**ASSIGNMENT**

**Course Outcome 1:**

1. Explain in detail the goals of distributed systems [8 M]

**Goals**

1. Connecting Users and Resources. 2. Transparency.

3. Openness. 4. Scalability.

**Connecting Users and Resources** (Making Resources Accessible)

1. Distributed system Make it easy for the users (and applications) to access remote resources

2. Distributed system to share them in a controlled and efficient way.

**Resources** - anything: printers, computers, storage facilities, data, files, Web pages, and networks, etc.

**Accessibility Issues**

1. Security. 2. Unwanted communication.

**Transparency**

Goal - hide the fact that its processes and resources are physically distributed across multiple computers systems should be transparent

**Different forms of transparency in a distributed system (ISO, 1995).**

|  |  |
| --- | --- |
| Transparency | Description |
| Access | Hide differences in data representation and how a resource is accessed |
| Location | Hide where a resource is located |
| Migration | Hide that a resource may move to another location |
| Relocation | Hide that a resource may be moved to another location while in use |
| Replication | Hide that a resource is replicated |
| Concurrency | Hide that a resource may be shared by several competitive users |
| Failure | Hide the failure and recovery of a resource |

**Degree of Transparency Issues**

**Timing**

e.g. requesting an electronic newspaper to appear in your mailbox before 7 A.M. local time, as usual, while you are currently at the other end of the world living in a different time zone.

**Synchronization**

e.g. a wide-area distributed system that connects a process in San Francisco to a process in Amsterdam limited by laws of physics - a message sent from one process to the other takes about 35 milliseconds.

· It takes several hundreds of milliseconds using a computer network.

· Signal transmission is not only limited by the speed of light, but also by limited processing capacities of the intermediate switches.

**Performance**

e.g. many Internet applications repeatedly try to contact a server before finally giving up. Consequently, attempting to mask a transient server failure before trying another one may slow down the system as a whole.

**Consistency**

e.g. need to guarantee that several replicas, located on different continents, need to be consistent all the time -  a single update operation may now even take seconds to complete, something that cannot be hidden from users.

**Context Awareness**

e.g. notion of location and context awareness is becoming increasingly important, it may be best to actually expose distribution rather than trying to hide it. -  consider an office worker who wants to print a file from her notebook computer. It is better to send the print job to a busy nearby printer, rather than to an idle one at corporate headquarters in a different country.

**Limits of Possibility**

Recognizing that full distribution transparency is simply impossible, we should ask ourselves whether it is even wise to pretend that we can achieve it.

**Openness**

Goal: offer services according to standard rules that describe the syntax and semantics of those services.

e.g.

1. Computer networks - standard rules govern the format, contents, and meaning of messages sent and received.

2. Distributed systems - services are specified through interfaces, which are often described in an

**Interface Definition Language (IDL)**

* Interface definitions written in an IDL nearly always capture only the syntax of services

3. Specify names of the available functions with types of parameters; return values, possible exceptions that can be raised, etc.

4. Allows an arbitrary process that needs a certain interface to talk to another process that provides that interface

5. Allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate in exactly the same way.

**Properties of specifications**

**Complete** - everything that is necessary to make an implementation has been specified.

**Neutral**

Specifications do not prescribe what an implementation should look like Lead to:

**Interoperability** - characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.

**Portability** characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.

**Goals**: an open distributed system should also be extensible. i.e.

1. be easy to configure the system out of different components (possibly from different developers).

2. be easy to add new components or replace existing ones without affecting those components that stay in place.

**Scalability**

 Scalability of a system is measured with respect to:

1. Size - can easily add more users and resources to the system.

2. Geographic extent - a geographically scalable system is one in which the users and resources may lie far apart.

3. Administrative scalability - can be easy to manage even if it spans many independent administrative organizations.

**Scalability Limitations of Size**

|  |  |
| --- | --- |
| **Concept** | **Example** |
| Centralized services | A single server for all users |
| Centralized data | A single on-line telephone book |
| Centralized algorithms | Doing routing based on complete information |

<http://csis.pace.edu/~marchese/CS865/Lectures/Chap1/Chapter1a.htm>

**Geographical scalability Limitations**

**Synchronization**

 e.g. currently hard to scale existing distributed systems designed for local-area networks is that they are based on synchronous communication.

1. A client requesting service blocks until a reply is sent back.

2. Works fine in LANs where communication between two machines is generally at worst a few hundred microseconds.

3. In a wide-area system, inter process communication may be hundreds of milliseconds, three orders of magnitude slower.

**Unreliability of communication**

1.  Communication in wide-area networks is inherently unreliable  and point-to-point.

2. local-area networks provide reliable communication based on broadcasting, making it much easier to develop distributed systems. For example, consider the problem of locating a service.

a. e.g. in a local-area system, a process can broadcast a message to every machine, asking if it is running the service it needs.

b. Only those machines that have that service respond, each providing its network address in the reply message.

c. Such a location scheme is unthinkable in a wide-area system: just imagine what would happen if we tried to locate a service this way in the Internet.

**Administrative scalability**

1.  How to scale a distributed system across multiple, independent administrative domains.

a. Major problem - conflicting policies with respect to resource usage (and payment), management, and security.

**Scaling Techniques**

 Three techniques for scaling:

1. Hiding communication latencies. 2. Distribution. 3. Replication.

**Hiding communication latencies** - important to achieving geographical scalability.

1. Try to avoid waiting for responses to remote service requests.

e.g, when a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the requester's side.

construct the requesting application in such a way that it uses only asynchronous communication. ]

2. Reduce the overall communication

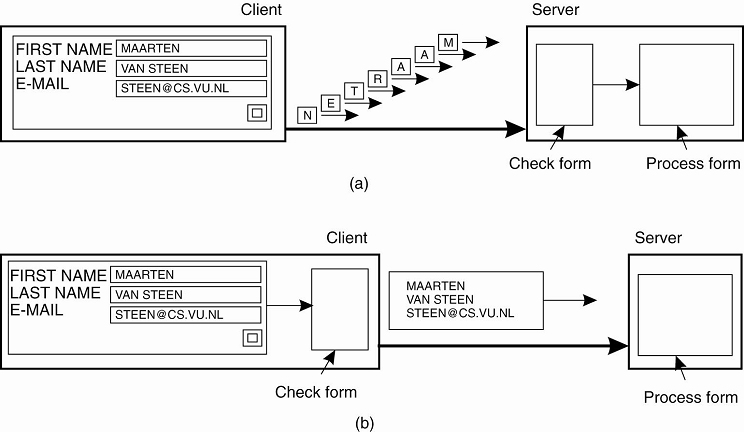
e.g. in interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer.

move part of the computation that is normally done at the server to the client process requesting the service.

O typical case - accessing databases using forms.

ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form - approach of shipping code is now widely supported by the Web in the form of Java applets and Javascript.

The difference between letting (a) a server or (b) a client check forms as they are being filled.



1. Explain the Client–Server Architecture with the help of diagrams [12M]

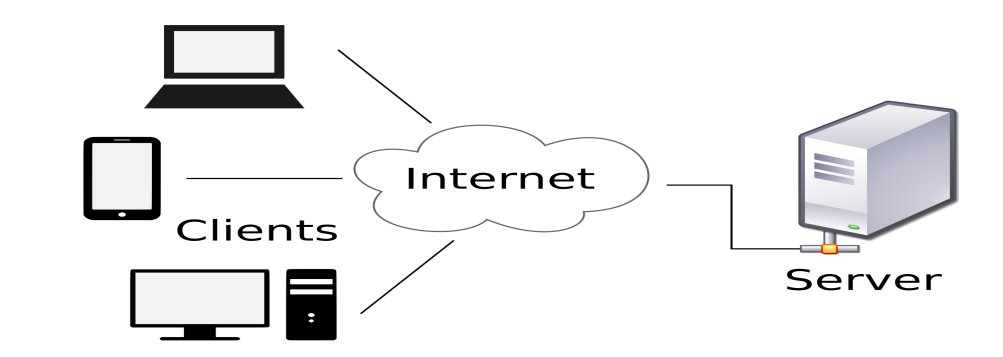
**Client-Server Architectures**

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect [processes](https://en.wikipedia.org/wiki/Process_(computing)) running on those CPUs with some sort of [communication system](https://en.wikipedia.org/wiki/Communication_system).

Distributed programming typically falls into one of several basic architectures: [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server), [three-tier](https://en.wikipedia.org/wiki/Three-tier_(computing)), [n-tier](https://en.wikipedia.org/wiki/Multitier_architecture), or [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer); or categories: [loose coupling](https://en.wikipedia.org/wiki/Loose_coupling), or [tight coupling](https://en.wikipedia.org/wiki/Computer_cluster).

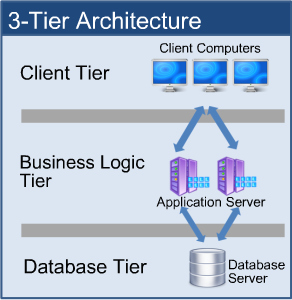
[**Client–server**](https://en.wikipedia.org/wiki/Client%E2%80%93server)

Architectures where smart clients contact the server for data then format and display it to the users. Input at the client is committed back to the server when it represents a permanent change.



[**Three-tier**](https://en.wikipedia.org/wiki/Three-tier_(computing))

Architectures that move the client intelligence to middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.

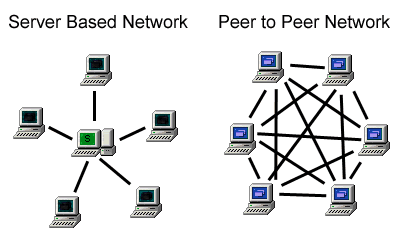


[**n-tier**](https://en.wikipedia.org/wiki/Multitier_architecture)

Architectures that refer typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of [application servers](https://en.wikipedia.org/wiki/Application_server).

[**Peer-to-peer**](https://en.wikipedia.org/wiki/Peer-to-peer)

Architectures where there is no special machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and as servers.

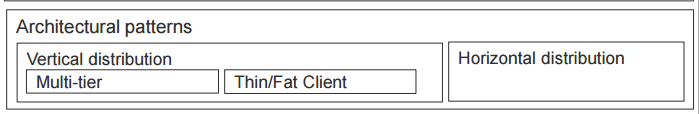


1. What is open distributed system? [6M]

An **Open Distributed System** is made up of components that may be obtained from a number of different sources, which together work as a single **distributed system**. In 1988 the International Standards Organization (ISO) began work on preparing standards for **Open Distributed** Processing (ODP).

**Course Outcome 2:**

1. What is vertical RPC? [10M]



1. What meant by remote object Invocation? [10M]

**Remote Object Invocation**

RMI (**Remote Method Invocation**) is a way that a programmer, using the Java programming language and development environment, can write **object**-oriented programming in which **object**s on different computers can interact in a distributed network.

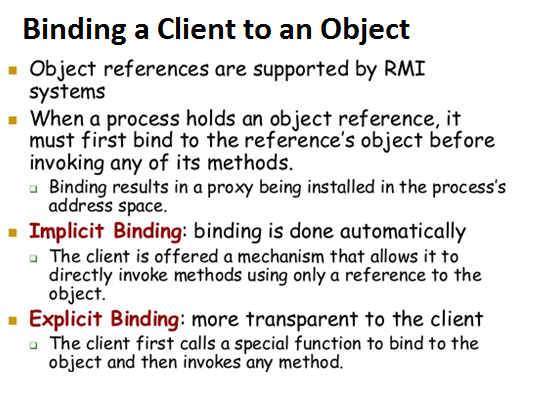
**Distributed objects** are objects (in the sense of [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming)) that are distributed across different [address spaces](https://en.wikipedia.org/wiki/Address_space), either in multiple [computers](https://en.wikipedia.org/wiki/Computer) connected via a [network](https://en.wikipedia.org/wiki/Computer_network) or even in different [processes](https://en.wikipedia.org/wiki/Process_(computing)) on the same computer, but which work together by sharing data and invoking methods.

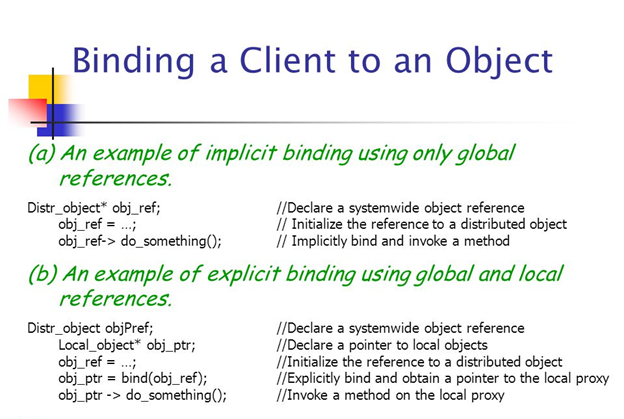
The main method of [distributed object communication](https://en.wikipedia.org/wiki/Distributed_object_communication) is with [remote method invocation](https://en.wikipedia.org/wiki/Remote_method_invocation), generally by message-passing: one object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

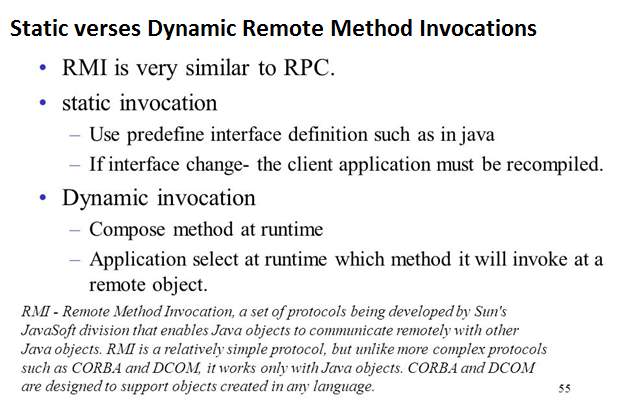
## Local vs. Distributed Objects

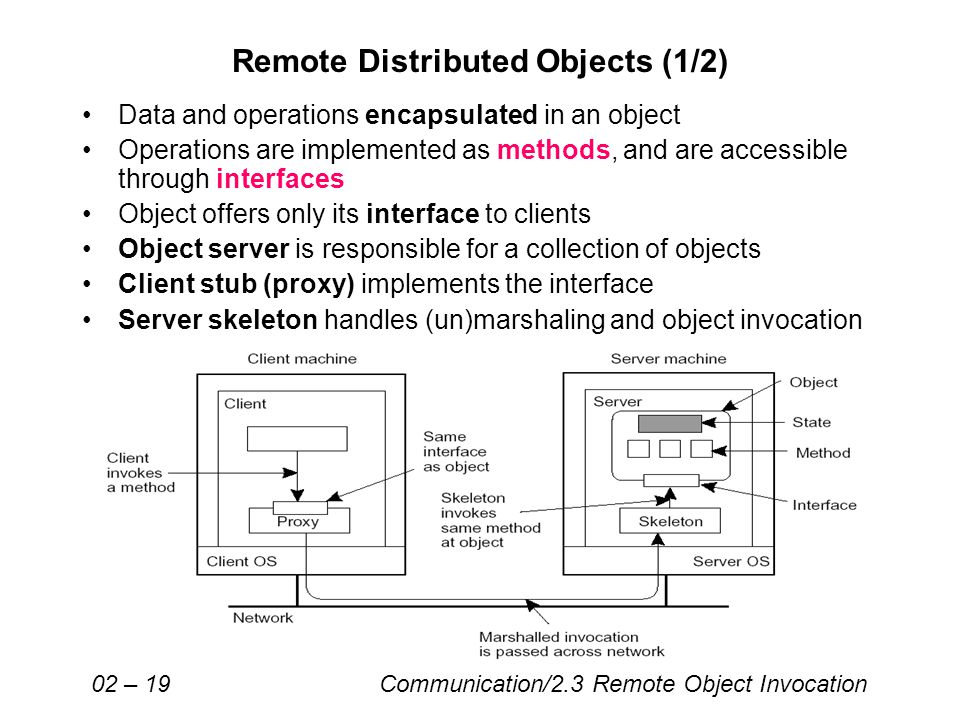
Local and distributed objects differ in many respects. Here are some of them:

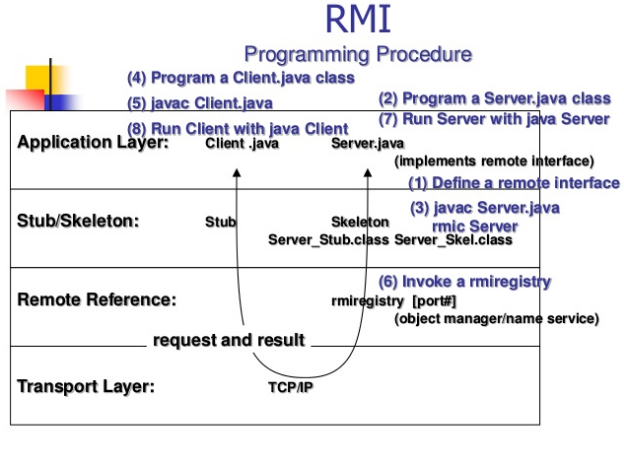
1. **Life cycle** : Creation, migration and deletion of distributed objects is different from local objects
2. **Reference** : Remote references to distributed objects are more complex than simple pointers to memory addresses
3. **Request Latency** : A distributed object request is orders of magnitude slower than local method invocation
4. **Object Activation** : Distributed objects may not always be available to serve an object request at any point in time
5. **Parallelism**: Distributed objects may be executed in parallel.
6. **Communication** : There are different communication primitives available for distributed objects requests
7. **Failure**: Distributed objects have far more points of failure than typical local objects.
8. **Security**: Distribution makes them vulnerable to attack.











1. How are synchronous and asynchronous transmissions different for data streams? [4M]

# Difference between Synchronous and Asynchronous Transmission

In the previous article, we have discussed [Serial and Parallel Transmission](http://techdifferences.com/difference-between-serial-and-parallel-transmission.html). As we know in Serial Transmission data is sent bit by bit, in such a way that each bit follows another. It is of two type namely, Synchronous and Asynchronous Transmission. One of the major differences is that in Synchronous Transmission, the sender and receiver should have synchronized clocks before data transmission. Whereas Asynchronous Transmission does not require a clock but it adds a parity bit to the data before transmission.

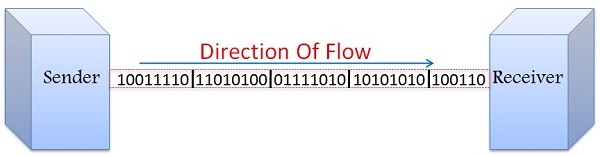
## Content: Synchronous Vs Asynchronous Transmission

1. [Comparison Chart](http://techdifferences.com/difference-between-synchronous-and-asynchronous-transmission.html#ComparisonChart)
2. [Definition](http://techdifferences.com/difference-between-synchronous-and-asynchronous-transmission.html#Definition)
3. [Key Differences](http://techdifferences.com/difference-between-synchronous-and-asynchronous-transmission.html#KeyDifferences)
4. [Conclusion](http://techdifferences.com/difference-between-synchronous-and-asynchronous-transmission.html#Conclusion)

### Comparison Chart

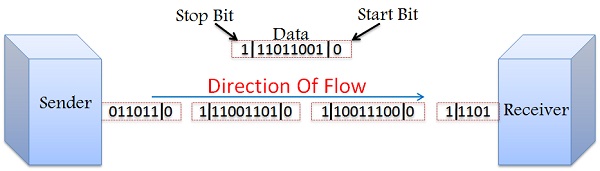
| BASIS FOR COMPARISON | SYNCHRONOUS TRANSMISSION | ASYNCHRONOUS TRANSMISSION |
| --- | --- | --- |
| Meaning | Sends data in the form of blocks or frames | Sends 1 byte or character at a time |
| Transmission Speed | Fast | Slow |
| Cost | Expensive | Economical |
| Time Interval | Constant | Random |
| Gap between the data | Absent | Present |
| Examples | Chat Rooms, Video Conferencing, Telephonic Conversations, etc | Letters, emails, forums, etc |

### Definition of Synchronous Transmission

In Synchronous Transmission, data flows in a full duplex mode in the form of blocks or frames. Synchronization between the sender and receiver is necessary so that the sender know where the new byte starts (since there is no gap between the data).  
  
Synchronous Transmission is efficient, reliable and is used for transferring a large amount of data. It provides real-time communication between connected devices. Chat Rooms, Video Conferencing, telephonic conversations, as well as face to face interactions, are some of the examples of Synchronous Transmission.

### Definition of Asynchronous Transmission

In Asynchronous Transmission data flows in a half duplex mode, 1 byte or a character at a time. It transmits the data in a continuous stream of bytes. In general, the size of a character sent is 8 bits to which a parity bit is added i.e. a start and a stop bit that gives the total of 10 bits. It does not require a clock for synchronization; rather it uses the parity bits to tell the receiver how to interpret the data.

  
It is simple, fast, economical and does not require a 2-way communication. Letters, emails, forums, televisions and radios are some of the examples of Asynchronous Transmission.

## Key Differences Between Synchronous and Asynchronous Transmission

1. In Synchronous Transmission data is transferred in the form of frames on the other hand in Asynchronous Transmission data is transmitted 1 byte at a time.
2. Synchronous Transmission requires a clock signal between the sender and receiver so as to inform the receiver about the new byte. Whereas, in Asynchronous Transmission sender and receiver does not require a clock signal as the data sent here has a parity bit attached to it which indicates the start of the new byte.
3. Data transfer rate of Asynchronous Transmission is slower than that of Synchronous Transmission.
4. Asynchronous Transmission is simple and economic whereas, Synchronous Transmission is complex and expensive.
5. Synchronous Transmission is efficient and has lower overhead as compared to the Asynchronous Transmission.

### Conclusion

Both Synchronous and Asynchronous Transmission have their advantages and disadvantages. Asynchronous is simple, economical and used for transmitting a small amount of data whereas, Synchronous Transmission is used for transferring the bulk of data as it is efficient and has less overhead. Hence, we conclude that both Synchronous and Asynchronous Transmission are necessary for data transmission.

**Course Outcome 3:**

1. Write short notes on Threads in distributed system. [10M]

**PROCESS**

A program under execution is called as a process.

**THREAD**

1. It is a light weight program.

2. **Traditional operating systems**: concerned with the “local” management and scheduling of processes.

3. **Modern distributed systems**: a number of other issues are of equal importance.

4. **There are three main areas of study**

a. Threads and virtualization within clients/servers.

b. Process and code migration.

c. Software agents.

5. Modern OSs provide “virtual processors” within which programs execute.

6. A programs execution environment is documented in the process table and assigned   
a PID.

7. 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**

1. Creating and managing processes is generally   
regarded as an expensive task (fork system call).

2. Making sure all the processes peacefully co-exist on the system is not easy (as concurrency transparency comes at a price).

3. **Threads** can be thought of as an “execution of a part of a program (in user-space)”.

4. 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.

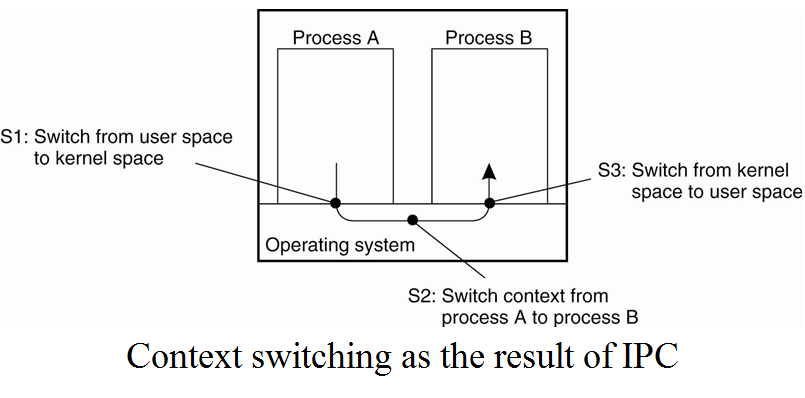
**Two Important Implications**

1.Threaded applications often run faster than non-threaded applications (as context-switches between kernel and user-space are avoided).

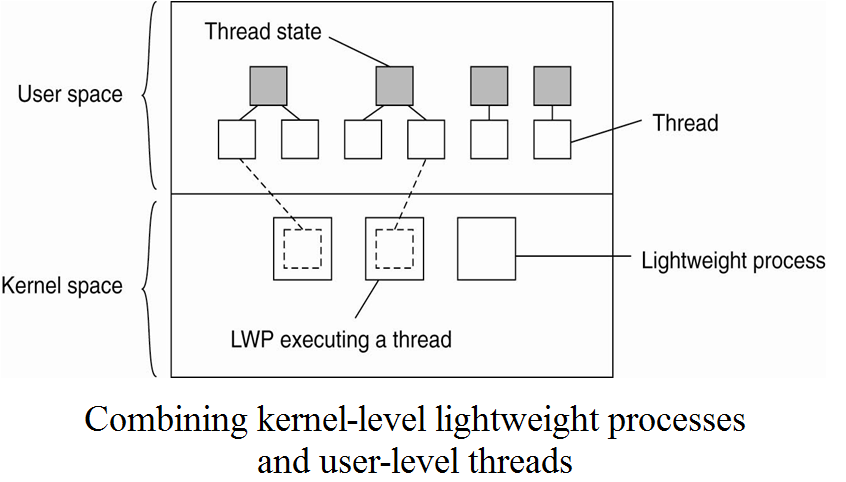
**2.** Threaded applications are harder to develop (although simple, clean designs can help here).

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**



**THREAD IMPLEMENTATION**

****

**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**

**1.** Important characteristic: a blocking call in a thread does not result in the entire process   
being blocked.

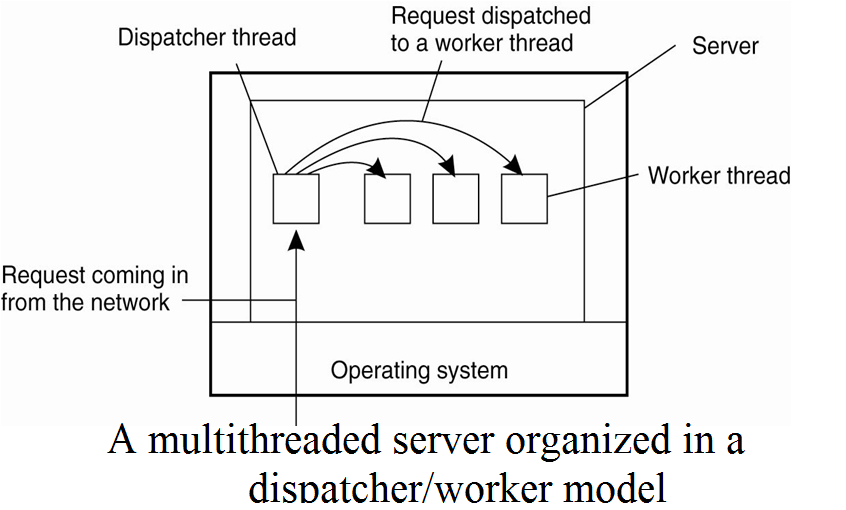
**2.** This leads to the key characteristic of threads within distributed systems:

“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).”

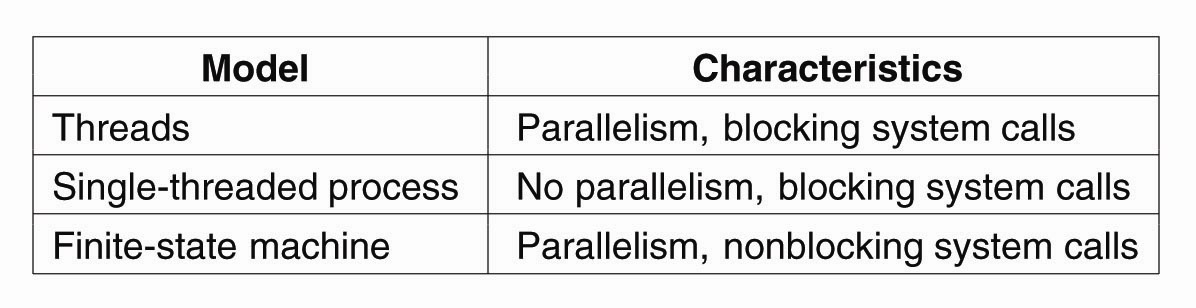
**Example: MT Clients and Servers**

* Mutli-Threaded Client: to achieve acceptable levels of perceived performance, it is often necessary to hide communications latencies.
* Consequently, a requirement exists to start communications while doing something else.
* Example: modern Web browsers.
* This leads to the notion of “truly parallel streams of data” arriving at a multi-threaded client application.
* Although threading is useful on clients, it is much more useful in distributed systems servers.
* The main idea is to exploit parallelism to attain high performance.
* A typical design is to organize the server as a single “dispatcher” with multiple threaded “workers”, as diagrammed overleaf.

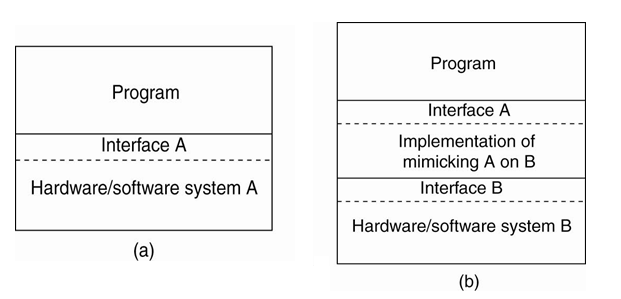
**MULTITHREADED SERVERS**



**MULTITHREADED SERVERS**

****

**THE ROLE OF VIRTUALIZATION IN DISTRIBUTED SYSTEMS**

****

**a. General organization between a program, interface, and system**

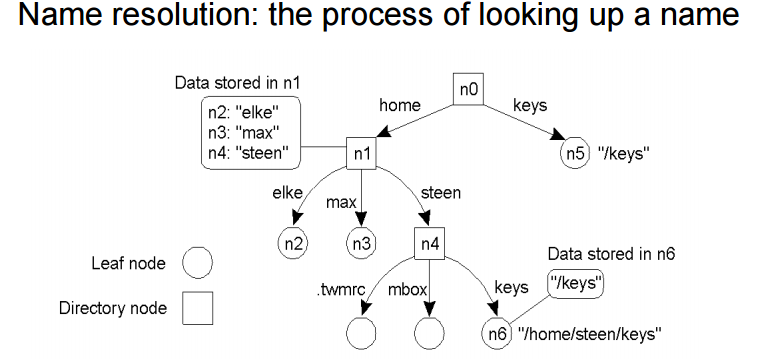
**b. General organization of virtualizing system A on top of system B**

1. Discuss about the implementation of namespace. [8M]

A name in a distributed system is a string of bits or characters that is refer to an entity – Example of entity? Advanced Operating Systems 3 Hosts, printers, disks, files, Processes, users, mailboxes, news groups, Web pages, graphical windows, message, network connections, And so on……

Three types of names: – Address: the name of an access point to an entity – Identifier: • An identifier refers to at most one entity • Each entity is referred to by at most one identifier • An identifier always refers to the same entity – Human-friendly name: • E.g. unix file name, DNS names • Names are always organized in a name space – A name space is an organization mechanism for a group of names

1. What is Name Resolution? [6M]



**Course Outcome 4:**

1. Explain CORBA services in detail. [10M]

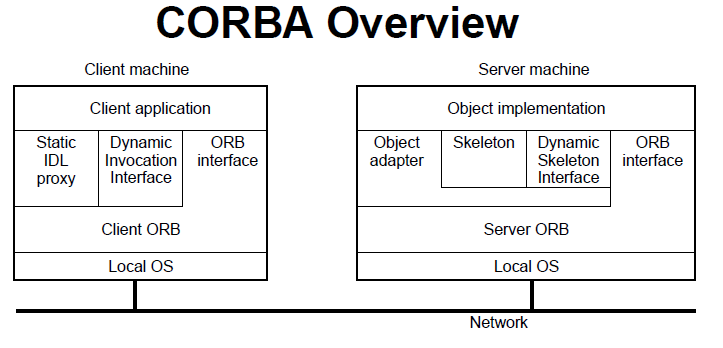
**CORBA (COMMON OBJECT REQUEST BROKER ARCHITECTURE)**

1. Developed by the Object Management Group (OMG) in response to industrial demands for object based middleware and currently in version #2.4 with #3 (almost) done

2. CORBA is a specification: different implementations of CORBA exist

3. Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like

**Essence:** CORBA provides a simple distributed-object model, with specifications for many supporting services.



**Object Request Broker (ORB):** CORBA’s object broker that connects clients, objects, and services

**Proxy/Skeleton:** Precompiled code that takes care of un-marshaling invocations and results

**Dynamic Invocation/Skeleton Interface (DII/DSI):** To allow clients to “construct” invocation requests at runtime instead of calling methods at a proxy, and having the server-side“reconstruct” those request into regular method invocations

**Object adapter:** Server-side code that handles incoming invocation requests.

**Interface repository:** Database containing interface definitions and which can be queried at runtime.

**Implementation repository:** Database containing the implementation (code, and possibly also state) of objects. Effectively: a server that can launch object servers.

**CORBA OBJECT MODEL**

**Essence:** CORBA has a “traditional” remote-object model in which an object residing at an object server is remote accessible through proxies

**Observation:** All CORBA specifications are given by means of interface descriptions, expressed in an IDL. CORBA follows an interface-based approach to objects:

1. Not the objects, but interfaces are the really important entities

2. An object may implement one or more interfaces \_

3. Interface descriptions can be stored in an interface repository, and looked up at runtime

4. Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, COBOL, ADA, and Java.

**CORBA SERVICES**

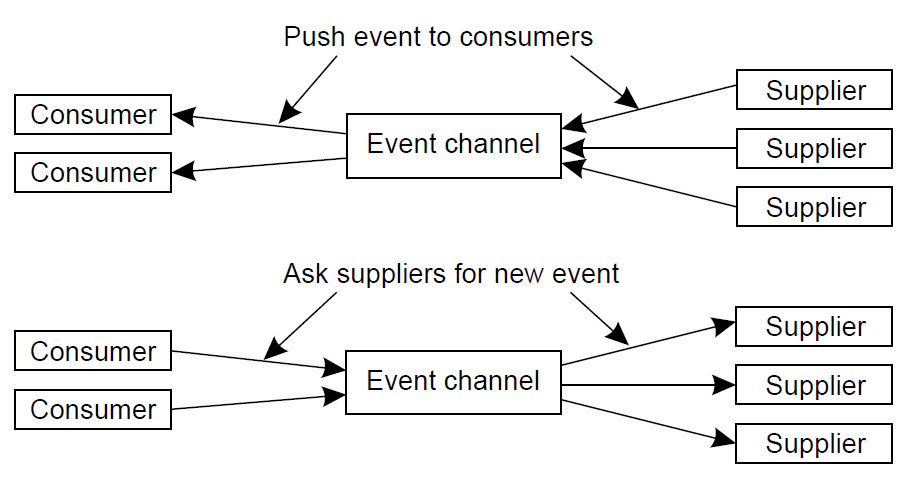
|  |  |
| --- | --- |
| **Service** | **Description** |
| Collection | Facilities for grouping objects into lists, queue, sets, etc. |
| Query | Facilities for querying collections of objects in a declarative manner |
| Concurrency | Facilities to allow concurrent access to shared objects. |
| Transaction | Flat and nested transactions on method calls over multiple objects |
| Event | Facilities for asynchronous communication through events |
| Notification | Advanced facilities for event-based asynchronous communication |
| Externalization | Facilities for marshaling and un-marshaling of objects |
| Life cycle | Facilities for creation, deletion, copying, and moving of objects |
| Licensing | Facilities for attaching a license to an object |
| Naming | Facilities for system wide naming of objects |
| Property | Facilities for associating (attribute, value) pairs with objects |
| Trading | Facilities to publish and find the services an object has to offer |
| Persistence | Facilities for persistently storing objects |
| Relationship | Facilities for expressing relationships between objects |
| Security | Mechanisms for secure channels, authorization, and auditing |
| Time | Provides the current time within specified error margins |

**COMMUNICATION MODELS**

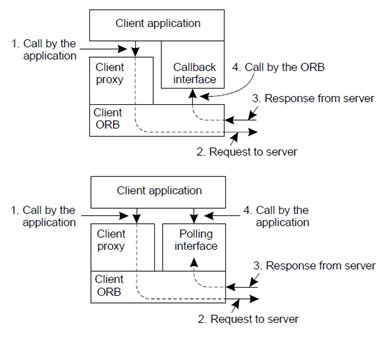
**Object invocations:** CORBA distinguishes three different forms of direct invocations:

|  |  |  |
| --- | --- | --- |
| **Request type** | **Failure sem.** | **Description** |
| Synchronous | At-most-once | Caller blocks |
| One-way | Unreliable | Non-blocking call |
| Deferred synchronous | At-most-once | Non-blocking, but can pickup results later |

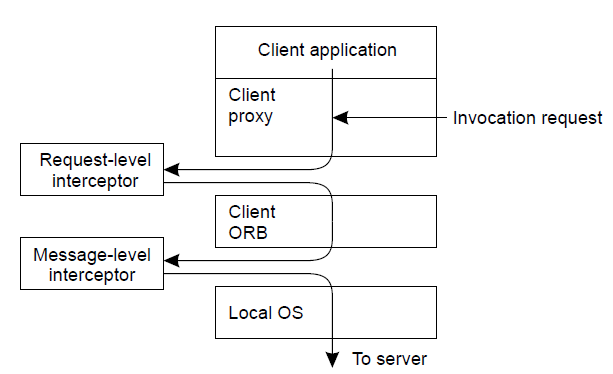
**Event communication:** There are also additional facilities by means of **event channels**:

****

**Messaging facilities:** reliable asynchronous and persistent method invocations:

**  
PROCESSES**

Most aspects of processes for in CORBA have been discussed in previous classes. What remains is the concept of **interceptors**:

****

**Request-level:** Allows you to modify invocation semantics (e.g., multicasting)

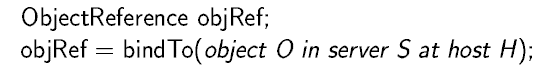
**Message-level:** Allows you to control message-passing between client and server (e.g., handle reliability and fragmentation)

**NAMING**

In CORBA, it is essential to distinguish specification-level and implementation-level object References

**Specification level:** An object reference is considered to be the same as a proxy for the referenced object having an object reference means you can directly invoke methods; there is no separate client to- object binding phase

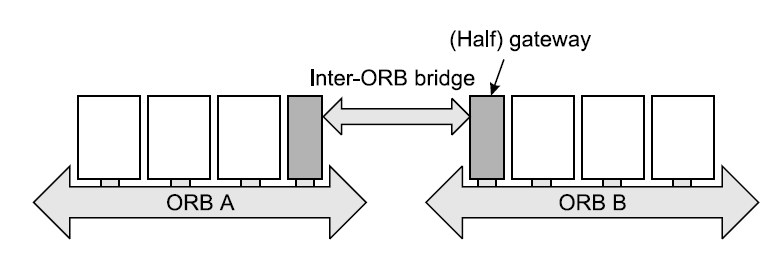
**Implementation level:** When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client address space.



**Conclusion:** Object references in CORBA used to be highly **implementation dependent**: different implementations of CORBA could normally not exchange their references.

**Interoperable Object References**

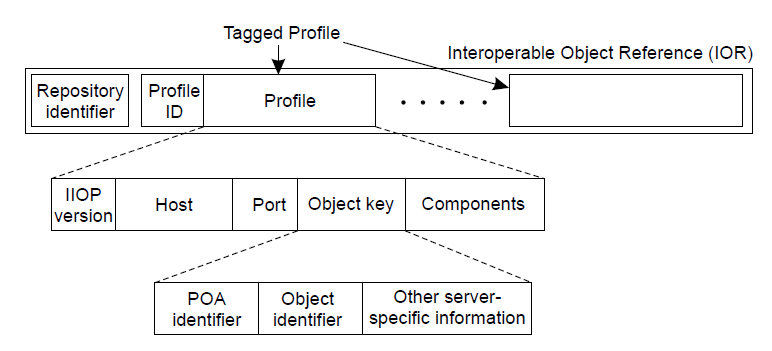
**Observation:** Recognizing that object references are implementation dependent, we need a separate referencing mechanism to cross ORB boundaries



**Solution:** Object references passed from one ORB to another are transformed by the bridge through which they pass (different transformation schemes can be implemented)

**Observation:** Passing an object reference from ORB A to ORB B circumventing the A-to-B bridge may be useless if ORB B doesn’t understand ref A.

**Observation:** To allow all kinds of *different* systems to communicate, we standardize the reference that is passed between bridges:



**NAMING SERVICE**

**Essence:** CORBA’s naming service allows servers to associate a name to an object reference, and have clients subsequently bind to that object by resolving its name

**Observation:** In most CORBA implementations, object references denote servers at specific hosts; naming makes it easier to relocate objects

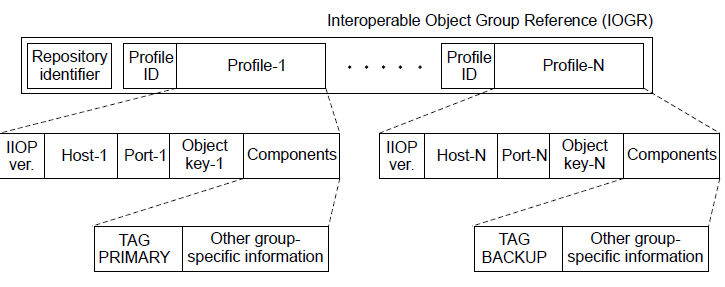
**Observation:** In the naming graph all nodes are objects; there are no restrictions to binding names to objects 􀀀 CORBA allows arbitrary naming graphs

**Question:** How do you imagine cyclic name resolution stops?

**Observation:** There is no single root; an initial context node is returned through a special call to the ORB. Also: the naming service can operate *across* different ORBs🡺 **interoperable naming service**

**FAULT TOLERANCE**

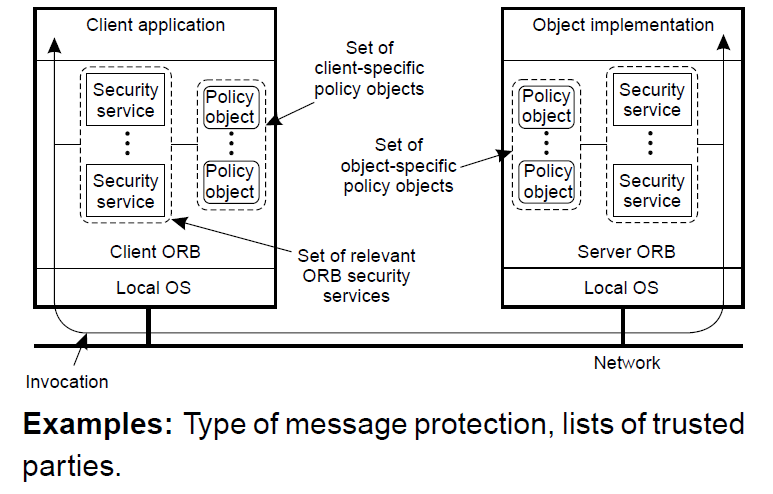
**Essence:** Mask failures through replication, by putting objects into **object groups**. Object groups are transparent to clients: they appear as “normal” objects. This approach requires a separate type of object reference: **Interoperable Object Group Reference**:



**Note:** IOGRs have the same structure as IORs; the main difference is that they are *used* differently. In IORs an additional profile is used as an alternative; in IOGR, it denotes another replica.

**SECURITY**

**Essence:** Allow the client and object to be mostly unaware of all the security policies, except perhaps at binding time; the ORB does the rest. Specific policies are passed to the ORB as (local) objects and are invoked when necessary:



**DISTRIBUTED COM (DCOM: DISTRIBUTED COMPONENT OBJECT MODEL)**

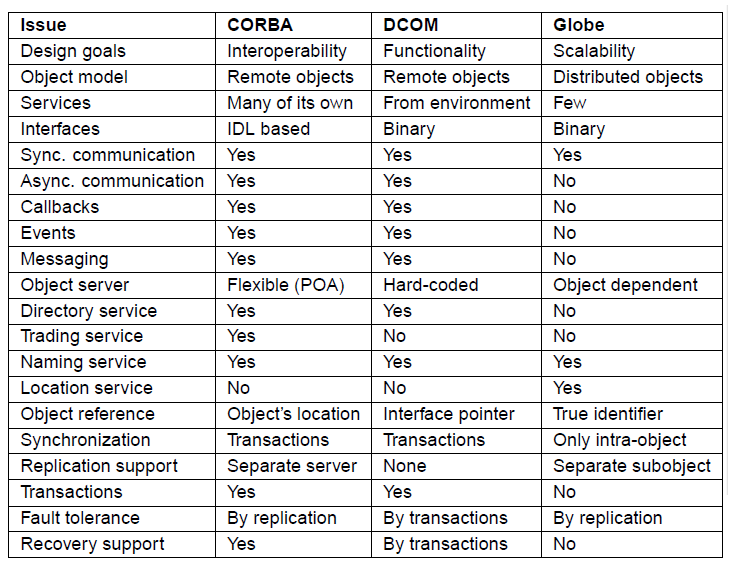
1. Microsoft’s solution to establishing inter-process communication, possibly across machine boundaries.

2. Supports a primitive notion of distributed objects

3. Evolved from early Windows versions to current NT-based systems (including Windows 2000)

4. Comparable to CORBA’s object request broker

1. Write short notes on COREA, DCOM, and Globe. [6M]



1. What is a Globe local object? Explain their types. [6M]

**GLOBE**

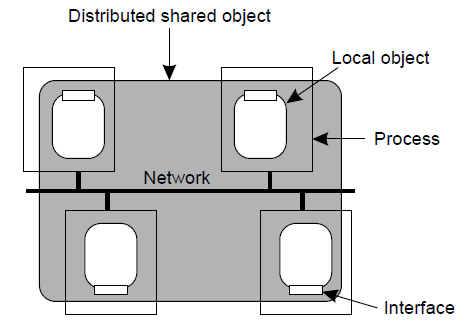
1. Experimental wide-area system currently being developed at Vrije Universiteit

2. Unique for its focus on scalability by means of truly distributed objects.

3. Prototype version up and running across multiplemachines distributed in NL and across Europe and the US.

**Object Model**

**Essence:** A Globe object is a **physically distributed shared object**: the object’s state may be physically distributed across several machines

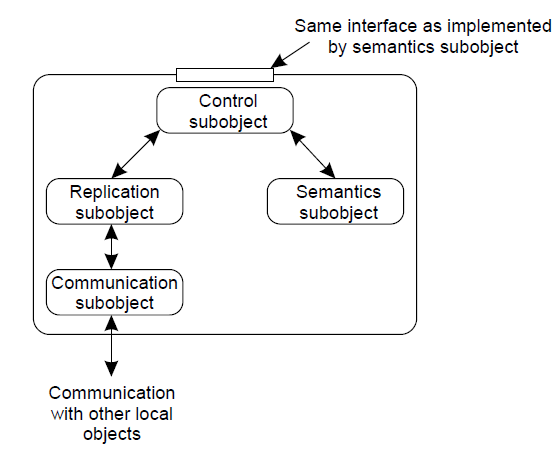


**Local object:** A non-distributed object residing a single address space, often representing a distributed shared object

**Contact point:** A point where clients can contact the distributed object; each contact point is described through a **contact address**

**Object Model**

**Observation:** Globe attempts to separate functionality from distribution by distinguishing different local Sub objects:



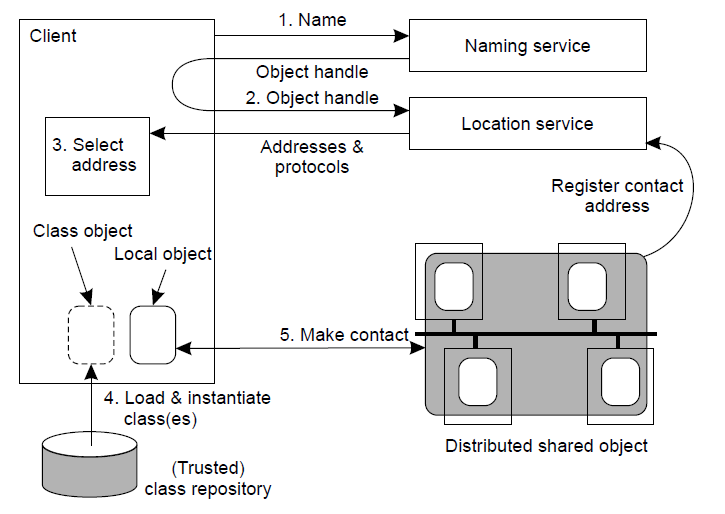
**Semantics sub-object:** Contains the methods that implement the functionality of the distributed shared object

**Communication sub-object:** Provides a (relatively simple), network-independent interface for communication between local objects

**Replication sub-object:** Contains the implementation of an **object-specific** consistency protocol that controls exactly when a method on the semantics sub-object may be invoked

**Control sub-object:** Connects the user-defined interfaces of the semantics sub-object to the generic, predefined interfaces of the replication sub-object

**Client-to-Object Binding**



**Observation:** Globe’s contact addresses correspond to CORBA’s object references

**GLOBE SERVICES**

|  |  |  |
| --- | --- | --- |
| **Service** | **Possible implementation** | **Av?** |
| Collection | Separate object that holds references to other objects | No |
| Concurrency | Each object implements its own concurrency control strategy | No |
| Transaction | Separate object representing a transaction manager | No |
| Event/Notif. | Separate object per group of events (as in DCOM) | No |
| Externalization | Each object implements its own marshaling routines | Yes |
| Life cycle | Separate class objects combined with per-object implementations | Yes |
| Licensing | Implemented by each object separately | No |
| Naming | Separate service, implemented by a collection of naming objects | Yes |
| Property | Separate service, implemented by a collection of directory objects | No |
| Persistence | Implemented on a per-object basis | Yes |
| Security | Implemented per object, combined with (local) security services | Yes |
| Replication | Implemented on a per-object basis | Yes |
| Fault tolerance | Implemented per object combined with fault-tolerant servers | Yes |

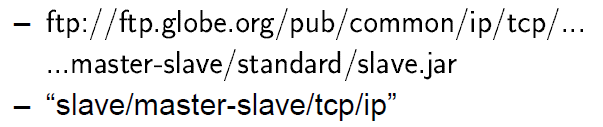
**Object References**

**Essence:** Globe uses location-independent object handles which are to be resolved to **contact addresses** (which describes **where** and **how** an object can be contacted):

1. Associated with a contact point of the distributed object

2. Specifies (for example) a transport-level network address to which the object will listen

3. Contains an **implementation handle**, specifying exactly what the client should implement if it wants to communicate through the contact point:



**Observation:** Objects in Globe have their own objectspecific implementations; there is no “standard” proxy that is implemented for all clients

**NAMING OBJECTS**

**Observation:** Globe separates naming from locating objects (as described in Chapter 04). The current naming service is based on DNS, using TXT records for storing object handles

**Observation:** The location service is implemented as a generic, hierarchical tree, similar to the approach explained in Chapter 04.

**CACHING AND REPLICATION**

**Observation:** Here’s where Globe differs from many other systems:

1. The organization of a local object is such that replication is inherently part of each distributed

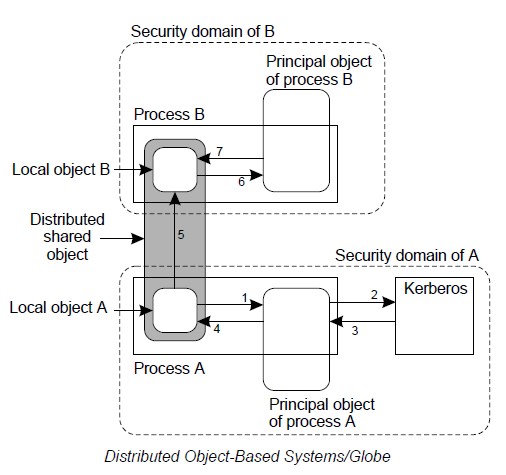
shared object

2. All replication sub-objects have the same interface:

|  |  |
| --- | --- |
| **Method** | **Description** |
| start | Called to synchronize replicas of the semantics subobjects, obtain locks if necessary, etc. |
| send | Provide marshaled arguments of a specific method, and pass invocation to local objects in other address spaces |
| invoked | Called after the control sub-object has invoked a specific method at the semantics subobject |

\_ This approach allows to implement any **object-specific** caching/replication strategy

**Security: Essence:** Additional security sub-object checks for authorized communication, invocation, and parameter values. Globe can be integrated with existing security services:



**Course Outcome 5:**

1. Discuss Real time scheduling for resource management in distributed multimedia systems. [8M]

**RESOURCE MANAGEMENT**

To provide a certain QoS level to an application, not only does a system need to have sufficient resources (performance), it also needs to make these resources available to an application when they are needed (scheduling).

**RESOURCE SCHEDULING**

Processes need to have resources assigned to them according to their priority. A resource scheduler determines the priority of processes based on certain criteria. Traditional CPU

schedulers in time-sharing systems often base their priority assignments on responsiveness and fairness: I/O intensive tasks get high priority to guarantee fast response to user requests, CPU-bound tasks get lower priorities and overall, processes in the same class are treated equally.

**RESOURCE SCHEDULING TYPES**

1. Fair scheduling. 2. Real-time scheduling.

**FAIR SCHEDULING** If several streams compete for the same resource it becomes necessary

To consider fairness and to prevent ill-behaved streams from taking too much bandwidth. A straightforward approach to ensure fairness is to apply round-robin scheduling to all streams in the same class. Whereas in [Nagle 1987] such a method was introduced on a packet-by-packet basis, in [Demers et al. 1989] the method is used on a bit-by-bit basis which provides more fairness with respect to varying packet sizes and packet arrival times. These methods are known as fair queuing. It is done by bit by bit or packet by packet.

All basic round-robin schemes assign the same bandwidth to each stream. To take the individual bandwidth of streams into account, the bit-by-bit scheme can be extended so that for certain streams a larger number of bits can be transmitted per cycle. This method is called weighted fair queuing.

**REAL-TIME SCHEDULING** Traditional real-time scheduling methods suit the model of regular continuous multimedia streams very well. Earliest-deadline-first (EDF) scheduling has almost become a synonym for these methods. An EDF scheduler uses a deadline that is associated with each of its work items to determine the next item to be processed: the item with the earliest deadline goes first. In multimedia applications, we identify each media element arriving at a process as a work item. EDF scheduling is proven to be optimal for allocating a single resource based on timing criteria: if there is a schedule that fulfils all timing requirements, EDF scheduling will find it.

**RATE-MONOTONIC (RM) SCHEDULING** is a prominent technique for real-time scheduling of periodic processes that achieves just this. RM scheduling has been shown to be optimal for situations that only utilize a given bandwidth by less than 69% . Using such an allocation scheme, the remaining bandwidth could be given to non-real-time applications.

**STREAM ADAPTATION**

Whenever a certain QoS cannot be guaranteed or can only be guaranteed with a certain

probability, an application needs to adapt to changing QoS levels, adjusting its performance accordingly. For continuous-media streams, the adjustment translates into different levels of media presentation quality.

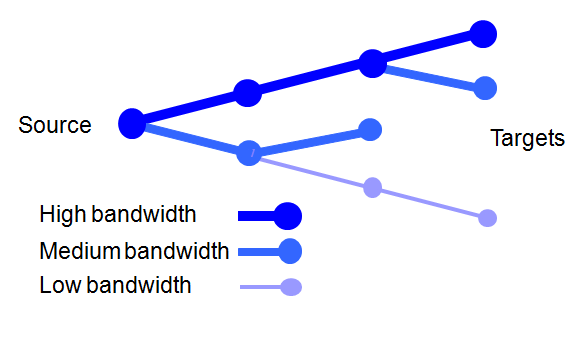
The simplest form of adjustment is to drop pieces of information. This is easily done in audio streams where samples are independent from each other, but it can immediately be noticed by the listener. Drop-outs in a video stream encoded in Motion JPEG, where each frame stands for itself are more tolerable. Encoding mechanisms such as MPEG, where the interpretation of a frame depends on the values of several adjacent frames, are less robust against omissions: it takes a longer time to recover from errors and the encoding mechanism may, in fact, amplify errors.

For non-interactive applications this may be acceptable, although it can eventually lead to buffer overflows as data is accumulated between the source and sink. For conferencing and other interactive applications, increasing delays are not acceptable, or must exist only for a short period.

**SCALING (RESIZING)**

If adaptation is performed at the target of a stream, the load on any bottleneck in the system is not decreased and the overload situation persists. It is useful to adapt a stream to the bandwidth available in the system before it enters a bottleneck resource in order to resolve contention. This is known as scaling. Scaling algorithms are media-dependent.

**FILTERING**



It can also be achieved by dropping a channel in a stereo transmission.

**Scaling methods can work at different granularities for video**

**1. TEMPORAL SCALING** reduces the resolution of the video stream in the time domain by decreasing the number of video frames transmitted within an interval. Temporal scaling is best suited for video streams in which individual frames are selfcontained and can be accessed independently. Delta compression techniques are more difficult to handle as not all frames can be easily dropped. Hence, temporal scaling is more suitable for Motion JPEG than for MPEG streams.

**2. SPATIAL SCALING** reduces the number of pixels of each image in a video stream. For spatial scaling, hierarchical arrangement is ideal because the compressed video is immediately available in various resolutions. Therefore, the video can be transferred over the network using different resolutions without recoding each picture before finally transmitting it. JPEG and MPEG-2 support different spatial resolutions of images and are well-suited for this kind of scaling.

**3. FREQUENCY SCALING** modifies the compression algorithm applied to an image. This result in some loss of quality, but in a typical picture, compression can be increased significantly before a reduction of image quality becomes visible.

**4. AMPLITUDINAL SCALING** reduces the colour depths for each image pixel. This scaling method is, in fact, used in H.261 encodings to arrive at a constant throughput although image content varies.

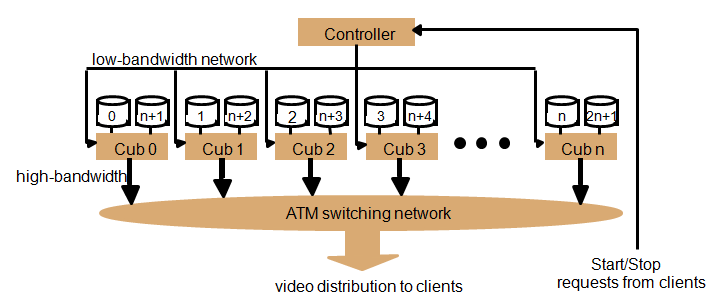
**5. COLOUR SPACE SCALING** reduces the number of entries in the colour space. One way to realize colour space scaling is to switch from colour to greyscale presentation. Obviously, combinations of these scaling methods are possible. A system to perform scaling consists of a monitor process at the target side and a scaler process at the source. The monitor keeps track of the arrival times of messages. When messages get delayed, it is an indication of some bottleneck in the system. The monitor then sends a Scale-Down message to the source and it reduces the bandwidth of the stream. After some period of time, the source scales the stream up again. Should the bottleneck still exist, the monitor will again detect a delay and scale the stream down 1993]. The fundamental problem of the scaling approach is to find good heuristics to avoid unnecessary Scale-Up operations and to prevent the system from oscillating.

**FILTERING**

As scaling modifies a stream at the source, it is not always suitable for applications that involve several receivers: when a bottleneck occurs on the route to one target, this target sends a Scale-Down message to the source and all targets receive the degraded quality although some would have no problem in handling the original stream.

Filtering is a method that provides the best possible QoS to each target by applying scaling at each relevant node on the path from the source to the target (Figure 15.9). RSVP is an example of a QoS negotiation protocol that supports filtering. Filtering requires that a stream can be partitioned into a set of hierarchical substreams, each adding a higher level of quality. The capacity of nodes on a path determines the number of sub-streams a target receives. All other sub-streams are filtered out as close to the source as possible (perhaps even at the source) to avoid transfer of data that is later thrown away. A sub-stream is not filtered at an intermediate node if somewhere downstream a path exists that can carry the entire sub-stream.

**TIGER VIDEO FILE SERVER HARDWARE CONFIGURATION**



**TIGER SCHEDULE**

1. What are the typical characteristics of multimedia data? [8M]

**CHARACTERISTICS OF TYPICAL MULTIMEDIA STREAMS**

It is likely that multimedia applications will remain in the window of scarcity (Say resources sufficient or not) for the foreseeable future. Advances in system performance are likely to be used to improve the quality of multimedia data, to include higher frame rates and greater resolution for video streams or to support many media streams concurrently, for example in **a video conferencing** system. More demanding applications, including virtual reality and real-time stream manipulation (“special effects”) can extend the window of scarcity almost indefinitely.

The term ‘continuous’ refers to the **user’s view of the data**. Internally, continuous media are represented as sequences of discrete values which replace each other over time. For example, the value of an image array is replaced 25 times per second to give the impression of a TV-quality view of a moving scene; a sound amplitude value is replaced 8000 times per second to convey telephone quality speech.

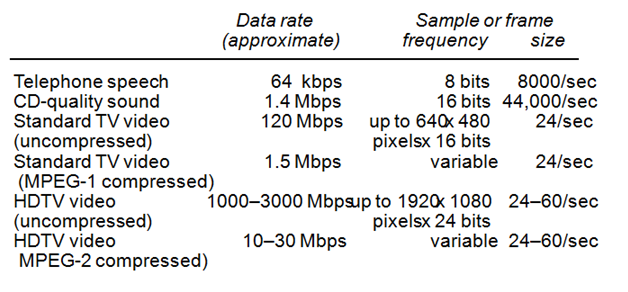
Multimedia streams are said to be time-based (or isochronous) because timed data elements in audio and video streams define the semantics or ‘content’ of the stream. The times at which the values are played or recorded affect the validity of the data. Hence systems that support multimedia applications need to preserve the timing when they handle continuous data.

Compression can reduce bandwidth requirements by factors between 10 and 100, but the timing requirements of continuous data are unaffected. Various compressed data formats such as GIF, TIFF and JPEG for still images and MPEG-1, MPEG-2 and MPEG-4 for video sequences.

For Compression we use of special-purpose hardware to process and despatch video and audio information – the video and audio coders/decoders (Codecs) found on video cards manufactured for personal computers. The compression method used for the MPEG video formats is asymmetric, with a complex compression algorithm and simpler decompression.

**The window of scarcity for computing and communication resources**





**TYPICAL INFRASTRUCTURE COMPONENTS FOR MULTIMEDIA APPLICATIONS**



1. What is Admission Control? [6M]

**ADMISSION CONTROL**

If the result of the resource evaluation is positive, the requested resources are reserved and the application is given a Resource Contract, stating the resources that have been reserved. The contract includes a time limit. The application is then free to run. If it changes its resource requirements it must notify the QoS Manager. If the requirements decrease, the resources released are returned to the database as available resources. If they increase, a new round of

negotiation and admission control is initiated.