What is Middleware?

- Infrastructure that supports (distributed) component-based application development
  - a.k.a. distributed component platforms
  - mechanisms to enable component communication
  - mechanisms to hide distribution information
  - (large) set of predefined components

- Standard for constructing and interconnecting components
  - interchange
  - upgrade
  - adaptation
  - aggregation

Middleware Requirements

- ## Network communication
  - □ marshalling/unmarshalling

- ## Coordination
  - □ activation/termination, threading, group requests, synchronicity

- ## Reliability
  - □ delivery guarantees, total/partial ordering, atomicity, replication

- ## Scalability
  - □ transparency of access/location/migration/replication, load balancing

- ## Heterogeneity
  - □ platform, operating system, network OS, programming language

Middleware Categories

- # Transactional
  - ☐ two-phase commit for distributed transactions
  - ☐ e.g., IBM's CICS

- # Message-oriented (MOM)
  - ☐ communication via message exchange
  - ☐ e.g., Sun's JMS

- # Procedural
  - ☐ remote procedure calls as the foundation
  - ☐ e.g., DCE RPC

- # Object-based
  - ☐ communication among and via distributed objects
  - ☐ e.g., CORBA, COM

- # Component-based
  - ☐ support for distributed components
  - ☐ e.g., EJB

Elements of Middleware

- ## Software components
    - component interfaces
        - properties
        - methods
        - events

- ## Containers
    - shared context of interaction with other components
    - provide access to system-level services

- ## Metadata
    - self-descriptive information used by a component to *flexibly* communicate with others

- ## Integrated development environment
    - e.g., VisualCafe for Java

Distribution Support in Middleware

- Hidden from application programmers
- Five distributed services required
  - remote communication protocols
    - e.g., RPC, message passing
  - directory services
    - for accessing shared, globally-available services
  - security services
    - protection of shared resources via authentication
  - transaction services
    - for concurrent data access/update
  - system management services
    - service monitoring, management, and administration

CORBA

- A middleware platform that supports a standardized OO architecture for software applications
  - □ *C*ommon *O*bject *R*equest *B*roker *A*rchitecture
  - □ Open standard - developed by the *O*bject *M*anagement *G*roup
  - □ CORBA is a component of OMG's *O*bject *M*anagement *A*rchitecture

- CORBA supports distributed object computing

- CORBA uses a *broker*
  - □ an intermediary handling requests in a system
  - □ facilitates communication between clients and server objects
  - □ separates a component's interface from its implementation

CORBA's Enhancements to Client/Server

- Extends distributed computing paradigms, such as DCE RPC, to distributed object computing

- Mutable client-server relationships

  - clients and servers not hard-wired to one another
  - dynamic discovery of component interfaces

- Interaction intermediary

  - ORB
  - object adapters
  - gateways for other object systems

- Both synchronous and deferred synchronous communication

  - deferred synchronous $\approx$ asynchronous

Main CORBA Features

- Object request broker (ORB)
- OMG interface definition language (IDL)
- Language mappings
- Stubs and skeletons
- Interface repository
- Dynamic invocation and dispatch
- Object adapters
- Inter-ORB protocols

CORBA Architecture
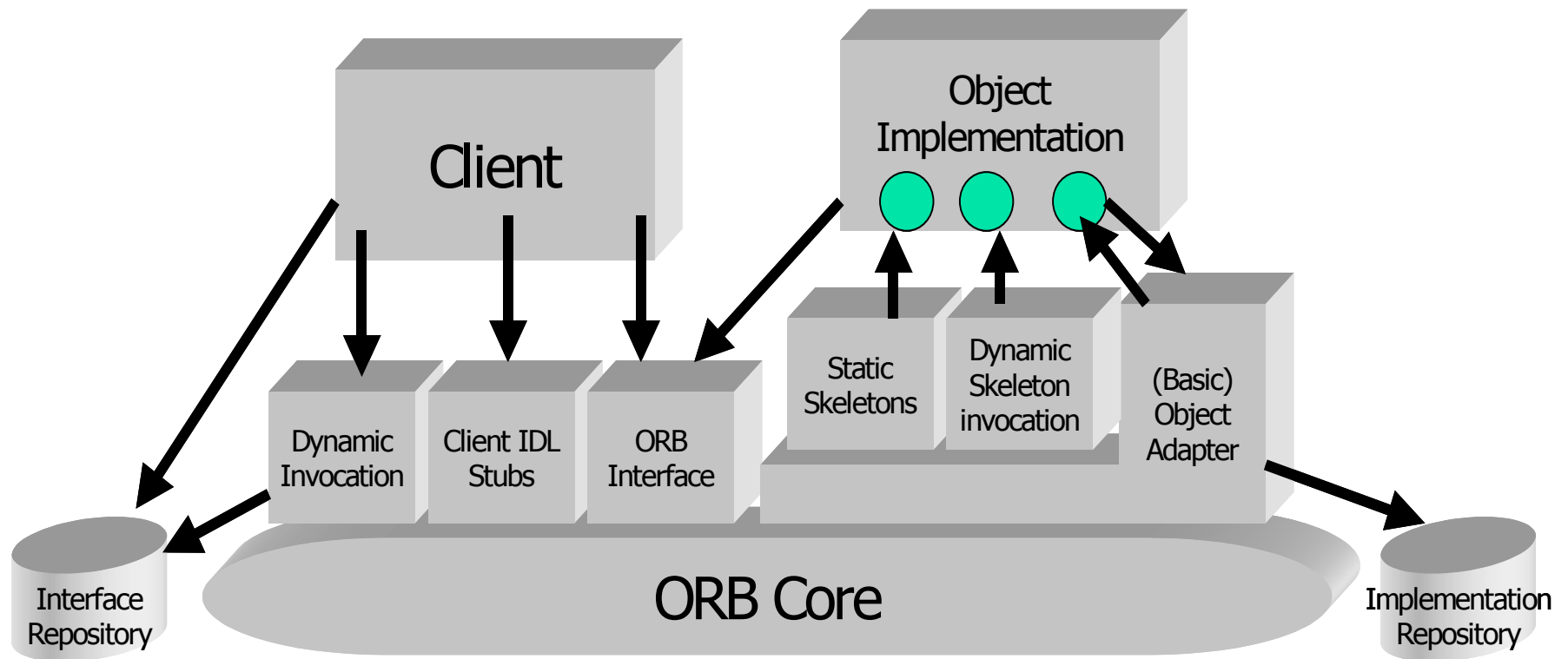
Object Request Broker

- Delivers client requests and server responses
  - □ provides a layer of indirection between clients and servers

- The ORB hides object
  - □ location
  - □ implementation
  - □ execution state
  - □ communication mechanisms

- Requests on an object are made using its reference

- Clients can obtain references in three ways
  - □ create an object to get its reference
    - □ uses *factory objects*
  - □ invoke a lookup service (naming, trading)
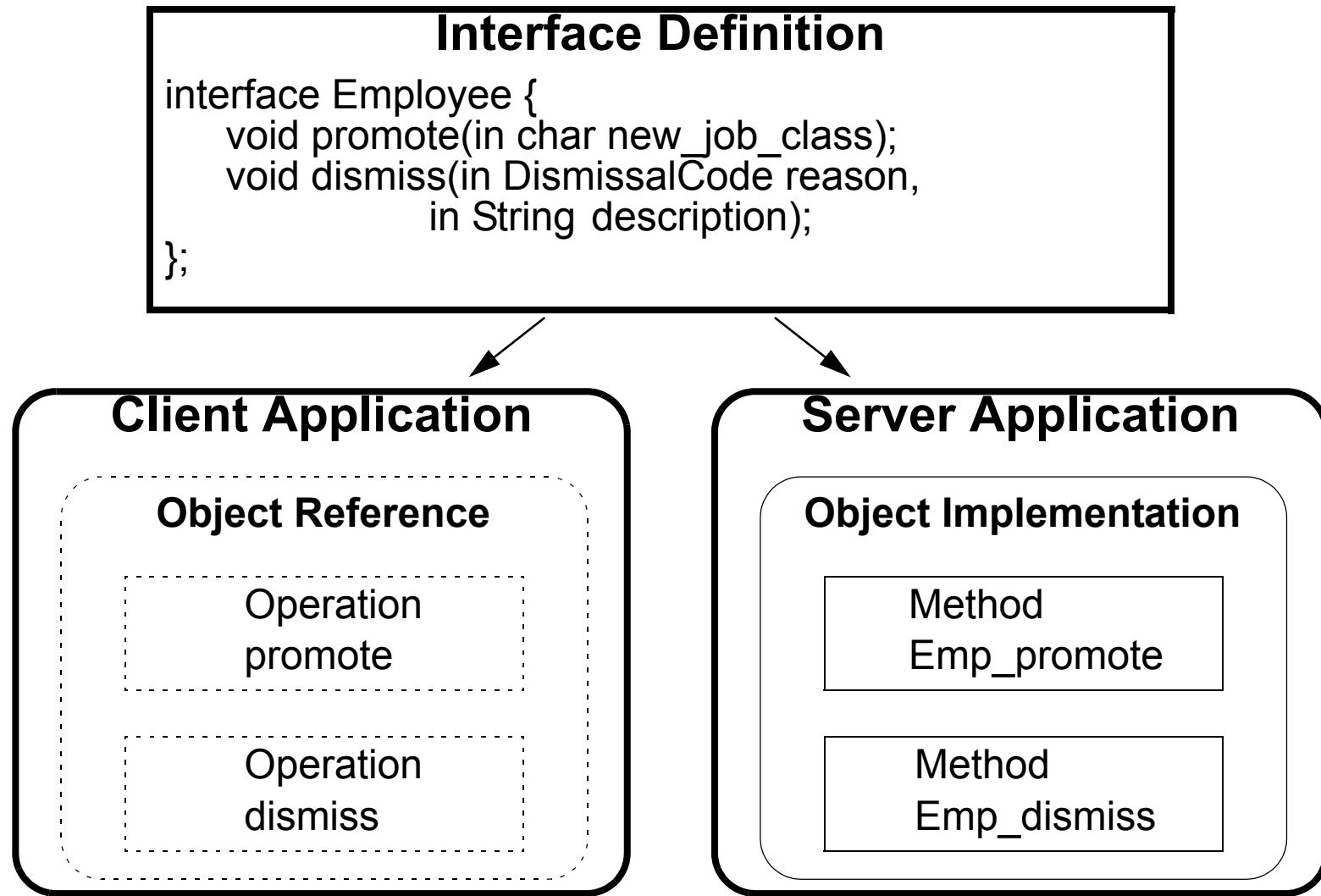 - □ use persistent references to objects

OMG IDL

- Specifies (implementation-independent) object interface

- Basic types
  - short, long, long long, float, double, long double, char, wchar, boolean, octet, enum, string, wstring
  - *Any*

- Constructed types
  - struct, union, array, sequence

- IDL is language-independent
  - Standardized language mappings for C, C++, Ada95, COBOL, Smalltalk, Java, ...
  - Marshalling
  - Unmarshalling

Stubs and Skeletons

- Used in CORBA's static invocation

- Programming language-specific counterparts to IDL definitions
    - Stubs and skeletons are generated using the IDL definitions

- *Stubs* create and issue requests on the client side
    - a.k.a. surrogates or proxies
    - perform marshalling of requests

- *Skeletons* receive and forward requests to objects on the server side
    - perform unmarshalling of requests
    - return results via the server and client ORBs to the stub

Example of Stubs and Skeletons

Interface Definition

```
interface Employee {
      void promote(in char new_job_class);
      void dismiss(in DismissalCode reason,
                   in String description);
};
```

Client Application

Object Reference

Operation
promote

Operation
dismiss

Server Application

Object Implementation

Method
Emp_promote

Method
Emp_dismiss

Static Method Invocation

- Define object classes using IDL

- Run IDL file through language precompiler

- Add implementation code to skeletons

- Compile code

- Bind class definitions to Interface Repository

- Register the run-time objects with Implementation Repository

- Instantiate the objects on the server

Interface Repository

- Used for performing operations on objects whose interface is not known at compile time

- Knowing interfaces of all objects *a priori* may be impractical
  - independently developed components
  - fast changing parts of the system
  - dynamic manipulation

- IR allows access to the IDL type system at runtime
  - IR is a CORBA object whose functionality is invoked like any other object
  - allows CORBA's dynamic invocation

- Not to confuse with Implementation Repository that contains information for locating and activating objects

Dynamic Method Invocation

- Obtain interface name from Interface Repository

- Obtain method description from Interface Repository

- Create argument list

- Create request

- Invoke request

- Dynamic vs. static invocation

  - harder to program

  - less robust type checking

  - slower (factor of 40)

  - harder to understand/document

CORBA Services

- Naming

- Life Cycle
  - □ create, copy, move, delete objects

- Events
  - □ register for interest in specific events (e.g., push, pull)

- Object Trader
  - □ yellow pages for objects based on services they provide

- Transactions
  - □ flat, nested
  - □ involving heterogeneous ORBs and non-ORBs

- Concurrency Control
  - □ coordinate access to shared resources using locks

CORBA Services (cont.)

- Object Security
  - □ authentication, audits, encryption
- Persistence
- Query
  - □ find objects of matching attributes
- Collections
  - □ manipulate objects in a group
- Relationships
  - □ dynamically create and keep track of relationships
- Time
- Licensing
  - □ license manager
- Properties

Inter-ORB Protocols

- CORBA standard does not cover all of CORBA
  - □ e.g., different protocols and object references are possible
- General ORB interoperability architecture
  - □ based on the General Inter-ORB Protocol (GIOP)
  - □ one instance of GIOP is the IIOP (built on top of TCP/IP)
  - □ Environment-Specific Inter-ORB Protocols (ESIOP)
    - □ allow CORBA and non-CORBA components to interact
  - □ one instance is the Distributed Computing Environment Common Inter-ORB Protocol (DCE-CIOP)
- Standard object reference format
  - □ Interoperable Object Reference (IOR)
- Portable Object Adapters (POA)

COM/DCOM

- Microsoft's middleware infrastructure in many ways similar to CORBA

- Defines a binary standard for component interoperability

  □ programming language independence

- Platform independent

  □ Windows (95, 98, NT)

  □ Mac

  □ Unix

- Distribution transparency

  □ does exploit operational characteristics

- Dynamic component loading and unloading

Basic COM Features

- Binary standard for component interactions
- Support for interfaces
  - defined in an IDL different from CORBA's
- Base interface with introspection support
  - dynamic discovery of other components' interfaces
  - garbage collection via reference counting
- Unique component ID mechanism
- Communication is synchronous by default
  - asynchronous communication supportable by callbacks and connection points

Binary Standard

- Component operation invocation via virtual tables — *vtables*
  - works for languages that support function pointers
    - e.g., C, C++, Smalltalk
  - the client contains a pointer to the vtable
  - the vtable contains a pointer to the server function
  - one layer of indirection over standard function calls
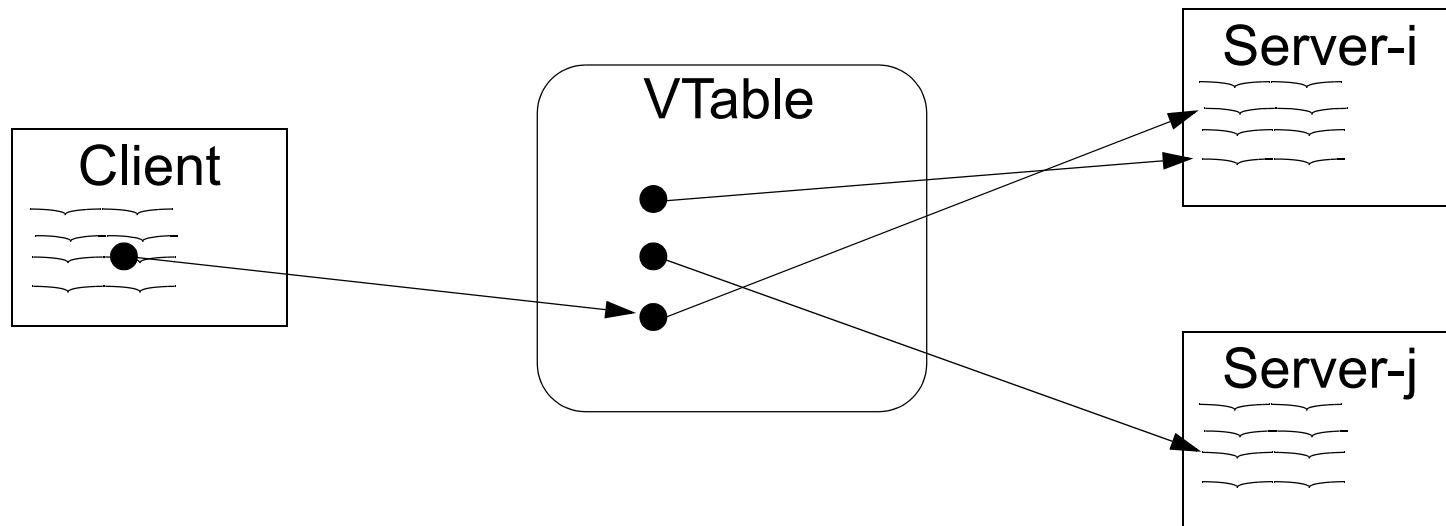
COM Components

- Compiled code that provides some service to a system
- A component may support a number of interfaces
  - an interface is a collection of semantically related functions
  - COM interfaces begin with "I" by convention
- All access to component services is via interface pointers
  - allows service reimplementation
- Each component supports base interface *IUnknown*
- Transparent remote access via proxy-stub pairs
  - similar to CORBA
- Every component has a globally unique identifier (GUID)
  - 128 bit integer
  - used to refer to component's TypeLibrary (metadata repository)

Notes on COM Interfaces

- Interface $\neq$ class
  - □ they contain no implementation
  - □ multiple implementations of an interface possible

- Interface $\neq$ component
  - □ interfaces are the (binary) component interaction standard

- Heterogeneous COM clients and servers interact via interface pointers

- A single COM component can implement multiple interfaces

- Interfaces are strongly typed
  - □ every interface has a GUID

- Interfaces are immutable
  - □ e.g., no subtyping, subclassing, inheritance
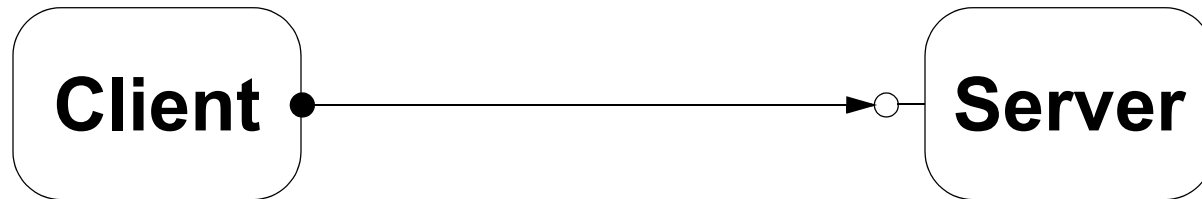
Interface *IUnknown*

- Must be implemented by all COM components
- Has three methods
  - QueryInterface
    - provides introspection capabilities
    - allows runtime discovery of component interface
    - delivers interface pointer to a client
  - AddRef
    - called when another component is using the interface
    - increments reference count
  - Release
    - called when another component stops using the interface
    - decrements reference count
- Supports garbage collection
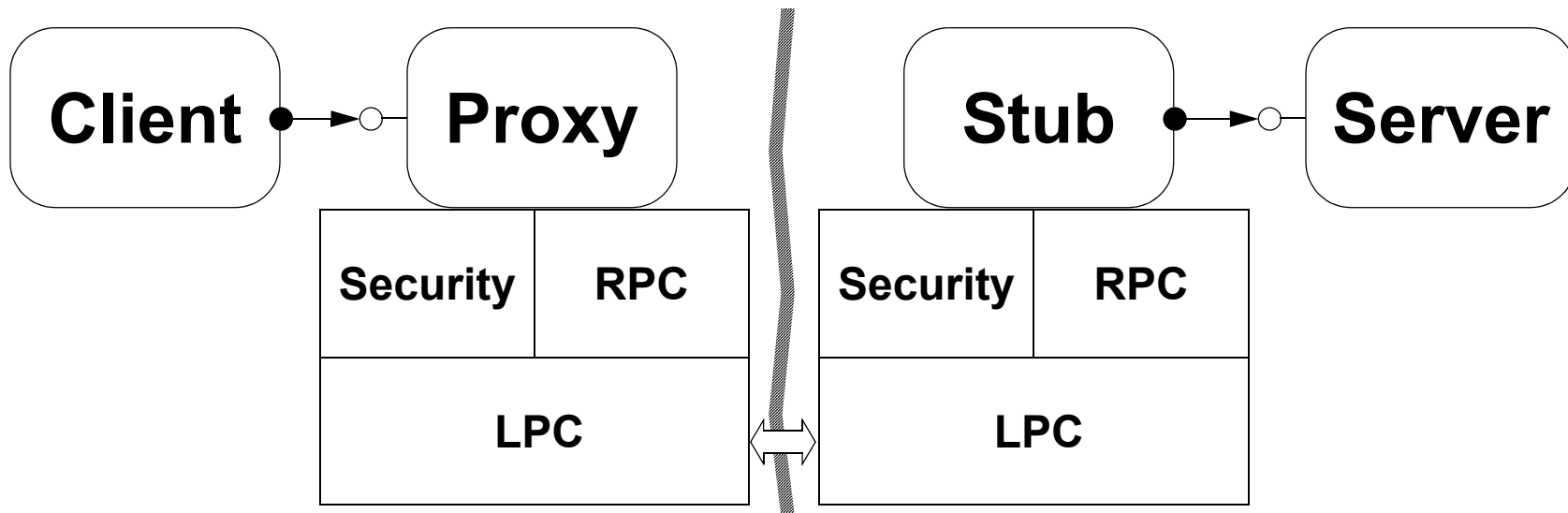 - a component can be unloaded when its reference count is 0

Distributed COM

- DCOM = COM binary standard
    +
    runtime infrastructure for communicating across distributed address spaces
    - initially only on Windows
    - recently adding Mac and Unix
- Uses OSF's DCE RPC as a basis for remote interaction
    - proxy/stub mechanism
- Attempts to address challenges of distributed computing
    - interacting components should be "close" to one another
    - some components' locations are fixed
    - inverse relationship between component size and flexibility
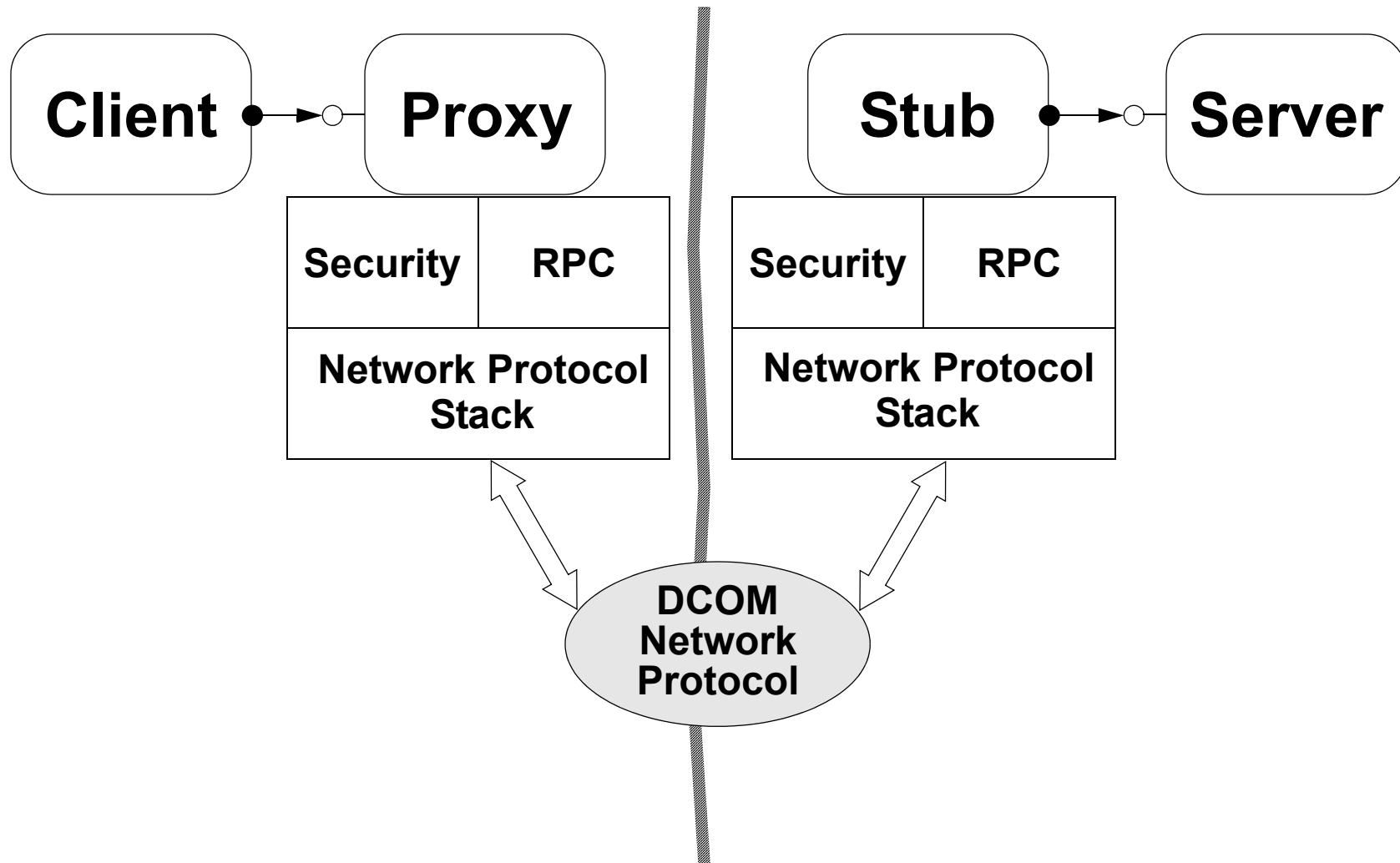   - direct relationship between component size and network traffic

Distribution in COM/DCOM — In-Process

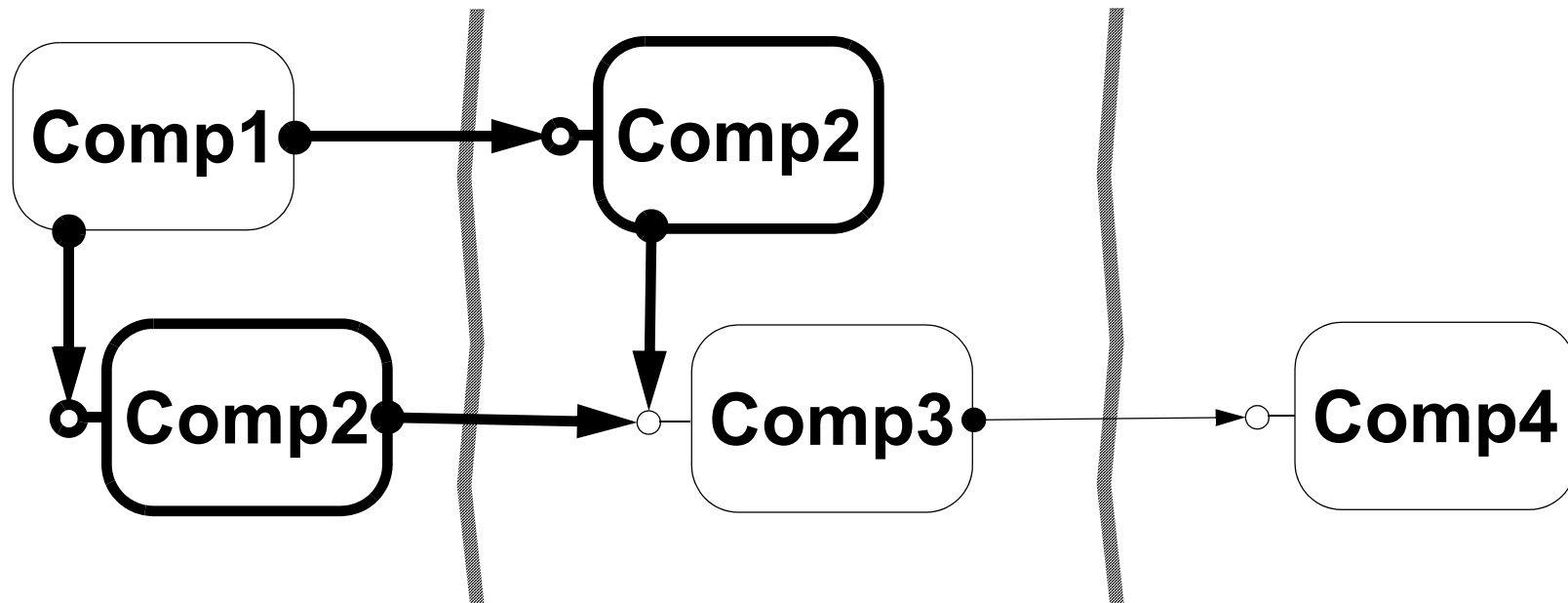Client → Server

Distribution in COM/DCOM — Inter-Process

Client → Proxy ⟷ Stub → Server

Security	RPC
LPC	

Security	RPC
LPC	

Distribution in COM/DCOM — Cross-Network

Client ● → ○ Proxy

Security	RPC
Network Protocol Stack	

Stub ● → ○ Server

Security	RPC
Network Protocol Stack	

DCOM Network Protocol

Distribution in DCOM

- Full distribution transparency
  - component location is hidden from clients (and client developers)
  - method invocation is identical
  - underlying communication mechanisms change

Garbage Collection in COM/DCOM

- ## Networks are fragile
  - □ connections may break for many reasons
  - □ if a connection to a client is broken, a server component should not needlessly consume resources

- ## COM/DCOM uses a pinging protocol to detect (in)active clients
  - □ a ping message is sent every two minutes from client to server
  - □ distributed garbage collection
  - □ transparent to the application (developer)
  - □ reference count is decremented if multiple (>3) ping periods pass without receiving a message

- ## The protocol is efficient
  - □ ping messages are piggybacked onto existing COM calls