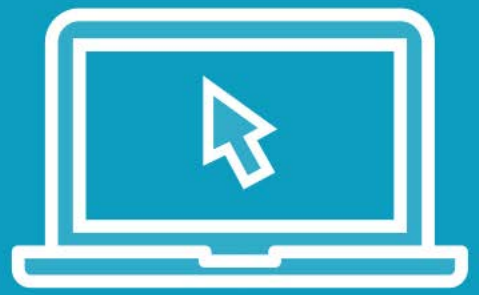
Understanding Topics, Partitions, and Brokers

Ryan Plant
COURSE AUTHOR

@ryan_plant blog.ryanplant.com

Demo



Basic Apache Kafka installation:

- Download the binary package
- Extract the archive
- Explore the installation directory contents

Prerequisites:

- Linux operating system
- Java 8 JDK installed
- Scala 2.11.x installed

Apache Kafka Topics

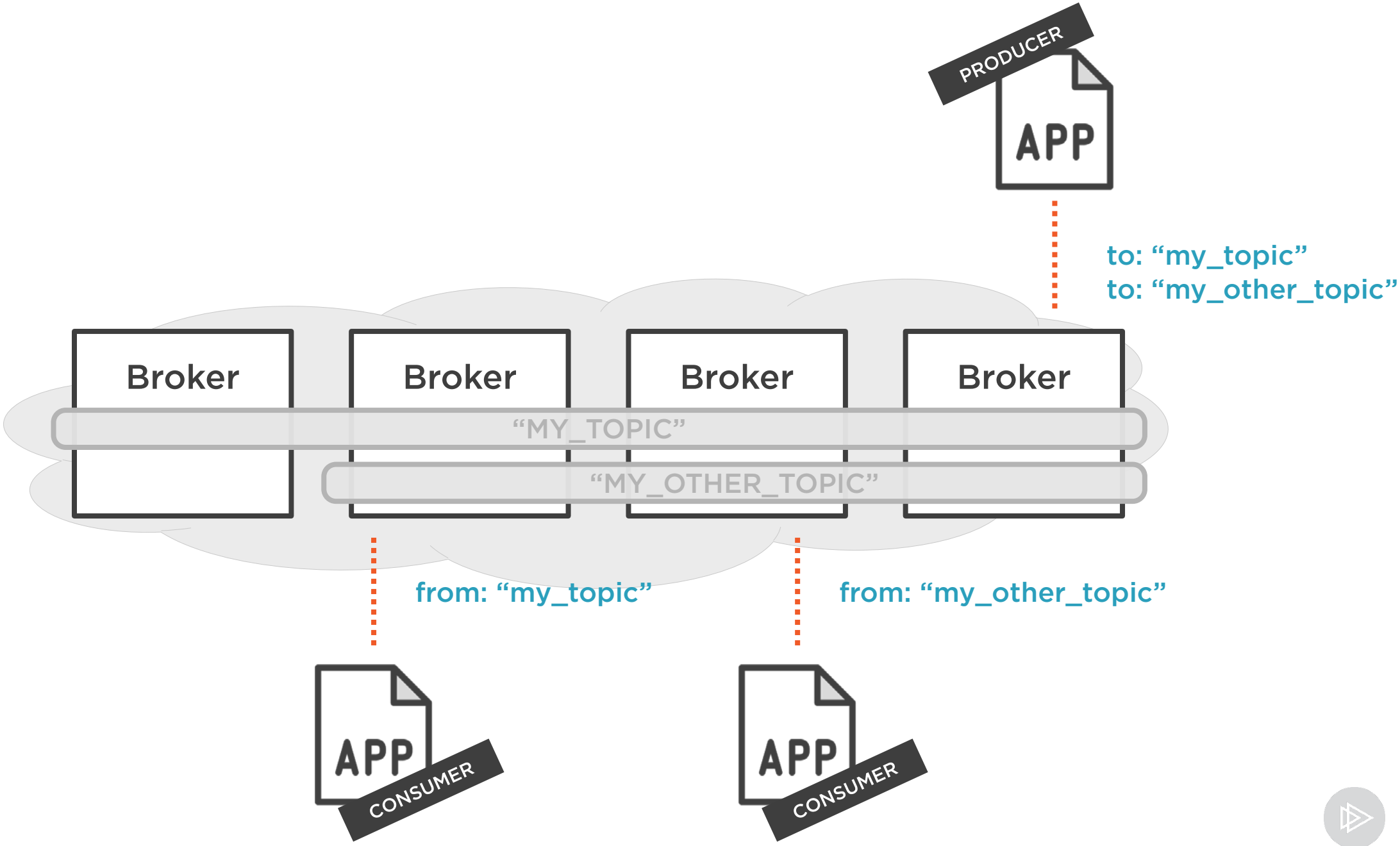
Central Kafka abstraction

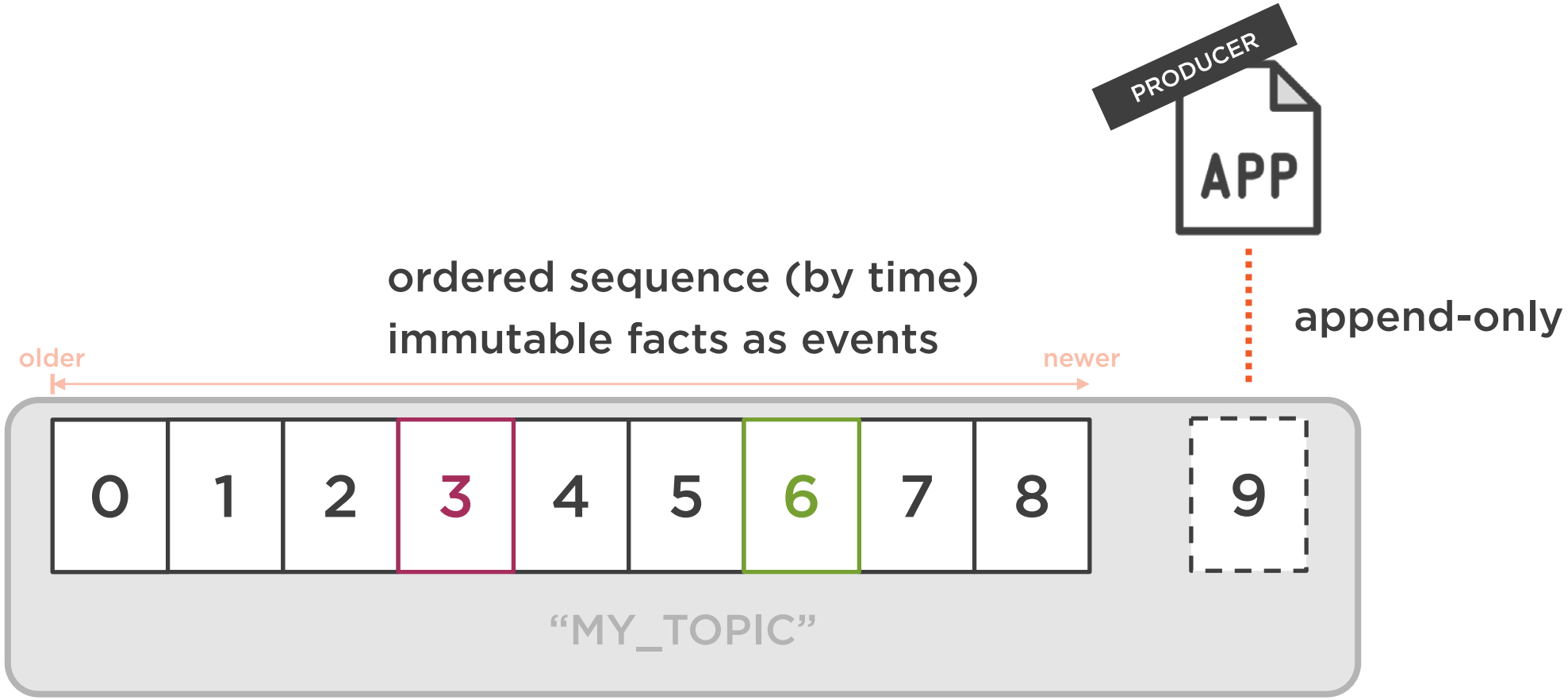
Named feed or category of messages

- Producers produce to a topic
- Consumers consume from a topic

Logical entity

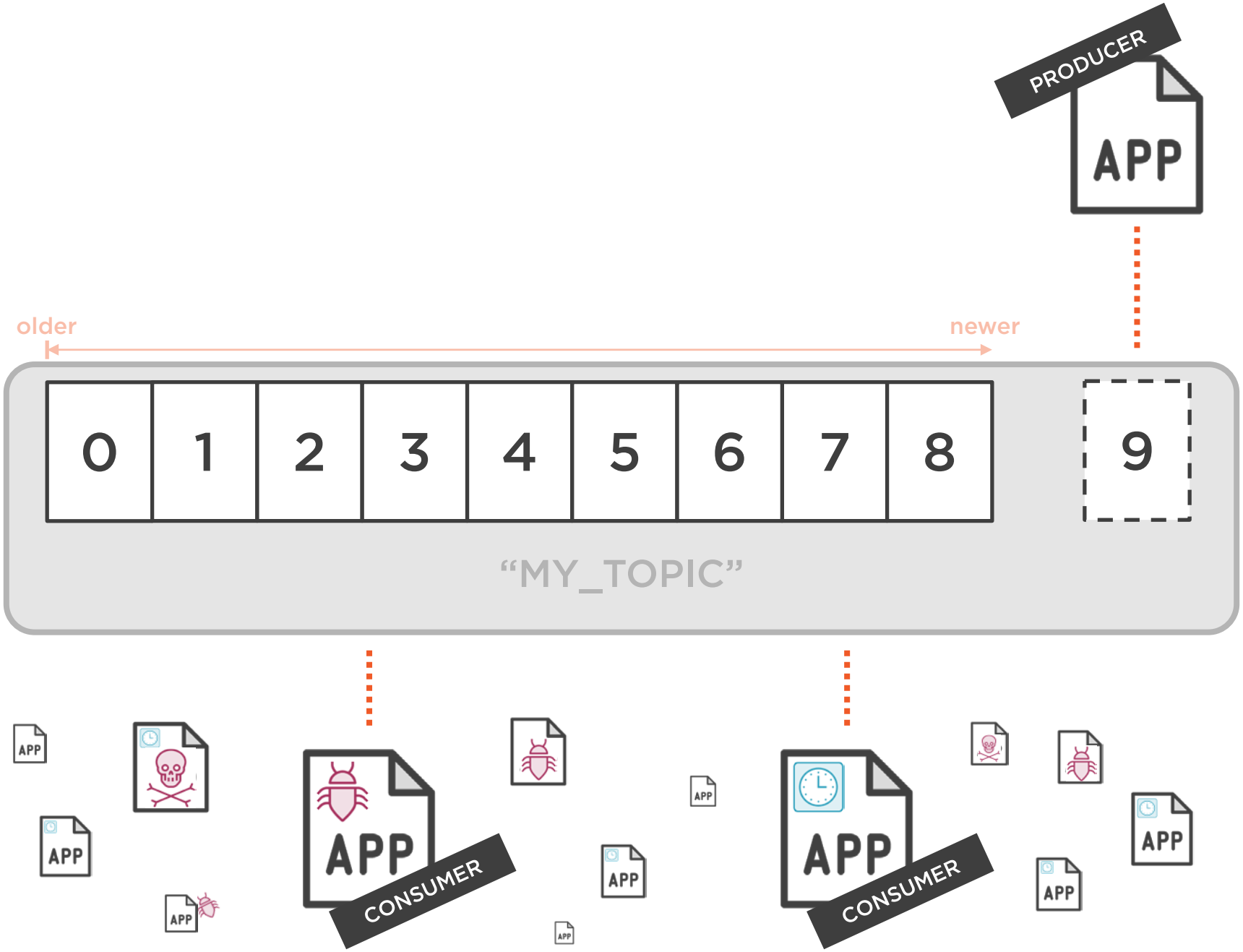
Physically represented as a log


PRODUCER
APP
to: “my_topic”
to: “my_other_topic”
Broker
Broker
Broker
Broker
“MY_TOPIC”
“MY_OTHER_TOPIC”
from: “my_topic”
from: “my_other_topic”
APP
CONSUMER
APP
CONSUMER

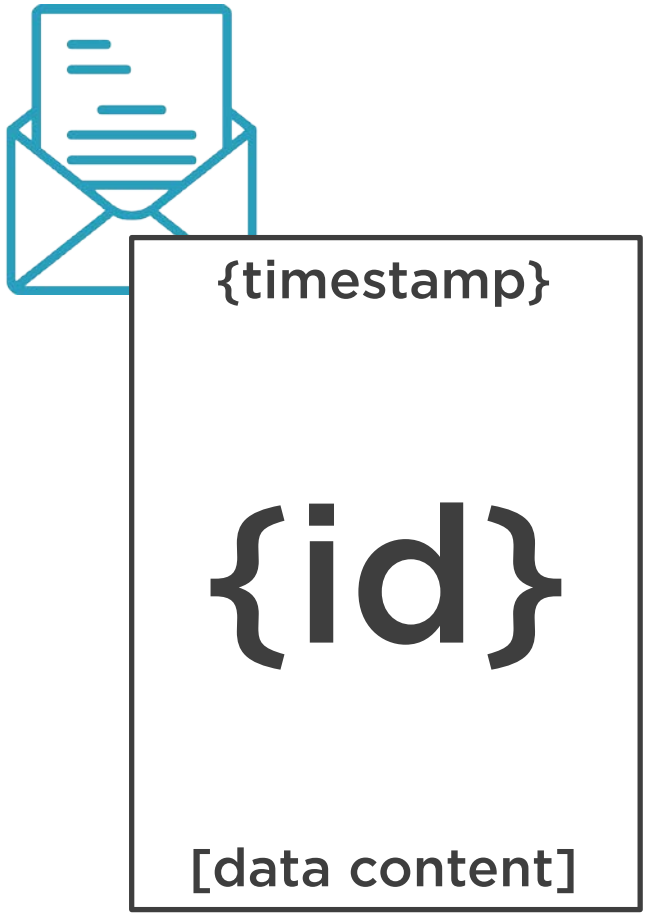

PRODUCER
APP
ordered sequence (by time)
immutable facts as events
append-only
older
newer
0
1
2
3
4
5
6
7
8
9
“MY_TOPIC”

Event Sourcing

An architectural style or approach to maintaining an application's state by capturing all changes as a sequence of time-ordered, immutable events.

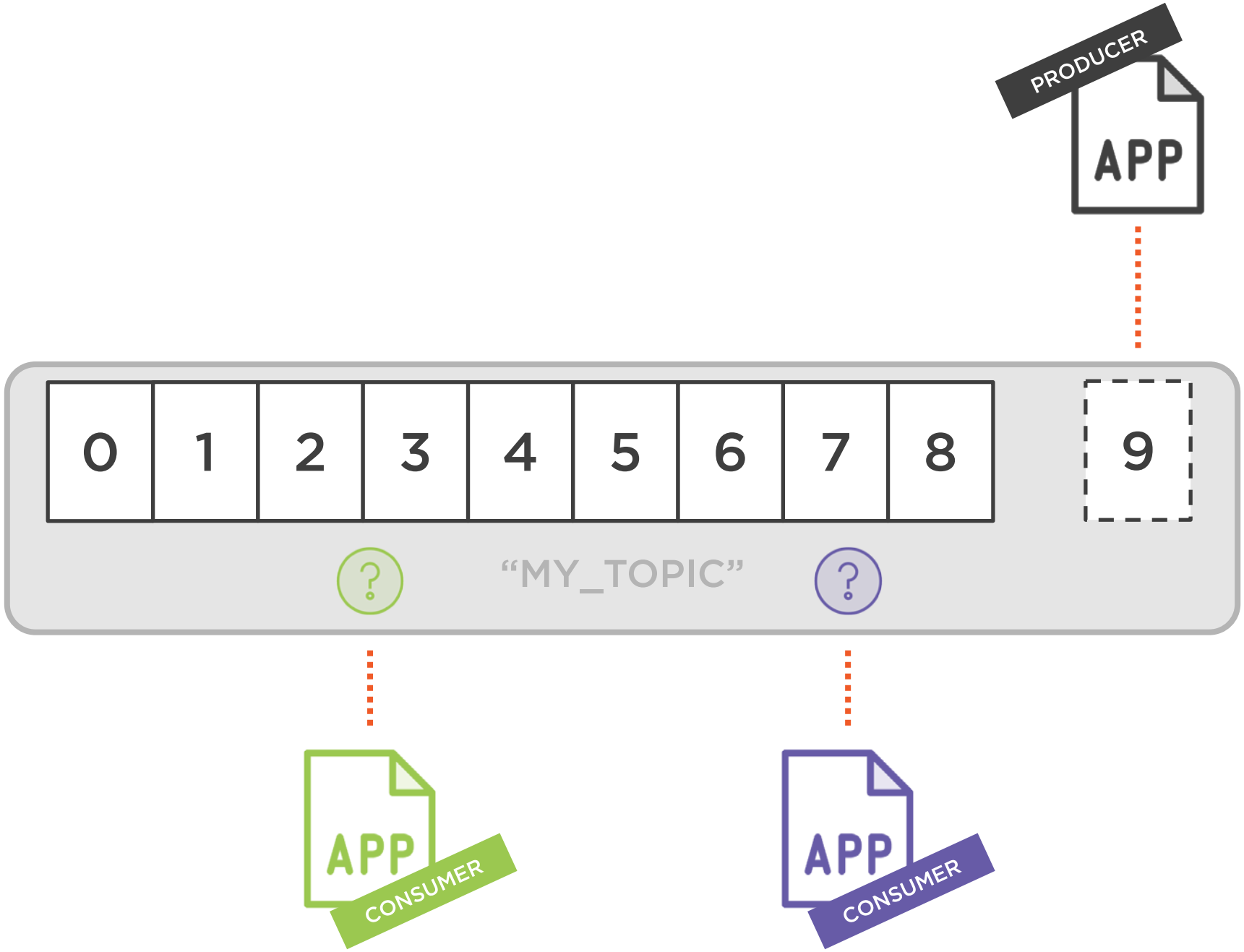

PRODUCER
APP
older
newer
0
1
2
3
4
5
6
7
8
9
"MY_TOPIC"
APP
CONSUMER
APP
CONSUMER

Message Content

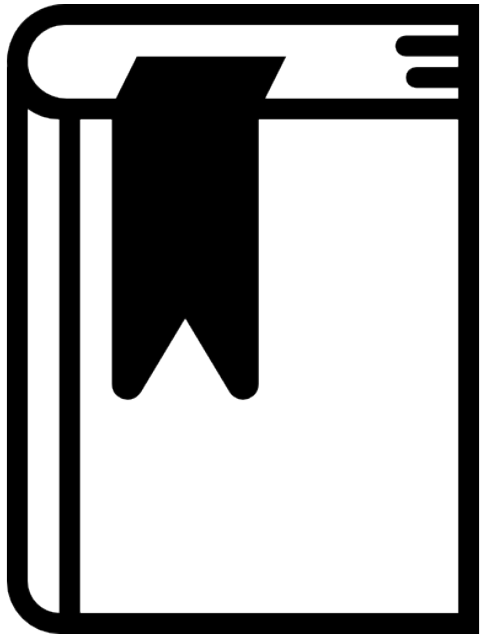


Each message has a:

- Timestamp
- Referenceable identifier
- Payload (binary)

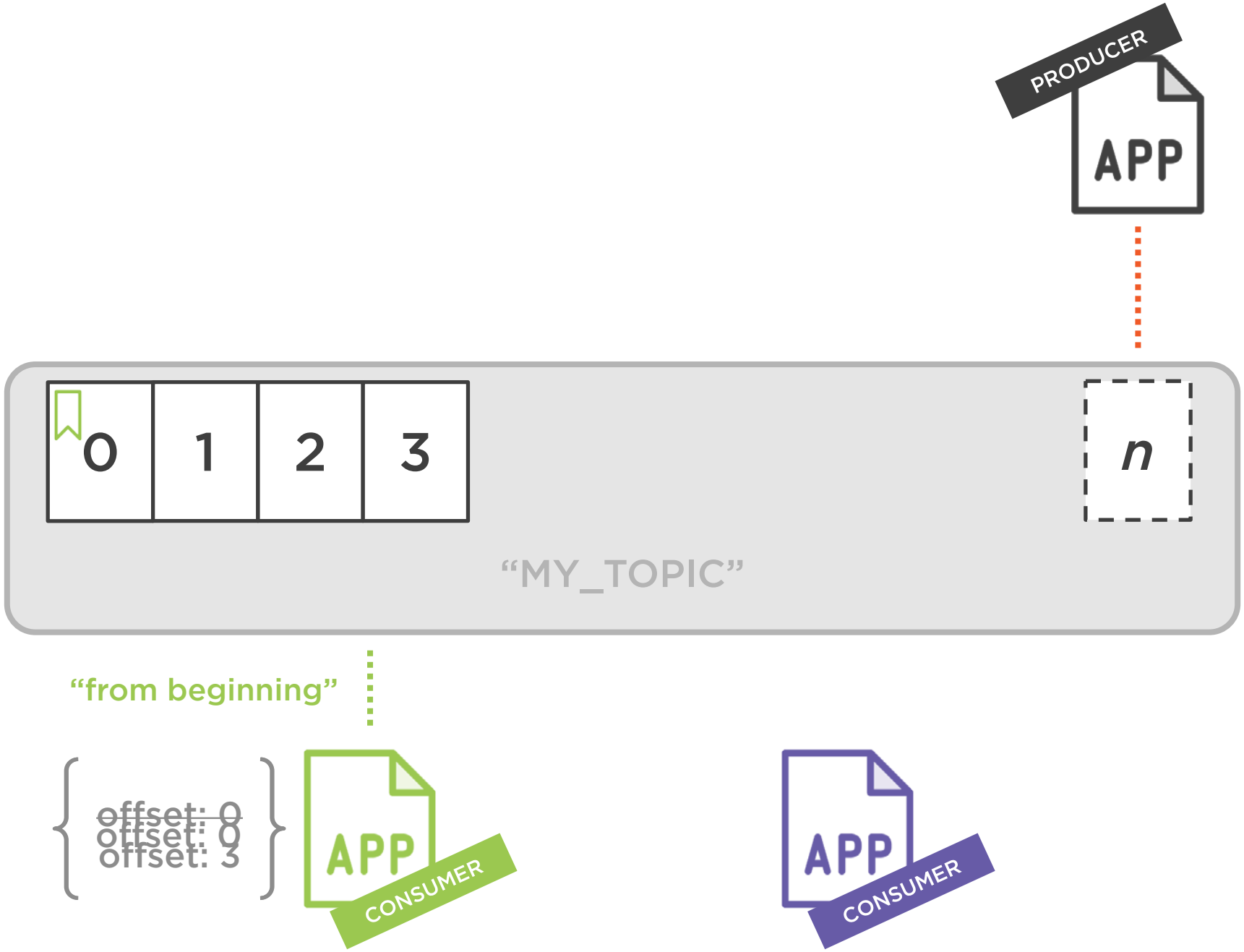

PRODUCER
APP
0
1
2
3
4
5
6
7
8
9
?
"MY_TOPIC"
?
APP
CONSUMER
APP
CONSUMER

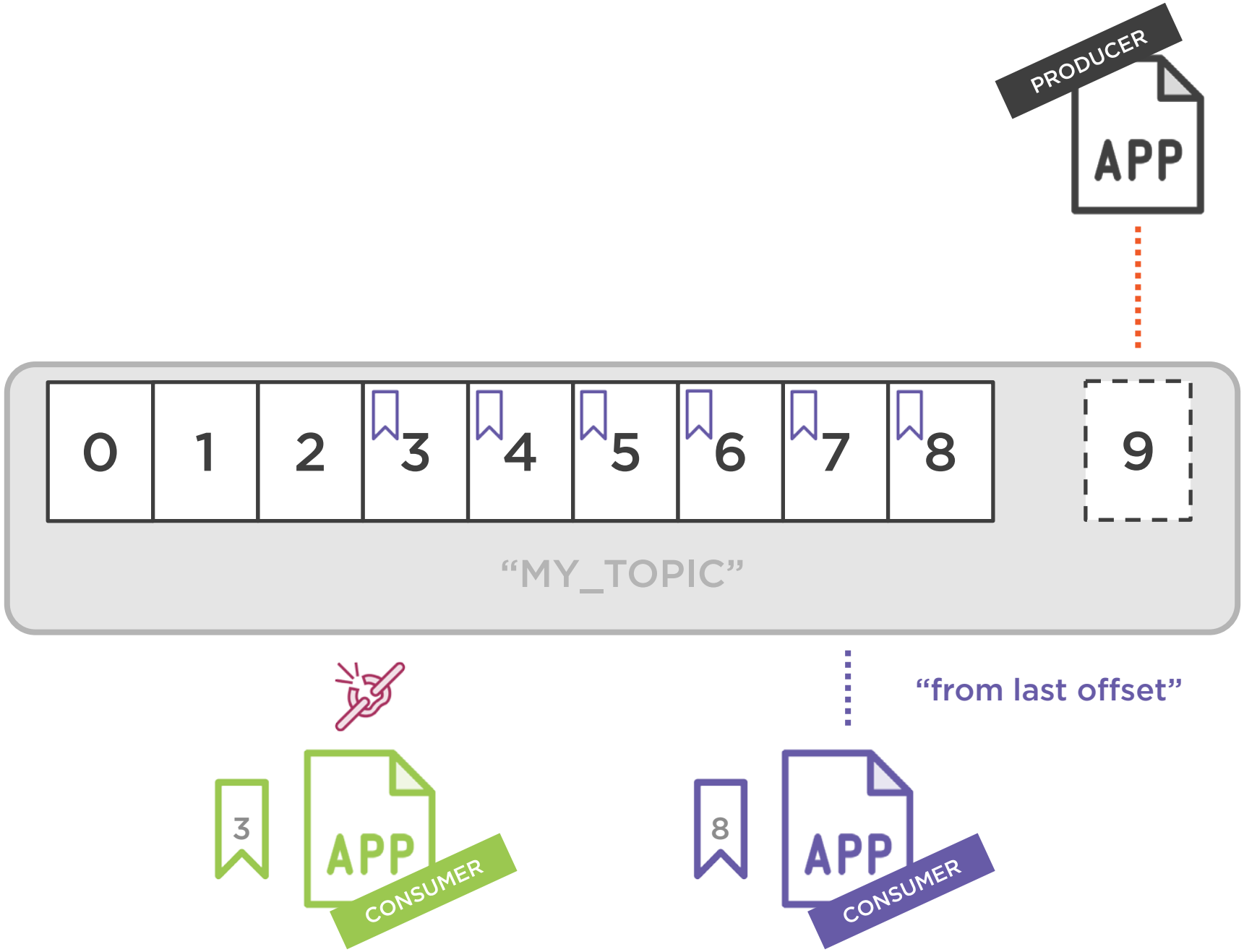
The Offset



A placeholder:

- Last read message position
- Maintained by the Kafka Consumer
- Corresponds to the message identifier


PRODUCER
APP
0
1
2
3
n
“MY_TOPIC”
“from beginning”
offset: 0
offset: 0
offset: 3
APP
CONSUMER
APP
CONSUMER


PRODUCER
APP
0
1
2
3
4
5
6
7
8
9
"MY_TOPIC"
"from last offset"
3
APP
CONSUMER
8
APP
CONSUMER

Message Retention Policy

Apache Kafka retains all published messages regardless of consumption

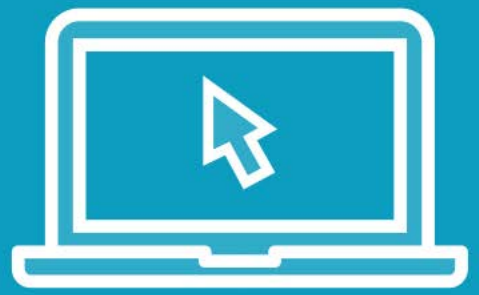
Retention period is configurable

- Default is 168 hours or seven days

Retention period is defined on a per-topic basis

Physical storage resources can constrain message retention

Demo



Simple Kafka cluster setup

Creating an Apache Kafka topic

Producing some messages to the topic

Consuming the messages from the topic

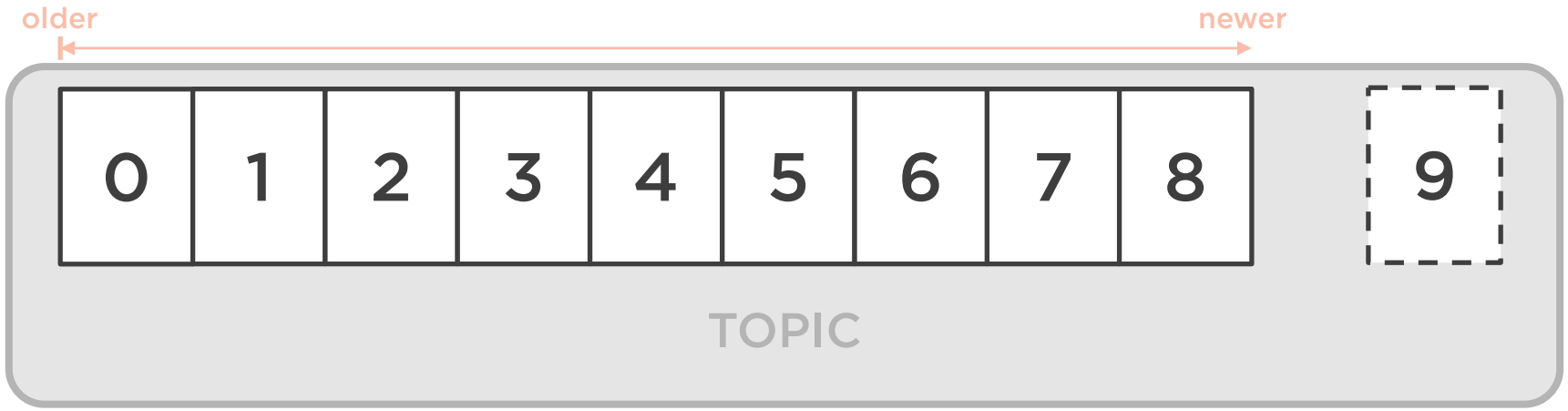
Look for:

- Built-in Producer and Consumer clients
- The ordering of the messages within a topic

Don’t get too caught up on:

- The command line parameters and options

Does Look This Look Familiar?



- ✓ Append-only
- ✓ Ordered sequence (by time)
- ✓ Immutable facts as events

Transaction or Commit Logs

Source of truth

Physically stored and maintained

Higher-order data structures derive from the log

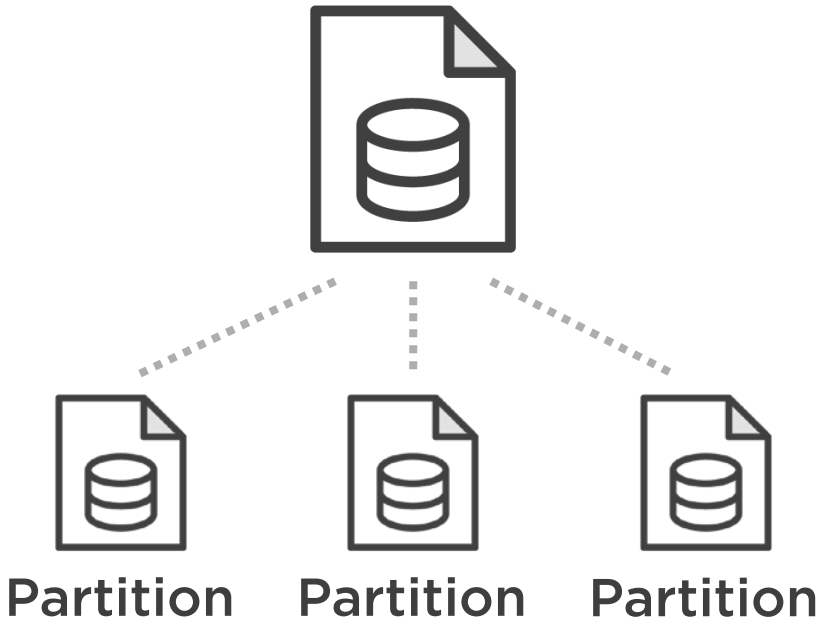
- Tables, indexes, views, etc.

Point of recovery

Basis for replication and distribution

Apache Kafka is publish-subscribe messaging rethought as a **distributed commit log**.

Kafka Partitions



(Partition == Log)

Each topic has one or more partitions

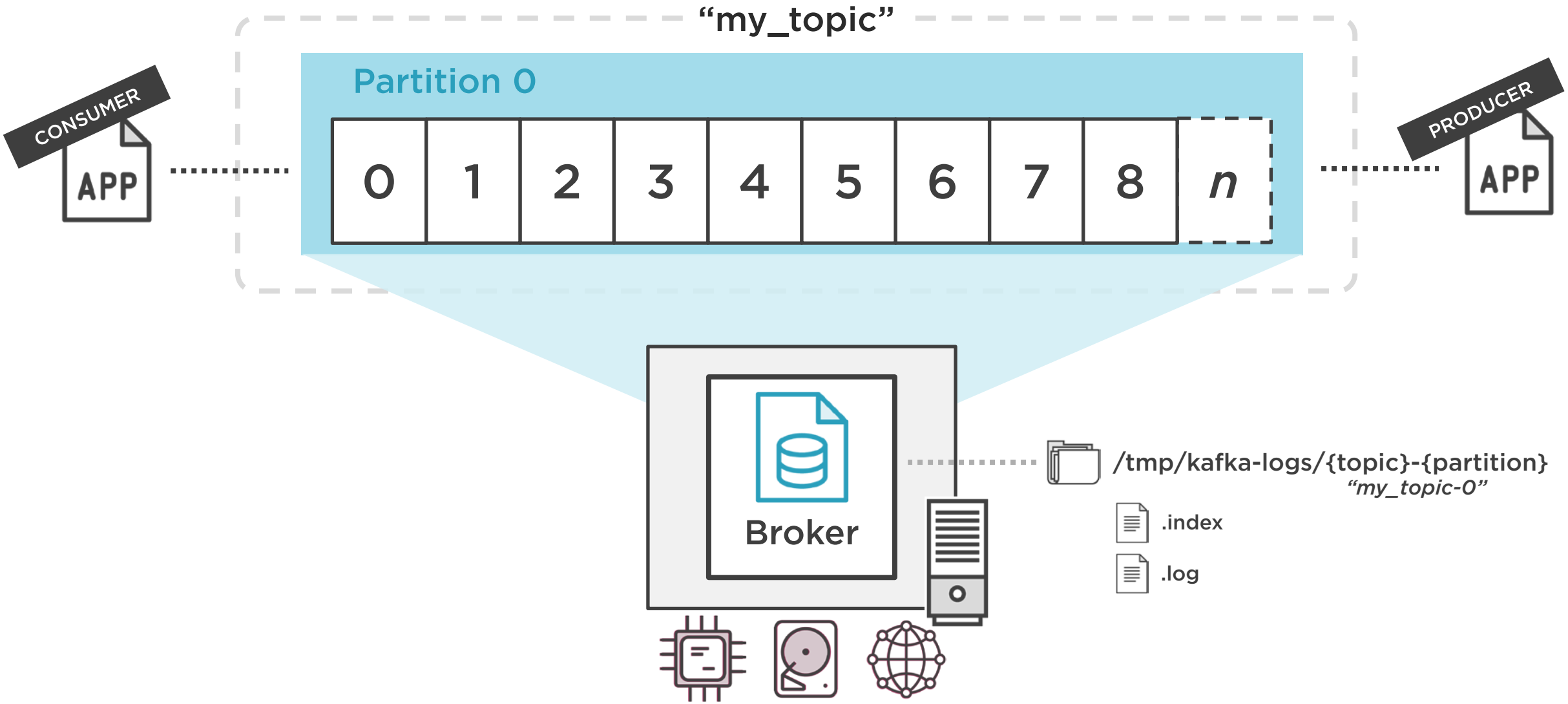
A partition is the basis for which Kafka can:

- Scale
- Become fault-tolerant
- Achieve higher levels of throughput

Each partition is maintained on at least one or more Brokers

Creating a Topic: Single Partition

```
~$ bin/kafka-topics.sh --create --topic my_topic \
> --zookeeper localhost:2181 \
> --partitions 1 \
> --replication-factor 1
```

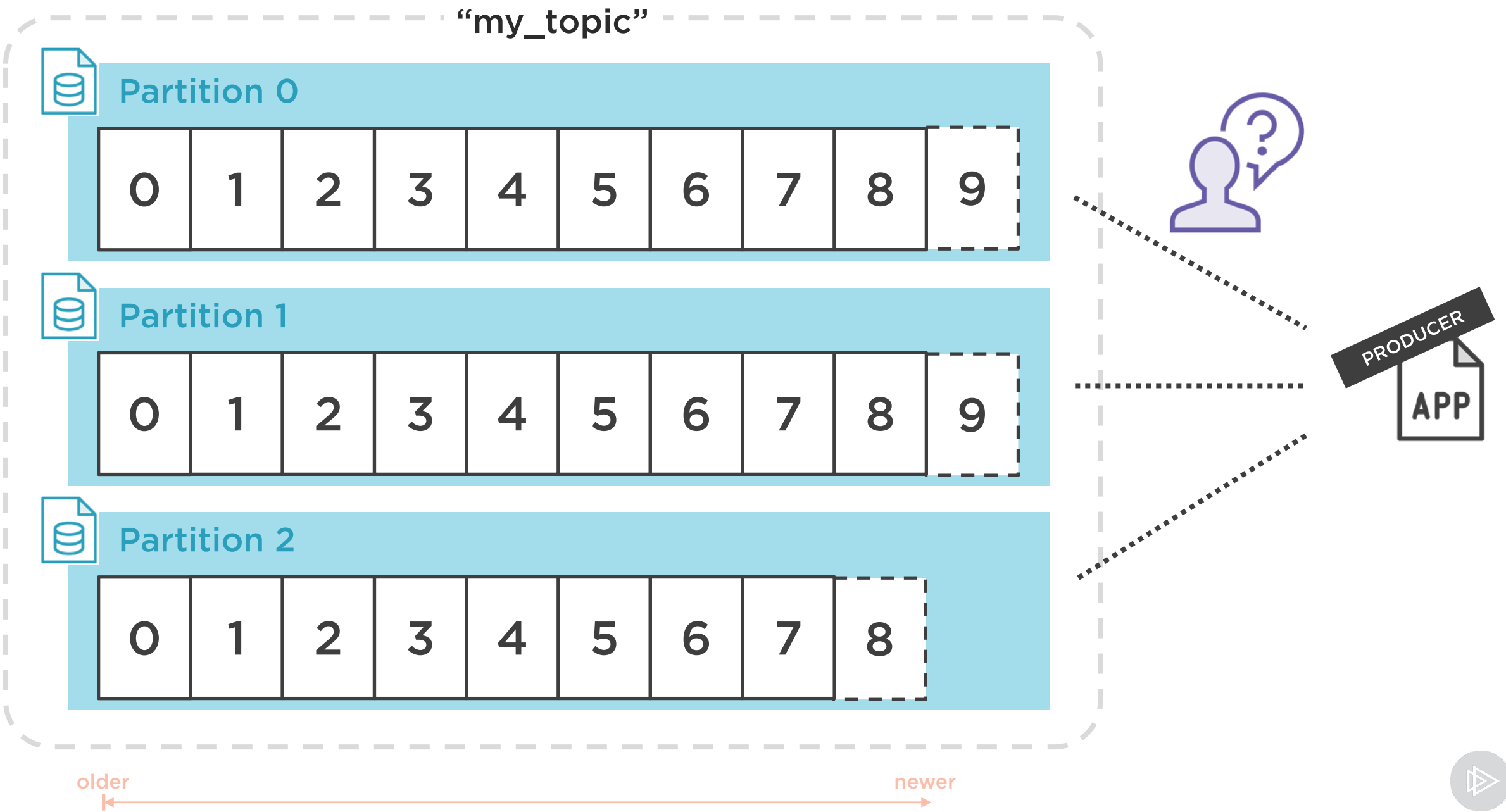


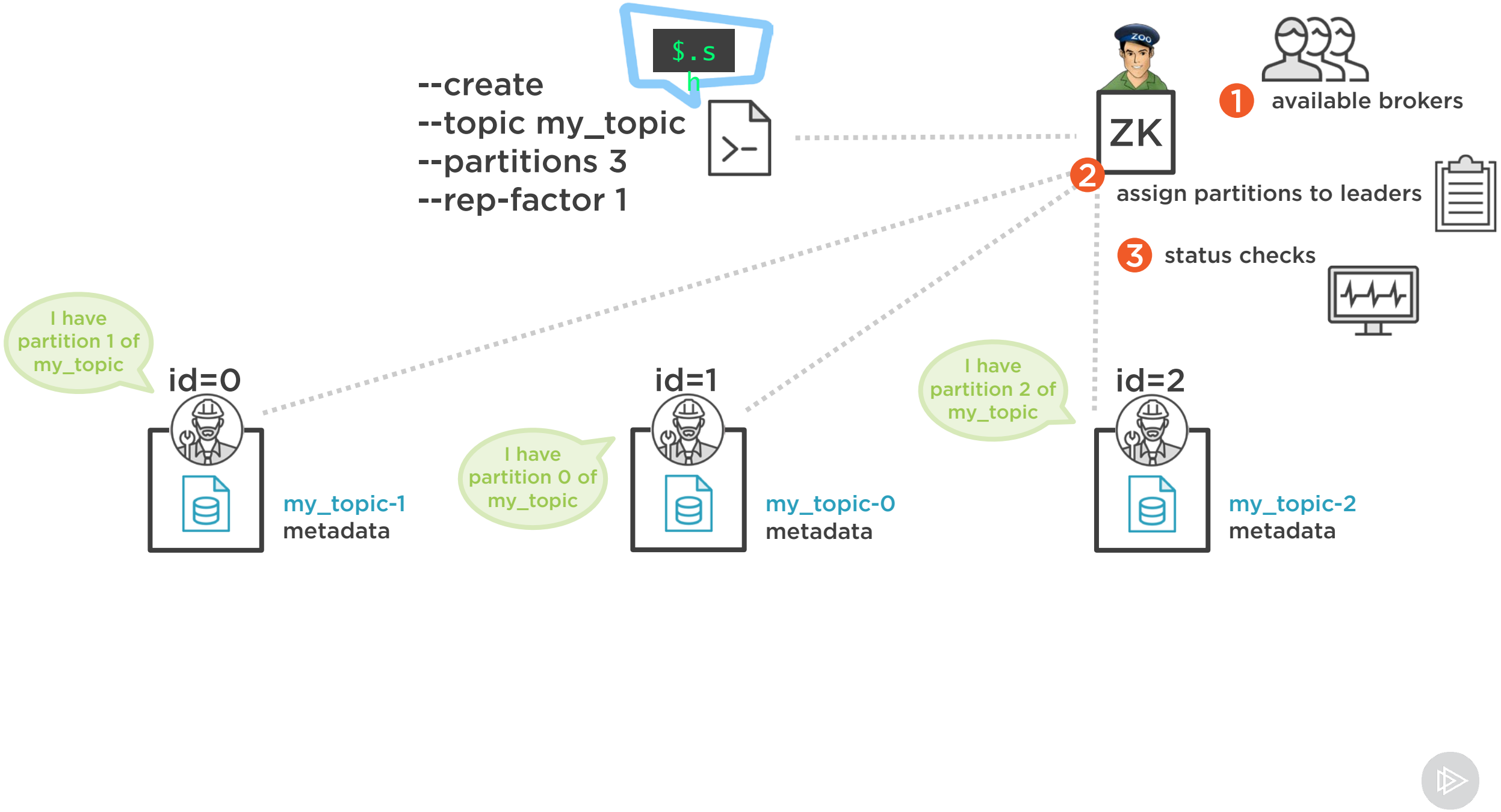
Each partition must fit entirely on one machine.

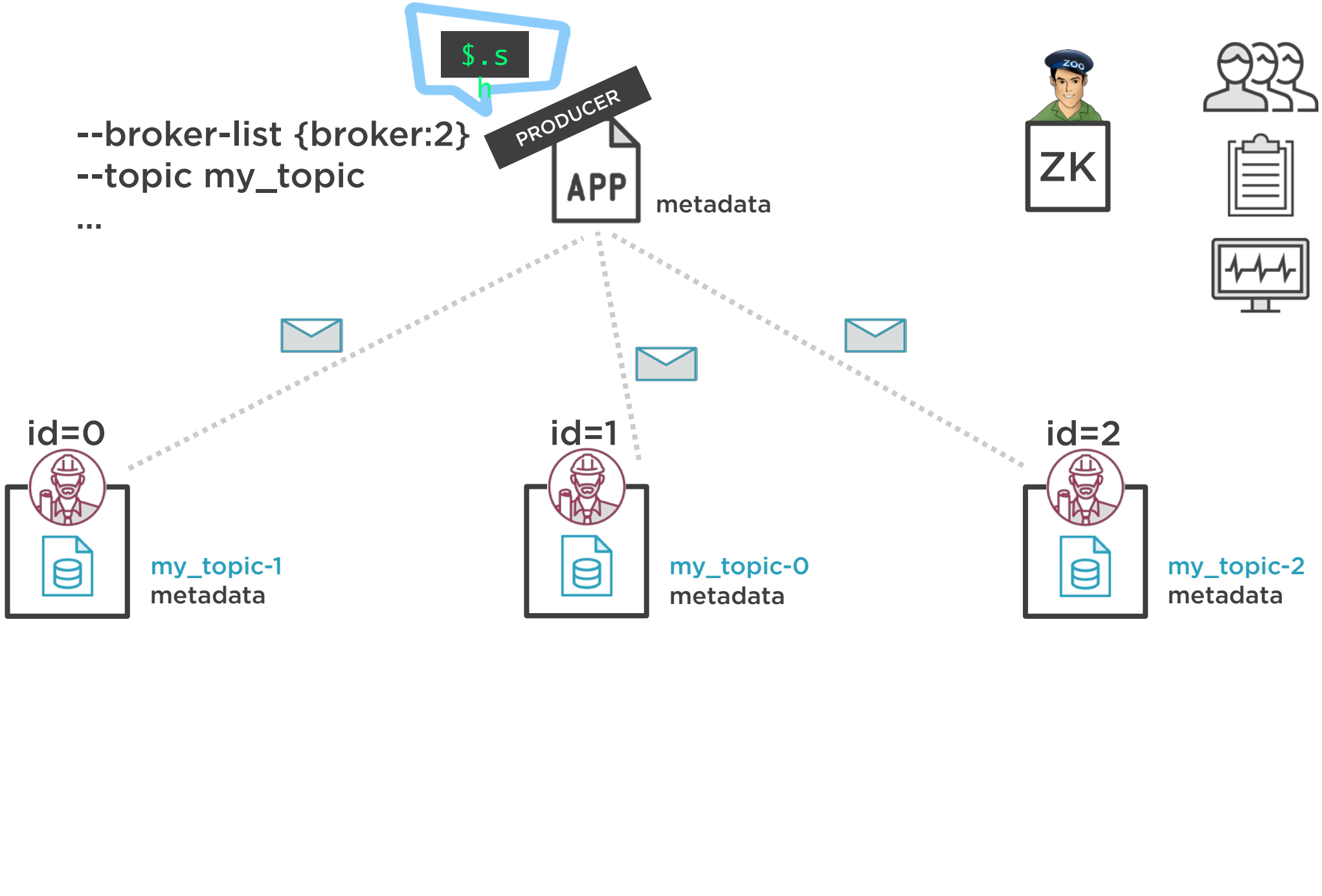
In general, the scalability of Apache Kafka is determined by the number of partitions being managed by multiple broker nodes.

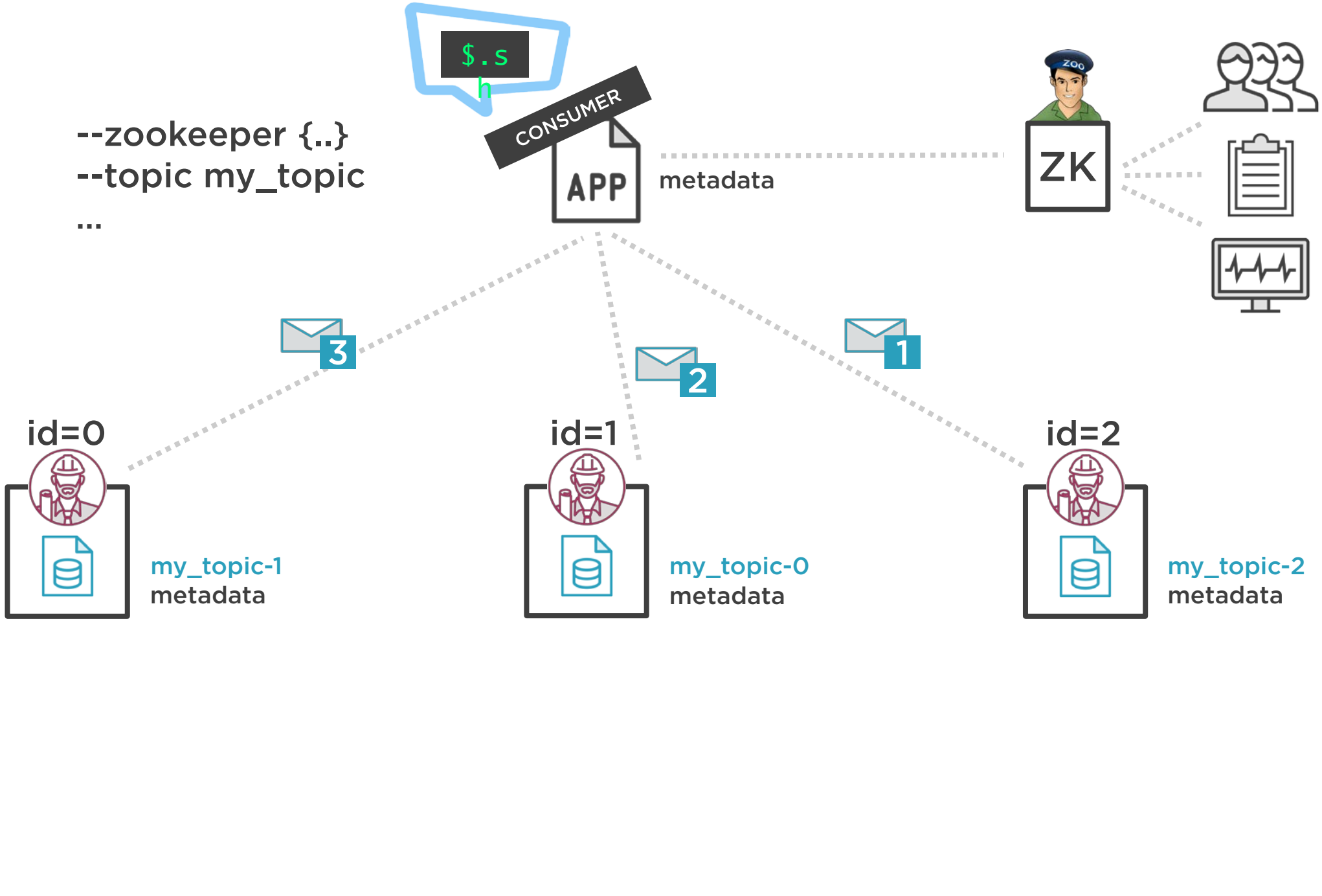
Creating a Topic: Multiple Partitions

```
~$ bin/kafka-topics.sh --create --topic my_topic \
> --zookeeper localhost:2181 \
> --partitions 3 \
> --replication-factor 1
```


"my_topic"
Partition 0
0 1 2 3 4 5 6 7 8 9
Partition 1
0 1 2 3 4 5 6 7 8 9
Partition 2
0 1 2 3 4 5 6 7 8
PRODUCER
APP
older
newer


$.s
h
--create
--topic my_topic
--partitions 3
--rep-factor 1
ZK
1 available brokers
2 assign partitions to leaders
3 status checks
I have partition 1 of my_topic
id=0
my_topic-1 metadata
I have partition 0 of my_topic
id=1
my_topic-0 metadata
I have partition 2 of my_topic
id=2
my_topic-2 metadata


$.s
h
PRODUCER
APP
--broker-list {broker:2}
--topic my_topic
...
metadata
ZK
id=0
id=1
id=2
my_topic-1
metadata
my_topic-0
metadata
my_topic-2
metadata


$.s
h
CONSUMER
APP
--zookeeper {..}
--topic my_topic
...
metadata
ZK
3
2
1
id=0
id=1
id=2
my_topic-1
metadata
my_topic-0
metadata
my_topic-2
metadata

Partitioning Trade-offs

The more partitions the greater the Zookeeper overhead

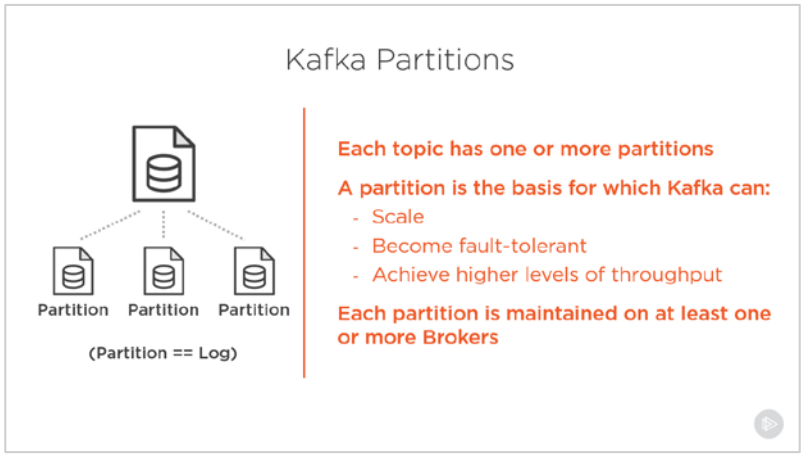
- With large partition numbers ensure proper ZK capacity

Message ordering can become complex

- Single partition for global ordering
- Consumer-handling for ordering

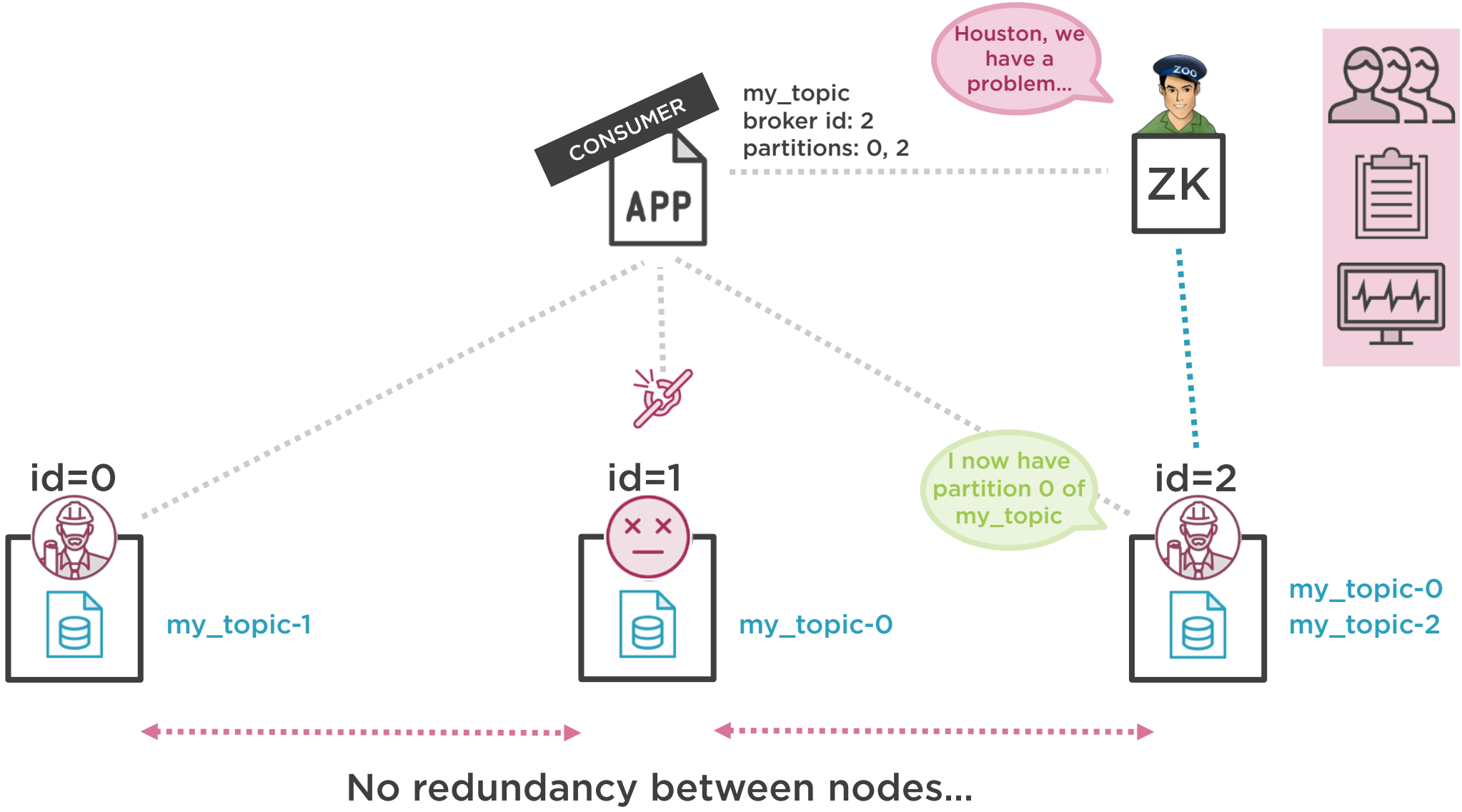
The more partitions the longer the leader fail-over time

Something Is Missing



What about fault-tolerance?

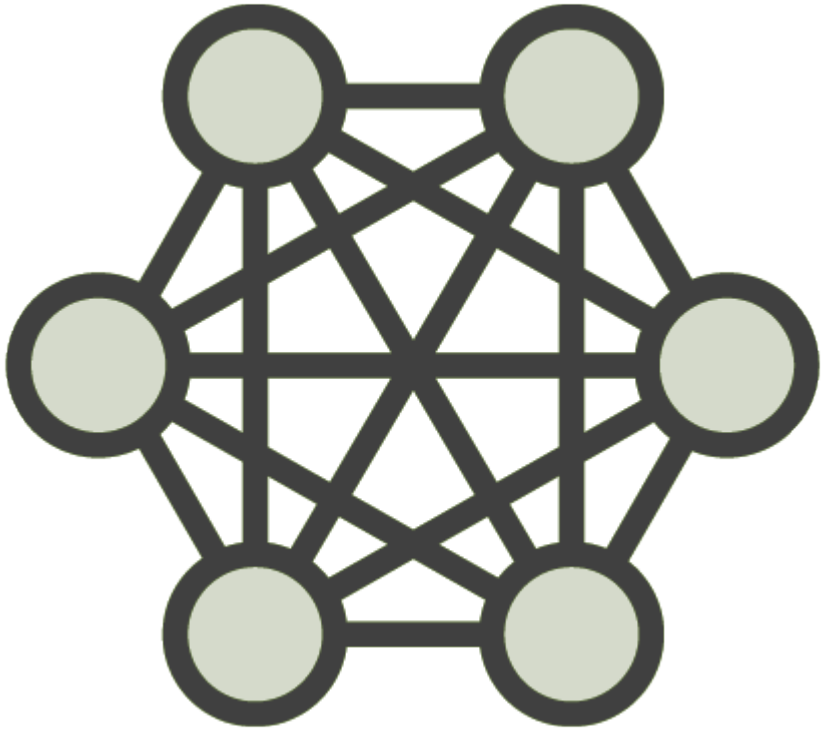
- Broker failure
- Network issue
- Disk failure


Houston, we have a problem...
CONSUMER
APP
my_topic
broker id: 2
partitions: 0, 2
ZK
id=0
id=1
id=2
I now have partition 0 of my_topic
my_topic-1
my_topic-0
my_topic-0
my_topic-2
No redundancy between nodes...

Don’t Forget the Replication Factor

```
~$ bin/kafka-topics.sh --create --topic my_topic \
> --zookeeper localhost:2181 \
> --partitions 3 \
> --replication-factor 1
```

Replication Factor



Reliable work distribution

- Redundancy of messages
- Cluster resiliency
- Fault-tolerance

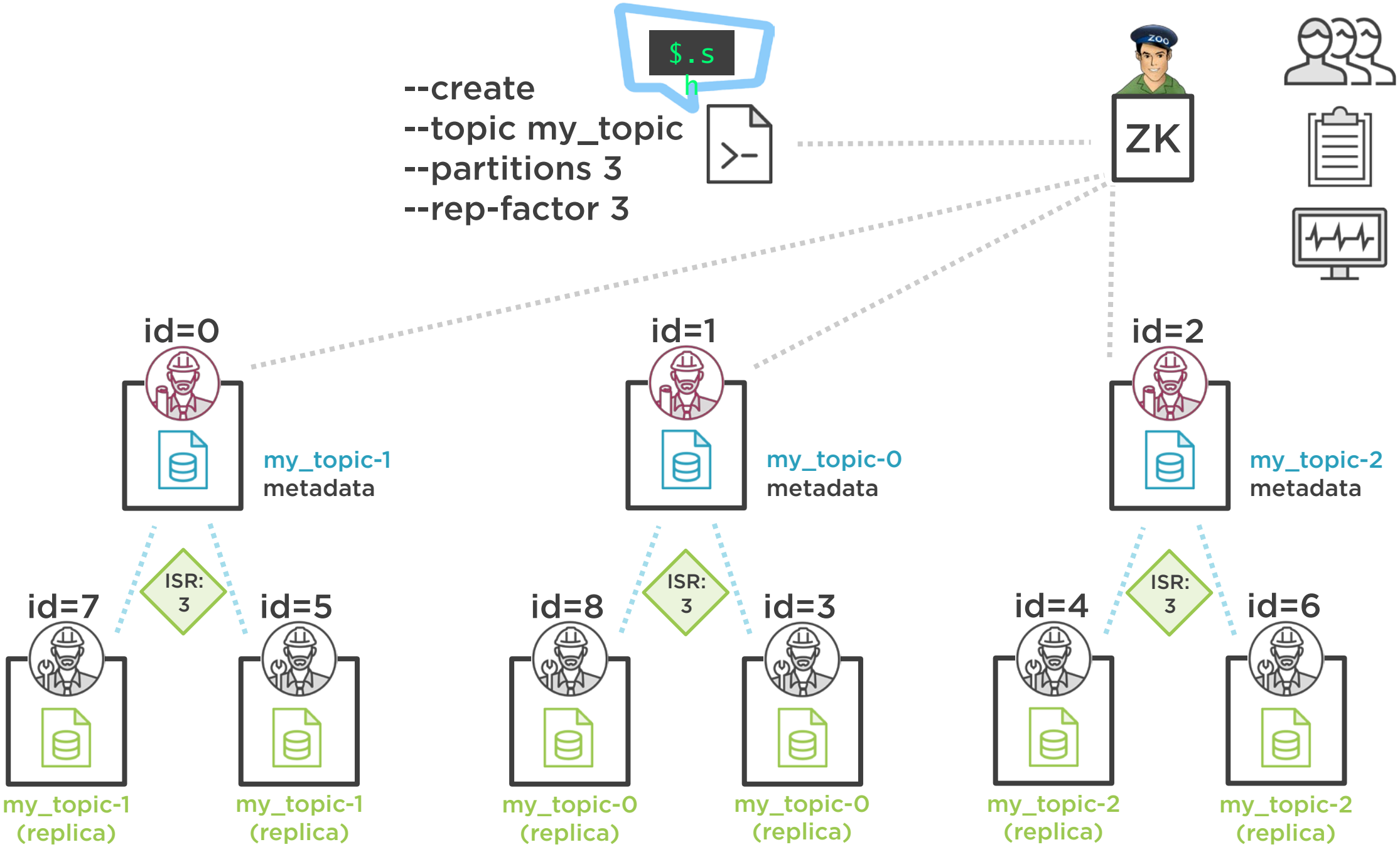
Guarantees

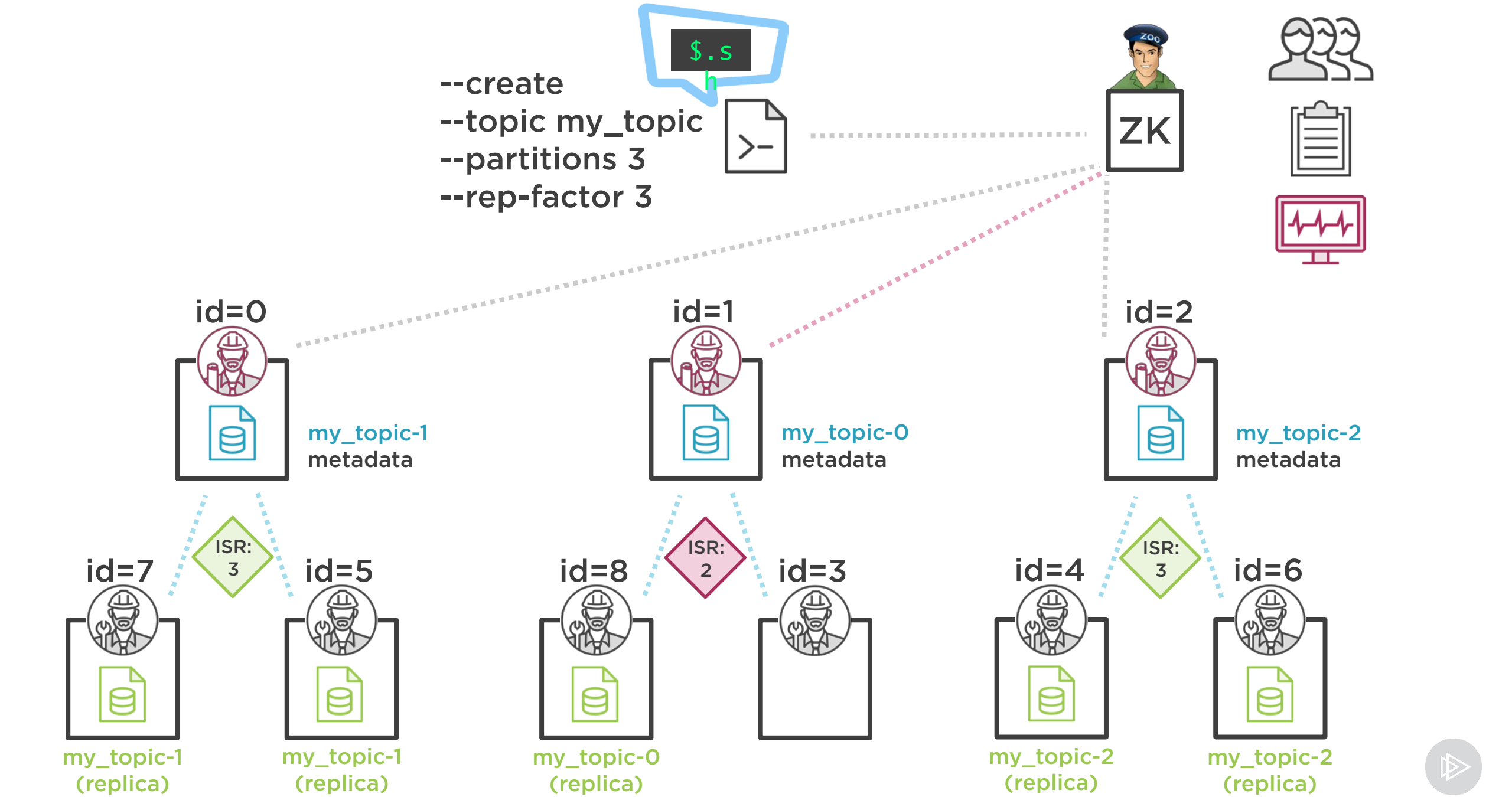
- N-1 broker failure tolerance
- 2 or 3 minimum

Configured on a per-topic basis

Multiple Replica Sets

```
~$ bin/kafka-topics.sh --create --topic my_topic \
> --zookeeper localhost:2181 \
> --partitions 3 \
> --replication-factor 3
```

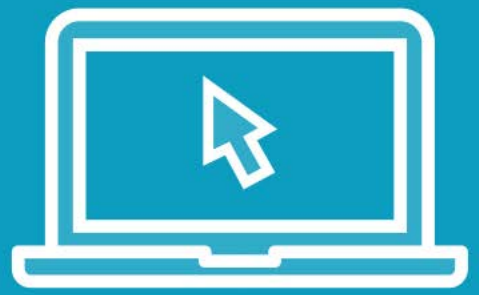

$.s
h
--create
--topic my_topic
--partitions 3
--rep-factor 3
ZK
id=0
my_topic-1
metadata
id=1
my_topic-0
metadata
id=2
my_topic-2
metadata
ISR:
3
ISR:
3
ISR:
3
id=7
id=5
id=8
id=3
id=4
id=6
my_topic-1
(replica)
my_topic-1
(replica)
my_topic-0
(replica)
my_topic-0
(replica)
my_topic-2
(replica)
my_topic-2
(replica)


$.s
h
--create
--topic my_topic
--partitions 3
--rep-factor 3
ZK
id=0
my_topic-1
metadata
id=1
my_topic-0
metadata
id=2
my_topic-2
metadata
ISR:
3
ISR:
2
ISR:
3
id=7
id=5
id=8
id=3
id=4
id=6
my_topic-1
(replica)
my_topic-1
(replica)
my_topic-0
(replica)
my_topic-2
(replica)
my_topic-2
(replica)

Viewing Topic State

```
~$ bin/kafka-topics.sh --describe --topic my_topic \
> --zookeeper localhost:2181
```

Demo



Multi-broker Kafka Setup

Single Partition Topic

Replication Factor of 3

Look for:

- Using the --describe command
- Failure handling
- Continued operation

Summary

Detailed explanation and view:

- Topics and Partitions
- Broker partition management and behavior

Aligned with distributed systems principles

- Leader election of partitions
- Work distribution and failover

Kafka in action

- Demos
- Configuration

Foundation upon which to dive deeper into Producers and Consumers