Lecture on

Distributed Objects

Prof. Walter Kriha,
HdM Stuttgart

Objective

This lecture intends to explain the object paradigm:

- seamlessly integrating remote calls into an OO-Application
- keeping object semantics across servers and systems

The lecture also intends to explain the price to be paid for this transparent integration and its final limitations.

A Remote Object Call

Java client

```
Bank bank = new Bank(“MyBank”);
Money m = new Money();
m = bank.withdraw(“kriha”, 50);
System.out.println(“balance” +
bank.getBalance(“kriha”);
```

C server

```
withdraw(“MyBank”,“kriha”, 50);
Return m;
……

Return getBalance(“MyBank”, “kriha”);
```

On the client side it looks like a regular and local object method. In reality, something gets transmitted to a remote machine and some answer is returned. On the server side, no real OO-language needs to be involved. But object-semantics need to be preserved (e.g. state)

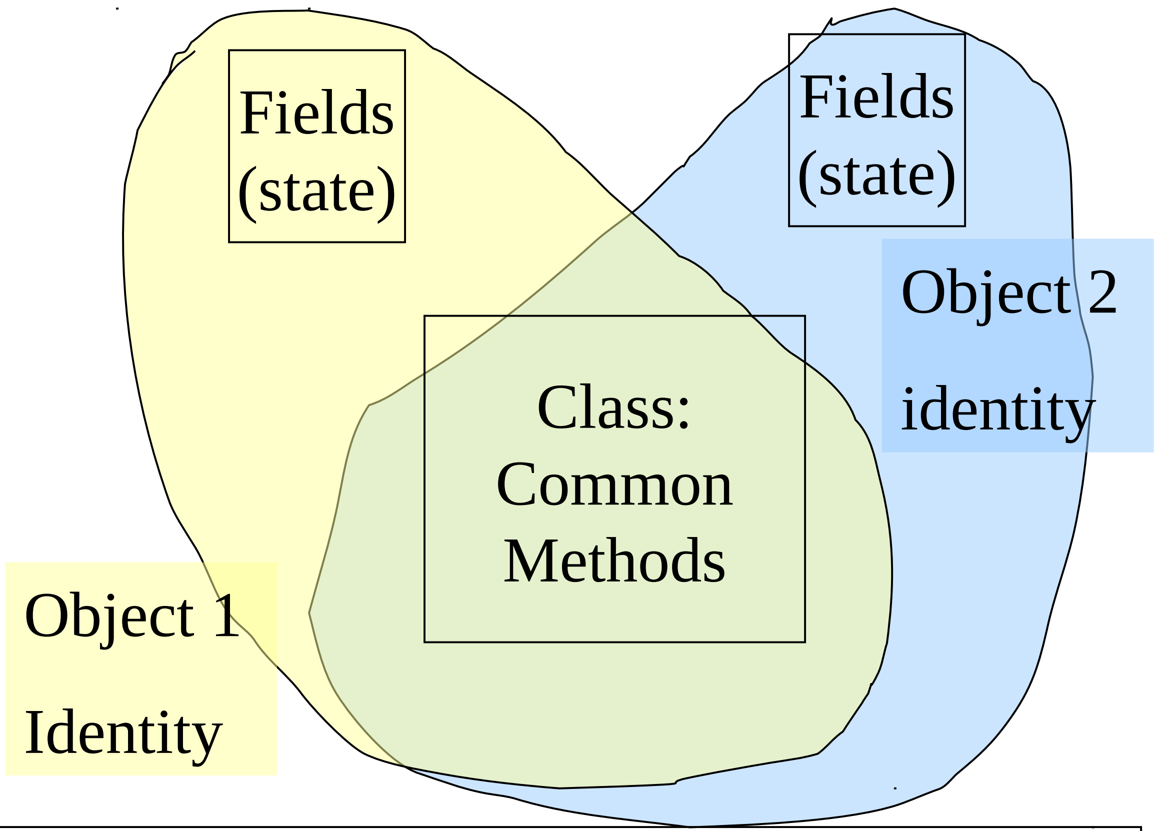
Overview

Fundamental Properties of Objects
Local and remote object references
Parameter passing
Object invocation types
Distributed Object Services
Object Request Broker Architectures
Interface Design
Example Java RMI

Objects vs. Abstract Data Types

Global or module data (optional)

Stateless Procedures (RPC)



Objects are supposed to have individual STATE and IDENTITY (named state). This is the fundamental difference to e.g. procedure calls. Are stateless objects useful? What does keeping state mean for a server? And how does an object-reference work in DS?

Fundamental Object Properties

- local objects are created with new()
- a local object reference is returned and can be used to call methods
- the object reference serves as a locally unique identity
- an object “belongs” to its creator who controls access to it
- objects hold state as long as the VM is alive and the objects in use
- objects have a lifecycle that usually ends with their creator
- objects have fine-granular interfaces and methods
- creation of objects is cheap
- objects can be small and many

“Objects” in an OO-language seem to have lots of properties which might be hard to keep in a concurrent and remote implementation!!

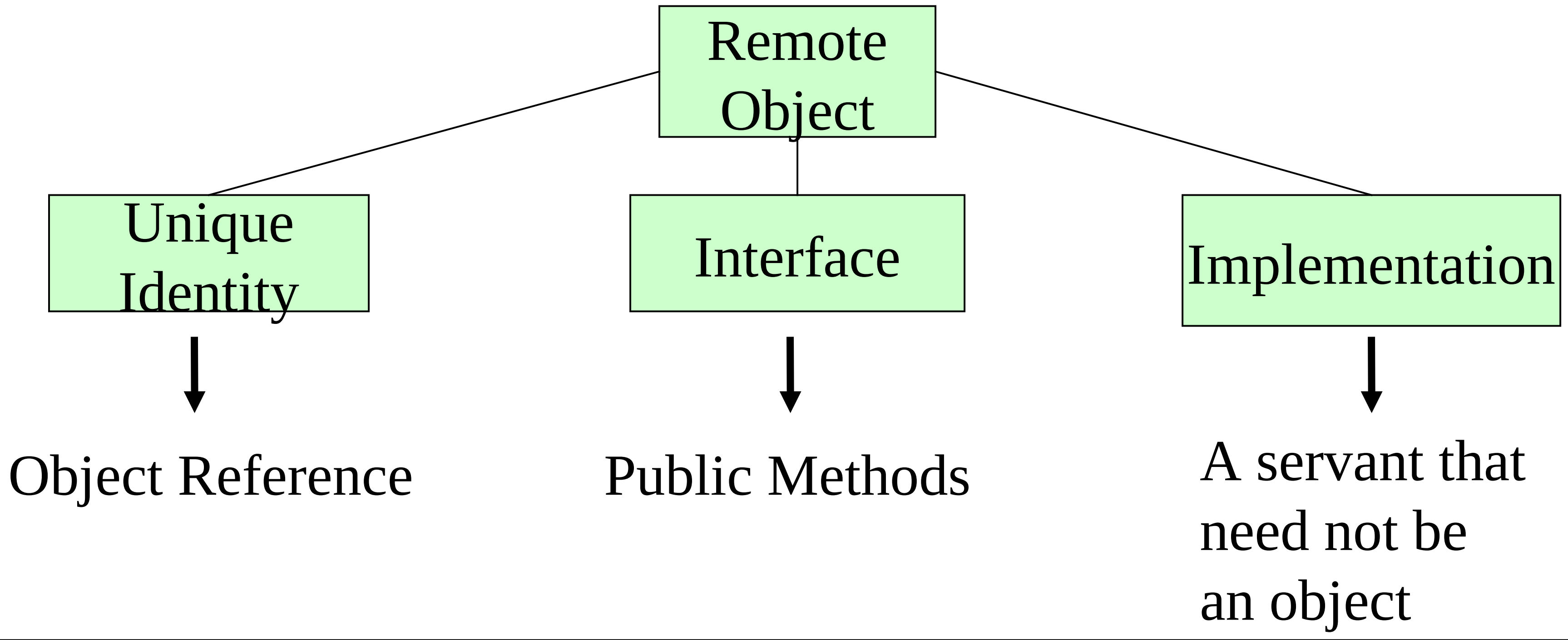
Remote Objects: some tough questions!

- Object identity is usually only valid locally (e.g. a memory address). This is fairly useless for a remote client. What is the identity of a RO?
- Who creates ROs? New() on the client won't cut it.
- How do clients get access to/find ROs?
- Who controls concurrent access to ROs?
- How long do ROs live in a server?
- Where do ROs go, when the server dies? Do clients lose their objects in that case?
- What happens to the state of a RO?
- What happens to a RO, when a client dies?
- How much does a RO cost (latency)?
- Does a RO have the same interface as a local object?

Keeping up object semantics across machines can become quite expensive and difficult...

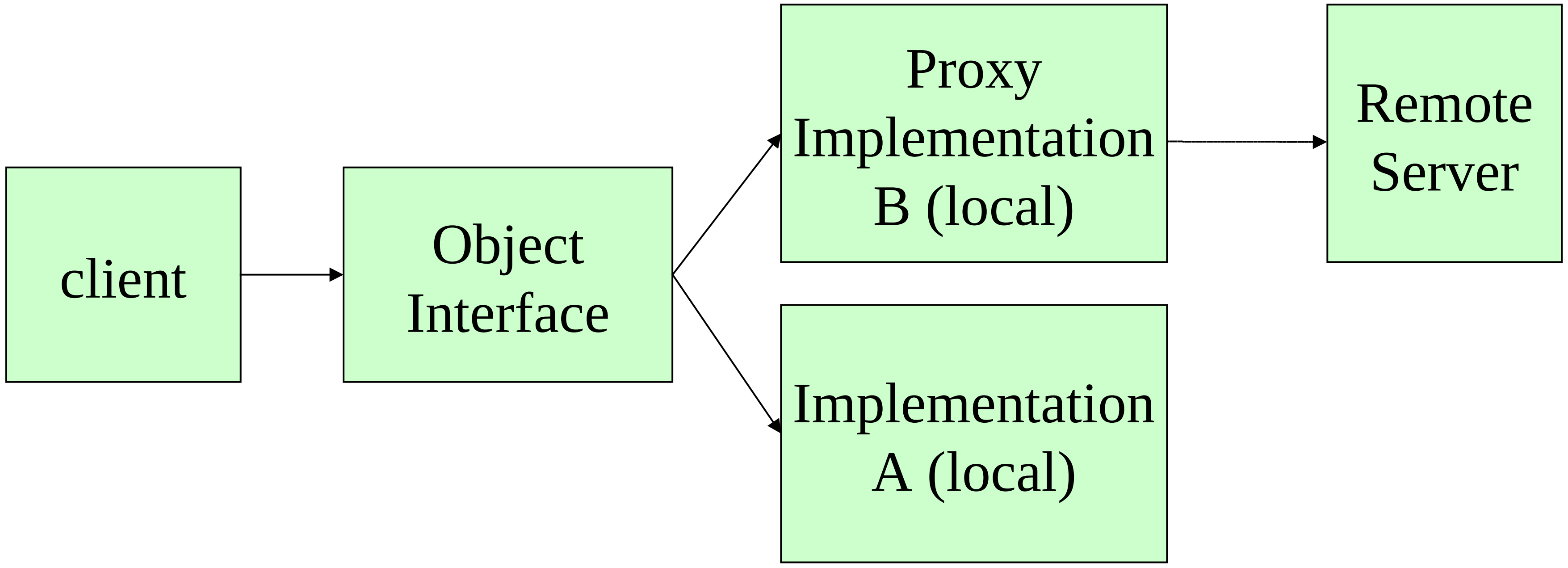
Remote Objects and Object References

What is a Remote Object?



A remote object is the combination of system-wide unique identity, interface and implementation with the special twist that clients KNOW the interface, USE the identity but do NOT KNOW about the implementation. To clients, the interface IS the implementation.

The Trick: Interface vs. Implementation



As long as the client only works with an interface, the implementations can change without breaking the client (or so the OO-theory says….). This promises complete transparency of remote calls behind object interfaces. But how far does this transparency go?

Object Model and Type System

CORBA

- Basic Types: sequence, string, array, record, enumerated, union
- Value Objects (Data)
- Remote Object References

(reference semantics)

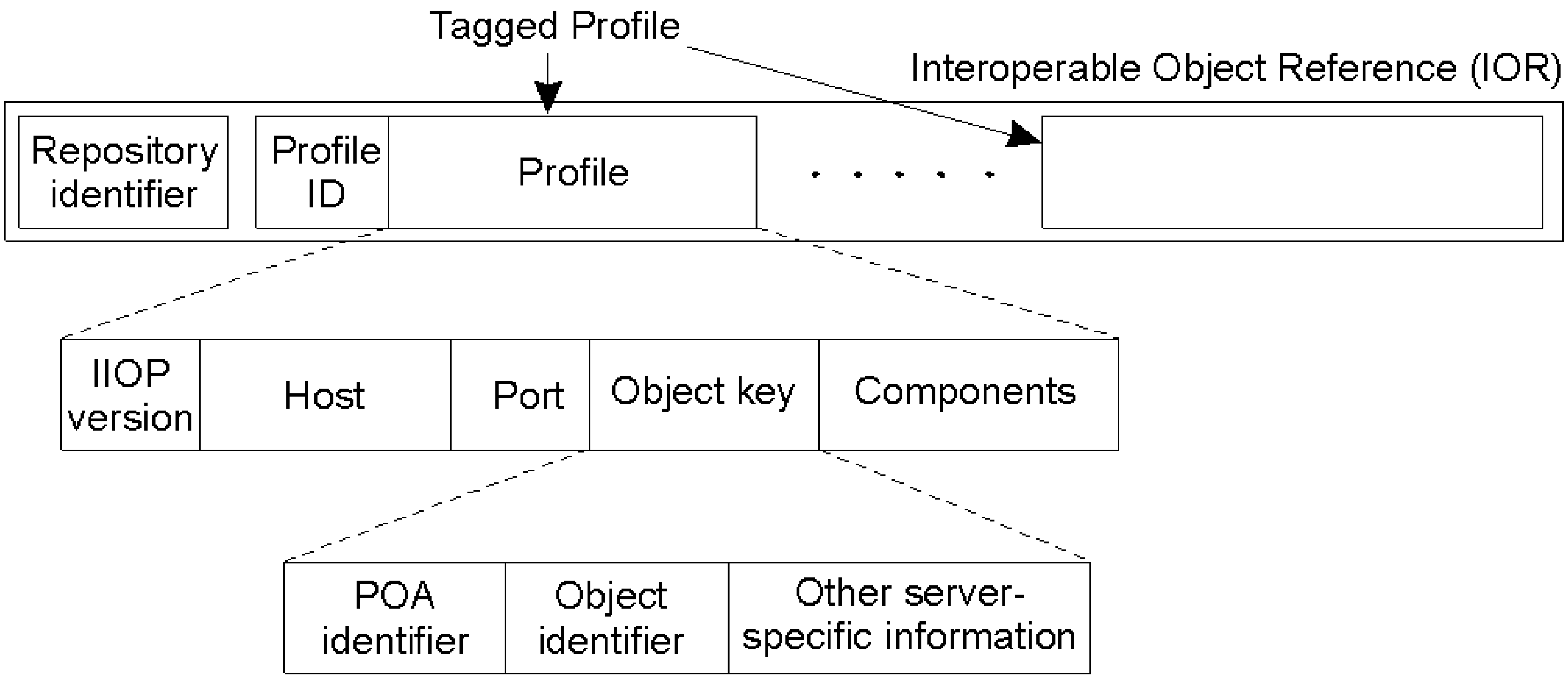
Java-RMI

- Basic Types of language (int, byte etc.)
- Serializable non-remote Objects (value semantics)
- Remote Object References

(reference semantics)

To achieve language independence, CORBA defines its own types. Nothing else can show up in the interfaces of remote objects. Especially no user defined classes. Only compositions of basic types and the OR. Note that Java-RMI allows classes if they are serializable!

Example: CORBA Remote Object Reference



The organization of an OR. (from van Steen, Tanenbaum). All the information a client needs to call an object. All the information a server needs to create/manage the object.

How do clients get access to remote objects?

- A Naming Service (like a directory)
- A web server (serialized OR)
- Via mail or a piece of paper
- From another remote object which serves as a “Factory”.

This really is the question: where do clients get the Remote Object Reference for a remote object from?

Supporting Middleware

- Broker Pattern and Architecture
- Remote Object Reference Construction
- Invocation of Remote Method Calls
- Interceptor/Filter Pattern
- Distributed Object Services

Interface Definition Language (IDL)

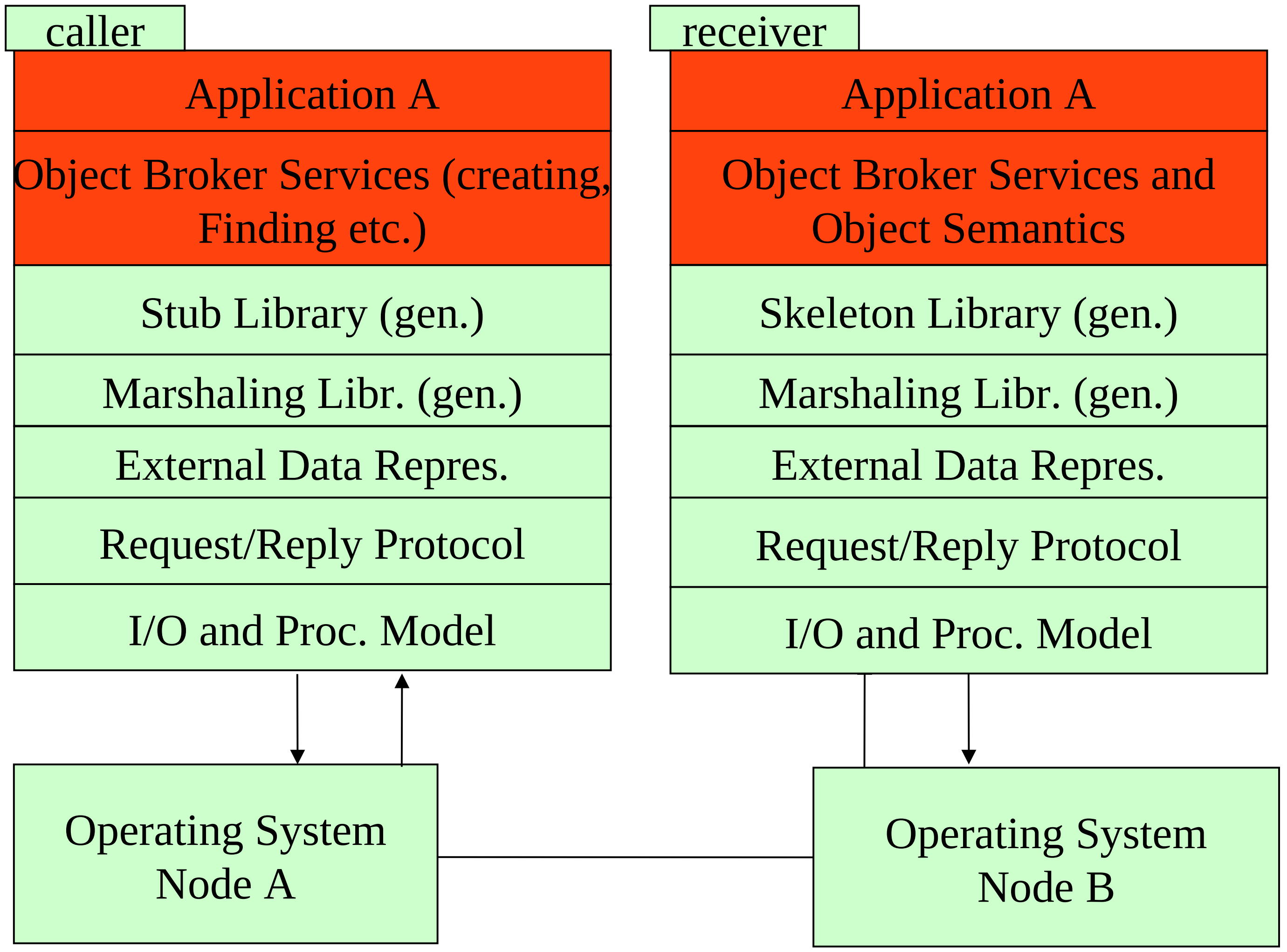
```
Module Count
{
    interface Count
      { attribute long sum;
        long increment();
      };
}
```

From: Orfali, Harkey, Client/Server Programming with Java and CORBA

CORBA's way to specify language independent remote objects. An interface is part of a type. It describes the externally visible part of an object. A class implements a type (and its interface).

Please note: CORBA has no real notion of a "class" in the OO sense!

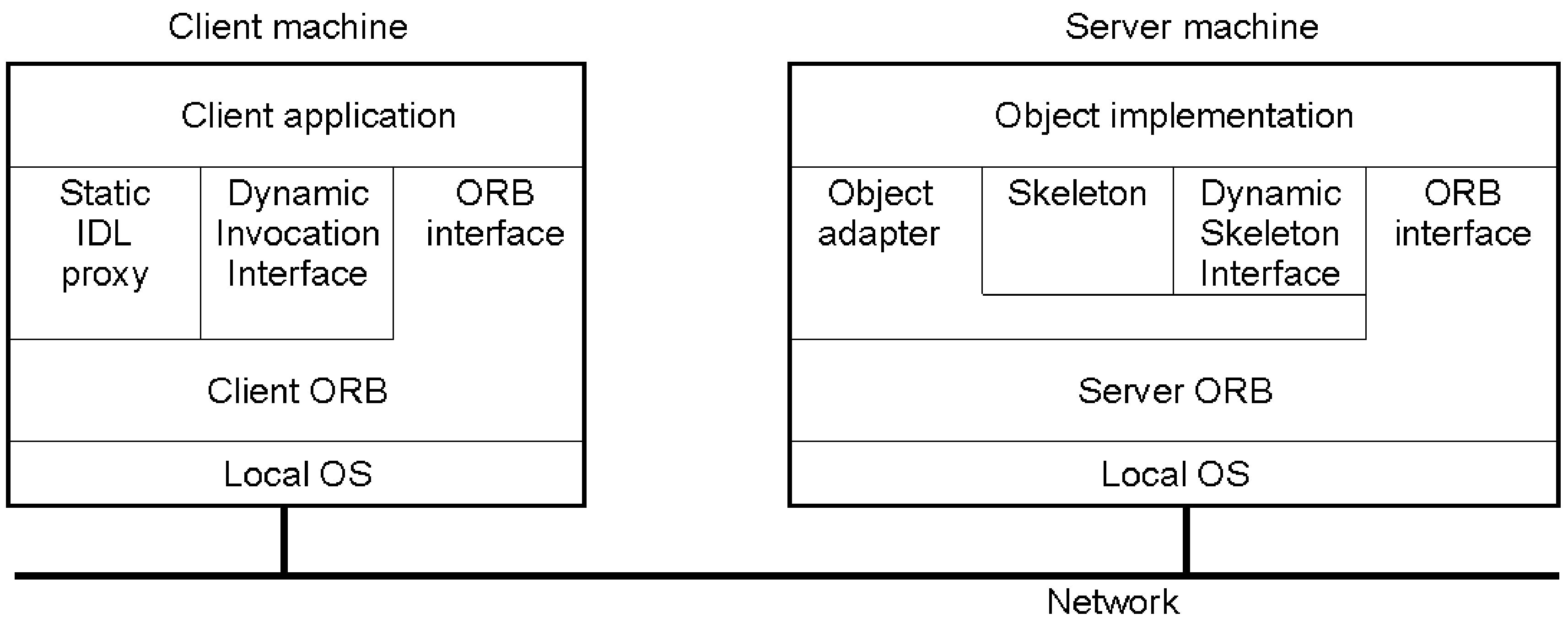
Distributed Objects



Just like a regular RPC!

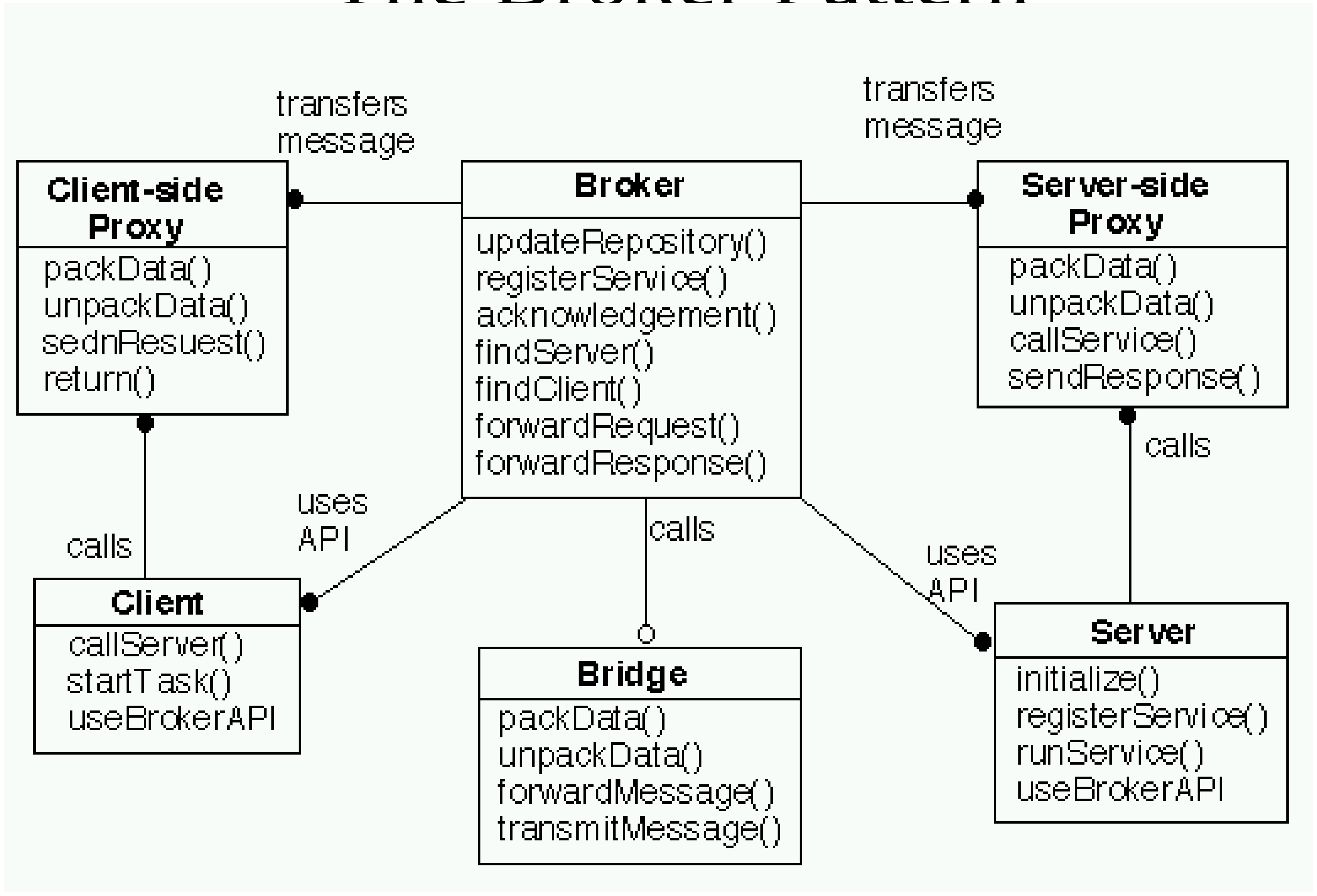
The main components of Distributed Objects. Not shown is the processing framework (threading, async. Etc.). Stub/skeleton libraries are generated from interface definitions.

CORBA System Architecture



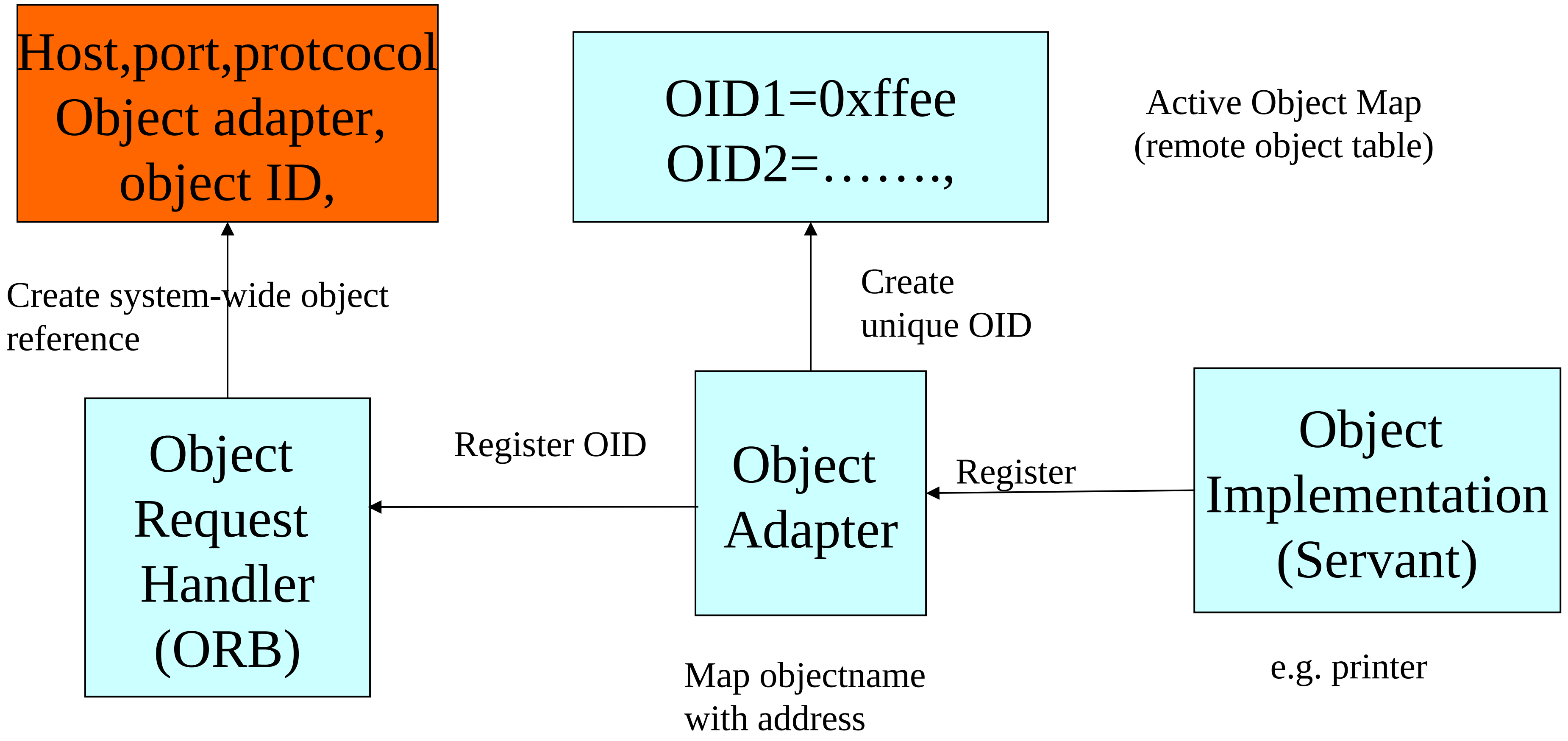
The orb interface solves bootstrap problems (e.g. where to get initial object references) and string2object/object2string conversions. (from van Steen, Tanenbaum). Not shown: Interface Repository, Naming Service etc.

The Broker Pattern



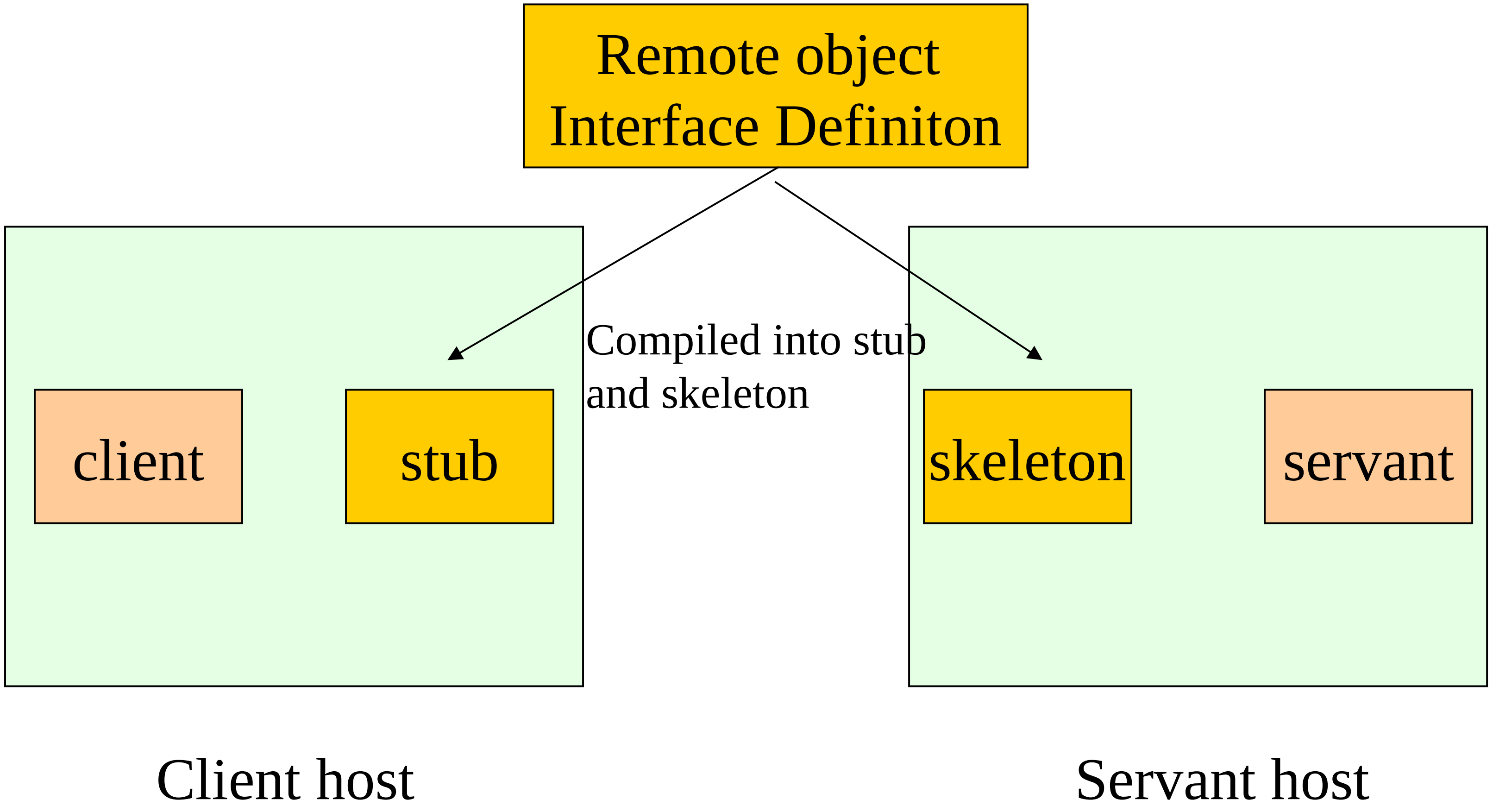
A broker introduces and mediates communication between de-coupled entities. (diagram from: http://www.eli.sdsu.edu/courses/spring04/cs635/)

Construction of an Remote Object Reference


Host,port,protcocol Object adapter, object ID,
OID1=0xffee OID2=…….,
Active Object Map (remote object table)
Create system-wide object reference
Create unique OID
Object Request Handler (ORB)
Register OID
Object Adapter
Register
Object Implementation (Servant)
Map objectname with address
e.g. printer

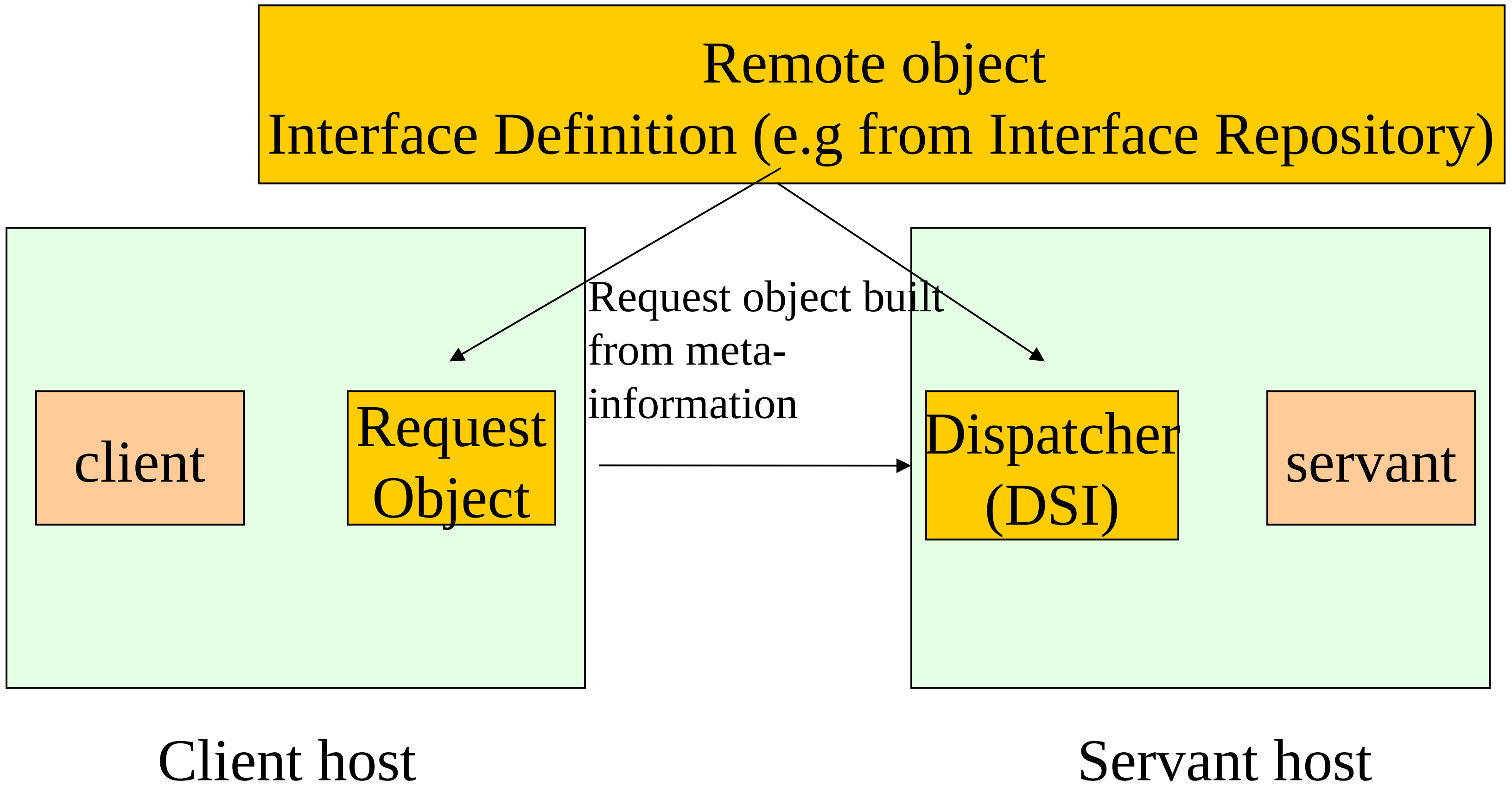
The new object reference can be marshaled and sent to a client. The construction of the server object/implementation can be lazy!
Object adapters manage large number of similar objects (like customers, orders etc.)

Static Remote Method Invocation



Stubs may be statically linked or dynamically downloaded. Clients KNOW the remote object interface IN ADVANCE of the use! The mechanism is identical to RPCs.

Dynamic Invocation


Remote object
Interface Definition (e.g from Interface Repository)
Request object built from meta-information
client
Request Object
Dispatcher (DSI)
servant
Client host
Servant host

Clients fill in a request object (built from meta-information of the remote object) and send it to a dispatcher on the servant host. The servant does not know that the request was dynamically built. The mechanism is similar to the reflection pattern.

Asynchronous Invocations

Most invocations are synchronous, the client waits for the results from the servant. Three asynchronous types are frequenty used:

a) one-way calls (they cannot have return values or out parameters. Delivery guarantee is best-effort.

b) Deferred synchronous (client continues and later checks for results (blocking). At-most-once delivery.

c) True asynchronous with server callbacks (server sees difference between sync. And async. Calls): Needs messaging middleware to achieve at-most-once delivery guarantees.

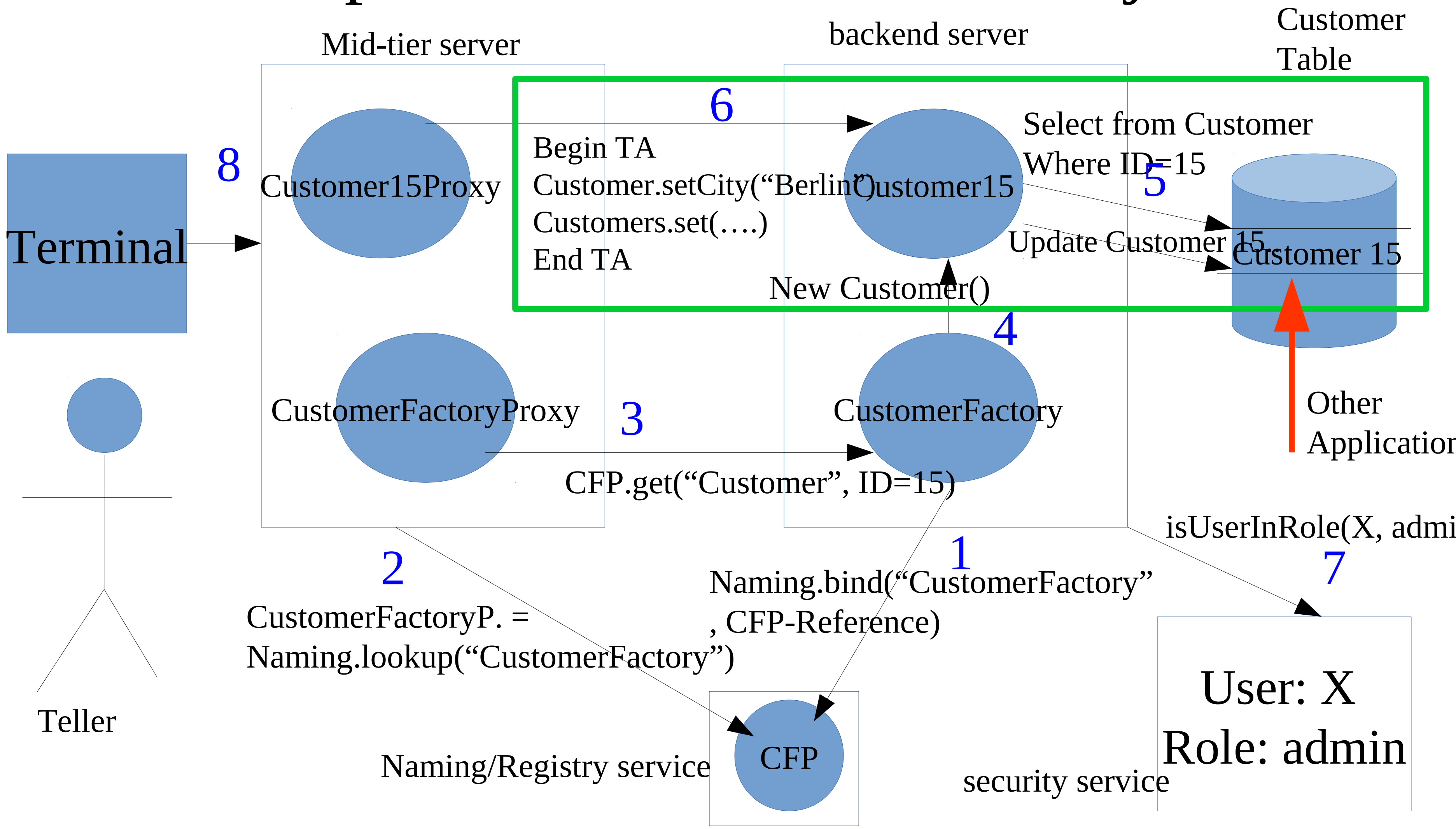
It depends on the implementation whether calls are really asynchronous (the client disconnects and the server later on builds a new connection to the client) or simulated (client continues but one client thread blocks waiting for the synchronous response from the server (In this case the server does not see a difference between sync. And async. Calls)

Main Distributed Object Services

- Finding Objects
  - Naming service (maps names to references),
  - Trading service (object offer services, clients search by constraint)
- Preserving Object State:
  - Persistence service to store object state transparently (and load it on demand)
  - Transaction Service: preserve object consistence across changes (several objects, distributed, nested or flat)
  - Concurrency Service: provide locks for shared objects
  - Security Service: check roles of principals
- Grouping of Objects, Collections

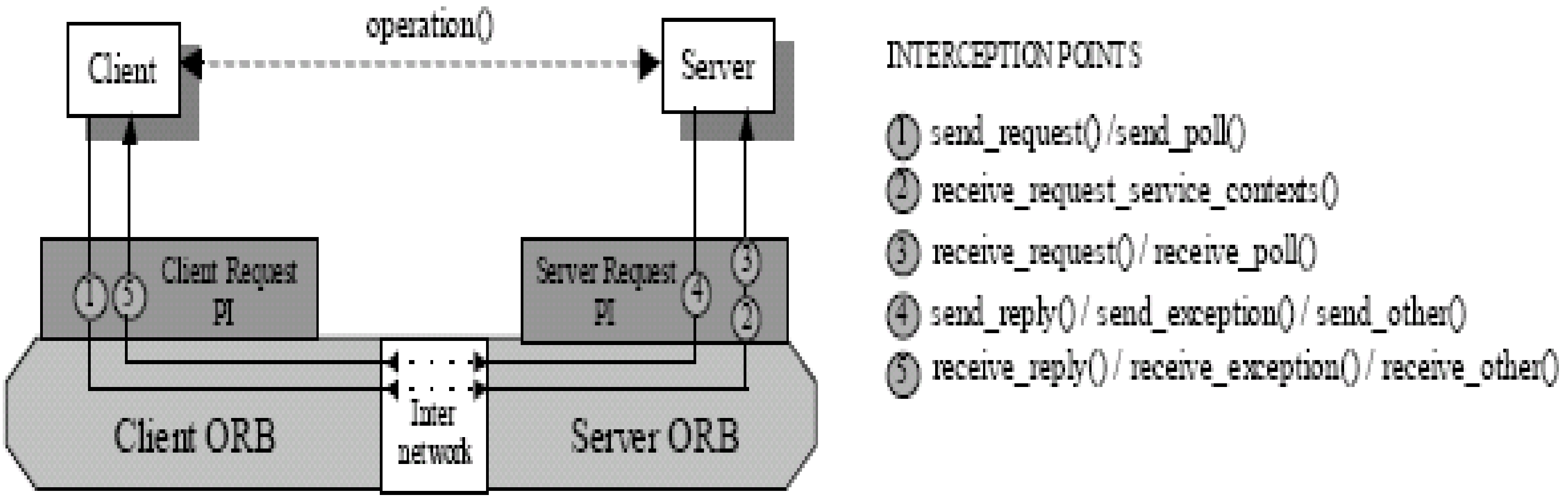
Same principle today: cloud collections: https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-reliable-services-reliable-collections

Example: Remote Customer Object



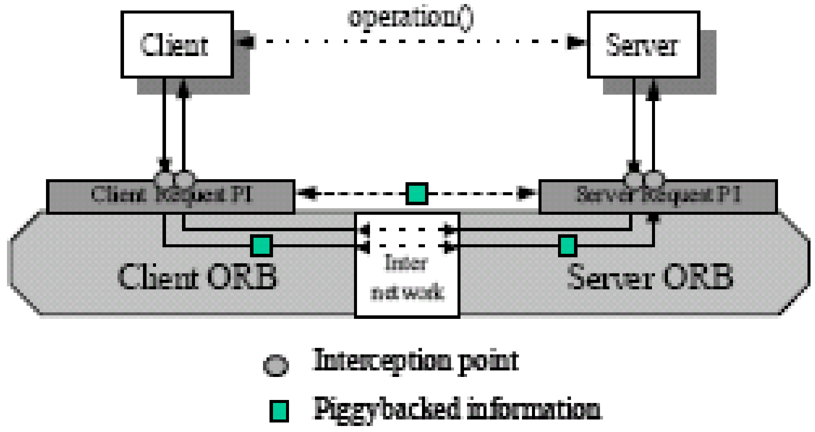
The illusion of object behavior across nodes is costly: it needs finding objects (naming service), modifying objects (transactions and locking) and saving object state (persistence service) and last but not least access control (security service). If you take it all together and automate it, you get Enterprise Java Beans. Without the special services and without automation you get Java RMI.

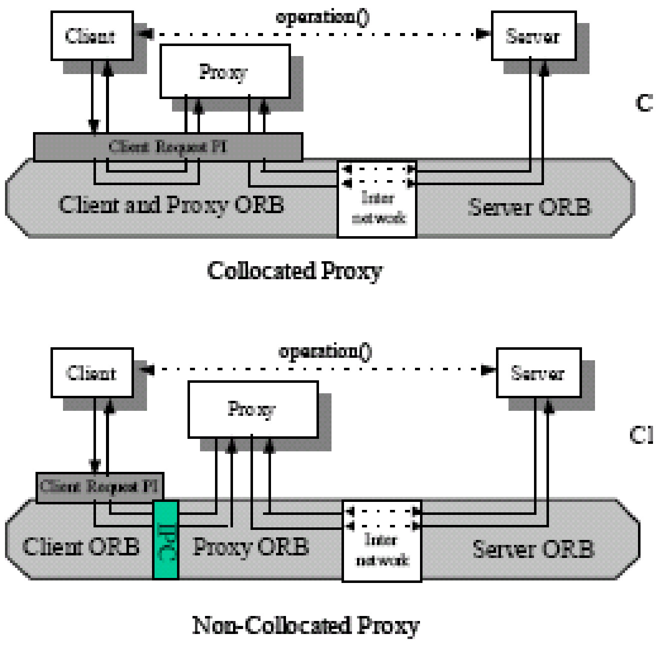
Filtering: Portable Interceptors 1


operation()
Client
Server
Client Request PI
Server Request PI
1
5
4
3
2
Client ORB
Inter network
Server ORB
INTERCEPTION POINTS
1 send_request() /send_poll()
2 receive_request_service_contexts()
3 receive_request() / receive_poll()
4 send_reply() / send_exception() / send_other()
5 receive_reply() / receive_exception() / receive_other()

By intercepting calls additional (context) information can be added and transported transparently between ORBs. The original protocol already knows security and transaction contexts but applications can define custom context information. The same technique is used in servlet filters, Aspect Oriented Programming etc. Diagram from Marchetti et.al, see resources)

Portable Interceptors 2





two ways to realize interception and call extension. Note that interceptors need to have access to the original call information and that subtle ordering problems can occur. Application Servers use this method e.g. to realize delegation of security information to backends. (from Marchetti et.al, see resources)

Remote Interface Design

Exercise 1: State, Concurrency, Performance

```
Interface Stack {
Push (object);
Object Pop();
Boolean IsEmpty();
}
```

- Is this a reasonable interface for an remote object?
- would you use exceptions? Where?
- does concurrency make a difference in this case!
- Would you add methods for performance reasons? Which ones?

Exercise 2: Granularity and Concurrency

```
Interface Address  {
        setStreet(String);
        setHouseNumber(String);
        setCity(String);
        set ZipCode(String);
}
```

- **How would you protect an object instance of Interface Address against concurrent use?**
- **How would a server instantiate such an instance? In case of client failures?**
- **Performance improvements?**

Exercise 3: API

```
Interface Foo {
      init();
      doIt(String);
      reset();
      }
```

- Objects of Interface foo: how are they initialized? Who does it?

- Design various use cases for this protocol. Which one is the most likely one?

Remote Interface Design

- Respect the possibility of concurrent calls in your interface design: Do not keep inconsistent state across method calls.
- Do not perform staged initialization e.g. the infamous “half-baked object” anti-pattern (B.Scheffold).
- Don’t use complicated or unclear orders of calls for your interface (what comes first? Shake hands with the king or kiss the queens hand?)

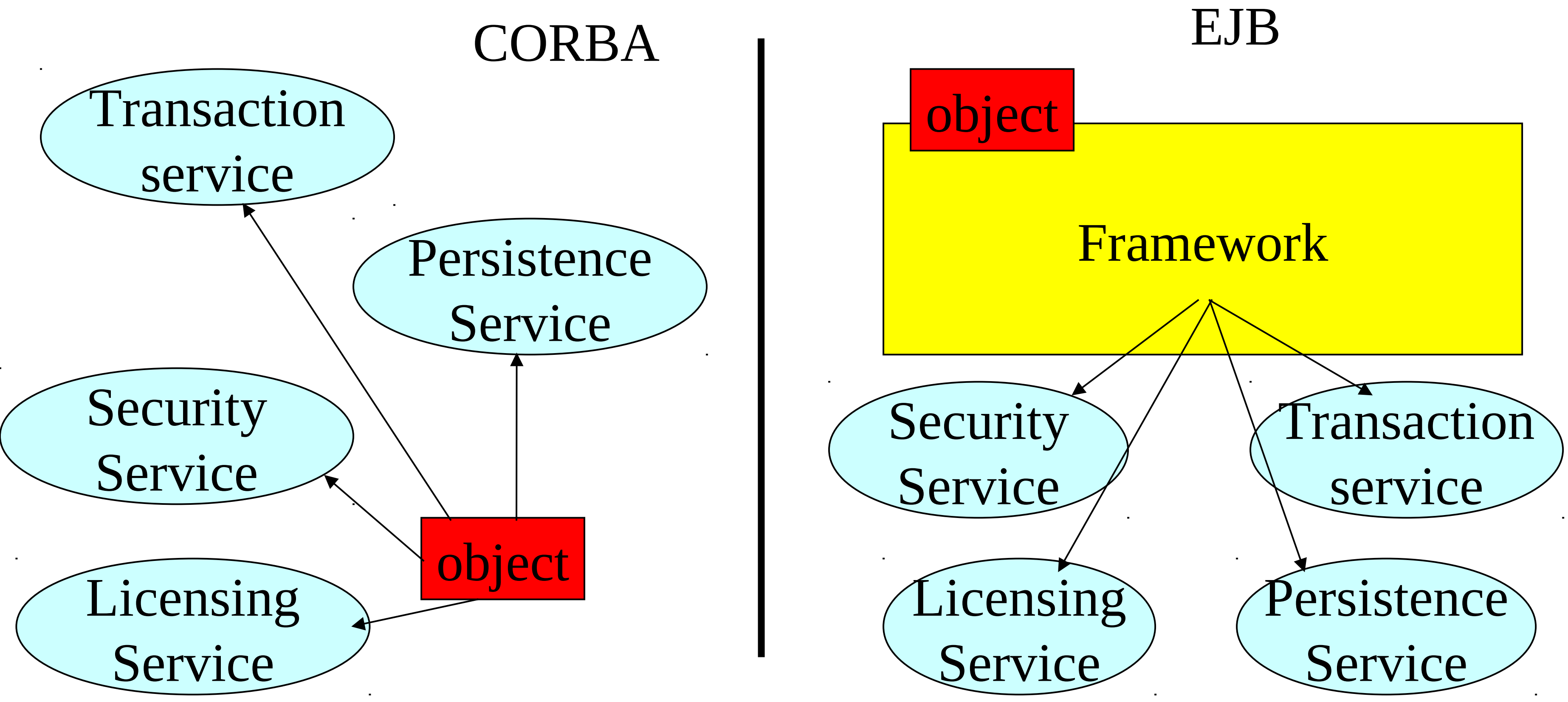
Have a look at: Mowbray, Malveau, Corba design patterns. The patterns are important for other object based middleware too.

Interfaces for Remote Objects have to be different from object models for local applications! This is because of concurrency (state) and performance (granularity of calls). You cannot re-use a local model for distributed applications!!!!!

The big problems of Remote Objects

- Interfaces: too granular and therefor slow
- No direct support for state handling on servers
- Bad for “data schlepping” applications (too expensive)
- Cross-language calls expensive to build
- No security in calls
- No transaction support

Distr. Obj. Services vs. Components



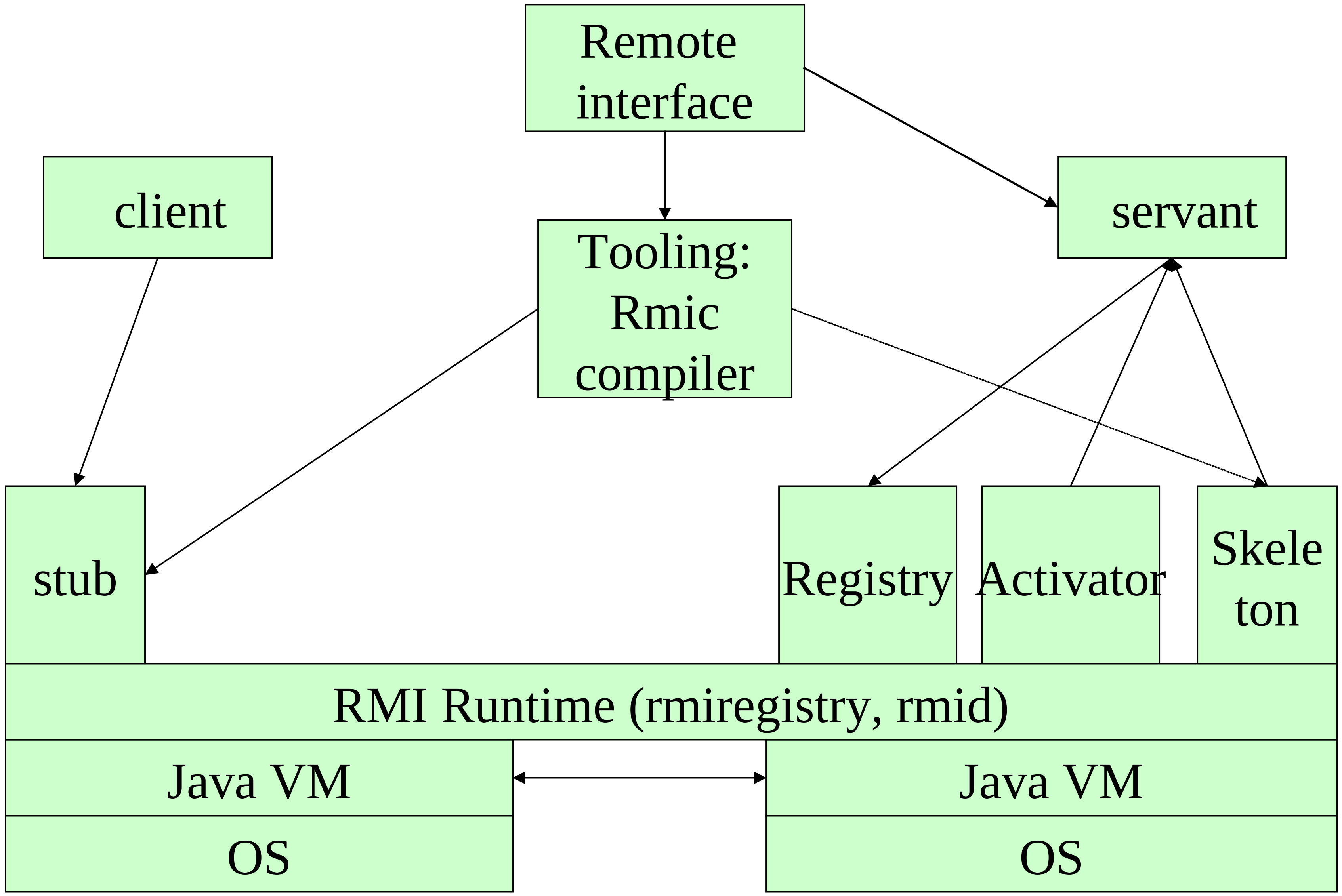
Service based code turned out to be less re-usable as expected. By letting the framework call services, objects don’t need to know WHAT services to call or WHEN!

Java Remote Method Invocation (RMI)

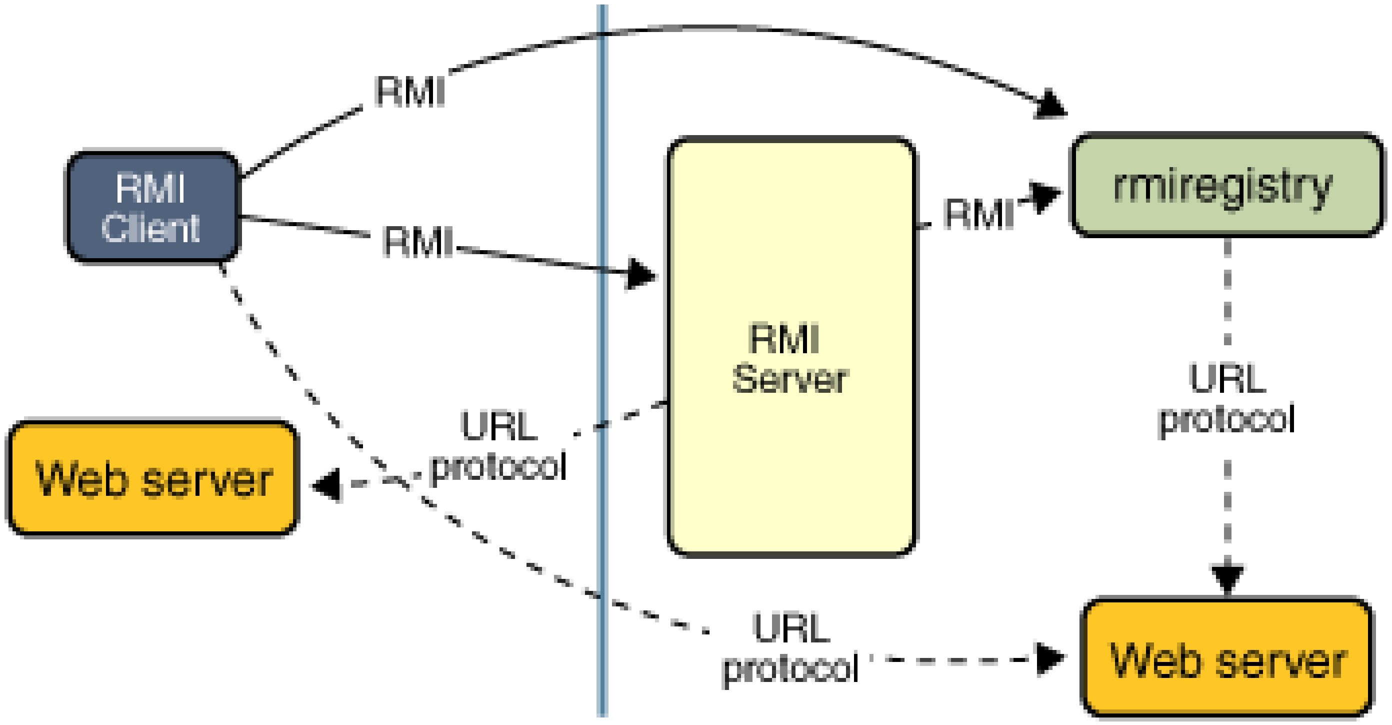
Java Remote Method Invocation (RMI)

- Architecture
- Request/reply protocol
- External data representation
- Parameter passing conventions
- Stub/Skeleton generation
- Important classes and tools
- Remote Object Registration
- Remote Object Activation
- Object finding/client bootstrap
- Garbage Collection
- New developments in RMI
- Specials

Java-RMI System Architecture (old)



RMI System (new)



From Oracle RMI tutorial (http://docs.oracle.com/javase/tutorial/rmi/overview.html).
Exported classes are loaded dynamically. RMIC no longer used.

Java RMI Request/Reply Protocols (1)

- JRMP: first protocol for RMI.
  - Bandwidth problems due to distributed garbage collection with short term leases and permanent reference counting
  - Dynamic download of code

If you want to understand garbage collection: read Paul Wilson's seminal paper on GC (see resources)

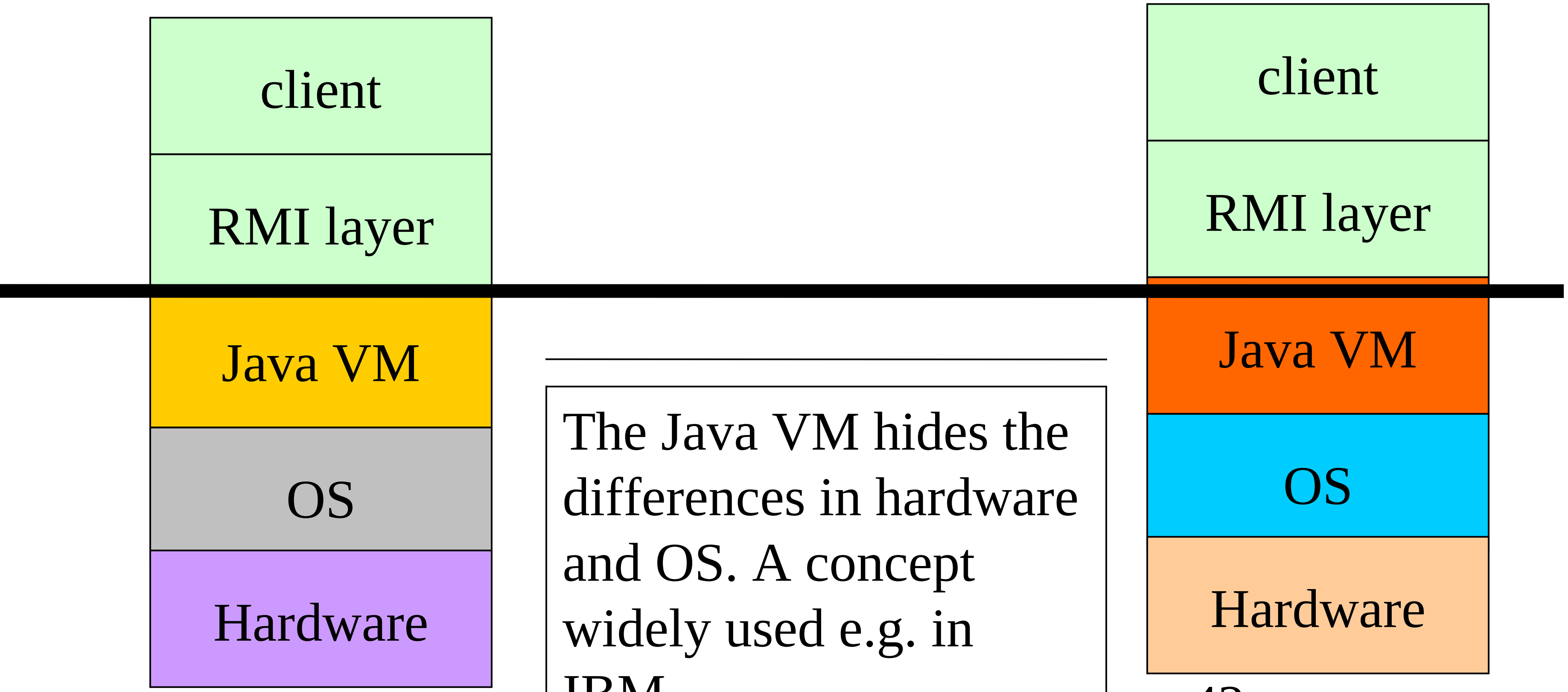
Java RMI Request/Reply Protocols (2)

- RMI-IIOP: RMI over CORBA’s Internet Inter-Orb Protocol
  - uses Java Naming and Directory Interfaces (JNDI) to lookup object references (EJB etc. all use it too). Persistent!
  - Requires code changes: PortableRemoteObject
  - Need to generate/define code for IIOP (rmic –iiop xxxxx)
  - Need to generate IDL file(s) for CORBA systems (rmic –idl JavaInterfaceName (Interface!!)
  - Code shipping?

Advantage: Move the idl file of your Java Remote Object Interface to a CORBA system, generate the CORBA stub with its IDL compiler and now the CORBA system can call your remote object. Or call into CORBA systems by yourself.

Java RMI External Data Representation

Who cares? Java RMI is only between Java objects and the virtual machine protects applications and middleware from hardware differences!



The Java VM hides the differences in hardware and OS. A concept widely used e.g. in IBM

Java RMI parameter passing rules

Parameter	Atomic	Non-remote (serialized)	Remote Object
Local Call	Call by Value	Call by Reference	Call by Reference
Remote Call	Call by Value	Call by Value	Call by Reference

Note that CORBA e.g. for a long time did not support serialized value objects.

Java RMI Stub/Skeleton Generation

- RMI connects implementations, not interfaces: you must run the stub generator (rmic) on the implementation class!
- Stubs can be dynamically downloaded from the registry or from a web-server
- Skeletons are generated dynamically on the server side using reflection. (need to learn about meta-object protocols and reflection? Read Gregor Kiczales, Andreas Paepcke: Open Implementations and Metaobject Protocols. A free and very easy tutorial available on the web)

What happens if you downloaded a stub to a client and it gets changed afterwards? New RMI versions allow dynamic generation and download of implementation code. Be careful to avoid mixing remote and local class sources (codebase parameter)

Java RMI: Important Classes and Tools

Remote	Remote Object Interfaces extend this class (tag interface)
RemoteException	All Remote Object methods throw this class
Naming	Clients use it to find remote object references, Servers register their objects with it.
UnicastRemoteObject	Remote Object Implementations extend it
rmic	Stub/skeleton/idl file generator
registry	Simple name server for java objects

Java Remote Interface Example

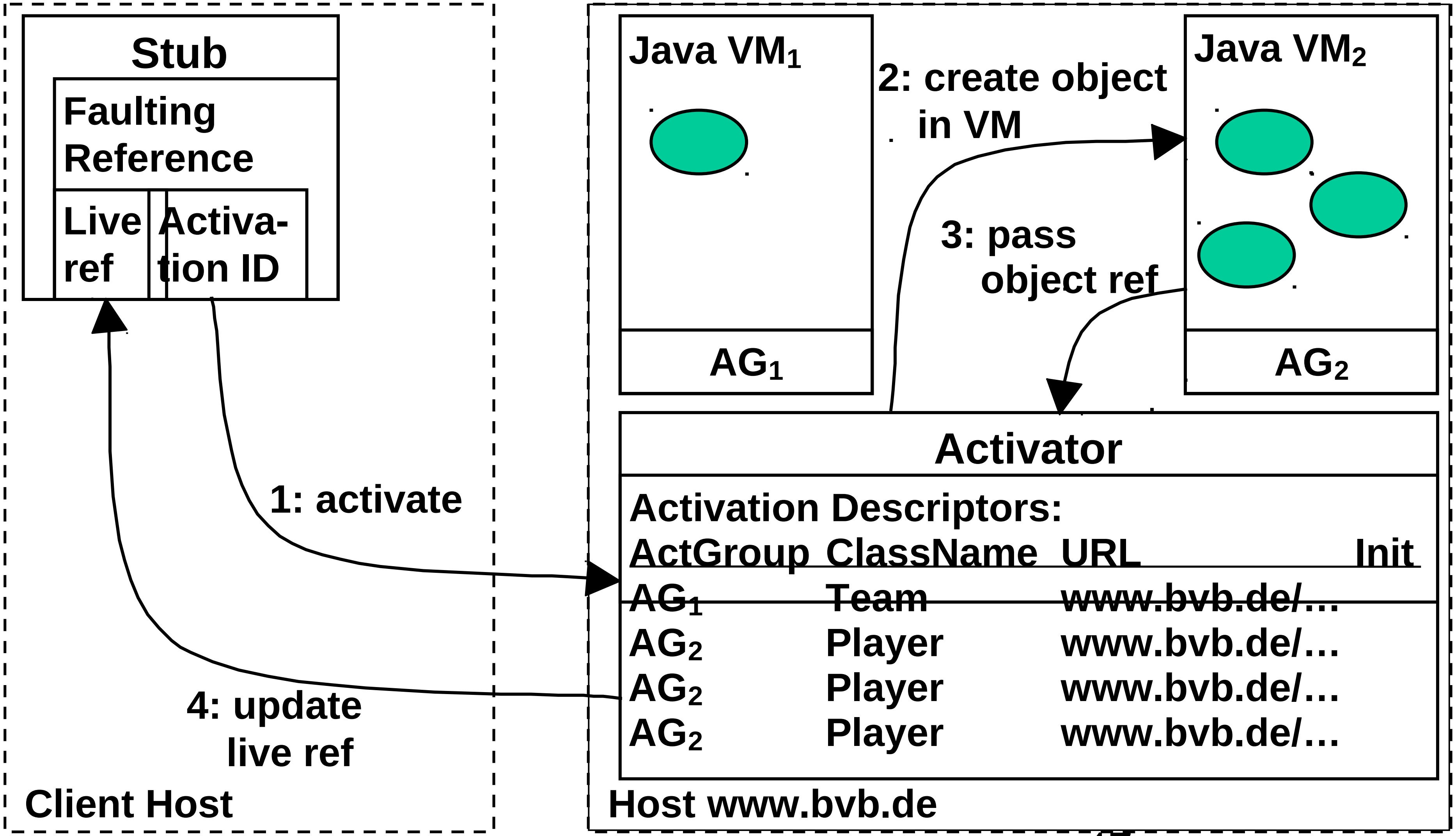
Package name   ***Interface name***

Declare it as remote

```
package soccer;
interface Team extends Remote {
public:
 String name() throws RemoteException;
 Trainer[] coached_by() throws RemoteException;
 Club belongs_to() throws RemoteException;
 Players[] players() throws RemoteException;
 void bookGoalies(Date d) throws RemoteException;
 void print() throws RemoteException;
};
```

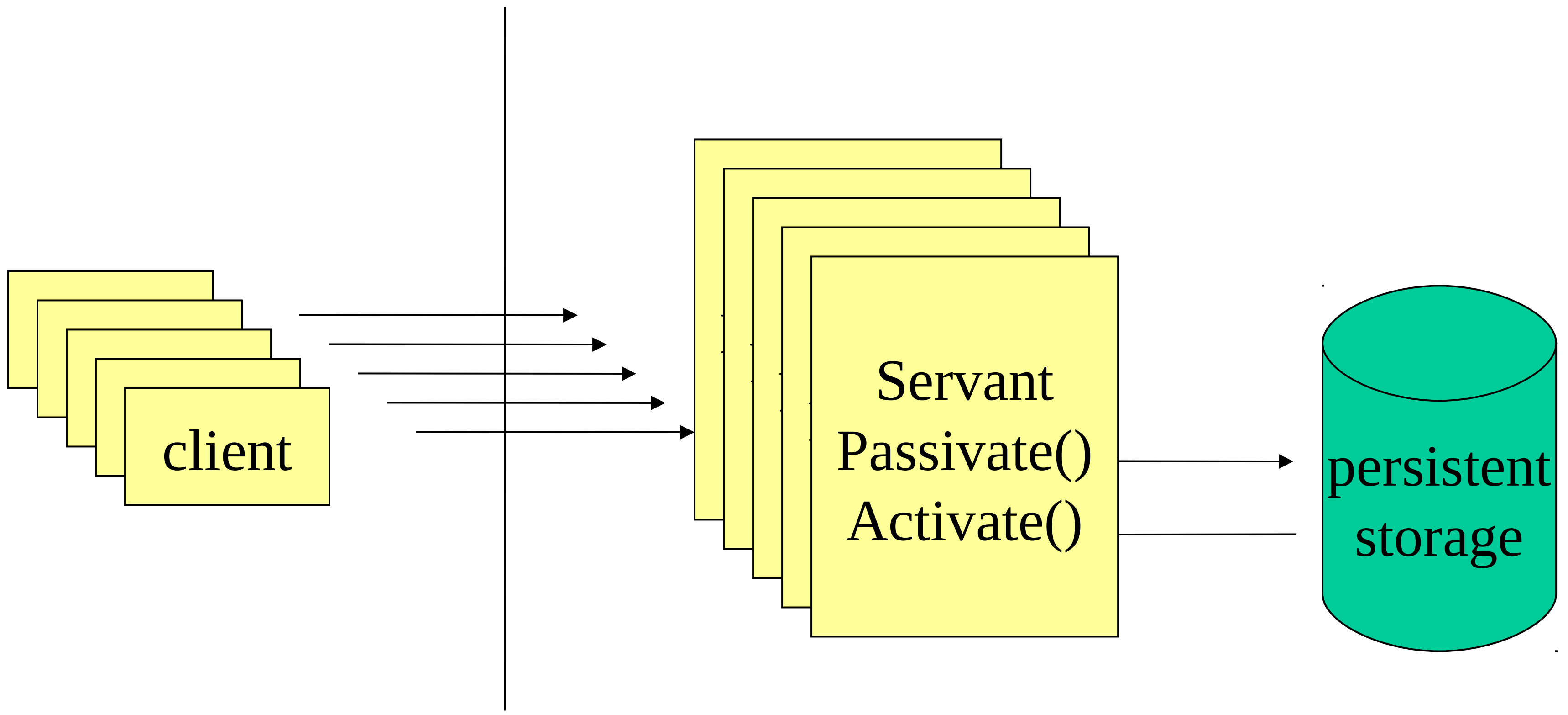
Remote operations   ***From: W.Emmerich***

Java RMI: Activation



ActGroup	ClassName	URL	Init
~~AG$_1$~~	~~Team~~	~~www.bvb.de/…~~	
AG$_2$	Player	www.bvb.de/…	
AG$_2$	Player	www.bvb.de/…	
AG$_2$	Player	www.bvb.de/…	

Why is activation so important?



If a server can transparently store servant state on persistent storage and re-create the servant on demand, then it is able to control its resources against memory exhaustion and performane degradation.

Security

- Specify the QOS of sockets used by RMI, e.g. SSL channel

- RMISecurityManager prevents or controls (policy) local access from downloaded implementations.
- A SecurityManager is now required on both client and server sides:

```
 if (System.getSecurityManager() == null) {
        System.setSecurityManager(new SecurityManager());
   }
```

- Fallback to HTTP-Post in case firewall blocks regular sockets (tunneling)

Transportable Behavior: a compute server

```
// regular interface (non-remote) for commands (pattern)
public interface Task {
    Object run();
}


// the compute server remote interface
import java.rmi.*;
public interface ComputeServer extends Remote {
    Object compute(Task task) throws RemoteException;
}


// the compute server implementation
import java.rmi.*;
import java.rmi.server.*;
public class ComputeServerImpl
    extends UnicastRemoteObject
    implements ComputeServer
{
    public ComputeServerImpl() throws RemoteException { }
    public Object compute(Task task) {
        return task.run();
    }
    public static void main(String[] args) throws Exception {
        // use the default, restrictive security manager
        System.setSecurityManager(new RMISecurityManager());
        ComputeServerImpl server = new ComputeServerImpl();
        Naming.rebind("ComputeServer", server);
        System.out.println("Ready to receive tasks");
        return;
    }
}
```

After: a compute server architecture, see resources

Resources

- Middleware für verteilte Systeme (MIKO ORB) mit Source Code. (dpunkt verlag hat noch ein Buch direkt zum MIKO angekündigt.
- Orfali/Harkey, Client/Server Programming with Java and CORBA (covers RMI as well, very good reading with lot's of code by the fathers of C/S computing)
- Paul Wilson, Garbage Collection
- Gregor Kiczales, Andreas Paepcke: Open Implementations and Metaobject Protocols. A free and very easy tutorial available on the web about reflection etc.)
- CORBA Request Portable Interceptors: A Performance Analysis C. Marchetti, L. Verde and R. Baldoni
- Van Steen, Tanenbaum: Chapter 9 (slides plus pdf from: http://www.prenhall.com/divisions/esm/app/author_tanenbaum/custom/dist_sys_1e/
- Oracle Tutorial on RMI: http://docs.oracle.com/javase/tutorial/rmi/overview.html

Resources

- Stefan Reich, Escape from Multi-threaded Hell http://www.drjava.de/e-presentation/html-english/img0.html (good intro to event-loops and “E”)
- A compute server architecture, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html

Resources

Doug Lead, Design for open Systems in Java http://gee.cs.oswego.edu/dl/coord/index.html

- Carl Hewitt's seminal paper ***The Challenge of Open Systems (http://citeseer.nj.nec.com/context/221753/0)*** explains the constraints of large scale mutually suspicious radically distributed systems.