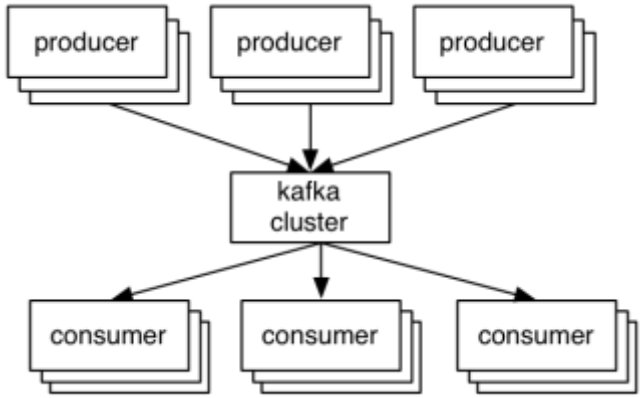
Kafka Core Concepts

Kafka core concepts

- **Records** have a **key (optional), value**, and **timestamp**.
- **Topic** a stream of records (e.g., orders), feed name
  - **Log** topic storage on disk
  - **Partition** / segments (parts of topic log)
- **Producer** API to produce streams of records
- **Consumer** API to consume streams of records

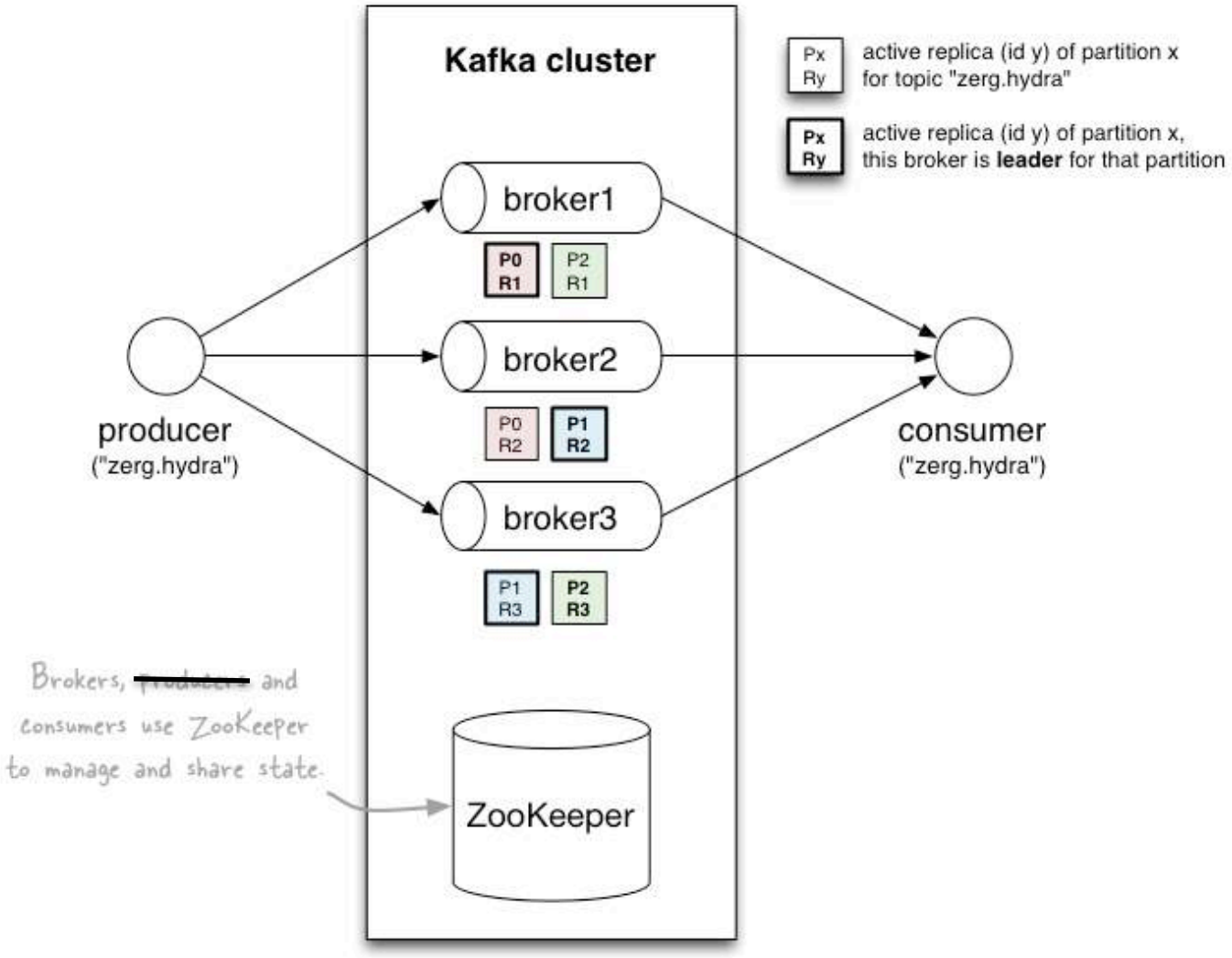
A first look

- The who is who
  - **Producers** write data to **brokers**.
  - **Consumers** read data from **brokers**.
  - All this is distributed.



- The data
  - Data is stored in **topics**.
  - **Topics** are split into **partitions**, which are **replicated**.

A first look

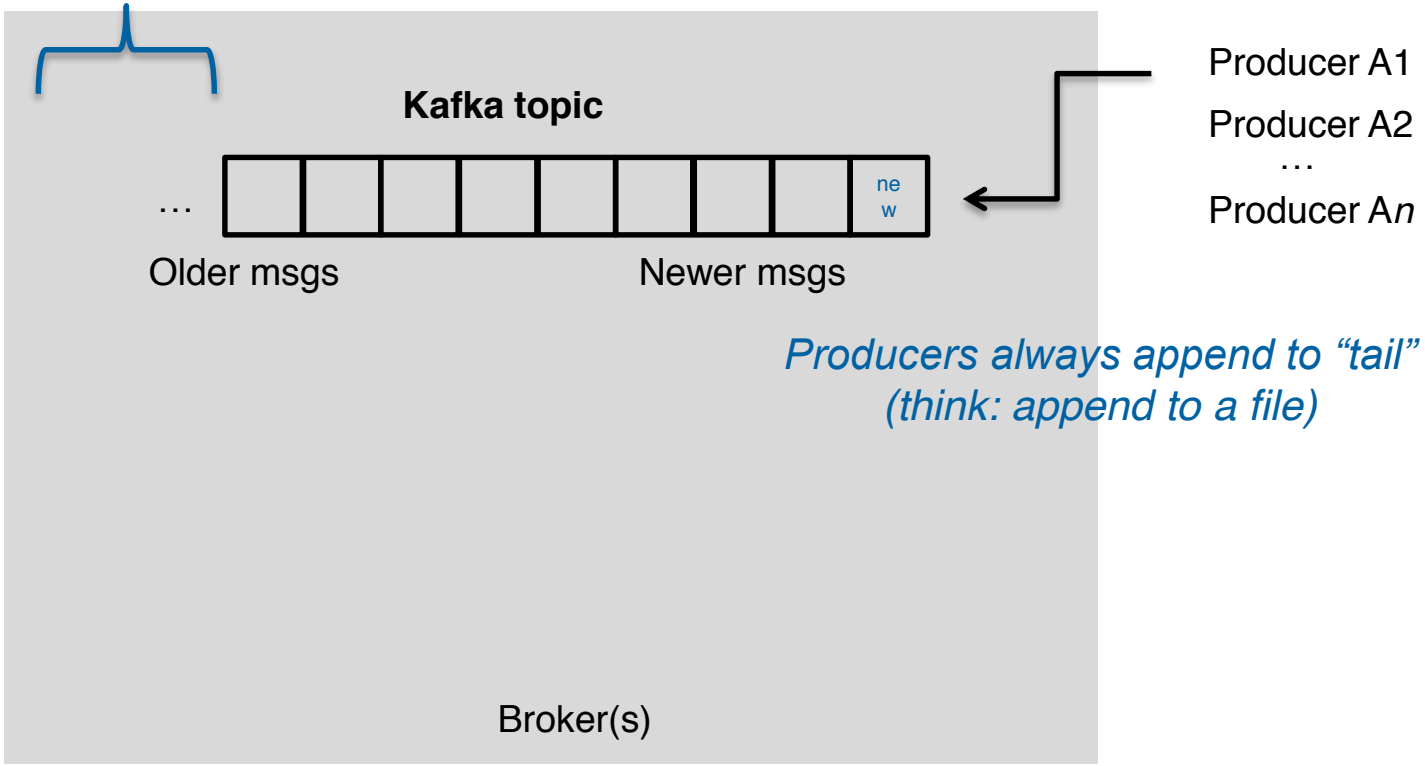


http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/

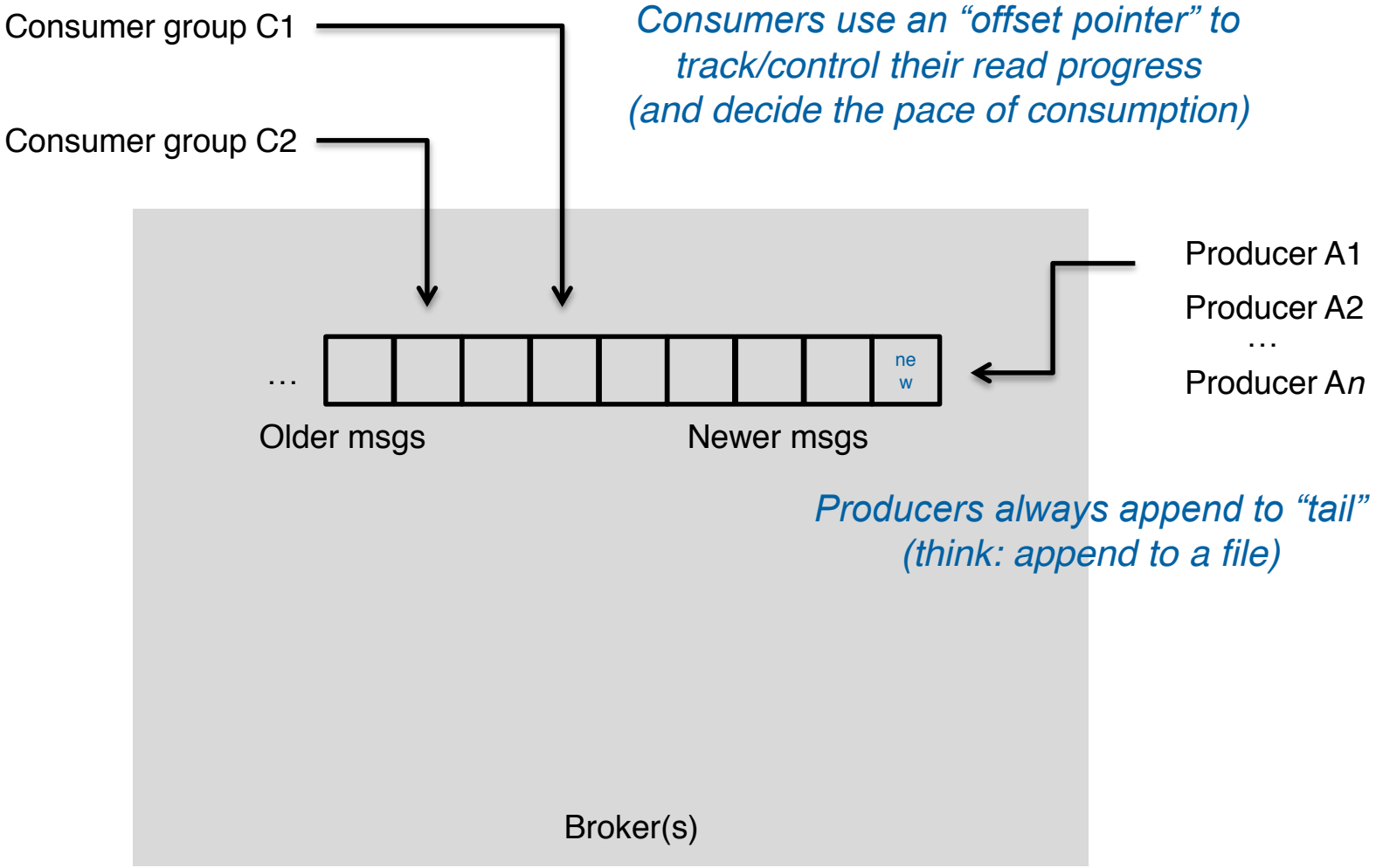
Topics

- **Topic:** feed name to which messages are published
  - Example: “zerg.hydra”

Kafka prunes “head” based on age or max size or “key”



Topics


Consumer group C1
Consumer group C2
Consumers use an "offset pointer" to track/control their read progress (and decide the pace of consumption)
Producer A1
Producer A2
…
Producer An
…
new
Older msgs
Newer msgs
Producers always append to "tail" (think: append to a file)
Broker(s)

Topics

- Creating a topic
  - CLI

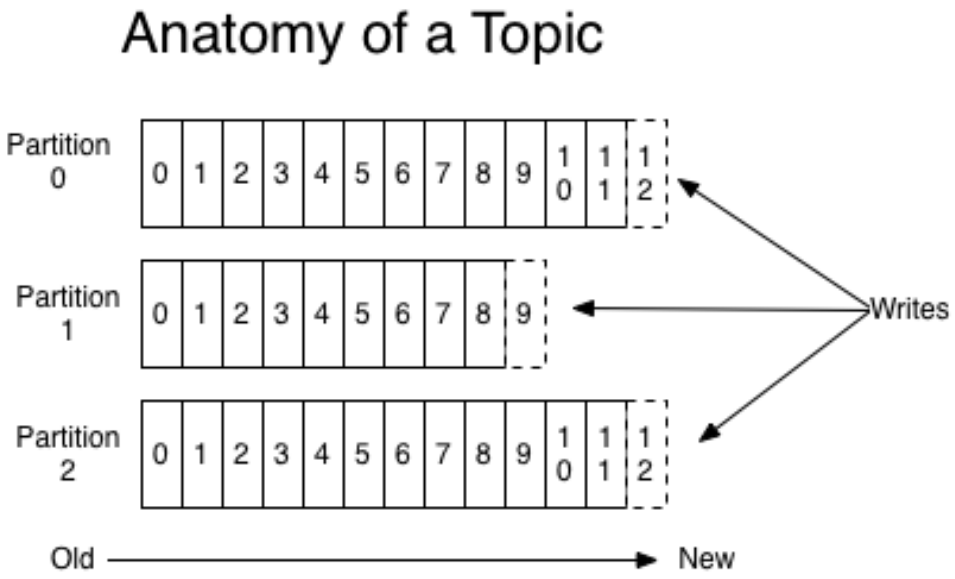
```
$ kafka-topics.sh --zookeeper zookeeper1:2181 --create --topic zerg.hydra \
    --partitions 3 --replication-factor 2 \
    --config x=y
```

  - API
    https://github.com/miguno/kafka-storm-starter/blob/develop/src/main/scala/com/miguno/kafkastorm/storm/KafkaStormDemo.scala
  - Auto-create via `auto.create.topics.enable = true`

- Modifying a topic
  - https://kafka.apache.org/documentation.html#basic_ops_modify_topic
- Deleting a topic: DON'T in 0.8.1.x!

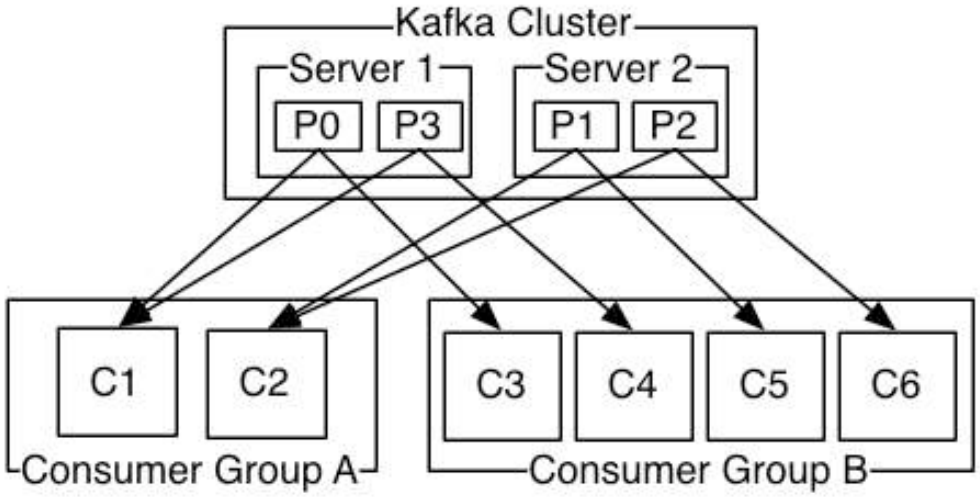
Partitions

- A topic consists of **partitions.**
- Partition: **ordered + immutable** sequence of messages that is continually appended to



Partitions

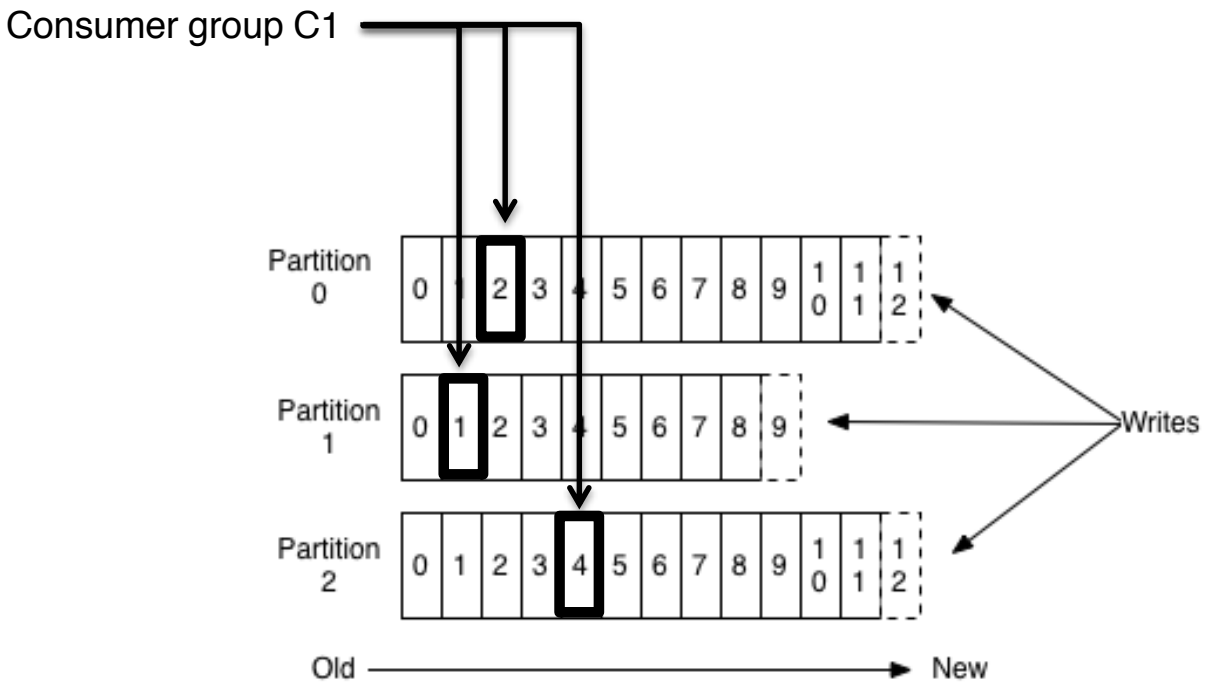
- #partitions of a topic is configurable
- #partitions determines **max** consumer (group) parallelism
  - Cf. parallelism of Storm's KafkaSpout via `builder.setSpout(,,N)`



  - Consumer group A, with 2 consumers, reads from a 4-partition topic
  - Consumer group B, with 4 consumers, reads from the same topic

Partition offsets

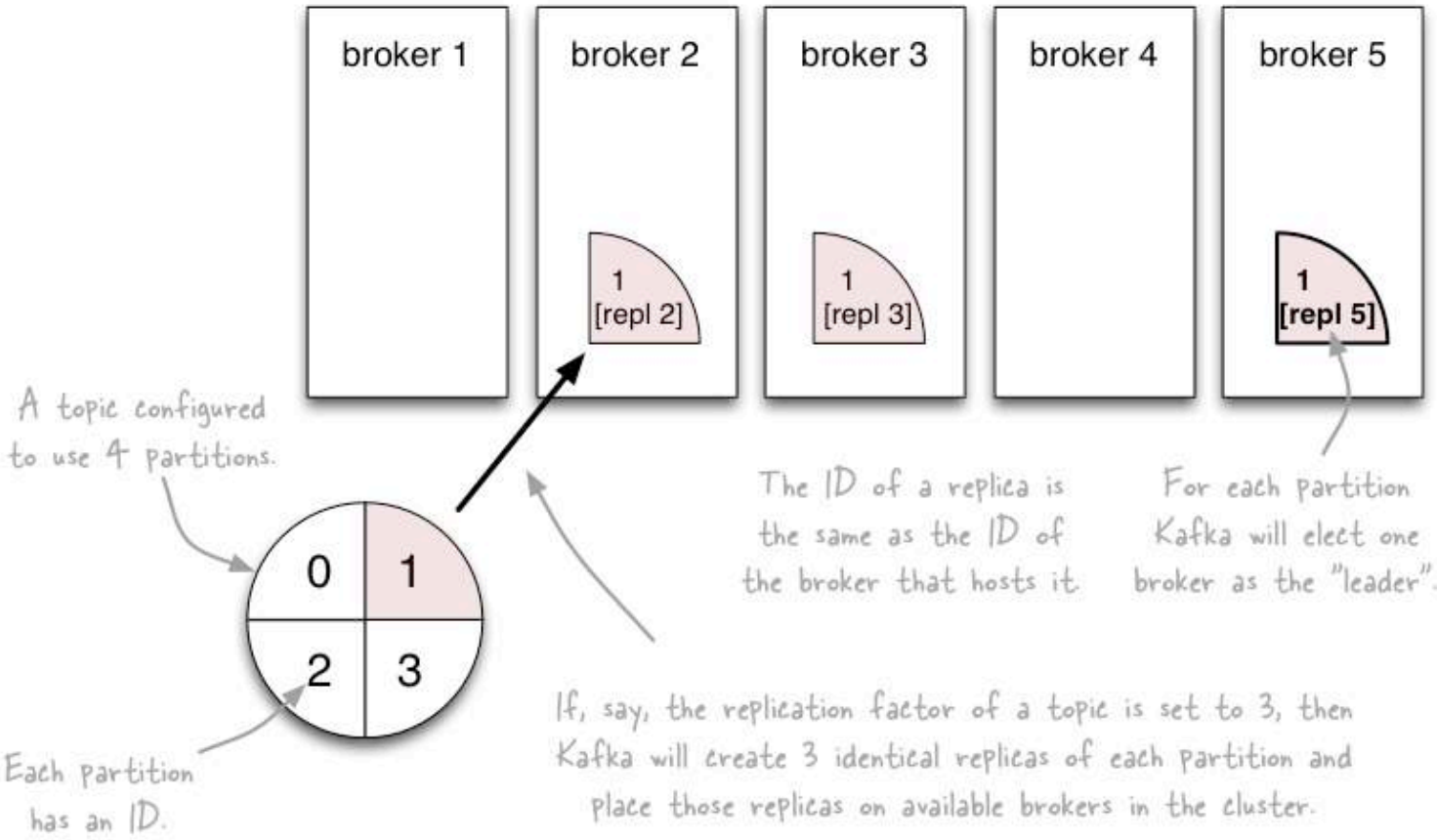
- **Offset**: messages in the partitions are each assigned a unique (per partition) and sequential id called the *offset*
  - Consumers track their pointers via *(offset, partition, topic)* tuples



Replicas of a partition

- **Replicas:** “backups” of a partition
  - They exist solely to prevent data loss.
  - Replicas are never read from, never written to.
    - They do NOT help to increase producer or consumer parallelism!
  - Kafka tolerates *(numReplicas - 1)* dead brokers before losing data
    - LinkedIn: numReplicas == 2 → 1 broker can die

Topics vs. Partitions vs. Replicas



http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/

Inspecting the current state of a topic

- `--describe` the topic

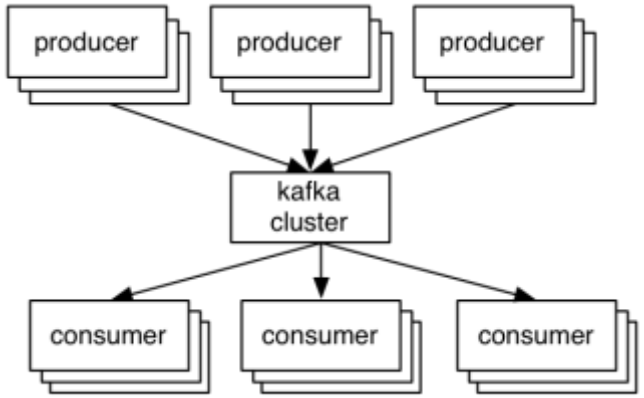
```
$ kafka-topics.sh --zookeeper zookeeper1:2181 --describe --topic zerg.hydra
Topic:zerg2.hydra PartitionCount:3 ReplicationFactor:2 Configs:
  Topic: zerg2.hydra Partition: 0 Leader: 1 Replicas: 1,0 Isr: 1,0
  Topic: zerg2.hydra Partition: 1 Leader: 0 Replicas: 0,1 Isr: 0,1
  Topic: zerg2.hydra Partition: 2 Leader: 1 Replicas: 1,0 Isr: 1,0
```

  - Leader: brokerID of the currently elected leader broker
    - Replica ID's = broker ID's
  - ISR = "in-sync replica", replicas that are in sync with the leader

- In this example:
  - Broker 0 is leader for partition 1.
  - Broker 1 is leader for partitions 0 and 2.
  - All replicas are in-sync with their respective leader partitions.

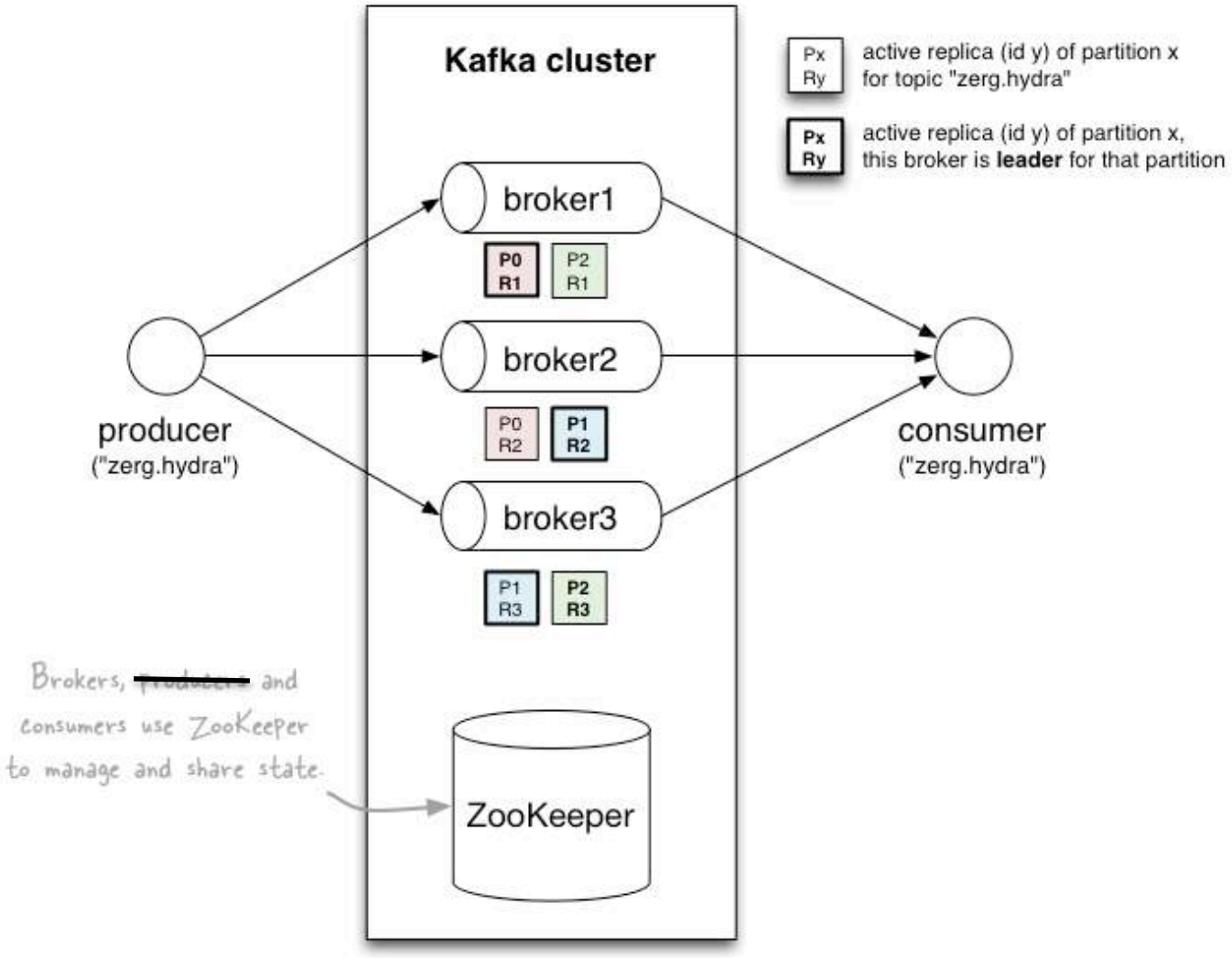
Let's recap

- The who is who
  - **Producers** write data to **brokers**.
  - **Consumers** read data from **brokers**.
  - All this is distributed.



- The data
  - Data is stored in **topics**.
  - **Topics** are split into **partitions** which are **replicated**.

Putting it all together



http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/