Kafka producers

Kafka Producer/Consumer Details

Tos

- ***Producers*** write to and ***Consumers*** read from ***Topic(s)***
- ***Topic*** associated with a log which is data structure on disk
- ***Producer***(s) append ***Records*** at end of Topic log
- Topic ***Log*** consist of Partitions
  - Spread to multiple files on multiple nodes
- ***Consumers*** read from Kafka at their own cadence
  - Each Consumer (Consumer Group) tracks offset from where they left off reading
- ***Partitions*** can be distributed on different machines in a cluster
  - high performance with horizontal scalability and failover with replication

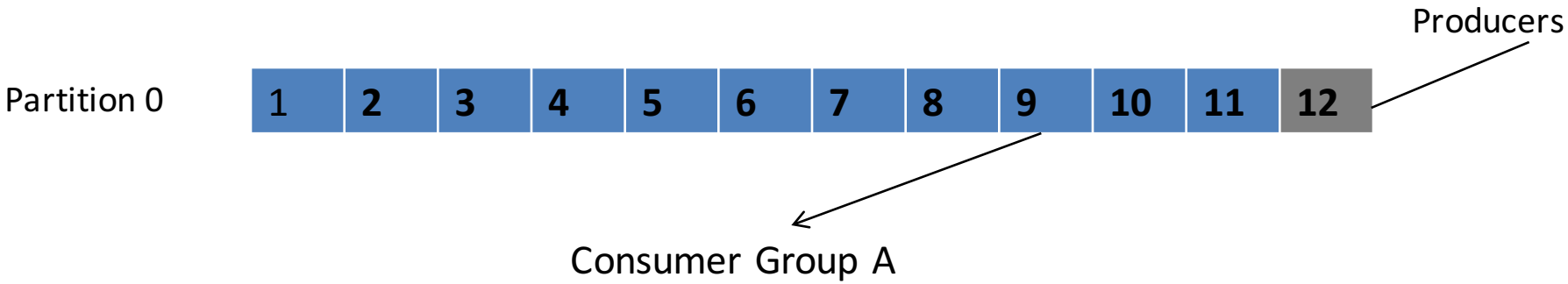
Kafka Producers

Tos

- ***Producers*** send records to topics
- ***Producer*** picks which partition to send record to per topic
  - Can be done in a ***round-robin***
  - Can be based on priority
  - Typically based on ***key*** of ***record***
  - Kafka ***default partitioner*** for Java uses hash of keys to choose partitions, or a round-robin strategy if no key
- Important: *Producer picks partition*

Kafka producers and consumers

Tos



Producers are writing at Offset 12
Consumer Group A is Reading from Offset 9.

Kafka Producers

Tos

- ***Producers*** write at their own cadence so order of Records cannot be guaranteed across partitions
- ***Producer*** configures consistency level (ack=0, ack=all, ack=1)
- ***Producers*** pick the ***partition*** such that Record/messages goes to a given same partition based on the data
  - Example have all the events of a certain 'employee Id' go to same partition
  - If order within a partition is not needed, a 'Round Robin' partition strategy can be used so Records are evenly distributed across partitions.

Producer Review

Tos

- Can Producers occasionally write faster than consumers?
- What is the default partition strategy for Producers without using a key?
- What is the default partition strategy for Producers using a key?
- What picks which partition a record is sent to?

Kafka Consumers

Kafka Consumer Groups

Tos

- Consumers are grouped into a ***Consumer Group***
  - ***Consumer group*** has a unique id
  - Each ***consumer group*** is a subscriber
  - Each ***consumer group*** maintains its own offset
  - Multiple subscribers = multiple consumer groups
  - Each has different function: one might delivering records to micro services while another is streaming records to Hadoop
- ***A Record*** is delivered to one ***Consumer*** in a ***Consumer Group***
- Each consumer in consumer groups takes records and only one consumer in group gets same record
- Consumers in Consumer Group ***load balance*** record consumption

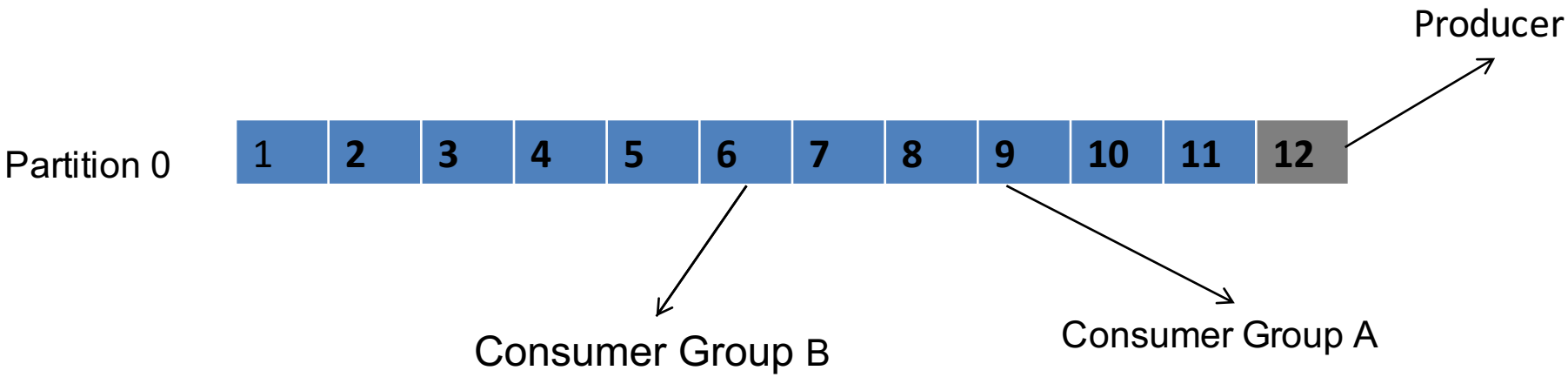
Kafka Consumer Load Share

Tos

- Kafka ***Consumer*** consumption ***divides*** partitions over consumers in a Consumer Group
- Each ***Consumer*** is exclusive consumer of a ***"fair share" of partitions***
- This is Load Balancing
- ***Consumer*** membership in ***Consumer Group*** is handled by the Kafka protocol dynamically
- If new Consumers ***join*** Consumer group, it gets a share of partitions
- If Consumer ***dies***, its partitions are split among remaining live Consumers in Consumer Group

Kafka Consumer Groups

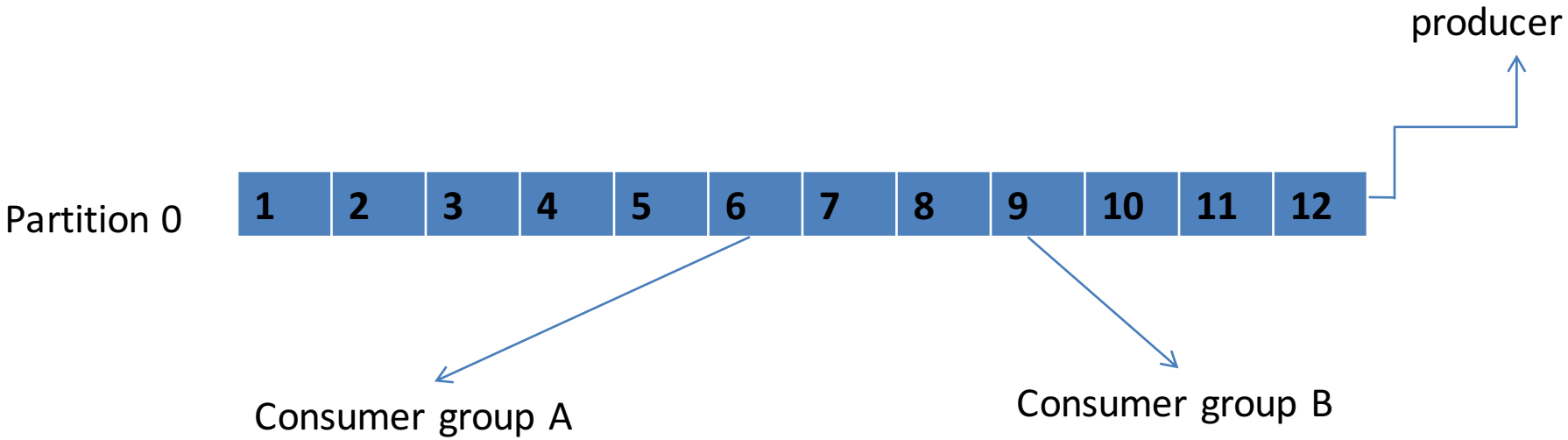
Tos



Consumers remember offset where they left off.
Consumers groups each have their own offset per partition

Kafka Topic Partition, Consumers, Producers

Tos



Consumer groups remember offset where they left off. Consumers groups each have their own offset.

Producer writing to offset 12 of partition0 while… consumer Group A is reading from offset 6.

Consumer Group B is reading from offset 9.

Kafka Consumer Groups Processing

Tos

- How does Kafka divide up topic so multiple ***Consumers*** in a ***Consumer Group*** can process a topic?
- You group consumers into consumers group with a group id
- ***Consumers*** with same id belong in same ***Consumer Group***
- One ***Kafka broker*** becomes ***group coordinator*** for Consumer Group
  - assigns partitions when new members arrive (older clients would talk direct to ZooKeeper now broker does coordination)
  - or reassign partitions when group members leave or topic changes (config / meta-data change
- When ***Consumer group*** is created, offset set according to reset policy of topic

Kafka Consumer Failover

Tos

- ***Consumers*** notify broker when it successfully processed a record
  - advances offset
- If ***Consumer*** fails before sending commit offset to Kafka broker,
  - different ***Consumer*** can continue from the last committed offset
  - some Kafka records could be reprocessed
  - ***at least once behaviour***
  - ***messages should be idempotent***

Kafka Consumer Offsets and Recovery

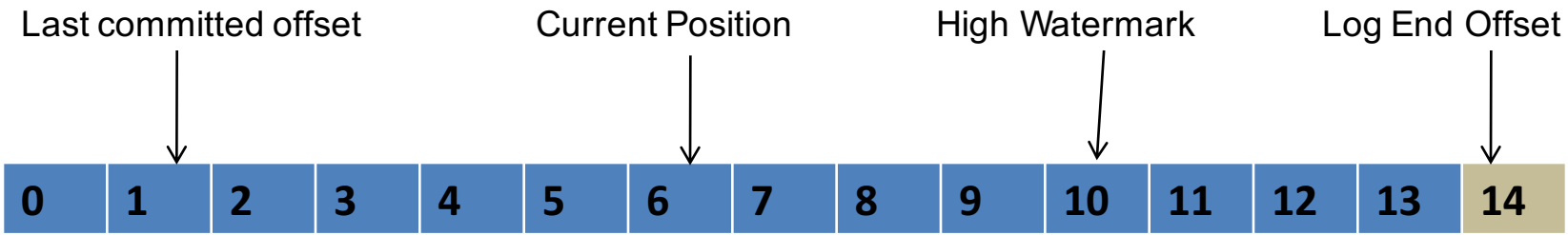
Tos

- Kafka stores offsets in topic called “consumer _ offset”
- Uses Topic Log Compaction
- When a consumer has processed data, it should commit offsets
- If consumer process dies, it will be able to start up and start reading where it left off based on offset stored in “ consumer _ offset”

Kafka Consumer: What can be consumed?

Tos

- ***"Log end offset"*** is offset of last record written to log partition and where ***Producers*** write to next
- ***"High watermark"*** is offset of last record successfully replicated to all partitions followers
- ***Consumer*** only reads up to “high watermark”.
  Consumer can’t read un-replicated data



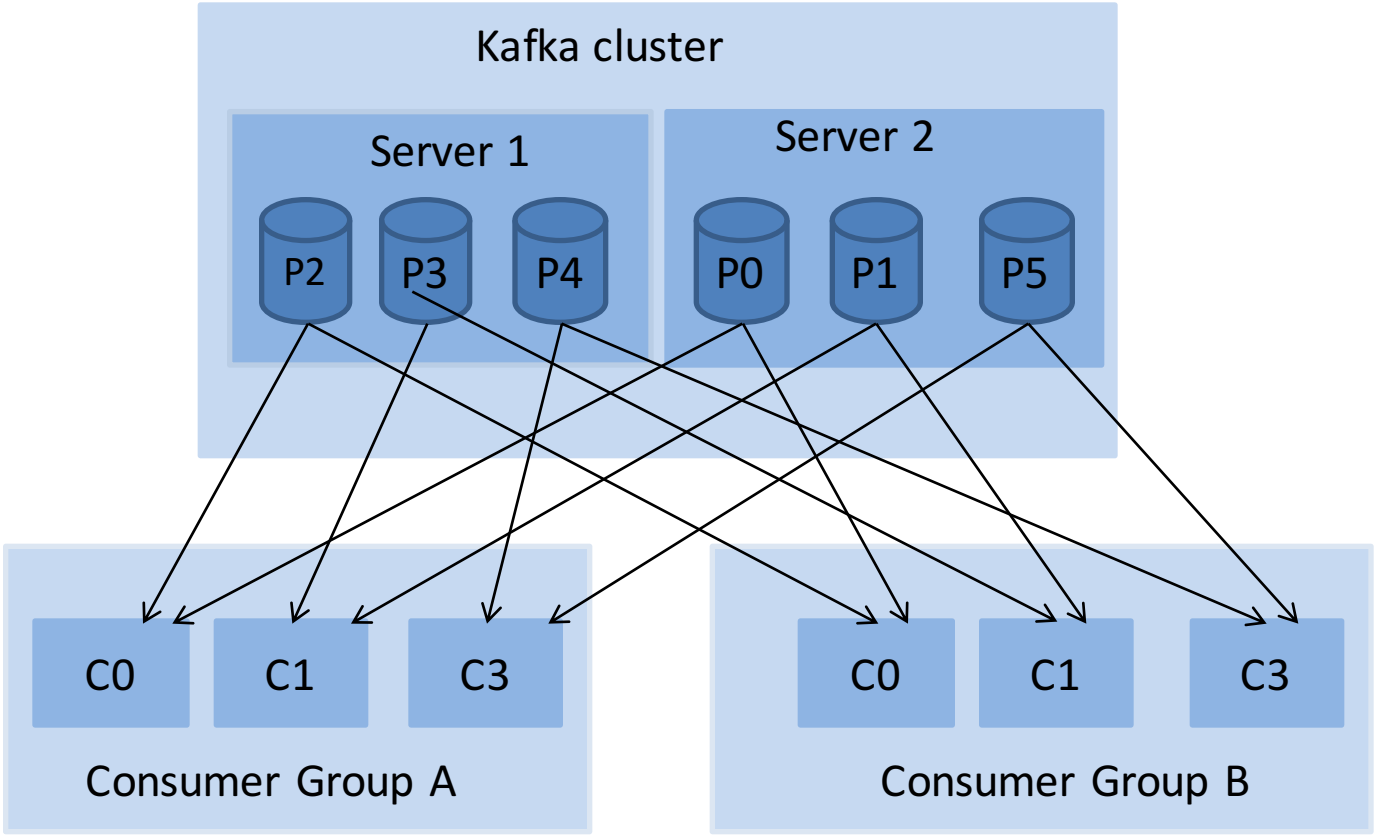
Consumer to Partition Cardinality

Tos

- Only a single ***Consumer*** from the same ***Consumer Group*** can access a single ***Partition***
- If ***Consumer Group*** count ***exceeds*** Partition count:
  - Extra Consumers remain idle; can be used for failover
- If more Partitions than Consumer Group instances,
  - Some Consumers will read from more than one partition

2 server Kafka cluster hosting 6 partitions(P0-P5)

Tos



Multi-threaded Consumers

Tos

- You can run more than one Consumer in a JVM process
- If processing records takes a while, a single Consumer can run multiple threads to process records
  - Harder to manage offset for each Thread/Task
  - One Consumer runs multiple threads
  - 2 messages on same partitions being processed by two different threads
  - Hard to guarantee order without threads coordination
- ***PREFER***: Multiple Consumers can run each processing record batches in their own thread
  - Easier to manage offset
  - Each Consumer runs in its thread
  - Easier to mange failover (each process runs X num of Consumer threads

Consumer Review

Tos

- What is a consumer group?
- Does each consumer have its own offset?
- When can a consumer see a record?
- What happens if there are more consumers than partitions?
- What happens if you run multiple consumers in many thread in the same JVM?

Run Kafka

Tos

- Run producer from command line
- Run consumer from command line

Run Kafka Producer

Tos

start-producer-console.sh

```
1  #!/usr/bin/env bash
2  cd ~/kafka-training
3
4  kafka/bin/kafka-console-producer.sh --broker-list \
5  localhost:9092 --topic my-topic
```

Run Kafka Consumer

Tos

start-consumer-console.sh

```
#!/usr/bin/env bash
cd ~/kafka-training

kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--topic my-topic --from-beginning
```

Running Kafka Producer and Consumer

Tos

```
new-employees
~/kafka-training/lab1/solution
$ ./start-producer-console.sh
This is message 1
This is message 2
This is message 3
Message 4
Message 5
Message 6
Message 7
```

```
Last login: Sat May 13 13:57:09 on ttys004
~/kafka-training/lab1/solution
$ ./start-consumer-console.sh
Message 4
This is message 2
This is message 1
This is message 3
Message 5
Message 6
Message 7
```

Kafka Single Node Review

Tos

- What server do you run first?
- What tool do you use to create a topic?
- What tool do you use to see topics?
- What tool did we use to send messages on the command line?
- What tool did we use to view messages in a topic?
- Why were the messages coming out of order?
- How could we get the messages to come in order from the consumer?

Lab : **Basic Kafka Operations - CLI (Producer & Consumer) – 25 Mins**