Parallel & Distributed System

Virtualization & Client-server Interaction

Md. Biplob Hosen

Lecturer, IIT-JU

Email: biplob.hosen@juniv.edu

Contents

- **Virtualization**
  - ✓ Types of Virtualization
  - ✓ Application of Virtual Machines to Distributed Systems
- **Clients**
  - ✓ Networked User Interfaces for Clients
  - ✓ Client-side Software for Distribution Transparency

- Reference Books
  - ✓ Distributed Systems: Principles and Paradigms, 3rd Edition by Andrew S. Tanenbaum & Maarten van Steen, Publisher: Pearson Prentice Hall **[CH-03]**.

Types of Virtualization

- To understand the differences in virtualization, it is important to realize that computer systems generally offer four different types of interfaces, at three different levels:

1. An interface between the hardware and software, referred to as the instruction set architecture (ISA), forming the set of machine instructions. This set is divided into two subsets:
   - Privileged instructions, which are allowed to be executed only by the operating system.
   - General instructions, which can be executed by any program.
2. An interface consisting of system calls as offered by an operating system.
3. An interface consisting of library calls, generally forming what is known as an application programming interface (API). In many cases, the aforementioned system calls are hidden by an API.

Continue…

- These different types are shown in the following figure.
- The essence of virtualization is to mimic the behavior of these interfaces.

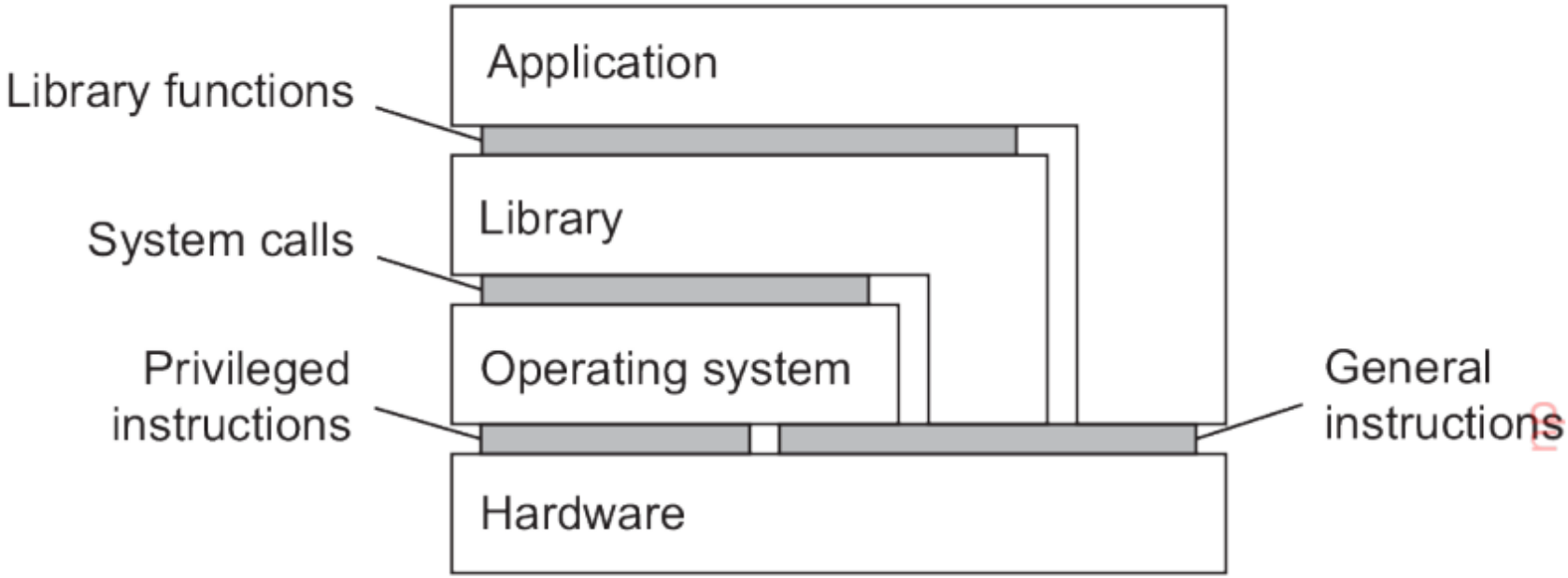
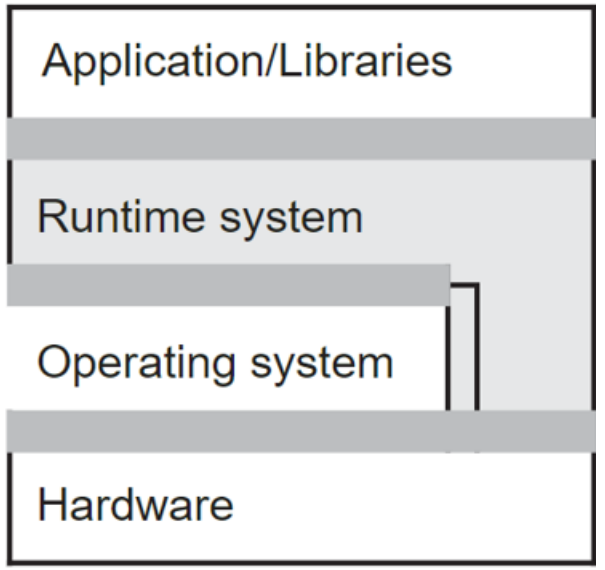


Figure : Various interfaces offered by computer systems

Continue…

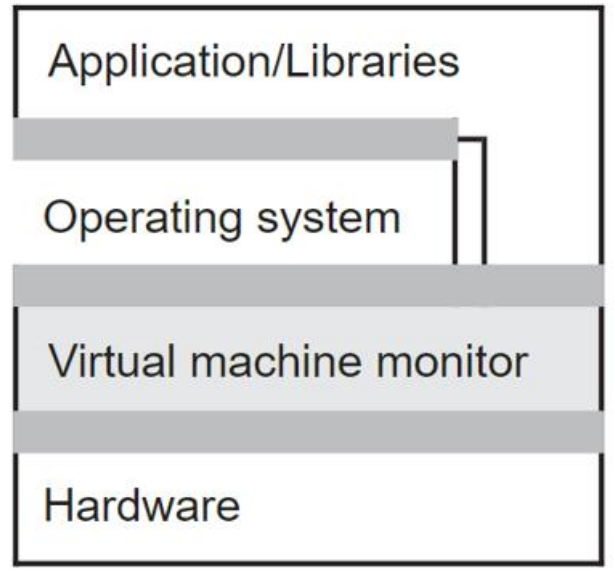
- Virtualization can take place in different ways.
- First, we can build a runtime system that essentially provides an abstract instruction set that is to be used for executing applications.
- Instructions can be interpreted (as is the case for the Java runtime environment), but could also be emulated as is done for running Windows applications on Unix platforms.
- This type of virtualization, shown in Figure-(a), leads to all a **process virtual machine**, stressing that virtualization is only for a single process.



(a) A process virtual machine

Continue…

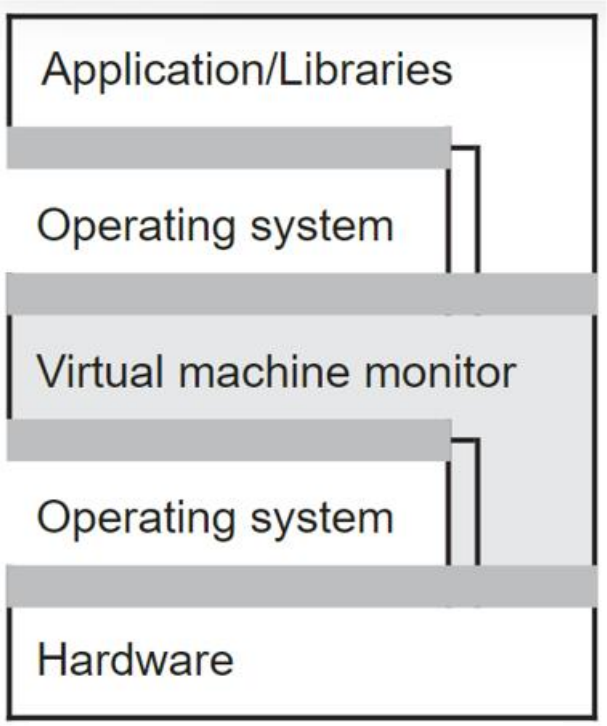
- An alternative approach toward virtualization, shown in Figure-(b), is to provide a system that is implemented as a layer shielding the original hardware, but offering the complete instruction set of that same as an interface.
- This leads to what is known as a **native virtual machine monitor**. It is called native because it is implemented directly on top of the underlying hardware.
- Note that the interface offered by a virtual machine monitor can be offered simultaneously to different programs.
- As a result, it is now possible to have multiple, and different guest operating systems run independently and concurrently on the same platform.



(b) A native virtual machine monitor

Continue…

- A native virtual machine monitor will have to provide and regulate access to various resources, like external storage and networks.
- Like any operating system, this implies that it will have to implement device drivers for those resources.
- Rather than doing all this effort anew, **a hosted virtual machine monitor** will run on top of a trusted host operating system as shown in Figure-(c).
- In this case, the virtual machine monitor can make use of existing facilities provided by that host operating system.
- Using a hosted virtual machine monitor is highly popular in modern distributed systems such as data centers and clouds.



(c) A hosted virtual machine monitor

Application of Virtual Machines to Distributed Systems

- From the perspective of distributed systems, the most important application of virtualization lies in **cloud computing**.
- Cloud providers offer roughly three different types of services:
  - Infrastructure-as-a-Service (IaaS) covering the basic infrastructure
  - Platform-as-a-Service (PaaS) covering system-level services
  - Software-as-a-Service (SaaS) containing actual applications
- Virtualization plays a key role in IaaS. Instead of renting out a physical machine, a cloud provider will rent out a virtual machine (monitor) that may be sharing a physical machine with other customers.
- The beauty of virtualization is that it allows for almost complete isolation between customers, who will indeed have the illusion that they have just rented a dedicated physical machine.

Continue…

- To make matters concrete, let us consider **the Amazon Elastic Compute Cloud,** or simply **EC2**.
- EC2 allows one to create an environment consisting of several networked virtual servers, thus forming basis of a distributed system.
- To make it easy, there is a number of preconfigured machine images available, referred to as **Amazon Machine Images,** or simply **AMIs**.
- An AMI is an installable software package consisting of an operating-system kernel along with a number of services.
- An example of a simple, basic AMI is a **LAMP image**, consisting of a Linux kernel, the Apache Web server, a MySQL database system, and PHP libraries.
- More elaborate images containing additional software are also available, as well as images based on other Unix kernels or Windows.
- In this sense, an AMI is essentially the same as a boot disk.

Continue…

- An EC2 customer needs to select an AMI, possibly after adapting or configuring one.
- An AMI can then be launched resulting in what is called an **EC2 instance**: the actual virtual machine that can be used to host a customer's applications.
- An important issue is that a customer will hardly ever know exactly where an instance is actually being executed.
- Obviously, it is running on a single physical machine, but where that machine is located remains hidden.
- The closest one can get to pinpointing the location where an instance should run is by selecting one of a few regions provided by Amazon (US, South America, Europe, Asia).

Continue…

- To communicate, each instance obtains two IP addresses: a **private** one that can be used for internal communication between different instances, making use of EC2's internal networking facilities, and a **public IP address** allowing any Internet clients to contact an instance.
- The public address is mapped to the private one using standard **network-address translation (NAT)** technology.
- A simple way to manage an instance is to make use of an **SSH connection**, for which Amazon provides the means for generating the appropriate keys.
- The EC2 environment in which an instance is executed provides different levels of the following services:
  - CPU: allows to select the number and type of cores, including GPUs.
  - Memory: defines how much main memory is allocated to an instance.
  - Storage: defines how much local storage is allocated.
  - Platform: distinguishes between 32-bit or 64-bit architectures.
  - Networking: sets the bandwidth capacity that can be used.

Continue…

- In addition, extra resources can be requested such as an additional networking interface.
- The local storage that comes with an instance is transient: when the instance stops, all the data stored locally is lost.
- In order to prevent data loss, a customer will need to explicitly save data to persistent store, for example, by making use of **Amazon's Simple Storage Service (S3)**.
- An alternative is to attach a storage device that is mapped to **Amazon's Elastic Block Store (Amazon EBS)**.

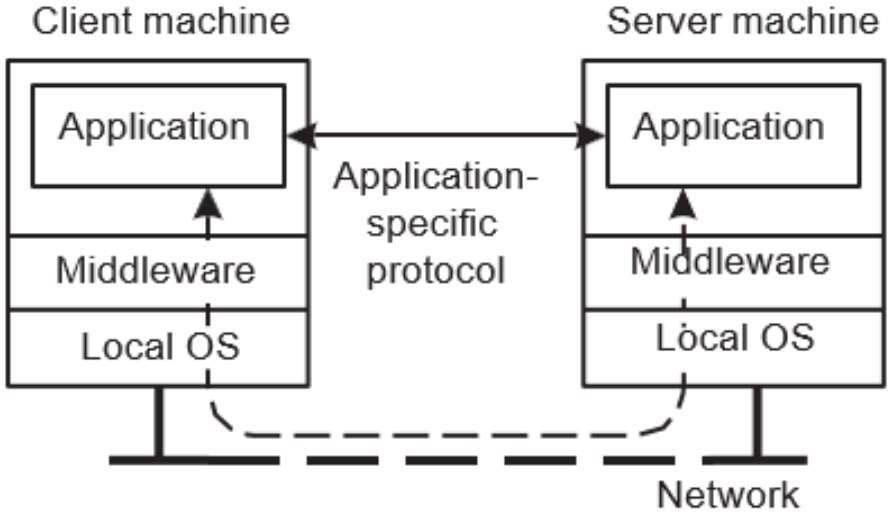
Continue…

- The IaaS as offered by EC2 allows a customer to create a number of virtual machines, each configured with resources as needed, and capable of exchanging messages through an IP network.
- In addition, these virtual machines can be accessed from anywhere over the Internet (provided a client has the proper credentials).
- As such, Amazon EC2, like many other IaaS providers, offers the means to configure a complete distributed system consisting of networked virtual servers and running customer-supplied distributed applications.
- At the same time, those customers will not need to maintain any physical machine, which by itself is often already a huge gain.

Networked User Interfaces for Clients

- A major task of client machines is to provide the means for users to interact with remote servers.
- There are roughly two ways in which this interaction can be supported.
- 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 synchronize with a remote, possibly shared calendar.
- In this case, an application-level protocol will handle the synchronization.

Client-server interaction

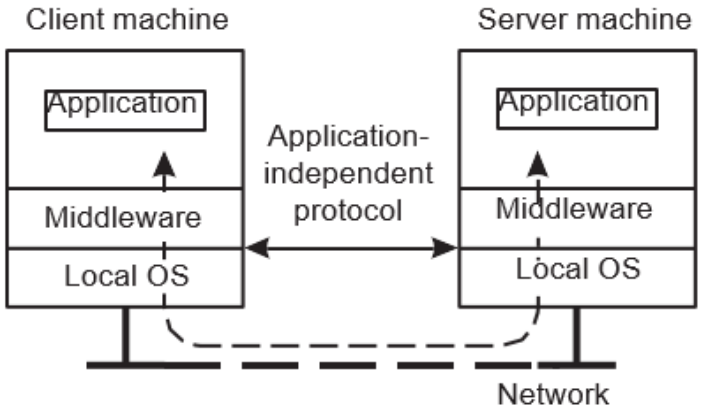


A networked application with its own protocol

Networked User Interfaces for Clients

- A second solution is to provide direct access to remote services by offering only a convenient user interface.
- Effectively, this means that the client machine is used only as a terminal with no need for local storage, leading to an application-neutral solution as shown in the figure.
- In the case of 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.

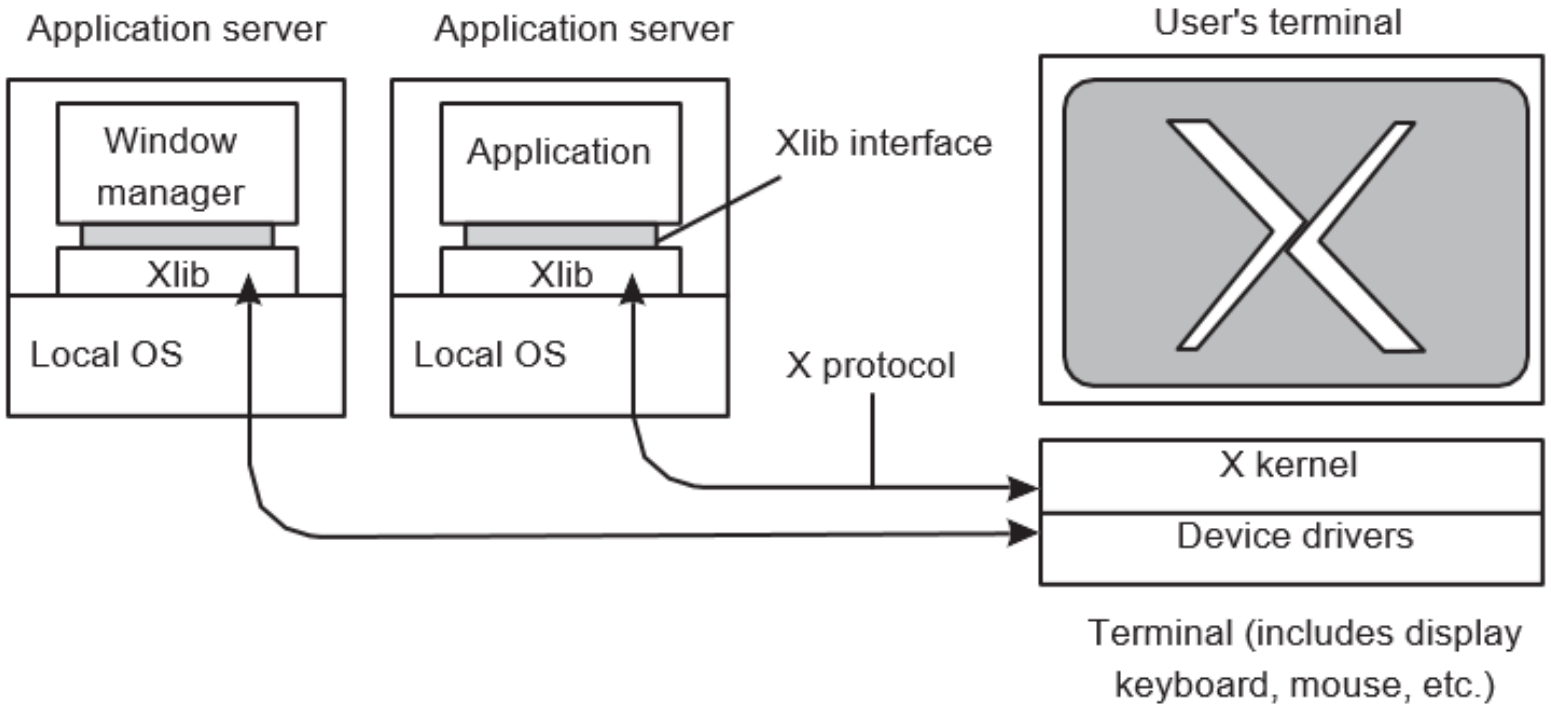
Client-server interaction



A general solution to allow access to remote applications

Example: The X Window system

Basic organization



X client and server

- The application acts as a client to the X-kernel, the latter running as a server on the client's machine.

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.
- A special class is formed by embedded client software, such as for automatic teller machines (ATMs), cash registers, barcode readers, TV set-top boxes, etc.
- In these cases, the user interface is a relatively small part of the client software, in contrast to the local processing and communication facilities.
- Besides, the user interface and other application-related software, client software comprises components for achieving distribution transparency.
- Ideally, a client should not be aware that it is communicating with remote processes.

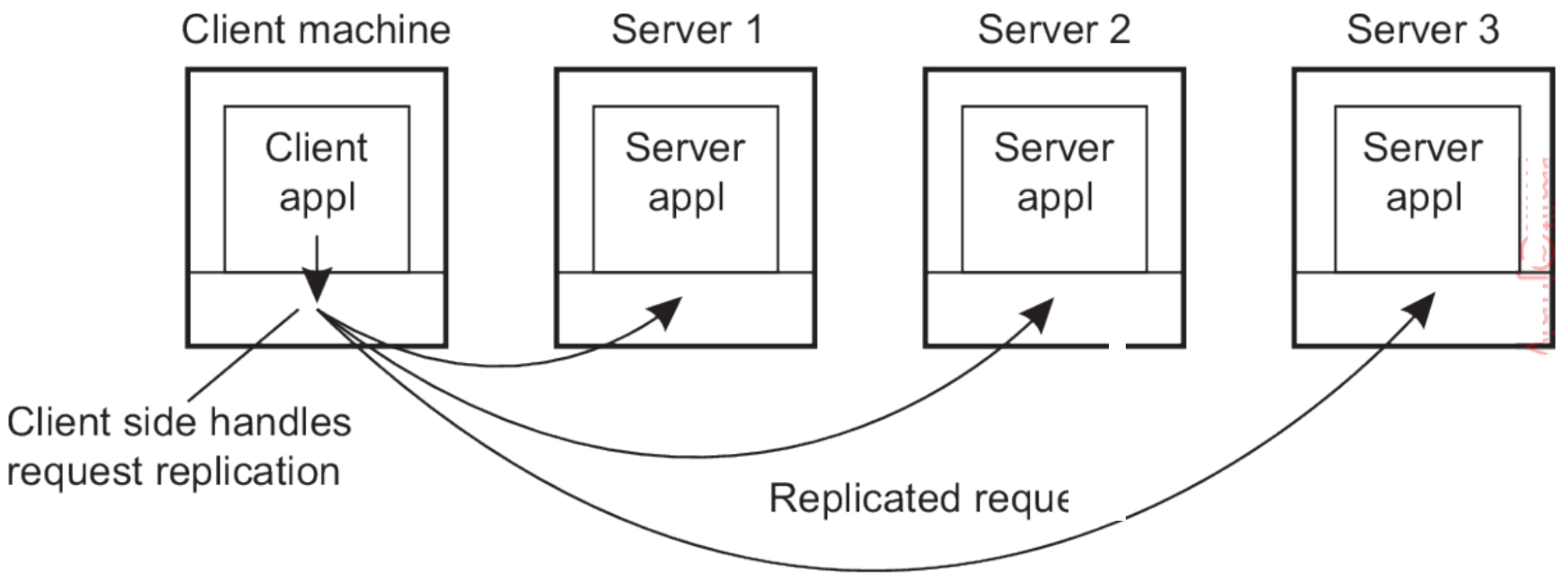
Continue…

- **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 communication.
- The client stub transforms local calls to messages that are sent to the server, and vice versa transforms messages from the server to return values as one would expect when calling an ordinary procedure.
- There are different ways to handle **location, migration, and relocation transparency**.
- Using a convenient naming system is crucial. In many cases, cooperation with client-side software is also important.

Continue…

- For example, when a client is already bound to a server, the client can be directly informed when the server changes location.
- In this case, the client's middleware can hide the server's current network location from the user, and also transparently rebind to the server if necessary. At worst, the client's application may notice a temporary loss of performance.
- In a similar way, many distributed systems implement **replication transparency** by means of client-side solutions.
- For example, imagine a distributed system with replicated servers, Such replication can be achieved by forwarding a request to each replica, as shown in the figure.
- Client-side software can transparently collect all responses and pass a single response to the client application.

Continue…



Transparent replication of a server using a client-side solution

- Regarding **failure transparency**, masking communication failures with a server is typically done through client middleware.
- For example, client middleware can be configured to repeatedly attempt to connect to a server, or perhaps try another server after several attempts.
- There are even situations in which the client middleware returns data it had cached during a previous session, as is sometimes done by Web browsers that fail to connect to a server.
- Finally, **concurrency transparency** can be handled through special intermediate servers, notably transaction monitors, and requires less support from client software.

Thank You ☺