M72

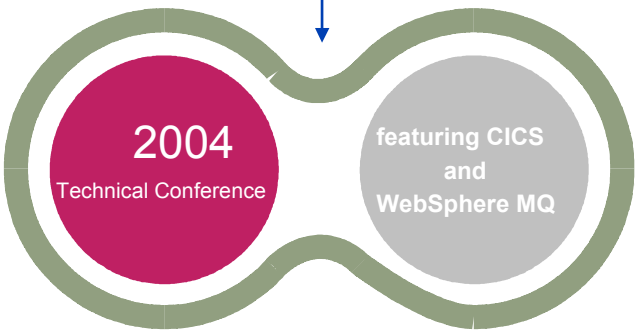
WBI Brokers: Designing for Performance

Tim Dunn
dunnt@uk.ibm.com

Transaction & Messaging Technical Conference

June 14-18 2004

Las Vegas Hilton, Las Vegas, NV



Notes

The purpose of this presentation is to highlight the major design issues which should be addressed when implementing a message processing solution using WebSphere Business Integration Brokers

The level of performance which is obtained with a message flow is a function of the efficiency of the message flow and the environment in which the message flows runs. Well performing systems rarely happen by accident and must be planned. This presentation discusses those aspects of message flow design and broker configuration which you must focus on in order to produce an implementation which performs well.

The issues raised in this presentation are cross platform and not dependent on any particular platform. The function discussed in the presentation is available on all of the platforms on which brokers are implemented.

The presentation is focused on the use of WebSphere Business Integration Message Brokers (Message Broker) with WebSphere MQ messages. The principles covered will apply equally to other message formats.

Trademarks and Service marks

The following terms are trademarks of the IBM Corporation
in the United States or other countries or both:

AIX, DB2, IBM, MQSeries, MQSeries Integrator, MQSI, WebSphere MQ Integrator, WebSphere Business Integration Brokers, WMQI, Netfinity, z/OS.

The following terms are trademarks of other companies:

Windows NT, Windows 2000, NEONFormatter, NEONRules, NEON, Solaris

Other company, product and service names may be trademarks or service marks of other Companies.

What you will get out of this presentation

- An understanding of the major factors which affect message processing costs.
- Awareness of those issues which you need to address when designing message flows.
- An understanding of why you need to consider the whole environment.

Notes

What can you hope to get out of this presentation ?

You will hopefully understand the component costs of processing messages in a WBI Message Broker message flow and understand how to minimize the costs of processing in order to produce more efficient message flows. The benefit of doing is that you can use less resource to achieve a given message rate or achieve a higher message rate with a given level of resource.

It is possible to process messages with Message Broker in a variety of ways by structuring data or message flows differently. You could for example have a single message flow to process all messages. This would have different processing characteristics from a flow which processed only one of the message formats. In order to produce message flows which perform well it is important to invest time and effort thinking about how you will address issues such as data format and structure, message flow properties and error processing. In this presentation we are going to look at the issues which you need to address and discuss recommendations.

In implementing a brokering solution it is important to consider all aspects of the system. Without sufficient CPU even the most efficient of message flows will not be able to process messages at the required rate.

Contents

Notes

The presentation starts with a few throughts on what message throughput means. We can talk about a particular message rate but what does it actually mean ? It is important to establish expectations before commencing work on a message flow.

With an understanding of the costs of processing a message in a message flow you are better able to understand the impact of making various design decisions. To help with this we will look at the major components of the Message Broker, see how message flows are implemented on each platform and then identify the different component costs which occur when a message is processed. By understanding where these costs arise it is possible to make decision decisions which help reduce the costs.

In the remaining sections we will look at the those aspects of message, message flow design and Message Broker configuration which have the most significant effect on overall message throughput. Decisions made in each of these areas will affect final performance.

Finally there will be a summary.

Message Throughput

- What does it mean ?
- What is being counted ?
- Messages/second or seconds/message ?
- What are the published capabilities ?
- Overall performance is the sum of the parts

Notes

A system might be capable of processing 5 messages per second or even 500 per second but what does it mean in practice. At a simplistic level it can give us some idea of the volume of work which the system is processing. 500 per second sounds much better than 5 per second but the numbers by themselves mean little since we have no idea of the complexity of the work being performed.

Message throughput is affected by a number of factors. These include the complexity of the message. How efficient the message flow is. Is it well designed and coded for instance.

What type of environment is the message flow running in. Are there all the required resources available or is the processing CPU constrained for example. If the same message flow is run on a different hardware configuration it is likely that we will get different results, so environment is important.

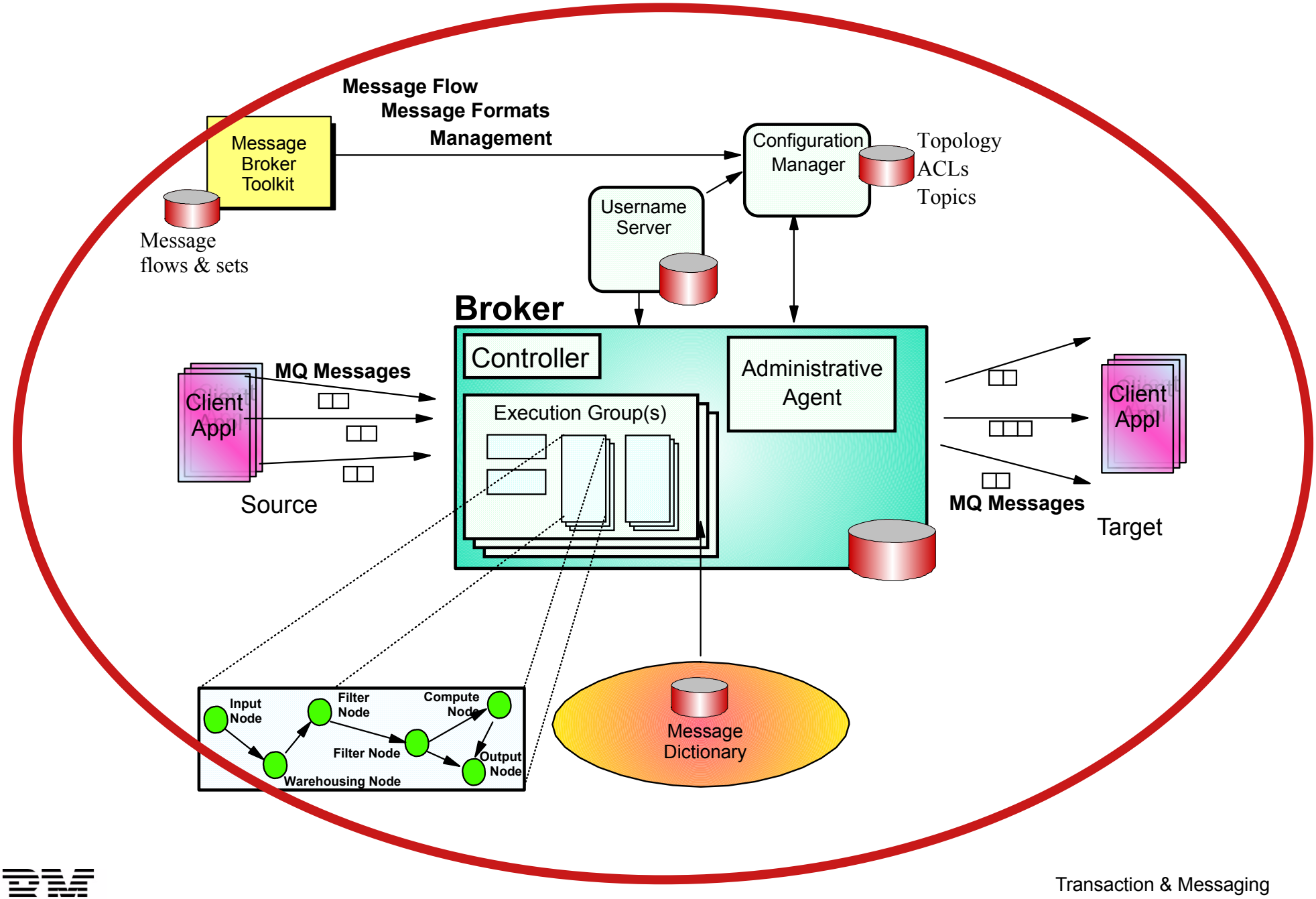
The format, size and complexity of messages will have a noticeable effect on the processing requirements of a message flow and so affect the message rate which can be obtained with a given processor.

Message rate is not a standard measure. It will depend on what is being counted. It could be the number of WebSphere MQ messages per second or the number of business transactions per second. A business transaction usually consists of more than one WebSphere MQ message so we would expect the business transaction rate to be the lower number. But because it is a higher level entity it does not easily represent the true level of work which is taking place.

Dependent on the complexity of the messages and the processing which is to be performed on them we might well be looking at a rate which is expressed as a number of messages per second. This could be tens, hundreds or even thousands of message per second. Equally with the large, multi-megabyte messages we could be looking at multiple seconds to process one message.

It is important to recognize that we should look at all aspects of message flow processing. Not only the logic of the flow but the environment in which it runs in order to develop an implementation which performs well and meets demands.

Major Components



Notes

This chart shows the major components of WebSphere Business Integration Brokers.

From a message throughput point of view it is the broker that we focus on.

Message flows are sent from the Configuration manager to the broker once a deploy of the flow is requested. The deployed message is received by the Administrative Agent which then parses the message from the Configuration Manager and starts the required execution groups. Once the execution groups are started the required message flows can also be started.

We also have the Controller, which is responsible for ensuring that all of the required execution groups are running. In the event of failure the Controller will request that a failed execution group be restarted. This helps with ensuring higher broker availability.

It is the message flows running in the execution groups which process messages. The message flow reads messages waiting to be processed using an input node. The message is then processed according to the rules specified in the message flow.

From a message throughput point of view we have to focus on the content of the message flow and the resources (CPU, memory, I/O) which are available to run it.

The Message Broker Toolkit , Configuration Manager and Username Server are all important components but they do not play a direct role in the processing of messages so we will set them aside in this presentation.

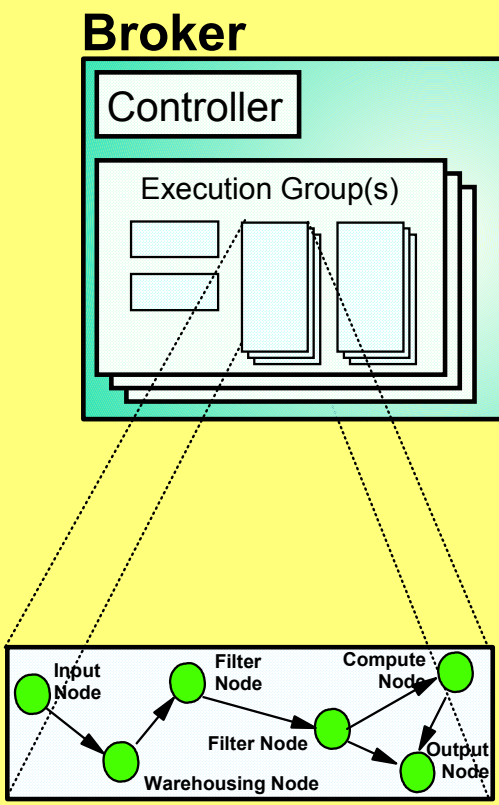
All of the components within the red circle constitute what is known as a broker domain.

A broker domain is essentially all the components connected to a Configuration Manger.

In a broker domain it is possible to have multiple brokers, running on one or more machines and over one or more of the supported platforms.

Through this ability to run brokers on more than one machine we have the ability to horizontally scale message throughput.

Dataflow Engine Implementation



Process/Address Spaces

Administrative Agent
Controller
Execution Groups

Thread/TCB

Message Flows

✓ Threading model provided
✓ Great potential for multiprocessor exploitation

Notes

Within a broker it is possible to have one or more execution groups.

Execution group is a broker term and is a container in which one or more message flows can run.

The implementation of an execution group varies with platform. On Windows and UNIX platforms it is implemented as an operating system process. On z/OS it is implemented as an address space.

The execution group provides separation at the operating system level. If you have message flows which you must separate for some reason, you can achieve this by assigning them to different execution groups.

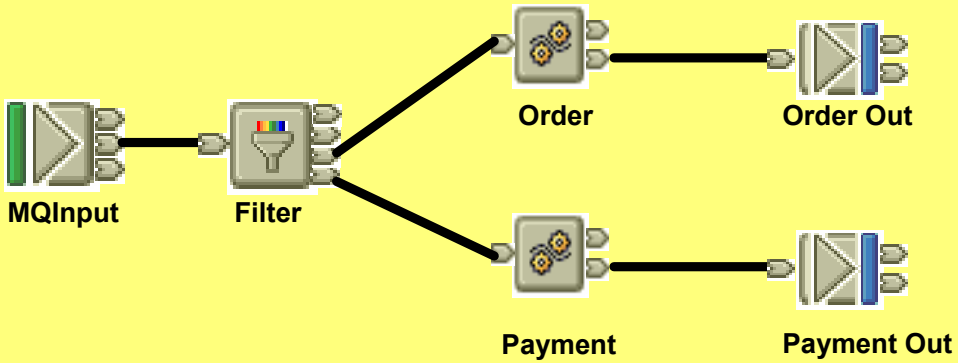
An individual message flow runs as an operating system thread on the Windows and UNIX platforms and as a Task Control Block (TCB) on z/OS.

It is possible to run more than one copy of a message flow in an execution group, in which case there will be multiple threads or TCBs running the same message flow. Equally you can run a message flow in more than one execution group. In which case there will be one or more threads or TCBs running the message flow in each of the processes or address spaces to which the message flow has been assigned.

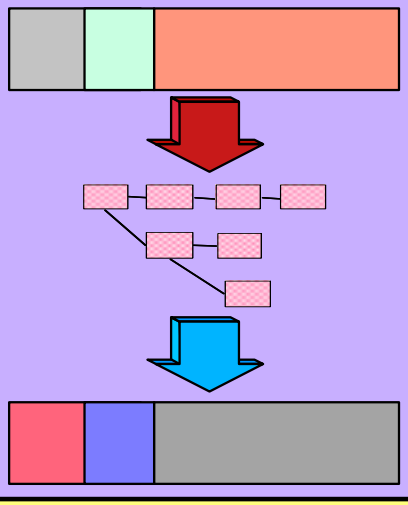
A significant benefit from WebSphere Business Integration Brokers is that a threading model is provided as standard. The message flow developer does not need to explicitly provide code in order to cope with the fact that multiple copies of the message flow might be run. How many copies and where they are run is an operational issue and not a development one. This provides significant flexibility.

The ability to use multiple threads or TCBs and also to replicate this over multiple processes or address spaces means that there is excellent potential to use and exploit multiprocessor machines.

Message Flow Processing Constituent Costs

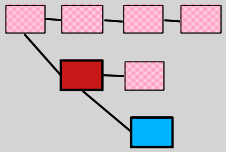


Parsing

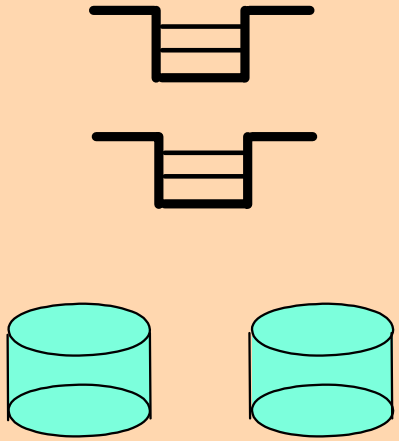


Navigation

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



Resources

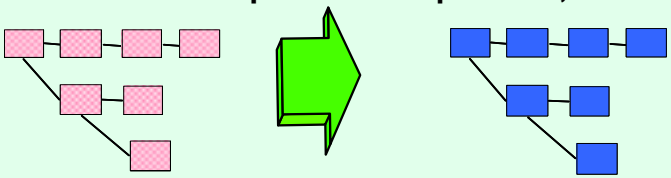


ESQL

```
SET Environment.Hold = THE (SELECT T.QMGR, T.QNAME FROM
Database.SMX.DESTLOOKUP AS T
WHERE T.MDASIGN = InputRoot.MRM."REC_INPUT"."MDALDEST"."MDASIGN"
    AND T.MDADEST =  InputRoot.MRM."REC_INPUT"."MDALDEST"."MDADEST");
```

Tree Copying

```
Set OutputRoot = InputRoot;
```



Notes

This foils shows a simple message flow. When this or any other message flow processes messages costs arise.

These costs are:

- Parsing. This has two parts. The processing of incoming messages and the creation of output messages. Before an incoming message can be processed by processing nodes or ESQL it must transformed from the sequence of bytes which is the input message into a structured object, which is the message tree. Some parsing will take place immediately such as the parsing of the MQMD, some will take place, on demand, as fields in the message payload are referred to within the message flow. The amount of data which needs to be parsed is dependent on the organization of the message and the requirements of the message flow. Not all message flows may require access to all data in a message. When an output message is created the message tree needs to be converted into an actual message. This is a function of the parser. The process of creating the output message is referred to as serialization or flattening the message tree. The creation of the output message is a simpler process than reading than an incoming message. The whole message will be written at once when an output message is created. We will discuss the costs of parsing in more detail later in the presentation.
- ESQL. It is ESQL which allows us to implement the message and business processing logic within the message flows. The cost of running this is dependent on the amount and complexity of the ESQL coded.
- Navigation. This is the process of "walking" the message tree to access the elements which are referred to in the ESQL. The cost of navigation is dependent on the complexity and size of the message tree which is dependent on the size and complexity of the input messages and the complexity of the processing within the message flow. Reference pointers reduce the cost of accessing elements of the message tree.
- Tree Copying. This occurs in nodes which are able to change the message tree such as Compute nodes. A copy of the message tree is taken for recovery reasons so that if a compute node makes changes and processing in node incurs or generates an exception the message tree can be recovered to a point earlier in the message flow. Without this a failure in the message flow downstream could have implications for a different path in the message flow. The tree copy is a copy of a structured object and so is relatively expensive. It is not a copy of a sequence of bytes. For this reason it is best to minimize the number of such copies, hence the general recommendation to minimize the number of compute nodes in a message flow. Tree copying does not take place in a Filter node for example since ESQL only references the data in the message tree and does not update it.
- Resources. This is the cost of performing WebSphere MQ and database requests. The extent of these costs is dependent on the number and type of the requests.

Knowing how processing costs are encountered puts you in a better position to make design decisions which minimize those costs. This is what we will cover during the course of the presentation.

Application Boundaries

- Message broker or an application server ?
- What goes where ?
- Whatever the split, have distinct boundaries
  - Avoid 'function calls' to external applications
- Cost of "function call" is > MQPUT and MQGET processing
  - Extra parsing cost
  - Loss of context information

Notes

A key design decision is deciding how much business logic to implement within your message flows. Given the capabilities of Message Broker there is the potential to implement large amounts of processing within a message flow using ESQL. Implementations using Message Broker vary significantly from simple routing of messages to complex validation and transformation. Some implementations also read data from a database and use that to populate messages.

Message Broker is at says on the can a message broker and not an application server so do not implement overly complex processing within it.

In the complex environments which many of you have it is sometimes difficult to draw functional boundaries. Should an existing application be replaced by logic which is implemented within a message flow or should use of brokers be restricted to connecting and or transforming the messages between communicating applications. This is a difficult issue to resolve. What is clear is that it is the more complex implementations which more often experience performance problems.

However much function you do decide to implement in a message flow it is important to have clear boundaries between pieces of application processing.

Where companies have tried to mix logic between brokers and an existing application by retaining some logic in legacy applications poor results have been obtained. That is low throughput and high resource utilization. By trying to mix processing in this way there is the danger of having a series of very expensive function calls. Each call consists of an MQPUT by the message flow, an MQGET by the application, some application processing, an MQPUT by the application and finally an MQGET by another message flow in order to continue processing within the broker. The function calls has resulted in 4 MQPUT and MQGET requests. Each message has also had to be parsed, since context was lost.

In some situations it will be necessary to retain legacy applications. If a message flow must interact with such an application give thought as to how the two will communicate. Keep such communications to a minimum.

Messages

- Messages are at the core of implementation
- Type, format and size have a significant effect
  - Persistent, non persistent
  - Generic XML, MRM XML, CWF, TDS
  - Bytes or Megabytes
- Well defined structure is not only useful but important
  - MRM versus BLOB
- Message design
  - Volume of data
    - XML is verbose
    - XML msgs larger than CWF but more obvious
  - Field size and order are important

Notes

Messages are at the core of the processing which you are implementing. The type, size and complexity of the messages have a significant effect on the amount of processing which is required to process the messages. It is important therefore that the messages are themselves efficient and are structured in a way to assist processing rather than hinder it. Having said that it is not always possible to specify the format of messages and in many situations the format of the input and output messages are already specified.

The published performance reports illustrate the significant differences in message rate which are achieved with non persistent and persistent messages. The same is also true with messages which are in Generic XML, MRM XML, Custom Wire Format(CWF) or Tagged Delimited String(TDS) format. So type and format clearly have an effect. So aim for those combinations which can be processed at the highest message rates, such as non persistent Generic XML messages.

Messages which have a well defined format can be processed with the Message Repository Manager (MRM). This reduces the complexity of the processing which you have to provide. You do not have to process messages as a series of bytes in a BLOB for instance.

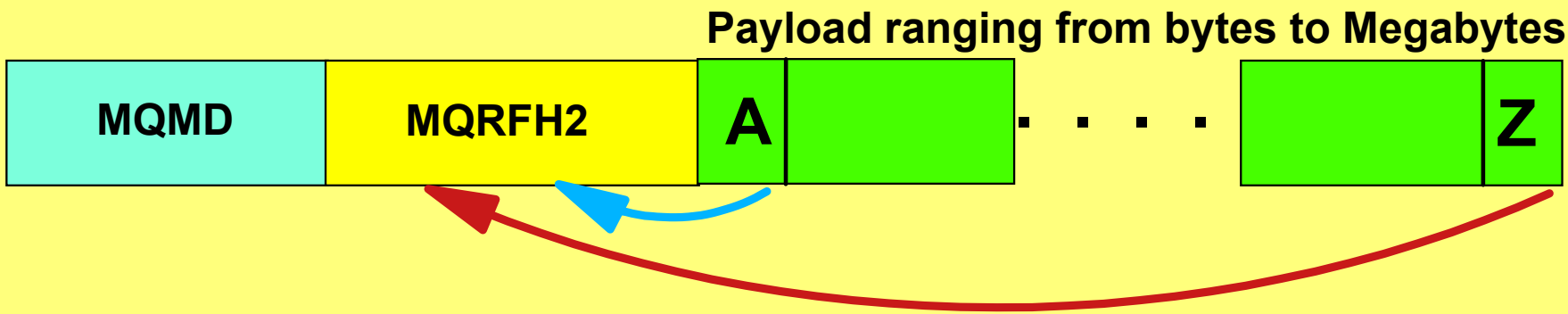
Where you do have influence over the content and structure of messages it is important to consider the format which is used. XML messages are becoming increasingly popular. This does result in some challenges though.

In general, XML messages are much larger than the equivalent message implemented as CWF. It is easy for the tag names to occupy considerably more space than the data which they are describing. As a result the size of messages increases significantly which results in more data to be parsed and processed.

It is sensible to examine the order of fields in a message. When designing a new message consider which fields might be used to route messages and place them at the beginning of the message so that you can benefit from the use of partial parsing.

Messages

- How is data to be referenced
  - Bytes in a BLOB or field names following a parse
- Which data do you need
  - Combine unused fields where possible
- Consider promoting/copying data to MQMD or MQRFH2



- Consider sending only changed data
  - Requires more application logic
  - Could reduce volume of data to transport and parse

Notes

Give thought as to how fields in a message will be accessed. How will the message be parsed, which parser will you use. Do you know the requirements and expectations of processing cost. Processing cost is proportional to the size of the message. It is important to have an understanding of how much data you need to be parsed in order to implement the required business logic.

For a message routing message flow you hopefully only need to access a few fields of data. You do not want to have a to parse the contents of a large message in order to get a single data value. This will significantly increase the cost of processing the message and so reduce the maximum message throughput which you can achieve.

Consider placing some of the message in one of the WebSphere MQ message headers. You may want to copy the data there rather than using that as the location for a particular data value. There are fields in the MQMD which can be used for this purpose. For example the AppIdentityData or CorrelID fields of the MQSeries Message Descriptor (MQMD). Another possibility is to hold a subset of the information in an MQSeries Rules and Formatting Header 2 (MQRFH2). Either of these approaches may save having to parse the body of a large message. The extent of the gain by doing this will depend on the placement of the field within the body of the message and the size of the message. Gains will be better for fields which would otherwise be at the back of large messages.

With message transformation it is likely that you will need to access many more of the fields in a message. In such circumstances field order may not be important.

If your applications are able to cope consider sending only changed data and avoid having to re-send the whole of the message. This is more of an issue with very large XML messages. In some situations messages which are megabytes in size are resent every day, even though most of the contents may not have changed. This significantly increases the load on the whole system.

Messaging Models

- Three models:
  - Single message
  - Coordinated Request/Reply
  - Publish/Subscribe
- Selected model significantly affects:
  - Processing requirements
  - Message flow complexity
  - Message throughput

Notes

The messaging model which you implement in a message flow can have a significant effect on the rate at which messages can be processed.

The three implementations I want to discuss are single message, coordinated request/reply and publish/subscribe.

With single message messages are routed/transformed on an individual basis. Each message is processed and does not have to be correlated with another message.

With coordinated request/reply a request message must be coordinated with one or more associated replies. A common example of this type of processing is where a holiday is to be booked and requests must be sent to book the flight, hotel and car. All the responses must be obtained before a consolidated reply can be produced.

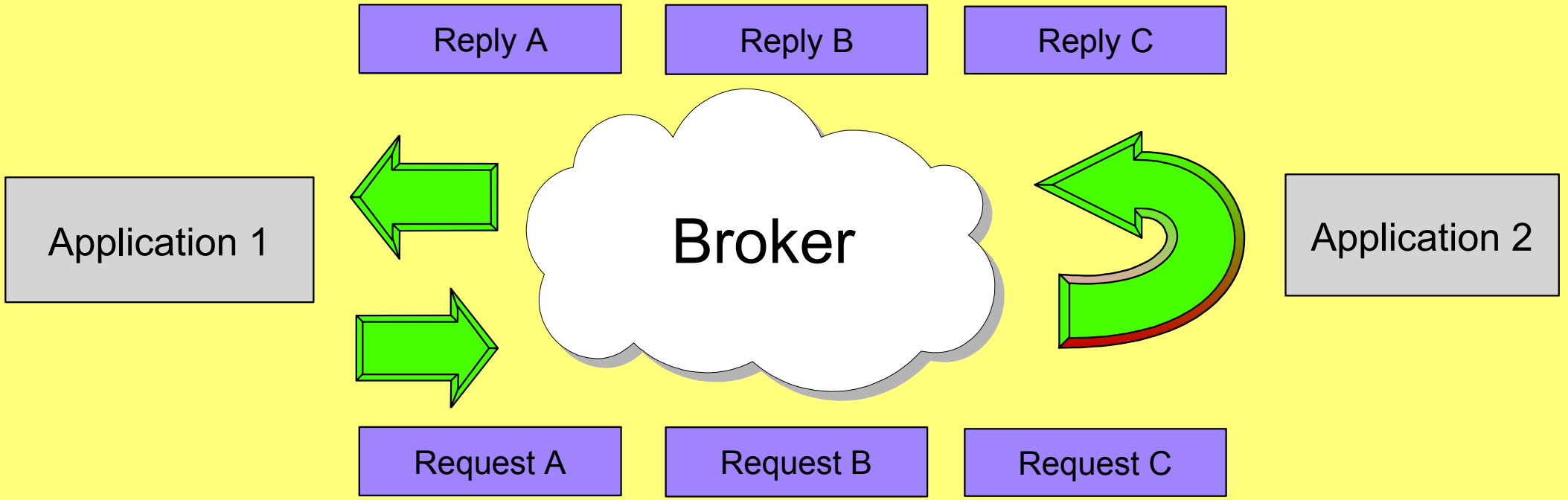
Publish/Subscribe is a processing model in which messages produced by publishers are distributed to registered subscribers. A major benefit of this processing model is that publishers and subscribers do not have to communicate directly. Publishers and/or Subscribers can be transient.

Each of these models is different, uses different functions within the broker and will apply to different situations. Each of the models also has different processing requirements and will have different throughput characteristics.

It may be appropriate to use one or more of these models in your message processing. The use of message routing, transformation, archival etc. are implemented on top of these models.

Single Message

- Process and pass on



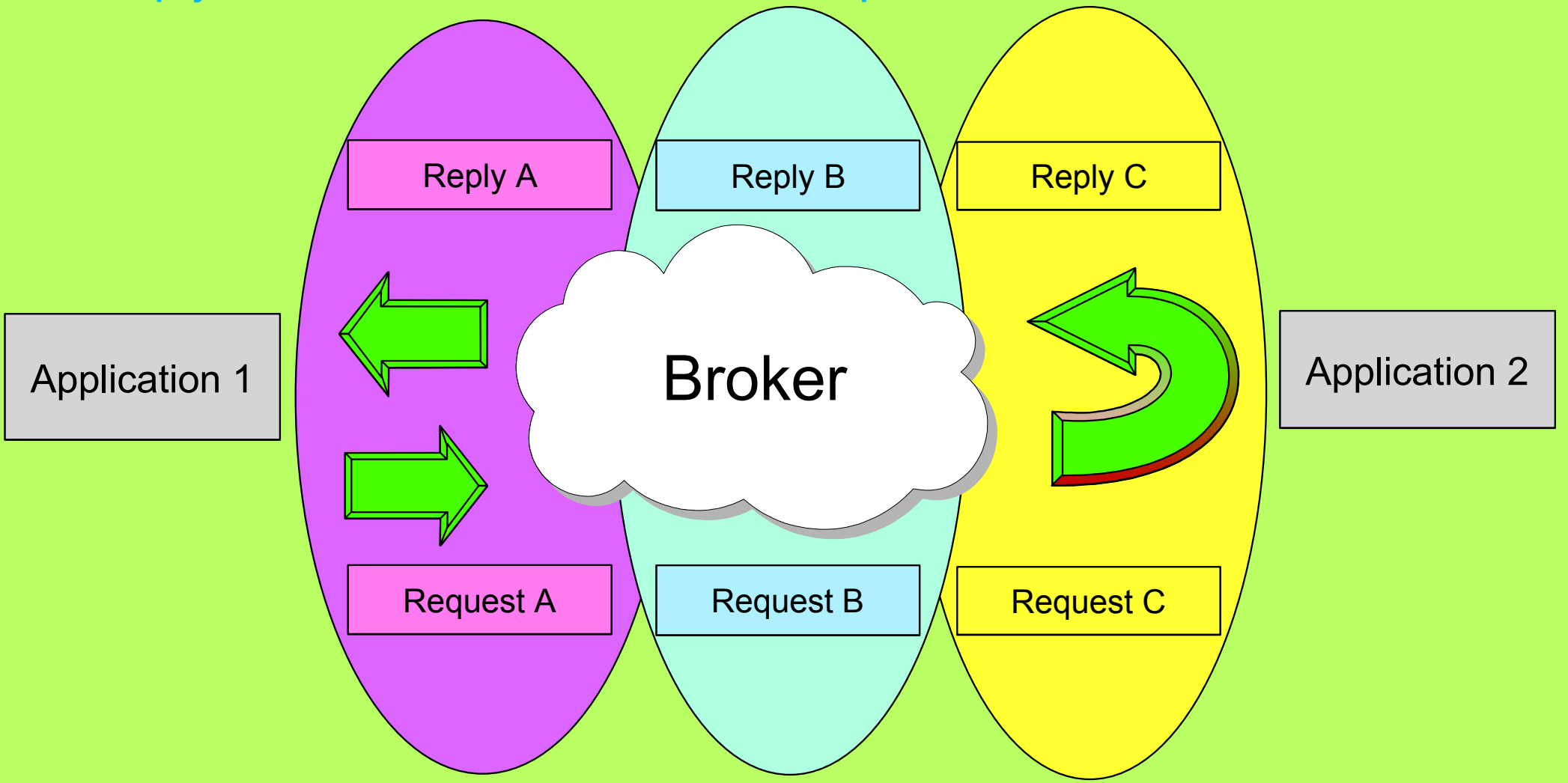
Notes

The Single message model supports the processing of one-off messages or request/reply processing. In this model the request and reply messages are not coordinated and are processed independently.

Message flows are written to perform the necessary processing to format the request and reply messages so that Application 1 and Application 2 are able to communicate. The only logic required is that for the message routing and/or transformation.

Coordinated Request/Reply

- Process, record and pass on request
- Process, retrieve response and produce final response
- Reply has to be coordinated with request



Coordinated Request/Reply

- Coordination of request and reply messages
  - Could be pairing or consolidation
- Different mechanisms for achieving the coordination
  - Use store to hold message id and possibly original message
  - Aggregation nodes introduced in WMQI 2.1 (for tuning advice see SupportPac IP05)
- Approaches have different characteristics and will suit different situations

Notes

With coordinated request/reply it is required to coordinate the request and reply messages.

This type of processing can be used in two ways:

1. It can be used to ensure that each request received a reply.

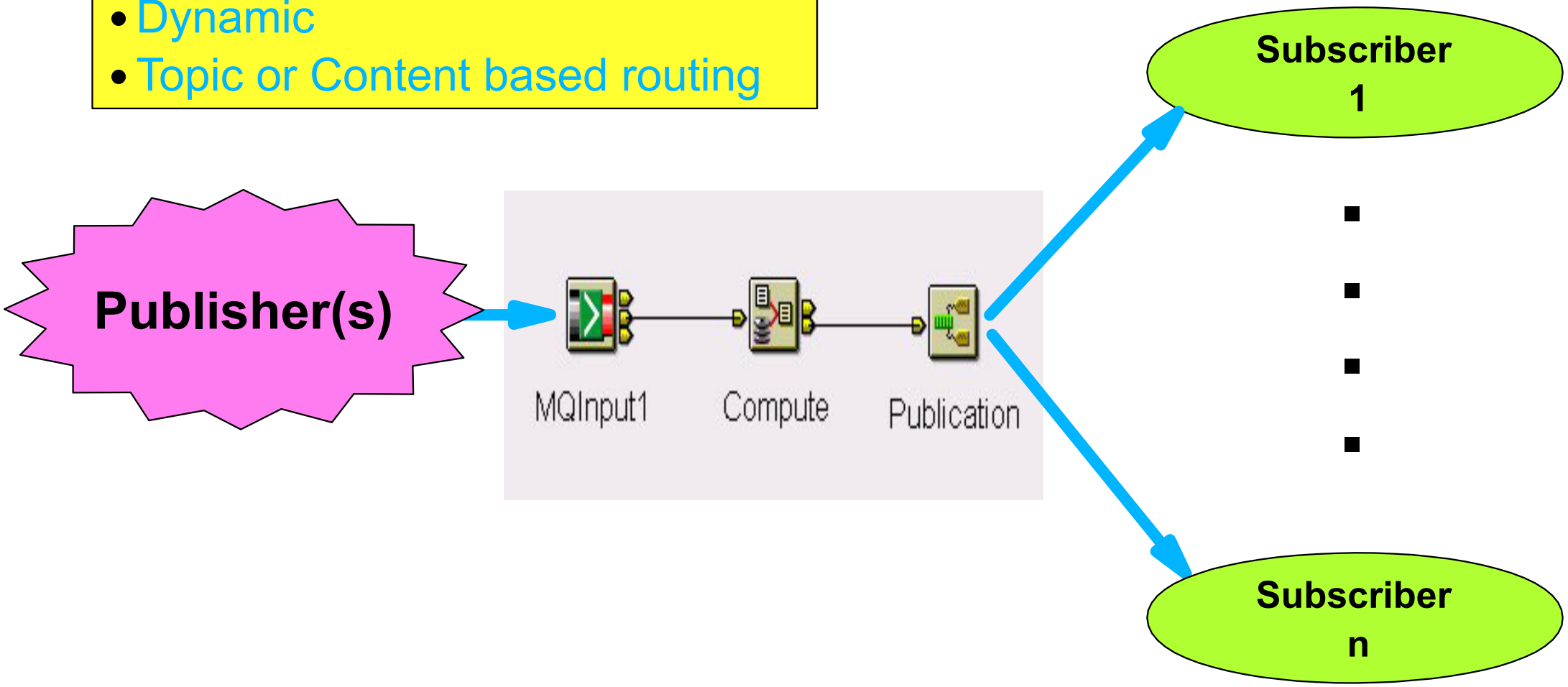
2. To coordinate replies from multiple applications and collect them together so that a final reply message can be generated with information from each of the messages. In our travel booking example we want responses from all of the requests which we sent for the flight, hotel and car. We will then present a consolidated reply to the lucky traveler.

The processing required to achieve the coordination of request and reply can be achieved by writing your own logic to do it and by using a store such as a database to store the fact that the request message was sent. Alternatively you can use the Aggregation nodes which are supplied with Message Broker.

Use of this type of processing can significantly add to the complexity of the message flows. In this model the initial request can result in multiple internal messages which will significantly increase the amount of processing within the system.

Publish/Subscribe Model

- Easy to implement
- Flexible
- Dynamic
- Topic or Content based routing



Notes

The publish/subscribe application model provides a very flexible and dynamic means of distributing information to subscribers.

The publisher is not aware of which applications have subscribed to receive publications. The distribution of the published information is the responsibility of the matching engine.

In WebSphere Business Integration Brokers the distribution of messages is invoked by the publisher using the publication node in a message flow. Publish/Subscribe processing can be combined with other processing in a message flow.

Subscribers are able to come and go as they require. When they no longer require information on a topic they unsubscribe. Unlike other message flows there is no specific processing in the message flow to specify which queues output messages will be written to. This means that new subscribers can be easily added without the need to change the message flow. The new application needs to issue a subscribe request to the broker and it will then start to receive messages on the given topic as they are published.

Publish/Subscribe with Message Broker offers both topic and content based routing. With content based routing it is possible to supply an SQL expression which can be applied against the contents of the message. If the expression evaluates to TRUE the subscriber will receive the message. If it evaluates to FALSE it will not receive the message. This is a very useful technique and can be used to significantly reduce the number of messages which are sent. With topic based routing a subscriber will be sent all messages for the topic on which they have registered. They may not want all of those messages and may have to discard many of them dependent on the content. Content based routing overcomes this wasted distribution and is a useful technique for reducing the number of messages which are sent. It allows you to target message delivery much more closely to the subscribers requirements.

Publish/Subscribe

- Subscribers
  - Total number
  - Number per topic
    - Affects number of MQPUTs
    - Effect on broker queue manager log for persistent messages
- Consider Collectives
  - For availability
  - To distribute subscribers
- Broker database performance
  - Frequent subscribe/unsubscribe
  - Retained publications

Notes

In implementing a system which uses Publish/Subscribe it is important to understand how many subscribers are expected to match on a particular topic. The number of subscribers per topic will have an effect on message throughput as this will determine how many output messages have to be written per publication. The messages will be written in a single unit of work so ensure that the broker queue manager log is tuned when using persistent messages.

Where the number of subscribers matching on a topic is high (in the hundreds or thousands) this may result in a message rate and message volume which is beyond the capabilities of a single queue manager. This will also depend on the publication rate. In this case consider using a collective of brokers in order to distribute load. The subscribers can then be allocated across the members of the collective rather than them all trying to use the same broker.

Use of a collective also improves availability of the Publish/Subscribe service. In the event of failure on any one broker, subscribers would need to connect to another broker and re-subscribe. A publisher may also need to reconnect to a broker in the collective.

If subscribers are constantly subscribing and unsubscribing on topics or you plan to use retained publication you will need to ensure that the broker database is well tuned. You will need to pay attention to the performance of the database manager log since these operations will cause insert and delete activity.

Message Flow Properties

- Specific versus Generic
- Parsing
- Data model
- Number of nodes in a message flow
- ESQL
- Environment
- Length
- Serialization
- Transactional support

Notes

A message flow contains many different properties. We will now look at the most the important ones.

Each of the items in this list affect the capability of a message flow to process messages. Some are obvious such as the amount of ESQL. Obviously the more ESQL the more involved the processing will be. Others will be less obvious such as the use of parsing and the extent to which partial parsing can be used for example.

You have the ability to directly influence the processing profile of the message flow by making decisions which will minimize the amount of processing, both directly and indirectly. Reducing the number of compute nodes will reduce the amount of tree copying for example. The data model use can also significantly affect the amount of indirect work which has to take place.

Specific versus Generic

- One flow per message type or one flow per group of messages
- Specific flows
  - ✗ Multiple flows to code and manage
  - ✓ Processing can be optimized for the message
- Generic flows
  - ✗ Processing is not optimized for the message
  - ✗ Balancing order of processing VS frequency of use
  - ✗ Can lead to multiple parses
  - ✓ Fewer flows to manage

Notes

In designing message flows there is the option of producing a message flow for each different type of input message or not. Where a unique message flow is produced for each different type of message it is referred to as a specific flow.

An alternative approach is to create message flows which are capable of processing multiple message types. There may be several such flows, each processing a different group of messages. We call this a generic flow.

There are advantages and disadvantages for both specific and generic flows.

With specific flows there are likely to be many different message flows. In fact as many message flows as there are types of input message. This does increase the management overhead which is a disadvantage.

A benefit of this approach is that processing can be optimized for the message type. The message flow does not have to spend time trying to determine the type of message before then processing it.

With the generic flow there is a processing cost over and above the specific processing required for that type of message. This is the processing required to determine which of the possible message types a particular message is. Generic flows can typically end up parsing an input message multiple times. Initially to determine the type and then subsequently to perform the specific processing for that type of message.

A benefit of the generic flow is that there are likely to be fewer of them as each is able to process multiple message types. This makes management easier.

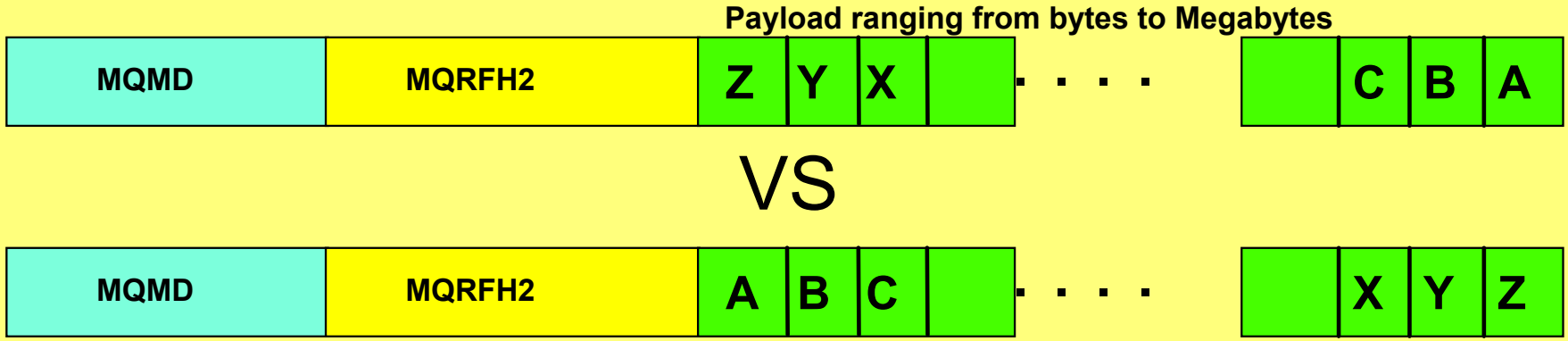
Generic flows tend to be more complex in nature. It is relatively easy to add processing for another type if message though.

With generic message flows it is important to ensure that the most common message types are processed first. If the distribution of message types being processed changes over time this could mean processing progressively becomes less efficient. You need to take account of this possibility in you message flow design. Use of the RouteToLabel node can help with this problem.

There are advantages to both the specific and generic approaches. From a message throughput point of view it is better to implement specific flows. From a management and operation point of view it is better to use generic flows. Which approach you choose will depend on what is important in your own situation.

Parsing

- Supported parsers
  - Generic XML, MRM XML, CWF, TDS, (RYO)
- Specify on MQInput node, RFH2 header or ResetContentDescriptor node
- Builds the initial message tree
- Reduce number of parses where possible
- Careful ordering of message fields can reduce the cost



Notes

Message Broker has parsers for a number of different message types. These are for Generic XML, MRM XML, CWF and TDS messages. If that does not cover what you need there is the possibility of Roll Your Own (RYO), that is writing your own.

You can specify which parser is to be used on an MQInput or a ResetContentDescriptor node or in an MQRFH2 header. The ResetContentDescriptor node allows you to re-parse a message using a different parser.

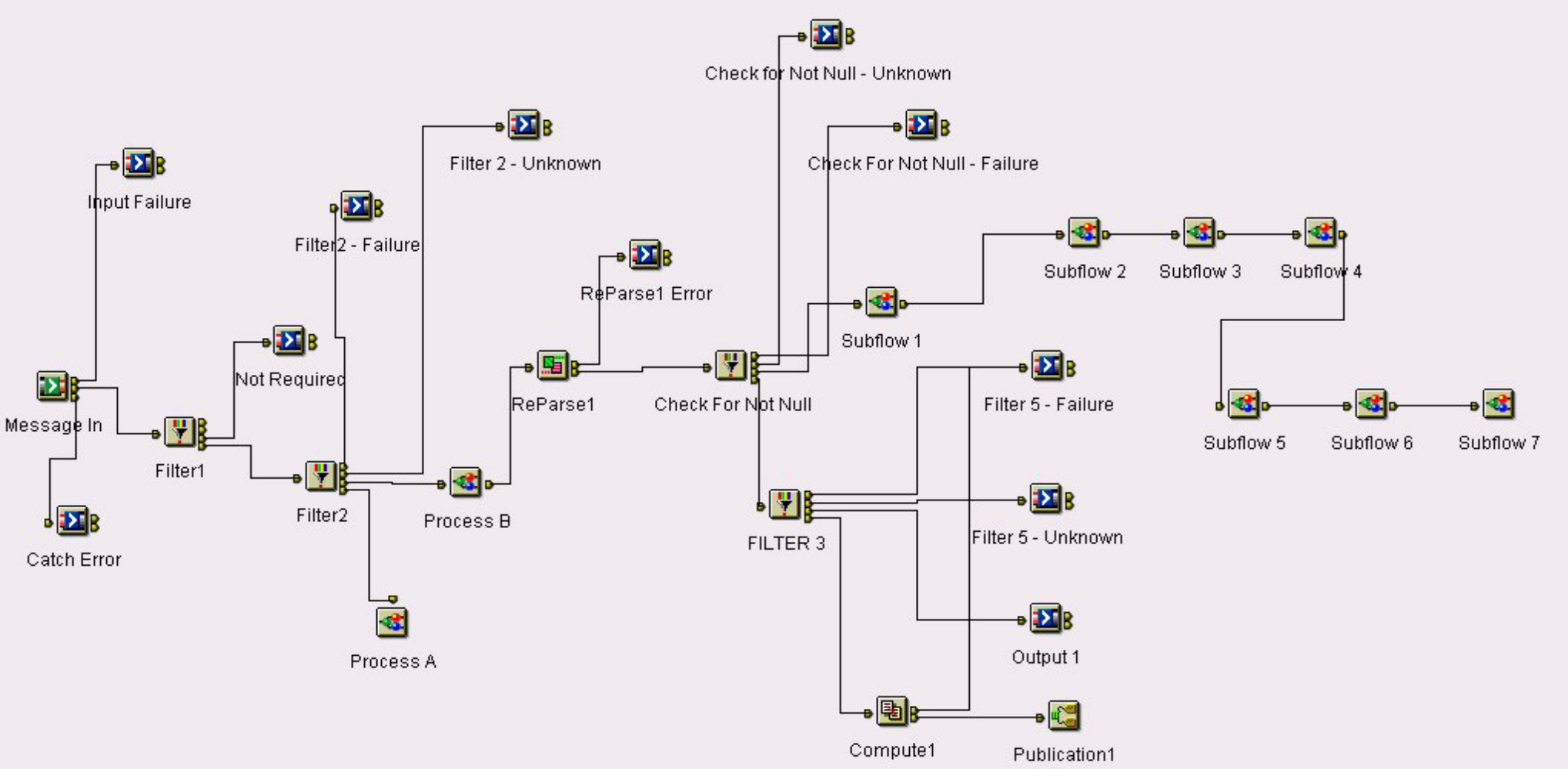
It is important to recognize that the parsers have different processing costs. A factor in this is different message formats. Read the performance reports for more information.

If you are able to identify the type of message which has been read this could save multiple parsers of the message. We discussed identifying messages earlier by promoting or copying data to fields in the MQMD or MQRFH2 headers.

Through careful message design it is possible to reduce the cost of parsing. Message Broker implements partial parsing of messages, which means that the parser only reads as much of a message as is needed to access the element which has been requested. In order to benefit from partial parsing you need to order the data in the message carefully. In a large message the difference between accessing field 'Z' and field 'A' could be very significant. Keep key routing fields at the start of message rather than at the end of the message therefore when possible.

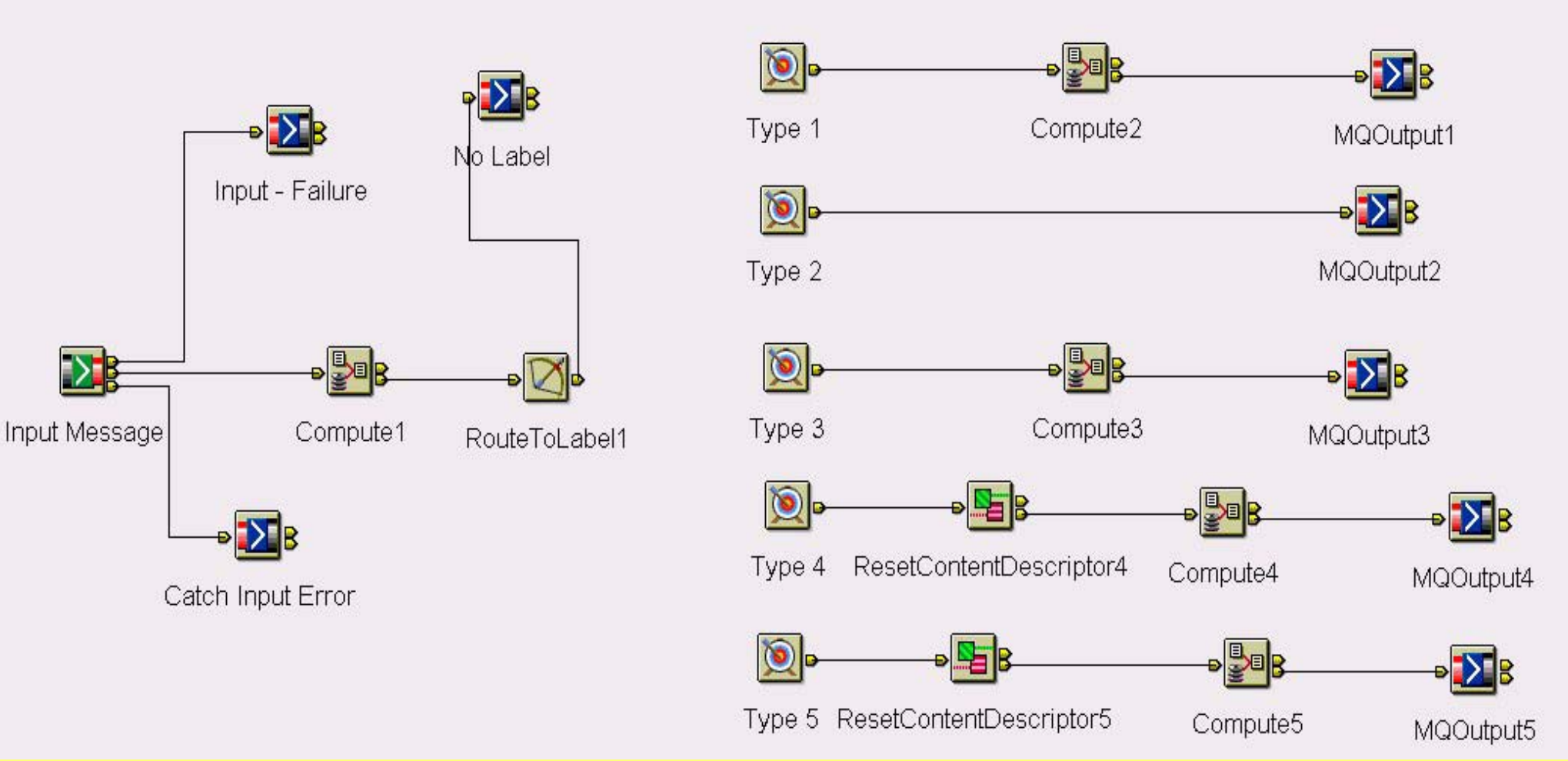
Flow Restructuring Example

- Message Flow processing multiple tag delimited input message formats of variable length



Flow Restructuring Example

- Rewritten as...



✓Throughput went from 5 (non persistent) to 138 (persistent) msgs/sec

Notes

This example shows how a message flow which processed multiple types of input message was restructured to provide a substantial increase in message throughput.

Originally the flow was implemented as a flow with a long critical path. Some of the most popular messages were not processed until late in the message flow resulting in a high overhead. The flow was complex. This was largely due to the variable nature of the incoming messages, which did not have a clearly defined format. Messages had to be parsed multiple times.

The message flow was restructured. This substantially reduced the cost of processing. The two key features of this implementation were to use a very simple message set initially to parse the messages and the use of the RouteToLabel and Label nodes within the message flow in order to establish processing paths which were specialized for particular types of message.

Because of the high processing cost of the initial flow it was only possible to process 5 non persistent message per second using one copy of the message flow. With the one copy of the restructured flow it was possible to process 138 persistent messages per second on the same system.

By running five copies of the message flow it was possible to:
Process around 800 non persistent messages per second when running with non persistent messages.
Process around 550 persistent messages per second when running with persistent messages.

Message Flow Data Model

- Message representation in the message flow
  - Message tree
  - Can affect ESQL and navigation costs
- "Given" versus "Standardized" format
- "Given" (by the parser)
  - InputRoot.MRM.FieldA
  - InputRoot.XML.Field1
- "Standardized"
  - InputRoot.MyCustomTree.FieldA
  - Involves additional processing at input and output stages
- "Given" format is recommended

Notes

The typical method of accessing and updating elements of a message tree is to work with the message tree as generated by the parser. This means working with references such as InputRoot.MRM.FieldA. This has the advantage that the message tree is being processed as it was generated by the parser and results in no additional overhead.

An alternative method used in some message flow designs is to implement an internal standard data format. In such cases all incoming message data has to be mapped to a standard tree structure (of your design) which incorporates all possible data formats. This then allows messages to be processed by a series of common subflow's which are able to reference element names. In such a case you would have copy InputRoot.MRM.FieldA to InputRoot.MyCustomTree.FieldA before then working with InputRoot.MyCustomTree.FieldA.

When an output message has to be created the required elements are copied from the standard tree format to the specific tree structure for that particular output message.

Whilst this approach has some merits in that standardized subflow's can be developed and potentially be reused across multiple message flows be aware that there will be an overhead for every message as additional tree copying takes place.

You need to decide how data will be handled in your message flows. I refer to the choice as the "Given" versus "Standardized" format. The "Given" format is the recommended approach.

Number of Nodes in a Message Flow

- Message flows can (and do) consist of many nodes
  - Message Broker Toolkit leads you to do this
  - Often makes it easier to understand logic
- Node movement Costs
  - Node initialisation/termination (cheap)
  - Message tree copying (can be expensive) for recovery
- Minimize number of nodes on critical paths
- Subflow's
  - Facilitate code reuse and are flattened at deploy time
  - Can lead to additional (hidden) costs to marshal/unmarshal
  - Consider using ESQL procedures

Notes

Message flows can and do have many processing nodes within the them. There are advantages to doing this. Processing can be split into modules. This often makes it easier to understand the processing sequence within a message flow. The visual representation outlines the structure more clearly than a single processing node with many lines of ESQL can. The Message Broker Toolkit tends to increase the use of processing nodes because it is a graphical interface.

It is important to understand the implications of having many processing nodes within a message flow though.
The costs of moving between processing nodes will vary dependent on the node type. With all nodes there is a small processing cost associated with the setup of the new node. This is node initiation. In some cases a copy of the message tree is made. This occurs in Compute nodes for example. This can be a relatively expensive operation as the tree copy is a copy of a structured object and cannot simply be copied as a sequence of bytes. As the tree copy is expensive it is important to minimize the number of such copies where possible. In some situations the copying is unavoidable. It may be required for recovery purposes for example. When it is needed it should be used. The intention here is to highlight the issue and make you aware of the cost so that you can reduce the number of tree copies where possible.

It is a good idea to minimize the number of nodes along the critical path of a message flow, especially where message throughput or CPU usage are an issue. Every bit of processing which can be saved the better in order to improve the efficiency of your message flows. The node rationalization which took place in WebSphere MQ Integrator V2.1 helps with this process. All SQL constructs (statements, operators and functions) are now usable in all nodes. All constructs have exactly the same effect regardless of the which type of node that are used. However there are still some differences in the data which can be updated. It is not possible to update a message in a Filter or Database node for example. It is possible to update the Environment though so data can be placed in the there and subsequently copied to the message in another node.

Subflow's are a useful facility which encourage the reuse of code. Subflow's which contain common routines can be embedded into message flows. The subflow's are 'in-lined' into the message flow at deploy time. There are no additional nodes inserted into the message flow as a result of using subflow's. However be aware of implicitly adding extra nodes into a message flow as a result of using subflow's. in some situations compute nodes are added to subflow's to perform marshaling of data from one part of the message tree into a known place in the message tree so that the data can be processed by the message flow. The result may then be copied to another part of the message tree before the subflow completes. This approach can easily lead to the addition of two compute nodes, each of which performs a tree copy. In such cases the subflow facilitates the reuse of logic but unwittingly adds an additional processing head.
Consider the use of ESQL procedures instead for code reuse.

ESQL

- **ESQL Rationalization**
  - Improves ability to minimize #nodes
  - All SQL constructs usable in all nodes
  - All constructs have same effect regardless of node type
  - Some differences in the data which can be updated.
- **Cost**
  - Dependent on amount and complexity
  - Includes navigation
- **Implementation Tips**
  - **Use**
    - Dynamic references
    - PROPAGATE
    - RETURN for early exit
    - Environment correlation
    - Parameter markers with PASSTHRU statement
  - **Minimize**
    - Conversion between types
    - Use of EVAL
    - Database operations
  - Avoid function calls in loop constructs
  - Save intermediate results in message tree

Notes

The use of ESQL allows you to implement significant and substantial processing in message flows. Most of the processing is concerned with validation, cross checking and message manipulation.

It is sensible to minimize the amount of ESQL where possible. Also aim to spread it as few nodes as possible.

The cost of processing ESQL is dependent on the quantity and complexity of the ESQL. The amount you need will depend on the situation and will vary from one case to another. There are some guidelines which you can follow in order to make processing as efficient as possible. These are:

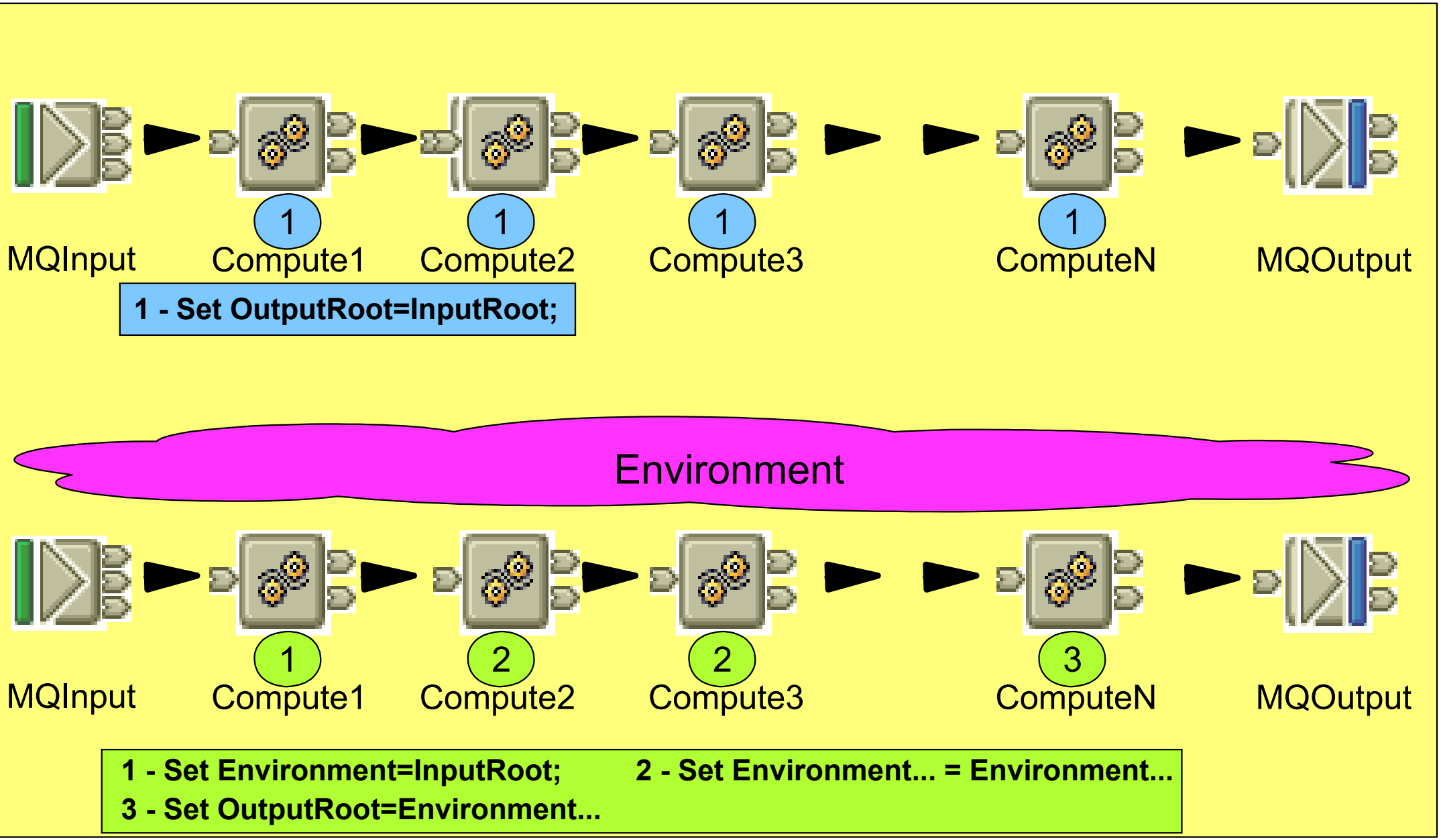
1. Use dynamic references in order to improve the efficiency of navigation
2. Use the PROPAGATE statement when processing large messages in order to reduce the volume of data being passed around in the message tree.
3. Use the RETURN statement to make an early exit from complex logic
4. Use the Environment Correlation to improve performance. This is discussed in more detail in the next foil
5. Use parameter markers '?' when using the PASSTHRU statement. This is so that the Dynamic SQL statement can be reused.
6. Minimize the conversion of data between different types.
7. Minimize use of the EVAL statement as this is expensive to process
8. Minimize the number of database operations which take place.
9. Avoid the use of function calls such as CARDINALITY in loop constructs as they can lead to an N^2 cost with large messages
10. Where possible save intermediate results in the message tree in order to save having to recalculate them

Environment Correlation

- A semi-durable global area
  - Does not survive the message flow invocation
- Can significantly reduce processing costs through
  - Reduced tree copying
  - Preserving intermediate results
- Need to consider effects on node rollback
  - No automatic rollback as with tree copy
- Contents can be seen across the whole flow
  - Could be good or bad

✓ Good for performance

Environment Correlation



Notes

This foil illustrates use of the Environment correlation.

Before Environment came into existence in WMQI V2.1 a message flow consisting of multiple compute nodes had to copy InputRoot to OutputRoot in each compute node in order to preserve message data. This resulted in a tree copy and so additional processing.

Environment is a scratchpad area which exists for the life of the message flow. The contents of Environment are accessible across the whole of the message flow. Environment is cleared down at the end of the message flow.

In order to reduce the amount of message tree copying which takes place within a message flow it is recommended that the first compute node copy InputRoot to Environment. Intermediate nodes then read and update values in the Environment instead of using the InputRoot and OutputRoot correlations as previously.

In the final compute node of the message flow OutputRoot must be populated with data from Environment. The MQOutput node will then serialize the message as usual. Serialization will not take place from Environment.

Whilst use of the Environment correlation is good from a performance point of view be aware that the any updates made by a node which subsequently generates an exception will remain in place. There is no backout of changes as there is when a message tree copy was made prior to the node exception.

Message Flow - Length and Serialization

- Multiple short flows versus fewer longer ones?
- Short Flows
  - ✓ Short units of work
  - ✓ Lower resource requirements
  - ✗ Risk of multiple parses
  - ✗ Loss of context
  - ✗ Higher level of MQ activity
- Long Flow
  - ✓ Maximizes use of state and parsing
- Serialization
  - Avoid it! - limits scalability
  - Minimize serialized processing

Notes

There is a choice of whether to implement the logic required in a business function into multiple messages flows, each with discrete function. Multiple message flows are then required to process a single input message.

Each time control moves from one message flow to another there is a processing overhead in doing this which is much greater than the cost of moving between processing nodes in a single message flow. This is because one or more output messages must be written only to be read in and processed by the next message flow in the sequence. At which point parsing and building of state must take place. Sometimes it will be necessary to recalculate information or intermediate results which were held in the previous flow but which were not written in the output message. This is repeated, wasted processing and can result in a significant overhead. It is recommended to combine message flows where possible.

Serialization of processing in a message flow is a major inhibitor to scaling message throughput with additional instances or multiple execution groups so avoid single threading such as sequencing messages by userid. When sequencing messages by userid you can only use additional instances to increase the number of copies of the message flow which are concurrently running. If you use more than one execution group sequencing cannot be guaranteed.

Where processing is serialized aim to minimize the serialized sections. Consider producing one message flow which contains the serialized processing and another which contains processing which can run in parallel. Use additional instances to increase throughput with this latter flow.

Avoid dependency on a particular instance of an application or machine. If a particular application cannot be upgraded to handle more requests for example you may find that message throughput is unnecessarily constrained and that this application effectively becomes a point of serialization.

Transactional Support

- WMQI provides transactional support by using WebSphere MQ
  - Local units of work for message processing
  - Global units of work for message and external data processing
- MQPUT/MQGET out of syncpoint
  - Early availability of messages
  - No back out
  - On distributed platforms use persistent messages in syncpoint
- If not using WMQI facilities
  - What will mange data integrity
  - Is there retry logic

Notes

For most applications it is essential that data is not lost. That is why many companies use WebSphere MQ for messaging. Assured delivery, once and only once is what WebSphere MQ is famous for. If you need this level of protection you must use persistent messages.

What ever the requirement it is important to understand what the recovery requirements are from the beginning of the message flow design/architecture phase. Data integrity must be part of the design and not something which you attempt to add at the end of the implementation. Providing integrity requires you to examine the configuration of the broker, its queue manager and any database from which business data is processed as well as the applications which create input for the message flows and consume the output messages.

WebSphere Business Integration Brokers offer two levels of data protection. These are local and global units of work. Local units of work are the protection of message operations. This means MQGET and MQPUT operations can take place within a unit of work and all will take place if the unit of work is committed or non if the unit of work is backed out. WBIMB uses the facilities of the WebSphere MQ queue manager for this recovery processing and does not provide its own unit of work processing capability.

Individual message operations can form part of a unit of work, so not all operations in a message flow have to be within a unit of work. In order to have message operations performed within a unit of work you must specify a value for transactionMode on the appropriate MQInput or MQOutput nodes in the message flow. Be aware of the different defaults. The default value for an MQInput node is Yes. For an MQOutput node the default value is Automatic.

Global units of work provide protection for messaging operations and database update/insert activity. That is database operations involving business data. This is data accessed through one of the database nodes, a filter node or a PASSTHRU statement in a compute node.

As with Local units of work, brokers use the recovery facilities of WebSphere MQ. In particular the two phase commit support which it provides. For Global units of work to function there must be an XA interface defined between the database manager and the brokers queue manager on the Windows and UNIX platforms. On z/OS RRS is used to perform the transaction coordination role.

The whole message flow must operate within a global unit of work. There is no selection of individual nodes as there is with local units of work. In order to select global units for a message flow the property coordinatedTransaction must be set for the message flow in the assignments view on the Eclipse Workbench.

Be aware that messages which are not processed as part of a unit of work will become visible almost immediately to other applications and will not wait for the completion of the unit of work. This may be an effect that you are after, on the other hand it may be a problem.

Broker Environment - Broker & Queue Manager

- Configuration is important
  - Broker
  - Broker Queue manager
  - Topology
  - Broker database
  - Business database
  - Hardware
  - Software
- Broker
  - Trusted application or not (not on AIX or z/OS)
  - Use of instances and execution groups
- Broker queue manager
  - Logging
  - Trusted channels and listeners
  - Performance of connected queue managers
- Topology
  - Distance between broker and data

Notes

So far we have focused on the contents of the message flow. It is important to have an efficient message flow but this is not the whole story. It is also important to have an efficient environment for the message flow to run in.

We collectively refer to this as the broker environment. It includes:

- configuration of the broker
- configuration of the broker queue manager,
- the topology of the environment in which the messages are processed
- broker database configuration
- business database configuration
- Hardware
- Software

It is important that each of these components is well configured and tuned. When this occurs we can get the best possible message throughput for the resources which have been allocated.

We will now briefly look at each of these aspects. The aim is to outline those aspects which you should be aware of and further investigate rather than to provide full guidance in this presentation.

Notes

As with other MQ applications it is possible to run a broker as a trusted application on the Windows NT, Windows 2000, Solaris and HP-UX platforms. The benefit of running in trusted mode is that the cost of an application communicating with the queue manager is reduced and MQ operations are processed more efficiently. It is thus possible to achieve a higher level of messaging with a trusted application. However when running in trusted mode it is possible for a poorly written application to crash and potentially corrupt the queue manager. The likelihood of this happening will depend on user code, such as a plugin node, running in the message flow and the extent to which it has been fully tested.

Using the Message Broker Toolkit it is possible to increase the number of instances of a message flow which are running. Similarly you can increase the number of execution groups and assign message flows to those execution groups. This gives the potential to increase message throughput as there are more copies of the message flow. There is no hard and fast rule as to how many copies of a message flow to run. For guidelines look at the details in the Additional Information section

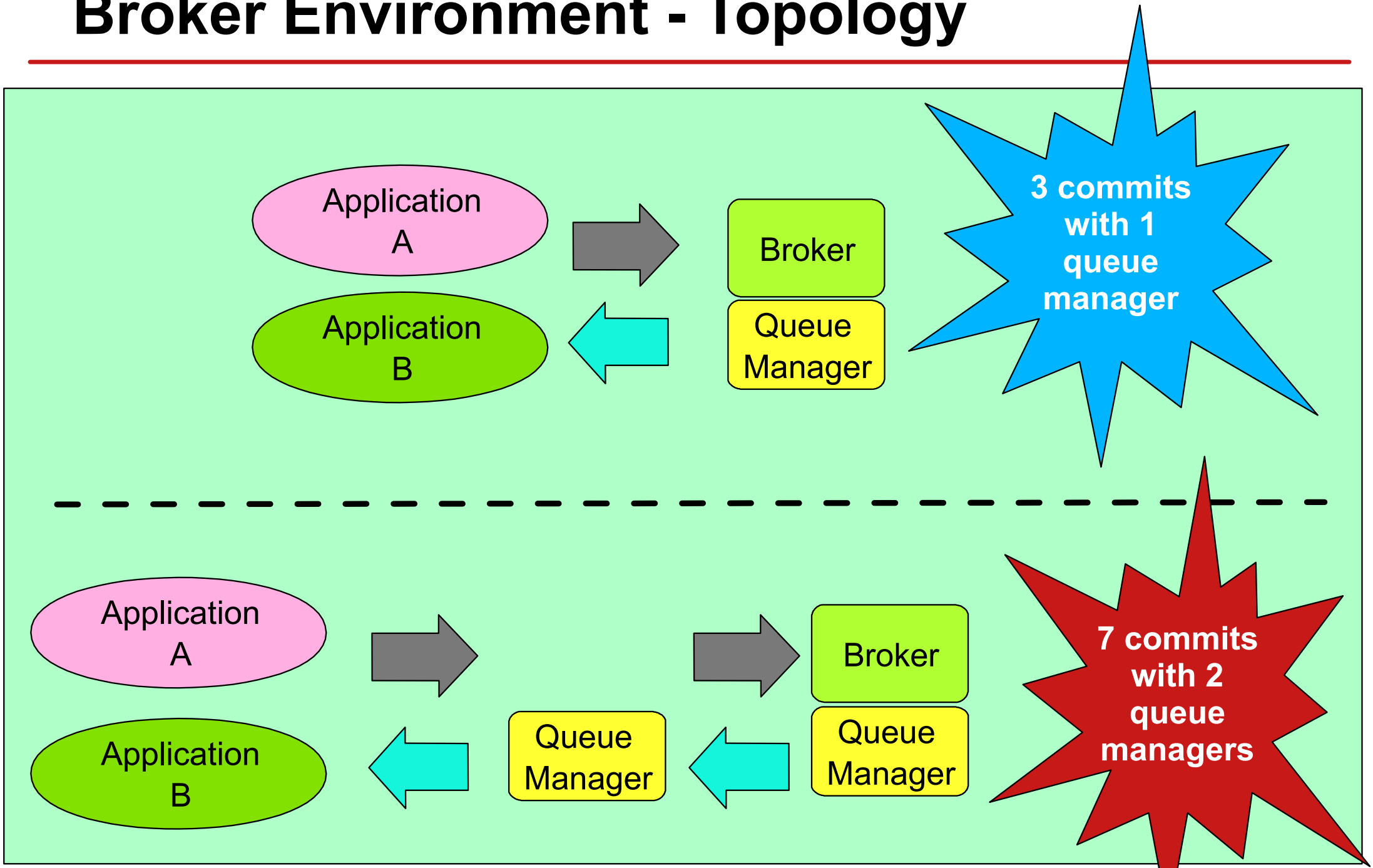
As the broker is dependent on the broker queue manager to get and put MQ messages the performance of this component is an important component of overall performance.

When using persistent messages it is important to ensure the queue manager log is efficient. Review log buffer settings and the speed of the device on which the queue manager log is located.

It is possible to run the WebSphere queue manager channels and listener as trusted applications. This can help reduce CPU consumption and so allow greater throughput.

In practice the broker queue manager is likely to be connected to other queue managers. It is important to examine the configuration of these other queue managers as well in order to ensure that they are well tuned.

Broker Environment - Topology



Notes

The input messages for a message flow do not magically appear on the input queue and disappear once they have been written to the output queue by the message flow.

The messages must some how be transported to the broker input queue and moved away from the output queue to a consuming application.

Let us look at the processing involved to process a message in a message flow passing between two applications with different queue manager configurations.

Where the communicating applications are local to the broker, that this they use the same queue manager as the broker, processing is optimized. Messages do not have to move over a queue manager to queue manager channel. When messages are non persistent no commit processing is required. When messages are persistent 3 commits are required in order to pass one message between the two applications. This involves a single queue manager.

When the communicating applications are remote from the broker queue manager, messages passed between the applications must now travel over two queue manager to queue manager channels (one on the outward, another on the return). When messages are non persistent no commit processing is required. When messages are persistent 7 commits are required in order to pass one message between the two applications. This involves two queue managers.

If the number of queue managers between the applications and broker where to increase so would the number of commits required to process a message. Processing would also become dependent on a greater number of queue managers.

Where possible keep the applications and broker as close as possible in order to reduce the overall overhead.

Where multiple queue managers are involved in the processing of messages it is important to make sure that all are optimally configured and tuned.

Broker Environment - Database

- Broker database
  - Level of use varies
  - Different requirements for different usage
  - Logging in some circumstances
- Business database
  - Level of use varies
  - Database manager configuration
  - Logging in some circumstances
  - Table design
  - Read only views

Notes

The extent to which the broker database will be used depends on which facilities are used in the message flows. Use of aggregation nodes, or publication nodes can lead to increased use of the broker database. In this case it will be necessary to ensure that the broker database is well tuned, especially to handle insert/delete activity. The performance of the database log will then become important and may be a gating factor in overall performance.

The extent to which business data is used will be entirely dependent on the business requirements of the message flow. With simple routing flows it is unlikely that there will be any database access. With complex transformations there might be involved database processing. This could involve a mixture of read, update, insert and delete activity.

It is important to ensure that the database manager is well tuned. Knowing the type of activity which is being issued against a database can help when tuning the database manager as you can focus tuning on the particular type of activity which is used.

Where there is update, insert or delete activity the performance of the database log will be a factor in overall performance. Where there is a large amount of read activity it is important to review buffer allocations and the use of table indices for example.

It might be appropriate to review the design of the database tables which are accessed from within a message flow. It may be possible to combine tables and so reduce the number of reads which are made for example. All the normal database design rules should be considered.

Where a message flow only reads data from a table, consider using a read only view of that table. This reduces the amount of locking within the database manager and reduces the processing cost of the read.

Broker Environment - Hardware & Software

- Understand the requirements of the message processing
  - Run the message flows with realistic data
  - Use early testing to plan production environment
- I/O or CPU bound
  - Depends on msg type and complexity of flows
- Number and speed of processors
- Software
  - Use latest levels
  - Ensure it is tuned

Notes

Allocating the correct resources to the broker is a very important implementation step.

Starve a CPU bound message flow of CPU and you simply will not get the throughput. Similarly neglect I/O configuration and processing could be significantly delayed by logging activities.

It is important to understand the needs of the message flows which you have created. Determine whether they are CPU bound or I/O bound.

Knowing whether a message flow is CPU bound or I/O bound is key to knowing how to increase message throughput. It is no good adding processors to a system which is I/O bound. It is not going to increase message throughput. Using disks which are much faster such as a solid state disk or SSA disks with a fast-write nonvolatile cache will have a beneficial effect though.

As we saw at the beginning Message Broker has the potential to use multiple processors by running multiple message flows as well as multiple copies of those message flows. By understanding the characteristics of the message flow it is possible to make the best of that potential.

In many message flows parsing and manipulation of messages is common. This is CPU intensive activity. As such message throughput is sensitive to processor speed. Brokers will benefit from the use of faster processors. As a general rule it would be better to allocate fewer faster processors than many slower processors.

It is important to ensure that software levels are as current as possible. The latest levels of software such as WebSphere Business Integration Brokers and WebSphere MQ offer the best levels of performance. Take advantage of that.

We have talked about ensuring that the broker and its associated components are well tuned and it is important to emphasize this. The default settings are not optimized for any particular configuration. They are designed to allow the product to function not to perform to its peak.

Additional Information

- Summary of Available Performance Information:
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0403_dunn/0403_dunn.html
- WebSphere MQ Integrator V2.1 Supportpacs:
  - IP11 - z/OS V2.1 Performance Report
  - IP66 - AIX V2.1 Performance Report
  - IP67 - Sun Solaris V2.1 Performance Report
  - IP68 - HP-UX V2.1 Performance Report
  - IP72 - Windows NT and Windows 2000
- WebSphere Business Integration Message Broker V5 Supportpacs:
  - IP12 - z/OS V5 Performance Report
  - IP69 - AIX V5 Performance Report
  - IP6A - HP-UX V5 Performance Report
  - IP6B - Sun Solaris V5 Performance Report
  - IP73 - Windows NT and Windows 2000 V5
  - IP73 - Linux V5
  - **All Performance Reports are available at http://www.ibm.com/software/integration/support/supportpacs/perfreppacs.html**
- Optimizing Usage of Aggregation Nodes
  - SupportPac IP05 available at http://www.ibm.com/software/integration/support/supportpacs/individual/supportpacs/ip05.pdf
- WebSphere Developer Domain Articles:
  - Design and Implementation Considerations for IBM WebSphere MQ Integrator Message Flows
  - Determining How Many Copies of a Message Flow Should Run Within WebSphere Business Integration Message Broker V5
  - How to Estimate Message Throughput For an IBM WebSphere MQ Integrator V2.1 Message Flow
  - Optimising DB2 UDB for use with WebSphere Business Integration Message Broker V5
  - Tuning a WebSphere MQ Integrator Message Broker
  - **URLs for these articles are given in the notes**

Notes

A number of articles have been written on various aspects of Message Broker performance. These are written to help you understand to tune and configure Message Broker.

The articles and their URLs are as follows:

- Design and Implementation Considerations for IBM WebSphere MQ Integrator Message Flows at http://www.ibm.com/developerworks/websphere/library/techarticles/0211_dunn/dunn.html
- Determining How Many Copies of a Message Flow Should Run Within WebSphere Business Integration Message Broker V5 at http://www.ibm.com/developerworks/websphere/library/techarticles/0311_dunn/dunn.html
- How to Estimate Message Throughput For an IBM WebSphere MQ Integrator V2.1 Message Flow at http://www7b.boulder.ibm.com/wsdd/library/techarticles/0208_dunn/dunn.html
- Optimizing DB2 UDB for use with WebSphere Business Integration Message Broker V5 at http://www.ibm.com/developerworks//websphere/library/techarticles/0312_acourt/acourt.html
- Tuning a WebSphere MQ Integrator Message Broker at http://www.ibm.com/developerworks/websphere/library/techarticles/0301_dunn/dunn.html

Summary

- Key factors affecting performance:
  - Parsing
  - ESQL
  - Navigation
  - Number of nodes
  - Resource usage
- Processing model
  - Look at the choices
  - Implementation by choice not simply re-implementing
- Flexibility is essential
  - Avoid restrictions/limitations
- Focus on all aspects of design and implementation
  - Messages, Message flow, Message Broker, Topology

Notes

To summarize:

Message flow performance is primarily affected by

1. The amount of parsing which takes place. This in turn is affected by the structure and organization of the message being processed. To reduce the costs of parsing consider promoting data to the MQMD or an MQRFH2 header where only a couple of fields are required such as in a simple routing message flow. In other cases placing frequently accessed fields at the start of the message rather than the end will significantly improve processing costs.
2. The quantity and complexity of ESQL within the message flow. Message flows with small amounts of ESQL or very simple ESQL will have lower costs than those with more complex processing. In most cases the amount and complexity of ESQL is driven by the business needs of message processing. Remember however that Message Broker is not an application server. It is what it says on the can!
3. The navigation costs of accessing elements of the message tree. The costs are dependent on the size of the tree and the way in which the ESQL is written. As we discussed earlier reference pointers can help significantly in reducing the costs of navigation as can the copying of messages at the highest level within the MRM.
4. The number of nodes. We briefly looked at the implications of having more nodes in a message flow. The main increase in cost comes from message tree copying when using Compute nodes. so to reduce this minimize the number of compute nodes.
5. Resource use. This is the number of WebSphere MQ and database requests issued by the message flow. Obviously the lower the number of requests the lower the costs but you obviously have to meet the business requirements of the message flow. When processing persistent WebSphere MQ messages process them with a transaction mode of Yes. This is more efficient, especially on the Windows and UNIX platforms. With database requests you could consider the use of Commit Count to batch requests. The use of stored procedures may be beneficial in some cases where a message flow issues multiple requests to a database in one invocation of the message flow.

There are a number of processing models possible with a message flow. These are the Single Message, Coordinated Request/Reply and Publish/Subscribe models. Pick the most appropriate model for your needs. The model used may well vary between different implementations. Make sure that you choose the most appropriate model in each case rather than adopting a universal approach or re-implementing whatever was used last time.

Make sure you have a large degree of flexibility in your approach. Do not build affinities to particular applications or machines in otherwise this will restrict your potential to scale message throughput in the long term.

Look at all aspects of system design. This includes the design of the messages, the message flow, the broker, the broker queue manager, the broker database and the hardware and software on which this will all run. It is likely that different components will be undertaken by different people. It is important that they all communicate. The database administrator needs to know what is expected of the database for example.

Please complete your feedback forms

This was session