Christoph Böhme

Analysis of library metadata with Metafacture

Agenda

13:00 — a short introduction to Metafacture

13:30 — warm-up exercises

15:00 — triples and counting

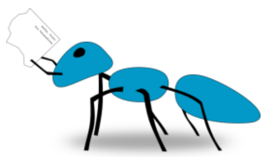
15:30 — exercises on counting data
(incl. coffee break)

16:00 — joining data sets and analysing them

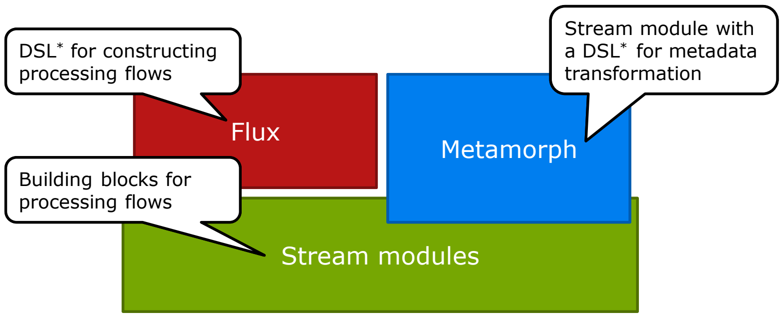
16:30 — exercises on joining data

18:40 — wrapping up

Part 1
A short introduction to Metafacture

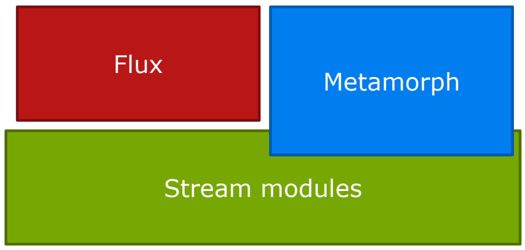


Overview of Metafacture

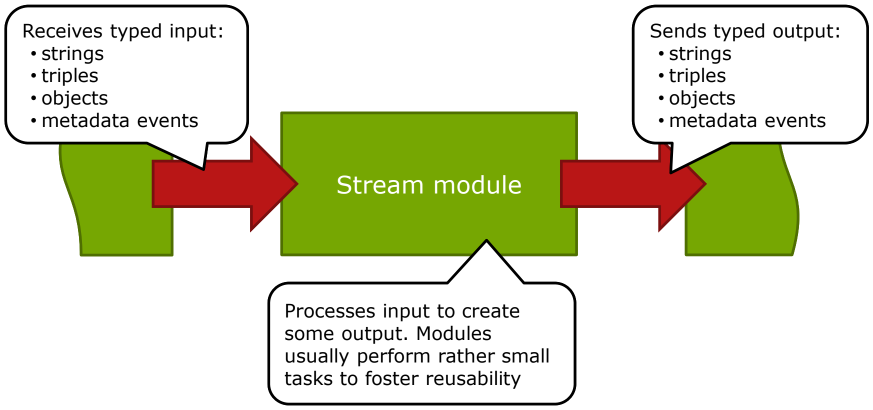

DSL* for constructing processing flows
Flux
Stream module with a DSL* for metadata transformation
Metamorph
Building blocks for processing flows
Stream modules

*DSL: Domain specific Language

Overview of Metafacture

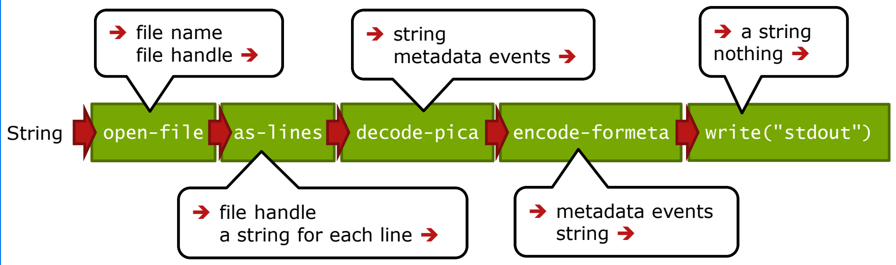


The basic building block of Metafacture

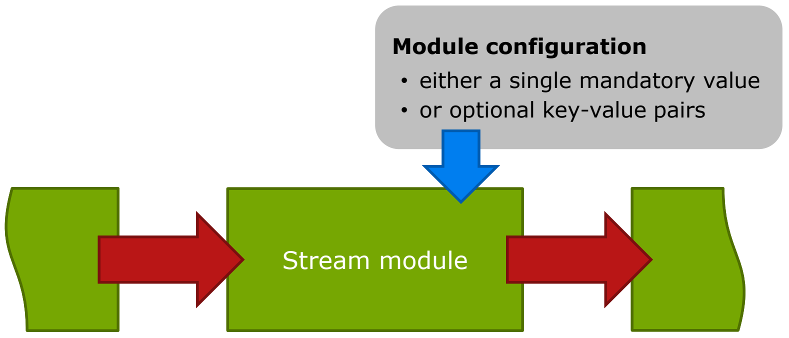


A simple processing flow

Read and print a file containing pica records:

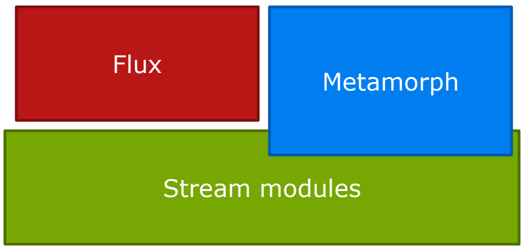


Module configuration

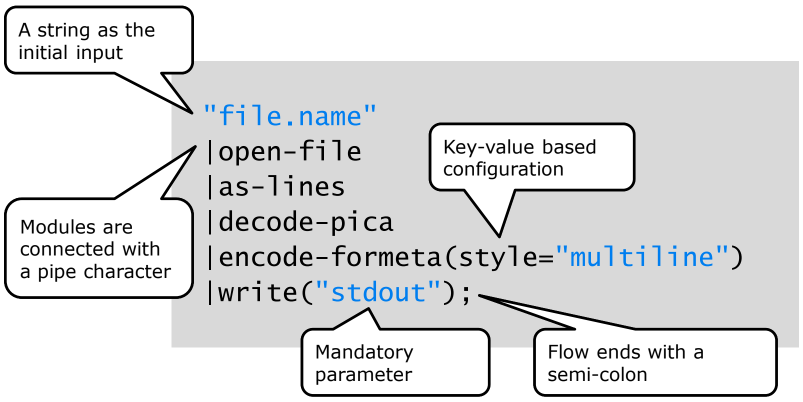


Module configuration
• either a single mandatory value
• or optional key-value pairs
Stream module

Overview of Metafacture



Describing flows with Flux



```
"file.name"
|open-file
|as-lines
|decode-pica
|encode-formeta(style="multiline")
|write("stdout");
```

A string as the initial input
Key-value based configuration
Modules are connected with a pipe character
Mandatory parameter
Flow ends with a semi-colon

Variables and comments in Flux

Define default values for the variables **in** and **out**

Comments start with two slashes

Use variables instead of directly entering a string

```
default in = "file.name";
default out = "stdout";

in
|open-file
// ...
|write(out);
```

Neat features for strings in Flux

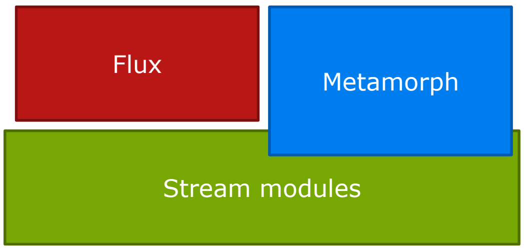
FLUX_DIR always contains the path of the parent folder of the flux file

Strings can be concatenated

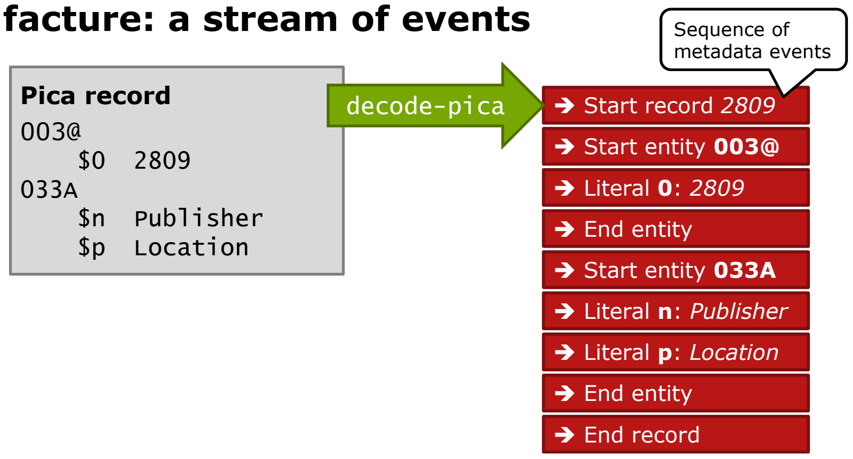
Strings can be multi-line

```
default in = FLUX_DIR + "file.name";

default haiku =
"A dropped ice cream cone
One red ant, then two, then four
Suddenly - hundreds!";
```

Overview of Metafacture



Representation of metadata in Metafacture: a stream of events

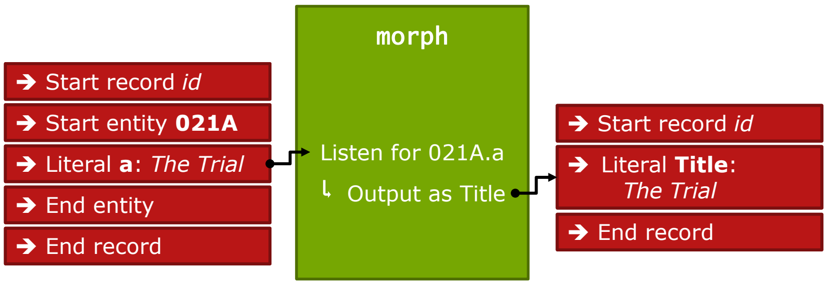


Pica record

```
003@
    $0  2809
033A
    $n  Publisher
    $p  Location
```

➔ Start record *2809*
➔ Start entity **003@**
➔ Literal **0**: *2809*
➔ End entity
➔ Start entity **033A**
➔ Literal **n**: *Publisher*
➔ Literal **p**: *Location*
➔ End entity
➔ End record

Processing metadata events with Metamorph



Metamorph: data statements

```
<?xml version="1.0" encoding="UTF-8"?>

<metamorph xmlns="http://www.culturegraph.org/metamorph"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   version="1" entityMarker=".">

      <rules>
         <data source="021A.a" name="Title" />
      </rules>

</metamorph>
```

Separator for entities and literal names

Name of the literal to listen for

Name of the literal that is output

Metamorph: modifying data

```
...

<rules>

   <data source="021A.a" name="Title">
       <regexp match="^(The) (.*)$" format="${2}, ${1}" />
   </data>

</rules>

...
```

Process the data value before outputting it. You can specify multiple functions here

Metamorph: combining data

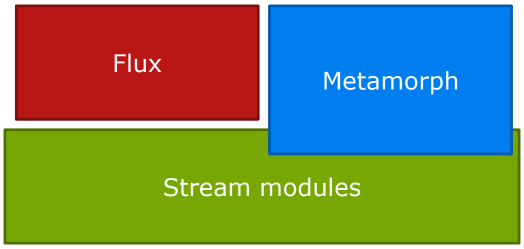
Name of the generated literal. It can include variables, too

Literal value constructed from the variables from the data statements below

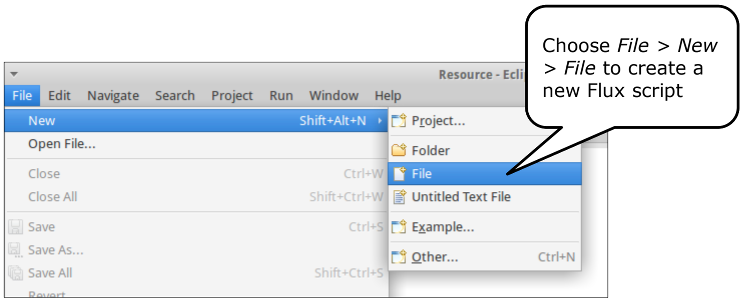
```
...

<rules>

   <combine name="Publisher" value="${Pub}: ${Loc}">
      <data source="033A.n" name="Pub" />
      <data source="033A.p" name="Loc" />
   </combine>

</rules>

...
```

The data statements do not generate output but create variables instead

Overview of Metafacture



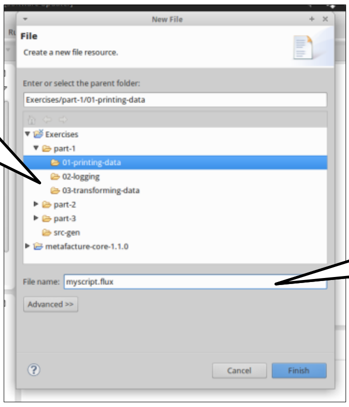
Creating Flux scripts in Eclipse



Choose *File* > *New* > *File* to create a new Flux script

Creating Flux scripts in Eclipse

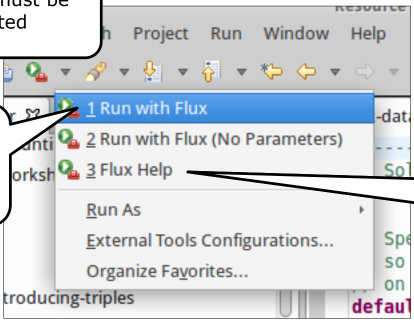
Select the project folder where you want to save your script



Enter a file name with the extension ".flux"

Running Flux scripts in Eclipse

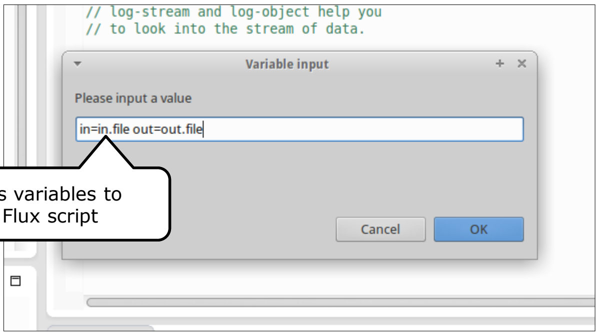
! The Flux script must be saved and selected



Choose "Run with Flux" to execute the selected Flux script

Choose "Flux Help" to get a list of all supported modules

Running Flux scripts in Eclipse



```
// log-stream and log-object help you
// to look into the stream of data.
```

```
in=in.file out=out.file
```

Pass variables to the Flux script

Running Flux scripts from the prompt

```
C:\> flux.bat myscript.flux
```

```
C:\> flux.bat myscript.flux in=in.file out=out.file
```

Use variables to pass user-defined values into a script

Running Flux scripts from the prompt

```
$ ./flux.sh myscript.flux
```

```
$ ./flux.sh myscript.flux in=in.file out=out.file
```

Use variables to pass user-defined values into a script

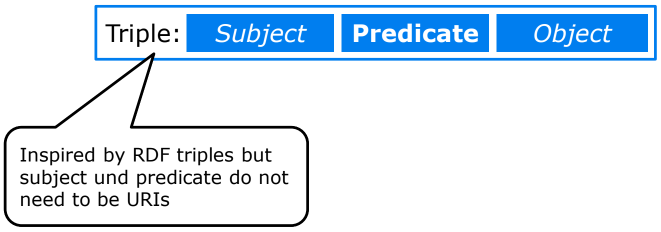
Exercises: Documentation

- The hand outs equip you with everything you need for the exercises
- Running "Flux Help" in Eclipse (or flux.sh/flux.bat without additional arguments from the prompt)
- Use auto-completion in Eclipse (press CTRL+Space)
- The Metafacture-Wiki: https://github.com/culturegraph/metafacture-core/wiki

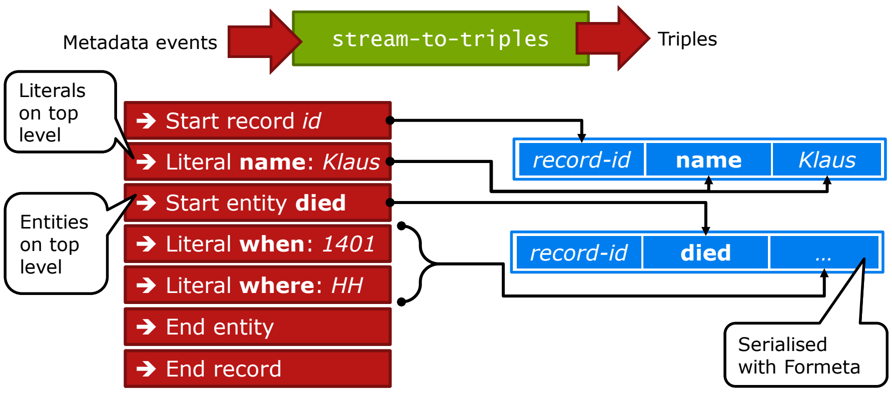
Exercises part 1
Warm-up

Part 2
Triples and counting

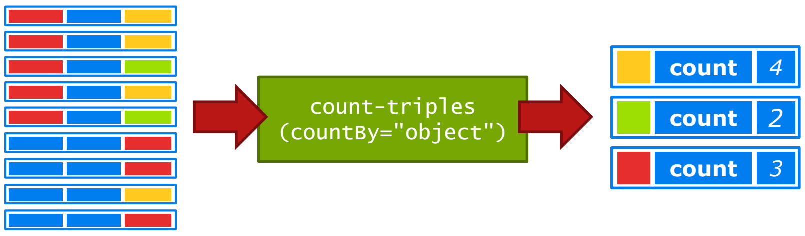
The triple


Triple:
Subject
Predicate
Object
Inspired by RDF triples but subject und predicate do not need to be URIs

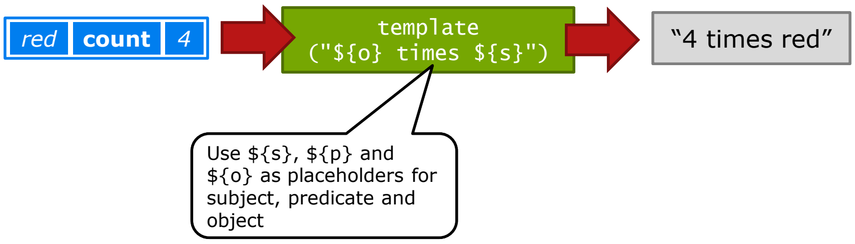
Generating triples



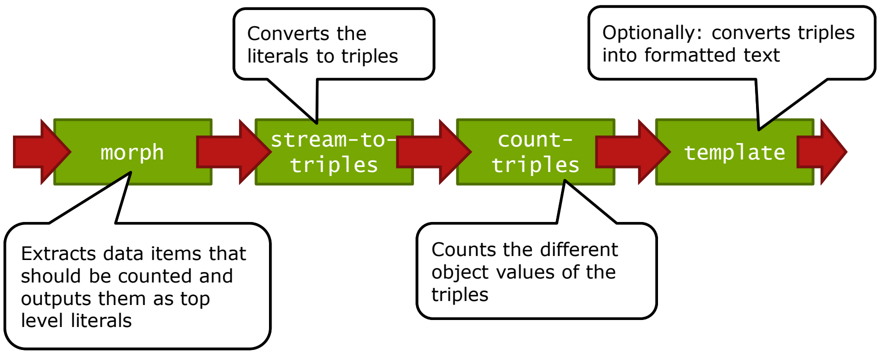
Counting triples



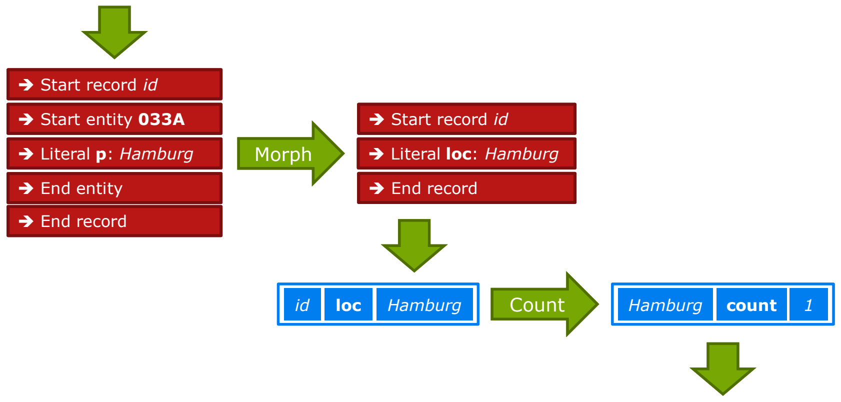
Outputting triples


red count 4
template
("${o} times ${s}")
"4 times red"
Use ${s}, ${p} and ${o} as placeholders for subject, predicate and object

Counting data values


Converts the literals to triples
Optionally: converts triples into formatted text
morph
stream-to-triples
count-triples
template
Extracts data items that should be counted and outputs them as top level literals
Counts the different object values of the triples

Counting data values: flow of data



Metamorph: choosing data

```
...
<rules>

   <choose name="Location">
      <data source="033A.p">
         <regexp match="^Ffm$" format="Frankfurt a. M." />
      </data>
      <data source="033A.p" />
   </choose>

</rules>
...
```

Only the value of the topmost data-statement that generates output is returned by the choose-statement

Metamorph: generating constant values

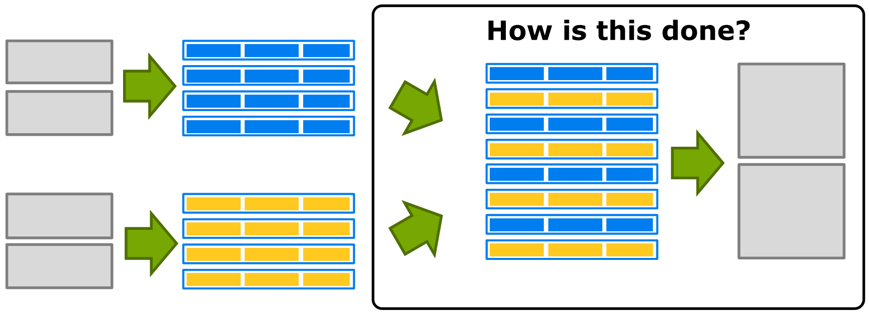
```
...

<rules>

   <data source="021A.a" name="Title">
       <constant value="All books have the same name" />
   </data>

</rules>

...
```

No matter what the value of literal 021A.a is, always output the defined value

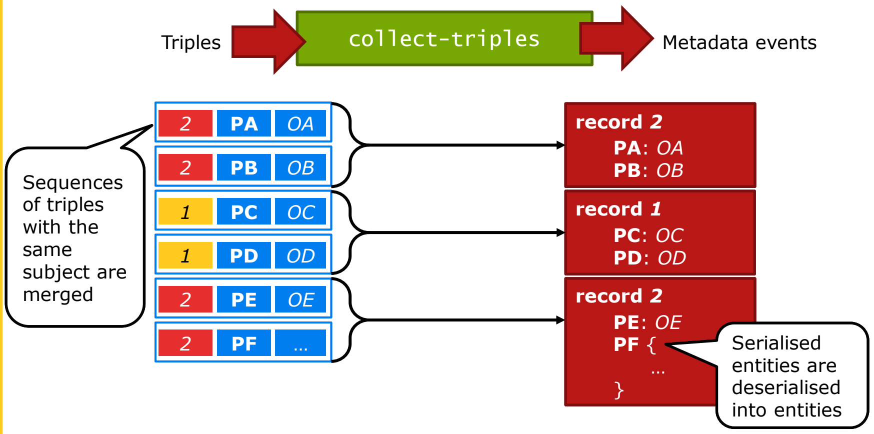
Exercises part 2
Triples and counting

Part 3
Joining data sets and analysing them

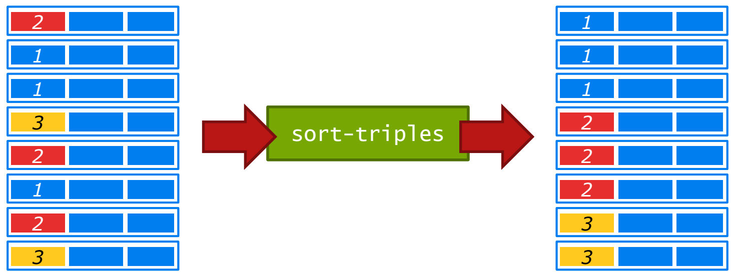
Joining streams of data



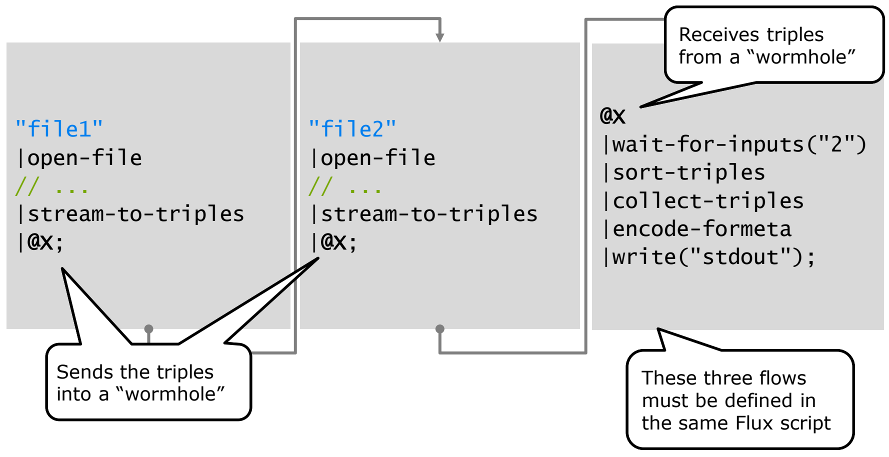
Converting triples into records


Triples
collect-triples
Metadata events
2 PA OA
2 PB OB
1 PC OC
1 PD OD
2 PE OE
2 PF ...
Sequences of triples with the same subject are merged
record 2 PA: OA PB: OB
record 1 PC: OC PD: OD
record 2 PE: OE PF { ... }
Serialised entities are deserialised into entities

Sorting triples



Linking streams in Flux with wormholes



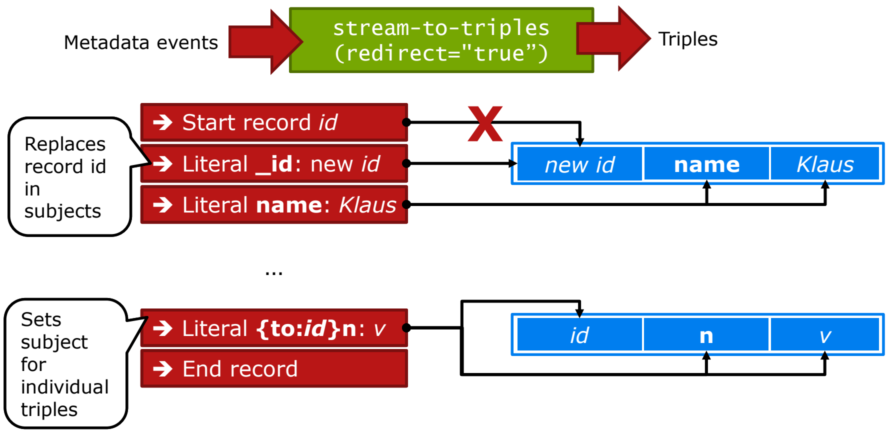
```
"file1"
|open-file
// ...
|stream-to-triples
|@X;
```

```
"file2"
|open-file
// ...
|stream-to-triples
|@X;
```

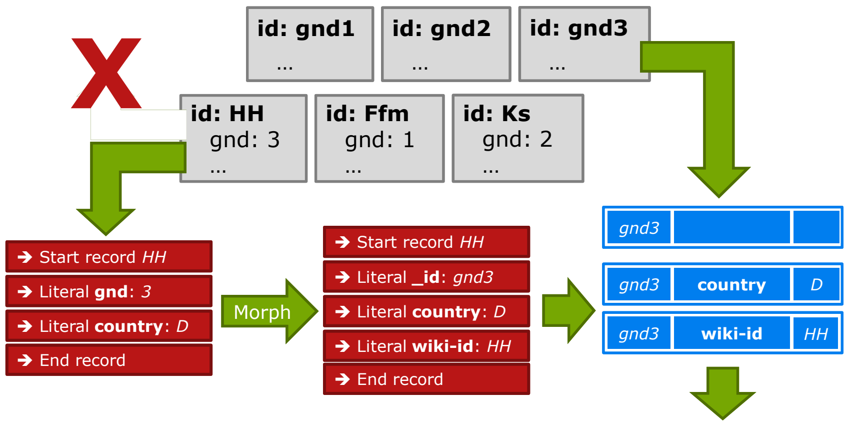
```
@X
|wait-for-inputs("2")
|sort-triples
|collect-triples
|encode-formeta
|write("stdout");
```

Receives triples from a “wormhole”
Sends the triples into a “wormhole”
These three flows must be defined in the same Flux script

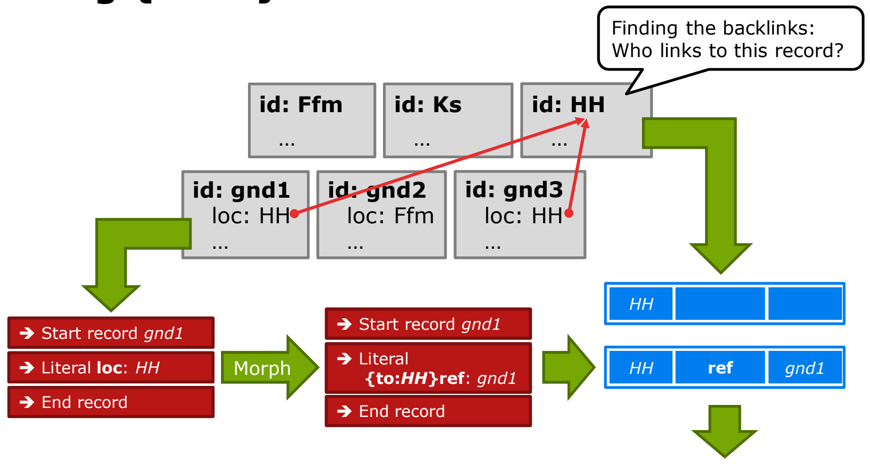
Advanced triplification: ID redirection


Metadata events
stream-to-triples (redirect="true")
Triples
Replaces record id in subjects
Start record *id*
Literal **_id**: new *id*
Literal **name**: *Klaus*
new id
name
Klaus
…
Sets subject for individual triples
Literal **{to:*id*}n**: *v*
End record
id
n
v

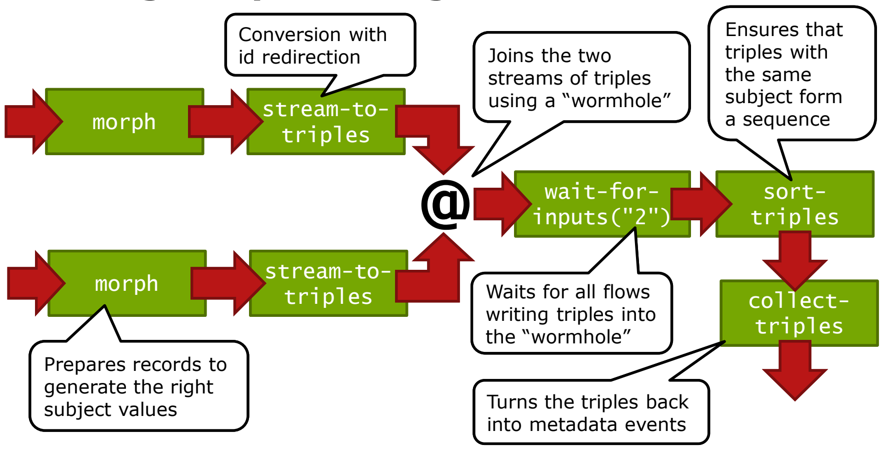
Using _id-redirection



Using {to:ID}-redirection



Putting the pieces together


Conversion with id redirection
Joins the two streams of triples using a “wormhole”
Ensures that triples with the same subject form a sequence
morph
stream-to-triples
@
wait-for-inputs("2")
sort-triples
morph
stream-to-triples
collect-triples
Waits for all flows writing triples into the “wormhole”
Prepares records to generate the right subject values
Turns the triples back into metadata events

Metamorph: what else?

```
...

<rules>

   <data source="_id" name="recordId" />
   <data source="033A.n" name="Publisher" />
   <data source="_else" />

</rules>

...
```

Metamorph makes the record id of the current record as `_id` available

Any literal not handled by any other data-statement is passed to this statement. It can be used to pass data through Metamorph

Exercises part 3

Joining data sets and analysing them

Wrapping up

What did we learn today?

- Foundations of processing metadata with Flux and Metamorph
- Exploring data sets by quantifying data values
- Joining data sets and analysing their relations
- Typical patterns for analysing data with Metafacture

  These patterns are similar to the way Hadoop operates: This makes migration from your desktop to a Hadoop cluster easy

Metafacture

- Not only designed for data analysis but for metadata processing in general
- Software tool and library: It can easily be integrated into other applications
- Flux and Metamorph are extendable
- It is open source at http://culturegraph.github.io/

Thank you very much!

Further questions?

Contact me at c.boehme@dnb.de
or join the mailing list:
http://lists.dnb.de/mailman/listinfo/metafacture

Workshop materials

Download from http://b3e.net/metafacture-swib14/workshop.zip