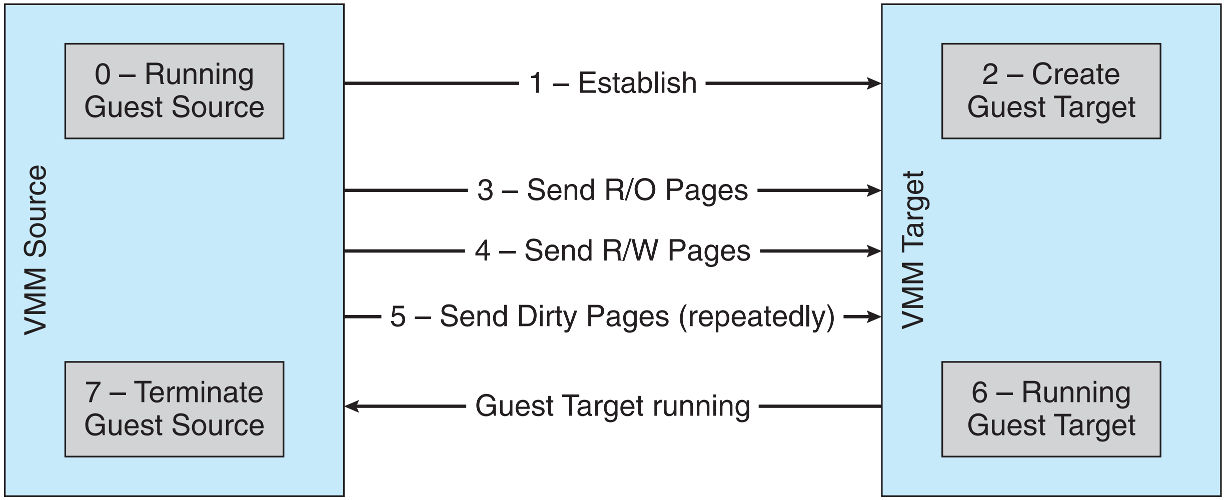
Live Migration

Running guest can be moved between systems, without interrupting user access to the guest or its apps

  - for resource management,
  - maintenance downtime windows, etc

- Migration from source VMM to target VMM
  - Needs to migrate all pages gradually, without interrupting execution (details in next slide)
  - Eventually source VMM freezes guest, sends vCPU’s final state, sends other state details, and tells target to start running the guest
  - Once target acknowledges that guest running, source terminates guest

Live Migration



- Migration from source VMM to target VMM
  - Source establishes a connection with the target
  - Target creates a new guest
  - Source sends all read-only memory pages to target
  - Source starts sending all read-write pages
  - Source VMM freezes guest, sends final stuff (VCPU's state etc)
  - Once target acknowledge that guest running, source terminates guest.

VIRTUAL APPLIANCES: “shrink-wrapped” virtual machines

- Developer can construct a virtual machine with
  - required OS, compiler, libraries, and application code
  - Freeze them as a unit … ready to run
- Customers get a complete working package
- Virtual appliances: “shrink-wrapped” virtual machines
- Amazon’s EC2 cloud offers many pre-packaged virtual appliances examples of *Software as a service*

- *Question: do we really have to include a whole kernel in a shrink wrapped VM?*

How to cite references

Purpose of including citation:

- Giving credit to the authors
- Mention the authority behind the claim

Unacceptable examples:

- ~~https://dl.acm.org/citation.cfm?id=224066~~
  - What is it about? Who wrote it? When? Why should I trust it?
- ~~Anderson et al, Serverless network file systems, ACM~~
  - ACM is not a journal or conference. When was this work done?
- T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In Proceedings of the 15th ACM Symp. on Operating System Principles, pages 109–126, December 1995
- Examples: IEEE template

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2021

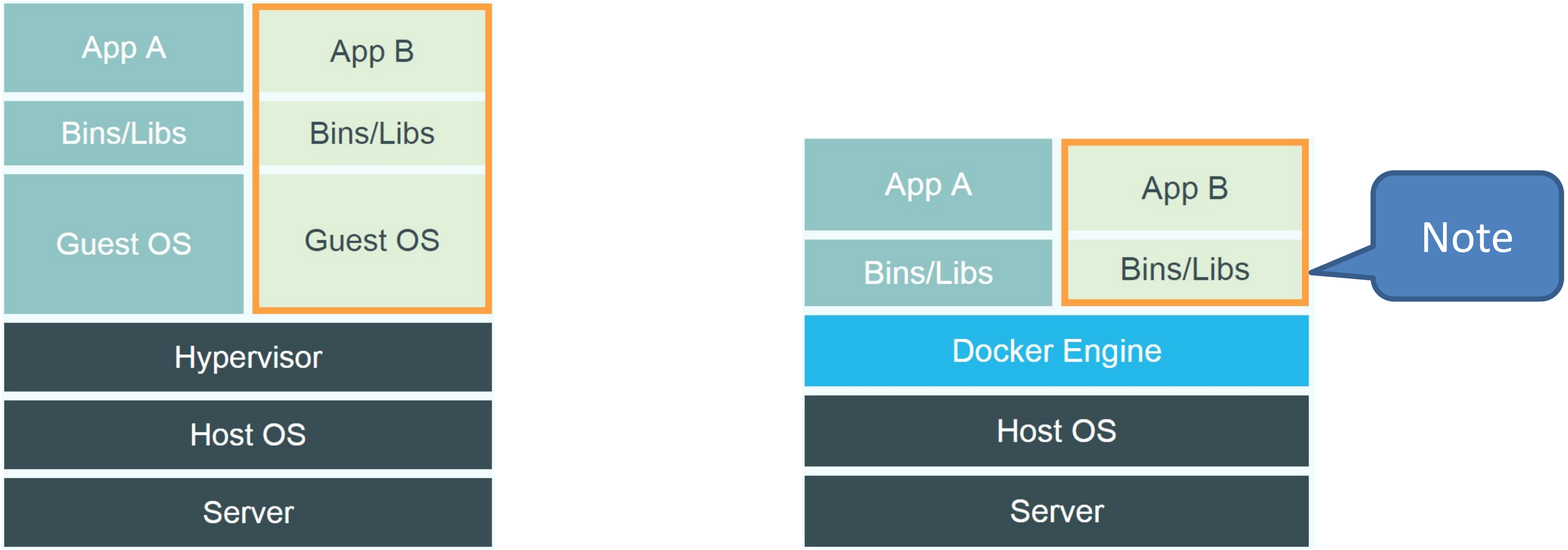
Containers

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Linux Containers and Docker

- Linux containers (LXC) are “lightweight” VMs
- Comparison between LXC/docker and VM



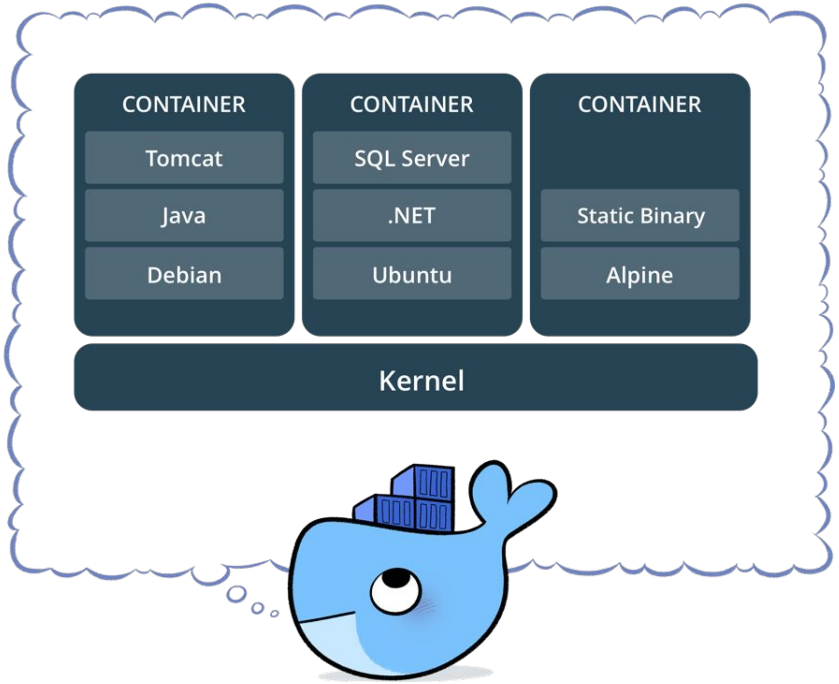
- Containers provide “OS-level Virtualization” vs “hardware level”.
- Containers can be deployed in seconds.
- Very little overhead during execution, just like Type 1.

Container: basis

Linux kernel provides

- "control groups" (cgroups) functionality for a set of processes
  - allows allocation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any VM
- "namespace isolation" functionality
  - allows complete isolation of an applications' view of the operating environment including Process trees, networking, user IDs and mounted file systems.
- Managed by Docker etc.
  - Docker: build, share, run and orchestrate containerized apps.
  - Kubernetes: orchestration platform for managing, automating, and scaling containerized applications

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Containers native to Windows Server 2016
- Docker: a popular container management service technology

Some Docker vocabulary

- **Docker Image**
  - The basis of a Docker container. Represents a full application
- **Docker Container**
  - The standard unit in which the application service resides and executes
- **Docker Engine**
  - Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider
- **Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))**
  - Cloud or server based storage and distribution service for images (can be **pull**ed or **push**ed)
- **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image using **docker build** command.

Correspondence: excecutable:image container:process

Some Docker vocabulary

- **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image using **docker build** command.
- Ex:

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Each instruction creates one layer:

- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client’s current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.

Docker Volumes

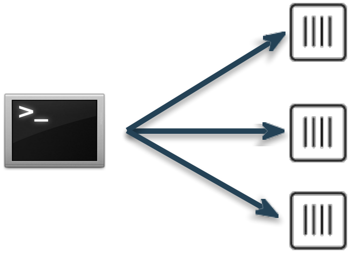
- Volumes mount a directory on the host into the container at a specific location

- Can be used to share (and persist) data between containers
  - Directory persists after the container is deleted
    - Unless you explicitly delete it

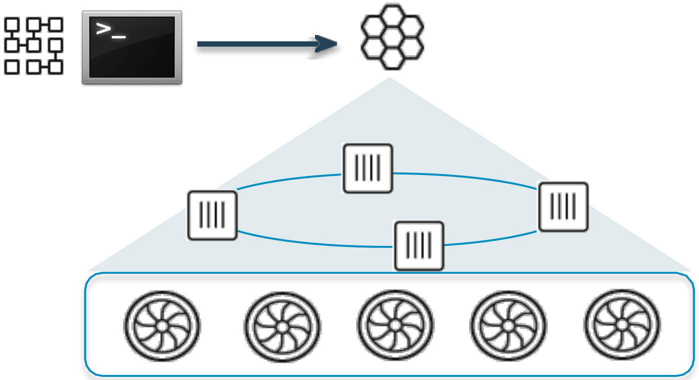
- Can be created in a Dockerfile or via CLI

Docker Compose: Multi Container Applications

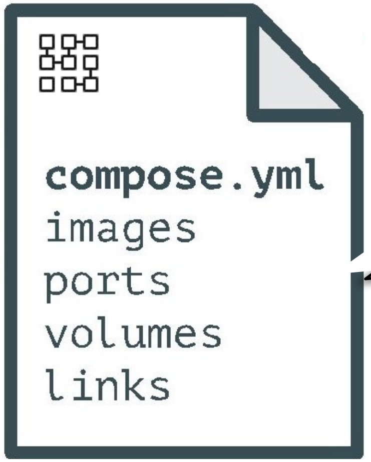
- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane



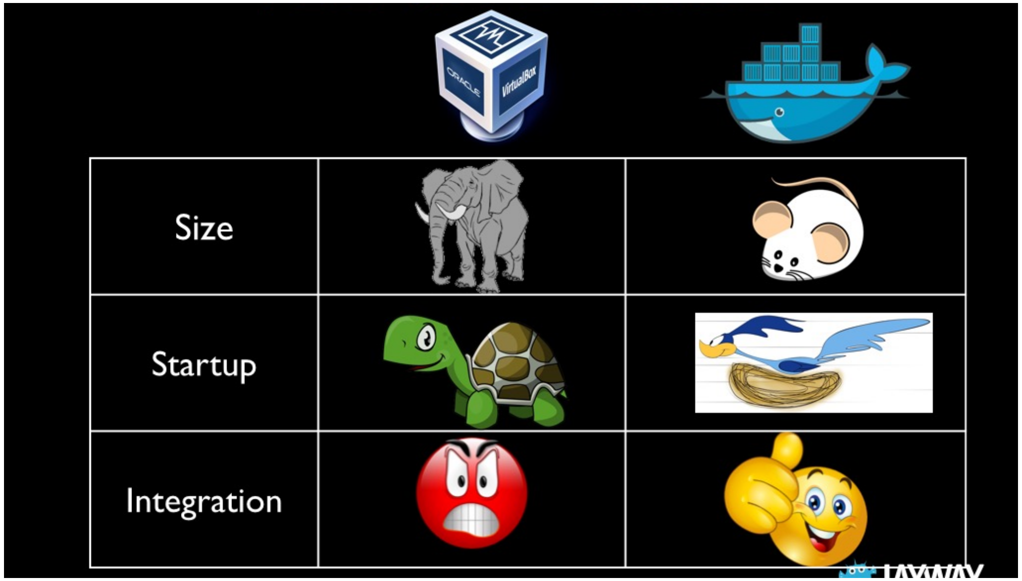
Docker Compose: Multi Container Applications



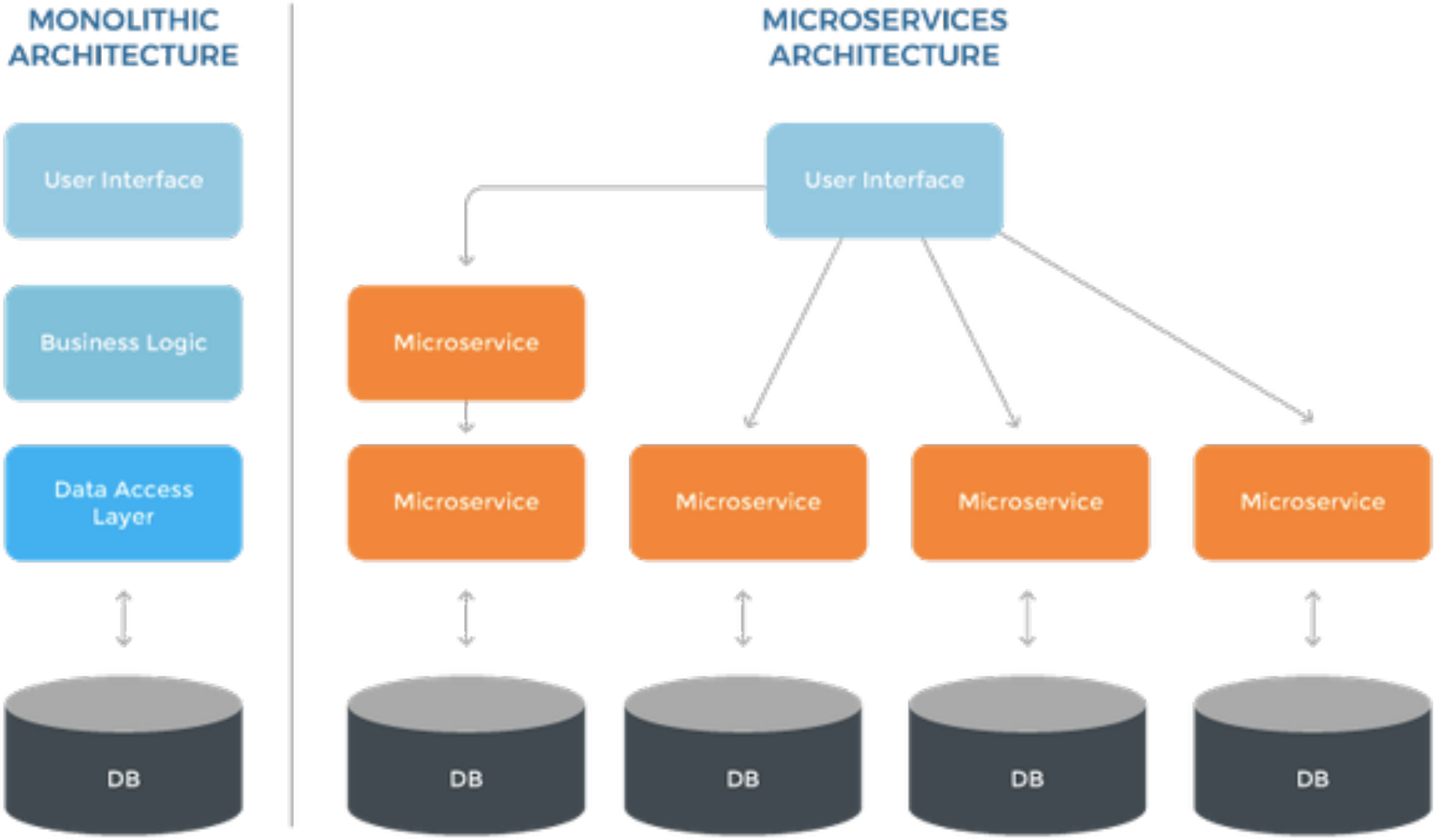
```
version: '2' # specify docker-compose version

# Define the services/containers to be run
services:
angular: # name of the first service
build: client # specify the directory of the Dockerfile
ports:
- "4200:4200" # specify port forewarding

express: #name of the second service
build: api # specify the directory of the Dockerfile
ports:
- "3977:3977" #specify ports forewarding

database: # name of the third service
image: mongo # specify image to build container from
ports:
- "27017:27017" # specify port forewarding
```

Unique features

- Containers run in the user space
- Each container has it own: process space, network interface, booting mechanism with configuration
- Share kernel with the host
- Can be packaged as Docker images to provide *microservices*.

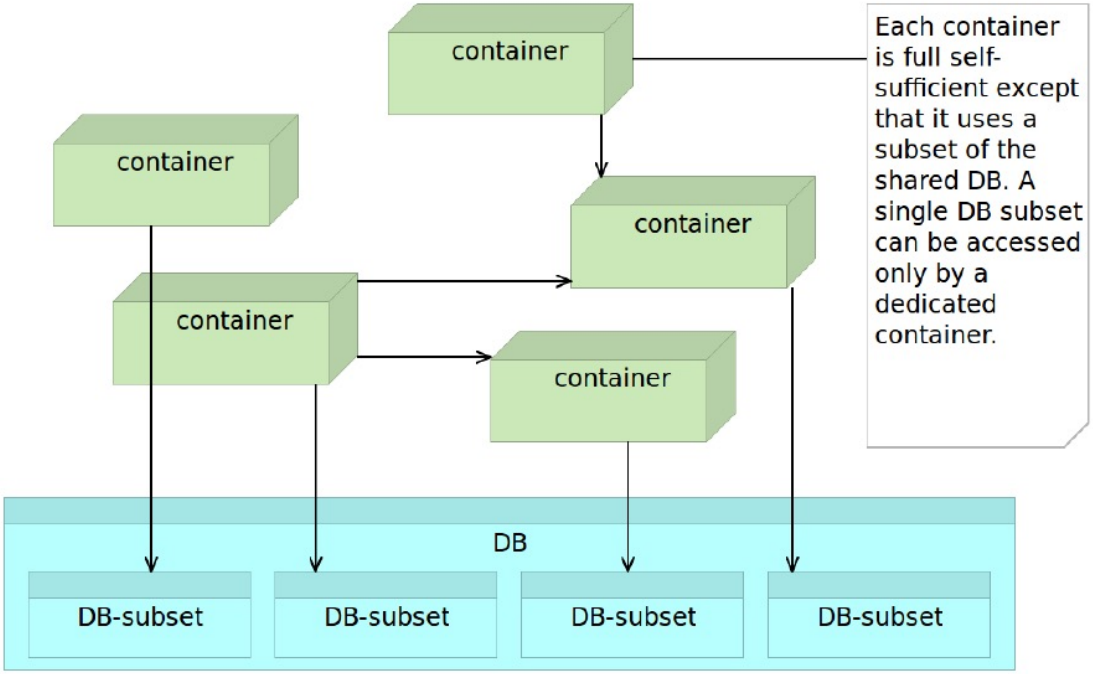


Monolithic architecture vs microservices



Colorado State University

Microservices Accessing the Shared Database

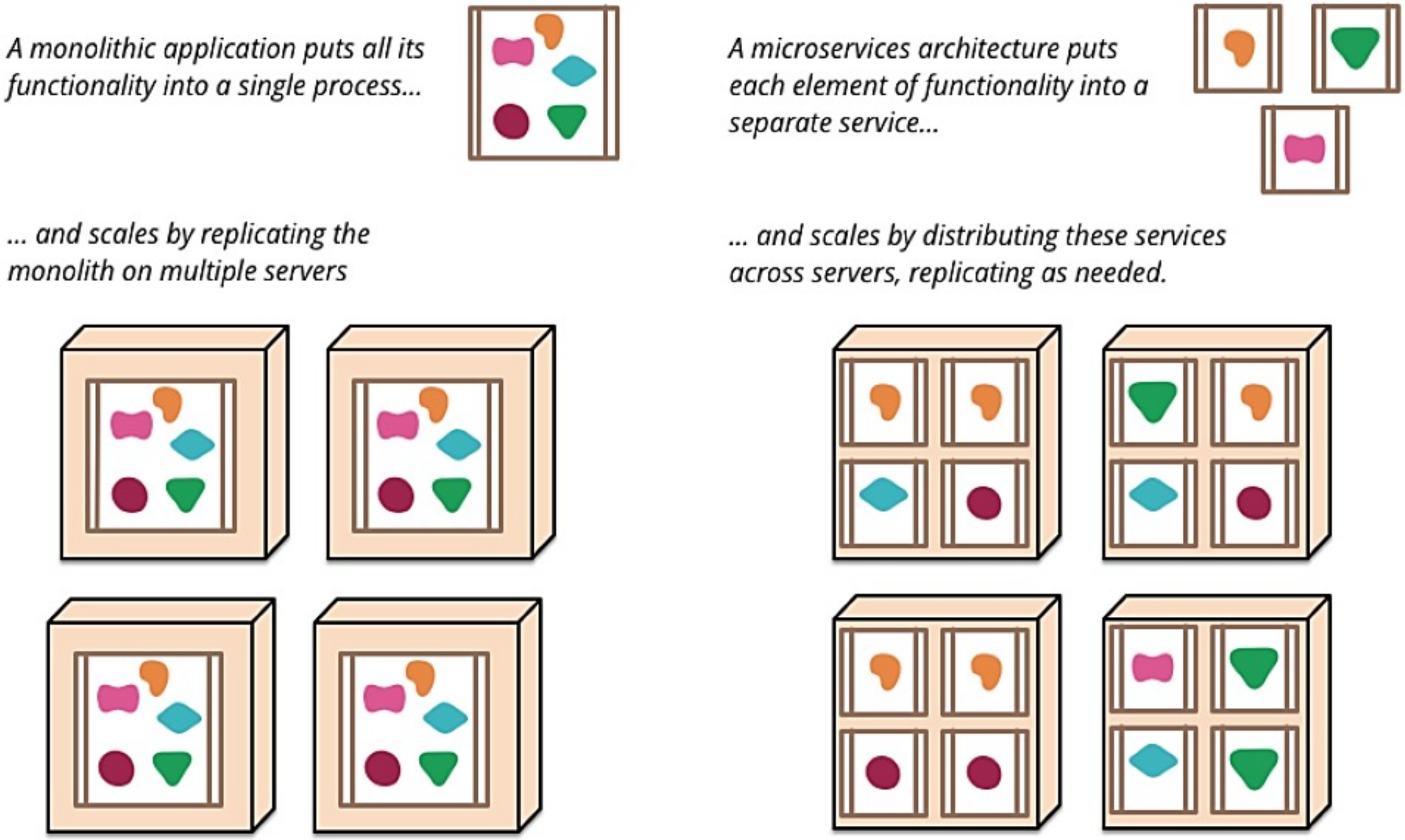


Each container is full self-sufficient except that it uses a subset of the shared DB. A single DB subset can be accessed only by a dedicated container.

Microservices Characteristics

- Many smaller (fine grained), clearly scoped services
  - Single Responsibility Principle
  - Independently Managed
- Clear ownership for each service
  - Typically need/adopt the “DevOps” model
- 100s of MicroServices
  - Need a Service Metadata Registry (Discovery Service)
- May be replicated as needed
- A microservice can be updated without interruption

Microservices. Scalability



A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2021

Data Centers & Cloud Computing

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Data Centers

- Large server and storage farms
  - 1000s-100,000 of servers
  - Many 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

Data Center architecture

- Traditional - static
  - Applications run on physical servers
  - System administrators monitor and manually manage servers
  - Storage Array Networks (SAN) or Network Attached Storage (NAS) to hold data
- Modern – dynamic with larger scale
  - Run applications inside virtual machines
  - Flexible mapping from virtual to physical resources
  - Increased automation, larger scale

Data Center architecture

Giant warehouses with:

- Racks of servers
- Storage arrays
- Cooling infrastructure
- Power converters
- Backup generators

Or with containers

- Each container filled with thousands of servers
- Can easily add new containers
- “Plug and play”
- Pre-assembled, cheaper, easily expanded

Server Virtualization

Allows a server to be "sliced" into Virtual Machines

  - VM has own OS/applications
  - Rapidly adjust resource allocations
  - VM migration within a LAN
- Virtual Servers
  - Consolidate servers
  - Faster deployment
  - Easier maintenance
- Virtual Desktops
  - Host employee desktops in VMs
  - Remote access with thin clients
  - Desktop is available anywhere
  - • Easier to manage and maintain

Data Center Challenges

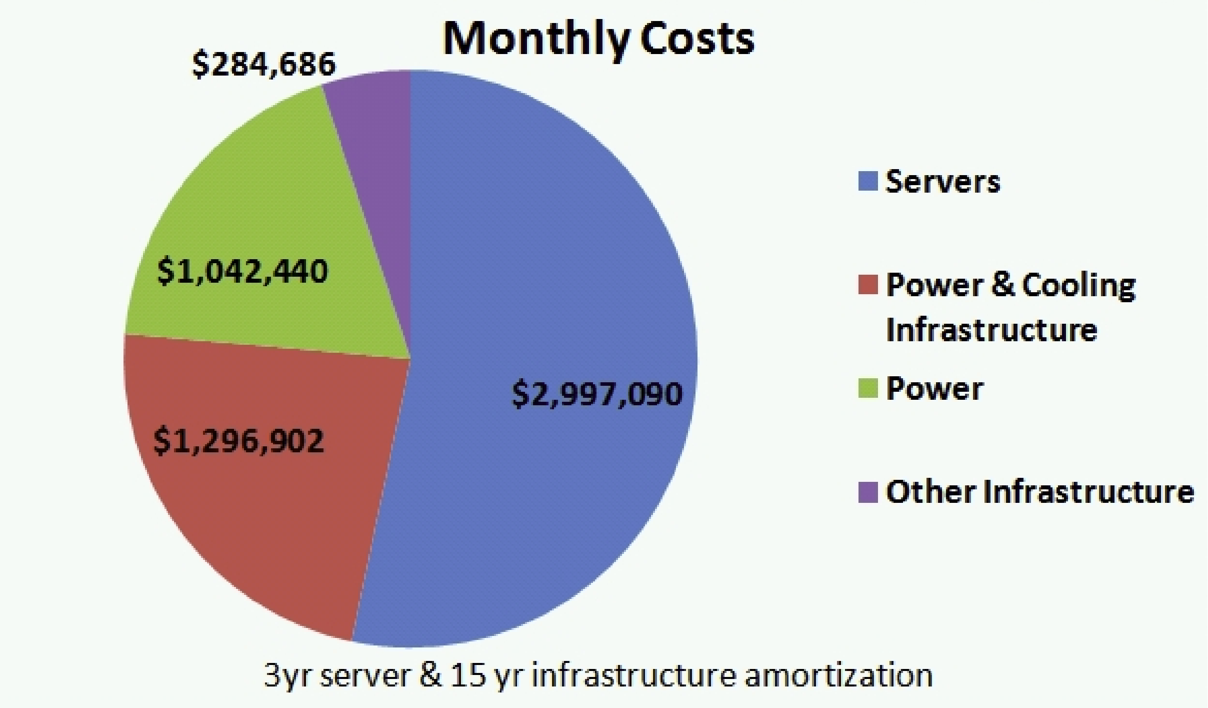
Resource management

- 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

Data Center Challenges



Power Efficiency measured as *Power Usage Effectiveness*

- *Power Usage Effectiveness* = Ratio of IT Power / Total Power
- typical: 1.7, Google PUE ~ 1.1)

http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx

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 small number of sys admins to manage thousands of servers
- General trend is towards larger mega data centers
- 100,000s of servers
- Has helped grow the popularity of cloud computing

Economy of Scale

Resource	Cost in Medium DC	Cost in Very Large DC	Ratio
CPU cycle cost	2 picocents	< 0.5 picocents	
Network	$95 / Mbps / month	$13 / Mbps / month	7.1x
Storage	$2.20 / GB / month	$0.40 / GB / month	5.7x
Administration	≈140 servers/admin	>1000 servers/admin	7.1x

Pico = 10^{-3} nano = 10^{-12}