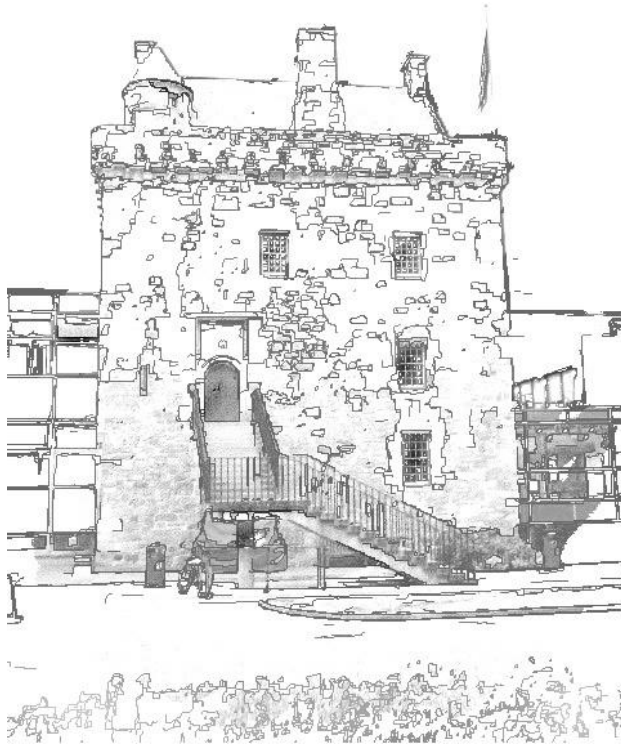




# .NET XML Web Services



Bill Buchanan



# Course Outline

Date	Title	Kalani
	Unit 0: Introduction to .NET	-
	Unit 0: Introduction to .NET	-
	Unit 1: Creating/Manipulating Datasets	Unit 1
	Unit 1: Creating/Manipulating Datasets	Unit 1
	Unit 2: Manipulating XML Data	Unit 2
	Unit 2: Manipulating XML Data	Unit 2
	Unit 3: .NET Remoting	Unit 3
	Unit 3: .NET Remoting	Unit 3
	Unit 4: Web Services	Unit 4
	Unit 4: Web Services	Unit 5

**1 Unit 1**

**1.1 Introduction to XML**

XML languages of the past, especially HTML, have been fixed in their tags. A new format named XML (Extensible Markup Language) can be used to create new tags, and provide a common platform for transferring information between different systems and packages. The first line of an XML file typically contains an optional and processing directive known as the XML declaration. This one contains general information that indicates the XML language version, the character set, and whether it can be used in a standalone entity. For example, the XML declaration that begins every valid XML file:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

The XML document conforms to the XML recommendations, and has a logical structure that is composed of declarations, elements, comments, character references, and processing instructions. It also has a physical structure which is composed of entities, starting with the root, or document entity.

XML uses a document type definition (DTD) which defines the rules of the document, such as the elements which are present and the structural relationship between all the elements. It then defines the tags that can be used and the tags that can contain other tags, the number and sequence of the tags, the attributes of the tag and, optionally, their values. DTDs allow documents to be properly validated, and is used within the production of an XML file. A schema is fundamentally equivalent to a DTD, but is written in XML. It consists of the base DTD, providing data typing, inheritance, and presentation rules.

The vocabulary of XML is a set of actual elements and the structure for a specific document type used in particular data formats. These are defined in a DTD that is the backbone for the vocabulary. One of the first of these languages is the Character Entity Set which is used to define Web pages that automatically send their contents to users through an "xml" redirection.

The XML object model defines a standard way in which the contents of the XML document are accessed. It is a fully object-oriented and uses properties, methods, and the actual content (data) contained in an object. This model extends how users interpret the files, and requires all data elements as objects, which can be accessed without any return tags to the screen. The XML object model uses the W3C standard known as Document Object Model.

XML Data Binding (XDB) is one of the first languages defined which uses a schema other than one that is defined in an XML form. It defines the:

- Form of elements that use child elements of others.
- Sequence in which the child elements can appear.
- Number of child elements.

It also defines whether an element is empty or not, include text. XDB is now well established and uses XML as its base language. A new standard known as XSD (XML Schema Definition) has been established by the W3C XML Schema Working Group.

An XML-based system typically uses an XML engine, which contains an XML parser, an XML processor, and schema engine. The XML parser reads the XML document and provides access to its content and structure. For this, it generates a hierarchical tree-based tree, and passes the data to various other applications for processing. A major function of the XSD parser is in checking the XML content and report any errors.

**1.1.1 XML Parser Language**

The XML parser language (XPParser) is a .NET-based schema which specifies the elements which are used in addressing the internal structure of XML documents. Its main aim is to provide references to elements, character strings, and other parts of XML documents, whether or not they have an explicit ID attribute. An XPParser consists of a series of location names, each of which specifies a location, usually relative to the location specified by the prior location name. Each location name has a forward slash in its ID, child, ancestor, and so on and can have arguments, such as an instance number, element type, or attribute. For example, the XPParser:

```
child::parent::
```

which refers to the third child element whose type is parent.

**1.1.2 XML Query Language (XQL)**

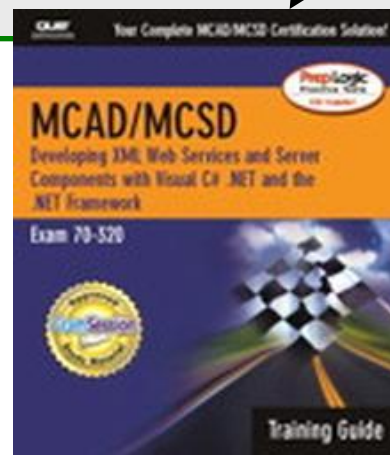
XQL is a set of extensions to XSL. Patterns that have been prepared by the W3C. It extends XSL so that it also includes search/lookup method. Unlike XSL documents, XQL defines ways to navigate XML, in order to create new documents, and to extract the content of existing documents.

**1.1.3 XML Schema Definition (XSD)**

XSD is a language which has been prepared by the W3C XML Schema Working Group and is used to defining schemas, which is required to become the standard method of defining XML schemas. XSD uses XML as its source language.

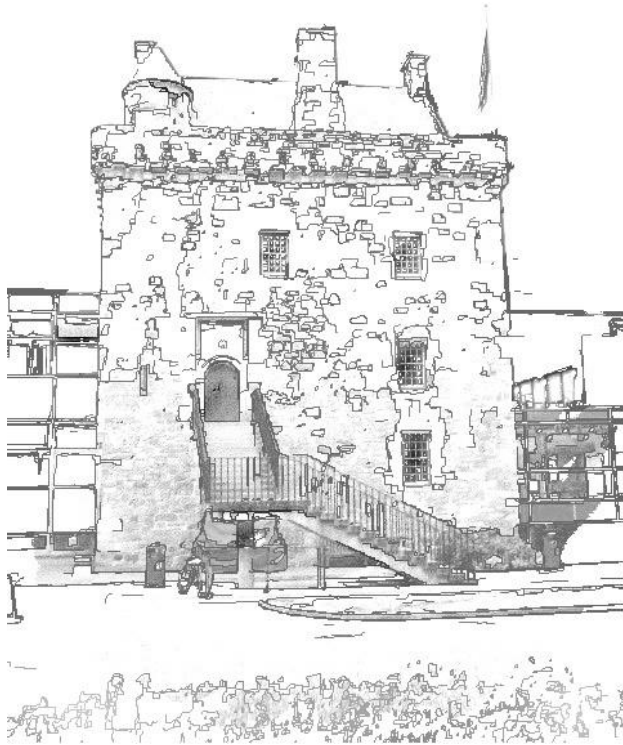
**1.1.4 Extensible Stylesheet Language (XSL)**

XSL is an XML-based language which transforms XML-based data into HTML, or other presentation formats, which can be displayed in a Web browser. This is achieved in a declarative way, which makes it independent of the original data language. It consists of XSL, use with formatting properties, which is transformed





# .NET Remoting



Bill Buchanan, SoC

---

Alistair Lawson, SoC





## **Creating and Managing Microsoft Windows® Services, Serviced Components, .NET Remoting Objects, and XML Web Services**

### **Create and consume a .NET Remoting object**

Implement server-activated components.

Implement client-activated components.

Select a channel protocol and a formatter. Channel protocols include TCP and HTTP. Formatters include SOAP and binary.

Create client configuration files and server configuration files.

Implement an asynchronous method.

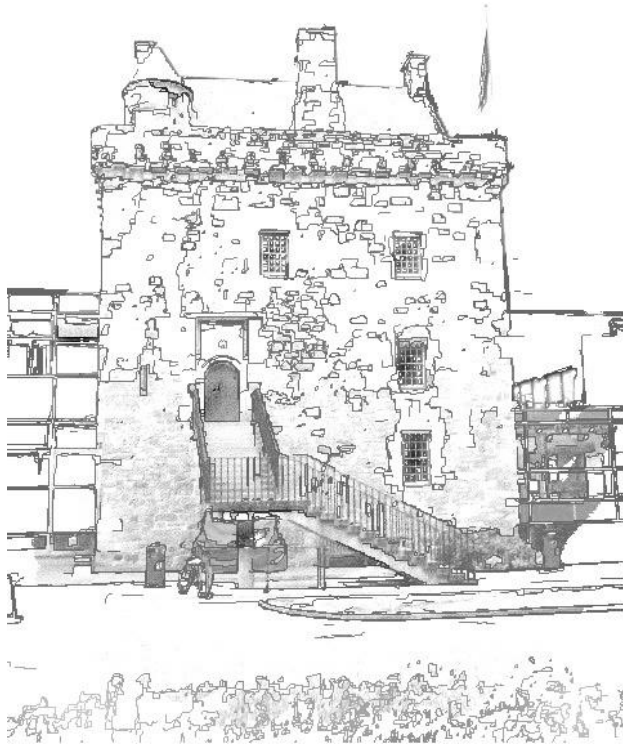
Create the listener service.

Instantiate and invoke a .NET Remoting object.





# .NET Remoting



Bill Buchanan, SoC

---

Alistair Lawson, SoC

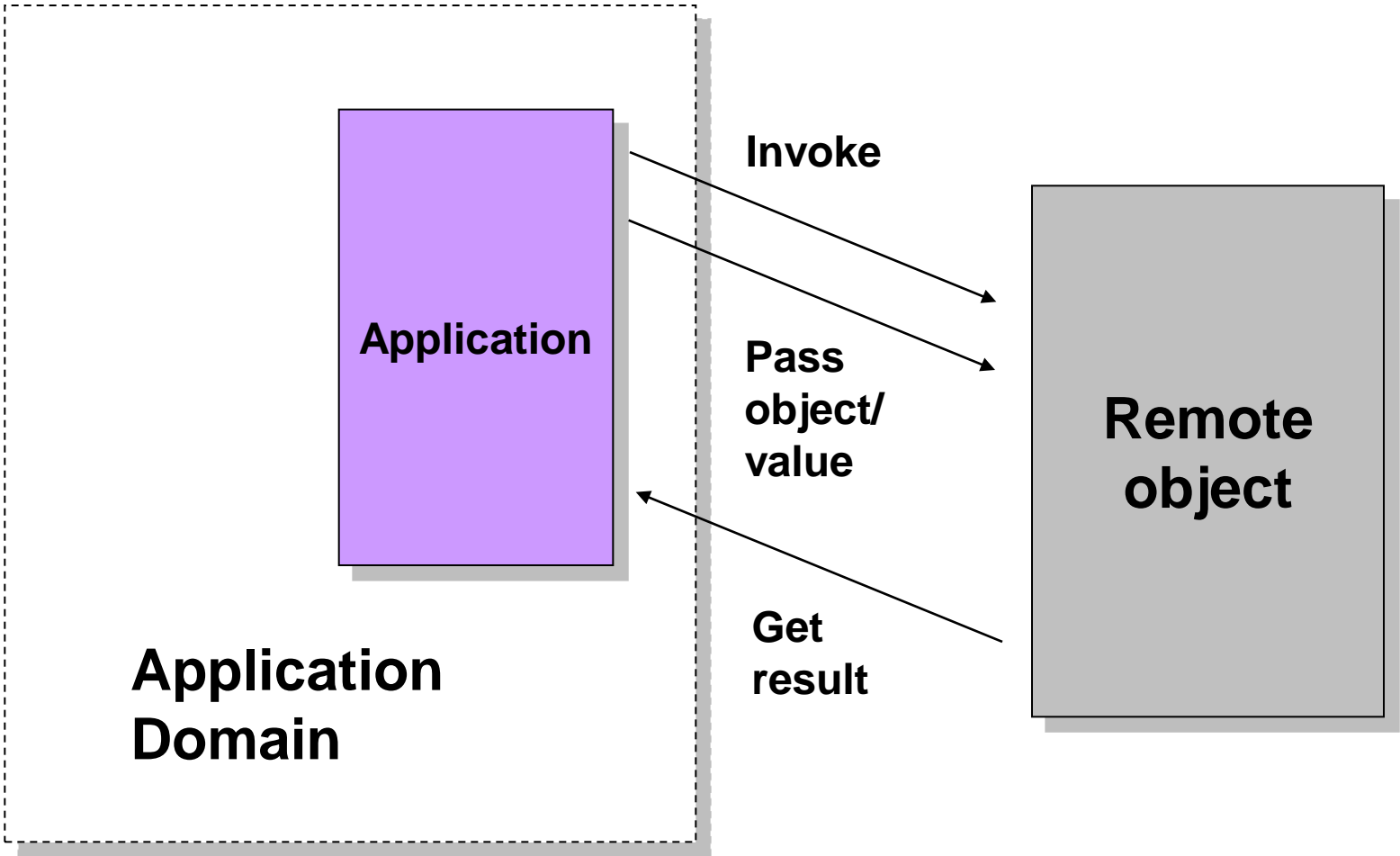


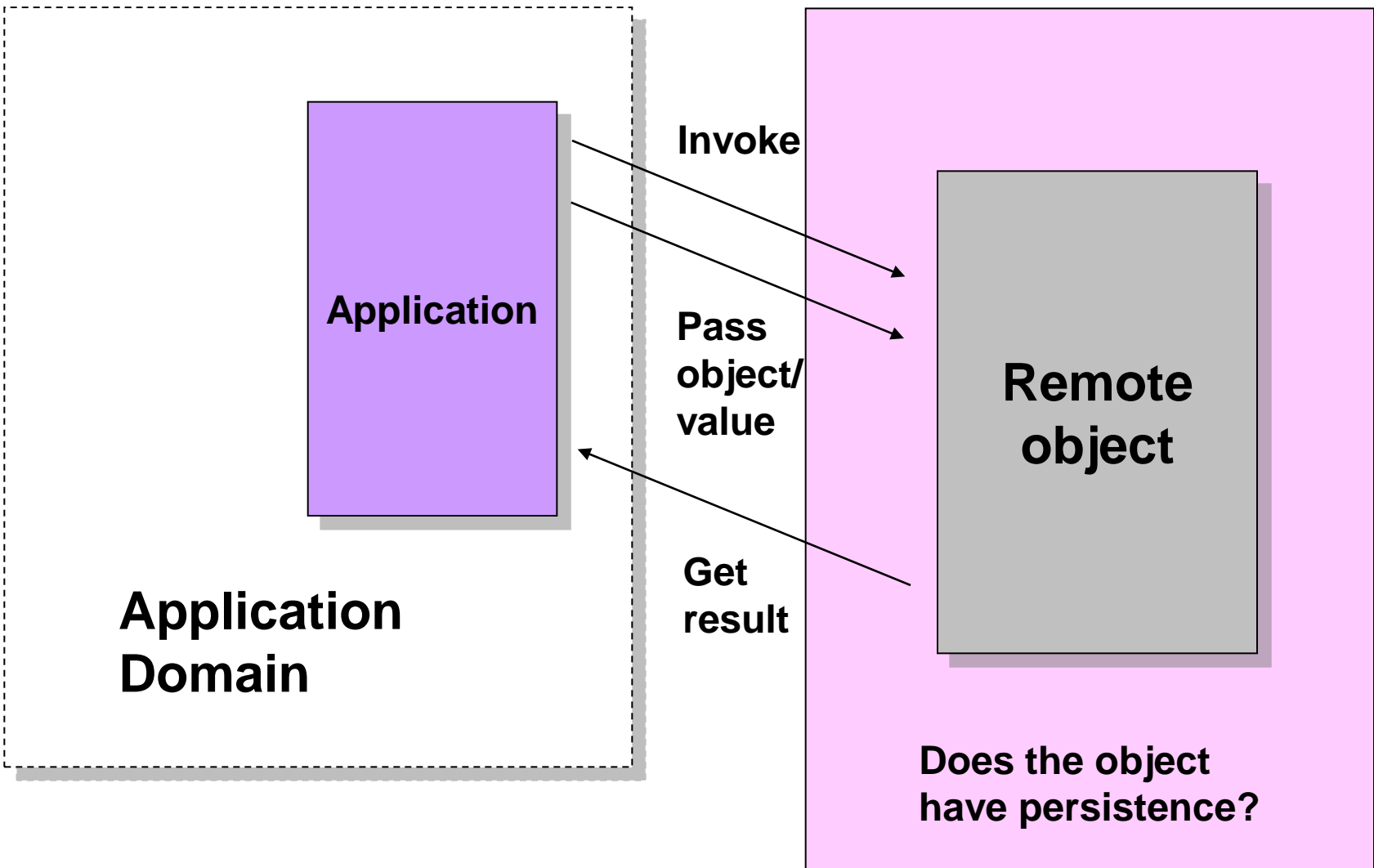


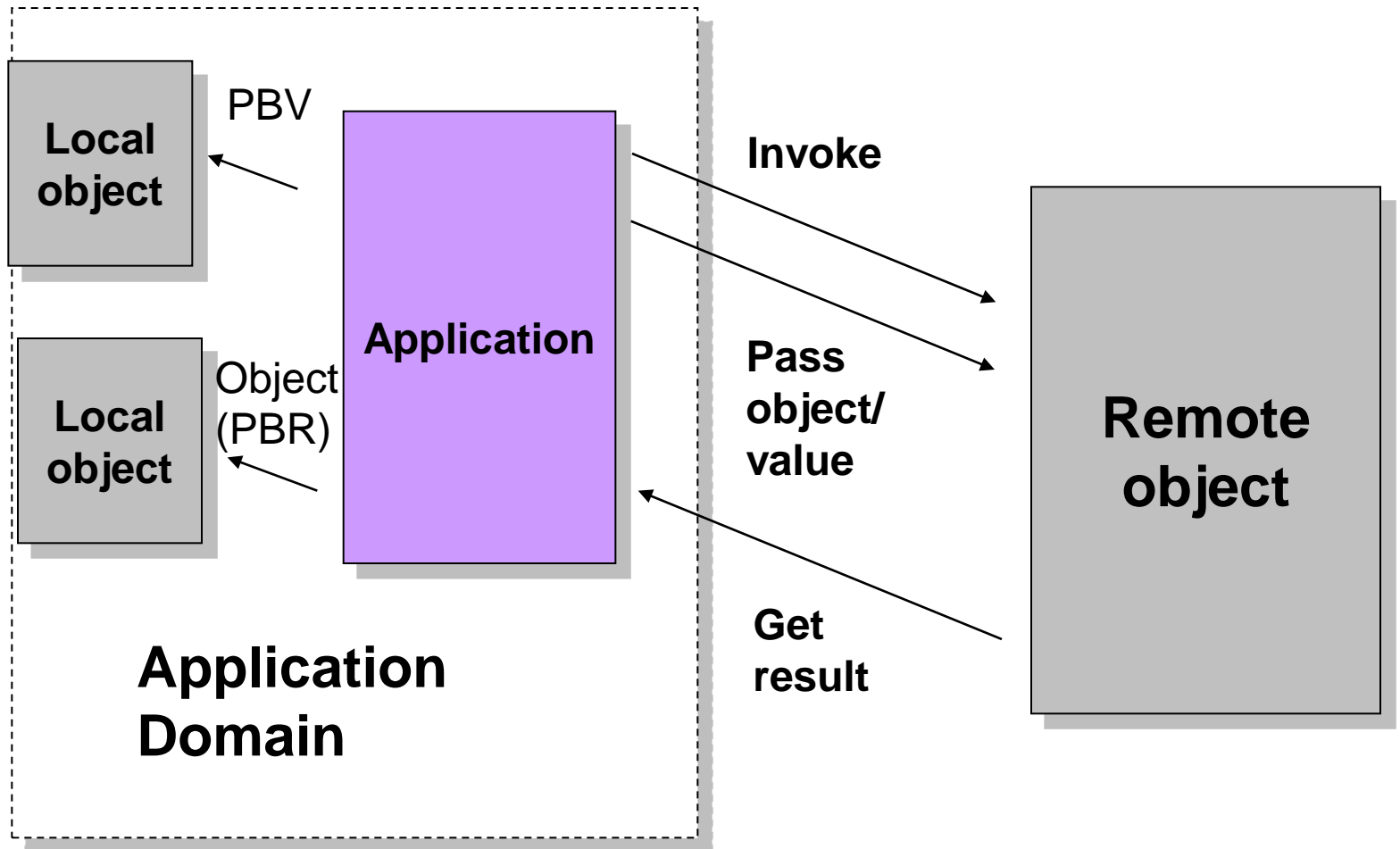
## Remote Objects

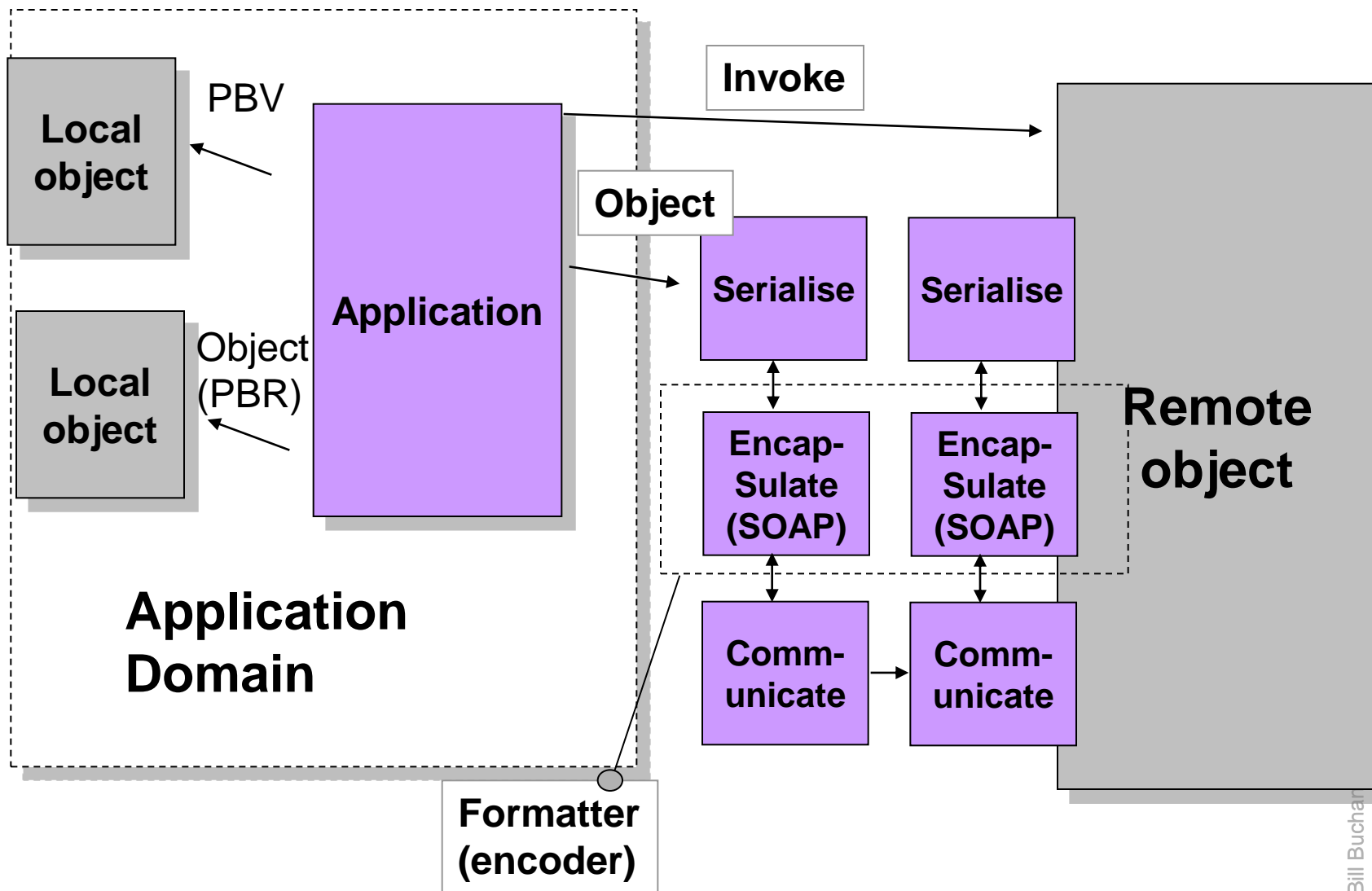
Bill Buchanan

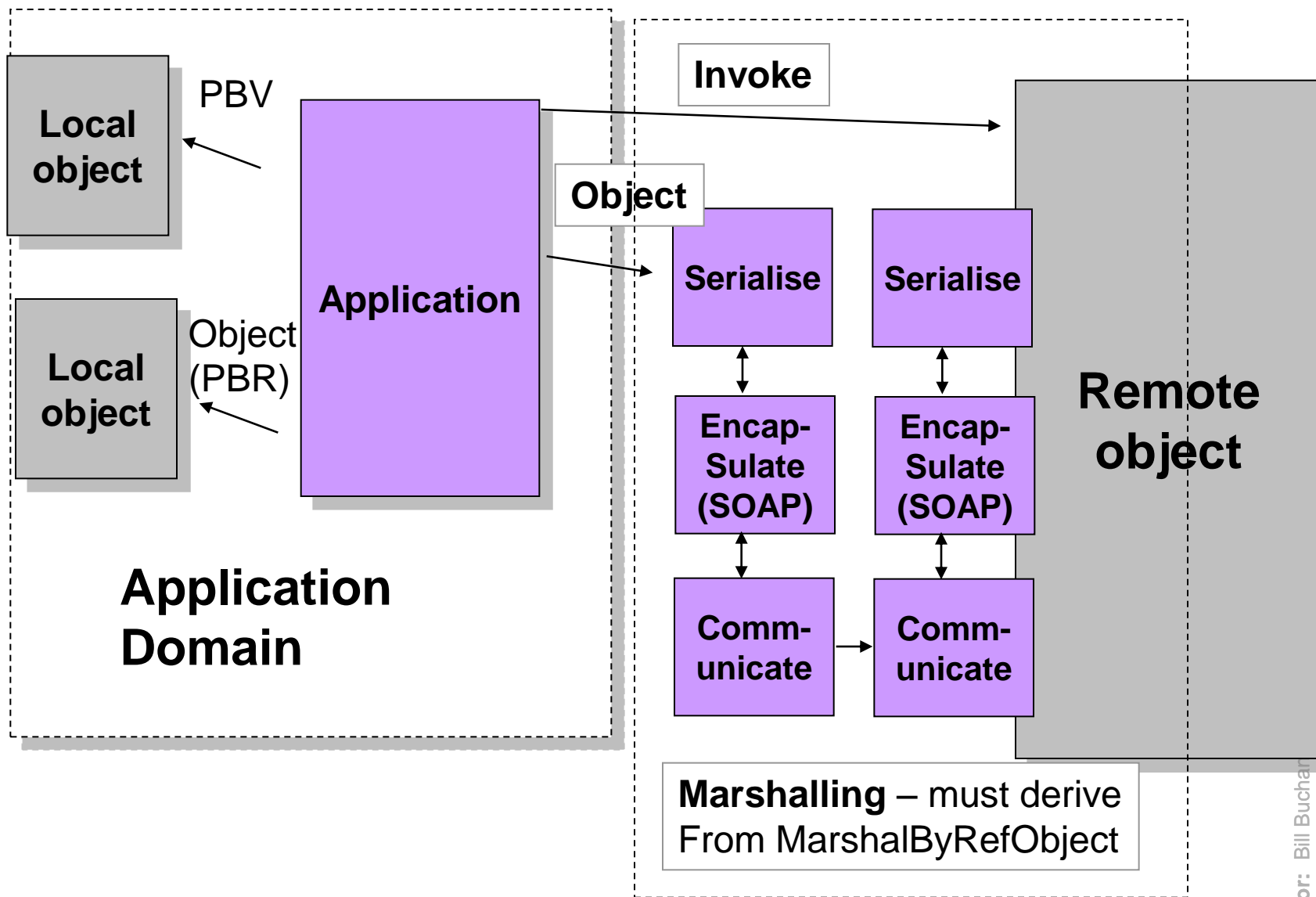










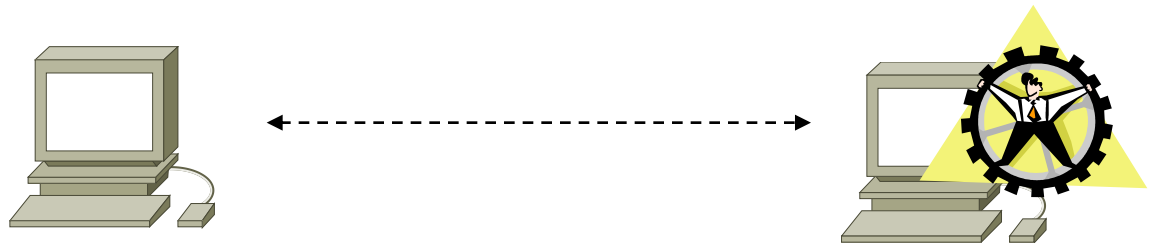




.NET remoting is a technology which allows objects to be placed remotely across a network, where an object can be activated, and communicate with local objects using a communications channel. A formatter is then used to encode and decode the messages as they pass between the remote object and the application. The format of these message can either be:

**Binary encoded.** This is used where **performance** is a critical factor.

**XML encoded.** This is used when **interoperability** is important, and use the standardized SOAP protocol.

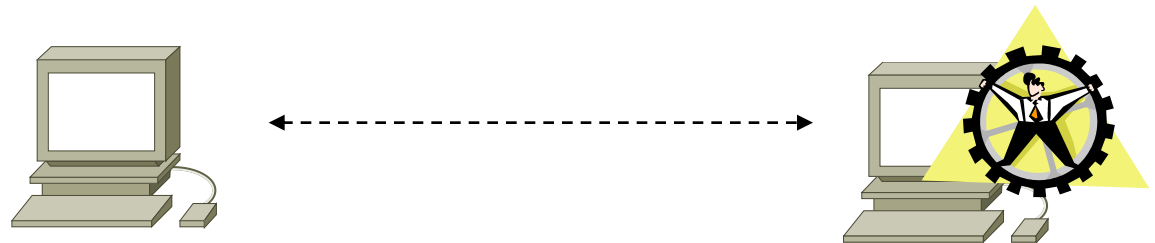


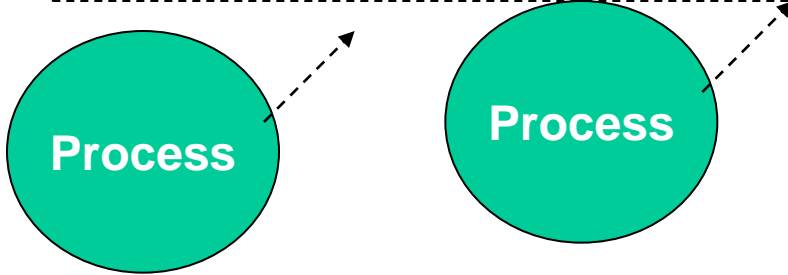
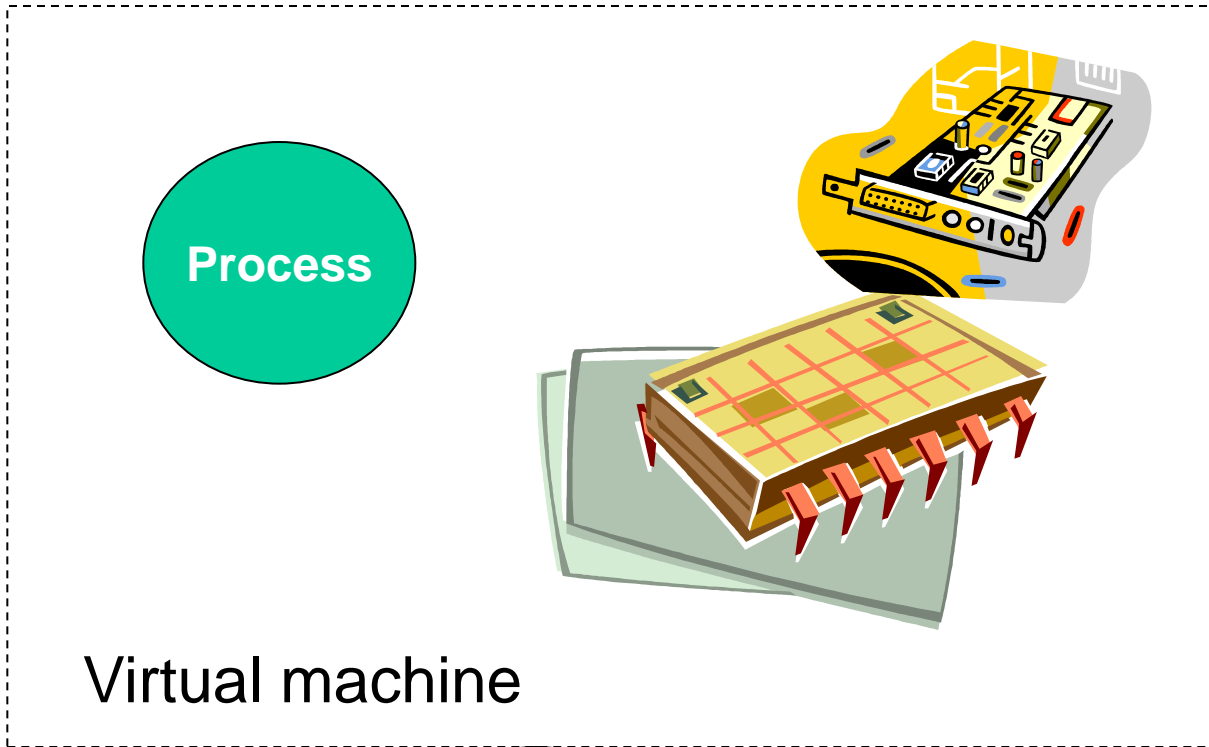


The remote objects are activated in different ways, these are:

**Client-activated objects.** These have a finite lease time, and once their lease has expired they are deleted (using the garbage collector).

**Server-activated objects.** These can either be defined with a "single call" or with a "singleton". A "single call" accepts one request from a client, and then performs the action, and is then deleted (with the garbage collector). It is defined as stateless, as it does not hold onto parameters from previous calls. Singletons are stateful and can accept multiple calls, where they remember previous calls, and retain their information. They can thus communicate with multiple clients. Also the lifetime of singletons is controlled by lease-based lifetime.

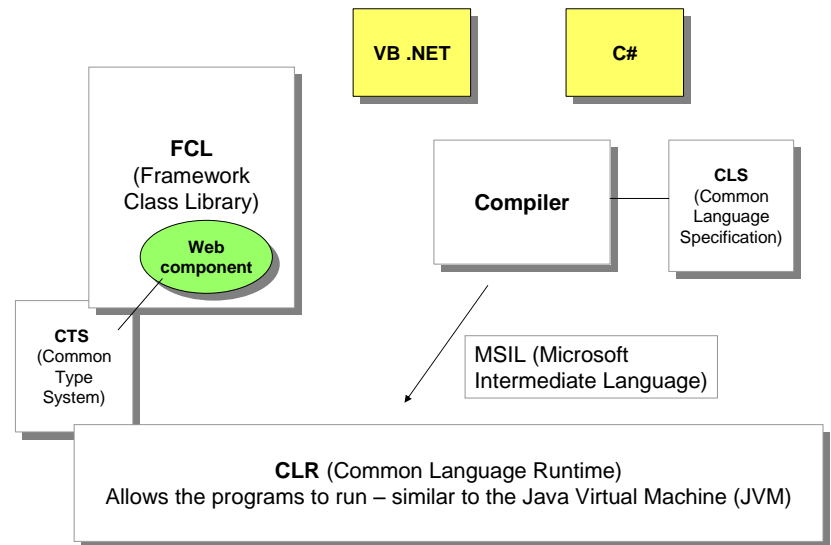
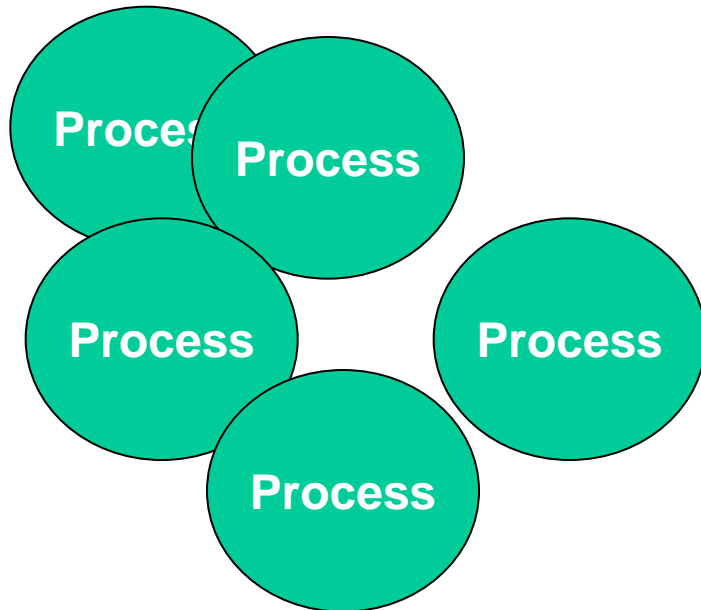
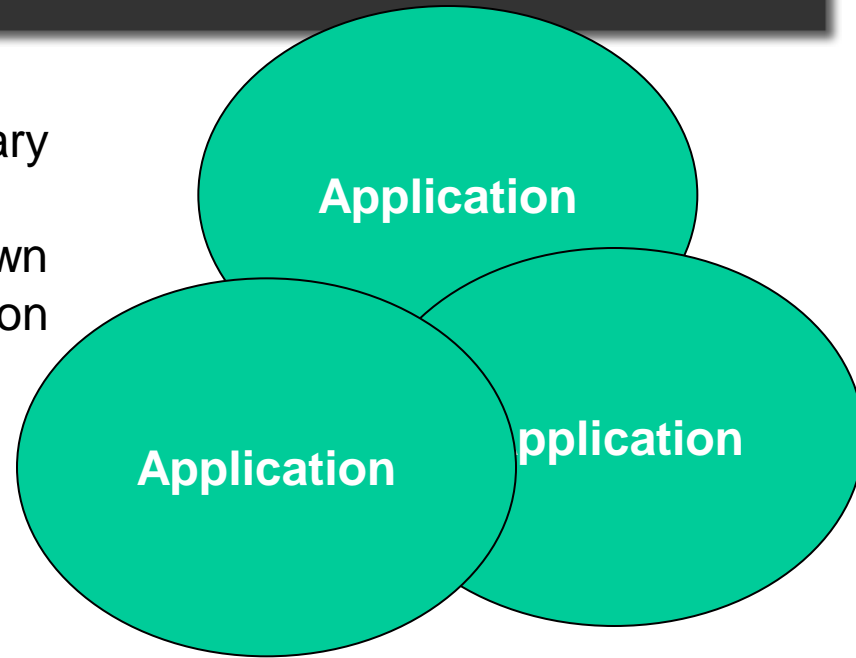






Thus the application boundary ensures that:

- Each application has its own code, data and configuration settings.
- No other application can interfere with any other applications.





# Distributed Systems

Bill Buchanan



# 3.1 Centralised v. Distributed

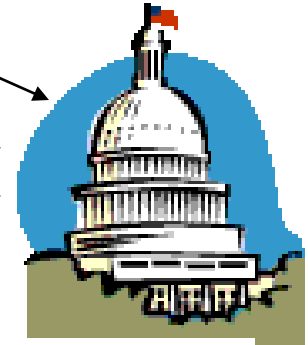


**Distributed:**  
Decision making  
Account management  
Logistics

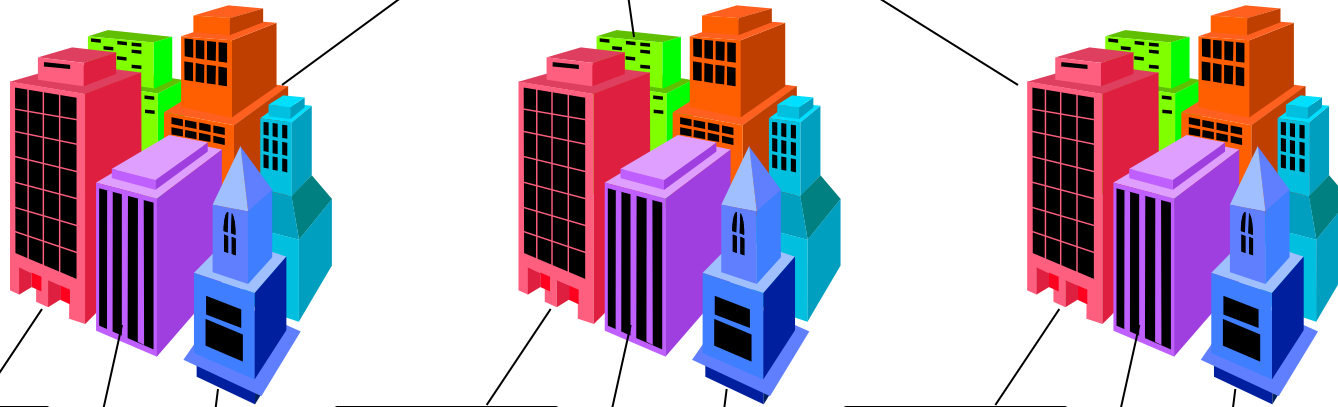


**Head Office**

Customers  
Staff  
Logistics



**Regional Office**



**Local Office**



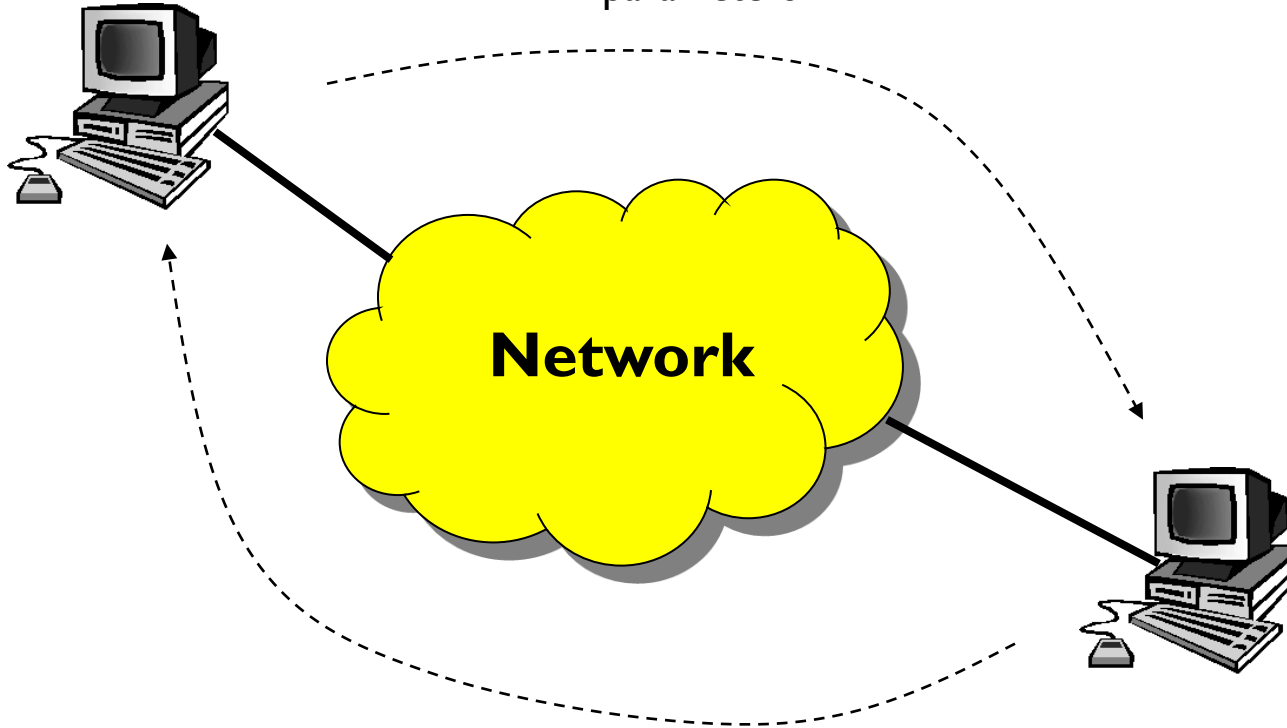
**ATM**



# Client/server architecture



Client requests  
a remote process  
and passes process  
parameters



Server runs process and  
returns the results to the  
client



In general distributed systems are generally more scaleable, more robust, and increase availability of services.

The most common distributed protocols are:

RPC (Remote Procedure Calls),  
Microsoft Distributed Object Model (DCOM),  
Common Object Request Broker Architecture (CORBA) and  
Java Remote Invocation (RMI).

These are typically applied to certain types of environments, such as DCOM on Windows-based systems and RPC in a UNIX environment.



## The main namespaces used for remoting:

**System.Net.** This includes classes relating to the networking elements of the distributed system.

**System.Runtime.Remoting.** This includes classes for the remote aspects of the .NET framework, such as mechanisms for remote communication between objects.

**System.Web.Services.** This includes protocols relating to Web services, such as those relating to HTTP and SOAP. This is defined as the ASP.NET Web services framework.



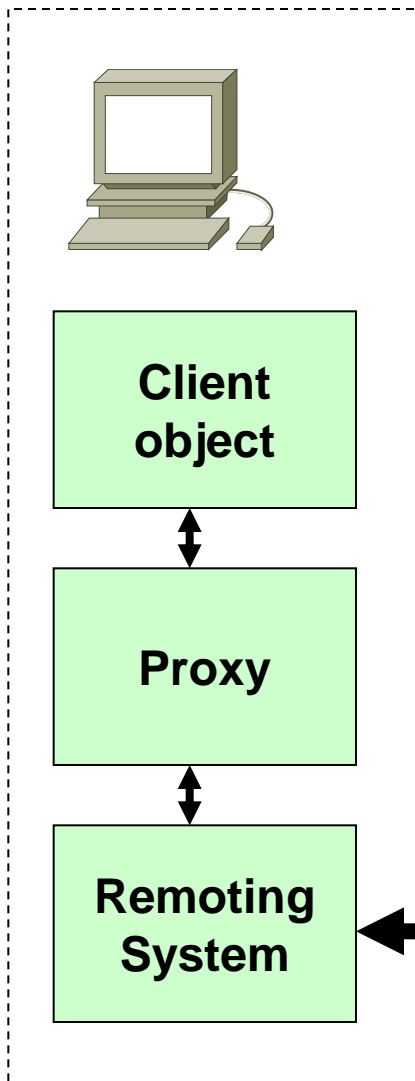
## Remote Objects

Bill Buchanan

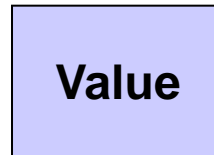
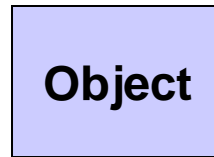


# Marshalling

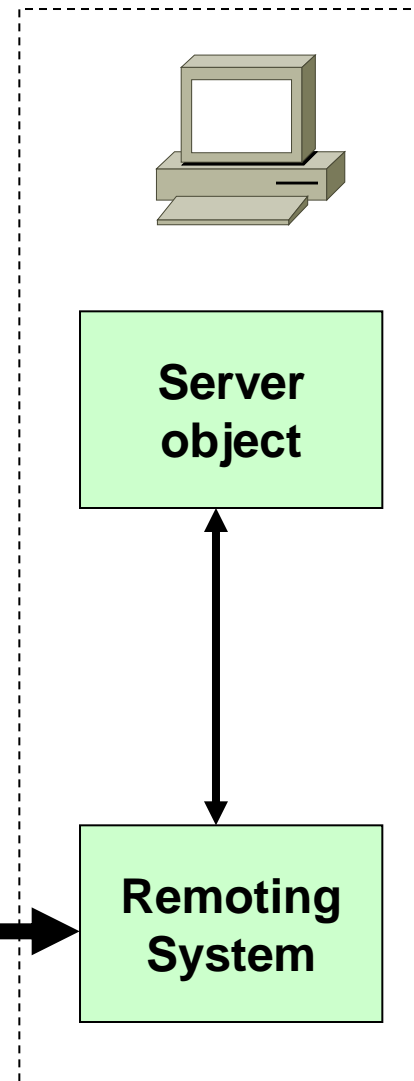
## Client Application Domain



Objects are passed by **reference** on a local machine – not possible to send a ref to remote machine.

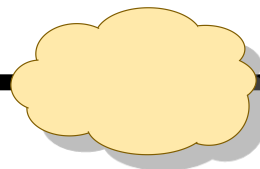


## Server Application Domain



Serialization

Channel



# Marshalling

Server Application Domain

```
public class ShowCapital : MarshalByRefObject
{
    public ShowCapital()
    {
    }
    public string show(string country)
    {
    }
}
```

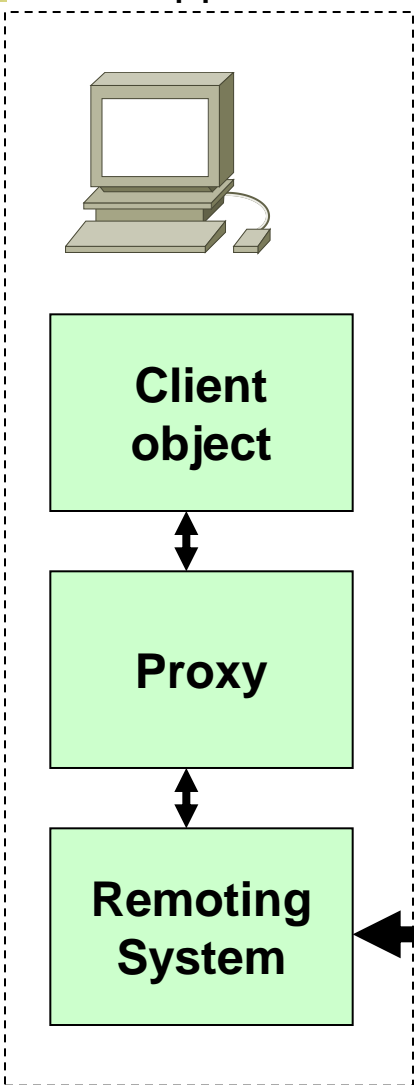
Derive from MarshalByRefObject

Creates a dynamic library (DLL)

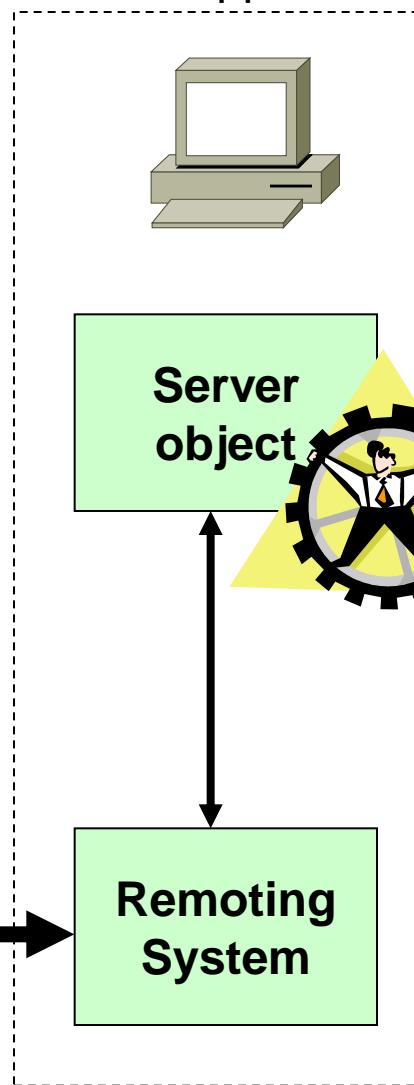
Remoting  
System



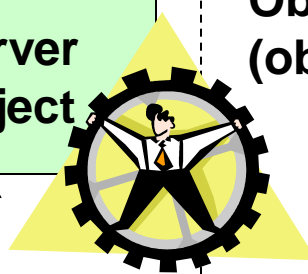
## Client Application Domain



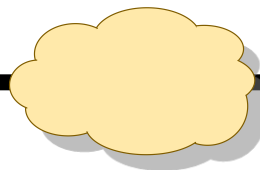
## Server Application Domain



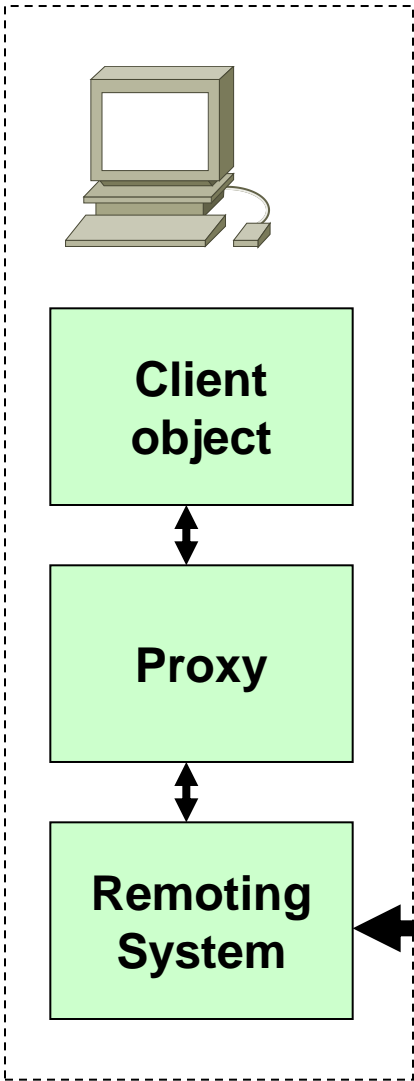
**Remotable  
Object  
(object.dll)**



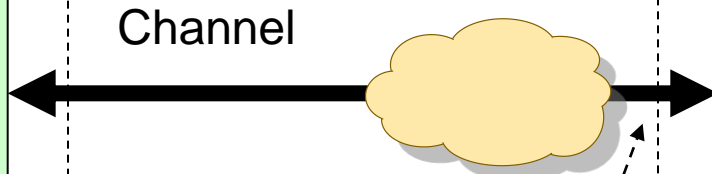
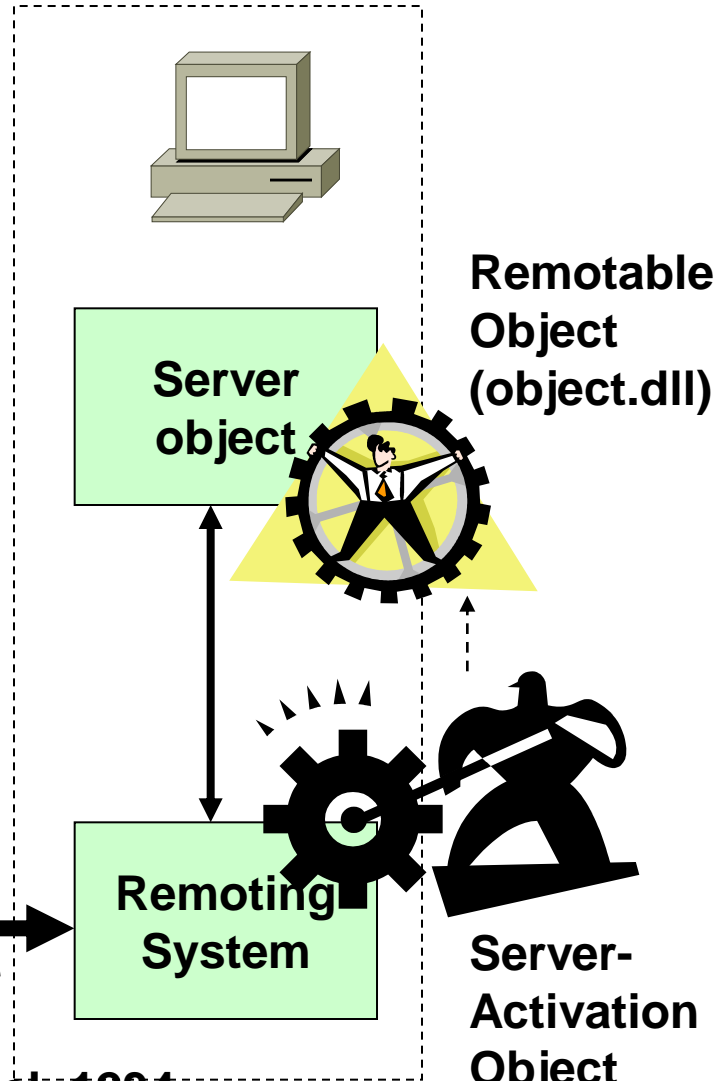
Channel



# Client Application Domain

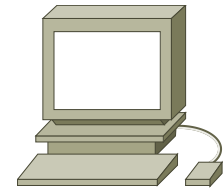


# Server Application Domain



Channel=1234

# Client Application Domain



Client object

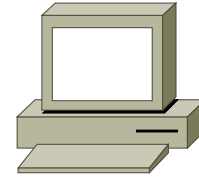
Proxy

Remoting System



Server invocation

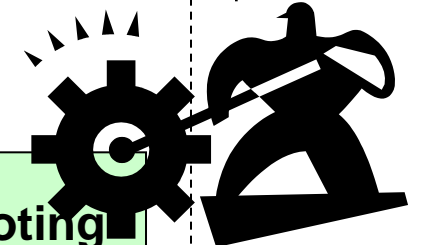
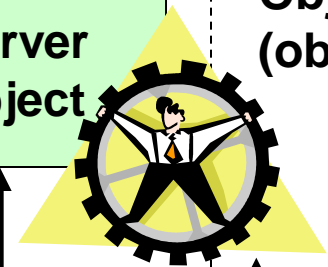
# Server Application Domain



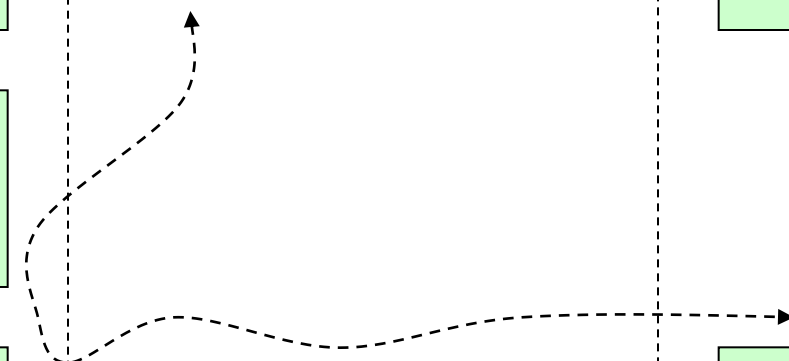
Server object

Remoting System

Remotable Object (object.dll)

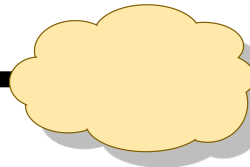


Server-Activation Object



Channel=1234

Channel=1234





# Channels

Bill Buchanan



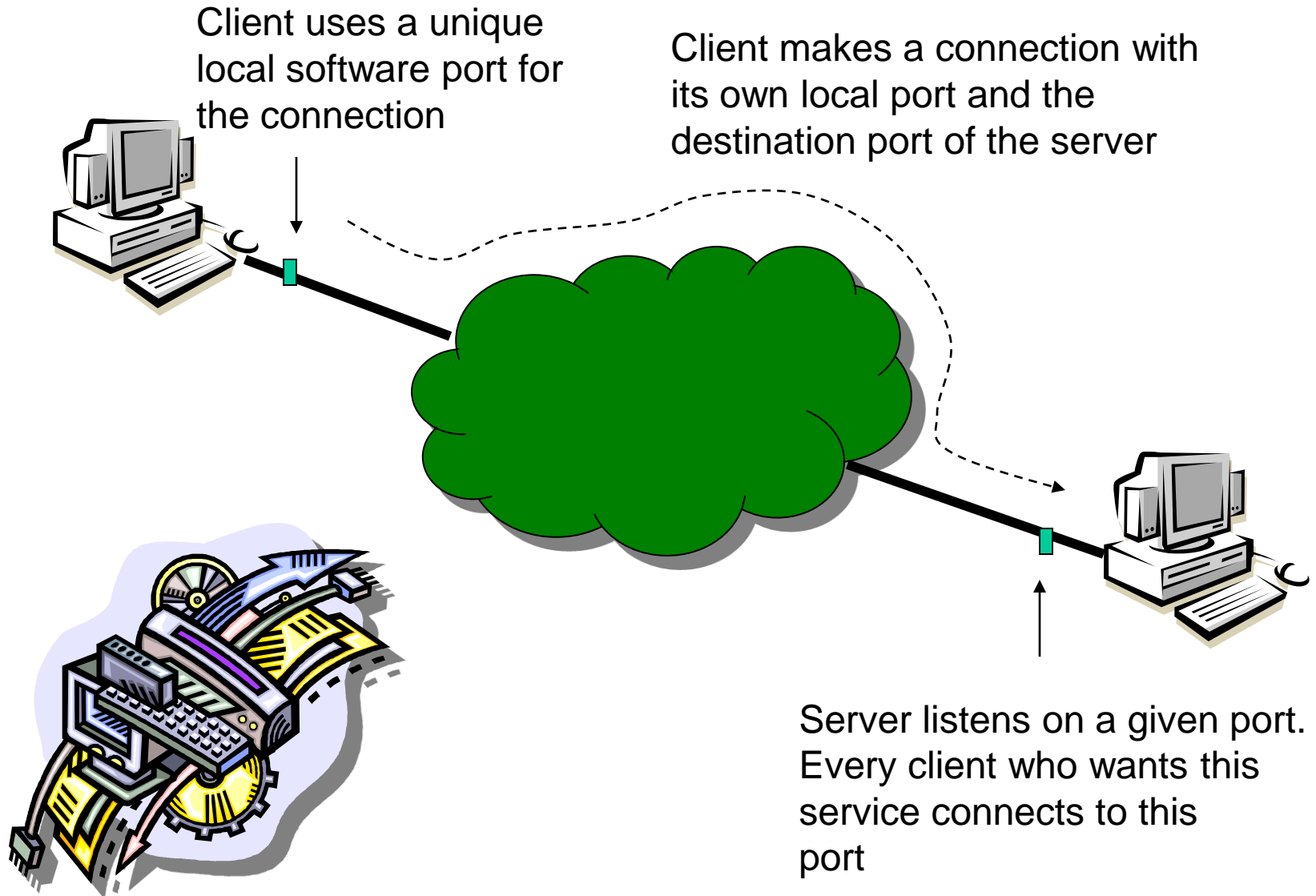


# TCP Ports

Bill Buchanan

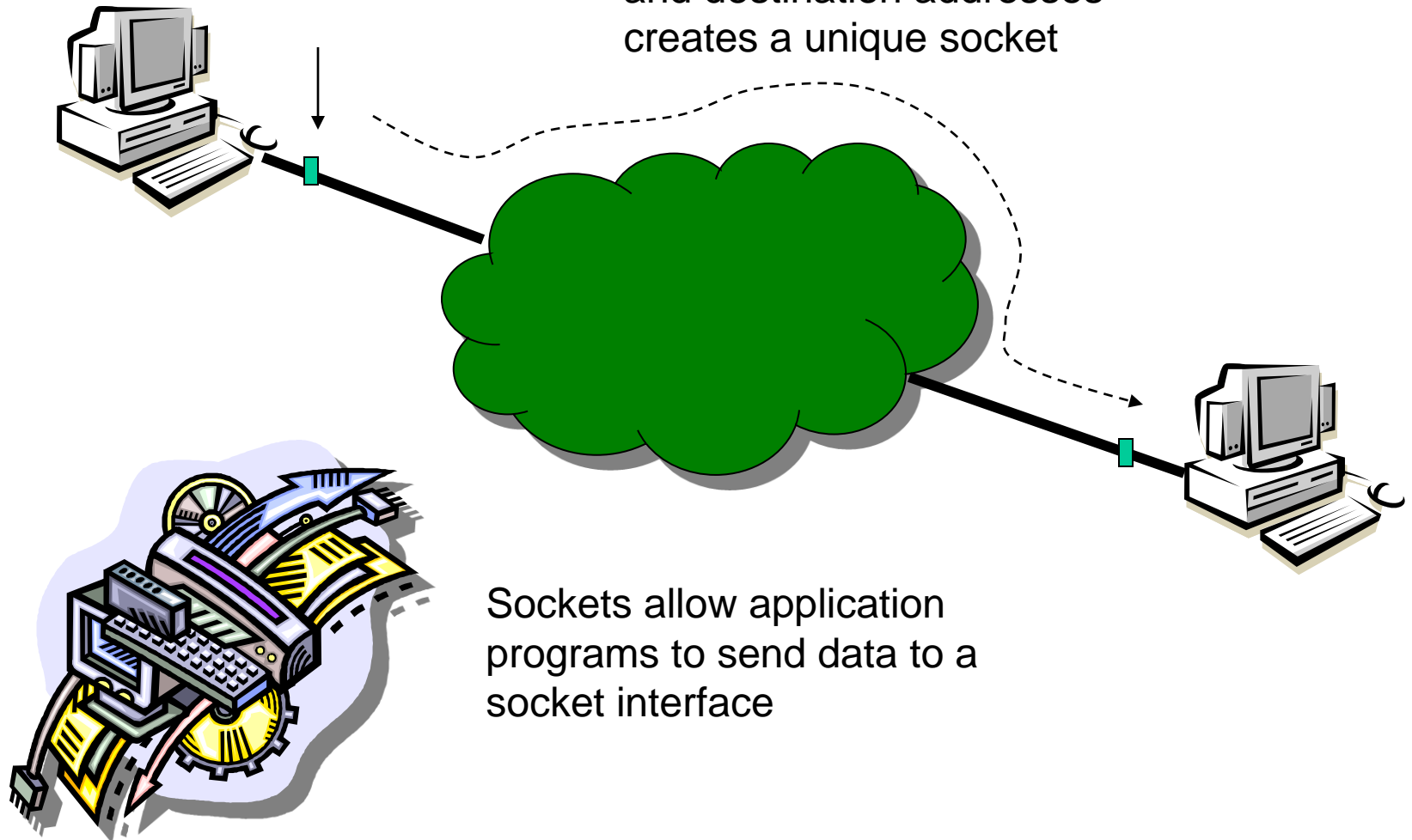


# TCP operation

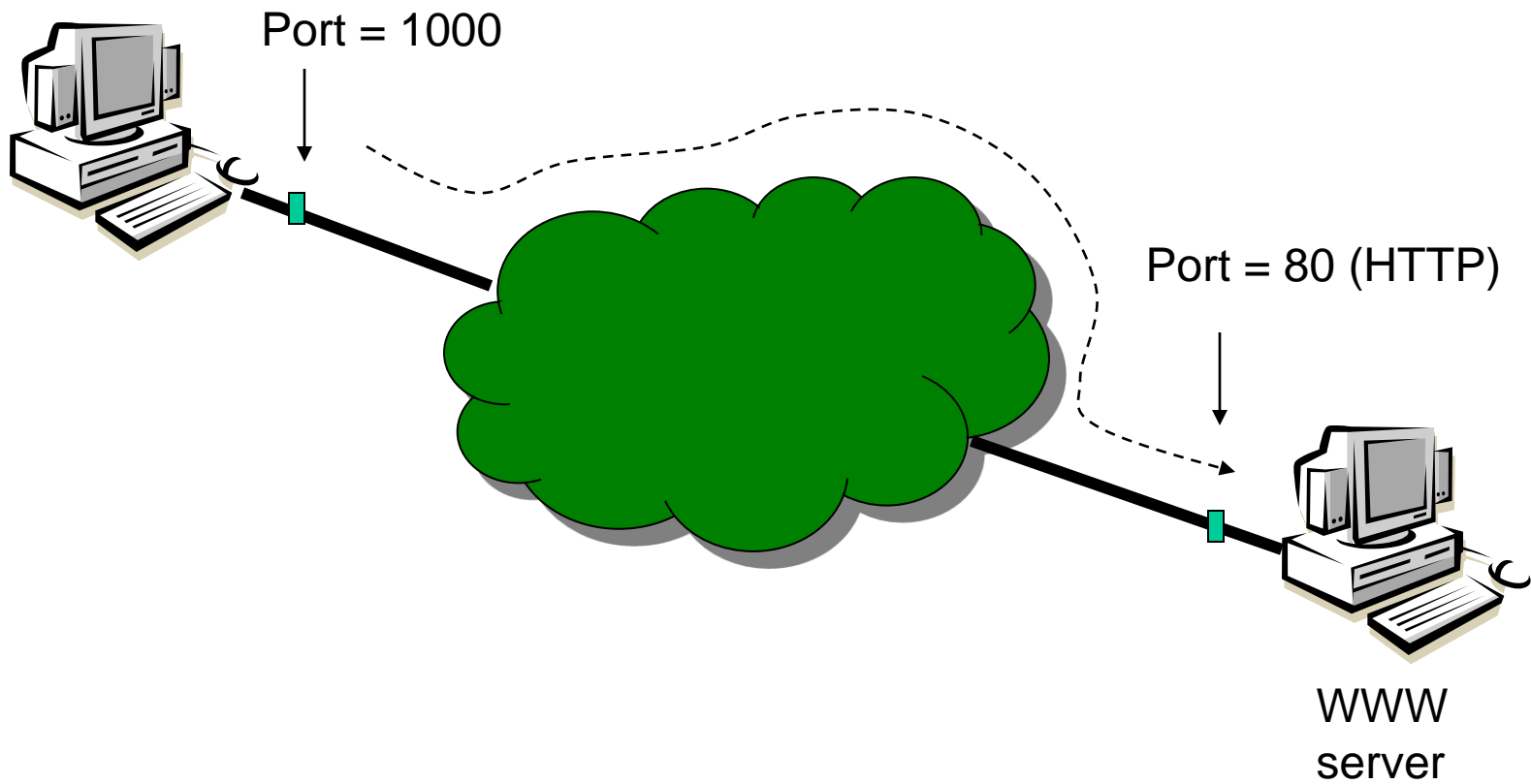


# TCP operation

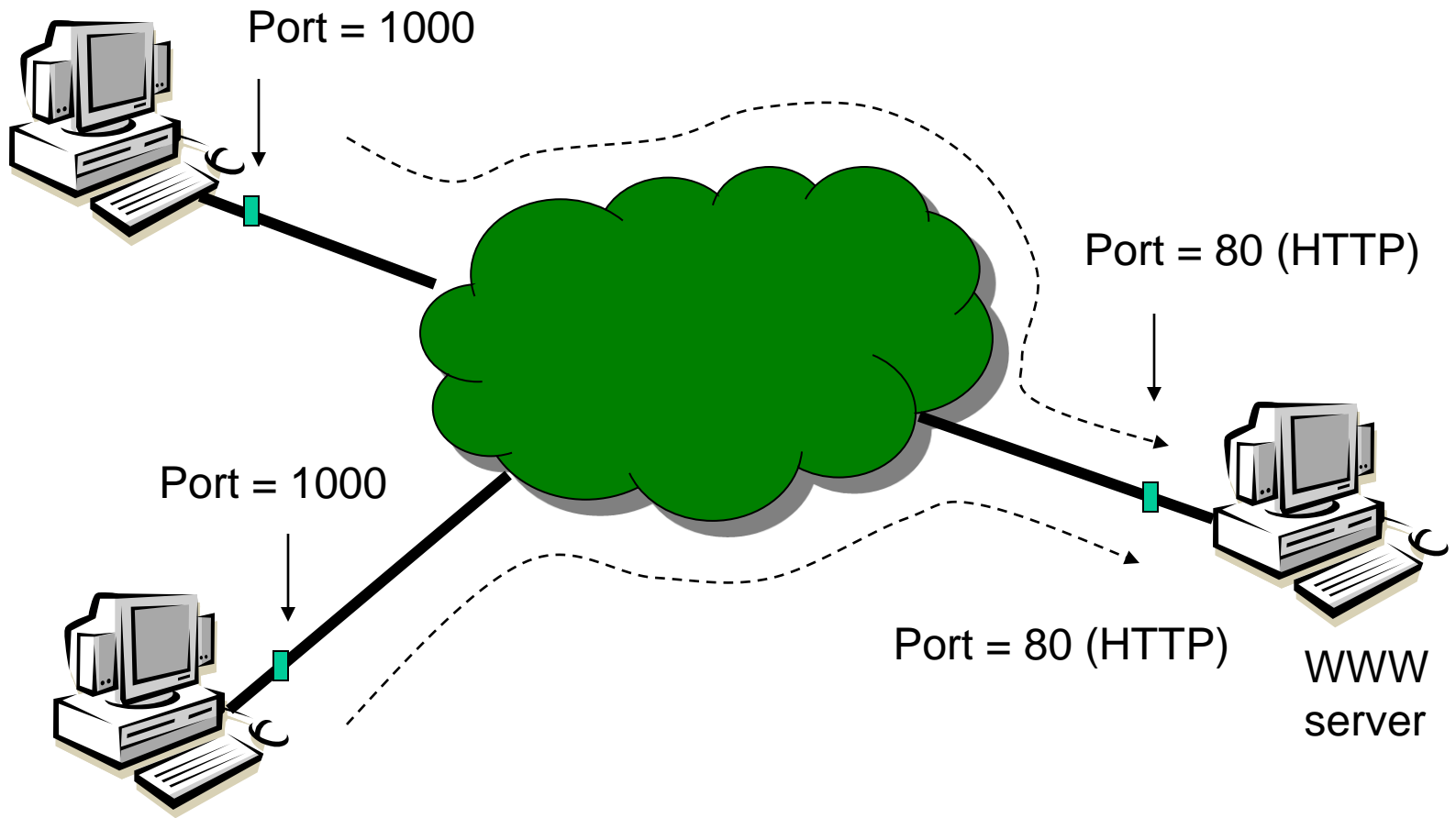
The binding of the local and destination port, and the local and destination addresses creates a unique socket



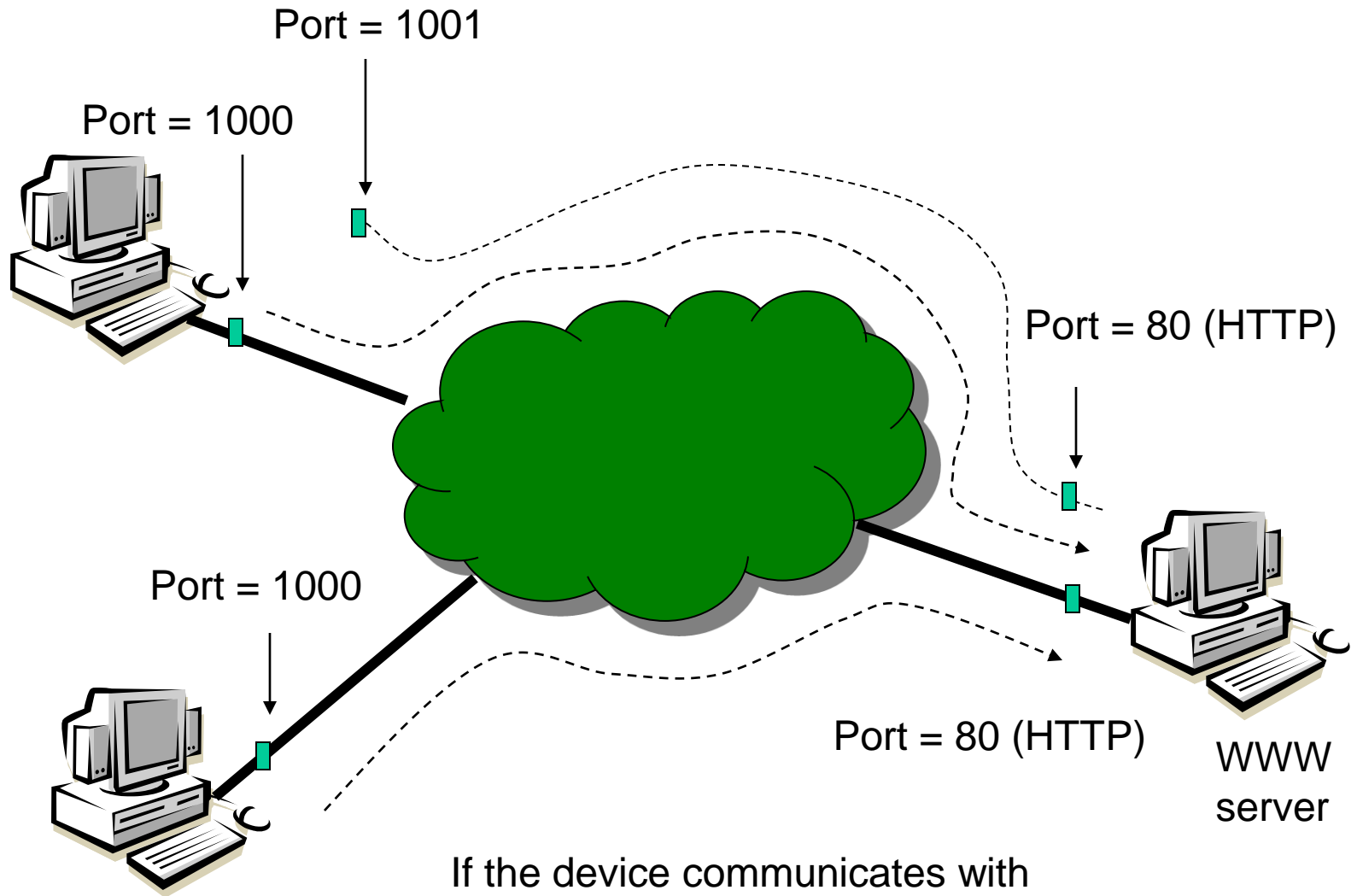
# Example



# Example

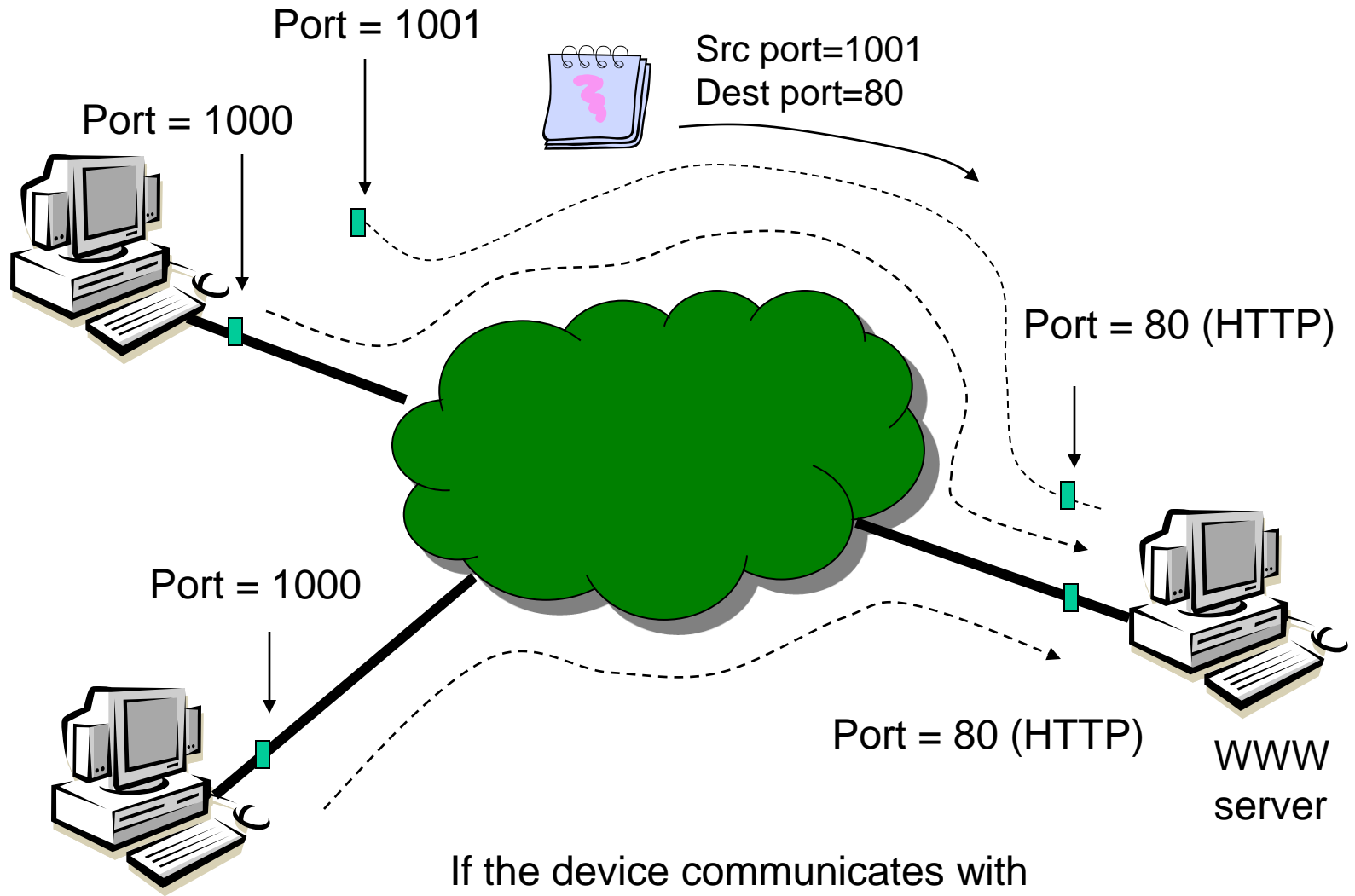


# Example



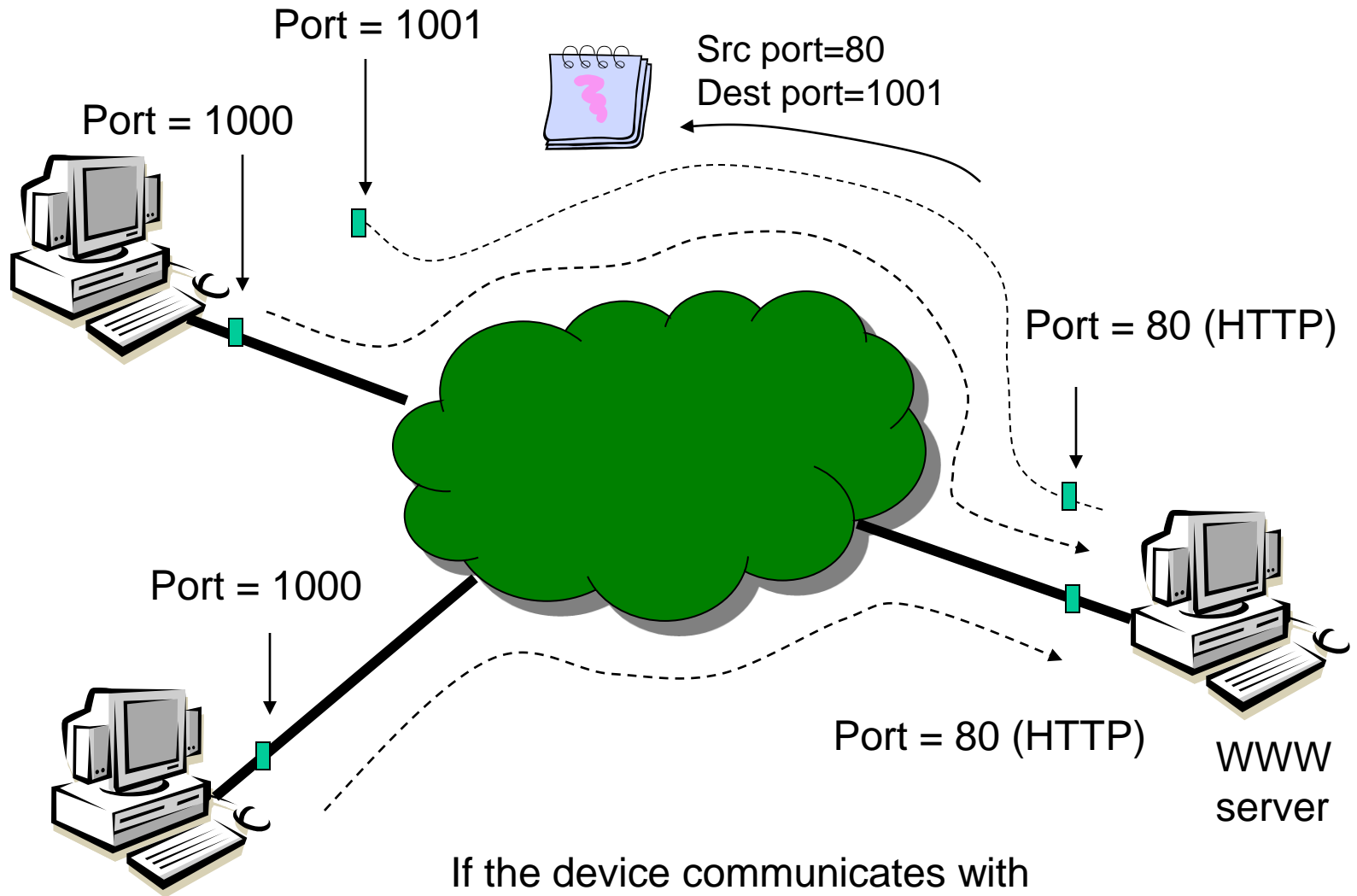
If the device communicates with the same server, it must create a new port (in this case, 1001)

# Example



If the device communicates with the same server, it must create a new port (in this case, 1001)

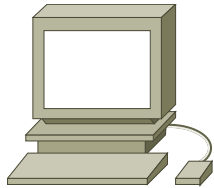
# Example



If the device communicates with the same server, it must create a new port (in this case, 1001)

# Activation

## Client Application Domain



**Client  
object**



**Proxy**



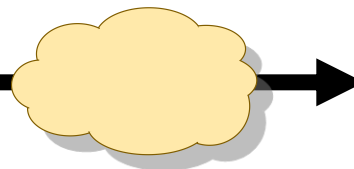
**Remoting  
System**

```
System.Runtime.Remoting.Channel.Http  
System.Runtime.Remoting.Channel.Tcp
```

```
TcpChannel channel = new TcpChannel(1234);
```

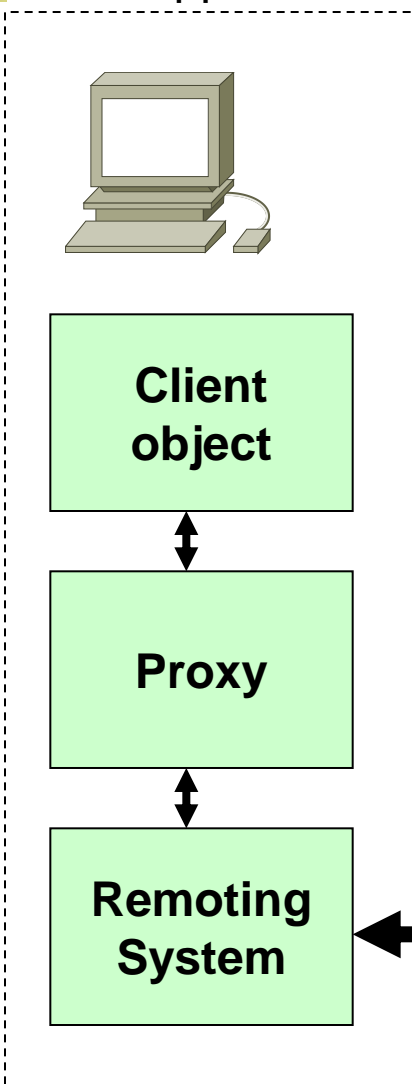
```
ChannelServices.RegisterChannel(channel);
```

Channel





## Client Application Domain

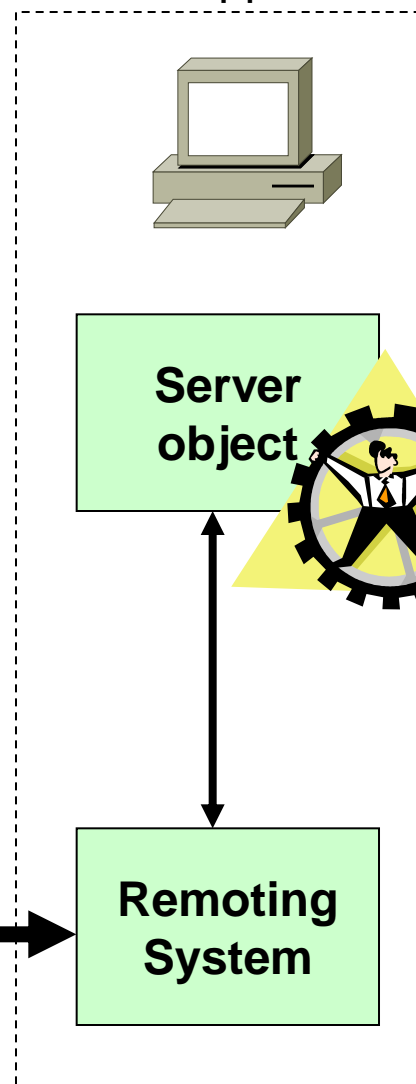


Channels map onto TCP ports.  
Standard ports:

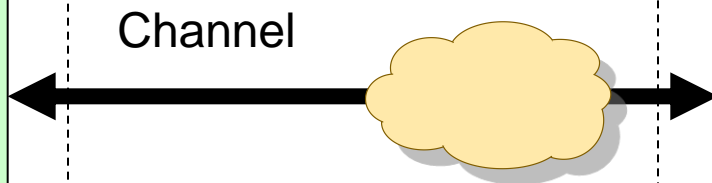
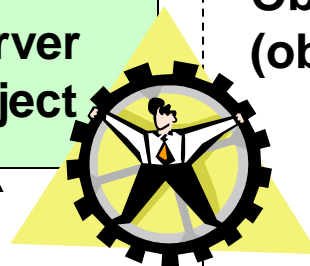
- 21 – FTP.
- 23 – TELNET.
- 56 – DNS.
- 80 – WWW.
- 110 – POP-3

Over 1024 for development

## Server Application Domain

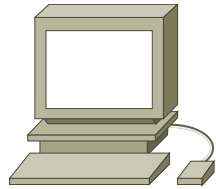


Remotable Object (object.dll)



# Activation

## Client Application Domain



**Client  
object**



**Proxy**

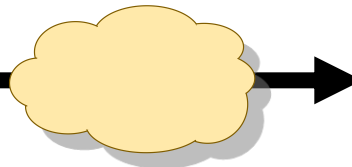


**Remoting  
System**

**Server activation** is typically used when remote objects **do not required to maintain their state** between method calls, or where there are multiple clients who call methods on the same object instance where the object maintains its state between function calls.

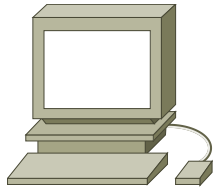
In a client-activated object, the client initiates the object and **manages it for its lifetime.**

Channel



# Activation

## Client Application Domain



**Client object**



**Proxy**



**Remoting System**

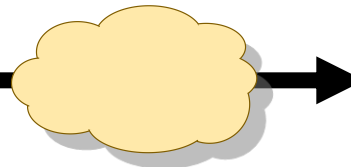
**Assembly name.** This defines the assembly in which the class is contained in.

**Type name.** This defines the data type of the remote object.

**Object URI.** This is the indicator that clients use to locate the object.

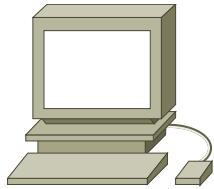
**Object mode.** This defines the server activation, such as SingleCall or Singleton.

Channel



# Activation

## Client Application Domain



**Client  
object**



**Proxy**



**Remoting  
System**

Remote objects are registered using the `RegisterWellKnownServiceType`, by passing the required parameters into the method, such as:

```
RemotingConfiguration.RegisterWellKnownServiceType  
(typeof(newclass.ShowCapital), "ShowCapital1",  
WellKnownObjectMode.SingleCall);
```

**Assembly name.** This defines the assembly in which the class is contained in.

**Type name.** This defines the data type of the remote object.

**Object URI.** This is the indicator that clients use to locate the object.

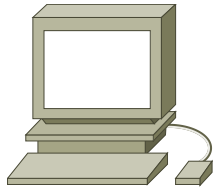
**Object mode.** This defines the server activation, such as `SingleCall` or `Singleton`.

Chanr



# Activation

## Client Application Domain



Client  
object



Proxy



Remoting  
System

... or with a configuration file with:

```
RemotingConfiguration.Configure(  
    "myconfig.config");
```

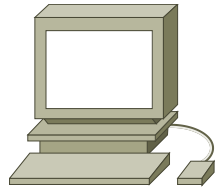
```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <system.runtime.remoting>  
    <application>  
      <service>  
        <wellknown mode="Singleton"  
type="newclass.ShowCapital, newclass"  
          objectUri="ShowCapital1" />  
      </service>  
      <channels>  
        <channel ref="tcp server" port="1234" />  
      </channels>  
    </application>  
  </system.runtime.remoting>  
</configuration>
```

C



# Initiation

## Client Application Domain



**Client  
object**



**Proxy**



**Remoting  
System**

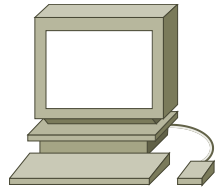
Once registered, the remote object will not instantiate itself, as this requires the client to initiate it, or call a method. This is achieved by a client which knows the URI of the remote object and by registering the channel it prefers using `GetObject`, such as:

```
TcpClientChannel channel = new TcpClientChannel();  
ChannelServices.RegisterChannel(channel);  
ShowCapital sh= (ShowCapital)  
    Activator.GetObject(  
        typeof(newclass.ShowCapital),  
        "tcp://localhost:1234/ShowCapital1");
```

"tcp://localhost:1234/ShowCapital1". Specifies that the end point is ShowCapital using TCP port 1234.

# Initiation

## Client Application Domain



Client  
object



Proxy



Remoting  
System

Along with this, the compiler requires type information about the ShowCapital class when this client code is compiled. This can be defined with **one** of the following:

- With a reference to the assembly where the ShowCapital class is stored.
- Using **SOAPSUDS** tool to extract metadata directly from the endpoint. SOAPSUDS connects to the endpoint, and extracts the metadata, and generates an assembly or source code that is then used in the client compilation.
- By splitting the remote object into an **implementation** and **interface class** and then use the interface as a reference when compiling the client.

"tcp://localhost:1234/ShowCapital1". Specifies that the end point is ShowCapital using TCP port 1234.



## Creating a remotable class

Bill Buchanan

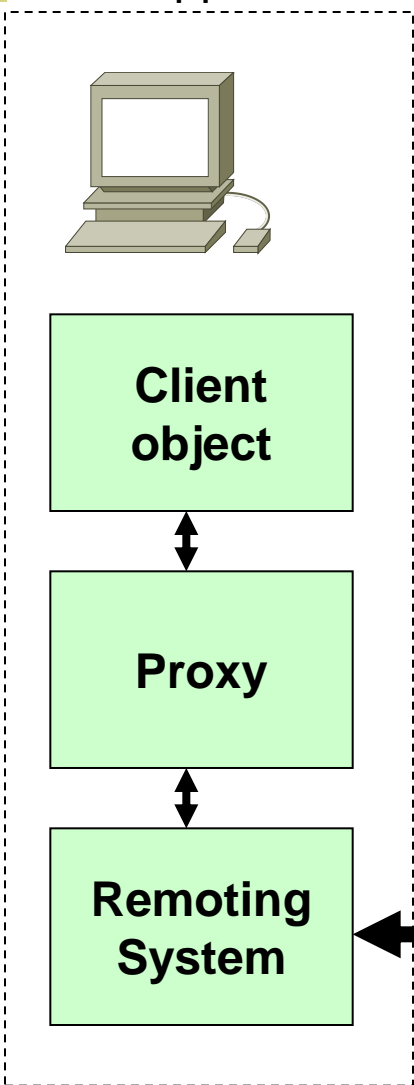




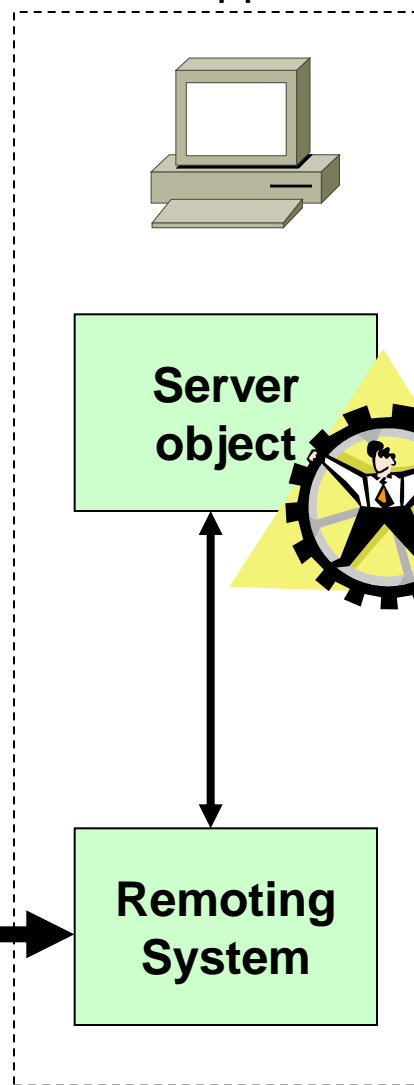
```
using System;
using System.Data;
namespace newclass
{
    public class ShowCapital : MarshalByRefObject
    {
        public ShowCapital()
        {
        }
        public string show(string country)
        {
            if (country.ToLower()=="england") return("London");
            else if (country.ToLower()=="scotland")
                return("Edinburgh");
            else return("Not known");
        }
    }
}
```



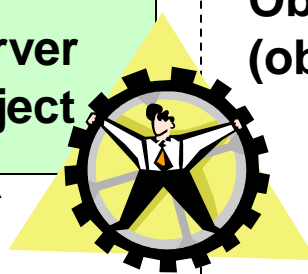
## Client Application Domain



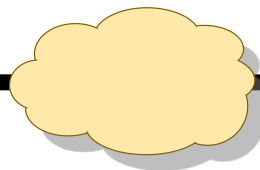
## Server Application Domain



**Remotable  
Object  
(object.dll)**



Channel





### Add New Project

Project Types:

- Agilent T&M Toolkit Projects
- Visual Basic Projects
- Visual C# Projects
- Visual J# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects

Templates:

- T&M Toolkit Project Wizard
- TM Developer Windows Form
- Windows Application
- Class Library
- TM Developer Class Library
- Windows Control Library

A project for creating classes to use in other applications

Name: newclass

Location: C:\AgilentTraining\70-320\src\dotnetre...

Project will be created at C:\AgilentTraining\70-320\...

### Solution Explorer - newclass

Solution 'dotnetremote1' (1 project)

- newclass
  - References
    - AssemblyInfo.cs
    - ShowCapital.cs

```
public Show...  
{  
}  
}  
public string  
{  
    if (count:  
    else if (  
    else retu  
}  
}
```

Index Results for TcpServerChannel class - 1 topics found

Ready

### Add Reference

.NET | COM | Projects

Component Name	Version	Path
System.Drawing.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.EnterpriseServices	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Management	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Messaging.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
<b>System.Runtime.Remoting</b>	<b>1.0.5000.0</b>	<b>C:\WINDOWS\Microsoft.NET\F...</b>
System.Runtime.Serialization.Fo...	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Security	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.ServiceProcess.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Web.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Web.Mobile.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Web.RegularExpression...	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...
System.Web.Services.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET\F...

Selected Components:

Component Name	Type	Source
System.Runtime.Remoting	.NET	C:\WINDOWS\Microsoft.NET\Fra...

Buttons: Browse..., Select, Remove, OK, Cancel, Help



newclass - Microsoft Visual C# .NET [design] - ShowCapital.cs\*

File Edit View Project Build Debug Tools T&M Toolkit Window Help

Object Browser ShowCapital.cs\* | newclass.ShowCapital | ShowCapital()

```
using System;
using System.Data;

namespace newclass
{
    public class ShowCapital : MarshalByRefObject
    {
        public ShowCapital()
        {
        }
        public string show(string country)
        {
            if (country.ToLower()=="england") return("London");
            else if (country.ToLower()=="scotland") return("Edinburgh");
            else return("Not known");
        }
    }
}
```

Solution Explorer - newclass

- Solution 'dotnetremotel' (1 project)
- newclass
  - References
  - AssemblyInfo.cs
  - ShowCapital.cs

Task List - 0 Build Error tasks shown (fil... X

Description
Click here to add a new task

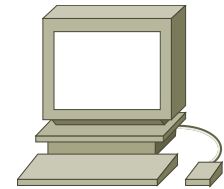
Search Results

Properties

Index Results for TcpServerChannel class - 1 topics found

Ready | Ln 11 | Col 1 | Ch 1 | INS

# Client Application Domain



Client object

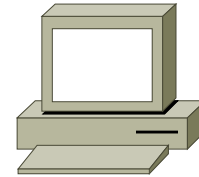


Proxy

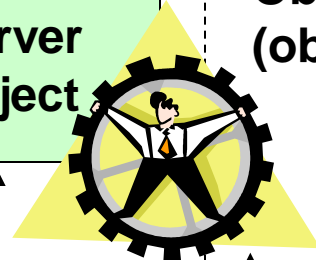


Remoting System

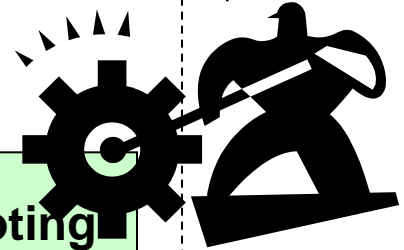
# Server Application Domain



Server object



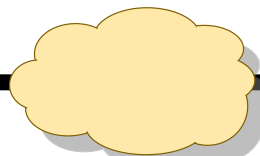
Remoting System



Remotable Object (object.dll)

Server-Activation Object

Channel



Channel=1234

Channel=1234



### Add New Project

**Project Types:**

- Agilent T&M Toolkit Projects
- Visual Basic Projects
- Visual C# Projects
- Visual J# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects

**Templates:**

- ASP.NET Mobile Web Application
- Web Control Library
- Console Application

A project for creating a command-line application

Name:

Location:

Project will be created at C:\AgilentTraining\70-320\src\dotnetremote1\newclass2\

### Add Reference

.NET | COM | Projects

Project Name	Project Directory
newclass	C:\AgilentTraining\70-320\src\dotnetremote1\newclass2\

Selected Components:

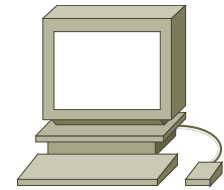
Component Name	Type	Source
System.Runtime.Remoting	.NET	C:\WINDOWS\Microsoft.NET\Framework\2.0.50727\System.Runtime.Remoting.dll
newclass	Project	C:\AgilentTraining\70-320\src\dotnetremote1\newclass2\

Buttons: Browse..., Select, Remove, OK, Cancel, Help



```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
namespace newclass2
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            TcpServerChannel channel = new TcpServerChannel(1234);
            ChannelServices.RegisterChannel(channel);
            RemotingConfiguration.RegisterWellKnownServiceType
                (typeof(newclass.ShowCapital), "ShowCapital",
                 WellKnownObjectMode.SingleCall);
            Console.WriteLine("Starting...");
            Console.ReadLine();
        }
    }
}
```

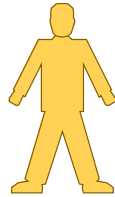
# Client Application Domain



Client object

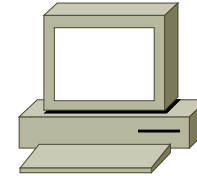
Proxy

Remoting System



Server invocation

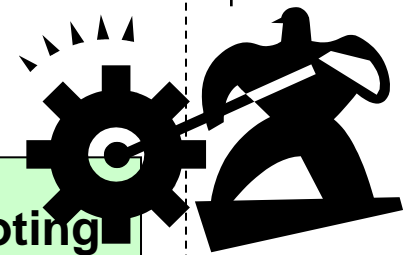
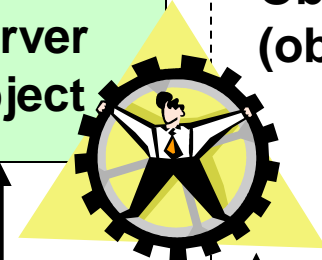
# Server Application Domain



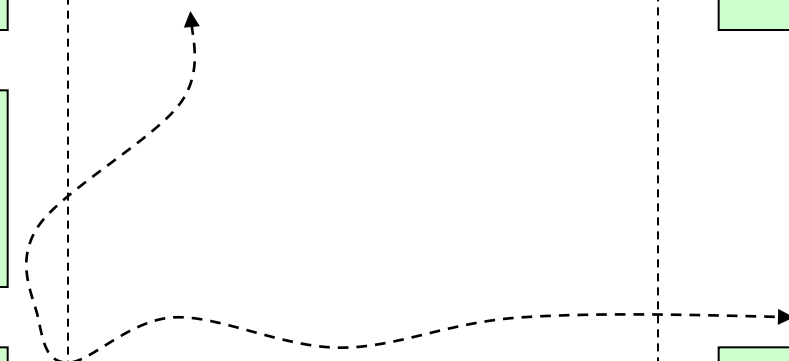
Server object

Remoting System

Remotable Object (object.dll)



Server-Activation Object



Channel=1234

Channel=1234

### Add New Project

**Project Types:**

- Agilent T&M Toolkit Projects
- Visual Basic Projects
- Visual C# Projects
- Visual J# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects

**Templates:**

- T&M Toolkit Project Wizard
- TM Developer Windows Form
- Windows Application

A project for creating an application with a Windows user interface

**Name:** WindowsApplication1

**Location:** C:\AgilentTraining\70-320\src\dotnetremote1

Project will be created at C:\AgilentTraining\70-320\src\dotnetremote1

OK

### Add Reference

.NET | COM | Projects

Project Name	Project Directory
newclass	C:\AgilentTraining\70-320\src\dotnetremote1\n...
newclass2	C:\AgilentTraining\70-320\src\dotnetremote1\n...

Browse...  
Select

**Selected Components:**

Component Name	Type	Source
System.Runtime.Remoting	.NET	C:\WINDOWS\Microsoft.NET\Fra...
newclass	Project	C:\AgilentTraining\70-320\src\do...

Remove

OK Cancel Help



```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using newclass;
    ShowCapital sh;
    private void button1_Click(object sender, System.EventArgs e)
    {
        string country, cap;
        country=textBox1.Text;
        cap=sh.show(country);
        textBox2.Text= cap;
    }
    private void Form1_Load(object sender, System.EventArgs e)
    {
        TcpClientChannel channel = new TcpClientChannel();
        ChannelServices.RegisterChannel(channel);
        RemotingConfiguration.RegisterWellKnownClientType(
            typeof(ShowCapital),
            "tcp://localhost:1234/ShowCapital");
        sh= new ShowCapital();
    }
```



### Solution 'dotnetremote1' Property Pages

Configuration:  Platform:  [Configuration Manager...](#)

- Common Properties
  - Startup Project**
  - Project Dependencies
  - Debug Source Files
  - Debug Symbol Files
- Configuration Properties

Single Startup Project

Multiple Startup Projects

Project	Action
newclass	None
newclass2	Start
WindowsApplication1	Start

[Move Up](#)  
[Move Down](#)

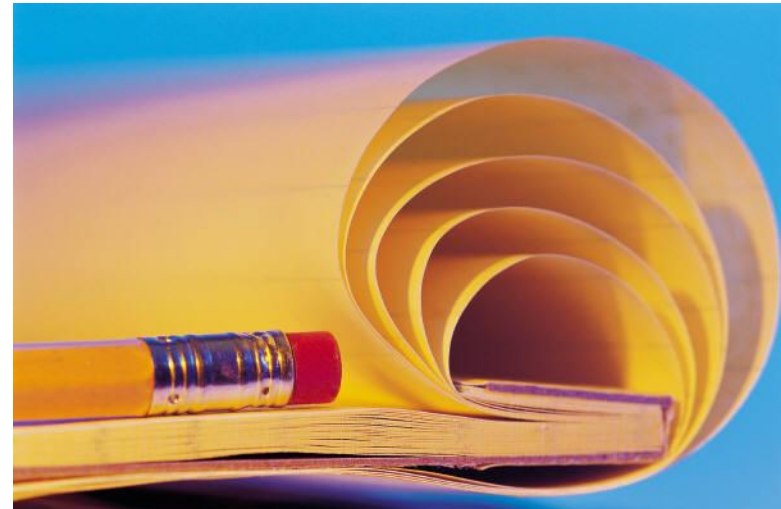
[OK](#) [Cancel](#) [Apply](#) [Help](#)





# Tutorial Session:

Page 19 and 20





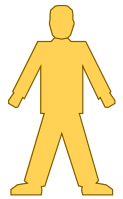
## Client-activated objects

Bill Buchanan



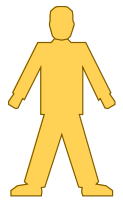


The lifetime of server-activated objects (**SAO**) is directly controlled by the **server**, whereas the lifetime of client-activated objects (**CAO**) is controlled by the calling application program, just as if it were local to the client.





The lifetime of server-activated objects (**SAO**) is directly controlled by the **server**, whereas the lifetime of client-activated objects (**CAO**) is controlled by the calling application program, just as if it were local to the client.



An SAO the remotable object is defined with:

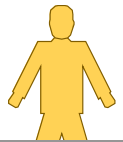
```
RemotingConfiguration.RegisterWellKnownServiceType  
    (typeof(newclass.ShowCapital), "ShowCapital",  
     WellKnownObjectMode.SingleCall);
```

A CAO the remotable object is defined with:

```
RemotingConfiguration.RegisterActivatedServiceType  
    (typeof(newclass.ShowCapital));
```



The lifetime of server-activated objects (**SAO**) is directly controlled by the **server**, whereas the lifetime of client-activated objects (**CAO**) is controlled by the calling application program, just as if it were local to the client.



When initiating and invoking a SAO:

```
RemoteConfiguration.RegisterWellKnownClientType(  
    typeof(ShowCapital),  
    "tcp://localhost:1234/ShowCapital");
```

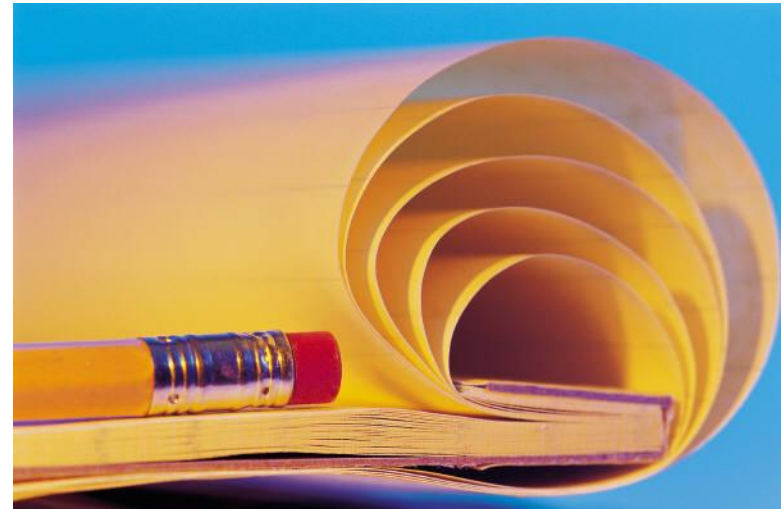
When initiating and invoking a CAO:

```
RemoteConfiguration.RegisterActivatedClientType(  
    (typeof(newclass.ShowCapital),  
    "tcp://localhost:1234");
```



# Tutorial Session:

Page 23





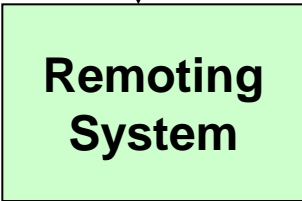
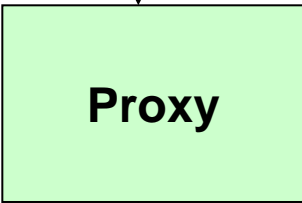
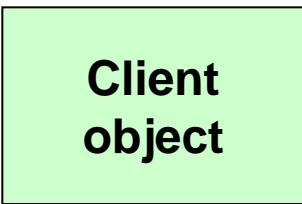
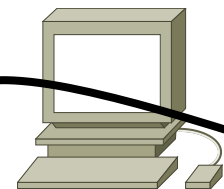
# Configuration Files for remoting

Bill Buchanan



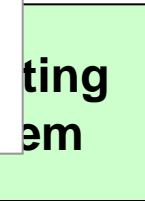
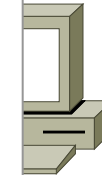


## Client Application



```
<configuration>  
  <system.runtime.remoting>  
    <application>  
      <lifetime>  
      </lifetime>  
      <service>  
        <wellknown/>  
        <activated />  
      </service>  
      <client>  
        <wellknown/>  
        <activated />  
      </client>  
      <channels>  
      </channels>  
    </application>  
  </system.runtime.remoting>  
</configuration>
```

## Application Domain





```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton"
          type="newclass.ShowCapital, newclass"
          objectUri="ShowCapital1" />
      </service>
    </application>
    <channels>
      <channel ref="tcp server" port="1234" />
    </channels>
  </system.runtime.remoting>
</configuration>
```

```
static void Main(string[] args)
{
  RemotingConfiguration.Configure(
    "..//..//dotnetremot1.exe.config");
  Console.WriteLine("Starting...");
  Console.ReadLine();
}
```



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton"
                    type="newclass.ShowCapital, newclass"
                    objectUri="ShowCapital1" />
      </service>
    <channels>
      <channel ref="tcp server" port="1234" />
    </channels>
  </application>
</system.runtime.remoting>
</configuration>
```

### Which achieves:

```
RemotingConfiguration.RegisterWellKnownServiceType
    (typeof(newclass.ShowCapital),
     "ShowCapital1", WellKnownObjectMode.Singleton);
```



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <wellknown type="newclass.ShowCapital, newclass"
          url="tcp://localhost:1234/ShowCapital" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

```
private void Form1_Load(object sender, System.EventArgs e)
{
  RemotingConfiguration.Configure("app2.config");
  sh= new ShowCapital();
}
```



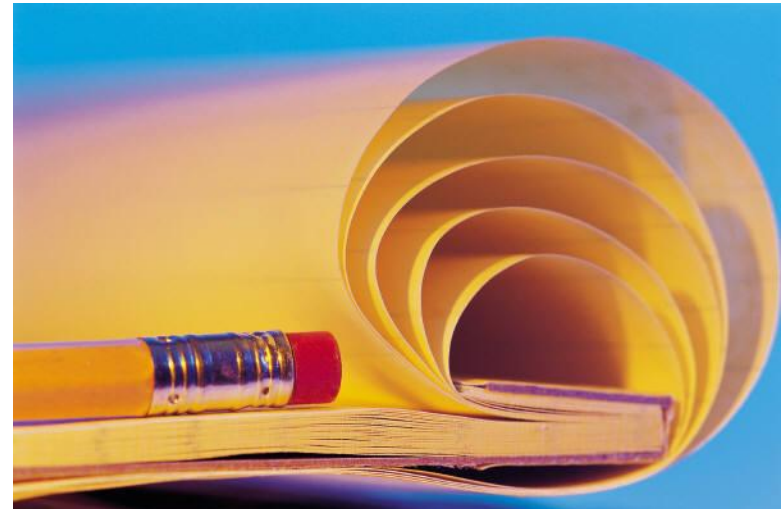
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton"
          type="newclass.ShowCapital, newclass"
          objectUri="ShowCapital1" />
      </service>
    </application>
  </system.runtime.remoting>
</configuration>
```

**Which achieves:**

```
private void Form1_Load(object sender, System.EventArgs e)
{
    RemotingConfiguration.Configure("app2.config");
    sh= new ShowCapital();
}
```



# Adding Interface Definitions





```
using System;
```

```
namespace IDCapital
```

```
{
```

```
    public interface IDCap
```

```
    {
```

```
        string show(string str);
```

```
    }
```

```
}
```



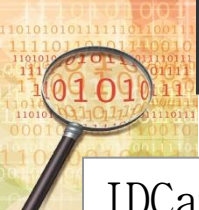
```
using System;
using System.Data;
using IDCapital;

namespace newclass
{
    public class ShowCapital : MarshalByRefObject, IDCapital.IDCap
    {

        public string show(string country)
        {
            if (country.ToLower()=="england") return("London");
            else if (country.ToLower()=="scotland") return("Edinburgh");
            else return("Not known");
        }
    }
}
```

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using IDCapital;

namespace newclass2
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            TcpServerChannel channel = new TcpServerChannel(1234);
            ChannelServices.RegisterChannel(channel);
            RemotingConfiguration.RegisterWellKnownServiceType
                (typeof(newclass.ShowCapital), "ShowCapital",
                 WellKnownObjectMode.SingleCall);
            Console.WriteLine("Starting...");
            Console.ReadLine();
        }
    }
}
```



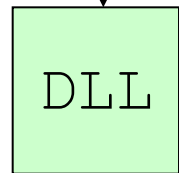
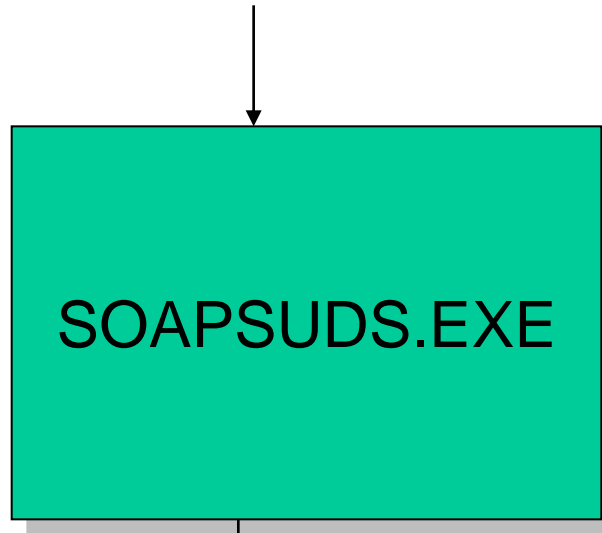
```
IDCap sh;
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    string country, cap;
    country=textBox1.Text;
    cap=sh.show(country);
    textBox2.Text= cap;
}
private void Form1_Load(object sender, System.EventArgs e)
{
    TcpClientChannel channel = new TcpClientChannel();
    ChannelServices.RegisterChannel(channel);
    sh= (IDCapital.IDCap) Activator.GetObject(typeof(IDCap),
    "tcp://localhost:1234/ShowCapital");
}
```



```
soapsuds -url:http://localhost:1234/ShowCapital?wsdl  
-oa:newclass.dll -nowp
```

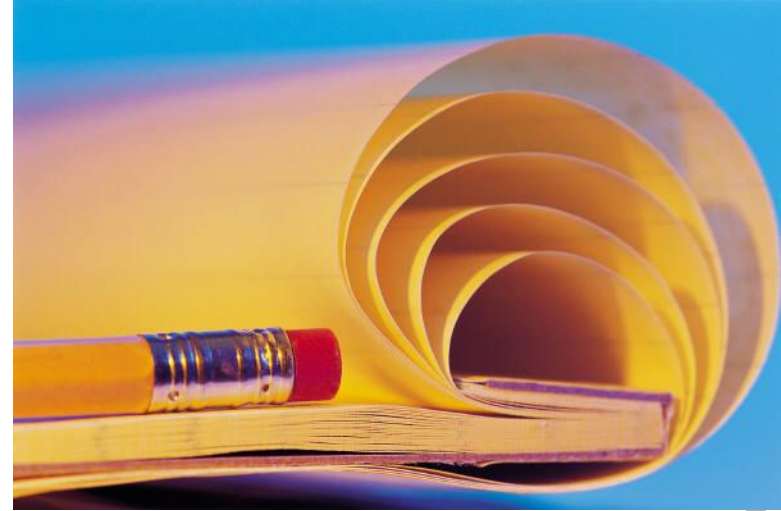
**Limitation:**  
Can only be  
used with HTTP  
and a Web Server



Add  
reference in  
project



# Cram





1. Client object requests an instance of the server object. The remoting system creates a proxy of the server object on the client side.
2. The client calls a methods on the server object, and the proxy sends the call through. The remoting system then sends a call to the server remoting system on the server.
3. The remoting system on the server receives the call, and invokes the object on the server.
4. The remote object on the server returns back the result to the remoting system on the server, which sends it to the remoting system on the client.
5. The remoting system on the client sends result to the proxy which forwards it to the client object.



Object marshaling: defines how a remote object is exposed to client applications

Marshal-by-value (MBV) objects: These are when objects are copied and passed from the server application domain to the client application domain

Which best defines a Marshal-by-reference (MBR) objects: These are when a proxy is used to access a server object, where the client keeps references to the objects

Marshal-by-value objects: They reside on the server, and are serialized, and sent over the network to the client. The access is then done locally on the client.



Marshal-by-value object:

```
[Serializable()]  
public class MyRemoteObject  
{  
    // ...  
}
```

Marshal-by-reference:

```
public class MyRemoteObject: MarshalByRefObject  
{  
    // ...  
}
```



Small objects, which type is likely to be the fastest:  
**Marshal-by-value**

Large objects, which type is likely to be the fastest:  
**Marshal-by-reference**

Objects which are only available on the server:  
**Marshal-by-reference**



Which is the most efficient methods of channels and formatting:  
**TCP channel, with a binary formatter**

Which is the least efficient methods of channels and formatting:  
**HTTP channel, with a SOAP formatter**

Which is the best methods of channels and formatting for interoperability:  
**HTTP channel, with a SOAP formatter**

Which is the worst methods of channels and formatting for interoperability:  
**TCP channel, with a binary formatter**

Default: TCP is BINARY

Default; HTTP is SOAP

The lifetime of the SAO is controlled by the server, while the lifetime of a CAO is controlled by the client