

# From “All-At-Once, Once-A-Day” To “A-Little-Each-Time, All-The-Time”

Emanuele Bardelli - OLX Group

#SAISExp5

*OLX Group and its Data Platform*

*Users Communication and Lifecycle Management*

*Near-Time Data Engineering with Spark*

*Takeaways & Questions*

## Our Mission

we fuel local economies  
by making it  
**super easy**  
for anyone to  
**buy or sell**  
almost anything  
through our platforms



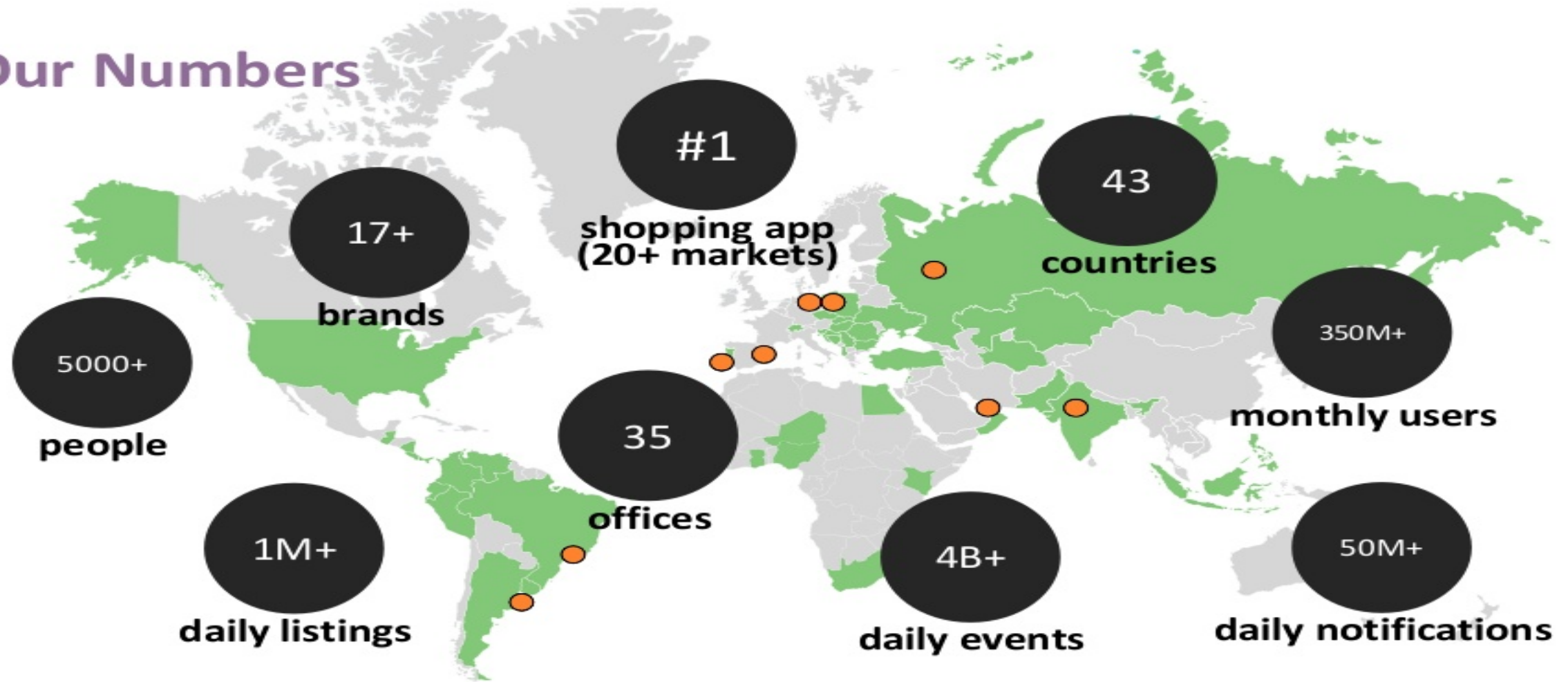
## Who We Are



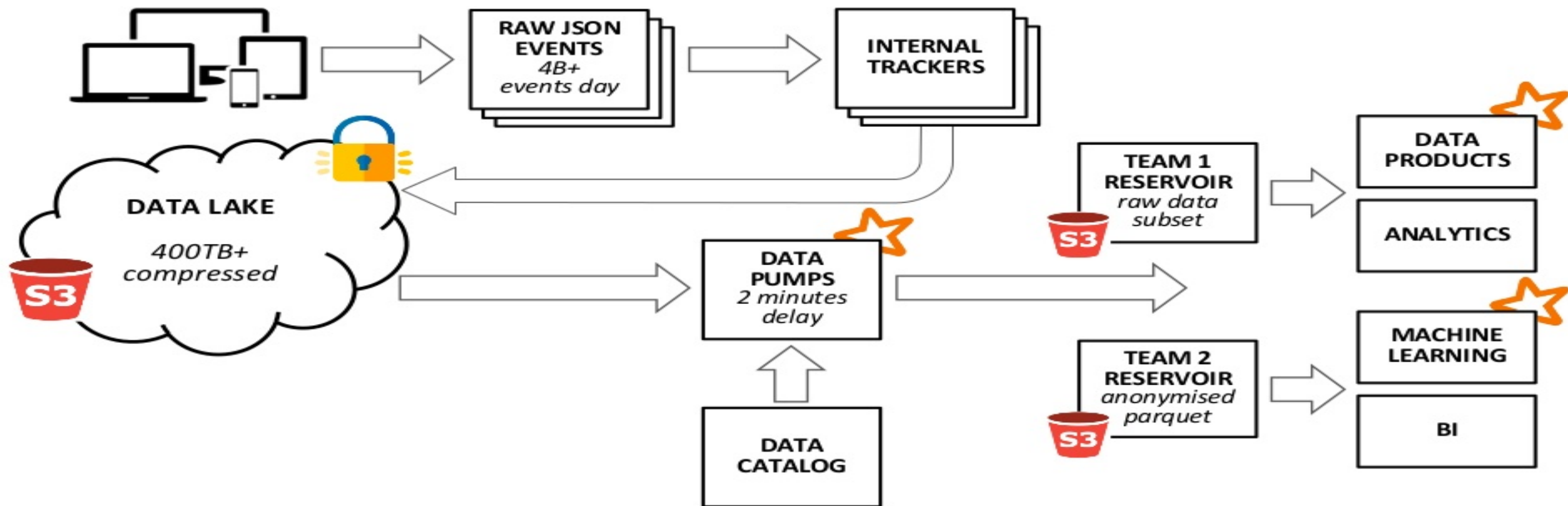
we operate a network of  
**market-leading trading platforms**  
in over **40 countries**  
that are used by more than  
**350 million people** every month  
to **buy and sell** almost anything



## Our Numbers



## Data Platform Overview





## Users Communication Team

*“optimize and deliver all OLX Group  
end-user communication”*

### Transactional

standard messages  
produced due to  
day-to-day interactions

*(reset password, post  
approved, new follower, ...)*

### Marketing

one-off messages  
sent to advertise  
specific initiatives

*(happy holidays, get the  
new OLX app, ...)*

### Lifecycle (CLM)

messages crafted to  
influence customers  
journey positively

*(we miss you, we picked  
this item just for you, ...)*

## Customers Lifecycle Management

**70%**

*of mobile APPs users are lost after 3 days*

**40%**

*increase in Daily Active Users*

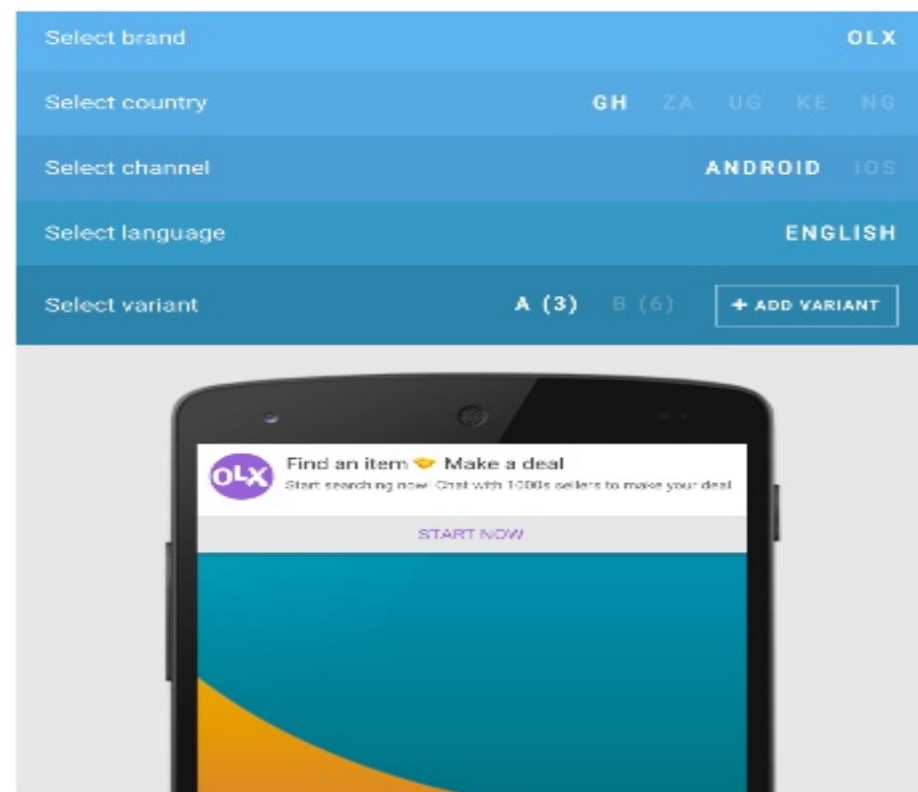
**15%**

*contribution to Value Added Services purchase*

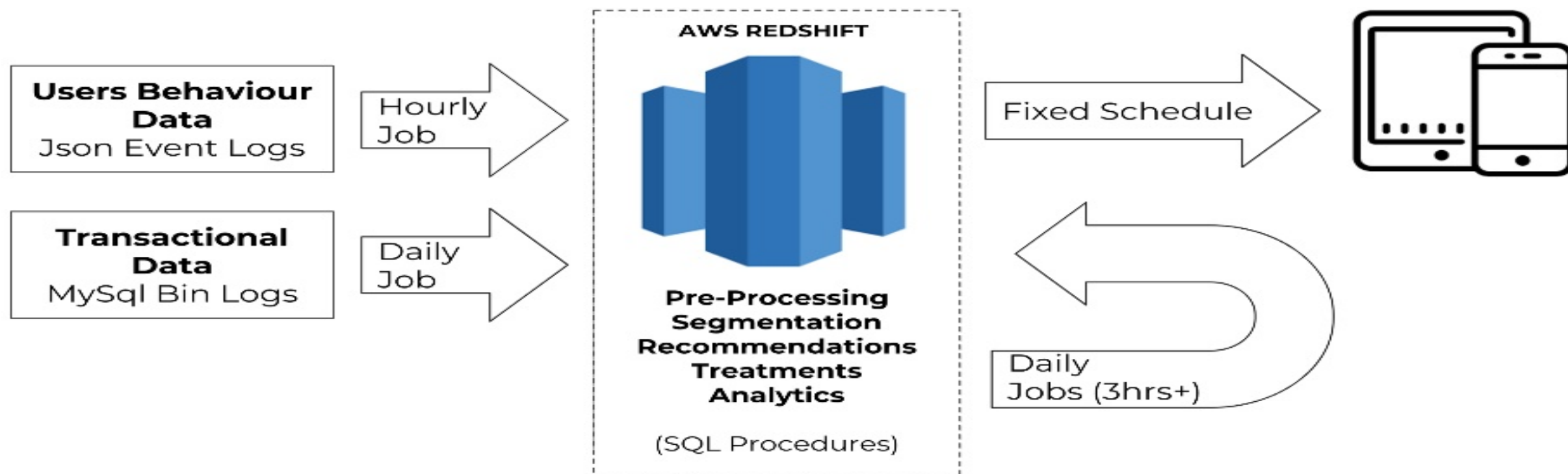


## CLM Services Ecosystem

- Segmentation
- Recommendations
- User-Devices Mapping
- Live / Control Group Bucketing
- Audience Generation
- Treatments Definition (A/B Testing)
- Messages Prioritization
- Frequency Capping
- Scheduling
- Analytics



## Former “Batch” Approach

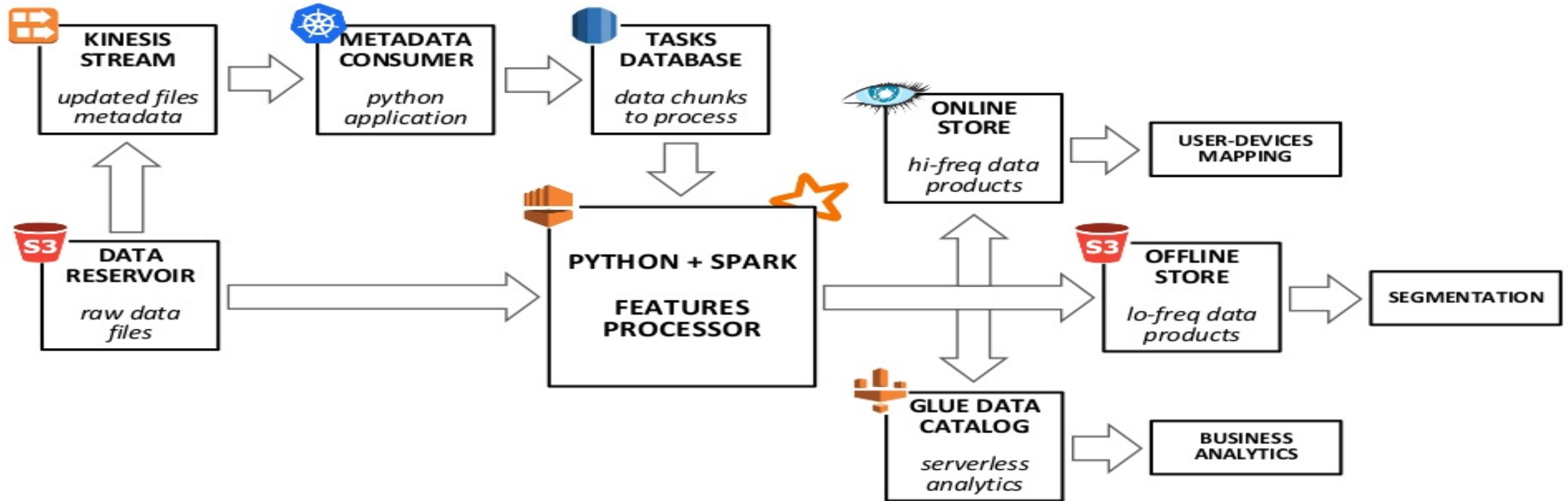


*From*  
*“All-At-Once, Once-A-Day”*  
*To*  
*“A-Little-Each-Time, All-The-Time”*

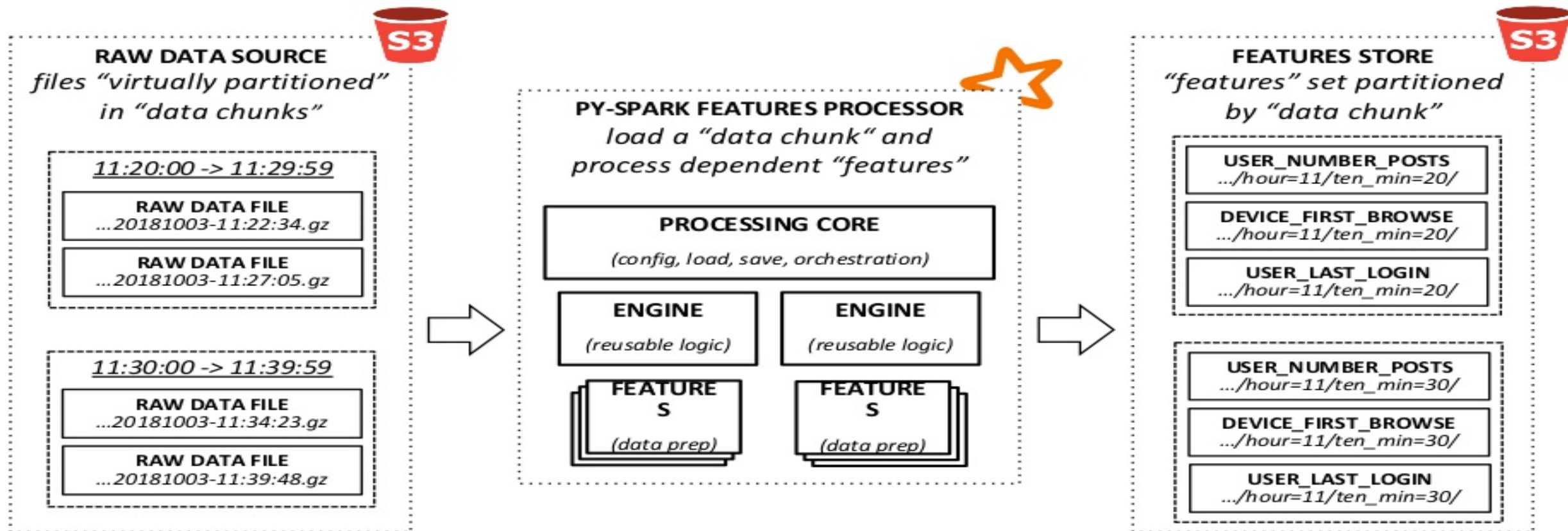
## New “Near-Time” Approach

- **Near-Time** - Produce and send messages within 10 minutes after a set of events happened
- **Open** - Give our users freedom to be more creative and experiment more
- **Reliable** - Powered by technologies that make it easy to apply engineering best practices
- **Flexible** - Adapt to Big Data challenges easily

## Near-Time Data Processing with Spark (Platform)



# Near-Time Data Processing with Spark (Data Flow)





## Data Chunks Metadata Example

*kinesis stream metadata (raw data files paths)*

```
s3://raw-data-reservoir/in/horizontals/olx/android/2018/08/01/201808-01-104234.gz
s3://raw-data-reservoir/in/horizontals/olx/android/2018/08/01/201808-01-104400.gz
s3://raw-data-reservoir/in/horizontals/olx/android/2018/08/01/201808-01-104954.gz

s3://raw-data-reservoir/in/horizontals/olx/android/2018/08/01/201808-01-105403.gz
s3://raw-data-reservoir/in/horizontals/olx/android/2018/08/01/201808-01-105744.gz
s3://raw-data-reservoir/in/horizontals/olx/android/2018/08/01/201808-01-105923.gz
```

*tasks database records (data chunks to process)*

source_bucket	file_path	data_chunk	ts_to_process
s3://raw-data-reservoir	in/horizontals/olx/android/	2018-08-01 10:40:00	2018-08-01 10:53:00
s3://raw-data-reservoir	in/horizontals/olx/android/	2018-08-01 10:50:00	2018-08-01 11:03:00

## Processed Features Data Example

```
parquet-tools head ../features_store/user_numb_posts/year=2018/month=08/day=01/hour=09/ten_mins=30/

id = c360f89568
country = gh
brand = olx
actions_count = 1

id = 8fe540bb38
country = za
brand = olx
actions_count = 3

parquet-tools head ../features_store/user_last_login/year=2018/month=08/day=01/hour=10/ten_mins=10/

id = 0947a3e926
country = cz
brand = letgo
last_action_timestamp = 1533118630

id = ff3ca262b3
country = ke
brand = olx
last_action_timestamp = 1533118751
```

# Features Code Example (Data Preparation)

```
class FeatureDeviceLastBrowse(EngineLastActionTime):
```

```
def _preprocess_engine_input_data(self):
    engine_input_data = self._source_data_frame

    # pre-process source data
    engine_input_data = \
        engine_input_data\
            .withColumn('device_id', clean_str('device_id'))\
            .withColumn('country', clean_str('country'))\
            .withColumn('brand', clean_str('brand'))\

    # filter data relevant to features
    events = ['app_open', 'view_listings', 'view_item',
              'search_start', 'chat_inbox', 'on_resume'] ★

    engine_input_data = \
        engine_input_data\
            .filter(filter_col('event_name', events))

    # expose columns engine requires
    engine_input_data = \
        engine_input_data\
            .select('device_id', 'country', 'brand', 'timestamp')

    return engine_input_data
```

```
class FeatureUserLastLogin(EngineLastActionTime):
```

```
def _preprocess_engine_input_data(self):
    engine_input_data = self._source_data_frame

    # pre-process source data
    engine_input_data = \
        engine_input_data\
            .withColumn('user_id', clean_str('user_id'))\
            .withColumn('country', clean_str('country'))\
            .withColumn('brand', clean_str('brand'))\

    # filter data relevant to features
    events = ['login_sign_in_complete'] ★

    engine_input_data = \
        engine_input_data \
            .filter(filter_col('event_name', events))

    # expose columns engine requires
    engine_input_data = \
        engine_input_data\
            .select('user_id', 'country', 'brand', 'timestamp')

    return engine_input_data
```



## Engines Code Example (Data Transformation)

```
class EngineLastActionTime(ProcessingCore):  
  
    # define input schema (raw data from feature)  
    def _engine_input_data_schema(self):  
        return StructType([  
            StructField('id', StringType(), True),  
            StructField('country', StringType(), True),  
            StructField('brand', StringType(), True),  
            StructField('timestamp', IntegerType(), True)  
        ])  
  
    # define logic transformation (input -> output)  
    def _engine_transform(self, df):  
        df = df.groupBy('id', 'country', 'brand')\  
                .agg(F.max('timestamp'))  
  
        return df  
  
    # define output schema (transformed data to save)  
    def _engine_output_data_schema(self):  
        return StructType([  
            StructField('id', StringType(), True),  
            StructField('country', StringType(), True),  
            StructField('brand', StringType(), True),  
            StructField('last_action_ts', IntegerType(), True)  
        ])
```

# Data Product Example (Serverless Analytics)

✓ New query 1 +

```
1 select
2   brand, country, count(distinct(id)) cnt
3 from
4   features_store.device_last_browse
5 where
6   year='2018' and month = '09' and day = '20'|
7 group by 1, 2
8 order by 3 desc
```

Run query Save as Create view from query (Run time: 6.65 seconds, Data scanned: 58.35MB)

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete Format query Clear

...

**Results** 📄

	brand	country	cnt
1	letgo	cz	893241
2	olx	za	761791
3	olx	ke	271683

## Spark Environment

- **PySpark 2.3.0** : Python3 + Spark SQL + Dataframes
- **AWS EMR 5.15** (Yarn) : r5.xlarge instances  
1 Master + 1 Core + N Task (SPOT)
- **Deployment Mode** : Client
- **Scheduler Mode** : Fair
- **Features Stores** : **Offline** S3 (Parquet) – **Online** Cassandra
- **CI / CD** : GitLab Pipeline (RPM Packaging)



## Takeaways

- Goodbye Schedulers, **Welcome Events!**
- Raw data is messy and unpredictable. **Constantly (re)processing small chunks of data** helps with accuracy and consistency
- **Separation of storage and compute** helps to optimize resources and allows scalability on-the-cheap
- Python + Spark : **Software + Data + Science engineered together** for effective big data processing

# THANK YOU! Questions?

## GET IN TOUCH WITH US!



**olxgroup.com**



**tech.olx.com**



**@olxtechberlin**



**manu@olx.com**



**/in/emanuelebardelli**

