

# Distributed Systems

## 9. Remote Procedure Calls: Case Studies

Paul Krzyzanowski

[pxk@cs.rutgers.edu](mailto:pxk@cs.rutgers.edu)

# Overview of RPC Systems

1. Remote Procedure Calls
2. Remote Objects
3. Web Services
4. Just Marshalling

# Sun (ONC) RPC

# Sun (ONC) RPC

---

- RPC for Unix System V, Linux, BSD, OS X
  - Also known as ONC RPC  
(Open Network Computing)
- Interfaces defined in an **Interface Definition Language (IDL)**
  - IDL compiler is *rpcgen*

# RPC IDL

---

Interface definition

**version 1**

Interface definition

**version 2**

# RPC IDL

---

```
program GETNAME {  
    version GET_VERS {  
        long GET_ID(string<50>) = 1;  
        string GET_ADDR(long) = 2;  
    } = 1;    /* version */  
} = 0x31223456;
```

# rpcgen

---

```
rpcgen name.x
```

produces:

- name.h            header
  - name\_svc.c        server stub (skeleton)
  - name\_clnt.c       client stub
  - [ name\_xdr.c ]    XDR conversion routines
- 
- Function names derived from IDL function names and version numbers
  - Client gets *pointer* to result
    - Allows it to identify failed RPC (null return)
    - Reminder: C doesn't have exceptions!

# What goes on in the system: server

---

## Start server

- Server stub creates a socket and binds any available local port to it
- Calls a function in the RPC library:
  - *svc\_register* to register program#, port #
  - contacts *portmap* (*rpcbind* on SVR4):
    - Name server
    - Keeps track of  
{program #, version #, protocol} → port # bindings
- Server then listens and waits to accept connections

# What goes on in the system: client

---

- Client calls **clnt\_create** with:
  - Name of server
  - Program #
  - Version #
  - Protocol#
- *clnt\_create* contacts port mapper on that server to get the port for that interface
  - **early binding** – done once, not per procedure call

# Advantages

---

- Don't worry about getting a unique transport address (port)
  - But with SUN RPC you need a unique program number per server
  - Greater portability
- Transport independent
  - Protocol can be selected at run-time
- Application does not have to deal with maintaining message boundaries, fragmentation, reassembly
- Applications need to know only one transport address
  - Port mapper (portmap process)
- Function call model can be used instead of send/receive

# DCE RPC

# DCE RPC

---

- **DCE**: set of components designed by The Open Group (merger of OSF and X/Open) for providing support for distributed applications
  - Distributed file system service, time service, directory service, ...
- Room for improvement in Sun RPC

# DCE RPC

---

- Similar to Sun's RPC
- Interfaces written in an **Interface Definition Notation (IDN)**
  - Definitions look like function prototypes
- Run-time libraries
  - One for TCP/IP and one for UDP/IP
- Authenticated RPC support with DCE security services
- Integration with DCE directory services to locate servers

# Unique IDs

---

Sun RPC required a programmer to pick a “unique” 32-bit number

DCE: get unique ID with **uuidgen**

- Generates prototype IDN file with a 128-bit Unique Universal ID (UUID)
- 10-byte timestamp multiplexed with version number
- 6-byte node identifier (ethernet address on ethernet systems)

# IDN compiler

---

Similar to rpcgen:

Generates header, client, and server stubs

# Service lookup

---

Sun RPC requires client to know name of server

DCE allows several machines to be organized into an administrative entity

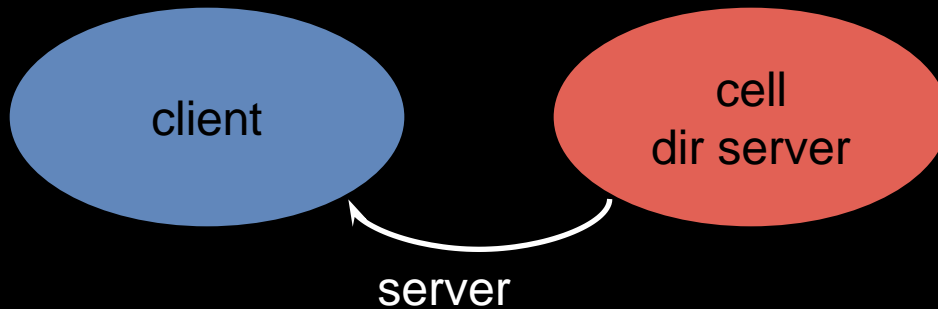
**cell** (collection of machines, files, users)

## Cell directory server

Each machine communicates with it for cell services information

# DCE service lookup

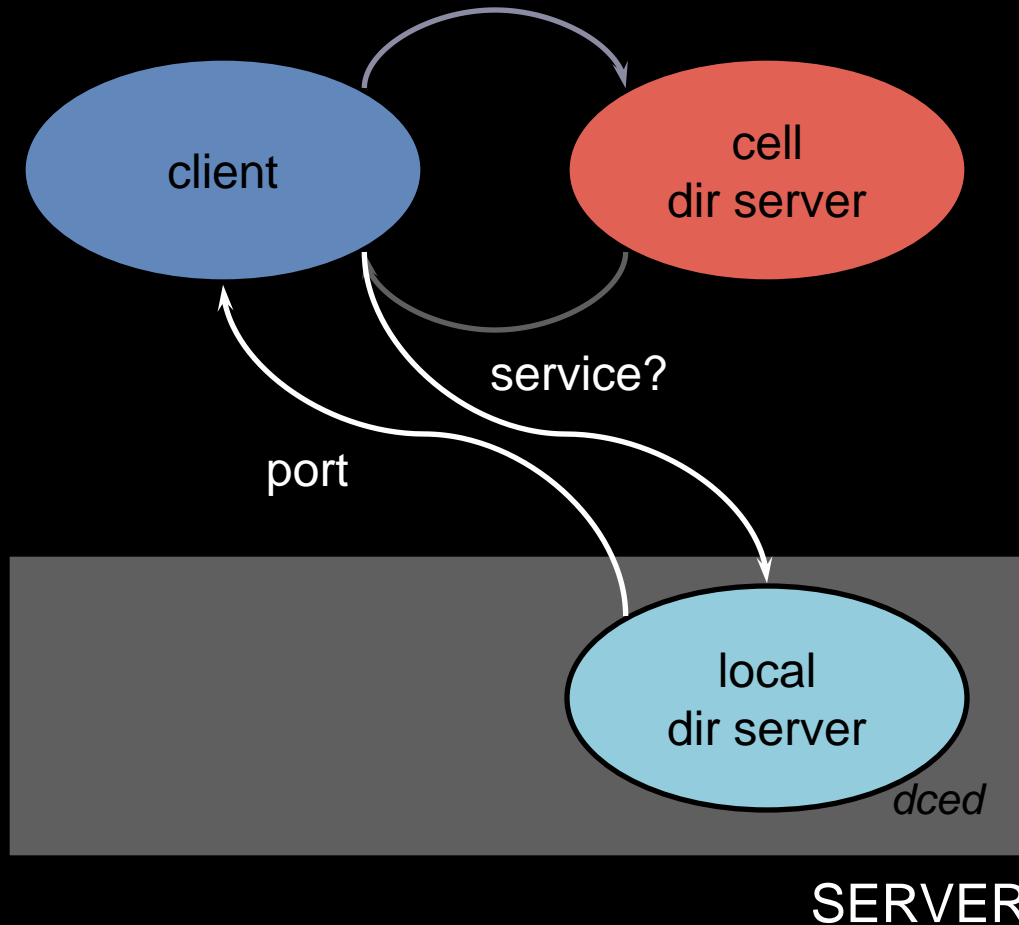
---



Request service  
lookup from cell  
directory server

Return server machine  
name

# DCE service lookup

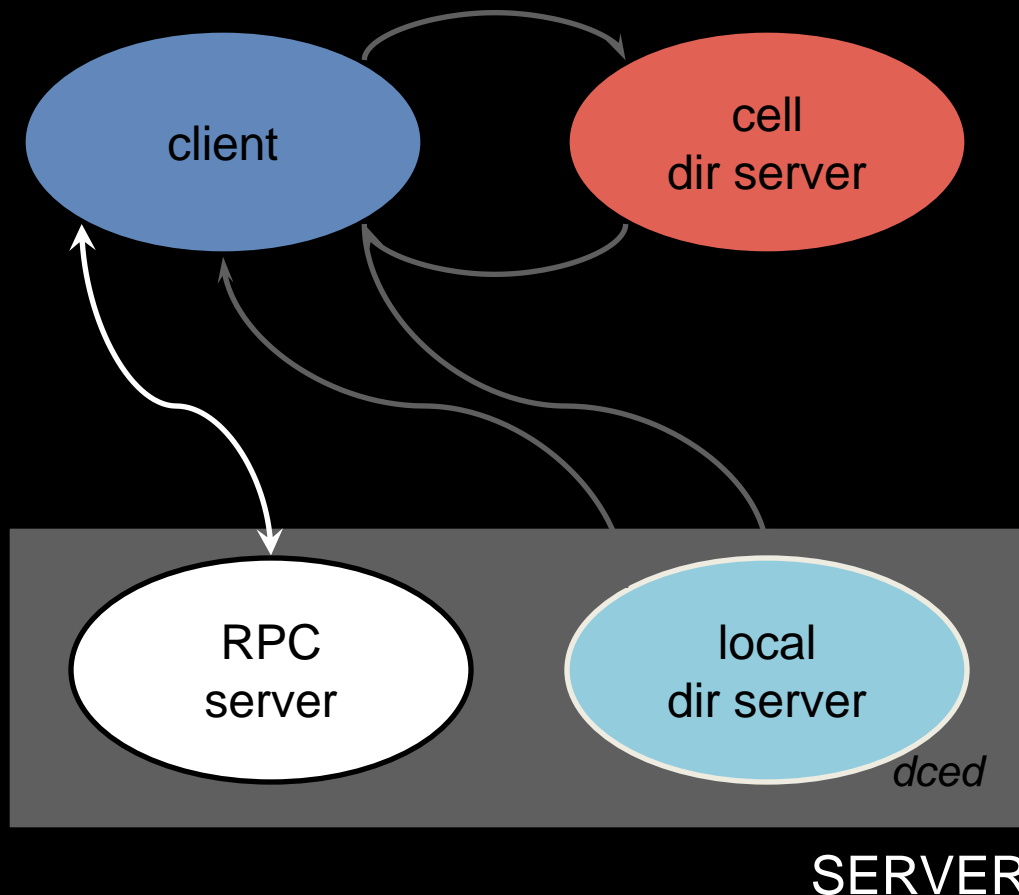


Connect to endpoint mapper service and get port binding from this local name server

# DCE service lookup

---

Connect to service and  
request remote  
procedure execution



# Marshalling

---

## Standard formats for data

- NDR: Network Data Representation

## Goal

- *Multi-canonical* approach to data conversion
  - Fixed set of alternate representations
  - Byte order, character sets, and floating-point representation can assume one of several forms
  - Sender can (hopefully) use native format
  - Receiver may have to convert

# What's Cool

---

- DCE RPC improved Sun RPC
  - Unique Universal ID
  - Multi-canonical marshalling format
  - *Cell* of machines with a cell directory server
    - No need to know which machine provides a service

# Sun and DCE RPC deficiencies

---

- If **server is not running**
  - Service cannot be accessed
  - Administrator responsible for starting it
- If a **new service is added**
  - There is no mechanism for a client to discover this
- Object oriented languages expect **polymorphism**
  - Service may behave differently based on data types passed to it

# The next generation of RPCs

Distributed objects:  
support for object oriented languages

# CORBA

# CORBA

---

## Common Object Request Architecture

- Evolving since 1989

Standard architecture for distributing objects

Defined by OMG (Object Management Group)

- Consortium of 347 companies\*

Goal: provide support for distributed, heterogeneous object-oriented applications

- Specification is independent of any language, OS, network

\*<http://www.omg.org/cgi-bin/apps/membersearch.pl>

# CORBA

---

## Basic paradigm:

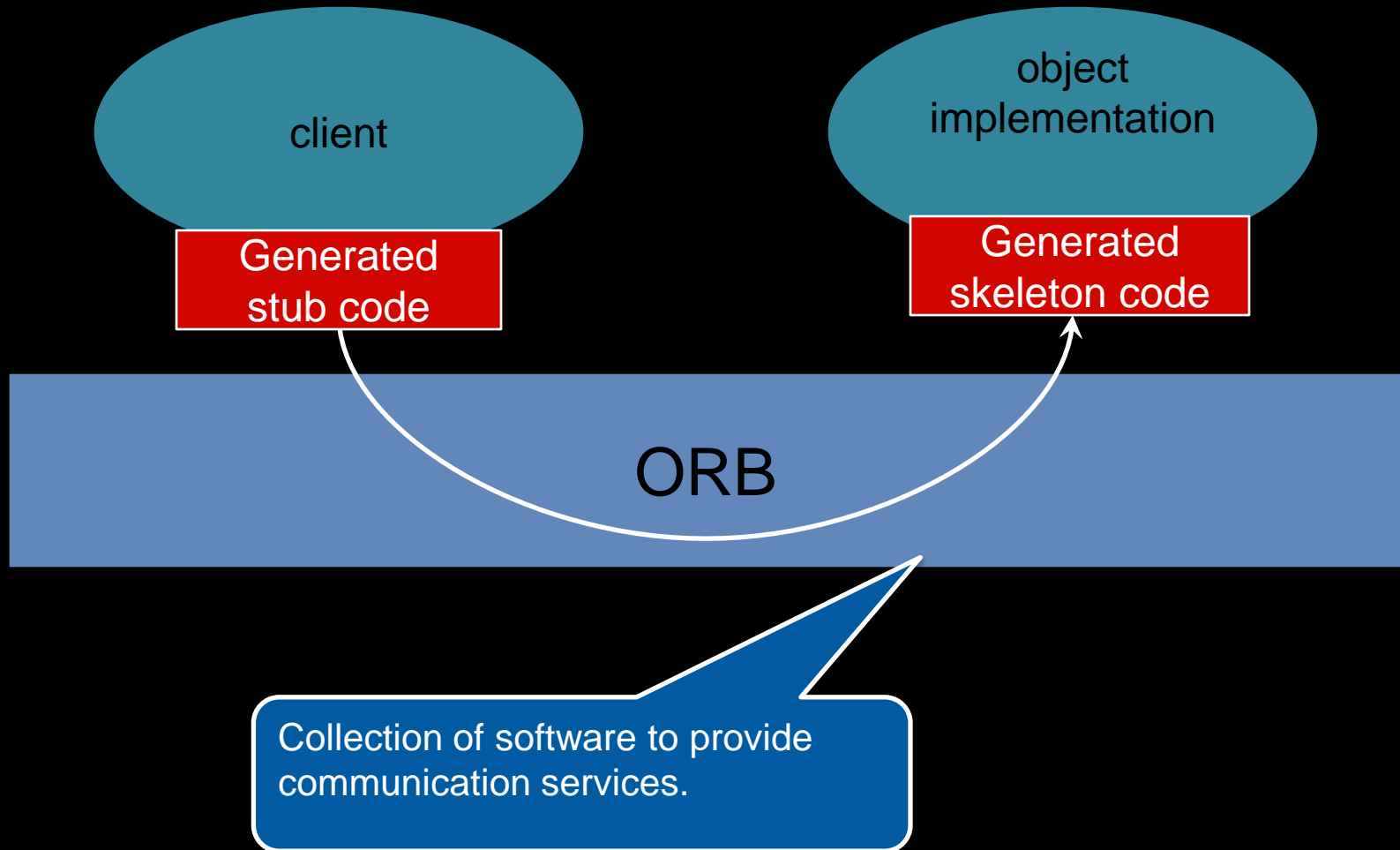
- Request services of a distributed object
- Interfaces are defined in an IDL
- Distributed objects are identified by object reference

## Object Request Broker (ORB)

- delivers request to the object and returns results to the client
- set of code that implements RPC

# CORBA logical view

---



# Object Request Broker (ORB)

---

Distributed service that implements the request to the remote object

- Locates the remote object on the network
- Communicates request to the object
- Waits for results
- Communicates results back to the client

Responsible for providing location transparency

- Same request mechanism used by client & CORBA object regardless of object location

Client request may be written in a different programming language than the implementation

# ORB functions

---

- Look up and instantiate objects on remote machines
- Marshal parameters
- Deal with security
- Publish data on objects for other ORBs to use
- Invoke methods on remote objects
  - Static or dynamic execution
- Automatically instantiate objects that aren't running
- Route callback methods
- Communicate with other ORBs

# IDL (Interface Definition Language)

---

- Indicates operations an object supports
  - Not *how* they are implemented
- Programming language neutral
  - Currently standardized language bindings for C, C++, Java, Perl, Python, Ada, COBOL, Smalltalk, Objective C, LISP
- IDL data types
  - Basic types: long, short, string, float, ...
  - Constructed types: struct, union, enum, sequence
  - Typed object references
  - The *any* type: a dynamically typed value

# IDL example

---

```
Module StudentObject {
    struct StudentInfo {
        string name;
        int id;
        float gpa;
    };
    exception Unknown {};
    interface Student {
        StudentInfo getinfo(in string name)
            raises(Unknown);
        void putinfo(in StudentInfo data);
    };
};
```

# CORBA IDL

---

## Compiled with IDL compiler

- Converted to target language
- Generates stub functions

# Objects

---

- Object references persist
  - They can be saved as a string
  - ... and be recreated from a string
- Client
  - Performs requests by having an **object reference** for object & desired operation
  - Client initiates request by
    - calling **stub** routines specific to an object
    - Or constructing request dynamically (**DII** interface)
- Server (object implementation)
  - Provides semantics of objects
  - Defines data for instance, code for methods

# Interoperability

---

- CORBA clients are portable
  - They conform to the API ... but may need recompilation
- Object implementations (servers)
  - generally need some rework to move from one vendor's CORBA product to another
- 1996: CORBA 2.0 added *interoperability* as a goal in the specification
  - Define network protocol called **IIOIP**
    - Inter-ORB Protocol
  - IIOIP works across any TCP/IP implementations

# IIOP

---

IIOP can be used in systems that do not even provide a CORBA API

- Used as transport for version of Java RMI (RMI over IIOP)
- Various application servers use IIOP but do not expose the CORBA API
- Programs written to different APIs can interoperate with each other and with programs written to the CORBA API

# CORBA Services (COS)

---

Set of distributed services to support the integration and interoperation of distributed objects

Defined on top of ORB

- Standard CORBA objects with IDL interfaces

# Popular services

---

- Object life cycle
  - Defines how CORBA objects are created, moved, removed, copied
- Naming
  - Defines how objects can have friendly symbolic names
- Events
  - Asynchronous communication
- Externalization
  - Coordinates the transformation of objects to/from external media

# Popular services

---

- Transactions
  - Provides atomic access to objects
- Concurrency control
  - Locking service for serializable access
- Property
  - Manage name-value pair namespace
- Trader
  - Find objects based on properties and describing service offered by object
- Query
  - Queries on objects

# CORBA vendors

---

- Lots of vendors
  - ORBit
    - Bindings for C, Perl, C++, Lisp, Pascal, Python, Ruby, and TCL (designed for GNOME)
  - Java ORB
    - Part of Java SDK
  - VisiBroker for Java
    - From Imprise; embedded in Netscape Communicator
  - OrbixWeb
    - From Iona Technologies
  - Websphere
    - From IBM
  - Many, many others

# Assessment

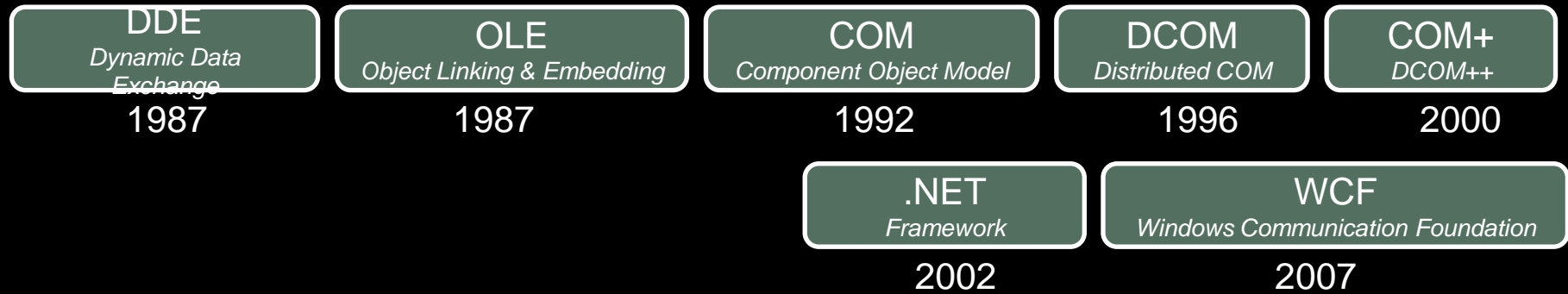
---

- Reliable, comprehensive support for managing services
- Standardized
- Complex
  - Steep learning curve
  - Integration with languages not always straightforward
- Pools of adoption
- Late to ride the Internet bandwagon

# Microsoft DCOM

# Microsoft DCOM/COM+

---



## COM+: Windows 2000

- Unified COM and DCOM plus support for transactions, resource pooling, publish-subscribe communication

Extends Component Object Model (COM) to allow objects to communicate between machines

# Activation on server

---

## Service Control Manager (SCM)

- Started at system boot. Functions as RPC server
- Maintains database of installed services
- Starts services on system startup or on demand
- Requests creation of object on server

## Surrogate process runs components: dllhost.exe

- Process that loads DLL-based COM objects

Can handle multiple clients simultaneously

# Beneath COM+

---

## Data transfer and function invocation

- Object RPC (**ORPC**)
- Extension of the **DCE RPC** protocol

### Standard DCE RPC packets plus:

- **Interface pointer identifier (IPID)**
  - Identifies interface and object where the call will be processed
  - Referrals: can pass remote object references
- Versioning & extensibility information

# Marshalling

---

- Marshalling mechanism: **NDR**  
Network Data Representation of DCE RPC
  - One new data type added: represents a marshaled interface
    - Allows one to pass interfaces to objects

# MIDL

---

MIDL files are compiled with an IDL compiler  
DCE IDL + object definitions

Generates C++ code for marshalling and unmarshalling

- Client side is called the *proxy*
- Server side is called the *stub*

*both are COM objects that are loaded  
by the COM libraries as needed*

# COM+ Distributed Garbage Collection

---

Object lifetime controlled by **remote reference counting**

- *RemAddRef*, *RemRelease* calls
- Object elided when reference count = 0

# COM+ Distributed Garbage Collection

---

## Abnormal client termination

- No message to decrement reference count set to server

## In addition to reference counting:

### Pinging

- Server has *pingPeriod*, *numPingsToTimeOut*
- Relies on client to ping
  - background process sends ping set – IDs of all remote objects on server
- If ping period expires with no pings received, all references are cleared

# Microsoft DCOM/COM+ Contributions

---

- Designed to address the threat of CORBA
- Fits into Microsoft COM model
- Generic server hosts dynamically loaded objects
  - Requires unloading objects (dealing with dead clients)
  - Reference counting and pinging
- Support for references to instantiated objects
- But... COM+ is a Microsoft-only solution
  - And it doesn't work well across firewalls because of dynamic ports

# Java RMI

# Java RMI

---

- Java language had no mechanism for invoking remote methods
- 1995: Sun added extension
  - Remote Method Invocation (RMI)
  - Allow programmer to create distributed applications where methods of remote objects can be invoked from other JVMs

# RMI components

---

## Client

- Invokes method on remote object

## Server

- Process that owns the remote object

## Object registry

- Name server that relates objects with names

# Interoperability

---

## RMI is built for Java only!

- No goal of OS interoperability (as CORBA)
- No language interoperability  
(goals of SUN, DCE, and CORBA)
- No architecture interoperability

## No need for external data representation

- All sides run a JVM

## Benefit: simple and clean design

# RMI similarities

---

## Similar to local objects

- References to remote objects can be passed as parameters (not really)
- Objects can be passed as parameters to remote methods (but not as a reference)
- Object can be cast to any of the set of interfaces supported by the implementation
  - Operations can be invoked on these objects

# RMI differences

---

- Non-remote arguments/results passed to/from a remote method by copy
- Remote object passed by reference, not by copying remote implementation
- Extra exceptions

# New classes

---

- **remote class:**
  - One whose instances can be used remotely
  - Within its address space: regular object
  - Other address spaces: can be referenced with an **object handle**
- **serializable class:**
  - Object that can be marshaled
  - If object is passed as parameter or return value of a remote method invocation, the value will be copied from one address space to another
    - If remote object is passed, only the object handle is copied between address spaces

# New classes

---

- **remote class:**

- One whose instances can be used remotely
  - Within its address space: regular object
  - Other address spaces: can be referenced with an object handle
- needed for remote objects**

- **serializable class:**

- Object that can be marshaled
  - If object is passed as parameter or return value of a remote method invocation, the value will be copied from one address space to another
  - If remote object is passed, only the object handle is copied between address spaces
- needed for parameters**

# Stubs

---

Generated by separate compiler

## rmic

- Produces Stubs and skeletons for the remote interfaces are generated (class files)

# Naming service

---

Need a remote object reference to perform remote object invocations

Object registry does this: `rmiregistry`

# Server

---

## Register object(s) with Object Registry

```
Stuff obj = new Stuff();  
Naming.bind("MyStuff", obj);
```

# Client

---

Contact *rmiregistry* to look up name

```
MyInterface test = (MyInterface)
    Naming.lookup("rmi://www.pk.org/MyStuff");
```

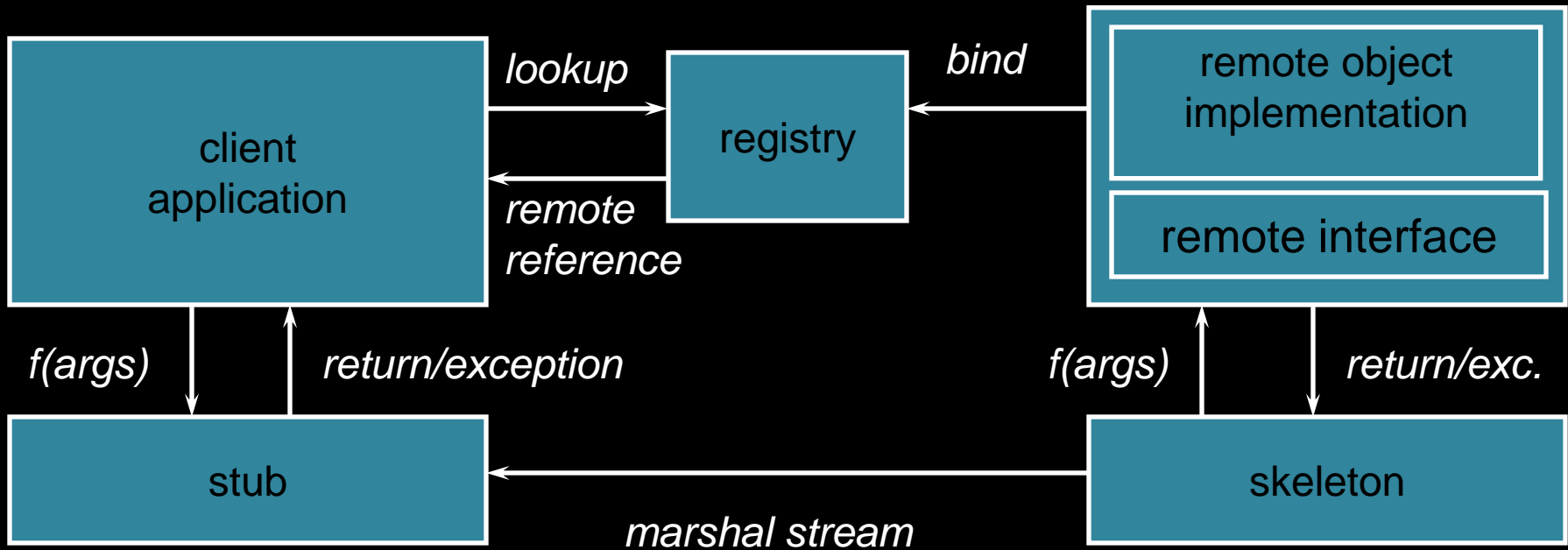
*rmiregistry* service returns a remote object reference.

*lookup* method gives reference to local stub.

Invoke remote method(s):

```
test.func(1, 2, "hi");
```

# Java RMI infrastructure



# RMI Distributed Garbage Collection

---

- Two operations: *dirty* and *clean*
- Local JVM sends a *dirty* call to the server JVM when the object is in use
  - The *dirty* call is refreshed based on the lease time given by the server
- Local JVM sends a *clean* call when there are no more local references to the object
- Unlike DCOM:  
no incrementing/decrementing of references

# The third generation of RPCs

Web services  
and  
Riding the XML Bandwagon



# From Web Browsing to Web Services

---

- Web browser:
  - Dominant model for user interaction on the Internet
- Not good for programmatic access to data or manipulating data
  - UI is a major component of the content

# Web Services

---

- We want
  - Remotely hosted services – that *programs* can use
- Problems
  - Web pages are content-focused
  - Traditional RPC solutions usually used a range of ports
    - And we need more than just RPC sometimes
  - Firewalls restrict ports & inspect the protocol

# Web Services

---

- Web Services
  - Set of protocols by which services can be published, discovered, and used in a technology neutral form
  - Applications will typically invoke multiple remote services
- Service Oriented Architecture (SOA)
  - Policies, business practices, and frameworks added on top of web services
  - Refers to an architectural design pattern, not a technology.

# XML RPC

# Origins

---

- Early 1998
- Data marshaled into XML messages
  - All request and responses are human-readable XML
- Explicit typing
- Transport over HTTP protocol
  - Solves firewall issues
- No true IDL compiler support for most languages
  - Great support in python and perl (and a few others)
  - Lots of support libraries for other languages

# XML-RPC example

---

```
<methodCall>
  <methodName>
    sample.sumAndDifference
  </methodName>
  <params>
    <param><value><int> 5 </int></value></param>
    <param><value><int> 3 </int></value></param>
  </params>
</methodCall>
```

# XML-RPC data types

---

- int
- string
- boolean
- double
- dateTime.iso8601
- base64
- array
- struct

# Assessment

---

- Simple (spec about 7 pages)
- Humble goals
- Good language support
  - Little/no function call transparency for some languages
- No garbage collection, remote object references, etc.
  - Focus is on data messaging over HTTP transport
- Little industry support
  - Mostly grassroots and open source

SOAP

# SOAP origins

---

## (Simple) (Object) Access Protocol

- 1998 and evolving (latest: v1.2 Jan 2007)
- Started with strong Microsoft & IBM support
- Specifies XML format for messaging
  - Not necessarily RPC
- Continues where XML-RPC left off:
  - XML-RPC is a 1998 simplified subset of SOAP
  - user defined data types
  - ability to specify the recipient
  - message specific processing control
  - and more ...
- Usually XML over HTTP

# SOAP

---

- Stateless messaging model
- Basic facility is used to build other interaction models
  - Request-response
  - Request-multiple response
- Marshalling and unmarshalling to SOAP-format XML
- Like XML-RPC, SOAP is about a messaging format
  - No garbage collection or object references
  - Does not define transport
  - Does not define stub generation

# From Messaging to Web Services

---

- Things like SOAP give us a messaging structure
- What else is useful for services?
  - Service definition: create software to create the right SOAP messages
  - Service discovery
  - Message delivery

# Web Services and WSDL

---

- Web Services Description Language
  - Analogous to an IDL
- Describe an organization's web services
  - Goal is that organizations will exchange WSDL documents
  - If you get WSDL document, you can feed it to a program that will generate software to send and receive SOAP messages

# WSDL Document Contents

## Service

*Name & address*

### Port 1

*Methods for accessing the service*

*Example ports: SOAP, HTTP GET, HTTP POST*

### Binding

*Describes the data format & protocol for a port*

*E.g., RPC style*

### PortType 1

Operations that the service can implement

Messages: definition of the input and output message for each operation

Port 2 ...

# WSDL Structure

---

<definitions>

<types>

data type used by web service: defined via XML Schema syntax

</types>

<message>

describes data elements of operations: parameters

</message>

<portType>

describes service: operations, and messages involved

</portType>

<binding>

defines message format & protocol details for each port

</binding>

</definitions>

# WSDL structure: port types

```
<definitions name="MobilePhoneService" target=...>
```

## 1. type definitions

```
<portType name="MobilePhoneService_port">
```

```
<operation name="getListOfModels">
```

```
<output message="ListOfPhoneModels" />
```

```
<operation name="getPrice">
```

```
<Input message="PhoneModel" />
```

```
<output message="PhoneModelPrice" />
```

## 3. messaging spec

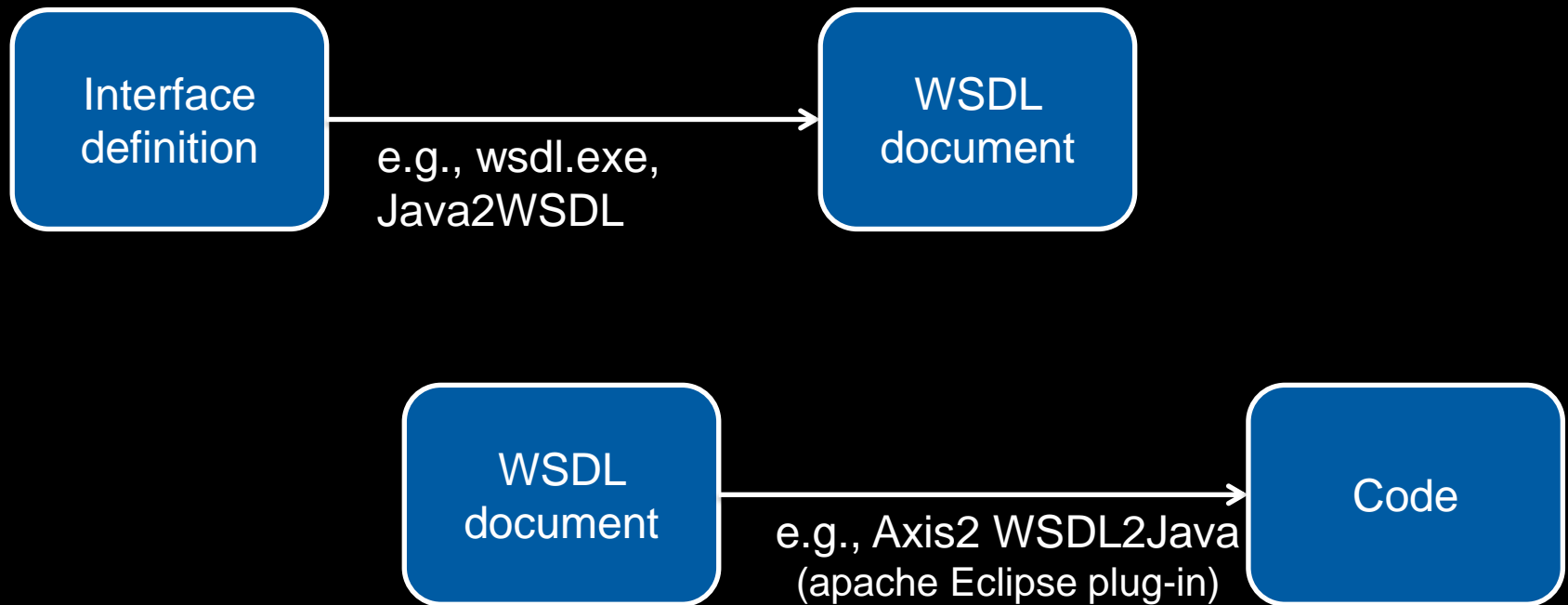
# WSDL part 3: messaging spec

```
<binding name="MobilePhoneService_Binding"
  type="MobilePhoneService_port">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getPrice">
    <soap:operation soapAction="urn:MobilePhoneService"/>
    <input>
      <soap:body encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:MobilePhoneService" use="encoded"/>
    </input>
    <output>
      <soap:body encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:MobilePhoneService" use="encoded" />
    </output>
  </operation>
</binding>
```

# What do we do with WSDL

---

- Not meant for human consumption



# Microsoft .NET Remoting

# Problems with COM+/DCOM

---

- Originally designed for object linking and embedding
- Relatively low-level implementation
- Objects had to provide reference counting explicitly
- Languages & libraries provided varying levels of support
  - A lot for VB, less for C++

# Microsoft .NET

---

## Microsoft's Internet strategy

- Not an OS
- Delivers software as web services
- Framework for universally accessible services
- Server-centric computing model

# Object runtime environment

---

## Common Language Runtime (CLR)

- Services compile to **Intermediate Language (IL)**
- Language neutral
  - C++, C#, VB, Jscript + 3<sup>rd</sup> party support
- Common class libraries
  - ADO.NET, ASP.NET, Windows Forms

# Common Language Runtime

---

## Implementation of common features:

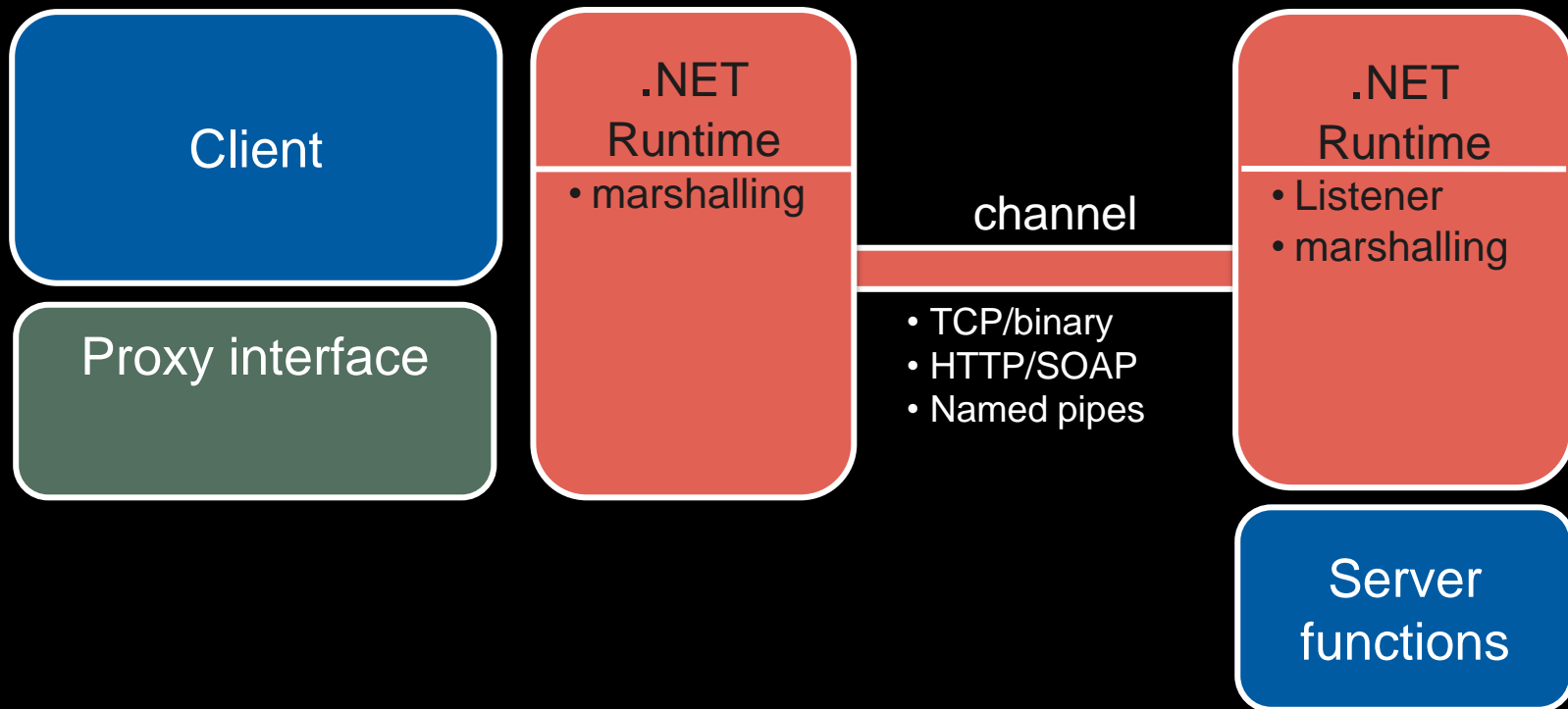
- lifetime management
- garbage collection
- security
- versioning

## When first loaded (prior to running):

- CLR runs just-in-time compiler to generate native code
- Never interpreted

# .NET Remoting

---



# .NET Remoting

---

- Object interaction across application domains
- Invoke remote objects
  - Remote object derived from **MarshalByRefObject** (similar to Java's *remote* class)
  - Proxy created when object is activated
  - CLR intercepts calls to the object
  - The CLR is told which classes are remote so it can do the right thing when the client requests a *new* object
- Passing objects as parameters
  - Objects implement **ISerializable** interface (analogous to Java's *serializable* class)

# Object Activation

---

- **Server Activated Objects**

- **Single Call**: new instance per call (stateless)
  - Created when the client invokes the first method
- **Singleton**: same instance for all requests
  - If the instance does not exist, the server creates one and all other requests from clients use it

- **Client Activated Objects**

- Lifetime is controlled by the client. Created when client calls *new*
- Similar to DCOM (COM+) model
- Supports distributed garbage collection

# Leasing Distributed Garbage Collector (LDGC)

- Used with Client Activated Objects
- **Lease Manager** manages object leases at a server
  - Server object is in use as long as its lease has not expired
  - If a client wants to be contacted when a lease expires, it needs to provide a **sponsor** object. The sponsor object can then extend the lease.
- Parameters:
  - **InitialLeaseTime**: initial lifetime of the remote object (5 min default)
  - **LeaseManagePollTime**: interval at which lease manager polls leases (10 sec default)
  - **sponsorshipTimeOut**: amount of time the framework waits for a
- Each time a method is called:
  - **renewOnCallTime**: amount of time to renew lease after each method call
  - Lease time set to **MAX(lease time - expired time, RenewOnCallTime)**
  - Requestor has to renew lease when **leaseTime** elapses
  - No more reference counting!

# .NET Remoting

---

- .NET Remoting: Homogenous environment – communicate among .NET processes
- .NET supports Web services but .NET Remoting isn't it.
  - Web Services Enhancement (WSE) focused on SOAP-based Web Services
- Successor to .NET facilities:  
Windows Communication Foundation (WCF)
  - Unifying communication framework
  - Supports interoperability with other platforms

Away from RPC...

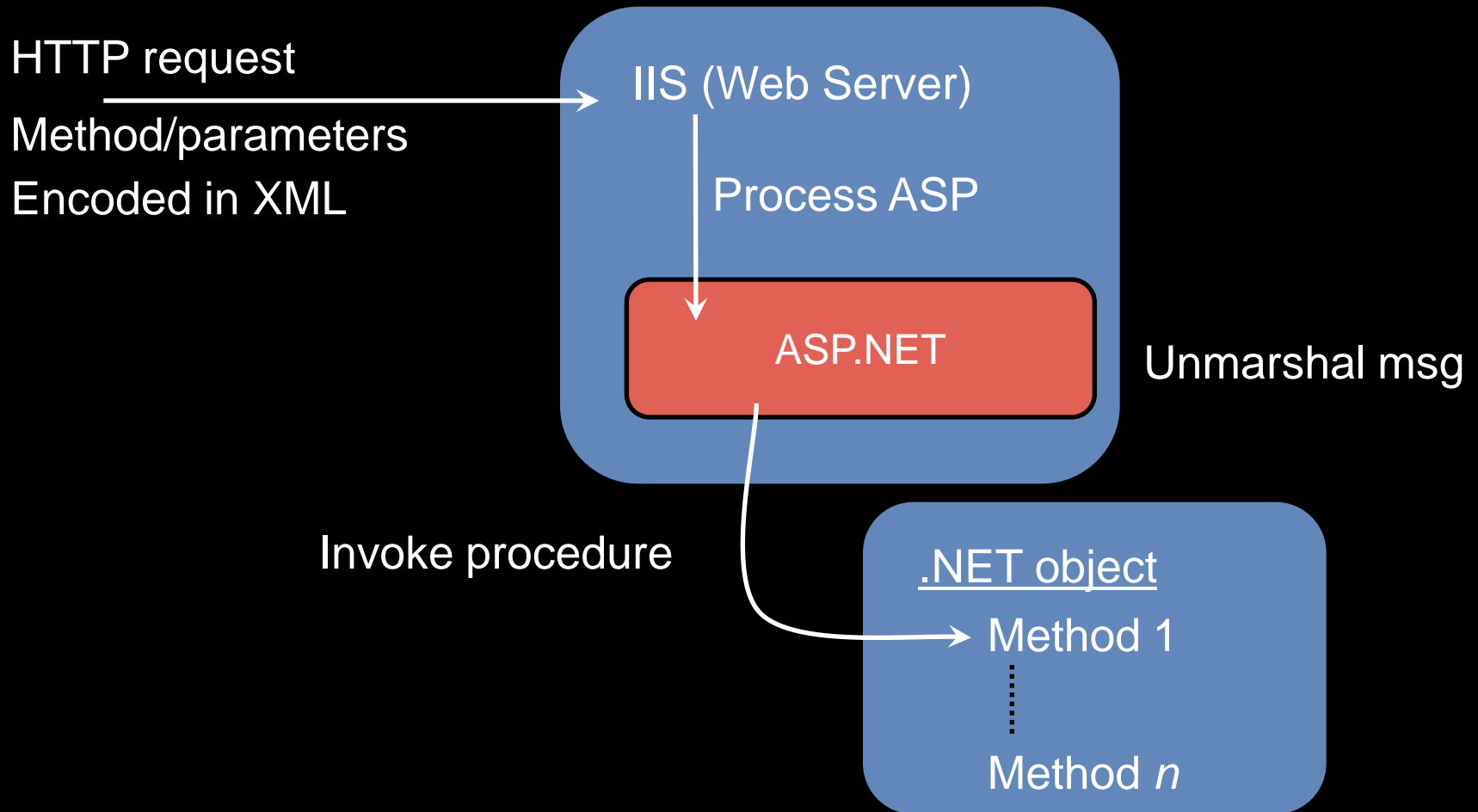
Broader Web Services

# .NET Web Services vs. SOAP

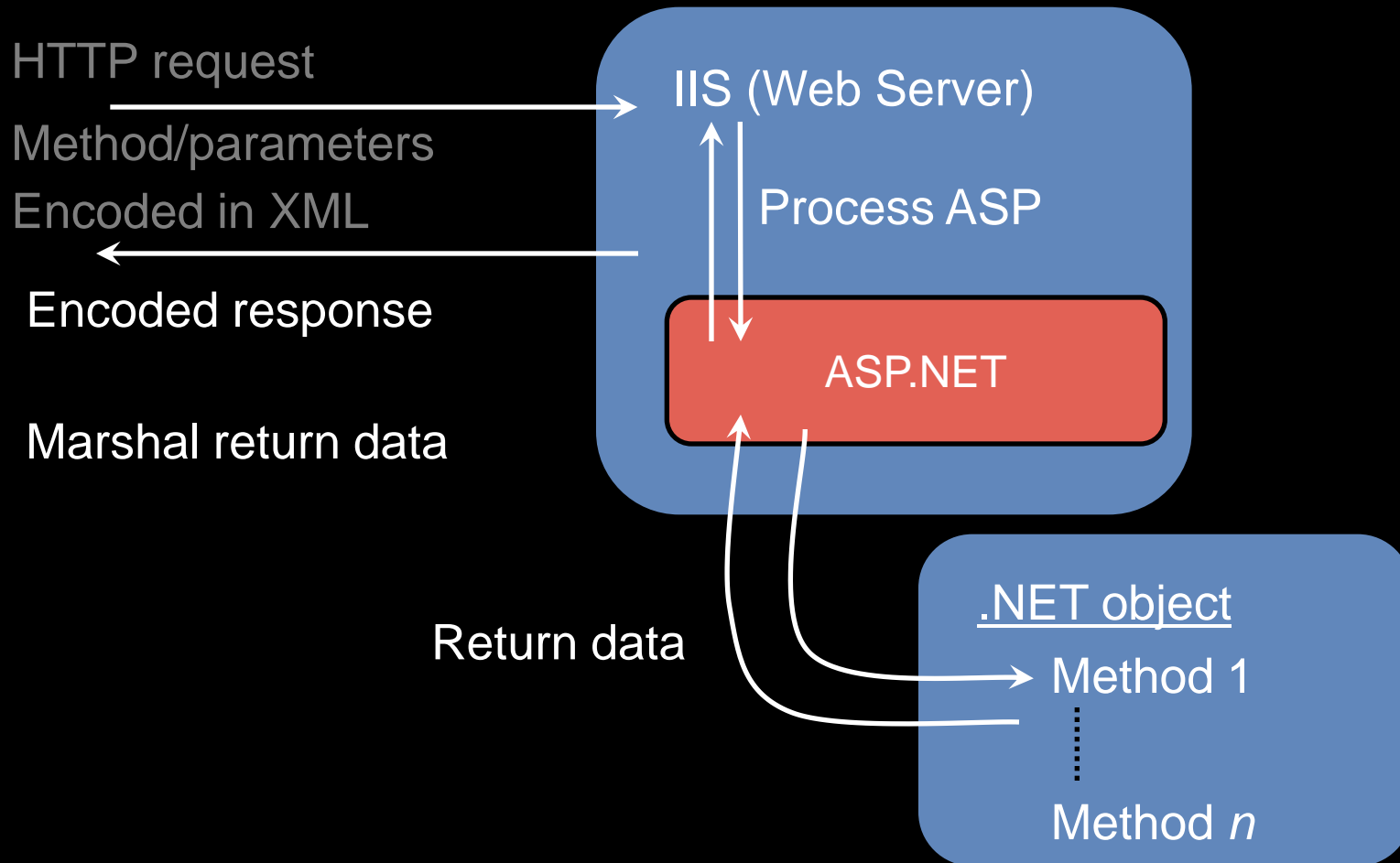
---

- SOAP is lower-level messaging protocol
- Web Services provides higher level of abstraction
- Write .NET object as if it were accessed by local clients
  - Mark it with attribute that it should be available to Web clients
  - ASP.NET does the rest
    - Hooks up an infrastructure that accepts HTTP requests and maps them to object calls
- Service description in WSDL
  - Automatically generated by examining metadata in .NET object

# Web Service invocation



# Web Service invocation



# Windows Communication Foundation

---

- **Service** is implemented in a *Service Class*
  - CLR-based language
  - Implements one or more methods
  - Contains
    - Service Contract: methods that a client can use
    - Data Contract: define data structures
- Runs in a **host process**
  - Service typically compiled into a library
  - Host process can be:
    - IIS (web server) host process (must use SOAP over HTTP)
    - Windows Activation Service
    - Arbitrary process
- **Endpoint definitions**
  - **Address**: URL
  - **Binding**: description of the protocols and security mechanisms
  - **Contract**: name indicating which Service Contract this endpoint exposes

# WCF Binding Options

---

BasicHttpBinding	SOAP over HTTP (optional HTTPS)
WsHttpBinding	same + support for reliable message transport, security, and transactions
NetTcpBinding	Binary-encoded SOAP over TCP, with support for reliable message transport, security, and transactions
WebHttpBinding	HTTP or HTTPS with no SOAP – ideal for RESTful communication; data in XML, JSON, or binary
NetNamedPipesBinding	binary-encoded SOAP over named pipes (WCF-WCF only)
NetMsmqBinding	binary-encoded SOAP over MSMQ (WCF-WCF only)

# Until 2006...

---

## Google Web APIs Developer Kit - SOAP

[www.google.com/apis/download.html](http://www.google.com/apis/download.html)

- A WSDL file you can use with any development platform that supports web services.
- A Java library that provides a wrapper around the Google Web APIs SOAP interface.
- An example .NET program which invokes the Google Web APIs service.
- Documentation that describes the SOAP API and the Java library.

# The future of SOAP?

---

- SOAP
  - Dropped by Google in 2006
  - Alternatives: AJAX, XML-RPC, REST, ...
  - Allegedly complex because “we want our tools to read it, not people”
    - unnamed Microsoft employee
- Microsoft
  - Provides SOAP APIs for Microsoft Live
  - <http://search.live.com/developer>
- Still huge support within web services and SOA

# AJAX

---

- Asynchronous JavaScript And XML
- Asynchronous
  - Client not blocked while waiting for result
- JavaScript
  - Request can be invoked from JavaScript (using XMLHttpRequest)
  - JavaScript may also modify the Document Object Model (DOM) – how the page looks
- XML
  - Data sent & received as XML

# AJAX & XMLHTTP

---

- Allow Javascript to make HTTP requests and process results (change page without refresh)
  - IE: `new ActiveXObject("msxml3.XMLHTTP")`
  - Mozilla/Opera/Safari:  
`new XMLHttpRequest()`  
`xmlhttp.open("HEAD", "index.html", true)`
- Tell object:
  - Type of request you're making
  - URL to request
  - Function to call when request is made
  - Info to send along in body of request

# AJAX on the Web

---

- Ushered in Web 2.0
- Google Maps, Amazon Zuggest, Del.icio.us Director, Writely, ...
- Microsoft ASP.NET AJAX 1.0
  - January 2007
  - Integrates client script libraries with ASP.NET server-based code
- Google recommends use of their AJAX Search API instead of SOAP Search API

# REST

---

## REpresentational State Transfer

- Stay with the principles of the web
  - Four HTTP commands let you operate on data (a resource):
    - PUT (insert)
    - GET (select)
    - POST (update)
    - DELETE (delete)
- In contrast to invoking operations on an activity.
- Message includes representation of data.

# Resource-oriented services

---

- Blog example
  - Get a snapshot of a user's blogroll:
    - **HTTP GET //rpc.bloglines.com/listsubs**
    - HTTP authentication handles user identification
  - TO get info about a specific subscription:
    - **HTTP GET http://rpc.bloglines.com/getitems?s={subid}**
- Makes sense for resource-oriented services
  - Bloglines, Amazon, flickr, del.icio.us, ...

# Resource-oriented services

---

- Get parts info  
**HTTP GET //www.parts-depot.com/parts**
- Returns a document containing a list of parts  
(implementation transparent to clients)

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
  <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
  <Part id="00347" xlink:href="http://www.parts-depot.com/parts/00347"/>
  <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```

# Resource-oriented services

---

- Get detailed parts info:  
**HTTP GET //www.parts-depot.com/parts/00345**
- Returns a document containing a list of parts (implementation transparent to clients)

```
?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part-ID>00345</Part-ID>
  <Name>Widget-A</Name>
  <Description>This part is used within the frap assembly</Description>
  <Specification xlink:href="http://www.parts-depot.com/parts/00345/specification"/>
  <UnitCost currency="USD">0.10</UnitCost>
  <Quantity>10</Quantity>
</p:Part>
```

# REST vs. RPC

---

Example from wikipedia:

## RPC

`getUser()`, `addUser()`, `removeUser()`, `updateUser()`,  
`getLocation()`, `AddLocation()`, `removeLocation()`

```
exampleObject = new ExampleApp("example.com:1234");  
exampleObject.getUser();
```

## REST

`http://example.com/users`

`http://example.com/users/{user}`

`http://example.com/locations`

```
userResource =  
    new Resource("http://example.com/users/001");  
userResource.get();
```

# REST-based Systems

---

- Yahoo! Search APIs
- Ruby on Rails 1.2
- Twitter
- Open Zing Services – Sirius radio

`svc://Radio/ChannelList`

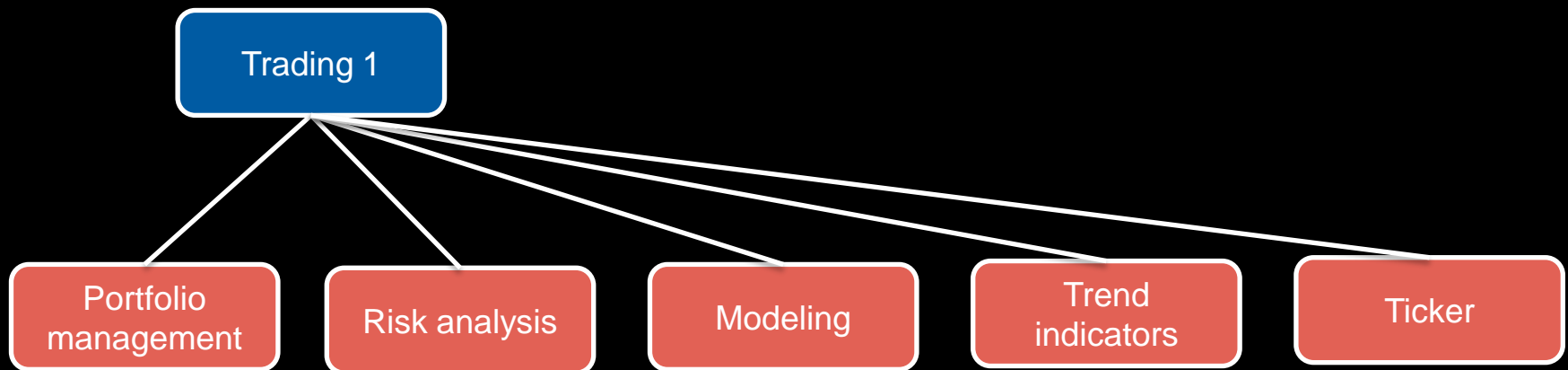
`svc://Radio/ChannelInfo?sid=001-siriushits1&ts=2007091103205`

# (Enterprise) Service Bus

# Motivation

---

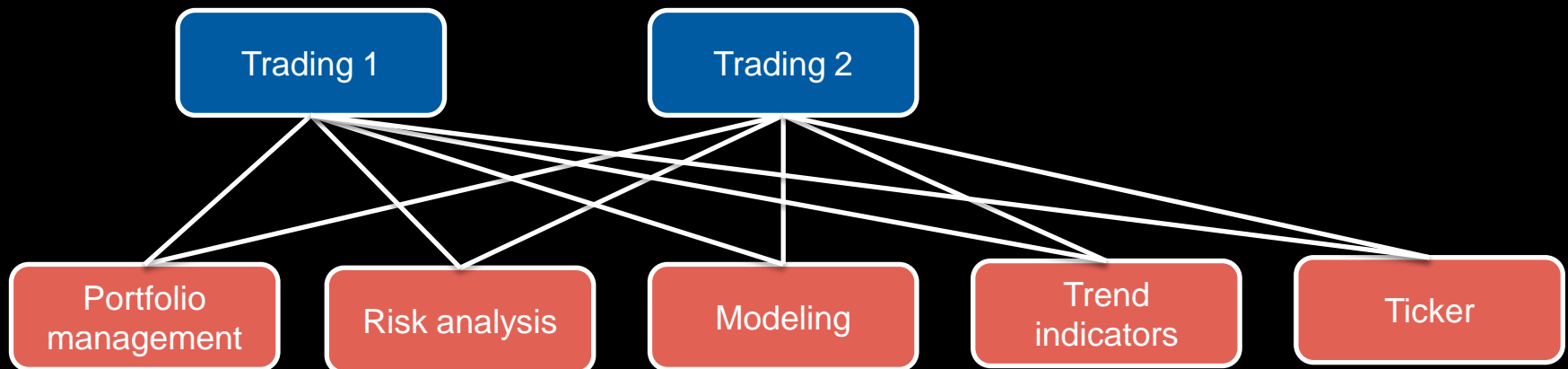
- An environment that comprises
  - Applications on different machines
  - Multiple vendors and platforms
  - Applications that generate and consume messages with each other



# Motivation

---

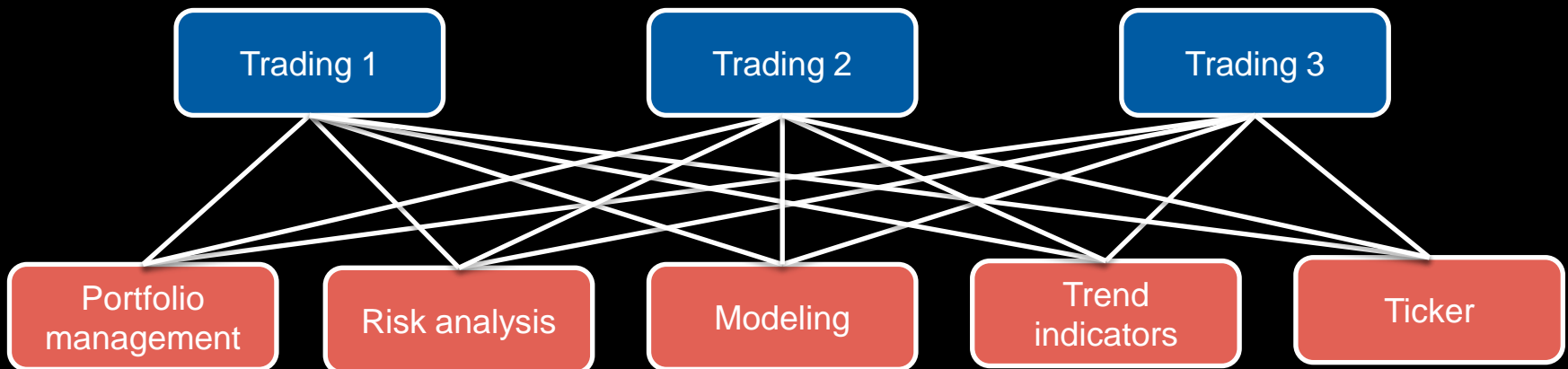
- An environment that comprises
  - Applications on different machines
  - Multiple vendors and platforms
  - Applications that generate and consume messages with each other



# Motivation

---

- An environment that comprises
  - Applications on different machines
  - Multiple vendors and platforms
  - Applications that generate and consume messages with each other

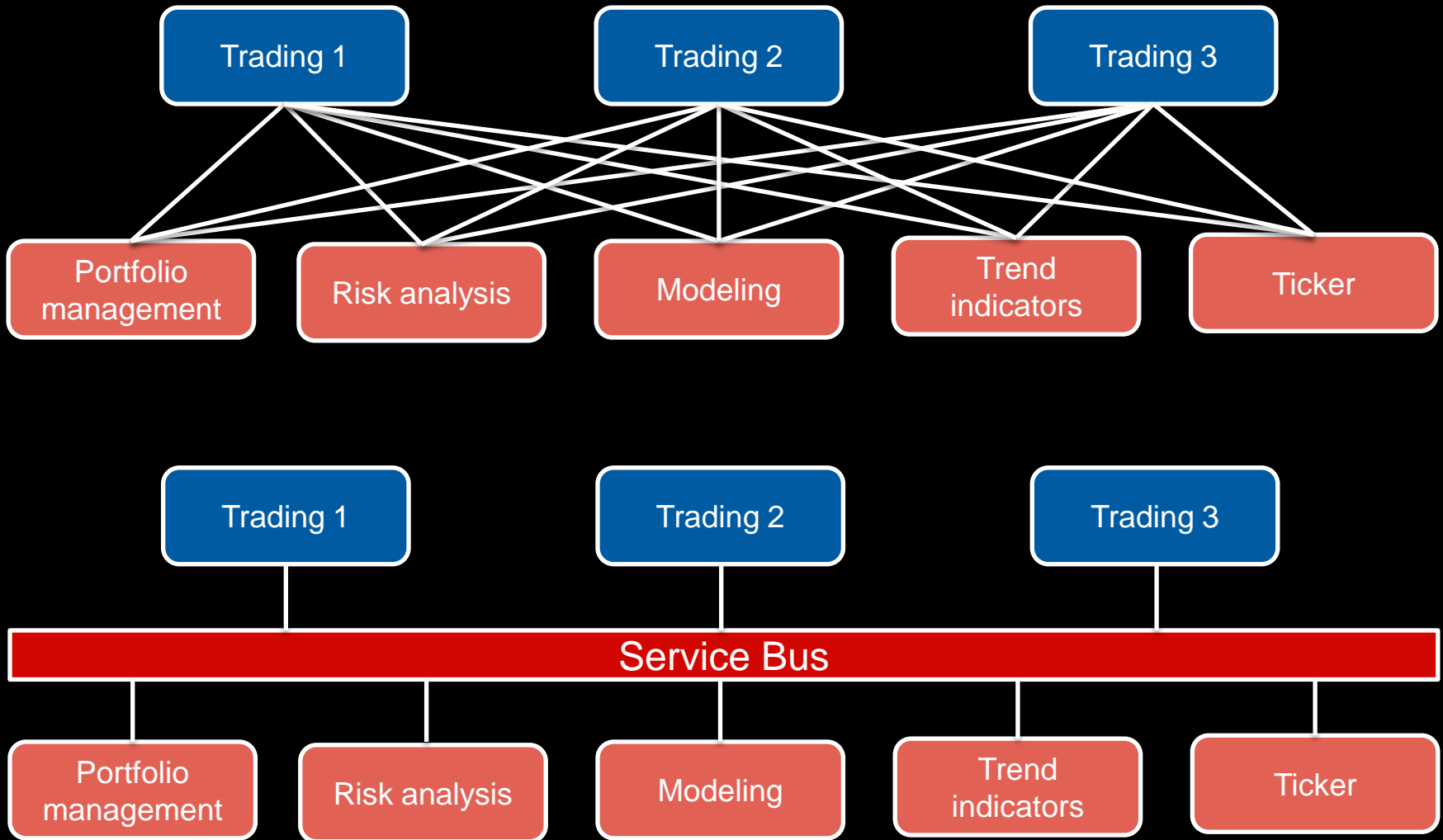


# Service Bus

---

- What is it?
  - Messaging infrastructure between applications
  - Sometimes called an *Enterprise Service Bus (ESB)*
- Applications to not connect to services
  - Instead, everyone talks to the message bus
- Contains
  - Agreed-upon message schemas (formats)
  - Common command messages
  - Shared communication infrastructure

# Reduce fan-out of each application



# What does a Service Bus do?

---

- Service Bus functions
  - Coordinate message sending, delivery, and subscriptions
    - Either a subscribe-publish or message routing model may be used
  - Prioritize or delay messages
  - Provide reliable delivery (if needed)
  - Logging/monitoring
- Supporting endpoint services
  - Naming & lookup (directories)
  - Marshalling/unmarshalling
  - Encryption
  - Filtering
  - Authentication

# Tons of ESBs out there!

---

- Apache ServiceMix
- Petals ESB
- Open ESB
- Mule ESB
- IBM WebSphere ESB
- Microsoft BizTalk Server
- Oracle Enterprise Service Bus
- *And lots more*

# Just Marshalling

# Google Protocol Buffers

---

- Efficient mechanism for serializing structured data
  - Much simpler, smaller, and faster than XML
- Language independent
- Define messages
  - Each message is a set of names and types
- Compile the messages to generate data access classes for your language
- Used extensively within Google. Currently over 48,000 different message types defined.
  - Used both for RPC and for persistent storage

# Example (from the Developer Guide)

---

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

<http://code.google.com/apis/protocolbuffers/docs/overview.html>

# Example (from the Developer Guide)

---

```
Person person;  
person.set_name("John Doe");  
person.set_id(1234);  
person.set_email("jdoe@example.com");  
fstream output("myfile", ios::out | ios::binary);  
person.SerializeToOstream(&output);
```

<http://code.google.com/apis/protocolbuffers/docs/overview.html>

# Efficiency example (from the Developer Guide)

```
<person>
  <name>John Doe</name>
  <email>jdoe@example.com</email>
</person>
```

XML version

```
person {
  name: "John Doe"
  email: "jdoe@example.com"
}
```

Text (uncompiled) protocol buffer

- Binary encoded message: ~28 bytes long, 100-200 ns to parse
- XML version: ≥69 bytes, 5,000-10,000 ns to parse

<http://code.google.com/apis/protocolbuffers/docs/overview.html>

# JSON

---

- Lightweight (relatively efficient) data interchange format
  - Introduced as the “fat-free alternative to XML”
- Human writeable and readable
- Language independent
- Easy to parse
- Based on JavaScript
- Currently converters for 46 languages

The End