Trabalhando com Big Data em Tempo Real

Maycon Viana Bordin
Julio C. S. dos Anjos
Raffael Bottoli Schemmer
Cláudio Geyer

Instituto de Informática
Universidade Federal do Rio Grande do Sul

.inf INSTITUTO DE INFORMÁTICA UFRGS

UFRGS UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

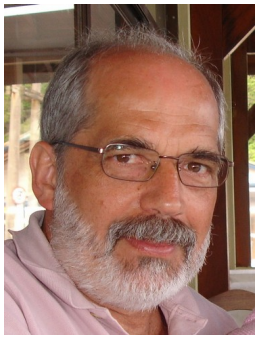
Quem somos ?

GPPD

.inf INSTITUTO DE INFORMÁTICA UFRGS

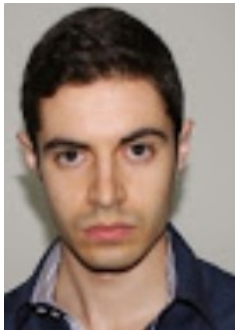
UFRGS UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Palestrantes



Cláudio Geyer

Julio Anjos



Maycon Bordin
(Vídeos de apps)

Raffael Schemmer

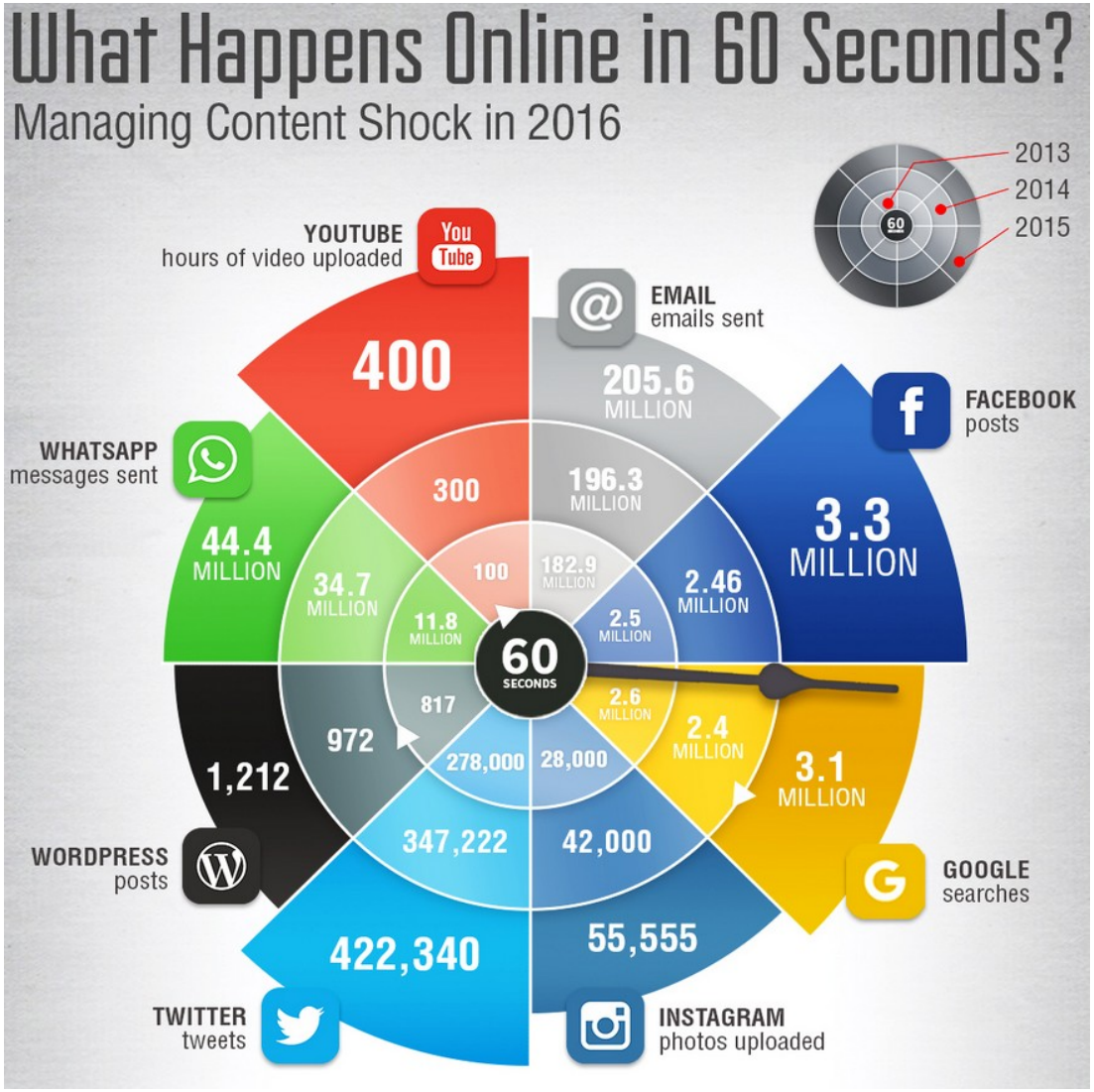
Alexandre Veith
(Demos de apps)

Agenda

- Introdução e Visão Geral
- Algumas aplicações
- Conceitos
- Operadores
- Modelo de Programação
- Plataformas
  - Storm, Spark e Flink
- Demos de Apps em Vídeos


GRANDE
quantidade de dados
estão sendo geradas
em real-time

O que representa 60 segundos para Big Data ?



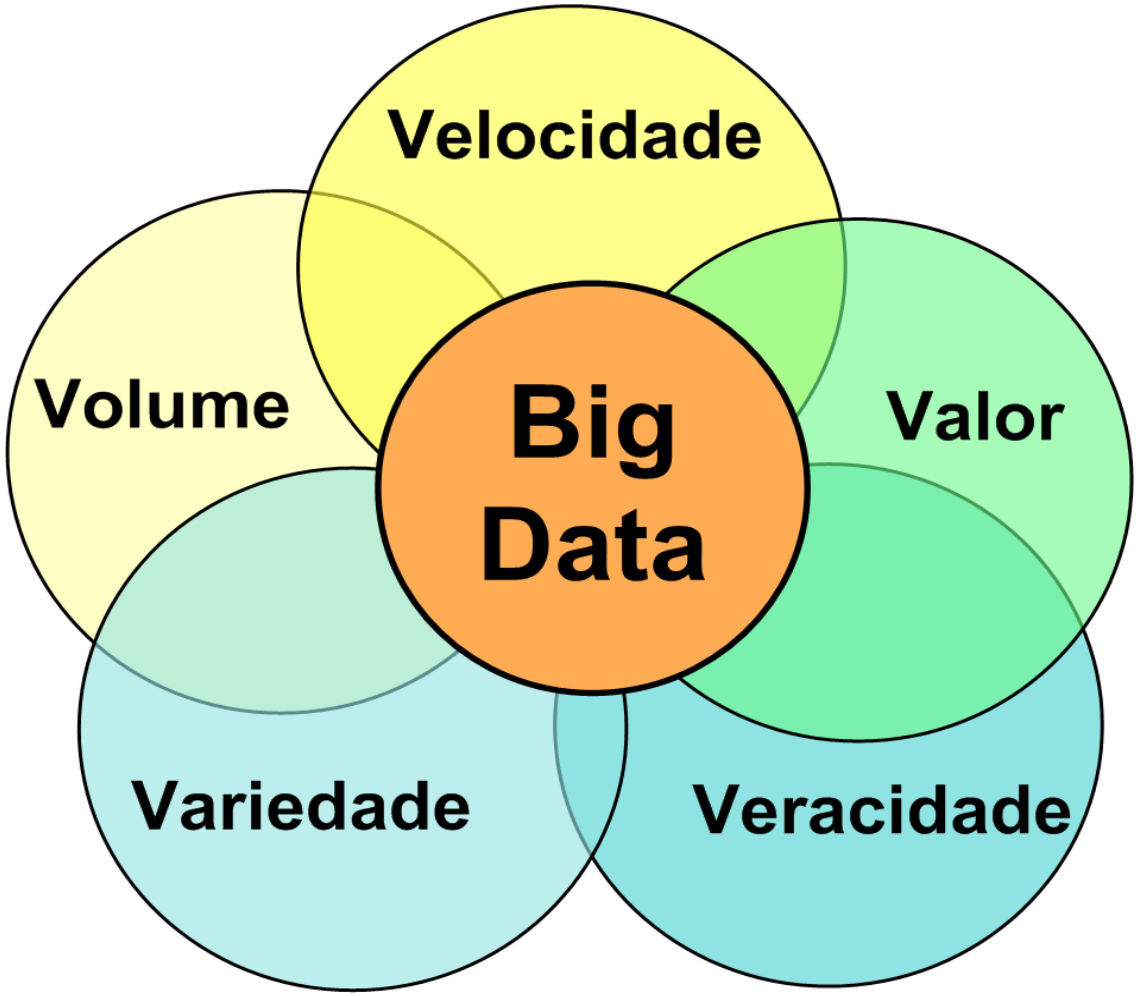
http://www.smartinsights.com/internet-marketing-statistics/happens-online-60-seconds/

O que significa Big Data ?

- Não é um termo claro e preciso;
- Falta consenso entre os cientistas;
- Ainda tem um significado amplo;

[Wu,Buyya, Ramamohanarao; 2015]

5 Vs de Big Data – uma Definição



Algumas aplicações

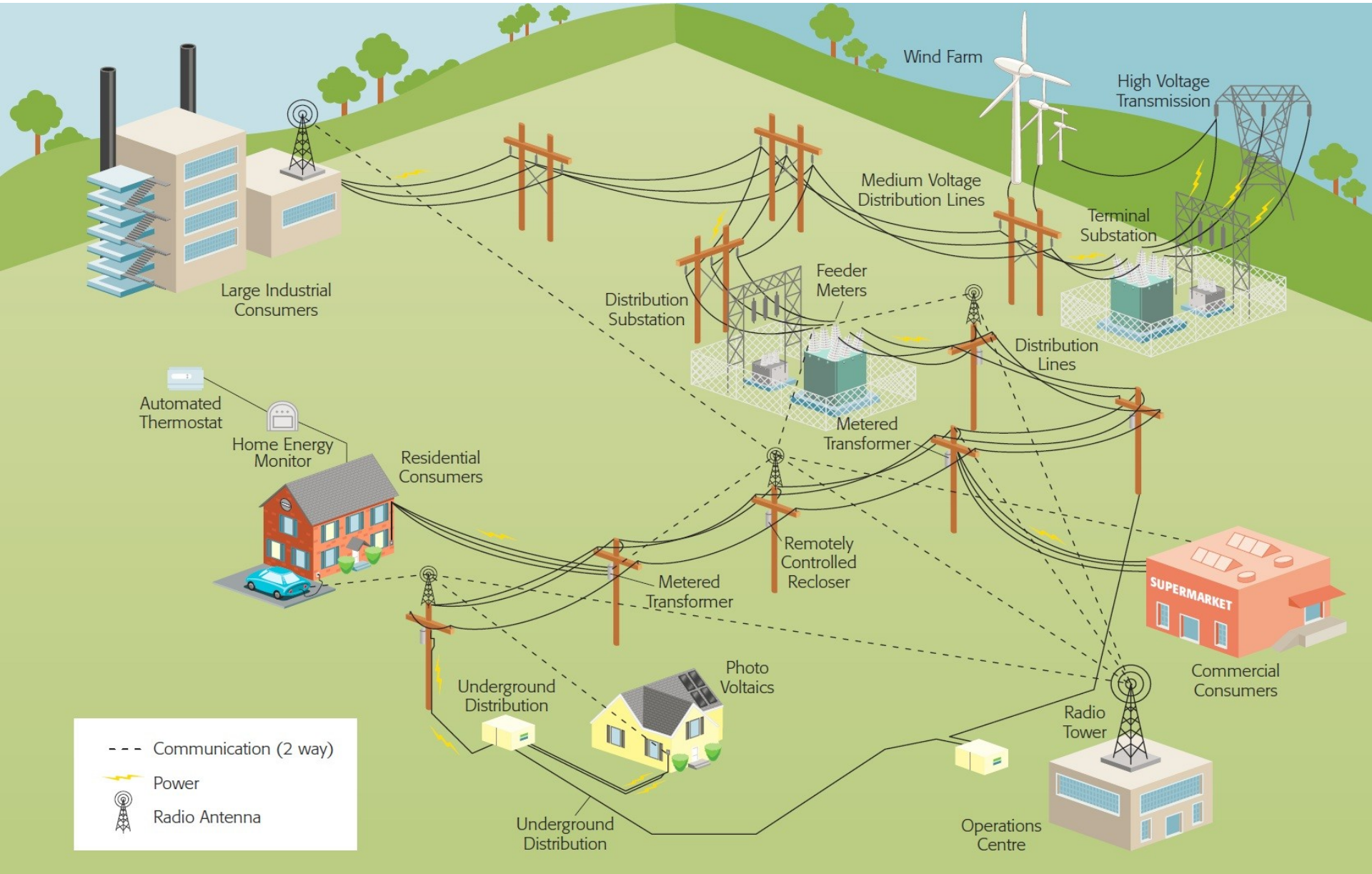
Monitoramento de Tráfego

Monitoramento de Tráfego de Automóveis

- Arquitetura para *Stream* e *complex event processing* (CEP *processing*).
- Entrada baseada em sensores.
- Usa informação colaborativa (crowdsourcing) para resolver problemas de insegurança da fonte de dados.
- Dataset de 13GB por mês da cidade de Dublin (Irlanda).

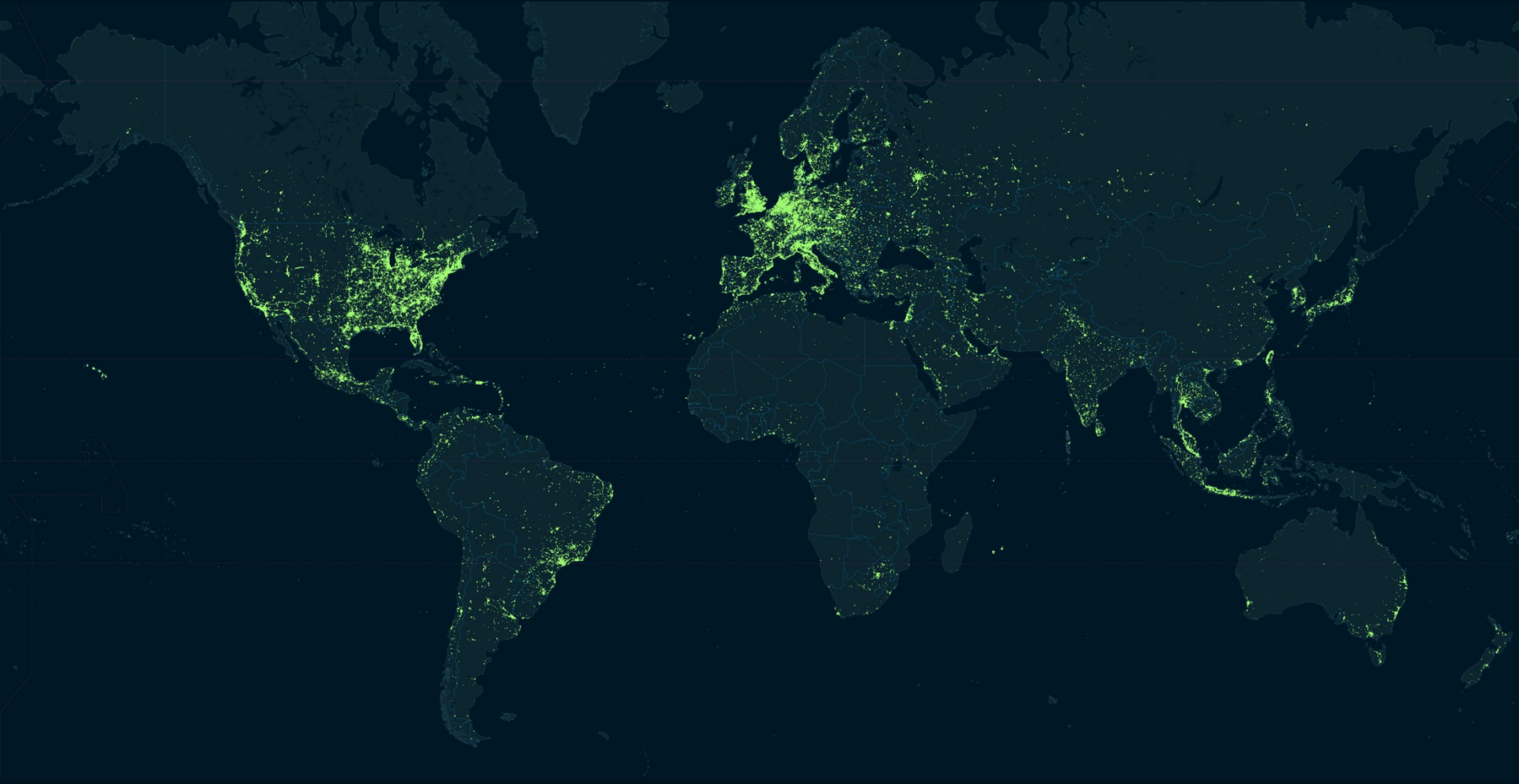
[Artikis, 2014]

Aplicações de Smart Grids



Projeto de Smart Grid em Los Angeles

- 1.4 milhões de consumidores
- Otimização de demanda energética
- Predição de pico de demanda
- Fonte de dados:
  - AMIs (Advanced Metering Infrastructure)
- 3 TB de dados por dia

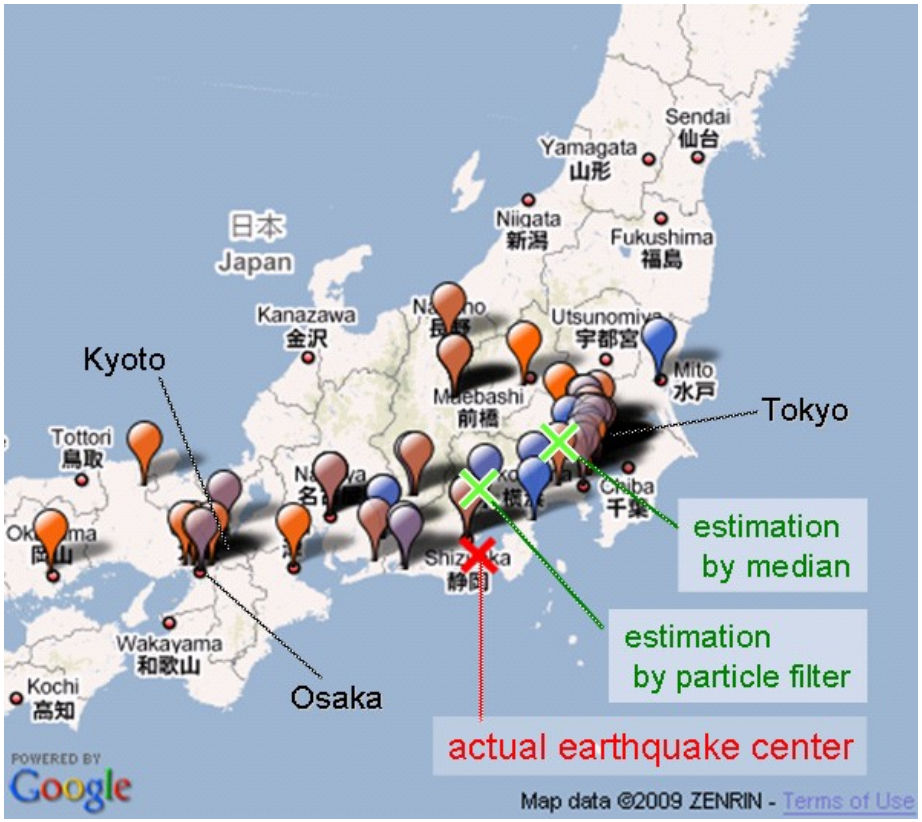


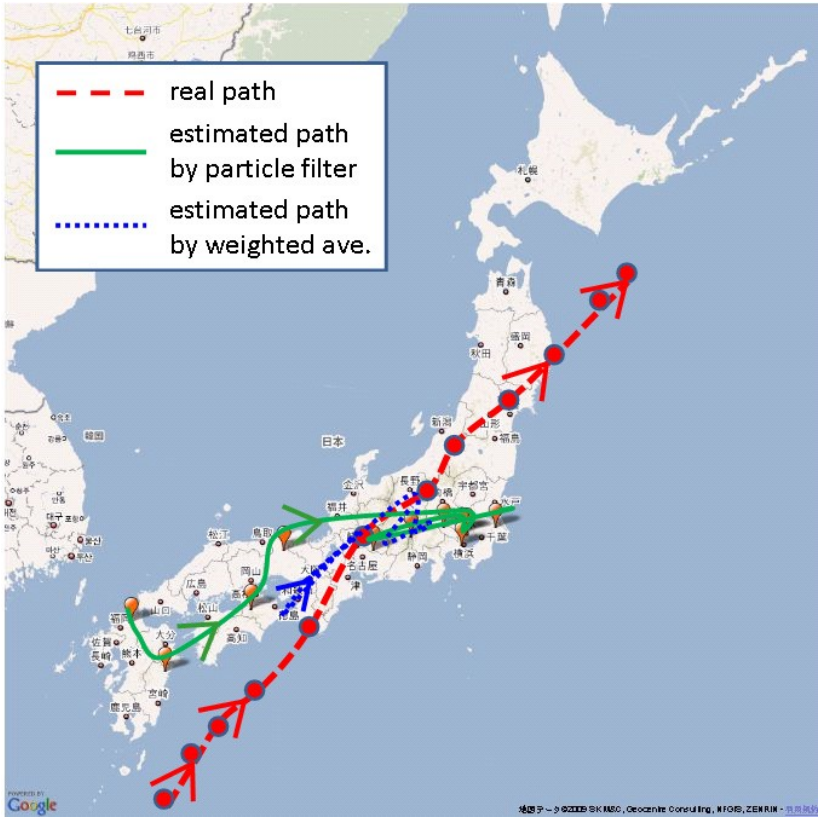
Redes de Sensores

INSTITUTO DE INFORMÁTICA UFRGS

UFRGS UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Detecção de eventos em tempo real usando Twitter





Agência de meteorologia do Japão

Detecção de eventos em tempo real usando Twitter

- Detecção de eventos como: tufões, terremotos,...
- Usuários de Twitter atuam como sensores;
- Estima localização usando Filtro de Kalman e de partículas;
- Detecta 96% de furacões reportados na agência de meteorologia do Japão [Sak10];

Aplicações necessitam...

Grande Volumes de Dados

Em real-time

Continuamente

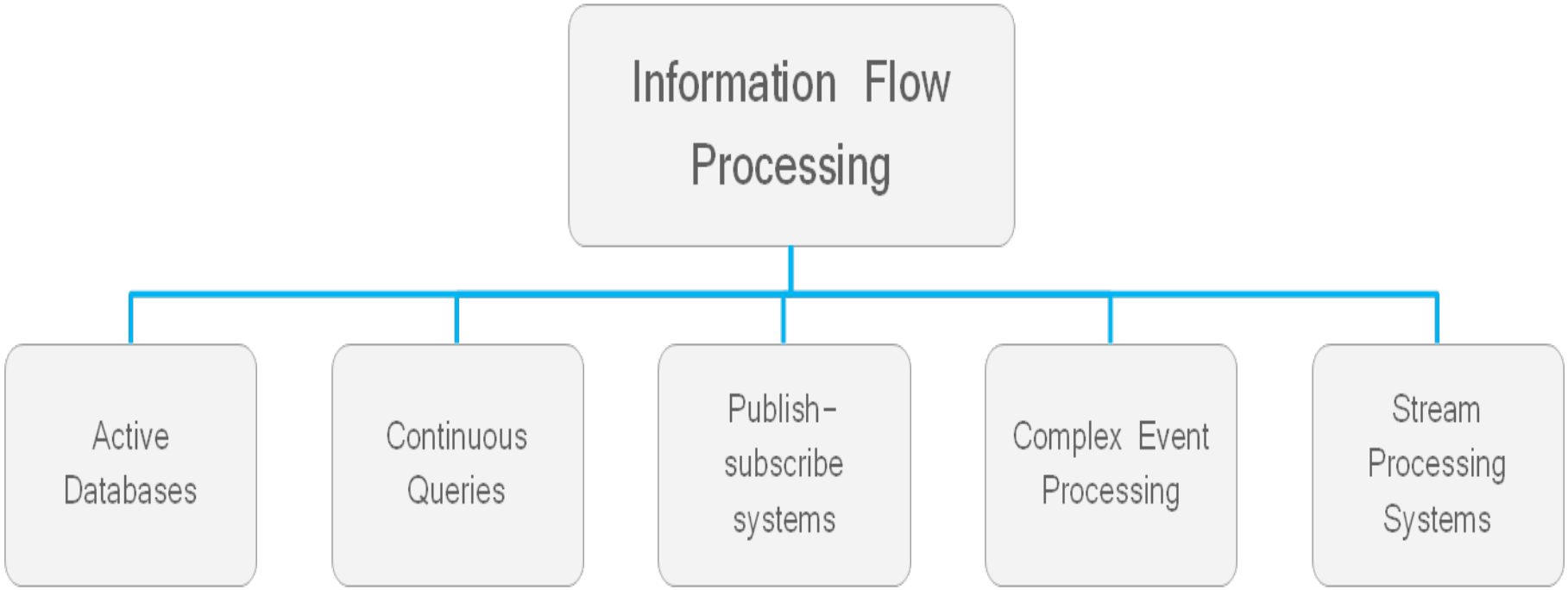
Produzindo ações e informações

Estas aplicações são caracterizadas como tecnologias de Processamento de Fluxo de Informações

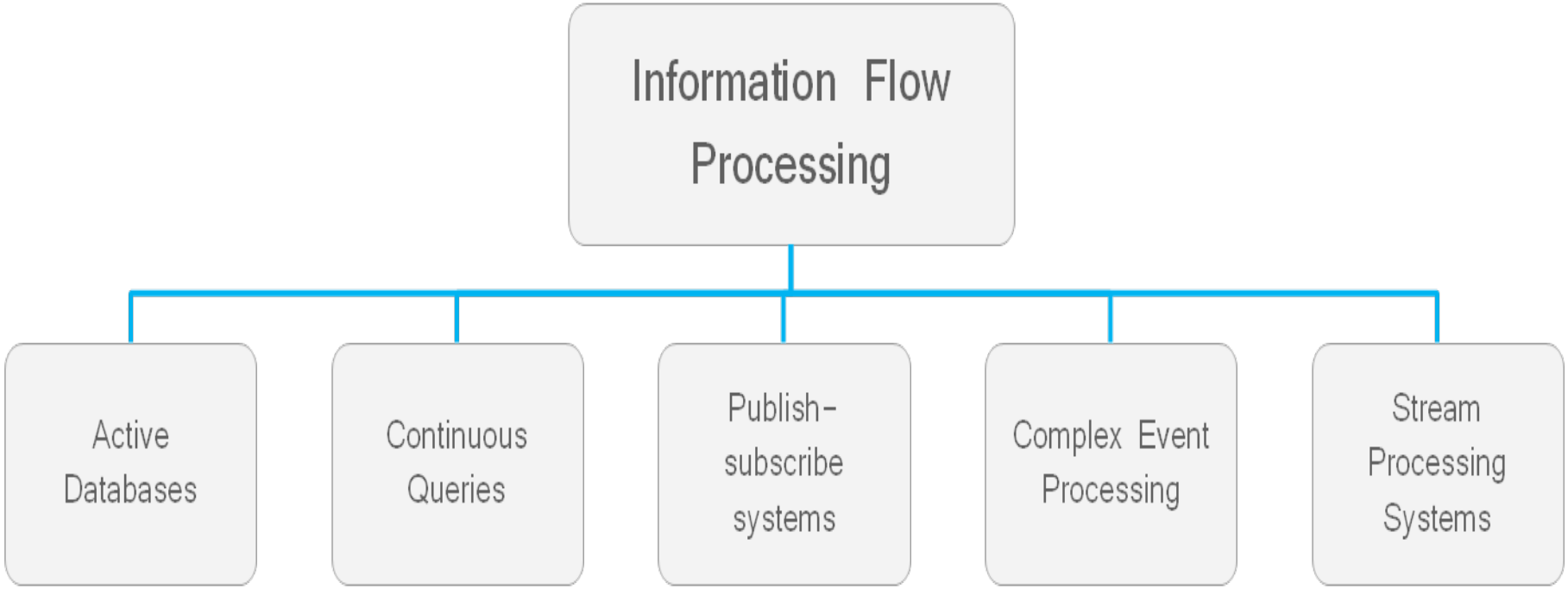
ou

Information Flow Processing

Evolução dos sistemas IFP

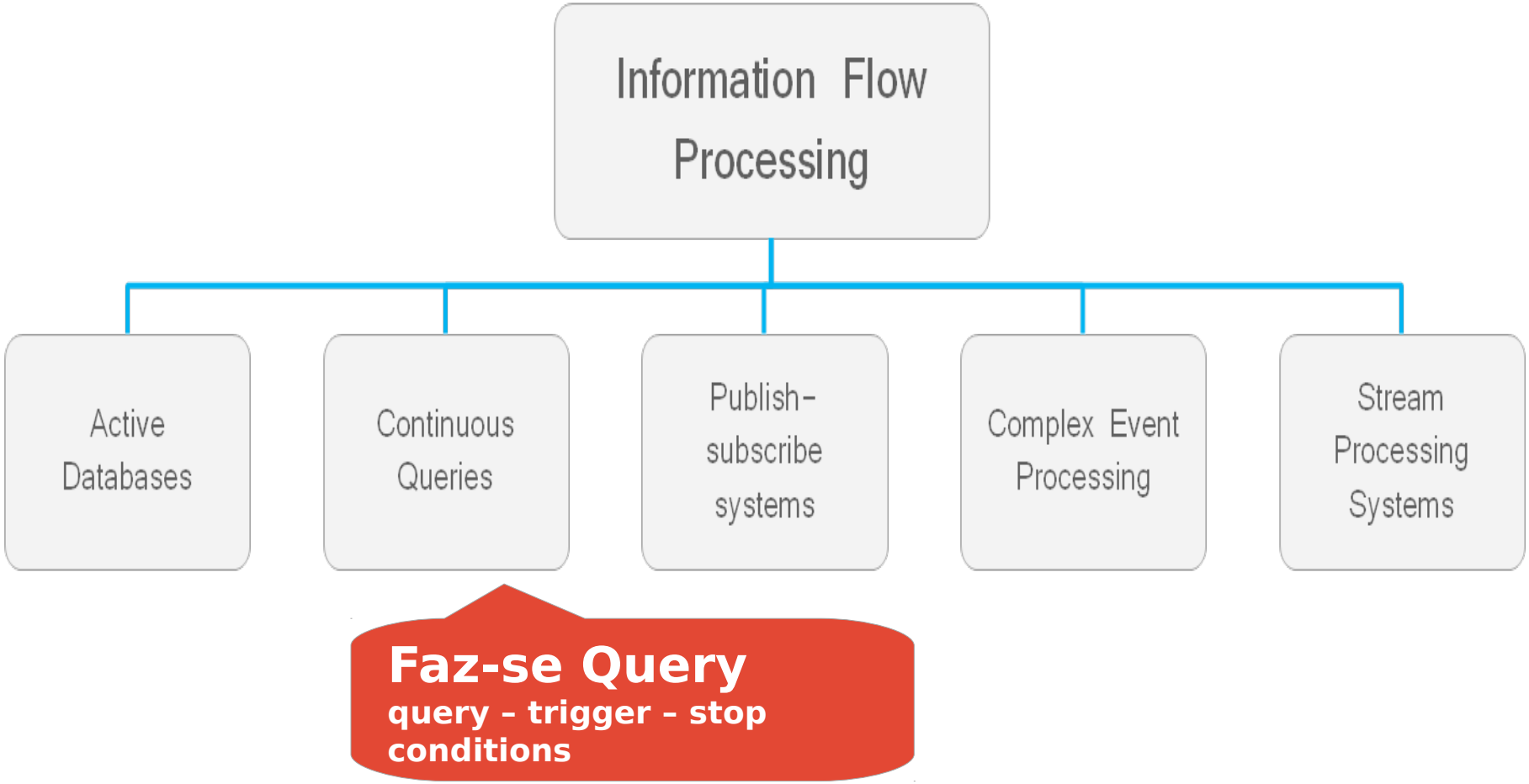


Evolução dos sistemas IFP

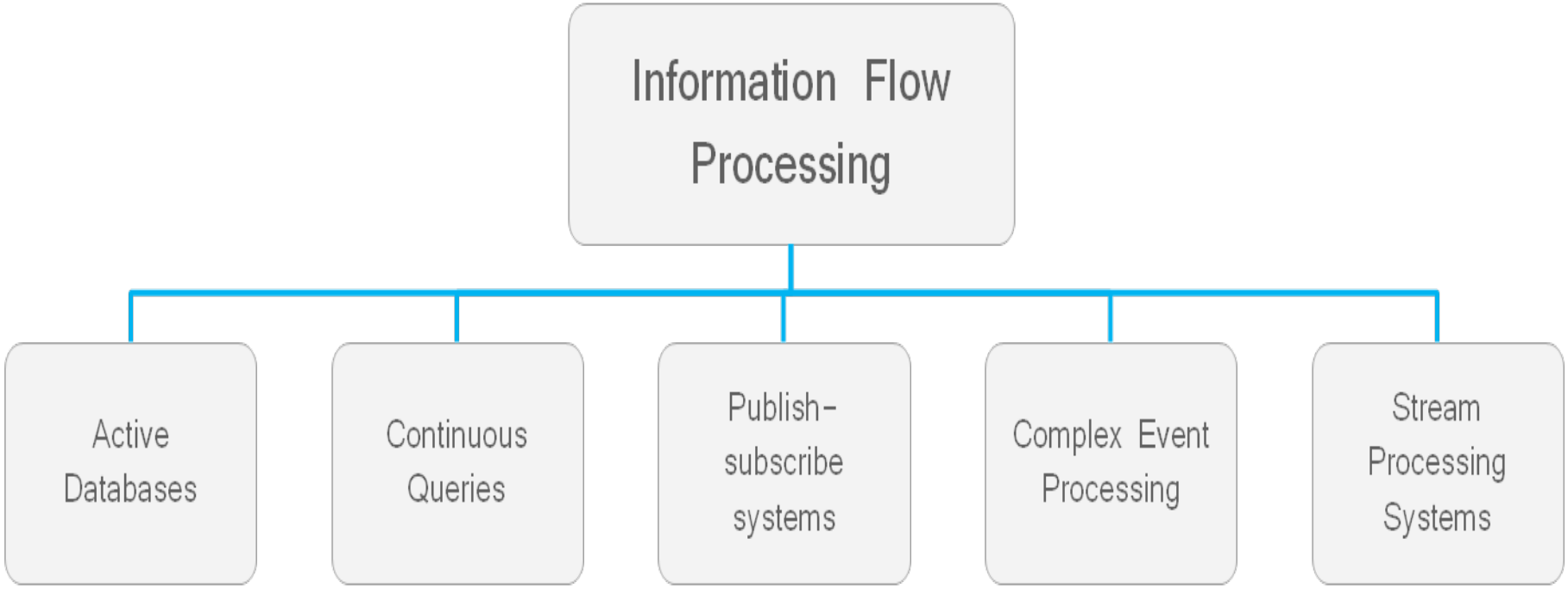


Regras de Filtragem

Evolução dos sistemas IFP

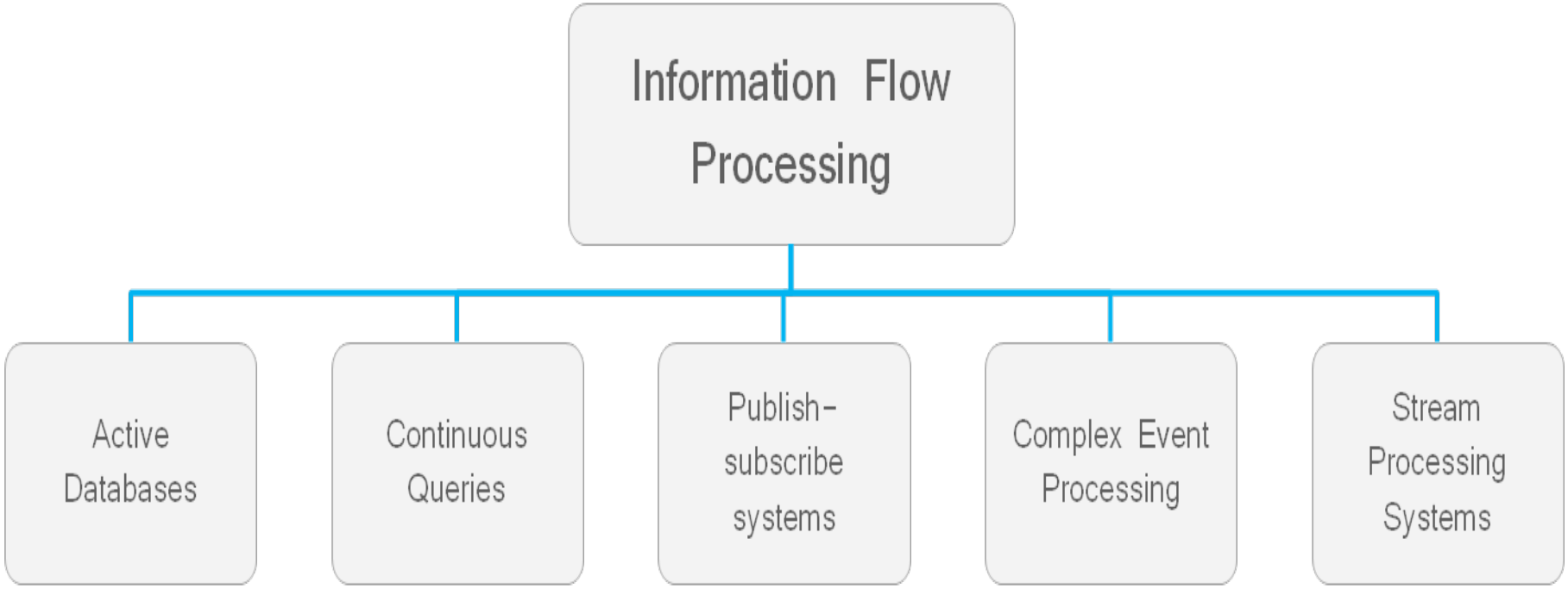


Evolução dos sistemas IFP



Desacoplamento de componentes tópicos e conteúdos

Evolução dos sistemas IFP

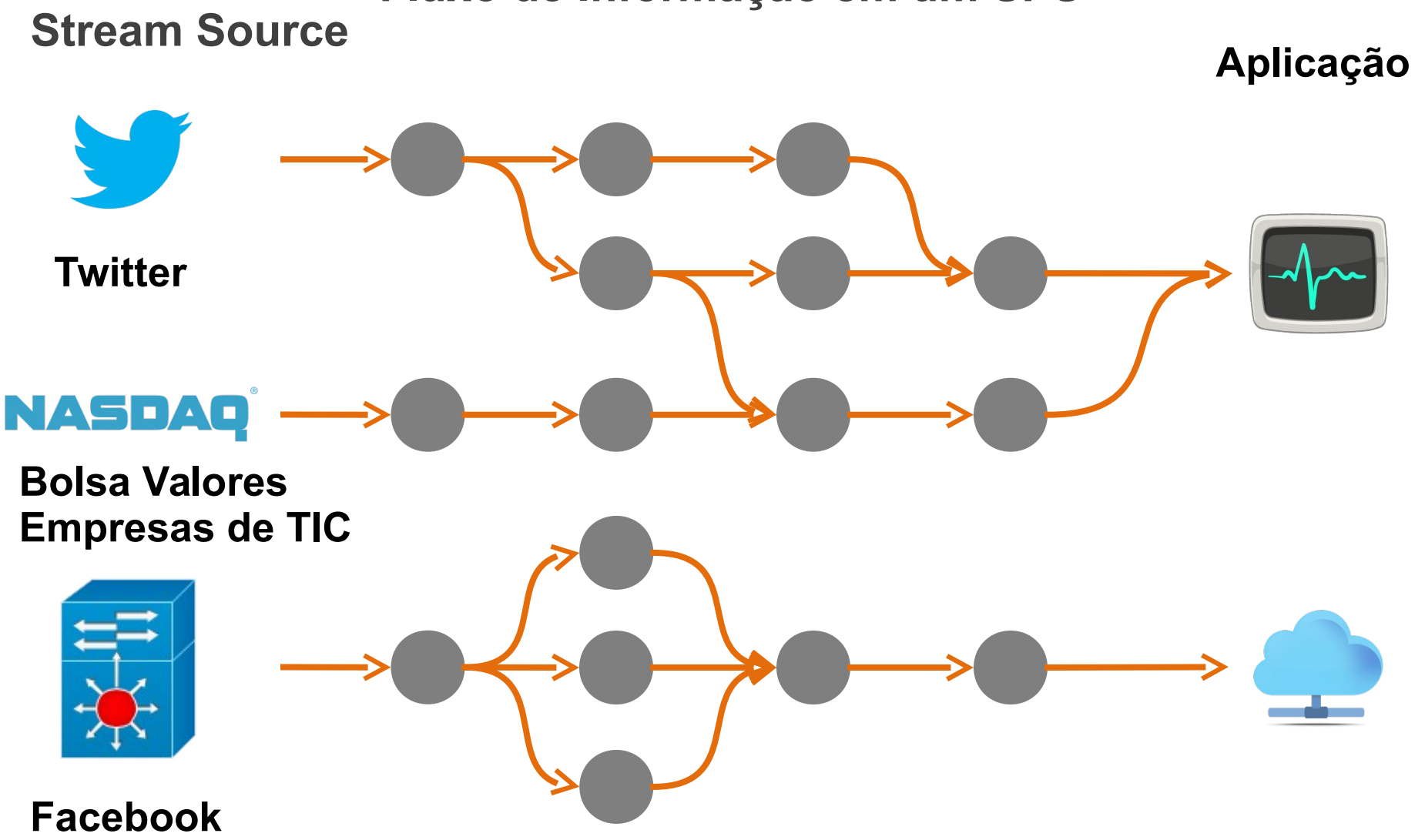


Detecta eventos baseados em regras e padrões

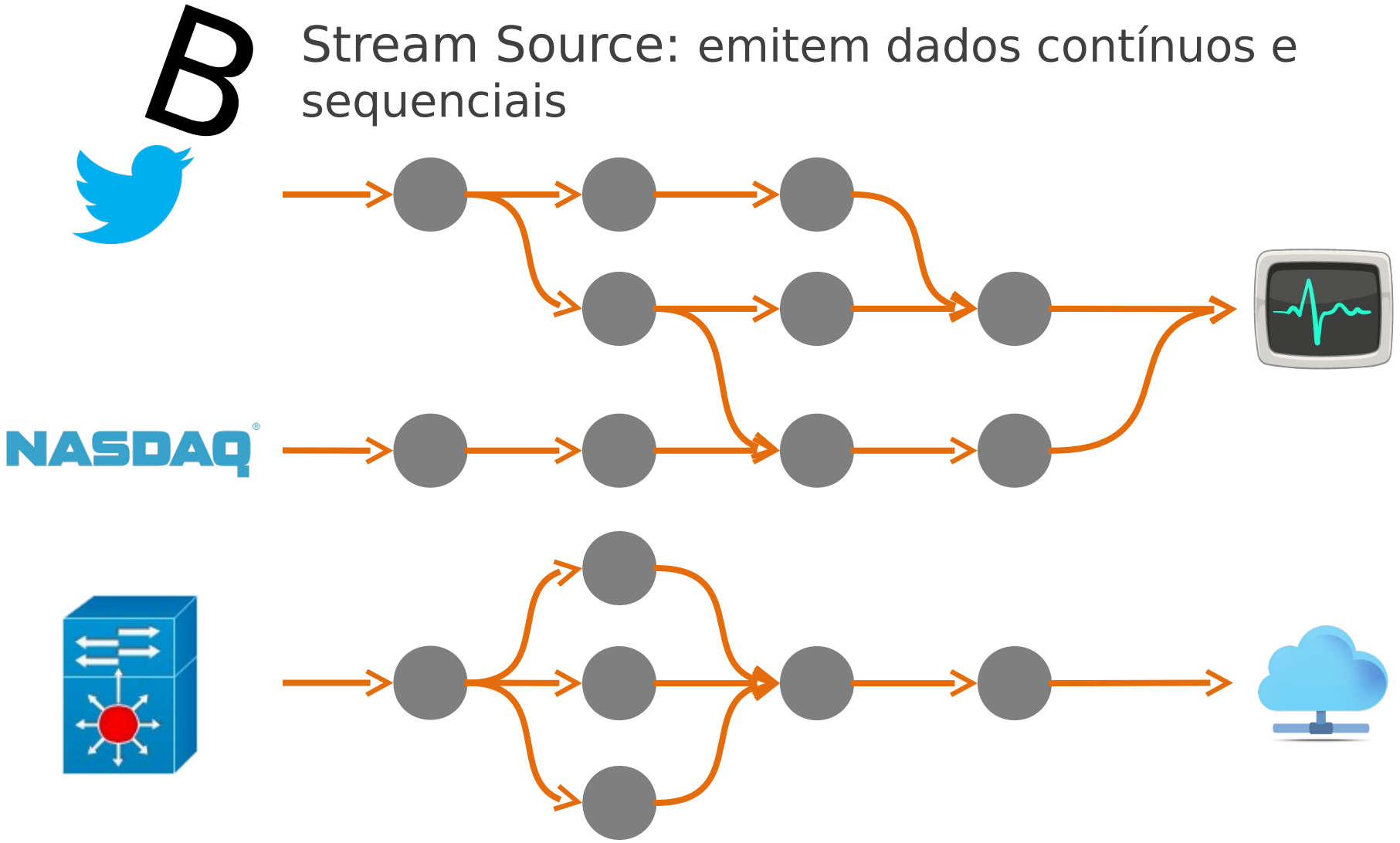
Stream Processing Systems (SPS)

Conceitos

Fluxo de Informação em um SPS



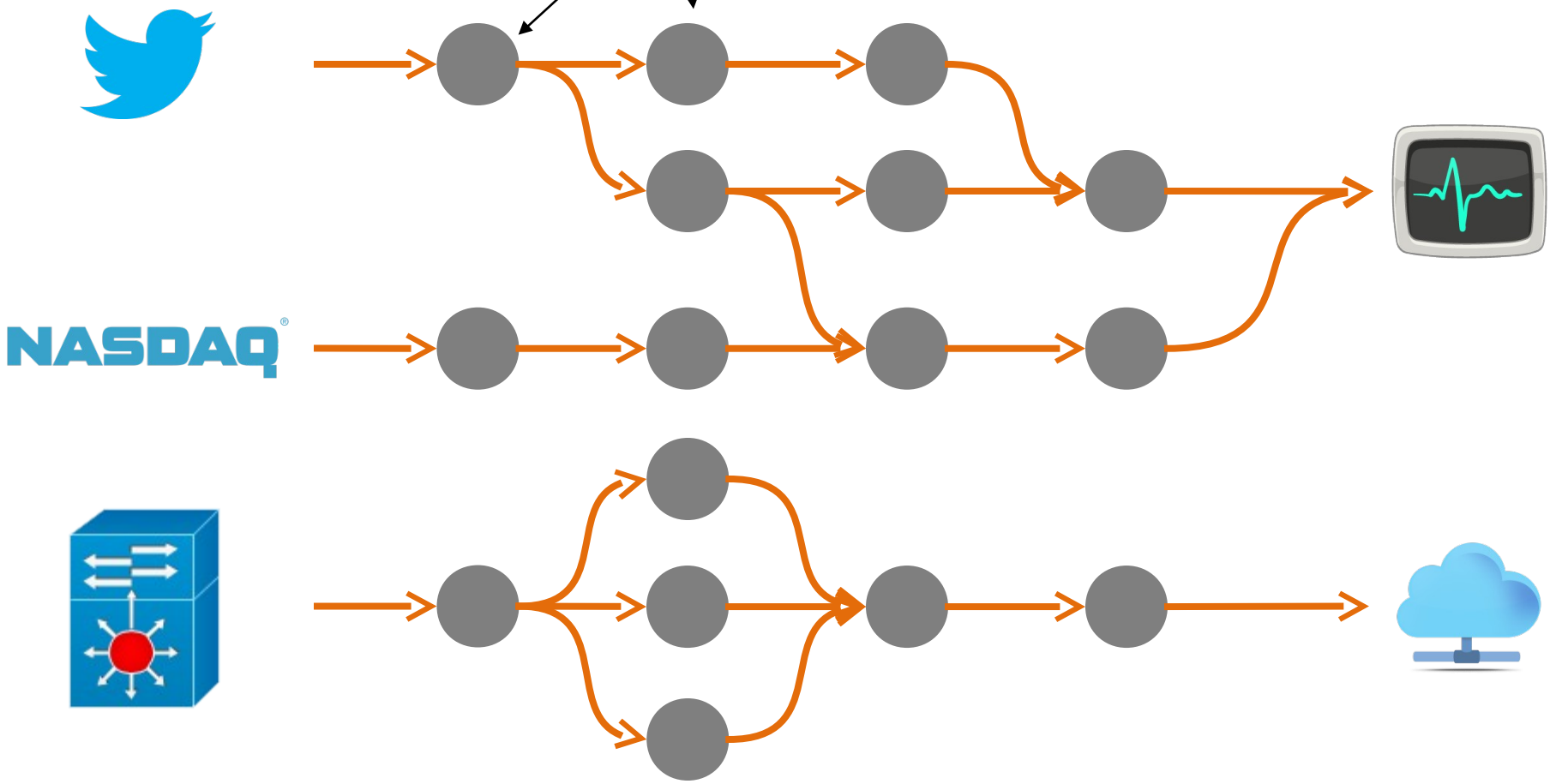
Stream Source: emitem dados contínuos e sequenciais



Taxas de chegada pode ser fixa ou imprevisível

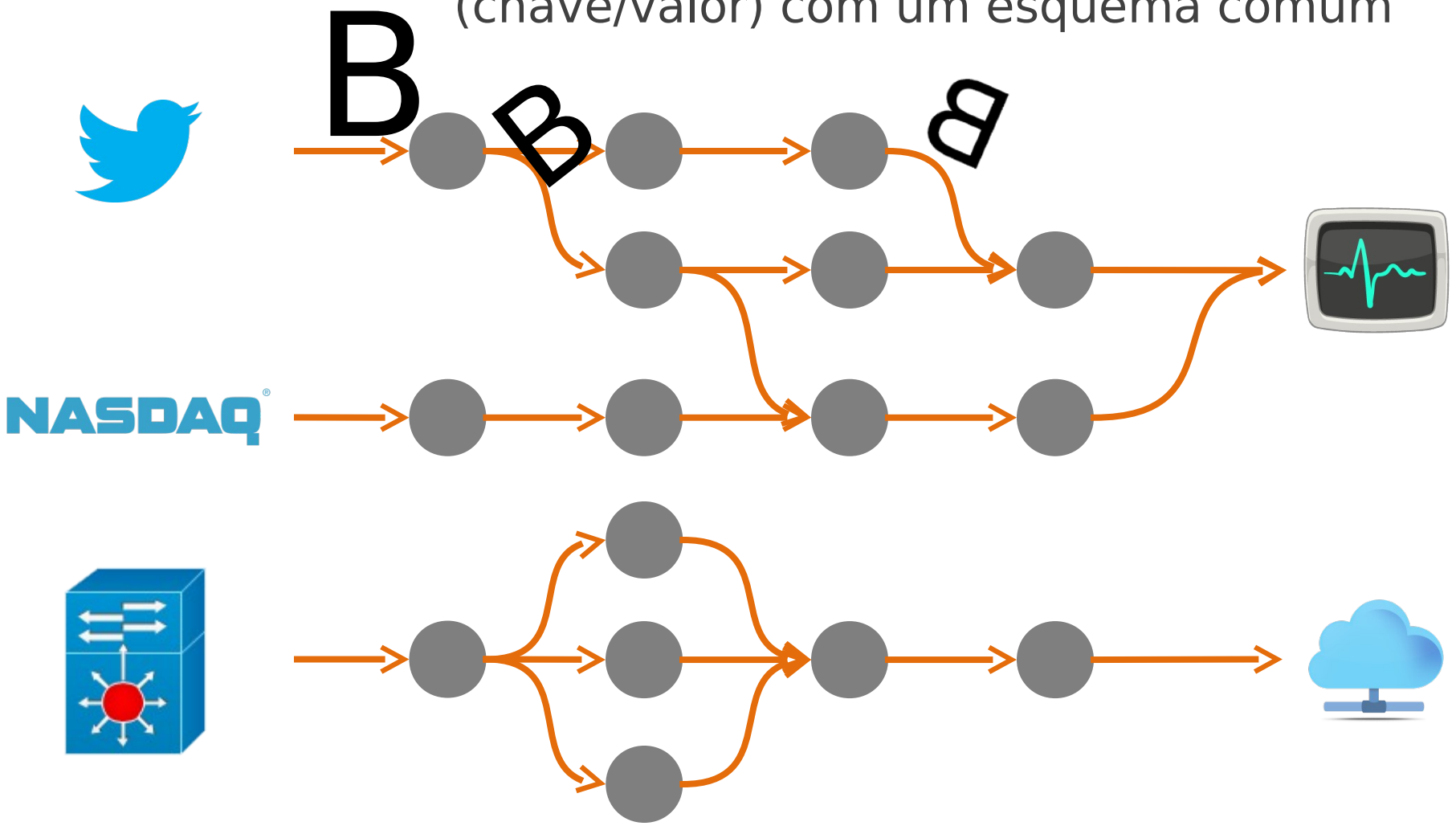
B

Operadores: unidades funcionais básicas de uma aplicação.



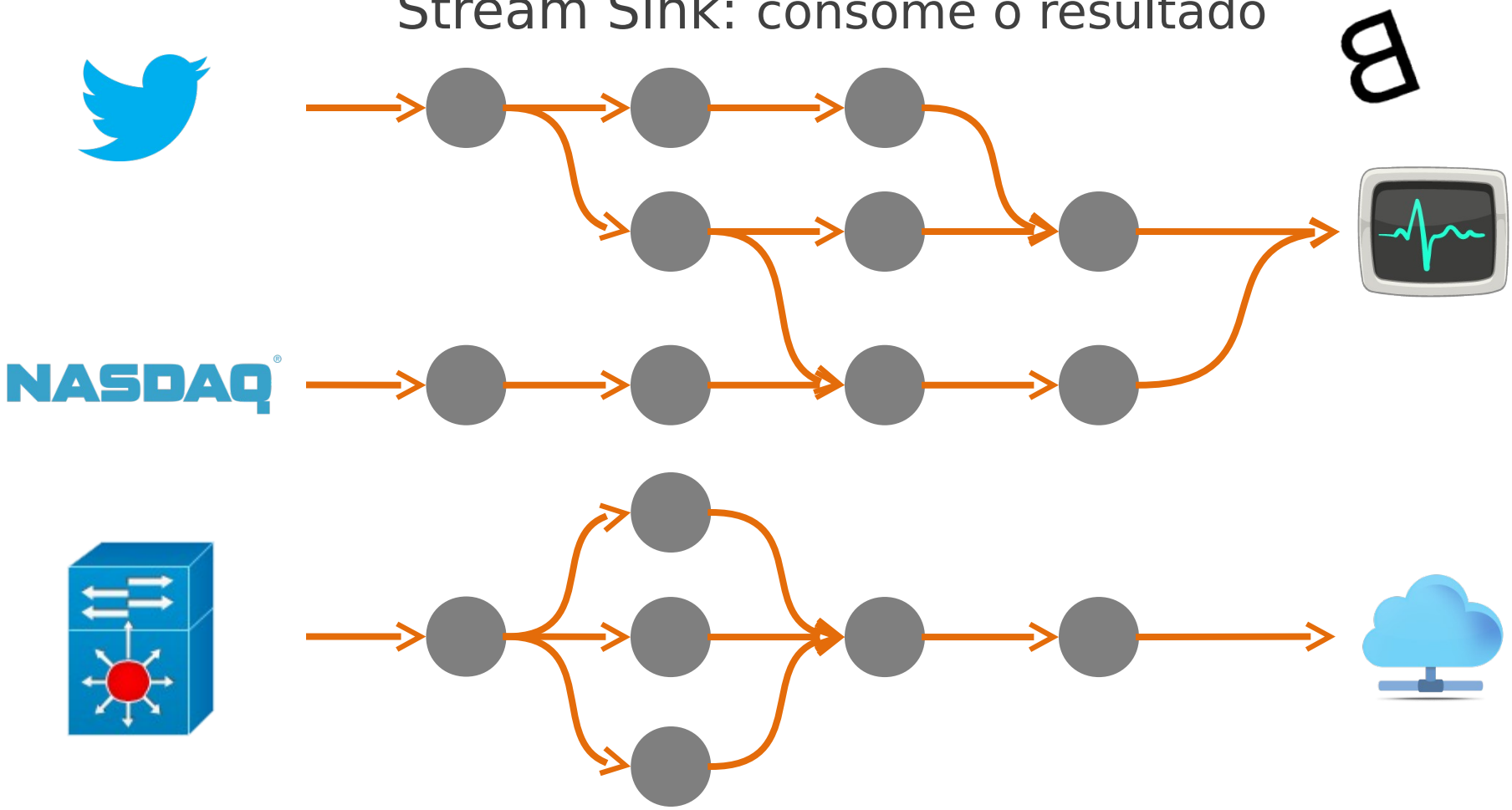
Operador recebe stream, aplica função e gera uma tupla ou conjunto de duplas

Data streams: sequência de tuplas (chave/valor) com um esquema comum

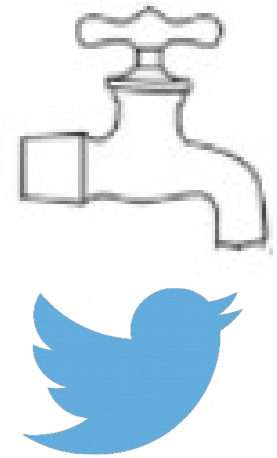


Operador gera mais data streams

Stream Sink: consome o resultado



Data Stream



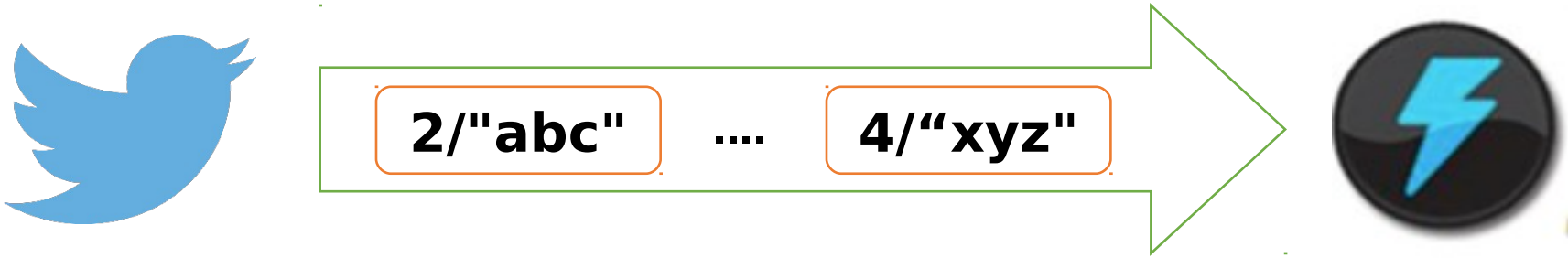
Definições

Tuplas são compostas por estruturas chave/valor:

Chave possui o *timestamp* da tupla.

Em ambientes de *Stream*:

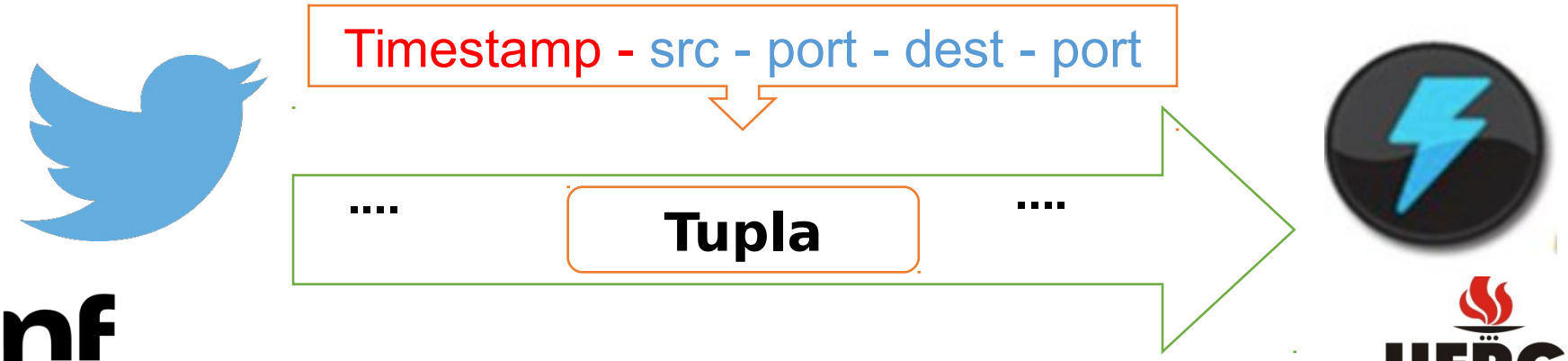
As fontes geradoras são implementadas por estruturas como por exemplo *Spouts no Storm*.



Definições

- **Tuplas podem ser:**
  - **Estruturadas ou não estruturadas:**
    - Necessário realização de pré-processamento dos dados.

- **O formato chave/valor das tuplas:**
  - É chamado de *data schema*.
  - Permite diferentes tamanhos e tipos de entradas.

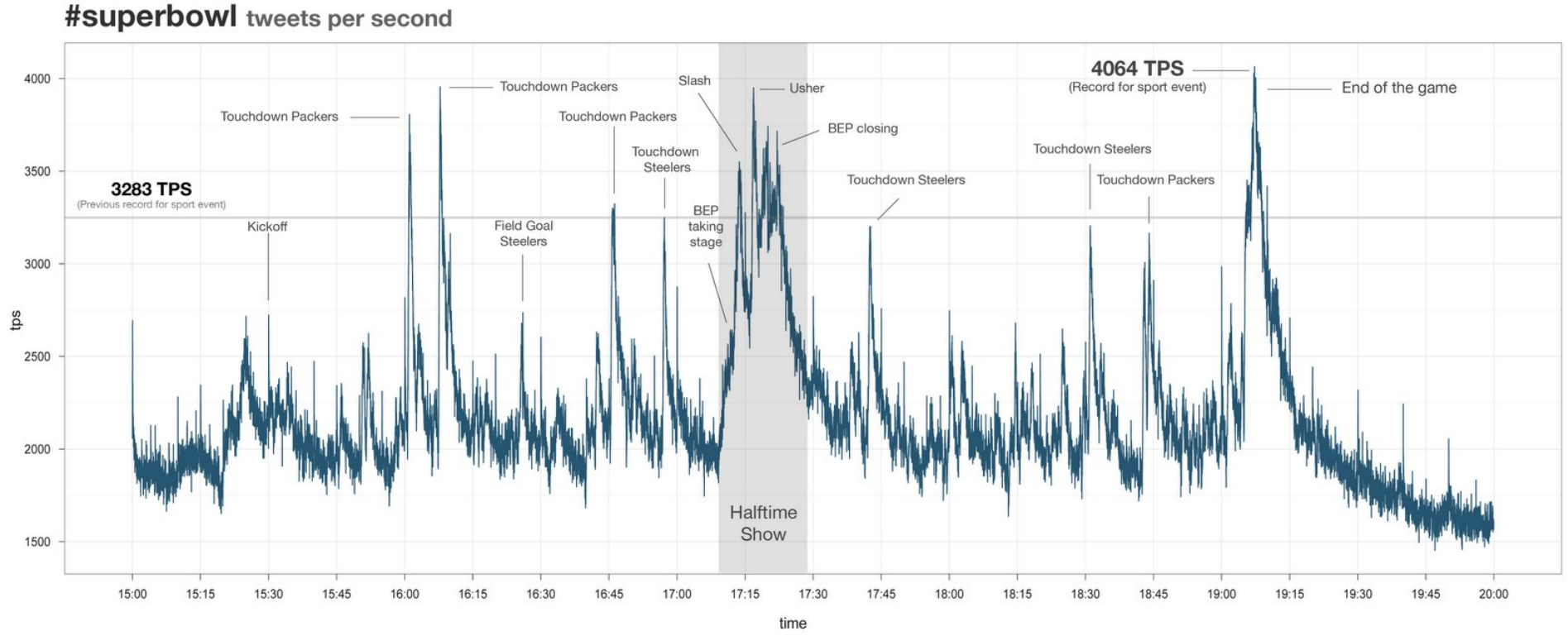


Dado no **stream source**

- A quantidade de dados recebidos é de tamanho ilimitado.

- O volume de dados é contínuo/variável:
  - Tipicamente imprevisível.

A entrada é variável e tipicamente imprevisível

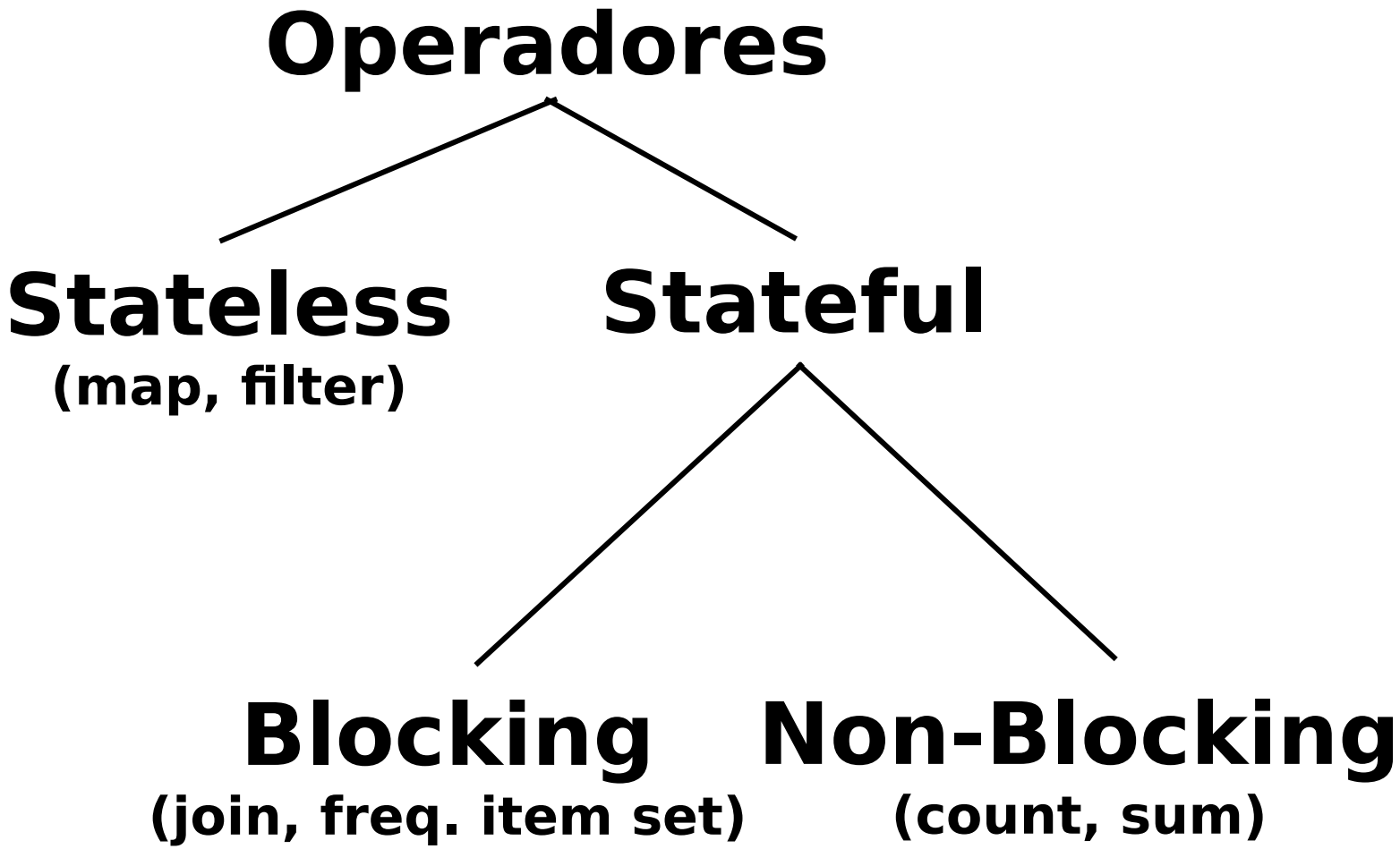


Operadores

Operadores

- Recebem tuplas.
- Processam/modificam as tuplas.
- Geram novas tuplas.

Definição dos Operadores



Definição dos Operadores

- Stateless (Sem Estado):
  Precisam apenas da tupla da entrada para geração da tupla de saída.
  Exemplos: map/filter.

- Statefull (Com Estado):

- Blocking (Bloqueante):
  Requer todas as tuplas necessárias para então realizar o processamento.
  Exemplos: join/freq.

- Non Blocking (Não Bloqueante):
  Requer o resultado do processamento das tuplas anteriores e da tupla atual.
  Exemplos: count/sum.

Tipos de Operadores

Filtragem (filter)

Sem estado (Stateless).

Agregação (aggregate)

Não Bloqueante (Statefull).

Junção (join)

Bloqueante (Statefull).

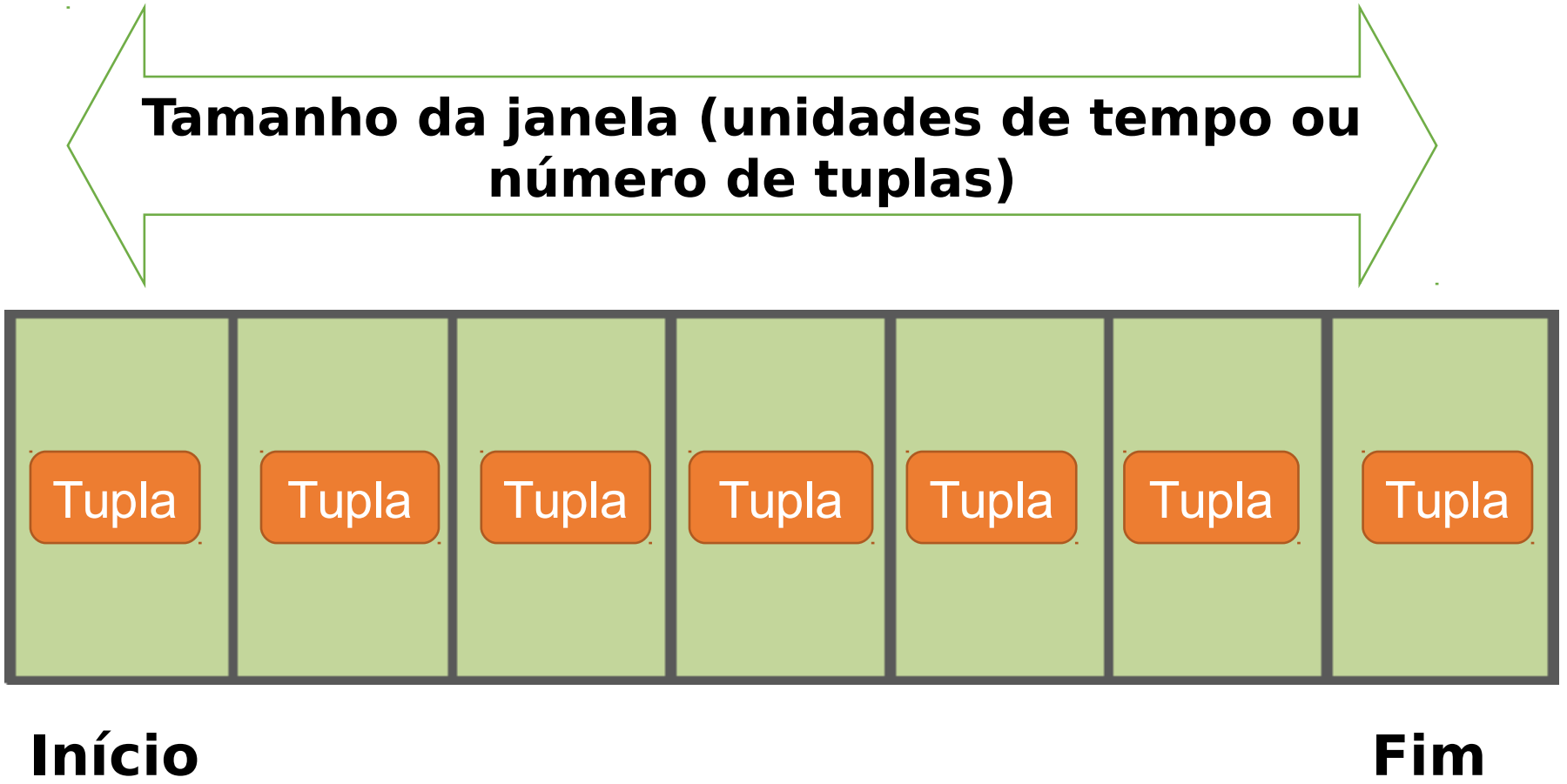
Divisão (split)

Sem estado (Stateless).

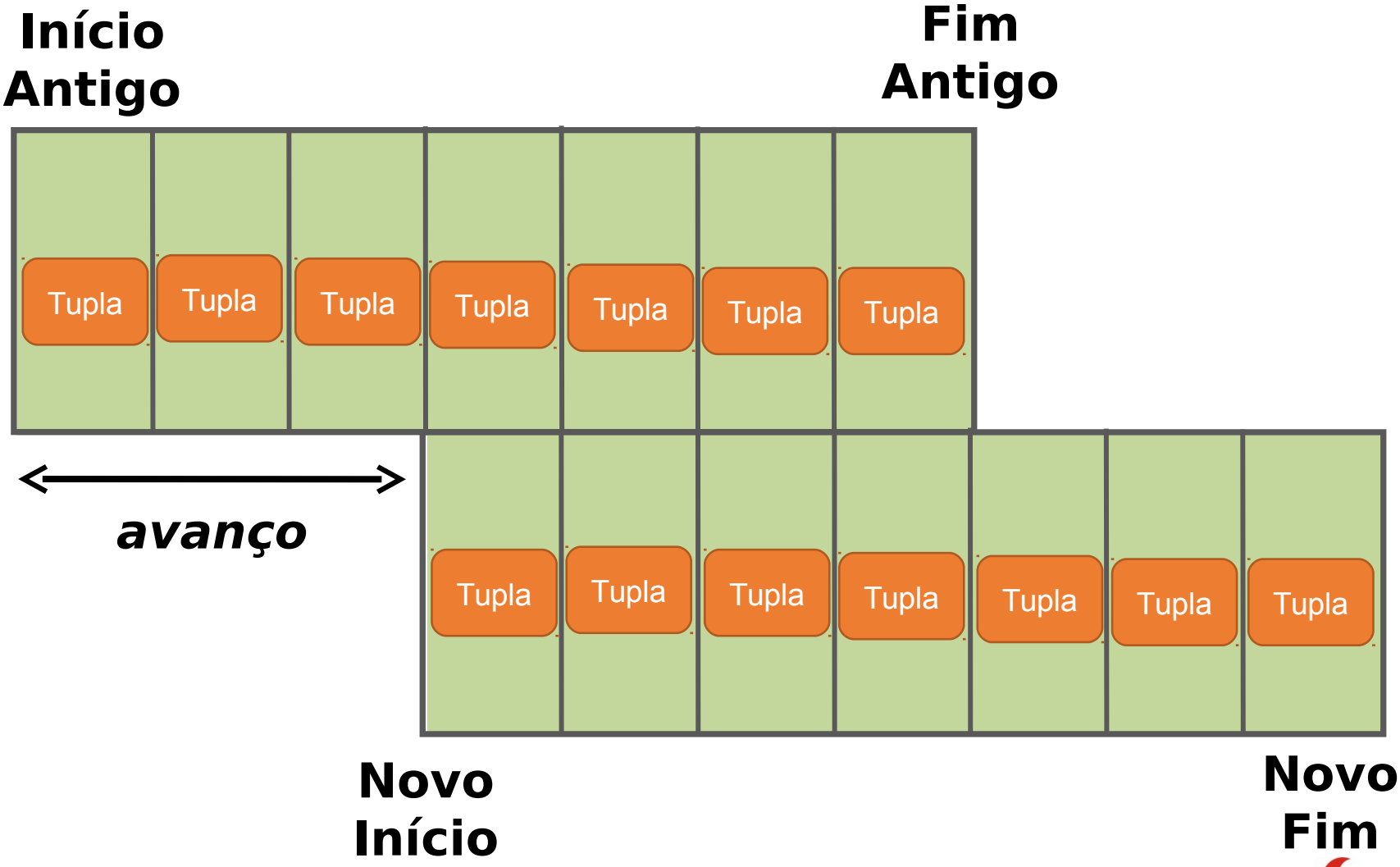
Machine Learning (complex)

Bloqueante/Não Bloqueante.

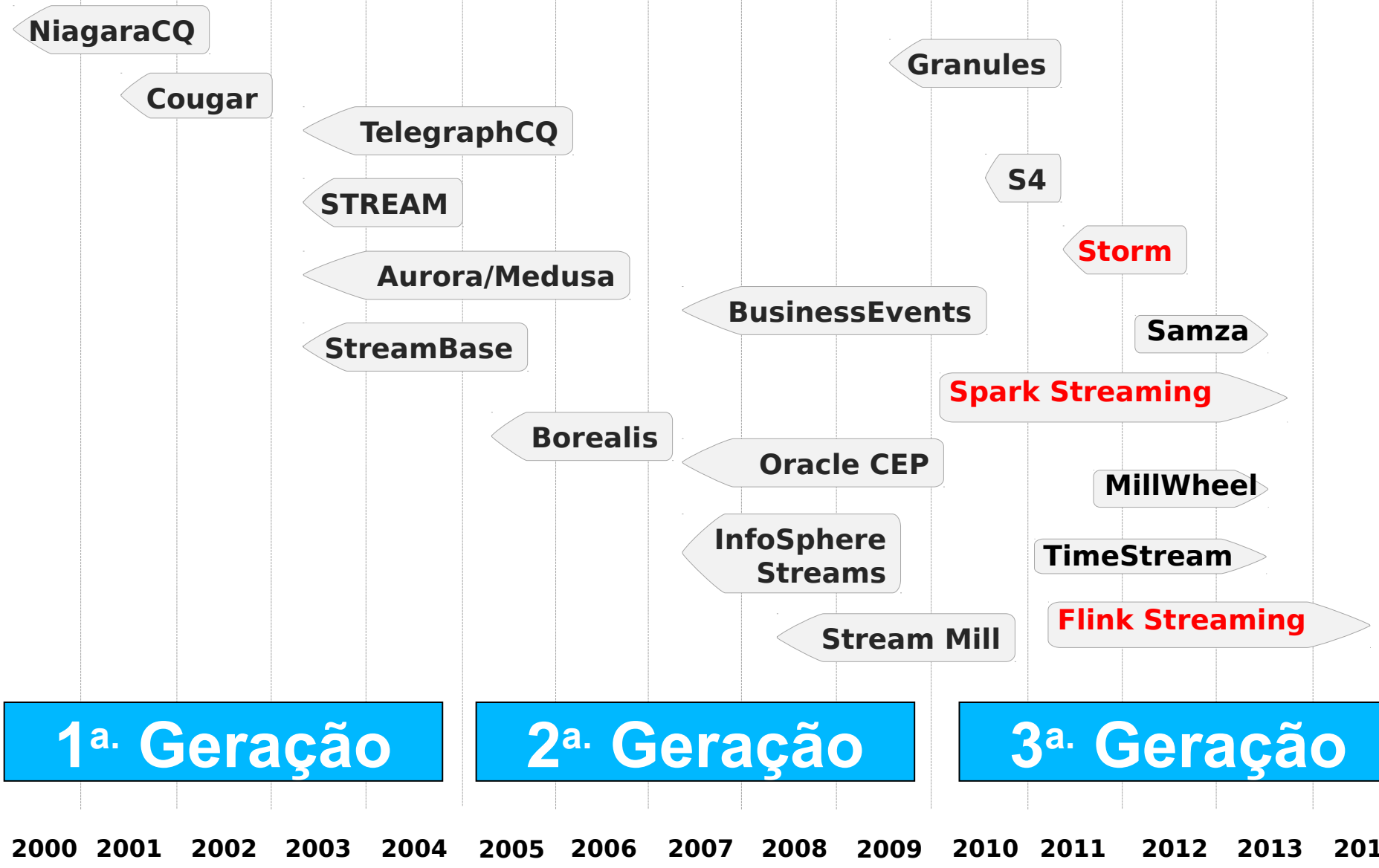
Definição de Janela



Definição de Avanço



Modelo de Programação

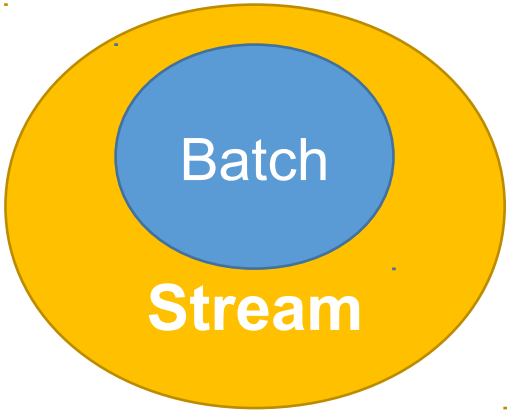

NiagaraCQ
Cougar
TelegraphCQ
STREAM
Aurora/Medusa
StreamBase
Borealis
Granules
S4
Storm
BusinessEvents
Samza
Spark Streaming
Oracle CEP
MillWheel
InfoSphere Streams
TimeStream
Stream Mill
Flink Streaming
1ª. Geração
2ª. Geração
3ª. Geração
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014

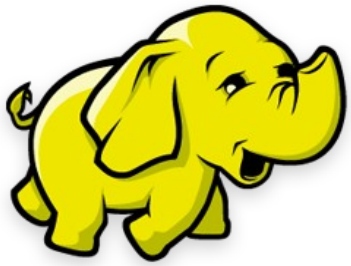
Gerações segundo a maioria dos autores

1ª Geração	2ª Geração	3ª Geração
Continuous Query	Data Stream Management System	Stream Processing System
Centralizado	Distribuído	Massivamente Paralelo
Única Máquina	Sistemas Distribuídos	Cloud computing
Baseados em SQL	SQL-based/ visual drag-and-drop	Linguagens comuns para dadados

Heinze (2014)

Batch Vs. Stream





Hadoop

Batch

Near Real-Time

Storm

Flink

Stream (RT)

Plataformas

Storm

Conceitos (Storm)

- **Proposto inicialmente por Nathan Mars (BackType):**
  - Escrito em Clojure e Java.
- **Utilizado com objetivo principal:**
  - Processamento de dados em tempo real (RT).
  - Principal ambiente SPS (RT) da atualidade.
- **Principal empresa/ferramenta utilizadora do Storm:**
  - Twitter (Adquiriu os direitos do Storm comprando a BackType).
  - Projeto foi doado a Apache posteriormente.

Conceitos (Storm)

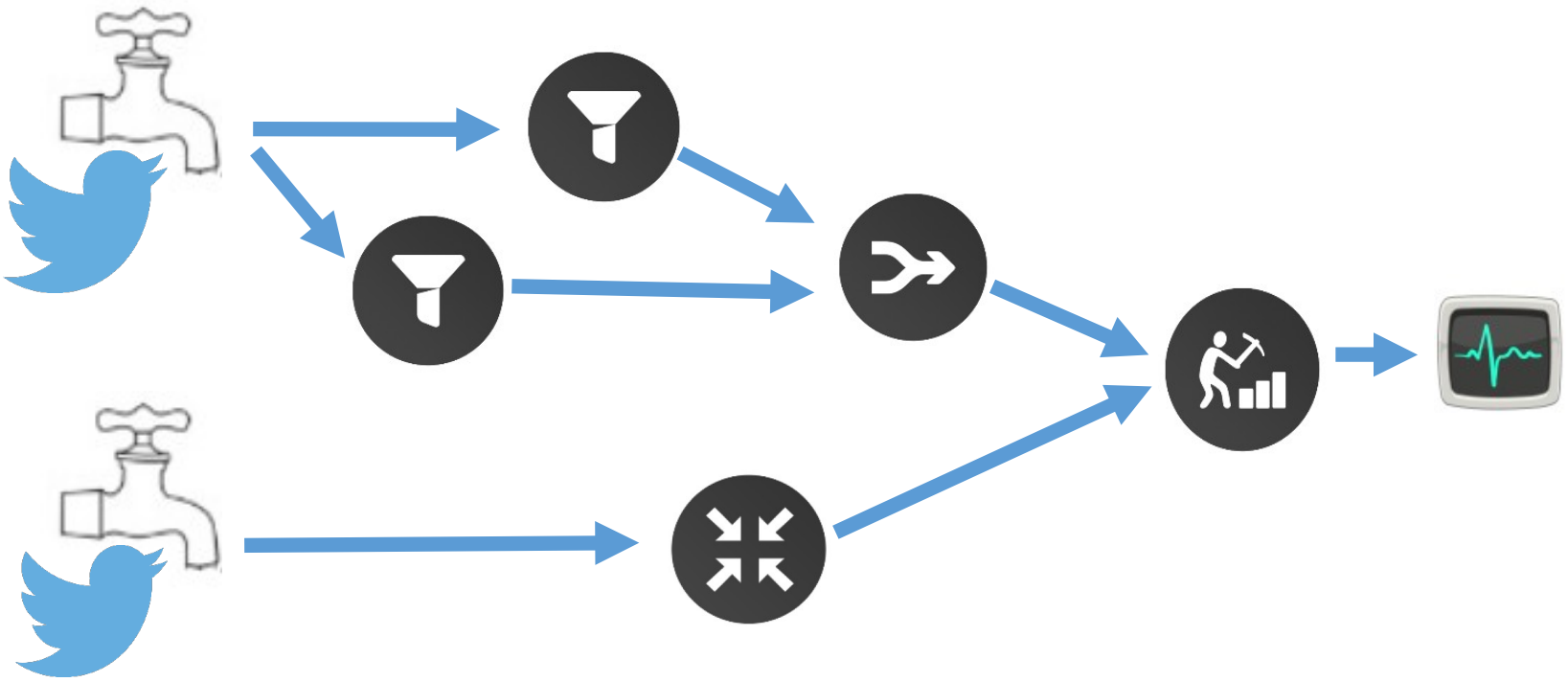
Principais componentes formadores de um grafo:

- **Spout:**
  - Fonte geradora do Streaming (Tuplas).

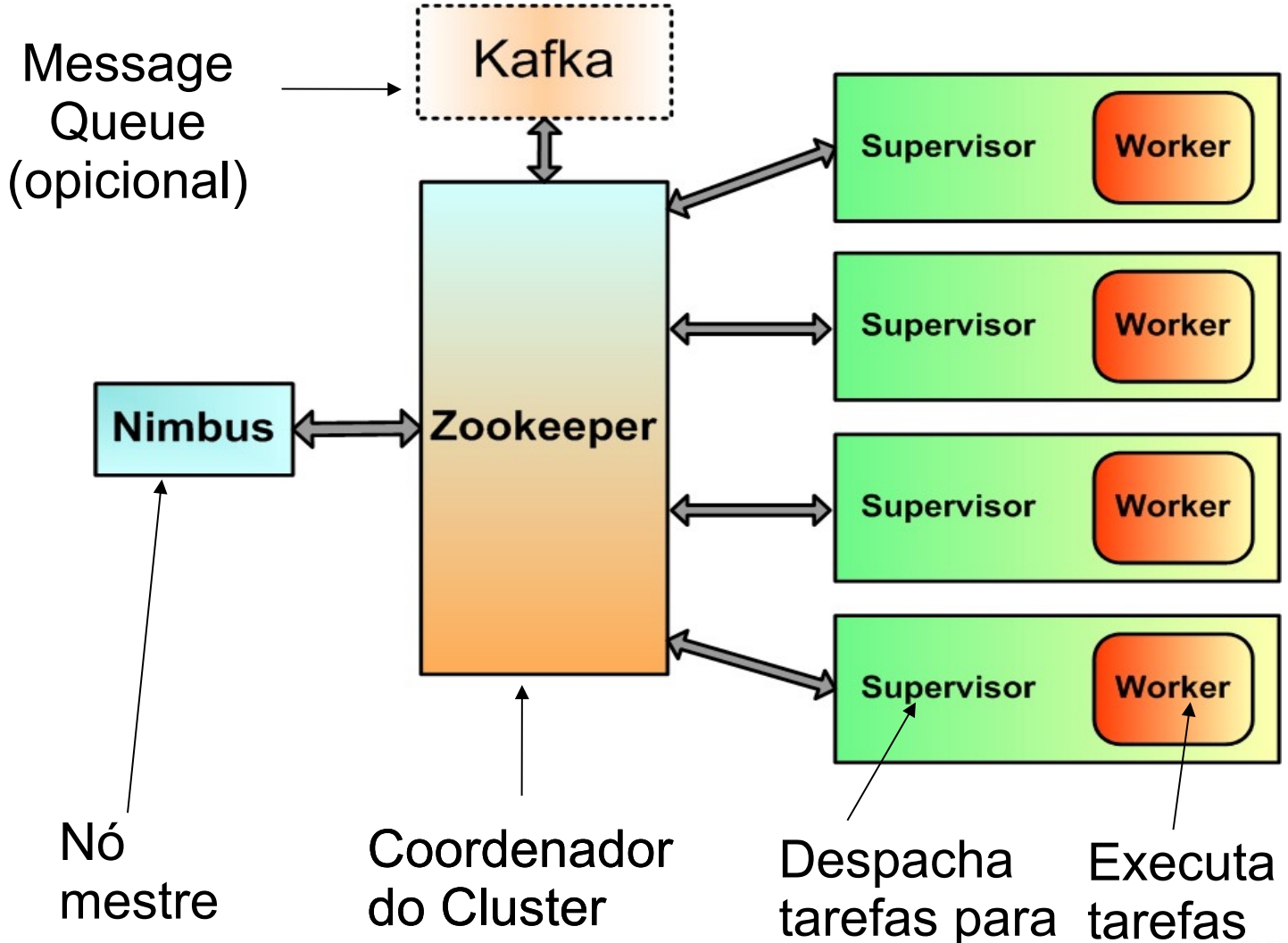
- **Bolt:**
  - Unidades de processamento/transformação dos dados (Tuplas).

- **Sink:**
  - Similar ao bolt porém não gera dados de saída.
  - Usado para exportação de dados do ambiente de SPS.

Topologia

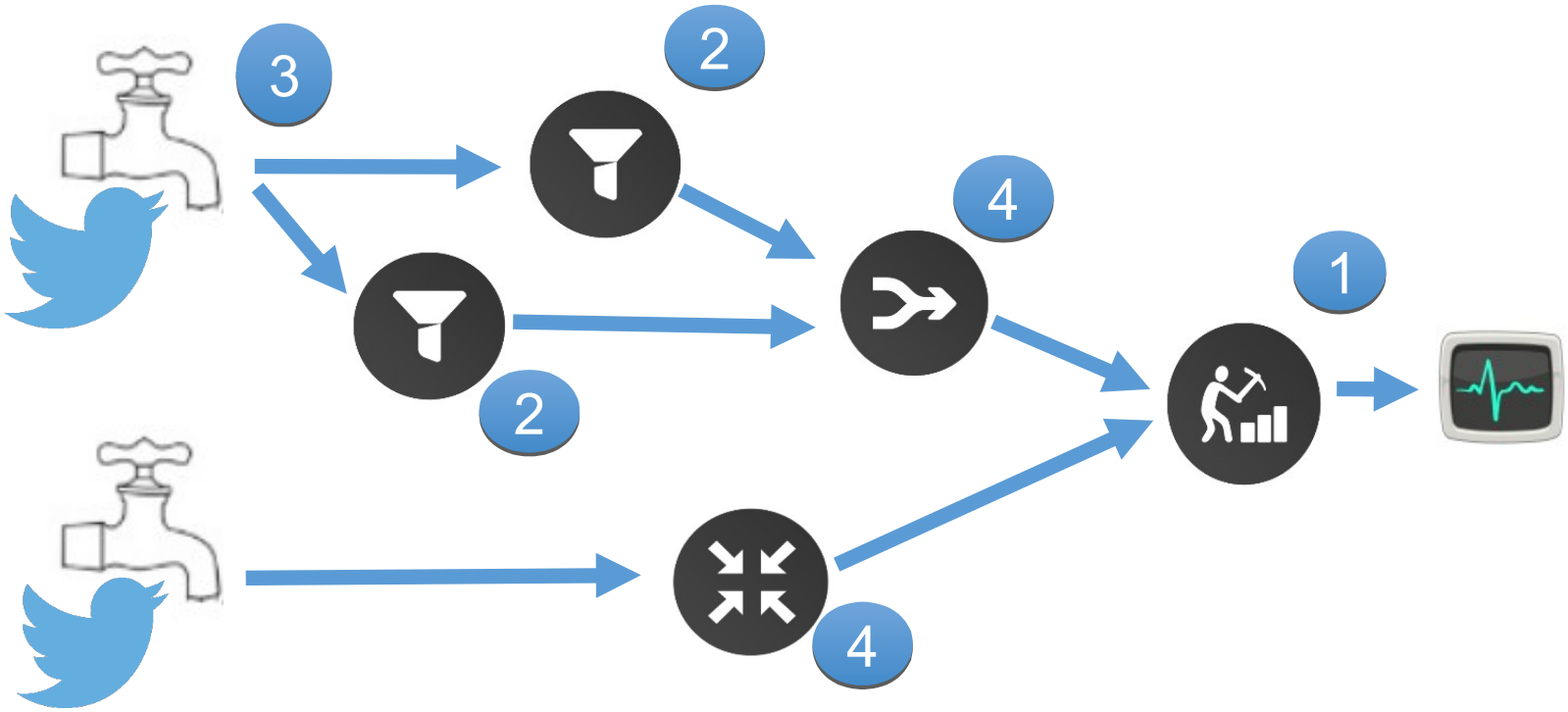


Arquitetura do Storm



Exploração do Paralelismo

Paralelismo Estático



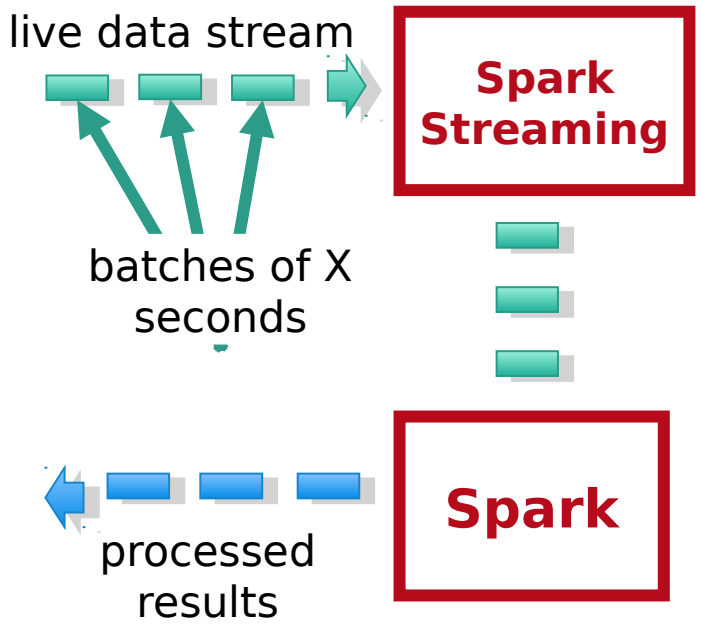
Plataformas

Spark Streaming

Processamento de fluxo de dados discreto

Roda um stream como uma sequência de **jobs determinísticos batchs muito pequenos**

- Corta o live stream em batches de X segundos
- Spark trata cada batch de dados como um RDDs e processa-os usando operadores
- Finalmente, os resultados processados da operação RDD são retornados em batches

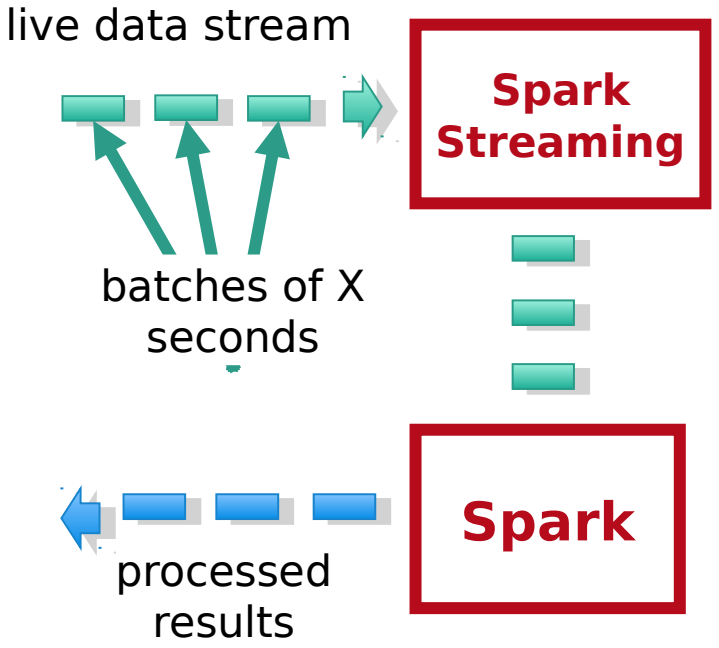


Resilient Distributed Dataset (RDD): é uma estrutura onde os dados estão em memória e podem ser recuperados sem a necessidade de replicação.

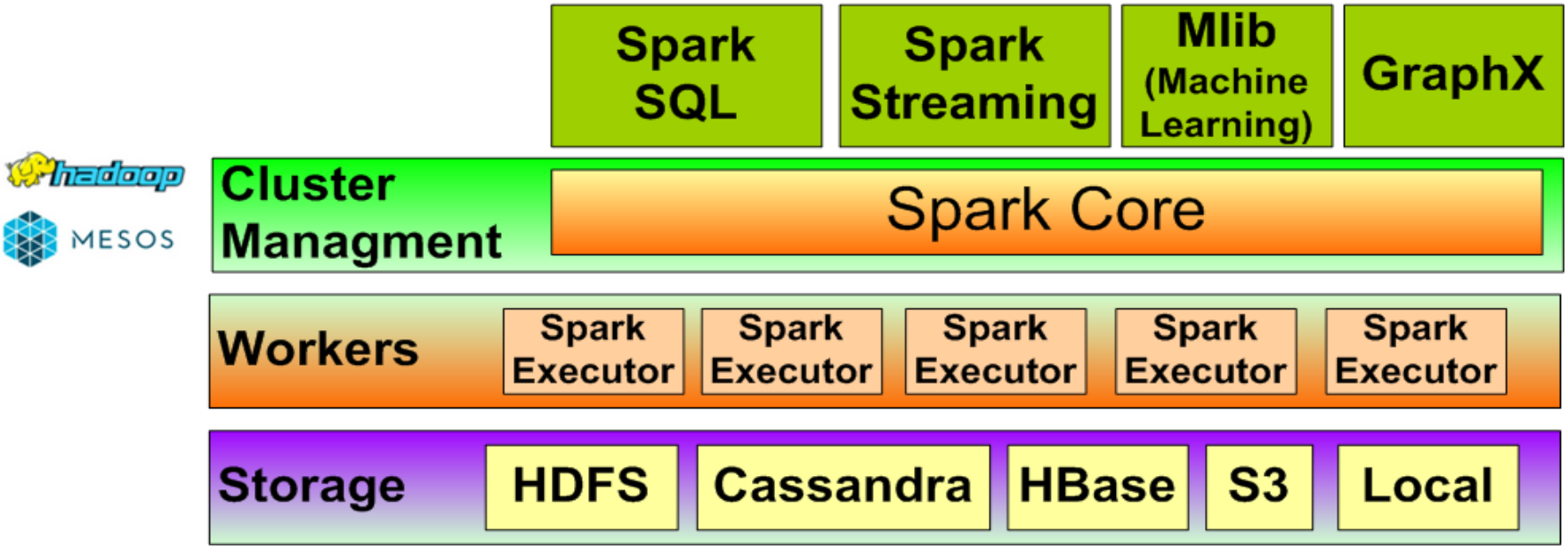
Processamento de fluxo de dados discreto

Roda um streaming como uma sequência de **jobs determinísticos batchs muito pequenos**

- O tamanho do batch é tão pequeno como ½ segundo, latência ~ 1 segundo
- Potencial para combinar processamento em batch e streaming no mesmo sistema



Spark Arquitetura



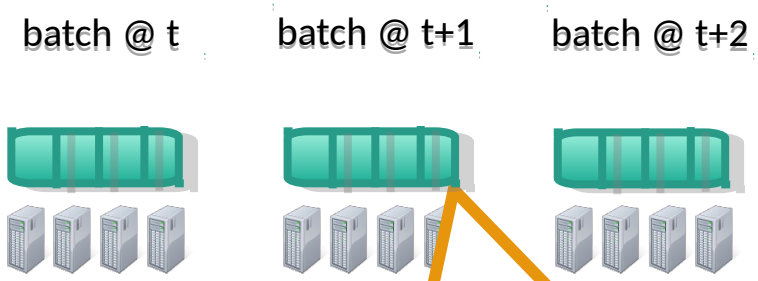
Exemplo 1 – Obter hashtags do Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
```

DStream: uma sequência de RDD representa, um stream de dados

Twitter Streaming API

tweets DStream



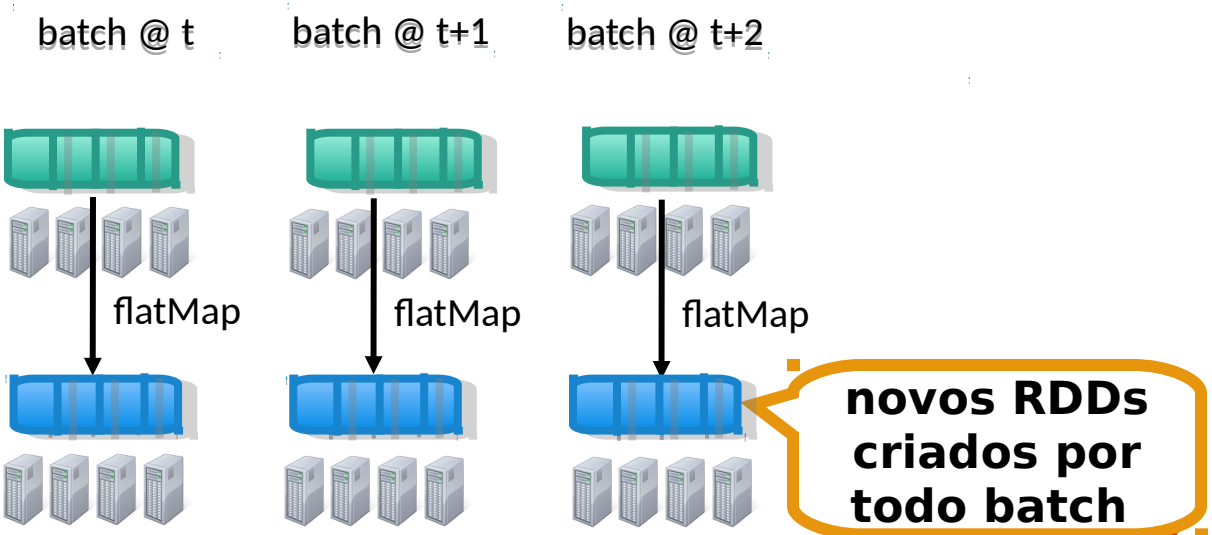
Armazenado na memória como um RDD (immutable, distributed)

Exemplo 1 – Obter hashtags do Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
```

new DStream

transformação: modifica os dados de um Dstream para criar outro DStream

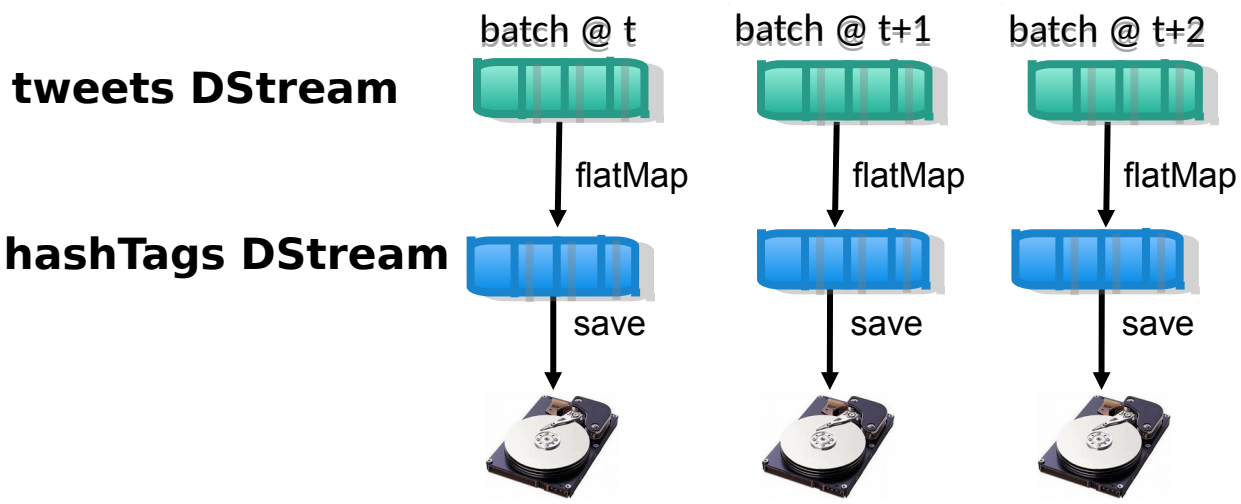


hashTags Dstream
[#cat, #dog, ...]

Example 1 – Obter hashtags do Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

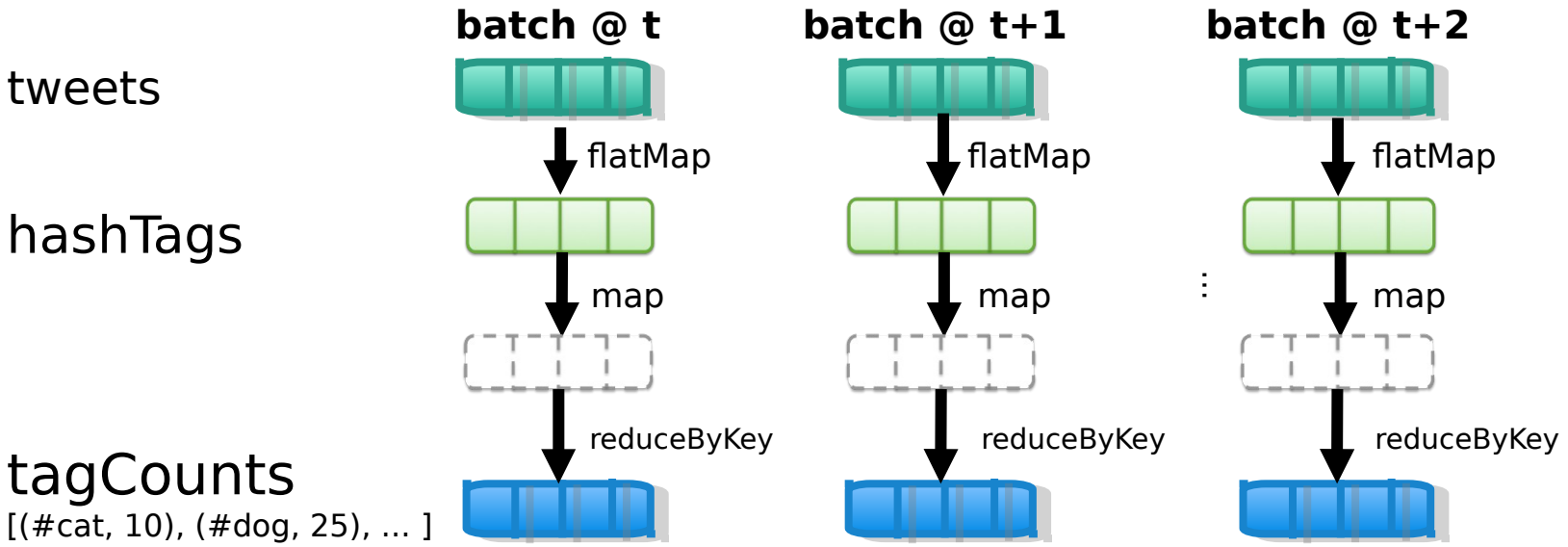
output operation: envia dados para o armazenamento externo



Cada batch é gravado no HDFS

Exemplo 2 – Contar as hashtags

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```

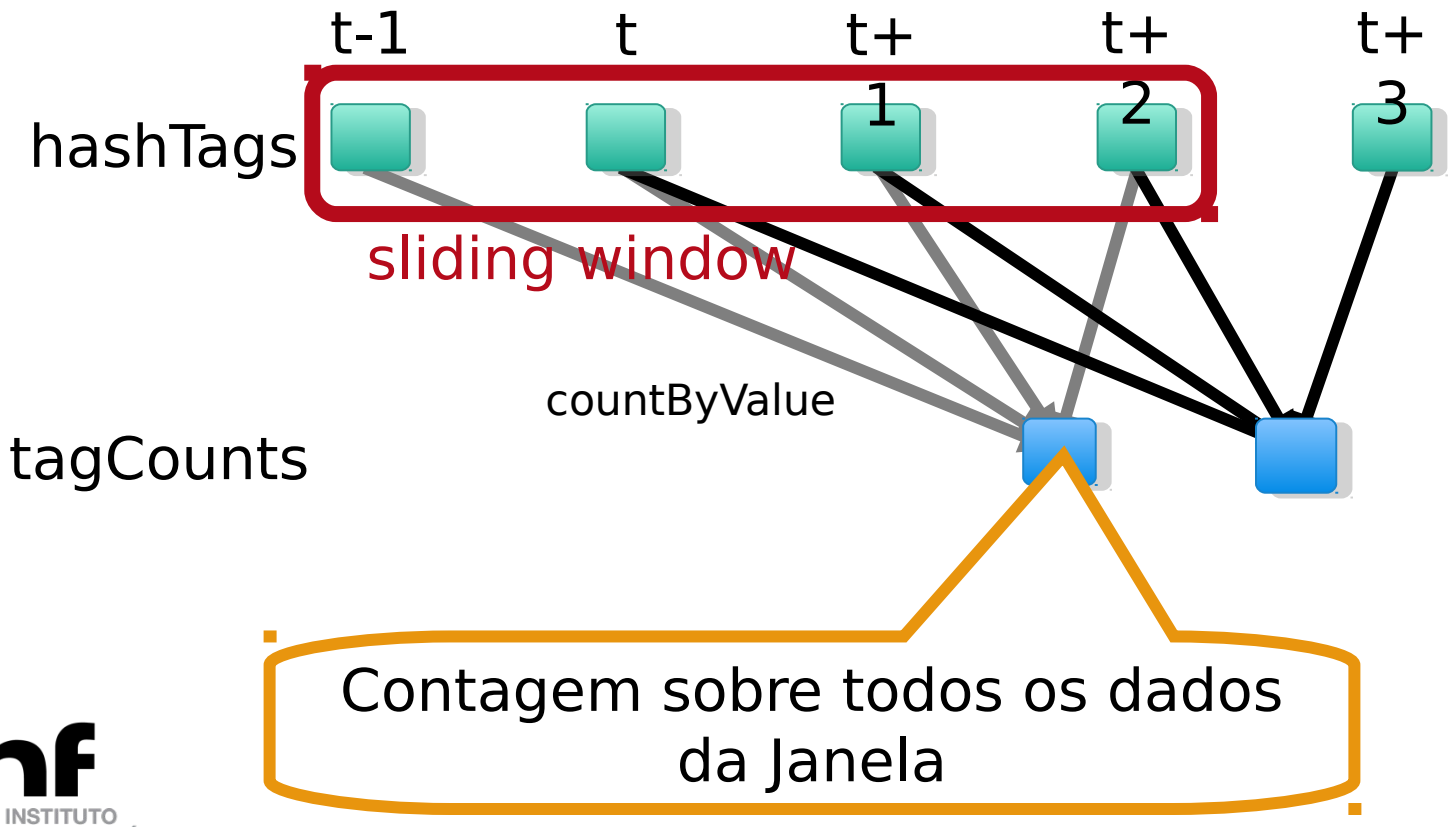


Exemplo 3 – Contagem de hashtags nos últimos 10 min

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```

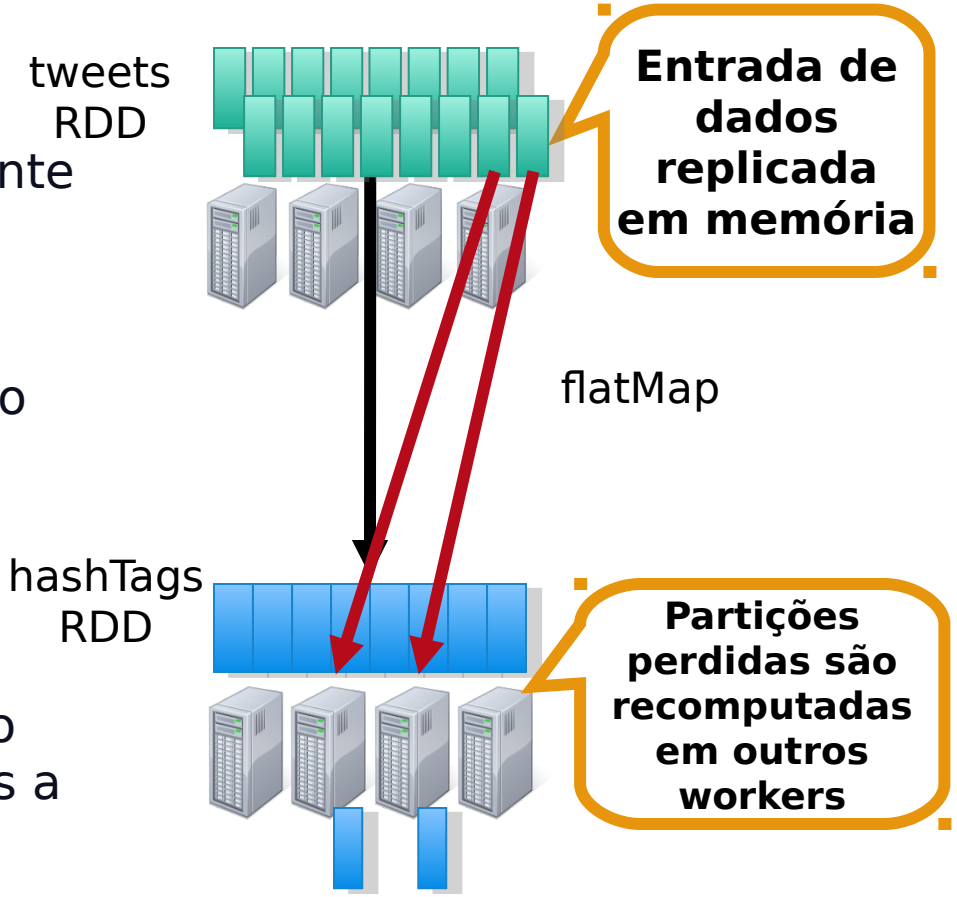
Example 3 – Contagem de hashtags nos últimos 10 min

```
val tagCounts = hashTags.window(Minutes(10),Seconds(1)).countByValue()
```



Mecanismo de Tolerância a Falhas

- RDDs sabem a seqüência de operações que o criou de uma entrada de dados original tolerante a falhas
- Batches de entrada de dados são replicados em memória de múltiplos nós workers, sendo assim tolerante a falhas
- Data perdidos devido a falhas do worker podem ser recomputados a partir da entrada de dados.

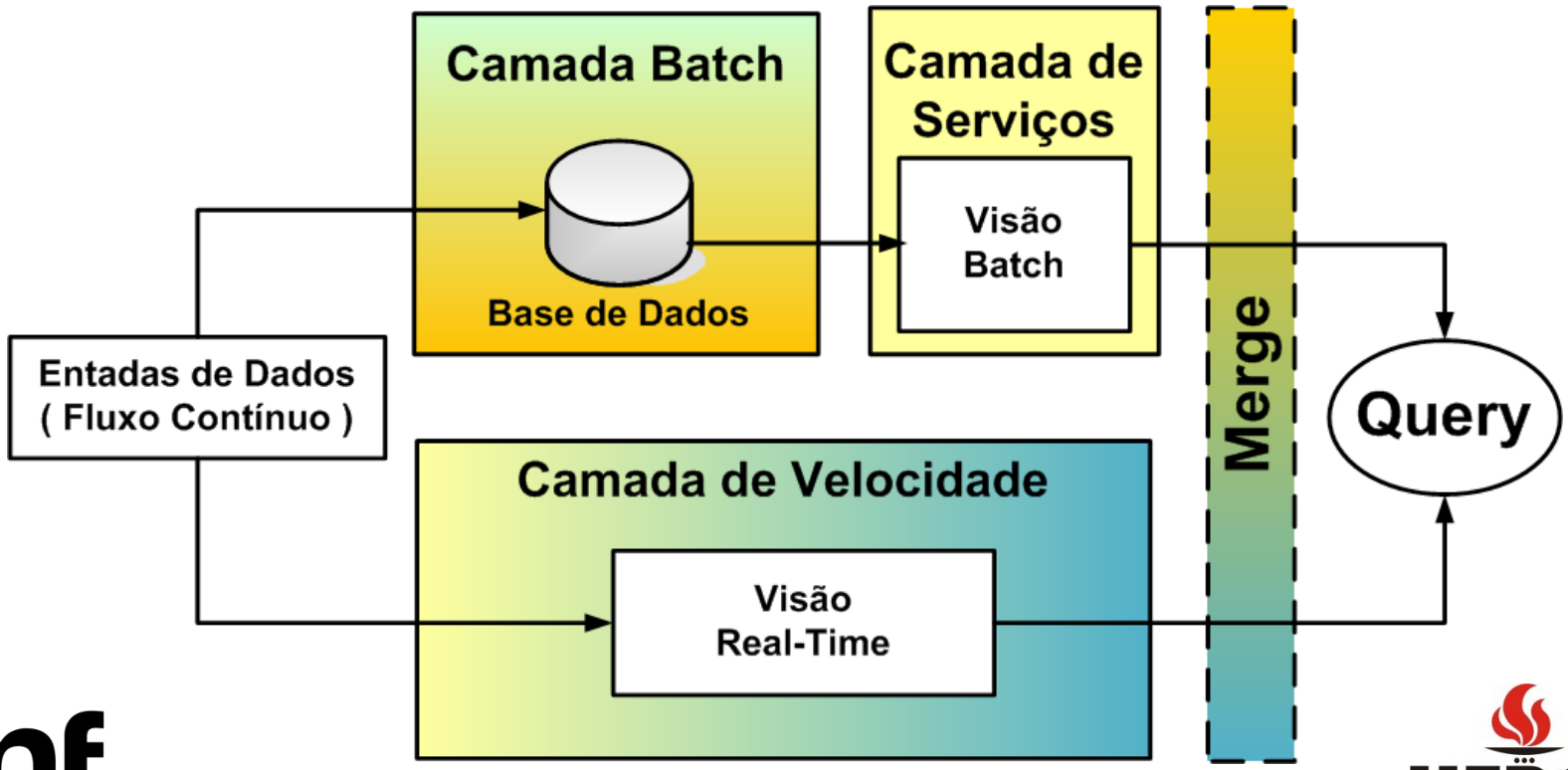


Plataformas

Flink

Arquitetura Lambda

A arquitetura Lambda [Marz, 2013], seria mais adequada para descrever os atuais sistemas de Big Data [Kir15], [Cae16] .



4ª. Geração de processamento de dados

- Alguns autores importantes como o Buyya [Cae16] , redefinem o Flink como um sistema de 4ª. Geração por suportar:
  - Algoritmos Iterativos;
  - Possuir mecanismos internos de otimização;
  - Por seguir o modelo da arquitetura Lambda;
  - Por ser uma arquitetura de programação híbrida;
    - Permitir execuções em batch e real-time simultâneo.

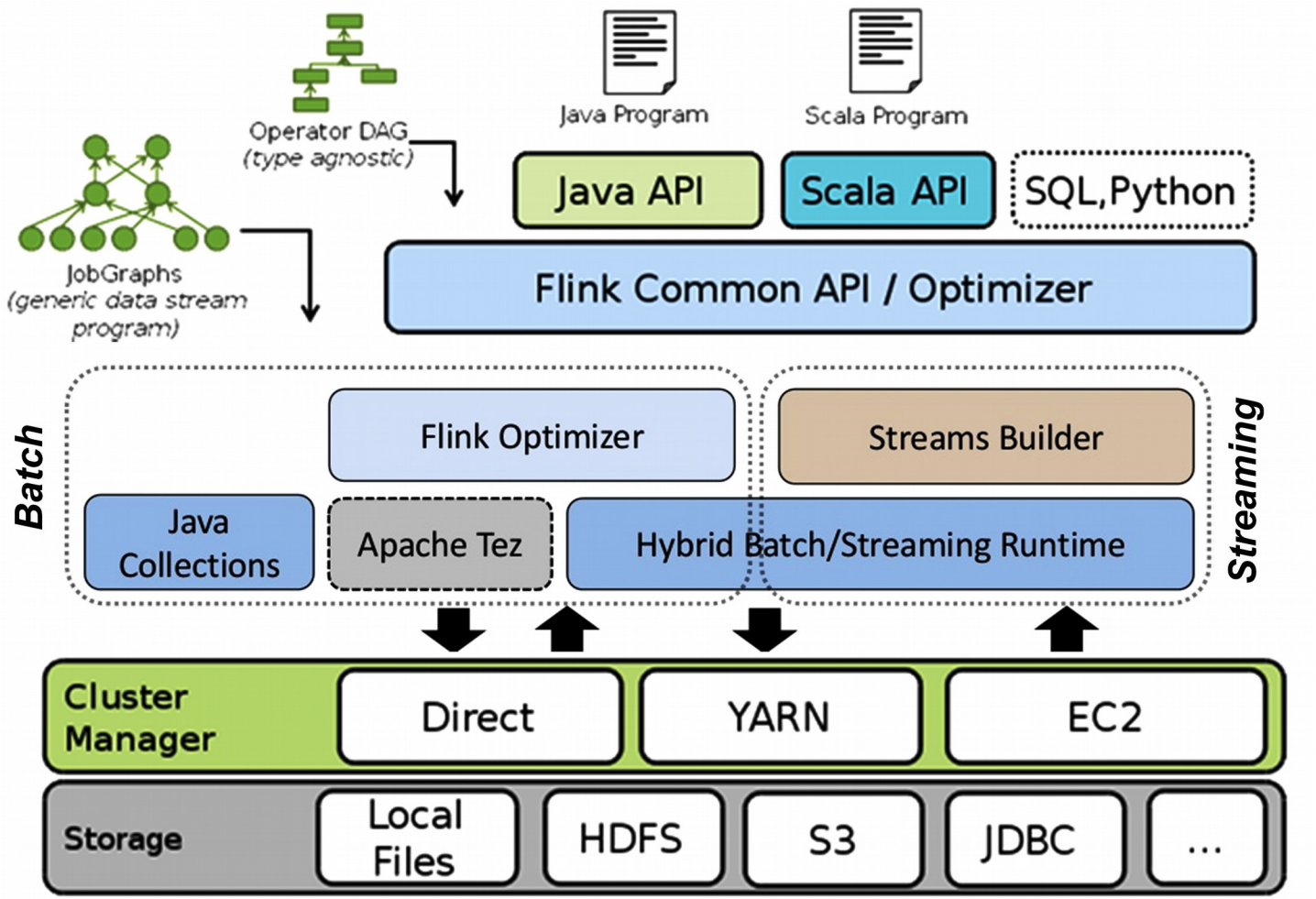
Plataforma Apache Flink

- Processamento batch e/ou stream (programação híbrida)
- Abstrações de programação em Java, Scala e Python
- Tem um gerenciador de execução de alto desempenho e otimização automática de código.
- Suporte nativo para iterações incrementais ou não incrementais
- Expressa DAGs de operações.

Plataforma Apache Flink

- A programação de transformações de conjuntos de dados.
  - Os conjuntos de dados são inicialmente criados a partir de alguma fonte (leitura de arquivos ou coleções locais).
- Os resultados são retornados via sinks.
- Não tem um Storage próprio.
- Grava diretamente em um arquivo distribuído ou em uma saída padrão.

Arquitetura do Apache Flink



Transformações Apache Flink

- **FlatMap:** A partir de um elemento produz zero ou mais elementos;
- **MapPartition:** Transforma uma partição paralela em uma única chamada de função e produz um número arbitrário de resultados;
- **Filter:** Avalia uma função booleana para cada elemento e mantém aqueles para os quais a função retorna verdadeiro;
- **Reduce:** Combina um grupo de elementos em um único elemento;
- **Aggregate:** Agrega um conjunto de valores em um único valor;
- E outras

Exemplo 1– WordCount - batch

```
public class WordCountExample {
    public static void main(String[] args) throws Exception {
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

        DataSet<String> hdfsLines = env.readTextFile("hdfs://nnHost:nnPort/path/to/my/textfile");

        DataSet<Tuple2<String, Integer>> wordCounts = text
            .flatMap(new LineSplitter())
            .groupBy(0)
            .sum(1);
        wordCounts.print();
    }

    public static class LineSplitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
        @Override
        public void flatMap(String line, Collector<Tuple2<String, Integer>> out) {
            for (String word : line.split(" ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }
}
```

Exemplo 2– WordCount - Streaming

```
public class WindowWordCount {
    public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<Tuple2<String, Integer>> dataStream = env
                .socketTextStream("localhost", 9999)
                .flatMap(new Splitter())
                .keyBy(0)
                .timeWindow(Time.seconds(5))
                .sum(1);
        dataStream.print();
        env.execute("Window WordCount");
    }

      public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
        @Override
        public void flatMap(String sentence, Collector<Tuple2<String, Integer>> out) throws Exception {
            for (String word: sentence.split(" ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }
}
```

Comparação entre Plataformas

Plataforma	Storm	Spark Streaming	Apache Flink
Ano	2011	2013	2014
Desenvolvedor	BackType	Berkeley	Technische Universitat Berlin
Categoria de Programação	Record-at-a-time	Micro-batch	Batch, Real-time e Micro-batch
Modelo de Programação	DAG	Monad	Dataflow Cíclico
Rebalanceamento	Sim	Não	Sim
Cluster Dinâmico	Sim	Sim	Não
Gerenciamento de Recursos	Standalone, Yarn, Mesos	Standalone, Yarn, Mesos	Standalone, Yarn, Mesos, Apache Tez e Cloud
Coordernação	Zookeeper	Built-in	Built-in
Persistência de Estado	Não	Sim	Sim
Grafo Dinâmico	Não	-	Sim

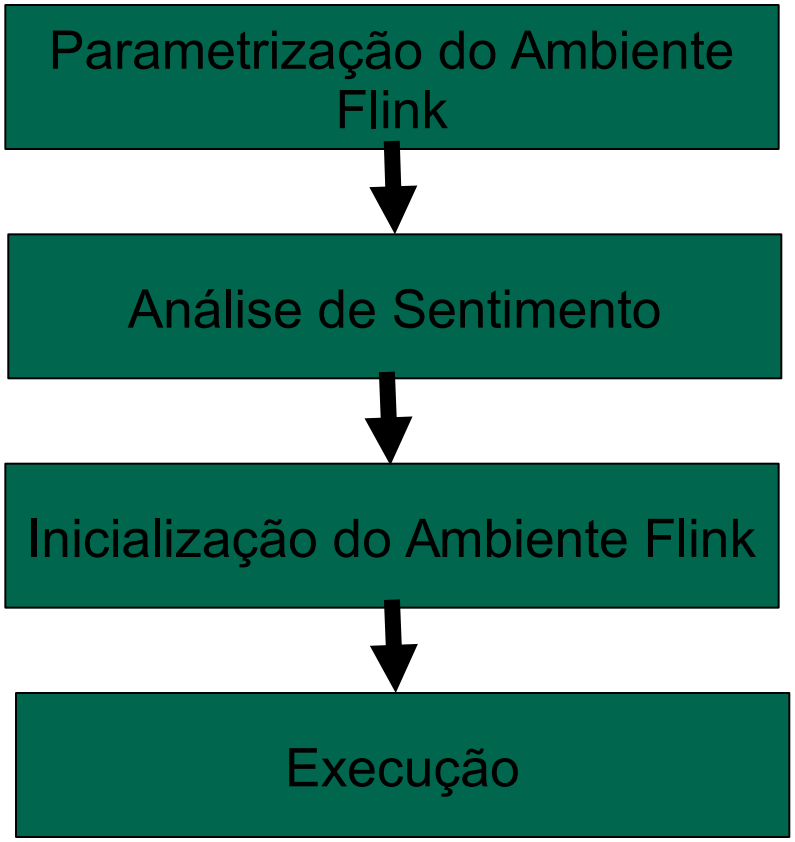
Plataforma	Storm	Spark Streaming	Apache Flink
Linguagem de Programação	Java, qualquer c/ Thrift	Java, Scala, Phyton	Java, Scala, Phyton
Linguagem de implementação	Java, Clojure	Java, Scala	Java, Scala
Operadores Nativos	Não	Sim	Sim
Determinístico	-	Sim	-
Sistema de Mensagens	Netty	Netty, Kafka	Kafka
Mobilidade de Dados	Pull	-	Pull
Garantias de Entrega	AMO, ALO	EO	EO, AMO, ALO
Tolerância à Falhas	Rollback recovery com upstream backup	Checkpoint, replication, parallel recovery	Replication, parallel recovery

EO (exactly once) , AMO (at most once) e ALO (at least once)
(uma única vez) (no máximo uma vez) (pelo menos uma vez)

Vídeos com exemplos de programação

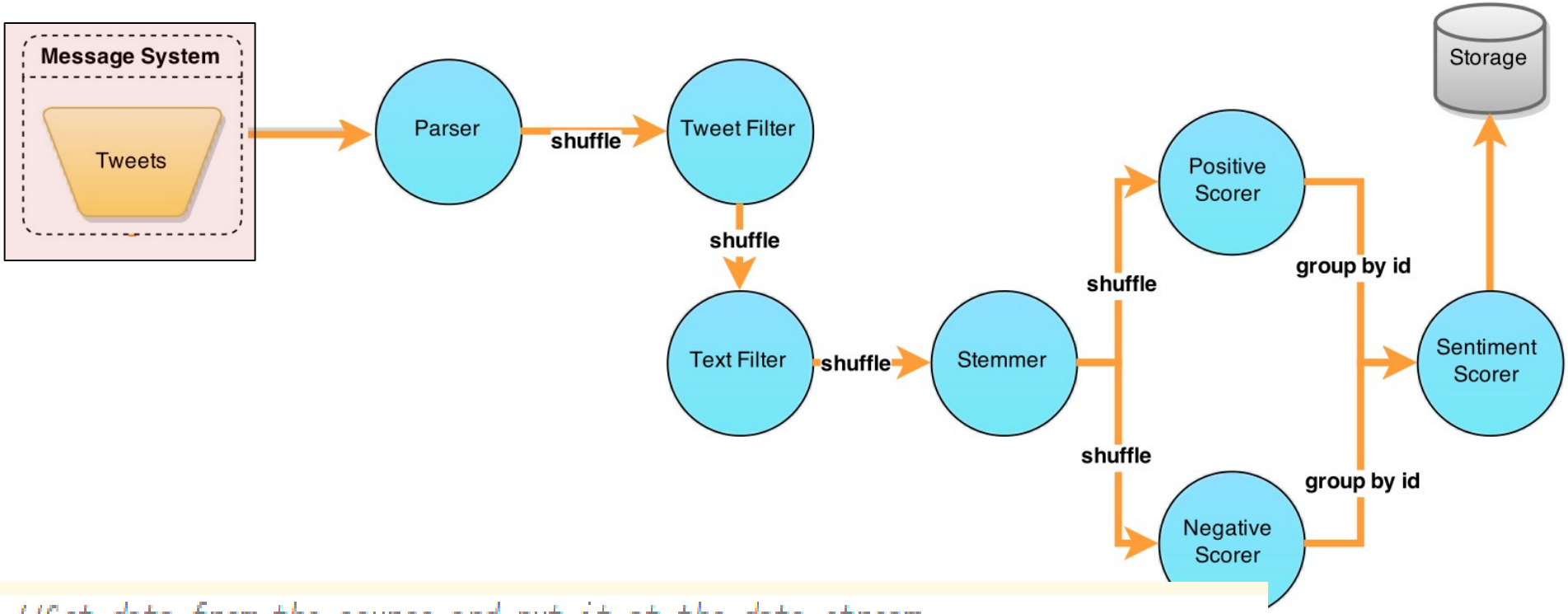
https://github.com/mayconbordin/erad2016-streamprocessing

Fluxo de Execução do APP Análise de Sentimento



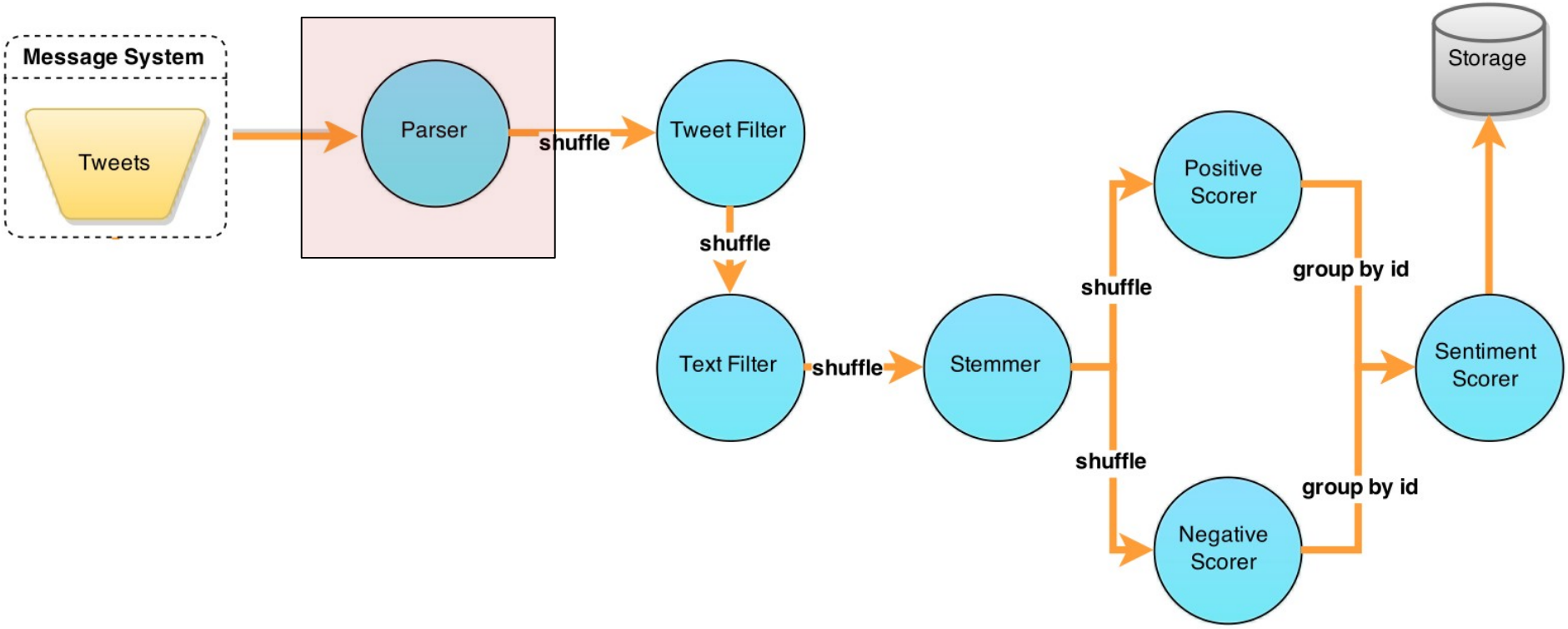
Parametrização do Ambiente Flink

```
// Get input parameters
final ParameterTool params = ParameterTool.fromArgs(args);
// set up the execution environment
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
// make parameters available in the web interface
env.getConfig().setGlobalJobParameters(params);
//set up the parallelism equal
env.setParallelism(params.getInt("parallelism", 1));
```



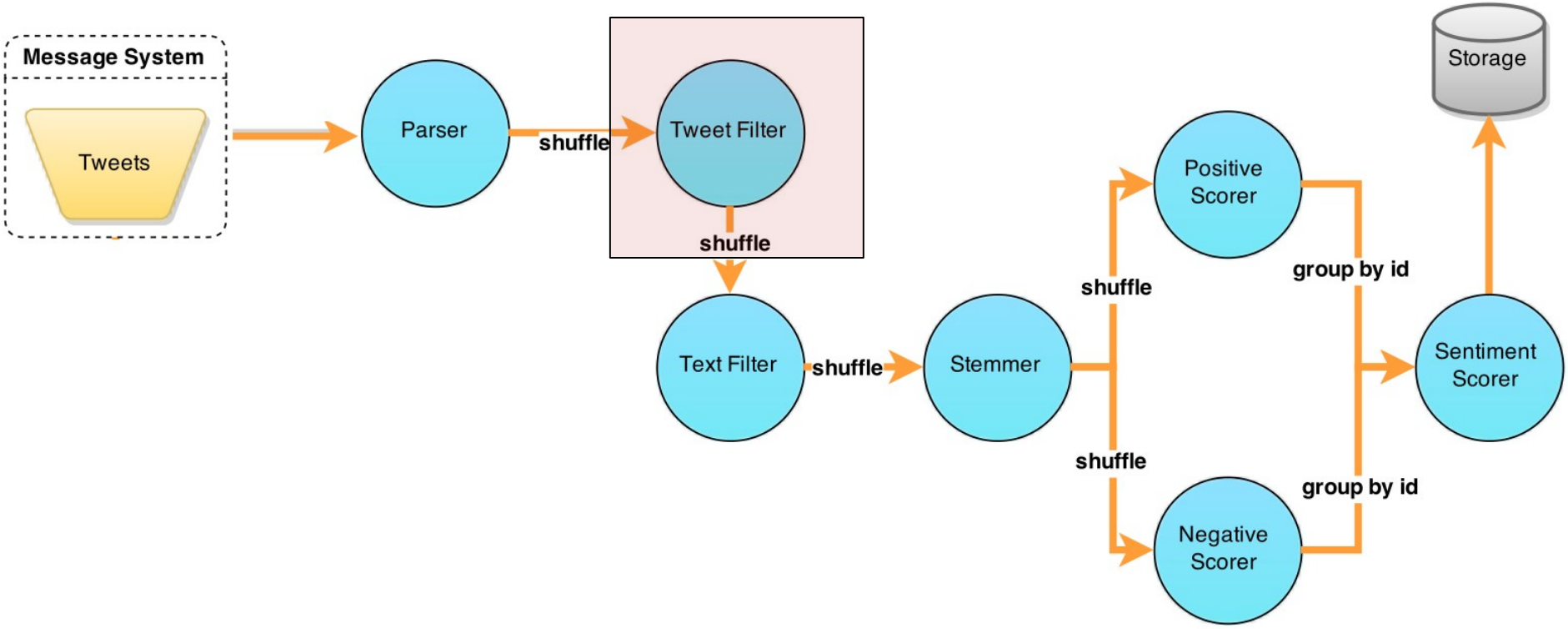

```
//Get data from the source and put it at the data stream
DataStream<String> streamSource = env.fromElements(TwitterExampleData.TEXTS);
```

```
//example data looking like tweets, but not acquired from Twitter
public class TwitterExampleData {
    public static final String[] TEXTS = new String[] {
            "{\"created_at\":\"Mon Jan 1 00:00:00 +0000 1901\",\"id\":0,\"id_str\":\"00000000000000000\",\
            "{\"created_at\":\"Mon Jan 1 00:00:00 +0000 1901\",\"id\":1,\"id_str\":\"00000000000000001\",\
            "{\"created_at\":\"Mon Jan 1 00:00:00 +0000 1901\",\"id\":2,\"id_str\":\"00000000000000002\",\
    };

    private TwitterExampleData() {
    }
}
```

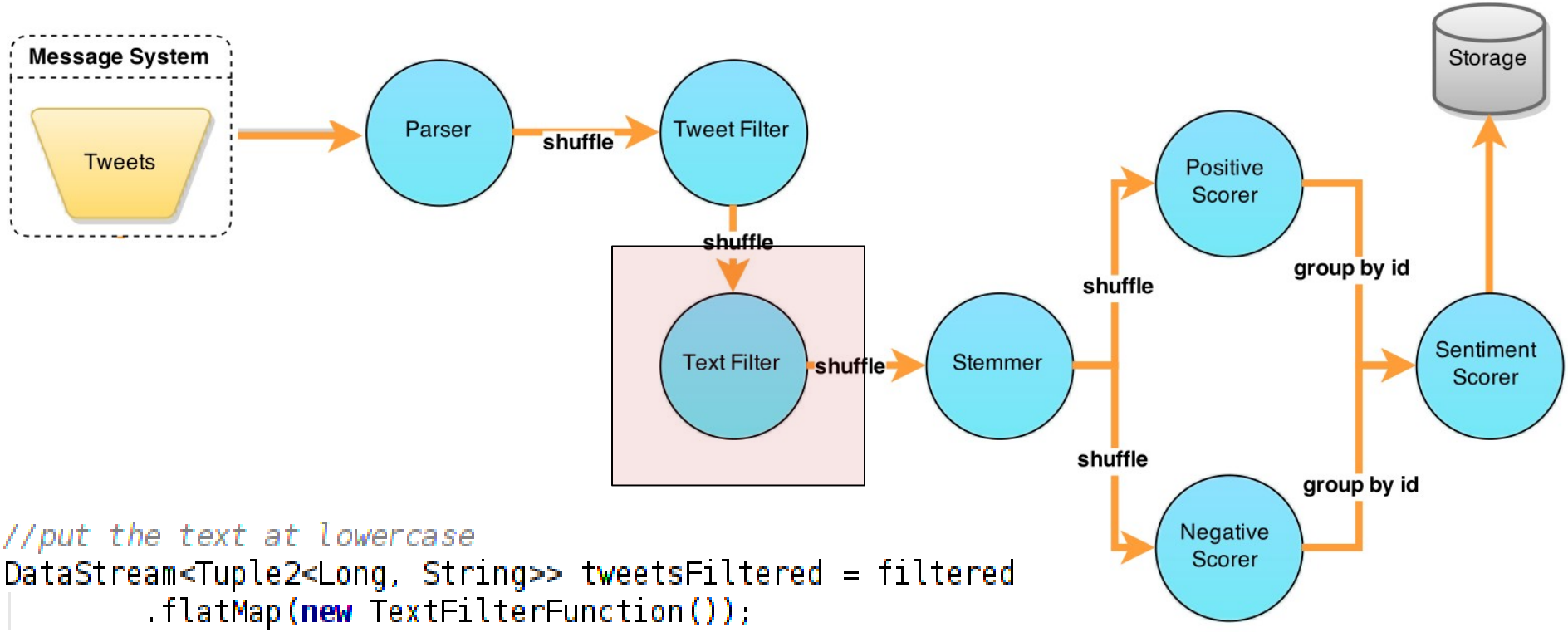



```
//get the ID and the text from the tweet, only English text-> using Json protocol
DataStream<Tuple2<Long, String>> tweets = streamSource
        .flatMap(new TwitterFilterFunction());
```

```java
public class TwitterFilterFunction implements FlatMapFunction<String, Tuple2<Long, String>> {
    private static final long serialVersionUID = 1L;

    private transient ObjectMapper jsonParser;

    /**
     * Select the language from the incoming JSON text
     */
    @Override
    public void flatMap(String value, Collector<Tuple2<Long, String>> out) throws Exception {
        if (jsonParser == null) {
            jsonParser = new org.codehaus.jackson.map.ObjectMapper();
        }
        JsonNode jsonNode = jsonParser.readValue(value, JsonNode.class);
        boolean isEnglish = jsonNode.has("user")
                                && jsonNode.get("user").has("lang")
                                && jsonNode.get("user").get("lang").getTextValue().equals("en");
        boolean hasText = jsonNode.has("text");
        boolean hasId = jsonNode.has("id");

        if (isEnglish && hasText && hasId) {
            // message of tweet
            String result = jsonNode.get("text").getTextValue();
            Long id = jsonNode.get("id").getLongValue();

                if (!result.equals("")) {
                    out.collect(new Tuple2<>(id, result));
                }
            }

    }
}
```

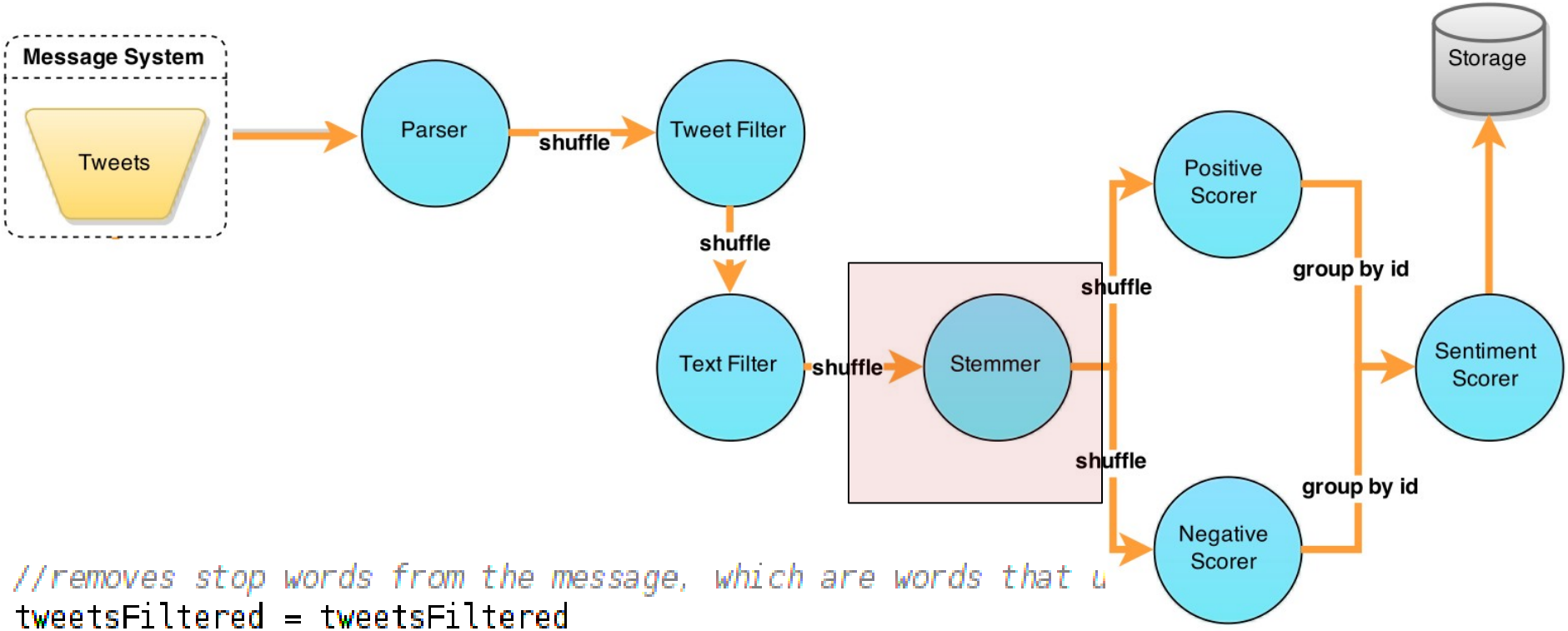



```
//Delete the null tweets
DataStream<Tuple2<Long, String>> filtered = tweets.filter(
        tweet -> tweet != null
);
```



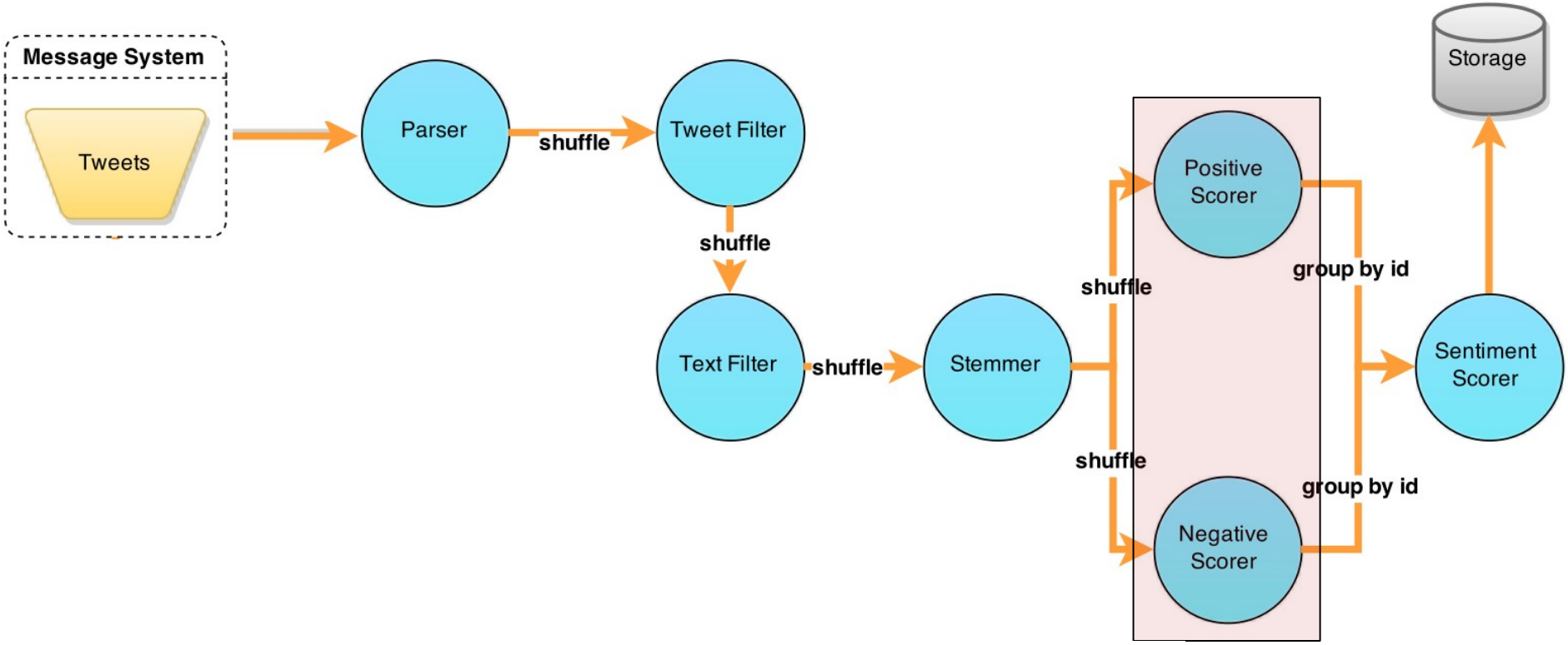

```
//put the text at lowercase
DataStream<Tuple2<Long, String>> tweetsFiltered = filtered
        .flatMap(new TextFilterFunction());
```

```
public class TextFilterFunction
    implements FlatMapFunction<Tuple2<Long, String>, Tuple2<Long, String>>
{
    private static final long serialVersionUID = 42l;

    @Override
    public void flatMap(Tuple2<Long,String> tweet,Collector<Tuple2<Long, String>> out) throws Exception
    {
        String text = tweet.f1;
        text = text.replaceAll("[^a-zA-Z\\s]", "").trim().toLowerCase();
        out.collect(new Tuple2<>(tweet.f0, text));
    }

}
```



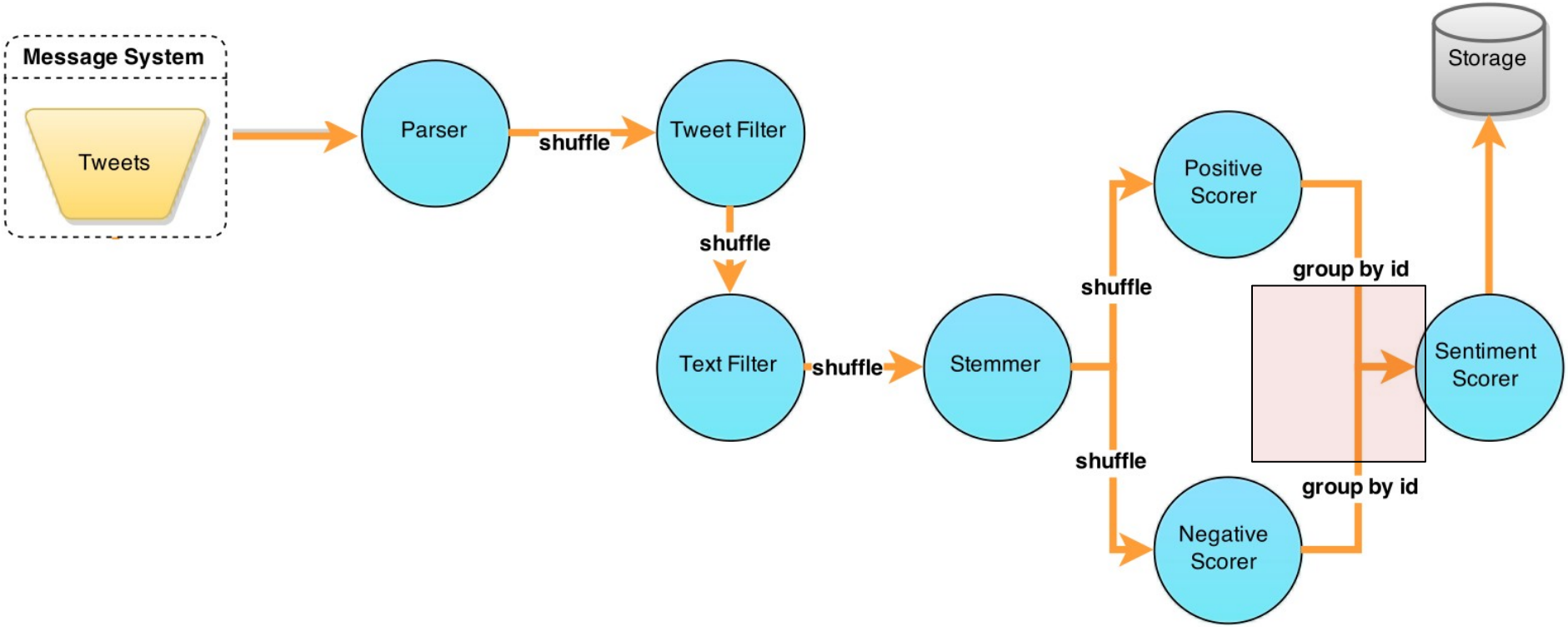

```
//removes stop words from the message, which are words that u
tweetsFiltered = tweetsFiltered
        .flatMap(new StemmingFunction());
```

```
public class StemmingFunction
    implements FlatMapFunction<Tuple2<Long, String>, Tuple2<Long, String>>
{
    private static final long serialVersionUID = 42l;

    public void flatMap(Tuple2<Long,String> tweet,Collector<Tuple2<Long, String>> out) throws Exception {
        String text = tweet.f1;
        List<String> stopWords = StopWords.getWords();
        for (String word : stopWords) {
            text = text.replaceAll("\\b" + word + "\\b", "");
        }

        out.collect(new Tuple2<>(tweet.f0, text));


    }
}
```

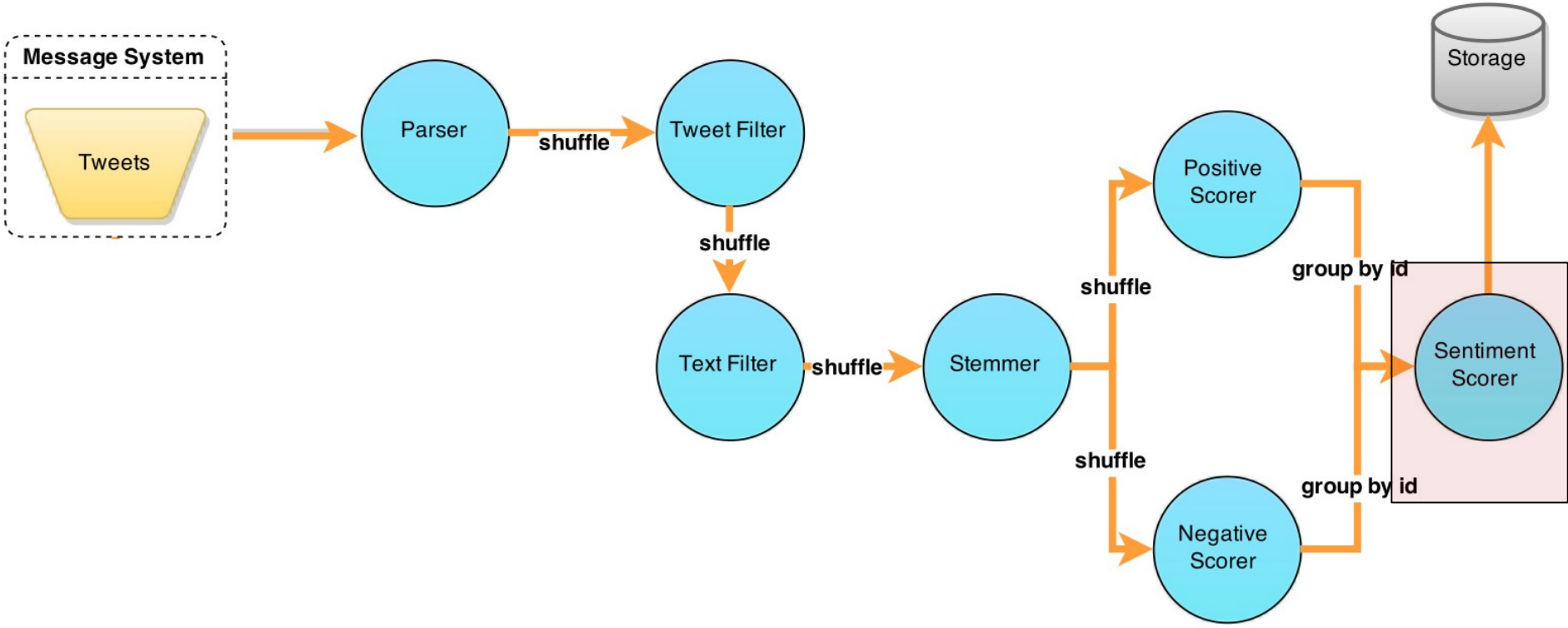



```
//count positive words from a dictionary
DataStream<Tuple3<Long, String, Float>> positiveTweets =
        tweetsFiltered.flatMap(new PositiveScoreFunction());

//count negative words from a dictionary
DataStream<Tuple3<Long, String, Float>> negativeTweets =
        tweetsFiltered.flatMap(new NegativeScoreFunction());
```

```java
public class PositiveScoreFunction
        implements FlatMapFunction<Tuple2<Long, String>, Tuple3<Long, String, Float>>

{
    private static final long serialVersionUID = 42l;


    @Override
    public void flatMap(Tuple2<Long, String> tweet,
                        Collector<Tuple3<Long, String, Float>> out) throws Exception {

        String text = tweet.f1;
        Set<String> posWords = PositiveWords.getWords();
        String[] words = text.split(" ");
        int numWords = words.length;
        int numPosWords = 0;
        for (String word : words) {
            if (posWords.contains(word))
                numPosWords++;
        }

        out.collect(new Tuple3<>(
                tweet.f0,
                tweet.f1,
                (float) numPosWords / numWords));

    }
}
```

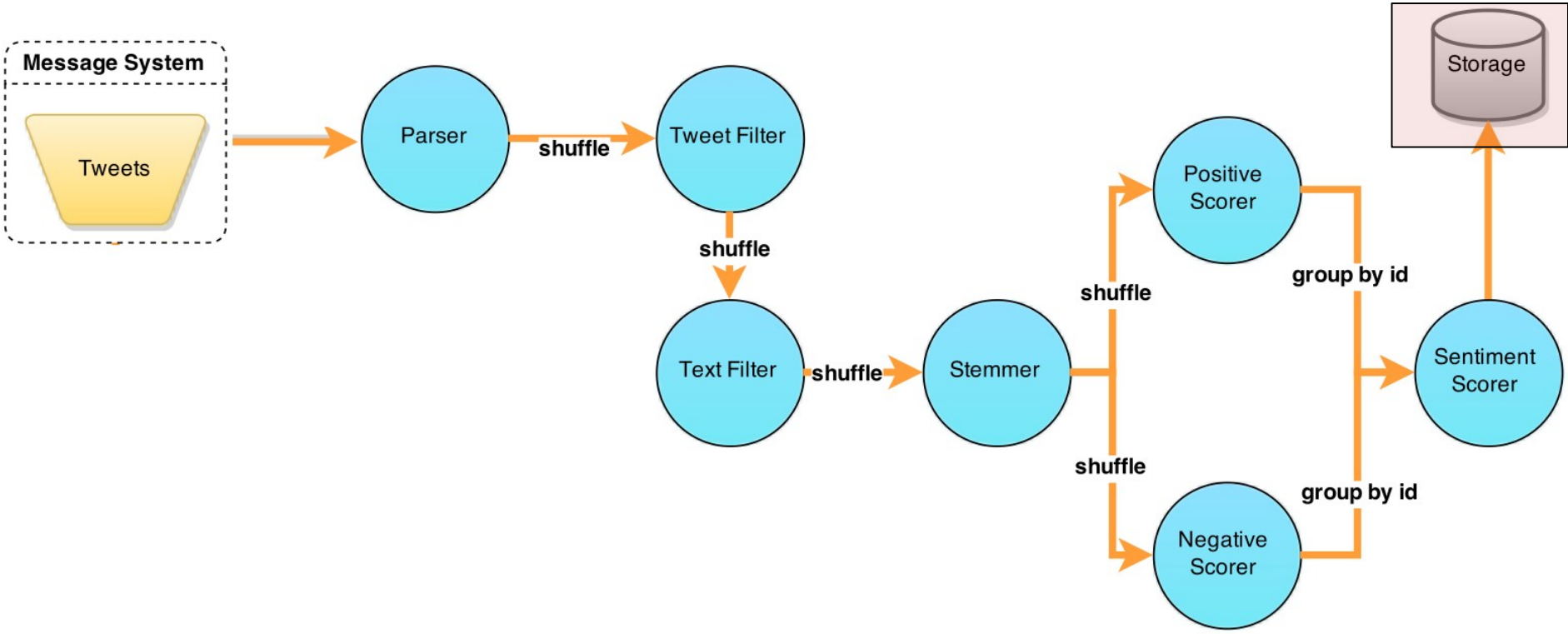

Message System
Tweets
Parser
shuffle
Tweet Filter
shuffle
Text Filter
shuffle
Stemmer
shuffle
shuffle
Positive Scorer
Negative Scorer
group by id
group by id
Sentiment Scorer
Storage

```
//join the positive and negative data streams
DataStream<Tuple4<Long, String, Float, Float>> scoredTweets = positiveTweets
        .join(negativeTweets)
        .onWindow(1000, TimeUnit.MILLISECONDS)
        .where(0,1)
        .equalTo(0,1)
        .with(new JoinFunction<Tuple3<Long, String, Float>,
                               Tuple3<Long, String, Float>,
                    Tuple4<Long, String, Float, Float>>() {
            @Override
            public Tuple4<Long,
                          String,
                          Float,
                          Float>
                join(Tuple3<Long,
                            String,
                            Float> positive,
                     Tuple3<Long,
                             String,
                             Float> negative)
                    throws Exception {
                return new Tuple4<>(positive.f0,
                                    positive.f1,
                                    positive.f2,
                                    negative.f2);
            }
        });
```




```
//determine if the tweet was positive or negative
DataStream<Tuple5<Long, String, Float, Float, String>> result =
        scoredTweets.flatMap(new ScoreTweetsFunction());
```

```java
public class ScoreTweetsFunction
    implements FlatMapFunction<Tuple4<Long, String, Float, Float>,
                               Tuple5<Long, String, Float, Float, String>>
{
    private static final long serialVersionUID = 42l;


    @Override
    public void flatMap(Tuple4<Long, String, Float, Float> tweet,
                        Collector<Tuple5<Long, String, Float, Float, String>> out) throws Exception {
        String score;
        if (tweet.f2 >= tweet.f3)
            score = "positive";
        else
            score = "negative";
        out.collect(new Tuple5<>(
                tweet.f0,
                tweet.f1,
                tweet.f2,
                tweet.f3,
                score));
    }
}
```




```
result.print();

streamSource.writeAsText("file:///home/veith/erad2016-streamprocessing/results/teste-flink");
```

Inicialização do Ambiente Flink

```
env.execute("Twitter Streaming Example");
```

Referências

[Abb13] Abbasoğlu, Mehmet Ali, Buğra Gedik, and Hakan Ferhatosmanoğlu. "Aggregate profile clustering for telco analytics." Proceedings of the VLDB Endowment 6.12 (2013): 1234-1237.

[Aki13] Akidau, T., Balikov, A., Bekiroglu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P., and Whittle, S. (2013). Millwheel: fault-tolerant stream processing at internet scale. Proceedings of the VLDB Endowment, 6(11):1033–1044.

[Alv13] Alvanaki, Foteini, and Sebastian Michel. "Scalable, continuous tracking of tag co-occurrences between short sets using (almost) disjoint tag partitions." Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks. ACM, 2013.

[Ani13] Aniello, Leonardo, Roberto Baldoni, and Leonardo Querzoni. "Adaptive online scheduling in storm." Proceedings of the 7th ACM international conference on Distributed event-based systems. ACM, 2013.

[Ara04] Arasu, Arvind, et al. "Linear road: a stream data management benchmark." Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004.

[Bel12] Bellavista, Paolo, Antonio Corradi, and Andrea Reale. "Design and Implementation of a Scalable and QoS-aware Stream Processing Framework: The Quasit Prototype." Green Computing and Communications (GreenCom), 2012 IEEE International Conference on. IEEE, 2012.

[Bou12] Bouillet, Eric, et al. "Processing 6 billion CDRs/day: from research to production (experience report)." Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. ACM, 2012.

[Cas13] Castro Fernandez, R., Migliavacca, M., Kalyvianaki, E., and Pietzuch, P. (2013). Integrating scale out and fault tolerance in stream processing using operator state management. In Proceedings of the 2013 international conference on Management of data, pages 725–736. ACM.

[Cha09] Chakravarthy, S. and Jiang, Q. (2009). Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing. Springer Publishing Company, Incorporated, 1st edition.

[Cha11] Chandramouli, Badrish, et al. "StreamRec: a real-time recommender system."Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011.

[Cha13] Chardonnens, T., Cudre-Mauroux, P., Grund, M., & Perroud, B. (2013, October). Big data analytics on high Velocity streams: A case study. In Big Data, 2013 IEEE International Conference on (pp. 784-787). IEEE.

[Cha05] Chaudhry, N. A., Shaw, K., and Abdelguerfi, M., editors (2005). Stream Data Management, volume 30 of Advances in Database Systems. Springer.

[Day13] Dayarathna, M. and Suzumura, T. (2013). A performance analysis of system s, s4, and esper via two level benchmarking. In Quantitative Evaluation of Systems, pages 225–240. Springer.

[Gir14] Girtelschmid, Sylva, et al. "On the application of Big Data in future large scale intelligent Smart City installations." International Journal of Pervasive Computing and Communications 10.2 (2014): 4-4.

[Gul12] Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M., Soriente, C., and Valduriez, P. (2012). Streamcloud: An elastic and scalable data streaming system. Parallel and Distributed Systems, IEEE Transactions on, 23(12):2351–2365.

[Lin11] Lin, Liwei, Xiaohui Yu, and Nick Koudas. "Pollux: Towards scalable distributed real-time search on microblogs." Proceedings of the 16th International Conference on Extending Database Technology. ACM, 2013.

[Loh11] Lohrmann, Björn, and Odej Kao. "Processing smart meter data streams in the cloud." Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on. IEEE, 2011.

[Men08] Mendes, M., Bizarro, P., and Marques, P. (2008). A framework for performance evaluation of complex event processing systems. In Proceedings of the second international conference on Distributed event-based systems, pages 313–316. ACM.

[Men09] Mendes, M. R., Bizarro, P., and Marques, P. (2009). A performance study of event processing systems. In Performance Evaluation and Benchmarking, pages 221–236. Springer.

[Nab14] Nabi, Zubair, Eric Bouillet, Andrew Bainbridge, and Chris Thomas (2014). Of Streams and Storms: A Direct Comparison of IBM InfoSphere Streams and Apache Storm in a Real World Use Case – Email Processing. IBM.

[Pan13] Pan, Lujia, et al. "NIM: Scalable Distributed Stream Process System on Mobile Network Data." Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on. IEEE, 2013.

[Qia13] Qian, Z., He, Y., Su, C., Wu, Z., Zhu, H., Zhang, T., Zhou, L., Yu, Y., and Zhang, Z. (2013). Timestream: Reliable stream computation in the cloud. In Proceedings of the 8th ACM European Conference on Computer Systems, pages 1–14. ACM.

[Rab12] Rabl, Tilmann, et al. "Solving big data challenges for enterprise application performance management." Proceedings of the VLDB Endowment 5.12 (2012): 1724-1735.

[Ram11] de Souza Ramos, T. L. A., Oliveira, R. S., de Carvalho, A. P., Ferreira, R. A. C., and Meira, W. (2011). Watershed: A high performance distributed stream processing system. In Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on, pages 191–198. IEEE.

[Sch13] Scharrenbach, T., Urbani, J., Margara, A., Della Valle, E., and Bernstein, A. (2013). Seven commandments for benchmarking semantic flow processing systems. In The Semantic Web: Semantics and Big Data, pages 305–319. Springer.

[Sim11] Simmhan, Yogesh, et al. "Adaptive rate stream processing for smart grid applications on clouds." Proceedings of the 2nd international workshop on Scientific cloud computing. ACM, 2011.

[Sim13] Simoncelli, Davide, et al. "Stream-monitoring with BlockMon: convergence of network measurements and data analytics platforms." ACM SIGCOMM Computer Communication Review 43.1 (2013): 29-36.

[Smi13] Smit, M., Simmons, B., and Litoiu, M. (2013). Distributed, application-level monitoring for heterogeneous clouds using stream processing. Future Generation Computer Systems, 29(8):2103–2114.

[Sak10] Sakaki, Takeshi, Okazaki, Makoto, Matsuo, Yutaka: Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors, Proceedings of the 19th International Conference on World Wide Web, ACM, 851–860, 2010

[Tho11] Thomas, Kurt, et al. "Design and evaluation of a real-time url spam filtering service." Security and Privacy (SP), 2011 IEEE Symposium on. IEEE, 2011.

[Val12] Vallés, Mariano. "An Analysis of a Checkpointing Mechanism for a Distributed Stream Processing System." (2012). Master thesis, Universitat Politecnica de Catalunya – BarcelonaTech.

[Zah12] Zaharia, M., Das, T., Li, H., Shenker, S., and Stoica, I. (2012). Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing, pages 10–10. USENIX Association.

[Zha12] Zhang, Y., Duc, P. M., Corcho, O., & Calbimonte, J. P. (2012). SRBench: a streaming RDF/SPARQL benchmark. In The Semantic Web–ISWC 2012 (pp. 641-657). Springer Berlin Heidelberg.

[Ram11] de Souza Ramos, T. L. A., Oliveira, R. S., de Carvalho, A. P., Ferreira, R. A. C., and Meira, W. (2011). Watershed: A high performance distributed stream processing system. In Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on, pages 191–198. IEEE.

[Hei14] Heinze, Thomas, et al. "Tutorial: Cloud-based Data Stream Processing." (2014).

[Art14] Artikis, Alexander, Matthias Weidlich, Francois Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann et al. "Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management." In EDBT, pp. 712-723. 2014.

[Bou12] Bouillet, Eric, et al. "Processing 6 billion CDRs/day: from research to production (experience report)." Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. ACM, 2012.

[Lak08] Lakshmanan, G. T., LI, Y., and Strom, R. Placement strategies for internet-scale data stream systems. Internet Computing, IEEE 12, 6 (2008), 50–60.

[Sim11] Simmhan, Yogesh, et al. "An informatics approach to demand response optimization in smart grids." NATURAL GAS 31 (2011): 60.

[Mar15] Marz, Nathan, Warren, James: , Big Data: Principles and best practices of scalable realtime data systems, 1st edition, Manning Publications, 328, May 2015

[Kir15] Kiran, M., Murphy, P., Monga, I., Dugan, J., Baveja, S. S.: Lambda architecture for cost-effective batch and speed big data processing, Big Data (Big Data), 2015 IEEE International Conference on, IEEE Computer Society, 2785–2792, Oct 2015

[Cae16] Wu, Caesar, Buyya, Rajkumar, Ramamohanarao, Kotagiri: Big Data Analytics: Machine Learning plus Cloud Computing, Big Data: Principles and Paradigms, Morgan Kaufmann, 1–27, Eds: Buyya, Rajkumar, Calheiros, Rodrigo N., Dastjerdi, Amir Vahid, January 2016

Agradecimentos

Trabalhando com Big Data em Tempo Real

Maycon Viana Bordin
Julio C. S. dos Anjos
Raffael Bottoli Schemmer
Cláudio Geyer

Instituto de Informática
Universidade Federal do Rio Grande do Sul