bonobo

Simple ETL in Python 3.5+

Romain Dorgueil
@rdorgueil

CTO/Hacker in Residence	L’Atelier BNP Paribas
Technical Co-founder	WeAreTheShops
(Solo) Founder	RDC Dist. Agency
Eng. Manager	Sensio/SensioLabs
Developer	AffiliationWizard

Felt too young in a Linux Cauldron
Dismantler of Atari computers
Basic literacy using a Minitel
Guitars & accordions
Off by one baby
Inception

L'Atelier BNP PARIBAS LAUNCHPAD

STARTUP ACCELERATION PROGRAMS

NO HYPE, JUST BUSINESS

launchpad.atelier.net

bonobo

Simple ETL in Python 3.5+

Plan

- History of Extract Transform Load
  - Concept ; Existing tools ; Related tools ; Ignition
- Practical Bonobo
  - Tutorial ; Under the hood ; Demo ; Plugins & Extensions ; More demos
- Wrap up
  - Present & future ; Resources ; Sprint ; Feedback

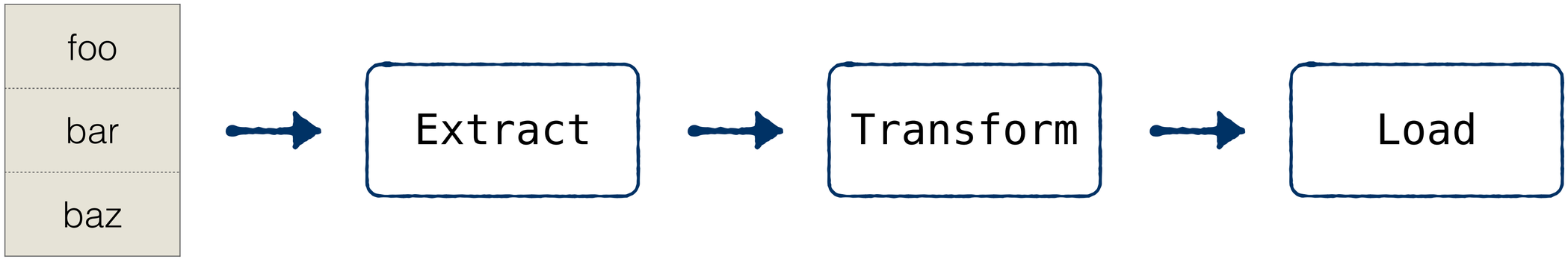

Once upon a time…

Extract Transform Load

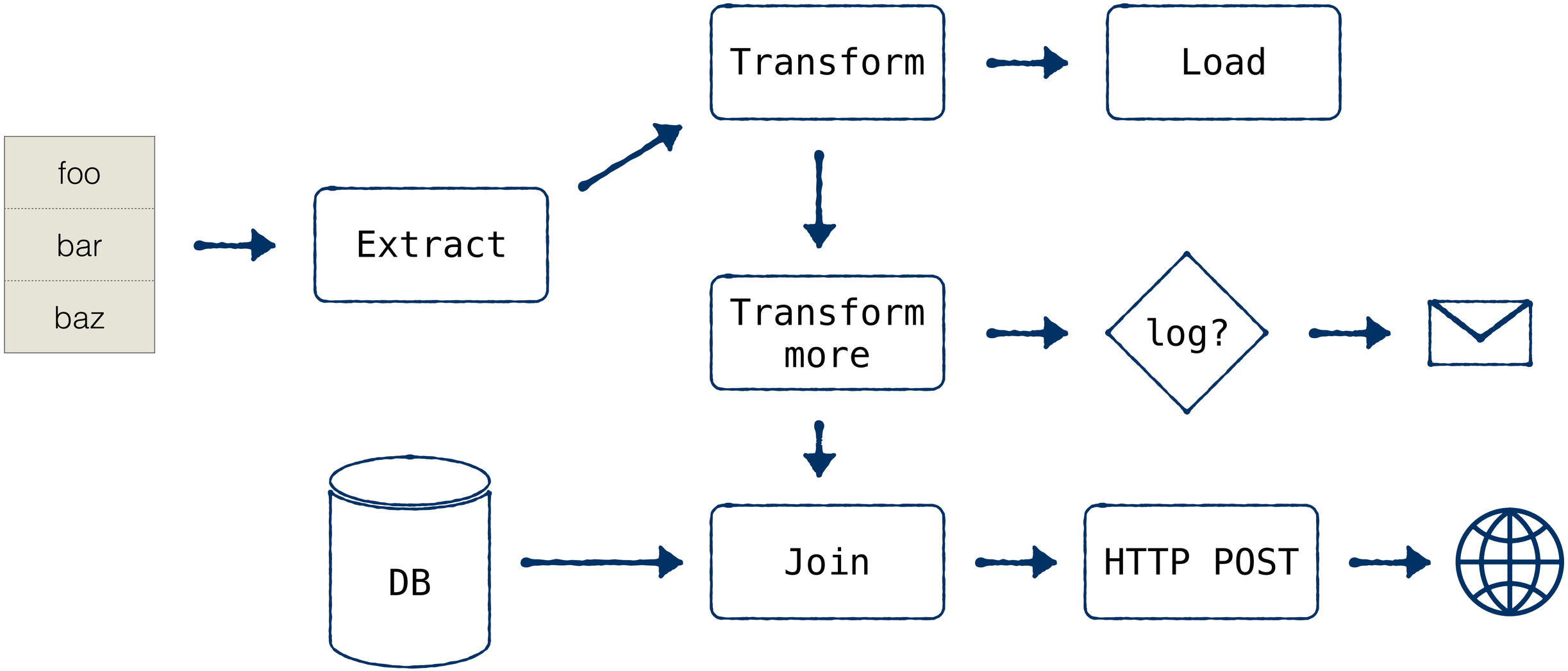
- **Not new.** Popular concept in the 1970s [1] [2]
- **Everywhere.** Commerce, websites, marketing, finance, …

[1] https://en.wikipedia.org/wiki/Extract,_transform,_load
[2] https://www.sas.com/en_us/insights/data-management/what-is-etl.html

Extract Transform Load

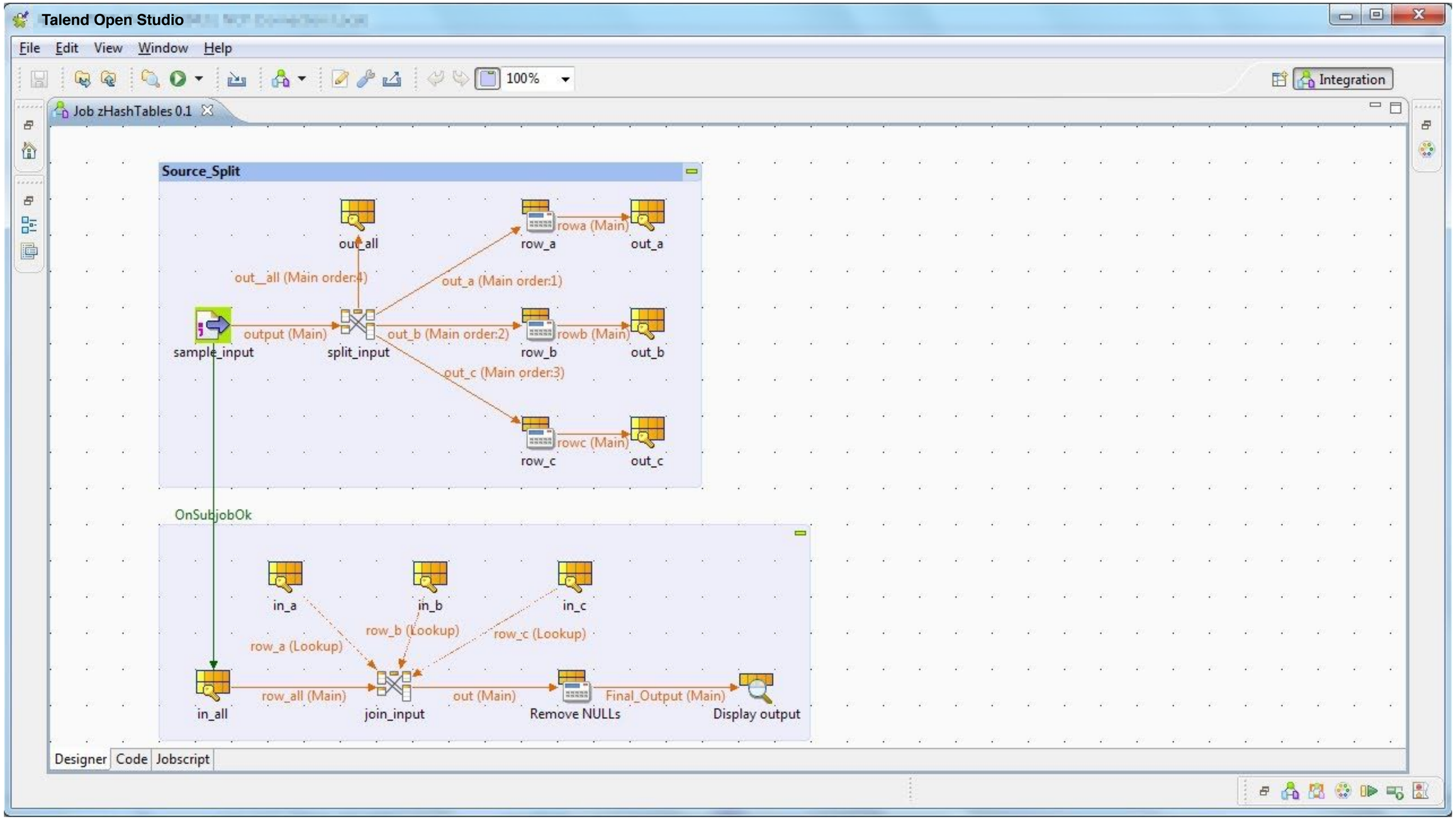


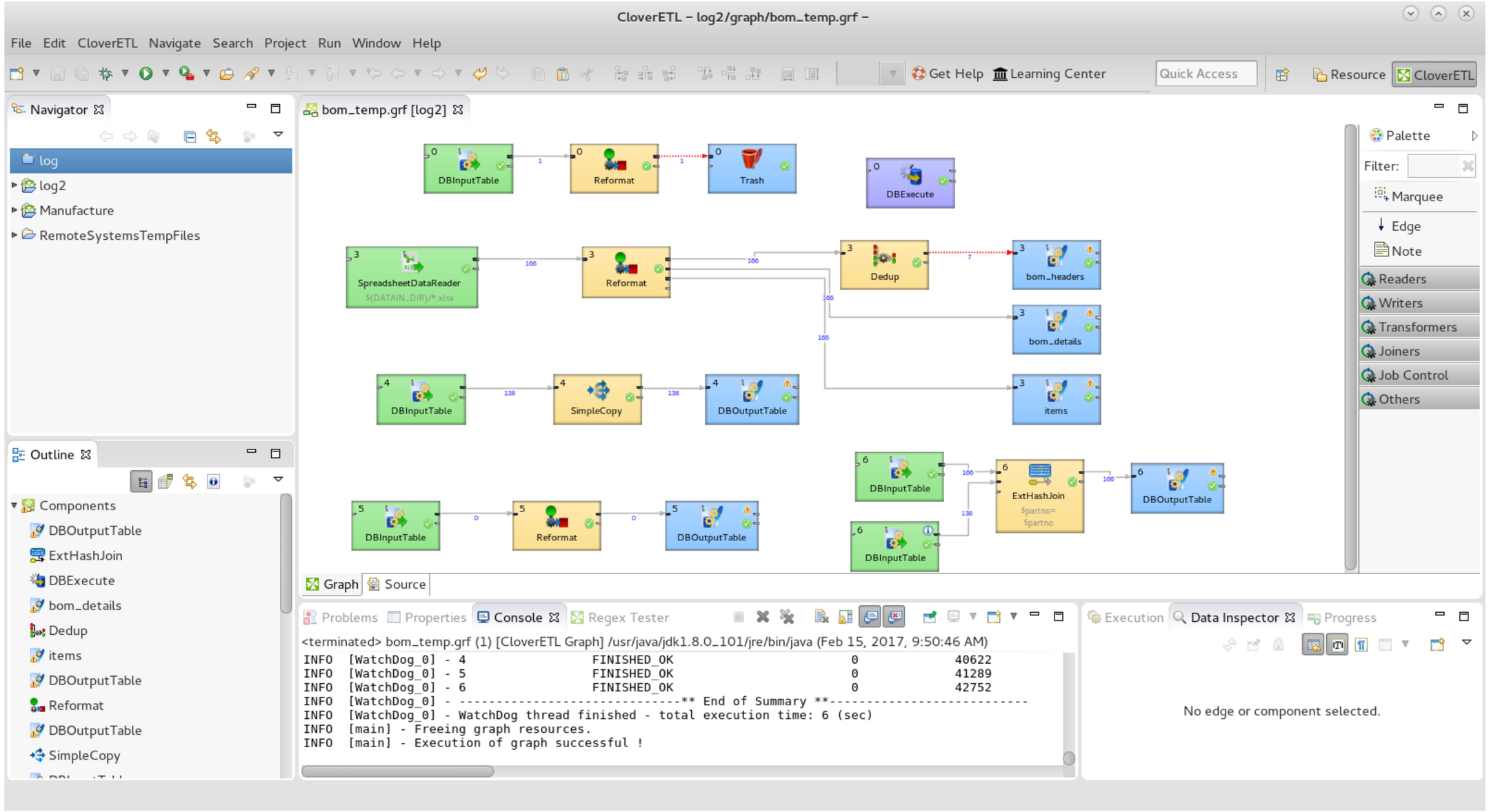
Extract Transform Load

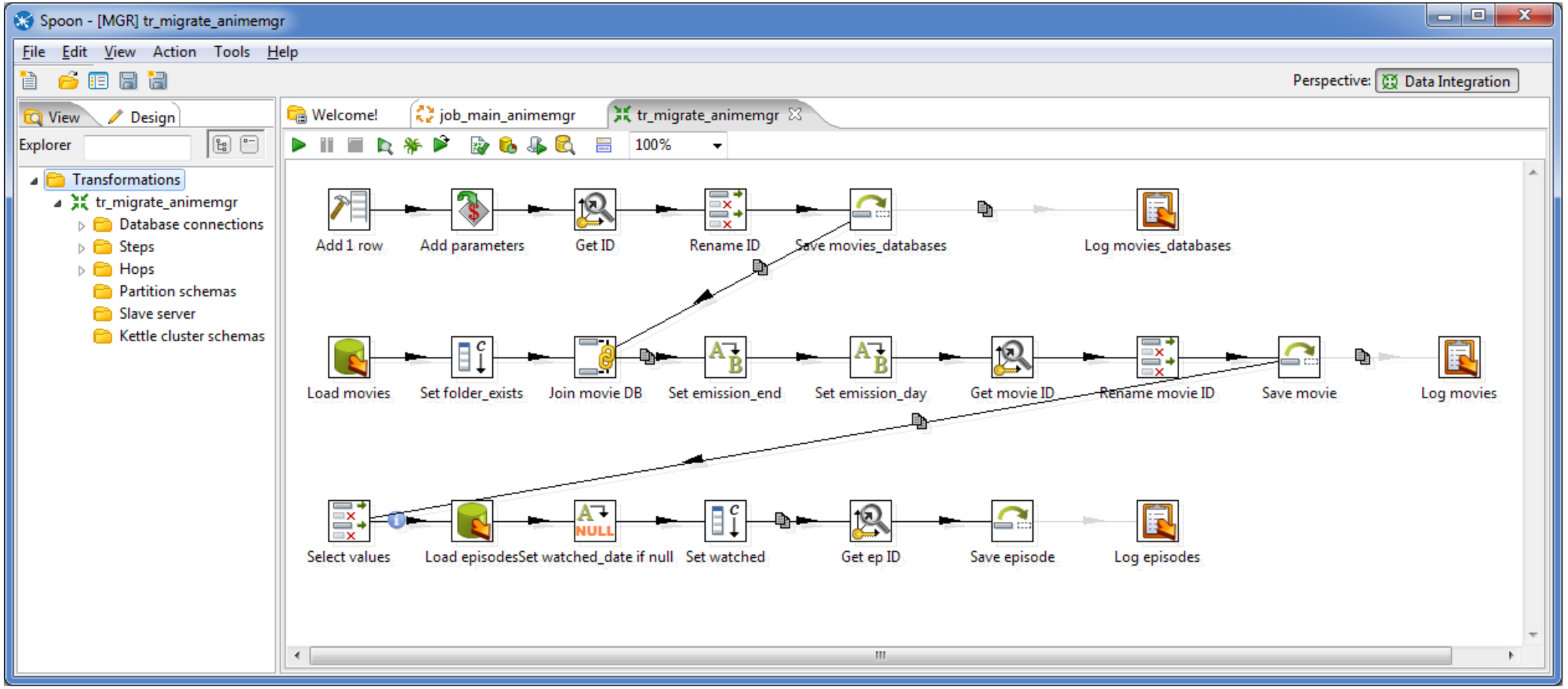


Data Integration Tools

- **Pentaho Data Integration** (IDE/Java)
- **Talend Open Studio** (IDE/Java)
- **CloverETL** (IDE/Java)


Talend Open Studio
File Edit View Window Help
100%
Integration
Job zHashTables 0.1
Source_Split
out_all
row_a
out_a
rowa (Main)
out__all (Main order:4)
out_a (Main order:1)
sample_input
output (Main)
split_input
out_b (Main order:2)
row_b
rowb (Main)
out_b
out_c (Main order:3)
row_c
rowc (Main)
out_c
OnSubjobOk
in_a
in_b
in_c
row_b (Lookup)
row_a (Lookup)
row_c (Lookup)
in_all
row_all (Main)
join_input
out (Main)
Remove NULLs
Final_Output (Main)
Display output
Designer
Code
Jobscript


CloverETL – log2/graph/bom_temp.grf –
File Edit CloverETL Navigate Search Project Run Window Help
Get Help
Learning Center
Quick Access
Resource
CloverETL
Navigator
log
log2
Manufacture
RemoteSystemsTempFiles
bom_temp.grf [log2]
DBInputTable
Reformat
Trash
DBExecute
SpreadsheetDataReader
${DATAIN_DIR}/*.xlsx
Reformat
Dedup
bom_headers
bom_details
items
DBInputTable
SimpleCopy
DBOutputTable
DBInputTable
DBInputTable
ExtHashJoin
$partno=$partno
DBOutputTable
DBInputTable
Reformat
DBOutputTable
Palette
Filter:
Marquee
Edge
Note
Readers
Writers
Transformers
Joiners
Job Control
Others
Graph
Source
Outline
Components
DBOutputTable
ExtHashJoin
DBExecute
bom_details
Dedup
items
DBOutputTable
Reformat
DBOutputTable
SimpleCopy
Problems
Properties
Console
Regex Tester
<terminated> bom_temp.grf (1) [CloverETL Graph] /usr/java/jdk1.8.0_101/jre/bin/java (Feb 15, 2017, 9:50:46 AM)
INFO [WatchDog_0] - 4 FINISHED_OK 0 40622
INFO [WatchDog_0] - 5 FINISHED_OK 0 41289
INFO [WatchDog_0] - 6 FINISHED_OK 0 42752
INFO [WatchDog_0] - ** End of Summary **
INFO [WatchDog_0] - WatchDog thread finished - total execution time: 6 (sec)
INFO [main] - Freeing graph resources.
INFO [main] - Execution of graph successful !
Execution
Data Inspector
Progress
No edge or component selected.


Spoon - [MGR] tr_migrate_animemgr
File Edit View Action Tools Help
Perspective: Data Integration
View Design
Explorer
Transformations
tr_migrate_animemgr
Database connections
Steps
Hops
Partition schemas
Slave server
Kettle cluster schemas
Welcome!
job_main_animemgr
tr_migrate_animemgr
100%
Add 1 row
Add parameters
Get ID
Rename ID
Save movies_databases
Log movies_databases
Load movies
Set folder_exists
Join movie DB
Set emission_end
Set emission_day
Get movie ID
Rename movie ID
Save movie
Log movies
Select values
Load episodes
Set watched_date if null
NULL
Set watched
Get ep ID
Save episode
Log episodes

Data Integration Tools

- Java + IDE based, for most of them
- Data transformations are blocks
- IO flow managed by connections
- Execution

GUI first, eventually code :-(

In the Python world …

- **Bubbles** (https://github.com/stiivi/bubbles)
- **PETL** (https://github.com/alimanfoo/petl)
- (insert a few more here)
- and now… **Bonobo** (https://www.bonobo-project.org/)

You can also use amazing libraries including
Joblib, **Dask**, **Pandas**, **Toolz**,
but ETL is not their main focus.

Other scales…

Small Automation Tools

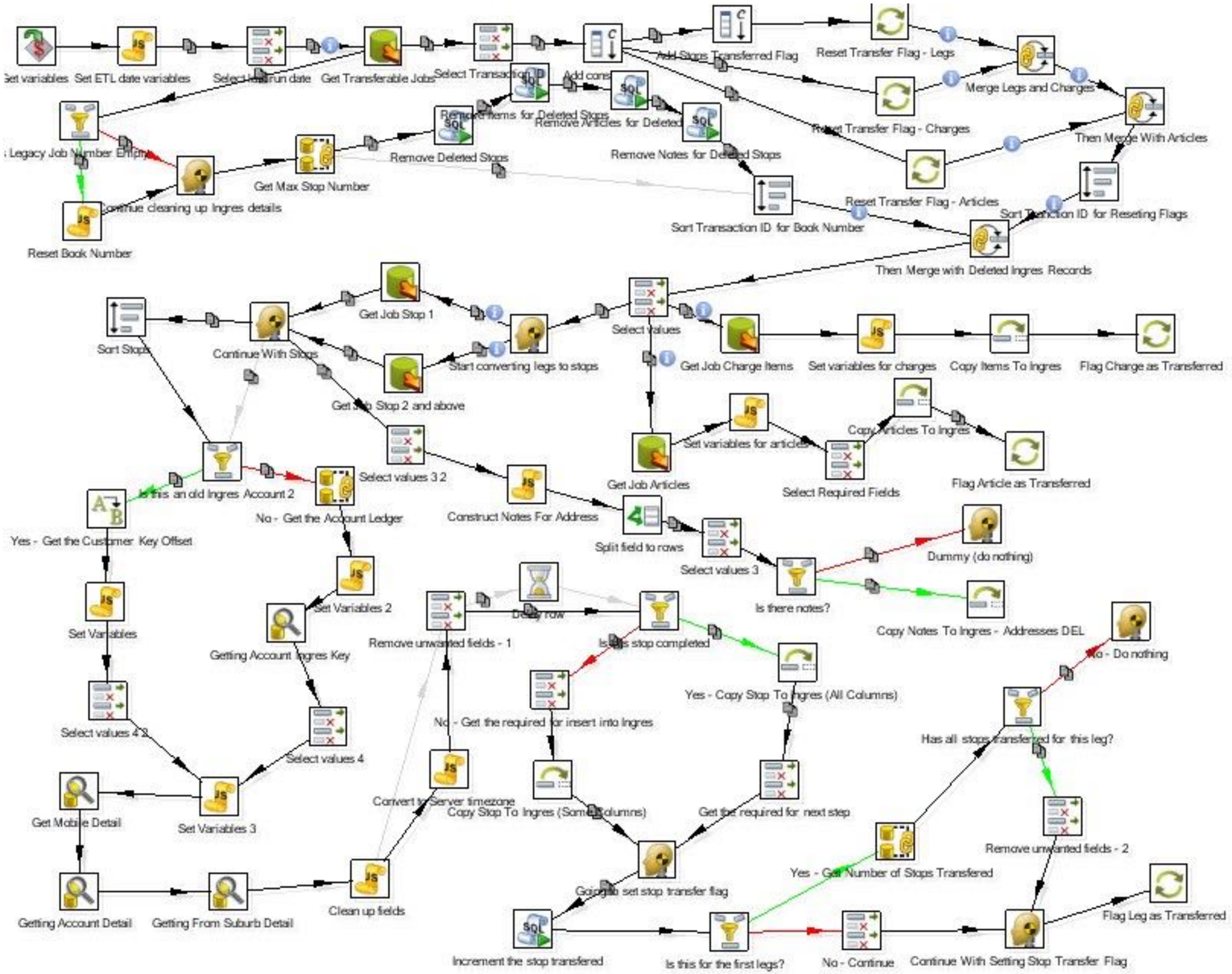
- Mostly aimed at simple recurring tasks.
- Cloud / SaaS only.

Big Data Tools

- Can do anything. And probably more. Fast.
- Either needs an infrastructure, or cloud based.

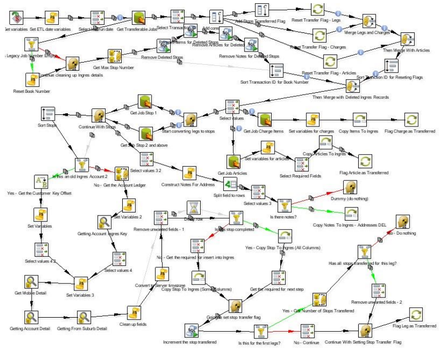
Story time

Partner 1 Data Integration

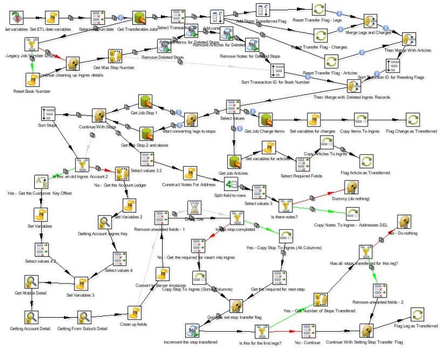



WE GOT DEALS !!!

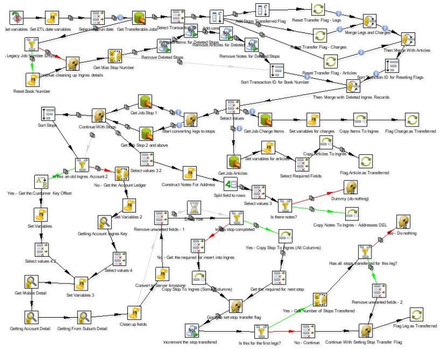
Partner 1



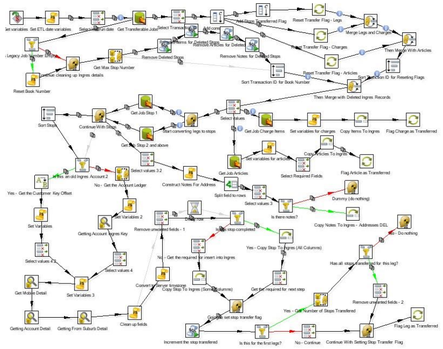
Partner 2



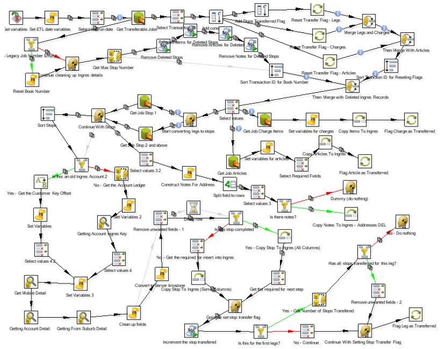
Partner 3



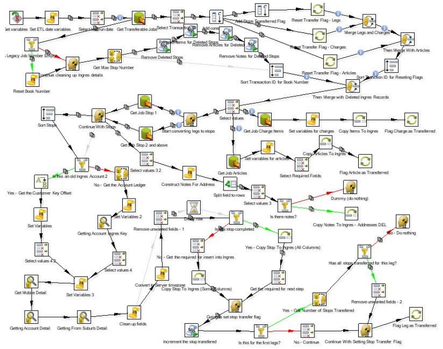
Partner 4



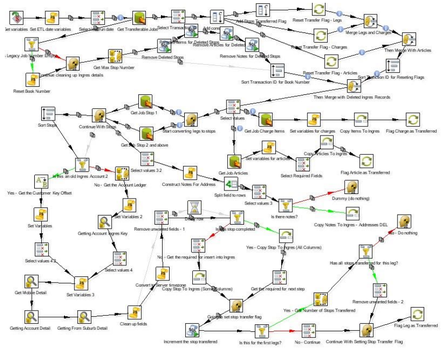
Partner 5



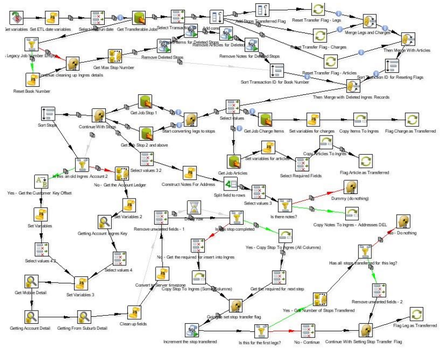
Partner 6



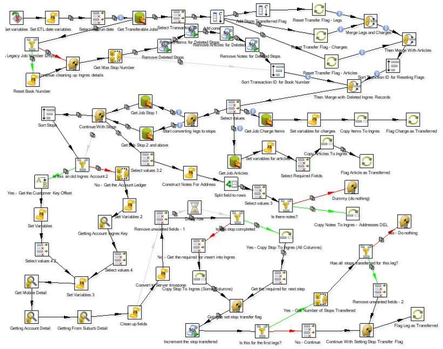
Partner 7



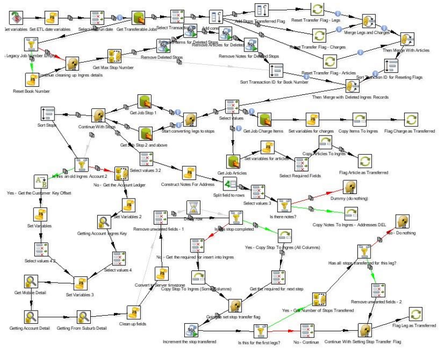
Partner 8

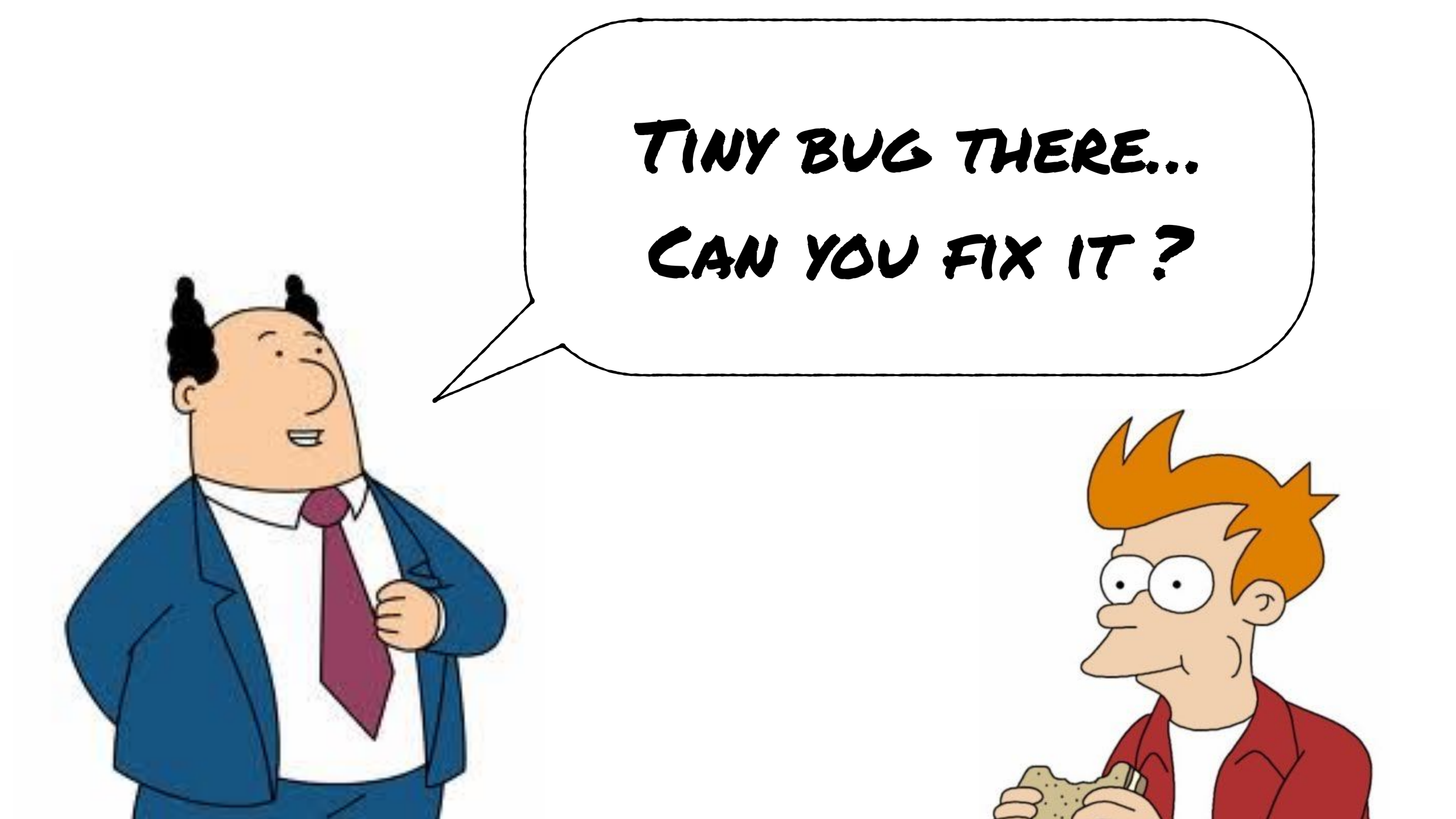


Partner 9



...




Tiny bug there...
Can you fix it ?

My need

- A data integration / ETL tool using **code** as configuration.
- Preferably **Python** code.
- Something that can be **tested** (I mean, by a machine).
- Something that can use **inheritance**.
- Fast & cheap **install on laptop**, thought for **servers** too.

And that’s Bonobo

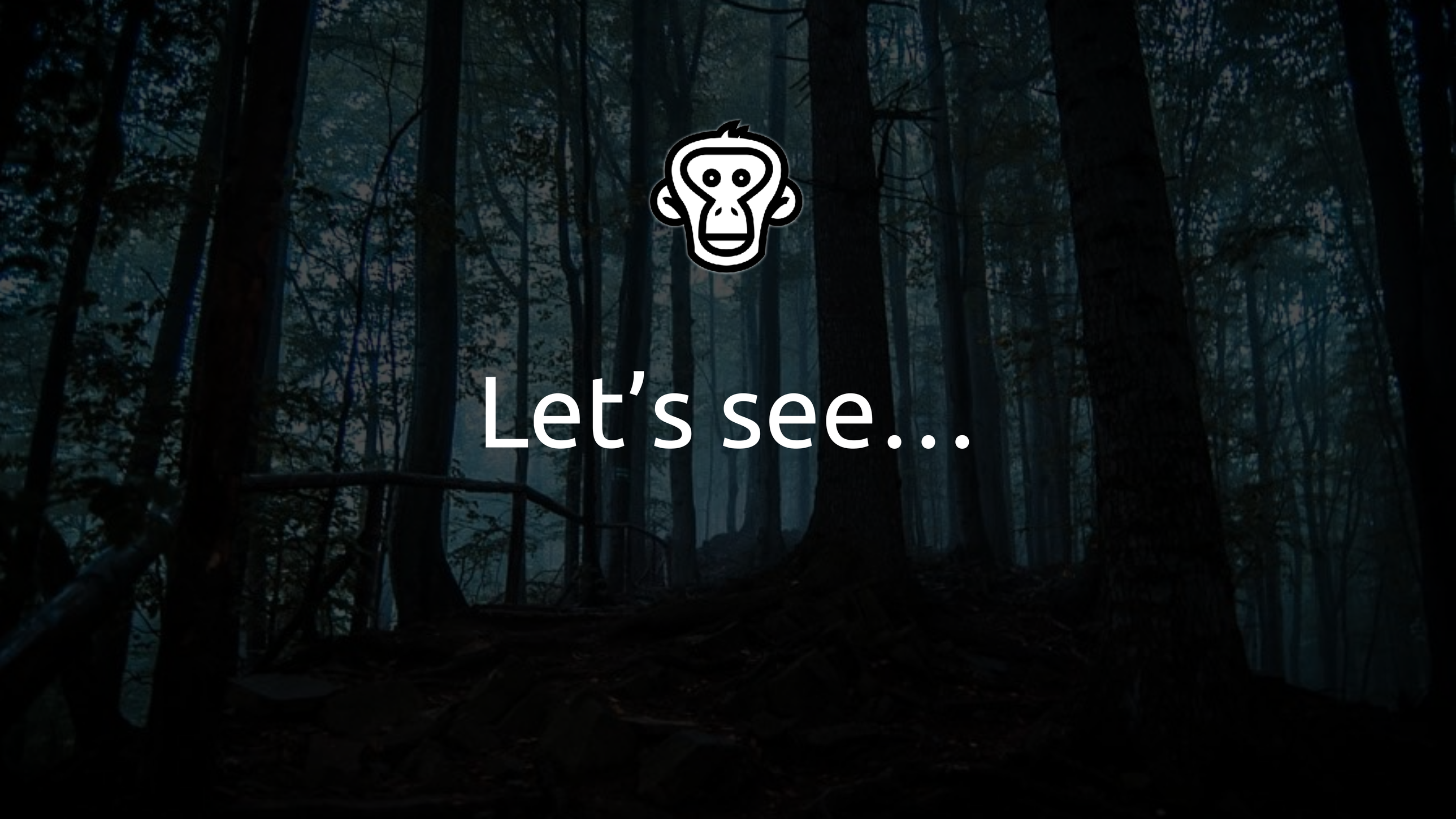
It is …

- A framework to **write ETL jobs in Python 3** (3.5+)
- Using the same concepts as the old ETLs.
- You can use OOP!

`Code first. Eventually a GUI will come.`

It is NOT …

- Pandas / R Dataframes
- Dask (but will probably implement a dask.distributed strategy someday)
- Luigi / Airflow
- Hadoop / Big Data / Big Query / …
- A monkey (spoiler : **it's an ape**, damnit french language…)

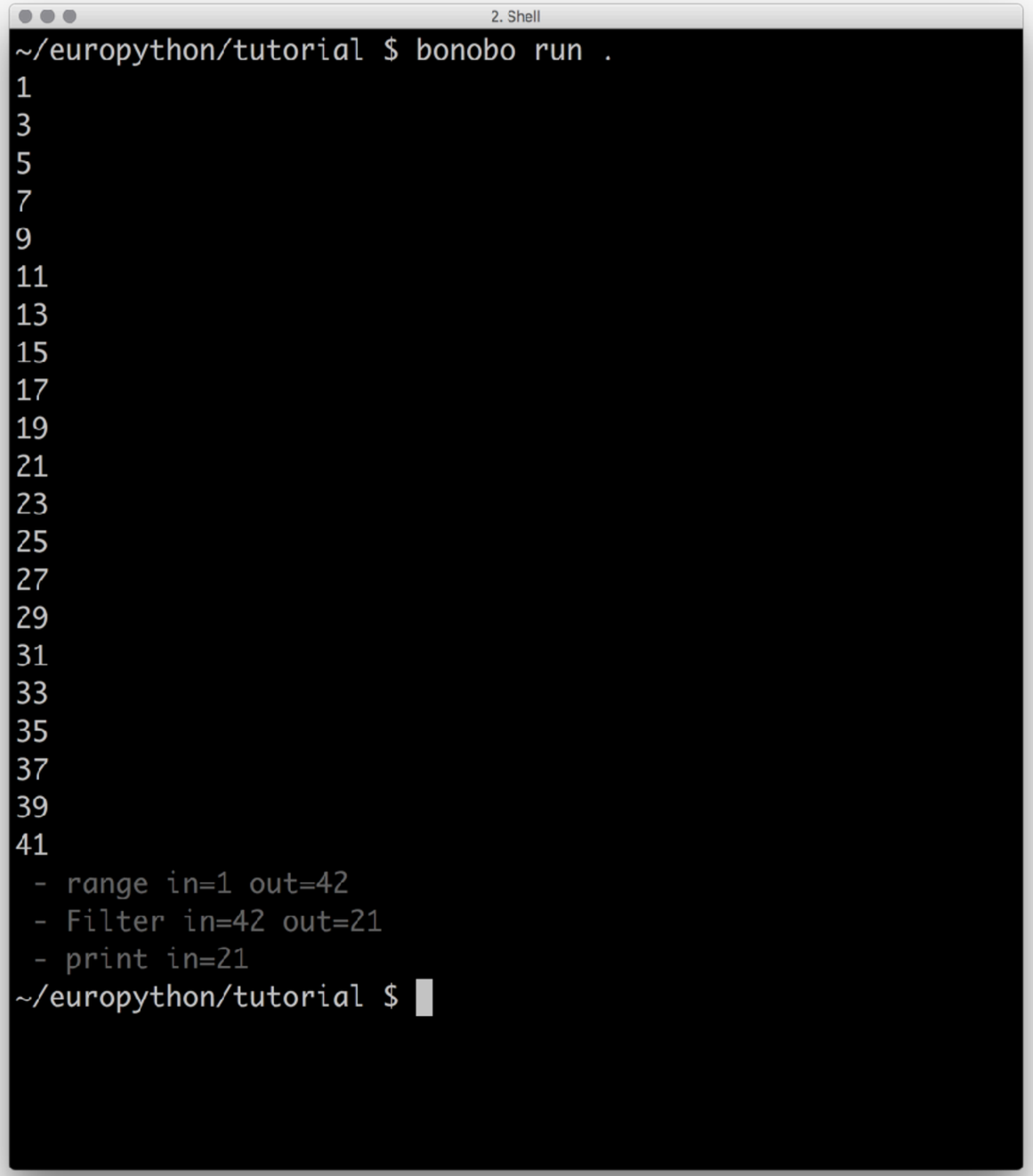

Let’s see…

Create a project

```
~ $ pip install bonobo

~ $ bonobo init europython/tutorial

~ $ bonobo run europython/tutorial
```

…demo

```
~ $ bonobo run .
```



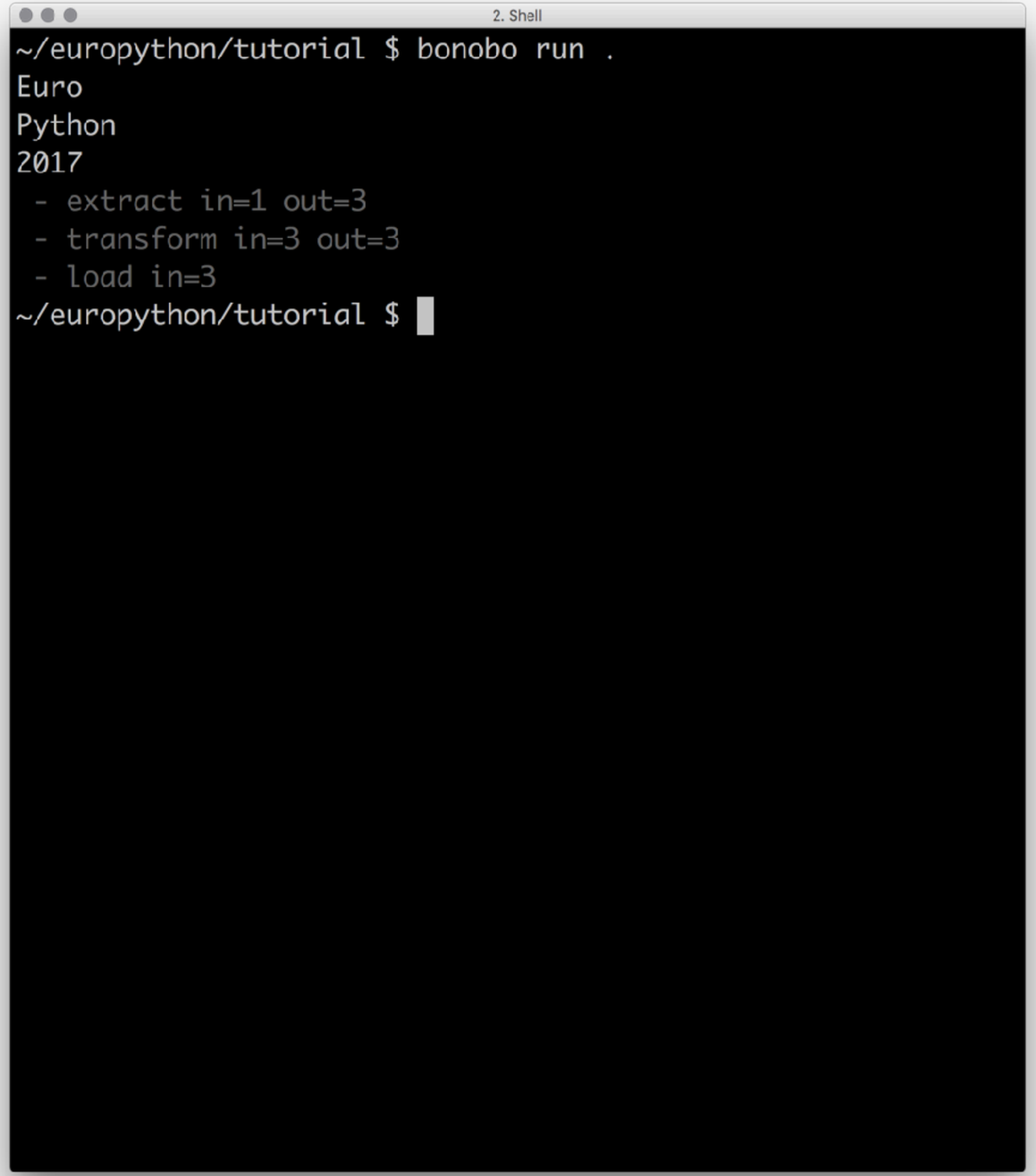
TEMPLATE

Write our own

```
import bonobo


def extract():
    yield 'euro'
    yield 'python'
    yield '2017'


def transform(s):
    return s.title()


def load(s):
    print(s)


graph = bonobo.Graph(
    extract,
    transform,
    load,
)
```

…demo

```
~ $ bonobo run .
```



EXAMPLE_1

…demo

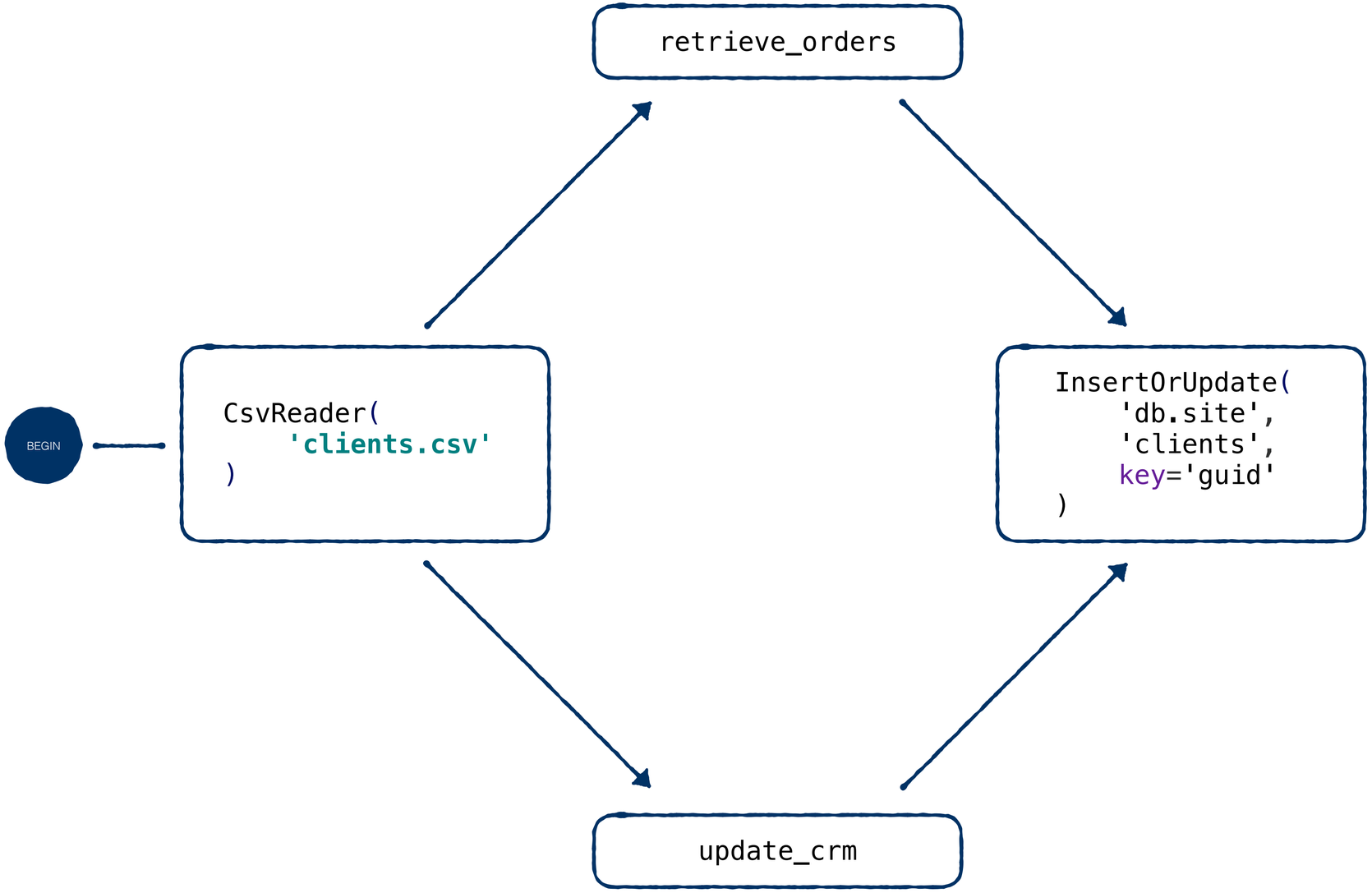
```
~ $ bonobo run first.py
```

EXAMPLE_1

```
2. Shell
  attendees = 456
• name = EuroPython 2008
  venue = Vilnius, Lithuania
  dates = July 7 - 12 2008.
  attendees = 206
• name = EuroPython 2007
  venue = Vilnius, Lithuania
  dates = July 9 - 11 2007.
  attendees = None
• name = EuroPython 2006
  venue = CERN, Geneva, Switzerland
  dates = July 3 - 5 2006.
  attendees = None
• name = EuroPython 2005
  venue = Göteborg, Sweden
  dates = June 27 - July 3 2005.
  attendees = None
• name = EuroPython 2004
  venue = Göteborg, Sweden
  dates = June 7 - 9 2004.
  attendees = None
• name = EuroPython 2003
  venue = Charleroi, Belgium
  dates = Jun 25 - 27 2003.
  attendees = 300
 - extract in=1 out=15
 - transform in=15 out=15
 - arg0_to_kwargs in=15 out=15
 - PrettyPrinter in=15
~/europython/tutorial $
```

Under the hood…

```
graph = bonobo.Graph(…)
```


retrieve_orders
BEGIN
CsvReader(
'clients.csv'
)
InsertOrUpdate(
'db.site',
'clients',
key='guid'
)
update_crm

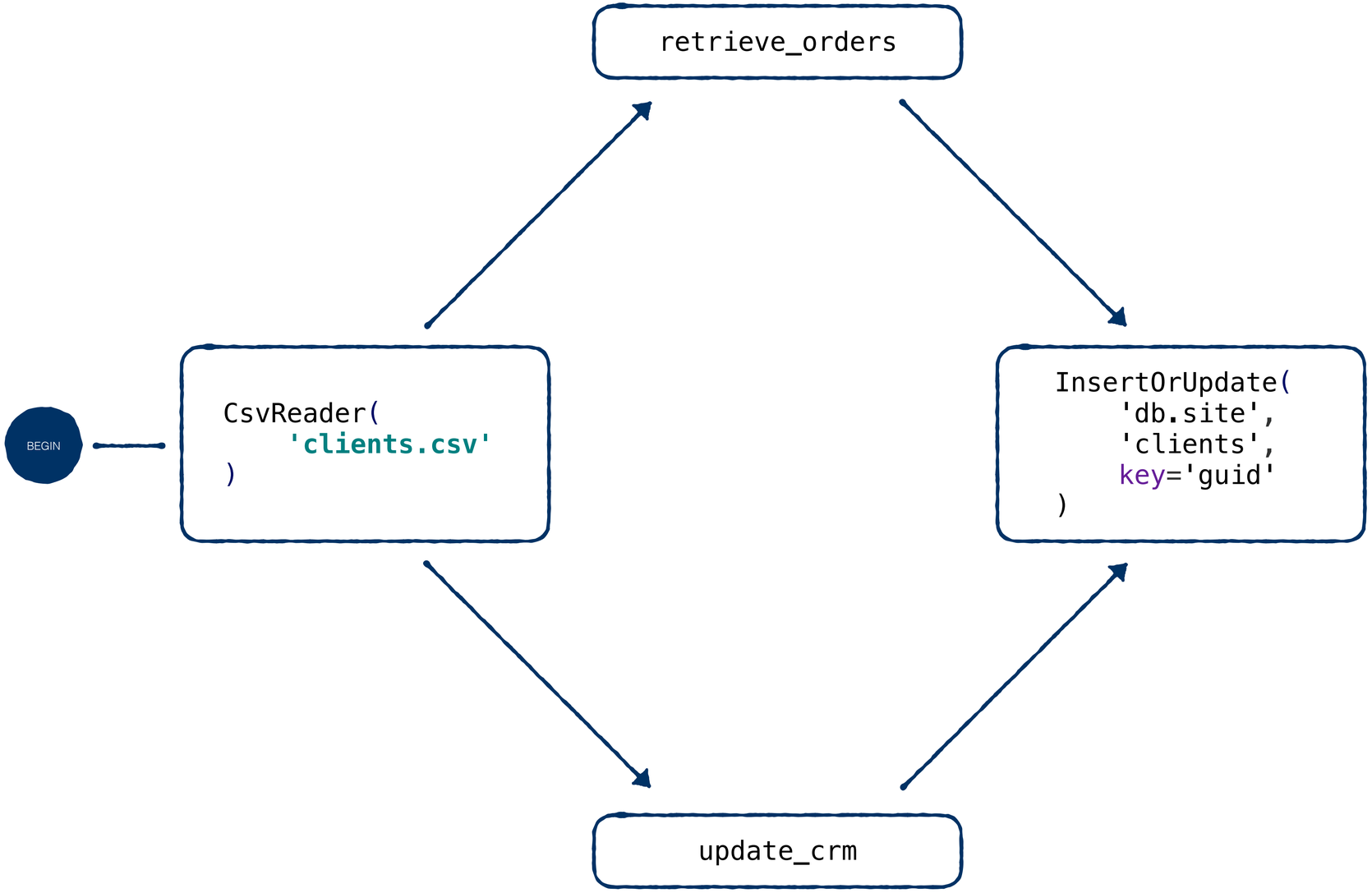
Graph…

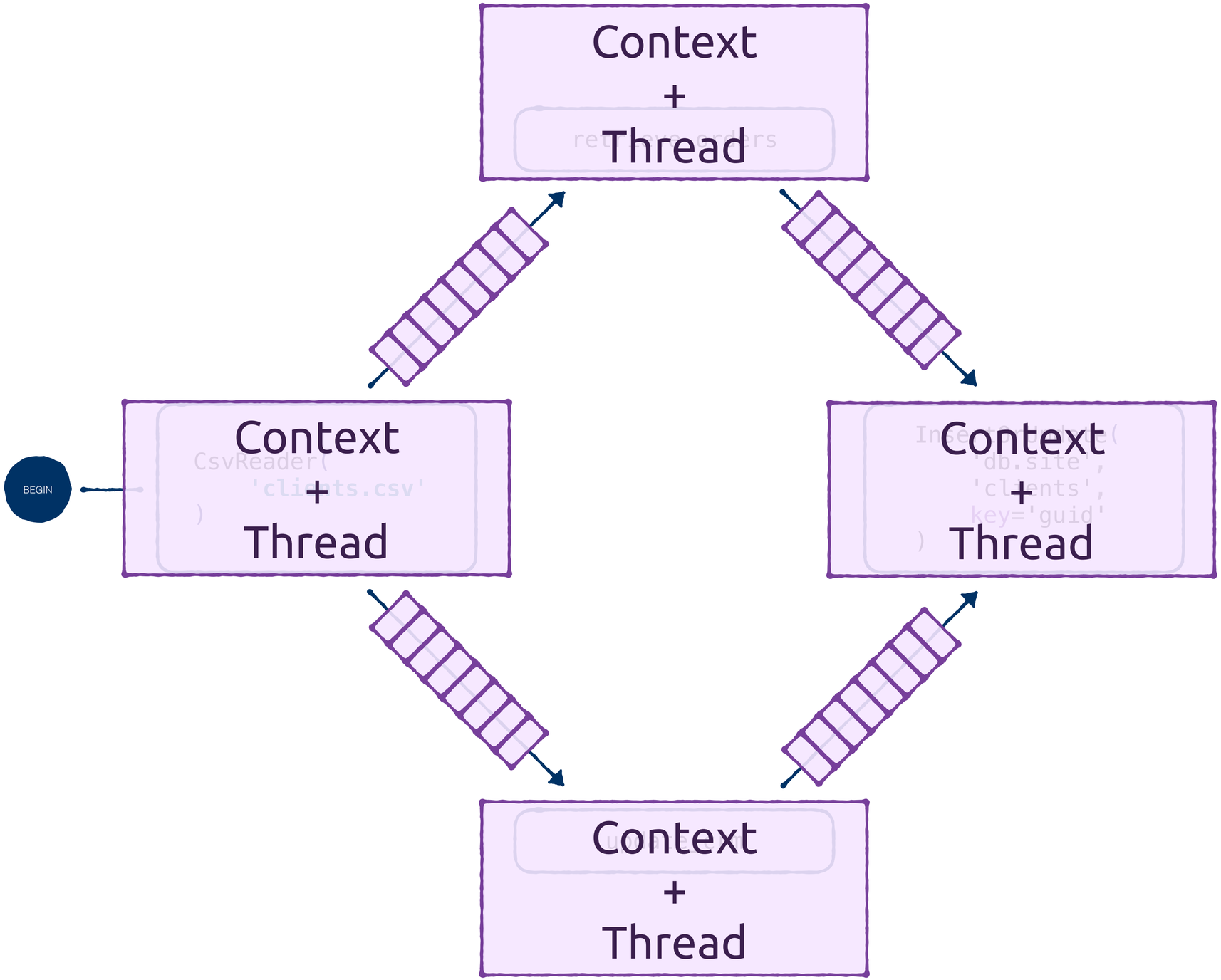
```
class Graph:

    def __init__(self, *chain):
        self.edges = {}
        self.nodes = []

        self.add_chain(*chain)

    def add_chain(self, *nodes, _input=None, _output=None):
        # ...
```

```
bonobo.run(graph)
```

or in a shell…

```
$ bonobo run main.py
```


retrieve_orders
BEGIN
CsvReader(
'clients.csv'
)
InsertOrUpdate(
'db.site',
'clients',
key='guid'
)
update_crm


Context
+
Thread
Context
+
Thread
Context
+
Thread
Context
+
Thread
BEGIN

Context…

```
class GraphExecutionContext:
    def __init__(self, graph, plugins, services):
        self.graph = graph
        self.nodes = [
            NodeExecutionContext(node, parent=self)
            for node in self.graph
        ]
        self.plugins = [
            PluginExecutionContext(plugin, parent=self)
            for plugin in plugins
        ]
        self.services = services
```

Strategy…

```
class ThreadPoolExecutorStrategy(Strategy):
    def execute(self, graph, plugins, services):
        context = self.create_context(graph, plugins, services)
        executor = self.create_executor()

        for node_context in context.nodes:
            executor.submit(
                self.create_runner(node_context)
            )

        while context.alive:
            self.sleep()

        executor.shutdown()

        return context
```

</ implementation details >

Transformations

a.k.a

nodes in the graph

Functions

```
def get_more_infos(api, **row):
    more = api.query(row.get('id'))

    return {
        **row,
        **(more or {}),
    }
```

Generators

```
def join_orders(order_api, **row):
    for order in order_api.get(row.get('customer_id')):
        yield {
            **row,
            **order,
        }
```

Iterators

```
extract = (
    'foo',
    'bar',
    'baz',
)

extract = range(0, 1001, 7)
```

Classes

```
class RiminizeThis:
    def __call__(self, **row):
        return {
            **row,
            'Rimini': 'Woo-hou-wo...',
        }
```

Anything, as long as it’s callable().

Configurable classes

```
from bonobo.config import Configurable, Option, Service

class QueryDatabase(Configurable):

    table_name = Option(str, default='customers')

    database = Service('database.default')

    def call(self, database, **row):
        customer = database.query(self.table_name, customer_id=row['clie
        return {
            **row,
            'is_customer': bool(customer),
        }
```

Configurable classes

```
from bonobo.config import Configurable, Option, Service

class QueryDatabase(Configurable):

    table_name = Option(str, default='customers')

    database = Service('database.default')

    def call(self, database, **row):
        customer = database.query(self.table_name, customer_id=row['clie
        return {
            **row,
            'is_customer': bool(customer),
        }
```

Configurable classes

```
from bonobo.config import Configurable, Option, Service

class QueryDatabase(Configurable):

    table_name = Option(str, default='customers')

    database = Service('database.default')

    def call(self, database, **row):
        customer = database.query(self.table_name, customer_id=row['clie
        return {
            **row,
            'is_customer': bool(customer),
        }
```

Configurable classes

```
from bonobo.config import Configurable, Option, Service

class QueryDatabase(Configurable):

    table_name = Option(str, default='customers')

    database = Service('database.default')

    def call(self, database, **row):
        customer = database.query(self.table_name, customer_id=row['clie
        return {
            **row,
            'is_customer': bool(customer),
        }
```

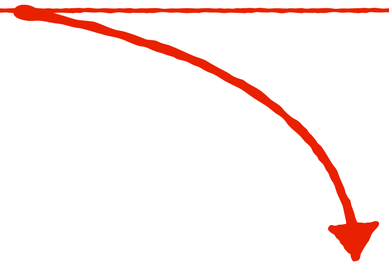
Configurable classes

```
query_database = QueryDatabase(
    table_name='test_customers',
    database='database.testing',
)
```

Services

Define as names

```
class QueryDatabase(Configurable):
    database = Service('database.default')

    def call(self, database, **row):
        return { … }
```



Runtime injection

```
import bonobo

graph = bonobo.Graph(...)

def get_services():
    return {
        'database.default': MyDatabaseImpl()
    }
```

Bananas!

Library

```
bonobo.FileReader(…)
bonobo.CsvReader(…)
bonobo.JsonReader(…)
bonobo.PickleReader(…)

bonobo.ExcelReader(…)
bonobo.XMLReader(…)

… more to come
```

```
bonobo.FileWriter(…)
bonobo.CsvWriter(…)
bonobo.JsonWriter(…)
bonobo.PickleWriter(…)

bonobo.ExcelWriter(…)
bonobo.XMLWriter(…)

… more to come
```

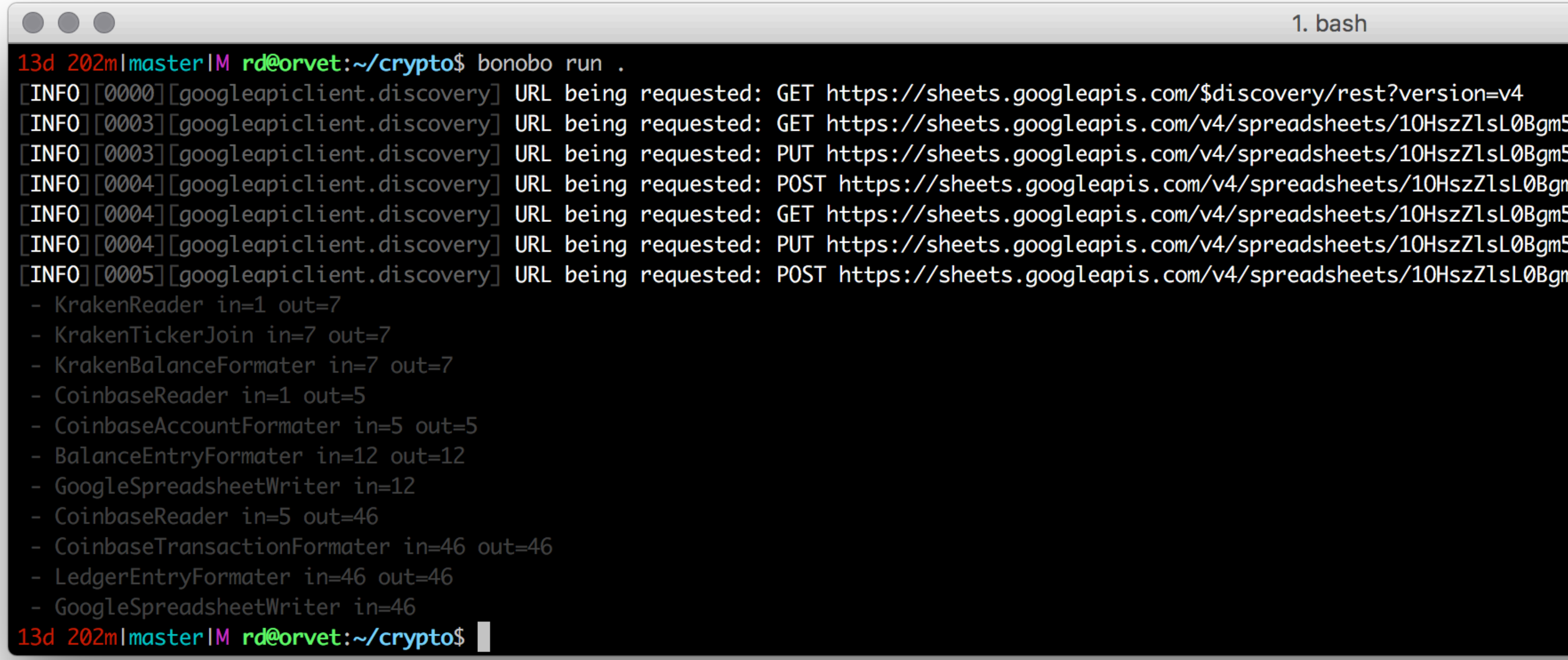
Library

```
bonobo.Limit(limit)
bonobo.PrettyPrinter()
bonobo.Filter(…)
```

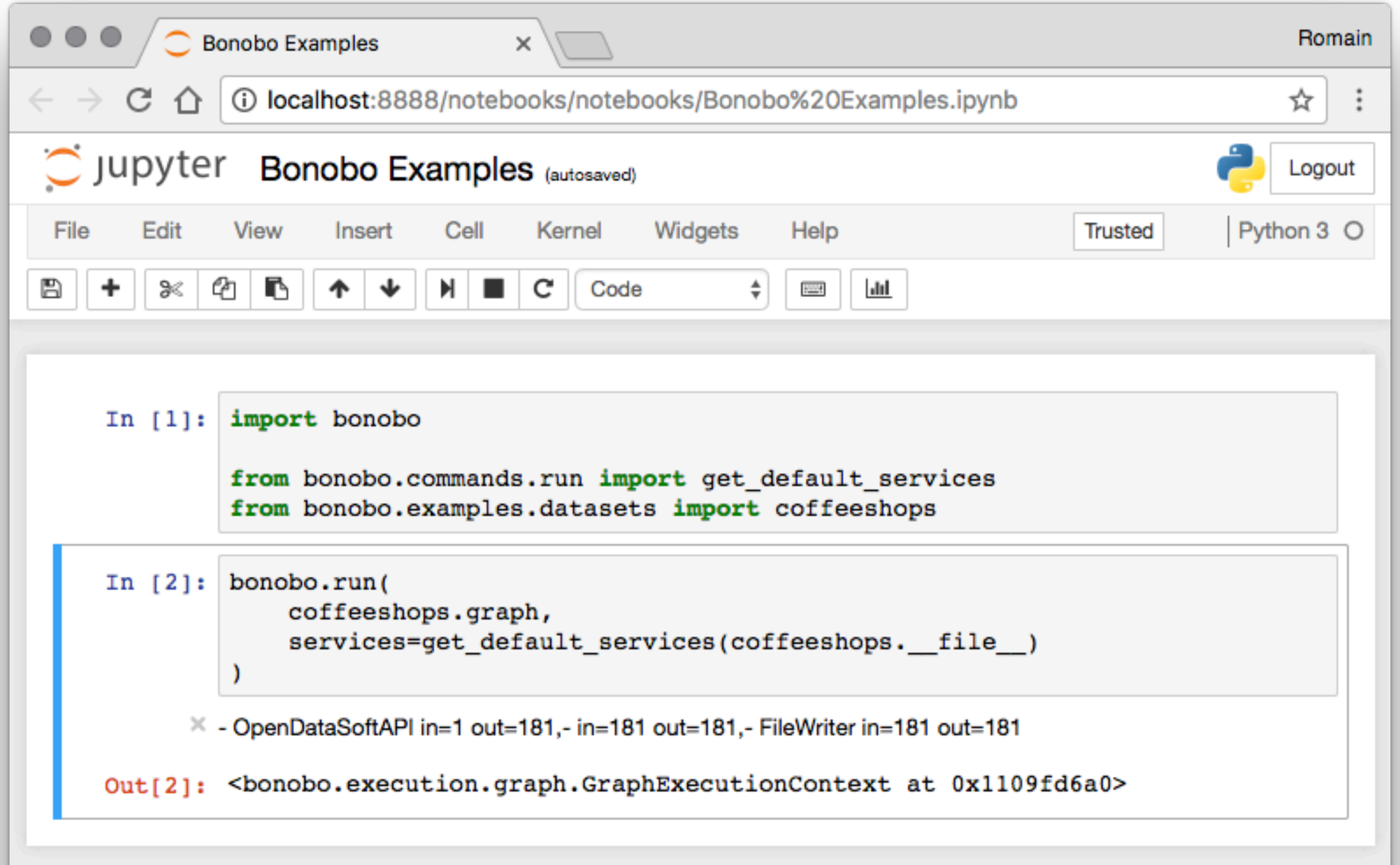
… more to come

Extensions & Plugins

Console Plugin



Jupyter Plugin



SQLAlchemy Extension

PREVIEW

```
bonobo_sqlalchemy.Select(
    query,
    *,
    pack_size=1000,
    limit=None
)

bonobo_sqlalchemy.InsertOrUpdate(
    table_name,
    *,
    fetch_columns,
    insert_only_fields,
    discriminant,
    …
)
```

Docker Extension

PREVIEW

```
$ pip install bonobo[docker]

$ bonobo runc myjob.py
```

Dev Kit

PREVIEW

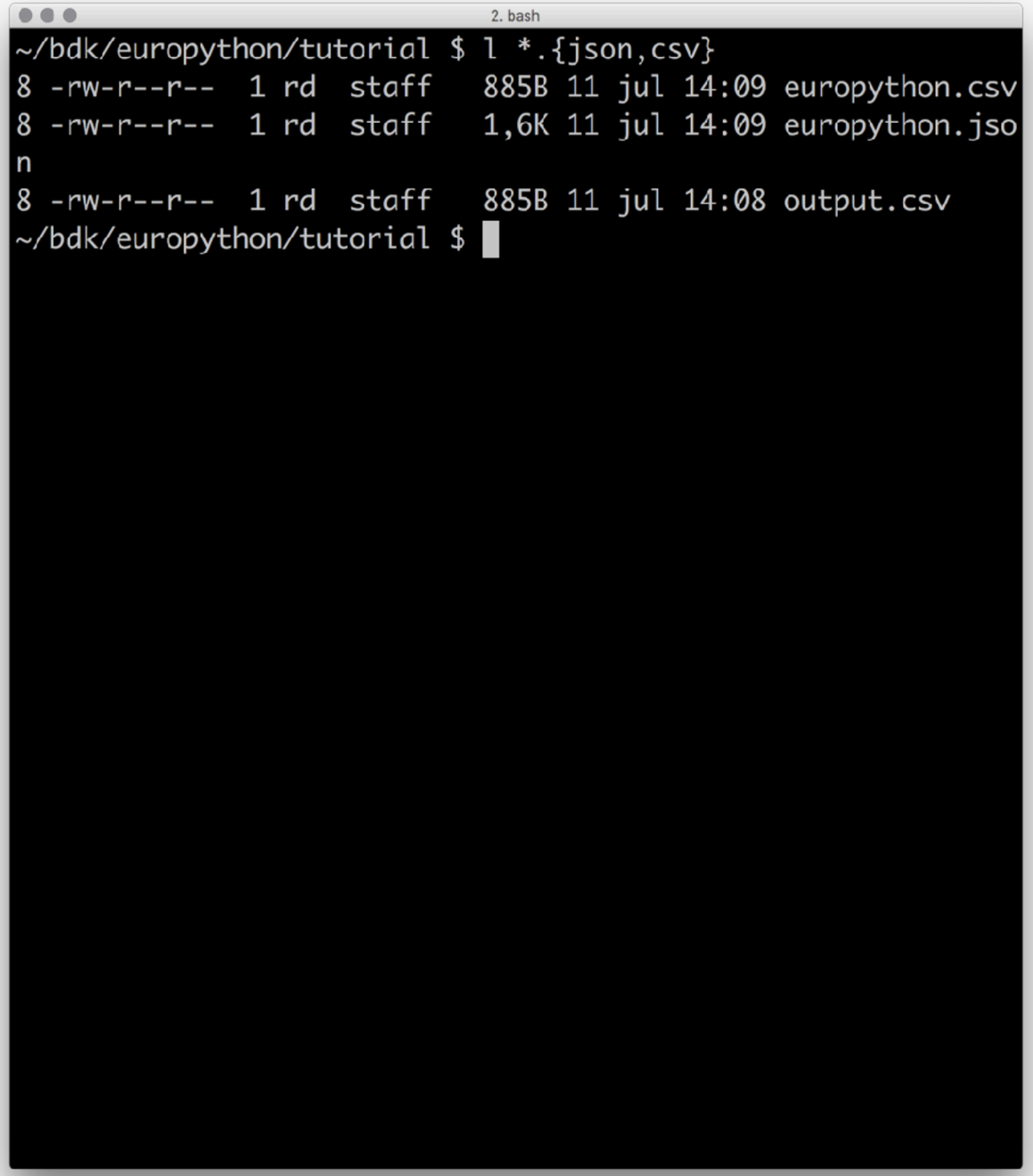
https://github.com/python-bonobo/bonobo-devkit


More examples
?

…demo

- Use filesystem service.
- Write to a CSV
- Also write to JSON

EXAMPLE_1 -> EXAMPLE_2



Rimini open data

EXAMPLE_3

Europython attendees

featuring…
jupyter notebook
selenium & firefox

~/bdk/demos/europython2017

French companies registry

featuring…

docker

postgresql

sql alchemy

~/bdk/demos/sirene

Wrap up

Young

- First commit : December 2016
- 23 releases, ~420 commits, 4 contributors
- Current « stable » 0.4.3
- Target : 1.0 early 2018

Python 3.5+

- {**}
- async/await
- (…, *, …)
- GIL :(

1.0

- 100% Open-Source.
- Light & Focused.
- Very few dependencies.
- Comprehensive standard library.
- The rest goes to plugins and extensions.

Small scale

- 1 minute to install
- Easy to deploy

- **NOT :** Big Data, Statistics, Analytics …
- **IS :** Lean manufacturing for data

Interwebs are crazy

spangry 50 days ago [-]

I haven't tried this yet, but am praying that it delivers even half of what it promises.

▲ Nydhal 50 days ago [-]

You mean excel?

▲ rkda 50 days ago [-]

All those references to monkeys hurt my head. Bonobos are not monkeys.

▲ nn3 50 days ago [-]

The picture looks more like a Gorilla than a Bonobo too

Vue d'ensemble

Temps réel

166

utilisateurs actifs sur le site

MOBILE ORDINATEUR TABLETTE

54% 41% 5%

[–] **glebaron** 14 points 1 month ago

Bonobos are apes, not monkeys. In some situations this distinction is very important.

permalink embed pocket buffer

[–] **alcalde** 9 points 1 month ago

Not here. Python not only has duck typing; it has the little known primate typing feature.

permalink embed parent pocket buffer

Data Processing for Humans

www.bonobo-project.org

docs.bonobo-project.org

bonobo-slack.herokuapp.com

github.com/python-bonobo

Let me know what you think!

Sprint

- Sprints at Europython are amazing
- Nice place to learn about Bonobo, basics, etc.
- Nice place to contribute while learning.
- You’re amazing.

Thank you!

@monkcage

@rdorgueil

https://goo.gl/e25eoa

bonobo

@monkcage