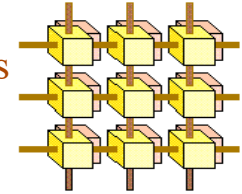


# **Basics of Distributed Object Systems**

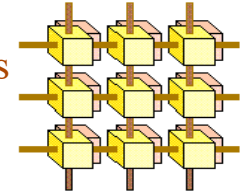
***ECE 6102: Spring 2009***





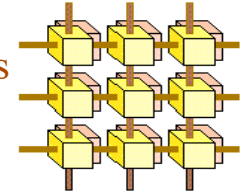
# Distributed Object Systems

- Extend object-oriented programming to distributed systems, i.e. allow objects to be distributed across multiple nodes
- Local process can operate on objects stored on other nodes in exactly the same way as operating on a local object (remote method invocation - RMI)
- Client/server model: local process is client, remote process hosting object is server
- Often allow interoperability of heterogeneous systems, e.g. different programming languages, different OSes
- Advantages: modularity, reusability, extensibility, interoperability



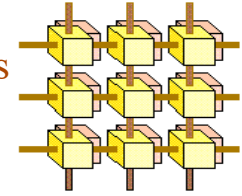
# Examples of Distributed Object Systems

- Java/RMI
  - **Java/RMI** relies on a protocol called the **Java Remote Method Protocol**
  - Portable across operating systems
  - Requires Java Virtual Machine (JVM) implementation
  - Uses TCP/IP for communication
- DCOM
  - Supports remote objects by running on a protocol called the **Object Remote Procedure Call**
  - Is language independent
  - Requires a COM platform, i.e. Windows machine
- CORBA
  - Uses a protocol called **Internet Inter-ORB Protocol (IIOP)**
  - Platform independent and language independent
  - Well-suited for complex heterogeneous systems (e.g. DoD systems)



## Distributed Object Systems vs. Other Technologies

- Web Services is *not* a distributed object system but rather a distributed system technology for exchanging XML documents
- Simple Object Access Protocol (SOAP) was created to allow distributed object systems to be implemented via Web Services messages - details of object type and method invocation are encoded in an XML file
- .NET supports both Web services and full distributed object systems (.NET was mainly intended to support Web Services but .NET remoting provides full distributed object support)



## Distributed Object Systems: Summary

- Single language, multiple platforms: Java RMI
- Single platform, multiple languages: DCOM, .NET remoting
- Multiple platforms, multiple languages: CORBA, SOAP (not a complete object system)

# SOAP: Simple Object Access Protocol

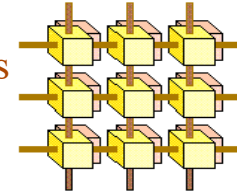
- A SOAP message is an XML document containing:
  - An Envelope element, which identifies the XML document as a SOAP message
  - A Header element, which contains application-specific header information
  - A Body element, which contains call and response information
  - A Fault element, inside the Body, which contains errors and status information

# Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

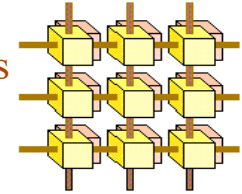


## SOAP Body - Method Call

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body>
    <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
      <m:Item>Apples</m:Item>
    </m:GetPrice>
  </soap:Body>

</soap:Envelope>
```



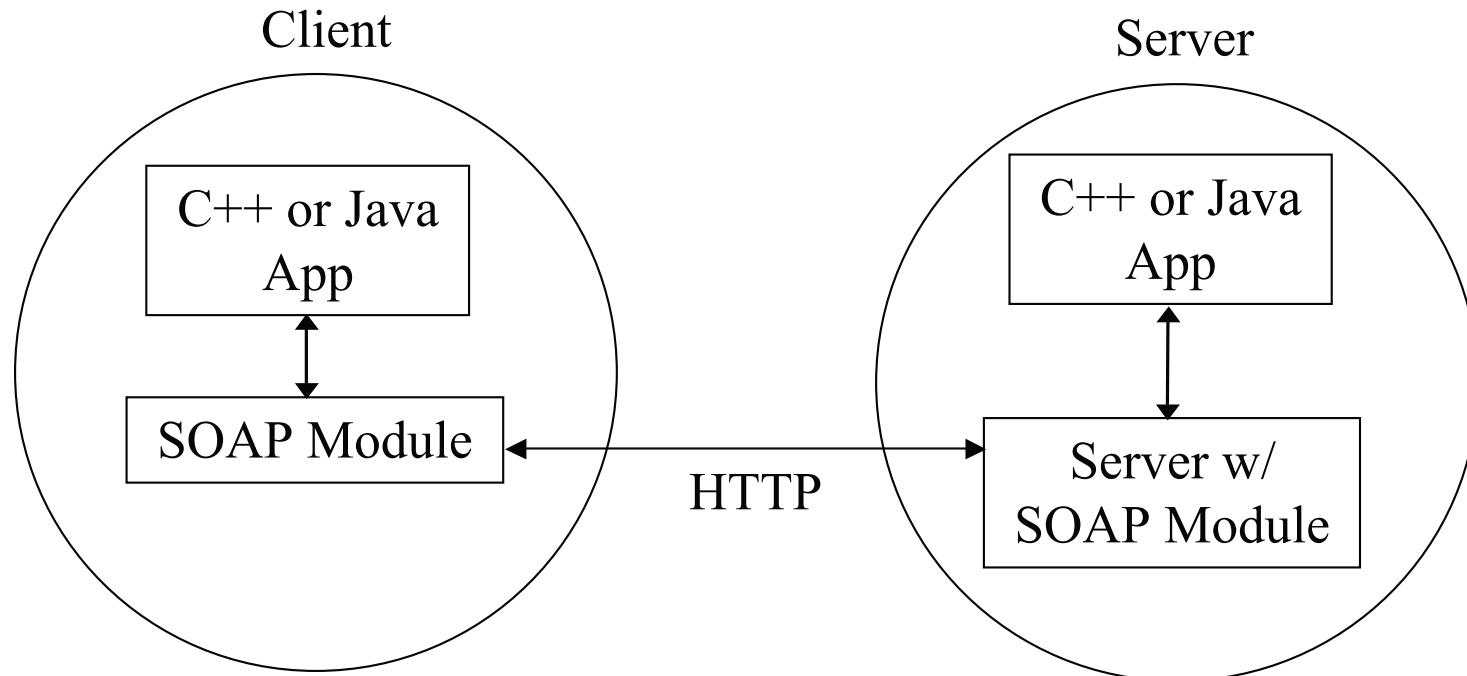
## SOAP Body - Response

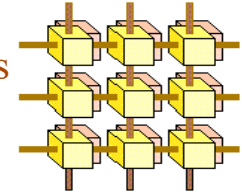
```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body>
    <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
      <m:Price>1.90</m:Price>
    </m:GetPriceResponse>
  </soap:Body>

</soap:Envelope>
```

# SOAP Operation

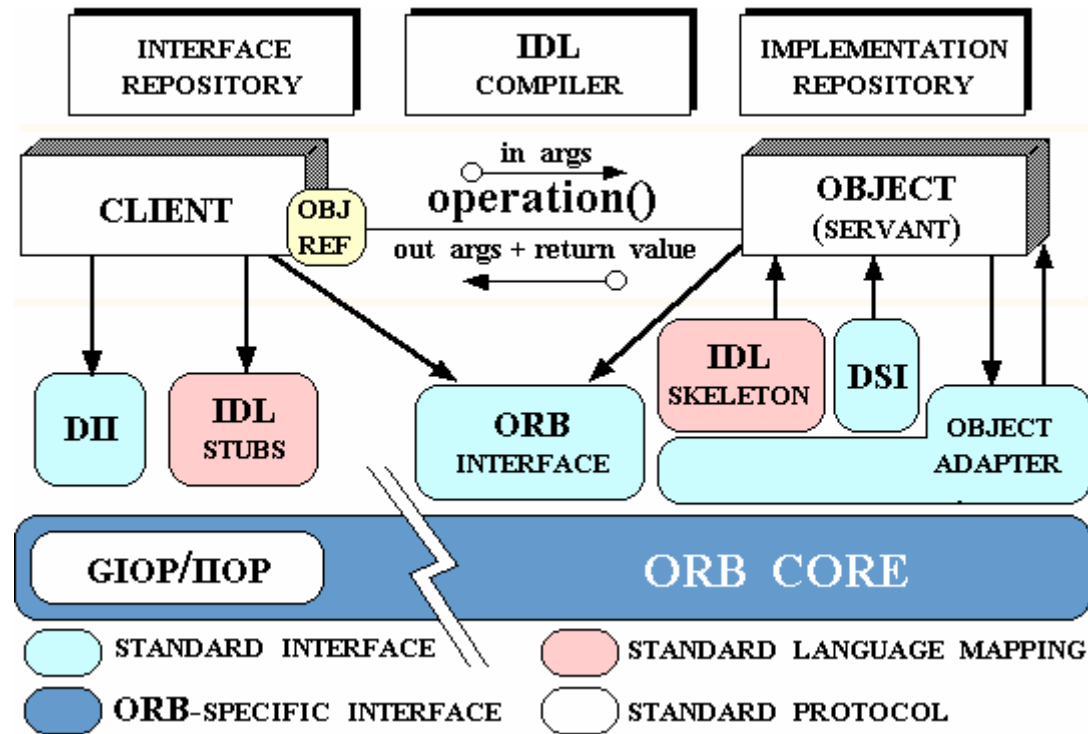


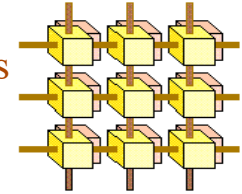


## Common Features of Distributed Object Systems

- Interface Definition Language (IDL): language for describing object interfaces, i.e. what methods the object has and the number and types of parameters for each method
- IDL allows clients and servers to execute remote method calls properly
- Dynamic invocation: ability for clients to discover object interfaces dynamically, allows clients to execute remote method calls without compile-time knowledge of an object interface

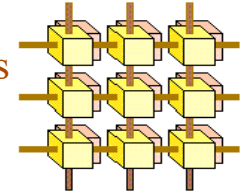
# CORBA Architecture





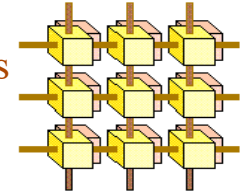
# CORBA Features

- Object Request Broker (ORB) contains mechanisms to:
  - find object implementation for a given remote method invocation
  - pass invocation request to object implementation
  - return invocation result to client
  - support various other services
- Object reference is used by client to access remote object
- OMG Interface Definition Language (IDL)
  - Language-independent specification of object interfaces
  - IDL compilers convert IDL to specific language code, e.g. C++ or Java
  - IDL compilers generate stubs (client) and skeletons (server) that interact for proper execution of a remote method invocation



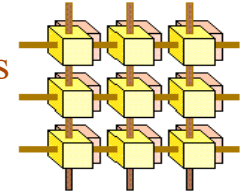
## CORBA Features (continued)

- Stubs/skeletons: one stub or skeleton for each method of an object, skeleton simply calls appropriate method on the referenced object
- Dynamic interfaces: dynamic invocation interface (DII) and dynamic skeleton interface (DSI)
  - one DII or DSI per process, each one can invoke any method on any object, DSI must be capable of invoking any method on server
- It is possible for a stub to call DSI and DII to call a skeleton (server chooses to have a separate skeleton for each method or one DSI for all methods, method calls access whichever type is available)



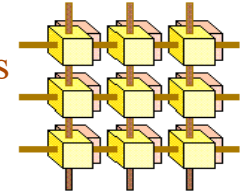
## CORBA Features (continued)

- Remote method invocation example (stub/skeleton case):
  - Client invokes method on a remote object reference
  - Code for method call on client resides in stub
  - Stub passes method call arguments to ORB
  - ORB sends object reference and arguments to skeleton on server
  - Skeleton invokes method call and gathers outputs
  - Skeleton passes method call result and outputs to ORB
  - ORB sends result and outputs back to stub
  - Stub processes result and outputs and returns to client application



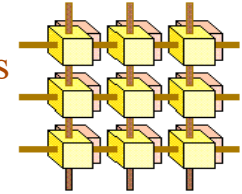
## CORBA Features (continued)

- Interface repository stores object interface information (methods and parameters) that is accessible at run time
- Dynamic object invocation
  - Client gets object interface information from interface repository at run time
  - Client uses DII to invoke a method call on an object
- General Inter-Orb Protocol (GIOP) specifies a standard transfer syntax and set of message formats for communication between ORBs
- Internet Inter-Orb Protocol (IIOP) specifies how GIOP messages are exchanged using TCP/IP



# Implementation Repository

- Maintain a registry of known servers
- For each server, record which host it is running on and which port it uses
- Start servers on demand if they are registered with the Implementation Repository
- Periodically invoke a one-way method on the ping object to monitor the registered servers



## ORB Implementation Types

- Client- and server-resident ORB: ORB is a separate process on each client/server machine, stubs/skeletons interact with ORB through standard IPC mechanisms (one ORB per machine)
- Server-based ORB: ORB is a centralized server with which all clients and servers communicate (one ORB per distributed system)
- System-based ORB: ORB is part of the underlying OS on each machine (one ORB per machine)
- Library-based ORB: ORB is implemented within libraries that are linked into client and server processes (one ORB per client or server)