Programação em Sistemas Distribuídos

MEI-MI-MSI

2018/19

3. Models of Distributed Computing

Prof. António Casimiro

Classes of distributed activities

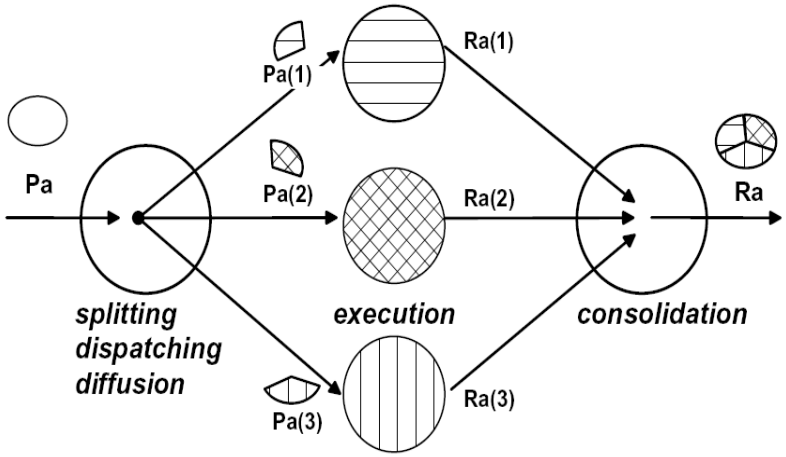
Ciências ULisboa

Coordination:	• parallelism • load balancing • functional separation
Sharing:	• concurrency control • linearizability • serializability
Replication:	• fault tolerance • performance • availability

Coordination
In distributed systems programming

Ciências ULisboa

- **Splitting and dispatching**
  - Source-based - requester splits job and dispatches/disseminates parallel task requests
  - Destination-based - whole job request is disseminated to participants, who do local splitting and dispatching, in a decentralized and deterministic manner
- **Diffusion**
  - Ordered diffusion may or may not be necessary, depending on application
  - Decentralized destination-based split/dispatch implies totally ordered reliable by default
  - View synchrony and membership maybe useful
- **Management**
  - Functional or structural splitting (e.g. sharded DBs, MapReduce)
- **Consolidation**
  - Source-based - participants consolidate the whole job result from partial task results
  - Destination-based - partial task results are sent to requester (knows how to consolidate)



Splitting

dividing job into several tasks

Dispatching

allocating tasks to participants

Diffusion

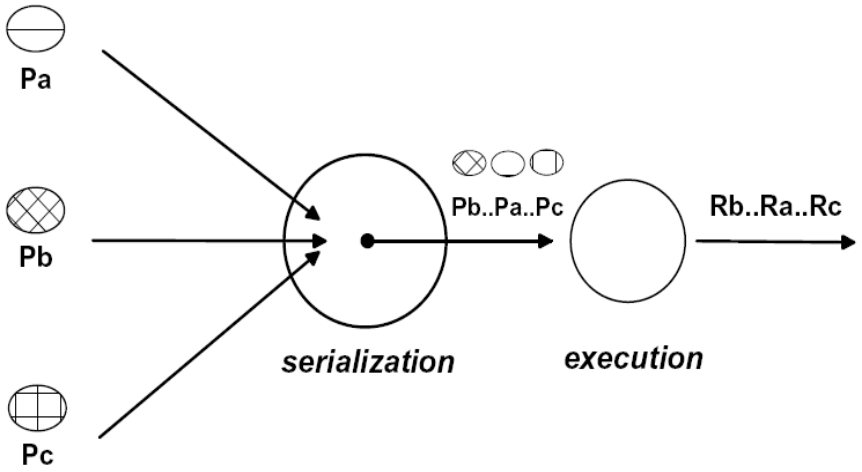
getting partial tasks to particip.

Consolidation

getting partial results into whole

Sharing
In distributed systems programming

- Diffusion
  - A protocol directs requests to the resource
- Serialization
  - Can be done at destination, or within the diffusion protocol
  - Directly-interacting clients - causal order
  - Non-directly-interacting clients - FIFO order
  - Commutative semantics - no order
  - Interleaving semantics - serializability



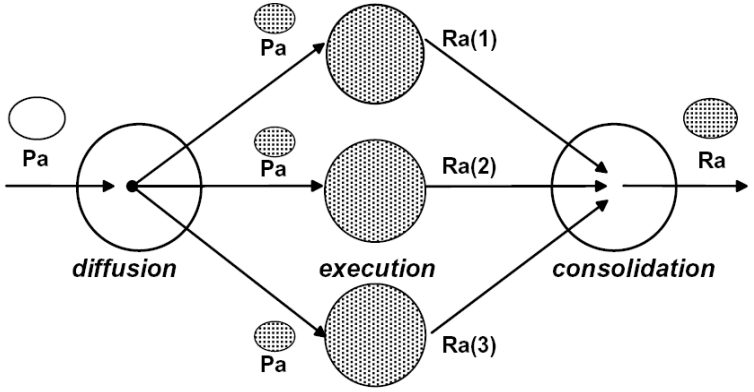
Serialization

order concurrent action requests directed at a common resource

Replication
In distributed systems programming

Ciências ULisboa

- **Diffusion**
  - Active rep. - totally ordered reliable delivery
  - Other - reliable and maybe FIFO
  - Commutative semantics - non-ordered
- **Execution**
  - Active rep.- parallel execution of the same set of actions to produce the same results
  - Semi-active rep. - leader executes first then tells order to followers
  - Passive rep.- primary executes, backups save checkpoints and log requests from last chkpt
- **Management**
  - Replica synchronization (lag bounding)
  - Backup (logging and checkpointing)
  - Role change (fail. det., leader election, takeover)
  - Replica determinism
- **Consolidation**
  - Omissive faults - one of the results
  - Affirmative faults - result of voting, consensus
  - Source-based - participants run an algorithm to consolidate a single result from replicated results
  - Destination-based - replicated results are sent to requester, faster but not transparent, implies knowledge by destination on how to consolidate



Diffusion

disseminating request to replicas)

Execution

executing requests in order to produce same results in all replicas

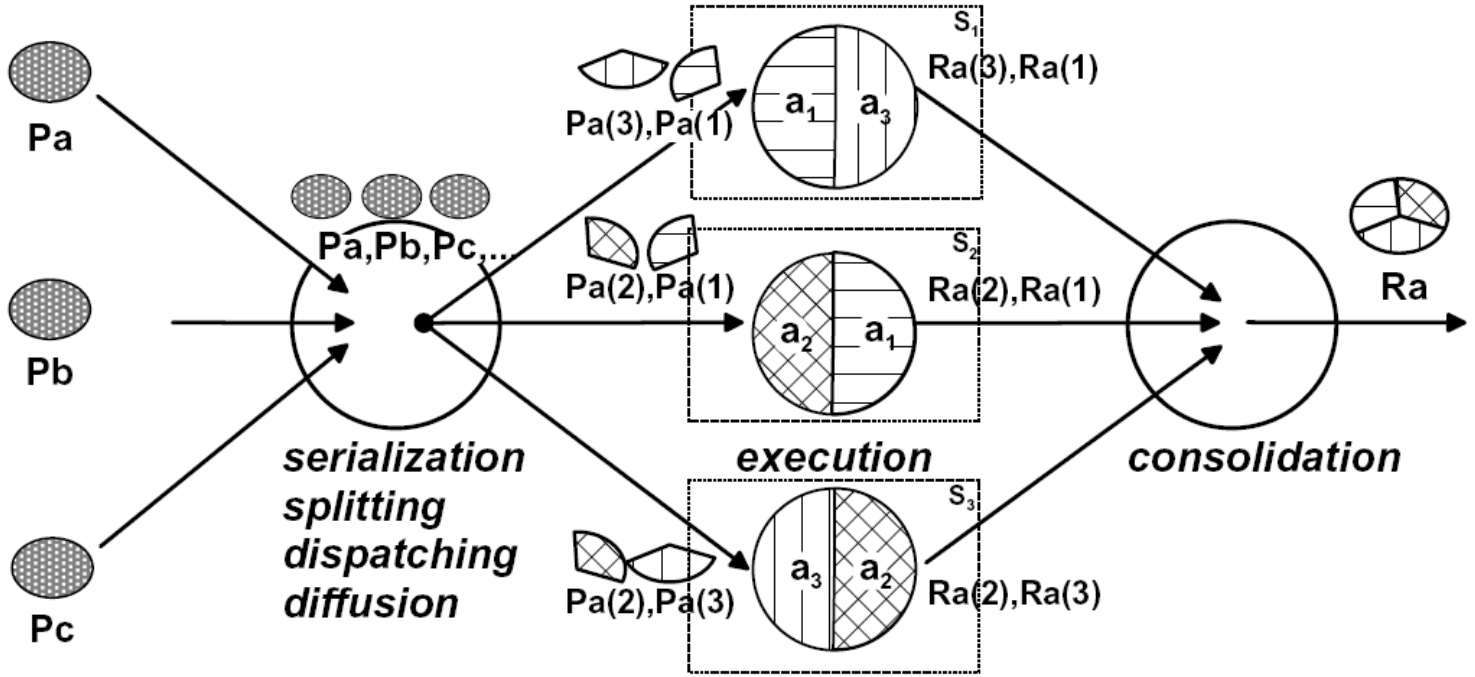
Consolidation

treat replicated results in order that correct result is delivered

Bringing it all together
Sharing, Replication, Coordination

Ciências ULisboa

- An academic example:
  - A replicated fragmented/sharded database accessed by interacting clients



Distributed Computing Models

Distributed Computing Models

Ciências ULisboa

- Actual distributed computing models aim at supporting specializations and/or combinations of the basic distributed activities
- The makings of the several classes of models lie in partial answers to these questions:
  - *Shared memory OR message passing?*
  - *Sequential OR concurrent flow?*
  - *Direct OR indirect interaction?*
- They define different styles of distributed computing

Styles of Distributed Computing

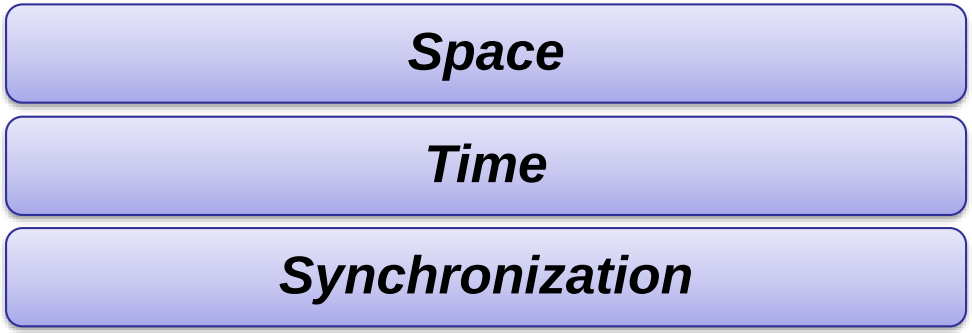
Ciências ULisboa

Shared memory	***Message passing***
Interactions amongst participants are based on objects in shared repositories (shared memory space, tuples, etc.)	Interactions amongst participants are based on remote invocations involving message sending and receiving (RPC, RMI, group comm's, etc.)
Sequential flow	***Concurrent flow***
Control flow is transferred in a blocking way amongst participant nodes (e.g. remote Client/Server executions)	Control flow is shared in a parallel, non-blocking way amongst participant nodes (e.g. DSM, group comm's, pub/sub)
Direct interaction	***Indirect interaction***
Direct and explicit coupling between participants (e.g. RPC, RMI)	Mediated or implicit coupling between participants (e.g. tuple spaces, brokers, groups, message queues, pub/sub)

Axes of coupling of distributed activities

Ciências ULisboa

- Another key factor of choice of the right distributed computing model has to do with the **degree of coupling** desired (or allowed) for the distributed activities in view, along several axes:



- The greater or lesser degree of coupling or uncoupling of activities along these axes determines the specific utility of a model

Axes of coupling of distributed activities

Ciências ULisboa

Space coupled	*Space uncoupled*
• The interacting participants need to know each other (name, address, location, number) as a configuration property • Adequate for static tightly managed systems	• The interacting participants do not need to know each other (name, location, number) • Facilitates change, dynamics of participants (replacement, update, replication, migration)
Time coupled	***Time uncoupled***
• The interacting participants need to be actively participating in the interaction at the same time • Adequate for performance oriented systems (e.g. R/T, multimedia, HPC)	• The interacting participants do not need to be actively participating in the interaction at the same time • Adequate for volatile environments where participants may come and go either expected or unexpectedly
Synchronized	***Non-Synchronized***
• The interacting participants block waiting for the conclusion of the action triggered (e.g. sending, receiving, operation request)	• The interacting participants do not need to block waiting for the conclusion of the action triggered

Distributed Computing Models
Main models, neither exhaustive nor air-tight

Ciências ULisboa

- **Prefigure combinations of activities, styles and coupling**:
  - Remote Operations (Client-Server RPC, RMI, WWW)
  - Distributed Objects
  - Distributed Shared Memory (DSM, Tuple Spaces)
  - Distributed Atomic Transactions
  - Message-oriented (Message Queue, Publish/Subscribe)
  - Stream
  - Group-Oriented
  - Peer-to-peer

Distributed Computing Models
vs. degree of coupling

Ciências ULisboa

MODEL	SPACE	TIME	SYNC
Remote Operations	C	C	S/N
Distributed Objects	C	C	S/N
Distributed Shared Memory	U	C/U	S
Tuple Spaces	U	U	S
Distributed Atomic Transactions	C	C	S
Message Queue	U	U	S/N
Publish/Subscribe	U	C/U	N
Stream	U	C	N
Group-Oriented	U	C	N
Peer-to-peer	U	U	N

How to use the right distributed computing model?

How to use the right paradigms in each model?

Next we review these models in detail

Distributed Computing Models

Main models, neither exhaustive nor air-tight

Ciências ULisboa

- **Client-Server (RPC, RMI, WWW) (Cliente-Servidor)**
- Distributed Objects (Objectos Distribuídos)
- Distributed Shared Memory (DSM, Tuples) (Memória Partilhada Distribuída)
- Distributed Atomic Transactions (Transacções Atómicas Distribuídas)
- Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)
- Stream (Corrente)
- Group-Oriented (Orientado para Grupos)
- Peer-to-peer (Inter-pares)

Remote-operations or Client-Server Model

Remote Procedure Call (RPC)

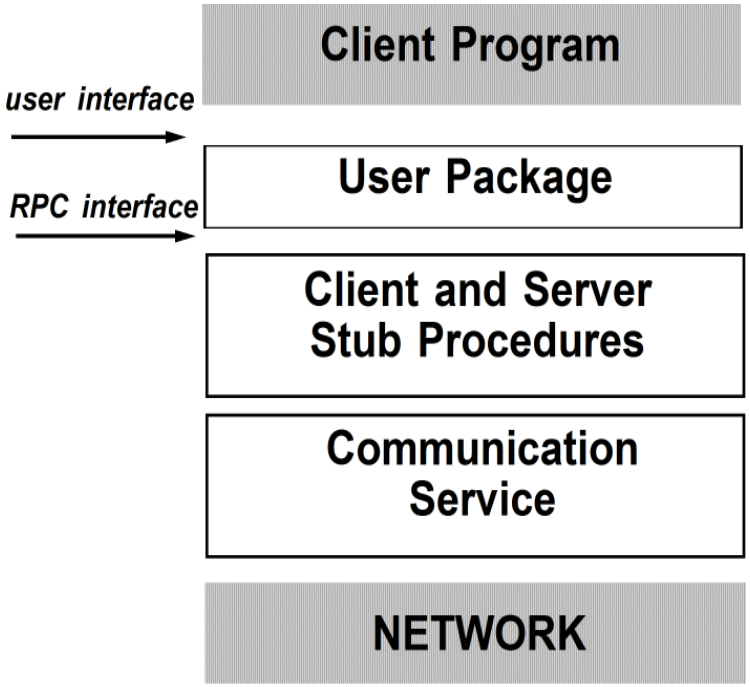
Ciências ULisboa

- Preserves the semantics of a local procedure call
- Supports the client/server model with remote operations
- Has some problems:
  - Different address spaces make it difficult to handle parameter passing
  - The communication, client or server may fail
  - Is asymmetric (client always initiates interaction)
  - Is unilateral (not suitable for groups)
  - Blocks the client for a significant time (call semantics)

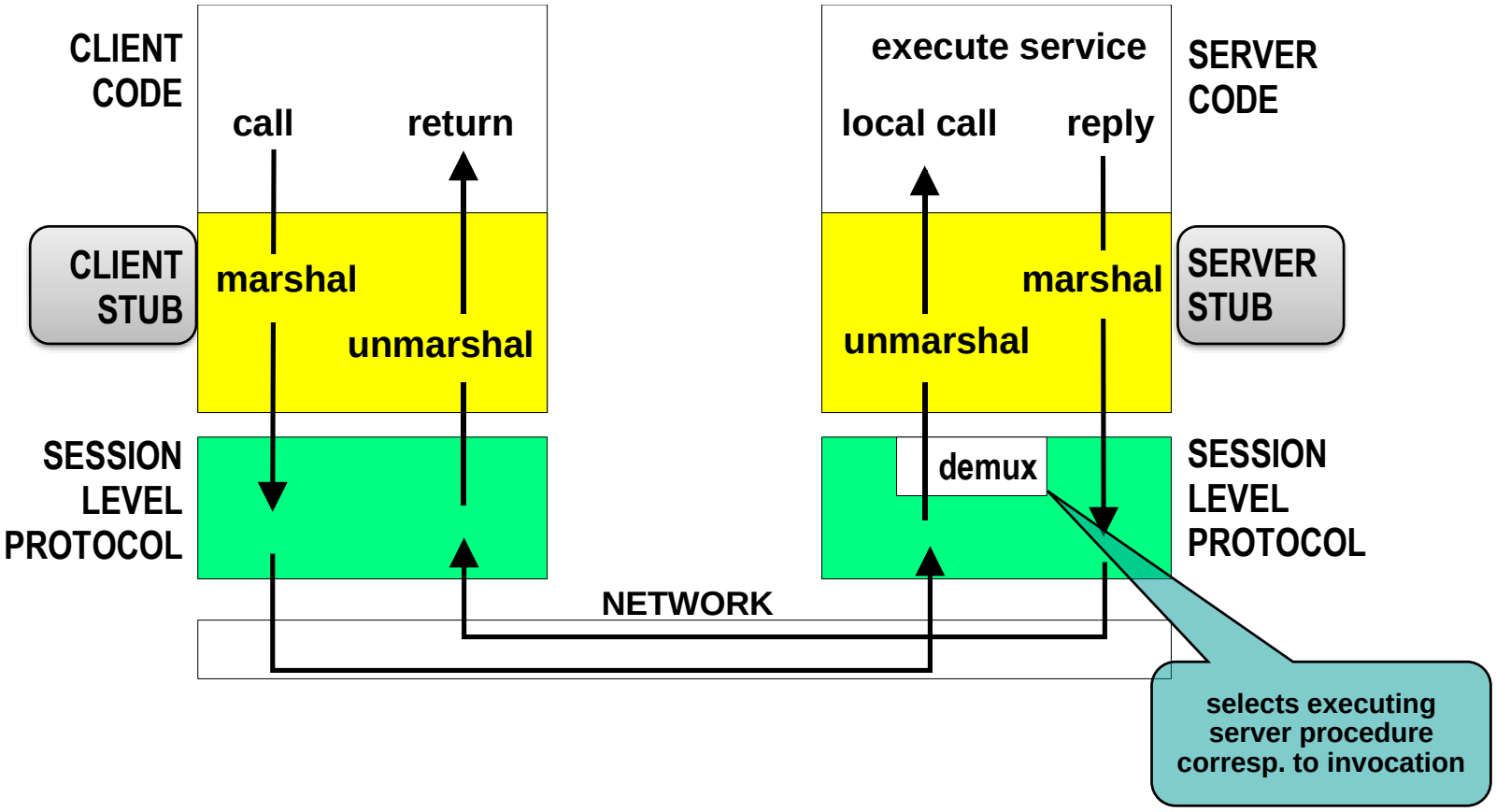
Remote Procedure Call
Architecture

Ciências ULisboa

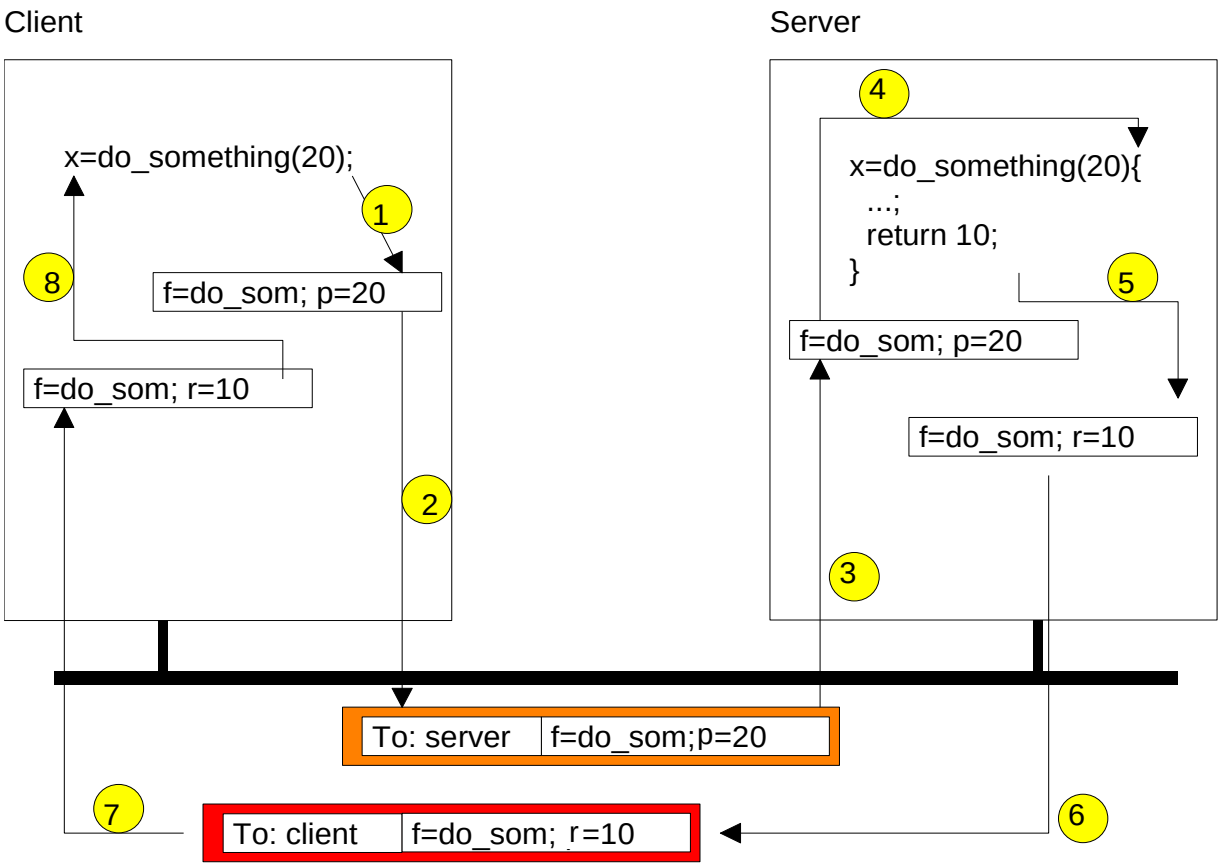
- **User Package**
  - Set of libraries to facilitate RPC programming, making it look like local invocations
- **Client and Server Stub Procedures**
  - To marshal/unmarshal function arguments into/from a message
- **Communication Service**
  - Establishes communication between client and server (UDP UNIX- sockets, sometimes TCP or other specialized remote operations protocols)



RPC in action



RPC in action
An example



Remote Method Invocation

Ciências ULisboa

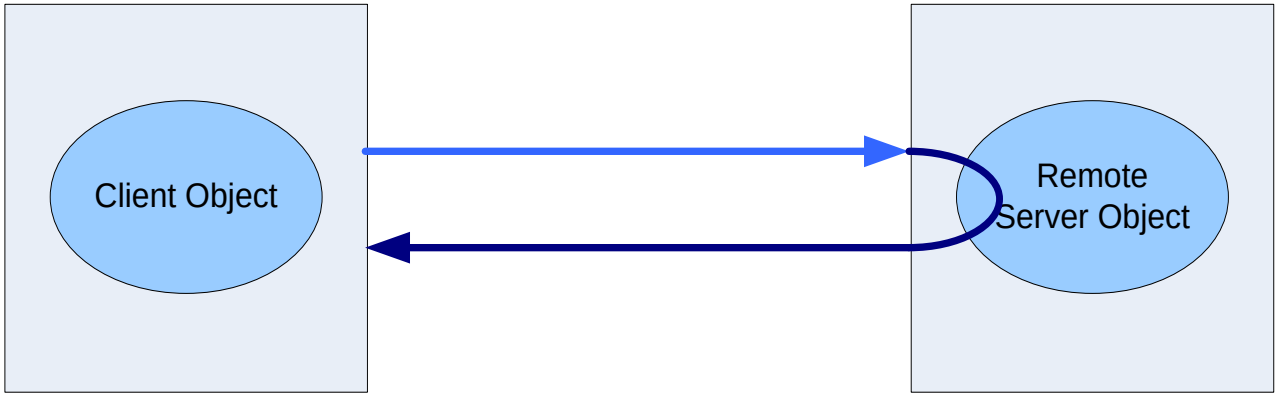
- **Remote method invocation (RMI)** similar to RPC but extended into the world of distributed **objects** under the Client/Server paradigm
- RMI:
  - Calling object can invoke a method in a potentially remote object
  - Power of O_O: objects, classes and inheritance, tools
  - Unique object references which can be passed as parameters
  - Creation of applications with distributed objects
- RMI vs. RPC:
  - Both support programming with interfaces and IDL
  - Both typically constructed on top of request-reply protocols
  - Both offer similar failure semantics

Remote Method Invocation

Remote vs. local transparency

Ciências ULisboa

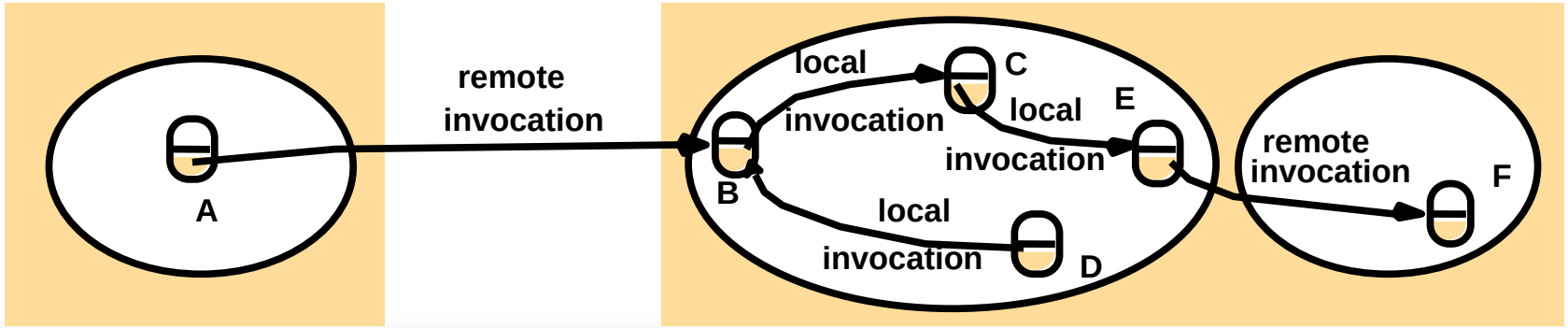
- **Client accesses remote object as if were local**
- Client request to invoke a method of an object is sent in a message to the remote server managing the object
- Invocation is carried out by executing a method of the object at the server
- The result is returned to the client in another message



Remote Method Invocation
Remote vs. local transparency

Ciências ULisboa

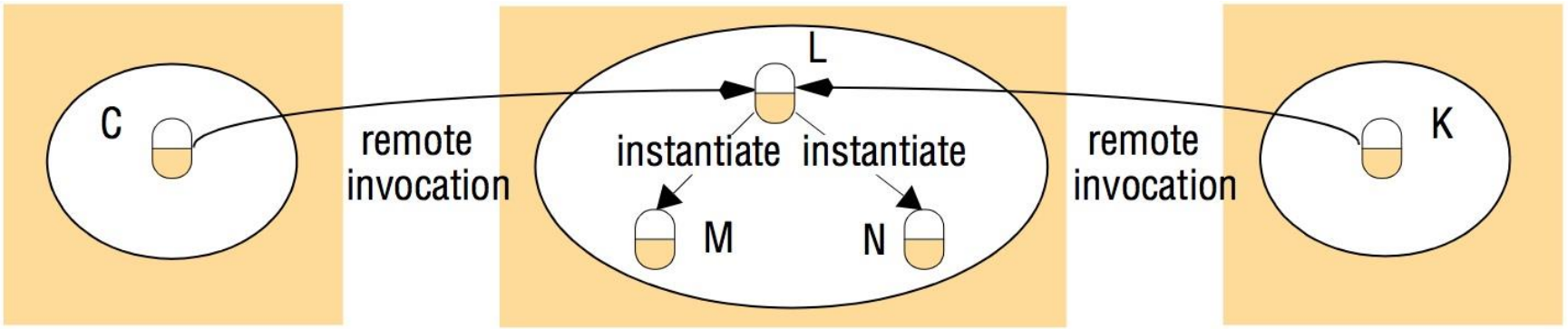
- Remote objects:
  - Each process contains a collection of objects, some of which can receive both local and remote invocations (B and F are remote objects)
  - For transparency, method invocations between objects in different processes, whether in the same computer or not, are known as remote method invocations
- Chains of related invocations:
  - Objects in servers may become clients of objects in other servers (e.g. E)
- Remote object references:
  - Objects can invoke methods of a remote object if they have access to its remote object reference (e.g., A must have a remote object reference for B)
- Remote interfaces:
  - A remote object specifies which of its methods can be invoked remotely through a remote interface (e.g. B and F must have remote interface)



Remote Method Invocation
Instantiation of remote objects

Ciências ULisboa

- Remote instantiation of objects
  - Needs remote objects to have methods for instantiating objects that can be accessed by RMI (e.g., object L)
  - If a newly instantiated object has a remote interface, it will be a remote object with a remote object reference (e.g., objects M and N, created by L, upon the remote invocations from C and K)



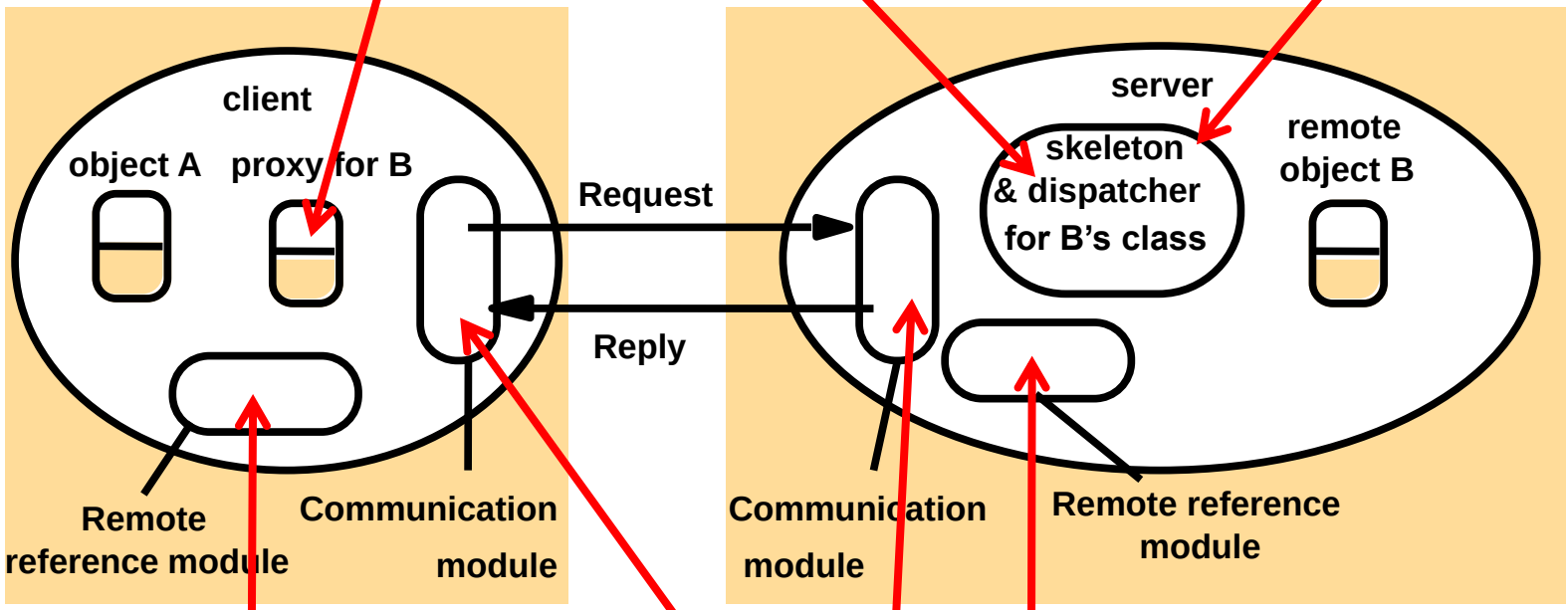
Remote Method Invocation in action

Ciências ULisboa

Proxy - makes RMI transparent to client. Class implements remote interface. Marshals requests and unmarshals results. Forwards request.

Dispatcher - gets request from communication module and invokes method in skeleton (using *methodID* in message).

Skeleton - implements methods in remote interface. Unmarshals requests and marshals results. Invokes method in remote object.



Carries out Request-reply protocol

Translates between local and remote object references and creates remote object references. Uses remote object table

RMI software - between application level objects and communication and remote reference modules

Remote Method Invocation in action

Ciências ULisboa

- **Communication modules**
  - Implement the request-reply protocol between the client and server
  - Responsible for specified invocation semantics, e.g. at-most-once
  - Server-side CM selects the dispatcher for the class of the object to be invoked, passing on its local reference
  - Local ref. is obtained from the remote reference module in return for the remote object identifier in the request message
- **Remote reference module**
  - Called by proxies and skeletons when they are marshalling and unmarshalling remote object references
  - Translates between local and remote (system-wide) object references
  - Creates remote object references in remote object table (ROT)
  - ROT has an entry for all the remote objects held by the process
  - ROT has an entry for all the proxies to remote objects
- **Servants**
  - Instance of a class that provides the body of a remote object
  - Created when remote objects are instantiated

Remote Method Invocation in action

Ciências ULisboa

- **Proxy (client-side)**
  - Makes remote object look like local, to make remote method invocation transparent to clients, and forwards the invocation to the remote object
  - Hides all distribution from client: remote object reference, marshalling of arguments, unmarshalling of results, message sending /receiving
- **Dispatcher (server-side)**
  - One dispatcher and skeleton for each class representing a remote object.
  - Receives request messages from the communication module, and uses the operation ID to select the appropriate method in the skeleton
- **Skeleton (server-side)**
  - Implements the methods of the remote interface
  - Unmarshals request message and invokes corresponding method in servant
  - Gets result and marshals reply message to the sending proxy's method

Ciências ULisboa

Building Client-Server Systems and Applications

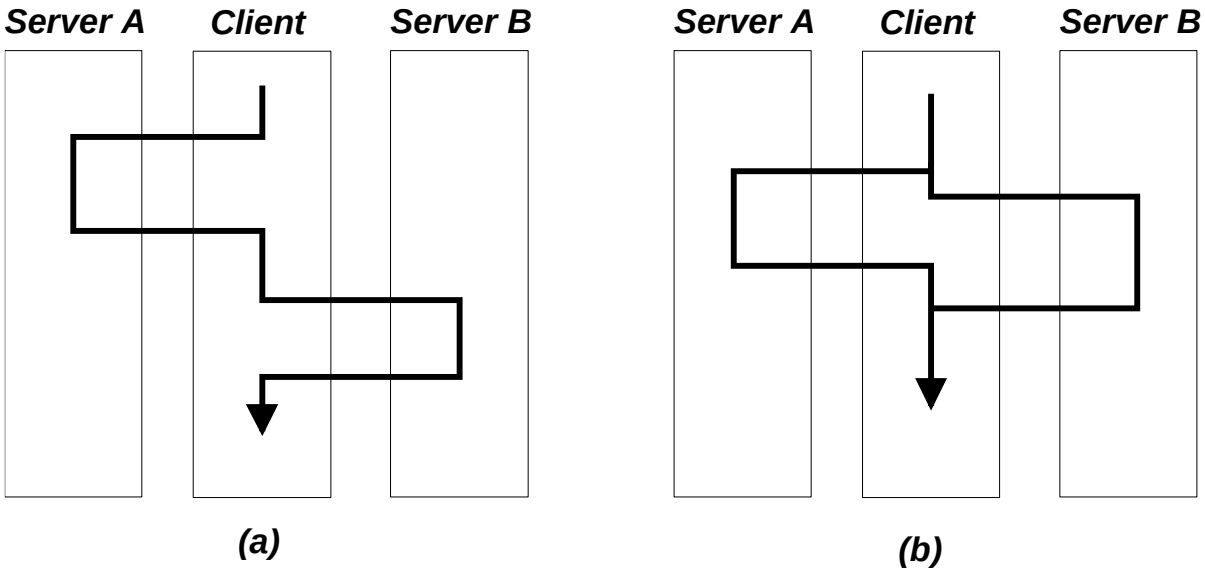
Remote Operations (RPC, RMI, WWW)
Generic Request-reply Interface

Ciências ULisboa

- ***doOperation (RemoteRef, operationID)***
  - Invokes *operationID* on the remote server *RemoteRef* through a request message, and returns the reply to caller
- ***getRequest***
  - Remote server acquires a client request via its port
- ***sendReply (reply, clientHostPort)***
  - Sends the reply message *reply* to the client at its Internet address and port

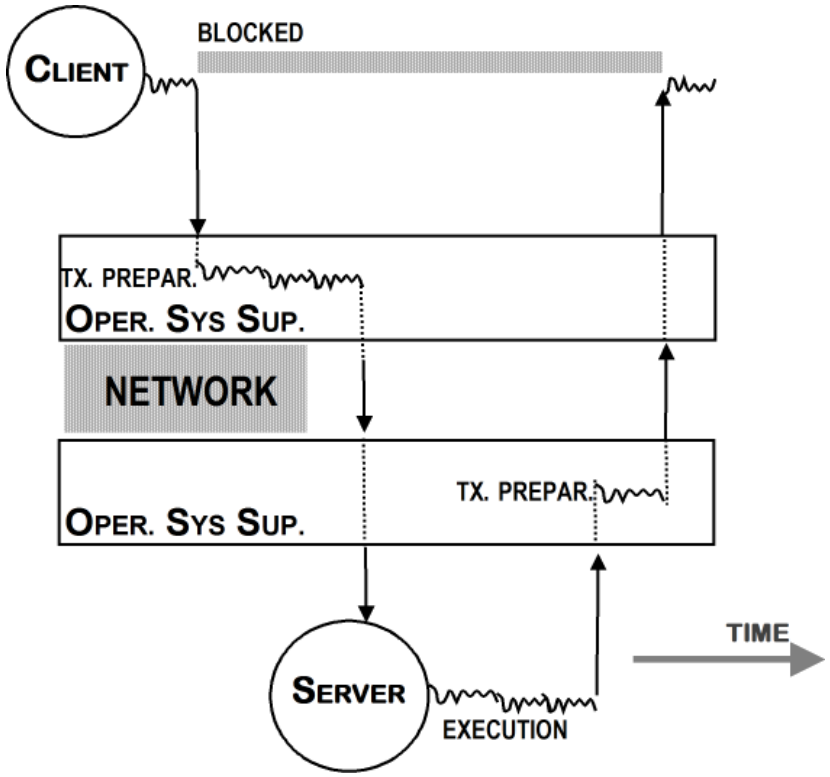
Types of Interface
Client Threads

- Single-threaded client – blocking call
- Multi-threaded client – non-blocking call



Types of Interface
Single-threaded client – blocking call

- When executing a remote operation it may be necessary to preserve the semantics of a local function call
- Client will block for a possibly long time, given that it needs to wait for the reply, to be sure that the call has been executed

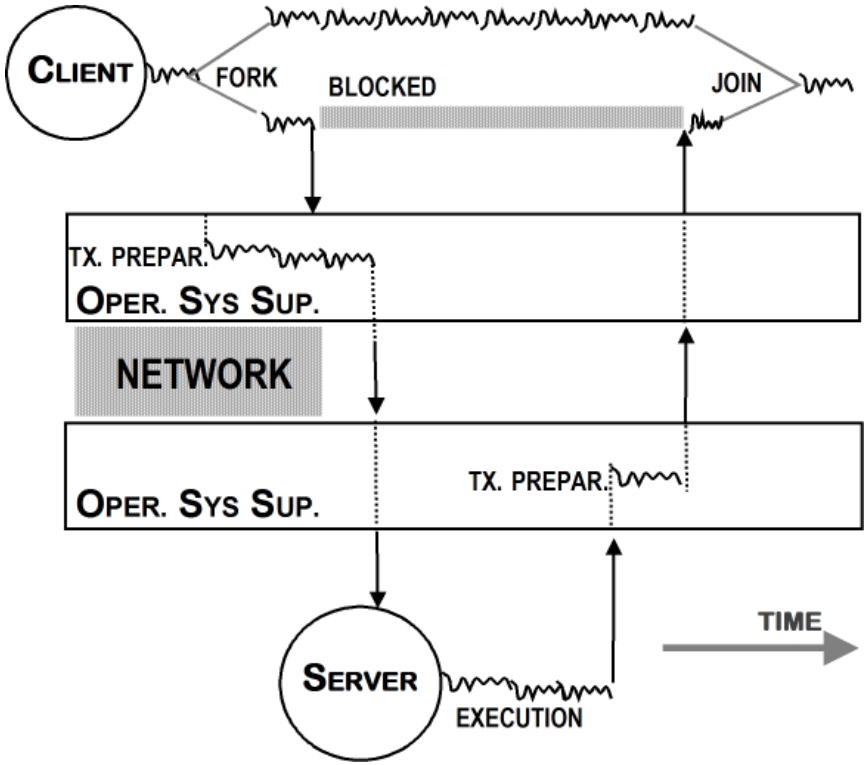


NOTE! A remote operation is performed in closed circuit

Types of Interface
Multi-threaded client – non-blocking call

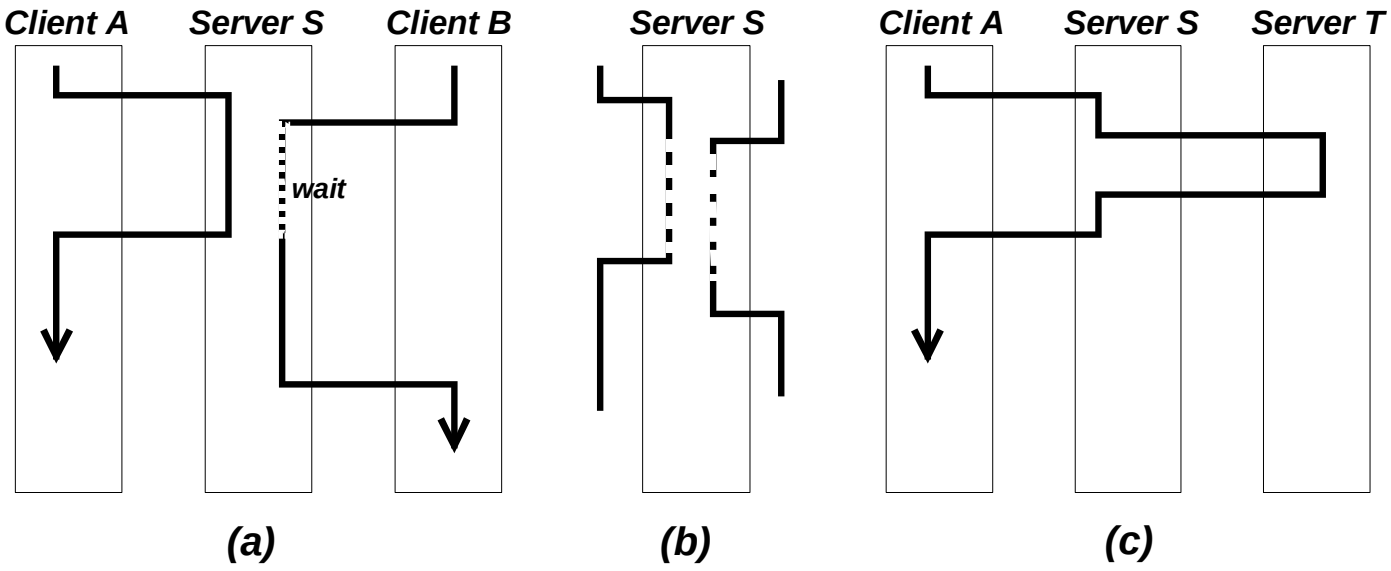
- In multi-threaded systems, the fork-and-call technique allows performance improvements
- Client continues execution until reply arrives, then it joins
- Threads are very efficient for this technique, due to being lightweight
- Requires careful programming

NOTE! After invoking an operation, the application cannot assume that the operation has already been completed – the semantics of a local function call is no longer provided



Types of Interface
Server Threads

- Single-threaded server– serializes subsequent calls
- Multi-threaded server– concurrent calls
- Multi-tiered operations



Client vs. Server Threads

Ciências ULisboa

- Multi-threaded client
  - An RPC is blocking
  - To avoid becoming blocked, the client starts a thread, which executes the RPC and blocks, while the other thread keeps working
  - When the reply arrives, that thread can be terminated
- Multi-threaded server
  - Using thread is very important, to avoid useless serialization of requests
  - The server creates (or assigns) a thread for each request, which is served according to the thread scheduling policy
  - Requests might be served in any order...

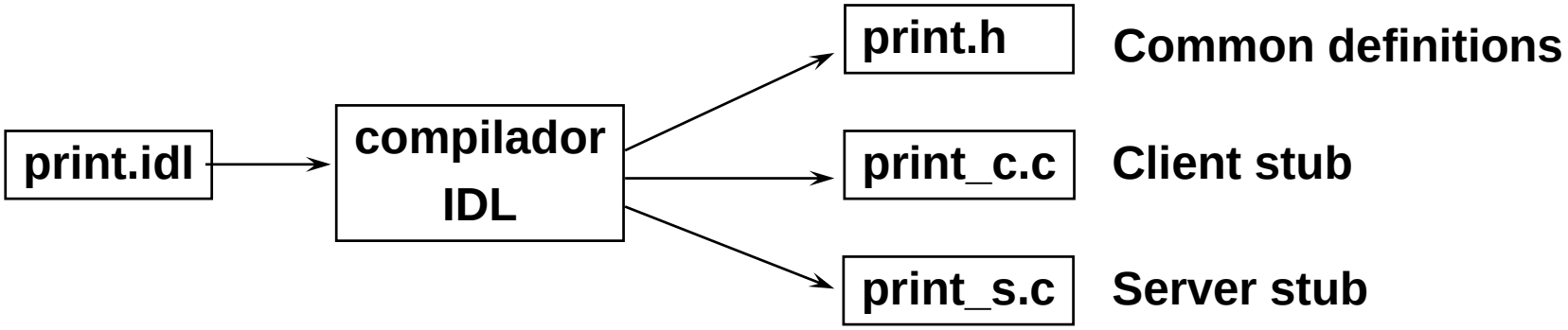
Reliability of Remote Operations
Failure semantics

Ciências ULisboa

- RPC failure semantics: definition of behaviour classes for an RPC subject to communication and/or server failures
  - **at-least-once**: (the service) is executed at least once, but may be executed more times
  - **at-most-once**: it is executed at most once, but may not be executed to completions
  - **zero-or-once**: atomic semantics, like at-most-once but without partial executions
  - **exactly-once**: it is always executed, and only once

Interface Definition Language (IDL)

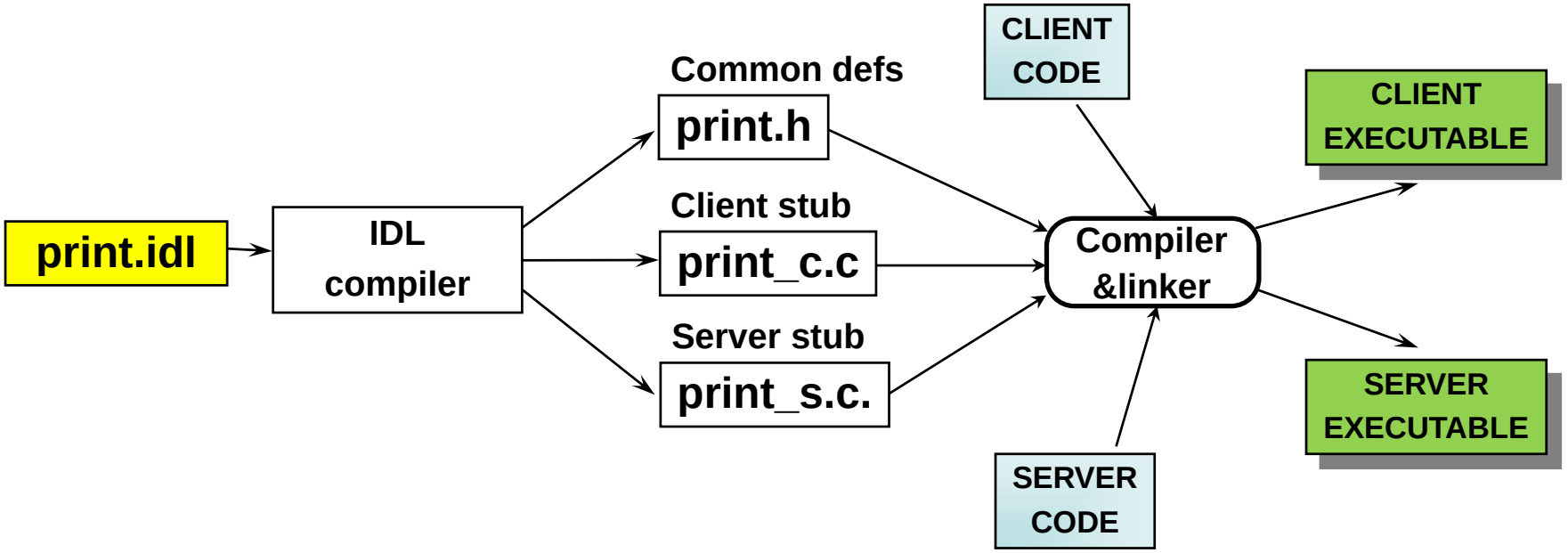
- IDL – Interface Definition Language
  - Language to specify interfaces between clients and servers
  - Describes the properties of the services exported by the server (e.g., procedure names, parameter types)
  - Adds new data types for IPC
  - Clients may invoke the procedures declared in the interface
  - Stubs are automatically generated by an IDL compiler



Design Issues
Developing an RPC subsystem

Ciências ULisboa

- RPC development life cycle



Distributed Computing Models

Main models, neither exhaustive nor air-tight

Ciências ULisboa

- Client-Server (RPC, RMI, WWW) (Cliente-Servidor)
- **Distributed Objects (Objectos Distribuídos)**
- Distributed Shared Memory (DSM, Tuples) (Memória Partilhada Distribuída)
- Distributed Atomic Transactions (Transacções Atómicas Distribuídas)
- Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)
- Stream (Corrente)
- Group-Oriented (Orientado para Grupos)
- Peer-to-peer (Inter-pares)

Distributed Objects Model

Object distribution

Ciências ULisboa

- Advantages
  - Encapsulation inherent in O_O, enhanced in distributed objects
  - Accessed via RMI or accessed directly if class available locally
  - Heterogeneous systems in different data formats at different sites may be used in the same application
  - Distributed objects systems may adopt client-server or other architectural models
  - Dynamic and extensible, e.g. by enabling introduction of new objects
- Distributed object middleware offers a programming abstraction based on object-oriented principles
  - Examples: Java RMI, CORBA
  - **Java RMI**: restricted to Java-based development
  - **CORBA**: multi-language, multi-platform solution allowing objects in different platforms, written in a variety of languages to interoperate

Object distribution vs. classic O_O

Objects	*Distributed objects*	*Description of distributed object*
Object references	Remote object references	Globally unique reference for a distributed object; may be passed as a parameter.
Interfaces	Remote interfaces	Provides an abstract specification of the methods that can be invoked on the remote object; specified using an interface definition language (IDL).
Actions	Distributed actions	Initiated by a method invocation, potentially resulting in invocation chains; remote invocations use RMI.
Exceptions	Distributed exceptions	Additional exceptions generated from the distributed nature of the system, including message loss or process failure.
Garbage collection	Distributed garbage collection	Extended scheme to ensure that an object will continue to exist if at least one object reference or remote object reference exists for that object, otherwise, it should be removed. Requires a distributed garbage collection algorithm.

ODP Model
Open Distributed Processing

Ciências ULisboa

- Reference model defined by ISO and ITU-T in the late 80s, which led to the RM-ODP standard in the mid-90s
- Standardization of “open distributed computing”
  - Set of international recommendations and standards
  - Conceptual framework integrating aspects of software distribution, interoperability and portability to achieve transparency regarding:
    - Hardware heterogeneity, operating systems, networks, programming languages, databases, …
- Object-based system specification
  - An object contains information and offers services
  - System is composed by objects and interactions between objects

ODP and CORBA

Ciências ULisboa

- ODP – Open Distributed Processing
  - Reference model, object-based, for open systems
  - First one defining brokers that are able to do trading and binding between client and server objects
- CORBA – Common Object Request Broker Architecture
  - Reference architecture, by OMG (Object Management Group)
  - From *interoperability in O_O systems* evolved to *O_O distributed computing platform*
  - Interoperability across platform vendors: through
- Related proposals: Active/X (Microsoft)
  - Replaced COM and OLE in the late 90s
  - Windows-oriented/dependant
  - Now deprecated

CORBA

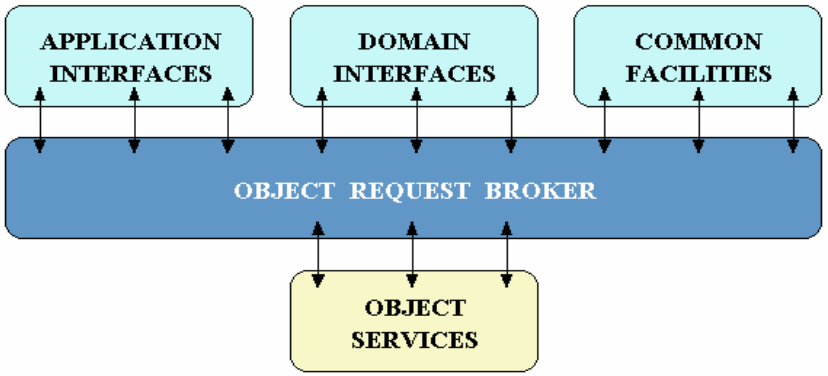
Ciências ULisboa

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- Build systems based on different architectures, with **object** as the integrator concept:
  - Encapsulations
    - «I don't care about how a car is made, I will just be driving it»
    - Implementation is hidden from the interface definition
  - Polymorphism
    - «I know how to drive a Fiat; I should be able to drive a Ford»
    - Coherent representation of procedures in all architectures
  - Inheritance
    - «I can drive a car; should be easy to learn how to drive a truck»
    - Selective reuse of existing definitions on new objects
- Do that in a distributed way: **object broker**

The CORBA Architecture
Overview

Ciências ULisboa

- **Object Services**:
  - System-level services (e.g. naming, events, persistent objects (files), transactions)
- **ORB – Object Request Broker**:
  - Interconnects these components and helps clients invoke methods on distr. objects
- **Common Facilities**:
  - User-oriented, shared by several applications (e.g., document management)
- **Domain Interfaces**:
  - Domain specific facilities (e.g., telecom, health)
- **Application Interfaces**:
  - Transformer objects for specific black-box legacy applications (e.g., Office)



CORBA Object Services (1)

Ciências ULisboa

CORBA Service	*Role*	*Further details*
Naming service	Supports naming in CORBA, in particular mapping names to remote object references within a given naming context (see Chapter 9).	[OMG 2004b]
Trading service	Whereas the Naming service allows objects to be located by name, the Trading service allows them to be located by attribute; that is, it is a directory service. The underlying database manages a mapping of service types and associated attributes onto remote object references.	[OMG 2000a, Henning and Vinoski 1999]
Event service	Allows objects of interest to communicate notifications to subscribers using ordinary CORBA remote method invocations (see Chapter 6 for more on event services generally).	[Farley 1998, OMG 2004c]
Notification service	Extends the event service with added capabilities including the ability to define filters expressing events of interest and also to define the reliability and ordering properties of the underlying event channel.	[OMG 2004d]

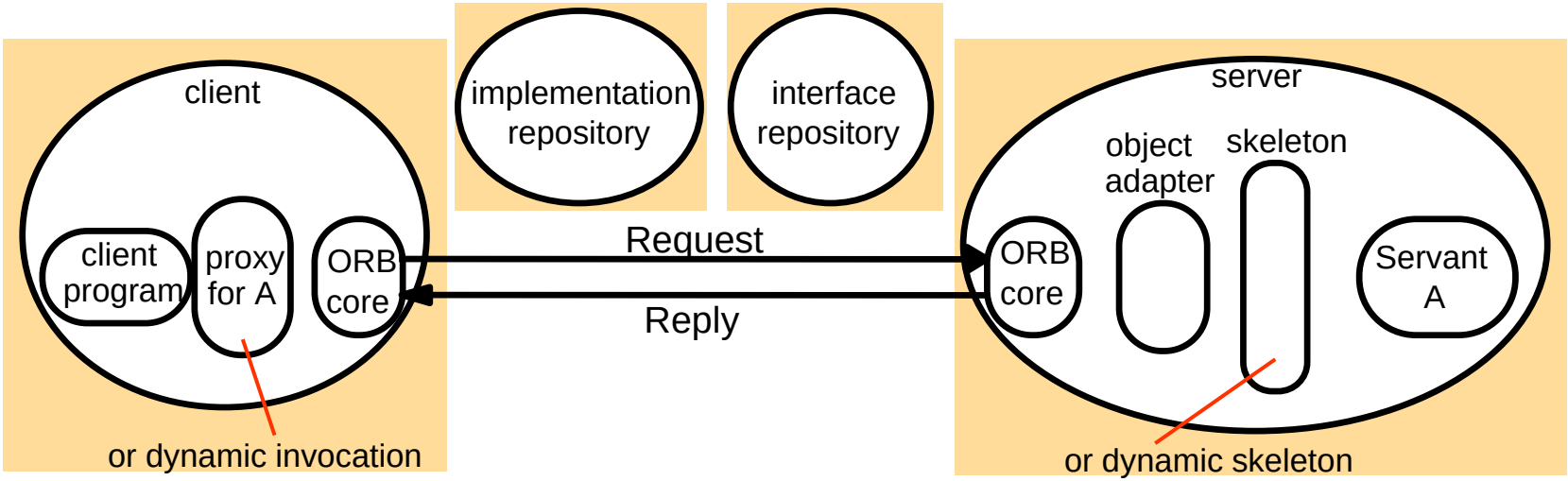
CORBA Object Services (2)

Ciências ULisboa

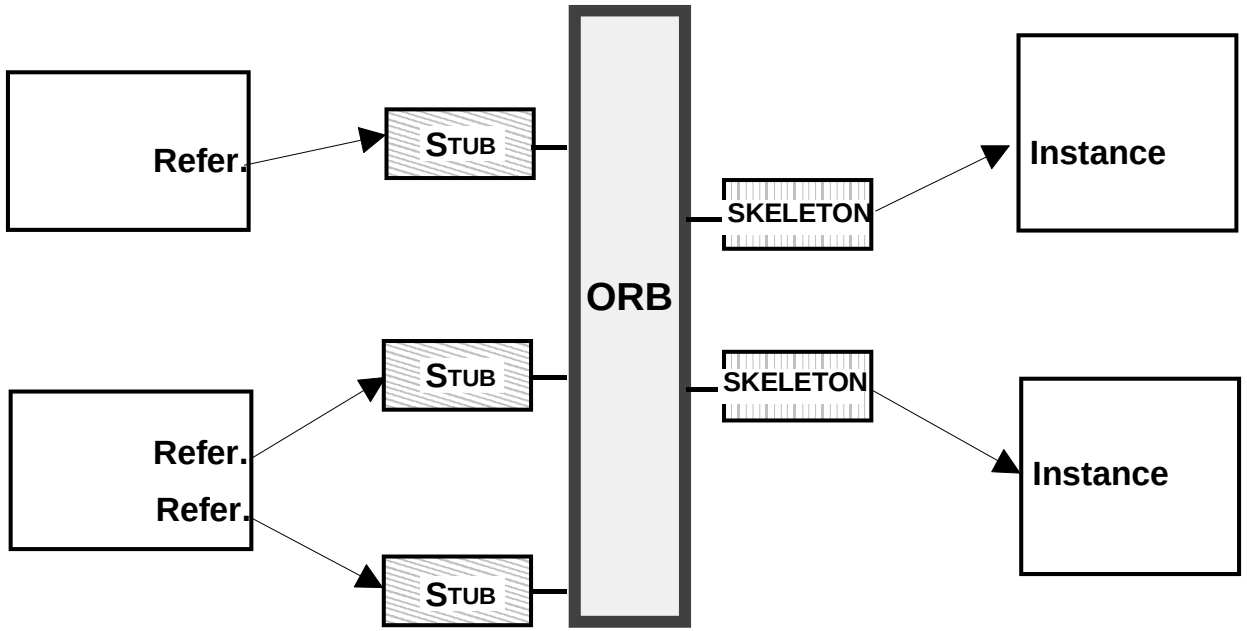
Security service	Supports a range of security mechanisms including authentication, access control, secure communication, auditing and nonrepudiation (see Chapter 11).	[Blakely 1999, Baker 1997, OMG 2002b]
Transaction service	Supports the creation of both flat and nested transactions (as defined in Chapters 16 and 17).	[OMG 2003]
Concurrency control service	Uses locks to apply concurrency control to the access of CORBA objects (may be used via the transaction service or as an independent service).	[OMG 2000b]
Persistent state service	Offers a persistent object store for CORBA, used to save and restore the state of CORBA objects (implementations are retrieved from the implementation repository).	[OMG 2002d]
Lifecycle service	Defines conventions for creating, deleting, copying and moving CORBA objects; for example, how to use factories to create objects.	[OMG 2002e]

Building distributed object-oriented applications

CORBA invocation model



CORBA invocation model



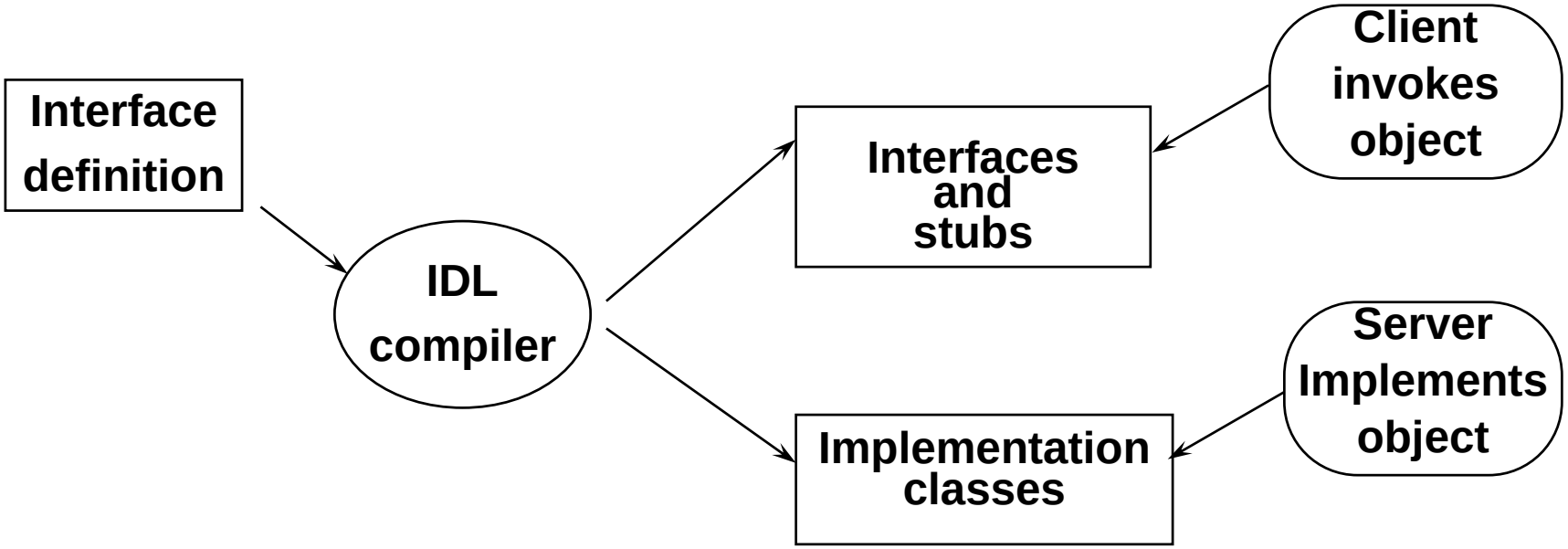
Stubs and Skeletons

Ciências ULisboa

- The IDL compiler generates what are known as **client stubs** and **server skeletons**
- Client stubs and server skeletons serve as a sort of "glue" that connects language-independent IDL interface specifications to language-specific implementation code
- A Client **Stub**, is a small piece of code that makes a particular CORBA server interface available to a client
- A Server **Skeleton**, is a piece of code that provides the "framework" on which the server implementation code for a particular interface is built

CORBA IDL

- Available for several languages (e.g. C++, Java)
- Automatic generation of object invocation interfaces and remote invocation stubs on the client side, and the classes for implementing the services on the server side



Inter-ORB Protocols

Ciências ULisboa

- CORBA General Inter-ORB Protocol (GIOP)
  - Standard for communication between various CORBA ORBs
  - CORBA specification is neutral with respect to network protocols
- Internet Inter-ORB Protocol (IIOP)
  - Specialization of the GIOP, the standard protocol for communication between ORBs on TCP/IP based networks
- Essentially, CORBA applications are built on top of GIOP-derived protocols such as IIOP
  - These protocols, in turn, rest on top of TCP/IP, DCE, or whatever underlying transport protocol the network uses