Operating Systems

Isfahan University of Technology
Electrical and Computer Engineering Department
1400-1 semester

Zeinab Zali

Session 4: Virtualization and Computing Environments

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 addres
- 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

```
#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)

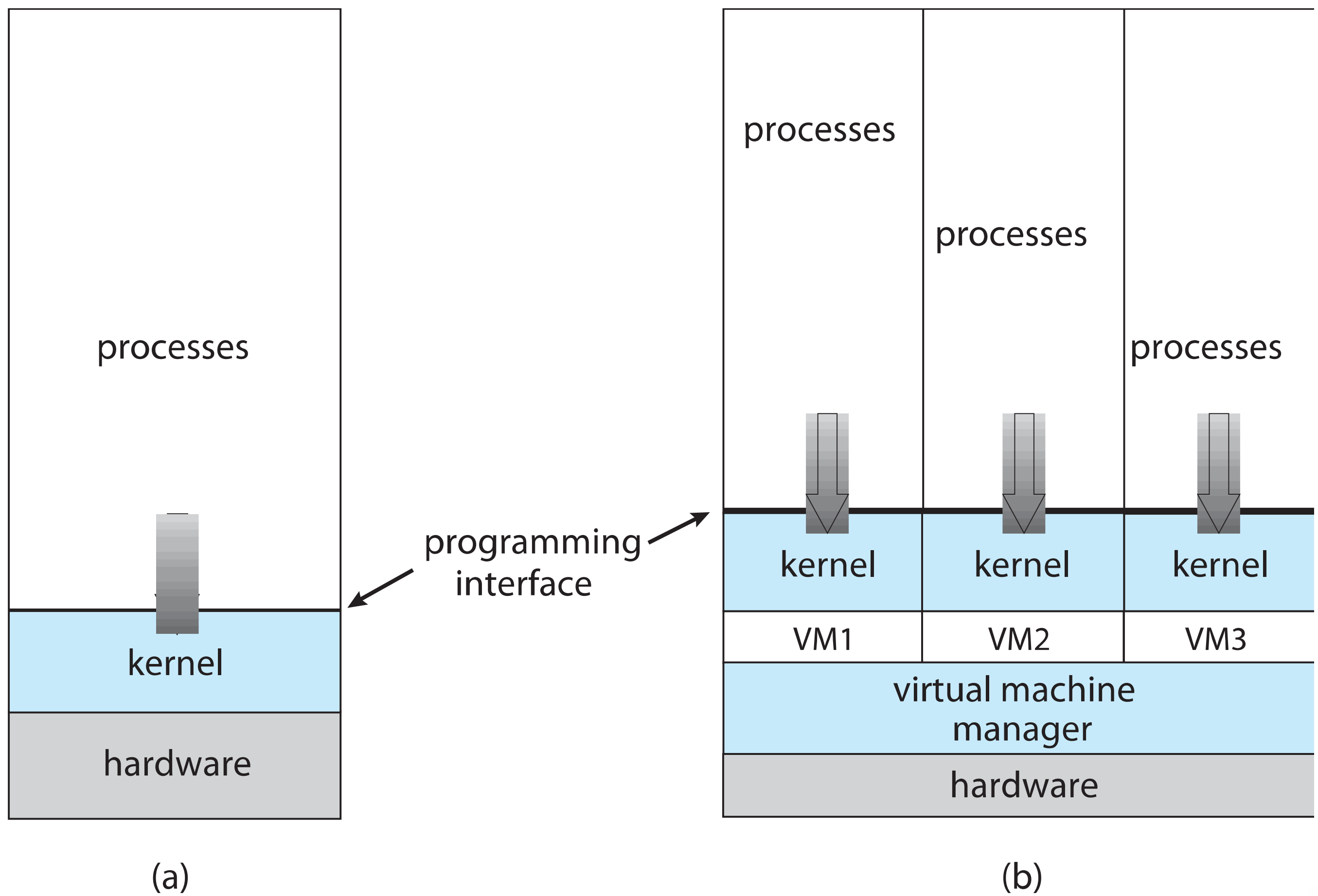
```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
#include "common_threads.h"

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:
```

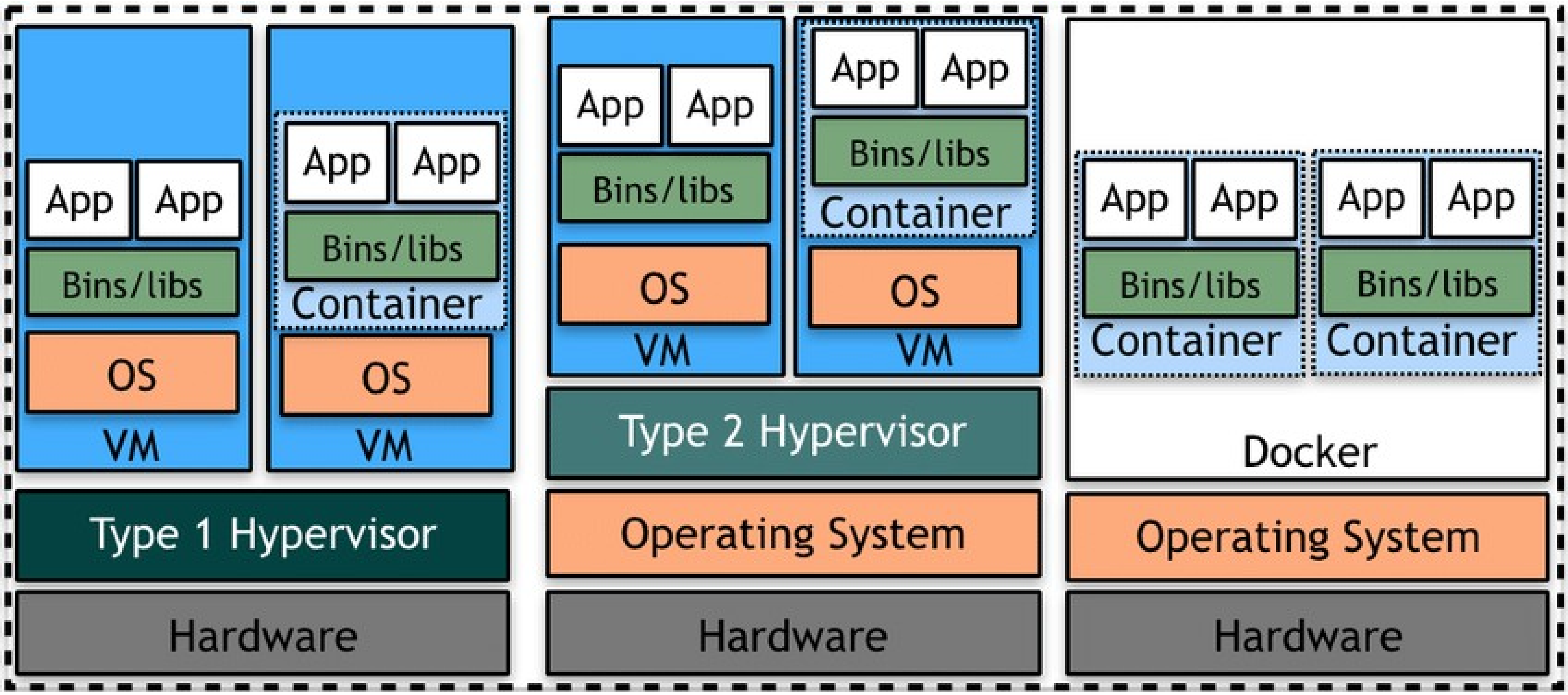
Virtualization

- 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.

Computing Environments - Virtualization



Container-based Virtualization



Computing Environments

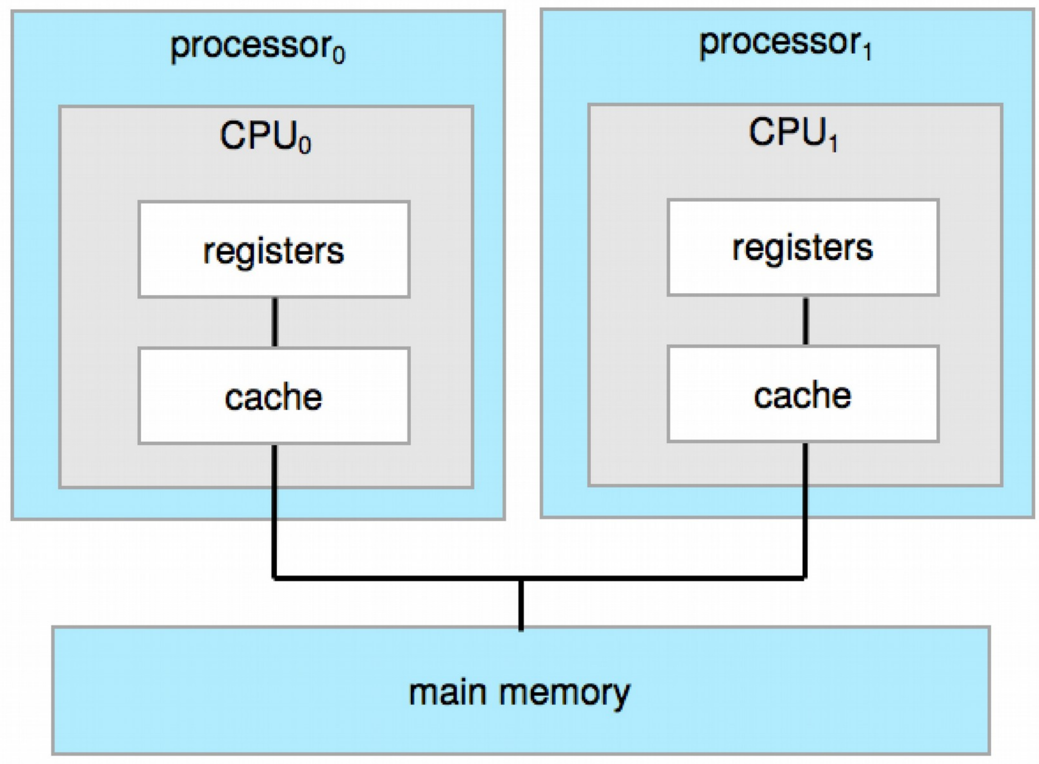
Distributed Systems

- 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

Computer-System Architecture

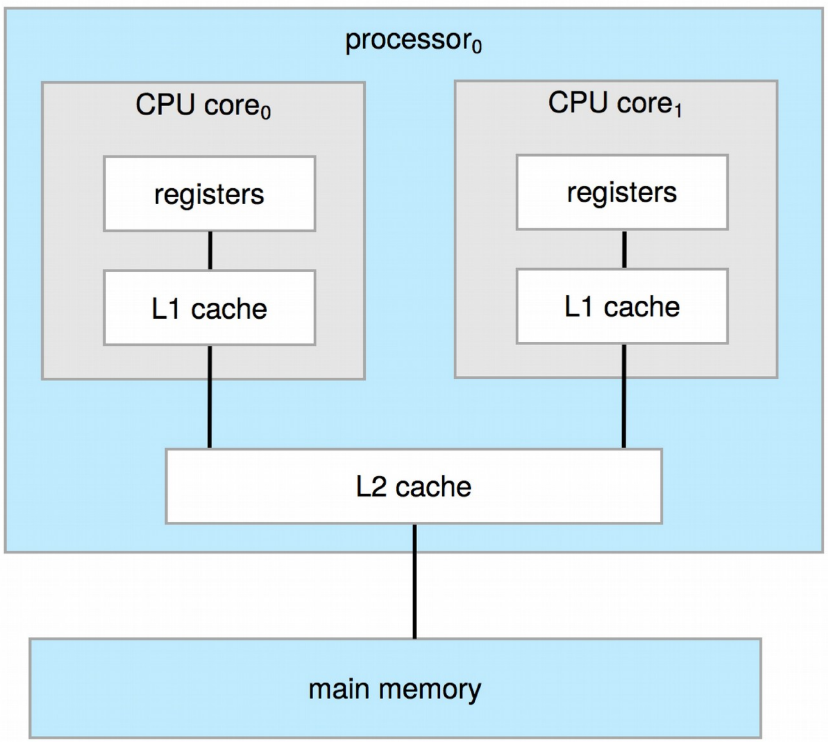
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance
  - Two types: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

Symmetric Multiprocessing Architecture



Dual-Core Design

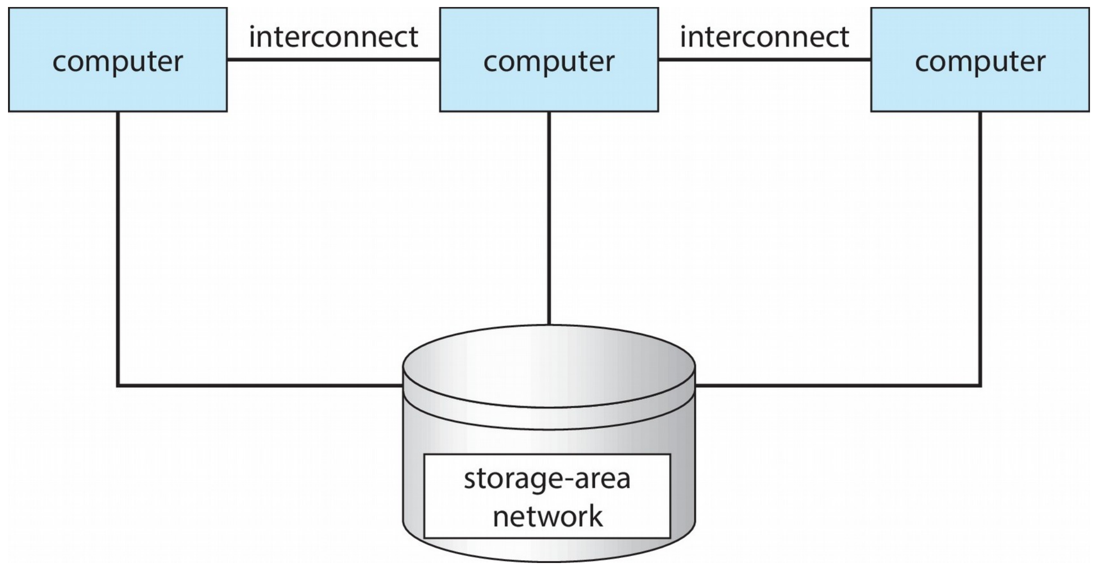
- Multi-chip and **multicore**
- Systems containing all chips
  - Chassis containing multiple separate systems



Clustered Systems

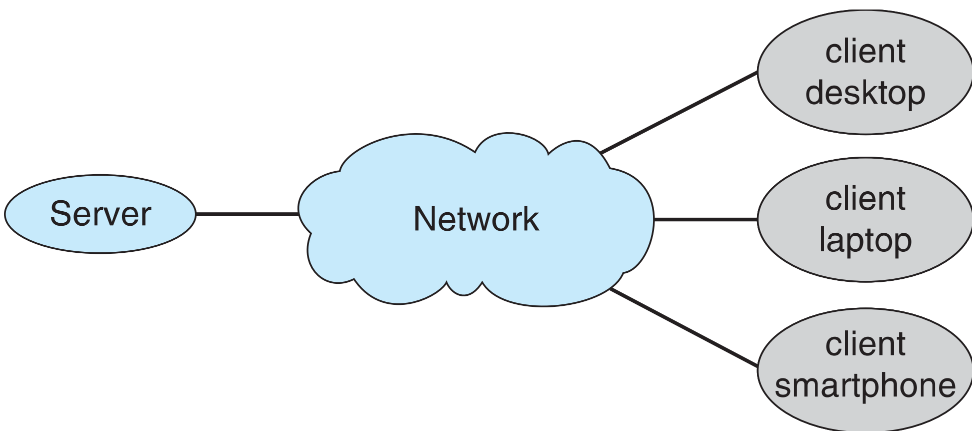
- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network** (**SAN**)
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode
    - **Symmetric clustering** 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

Clustered Systems



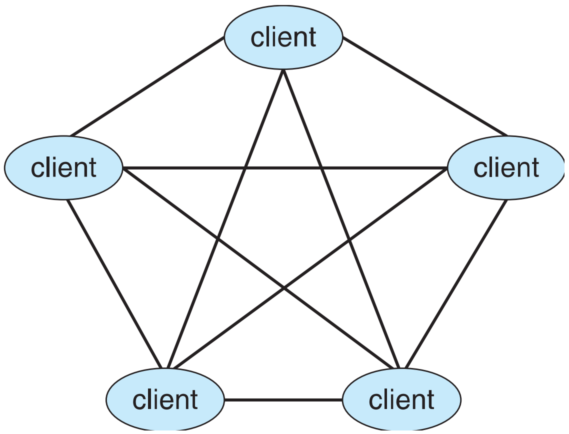
Client Server

- 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



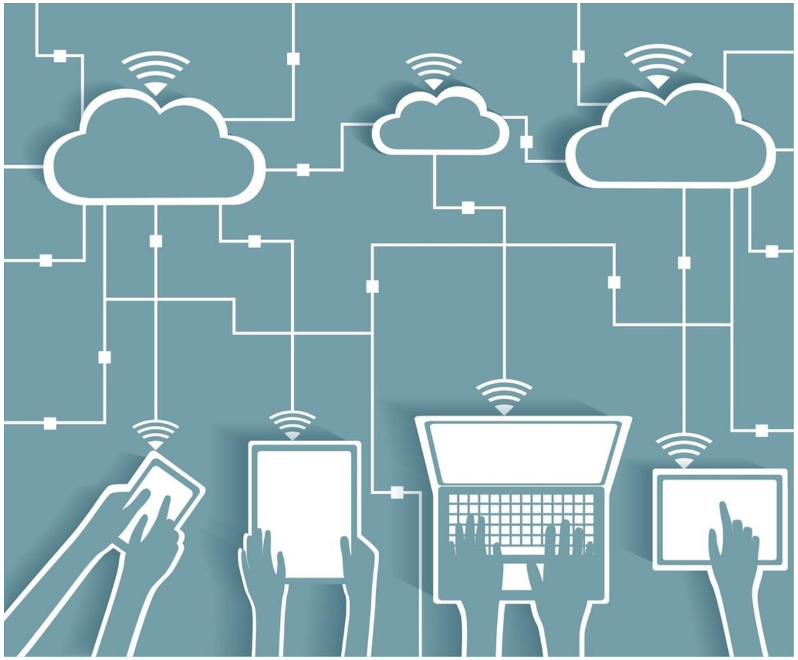
Peer-to-Peer

- 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



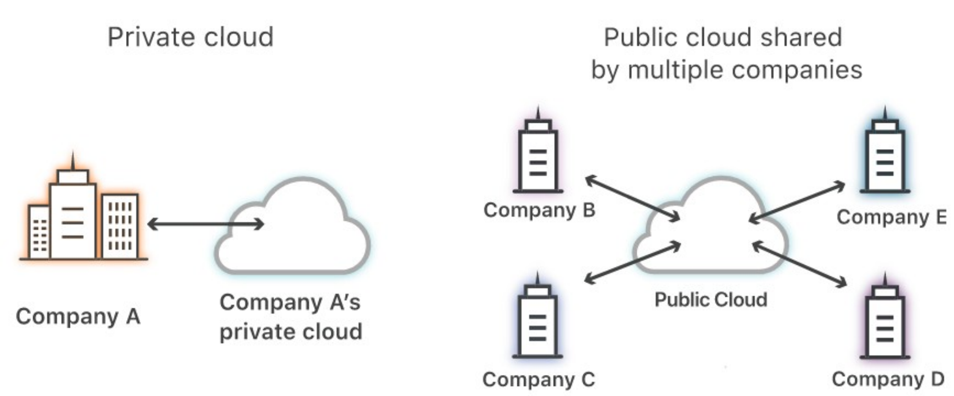
Cloud Computing

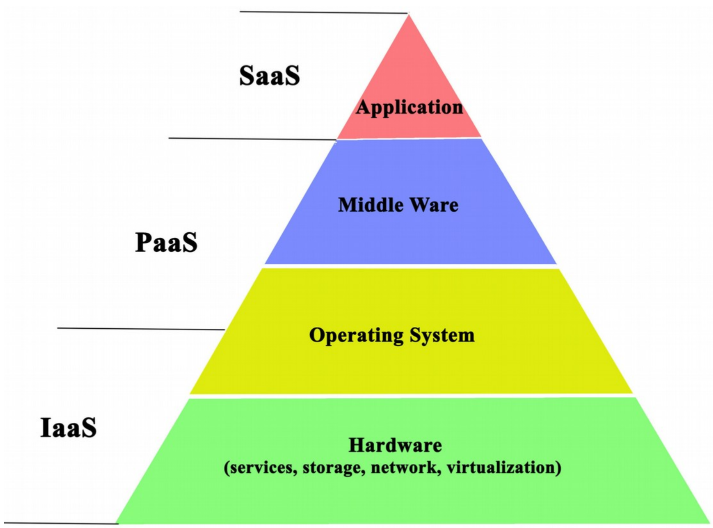
- Delivers computing, storage, even apps **as a service** across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet,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



Cloud Computing types

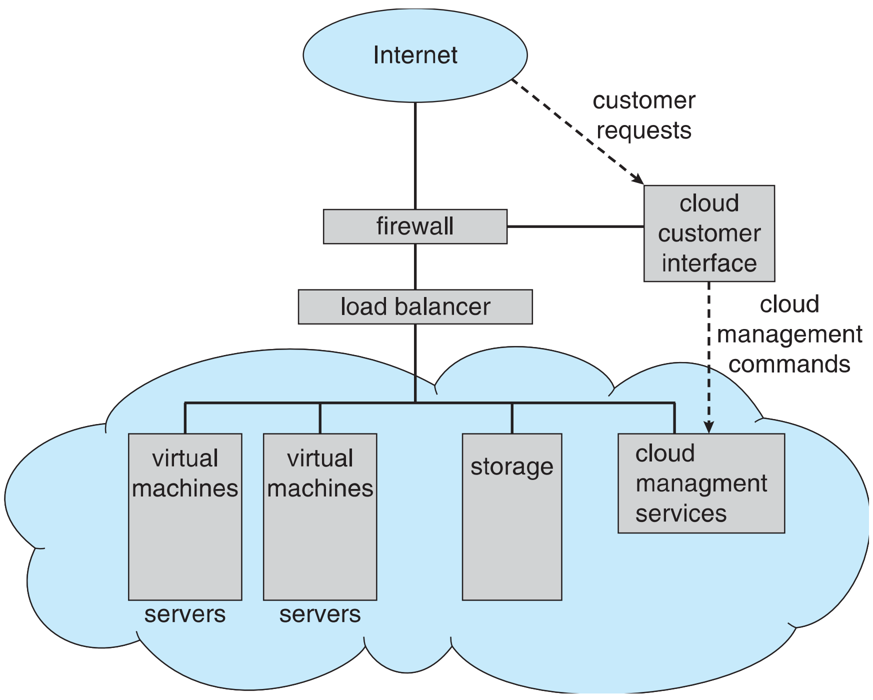
- **Public cloud** – available via Internet to anyone willing to pay
- **Private cloud** – run by a company for the company's own use
- **Hybrid cloud** – includes both public and private cloud components
- Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., 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





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



Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
  - Vary considerable, special purpose, limited purpose OS, **real-time OS**
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met
- Examples of RTOS (Real-Time OS)
  - VxWorks
  - LynxOS