MQ Publish/Subscribe Topics - *objects, nodes, strings?*

David Ware

Agenda

- Publish/Subscribe in WebSphere MQ
- Administration of publish/subscribe
- Management of publish/subscribe
- Subscriptions and publications
- Topologies

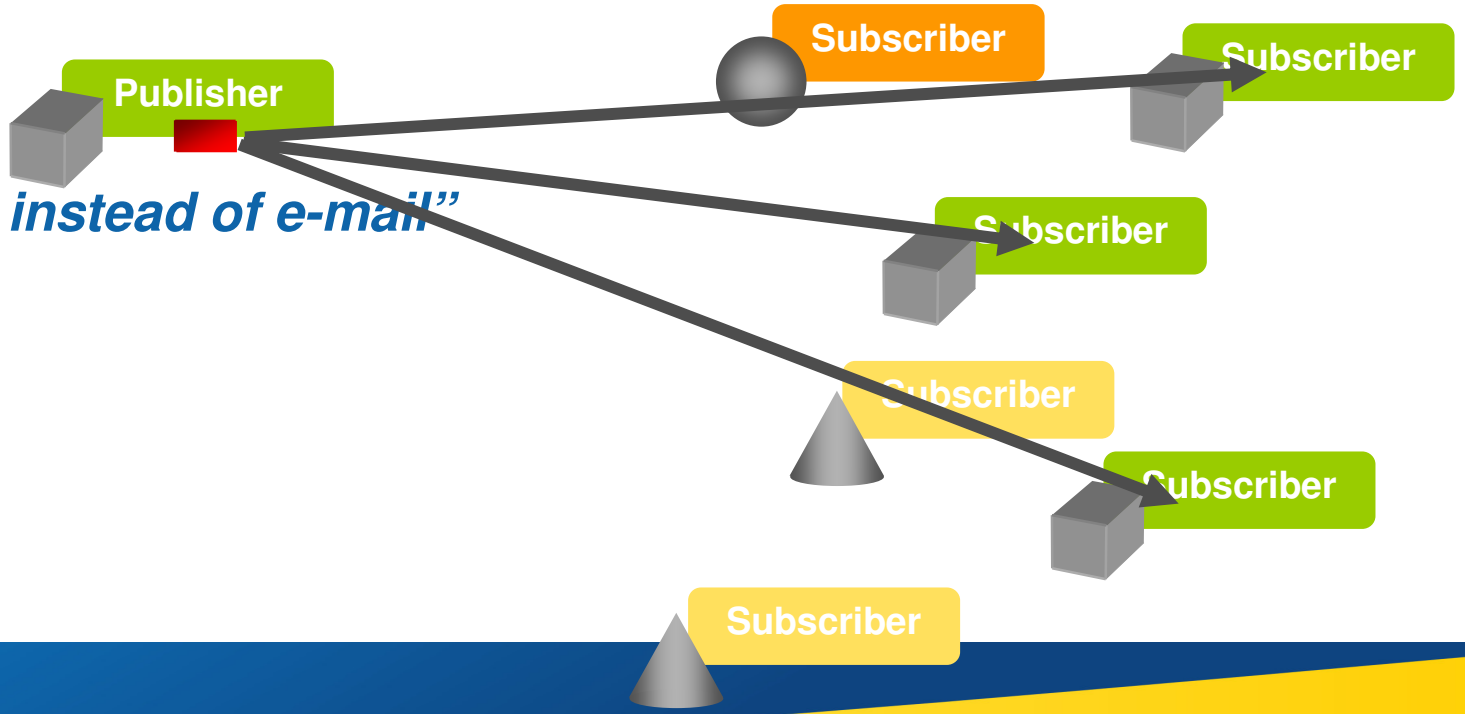
What is publish/subscribe?

- In words (Wikipedia® has the answer)…

> *"Publish–subscribe is a messaging pattern where senders of messages, called **publishers**, do not program the messages to be sent directly to specific receivers, called **subscribers**. Instead, published messages are characterized into **classes***, without knowledge of what, if any, subscribers there may be. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of what, if any, publishers there are."*

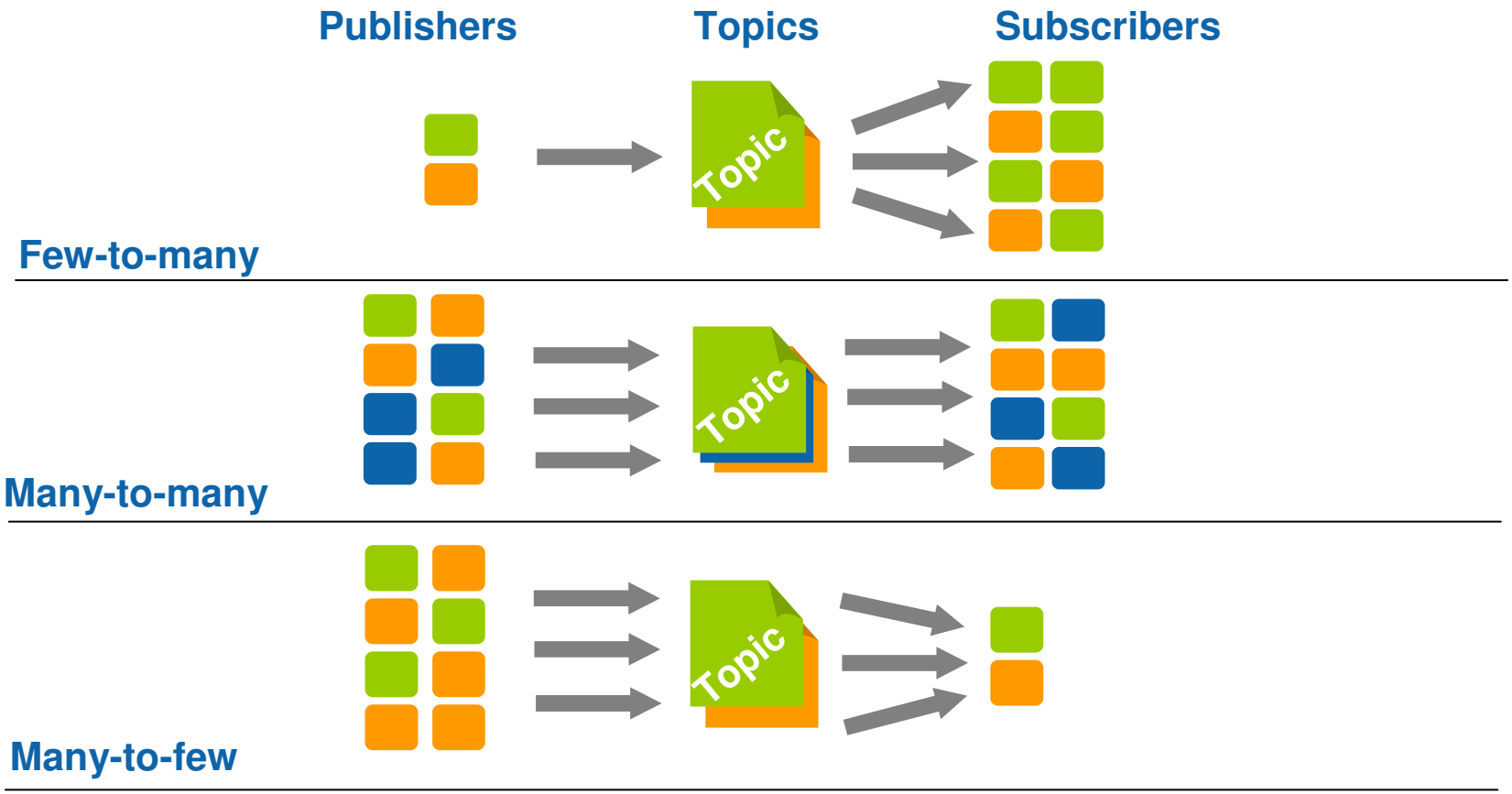
*Change "**classes**" to "**topics**" in *MQ-speak* and that's a pretty good description.

- Or in pictures…



"It's like Twitter, instead of e-mail"

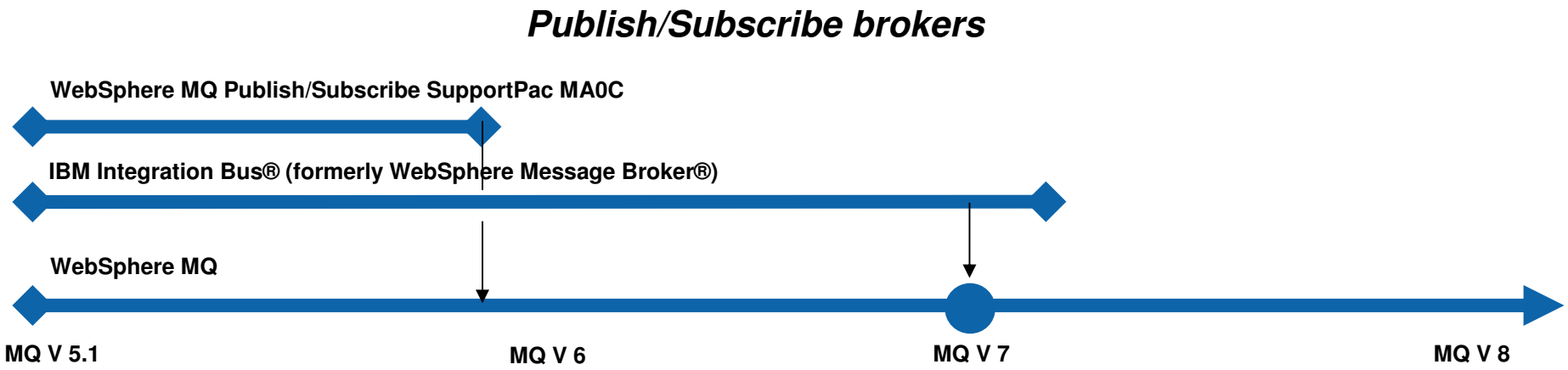
Publish/Subscribe, many patterns

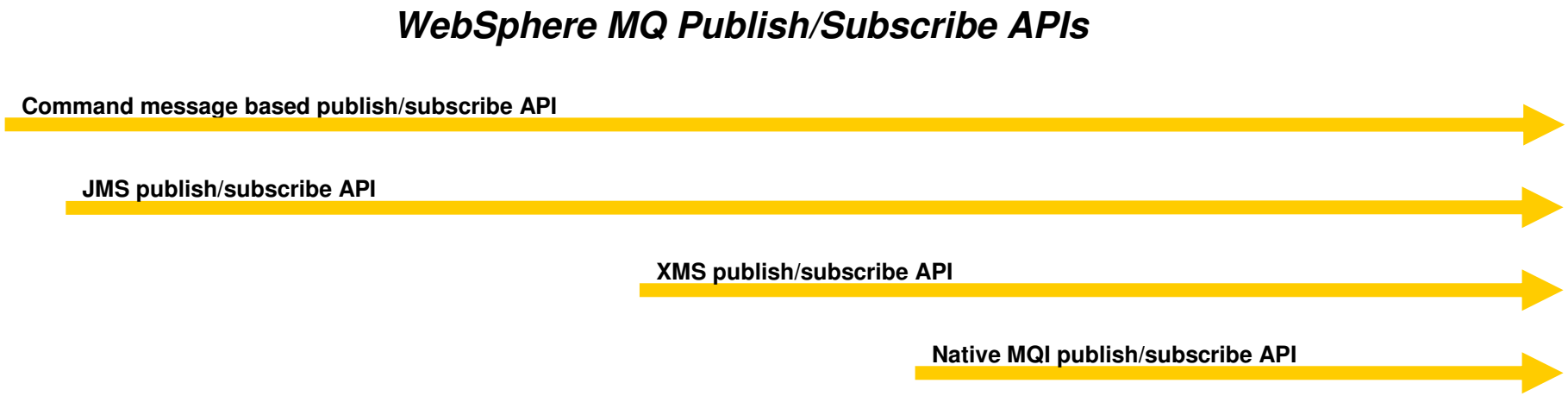


Static or Dynamic?

Publish/Subscribe in WebSphere MQ

WebSphere MQ's publish/subscribe over the years





Terminology in WebSphere MQ

MQ's use of publish/subscribe terminology:

- **Topic**
  This can mean ***topic strings***, ***topic objects*** and even ***topic nodes***, not to mention ***JMS topics***!

- **Publisher**
  A publisher is an application that puts messages to a topic as opposed to a queue.
  - A publisher can either open a topic object or a topic string.

- **Publication**
  A publication is simply a message that is put to a topic rather than a queue

- **Subscription**
  A subscription is an artefact that exists on a queue manager that describes where copies of messages published for a particular topic string are delivered to
  - A subscription identifies a queue for those messages

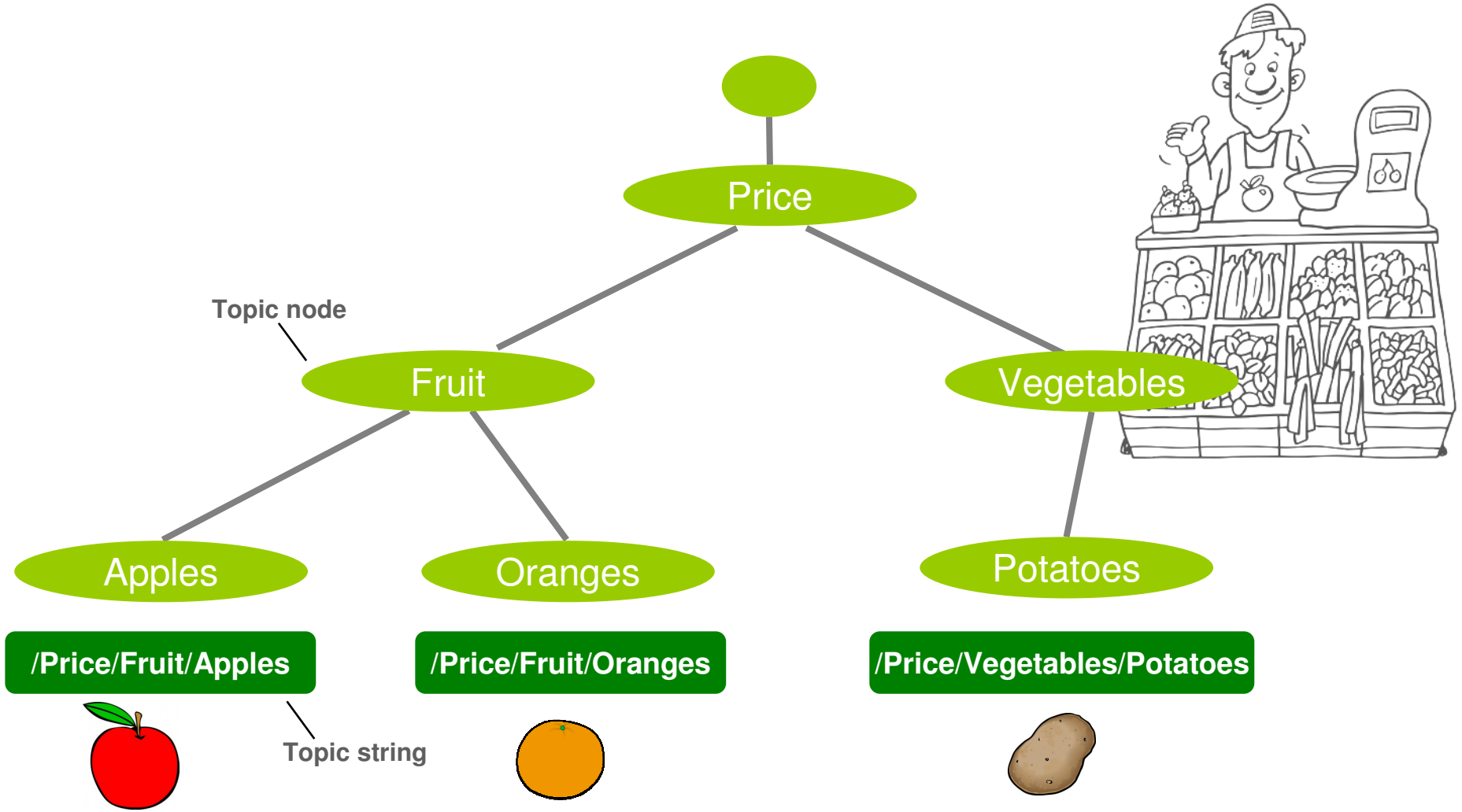
- **Subscriber**
  A subscriber is an application that consumes messages from a subscription

Publish/Subscribe since V7

Publish/Subscribe in WebSphere MQ

- **The queue manager holds a view of all the topic strings you are using in a hierarchical construct known as the topic tree (see next page). This topic tree is the central control point for all publish/subscribe. As a user you will interact with the topic tree in several different ways.**
- **You can configure the behaviour of the topic tree by defining topic objects and changing attributes on them. Of course you only need to do this if you want to change the default behaviour. You may not need any topic objects – we will look at this in more detail later.**
- **When using the MQI API you can programmatically interface with the topic tree as a subscriber using MQSUB and as a publisher using MQOPEN and MQPUT.**
  - Applications using other APIs that already support publish/subscribe (e.g. JMS) continue to work as before.
- **You can monitor the use of your topic tree by such applications using the Topic status command, the Subscription status command and the commands to display connections and their handles.**

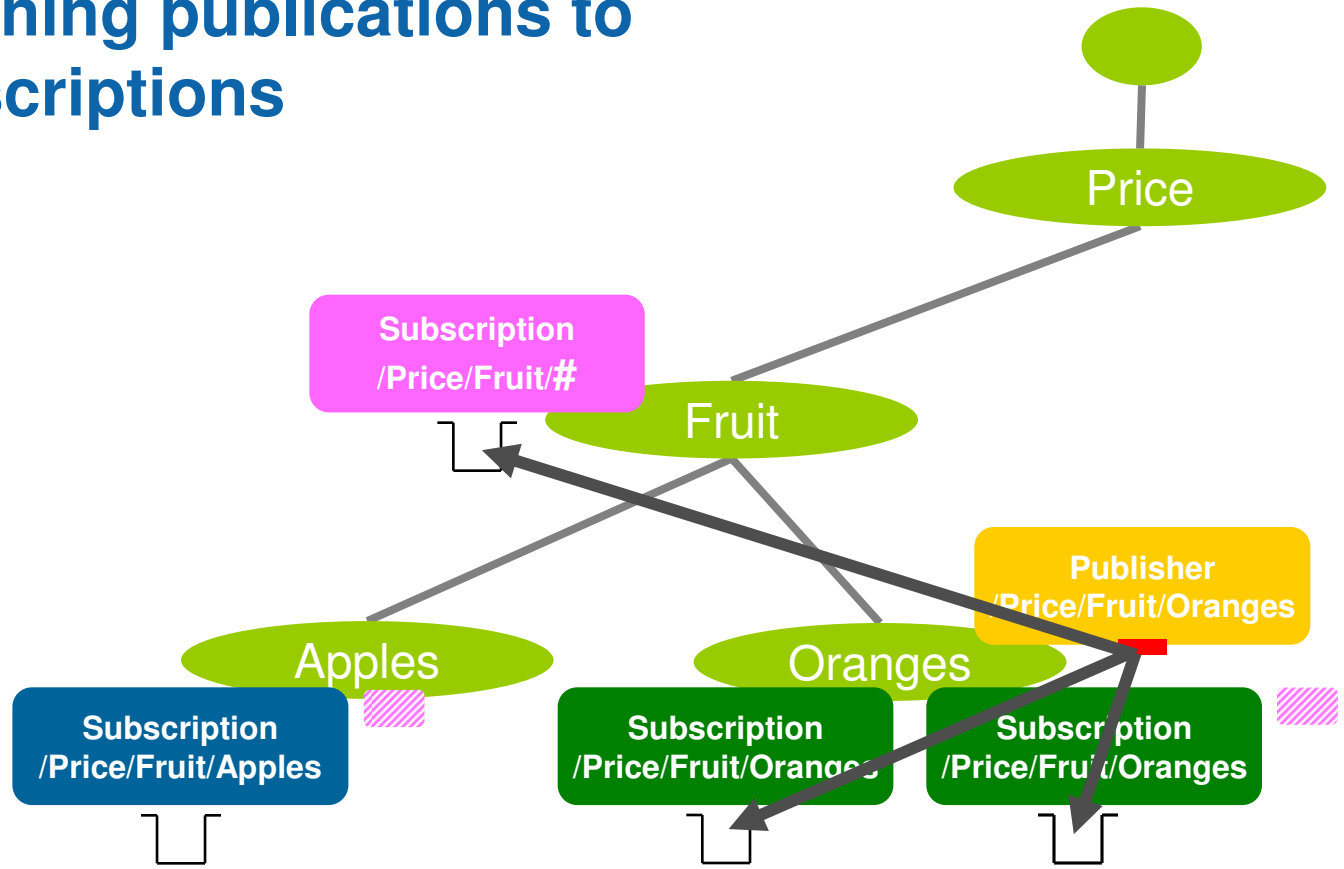
It's all about the ***topic tree***



Topic strings and topic tree

- A *topic string* is broken up into '/' delimited parts. Each part is represented as a *node* in WebSphere MQ's topic tree.
- As WebSphere MQ becomes aware of new topic strings, new topic nodes are automatically created in the topic tree.
- Topic strings can be any characters you choose. You can, and should, add structure to you topic strings using the '/' character. This produces a topic tree with a hierarchical structure, as the example on this foil shows. Although this hierarchical topic tree was created by the use of the topic strings shown, we generally picture it as a tree such as this.
- There are some special characters, apart from the '/' character that you should avoid in your topic strings. These are '#', '+', '*' and '?'. We will look at these in more detail later when we discuss wildcards.

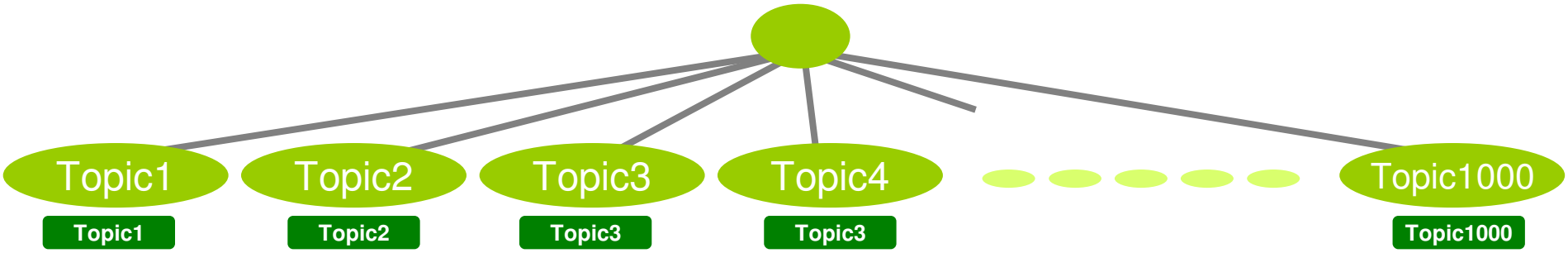
Matching publications to subscriptions



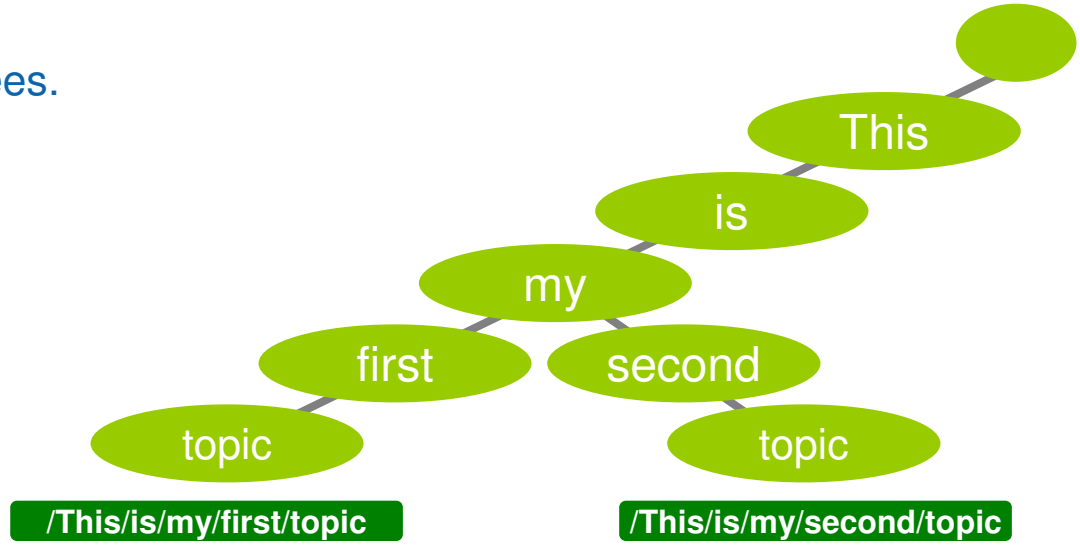
- Subscriptions are attached to matching nodes in the topic tree
- Publications identify the relevant topic node
- A copy of the publication is delivered to the queue identified by *each* matching subscription
- ***Wildcarding*** subscriptions at the topic node level can receive messages from multiple topic strings

Designing your topic tree structure

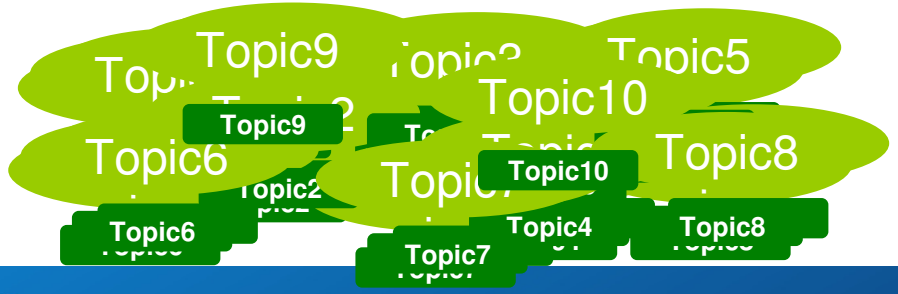
- Make it extendable, and understandable.



- Avoid excessively wide or deep topic trees.
  - Use structure where appropriate.
  - Limit it to *subscribable* content.



- Avoid a rapidly changing set of topics.



NOTES

Designing your topic tree structure

- No ‘standard’ way of designing the shape of a topic tree
  - Depends on the specific data
  - A well structured topic hierarchy should be intuitive to the user (but should also consider the later design considerations!)
- General recommendation is to describe the high level / broad information, then break this down into finer detail
  - Start by looking at the ‘big picture’
  - Topic Tree structure may evolve over time as user’s requirements become clearer
    - Build expandability into the design (e.g. always branch off from the root to allow future branches)
- If using distributed pub/sub, avoid clustering the root node – consider using high level topics such as ‘/global’ and ‘/local’ to make their scope clear.
- It’s possible to combine a topic hierarchy with message properties and selection strings to best fit the usage pattern
  - Potentially use where normally a subscriber may not require the further selection criteria but occasionally they may
    - E.g. Don’t encode the price into the topic for the rare occasion where a subscriber wants to know when it reaches £100, add it as a message property and allow a selector to be used.
  - Use of selectors can have an impact on performance
  - Selectors are not flowed around a multiple queue manager publish/subscribe topology
    - Selector parsing is only performed on the queue managers where the subscriptions exist

Why worry about the size and shape of the topic tree?

- First, it makes your administration simpler! The less topic objects and topic nodes there are, the less there are to worry about.
- Second, WebSphere MQ can efficiently scale to tens or hundreds of thousands of topic nodes but certain topic designs can cause WebSphere MQ pain, which impacts the user.
  - **WebSphere MQ V8 has made significant improvements**
- *It's best to understand what's happening internally…*
- When an application opens a topic string…
  - The topic string is looked up in a hash table
    - The more topic strings the bigger the hash table
  - If found, *job done*
  - If the topic string is not found the topic string is stripped down to its '/' delimited parts and every new topic node that is referenced in the topic string is dynamically created. Each new node is linked to its parent.
  - For every topic node created, that branch of the topic tree is walked backwards to discover
    - The node's configuration
    - The set of wildcard subscriptions that may match it
  - The more topic nodes the more memory consumed by the queue manager
- *For this reason:*
  - *Very broad topic trees put stress on certain parent topic nodes*
  - *Very deep topic trees add unnecessary topic nodes and levels of the tree that need to be walked.*
  - *Very many topics can put a strain on memory resources of the system*
  - *Subscriptions with # wildcards at the start or middle of the topic string put them higher up the tree and can therefore mean they will need to be checked for almost all topic nodes being created*

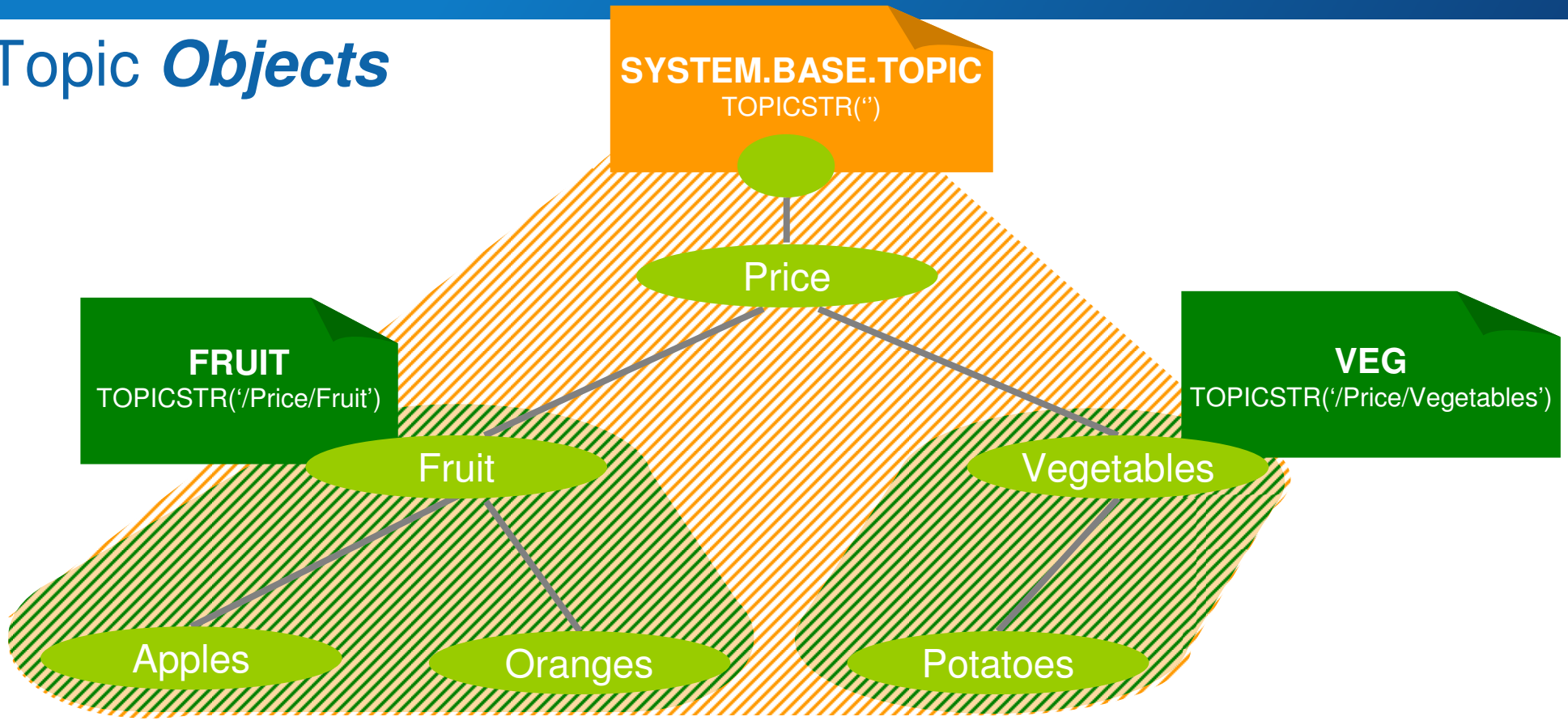
NOTES

Why worry about the size and shape of the topic tree?

- When an application publishes a message on a topic string
  - For every subscription associated with the topic node:
    - A decision is made to deliver a copy of the message or not, if the subscription has a selector the message properties are parsed to assess its suitability
- For this reason:
  - By example, if there is a single topic with a thousand subscriptions, all with selectors, each publication will result in a thousand checks.
  - However, if there are one hundred topics, each with ten subscriptions, all with selectors, each publication will result in ten checks.
  - Both would result in the same number of publications, the latter with a lot less work
- And finally, in the background…
  - Periodically the topic nodes are scanned to discover if they are still *needed*
    - An unneeded topic node is a leaf topic with no publishers or subscriptions currently directly attached
    - And the topic node has not been used for a period of time (minimum 30 minutes by default, controlled by the TREELIFE queue manager attribute)
  - In the case of unneeded topics they are deleted and unlinked from their parent
    - If new topic strings are continually being created, old ones need to be cleaned up at the same pace to prevent a problem.
    - From WebSphere MQ V8, use **DISPLAY PUBSUB ALL** to show the **TPCOUNT** for the queue manager. This is the current number of topic nodes in the tree. If this number continually grows, a problem may be present.
- This scanning is necessary to keep a cap on memory use and a cluttered topic tree when viewing.
- For this reason:
  - The bigger the topic hierarchy, the more additional work incurred.
  - Try not to have topics that are never or very rarely reused.
    - E.g. Don't encode a unique job ID into a topic string, add it as a message property.
  - Don't encode too many additional '/' delimited layers in the tree where unnecessary.

Publish/Subscribe configuration

Topic *Objects*



- Topic objects are a point of administration associated with a node in the topic tree.
- You start with a base object defined for the ' ' node … the rest are ***optional***.
- They provide hook points in the topic tree to configure specific pub/sub behaviour for a branch.
- A dynamically created topic node **inherits** its attributes from administered topic objects associated with topic nodes above it in the topic tree.

NOTES

Topic Objects

- Topic objects are a point of administration for a node or sub tree in the topic tree
  - SYSTEM.BASE.TOPIC is at the root of the topic tree
  - An administrator can define a topic object tied to any (non wildcard) topic string
  - Topic object definitions are configured in the same way as queue definitions
- They provide a 'hook' in the Topic Tree to configure:
  - Security / access controls
  - Scope (local to queue manager or global)
  - …many other configurable attributes
- A dynamically created topic node inherits its attributes from administered topic objects associated with topic nodes above it in the topic tree
  - Similarly one administered topic object can be configured to inherit some attributes from others above it
- Configuring topic objects is optional
  - Only define topic objects where they are needed, additional objects will only add administrative overhead and possible confusion

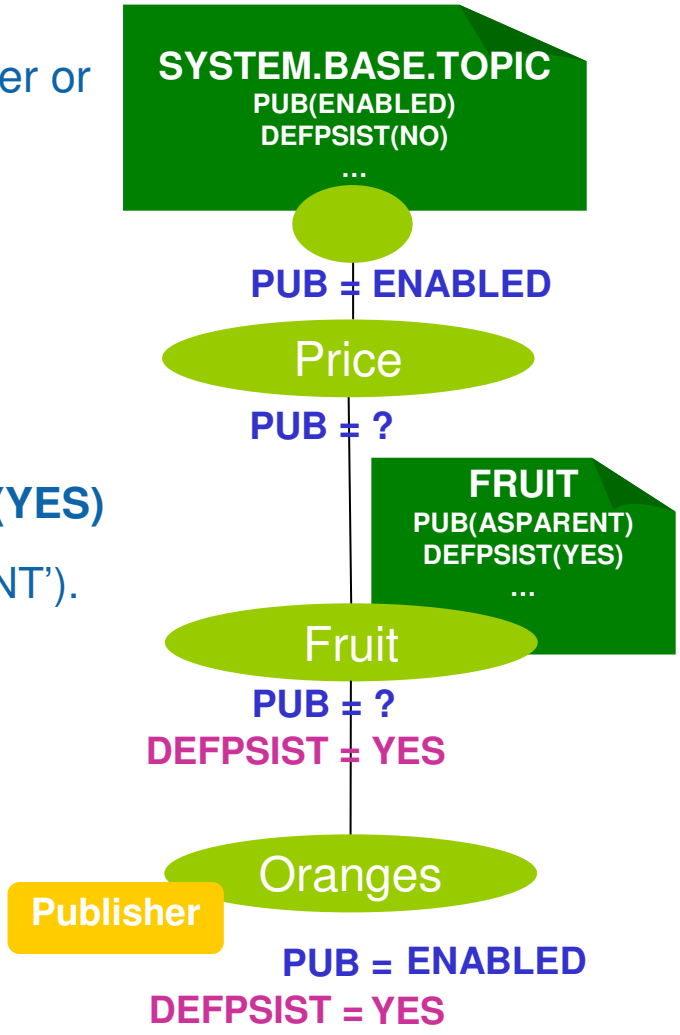
Topic object attributes

- Many attributes can be set on topic objects to effect a publisher or subscriber's behaviour.

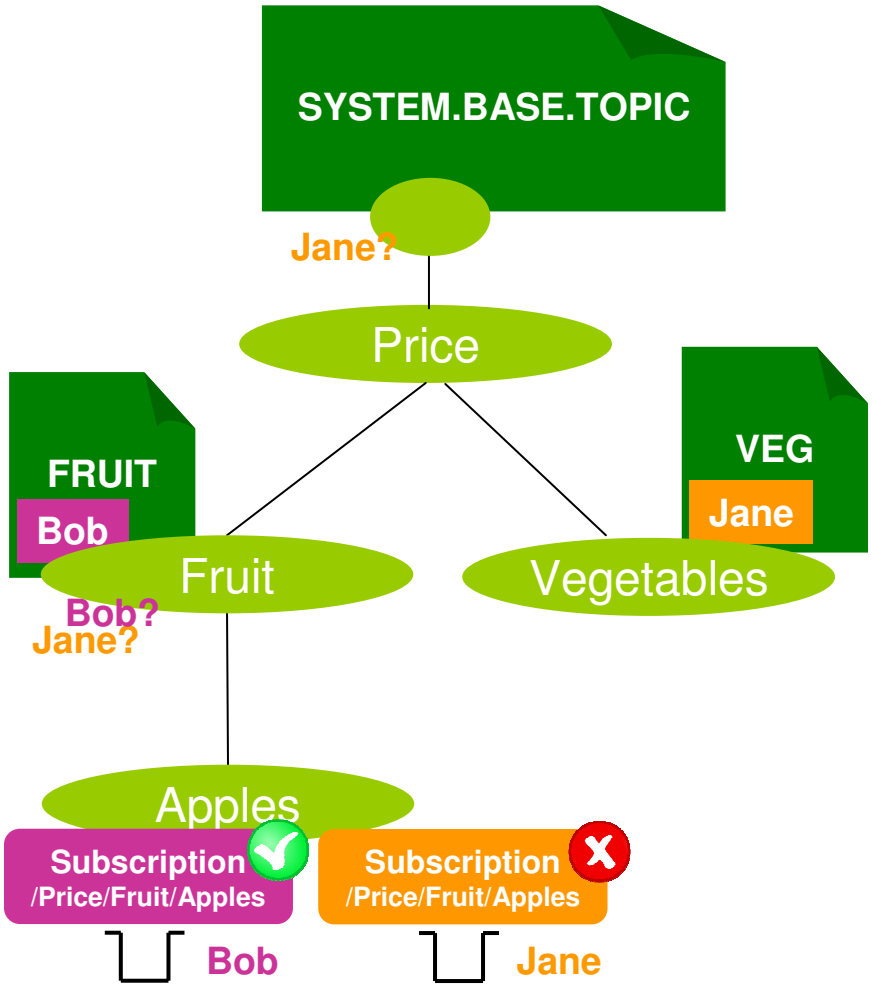
- Dynamic nodes inherit their behaviour from nodes above.

- Create a topic object for topic string '/**Price/Fruit**'
  - **DEFINE TOPIC(FRUIT) TOPICSTR('/Price/Fruit') DEFPSIST(YES)**
  - Attributes default to *inherit settings from above* (e.g. 'ASPARENT').
    - *(So by default, a new object does nothing)*

- Publish a message to topic string '/**Price/Fruit/Oranges**'
  - **Are publications enabled?**
  - **What message persistence to use?**



Topic Security



- Access control is set for a defined topic *object*
  - ***not a topic string!***
- Authority checks performed on the topic tree
  - Walk up the tree, just like attributes.
  - Keep checking until an authorisation is found or we run out of topic tree.
- Authority check on a subscription's destination queue
  - Check is for PUT to that queue
  - *(hang on for more on subscription queues)*
- ***Pick a suitable layer of the topic hierarchy and set access control at this point.***
- ***Think hard before adding additional access control at higher levels in the tree as this can cause confusion and grant very wide authorisations.***

Topic Security

- To set access control on a branch of the topic tree:
  - First define a topic object that represents that point in the topic tree.
  - Define the access control based on the name of that topic object.

- For example:
  - **DEFINE TOPIC(FRUIT) TOPICSTR('/Price/Fruit')**
  - **setmqaut -m QMGR1 -t topic -n FRUIT -p BOB +sub**

Managing topics

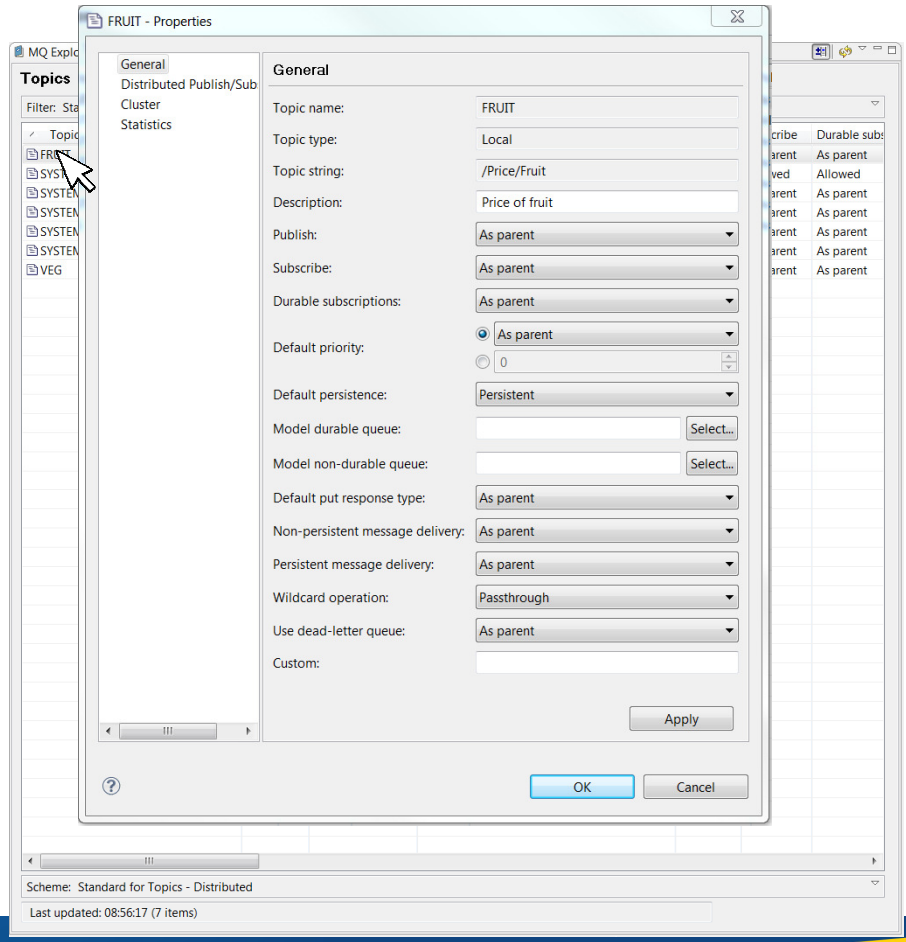
Managing topics

- Displaying topic object definitions
  - This shows how the administered topic objects were configured

```
5724-H72 (C) Copyright IBM Corp. 1994, 2014.
Starting MQSC for queue manager QMGR1. DISPLAY

DISPLAY TOPIC(*)
     1 : DISPLAY TOPIC(*)
AMQ8633: Display topic details.
   TOPIC(FRUIT)                            TYPE(LOCAL)
AMQ8633: Display topic details.
   TOPIC(SYSTEM.BASE.TOPIC)                TYPE(LOCAL)
AMQ8633: Display topic details.
   TOPIC(SYSTEM.BROKER.ADMIN.STREAM)       TYPE(LOCAL)
AMQ8633: Display topic details.
   TOPIC(SYSTEM.BROKER.DEFAULT.STREAM)     TYPE(LOCAL)
AMQ8633: Display topic details.
   TOPIC(SYSTEM.BROKER.DEFAULT.SUBPOINT)   TYPE(LOCAL)
AMQ8633: Display topic details.
   TOPIC(SYSTEM.DEFAULT.TOPIC)             TYPE(LOCAL)
AMQ8633: Display topic details.
   TOPIC(VEG)                              TYPE(LOCAL)

DISPLAY TOPIC(FRUIT)
     2 : DISPLAY TOPIC(FRUIT)
AMQ8633: Display topic details.
   TOPIC(FRUIT)                            TYPE(LOCAL)
   TOPICSTR(/Price/Fruit)                  DESCR(Price of fruit)
   CLUSTER( )                              CLROUTE(DIRECT)
   DURSUB(ASPARENT)                        PUB(ASPARENT)
   SUB(ASPARENT)                           DEFPSIST(YES)
   DEFPRTY(ASPARENT)                       DEFPRESP(ASPARENT)
   ALTDATE(2014-04-03)                     ALTTIME(08.44.48)
   PMSGDLV(ASPARENT)                       NPMSGDLV(ASPARENT)
   PUBSCOPE(ASPARENT)                      SUBSCOPE(ASPARENT)
   PROXYSUB(FIRSTUSE)                      WILDCARD(PASSTHRU)
   MDURMDL( )                              MNDURMDL( )
   MCAST(ASPARENT)                         COMMINFO( )
   USEDLQ(ASPARENT)                        CUSTOM( )
```



NOTES

Managing topic definitions

- **DISPLAY TOPIC(*) TOPICSTR**
  - Display all locally defined topic objects on this queue manager, including the topic string they are associated with.

- Although not yet covered in this presentation, but useful to list here, when using 'clustered topics' use the following command to show all the clustered topic objects known by this queue manager:
  - **DISPLAY TOPIC(*) TYPE(CLUSTER) TOPICSTR**

- **DISPLAY TOPIC(<name>)**
  - Show all configured attributes of the topic definition. This includes where a value is to be inherited from a higher topic object or not (check for 'ASPARENT' or ' ' values).
  - (See the next slide for how to view the inherited values of a topic)

Managing topics

- Displaying the topic tree
  - This shows how the nodes in the topic tree behave

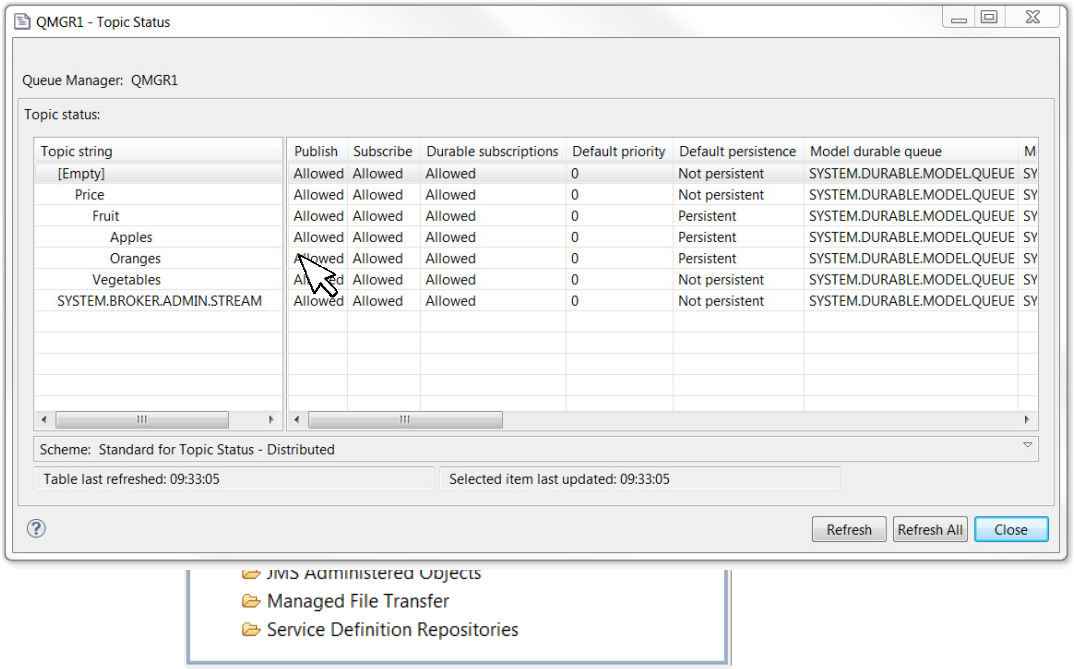
```
5724-H72 (C) Copyright IBM Corp. 1994, 2014.
Starting MQSC for queue manager QMGR1. DISPLAY

DISPLAY PUBSUB ALL
     1 : display PUBSUB all
AMQ8723: Display pub/sub status details.
   QMNAME(QMGR1)                           TYPE(LOCAL)
   STATUS(ACTIVE)                          SUBCOUNT(5)
   TPCOUNT(11)

DISPLAY TPSTATUS(‘#’) TOPICSTR WHERE(SUBCOUNT GT 0)
 22 : DISPLAY TPSTATUS('#') TOPICSTR where(SUBCOUNT GT 0)
AMQ8754: Display topic status details.
   TOPICSTR(/Price/Vegetables/Potatoes)    SUBCOUNT(1)
AMQ8754: Display topic status details.
   TOPICSTR(/Price/Fruit/Oranges)          SUBCOUNT(1)
AMQ8754: Display topic status details.
   TOPICSTR(/Price/Fruit/Apples)           SUBCOUNT(1)

DISPLAY TPSTATUS('/Price/Fruit/Apples')
    23 : DISPLAY TPSTATUS('/Price/Fruit/Apples')
AMQ8754: Display topic status details.
   TOPICSTR(/Price/Fruit/Apples)           ADMIN( )
   CLUSTER( )
   COMMINFO(SYSTEM.DEFAULT.COMMINFO.MULTICAST)
   MDURMDL(SYSTEM.DURABLE.MODEL.QUEUE)
   MNDURMDL(SYSTEM.NDURABLE.MODEL.QUEUE)
   CLROUTE(NONE)                           DEFPSIST(YES)
   DEFPRTY(0)                              DEFPRESP(SYNC)
   DURSUB(YES)                             PUB(ENABLED)
   SUB(ENABLED)                            PMSGDLV(ALLDUR)
   NPMSGDLV(ALLAVAIL)                      RETAINED(NO)
   MCAST(DISABLED)                         PUBCOUNT(0)
   SUBCOUNT(1)                             PUBSCOPE(ALL)
   SUBSCOPE(ALL)                           USEDLQ(YES)
```

V8



NOTES

Managing topic status

- **DISPLAY PUBSUB ALL**
  - Returns TPCOUNT which gives an idea of the total number of topic nodes currently in the topic tree of this queue manager (Added in WebSphere MQ V8).
  - This can be used to see if the tree is continually growing (introducing a memory usage risk) or stable. If growing it may indicate a problem in the topic tree design (too many unique topic strings). Investigate the 'TREELIFE' attribute of the queue manager for ways of improving this.
- **DISPLAY TPSTATUS('#')**
  - This will return an entry for every node in the topic tree.
  - Note the use of quoted topic string wildcard notation ('#') to achieve this, not the traditional MQSC wildcarding (*).
- **DISPLAY TPSTATUS('#') TOPICSTR WHERE(SUBCOUNT GT 0)**
  - Perhaps more useful, this command will only return those topic nodes with one or more subscriptions registered against it.
- **DISPLAY TPSTATUS('<topic string>')**
  - Displaying an individual topic node (using its topic string value) will return all resolved attributes of that node. This replaces any inheritable values with what they resolve to in the topic tree (so you won't see 'ASPARENT' in this output).
  - It also returns current status for this node, such as the number of registered subscriptions (SUBCOUNT) and currently connected publishers (PUBCOUNT).

Applications

Applications and *topics*

- *When creating subscriptions or opening topics to publish on, do I use a topic string or a topic object?*
  - A **topic string**. No, a **topic object**. No, **both**. Actually, er, ***any of them!***
- *So which should I use?*
  - Using the topic string is probably the easiest, it's closest to what the application is expecting
    - *Sub(- , '**/Price/Fruit/Apples**')* → **/Price/Fruit/Apples**
  - Using a topic object maps the operation to the topic string of that topic object
    - *Sub(**FRUIT**, '')* → **/Price/Fruit**
  - If you use both, you get both!
    - The topic string is appended to the topic string of the object
    - *Sub(**FRUIT**, '**Apples**')* → **/Price/Fruit/Apples**
- *If in doubt, check the topic tree for which nodes are actually being used*

Subscriptions

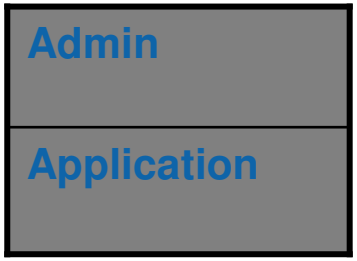
Subscription types

Different aspects of a subscription can be combined, don't assume it's one or the other…

Subscription types

Subscription creation and deletion

- Application **created subscriptions**
  - Applications use an API to dynamically create and delete subscriptions
- Administratively **created subscriptions**
  - An administrator defines subscriptions that can be accessed by applications
  - Applications can either use the publish/subscribe APIs to access these subscriptions or access their associated queue using point-to-point APIs.



NOTES

Types of Subscriptions

Subscription creation and deletion

- Application **created subscriptions**
  - Applications use an API to dynamically create and delete subscriptions
    - For example:
      - **MQI** uses the **MQSUB** to create and **MQCLOSE** to delete
      - **JMS** uses **TopicSession.createDurableSubscriber()** and **TopicSession.unsubscribe()**
- Administratively **created subscriptions**
  - An administrator defines subscriptions that can be accessed by applications
    - For example, in MQSC:
      - **DEFINE SUB('SUB1') TOPICSTR('/Price/Fruit/Apples').....**
      - **DELETE SUB('SUB1')**
  - Applications can either use the publish/subscribe APIs to access these subscriptions or access their associated queue using point-to-point APIs.

Subscription types

Subscription lifetime

- Durable **subscriptions**
  - The lifetime of the subscription is independent of any application
- Non-durable **subscriptions**
  - The lifetime of the subscription is bounded by the creating application
    - Subscriptions are automatically deleted when the application closes

	Durable	Non-durable
Admin		
Application		

Types of Subscriptions

Subscription lifetime

- Durable **subscriptions**
  - The lifetime of the subscription is independent of any application
    - Explicit creation and deletion of the subscription is required
    - Every durable subscription must be uniquely named within a queue manager
    - Subscriptions can be set to expire
- Non-durable **subscriptions**
  - The lifetime of the subscription is bounded by the creating application
    - Subscriptions are automatically deleted when the application closes
  - These cannot be administratively created

Subscription types

Subscription queue management

- **A subscription maps a topic to a queue. The queue relationship is either explicit or implicit…**
- Managed **subscription queue**
  - The subscription automatically creates and deletes a queue for the use of queuing any matching publications.
- Unmanaged **subscription queue**
  - When the subscription is created the name and location of an existing queue must be provided by you.

	Managed		Unmanaged	
	Durable	Non-durable	Durable	Non-durable
Admin	✓	✗	✓	✗
Application	✓	✓	✓ *(Not JMS)*	✓ *(Not JMS)*

NOTES

Types of Subscriptions

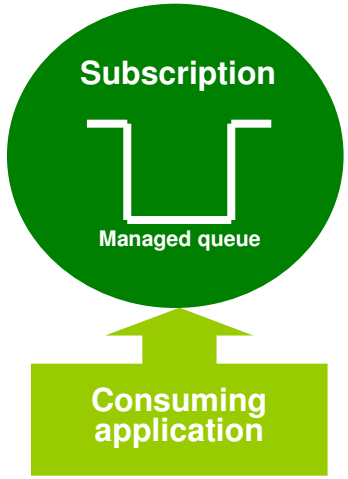
Subscription queue management

- Over simplifying things, a subscription essentially maps a topic to a queue
- The queue is either explicit or implicit…
- Managed **subscription queue**
  - The creation of the subscription automatically creates a queue to be used by this subscription for any matching publications.
  - Deleting the subscription automatically deletes the managed queue
  - JMS and XMS application created subscriptions are always managed subscriptions
- Unmanaged **subscription queue**
  - When the subscription is created the name and location of an existing queue is provided
  - When the subscription is deleted the queue is not affected
  - Multiple subscriptions can point to the same unmanaged queue
  - Unmanaged subscriptions can be administratively created or programmatically using the MQI interface

Accessing a subscription's messages

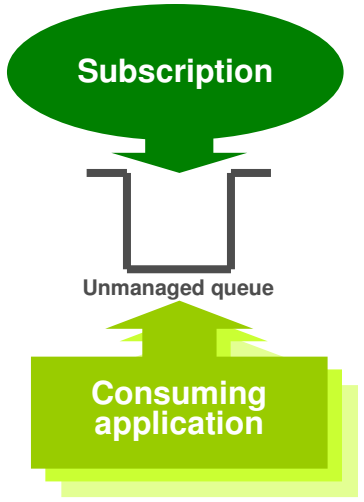
Via the *subscription*

- **An application opens the subscription**
  - *A true pub/sub application*
- **Works with managed and unmanaged subscription queues**
- **Limited to one attached consuming application at a time**
  - Unless you're using JMS cloned/shared subscriptions
- **Generally better pub/sub status feedback**



Via the *queue*

- **An application opens the queue associated with the subscription**
  - *This is really a point-to-point application*
- **Only works with unmanaged subscription queues**
- **Allows more freedom in what can be done**
  - For example, multiple concurrent consuming applications possible from any API



Controlling a subscription's specific queue

- Manually create the subscription through administration
  - For example, using runmqsc:

  DEFINE SUB(…) TOPICSTR('…') DEST*(name of queue)* DESTQMGR*(location of queue)* …

- Programmatically create the subscription
  - Through certain APIs
    - For example MQI, using the MQSUB verb
    - *Not supported with JMS*
  - Through programmatically sending an administrative command
    - By generating an MQCMD_CREATE_SUBSCRIPTION PCF command message
  - Through the *queued publish/subscribe* mechanism
    - Matches the pre MQ V7 model of sending subscription registration messages (MQPS_REGISTER_SUBSCRIBER) to a queue manager
    - Provided for compatibility with pre-V7 systems

Managing subscriptions

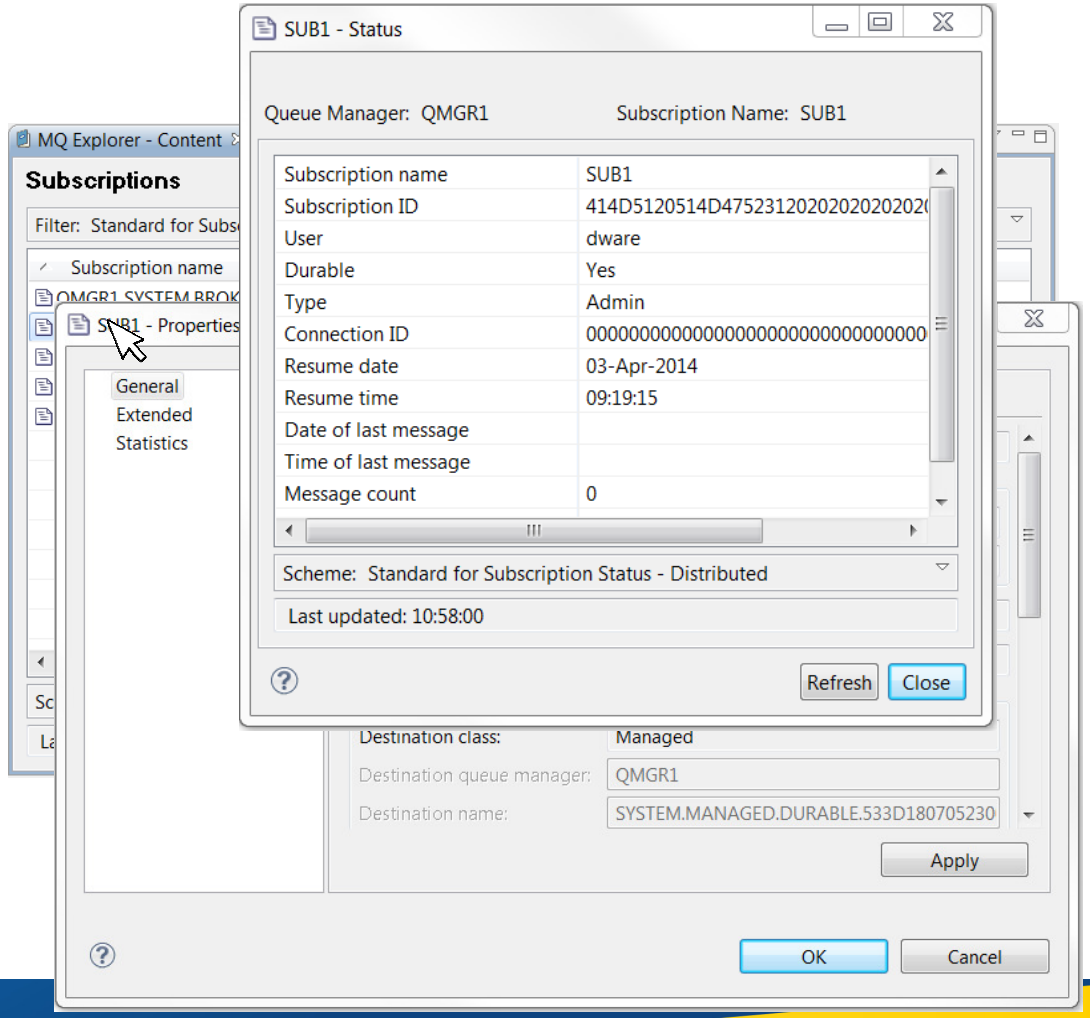
- Displaying subscriptions
  - This shows the subscriptions on a queue manager

```
5724-H72 (C) Copyright IBM Corp. 1994, 2014.
Starting MQSC for queue manager QMGR1. DISPLAY

DISPLAY SUB(SUB1)
     2 : DISPLAY SUB(SUB1)
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D4752312020202020202007183D5320002306)
   SUB(SUB1)                               TOPICSTR(/Price/Fruit/Apples)
   TOPICOBJ( )
   DEST(SYSTEM.MANAGED.DURABLE.533D180705230020)
   DESTQMGR(QMGR1)                         PUBAPPID( )
   SELECTOR( )                             SELTYPE(NONE)
   USERDATA( )
   PUBACCT(16010515000000DEA960DF651724E4B97C192FE8030000000000B)
   DESTCORL(414D5120514D4752312020202020202007183D5320002306)
   DESTCLAS(MANAGED)                       DURABLE(YES)
   EXPIRY(UNLIMITED)                       PSPROP(MSGPROP)
   PUBPRTY(ASPUB)                          REQONLY(NO)
   SUBSCOPE(ALL)                           SUBLEVEL(1)
   SUBTYPE(ADMIN)                          VARUSER(ANY)
   WSCHEMA(TOPIC)                          SUBUSER(xxxx)
   CRDATE(2014-04-03)                      CRTIME(09:19:15)
   ALTDATE(2014-04-03)                     ALTTIME(09:19:15)

DISPLAY SBSTATUS(SUB1)
     3 : DISPLAY SBSTATUS(SUB1)
AMQ8099: WebSphere MQ subscription status inquired.
   SUB(SUB1)
   SUBID(414D5120514D4752312020202020202007183D5320002306)
   SUBUSER(xxxx)                           RESMDATE(2014-04-03)
   RESMTIME(09:19:15)                      LMSGDATE( )
   LMSGTIME( )
   ACTCONN(000000000000000000000000000000000000000000000000)
   DURABLE(YES)                            MCASTREL( , )
   NUMMSGS(0)                              SUBTYPE(ADMIN)
   TOPICSTR(/Price/Fruit/Apples)

DISPLAY QLOCAL(SYSTEM.MANAGED.DURABLE.533D180705230020)
```



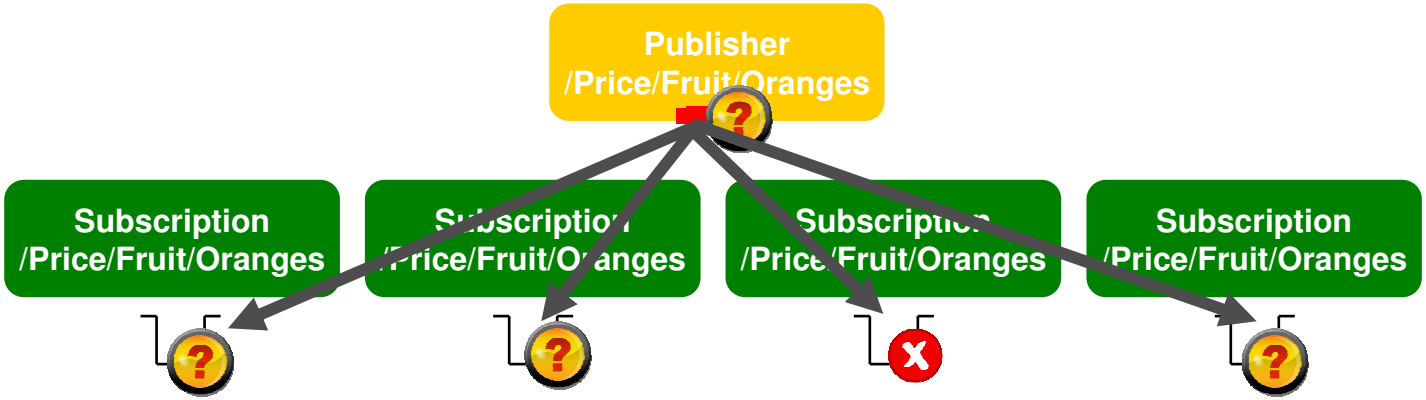
NOTES

Managing subscriptions

- **DISPLAY PUBSUB ALL**
  - Gives an idea of the total number of subscriptions currently registered on this queue manager (Added in WebSphere MQ V8).
- **DISPLAY SUB(*) TOPICSTR**
  - List all the subscriptions, including the topic string that they are subscribing to.
- **DISPLAY SUB(SUB1)**
  - Display the configuration of a subscription.
- **DISPLAY SBSTATUS(SUB1)**
  - Display the status of a subscription. This includes things like the last time it was resumed (opened) and approximately how message messages have matched this subscription since it was created (or since a queue manager restart, if later).
- **DISPLAY SBSTATUS(*) WHERE(ACTCONN NE '000000000000000000000000000000000000000000000000')**
  - Another attribute returned by SBSTATUS is ACTCONN, this is the connection ID if the subscription is currently in use. The above query will return all in use subscriptions. The last 16 characters of ACTCONN can be used with DISPLAY CONN to show further application information.
- **DISPLAY QLOCAL(<subscription's DEST>)**
  - One property returned by DISPLAY SUB is the DEST, this is the queue where messages will be delivered. Standard queue queries can be used on that queue.

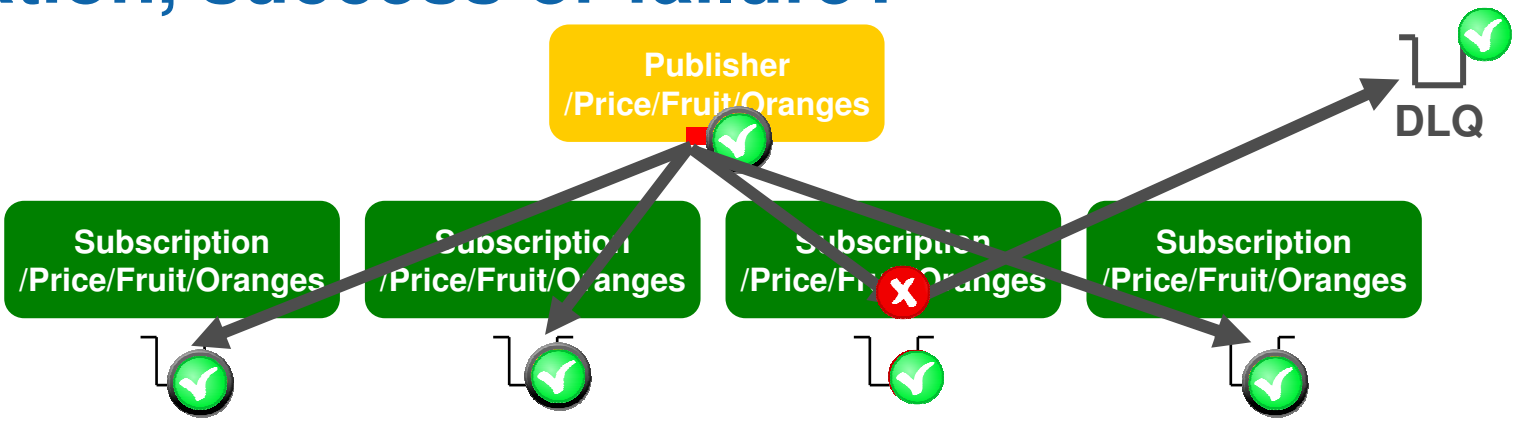
Publishing

Publication, success or failure?



- Point-to-point is nice and simple:
  - Did the message get onto the queue?
  - Was it persistent and transacted?
- Publish/subscribe is not so clear cut:
  - **Persistence and transactions still ensures integrity of publications.**
  - But if one or more subscriptions can't receive the publication, ***should the publish fail?***

Publication, success or failure?



- *Should those subscriptions impact the others?*
- *What if the subscriptions are non-durable, or the publication is non-persistent?*

- Controlled at the topic level
  - Persistent Message Delivery (**PMSGDLV**) and Non-persistent Message Delivery (**NPMSGDLV**).
    - **ALL, ALLDUR, ALLAVAIL**

- Don't forget that being able to DLQ a publication is still counted as a *success!*
  - **USEDLQ** on the topic to fine tune this behaviour. V7.1

- *And finally, remember – when there are no subscriptions, no-one gets it. That's still a successful publish!*

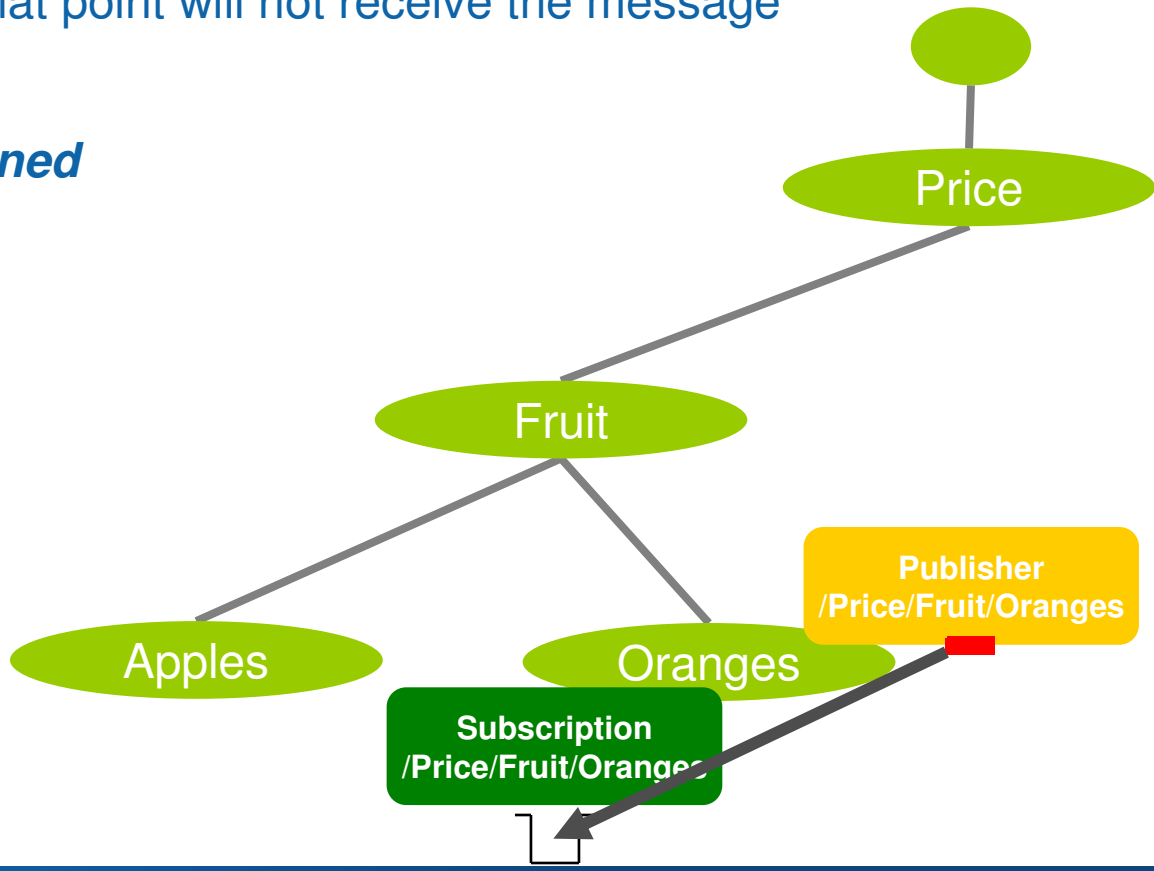
NOTES

Publication, success or failure?

- The publish/subscribe model encourages separation between the publishers of information and each of the subscribers to the information.
- If a subscriber cannot receive a message published to a topic (e.g. it's subscription queue is currently full), should that affect the other subscribers? And should the publisher be aware of the problem?
- The default behaviour in WebSphere MQ depends on the persistence of the message and the durability of the subscription:
  - Persistent messages (PMSGDLV=ALLDUR):
    - The publish will fail if it cannot be delivered to one or more **durable** subscriptions. Failures to any **non-durable** subscriptions are *acceptable*.
  - Non-persistent messages (NPMSGDLV=ALLAVAIL):
    - Any failures to subscriptions are acceptable, the publish will succeed (even if no-one can receive the message).
- This behaviour can be changed through configuration of the PMSGDLV and NPMSGDLV attributes of a topic.
- If a dead letter queue is configured, a failure to put a publication to a subscription's queue will result in an atempt to DLQ it. If that is successful the put to the subscription is classed as a success.
  - This can be controlled through the use of the USEDLQ option on a topic (available from WebSphere MQ V7.1 onwards).

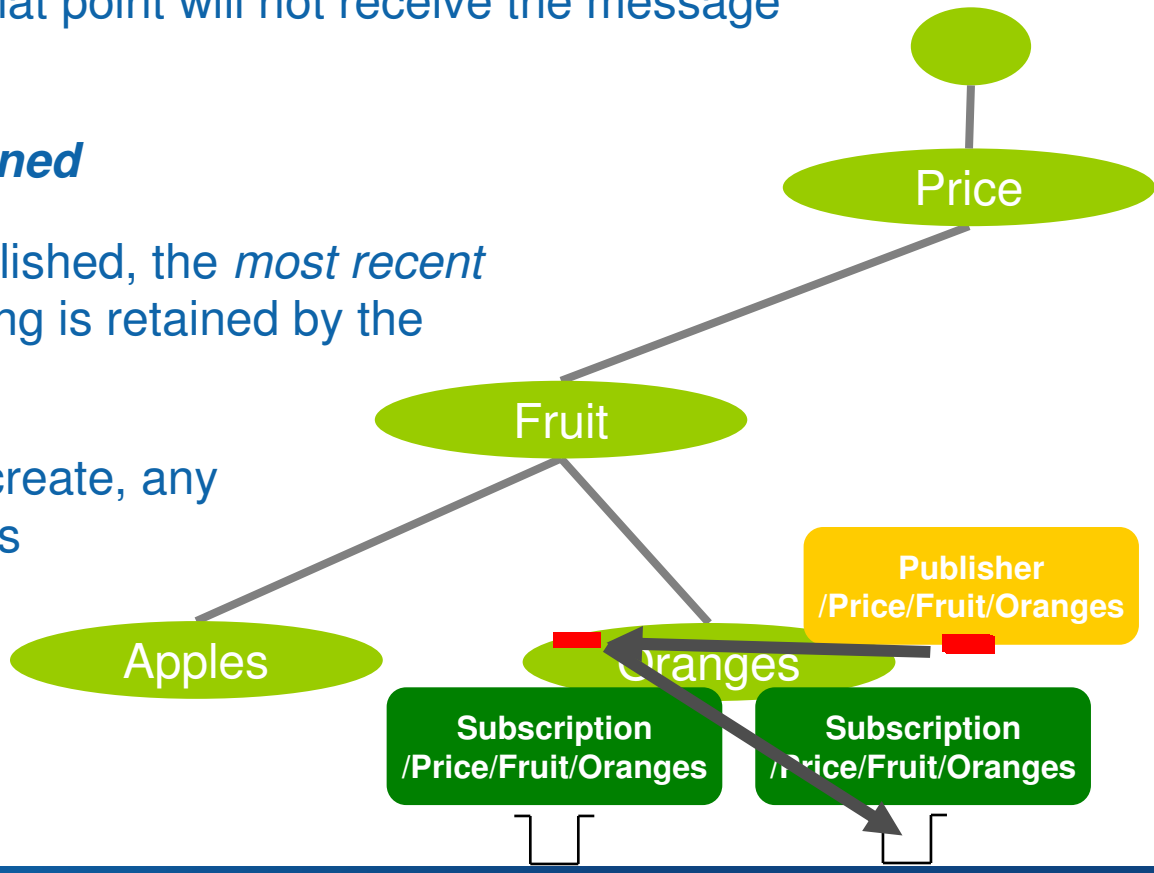
Retained publications

- When a message is published to a topic string, it is delivered to each matching subscription registered at that time.
- Subscriptions created after that point will not receive the message only newly published ones.
- Unless publications are ***retained***



Retained publications

- When a message is published to a topic string, it is delivered to each matching subscription registered at that time.
- Subscriptions created after that point will not receive the message only newly published ones.
- Unless publications are ***retained***
- Every time a message is published, the *most recent* publication for each topic string is retained by the queue manager.
- When a new subscription is create, any matching retained message is delivered to it.
- ***Take care, it's subtle***



NOTES

Retained publications

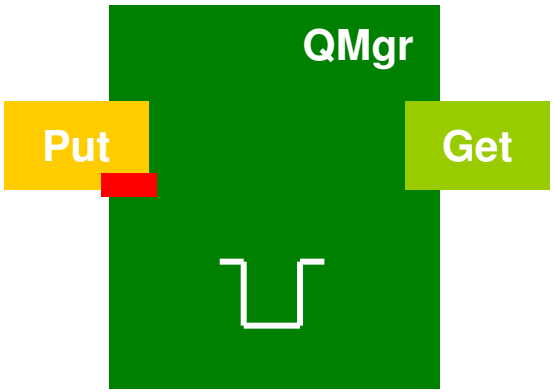
- Retained publications can be useful if information on a topic is infrequently published, and a new subscriber needs to see the latest update as soon as they are created.
- These messages are stored by the queue manager on a queue. The same recommendation of not building a very deep application queue applies equally here. Therefore, calculate the total number of topic strings where publications would be retained when considering using retained publications.
- Retained publications in a multi queue manager pus/sub topology (not yet covered in this presentation) introduce complexities that need to be understood.

- If you are only connected to the system from time to time, or if regular updates are not what you require, you can choose to only get publications when you request them (MQI only)
  - You make a subscription in the normal way, additionally using the MQSO_PUBLICATIONS_ONLY_REQUEST option. Then whenever you want the latest state, you make a request for it using the MQSUBRQ verb. This will just send you the latest publication on the topic you have subscribed on.
- If you have a wild-carded subscription you will get the latest publication on each topic string that matches your subscription. When using the MQI, the field called NumPubs in the MQSRO structure will detail how many publications have been sent to you.

Topologies

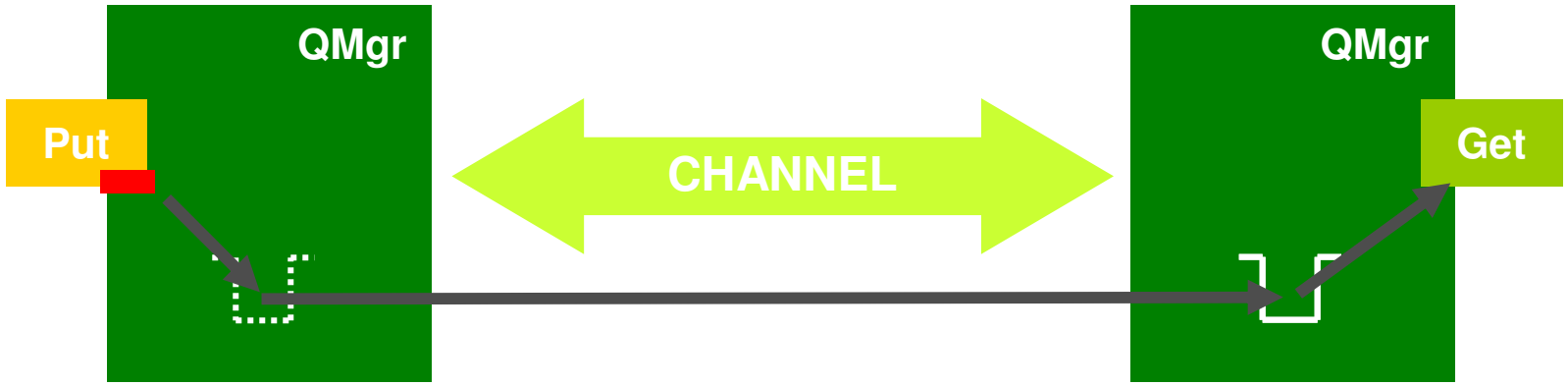
Distributed queuing

- We all know how point-to-point works…



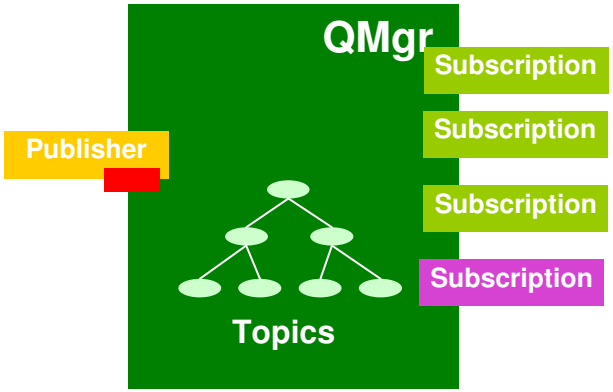
Distributed queuing

- We all know how point-to-point works
- Even across multiple queue managers



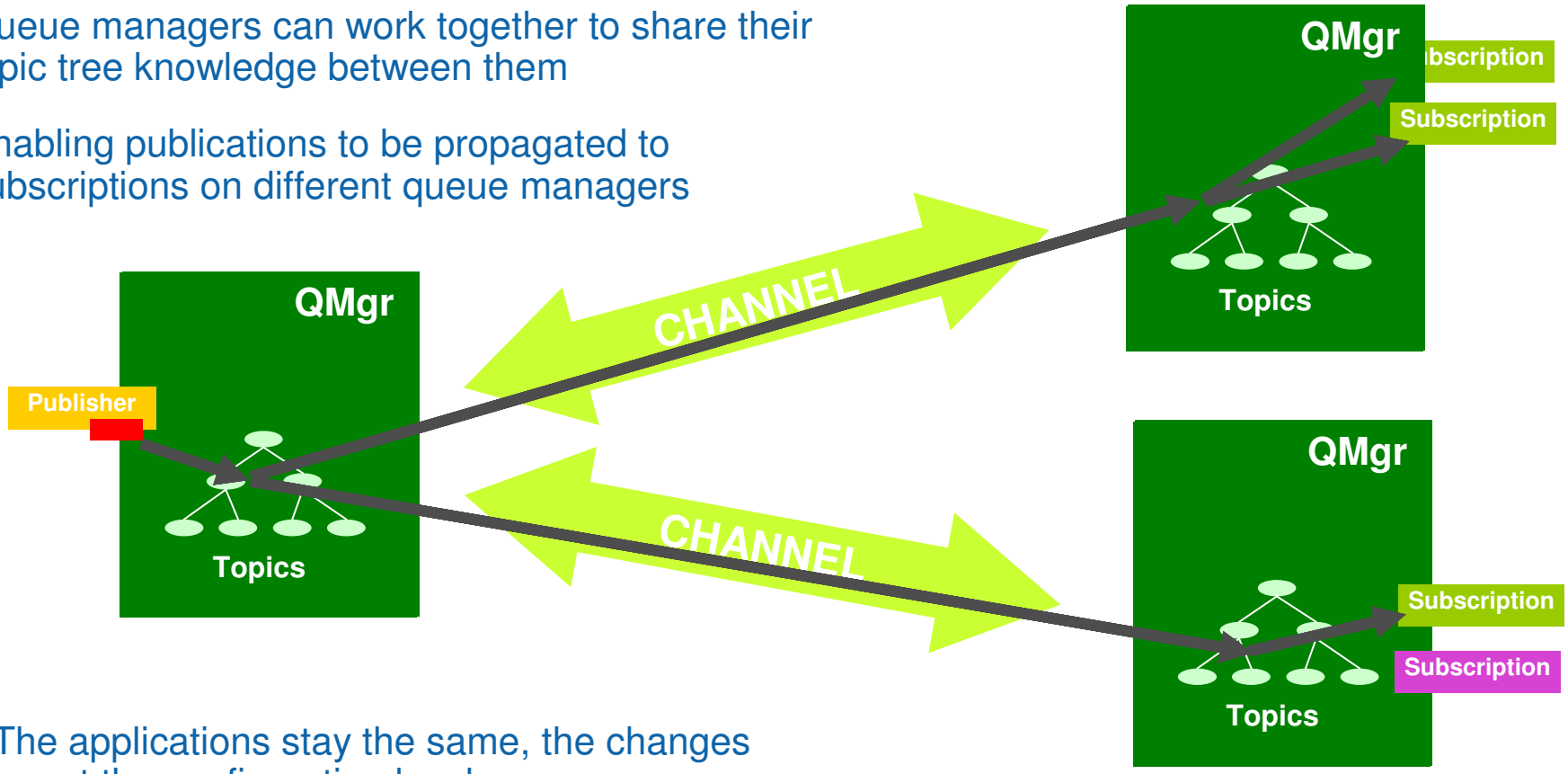
Distributed publish/subscribe

- And we know how everything revolves around the topic tree, dynamically built up in a queue manager



Distributed publish/subscribe

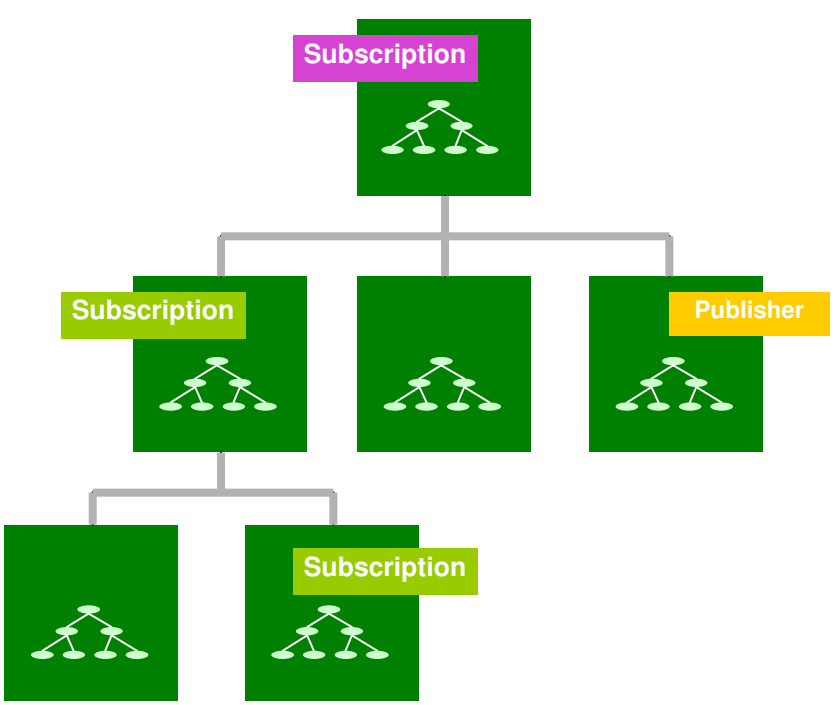
- And we know how everything revolves around the topic tree, dynamically built up in a queue manager
- Queue managers can work together to share their topic tree knowledge between them
- Enabling publications to be propagated to subscriptions on different queue managers

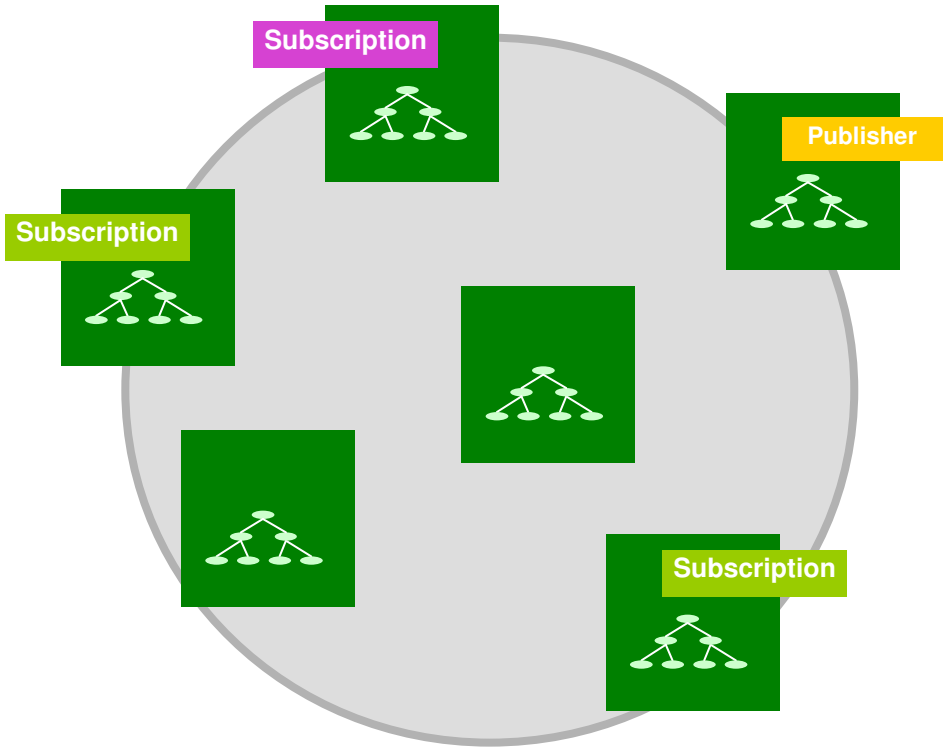


- The applications stay the same, the changes are at the configuration level.

Distributed publish/subscribe topologies

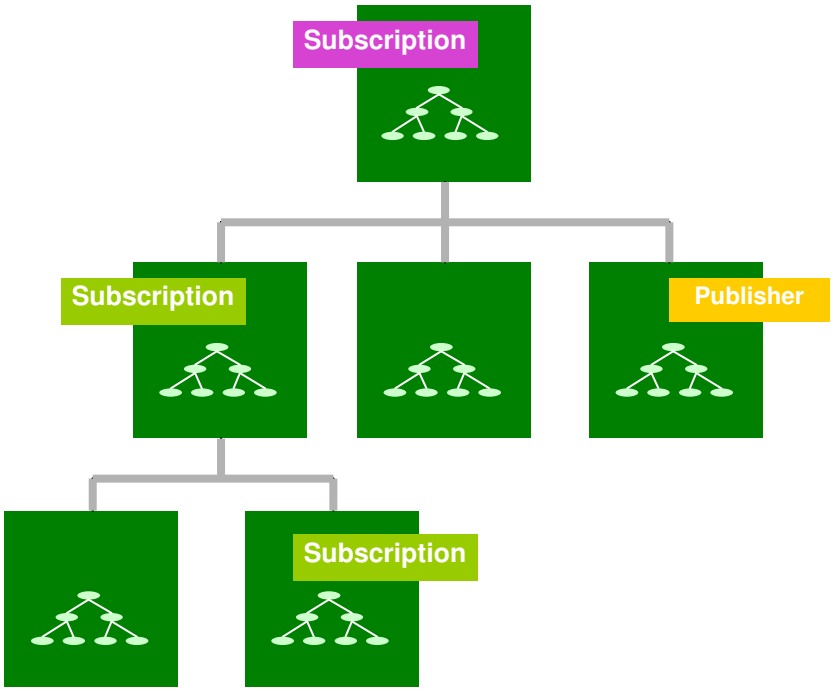
- Publish/subscribe topologies can either be created as a defined ***hierarchy*** or more dynamically as a ***cluster***

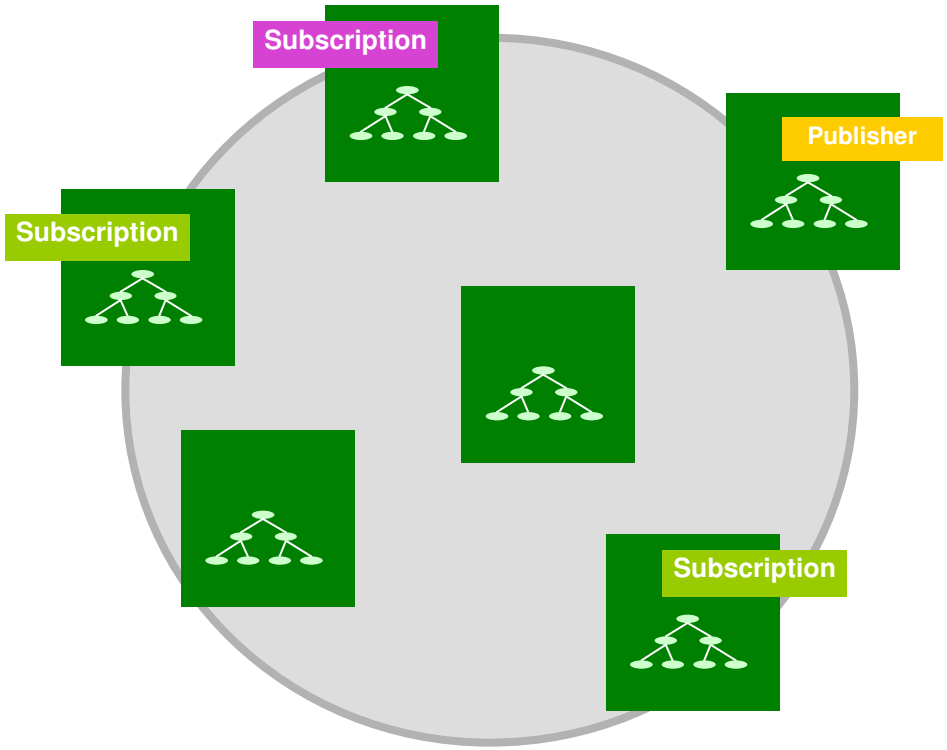




Distributed publish/subscribe topologies

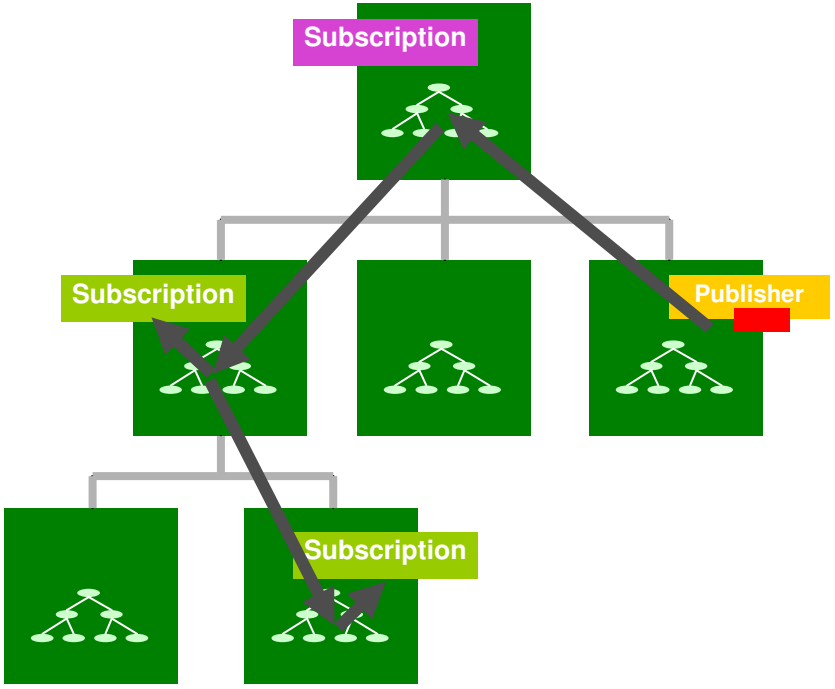
- A hierarchy manually defines the relationship between each queue manager.
- Messages are flowed via those relationships.





Distributed publish/subscribe topologies

- A hierarchy manually defines the relationship between each queue manager.
- Messages are flowed via those relationships.



NOTES

Pub/Sub Hierarchies

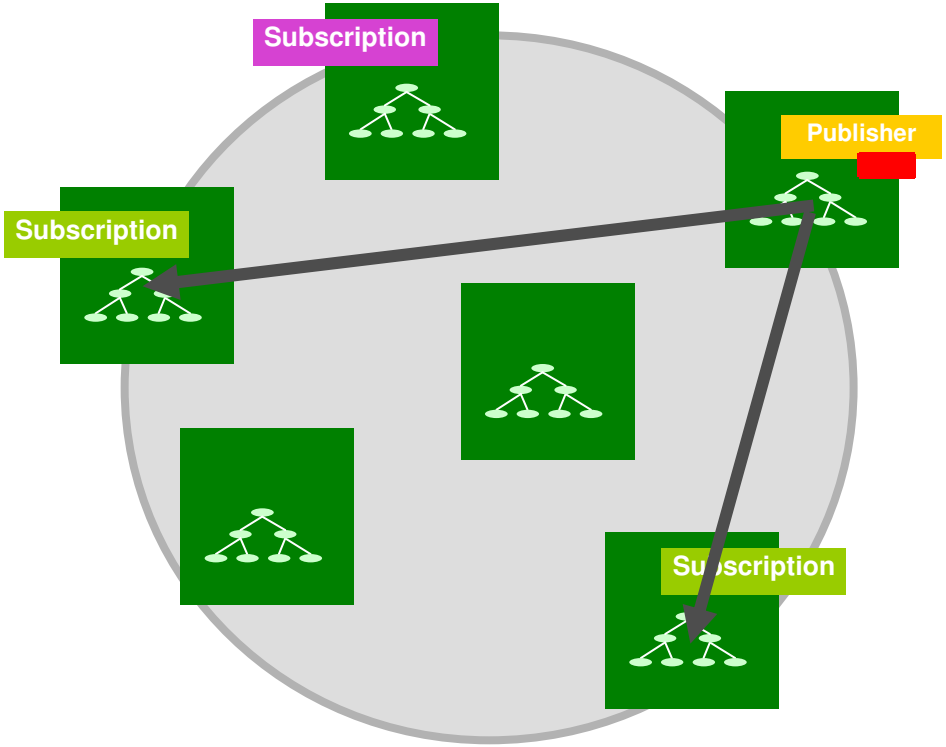
- By default publications **will** flow between queue managers from publishers to subscribers in a hierarchy.

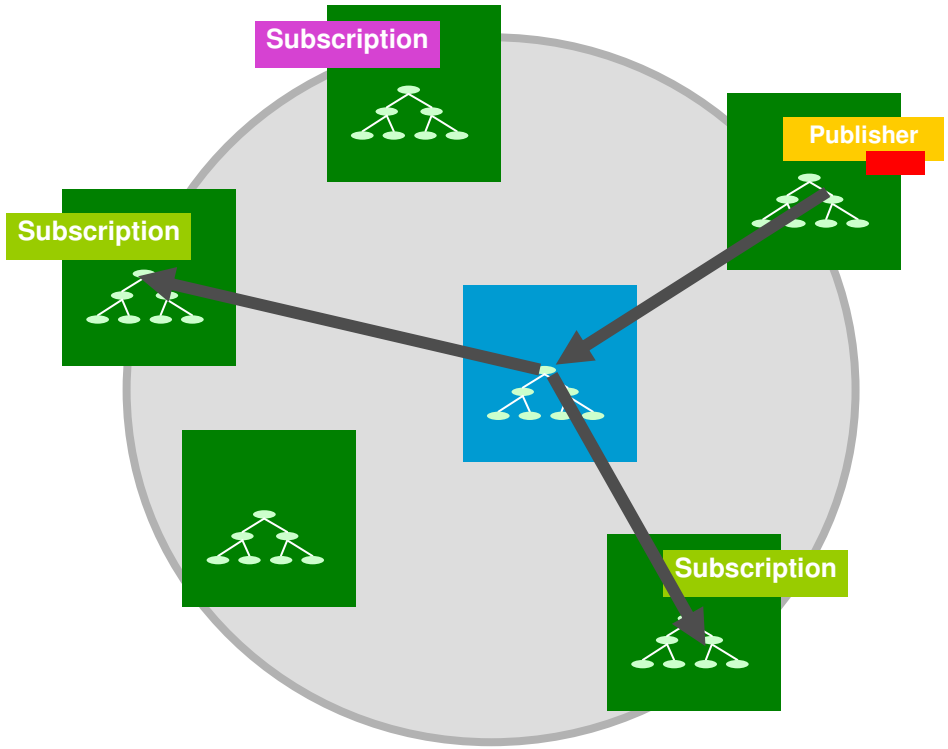
Points to consider:

- Consider the scalability of the *higher* nodes in the hierarchy, especially the root as more publication traffic is expected to flow through these.
- As for clusters, all topic ***strings*** subscribed to are propagated throughout the cluster, not just to the immediate neighbours
  - Rapid change to the set of subscriptions can become a limiting factor
- Propagating is asynchronous, so early publications may not be sent following a subscription to a new topic string

Distributed publish/subscribe topologies

- A cluster can be used for publish/subscribe through simple configuration.
- Any queue manager can publish and subscribe to topics
- Published messages can go **direct** between queue managers





V8

- Or be **routed** via specific queue managers

NOTES

Pub/Sub Clusters

- Publications **will not** flow between queue managers in a cluster unless the branch of the topic tree is ***clustered***.
  - This is achieved by setting the '**CLUSTER**' property of an administered topic object at the root of that topic tree to the name of the cluster.
  - The topic object can be defined in a single queue manager in the cluster, clustering it will automatically share the topic object definition with *all* other queue managers in the cluster.

Points to consider:

- A pub/sub cluster behaves very differently to a pure point-to-point cluster, plan accordingly:
  - All queue managers automatically know of all other queue managers
  - All queue managers with subscriptions to clustered topics will create channels to all other queue managers in the cluster
  - The above affects queue managers that have no publishers or subscribers but are in the same cluster as the topics.
- All clustered topic ***strings*** subscribed to are propagated throughout the cluster
  - Rapid change to the set of subscriptions can become a limiting factor
- Propagating is asynchronous, so early publications may not be sent following a subscription to a new topic string

Summary

- Publish/Subscribe in WebSphere MQ
- Administration of publish/subscribe
- Management of publish/subscribe
- Subscriptions and publications
- Topologies

Questions & Answers

Legal Disclaimer