WCF Architecture

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Outline

- **WCF design goals**
- **Building blocks and layers**
- **Writing services and consumers**

WCF design goals

- **Communication stack for service-based applications**
  - built with the four tenets in mind
  - rich extensibility and metadata support
  - messaging-based
- **Unified communication API**
  - different communication APIs had different features sets (sockets, DCOM, Remoting, ASMX, MSMQ…)
- **Common feature set over arbitrary transports**
  - e.g. security, transactions, reliability

- **F == Foundation (!= Framework)**

WCF fundamental concepts

- **Messages**
- **Channels**
- **Encoders**

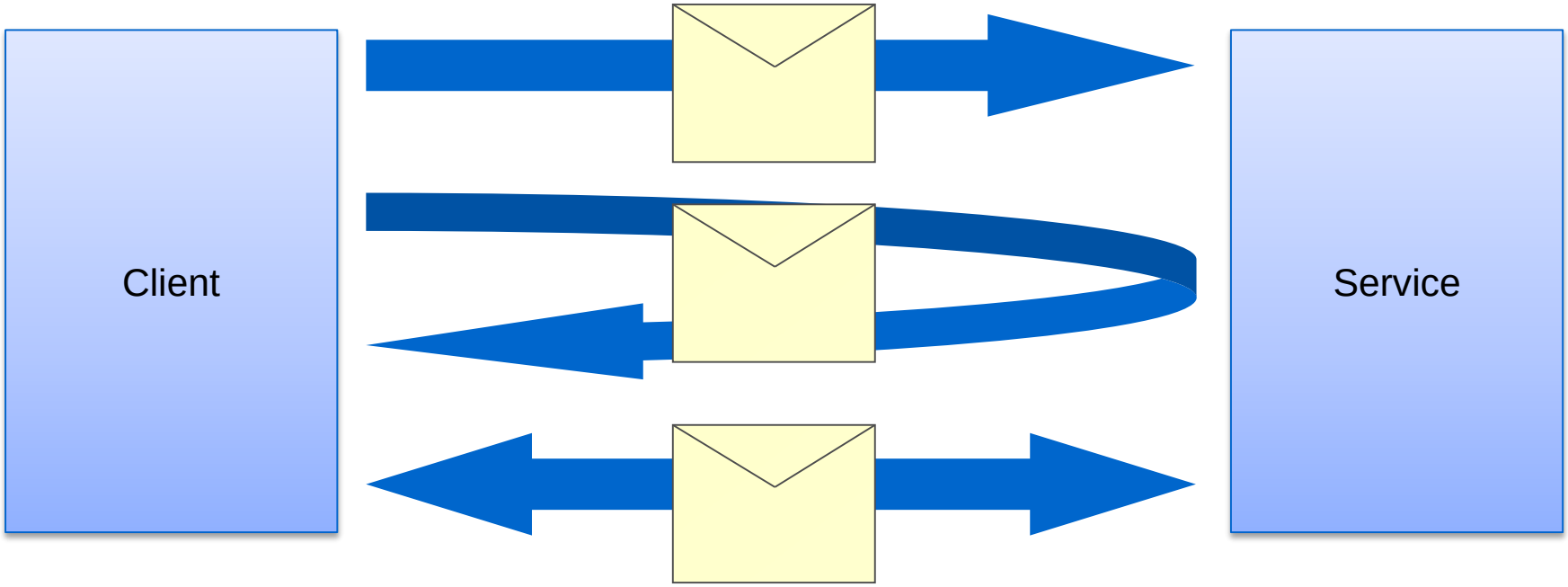
Messages

- **Messages are the smallest unit of transmission**
  - modelled on SOAP messages (InfoSet)
    - envelope, header, body
    - addressing
- **Message modeled in `System.ServiceModel.Channels.Message` class**
- **Messages have a SOAP and addressing version number**
  - `MessageVersion` class

```
public sealed class MessageVersion
{
  …
  public static MessageVersion Default { get; }
  public static MessageVersion None { get; }
  public static MessageVersion Soap11 { get; }
  public static MessageVersion Soap11WSAddressing10 { get; }
  public static MessageVersion Soap11WSAddressingAugust2004 { get; }
  public static MessageVersion Soap12 { get; }
  public static MessageVersion Soap12WSAddressing10 { get; }
  public static MessageVersion Soap12WSAddressingAugust2004 { get; }
}
```

Messages

- **WCF applications exchange messages**
- **Message Exchange Pattern (MEP) describes how messages are exchanged**
  - one-way, request response, duplex



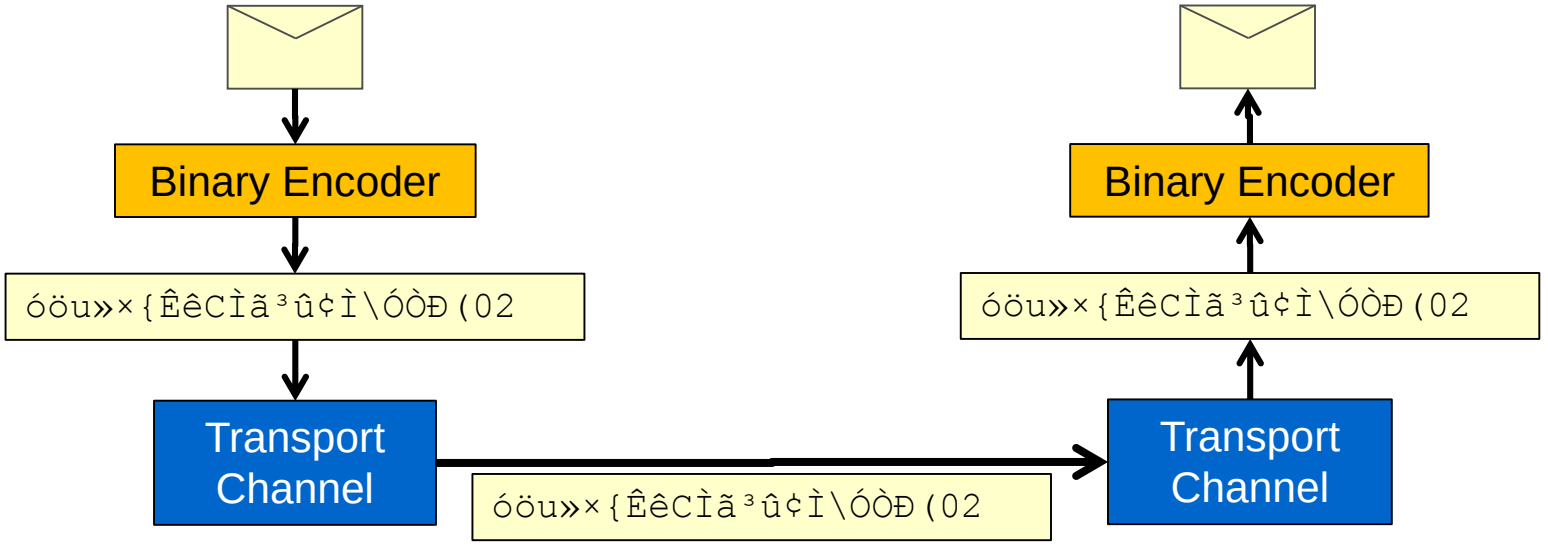
Channels

- **Messages are the payload**
- **Channels provide the transmission system**
- **Protocol channels**
  - layer in services independent of transport
  - security, transactions, reliable messaging (e.g. based on WS-*)
  - extensible
- **Transport channels**
  - physically manage the movement of bytes
  - HTTP, TCP, MSMQ, P2P, Named Pipes
  - extensible

Encoders

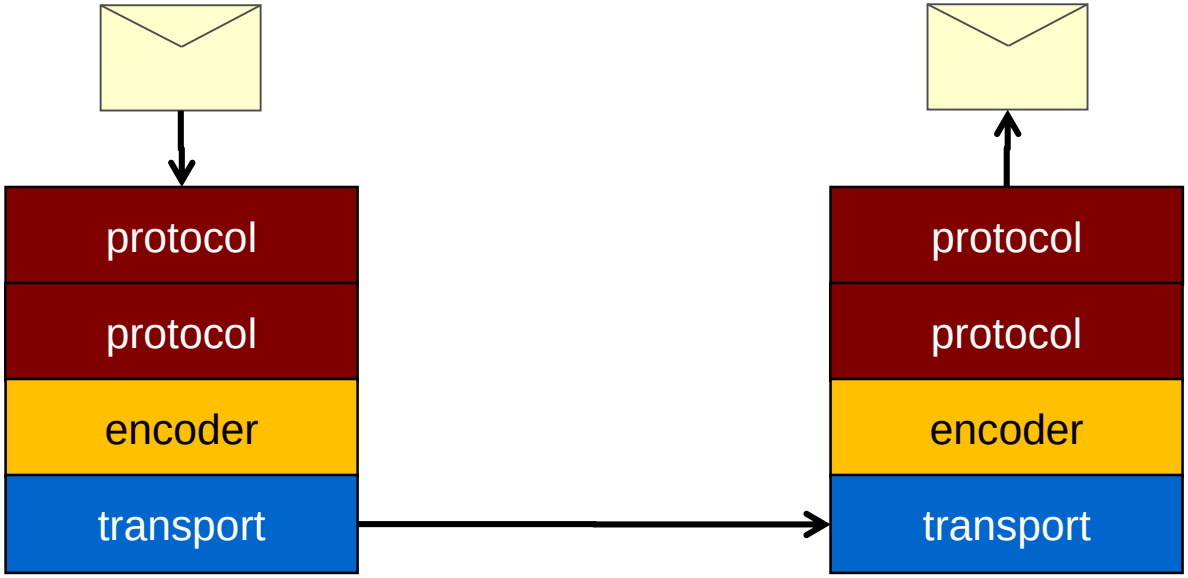
- **Encoders turn Infosets into a stream of bytes**

Text	Interoperable textual XML
JSON/POX	Interoperable textual XML/JSON
MTOM	Interoperable textual XML with binary attachments
Binary	Non-interoperable compact



Channel stacks

- **Functionality is composed by chaining channels**
  - channels form a channel stack



Channel: the socket of WCF

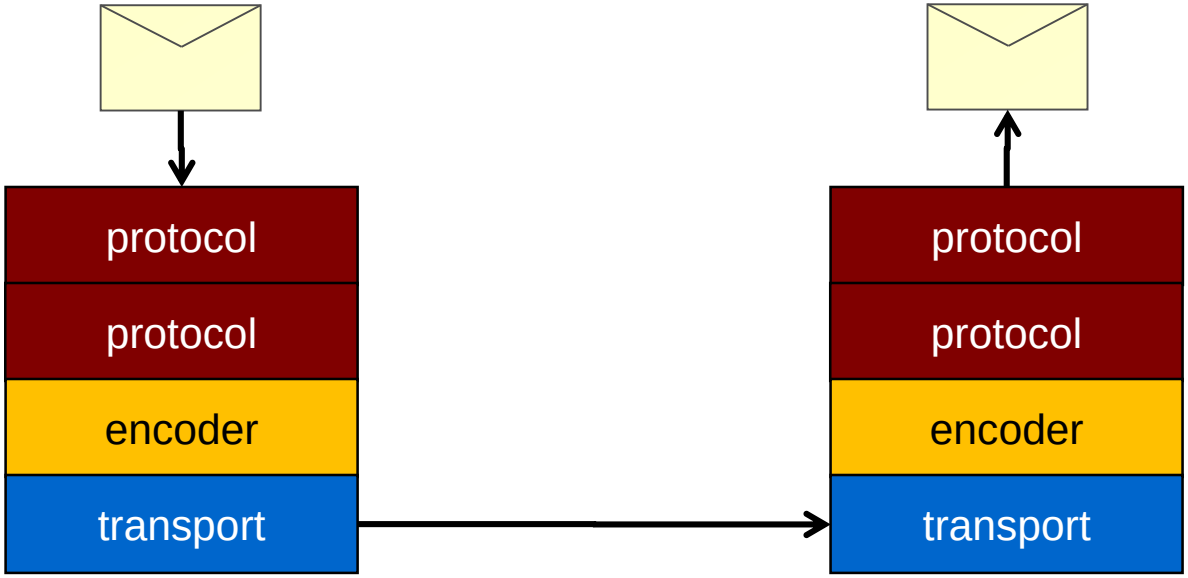
```
void SocketListen()
{
    Socket listener = GetSocket();
    listener.Listen(100);

    while (true)
    {
        Socket connection = listener.Accept();
        byte[] reply = HandleStream(connection);

        connection.Send(reply);
        connection.Shutdown(SocketShutdown.Both);
        connection.Close();
    }
}
```

Channel: the socket of WCF

```
void ChannelListen()
{
  IChannelListener<IReplyChannel> listener = GetListener();
  listener.Open();

  while (listener.WaitForChannel())
  {
    IReplyChannel channel = listener.AcceptChannel();
    channel.Open();

    RequestContext request = channel.ReceiveRequest();
    Message reply = HandleMessage(request.RequestMessage);

    request.Reply(reply);
    request.Close(); channel.Close();
  }
}
```

Channel layer

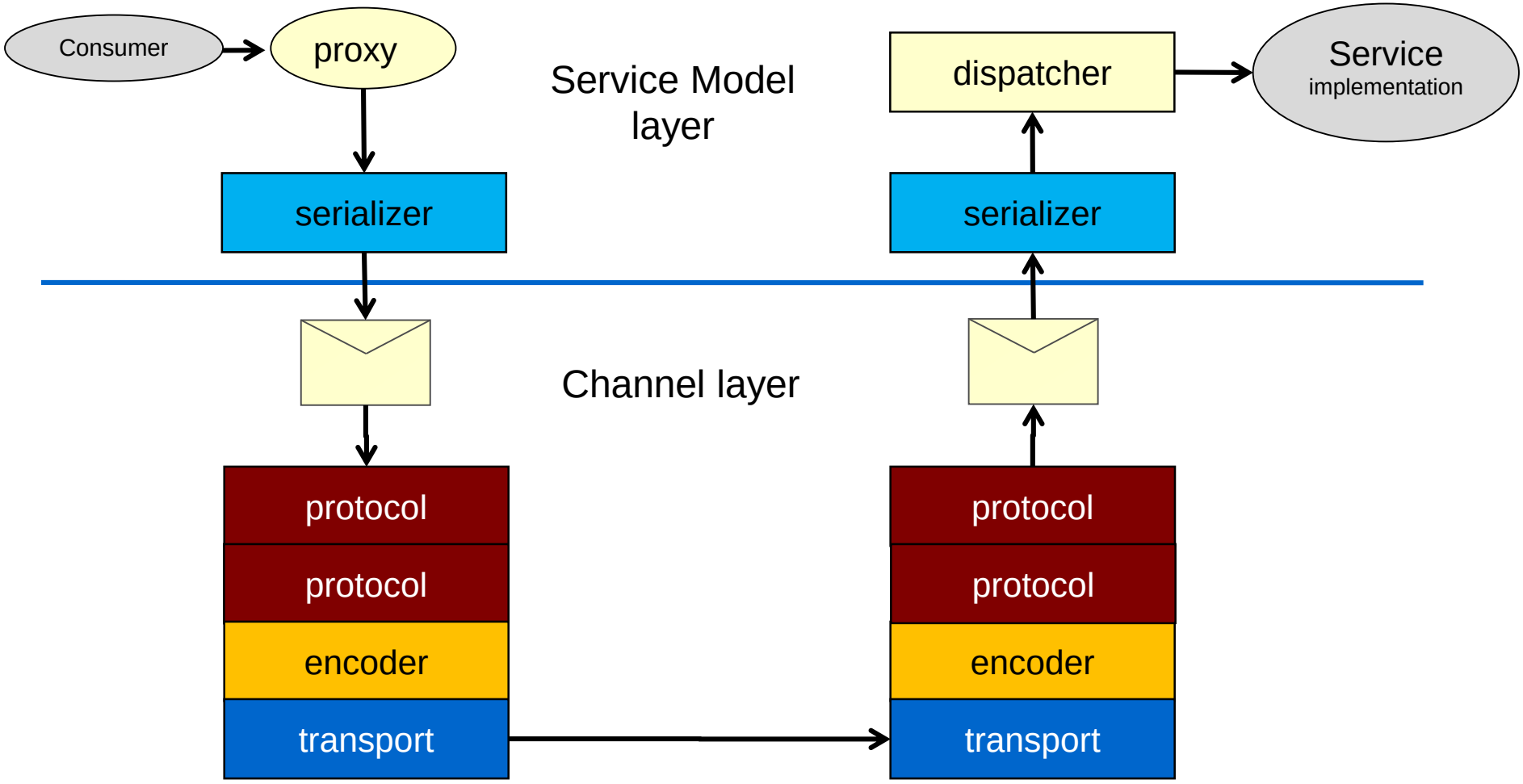
- **Messages, channels, and encoders form the channel layer**



Service Model layer

- **Channel layer is generally too low level for most situations**
  - manual handling of listening, dispatching, threading etc.
  - raw message handling
- **.NET-typed proxies and services seem more productive**
  - service consumer uses familiar method-based programming model
  - messages automatically mapped to object method via message action
- **.NET-typed layer sits on top of channel layer**
  - uses object serialization to generate message XML
  - known as service model layer

Service Model layer & channel layer



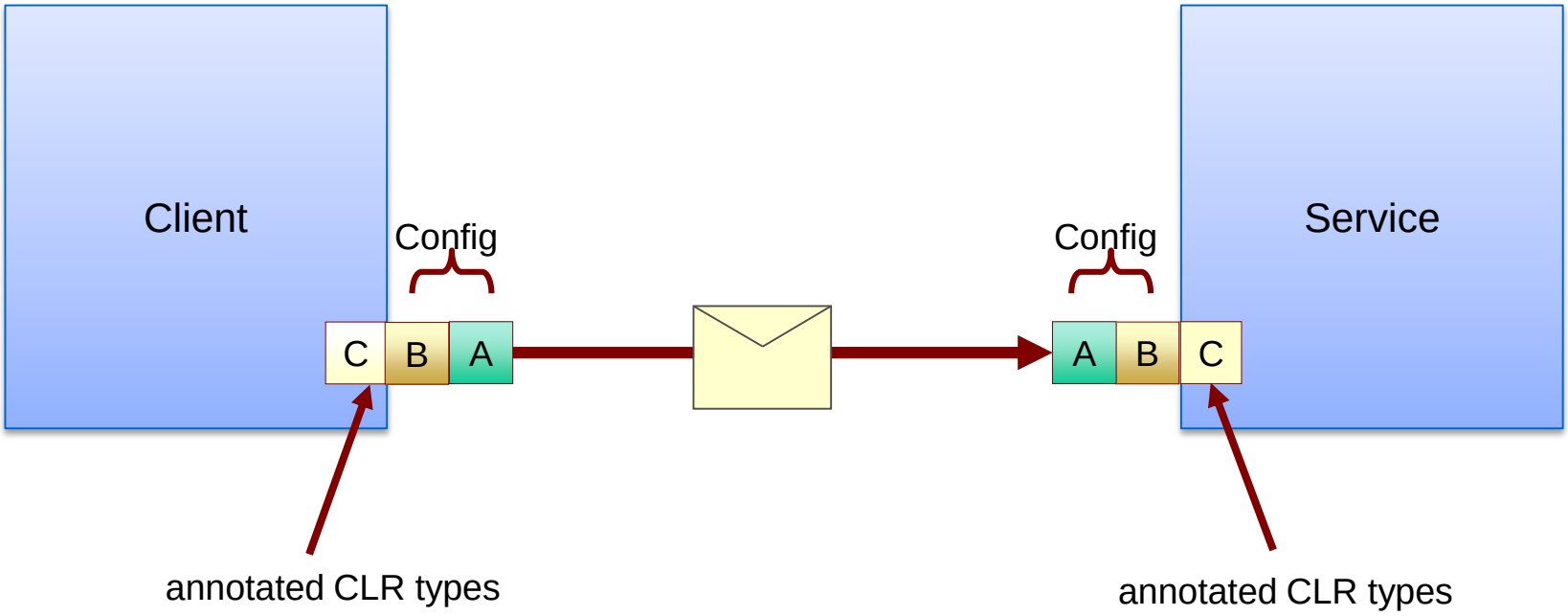
WCF service building blocks

- **Endpoints**
  - communication details and functionality
- **Behaviors**
  - local functionality

Endpoints

- **Application communicates through endpoints**
- **Endpoint is defined by the 'WCF ABC'**
  - Address
    - where is the service?
  - Binding
    - how does message exchange/communication take place?
  - Contract
    - what operations/messages/data are available?

Programming model ABC



Contracts

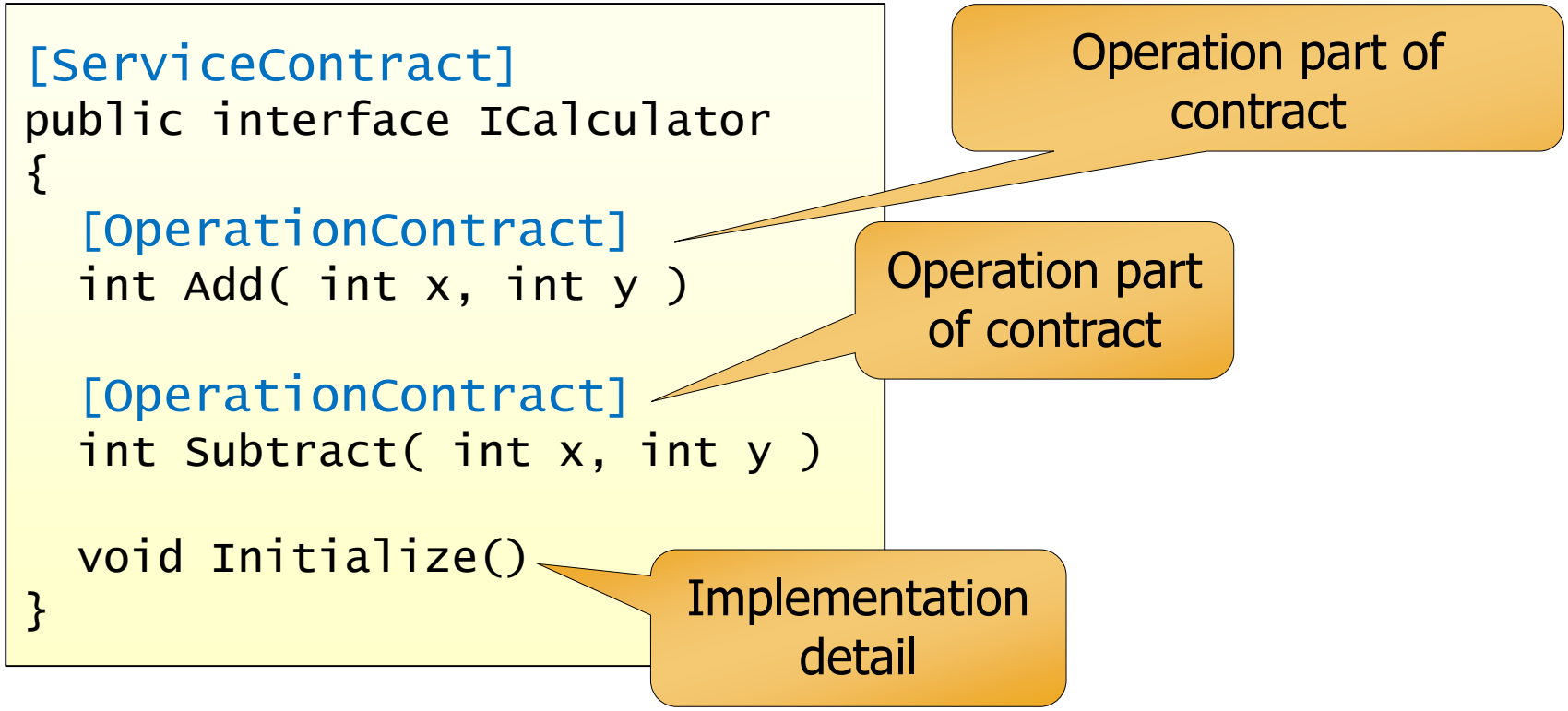
- **Annotated CLR types define contract**
  - type annotated with `[ServiceContract]` attribute

```
[ServiceContract]
interface ICalculator
{
  …
}
```

Operations defined here

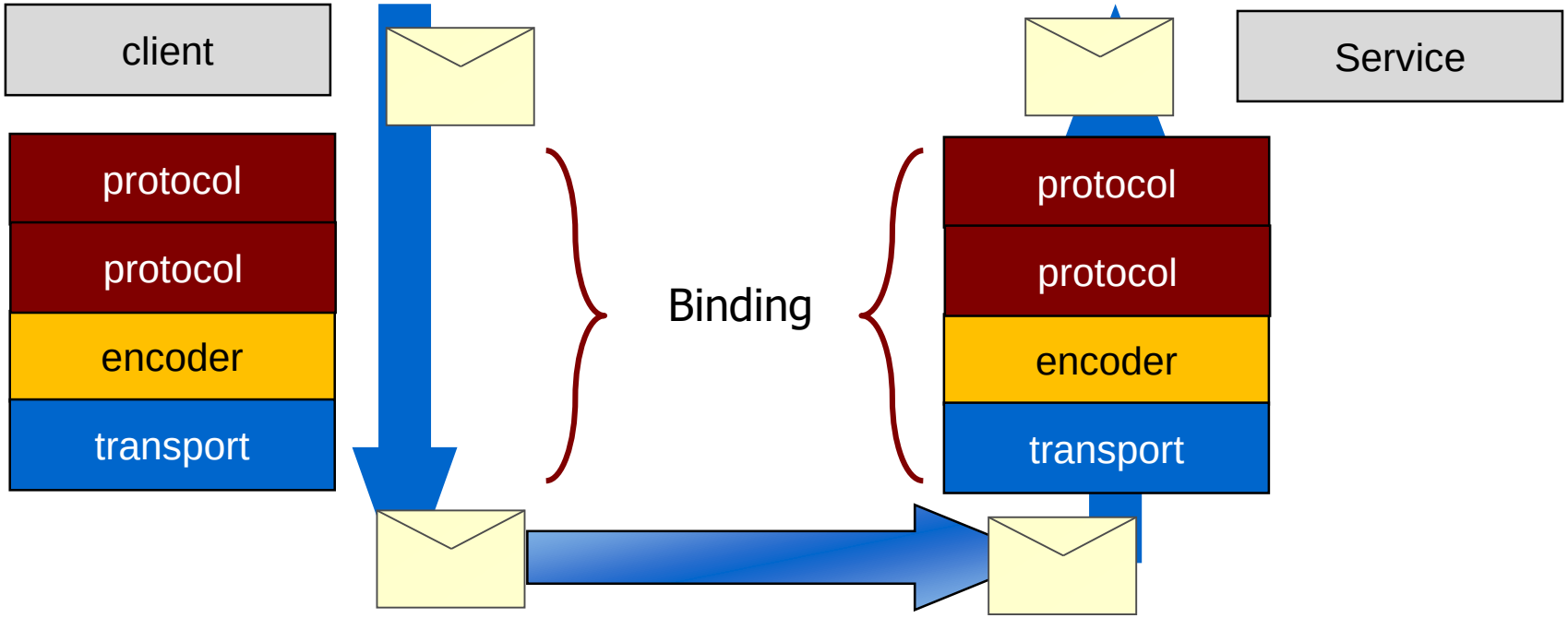
Contracts (cont.)

- **Methods are exposed explicitly as operations**
  - opt-in via `[OperationContract]` attribute



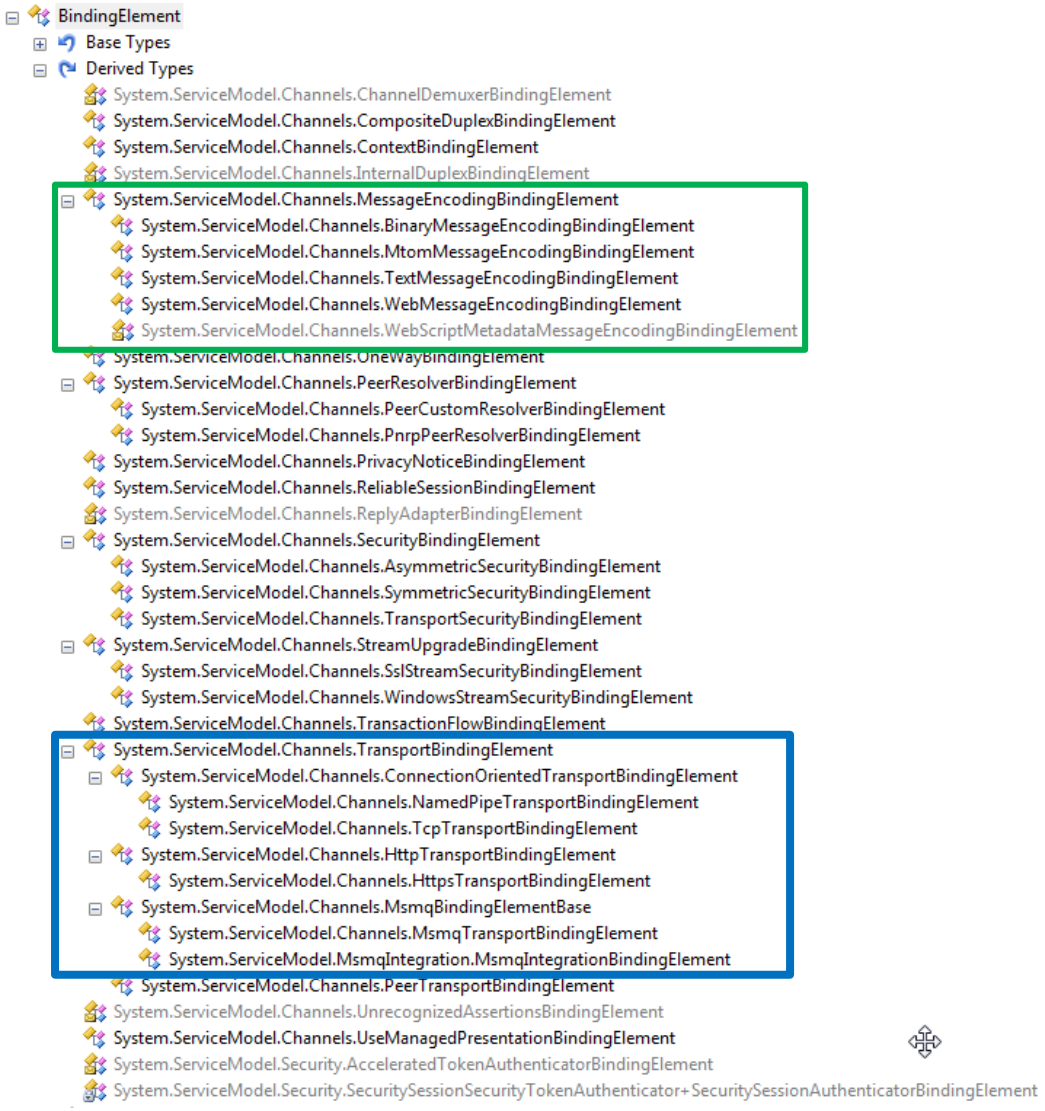
Bindings

- **Bindings specify how messages are transmitted**
- **Defines channel stack**
  - pre-defined standard bindings
  - custom and user-defined bindings



Binding elements

- **Bindings consist of primitive binding elements in a stacked fashion**
- **Three big families of binding elements**
  - protocols (application-level)
  - encoders
  - transports



Binding elements in bindings

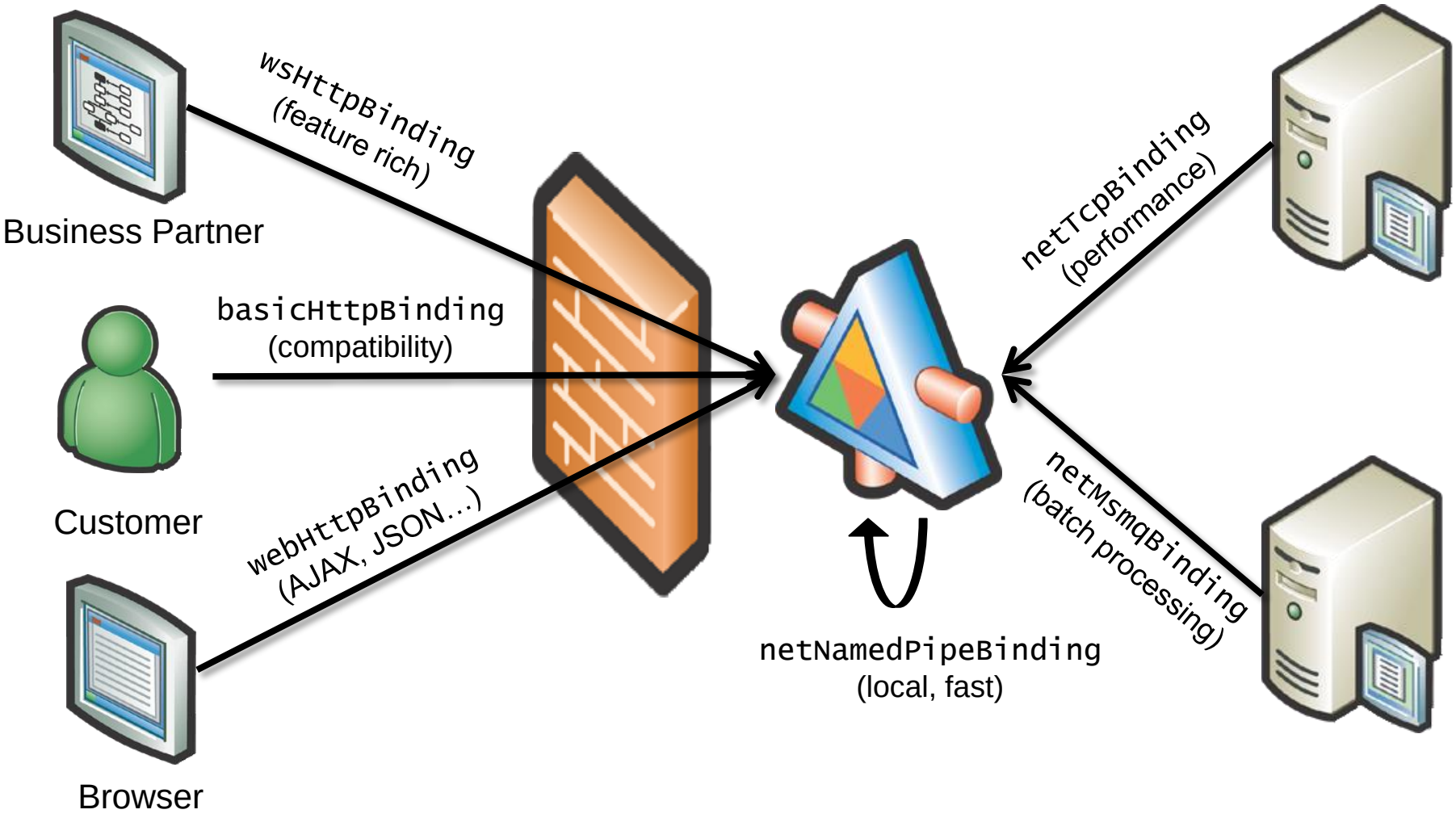
- `wsHttpBinding` **example**
  - `CreateBindingElements` of `Binding` class return stack of binding elements

```
public override BindingElementCollection CreateBindingElements()
{
    BindingElementCollection elements = new BindingElementCollection();
    elements.Add(this.txFlow);
    if (this.reliableSession.Enabled)
    {
        elements.Add(this.session);
    }
    SecurityBindingElement sec = this.CreateMessageSecurity();
    if (sec != null)
    {
        elements.Add(sec);
    }
    WSMessageEncodingHelper.SyncUpEncodingBindingElementProperties(
           this.textEncoding, this.mtomEncoding);
    if (this.MessageEncoding == WSMessageEncoding.Text)
    {
        elements.Add(this.textEncoding);
    }
    else if (this.MessageEncoding == WSMessageEncoding.Mtom)
    {
        elements.Add(this.mtomEncoding);
    }
    elements.Add(this.GetTransport());
    return elements.Clone();
}
```

Choosing a binding

- **Interoperable**
  - `basicHttpBinding` (WS-I Basic Profile)
  - `wsHttpBinding`, `wsFederationHttpBinding` (WS-*)
  - `webHttpBinding` (REST)
- **WCF specific**
  - `netTcpBinding`
  - `netPeerTcpBinding`
  - `netNamedPipeBinding`
  - `netMsmqBinding`
- **MSMQ interop**
  - `msmqIntegrationBinding`
- **Write your own for custom features**

Bindings in distributed scenarios



Default bindings

- **Default binding configuration valid for all endpoints using the binding type**
  - default binding configuration has no or empty `name` attribute value

```
<system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding maxReceivedMessageSize="9999999">
          <readerQuotas maxArrayLength="9999999"/>
        </binding>
      </basicHttpBinding>
    </bindings>
  </system.serviceModel>
```

Hosting services

- **WCF supports two hosting models**
  - self hosting
    - use own process for hosting the service, e.g. Windows Service
  - WAS hosting
    - Windows Process Activation Service
    - supported by IIS7 and higher
    - emulated for IIS6 for HTTP/S (web hosting)

Self hosting

- Create an instance of `ServiceHost`
- Add endpoints
- Call `Open()` to make service available

```
ServiceHost host = new ServiceHost(typeof(Service));
host.AddServiceEndpoint(typeof(IService),
  new netTcpBinding(),
  "net.tcp://localhost/Service");

host.Open();

Console.WriteLine("Service ready ...");
Console.ReadLine();

host.Close();
```

Default endpoints & protocol mappings

- **If no endpoints specified then default endpoints will be added by WCF**
  - based on the `ServiceHost`'s base addresses
- **Call `host.AddDefaultEndpoints()` to add set of default endpoints manually**
- **Can redefine default protocol mappings for bindings per scheme for global use**

```
<protocolMapping>
  <add scheme="http" binding="basicHttpBinding"/>
  <add scheme="net.tcp" binding="netTcpBinding"/>
  <add scheme="net.pipe" binding="netNamedPipeBinding"/>
  <add scheme="net.msmq" binding="netMsmqBinding"/>
</protocolMapping>
```

Specifying endpoint in config

- **Endpoints can be also specified in configuration files**
- **Part of the .NET config file**
  - Visual Studio provides IntelliSense
  - *SvcConfigEditor.exe* available as config GUI

```
<services>
  <service name="Calc">
    <endpoint address="net.tcp://localhost/Service"
              binding="netTcpBinding"
              contract="IService" />
  </service>
</services>
```

Consuming services

- **Create client-side proxy using a channel factory**

```
NetTcpBinding bin = new NetTcpBinding();
EndpointAddress a =
  new EndpointAddress("net.tcp://localhost/Service");

ChannelFactory<IService> factory =
  new ChannelFactory<IService>(bin, a);

IService proxy = factory.CreateChannel();

// use proxy

((IClientChannel)proxy).Close();
```

Behaviors

- **WCF is highly customizable**
- **Behaviors affect the internal conduct of WCF**
  - not what goes over the wire
- **WCF built-in behaviors respond to most needs**
- **Fully extensible – you can write you own behavior**
  - main point of interception if you don't need to modify the wire format

Specifying behaviors

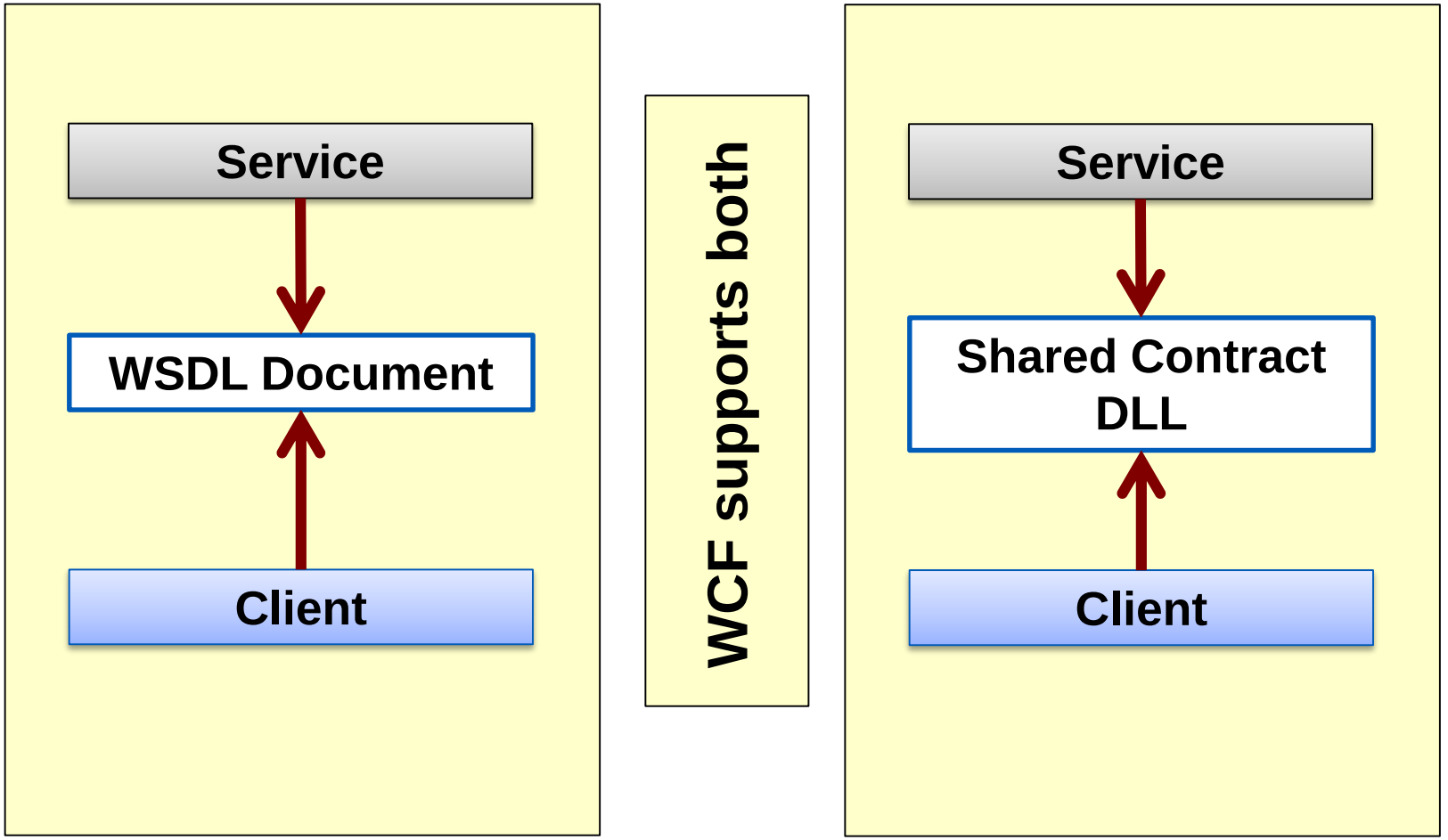
- **Use custom attributes for stable behavior**
  - `[ServiceBehavior]` and `[OperationBehavior]`
  - e.g. transaction flow
- **Use configuration file for less stable behavior**
  - e.g. throttling
- **Use code for behaviors that change at runtime**
  - e.g. client credentials
- **Behavior is applied in the above order**

Default behaviors

- **Default behavior configuration valid for all services and/or endpoints**
  - service behaviors and endpoint behaviors
  - default behavior configuration has no or empty `name` attribute value

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <serviceMetadata httpGetEnabled="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Metadata vs. shared contracts



Metadata

- **Metadata used to interrogate service**
  - what contracts?
  - what message formats?
  - what protocols are supported and required?
- **Metadata not exposed automatically**
  - must be turned on by adding a `serviceMetadata` behavior
  - requires HTTP based base-address

```
<serviceBehaviors>
  <behavior name="myServiceBehaviors" >
    <serviceMetadata httpGetEnabled="true" />
  </behavior>
</serviceBehaviors>
```

Creating a client from metadata

- **Command line: svcutil.exe generates proxies**
  - creates contract-based proxy code
  - creates binding config file
- **“Add Service Reference” in Visual Studio**
- **Requires service to expose metadata**
  - WSDL via HTTP/S
  - MEX

```
C:\>svcutil http://localhost/Basics/service
          /config:app.config /out:generatedProxy.cs
```

Summary

- **WCF unifies communication programming model**
  - based on XML messaging
  - implements WS-*/W3C standards for advanced features & interoperability
  - can be extended for any protocol / format
- **WCF architected as channel layer and service model layer**
- **Endpoint = Address + Binding + Contract**
- **Internals fully customizable using behaviors and channels**