

# Message Queues

# Why?

# Warum überhaupt?

## **Verteilte Systeme**

- Kommunikation
- Koordination
- Vereinfachung entkoppelter Applikationen
- Verbessern Performance
- Verbessern Robustheit
- Verbessern Erweiterbarkeit

# Was machen Sie?



- Nachrichten werden gespeichert bis sie verarbeitet oder gelöscht werden.
- Jede Nachricht wird nur ein einziges Mal von genau einem Consumer verarbeitet.
- Sollte das System abstürzen, sind die Nachrichten persistiert und überleben so den Absturz.

# Entkopplung

# Entkopplung

- Kopplung beschreibt wie sehr ein Teil einer Applikation von einem anderen Teil abhängt.
- Entkopplung heißt Funktionen von einander zu trennen, sodass sie auch unabhängig von einander funktionieren.
- Sie können auch in komplett verschiedenen Sprachen geschrieben sein und auf unterschiedlicher Hardware laufen.

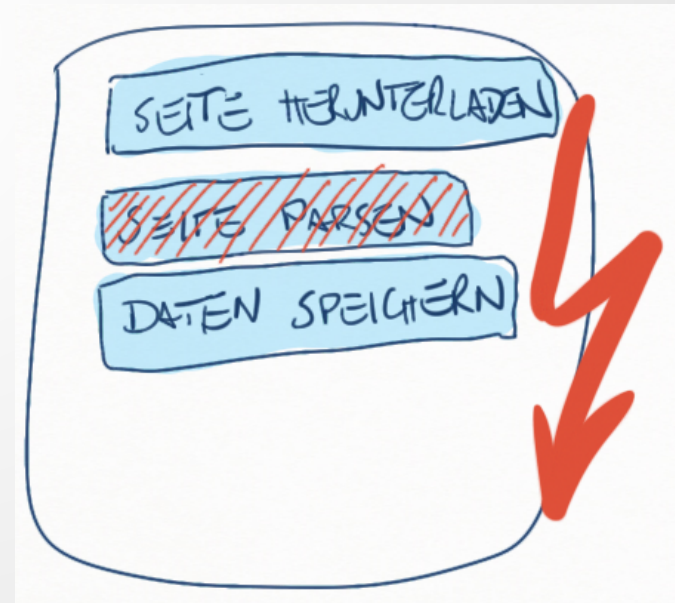
# Beispiel: Web-Scraper

- Sie entwickeln einen Web-Scraper
  - Seiten herunterladen
  - Seiten parsen und Daten speichern



# Beispiel: Web-Scraper

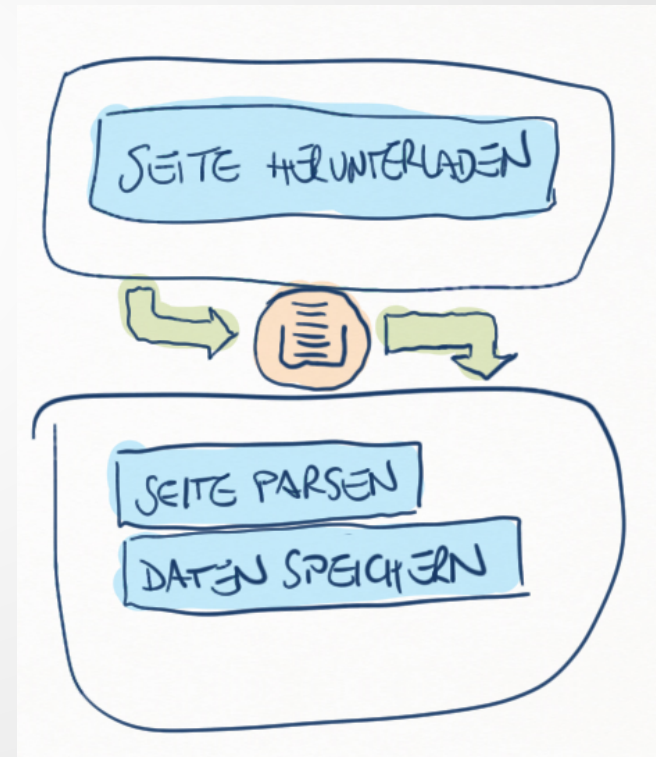
- Wenn jetzt der Parser abschmiert, steht das gesamte Programm
- Auch Webpages werden keine mehr heruntergeladen und Daten können dadurch verloren gehen





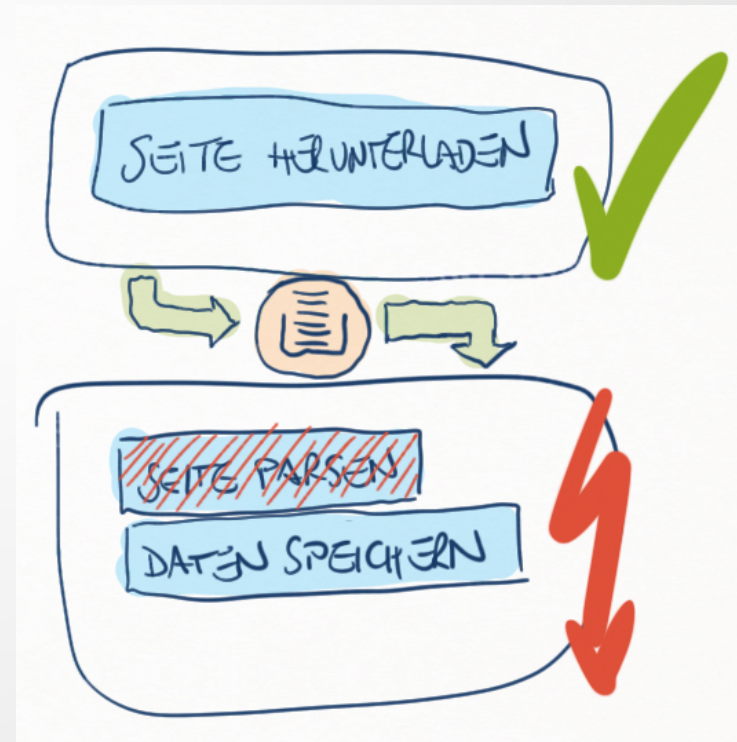
# Beispiel: Web-Scraper

- Entkoppelt man nun aber die beiden Teile durch die Kommunikation über eine Message-Queue...



# Beispiel: Web-Scraper

- ... so kann im Schadensfall der Downloader weiterlaufen.
- Die Applikation wird robuster.



# Skalierung

# Skalierung

- Skalierung beschreibt die Art wie eine Applikation wachsen kann, um den Anforderungen gerecht zu werden.
- Je unabhängiger die einzelnen Teile sind, umso unabhängiger von einander können sie auch skalieren.

# Skalierung

- Wenn ich draufkomme, dass das Bottleneck in meiner Applikation das Downloaden von Webseiten ist, dann kann ich, dank der Entkopplung, mehrere Download-Server in der Cloud starten, die alle in die gleiche Queue schreiben.

# Skalierung

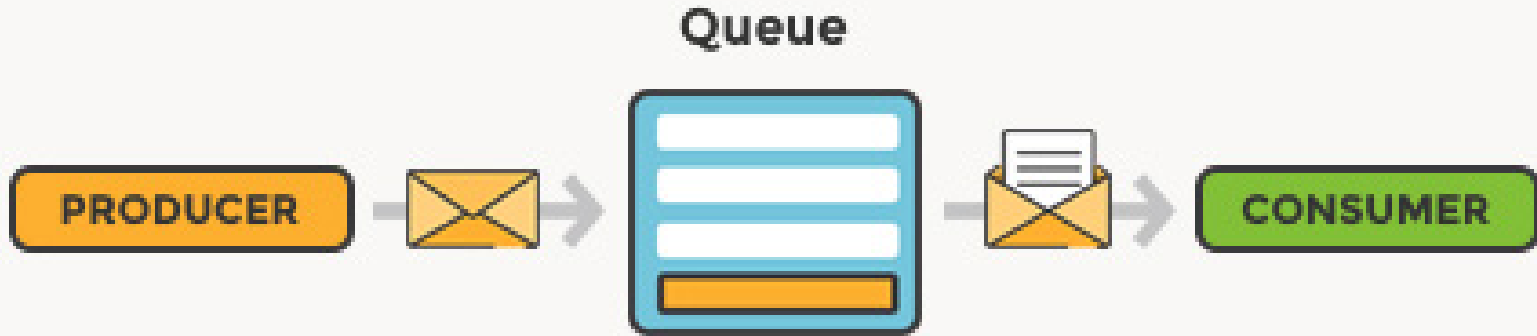
- Wenn ich aber bemerke, dass das Bottleneck in meiner Applikation das Parsen von Webseiten ist, dann kann ich, dank der Entkopplung, mehrere Compute-Server starten oder bauen und in Betrieb nehmen, die alle auf die gleiche Queue horchen.

Compute-Server haben vielleicht auch andere Hardware-Anforderungen als Download-Server.

So kann die Applikation optimaler skaliert werden.

# Producer - Consumer Pattern

# Producer - Consumer Pattern



- Ein oder mehrere Producer schicken Messages in eine Queue
- Ein oder mehrere Consumer entnehmen diese Messages zur Bearbeitung



# Producer - Consumer Pattern

- Wird genommen, wenn eine Nachricht genau einmal von einem einzigen Consumer verarbeitet werden soll.
- Mehrere Consumer 'streiten' sich um die Daten (-> competing consumer)

## **Garantien**

- one-time delivery
  - no more
  - no less

# RabbitMQ

(stellvertretend für viele  
Implementierungen... zum Beispiel  
JMQ)

# RabbitMQ

Eine gängige Message-Queue  
Implementierung

- Unterstützt verschiedene Protokolle
  - AMQP 0-9-1 and extensions
  - STOMP
  - MQTT
  - AMQP 1.0
  - HTTP und WebSockets  
(mit STOMP und MQTT)
  - RabbitMQ Streams

# RabbitMQ

- Ist leicht zu installieren (der Broker)
  - Kubernetes
  - BOSH
  - Chef
  - Docker
  - Puppet
- Distributed Deployment
  - Clusters
  - Federation  
(lose Kopplung; nur bestimmte Nachrichten werden an andere Server geschickt; kein volles Clustering)

# RabbitMQ

- Unterstützt verschiedenste Programmiersprachen
  - Java
  - .NET
  - PHP
  - Python
  - JavaScript
  - Ruby
  - Go
  - ...

# RabbitMQ

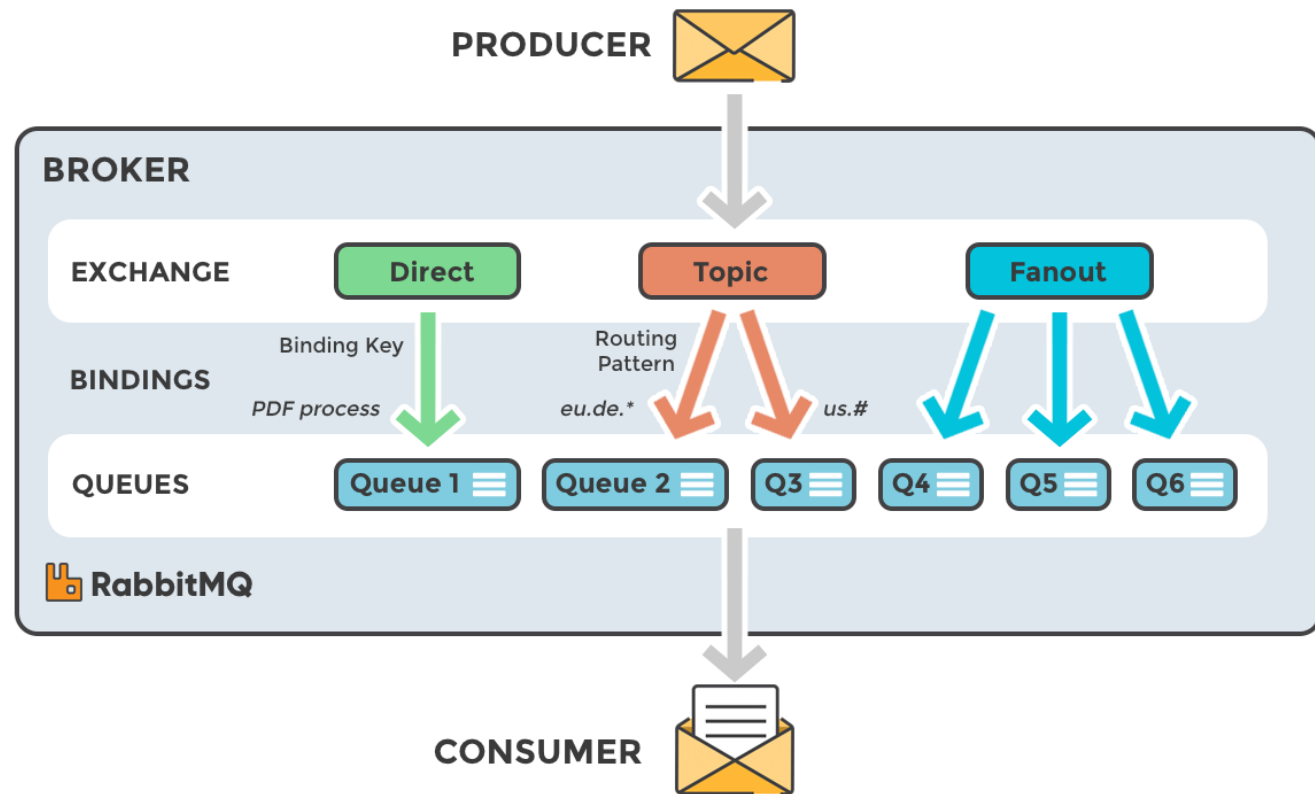
- Kann auch komplexes Message-Routing (durch Mitgabe von Zusatzdaten kann der Broker entscheiden in welche Queue die Nachricht geschickt wird)

# RabbitMQ

Exchange

# RabbitMQ - Exchange

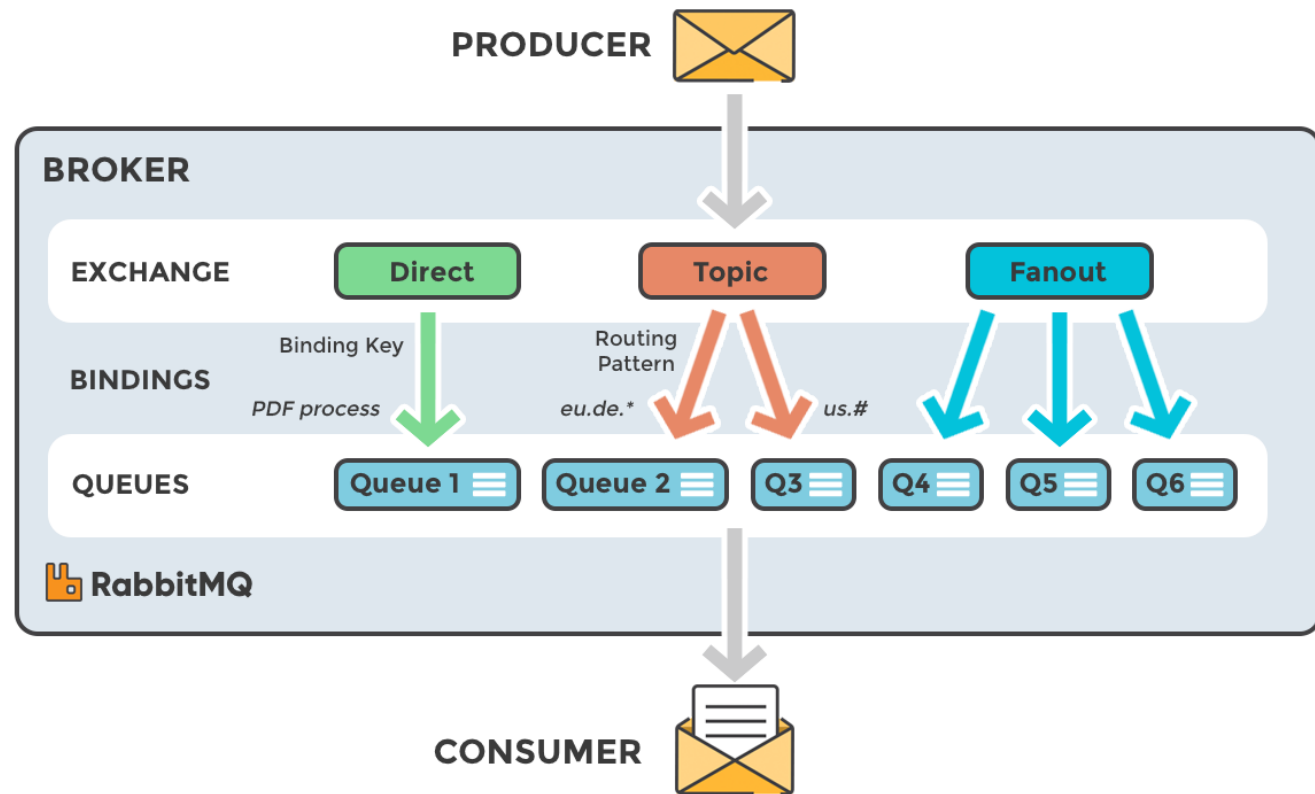
Ist eine Abstraktion, die es erlaubt mehrere verschiedene Queue-Paradigmen zu verwenden.





# RabbitMQ - Exchange

In der Exchange steht die Art der Verwendung...

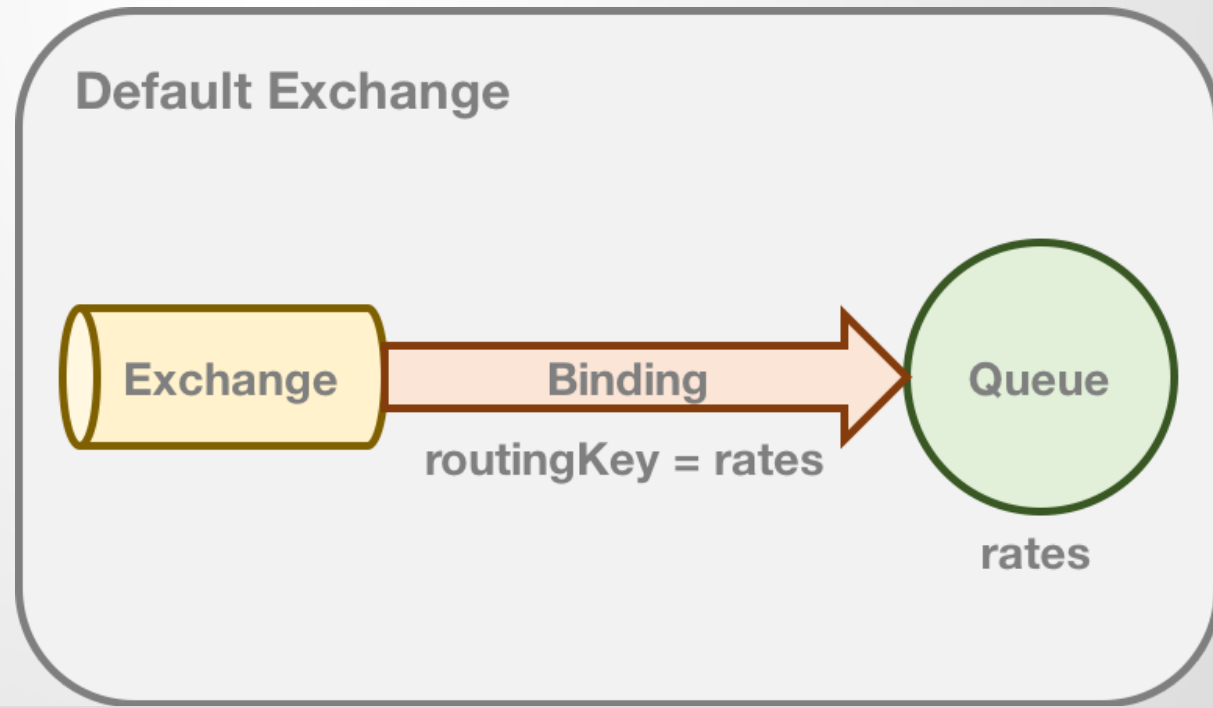


# RabbitMQ

Default Exchange

# Exchange - Default

- Routing wird zwar mit Keys gemacht, aber der Key heißt wie die Queue (1:1 Binding).
- Consumer horchen immer auf einer Queue.

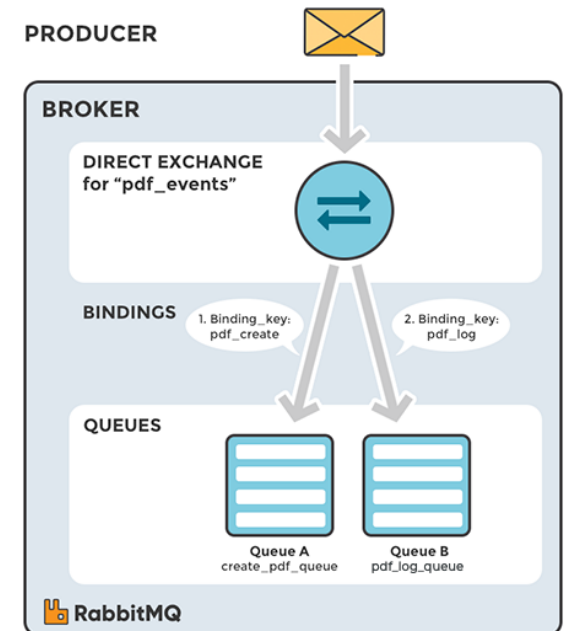


# RabbitMQ

Direct Exchange

# Exchange - Direct

- Routing kann mit Keys gemacht werden, die exakt getroffen werden müssen.
- Mehrere Queues möglich, **aber es kann immer nur eine getroffen werden.**
- Consumer horchen immer auf einer Queue.

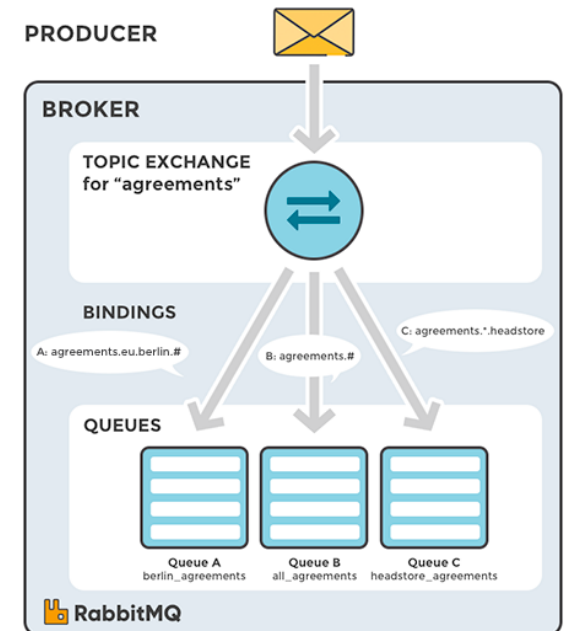


# RabbitMQ

Topic Exchange

# Exchange - Topic

- Routing kann mit Keys gemacht werden, die auch Wildcards enthalten können.
- Mehrere Queues möglich und **es können auch mehrere getroffen werden.**
- Consumer horchen immer auf einer Queue.



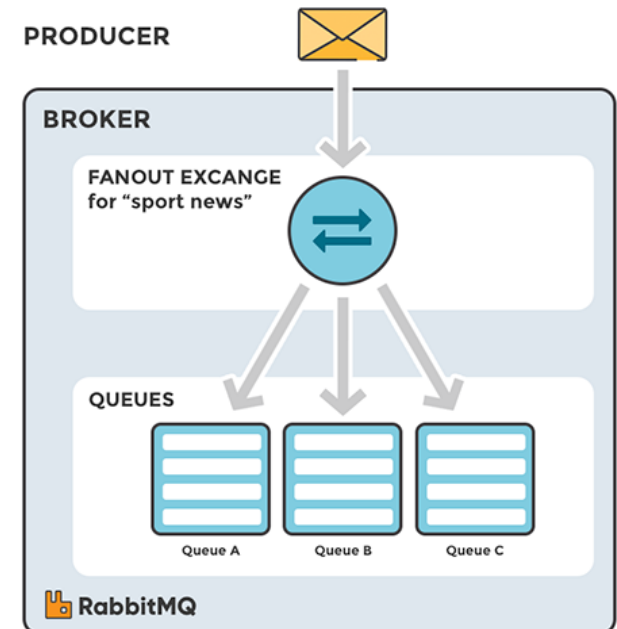
# RabbitMQ

Fanout Exchange



# Exchange - Fanout

- **Routing Keys werden komplett ignoriert.**
- Mehrere Queues möglich und **es wird immer an alle geschickt.**
- Consumer horchen immer auf einer Queue.

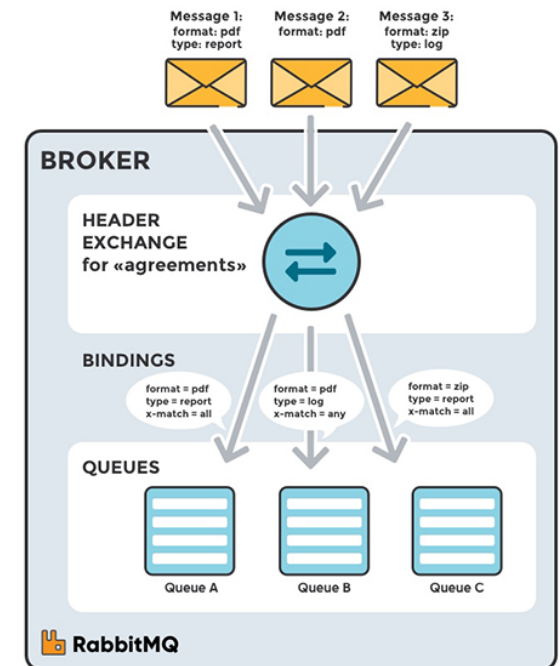


# RabbitMQ

Headers Exchange

# Exchange - Headers

- **Routing Keys werden komplett ignoriert.**
- Routing über Header Felder (komplexer, verschiedene Types...)
- Match-Any / Match-All / ...
- Mehrere Queues möglich.
- Consumer horchen immer auf einer Queue.

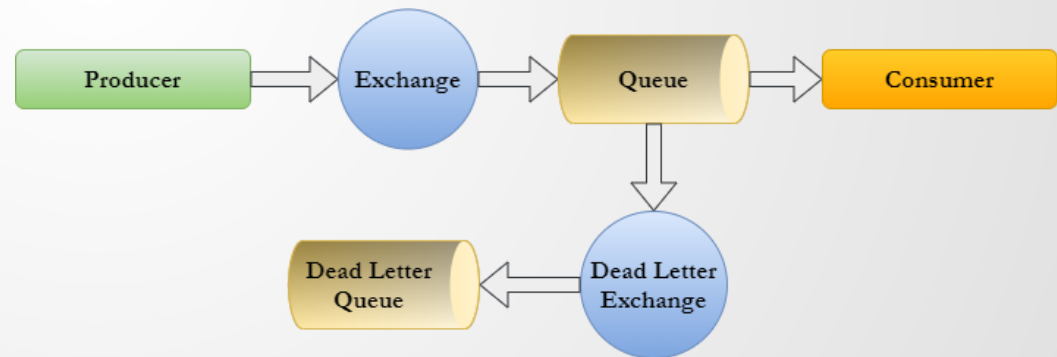


# RabbitMQ

Dead Letter Exchange

# Exchange - Dead Letter

- Wenn keine Queue getroffen wird oder die Nachricht nicht verarbeitet werden kann, dann wird die Nachricht an diese Exchange geschickt, falls sie implementiert wurde.



# Other Queue Implementations

JMS

ZeroMQ

# JMS

- **J**ava **M**essaging **S**ervice
- Java-Entwicklung für EE Server  
Um entkoppelbare RPCs zu bauen
- Unterstützt Point-to-Point (1:1 mit einem Consumer) und Topics
- Viele verschiedene Implementierungen
  - OpenMQ (Glassfish)
  - HornetQ (JBoss)
  - ActiveMQ
  - **J**ava **M**essage **Q**ueue (JMQ von Sun)
  - ...

# ZeroMQ

- Simple Queue-Implementierung.
- Wird oft als Socket-Middleware verwendet.
- Beliebige Nachrichten über Sockets an eine Queue schicken.
- Extrem schnell.
- Fully distributed.  
Kein Server (Broker).





# References

- <https://www.rabbitmq.com>
- <https://www.rabbitmq.com/#features>
- <https://www.cloudamqp.com/blog/what-is-message-queuing.html>
- <https://hevo.com/learn/rabbitmq-exchange-type/#:~:text=A%20headers%20RabbitMQ%20exchange%20type,routed%20based%20on%20header%20values.>
-