

# *IIB Design for Performance and Tuning*

Christopher Frank

IBM Hybrid Cloud - Integration

# Agenda

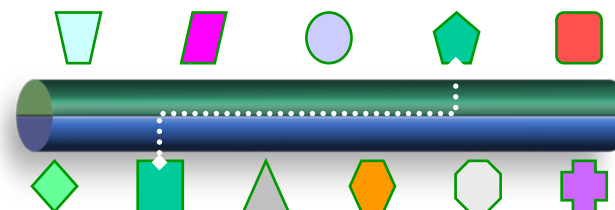
- Architecture
- What's New
- Flow Design and Optimization
- Deployment Considerations
- Tuning, Tools and Transports
- Transformation Code Optimization
- Summary



# IBM Integration Bus

## ■ Universal Connectivity FROM anywhere, TO anywhere

- ▶ Simplify application connectivity for a flexible & dynamic infrastructure



## ■ Comprehensive Protocols, Transports, Data Formats & Processing

- ▶ Connect to applications, services, systems and devices
  - MQ, JMS 1.1, HTTP(S), SOAP, REST, File (incl. FTP, FTE, ConnectDirect), Database, TCP/IP, MQTT, CICS, IMS, SAP, SEBL, .NET, PeopleSoft, JDEdwards, SCA, CORBA, email...
- ▶ Understand the broadest range of data formats
  - Binary (C/COBOL), XML, CSV, DFDL, JSON, Industry (SWIFT, EDI, HL7...), IDOCs, user-defined
- ▶ Built-in suite of request processors
  - Route, Filter, Transform, Enrich, Monitor, Publish, Decompose, Sequence, Correlate, Detect...

# Tuning Options Can be Confusing!

- Lots of dials and switches!
- Can seem overwhelming
  - ▶ But doesn't need to be
- We'll try sorting some of these out
- Performance evaluation and improvement is an iterative process
- Important to have the philosophy, process, infrastructure and tools to enable this



# Performance Objectives

- **Objectives are very important to focus the mind**
  - ▶ Important to set the right objectives at the very start
    - Capable of processing 5000 messages per second
    - Running within 1GB of memory
    - Response time of less than 100 ms
  
  - Being fast enough
  - Not using too much memory
  - Fast
  
- **When setting objectives be realistic**
  - ▶ Understand what is physically possible
  - ▶ Ensure the objectives meet the needs of all interested parties
  - ▶ Ensure all interested parties agree to the objectives
  - ▶ Establish metrics by which you can measure the objective



## Design, Code, Run, Measure...Change, Run, Measure

- **Delivering the right level of performance the first time out is unlikely**
  - ▶ Well done though if you do it!
- **Performance evaluation and improvement is an iterative process:**
  - ▶ Design
  - ▶ Code
  - ▶ Run
  - ▶ Measure
  - ▶ Identify hotspots
- **Important to have the philosophy, process, infrastructure and tools to enable this**

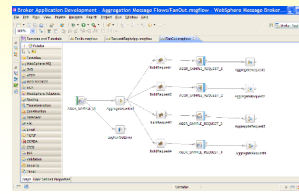


# IBM Integration Bus Architecture

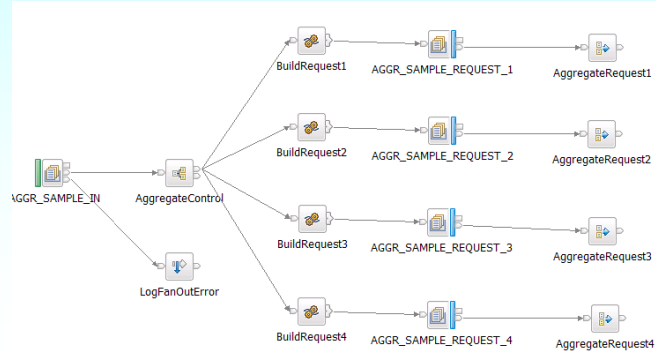


# IBM Integration Bus Architecture

Development  
& Operations



Runtime

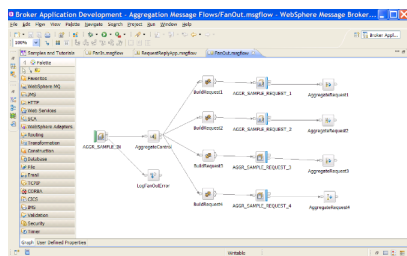


MQ  
HTTP  
SOAP  
JMS  
Database  
EIS...

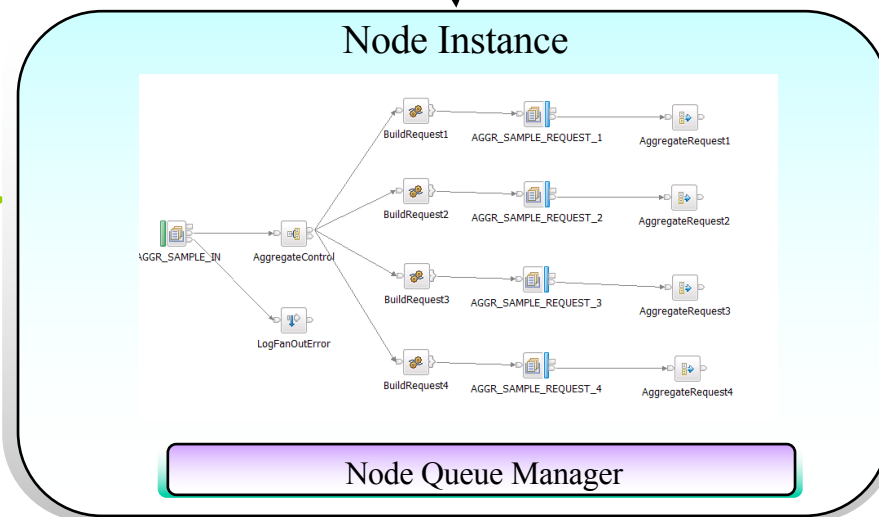
MQ  
HTTP  
SOAP  
JMS  
Database  
EIS...



# IBM Integration Bus Architecture

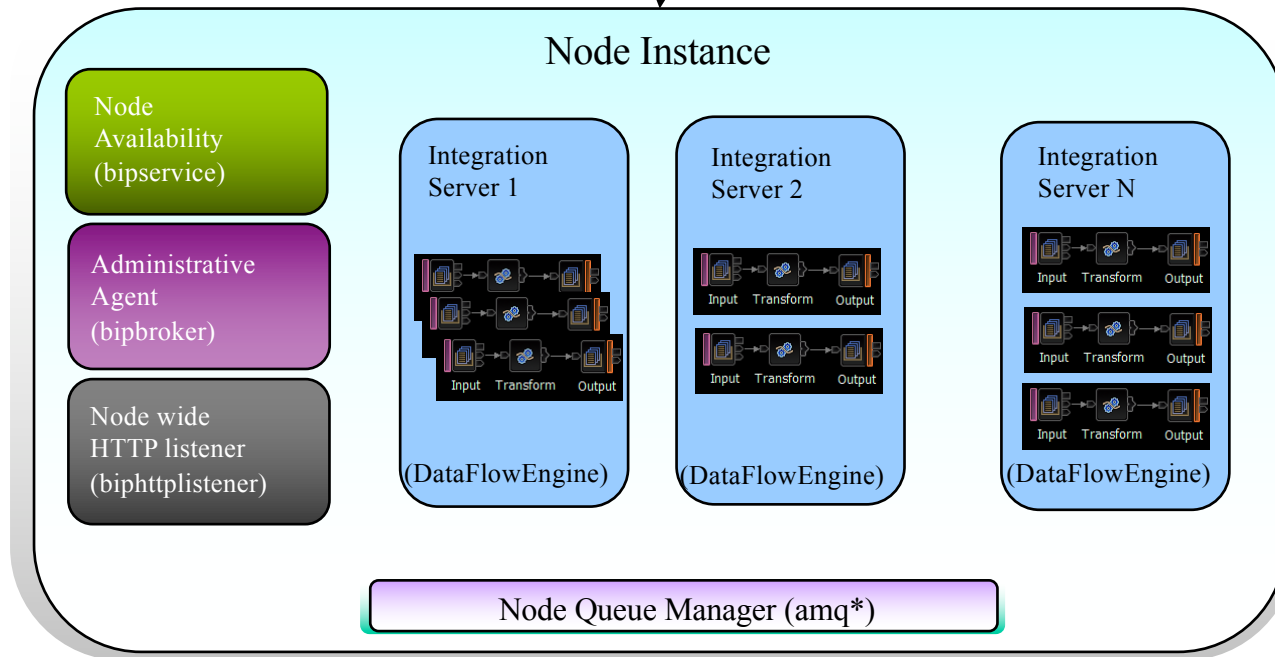
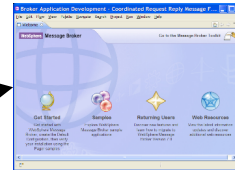


MQ  
HTTP  
SOAP  
JMS  
Database  
EIS...



MQ  
HTTP  
SOAP  
JMS  
Database  
EIS...

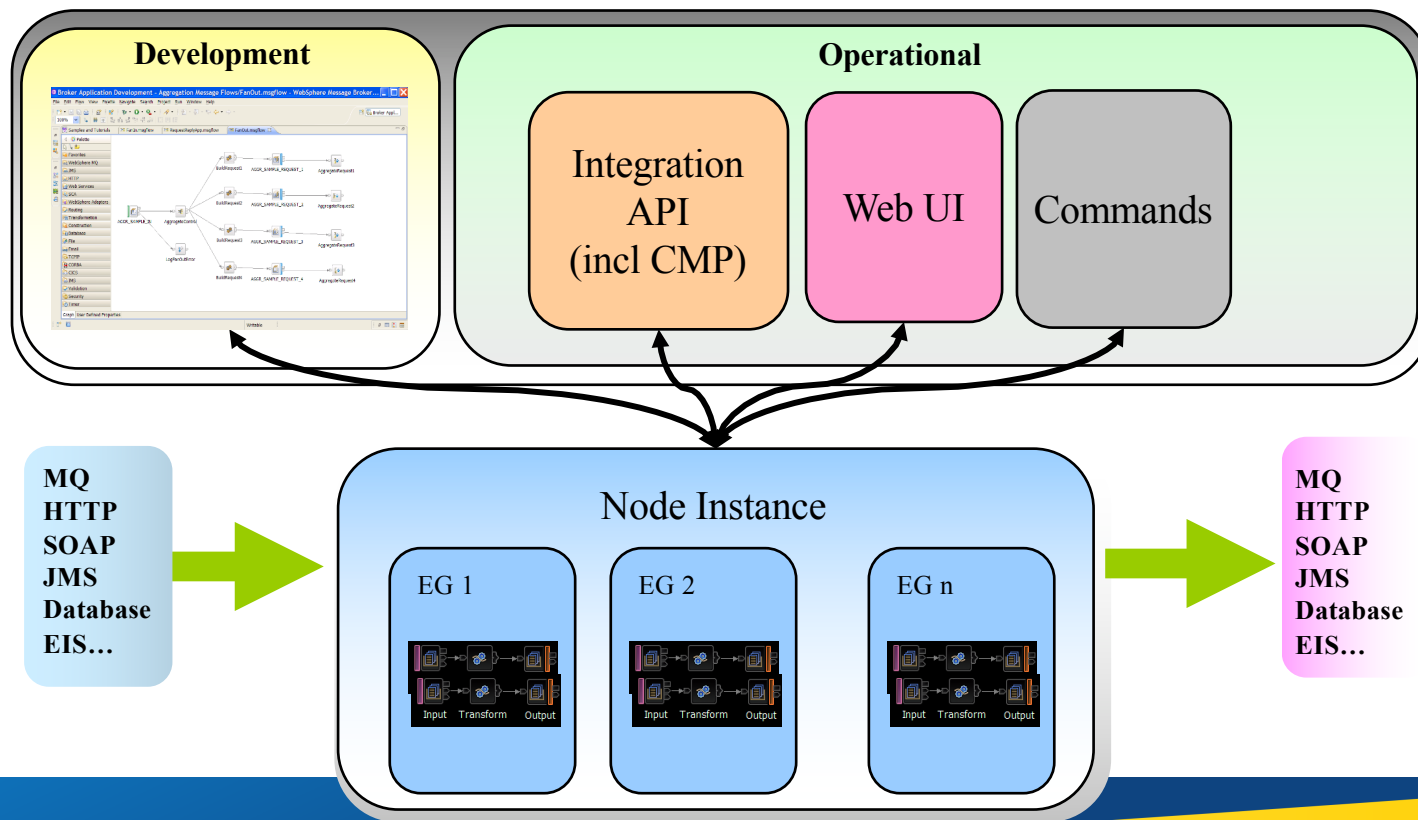
# Under the Covers



# Integration Servers

- **An Integration Server (DataFlowEngine) is a runtime container for message flow and related artifacts**
  - ▶ Previously called “Execution Groups”
  - ▶ Can support one or many message flows per execution group
  - ▶ Number of DataFlowEngines per Node Instance varies depending on IIB Edition
- **Consists of:**
  - ▶ Message flows
  - ▶ Message sets/schemas/maps
  - ▶ Embedded JVM
  - ▶ Configurable services
  - ▶ Network connections (HTTP/TCP)
  - ▶ Node management infrastructure threads
- **Integration Servers provide a good level of separation**
  - ▶ Process level verses Thread level

# A Node Instance – Development and Operation



#MaximizeYourGame

## What's New (or not so new...)

# Performance Improvements over Time

- **The last few releases of WMB/IIB have seen significant improvements in performance**
  - ▶ Message parsing and serialisation
  - ▶ Graphical Mapping
  - ▶ ESQL performance
  - ▶ Runtime code optimization
  - ▶ Streamlined processing, removal of superfluous function
- **Some of these you get just by upgrading**
  - ▶ ESQL performance
  - ▶ Runtime code optimization
- **Others through new, more efficient approaches**
  - ▶ Modern parsers and mapping technologies
- **Others provide new tools, approaches that can be used**
  - ▶ Better Resources Statistics, Activity Logging
- **We'll quickly review what these were for the last few product releases**

# Performance improvements in WMB V8 over V7

- **Significant improvements introduced in WMB V8**
- **Message parsing and serialisation**
  - ▶ DFDL - Industry standard for binary, text and industry data formats
  - ▶ Parser has excellent performance compared to MRM parser
  - ▶ Parse and serialisation improvements have been measured **up to 70%** when compared to MRM
    - Typical improvement of **~50%**.
- **Graphical Mapping**
  - ▶ Graphical Data Mapper (GDM) introduced in V8 provides better visual transformation experience
  - ▶ Also has **excellent performance** characteristics
    - Make it a viable option for performance sensitive transformations.
  - ▶ Tests have been measured performing **close to optimised programmatic transformations in ESQL, Java and .Net**, with the typical measurement being 50%.
- **Performance Analysis of message flows using Resources Statistics and Activity Log**
  - ▶ Not a performance enhancement per se
  - ▶ Allows deeper analysis, understanding of performance related issues, quicker resolution/improvement

## Performance improvements in WMB V8 over V7

N

- Significant performance improvements introduced in WebSphere Message Broker V8.
- DFDL was introduced as a new message parsing and serialisation technology. DFDL is an industry standard for binary, text and industry data formats. The implementation of this new parser was designed with performance in mind, and has excellent performance characteristics. DFDL parse and serialisation improvements have been measured **up to 70%** when compared to existing MRM technology, with a typical improvement of **50%**.

O

- A new Graphical Data Mapper (GDM) was also introduced. The new mapping node has **excellent performance** characteristics, and is a viable option for performance sensitive transformations. Tests have been measured performing **close to optimised programmatic transformations in ESQL, Java and .Net**, with the typical measurement being 50%.

T

- Performance Analysis of message flows using Resources Statistics and Activity Log is not a performance enhancement per se, but tools that enable deeper analysis and understanding of performance related issues, enabling more rapid identification of performance-related problems and quicker resolution and/or improvement, either through tuning of WMB itself, but also in understanding and modifying message flows to maximize throughput and minimize resource consumption.

E

S



## Performance improvements in IIB V9 over WMB V8

- **Significant improvement for message flows that use DFDL and GDM**
  - ▶ > %20 performance improvement in V9 over V8
- **Similar performance gains for flows processing messages over HTTP and TCP/IP**
- **A new Web UI statistics view enables runtime performance to be analysed and monitored**
  - ▶ More detailed information available, with less overhead to gather

## Performance improvements in IIB V9 over WMB V8

N

- Still more performance improvements were incorporated into IIB V9. Performance of V9 exceeds that of WMB V8 by **more than 20%** when using the DFDL parser for data parsing and serialisation, and the Graphical Data Mapper for message transformation.

O

- Message flows that process messages over HTTP and TCP/IP also saw similar performance gains. And in all other areas, performance is equal to or better than WMB V8.

T

- A new WebUI statistics view enables more detailed monitoring and analysis of message flow runtime performance.

E

S

# Performance improvements in IIB V10 over IIB V9

- **IIB V10 runtime performance exceeds that of V9 in a number of key areas**
  - ▶ DFDL scenarios show a **~20%** reduction in cost per message
  - ▶ GDM maps accessing a Database show a **~10%** reduction in cost per message
  - ▶ JSON parser shows an **~18%** reduction in cost per message
  - ▶ HTTP Nodes and SOAP nodes over HTTP show a **~5%** reduction in cost per message
  - ▶ SOAP over JMS using MQ show a **~10%** reduction in cost per message
  - ▶ FTE nodes show an **~18%** reduction in cost per message
  - ▶ Reduction in cost of **~50%** when declaring statics in an ESQL compute node
- **Significant improvement for transacted flows that do not use MQ**
  - ▶ Prior to V10, MQ was always committed for every transaction, whether it was used or not
  - ▶ V10 only commits MQ when enlisted in the transaction
  - ▶ On a HTTPInput -> HTTPOutput flow, we saw a 30% performance increase by removing this.
- **Administration response times have improved significantly**
  - ▶ By as much as **33%** for functions such as connect and deploy, and **23%** for start and stop.
- **Improved out-of-the-box defaults**
  - ▶ e.g. ensuring that TCPNoDelay and connection persistence options are set optimally for SOAP, HTTP and TCP/IP
- **Reduced memory requirements for some key customer scenarios**
  - ▶ By the implementation of shared libraries as opposed to static libraries in v8/v9
  - ▶ Changes to the internal data structures for flows and subflows.

## Performance improvements in IIB V10 over IIB V9

N  
O  
T  
E  
S

- The IBM Integration Bus V10 runtime performance exceeds that of V9 in a number of key areas. Scenarios where DFDL is used for parsing should see a **~20%** reduction in cost per message. When accessing a database via the Graphical Data Mapper should see a **~10%** reduction in cost per message. The JSON parser was optimized, resulting in an **~18%** reduction in cost per message. Further optimizing of the HTTP nodes (and SOAP nodes over HTTP) should see a **~5%** reduction in cost per message. SOAP over JMS using MQ show a **~10%** reduction in cost per message. The FTE nodes have been upgraded to a more current version of MQ MFT and show an **~18%** reduction in cost per message. And significantly, implementation changes in ESQL result in a reduction in cost of **~50%** when declaring statics in an ESQL compute node.
- Changes to transaction management in V10 yielded an unexpected gain - prior to V10, MQ was always committed for every transaction, whether MQ nodes were used or not. With V10, MQ is only called at commit time when enlisted in the transaction - on a HTTPInput -> HTTPOutput flow, we saw a 30% performance increase by removing this.
- The changes to the admin interface improved response times for administration requests by as much as **33%** for functions such as connect and deploy, and **23%** for start and stop.
- Out-of-the-box defaults have been improved for SOAP, HTTP and TCP/IP ensuring that TCPNoDelay and connection persistence options are set optimally.
- Reduced memory in the Integration Server (nee Execution Group) runtime for some key customer scenarios, by the implementation of shared libraries (as opposed to libraries in v8/v9 which were static), ESQL static improvements, and changes to the internal data structures for flows and subflows.

#MaximizeYourGame

# Flow Design and Optimization



# Message Flow Design Considerations

- ▶ A look at the key considerations for message flow design

## Consider the end-to-end flow of data



- **Interaction between multiple applications can be expensive!**
  - ▶ Store and Forward
  - ▶ Serialise ... De-serialise
- **Some questions to ponder:**
  - ▶ Where is data being generated? Where is data being consumed?
  - ▶ What applications are interacting in the system?
  - ▶ What are the interactions between each of these applications?
- **And as data passes through IIB:**
  - ▶ Where are messages arriving? Where are they leaving?
  - ▶ Are multiple message flows being invoked?
    - Request/Reply?
    - Flow-to-flow?

# Application Styles

## ■ Request & Reply

- ▶ Uni-directional (Request only)
- ▶ Bi-directional (Request & Reply)
- ▶ Aggregation (Fan-out & Fan-in)

## ■ Data Replication

- ▶ To database
- ▶ From database
- ▶ EIS data replication

## ■ File Processing

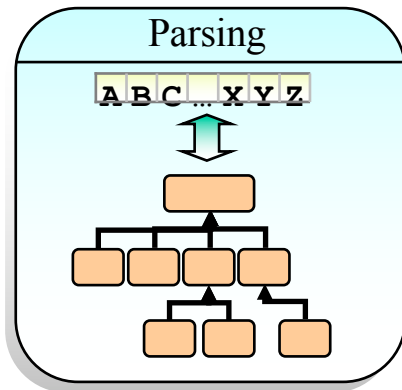
- ▶ Batch modernisation – batch meets web services

## ■ Web Services

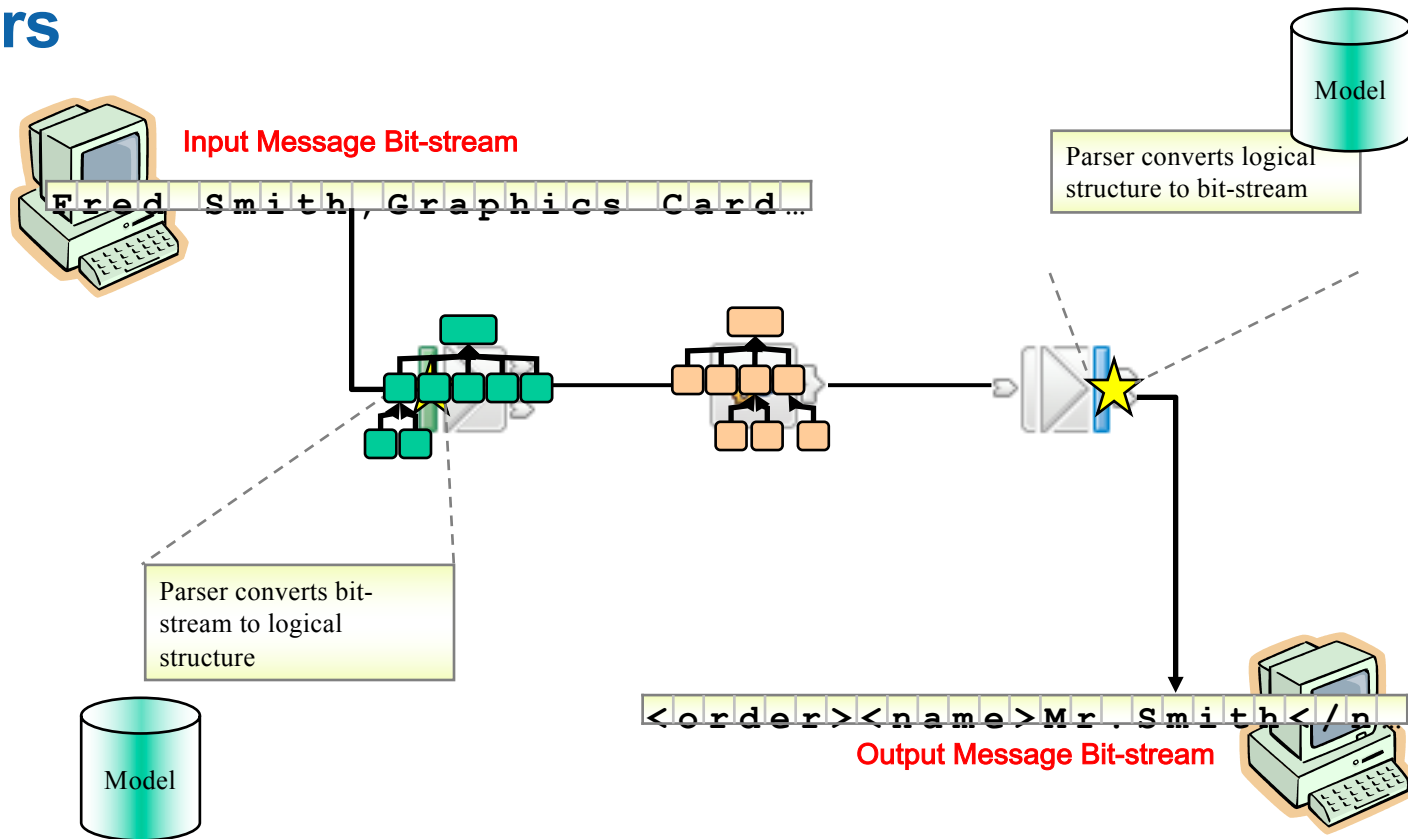
- ▶ Service Proxy
- ▶ Service end-point
- ▶ Service façade



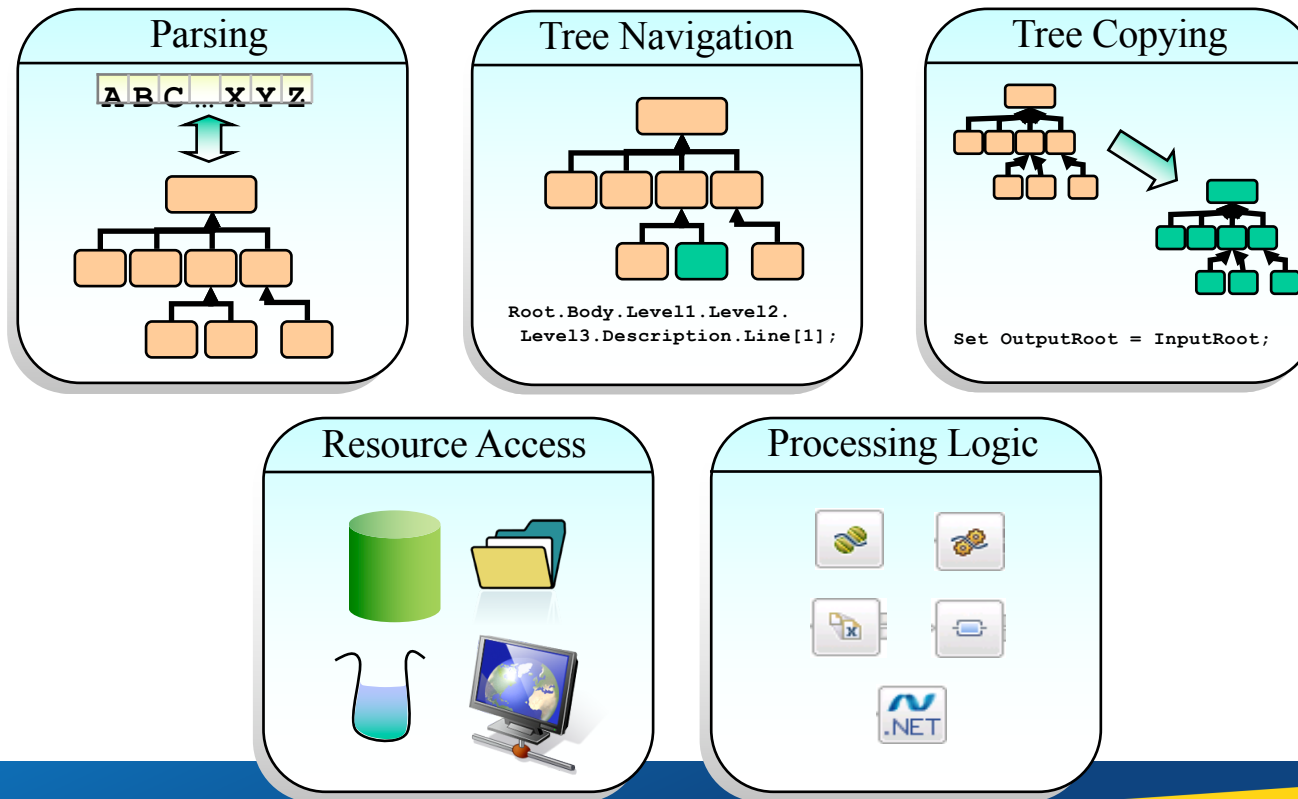
## What are the main performance costs in message flows?



# Parsers



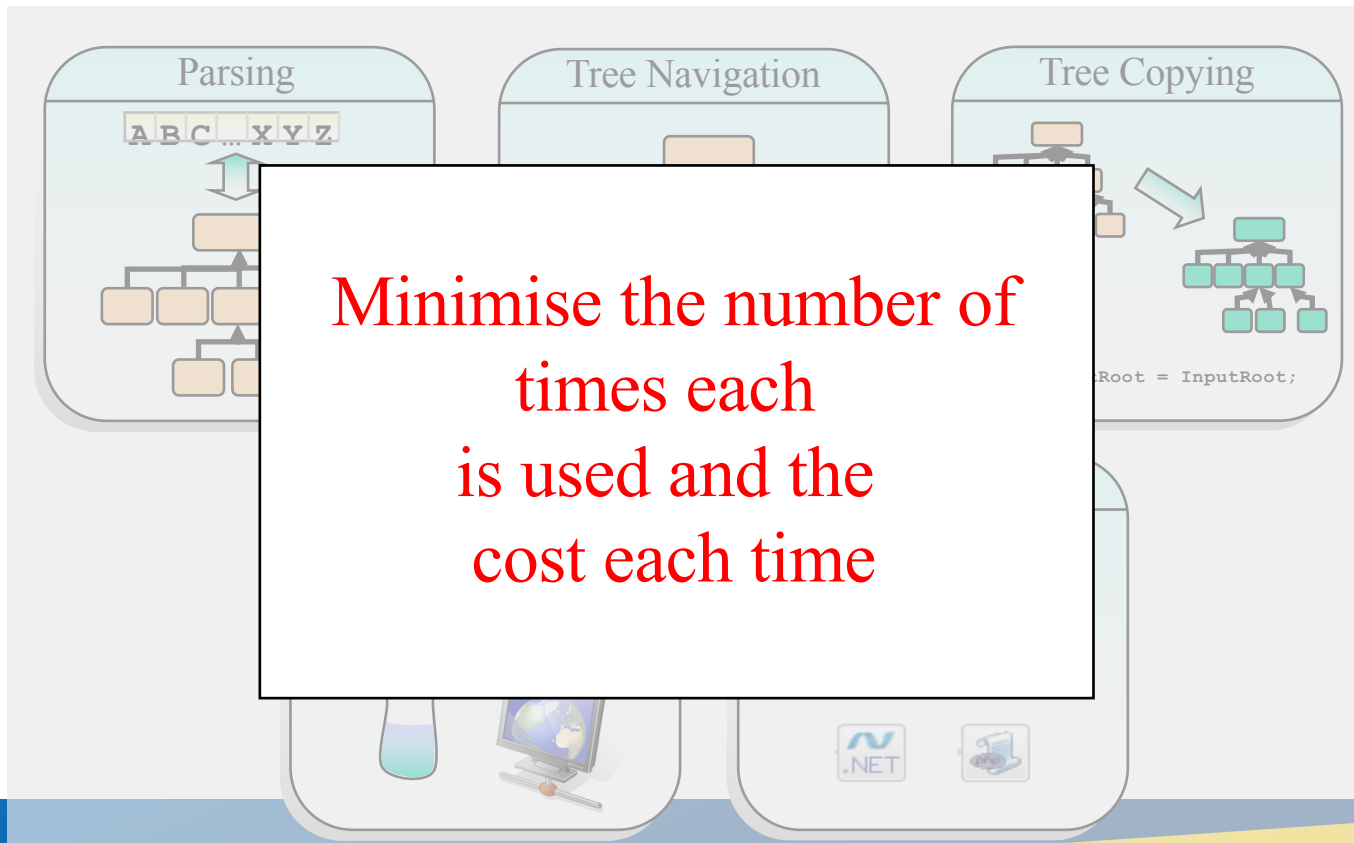
## What are the main performance costs in message flows?



## Same Techniques and Technologies yet..

- **Possible to observe substantially different:**
  - ▶ *Processing rates* – tens/thousands per second or seconds per message
  - ▶ *Resource consumption* – minimal CPU vs 1 CPU/message
  - ▶ *Response times* – Hours vs milliseconds
- **Quantity & mix of techniques & technologies is important**
  - ▶ Some technologies work better together
    - primarily those using the message tree
- **Often a situation will dictate use of certain approaches & technologies**
  - ▶ Still significant potential to get it wrong

## Techniques to Help Optimise Performance



# Interaction Style and Service Composition

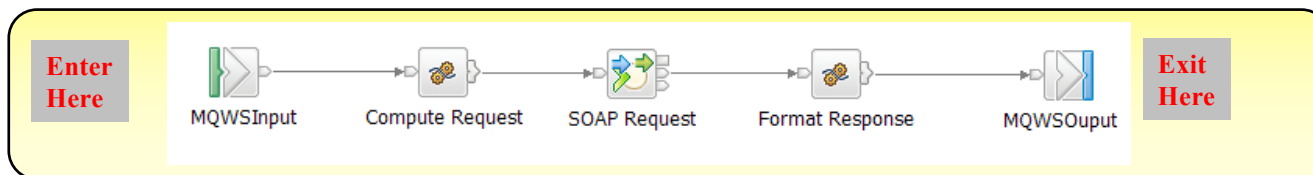
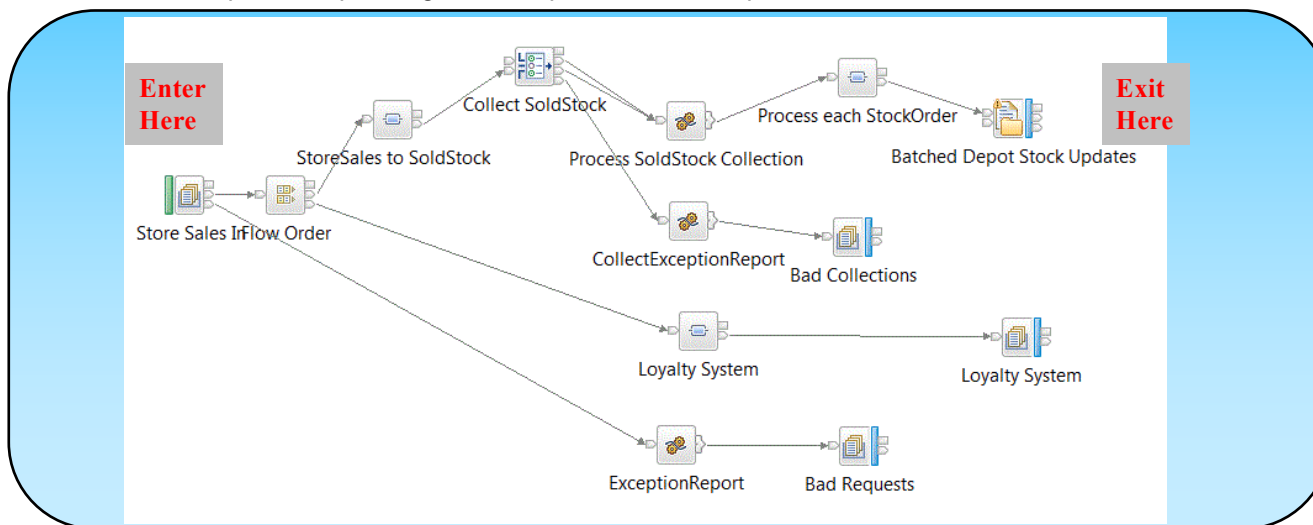
- **When enabling communications between Applications & Services ensure:**
  - ▶ Communication is at appropriate level
    - Do not want a service call to set address line 1, another for address line 2 etc.
  - ▶ Cost of communicating is the cheapest possible
    - ESQL procedure vs message flow termination and initiation of another message flow
  
- **Choices made will directly quantify the expense of processing**
  - ▶ Poor design cannot be coded and configured around

# Message Flow Structures

- Synchronous
- Asynchronous
- Single function
- Multi-function
  
- Cross cutting considerations
  - ▶ Minimise Parsing
  - ▶ Modular Approach
  - ▶ Length of message flows
  - ▶ Storing state (incl. original message) for later processing

# Synchronous

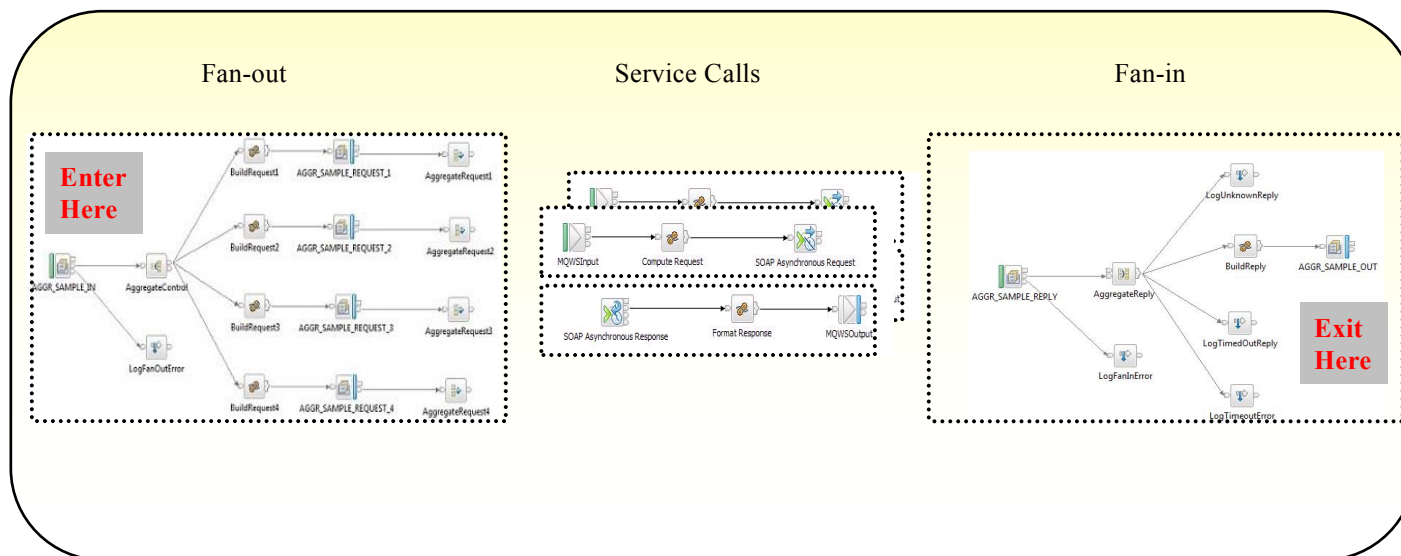
- Performs all processing within a single sequence and message flow
  - ▶ May have one or more exit points depending on the input data and sequence of events





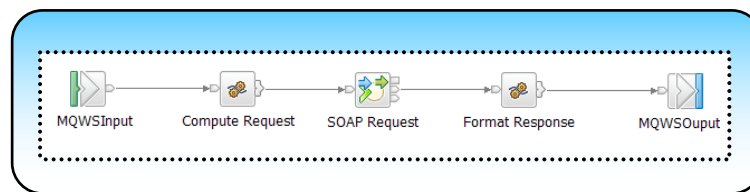
# Asynchronous

- Involves more than one message flow for processing
  - ▶ May have many exit points depending on the input data and sequence of events
  - ▶ Good where multiple services invoked in parallel or service times are long



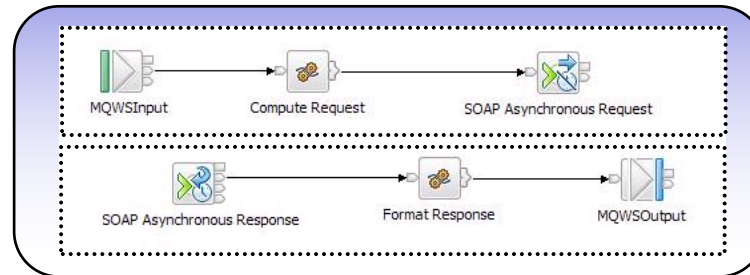
# Asynchronous

- Fast vs slow invoked services



Good only when response is quick (milliseconds)

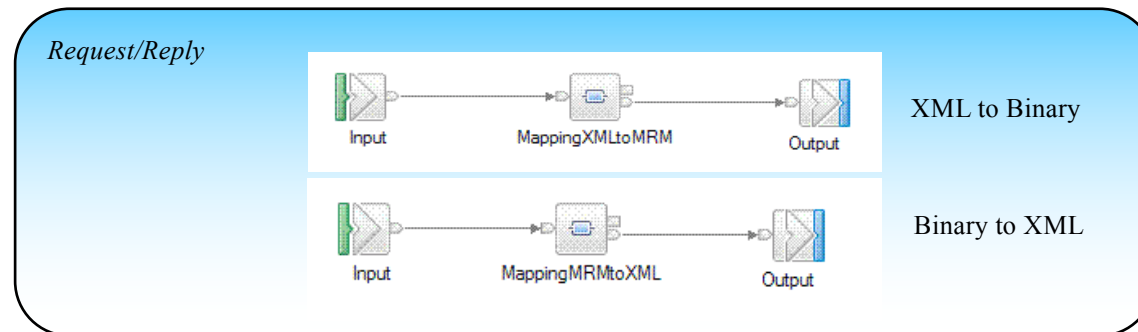
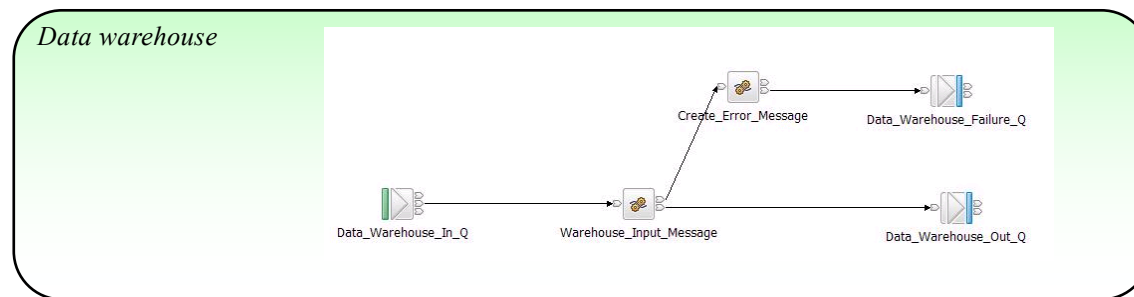
VS



Good when response is slow (seconds)

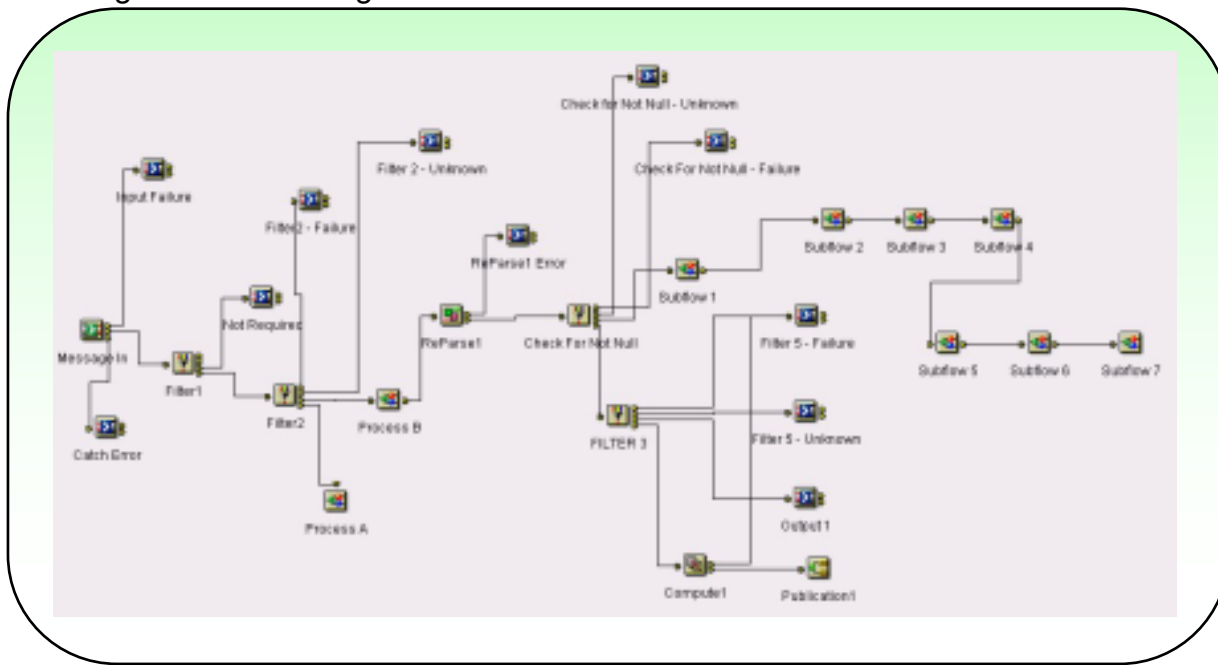
# Single Function

- **Processes a single type of message or request type**
  - ▶ Might be one message flow or a pair of message flows



# Multi-function

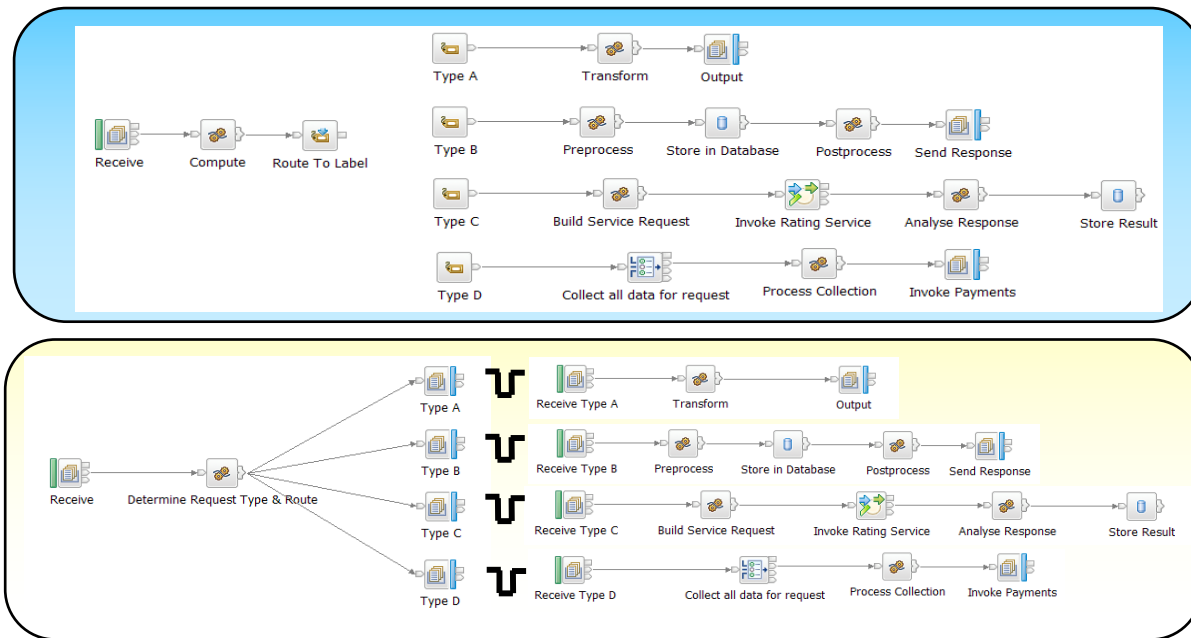
- Separation of multiple types of message of request type
  - Requires logic to first recognise the message or request type
  - Requires careful design to ensure design is efficient



# Modular Approach

## ■ Separation of processing

- ▶ Split out incoming different requests received at single entry point
- ▶ Both have single point of entry, process multiple request types but have different processing costs



# When it is Right Make a Pattern of It

- Patterns provide top-down, parameterized connectivity of common use cases

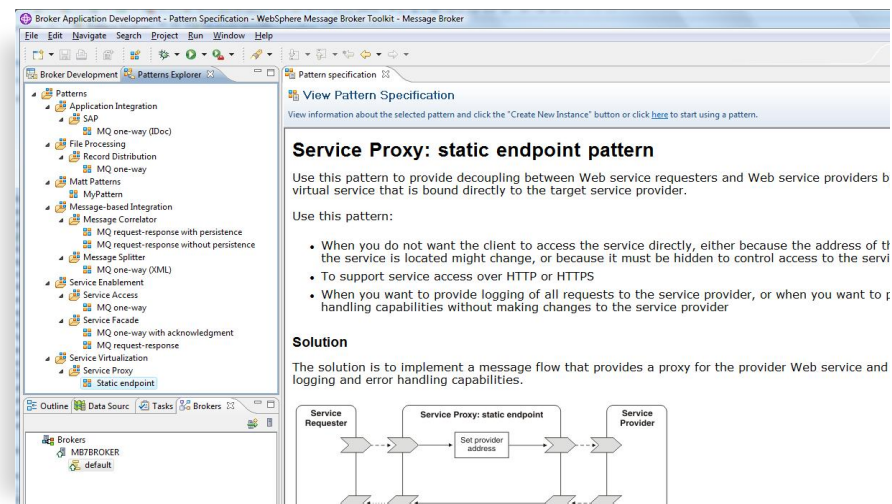
- ▶ e.g. Web Service façades, Message oriented processing, Queue to File...

- They help describe best practice for efficient message flows

- ▶ There also quick to write and less prone to errors

- Make your own patterns

- ▶ Help new message flow developers come on board more quickly
  - ▶ Spread good practice



# Design and Code Optimisations

- ▶ A look at the key considerations for message flow design

# Guidelines for Best Runtime Performance

## ■ Transformation Technologies

1. ESQL, Java, .Net
2. Mapping node
3. XSLT

## ■ Parsers

- ▶ XMLNSC for XML
- ▶ DFDL for non-XML

## ■ Short Term Data storage

1. Queue
2. Database
3. Cache

## ■ Messaging

1. Non-persistent
2. Persistent
3. Transactional

## ■ Ordering

1. Parallel
2. Sequence

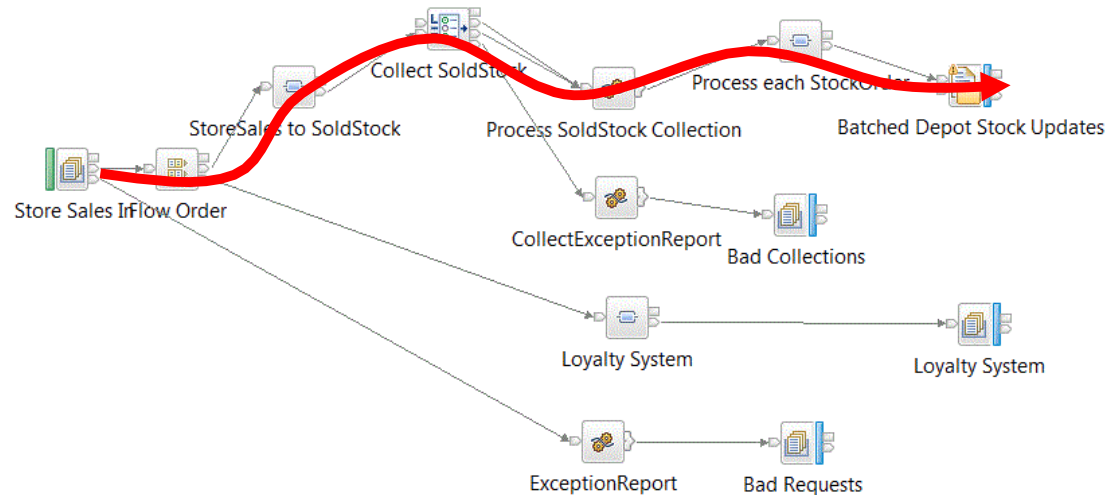


# Message Flow and Transformation Optimisations

- **Message Flow**
- **Parsers**
- **Message Tree Navigation & Copying**
- **Transformation Technologies**
  - ▶ Graphical Mapper
  - ▶ ESQL
  - ▶ Java
  - ▶ XSLT
- **Short Term Data Storage**

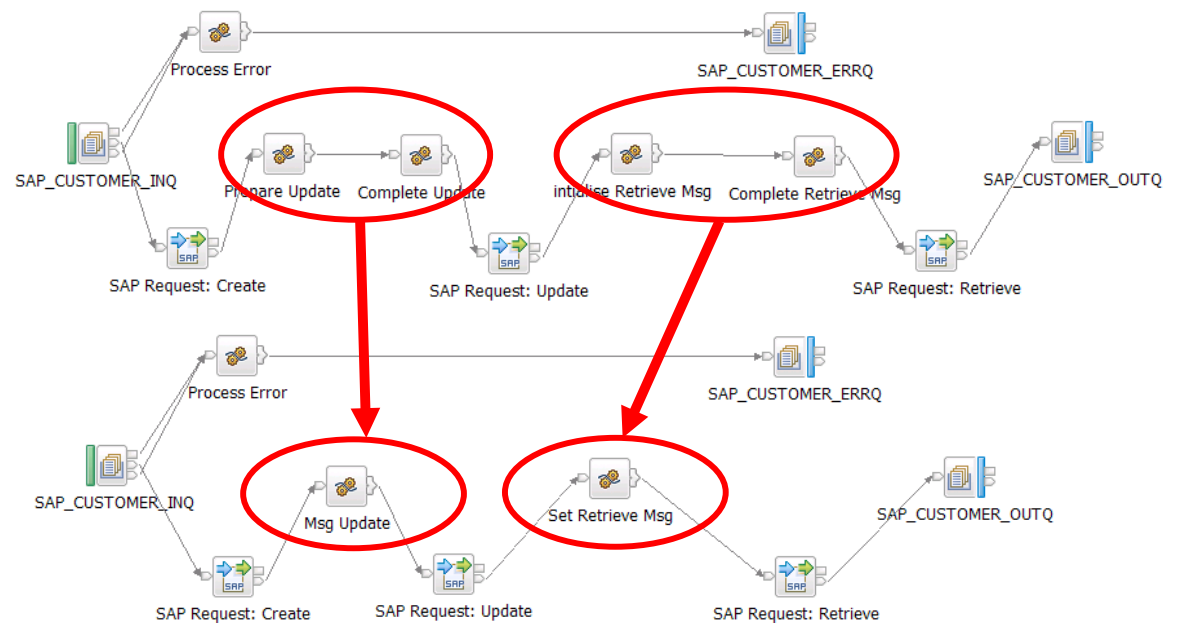
# Message Flow Coding Recommendations

- **Identify the critical processing path in a message flow**
  - ▶ Important to minimise the length and cost of this path
  - ▶ Error processing is not important



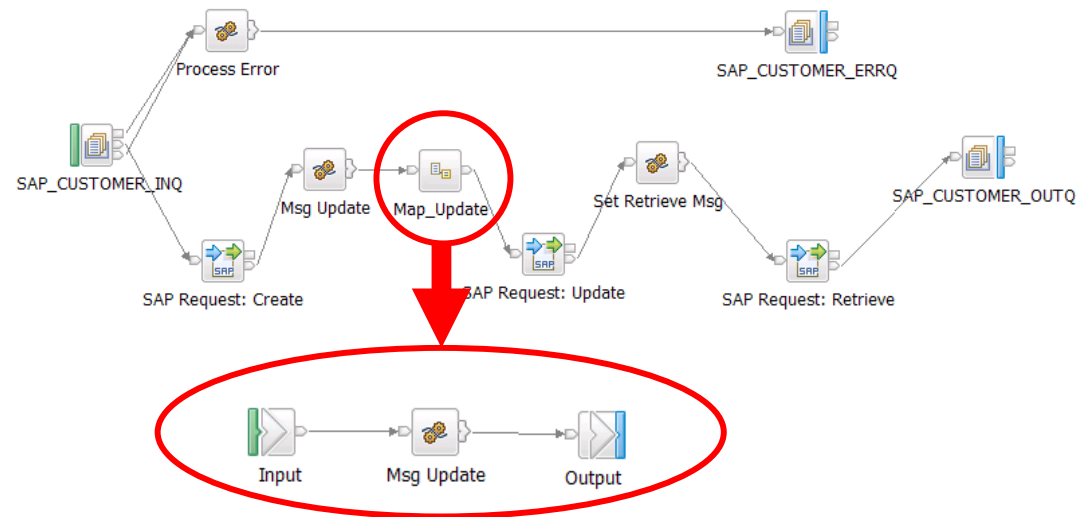
# Message Flow Coding Recommendations

- Use minimum number of nodes to do the job
- Temptation can be to use many discrete nodes as message flow development is visual
- More nodes can make flow more self-documenting
- But more nodes can also add overhead without enhancing function



# Watch Subflows

- Subflows are useful but may have hidden side-effects
  - ▶ Watch for consecutive Compute nodes



# CommitCount and CommitInterval

- Can be used to control the UOW size and duration for a message flow
  - ▶ CommitCount - Number of input messages to process before issuing a commit
    - Only applies when using MQInput node
  - ▶ CommitInterval - Max time since the last input message before issuing a commit
    - Use to prevent unfulfilled UOWs running too long
  - ▶ Specified on the BAR file (requires a redeploy to change)
  - ▶ Defaults to CommitCount(1) and CommitInterval(0)
- May be useful when processing messages under high volume
  - ▶ Especially when the flow is putting messages
  - ▶ Effect is to batch messages and commit as a group
- There are things to think about, though
  - ▶ Count only includes number of MQGet operations performed by the MQInput node
  - ▶ Values apply to each flow instance (thread)
  - ▶ Can impact queue performance if too many flow instances
  - ▶ Can impact the MQ log (AMQ7469, etc)
  - ▶ Benefit can be very dependant on MQ logger tuning/performance (log buf size, etc)
  - ▶ Can impact getting applications

# CommitCount and CommitInterval

N

O

T

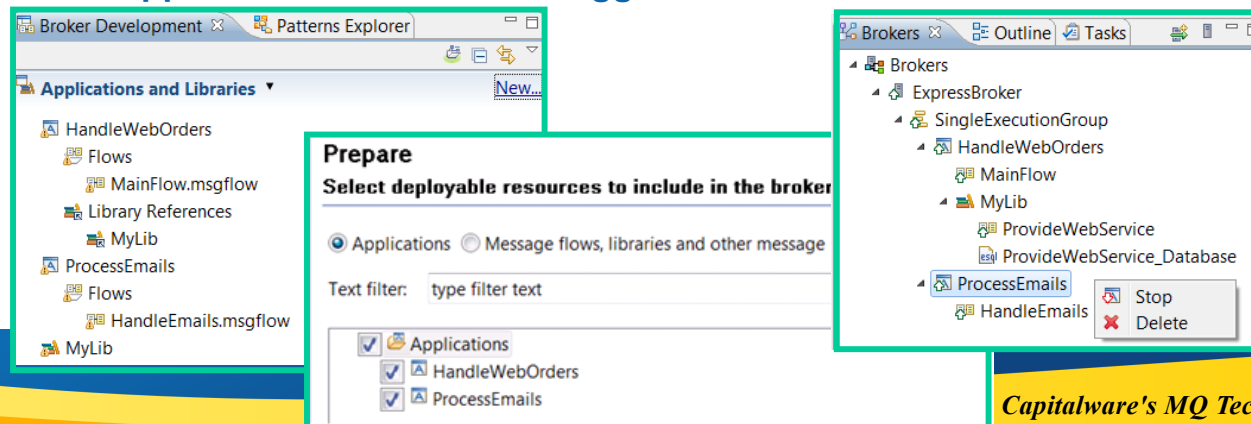
E

S


- The CommitCount and CommitInterval message flow properties can be used to control the size, and to a lesser extent the duration, of a UOW when processing MQ messages.
- It is common knowledge that In WebSphere MQ, the performance of persistent messages is almost always improved by putting and getting messages as part of a transaction. This allows the queue manager to “batch up” messages when writing them to the transaction log. CommitCount takes advantage of this.
- CommitCount says how many messages can be processed by an MQInput node before a commit is issued. If the flow does other MQ work (putting to queues, publishing, etc) this does not count towards the CommitCount total – only the MQGets done by the MQInput node count towards this.
- CommitInterval says how long to wait after processing the most recent input message before issuing a commit, in the case where the current “batch” of messages is not complete. This should be used to prevent unfulfilled UOWs running too long, potentially impacting the MQ log (AMQ7469 errors), preventing messages from being processed downstream in a timely manner, etc.
- The defaults for these are CommitCount(1) and CommitInterval(0).
- These values are specified on the BAR file. They cannot be changed using IIBs operational tools – to change either of these values require that the BAR file be changed and redeployed.
- Generally speaking, customers leave these values to default. However, if you know message arrival rates can be very high, or perhaps very spiky, you may gain some throughput improvement through prudent use of these parameters.
- Why “prudent use”? Because there are things that must be taken into consideration:
  - Only applies to flows initiated by an MQInput node. Has no effect when other input nodes used
  - Only includes the MQGet operations performed by the MQInput node that started the flow – other MQ operations are not included in the count. Thus a CommitCount of 10 for a flow with 1 MQInput node and 3 MQOutput nodes will result in as many as 40 messages being held under syncpoint.
  - Applies to each instance of the flow – not the instances in aggregate. Thus in the previous scenario, in there were 20 flow instances there could be as many as 800 messages being held under syncpoint.
  - Can degrade overall MQ queue performance, especially if there are a non-trivial number of instances assigned to the flow – see previous examples of why this may be the case.
  - Can also impact the behavior of the MQ log (AMQ7469, etc) – again, see previous examples.
  - Benefit can be very dependant on MQ logger tuning/performance – e.g. a too-small LogBufferPages value may result in little benefit resulting from using this.
  - Also give thought to the impact to downstream getting applications – again, see previous examples.

# Applications, Libraries, Services and REST APIs

- **Applications package end-to-end connectivity solutions**
  - ▶ The concept of an application is shared between the toolkit and runtime
  - ▶ Applications are deployed and managed as a single unit of isolation
- **Libraries package resources for reuse (flows, scripts, models)**
- **Resources in an application are not visible to anything else**
  - ▶ Use applications to manage your solutions inside an execution group
- **A Service is an Application with a well defined interface (WSDL)**
- **A REST API is an Application built from a Swagger definition**



# Shared Libraries



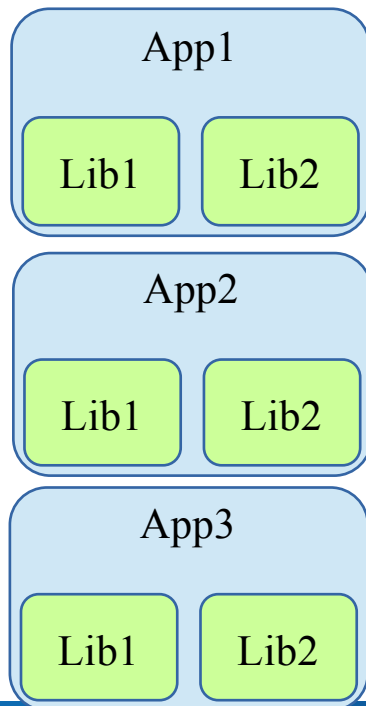
V10

- **New shared container for reusable artefacts**
  - ▶ Subflows, message models (XSD, DFDL)
  - ▶ ESQL, maps
  - ▶ NOT flows!
- **Saves memory**
  - ▶ Multiple applications can reference a single copy of a shared lib
- **Separately deployable from the application**
- **Shared libraries have no running state**
  - ▶ Cannot be started or stopped
  - ▶ No runtime threads assigned
- **Pre-V10 Static Libraries continue to work as before**

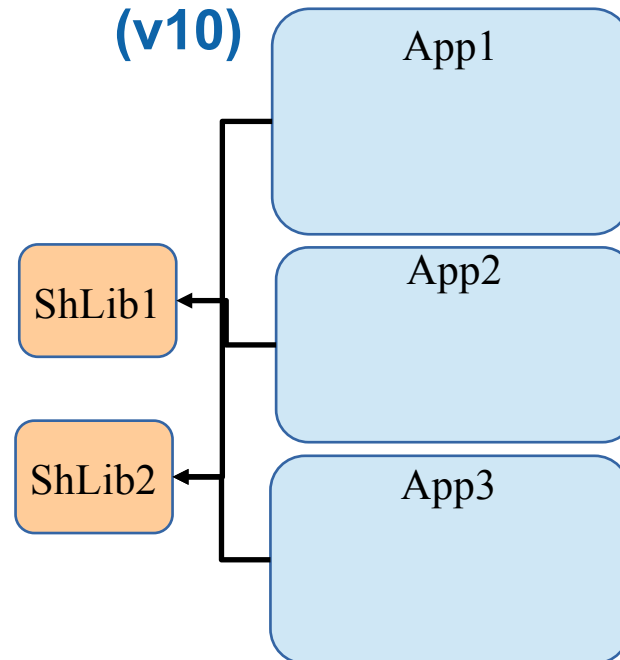


# Static Libraries versus Shared Libraries

## ■ Static Libraries



## ■ Shared Libraries (v10)



# Parsing

## ■ The means of populating and serializing the tree from data

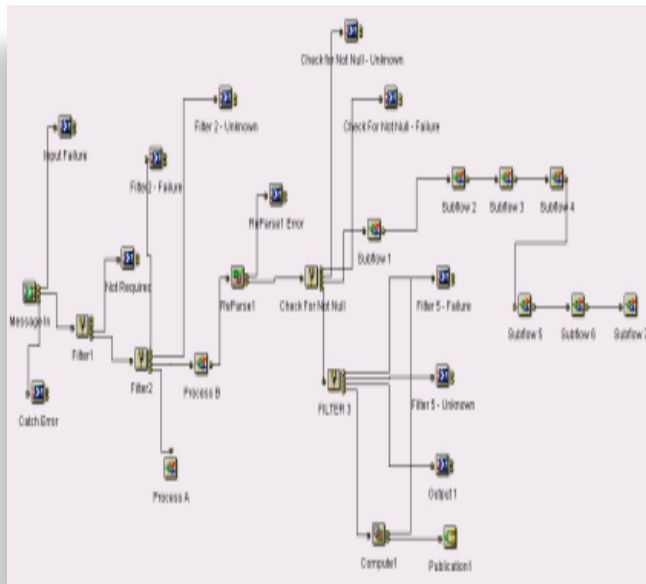
- ▶ Can occur whenever the message body is accessed – explicit and implicit parsing
- ▶ Multiple Parsers available: XMLNSC, MRM XML, CWF, TDS, MIME, JMSMap, JMSStream, BLOB, IDOC, RYO
- ▶ Message complexity varies significantly ...and so do costs!

## ■ Several ways of minimizing parsing costs

- ▶ Use cheapest parser possible, e.g. XMLNSC for XML parsing, DFDL for non-XML
- ▶ Identify the message type quickly
- ▶ Use parser optimization techniques
  - Parsing avoidance
  - Partial parsing or parsing on-demand
  - Opaque parsing

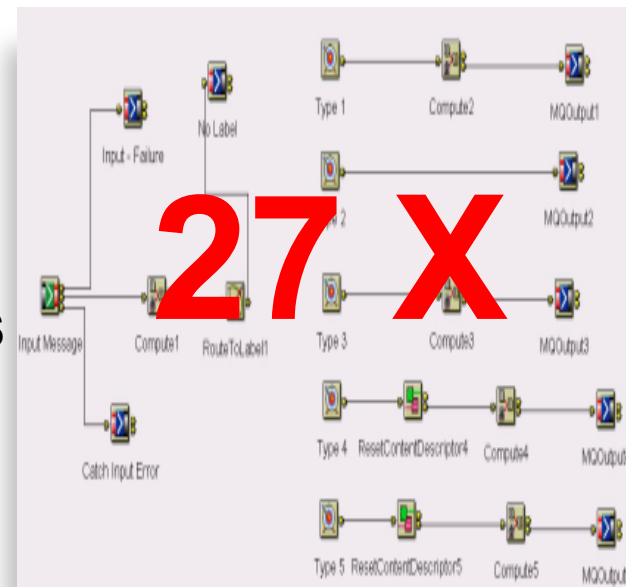
# Identifying the message type quickly

- Avoid multiple parses to find the message type



5 msgs/sec

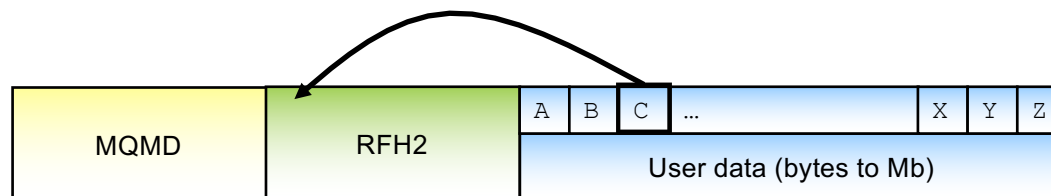
VS



138 msgs/sec

# Parser avoidance

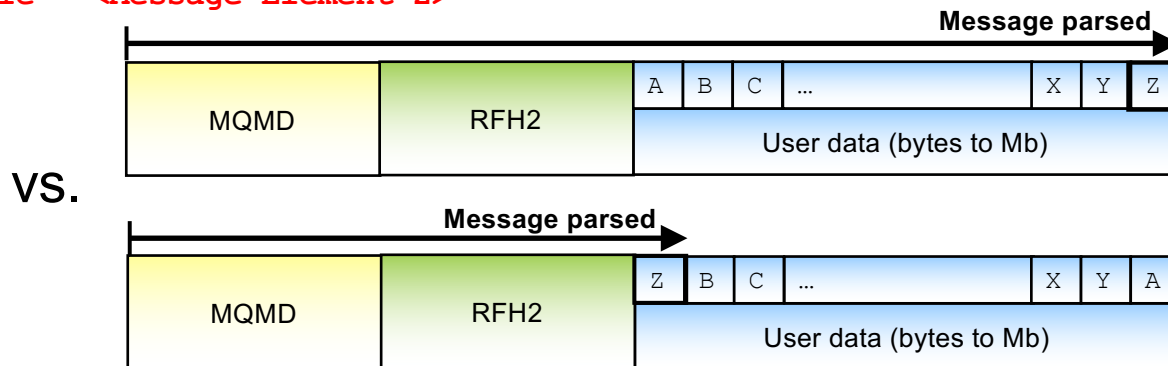
- **If possible, avoid the need to parse at all!**
  - ▶ Consider only sending changed data
  - ▶ Promote/copy key data structures to MQMD, MQRFH2 or JMS Properties
    - May save having to parse the user data
    - Particularly useful for message routing



## Partial parsing (On Demand)

- Typically, IIB parses elements up to and including the required field
  - Elements that are already parsed are not reparsed
- If possible, put important elements nearer the front of the user data

Set MyVariable = <Message Element Z>



Typical Ratio of CPU Costs	1K msg	16K msg	256K msg
Filter First	1	1	1
Filter Last	1.4	3.4	5.6

# Opaque parsing

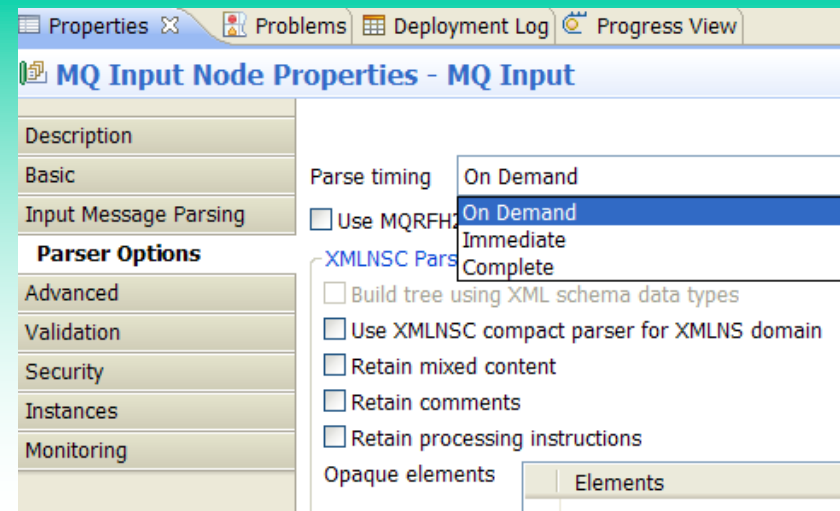
- Treat elements of an XML document as an unparsed BLOB
- Reduces message tree size and parsing costs
- Cannot reference the sub tree in message flow processing
- Configured on input nodes (“Parser Options” tab)
  - ▶ Only provides benefit when parse “On Demand” is selected

<code>&lt;order&gt;</code>	
<code>&lt;name&gt;</code> <code>&lt;first&gt;John&lt;/first&gt;</code> <code>&lt;last&gt;Smith&lt;/last&gt;</code> <code>&lt;/name&gt;</code>	DON'T PARSE THIS
<code>&lt;item&gt;Graphics Card&lt;/item&gt;</code> <code>&lt;quantity&gt;32&lt;/quantity&gt;</code> <code>&lt;price&gt;200&lt;/price&gt;</code>	
<code>&lt;date&gt;06/24/2010&lt;/date&gt;</code>	OR THIS
<code>&lt;/order&gt;</code>	

# Specifying Partial Parsing

- Applies to Input nodes
  - On Demand
  - Immediate
  - Complete

- **Parser Options Tab of Input Node:**



# Specifying Opaque parsing

- Only on:



MQ Input



HTTP Input



TCPIP Client Input



TCPIP Server Input

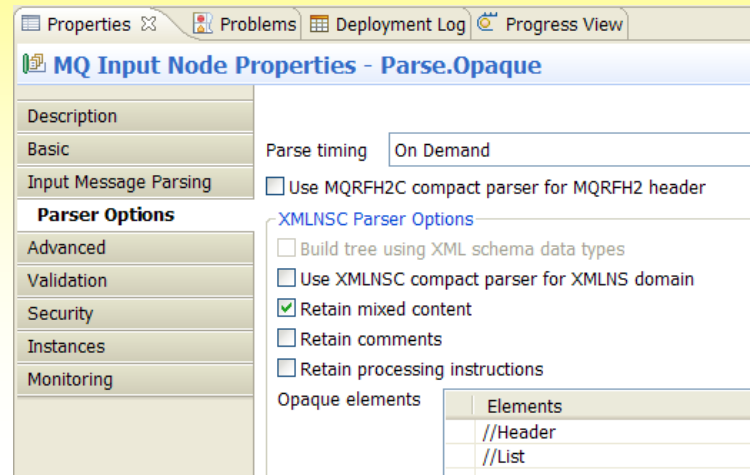


JMS Input



JMS Receive

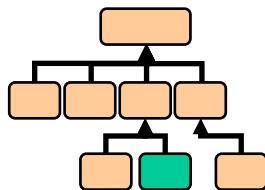
- Opaque:





# Navigation

- **The logical tree is walked every time it is evaluated**
  - ▶ This is not the same as parsing!



```
SET Description = Root.Body.Level1.Level2.Level3.Description.Line[1];
```

- **Long paths are inefficient**
  - ▶ Minimise their usage, particularly in loops
  - ▶ Use reference variables/pointers (ESQL/Java)
  - ▶ Build a smaller message tree if possible
    - Use compact parsers (XMLNSC, DFDL, MRM XML, RFH2C)
    - Use opaque parsing

# Message Tree Elements Definition Order

- **Watch the order in which you define message tree elements**

- ▶ When constructing the OutputRoot message tree structure (for an XML message) individual elements must be created in the correct sequence as defined in the XSD and message set/schema.
- ▶ The parser does not re-order the elements [True for ESQL and Java]
- ▶ Example of defining order in ESQL:

```
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Surname';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Inits';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Addr1';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Addr2';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Addr3';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Postcode';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Account_Number';
CREATE LASTCHILD OF OutputRoot.XMLNSC.MsgStruct NAME 'Account_Bal';
```

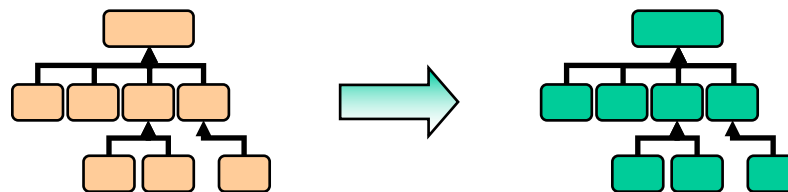
- This code creates the right elements, and in the correct sequence. Then later on, when the elements are populated (generally using code like

- **When setting elements like `SET OutputRoot.XMLNSC.MsgStruct.<element> = ...`, each is already there and so the sequence is maintained.**

**Note:** Not necessary to code the DOMAIN clause on every CREATE LASTCHILD statement. When creating a child node under parent P, the child is created by P's parser. So the parser (i.e. the domain) automatically propagates down the tree from the root. There is a performance and memory gain to be had as a result of not coding DOMAIN on the creation of the child elements.

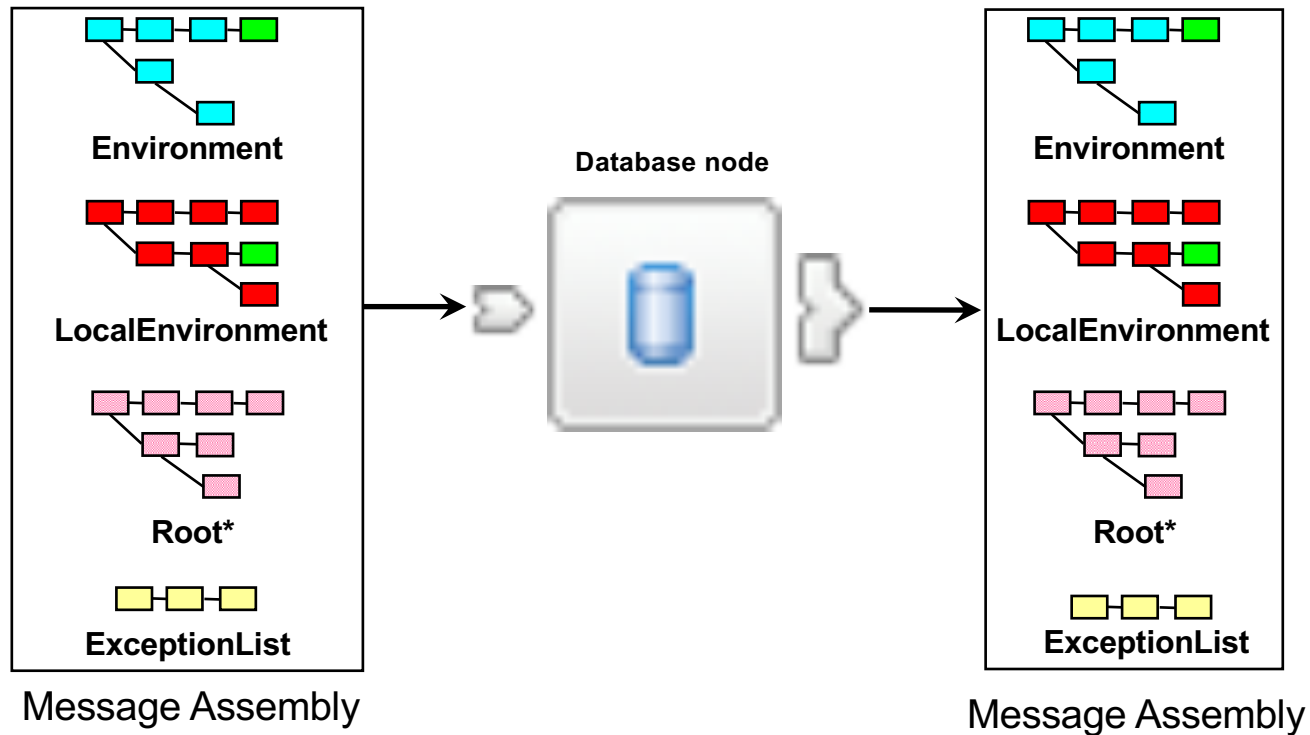
# Message tree copying

```
SET OutputRoot.XML.A = InputRoot.XML.A;
```



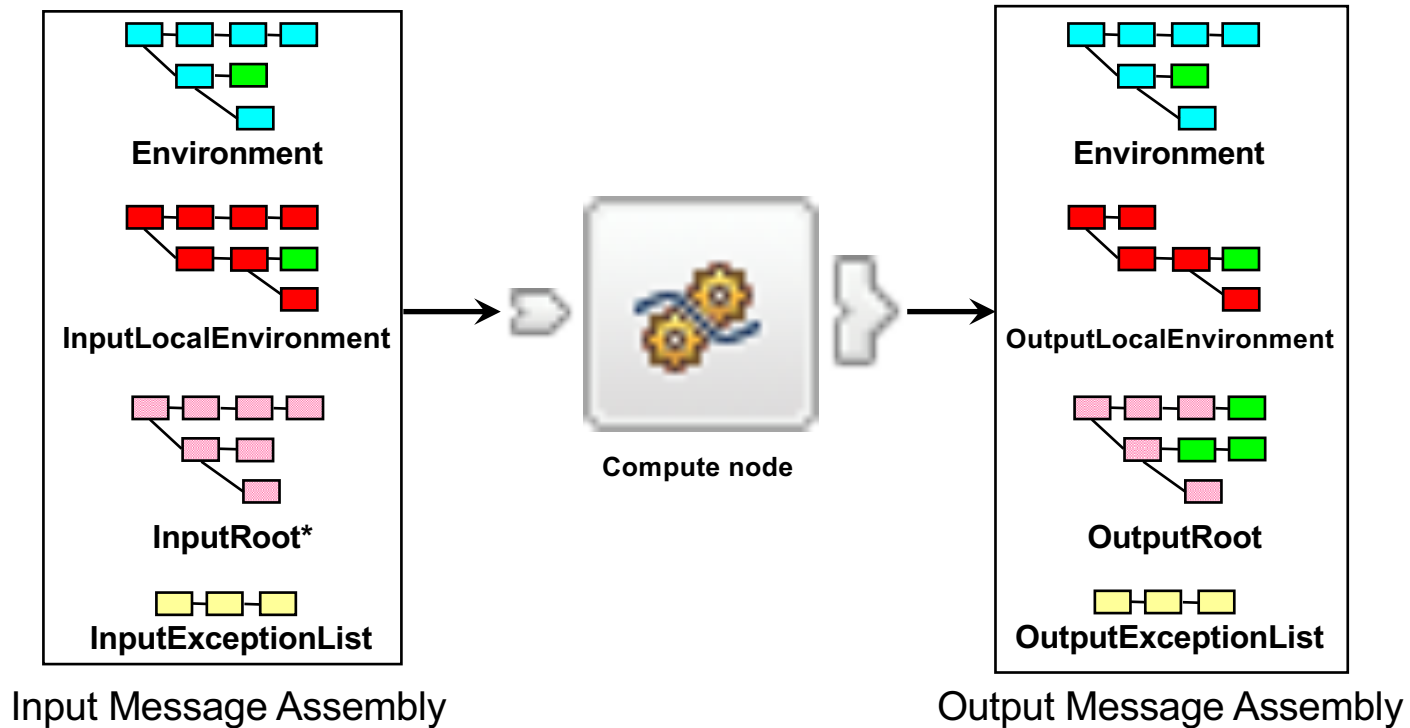
- Message tree copying causes the logical tree to be duplicated in memory... and this is computationally expensive
- Reduce the number of times the tree is copied
  - ▶ Reduce the number of *Compute* and *JavaCompute* nodes in a message flow
  - ▶ See if “Compute mode” can be set to not include “message”
  - ▶ Copy at an appropriate level in the tree (copy once rather than for multiple branch nodes)
  - ▶ Copy data to the Environment (although changes are not backed out)
- Minimize the effect of tree copies
  - ▶ Produce a smaller message tree (again)!

## How Many Trees? (Database Node)



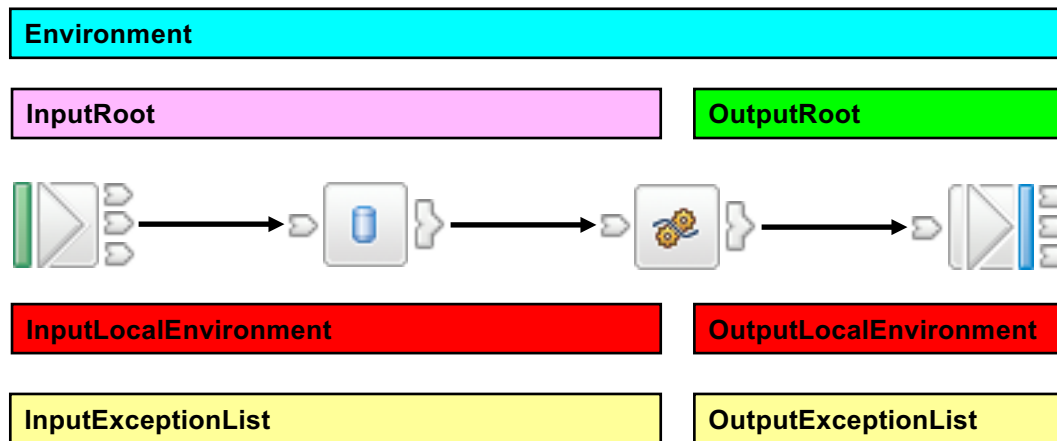
\* = Normally Read Only (not modifiable)

## How Many Trees? (Compute Node)



\* = Normally Read Only (not modifiable)

# Message Tree Timelines: From the Compute Node Perspective

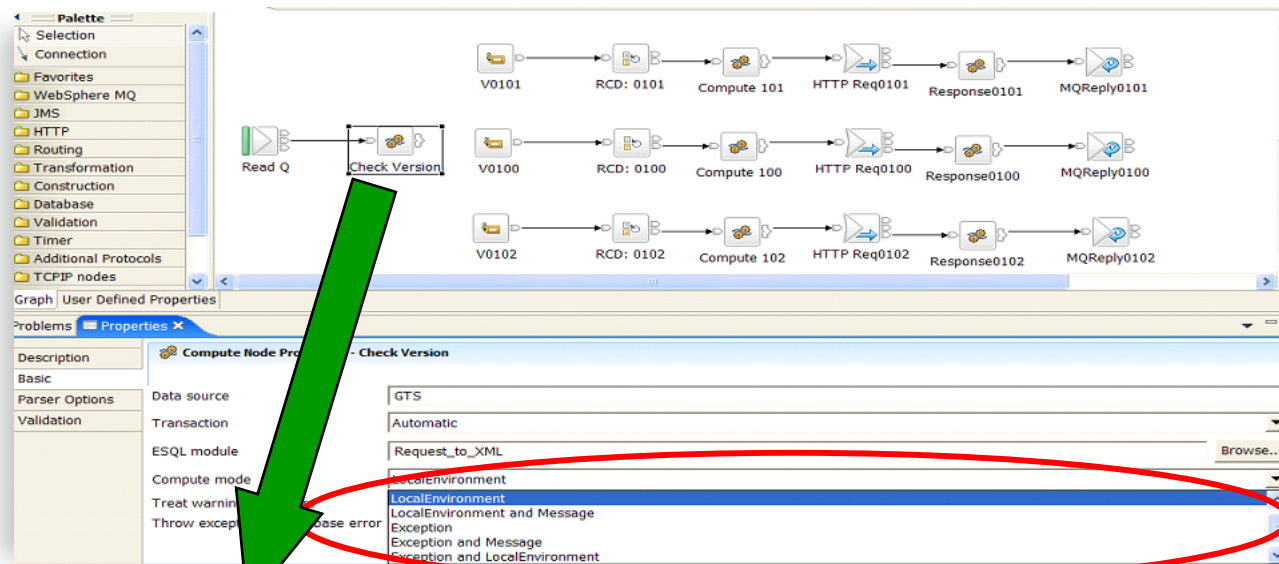


**Compute Mode Property**

- Message
- LocalEnvironment
- LocalEnvironment And Message
- Exception
- Exception And Message
- Exception And LocalEnvironment
- All

*Note: The Input\* trees actually exist beyond the Compute node, It's just you cannot access them*

## Example of avoiding a tree copy



```
-- Set the version numbers this flow supports
SET Environment.SupportedVersions = '0102 0101 0100';

-- getMessageVersion returns label name for the message version (for example, '0102')
PROPAGATE TO LABEL 'V' || getMessageVersion(Environment, InputRoot, TRUE);

-- PROPAGATE already passed the message; do not propagate to "Out"
RETURN FALSE;
```

#MaximizeYourGame

# Deployment Considerations



# Deployment Considerations

- ▶ A discussion of strategies to use when deploying message flows
  - Additional *Instances* vs Multiple *Integration Servers*
  - Maximising availability
  - Determining how many copies to deploy

**How many Integration Servers should I have?  
How many Additional Instances should I add?**

**Integration Servers**

- Results in a new process/address-space
- Increased memory requirement
- Multiple threads including management
- Operational simplicity
- Gives process level separation
- Scales across multiple servers

**Additional Instances**

- Results in more processing threads
- Low(er) memory requirement
- Thread level separation
- Can share data between threads
- Scales across multiple servers

**Recommended Usage**

- Check resource constraints on system
  - How much memory available?
  - How many CPUs?
- Start low (1 server, No additional instances)
- Group applications in a single integration server
- Assign heavy resource users to their own integration server
- Increment integration servers and additional instances one at a time
  - Keep checking memory and CPU on machine
- Don't assume configuration will work the same on different machines
  - Different memory and number of CPUs

Ultimately  
have  
to  
balance  
Resource  
Manageability  
&  
Availability

# Message Flows – Development and Execution

- **Message flows are developed in the Integration Toolkit**
  - ▶ This is a development environment only and does not provide runtime support
    - Although Note! By default a node instance is provided OOTB from v10 onwards.
  - ▶ Require a node instance with at least one Integration Server to run a message flow
  - ▶ Toolkit can be co-located with a node instance – often the case in development
  
- **Solutions (Applications, Services, message flows, etc) deployed using one of:**
  - ▶ mqsideploy command
  - ▶ Integration API
  - ▶ Integration Toolkit
  - ▶ Integration Explorer (not in v10, where Web UI BAR deploy is introduced)
  
- **Key questions**
  - ▶ How many copies of the message should be run
  - ▶ Where should they be run

# Integration Servers

- **An Integration Server (DataFlowEngine) is a runtime container for Applications, etc**
  - ▶ Hosts the message flow(s) which do the integration work
  - ▶ Can support one or many message flows per Integration Server
  
- **Consists of:**
  - ▶ Message flows
  - ▶ Message sets/schemas
  - ▶ Embedded JVM
  - ▶ Configurable services
  - ▶ Network connections (HTTP/TCP)
  - ▶ Node management infrastructure threads
  
- **Integration Servers provide a good level of separation**
  - ▶ Process level verses Thread level

## Estimate of Resources Used for Additional Capacity

	Message Flow Instance[1]	Integration Server	Node Instance
<b>Memory</b>	Few MB	100 MB+	MB - GB[2]
<b>CPU</b>	Depends on flow [3]	Depends on flow [3]	Depends on flow [3]
<b>Disk</b>	None	None	100 MB + Queue manager instance

**Notes:**

[1] For a message flow instance in an existing Integration Server

[2] Depends on complexity of configuration

[3] Will be the same level of usage for each option

# Process Limits/Constraints

## ■ Constraints

- ▶ Subject to per process limits. For example:
  - data - max data size (KB)
  - fsize - maximum filesize (KB)
  - nofile - max number of open files
  - stack - max stack size (KB)
  - nproc - max number of processes
  - locks - max number of file locks the user can hold
  - msgqueue - max memory used by POSIX message queues (bytes)

## ■ An Integration Server has a memory overhead which is larger than the per flow overhead

- Due to executables, JVM, management threads
- Overhead of an Integration Server running on one node versus another is no different

## ■ 64-bit addressing removes virtual storage constraint than existed with 32-bit

- ▶ This was the major inhibitor to assigning many message flows to an Integration Server

## ■ Maximum of 256 threads for additional instances

- ▶ On a message flow and for number of threads for an input node

## Additional Flow “Instances” – How Many?

- **Instances = Threads**
- **Easy to misuse (and often is!)**
  - ▶ e.g. Temptation is to increase instances when input queue backing up
  - ▶ This often backfires
- **More is not always better**
- **Maximum of 256 threads for additional instances**
  - ▶ On a message flow and for number of threads for an input node
  - ▶ Specify more and Deploy will fail

## How Many Copies of a Message Flow Are Needed

- **Each message flow is different**
  - ▶ Will use certain different level of resources (CPU, memory, I/O) and have a particular performance profile
  
- **In deciding number of copies to run need to know**
  1. Availability Requirements
  2. Target throughput (messages/second) rate
  3. Target response time
  
- **Number of copies of each message flow needed will vary**
  - ▶ So do not decide the same number of copies are sufficient in all cases



## Availability

- **An Integration Server will be restarted automatically after failure**
  - ▶ Should return within a short time – provided by Integration Bus
- **However there will be an outage during the restart**
  - ▶ So if a message flow is deployed to only one Integration Server there could be outages
  - ▶ May wish to consider deploying a message flow to more than one Integration Server solely for this reason, particularly for critical applications or services
- **Remember to treat all message flows for an application or service equally**
  - Often more than one message flow involved – Request *and* Reply for example
- **In planning of more robust systems look at active/active systems**
  - Running duplicate deployments over separate machines in different locations

# Levels of Availability and Permitted Downtime

- **What does your SLA say?**

- ▶ Scheduled and unscheduled outages

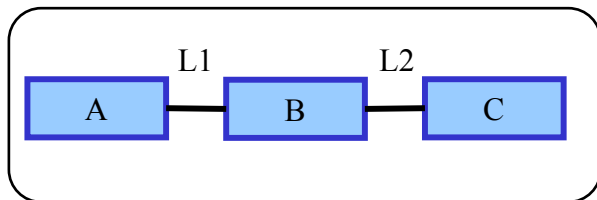
Availability %	Downtime per year	Downtime per month	Downtime per week
90 (one nine)	36.5 days	72 hours	16.8 hours
99	3.65 days	7.2 hours	1.68 hours
99.9	8.76 hours	43.2 min	10.10 min
99.99	52.6 min	4.32 min	1.01 min
99.999 ("five nines")	5.26 min	25.9 secs	6.05 secs
99.9999	31.5 secs	2.59 secs	0.605 secs

- **Ensure you have the technology to meet the SLA**

- ▶ Redundancy in applications
- ▶ Clustering, PowerHA (HACMP)
- ▶ What about applying service?

# Availability – Lets do the maths!

- Consider a system with three components (A,B,C) & two links (L1, L2)



## Availability:

- Let's imagine that components A and C, and the links L1 and L2 are 99.999% available
- Let's imagine that component B is significantly less reliable at 90%
- System availability as a whole is given by a product of availabilities:

$$(0.99999) * (0.99999) * (0.90) * (0.99999) * (0.99999) = 89.996\%$$

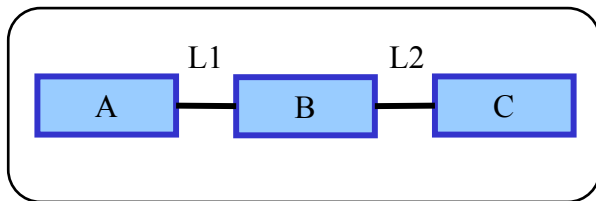
**A**                      **L1**                      **B**                      **L2**                      **C**

### Notes:

- [1] System availability as a whole is below the 90% level of the weakest component (B)
- [2] The "five nines" of the components A,C,L1 and L2 do not compensate for component B!
- [3] Availability under 90% is equivalent to over 2.4 hours downtime **per day!**

# Availability – The benefit of redundancy!

- Consider a system with three components (A,B,C) & two links (L1, L2)



## Availability:

- Let's imagine that components A and C, and the links L1 and L2 are 99.999% available
- Let's imagine that component B is significantly less reliable at 90%
- System availability as a whole is given by a product of availabilities:

$$(0.99999) * (0.99999) * (0.90) * (0.99999) * (0.99999) = 89.996\%$$

**A**

**L1**

**B**

**L2**

**C**

### Notes:

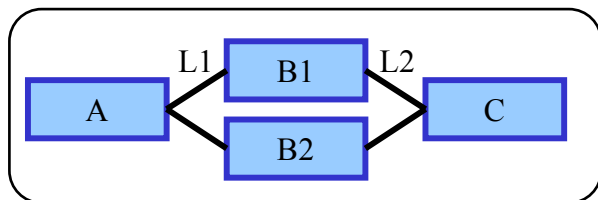
[1] System availability as a whole is below the 90% level of the weakest component (B)

[2] The "five nines" of the components A,C,L1 and L2 do not compensate for component B!

[3] Availability under 90% is equivalent to over 2.4 hours downtime **per day!**

# Availability – The benefit of redundancy!

- Component B is the weakest part of the system, so let's duplicate it!



## Availability:

- As before, components A and C, and the links L1 and L2, are 99.999% available
- Although duplicated in the architecture, components B1 and B2 are still significantly less reliable at 90%
- Availability of B1 and B2 together is given by:

$$1 - [ (1-0.90) * (1-0.90) ] = 0.99$$

- So, the new system availability as a whole is given by:

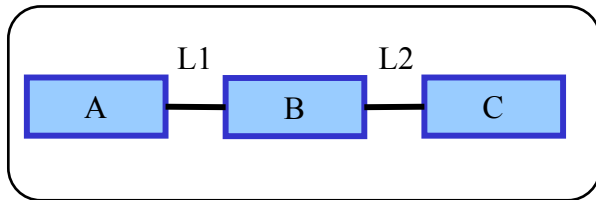
$$(0.99999) * (0.99999) * (0.99) * (0.99999) * (0.99999) = 98.996\%$$

**A**      **L1**   **B1 or B2**   **L2**      **C**

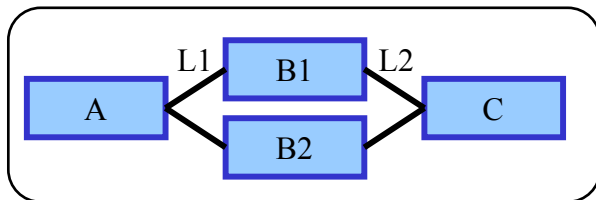
### Notes:

- Availability is now approximately 14.5 minutes per day!

## Availability – Comparing the two!



**89.996%**  
Availability  
[~2.4 hours down per day]



**98.996%**  
Availability  
[~14.5 mins down per day]

- **Same processing components: A, L1, B, L2, C**
- **Same levels of availability for each component**
  - ▶ A, C, L1, L2 are 99.999% available.
  - ▶ B, B1 and B2 are 90% available
- **Increased availability comes from duplication of components**

# Throughput and Response Time

- **Throughput (messages per time) for a message flow will depend on:**
  - ▶ Complexity of the processing
  - ▶ Level of I/O activity
  - ▶ Response time of invoked services or applications
  - ▶ Level of resource (CPU, memory. I/O) available to the node
  
- **Same message flow on different systems will achieve different level of performance**
  - ▶ Faster CPUs should reduce time taken for the CPU component
  - ▶ Slower response from service called in-line will slower throughput
  
- **Important to measure the throughput on the target system**
  - ▶ If one copy of the message flow process 500 messages/second and 700 messages/second needed then 2 copies should be sufficient
  - ▶ But if only 50 messages/second then could need 15-20 copies

# Message Flow Assignment to Integration Servers

- **Once number of copies of the message has been determined to meet availability and throughput requirements question is how to allocate them**
  - ▶ Few resource constraints allocation can be driven by policy – not technical limitations
  
- **Factors to consider;**
  - ▶ Availability requirements – co-locate those with similar requirements. Allows an Integration Server to be shutdown if required. Otherwise it might never be possible
  - ▶ Use an EG per application/service or group of applications/services
  - ▶ Balancing of load – do not put all high volume message flows in one EG
  - ▶ Separate ‘anti-social’ applications into their own Integration Server
  - ▶ Level of redundancy and resilience
    - 1 EG with 2 instances VS 2 EG with 1 instance each
  
- **Consider allocating high profile, key services in (multiple) dedicated Integration Servers. Other services can be allocated in shared Integration Servers**
  - ▶ Provides isolation and resiliency for the high profile services and reduces over all resource consumption through the shared Integration Servers.



# Topology Summary

- **Great flexibility in topology makes for easy and flexible vertical & horizontal scaling**
  - ▶ Node
  - ▶ Integration Servers
  - ▶ Message Flows
  - ▶ Message Flow instances
  
- **Suggestions**
  - ▶ One node per operating image (LPAR)
  - ▶ Low 10's Integration Servers per node
  - ▶ 10-100's message flows including instances within the node
  
- **Reasons**
  - ▶ Single node easily capable of utilising 4/8/16/32/64 SMP system
  - ▶ Easily manageable
  - ▶ Sufficiently responsive for administrative operations

#MaximizeYourGame

# Tuning, Tools and Transports

# Tuning

- ▶ A look at the options for tuning the runtime component of IIB and also a discussion of some specific tuning for particular transports

# Tuning the Runtime Component

- **Goal - To meet the throughput and response time requirements**
  - ▶ Combination of resources used and tuning applied
  
- **Areas to consider**
  - ▶ Resources
  - ▶ Node
  - ▶ Transports
  - ▶ Databases
  - ▶ Thread Pools
  - ▶ Transactional Processing

# Resources

## ■ Processor (CPU)

- ▶ Processing is generally CPU intensive so ensure there is sufficient CPU available
  - Number and speed of CPUs
- ▶ In a virtualized environment best to have dedicated CPUs

## ■ Memory

- ▶ Amount of memory required depends on messages and message flow
- ▶ Guide is 3-4 GB per CPU

## ■ I/O

### ▶ Disk

- Speed of write is very important where logs are used (queue manager and database)
- Also for database when BLOBs are inserted
- SAN with fast write non-volatile cache is best. Goal is < 1ms for a 10K write

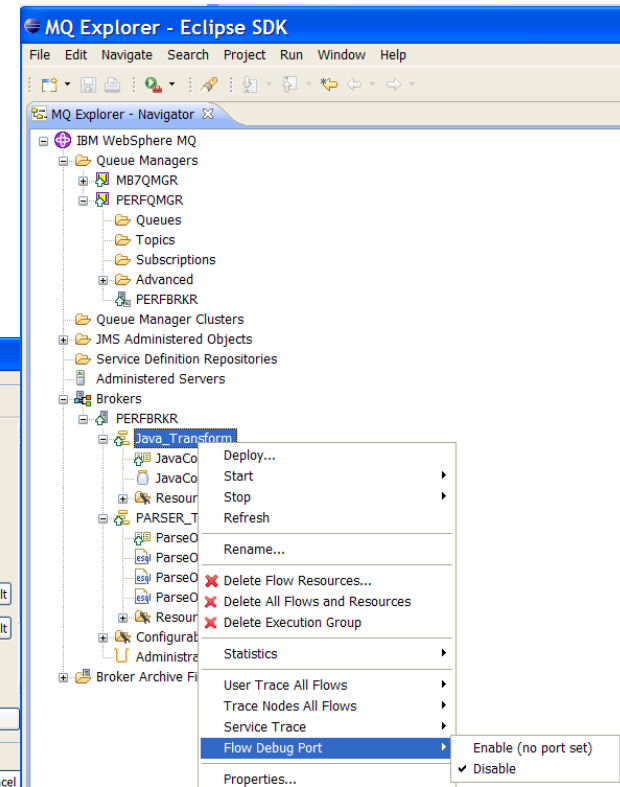
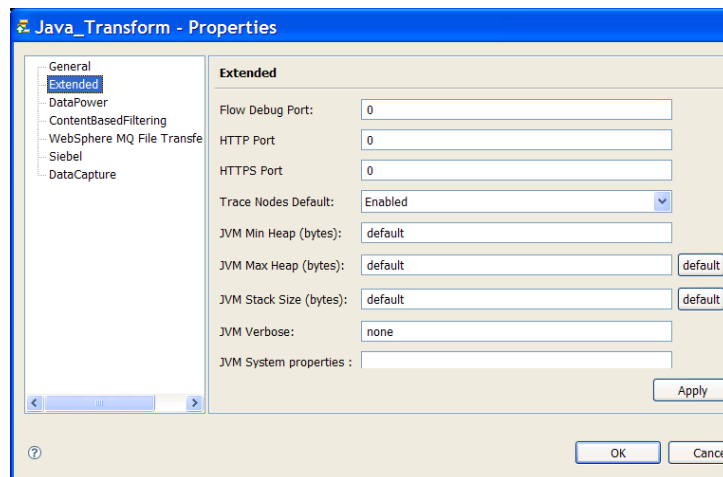
### ▶ Network

- Most environments involve more than a single machine
- Today volume of messages is increasing as is size - Recommend 10 Gb network to ensure it is not a bottleneck

# IIB Tuning

## ■ Items to check:

- ▶ Trace in all execution groups and flow
  - Service, user and trace nodes
- ▶ Java Debug Port
  - Must be disabled and set to 0



## Some tools to understand your integration node's behavior

- **PerfHarness** – Drive realistic loads through the runtime
  - ▶ <https://ibm.biz/JMSPerfHarness> and search for “PerfHarness”.
- **OS Tools**
  - ▶ Run your message flow under load and determine the limiting factor.
  - ▶ Is your message flow CPU, memory or I/O bound?
  - ▶ e.g. “perfmon” (Windows), “vmstat” or “top” (Linux/UNIX) or “SDSF” (z/OS).
  - ▶ This information will help you understand the likely impact of scaling (e.g. additional instances), faster storage and faster networks
- **Third Party Tools**
  - ▶ *RFHUtil* – useful for sending/receiving MQ messages and customising all headers
  - ▶ *NetTool/Fiddler* – useful for testing HTTP/SOAP
  - ▶ *Process Explorer* – Windows tool to show which files are in use by which processes
  - ▶ *Java Health Center* – to diagnose issues in Java nodes
- **MQ / Integration Explorer**
  - ▶ Queue Manager administration
  - ▶ Useful to monitor queue depths during tests
  - ▶ Resource Statistics, Activity Log
- **IBM Integration Bus Web UI (v9 onwards, enhancements in v10)**

# View runtime statistics using the WebUI

- Control statistics at all levels
- Easily view and compare flows, helping to understand which are processing the most messages or have the highest elapsed time
- Easily view and compare nodes, helping to understand which have the highest CPU or elapsed times.
- View all statistics metrics available for each flow
- View historical flow data



**Coordinated Request Reply MQ Application**

Start

Stop

**Statistics on**

Statistics off

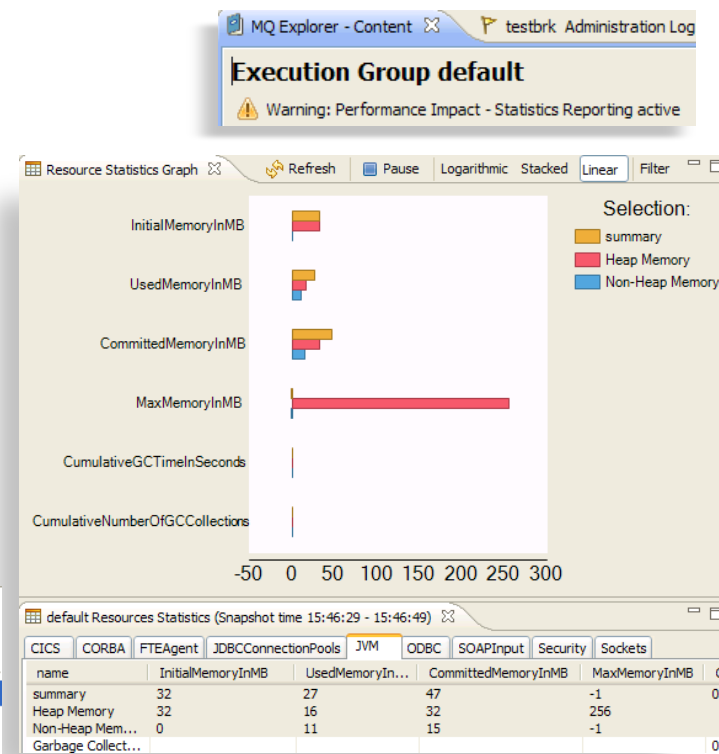
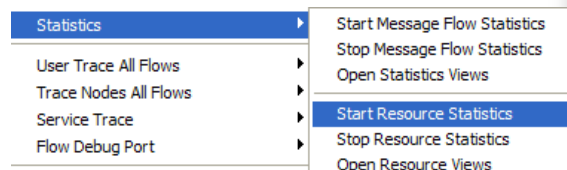
▼ Throughput per message flow for last 15 seconds. Last updated at 09:50:29 GMT Daylight Time.

Flow name	Message Rate (messages/s)	Average Elapsed Time/Invocation (ms)	Average CPU Time/Invocation (ms)
Request	1.00	4.4	0.8
Reply	1.00	3.9	0.5
BackendReplyApp	1.00	1,001.7	1.1



# Integration node resource statistics

- The IBM Integration Bus Explorer enables you to start/stop resource statistics on the integration node, and view the output.
- Warnings are displayed advising there may be a performance impact (typically ~1%)
- In V10 this moves to the Web UI



# Activity Log

## ■ Activity Logging Allows users to understand what a message flow is doing

- ▶ Complements current extensive product trace by providing end-user oriented trace
- ▶ Can be used by developers, but target is operators and administrators
- ▶ Doesn't require detailed product knowledge to understand behaviour
- ▶ Provides qualitative measure of behaviour

## ■ End-user oriented

- ▶ Focus on easily understood actions & resources
- ▶ "GET message queue X", "Update DB table Z"...
- ▶ Complements quantitative resource statistics

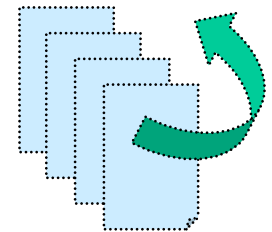
## ■ Flow & resource logging

- ▶ User can observe all events for a given flow, e.g. "GET MQ message", "Send IDOC to SAP", "Commit transaction"...
- ▶ Users can focus on individual resource manager if required, e.g. SAP connectivity lost, SAP IDOC processed
- ▶ Use event filters to create custom activity log, e.g. capture all activity on JMS queue REQ1 and C:D node CDN1

## ■ Comprehensive Reporting Options

- ▶ Reporting via IB Explorer, Web UI, log files and programmable management (CMP API)
- ▶ Extensive filtering & search options, also includes save data to CSV file for later analysis
- ▶ Rotate resource log file when reaches using size or time interval

Message...	Timestamp	Message Summary
i BIP12001I	17-Jun-2011 10:10:50.85...	Connected to JMS provider 'WebSphere_MQ'
i BIP12002I	17-Jun-2011 10:10:50.85...	Created a 'Transaction_None' session for JMS provider 'WebSphere_MQ'
i BIP12004I	17-Jun-2011 10:10:50.93...	Created JMS producer for destination 'ASYNCREQUESTQ'
i BIP12007I	17-Jun-2011 10:10:50.93...	Sent a JMS message to queue 'ASYNCREQUESTQ'
i BIP12004I	17-Jun-2011 10:10:50.52...	Created JMS producer for destination 'ASYNCRECEIVEQ'
⊗ BIP12014E	17-Jun-2011 13:47:51.65...	Failed to send message to 'ASYNCRECEIVEQ'
i BIP12001I	17-Jun-2011 13:47:54.99...	Connected to JMS provider 'WebSphere_MQ'
i BIP12004I	17-Jun-2011 13:47:55.00...	Created JMS producer for destination 'ASYNCRECEIVEQ'



# Transports Tuning

- Tuning is specific to the transport
- Commonly used transports
  - ▶ JMS
  - ▶ MQ
  - ▶ HTTP/SOAP
  - ▶ TCP/IP

# JMS

- **Follow provider instructions**

- ▶ Follow MQ tuning if using MQ as JMS provider

- ▶ [http://www.ibm.com/developerworks/websphere/library/techarticles/0604\\_bicheno/0604\\_bicheno.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0604_bicheno/0604_bicheno.html)

# IBM MQ

- **MQ defaults are set to provide messaging capability with small footprint**
  - ▶ Aim is to keep memory (buffers) and disk sizes (log extents) down
  - ▶ Not good for peak performance and tuning is needed in many cases
  
- **Tuning required depends on type of processing being performed**
  - ▶ For non-persistent messages look at queue buffer sizes
  - ▶ For persistent messages look at queue buffer sizes and log buffer and extent sizes
  
- **Persistent messages only where required**
  - ▶ Do use persistent messages as part of a co-ordinated transaction though
    - Not doing so can dramatically slow throughput – Especially when using Additional Instances!
  
- **Use fast storage if using persistent messages**
  
- **For detailed tuning advice see**  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0712\\_dunn/0712\\_dunn.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0712_dunn/0712_dunn.html)

# HTTP

- **Choice of listeners**

- ▶ Node wide or Integration Server specific

- **Node wide listener**

- ▶ Benefits

- Exposes a single port address externally
- Excellent opportunity for scalability as any URI can be hosted in any integration server(s)
- Opportunity to vary number of instances of the message flow by integration server
- HTTPInput and HTTPReply nodes can be in different message flows and servers

- ▶ Drawbacks

- Single port – this can be extended though through use of the proxy servlet
- All traffic must flow over a common pair of MQ queues

- **Integration Server specific**

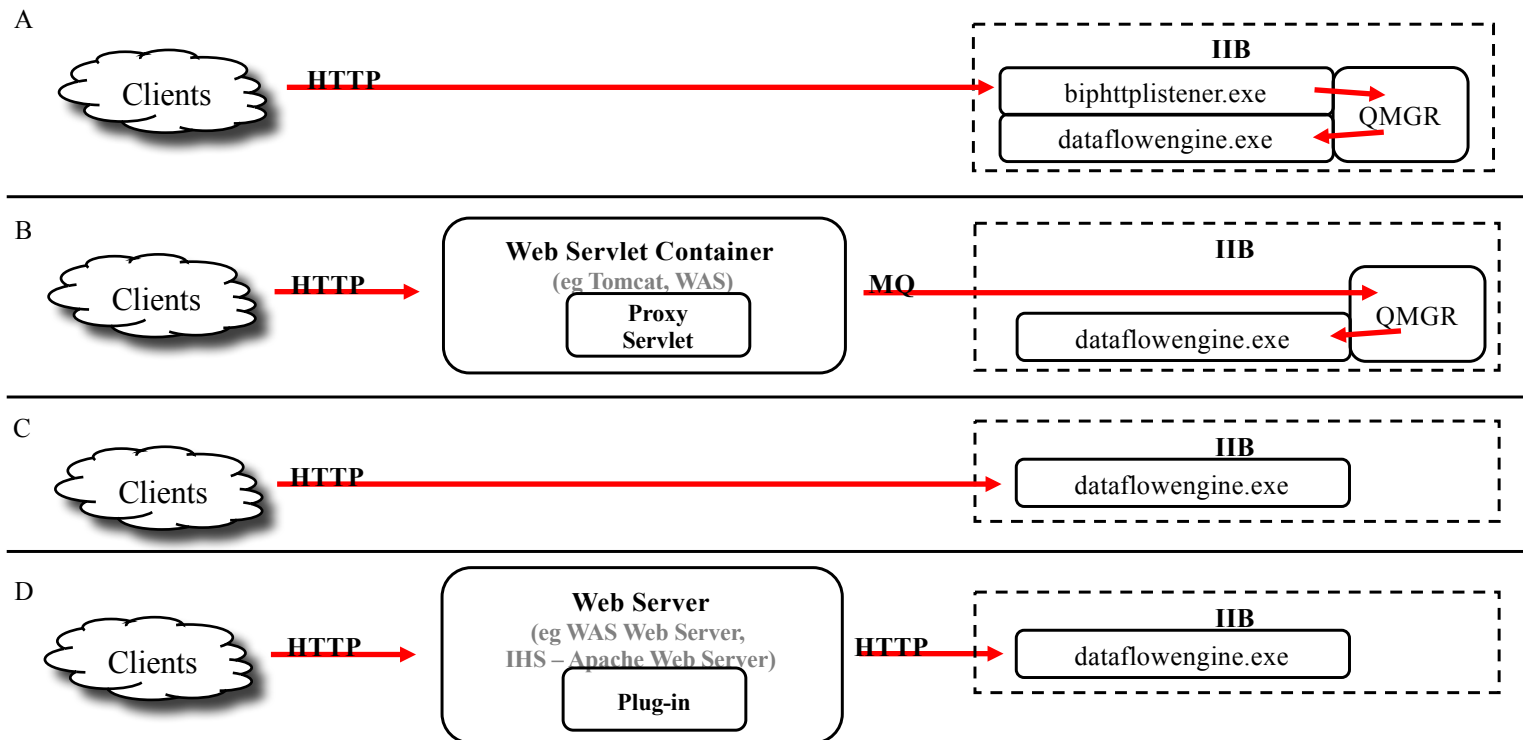
- ▶ Benefits

- Server listener always available for HTTP & HTTPS messages; both listeners are configured initially with a default configuration, and are started by the node when required.
- No intermediate queues between listener and message flow
- Can deploy message flows to different integration servers so HTTP & HTTPS messages can be handled by multiple listeners on multiple ports to meet high throughput needs

- ▶ Drawbacks

- Both HTTPInput and HTTPReply nodes must be in the same message flow, or in separate flows but the same Server

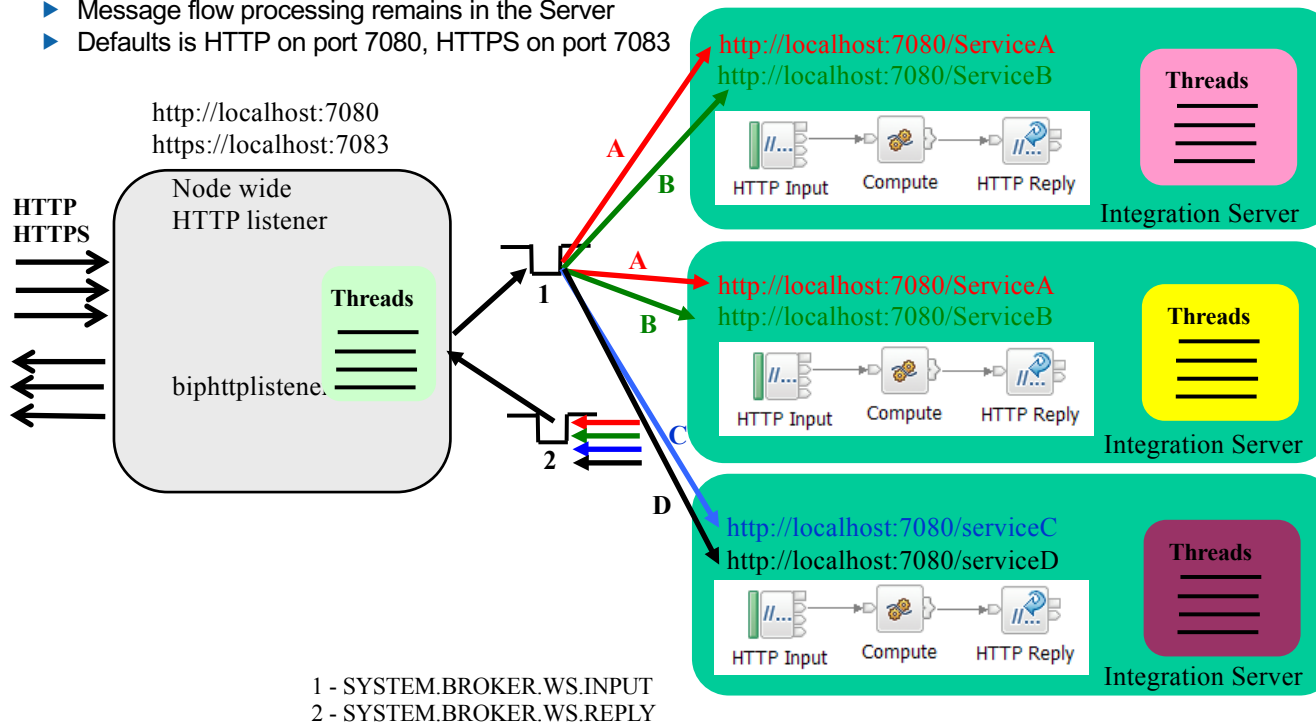
# HTTP transport and the IIB node-wide listener



# Node Wide HTTP(S) Listener

- Receives incoming HTTP requests & sends replies

- ▶ Message flow processing remains in the Server
- ▶ Defaults is HTTP on port 7080, HTTPS on port 7083



Well known mapping of URL to MQ Correlid allows any Server to host a url/service



## Node Wide HTTP(S) Listener

- **Key Settings are on**

- ▶ HTTPConnector, HTTPSConnector
- ▶ HTTPListener is listed as an item but unlikely to need to change this

- **Commands**

- ▶ Display using mqsireportproperties command
- ▶ Change using mqsichangeproperties command

# Displaying Node Level HTTPConnector Settings

```
mqsireportproperties TESTNODE -b httplistener -o HTTPConnector -r
```

```
HTTPConnector
  uuid='HTTPConnector'
  address=''
  port='7000'
  allowTrace=''
  maxPostSize=''
  acceptCount=''
  bufferSize=''
  compressableMimeTypes=''
  compression=''
  connectionLinger=''
  connectionTimeout=''
  maxHTTPHeaderSize=''
  maxKeepAliveRequests=''
  maxSpareThreads=''
  maxThreads=''
  minSpareThreads=''
  noCompressionUserAgents=''
  restrictedUserAgents=''
  socketBuffer=''
  tcpNoDelay=''
  enableLookups='false'
```

```
HTTPSConnector
  uuid='HTTPSConnector'
  algorithm='Platform Default'
  clientAuth='Platform Default'
  keystoreFile='Platform Default'
  keystorePass='*****'
  keystoreType='Platform Default'
  sslProtocol='Platform Default'
  ciphers='Platform Default'
  address=''
  port=''
  allowTrace=''
  maxPostSize=''
  acceptCount=''
  bufferSize=''
  compressableMimeTypes=''
  compression=''
  connectionLinger=''
  connectionTimeout=''
  maxHTTPHeaderSize=''
  maxKeepAliveRequests=''
  maxSpareThreads=''
  maxThreads=''
  minSpareThreads=''
  noCompressionUserAgents=''
  restrictedUserAgents=''
  socketBuffer=''
  tcpNoDelay=''
  enableLookups='false'
```

- Key performance related settings

- maxKeepAliveRequests (default value is 100) – # HTTP requests before connection close
- maxThreads (default is 200)
- tcpNoDelay (default is '' which is platform default value) Controls Nagles algorithm setting.

Set to *true* to avoid delaying the sending of TCP packets

# Changing Node Level HTTPConnector Settings

```
mqsichangeproperties TESTNODE -b httplistener -o HTTPConnector -n maxKeepAliveRequests -v -1 (turns off keepalive)
```

```
mqsireportproperties TESTNODE -b httplistener -o HTTPConnector -r
```

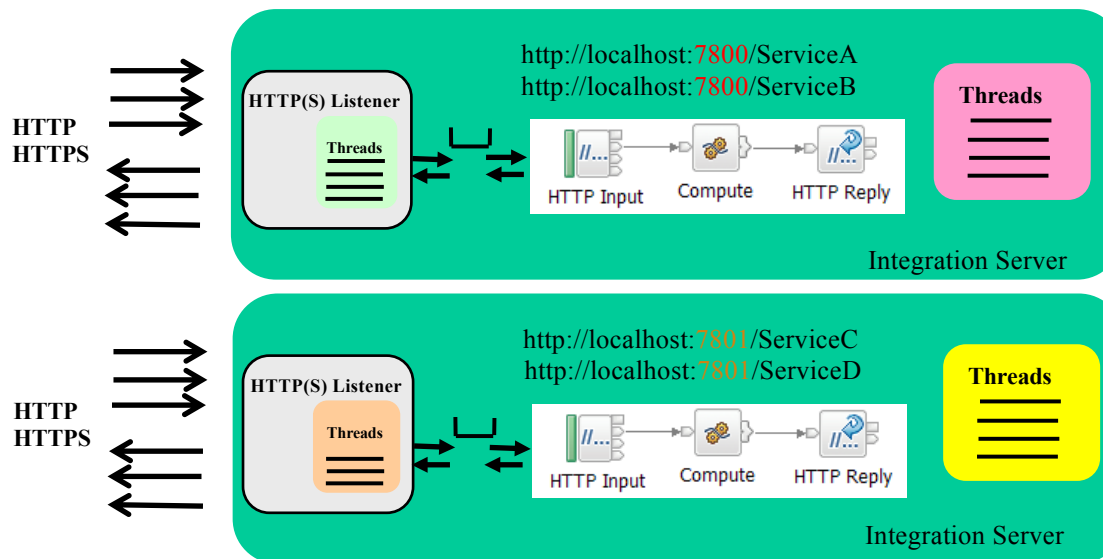
```
HTTPConnector
  uuid='HTTPConnector'
  address=''
  port='7000'
  allowTrace=''
  maxPostSize=''
  acceptCount=''
  bufferSize=''
  compressableMimeTypes=''
  compression=''
  connectionLinger=''
  connectionTimeout=''
  maxHttpRequestSize=''
  maxKeepAliveRequests='-1'
  maxSpareThreads=''
  maxThreads=''
  minSpareThreads=''
  noCompressionUserAgents=''
  restrictedUserAgents=''
  socketBuffer=''
  tcpNoDelay=''
  enableLookups='false'
```

```
HTTPSConnector
  uuid='HTTPSConnector'
  algorithm='Platform Default'
  clientAuth='Platform Default'
  keystoreFile='Platform Default'
  keystorePass='*****'
  keystoreType='Platform Default'
  sslProtocol='Platform Default'
  ciphers='Platform Default'
  address=''
  port=''
  allowTrace=''
  maxPostSize=''
  acceptCount=''
  bufferSize=''
  compressableMimeTypes=''
  compression=''
  connectionLinger=''
  connectionTimeout=''
  maxHttpRequestSize=''
  maxKeepAliveRequests='-1'
  maxSpareThreads=''
  maxThreads=''
  minSpareThreads=''
  noCompressionUserAgents=''
  restrictedUserAgents=''
  socketBuffer=''
  tcpNoDelay=''
  enableLookups='false'
```

- Key performance related settings
    - maxKeepAliveRequests (default value is 100)
    - maxThreads (default is 200)
    - tcpNoDelay (default is '' which is platform default value)
- Set to *true* to avoid delaying the sending of TCP packets

# Integration Server Level HTTP(S) Listener

- Receives incoming HTTP requests & sends replies
  - HTTP processing and message flow processing remains in the Integration Server
  - Defaults is HTTP on port 7800, HTTPS on port 7843



# Integration Server Level HTTP(S) Listener

## ■ Key Settings are on

- ▶ Integration Server
- ▶ HTTPConnector, HTTPSConnector
- ▶ [ ComIbmSocketConnectionManager for HTTP and SOAP outbound requests]

## ■ Commands

- ▶ display properties using mqsireportproperties command
- ▶ Change properties using mqsichangeproperties command

# HTTP & SOAP Outbound Requests

- Settings are specified on the `ComIbmSocketConnectionManager` component in every Integration Server

```
mqsireportproperties MB7BROKER -e EG1 -o ComIbmSocketConnectionManager -r
```

```
ComIbmSocketConnectionManager
  uuid='ComIbmSocketConnectionManager'
  userTraceLevel='none'
  traceLevel='none'
  userTraceFilter='none'
  traceFilter='none'
  resourceStatsReportingOn='inactive'
  resourceStatsMeasurements='<ResourceStatsSwitches ResourceType="Sockets" version='1'> <Measurement name="TotalSockets"
collect="on" /> <Measurement name="TotalMessages" collect="on" /> <Measurement name="TotalDataSent_KB" collect="on" />
<Measurement name="TotalDataReceived_KB" collect="on" /> <Measurement name="SentMessageSize_0-1KB" collect="on" /> <Meas
urement name="SentMessageSize_1KB-10KB" collect="on" /> <Measurement name="SentMessageSize_10KB-100KB" collect="on" /> <
Measurement name="SentMessageSize_100KB-1MB" collect="on" /> <Measurement name="SentMessageSize_1MB-10MB" collect="on" /
> <Measurement name="SentMessageSize_Over10MB" collect="on" /> <Measurement name="ReceivedMessageSize_0-1KB" collect="on"
/ > <Measurement name="ReceivedMessageSize_1KB-10KB" collect="on" /> <Measurement name="ReceivedMessageSize_10KB-100KB"
collect="on" /> <Measurement name="ReceivedMessageSize_100KB-1MB" collect="on" /> <Measurement name="ReceivedMessageSiz
e_1MB-10MB" collect="on" /> <Measurement name="ReceivedMessageSize_Over10MB" collect="on" /> </ResourceStatsSwitches>'
  maxSocketAge='4'
  maxKeepAliveRequests='-1'
  resetCollectionStatistics=''
  tcpNoDelay='Platform Default'
  tcpNoDelaySSL='Platform Default'
```

**Note:** Platform default will vary by platform!

Values in effect on different platforms will be different even though the component setting is the same

AIX has a default of `tcpNoDelay=false` effective. For Windows it is `tcpNoDelay=true`.

Setting it explicitly makes it clear

# TCP/IP Nodes

- **TCP/IP Client nodes & TCP/IP Server nodes**

- Identical in function for datastream processing but use client and server connections

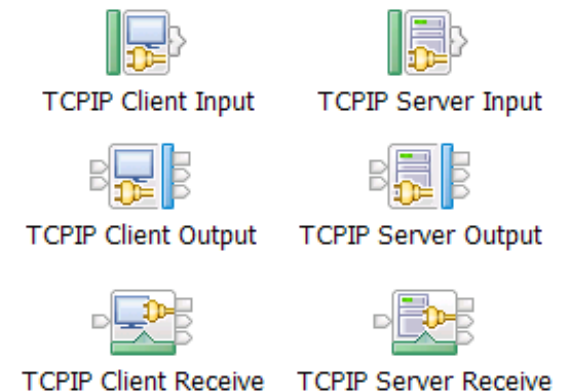
- **Nodes:**

- ▶ TCPIPClientInput and TCPIPServerInput
- ▶ TCPIPClientOutput and TCPIPServerOutput
- ▶ TCPIPClientReceive and TCPIPServerReceive

- **Single 'listener' receives incoming messages on the named port and passes on to required message flow instance**

- **Tuning**

- ▶ Persistent connections via ExpireConnectionSec setting
- ▶ Specify TCP\_NODELAY as true
- ▶ Use smallest number of message flow instances possible



# Displaying TCP/IP Settings for Default Values

```
mqsireportproperties TESTNODE -c TCPIPClient -o AllReportableEntityNames -r
```

```
TCPIPClient
Default
AlternativeAddresses=''
ExpireConnectionSec='-1'
Hostname='localhost'
MaxReceiveRecordBytes='100000000'
MaximumConnections='100'
MinimumConnections='0'
Port='0'
SO_KEEPAALIVE='false'
SO_LINGER='false'
SO_LINGER_TIMEOUT_SEC='-1'
SO_RCVBUF='0'
SO_SNDBUF='0'
SSLCiphers=''
SSLProtocol=''
TCP_NODELAY='false'
TrafficClass='-1'
UseUniqueConnectionPool='false'
```

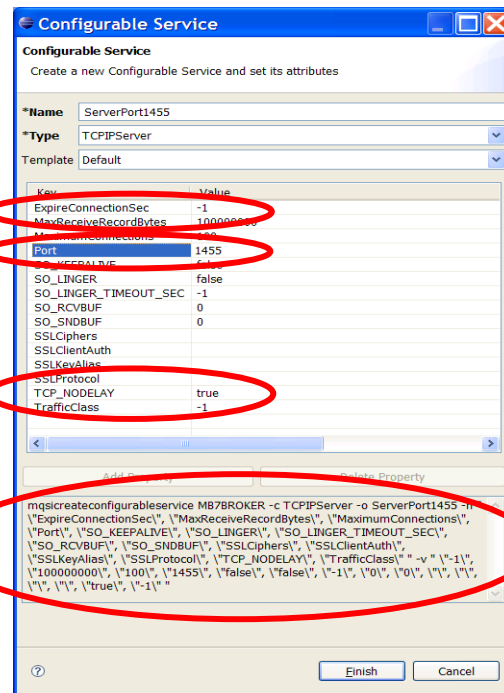
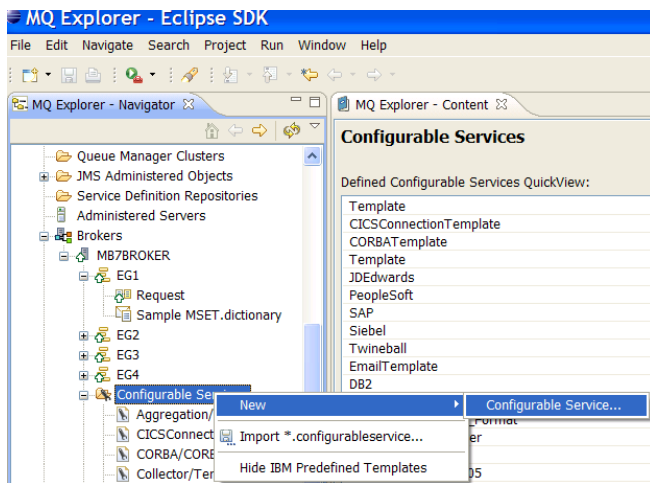
```
mqsireportproperties TESTNODE -c TCPIPServer -o AllReportableEntityNames -r
```

```
TCPIPServer
Default
ExpireConnectionSec='-1'
MaxReceiveRecordBytes='100000000'
MaximumConnections='100'
Port='0'
SO_KEEPAALIVE='false'
SO_LINGER='false'
SO_LINGER_TIMEOUT_SEC='-1'
SO_RCVBUF='0'
SO_SNDBUF='0'
SSLCiphers=''
SSLClientAuth=''
SSLKeyAlias=''
SSLProtocol=''
TCP_NODELAY='false'
TrafficClass='-1'
```



# TCP/IP Tuning

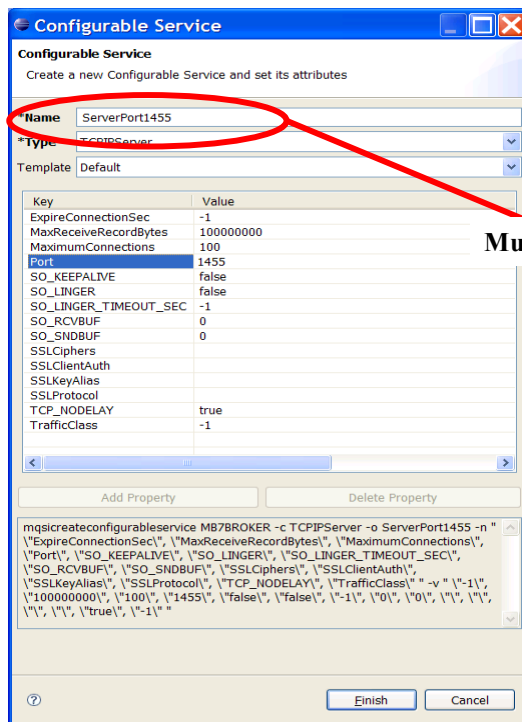
- Tuning for persistent connections is applied by defining a configurable service with required settings. Principle is same for Client & Server



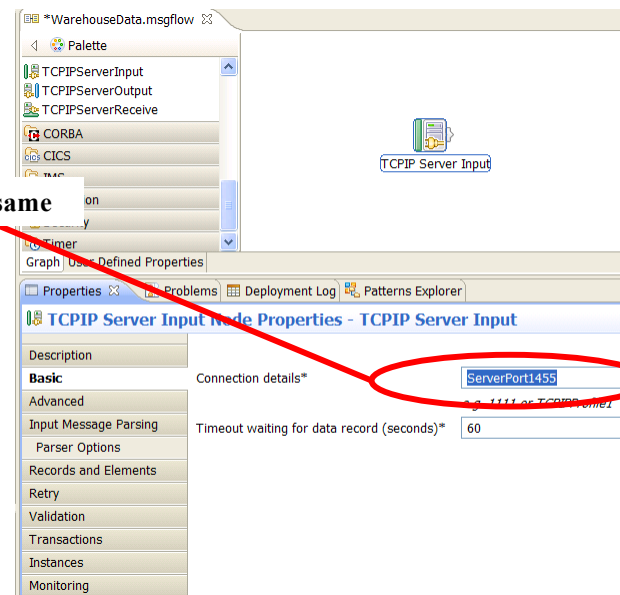
```
mqsicreateconfigurableService MB7BROKER -c TCPIPServer -o ServerPort1455 -n "
\ExpireConnectionSec\", \"MaxReceiveRecordBytes\", \"MaximumConnections\", \"Port\",
\"SO_KEEPALIVE\", \"SO_LINGER\", \"SO_LINGER_TIMEOUT_SEC\", \"SO_RCVBUF\",
\"SO_SNDBUF\", \"SSLCiphers\", \"SSLClientAuth\", \"SSLKeyAlias\", \"SSLProtocol\",
\"TCP_NODELAY\", \"TrafficClass\" -v \"-1\", \"100000000\", \"100\", \"1455\", \"false\",
\"false\", \"-1\", \"0\", \"0\", \"\", \"\", \"\", \"\", \"\", \"\", \"true\", \"-1\" "
```

# TCP/IP Tuning

- Configurable Service name must be given in the Connection details field of the TCP/IP input node.



Must be the same



## Reviewing TCP/IP Configurable Service Properties

Through IBX:

The screenshot shows the MQ Explorer interface. The left pane displays a tree view of various providers, with 'TCP/IP Server/ServerPort1455' selected. The right pane shows the 'Configurable Service ServerPort1455' properties. The 'Properties QuickView' section lists the following attributes:

Name	Value
Name	ServerPort1455
Type	TCP/IP Server
ExpireConnectionSec	-1
MaxReceiveRecordBytes	100000000
MaximumConnections	100
Port	1455
SO_KEEPALIVE	false
SO_LINGER	false
SO_LINGER_TIMEOUT_SEC	-1
SO_RCVBUF	0
SO_SNDBUF	0
SSLCiphers	
SSLClientAuth	
SSLKeyAlias	
SSLProtocol	
TCP_NODELAY	true
TrafficClass	-1

Below the properties, the 'Administration Log' is visible, showing a series of messages from the MB7BROKER source.

With a command:

```
mqsireportproperties TESTNODE -c TCPIPServer -o AllReportableEntityNames -r
```

```
TCPIPServer
Default
  ExpireConnectionSec=' -1 '
  MaxReceiveRecordBytes=' 100000000 '
  MaximumConnections=' 100 '
  Port=' 0 '
  SO_KEEPALIVE=' false '
  SO_LINGER=' false '
  SO_LINGER_TIMEOUT_SEC=' -1 '
  SO_RCVBUF=' 0 '
  SO_SNDBUF=' 0 '
  SSLCiphers=' '
  SSLClientAuth=' '
  SSLKeyAlias=' '
  SSLProtocol=' '
  TCP_NODELAY=' false '
  TrafficClass=' -1 '
ServerPort1455
  ExpireConnectionSec=' -1 '
  MaxReceiveRecordBytes=' 100000000 '
  MaximumConnections=' 100 '
  Port=' 1455 '
  SO_KEEPALIVE=' false '
  SO_LINGER=' false '
  SO_LINGER_TIMEOUT_SEC=' -1 '
  SO_RCVBUF=' 0 '
  SO_SNDBUF=' 0 '
  SSLCiphers=' '
  SSLClientAuth=' '
  SSLKeyAlias=' '
  SSLProtocol=' '
  TCP_NODELAY=' true '
  TrafficClass=' -1 '
```

## Specifying Instances for TCP/IP Nodes

- Instances are specified in the same way as for other input nodes through the Instances tab of the node properties
- TCPIP Client Input uses the same approach

The screenshot displays the IBM MQ Design Center interface. The top window shows a palette of nodes, including TCPIP, TCPIPClientInput, TCPIPClientOutput, TCPIPClientReceive, TCPIPServerInput, TCPIPServerOutput, TCPIPServerReceive, CORBA, and CICS. A 'TCPIP Server Input' node is placed on the graph. Below the graph, the 'User Defined Properties' tab is active, showing the 'TCPIP Server Input Node Properties - TCPIP Server Input' configuration. The 'Instances' tab is selected, showing the following settings:

Property	Value
Additional instances pool	Use Pool Associated with Message Flow
Additional instances	0

# Message Flow Threads

- **A message flow requires a way to execute**

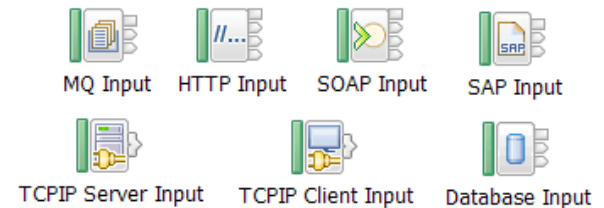
- ▶ On Windows/UNIX/Linux it is a POSIX thread
- ▶ On z/OS it is a POSIX thread which maps to a TCB

- **Threads are specified in several ways**

1. Through the use of input nodes – MQInput, HTTPInput, FileInput etc.
2. Specifying additional threads
  - Instances at the input node level
  - Instances at the message flow level (in the bar file)
  - Instances through use of IB Explorer / Web UI and Integration API (CMP)

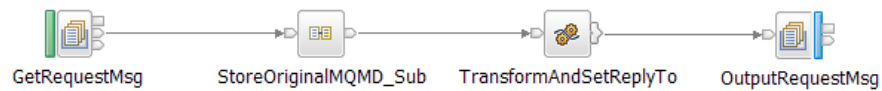
- **Only required number of instances will be used**

- ▶ Number depends on volume of incoming messages and flow execution time

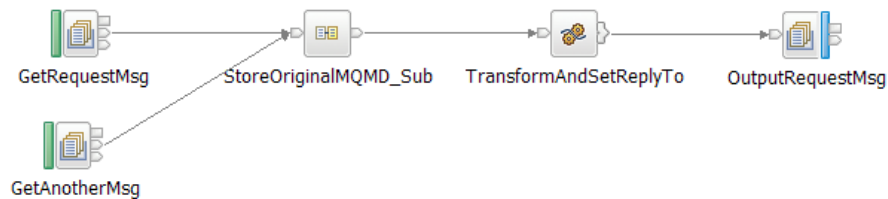


## Message Flow Threads...

- One thread per input node
- This message flow has one thread:



- This message flow has two threads:

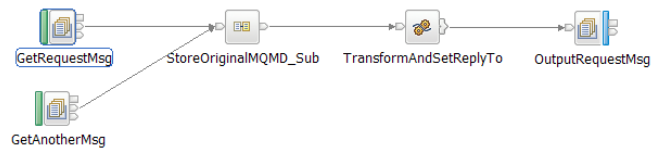


# Specifying More Instances on the Input Nodes

## ■ Most input nodes have an *Instances* tab

▶ Option to specify a node specific thread pool or use the message flow thread pool

- *Use Pool Associated with Node* specifies a **node specific** thread pool – use that pool only
- *Use Pool Associated with Message Flow* means use message flow instances for additional threads



**MQ Input Node Properties - GetRequestMsg**

Description	
Basic	Additional instances pool <input type="text" value="Use Pool Associated with Message Flow"/>
Input Message Parsing	Additional instances <input type="text" value="Use Pool Associated with Message Flow"/>
Parser Options	<input type="text" value="Use Pool Associated with Node"/>
Advanced	
Validation	
Security	
<b>Instances</b>	
Monitoring	

**MQ Input Node Properties - GetRequestMsg**

Description	
Basic	Additional instances pool <input type="text" value="Use Pool Associated with Node"/>
Input Message Parsing	Additional instances <input type="text" value="5"/>
Parser Options	
Advanced	
Validation	
Security	
<b>Instances</b>	
Monitoring	

# Specifying More Instances at Message Flow Level

- **Additional Instances for a message flow can be specified in the bar file**
  - ▶ Select the required message flow in the bar file editor
  - ▶ Specified required number of instances – instances shared across all input nodes in the message flow
  - ▶ Can edit flows in the same bar file to have different number of instances

The image contains two screenshots of the IBM MQ console interface, each showing a different message flow being configured.

**Left Screenshot: Changing Request message flow to have 3 additional instances**

The screenshot shows the 'Manage' view for the 'Request.cmf' message flow. The 'Additional Instances' field is set to 3. The 'Commit Count' is 1 and the 'Commit Interval' is 0.

Name	Type
IN_OUT.cmf	Compiled message flow
Request.cmf	Compiled message flow

**Right Screenshot: Changing IN\_OUT message flow to have 5 additional instances**

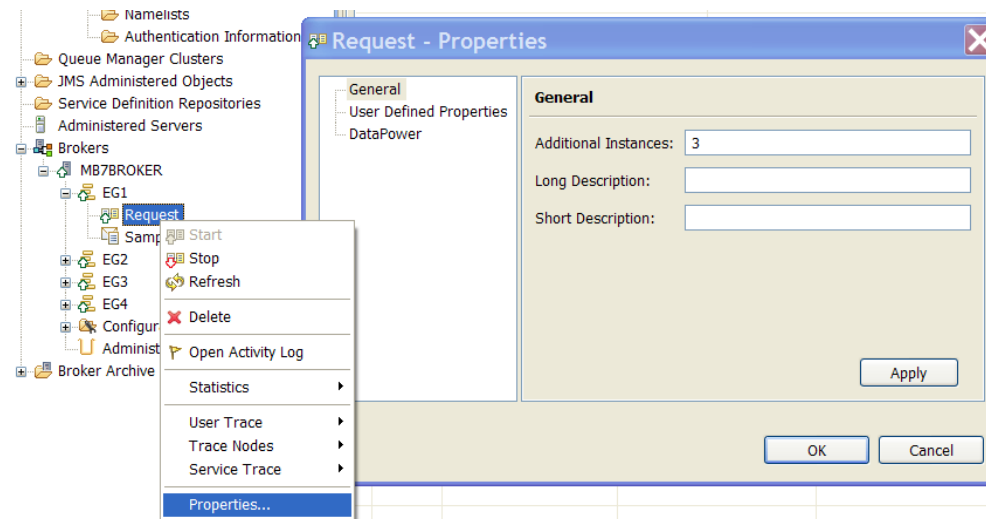
The screenshot shows the 'Manage' view for the 'IN\_OUT.cmf' message flow. The 'Additional Instances' field is set to 5. The 'Commit Count' is 1 and the 'Commit Interval' is 0.

Name	Type	Modified
IN_OUT.cmf	Compiled message flow	18-May-2012 18:00
Request.cmf	Compiled message flow	18-May-2012 18:00



# Specifying More Instances Using IBX

- Integration Bus Explorer (and CMP API) allow the number of instances of a message flow to be changed dynamically in the running node:
  - ▶ Select the required message flow in IBX
  - ▶ Right click, select properties
  - ▶ Change to the required number of additional instances



# Viewing Additional Instances in the Web UI

The screenshot displays the IBM MQ Web UI interface. On the left is a navigation tree under the heading 'Filter Options...'. The tree structure is as follows:

- TESTNODE\_gb043390
  - Servers
    - default
      - Services
      - REST APIs
      - Applications
        - Additional
          - Libraries
          - Message Flows (highlighted)
            - Flow (highlighted)
            - Subflows
            - Resources
            - References
          - Libraries
          - Shared Libraries
          - Message Flows
          - Subflows
          - Resources

The main content area is titled 'Flow - Message Flow' and features four tabs: 'Overview' (selected), 'Statistics', 'Operational Policy', and 'Activity Log'. Below the tabs is a 'Quick View' section containing a table with the following data:

Message Flow Name	Flow
Version	
UUID	e75f04e9-7980-4815-ba61-852eba1b4084
Service Trace Level	none
Commit Count	1
Short Description	
Additional Instances	5
Start Mode	Maintained
Coordinated Transaction	no
Commit Interval	0
Long Description	
Running	true
Run Mode	running

## Attaching a Workload Management policy in the Web UI

The screenshot displays the IBM MQ Web UI interface. On the left, a navigation tree shows the hierarchy: TESTNODE\_gb043390 > Servers > default > Applications > Additional > Message Flows > Flow. The main content area is titled 'Operational Policy' and features a green notification banner at the top stating 'The policy was successfully attached' with a close button. Below the banner are tabs for 'Overview', 'Statistics', 'Operational Policy', and 'Activity Log'. The 'Operational Policy' tab is active, showing a 'Flow Policy' section with an 'Apply' button and a 'Node Policies' section. A table lists available policies for attachment:

Available policies	Attach policy
No attached policy	<input type="radio"/>
AddInstancesPolicy	<input checked="" type="radio"/>

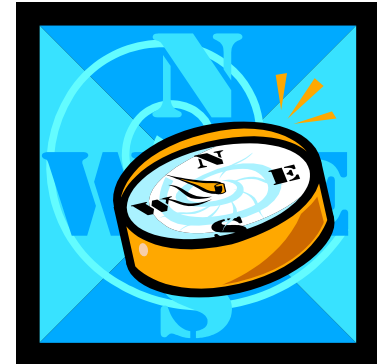
At the bottom of the main area, there is a 'Zoom level' slider.

#MaximizeYourGame

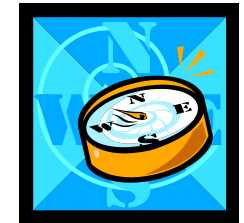
# Transformation Code Optimisation

## Consider your transformation options

- **Many options available!**
  - ▶ ESQL
  - ▶ Java
  - ▶ .Net
  - ▶ Mapping
  - ▶ XSL
  - ▶ IBM Transformation Extender (nee WTX) – A separate product
  
- **Performance characteristics of the different options is rarely an issue**
  - ▶ Select based on skills, ease-of-use & suitability for the scenario
  
- **Note: XSL and ITX are embedded technologies which process an incoming BLOB and produce a different BLOB**
  - ▶ Do not operate on the message tree directly
  - ▶ As such there can be increased costs through increased parsing when mixing with other technologies



# When Coding, Little Things are Important!



## ■ So far looked at:

- ▶ Message flow design
- ▶ Use of different parsers including no parser

## ■ Transformation-specific Optimizations

- ▶ ESQL
- ▶ Java
- ▶ XPath

# ESQL

- Array processing
- Cardinality function
- Create statement
- Declare statement
- EVAL Statement
- FORMAT Clause
- IF and CASE statements
- Case statements
- PASSTHRU
- Propagate
- Reference Variables
- Shared Variables

# ESQL Array Processing

- **Avoid array subscripts [ ] wherever possible as it is expensive**
  - ▶ Because of way in which subscript is evaluated at runtime
    - [1] access first element
    - [5] starts at 1 then 2,3,4,5
    - [10] starts at 1 then 2,3,4,5,6,7,8,9,10
  - ▶ Does this because array subscript has to be evaluated dynamically
    - Could create or delete a member of the the array
  - ▶ Reference variables maintain a pointer into the array then can be reused
    - Point to 10<sup>th</sup> element, can then then move straight to 11<sup>th</sup> – does not start at 1 again

```
DECLARE myref REFERENCE TO InputRoot.XML.Invoice.Purchases.Item[1];

-- Continue processing for each item in the array

WHILE LASTMOVE(myref)=TRUE DO
  -- Add 1 to each item in the array
  SET myref = myref + 1;
  -- Do some processing
  -- Move the dynamic reference to the next item in the array
  MOVE myref NEXTSIBLING;
END WHILE;
```



# ESQL Array Processing Example 1

- Reference Variables work well for InputRoot
  - ▶ Little more work required with OutputRoot but still possible to use

```
DECLARE A1 INTEGER 1;
DECLARE A2 INTEGER CARDINALITY(InputRoot.XMLNSC.TGT_I0089_GLPOSTING_MSGDEF.IDOC.Z1IF_DATA[]);
DECLARE A1_REF REFERENCE TO InputRoot.XMLNSC.TGT_I0089_GLPOSTING_MSGDEF.IDOC.Z1IF_DATA[1];

WHILE A1 <= A2 DO
    MOVE A1_REF TO InputRoot.XMLNSC.TGT_I0089_GLPOSTING_MSGDEF.IDOC.Z1IF_DATA[A1];
    SET
    OutBodyRef.sapzfglaccdoc01:SapZfglAccDoc01.SapZfglAccDoc01IDocBO.SapZfglAccDoc01DataRecord.SapZfglAccDoc01Z2
    ifData000[A1].WapVendor = A1_REF.WAP_VENDOR;

    SET
    OutBodyRef.sapzfglaccdoc01:SapZfglAccDoc01.SapZfglAccDoc01IDocBO.SapZfglAccDoc01DataRecord.SapZfglAccDoc01Z2
    ifData000[A1].WapDivision = A1_REF.WAP_DIVISION;

    SET
    OutBodyRef.sapzfglaccdoc01:SapZfglAccDoc01.SapZfglAccDoc01IDocBO.SapZfglAccDoc01DataRecord.SapZfglAccDoc01Z2
    ifData000[A1].WapStore = A1_REF.WAP_STORE;

    SET
    ...

    SET
    ...

    SET A1 = A1 + 1;
END WHILE;
```

# ESQL Array Processing Example 1...

## Was changed to this:

```
DECLARE A1 INTEGER 1;
DECLARE A2 INTEGER CARDINALITY(InputRoot.XMLNSC.TGT_I0089_GLPOSTING_MSGDEF.IDOC.Z1IF_DATA[]);
DECLARE A1_REF REFERENCE TO InputRoot.XMLNSC.TGT_I0089_GLPOSTING_MSGDEF.IDOC.Z1IF_DATA[1];

WHILE A1 <= A2 DO
    MOVE A1_REF TO InputRoot.XMLNSC.TGT_I0089_GLPOSTING_MSGDEF.IDOC.Z1IF_DATA[A1];
    SET
    OutBodyRef.sapzfglaccdoc01:SapZfglAccDoc01.SapZfglAccDoc01IDocBO.SapZfglAccDoc01DataRecord.SapZfglAccDoc01Z2ifData000[A1].WapVendor =
    A1_REF.WAP_VENDOR;

    DECLARE ArrayRef REFERENCE TO
    OutBodyRef.sapzfglaccdoc01:SapZfglAccDoc01.SapZfglAccDoc01IDocBO.SapZfglAccDoc01DataRecord.SapZfglAccDoc01Z2ifData000[A1];
    SET ArrayRef.WapDivision = A1_REF.WAP_DIVISION;
    SET ArrayRef.WapStore = A1_REF.WAP_STORE;
    SET ArrayRef.WapInvoice = A1_REF.WAP_INVOICE;
    SET ArrayRef.WapBatch = A1_REF.WAP_BATCH;
    SET ArrayRef.WapPo = A1_REF.WAP_PO;
    SET ArrayRef.WapAccount = A1_REF.WAP_ACCOUNT;
    SET ArrayRef.WapInvDate = A1_REF.WAP_INV_DATE;
    SET ArrayRef.WapPostDate = A1_REF.WAP_POST_DATE;
    SET ArrayRef.WapInvCost = A1_REF.WAP_INV_COST;
    SET ArrayRef.WapRetlAmt = A1_REF.WAP_RET_L_AMT;
    SET ArrayRef.WapDept = A1_REF.WAP_DEPT;
    SET ArrayRef.WapOsiTrId = A1_REF.WAP_OSI_TR_ID;
    SET ArrayRef.WapCountry = A1_REF.WAP_COUNTRY;
    SET ArrayRef.WapCurrency = A1_REF.WAP_CURRENCY;
    SET ArrayRef.WapTimeStmp = A1_REF.WAP_TIME_STMP;
    SET A1 = A1 + 1;
END WHILE;
```

- **50% improvement in performance**

- Simply because not accessing

- OutBodyRef.sapzfglaccdoc01:SapZfglAccDoc01.SapZfglAccDoc01IDocBO.SapZfglAccDoc01DataRecord.SapZfglAccDoc01Z2ifData000[A1] every time

## ESQL Array Processing Example 2

- ESQL used to process records from a database read.
  - Process result set of ~200,000 rows to create message for later processing
  - Taking 6-8 hours to run

```
SET Environment.Variables.DBDATA[] =
(
SELECT T.*
FROM Database.{'ABC'}.{'XYZ'} as T
);

DECLARE A INTEGER 1;
DECLARE B INTEGER CARDINALITY(Environment.Variables.*[]);
SET JpCntFODS = B;
WHILE A <= B DO
    CALL CopyMessageHeaders();
    CREATE FIELD OutputRoot.XML.FODS;
    DECLARE outRootRef REFERENCE TO OutputRoot.XML.Data;

    SET outRootRef.Field1 = Trim(Environment.Variables.DBDATA[A].Field1);
    SET outRootRef.Field2 = Trim(Environment.Variables.DBDATA[A].Field2);
    SET outRootRef.Field3 = Trim(Environment.Variables.DBDATA[A].Field3);
    SET outRootRef.Field4 = Trim(Environment.Variables.DBDATA[A].Field4);
    SET outRootRef.Field5 = Trim(Environment.Variables.DBDATA[A].Field5);
    . . .
    SET outRootRef.Field37 = CAST(Environment.Variables.DBDATA[A].Field37)

    SET A = A + 1;
    PROPAGATE;
END WHILE;
```

## ESQL Array Processing Example 2...

- Problem is the repeated use of array subscripts throughout
  - Such as Environment.Variables.DBData[A].

```
SET outRootRef.Field1 = Trim(Environment.Variables.DBData[A].Field1);  
SET outRootRef.Field2 = Trim(Environment.Variables.DBData[A].Field2);  
SET outRootRef.Field3 = Trim(Environment.Variables.DBData[A].Field3);  
SET outRootRef.Field4 = Trim(Environment.Variables.DBData[A].Field4);  
SET outRootRef.Field5 = Trim(Environment.Variables.DBData[A].Field5);
```

- Use REFERENCE variables and the LASTMOVE and the time dropped to minutes.

# ESQL CARDINALITY Function

- **Avoid use of CARDINALITY in a loop**

- ▶ Consider the statement

```
WHILE ( I < CARDINALITY (InputRoot.MRM.A.B.C[]
```

- **CARDINALITY function has to be evaluated each time the loop is traversed**

- ▶ Can be a problem with large arrays where cost of evaluating CARDINALITY is expensive and as the array is large we also loop more often

- **Best to determine the size of the array before the while loop (unless it will change in the loop) so it is only evaluated once**

```
SET ARRAY_SIZE = CARDINALITY (InputRoot.MRM.A.B.C[] WHILE ( I < ARRAY_SIZE )
```

# ESQL DECLARE & EVAL Statements

## ■ DECLARE

- ▶ Reduce number of DECLARE statements (and so cost) by DECLAREing a variable and setting its initial value within a single statement
  - Alternatively, DECLARE multiple variables of the same data type within a single ESQL statement rather than in multiple statements. Also helps to reduce memory usage

## ■ EVAL Statement

- ▶ Used sometimes when there is need to dynamically determine correlation names. It is expensive in CPU use though as it effectively involves double execution of a statement

# ESQL PASSTHRU Statement

## ■ Calling PASSTHRU

- ▶ Avoid the use of the PASSTHRU statement with a CALL statement to invoke a stored procedure. Use the "CREATE PROCEDURE ... EXTERNAL ..." and "CALL ..." commands instead

## ■ Host variables

- ▶ Important to use host variables so dynamic SQL statements can be re-used
- ▶ Host variables map a column value to a variable
- ▶ SQL PREPARE is expensive so want to reuse where possible.

### ▶ Statement:

```
PASSTHRU('UPDATE SHAREPRICES AS SP SET Price = 100 WHERE SP.COMPANY = 'IBM');
```

- Can only be used when Price is 100 and Company is IBM.
- When Price or Company change need another statement, with another PREPARE

### ▶ Re-coding allows Price and Company to change but still use same statement

```
PASSTHRU('UPDATE SHAREPRICES AS SP  
SET Price = ? WHERE SP.COMPANY = ?',  
InputRoot.XMLNSC.Message.Price, InputRoot.XMLNSC.Message.Company);
```

# ESQL PASSTHRU Statement

**Note: To see the level of dynamic statement cache activity with db2 use the commands:**

```
db2 connect to <database name>  
db2 get snapshot for database on <database name>
```

**To see the contents of the dynamic statement cache use the commands:**

```
db2 connect to <database name>  
db2 get snapshot for dynamic SQL on <database name>
```



# ESQL Reference Variables

- Use to refer to long correlation names like `InputRoot.MRM.A.B.C.D.E`
  - ▶ Declare a reference pointer using code like

```
DECLARE refPtr REFERENCE to InputRoot.MRM.A.B.C.D;
```

- ▶ To access element E of the message tree use the correlation name `refPtr.E`.
- Use “REFERENCE” and “MOVE” statements to help reduce the amount of navigation within the message tree
  - ▶ Very useful when constructing large numbers of “SET” or “CREATE” statements.
  - ▶ Instead of navigating to the same branch in the tree you can use a REFERENCE variable to establish a pointer to the branch and then use the MOVE statement to process one field at a time

# ESQL String Functions

- **All string manipulation functions used within ESQL are CPU intensive**
  - ▶ LENGTH, SUBSTRING, RTRIM etc. need to access individual bytes in the message tree which makes them expensive to run
  - ▶ Minimise use of such functions if possible.
  - ▶ If you do need to use them avoid repeatedly executing the same concatenations by storing intermediate results in variables for example.

# Java – Tree References

## ■ Storing Intermediate tree references

- ▶ Avoid building and navigating trees without storing intermediate references. For example:

```
MbMessage newEnv = new MbMessage(env);
newEnv.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME, "Destination", null);
newEnv.getRootElement().getFirstChild().createElementAsFirstChild(MbElement.TYPE_NAME, "MQDestinationList", null);
newEnv.getRootElement().getFirstChild().getFirstChild()
createElementAsFirstChild(MbElement.TYPE_NAME, "DestinationData", null);
```

- ▶ This repeatedly navigates from root to build the tree. It is better to store references as follows:

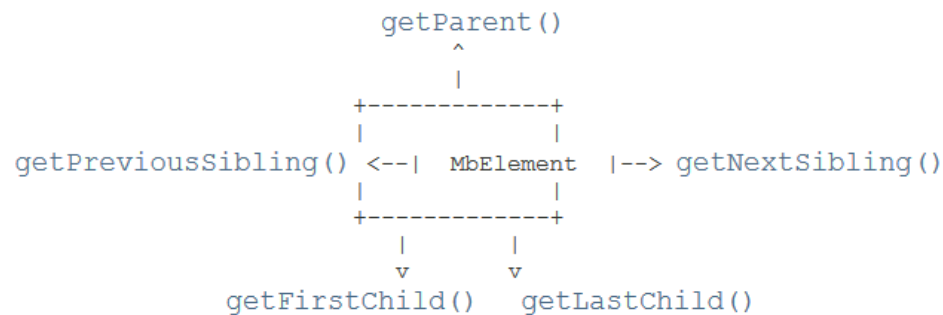
```
MbMessage newEnv = new MbMessage(env);
MbElement destination = newEnv.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME, "Destination", null);
MbElement mqDestinationList = destination.createElementAsFirstChild(MbElement.TYPE_NAME, "MQDestinationList", null);
mqDestinationList.createElementAsFirstChild(MbElement.TYPE_NAME, "DestinationData", null);
```

# Java – Tree References

- **MBElement**

- ▶ `com.ibm.broker.plugin.MbElement`

- **MbElement represents the syntax elements in the logical (hierarchical) view of the message. Methods are provided for navigating and modifying the hierarchy.**



- **Warning: caching MbElement objects over multiple message flow invocations is unsupported because the internal state may be reset at the end of the current message invocation.**

## Example of how to use correctly

```
import com.ibm.broker.javacompute.MbJavaComputeNode;

public class SetLocalEnv extends MbJavaComputeNode {
    private static MbElement ptrEnvironment;

    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage inMessage = inAssembly.getMessage();
        MbMessageAssembly outAssembly = null;

        ptrEnvironment = inAssembly.getLocalEnvironment().getRootElement();
    }
}
```



MbElement is held as private static – **not supported** as state maybe reset by IIB without notification at end of flow processing

```
import com.ibm.broker.javacompute.MbJavaComputeNode;

public class SetLocalEnv extends MbJavaComputeNode {

    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage inMessage = inAssembly.getMessage();
        MbMessageAssembly outAssembly = null;

        MbElement ptrEnvironment = inAssembly.getLocalEnvironment().getRootElement();
    }
}
```



When MbElement is held as local variable only then no issue

# Java String Concatenation

- To concatenate `java.lang.String` objects use `StringBuffer` class and `append` method rather than the `+` operator  
+ operator is expensive since it (internally) involves creating a new `String` object for each concatenation

## Code such as

```
keyforCache = hostSystem + CommonFunctions.separator
               + sourceQueueValue + CommonFunctions.separator
               + smiKey + CommonFunctions.separator
               + newElement;
```

## will perform better written as:

```
StringBuffer keyforCacheBuf = new StringBuffer();
keyforCacheBuf.append(hostSystem);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(sourceQueueValue);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(smiKey);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(newElement);
keyforCache = keyforCacheBuf.toString();
```

# Java BLOB Processing

- Sometimes need to process a BLOB – to cut it into chunks or insert characters for example.
  - ▶ JavaCompute node using Java string processing capabilities may be better than using ESQL with its string manipulation facilities such as SUBSTRING.
- If using the JavaCompute node use *ByteArrays* and *ByteArrayOutputStream* to process the BLOB.

# Java Process Forking

- **Runtime.getRuntime().exec is used to fork a process**

- ▶ That is what you asked for but...the memory requested for the new process will be the same as the existing process, the one issuing the exec method call – the Execution Group (DataFlowEngine).
- ▶ If the execution group has many message flows deployed to it and has been processing lots of large messages it could be 100's MB or GB in size. The exec call will double this amount of memory being used
  - Could lead to surge in demand for real memory dependent on the size of the execution group and frequency of fork requests and introduce performance problems that threaten the whole machine
- ▶ **Two alternative approaches:**
  - Put a message to a queue so another message flow or application program can read & process it
  - Assign the flow that issues the exec call its own Execution Group so that the process size is small and the memory impact of the fork is far less.



# Java Manual GC

- **Avoid manually invoking garbage collection**
  - ▶ Such as by using the call `Runtime.getRuntime().gc()`.
- **Garbage collection causes a cessation of processing in the JVM whilst it is taking place and so it an inhibitor to message throughput.**
- **As example:**
  - ▶ Java based transformation message flow ran at 1300 msg/sec without manual GC. Adding one manual GC call per message being processed took the rate down to 100 msg/sec!

# XPath Optimisations

- **Where possible number instances of an element required explicitly**
  - ▶ `/element[1]` - with find the first occurrence and stop
  - ▶ `/element[2]` - will find the second occurrence and stop
  - ▶ `/element` – will return a node set of all instances in the message so driving a full parse of the message – even if you do not want that

```
private final MbXPath setCustomer = new MbXPath(  
    "$Statement[?@Type[set-value('Monthly')]]" +  
    "[?@Style[set-value('Full')]]" +  
    "[?Customer[?Initials[set-value(concat($invoice/Initial[1], $invoice/Initial[2])]]]" +  
    "[?Name[set-value($invoice/Surname)]]" +  
    "[?Balance[set-value($invoice/Balance)]]" +  
    "/?Purchases");
```

See <http://www.xml.com/lpt/a/1018> for Top Ten Tips for XPath

# XSLT

- **Attractive to use in a message flow**

- ▶ Good potential for code re-use, especially for XSLT developed on other projects
- ▶ Encapsulated logic
- ▶ Caching is available within IIB (separate to the global cache!)
  - Only applies to a loaded stylesheet though and not an in-line stylesheet

- **However**

- ▶ Unable to interact with the Message Tree as with ESQL and Java
- ▶ When used mid-stream with other IIB technologies, it will result in a higher level of message parsing and serialisation
- ▶ Not able to share intermediate data with other technologies
  - Temporary data held in the *Environment* correlation for example

# XSLT Optimizations

## ■ Templates Object

- ▶ Use a Templates object (with a different Transformers for each transformation) to perform multiple transformations with the same set of stylesheet instructions (see [Multithreading](#))

## ■ Set up your stylesheets to function efficiently.

- ▶ Don't use "/" (descendant axes) patterns near the root of a large document.
- ▶ Use `xsl:key` elements and the `key()` function as an efficient way to retrieve node sets.
- ▶ Where possible, use pattern matching rather than `xsl:if` or `xsl:when` statements.
- ▶ `xsl:for-each` is fast because it does not require pattern matching.
- ▶ Keep in mind that `xsl:sort` prevents incremental processing.
- ▶ When creating variables then  
`<xsl:variable name="fooElem" select="foo"/>`  
is usually faster than  
`<xsl:variable name="fooElem"><xsl:value-of select="foo"/></xsl:variable>`
- ▶ Use the `last()` function wisely
- ▶ Use of index predicates within match patterns can be expensive
- ▶ Decoding and encoding are expensive

#MaximizeYourGame

## In Summary...

## Summary

- **Multiple areas in which recommendations can be applied**
  - ▶ Design
  - ▶ Coding
  - ▶ Deployment
- **Version upgrades include numerous performance enhancements**
  - ▶ Many of which are picked up simply by upgrading
- **Most resilient and best performing systems are those that are:**
  - ▶ Loosely coupled and have parallel execution (threads and processes)
  - ▶ Implications for your message flow design will depend on the systems that IIB interacts with
- **Vital to do performance testing:**
  - ▶ Before production
  - ▶ In production like environment
  - ▶ Allows time to evaluate and refactor code/config if needed

## Questions & Answers

