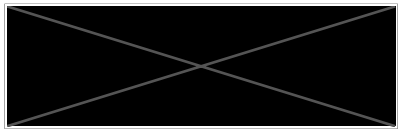
Programming 6

Messaging

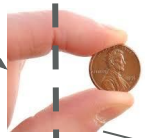


Putting it all together

Put_In

Appoint.

Take_Out



ReceivedPiggy

ActivityEvent

Synchronous communication

- Mostly sync:
  - REST calls
  - Database calls
  - Method calls
  - Commands via Facade within monolith
    - See Hexagonal Architecture
  - Events via Spring Application events within monolith
    - See Hexagonal Architecture

Advantages:

- Easy to manage
- Transactional boundaries very clear
- Acknowledgement assured
- Calls in order

Disadvantages:

- Tight coupling
- Blocking
- Error handling

Asynchronous communication

- Mostly async: via Message Oriented Middlewares
  - Event Bus
  - Queues
  - Streams
  - Examples: Akka, Kafka, RabbitMQ, ActiveMQ

Advantages:

- Loose coupling
- Non-blocking
- Simpler to scale
- Durable
- Error handling and resilience.

Disadvantages:

- Complexity
- Transactional boundaries
- Order and idempotency

Challenges

- Ordering
  - Do we get the messages in the right order?
  - What if we get an activity event for a Piggy Bank that we don't know exists yet?
  - What if we get a delete event before a create event?
- Duplicate messages
  - What if we get duplicate activities?
- Error handling
  - What if we cannot interpret a message, or cannot handle it?
- Distributed transactions
  - What if we have a business transaction spanning multiple contexts failing halfway through?
- Durability, retention, clustering, security… and many more…

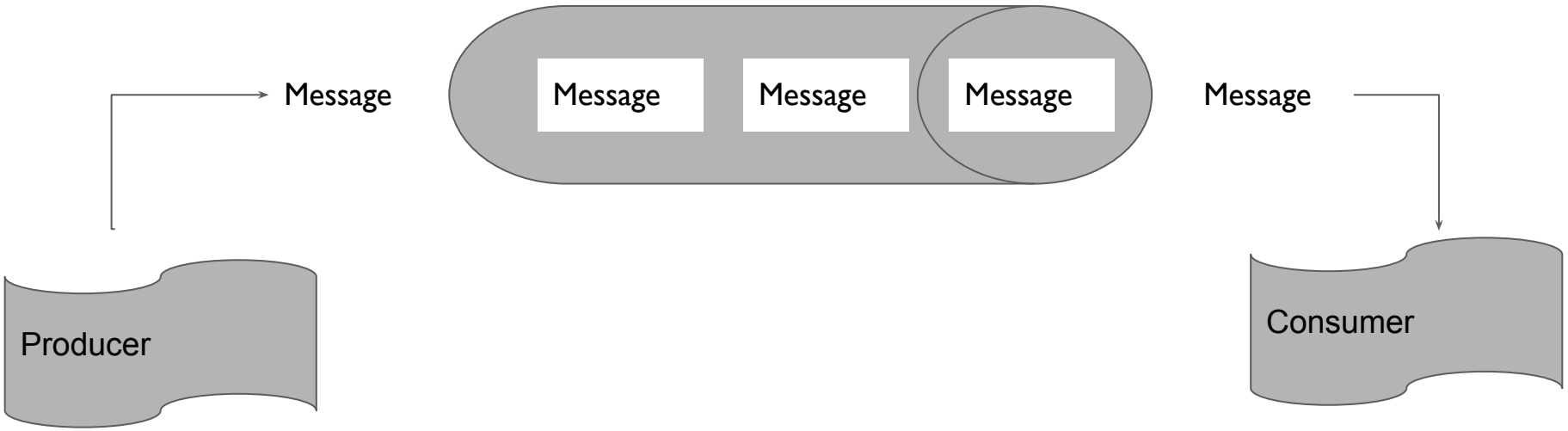
Messaging

There are a lot of different solutions out there.

Choose wisely!

The main idea is the same a producer puts something on a message broker and a consumer (or multiple) consumers reads it from the message broker.

If a consumer is down: the message is not lost, can be done async, easier to scale (just add another consumer), can be clustered and replicated, etc…



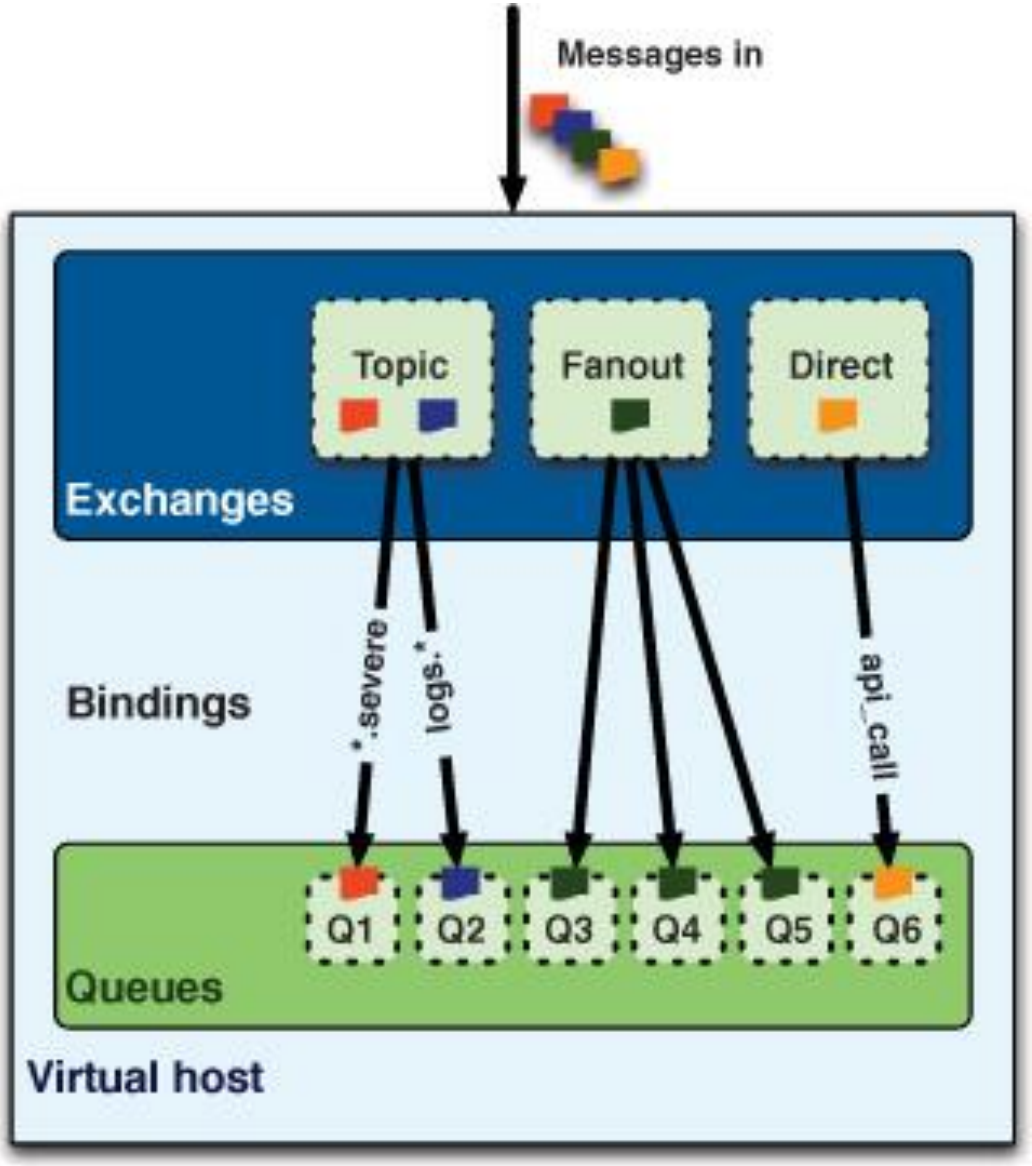
RabbitMQ vs Kafka

Kafka vs RabbitMQ	RabbitMQ	Kafka
Performance	4K-10K messages per second	1 million messages per second
Message Retention	Acknowledgment based	Policy-based (e.g., 30 days)
Data Type	Transactional	Operational
Consumer Mode	Smart broker/dumb consumer	Dumb broker/smart consumer
Topology	Exchange type: Direct, Fan out, Topic, Header-based	Publish/subscribe based
Payload Size	No constraints	Default 1MB limit
Usage Cases	Simple use cases	Massive data/high throughput cases

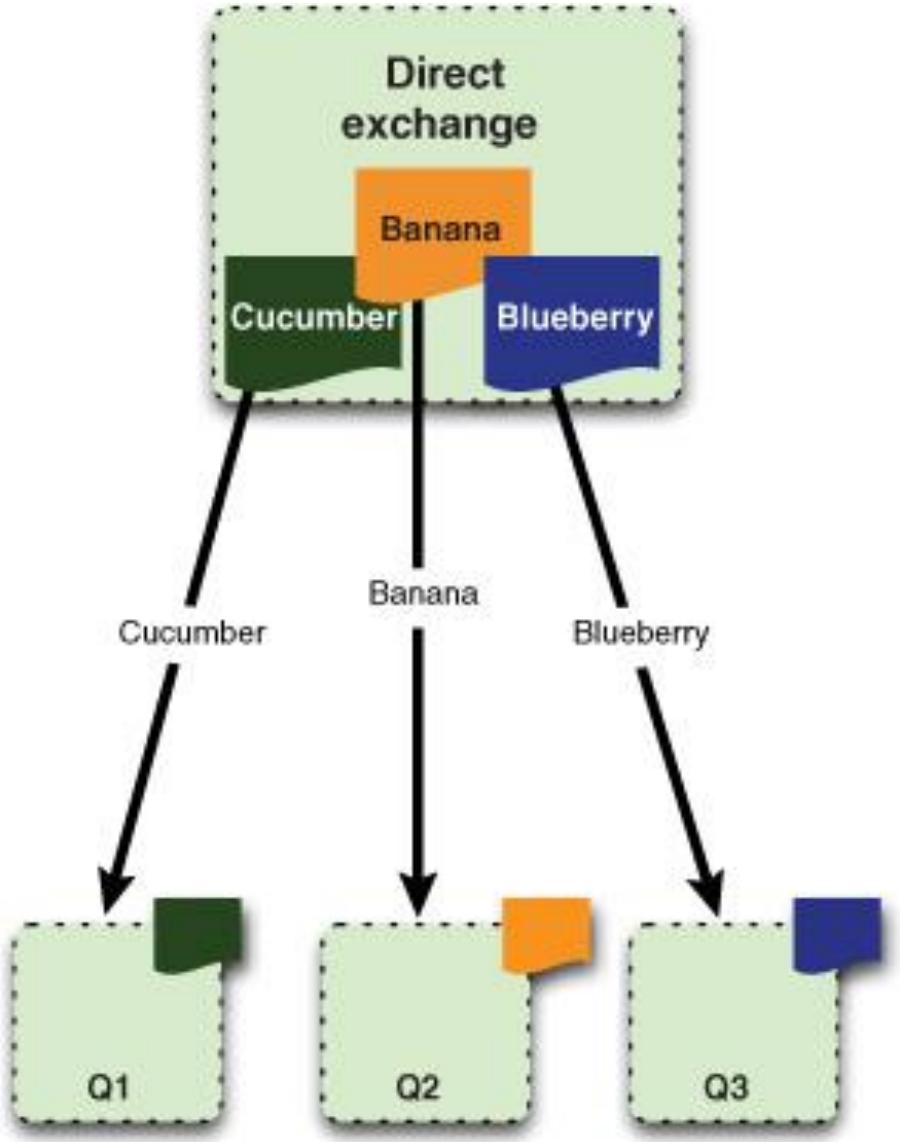
RabbitMQ

- Queue:
  - Smallest building block, keeps the messages that have been received and holds configuration data. Consumers listen to queues.
- Exchange:
  - Is a virtual building block, receives messages from producers and determines which queues they should be forwarded to.
    An exchange can be seen as a set of routing rules.
- Producer:
  - Sends messages to the broker, to an exchange. A message consists of a payload and some labels. The labels is metadata for the exchange to use to evaluate its routing rules.
- Consumer:
  - Subscribes to a queue and will acknowledge a message when read successfully
- Broker:
  - Applications talk to the broker. RabbitMQ has a fixed connection to the broker using an AMQP protocol.
- Binding:
  - You can bind a queue to an exchange, the binding will explain the routing rules to the exchange.

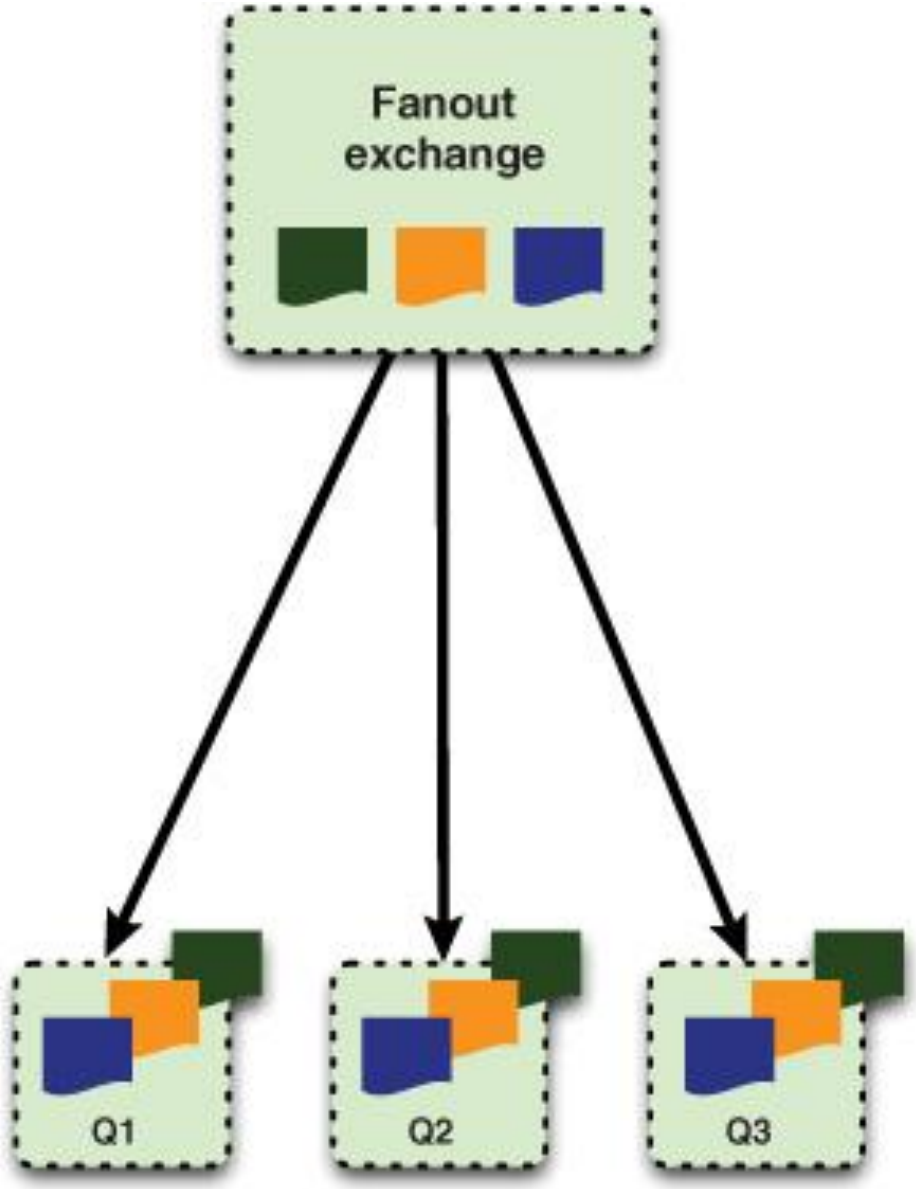
Message Paradigms in RabbitMQ



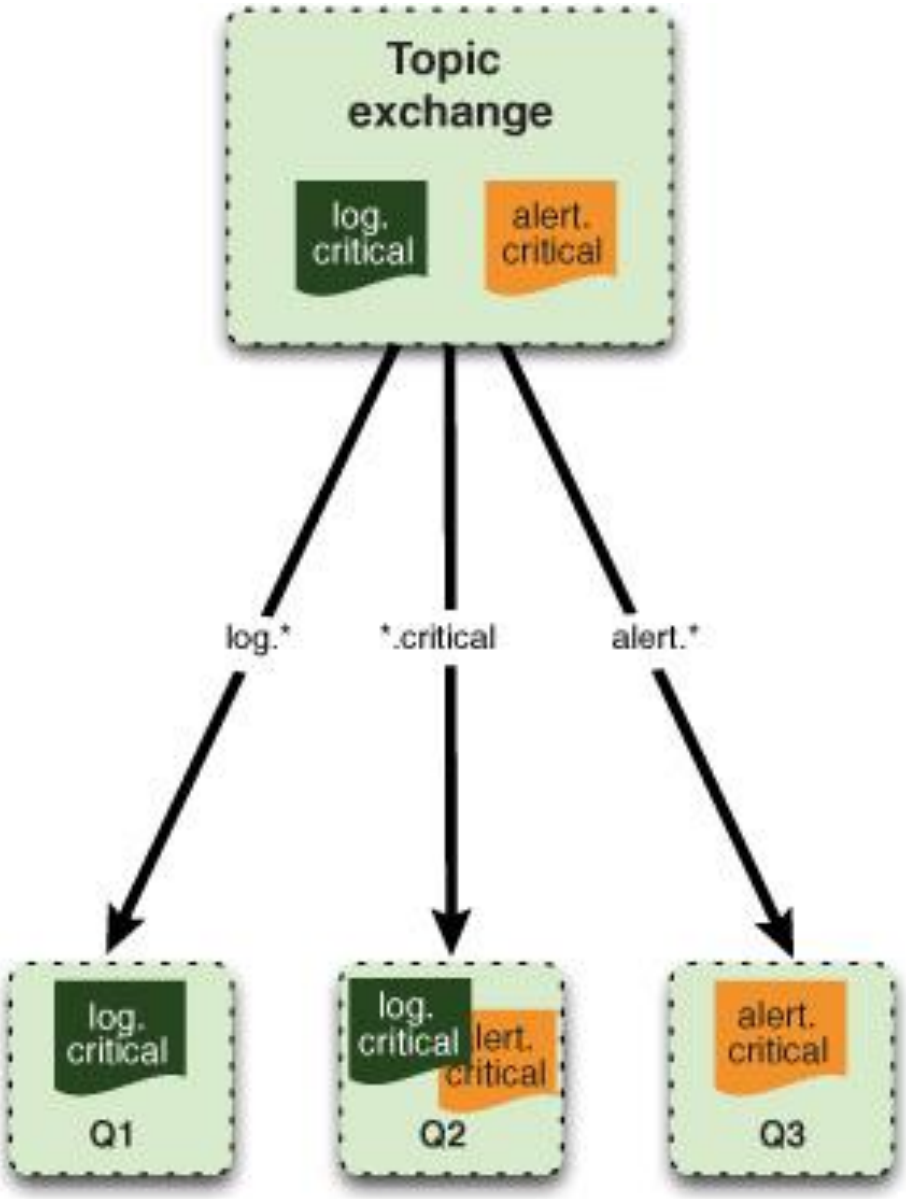
Direct exchange



Fanout exchange



Topic exchange



Distributed systems

- A RabbitMQ can be clustered for throughput and resilience.

  BUT:

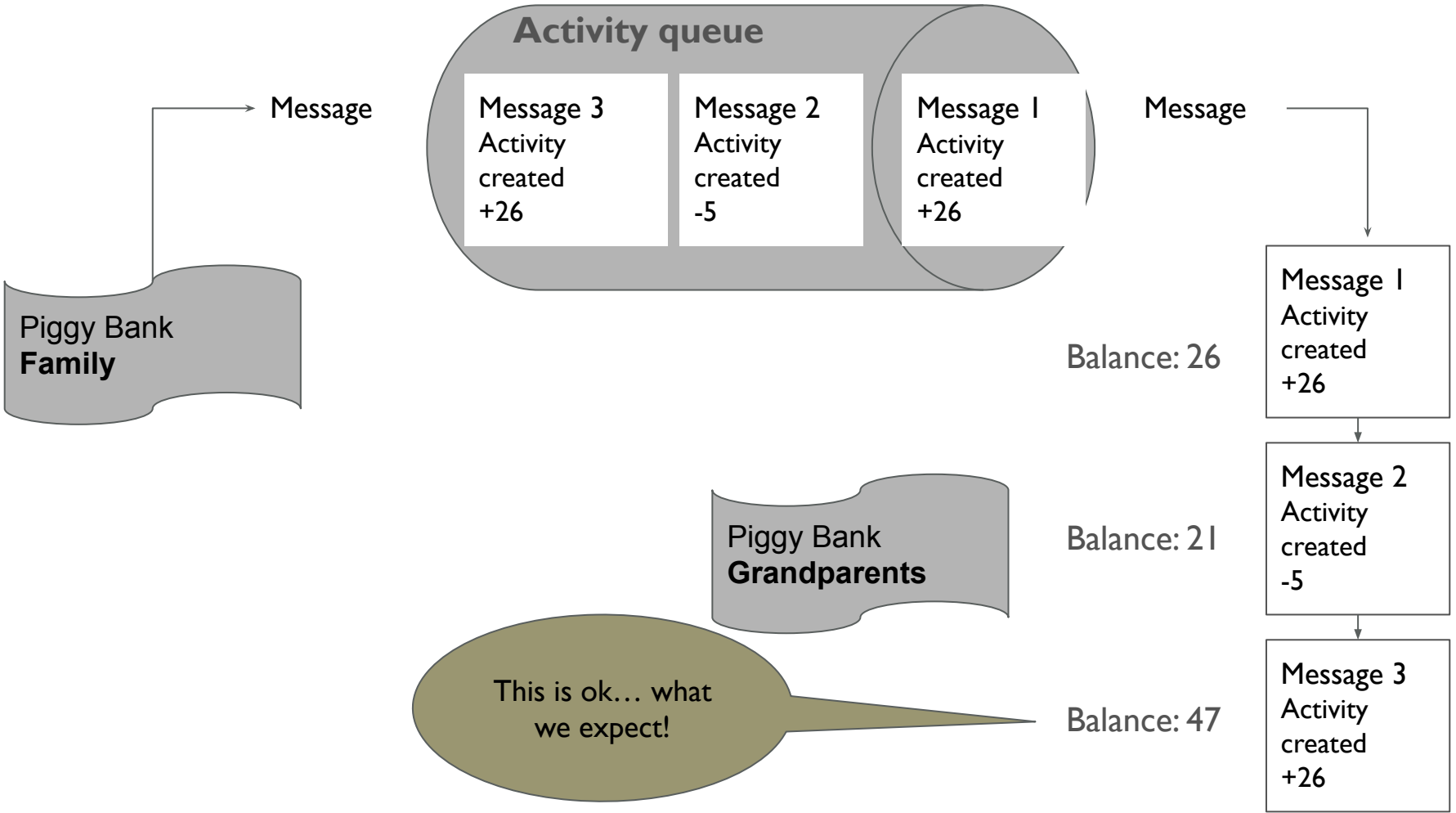
- At most-once delivery:
  - In case of failure in the message delivery, no retry is done.
  - Message loss can be an option.
  - No duplication
- At least-once delivery:
  - In case of failure in the message delivery, retries are done until acknowledgement
  - No message loss
  - Possible duplication of message
- Exactly-once delivery:
  - Message are ensured to be delivered exactly once
  - Most desirable

Things to keep in mind

- There is no such thing as **exactly-once delivery**

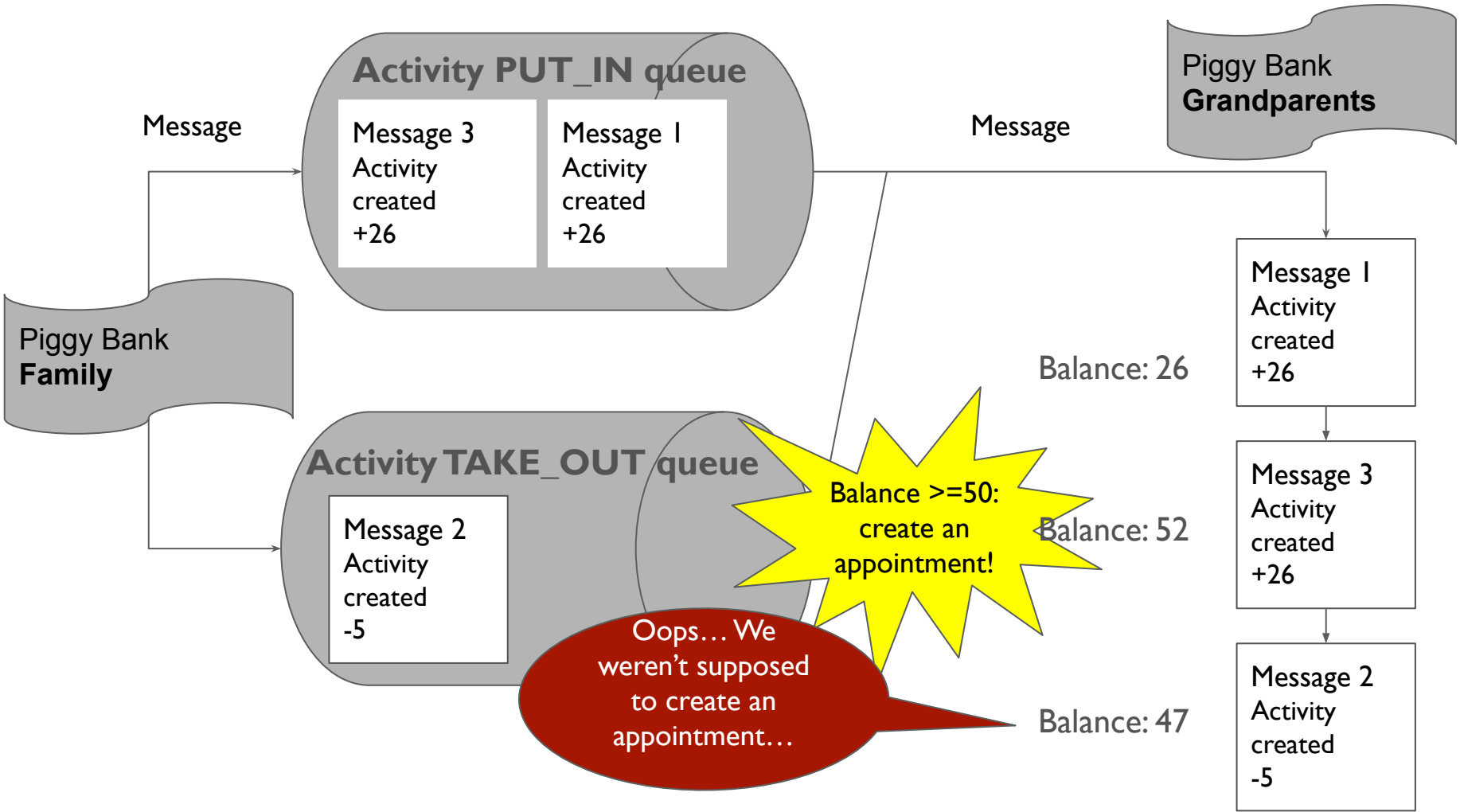
Message ordering

- Listeners to a queue process messages in FIFO (First In First Out) order…
- So when does ordering become an issue?



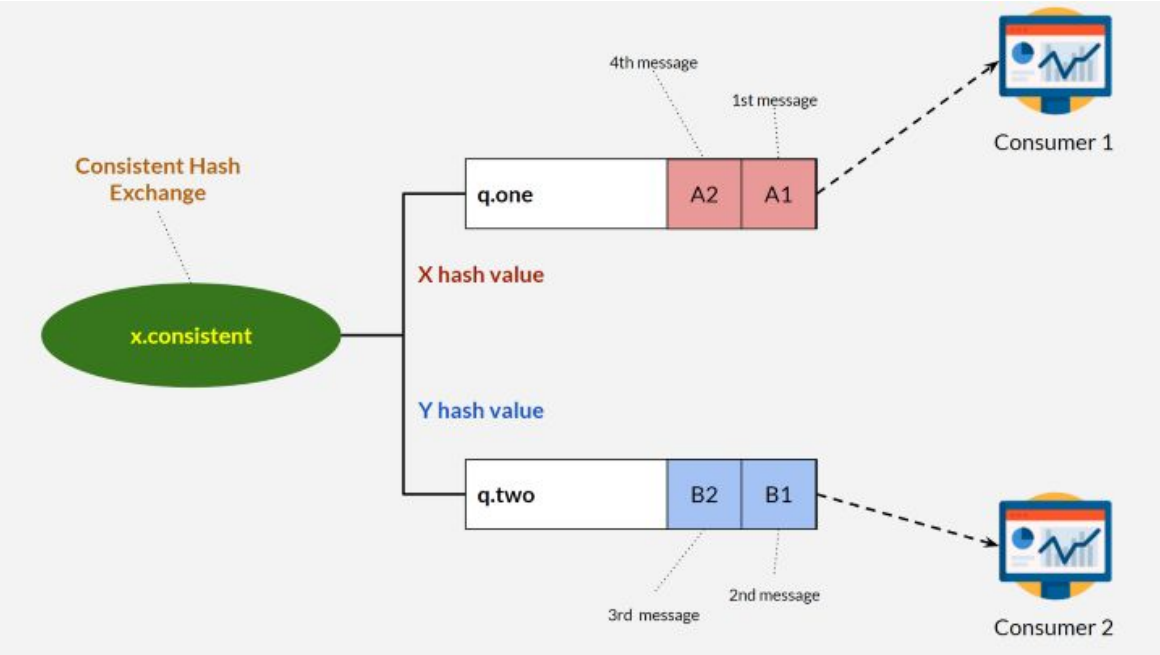
Message ordering

- **Ordering becomes an issue when we split the same message type across 2 or more queues!**
- Ordering is only guaranteed within a single queue



Message ordering

- The solution?
- Kafka: out of the box, using partition keys
- RabbitMQ: this might become a problem
  - But solution with new exchange: Consistent Hash Exchange (plugin)
- Use identifier as routing key, RabbitMQ will hash identifier and put it on the same queue.
  - PiggyBankUUID's as identifier for instance.



Informational

Questions?