# Distributed Systems
# Direct Communication
# PART II

Wouter Joosen

DistriNet, KULeuven

October 3, 2023






# This chapter: overview

- Part I
  - Data representation
  - Message passing
  - Request-reply protocols
  - Remote procedure calls
- **Part II**
  - **Object request brokers**

# Object request brokers

- Basics:

Object request broker =

Objects +

RPC

+ ...

# Object request brokers

- Examples:
  - CORBA (Common Object Request Broker Architecture)
  - DCOM (Distributed Component Object Model)
  - Java RMI
  - .NET Remoting

# Object request brokers

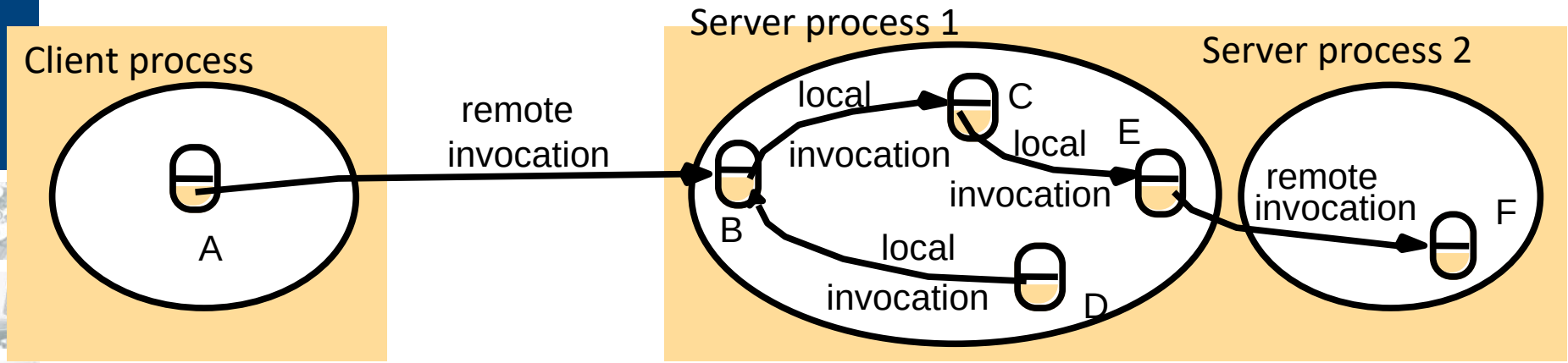
- Overview
  - distributed object systems
  - case study: Java RMI

# ORB: Distributed object systems

- Why objects?
  - …
- Why distributed objects?
  - Maps naturally to distributed services
  - Services are modeled as objects
  - Functionality emerges by cooperation between services and clients

# ORB: Distributed object model

- Local ⇔ remote invocation



# ORB: Distributed object model

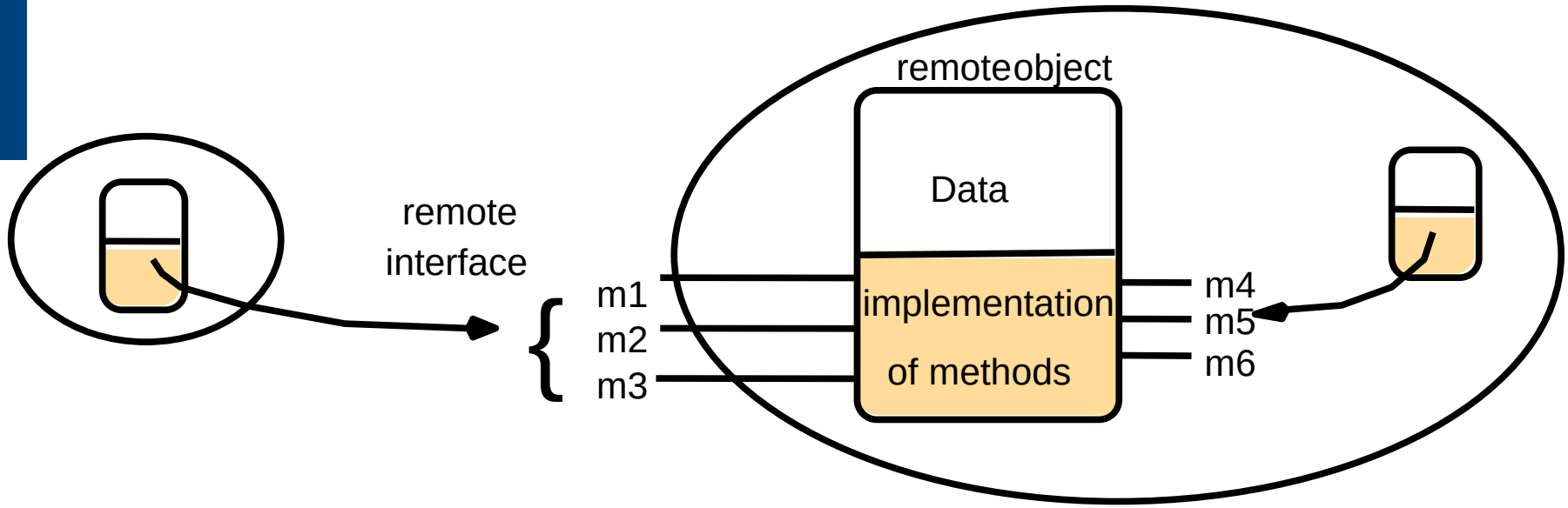
- Requirements
  - synchronous invocation semantics
    - location transparency: location transparent object references
    - access transparency: local & remote object invocations identical
  - inheritance
  - polymorphism

# ORB: Distributed object model

- A distributed object consists of
  - an implementation-independent service definition specified by an interface
    - method signature
    - multiple inheritance
  - a concrete service implementation
    - implements one or more interfaces
    - provides the concrete implementation

# ORB: Distributed object model

- Remote object & its remote interface



# ORB: Distributed object model

- Distributed objects act like normal objects
  - can be used as parameters and return values
  - can be invoked (across address space boundaries)
    - Location transparency
    - Access transparency
  - can be subclassed (inheritance)
  - are invoked synchronously

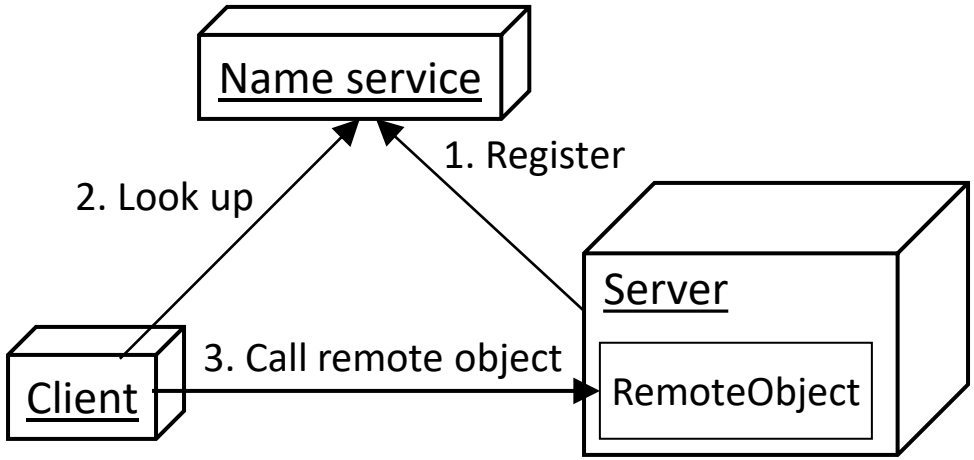
# ORB: Distributed object model

- But there is some complexity (*non-transparent!*)
  - Invocation semantics
    - Java RMI: at most once
    - CORBA: at most once or maybe (if no result)

    } ⇔ exactly once
  - Parameter passing
    - distributed objects are passed by reference (≠ pointer)
    - normal objects are passed by value
  - Additional exceptions due to distribution
    - distribution is not transparent
  - Concurrent access
  - Latency (performance)

# ORB: Distributed object model

- Typical architecture of distributed object systems:
  - 2 separate processes: client & server
    - Server hosts remote object(s) and waits for incoming calls.
    - Client obtains references to remote objects and makes calls.
  - Name service
    - Helps clients to locate remote objects from their remote object references. (name ⇒ address)



# ORB: Distributed object model

- Distributed garbage collection
  - Garbage collection of remote objects.
  - In cooperation with local garbage collector.
- Example: Java distributed garbage collection algorithm:
  - Reference counting:

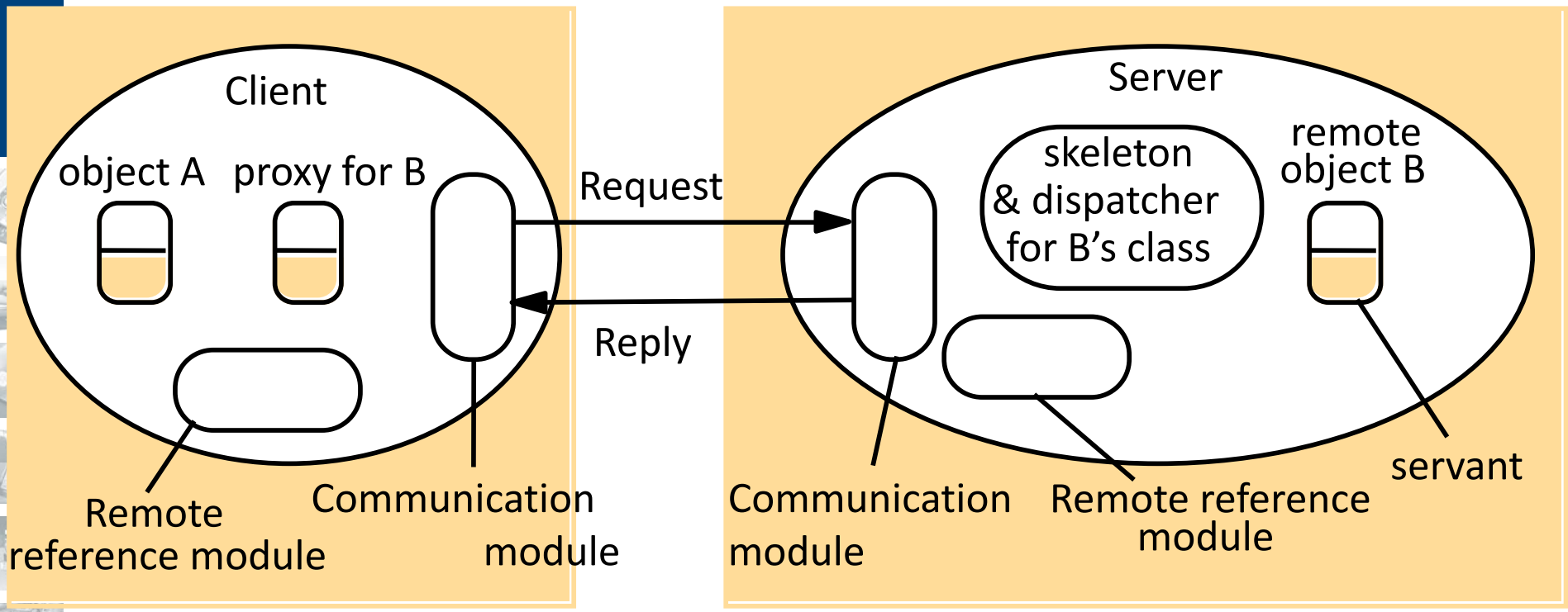
    ***if*** distributed reference count == 0

    ***garbage collect*** remote object
  - Leasing-based: client obtains lease for a period of time.

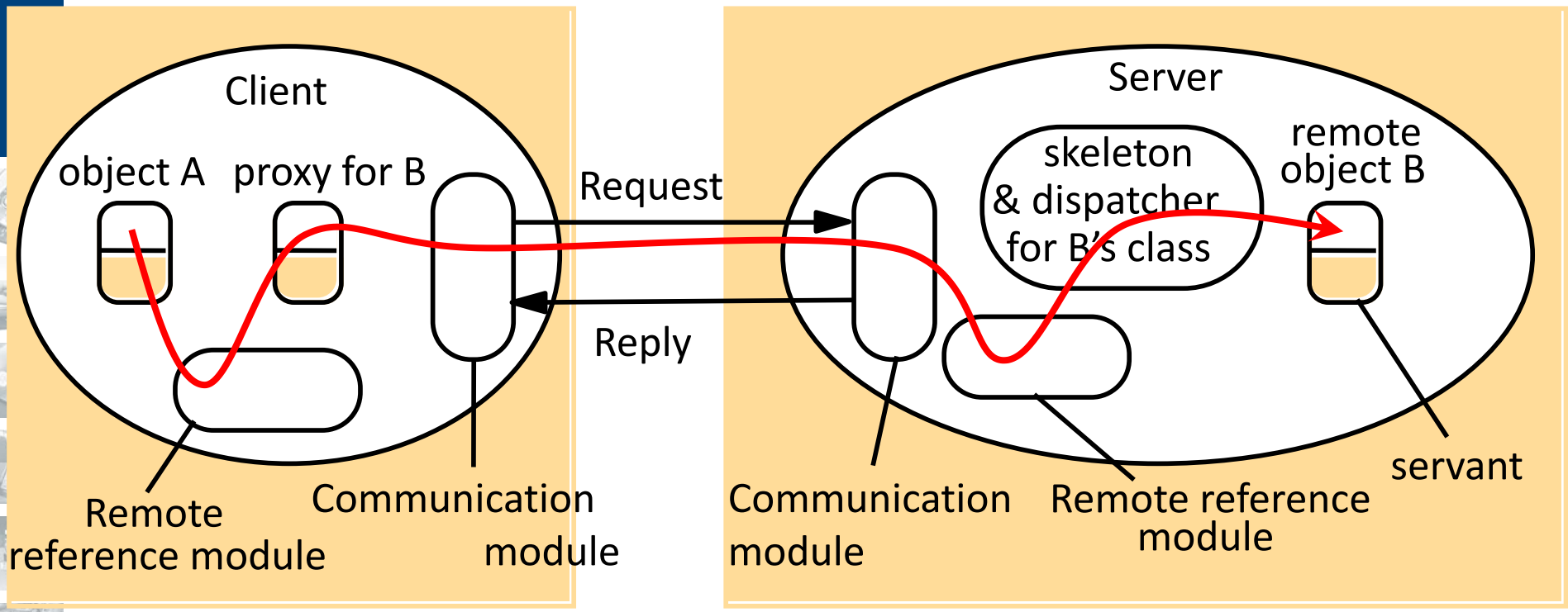
    ➔Tolerates failure of client processes

    ➔Tolerates failure of communication

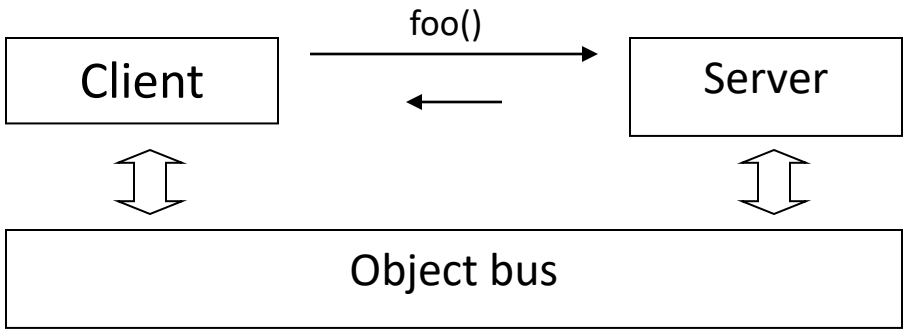
# ORB: Object invocations



# ORB: Object invocations

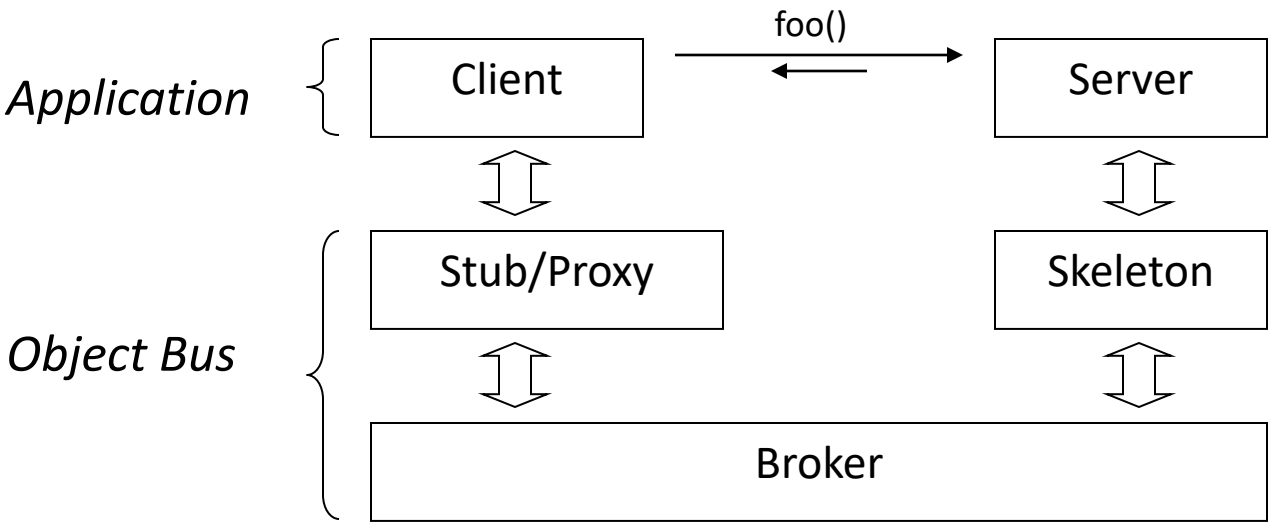


# ORB: The object bus




Client - the invoking application object
Server - the invoked application object

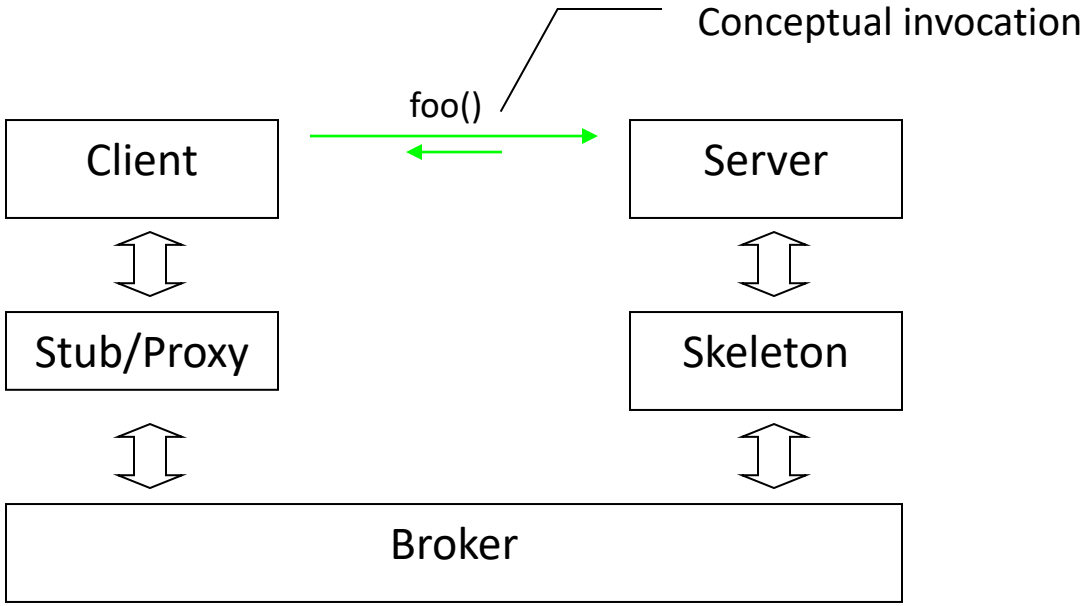
# ORB: The object bus



# ORB: The object bus

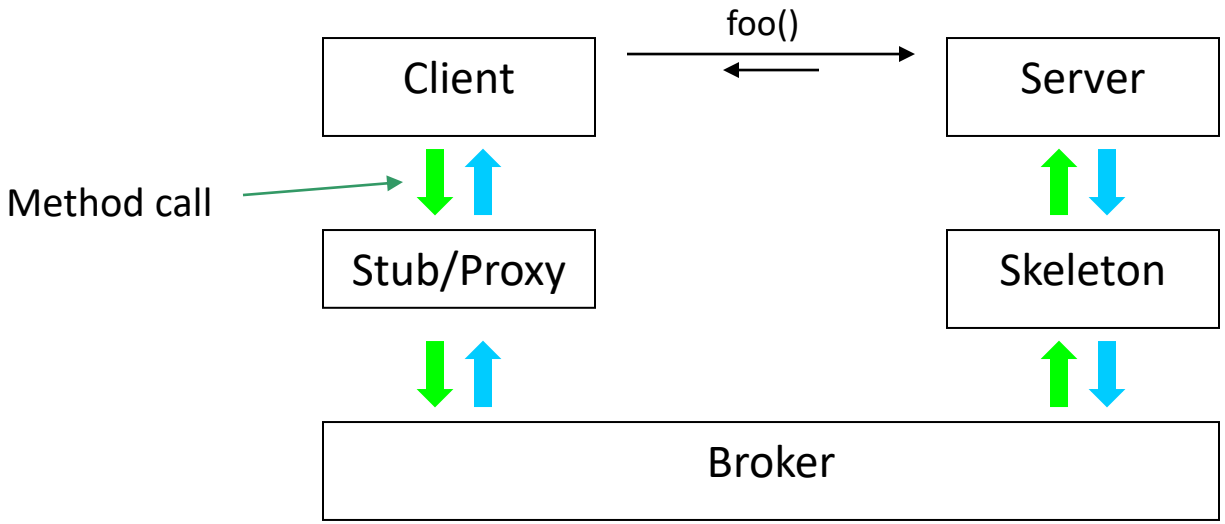
- Client - the invoking application object
- Server - the invoked application object
- Stub/Proxy - reifies the invocation
- Skeleton - dispatches the invocation to the actual object implementation
- Broker - invocation distributor

# ORB: The object bus



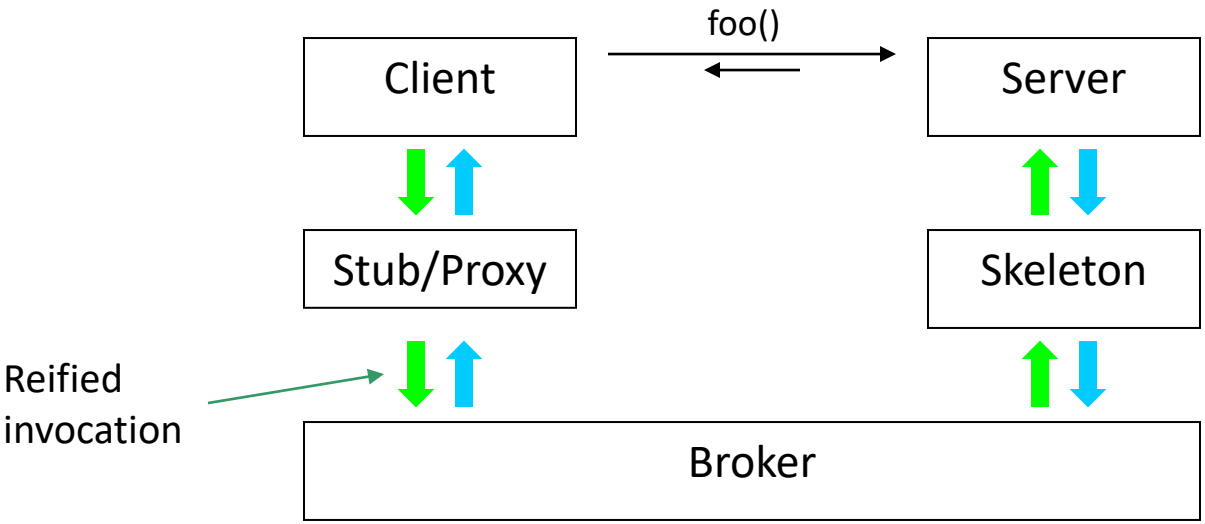
# ORB: The object bus

- Look up remote object (in remote reference module)
  - Not local ⇨ Create / return stub
  - Invoke stub



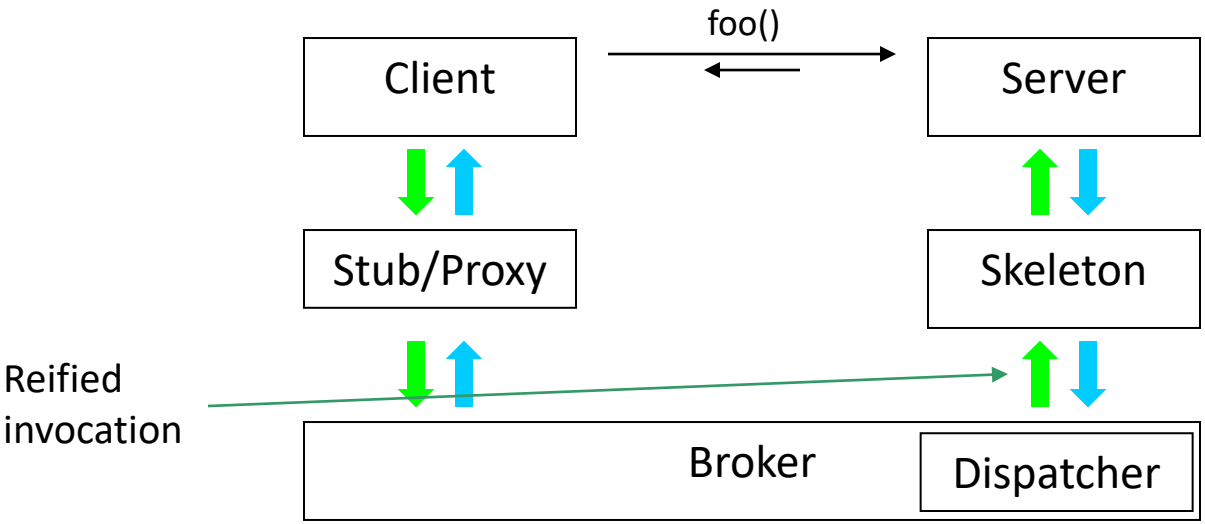
# ORB: The object bus

- Forward invocation in a message to the remote object
  - Marshalling of object reference, method name and arguments into *request* message



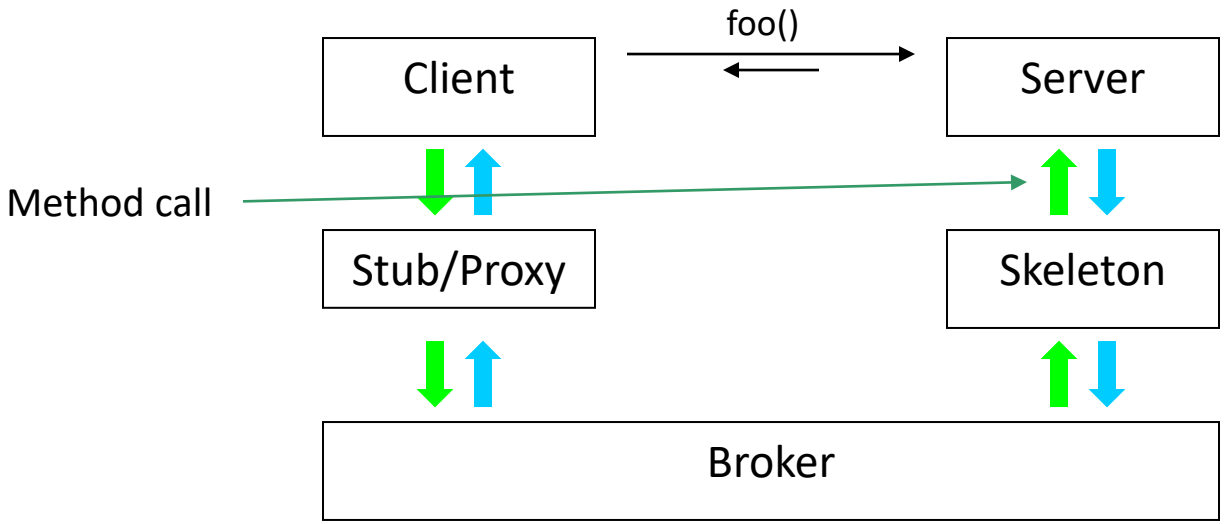
# ORB: The object bus

- Look up remote object (at server side)
  - Local ⇨ pass local reference of target object to dispatcher
  - Dispatcher sends *request* to appropriate method in skeleton



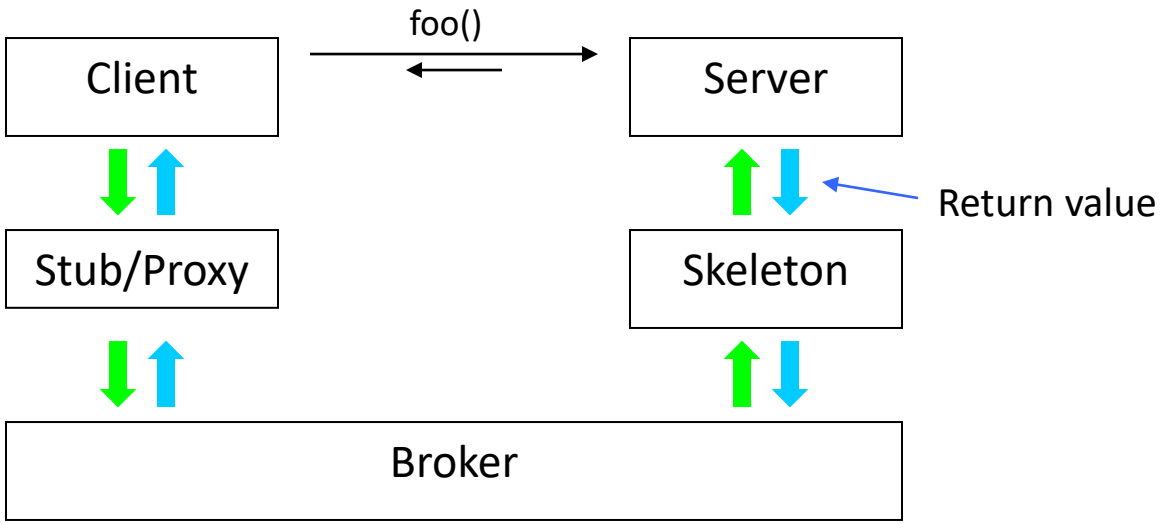
# ORB: The object bus

- Forward invocation to actual remote object (i.e. server)
  - Unmarshalling arguments in *request* message
  - Invoking corresponding method of target object



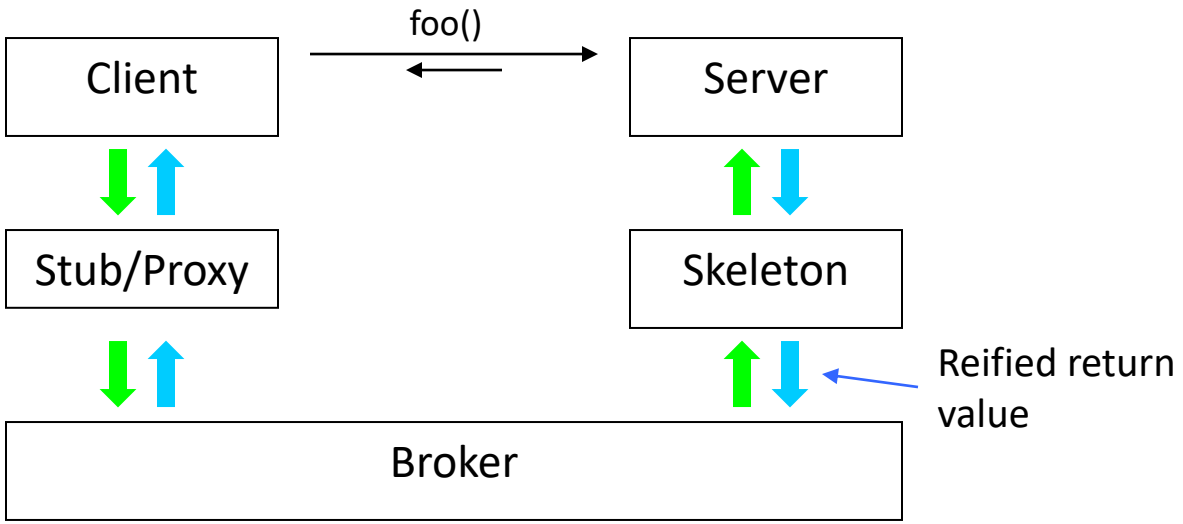
# ORB: The object bus

- Return result to skeleton



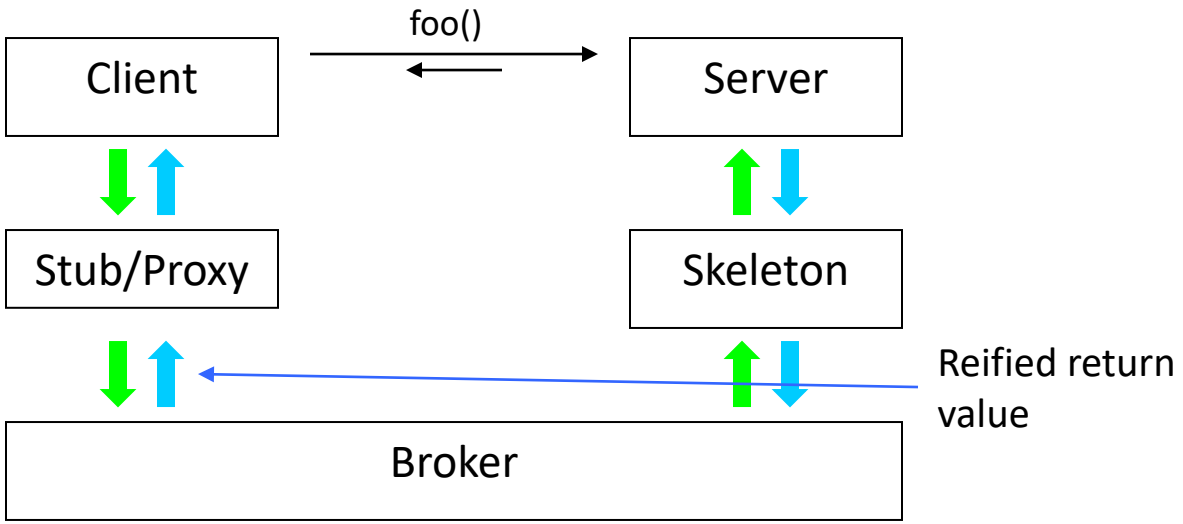
# ORB: The object bus

- Forward return in a message to the waiting stub
  - Marshalling of return value (and exceptions) into *reply* message



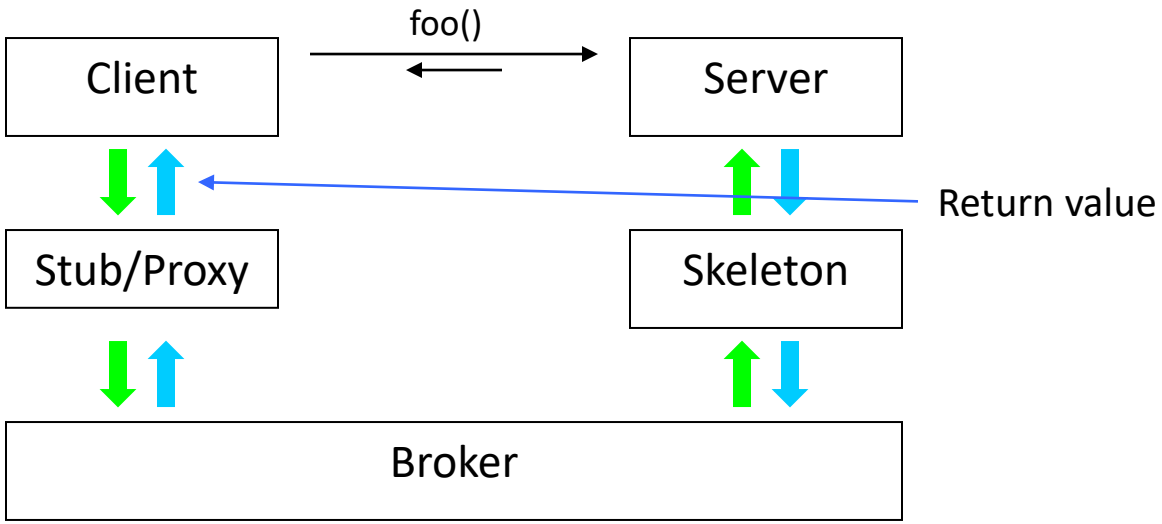
# ORB: The object bus

- Waiting for *reply* message



# ORB: The object bus

- Return result to invoking object (i.e. client)
  - Unmarshalling *reply* message



# Questions?