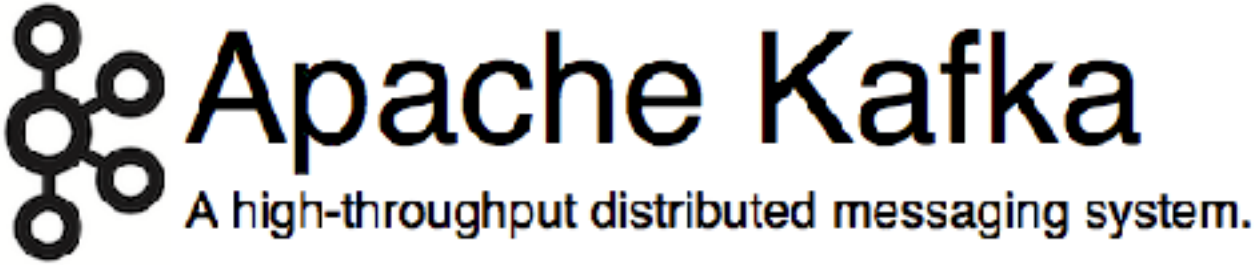
APACHE KAFKA

Kafka?



- http://kafka.apache.org/
- Originated at LinkedIn, open sourced in early 2011
- Implemented in Scala, some Java
- 9 core committers, plus ~ 20 contributors

Kafka?

- LinkedIn's motivation for Kafka was:
  - "A unified platform for handling all the real-time data feeds a large company might have."
- Must haves
  - High throughput to support high volume event feeds.
  - Support real-time processing of these feeds to create new, derived feeds.
  - Support large data backlogs to handle periodic ingestion from offline systems.
  - Support low-latency delivery to handle more traditional messaging use cases.
  - Guarantee fault-tolerance in the presence of machine failures.

Kafka @ LinkedIn, 2014

- Multiple data centers, multiple clusters
  - Mirroring between clusters / data centers
- What type of data is being transported through Kafka?
  - Metrics: operational telemetry data
  - Tracking: everything a LinkedIn.com user does
  - Queuing: between LinkedIn apps, e.g. for sending emails
  - To transport data from LinkedIn's apps to Hadoop, and back

Kafka @ LinkedIn, 2014

- In total ~ 200 billion events/day via Kafka
  - Tens of thousands of data producers, thousands of consumers
  - 7 million events/sec (write), 35 million events/sec (read) <<< may include replicated events
  - But: LinkedIn is not even the largest Kafka user anymore as of 2014

Kafka @ LinkedIn, 2014

"For reference, here are the stats on one of LinkedIn's busiest clusters (at peak):

15 brokers
15,500 partitions (replication factor 2)
400,000 msg/s inbound
70 MB/s inbound
400 MB/s outbound"

Kafka adoption and use cases

- LinkedIn: activity streams, operational metrics, data bus
  - 400 nodes, 18k topics, 220B msg/day (peak 3.2M msg/s), May 2014
- Netflix: real-time monitoring and event processing
- Twitter: as part of their Storm real-time data pipelines
- Spotify: log delivery (from 4h down to 10s), Hadoop
- Loggly: log collection and processing
- Mozilla: telemetry data
- Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, Uber, …

How fast is Kafka?

- “Up to 2 million writes/sec on 3 cheap machines”
  - Using 3 producers on 3 different machines, 3x async replication
    - Only 1 producer/machine because NIC already saturated
- Sustained throughput as stored data grows
  - Slightly different test config than 2M writes/sec above.
- Test setup
  - Kafka trunk as of April 2013, but 0.8.1+ should be similar.
  - 3 machines: 6-core Intel Xeon 2.5 GHz, 32GB RAM, 6x 7200rpm SATA, 1GigE

Why is Kafka so fast?

- Fast writes:
  - While Kafka persists all data to disk, essentially all writes go to the page cache of OS, i.e. RAM.
  - Cf. hardware specs and OS tuning (we cover this later)
- Fast reads:
  - Very efficient to transfer data from page cache to a network socket
  - Linux: sendfile() system call
- Combination of the two = fast Kafka!
  - Example (Operations): On a Kafka cluster where the consumers are mostly caught up you will see no read activity on the disks as they will be serving data entirely from cache.

Why is Kafka so fast?

- Example: Loggly.com, who run Kafka & Co. on Amazon AWS
  - “99.99999% of the time our data is coming from disk cache and RAM; only very rarely do we hit the disk.”
  - “One of our consumer groups (8 threads) which maps a log to a customer can process about 200,000 events per second draining from 192 partitions spread across 3 brokers.”
    - Brokers run on m2.xlarge Amazon EC2 instances backed by provisioned IOPS

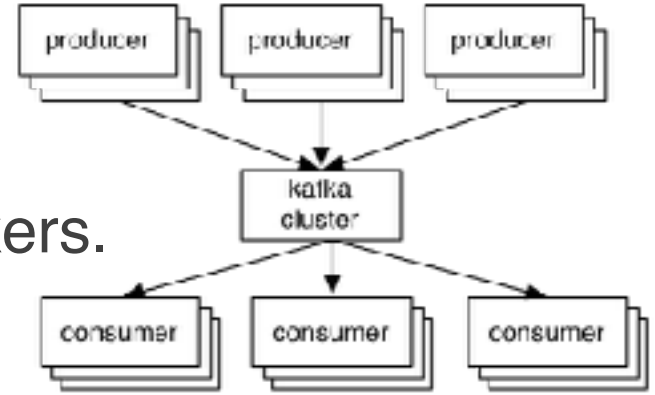
Kafka + X for processing the data?

- Kafka + Storm often used in combination, e.g. Twitter
- Kafka + custom
  - “Normal” Java multi-threaded setups
  - Akka actors with Scala or Java, e.g. Ooyala
- Recent additions:
  - Samza (since Aug ’13) – also by LinkedIn
  - Spark Streaming, part of Spark (since Feb ’13)
- Kafka + Camus for Kafka->Hadoop ingestion

PART 2: KAFKA CORE CONCEPTS

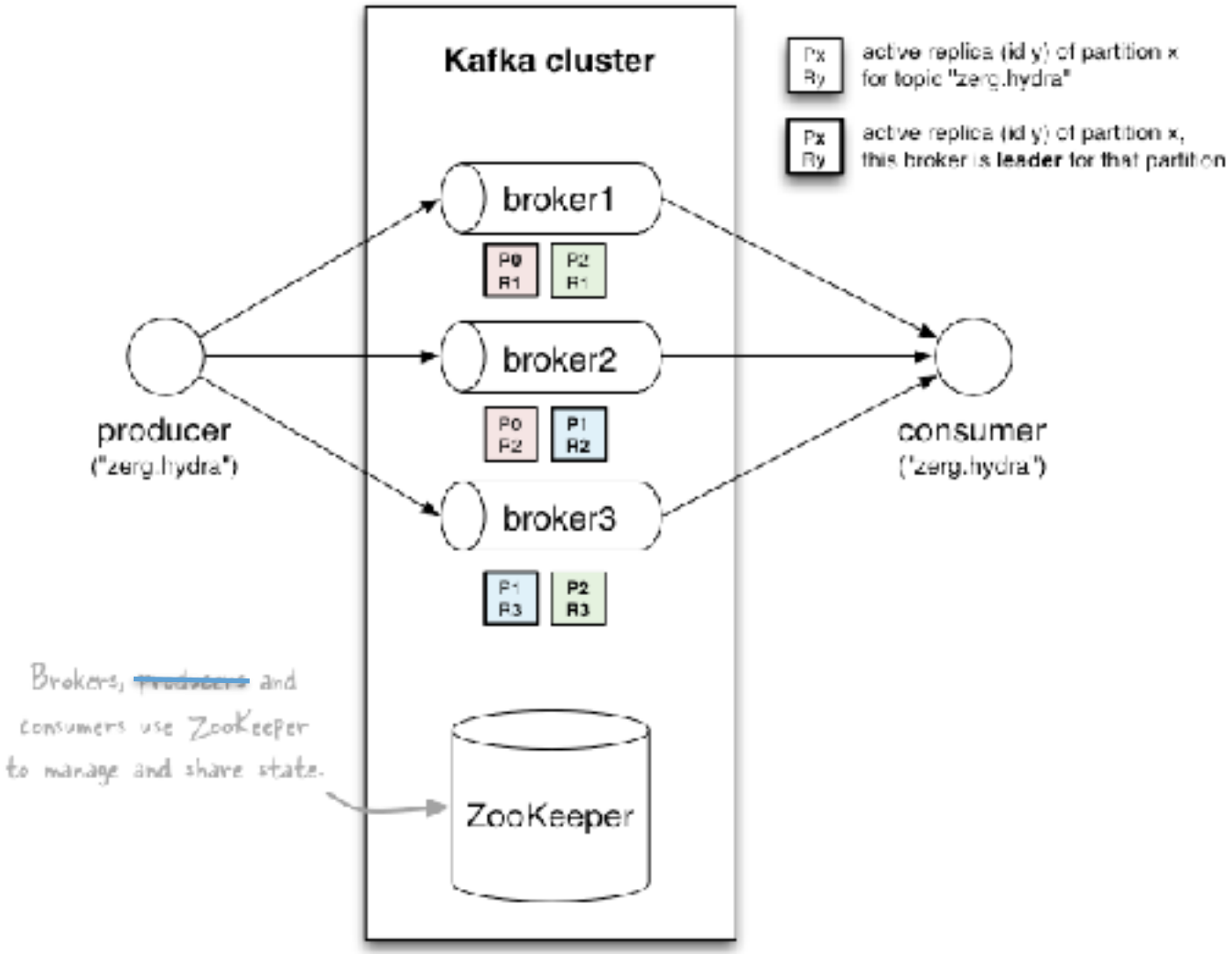
A first look

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



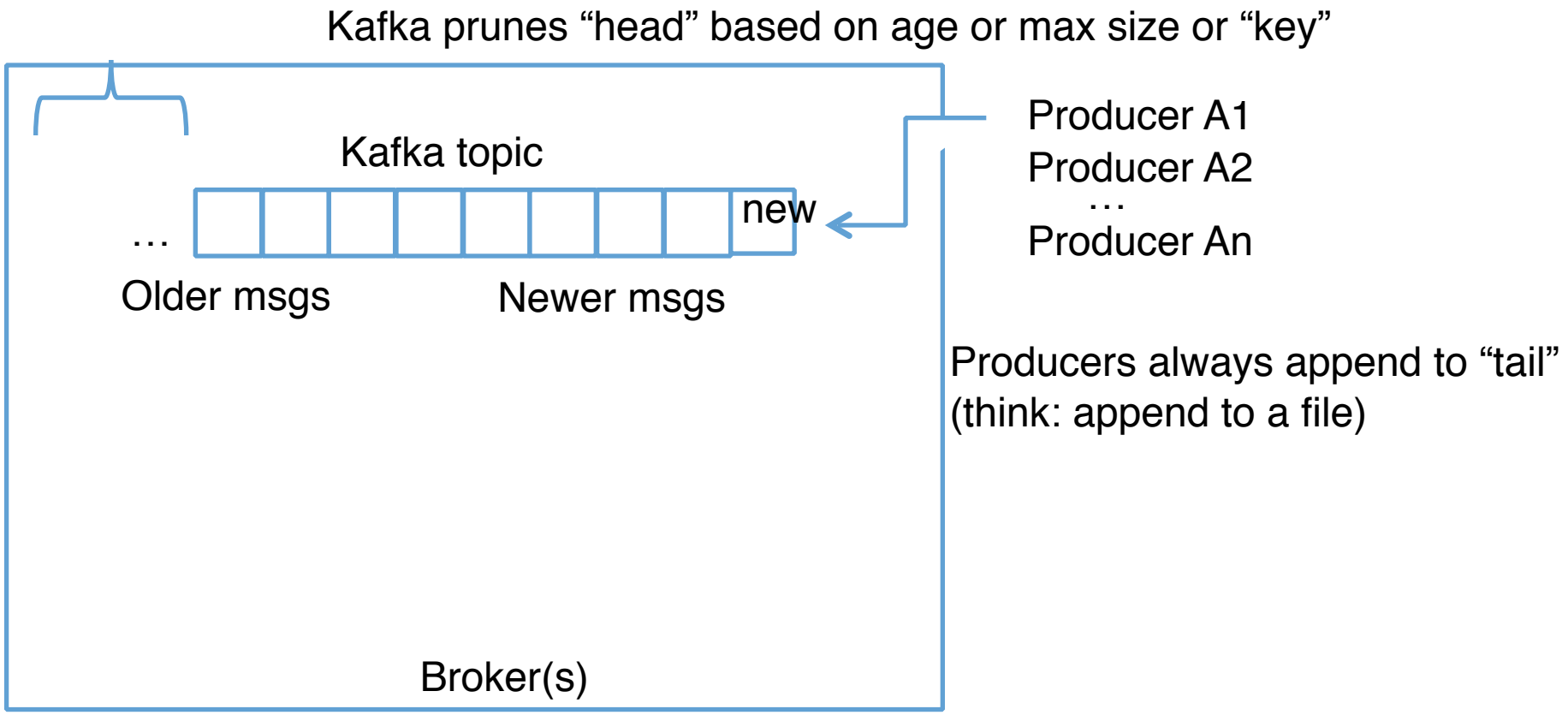
- The data
  - Data is stored in topics.
  - Topics are split into partitions, which are replicated.

A first look

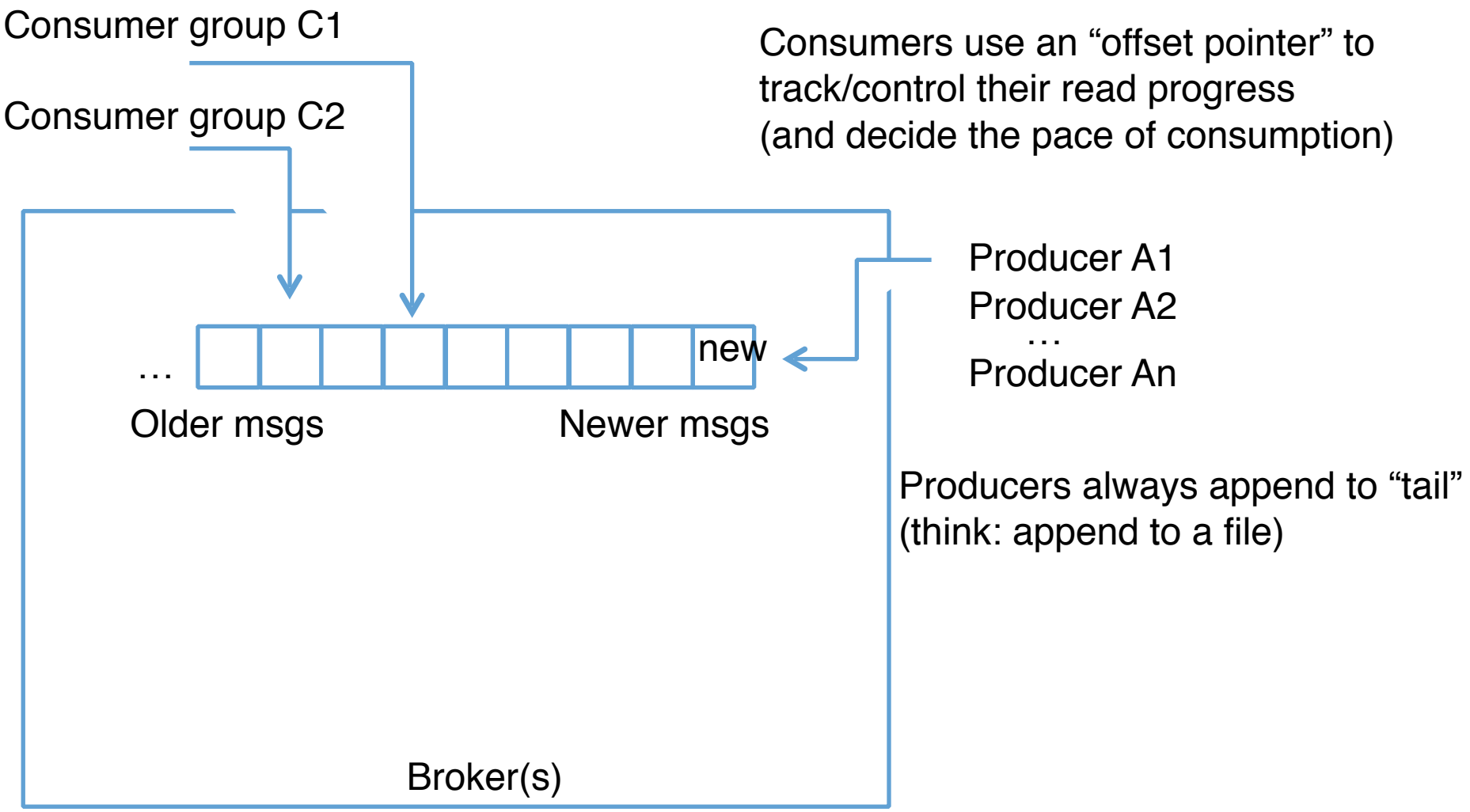

Kafka cluster
Px Ry
active replica (id y) of partition x for topic "zerg.hydra"
Px Ry
active replica (id y) of partition x, this broker is leader for that partition
broker1
P0 R1
P2 R1
broker2
P0 P2
P1 R2
broker3
P1 R3
P2 R3
producer ("zerg.hydra")
consumer ("zerg.hydra")
Brokers, ~~Producers~~ and consumers use ZooKeeper to manage and share state.
ZooKeeper

Topics

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



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

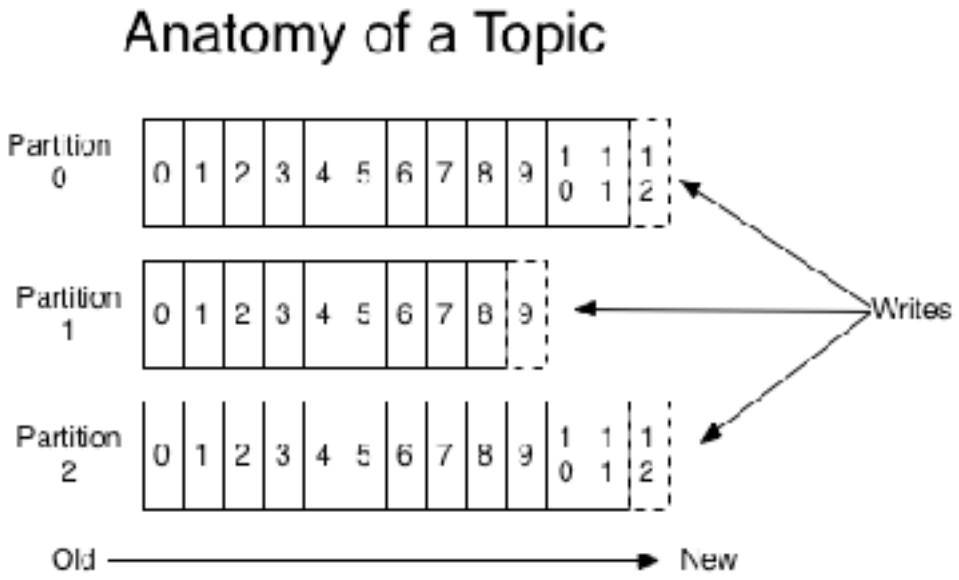
```
$ kafka-topics.sh --zookeeper zookeeper1:2181 --create --topic zerg.hydra
--partitions 3 --replication-factor 2 \
    --config x=y
```

  - 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

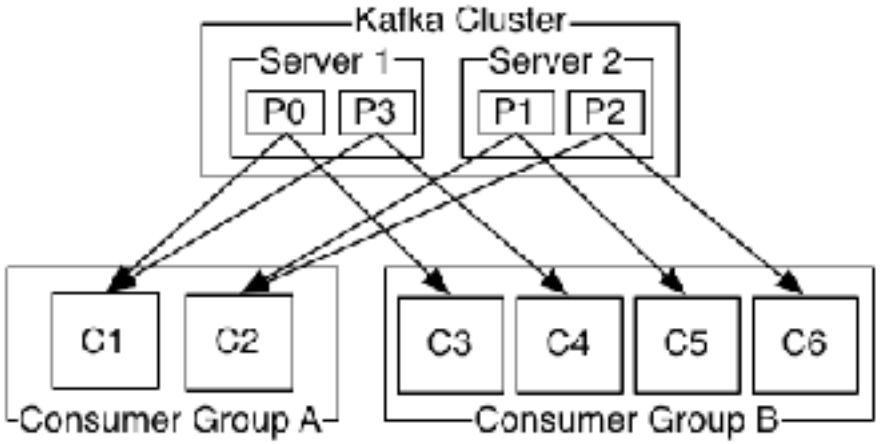
Partitions

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



Partitions

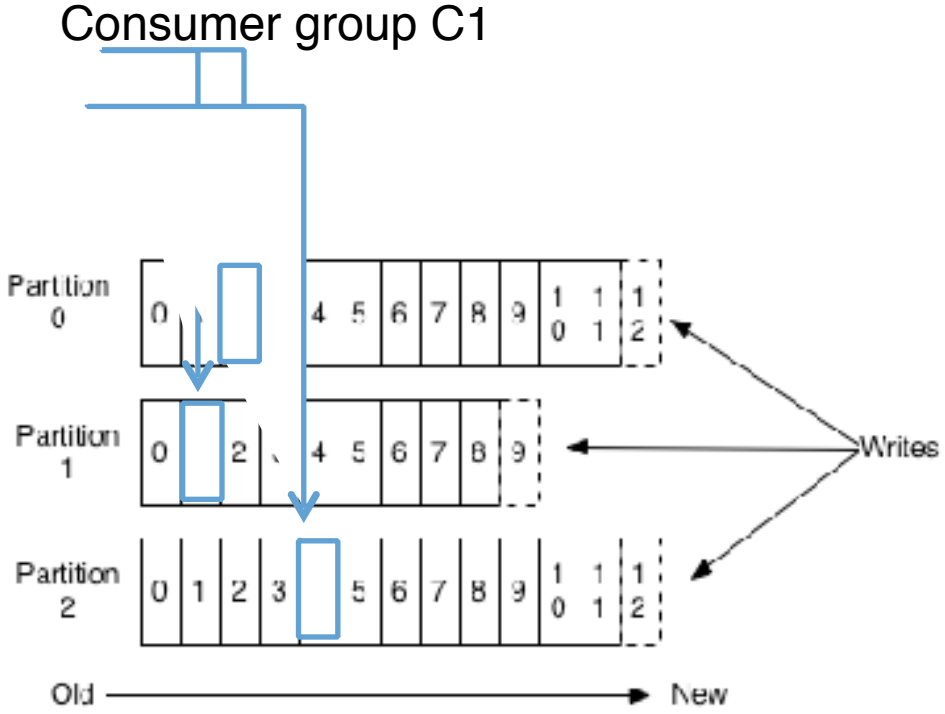
- #partitions of a topic is configurable
- #partitions determines max consumer (group) parallelism
  - Cf. parallelism of Storm's KafkaSpout via builder.setSpout(,,N)



  - 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

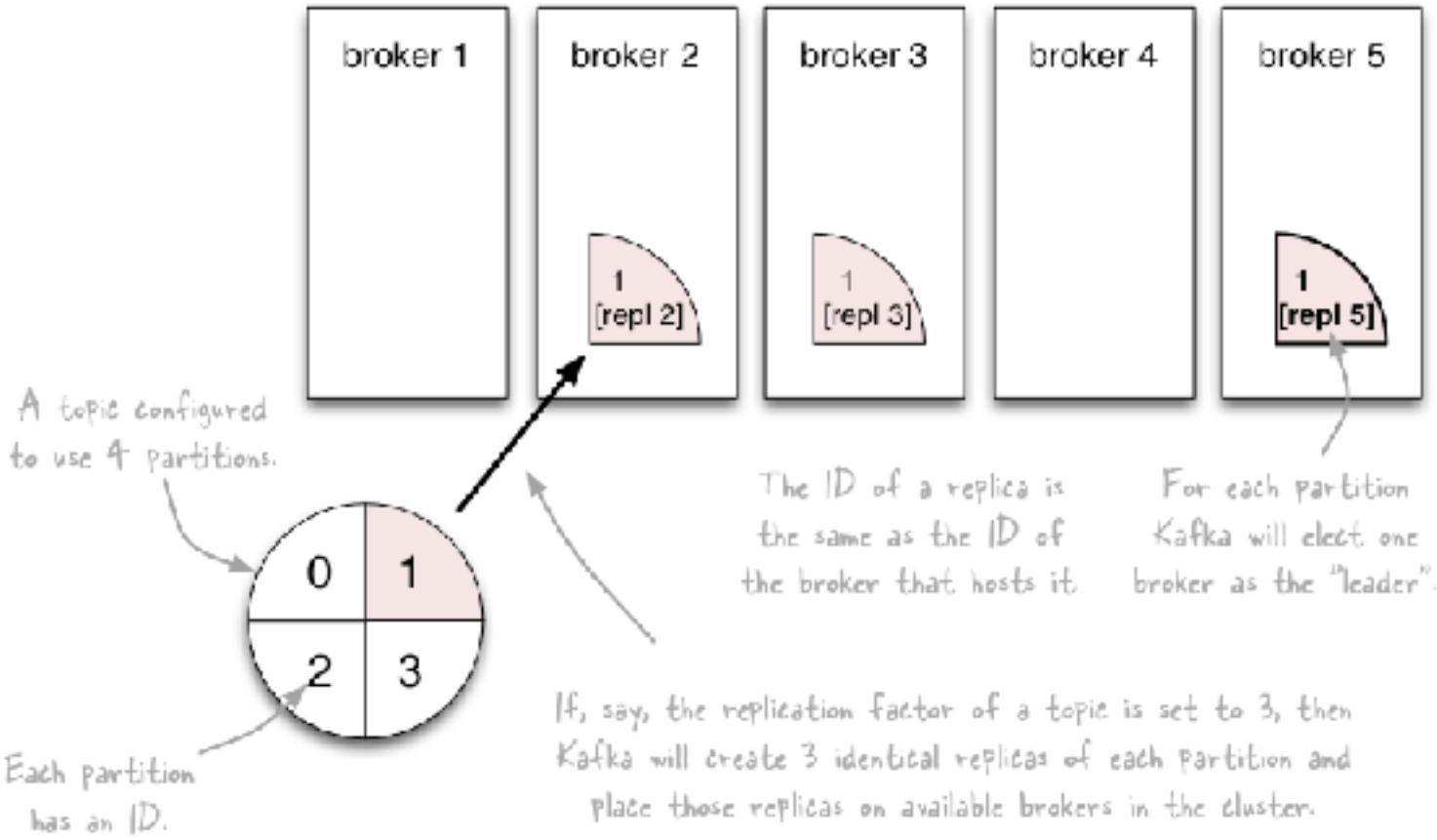
- Offset: messages in the partitions are each assigned a unique (per partition) and sequential id called the offset
  - Consumers track their pointers via (offset, partition, topic) tuples



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



Inspecting the current state of a topic

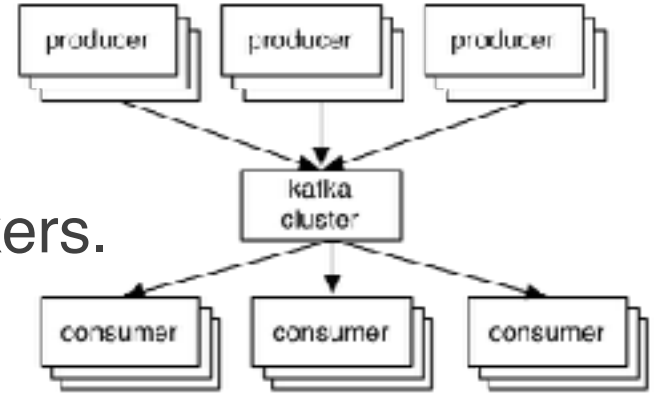
- --describe the topic

```
$ 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

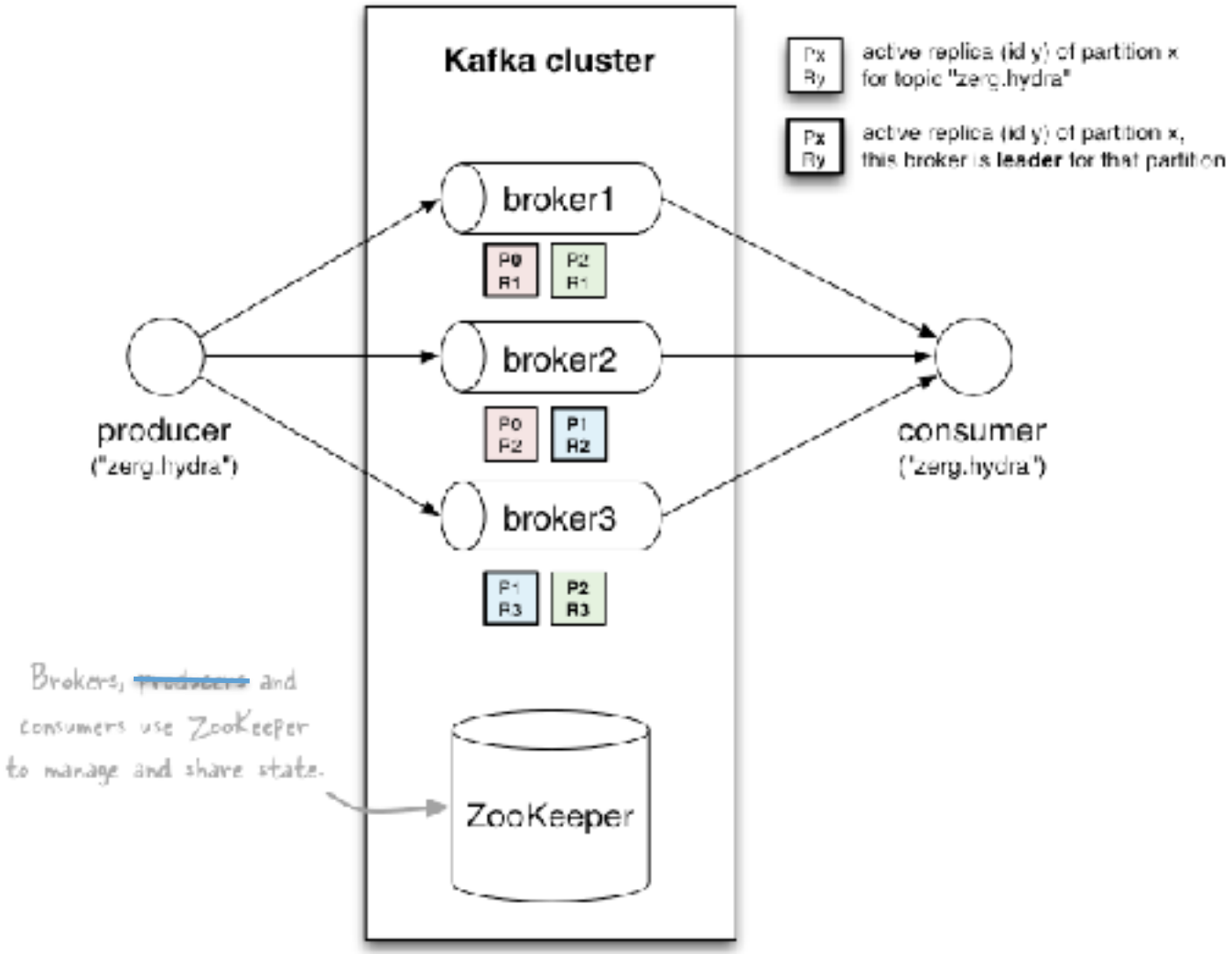
Let’s recap

- 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



WRAPPING UP

Where to find help

- No (good) Kafka book available yet.
- Kafka documentation
  - http://kafka.apache.org/documentation.html
  - https://cwiki.apache.org/confluence/display/KAFKA/Index
- Kafka ecosystem, e.g. Storm integration, Puppet
  - https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem
- Mailing lists
  - http://kafka.apache.org/contact.html
- Code examples
  - examples/ directory in Kafka, https://github.com/apache/