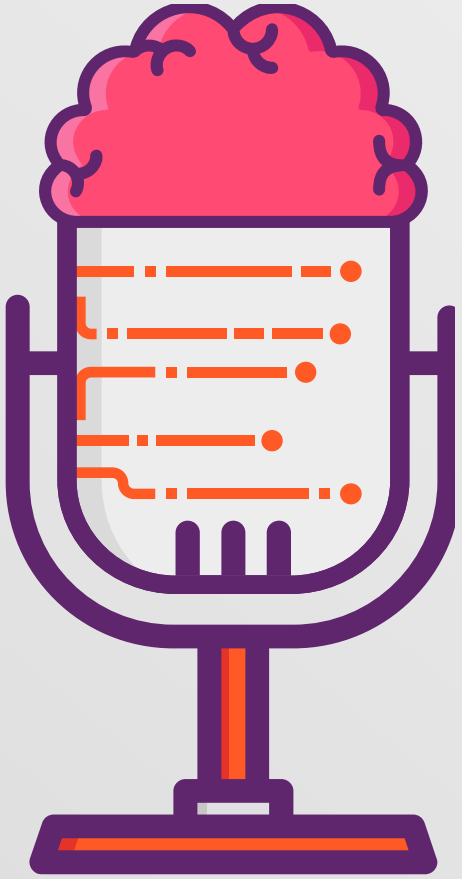
the NeRd Talks

RabbitMQ

Relatore: Andrea Tamburrino



RabbitMQ

An open-source message broker

Outline

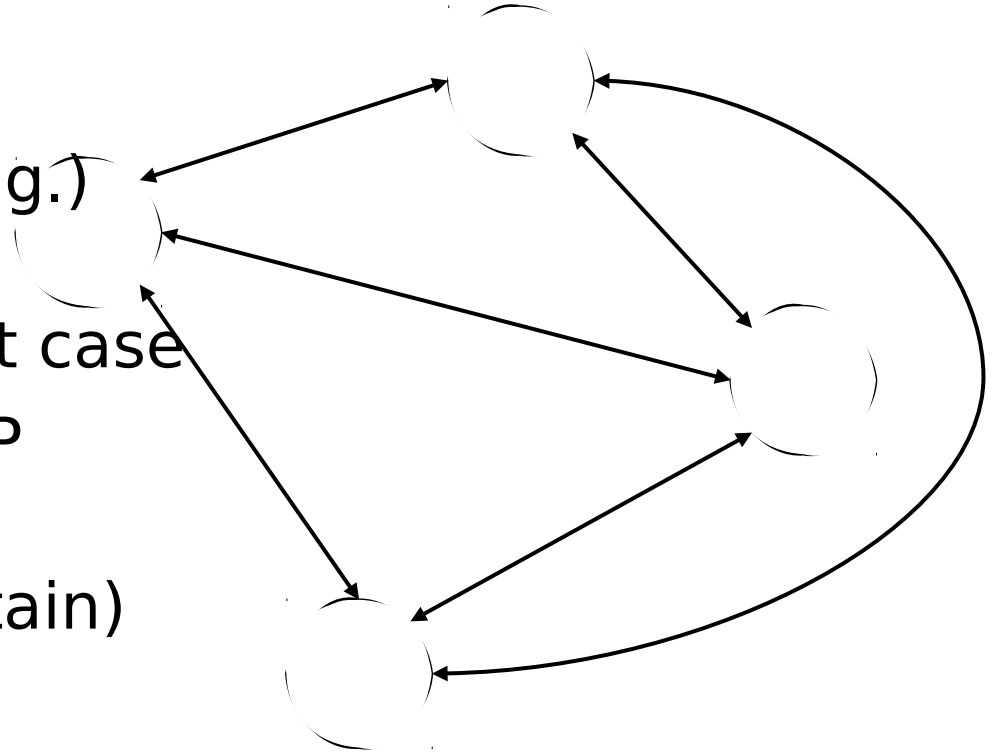
- Inter-process communication
- Message broker
- Advanced Message Queuing Protocol
- RabbitMQ
- A use case from Kalliope

Inter-process communication

- Signals
- Pipes
- Shared memory
- Sockets
- Message queues
- ...

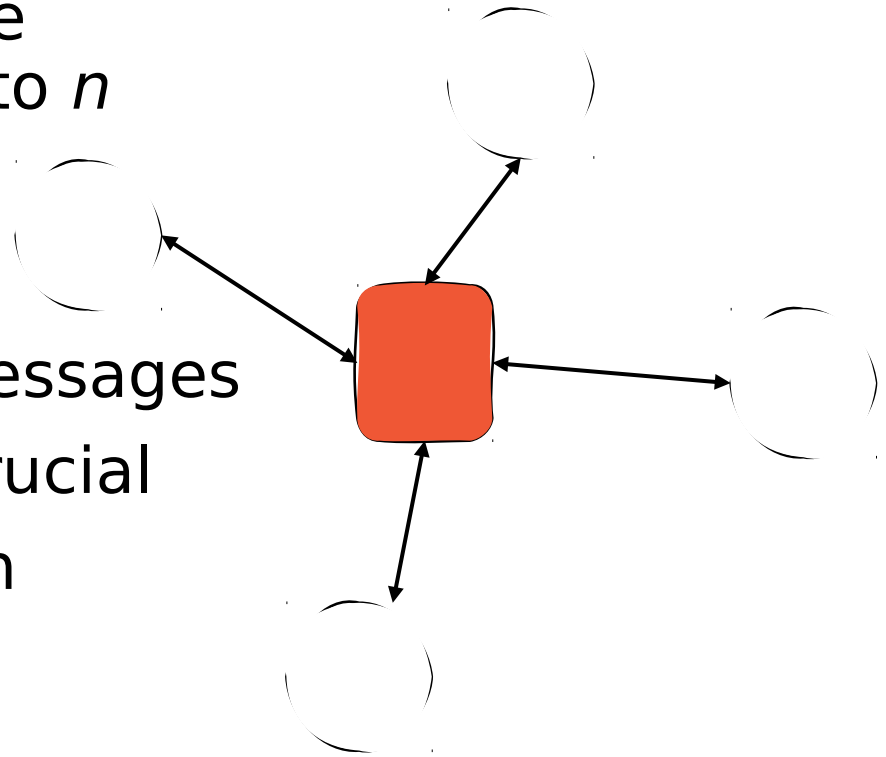
Inter-process communication

- Assume *n* processes need a way to communicate with each other
- A solution may be to create a TCP (e.g.) connection between each pair
- *n*(*n*-1)/2 TCP connections in the worst case
- Each process has to maintain *n*-1 TCP connections in the worst case
- Think if you have to write (and maintain) the source code for these services
- Does not scale very well



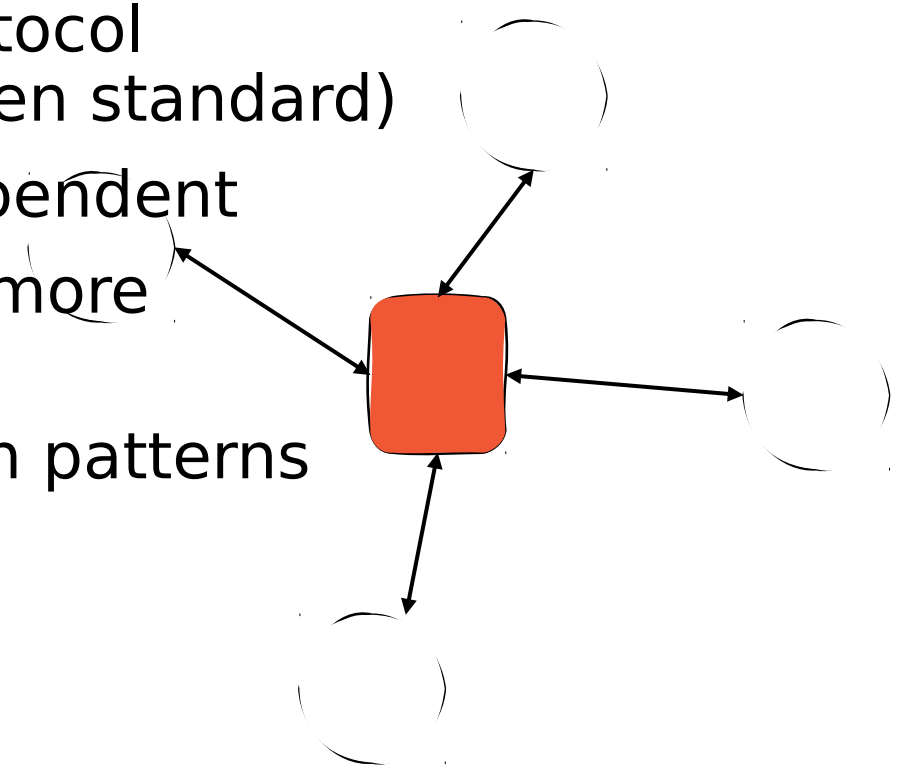
Inter-process communication

- If we have «something» in the middle
  we can reduce communication links to *n*
- A shared filesystem
- A shared database
- Processes should avoid polling for messages
- Efficient message delivery may be crucial
- A *message broker* is a better solution



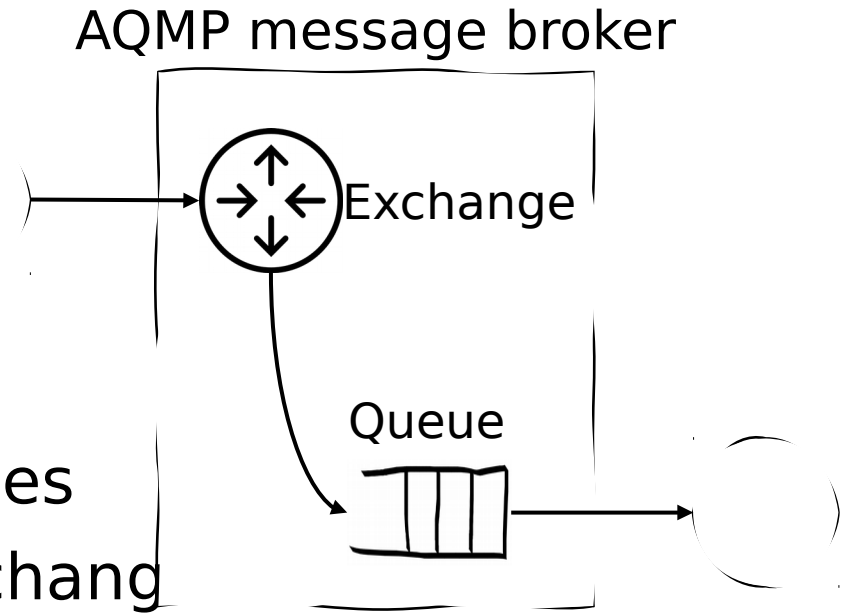
Message broker

- A message broker implements a protocol for messages exchange (better if open standard)
- Communication is architecture-independent
- Provides message routing to one or more destination processes
- Implements common communication patterns such as
  - publish/subscribe
  - point to point
  - competing consumers
  - ...



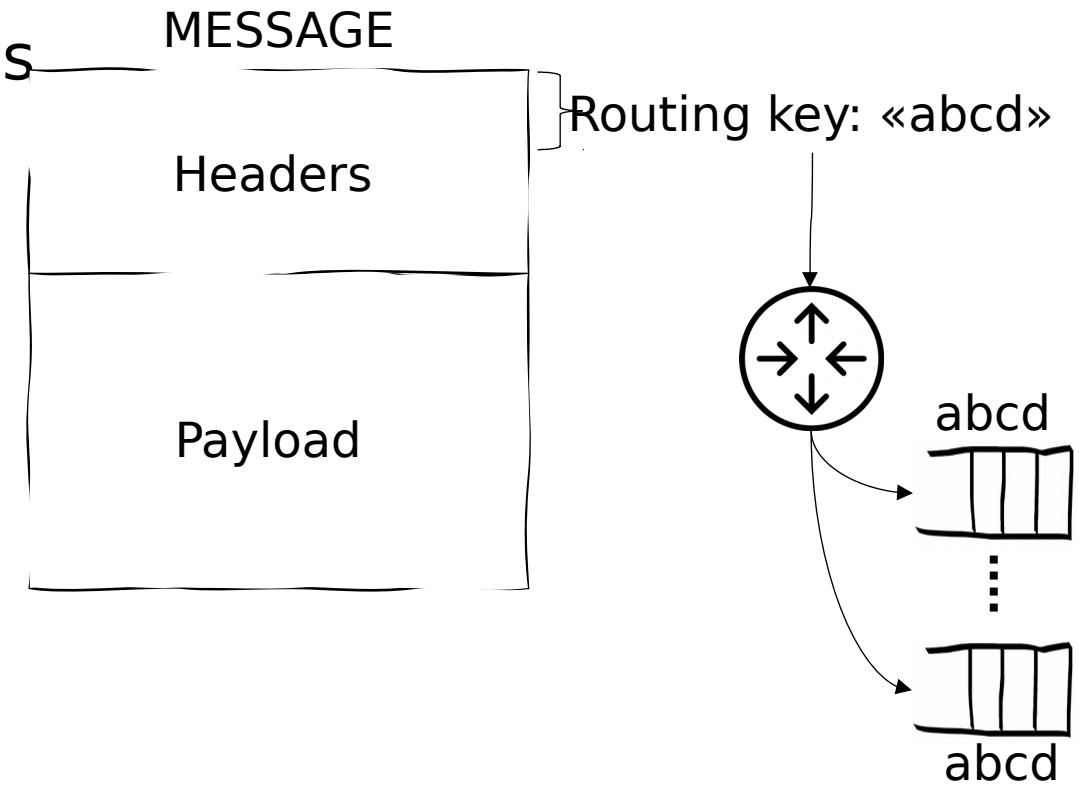
AMQP

- Advanced Message Queueing Protocol
- Defines *exchanges*, *queues* and *bindings*
- An exchange is a message routing agent that implements some routing logic
- Queues (trivially) are message buffers from which a client can obtain its messages
- Bindings are used to «link» queues to exchang and thus represent the «rules» followed by the routing logic of the exchange



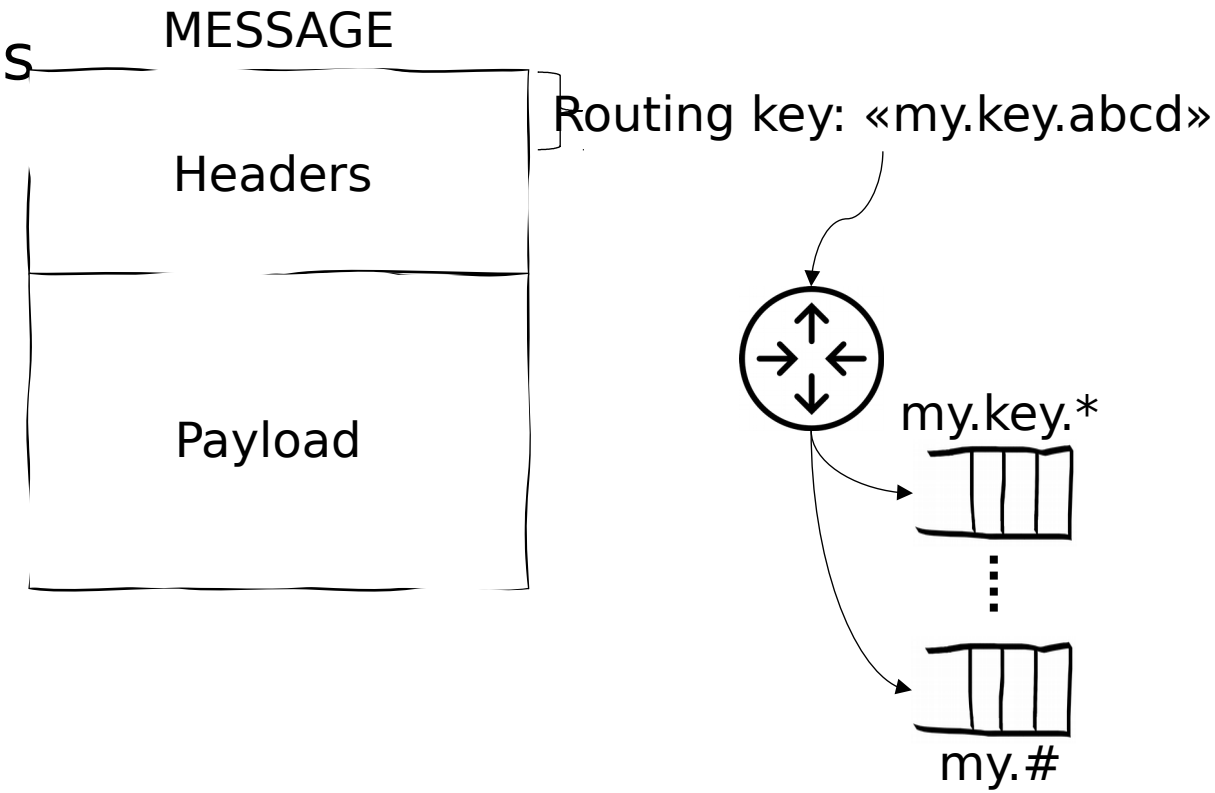
AMQP

- Echanges can be of different types according to their routing logic mechanisms
- *Direct exchange*
  Message is delivered to queue(s) that have a binding key exactly equal to the message routing key



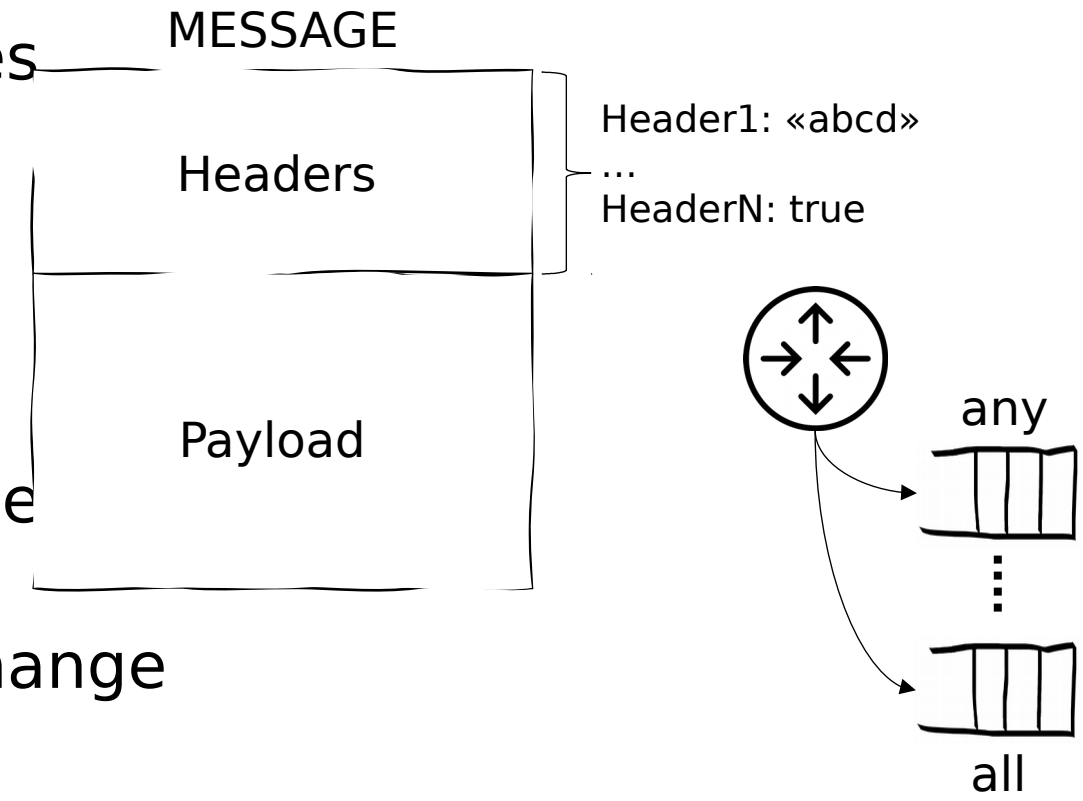
AMQP

- Echanges can be of different types according to their routing logic mechanisms
- *Topic exchange*
  Message is delivered to queue(s) if routing key matches a pattern indicated in queue's binding key
- Routing key is required to be a dot-separated list of words (e.g. *my.routing.key.abcd*)
- Useful for multicast delivery



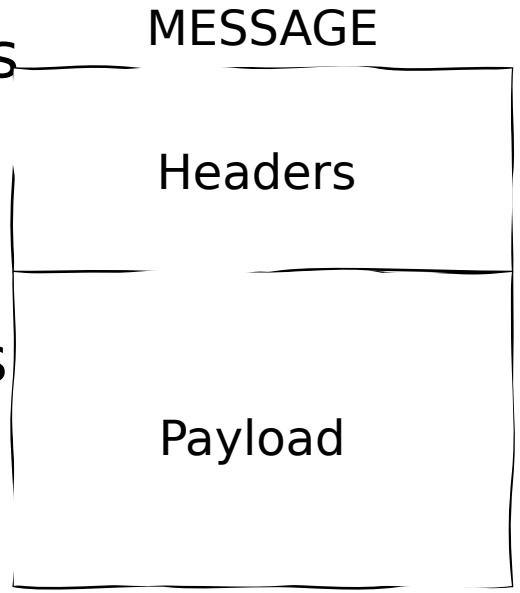
AMQP

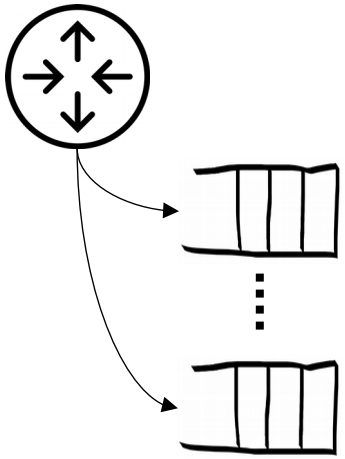
- Echanges can be of different types according to their routing logic mechanisms
- *Headers exchange*
  Message is delivered to queue(s) if one or more values of headers match the ones indicated in queue binding
- Not very different from topic exchange
- Useful for «filtered» delivery



AMQP

- Echanges can be of different types according to their routing logic mechanisms
- *Fanout exchange*
  Message is delivered to all queues bound to the exchange
- Useful for broadcast delivery



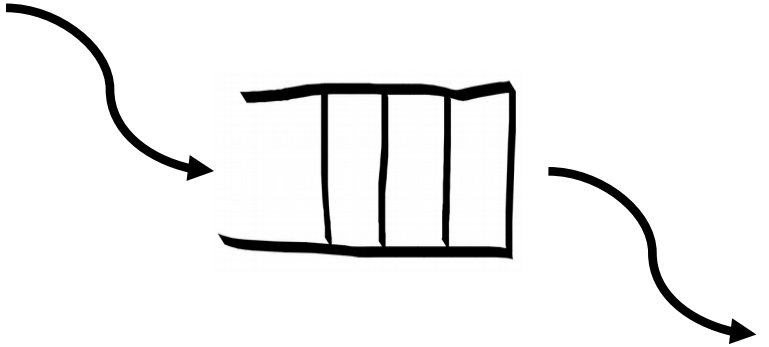


RabbitMQ

- RabbitMQ is an open-source message
- Written in Erlang
- Implements AMQP (up to 0.9.x)
- A client only needs one TCP (TLS) con
- Faces scalability and reliability issues (e.g. single point of failure)
- HA mechinism
- Mirrored queues
- Clustering

RabbitMQ

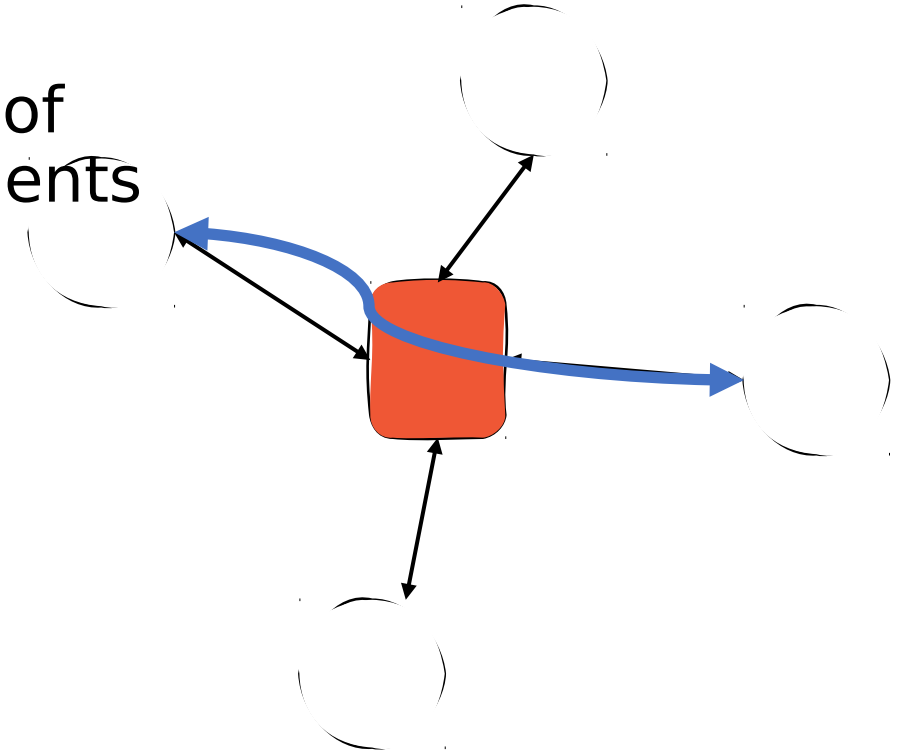
- Queues can have different attributes
- *Exclusive*
  The queue is used by only one connection (no concurrent consumers) and it is deleted when connection is closed



- *Durable*
  Will survive a broker restart, including its messages if marked *durable* as well (persistent storage)
- *Auto-delete*
  It is automatically removed when its last consumer unsubscribes
- Other «security» attributes
  messages TTL, max queue size, max number of messages, drop policies…

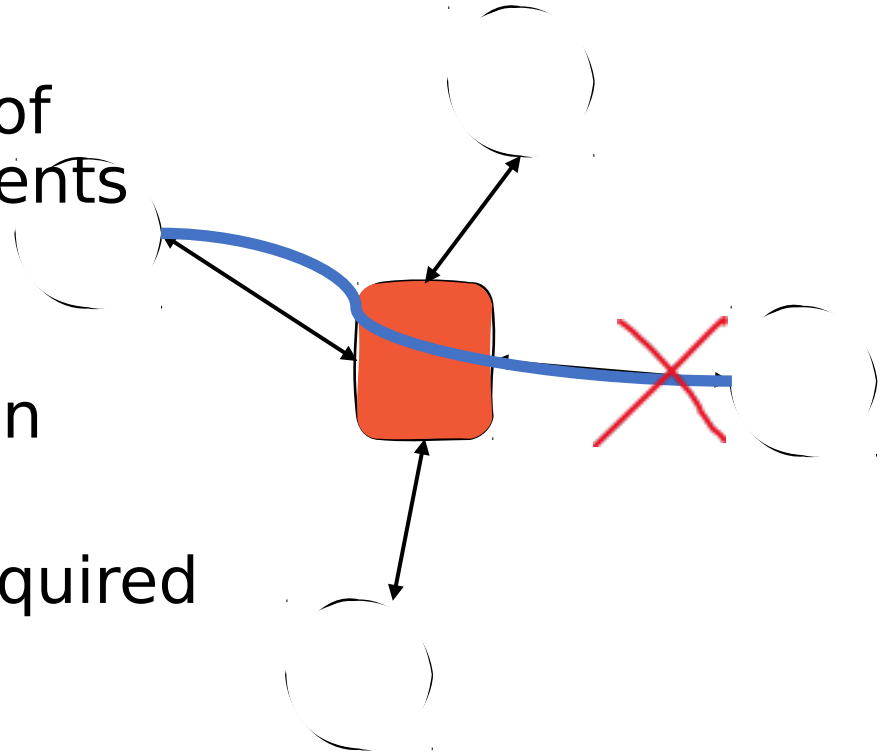
RabbitMQ

- Clients are completely decoupled
- A broker is not required to be aware of communication coupling between clients (without a broker we would have direct connections between clients)



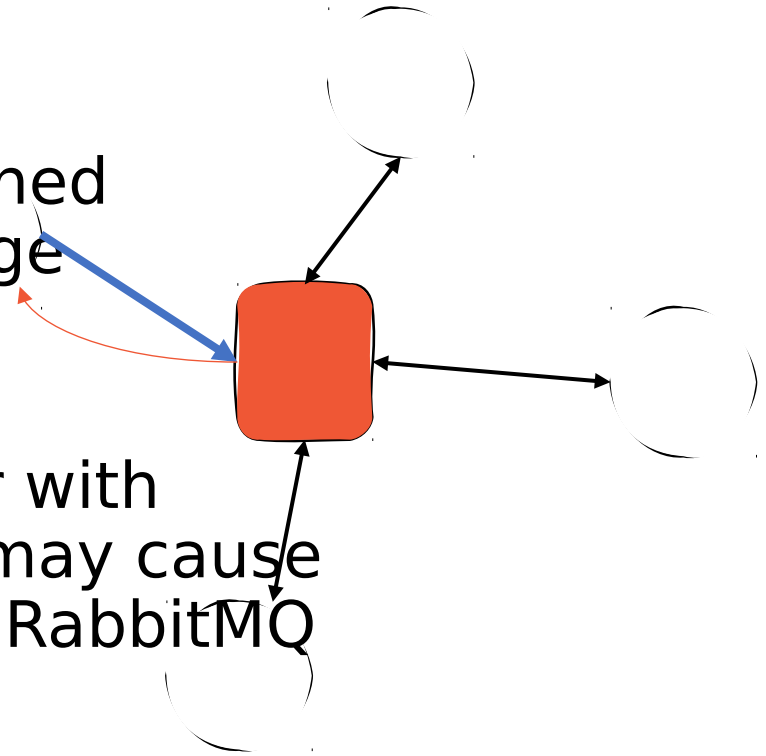
RabbitMQ

- Clients are completely decoupled
- A broker is not required to be aware of communication coupling between clients (without a broker we would have direct connections between clients)
- How can a client be aware of a broken connection with another client?
- An application-level mechanism is required



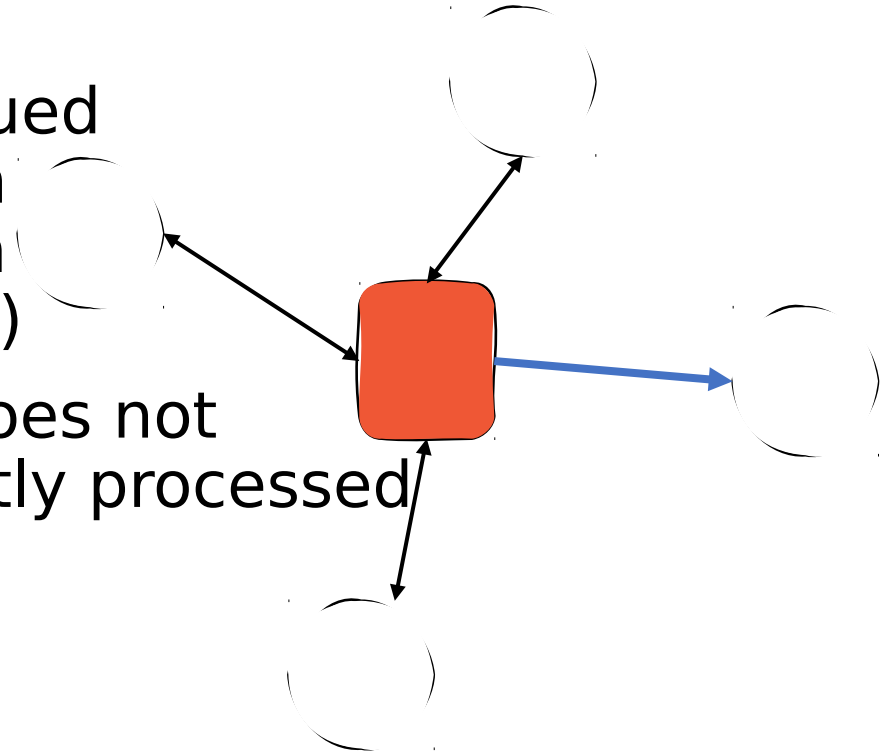
RabbitMQ

- If enabled, *publish confirms* can be sent by RabbitMQ to the publisher
- Sent after RabbitMQ has correctly published the message on a queue (correct message transmission by publisher to RabbitMQ through socket is not a guarantee)
- E.g. if a queue was declared exclusive or with auto-delete property, a connection loss may cause it to be removed and publishing will fail: RabbitMQ will send a nack to the publisher



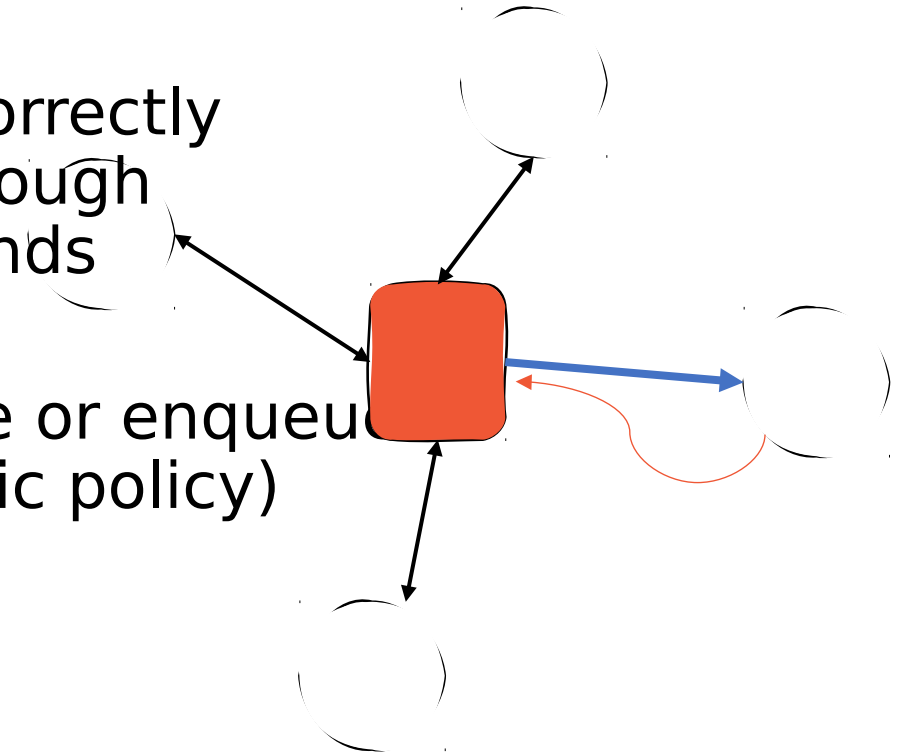
RabbitMQ

- Two types of acks consumer-side
- *Auto-ack*: RabbitMQ considers a queued message consumed once it has been correctly sent to a consumer through socket connection (i.e. not a real ack)
- It is very efficient, of course, but it does not guarantee that consumer has correctly processed the message



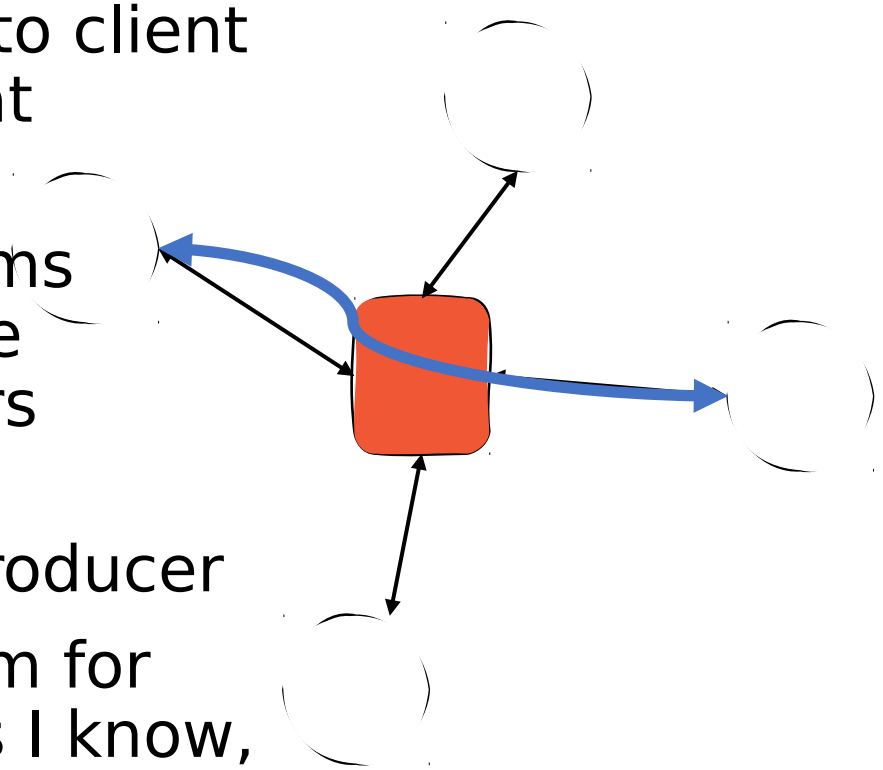
RabbitMQ

- Two types of acks consumer-side
- *Explicit consumer ack*: once it has correctly processed the message received through the socket connection, consumer sends an explicit ack/nack
- RabbitMQ removes it from the queue or enqueu it back again (it depends on a specific policy)
- Cumulative acks are also allowed



RabbitMQ

- In conclusion, a 100% reliable client to client transmission must be implemented at application level
- A solution (not always feasible in terms of real-time performances) is to make both clients consumers and producers
- In case of one-way communication a consumer will only be an ack/nack producer
- Probably AMQP provides a mechanism for reliable communication but, as far as I know, RabbitMQ does not implement it yet



RabbitMQ

Log out

RabbitMQ 3.4.1, Erlang 17.3.2

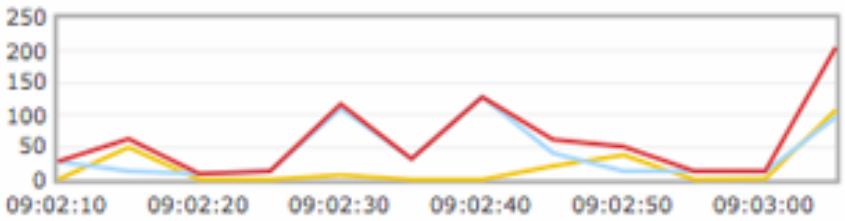
Overview | Connections | Channels | Exchanges | Queues | Admin

Virtual host: All

Overview

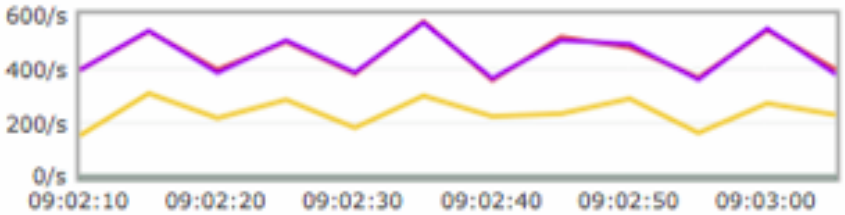
Totals

Queued messages (chart: last minute) (?)



Ready	0 msg
Unacked	12 msg
Total	12 msg

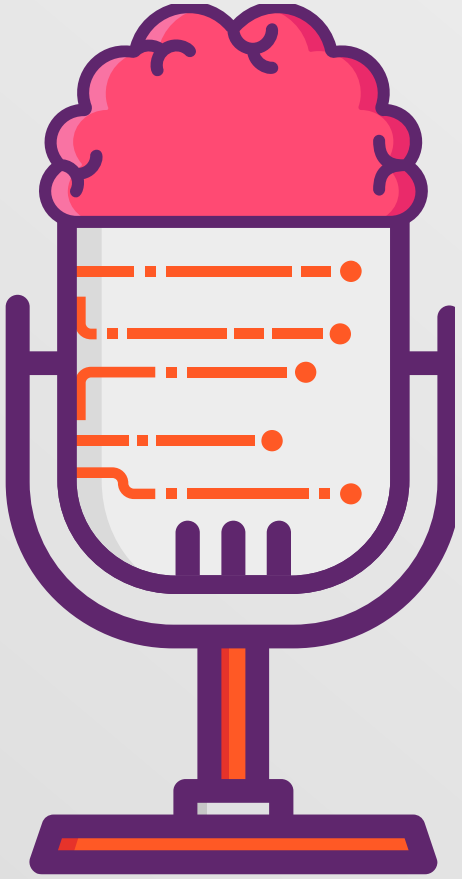
Message rates (chart: last minute) (?)



Publish	230/s
Confirm	0.00/s
Deliver	397/s
Redelivered	0.00/s
Acknowledge	379/s
Get	0.00/s
Get (noack)	0.00/s

References and useful links

- rabbitmq.com
- www.amqp.com
- Our Qt Rabbit client: https://gitlab.netresults.dev:10443/netresults/prodotti/kalliope/pbx/fw/daemons/common_classes/-/tree/master/rabbit



Thank you for your attention!

Questions?