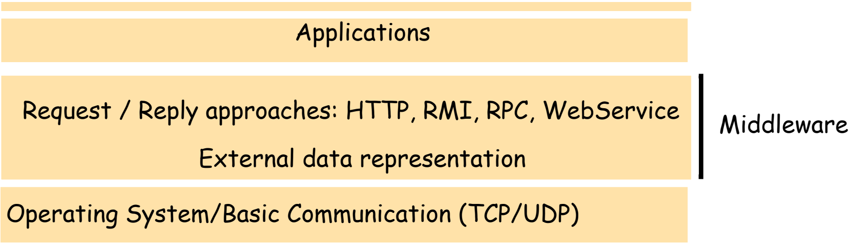
Request Reply
Remote Method Invocation
Remote Service Invocation

Middleware layers



Applications

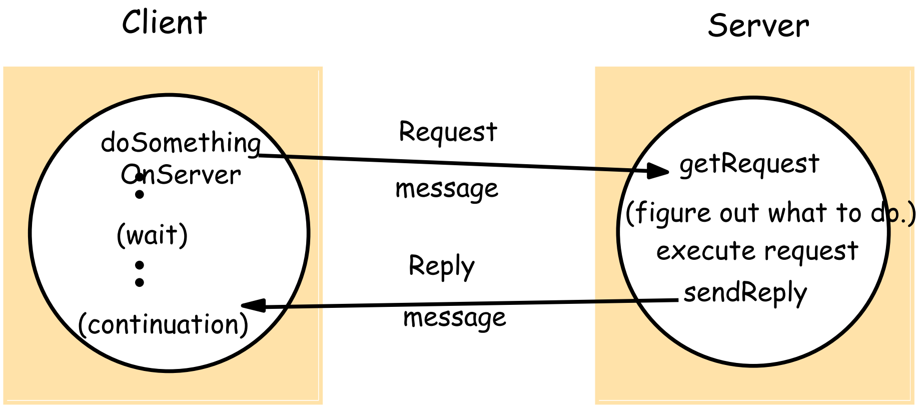
Request / Reply approaches: HTTP, RMI, RPC, WebService

External data representation

Middleware

Operating System/Basic Communication (TCP/UDP)

Client-Server communication



Client

Server

doSomething
OnServer

(wait)

(continuation)

Request

message

Reply

message

getRequest

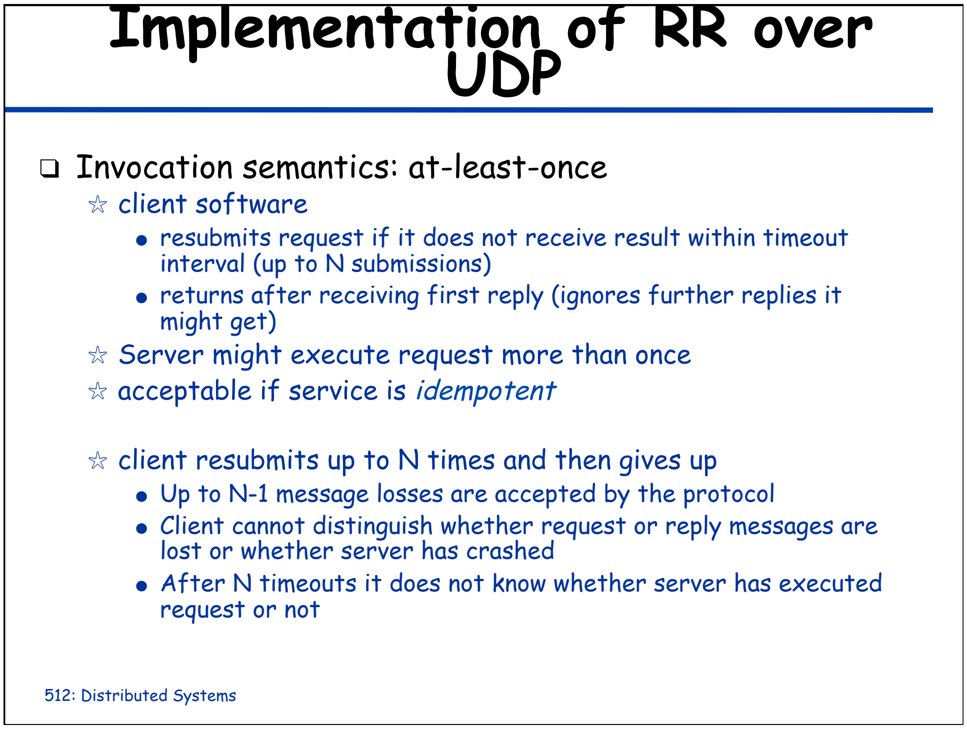
(figure out what to do.)

execute request

sendReply

Implementation of RR over UDP

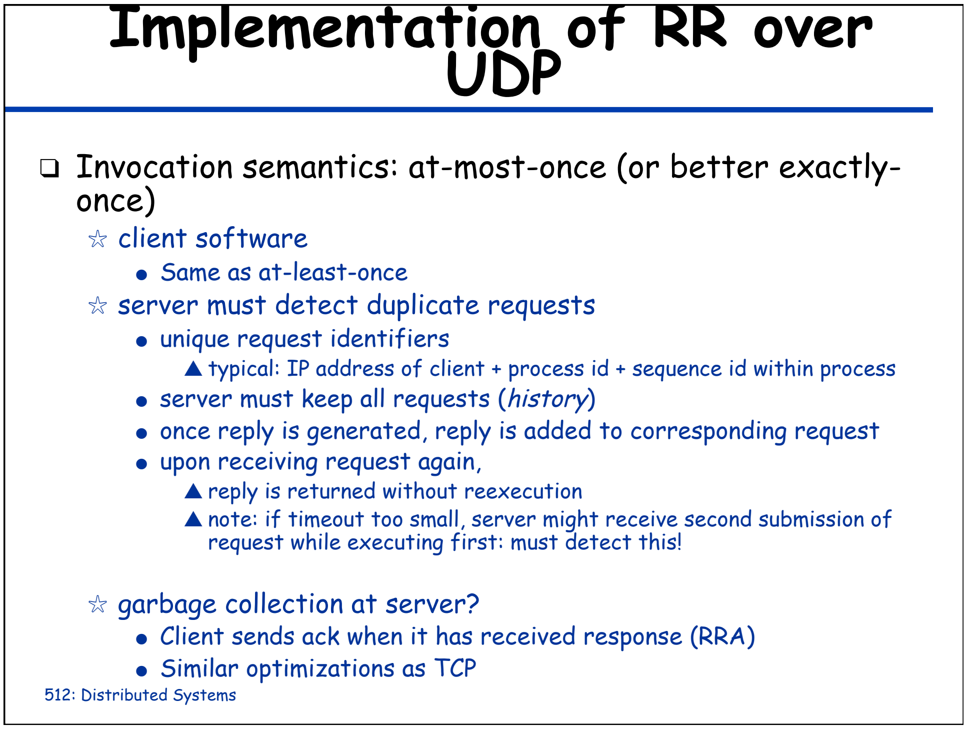
- ❑ Failure model:
  - ☆ message loss, process crash
- ❑ Invocation semantics:
  - ☆ *maybe*
    - ● provides *best-effort*
    - ● avoids blocking by *timeout*
      - ▲give client failure exception after timeout



Implementation of RR over UDP

- Invocation semantics: at-least-once
  - client software
    - resubmits request if it does not receive result within timeout interval (up to N submissions)
    - returns after receiving first reply (ignores further replies it might get)
  - Server might execute request more than once
  - acceptable if service is *idempotent*

  - client resubmits up to N times and then gives up
    - Up to N-1 message losses are accepted by the protocol
    - Client cannot distinguish whether request or reply messages are lost or whether server has crashed
    - After N timeouts it does not know whether server has executed request or not



Implementation of RR over UDP

- Invocation semantics: at-most-once (or better exactly-once)
  - client software
    - Same as at-least-once
  - server must detect duplicate requests
    - unique request identifiers
      - typical: IP address of client + process id + sequence id within process
    - server must keep all requests (*history*)
    - once reply is generated, reply is added to corresponding request
    - upon receiving request again,
      - reply is returned without reexecution
      - note: if timeout too small, server might receive second submission of request while executing first: must detect this!

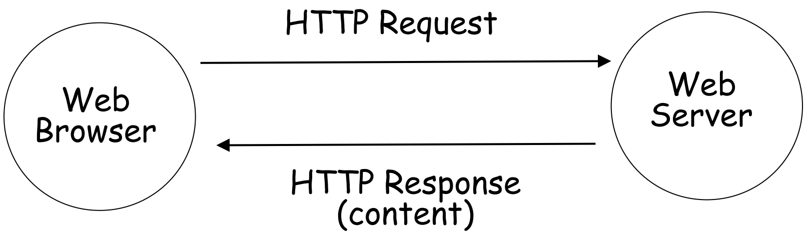
  - garbage collection at server?
    - Client sends ack when it has received response (RRA)
    - Similar optimizations as TCP

Comparison

- ❑ Compare at-most-once RR over UDP with TCP
  - ☆ Similar mechanisms but slightly different implementation
    - ● Exactly-once service execution vs. exactly-once message delivery
- ❑ Compare at-most-once RR over UDP with RR using simply TCP
  - ☆ Both provide the same semantics (exactly-once as long as N tries are sufficient)
  - ☆ Performance different
    - ● Different number of messages sent

HTTP: Hypertext Transfer Protocol

- ❑ On top of TCP

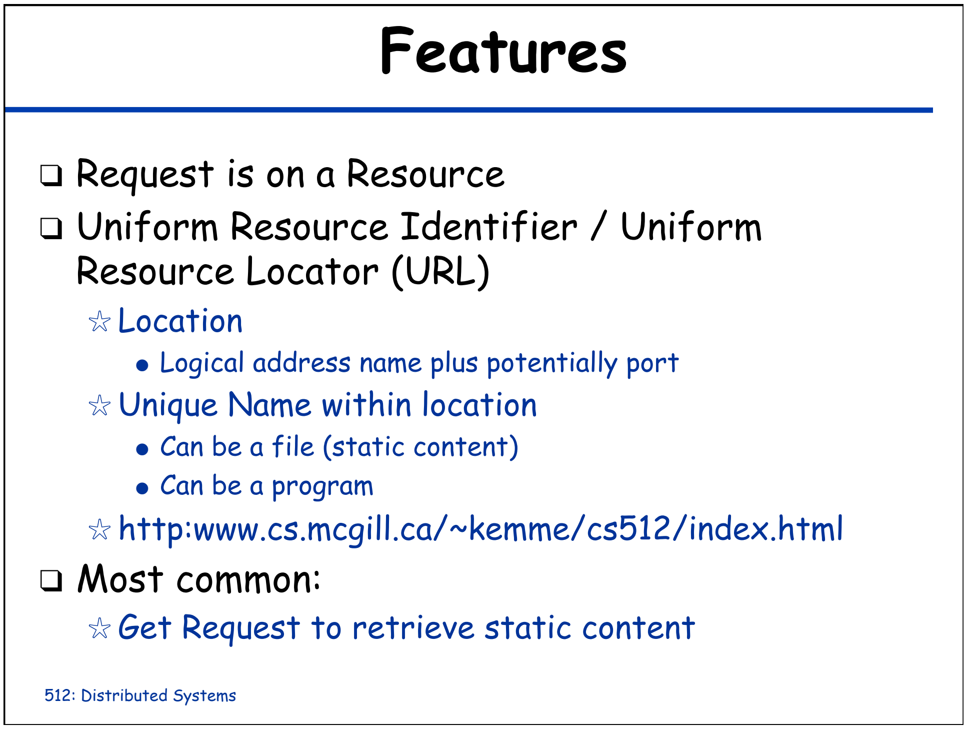


HTTP Request

Web Browser

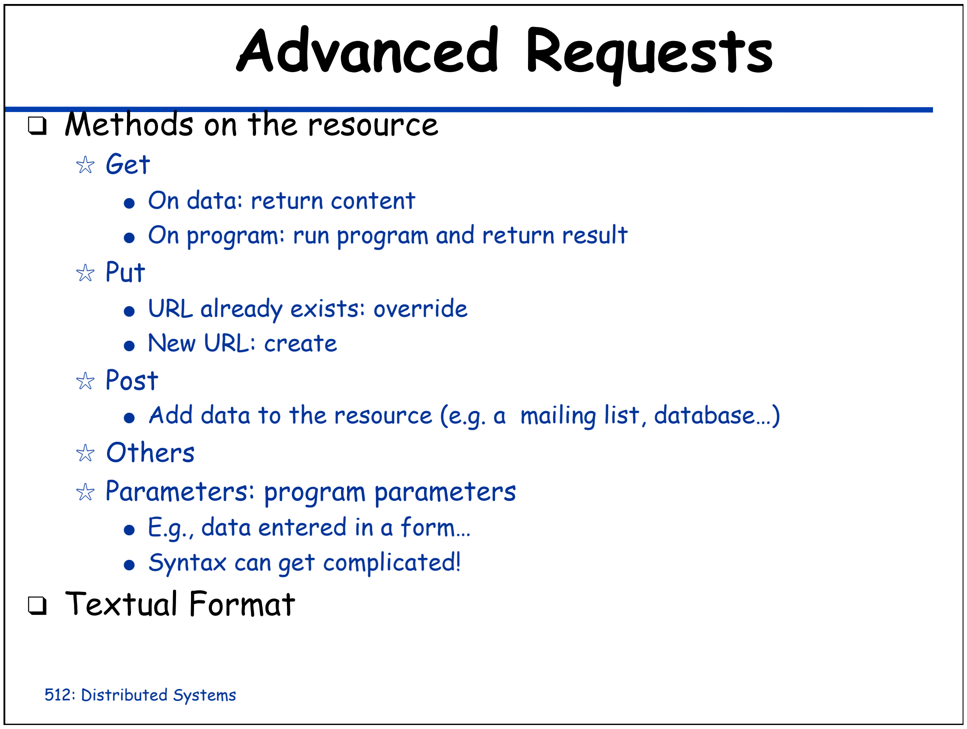
Web Server

HTTP Response (content)



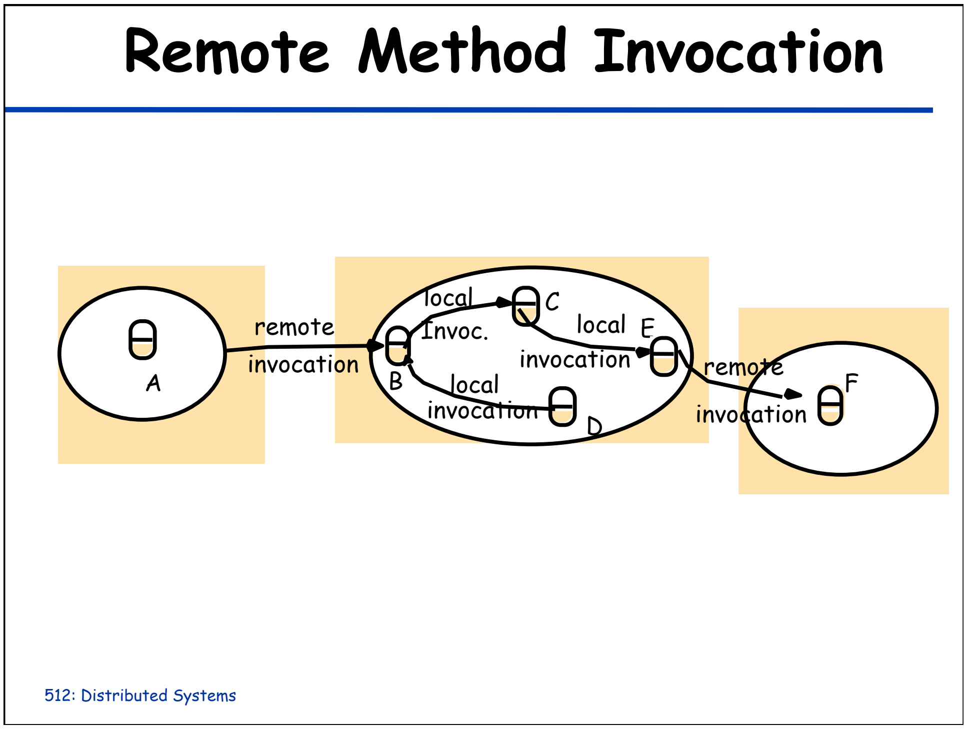
Features

- ❑ Request is on a Resource
- ❑ Uniform Resource Identifier / Uniform Resource Locator (URL)
  - ☆ Location
    - Logical address name plus potentially port
  - ☆ Unique Name within location
    - Can be a file (static content)
    - Can be a program
  - ☆ http:www.cs.mcgill.ca/~kemme/cs512/index.html
- ❑ Most common:
  - ☆ Get Request to retrieve static content



Advanced Requests

- ❑ Methods on the resource
  - ☆ Get
    - On data: return content
    - On program: run program and return result
  - ☆ Put
    - URL already exists: override
    - New URL: create
  - ☆ Post
    - Add data to the resource (e.g. a mailing list, database…)
  - ☆ Others
  - ☆ Parameters: program parameters
    - E.g., data entered in a form…
    - Syntax can get complicated!
- ❑ Textual Format



Remote Method Invocation

A

remote invocation

B

local Invoc.

C

local invocation

E

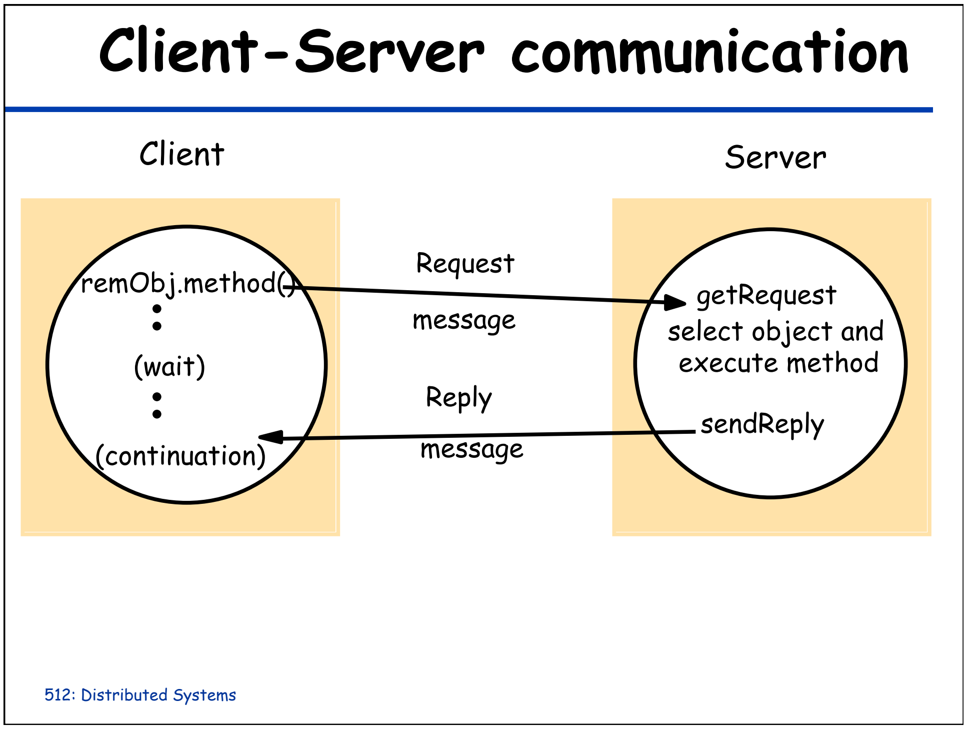
local invocation

D

remote invocation

F

512: Distributed Systems



Client-Server communication

Client

Server

remObj.method()

(wait)

(continuation)

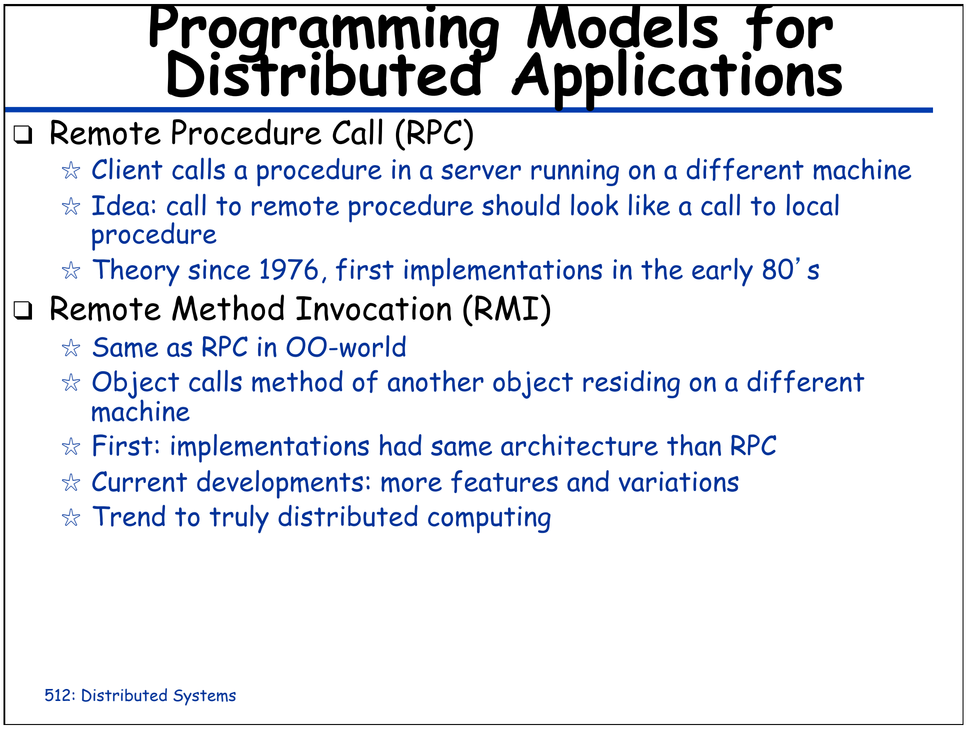
Request message

Reply message

getRequest
select object and
execute method

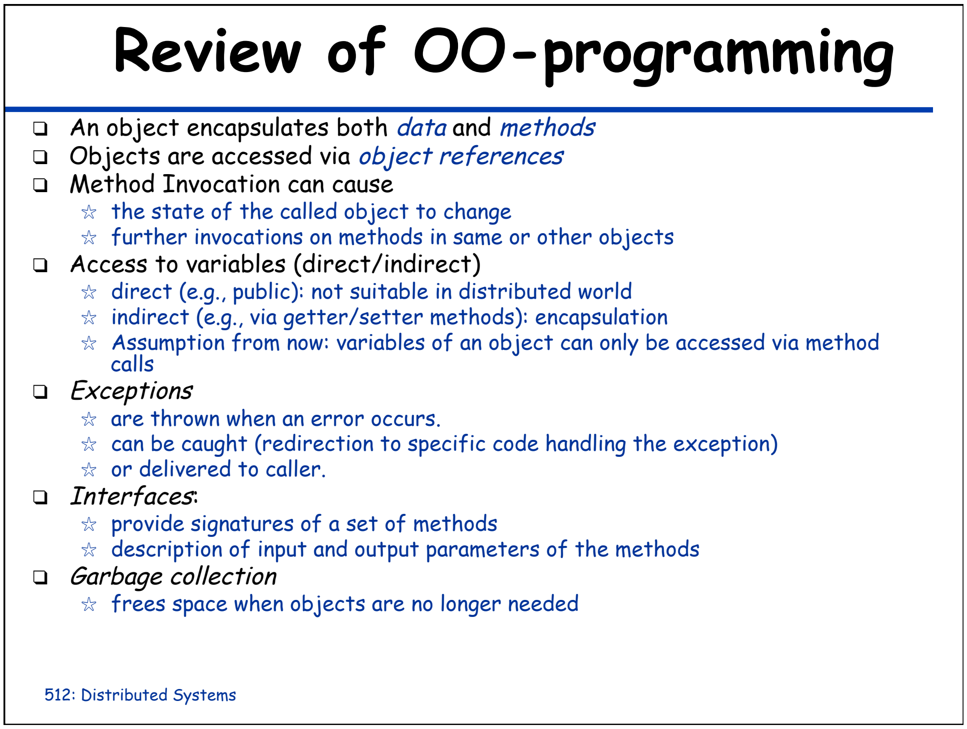
sendReply

512: Distributed Systems



Programming Models for Distributed Applications

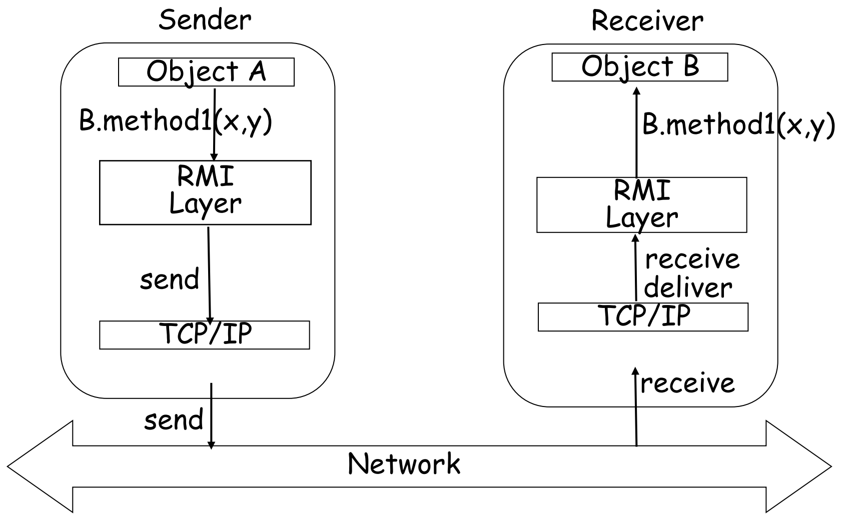
- ❑ Remote Procedure Call (RPC)
  - ☆ Client calls a procedure in a server running on a different machine
  - ☆ Idea: call to remote procedure should look like a call to local procedure
  - ☆ Theory since 1976, first implementations in the early 80's
- ❑ Remote Method Invocation (RMI)
  - ☆ Same as RPC in OO-world
  - ☆ Object calls method of another object residing on a different machine
  - ☆ First: implementations had same architecture than RPC
  - ☆ Current developments: more features and variations
  - ☆ Trend to truly distributed computing



Review of OO-programming

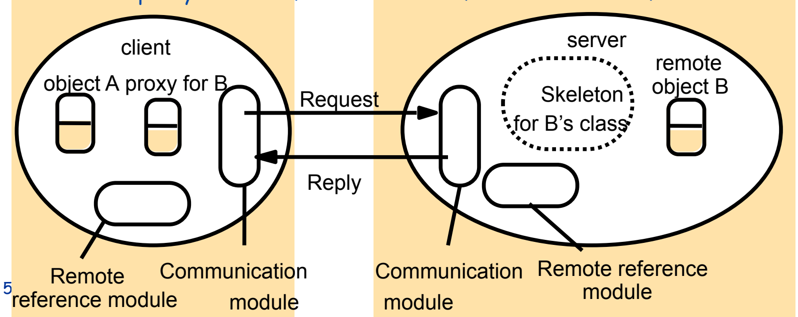
- ❑ An object encapsulates both *data* and *methods*
- ❑ Objects are accessed via *object references*
- ❑ Method Invocation can cause
  - ☆ the state of the called object to change
  - ☆ further invocations on methods in same or other objects
- ❑ Access to variables (direct/indirect)
  - ☆ direct (e.g., public): not suitable in distributed world
  - ☆ indirect (e.g., via getter/setter methods): encapsulation
  - ☆ Assumption from now: variables of an object can only be accessed via method calls
- ❑ *Exceptions*
  - ☆ are thrown when an error occurs.
  - ☆ can be caught (redirection to specific code handling the exception)
  - ☆ or delivered to caller.
- ❑ *Interfaces*:
  - ☆ provide signatures of a set of methods
  - ☆ description of input and output parameters of the methods
- ❑ *Garbage collection*
  - ☆ frees space when objects are no longer needed

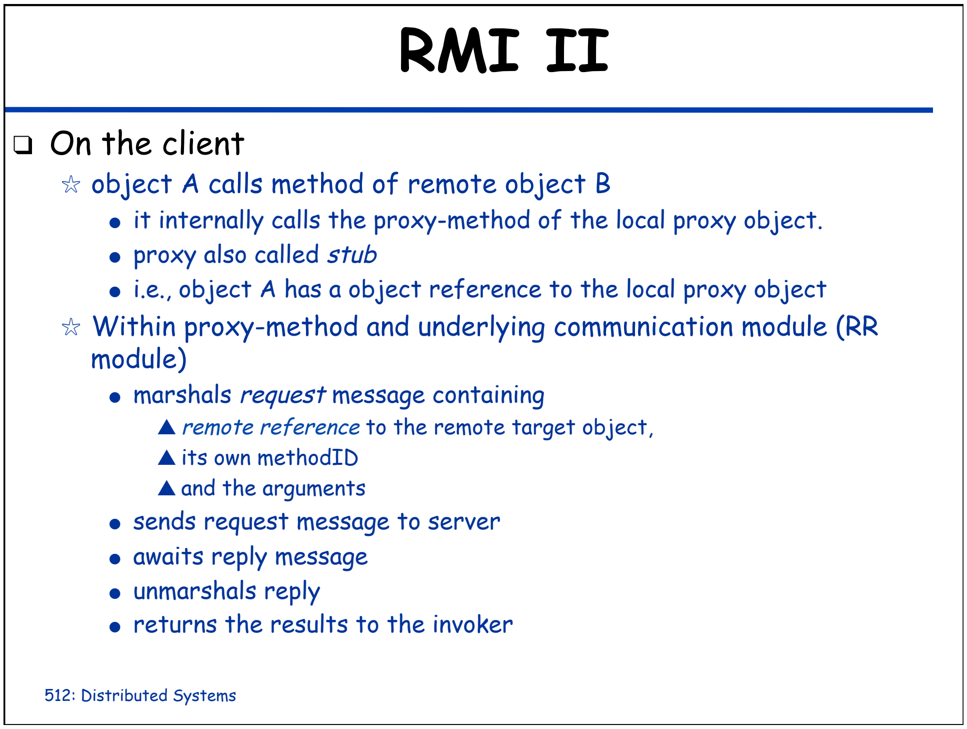
RMI I



RMI I

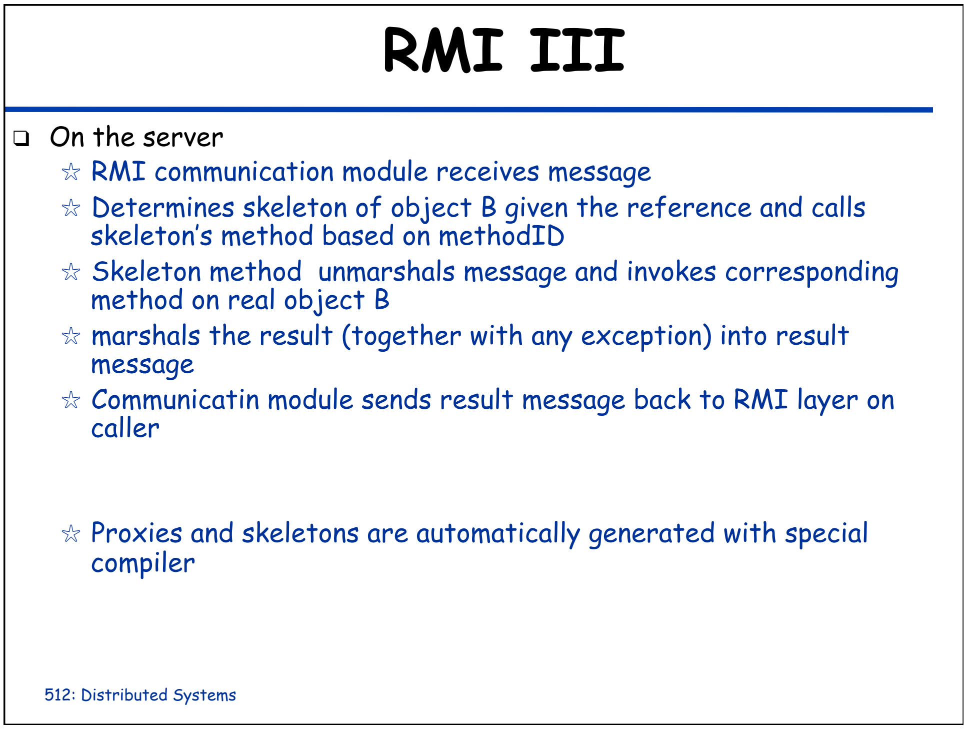
- *Remote Interface File* of an object class
  - ☆ Interface file for the set of methods that can be called remotely
  - ☆ object class can have further methods that can only be called locally
- Method description in remote interface file
  - ☆ input and output parameters are specified
  - ☆ a parameter can be an object
- Implementation of local and remote methods
  - ☆ only at server
- Proxy object at client (stub)
  - ☆ contains proxy methods for all methods of the remote interface





RMI II

- On the client
  - ☆ object A calls method of remote object B
    - it internally calls the proxy-method of the local proxy object.
    - proxy also called *stub*
    - i.e., object A has a object reference to the local proxy object
  - ☆ Within proxy-method and underlying communication module (RR module)
    - marshals *request* message containing
      - ▲ *remote reference* to the remote target object,
      - ▲ its own methodID
      - ▲ and the arguments
    - sends request message to server
    - awaits reply message
    - unmarshals reply
    - returns the results to the invoker



RMI III

- On the server
  - ☆ RMI communication module receives message
  - ☆ Determines skeleton of object B given the reference and calls skeleton's method based on methodID
  - ☆ Skeleton method unmarshals message and invokes corresponding method on real object B
  - ☆ marshals the result (together with any exception) into result message
  - ☆ Communicatin module sends result message back to RMI layer on caller

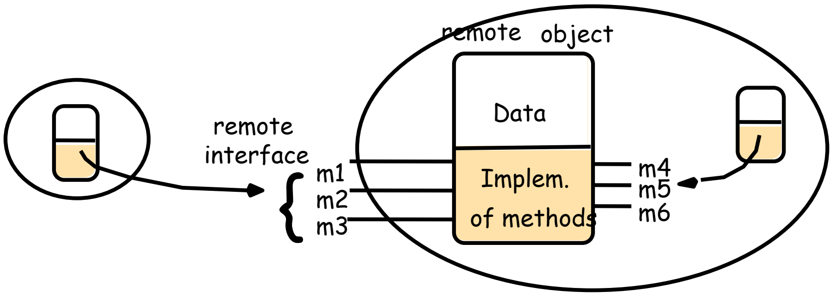
  - ☆ Proxies and skeletons are automatically generated with special compiler

Interface File in Java RMI

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface NumberStack extends Remote {
    public int getNumber() throws RemoteException;
    public void putNumber(int num) throws RemoteException; }
```

- ❑ By extending java.rmi.Remote, this interface's methods can be called from any virtual machine. Any object that implements this interface becomes a remote object.
- ❑ As a member of a remote interface, the getNumber method is a remote method. Therefore the method must be defined as being capable of throwing a java.rmi.RemoteException.
- ❑ This exception is thrown by the RMI system during a remote method when communication failure or a protocol error has occurred.
- ❑ Any code making a call to a remote method needs to handle this exception by either catching it or declaring it in its throws clause.

A remote object and its remote interface



Interface Definition Languages

- ❑ Java has the concept of interfaces and hence, the remote interface can be described as usual + extending interface to be REMOTE
- ❑ In general interfaces are described with Interface Definition Languages
- ❑ Examples RMI
  - ☆ Corba IDL
  - ☆ DCOM IDL for Microsoft's Distributed Common Object Model (DCOM) RMI
- ❑ Examples RPC
  - ☆ Sun XDR
  - ☆ DCE IDL: RPC system of OSF's Distributed Computing Environment DCE

Remote Object Reference

32 bits	*32 bits*	*32 bits*	*32 bits*
Internet address	port number	time	objectID

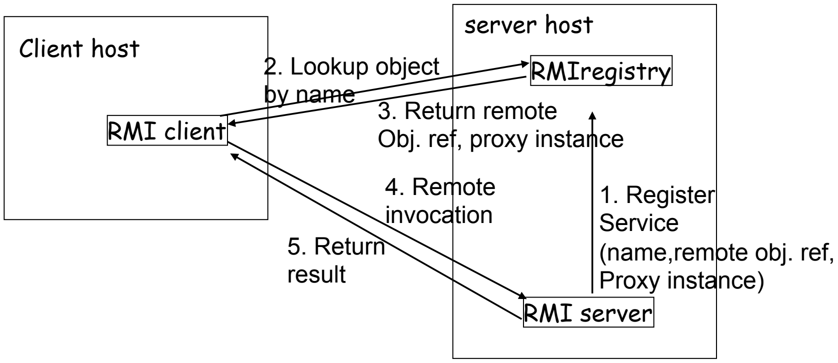
- ❑ In non-distributed execution, objects can be accessed via object reference.
- ❑ In Java, a variable that appears to hold an object actually holds a reference to that object.
- ❑ An object A can invoke the method of a remote object B, if it has access to B's remote object reference.
- ❑ Generally: A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object.
- ❑ Note: In some RMI online information the remote object reference actually refers to a stub/proxy instance

- ❑ Conceptually like a URL

Remote reference module

- Maintains remote object table
  - entry
    - remote object reference and local proxy reference (on client)
    - remote object reference and real object reference (on server)
- Some Tasks
  - On a server: when a remote object reference arrives in a request message it looks for the corresponding local object reference which points to the real object
  - On a client: when a remote object reference arrives in a reply message it looks for the proxy object.
  - When a remote object is to be passed as argument or result for the first time, the remote reference module creates a remote object reference and adds it to the table

A basic Java RMI call



Client host

RMI client

server host

RMIregistry

RMI server

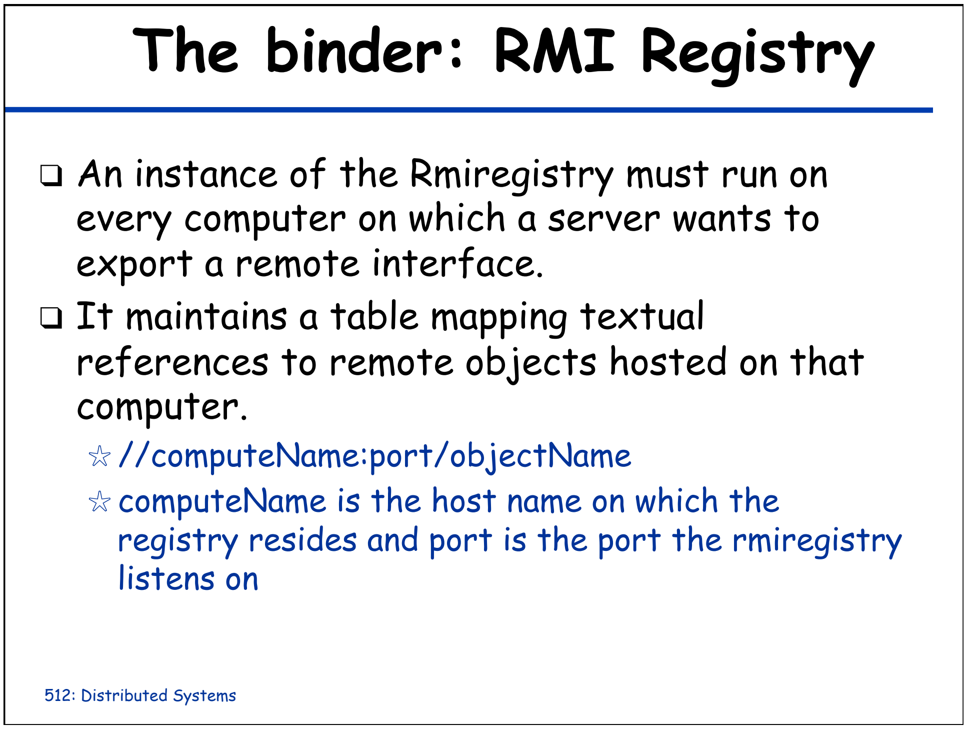
1. Register Service (name,remote obj. ref, Proxy instance)

2. Lookup object by name

3. Return remote Obj. ref, proxy instance

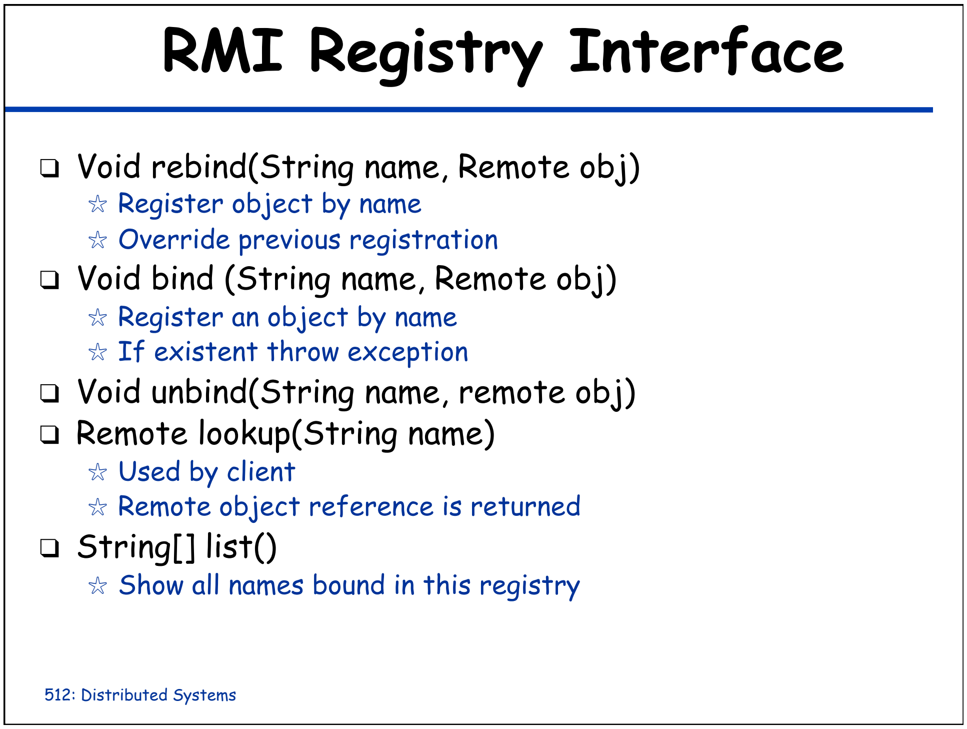
4. Remote invocation

5. Return result



The binder: RMI Registry

- ❑ An instance of the Rmiregistry must run on every computer on which a server wants to export a remote interface.
- ❑ It maintains a table mapping textual references to remote objects hosted on that computer.
  - ☆ //computeName:port/objectName
  - ☆ computeName is the host name on which the registry resides and port is the port the rmiregistry listens on



RMI Registry Interface

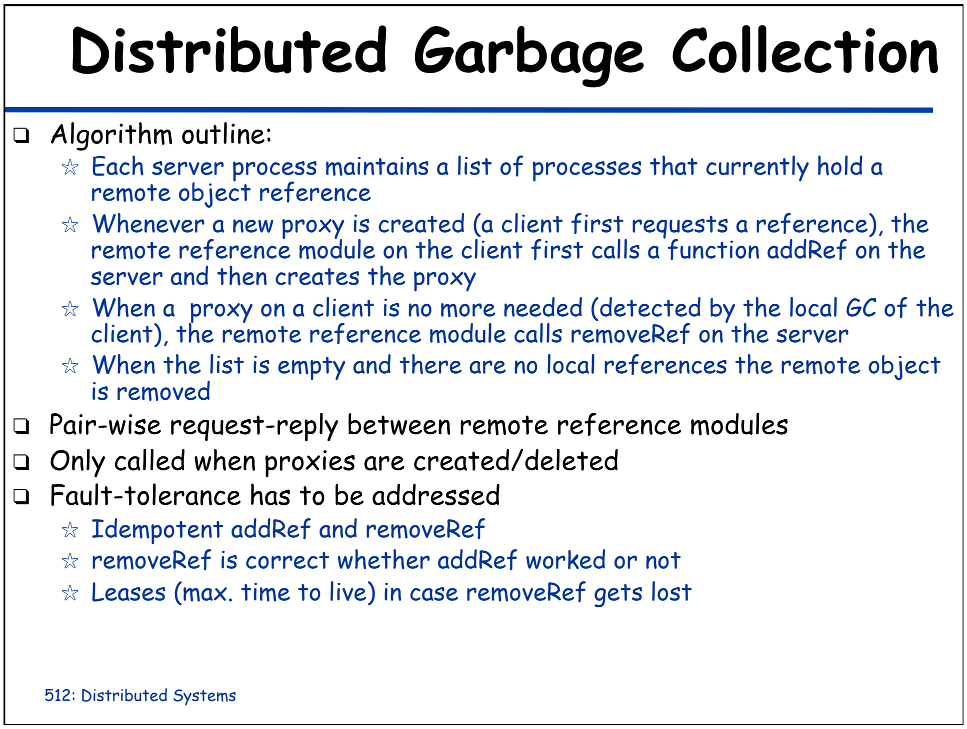
- ❑ Void rebind(String name, Remote obj)
  - ☆ Register object by name
  - ☆ Override previous registration
- ❑ Void bind (String name, Remote obj)
  - ☆ Register an object by name
  - ☆ If existent throw exception
- ❑ Void unbind(String name, remote obj)
- ❑ Remote lookup(String name)
  - ☆ Used by client
  - ☆ Remote object reference is returned
- ❑ String[] list()
  - ☆ Show all names bound in this registry

Method Parameters

- ❑ Input and output parameters can be
  - ☆ Primitive types (pass by value)
  - ☆ Remote objects (pass the remote object reference)
    - If the recipient of a remote object reference does not have the object proxy, then the object proxy is automatically downloaded
  - ☆ Local objects. They are passed by copy, using object serialization (the object must implement the java.io.Serializable Interface).
    - If the recipient of an object passed by value does not already possess the class of the object, its code is automatically downloaded.

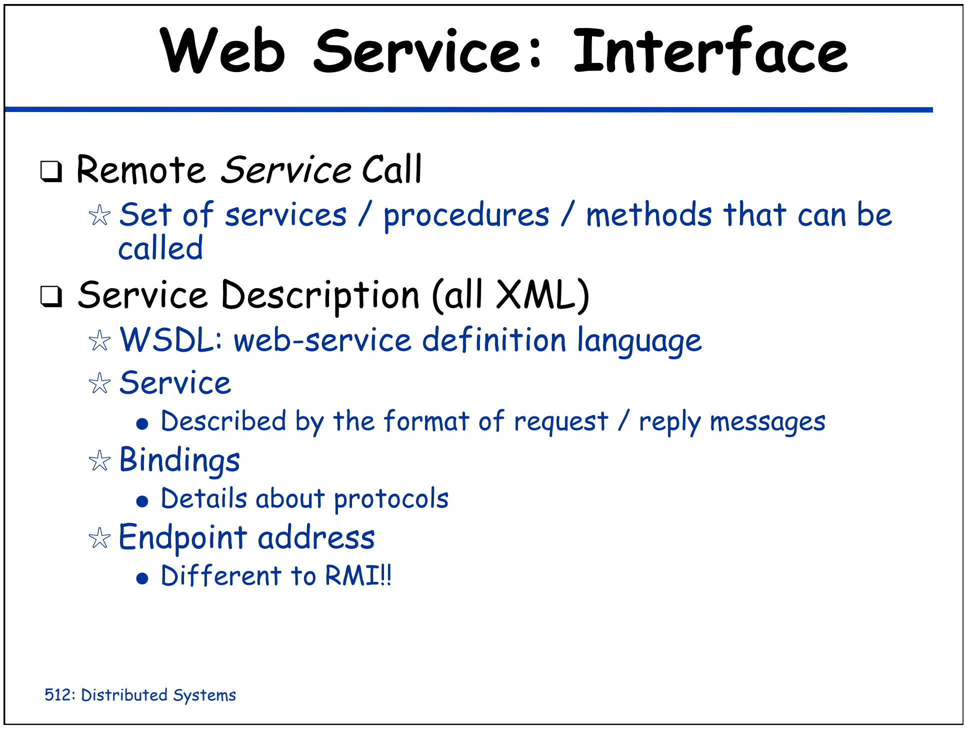
Distributed Garbage Collection

- ❑ An object must consist as long as there is a local or remote reference held
- ❑ Whenever the last references is deleted the object should be garbage collected.
- ❑ A classical distributed algorithm problem:
  - ☆ Solution: reference counting
- ❑ 2-level
  - ☆ local proxy garbage collection on each client
    - Whenever a remote object reference enters a process, a proxy is created, when the last reference to this object on the process is deleted, the proxy is deleted
  - ☆ object maintenance on the server
    - whenever there is no proxy anymore **and** there is no local reference, you can delete the object



Distributed Garbage Collection

- ❑ Algorithm outline:
  - ☆ Each server process maintains a list of processes that currently hold a remote object reference
  - ☆ Whenever a new proxy is created (a client first requests a reference), the remote reference module on the client first calls a function addRef on the server and then creates the proxy
  - ☆ When a proxy on a client is no more needed (detected by the local GC of the client), the remote reference module calls removeRef on the server
  - ☆ When the list is empty and there are no local references the remote object is removed
- ❑ Pair-wise request-reply between remote reference modules
- ❑ Only called when proxies are created/deleted
- ❑ Fault-tolerance has to be addressed
  - ☆ Idempotent addRef and removeRef
  - ☆ removeRef is correct whether addRef worked or not
  - ☆ Leases (max. time to live) in case removeRef gets lost



Web Service: Interface

- ❑ Remote *Service* Call
  - ☆ Set of services / procedures / methods that can be called
- ❑ Service Description (all XML)
  - ☆ WSDL: web-service definition language
  - ☆ Service
    - ● Described by the format of request / reply messages
  - ☆ Bindings
    - ● Details about protocols
  - ☆ Endpoint address
    - ● Different to RMI!!

Web-Service: Communication

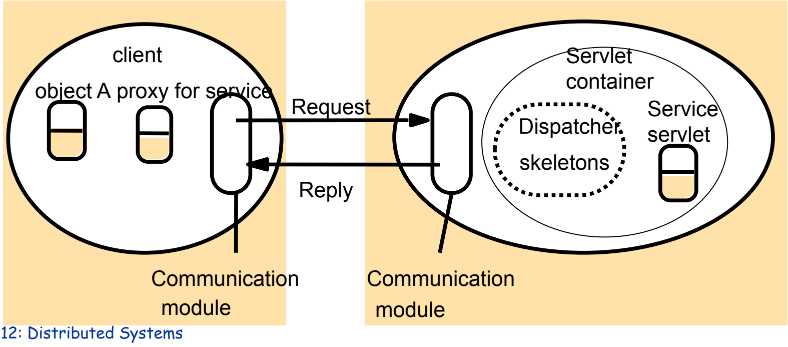
- ❑ SOAP: Simple Object Access Protocol
- ❑ Messages in XML
  - ☆ SOAP messages have a specific format
- ❑ Protocol Pattern
  - ☆ Request/reply
    - Based on HTTP
    - HTTP contains endpoint address and action for efficiency
  - ☆ Other models are offered
    - In-only
    - Out-only
    - Reliable versions

Web-Service with Java

- ❑ Java API for developing web-services: JAX-RPC
- ❑ Hides all the details of web-services
  - ☆ Define your interface using Java interface
    - Some restrictions for parameter types (objects are allowed, but no remote objects)
  - ☆ Web-service is a single object that offers the services

Web-Service: Components

- ❑ Communication Module: Http
- ❑ Container:
  - ☆ Dispatcher calls skeleton (according to http header URL)
  - ☆ Skeleton (per servlet) translates SOAP message to java servlet call; translates reply back to SOAP


client
object A
proxy for service
Request
Reply
Servlet container
Dispatcher
skeletons
Service servlet
Communication module
Communication module