Operating Systems

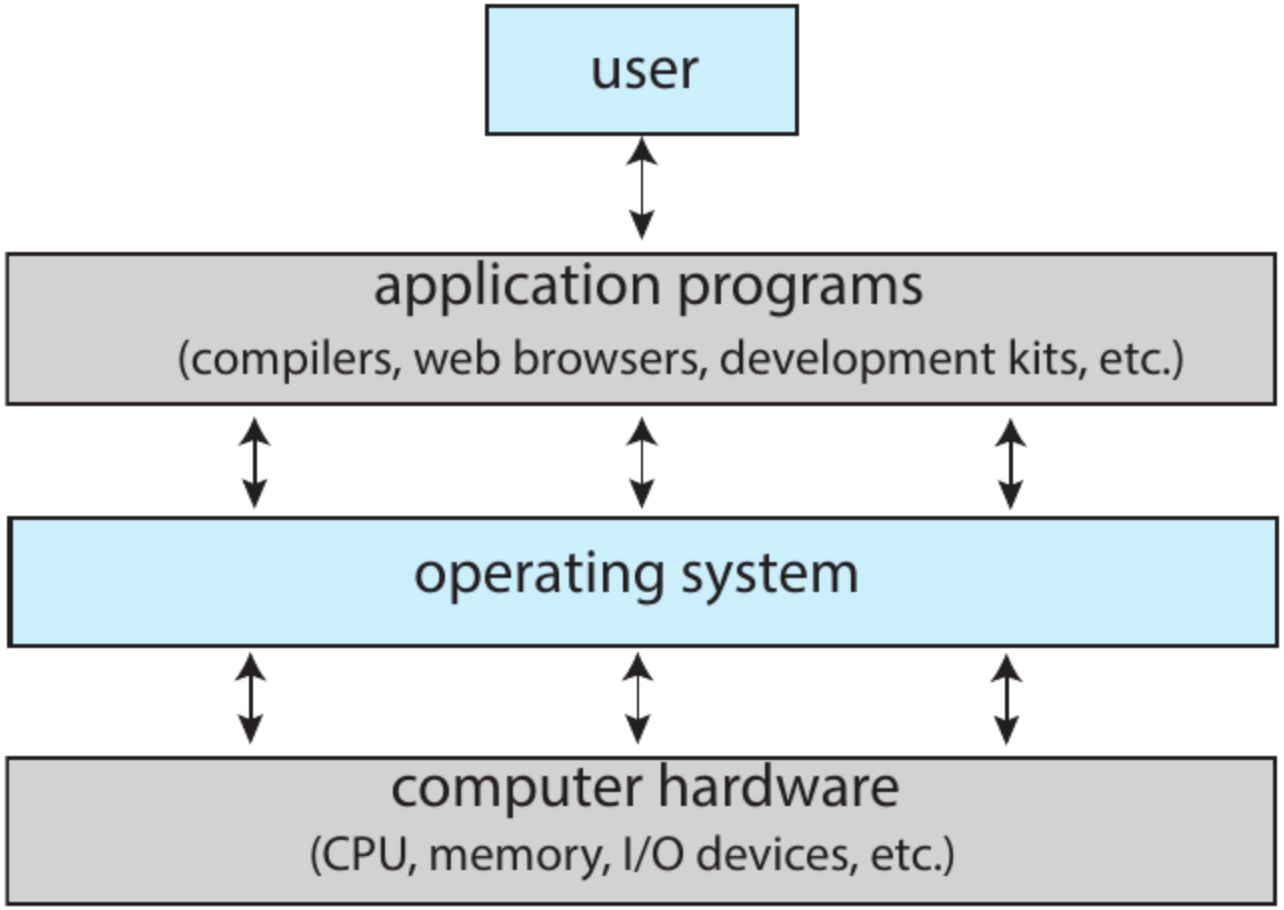
Isfahan University of Technology
Electrical and Computer Engineering Department
1402

Zeinab Zali

Session 2: Introduction to Operating System & Computer Environments

جلسه 2: مقدمه ای بر سیستم عامل و محیط های کامپیوتری

Four Components of a Computer System



چهار جزء یک سیستم کامپیوتری

Computer System Structure

- Computer system can be divided into four components
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

ساختار سیستم کامپیوتری
سیستم کامپیوتری را می توان به چهار جزء تقسیم کرد
سخت افزار: منابع محاسباتی اولیه را فراهم می کند
CPU، حافظه، دستگاه های ورودی/خروجی
سیستم عامل:
کنترل و هماهنگی استفاده از سخت افزار در بین برنامه ها و کاربران مختلف
برنامه های کاربردی: روش هایی را که در آن منابع سیستم برای حل مشکلات محاسباتی کاربران استفاده می شود را تعریف می کند
پردازشگرهای کلمه، کامپایلرها، مرورگرهای وب، سیستم های پایگاه داده، بازی های ویدیویی
کاربران:
مردم، ماشین ها، کامپیوترهای دیگر

Whats is an OS?

- There is a body of software, in fact, that is responsible for
  - ✔ making it easy to run programs (even allowing you to seemingly run many at the same time)
  - ✔ allowing programs to share memory and CPU
  - ✔ enabling programs to interact with devices
  - ✔ and other fun stuff
- That body of software is called the operating system (OS)
- OS is in charge of making sure the system operates correctly and efficiently in an easy-to-use manner

سیستم عامل چیست؟
یک مجموعه نرم افزاری وجود دارد که در واقع مسئول آن است

✓ اجرای آسان برنامه ها (حتی به شما این امکان را می دهد که به ظاهر چندین برنامه را همزمان اجرا کنید)
✓ به برنامه ها اجازه می دهد حافظه و CPU را به اشتراک بگذارند
✓ فعال کردن برنامه ها برای تعامل با دستگاه ها
✓ و چیزهای سرگرم کننده دیگر

آن مجموعه نرم افزار را سیستم عامل (OS) می نامند.
سیستم عامل مسئول اطمینان از عملکرد صحیح و کارآمد سیستم به روشی آسان است.

وظیفه سیستم عامل:
برنامه ها بتوانند از منابع اصلی کامپیوتر مثل سی پی یو و مموری به طور همزمان و عادلانه استفاده بکنند

OS three pieces

- Virtualization
  - ✔A primary way the OS does its roles
    - Specially its key role as a resource manager for managing two main resource **CPU and Memory**

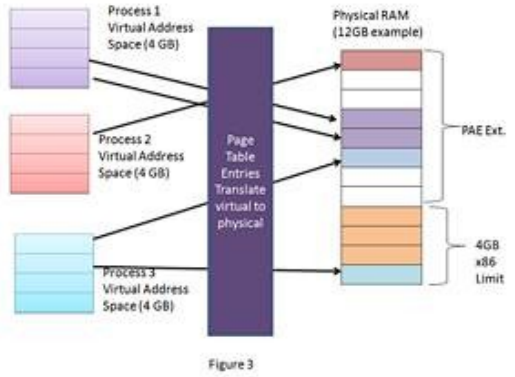
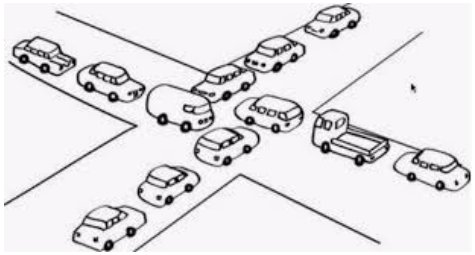


Figure 3

- Concurrency
  - ✔A conceptual term to refer to a host of problems that arise, and must be addressed, when working on many things at once in the same program
    - OS itself
    - Multi-thread or multi-process programs



- Persistence
  - ✔storing any files the user creates in a reliable and efficient manner on the disks of the system

Virtualization

- Process: OS abstraction of the processor, main memory and I/O devices for a running program
  - Multiple processes can concurrently run, each thinking itself as the exclusive user of the hardware.
- Virtual Memory: OS abstraction of Memory
  - Each process perceives the same picture of memory used only by itself (its address space).
- File: OS abstraction of I/O devices
  - All input and output in the system is performed by reading and writing files

CPU Virtualization

- virtualizing the CPU: Turning a single CPU (or small set of them) into a seemingly infinite number of CPUs and thus allowing many programs to seemingly run at once

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}
```

CPU Virtualization

- virtualizing the CPU: Turning a single CPU (or small set of them) into a seemingly infinite number of CPUs and thus allowing many programs to seemingly run at once

```
zeinabzali:./cpu A
A
A
A
A
A
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}
```

CPU Virtualization

- virtualizing the CPU: Turning a single CPU (or small set of them) into a seemingly infinite number of CPUs and thus allowing many programs to seemingly run at once

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}
```

```
zeinabzali:./cpu A & ./cpu B & ./cpu C
[1] 27878
[2] 27879
A
B
C
A
B
C
A
B
C
A
B
C
A
B
C
A
```

Memory Virtualization

- Memory is just an array of bytes
  - to read memory, one must specify an address to be able to access the data stored there
  - to write (or update) memory, one must also specify the data to be written to the given address
- A program keeps all of its data structures in memory, and accesses them through various instructions
  - loads and stores
- Each instruction of the program is in memory too
  - thus memory is accessed on each instruction fetch

Memory Virtualization

- virtualizing the Memory: Each process accesses its own private virtual address space which the OS somehow maps onto the physical memory of the machine

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: mem <value>\n");
        exit(1);
    }
    int *p;
    p = malloc(sizeof(int));
    assert(p != NULL);
    printf("(%d) addr pointed to by p: %p\n", (int) getpid(), p);
    *p = atoi(argv[1]); // assign value to addr stored in p
    //p = atoi(argv[1]);
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) value of p: %d\n", getpid(), *p);
    }
    return 0;
}
```

Memory Virtualization

- virtualizing the Memory: Each process accesses its own private virtual address space which the OS somehow maps onto the physical memory of the machine

```
setarch $(uname --machine) --addr-no-randomize /bin/bash
```

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: mem <value>\n");
        exit(1);
    }
    int *p;
    p = malloc(sizeof(int));
    assert(p != NULL);
    printf("(%d) addr pointed to by p: %p\n", (int) getpid(), p);
    *p = atoi(argv[1]); // assign value to addr stored in p
    //p = atoi(argv[1]);
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) value of p: %d\n", getpid(), *p);
    }
    return 0;
}
```

```
zeinabzali:./mem 10
(28692) addr pointed to by p: 0x555555756260
(28692) value of p: 11
(28692) value of p: 12
(28692) value of p: 13
(28692) value of p: 14
```

Memory Virtualization

- virtualizing the Memory: Each process accesses its own private virtual address space which the OS somehow maps onto the physical memory of the machine

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: mem <value>\n");
        exit(1);
    }
    int *p;
    p = malloc(sizeof(int));
    assert(p != NULL);
    printf("(%d) addr pointed to by p: %p\n", (int) getpid(), p);
    *p = atoi(argv[1]); // assign value to addr stored in p
    //p = atoi(argv[1]);
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) value of p: %d\n", getpid(), *p);
    }
    return 0;
}
```

```
zeinabzali:./mem 10 & ./mem 100 & ./mem 1000
[1] 28699
[2] 28700
(28699) addr pointed to by p: 0x555555756260
(28700) addr pointed to by p: 0x555555756260
(28701) addr pointed to by p: 0x555555756260
(28699) value of p: 11
(28700) value of p: 101
(28701) value of p: 1001
(28699) value of p: 12
(28700) value of p: 102
(28701) value of p: 1002
(28699) value of p: 13
(28700) value of p: 103
(28701) value of p: 1003
(28699) value of p: 14
(28700) value of p: 104
(28701) value of p: 1004
(28699) value of p: 15
(28700) value of p: 105
(28701) value of p: 1005
```

Concurrency

- problems that arise, and must be addressed, when working on many things at once (i.e., concurrently)

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
#include "common_threads.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: threads <loops>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);
    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value   : %d\n", counter);
    return 0;
}
```

Concurrency

- problems that arise, and must be addressed, when working on many things at once (i.e., concurrently)

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
#include "common_threads.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: threads <loops>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);
    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value   : %d\n", counter);
    return 0;
}
```

```
zeinabzali:./threads 10
Initial value : 0
Final value   : 20
zeinabzali:./threads 100
Initial value : 0
Final value   : 200
zeinabzali:./threads 1000
Initial value : 0
Final value   : 2000
zeinabzali:10000
10000: command not found
zeinabzali:./threads 10000
Initial value : 0
Final value   : 14991
zeinabzali:
```

OS design goals

- build up some abstractions in order to make the system convenient and **easy to use.**
  - **Abstractions are fundamental to everything we do in computer science**
- **high performance**; minimizing the overheads of the OS
- **Protection** between applications, as well as between the OS and applications
  - making sure that the malicious or accidental bad behavior of one program does not harm others;
  - **isolating processes**
- **Reliability**; OS must run non-stop; when it fails, all applications running on the system fail as well
- **Security**; an extension of protection, really against malicious applications especially in these highly-networked times
- Others: mobility, energy efficiency, ...

Sum up: What is an Operating System?

- ✓ A program that acts as an intermediary between a user of a computer and the computer hardware.
- ✓ Operating system operations
  - Process Management
  - Memory Management
  - Storage Management
  - I/O handling
  - Protection and Security
    - user-ID, Group-ID, permission
    - Viruses, attacks, intrusion

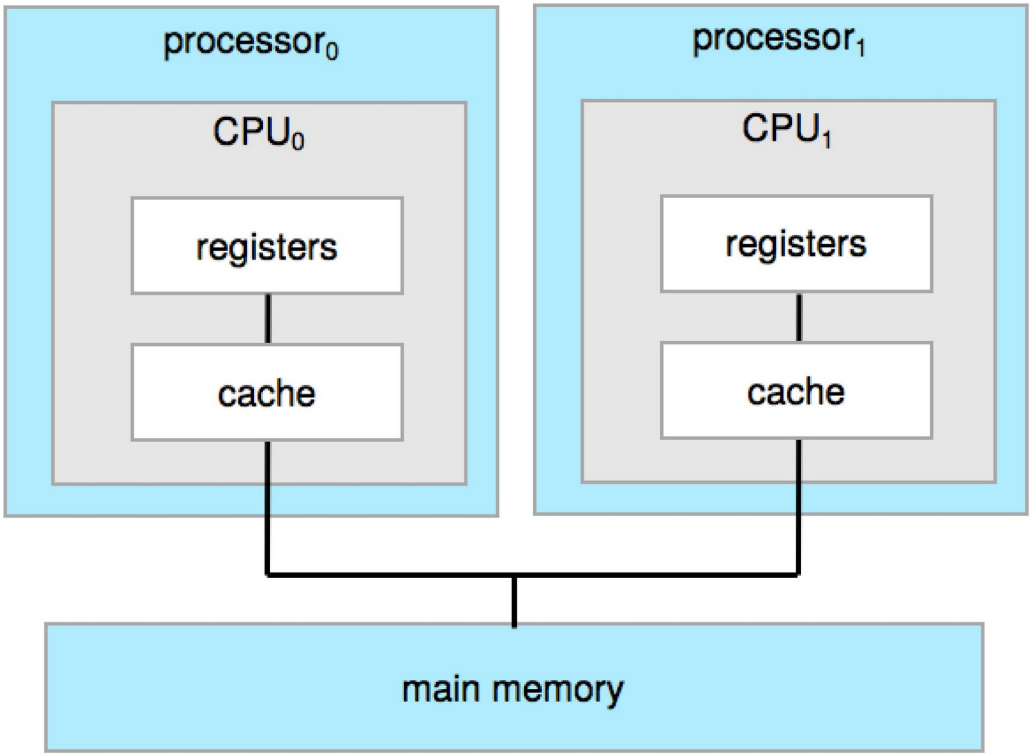
Computer Environments

Multiprocessors

- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance
  - Two types:
    1. **Asymmetric Multiprocessing** – each processor is assigned a specie task.
    2. **Symmetric Multiprocessing** – each processor performs all tasks

-
در اول یک سیستمی داشتیم که یک پروسسور داشت حالا میتونیم این پروسسور ها رو برای اینکه کارایی بیشتری داشته باشند و ازش بتونیم کارکرد بیشتری بکشیم می تونیم چندتا پروسسور رو کنار هم بذاریم و همزمان ازشون استفاده بکنیم این باعث میشه سیستم مالتی پروسسورمون هم throughput بالاتری داشته باشه و هم scale بالاتری داشته باشه و هم reliability باشه یعنی اگر یکی از پروسسور ها به هر دلیلی خراب شد برنامه هامون دچار مشکل نشن و سریع بتونیم از بقیه پروسسور ها استفاده بکنیم
اینکه چجوری پروسسور ها رو کنار هم گذاشتیم و ازش استفاده کردیم دوتا نوع داره:
1- چند پردازش نا متقارن : یک سی پی یو مستر داریم و یه تعدادی سی پی یو که مستر نیستند و این یکسری مدیریت اجرا روی این سی پی یوهای مختلف رو بهمون میده
2- متقارن چند پردازشی : ولی وقتی یک سیستم فیزیکی واحد داریم که توش چندتا پروسسور گذاشتیم حالت متقارن داریم

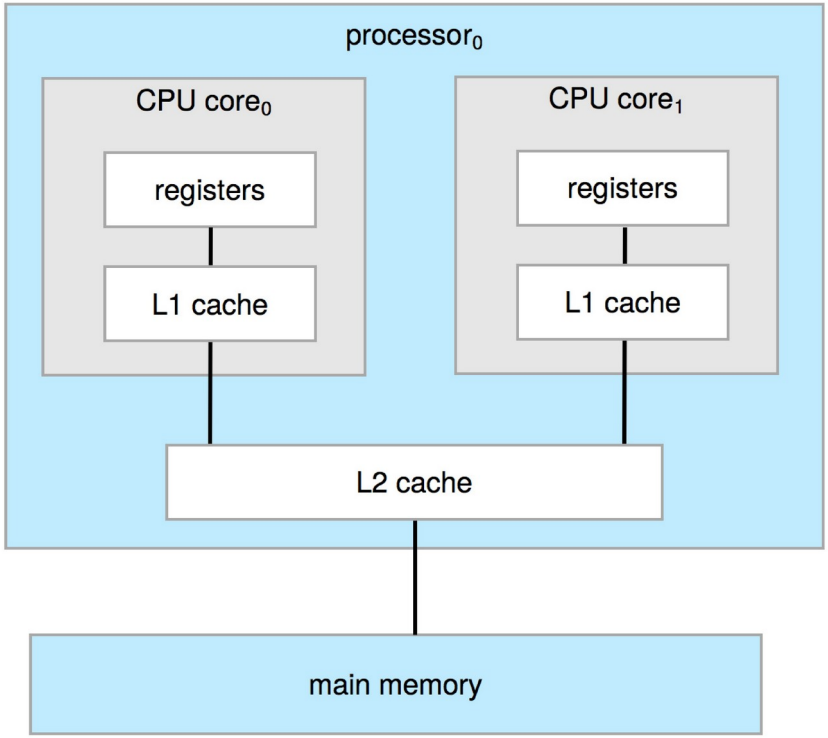
Symmetric Multiprocessing Architecture



-
یک معماری متقارن داریم
این توی یک سیستم سخت افزاری قرار گرفته و دوتا پروسسور داره که از طریق باس سیستم بهم وصل هستند و دارند به صورت مشترک از یک مین مموری استفاده میکنند

Dual-Core Design

- Multi-chip and **multicore**
- Systems containing all chips
  - Chassis containing multiple separate systems

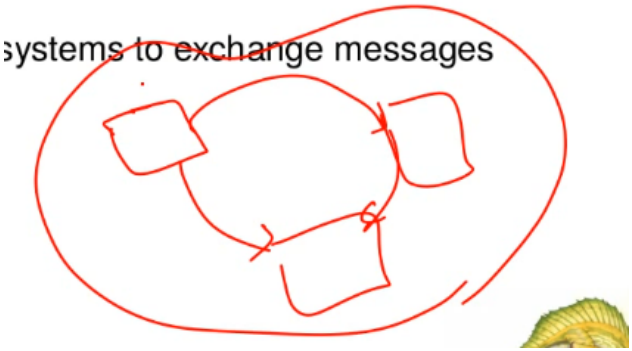


-
توی سیستم های فعلی multicore داریم ینی چندتا سی پی یو رو روی یک چیپ قرار دادند وکل پروسسورمون این یک چیپ میشه
روی همچین سیستم هایی دو تا لایه کش داریم که یک کش مال هر core است و یک کش هم مال کل core هاست
و اینجا نیاز به باس نیست برای ارتباط این سی پی یوها

Distributed Systems

- Collection of separate, possibly heterogeneous, systems networked together
  - **Network** is a communications path, **TCP/IP** most common
    - **Local Area Network** (**LAN**)
    - **Wide Area Network** (**WAN**)
    - **Metropolitan Area Network** (**MAN**)
    - **Personal Area Network** (**PAN**)
- **Network Operating System** provides features between systems across network
  - Communication scheme allows systems to exchange messages
  - Illusion of a single system

-
سیستم های توزیع شده ینی چند تا سیستم رو همزمان با هم می خوایم استفاده بکنیم که معمولا همچین سیستم های برای اتصال از network استفاده میکنند
حالا اگر بخوایم این کامپیوترهایی که توی نت ورک بهم وصل شدند رو در واقع یک جور واحد ببینیم و مثلا بخوایم برنامه روشون اجرا کنیم فکر کنیم مثلا این سه تا یک جز واحد هستند سیستم عاملی هم که برای اینا نیاز داریم هم باید یه همچین کاری برای ما بکنه که بهش میگن network operating system که یک دیدی به ما بده که انگار این سه تا یکجا دارن استفاده میشن

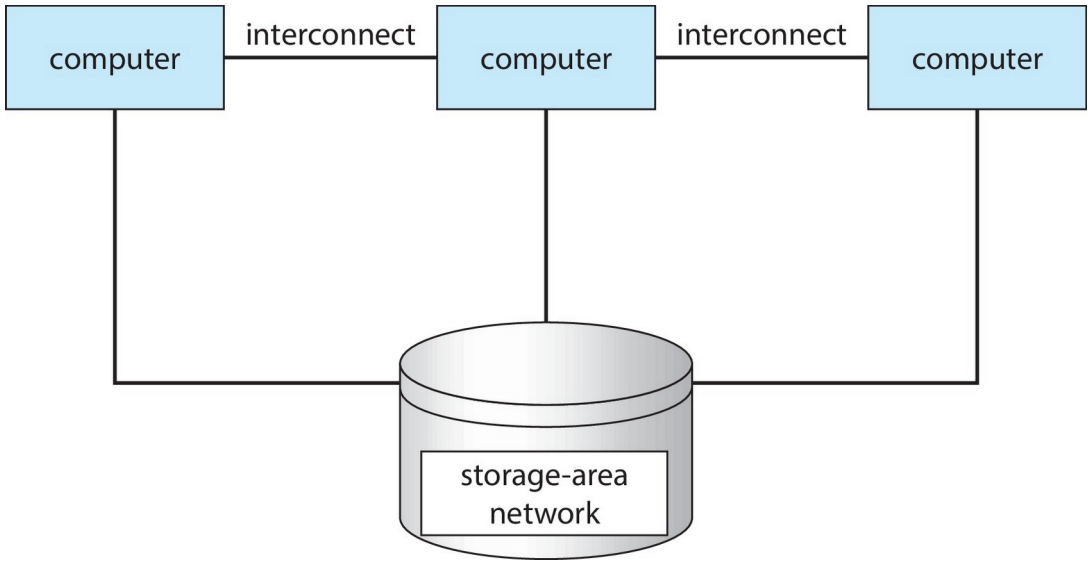


Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network** (**SAN**)
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode
    - **Symmetric clustering** has multiple nodes running applications, monitoring each other
  - Some clusters are for **high-performance computing** (**HPC**)
    - Applications must be written to use **parallelization**
  - Some have **distributed lock manager** (**DLM**) to avoid conflicting operations

-
حالا یه موقعی است که سیستم ها چه مالتی پروسسور چه سینگل پروسسور چه multicore باشند می تونیم این هارو یه تعدادیشون رو کنار هم بذاریم و از طریق شبکه اینارو بهم وصل کنیم که حتی می تونند فضای storage خارجی داشته باشند که به صورت مشترک استفاده میشه مثل صفحه بعدی
به همچین سیستم هایی میگیم clustered system که این ها معمولا از یک storage-area network استفاده می کنند که همین storage-area network هم خودش ممکنه به صورت توزیع شده باشه ینی از چندتا storage کنار هم استفاده شده که بهم شبکه شدند و این کامپیوترها دارند ازشون استفاده میکنند
دو نوع Clustered داریم :
Clustered متقارن: ؟
Clustered نامتقارن:؟
این Clustered هارو معمولا توی HPC استفاده میکنند ینی جاهایی که ما محاسبات با عملکرد بالا نیاز داریم

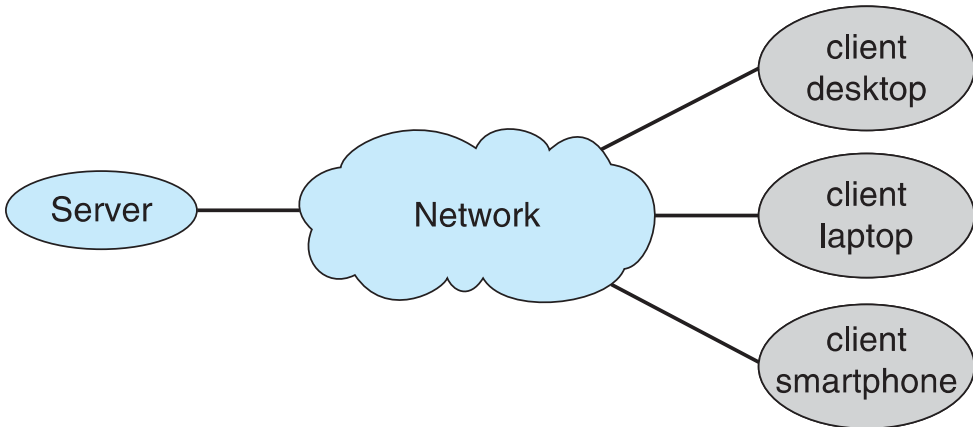
Clustered Systems



سیستم های خوشه ای

Client Server

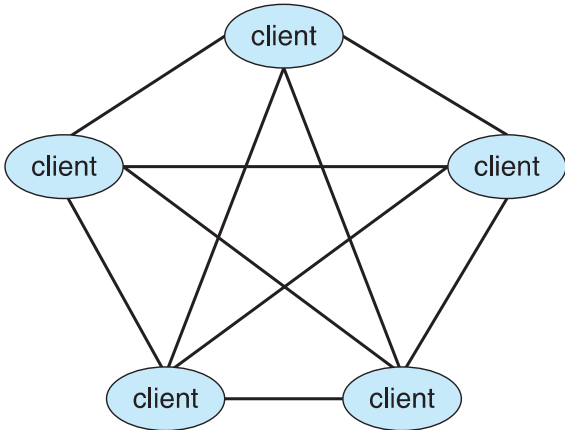
- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (i.e., database)
    - **File-server system** provides interface for clients to store and retrieve files

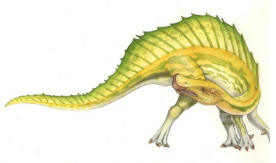


-
Client Server ینی روی شبکه ما یک جایی یک ماشینی داریم به عنوان سرور و یک جاهای دیگه یه سری کلاینت که از طریق نت ورک بهم وصل شدند ینی مثلا سرور توی یک کشور دیگه است و مثلا کلاینت ها یه جاهای دیگه
ولی بحث clustered اینطوری نیست ینی معمولا یکجا قرار دارند ولی کلاینت سرور نه ینی جاهای مختلفی هستند

Peer-to-Peer

- Another model of distributed system
- **P2P does not distinguish clients and servers**
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via ***discovery protocol***
  - Examples include Napster and Gnutella, **Voice over IP** (**VoIP**) such as Skype

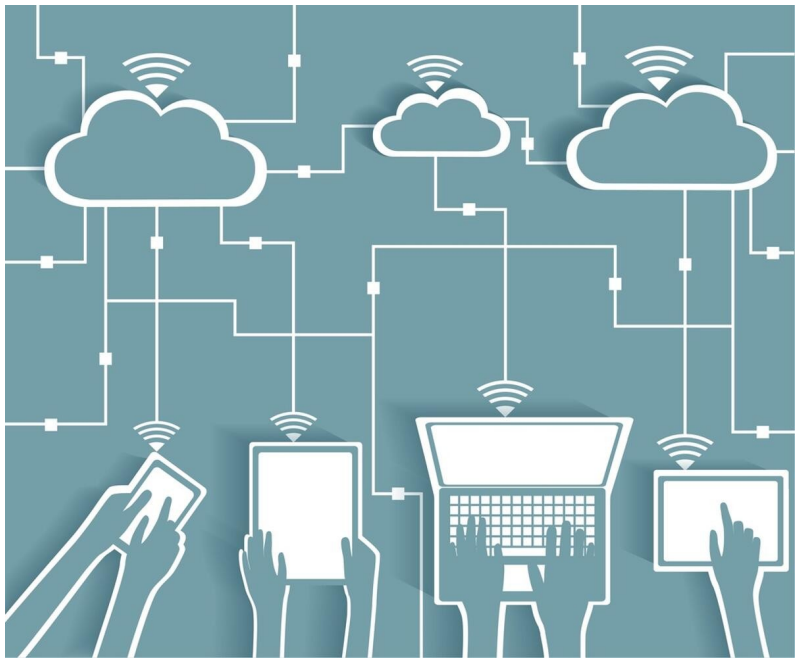




-
اینجا هم کلاینت هارو از طریق یک شبکه بهم وصل میکنیم ولی دیگه این وسط سروری نداریم ینی همه این اجزایی که بهم وصل شده اند حکم کلاینت رو دارند

Cloud Computing

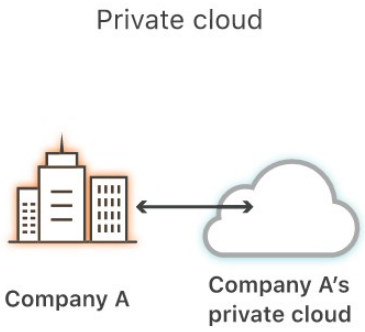
- Delivers computing, storage, even apps **as a service** across a network
- Logical extension of virtualization because it uses virtualization as the base for its functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage

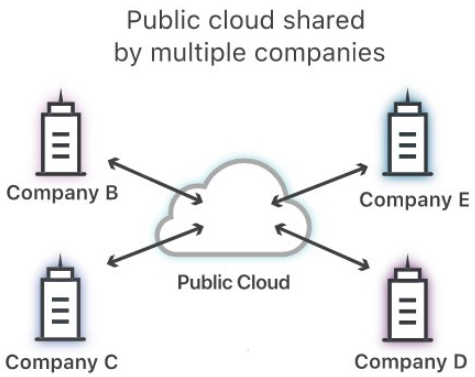


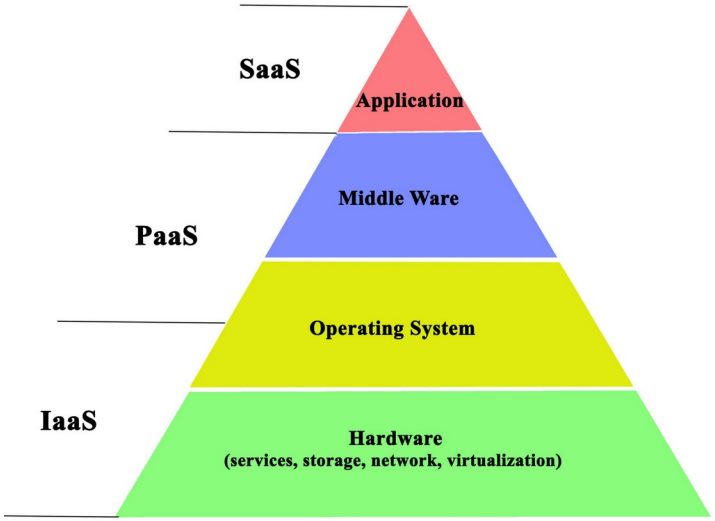
-
نوع کارکردش جوریه که اجازه محاسبات و فضای ذخیره سازی به کلاینت ها میده به صورتی که کلاینت ها اینارو به صورت یک سرویس می بینند
چیزی که cloud رو از Clustered جدا میکنه سرویس بودن است ینی نوع استفاده cloud به صورت سرویس است
توی این شکل کلاینت هایی مختلفی است که به یک ابری وصل اند که این ابره از طریق سیستم های Clustered شده ای ایجاد شده و یکسری نرم افزارهایی روش نصب شده که استفاده این کلاینت ها از این سیستم های سخت افزاری روی Clustered های cloud رو راحت میکنه
توی پیاده سازی cloudها از virtualization استفاده میشه
یکی از cloud ها amazon EC2 است

Cloud Computing types

- **Public cloud** – available via Internet to anyone willing to pay
- **Private cloud** – run by a company for the company's own use
- **Hybrid cloud** – includes both public and private cloud components
- Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
- Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
- Infrastructure as a Service (**IaaS**) – servers or storage available over Internet



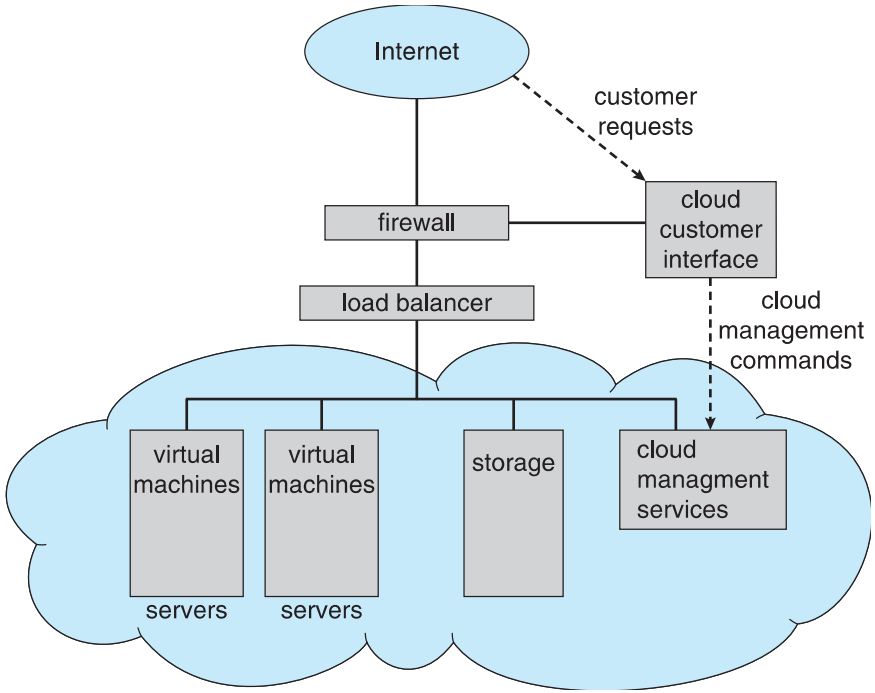




-
سه مدل ابر داریم:
Saas , Paas , IaaS
ما اکثرا از SaaS استفاده کردیم

Cloud Computing Environments

- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
  - Internet connectivity requires security like firewalls
  - Load balancers spread traffic across multiple applications



توی ابر

+ محیط های رایانش ابری متشکل از سیستم عامل های سنتی، به علاوه VMM، به علاوه ابزارهای مدیریت ابری
- اتصال به اینترنت نیاز به امنیت مانند فایروال دارد
- متعادل کننده بار ترافیک را در چندین برنامه پخش می کند

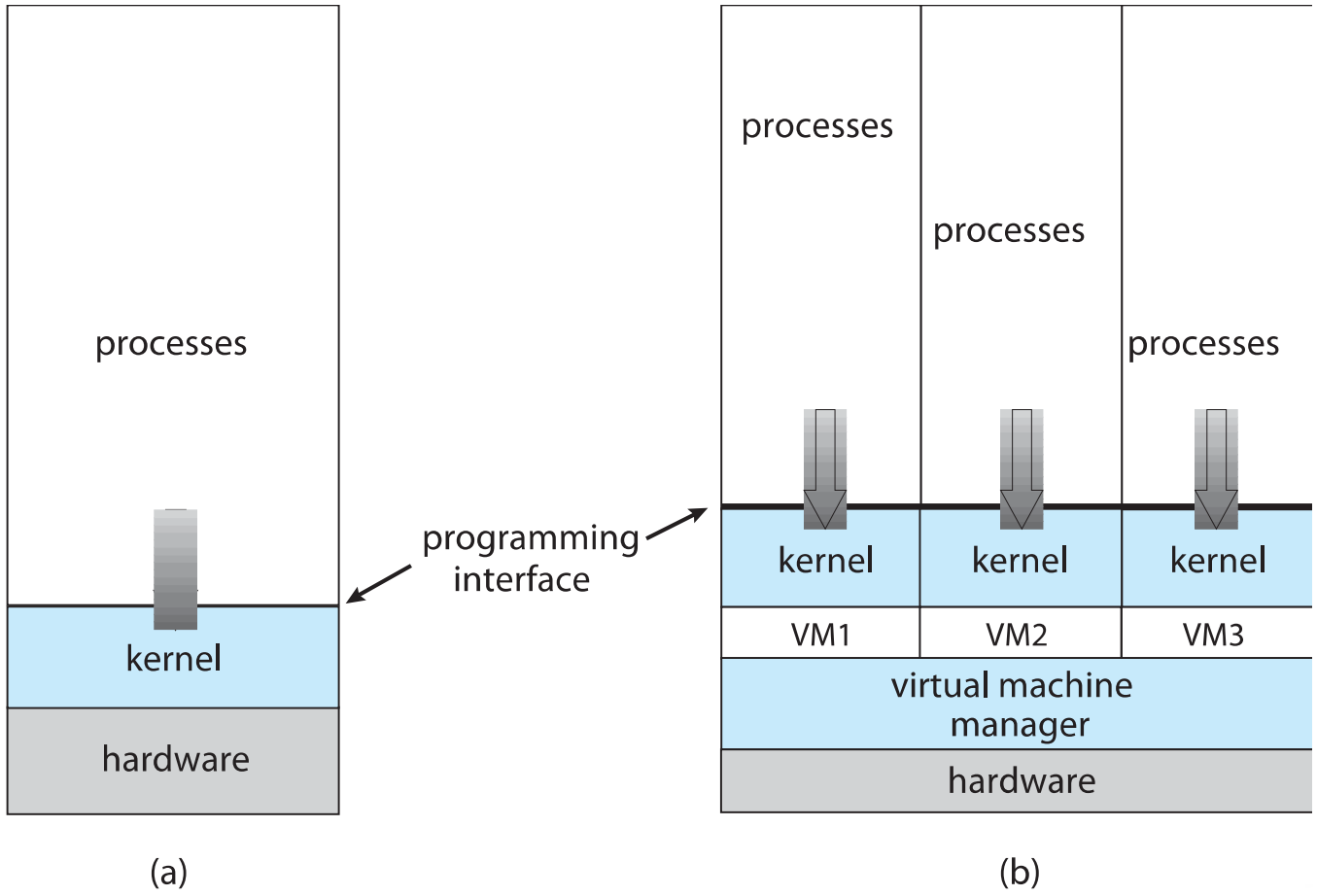
load balancer : چون سرورهای مختلفی توی این ابر داریم باید به صورت عادلانه ای این منابع سرورهارو به کلاینت های مختلف بدیم ینی نباید سر یک سرور خیلی شلوغ باشه و سر یک سرور دیگه خلوت تر
و کلاینت ها که جاهای مختلف وجود دارند از طریق اینترنت و api های مخصوص این ابر می تونند وصل بشن به این ابر

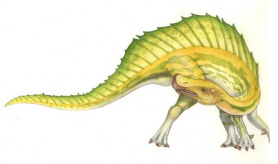
Virtualization

- Virualization is a technology that allows us to abstract the hardware of a single computer (the CPU , memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate environment is running on its own private computer.
- A user of a virtual machine can switch among the various operating systems in the same way a user can switch among the various processes running concurrently in a single operating system.
- Emulation is simulating computer hardware in software.
- Broadly speaking, virtualization software is one member of a class that also includes emulation.
  - Emulation is typically used when the source CPU type is different from the target CPU type.

-
جاهای دیگه هم از مجازی سازی استفاده میکنیم به طور کلی تکنولوژی مجازی سازی که میگیم ینی وقتی که یک سخت افزاری رو که مربوط به یک کامپیوتر واحد است حالا ممکنه سی پی یوش باشه یا دیسکش باشه یا ... یه جوری به تعداد زیادی محیط اجرایی مپش کنیم ینی مثلا سی پی یو رو برای تعداد نامتناهی برنامه بتونیم استفاده بکنیم به طور کلی اگر کل سخت افزار سیستم رو یه همچین کاری باهاش بکنیم همون کار مجازی سازی است
حالا گاها چی کار میکنن؟ به نوعی تمام این هارو مثل سی پی یو و.. یه قسمت هاییش رو انگار جدا میکنیم و برای یک سیستم در نظر میگیریم که بهش میگیم virtual machine که مثلا با vmvar این کارو کردیم توضیح دقیق صفحه بعد
emulation یا شبیه سازی : وقتی که یک سخت افزاری رو به صورت نرم افزار بیایم نشونش بدیم مثلا روی سیستم سی پی یو IBM داشتیم حالا میخوایم یک برنامه ای می خوایم داشته باشیم ؟ نفهمیدم
نکته: گاها توی جاهایی که virtualization داریم یک شبیه سازی هم ممکنه داشته باشیم

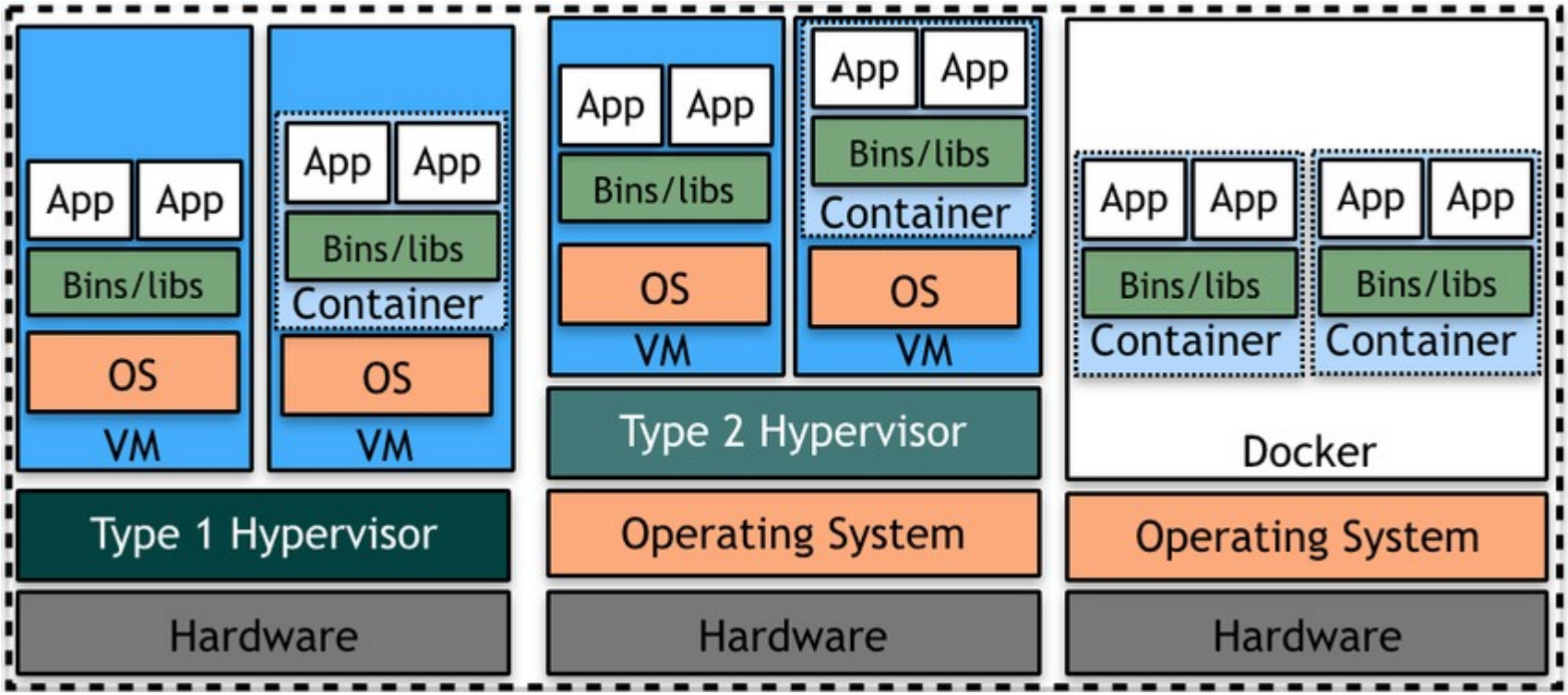
Computing Environments - Virtualization

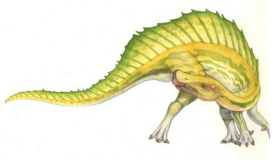




-
مثلا ما یک سخت افزاری داشتیم بعد یک سیستم عامل روش نصب کرده که یک کرنلی داره ولی اگر یک virtual machine manager نصب کنیم مثل vmware بعد می تونیم چندتا سیستم عامل همزمان اینجا داشته باشیم در واقع این virtual machine manager باعث شده ما سه تا سیستم کاملا متفاوت ببینیم روی این سخت افزار

Container-based Virtualization





-
توی سیستم اول (از چپ به راست) ما یک سخت افزار داریم و یک type 1 hypervisor داریم که این همون virtual machine است که مثلا توی مثال قبلی یک vmware بود
ولی چیزی که ما استفاده میکنیم دومی است تقریبا که اول یک سیستم عامل داریم که بهش می گیم میزبان و بعد روی اون یک virtual machine داریم که بعد سیستم های دیگر میشن مهمان ولی توی شکل اول میزبان نداریم
اما container چی؟ این هم یک virtualization است ولی سطحش بالاتر از چیزایی است که تا حالا بیان شده - اینجا ما یک os داریم و VM های ما اینجا نوعشون container-based است
توی این VM دیگه os را نمی بینیم - این روش سبک تر است چون os نداره
یکی از ابزارهایی که باهاش می شه container بسازیم داکر است