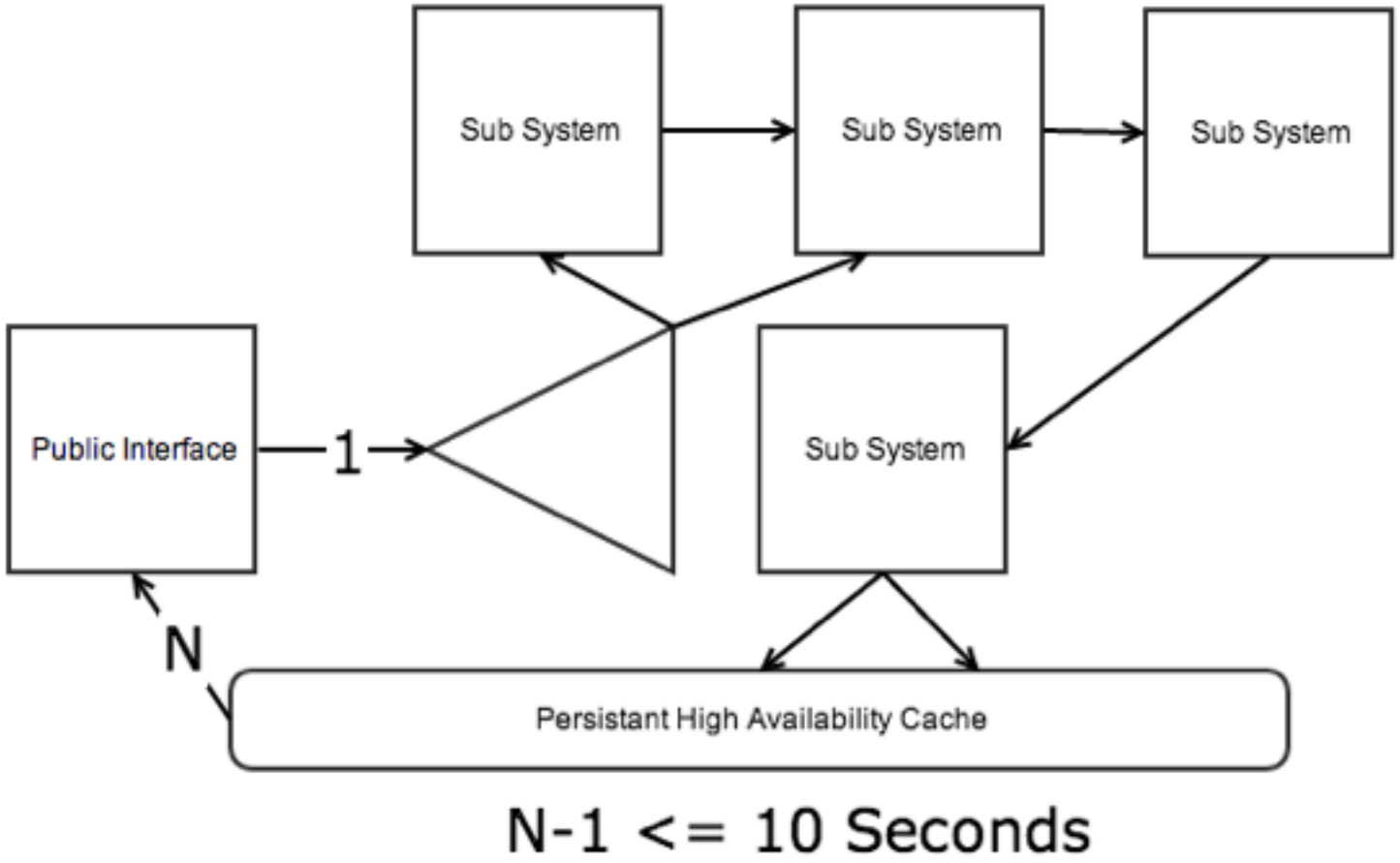
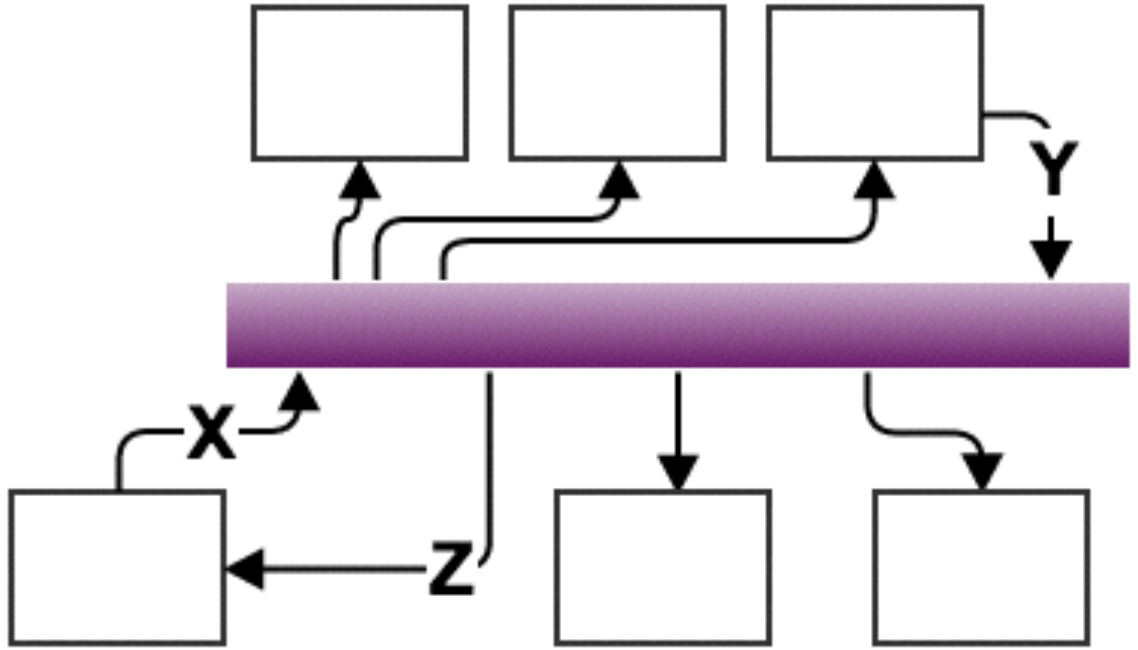
REAL-TIME STREAMING AND DATA PIPELINES WITH APACHE KAFKA

Real-time Streaming



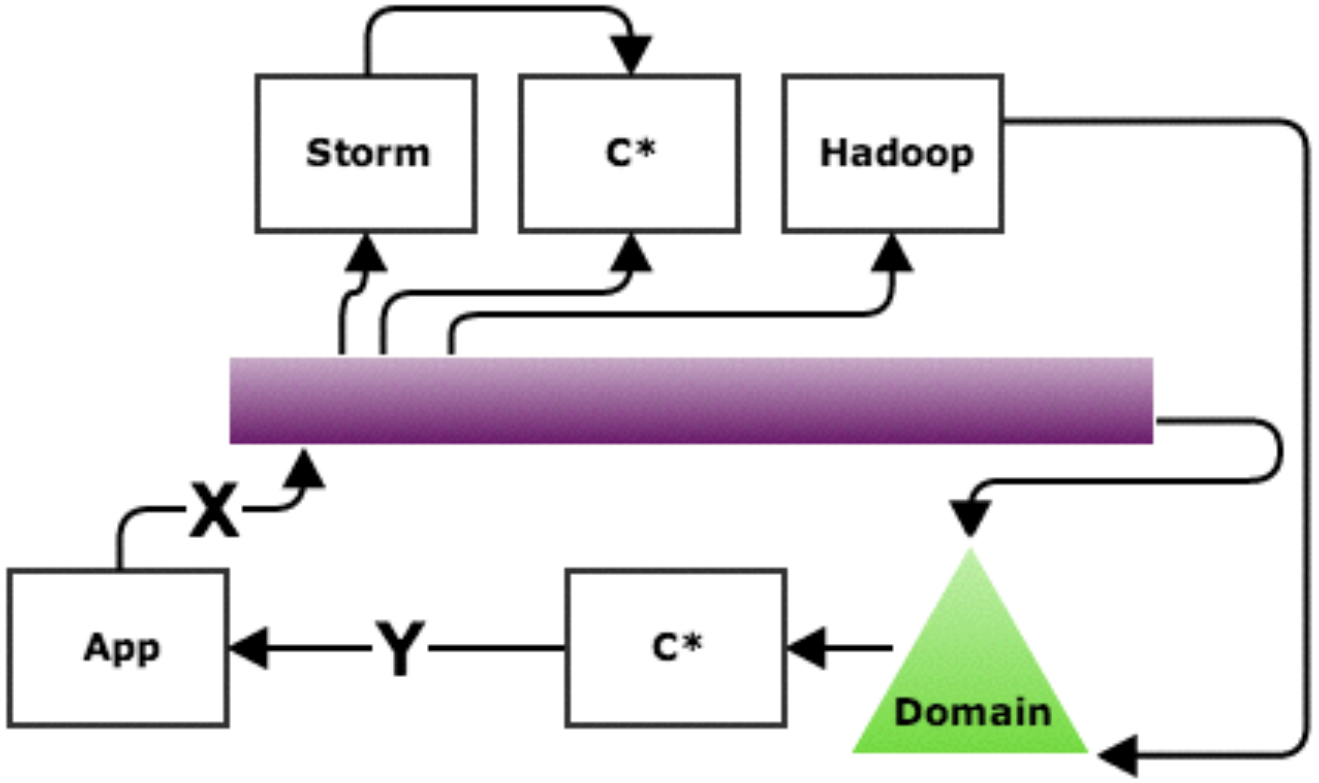
Real-time Streaming with Data Pipelines



Sync
- Z-Y < 20ms
- Y-X < 20ms

Async
- Z-Y < 1ms
- Y-X < 1ms

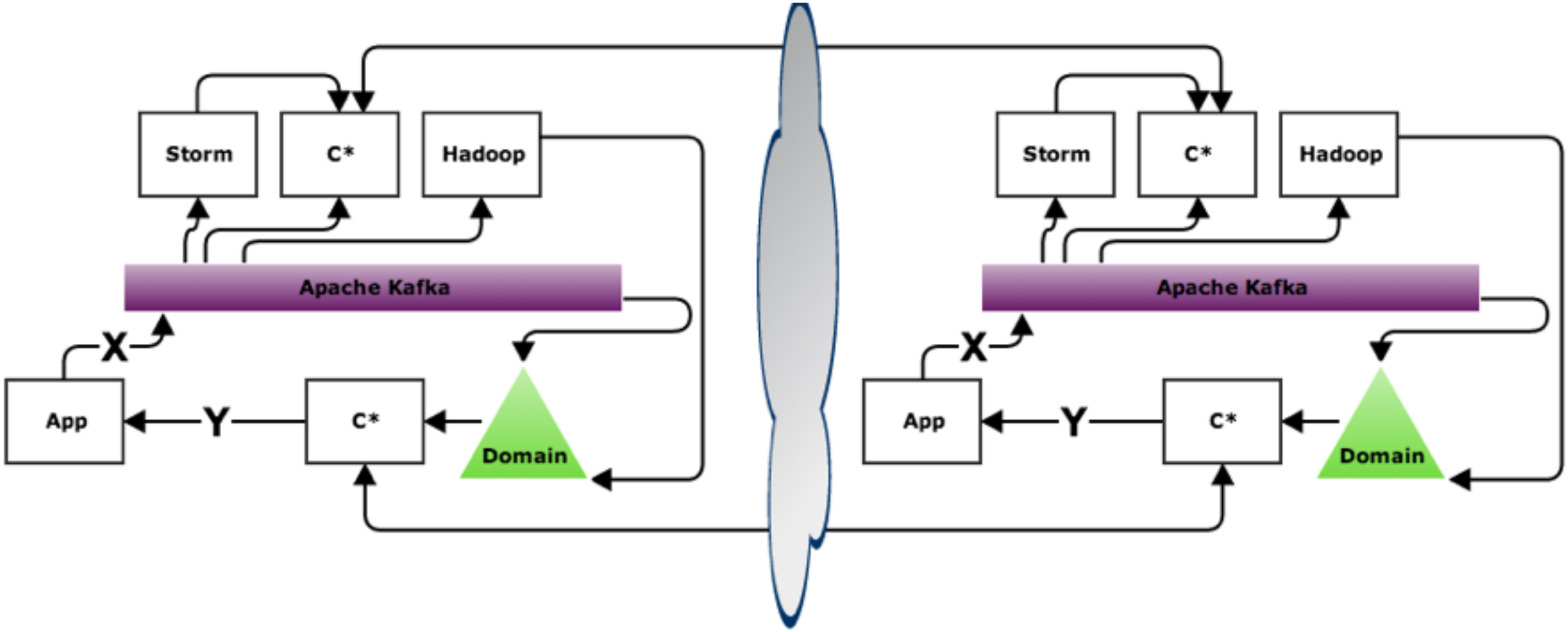
Real-time Streaming with Data Pipelines



Sync
Y-X < 20ms

Async
Y-X < 1ms

Real-time Streaming with Data Pipelines



Before we get started

- Samples
  - https://github.com/stealthly/scala-kafka
- Apache Kafka 0.8.0 Source Code
  - https://github.com/apache/kafka
- Docs
  - http://kafka.apache.org/documentation.html
- Wiki
  - https://cwiki.apache.org/confluence/display/KAFKA/Index
- Presentations
  - https://cwiki.apache.org/confluence/display/KAFKA/Kafka+papers+and+presentations
- Tools
  - https://cwiki.apache.org/confluence/display/KAFKA/Replication+tools
  - https://github.com/apache/kafka/tree/0.8/core/src/main/scala/kafka/tools

Apache Kafka

- Fast - A single Kafka broker can handle hundreds of megabytes of reads and writes per second from thousands of clients.
- Scalable - Kafka is designed to allow a single cluster to serve as the central data backbone for a large organization. It can be elastically and transparently expanded without downtime. Data streams are partitioned and spread over a cluster of machines to allow data streams larger than the capability of any single machine and to allow clusters of co-ordinated consumers
- Durable - Messages are persisted on disk and replicated within the cluster to prevent data loss. Each broker can handle terabytes of messages without performance impact.
- Distributed by Design - Kafka has a modern cluster-centric design that offers strong durability and fault-tolerance guarantees.

Up and running with Apache Kafka

1) Install Vagrant http://www.vagrantup.com/
2) Install Virtual Box https://www.virtualbox.org/
3) git clone https://github.com/stealthly/scala-kafka
4) cd scala-kafka
5) vagrant up

Zookeeper will be running on 192.168.86.5

BrokerOne will be running on 192.168.86.10

All the tests in ./src/test/scala/* should pass (which is also /vagrant/src/test/scala/* in the vm)

6) ./sbt test
[success] Total time: 37 s, completed Dec 19, 2013 11:21:13 AM

Maven

```
<dependency>
        <groupId>org.apache.kafka</groupId>
                <artifactId>kafka_2.10</artifactId>
                <version>0.8.0</version>
                <exclusions>
                        <!-- https://issues.apache.org/jira/browse/KAFKA-1160 -->
                <exclusion>
                        <groupId>com.sun.jdmk</groupId>
                        <artifactId>jmxtools</artifactId>
                </exclusion>
                        <exclusion>
                        <groupId>com.sun.jmx</groupId>
                         <artifactId>jmxri</artifactId>
                 </exclusion>
          </exclusions>
</dependency>
```

SBT

"org.apache.kafka" % "kafka_2.10" % "0.8.0“ intransitive()

Or

libraryDependencies ++= Seq(

  "org.apache.kafka" % "kafka_2.10" % "0.8.0",

)

https://issues.apache.org/jira/browse/KAFKA-1160

https://github.com/apache/kafka/blob/0.8/project/Build.scala?source=c

Apache Kafka

Producers	Consumers	Brokers
▪ Topics ▪ Brokers ▪ Sync Producers ▪ Async Producers ▪ (Async/Sync)=> Akka Producers	▪ Topics ▪ Partitions ▪ Read from the start ▪ Read from where last left off	▪ Partitions ▪ Load Balancing Producers ▪ Load Balancing Consumer ▪ In Sync Replicaset (ISR) ▪ Client API

Producers

- Topics
- Brokers
- Sync Producers
- Async Producers
- (Async/Sync)=> Akka Producers

Producer

```
/** at least one of these for every partition **/
val producer = new KafkaProducer(“topic","192.168.86.10:9092")
producer.sendMessage(testMessage)
```

Kafka Producer

wrapper for core/src/main/kafka/producer/Producer.Scala

```
case class KafkaProducer(
  topic: String,
  brokerList: String,
  synchronously: Boolean = true,
  compress: Boolean = true,
  batchSize: Integer = 200,
  messageSendMaxRetries: Integer = 3
) {

  val props = new Properties()
  val codec = if(compress) DefaultCompressionCodec.codec else NoCompressionCodec.codec
  props.put("compression.codec", codec.toString)
  props.put("producer.type", if(synchronously) "sync" else "async")
  props.put("metadata.broker.list", brokerList)
  props.put("batch.num.messages", batchSize.toString)
  props.put("message.send.max.retries", messageSendMaxRetries.toString)

  val producer = new Producer[AnyRef, AnyRef](new ProducerConfig(props))

  def sendMessage(message: String) = {

    try {
      producer.send(new KeyedMessage(topic,message.getBytes))
    } catch {
      case e: Exception =>
        e.printStackTrace
        System.exit(1)
    }
  }
}
```

Producer Config

Docs

http://kafka.apache.org/documentation.html#producerconfigs

Source https://github.com/apache/kafka/blob/0.8/core/src/main/scala/kafka/producer/ProducerConfig.scala?source=c

Akka Producer

```
class KafkaAkkaProducer extends Actor with Logging {


  private val producerCreated = new AtomicBoolean(false)
  var producer: KafkaProducer = null

  def init(topic: String, zklist: String) = {
    if (!producerCreated.getAndSet(true)) {
      producer = new KafkaProducer(topic,zklist)
  }


  def receive = {
    case t: (String, String) ⇒ {
        init(t._1, t._2)
    }
    case msg: String ⇒ {
        producer.sendString(msg)
    }
  }
}
```

Consumers

- Topics
- Partitions
- Read from the start
- Read from where last left off

```
class KafkaConsumer(
    topic: String,
    groupId: String,
    zookeeperConnect: String,
    readFromStartOfStream: Boolean = true
  ) extends Logging {

  val props = new Properties()
  props.put("group.id", groupId)
  props.put("zookeeper.connect", zookeeperConnect)
  props.put("auto.offset.reset", if(readFromStartOfStream) "smallest" else "largest")

  val config = new ConsumerConfig(props)
  val connector = Consumer.create(config)
  val filterSpec = new Whitelist(topic)
  val stream = connector.createMessageStreamsByFilter(filterSpec, 1, new DefaultDecoder(), new DefaultDecoder()).get(0)


def read(write: (Array[Byte])=>Unit) = {
  for(messageAndTopic <- stream) {
      write(messageAndTopic.message)
    }
 }
 def close() {
    connector.shutdown()
 }
}
```

Low Level Consumer

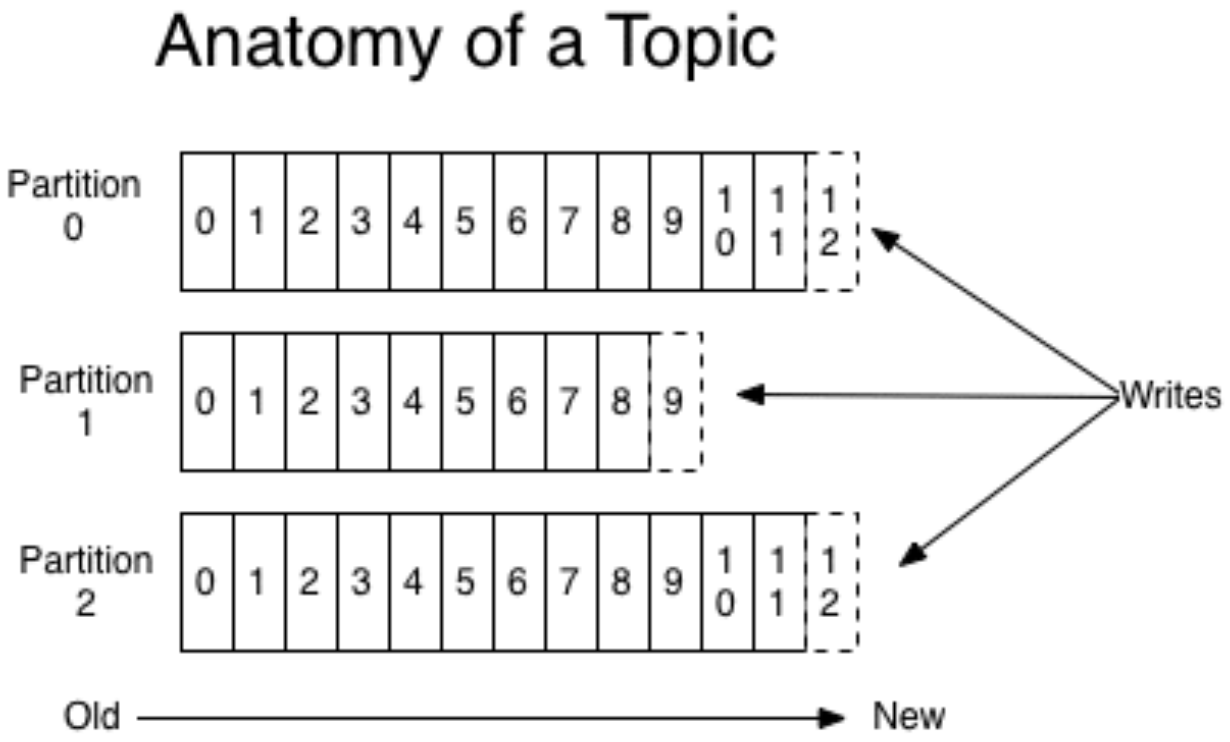
Source Sample https://github.com/apache/kafka/blob/0.8/core/src/main/scala/kafka/tools/SimpleConsumerShell.scala?source=c

```
val topic = “publisher”
val consumer = new KafkaConsumer(topic ,”loop”,"192.168.86.5:2181")
val count = 2
val pdc = system.actorOf(Props[KafkaAkkaProducer].withRouter(RoundRobinRouter(count)), "router")
1 to countforeach { i =>(
 pdc ! (topic,"192.168.86.10:9092"))
}
def exec(binaryObject: Array[Byte]) = {
  val message = new String(binaryObject)
  pdc ! message
}
pdc ! “go, go gadget stream 1”
pdc ! “go, go gadget stream 2”
consumer.read(exec)
```

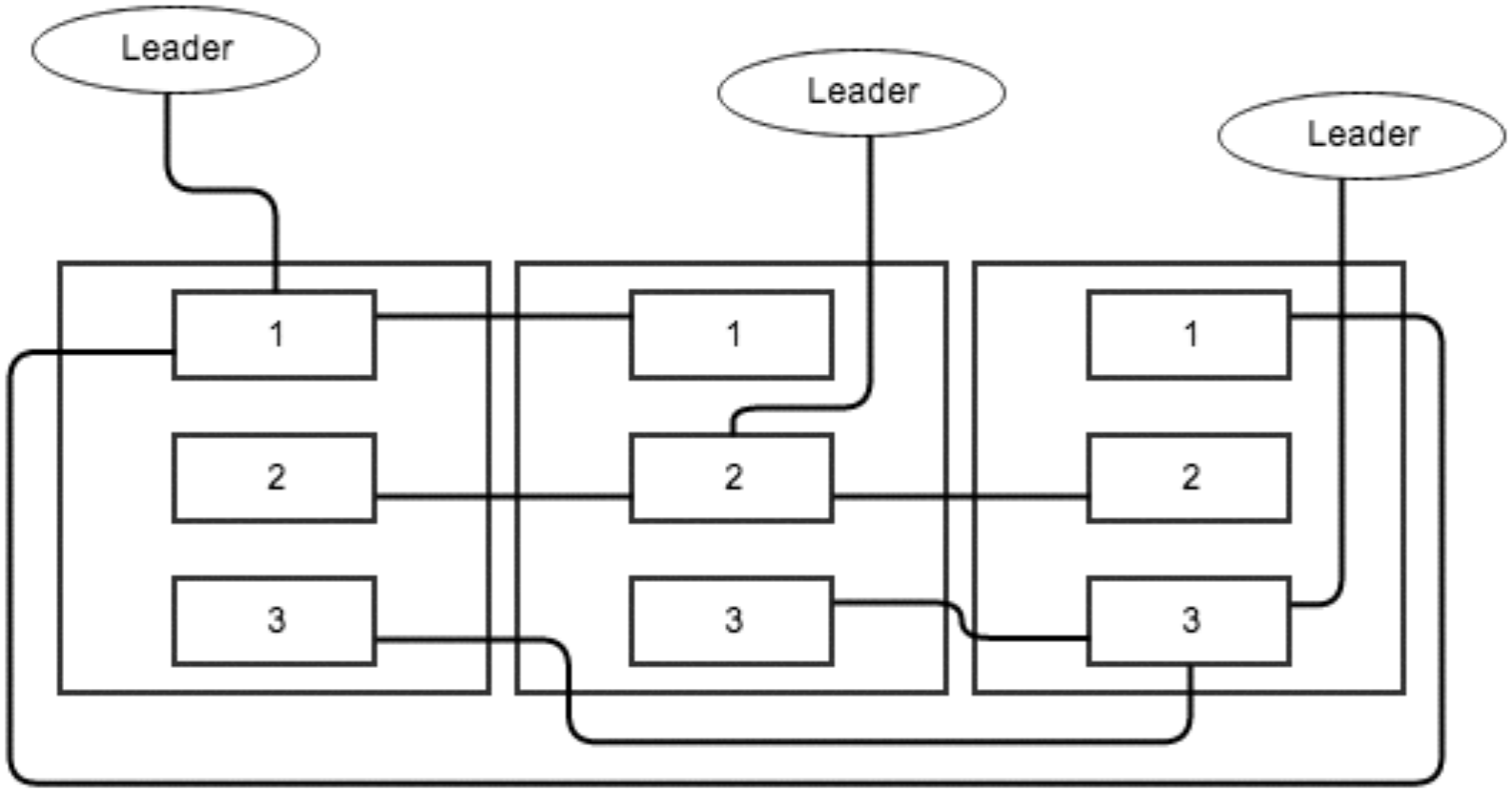
Brokers

- Partitions
- Load Balancing Producers
- Load Balancing Consumer
- In Sync Replicaset (ISR)
- Client API

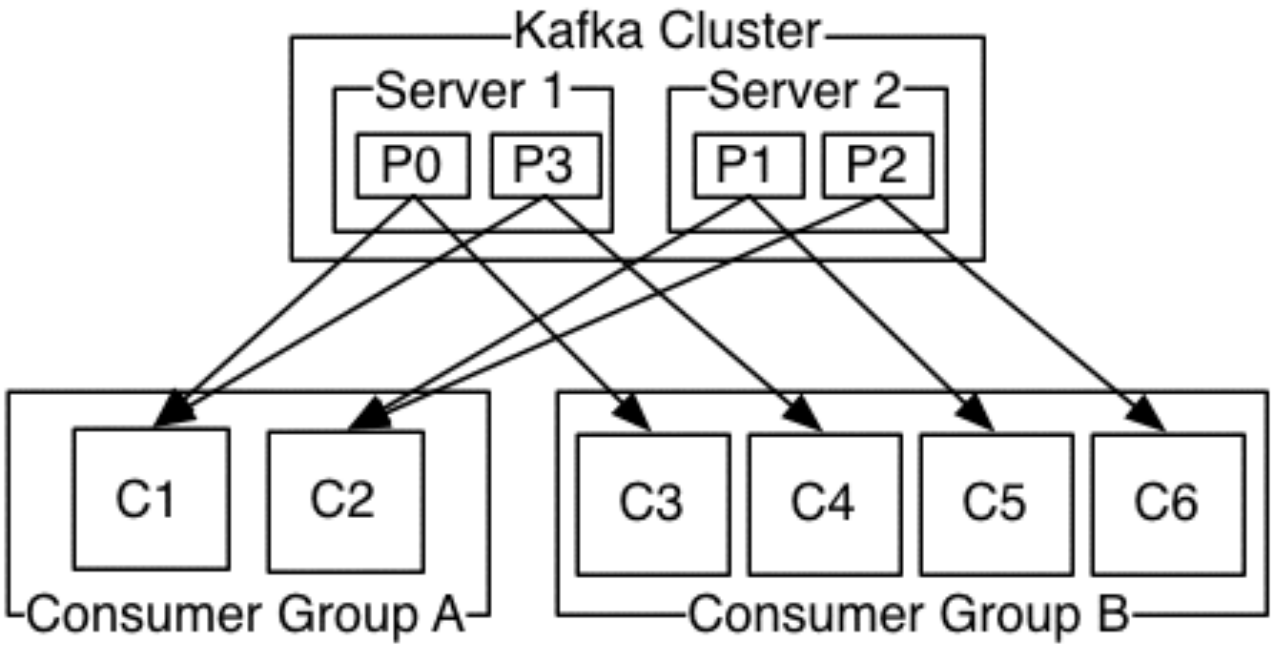
Partitions make up a Topic



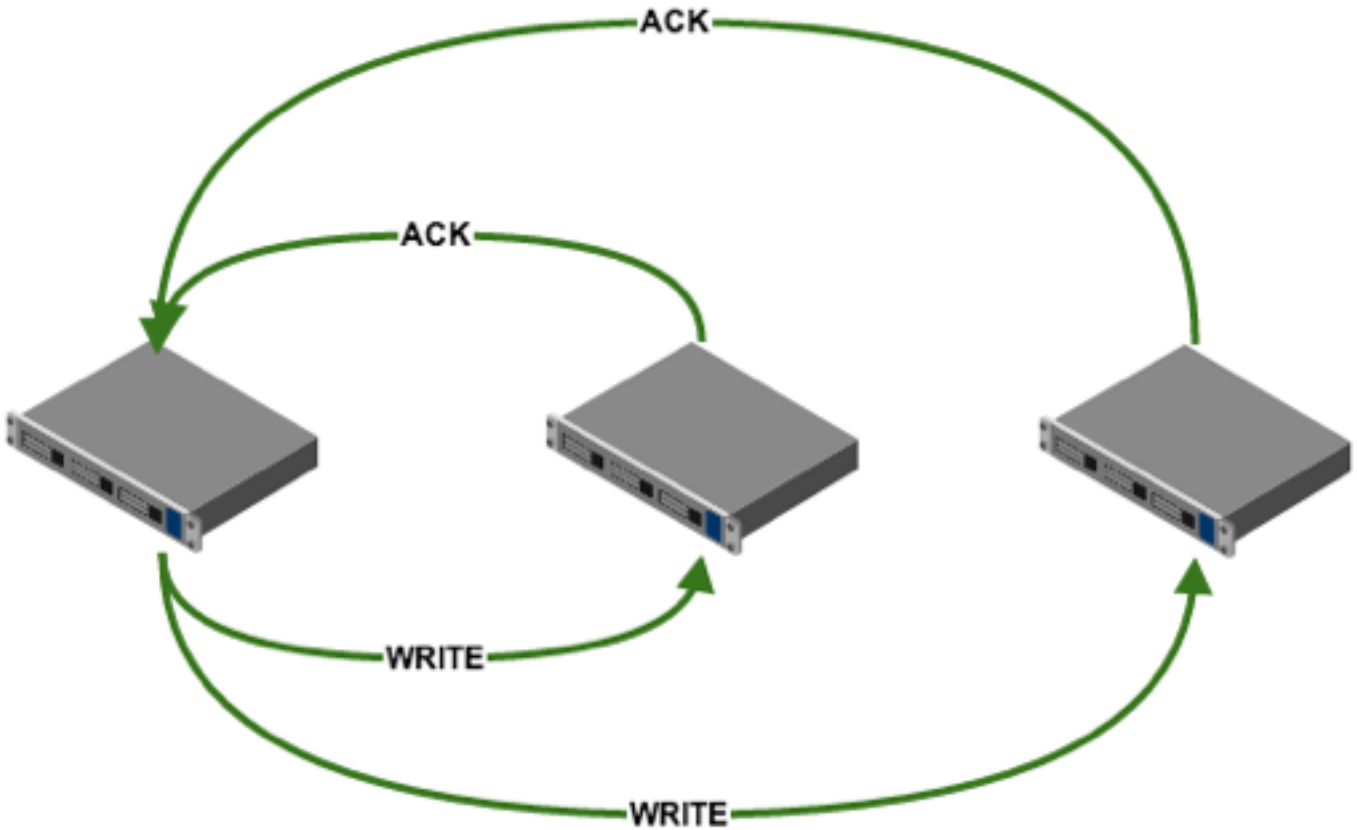
Load Balancing Producer By Partition



Load Balancing Consumer By Partition



Replication – In Sync Replicas



Client API

- Developers Guide https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol
- Non-JVM Clients https://cwiki.apache.org/confluence/display/KAFKA/Clients
  - Python
    - Pure Python implementation with full protocol support. Consumer and Producer implementations included, GZIP and Snappy compression supported.
  - C
    - High performance C library with full protocol support
  - C++
    - Native C++ library with protocol support for Metadata, Produce, Fetch, and Offset.
  - Go (aka golang)
    - Pure Go implementation with full protocol support. Consumer and Producer implementations included, GZIP and Snappy compression supported.
  - Ruby
    - Pure Ruby, Consumer and Producer implementations included, GZIP and Snappy compression supported. Ruby 1.9.3 and up (CI runs MRI 2.
  - Clojure
    - https://github.com/pingles/clj-kafka/ 0.8 branch

Questions?

/**********

Joe Stein

Founder, Principal Consultant

Big Data Open Source Security LLC

http://www.stealth.ly

Twitter: @allthingshadoop

**********/