Module 6: Virtualization & File System

Virtualization

[Ref 1. Hennessey Book on computer architecture

2. Case Study on VMware Virtual machines]

Virtual machines

- An idea related to virtual memory that is almost as old is Virtual Machines (VM)

- They were first developed in the late 1960s and they have remained important ever since

Why are VM's important in today's Computing?

- Increasing importance of isolation and security in modern systems
- Failures in security and reliability of standard operating systems
- Sharing of a single computer among many unrelated users
- Dramatic increases in raw speed of processors, which makes the overhead of VMs more acceptable

How can you emulate a VM?

- Standard software interfaces like JAVA VM

- VMs that provide complete system environment at the binary ISA (system Virtual Machines)→This is what we are interested in

- There are VM's available that run different ISAs

Example System Virtual Machines

- IBM VM/370, VMware ESX Server, and Xen are examples
- They present the illusion that the users of a VM have an entire computer to themselves, including a copy of the operating system
- A single computer runs multiple VMs and can support a number of different operating systems (OSes)

Virtual Machines

- Conventionally, a single OS owns all the hardware resources

- With VM, multiple OSes share the hardware resources

Hypervisor

- Software that supports VMs is called a virtual machine monitor (VMM) or hypervisor
- Underlying hardware platform → Host
- Its resources shared among → Guest VMs

Mapping Virtual Resources to Physical Resources

- Hypervisor determines the mapping

- A physical resource may be
  - Time Shared
  - Partitioned
  - Or even emulated in software

Continued…

- Hypervisor is much smaller than a OS
- Virtualization overhead is
  - Minimal for processor bound user programs
  - High for I/O intensive workloads

Functionalities of Hypervisor

- Managing Software

- Managing Hardware

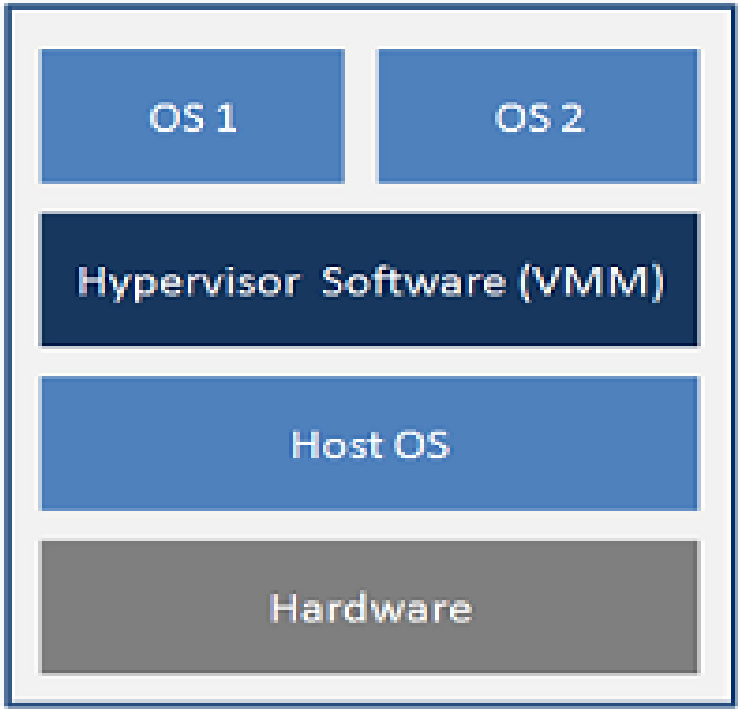
Continued…

- Should control
  - Access to privileged states
  - Address translation
  - I/O
  - Exceptions and Interrupts

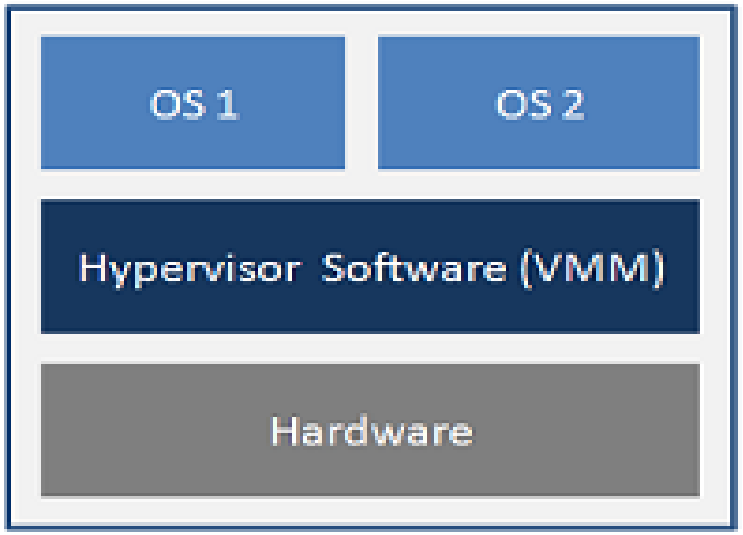
Hypervisor and protection

- Similar to paging
  - Two processor modes → System and User
  - A privileged subset of instructions that is available only in system mode
    - Resulting in Trap if executed in user mode
    - All system resources only controllable via these instructions
- Ensures a guest OS talks to virtual resources
  - VM runs as a user mode program with virtual resources → Can be provided by ISA
    - If not supported by ISA, hypervisor should take care of the above

Types of Hypervisor



Hosted Architecture



Bare-Metal Architecture

Impact of VM on virtual memory and I/O

- How to virtualize virtual memory?
- How will each guest OS manages its page table?

Mapping Virtual → Real → Physical Memory

- What is Real Memory?
  - Intermediate level between virtual and physical memory
    - In other words it is the memory allocated per machine
    - Some refer this as Machine memory
- Guest OS maps virtual memory to real memory via page tables
- Hypervisor Page Tables map the guest's real memory to physical memory

Shadow Page Table

- To reduce extra level of indirection on every memory access, shadow page table is used

- It consists of guest virtual address to physical address space
  - Hypervisor ensure the shadow page table entries used correspond to those of the guest OS environment
  - If the guest OS tries to change the page table, a trap will be generated and the hypervisor will ensure write protection to the guest page table

Virtualizing TLB

- Manages the real TLB and has a copy of the contents of the TLB of each guest VM
- Instructions to access TLB must trap
- TLB must support mix of entries from different VMs to avoid flushing of TLB during VM switch

Virtualize I/O

- The VM illusion can be maintained by giving each VM generic versions of each type of I/O device driver, and then leaving it to the VMM to handle real I/O
- Mapping a virtual to physical I/O device depends on the type of device
- For example, physical disks are normally partitioned by the VMM to create virtual disks for guest VMs, and the VMM maintains the mapping of virtual tracks and sectors to the physical ones

Case Study : The Xen VM

- *Para-virtualization* : Allowing small modifications to the guest OS to simplify virtualization

- The Xen VMM provides a guest OS with a virtual machine abstraction that is similar to the physical hardware, but it drops many of the troublesome pieces

Case Study : VMware

- Terminology
  - ***Host physical memory*** refers to the memory that is visible to the hypervisor as available on the system
  - ***Guest physical memory*** refers to the memory that is visible to the guest operating system running in the virtual machine
  - ***Guest virtual memory*** refers to a continuous virtual address space presented by the guest operating system to applications

Continued…

- ***Guest level paging*** The memory transfer between the guest physical memory and the guest swap device is referred
  - Driven by the guest operating system
- ***Hypervisor swapping*** The memory transfer between guest physical memory and the host swap device
  - Driven by hypervisor

Continued…

- When running a virtual machine, the hypervisor creates a contiguous addressable memory space for the virtual machine
  - Same properties as virtual address space
- This allows the hypervisor to run multiple virtual machines simultaneously while protecting the memory of each virtual machine from being accessed by others

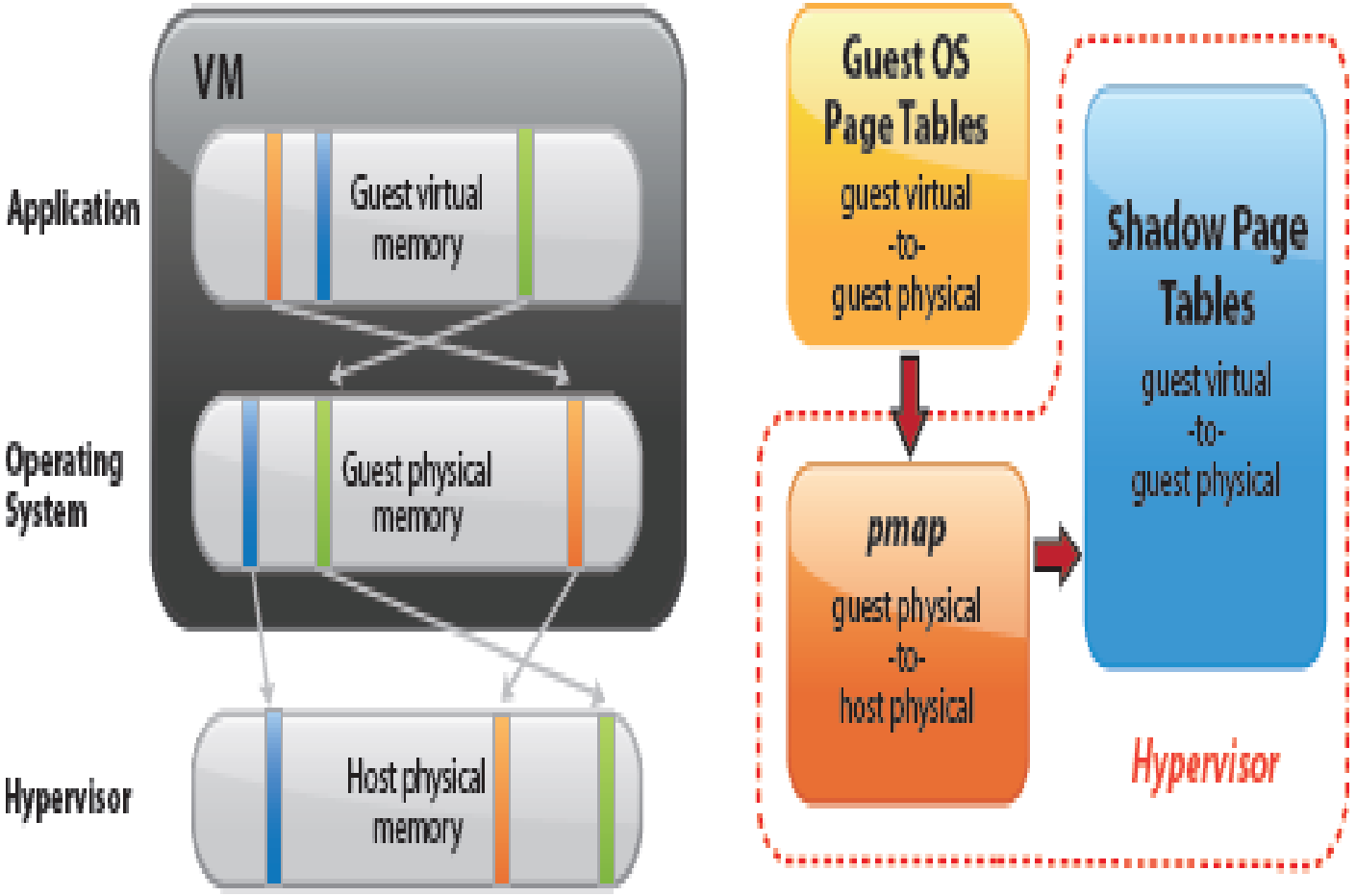
Continued…

- From the view of the application running inside the virtual machine, the hypervisor adds an extra level of address translation that maps the guest physical address to the host physical address

Three Virtual Memory Layers

- Guest virtual memory
- Guest physical memory
- Host physical memory

Continued…


VM
Application
Guest virtual memory
Operating System
Guest physical memory
Hypervisor
Host physical memory
Guest OS Page Tables
guest virtual -to- guest physical
pmap
guest physical -to- host physical
Shadow Page Tables
guest virtual -to- guest physical
Hypervisor

Continued..

- *pmap*
  - Address translation between guest physical memory and host physical memory

- Hypervisor intercepts all virtual machine instructions that manipulate TLB or guest OS page tables which contain virtual address to physical address mapping

Continued…

- Shadow Page Table
  - In hypervisor
  - Contain the guest virtual to host physical address mapping

Virtual Machine memory allocation

- Like an application, when a virtual machine first starts, it has no pre-allocated physical memory
- Like an operating system, the virtual machine cannot explicitly allocate host physical memory through any standard interfaces
- The hypervisor also creates the definitions of “allocated” and “free” host memory in its own data structures
- The hypervisor intercepts the virtual machine’s memory accesses and allocates host physical memory for the virtual machine on its first access to the memory

Continued…

- In order to avoid information leaking among virtual machines, the hypervisor always writes zeroes to the host physical memory before assigning it to a virtual machine

- Virtual machine memory de-allocation acts just like an operating system, such that the guest operating system frees a piece of physical memory by adding these memory page numbers to the guest free list
  - But the data is not cleared

Continued…

- It is difficult for the hypervisor to know when to free host physical memory upon virtual machine memory de-allocation
  - The guest operating system free list is generally not publicly accessible
- The hypervisor just allocates host physical memory for the first memory allocation and then the guest reuses the same host physical memory for the rest of allocations

Continued…

- If a virtual machine's entire guest memory has been backed by the host physical memory, the hypervisor does not need to allocate any host physical memory for this virtual machine any more

VM's host memory usage <= VM's guest memory size + VM's overhead memory

How can hypervisor Reclaim Physical Memory allocated to host?

- If hypervisor cannot reclaim physical memory from VMs, it must reserve enough physical memory to back all virtual machine's guest physical memory (plus their overhead memory) in order to prevent any virtual machine from running out of host physical memory
  - So can't there be memory over-commitment?

Memory Over-commitment

- Host memory is overcommitted when the total amount of guest physical memory (allocated) of the running virtual machines is larger than the amount of actual host memory

- VMWare ESX supports memory overcommitment

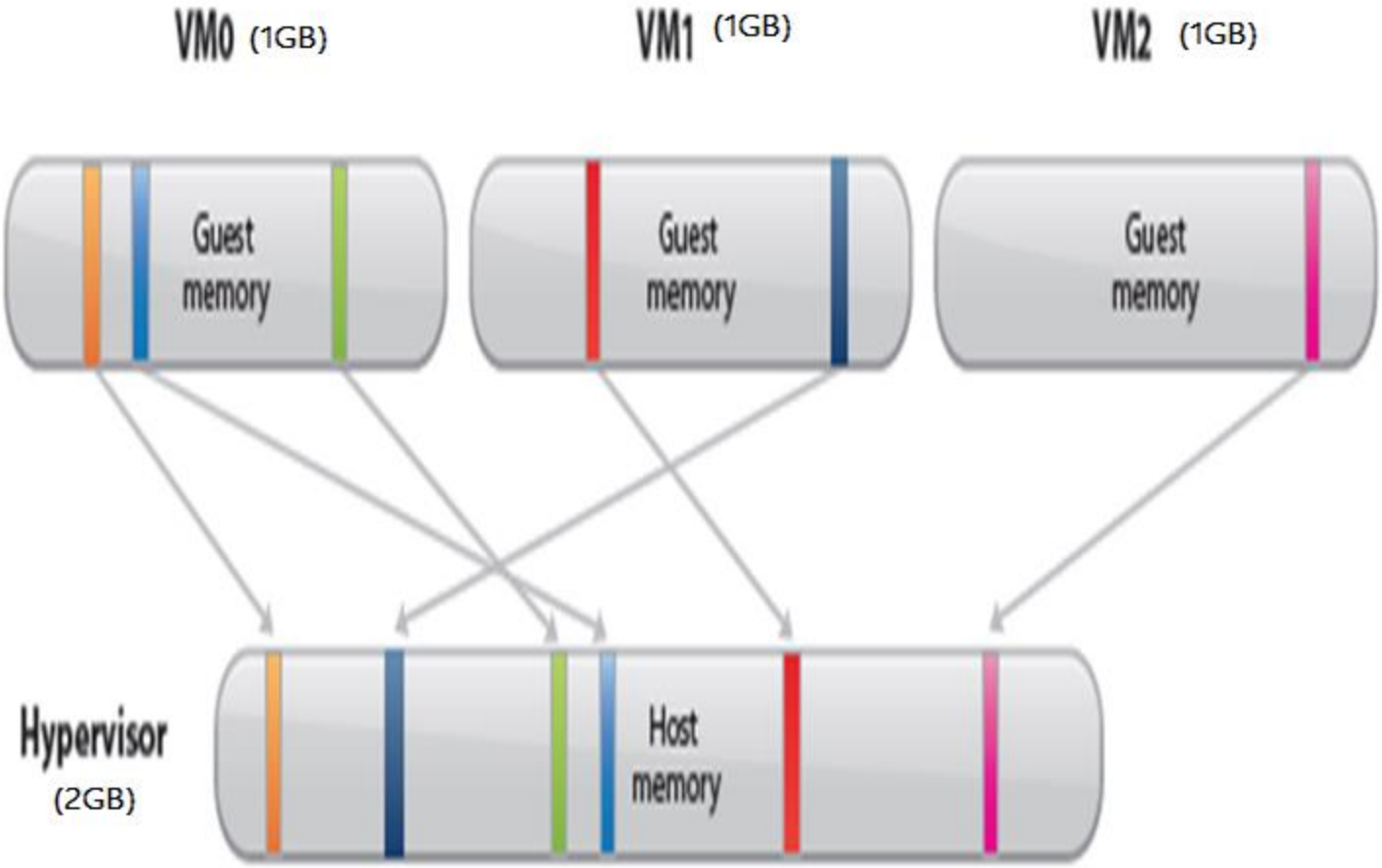
Benefits of Memory Over-commitment

- Higher memory utilization
  - Host memory is consumed by active guest memory as much as possible
- Higher consolidation ratio
  - With memory over-commitment, each virtual machine has a smaller footprint in host memory usage, making it possible to fit more virtual machines on the host while still achieving good performance for all virtual machines

How Over-commitment Supported?

- Through efficient memory reclamation
  - Transparent Page Sharing
    - Share pages between VMs if they have identical content
  - Ballooning (most popular and this alone will be dealt in this course in detail)
  - Host Swapping

Example of Memory Over-commitment



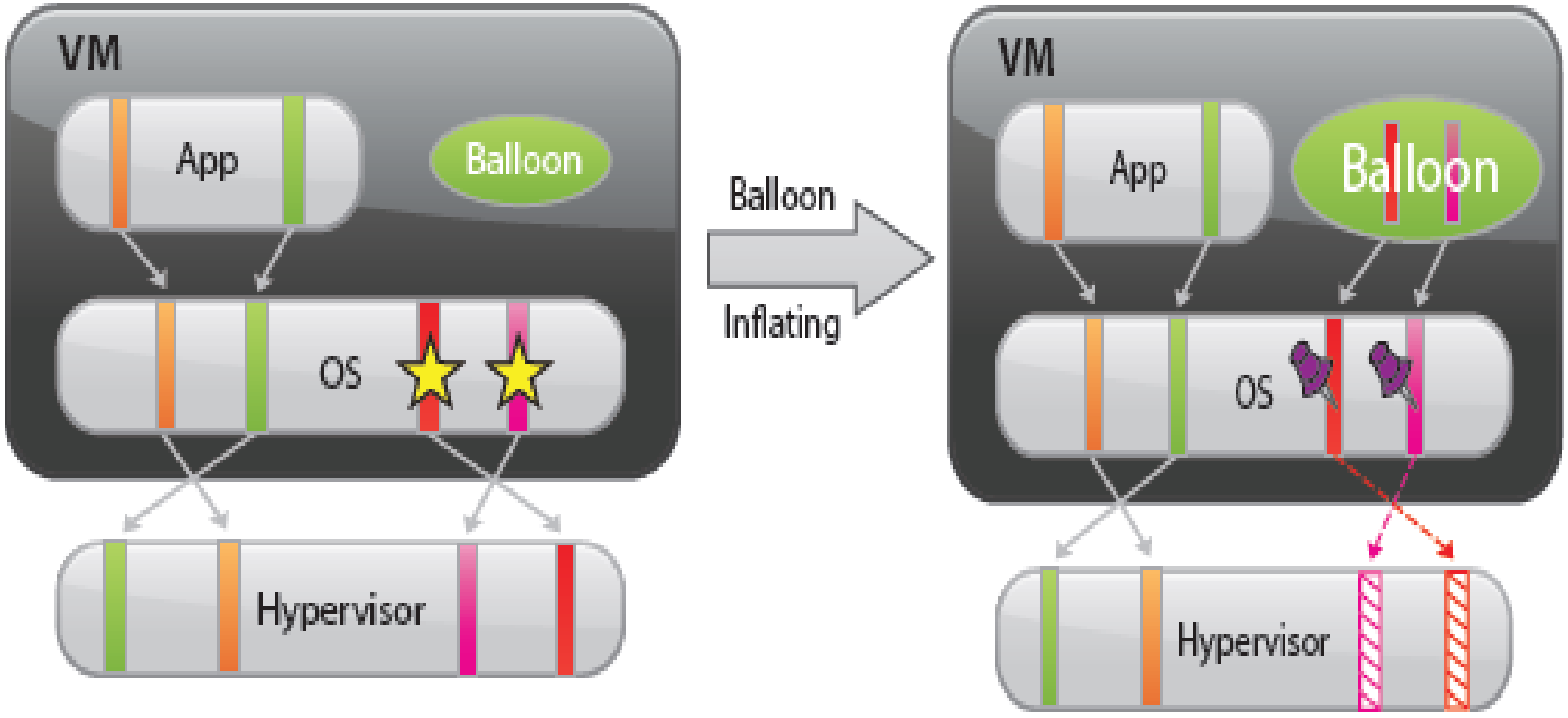
Ballooning

- 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 as a pseudo-device driver
- It has no external interfaces to the guest operating system and communicates with the hypervisor through a private channel

Continued…

- The balloon driver polls the hypervisor to obtain a target balloon size
- If the hypervisor needs to reclaim virtual machine memory, it sets a proper target balloon size for the balloon driver
- This will “inflate” by allocating guest physical pages within the virtual machine

Example


VM
App
Balloon
OS
Hypervisor
Balloon
Inflating
VM
App
Balloon
OS
Hypervisor

Continued…

- Four guest physical pages are mapped in the host physical memory
- Two of the pages are used by the guest application
- Other two pages (marked by stars) are in the guest operating system free list

Continued…

- Since the hypervisor cannot identify the two pages in the guest free list, it cannot reclaim the host physical pages that are backing them
- Assuming the hypervisor needs to reclaim two pages from the virtual machine, it will set the target balloon size to two pages
- After obtaining the target balloon size, the balloon driver allocates two guest physical pages inside the virtual machine and pins them

Continued…

- “pinning” is achieved through the guest operating system interface
  - Ensures that the pinned pages cannot be paged out to disk under any circumstances
- Balloon driver notifies the hypervisor the page numbers of the pinned guest physical memory
- Hypervisor can reclaim the host physical pages that are backing them
- When the hypervisor decides to deflate the balloon — by setting a smaller target balloon size — the balloon driver de-allocates the pinned guest physical memory, which releases it for the guest’s applications

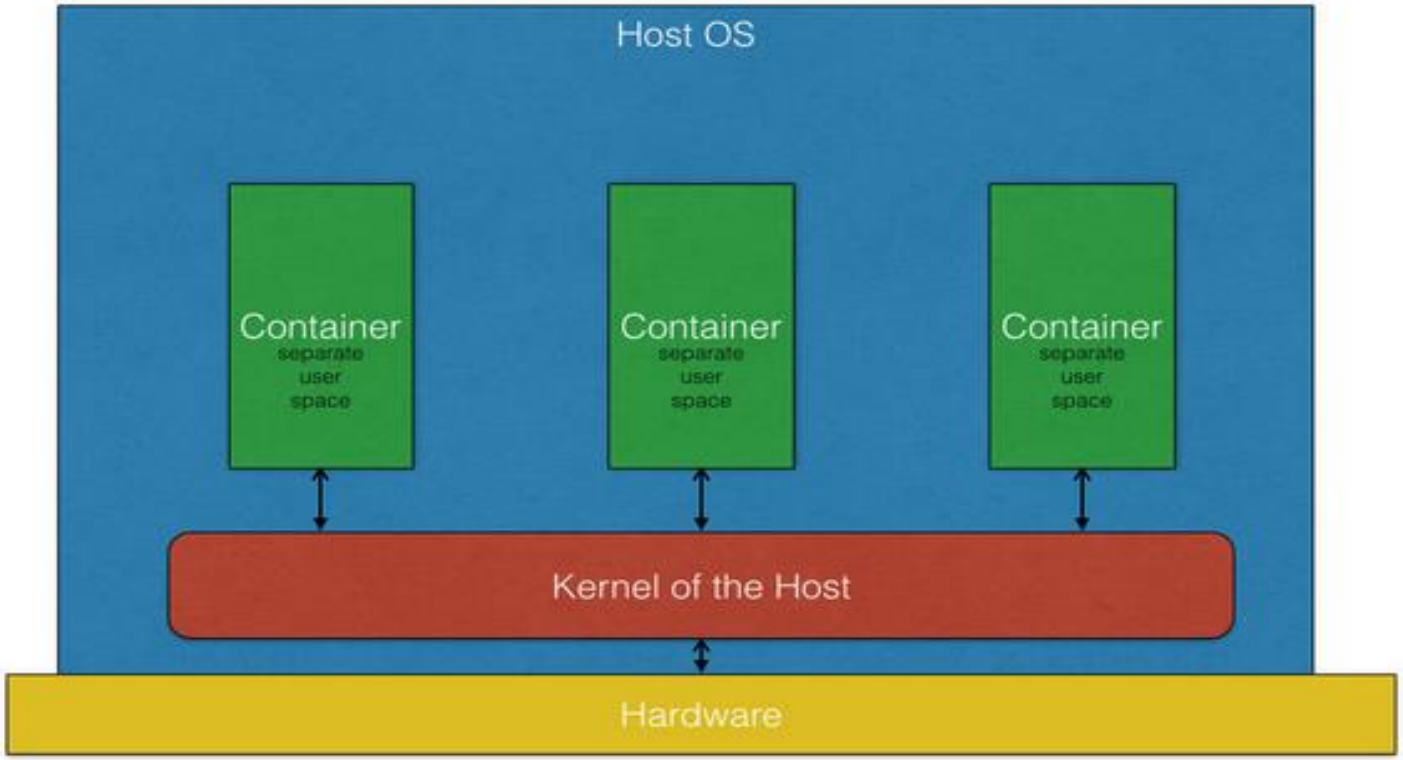
Hypervisor Swapping

- As a last effort to manage excessively overcommitted physical memory, the hypervisor will swap the virtual machine's memory
- When starting a virtual machine, the hypervisor creates a separate swap file for the virtual machine
- The hypervisor can directly swap out guest physical memory to the swap file, which frees host physical memory for other virtual machines

Containers Versus Virtual Machines

What are Containers?

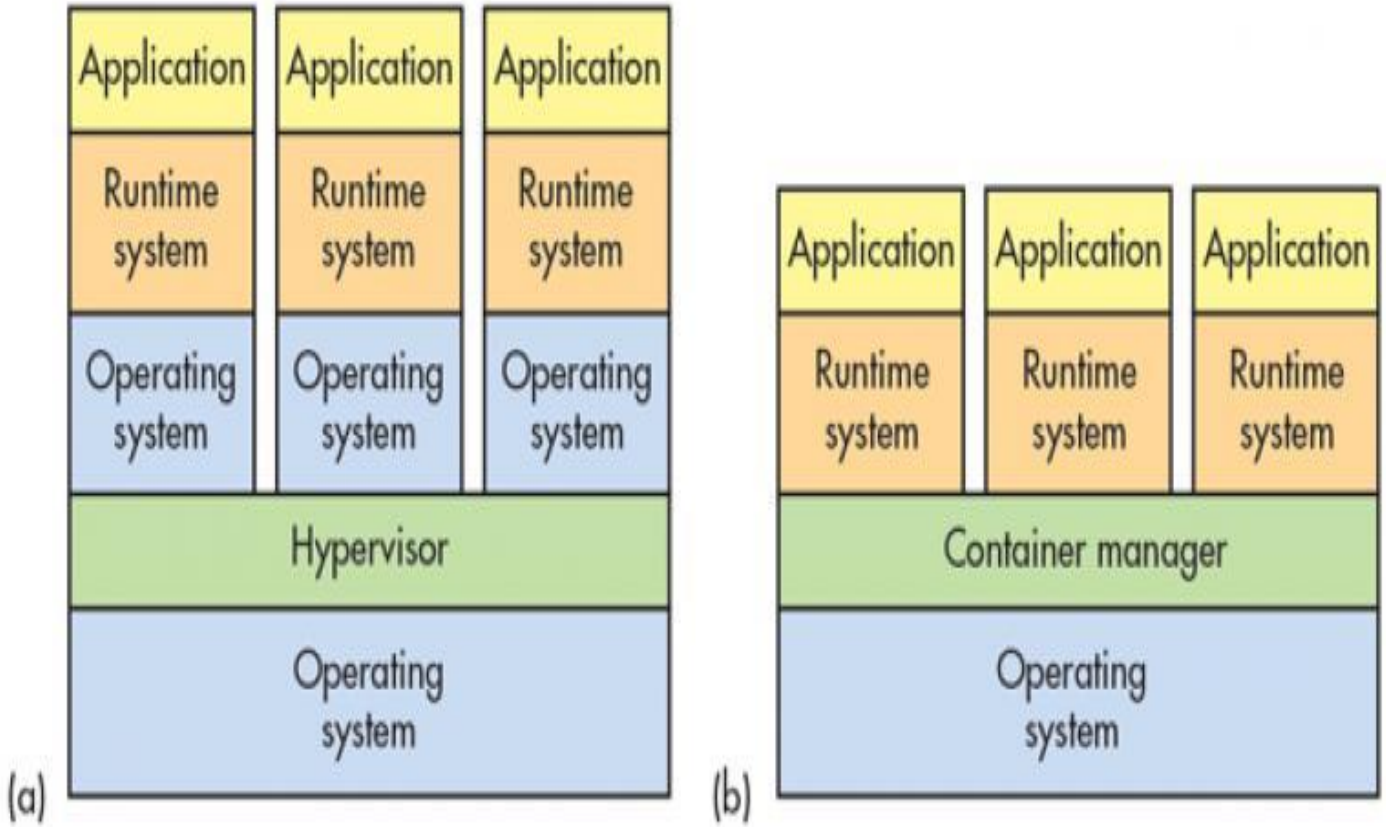
- Containers are the products of operating system virtualization



Operating System/Container Virtualization

Difference between VMs and Containers

- Containers → OS Level Virtualization
- Hypervisor Provides Hardware Level Virtualization
- Containers share the same kernel of the host system
- Type of containers that can be installed on the host should work with the kernel of the host
  - Cannot install a Windows container on a Linux host or vice-versa
- VMs are better with security compared to containers



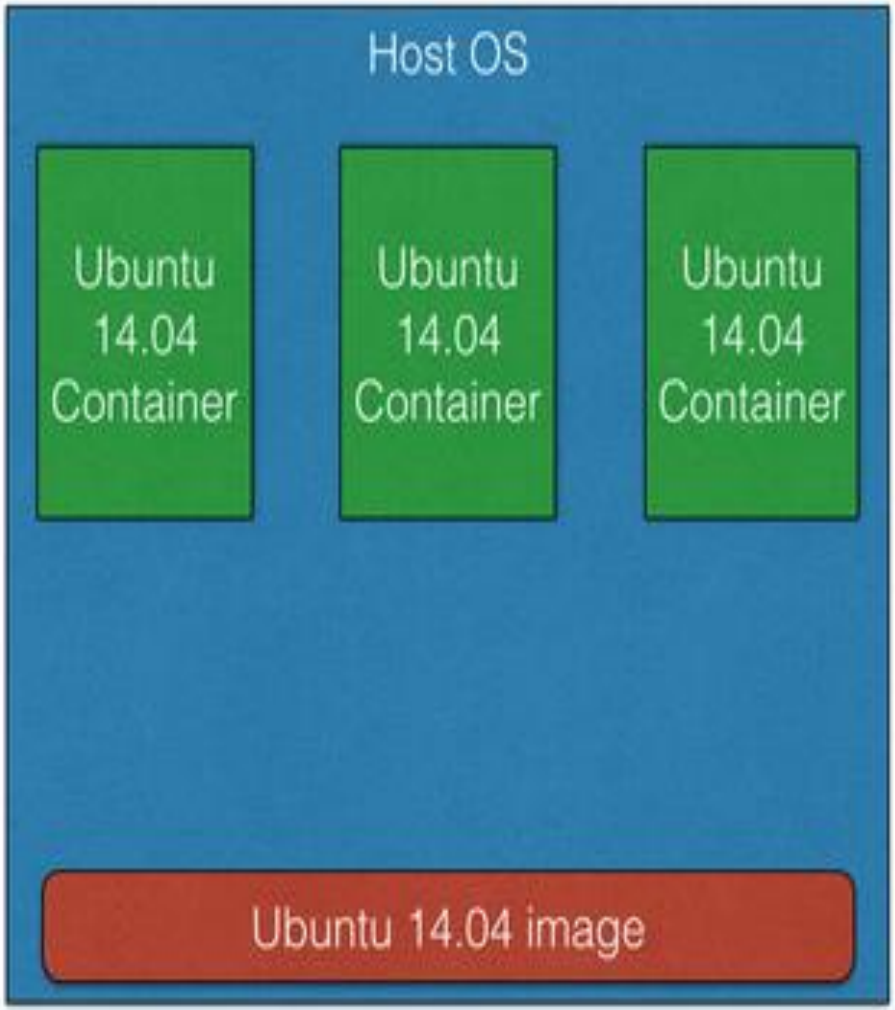
Virtual Machine

Container

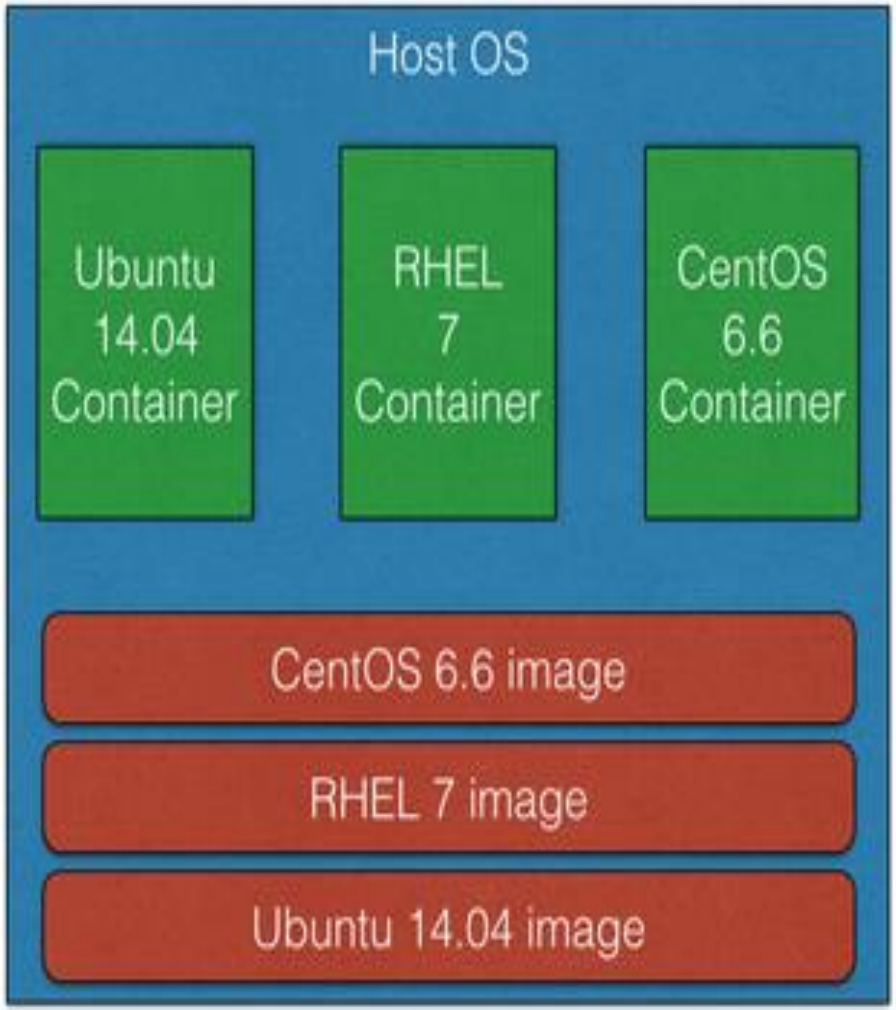
When to use Containers?

- OS Containers
  - share the kernel of the host
  - provide user space isolation
  - Used to run a fleet of identical or different flavours of OS
- Application containers
  - Designed to package and run a single service
  - Eg: Docker and Rocket

OS Container



Identical OS containers



Different flavoured OS containers

Cost benefits of Containers

- Less need to reproduce OS code
  - Pack more on server with container than the server running hypervisor
- Example
  - Server to host 10 applications in 10 VMs
    - 10 copies of the OS running in each VM
    - In container based model one OS with 10 applications

Container Vs. VMs

Feature	Container	Virtual machine
Operating system	Shares the host operating system's kernel	Has its own kernel
Portability	More portable	Less portable
Speed	Faster to start up and shut down	Slower to start up and shut down
Resource usage	Uses fewer resources	Uses more resources
Use cases	Good for portable and scalable applications	Good for isolated applications

Will Container Technology replace VMs?

- Cost wise container are relatively better

- But VMs are still important for the following features
  - Security
  - Tooling
  - Management and Process

Topics to be Discussed

- How OS enables CPU and I/O communication?

- File Systems

How Linux Organizes Data?

- Linux receives data from, sends data to, and stores data on *devices*

- A device usually corresponds to a hardware unit, such as a keyboard or serial port
  - The kernel creates several *pseudodevices* that you can access as devices but that have no physical existence

Sample Typical Linux Devices

Device	Description
atibm	Bus mouse
audio	Sound card
cdrom	CD-ROM drive
console	Current virtual console
fd *n*	Floppy drive (*n* designates the drive; for example, fd0 is the first floppy drive)
ftape	Streaming tape drive not supporting rewind
hd *xn*	Non-SCSI hard drive (*x* designates the drive and *n* designates the partition; for example, hda1 is the first partition of the first non-SCSI hard drive)
inportbm	Bus mouse
lp *n*	Parallel port (*n* designates the device number; for example, lp0 is the first parallel port)
modem	Modem
mouse	Mouse

File systems

- Whether you're using Microsoft Windows or Linux, you must format a partition before you can store data on it

- When you format a partition, Linux writes special data, called a *filesystem*, on the partition
  - One of the goals of having different partitions is to achieve higher data security in case of disaster. By dividing the hard disk in partitions, data can be grouped and separated
  - The filesystem organizes the available space and provides a directory that lets you assign a name to each *file*, which is a set of stored data.

Continued…

- The use of partitions remains for security and robustness reasons, so a breach on one part of the system doesn't automatically mean that the whole computer is in danger

Partition layout and types

- Two kinds of major partitions on a Linux system

- *data partition*
  - normal Linux system data, including the *root partition* containing all the data to start up and run the system
- *swap partition*
  - Expansion of the computer's physical memory, extra memory on hard disk

Continued…

- Most Linux systems use **fdisk** at installation time to set the partition type
- The standard Linux partitions have number 82 for swap and 83 for data
- Apart from these two, Linux supports a variety of other file system types,
  - JFS, NFS, FATxx

Continued…

- The standard root partition (indicated with a single forward slash, */*) is about 100-500 MB
  - contains the system configuration files, most basic commands and server programs, system libraries, some temporary space and the home directory of the administrative user
- Swap space (indicated with *swap*) is only accessible for the system itself
  - hidden from view during normal operation
  - Swap is the system that ensures, like on normal UNIX systems, that you can keep on working, whatever happens

Mount Point

- All partitions are attached to the system via a mount point.
- The mount point defines the place of a particular data set in the file system
- Usually, all partitions are connected through the *root* partition
- On this partition, which is indicated with the slash (/), directories are created

The file system in reality

- For most users and for most common system administration tasks, it is enough to accept that files and directories are ordered in a tree-like structure

- The computer, however, doesn't understand a thing about trees or tree-structures

Continued…

- Every partition has its own file system
- In a file system, a file is represented by an *inode*, a kind of serial number containing information about the actual data that makes up the file
  - to whom this file belongs, and where is it located on the hard disk

File System Implementation

- A boot control block or Boot Block
  - Boot an OS from that volume
  - No OS → this block can be empty
  - In NTFS → Partition boot sector
- Super Block
  - Meta data about the data blocks
- Directory structure
  - Used to organize the files
- Data blocks

Directory Implementation

- Linear List
  - Linear list of file names with pointer to data blocks
  - Time consuming
- Hash Table
  - Hash table takes values from file name and returns the pointer to the file in the list
  - Disadvantages
    - Collision
    - Fixed Size

How to allocate disk space to files effectively?

- Contiguous Allocation
  - File occupy set of contiguous blocks
  - Linear ordering of addresses
  - Less *disk seeks*
  - Disadvantages
    - Finding space for a new file
    - External Fragmentation
  - Extents

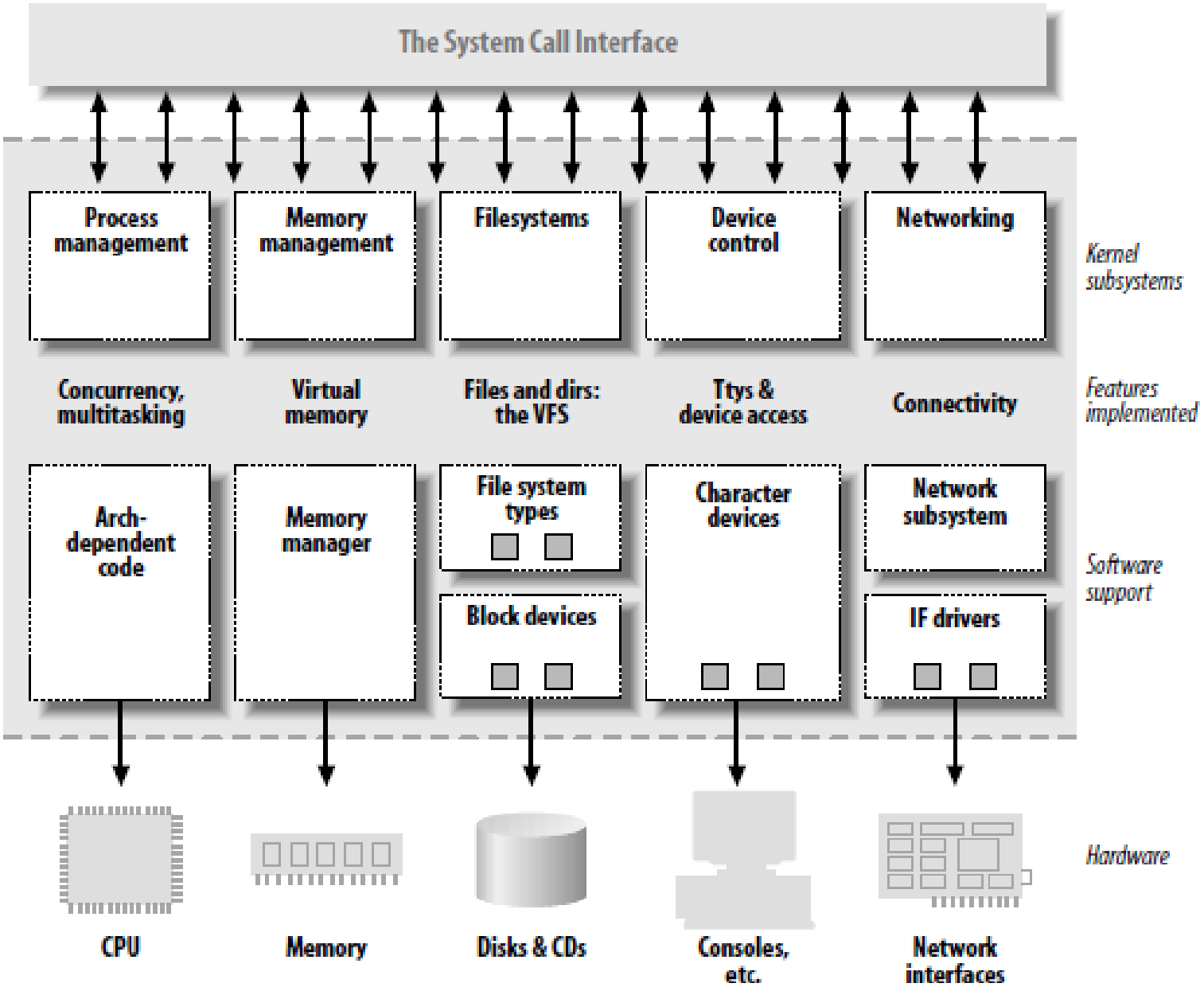
Continued…

- Linked Allocation
  - Each file is a linked list of disk blocks
  - The disk blocks can be anywhere on the disk
  - Size of the file can change after creation
  - Disadvantage:
    - Sequential access of file
    - Space for pointers
      - » Allocate clusters

Continued…

- Indexed Allocation
  - All the pointers are brought into *Index Block*
- Combined (Linked + Indexed)

Virtual File System (VFS)


The System Call Interface
Process management
Memory management
Filesystems
Device control
Networking
Kernel subsystems
Concurrency, multitasking
Virtual memory
Files and dirs: the VFS
Ttys & device access
Connectivity
Features implemented
Arch-dependent code
Memory manager
File system types
Block devices
Character devices
Network subsystem
IF drivers
Software support
CPU
Memory
Disks & CDs
Consoles, etc.
Network interfaces
Hardware
features implemented as modules

What is File Allocation Table?

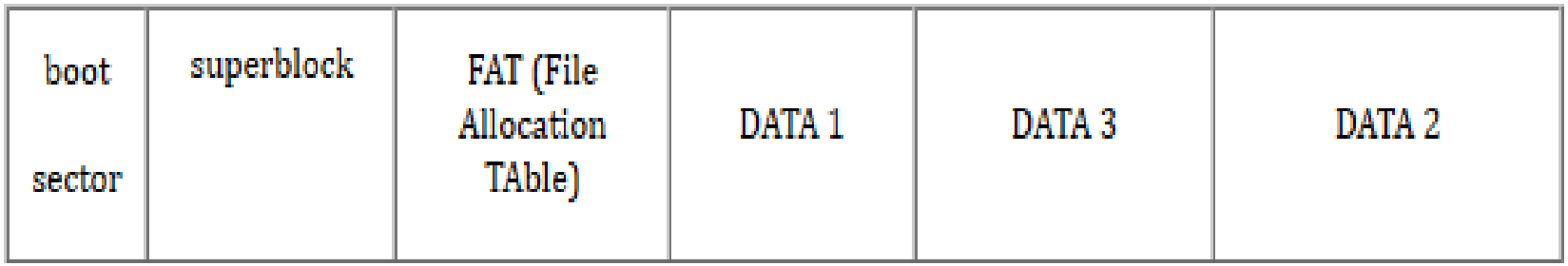
- Computer file system architecture
- FAT12, FAT16 and FAT32
  - As disk drives evolved, the capabilities of the file system have been extended accordingly
- Original Version FAT with 8 bit table

Continued…

- A file allocation table (FAT) is a table that an operating system maintains on a hard disk that provides a map of the clusters (the basic units of logical storage on a hard disk) that a file has been stored in

- Resides at the beginning of the volume

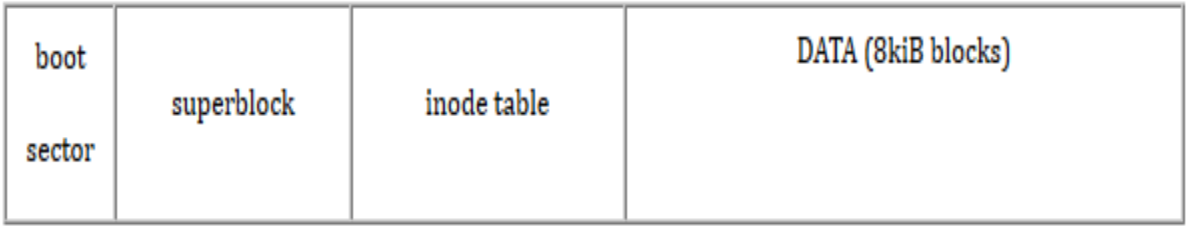
- The file allocation tables and the root folder must be stored in a fixed location so that the files needed to start the system can be correctly located



- Boot sector contains the bootstrapping programs to start the system upon a boot
- Superblock contains all the metadata about the file system, such as the version number, the size of the entire system, the root directory, the pointer to the first data block, and more
- The file allocation table (FAT) contains the FAT structure, which is a map of the data region

Continued….

- The FAT table is an array of entries where each entry represents the corresponding data block
- Each entry either contains:
  - the number of the next block that is part of the file
  - 0 which means the current block is the end of the file
  - -1, which means the block is free



- **Boot Sector**: like in FAT, contains information for booting and loading the OS

- **Super Block**: like in FAT, holds metadata about the file system. This includes data like: version number, size, amount of free space, and location of rootdir

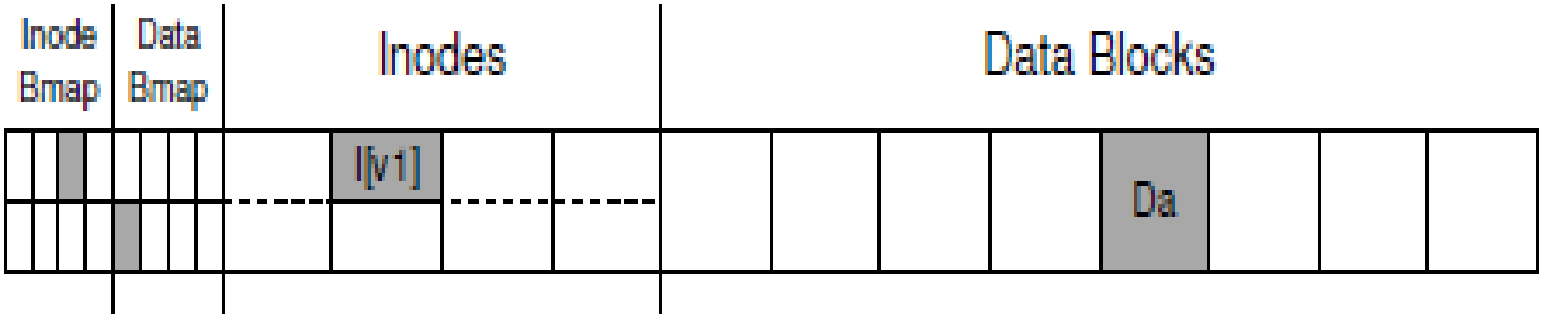
Continued…

- **Bitmap Block**: Used to locate free and used blocks

- Free blocks = 0 and used blocks = 1

Crash Consistency: FSCK and Journaling

- Due to power failure or system crash → data is inconsistent in the disk (while writing)



I[V1] Content

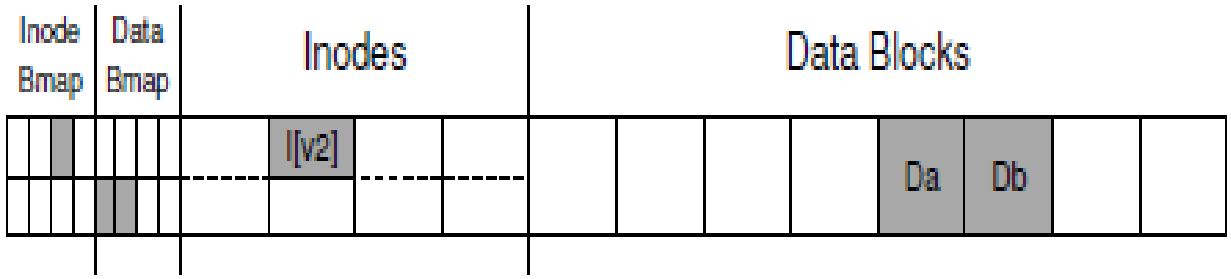
```
owner       : remzi
permissions : read-only
size        : 1
pointer     : 4
pointer     : null
pointer     : null
pointer     : null
```

- Assume we are appending data to the file

Updated I[V2]

```
owner       : remzi
permissions : read-only
size        : 2
pointer     : 4
pointer     : 5
pointer     : null
pointer     : null
```

Final Disk image expected



Crash Scenarios

- Just the data block (Db) is written to disk
- Just the updated inode (I[v2]) is written to disk
- Just the updated bitmap (B[v2]) is written to disk

Solution #1: The File System Checker

- let inconsistencies happen and then fix them later

- fsck

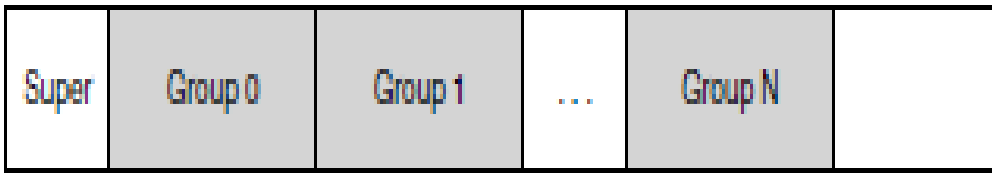
Solution #2: Journaling (orWrite-Ahead Logging)

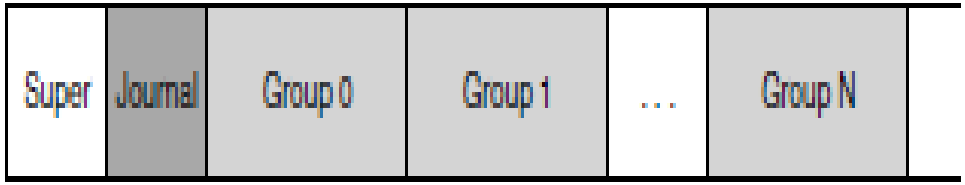
- write-ahead logging

- When updating the disk, before overwriting the structures in place, first write down a little note (somewhere else on the disk, in a well-known location) describing what you are about to do

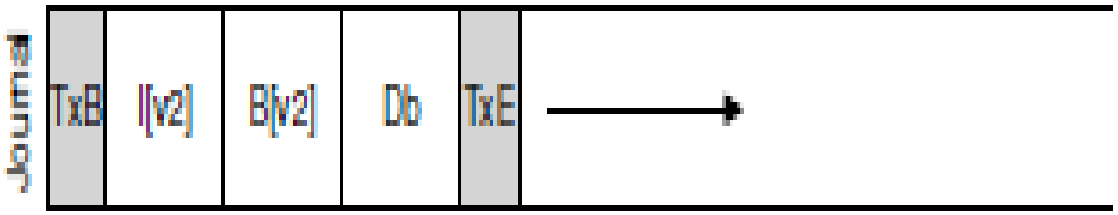
Continued…

- Linux ex2, ext3 → popular journaling file systems





Data Journaling



- 1. **Journal write:** Write the transaction, including a transaction-begin block, all pending data and metadata updates, and a transaction end block, to the log; wait for these writes to complete.
- 2. **Checkpoint:** Write the pending metadata and data updates to their final locations in the file system.

Solution #3

- Copy-on-Write (COW)
- Backpointer-based consistency

Log-structured File Systems

- Memory sizes were growing
- Large and growing gap between random I/O performance and sequential I/O performance
- Existing file systems perform poorly on many common workloads
- File systems were not RAID-aware: