

Enterprise Programming 2

Lesson 10:

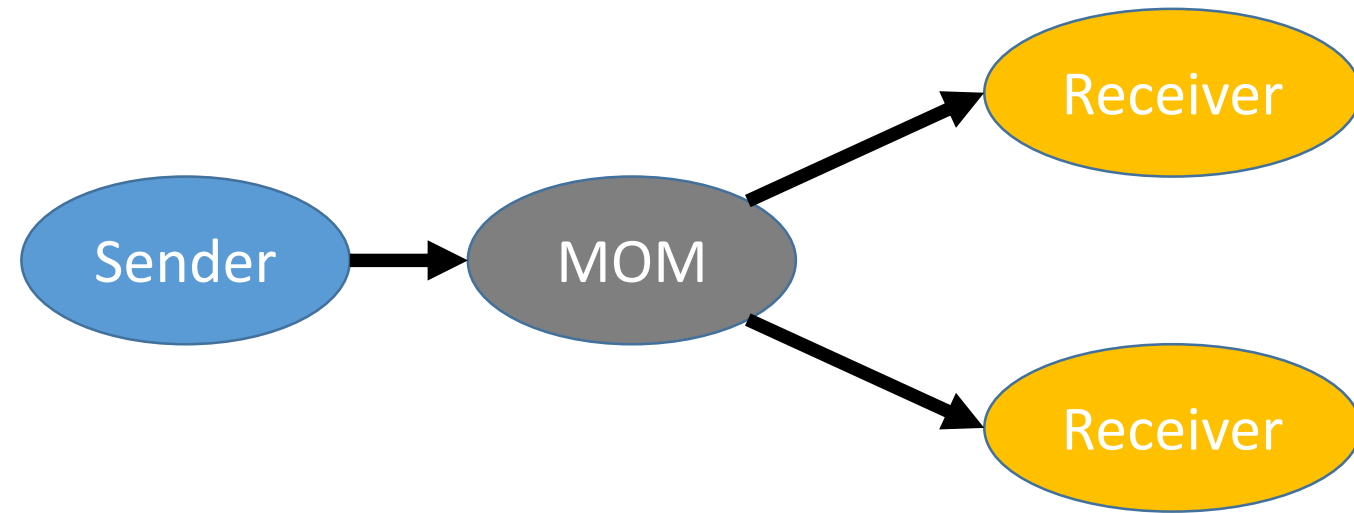
AMQP and RabbitMQ

Prof. Andrea Arcuri

Goals

- Understand the need for a Message Oriented Middleware (MOM) in MicroServices
- Learn different topologies of MOM communications
- Learn how to use RabbitMQ from Spring

MOMs

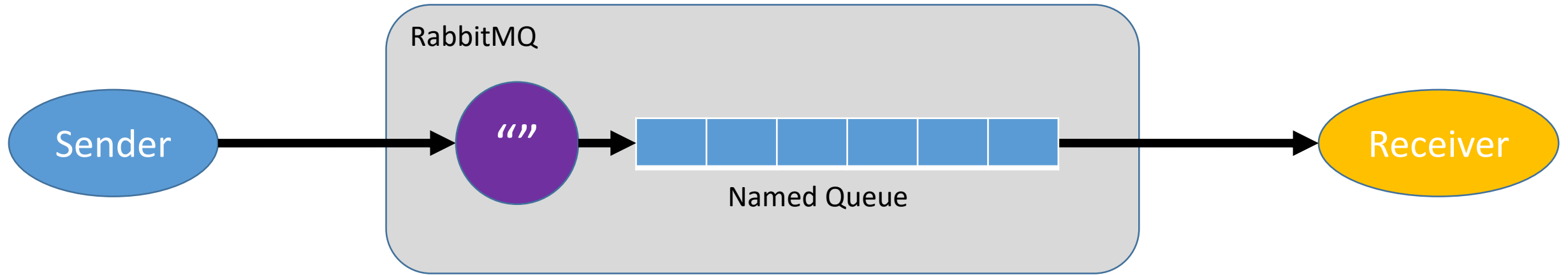


- Decoupling *sender* from *receiver(s)*: the sender does not know who the receivers are
- *Sender* will *publish* messages to a *queue*, and *receivers* will *subscribe* to such messages
- Why?
 - **Maintainability**: can add/remove senders/receivers in the future with little to no change in the architecture
 - Services can react to events asynchronously

AMQP/RabbitMQ

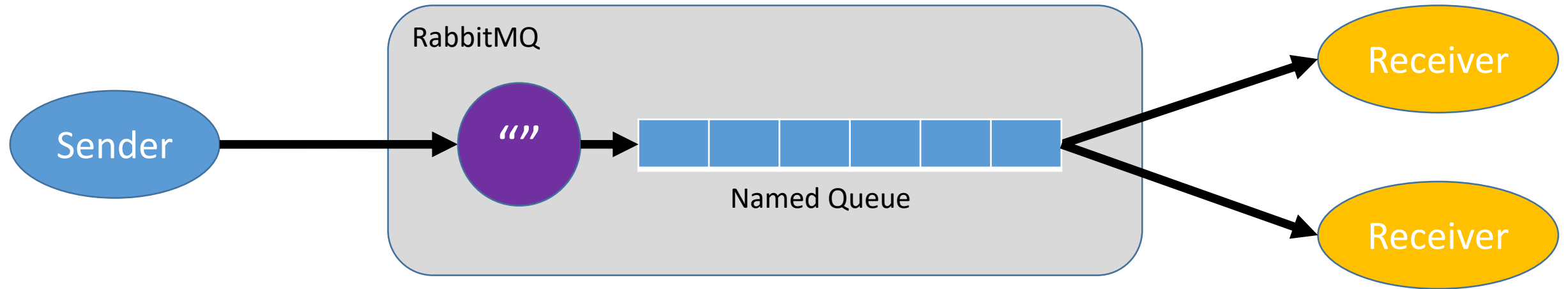
- Advanced Message Queuing Protocol (*AMQP*)
- Defines protocol of how messages should be formatted and sent
- *RabbitMQ* is a MOM using *AMQP*
- Written in Erlang
 - we will start it via Docker

Queue



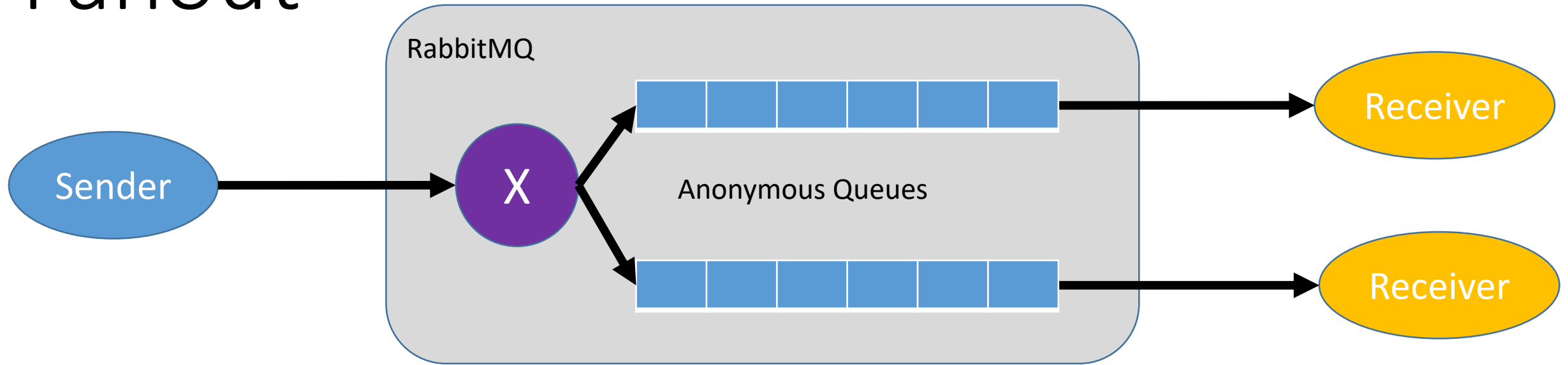
- RabbitMQ keeps queues of messages
- Each queue has a name
- Sender sends to a named exchange in RabbitMQ
- Default exchange "" (empty name) copies messages to the specified queue by name
- Receiver pulls from such queue

Distributed Work



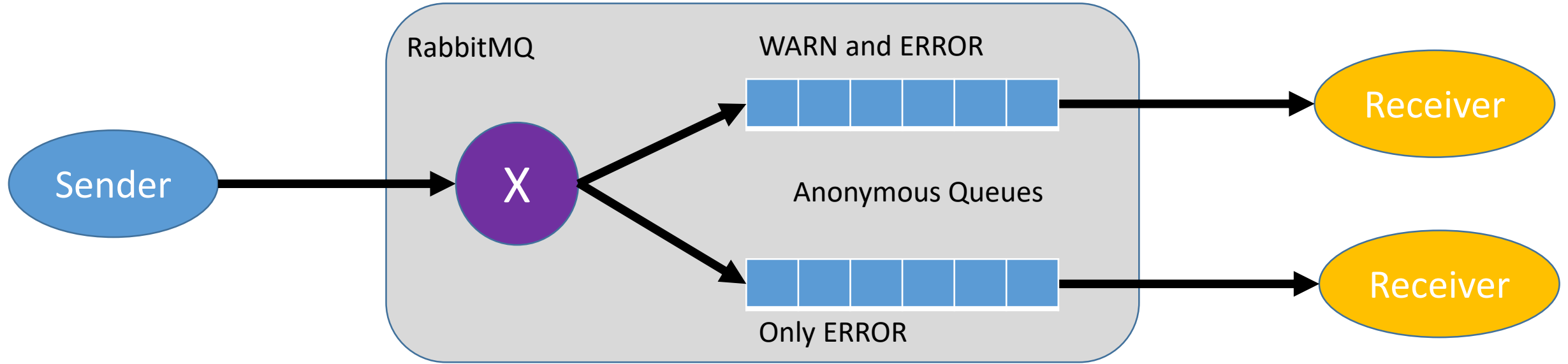
- There can be several Receivers pulling from a queue
- Useful when a message requires non-trivial processing
 - If a Receiver is stuck with a long task, the other Receivers can meanwhile work on all the other messages pushed on the queue
- A message can only go to 1 Receiver
- *Prefetch*: for optimization reasons, Receiver could pull several messages at same time, instead of waiting to process current one before pulling the next
 - important if time delay from RabbitMQ is significant compared to cost of processing 1 message

Fanout



- Broadcast of a message to several Receivers
- Sender needs to specify name of the exchange (eg, *X*)
- Each receiver creates its own anonymous queue
 - Queue with a random unique name, which we don't care for
- Exchange *X* copies incoming messages on all queues

Direct Exchange



- Broadcast like Fanout, but each message has a *Key*
 - eg, INFO, WARN and ERROR when dealing with log messages
- When Receiver creates a queue, it specifies the Keys it wants to be notified to
- In above example:
 - INFO messages copied to no queue
 - WARN messages copied to only 1 queue
 - ERROR messages copied to both queues

Topic Exchange

- Broadcast like Direct Exchange
- But finer grained way to specify routing to queues
- *Topic*: list of words separated by “.”
- Receiver specifies the topic it wants to pull for
- Special symbols: “*” substitutes 1 word, “#” substitute 0 or more words
- Ex, consider topic “*author.country.kind*” for news
 - “**.norway.**”: any news from Norway, regardless of author or kind
 - “*smith.#*”: any news from author Smith

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **advanced/amqp/base-queue**
- **advanced/amqp/distributed-work**
- **advanced/amqp/fanout**
- **advanced/amqp/direct-exchange**
- **advanced/amqp/topic-exchange**
- **advanced/amqp/amqp-rest**