RabbitMQ
Open Source Enterprise Messaging

Messaging with AMQP and RabbitMQ

Systems Integration

PBA Softwareudvikling/BSc Software Development

Tine Marbjerg

Fall 2017

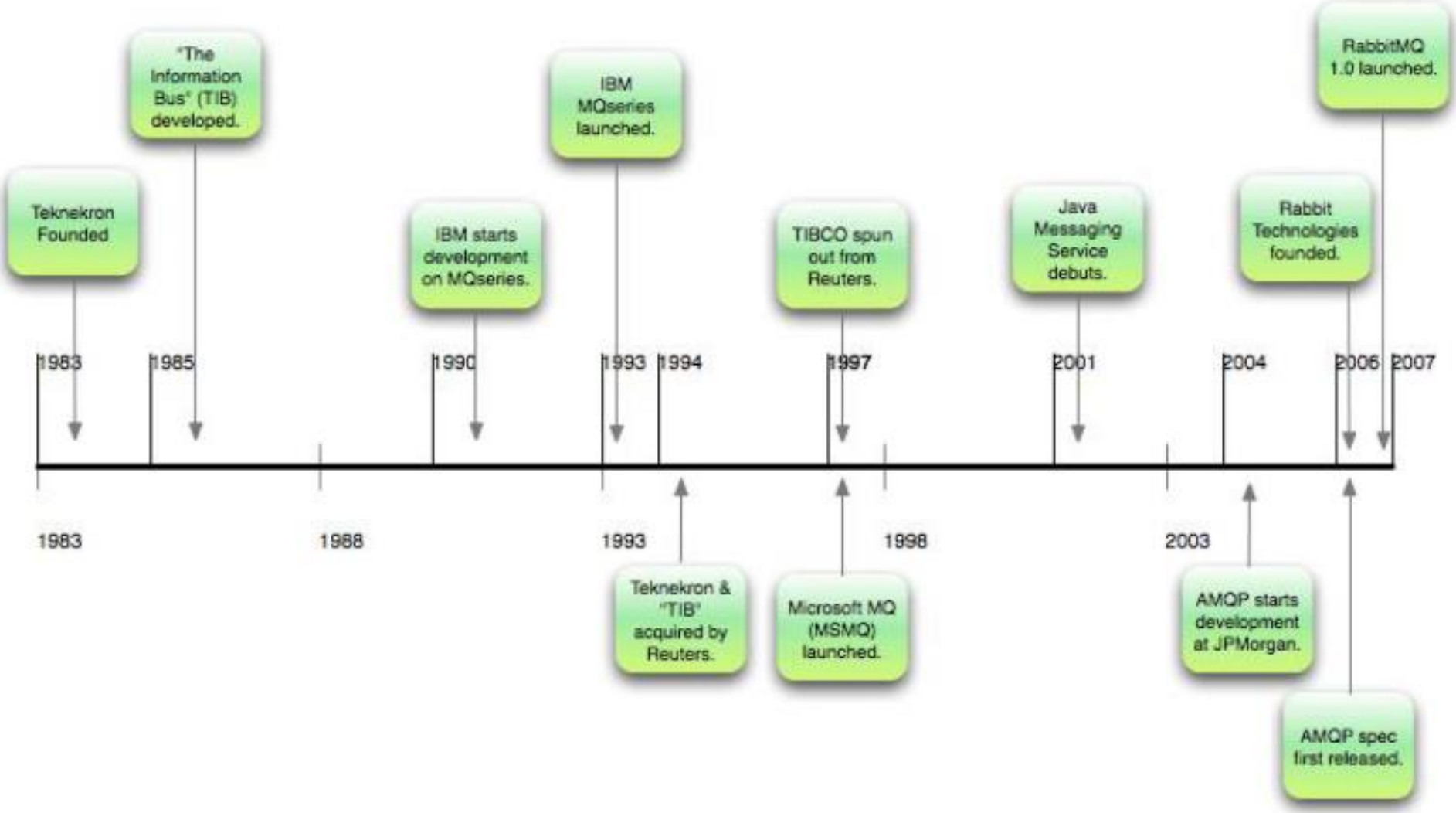
Agenda for Today

- C# /MSMQ exercises from last week
- Routing (EIP chap. 7)
- Intro to RabbitMQ message broker
  - AMQP message protocol
  - AMQP terminology
  - Implementations of selected messaging patterns (demos)
- RabbitMQ exercises

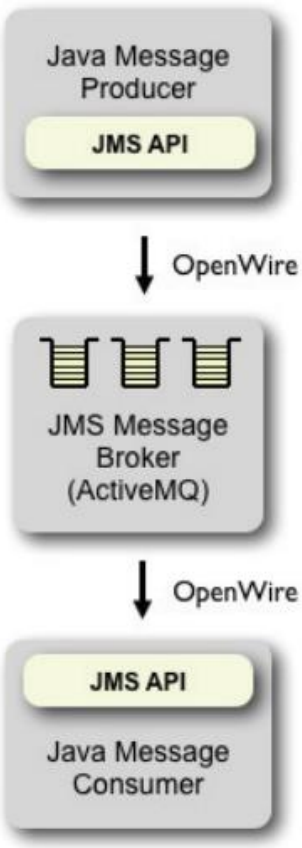
Messaging Implementations

- **MSMQ (Microsoft Message Queuing)**
  - Microsoft’s own messaging system non-open, non-free standard
  - One vendor only
  - API's to few languages at codeplex.com (moving to github ult. 2017) etc.
  - Requires MS Windows to run
- **JMS (Java Message Service)**
  - A standard messaging API = interoperability between vendors of JMS implementations
  - Pretty much bound to Java
- **AMQP (Advanced Message Queuing Protocol)**
  - Open standard (defined by Oasis) over-the-wire protocol = (theoretical) full interoperability between vendor implementations
  - Language agnostic (Libs for 10+ languages)

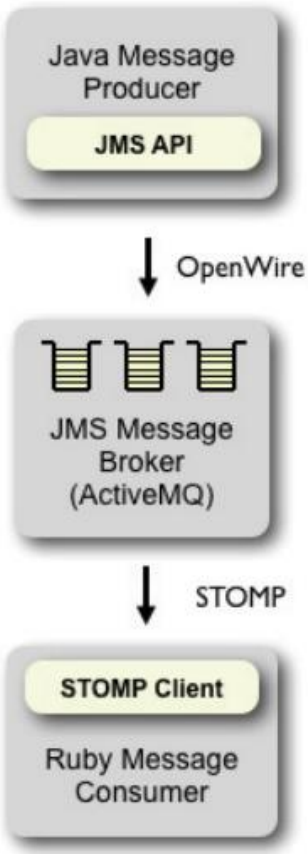
Timeline of Message Queueing



Connecting Messaging Systems

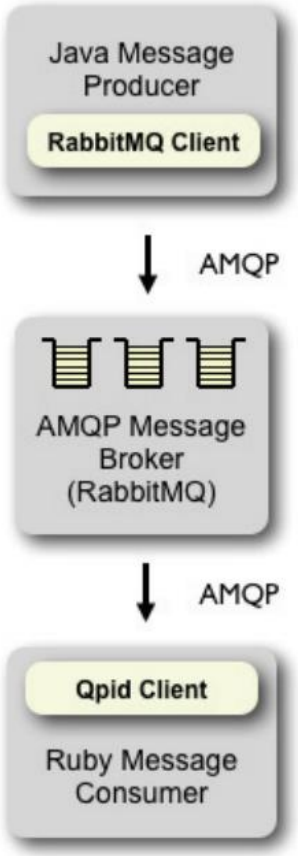


Same language
all OK



Different language
Different protocol
~Vendor lock-in

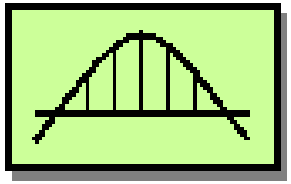
Different language
Different protocol
No vendor support



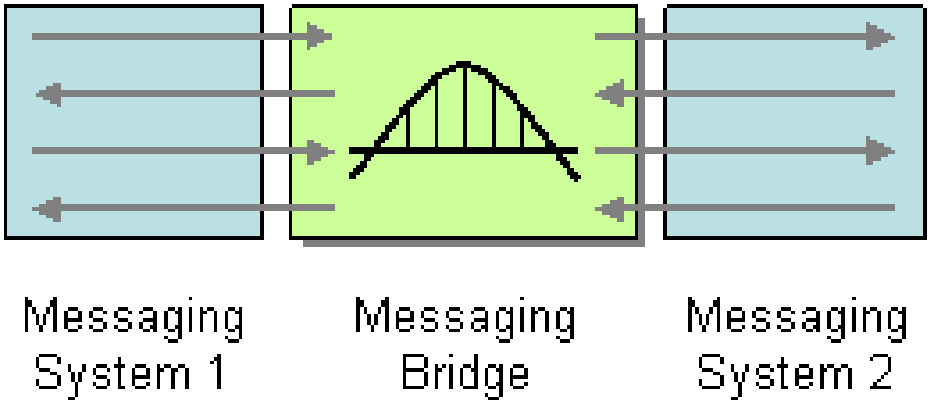
Same protocol
all OK

Source:
http://www.wmrichards.com/amqp.pdf

Messaging Bridge (133)



- How can multiple messaging systems be connected so that messages available on one system are also available on the others?



Ex: ActiveMQ has built-in message bridge between OpenWire (JMS) and STOMP (Ruby)

- Use a *Messaging Bridge*, a connection between messaging systems, to replicate messages between systems.

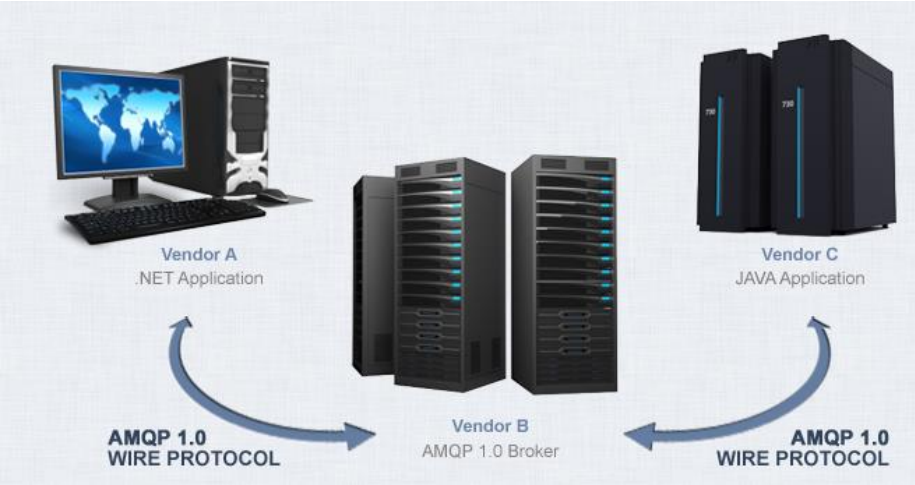
- The bridge acts as map from one set of channels to the other, and transforms the message format of one system to the other

AMQP

- Standard messaging protocol across platforms featuring:
  - message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security.
- No standard API to program up against
  - Rather, it is specification for industry standard wire-level binary protocol that describes how messages are structured and sent across the network
- So what client API and message broker should you use?
  - It doesn't matter ☺
  - Use AMQP compliant message broker
  - Use AMQP compliant client library

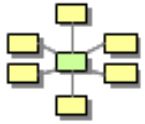
AMQP Message Broker Implementations

- Several broker implementations:
  - RabbitMQ by VMware
  - Qpid by Apache
  - MRG by Redhat (variant of Qpid)
  - ActiveMQ by Apache
  - ...

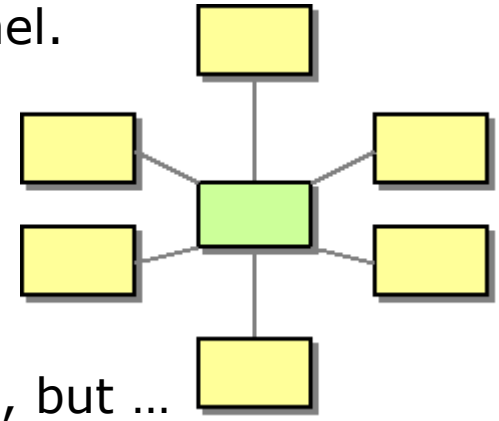


- Several big business users:
  - OpenStack platform
  - JPMorgan
  - Deutche Börse (German stock exchange)
  - AT&T
  - Google and many more.

Message Broker (322)



- How can you decouple the destination of a message from the sender and maintain central control over the flow of messages?
- Use a central Message Broker that can receive messages from multiple destinations, determine the correct destination, and route the message to the correct channel.



- It's a hub-and-spoke architectural style, but …
  - *Message Broker* isn't monolithic component. Internally, it uses the design patterns presented in the Routing chapter 7.
  - RabbitMQ is message broker

Alternatives to RabbitMQ?

- http://www.brokeragesdaytrading.com/article/848958483/activemq-vs-rabbitmq-vs-zeromq-vs-apache-qpid-vs-kafka-vs-ironmq-message-queue-comparision/
- http://christopher-batey.blogspot.dk/2014/07/rabbit-mq-vs-apache-qpid-picking-your.html
- https://dzone.com/articles/concise-comparison-rabbitmq

RabbitMQ

- Message Broker written in Erlang
- Can scale to over 15.000 messages pr. node pr. sec.
- Can have over 100 million concurrent queues

Fast like:

Chews messages like:

Source: rabbitmq.com

Connections & Channels

- By connecting to RabbitMQ, you're creating a TCP connection between your app and RabbitMQ.
- Once the TCP connection is open (and you've authenticated), your app creates an AMQP channel.
- This channel is a "virtual" connection inside the "real" TCP connection, and you issue AMQP commands over the channel
  - No limit to no. of AMQP channels (like bundle of fiber strands in fiber optic cable)
  - Channels ~Threads (i.e. no extra load on O/S's TCP stack)

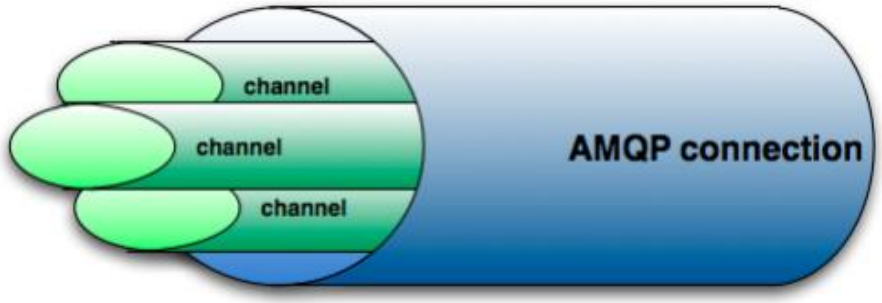
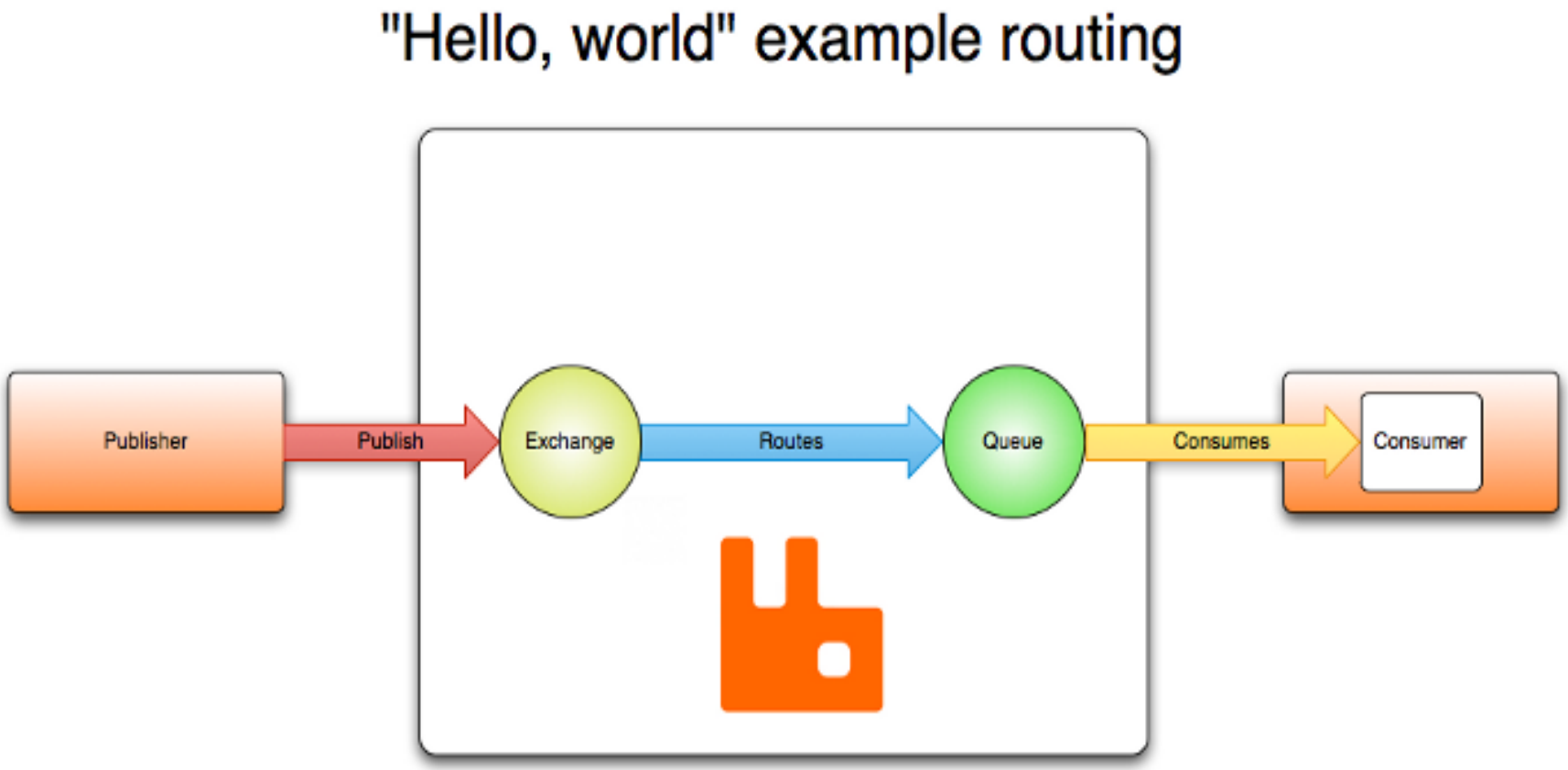


Figure 2.2 Understanding AMQP channels and connections.

Source: RabbitMQ in Action

AMQP Terminology



AMQP Terminology 2

- Imagine that you could only send mails to @something, not someone@something.

Exchanges are rule-based mailmen:

  - If someone sends you an e-mail, the exchange will try to find your mailbox based on rules.

Queues are like mailboxes:

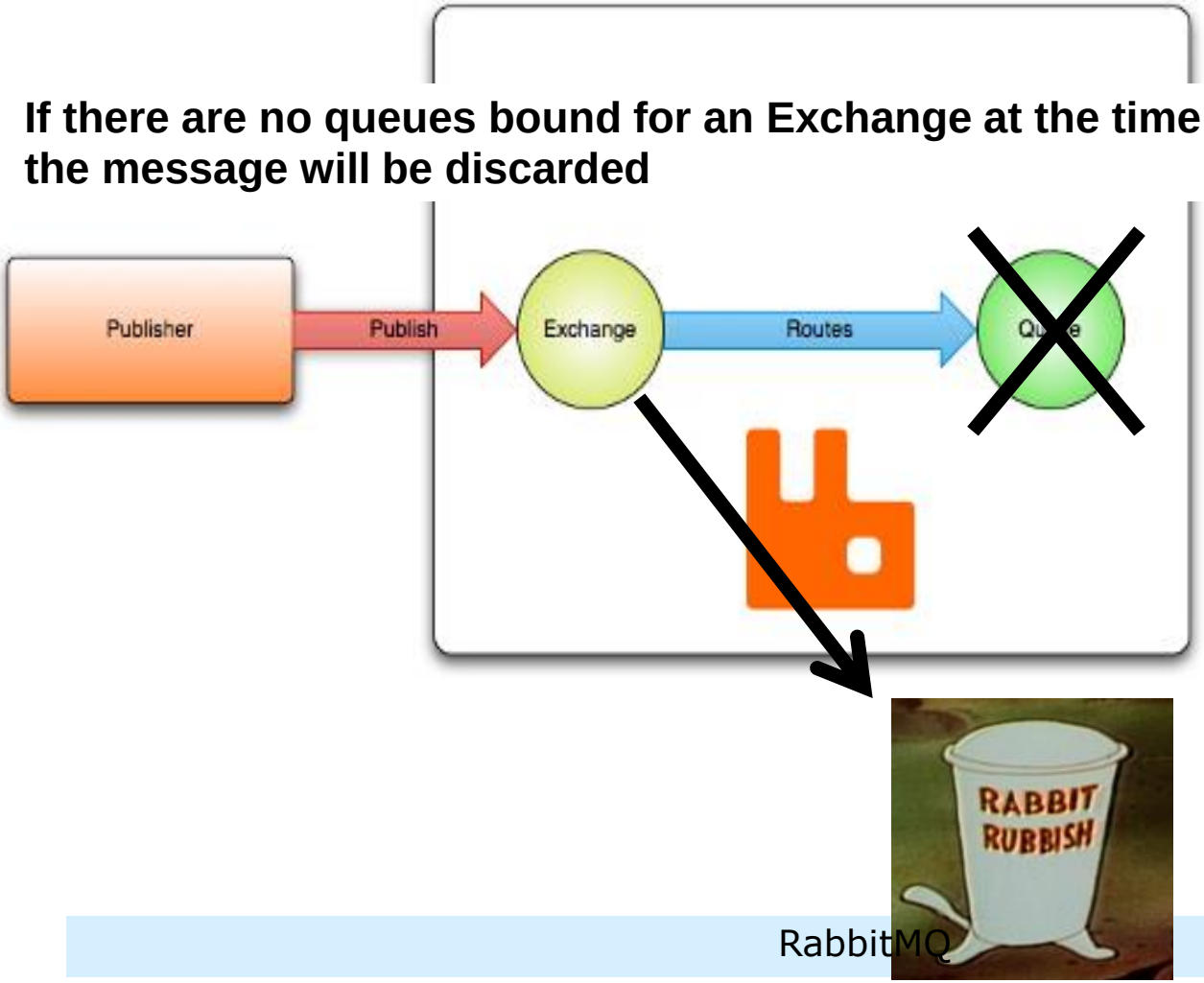
  - Queues are the 'someone' in the mail address.

- **Bindings** are like rules:
  - The way the mailman routes the messages to a specific queue based on a routing key.
    - If there is no queue to the exchange, the message is lost
- **Routing Keys** are like subjects:
  - In order to avoid having the AMQP server scan the full contents of all your messages, bindings only apply to routing keys

Exchange without Queue

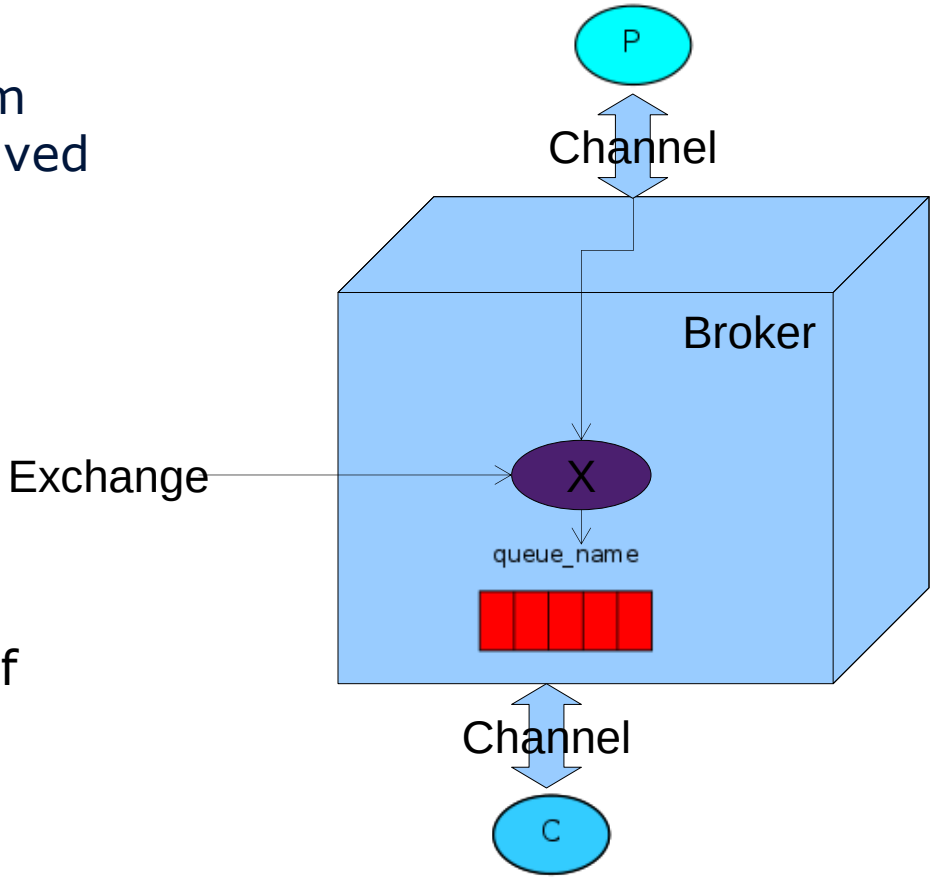
"Hello, world" example routing

If there are no queues bound for an Exchange at the time of publishing a message, the message will be discarded



Producer Consumer

- Messages are send from **P**roducer app and received by **C**onsumer app



- The queue is a buffer of messages

Break!

Producer Consumer Basic send EXAMPLE


```
public class Send {

  private final static String QUEUE_NAME = "hello";

  public static void main(String[] argv) throws Exception {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false, false, false, null);
    String message = "Hello World!";
    channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
    System.out.println(" [x] Sent '" + message + "'");

    channel.close();
    connection.close();
  }
}
```

1. Make the factory
2. Point at the broker
3. Make conn. to broker
4. Make a channel
5. Make a queue through the channel
6. Send the message

Producer Consumer Basic send 2

- But wait a minute
- Aren't we actually sending through the queue here?
  - No, RabbitMQ uses the default exchange to bind the queue.

```
String message = "Hello World!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
System.out.println(" [x] Sent '" + message + "'");
```

Producer Consumer Basic receive EXAMPLE


```
public class Recv {

  private final static String QUEUE_NAME = "hello";

  public static void main(String[] argv) throws Exception {

  ConnectionFactory factory = new ConnectionFactory();
  factory.setHost("localhost");
  Connection connection = factory.newConnection();
  Channel channel = connection.createChannel();

  channel.queueDeclare(QUEUE_NAME, false, false, false, null);
  System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

  QueueingConsumer consumer = new QueueingConsumer(channel);
  channel.basicConsume(QUEUE_NAME, true, consumer);

  while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println(" [x] Received '" + message + "'");
  }
 }
}
```

Same as in sender

Make the consumer
Attach the consumer to the queue

Polling consumer

Connections & Channels - again

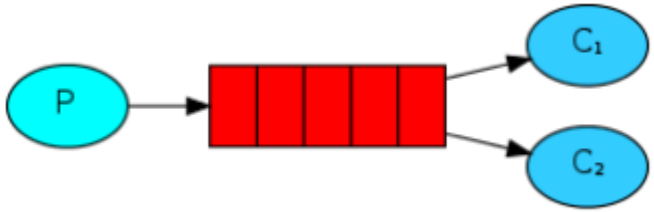
- Every channel has a unique ID assigned to it (your AMQP library of choice will handle remembering the ID for you)
- Whether you're publishing a message, subscribing to a queue or receiving a message, it's all done over a channel.

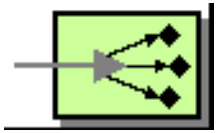
Channel	Virtual host	User name	Mode (?)	Prefetch	Unacked	Unconfirmed	Status
195.254.169.167:37579 (1)	/	guest		1	0	0	Idle
195.254.169.167:37598 (1)	/	guest		1	0	0	Idle
2.105.179.122:54510 (1)	/	tm		0	0	0	Idle

Demo

- Simple java program that sends, and receives
- The Http interface to Rabbit at http://datdb.cphbusiness.dk:15672

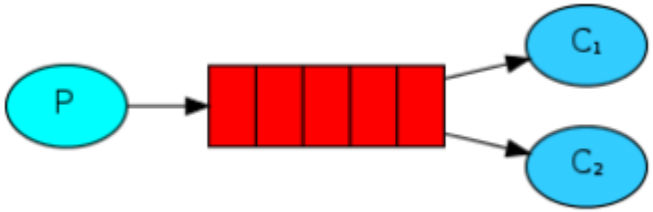
Work Queues (Competing Consumers)





- Distribution of work through several consumers
- Normally in round robin style
  - All consumers get equal amount of messages
  - Does not take into account that processing time could differ from message to message, leaving a large queue on one node, while idling others.

Work Queues – fair dispatch

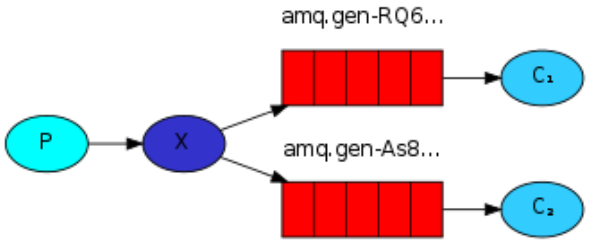


- Set the prefetchcount=1 in channel.basicQos to make sure that every consumer/queue only gets stacked one message at a time

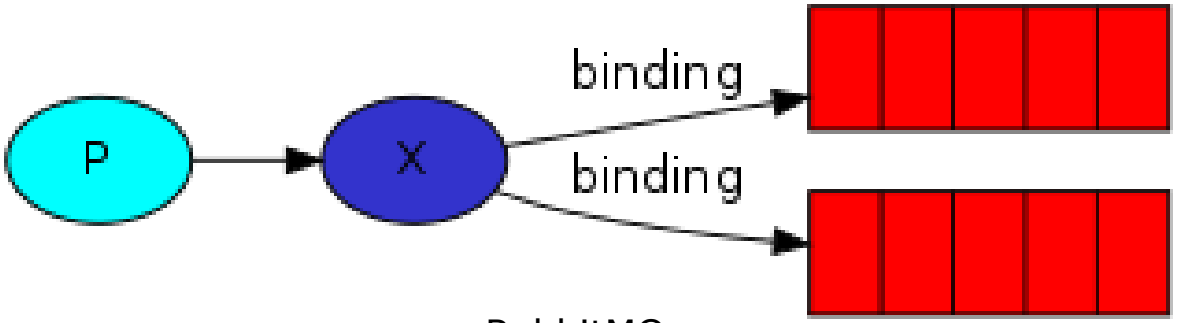
```
int prefetchCount = 1;
channel.basicQos(prefetchCount);
```

- Same codebase as with basic produce/consume

Publish Subscribe

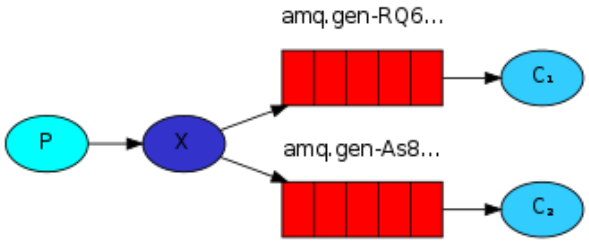


- **Fanout exchange** is used when we have one message to several consumers.
- Producer makes a name given exchange of the type Fanout
- Each Consumer makes a queue and binds the queue to the exchange.
- When the broker receives a message, it looks at the bindings for the exchange. If no bindings are made, it deletes the message. Otherwise it sends the message to the bounded queues.



RabbitMQ

Publish Subscribe - sender




```
private static final String EXCHANGE_NAME = "logs";

public static void main(String[] argv) throws Exception {

  ConnectionFactory factory = new ConnectionFactory();
  factory.setHost("localhost");
  Connection connection = factory.newConnection();
  Channel channel = connection.createChannel();

  channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

  String message = getMessage(argv);

  channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
  System.out.println(" [x] Sent '" + message + "'");

  channel.close();
  connection.close();
}
```

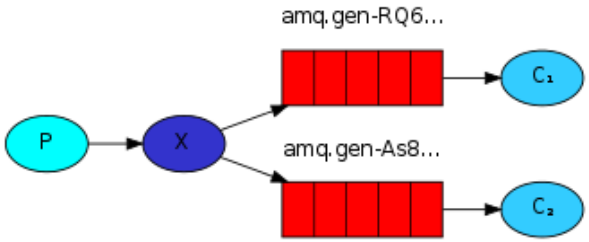
Same

Declare the exchange

Send the message

Not a word about the queues ☺

Publish Subscribe - receiver




```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, "");

System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(queueName, true, consumer);

while (true) {
  QueueingConsumer.Delivery delivery = consumer.nextDelivery();
  String message = new String(delivery.getBody());

  System.out.println(" [x] Received '" + message + "'");
}
```

Make the connection to the exchange

Make a temp queue with auto generated name

Bind the queue and exchange together

Demo

- Pub/sub app
- See the exchanges and queues in Rabbit admin console

.

Exercise

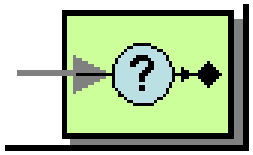
- Try out for yourself RabbitMQ tutorial 1-3:
  - basic send-receive
  - Competing consumes
  - Pub/sub

- Get the client jar files from either Maven, or RabbitMQ

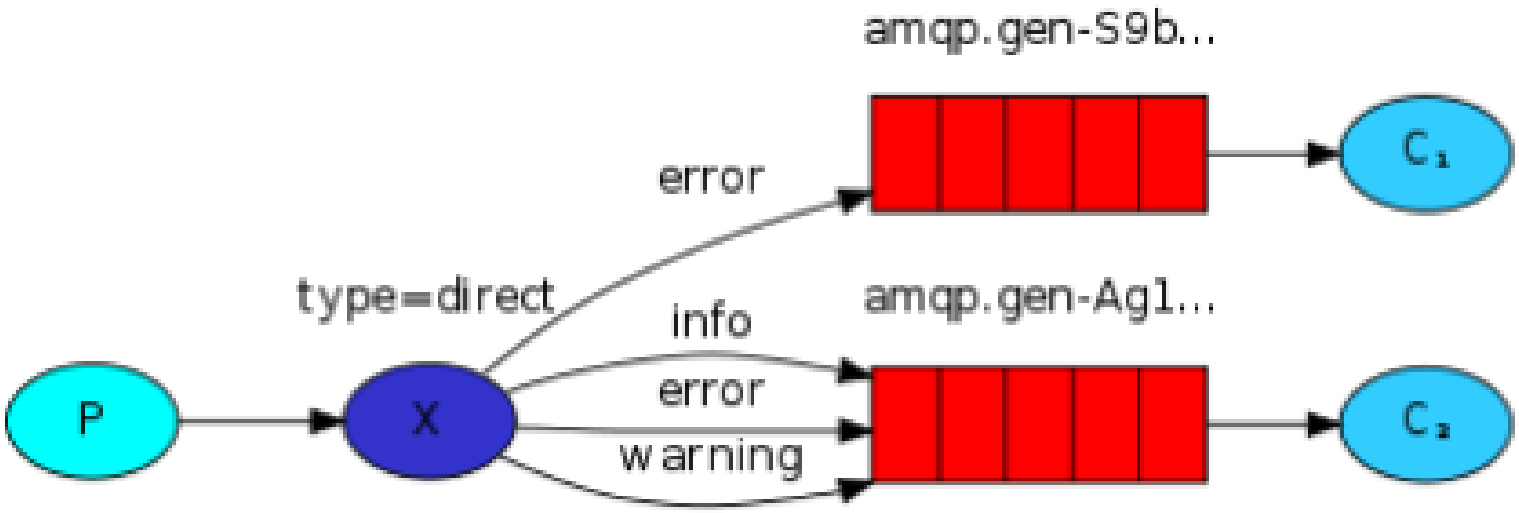
- See the exchanges and queues in Rabbit admin console

Break!

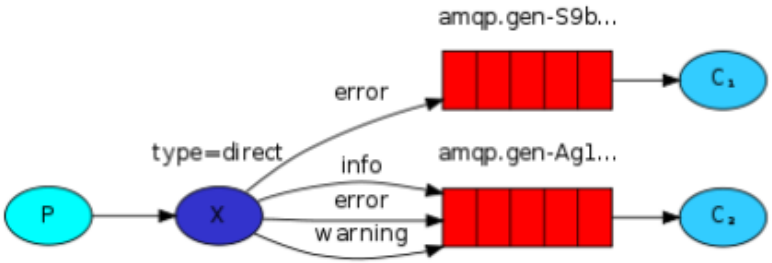
Routing (Selective Consumer) Key binding



- A way to selectively consume messages based on the routing key
- Distributes the messages based on the binding key. If queue has the same key K as message.routing_key R (K=R) then send the message to that queue.



Routing
Key binding




```
channel.exchangeDeclare(EXCHANGE_NAME, "direct");

String severity = getSeverity(argv);
String message = getMessage(argv);

channel.basicPublish(EXCHANGE_NAME, severity, null, message.getBytes());
System.out.println(" [x] Sent '" + severity + "':'" + message + "'");

channel.close();
connection.close();
}
```

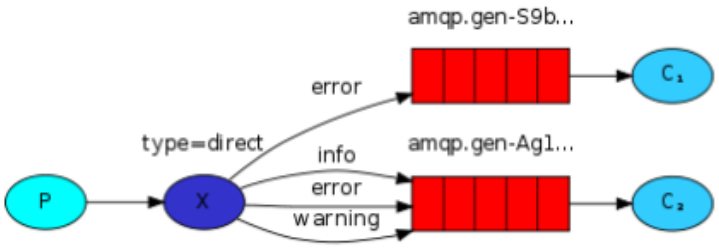
Make direct exchange

Make the routing key

Make the message

Send the message through the exchange with the given routing key

Routing
Key binding




```
private static final String EXCHANGE_NAME = "direct_logs";

 channel.exchangeDeclare(EXCHANGE_NAME, "direct");
 String queueName = channel.queueDeclare().getQueue();

 for(String severity : argv){
   channel.queueBind(queueName, EXCHANGE_NAME, severity);
 }

 System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

 QueueingConsumer consumer = new QueueingConsumer(channel);
 channel.basicConsume(queueName, true, consumer);

 while (true) {
   QueueingConsumer.Delivery delivery = consumer.nextDelivery();
   String message = new String(delivery.getBody());
   String routingKey = delivery.getEnvelope().getRoutingKey();

   System.out.println(" [x] Received '" + routingKey + "':'" + message + "'");
```

Make the exchange

Make an exchange binding for every routing key

Routing key = Severity level

Consumer signals message completion

Auto acknowledgment

- A way to be sure that every task that is sent, is also completed

```
QueueingConsumer consumer = new QueueingConsumer(channel);
boolean autoAck = false;
channel.basicConsume("hello", autoAck, consumer);

while (true) {
  QueueingConsumer.Delivery delivery = consumer.nextDelivery();
  //...
  channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
}
```

To reject a message call one of the following:

```
channel.basicNack(delivery.getEnvelope().getDeliveryTag(), false, true);
//or
channel.basicReject(delivery.getEnvelope().getDeliveryTag(), true);
```

RECAP!

Exchange Types

Fanout

- “Dumb” forwarding messages into the bound queues. Makes copies of the message

Default

- Takes the queue name and make it into the binding key on the default exchange. Therefore it looks as if you can connect directly with the queue.
- This exchange type is used in the producer/consumer and work queue example.

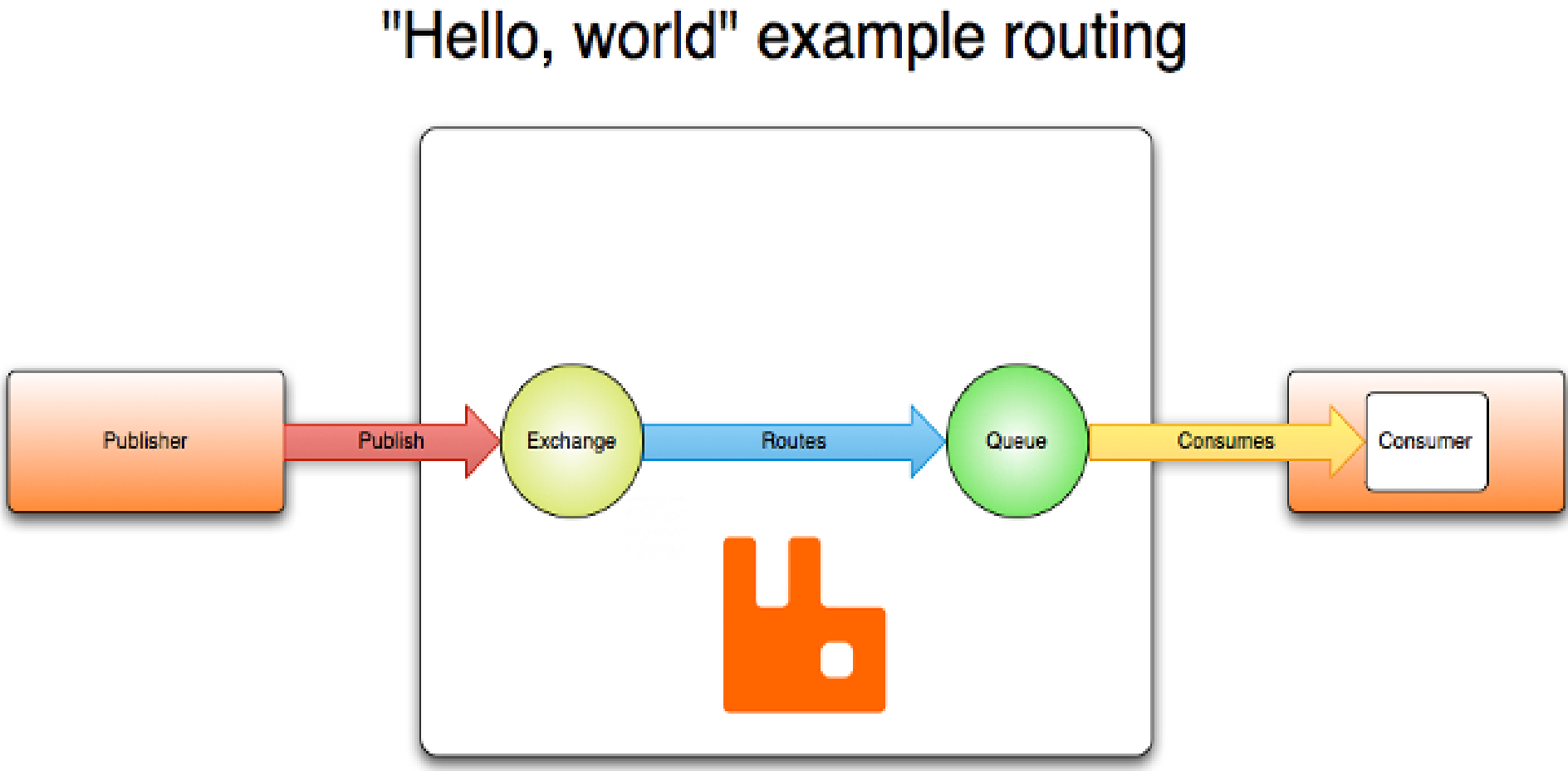
Direct

- Distributes messages based on binding key. If queue has the same key K as message.routing_key R (K=R), the message is sent to that queue.

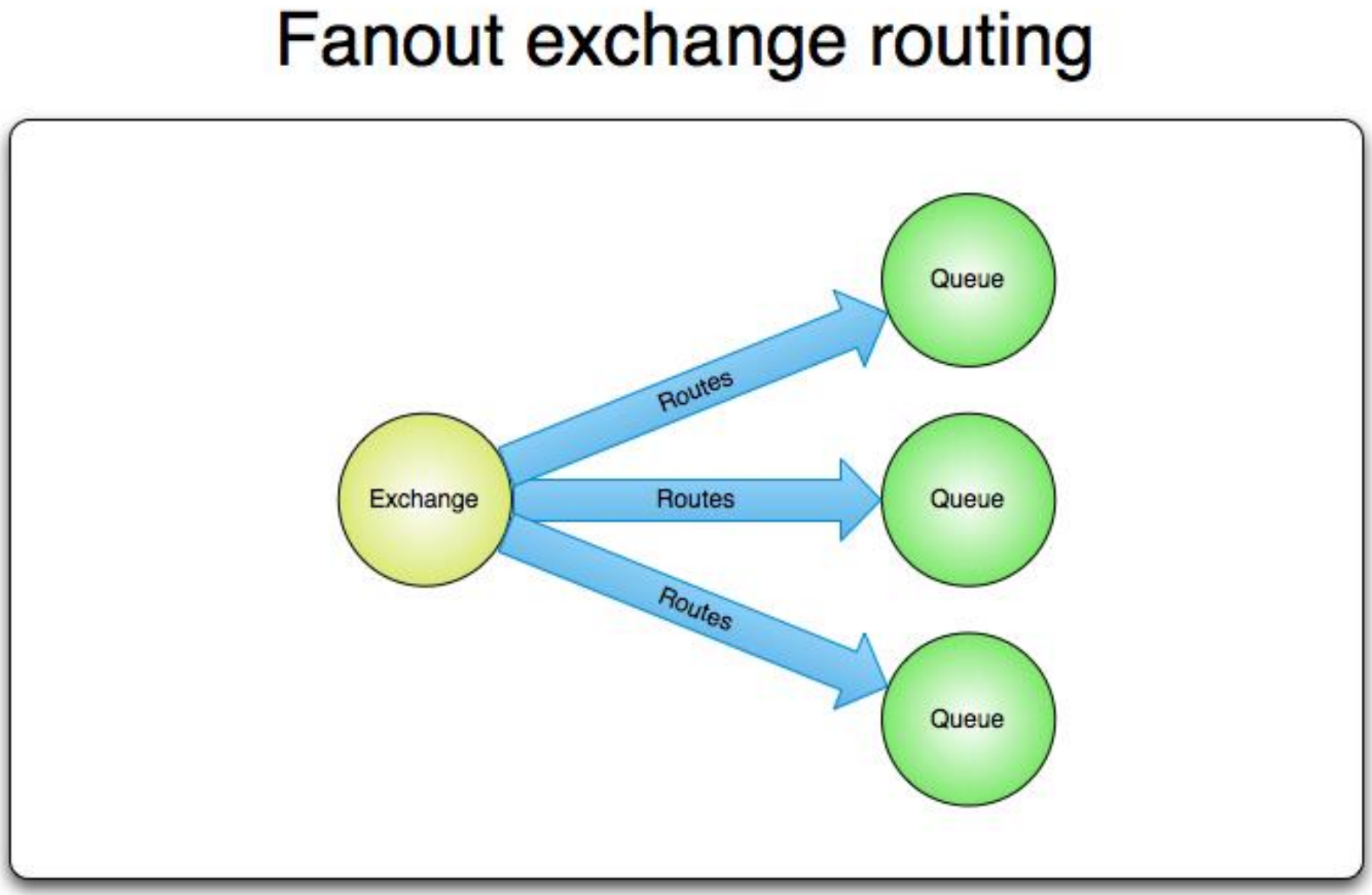
Topic

- Covered very soon (today)

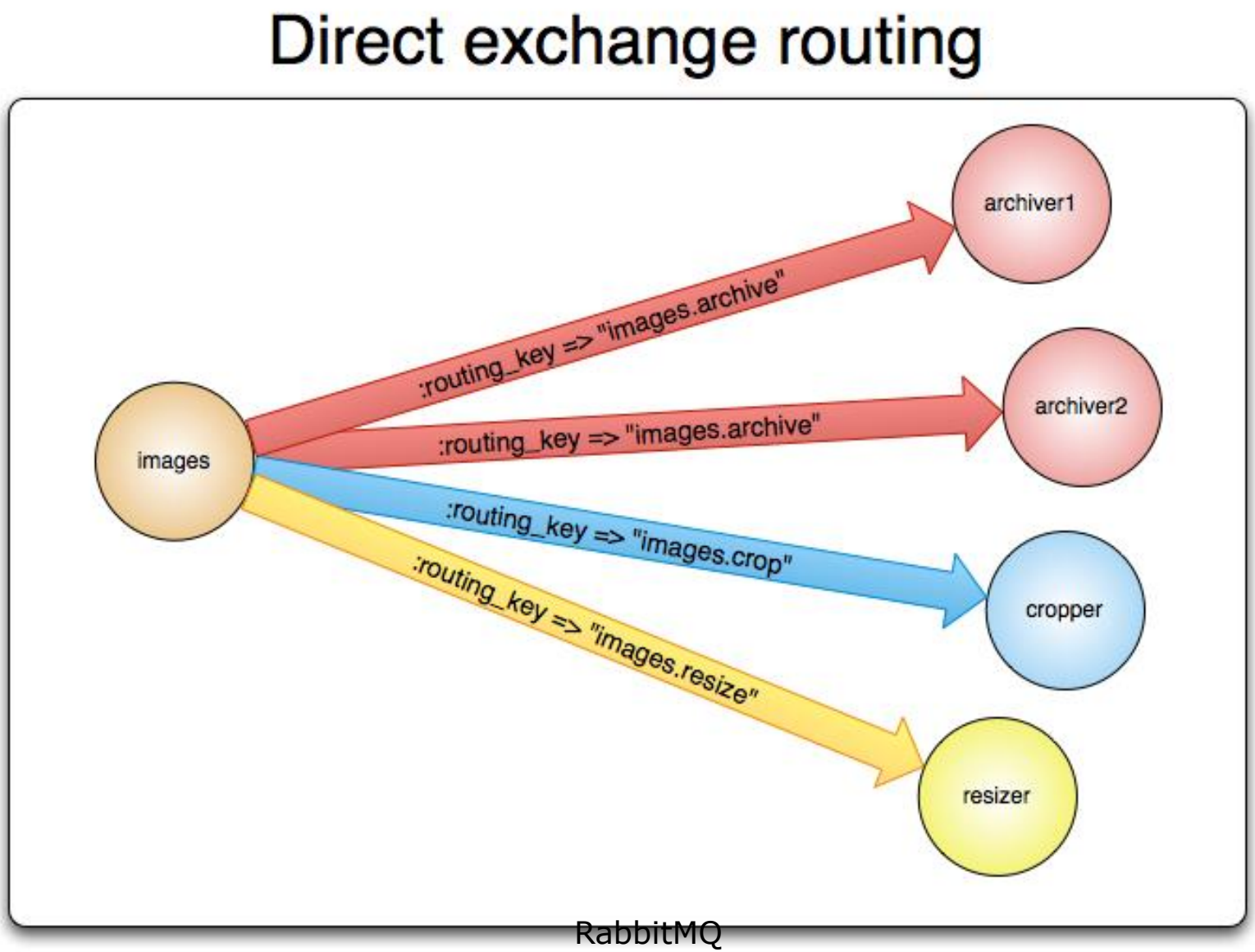
Terminology



Terminology

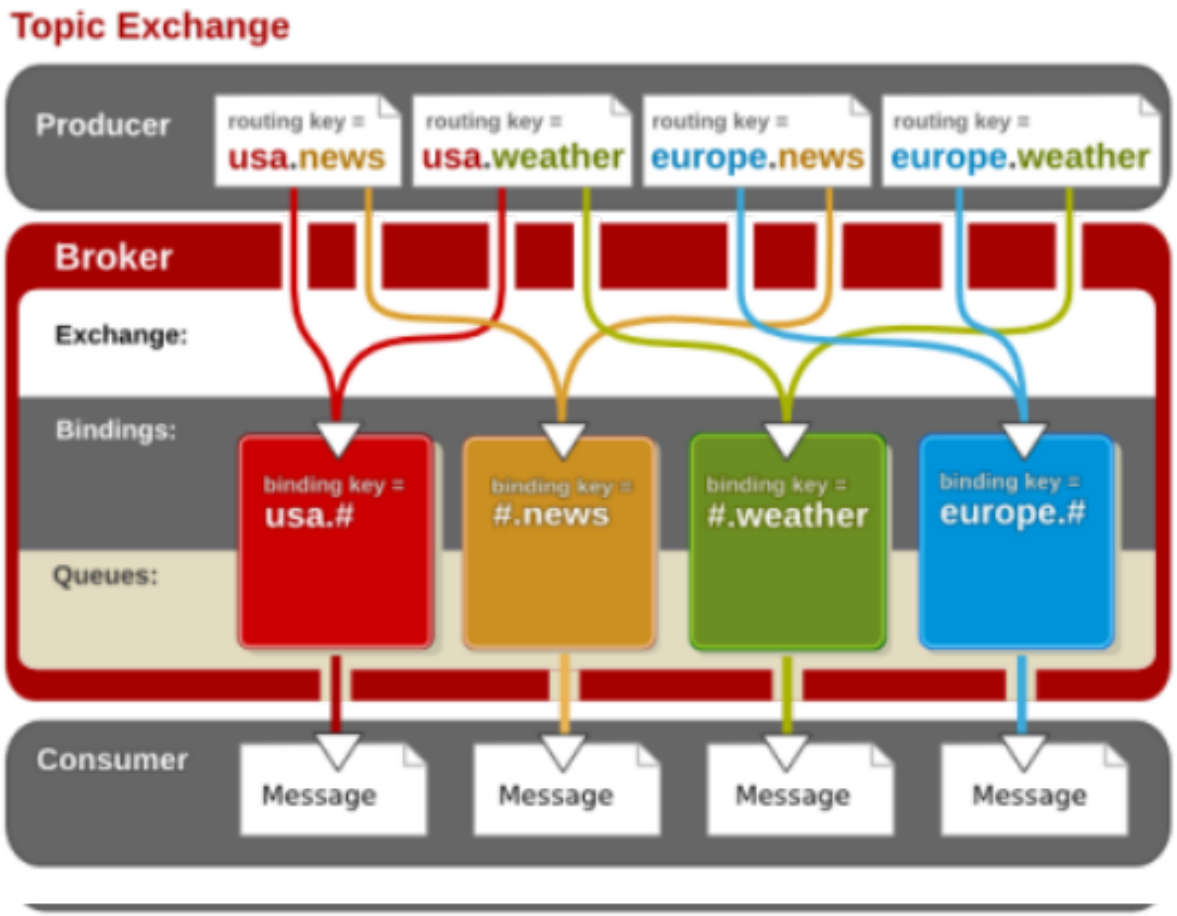


Terminology



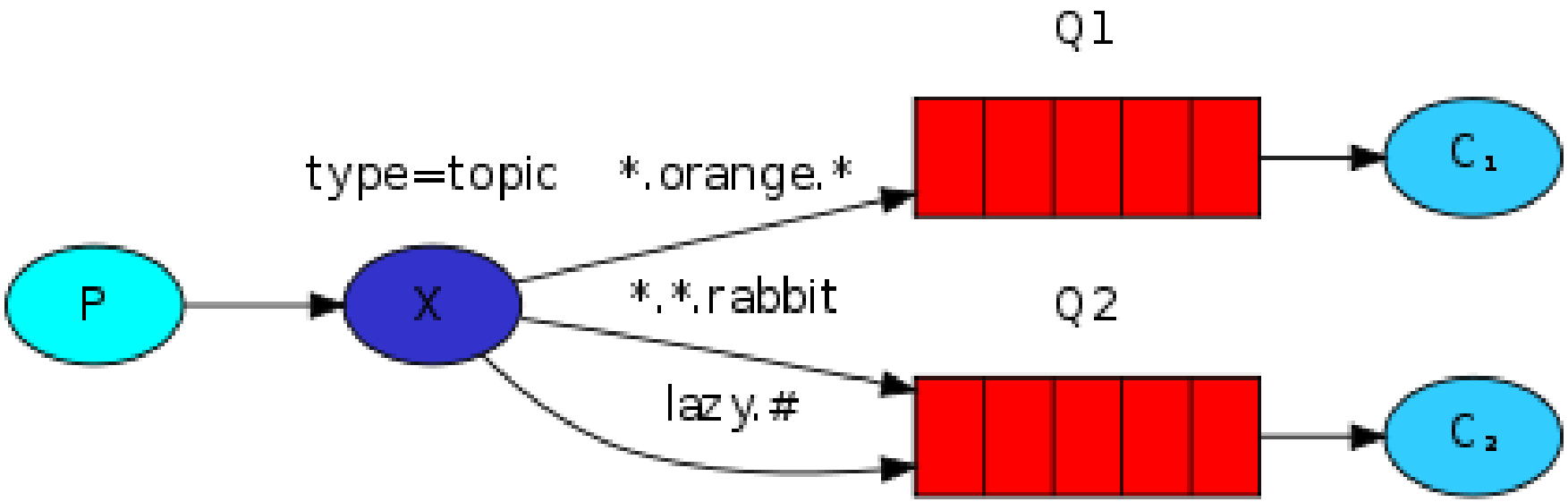
Topic Exchange

- Fine grained way of distributing messages.
- Example:



Binding key formats

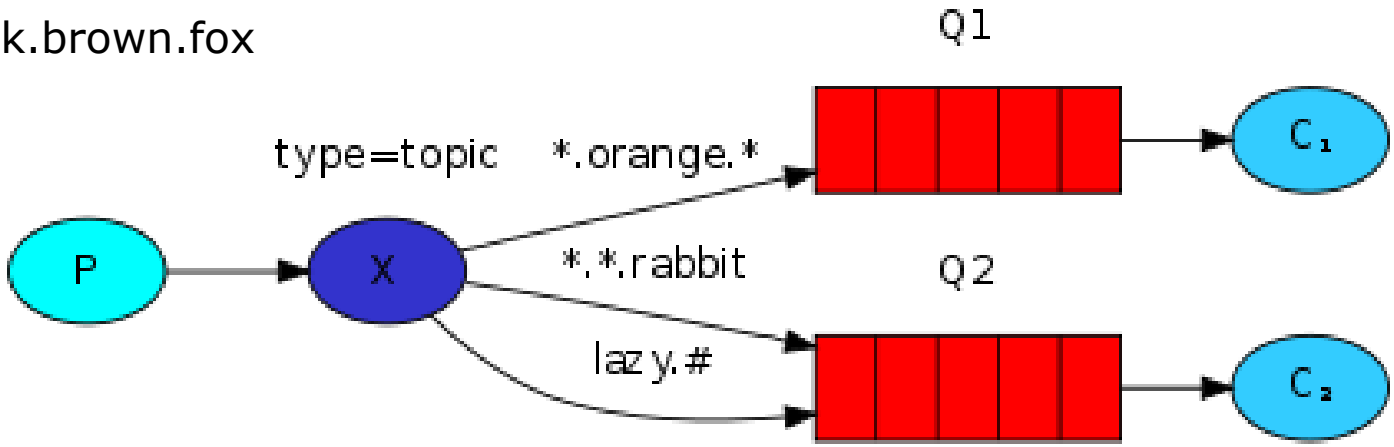
- Keywords can separated with dots like “error.server2.tomcat”
- 2 special cases routing keys:
  - * (star) for exactly one word
  - # (hash) for zero or more words



Who will get these messages?

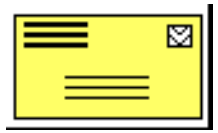
Rounting key examples:

quick.orange.rabbit
lazy.orange.elephant
quick.orange.fox
lazy.brown.fox
lazy.pink.rabbit
quick.brown.fox



Tutorial code in Java here: https://www.rabbitmq.com/tutorials/tutorial-five-java.html

Reply-to channel



- Reply to is set in the properties of the message.
- You need to do the following:

```
Builder builder = new BasicProperties.Builder();
builder.replyTo(queueReply1);
BasicProperties props =builder.build();
channel.basicPublish("", task1Queue, props,("msg nr: "+i).getBytes());
```

Make a builder

Set .replyTo to the queue or exchange you want the reply to.

Build the new Properties.

Add them to the message

- It's the same for
  - correlation id
  - message id
  - and other AMQP meta data fields.

Resources

- Tutorials at http://www.rabbitmq.com/getstarted.html
- Rabbit in Action chap. 4 (examples in Python) at github
- Netbeans examples in Java at github
- RabbitMQ .NET Client Library User Guide at http://www.rabbitmq.com/releases/rabbitmq-dotnet-client/v2.1.0/rabbitmq-dotnet-client-2.1.0-user-guide.pdf

Exercises for now + until next time

1. Get RabbitMQ examples up running:
   - Simple Hello World
   - Competing consumers (work queues with equal distribution of messages)
   - Pub-sub with a ”dumb” fanout
   - Routing with direct exchange

2. Try both polling consumer (my examples) and asynchronous callback version with `DefaultConsumer` (see examples on https://www.rabbitmq.com/tutorials/tutorial-one-java.html

3. Make student enrollment simulation in RabbitMQ: client app sends request for enrollment. Admin app sends response (accepted/rejected). See next slide.

Student enrollment exercise in RabbitMQ

- Make student enrollment integration solution using RabbitMQ
  - Client app sends request for student enrollment
  - Admin app sends response (accepted/rejected)

- **Version 1:**
  - As above – one producer & one consumer plus invalid letter channel
- **Version 2**
  - Two admin consumers – both for AP students
- **Version 3**
  - More admin consumers
    - One for AP students
    - One for PBA students
    - One for international students