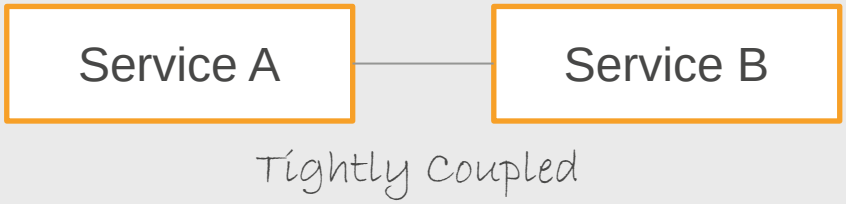
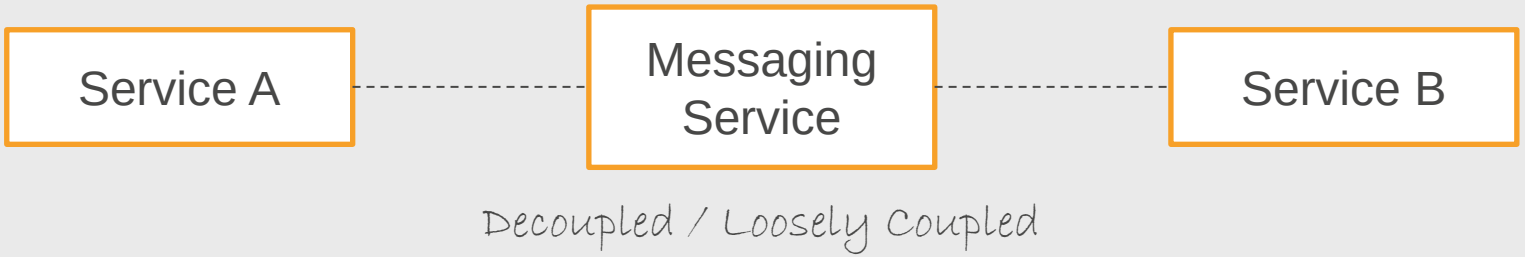
Messaging

Queues, Notification Service, Event Bus, Streams

Tight, Loose Coupling





Tightly coupled

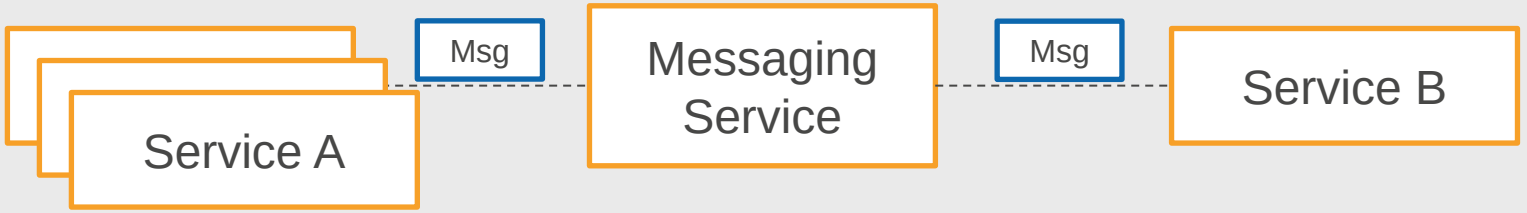
Pros

- Easy to understand
- Dependencies are clear
- Simple to design

Cons

- Ripple effect due to failure
- Scaling challenges
- No buffering or shock absorber

Loosely Coupled



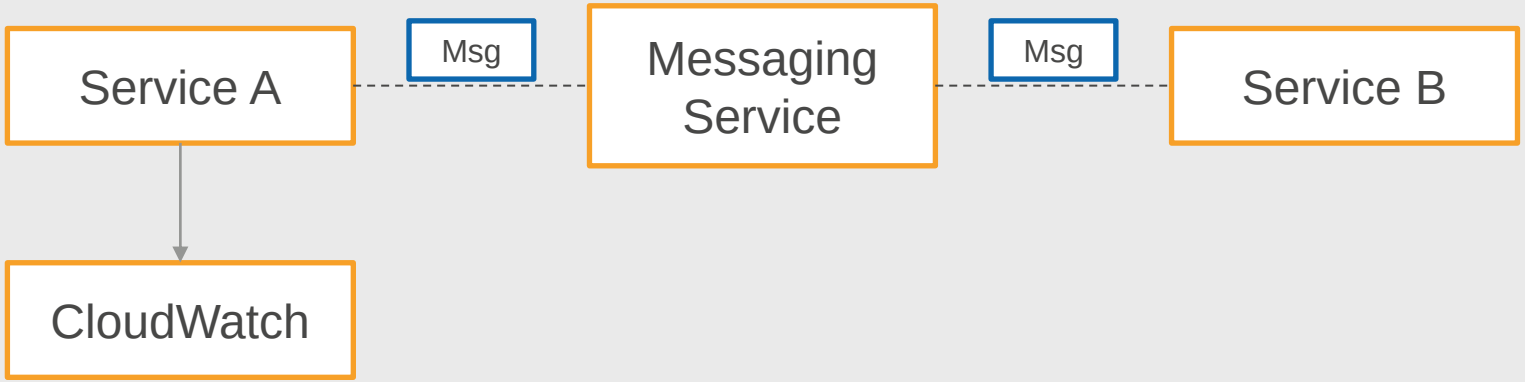
Pros

- Services are decoupled
- Scale, Fail independently
- Buffer to handle sudden surge in workload
- Resiliency to handle temporary or intermittent issues

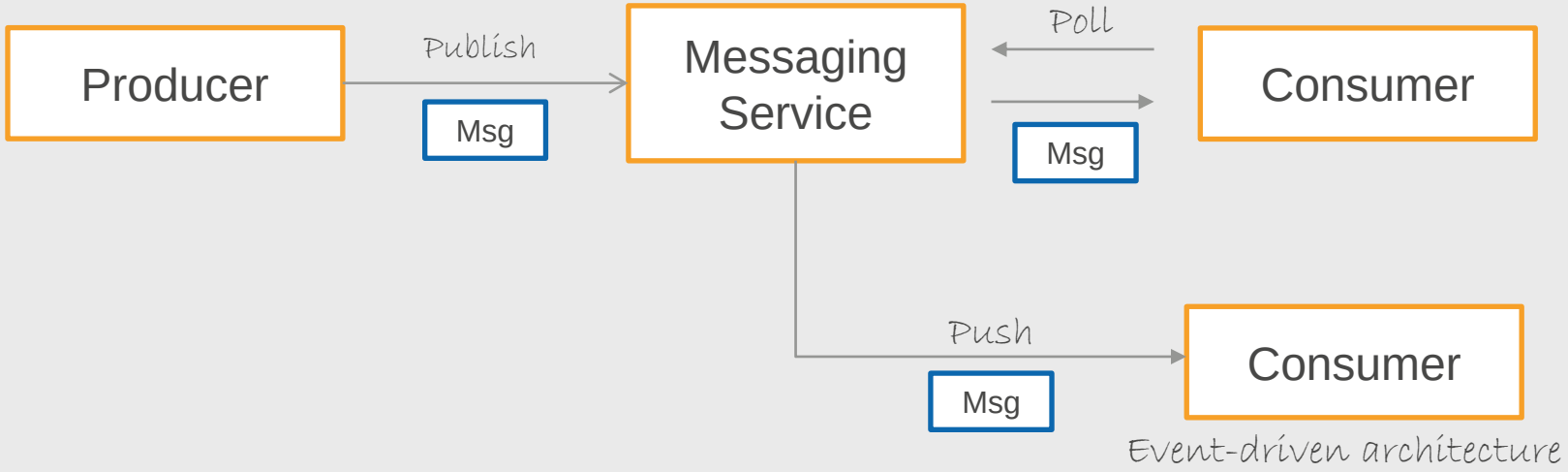
Cons

- More complex
- Communication over network
- Additional failure points, delay

Mix of both



Poll versus Push



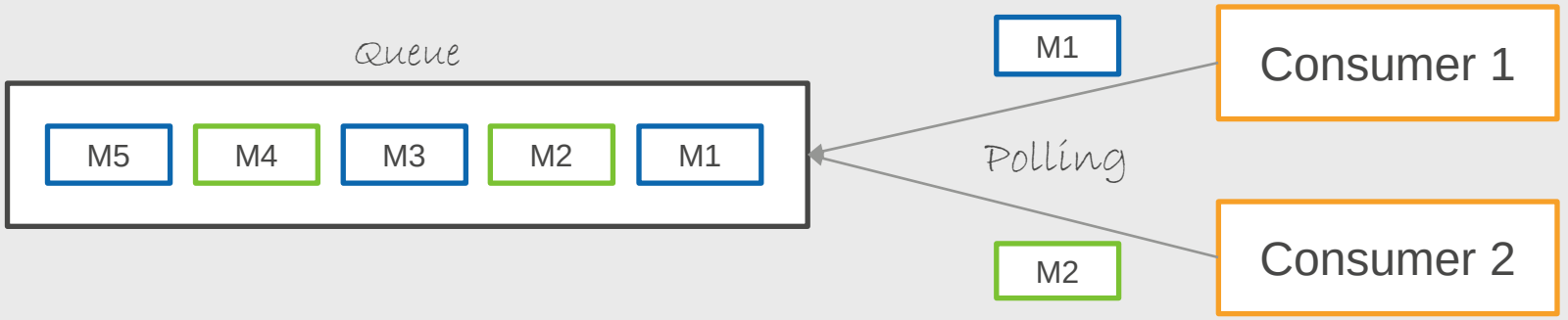
AWS Messaging Services

1. Queues
2. Notification Service and Event Bus
3. Streams

Queue Message Processing Semantics

Consumers poll the queue for any new messages

A message is processed once and removed



AWS Queuing Services

Simple Queue Service (SQS)

- Highly scalable, simple to use
- Recommended for new applications on AWS

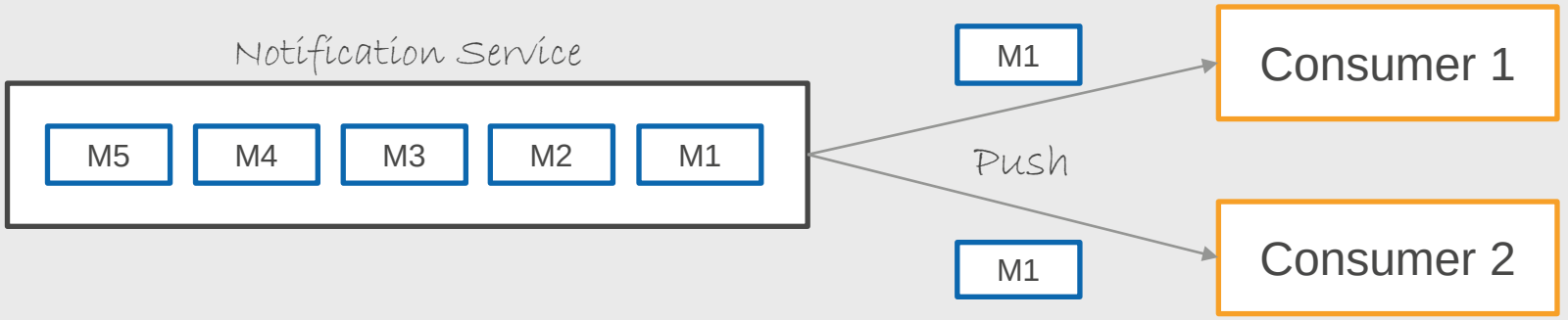
Amazon MQ

- Support for Apache ActiveMQ, RabbitMQ
- Migrate existing applications to cloud

Notification Service and Event Bus

Broadcast a message to many consumers

Push notification and Event Driven Architecture



AWS Services for Push-notification

Simple Notification Service (SNS)

EventBridge (previously known as CloudWatch Events)

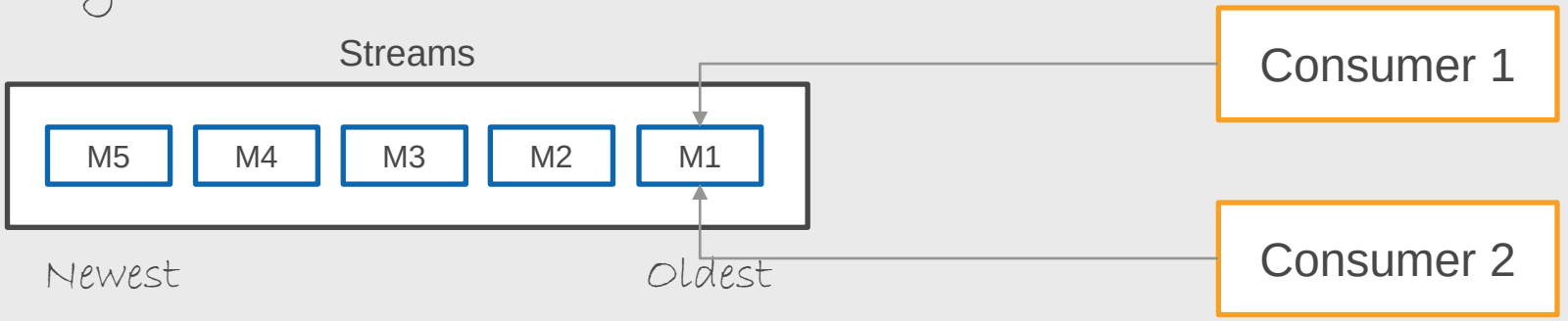
Streams

Streams maintains a history of all messages, ordered by arrival time

Messages are called "records"

Records are kept for the retention duration

Many consumers can read all the records



AWS Services for Streaming

Kinesis

- Data Streams – ingest vast amount of streaming data, custom real-time applications
- Firehose – Load streaming data to S3, Redshift, Splunk, Elasticsearch

Managed Apache Kafka (MSK)

- AWS managed opensource streaming product

Recap

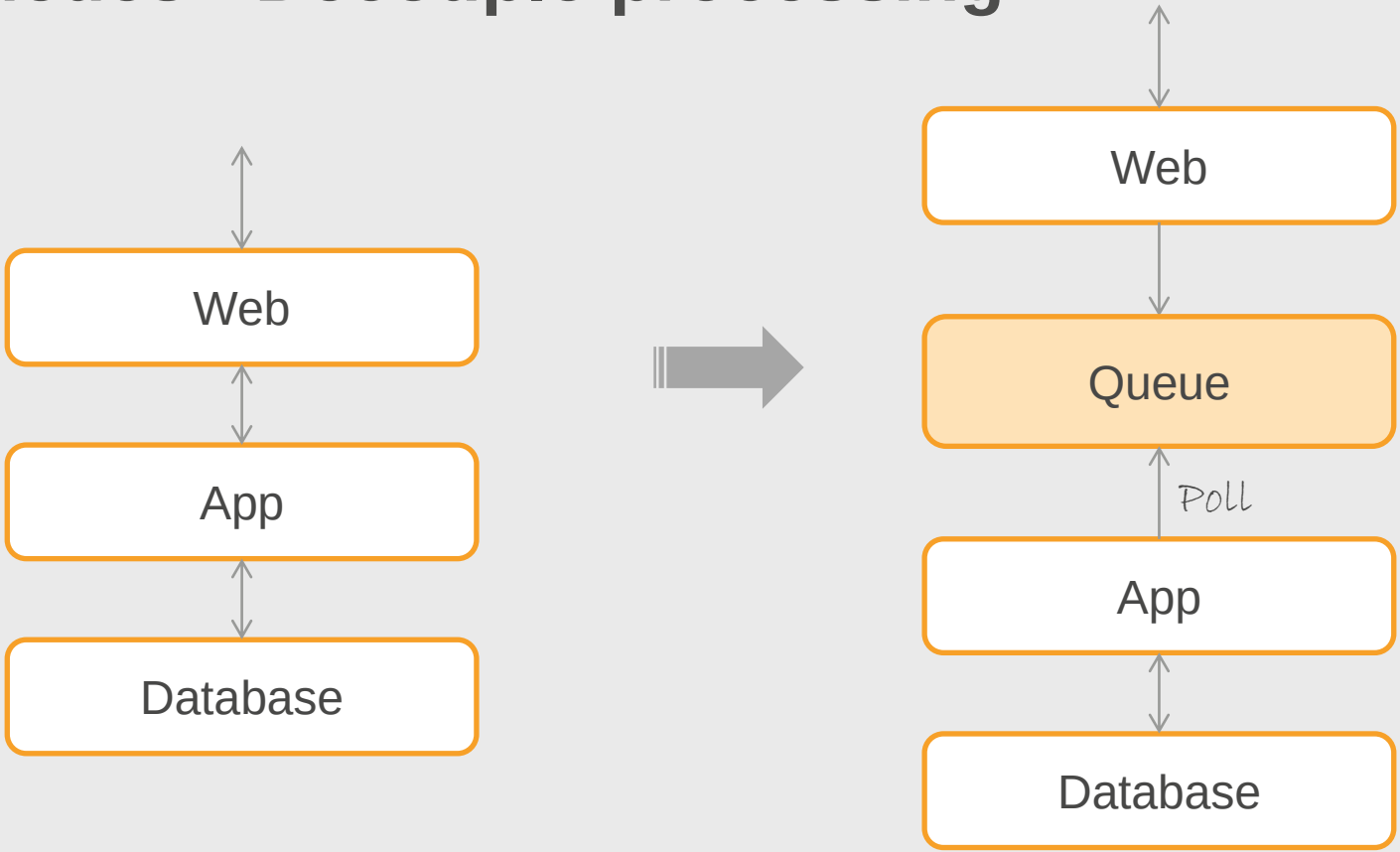
Modern applications are built-on decoupled services

App components communicate using messaging service

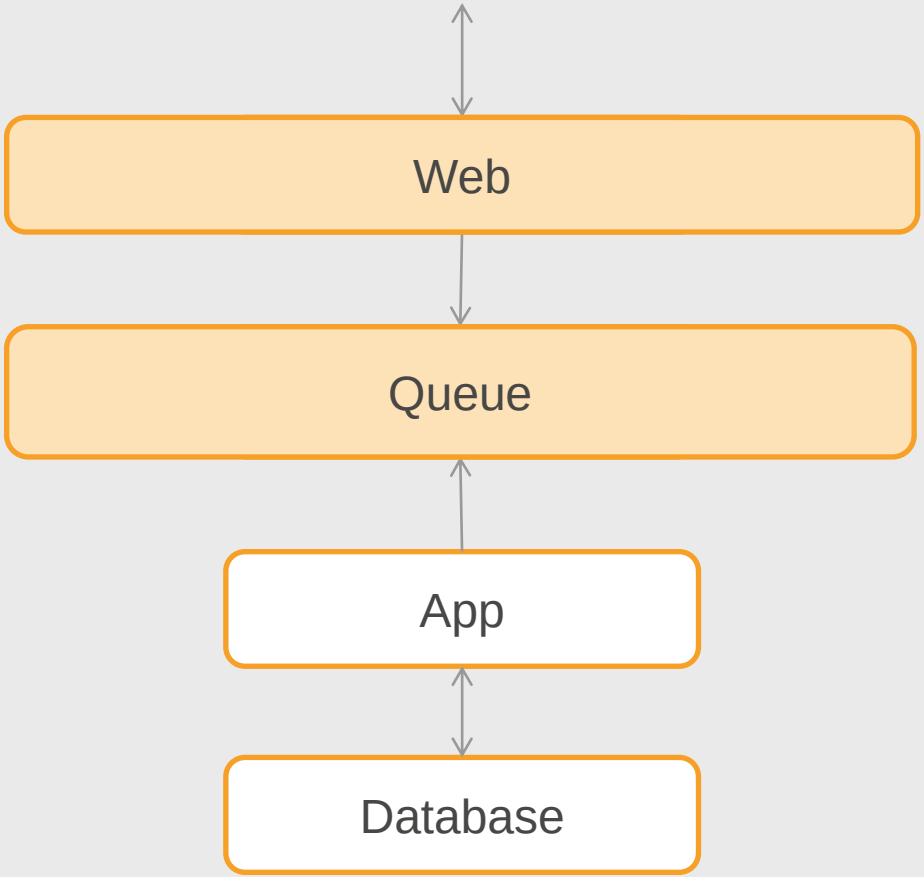
Scale and fail independently

Queues, Notification Service and EventBridge, Streams

Queues - Decouple processing



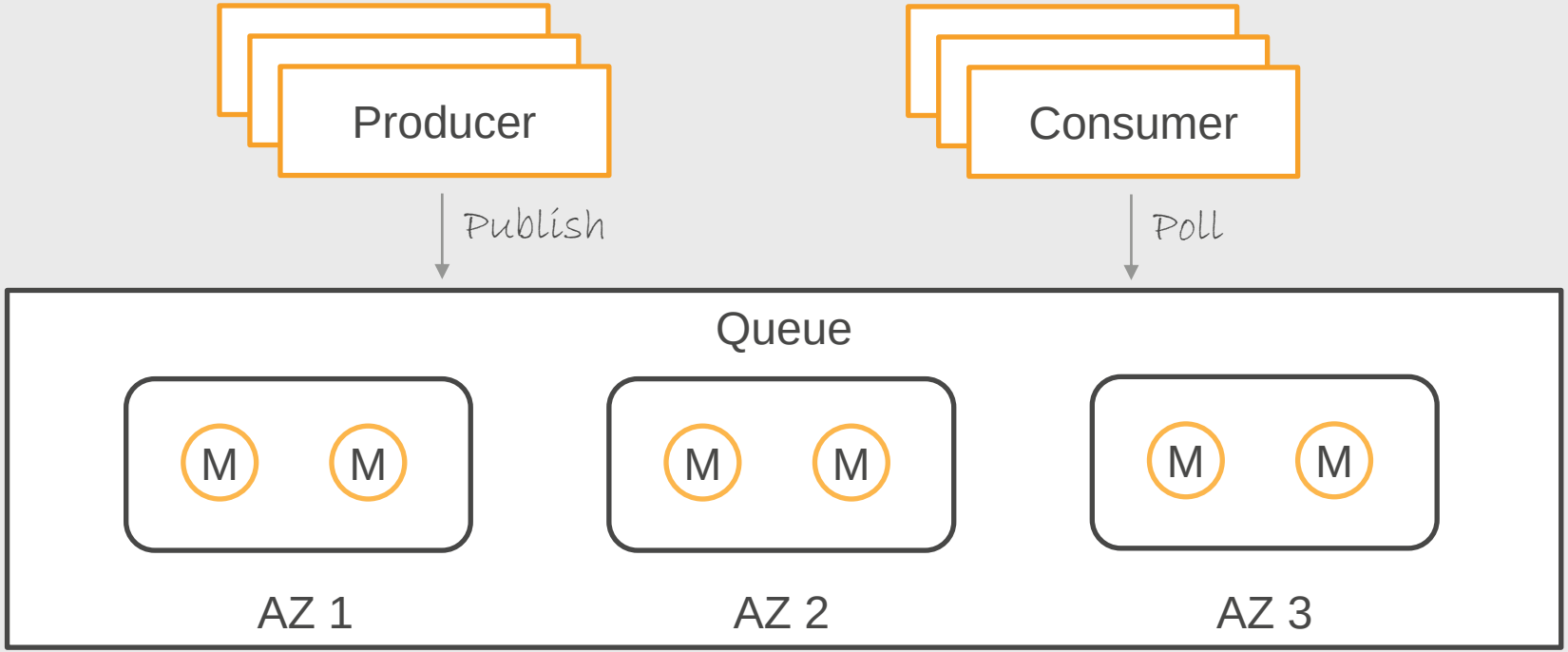
Decouple Layers using a Queue - Scaling



- Scale web layer to handle traffic increase
- Queue is elastic and automatically scales
- App layer can scale based on pending queue items and/or limit imposed by database
- Shield from intermittent failures

SQS

Messages are replicated in multiple AZ (handles AZ and device failures)
Multiple producers and consumers can simultaneously use the queue



SQS Types

Standard – approximate ordering of messages

FIFO – strict ordering of messages

Standard Queue Analogy

People waiting for checkout

Processing order is approximate

Photo by: Velela, https://commons.wikimedia.org/wiki/File:Supermarket_check_out.JPG

FIFO Queue Analogy

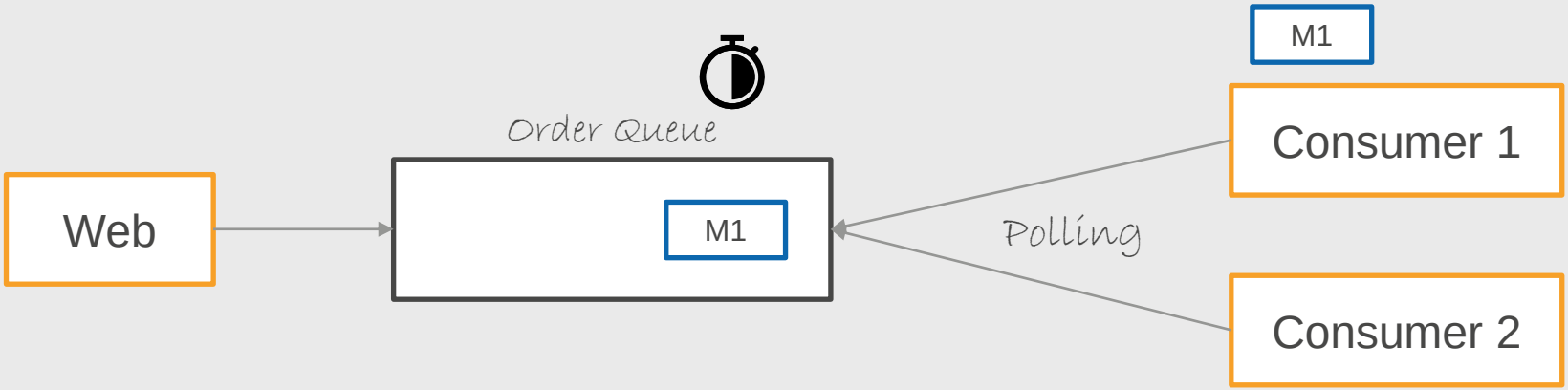
Food needs to be delivered in a specific order – Appetizer, Main course and Dessert

FIFO preserves ordering of messages

Message group to order related messages (maintain order within a table, but process different tables in parallel)

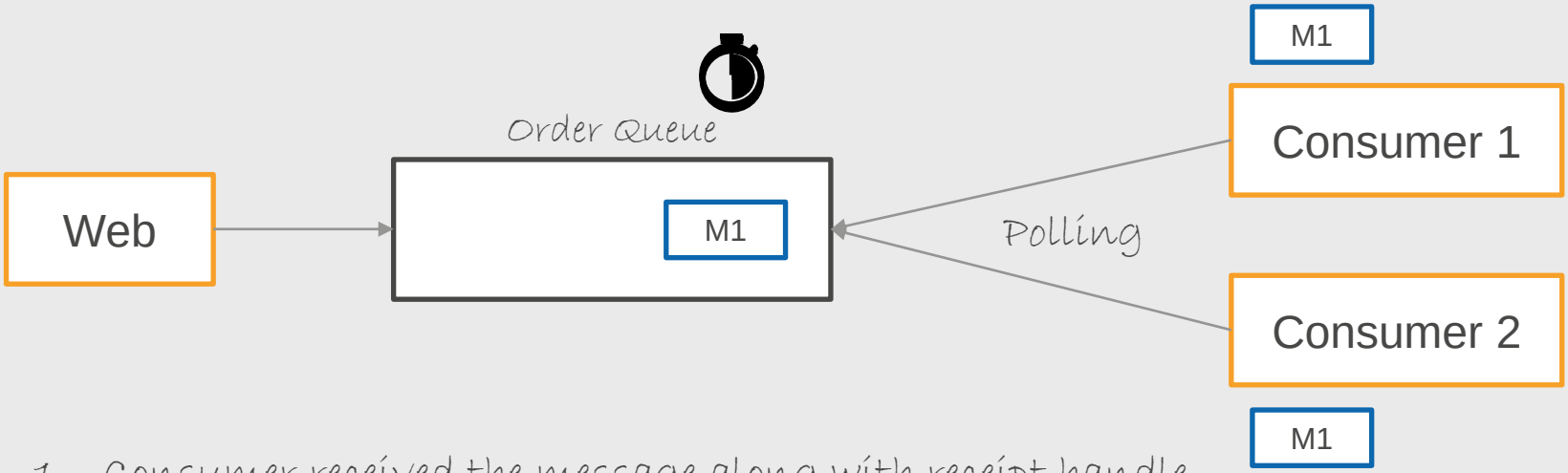
Photo by: Alan Light, https://commons.wikimedia.org/wiki/File:Waitress_taking_an_order.jpg

Standard Queue - Lifecycle of a message



1. Consumer received the message and receipt handle
2. Queue locked the message and started a timer (Visibility Timeout)
3. Consumer processed the message before timeout expired
4. Consumer deleted the message by providing the receipt handle
5. All good!

Standard Queue – Failure Scenario



1. Consumer received the message along with receipt handle
2. Queue locked the message and started a timer (Visibility Timeout)
3. Consumer crashed
4. Queue waited until message visibility timeout expired
5. Message not deleted
6. Queue unlocked the message and assigns to next consumer

Visibility Timeout

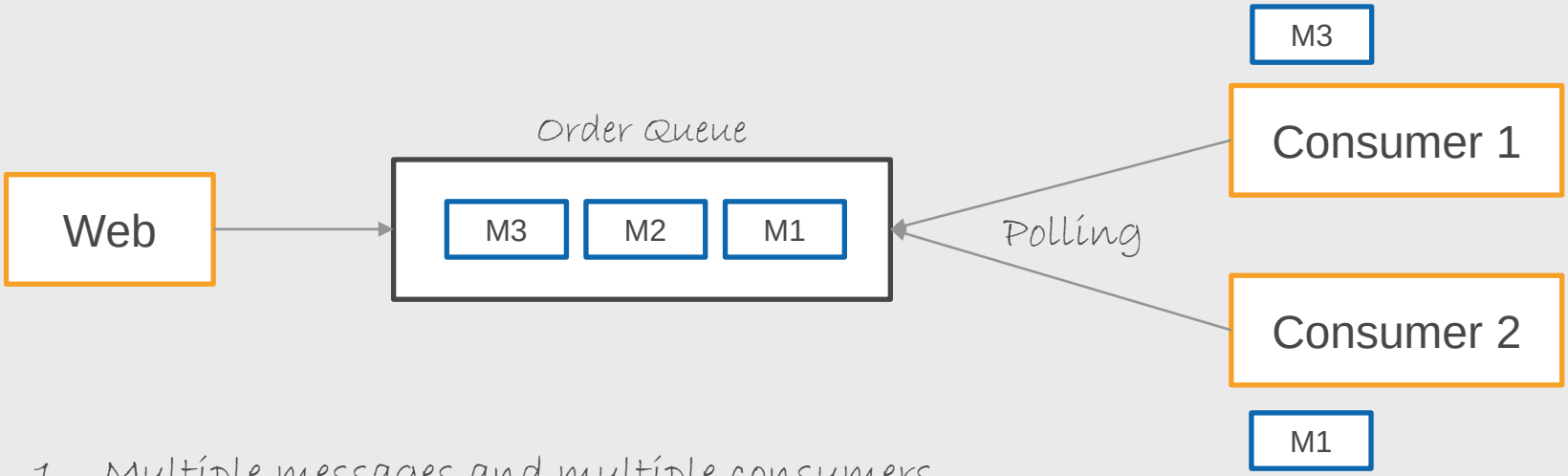
Visibility timeout used for handling failures

Default timeout is 30 seconds, maximum is 12 hours

Queue level configuration

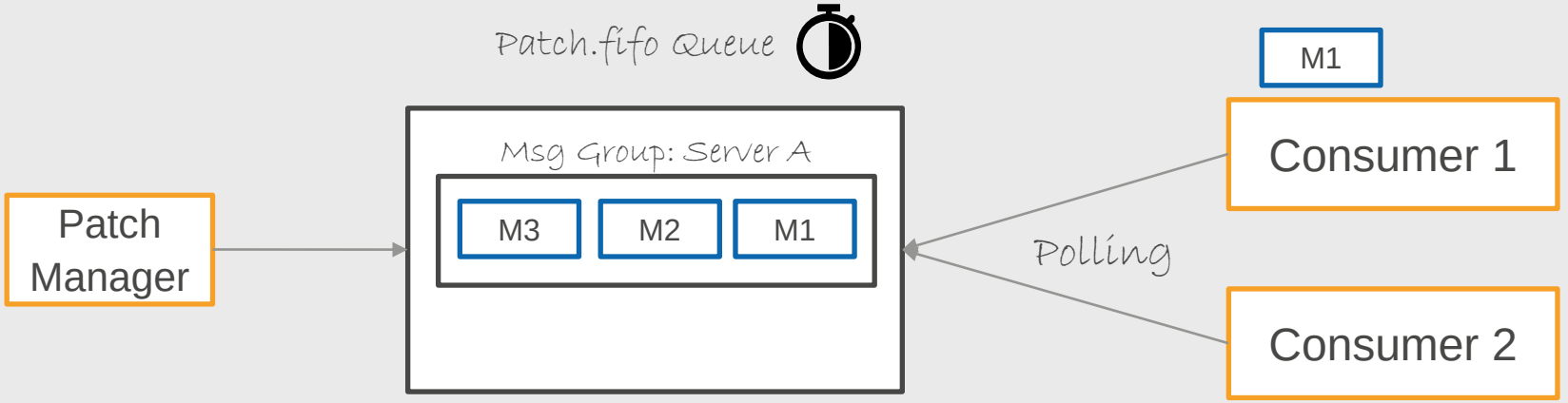
Consumer can increase timeout for a specific message

Standard Queue – Multiple message



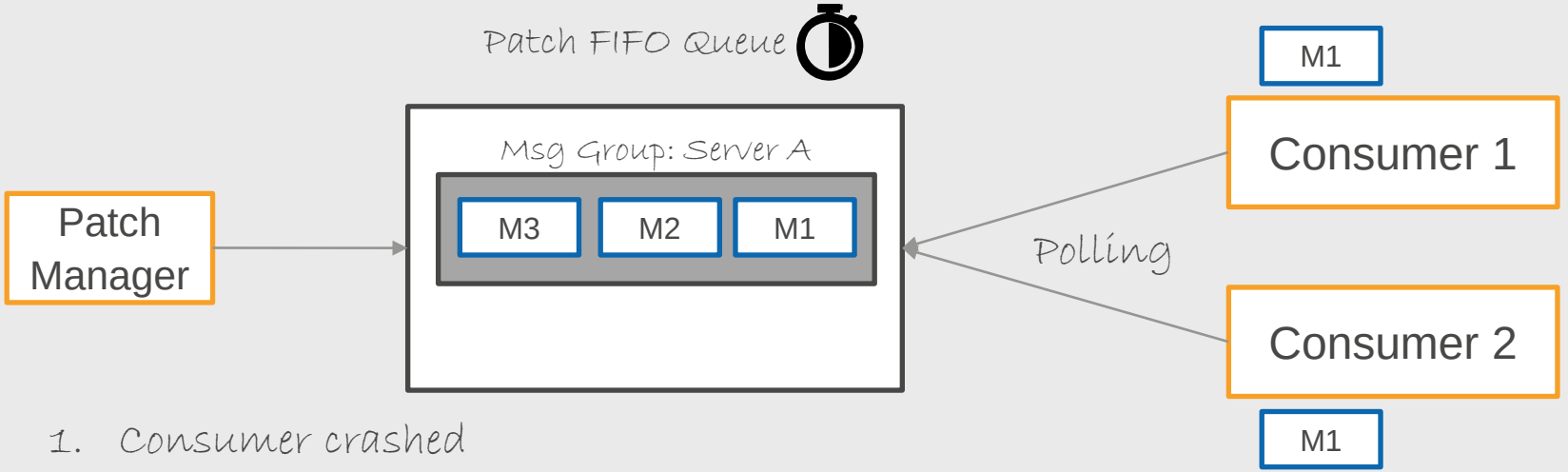
1. Multiple messages and multiple consumers
2. Messages are processed concurrently
3. Message arrival order and processing order is different in a standard queue

FIFO Queue - Lifecycle of a message



1. Consumer receives the message and receipt handle
2. Queue locks the message group and starts a timer for the message (Visibility Timeout)
3. Consumer processes the message before timer expires
4. Consumer deletes the message by providing the receipt handle
5. Queue unlocks the message group
6. Another consumer can now receive the message

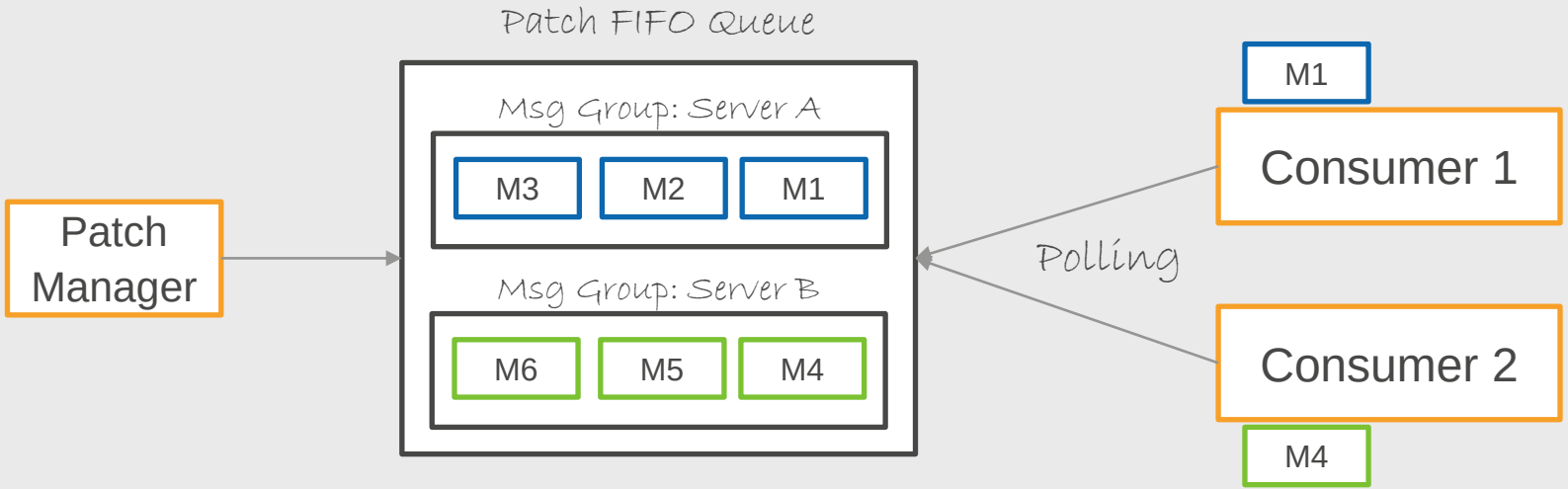
FIFO Queue – Failure Scenario



1. Consumer crashed
2. FIFO Queue waits for visibility timeout to expire
3. FIFO Queue unlocks the message group
4. Assigns the first message to the next consumer
5. All good!

FIFO Queue – Message Group

1. Message group attribute – free to use any value
2. Messages in a message group are processed sequentially
3. Messages in different message groups are processed in parallel
4. FIFO queue decides which message group to process next



SQS Message Lifecycle - Summary

Message lifecycle – receive, process, delete

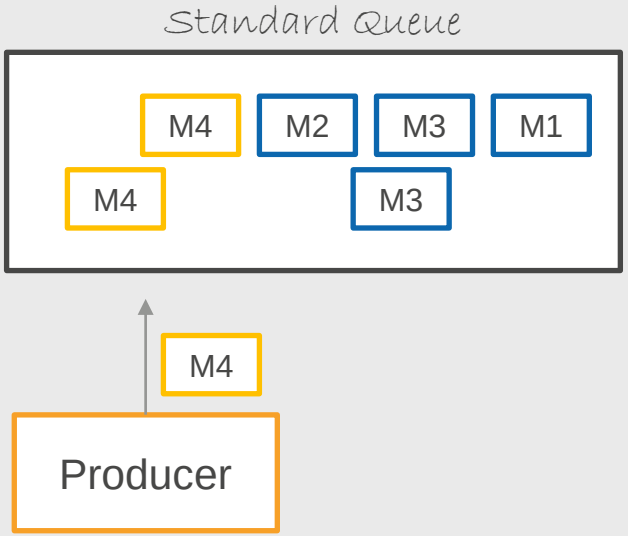
SQS uses visibility timeout to detect failure of consumers

Queue will unlock message with expired visibility timeout and make it available to other consumers

Standard queue – concurrent processing of different messages

FIFO queue – messages in the same message group are processed sequentially, and multiple groups are processed in parallel

Standard Queue – duplicate message



Designed for very high throughput

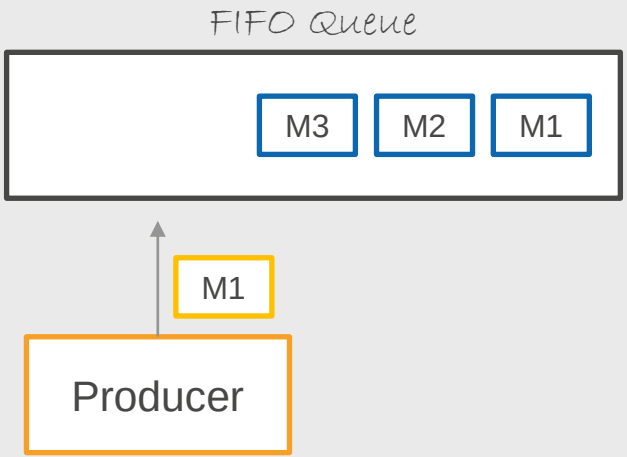
Approximate ordering of message

Duplicate messages possible

- Standard queue may send same message to multiple consumers (rare)
- Producer can introduce duplicate message (during retry)
- Consumer did not complete before visibility timeout

Consumer needs to have logic to see if a message was already processed. For example, producer adds a unique ID to a message that consumer can check against a database

FIFO Queue – No duplicate message



Preserves ordering of message

Less throughput when compared to Standard queue

- 300 messages per second
- 3000 messages per second (with batching)

Duplicates

- FIFO does not introduce duplicates
- Content-based deduplication - Automatic check if a message is a duplicate of an existing message (rolling 5-minute window)
- Content-based deduplication (Disabled) - Specify deduplication-ID attribute in message

Consumer needs to have logic to see if a message was already processed

SQS – Long Polling

When queue is empty, consumers will keep polling the queue

- Lots of empty responses
- Increase in usage charge

receive-message call supports wait-time parameter

- Short-polling (wait time is zero) - Call will return immediately
- Long-polling (wait time is greater than zero)
  - Call waits until new message is received or wait time expires
  - Maximum wait time is 20 seconds
  - Reduce number of calls, minimize empty responses, and eliminate false-empty responses

SQS Batching

Publish up to 10 messages in a single send-message-batch call

Receive up to 10 messages (max-number-of-messages parameter)

Delete and Change visibility timeout also supports batch

Reduce the number of round trips

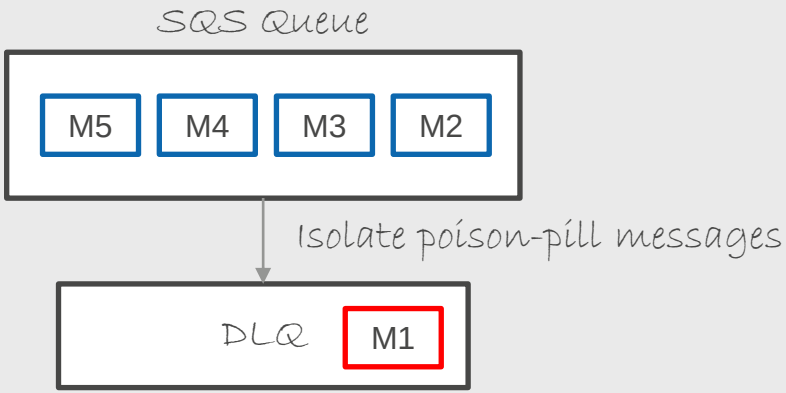
SQS Retention

Keep unprocessed messages for up to 14 days

Queue level configuration

Unprocessed messages older than retention period are removed

Dead Letter Queue (DLQ)



Poison pill messages

- Messages cannot be processed by consumer
- Bug in consumer
- Unexpected message format or content

Receive Count – Tracks number of times a message was received by consumer

Large number of retries indicates a poison pill message

SQS can move messages that exceed a specified receive count to a DLQ

- DLQ must be of same type as the source queue
- Redrive-policy setting

SQS Encryption of messages

At-rest encryption using server-side encryption

Use AWS managed Key or specify your customer master key

Keys are managed in KMS (Key Management Service)

Metadata is not encrypted; only message body

- Don't store sensitive information in metadata
- Up to 10 metadata attributes per message

SQS Payload

Maximum message size is 256 KB

For larger messages, store content in S3 and reference the S3 location in the message body

Billing - request is counted in increments of 64 KB

- A 256 KB message is counted as four requests

SQS - Summary

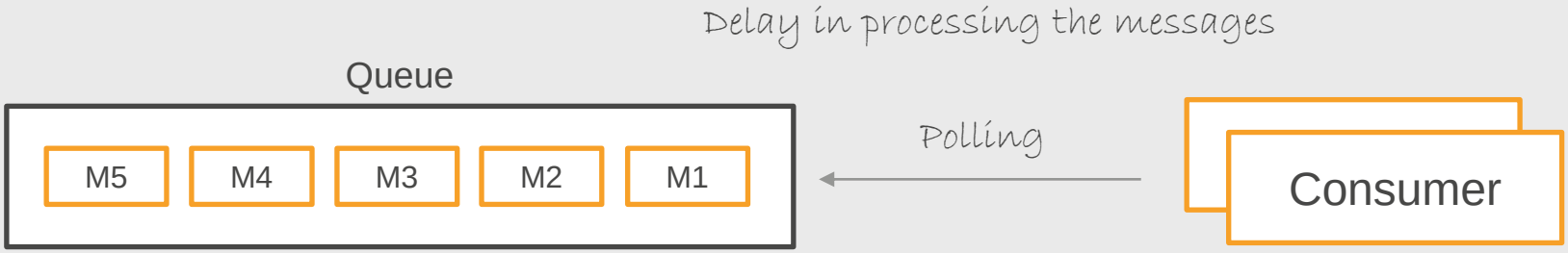
Queues are used in decoupled systems

Shield various components in a system

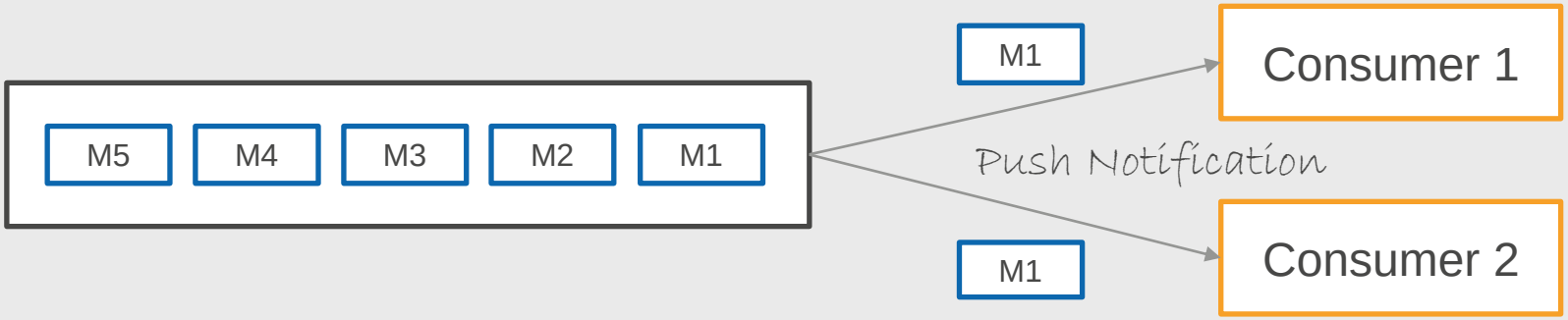
- Buffer message (to absorb sudden increase in traffic)
- Handle intermittent errors (Consumer or producer down)

Two types of SQS queues: Standard, FIFO

Issue with Queue – Processing Delay



Notification Service and EventBridge



Broadcast a message to many consumers (Fanout)

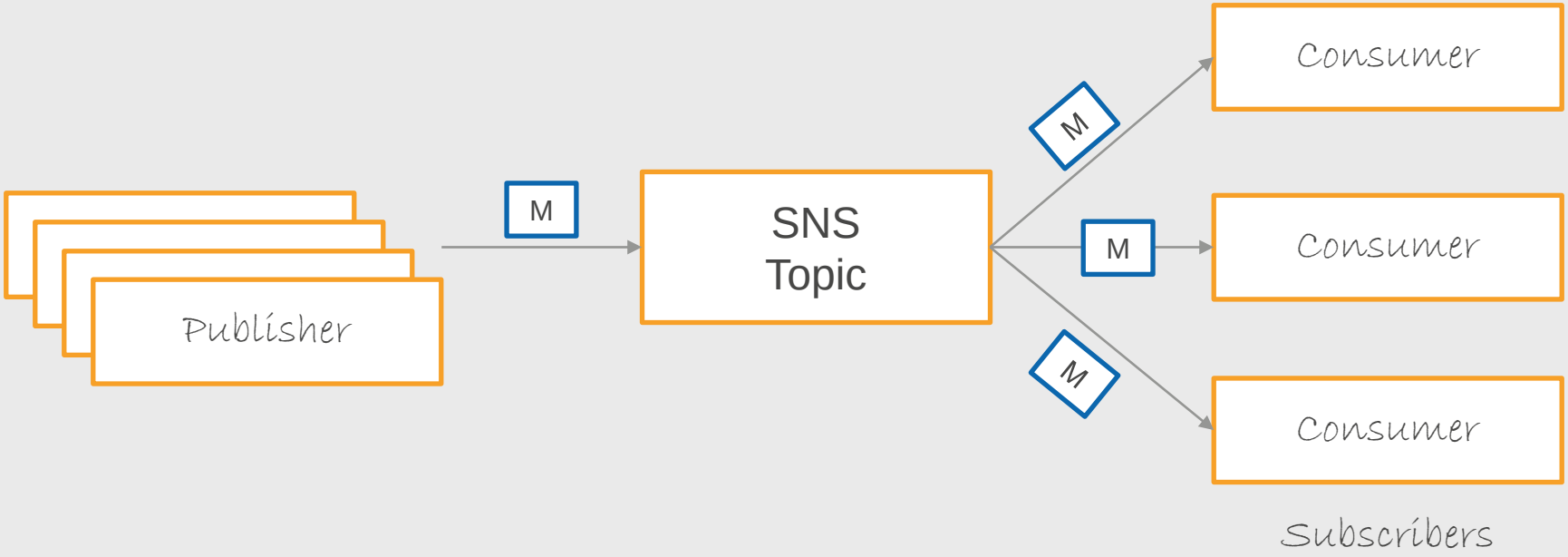
Event Driven Architecture

Event Driven Applications

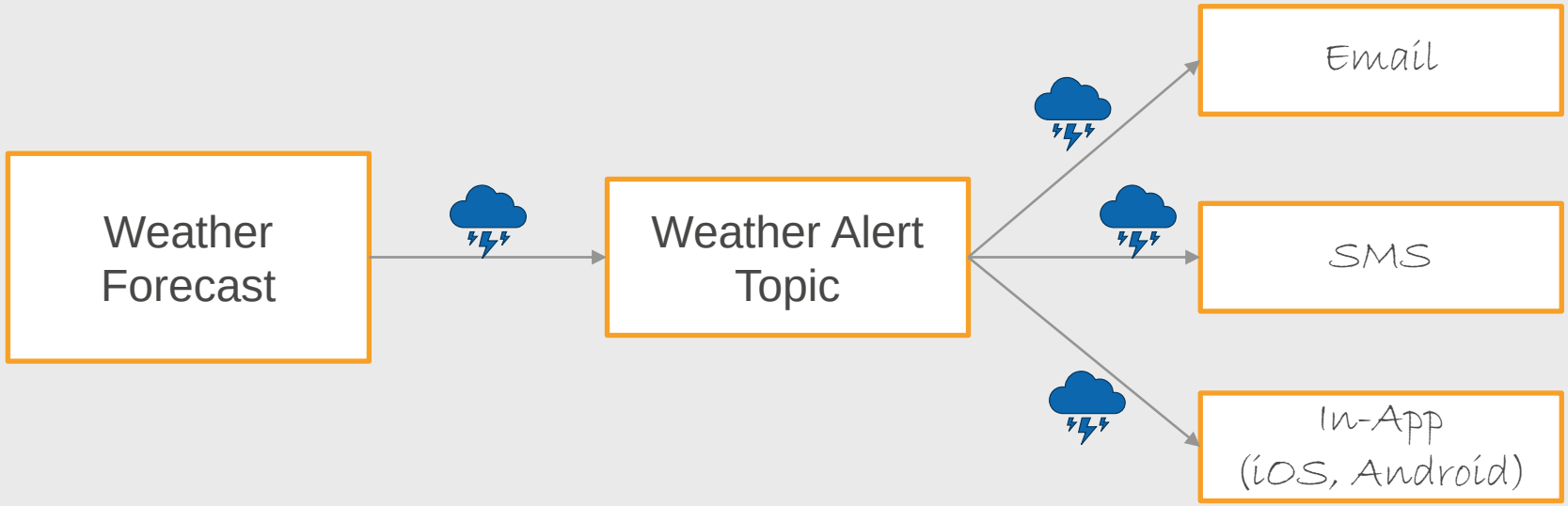
1. Simple Notification Service (SNS)

2. EventBridge (CloudWatch Events)

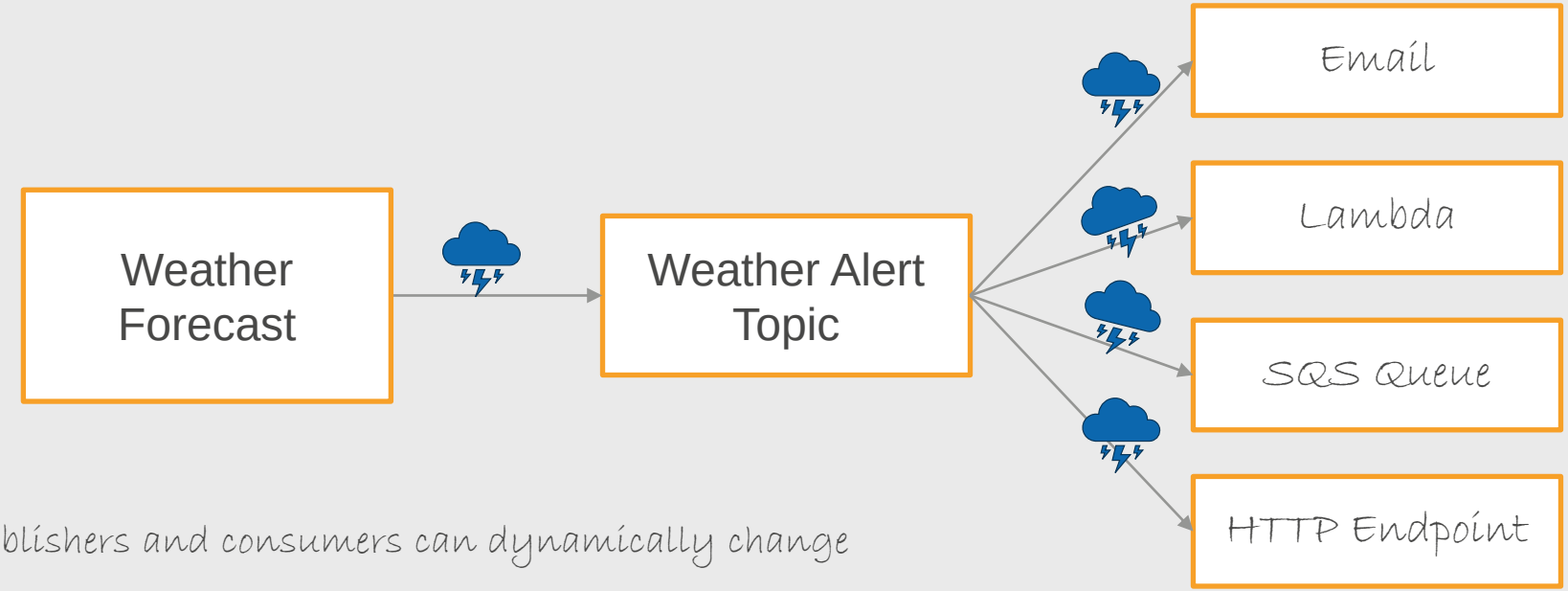
SNS



SNS – Application to Person



SNS – Application to Application



Publishers and consumers can dynamically change

SNS decouples publishers and consumers

SNS has automatic retry if subscriber is not online

SNS – Lambda

Lambda is very popular as it is serverless

Write your function

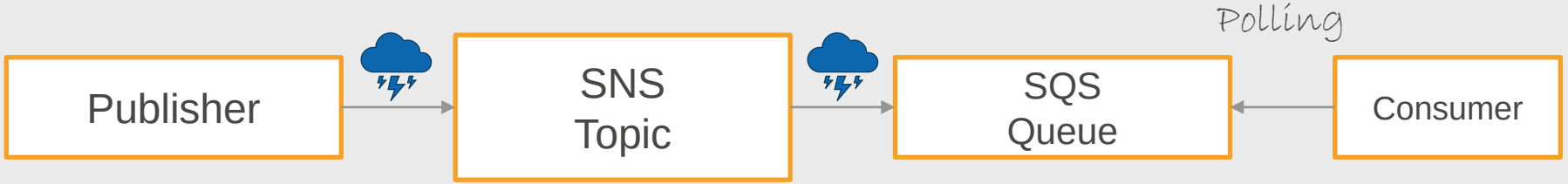
Subscribe to SNS topic

Lambda function is invoked when there is a new message

SNS – HTTP Endpoint

HTTP Endpoint is useful to invoke your service in response to an event

SNS – SQS



SQS Queue to hold the message until consumer is ready to consume

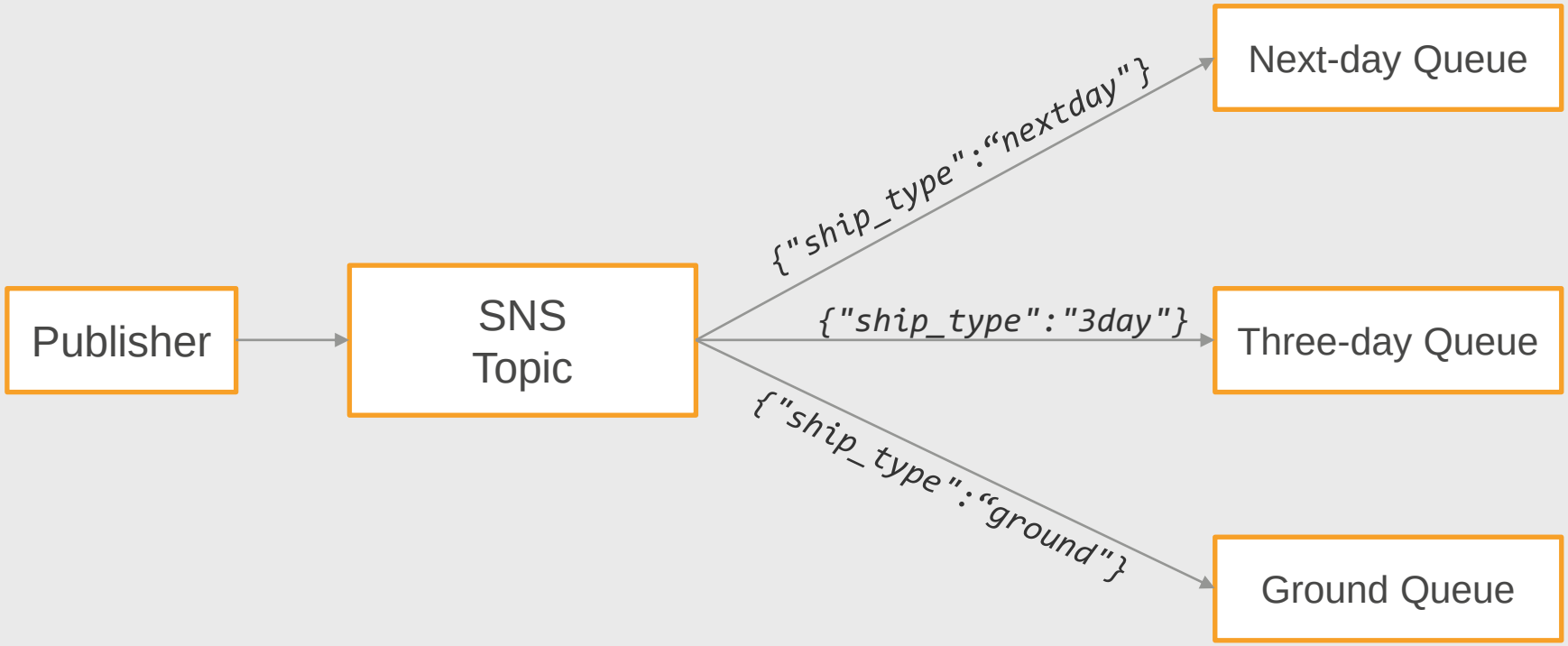
Handles scenario where consumer is down possibly for an extended period

SNS Topic Types

1. **Standard Topic**
   - Suitable for many scenarios, flexible subscriber endpoints
   - High Throughput
   - Out of order and duplicate messages possible

2. **FIFO Topic**
   - Preserves order of messages
   - Message group and deduplication (similar to SQS FIFO)
   - Subscriber: SQS FIFO Queue

SNS Filtering



use filtering to deliver messages to the correct subscriber

SNS Failure and Retry

HTTP Endpoint is down

Lambda function deleted

Security policy changes (topic does not have permission to invoke the lambda function)

Too many messages – throttling at subscriber

SNS Retry

No retry for permission related issues

AWS managed endpoints SQS and Lambda

Throttling or service availability issues
SNS retries using exponential back off strategy
Increase delay between each retry

Consumer endpoints Email, SMS, mobile push

Retry for several hours before discarding message

HTTP Endpoints – configurable retry policy

Dead Letter Queue

For undelivered messages, you can configure a SQS Queue as DLQ

DLQ holds all undelivered messages

DLQ needs to be specified for each subscriber (redrive policy)

Delivery Status

Configure SNS topic to log delivery status

- HTTP Endpoints
- Lambda
- SQS

Logs are delivered to CloudWatch Logs

SNS Event Source

Several AWS services natively integrate with SNS Topics

- CloudWatch, EC2, RDS, S3, and more

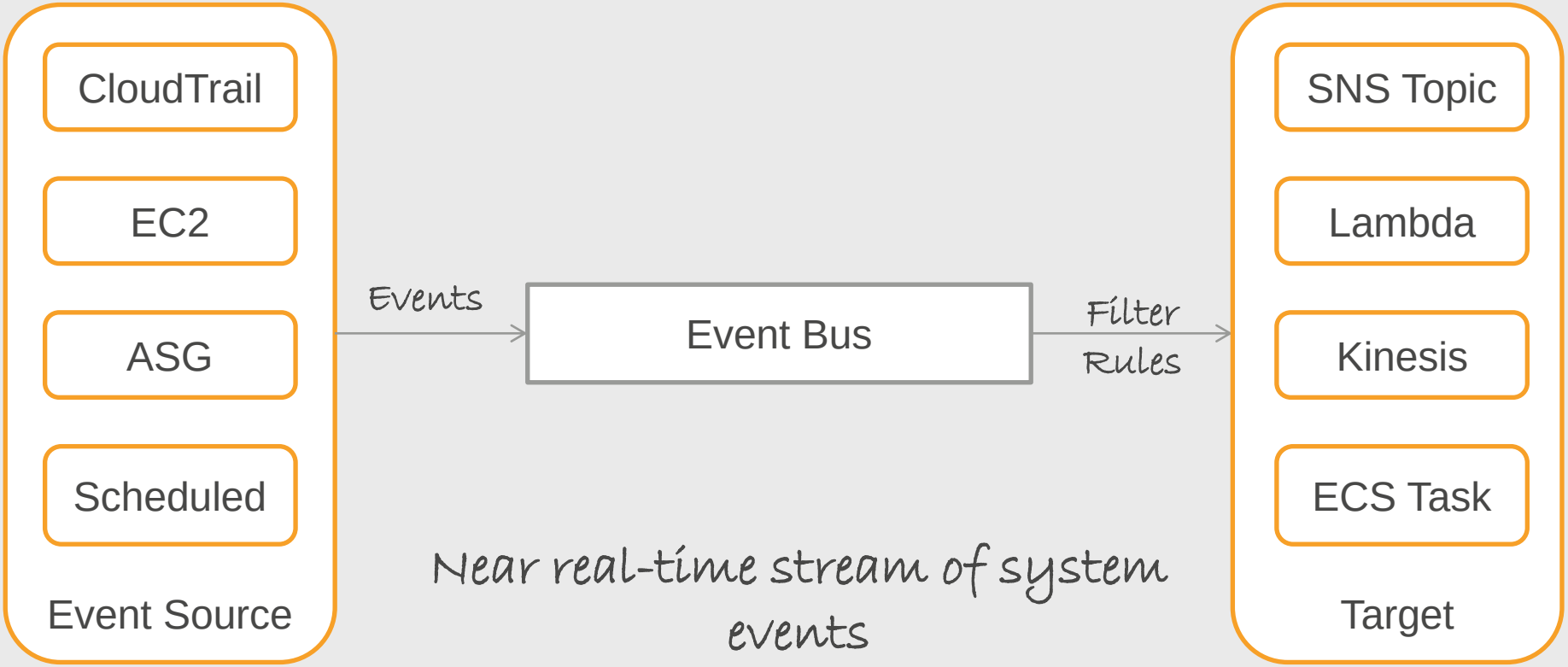
EventBridge

Events

CloudWatch Events

EventBridge

EventBridge



Example of Events

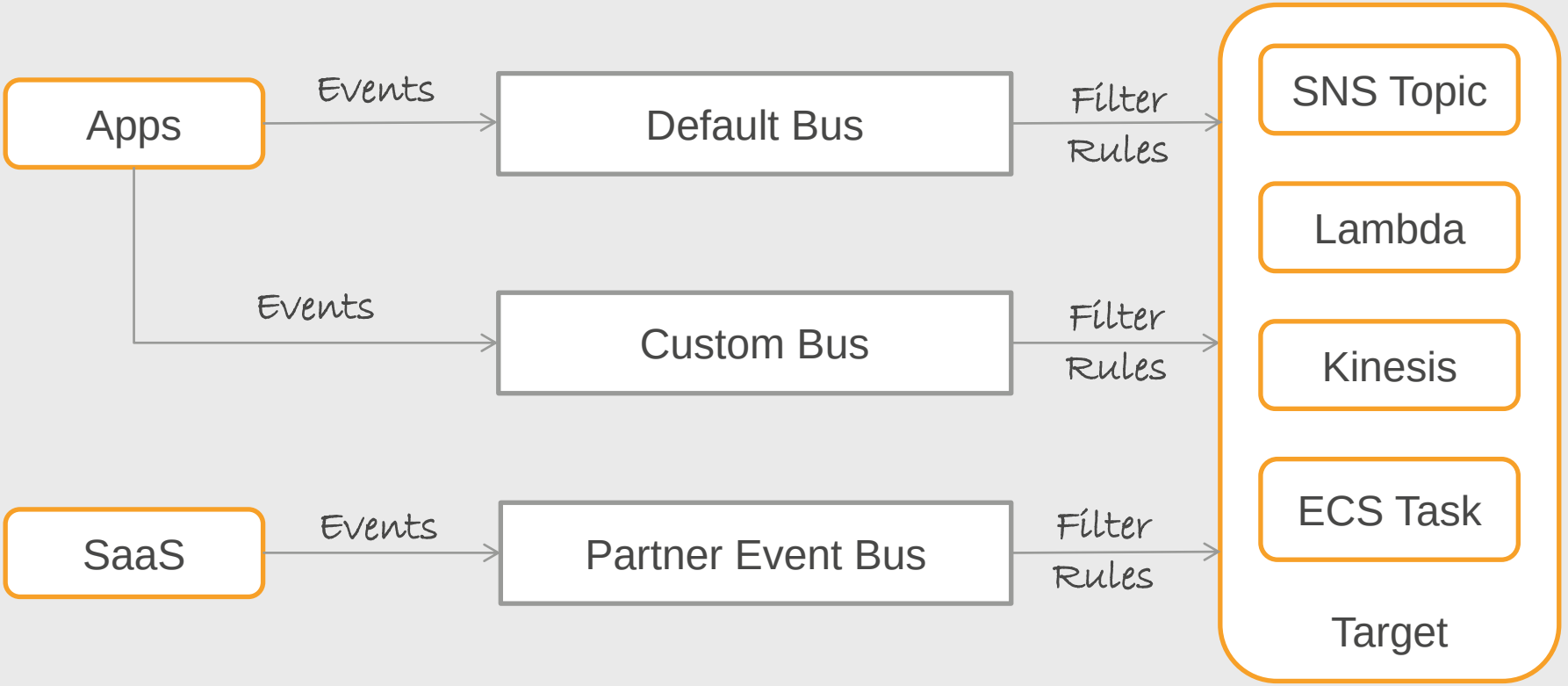
EC2 instance state changes (Pending -> Running)

Schedule EBS Volume Snapshot

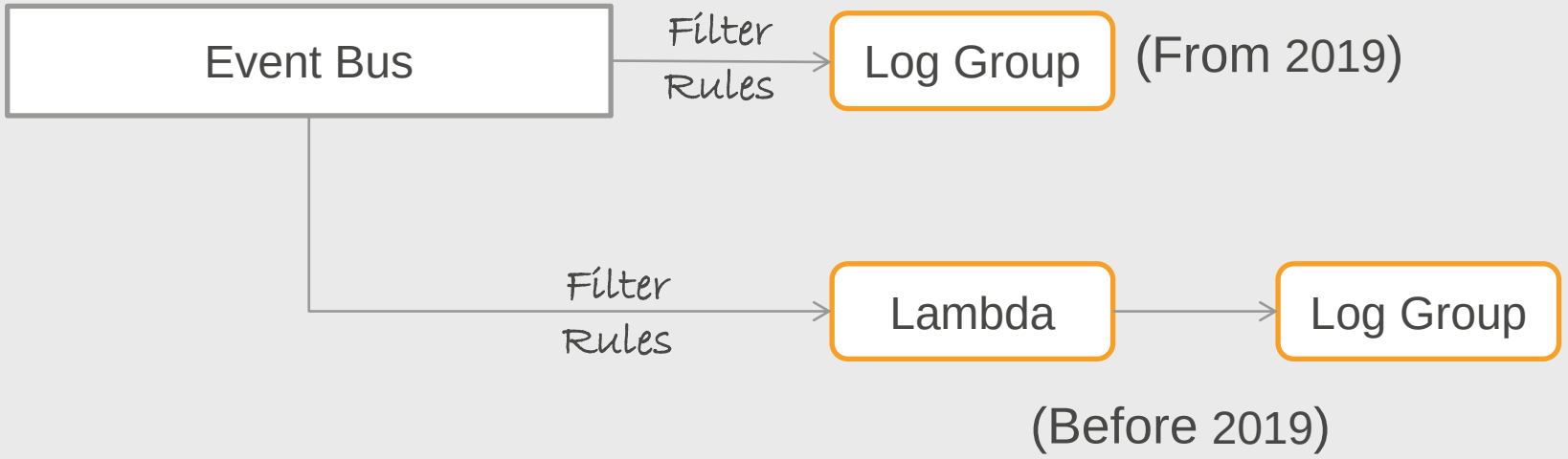
ASG event when a new instance is added or removed

CloudTrail event for every API action

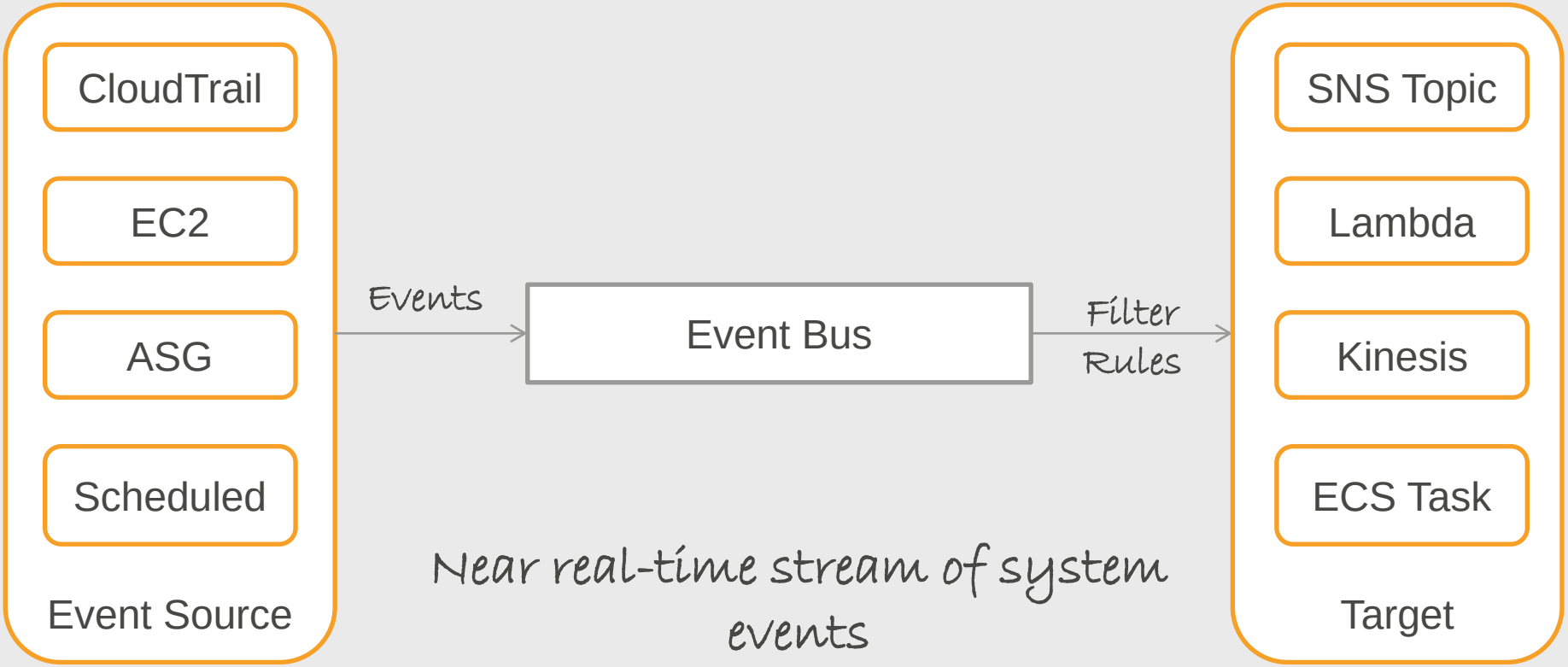
EventBridge – Custom Bus, SaaS Partner Bus



Logging Events to CloudWatch Log



EventBridge



EventBridge (CloudWatch Events)

Provides a near real-time stream of system events

Respond to resource changes as they occur

Asynchronously connect application components

Scheduled events

Reliable event delivery (24 hours retry)

Comparison

EventBridge

- Recommended when you need to process events from AWS services (90+)
- SaaS partner integration
- Well defined JSON structure for events
- Schema registry
- Generate code to process events

SNS

- High throughput, low-latency messages
- AWS Service integration (30+)
- Fanout to millions of subscribers
- Messages are unstructured and supports any format

Both services support filtering to deliver messages to relevant subscribers

Stream and Batch Processing

Streaming Data

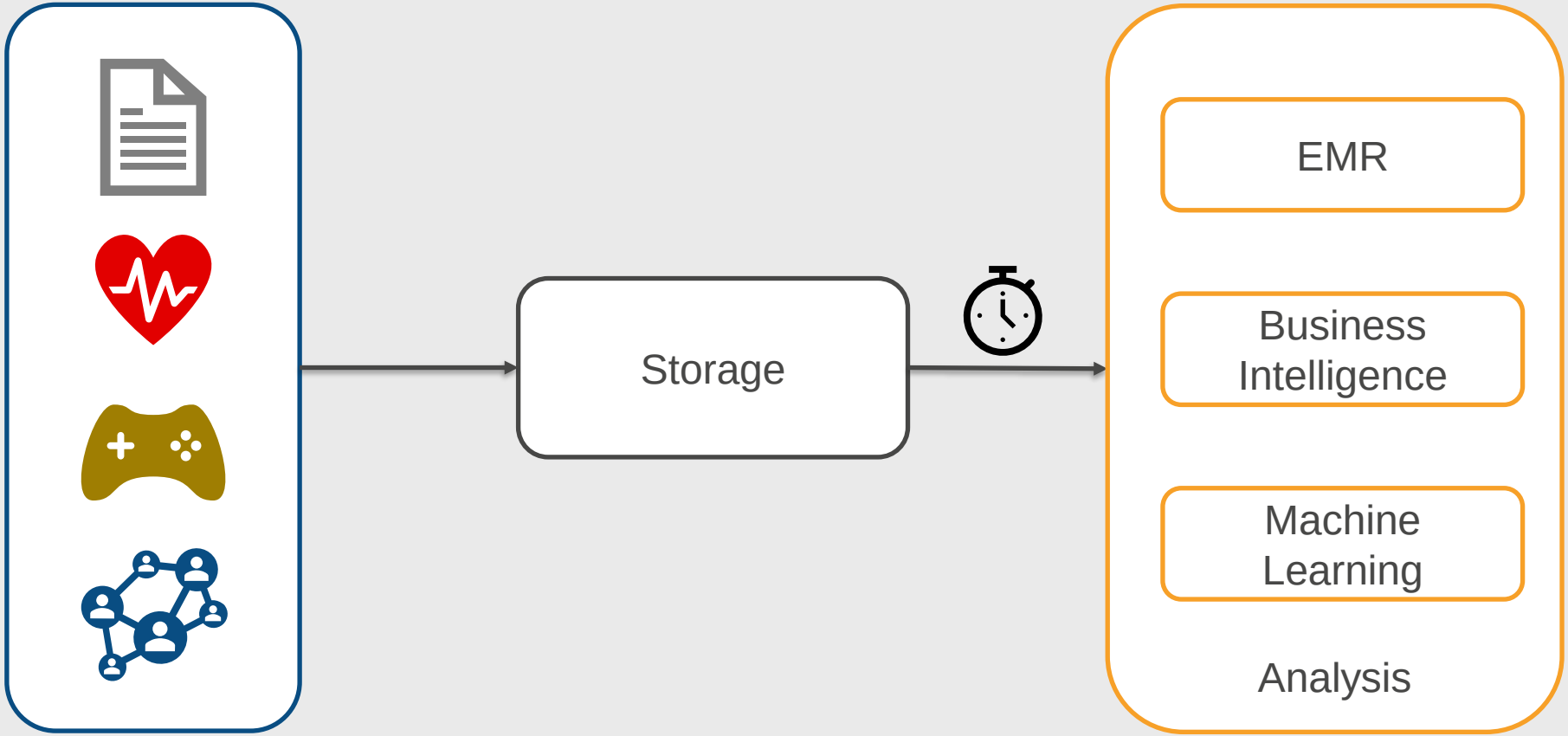
Generated Continuously

Thousands of sources

Small Payloads




Batch Processing

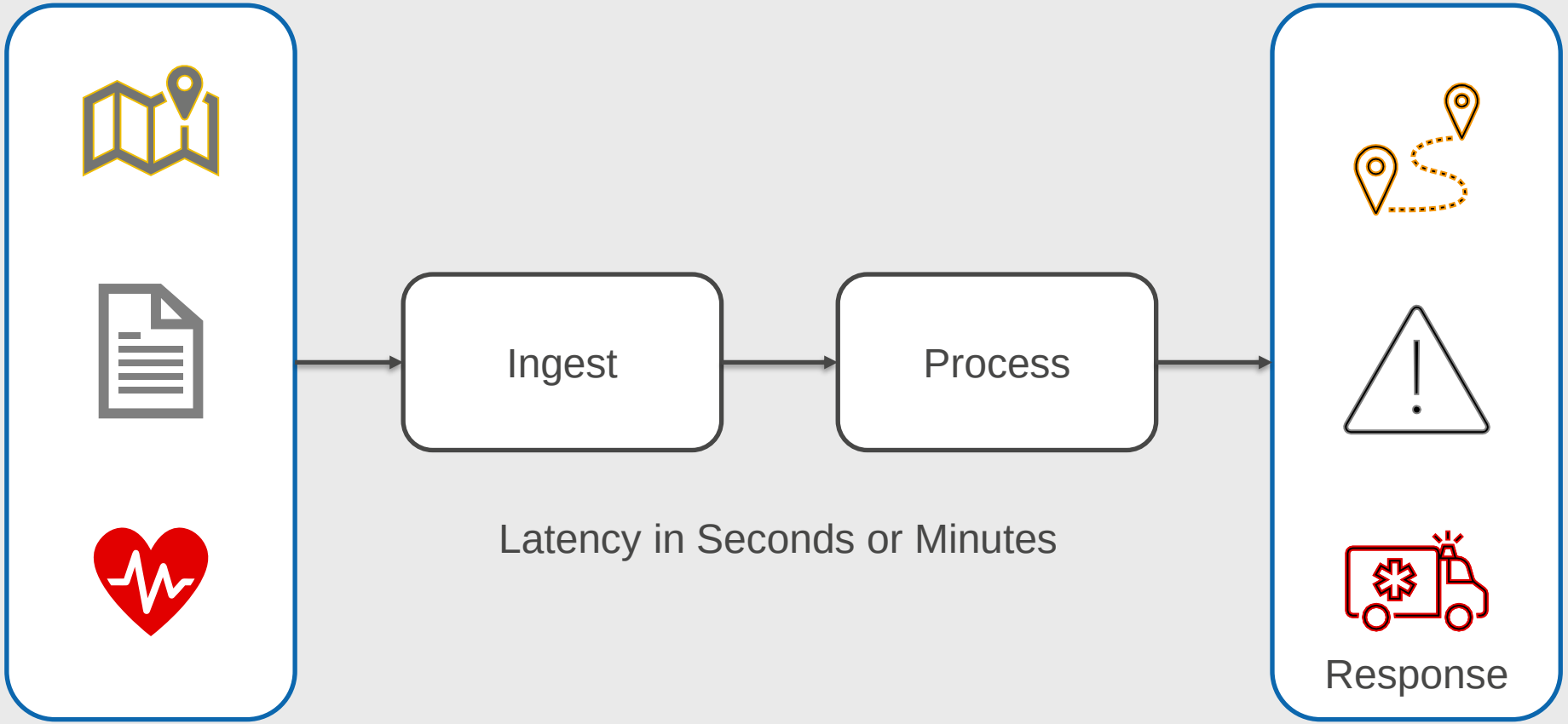


Batch Processing Use Cases

Utility bill generation

Daily, monthly manufacturing reports

Stream Processing



Amazon Kinesis

Collect, Process, Analyze Streaming Data

Amazon Kinesis

“Amazon Kinesis enables you to ingest, buffer and process streaming data in real-time”

“you can derive insights in seconds or minutes.”

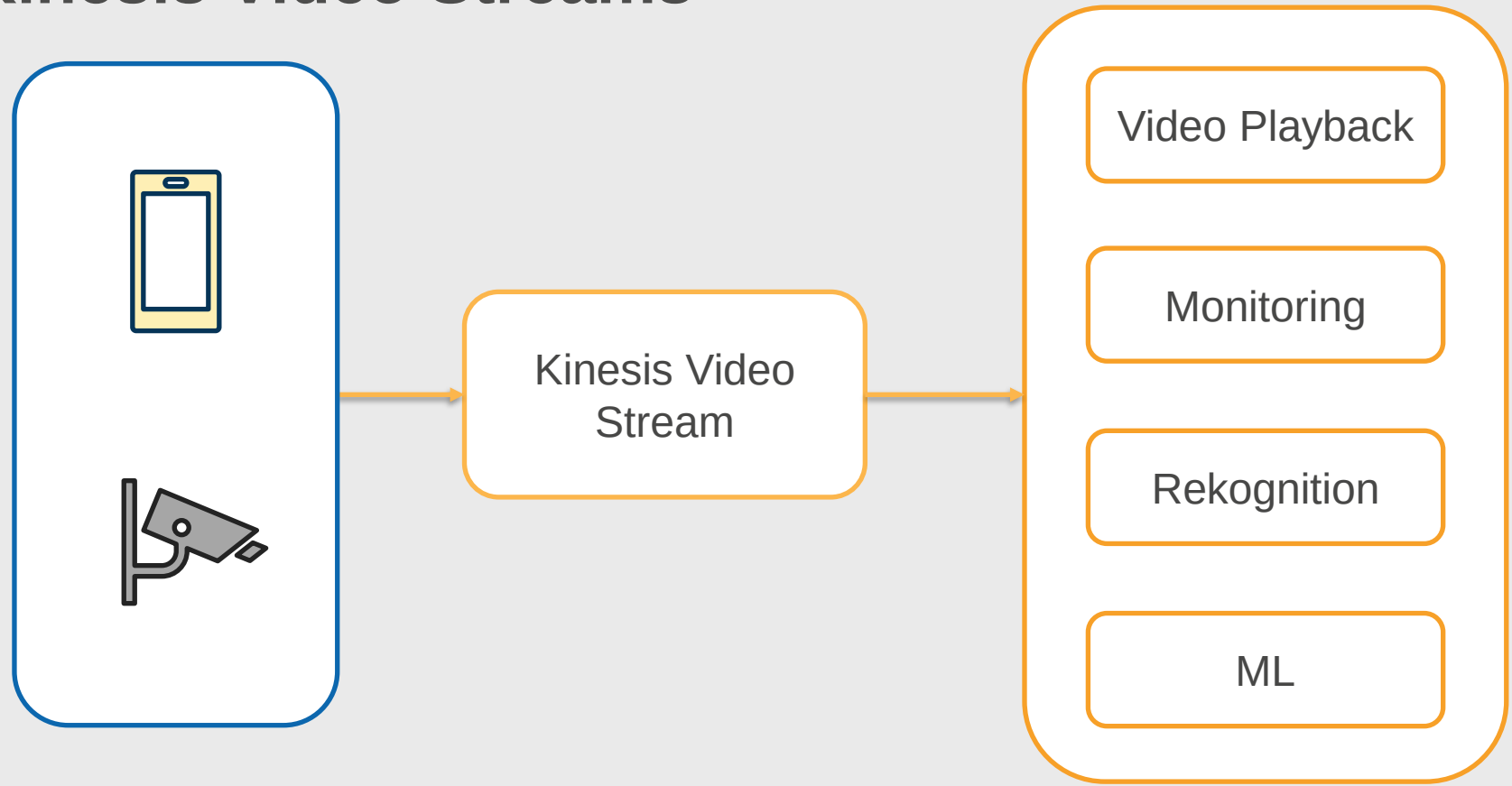
“Handle any amount of streaming data from hundreds of thousands of sources with very low latencies”

Reference: Amazon Kinesis, https://aws.amazon.com/kinesis/

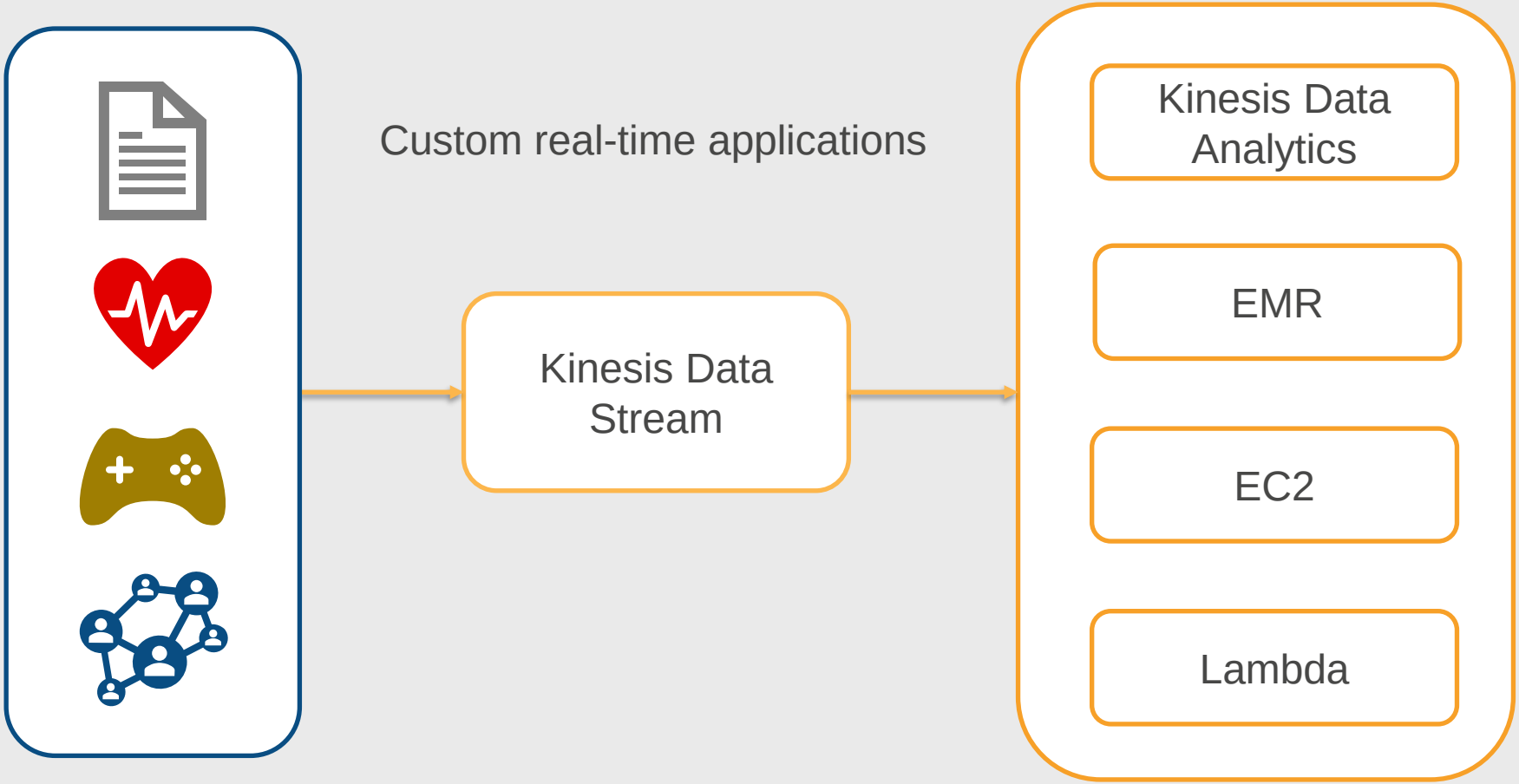
Kinesis Family

Service	Purpose	Use
Video Streams	Capture and Analyze Video Stream	Security Monitoring, Video Playback, Face detection
Data Streams	Capture and Analyze Data Stream	Custom real-time application
Firehose	Capture and Deliver Data Stream to AWS Data Stores	Use Existing BI tools for Streaming Data: S3, Redshift, ElasticSearch, Splunk
Data Analytics	Analyze Data Stream with SQL and Java	Real-time analytics, Anomaly detection

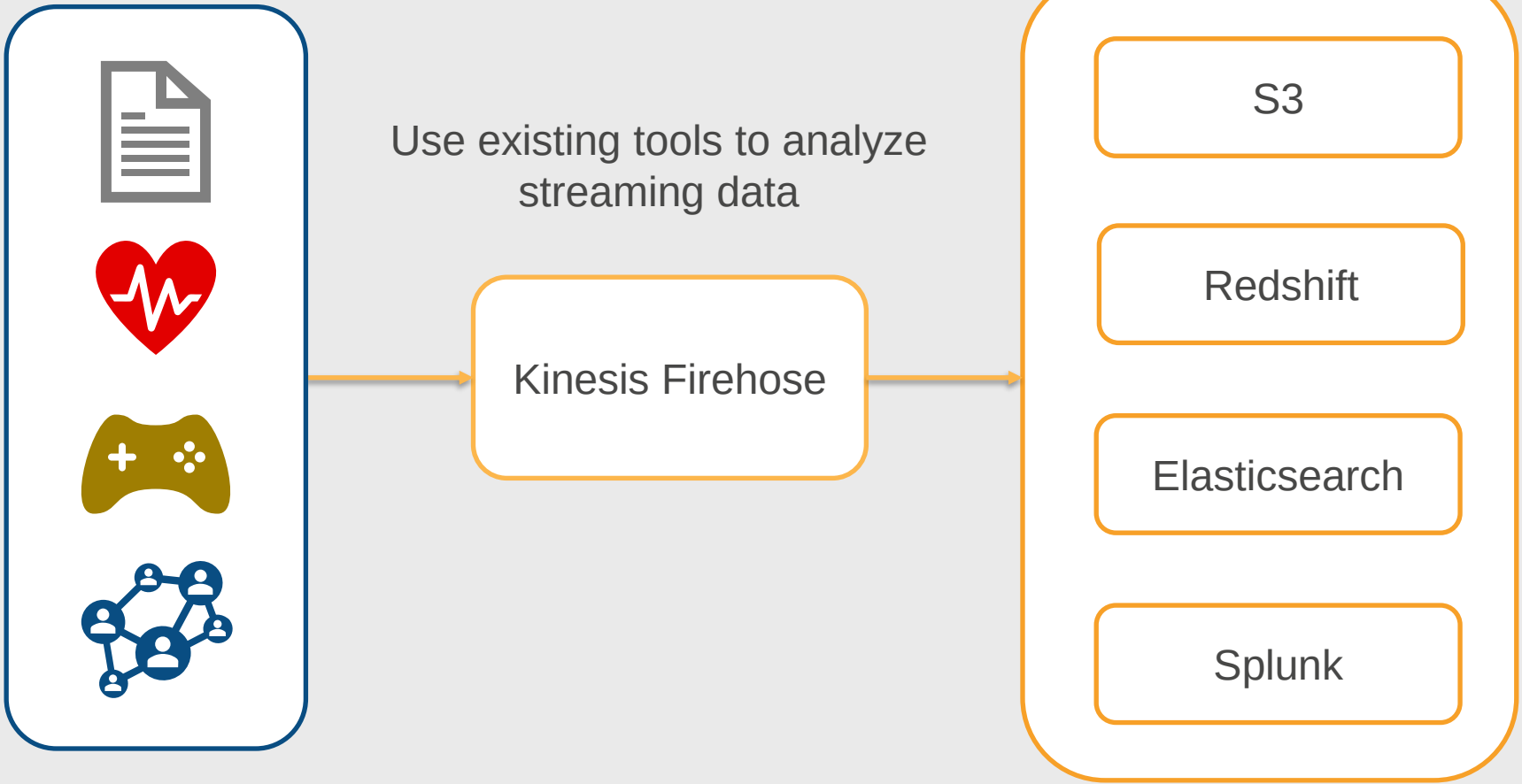
Kinesis Video Streams



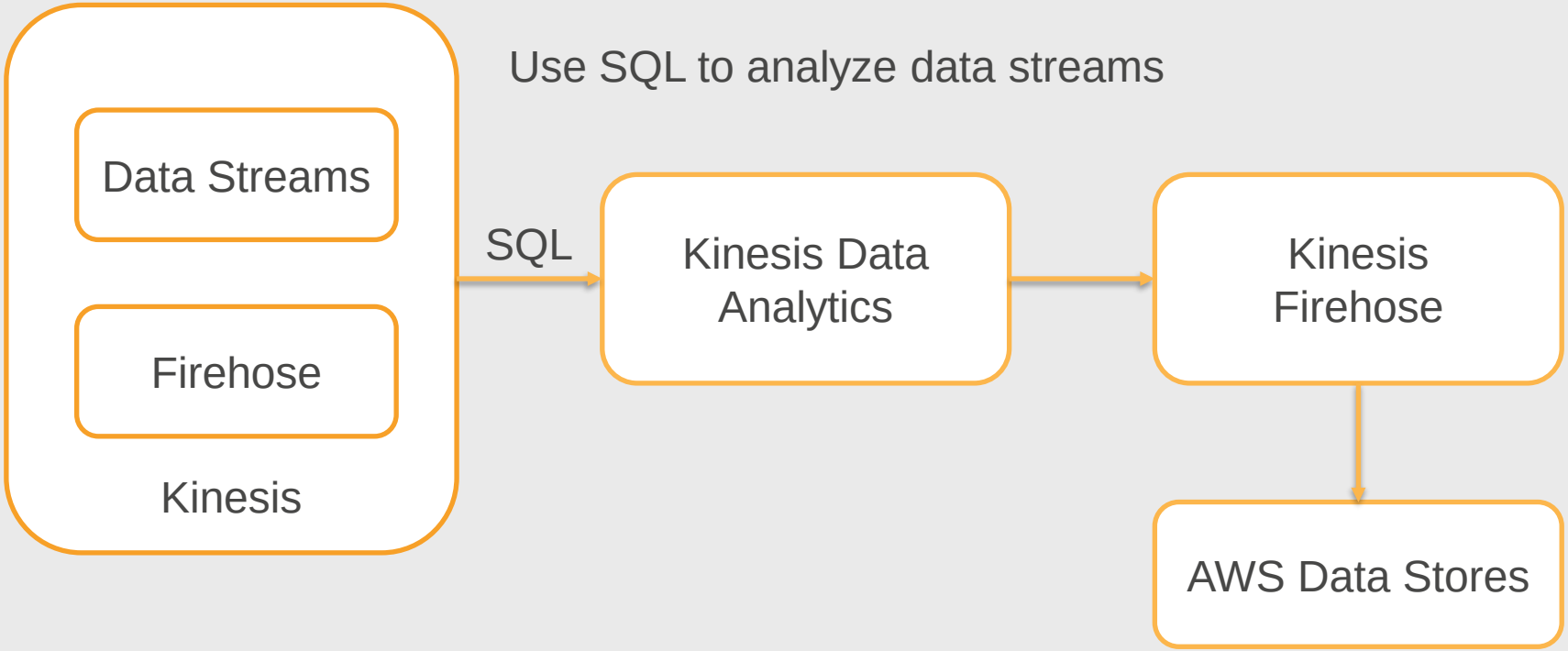
Kinesis Data Streams



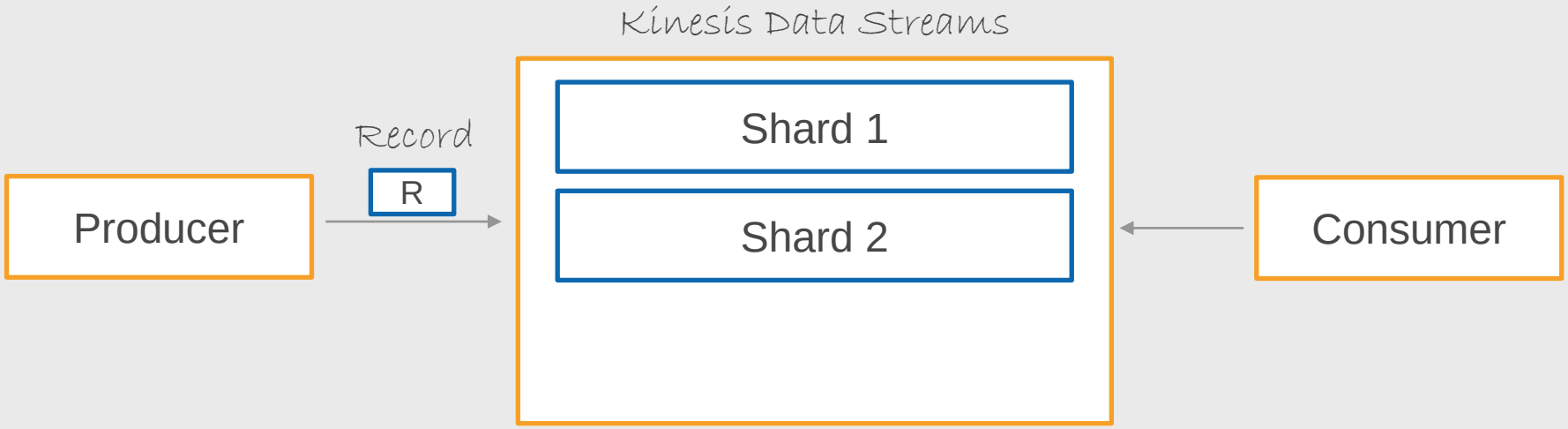
Kinesis Data Firehose



Kinesis Data Analytics



Kinesis Data stream

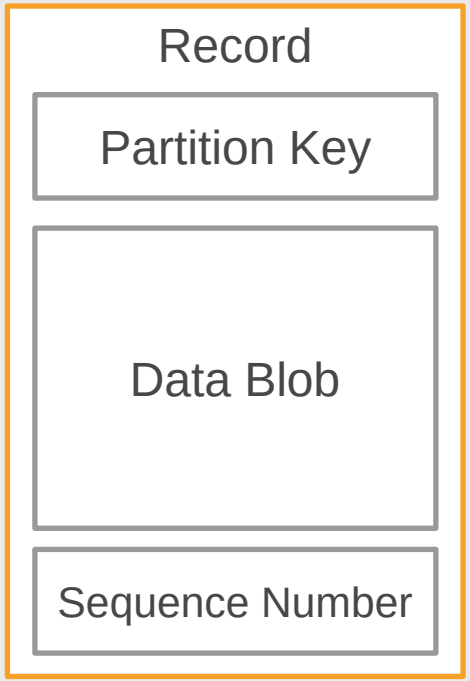


Records are organized in partitions (shards)

Throughput per shard 1 MB/sec (for ingestion) and 2 MB/sec (for reading)

Increase throughput by adding additional shards

Data Streams Record



Publisher specifies a value for the partition key

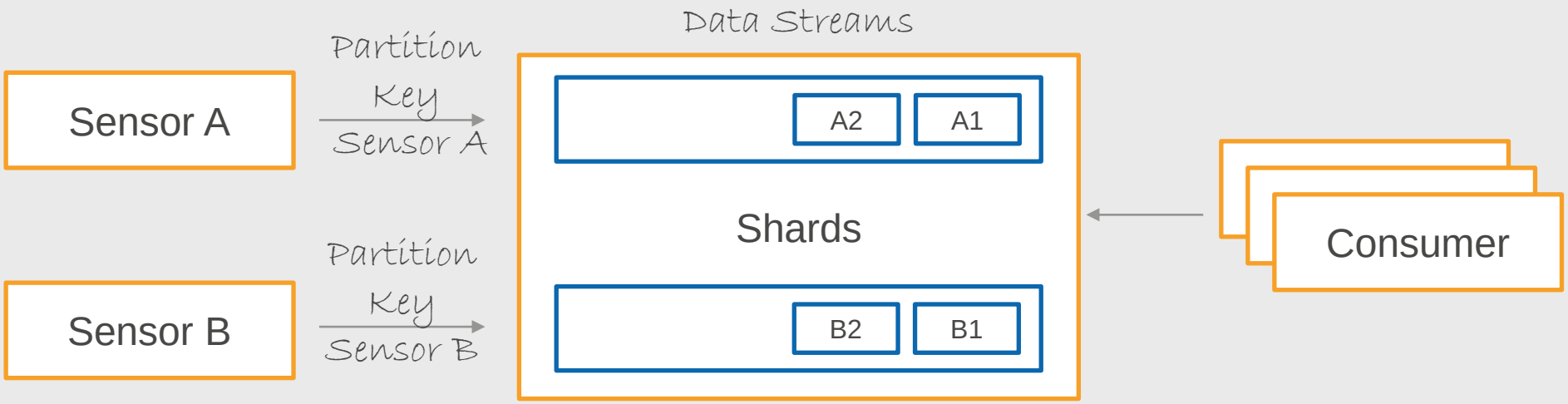
Determines which shard to use for the record

1 MB of data

Base64 encoded

Automatically added by data streams

Kinesis Data stream



Records with same partition key are added to same shard
All records for a sensor are available in a single shard
Records are ordered by arrival time

Kinesis Data Streams Retention

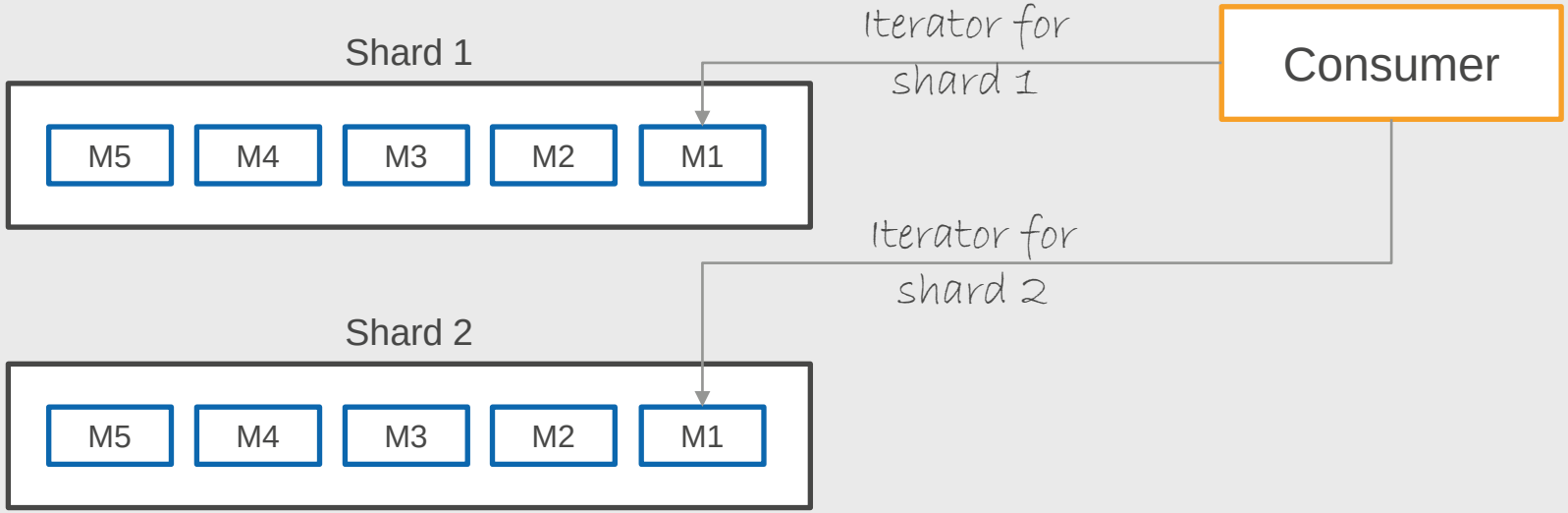
Default retention period is 1 day (24 hours)

Extended Retention for 7 days (useful to reprocess data)

Long-term retention for up to 1 year

- Available from end of 2020

Reading from Kinesis Data stream

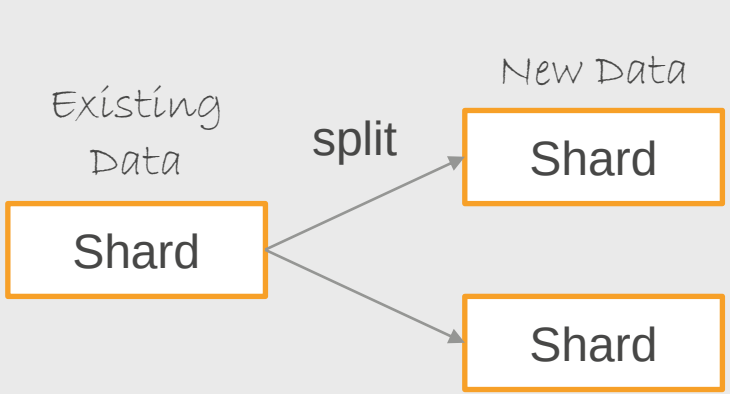


1. List all available shards
2. Get a shard iterator for each shard
3. Read records in each shard
4. Shard Iterator specifies the position from which to read records

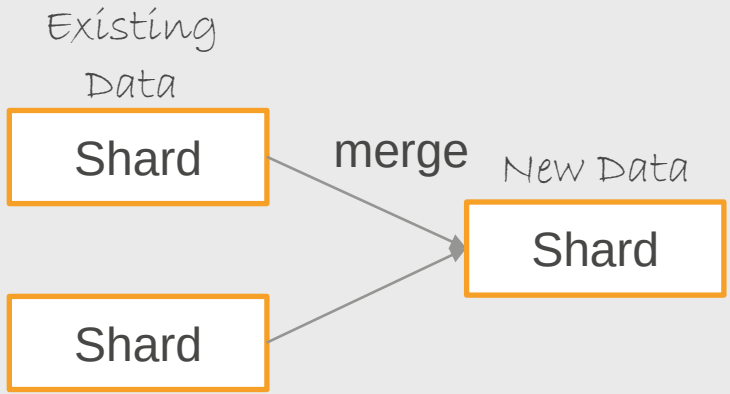
Shard Iterator Types

1. At Sequence Number – read records from a specific sequence number
2. After Sequence Number – read records after a sequence number
3. At Timestamp – read records from the position indicated by timestamp
4. Trim Horizon – read from oldest record available in the shard
5. Latest – read from the most recent record

Resharding



Increase Throughput



Reduce Throughput and Cost

In the past, resharding was confusing and tedious – you need to keep track partition key range used by each shard and ensure no gaps in partition key range

AWS now supports update-shard-count API – automatically adjusts the number of shards

Kinesis Library

Kinesis Producer Library (KPL)

For producers to publish records to data streams

Kinesis Client Library (KCL)

For consumers to read records from data streams

Java, Python, Node.js, .NET, Ruby

Uses DynamoDB to track records processed by each application

Supports multiple worker processes

Automatically handles resharding

Kinesis Firehose

Fully Managed

No need to worry about shards and streams

Continuously store streaming data to S3, Redshift, Elasticsearch, Splunk

Kinesis Data Streams

One or more shards

Records ordered by arrival time in a shard

Retention – 1 day, 7 days, 365 days

Resharding to adjust number of shards

Kinesis Client Library to write consumer apps

Streams

Ingest vast amount of data

Maintain data in a time-ordered sequence

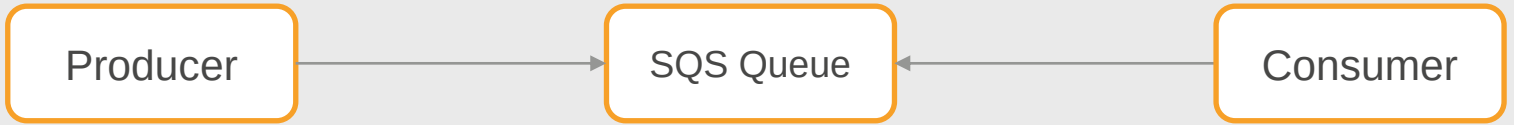
Two products

- Amazon Kinesis family
- Managed Apache Kafka

Permission Management

Producers
Consumers
SQS, SNS, EventBridge, Kinesis

SQS Queue – Three ways to manage access



Identity-based Policy
- Grant permission to send-message to Queue
- IAM Role, Users or Groups

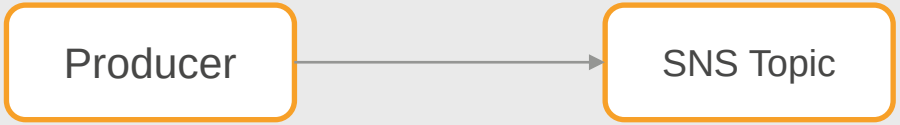
Queue Resource-based Policy
- Access Policy
- Grant permission to AWS service or identity
- send-message and receive-message

Identity-based Policy
- Grant permission to receive-message from Queue
- IAM Role, Users or Groups

Same-account access – any of these options would work

Cross-account access
- Grant access to other account using the queue resource-based policy
- Create IAM Role in the queue owner account and grant assume role permission to the other account

SNS Topic - Producer



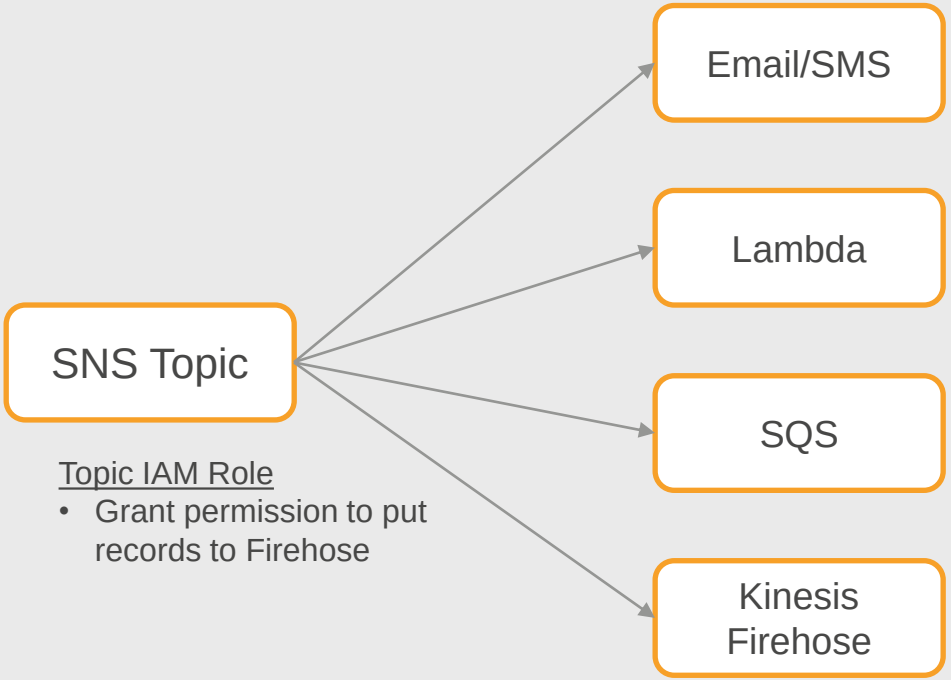
Identity-based Policy
- Grant permission to publish-message to Topic
- IAM Role, Users or Groups

Topic Resource-based Policy
- Grant permission to AWS service or identity to publish-message

Same-account access – any of these options would work

Cross-account access
- Grant access to other account using Resource-based policy
- Create IAM Role in the resource owner account and grant assume role permission to the other account

SNS Topic - Subscriber

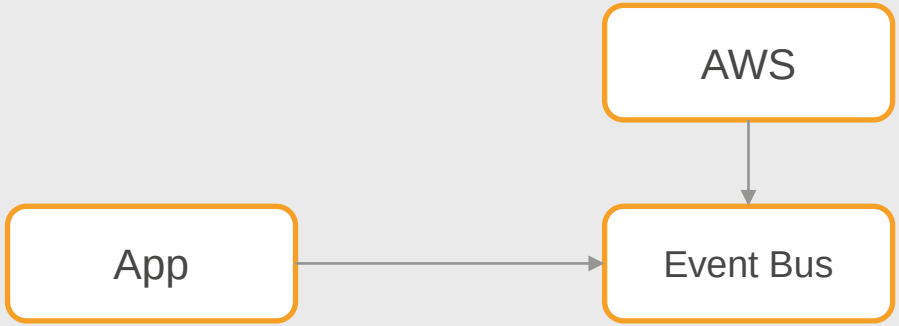


Lambda Resource-based policy
- Grant permission to Topic to invoke function

SQS Resource-based policy
- Grant permission to Topic to send-message

Topic IAM Role
- Grant permission to put records to Firehose

EventBridge - Producer



- AWS Services automatically publish events to Event Bus
- No permission needed

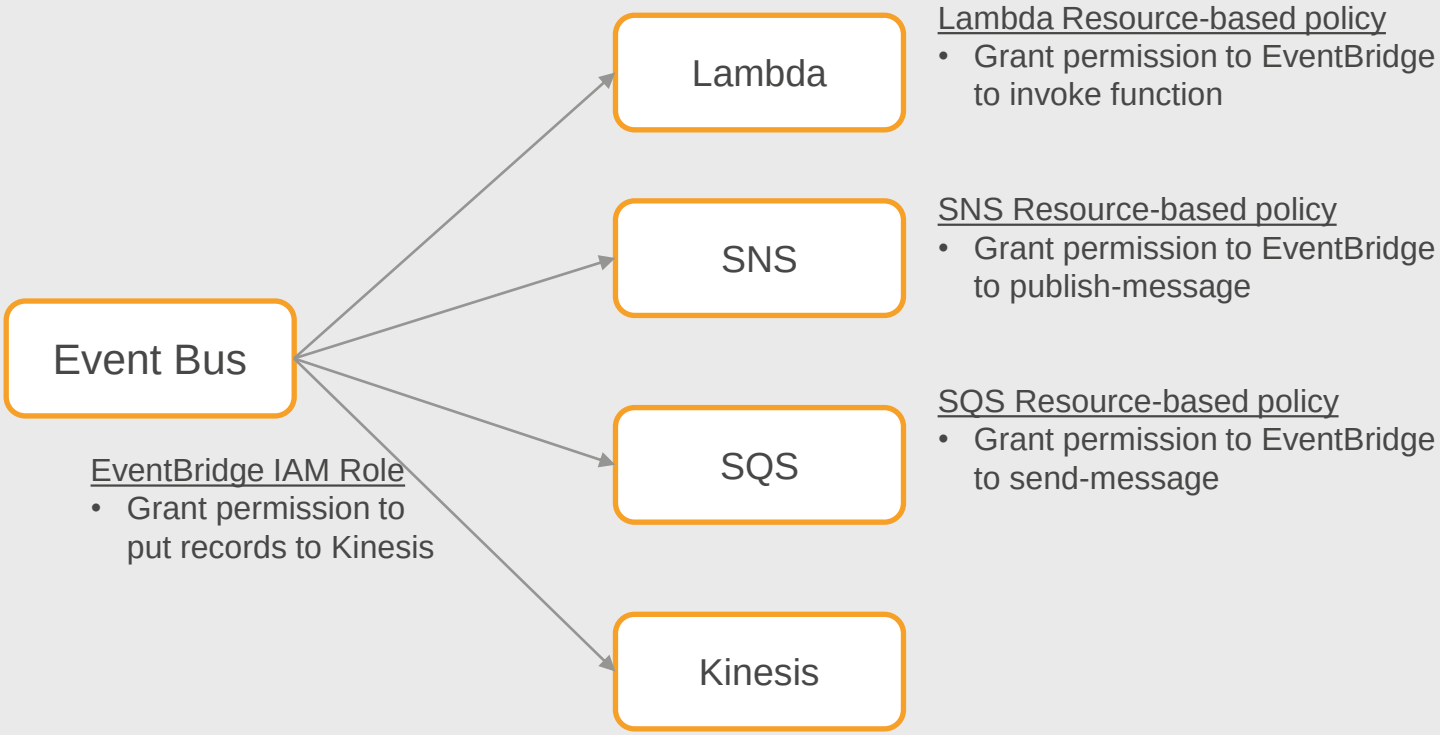
Identity-based Policy
- Grant permission to publish events
- IAM Role, Users or Groups

Event Bus Resource-based Policy
- Grant permission to custom application

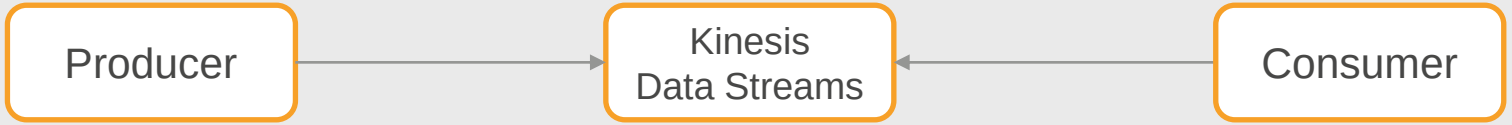
Same-account access – any of these options would work

Cross-account access
- Grant access to other account using Resource-based policy
- Create IAM Role in the resource owner account and grant assume role permission to the other account

EventBridge - Target



Kinesis Data Streams



Identity-based Policy
- Grant permission to put-record
- IAM Role, Users or Groups

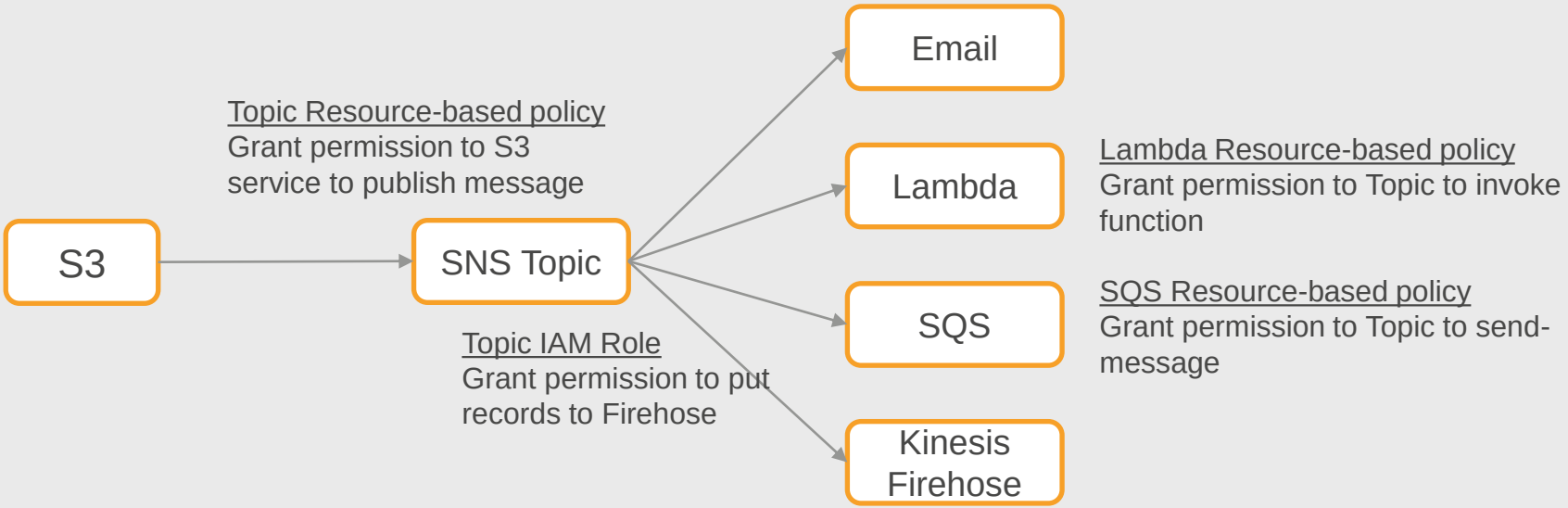
Identity-based Policy
- Grant permission to get-record
- IAM Role, Users or Groups

Kinesis Firehose

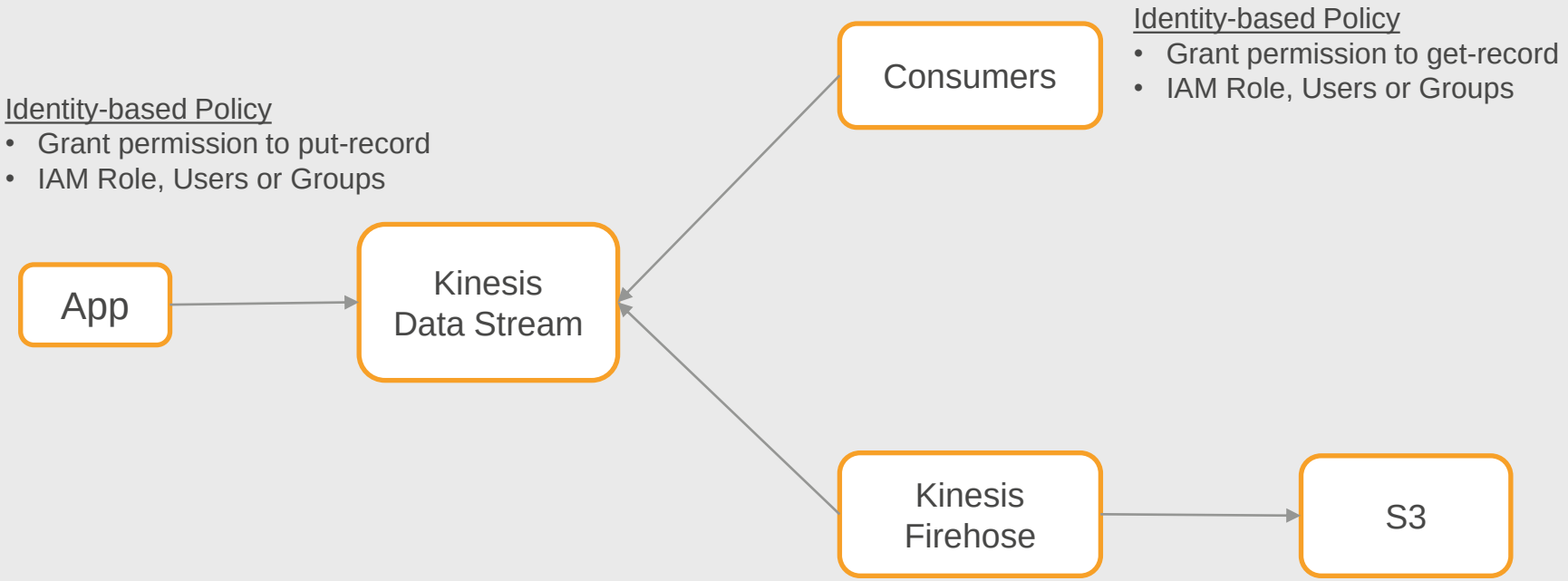
Identity-based Policy
- Grant permission to put-record
- IAM Role, Users or Groups

Firehose IAM Role
- Grant permission to store records to destination

Example 1 – S3 event notification



Example 2 – Streaming Data



Messaging Services Summary

	SQS	SNS, Event Bridge	Kinesis Data Streams
Poll or Push	Polling	Push notification	Polling
Message Processing	Message is processed once	Message is broadcast to all subscribed consumers	Many consumers can read ALL messages (like a database)
Retention	Unprocessed messages can be kept for up to 14 days	Immediate Delivery with retry	Default 1 day. Extended retention up to 1 year
Processing Time	Visibility Timeout dictates how long a consumer can take to process a message. Max 12 hours	Consumer can take as long as needed (each consumer gets a copy of a message)	Consumers must process messages by retention duration
Strength	Buffer messages	Broadcast messages for time sensitive processing	Time ordered messages for real-time analytics

Lab – Standard Queue

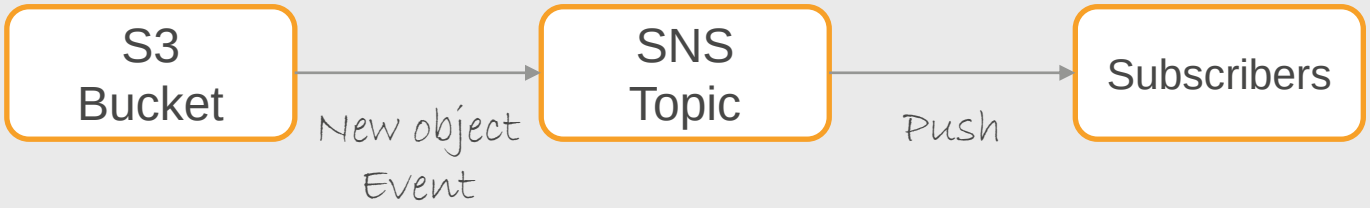
- How to send and receive messages using CLI
- Out-of-order delivery
- Visibility Timeout
- Dead Letter Queue
- Long Polling

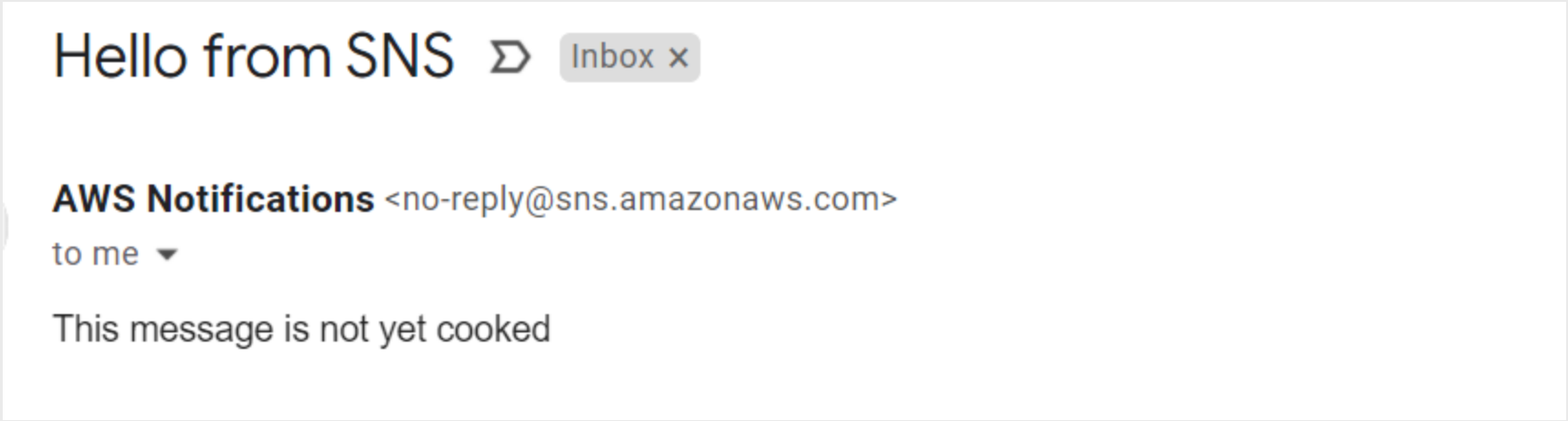
Lab – FIFO Queue

- FIFO ordering of messages
- Handling of duplicate messages
- Grouping related messages

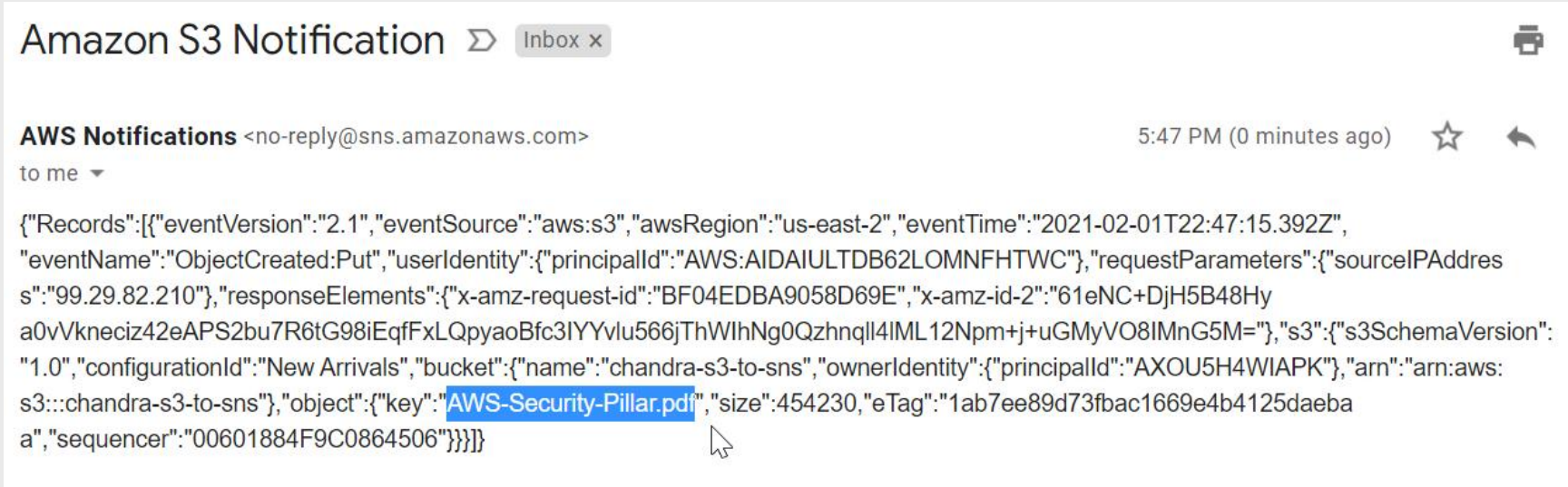
Lab – SNS Topic

- Fanout message to multiple subscribers
- S3 bucket to notify SNS Topic of new object events





S3 Event Notification



Chandra Lingam

57,000+ Students

aws certified Solutions Architect Associate

aws certified Developer Associate

aws certified SysOps Administrator Associate

aws certified Solutions Architect Professional

aws certified Machine Learning Specialty

For AWS self-paced video courses, visit:

https://www.cloudwavetraining.com/

Cloud Wave LLC