Introduction to

Cascading

What is Cascading?

What is Cascading?

- abstraction over MapReduce

What is Cascading?

- abstraction over MapReduce
- API for data-processing workflows

Why use Cascading?

Why use Cascading?

- MapReduce can be:

Why use Cascading?

- MapReduce can be:
  - the wrong level of granularity

Why use Cascading?

- MapReduce can be:
  - the wrong level of granularity
  - cumbersome to chain together

Why use Cascading?

Why use Cascading?

- Cascading helps create

Why use Cascading?

- Cascading helps create
  - higher-level data processing abstractions

Why use Cascading?

- Cascading helps create
  - higher-level data processing abstractions
  - sophisticated data pipelines

Why use Cascading?

- Cascading helps create
  - higher-level data processing abstractions
  - sophisticated data pipelines
  - reusable components

Credits

- Cascading written by Chris Wensel
- based on his users guide

http://bit.ly/cascading

```
package cascadingtutorial.wordcount;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      String inputPath  = args[0];
      String outputPath = args[1];

      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));

      wcPipe = new GroupBy(wcPipe, new Fields("word"));
      wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));

      Properties properties = new Properties();
      FlowConnector.setApplicationJarClass(properties, Main.class);

      Flow parsedLogFlow = new FlowConnector(properties)
        .connect(sourceTap, sinkTap, wcPipe);
      parsedLogFlow.start();
      parsedLogFlow.complete();
    }
  }
```

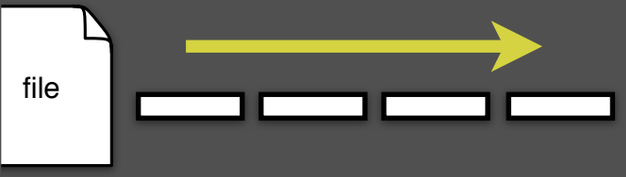
Data Model

Pipes

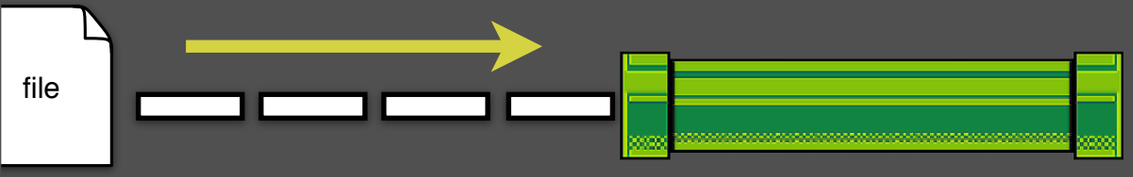
Filters

Data Model

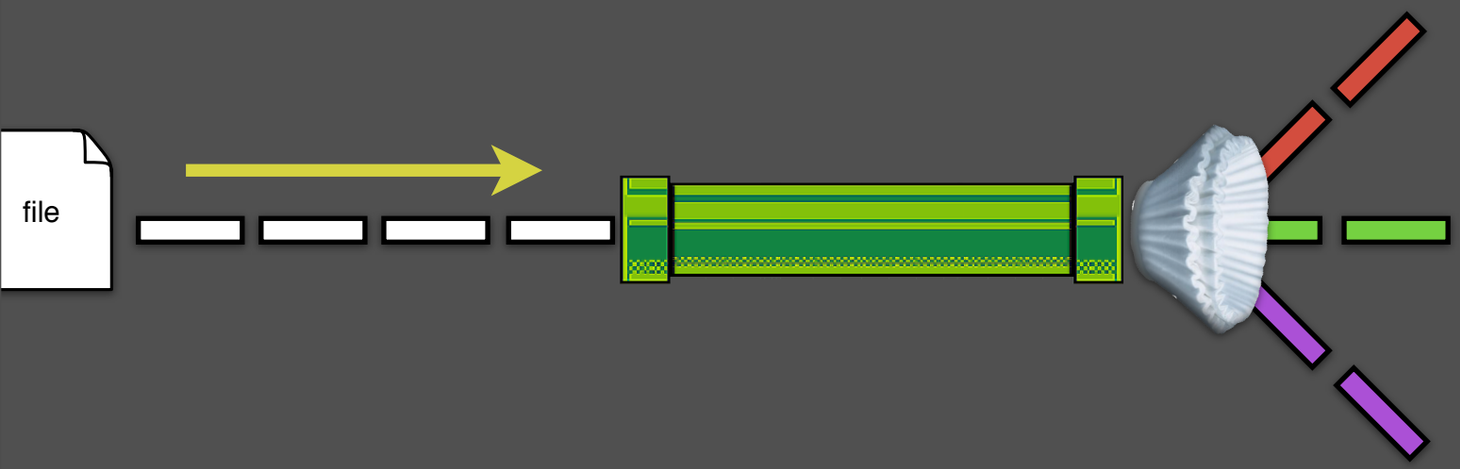
Data Model



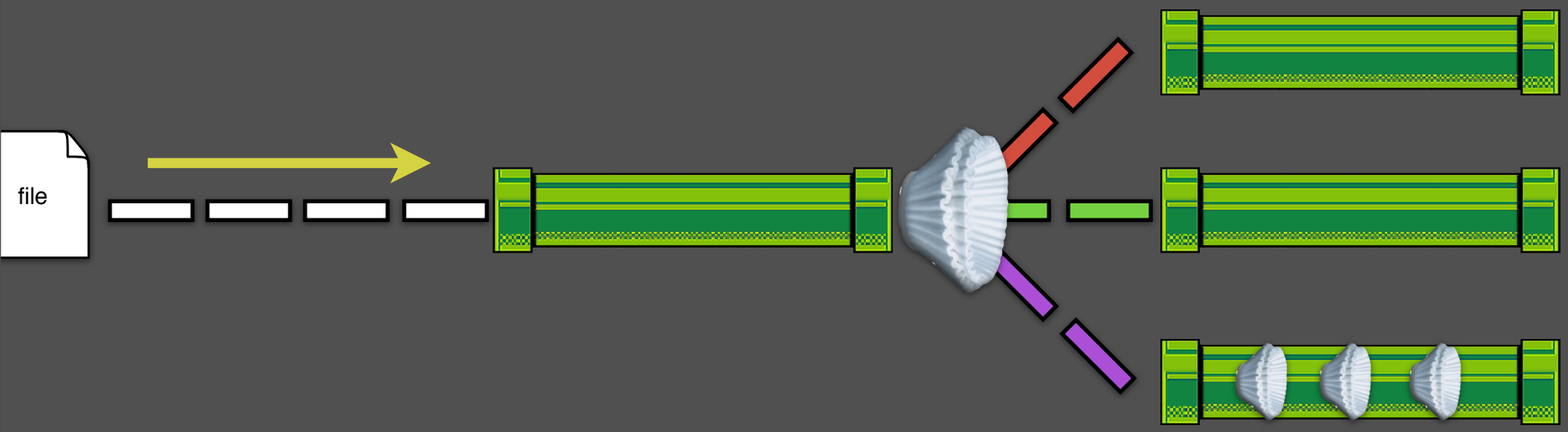
Data Model



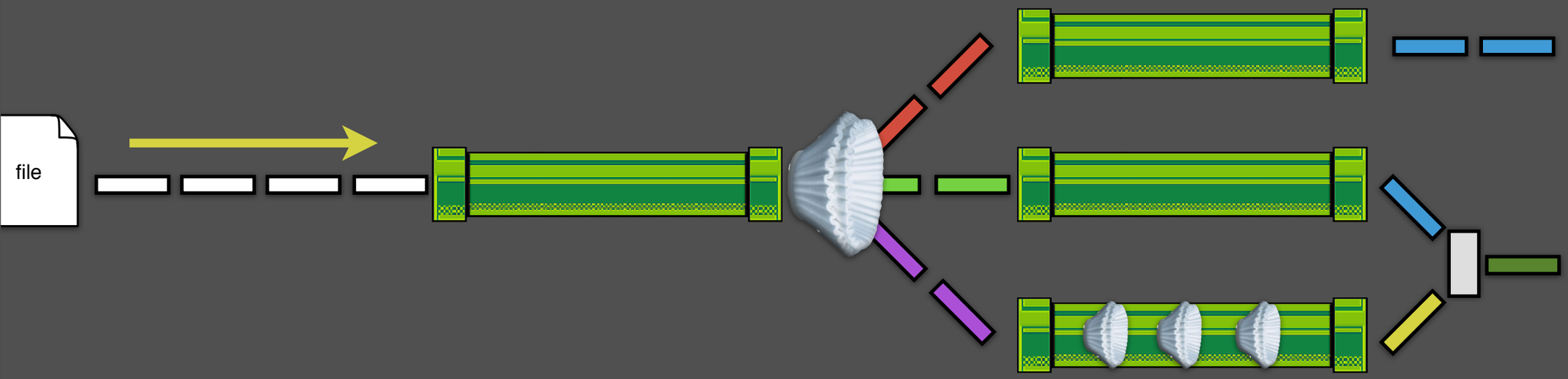
Data Model



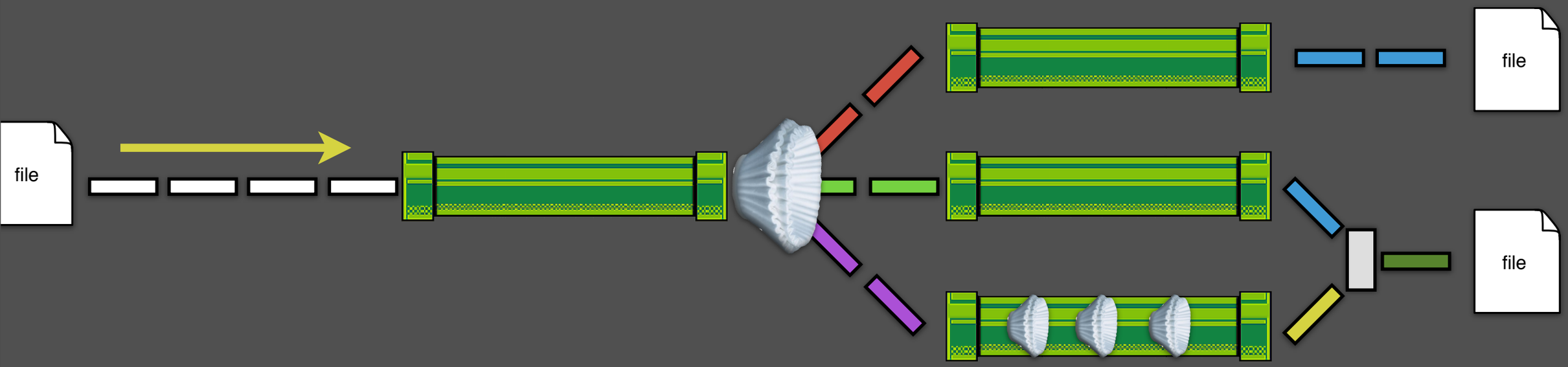
Data Model



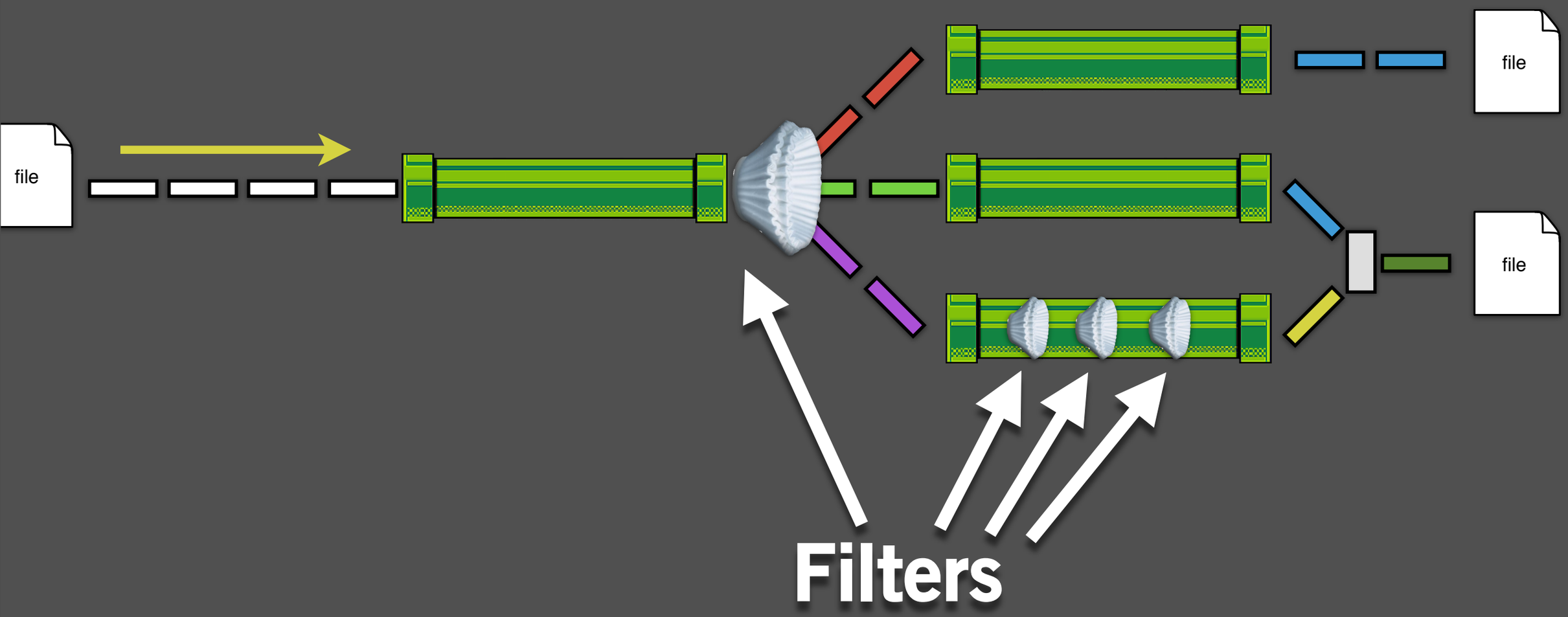
Data Model



Data Model



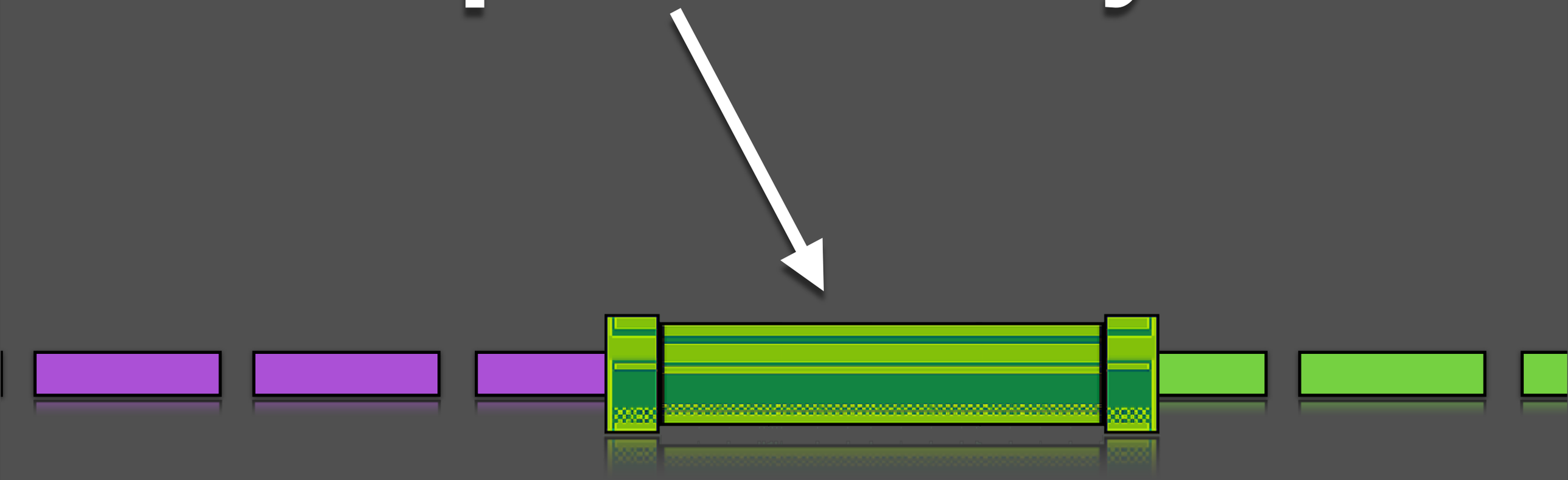
Data Model



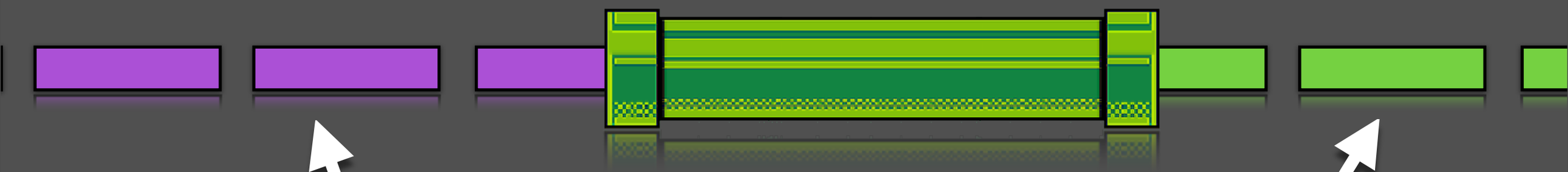
Tuple

123	vis-abc	tacos	125668-3278	1

Pipe Assembly



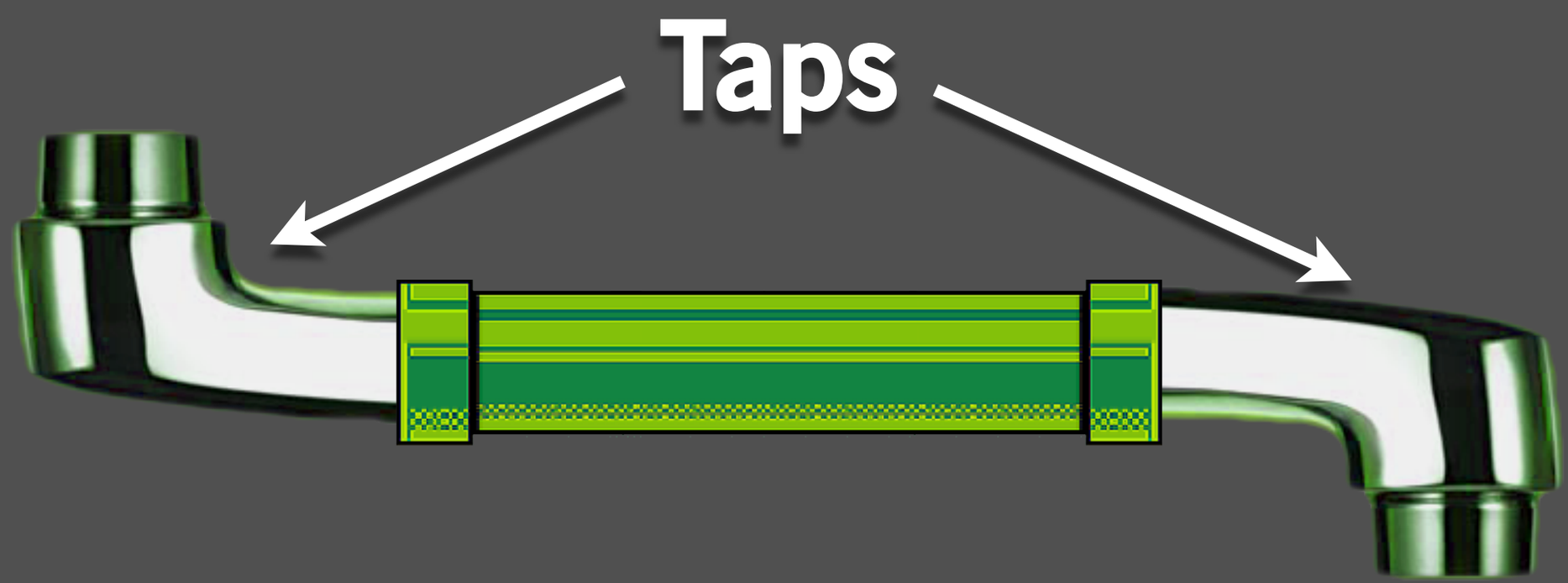
Pipe Assembly



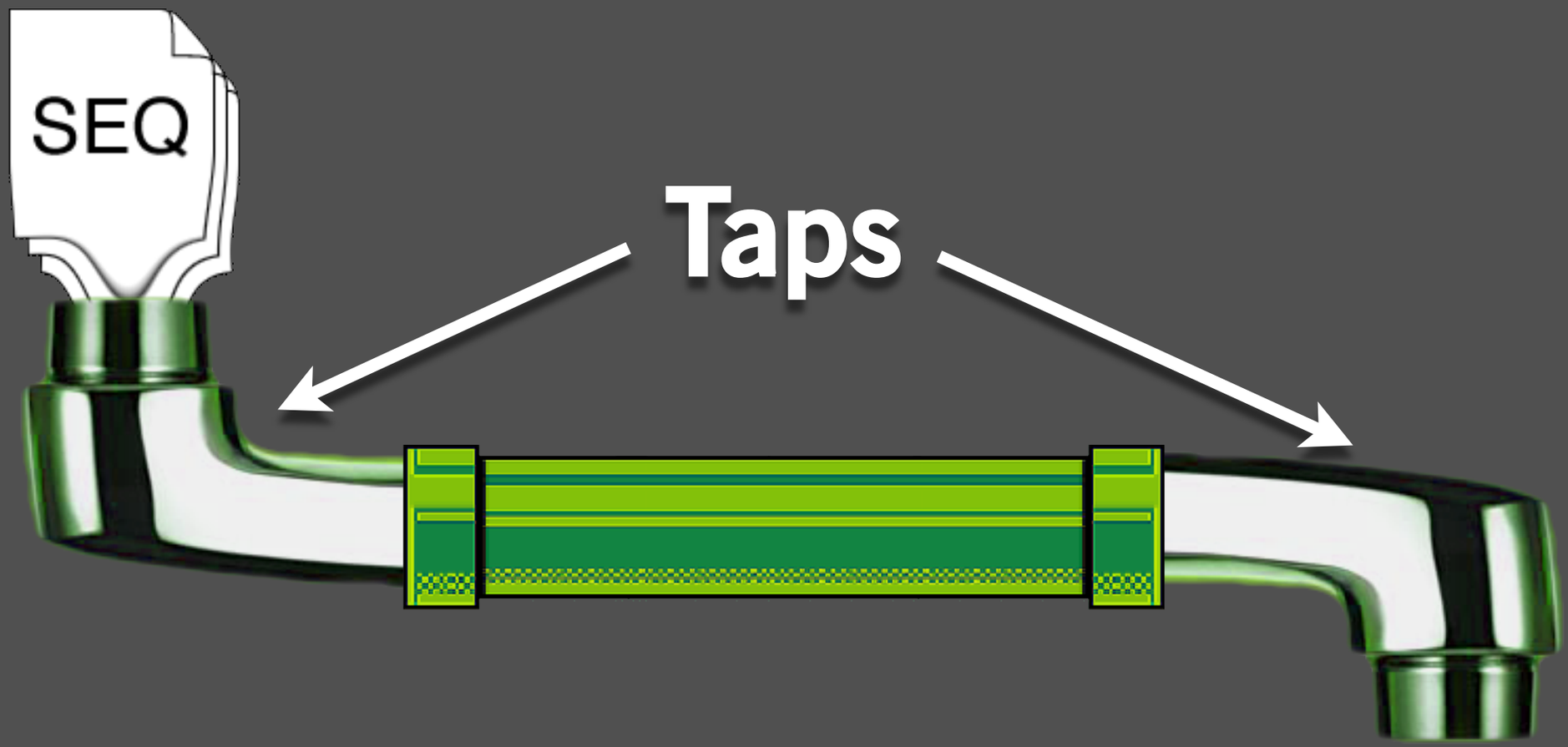
Tuple Stream

Taps: sources & sinks

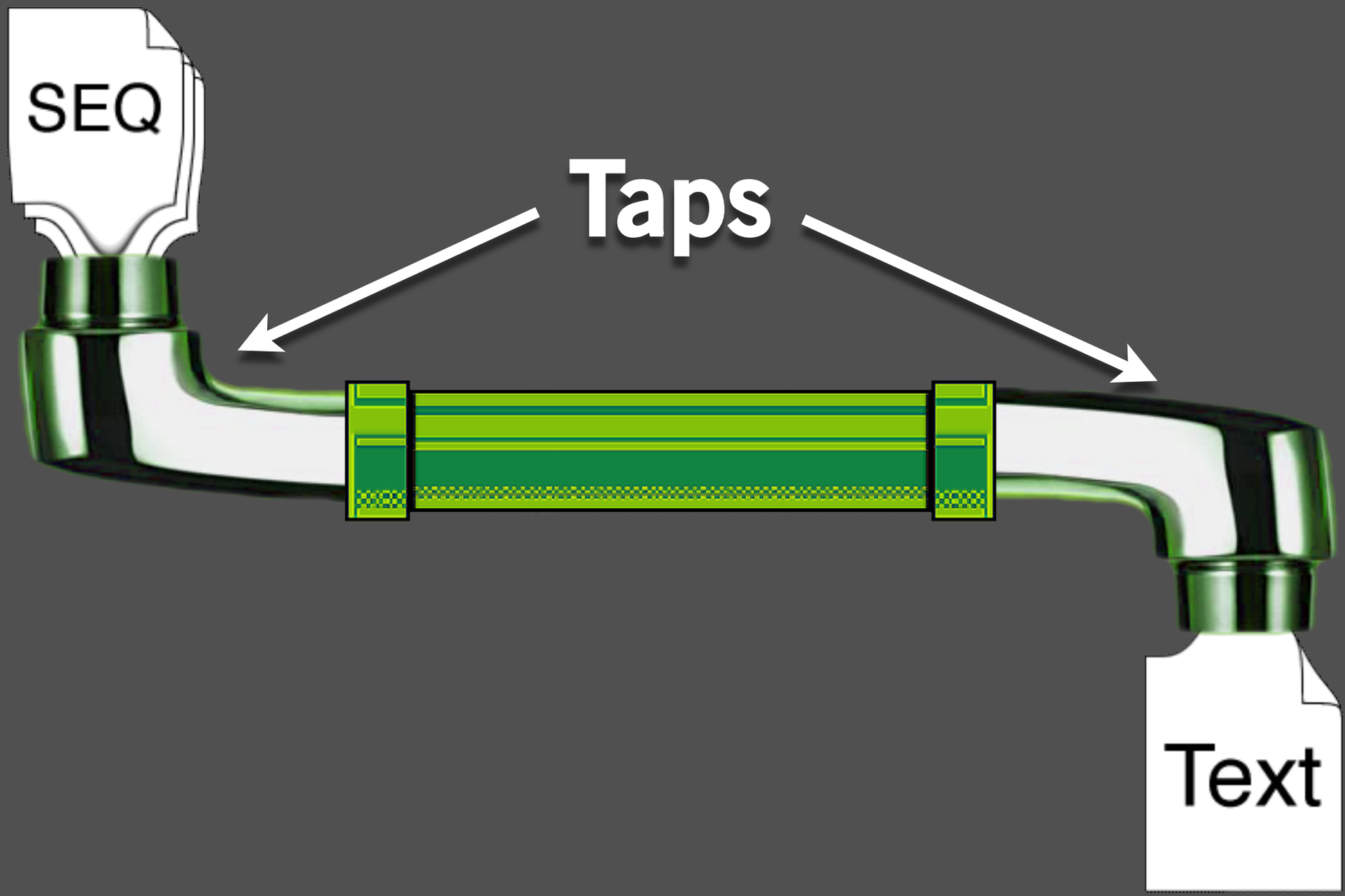
Taps: sources & sinks



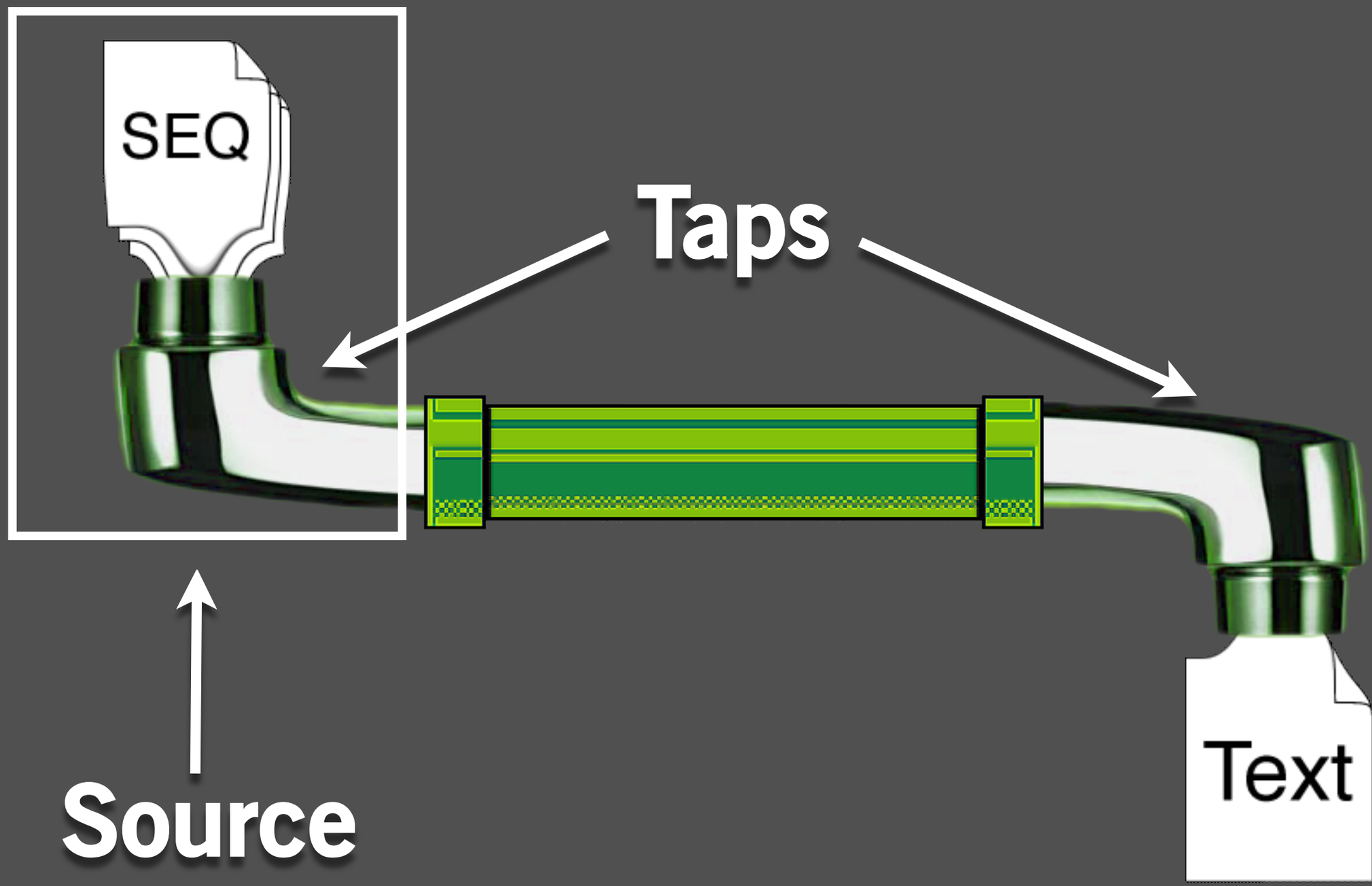
Taps: sources & sinks



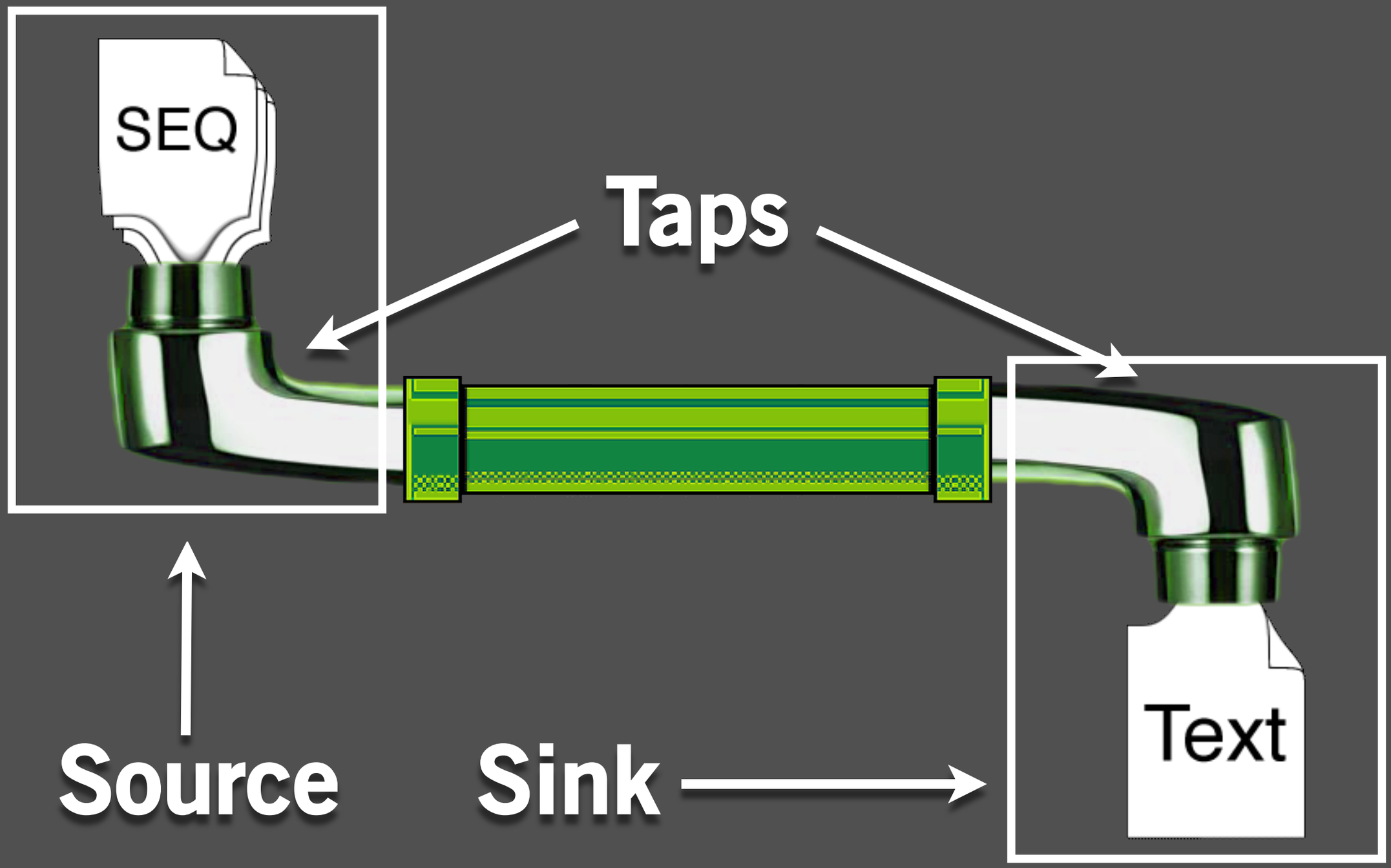
Taps: sources & sinks

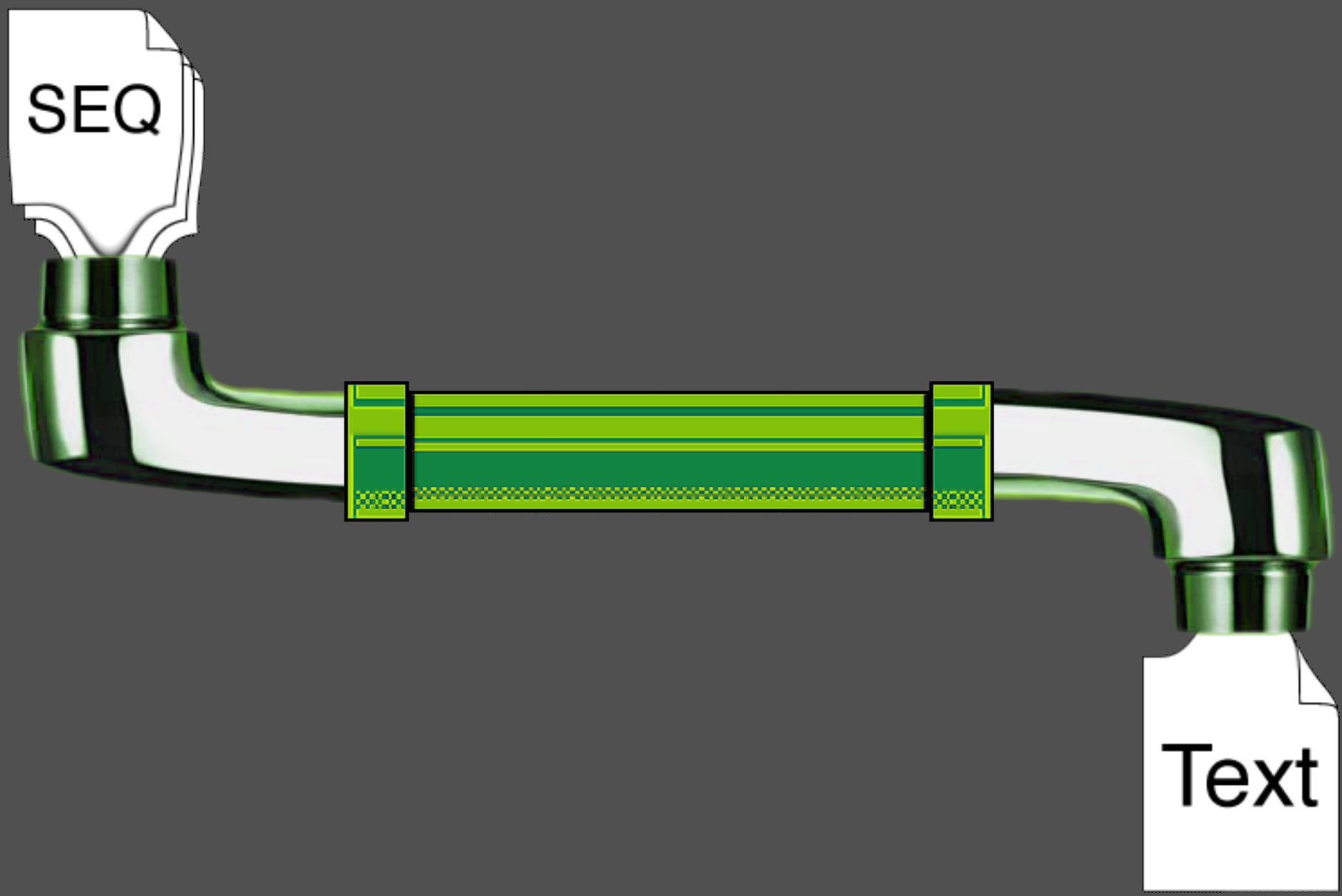


Taps: sources & sinks

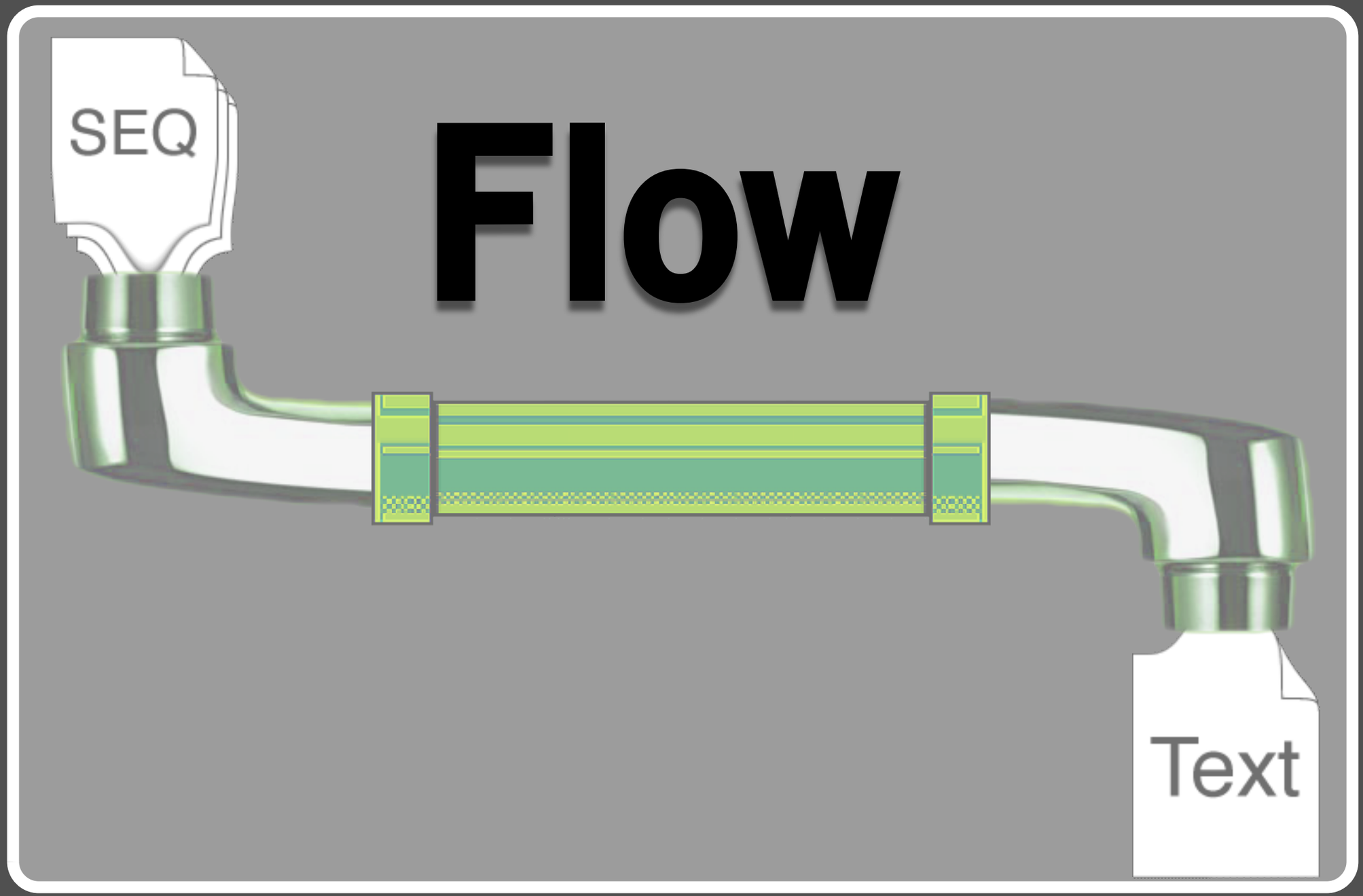


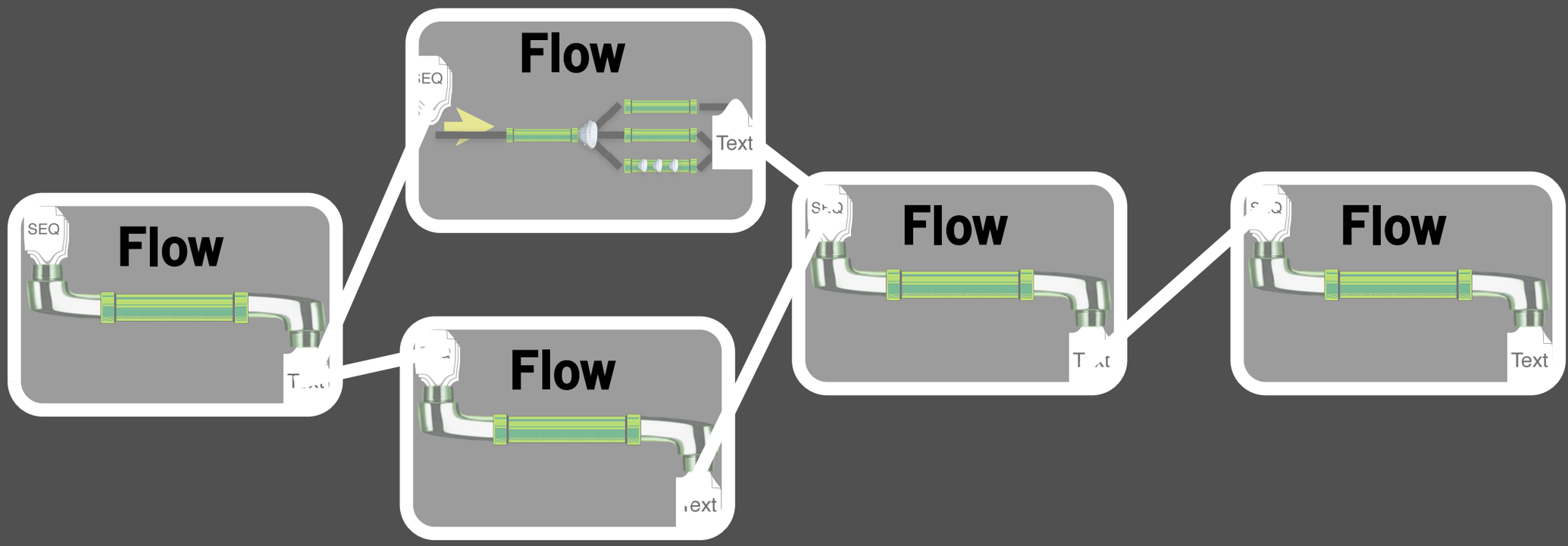
Taps: sources & sinks



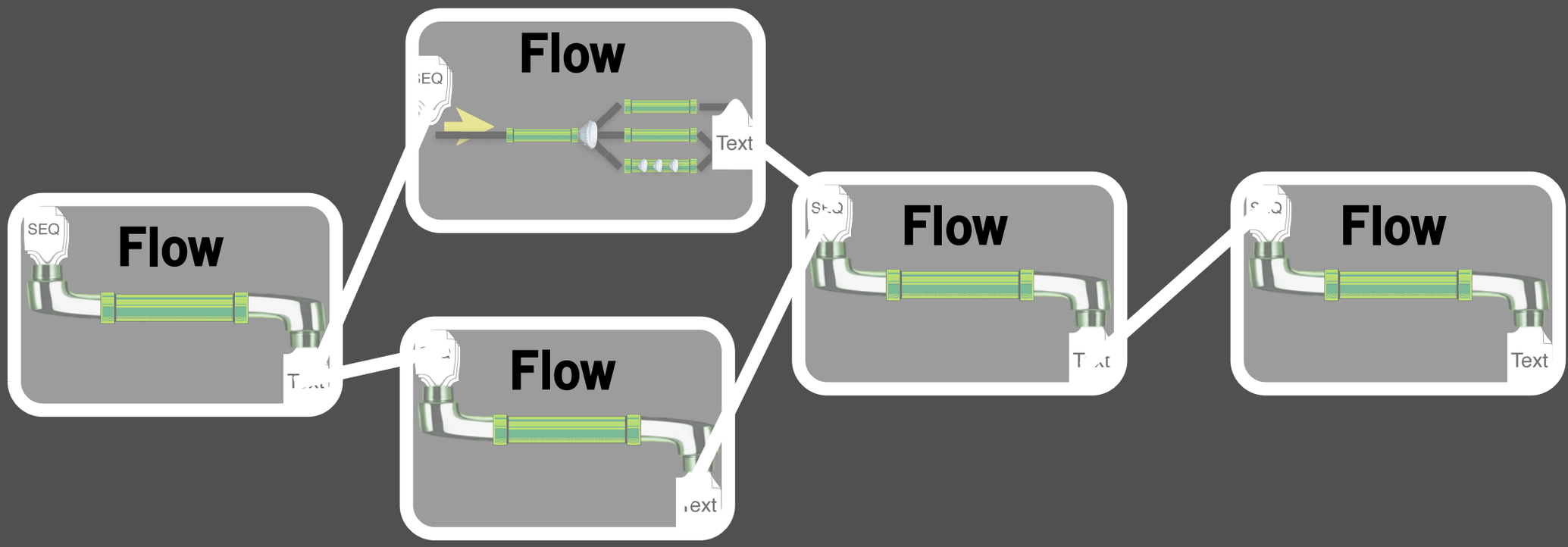

SEQ
Text

Pipe + Taps = Flow

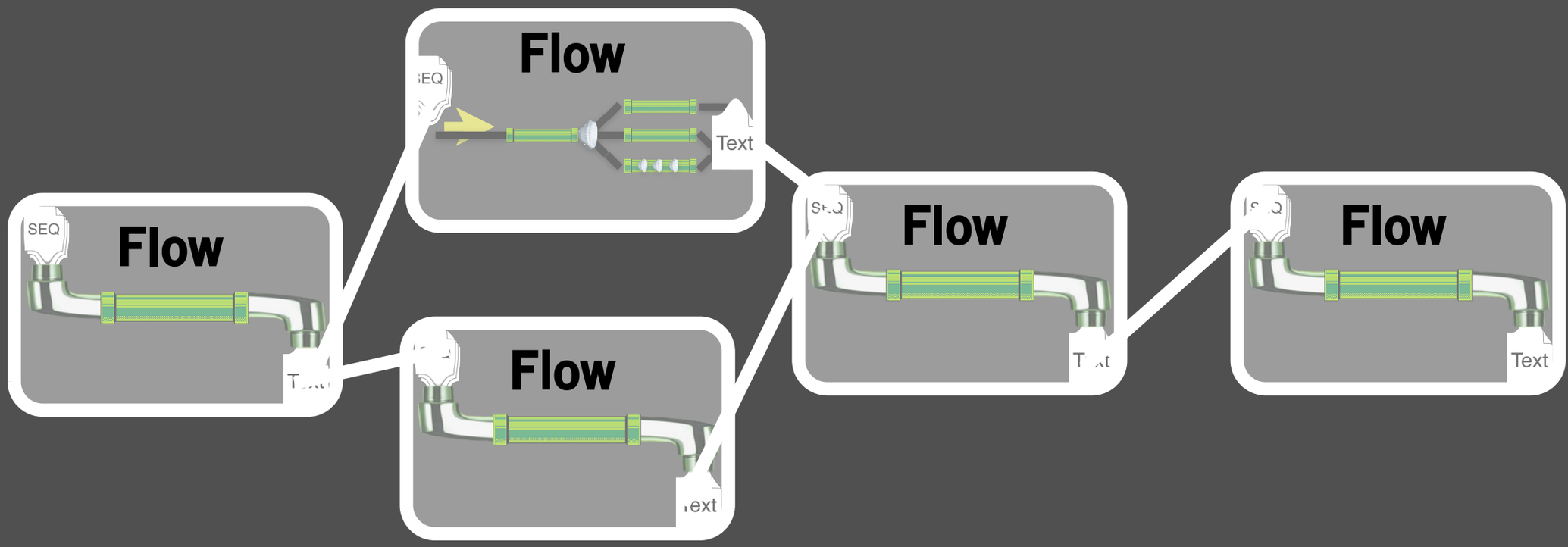



Flow
SEQ
Text
Flow
SEQ
Text
Flow
Text
Flow
SEQ
Text
Flow
Text

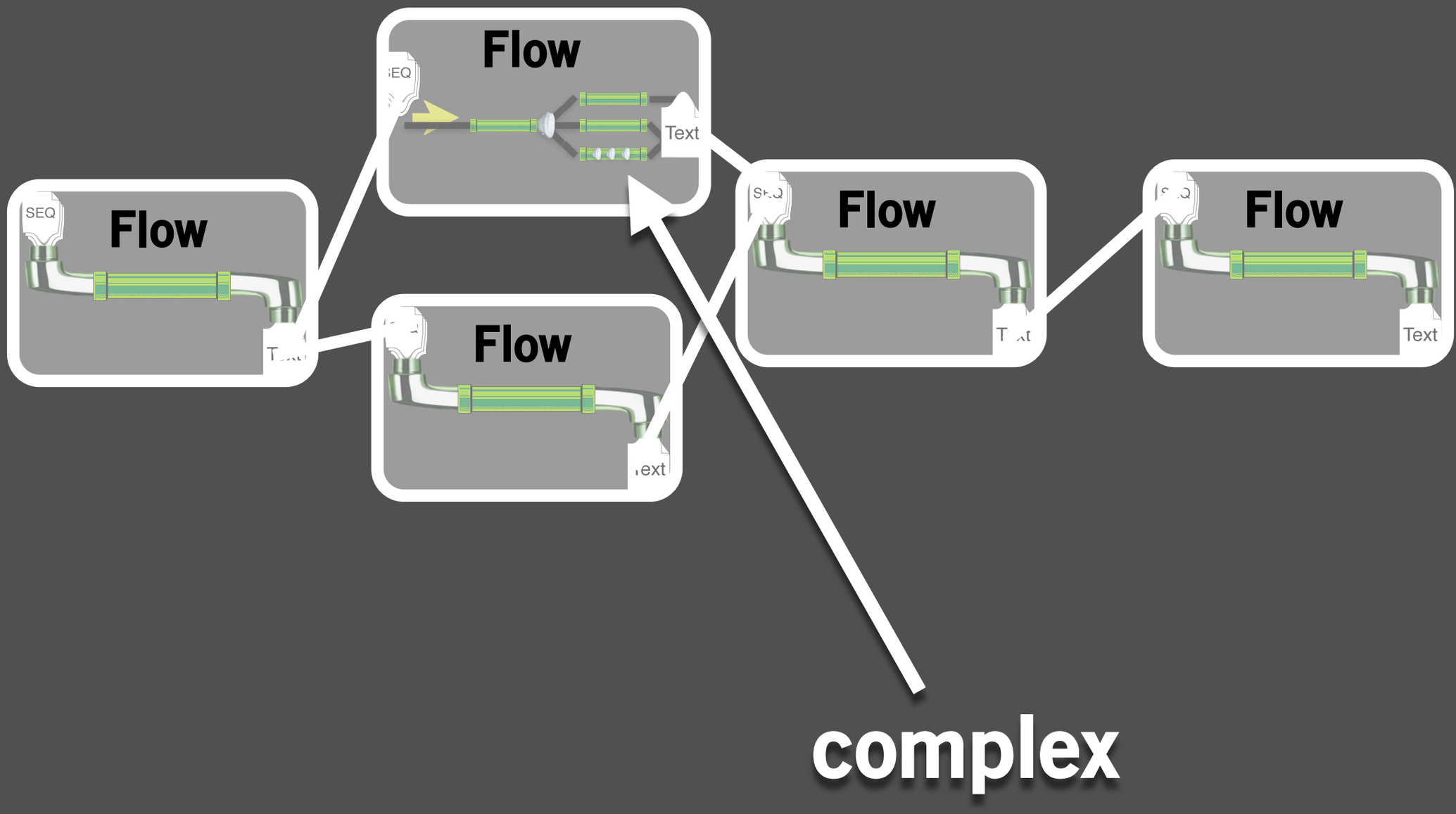
n Flows



n Flows = Cascade



n Flows = Cascade



Example

```
package cascadingtutorial.wordcount;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      String inputPath  = args[0];
      String outputPath = args[1];

      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));

      wcPipe = new GroupBy(wcPipe, new Fields("word"));
      wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));

      Properties properties = new Properties();
      FlowConnector.setApplicationJarClass(properties, Main.class);

      Flow parsedLogFlow = new FlowConnector(properties)
        .connect(sourceTap, sinkTap, wcPipe);
      parsedLogFlow.start();
      parsedLogFlow.complete();
    }
  }
```

```
package cascadingtutorial.wordcount;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      String inputPath  = args[0];
      String outputPath = args[1];

      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));

      wcPipe = new GroupBy(wcPipe, new Fields("word"));
      wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));

      Properties properties = new Properties();
      FlowConnector.setApplicationJarClass(properties, Main.class);

      Flow parsedLogFlow = new FlowConnector(properties)
        .connect(sourceTap, sinkTap, wcPipe);
      parsedLogFlow.start();
      parsedLogFlow.complete();
    }
  }
```

```
import cascading.tuple.Fields;
import java.util.Properties;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));
```

```
import cascading.tuple.Fields;
import java.util.Properties;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));
```

```
import cascading.tuple.Fields;
import java.util.Properties;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));
```

TextLine()

```
import cascading.tuple.Fields;
import java.util.Properties;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)    :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap = outputPath.matches( "^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath)  :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));
```

TextLine() SequenceFile()

```
import cascading.tuple.Fields;
import java.util.Properties;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));
```

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();
```

```
      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)    :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);
```

```
      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));

      wcPipe = new GroupBy(wcPipe, new Fields("word"));
      wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {
  public static void main( String[] args )
    {
    Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
    Scheme outputScheme = new TextLine();
```

hdfs://master0:54310/user/nmurray/data.txt

```
    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)    :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);
```

```
    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));

    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {
  public static void main( String[] args )
    {
    Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
    Scheme outputScheme = new TextLine();
```

hdfs://master0:54310/user/nmurray/data.txt

new Hfs()

```
    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)    :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);
```

```
    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));

    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {
  public static void main( String[] args )
    {
    Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
    Scheme outputScheme = new TextLine();
```

hdfs://master0:54310/user/nmurray/data.txt

new Hfs()

data/sources/obama-inaugural-address.txt

```
    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)    :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);
```

```
    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));

    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {
  public static void main( String[] args )
    {
    Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
    Scheme outputScheme = new TextLine();

    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)    :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);

    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));

    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

hdfs://master0:54310/user/nmurray/data.txt new Hfs()

data/sources/obama-inaugural-address.txt new Lfs()

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {
  public static void main( String[] args )
    {
    Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
    Scheme outputScheme = new TextLine();
```

hdfs://master0:54310/user/nmurray/data.txt new Hfs()

data/sources/obama-inaugural-address.txt new Lfs()

```
    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)    :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);
```

S3fs()

```
    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));

    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {
  public static void main( String[] args )
    {
    Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
    Scheme outputScheme = new TextLine();

    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)    :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);

    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));

    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

hdfs://master0:54310/user/nmurray/data.txt new Hfs()

data/sources/obama-inaugural-address.txt new Lfs()

S3fs()

GlobHfs()

```
/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();
```

```
      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)    :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);
```

```
      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));

      wcPipe = new GroupBy(wcPipe, new Fields("word"));
      wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
    String inputPath  = args[0];
    String outputPath = args[1];

    Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
      new Hfs(inputScheme, inputPath)   :
      new Lfs(inputScheme, inputPath);
    Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
      new Hfs(outputScheme, outputPath) :
      new Lfs(outputScheme, outputPath);
```

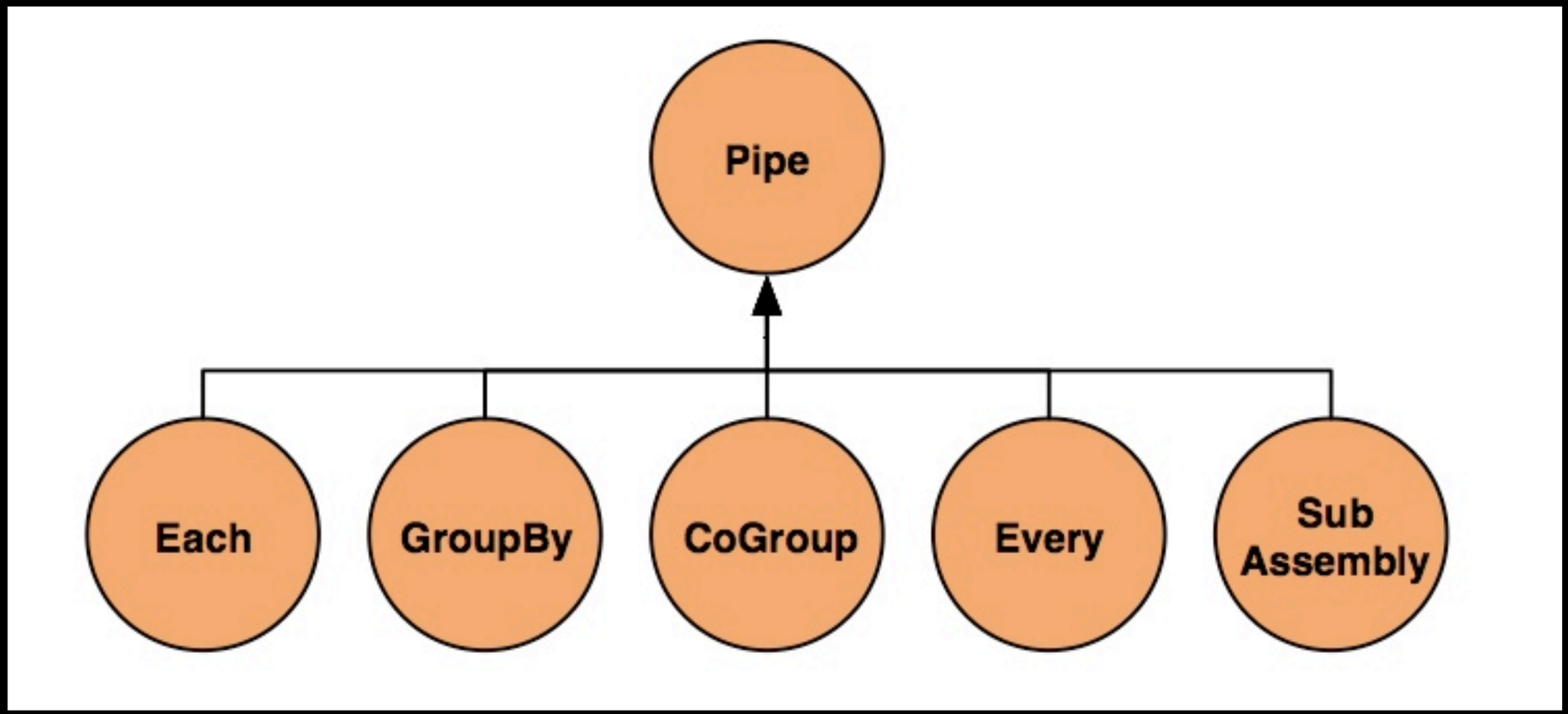
```
    Pipe wcPipe = new Each("wordcount",
        new Fields("line"),
        new RegexSplitGenerator(new Fields("word"), "\\s+"),
        new Fields("word"));
    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));


    Properties properties = new Properties();
    FlowConnector.setApplicationJarClass(properties, Main.class);

    Flow parsedLogFlow = new FlowConnector(properties)
      .connect(sourceTap, sinkTap, wcPipe);
    parsedLogFlow.start();
    parsedLogFlow.complete();
  }
}
```

```
String inputPath  = args[0];
String outputPath = args[1];

Tap sourceTap = inputPath.matches("^[^:]+://.*") ?
  new Hfs(inputScheme, inputPath)
  new Lfs(inputScheme, inputPath);
Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
  new Hfs(outputScheme, outputPath) :
  new Lfs(outputScheme, outputPath);
```

Pipe Assembly

```
Pipe wcPipe = new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));


    Properties properties = new Properties();
    FlowConnector.setApplicationJarClass(properties, Main.class);

    Flow parsedLogFlow = new FlowConnector(properties)
      .connect(sourceTap, sinkTap, wcPipe);
    parsedLogFlow.start();
    parsedLogFlow.complete();
  }
}
```

Pipe Assemblies



Pipe Assemblies

Pipe Assemblies

- Define work against a Tuple Stream

Pipe Assemblies

- Define work against a Tuple Stream
- May have multiple sources and sinks

Pipe Assemblies

- Define work against a Tuple Stream
- May have multiple sources and sinks
- Splits

Pipe Assemblies

- Define work against a Tuple Stream
- May have multiple sources and sinks
  - Splits
  - Merges

Pipe Assemblies

- Define work against a Tuple Stream
- May have multiple sources and sinks
  - Splits
  - Merges
  - Joins

Pipe Assemblies

Pipe Assemblies

- Pipe

Pipe Assemblies

- Pipe
  - Each
  - GroupBy
  - CoGroup
  - Every
  - SubAssembly

Pipe Assemblies

- Pipe
- Each
- GroupBy
- CoGroup
- Every
- SubAssembly

Pipe Assemblies

- Pipe
- Each
- GroupBy
- CoGroup
- Every
- SubAssembly

Applies a `Function` or `Filter` Operation to each Tuple

Pipe Assemblies

- Pipe
- Each
- GroupBy
- CoGroup
- Every
- SubAssembly

Pipe Assemblies

- Pipe
- Each
- GroupBy
- CoGroup
- Every
- SubAssembly

Pipe Assemblies

- Pipe
- Each
- GroupBy
- CoGroup
- Every
- SubAssembly

Group
(& merge)

Pipe Assemblies

- Pipe
  - Each
  - GroupBy
  - CoGroup
  - Every
  - SubAssembly

Joins
(inner, outer, left, right)

Pipe Assemblies

- Pipe
  - Each
  - GroupBy
  - CoGroup
  - Every
  - SubAssembly

Pipe Assemblies

- Pipe
  - Each
  - GroupBy
  - CoGroup
  - Every
  - SubAssembly

Applies an `Aggregator` (count, sum) to every group of Tuples.

Pipe Assemblies

- Pipe
  - Each
  - GroupBy
  - CoGroup
  - Every
  - SubAssembly

Nesting

Each vs. Every

Each vs. Every

- Each() is for individual Tuples

Each vs. Every

- Each() is for individual Tuples
- Every() is for groups of Tuples

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
import cascading.tuple.Fields;
import java.util.Properties;

/**
 * Wordcount example in Cascading
 */
public class Main
  {

  public static void main( String[] args )
    {
      Scheme inputScheme  = new TextLine(new Fields("offset", "line"));
      Scheme outputScheme = new TextLine();

      String inputPath  = args[0];
      String outputPath = args[1];

      Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
        new Hfs(inputScheme, inputPath)   :
        new Lfs(inputScheme, inputPath);
      Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
        new Hfs(outputScheme, outputPath) :
        new Lfs(outputScheme, outputPath);

      Pipe wcPipe = new Each("wordcount",
          new Fields("line"),
          new RegexSplitGenerator(new Fields("word"), "\\s+"),
          new Fields("word"));
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Fields()

Each

offset	line
0	the lazy brown fox

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

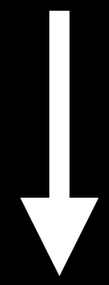
Each

offset	line
0	the lazy brown fox

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

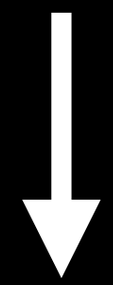
~~offset~~	line
0	the lazy brown fox



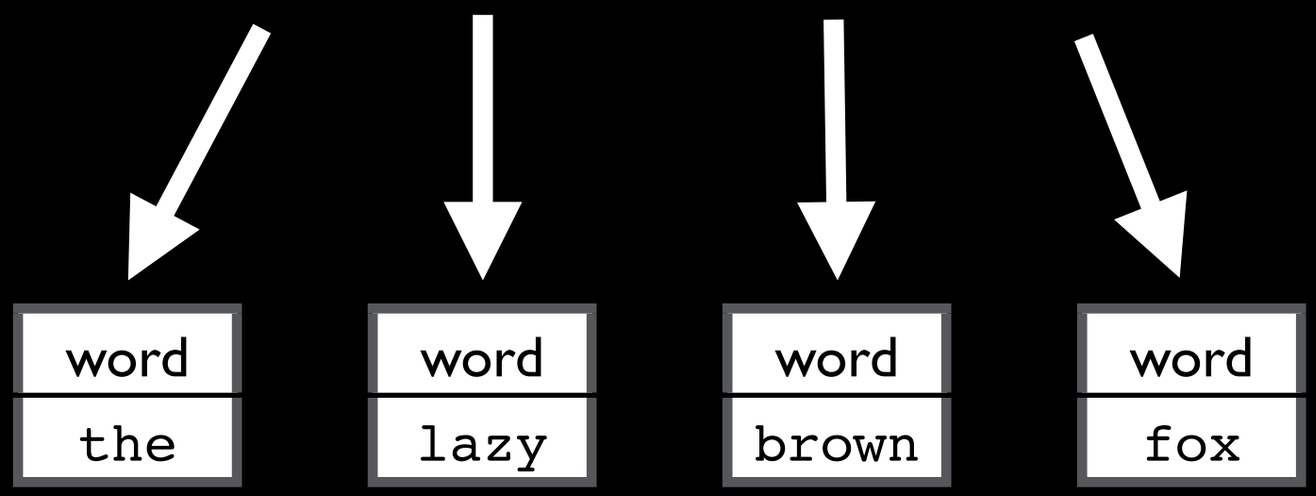
```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

~~offset~~	line
0	the lazy brown fox



```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```



word	word	word	word
the	lazy	brown	fox

Operations: Functions

Operations: Functions

- Insert()

Operations: Functions

- Insert()
- RegexParser() / RegexGenerator() / RegexReplace()

Operations: Functions

- Insert()
- RegexParser() / RegexGenerator() / RegexReplace()
- DateParser() / DateFormatter()

Operations: Functions

- Insert()
- RegexParser() / RegexGenerator() / RegexReplace()
- DateParser() / DateFormatter()
- XPathGenerator()

Operations: Functions

- Insert()
- RegexParser() / RegexGenerator() / RegexReplace()
- DateParser() / DateFormatter()
- XPathGenerator()
- Identity()

Operations: Functions

- Insert()
- RegexParser() / RegexGenerator() / RegexReplace()
- DateParser() / DateFormatter()
- XPathGenerator()
- Identity()
- etc...

Operations: Filters

Operations: Filters

- RegexFilter()

Operations: Filters

- RegexFilter()
- FilterNull()

Operations: Filters

- RegexFilter()
- FilterNull()
- And() / Or() / Not() / Xor()

Operations: Filters

- RegexFilter()
- FilterNull()
- And() / Or() / Not() / Xor()
- ExpressionFilter()

Operations: Filters

- RegexFilter()
- FilterNull()
- And() / Or() / Not() / Xor()
- ExpressionFilter()
- etc...

Each

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

Each

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
String inputPath  = args[0];
String outputPath = args[1];

Tap sourceTap = inputPath.matches( "^[^:]+://.*") ?
  new Hfs(inputScheme, inputPath)   :
  new Lfs(inputScheme, inputPath);
Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
  new Hfs(outputScheme, outputPath) :
  new Lfs(outputScheme, outputPath);
```

```
Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
    wcPipe = new GroupBy(wcPipe, new Fields("word"));
    wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));

    Properties properties = new Properties();
    FlowConnector.setApplicationJarClass(properties, Main.class);

    Flow parsedLogFlow = new FlowConnector(properties)
      .connect(sourceTap, sinkTap, wcPipe);
    parsedLogFlow.start();
    parsedLogFlow.complete();
  }
}
```

```
    new Lfs(inputScheme, inputPath);
Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
  new Hfs(outputScheme, outputPath) :
  new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);

Flow parsedLogFlow = new FlowConnector(properties)
  .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

```
    new Lfs(inputScheme, inputPath);
Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
  new Hfs(outputScheme, outputPath) :
  new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);

Flow parsedLogFlow = new FlowConnector(properties)
  .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

```
    new Lfs(inputScheme, inputPath);
Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
  new Hfs(outputScheme, outputPath) :
  new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);

Flow parsedLogFlow = new FlowConnector(properties)
  .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

Every

```
new Every(previousPipe,
          argumentSelector,
          operation,
          outputSelector)
```

Every

```
new Every(previousPipe,
          argumentSelector,
          operation,
          outputSelector)
```

```
new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
new Every(wcPipe, new Count(), new Fields("count", "word"));
```

Operations: Aggregator

Operations: Aggregator

- Average()

Operations: Aggregator

- Average()
- Count()

Operations: Aggregator

- Average()
- Count()
- First() / Last()

Operations: Aggregator

- Average()
- Count()
- First() / Last()
- Min() / Max()

Operations: Aggregator

- Average()
- Count()
- First() / Last()
- Min() / Max()
- Sum()

Every

```
new Every(previousPipe,
          argumentSelector,
          operation,
          outputSelector)
```

Every

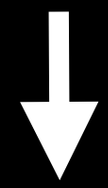
```
new Every(previousPipe,
          argumentSelector,
          operation,
          outputSelector)
```

```
new Every(wcPipe, new Count(), new Fields("count", "word"));
```

Every

```
new Every(previousPipe,
          argumentSelector,
          operation,
          outputSelector)
```

```
new Every(wcPipe, new Count(), new Fields("count", "word"));
```



```
new Every(wcPipe, Fields.ALL, new Count(), new Fields("count", "word"));
```

Field Selection

Predefined Field Sets

Predefined Field Sets

Fields.ALL	all available fields

Predefined Field Sets

Fields.ALL	all available fields
Fields.GROUP	fields used for last grouping

Predefined Field Sets

Fields.ALL	all available fields
Fields.GROUP	fields used for last grouping
Fields.VALUES	fields not used for last grouping

Predefined Field Sets

Fields.ALL	all available fields
Fields.GROUP	fields used for last grouping
Fields.VALUES	fields not used for last grouping
Fields.ARGS	fields of argument Tuple (for Operations)

Predefined Field Sets

Fields.ALL	all available fields
Fields.GROUP	fields used for last grouping
Fields.VALUES	fields not used for last grouping
Fields.ARGS	fields of argument Tuple (for Operations)
Fields.RESULTS	replaces input with Operation result (for Pipes)

Field Selection

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

```
pipe = new Each(pipe,
           new Fields("timestamp"),
           new DateParser("yyyy-MM-dd HH:mm:ss"),
           new Fields("timestamp"));
```

Field Selection

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

Operation Input Tuple =

Original Tuple ∩ Argument Selector

Field Selection

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

Argument Selector

timestamp

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

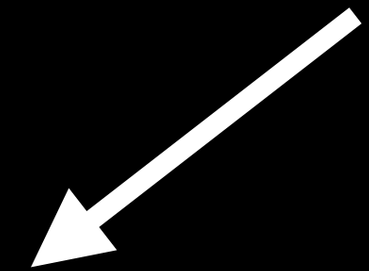
Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

Argument Selector

timestamp



```
pipe = new Each(pipe,
               new Fields("timestamp"),
               new DateParser("yyyy-MM-dd HH:mm:ss"),
               new Fields("ts", "search_term", "visitor_id"));
```

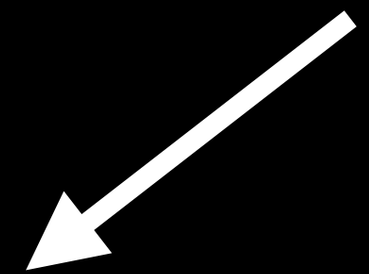
Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

Argument Selector

timestamp



```
pipe = new Each(pipe,
               new Fields("timestamp"),
               new DateParser("yyyy-MM-dd HH:mm:ss"),
               new Fields("ts", "search_term", "visitor_id"));
```

Input to DateParser will be:

timestamp

Field Selection

```
new Each(previousPipe,
         argumentSelector,
         operation,
         outputSelector)
```

Output Tuple =

Original Tuple ⊕ Operation Tuple ∩ Output Selector

Field Selection

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

DateParser Output

⊕

ts

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

DateParser Output

ts

Output Selector

ts	search_ term	visitor_ id

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

DateParser Output

ts

Output Selector

ts	search_ term	visitor_ id

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

DateParser Output

ts

Output Selector

ts	search_ term	visitor_ id

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

DateParser Output

ts

Output Selector

ts	search_ term	visitor_ id

```
pipe = new Each(pipe,
            new Fields("timestamp"),
            new DateParser("yyyy-MM-dd HH:mm:ss"),
            new Fields("ts", "search_term", "visitor_id"));
```

Output of Each will be:

ts	search_ term	visitor_ id

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

DateParser Output

ts

Output Selector

ts	search_ term	visitor_ id

```
pipe = new Each(pipe,
           new Fields("timestamp"),
           new DateParser("yyyy-MM-dd HH:mm:ss"),
           new Fields("ts", "search_term", "visitor_id"));
```

Output of Each will be:

ts	search_ term	visitor_ id

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

DateParser Output

⊕

ts

Output Selector

ts	search_ term	visitor_ id

```
pipe = new Each(pipe,
           new Fields("timestamp"),
           new DateParser("yyyy-MM-dd HH:mm:ss"),
           new Fields("ts", "search_term", "visitor_id"));
```

Output of Each will be:

ts	search_ term	visitor_ id

Field Selection

Original Tuple

id	visitor_ id	search_ term	time stamp	page_ number

⊕

DateParser Output

ts

Output Selector

ts	search_ term	visitor_ id

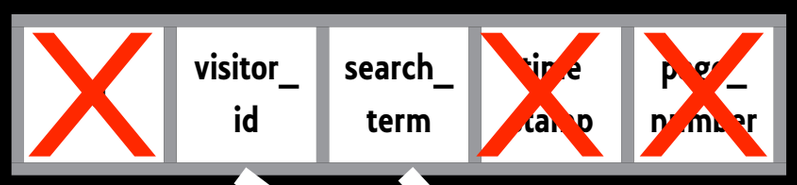
```
pipe = new Each(pipe,
           new Fields("timestamp"),
           new DateParser("yyyy-MM-dd HH:mm:ss"),
           new Fields("ts", "search_term", "visitor_id"));
```

Output of Each will be:

ts	search_ term	visitor_ id

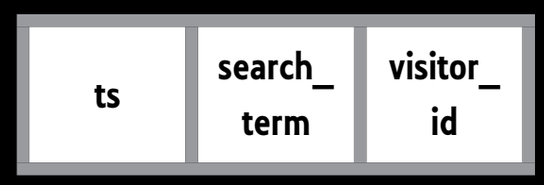
Field Selection

Original Tuple



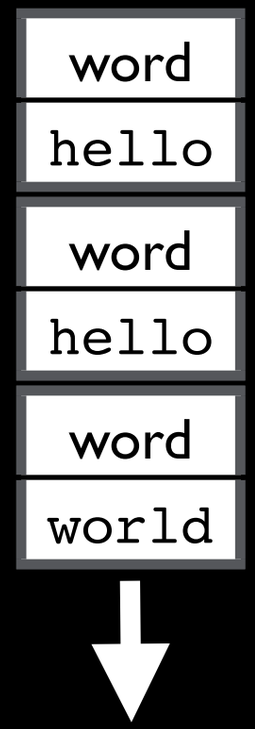
DateParser Output

Output Selector



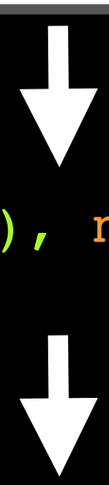

```
pipe = new Each(pipe,
           new Fields("timestamp"),
           new DateParser("yyyy-MM-dd HH:mm:ss"),
           new Fields("ts", "search_term", "visitor_id"));
```

Output of Each will be:




```
new Every(wcPipe, new Count(), new Fields("count", "word"));
```

word
hello
word
hello
word
world



```
new Every(wcPipe, new Count(), new Fields("count", "word"));
```


count	word
2	hello

count	word
1	world

```
    new Lfs(inputScheme, inputPath);
Tap sinkTap   = outputPath.matches("^[^:]+://.*") ?
  new Hfs(outputScheme, outputPath) :
  new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));
```

```
wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);

Flow parsedLogFlow = new FlowConnector(properties)
  .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

```
    new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));

wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);
```

```
Flow parsedLogFlow = new FlowConnector(properties)
    .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

```
    new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));

wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);
```

```
Flow parsedLogFlow = new FlowConnector(properties)
    .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

```
    new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));

wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);

Flow parsedLogFlow = new FlowConnector(properties)
  .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
  }
}
```

```
    new Lfs(outputScheme, outputPath);

Pipe wcPipe =
new Each("wordcount",
    new Fields("line"),
    new RegexSplitGenerator(new Fields("word"), "\\s+"),
    new Fields("word"));

wcPipe = new GroupBy(wcPipe, new Fields("word"));
wcPipe = new Every(wcPipe, new Count(), new Fields("count", "word"));
```

```
Properties properties = new Properties();
FlowConnector.setApplicationJarClass(properties, Main.class);

Flow parsedLogFlow = new FlowConnector(properties)
  .connect(sourceTap, sinkTap, wcPipe);
parsedLogFlow.start();
parsedLogFlow.complete();
```

```
    }
}
```

Run

input:

```
mary had a little lamb
little lamb
little lamb
mary had a little lamb
whose fleece was white as snow
```

input:

```
mary had a little lamb
little lamb
little lamb
mary had a little lamb
whose fleece was white as snow
```

command:

```
hadoop jar ./target/cascading-tutorial-1.0.0.jar \
  data/sources/misc/mary.txt data/output
```

output:

```
2       a
1       as
1       fleece
2       had
4       lamb
4       little
2       mary
1       snow
1       was
1       white
1       whose
```

What’s next?

apply operations to tuple streams

apply operations to tuple streams

repeat.

Cookbook

Cookbook

```
// remove all tuples where search_term is '-'
pipe = new Each(pipe,
          new Fields("search_term"),
          new RegexFilter("^(\\-)$", true));
```

Cookbook

```
// convert "timestamp" String field to "ts" long field
pipe = new Each(pipe,
           new Fields("timestamp"),
           new DateParser("yyyy-MM-dd HH:mm:ss"),
           Fields.ALL);
```

Cookbook

```
// td - timestamp of day-level granularity
pipe = new Each(pipe,
            new Fields("ts"),
            new ExpressionFunction(
                new Fields("td"),
                "ts - (ts % (24 * 60 * 60 * 1000))",
                long.class),
            Fields.ALL);
```

Cookbook

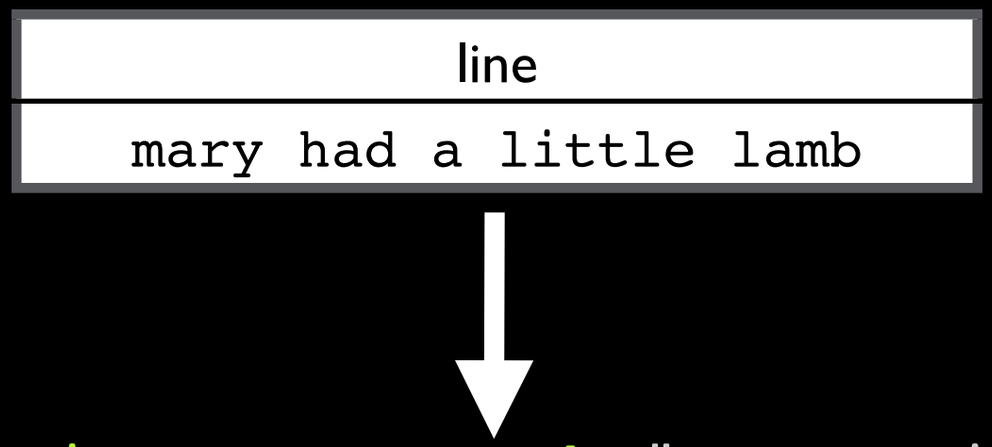
```
// SELECT DISTINCT visitor_id

// GROUP BY visitor_id ORDER BY ts
pipe = new GroupBy(pipe,
           new Fields("visitor_id"),
           new Fields("ts"));

// take the First() tuple of every grouping
pipe = new Every(pipe,
           Fields.ALL,
           new First(),
           Fields.RESULTS);
```

Custom Operations

Desired Operation

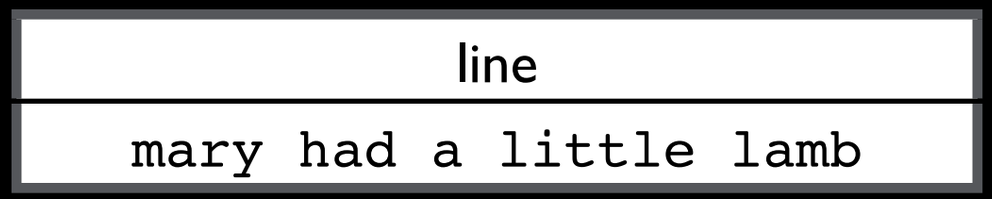
```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```

Desired Operation

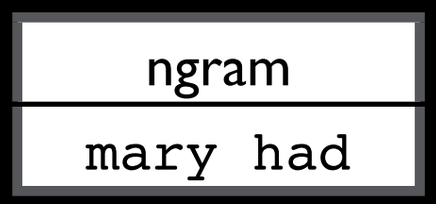



```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```

Desired Operation

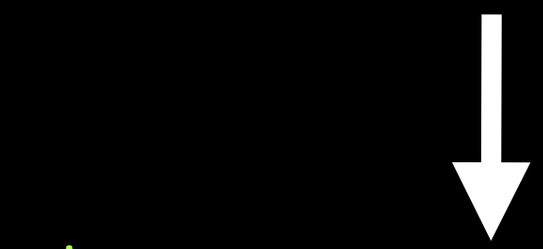



```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```

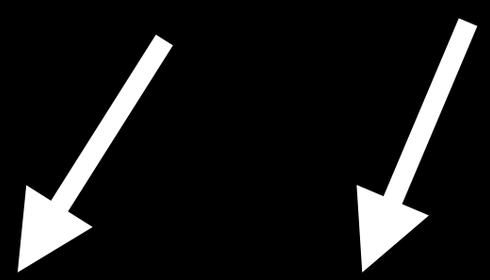


Desired Operation

line
mary had a little lamb



```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```



ngram	ngram
mary had	had a

Desired Operation

line
mary had a little lamb

```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```

ngram	ngram	ngram
mary had	had a	a little

Desired Operation

line
mary had a little lamb

```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```

ngram	ngram	ngram	ngram
mary had	had a	a little	little lamb

Custom Operations

Custom Operations

- extend `BaseOperation`

Custom Operations

- extend `BaseOperation`
- implement `Function` (or `Filter`)

Custom Operations

- extend `BaseOperation`
- implement `Function` (or `Filter`)
- implement `operate`

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);
    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

Tuple

123	vis-abc	tacos	2009-10-08 03:21:23	1

Tuple

zero-indexed, ordered values

Tuple

123	vis-abc	tacos	2009-10-08 03:21:23	1

Fields

id	visitor_ id	search_ term	time stamp	page_ number

list of strings representing field names

Tuple					
	123	vis-abc	tacos	2009-10-08 03:21:23	1

Fields					
	id	visitor_ id	search_ term	time stamp	page_ number

TupleEntry

id	visitor_id	search_term	time stamp	page_number
123	vis-abc	tacos	2009-10-08 03:21:23	1

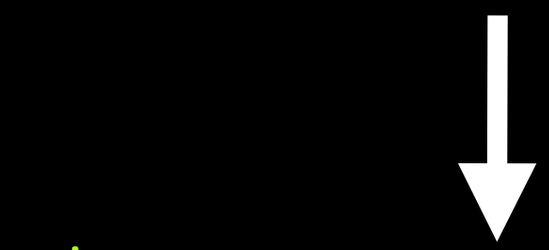
```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

```
public class NGramTokenizer extends BaseOperation implements Function
{
  private int size;

  public NGramTokenizer(int size)
  {
    super(new Fields("ngram"));
    this.size = size;
  }

  public void operate(FlowProcess flowProcess, FunctionCall functionCall)
  {
    String token;
    TupleEntry arguments = functionCall.getArguments();
    com.attinteractive.util.NGramTokenizer t =
      new com.attinteractive.util.NGramTokenizer(arguments.getString(0), this.size);

    while((token = t.next()) != null) {
      TupleEntry result = new TupleEntry(new Fields("ngram"), new Tuple(token));
      functionCall.getOutputCollector().add(result);
    }
  }
}
```

Success

line
mary had a little lamb



```
Pipe pipe = new Each("ngram pipe",
    new Fields("line"),
    new NGramTokenizer(2),
    new Fields("ngram"));
```

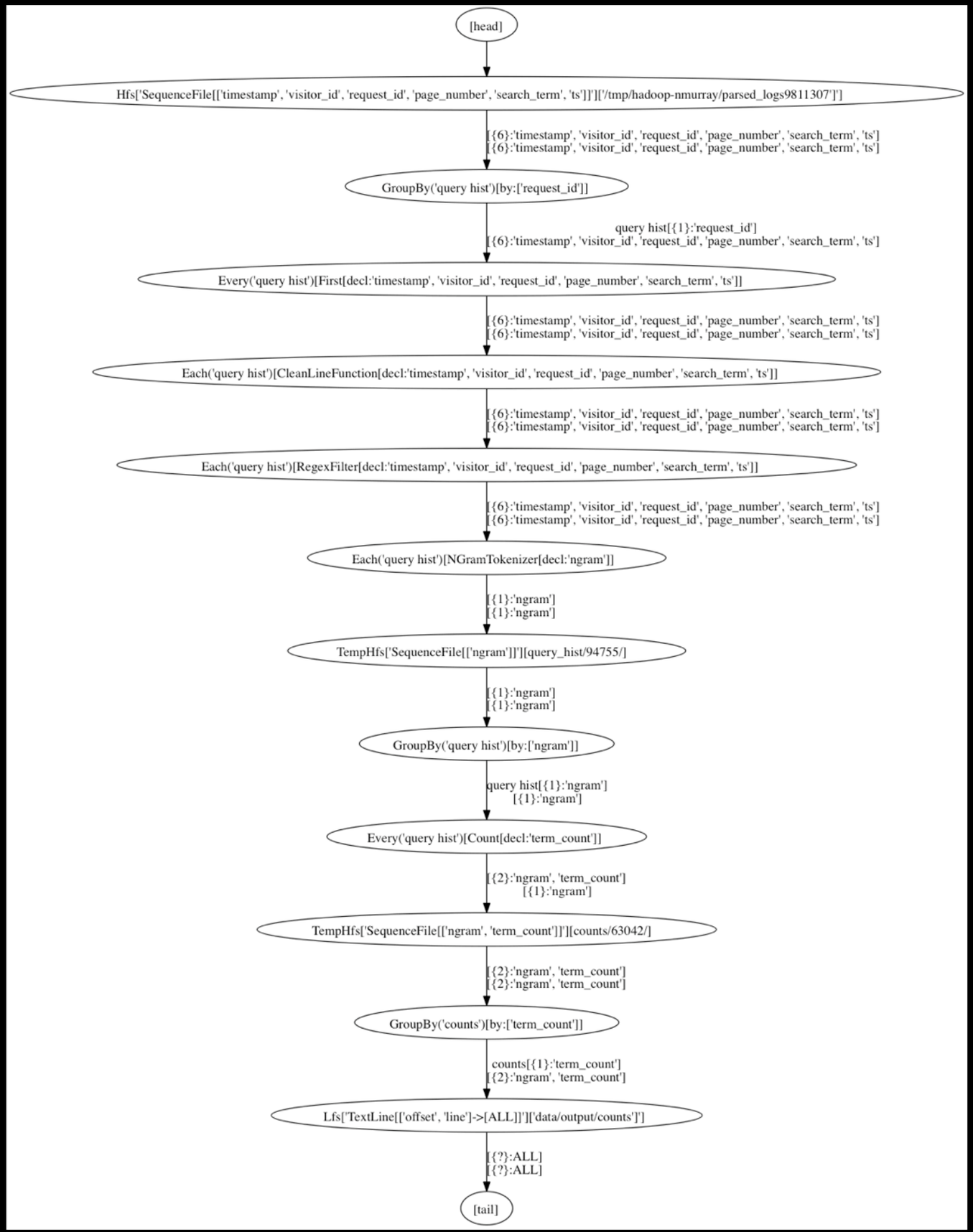
ngram	ngram	ngram	ngram
mary had	had a	a little	little lamb

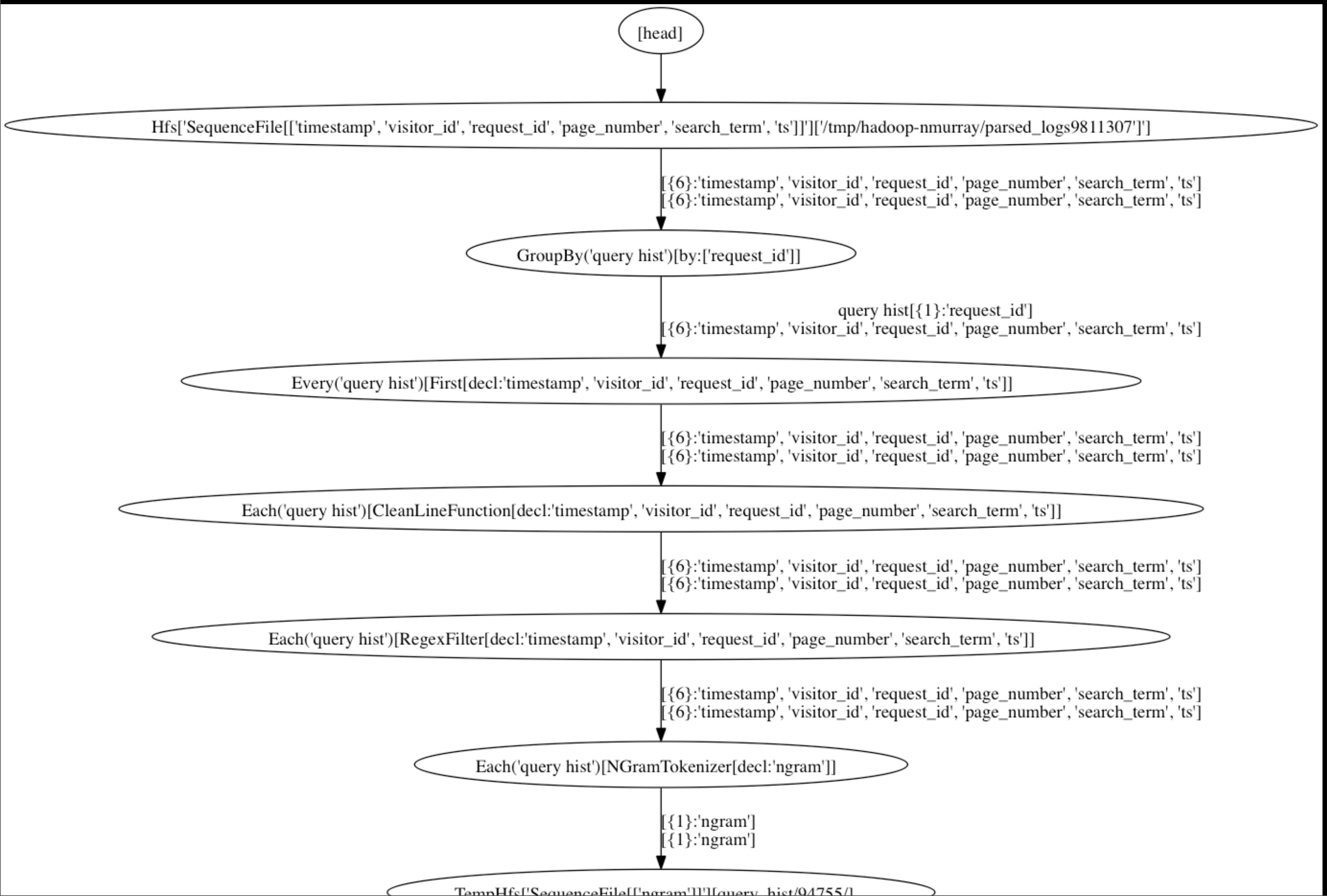
code examples

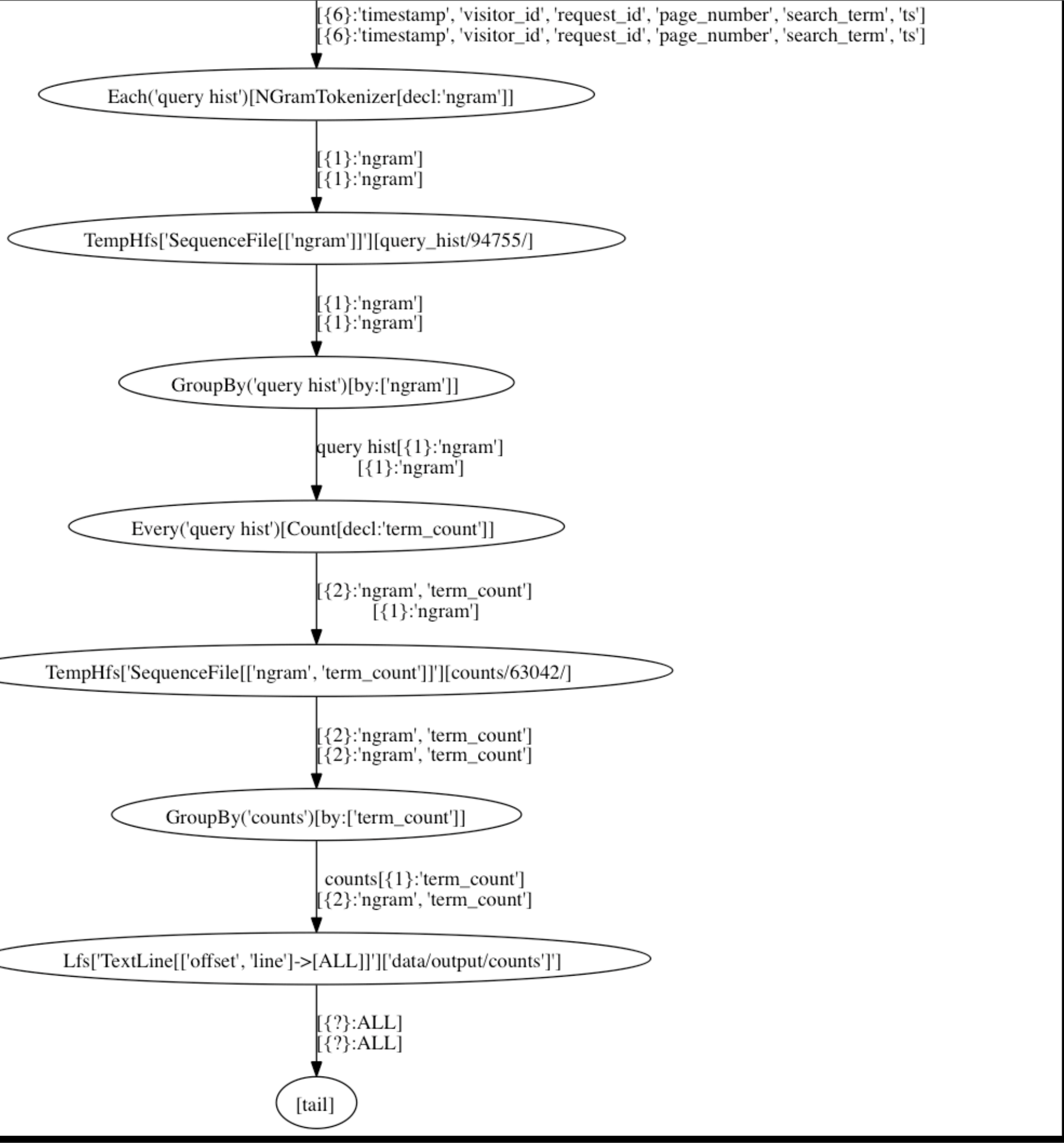
dot

dot

```
Flow importFlow = flowConnector.connect(sourceTap, tmpIntermediateTap, importPipe);
importFlow.writeDOT( "import-flow.dot" );
```


[head]
Hfs['SequenceFile[['timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]']['/tmp/hadoop-nmurray/parsed_logs9811307']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
GroupBy('query hist')[by:['request_id']]
query hist[{1}:'request_id']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Every('query hist')[First[decl:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[CleanLineFunction[decl:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[RegexFilter[decl:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[NGramTokenizer[decl:'ngram']]
[{1}:'ngram']
[{1}:'ngram']
TempHfs['SequenceFile[['ngram']]']['query_hist/94755/']
[{1}:'ngram']
[{1}:'ngram']
GroupBy('query hist')[by:['ngram']]
query hist[{1}:'ngram']
[{1}:'ngram']
Every('query hist')[Count[decl:'term_count']]
[{2}:'ngram', 'term_count']
[{1}:'ngram']
TempHfs['SequenceFile[['ngram', 'term_count']]']['counts/63042/']
[{2}:'ngram', 'term_count']
[{2}:'ngram', 'term_count']
GroupBy('counts')[by:['term_count']]
counts[{1}:'term_count']
[{2}:'ngram', 'term_count']
Lfs['TextLine[['offset', 'line']->[ALL]]']['data/output/counts']
[{?}:ALL]
[{?}:ALL]
[tail]


[head]
Hfs['SequenceFile[['timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]']['/tmp/hadoop-nmurray/parsed_logs9811307']']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
GroupBy('query hist')[by:['request_id']]
query hist[{1}:'request_id']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Every('query hist')[First[decl:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[CleanLineFunction[decl:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[RegexFilter[decl:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']]
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[NGramTokenizer[decl:'ngram']]
[{1}:'ngram']
[{1}:'ngram']


[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
[{6}:'timestamp', 'visitor_id', 'request_id', 'page_number', 'search_term', 'ts']
Each('query hist')[NGramTokenizer[decl:'ngram']]
[{1}:'ngram']
[{1}:'ngram']
TempHfs['SequenceFile[['ngram']]']['query_hist/94755/']
[{1}:'ngram']
[{1}:'ngram']
GroupBy('query hist')[by:['ngram']]
query hist[{1}:'ngram']
[{1}:'ngram']
Every('query hist')[Count[decl:'term_count']]
[{2}:'ngram', 'term_count']
[{1}:'ngram']
TempHfs['SequenceFile[['ngram', 'term_count']]']['counts/63042/']
[{2}:'ngram', 'term_count']
[{2}:'ngram', 'term_count']
GroupBy('counts')[by:['term_count']]
counts[{1}:'term_count']
[{2}:'ngram', 'term_count']
Lfs['TextLine[['offset', 'line']->[ALL]]']['data/output/counts']']
[{?}:ALL]
[{?}:ALL]
[tail]

What next?

What next?

What next?

- http://www.cascading.org/

What next?

- http://www.cascading.org/
- http://bit.ly/cascading

Questions?

<nmurray@att.com>