Scaling Simple Python Data Apps

Paweł Kordek
SOFTWARE ENGINEER

@pawel_kordek https://kordek.github.io

Dask

Dataset

Python variant

Dask rewrite

Dataset


CANADA
NORTH AMERICA
UNITED STATES
MEXICO
GULF OF MEXICO
CUBA
BAHAMAS
HAITI
DOMINICAN REPUBLIC
JAMAICA
BELIZE
GUATEMALA
HONDURAS
EL SALVADOR
NICARAGUA
COSTA RICA
PANAMA
CARIBBEAN SEA
VENEZUELA
COLOMBIA
GUYANA
SURINAME
FRENCH GUIANA
ECUADOR
PERU
BRAZIL
BOLIVIA
PARAGUAY
CHILE
ARGENTINA
SOUTH AMERICA
NORTH ATLANTIC OCEAN
PACIFIC OCEAN
Galapagos Islands (ECUADOR)
Easter Islands (CHILE)
Pitcairn Island (U.K.)
Bermuda (U.K.)
PUERTO RICO (U.S.)
DOMINICA
BARBADOS
TRINIDAD & TOBAGO
GRENADA
Seattle
Vancouver
Portland
Spokane
Helena
Bismarck
Ottawa
Quebec
Toronto
Chicago
Detroit
Boston
New York
Omaha
Lincoln
Columbus
Indianapolis
Washington D.C.
Salt Lake City
Denver
Eureka
San Francisco
Las Vegas
Phoenix
Los Angeles
San Diego
Tucson
Fort Worth
Dallas
Atlanta
Jackson
Mobile
Jacksonville
Houston
New Orleans
Miami
Havana
Monterrey
Tampico
Mexico
Islas Revillagigedo
Caracas
Panama
Bogota
Quito
Manaus
Belem
Porto Velho
Campo Grande
Lima
La Paz
Sucre
Antofagasta
Santiago
Rosario
Marquesas Islands
FRENCH POLYNESIA

Map Data

OpenStreetMap

OpenStreetMap

Convenient format

Smaller extract – New York City

Bigger extract – South America

```
<node id="26769792" lat="40.6962016" lon="-74.1779077”/>
```

◀ Nodes

```
<node id="26769792" lat="40.6962016" lon="-74.1779077”/>


<node id="26769800" lat="40.685869" lon="-74.1908483">
    <tag k="railway" v="switch"/>
</node>
```

- Nodes
- Tags

```
<node id="26769792" lat="40.6962016" lon="-74.1779077”/>


<node id="26769800" lat="40.685869" lon="-74.1908483">
    <tag k="railway" v="switch"/>
</node>


<way id="5670798">
    <nd ref="42436930"/>
    <nd ref="1764672569"/>
    <nd ref="42434372"/>
    <tag k="name" v="West 217th Street"/>
    <tag k="name_1" v="West 217 Street"/>
    <tag k="oneway" v="yes"/>
    <tag k="highway" v="residential"/>
</way>
```

- Nodes
- Tags
- Ways

```
<node id="26769792" lat="40.6962016" lon="-74.1779077”/>
```

```
<node id="26769800" lat="40.685869" lon="-74.1908483">
    <tag k="railway" v="switch"/>
</node>
```

```
<way id="5670798">
    <nd ref="42436930"/>
    <nd ref="1764672569"/>
    <nd ref="42434372"/>
    <tag k="name" v="West 217th Street"/>
    <tag k="name_1" v="West 217 Street"/>
    <tag k="oneway" v="yes"/>
    <tag k="highway" v="residential"/>
</way>
```

```
<relation id="9465675">
    <member type="way" ref="682202161" role="from"/>
    <member type="node" ref="3577587229" role="via"/>
    <member type="way" ref="351961106" role="to"/>
    <tag k="type" v="restriction"/>
    <tag k="restriction" v="no_right_turn"/>
</relation>
```

- Nodes
- Tags
- Ways
- Relations

Demo

Simple data processing task

No Dask whatsoever

Standard Python vs. big data

The Task

Find five most frequent tags

Lazy reading

Counter needs complete collection

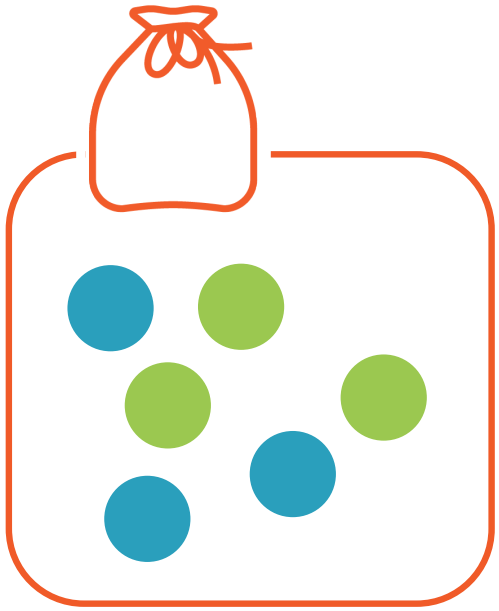
Many other utilities too

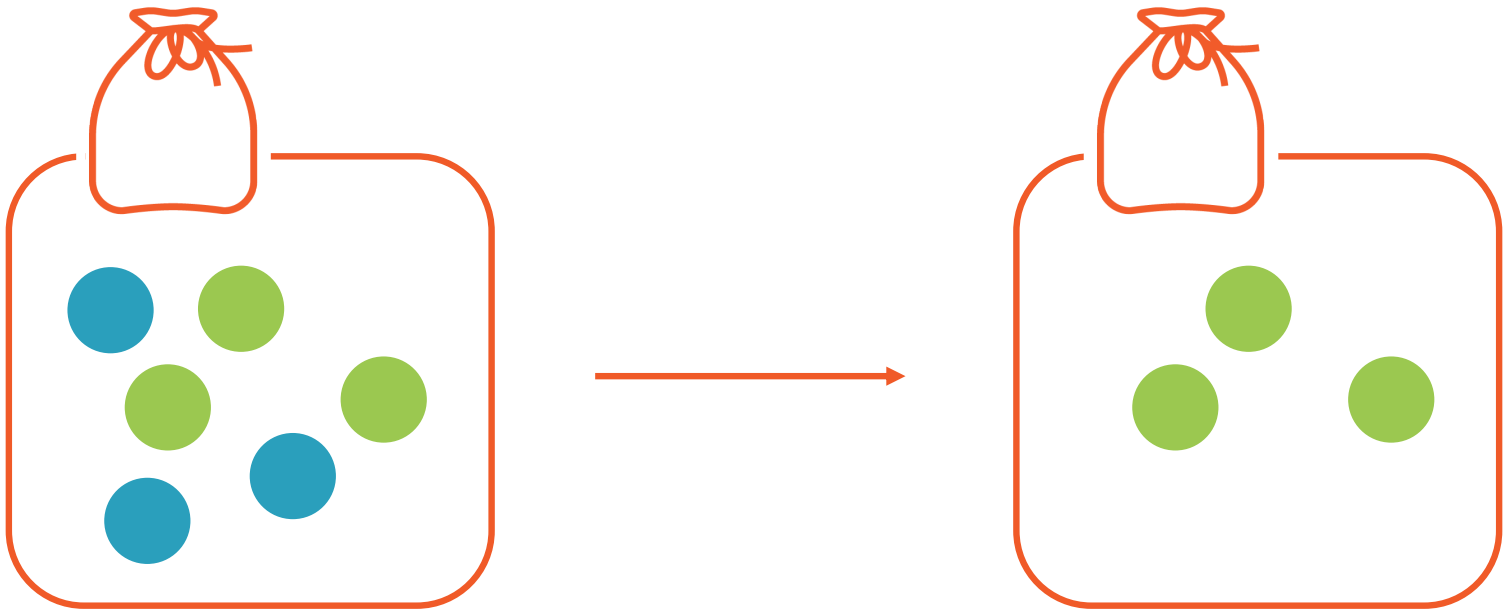
Other strategies are labor-intensive

Dask Bags

Bag

An unordered collection,
that can contain duplicate elements.





Creating Bags

Creating Bags

```
import dask.bag as db

a_list = [1, 4, 10, 20]
simple_bag = db.from_sequence(a_list)
```

Creating Bags

```
import dask.bag as db

a_list = [1, 4, 10, 20]
simple_bag = db.from_sequence(a_list)
files_bag = db.read_text("/some/directory/*csv")
```

Creating Bags

Both bags are not materialized

LAZINESS

```
import dask.bag as db


a_list = [1, 4, 10, 20]

simple_bag = db.from_sequence(a_list)

files_bag = db.read_text("/some/directory/*csv")
```

Operations on Bags

```
column_bag = files_bag.map(lambda x: x.split(",")[2])

add_bag = simple_bag.map(lambda x: x + 1)
```

Operations on Bags

Operations on Bags

```
column_bag.compute()
```

Operations on Bags

```
column_bag.compute()
```

```
column_bag.take(5)
```

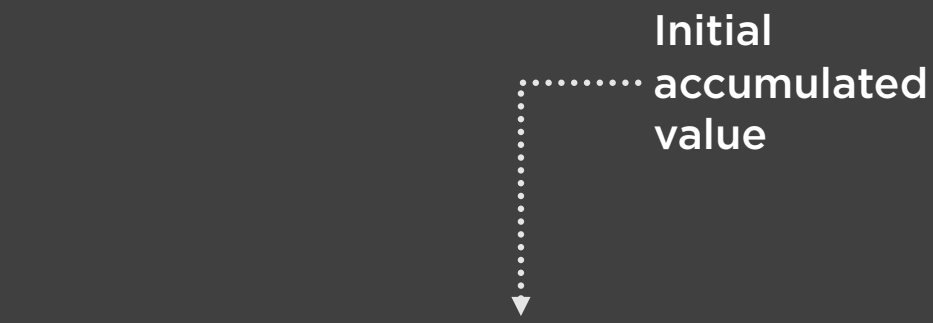
Operations on Bags

```
column_bag.compute()

column_bag.take(5)

column_bag.to_textfiles("/some/directory")
```

Fold

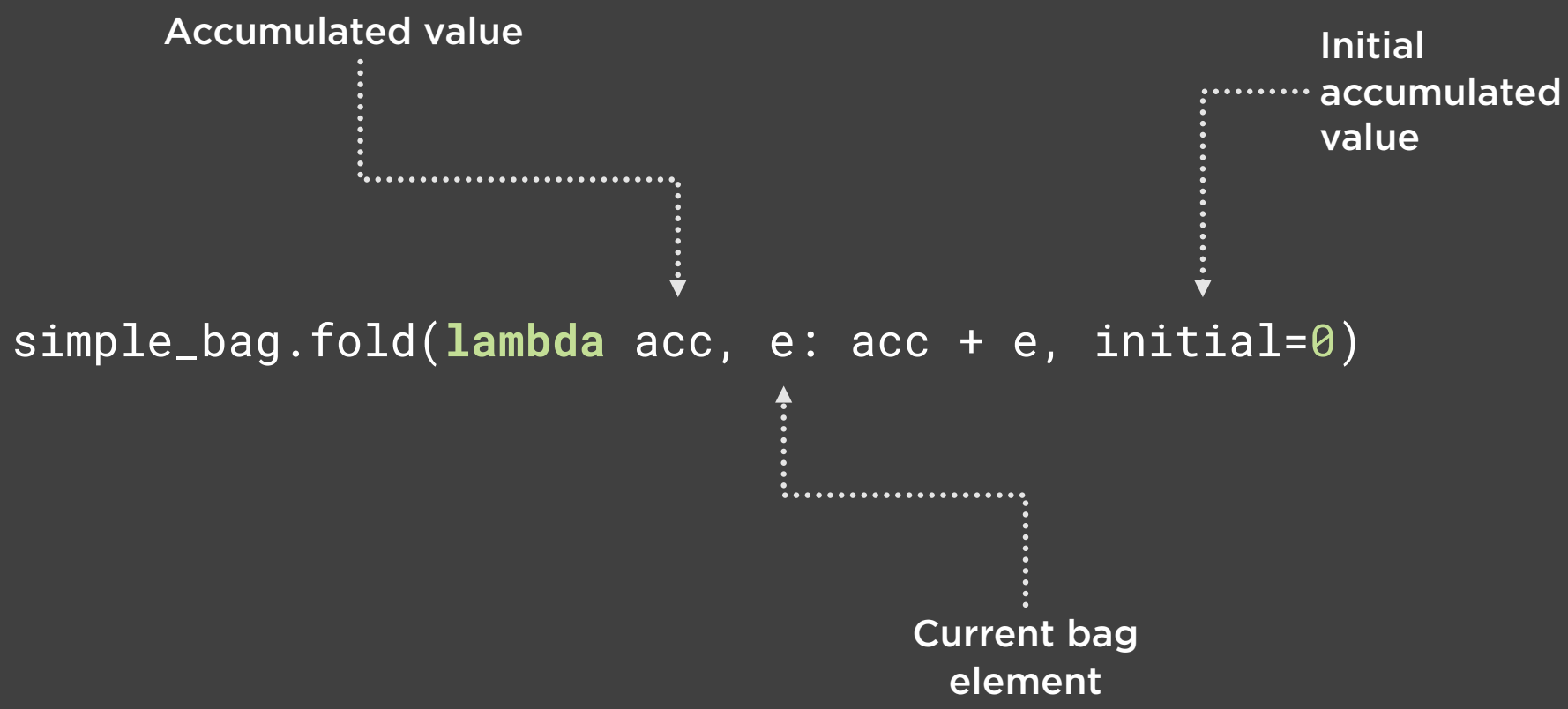
```
simple_bag.fold(lambda acc, e: acc + e, initial=0)
```

Fold




```
simple_bag.fold(lambda acc, e: acc + e, initial=0)
```

Fold


Accumulated value
Initial accumulated value
simple_bag.fold(lambda acc, e: acc + e, initial=0)
Current bag element

Fold Example

Arg	Value
acc	0
e	-

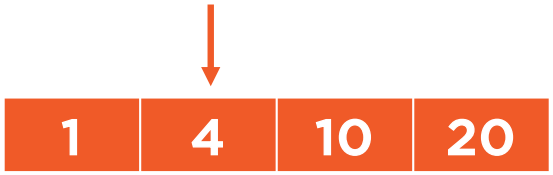
Fold Example

Arg	Value
acc	0
e	1

Fold Example

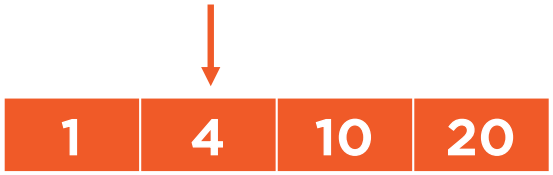
Arg	Value
acc	1
e	1

Fold Example



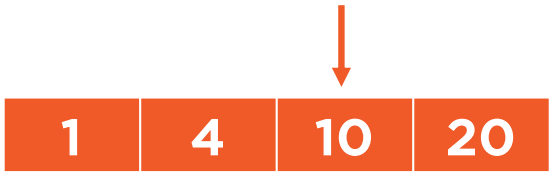
Arg	Value
acc	1
e	4

Fold Example



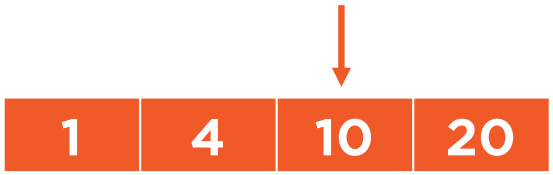
Arg	Value
acc	5
e	4

Fold Example



Arg	Value
acc	5
e	10

Fold Example



Arg	Value
acc	15
e	10

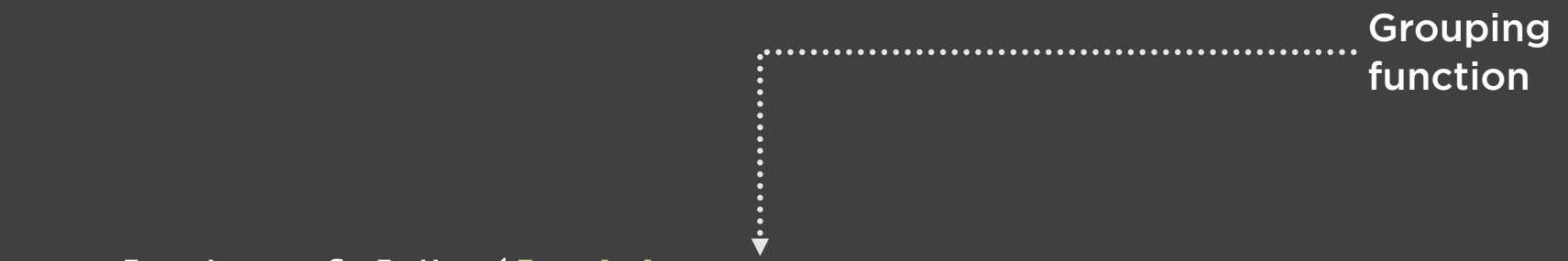
Fold Example

Arg	Value
acc	15
e	20

Fold Example

Arg	Value
acc	35
e	20

Fold with Grouping




```
simple_bag.foldby(lambda x: x % 2,
                  lambda acc, e: acc + e,
                  initial=0)
```

Demo

Warm-up

Tackling South America

Summary

Abstracts the complexity

Consistent and elegant API

Applies to a wide variety of problems