Microsoft .NET

- Group 4
  - Geir Arveschoug Erikstad
  - Tommy Gudmundsen
  - Christian Mikalsen
- Agenda
  - Introduction to the .NET framework
  - Introduction to COM and COM+
  - .NET remoting

.NET Framework

- The common language runtime
- Class libraries

Common Language Infrastructure

- Open standard from Microsoft
- Describes the executable code and runtime environment that form the core of the Microsoft .NET Framework

Common Language Runtime

- Microsofts implementation of CLI
- Code that targets the runtime is called managed code
- Features cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services

Common Type System

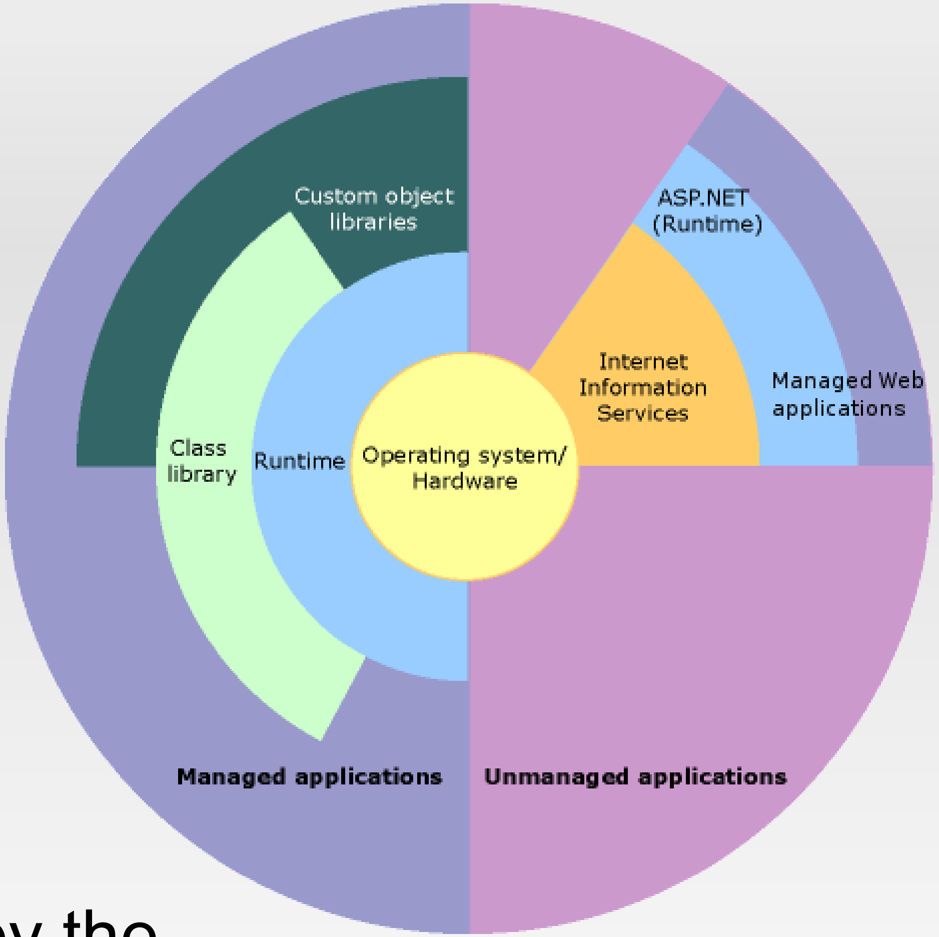
- Enable cross-language integration, type safety, and high performance code execution
- Object-oriented model that supports the complete implementation of many programming languages
- Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other

Common Language Specification

- Set of basic language features needed by many applications
- Subset of the common type system
- If you only use CLS features in the API that it exposes to other code -> component is guaranteed to be accessible from any programming language that supports the CLS

Managed code and data

- Managed code
  - Targets the services of the common language runtime
  - C#, VB.net, JScript.net is managed by default
  - Visual Studio .NET C++ is not managed by default
- Managed data
  - Allocated and de-allocated by the CLR's garbage collector
  - C#, VB.net, JScript.net by default, can turn of in C#
  - C++ not by default, but possible with Managed Extensions

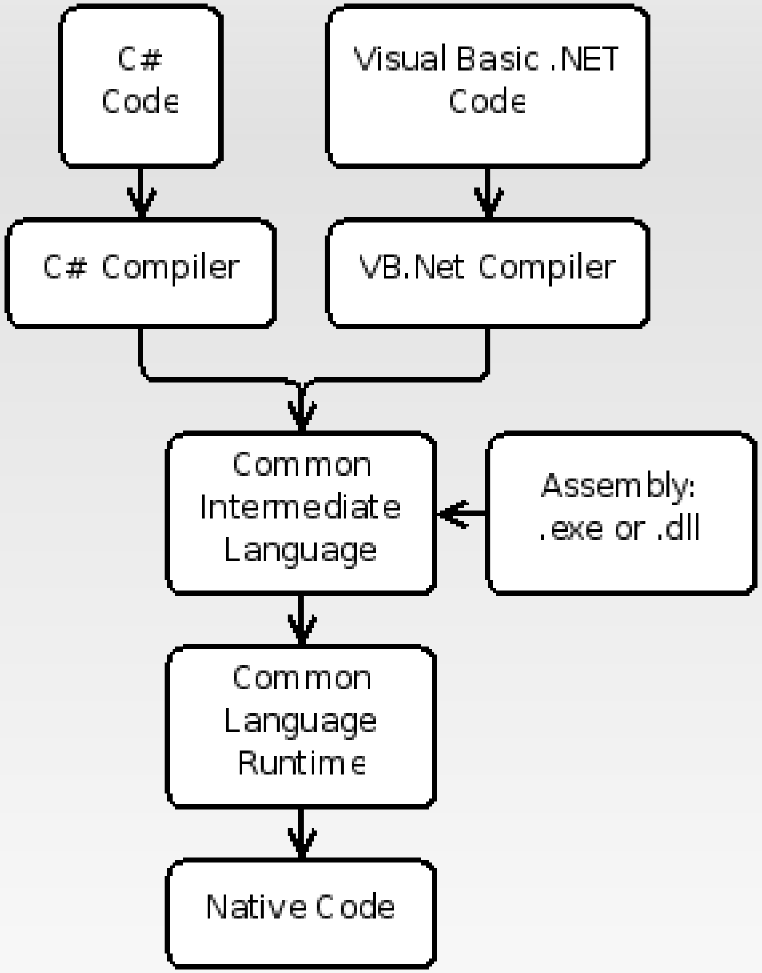


Assembly

- Primary building block of a .NET Framework application
- A collection of functionality that is built, versioned, and deployed as a single implementation unit (as one or more files)
- Self-describing by means of their manifest

From high level code to native code

- Compiled to an assembly
- CLR uses a JIT-compiler to make native code
- Must pass verification



.NET components

- Special type of executable built from a .NET project
- .NET components provide a programmable interface that is accessed by consumer (client) applications
- Built and tested as independent .NET projects
- Can be added to many .NET applications as plug-in service providers

Introduction to COM and COM+

- COM was introduced by Microsoft in 1993
  - Originated from DDE and OLE
    - Object Architecture: Dealing with the Unknown or Type Safety in a Dynamically Extensible Class (1988)
    - On Inheritance: What It Means and How To Use it (1990)
- A language-neutral way of implementing objects such that they can be used in environments different from the one they were created in, also across machine boundaries.

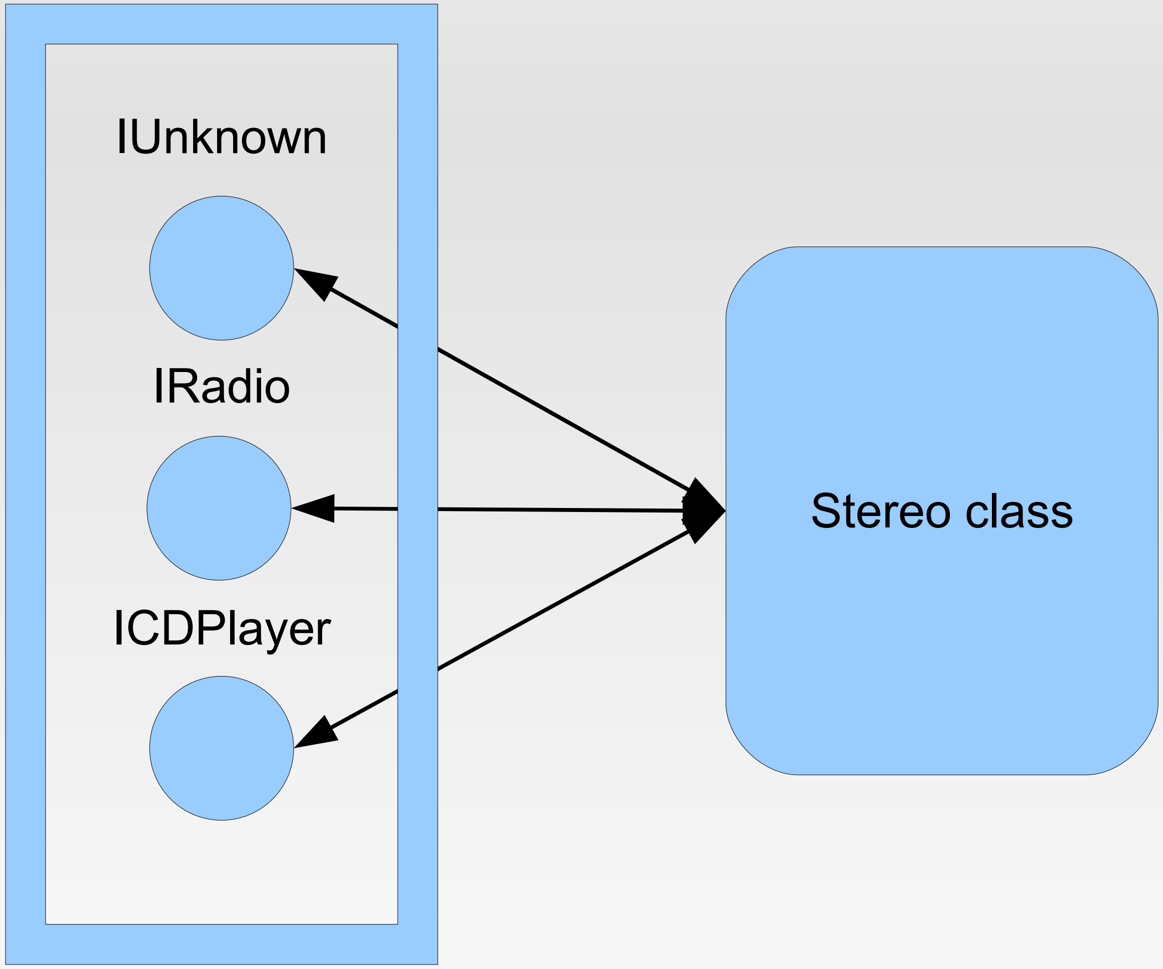
COM+

- Extension of the original COM
- Originated with DCOM
  - Goal of DCOM is to provide support for components distributed on different machines.
- Offers a framework of distributed functionality
  - Makes COM easier to use
  - Distributed transactions
  - Resource pooling
  - Thread management
- COM/COM+ are used interchangeably

Classes and interfaces in COM

- Interfaces
  - Interfaces play a vital role in COM (point of contact)
  - Interfaces are identified by a IID/GUID.
  - COM defines a binary standard for interfaces.
- CoClasses
  - A CoClass contains a concrete implementation of one or more interfaces, and is identified by a CLSID.
  - All CoClasses are required to implement *IUnknown*, which contains the functions *AddRef*, *Release* and *QueryInterface*.

Classes and interfaces in COM



Interfaces retrieved using *QueryInterface*

IUnknown

- *AddRef* and *Release*
  - Used to keep reference count of components.
- *QueryInterface*
  - Can be called to check if a component implements a specific interface (IID). If the interface is supported, an interface pointer is returned.
  - Used to provide casting between the different interfaces of a component.

COM servers

- A COM server contains one or more CoClasses, and is either a DLL or executable.
- COM servers need to be registered in Windows before they can be used, which typically adds an entry to the Windows registry.

Interface Definition Language

- Interfaces, classes and types are specified in an Interface Defintion Language (IDL).
- The IDL can be compiled into header files and/or proxy objects in different languages.
- IDL can also be compiled into type library (TLB) files, which can be imported into many popular IDEs.

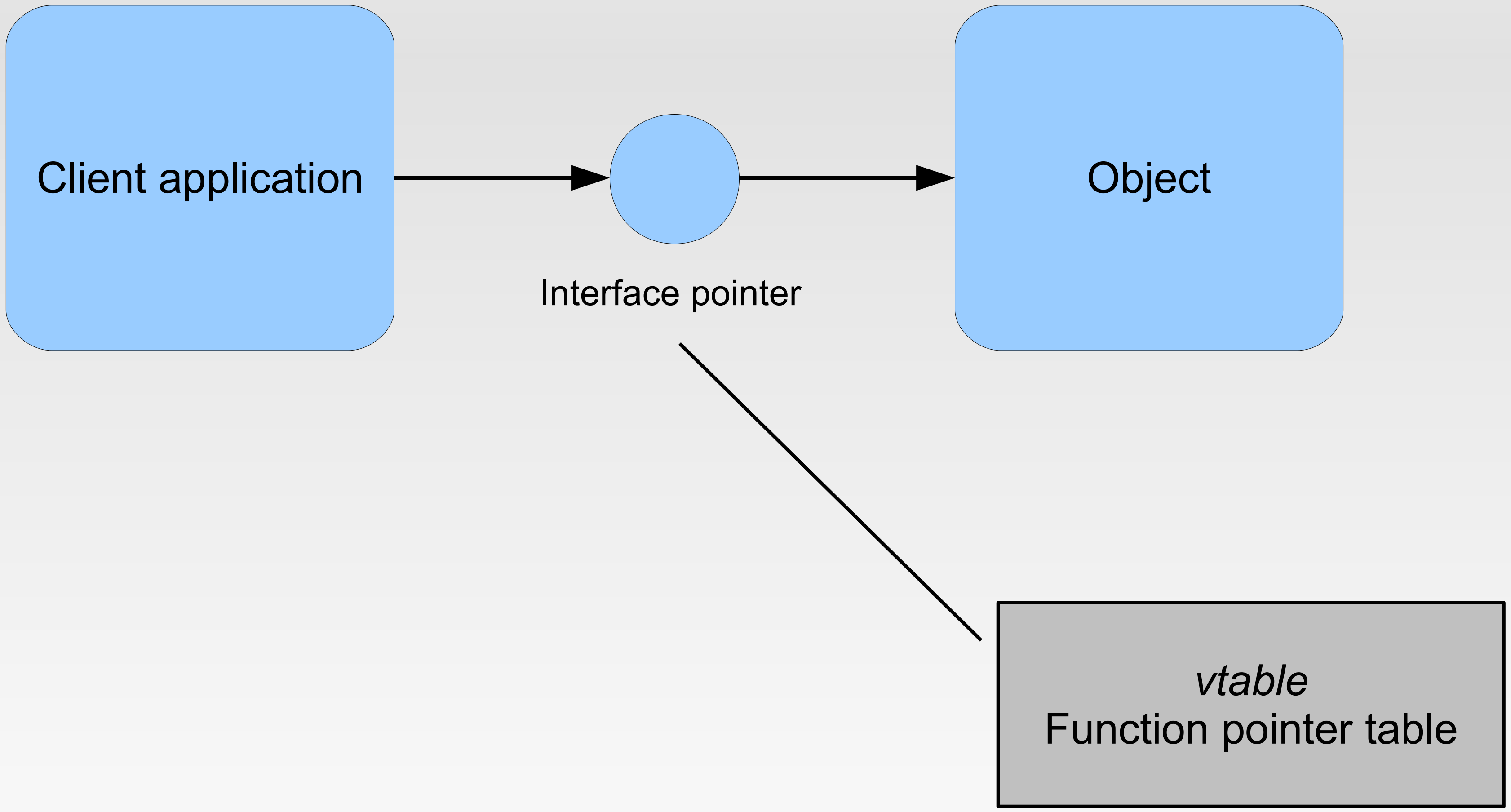
Object in activation in COM

- Clients call *CoCreateInstance*
  - Wanted CLSID and IID is specified as parameters.
  - A Service Control Manager performs a lookup in the COM database and instantiates a server process.
  - The function returns an interface pointer to the requested interface of the newly created CoClass.
  - Typically, class factories are used to create instances. Class factories are themselves CoClasses implementing IClassFactory.

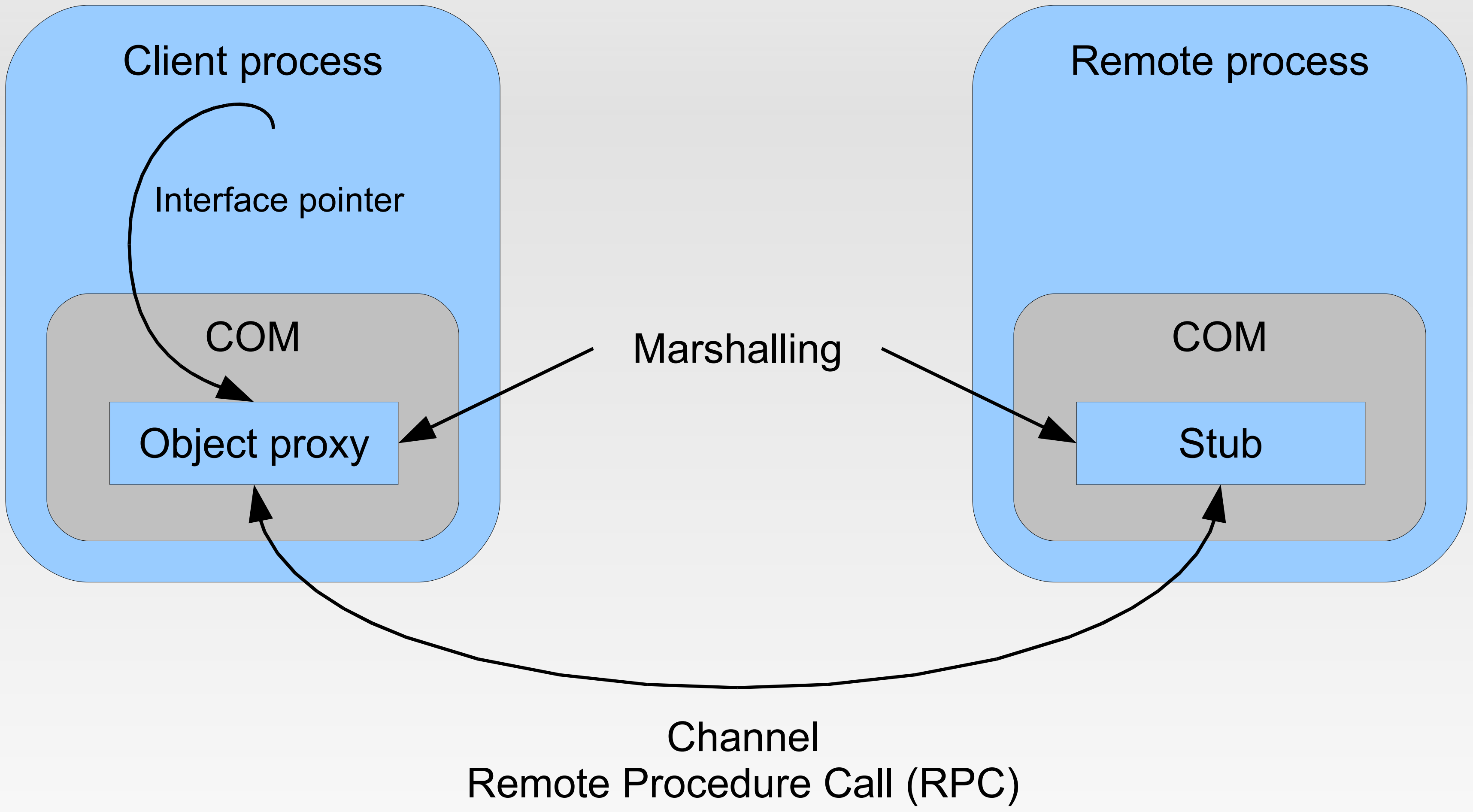
Method invocations in COM

- **In-process server:**
  A client talks directly to the library containing the server.
- **Local Object Proxy:**
  A client talks to a server running in a different process (on the same machine) through interprocess-communication. This is similar to a lightweight Remote Procedure Call.
- **Remote Object Proxy:**
  A client talks to a server running on another machine. Communication is handled through RPC, previously distinguished by the name DCOM.

In-process server



Remote object proxy

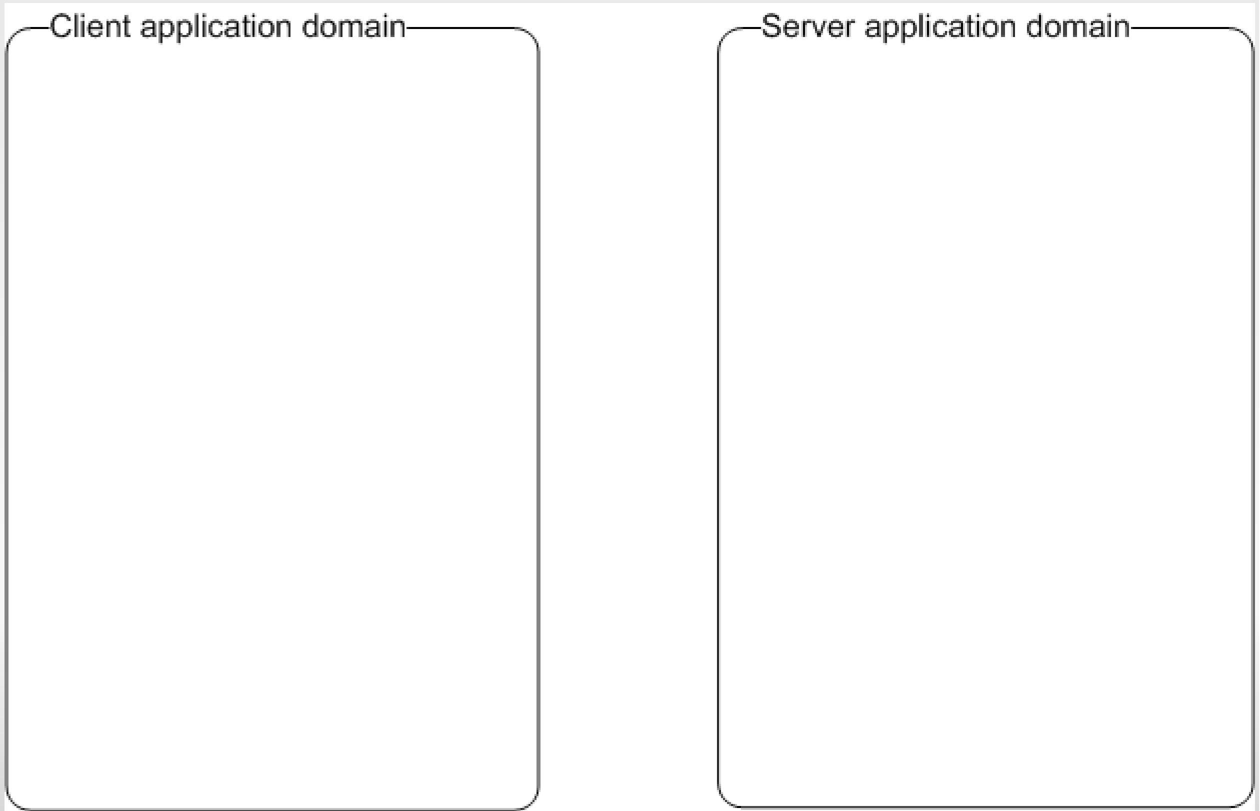


.NET Remoting

- What is .NET Remoting?
  - A framework for building distributed applications and systems.
  - It supports collaboration among objects in different application domains.
  - It hides much of the complexity of calling methods on remote objects.

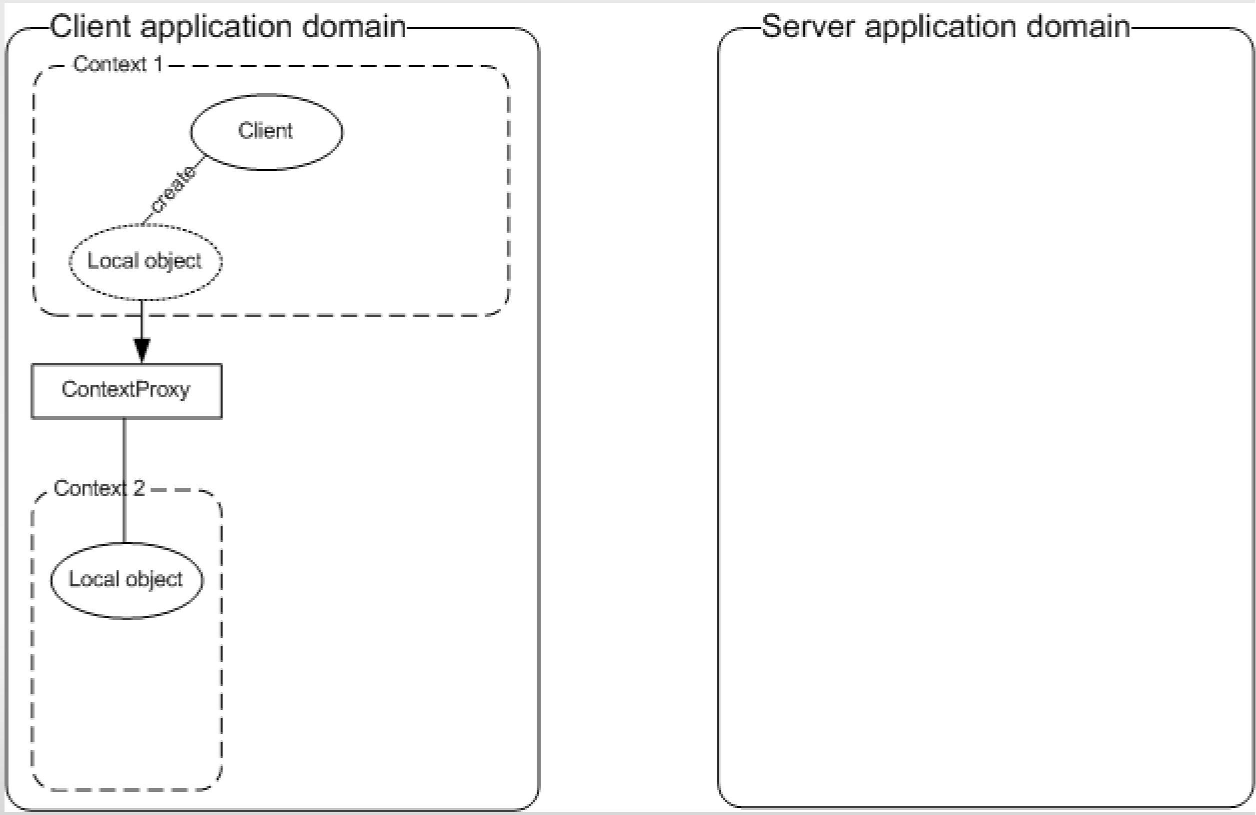
Application Domains

- The .NET boundary for Interprocess Communication.
- A more granular level of separation and better security than traditional processes.



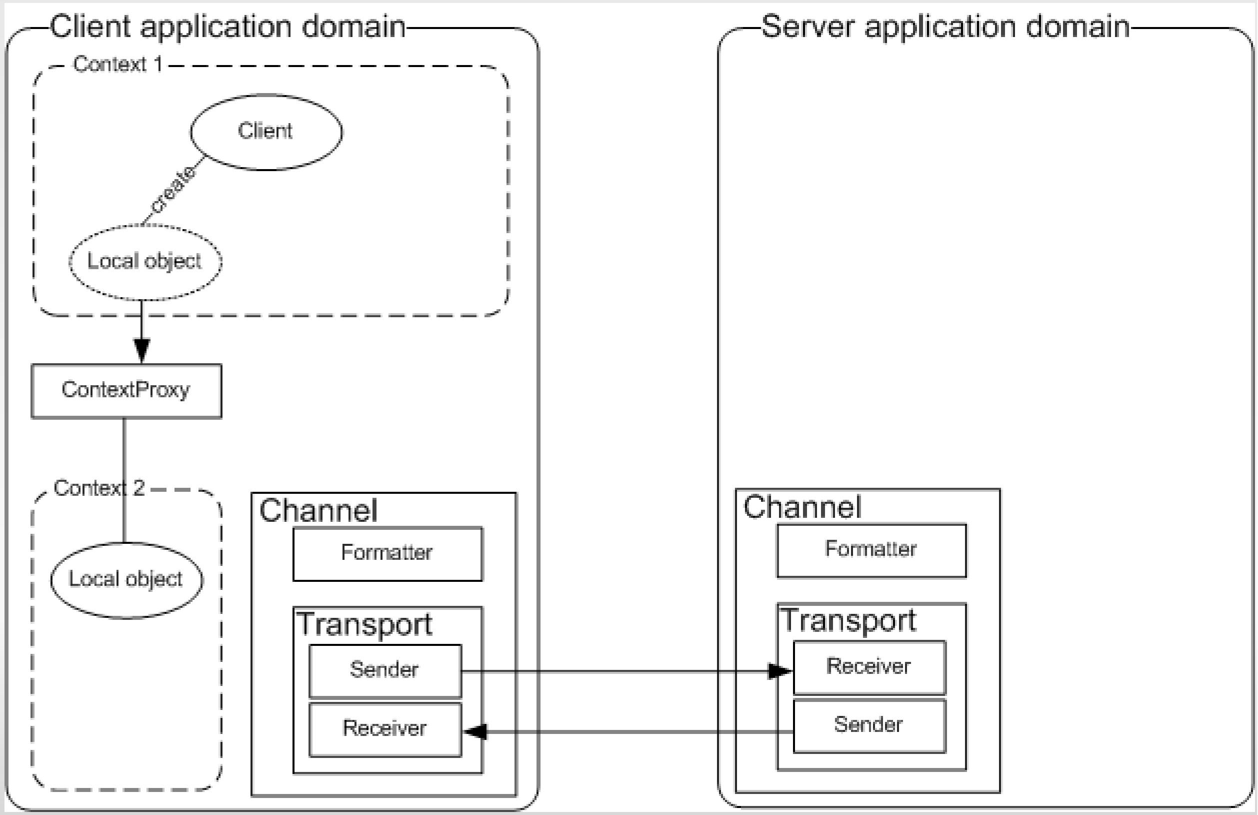
Contexts

- A boundary that contains objects with similar runtime properties.
- Several contexts may exist within one application domain, but then proxies do the marshaling locally.



Channels

- Transports messages between application domains.
- Registered by, and shared among objects.
- The formatter serializes the message.

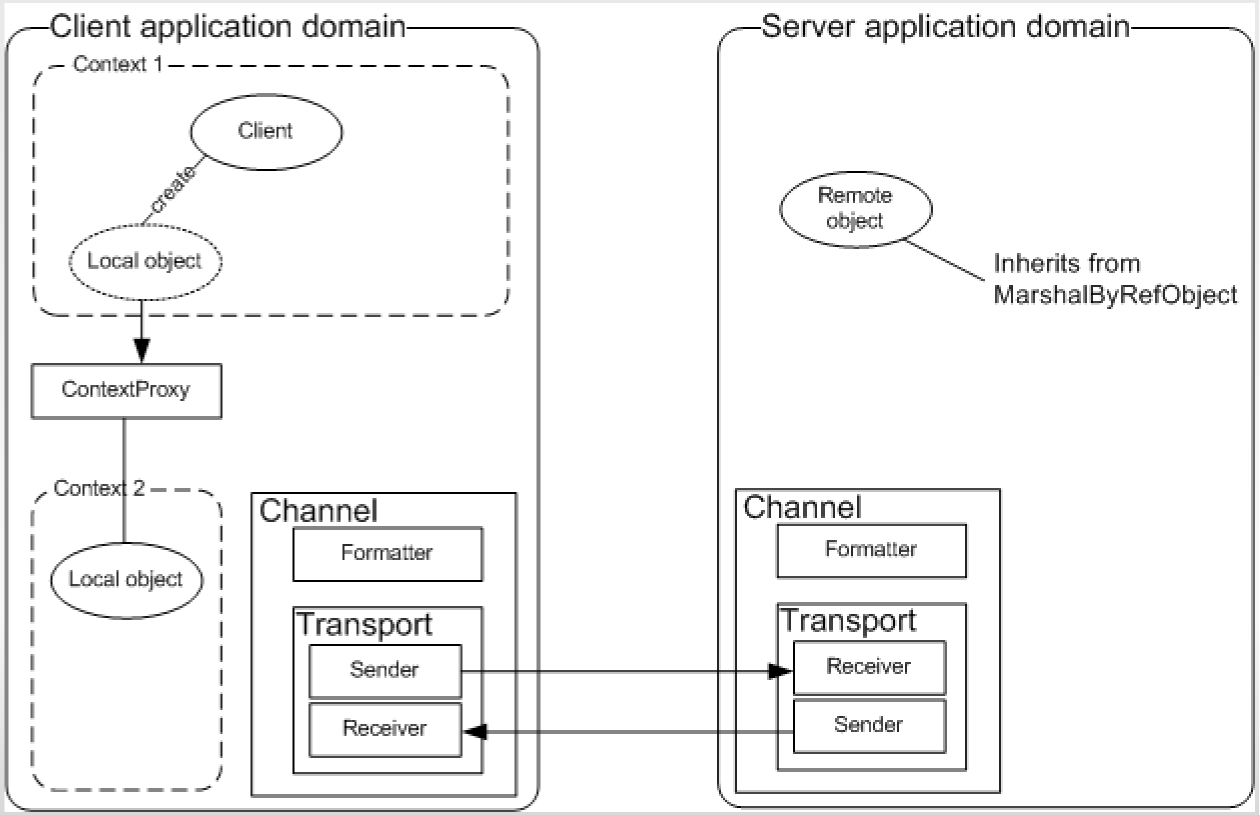


Remote Objects 1

- Any object outside the application domain should be considered remote, even if the objects are executing on the same machine.
- Marshaling (packaging):
  - Marshal by value - objects must be serializable.
  - Marshal by reference.
- The MarshalByRefObject class – base class for remote objects.
- The ObjRef class – A remote object reference.

Remote Objects 2

- Three types of remote objects:
- Single call
- Singleton
- Client-activated

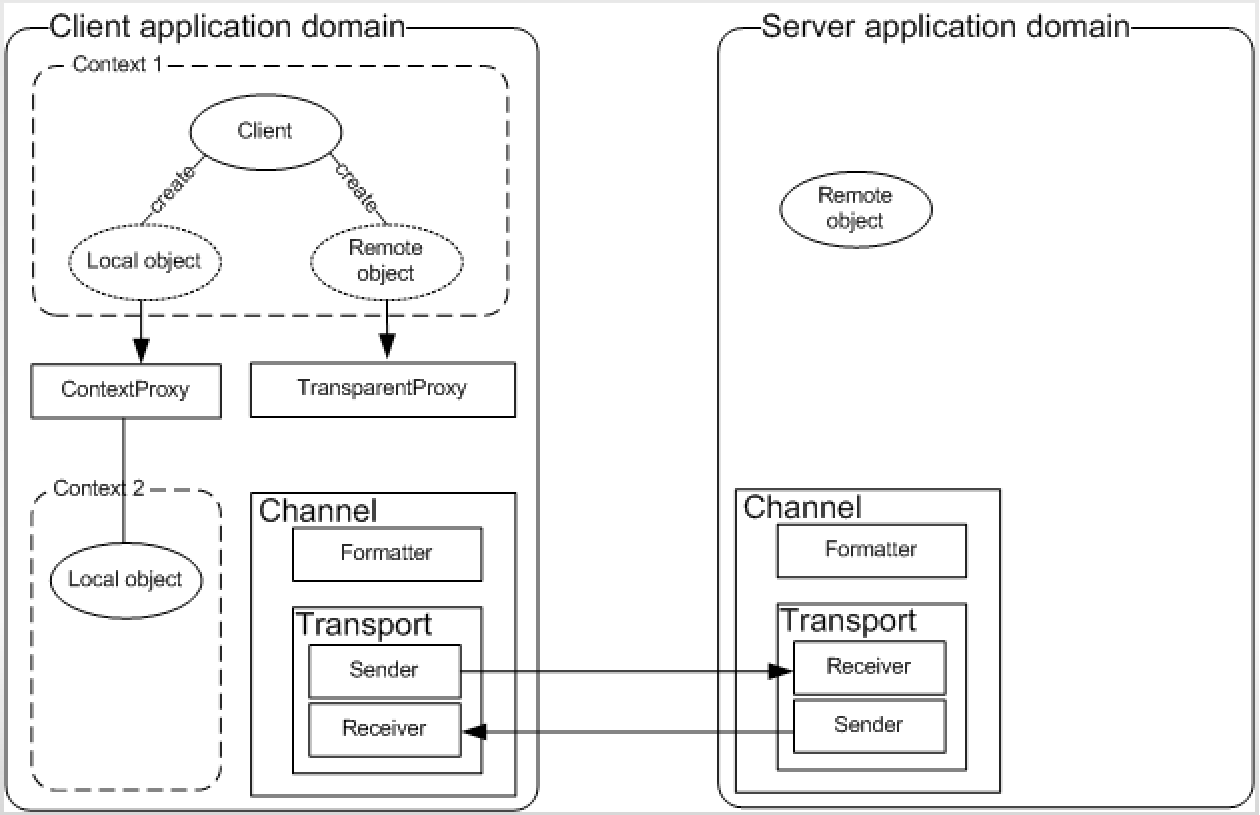


Object Activation

- Two kinds of object activation:
  - Server-activated objects.
  - Client-activated objects.

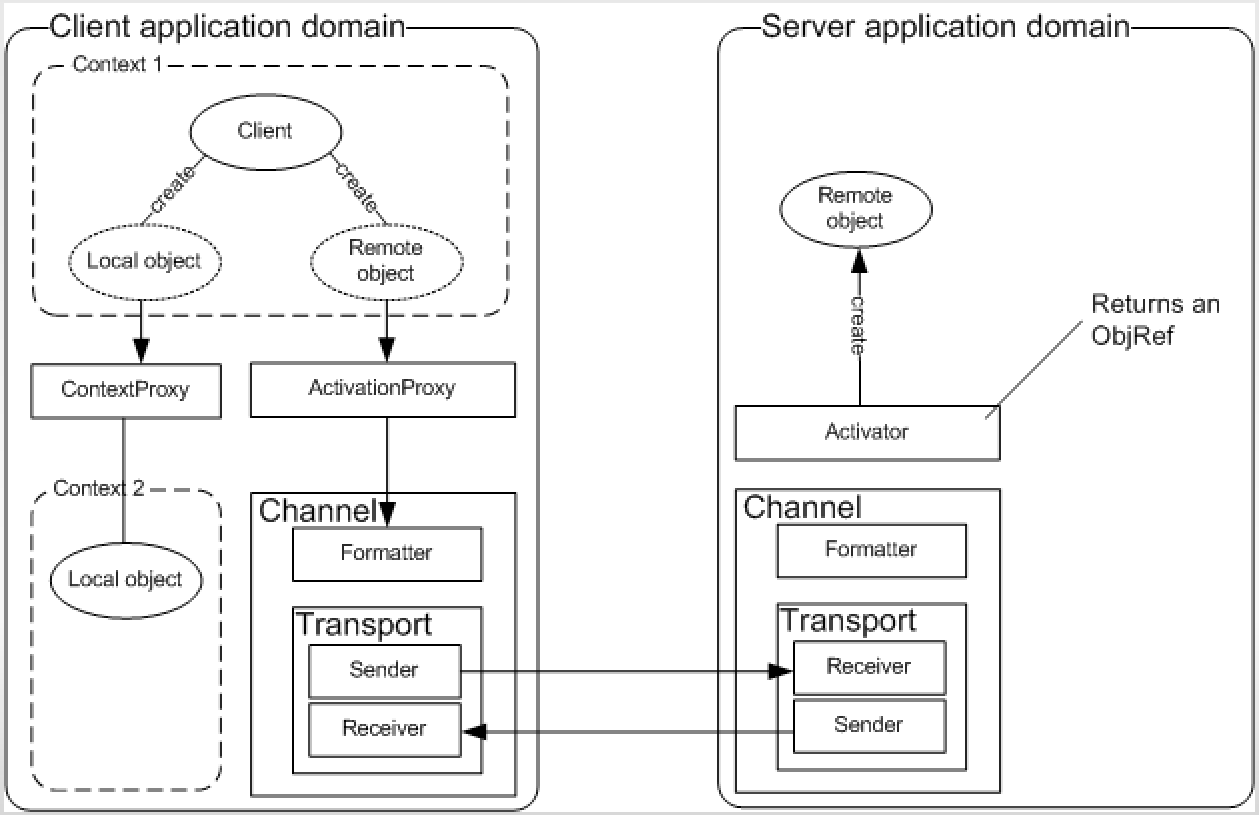
Activation – Server-activated

- No calls are made to the remote object when the client activates a new object. TransparentProxy is created.
- The remote object is activated when the first method call is performed.
- Applies to:
- Single call
- Singleton



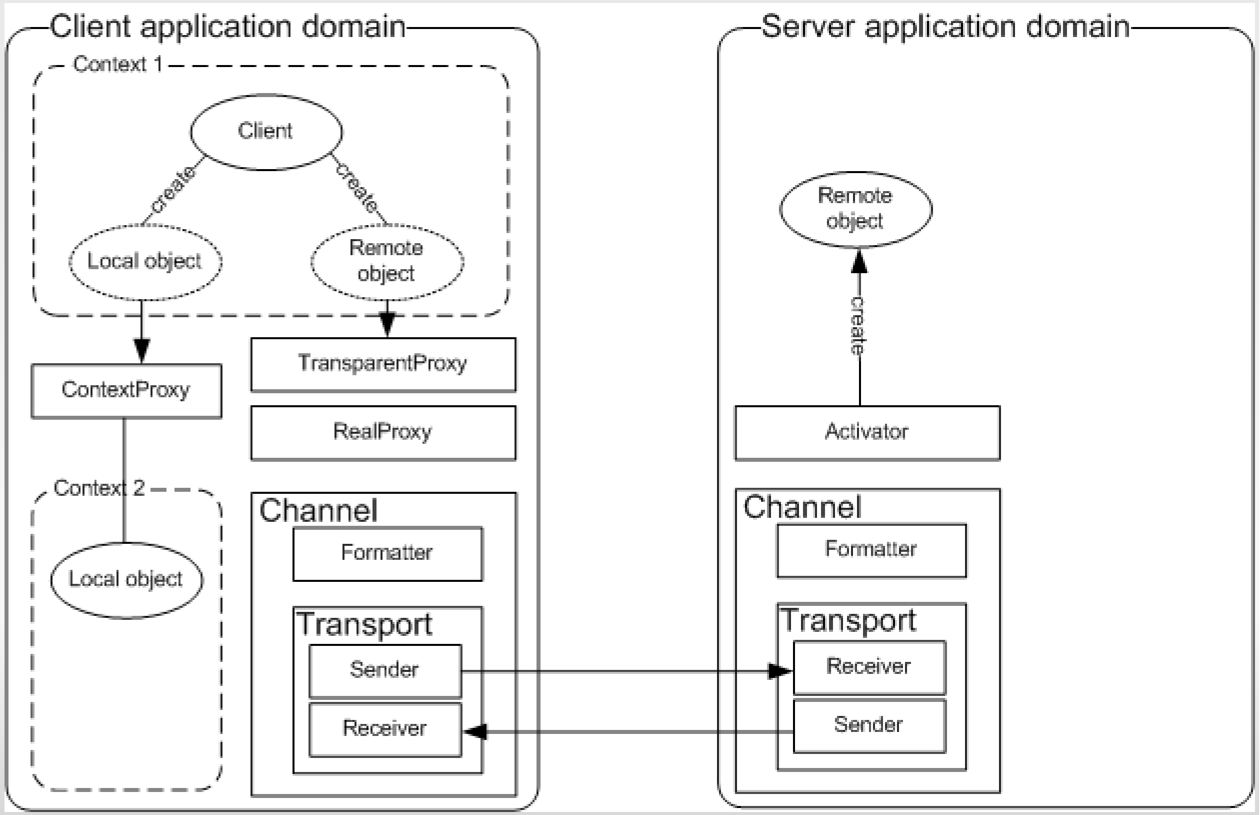
Activation – Client-activated 1

- Remote object activated upon creation.
- Remote call is made through an ActivationProxy.
- Supports constructor parameters.
- Applies to:
- Client-activated.



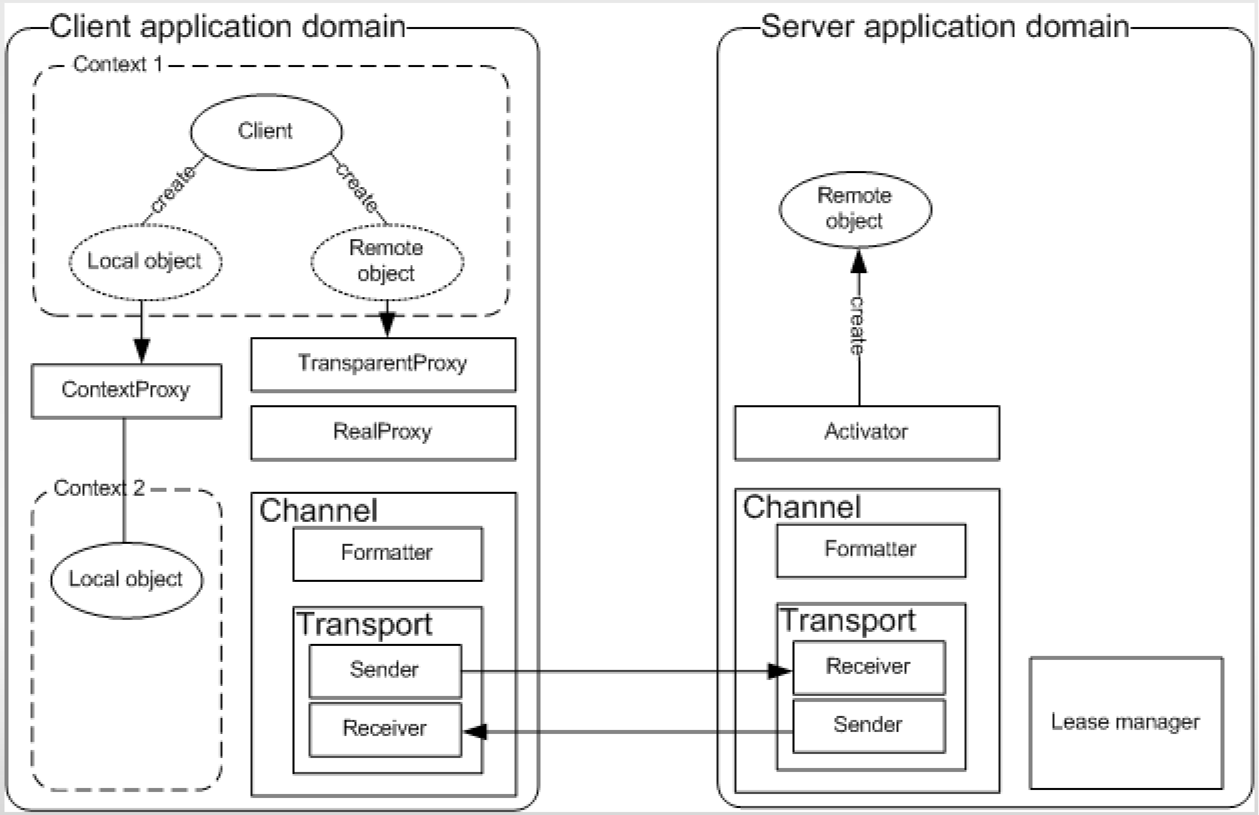
Activation – Client-activated 2

- When unmarshaling the ObjRef returned, the RealProxy and TransparentProxy objects are created.



Lifetime management

- Remote object lifetime management is based on a lease system.
- Every application domain have a lease manager.



Questions?