

# Object-based distributed systems

INF 5040/9040 autumn 2011

**Lecturer: Lucas Provensi**

Frank Eliassen, SRL & Ifi/UiO

1

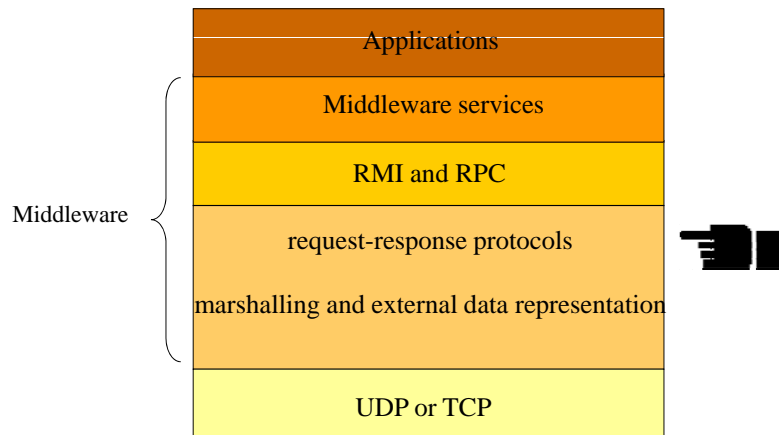
## Plan

- Request-response protocols
- Characteristics of distributed objects
- Communication between distributed objects (RMI)
- Object-servers
  - Multi-threaded object servers
- CORBA middleware
- Java RMI

Frank Eliassen, SRL & Ifi/UiO

2

# Layering of middleware



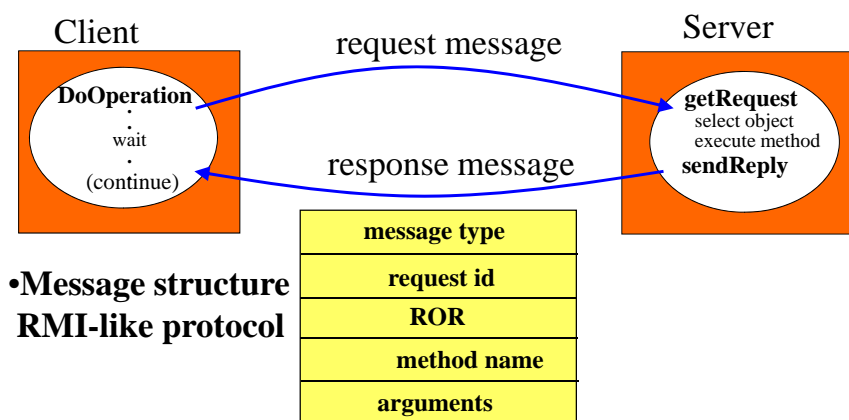
Frank Eliassen, SRL & Ifi/UiO

3

# Request-reponse protocols

## Basis for client-server communication

- Usually based on UDP or TCP



Frank Eliassen, SRL & Ifi/UiO

4

## Failure model for request-response protocols

- Protocol can be exposed to
  - omission failure
  - process crash failure
  - message order not guaranteed (UDP)
- Failure is detected as *timeout* in the primitive DoOperation:
  - recovery actions depend on the offered delivery guarantee

Frank Eliassen, SRL & Ifi/UiO

5

## Failure and recovery for request-response protocols (I)

- Timeout DoOperation
  - Send request message repeatedly until
    - response is available, or
    - assume server has failed (max no of retrans.)
- Duplicate request messages
  - occur when request message is sent more than once
  - can lead to operations being executed more than once for the same request
  - => must be able to filter duplicate requests (role of request id)
- Lost response messages
  - server has already sent response message when it receives a duplicate request message
  - => may have to execute the operation again to get the right response
    - OK for operations that are "idempotent"

Frank Eliassen, SRL & Ifi/UiO

6

## Failure and recovery for request-response protocols (II)

- Logs (histories):
  - used by servers offering operations that are not “idempotent”
  - contains response messages already sent
- Disadvantage of logs:
  - storage requirement
- if a client is allowed to do only one request at a time to the same server, the log can be limited in size (bounded by the number of concurrent clients)
- at reception of the next request message from the same client the server may delete the last response message for that client from the log

Frank Eliassen, SRL & Ifi/UiO

7

## Classification of request-response protocols

- Classification after (Spector, 1982):
  - basis for implementing different types of RMI and RPC (with different levels of delivery guarantees)
- Request (R) protocol
  - Only Request-message. No response message from server
  - No confirmation that operation has been performed (**one-way** operation)
- Request-Reply (RR) protocol
  - Reply-message confirms that the Request-message has been performed
  - A new request from the client confirms reception of Reply-message
- Request-Reply-Acknowledge (RRA) protocol
  - separate message from client to confirm reception of Reply-message
  - tolerates loss of Ack-message
    - Ack with a given *request id* confirms all lower requests ids

Frank Eliassen, SRL & Ifi/UiO

8

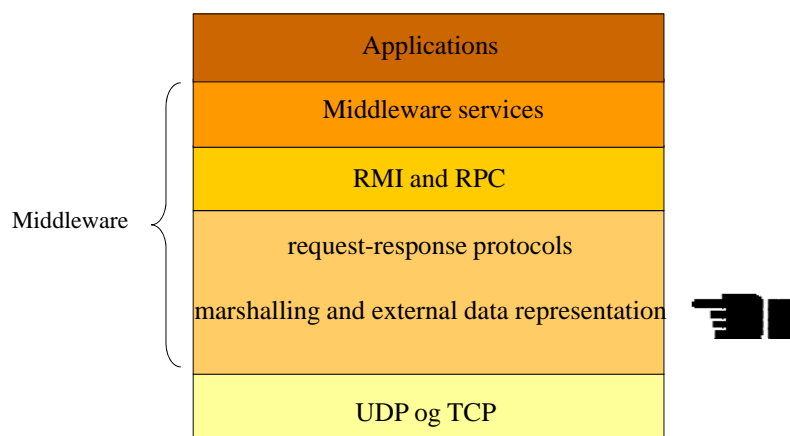
## A note on request-response protocols: TCP vs UDP

- UDP has limited packet size
  - => need for fragmentation/defragmentation protocols
- request-response protocols over TCP avoids this problem
  - TCP ensures reliable delivery of byte streams
- Problem:
  - much overhead if the connection has to be created at each request
    - => need for optimization (leave connection open for later reuse)
  - upper bound on number of concurrent TCP-connections could cause problems

Frank Eliassen, SRL & Ifi/UiO

9

## Layering of middleware



Frank Eliassen, SRL & Ifi/UiO

10

## Message data representation

- Data structures must be flattened before transmission and rebuilt on arrival
- Issue: the representation of data structures and primitive data types can be different between systems
- Two methods for exchanging binary data values
  - Use external format
    - sender converts data values to an agreed external format
    - recipient converts to local form
  - Use sender's format
    - recipient converts the values if necessary
    - message carries indication of format used

Frank Eliassen, SRL & Ifi/UiO

11

## Marshalling

- "marshalling"
  - serialize data structures to messages (sequence of data values)
  - translate sequence of data values to an external representation
- "unmarshalling"
  - inverse of "marshalling"

Frank Eliassen, SRL & Ifi/UiO

12

## Some external data representation formats

- Sun XDR (representation of most used data types)
- ASN.1/BER (ISO standard, based on "type-tags", open)
- NDR (used in DCE RPC)
- CDR (used in CORBA RMI, binary layout of IDL types)
- Java Object Serialization (JOS)
- XML (used i SOAP: "RMI" protocol for Web Services)

Frank Eliassen, SRL & Ifi/UiO

13

## Example: Extensible Markup Language (XML)

- XML definition of data structure

```
<person id="123456789">  
  <name>Smith</name>  
  <place>London</place>  
  <year>1984</year>  
</person>
```


- XML namespaces
  - Provide a means for scoping names
- XML schemas
  - Defines the elements and attributes that can appear in a document

Frank Eliassen, SRL & Ifi/UiO

14

# Plan

---

- Request-response protocols
- Characteristics of distributed objects 
- Communication between distributed objects (RMI)
- Object-servers
  - Multi-threaded object servers
- CORBA middleware
- Java RMI

## Characteristics of distributed objects - I

---

- Distributed objects execute in different processes.
  - each object has a **remote interface** for controlling access to its methods and attributes that can be accessed from other objects in other processes located on the same or other machines
    - declared via an “Interface Definition Language” (IDL)
  - Remote Method Invocation (RMI)
    - method call from an object in one process to a (remote) object in another process



## Characteristics of distributed objects - II

- Distributed objects have a unique identity referred to a **Remote Object Reference (ROR)**
- Other objects that want to invoke methods of a remote object needs access to its ROR
- RORs are “first class values”
  - can occur as arguments and results in RMI
  - can be assigned to variables
- Distributed objects are encapsulated by interfaces
- Distributed objects can raise “exceptions” as a result of method invocations
- Distributed objects have a set of named attributes that can be assigned values

Frank Eliassen, SRL & Ifi/UiO

17

## The type of a distributed object

- Attributes, methods and exceptions are properties objects can export to other objects
- These properties determine the type of an object
- Several objects can export the same properties (same type of objects)
- The type is defined once
- The object type is defined by the interface specification of the object

Frank Eliassen, SRL & Ifi/UiO

18

## Declaration of remote methods

- A remote method is declared by its signature
- In CORBA the signature consists of
  - a name
  - a list of **in**, **out**, and **inout** parameters
  - a return value type
  - a list of exceptions that the method can raise
- **void select (in Date d) raises (AlreadySelected);**

Frank Eliassen, SRL & Ifi/UiO

19

## RPC/RMI invocation semantics: design choices

- Reliability semantics of RPC/RMI under partial failures

Fault tolerance measures			Invocation semantics
Retransmit request message on timeout?	Duplication filtering	Re-execute method <i>or</i> retransmit Reply	
No	-	-	Maybe
Yes	No	Re-execute method	At-least-once
Yes	Yes	Retransmit Reply	At-most-once

Frank Eliassen, SRL & Ifi/UiO

20

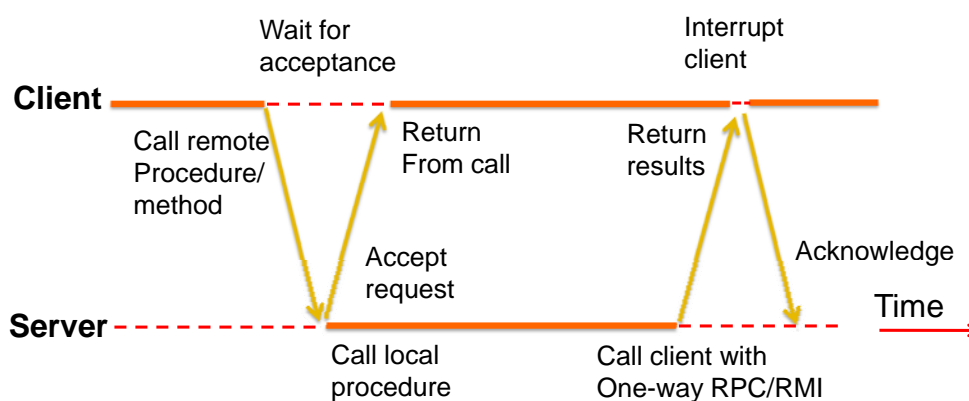
## RMI invocation semantics in object and RPC middlewares

- RMI in CORBA and Java have “at-most-once” invocation semantics under partial failures
  - referred to as synchronous requests
- CORBA allows other forms of synchronization that provides other invocation semantics
  - One-way operations: maybe-semantics
  - Deferred synchronous RMI
- SUN RPC: at-least-once semantics

Frank Eliassen, SRL & Ifi/UiO

21

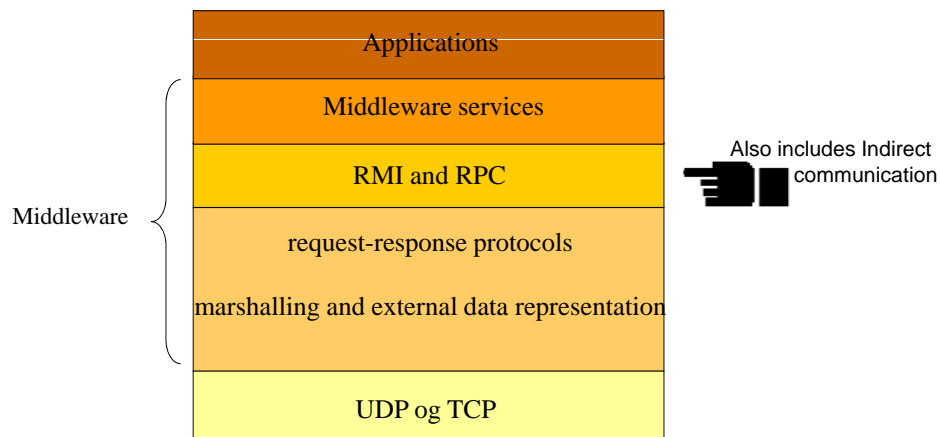
## Deferred synchronous RPC (also applicable to RMI)



Frank Eliassen, SRL & Ifi/UiO

22

## Communication between distributed objects



Frank Eliassen, SRL & Ifi/UiO

23

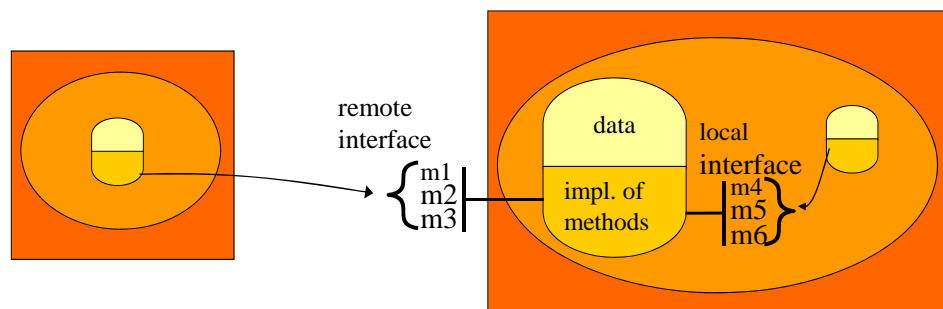
## Remote method invocations

- A client object can request the execution of a method of a distributed, remote object
- Remote methods are invoked by sending a message (including method name and arguments) to the remote object
- The remote object is identified and located using the remote object reference (ROR)
- Clients must be able to handle exceptions that the method can raise

Frank Eliassen, SRL & Ifi/UiO

24

# Remote object with remote interface



Frank Eliassen, SRL & Ifi/UiO

25

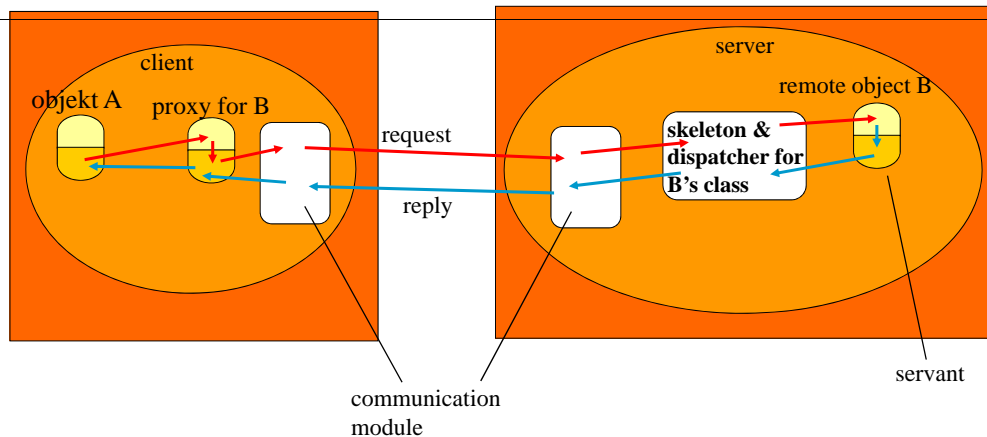
# Implementation of RMI

- Three main tasks :
- Interface processing
  - Integration of the RMI mechanism into a programming language.
  - Basis for realizing access transparency
- Communication
  - message exchange (request-reply protocol)
- Object location, binding and activation
  - Locate the server process that hosts the remote object and bind to the server
  - Activate an object-implementation
  - Basis for realizing location transparency

Frank Eliassen, SRL & Ifi/UiO

26

## RMI interface processing: role of proxy and skeleton



Frank Eliassen, SRL & Ifi/UiO

27

## Elements of the RMI software (I)

### ➤ RMI interface processing: Client proxy

- Local "proxy" object for each remote object a client holds a ROR ("stand-in" for remote object).
- The class of the proxy-object has the same interface as the class of the remote object. Can perform type checking on arguments
- Performs marshalling of requests and unmarshalling of responses
- Transmits request-messages to the server and receive response messages.

Frank Eliassen, SRL & Ifi/UiO

28

## Elements of the RMI software (II)

### ➤ RMI interface processing: Dispatcher

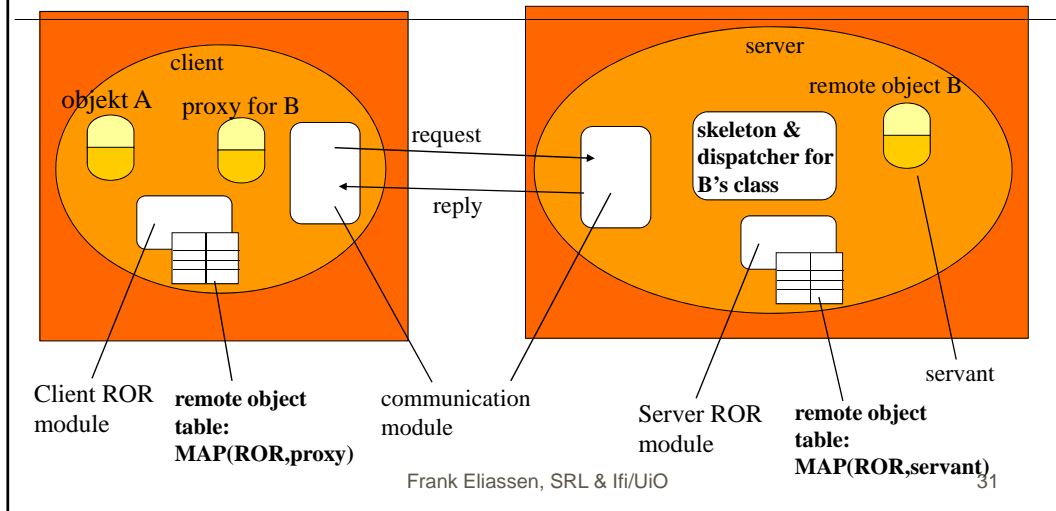
- A server has one dispatcher for each class representing a remote object.
- Receives requests messages
- Uses *method id* in the request message to select the appropriate method in the skeleton (provides the methods of the class) and passes on the request message

## Elements of the RMI software (III)

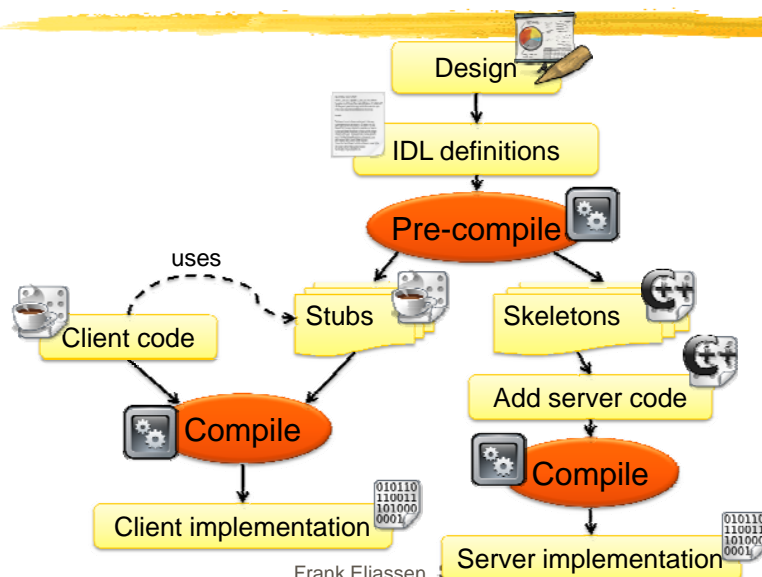
### ➤ RMI interface processing: Skeleton

- A server has one skeleton for each class representing a remote object
- Provides the methods of the remote interface
- A skeleton method unmarshals the arguments in the request message and invokes the corresponding method in the remote object.
- It waits for the invocation to complete and then marshals the result, together with any exceptions, in a reply message to the sending proxy's method.

# RMI interface processing: role of remote object reference module



# Generation of proxies, dispatchers and skeletons



32



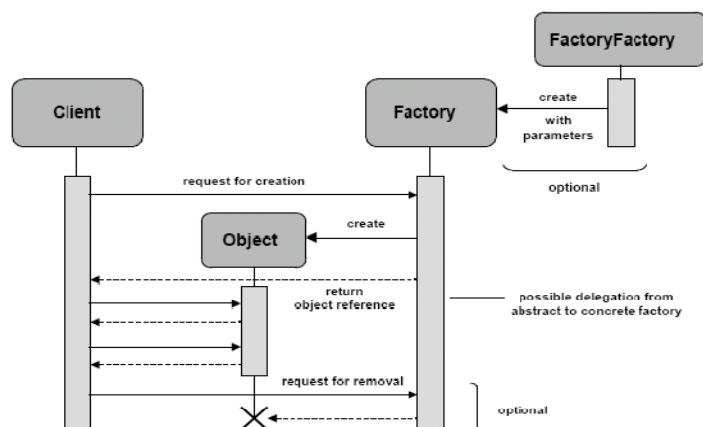
# Server and client programs

- Server program contains
  - the classes for the dispatchers and skeletons
  - the implementation classes of all the servants that it supports
  - an initialization section: creates and initializes at least one servant
    - additional servants (objects) may be created in response to client requests
  - register zero or more servants with a *binder*
  - potentially one or more *factory methods* that allow clients to request creation of additional servants (objects)
- Client program contains
  - the classes and proxies for all the remote objects that it will invoke

Frank Eliassen, SRL & Ifi/UiO

33

# Factory pattern for creating additional objects



source: S. Krakowiak, Middleware Architecture with Patterns and Frameworks,  
<http://sardes.inrialpes.fr/~krakowia/MW-Book/>

Frank Eliassen, SRL & Ifi/UiO

34

# RMI name resolution, binding, and activation

## ➤ Name resolution

- corresponds to mapping a symbolic object name to an ROR
- performed by a name service (or similar)

## ➤ Binding in RMI

- corresponds to locating the server holding a remote object based on the ROR of the object and placing a proxy in the client process's address space

## ➤ Activation in RMI

- corresponds to creating an active object from a corresponding passive object (e.g., on request). Performed by an *activator*
  - register passive objects that are available for activation
  - activate server processes (and activate remote object within them)

Frank Eliassen, SRL & Ifi/UiO

35

# Locating the server of a remote object

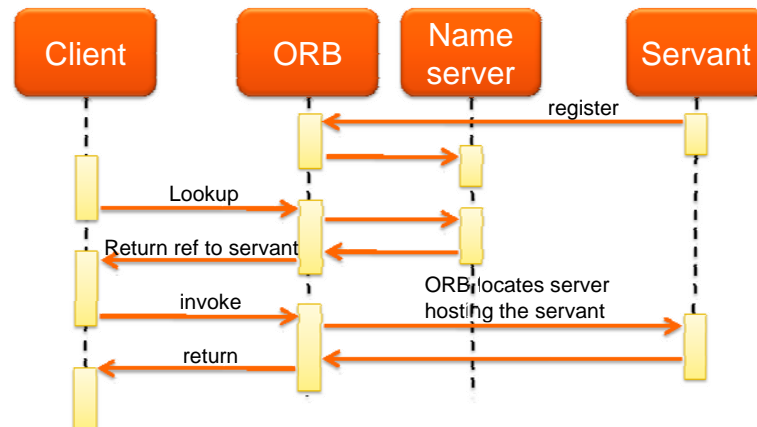
## ➤ Corresponds to mapping an ROR to a communication identifier.

- integrated in ROR
  - Address can be extracted directly from the object reference
- location service
  - A location service is used by the client proxy at each request
- cache/broadcast
  - Each client has cache of bindings (ROR, comm. identifier)
  - If ROR not in cache, perform broadcast with ROR
  - Servers that host the object respond with comm.identifier
- forward pointers or address hint (to e.g., location service)
  - Used at object migration
- Combinations of the above

Frank Eliassen, SRL & Ifi/UiO

36

## Remote method invocation



source: S. Krakowiak, Middleware Architecture with Patterns and Frameworks,  
<http://sardes.inrialpes.fr/~krakowia/MW-Book/>  
Frank Eliassen, SRL & Ifi/UiO

37

## Implicit and explicit binding


```
Distr_object* obj_ref;           // Declare a system wide object reference
obj_ref = lookup(obj_name);      // Initialize the reference to a distrb. obj
obj_ref->do_something();         // Implicit bind and invoke method
```

```
Distr_object* obj_ref;           // Declare a system wide object reference
Local_object* obj_ptr           // Declare a pointer to a local object
obj_ref = lookup(obj_name);      // Initialize the reference to a distrb. obj
obj_ptr = bind(obj_ref);        // Explicitly bind and get pointer to local proxy
obj_ptr->do_something();         // Invoke a method on the local proxy
```

Frank Eliassen, SRL & Ifi/UiO

38

# Plan

- Request-response protocols
- Characteristics of distributed objects
- Communication between distributed objects (RMI)
-  ➤ Object-servers
  - Multi-threaded object servers
- CORBA middleware
- Java RMI

Frank Eliassen, SRL & Ifi/UiO

39

## Object-server: Server tailored to support distributed objects

- Services realized as objects that the server encapsulates
  - Services can be added or removed by creating and removing remote objects
- Object servers act as places where objects can live
- Object servers activate remote objects on demand
  - Several ways to activate an object

Frank Eliassen, SRL & Ifi/UiO

40

## Object servers must assign processing resources to objects when they are activated

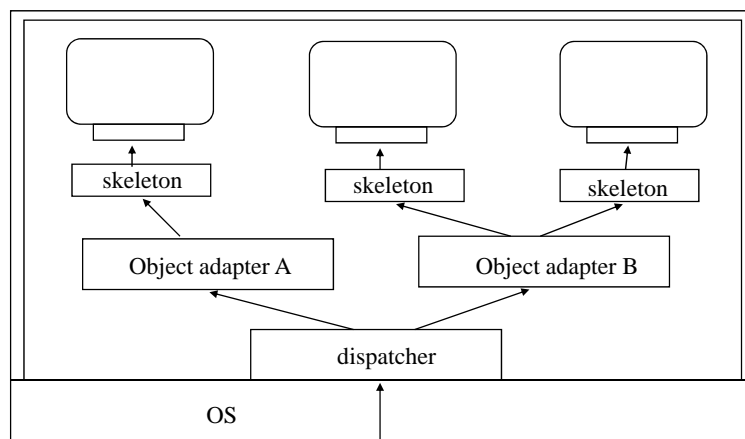
- When an object is activated, which processing resources should be assigned to the implementation?
- Activation policy
  - A particular way of activating an object
  - Different dimensions
    - How to translate between ROR and local implementation?
    - Should the server be single-threaded or multi-threaded?
    - If multi-threaded, how to assign threads to objects and requests? One thread per object? One per request?
    - Transient vs persistent objects, etc
- No single activation policy that fits all needs
  - Object servers should support several concurrent activation policies
  - Objects can be grouped according to which activation policy they are governed by

Frank Eliassen, SRL & Ifi/UiO

41

## Organization of object servers that support different activation policies

- **Object-adapter:** software that implements a specific activation policy (supported by CORBA Portable Object Adapter (POA))



Frank Eliassen, SRL & Ifi/UiO

42

# Object references

## ➤ Remote-object-reference (ROR)

- Identifier for remote objects that is valid in a distributed system
- Must be generated in a way that ensures uniqueness over time and space (=> a ROR can not be reused)
- Example:

Internet address	port number	adapter name	object key	interface of remote object
------------------	-------------	--------------	------------	----------------------------

Frank Eliassen, SRL & Ifi/UiO

43

# Plan

- Request-response protocols
- Characteristics of distributed objects
- Communication between distributed objects (RMI)
- Object-servers
  - Multi-threaded object servers
- CORBA middleware
- Java RMI



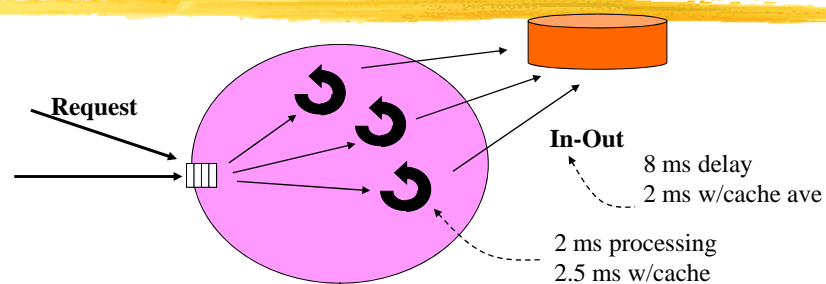
Frank Eliassen, SRL & Ifi/UiO

44

## Object-servers must assign processing resources to objects when objects are activated

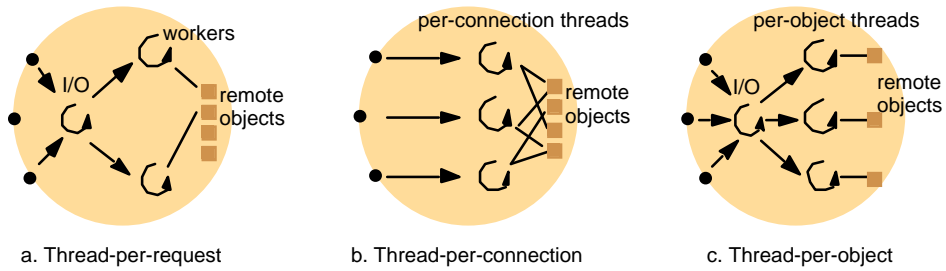
- When an object is activated, which processing resources should be assigned to its implementation?
  - Create a new process or thread?
  - Are there several ways this can be done?
  - Is there a best way (cf. activation policies)?

## Multi-thread server



	# processors	# threads	Max. calls/sec
no disk caching	1	1	100
no disk caching	1	2	125
disk caching	1	2	400
disk caching	2	2	444
disk caching	2	3	500

## Alternative threading-policies for object activation



Frank Eliassen, SRL & Ifi/UiO

47

## Plan

- Principles for realising remote methods invocations (RMI)
- Object-servers
- Multi-threaded object servers
- CORBA RMI
- Java RMI

Frank Eliassen, SRL & Ifi/UiO

48



## ➤ **Common Object Request Broker Architecture (CORBA)**



Frank Eliassen, SRL & Ifi/UiO

49

## CORBA middleware

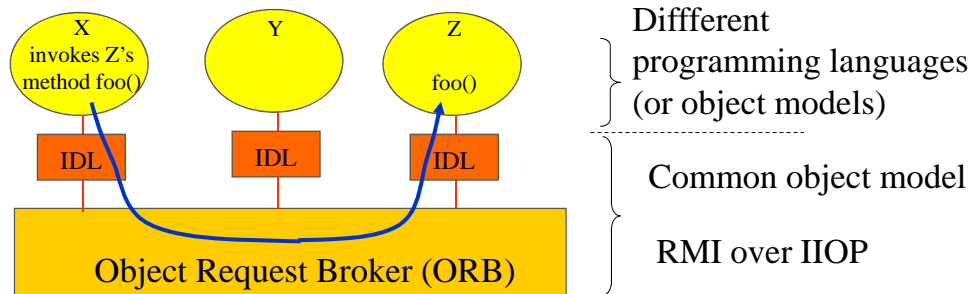
- Offers mechanisms that allow objects to invoke remote methods and receive responses in a transparent way
  - location transparency
  - access transparency
- The core of the architecture is the Object Request Broker (ORB)
- Specification developed by members of the Object Management Group ([www.omg.org](http://www.omg.org))

Frank Eliassen, SRL & Ifi/UiO

50

# CORBA RMI

Clients may invoke methods of remote objects without worrying about:  
object location, programming language,  
operating system platform, communication  
protocols or hardware.



Frank Eliassen, SRL & Ifi/UiO

51

## CORBA supports language heterogeneity

- CORBA allows interacting objects to be implemented in different programming languages
- Interoperability based on a common object model provided by the middleware
- Need for advanced mappings (language bindings) between different object implementation languages and the common object model

Frank Eliassen, SRL & Ifi/UiO

52

# Elements of the CORBA common object model

- Metalevel model for the type system of the middleware
- Defines the meaning of e.g.,
  - object identity
  - object type (interface)
  - operation (method)
  - attribute
  - method invocation
  - exception
  - Subtyping / inheritance
- Must be general enough to enable mapping to common programming languages
- CORBA Interface Definition Language (IDL)

Frank Eliassen, SRL & Ifi/UiO

53

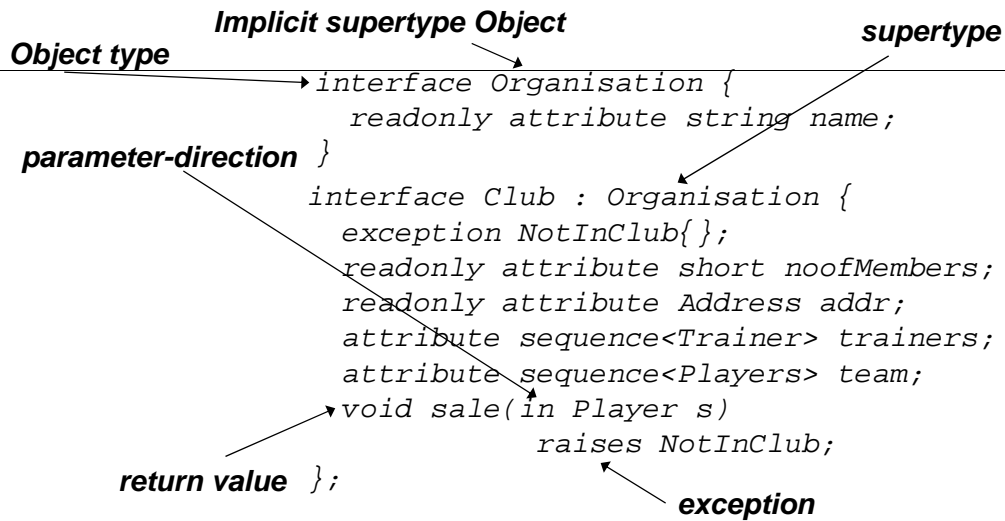
# CORBA IDL

- Language for specifying CORBA object types (i.e. object interfaces)
- Can express all concepts in the CORBA common object model
- CORBA IDL is
  - not dependent on a specific programming language
  - syntactically oriented towards C++
  - not computationally complete
- Different bindings to programming languages available

Frank Eliassen, SRL & Ifi/UiO

54

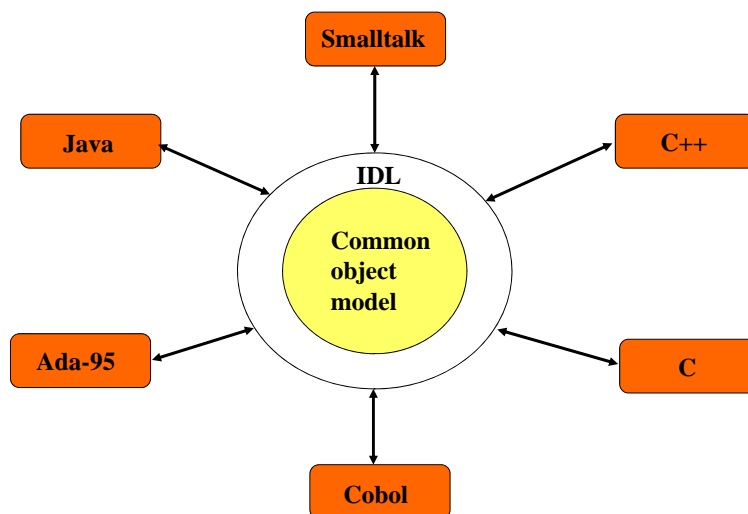
# A CORBA IDL specification



Frank Eliassen, SRL & Ifi/UiO

55

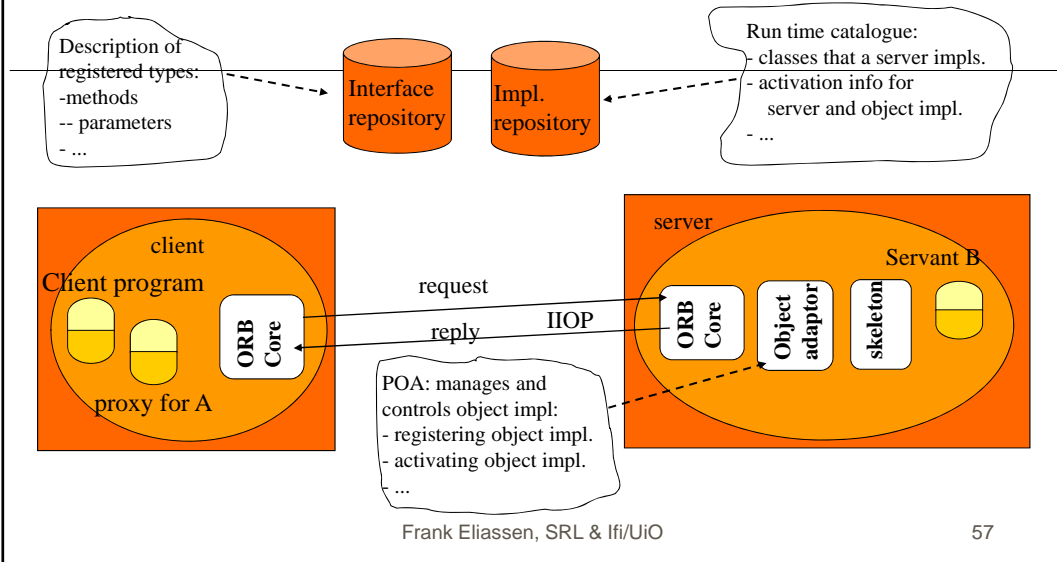
# Language bindings for CORBA IDL



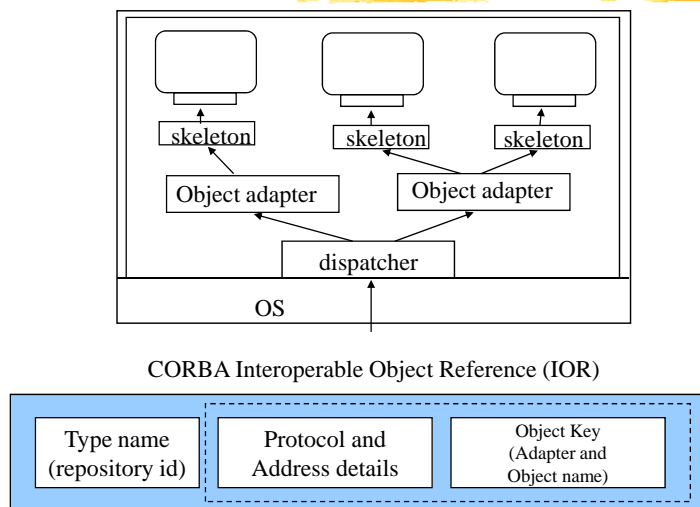
Frank Eliassen, SRL & Ifi/UiO

56

# CORBA architecture

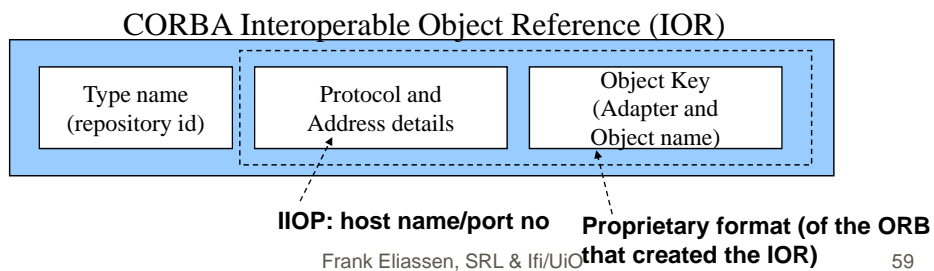


# CORBA ROR format (IOR) reflects the organization of object servers

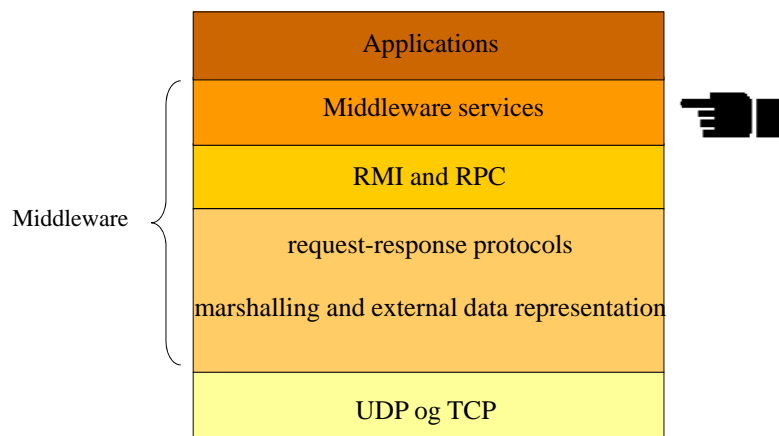


# CORBA RMI binding

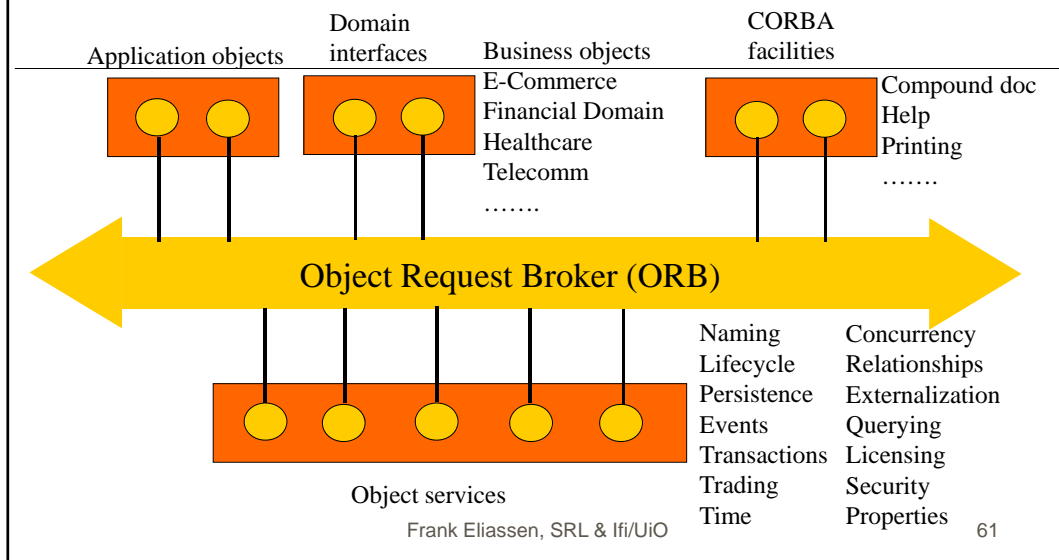
- Binding in RMI corresponds to mapping object references (ROR) to “servants”
  - servant: implementation of one or more CORBA objects
- ROR in CORBA: Interoperable Object Reference (IOR)
- Location process:
  - based on information encoded in the object reference



# Layering of middleware



# CORBA Services



# CORBA binding for Java example

- 1<sup>st</sup> step: define the IDL for the remote methods:

```

module HelloApp {
    interface Hello {
        string sayHello(in string message);
    };
};
    
```

## CORBA binding for Java example

- 2<sup>nd</sup> step: Compile the interface using the IDL compiler for Java (IDLJ):

```
idlj -fall -oldImplBase -td <toFolder> Hello.idl
```

- **fall**: Create client and server code (stub and skeleton)
- **oldImplBase**: Compatibility mode with previous versions. New versions use just POA implementation

Frank Eliassen, SRL & Ifi/UiO

63

## CORBA binding for Java example

- 3<sup>rd</sup> step: Implement the Servant class that must extend the `_*ImplBase` generated class.
- 4<sup>th</sup> step: Implement the Server:
  - Create and initialize the CORBA ORB;
  - Instantiate the servant and register it;
  - Bind the object reference in the Naming service.

Frank Eliassen, SRL & Ifi/UiO

64



## CORBA binding for Java example

### ➤ 5<sup>th</sup> steep: Implement the Client:

- Create and initialize the CORBA ORB in the client side;
- Resolve the remote object name in the naming service;
- Invoke the remote method.

Frank Eliassen, SRL & Ifi/UiO

65

## CORBA binding for Java example

### ➤ To run the application:


- Run the name service with the **tnameserv** command;
- Run the Server java application;
- Run the Client Java application;

Frank Eliassen, SRL & Ifi/UiO

66

## Plan

---

- Principles for realising remote methods invocations (RMI)
- Object-servers
- Multi-threaded object servers
- CORBA RMI
-  ➤ Java RMI

Frank Eliassen, SRL & Ifi/UiO

67

- 
- Java Remote Method Invocation (RMI)



Frank Eliassen, SRL & Ifi/UiO

68

## Java RMI

- Remote Method Invocation (RMI) supports communication between different Java Virtual Machines (VM), and possibly over a network
- Provides tight integration with Java
- Minimizes changes in the Java language/VM
- Works for homogeneous environments (Java)
- Clients can be implemented as *applet* or *application*

Frank Eliassen, SRL & Ifi/UiO

69

## Java Object Model

- Interfaces and Remote Objects
- Classes
- Attributes
- Operations/methods
- Exceptions
- Inheritance

Frank Eliassen, SRL & Ifi/UiO

70

## Java interfaces to remote objects

- Based on the ordinary Java interface concept
- RMI does not have a separate language (IDL) for defining remote interfaces
- Pre-defined interface Remote
- All RMI communication is based on interfaces that extends java.rmi.Remote
- Remote classes implement java.rmi.Remote
- Remote objects are instances of remote class

Frank Eliassen, SRL & Ifi/UiO

71

## Java remote interface: Example

*interface name*      *declares the Team interface as "remote"*

```
interface Team extends Remote {
public:
    String nama()throws RemoteException;
    Trainer[] trained_by() throws RemoteException;
    Club club() throws RemoteException;
    Player[] player() throws RemoteException;
    void chooseKeeper(Date d) throws RemoteException;
    void print() throws RemoteException;
};
```

*remote operation*

Frank Eliassen, SRL & Ifi/UiO

72

## Java RMI parameter passing

- Atomic types transferred *by value*
- Remote objects transferred *by reference*
- None-remote objects transferred *by value*

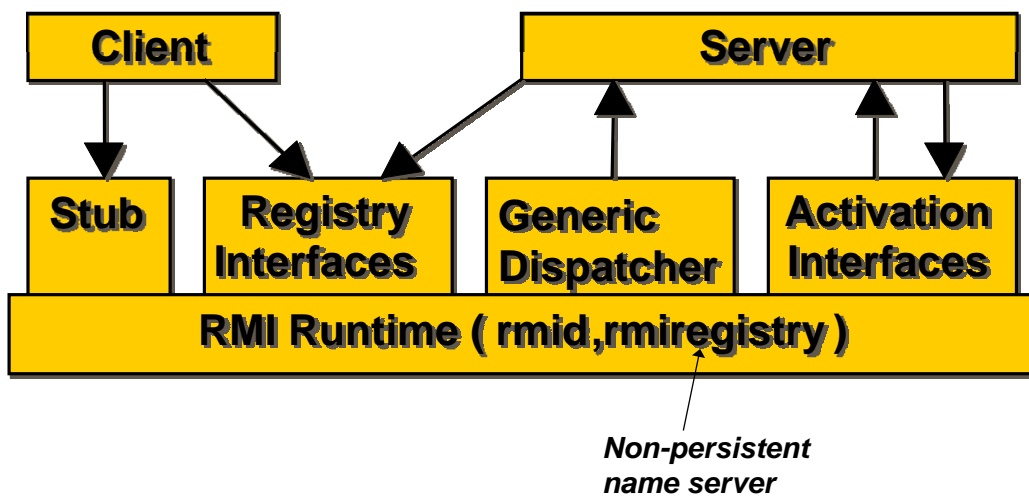
```
class Address {  
    public:  
    String street;  
    String zipcode;  
    String town;  
};  
interface Club extends Organisation, Remote {  
    public:  
    Address addr() throws RemoteException;  
    ...  
};
```

*Returns a copy of the Address-object*

Frank Eliassen, SRL & Ifi/UiO

73

## Architecture of Java RMI



Frank Eliassen, SRL & Ifi/UiO

74

## Summary - I

- Request-response protocols
- Distributed objects executes in different processes.
  - remote interfaces allow an object in one process to invoke methods of objects in other processes located on the same or on other machines
- Object-based distribution middleware:
  - middleware that models a distributed application as a collection of interacting distributed objects (e.g., CORBA, Java RMI)

Frank Eliassen, SRL & Ifi/UiO

75

## Summary - II

- Implementation of RMI
  - proxies, skeletons, dispatcher
  - interface processing, binding, location, activation
- Invocation semantics (under partial failure)
  - maybe, at-least-once, at-most-once
  - Reliability of RMI is at best "at-most-once"
- Multi-threaded servers
  - can in some cases be used to increase the throughput (method calls/time unit) if, e.g., I/O is the bottleneck
- Principles of CORBA
  - Clients may invoke methods of remote objects without worrying about: object location, programming language, operating system platform, communication protocols or hardware.
- Principles of Java RMI
  - Similar to CORBA but limited to a Java environment

Frank Eliassen, SRL & Ifi/UiO

76