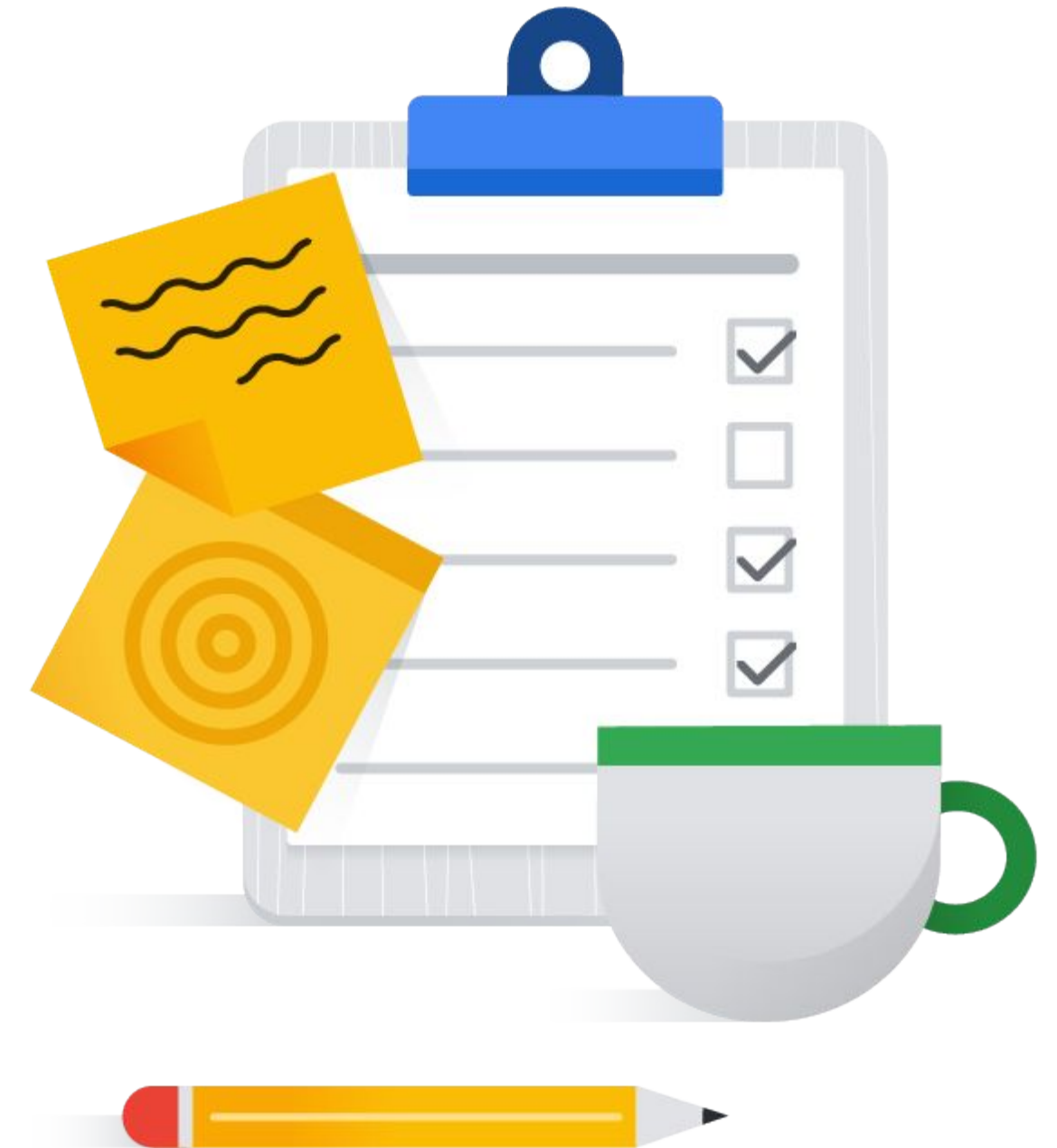




# Serverless Messaging with Pub/Sub

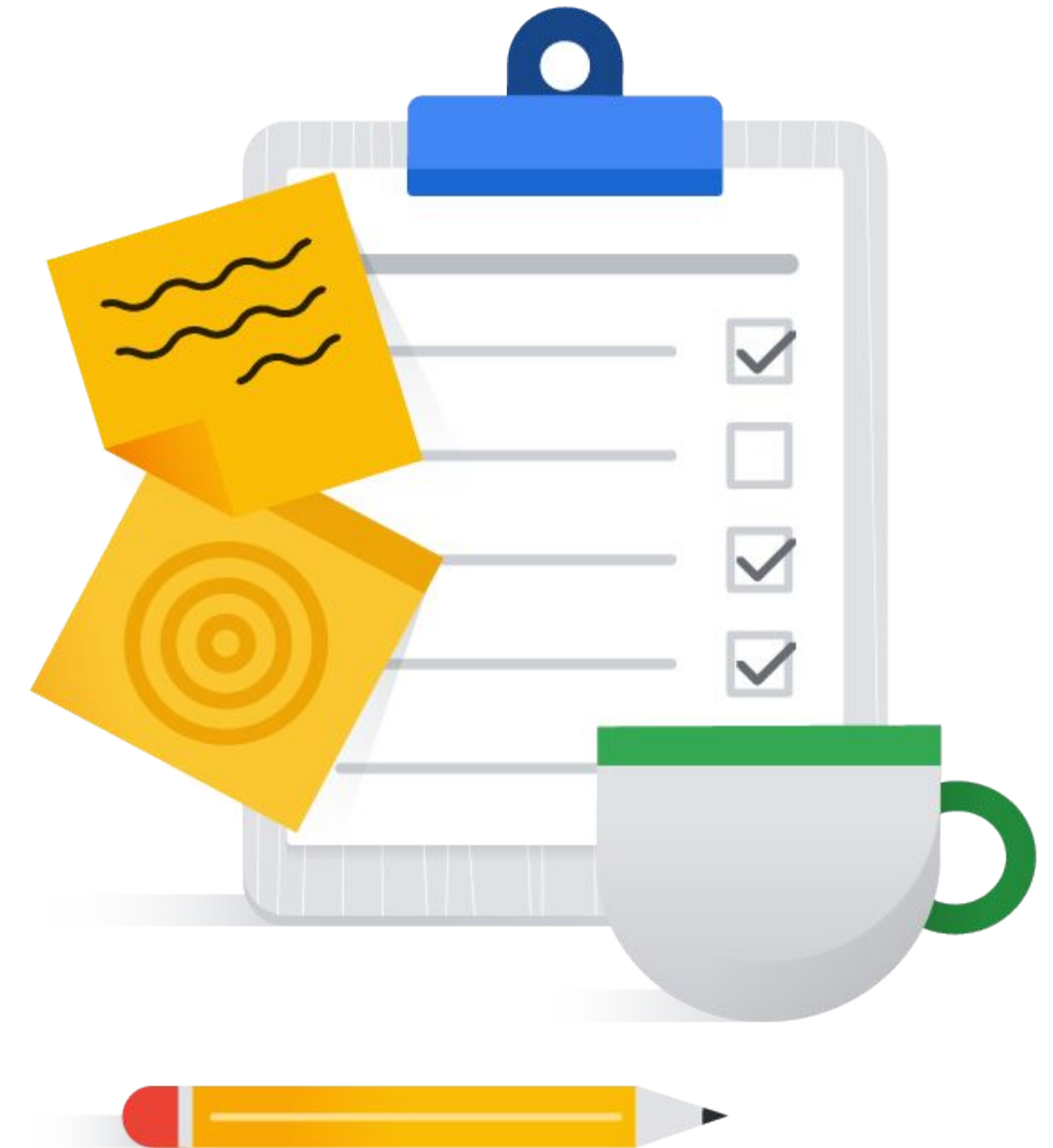
# Serverless Messaging with Pub/Sub

- |    |                              |
|----|------------------------------|
| 01 | Introduction to Pub/Sub      |
| 02 | Pub/Sub push versus pull     |
| 03 | Publishing with Pub/Sub code |

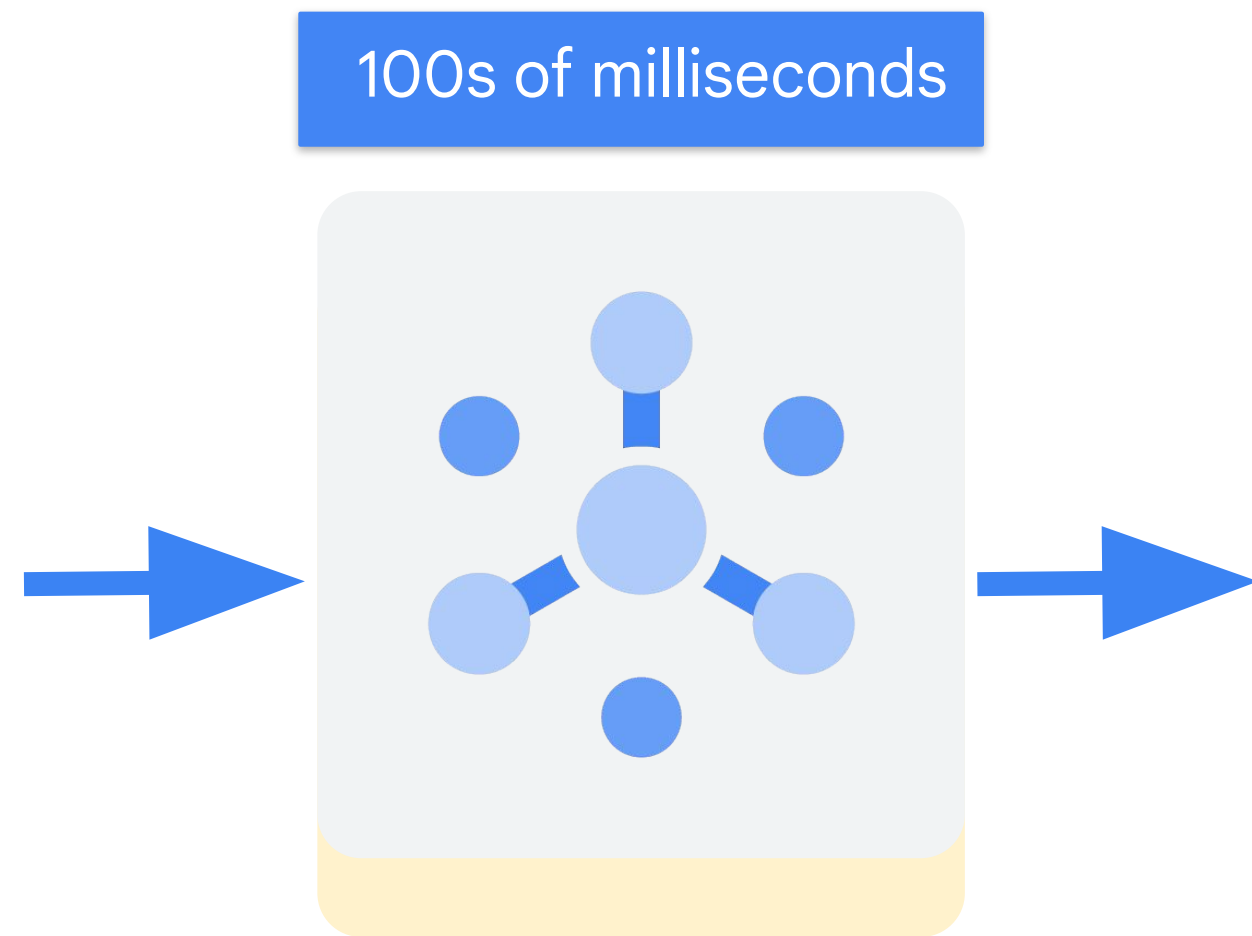


# Serverless Messaging with Pub/Sub

- |    |                              |
|----|------------------------------|
| 01 | Introduction to Pub/Sub      |
| 02 | Pub/Sub push versus pull     |
| 03 | Publishing with Pub/Sub code |



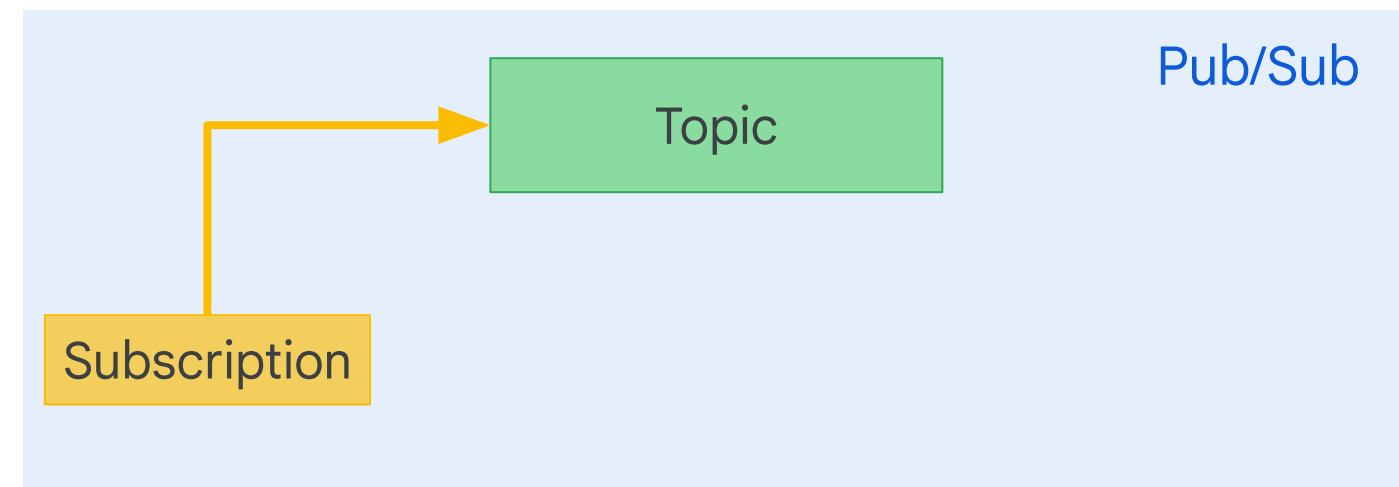
# Pub/Sub



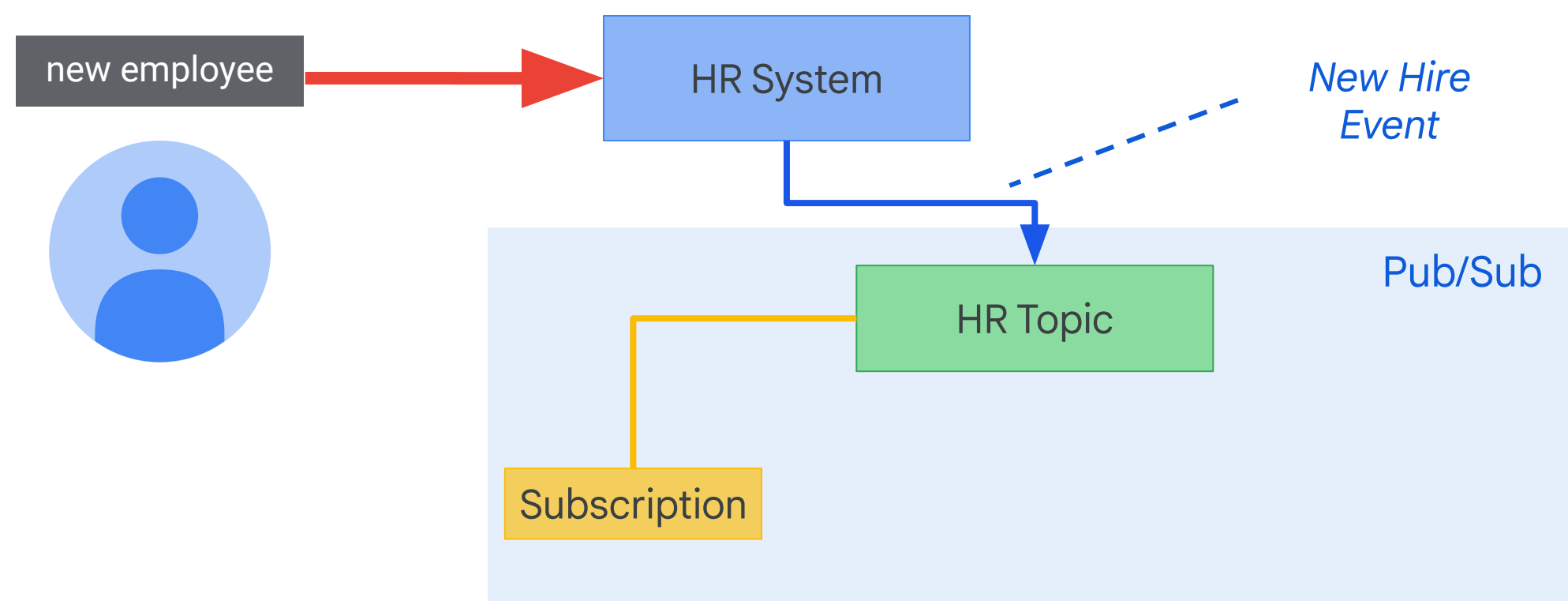
Qualities that Pub/Sub contribute to data engineering solutions:

- ✓ Availability
- ✓ Durability
- ✓ Scalability

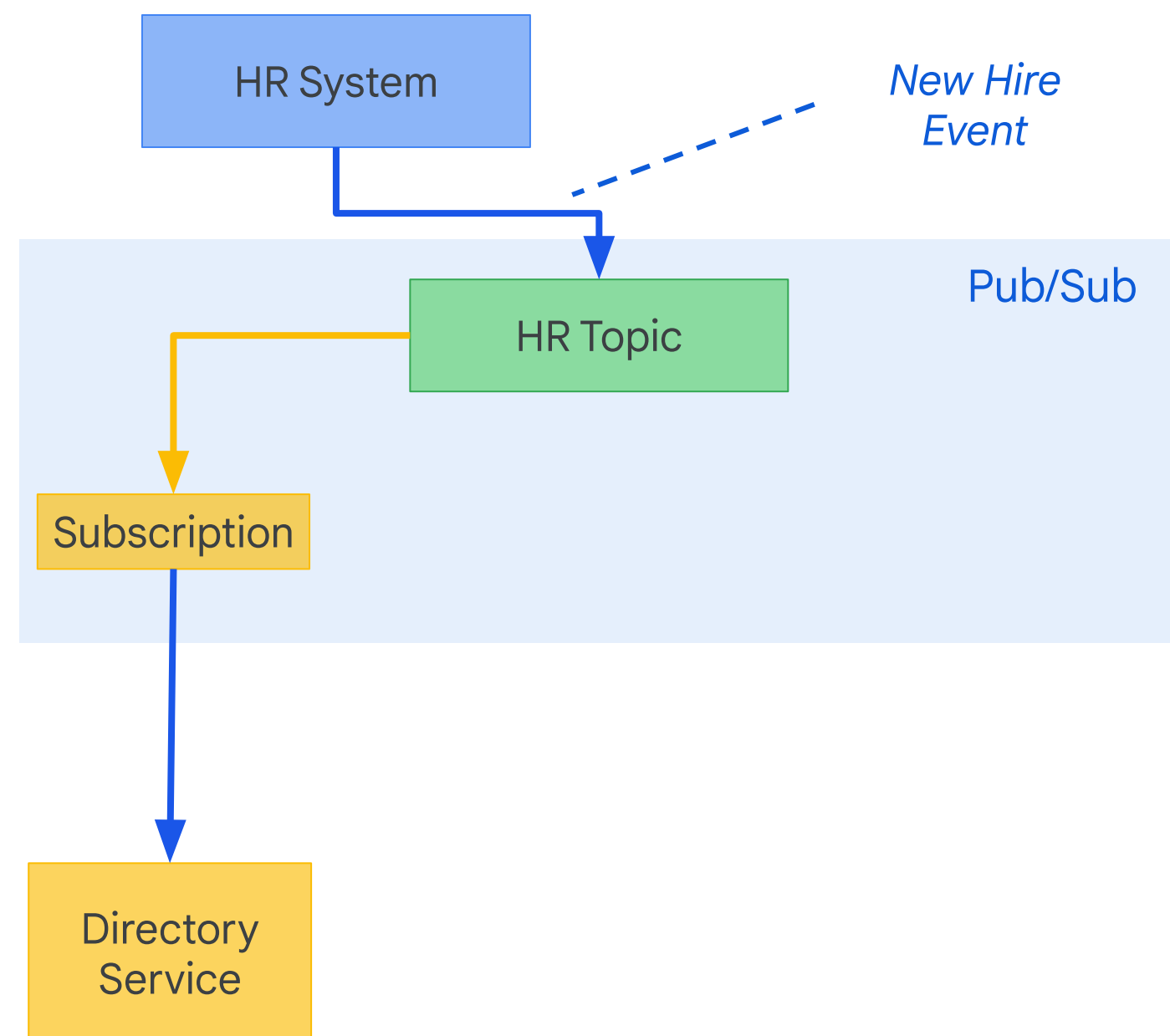
# Example of a Pub/Sub application



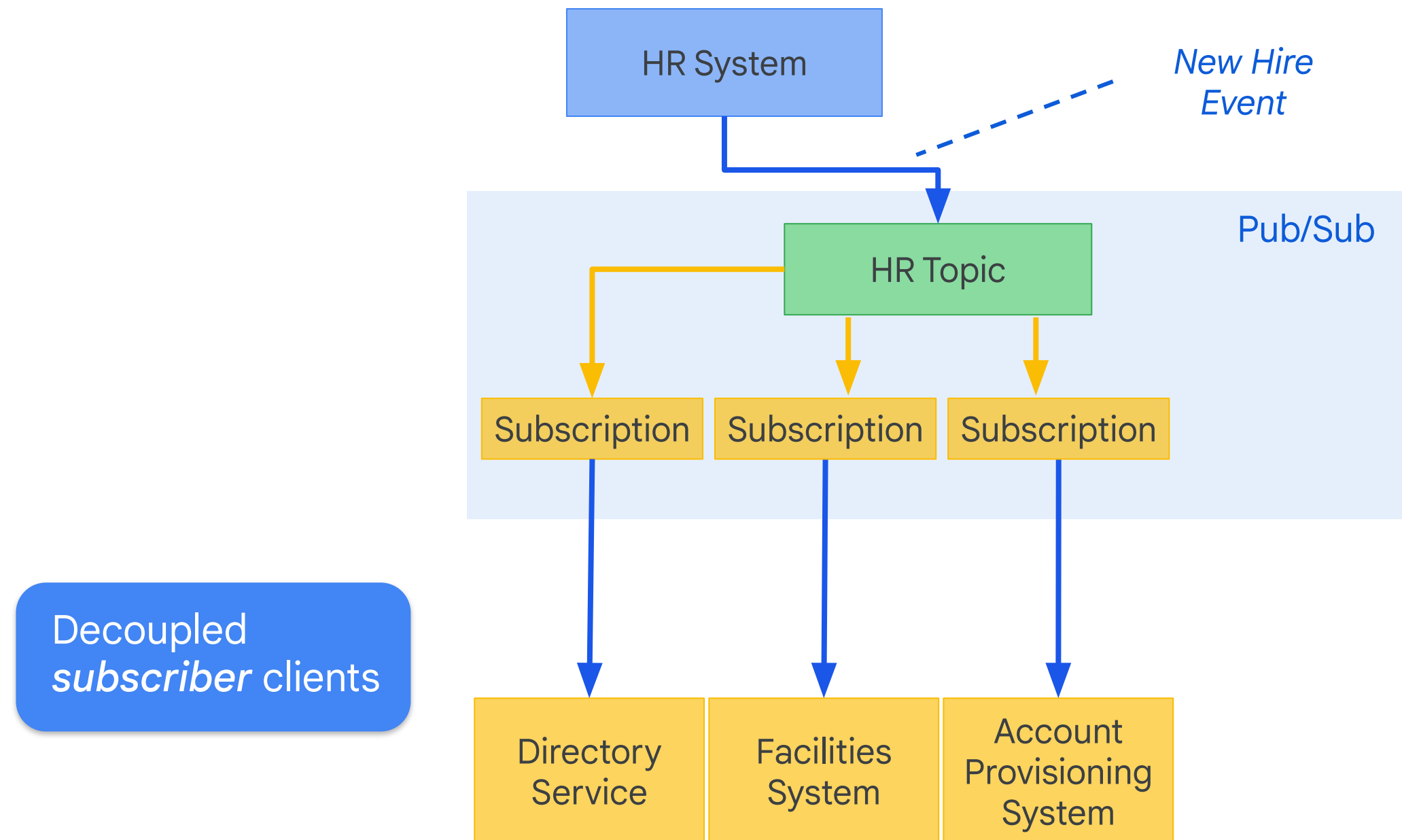
# A new employee arrives causing a new hire event



# The message is sent from Topic to Subscription

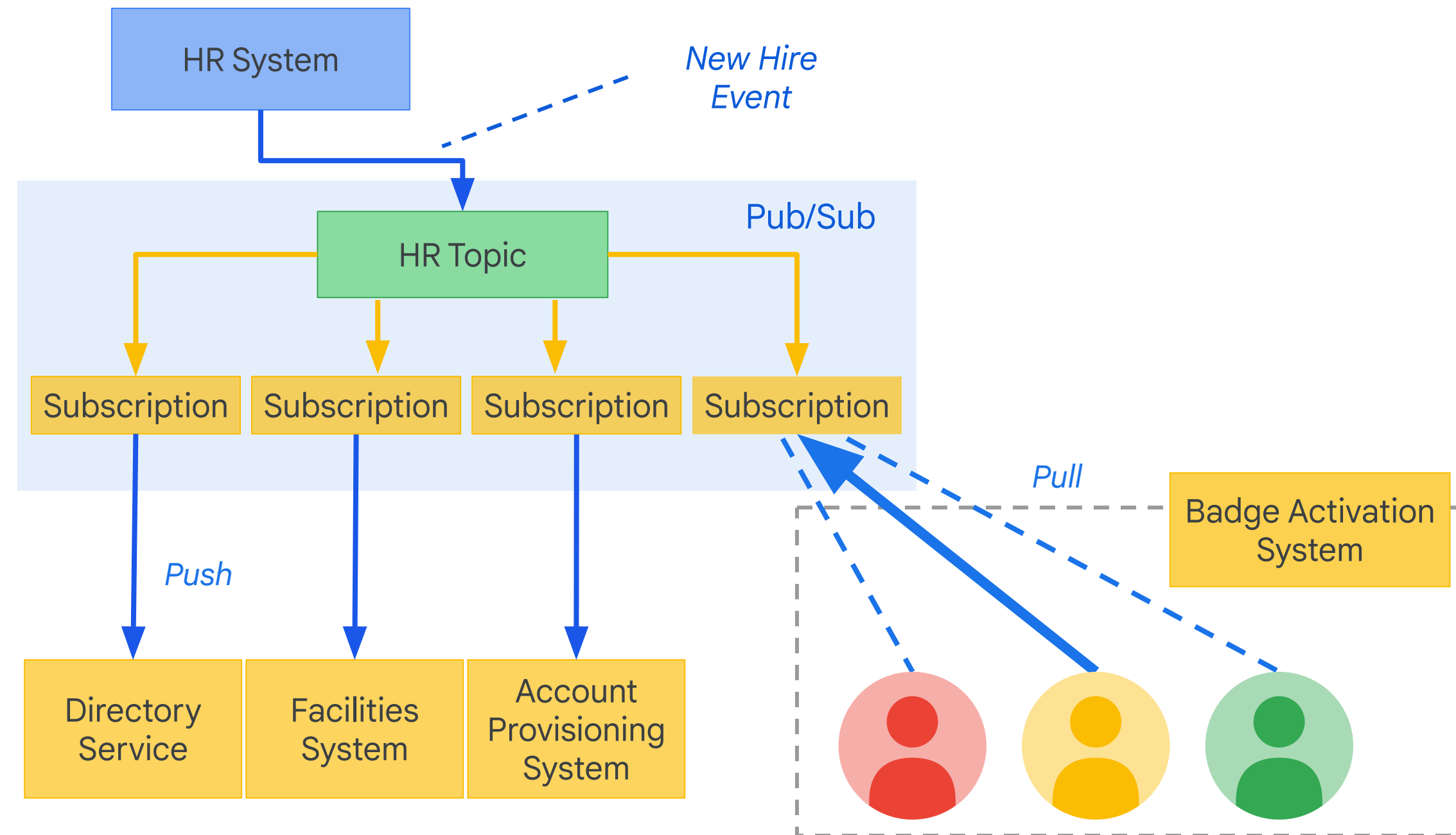


# There can be multiple Subscriptions for each Topic

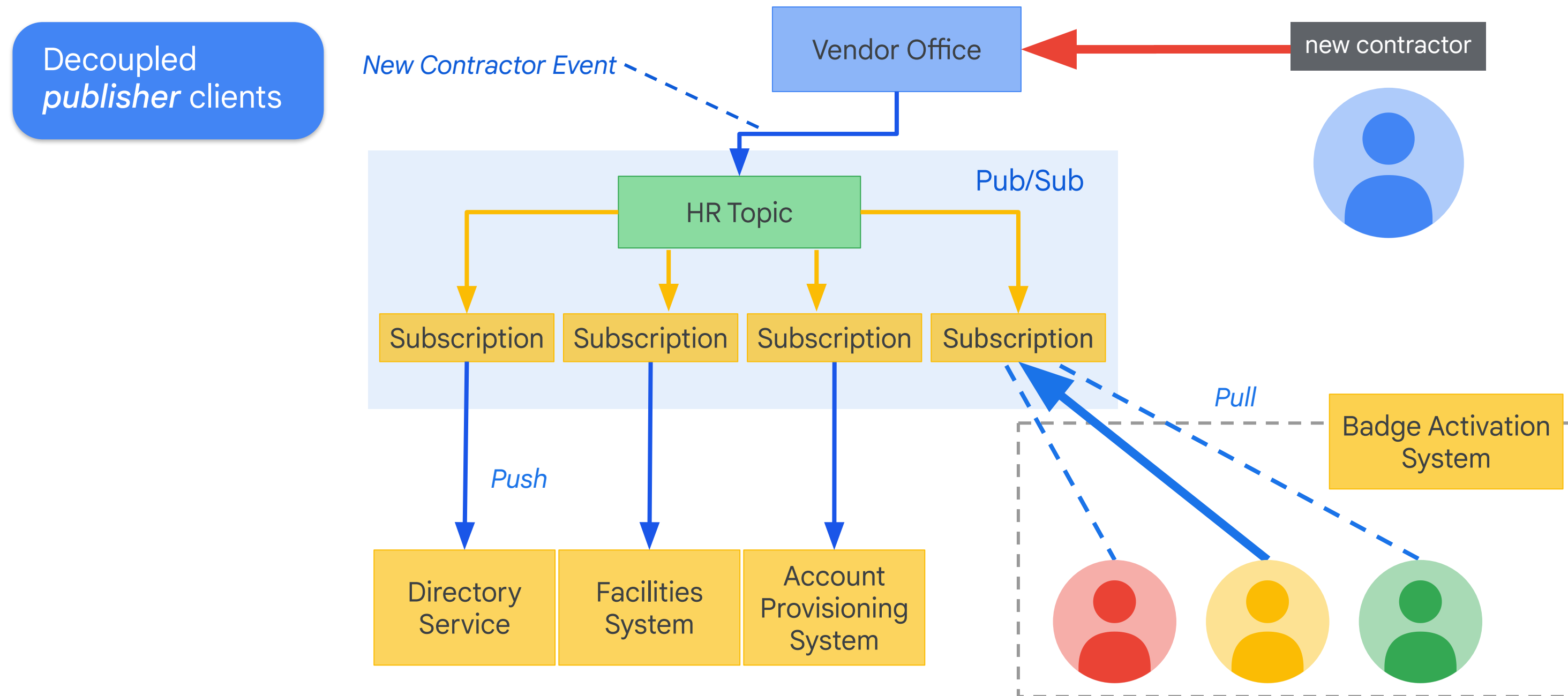




# And there can be multiple subscribers per Subscription

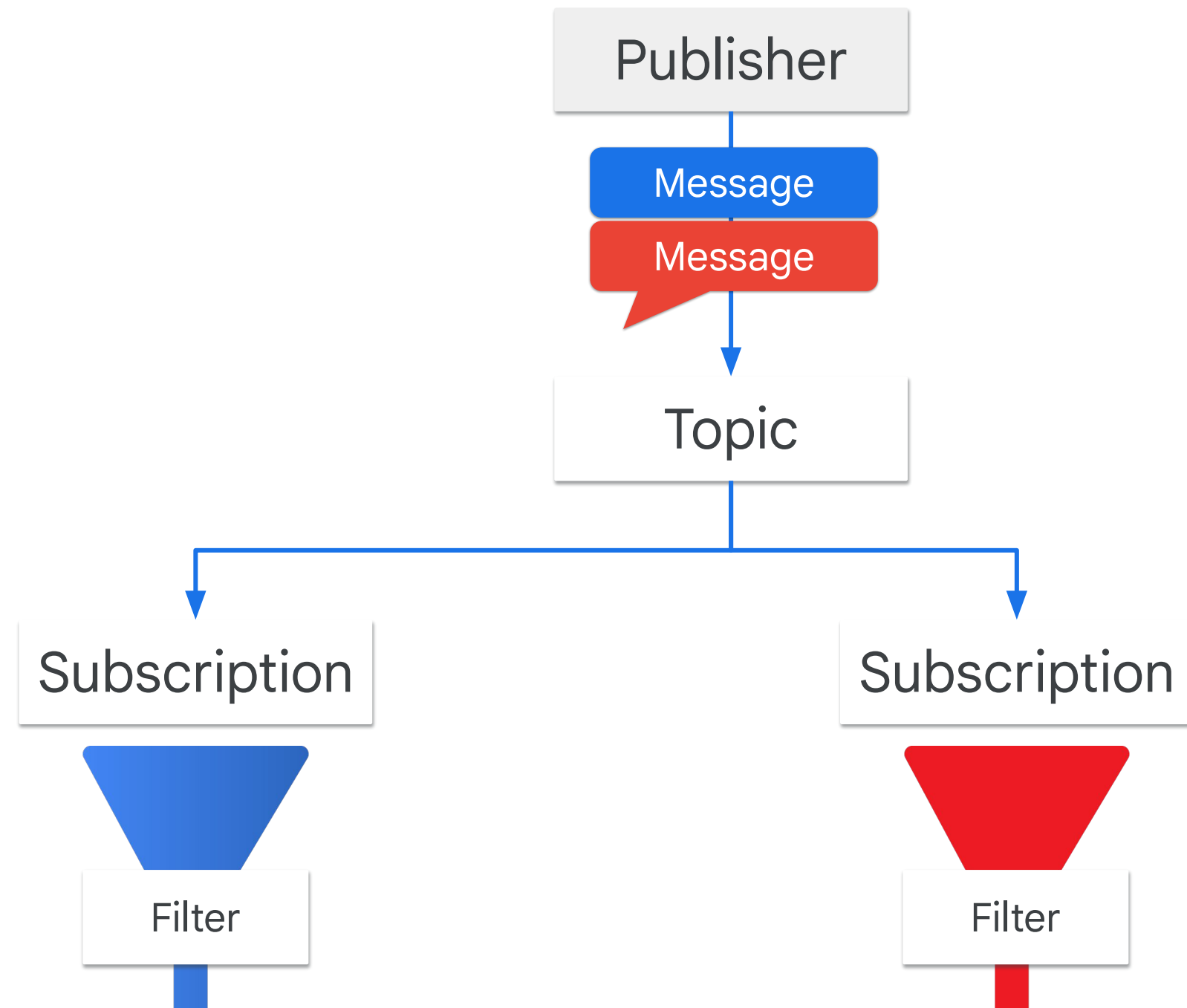


# And there can be multiple publishers to the Topic



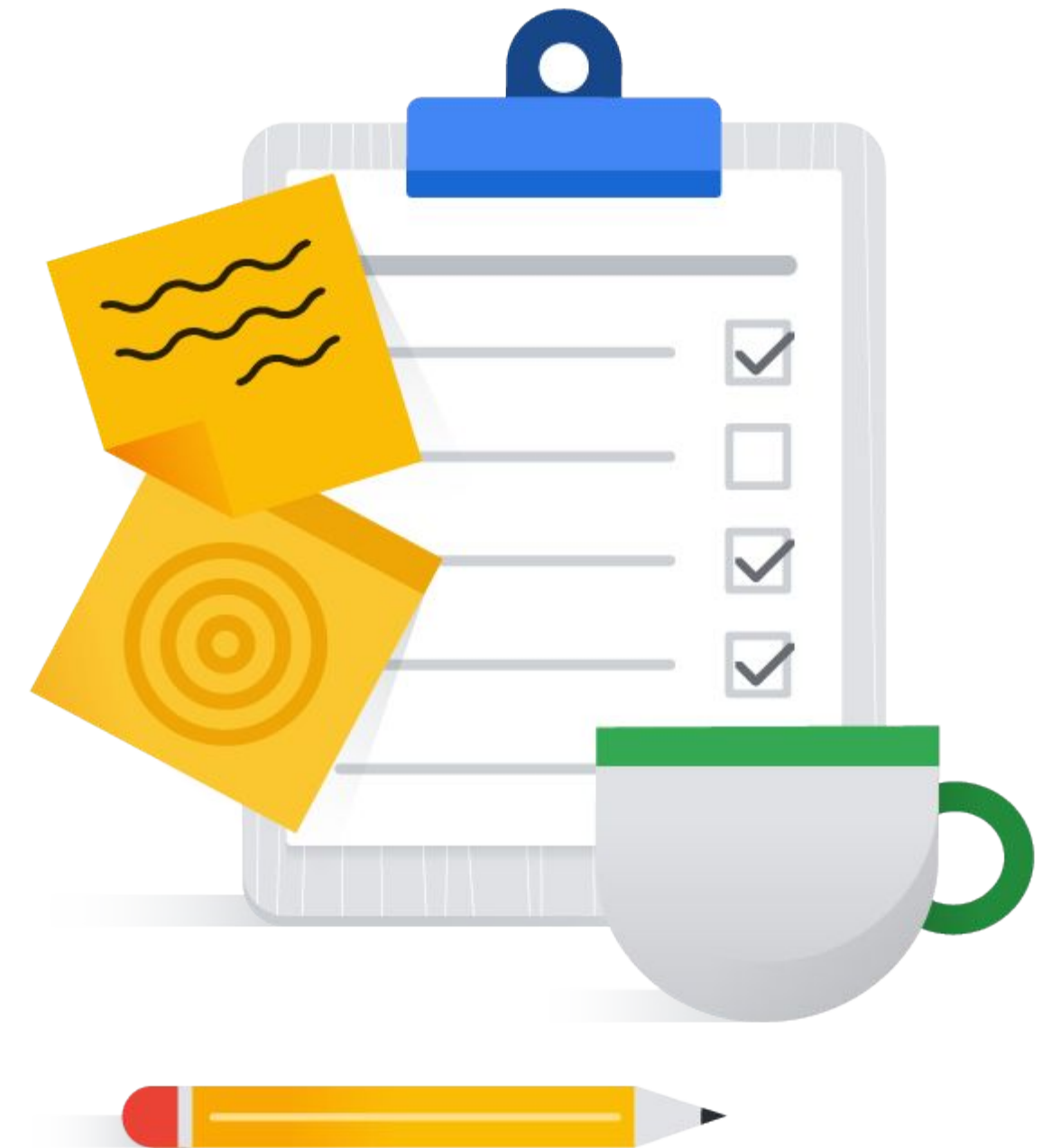
# You can filter messages by their attributes

- Filter the Pub/Sub events on the message attributes.
- Configure via the Cloud Console, the `gcloud` command-line tool, or the Pub/Sub API.

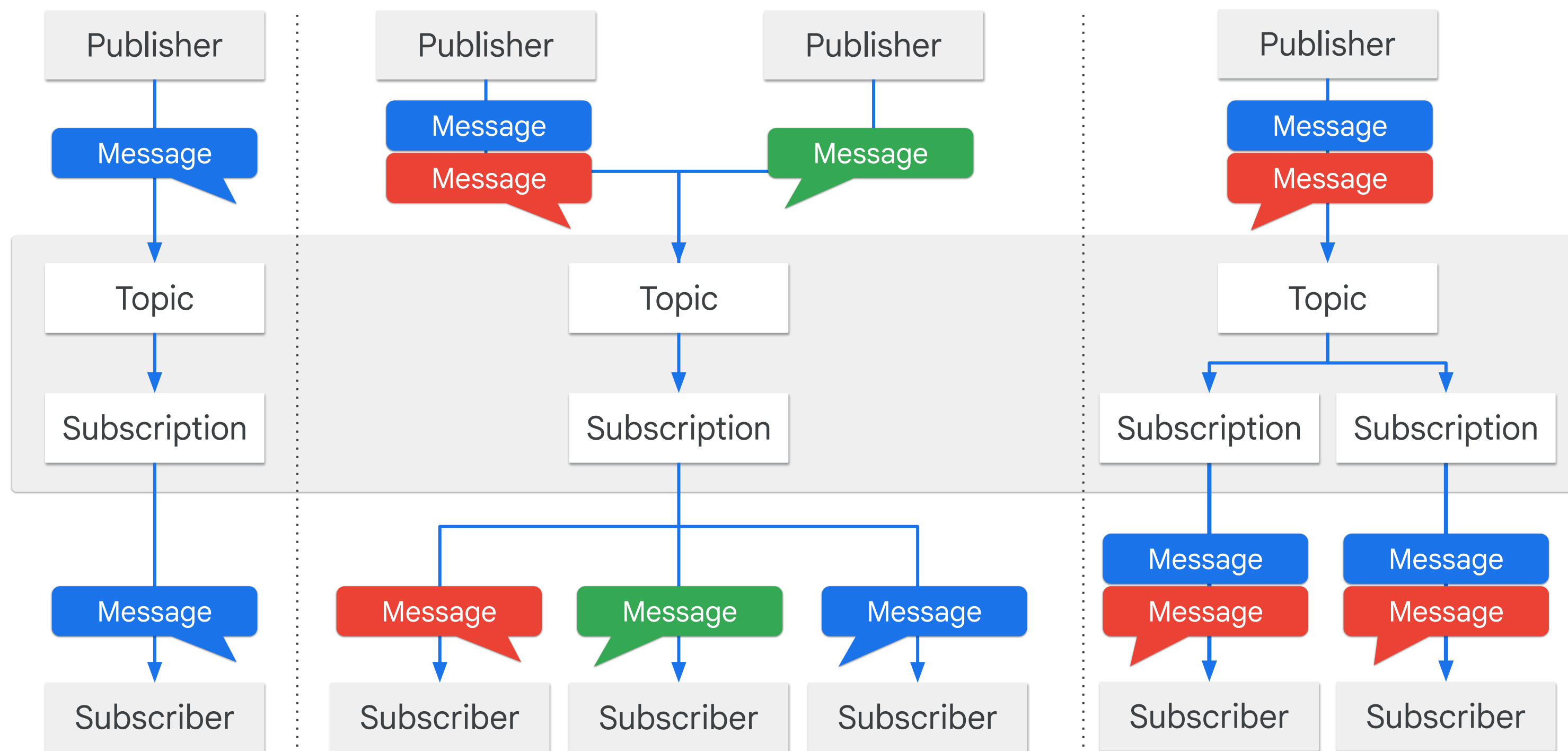


# Serverless Messaging with Pub/Sub

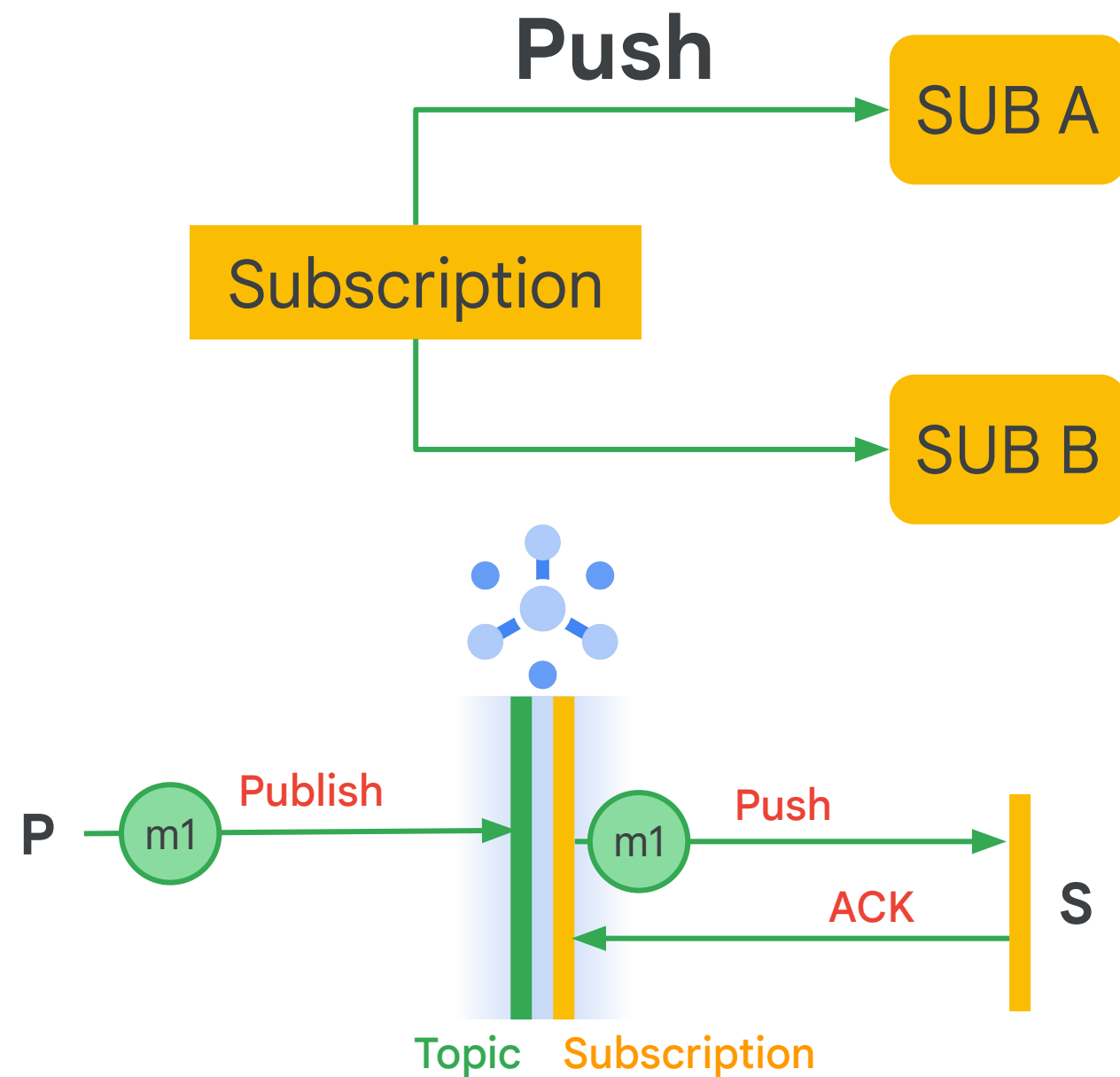
01	Introduction to Pub/Sub
02	Pub/Sub push versus pull
03	Publishing with Pub/Sub code



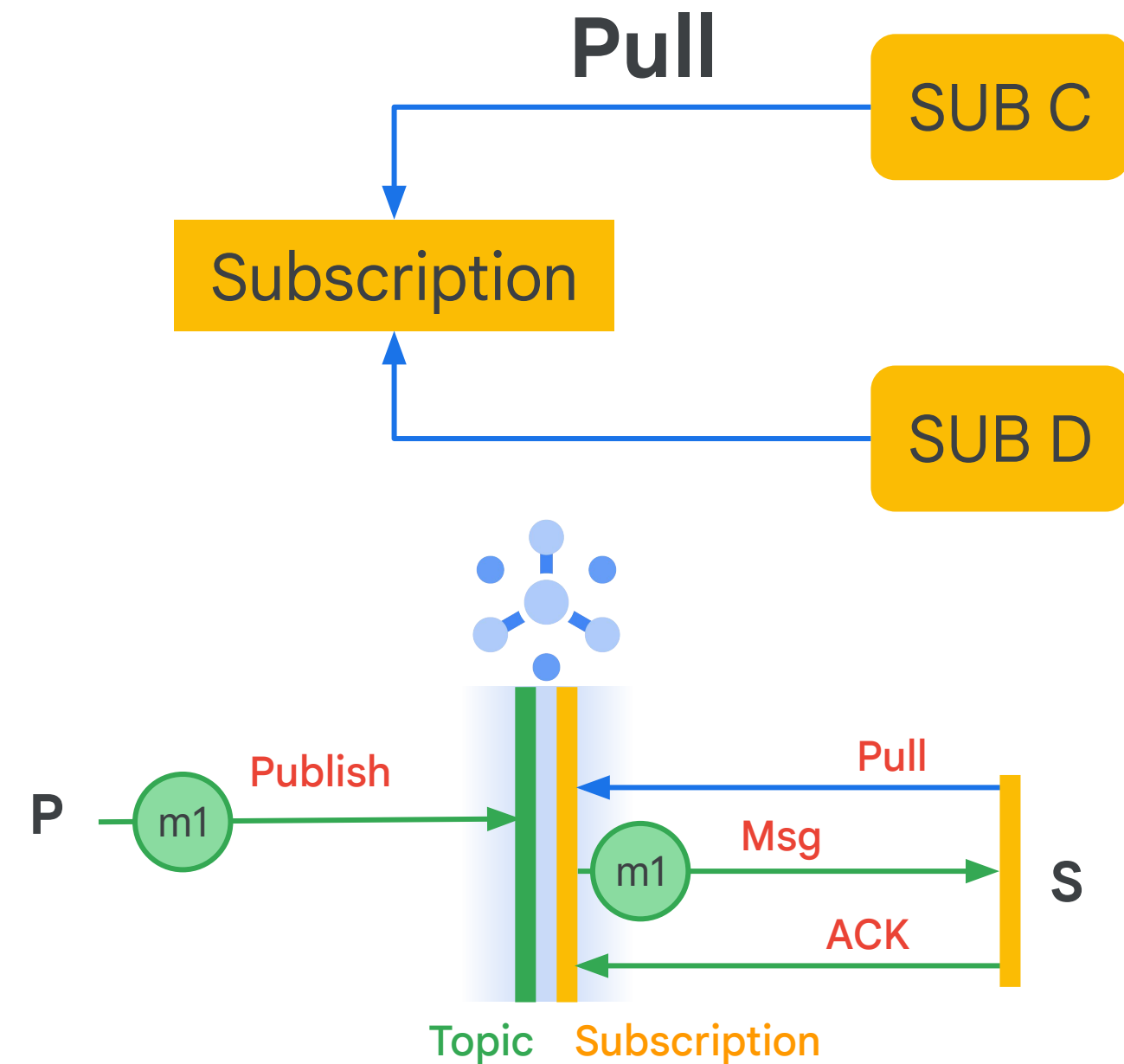
# Publish/Subscribe patterns



# Pub/Sub provides both Push and Pull delivery



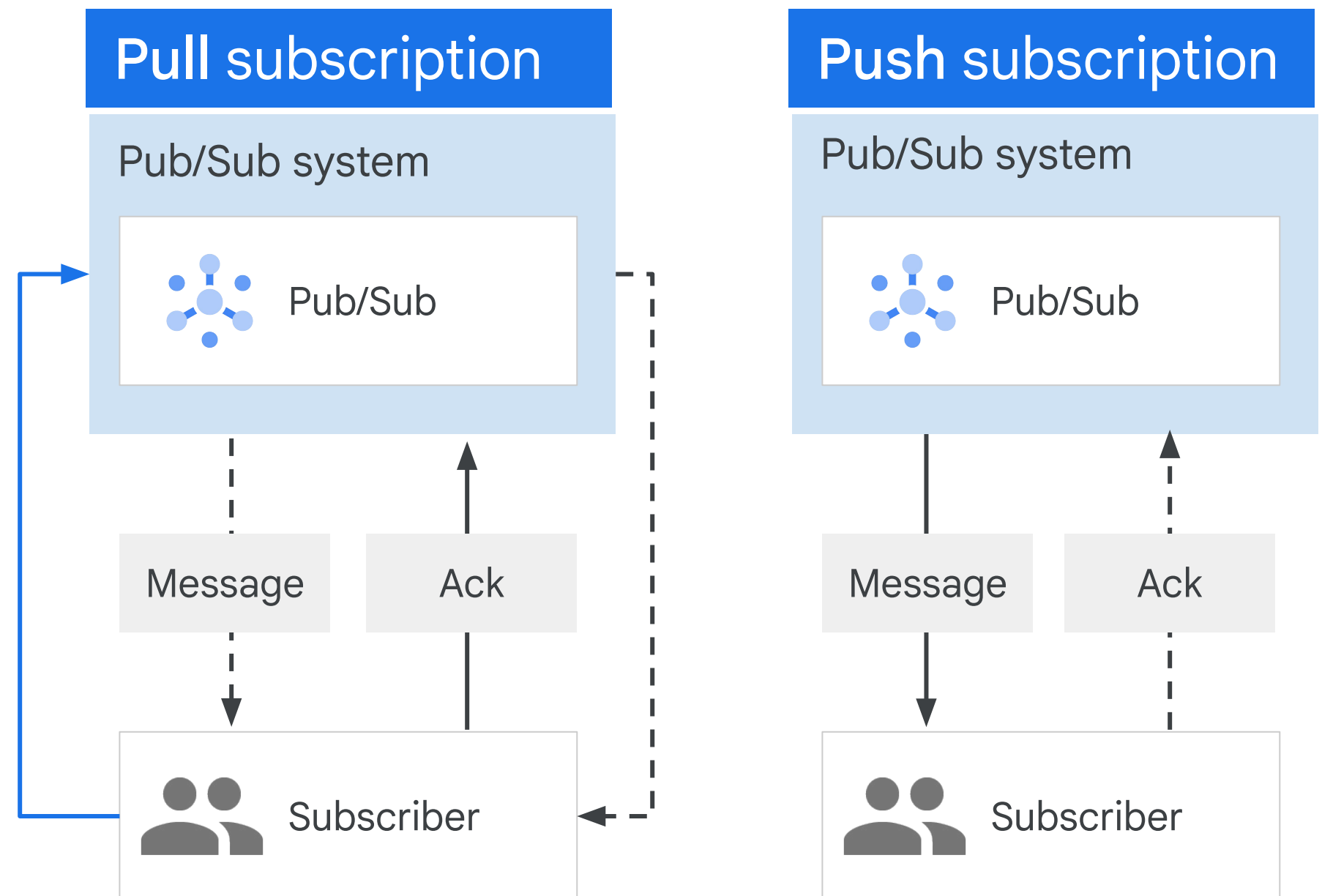
Acknowledgement used for dynamic rate control



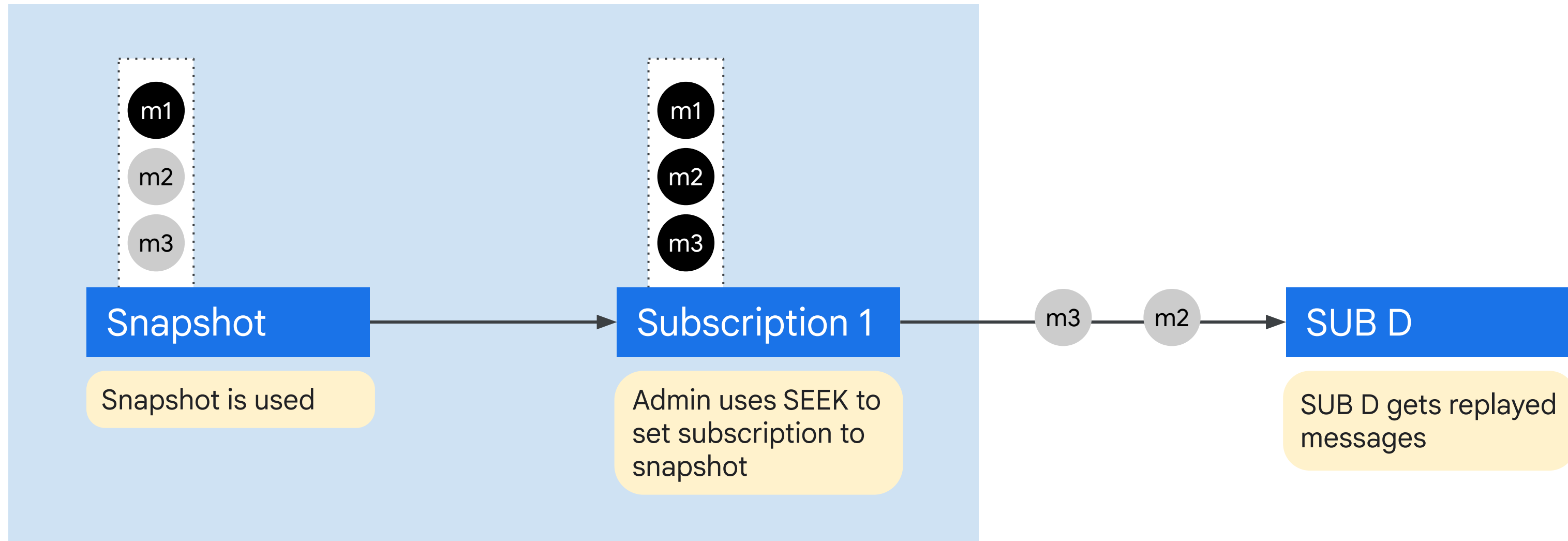
Messages are stored up to 7 days

# At least once delivery guarantee

- A subscriber ACKs each message for every subscription
- A message is resent if subscriber takes more than **ackDeadline** to respond
- Messages are stored for up to 7 days
- A subscriber can extend the deadline per message

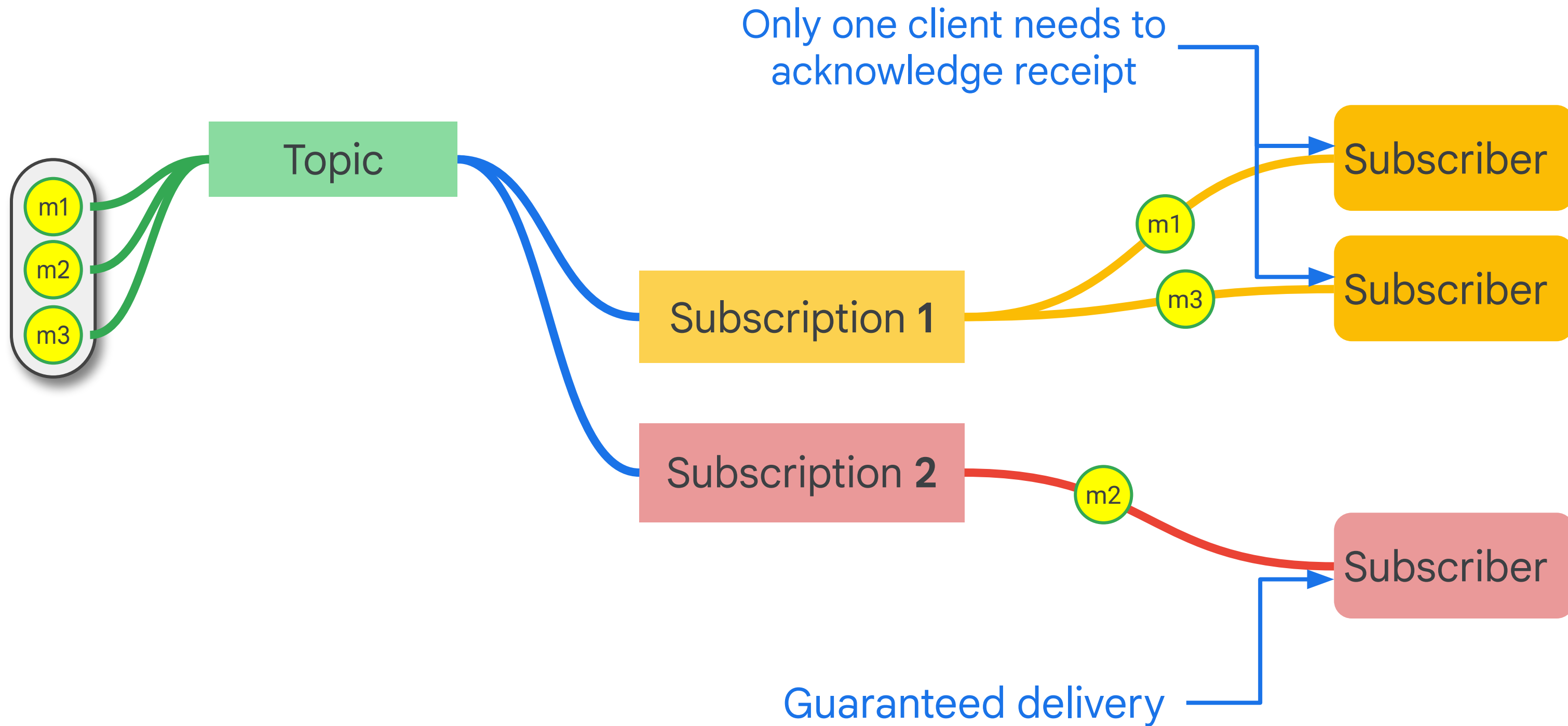


# Message replay



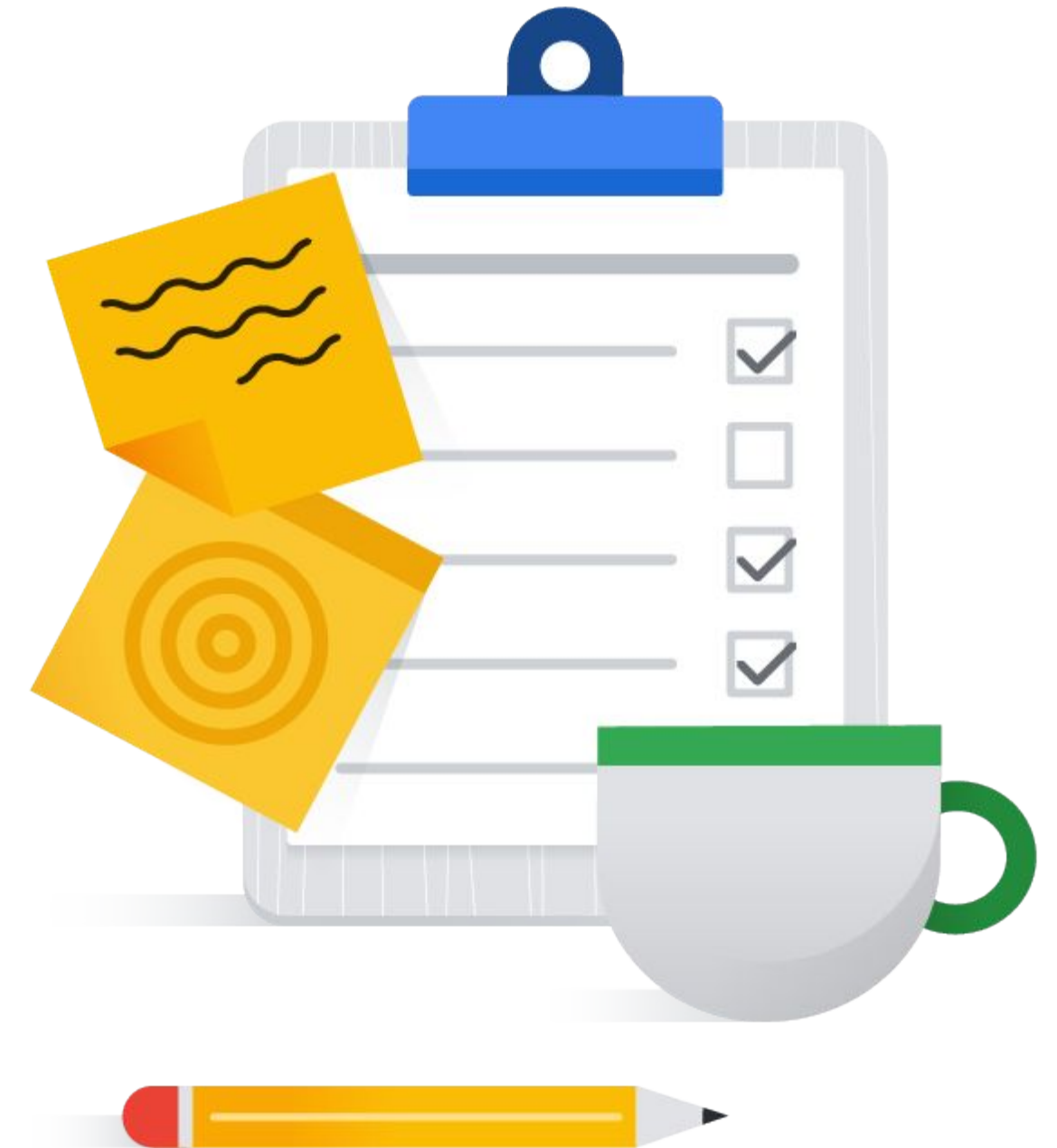


# Subscribers can work as a team or separately



# Serverless Messaging with Pub/Sub

- |    |                              |
|----|------------------------------|
| 01 | Introduction to Pub/Sub      |
| 02 | Pub/Sub push versus pull     |
| 03 | Publishing with Pub/Sub code |



# Publishing with Pub/Sub

```
gcloud pubsub topics create sandiego
```

Create topic

```
gcloud pubsub topics publish sandiego --message "hello"
```

Publish to topic

```
import os
from google.cloud import pubsub_v1

publisher = pubsub_v1.PublisherClient()

topic_name = 'projects/{project_id}/topics/{topic}'.format(
    project_id=os.getenv('GOOGLE_CLOUD_PROJECT'),
    topic='MY_TOPIC_NAME',
)

publisher.create_topic(topic_name)
publisher.publish(topic_name, b'My first message!', author='dylan')
```

Python

Set topic name

Message

Send attribute

Create a client

# Subscribing with Pub/Sub using async pull

```
import os
from google.cloud import pubsub_v1

subscriber = pubsub_v1.SubscriberClient()
topic_name = 'projects/{project_id}/topics/{topic}'.format(
    project_id=os.getenv('GOOGLE_CLOUD_PROJECT'),
    topic='MY_TOPIC_NAME'
)
subscription_name = 'projects/{project_id}/subscriptions/{sub}'.format(
    project_id=os.getenv('GOOGLE_CLOUD_PROJECT'),
    sub='MY_SUBSCRIPTION_NAME'
)
subscriber.create_subscription(
    name=subscription_name, topic=topic_name)

def callback(message):
    print(message.data)
    message.ack()

future = subscriber.subscribe(subscription_name, callback)
```

Python

Create a client

Pull method  
Callback function

# Subscribing with Pub/Sub using synchronous pull

```
gcloud pubsub subscriptions create --topic sandiego mySub1
```

Create subscription

```
gcloud pubsub subscriptions pull --auto-ack mySub1
```

Pull subscription

```
import time
```

```
from google.cloud import pubsub_v1
```

```
subscriber = pubsub_v1.SubscriberClient()
```

```
subscription_path = subscriber.subscription_path(project_id, subscription_name)
```

----- Set subscription name

Create a client

``projects/{project_id}/subscriptions/{subscription_name}``

subscription\_path  
format

```
NUM_MESSAGES = 2
```

```
ACK_DEADLINE = 30
```

```
SLEEP_TIME = 10
```

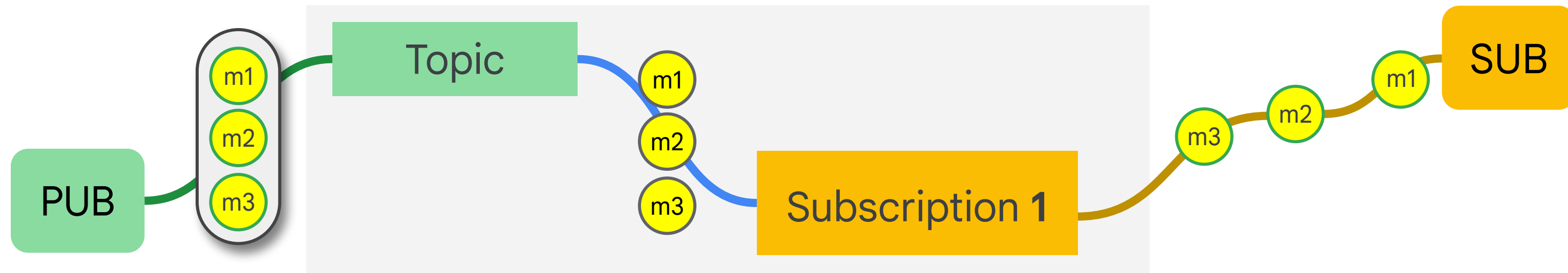
```
# The subscriber pulls a specific number of messages.
```

```
response = subscriber.pull(subscription_path, max_messages=NUM_MESSAGES)
```

Subscriber is  
non-blocking

Keep the main thread  
from exiting to allow it  
to process messages  
synchronously

By default, the Pub/Sub publishing engine batches messages; turn this off if you desire lower latency



Batching messages: throughput versus latency

# Changing the batch settings in Pub/Sub

**Python**

```
from google.cloud import pubsub
from google.cloud.pubsub import types

client = pubsub.PublisherClient(
    batch_settings=BatchSettings(max_messages=500),
)
```

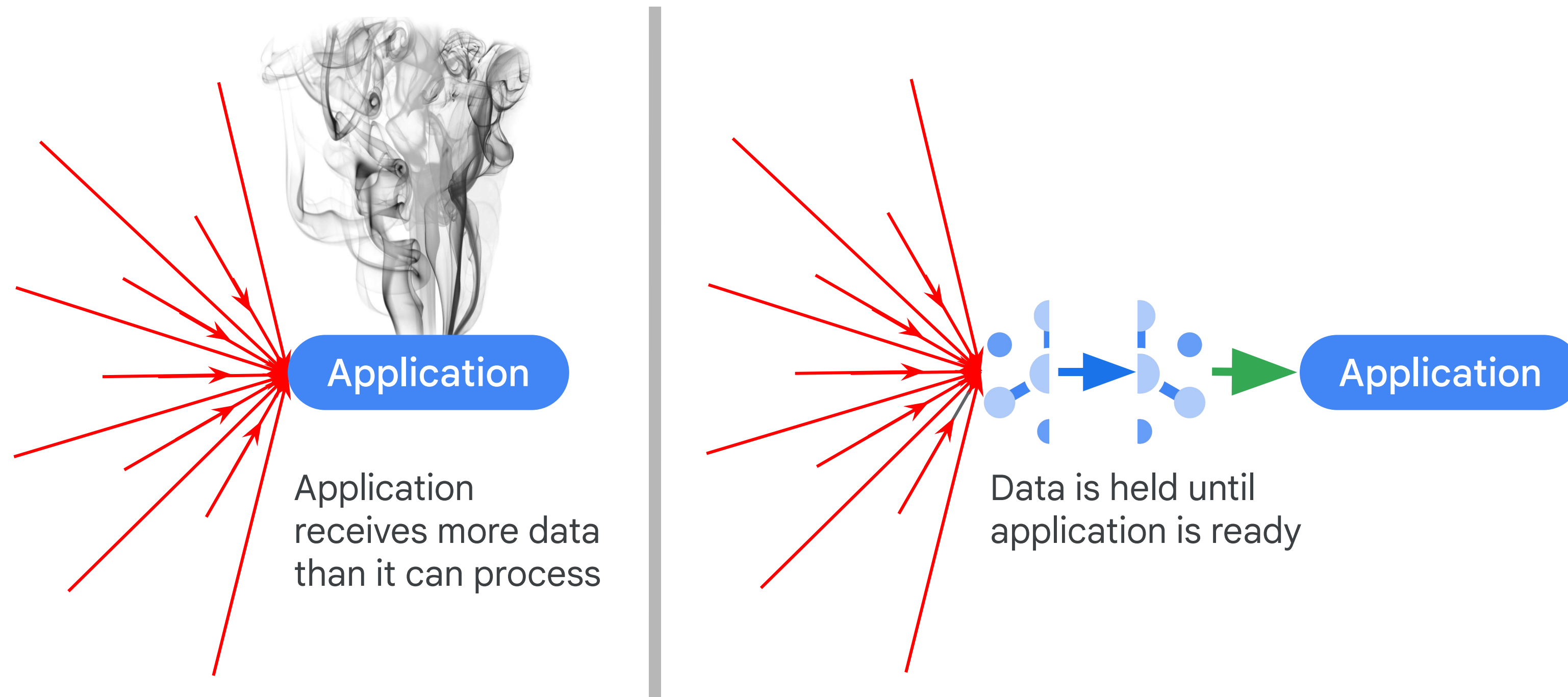
Change batch setting

# Pub/Sub: Latency, out-of-order, duplication will happen

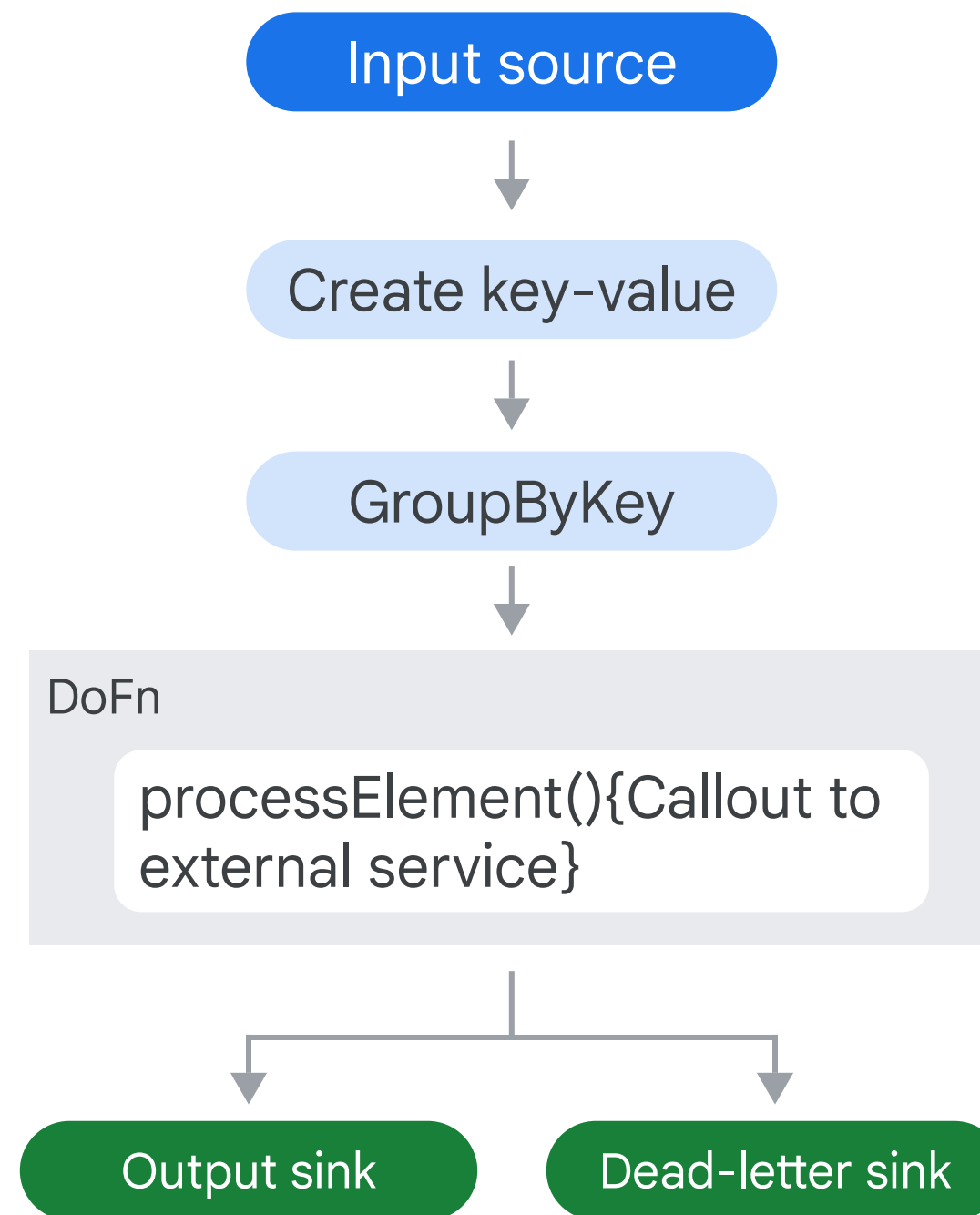
- If messages have the same ordering key and are in the same region, you can enable message ordering.
- To receive the messages in order, set the message ordering property on the subscription you receive messages from using the Cloud Console, the `gcloud` command-line tool, or the Pub/Sub API.
- Receiving messages in order might increase latency.



# Use Pub/Sub for streaming resilience



# Dead-letter sinks and error logging



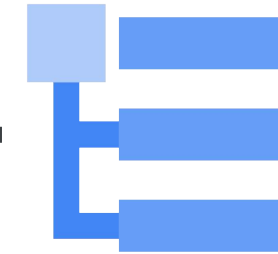
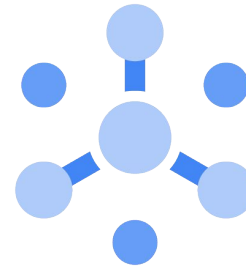
# Exponential backoff

- Pub/Sub lets you configure an exponential backoff policy for better flow control.
- The idea behind exponential backoff is to add progressively longer delays between retry attempts.
- To create a new subscription with an exponential backoff retry policy, run the `gcloud pubsub create` command or use the Cloud console.

# Security, monitoring, and logging for Pub/Sub

## Cloud Logging metrics

pubsub\_topic  
pubsub\_subscription



## Cloud Audit Logs

Admin = on  
Data Access = off

## Authentication

Service accounts  
User accounts



## Access control

Pub/Sub roles per topic

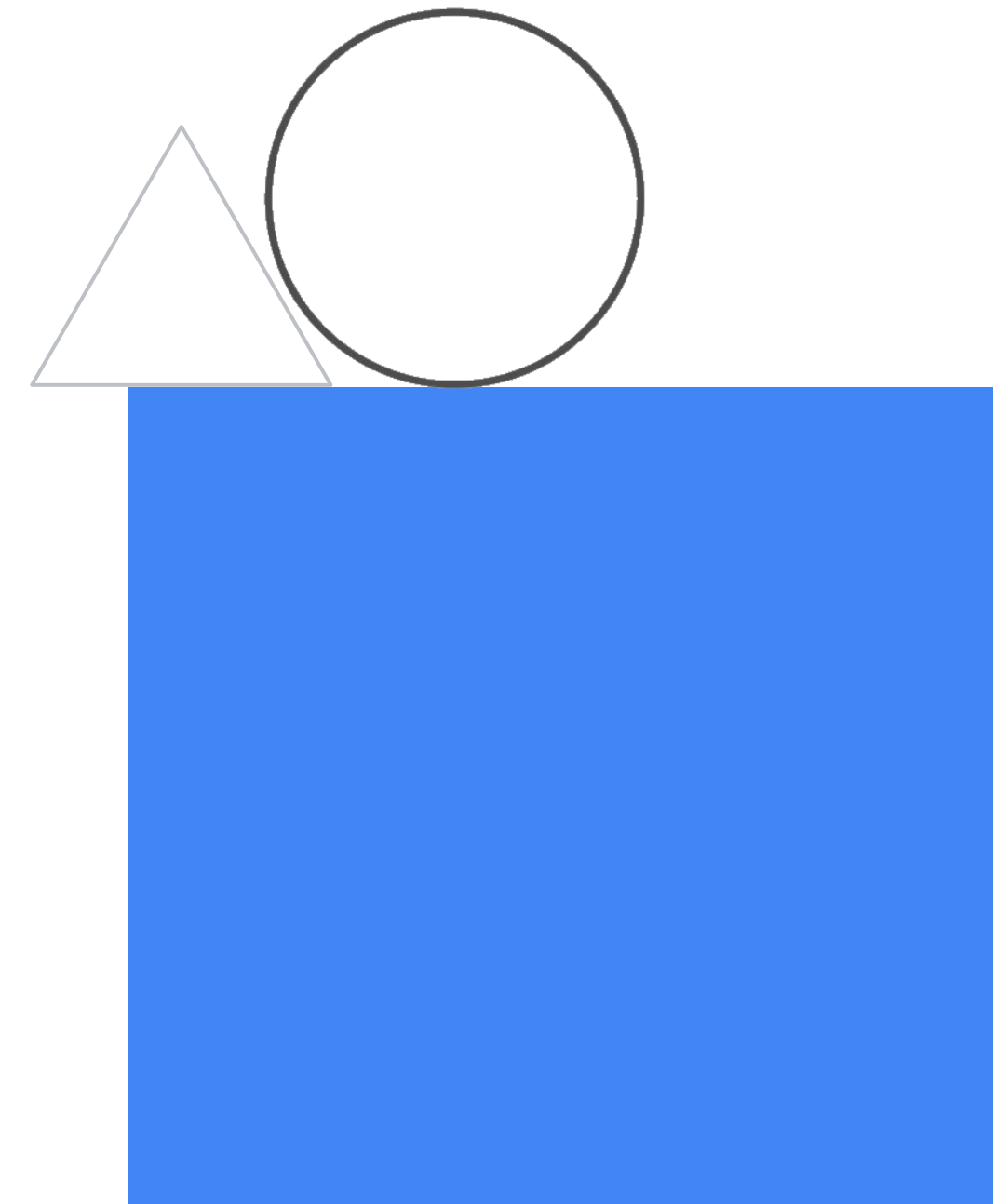
# Summary

Latency, out-of-order, duplication will happen

Pub/Sub with Dataflow: Exactly once, ordered processing

# Lab Intro

Streaming Data Processing:  
Publish Streaming Data into Pub/Sub



# Lab objectives

01

Create a Pub/Sub topic and subscription

02

Simulate your traffic sensor data into Pub/Sub



