Components of a data platform

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

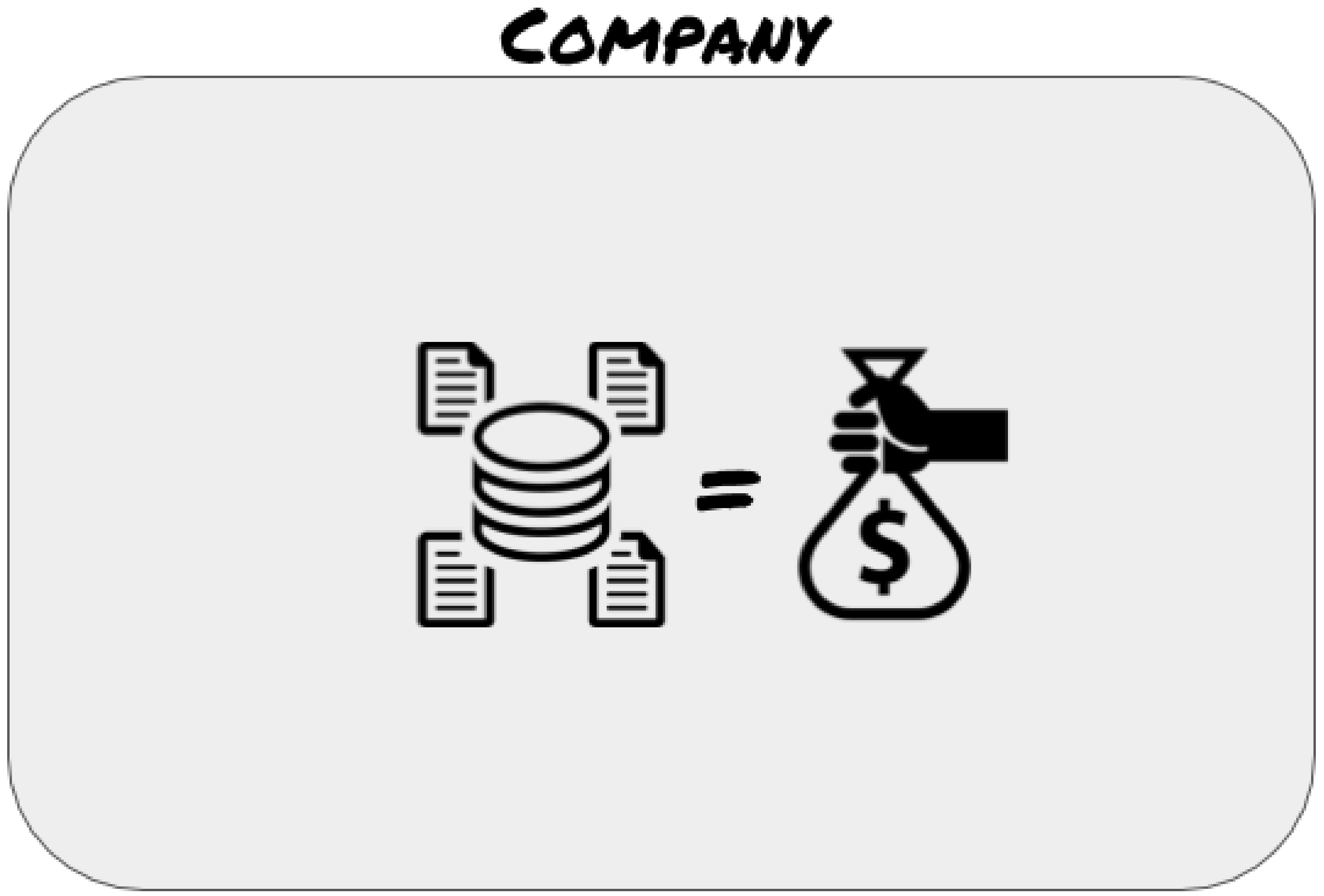
Oliver Willekens
Data Engineer at Data Minded

Course contents

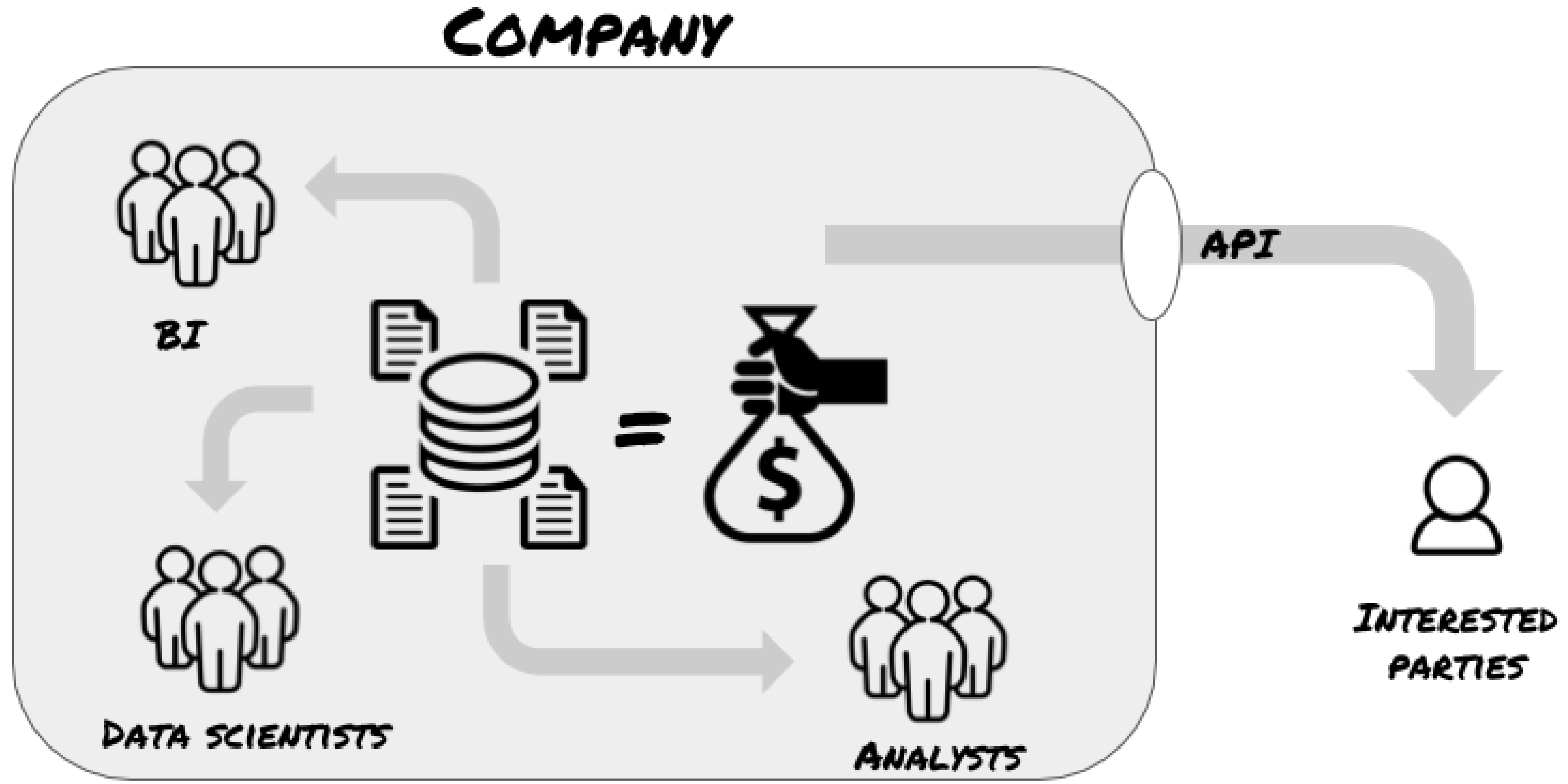
- ingest data using Singer
- apply common data cleaning operations
- gain insights by combining data with PySpark
- test your code automatically
- deploy Spark transformation pipelines

=> intro to data engineering pipelines

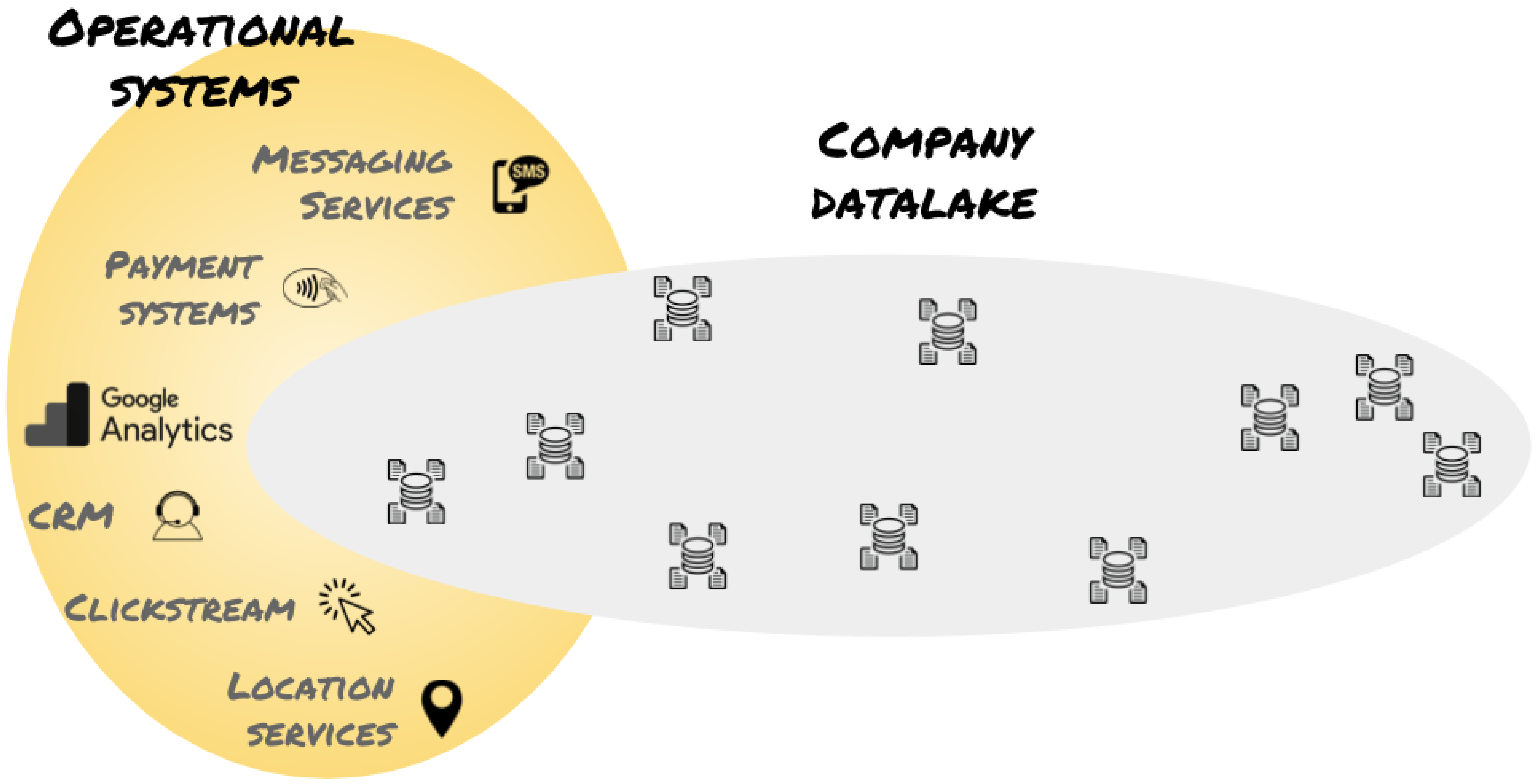
Data is valuable



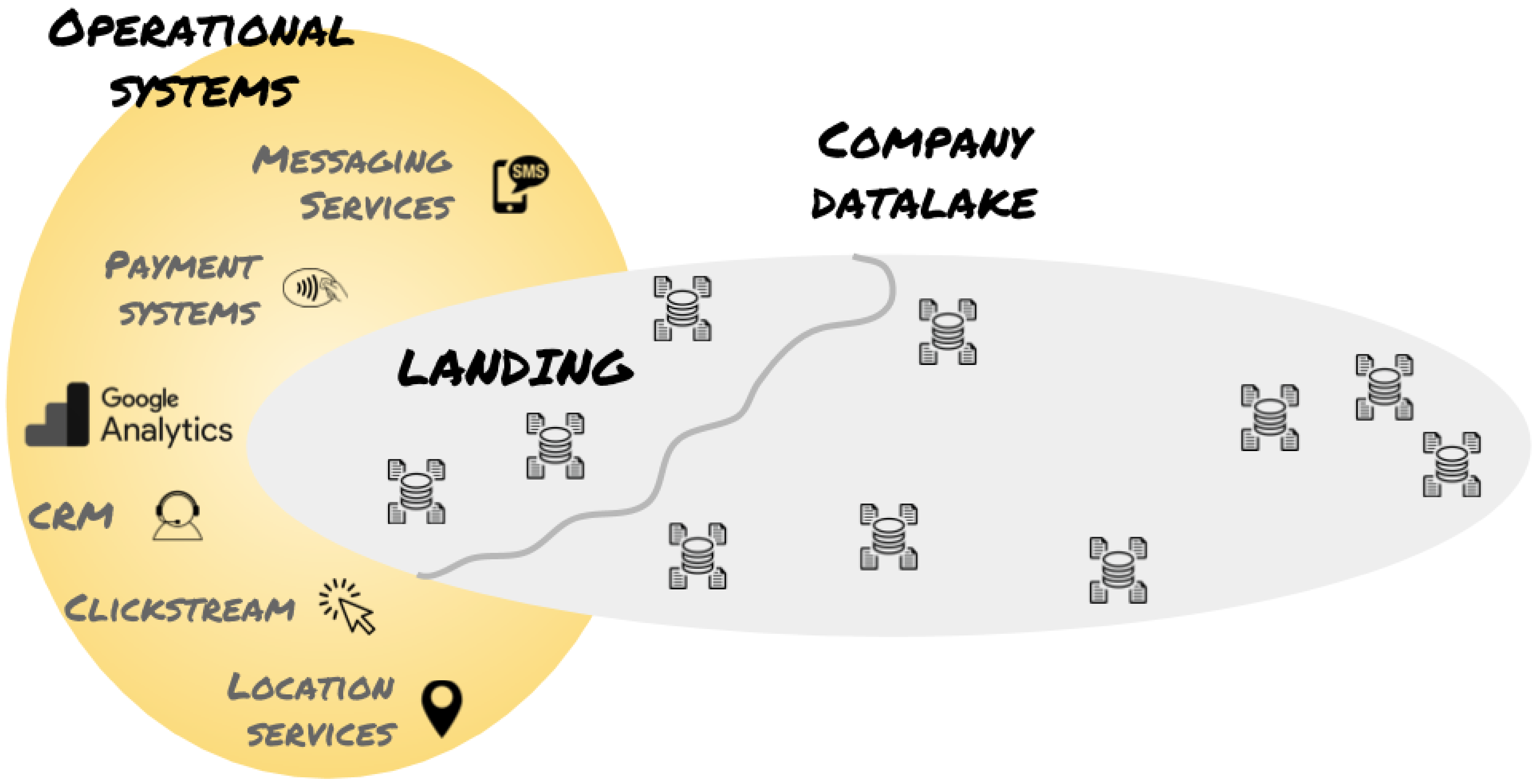
Democratizing data increases insights



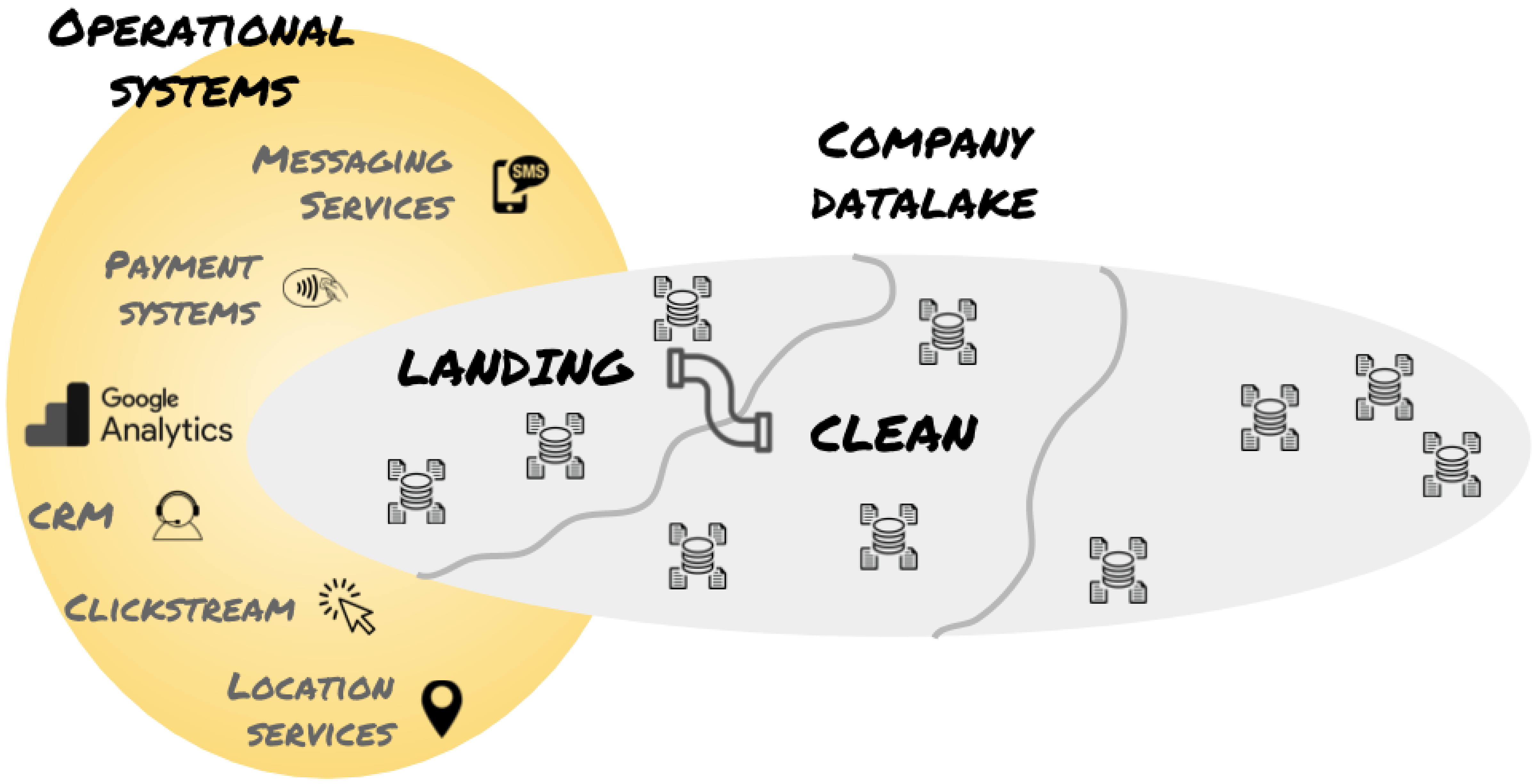
Genesis of the data



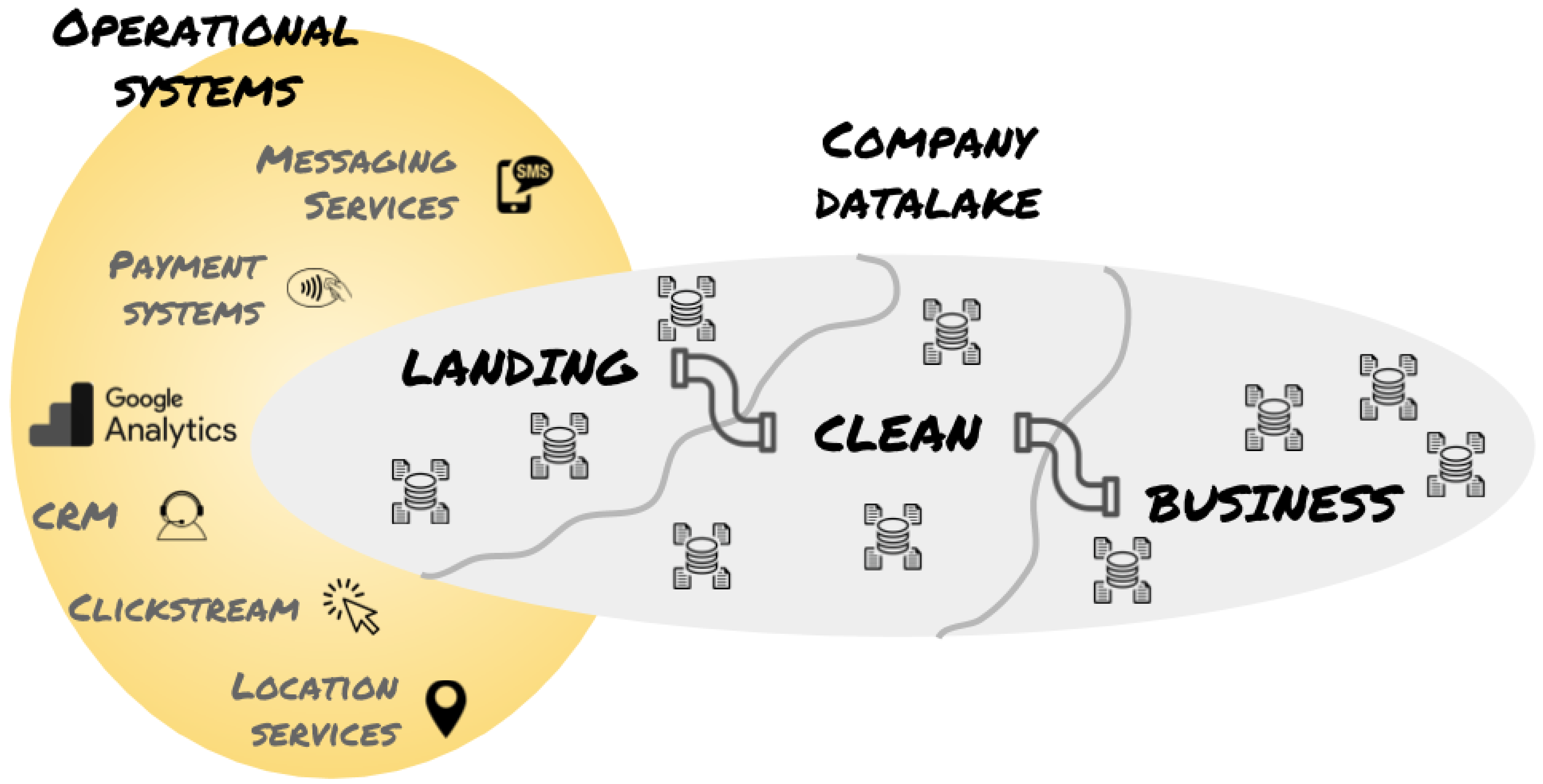
Operational data is stored in the landing zone



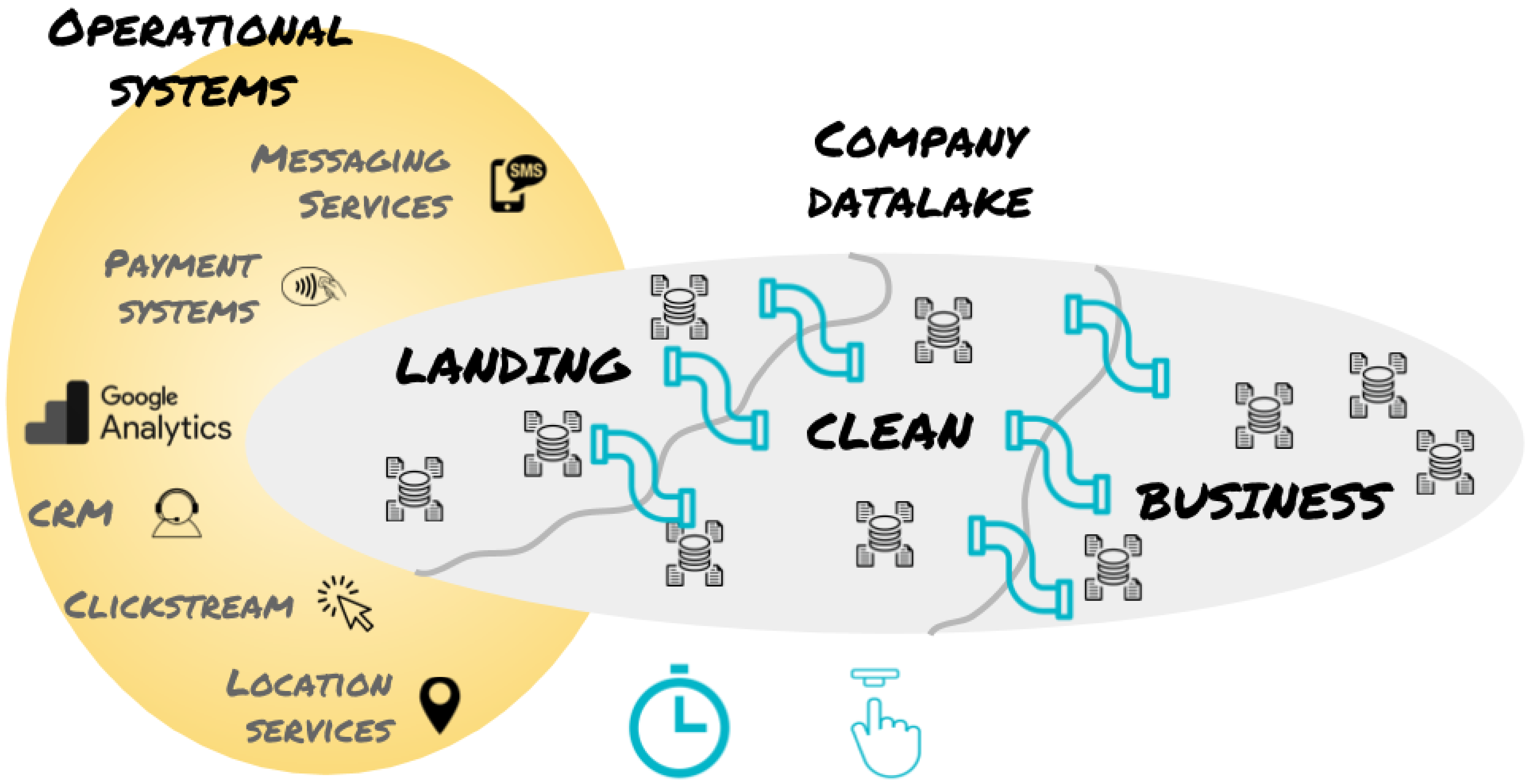
Cleaned data prevents rework



The business layer provides most insights



Pipelines move data from one zone to another



Let's reason!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Introduction to data ingestion with Singer

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

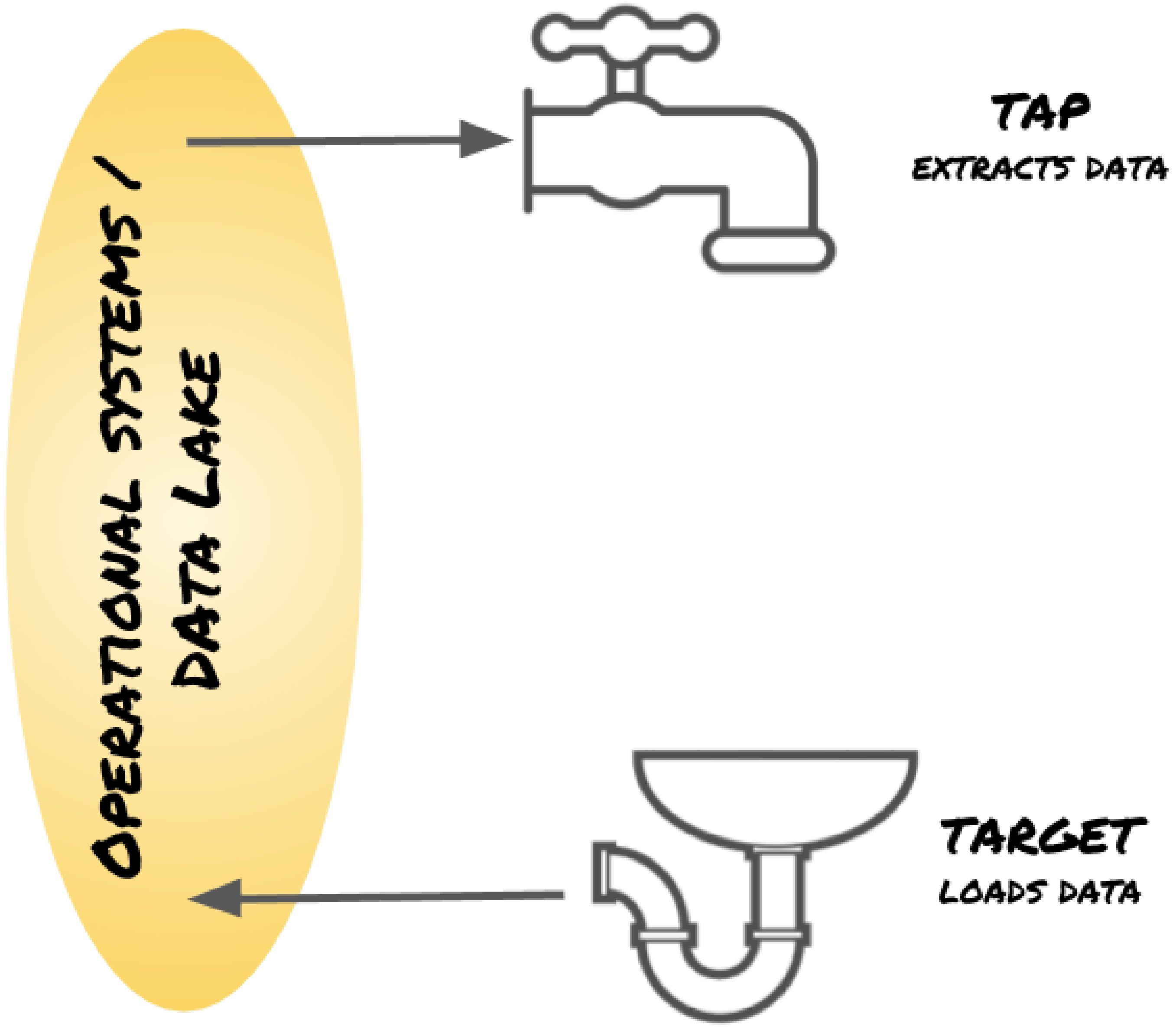
Oliver Willekens
Data Engineer at Data Minded

Singer's core concepts

Aim: "The open-source standard for writing scripts that move data"

Singer is a *specification*

- data exchange format: *JSON*
- extract and load with *taps* and *targets*
  - => language independent

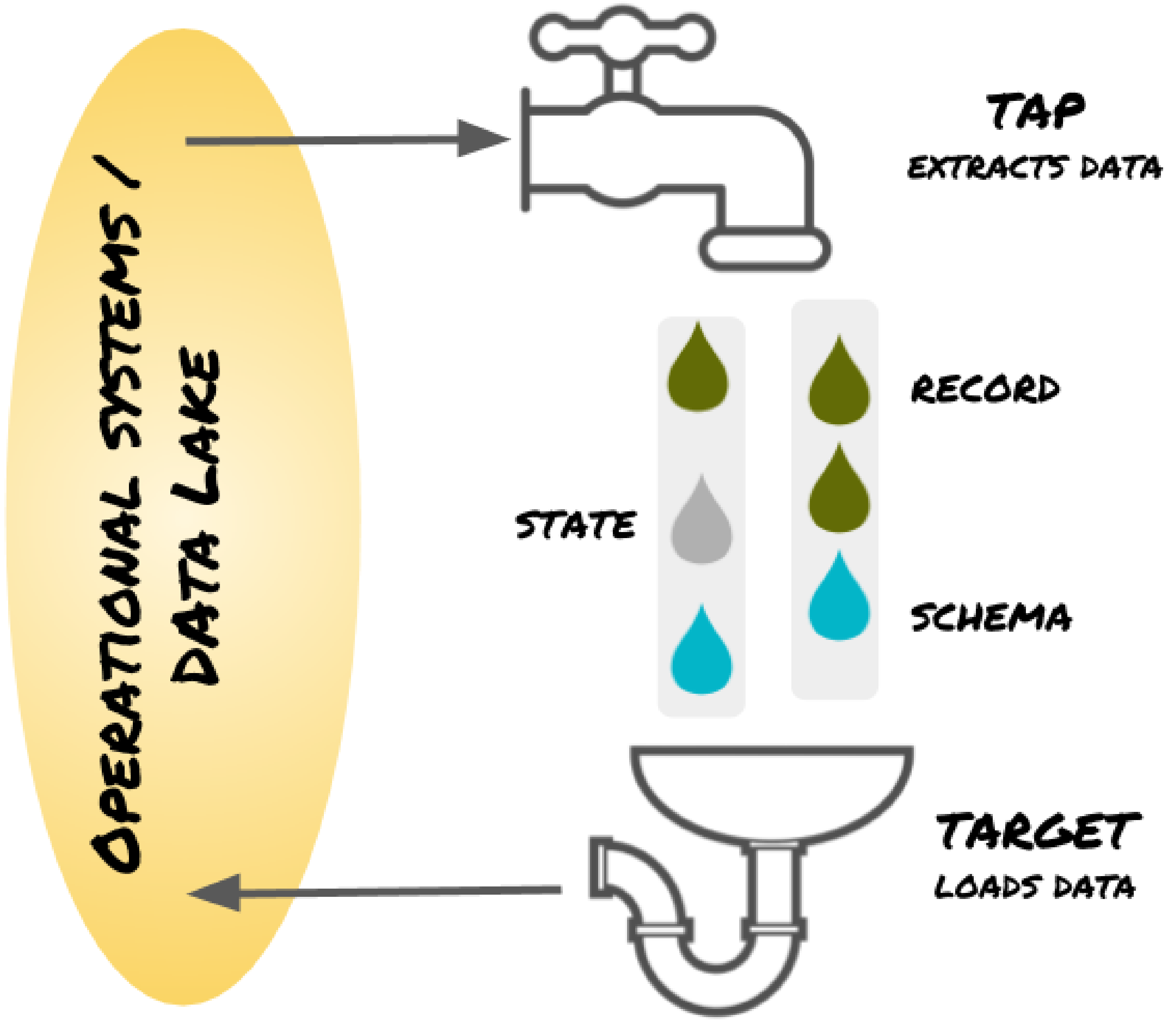


Singer's core concepts

Aim: "The open-source standard for writing scripts that move data"

Singer is a *specification*

- data exchange format: *JSON*
- extract and load with *taps* and *targets*
    - => language independent
- communicate over *streams*:
    - schema (metadata)
    - state (process metadata)
    - record (data)

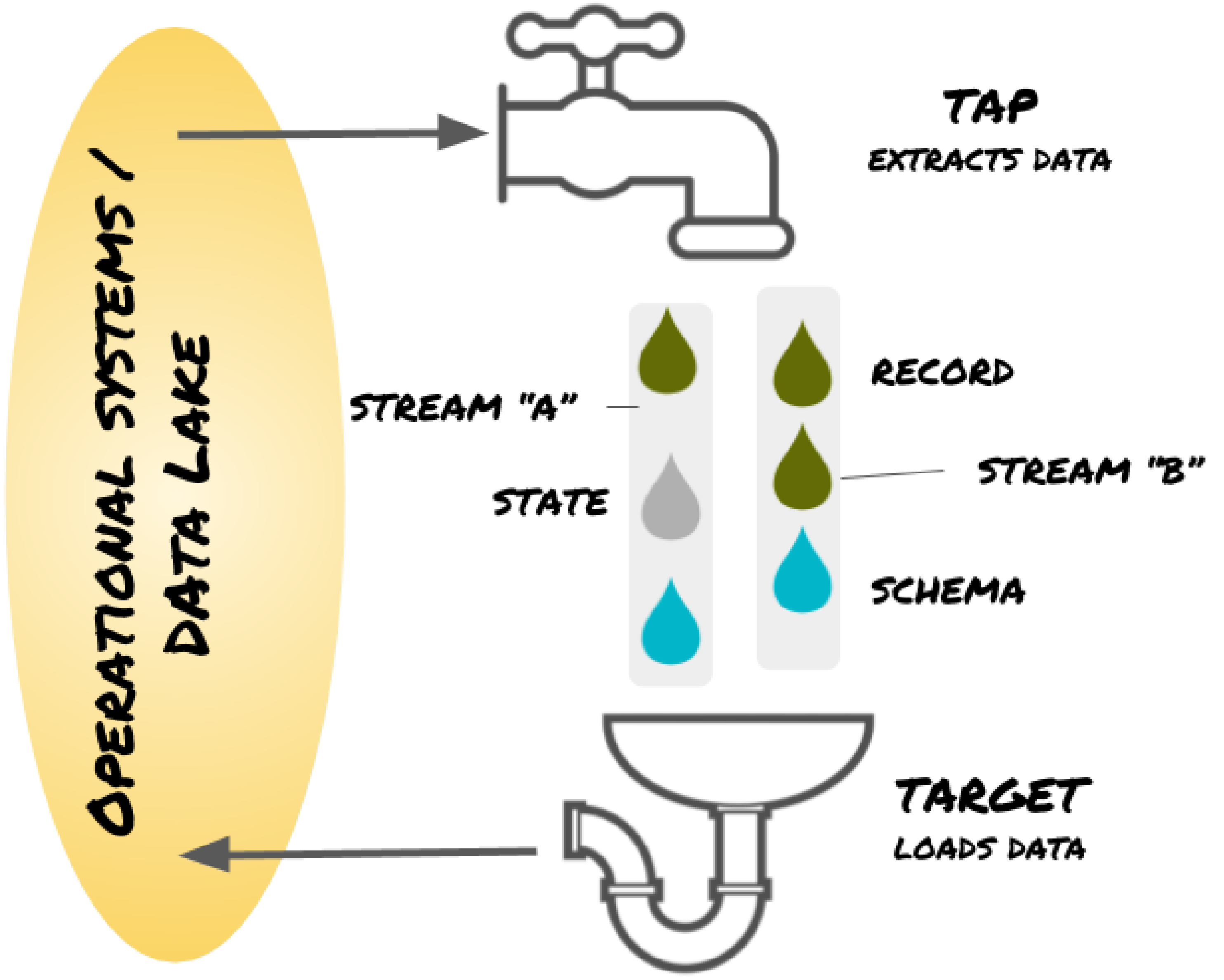


Singer's core concepts

Aim: "The open-source standard for writing scripts that move data"

Singer is a *specification*

- data exchange format: *JSON*
- extract and load with *taps* and *targets*
  - => language independent
- communicate over *streams*:
  - schema (metadata)
  - state (process metadata)
  - record (data)



Describing the data through its schema

```
columns = ("id", "name", "age", "has_children")
users = {(1, "Adrian", 32, False),
         (2, "Ruanne", 28, False),
         (3, "Hillary", 29, True)}
json_schema = {
    "properties": {"age": {"maximum": 130,
                           "minimum": 1,
                           "type": "integer"},
                   "has_children": {"type": "boolean"},
                   "id": {"type": "integer"},
                   "name": {"type": "string"}},
    "$id": "http://yourdomain.com/schemas/my_user_schema.json",
    "$schema": "http://json-schema.org/draft-07/schema#"}
```

Describing the data through its schema

```python
import singer
singer.write_schema(schema=json_schema,
                    stream_name='DC_employees',
                    key_properties=["id"])
```

```
{"type": "SCHEMA", "stream": "DC_employees", "schema": {"properties":
{"age": {"maximum": 130, "minimum": 1, "type": "integer"}, "has_children":
{"type": "boolean"}, "id": {"type": "integer"}, "name": {"type": "string"}},
"$id": "http://yourdomain.com/schemas/my_user_schema.json",
"$schema": "http://json-schema.org/draft-07/schema#"}, "key_properties": ["id"]}
```

Serializing JSON

```python
import json


json.dumps(json_schema["properties"]["age"])
```

```
'{"maximum": 130, "minimum": 1, "type": "integer"}'
```

```python
with open("foo.json", mode="w") as fh:
    json.dump(obj=json_schema, fp=fh)  # writes the json-serialized object
                                       # to the open file handle
```

Let's practice!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Running an ingestion pipeline with Singer

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Oliver Willekens
Data Engineer at Data Minded

Streaming record messages

```
columns = ("id", "name", "age", "has_children")
users = {(1, "Adrian", 32, False),
         (2, "Ruanne", 28, False),
         (3, "Hillary", 29, True)}
```

```
singer.write_record(stream_name="DC_employees",
                    record=dict(zip(columns, users.pop())))
```

```
{"type": "RECORD", "stream": "DC_employees", "record": {"id": 1, "name": "Adrian", "age": 32, "has_children": false}}
```

```
fixed_dict = {"type": "RECORD", "stream": "DC_employees"}
record_msg = {**fixed_dict, "record": dict(zip(columns, users.pop()))}
print(json.dumps(record_msg))
```

Chaining taps and targets

```
# Module: my_tap.py
import singer


singer.write_schema(stream_name="foo", schema=…)
singer.write_records(stream_name="foo", records=…)
```

Ingestion pipeline: **Pipe** the tap's output into a Singer target, using the `|` symbol (Linux & MacOS)

```
python my_tap.py | target-csv
python my_tap.py | target-csv --config userconfig.cfg
my-packaged-tap | target-csv --config userconfig.cfg
```

Modular ingestion pipelines

```
my-packaged-tap | target-csv
my-packaged-tap | target-google-sheets
my-packaged-tap | target-postgresql --config conf.json
```

```
tap-custom-google-scraper | target-postgresql --config headlines.json
```

Keeping track with state messages

Keeping track with state messages

id	name	last_updated_on
1	Adrian	2019-06-14T14:00:04.000+02:00
2	Ruanne	2019-06-16T18:33:21.000+02:00
3	Hillary	2019-06-14T10:05:12.000+02:00

```
singer.write_state(value={"max-last-updated-on": some_variable})
```

Run this `tap-mydelta` on 2019-06-14 at 12:00:00.000+02:00 (2nd row wasn't yet present then):

```
{"type": "STATE", "value": {"max-last-updated-on": "2019-06-14T10:05:12.000+02:00"}}
```

Let’s practice!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON