AWS SQS and SNS

CS516 – Cloud Computing

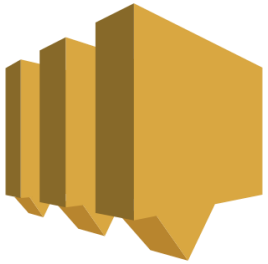
Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- SNS
  - App2App and App2Person
  - Topic, Subscriber, Publisher
  - Message filter
- SQS
  - Standard and FIFO queues
  - Deduplication id & group id
  - Visibility timeout
  - Long polling & short polling
  - Dead letter queue
- Application decoupling



Simple Notification Service - SNS

SNS is a fast, flexible, fully-managed **push** notification service that sends messages to large numbers of recipients based on **events** that happen in your AWS account.

A notification service, generally used in conjunction with CloudWatch to email or text alerts when some event occurs in an account.

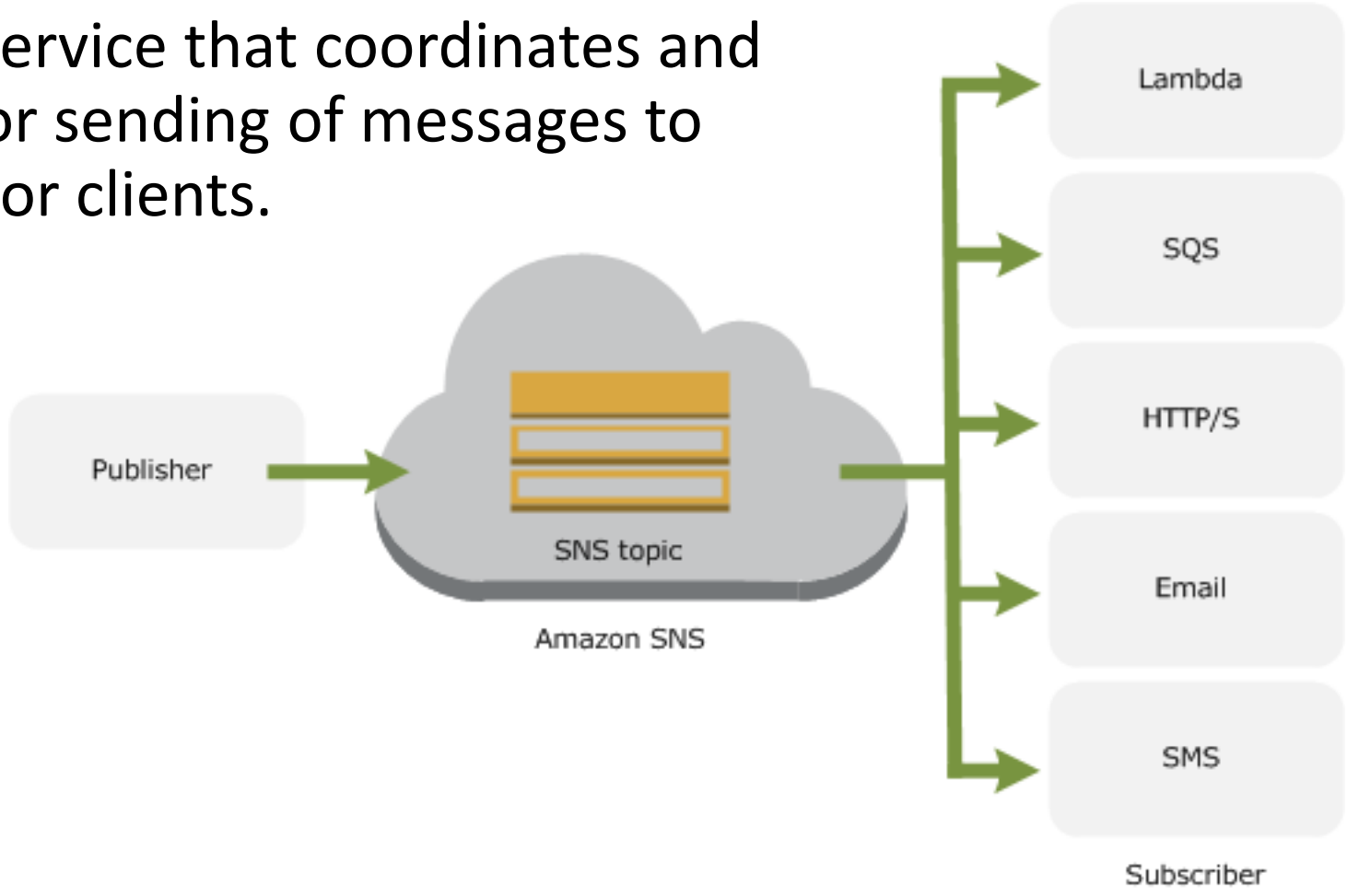
SNS Basic Components

1. **Publishers** communicate **asynchronously** with subscribers by producing and sending a message to a topic.
2. **Topic** is a logical access point and communication channel.
3. **Subscribers** consume or receive the message or notification over one of the supported protocols when they are subscribed to the topic.

There are two types of clients, publishers and subscribers.

What is Amazon SNS?

Amazon SNS is a web service that coordinates and manages the delivery or sending of messages to subscribing **endpoints** or clients.



Read more about [Amazon SNS]

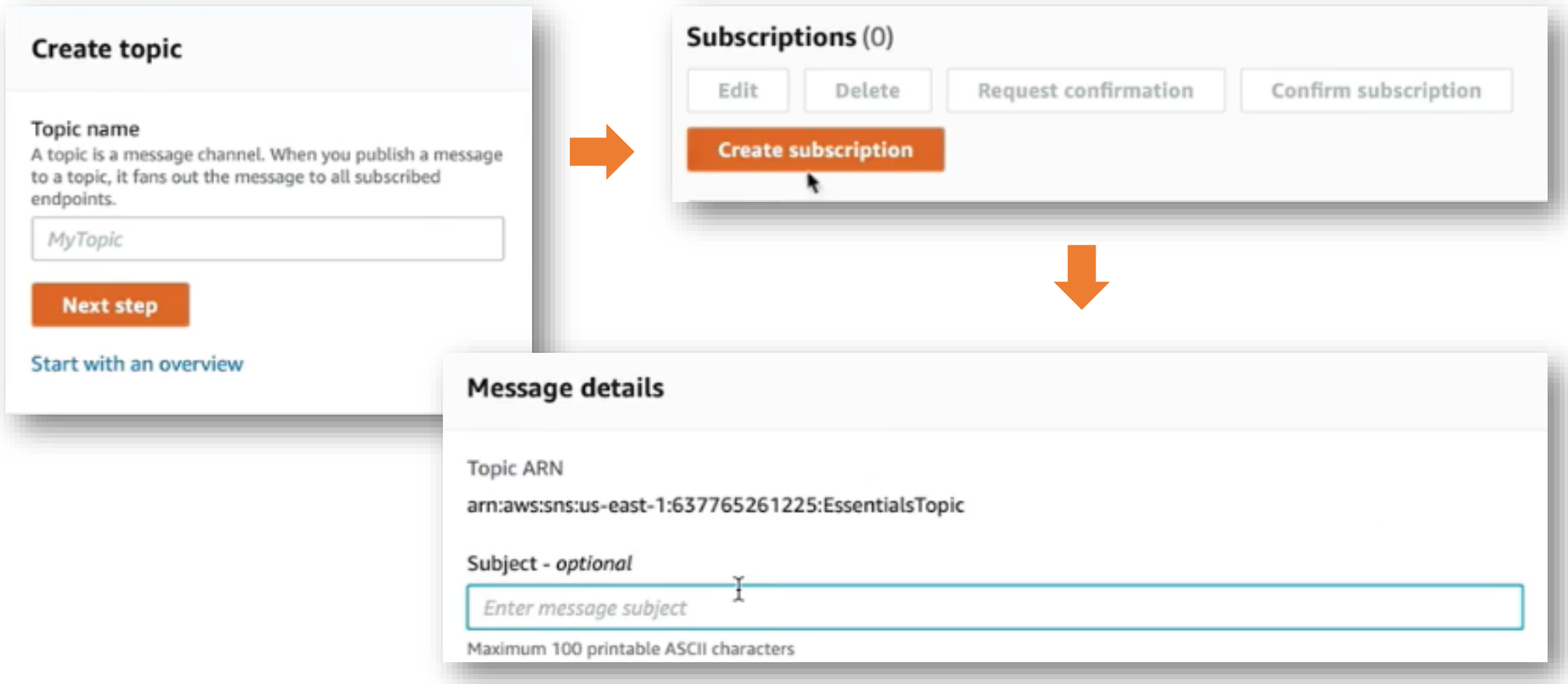
Application-to-application (A2A)

Amazon SNS lets you decouple publishers from subscribers. This is useful for application-to-application messaging for microservices, distributed systems, and serverless applications.

Application-to-person (A2P)

Amazon SNS lets you send push notifications to mobile apps, text messages to mobile phone numbers, and plain-text emails to email addresses. You can fan out messages with a topic, or publish to mobile endpoints directly.

Create a Topic, Subscription, Publish



Amazon SNS message filtering

By default, an Amazon SNS topic subscriber receives every message published to the topic. To receive a subset of the messages, a subscriber must assign a filter policy to the topic subscription.

You need to pass the **message attributes** (MessageAttributes) parameter which is a map consists of a key and value.

A policy that accepts messages

If any single attribute in this policy doesn't match an attribute assigned to the message, the policy rejects the message.

```
"MessageAttributes": {
   "customer_interests": {
      "Type": "String.Array",
      "Value": "[\"soccer\", \"rugby\", \"hockey\"]"
   },
   "store": {
      "Type": "String",
      "Value":"example_corp"
   },
   "event": {
      "Type": "String",
      "Value": "order_placed"
   },
   "price_usd": {
      "Type": "Number",
      "Value":210.75
   }
}
```

Message attributes in the message

Policy

```
{
   "store": ["example_corp"],
   "event": [{"anything-but": "order_cancelled"}],
   "customer_interests": [
      "rugby",
      "football",
      "baseball"
   ],
   "price_usd": [{"numeric": [">=", 100]}]
}
```

A policy that rejects messages

If any mismatches occur, the policy rejects the message. The encrypted attribute isn't present in the message, this causes the message to be rejected regardless of the value assigned to it.

```
"MessageAttributes": {
   "customer_interests": {
      "Type": "String.Array",
      "Value": "[\"soccer\", \"rugby\", \"hockey\"]"
   },
   "store": {
      "Type": "String",
      "Value":"example_corp"
   },
   "event": {
      "Type": "String",
      "Value": "order_placed"
   },
   "price_usd": {
      "Type": "Number",
      "Value":210.75
   }
}
```

```
{
   "store": ["example_corp"],
   "event": ["order_cancelled"],
   "encrypted": [false],
   "customer_interests": [
      "basketball",
      "baseball"
   ]
}
```

Simple Queue Service - SQS

SQS offers a secure, durable, and available hosted queue that lets you **integrate** and **decouple** distributed software systems and components (microservices).

SQS and SNS provides nearly **unlimited scalability**.

SQS **locks** messages during processing. The maximum message size is **256** KB. If the message is larger than 256 KB, you can store that message in S3 or DynamoDB and send the pointer or key as a message to the queue.

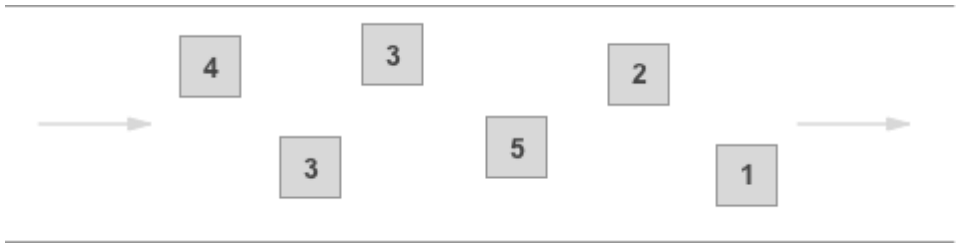
You can store messages up to **14 days**.

There are 2 types of queues, standard and FIFO.

Standard queue

Send data between applications when the throughput is important.

- **Unlimited throughput** - Standard queues support a nearly unlimited number of API calls per second, per API action (SendMessage, ReceiveMessage, or DeleteMessage).
- **At-least-once delivery** - A message is delivered at least once, but occasionally more than one copy of a message is delivered.
- **Best-effort ordering** - Occasionally, messages are delivered in an order different from which they were sent.



FIFO queue

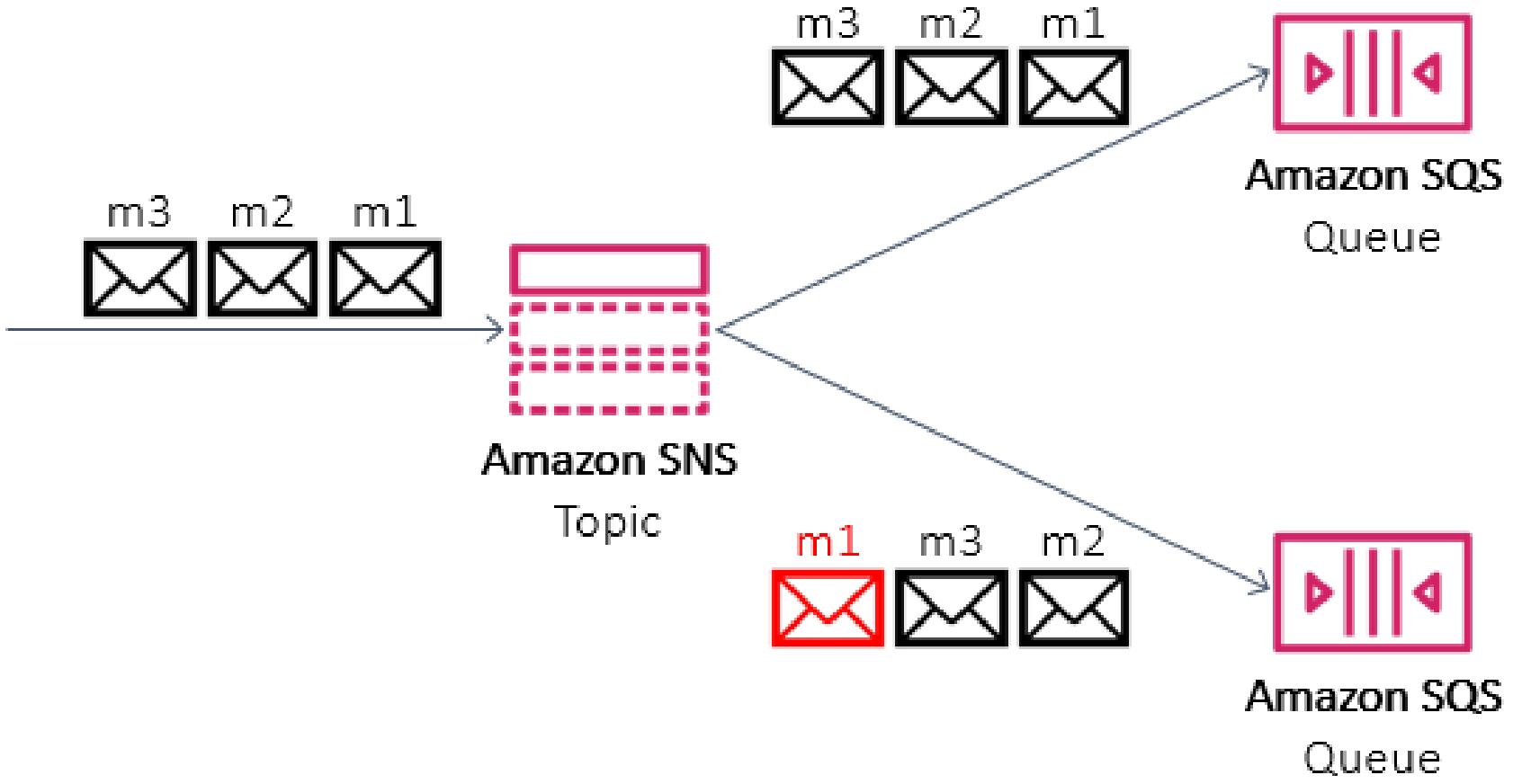
Send data between applications when the order of events is important. Data consistency is essentially zero message loss and strict message ordering.

- **High Throughput** - If you use **batching**, FIFO queues support up to 3,000 messages per second (300 API calls, 10 messages per call).
- **Exactly-Once processing** - A message is delivered once and remains available until a consumer processes and deletes it. Duplicates aren't introduced into the queue.
- **First-In-First-Out delievery** - The order in which messages are sent and received is strictly preserved.

Message ordering and deduplication concepts are also available in SNS.

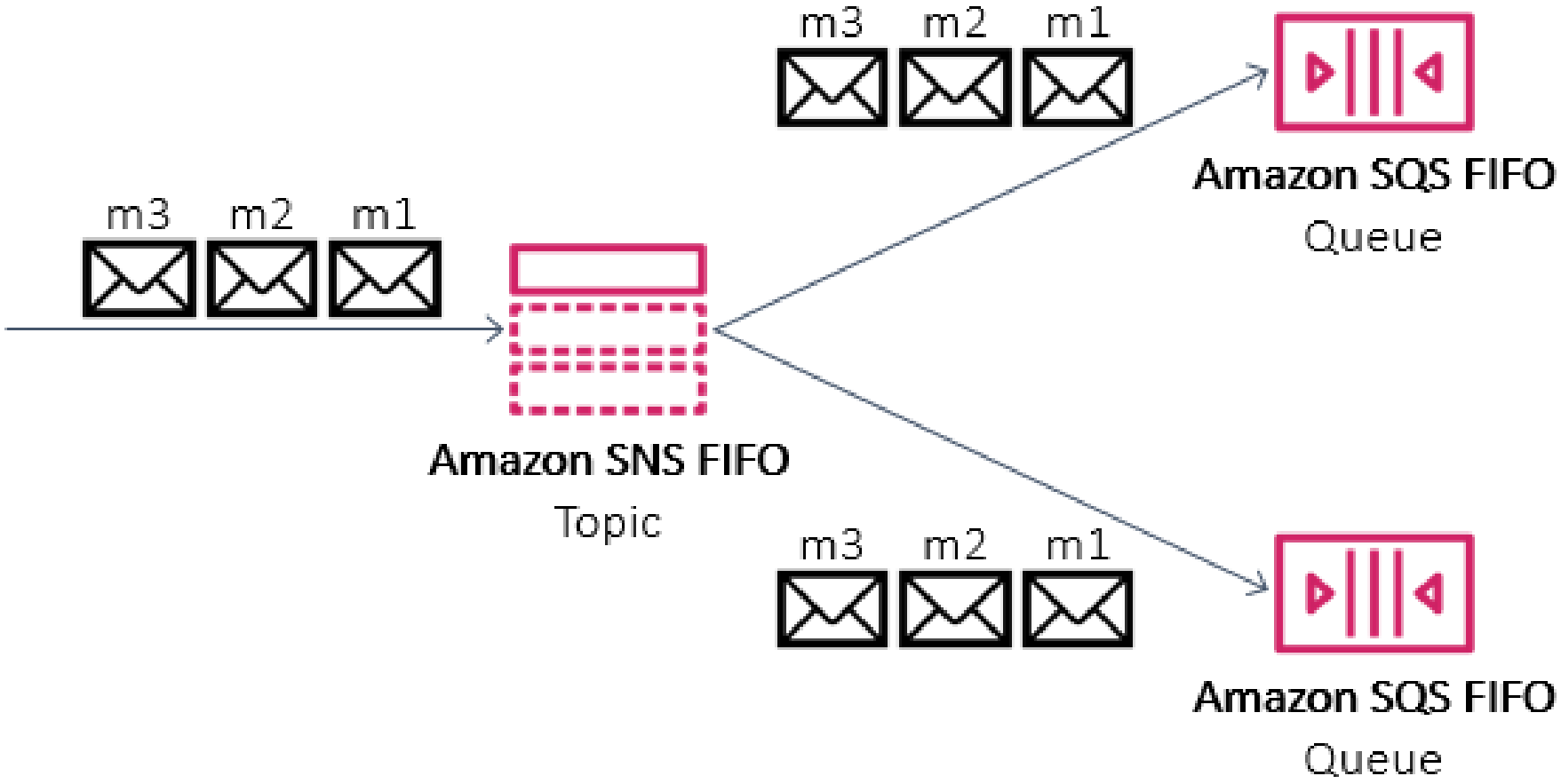
Best-effort message ordering

Standard SNS and SQS provide best-effort message ordering with rarely out of order delivery.



Strict message ordering

SNS FIFO and SQS FIFO provide strict message ordering.



Message deduplication ID

The message deduplication ID is the token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute **deduplication interval**.

If content-based deduplication for the queue is enabled. The producer can omit the message deduplication ID.

Provide deduplication id if:

- Identical bodies must be treated as unique.
- Different contents but must be treated as duplicates.
- Identical content but different attributes must be treated as unique.

Message group ID

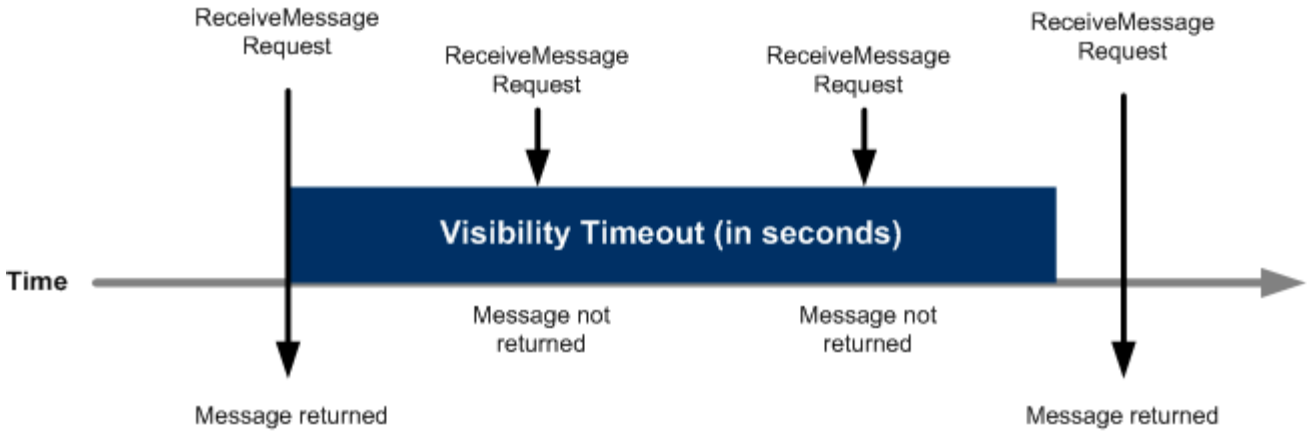
The message group ID is the tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one.

- Processing messages parallelly.
- Max number of inflight messages is 20,000. Use group ID to avoid it.
- There could be bottleneck on the group of messages if the current message is not being processed properly by a consumer.

A message is considered to be **in flight** after it is received from a queue by a consumer, but not yet deleted from the queue

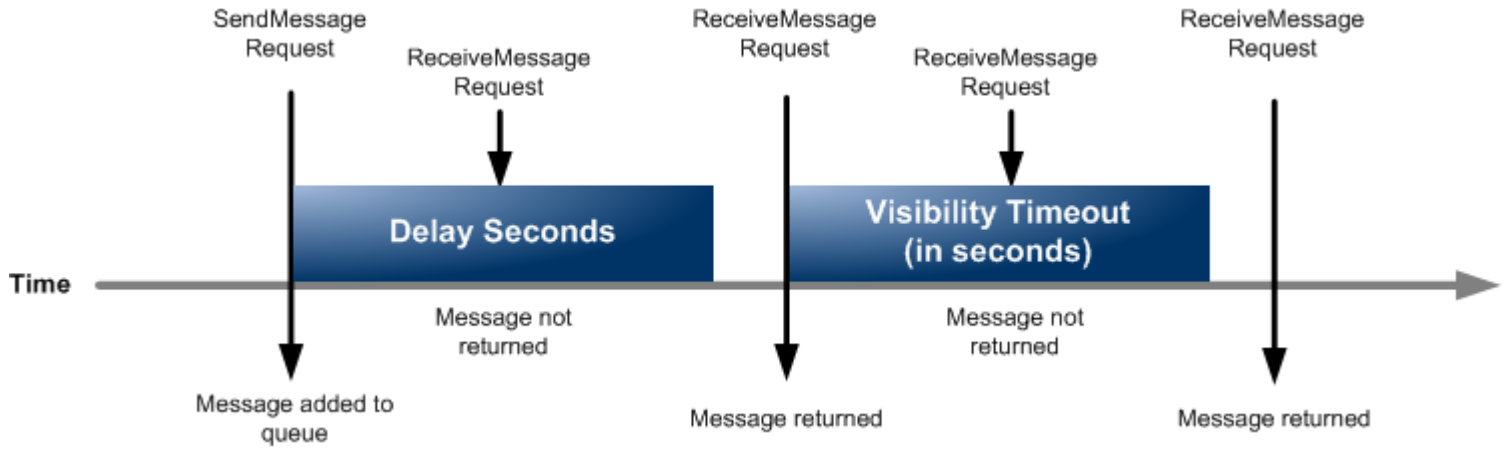
SQS visibility timeout

When a consumer receives and processes a message from a queue, the message **remains** in the queue. SQS **doesn't automatically delete** the message. Because SQS is a **distributed system**, there's no guarantee that the consumer actually receives the message due to a connectivity issue, or an issue in the consumer application. Thus, the consumer **must delete** the message from the queue after receiving and processing it.

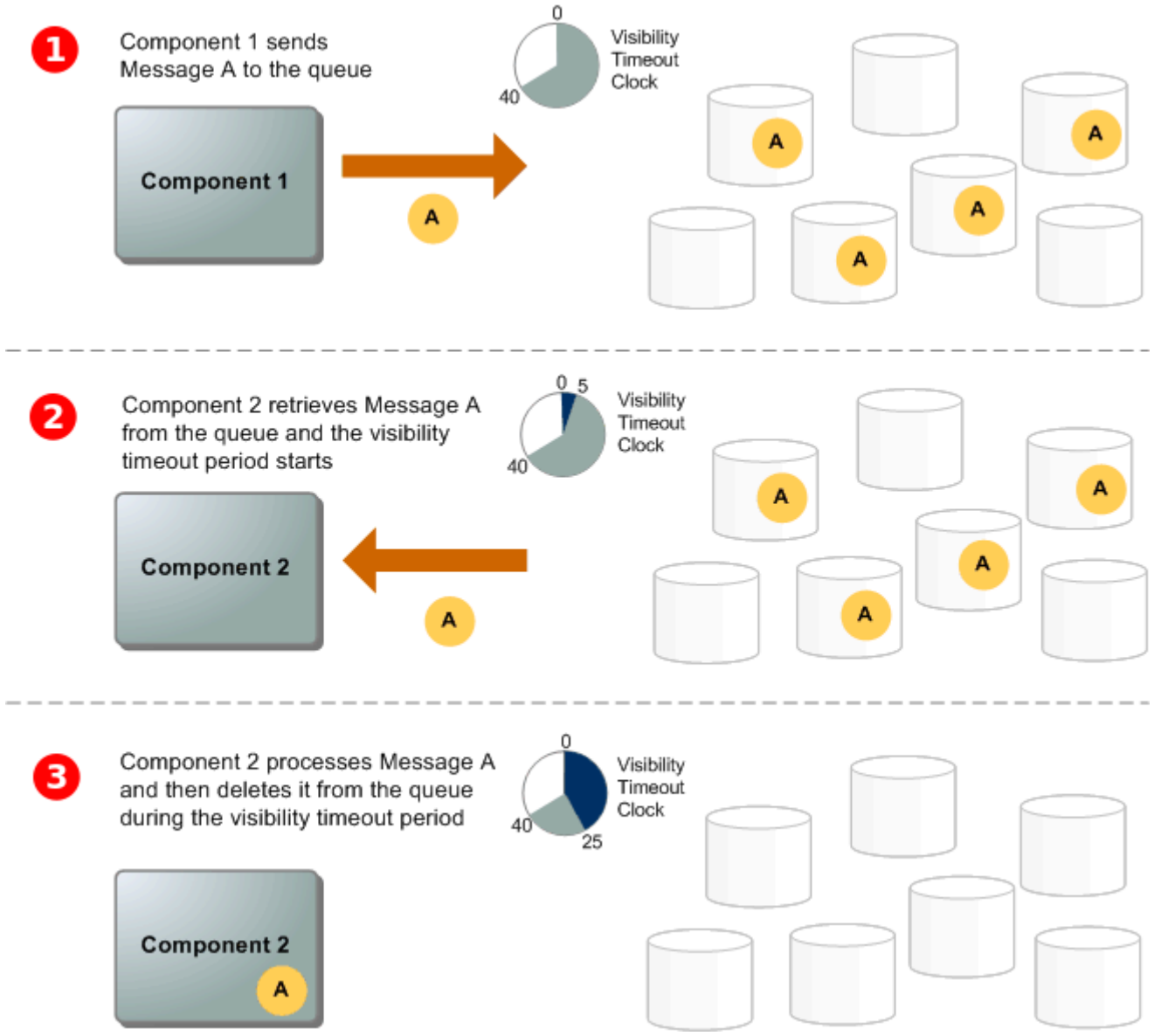


SQS delay queues

Delay queues let you postpone the delivery of new messages to a queue for a number of seconds, for example, when your consumer application needs additional time to process messages.



Message lifecycle



SQS short and long polling

Amazon SQS provides short polling and long polling to receive messages from a queue. By default, queues use short polling.

- With **short polling**, the ReceiveMessage request queries only a **subset of the servers** to find messages that are available to include in the response. Amazon SQS sends the response right away, even if the query found **no messages**.
- With **long polling**, the server keeps the client connection open and the ReceiveMessage request queries **all of the servers** for messages. Amazon SQS sends a response after it collects **at least one available message**, up to the maximum number of messages specified in the request. Amazon SQS sends an empty response only if the polling wait time expires.

Amazon SNS dead-letter queue

A **dead-letter queue** is an Amazon SQS queue that an Amazon SNS subscription can target for messages that **can't be delivered to subscribers successfully**.

Messages that can't be delivered due to **client errors or server errors** are held in the dead-letter queue for further analysis or reprocessing.

You can have a dead-letter queue for other AWS services as well.

Application Decoupling

Many applications start to grow in complexity as they mature, making it harder for developers to maintain code or add new features. Decoupling helps these issues.

- Easier to maintain code and change implementations
- Cross-platform, different languages and technologies
- Independent releases
- Better application performance by doing the action asynchronously.
- Fault tolerant – You can keep pushing messages back to the queue if the listener service is not working.