Trident

Высокоуровневая абстракция поверх Storm

Зачем нужен Trident ?

- Выше уровень абстракции по сравнению со storm
  - Потоки данных и функции
  - DSL (Domain specific language) для создания топологий
- Транзакционные топологии
- Сохранения состояния во внешних системах

Идея Trident

- Граф потоков данных задается с помощью dsl

```
Stream wordCounts = topology.newStream("spout1", spout)
        .parallelismHint(1).shuffle()
        .each(new Fields("line"), new SplitFunction(), new Fields("word")).shuffle()
        .each(new Fields("word"), new ToUpperFunction(), new Fields("upper_word"))
        .each(new Fields("upper_word"), new PrintFilter())
        .partitionBy(new Fields("upper_word"))
        .partitionAggregate(new Fields("upper_word"), new WordAggregator(), new Fields("counts"))
        .parallelismHint(10)
        .each(new Fields("counts"), new PartitionCountPrinter(), new Fields()
        );
```

- Потоки данных trident компилируются в обычные топологии storm

Запуск локальной топологии Trident

```
TridentTopology topology = new TridentTopology();
...
Config conf = new Config();
conf.put(SimplePollSpout.POLL_DIR, args[0]);
conf.put(SimplePollSpout.PROCESSED_DIR, args[1]);
LocalCluster cluster = new LocalCluster();
cluster.submitTopology("poll", conf, topology.build());
```

Trident Spout

- В Trident spout должен уметь генерировать tuple пакетами
- Базовый интерфейс Spout :

```
public interface IBatchSpout extends ITridentDataSource {
    void open(Map map, TopologyContext topologyContext);
    void emitBatch(long batchId, TridentCollector tridentCollector);
    void ack(long batchId);
    void close();
    Map getComponentConfiguration();
    Fields getOutputFields();
}
```

Команды DSL
.each

- Фильтрует поток данных

  Stream.each(Fields inputFields, Filter filter)

- Применяет функцию к потоку данных и генерирует дополнительные поля

  Stream.each(Fields inputFields, Function function, Fields functionFields)

- Примеры

```
stream.each(
  new Fields("line"),
  new SplitFunction(),
  new Fields("word")
)  ;
stream.each(new Fields("upper_word"), new PrintFilter())
```

Функции в .each

- Наследуем класс от BaseFunction
- Перегружаем методы prepare и execute

```
public static class SplitFunction extends BaseFunction {
        TridentOperationContext context;
        @Override
        public void prepare(Map conf, TridentOperationContext context) {
            this.context = context;
        }
        @Override
        public void execute(TridentTuple objects, TridentCollector tridentCollector) {
            String s = (String) objects.get(0);
            String[] list = StringTools.splitAndRemoveNonSymbols(s);
            for (String word : list) {
                tridentCollector.emit(new Values(word));
            }
        }
    }
```

-

Фильтры в .each

- Наследуем класс от BaseFilter

```
public static class RegexpFilter extends BaseFilter {
    private String regexp;

    public RegexpFilter() {
    }

    public RegexpFilter(String regexp) {
        this.regexp = regexp;
    }

    @Override
    public boolean isKeep(TridentTuple objects) {
        String value = objects.getString(0);
        return value.matches(regexp);
    }
}
```

Полное агрегирование .aggregate

- Применяет функцию аггрегирования ко всему потоку данных в рамках одного пакета данных - batch
- Результат функции аггрегирования поступает в выходной поток данных
- В случае использования CombinerAggregator производится оптимизация аггрегации -
  - Trident создает дерево bolt которые паралеллельно производят аггрегацию
- В случае использования ReducerAggregator или Aggregator –
  - Trident создает один болт который производит агрегацию всего потока
- Пример :

```
Stream wordCounts = topology.newStream("spout1", spout)
        .parallelismHint(1).shuffle()
        .each(new Fields("line"), new SplitFunction(), new Fields("word"))..shuffle()
        .each(new Fields("word"), new ToUpperFunction(), new Fields("upper_word"))
        .each(new Fields("upper_word"), new PrintFilter())
        .aggregate(new Fields("upper_word"), new WordAggregator(), new Fields("counts"))
        .parallelismHint(10)
        .each(new Fields("counts"), new PartitionCountPrinter(), new Fields()
        );
```

Пример CombinerAggregator

```
public static class WordCombiner implements CombinerAggregator<Map<String, Integer>> {
    public Map<String, Integer> init(TridentTuple objects) {
        HashMap<String, Integer> result = Maps.newHashMap();
        String word = objects.getString(0);
        result.put(word, 1);
        return result;
    }
    public Map<String, Integer> combine(Map<String, Integer> map1, Map<String, Integer> map2) {
        Map<String, Integer> result = Maps.newHashMap();
        result.putAll(map1);
        for (String key : map2.keySet()) {
            Integer value = map2.get(key);
            if(value!=null) {
                Integer existValue = result.containsKey(key) ? result.get(key) : 0;
                result.put(key, existValue+value);
            }
        }
        return result;
    }
    public Map<String, Integer> zero() {
        return Maps.newHashMap();
    }
}
```

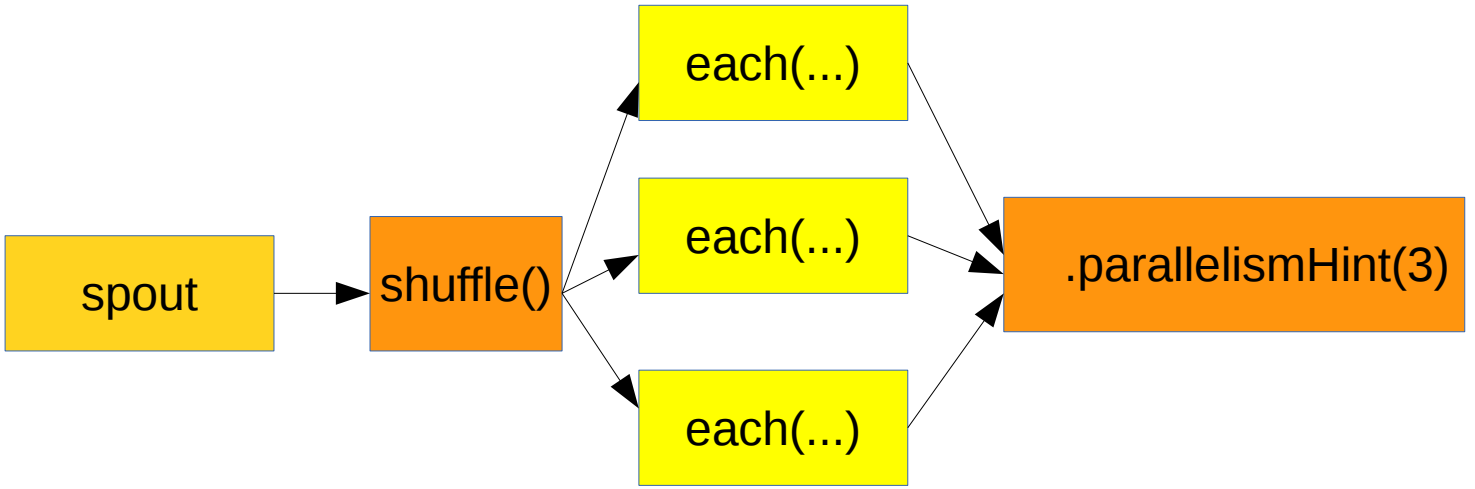
Пример BaseAggregator

```
public static class WordAggregator extends BaseAggregator<Map<String, Integer>> {
    @Override
    public Map<String, Integer> init(Object batchId, TridentCollector tridentCollector) {
        return new HashMap<String, Integer>();
    }
    @Override
    public void aggregate(Map<String, Integer> stringIntegerMap,
                          TridentTuple objects, TridentCollector tridentCollector) {
        String word = objects.getString(0);
        Integer count = stringIntegerMap.get(word);
        count = count == null ? 1 : count + 1;
        stringIntegerMap.put(word, count);
    }
    @Override
    public void complete(Map<String, Integer> stringIntegerMap, TridentCollector tridentCollector) {
        tridentCollector.emit(new Values(stringIntegerMap));
    }
}
```

Партиционирование

- .shuffle() – осуществляет случайное партиционирование
- .partitionBy(Fields fields) осуществляет партиционирование гарантируя одинаковому набору значений заданных полей попадание на один узел
- .parallelismHint(int hint) задает колчество параллельных потоков от этого места до предыдущей операции партиционирования

Примеры партиционирования



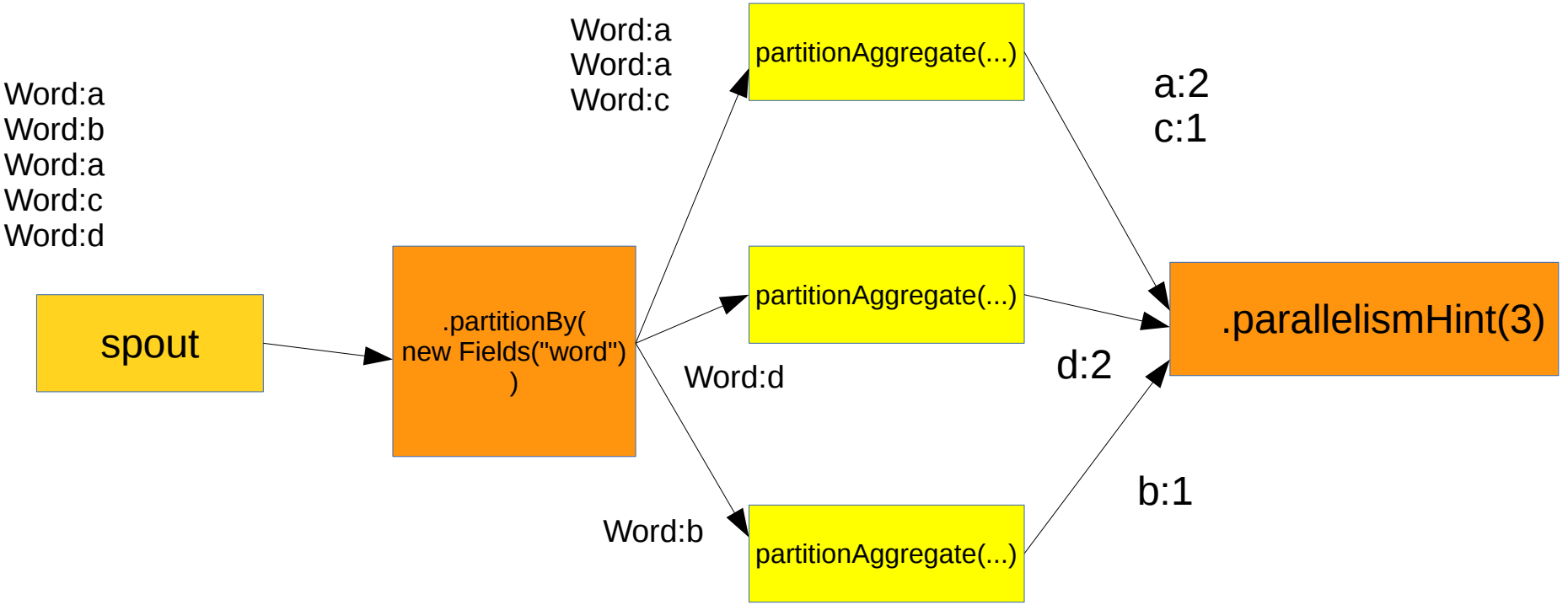

```
topology.newStream("spout1", spout)
  .shuffle()
  .each(new Fields("line"), new SplitFunction(), new Fields("word"))
  .parallelismHint(3)
```

Агрегирование в рамках partition .partitionAggregate

- Аналогично полному аггрегированию, но производится параллельно и независимо в каждом partition
- Пример :

```
Stream wordCounts = topology.newStream("spout1", spout)
        .parallelismHint(1)
        .shuffle()
        .each(new Fields("line"), new SplitFunction(), new Fields("word")).parallelismHint(10)
        .shuffle()
        .each(new Fields("word"), new ToUpperFunction(), new Fields("upper_word"))
        .each(new Fields("upper_word"), new PrintFilter())
        .partitionBy(new Fields("upper_word"))
        .partitionAggregate(new Fields("upper_word"),
                            new WordAggregator(), new Fields("counts"))
        .parallelismHint(10)
        .each(new Fields("counts"), new PartitionCountPrinter(), new Fields()
        );
```

Примеры партиционирования



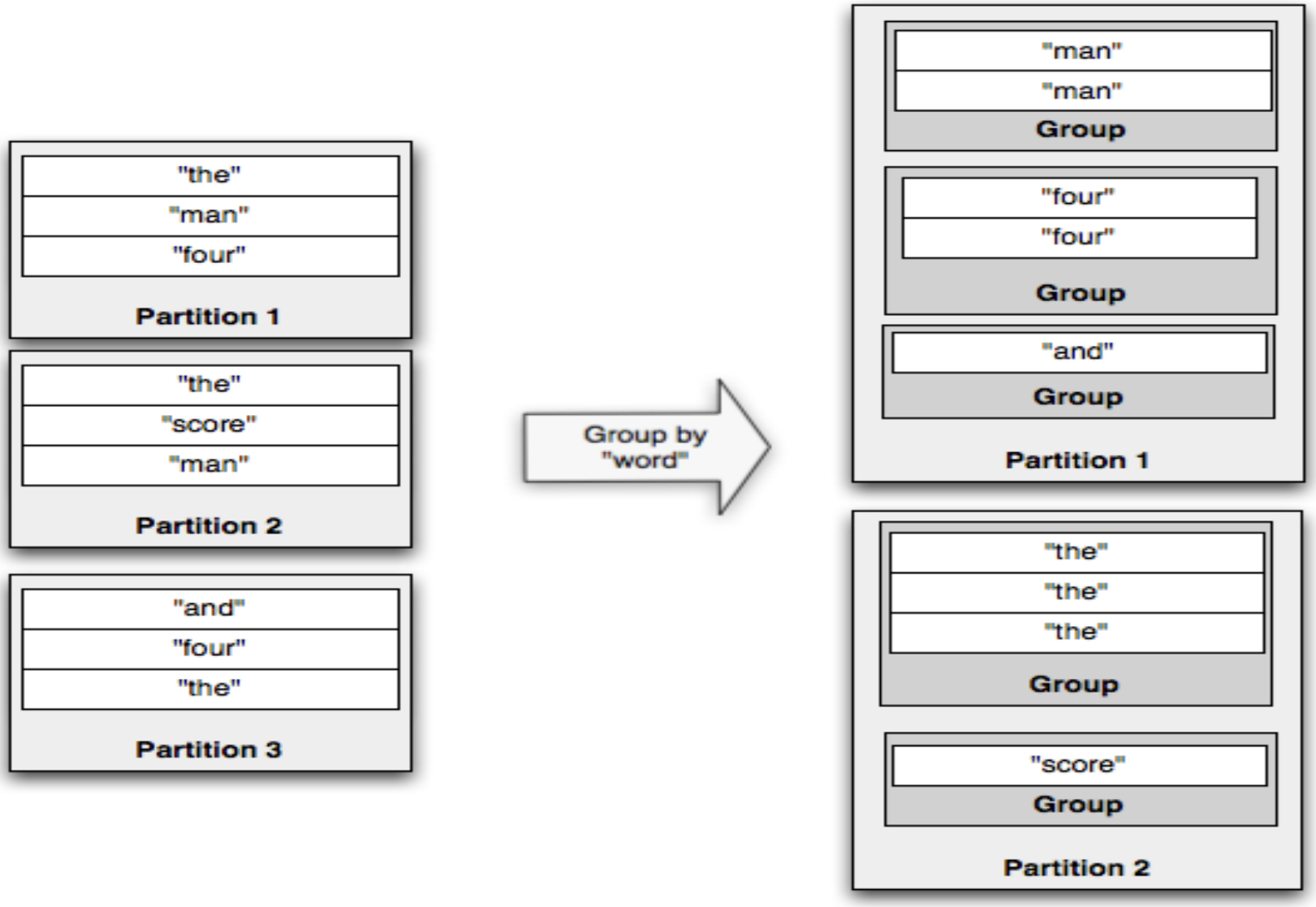

```
topology.newStream("spout1", spout)
  .partitionBy(new Fields("Word"))
  .partitionAggregate(new Fields("Word"), new WordAggregator(), new
Fields("counts"))
  .parallelismHint(3)
```

Группировка .groupBy

- Группировка отличается от партиционирования тем, что внутри partition сообщения дополнительно группируются по ключу
- Функция агрегирования будет вызываться не для всех сообщений в partition, а для каждого ключа отдельно

```
...
.groupBy(new Fields("upper_word"))
.aggregate(new Fields("upper_word"), new WordAggregator(), new Fields("counts"))
```

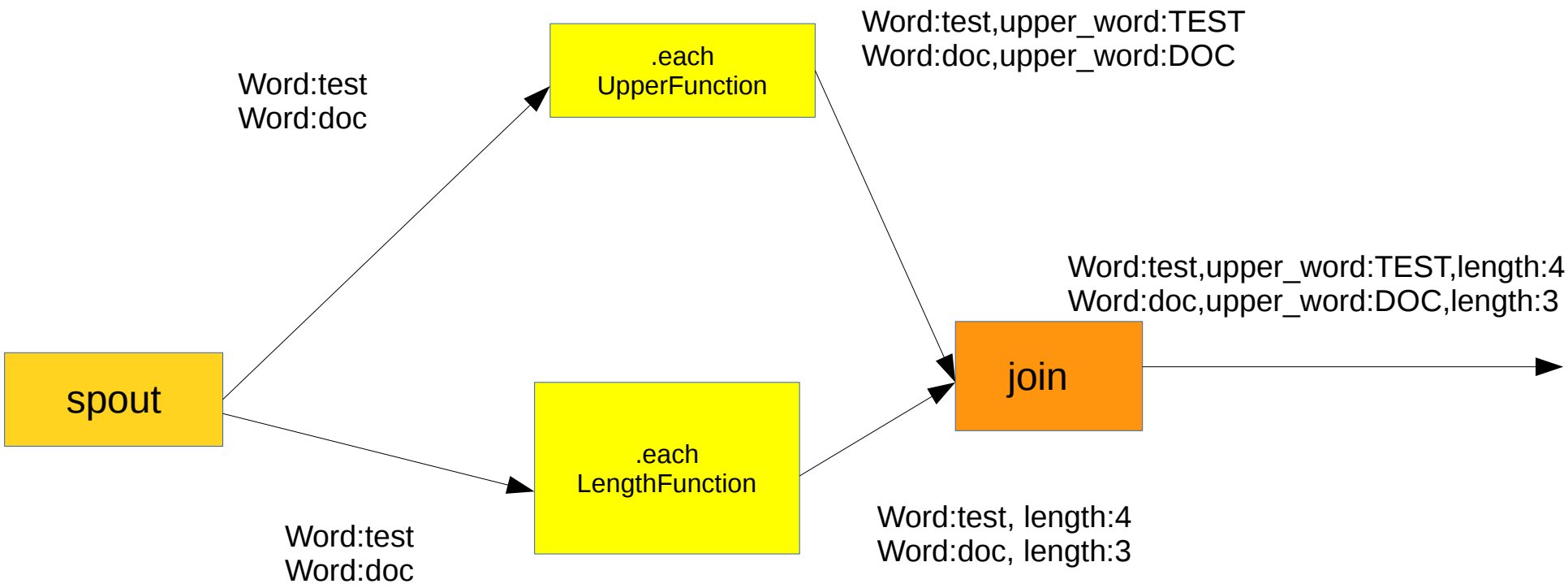
groupBy


"the"
"man"
"four"
Partition 1
"the"
"score"
"man"
Partition 2
"and"
"four"
"the"
Partition 3
Group by "word"
"man"
"man"
Group
"four"
"four"
Group
"and"
Group
Partition 1
"the"
"the"
"the"
Group
"score"
Group
Partition 2

Join

- Связывает два потока данных по ключу
- Join осуществляется только в рамках одного пакета данных – batch
- В случае если потоки данных не порождены одним spout осуществляется координация spout – join производится в рамках batch порожденных одним txid

Join


Word:test,upper_word:TEST
Word:doc,upper_word:DOC
.each
UpperFunction
Word:test
Word:doc
Word:test,upper_word:TEST,length:4
Word:doc,upper_word:DOC,length:3
spout
join
.each
LengthFunction
Word:test, length:4
Word:doc, length:3
Word:test
Word:doc

Пример Join

```
Stream stream1 = topology.newStream("spout1", spout)
        .parallelismHint(1)
        .shuffle()
        .each(new Fields("line"), new SplitFunction(), new Fields("word")).parallelismHint(10);
    Stream stream2 = stream1
        .shuffle()
        .each(new Fields("word"), new ToUpperFunction(), new Fields("upper_word"))
        .project(new Fields("word", "upper_word"));
    Stream stream3 = stream1
            .shuffle()
            .each(new Fields("word"), new SizeFunction(), new Fields("size"))
            .project(new Fields("word", "size"));
    Stream join = topology.join(stream2, new Fields("word"),
                    stream3, new Fields("word"),
                    new Fields("word","upper_word", "size")
                   )
        .each(new Fields("word","upper_word", "size"), new PrintAllFilter());
```