Distributed Systems

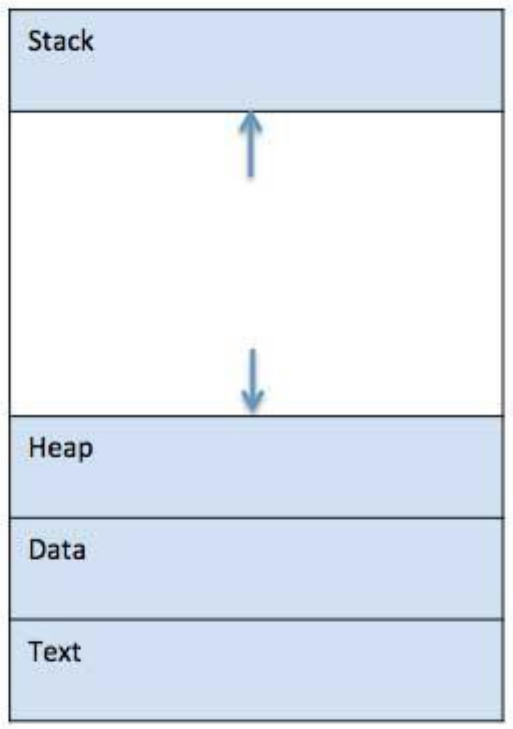
Processes
Chapter 3

Processes

- *Communication* takes place between *processes*.
- **But, what's a process?** *"A program in execution"*.
- **Traditional operating systems**: concerned with the "local" management and scheduling of processes.
- **Modern distributed systems**: a number of other issues are of equal importance.
- There are three main areas of study:

1. Threads and virtualization within clients/servers
2. Process and code migration
3. Software agents

Process



process

- **Stack**

The process Stack contains the temporary data such as method/function parameters, return address and local variables.

- **Heap**

This is dynamically allocated memory to a process during its run time.

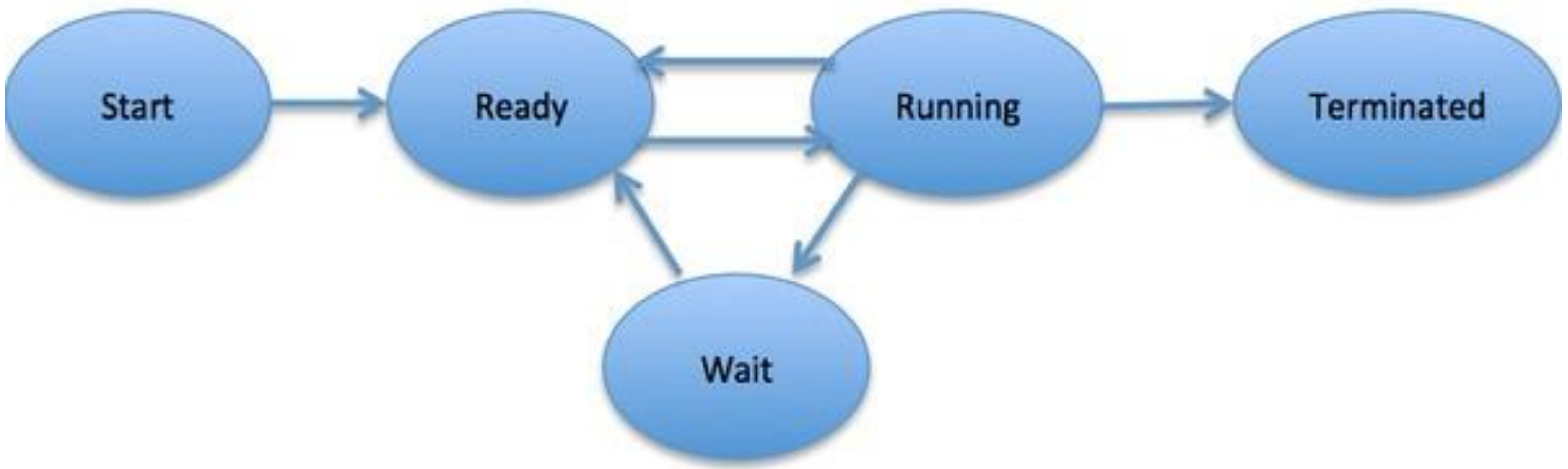
Text

- This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

Data

This section contains the global and static variables

Process cycle



Process cycle

Start

This is the initial state when a process is first started/created

Ready

The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run.

Running

Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Waiting

Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

Terminated or Exit: Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory

Introduction to Threads

- Modern OSs provide “virtual processors” within which programs execute.
- A programs *execution environment* is documented in the *process table* and assigned a *PID*.
- To achieve acceptable performance in distributed systems, relying on the OS’s idea of a process is often not enough – *finer granularity* is required.
- The solution: Threading.

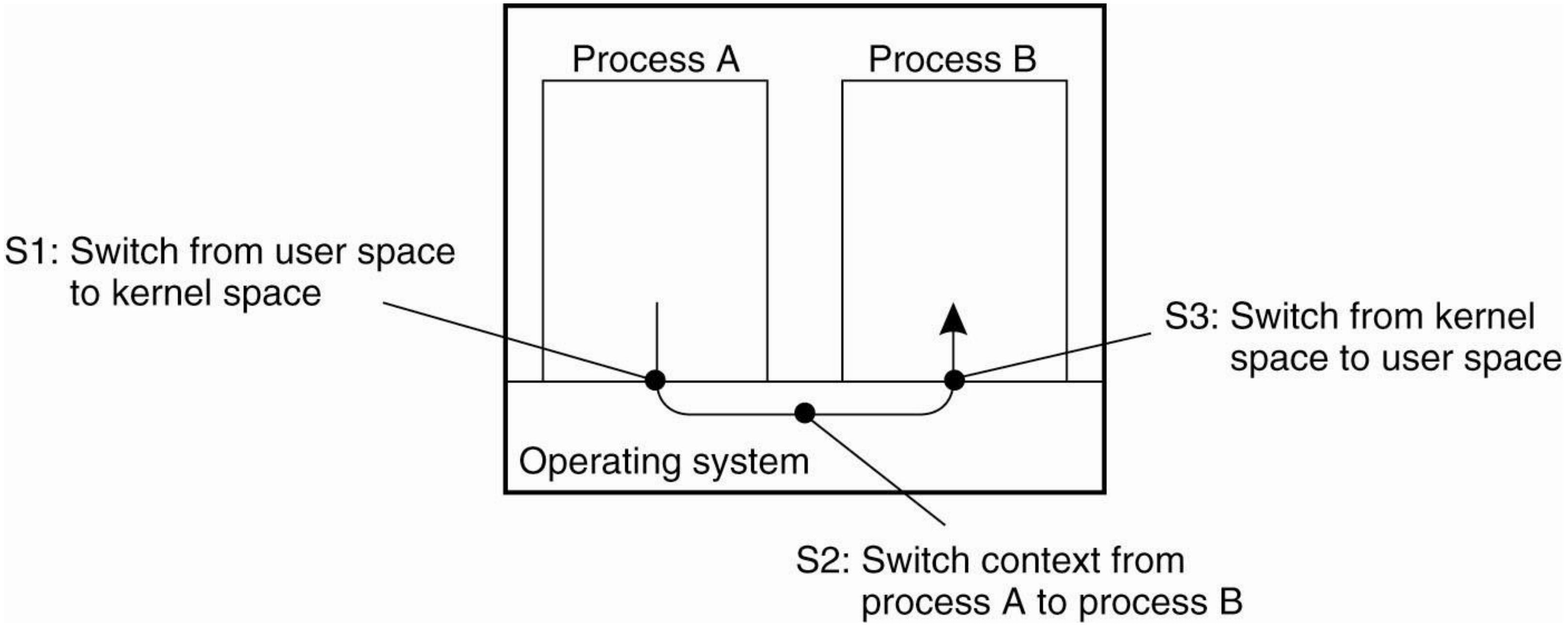
Problems with Processes

- Creating and managing processes is generally regarded as an *expensive* task (`fork` system call).
- Making sure all the processes peacefully co-exist on the system is not easy (as *concurrency transparency* comes at a price).
- **Threads** can be thought of as an "execution of a part of a program (in user-space)".
- Rather than make the OS responsible for concurrency transparency, it is left to the *individual application* to manage the creation and scheduling of each thread.

Important Implications

- Two Important Implications:
  1. Threaded applications often run faster than non-threaded applications (as context-switches between kernel and user-space are avoided).
  2. Threaded applications are harder to develop (although simple, clean designs can help here).
- Additionally, the assumption is that the development environment provides a Threads Library for developers to use (most modern environments do).

Thread Usage in Non-distributed Systems

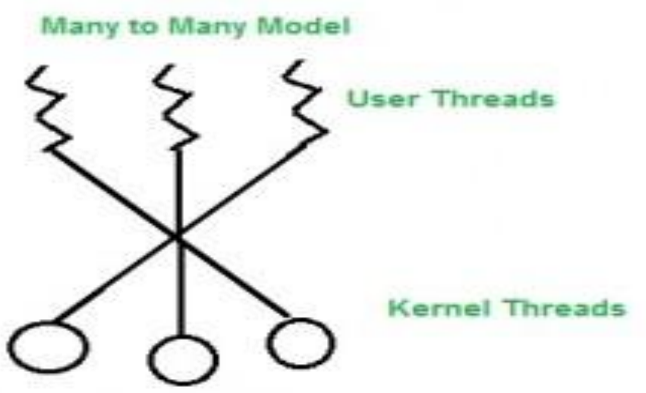

Process A
Process B
S1: Switch from user space to kernel space
S3: Switch from kernel space to user space
Operating system
S2: Switch context from process A to process B

Context switching as the result of IPC

Thread model

Many to Many Model

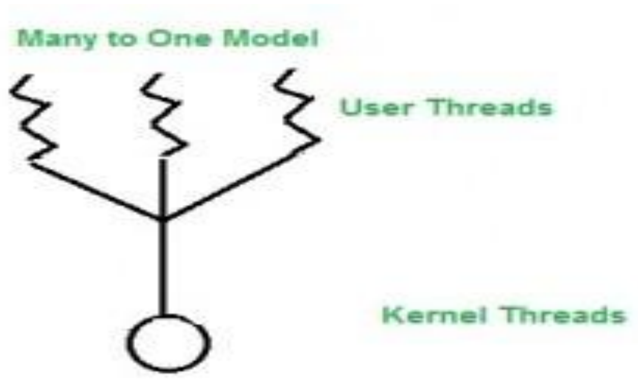
- multiple user threads multiplex to same or lesser number of kernel level threads
- advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread. Thus, System doesn't block if a particular thread is blocked.



Thread model

Many to One Model

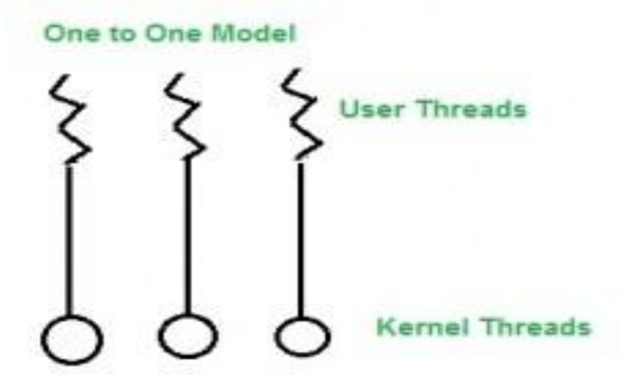
- multiple user threads mapped to one kernel thread.
- In this model when a user thread makes a blocking system call entire process blocks.
- multiple threads are not able access multiprocessor at the same time.



Thread model

One to One Model

- one to one relationship between kernel and user thread
- In this model multiple thread can run on multiple processor. Problem with this model is that creating a user thread requires the corresponding kernel thread.



differences

mparison Basis	Process	Thread
Definition	A process is a program under execution i.e .an active program	A thread is a lightweight process that can be .managed independently by a scheduler
Context switching time	Processes require more time for context .switching as they are more heavy	Threads require less time for context .switching as they are lighter than processes
Memory Sharing	Processes are totally independent and don't .share memory	A thread may share some memory with its .peer threads
Communication	Communication between processes requires .more time than between threads	Communication between threads requires . less time than between processes
Blocked	If a process gets blocked, remaining .processes can continue execution	If a user level thread gets blocked, all of its .peer threads also get blocked
Resource Consumption	Processes require more resources than .threads	Threads generally need less resources than .processes
Dependency	Individual processes are independent of .each other	Threads are parts of a process and so are .dependent
Data and Code sharing	Processes have independent data and code .segments	A thread shares the data segment, code .segment, files etc. with its peer threads
Treatment by OS	All the different processes are treated .separately by the operating system	All user level peer threads are treated as a .single task by the operating system
Time for creation	.Processes require more time for creation	.Threads require less time for creation
Time for termination	.Processes require more time for termination	.Threads require less time for termination

Threads in Non-Distributed Systems

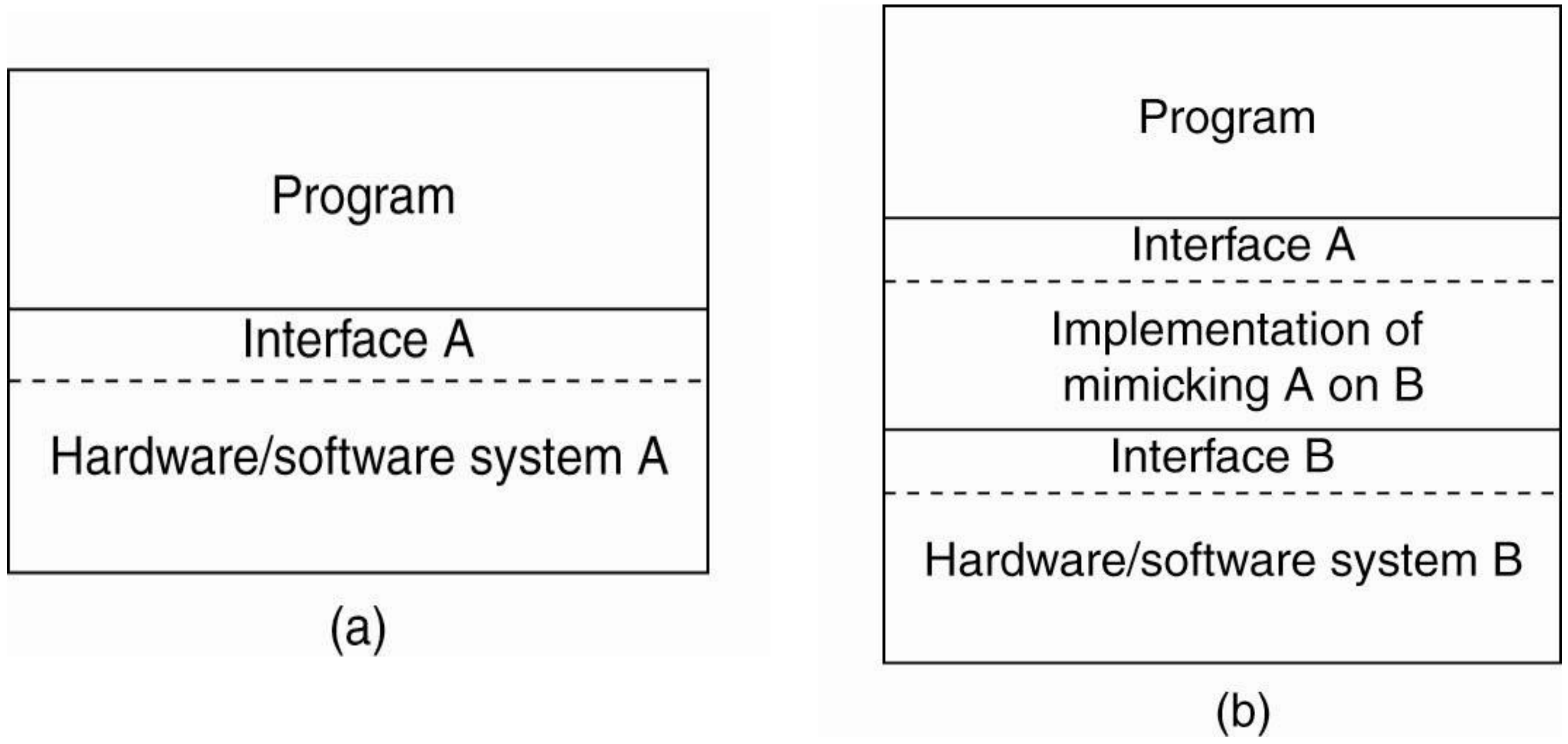
- Advantages:

1. Blocking can be *avoided*
2. Excellent support for multi-processor systems (each running their own thread).
3. Expensive context-switches can be avoided.
4. For certain classes of application, the design and implementation is made considerably easier.

Threads in Distributed Systems

- Important characteristic: a *blocking call* in a thread does not result in the entire process being blocked.
- This leads to the key characteristic of threads within distributed systems:
  - "We can now express communications in the form of maintaining multiple logical connections at the same time (as opposed to a single, sequential, blocking process)."

The Role of Virtualization in Distributed Systems

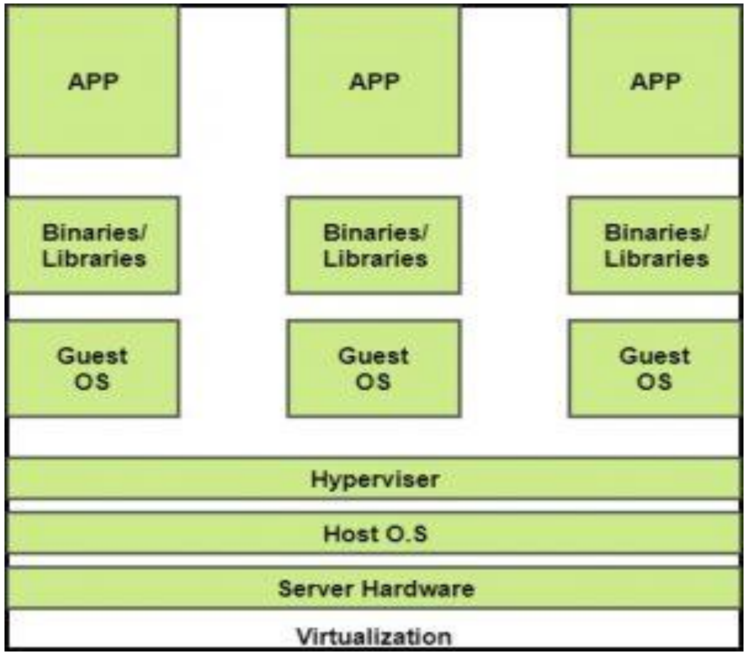

Program
Interface A
Hardware/software system A
(a)
Program
Interface A
Implementation of mimicking A on B
Interface B
Hardware/software system B
(b)

(a) General organization between a program, interface, and system

(b) General organization of virtualizing system A on top of system B

Virtualization in Distributed Systems

one of the main cost effective, hardware reducing, and energy saving techniques used by cloud providers is virtualization. Virtualization allows to share a single physical instance of a resource or an application among multiple customers and organizations at one time.



Virtualization in Distributed Systems

With the help of Virtualization, multiple operating systems and applications can run on same machine and its same hardware at the same time, increasing the utilization and flexibility of hardware.

Types of Virtualization:

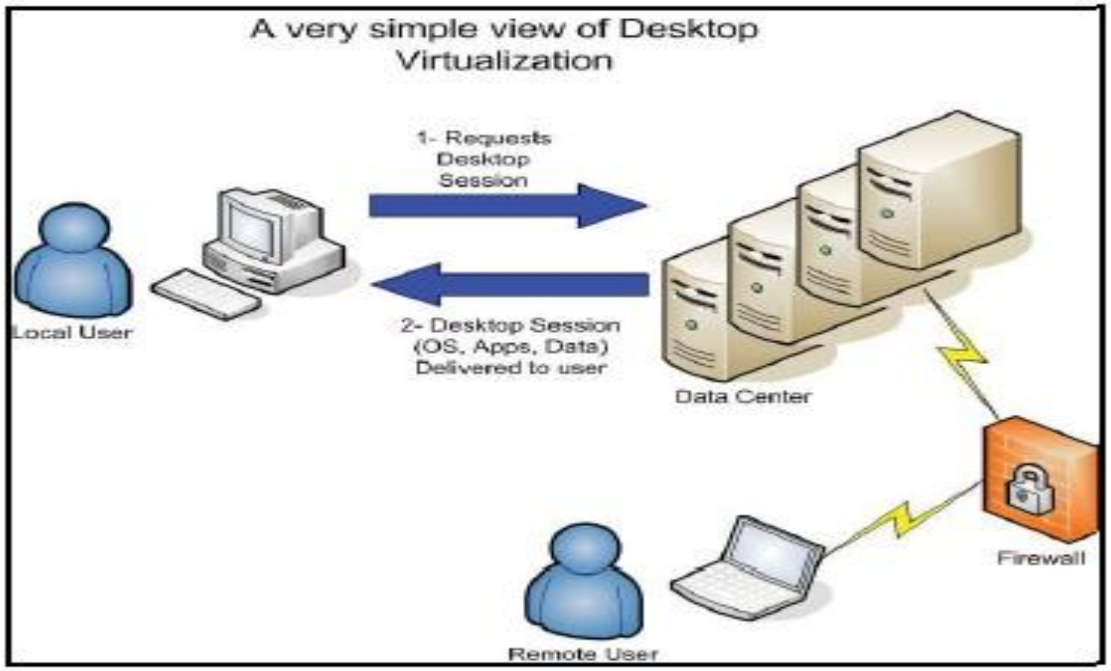
- Desktop virtualization
- Application virtualization
- Server virtualization
- Network virtualization
- Storage virtualization

Architectures of Virtual Machines (1)

- There are interfaces at different levels.
- An interface between the hardware and software, consisting of machine instructions
  - that can be invoked by any program.
- An interface between the hardware and software, consisting of machine instructions
  - that can be invoked only by privileged programs, such as an operating system.

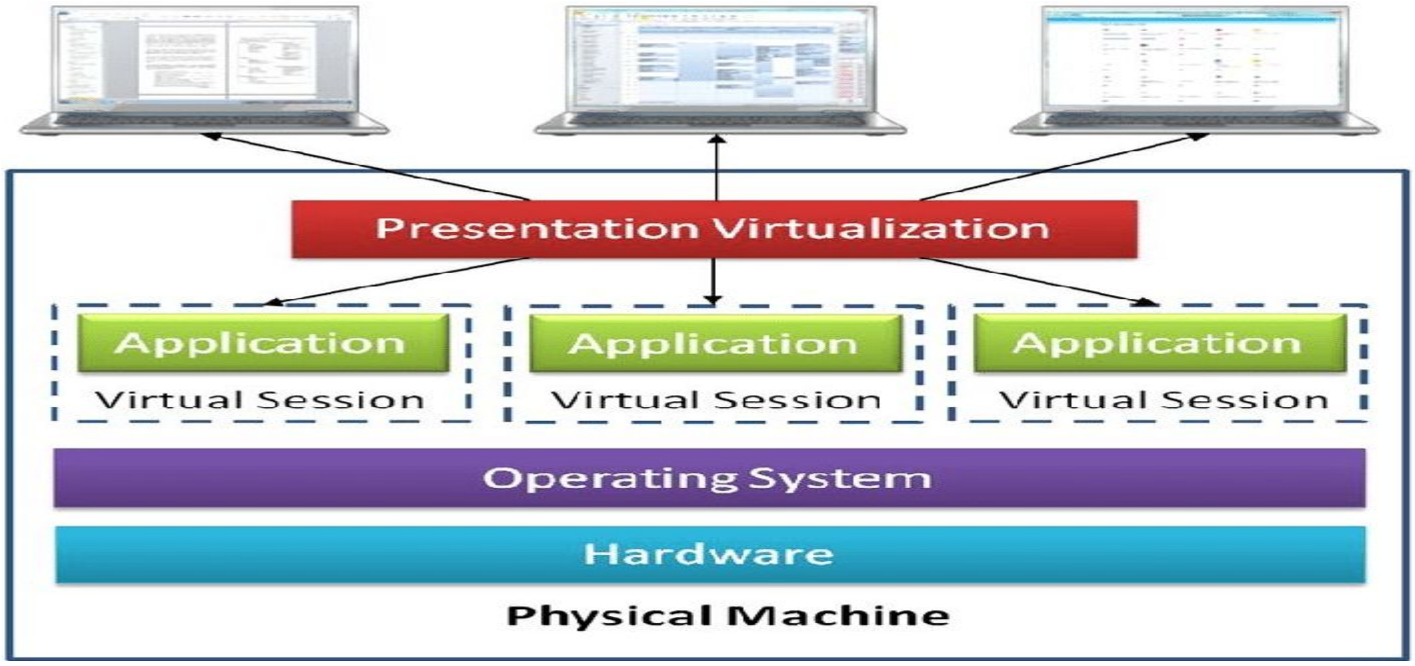
Desktop Virtualization

- Desktop virtualization allows the users' OS to be remotely stored on a server in the data centre. It allows the user to access their desktop virtually, from any location by a different machine.
- easy management of software installation, updates


A very simple view of Desktop Virtualization
1- Requests Desktop Session
2- Desktop Session (OS, Apps, Data) Delivered to user
Local User
Data Center
Firewall
Remote User

Application Virtualization

It is abstracting the application layer away from the operating system. This way, the application can run in an encapsulated form without being depended upon on by the operating system underneath. This can allow a Windows application to run on Linux and vice versa, in addition to adding a level of isolation.



Server Virtualization

- This is a kind of virtualization in which masking of server resources takes place. Here, the central-server(physical server) is divided into multiple different virtual servers by changing the identity number, processors. So, each system can operate its own operating systems in isolate manner. Where each sub-server knows the identity of the central server.
- It causes an increase in the performance and reduces the operating cost by the deployment of main server resources into a sub-server resource. It's beneficial in virtual migration, reduce energy consumption, reduce infrastructural cost, etc.

Network Virtualization

- The ability to run multiple virtual networks with each has a separate control and data plan. It co-exists together on top of one physical network. It can be managed by individual parties that potentially confidential to each other.
- Network virtualization provides a facility to create and provision virtual networks—logical switches, routers, firewalls, load balancer, Virtual Private Network (VPN), and workload security within days or even in weeks

Storage Virtualization

- Storage virtualization is an array of servers that are managed by a virtual storage system. The servers aren’t aware of exactly where their data is stored, and instead function more like worker bees in a hive.
- It makes managing storage from multiple sources to be managed and utilized as a single repository. storage virtualization software maintains smooth operations, consistent performance and a continuous suite of advanced functions .

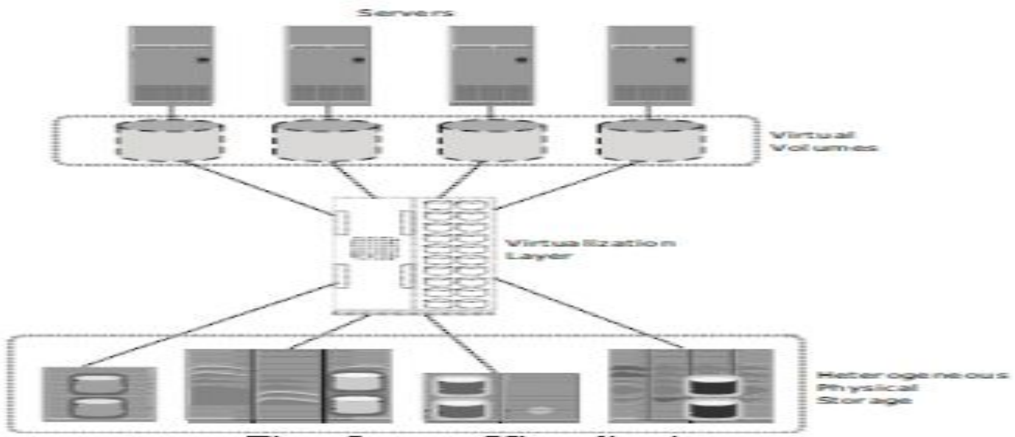
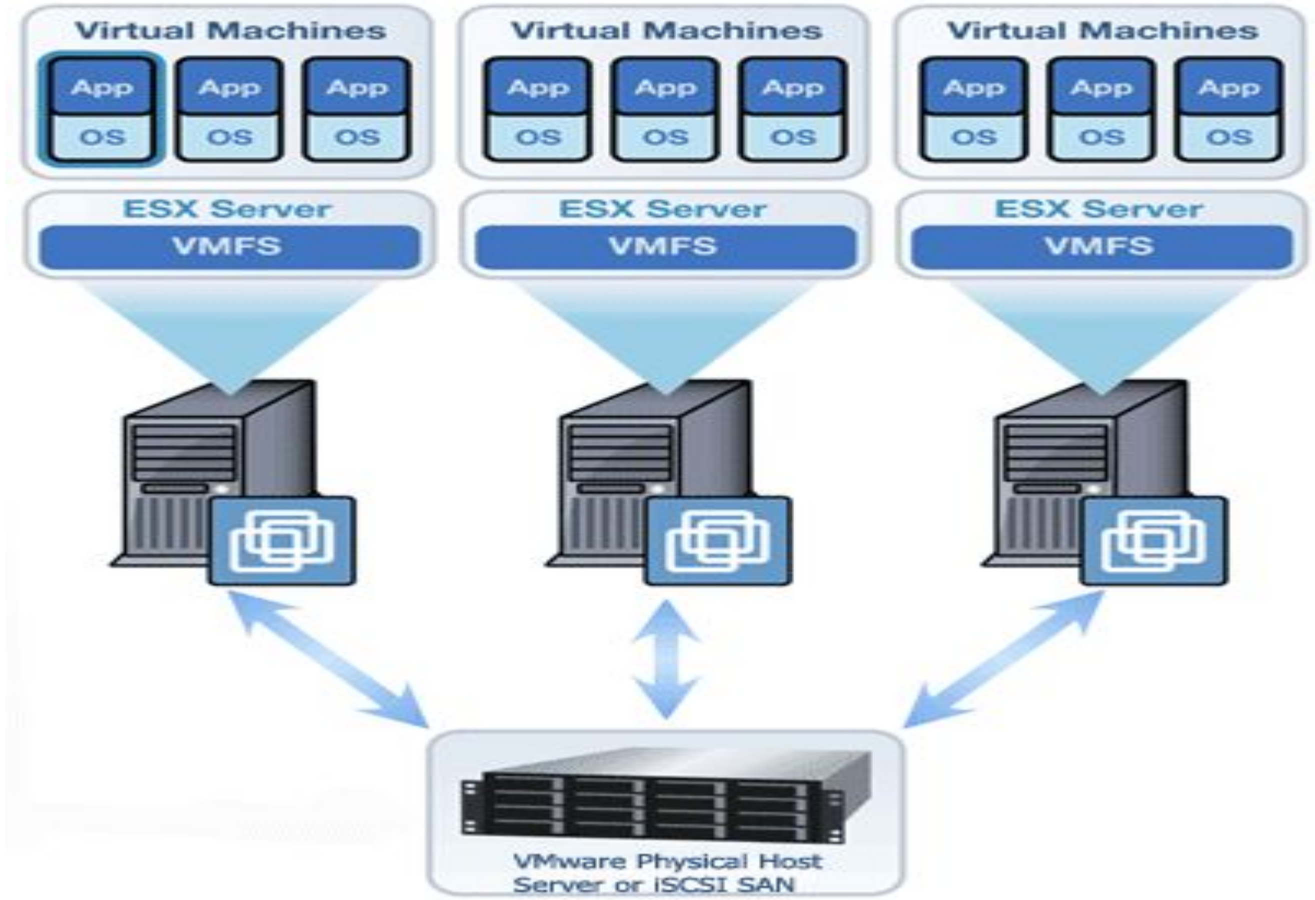


Fig a Storage Virtualization

Server Virtualization



Advantages of virtualization

- **Lower costs.** Virtualization reduces the amount of hardware servers necessary within a company and data center. This lowers the overall cost of buying and maintaining large amounts of hardware.
- **Easier disaster recovery.** It is very simple in a virtualized environment. Regular snapshots provide up-to-date data, allowing virtual machines to be feasibly backed up and recovered. Even in an emergency, a virtual machine can be migrated to a new location within minutes.
- **Easier testing.** Testing is less complicated in a virtual environment. Even if a large mistake is made, the test does not need to stop and go back to the beginning. It can simply return to the previous snapshot and proceed with the test.

Advantages of virtualization

- **Quicker backups.** Backups can be taken of both the virtual server and the virtual machine. Automatic snapshots are taken throughout the day to guarantee that all data is up-to-date. Furthermore, the virtual machines can be easily migrated between each other and efficiently redeployed.

- **Improved productivity.** Fewer physical resources result in less time spent managing and maintaining the servers. Tasks that can take days or weeks in a physical environment can be done in minutes.
- It provides energy saving

disadvantages of virtualization

- **Implementation:**

Although it is mentioned that the virtualization is highly cost effective still it needs more investment when it comes to implementation.some instance the hardwares and softwares are required which means that devices needs to be purchased to make the virtualization possible.

- **Limitations:**

Every server and application out there is not virtualization compatible. Hence, some of the IT infrastructure of the organizations will not be supporting the virtualized solutions.

- **Security:** Data security is often questioned in a virtualized environment since the server is managed by managed by the third party providers. Therefore, it is important to choose the virtualization solution wisely so that it can provide adequate protection

disadvantages of virtualization

- **Availability:** the data needs to be connected for a prolonged period of time. If not the organization will be going to loose the competition in the industry. The issue with the availability can come from the virtualization servers. The virtualization servers has the tendency to go offline.

Clients

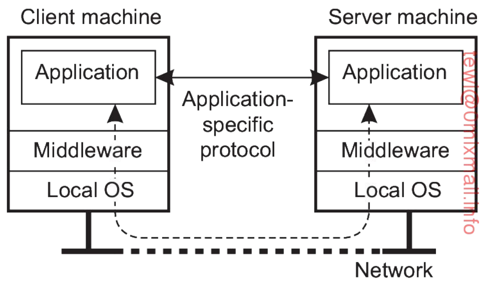
The client is the machine running the front-end applications. It interacts with a user through the keyboard, display. The client has no direct data access responsibilities. It simply requests processes from the server and displays data managed by the server. The client workstation can be optimized for its job. For example, it might not need large disk capacity, or it might benefit from graphic capabilities.

Networked user interfaces

- A major task of client machines is to provide the means for users to interact with remote servers.
- for each remote service the client machine will have a separate counterpart that can contact the service over the network.
- There are roughly two ways in which this interaction can be supported

Clients

- First, for each remote service the client machine will have a separate counterpart that can contact the service over the network. A typical example is a calendar running on a user's smartphone that needs to synchro- nize with a remote, possibly shared calendar.



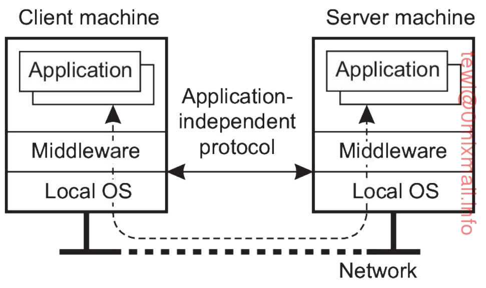


Figure : (a) A networked application with its own protocol. (b) A general solution to allow access to remote applications.

Clients

- A second solution is to provide direct access to remote services by offer-ing only a convenient user interface.
- Effectively, this means that the client machine is used only as a terminal with no need for local storage.
- networked user interfaces, everything is processed and stored at the server.
- This **thin-client approach** has received much attention with the increase of Internet connectivity and the use of mobile devices. Thin-client solutions are also popular as they ease the task of system management.

Client-side software for distribution transparency

- Client software comprises more than just user interfaces. In many cases, parts of the processing and data level in a client-server application are executed on the client side as well.
- application-related software, client soft- ware comprises components for achieving distribution transparency. Ideally, a client should not be aware that it is communicating with remote processes. In contrast, distribution is often less transparent to servers for reasons of performance and correctness.
- Access transparency is generally handled through the generation of a **client stub** from an interface definition of what the server has to offer. The stub provides the same interface as the one available at the server, but hides the possible differences in machine architectures, as well as the actual commu- nication.

Servers

- A server is a piece of computer hardware or software that provides functionality for other programs or deives,called clients.
- Server can provide various functionalities ,often called services, such as data or resources among multiple clients,and a single client can use multiple servers.
- Typically servers are database servers, file servers,mail servers,web servers, game servers.

General server design issues in DS

- A server is a process implementing a specific service on behalf of a collection of clients. In essence, each server is organized in the same way: it waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.

Concurrent versus iterative servers

- In the case of an **iterative server**, the server itself handles the request and, if necessary, returns a response to the requesting client.
- This is single thread server .
- While processing the current request it adds incoming request in a waiting queue and once the processing is done ,it takes in the next request in queue.
- A **concurrent server** does not handle the request itself, but passes it to a separate thread or another process, after which it immediately waits for the next incoming request.
- A multithreaded server is an example of a concurrent server. An alternative implementation of a concurrent server is to fork a new process for each new incoming request

General server design issues in DS

Contacting a server end points

the client can determine the server it need to communicates using one of the two techniques.

•Hard code the port number of the server in the client.If the server moves then client will be recompiled.

•A directory service is used to register a service with a port number and IP address.The client connects to the directory service to determine the location of the server with particular service.

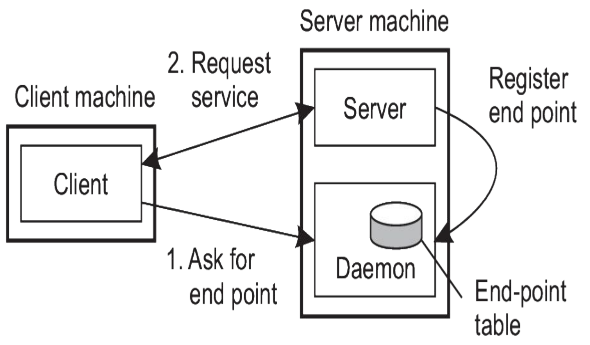
•HTTP server always listen to TCP port 80.

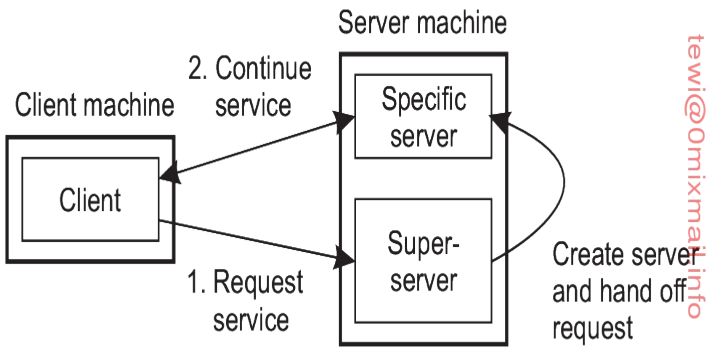
General server design issues in DS

Contacting a server end points

There are many services that do not require a preassigned end point. For example, a time-of-day server may use an end point that is dynamically assigned to it by its local operating system.

One solution is to have a special daemon running on each machine that runs servers. The daemon keeps track of the current end point of each service implemented by a co-located server. The daemon itself listens to a well-known end point. A client will first contact the daemon, request the end point, and then contact the specific server.





More on Servers

- What's a server?
- *Definition*: "A process that implements a specific service **on behalf of** a collection of clients".
- Typically, servers are organized to do one of two things:
  1. Wait
  2. Service

… wait … service … wait … service … wait …

Servers: Iterative and Concurrent

- *Iterative*: server handles request, then returns results to the client; any new client requests *must wait* for previous request to complete (also useful to think of this type of server as *sequential*).
- *Concurrent*: server does not handle the request itself; a separate thread or sub-process handles the request and returns any results to the client; the server is then free to immediately service the next client (i.e., there's no waiting, as service requests are processed in *parallel*).

”Server “States

- **Stateless servers** – no information is maintained on the current “connections” to the server. The Web is the classic example of a *stateless service*. As can be imagined, this type of server is **easy** to implement.
- **Stateful servers** – information is maintained on the current “connections” to the server. Advanced file servers, where copies of a file can be updated “locally”, then applied to the main server (as it knows the state of things) - more **difficult** to implement.
- But, what happens if something *crashes*? (more on this later).

Interrupting a server

- Another issue that needs to be taken into account when designing a server is whether and how a server can be interrupted.
- For example, consider a user who has just decided to upload a huge file to an FTP server. Then, suddenly realizing that it is the wrong file, he wants to interrupt the server to cancel further data transmission.
- There are several ways to do this.

One approach that works only too well in the current Internet (and is sometimes the only alternative) is for the user to abruptly exit the client application (which will automatically break the connection to the server), immediately restart it, and pretend nothing happened.

Interrupting a server

- which is data that is to be processed by the server before any other data from that client. One solution is to let the server listen to a separate control end point to which the client sends out-of-band data.

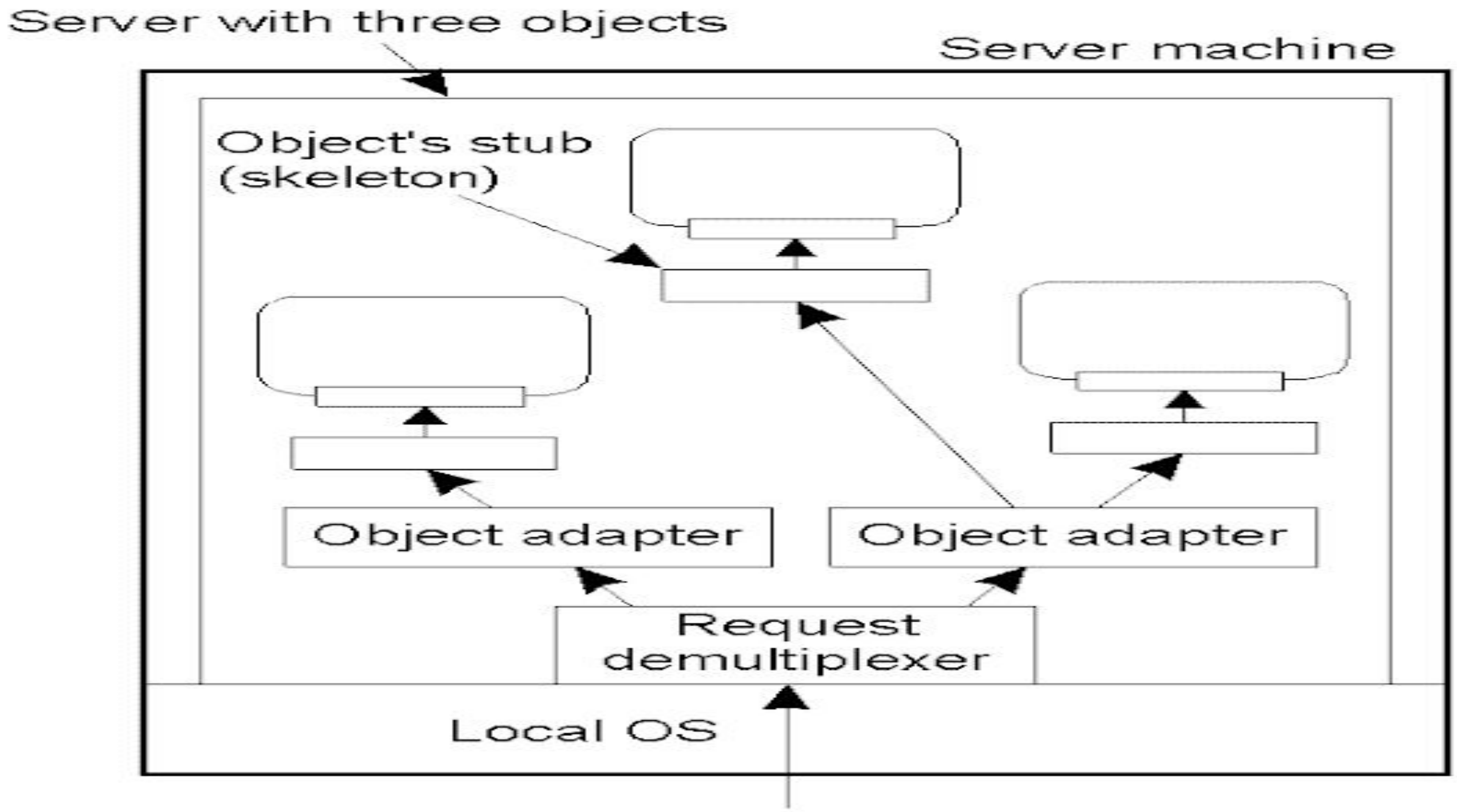
?”Problem: Identifying “end-points

- How do clients know which end-point (or port) to contact a server at? How do they “bind” to a server?
  - Statically assigned end-points (IANA).
  - Dynamically assigned end-points (DCE).
  - A popular variation:
    - the “super-server” (inetd on UNIX).

A Special Type: Object Servers

- A server tailored to support distributed objects.
- Does not provide a specific service.
- Provides a facility whereby objects can be remotely invoked by non-local clients.
- Consequently, object servers are highly adaptable.
- *"A place where objects live"*.

Object Adapter



Organization of an object server supporting different activation policies.

Reasons for Migrating Code

- Why? Biggest single reason: **better performance**.
- The big idea is to move a compute-intensive task from a *heavily loaded* machine to a *lightly loaded* machine
  "on demand" and "as required".

Code Migration Examples

- *Moving (part of) a client to a server –* processing data close to where the data resides. It is often too expensive to transport an entire database to a client for processing, so move the client to the data.
- *Moving (part of) a server to a client –* checking data prior to submitting it to a server. The use of local error-checking (using JavaScript) on Web forms is a good example of this type of processing. Error-check the data close to the user, not at the server.

Classic" Code Migration Example"

- Searching the Web by "roaming".
- Rather than search and index the Web by requesting the transfer of each and every document to the client for processing, the client relocates to each site and indexes the documents it finds "in situ". The index is then transported from site to site, in addition to the executing process.

Major Disadvantage

- **Security Concerns**.
- "Blindly trusting that the downloaded code implements only the advertised interface while accessing your unprotected hard-disk and does not send the juiciest parts to heaven-knows-where may not always be such a good idea".

Code Migration Models

- A running process consists of three “segments”:

1. *Code* – instructions.
2. *Resource* – external references.
3. *Execution* – current state.

Migration in Heterogeneous Systems

- Three ways to handle migration (which can be combined):

1. Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
2. Stopping the current virtual machine; migrate memory, and start the new virtual machine.
3. Letting the new virtual machine pull in new pages as needed, that is, let processes start on the new virtual machine immediately and copy memory pages on demand.