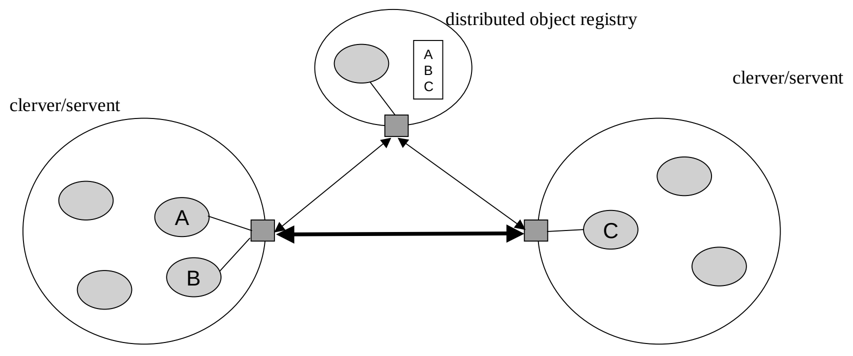
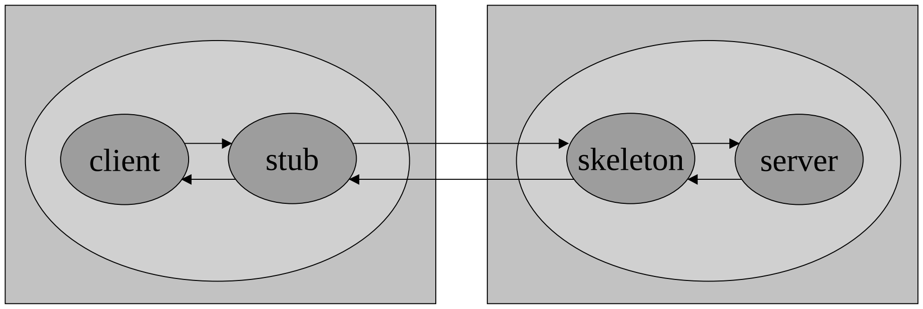
Distributed Objects

Java Remote Method Invocation
Enterprise Java Beans

DO Basic Idea

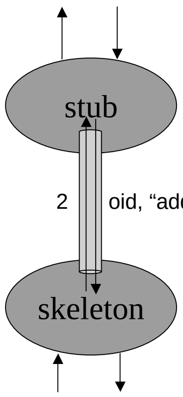


Marshalling Parameters



Marshalling Parameters

```
int result = arithmeticServer.addOne(1)
```



```
public int addOne(int x) { return x+1; }
```

Three Major Standards

- CORBA
  - Common Object Request Broker Architecture
  - Industry sponsored standard
- DCOM
  - Distributed Component Object Model
    - Microsoft
    - from COM from OLE
- Java RMI
  - Remote Method Invocation
- all can be made to be inter-operable

Java RMI Client Code

```
public interface ArithmeticServer extends java.rmi.Remote {
   public int addOne(int i) throws java.rmi.RemoteException;
}



public class ArithmeticClient {
    public static void main(String args[]) throws Exception {
        ArithmeticServer =
             (ArithmeticServer)java.rmi.Naming.lookup(
                            "rmi://penny.dhcp/ArithmeticServer");
        System.out.println(as.addOne(1));
    }
}
```

Java RMI Server Code

```
public interface ArithmeticServer extends java.rmi.Remote {
   public int addOne(int i) throws java.rmi.RemoteException;
}

public class ArithmeticServerImpl
    extends    java.rmi.server.UnicastRemoteObject
    implements ArithmeticServer
{
    public ArithmeticServerImpl() throws java.rmi.RemoteException {
        super();
    }

    public int addOne(int i) { return i+1; }

    public static void main(String[] args) throws Exception {
        java.rmi.Naming.rebind("ArithmeticServer",
                               new ArithmeticServerImpl());
    }
}
```

Compilation

```
[CLIENT]
% javac ArithmeticServer.java ArithmeticClient.java



[SERVER]
% javac ArithmeticServer.java ArithmeticServerImpl.java
% rmic –keep ArithmeticServerImpl
% javac ArithmeticServer_Stub.java ArithmeticServer_Skel.java
```

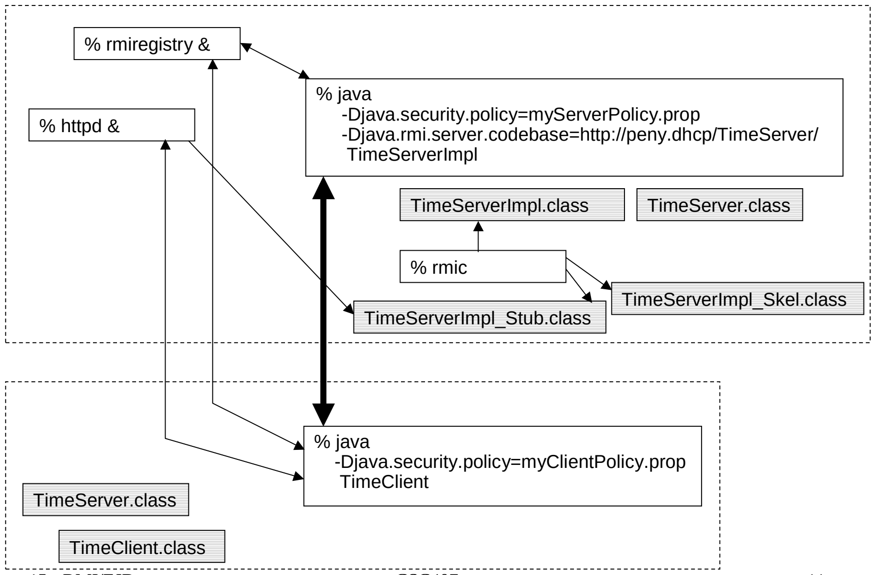
Generated Stub Code

```
public final class ArithmeticServerImpl_Stub
    extends RemoteStub
    implements ArithmeticServer, Remote
{
    private static final java.rmi.server.Operation[] operations =
          { new java.rmi.server.Operation("int addOne(int)") };

    private static final long interfaceHash = 2100571976616716783L;

    public int addOne(int param_int_1) throws java.rmi.RemoteException {
        java.rmi.server.RemoteCall call = super.ref.newCall(
                (java.rmi.server.RemoteObject) this, operations, 0, interfaceHash);

        java.io.ObjectOutput out = call.getOutputStream();
        out.writeInt(param_int_1);

        super.ref.invoke(call);

        int result;
        java.io.ObjectInput in = call.getInputStream();
        result = in.readInt();

        ref.done(call);
        return result;
    }
}
```

Generated Skeleton Code

```
public final class ArithmeticServerImpl_Skel implements java.rmi.server.Skeleton {

    public void dispatch(Remote obj, RemoteCall call, int opnum, long hash) {
          if (hash != interfaceHash)
              throw new SkeletonMismatchException("interface hash mismatch");

          ArithmeticServerImpl server = (ArithmeticServerImpl) obj;
          switch (opnum) {
          case 0: // addOne(int)
          {
            int param_int_1;

            java.io.ObjectInput in = call.getInputStream();
            param_int_1 = in.readInt();
            call.releaseInputStream();

            int $result = server.addOne(param_int_1);

            java.io.ObjectOutput out = call.getResultStream(true);
            out.writeInt($result);

            break;
          }

          default: throw new UnmarshalException("invalid method number");
    }
}
```

Execution



```
% java
    -Djava.security.policy=myServerPolicy.prop
    -Djava.rmi.server.codebase=http://peny.dhcp/TimeServer/
     TimeServerImpl
```

```
% java
    -Djava.security.policy=myClientPolicy.prop
     TimeClient
```

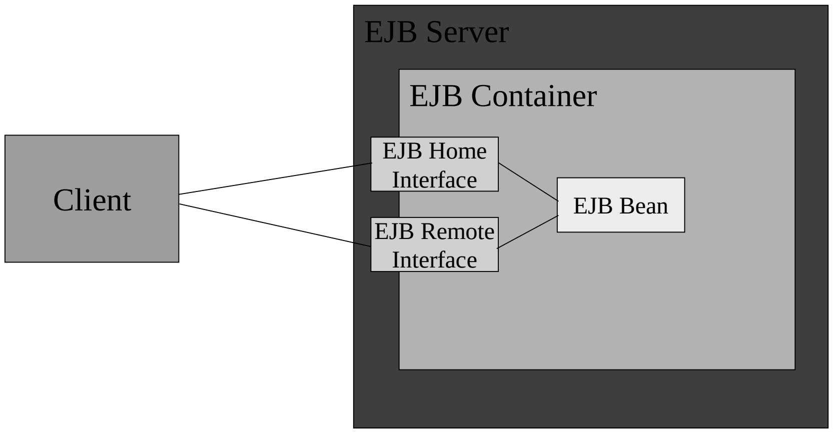
Performance

- Latency: arithmeticServer.addOne(1);
  - Local method calls
    - .07 usec
  - Remote method call (same machine)
    - 656 usec
  - Remote method call (network)
    - 2000 usec
- DB access
    - 1600 usec

Enterprise Java Beans

- Component Object Model
  - Distributed
  - Persistent
  - Secure
  - Transactional
    - ACID
      - Atomicity: all or none
      - Consistency: database will always be in a consistent state
      - Isolation: intermediate state not visible until completed
      - Durability: when completed, the changes are stored permanently
- EJBs are a standard
  - allows application developers to write simple, standard code
  - allows implementers to get all the underlying stuff done well

EJB Architecture



Context

```
public interface javax.ejb.EJBContext {
    public abstract Identity getCallerIdentity();
    public abstract EJBHome getEJBHome();
    public abstract Properties getEnvironment();
    public abstract boolean getRollbackOnly();
    public abstract UserTransaction getUserTransaction();
    public abstract boolean isCallerInRole(Identity role);
    public abstract void setRollbackOnly();
}
```

Types of EJBs

- Two types of beans:
  - Session bean
    - encapsulates transactional operations
    - stateful/stateless
  - Entity bean
    - encapsulates persistent state
    - container-managed persistence / bean-managed persistence

EJBs

- Remote Interface

```
public interface GroceryOrder extends javax.ejb.EJBObject {
    public Date getDate() throws RemoteException;
    public void setDate() throws RemoteException;
    …
}
```

- Home Interface

```
public interface GroceryOrderHome extends javax.ejb.EJBHome {
    public GroceryOrder create(int id)
        throws CreateException, RemoteException;
    public GroceryOrder findByPrimaryKey(GroceryOrderPK pk)
        throws FinderException, RemoteException;
}
```

EJB Implementation Class

```
public class GroceryOrderBean implements javax.ejb.EntityBean {
    public int id;
    public Date date;

    public void ejbCreate(int id) { this.id = id; }
    public void ejbPostCreate(int id) { }
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }
    public void setEntityContext(EntityContext ctx) { }
    public void unsetEntityContext() { }
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbLoad() { }
    public void ejbStore() { }
    public void ejbRemote() { }
}
```

Session Beans

```
public class ShopperBean implement javax.ejb.SessionBean {
    public Customer customer;
    public GorceryOrder order;

    public void ejbCreate(Customer cust) { customer = cust; }

    public Receipt processOrder(CreditCard card)
        throws RemoteException,
               IncompleteConversationalState,
               BadCredit
    {
         if(customer==null||order==null) throw new IncompleteConversationalState();

         ProcessOrderHome poh = (ProcessOrderHome)getHome("ProcessOrderHome");
         ProcessOrder po = poh.create(customer, order);

         ProcessPaymentHome pph = (ProcessPaymentHome)getHome("ProcessPaymentHome");
         ProcessPayment pp = ppHome.create();

         pp.byCreditCard(customer, card, order.price());
         po.process();

         Receipt r = new Receipt(customer, order, card);
         return r;
    }
```

EJB Summary

- Transparent
  - Distribution
    - ejb can be anywhere
  - Replication & Load-Balancing
    - ejb can be moved around
    - ejb can be replicated (e.g., Toronto – London)
  - Resource Management
    - ejb shells can be reused
    - persistent data can be cached
  - Persistence Management
    - ejb automatically mapped to persistent storage
  - Transaction Management
    - session beans mapped to transactional system
  - Security
    - Identities, roles, access control lists

EJB Implementations

- Still pretty flaky and none support everything on the previous list.
  - WebLogic
  - EJBHome
  - SapphireWeb
  - BEA
  - Gemstone
  - IBM CICS/EJB, ComponentBroker, WebSphere
  - NetDynamics
  - Oracle Application Server
  - …