

AUTO PROCESSES

SELEZNEV ARTEM
DS TEAM LEADER @ SBER

НА СЕГОДНЯ

- **Ответы на вопросы**

НА СЕГОДНЯ

- **Ответы на вопросы**

- **Bonobo**



НА СЕГОДНЯ

- **Ответы на вопросы**
- **Bonobo**
- **AirFlow**



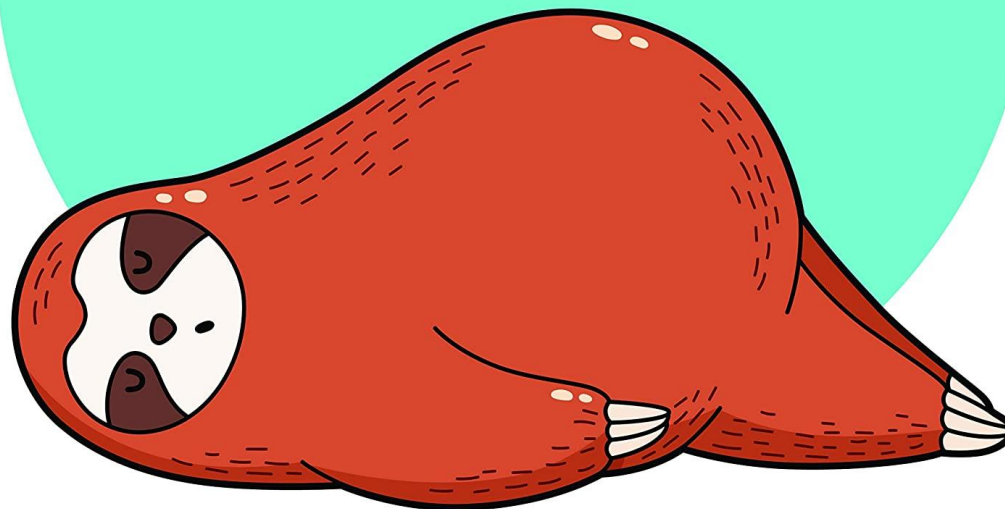
НА СЕГОДНЯ

- **Ответы на вопросы**
- **Bonobo**
- **AirFlow**
- **AirFlow + Kedro**



ЗАПОМНИМ

JUST
RELAX



ЗАПОМНИМ

Не всегда нужен сложный инструмент!



ЗАПОМНИМ

KISS:

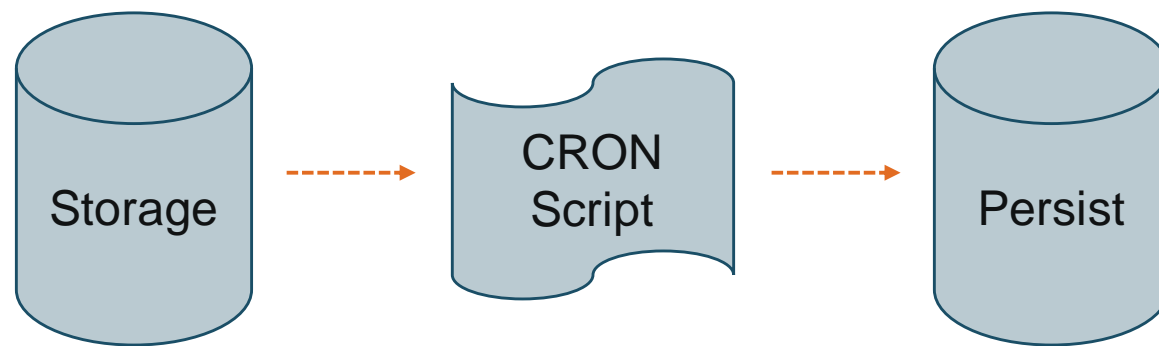
- не стоит подключать огромную библиотеку, если вам от неё нужна лишь пара функций



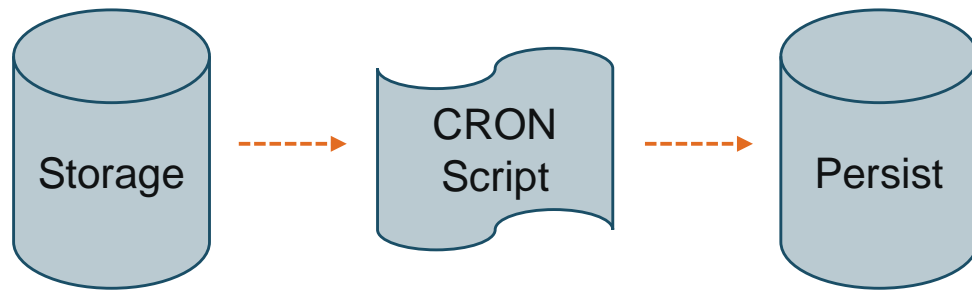
ETL



ETL



ETL



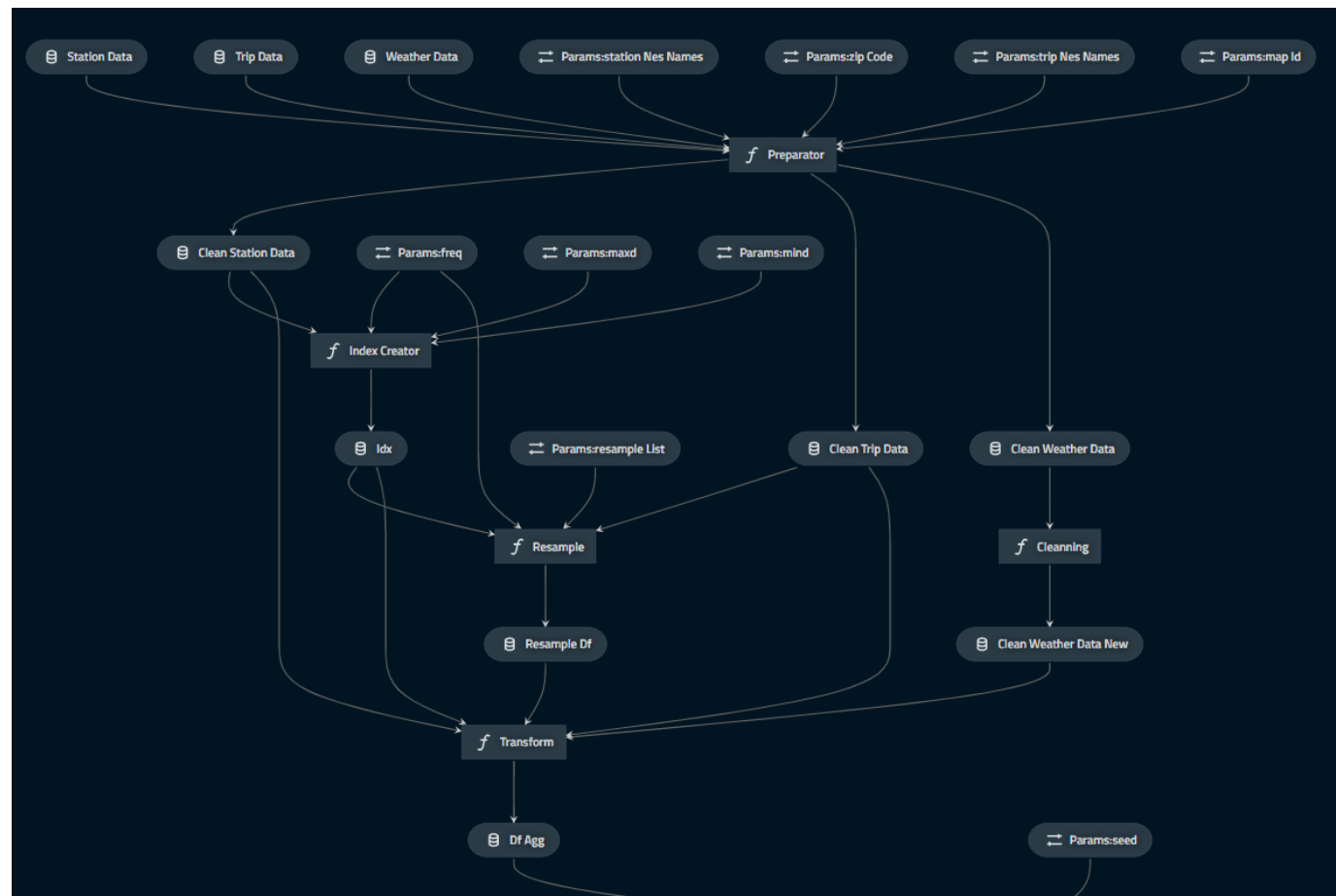
```
def create_pipeline(**kwargs):
    return Pipeline(
        [
            node(
                func=preparator,
                inputs = ["station_data", "trip_data", "weather_data",
                        "params:station_nes_names", "params:zip_code",
                        "params:trip_nes_names", "params:map_id"],
                outputs = ["clean_station_data",
                        "clean_trip_data",
                        "clean_weather_data"]
            ),
            ...
            node(
                split,
                ["df_agg", "params:seed"],
                ["train_x", "train_y", "test_x", "test_y"]
            )
        ]
    )
```

ETL ПРОБЛЕМЫ

- Масштабируемость
- Работа с «неудачными»
выполнениям
- Мониторинг
- Зависимости
- Сохранение историчности

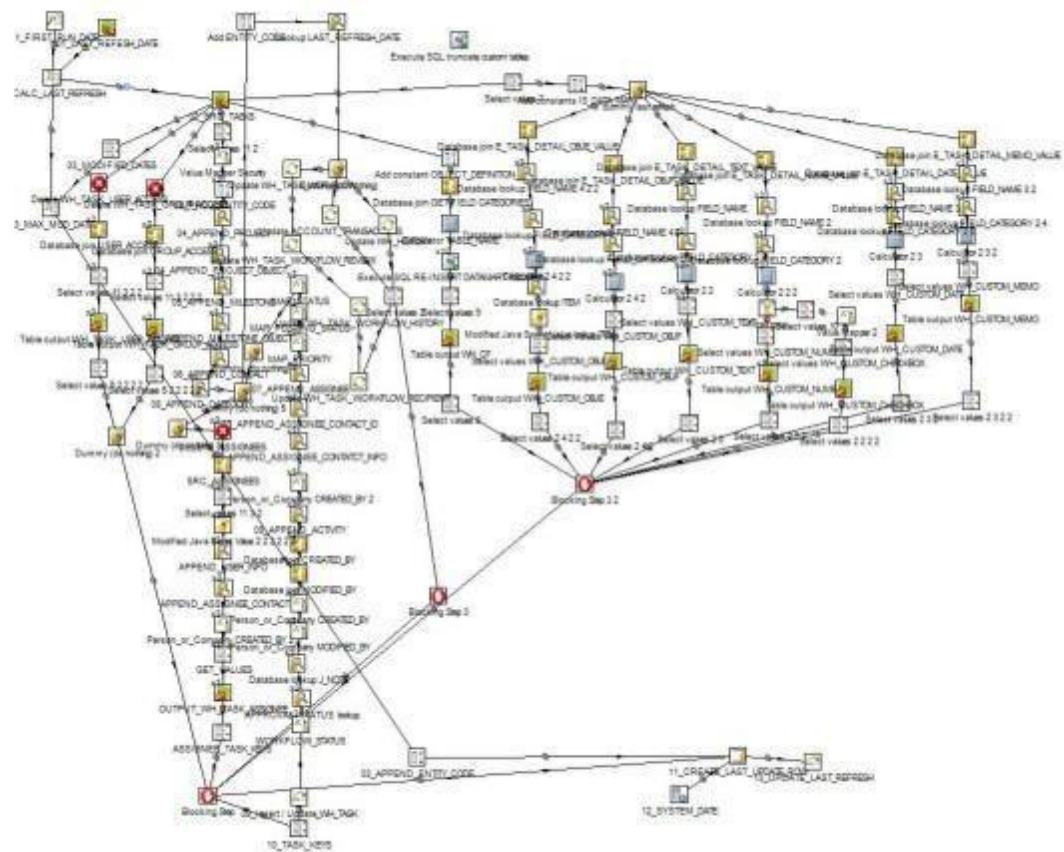
ETL ПРОБЛЕМЫ

- Масштабируемость



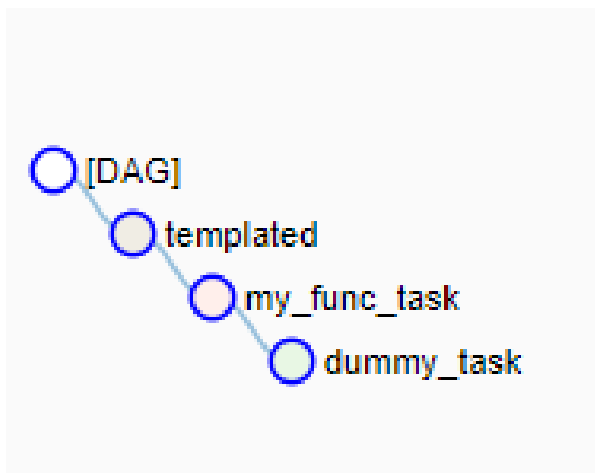
ETL ПРОБЛЕМЫ

- Масштабируемость



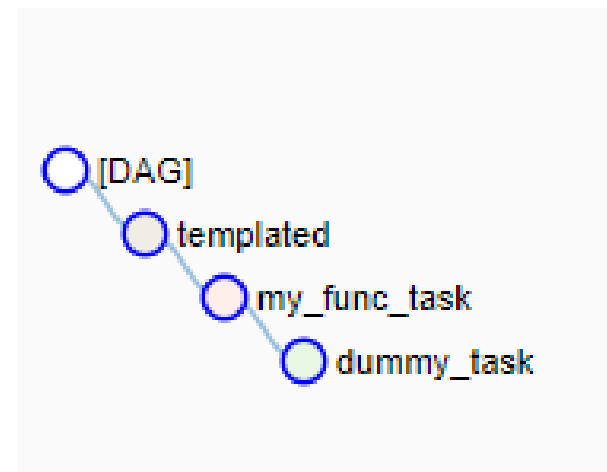
ETL ПРОБЛЕМЫ

- Масштабируемость
- Работа с «неудачными»
выполнениям



ETL ПРОБЛЕМЫ






















- Масштабируемость
- Работа с «неудачными»
выполнениям



```
]default_args = {  
    "owner": "airflow",  
    "depends_on_past": False,  
    "start_date": datetime(2020, 12, 21),  
    "retries": 1,  
}
```

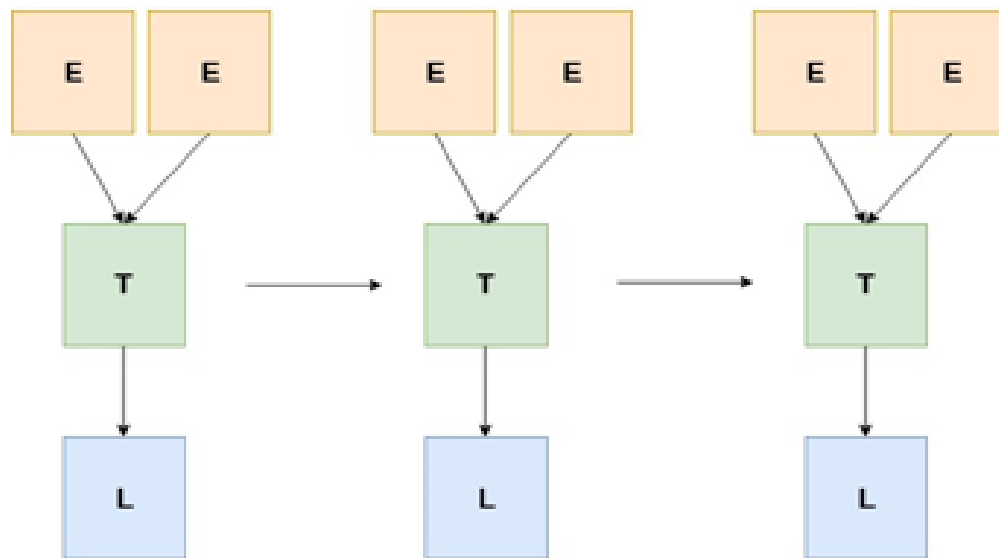

ETL ПРОБЛЕМЫ

- Масштабируемость
- Работа с «неудачными»
выполнениям
- Мониторинг

Search: <input type="text"/>							
	DAG	Schedule	Owner	Recent Tasks ¹	Last Run ¹	DAG Runs ¹	Links
	 aaasteps_in_DAG	0 0 * * *	airflow	      	2020-12-19 12:58 ¹	  	        

ETL ПРОБЛЕМЫ

- Масштабируемость
- Работа с «неудачными» выполнениям
- Мониторинг
- Зависимости



ETL ПРОБЛЕМЫ

- Масштабируемость
- Работа с «неудачными»
выполнениям
- Мониторинг
- Зависимости
- Сохранение историчности



BONOBO



BONOVO

- Код – конфигурация
- Простота и тестируемость
- Pipelines могут работать параллельно
- Быстрый и простой
+ легко объяснить даже **M**

!?

BONOBO



- Каждый процесс возвращает: iter / iterable / callable
- Шаги могут выполняться параллельно / независимо
- Первый пришёл – первый ушёл (FIFO)

BONOBO - TOY EXAMPLE

```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```

!?

BONOBO - TOY EXAMPLE

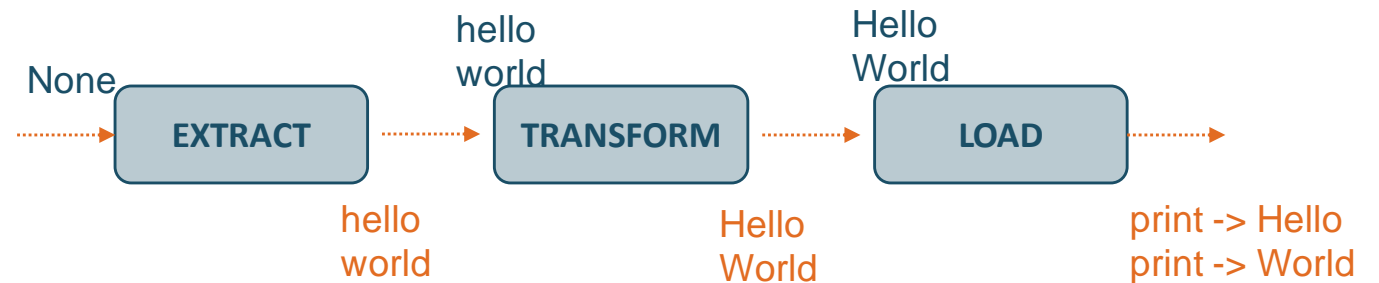
```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```



BONOBO - TOY EXAMPLE

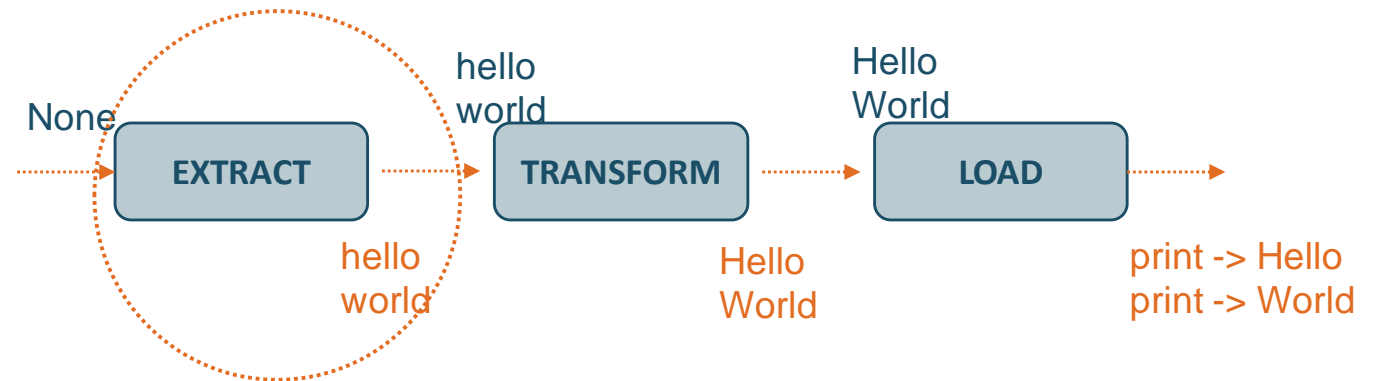
```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```



BONOBO - TOY EXAMPLE

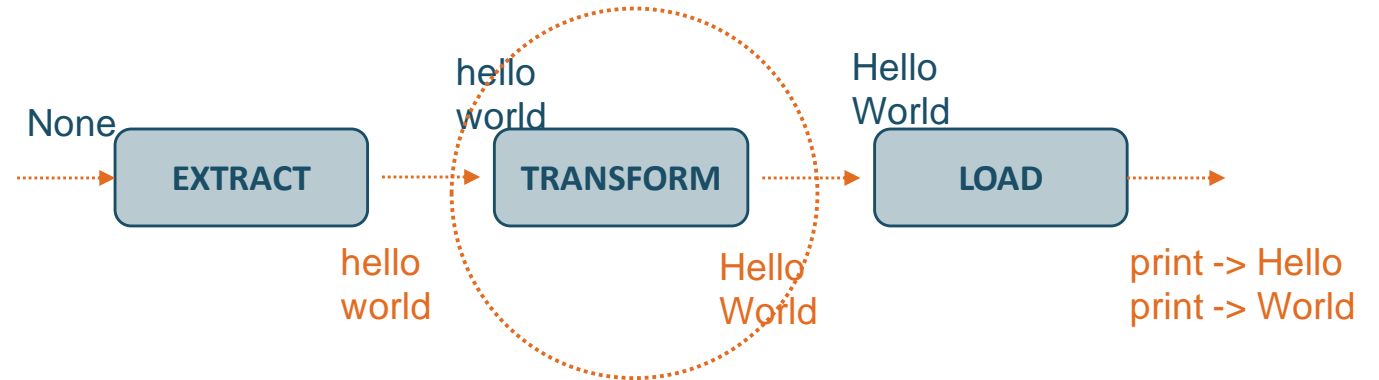
```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```



BONOBO - TOY EXAMPLE

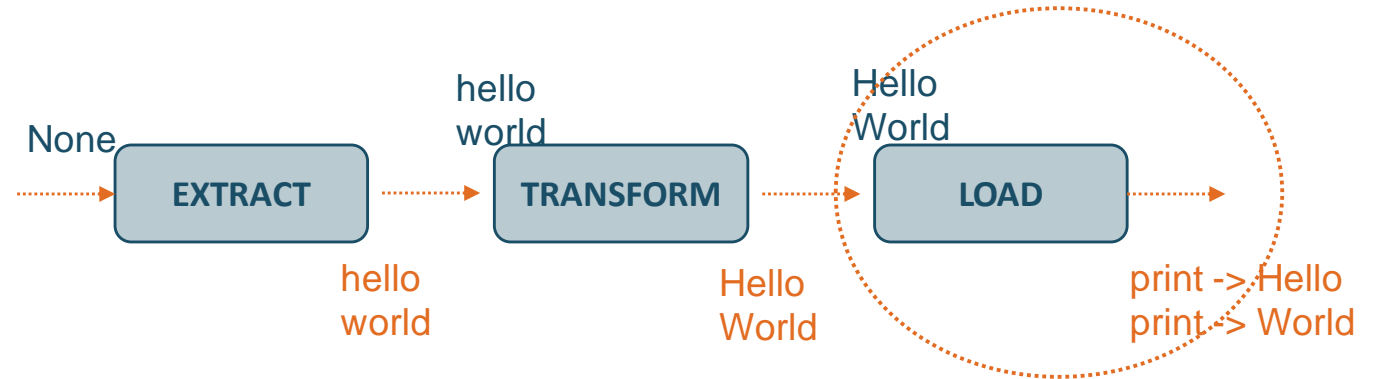
```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```



BONOBO - TOY EXAMPLE

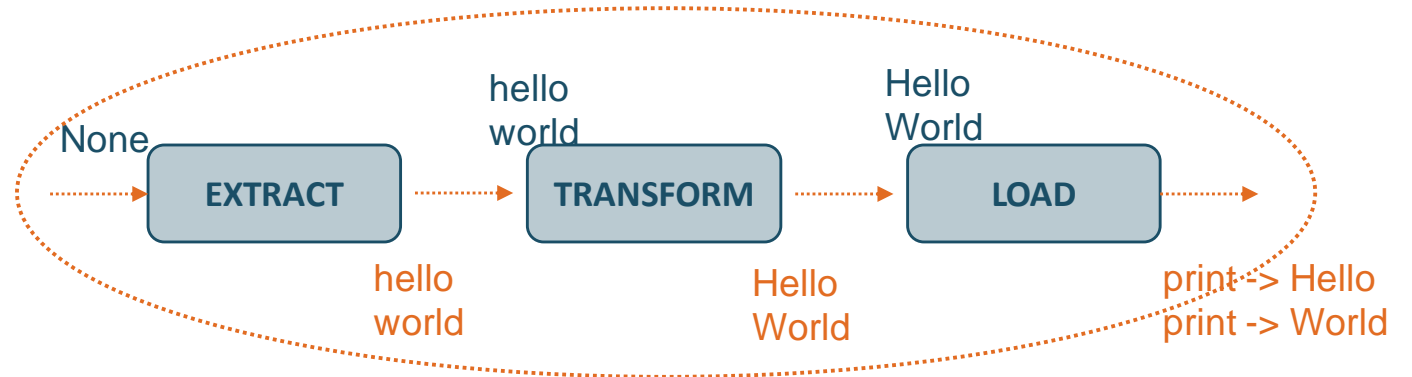
```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```



BONOBO - TOY EXAMPLE

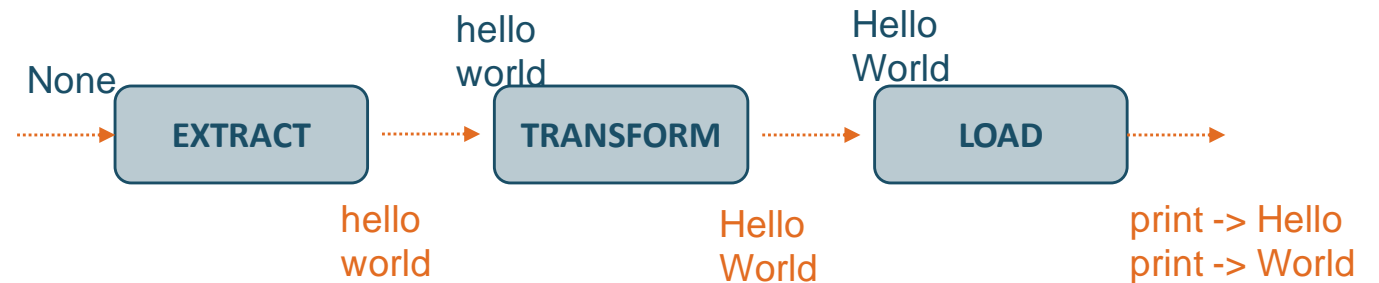
```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```



- extract in=1 out=2 [done]
- transform in=2 out=2 [done]
- load in=2 [done]

BONOBO - TOY EXAMPLE

```
import bonobo

def extract():
    yield 'hello'
    yield 'world'

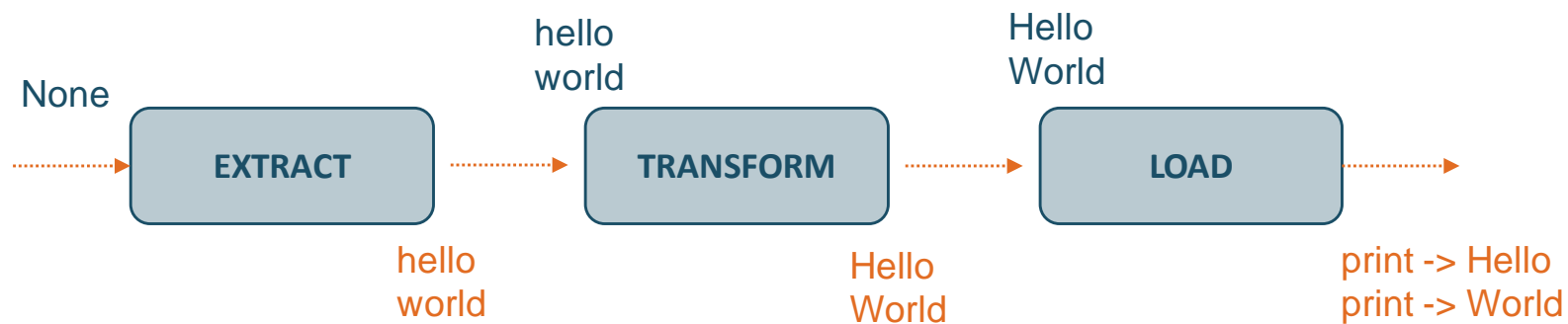
def transform(*args):
    yield tuple(map(str.capitalize, args))

def load(*args):
    print(*args)

def get_graph(**options):
    graph = bonobo.Graph()
    graph.add_chain(extract, transform, load)
    return graph
```

- Код – конфигурация
- Простота и тестируемость
- Быстрый и простой
+ легко объяснить даже **M**

BONOBO



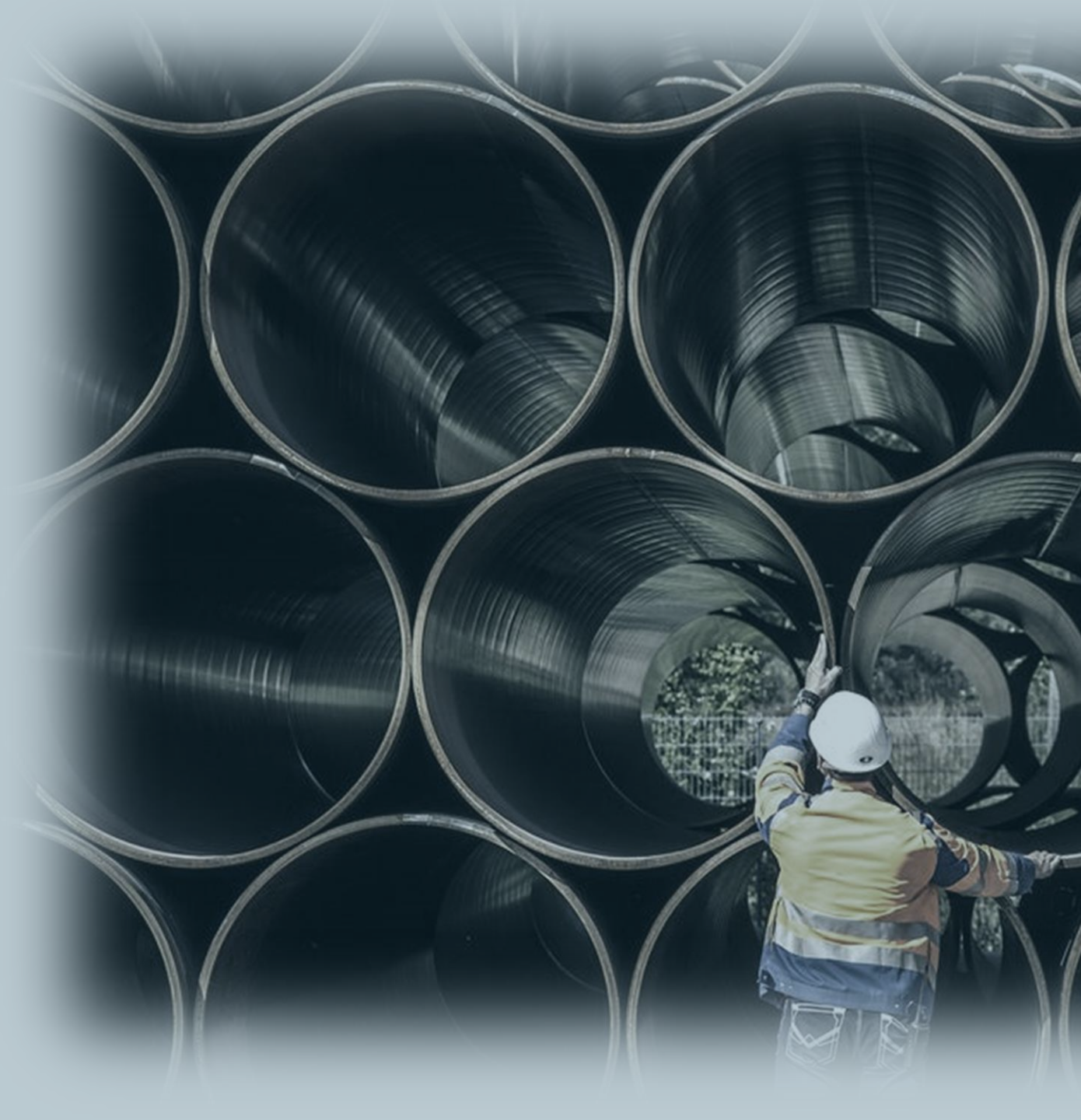
- Pipelines могут работать параллельно



PRACTICE BONOBO

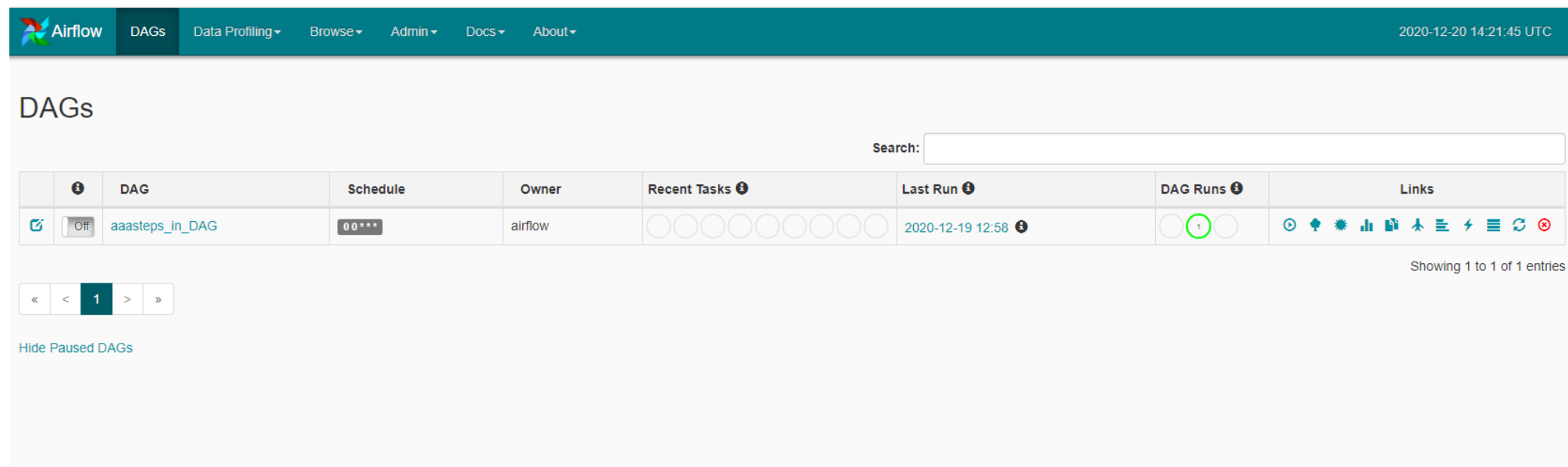


AIRFLOW













ПОЧЕМУ AIRFLOW

- наличие наглядного веб-GUI для визуализации конвейеров обработки данных



The screenshot displays the Apache Airflow web interface. At the top, a teal navigation bar contains the Airflow logo, a menu with 'DAGs', 'Data Profiling', 'Browse', 'Admin', 'Docs', and 'About', and a timestamp '2020-12-20 14:21:45 UTC'. Below the navigation bar, the 'DAGs' section is titled. A search bar is located on the right. A table lists the DAGs, with one entry visible: 'aaasteps_in_DAG'. The table columns are: a checkbox (checked), a status icon (Off), the DAG name, the schedule (00***), the owner (airflow), recent tasks (represented by 10 empty circles), the last run time (2020-12-19 12:58), DAG runs (represented by 3 circles, with the first one containing the number 1), and a links column with various icons. Below the table, there is a pagination control showing '1' of 1 entries and a link to 'Hide Paused DAGs'.

Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>	Off	aaasteps_in_DAG	00***	airflow	○ ○ ○ ○ ○ ○ ○ ○ ○ ○	2020-12-19 12:58	○ 1 ○	         


Showing 1 to 1 of 1 entries

« < 1 > »

[Hide Paused DAGs](#)

ПОЧЕМУ AIRFLOW

- наличие наглядного веб-GUI для визуализации конвейеров обработки данных
- есть scheduler (у остальных, только cron)

Schedule	Owner	Recent Tasks ⓘ	Last Run ⓘ
00***	airflow		2020-12-19 12:58 ⓘ

ПОЧЕМУ AIRFLOW

- наличие наглядного веб-GUI для визуализации конвейеров обработки данных
- есть scheduler (у остальных, только cron)
- пакетный характер работы с данными (batch processing)

ПОЧЕМУ AIRFLOW

- наличие наглядного веб-GUI для визуализации конвейеров обработки данных
- есть scheduler (у остальных, только cron)
- пакетный характер работы с данными (batch processing)
- популярность в области Big Data (много контекстов для BigData решений)

ПОЧЕМУ AIRFLOW

- наличие наглядного веб-GUI для визуализации конвейеров обработки данных
- есть scheduler (у остальных, только cron)
- пакетный характер работы с данными (batch processing)
- популярность в области Big Data (много контекстов для BigData решений)
- есть историчность данных

The screenshot displays the Apache Airflow web interface. At the top, there is a text input field containing the timestamp "2020-12-19 12:58:17". To its right are two dropdown menus: "Number of runs:" set to "25" and "Run:" set to "manual__2020-12-19T12:58:16.775159+00:00". Below the timestamp field, a date picker calendar is open, showing the month of December 2020. The calendar grid highlights the 19th as the selected date. At the bottom of the calendar, there are time selection dropdowns for "12" and "58", and two buttons: a green "Apply" button and a white "Cancel" button.

ПОЧЕМУ AIRFLOW

- наличие наглядного веб-GUI для визуализации конвейеров обработки данных
- есть scheduler (у остальных, только cron)
- пакетный характер работы с данными (batch processing)
- популярность в области Big Data (много контекстов для BigData решений)
- есть историчность данных
- есть общее хранилище для переменных и параметров
- есть сенсоры, которые могут управлять DAG

НЕ LUGI

- отсутствие механизма запуска задач по расписанию (использовать cron)
- трудности масштабирования из-за слишком тесной связи DAG-задач с cron-заданиями
- неудобство GUI
- отсутствие предварительной проверки выполнения задачи.

ПОЧЕМУ AIRFLOW | HE LUIGI

```
import luigi

class HelloWorldTask(luigi.Task):
    task_namespace = 'examples'

    def run(self):
        print("{task} says: Hello world!".format(task=self.__class__.__name__))

if __name__ == '__main__':
    luigi.run(['examples.HelloWorldTask', '--workers', '1', '--local-scheduler'])
```

```
class Foo(luigi.WrapperTask):
    task_namespace = 'examples'

    def run(self):
        print("Running Foo")

    def requires(self):
        for i in range(10):
            yield Bar(i)

class Bar(luigi.Task):
    task_namespace = 'examples'
    num = luigi.IntParameter()

    def run(self):
        time.sleep(1)
        self.output().open('w').close()

    def output(self):
        """
        Returns the target output for this task.

        :return: the target output for this task.
        :rtype: object (:py:class:`~luigi.target.Target`)
        """
        time.sleep(1)
        return luigi.LocalTarget('/tmp/bar/%d' % self.num)
```

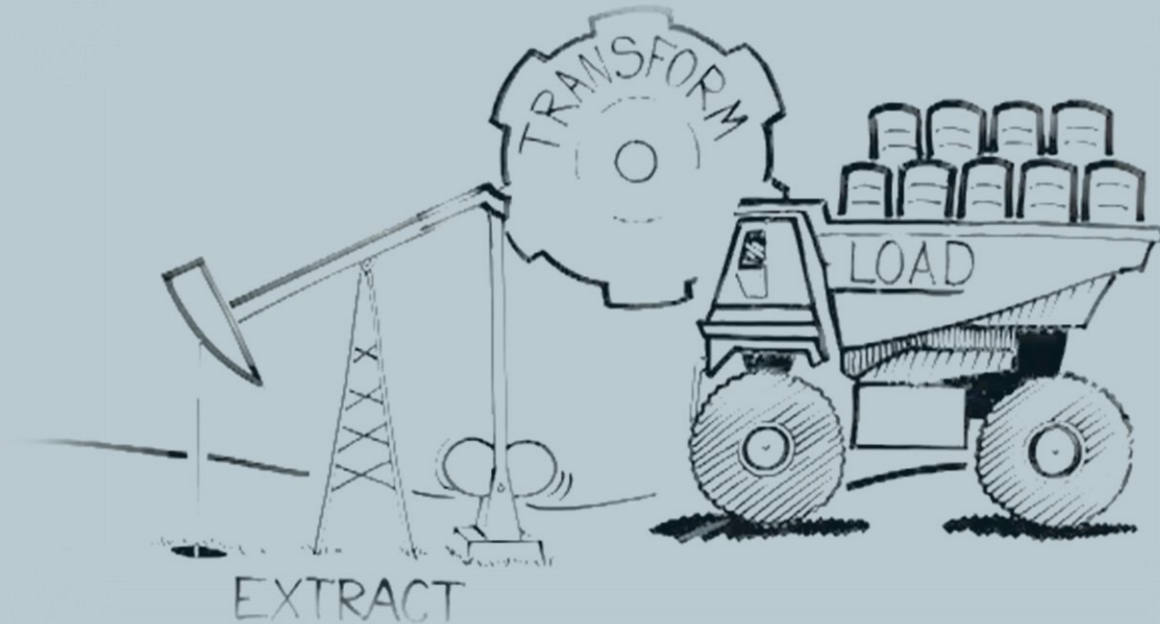
AIRFLOW | LUIGI

LUIGI	AIRFLOW
-	GUI
-	WebServer
Трудно масштабируется	Масштабируемость
API / Classes	API
-	Triggers
Самостоятельный продукт	Много зависимостей

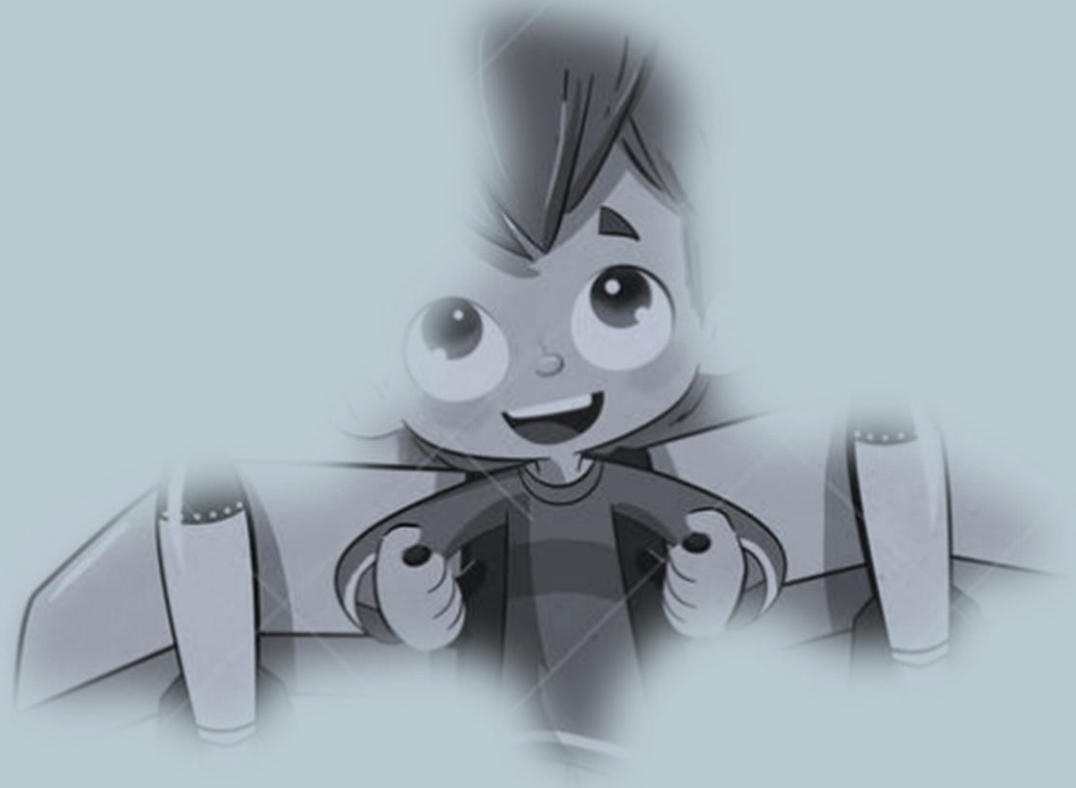
AIRFLOW – 5 ШАГО К УСПЕХУ

1. Импорт модулей
2. Определить функции, аргументы и переменные
3. Инициировать DAG
4. Указать задания
5. Указать зависимости (порядок выполнения задач)

PRACTICE AIRFLOW



ЗАДАНИЕ



ПРОЦЕССЫ В AIRFLOW

- Создать DAG выполнения для вашего DE pipeline (основная задача)
- Создать DAG выполнения для остальных pipeline (при необходимости)
- Используйте docker-compose файл для деплоя Apache Airflow + PostgreSQL
- Добавьте DAG вашего процесса в Apache Airflow и проверьте его пополняемость