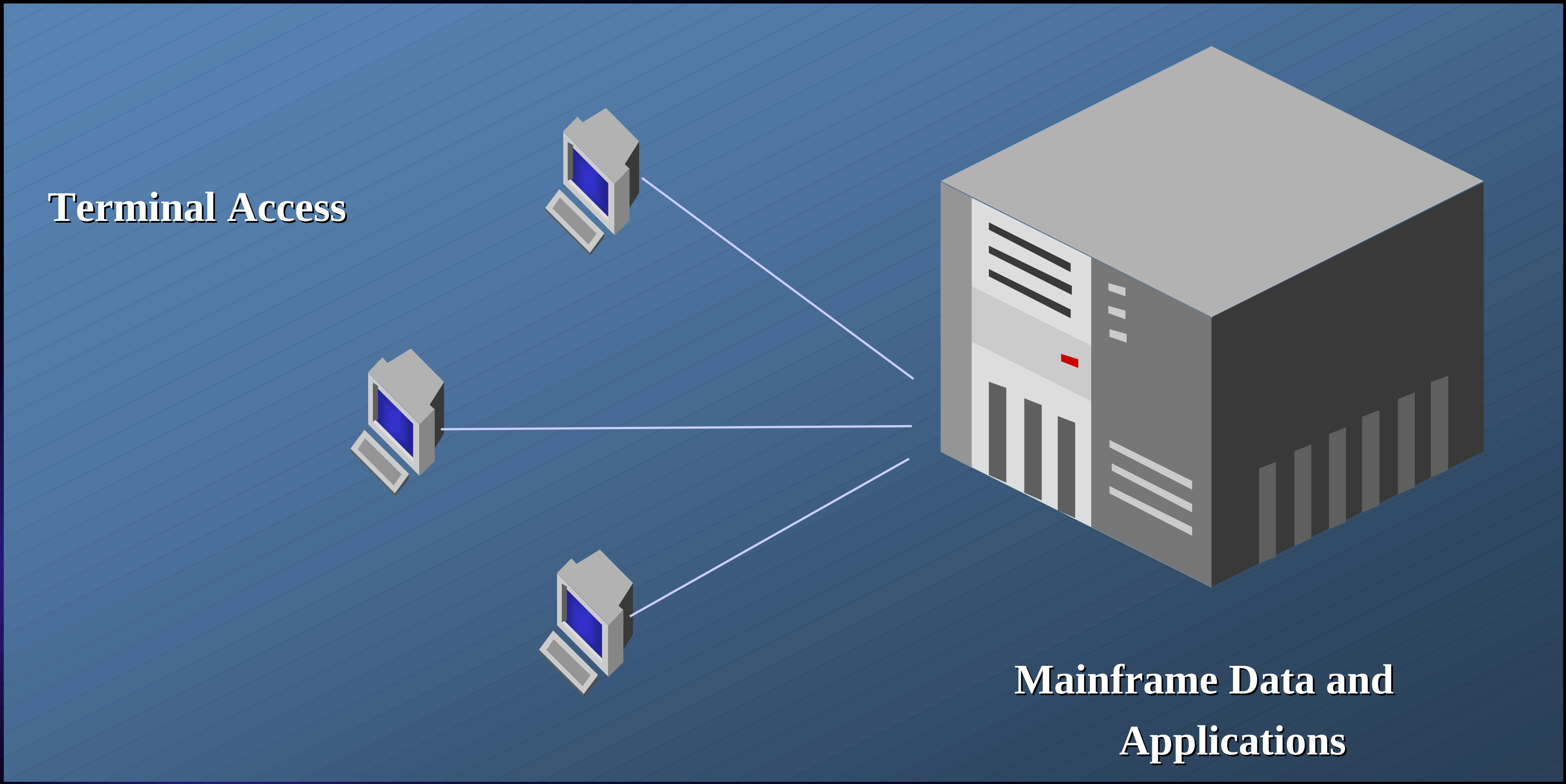
AN INTRODUCTION TO CORBA

Paul Jonusaitis
jonusait@ix.netcom.com

Topics for this presentation:

- The need for and origins of CORBA
- Basic elements:
  - ORBs, stubs, skeletons, IIOP, IDL
- Simple code examples in Java and C++
- CORBA services:
  - naming, events, notification, transaction
- the future of CORBA and Java/EJB
- Overview of CORBA implementations
- CORBA resources

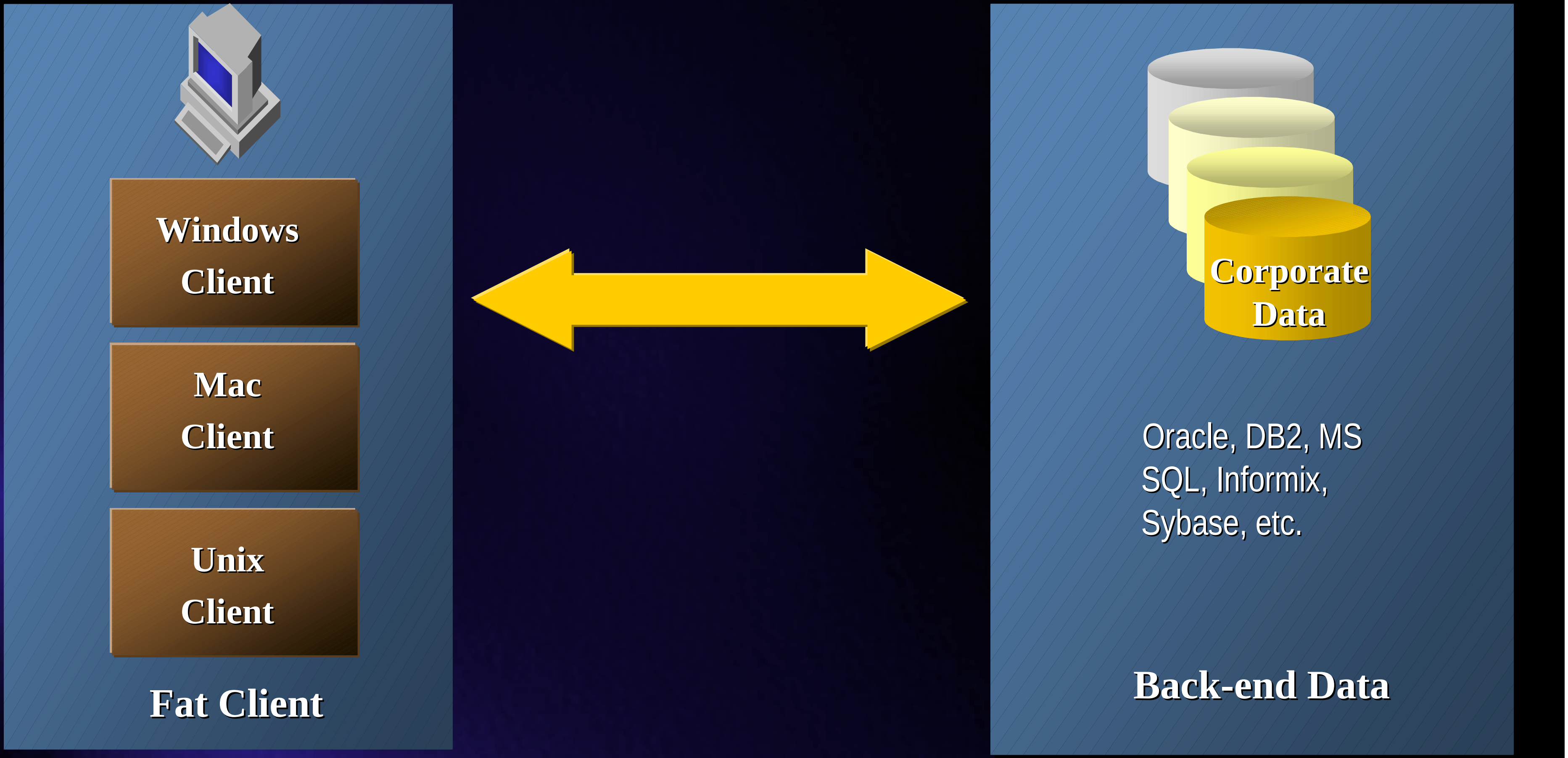
From mainframe applications...



Terminal Access

Mainframe Data and Applications

...to client/server applications...



Windows Client

Mac Client

Unix Client

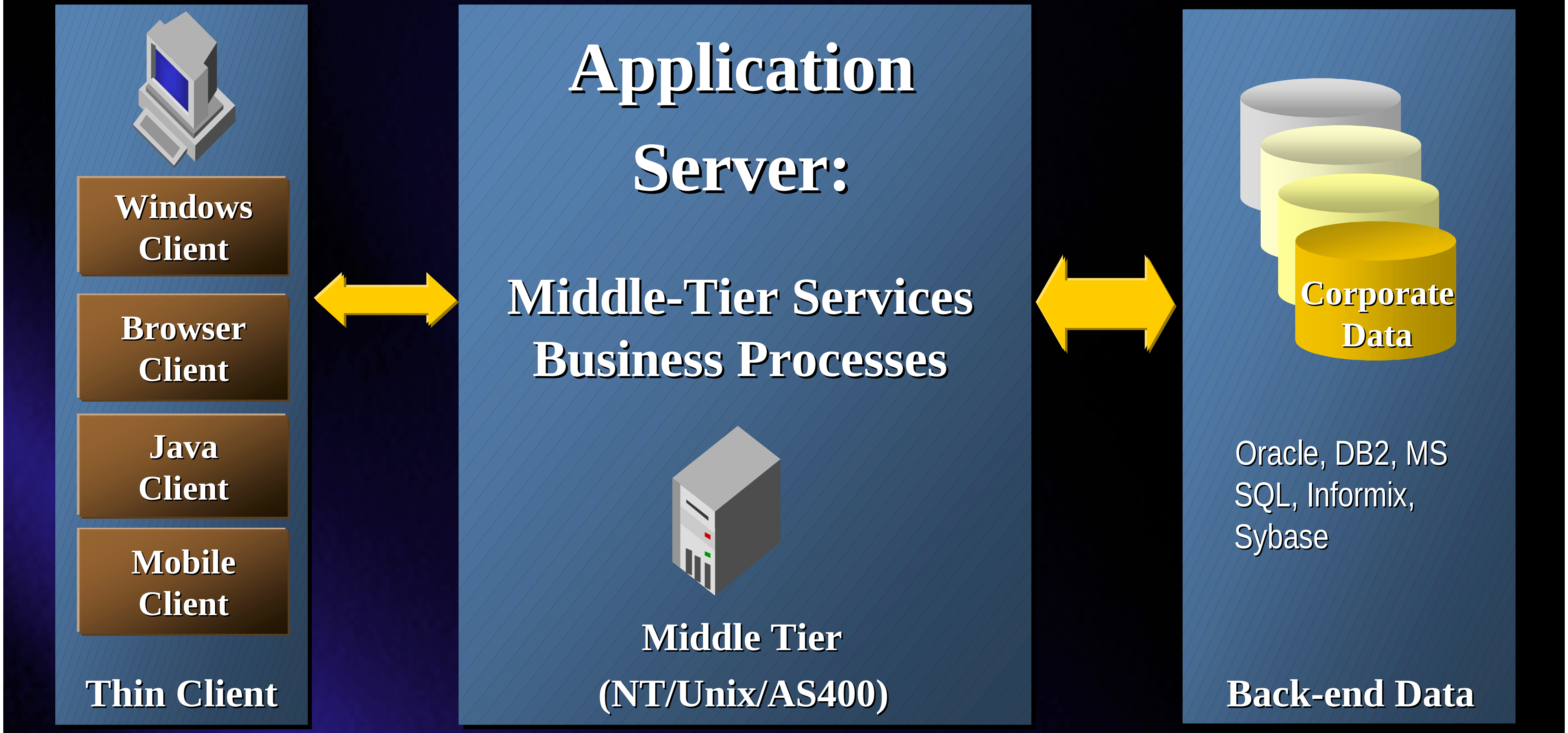
Fat Client

Corporate Data

Oracle, DB2, MS SQL, Informix, Sybase, etc.

Back-end Data

...to multi-tier distributed applications



Windows Client

Browser Client

Java Client

Mobile Client

Thin Client

Application Server:

Middle-Tier Services
Business Processes

Middle Tier
(NT/Unix/AS400)

Corporate Data

Oracle, DB2, MS SQL, Informix, Sybase

Back-end Data

Enterprise computing

- Enterprises have a variety of computing platforms
  - Unix, 95/98/NT, MVS, AS/400, VMS, Macintosh, NC’s, VxWorks, etc.
- Enterprises write applications in a variety of programming languages
  - C, C++, Java, COBOL, Basic, Perl, Smalltalk, etc.
- Enterprises need an open architecture to support the heterogeneous environment

Object-oriented computing for the enterprise

- Enterprise applications are being written in terms of *objects* - reusable components that can be accessed over the enterprise network
- CORBA supplies the architecture for distributed applications based on open standards

Distributed application advantages

- Scalability
  - Server replication
  - Thin, heterogeneous clients
- Re-usability
- Partitioned functionality = easy updating of either clients or servers

Competing technologies for distributed objects

- Open standards based solutions
  - Java, CORBA, EJB, RMI, IIOP, JTS/OTS, JNDI, JDBC,, Servlets, JSP, Java Security
- The All-Microsoft solution
  - COM, COM+, ActiveX, Visual C++, MTS, ASP, IIS, etc.
- Other proprietary solutions
  - Message oriented middleware (MOMs - MQSeries, etc.)
  - TP monitors

TP monitors, web front-ends

Example: BEA Jolt

- Quickly extends an existing application for access from the web
- Client context maintained by server

- Limited to single process, single machine
- Not object oriented or truly distributed
- Jolt server consumes an additional process
- Jolt client classes must be either pre-installed or downloaded

COM/DCOM, COM+

- Rich, well-integrated platform
- Object-oriented
- Web client access via:
  - ActiveX controls & COM/DCOM
  - Active Server Pages, HTTP and IIS
- Distributed - as long as its Windows

- NT only
- Firewall issue
- Limited flexibility
- Security

CORBA vs. ad-hoc networked apps

- Technical considerations:
- CORBA/EJB implementations have integration with object databases, transaction services, security services, directory services, etc.
- CORBA implementations automatically optimize transport and marshalling strategies
- CORBA implementations automatically provide threading models

CORBA vs. ad-hoc networked apps

- Business considerations:
- Standards based
- Multiple competing interoperable implementations
- Buy vs. build tradeoffs
- Resource availability
  - software engineers
  - tools

The Object Management Group (OMG)

- Industry Consortium with over 855 member companies formed to develop a distributed object standard
- Accepted proposals for the various specifications put forth to define:
  - Communications infrastructure
  - Standard interface between objects
  - Object services
- Developed the spec for the Common Object Request Broker Architecture (CORBA)

CORBA design goals/characteristics:

- No need to pre-determine:
  - The programming language
  - The hardware platform
  - The operating system
  - The specific object request broker
  - The degree of object distribution
- Open Architecture:
  - Language-neutral Interface Definition Language (IDL)
  - Language, platform and location transparent
- Objects could act as clients, servers or both
- The Object Request Broker (ORB) mediates the interaction between client and object

IIOP - Internet Inter-ORB Protocol

- Specified by the OMG as the standard communication protocol between ORBs
- Resides on top of TCP/IP
- Developers don’t need to “learn” IIOP; the ORB handles this for them
- Specifies common format for:
  - object references, known as the Interoperable Object Reference (IOR)
  - Messages exchanged between a client and the object

Key definitions: ORB and BOA

- Object Request Broker (ORB)
  - Transports a client request to a remote object an returns the result. Implemented as:
    - a set of client and server side libraries
    - zero or more daemons in between, depending on ORB implementation, invocation method, etc.
- Object Adapter (OA), an abstract specification
  - Part of the server-side library - the interface between the ORB and the server process
  - listens for client connections and requests
  - maps the inbound requests to the desired target object instance
- Basic Object Adapter (BOA), a concrete specification
  - The first defined OA for use in CORBA-compliant ORBs
  - leaves many features unsupported, requiring proprietary extensions
  - superceded by the Portable Object Adapter (POA), facilitating server-side ORB-neutral code

What is an object reference?

- An object reference is the distributed computing equivalent of a pointer
  - CORBA defines the Interoperable Object Reference (IOR)
    - IORs can be converted from raw reference to string form, and back
    - Stringified IORs can be stored and retrieved by clients and servers using other ORBs
  - an IOR contains a fixed object key, containing:
    - the object's fully qualified interface name (repository ID)
    - user-defined data for the instance identifier
  - An IOR can also contain transient information, such as:
    - The host and port of its server
    - metadata about the server's ORB, for potential optimizations
    - optional user defined data

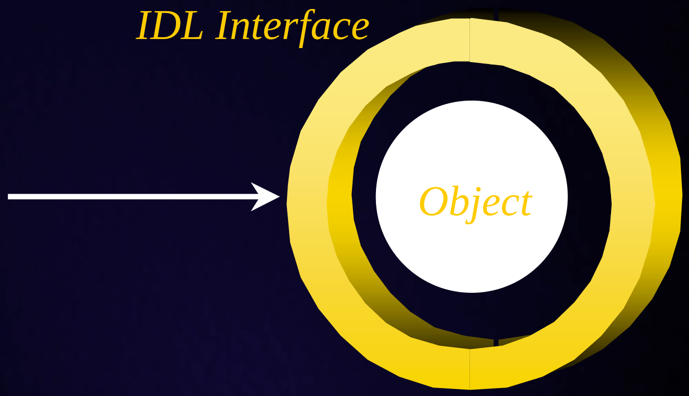
CORBA object characteristics

- CORBA objects have identity
  - A CORBA server can contain multiple instances of multiple interfaces
  - An IOR uniquely identifies one object instance
- CORBA object references can be persistent
  - Some CORBA objects are transient, short-lived and used by only one client
  - But CORBA objects can be shared and long-lived
    - business rules and policies decide when to “destroy” an object
    - IORs can outlive client and even server process life spans
- CORBA objects can be relocated
  - The fixed object key of an object reference does not include the object’s location
  - CORBA objects may be relocated at admin time or runtime
  - ORB implementations may support the relocation transparently
- CORBA supports replicated objects
  - IORs with the same object key but different locations are considered replicas

CORBA server characteristics

- When we say “server” we usually mean server process, not server machine
- One or more CORBA server processes may be running on a machine
- Each CORBA server process may contain one or more CORBA object instances, of one or more CORBA interfaces
- A CORBA server process does not have to be “heavyweight”
  - e.g., a Java applet can be a CORBA server

Interfaces vs. Implementations



CORBA Objects are fully encapsulated

Accessed through well-defined interface

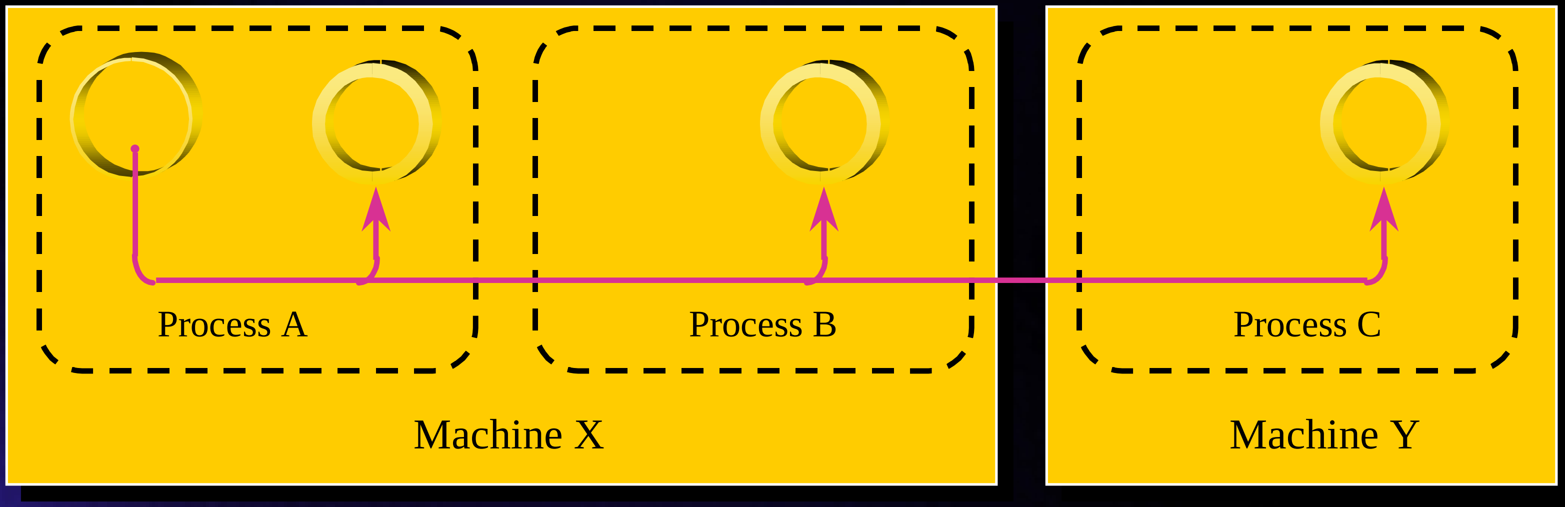
Internals not available - users of object have no knowledge of implementation

Interfaces & Implementations totally separate

For one interface, multiple implementations possible

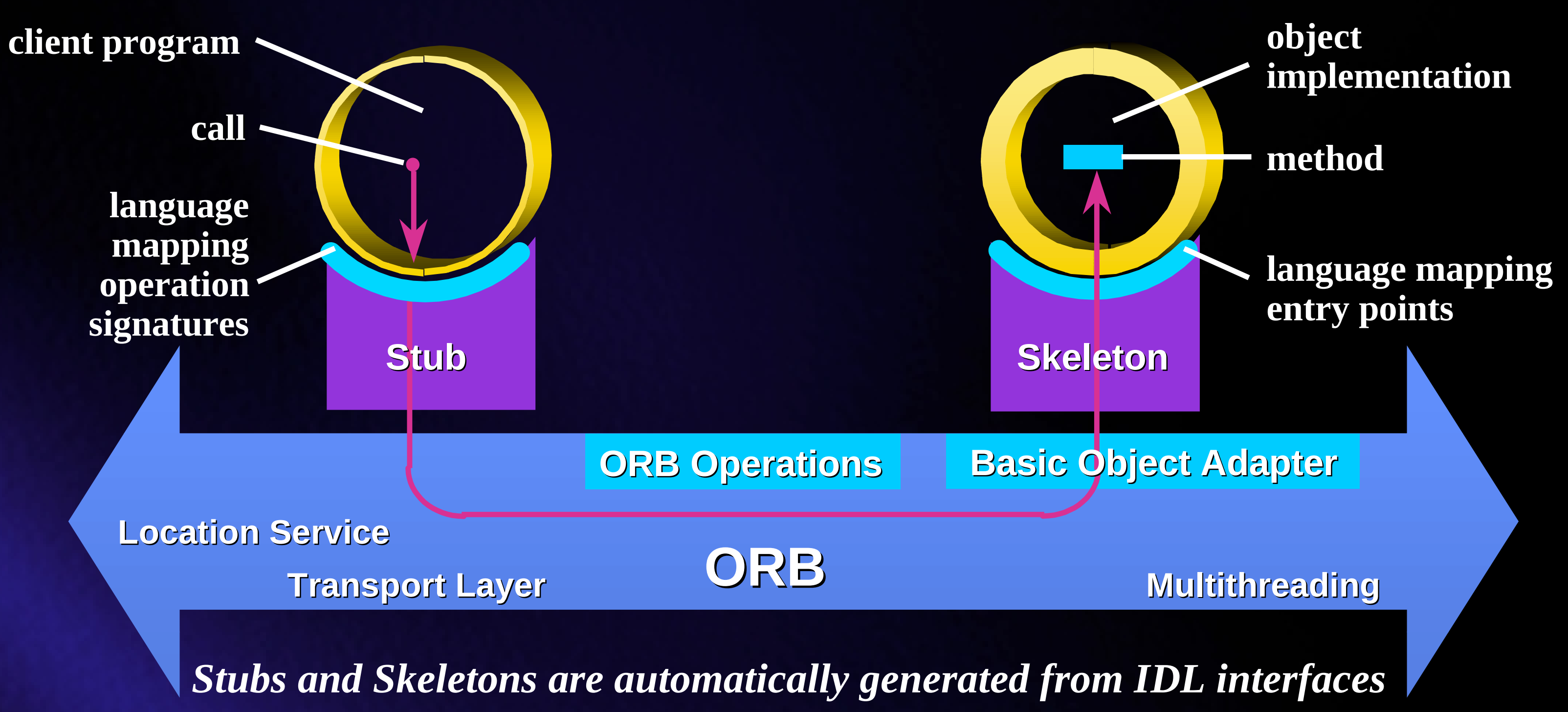
One implementation may be supporting multiple interfaces

Location Transparency



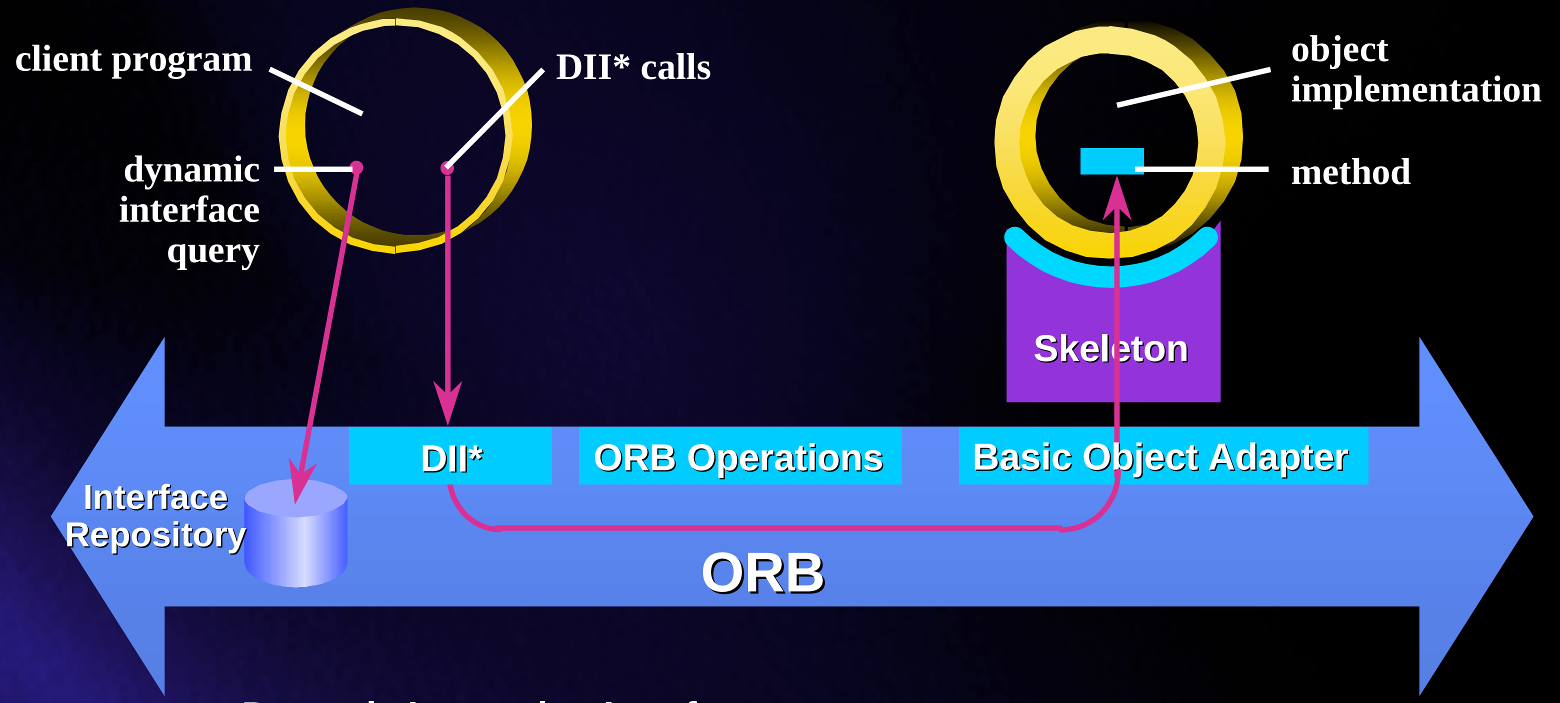
A CORBA Object can be local to your process, in another process on the same machine, or in another process on another machine

Stubs & Skeletons



Stubs and Skeletons are automatically generated from IDL interfaces

Dynamic Invocation Interface



* Dynamic Invocation Interface

Why IDL?

- IDL reconciles diverse object models and programming languages
- Imposes the same object model on all supported languages
- Programming language independent means of describing data types and object interfaces
  - purely descriptive - no procedural components
  - provides abstraction from implementation
  - allows multiple language bindings to be defined
- A means for integrating and sharing objects from different object models and languages

IDL simple data types

- Basic data types similar to C, C++ or Java
  - long, long long, unsigned long, unsigned long long
  - short, unsigned short
  - float, double, long double
  - char, wchar (ISO Unicode)
  - boolean
  - octet (raw data without conversion)
  - any (self-describing variable)

IDL complex data types

- string - sequence of characters - bounded or unbounded
  - string<256> msg   // bounded
  - string msg   // unbounded
- wstring - sequence of Unicode characters - bounded or unbounded
- sequence - one dimensional array whose members are all of the same type - bounded or unbounded
  - sequence<float, 100> mySeq   // bounded
  - sequence<float> mySeq   // unbounded

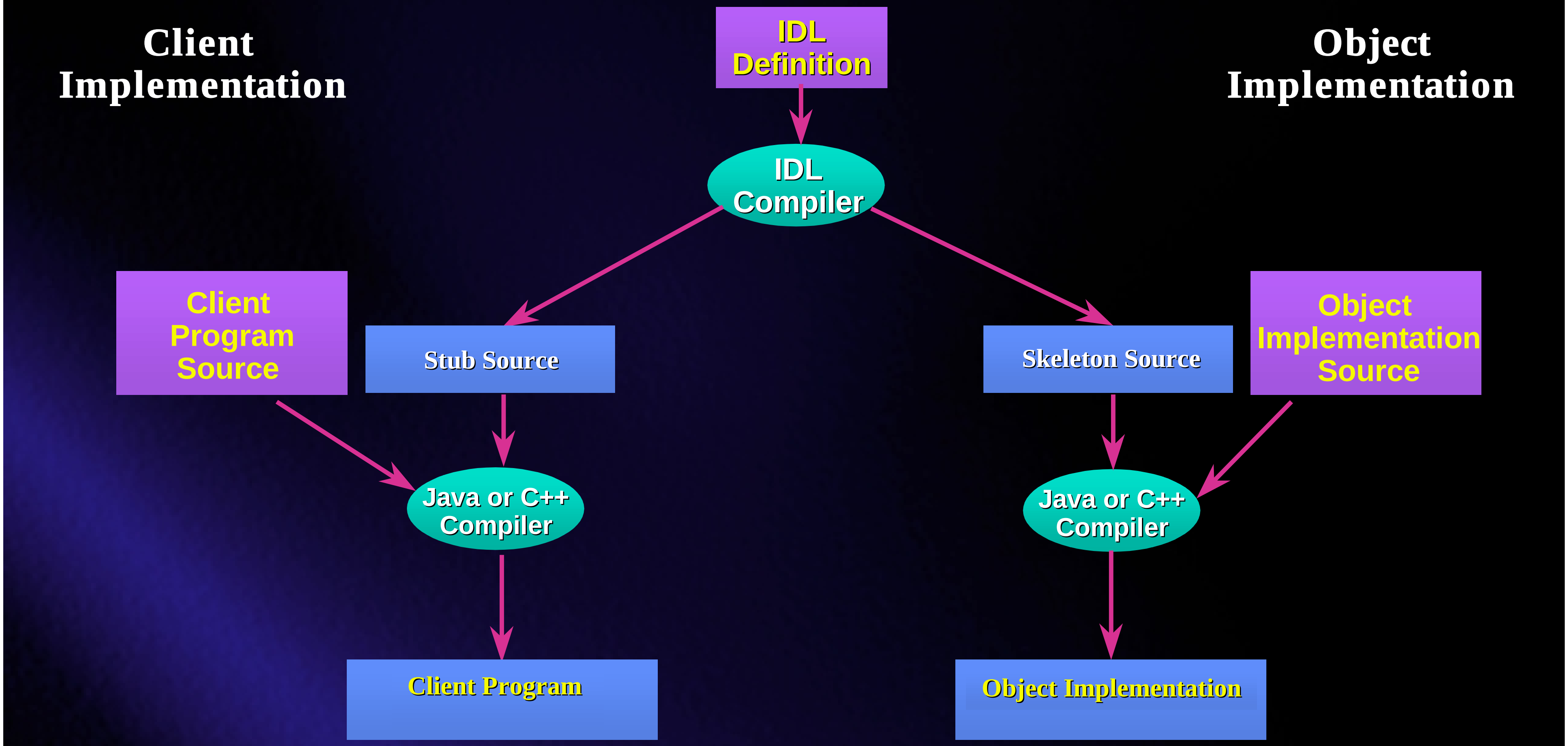
IDL user defined data types

- Facilities for creating your own types:
  - typedef
  - enum
  - const
  - struct
  - union
  - arrays
  - exception
- preprocessor directives - #include #define

Operations and parameters

- Return type of operations can be any IDL type
- each parameter has a direction (in, out, inout) and a name
- similar to C/C++ function declarations

CORBA Development Process Using IDL



A simple example: IDL

```
// module Money
{
    interface Accounting
    {
      float get_outstanding_balance();
    };
};
```

A Java client

```
import org.omg.CORBA.*;
public class Client
{
        public static void main(String args[]) {
                try {
                        // Initialize the ORB.
                        System.out.println("Initializing the ORB...");
                        ORB orb = ORB.init(args, null);
                        // bind to an Accounting Object named "Account"
                        System.out.println("Binding...");
                        Money.Accounting acc =Money.AccountingHelper.bind(orb,"Account");
                        // Get the balance of the account.
                        System.out.println("Making Remote Invocation...");
                        float balance = acc.get_outstanding_balance();
                        // Print out the balance.
                        System.out.println("The balance is $" + balance);
                }
                        catch(SystemException e) {
                          System.err.println("Oops!  Caught: " + e);
                }
        }
}
```

A Java server object

```
import Money.*;
import org.omg.CORBA.*;
class AccountingImpl extends _AccountingImplBase
{
public float get_outstanding_balance()
          {
                    float bal = (float)14100.00; // Implement real outstanding balance function here
                    return bal;
          }
public static void main(String[] args)
          {
 try {
                    ORB orb = ORB.init(args, null); // Initialize the ORB.
                     BOA boa = orb.BOA_init();      // Initialize the BOA.
                    System.out.println("Instantiating an AccountingImpl.");
                    AccountingImpl impl = new AccountingImpl("Account");
                    boa.obj_is_ready(impl);
                    System.out.println("Entering event loop."); // Wait for incoming requests
                    boa.impl_is_ready();
          }
          catch(SystemException e) {
                    System.err.println("Oops!  Caught: " + e);
          }
     }
}
```

A C++ client

```
#include <Money_c.hh>

int main (int argc, char* const* argv)
{

  try {
    cout << "Initializing ORB..." << endl;
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    cout << "Binding..." << endl;
    Money::Accounting_var acc = Money::Accounting::_bind();

    cout << "Making Remote Invocation..." << endl;
    cout << "The outstanding balance is "
      << acc->get_outstanding_balance()
                  << endl;
  }
  catch (CORBA::Exception& e) {
    cerr << "Caught CORBA Exception: " << e << endl;
  }
  return 0;
}
```

A C++ server object

```
#include <Money_s.hh>
class AccountingImpl : public _sk_Money::_sk_Accounting
{
public:
  AccountingImpl(const char* name) : _sk_Accounting(name) {}
  CORBA::Float get_outstanding_balance()
  {
    // implement real outstanding balance function here
    return 3829.29;
  }
};

int main (int argc, char* const* argv)
{
  // Initialize ORB.
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
  CORBA::BOA_var boa = orb->BOA_init(argc, argv);
  cout << "Instantiating an AccountingImpl" << endl;
  AccountingImpl impl("Accounting");
  boa->obj_is_ready(&impl);
  cout << "Entering event loop" << endl;
  boa->impl_is_ready();
  return 0;
}
```

CORBA services

- The OMG has defined a set of Common Object Services
- Frequently used components needed for building robust applications
- Typically supplied by vendors
- OMG defines interfaces to services to ensure interoperability

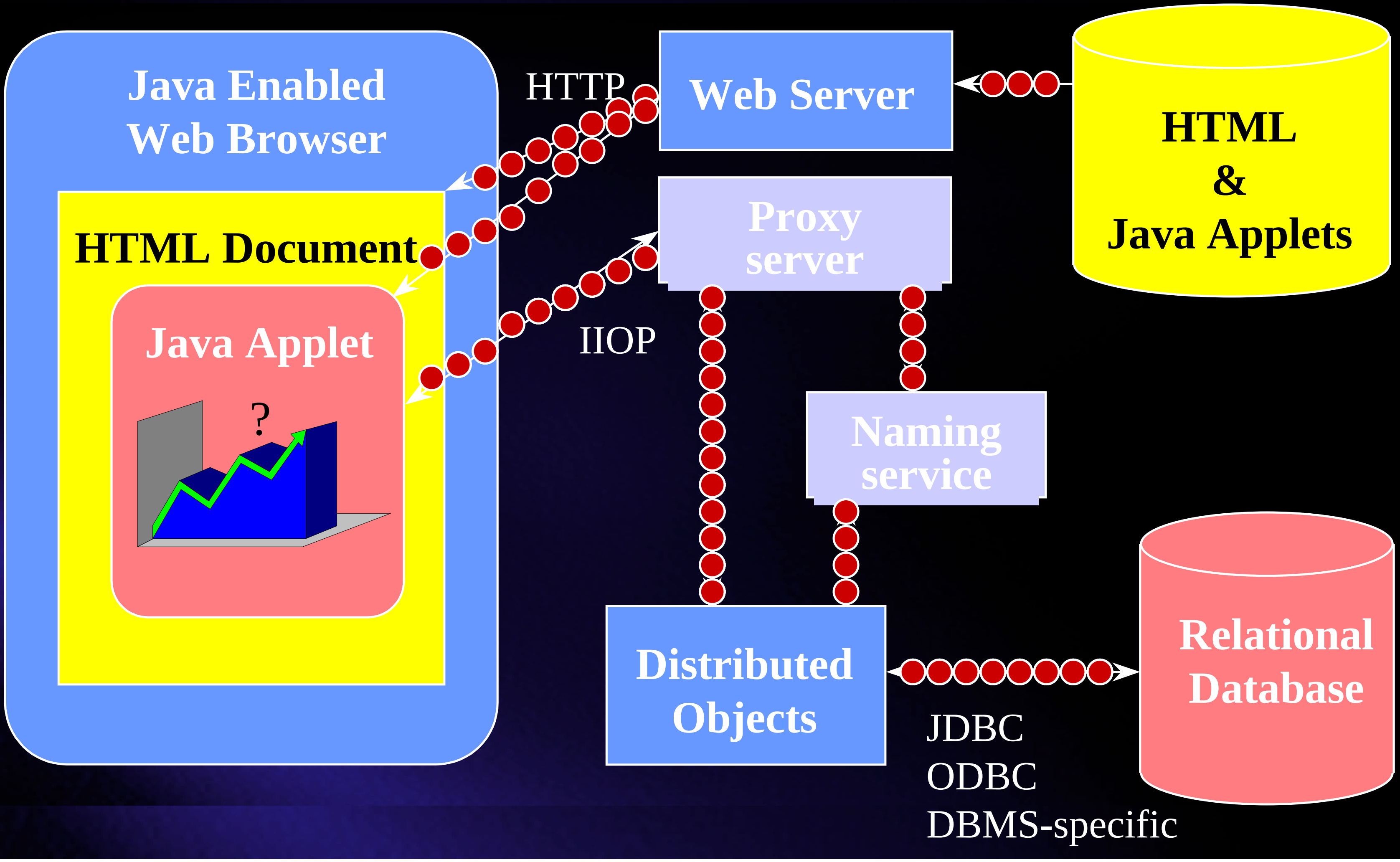
Popular CORBA services

- Naming
  - maps logical names to to server objects
  - references may be hierarchical, chained
  - returns object reference to requesting client
- Events
  - asynchronous messaging
  - decouples suppliers and consumers of information

Popular CORBA services

- Notification
  - More robust enhancement of event service
  - Quality of Service properties
  - Event filtering
  - Structured events
- Transaction
  - Ensures correct state of transactional objects
    - Manages distributed commit/rollback
    - Implements the protocols required to guarantee the ACID (Atomicity, Consistency, Isolation, and Durability) properties of transactions

CORBA Internet Access via IIOP



The future: CORBA 3

- Spec is complete. Final adoption due in November.
- Internet related features:
- Standard for callbacks through firewalls
  - currently not allowed by most firewalls, proprietary
- Interoperable naming service
  - standard bootstrapping mechanism to find naming services
  - iiopploc://www.myserver.com/mynamingservice

CORBA 3

- Quality of service enhancements
  - Asynchronous Messaging
    - invocation result retrieval by polling or callback
  - Quality of Service Control
    - Clients and objects may control ordering (by time, priority, or deadline); set priority, deadlines, and time-to-live
    - set a start time and end time for time-sensitive invocations
    - control routing policy and network routing hop count

CORBA 3

- Minimum, Fault-Tolerant, and Real-Time CORBA
  - minimum CORBA - for embedded systems
    - strips out unnecessary pieces - dynamic invocation, etc.
  - Real-time CORBA
    - standardizes resource control - threads, protocols, connections
    - uses priority models to achieve predictable behavior for both hard and statistical realtime environments
  - Fault-tolerant CORBA
    - entity redundancy and fault management control
    - spec is still in process

CORBA 3

- CORBA Component Model (CCM)
  - Spec approved on September 2, 1999
  - Support for Java, COBOL, Microsoft COM/DCOM, C++, Ada, C and Smalltalk
  - Container environment that is persistent, transactional, and secure
  - Containers will provides interface and event resolution
  - Integration/interoperability with Enterprise JavaBeans (EJBs)

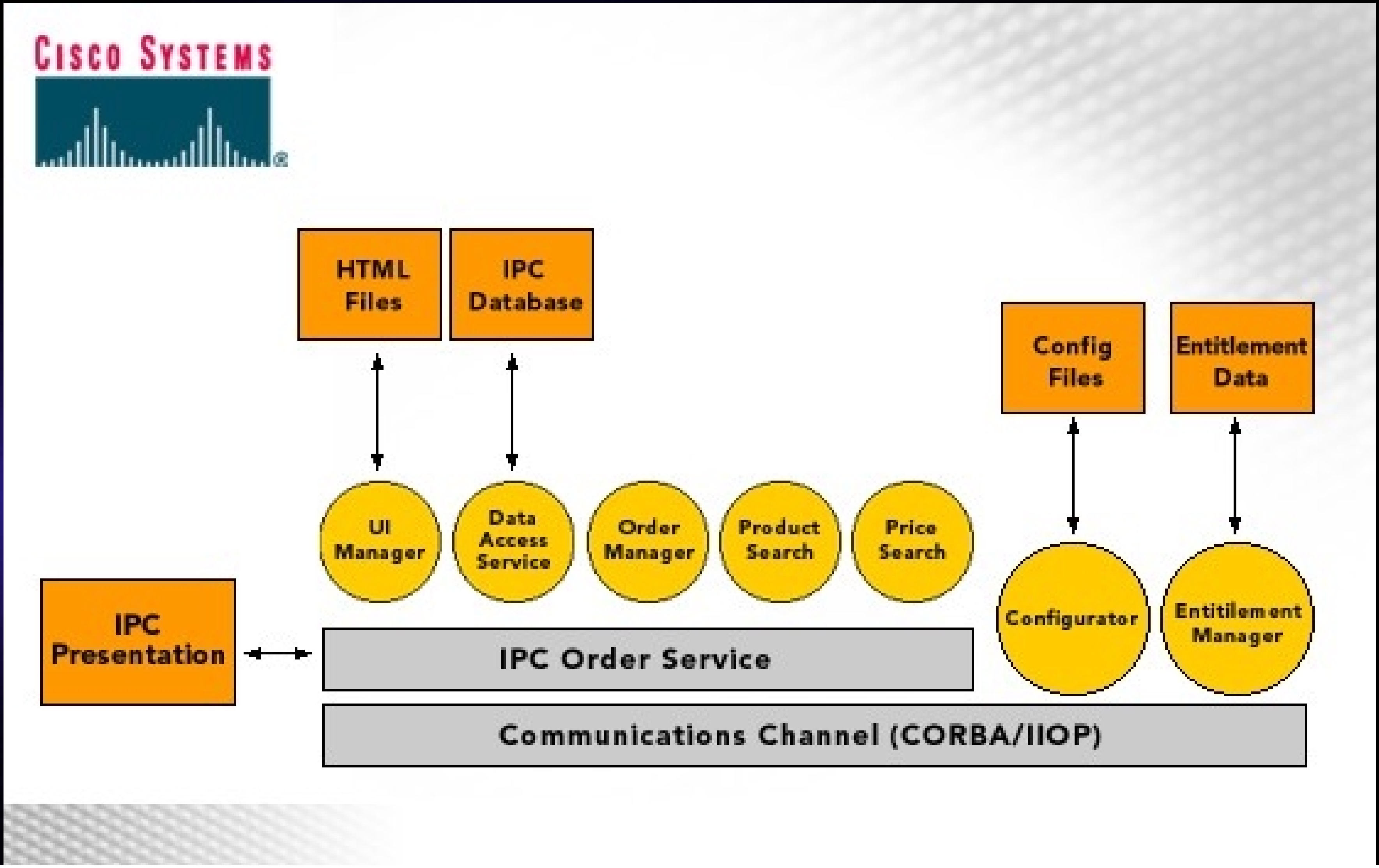
CORBA vendors

- Inprise/Borland VisiBroker:
  - http://www.borland.com/visibroker/
- Iona Orbix:
  - http://www.iona.com
- Rogue Wave Nouveau:
  - http://www.roguewave.com/products/nouveau/
- ObjectSpace Voyager:
  - http://www.objectspace.com/products/vgrOverview.htm

Real-world implementations

- Commercial products
  - Oracle8i
  - SilverStream Application Server
  - BEA WebLogic Server
  - Vitria BusinessWare enterprise integration server
  - Evergreen Ecential ecommerce engine
  - enCommerce getAccess security server
- End-user applications:
  - http://www.borland.com/visibroker/cases/
  - http://www.iona.com/info/aboutus/customers/index.html

Example: Cysive - Cisco Internetworking Products Center



Example: Cisco IPC

- Server-side Java system
  - Provides extreme scalability and greatly accelerated performance
    - allows IPC to share data and system resources across multiple transactions
    - maintains continuous server connections throughout long, complex transactions
    - process many more orders in a shorter period of time

Example: Cisco IPC

- Significant improvement of extensibility
  - Built on an object-oriented foundation, providing a modular infrastructure
  - New features can be added
  - Back-end applications, such as Oracle Financials, can be linked to IPC quite easily
  - System offers greater availability than the earlier version, requiring almost no downtime—planned or unplanned—as capabilities are added

Resources: Web

- Web sites:
  - OMG: http://www.omg.org/
  - Washington University: http://www.cs.wustl.edu/~schmidt
  - Free CORBA page
    - http://adams.patriot.net/~tvalesky/freecorba.html
  - Cetus links (links to CORBA vendors, benchmarks, etc.):
    - http://www.cetus-links.org/oo_object_request_brokers.htm
- Newsgroups:
  - comp.object.corba
  - comp.lang.java.corba

Resources: books

- Client/Server Programming With Java and CORBA (2nd edition)
  - by Robert Orfali and Dan Harkey
- Programming with VisiBroker, A Developer's Guide to VisiBroker for Java
  - by Doug Pedrick, Jonathan Weedon, Jon Goldberg, and Erik Bleifield
- Advanced CORBA Programming with C++
  - by Michi Henning and Steve Vinoski