Introduction to MQSeries Publish/Subscribe

Steve Bolam
MQSeries Development

http://www.software.ibm.com/ts/mqseries

Contents

Contents

NOTES

- This presentation introduces MQSeries Publish/Subscribe, in particular it
  - describes the publish/subscribe model
  - introduces some of the key terms and concepts
  - shows how messages (and information) flow within an MQSeries Publish/Subscribe network
  - describes the format of the messages used by applications using MQSeries Publish/Subscribe
  - describes the commands that are available for controlling MQSeries Publish/Subscribe
  - illustrates the features available to the systems programmer
- For more information consult the MQSeries Publish/Subscribe User's Guide.

What is Publish/Subscribe ?

Publish/Subscribe is a term used to define an application model in which the provider of some information is decoupled from the consumers of that information.

- providers of information need have no knowledge of consumers
- consumers of information need have no knowledge of providers
- new providers/consumers can be added without disruption

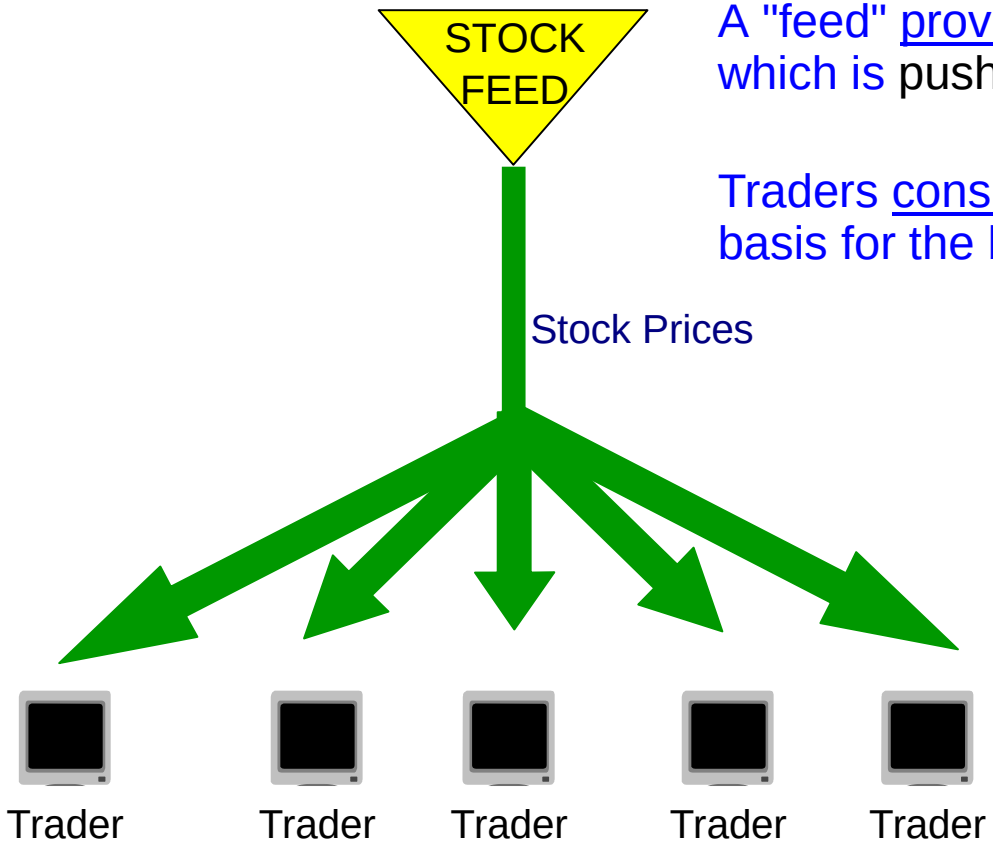
What is Publish/Subscribe

NOTES

- Publish/subscribe systems have become very popular in recent years as a way of distributing data messages from publishing computers to subscribing computers. Such systems are especially useful where data supplied by a publisher is constantly changing and a large number of subscribers needs to be quickly updated with the latest data. Perhaps the best example of where this is useful is in the distribution of stock market data.

- In such systems, publisher applications of data messages do not need to know the identity or location of the subscriber applications which will receive the messages. Similarly, the subscribing applications do not need to know the identity or location of the publishing application providing their information. In this sense the providers and consumers are said to be loosely-coupled.

The classic example



A "feed" provides a continuous flow of information which is pushed to interested parties

Traders consume this information and use it as a basis for the buying and selling stock

What is Publish/Subscribe

NOTES

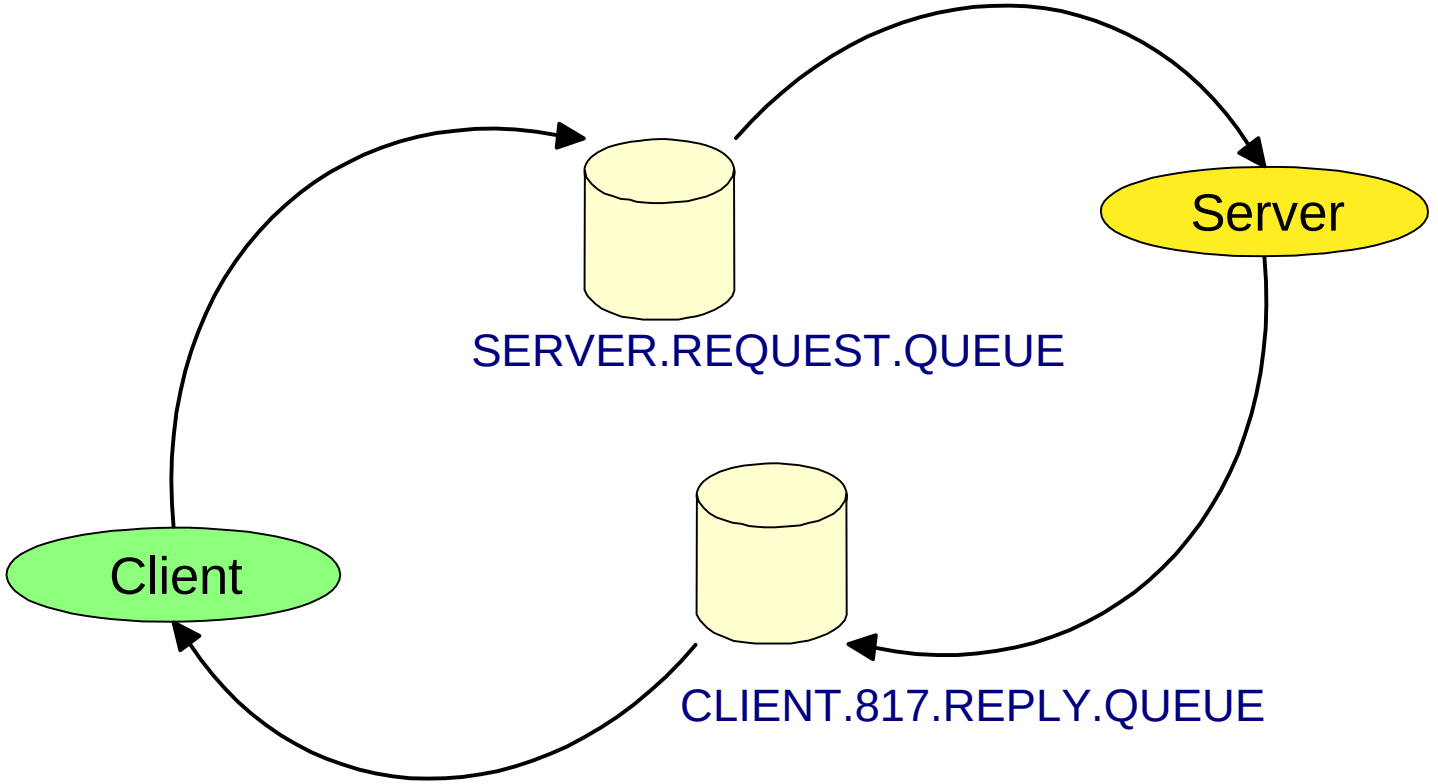
- Perhaps the most-commonly quoted example of a Publish/Subscribe system is one which provides stock-market information. Here a "feed" provides (publishes) a continuous flow of information containing the latest stock prices. The latest stock prices are required by traders who need this information in order to conduct trades. Traders register their interest in (subscribe to) particular stock prices and receive updates as prices change. Traders can be added/removed without disruption to the providers of the information who have no knowledge of who is receiving their information.

- The terms "**push**" and "**pull**" are also becoming increasingly popular when describing the flow of information between applications. If we concentrate on this example, traders receive new information from the stock-feed as soon as a stock price changes. In this sense the information can be thought of as being pushed directly to them. This pushing of information from provider to consumer is one of the major differentiators between publish/subscribe and more conventional systems. Our stock market example could have equally been designed in such a way that updated stock prices only flowed to the traders when they specifically requested, or pulled them from a central repository (server) of all stock prices. In such a system, the empahasis would instead be on the traders, to request a refresh of their stock prices on a continual basis.

- In fact MQSeries Publish/Subscribe supports both modes of operation.

The Request/Reply model

Compare with the MQSeries "Request/Reply" model in which applications communicate via named queues



The Request/Reply model

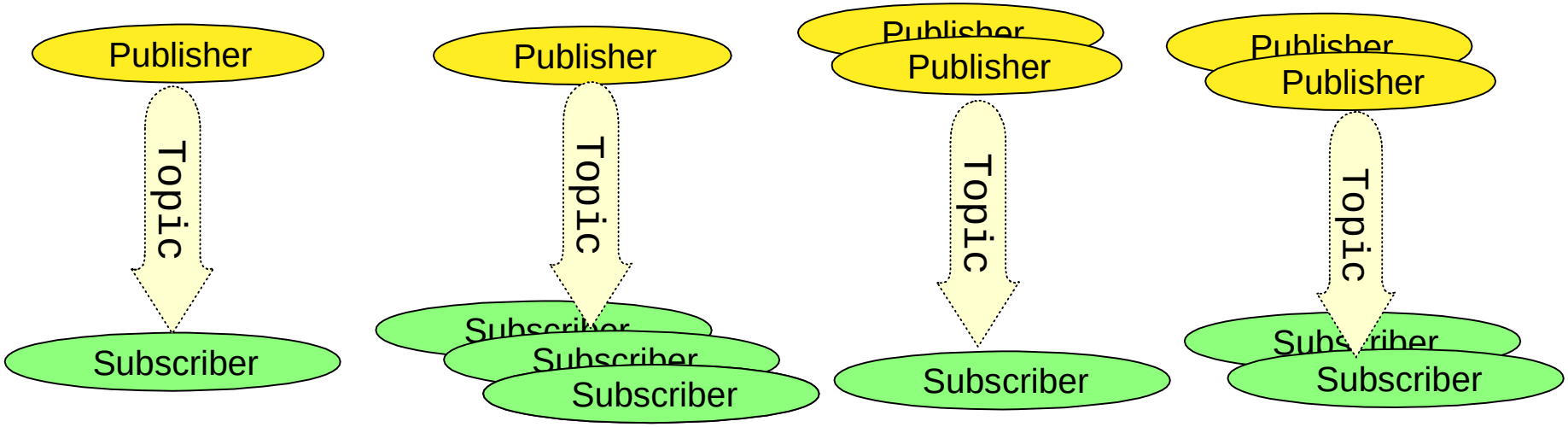
NOTES

- Before continuing with publish/subscribe let us consider how it differs from one of the most popular communication models provided by MQSeries, the "Request/Reply" model.

- The Request/Reply model is particularly suited to a client-server architecture where clients request some service from a Server application. In the previous foil the provider (server) and consumer (client) had very little knowledge of each other. In the "Request/Reply" model the applications are more tightly-coupled since they need to communicate with one another via named queues. The client puts his request on a named queue which is being read by the server. The Server passes back the information required by the client in a specific queue nominated by the client. Unlike, the Publish/Subscribe model, the Request/Reply model is very much a "pull" interface since information only flows from the provider to the consumer when the consumer requests it specifically.

Loose-coupling with Publish/Subscribe

Publish/Subscribe applications need only be loosely-coupled by a topic associated with each piece of information



1:1 1:many many:1 many:many

Loose-coupling with Publish/Subscribe

NOTES

- In the MQSeries Publish/Subscribe model the only thing which connects publishing and subscribing applications is the topic or subject which the publisher associates with his information. Publishers and subscribers need only agree on the topic to become connected to one another. Each different piece of information has its own topic associated with it. Subscribers nominate which types of information they want to receive by subscribing to specific topics.

- Publishers of information are unaware of subscribers to the extent that they may publish information even if there are no subscribing applications requiring it. Publishing and subscribing are completely dynamic processes. New subscribers and new publishers can be added to the system without disruption.

- With respect to a given topic, or piece of information, all possible combinations of publishers/subscribers are possible, that is:
  - information about each topic may be provided by a single or multiple publishing applications
  - the information may be received and processed by one or more subscribing applications

- The number of publishers and subscribers connected by a single topic depends upon the type of information which is flowing between them. As we will see later, MQSeries Publish/Subscribe supports both state and event based information, or topics.

Topic strings

- **Specifies the subject matter of publication**
- **Can be any length, any SBCS characters can be used**
- **Recommended to add structure using the / character**
- **For example, a topic string format used to group stock prices from stock markets around the world:**

<Market>/<Sector>/<Company>

Topic strings

NOTES

- Topic strings are the only thing needed to connect publishing and subscribing applications. When designing a system to use MQSeries Publish/Subscribe, firstly analyze the required flow of information between the applications. Once this task has been carried out, each piece of information will need to be described by a topic string.
- Topic strings are of arbitrary length, and can contain SBCS (single-byte character set) characters only. Topic strings are case sensitive and a blank character has no special meaning. A null character terminates the string and is not considered to be part of it.
- When designing a set of topics, it is strongly recommended to use the / character to add structure to the topic strings. Apart from making the topic strings more readable, they allow subscribing applications to request whole subsets of the available information using wildcard topics. Structuring topic strings using the / character will also ensure compatibility with other MQSeries business integration products.
- Consider our stock market example and the possible set of topic strings which could be used. We need to provide information about the stock prices from various markets. The name of the market will form the first part of our topic string. We would also like to identify in the topic string, the name of the company for which the stock price applies, and what sector or classification that the company belongs to. For example, IBM would be classified as an I.T. company and the topic string "NewYork/Information Technology/IBM" would represent the IBM stock price on the New York stock exchange.

Wildcard topics

- **Used by subscribers to receive information on multiple topics**
  - * matches with zero or more characters
  - ? matches with exactly one character

- **E.G. With topic <Market>/<Sector>/<Company>**

*	receives all stock prices from all markets
London/*	receives all stock prices from the London market
NewYork/Banks/*	receives stock prices for all banking companies from the New York market
//IBM	receives the IBM stock price from all markets

Wildcard topics

NOTES

- Subscribers can add wildcard characters to their topic strings in order to receive information on multiple topics. Two wildcard characters are permissable:
  - ***** will match any sequence of zero or more characters
  - **?** will match with exactly one character

- If the / character has been used to structure the topics then it will be easier for an application to subscribe to a particular subset of the information using a single wildcard topic string and without it needing to know exactly which topics are being published on. For example, wildcard topic:
  - '*****' receives all the stock prices from all stock markets
  - '**London/***' receives all of the stock prices from the London market
  - '**NewYork/Banks/***' receives all of the stock prices for banking companies from the New York market
  - '***/*/IBM**' receives the IBM stock price from all of the stock markets

- The use of wildcard topics also makes systems more extendible. For example, the subscriber using the "NewYork/Banks/*" topic string to receive banking stock prices needn't know the names of all the banking companies being published on. Likewise when a new banking company is added to this part of the market the subscribing application will become aware of its existence as soon as that company becomes quoted, and the new topic is published on for the first time.

Publishers, subscribers and brokers

Providers of information are called **publishers**

Consumers of information are called **subscribers**

Publishers and subscribers are connected to one another by the services provided by a **broker**

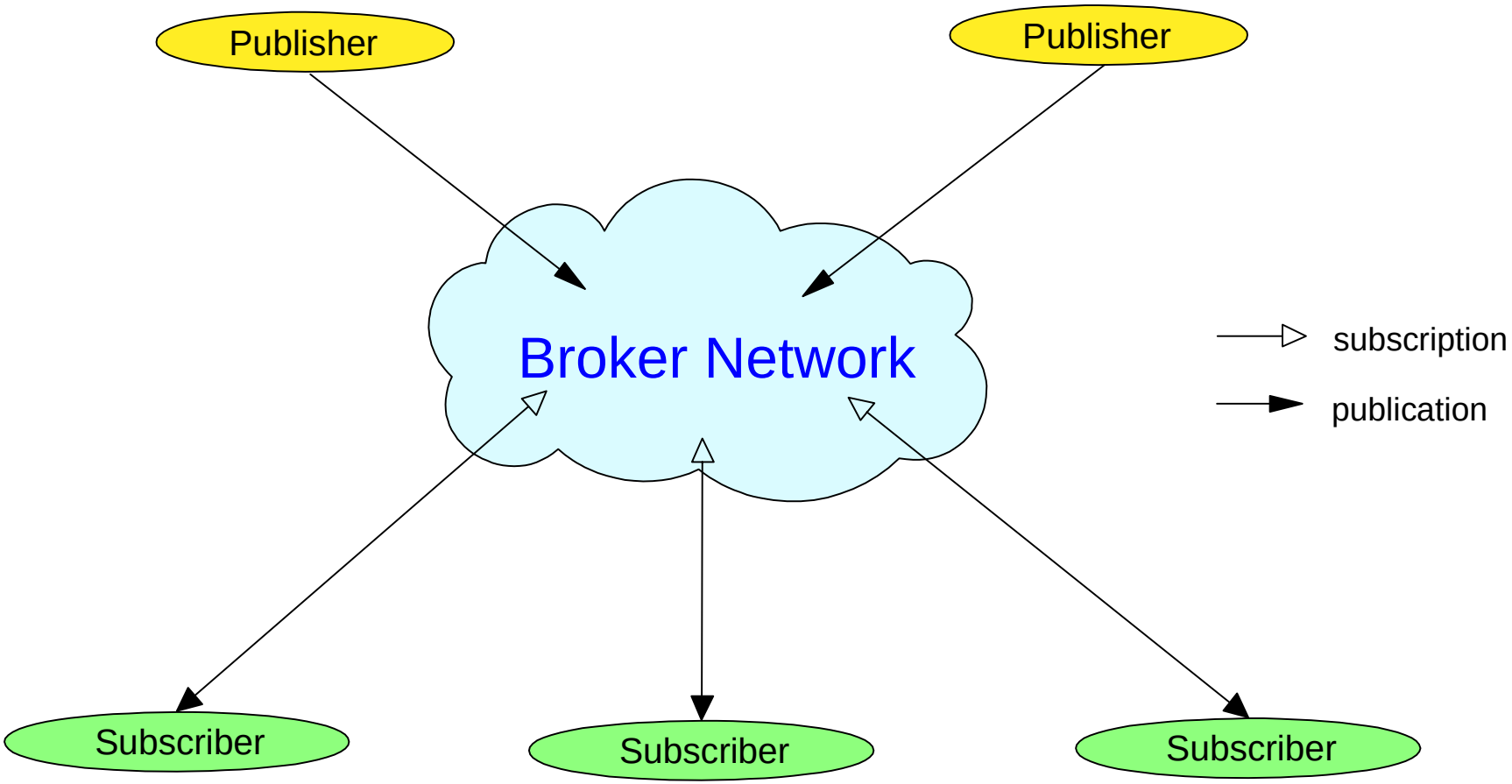
Queue manager hosts a single broker

Publishers, subscribers and brokers

NOTES

- Just to recap, applications which provide information are called publishers. Applications which consume information are called subscribers. We have already seen how topics are used to logically connect publishers and subscribers together. In practise however, a special application called a broker is needed to physically connect the publishing and subscribing applications together. A broker can be thought of as a special MQ application which runs at different queue managers within an MQ network. Each queue manager can only host a single broker.

Publishers, Subscribers and Brokers



Publishers, subscribers and brokers

NOTES

- Publishing and subscribing applications don't talk to one another directly, instead they each talk to a broker who acts as a middleman to ensure that information is passed from the publishers to the subscribers. In fact, publishing and subscribing applications don't need to talk to the same broker in order for information to flow between them. Instead they can be connected by a **broker network**, a network of connected brokers each running at different queue managers.

Publications and subscriptions

Subscribers send **subscriptions** to the broker to register their interest in information relating to specific topics

Publishers provide information about specific topics by sending **publications** to the broker

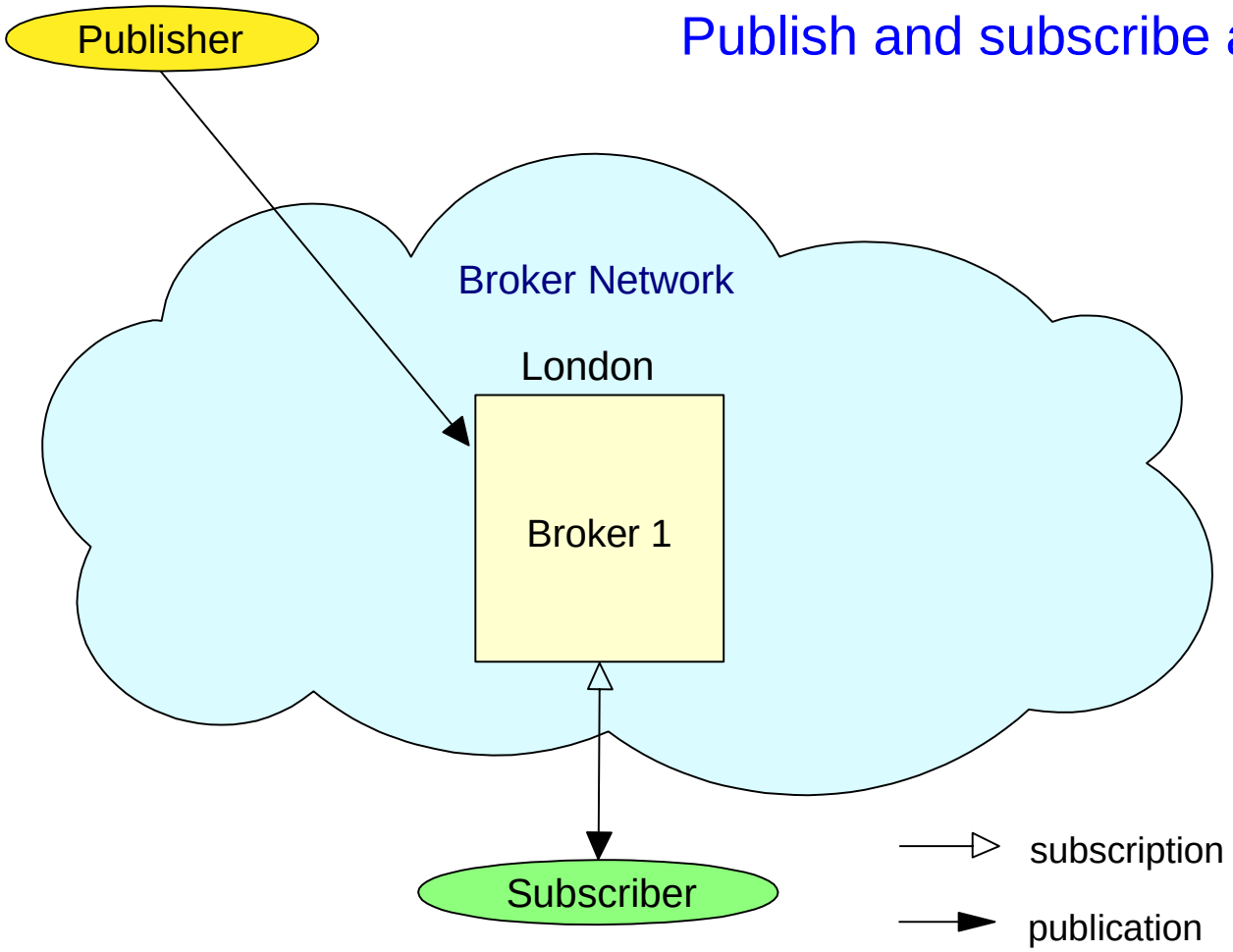
The broker forwards each publication it receives to all subscribers with a subscription which matches the associated topic

Publications and subscriptions

NOTES

- A subscriber specifies which topics it is interested in receiving information about by specifying them in a special message known as a **subscription** which it sends to the broker. A subscriber may send multiple subscriptions to the broker, each of which specifies one or more topics.
- A publisher publishes its information in another special message which it sends to the broker known as a **publication**. Contained within this message will be both the topic being published on and the information which the publisher is providing.
- It is the job of the broker, or broker network if multiple brokers running at different queue managers have been connected together, to ensure that all subscribing applications with matching subscriptions to the topic being published on receive the publisher's message.

Single Broker Network



Single Broker Network

NOTES

- Each queue manager can only play host to a single broker. In this example, the broker network consists of just a single broker, Broker1. The queue manager playing host to this broker is in London.
- The publisher publishes its information to Broker1, it does this by sending a message to one of the broker's queues. The subscriber subscribes at and receives its publications from the same broker. To subscribe it sends one or more messages to another of the broker's queues.
- Soon we will see publishers and subscribers residing at different brokers within a multi-broker network.

Broker queues

Publications are sent to a **stream queue**

- **SYSTEM.BROKER.DEFAULT.STREAM by default**

Subscriptions are sent to the broker's **control queue**

- **SYSTEM.BROKER.CONTROL.QUEUE**

Broker state, e.g. subscriptions, is stored on internal broker queues

- **SYSTEM.BROKER.***

Broker queues

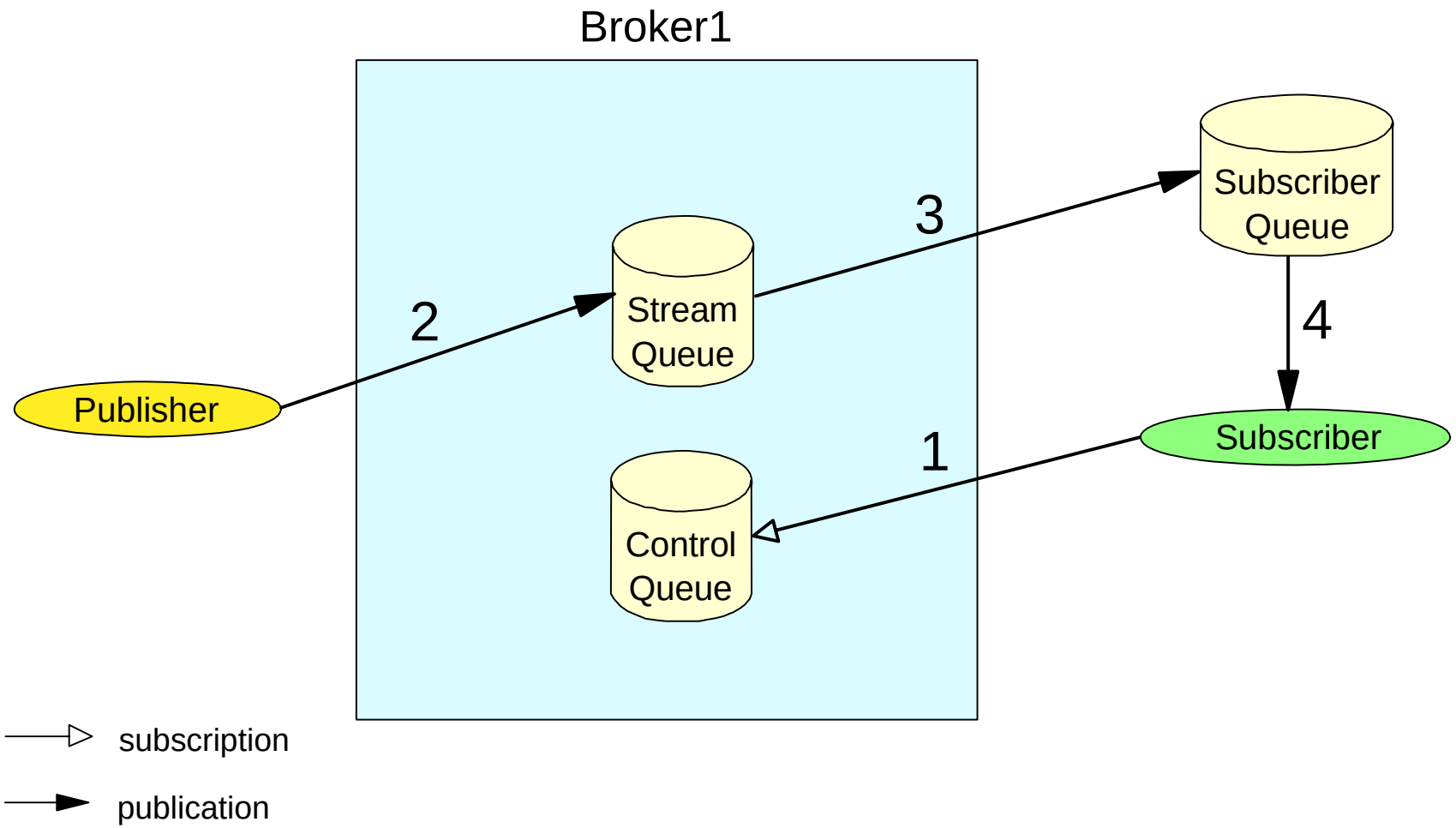
NOTES

- Publishers and subscribers send their message to specific queues which the broker is listening on. Publications and subscriptions are sent to different queues. Publications are sent to one of the stream queues which the broker is listening on. Streams are a method of grouping different kinds of related topics together and are explained later. A default stream queue exists on every broker called SYSTEM.BROKER.DEFAULT.STREAM. Subscriptions are always sent to the same queue, this is called broker's control queue, SYSTEM.BROKER.CONTROL.QUEUE.

- Apart from the queues which the broker reads from, it also uses internal queues to store its persistent state, for example subscriptions. Subscriptions are persistent, meaning that they survive both a broker and a queue manager restart. Subscriptions need to be explicitly deregistered before a subscriber will stop receiving information about a particular topic. As an alternative, subscriptions can have an expiry associated with them and only be active for a nominated period of time.

- All of the standard facilities provided by the MQI are available to publishing and subscribing applications. Messages can be sent to, or received from, the broker both inside and outside of syncpoint. As well as subscription messages having an expiry associated with them, publication messages can also be sent to the broker with an expiry set, the information contained within them may only be relevant for a defined period of time.

Message flows



Message flows

NOTES

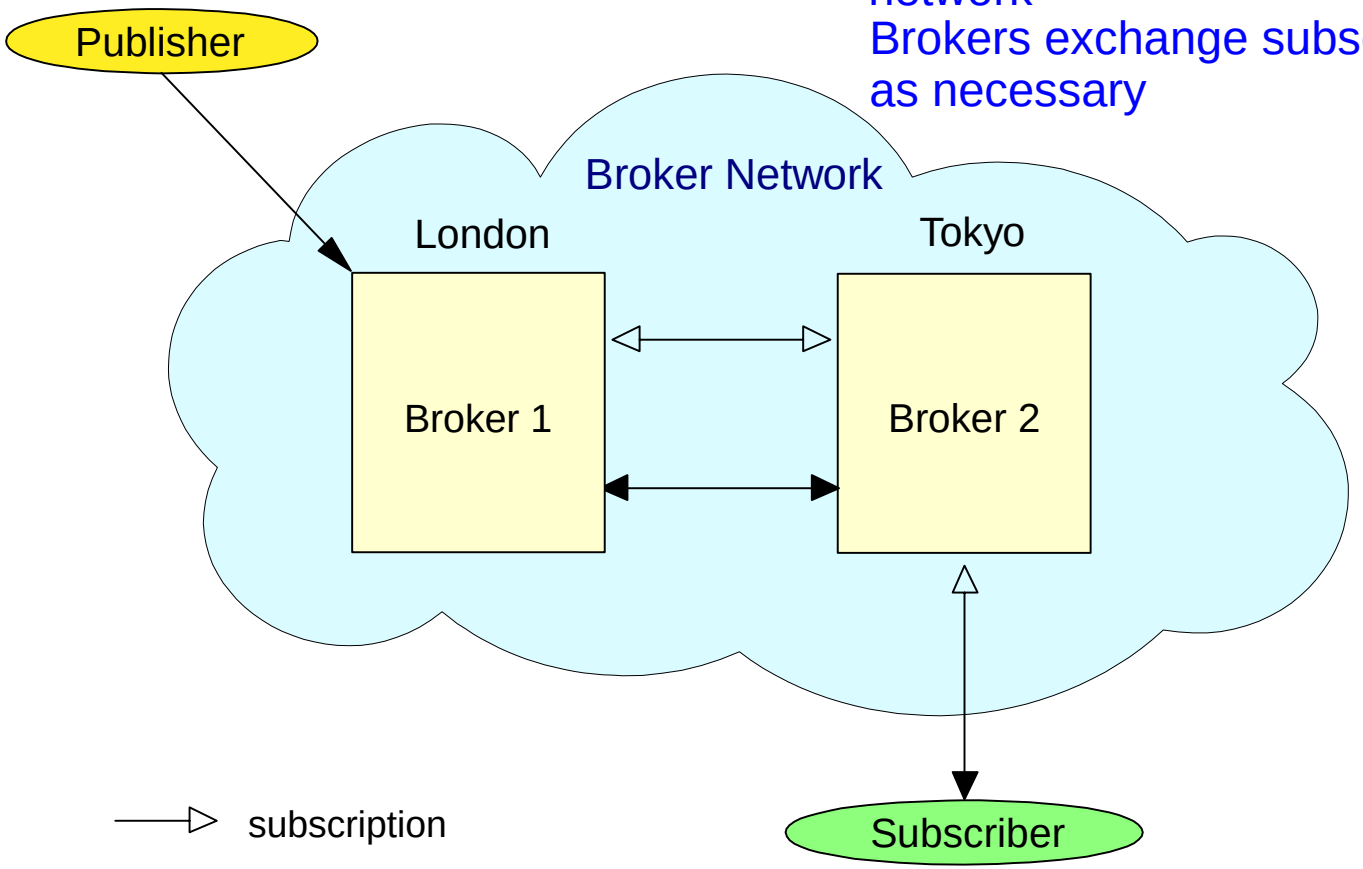
- This foil describes the message flows which occur between the broker, a publisher and a subscriber. In particular it describes how information published by a publisher finds it way to a subscriber via the broker. When a subscriber registers its interest in some information it needs to nominate a queue where it wants its information to be sent.
  1. A subscriber sends a message to the broker's control queue, SYSTEM.BROKER.CONTROL.QUEUE, containing his subscription to some topic, say Topic1. This is read by the broker which remembers details of the subscription such as the queue where the subscriber wants his publications to be sent.
  2. Information is published on the same topic, Topic1, by a publishing application in a message sent to stream queue SYSTEM.BROKER.DEFAULT.STREAM.
  3. The publish message is read by the broker and forwarded to the subscriber's queue.
  4. The subscribing application reads the publication from its subscriber queue.

- As with all MQSeries messaging the flow of information from publisher to subscriber can be asynchronous. If the broker isn't running then the publication will remain on the stream queue until the broker is restarted. If the subscribing application is not active then publications will build up on its subscriber queue.

Multi-Broker Network

Publish and subscribe at any broker within network

Brokers exchange subscriptions and publications as necessary



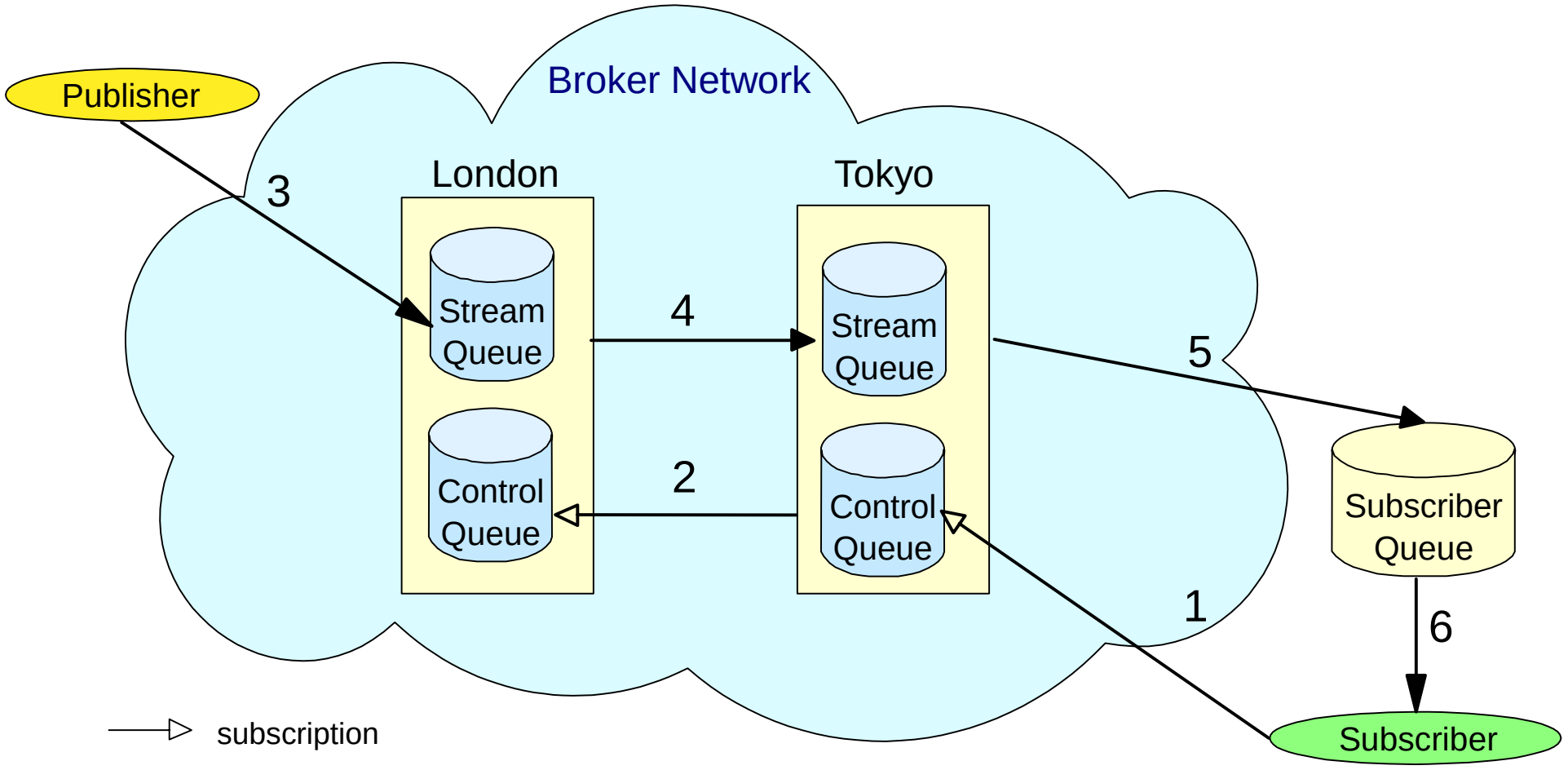
Multi-Broker Network

NOTES

- Consider the same example in a simple multi-broker network in which a further broker, Broker2, running at a queue manager in Tokyo, is added to the network. The publisher still publishes his information in London, but this time the information must find its way to a subscribing application at Broker2 in Tokyo. To achieve this publications and subscriptions are exchanged between the two brokers. In effect the Tokyo broker acts as a subscriber to publications being published at Broker1 in London, and then republishes them to its local subscribers.

- From an MQ perspective messages must be able to flow between the two queue managers concerned. Normally there will be direct channels in both directions between the queue managers.

Message flows



Message flows

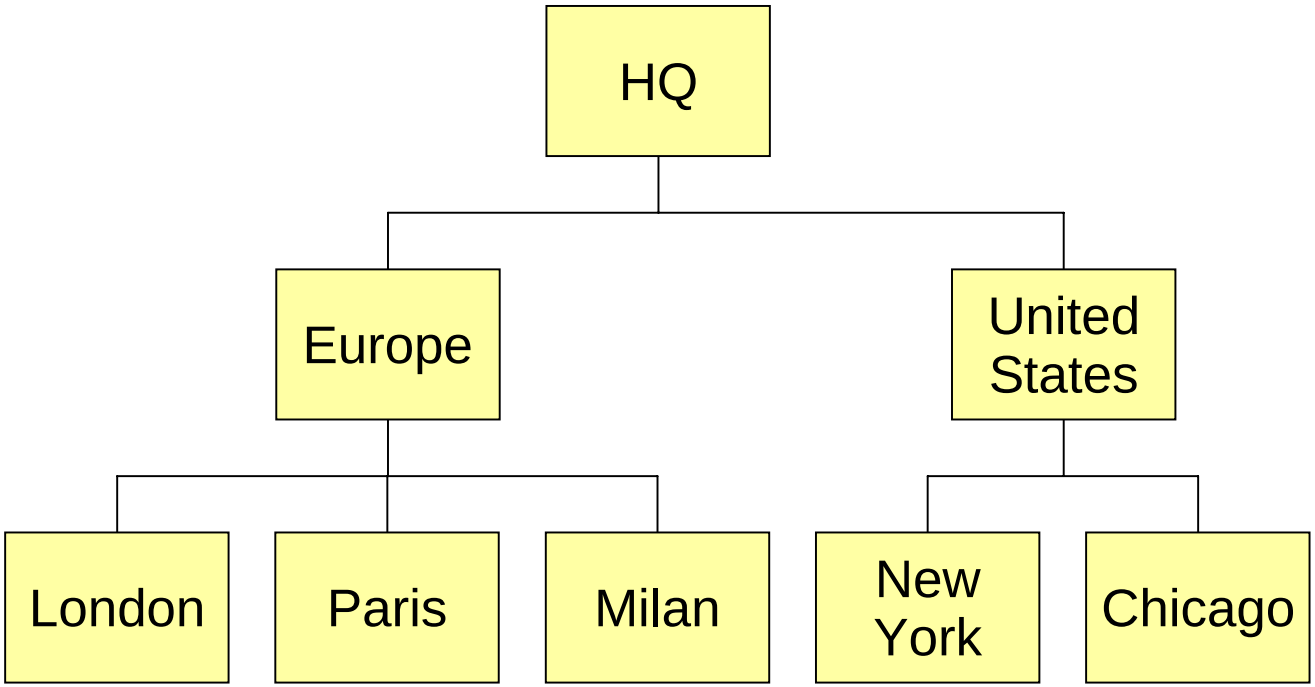
NOTES

- Let us consider the message flows which occur between the broker, a publisher and a subscriber in our simple multi-broker network. This time the subscribing application is subscribing at a different broker to the application which is providing it with its information.
  1. The subscribing application sends his subscription message to Topic1 to the SYSTEM.BROKER.CONTROL.QUEUE of the Tokyo broker.
  2. The Tokyo broker propagates this subscription to Topic1 by sending its own subscription to this topic to the control queue, SYSTEM.BROKER.CONTROL.QUEUE, of the London broker.
  3. Sometime later, a publishing application provides some information about Topic1 in a message sent to the stream queue at London broker.
  4. The London broker notices that it has a single subscription for information about this topic, this being from the Tokyo broker. This causes the publisher's message to be sent to same stream queue at the Tokyo broker.
  5. Tokyo broker reads the publication from its stream queue, it too has a single subscription for information about this topic, this being from the subscribing application. The publisher's message is then forwarded to the subscriber's queue.
  6. The subscribing application in Tokyo gets the message from its subscriber queue.

- Note that brokers consolidate subscriptions, a further subscription to Topic1 at the Tokyo broker from another subscriber would not result in another subscription being lodged at the London broker.

Broker Network

- Arranged in a hierarchy, cycles detected
- Network consists of parent and child brokers
- The parent broker at the top is called the root broker



Broker network

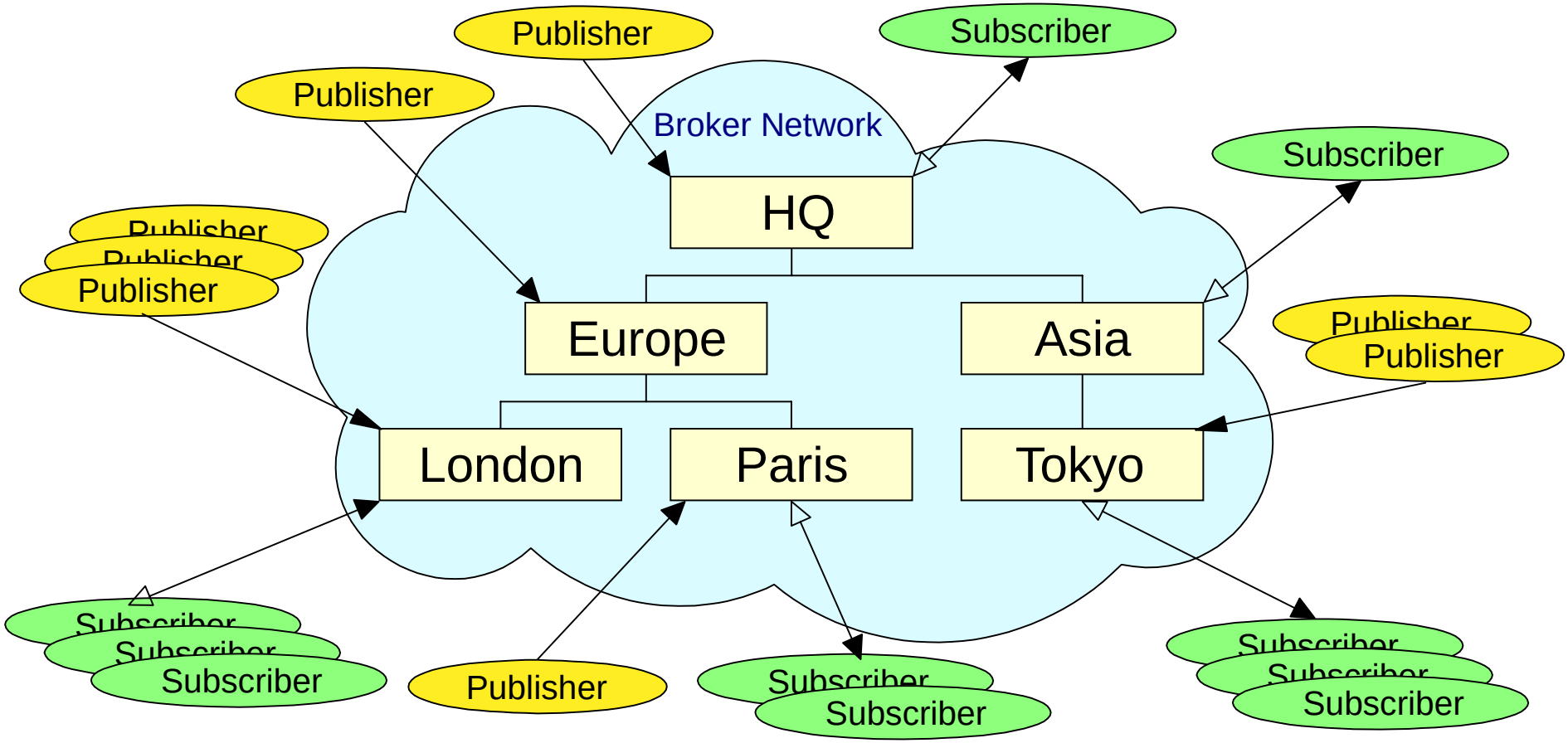
NOTES

- The broker should be thought of as a single entity, however it may actually be built up of multiple brokers. This may be desirable for geograpical reasons, for example to link applications which are based at different locations within an enterprise. It may also be desirable for performance reasons to balance the workload over more than just a single broker. Multiple brokers must be connected in a hierarchy, though this will be hidden from other applications since the publishing and subscribing can be performed anywhere within the network.

- Both subscription and publication traffic take a hierarchic route to their destinations. Subscriptions flow to all nodes in the broker network. A broker consolidates all of the subscriptions it has received. For example, if a broker has received ten subscriptions to a particular topic it will only register a single subscription on behalf of these subscribers to each of its neighboring brokers.

- Whereas subscriptions flow to all nodes in the broker network, publications only flow between brokers when necessary. When a broker receives a publication about a topic a broker will only forward the publication to neighbouring brokers if they are subscribing to that topic on behalf of a subscribing application, either directly or indirectly. Soon we will see how subscriptions and publications flow through a broker network.

Publish anywhere, subscribe anywhere



Publish anywhere, subscribe anywhere

NOTES

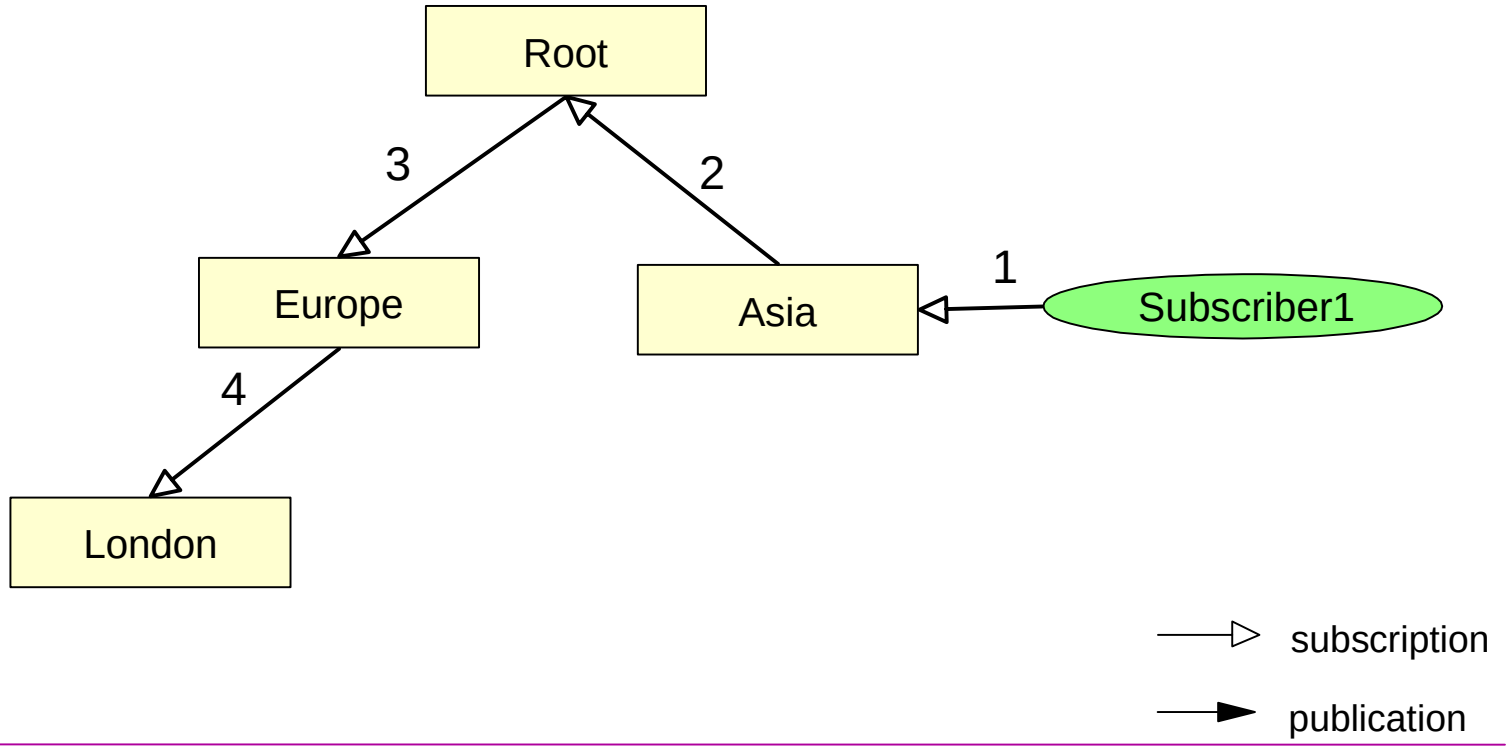
- We have seen how multiple brokers can be connected together to form a broker network. The network topology is a hierarchy, publishing and subscribing applications can reside anywhere within the hierarchy, and MQSeries Publish/Subscribe will ensure that information flows exactly to those applications which need it.

- Publication messages take a hierarchical route from publisher to subscriber. For example, a publication at the Tokyo broker would need to pass through the Asia, HQ and Europe brokers before a subscriber to that information at the London broker would receive it. Note that this is the case even if there was a dedicated channel between the Tokyo and London queue managers.

- From a performance point of view the same rules used to design an MQSeries network also need to be applied to the design of a broker network. In all cases the number of messages, predominantly publication messages, needs to be known when estimating the workload of an individual broker. In general, since publications take a hierarchical route, parent brokers will usually need to cope with a heavier workload than their children. The heaviest workload will usually be at the root broker, the parent broker at the top of the hierarchy.

Subscribing in a multi-broker network

Subscriptions flow to all brokers in the network



Subscribing in a multi-broker network

NOTES

- The next three foils illustrate how subscriptions and publications flow between brokers. In particular they will show how the information published at the Europe broker reaches different subscribing applications at both the Root and Asia brokers. In this first foil, the first subscribing application, Subscriber1, sends his subscription to his local Asia broker. His subscription is then propagated to all of the other brokers in the network.

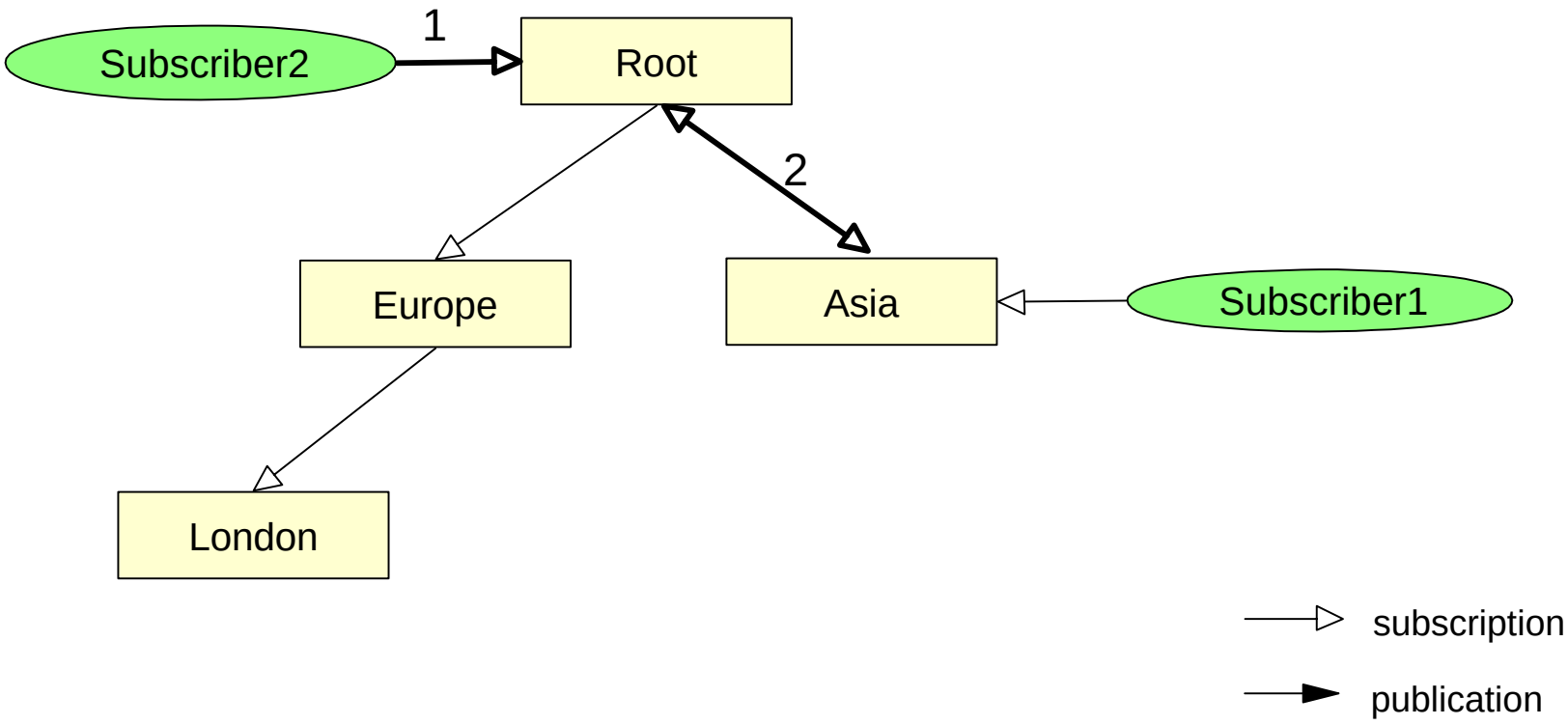
  1. Subscriber1 subscribes to Topic1 at the Asia broker.
  2. This is the first subscription to Topic1 which the Asia broker has received. As a consequence it has no subscriptions to this topic at its neighbouring brokers. The Asia broker lodges a subscription at the Root broker for Topic1.
  3. The Root broker in turn subscribes to Topic1 at the Europe broker.
  4. Finally the Europe broker subscribes to Topic1 at the London broker.

- The subscription has flowed to all brokers in the network. A route now exists such that Subscriber1 can receive information published at any node within the broker network on Topic1.

- Note that subscription propagation around a network is an asynchronous operation, channels or brokers may be down or busy. Until a subscription has reached all brokers within the network the possibility exists that subscribers will not be given publications being published in parts of the network that the subscription has yet to reach.

Further subscription in a multi-broker network

Subscriptions are only propagated when necessary



Further subscription in a multi-broker network

NOTES

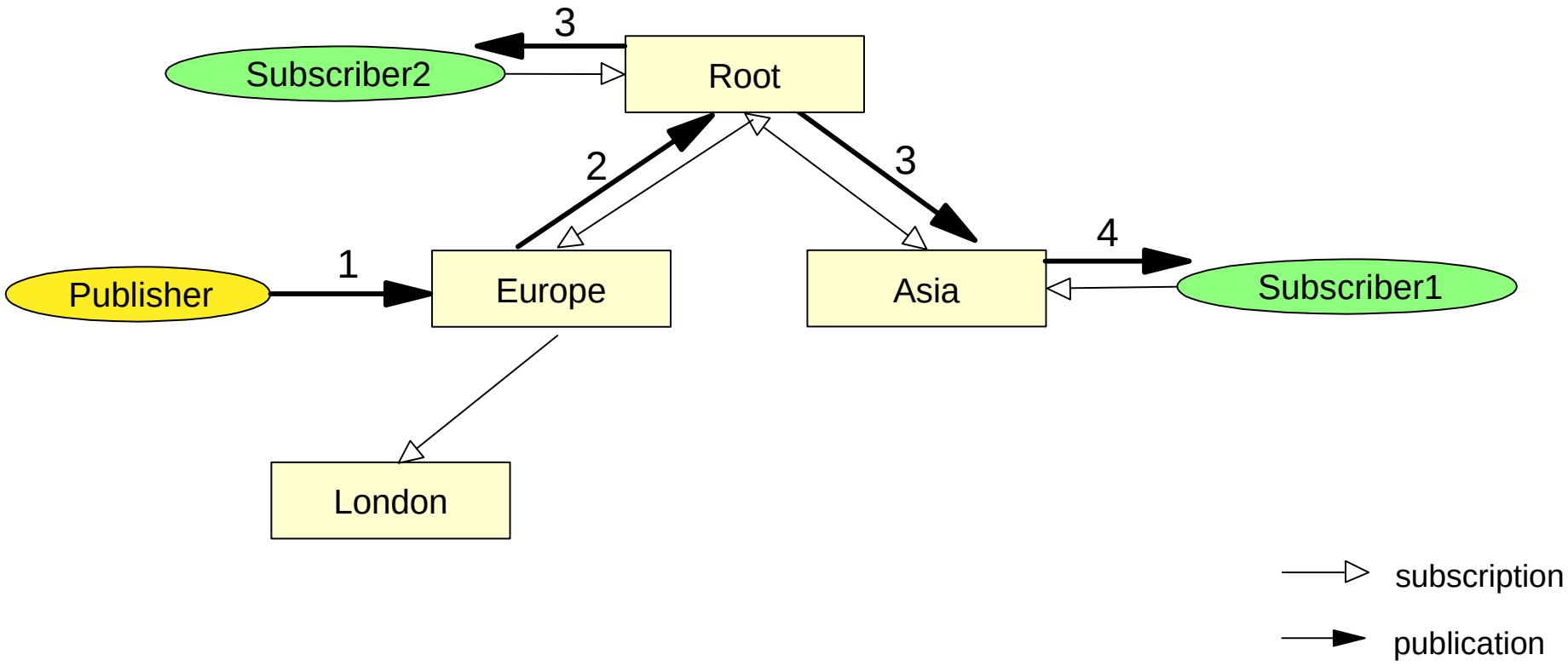
- In this next foil, a further subscribing application, Subscriber2, also registers its interest in receiving information published on the same topic, Topic1. This time the subscribing application is at a different node, the Root broker. Its subscription also needs to be propagated around the broker network, but as we will see, brokers consolidate subscriptions to the same topics, as a consequence, the subscription doesn't need to be propagated down the left sub-tree of the hierarchy.

  1. The additional subscribing application, Subscriber2, sends his subscription to Topic1 to the Root broker.
  2. Subscriptions are only propagated where necessary. The Root broker already has a subscription to the Europe broker, the one which it lodged when Subscriber1's subscription was made. Instead the Root broker just needs to subscribe to publications to Topic1 on the Asia broker.

- Note that the Asia broker was already subscribing to this topic at the Root broker. We know have the situation whereby the Root and Asia broker's are both subscribing to each other's publications on Topic1. This is not the case elsewhere in the network, the Europe and London brokers aren't actually subscribing to the topic. They are merely being subscribed to on this topic.

Publishing in a multi-broker network

Publications are only passed where subscriptions exists



Publishing in a multi-broker network

NOTES

- Now consider what happens when a publisher at the Europe broker publishes information on Topic1. As we will see, the publication isn't just sent to every broker within the broker network. Instead publications only ever flow where subscriptions exist. In the foil, publications only flow in the opposite direction to the subscription arrows.

- Look at the direction the subscription arrows are pointing and try to forget about the word subscription. Instead try to think about who is subscribing to who. For example, the Root broker is subscribing to the Europe broker, the subscription arrow points from the Root broker to the Europe broker. Thinking about subscriptions in this way may help you to understand the diagram.
  1. The publishing application publishes information about Topic1 by sending a message to the Europe broker.
  2. The publication is passed to the Root broker. It doesn't pass to the London broker because that broker is not subscribing to the Europe broker (no subscription exists from London to Europe broker on Topic1)
  3. The Root broker receives the publication. It has two subscriptions to Topic1. From its local subscribing application, Subscriber2, and the Asia broker. The publication is sent to each of these 'subscribers'.
  4. The Asia broker receives the publication from the Root broker and forwards it to its subscribing application, Subscriber1.
- The Asia broker actually has two subscription arrows pointing towards it. The second is from the Root broker, for obvious reasons the publication isn't published again to the Root broker.

Types of publications

- **Events**
  - Continuing succession of logically independent messages, for example:
    - trades
    - customer buying an airline ticket
  - Subscribers receive as available

- **State**
  - Information that is being regularly updated or replaced, for example:
    - stock prices
    - furnace temperatures
  - Brokers can retain copy of last publication
  - Subscribers may receive immediately or check at their own initiative

Types of publications

NOTES

- When a publish/subscribe system is being designed it is important to decide whether the information being published on each topic is either state or event related.

- Event publications are usually independent from one another. They usually indicate that some further action or processing is needed. A subscriber missing an event may be disasterous and generally subscriptions to event publications all need to be in place before any events are published. There may be more than one publisher of event publications for a given topic. Examples of this type of information are:
  - a stock trade
  - a customer buying an airline ticket

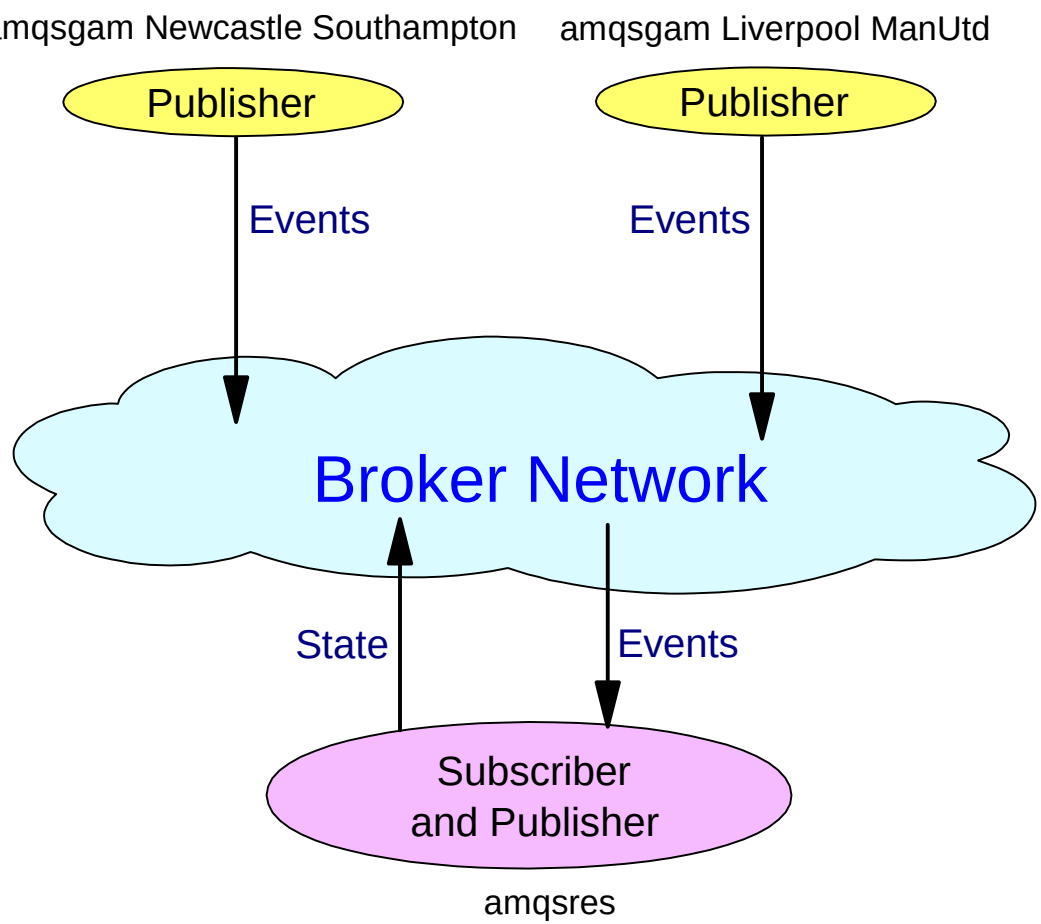
- State publications usually contain information that is being updated at regular intervals. If a subscriber misses a state publication then generally it isn't a problem since an updated view of the state will about to be published again. The broker may also be instructed to retain the last copy of a state publication. This can be sent to new subscribers to that state topic rather than letting them wait for the information to be published again. Usually there is only a single publisher per state topic. Examples of this type of information are:
  - a stock price
  - the temperature of a furnace

"Results Service" sample programs

Match Simulators:

Events:

Match started
Goal scored
Match Ended



Results Service:

State:

Latest scores

"Results Service" sample programs

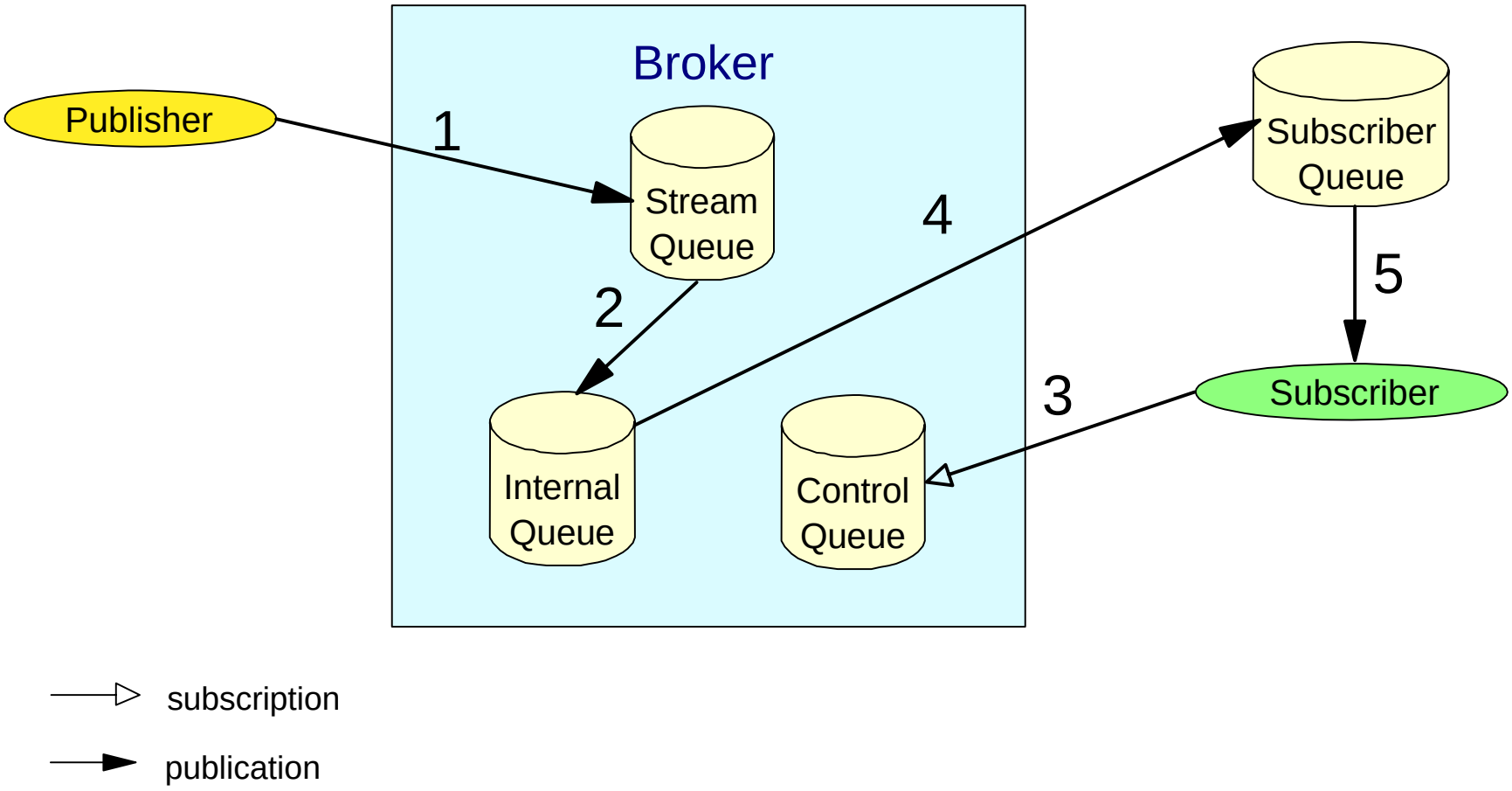
NOTES

- Two sample programs are provided. Together these form a results gathering service for sport events, in this case soccer matches. One of the sample programs, **amqsgam**, simulates a soccer match. It does this by publishing **event** publications to the broker, when the soccer match starts; when a goal has been scored in the match; and when the match has ended. There can be multiple matches, and consequently multiple instances of amqsgam, all running simultaneously. With event information it is usual for there to be more than one publisher on the same topics. Information such as which team has scored is published as data. The topics used by amqsgam are:
  - Sport/Soccer/Event/MatchStarted
  - Sport/Soccer/Event/ScoreUpdate
  - Sport/Soccer/Event/MatchEnded

- Another sample program, **amqsres**, acts a a Results Service for all soccer matches being played within the broker network. It subscribes with a wildcard topic to receive all event publications published by the match simulators. It uses them to keep track of the latest scores in all soccer matches currently being played. The same program publishes the latest scores in each of these matches as a series of **state** publications. For state information there is usually only one publisher on each topic, for the example teams in the foil the following topics would be used:
  - Sport/Soccer/State/LatestScore/Newcastle Southampton
  - Sport/Soccer/State/LatestScore/Liverpool ManUtd

Retained publications

Subscribers can be given a copy of a state publication when they first subscribe



Retained publications

NOTES

- In the previous foil we described the two different types of information. We also said that one of these types, state-based information, was suitable for being retained by an MQSeries Publish/Subscribe broker. In this foil we describe the message flows which occur between the broker, publisher and subscriber applications when a subscriber is sent a copy of the last publication to be published on a particular topic.
  1. The publication is published **before** the subscriber has made his subscription.
  2. The publisher also asks the broker to retain the publication which it does on one of its internal queues.
  3. The subscriber sends a subscription to that topic to the broker's control queue.
  4. The broker retrieves the retained publication from its internal queue and forwards it onto the subscriber's queue.
  5. The subscriber receives the publication from the queue.
- The advantage of using retained publication is that new subscribers can receive the latest information about a topic as soon as they subscribe. The alternative would be that the new subscriber would need to wait for the publisher to publish the information again.
- Broker only retain a single publication per topic. A new retained publication will replace an existing one.

Streams

Related publications (and topics) can be grouped into a **stream**

Can be thought of as a high-level qualifier for the topic

- separate name space
- wildcards do NOT span streams

Each stream has a separate stream queue

A given stream does not have to be supported by every broker within a broker hierarchy, though some are:

- SYSTEM.BROKER.DEFAULT.STREAM
- SYSTEM.BROKER.ADMIN.STREAM

Streams

NOTES

- A stream queue is the queue which publishing applications send their publication messages to. We now describe what a stream is, and give some examples of when their use may be appropriate. Brokers all support a default stream, SYSTEM.BROKER.DEFAULT.STREAM, which could be used by all applications, without the need for additional streams.

- A stream can be thought of as a group of related topics. For example, all topics being used to provide applications with the latest stock prices could be thought of as a separate stream from those topics being used by a service which provides information about the weather in various parts of the world.

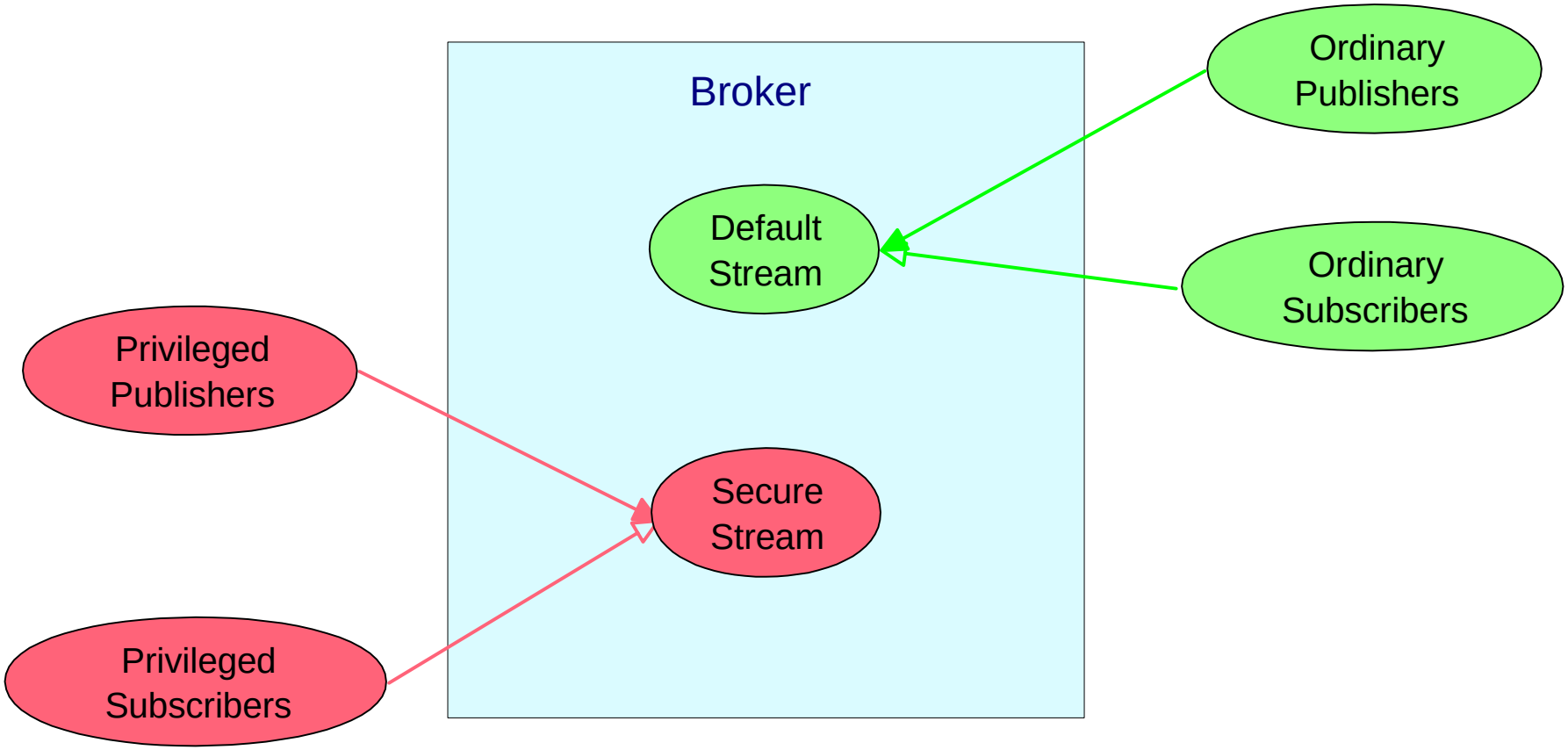
- Streams partition information into separate groups, however, the grouping could equally be performed without using separate streams but using different topic prefixes instead. For example, the default stream, SYSTEM.BROKER.DEFAULT.STREAM could consist of information about:
  - stock prices using topics 'Stock/Prices/....'
  - and weather info using topics 'Weather/Temperature/....'

  Equally the information could have been grouped into separate streams:
  - a stream called STOCK.STREAM using topics 'Prices/.....'
  - and a stream called WEATHER.STREAM using topics 'Temperature/.....'

- In that respect a stream could be thought of as the high-level qualifier for each topic. The stream to which a publication message belongs is implicit from stream queue which it was sent to. If we had a separate stock stream then a queue called STOCK.STREAM would need to be created and publishing applications would need to send their messages to this queue instead.

Use - restricting access

Publish and subscribe authority to sensitive information can be restricted

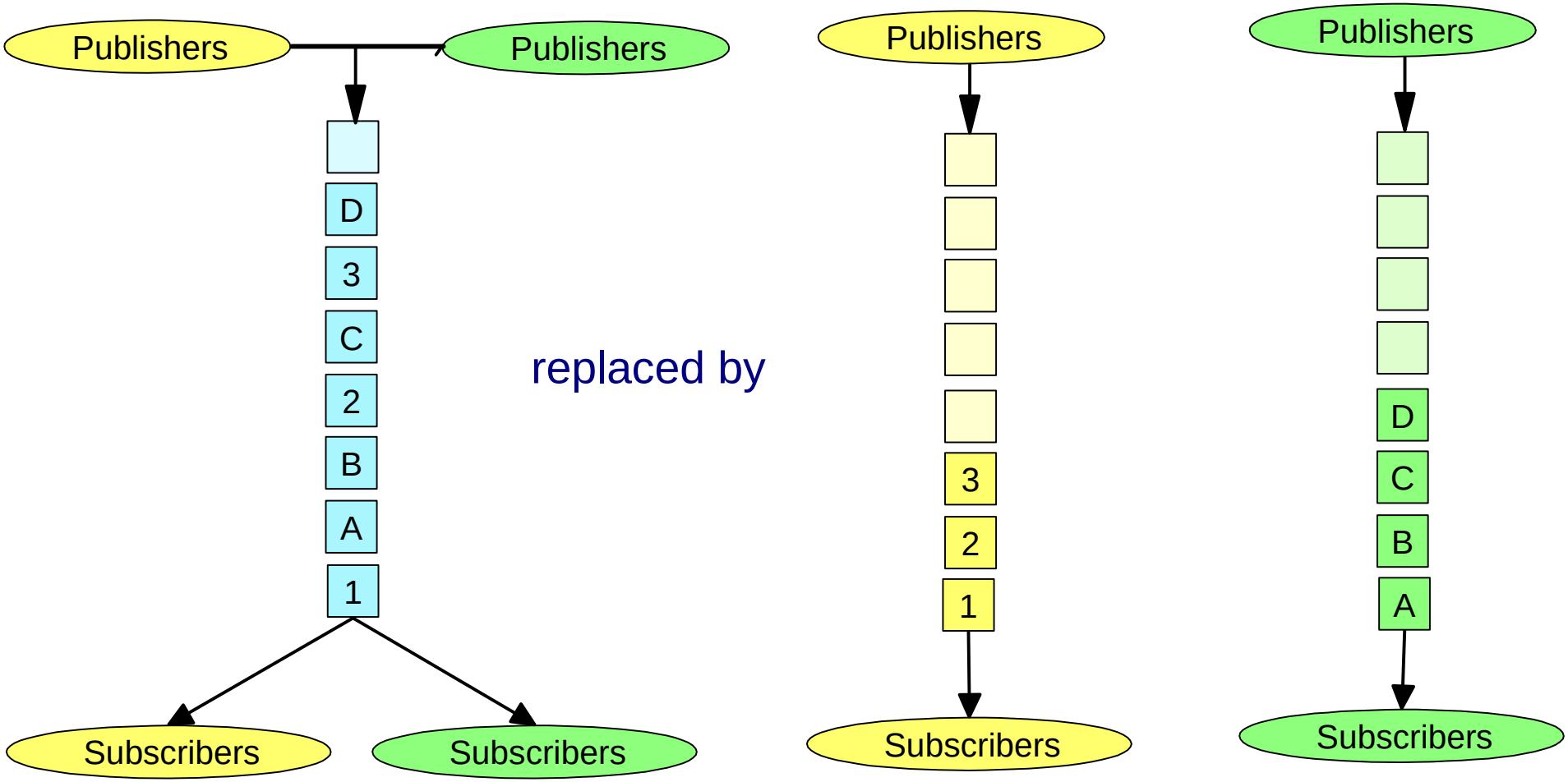


Use - restricting access

NOTES

- The next few foils describe situations where it might be appropriate to use more that just the default stream. The first possible use for splitting some information into a separate stream would be to restrict access to it. Authority checks performed by the MQSeries Publish/Subscribe broker are performed on the stream, or stream queue, and not on individual topics. A subscriber with authority to receive information about one topic within a stream has authority to receive information about all other topics in that stream. If some information is of a sensitive nature then it may be advisable to remove it from the default stream and put it into a new stream.
- Underlying queue manager authority is used to restrict both publisher and subscriber access to information on each stream. A user has authority to publish information for a stream if he has the necessary queue manager authority to put messages to the stream queue of that stream. A subscriber is allowed to receive information being published on a stream if he has browse authority for the stream queue.
- In the example shown, a new stream, and stream queue SECURE.STREAM, has been created. The administrator will then be able to restrict who is allowed to send messages to that queue, and who is allowed to receive messages from the queue, using the MQSeries setmqaut command.
- All authority checks are performed at the broker which the application publishes or subscribes, not at the brokers to which the publication or subscription may be propagated

Use - parallel publishing



Use - parallel publishing

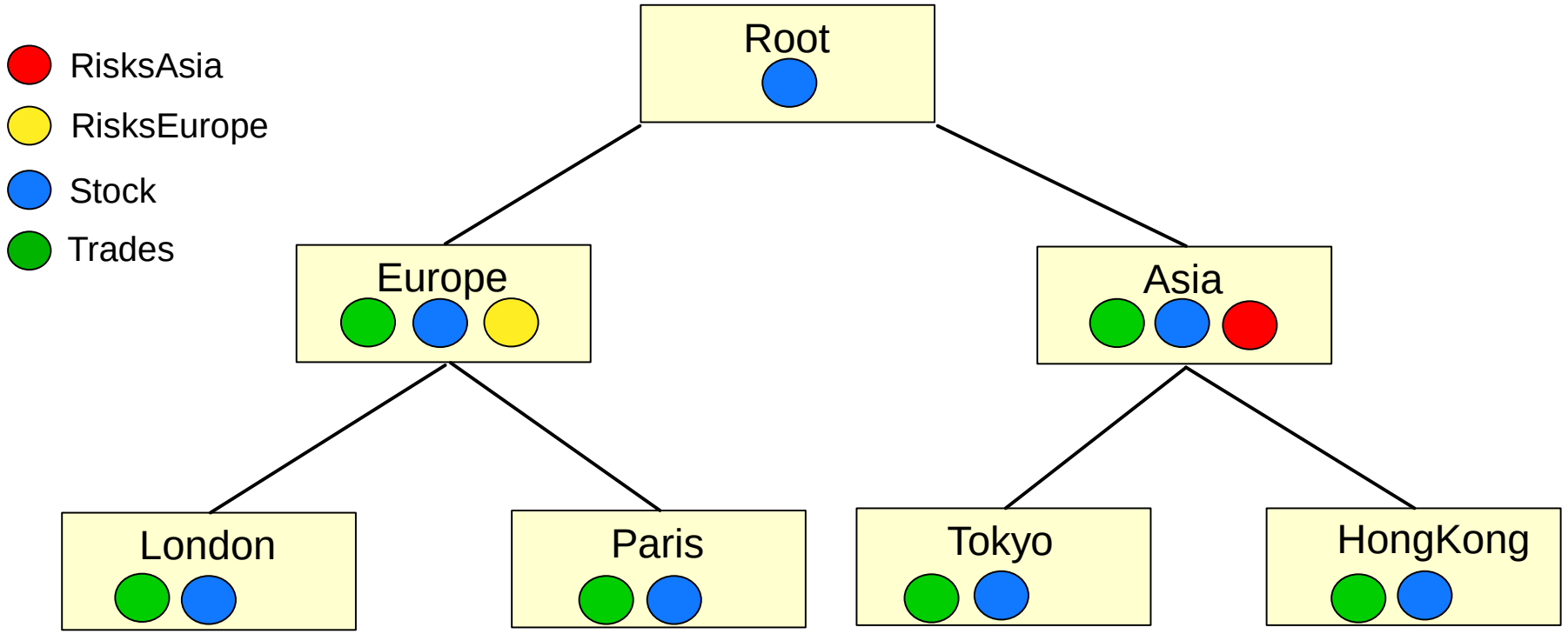
NOTES

- Each broker will support a number of streams, and for each of these streams the broker will schedule a separate thread to process publications being published on each stream queue. If publication messages are arriving on the default stream at a faster rate then they can be processed then it may be appropriate to balance the workload over two streams instead. Publications can be processed in parallel if they are in different streams.
- Publications arriving on a stream queue are processed by the broker in order. Splitting a stream into two separate streams is only applicable if the information being published on each of the new streams is unrelated and consequently the order of arrival of both sets of information at a subscribing application is also independent. If a subscribing application is subscribing to information being published on more than one stream, and it needs to process that information in the same order which it was published, then the safest method of ensuring this would be to publish it on the same stream.
- Stream queues are like any other MQSeries queue, and can be ordered by priority instead of FIFO, this would allow certain publications to take precedence over others if these publications had different performance goals and needed to be published immediately upon arrival.
- In the example shown in the foil, a single stream is being used to publish two types of information, letters and numbers. This is then split into two separate streams which can be processed in parallel by the broker.

Use - to restrict propagation of subscriptions

Each broker can be configured to support only certain streams

This can be used to create a separate stream hierarchy on top of the underlying broker hierarchy



Use - to restrict propagation of subscriptions

NOTES

- We have seen how subscriptions are propagated to all brokers within a broker network. This allows a subscribing application to subscribe to information published anywhere within the broker network. In a large broker network this may be inefficient, especially if information for these topics is known to be published at fixed locations within the network. Streams can also be used to restrict the propagation of subscriptions to only the necessary parts of the network. A broker will only propagate a subscription to a neighboring broker if that broker also supports the stream in question. In that sense each broker hierarchy can be thought of as having one or more stream hierarchies superimposed on top of it. All brokers support the default stream so subscriptions for information being published on this stream will be always be propagated to every broker within the network.

- The foil has a number of streams, not all are supported by every broker. As a consequence both subscription and information flow is restricted. The STOCK stream is supported by every broker within the hierarchy. Stock prices from all markets within the world are available to subscribers everywhere. The TRADES stream has information about all stock trades. The Root broker doesn't support it, allowing information about the trades in each continent to be separate. The Root broker doesn't support this stream, any subscriptions made within Europe don't flow to the brokers within Asia and vice-versa.

- Lastly two separate streams provide risk investment information based upon information published on the TRADES stream. These are supported by only a single broker within the network. The RISKSASIA stream by the Asia broker and RISKSEUROPE stream by the Europe broker. Since no neighboring brokers support these streams, this information is only available at these brokers.

Command messages

Publisher commands:

- Publish
- DeletePublication
- RegisterPublisher
- DeregisterPublisher

Subscriber commands:

- RegisterSubscriber
- DeregisterSubscriber
- RequestUpdate

Command messages

NOTES

- Publishing and subscribing applications send messages to the broker, they can optionally request the broker to send back a reply message indicating the success or failure of their command message. Command messages available are:

Publish

  - A publishing application uses this command message to provide new information about a topic

DeletePublication

  - A publishing application can request the broker to retain its publications. This command message can be used to remove a retained publication.

RegisterPublisher

  - Publishing applications can identify themselves as being a provider of information on specific topics. Among other things this makes them visible to administration applications.

DeregisterPublisher

  - Used by a publishing application when it is no longer providing information on specific topics

RegisterSubscriber

  - Used by a subscribing application to send subscriptions to specific topics

DeregisterSubscriber

  - Unless a subscription has an expiry associated with it, a subscribing application will continue to receive information about that topic until it uses this message to remove its subscription.

RequestUpdate

  - Used by a subscribing application to request a retained publication.

MQRFH Format

Command messages are in MQRFH format

- Rules and Formatting header

Header contains command string in the form of name/value pairs

In a publish command message information follows header in format chosen by publisher

- User-defined format or built-in (e.g. MQFMT_STRING)
- DBCS publication data supported

MQRFH Format

NOTES

- Publishing and subscribing applications send their messages to the broker in MQRFH format. This is a variable length header structure which publishing applications prefix to their publication data. Other command messages don't need to send associated publication data and merely consist of the header structure itself.

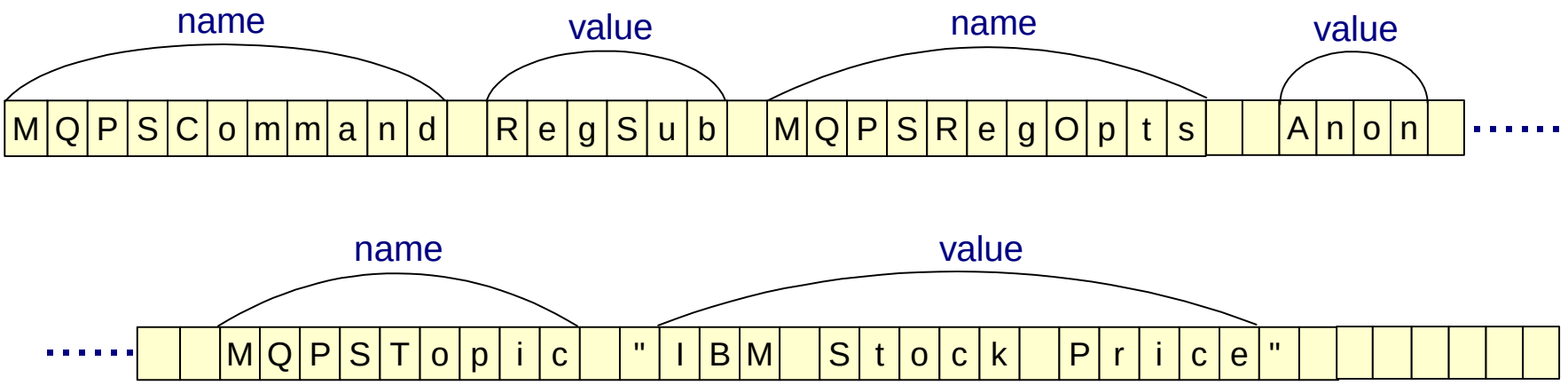
```
typedef struct tagMQRFH {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    StrucLength;      /* Total length of MQRFH */
  MQLONG    Encoding;         /* Data encoding */
  MQLONG    CodedCharSetId;   /* Coded character-set identifier */
  MQCHAR8   Format;           /* Format name */
  MQLONG    Flags;            /* Flags */
} MQRFH;
```

- The header also includes a variable-length NameValueString which isn't included within the structure definition. Within this string publishing and subscribing applications specify the command which they want the broker to perform. The StrucLength field in the header defines the length of the header structure inclusive of the variable length NameValueString at the end of the structure. The string can be null-terminated or blank-padded to the specified length.

- The Encoding, CodedCharSetId and Format fields within the header describe the structure of any (publication) data which follows the header in the message.

Example command string

Register Subscriber : To information about the IBM stock price



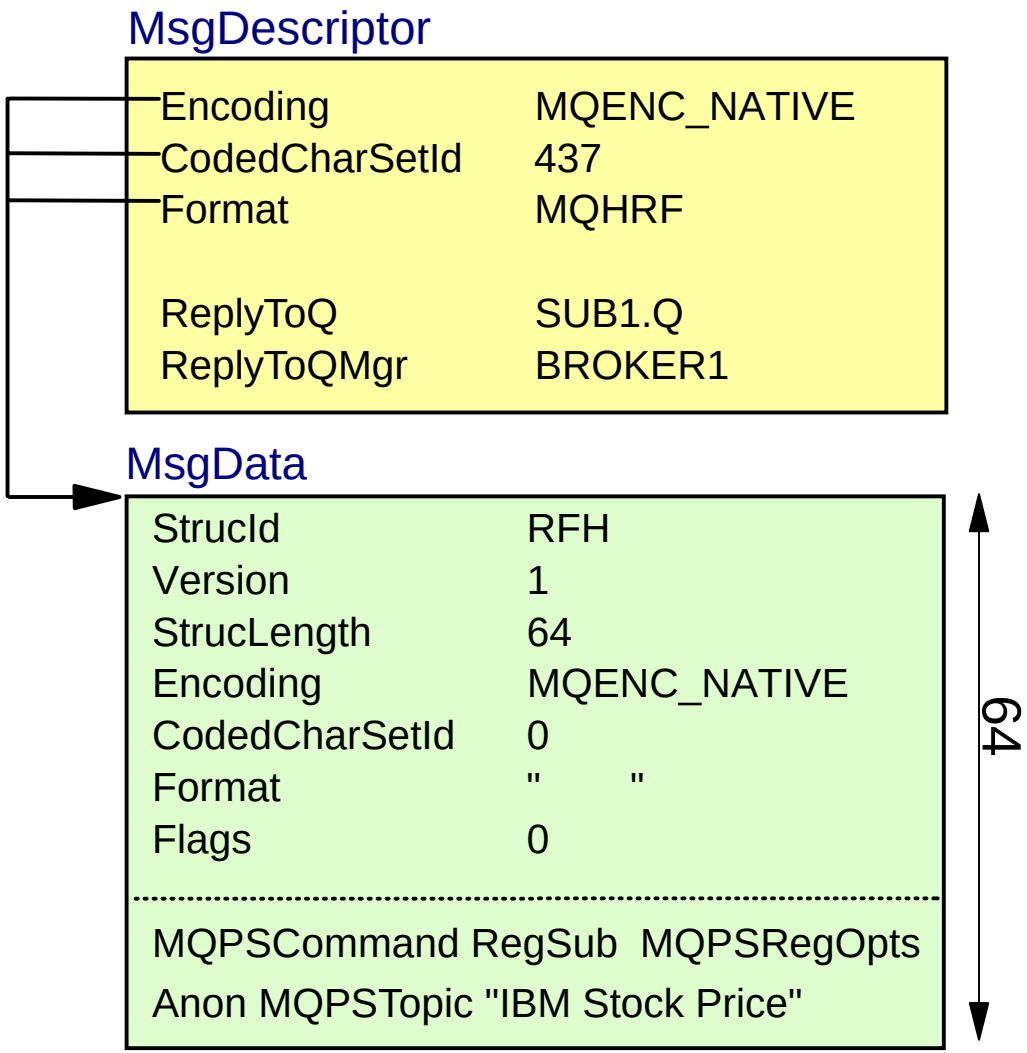
- MQPS parameter names
- Quoted topic value since contains significant blanks
- All blanks between name/values ignored

Example name/value string

NOTES

- The next few foils show the structure of some example messages in MQRFH format. This foil illustrates the format of a command string which a subscribing application might use to subscribe to information on a specfic topic, in this case the IBM stock price. The command string is part of the MQRFH header structure and consists of a number of parameter names and values separated by blanks.
- The parameters all have names which start with the prefix, MQPS, for example MQPSTopic. The first parameter must specify the command, MQPSCommand, which the application requires the broker to perform. Each parameter is followed by its corresponding value, in this case the value is RegSub, which indicates that this is a RegisterSubscriber command.
- Names and values can be separated from one another by any number of blank characters.
- Note that the topic string includes significant blank characters and thus has been enclosed within a pair of double quotes. These double quotes aren't part of the value of MQPSTopic.
- Apart from parameters which specify the command and topic, a further parameter, MQPSRegOpts, has been included for the sake of this example only. This is a subscription option, its value 'Anon' means that this is an anonymous subscription.

Example register subscriber message



- MsgData consists of header structure only
- MsgDescriptor describes encoding/ccsid of MQRFH
- No data following header so encoding,ccsid, and format fields in RFH not set
- Subscriber nominates his subscriber queue to be the same as his reply queue

Example register subscriber message

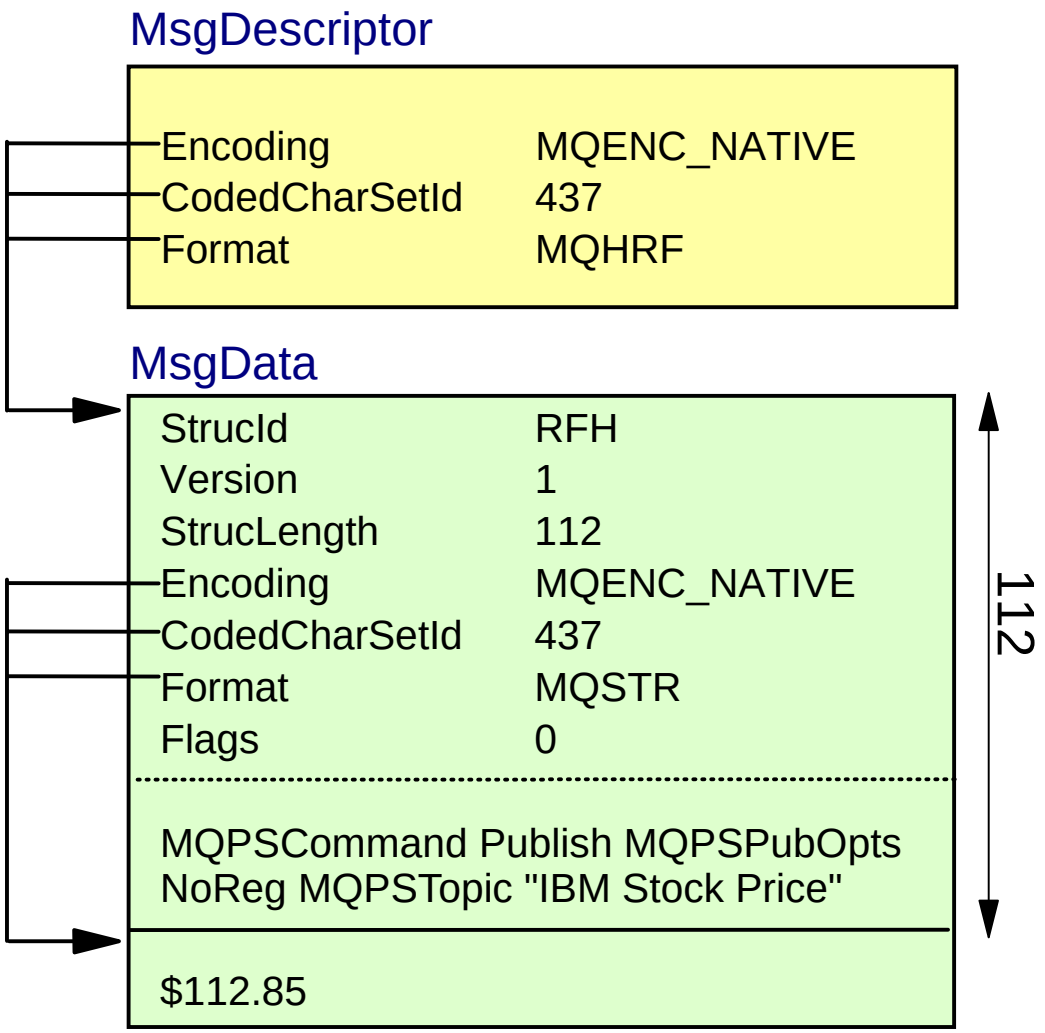
NOTES

- The Format field in the MsgDescriptor needs to be set to MQFMT_RF_HEADER, indicating that the message is in MQRFH format. The Encoding and CCSID fields in the MsgDescriptor describe the numeric and character data within the header. The header itself also has Format, Encoding and CCSID fields. These are used to describe the format of any data which follows the header structure in the message data. In the case of a RegisterSubscriber message there is no following data and these header fields needn't be set by the subscribing application.

- The example shows the structure of a message which could be used by a subscriber to register a subscription to the IBM stock price. The sending application needs to set the StrucLength field within the header to indicate the length of the header structure. This needs to take into account the length of the variable length NameValueString which contains the command string. Since the message contains no following data the length of the message data is the same the length of the header and would also be specified as BufferLength on the subscriber's MQPUT call.

- Further parameters could have also been added to the command string to indicate the name of the queue which the subscriber wants to receive publications about the IBM stock price. Instead these have been left to default to the ReplyToQ and ReplyToQMgr fields within the MsgDescriptor. It is quite common for a subscriber queue to be the same queue that an application is using for reply messages from the broker.

Example publish message



- Header preceeds publication in MsgData
- MsgDescriptor describes encoding/ccsid of MQRFH
- Fields in RFH describe format,encoding,ccsid of publication data
- StrucLength of RFH includes name/value string

Example publish message

NOTES

- We have seen the format of a message which a subscribing application could have used to subscribe to information about the IBM stock price. This example shows the format of the message which a publishing application could use to provide that information. This message is also in MQRFH format, but in this case the header is followed by the data which the publisher is associating with its topic. In this case the publisher is publishing the stock price as string data and this follows immediately after the end of the header. The broker sends this exact message to the subscriber queue nominated by the subscribing application.
- The publisher needs to specify a different command string to the subscriber, firstly it needs to indicate that this is a Publish command message.
- Whereas the subscriber didn't need to set the Format, Encoding and CCSID fields in the header, a publishing application needs to indicate to the subscribing applications what format his data is in. The publisher's message may also need to be converted if the subscribing application is running on a different platform to the publisher.
- In this example the length of the RFH header is 112 bytes long. This isn't the length of the actual message since this is longer by the length of the string which is being used to publish the stock price.

Control commands

strmqbrk	Start/create a broker
dspmqbrk	Display broker status
endmqbrk	End a broker
dltmqbrk	Delete a broker
clrmqbrk	Tidy up a broken hierarchy

Control commands

NOTES

- The following commands are available to control a single MQSeries Publish/Subscribe broker:

  strmqbrk
  - normally used to start the broker, but will also create a broker if one doesn't exist on that queue manager.
  - this is also the command which is used to create broker hierarchies, an optional parameter can be specfied which nominates the broker's parent

  dspmqbrk
  - displays the status of the broker

  endmqbrk
  - ends the broker

  dltmqbrk
  - deletes a broker

  clrmqbrk
  - is normally used only in exceptional circumstances to tidy up or make changes to a broken hierarchy

System programming

The Admin stream

- Which streams does a broker support ?
- Who is a broker's parent and children ?
- Broker events

Metatopics

- Who is subscribing and to what topics on each stream ?
- Who is publishing and on what topics on each stream ?

Sample administration application

- Subscribes to broker publications to display administrative information

Broker exit

- Invoked at point broker forwards publication to subscriber

System programming

NOTES

- For the administrator or system programmer MQSeries Publish/Subscribe provides:
  - A special administation stream called SYSTEM.BROKER.ADMIN.STREAM which the broker itself publishes information on. This contains various topics which can be subscribed to by an administration application to:
    - find out who that brokers parent is
    - find out who that brokers children are
    - listen for event messages to be published
  - Each stream also supports a special set of topics known as meta-topics, again these can be subscribed to find out information such as:
    - who is subscribing and to what topics
    - who is publishing information and on what topics
  - A sample administration application, amqspsd, is provided in both source and executable form which subscribes to and displays the above mentioned broker information
  - The broker also has an exit. When the exit is configured it will be invoked just before the broker is about to send a publication message to a subscriber