Hochschule **Rosenheim**
University of Applied Sciences

Verteilte Verarbeitung

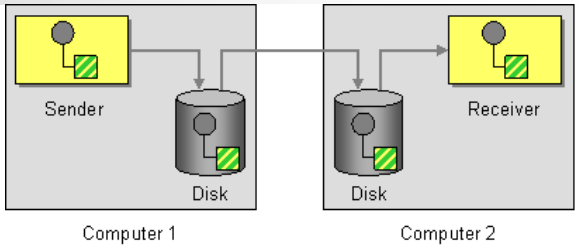
Kapitel 11

Grundlegende Eigenschaften Messaging

Agenda

- Sichere Übertragung mit Acknowledge
- Publish und Subscribe
- Routing
- Request Reply

Garantierte Übertragung



- = sicherstellen, dass Nachrichten ankommen
  - Wenn Sender abstürzt
  - Wenn das Netzwerk nicht da ist
  - Wenn der Empfänger gerade nicht läuft
- Verfahren = Store and Forward
  - Mehrere Zwischenspeicher des AMQP Providers (Hauptspeicher und/oder nicht flüchtiges Medium, z.B. DBMS)
  - Weiterleiten der Nachricht, wenn nächster Zwischenspeicher erreichbar
  - Löschen der Nachricht aus Zwischenspeicher erst bei Acknowledge

Garantierte Übertragung
Manuelles und Automatisches Acknowledge

- ***Automatisches Acknowledge*** erfolgt bei Empfang der Nachricht beim anmelden des Callbacks (deliveryCallback)

```
Channel channel = …
DeliverCallback deliverCallback = new DeliverCallback() {
    public void handle(String cT, Delivery msg) throws IOException { … }
};

boolean autoAcknowledge = true;
channel.basicConsume(<Queue>, autoAcknowledge, deliverCallback, …)
```

- ***Manuelles Acknowledge*** durch den Client …

```
boolean autoAcknowledge = false;
channel.basicConsume(<Queue>, autoAcknowledge, deliverCallback, …)
```

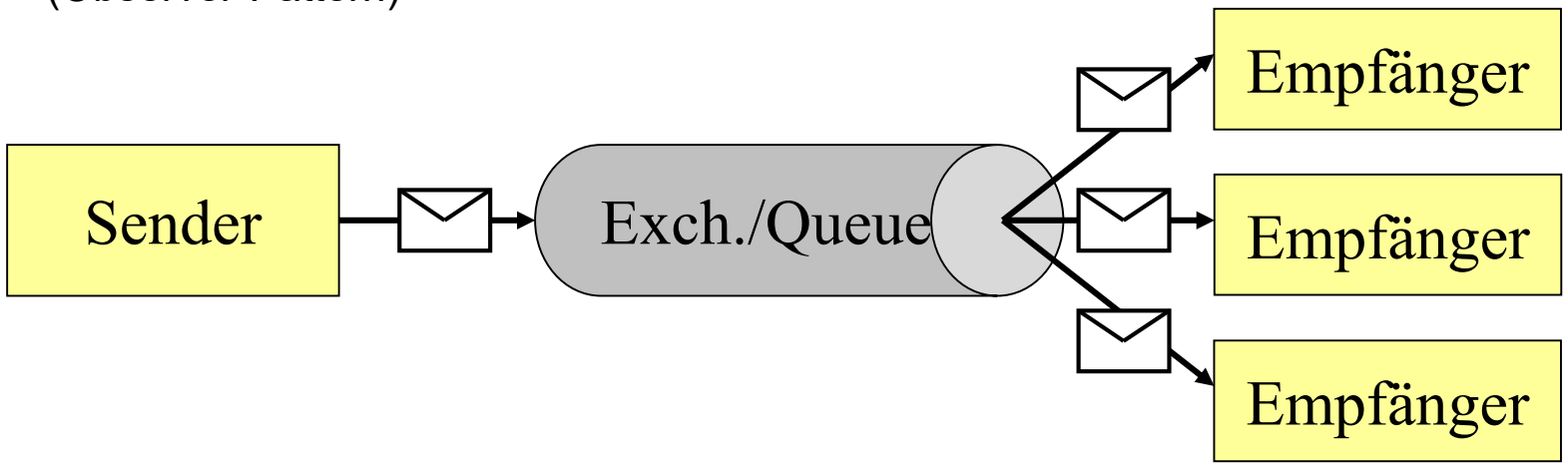
- Acknowledge dann über den Channel:

```
DeliverCallback deliverCallback = new DeliverCallback() {
  public void handle(String cT, Delivery msg) throws IOException {
    …
    channel.basicAck(message.getEnvelope().getDeliveryTag(), false);
  }
};
```

Producer/Consumer

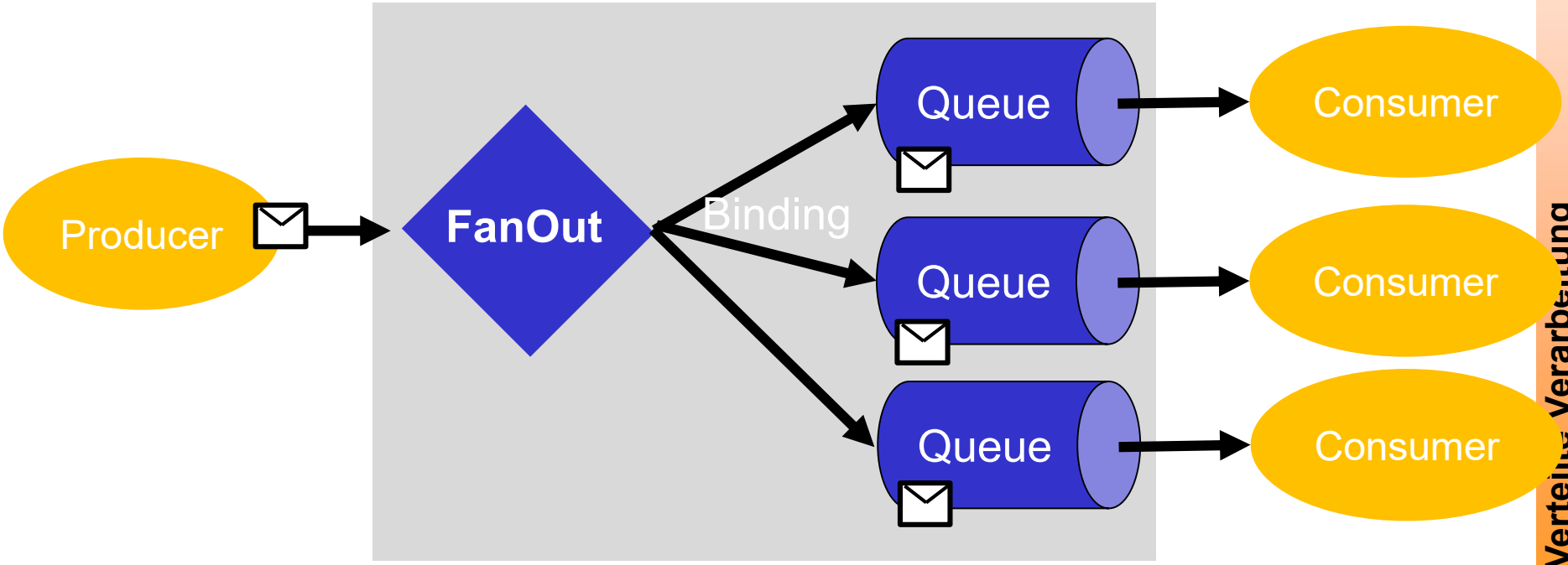
Messaging Konzepte
Publish/Subscribe

- Ein Sender, viele Empfänger:
  - Sender schicken Nachricht an Fanout-Exchange (AMQP) / Topic (JMS)
  - Viele Empfänger melden sich bei entspr. Queues (AMQP) bzw. dem Topic (JMS) an
  - Alle Empfänger erhalten Nachricht (***nicht verbrauchend***)
  - Empfänger müssen jedoch online sein
- Sender und Empfänger entkoppelt
  - Sender kennt Empfänger nicht (Observer-Pattern)



Umsetzung: Publisher Subscriber

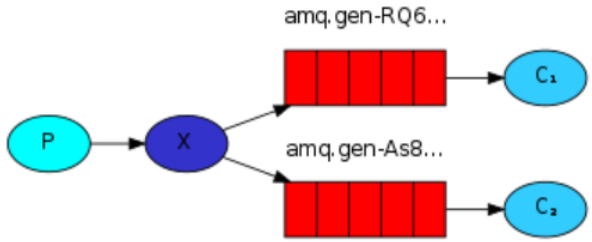
- FanOut liefert aus an alle Queues, die über das Binding angemeldet sind
- = asynchrones Client / Server
- Unabhängig vom Binding Key



AMQP über „fanout“ – Exchange

- Am Publisher

```
Channel channel =
channel.exchangeDeclare(“log-exchange”, "fanout");

String message = "Nachricht";
channel.basicPublish((“log-exchange”, "",
      null, message.getBytes("UTF-8"));
```



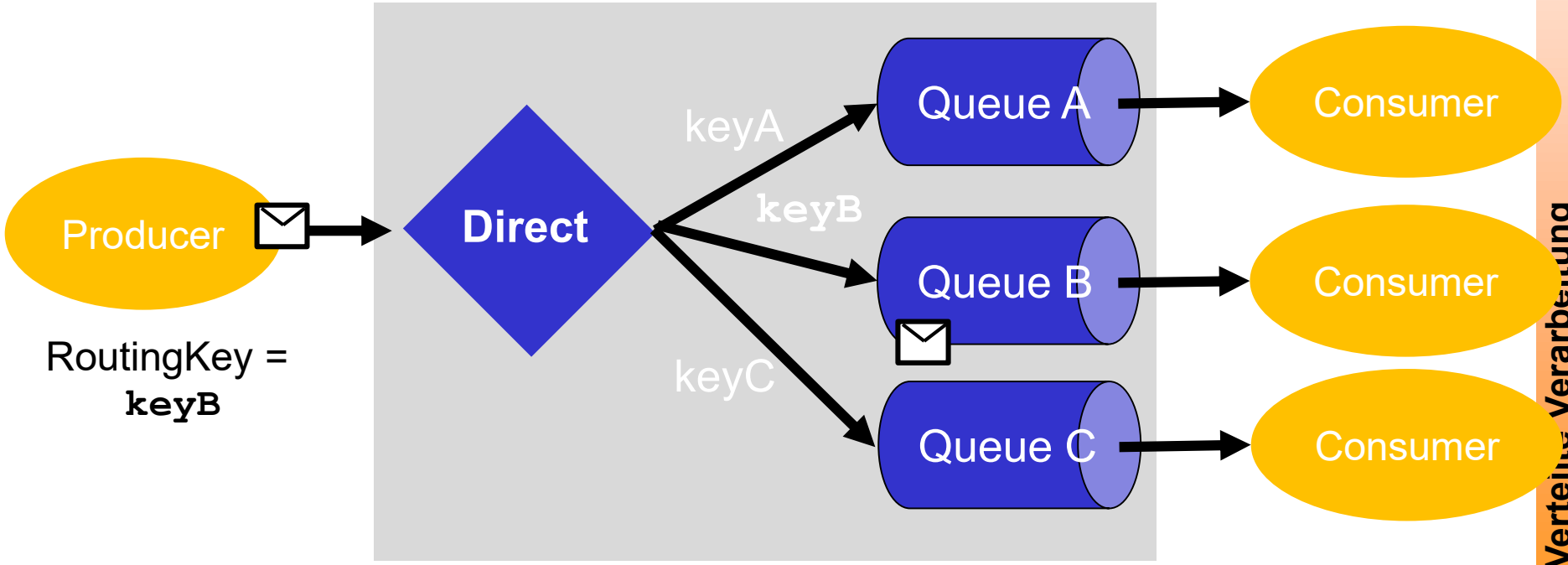
- Am Subscriber

```
Channel channel = connection.createChannel();
channel.exchangeDeclare(“log-exchange”, "fanout");
String qName = channel.queueDeclare().getQueue();
channel.queueBind(qName, (“log-exchange”, "");

DeliverCallback deliverCallback = (cT, delivery) -> {
  String message = new String(delivery.getBody(), "UTF-8");
  System.out.println(" [x] Received '" + message + "'");
};
channel.basicConsume(qName, true, deliverCallback, cT -> { });
}
```

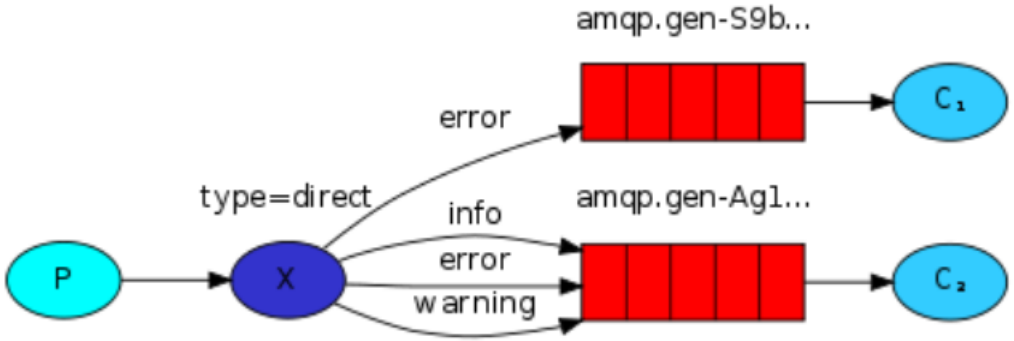
Routing

Routing über Routing Keys

- Direct: Routing Key = Binding Key
- Damit direkt Queue und Consumer adressieren



Java Beispiel



- Am Producer

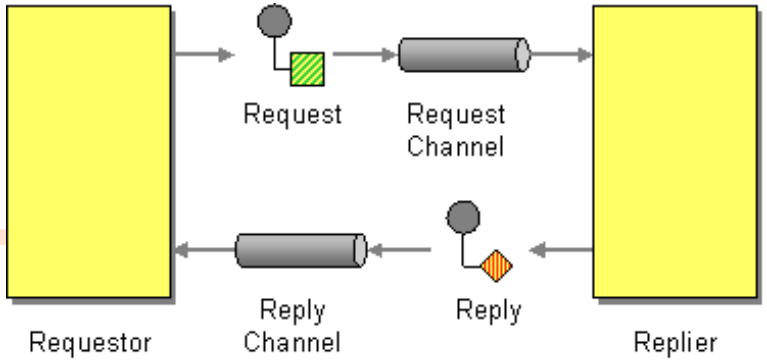
```
channel.exchangeDeclare(EXCHANGE_NAME, "direct");

String message = "Nachricht";
String severity = "error";
channel.basicPublish(EXCHANGE_NAME, severity,
     null, message.getBytes("UTF-8"));
```

- Am Consumer

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

String queueName = channel.queueDeclare().getQueue();
channel.queueBind(queueName, EXCHANGE_NAME, "error");
channel.queueBind(queueName, EXCHANGE_NAME, "warning");
channel.queueBind(queueName, EXCHANGE_NAME, "info");
```

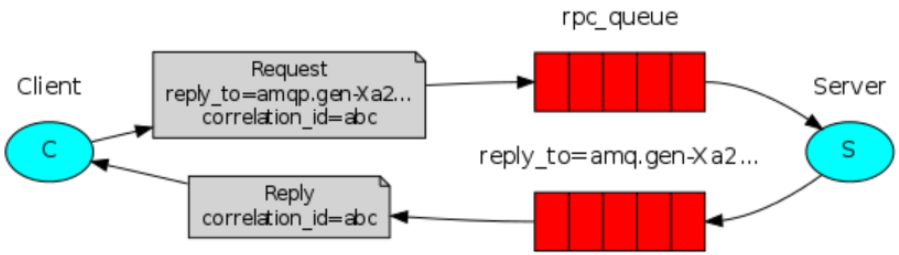
Request and Reply

Request and Reply



- Messaging typischerweise Kommunikation nur in eine Richtung
- Request/Reply (wie bei RMI/RPC) kann synchron nachgebaut werden
  - Requestor blockiert nach dem Senden der Request-Nachricht
  - Requestor arbeitet weiter nach Empfang der Reply-Nachricht
  - Problem: Was passiert, wenn Replier nicht online / Absturz?
  - Sonderform (Asynchronous-Callback): Requestor spaltet eigenen Thread ab, der auf die Reply-Nachricht wartet
- Anwendungen
  - Messaging RPC (CommandMessage -> ResultMessage)
  - Messaging Query (QueryCommandMessage -> ResultMessage)
  - Notify/Acknowledge (Empfangsbestätigung)

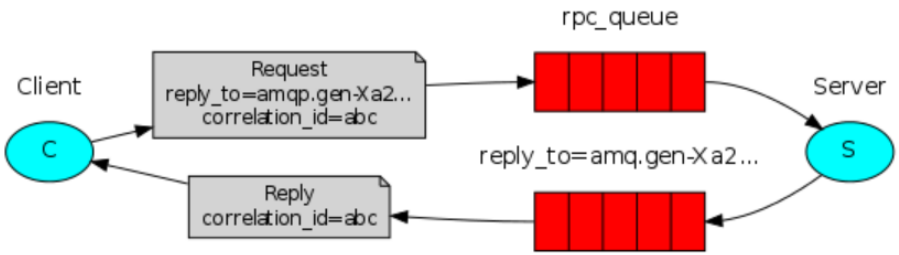
Synchrones Request / Reply



- Am Client

```
final String corrId = UUID.randomUUID().toString();

String replyQueueName = channel.queueDeclare().getQueue();
AMQP.BasicProperties props = new AMQP.BasicProperties
    .Builder()
    .correlationId(corrId)
    .replyTo(replyQueueName)
    .build();

channel.basicPublish("", requestQueueName, props,
message.getBytes("UTF-8"));

// …

String ctag = channel.basicConsume(replyQueueName, true,
    (consumerTag, delivery) -> {
       if  (delivery.getProperties()
                    .getCorrelationId().equals(corrId)) {
          // ...
       }
    }, consumerTag -> {});
```

Synchrones Request / Reply



- Am Server

```
DeliverCallback deliverCallback = (consumerTag, delivery) -> {
  AMQP.BasicProperties replyProps = new AMQP.BasicProperties
      .Builder()
      .correlationId(delivery.getProperties().getCorrelationId())
      .build();
  // … Berechnung
  channel.basicPublish("", delivery.getProperties().getReplyTo(),
      replyProps, response.getBytes("UTF-8"));
  channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
}


channel.basicConsume(RPC_QUEUE_NAME, false,
      deliverCallback, (consumerTag -> {}));
```