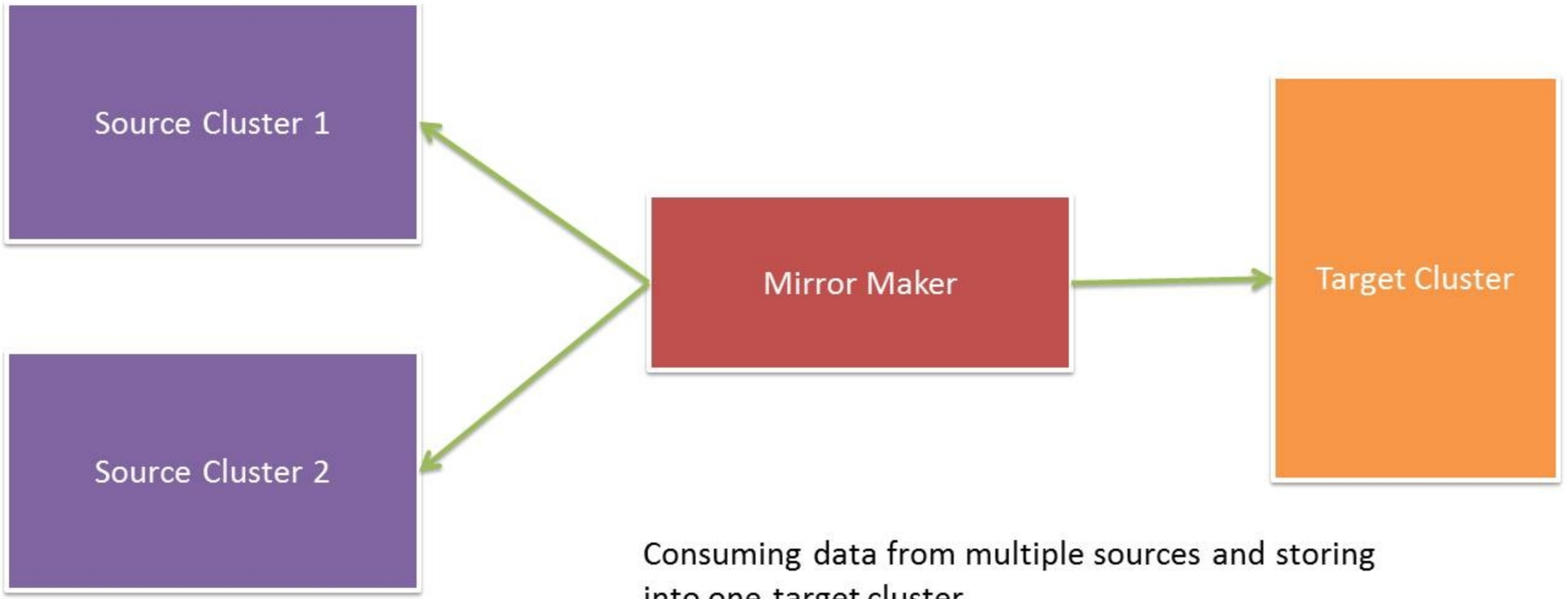
Welcome to the World of Distributed Messaging Queue

Mirroring- Multiple Source, One target

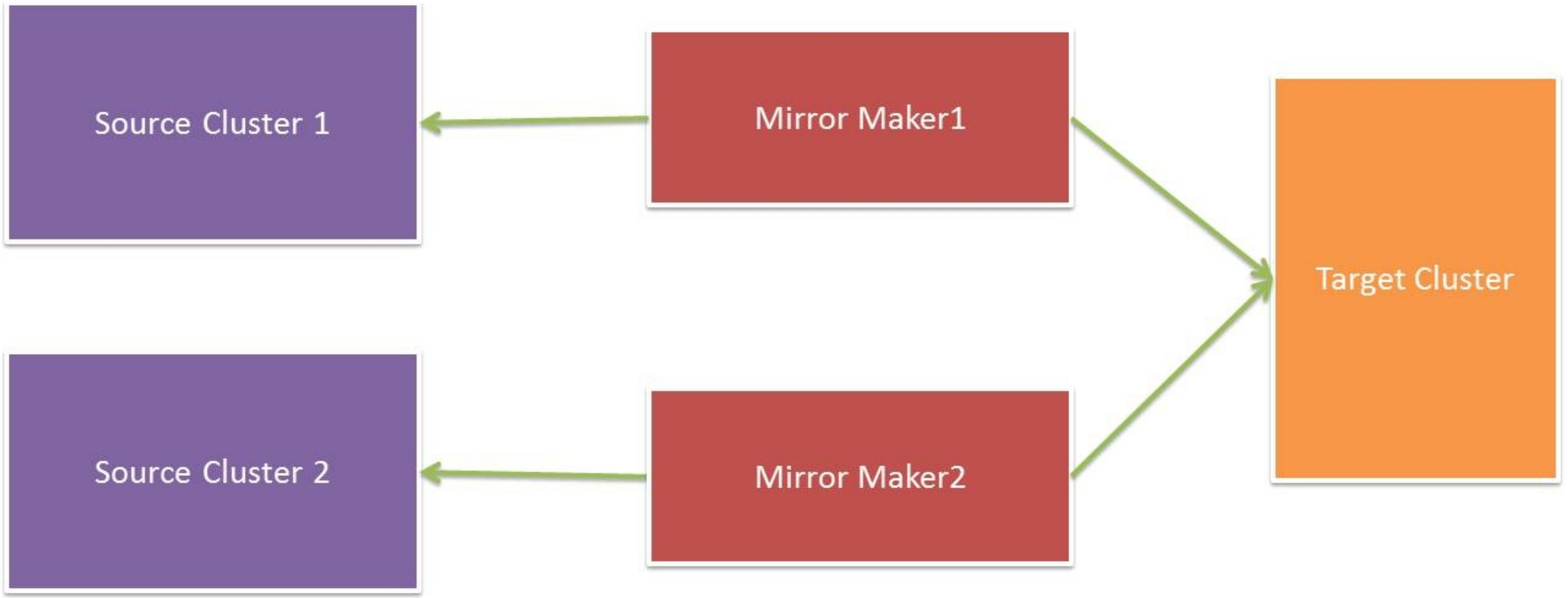


Consuming data from multiple sources and storing into one target cluster.

Mirror Maker - Multiple Source, One target

- Start the mirrorMaker
  - bin/kafka-run-class.sh kafka.tools.MirrorMaker --consumer.config **consumer1.properties** --consumer.config **consumer2.properties** --num.streams 2 --producer.config **producer.properties** --whitelist=".*“
  - The **consumer.properties** contains the details of source cluster.
  - The **producer.properties** contains the details of target cluster

Multiple Source, Multiple Mirror Maker



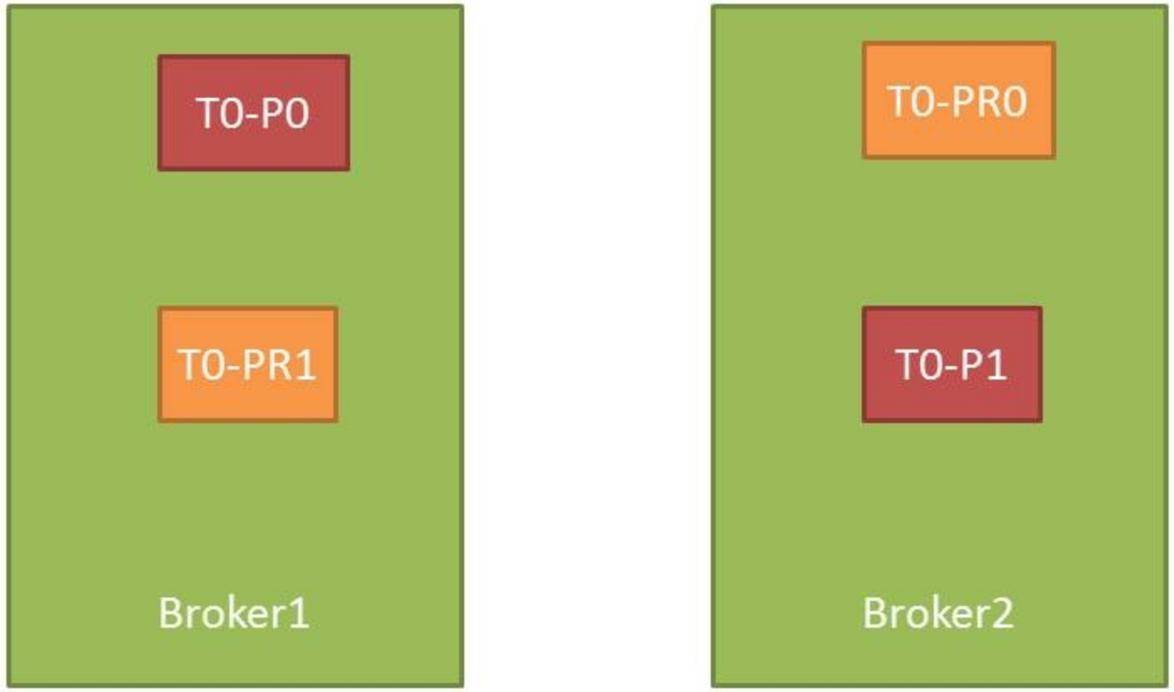
Multiple Source, Multiple Mirror Maker

- Start the MirrorMaker on machine 1
  - bin/kafka-run-class.sh kafka.tools.MirrorMaker --consumer.config **consumer1.properties** --num.streams 2 --producer.config **producer.properties** --whitelist=".*“
- Start the MirrorMaker on machine 2
  - bin/kafka-run-class.sh kafka.tools.MirrorMaker --consumer.config **consumer2.properties** --num.streams 2 --producer.config **producer.properties** --whitelist=".*“

Balancing Leadership

- What happens when any node in a cluster goes down?
  - The leadership for that broker's partitions transfers to other replicas.
  - By default, if any node up in cluster, then it will only be a follower for all its partitions

Balancing Leadership



1. Two nodes kafka cluster.
2. Topic 0
    1. Partition →2
    2. Replication →2
3. Now, stopping broker2.
4. The replica of partition P1 will become leader on Broker1 machine.
5. Now starting the Broker 2.
6. All the partitions assign to broker 2 is a follower.

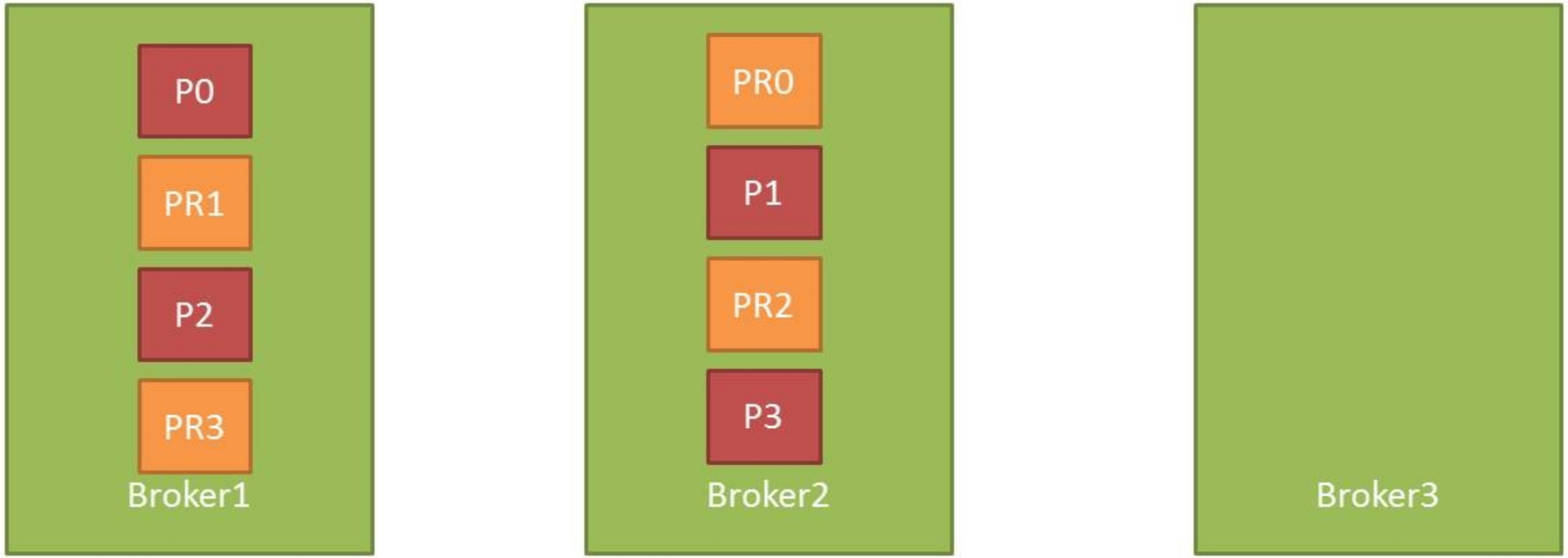
Balancing Leadership

- We can rebalance the leadership between the node by running the below command on Kafka cluster.
  - bin/kafka-preferred-replica-election.sh --zookeeper zk_host:port
- The above command we need to run manually for each time we want to rebalance the cluster.
- Is it possible to automatically configure the rebalancing?
  - Yes
  - By setting the property **auto.leader.rebalance.enable=true**

Scaling cluster

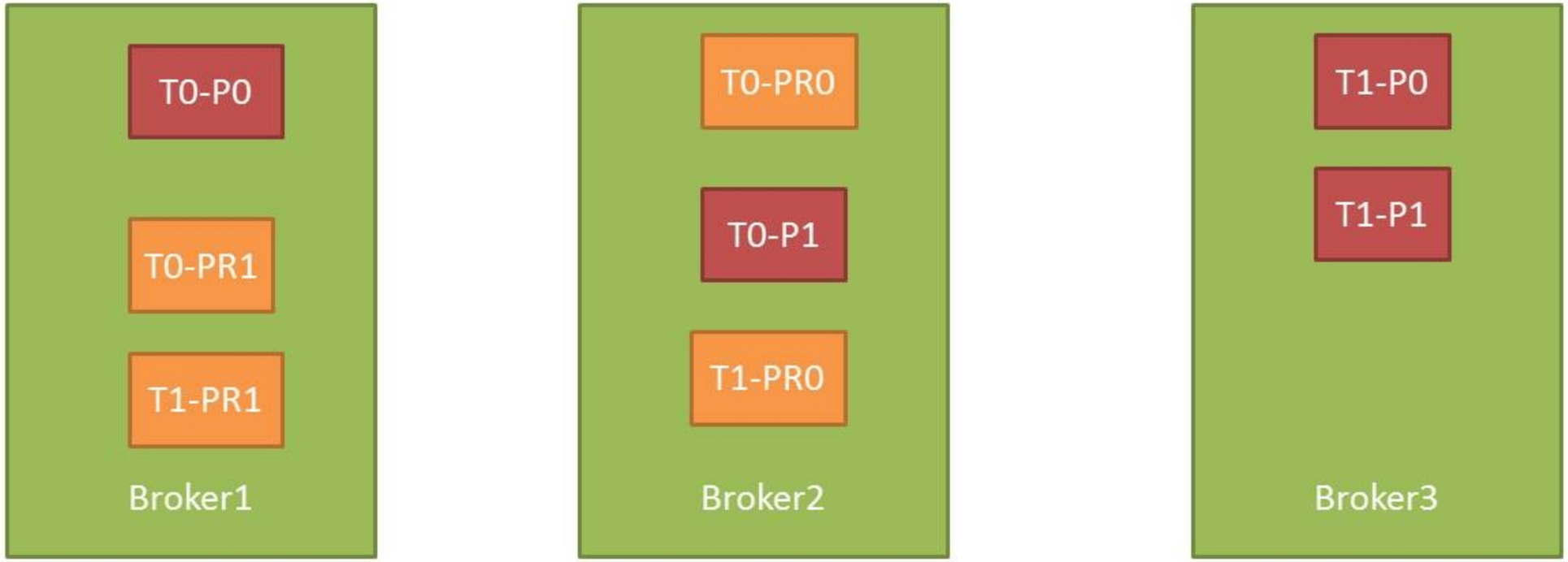
- Can we scale the Kafka cluster?
  - Yes
- How?
- By starting a new node which has unique broker id and pointing to the same zookeeper cluster.

Scaling Cluster



- Adding a new node in a cluster
- No existing partition is assigned to new node

Scaling Cluster – Create new topic



- Adding a new node in a cluster
- No existing partition is assigned to new node
- Creating new topic (Topic1)
- Partitions only assign to new node, when new topics are created

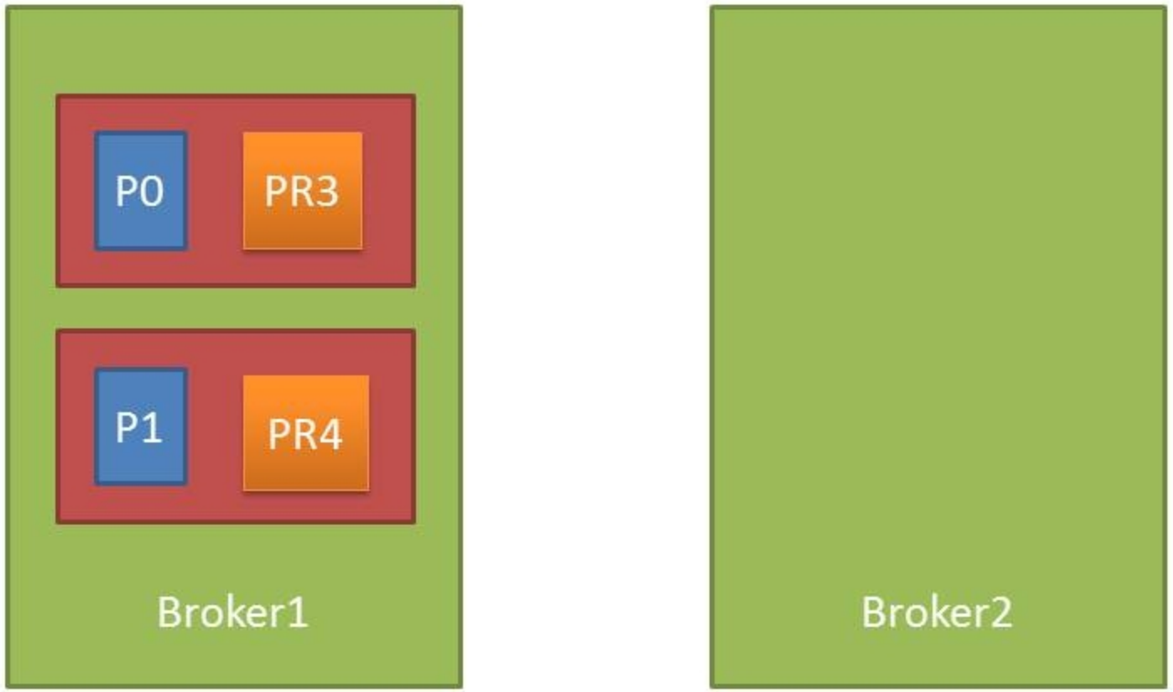
T0- Topic 0
T1 - Topic1

Scaling Cluster

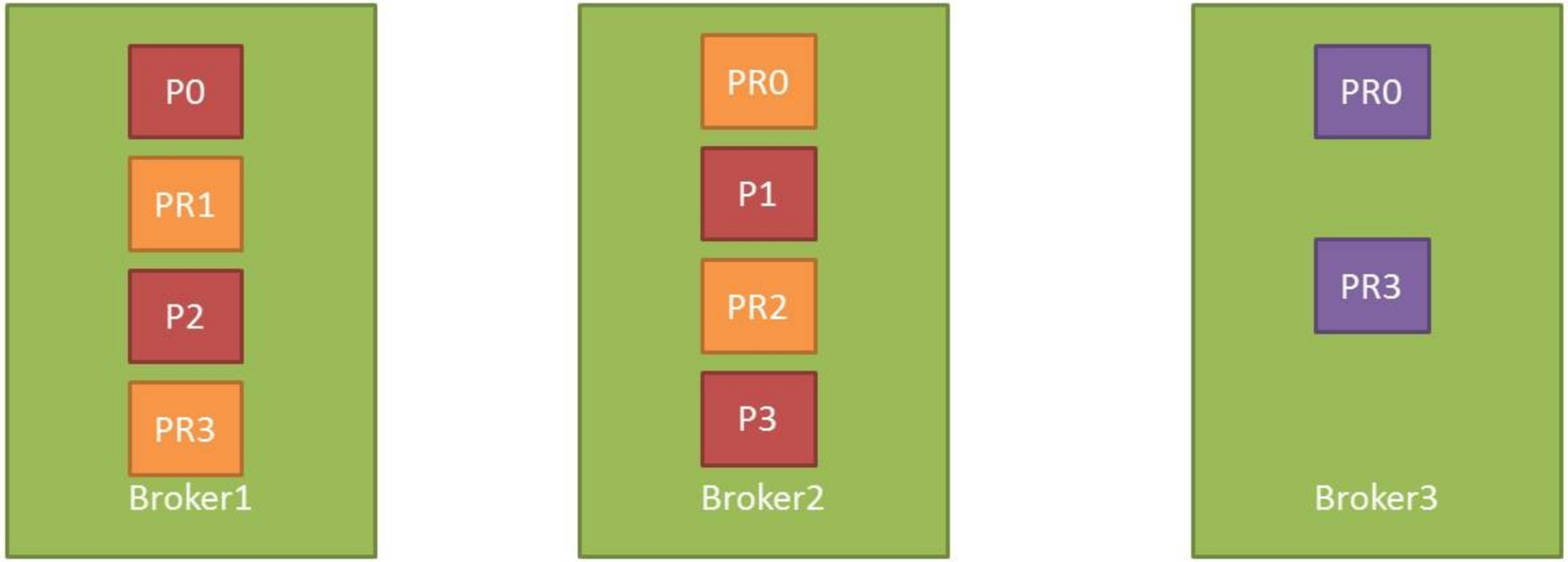
- Can we assign some existing partitions to these new server automatically?
  - No
- Basic Requirement:
  - When you add nodes to your cluster, you definitely want to migrate some existing data to these nodes to release some load from existing nodes.
  - How we can achieve data re-distribution?
- We can manually move the partition between nodes.

Scaling cluster

- The process of migrating data is manually initiated but fully automated.
- How?
- Kafka will add new server as a follower of the partition it is migration and allow it to fully replicate the existing data in that partition.
- When the migration done successfully, the follower assign to new server is added in ISR list and one of the existing replicas will delete their partition's data.



Scaling cluster



- Adding a new node in a cluster
- No existing partition is assigned to new node
- Consider, user is manually triggering the partitions reassignment command
- The data of some partitions are started migrating on new node
- After successfully migration, the one follower of each moved partitions are removed.

Partition Reassignment

- Why Partition reassignment is require?
  - Even data load and partition sizes across all brokers.
  - The partition reassignment tool can be used to move some topics off of the current set of brokers to the newly added brokers. This is typically useful while expanding an existing cluster since it is easier to move entire topics to the new set of brokers, than moving one partition at a time.
- In 0.8.1, the partition reassignment tool does not have the capability to automatically study the data distribution in a Kafka cluster and move partitions around to attain an even load distribution.
- The admin/user has to figure out which topics or partitions should be moved around.

Partition Reassignment

- The partition reassignment tool has three operation modes.
  --generate: This mode generate the current list of topics and list of brokers. The tool also generates a candidate reassignment to move all partitions of the specified topics to the new brokers.
  --execute: In this mode, the tool kicks off the reassignment of partitions based on the user provided reassignment plan.
  --verify: In this mode, the tool verifies the status of the reassignment for all partitions listed during the last --execute. The possible values of --verify command are **successfully completed**, **failed** or **in progress**

Partition Reassignment - Automatically migrating

- Automatically migrating data to new machines
  - The partition reassignment tools accepts the input list of topics as a json file, you first need to identify the topics you want to move.
  - Create the json file “test.json”:
    - {"topics": [{"topic": “topic1"}, {"topic": “topic2"}], "version":1 }

Partition Reassignment - Automatically migrating

- Once the json file is ready, use the partition reassignment tool to generate a candidate assignment
  - bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --topics-to-move-json-file **test.json** --broker-list "5,6" –generate

Current partition replica assignment

{"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[1,2]},
{"topic":"topic1","partition":0,"replicas":[3,4]}, {"topic":"topic2","partition":2,"replicas":[1,2]},
{"topic":"topic2","partition":0,"replicas":[3,4]}, {"topic":"topic1","partition":1,"replicas":[2,3]},
{"topic":"topic2","partition":1,"replicas":[2,3]}] }

Proposed partition reassignment configuration

{"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[5,6]},
{"topic":"topic1","partition":0,"replicas":[5,6]}, {"topic":"topic2","partition":2,"replicas":[5,6]},
{"topic":"topic2","partition":0,"replicas":[5,6]}, {"topic":"topic1","partition":1,"replicas":[5,6]},
{"topic":"topic2","partition":1,"replicas":[5,6]}] }

Partition Reassignment - Automatically migrating

- However, that at this point, the partition movement has not started, it merely tells you the current assignment and the proposed new assignment. The current assignment should be saved in case you want to rollback to it. The new assignment should be saved in a json file (e.g. e**xpand-cluster-reassignment.json**) to be input to the tool with the --execute option as follows-

- bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file **expand-cluster-reassignment.json** --execute

  Current partition replica assignment

  {"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[1,2]},
  {"topic":"topic1","partition":0,"replicas":[3,4]}, {"topic":"topic2","partition":2,"replicas":[1,2]},
  {"topic":"topic2","partition":0,"replicas":[3,4]}, {"topic":"topic1","partition":1,"replicas":[2,3]},
  {"topic":"topic2","partition":1,"replicas":[2,3]}] }

  Successfully started reassignment of partitions

  {"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[5,6]},
  {"topic":"topic1","partition":0,"replicas":[5,6]}, {"topic":"topic2","partition":2,"replicas":[5,6]},
  {"topic":"topic2","partition":0,"replicas":[5,6]}, {"topic":"topic1","partition":1,"replicas":[5,6]},
  {"topic":"topic2","partition":1,"replicas":[5,6]}] }

Partition Reassignment - Automatically migrating

– Finally, the --verify option can be used with the tool to check the status of the partition reassignment. Note that the same **expand-cluster-reassignment.json** (used with the --execute option) should be used with the --verify option

> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file **expand-cluster-reassignment.json** --verify

Status of partition reassignment:

Reassignment of partition [topic1,0] completed successfully
Reassignment of partition [topic1,1] is in progress
Reassignment of partition [topic1,2] is in progress
Reassignment of partition [topic2,0] completed successfully
Reassignment of partition [topic2,1] completed successfully
Reassignment of partition [topic2,2] completed successfully

Partition Reassignment - Automatically migrating

- Automatically migrating data to new machines
  - The partition reassignment tools accepts the input list of topics as a json file, you first need to identify the topics you want to move.
  - Create the json file “test.json”:
    - {"topics": [{"topic": “topic1"}, {"topic": “topic2"}], "version":1 }

Partition Reassignment - Automatically migrating

- Once the json file is ready, use the partition reassignment tool to generate a candidate assignment
  - bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --topics-to-move-json-file **test.json** --broker-list "5,6" –generate

Current partition replica assignment

{"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[1,2]},
{"topic":"topic1","partition":0,"replicas":[3,4]}, {"topic":"topic2","partition":2,"replicas":[1,2]},
{"topic":"topic2","partition":0,"replicas":[3,4]}, {"topic":"topic1","partition":1,"replicas":[2,3]},
{"topic":"topic2","partition":1,"replicas":[2,3]}] }

Proposed partition reassignment configuration

{"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[5,6]},
{"topic":"topic1","partition":0,"replicas":[5,6]}, {"topic":"topic2","partition":2,"replicas":[5,6]},
{"topic":"topic2","partition":0,"replicas":[5,6]}, {"topic":"topic1","partition":1,"replicas":[5,6]},
{"topic":"topic2","partition":1,"replicas":[5,6]}] }

Partition Reassignment - Automatically migrating

- However, that at this point, the partition movement has not started, it merely tells you the current assignment and the proposed new assignment. The current assignment should be saved in case you want to rollback to it. The new assignment should be saved in a json file (e.g. e**xpand-cluster-reassignment.json**) to be input to the tool with the --execute option as follows-

- bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file **expand-cluster-reassignment.json** --execute

  Current partition replica assignment

  {"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[1,2]},
  {"topic":"topic1","partition":0,"replicas":[3,4]}, {"topic":"topic2","partition":2,"replicas":[1,2]},
  {"topic":"topic2","partition":0,"replicas":[3,4]}, {"topic":"topic1","partition":1,"replicas":[2,3]},
  {"topic":"topic2","partition":1,"replicas":[2,3]}] }

  Successfully started reassignment of partitions

  {"version":1, "partitions":[{"topic":"topic1","partition":2,"replicas":[5,6]},
  {"topic":"topic1","partition":0,"replicas":[5,6]}, {"topic":"topic2","partition":2,"replicas":[5,6]},
  {"topic":"topic2","partition":0,"replicas":[5,6]}, {"topic":"topic1","partition":1,"replicas":[5,6]},
  {"topic":"topic2","partition":1,"replicas":[5,6]}] }

Partition Reassignment - Automatically migrating

– Finally, the --verify option can be used with the tool to check the status of the partition reassignment. Note that the same **expand-cluster-reassignment.json** (used with the --execute option) should be used with the --verify option

> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file **expand-cluster-reassignment.json** --verify

Status of partition reassignment:

Reassignment of partition [topic1,0] completed successfully
Reassignment of partition [topic1,1] is in progress
Reassignment of partition [topic1,2] is in progress
Reassignment of partition [topic2,0] completed successfully
Reassignment of partition [topic2,1] completed successfully
Reassignment of partition [topic2,2] completed successfully

Partition Reassignment – Manual migrating

- Custom partition assignment and migration
  - The partition reassignment tool can also be used to selectively move replicas of a partition to a specific set of brokers.
  - We can skip the --generate step, as we are manually creating and passing the reassignment logic

Partition Reassignment – Manual migrating

- Custom partition assignment and migration
  - The first step is to hand craft the custom reassignment plan in a json file-

    > cat custom-reassignment.json

    {"version":1,"partitions":[{"topic":"foo1","partition":0,"replicas":[5,6]},{"topic":"foo2","partition":1,"replicas":[2,3]}]}

    Then, use the json file with the --execute option to start the reassignment process

    -> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file custom-reassignment.json --execute

    Current partition replica assignment

    {"version":1, "partitions":[{"topic":"foo1","partition":0,"replicas":[1,2]}, {"topic":"foo2","partition":1,"replicas":[3,4]}] }

    Save this to use as the --reassignment-json-file option during rollback

    Successfully started reassignment of partitions

    {"version":1, "partitions":[{"topic":"foo1","partition":0,"replicas":[5,6]}, {"topic":"foo2","partition":1,"replicas":[2,3]}] }

Partition Reassignment – Increasing repliaction

- Increasing replication
  - Increasing the replication factor of an existing partition is easy. Just specify the extra replicas in the custom reassignment json file and use it with the --execute option to increase the replication factor of the specified partitions.

Partition Reassignment – Increasing repliaction

- The first step is to hand craft the custom reassignment plan in a json file-
  > cat increase-replication-factor.json
  {"version":1, "partitions":[{"topic":"foo","partition":0,"replicas":[5,6,7]}]}

  Then, use the json file with the --execute option to start the reassignment process
  -> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file increase-replication-factor.json --execute

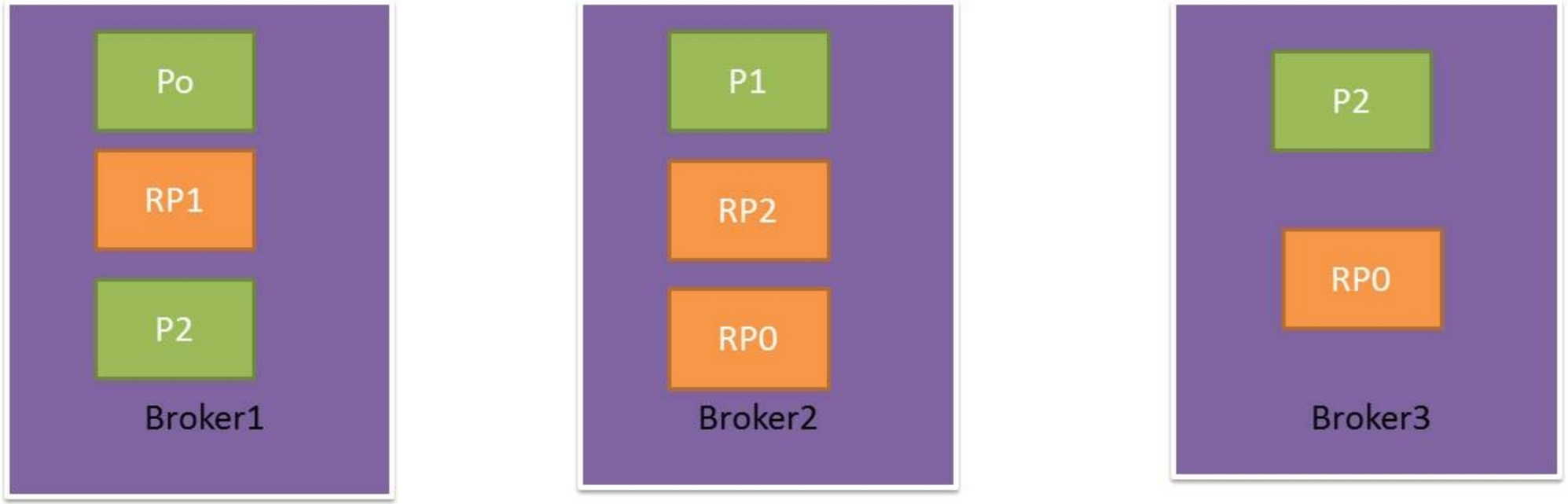
  Current partition replica assignment
  {"version":1, "partitions":[{"topic":"foo","partition":0,"replicas":[5]}]}

  Save this to use as the --reassignment-json-file option during rollback
  Successfully Started reassignment of partitions
  {"version":1, "partitions":[{"topic":"foo","partition":0,"replicas":[5,6,7]}]}

Decommissioning brokers

- The partition reassignment tool does not have the ability to automatically generate a reassignment plan for decommissioning brokers yet.
- As such, the admin has to come up with a reassignment plan to move the replica for all partitions hosted on the broker to be decommissioned, to the rest of the brokers.

Decommissioning brokers



- Want to stop the broker3.
- We need to manually generate the reassignment plan to move topics of broker3 to available brokers.
- Execute the reassignment plan. (P2 → Broker 1 and RP0→ Broker2)
- verify the reassignment plan.
- Once the reassignment done.
- Stop the broker 3

Updating Topics

- Change the configuration or partitions of topic.
- Increase the number of partition
  - bin/kafka-topics.sh --zookeeper zk_host:port --alter --topic TOPIC_NAME --partitions 10
  - The custom data partition approach may affect by adding new partitions for a topic

Updating Topics

- Add new configuration:
  - bin/kafka-topics.sh --zookeeper zk_host:port --alter --topic TOPIC_NAME --config CONFIG_NAME=CONFIG_VALUE
- Delete configuration:
  - bin/kafka-topics.sh --zookeeper zk_host:port --alter --topic TOPIC_NAME --deleteConfig CONFIG_NAME
- Deleting topic:
  - bin/kafka-topics.sh --zookeeper zk_host:port --delete --topic TOPIC_NAME

JMX Monitoring

- What is monitoring?
  - **Monitoring** is the systematic process of collecting, analyzing and using information to track a programme's progress toward reaching its objectives and to guide management decisions.
- The JMX is a set of specifications for management and monitoring of application and devices in the J2EE environment.
- Start the kafka by specifying the JMX port
  - **env JMX_PORT=9999** bin/kafka-server-start.sh config/server1.properties
- Kafka start pushing metrics on **JMX_PORT** 9999
- We can open the java **jconsole** to connect JMX port to view the kafka metrics
  - Open JCONSOLE
  - $JAVA_HOME/bin/jconsole

Compression

- Why compression is require?
  - Reduce the disk space
  - Ohh really, its the only reason.
  - NO
  - Compression helps to increase the performance of I/O intensive applications.
  - How?
  - The reason is simple – disks are slow. Compression reduces the disk footprint of your data leading to faster reads and writes. But at the same time, you invest CPU cycles in decompressing the data read from disk. So it is about striking a balance between I/O load and CPU load.

Compression

- If the application starves on disk capacity but has plenty of CPU cycles to spare, then picking a compression algorithm that yields the largest compression ratio makes sense.
- If compressed data read from disk needs to be transferred over the network, compression ratio directly affects the number of roundtrips required to read compressed data over the wire

Compression

- Kafka support GZip and Snappy compressions
  - Gzip: GZIP is known for large compression ratios, but poor decompression speeds and high CPU usage.
  - Snappy: Snappy has poor compression ratio compare the Gzip but good compression and decompression speeds.

GZip OR Snappy?

- In Kafka 0.7, the compression took place on the producer where it compressed a batch of messages into one compressed message.
- The compressed message received from producer gets appended, as is, to the Kafka broker's log file. When a consumer fetches compressed data, it decompresses the underlying compressed data and hands out the original messages to the user.
- So once data is compressed at source, it stays compressed until it reaches the end consumer. The broker pays no penalty as far as compression overhead is concerned.
- Producer compress the data – (Less data flow in network)
- Consumer decompress the data – (Less data flow in network)
- Broker stored the compress data and doesn't perform any processing.
- Replication is not supported in version 0.7
- LinkedIn team was using Gzip compression with Kafka 0.7 version

GZip OR Snappy?

- In Kafka 0.8, partitions are replicated.
- Leader partition assign unique number (offset) to each message.
- Now, if the data is compressed, the leader has to decompress the data in order to assign offsets to the messages inside the compressed message.
- So the leader has to perform following activity in case of compress data
  - Decompresses data, assigns offsets, compresses it again and then appends the re-compressed data to disk.
- Leader has to do that for every compressed message received

GZip OR Snappy?

Point	GZIp	Snappy
CPU (20 Consumer instances, 300 topics)	More CPU usage (2X)	Less CPU usage (1X)
1GB Data	Compression ratio = 2.8X (More compression ratio)	Compression ratio = 2X
Consumer Throughput	Less data flow in network. More Time require to decompress the data.	Savings in decompression cost are offset by the overhead of making more roundtrips to the Kafka brokers.
Producer Throughput	Low performance	150% higher than Gzip.

Reference: http://nehanarkhede.com/

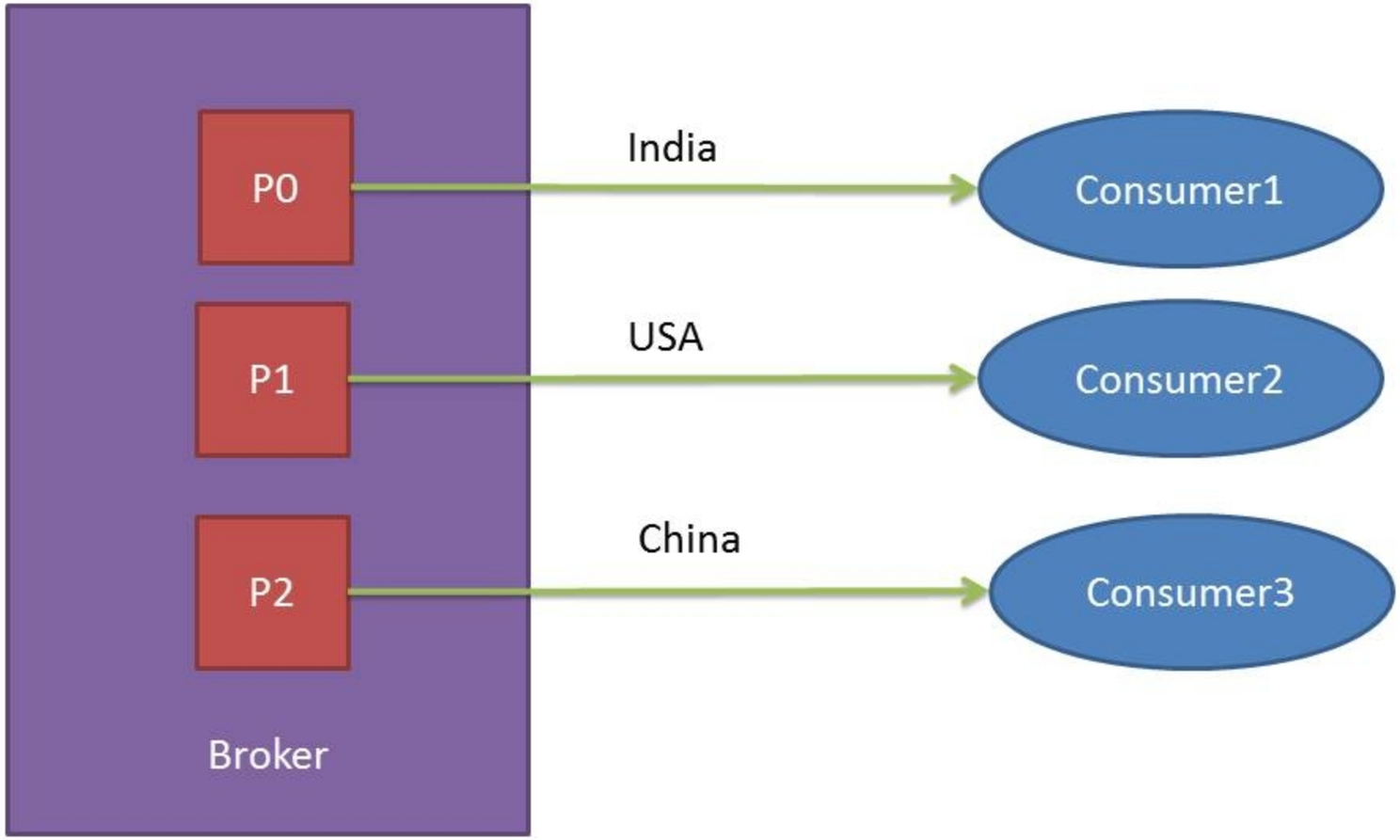
Enable Compression

- Set the following property in produce **compression.codec**. This parameter allows us to specify the compression codec for all data generated by this producer. Valid values are "none", "gzip" and "snappy".
- The another property is **compressed.topics**. This property contains the list of topic we want to compress. If the value of **compressed.topics = null**, then either all the topics are compressed or none topic is compressed, depends on the value of **compression.codec**.

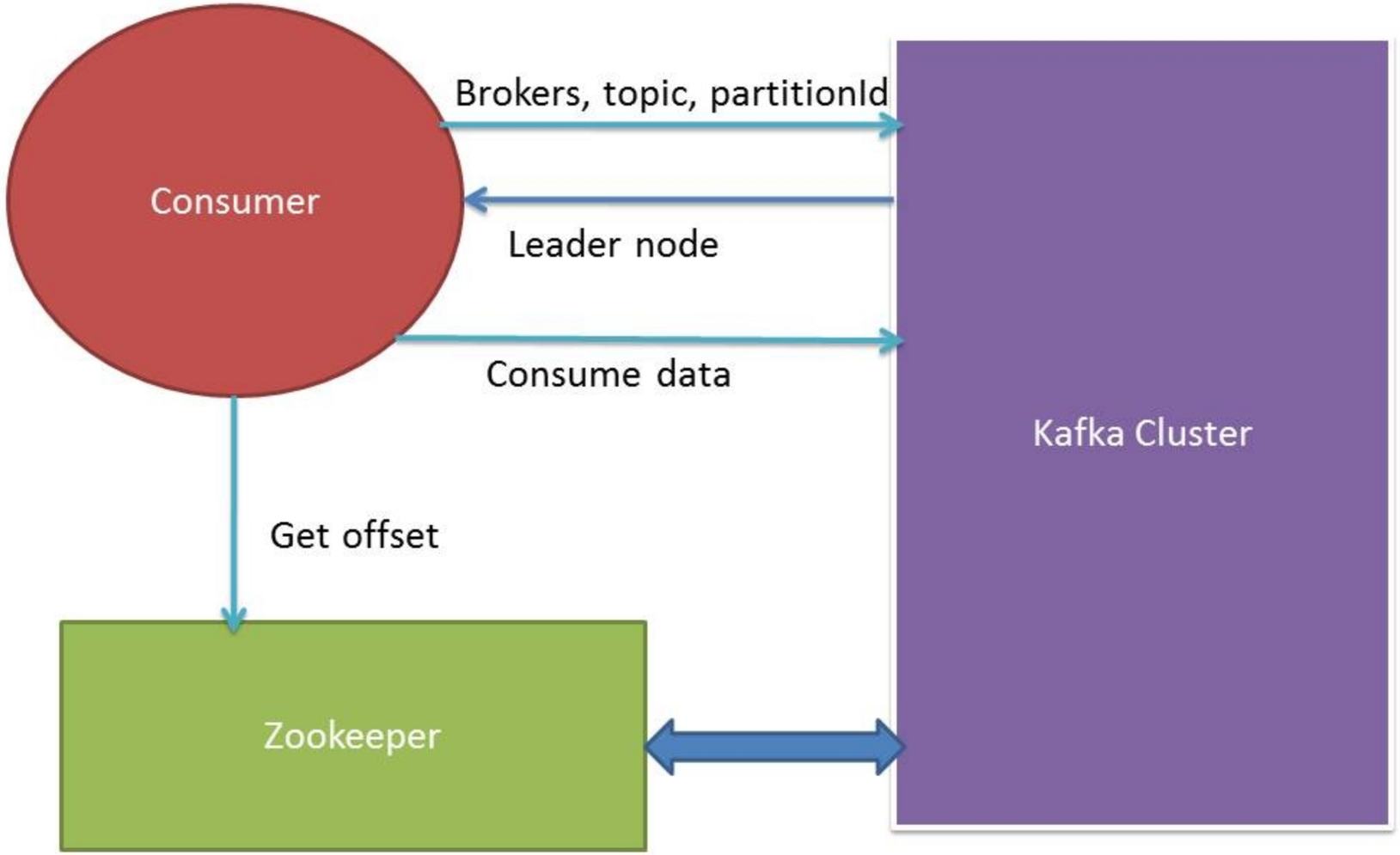
Kafka low level consumer

- We can read a message multiple times.
- We can control the message reading mechanism
- We need to identify the active Broker and also find out which Broker is the leader for your topic and partition.
- Build the input request that define what data we want to read.
- Fetch the data from partition.
- Identify and recover from leader changes

Control Message Reading Mechanism



Low level consumer Hands-On



1. Consumer send the request to find the leader partition of a broker (request contains the broker list, topicName, partitionId)
2. Return the leader node
3. Get the offset from which we need to start the data read
4. Start data consuming
5. Re-elect the leader, if leader node goes down

Data Retention

- The Kafka cluster retains all published messages whether or not they have been consumed for a configurable period of time.
- For example if the log retention is set to two days, then for the two days after a message is published it is available for consumption, after which it will be discarded to free up space.
- Kafka's performance is effectively constant with respect to data size so retaining lots of data is not a problem.

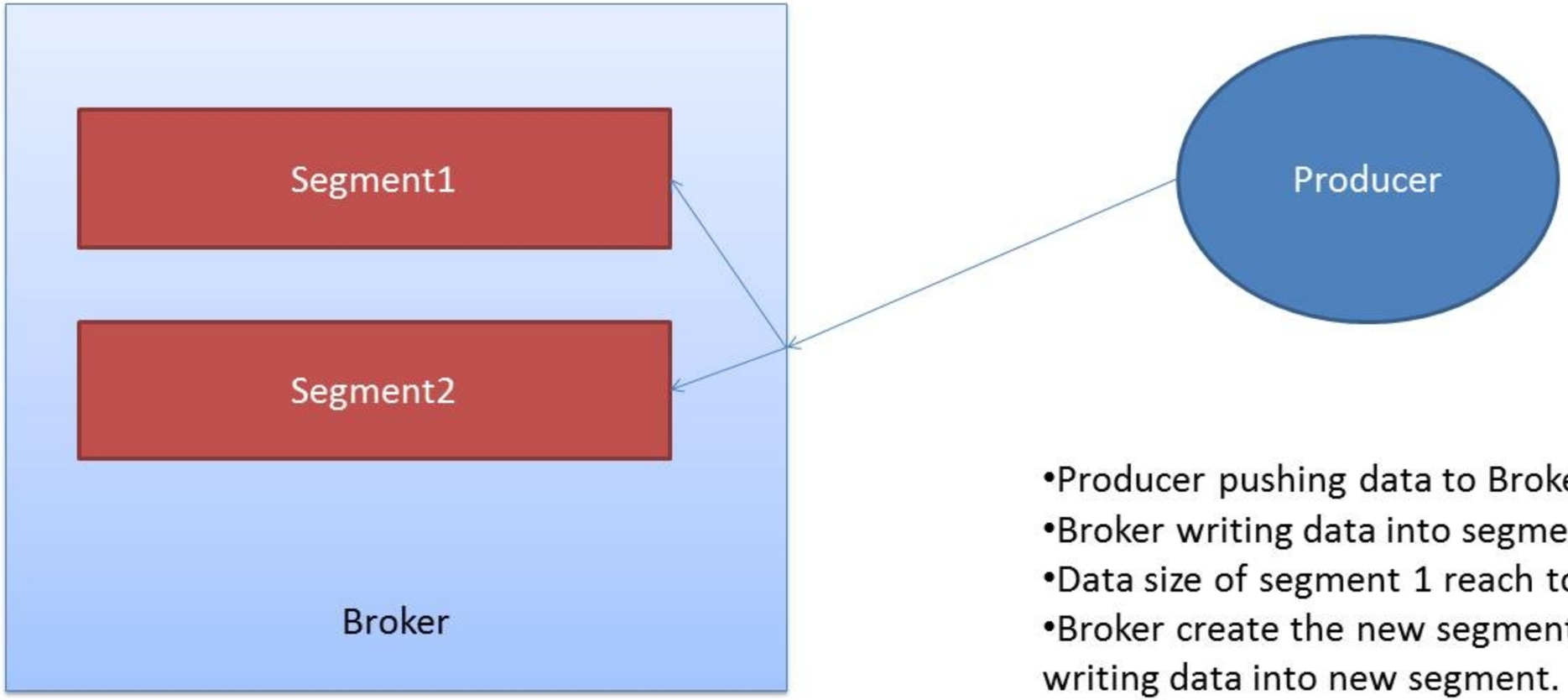
Data Retention

- The property **log.retention.{minutes,hours}** define the amount of time to keep a log segment before it is deleted, i.e. the default data retention window for all topics. The default value of this property is 7 days.
- The property **log.retention.bytes** define the amount of data to retain in the log for each topic-partitions. Note that this is the limit per-partition so multiply by the number of partitions to get the total data retained for the topic. The default value of this property is -1.
- Also note that if both **log.retention.hours** and **log.retention.bytes** are both set we delete a gsegment when either limit is exceeded.
- We can overwrite this property by setting **retention.bytes** and **retention.ms** properties at the time of topic creation.
- The propety **log.retention.check.interval.ms** define the period with which we check whether any log segment is eligible for deletion to meet the retention policies. The default value of this property is 5 minutes.

Log Segment

- The property **log.segment.bytes** define the log for a topic partition is stored as a directory of segment files. This setting controls the size to which a segment file will grow before a new segment is rolled over in the log. The default value is 1GB.
- The property **log.roll.hours** define the setting will force Kafka to roll a new log segment even if the log.segment.bytes size has not been reached. The defaulr value is 168 hours.

Log Segment



- Producer pushing data to Broker.
- Broker writing data into segment 1.
- Data size of segment 1 reach to 1 GB.
- Broker create the new segment and start writing data into new segment.

Log Segment and Data Retention

- Kafka doesn’t delete single message but delete all the records belong to one segment in one go.
- It only mark the data deleted (soft delete).
  - Data inserted before this offset is marked as deleted.
- Why it does not delete the single record?
  - Deleting a single record from a file is very performance incentive task

Assignment

- Create a three nodes kafka cluster.
- Let's assume the file contains data of four countries india, usa, uk and chine.
  - Log file must contains following four fields
  - Tweet text, country, time, username
- Read the data from multiple files, convert the record into Map<String,Object> and pushed into Kafka using Sync producer.
- Write a Map encoder/decoder to convert the Map to bytes and bytes to Map.
- Run the producer on machine other then brokers.
- Create a topic having four partitions and replication factor 3.
- Create a partition class to push data of India on partition 0, data of USA on partition 1 and so on.
- Consume the data from Kafka and store all the data of India on 1 file, data of USA on other file and so on.
- Run the consumer on machine other then brokers

Apache Storm

- ✓ High Distributed real time **computation** system
- ✓ Horizontally scalable
- ✓ Fault Tolerance
- ✓ Can easily be used with any programming language
- ✓ Guaranteed message processing

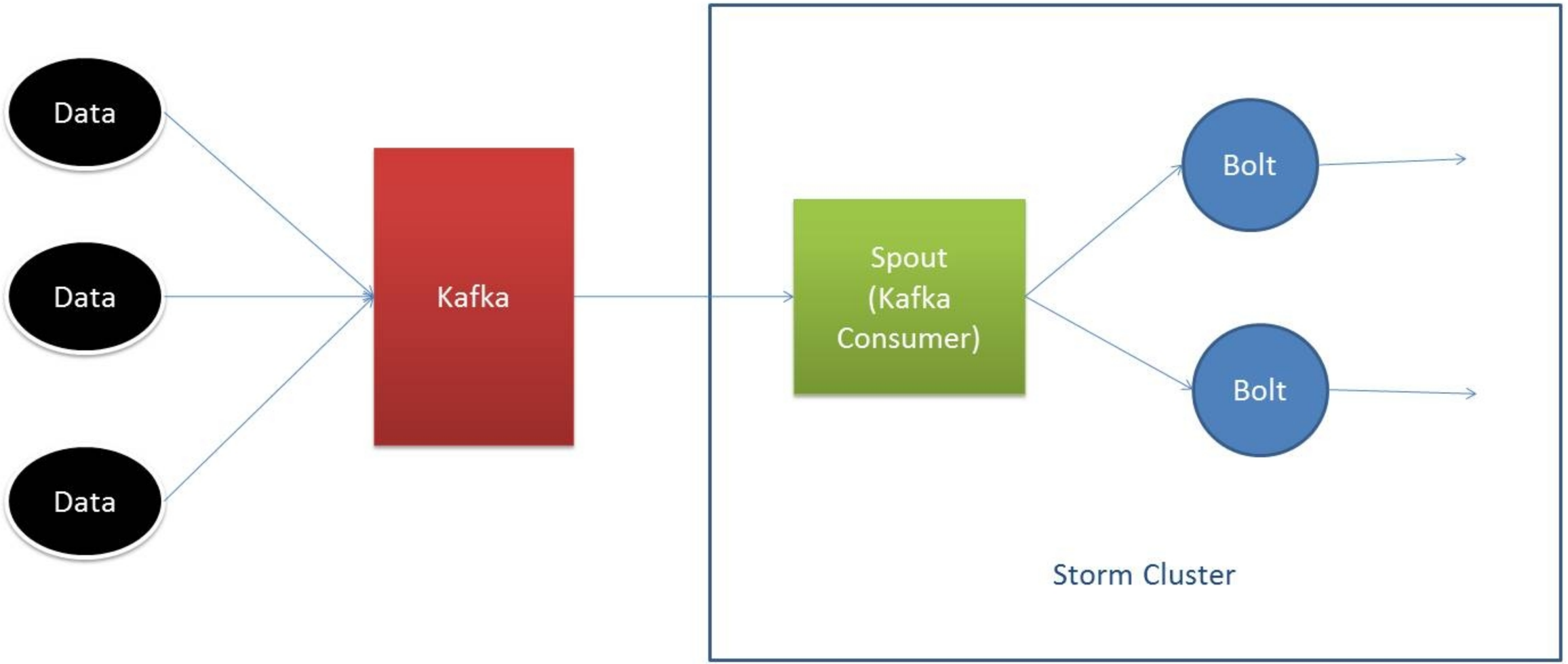
Apache Storm

- Apache 2.0 license
- Written in closure and API are exposed in Java
- Master/Slave architecture
- Rich community
- Easy to operate:
  - Storm is much easy to deploy and manage.
- Fast:
  - Storm Cluster can process **billion of records** per second

Kafka with Storm

- Consider, we have a real time app handling high volume data.
- Storm Spout doesn’t buffer/Queue the data.
- We would require external buffer/Queue for storing that data.
- Kafka is best choice for Queuing high volume data.
- Storm will read the data from Kafka and applies some required manipulation.

Kafka with Storm



DataFlair Web Services Pvt Ltd

+91-8451097879
info@data-flair.com
http://data-flair.com