Apache Kafkaによる
スケーラブル
アプリケーション開発 <V2>

yuuki takezawa <ytake>
Developers Summit 2018 - 2/16

Agenda

- What Apache Kafka
- Kafka Connect / Kafka Streams
- Kappa Architecture
- アプリケーションで活用

Scalability

- 規模透過性

  負荷の高低に合わせてリソース・プールを

  拡大・縮小できること
- 位置透過性

  ユーザーやリソースがどれだけ離れているか意識せずに、

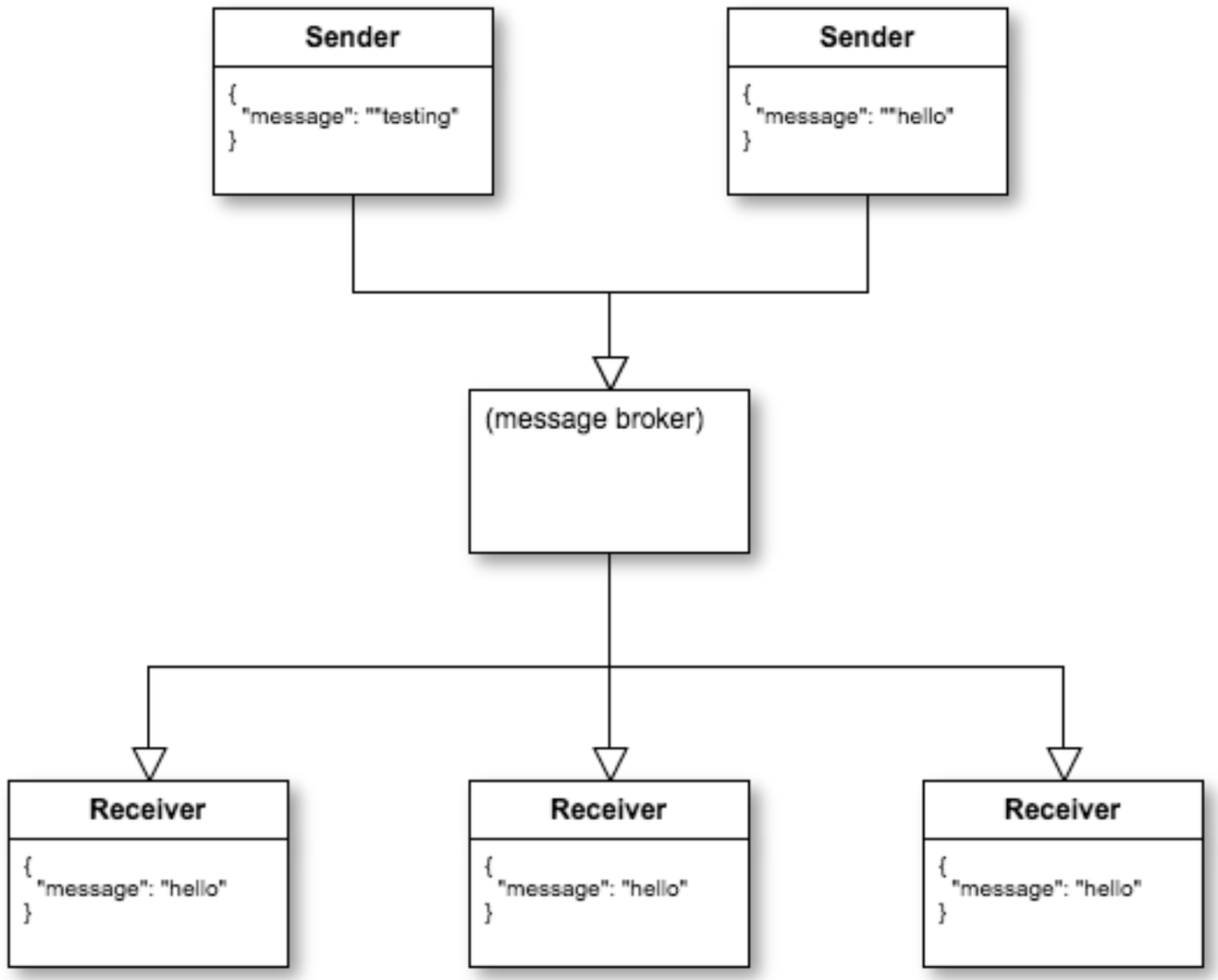
  変わらない使い勝手でシステムが利用できること
- 異種透過性

  システムを構成する機器やソフトウェアが異なっていること

  とを意識せずに管理・利用できること

What is Apache Kafka?

Message Broker



Apache Kafka

- Zookeeperを利用したクラスタリングによる高可用性
- メッセージの永続化、レプリケーション、再取得可
- ビッグデータ対応
- ファイルシステム利用で、
  シーケンシャルアクセスによる高速化
- ストリーム対応のメッセージングミドルウェア
- Kafka Connectによる周辺システムとの高い親和性

  (Amazon kinesisとほぼ同じ)

Apache Kafka + ZooKeeper Architecture

Apache Kafka概要

- Producer

  メッセージ配信を行う

  各言語のクライアントライブラリを利用

- Consumer

  メッセージ購読を行う

  消費されたメッセージは破棄されず、

  一定期間保管される

- Broker

  Kafka本体で、Producer、Consumer間のキュー

Apache Kafka概要

- Topic

  Producerからのメッセージはこのtopicに格納される

  メッセージは一意に管理、FIFO(後述partition)で処理

- Partition

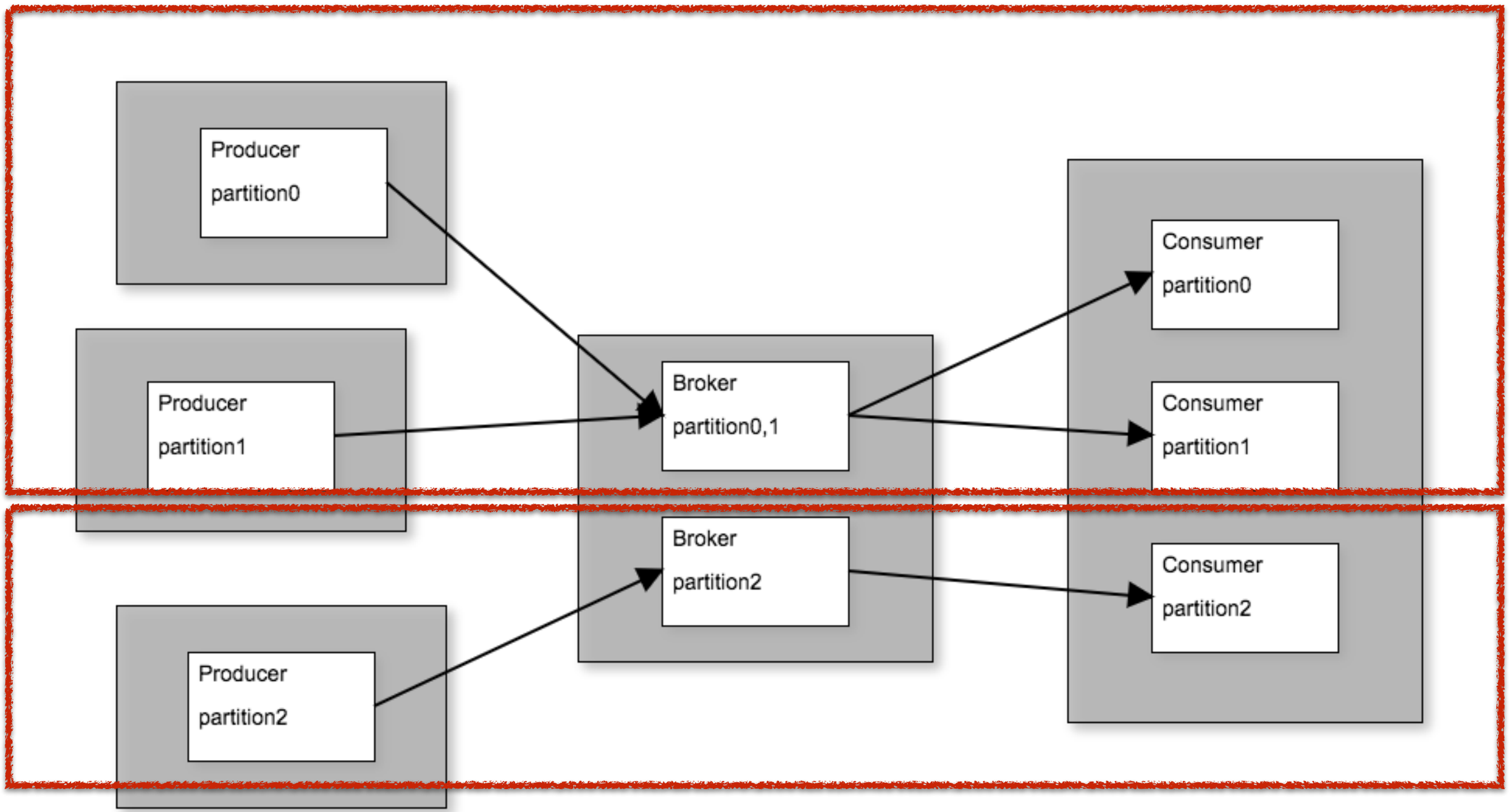
  負荷分散用途に利用

  複数のConsumerがそれぞれのPartitionを参照し、

  それぞれが処理を行う

  処理フローのデザインによって多様な利用方法

Example Partition



Producer サンプル

https://github.com/istyle-inc/go-example-developers-summit

```
import (
    "github.com/confluentinc/confluent-kafka-go/kafka"
)

func (kc *KafkaClient) PrepareProducer() {
    p, err := kafka.NewProducer(&kafka.ConfigMap{
        "bootstrap.servers":          *kc.BootstrapServers,
        "broker.address.family":      "v4",
        "queue.buffering.max.messages": "1000",
        "client.id":                  "testingClient",
    })
    // 省略
}
```

接続するKafkaを指定

Kafka Connect

- Kafka Connectとは、

  周辺システムからのデータを取り込み(Source)、

  データ送信(Sink)の二種類をサポートする機能

- Amazon SQSやMongoDBのデータをKafkaで取込む、

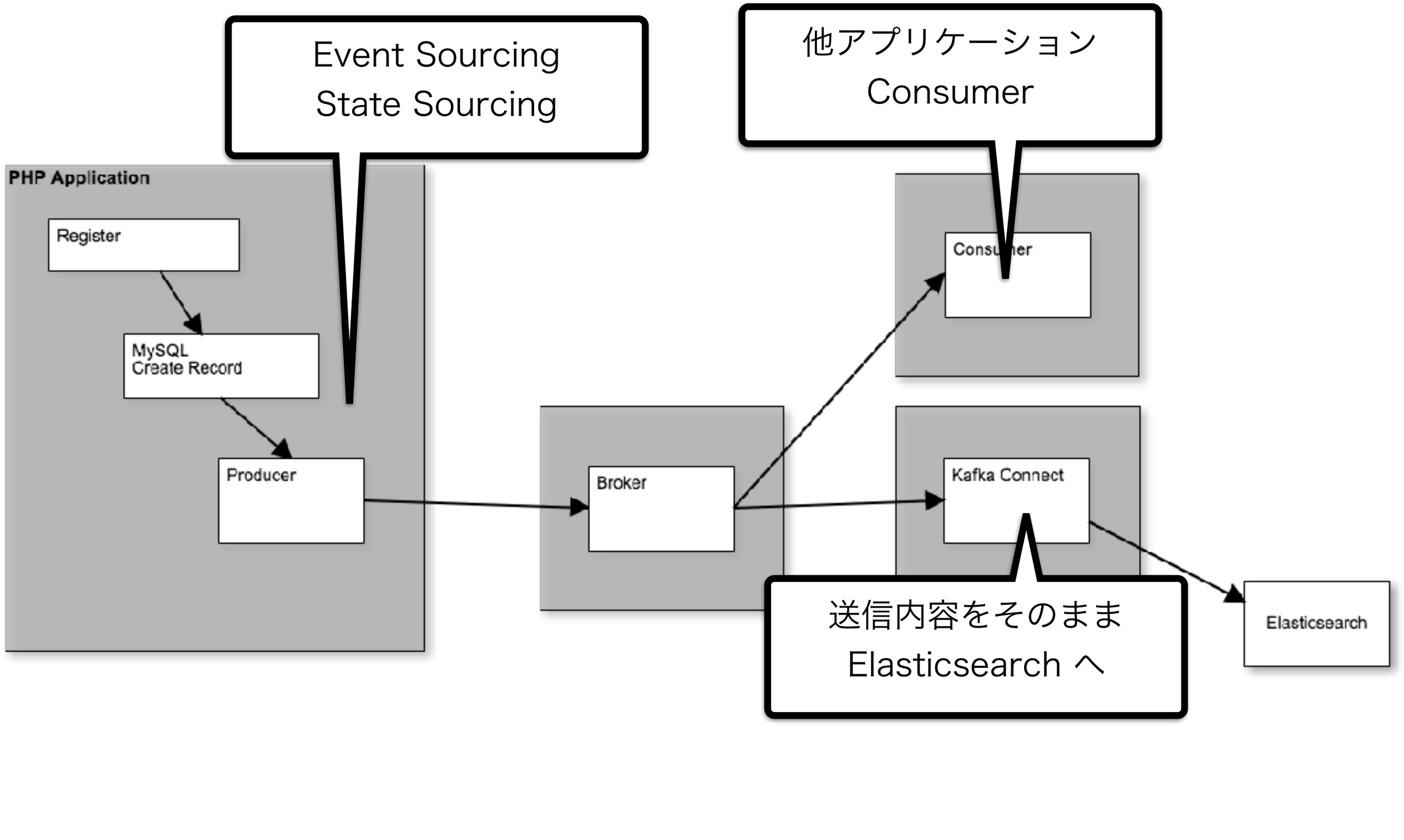
  メッセージをそのままElasticsearchやRDBMSに格納、

  が行える

- Connectは自由に拡張して独自Connectを実装可能

  (java, Scala)

Example Kafka Connect



Kafka Connect Sink - Elasticsearch

```
bootstrap.servers=192.168.10.10:9092
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false

internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000

rest.port=8093
plugin.path=/usr/share/java/
```

elasticsearch connect

```
name=elasticsearch-sink
connector.class=io.confluent.connect.elasticsearch.ElasticsearchSinkConnector
tasks.max=1
topics=sample.append_employees
key.ignore=true
connection.url=http://localhost:9200

schema.ignore=true
type.name=kafka-connect
```

対象のtopicを選択

connectを登録

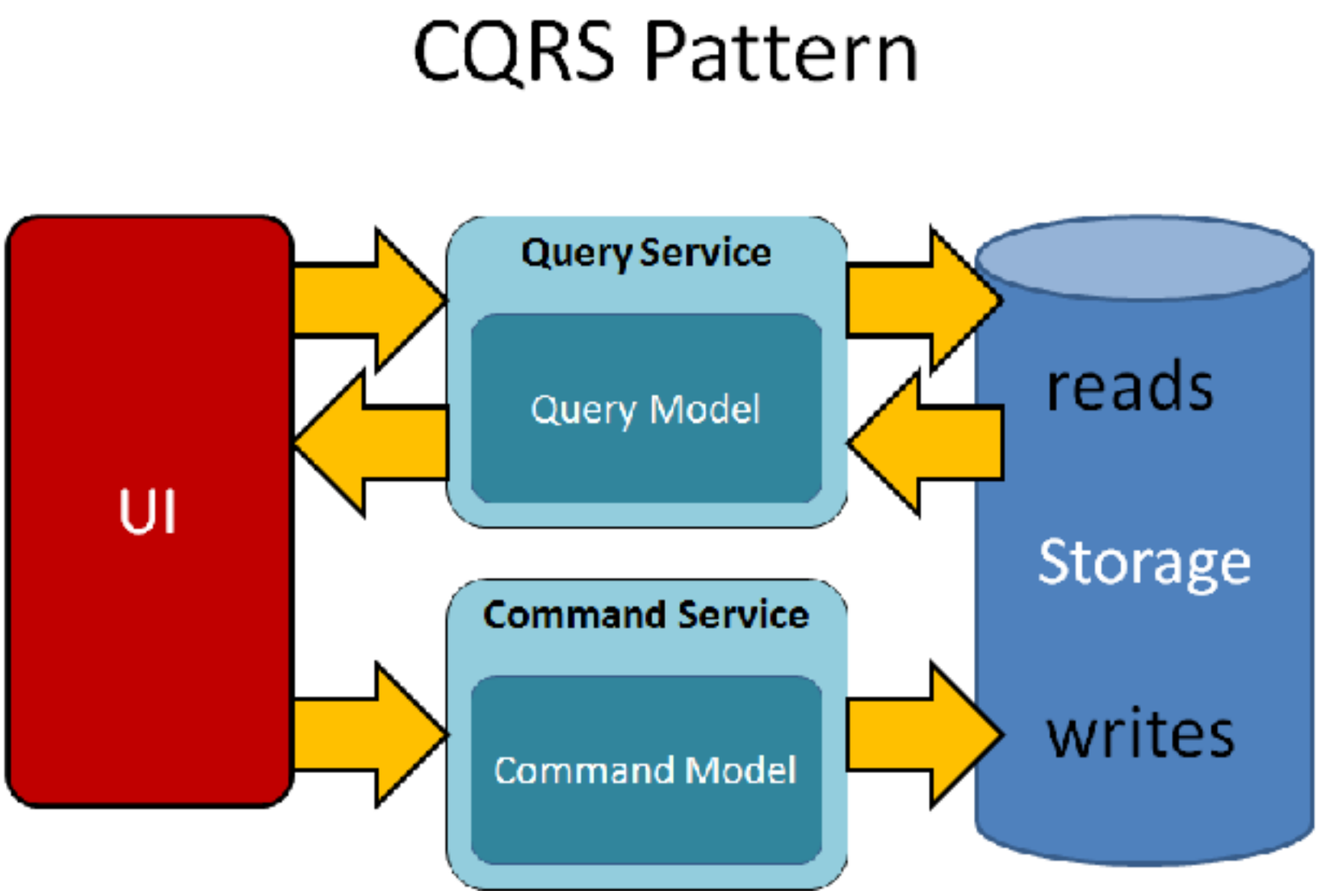
```
$ confluent load elasticsearch-sink -d \
 /etc/kafka-connect-elasticsearch/sample-es.properties
```

```
$ connect-standalone \
-daemon /etc/schema-registry/elasticsearch-connect.properties  \
/etc/kafka-connect-elasticsearch/sample-es.properties
```

転送が開始されます

```
"hits": {
  "total": 50,
  "max_score": 1,
  "hits": [
    {
      "_index": "sample.append_employees",
      "_type": "kafka-connect",
      "_id": "sample.append_employees+0+1",
      "_score": 1,
      "_source": {
        "Designation": "President",
        "Salary": "110000",
        "MaritalStatus": "Unmarried",
        "DateOfJoining": "2013-02-07",
        "Address": "16 Manor Station Court Huntsville, AL 35803",
        "FirstName": "JENNEFER",
        "LastName": "WENIG",
        "Gender": "Female",
        "Age": 45,
        "Interests": "String Figures,Working on cars,Button Collecting,Surf Fishing
,Developers Summit"
      }
    },
    {
      "_index": "sample.append_employees",
      "_type": "kafka-connect",
      "_id": "sample.append_employees+0+4",
      "_score": 1,
```

Command Query Responsibility Segregation



Kafka Streams

- Kafkaでストリーム処理を実装する機能

- あるトピックにデータが格納される時に
  何か処理を実行し他のトピックに格納する

- Consumerなしで上記の処理を実現

- 直近30分におけるPVランキング、
  メッセージ内の文字列変更、追加など

What is Stream?

ストリーム処理

- 大量のデータをリアルタイムで処理するのが、

  ストリームデータ処理の目的
- 終わりがなく、無限にやってくるものへのアプローチ
- メモリ内で処理され、その後破棄される
- 監視系の処理よく利用されているもの
- センサーを利用したアプリケーションなど

Kafka Streams

- KStream / KTable
- KStreamはストリームで流れてくるデータがKVでそのまま流れてくる
- KStreamはシンプルにfilter処理や、置き換え用途

ログや、ツイート、タイムライン的なものに

簡単なサンプル

https://github.com/istyle-inc/developers-summit-kafka-streams

```
def main(args: Array[String]) {

  println("Kafka Streams Sample.")

  val config: Properties = {
    val prop = new Properties()
    prop.load(new java.io.FileInputStream(args(0).toString))
    prop.put(StreamsConfig.APPLICATION_ID_CONFIG, prop.getProperty("sample.app_id"))
    prop.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, prop.getProperty("sample.bootstrap.servers"))
    prop.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass)
    prop.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass)
    // exactly once
    prop.put(StreamsConfig.PROCESSING_GUARANTEE_CONFIG, StreamsConfig.EXACTLY_ONCE)
    prop
  }
}
```

kafka Streamで実行されるるdef

サービス全体で同じ app_id は使わない

接続するserverを指定

```
val ft: String = config.getProperty("sample.stream.topic")
val tt: String = config.getProperty("sample.streamed.topic")

println("stream topic: from " + ft)
println("to " + tt)

val stringSerde: Serde[String] = Serdes.String()
val builder: StreamsBuilder = new StreamsBuilder
val rawStream: KStream[String, String] = builder.stream(ft)
val mapStream: KStream[String, String] = rawStream.mapValues(new ValueMapper[String, String] {
  override def apply(value: String): String = new ElementAppender(value).append()
})
mapStream.to(stringSerde, stringSerde, tt)
val streams: KafkaStreams = new KafkaStreams(builder.build(), config)
streams.start()
```

Stream処理対象topic

Stream処理後に格納するtopc

KStreamで処理

データの置き換え

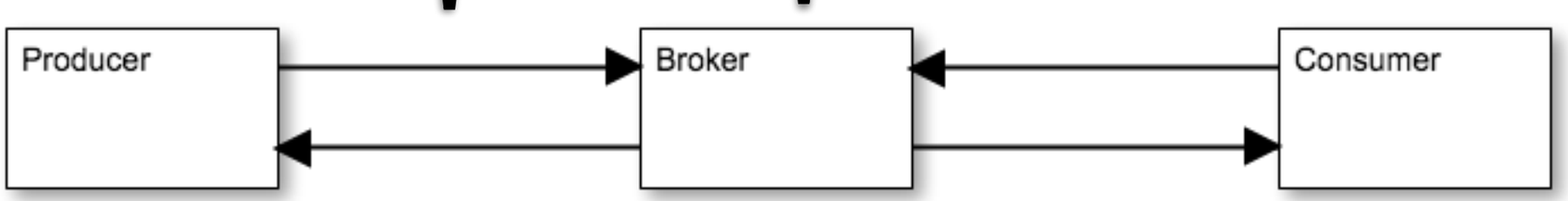
処理後にtopic格納を行う

Stream処理命令 継続的に実行される

メッセージの欠損

送信失敗

Brokerダウン



ack受信失敗

メッセージ受信失敗

Exactly Once

- 0.11から追加

- Producer, Broker間のトランザクション
  Broker, Consumer間のトランザクション

- 正確に一度だけ送信

メッセージ伝達の保証

- At least once semantics
  重複を許可

- At most once semantics
  欠損を許可

- Exactly once semantics
  重複、および欠損を許可しない

"Exactly-once Semantics are Possible: Here’s How Kafka Does it". Confluent APACHE KAFKA. https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/ 参照

Apache Kafkaで課題解決

- 流れてくるデータに合わせて

  開発者が分散方法を設計することができる

- ログのトランザクション

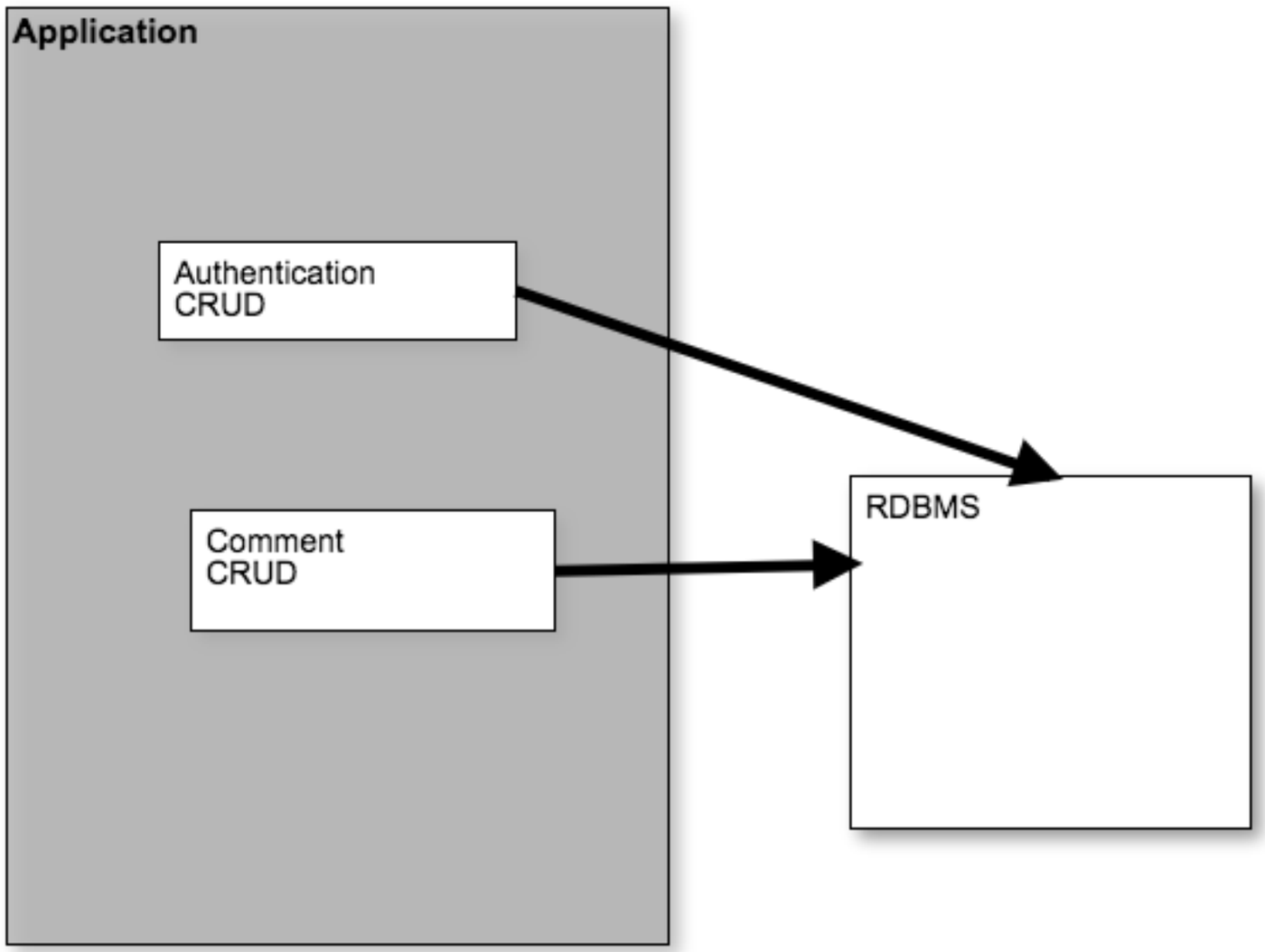
  ES・CQRSでアプリケーションに取り込むことが可能

- Kafka Connect、Kafka Streamsが強力

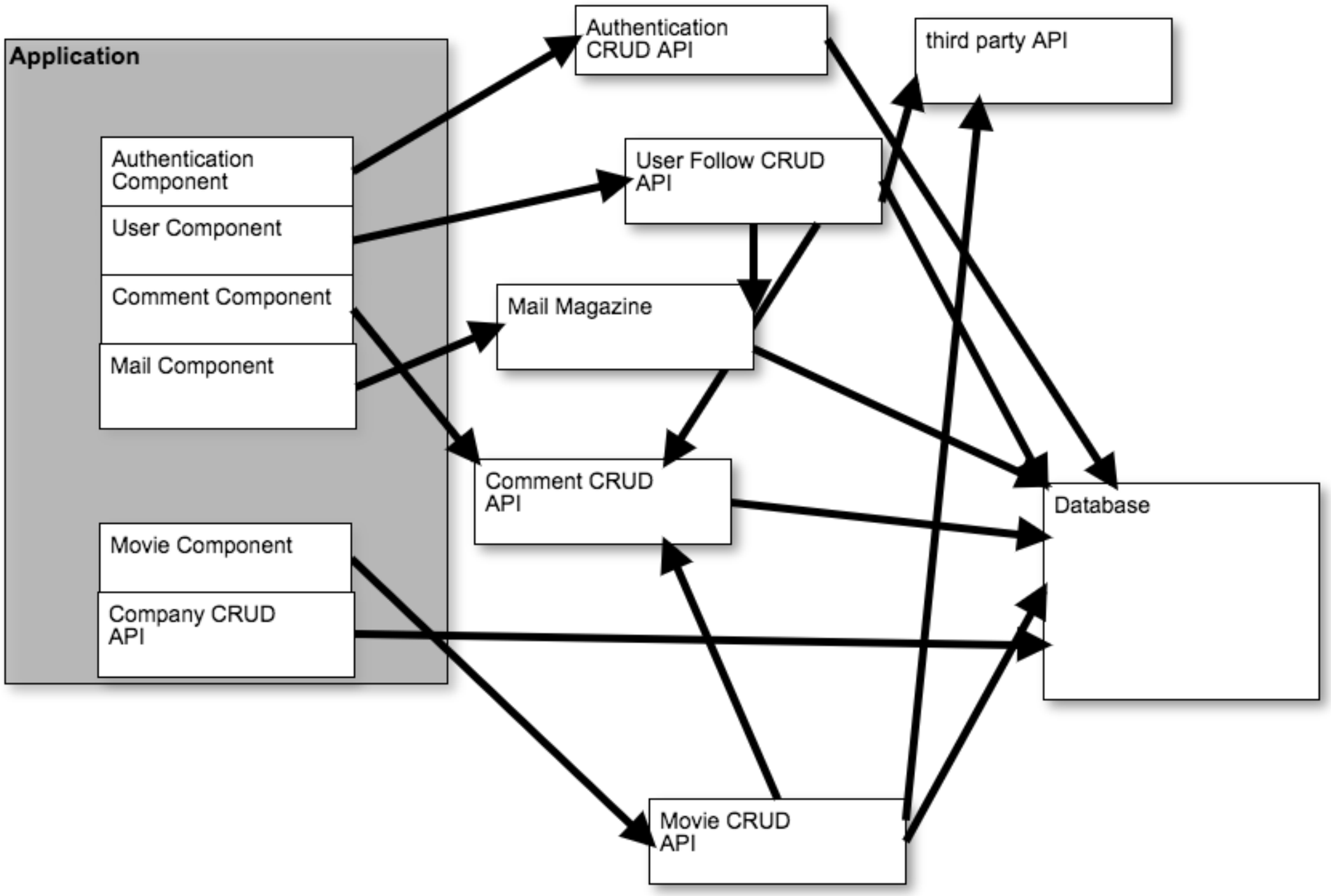
- Hadoopエコシステムで親和性

巨大化するアプリケーションに

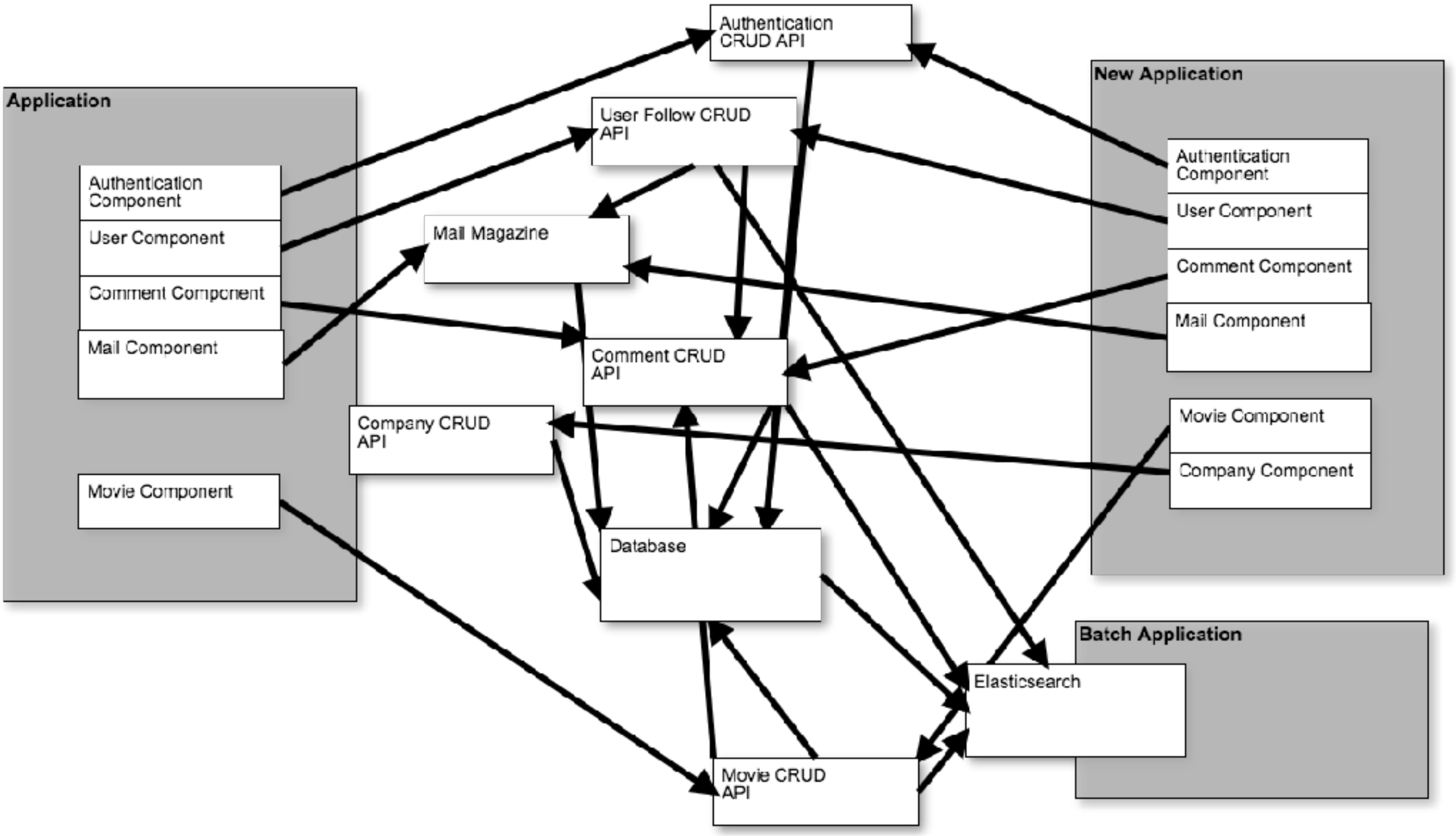
シンプルなアプリケーション構成



シンプルなアプリケーション構成



巨大なアプリケーションへ



複雑になるアプリケーション

- データ同期の複雑化

  リアルタイムでデータが欲しい

  インデックス更新が定期実行

- その他

  どこかのサービスが修正されたら障害に

- 複雑化する実装コード

背景

- 大きく複雑化するアプリケーションの責務

  -> **限られた機能のみ提供する**

  -> マイクロサービスよりにする必要があった

- 如何にスケールさせていくか

  -> いいハードウェアで頑張るにも限界がある

- 高可用・障害耐性

  -> データ損失担保、リトライなどが可能なものが！

通知機能に導入

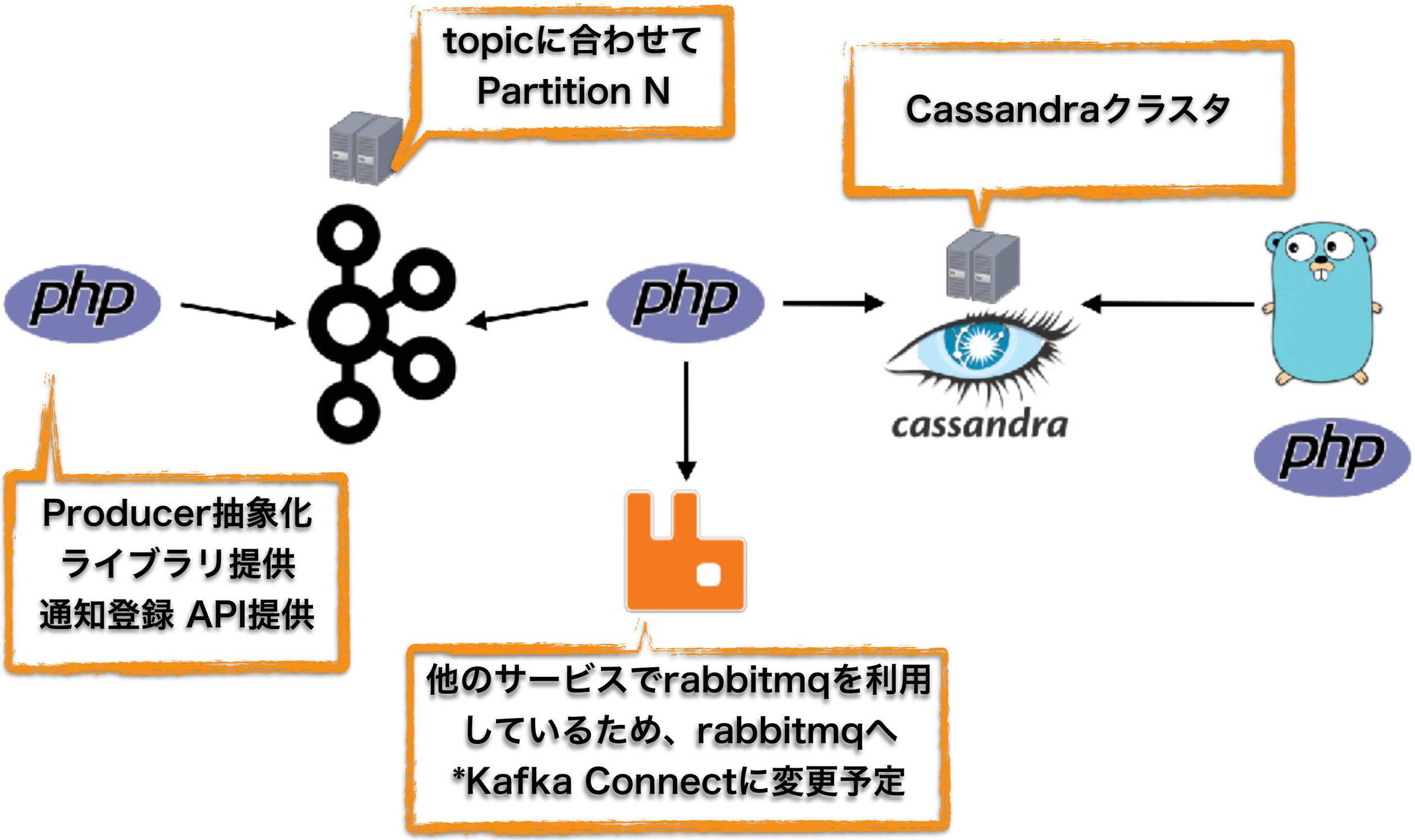
- ななにさんがどこで何しました

  何ポイント獲得しました！ etc
- サービス横断で使われる
- 数100万分のEvent (多いとき)
- データ加工にRDBMSと組み合わせる必要があり、

  サービスによっては複雑になるものもあるため、

  Consumerの負荷分散を容易にしたい
- メッセージ損失をなるべく防ぎたい

通知機能


topicに合わせて
Partition N
Cassandraクラスタ
php
php
cassandra
php
Producer抽象化
ライブラリ提供
通知登録 API提供
他のサービスでrabbitmqを利用
しているため、rabbitmqへ
*Kafka Connectに変更予定

導入後

- 一年間のうち目立った障害はなし

- メッセージ確認が容易で、疎通確認など重宝

- Kafka、Cassandraの組み合わせで捌けすぎてしまい、

  いつもスカスカ

- rabbitmqのバックアップとしても稼働

  *社内でrabbitmq利用サービスの方が多いため

RabbitMQ

APACHE kafka™

A distributed streaming platform

Kafka Connect Source - Rabbitmq

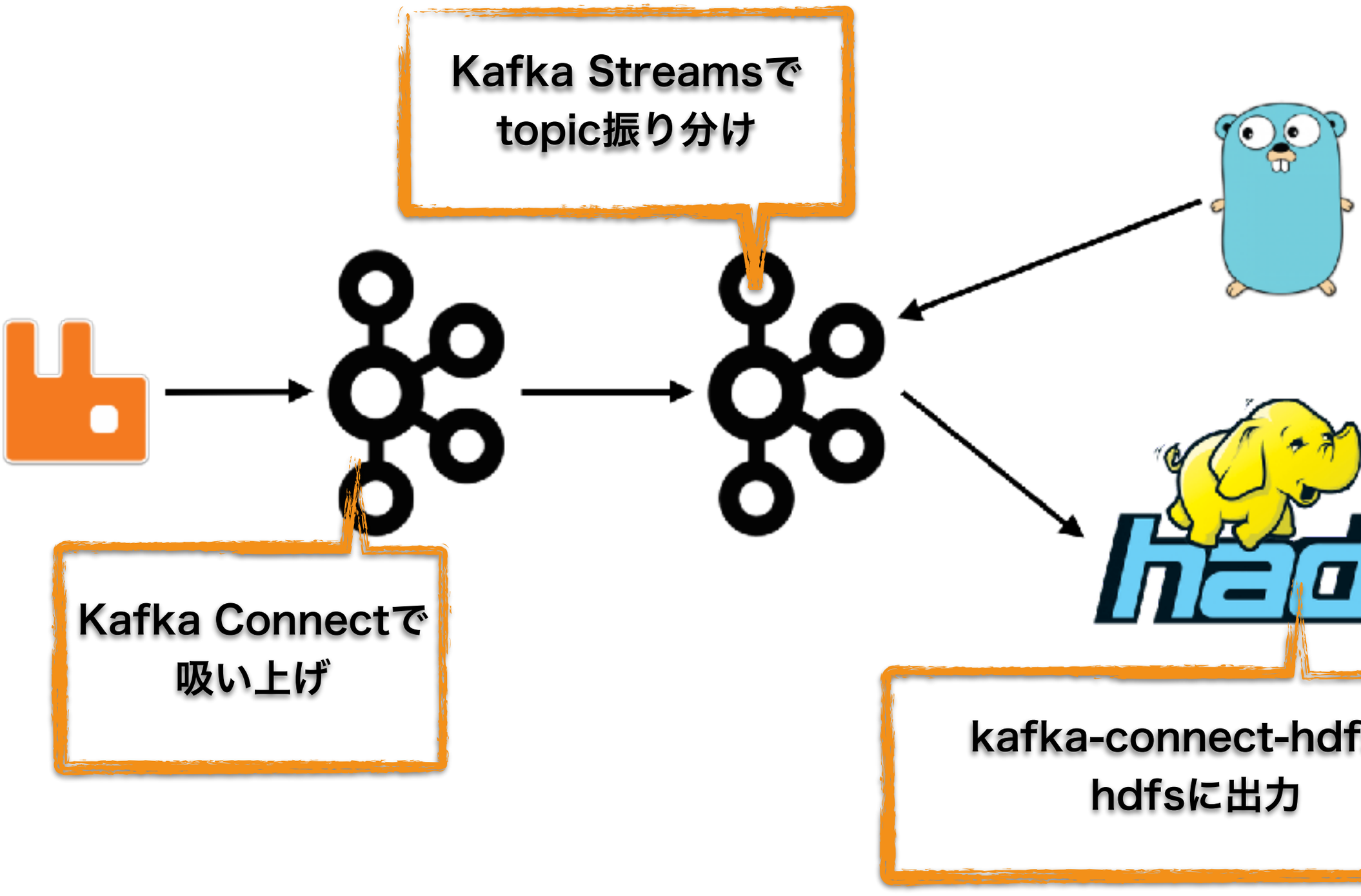
```
name=named-kafka-connect-rabbitmq
tasks.max=1
connector.class=com.github.jcustenborder.kafka.connect.rabbitmq.RabbitMQSour
ceConnector
rabbitmq.prefetch.count=500
rabbitmq.automatic.recovery.enabled=true
rabbitmq.network.recovery.interval.ms=10000
rabbitmq.topology.recovery.enabled=true
rabbitmq.queue=action_api.create_object
kafka.topic=rabbitmq.source_topic

rabbitmq.host=rabbitmqhost.local
rabbitmq.password=password
rabbitmq.username=username
```

```
$ confluent load named-kafka-connect-rabbitmq -d \
 /etc/kafka-connect-rabbitmq/rabbitmq-source-connect.properties
```

```
$ connect-standalone \
-daemon /etc/schema-registry/connect-json-standalone.properties  \
/etc/kafka-connect-rabbitmq/rabbitmq-source-connect.properties
```

RabbitMQ -> Kafka



Lambda Architecture

BigDataに伴うアプリケーションの課題

- それぞれのアプリケーションで実行していたバッチ処理が終わらない
- レコード数も短い期間で数億と膨大になり、

  データベースのindexよりもI/Oが厳しい
- レプリケーション遅延担保が難しい
- 数千万ユーザーのリアルタイムの分析をするには厳しい
- 分散したデータベースにどう立ち向かうか

BigDataへのアプローチ

- データそのものの集約
- 前日までに集計しておけば良いデータをあらかじめ用意する
- リアルタイムに入力されるデータに対しての

高速なMessage処理と、分散可能なデータストレージ

Lambda Architecture

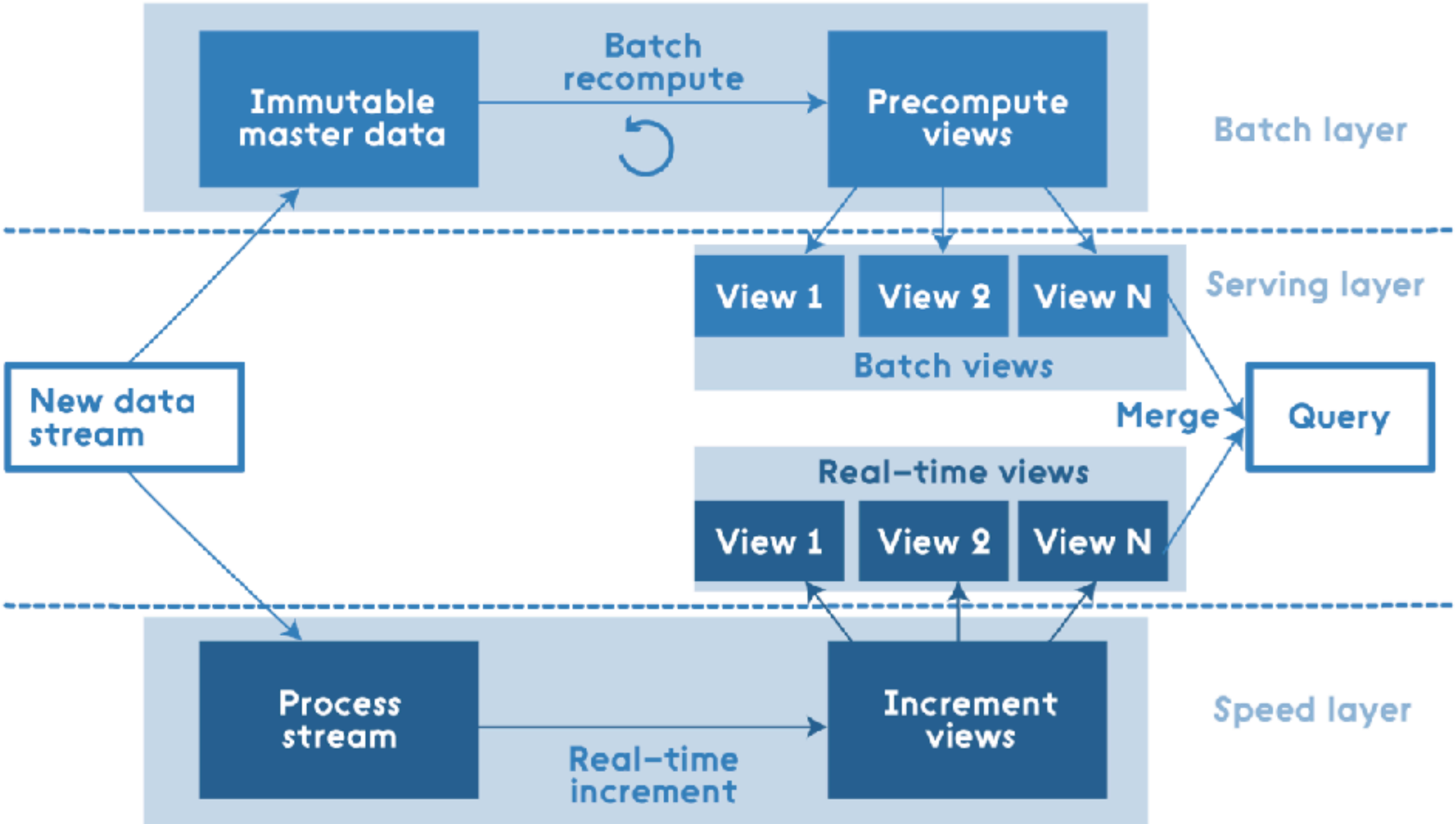
- バッチ層、サービス層、スピード層で構成
- バッチ層は、大きなデータの集計や、大量データの分析などを担当する -> **Hadoop(MapReduce), Spark**
- サービス層はバッチ層の集約結果を提供する

  Hive, HBase, ElephantDB, Splout SQL, pipelineDB…
- スピード層はリアルタイム処理の結果を提供する層

  Spark, Storm, Kafka, Cassandra etc..
- サービス層とスピード層の両方の値をマージして返却

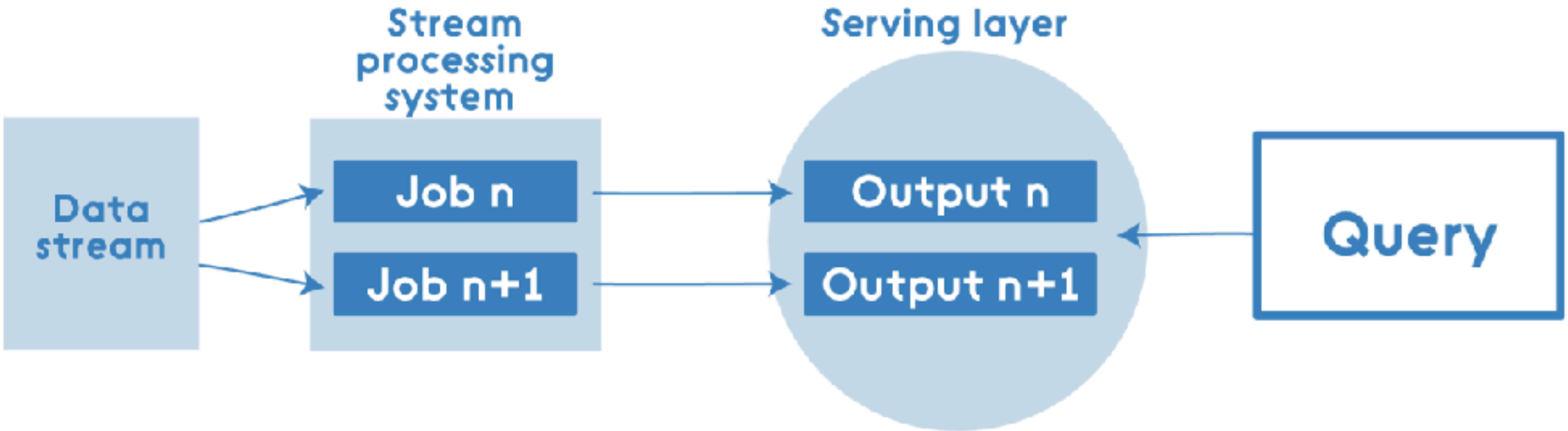
  サービスによっては難易度が高い・・！

Lambda Architecture

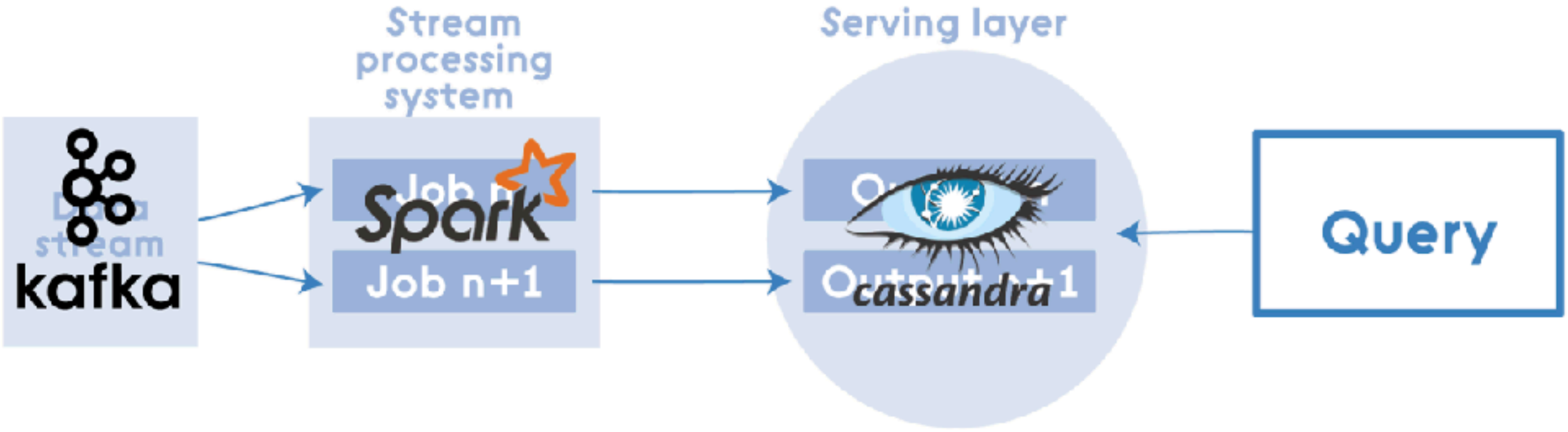


Kappa Architecture

Kappaアーキテクチャ



Kappaアーキテクチャ



```
def main(args: Array[String]) {
 val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> "localhost:9092",
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "kafka_builderscon_stream",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (false: java.lang.Boolean)
 )
 //
 val spark = SparkSession
  .builder
  .master("local[*]")
  .appName("buildersconSmaple")
  .config("spark.cassandra.connection.host", "192.168.10.10")
  .getOrCreate()
 val streamingContext = new StreamingContext(spark.sparkContext, Seconds(5))
```

kafka Streamで実行されるるdef

Kafka 接続情報

最新の流れて来たデータを使って常に処理

Spark App名

Streaming処理結果をCassandraに
5秒おきに 全データを書き込む

処理データの経過を書き込む
死んでもここからやり直す

```
streamingContext.checkpoint("/tmp/")
val topics = Array("message-topic")
val stream = KafkaUtils.createDirectStream[String, String](
  streamingContext,
  PreferConsistent,
  Subscribe[String, String](topics, kafkaParams)
)
val pairs = stream.map(record => (record.value, 1))
val count = pairs.updateStateByKey(updateFunc)
```

Kafka Topicのデータを
SparkのStreamに流し込む

流れてくるRDDの処理開始

```
count.foreachRDD((rdd, time) => {
  val count = rdd.count()
  if (count > 0) {
    rdd.map(record => ("spark", streamMessageParse(record._1.toString).message, record._2))
      .saveToCassandra("builderscon", "counter", SomeColumns("stream", "message", "counter"))
  }
 })
 count.print()
 streamingContext.start()
 streamingContext.awaitTermination()
}
```

集計結果をCassandraのカラムにmapして書き込む

for Application

メッセージのバリデーションはどこで？

- Producerでバリデーション

  -> 最低限の型守る

  -> 他サービスの知識が必要になる場合は実装しない

- Consumerでバリデーション

  受信したメッセージを調べ、

  正しくないものをスキップ

メッセージのバリデーションはどこで？

- Kafka Streamという選択肢

  javaまたはScalaで実装可能

  -> 実装自体はそんなに難しくない

- サービスの知識は専用のStreamで行う

  -> マイクロサービスであれば

PipelineDBという選択肢

- 流れてくるメッセージに対して、

  リアルタイムに、SQL Likeに取得したい

- かつストレージを圧迫させたくない

- PostgreSQL Compatible

- PDOで接続可能！

PipelineDBという選択肢

```
CREATE EXTENSION pipeline_kafka;
SELECT pipeline_kafka.add_broker('localhost:9092');

CREATE STREAM logs_stream (payload json);

CREATE CONTINUOUS VIEW message_count
AS SELECT COUNT(*) FROM logs_stream;
```

KSQL

- Kafkaに保管されているメッセージに

  SQLインターフェースでアクセスするもの

- 現在Developer Preview

- RESTでアクセスすることができます

一歩進んだ分散アプリケーション

- Prestoを利用し、

  RDBMSやRedisと組み合わせたデータ取得が可能

  RedisのランキングとRDBMSの正規化されたデータ、

  Kafkaを使った直近のログデータ掛け合わせ

for Prestodb

- 型定義 etc/kafka/table名.json
- 接続情報 “etc/catalog/kafka_tests.properties”
- prestoのカタログに接続情報を記述
- topic名は hoge.fuga 形式とする

```
{
   "tableName": “action",
   "schemaName": "analyze",
   "topicName": "analyze.action",
   "message": {
      "dataFormat": "json",
      "fields": [
         {
            "name": "uuid",
            "mapping": "uuid",
            "type": "VARCHAR"
         },
         {
            "name": "uri",
            "mapping": "uri",
            "type": "VARCHAR"
         },
         {
            "name": "name",
            "mapping": "name",
            "type": "VARCHAR"
         }
      ]
   }
```

kafkaのtopic情報を記載

カラムを定義 プリミティブな型に変換

connctor名はkafka

```
connector.name=kafka
```

kafkaのクラスターを指定

```
kafka.nodes=127.0.0.1:9092
```

kafkaのtopicを指定

```
kafka.table-names=analyze.action
kafka.hide-internal-columns=false
```

kafkaのシステムカラム有無

```
SELECT redttt._key, redttt._value, test_id,
test_name, created_at, uri, uuid
FROM my_tests.testing.tests AS myttt
INNER JOIN red_tests.test.string AS redttt ON
redttt._key = myttt.test_name
INNER JOIN kafka_tests.analyze.action AS kafkataa
ON kafkataa.name = myttt.test_name
WHERE myttt.test_name = '{$name}' LIMIT 1";
```

Apache Kafkaの始め方

- Apache Kafka公式ページからダウンロード
- Confluentからダウンロード
- Kafka Connectを利用するのであれば、
  Confluent利用がオススメ
- Hadoop等がなくても利用可能です

Apache Kafka GUI

- **Cluster管理**

  https://github.com/yahoo/kafka-manager
- **Message管理**

https://github.com/landoop/kafka-topics-ui

https://github.com/ldaniels528/trifecta

https://github.com/Landoop/kafka-topics-ui

https://github.com/Landoop/kafka-connect-ui

など

まとめ

- 複雑化したアプリケーション・分散アプリケーションの問題解決の一つ

- Kafka以外のMessage Queueも選択肢に

- Stream処理などが必要であればKafkaを推奨

- 規模にあわせたミドルウェア選定・障害対応・運用を