Chapter 4

In-depth study of Middleware Architectures

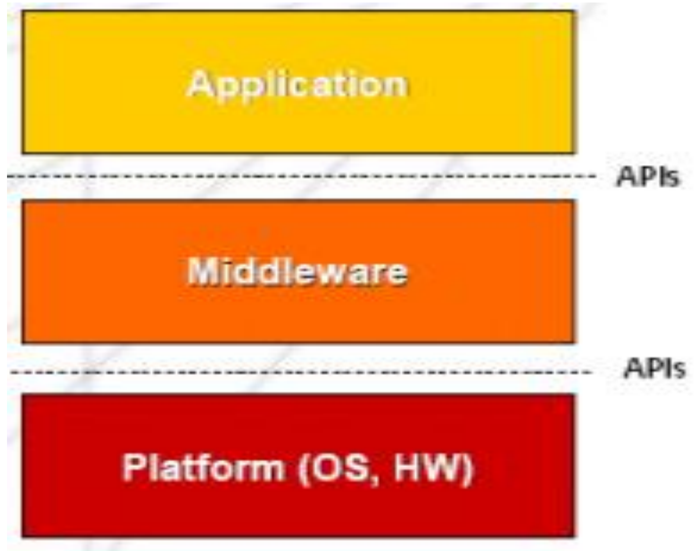
E.g. COM/DCOM, CORBA, .NET

Outline

- What is Middleware
- Usage/Advantages
- Architectural Significance
- Middleware Types
- Advantages and Disadvantages of Middleware

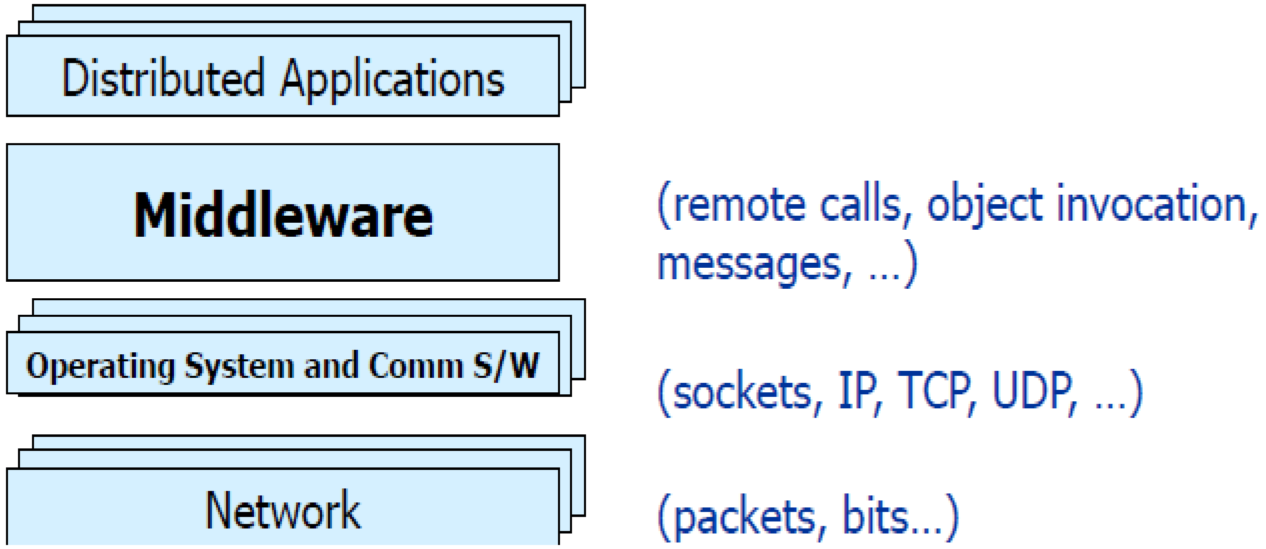
Middleware - Definition

- **Middleware** is **connectivity software** that consists **of a set of enabling services** that **allow multiple processes running on one or more machines to interact with each other**.
- **Middleware** is the "**glue**" that **connects diverse computer systems**. Typically, legacy systems store information in **proprietary formats**, use proprietary protocols to communicate, and may even be running on hardware that's no longer manufactured or supported.

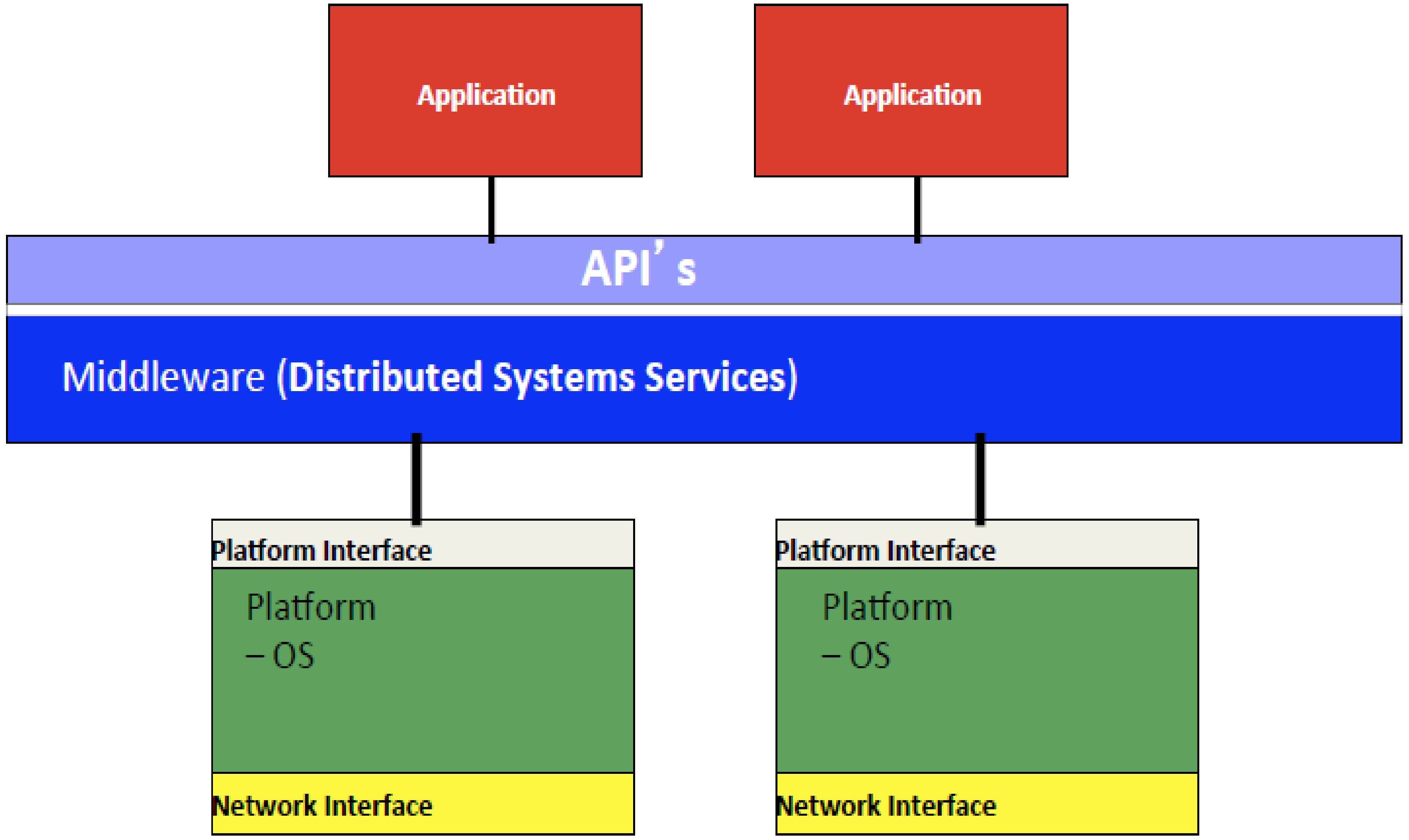


Middleware

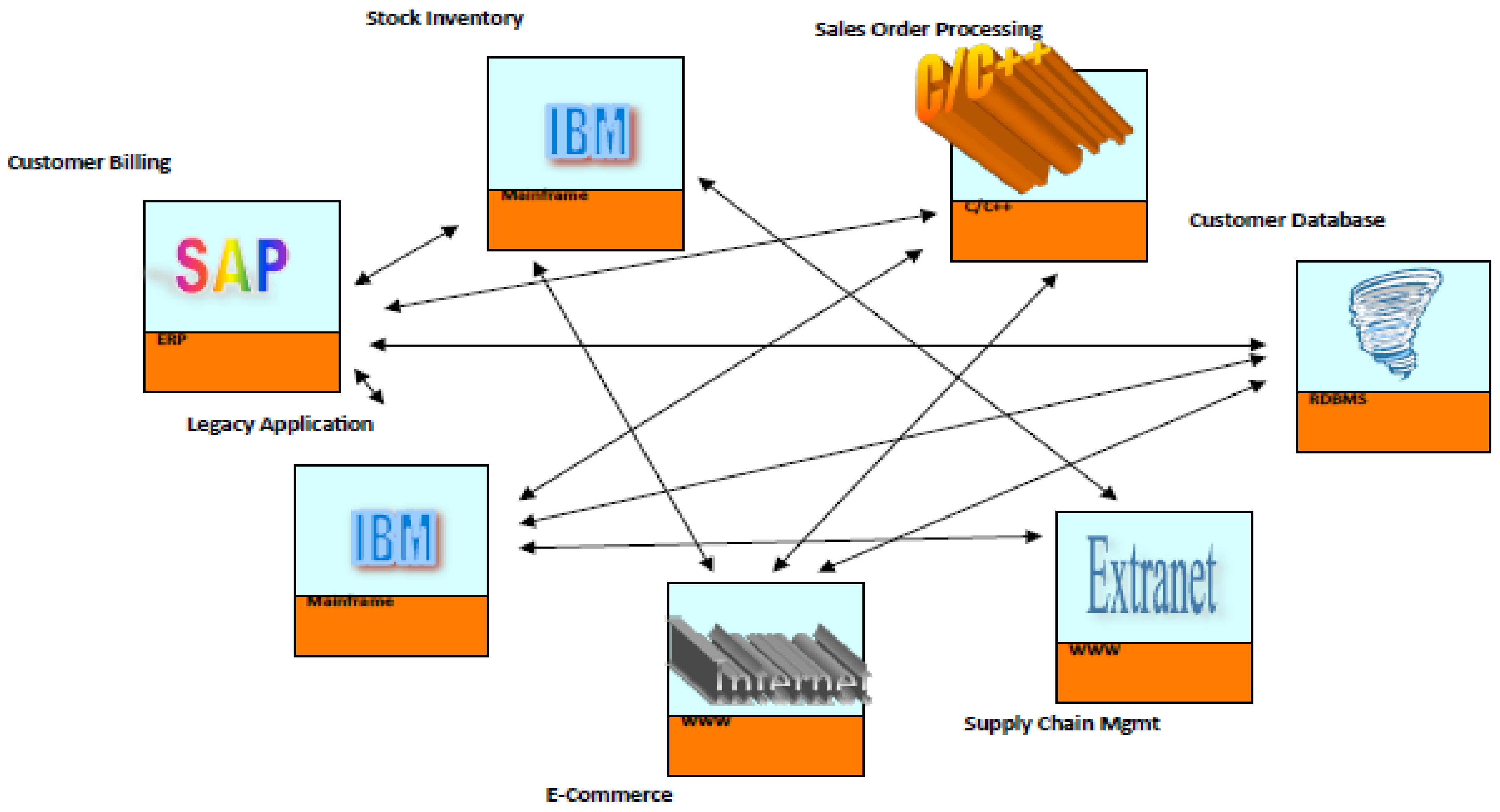
- What is Middleware?
  - Layer between **OS** and **distributed applications**
  - Hides complexity and heterogeneity of distributed system
  - Bridges gap between low-level OS communications and programming language abstractions
  - Provides common programming abstraction and infrastructure for distributed applications


Distributed Applications
Middleware
(remote calls, object invocation, messages, ...)
Operating System and Comm S/W
(sockets, IP, TCP, UDP, ...)
Network
(packets, bits...)

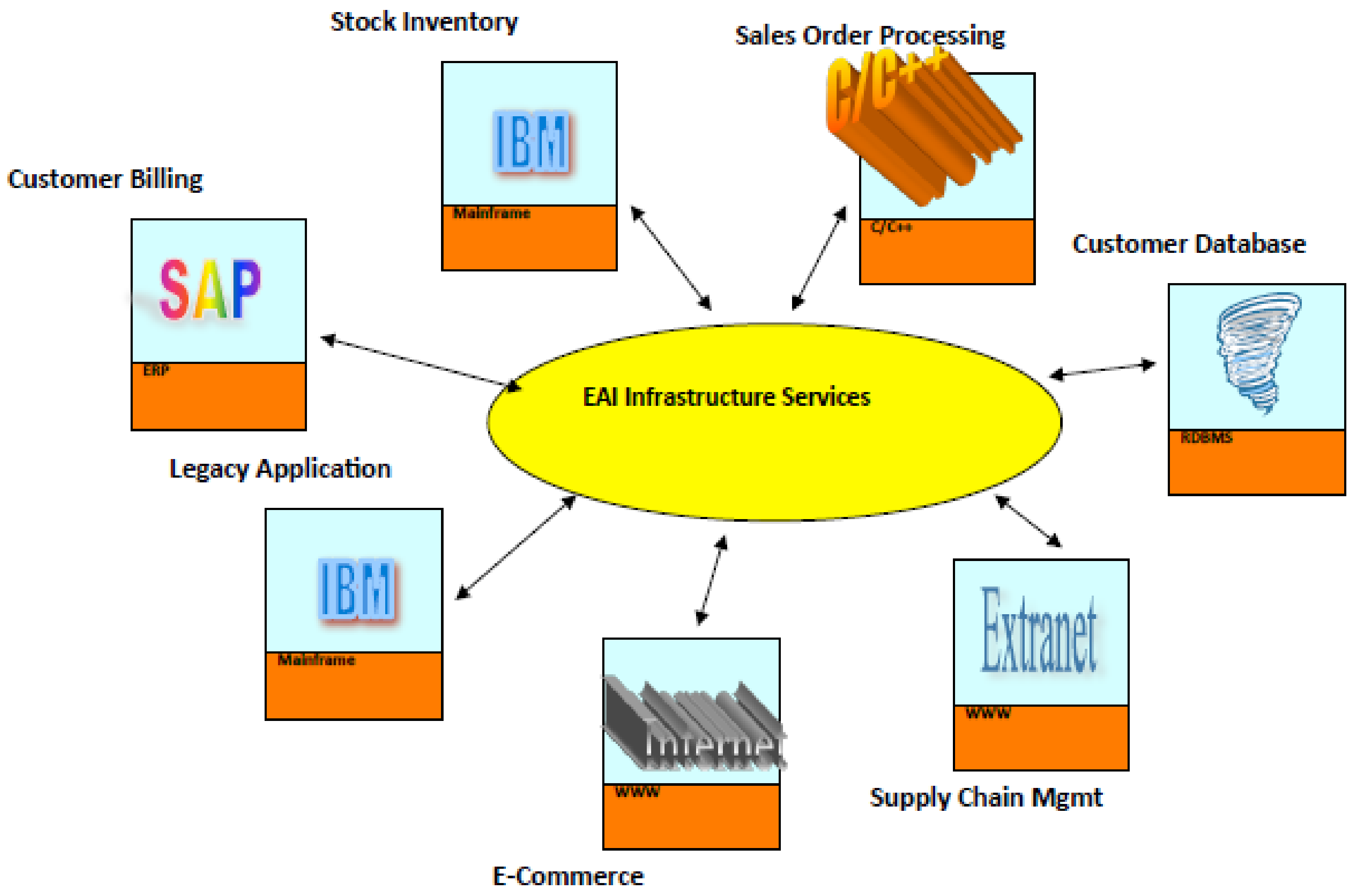
Middleware Architecture



Legacy enterprise situation



Middleware solution



Middleware - Usage

- Middleware services provide a more functional set of API than OS and network services to allow an application to:
  - Locate transparently across the network, providing interaction with other application or service
  - Be independent from network services.
  - Be reliable and available.
  - Scale-up in capacity without losing functionality.
  - Real time information access among systems
  - Streamlines business processes and helps raise organizational efficiency.
  - Maintains information integrity across multiple systems .

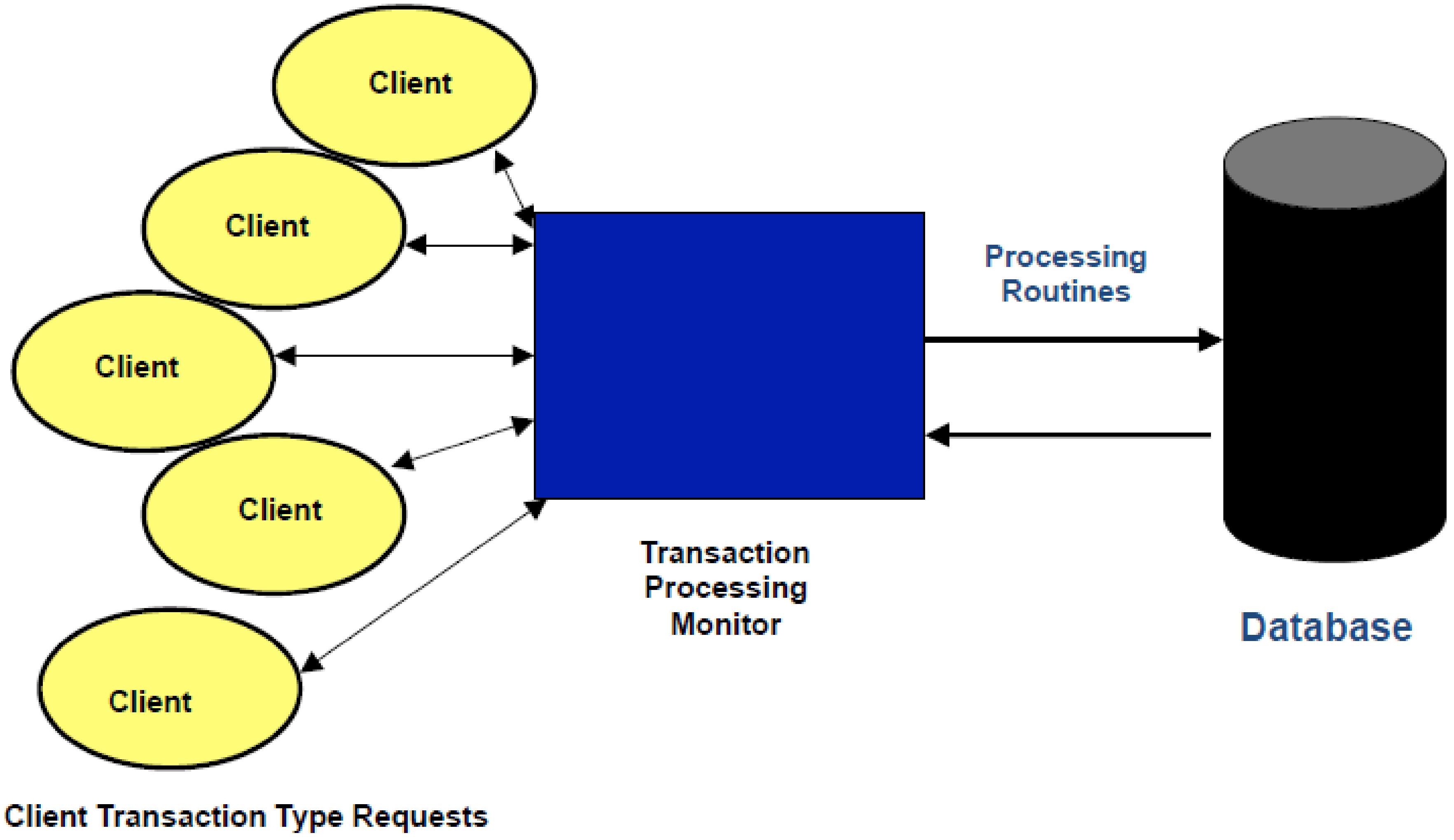
Disadvantages of Middleware

- Prohibitively high development costs.
- **EAI**(Enterprise Application Integration) implementations are very time consuming, and need a lot of resources.
- There are few people with experience in the market place.
- There exists relatively few satisfying standards.
- The tools are not good enough.
- Too many platforms to be covered.
- Middleware often threatens the real-time performance of a system.
- Middleware products are not very mature.

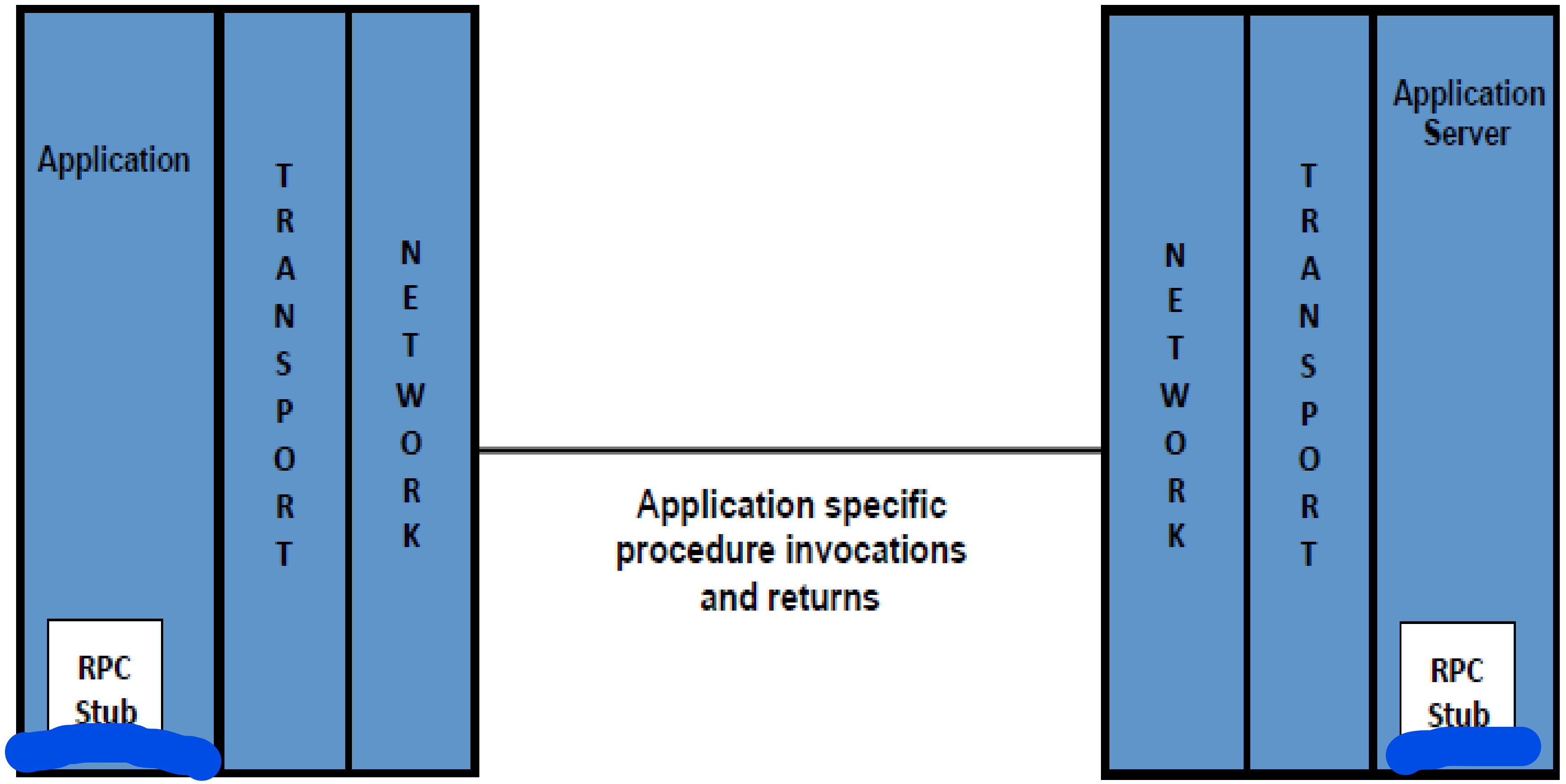
Middleware Evolutions

- Transaction Processing (TP) monitors
- Remote Procedure Calls (RPC)
- Message Oriented Middleware (MOM)
- Object Request Brokers (ORBs)
- Service oriented Architecture(SOA)

TP Monitors - Demonstration

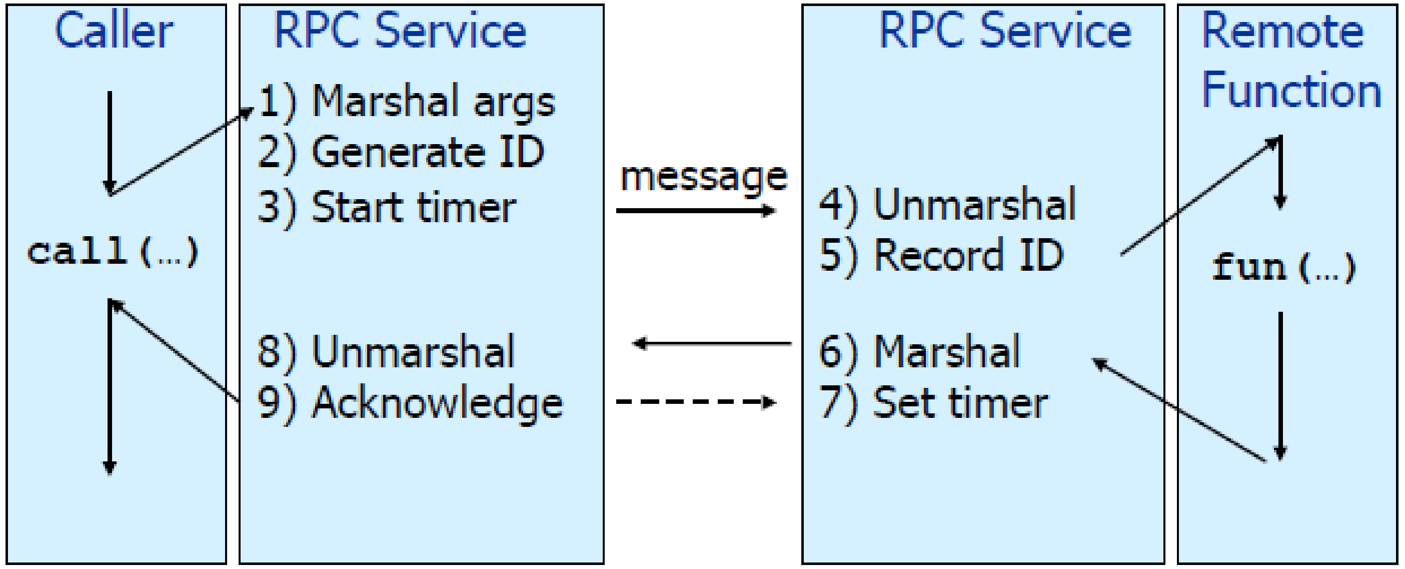

Client
Client
Client
Client
Client
Processing Routines
Transaction Processing Monitor
Database
Client Transaction Type Requests

Remote Procedure Call (RPC)



Remote Procedure Call (RPC)

- Masks remote function calls as being local
- Client/server model
- Request/reply paradigm usually implemented with message passing in RPC service
- Marshalling of function parameters and return value



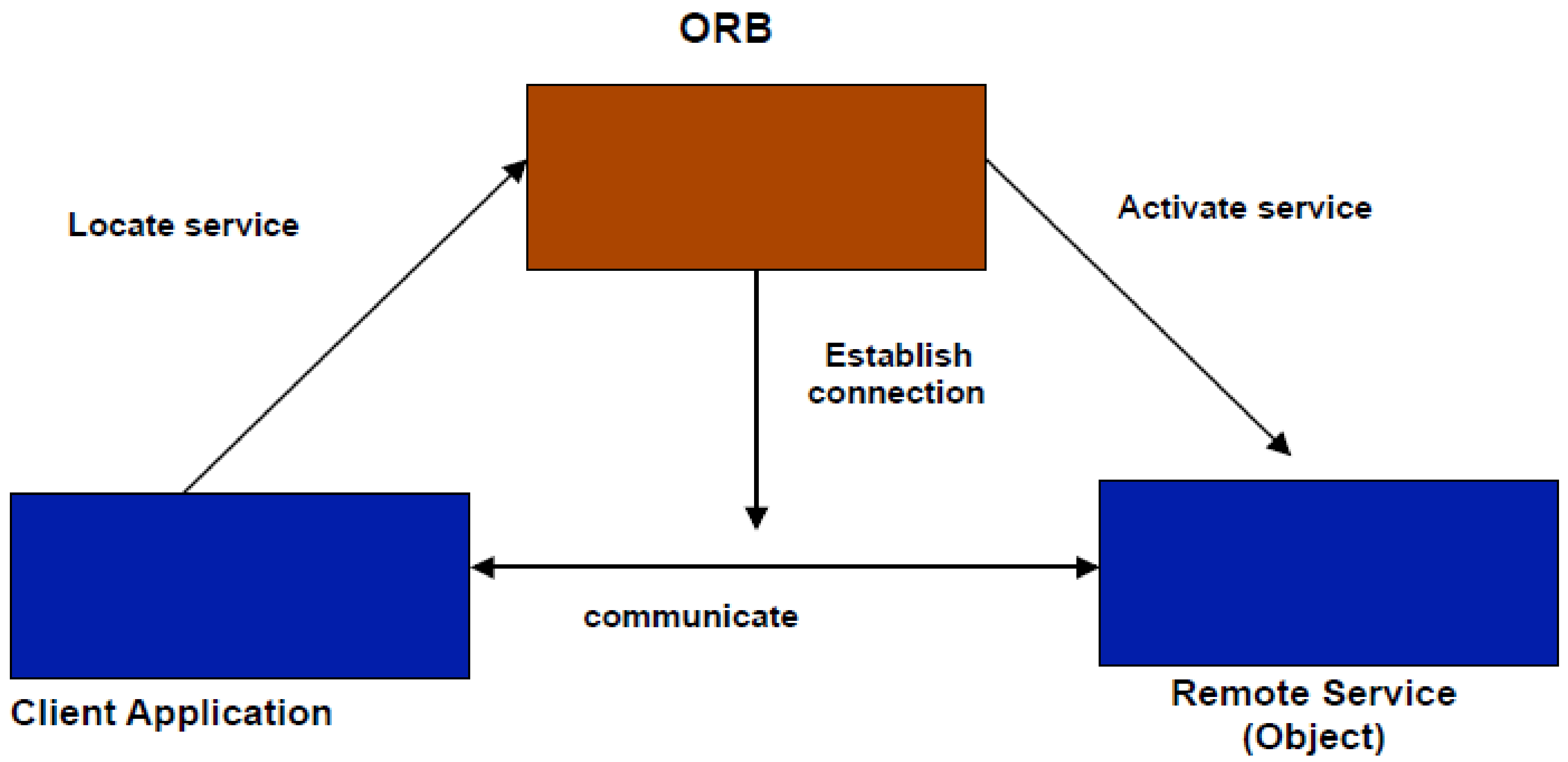
Properties of RPC

- Language-level pattern of **function call**
  - easy to understand for programmer
- Synchronous request/reply interaction
  - natural from a programming language point-of-view
  - matches replies to requests
  - built in synchronization of requests and replies
- Distribution transparency (in the no-failure case)
  - hides the complexity of a distributed system
- Various reliability guarantees
  - deals with some distributed systems aspects of failure

Disadvantages of RPC

- Synchronous request/reply interaction
  - tight coupling between client and server
  - client may block for a long time if server loaded
- Leads to multi-threaded programming at client
  - slow/failed clients may delay servers when replying
  - multi-threading essential at servers
- Distribution Transparency problem
  - Not possible to mask all problems
- RPC paradigm is not object-oriented
  - invoke functions on servers as opposed to methods on objects

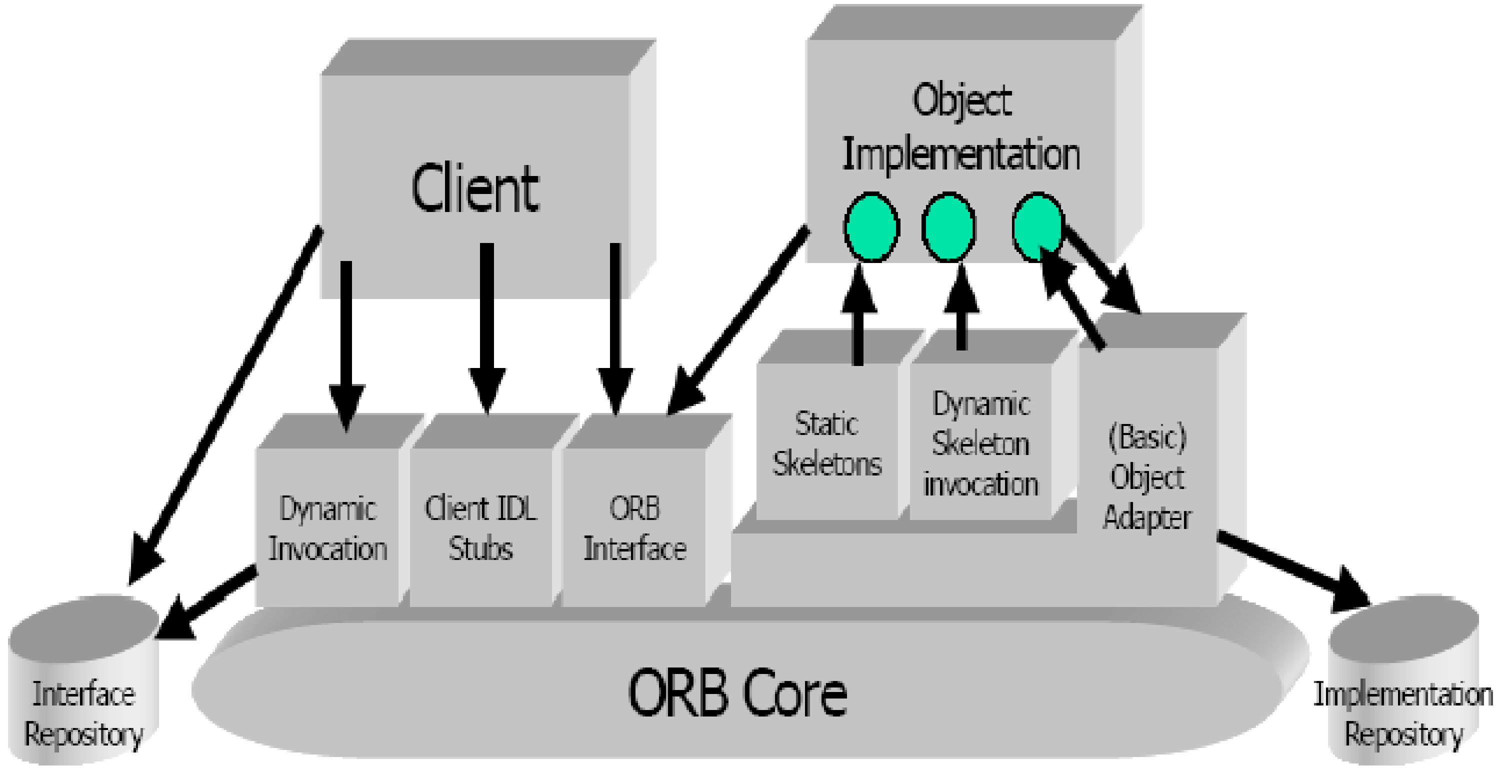
Object Request Broker(ORB)



Object Request Broker (ORB)

- ORB is a middleware application component that uses the common object request broker architecture (CORBA) specification, enabling developers to make application calls within a computer network.
- ORB is an agent that transmits client/server operation invocations in a distributed environment and ensures transparent object communication.
- **Major functionality includes:**
  - Interface definition
  - Location and activation of remote objects
  - Communication between clients and objects

CORBA architecture



CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - Open standard by the OMG (Object Management Group)
  - Language and platform independent
- **Object Request Broker** (**ORB**)
  - General Inter-ORB Protocol (GIOP) for communication
  - Interoperable Object References (IOR) contain object location
  - CORBA **Interface Definition Language** (IDL)
- **Stubs (proxies) and skeletons created by IDL compiler**
  - Dynamic remote method invocation
- **Interface Repository**
  - Querying existing remote interfaces
- **Implementation Repository**
  - Activating remote objects on demand

CORBA Services (selection)

- Naming Service
  - Names **->** remote object references
- Trading Service
  - Attributes (properties) **->** remote object references
- Persistent Object Service
  - Implementation of persistent CORBA objects
- Transaction Service
  - Making object invocation part of transactions
- Event Service and Notification Service
  - In response to applications‘ need for asynchronous communication
  - built above synchronous communication with ***push*** or ***pull*** options
  - *not* an integrated programming model with general IDL messages

Main CORBA features

- Object request broker (ORB)
- OMG(Object Management Group) interface definition language(IDL)
- Language mapping
- Stub and skeletons
- Interface repository
- Dynamic invocation and dispatch
- Object adapters
- Inter ORB protocols

Object Oriented Middleware(OOM)

- **Objects** can be ***local*** or ***remote***
- **Object references** can be *local* or *remote*
- Remote objects have visible **remote interfaces**
- Masks remote objects as being local using **proxy objects**
- **Remote method invocation**

Properties of OOM

- Support for object-oriented programming model
  - objects, methods, interfaces, encapsulation, …
  - exceptions (were also in some RPC systems e.g. Mayflower)
- Synchronous request/reply interaction
  - same as RPC
- Location Transparency
  - system (ORB) maps object references to locations
- Services comprising multiple servers are easier to build with OOM
  - RPC programming is in terms of ***server-interface (operation)***
  - RPC system looks up server address in a location service

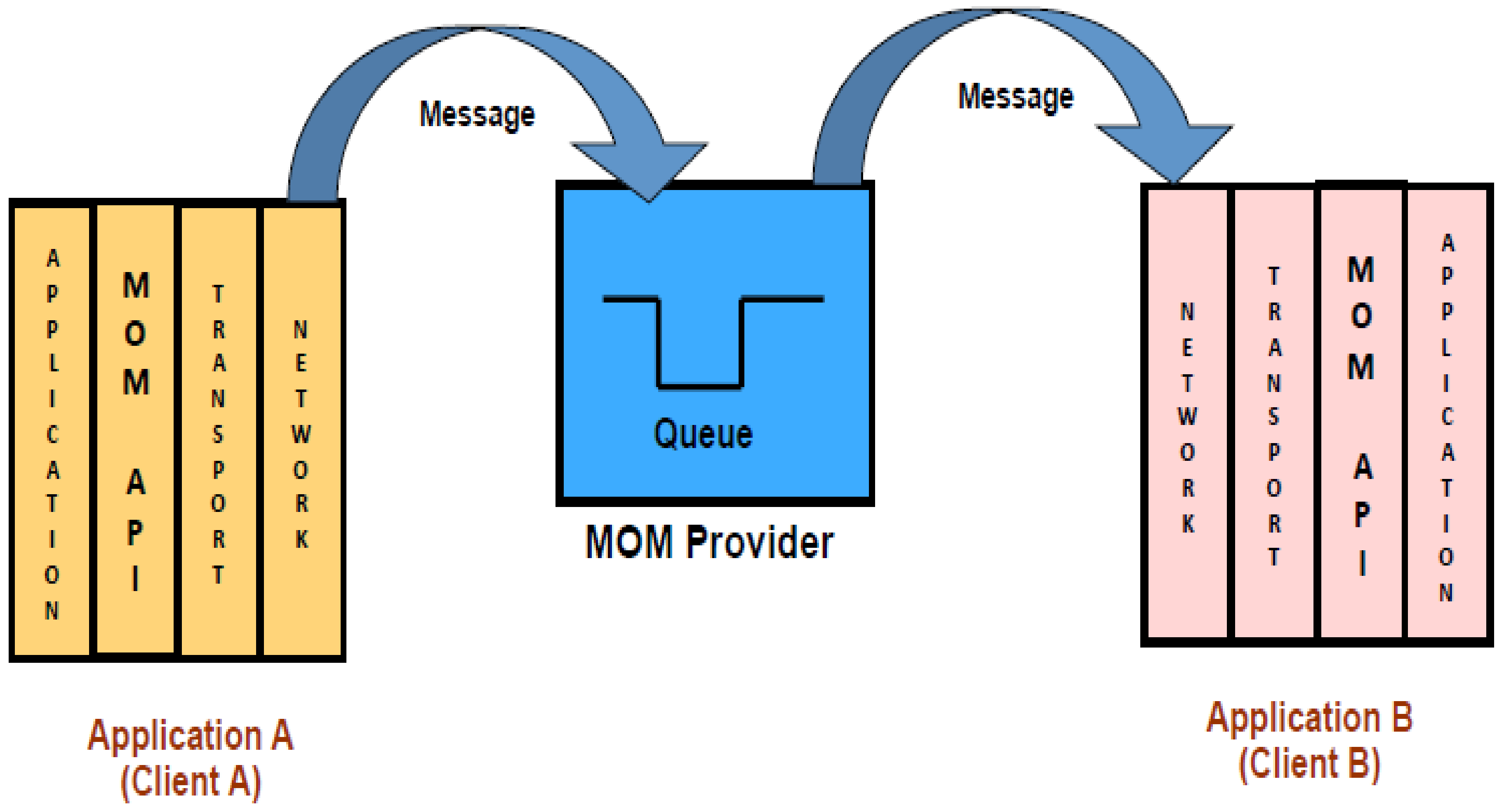
Disadvantages of OOM

- Synchronous request/reply interaction only
  - So CORBA **one way** semantics added and -
  - Asynchronous Method Invocation (AMI)
  - But *implementations* may not be loosely coupled
- Distributed garbage collection
  - Releasing memory for unused remote objects
- OOM rather static and heavy-weight
  - Bad for ubiquitous systems and embedded devices

Message Oriented Middleware(MOM)

- MOM (Message Oriented Middleware) is a client / server infrastructure which allows the application to be **distributed over multiple heterogeneous platforms.**
- Reduces complexity of applications spanning operating systems and network protocols by insulating them from un-necessary details.
- Data is exchanged by message passing and/or message queuing supporting **both synchronous and asynchronous** interactions between distributed computing processes.
- The MOM system ensures message delivery by using **reliable queues** and by providing the **directory**, **security**, and **administrative services** required to support messaging.

MOM - Demonstration

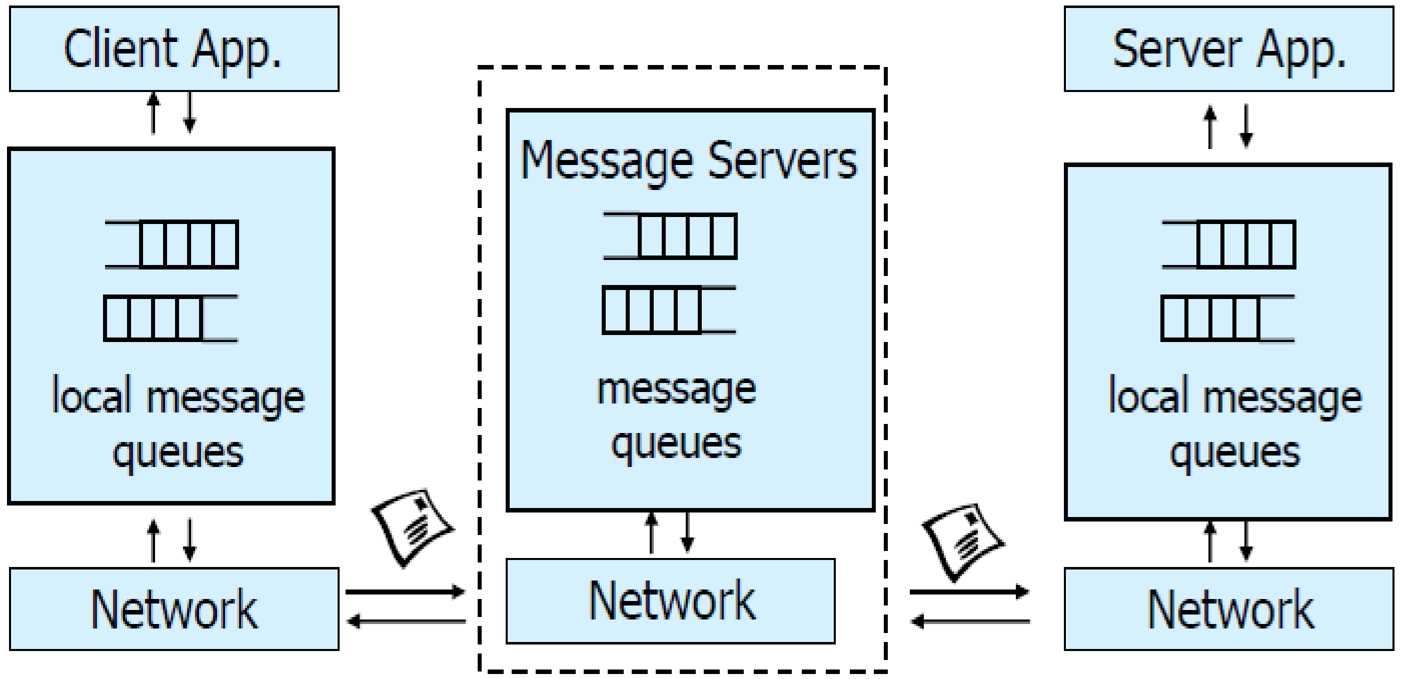


MOM - Advantages

- Asynchronous
- Flexible
- Portability
- Interoperability
- Reduces Complexity

Message-Oriented Middleware (MOM)

- Communication using **messages**
- Messages stored in **message queues**
- **message servers** decouple client and server
- Various assumptions about **message content**



Properties of MOM

- **Asynchronous interaction**
  - Client and server are only **loosely coupled**
  - Messages are queued
  - Good for application integration
- Support for **reliable delivery service**
  - Keep queues in persistent storage
- Processing of messages by intermediate message server(s)
  - May do filtering, transforming, logging, …
  - Networks of message servers
- Natural for database integration

Java Message Service (JMS)

- API **specification** to access MOM implementations
- Two modes of operation *specified*:
  - **Point-to-point**
    - one-to-one communication using queues
  - **Publish/Subscribe**
    - Event-Based Middleware
- **JMS Server** implements JMS API
- JMS Clients connect to JMS servers
- Java objects can be serialized to JMS messages
- A JMS interface has been provided for message queue(MQ)
- pub/sub (one-to-many) - just a specification?

Web Services

- A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service
- Use well-known web standards for distributed computing
  - **Communication**
    - Message content expressed in **XML**
    - Simple **O**bject **A**ccess **P**rotocol (**SOAP**)
      - Lightweight protocol for sync/async. communication
  - **Service Description**
    - **W**eb **S**ervices **D**escription **L**anguage (**WSDL**)
      - Interface description for web services
  - **Service Discovery**
    - **U**niversal **D**escription **D**iscovery and **I**ntegration (**UDDI**)
      - Directory with web service description in WSDL

Properties of Web Services

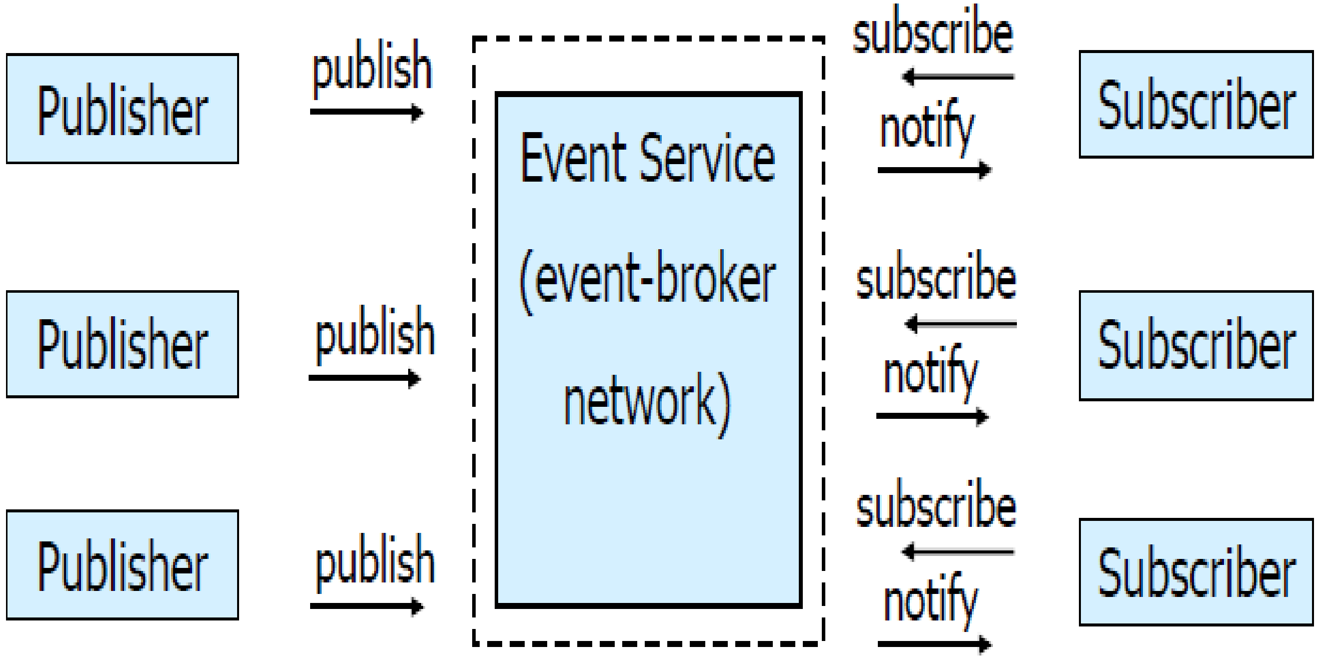
- Language-independent and open standard
- **SOAP** offers OOM and MOM-style communication:
  - Synchronous request/reply like OOM
  - Asynchronous messaging like MOM
  - Supports internet transports (http, smtp, ...)
  - Uses XML Schema for marshalling types to/from programming language types
- **WSDL** says how to use a web service
- **UDDI** helps to find the right web service
  - Exports SOAP API for access

Disadvantages of Web Services

- Low-level abstraction
  - leaves a lot to be implemented
- Interaction patterns have to be built
  - one-to-one and request-reply provided
  - one-to-many?
  - still synchronous service invocation, rather than notification
  - No nested/grouped invocations, transactions, ...
- No location transparency

Event-Based Middleware (Publish/Subscribe)

- **Publishers** (*advertise* and) *publish* **events** (messages)
- **Subscribers** express interest in events with *subscriptions*
- **Event Service** *notifies* interested subscribers of published events
- Events can have arbitrary content (typed) or name/value pairs



Properties of Publish/Subscribe

- Asynchronous communication
  - Publishers and subscribers are loosely coupled
- Many-to-many interaction between pubs. and subs.
  - Scalable scheme for large-scale systems
  - Publishers do not need to know subscribers, and vice-versa
  - Dynamic join and leave of pubs, subs
- (Topic and) Content-based pub/sub very expressive
  - Filtered information delivered only to interested parties
  - Efficient content-based routing through a broker network

COM/DCOM

- Component Object Model (COM)/Distributed Component Object Model(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

DCOM

- DCOM = COM binary standard

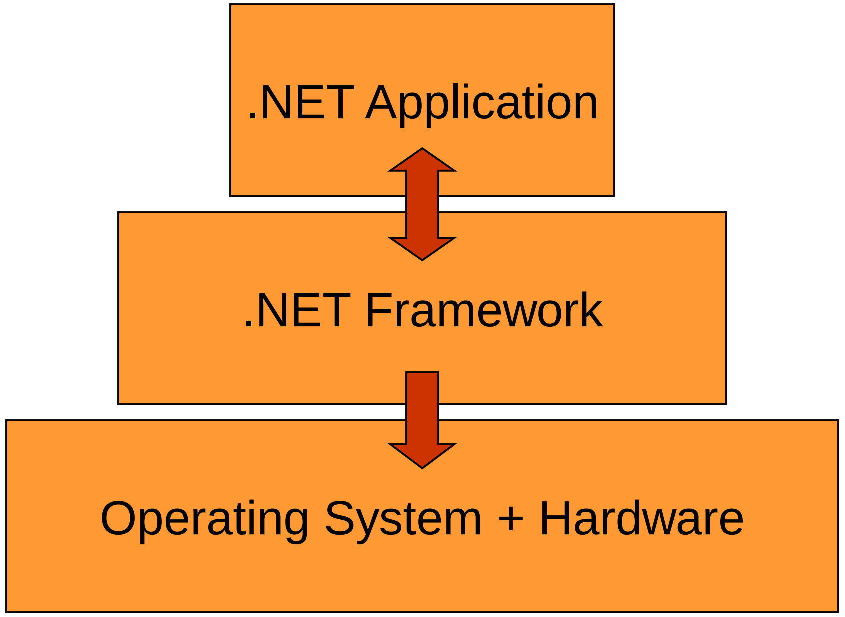
  +

  - runtime infrastructure for communicating across distributed address spaces initially only on Windows
- adding Mac and Unix
- 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

.NET – What Is It?

- Software platform
- Language neutral
- In other words:
  - .NET is not a language (Runtime and a library for writing and executing written programs in any compliant language)
  - .NET is a new framework for developing web-based and windows-based applications within the Microsoft environment.
  - The framework offers a fundamental shift in Microsoft strategy: it moves application development from client-centric to server-centric.
  - Dramatically simplifies development and deployment
  - Supports multiple programming languages

.NET – What Is It?



.NET Framework Services

- Common Language Runtime
- Windows Forms
- ASP.NET
  - Web Forms
  - Web Services
- ADO.NET, evolution of ADO
- Visual Studio.NET

Note

- Common Language Runtime
  - Common, secure execution environment.
  - We’ll drill into this in some detail in the first parts of the presentation.
- Windows® forms
  - Framework for building rich clients
  - A demonstration will highlight some of these features, such as the delegate-based event model.
- ASP.NET
  - Web forms
    - Manageable code (non spaghetti)
    - Logical evolution of ASP (compiled)
    - Again, we’ll drill into a hint at the power of Web Forms with a demonstration
  - Web Services
    - Programming the Internet to leverage the "power at the edge of the cloud".
    - We will cover this in detail, as this – along with the CLR – is one of the more powerful aspects of .NET Framework.
- ADO.NET, evolution of ADO
  - New objects (e.g., DataSets, Datareader)
- Visual Studio.NET
  - Most productive development environment gets better and fully supports the .NET Framework

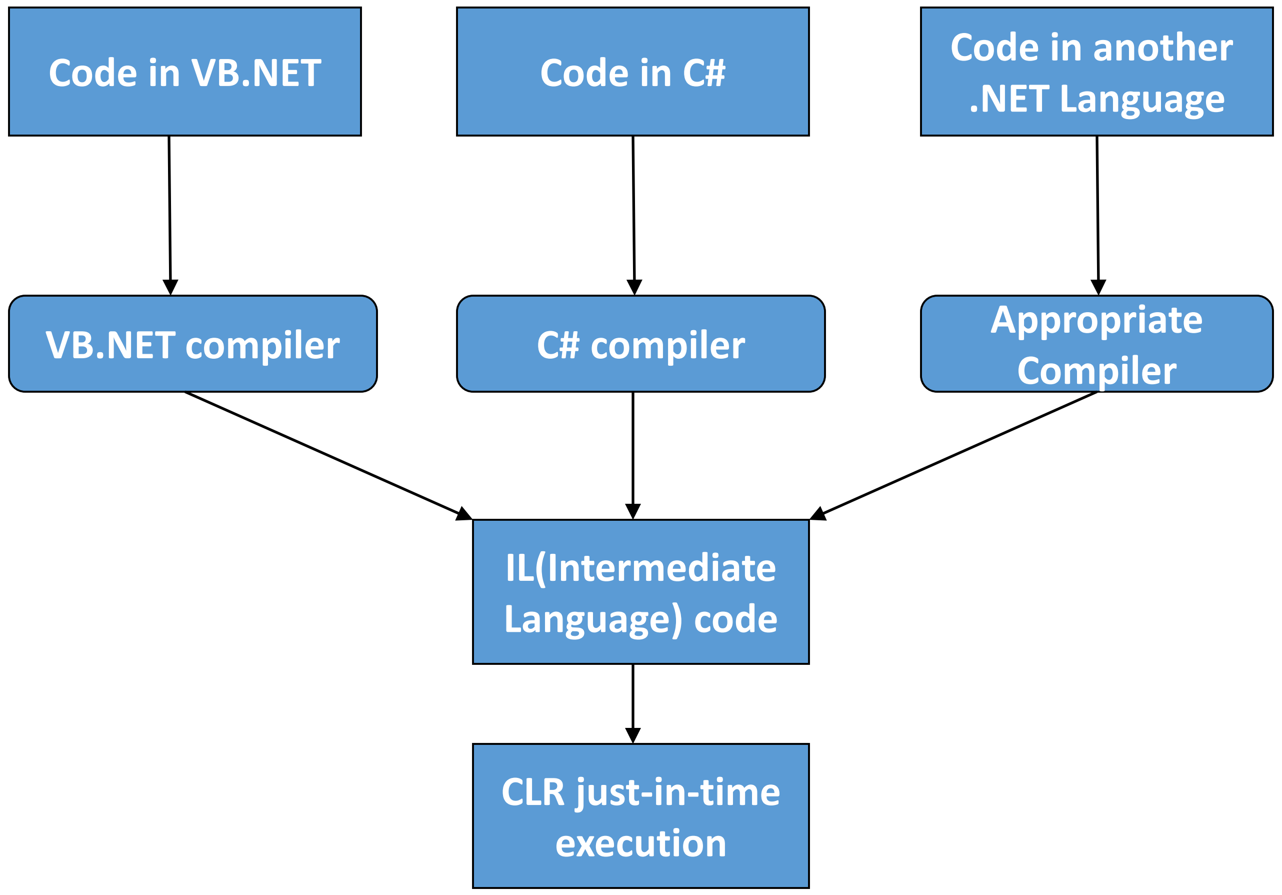
Common Language Runtime (CLR)

- CLR works like a virtual machine in executing all languages.
- All .NET languages must obey the rules and standards imposed by CLR. Examples:
    - Object declaration, creation and use
    - Data types, language libraries
    - Error and exception handling
    - Interactive Development Environment (IDE)
- Development
    - Mixed language applications
        - Common Language Specification (CLS)
        - Common Type System (CTS)
        - Automatic memory management
    - Consistent error handling
- Deployment
    - Removal of registration dependency

Contd..

- CTS is a rich type system built into the CLR
  - Implements various types (int, double, etc)
  - And operations on those types
- CLS is a set of specifications that language and library designers need to follow
  - This will ensure interoperability between languages
- CLR sits on top of OS to provide a *virtual environment* for hosting managed applications
  - What is CLR similar to in Java?
  - Java Virtual Machine (JVM)
- CLR loads modules containing executable and executes their code
- IL(Intermediate Language) instructions are just-in-time (JIT) compiled into native machine code at run time

Compilation in .NET


Code in VB.NET
Code in C#
Code in another .NET Language
VB.NET compiler
C# compiler
Appropriate Compiler
IL(Intermediate Language) code
CLR just-in-time execution

Intermediate Language (IL)

- .NET languages are not compiled to machine code.
- They are compiled to an Intermediate Language (IL).
- CLR accepts the IL code and recompiles it to machine code.
- The recompilation is just-in-time (JIT) meaning it is done as soon as a function or subroutine is called.
- The JIT code stays in memory for subsequent calls.
- In cases where there is not enough memory it is discarded thus making JIT process interpretive.

Languages

- Languages provided by MS
  - VB, C++, C#, J#, JScript
- Third-parties are building
  - APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk…

Windows Forms

- Framework for Building Rich Clients
  - RAD (Rapid Application Development)
  - Rich set of controls
  - Data aware
  - Printing support
  - UI inheritance

Thank You !