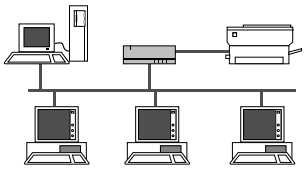


# Distributed Systems
# 600.437
## Distributed Objects

Department of Computer Science
The Johns Hopkins University



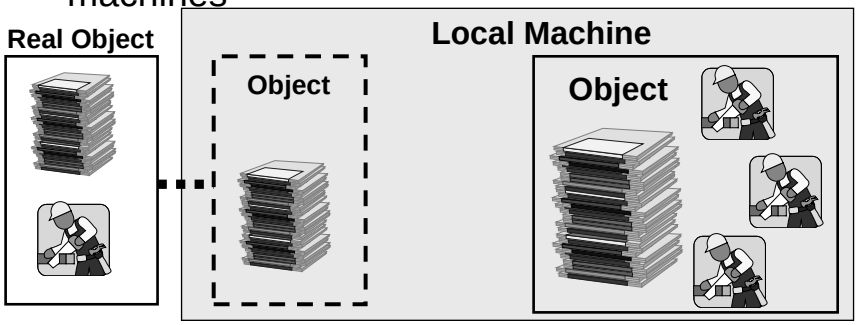
# Distributed Objects

## Lecture 10

Corba Info http://www.omg.com/corba/.
JavaSpaces Specification 1.0 from Sun.

# What are (Distributed) Objects?

- Well defined collection of data variables and the functions that act on them
- Distributed Objects look like they are 'normal' objects, but they might be located on different machines






# Why Distribute Objects?

- Abstract model
- Fault-tolerance
- Scalability
- Availability
- Performance

# Problems with Distributed Objects

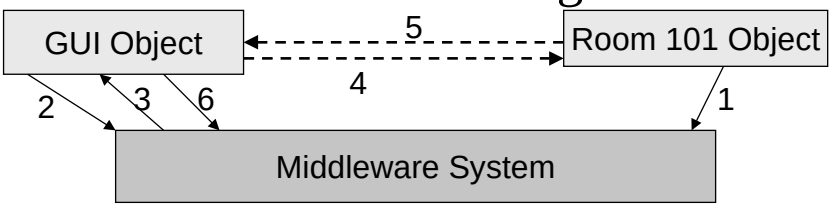
- Abstraction Model
- Performance
- Latency
- Partial Failure
- Synchronization
- Complexity



# Ways of Distributing Objects

- Message Passing
  - Object knows about network, Network data = Min needed.
- Argument/Return Passing
  - Like RPC, Network data = Size(Args) + Size(Return) + Size(Name).
- Serializing & Sending Object
  - Actual object including code is sent. Might need synchronization. Network data = Size(object code) + Size(object state) + Size(synchronization)
- Shared Memory
  - Rely on DSM implementation. Network data = Size(data touched) + Size(synchronization).

# Application Example: Room Booking




1) Register control of Room 301.
2)Who controls Room 301?
3) Object 5a62b9D does.
4) Object 5a62b9D, register room 301 for 3pm to 5pm on Wed, Nov 15,2000.
5) Return value TRUE, Registration succeeded.
6) Release Object 5a62b9D.

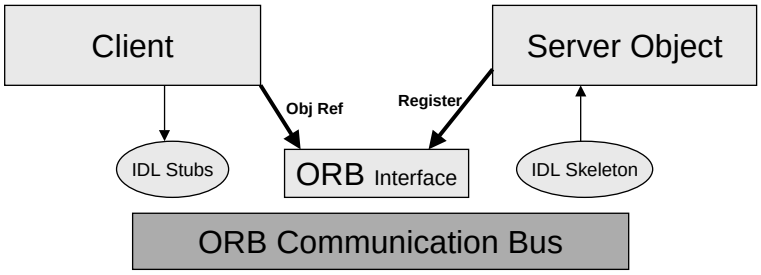


# Necessary Services

- Naming & Directory Service
  - Where can I find an object called X?
- Transaction Service
  - Please do all of (X,Y,Z) or none of them.
- Fault Detection Service
  - Which service or object has died
- Trading Service
  - Where can I find an object that does Y?

# CORBA Introduction

- Created by OMG
- Involves 3 parties: Clients, ORBs, Servers
- Cross-platform






# CORBA IDL

IDL is Interface Definition Language

It defines the exported methods and modules by using a standard set of data types (string, integer, float, object, …) which define the parameters of the methods and their return values.

**Key Points:**

- Platform Independent
- Very Abstract--No details of actual object

```
// SimpleHelloWorld.idl

module SimpleHelloWorld {
    interface GoodDay {
        string hello();
    };
};
```

# CORBA

## Steps to Creating a CORBA object:

**1)** Write some IDL that describes the interfaces to the object.

**2)** Compile the IDL using the IDL compiler provided by the particular ORB. This produces the stub and skeleton code that implements location transparency. That is, it will convert an object reference into a network connection to a remote server, and then marshal the arguments we provide to an operation on the object reference, convey them to the correct method in the object denoted by our object reference, execute the method, and return the results.

**3)** Write an implementation of the object that actually does the work.

**4)** Write code to initialize the ORB and inform it of any CORBA objects that we have created.

**5)** Compile all the generated code and our application code.

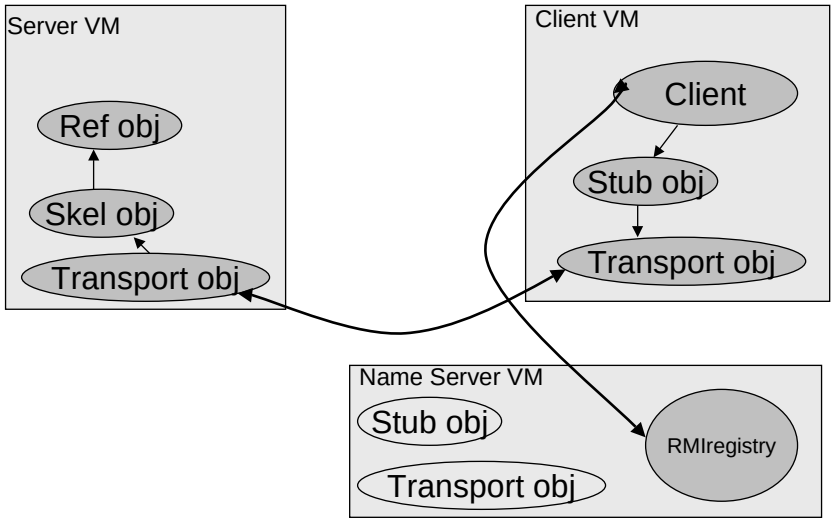
**6)** Run the distributed application.



# Java RMI Introduction

- Remote Method Invocation
- Uses Java's Object Serialization and Reflection capabilities
- Actual Transport protocols not specified…
  - Default is TCP
  - Improvements(Caching, Reliable UDP, Multicast)
  - …Are they really improvements?

# Java RMI






# How is OLE Different?

- Some syntax differences, uses UUID's
- Mechanism -- fast library linking
  - Shared library linking
  - No need for marshalling/serializing data formats
  - No overhead of communications
- Includes embedding specification
  - Excel Spreadsheet inside Word Document
  - Specifies WHAT KIND of object can be used, not just how to distribute ANY object

# Performance Issues

Network Delays

ORB/middleware overhead

Protection barrier transitions

Marshalling/serialization

Biggest one:

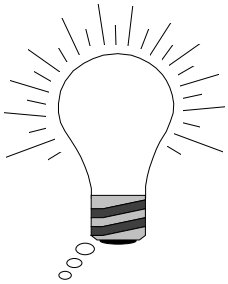
**UNCERTAINTY**



# Handling Failures

- ORB dies
- Network partitions
- Machine dies
- Clients can't directly detect what is really going on--both good and bad
  - Can have replicated server objects
  - Can transparently handle failures by spawning new server objects
  - Client application has to have other ways to react to failures that are not contained by the ORB

# JavaSpaces



Message Passing is hard

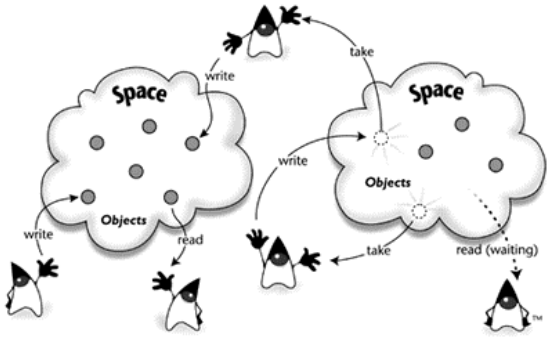
….Objects are still complicated

…Shared Memory is easier to program, but preserving semantics is costly in performance…

**What about something like shared memory but with weaker, but still useful semantics.**



# Model of JavaSpaces




- Spaces
- Entities

# Model continued

- Simple
  - Only 4 basic primitives and each only does one action.
- Expressive
  - Many algorithms can be expressed as moving entities in and out of shared spaces.
  - One could view it as high-level shared memory.
- Supports client/server
- Loosely coupled
  - Works best when entities are high level and represent a non-trivial amount of work.



# Historical Perspective--Linda

- Original model of virtual spaces of tuples
- Created in early 1980's
- Added transactions, queries, fault-tolerance in early/mid 1990's
- Java adds:
  - strongly typed fields in tuples
  - objects--not data stored in spaces
  - match object type as well as fields
  - leases
  - atomic actions across multiple spaces

# Operations

- write
  - Insert an Entity into a particular JavaSpace
- take
  - Remove an Entity from a JavaSpace and add it to your local VM
- read
  - Copy an Entity into your local VM, leave the original in the JavaSpace
- notify
  - Register interest in any Entity which matches a template



# Advanced Operations

- Transactions
  - Database semantics--They are atomic
  - Can tradeoff isolation with liveness by using timeout values on read/take
  - Persistence is not guaranteed
- Leases
  - Allows garbage collection
  - Can be used to create Entities that age

# Simple Example Program

This is a Hello World, which counts how many times the client program has printed “Hello, World”.

It uses 3 classes, one for the Entry in JavaSpace, one for the main HelloWorld monitor class, and one for the client HelloWorld class.

```
Public class Message implements Entry {
        public String content;
        public Integer counter;
public Message() { }
public Message(String content, int initVal) {
        this.content = content;
        counter = new Integer(initVal); }
public String toString() {
        return content + “ read “ + counter + “ times.”; }
public void increment() {
        counter = new Integer(counter.intValue() + 1); }
}
```



# Example Program cont.

```
Public class HelloWorld {
        public static void main(String[] args) {
                Message msg = new Message( “Hello World”, 0);
                JavaSpace space = SpaceAcessor.getSpace();
                space.write(msg, null, Lease.FOREVER);
                Message template = new Message();
                for (;;) {
                        Message result = (Message)space.read(template,
null, Long.MAX_VALUE);
                        system.out.println(result);
                        Thread.sleep(1000);
                }
        }
}
```

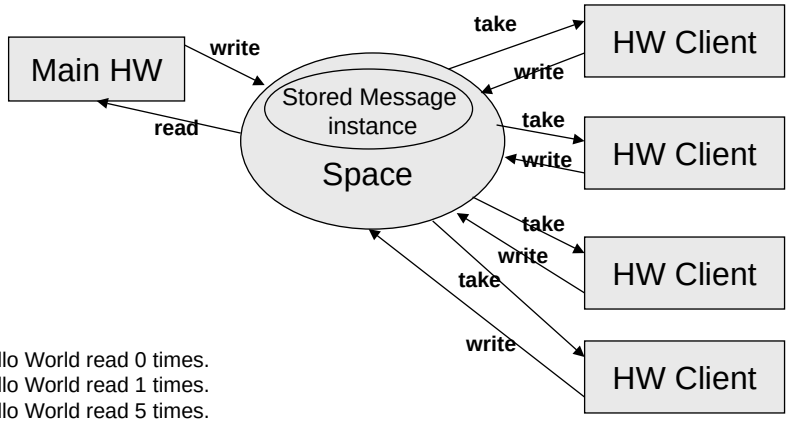
# Example Program cont.

```
Public class HelloWorldClient {
        public static void main(String[] args) {
                JavaSpace space = SpaceAcessor.getSpace();
                Message template = new Message();
                for (;;) {
                        Message result = (Message)space.take(template, null,
Long.MAX_VALUE);
                        result.increment();
                        space.write(result, null, Lease.FOREVER);
                        Thread.sleep(1000);
                }
        }
}
```



# Example Program execution

What should happen?




Hello World read 0 times.
Hello World read 1 times.
Hello World read 5 times.
Hello World read 10 times.

# Application Design Patterns

- Replicated Workers
  - Rendering CG Frames
  - Parallel jobs (simulations, Monte-Carlo, etc)
- Marketplace
  - A number of independent actors each have programs that monitor certain spaces and act on Entities they find there that match what they are interested in. They then place the resulting Entities into a different space (or maybe the same) for other actors to use.



# Other Application Designs

- Collaborative tools
- Brokers/E-commerce/Auctions
- Parallel Job Distribution