UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Streaming processing of cosmic rays using Drift Tubes detectors

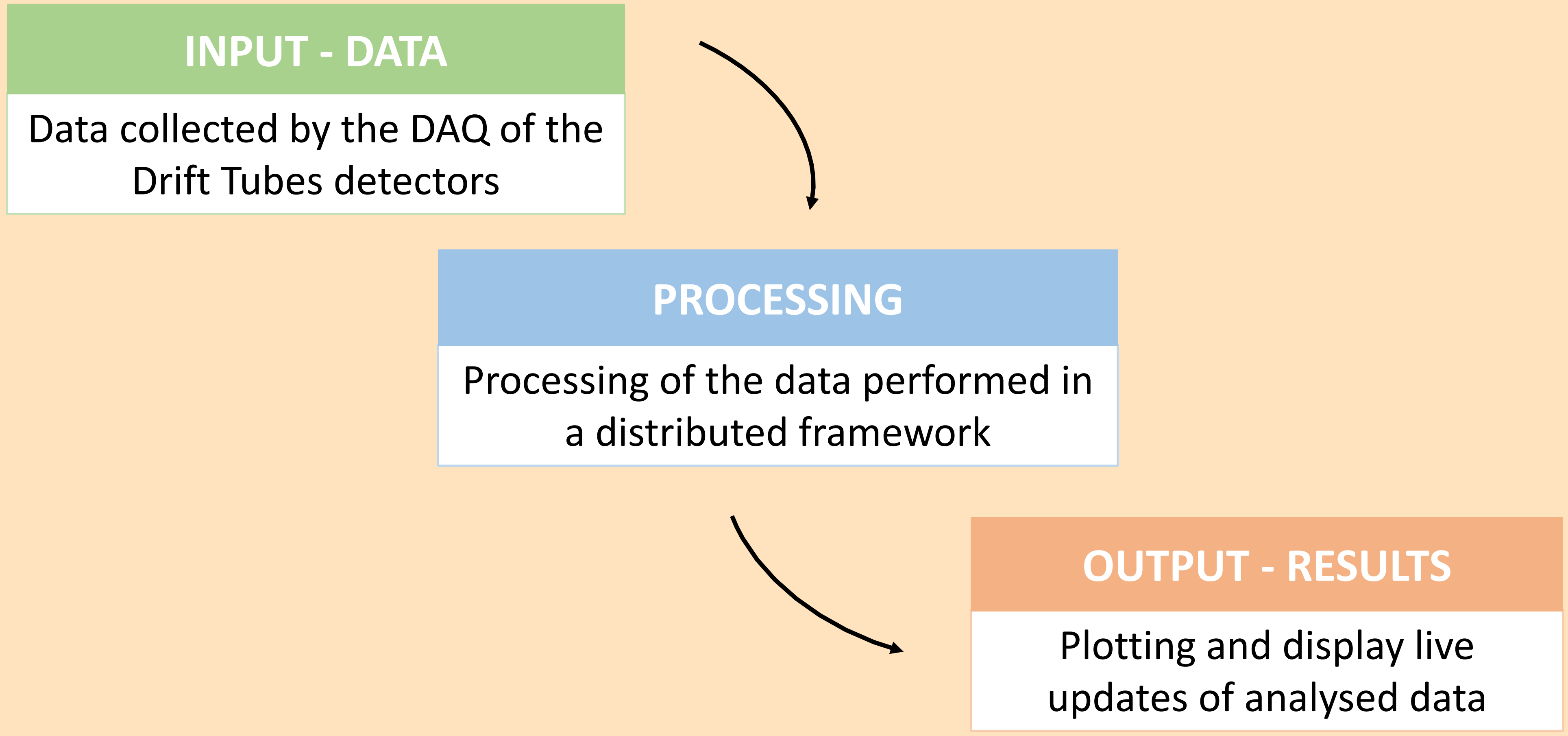
Buriola Lorenzo
Campesan Giulia
Giorgetti Sabrina

Overview

- Project intro
  - Cluster creation
  - Kafka and Spark settings for the VM
- Kafka and Spark implementations
  - Kafka producer + consumer
  - Kafka/Spark interface
- Analysis of data in analysis
  - Spark to Kafka
- Results

Framework

Implemented in a cluster of computers



INPUT - DATA

Data collected by the DAQ of the Drift Tubes detectors

PROCESSING

Processing of the data performed in a distributed framework

OUTPUT - RESULTS

Plotting and display live updates of analysed data

Cluster configuration

On Cloud Veneto we had 5 VM, the following steps have been performed in order to configure the cluster:

1. Access VM:

```
$ ssh -L3232:localhost:3232 -L2080:10.67.22.248:8080 -L2040:10.67.22.248:4040 user@gate.cloudveneto.it

$ ssh -L3232:localhost:3232 root@10.67.22.248

$ jupyter notebook --ip 127.0.0.1 --port 3232 --no-browser --allow-root
```

We used the master 10.67.22.248 as our main machine to run Jupyter

SPARK

Master

MAPD-B_Gr12-1 10.67.22.248

Slaves

MAPD-B_Gr12-2 10.67.22.110
MAPD-B_Gr12-3 10.67.22.177
MAPD-B_Gr12-5 10.67.22.16

KAFKA BROKER + ZOOKEEPER

MAPD-B_Gr12-4 10.67.22.185

Cluster configuration

On Cloud Veneto we had 5 VM, the following steps have been performed in order to configure the cluster:

2. *$ yum install openssh-server openssh-client*

3. Setting the machines names:

$ sudo vim /etc/hosts

```
10.67.22.248  master
10.67.22.185  kafka
10.67.22.110  slave01
10.67.22.177  slave02
10.67.22.16   slave03
```

4. All machines were connected with ssh passwordless:

A. Generate key : *$ ssh − keygen − t rsa – P “ ”*
B. Make key an authorized one cat*: $ ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys*
C. Copy key in all machines: *$ ssh-copy-id root@<IP>*

SPARK

Master
MAPD-B_Gr12-1 10.67.22.248

Slaves
MAPD-B_Gr12-2 10.67.22.110
MAPD-B_Gr12-3 10.67.22.177
MAPD-B_Gr12-5 10.67.22.16

KAFKA BROKER + ZOOKEEPER

MAPD-B_Gr12-4 10.67.22.185

Cluster configuration: Spark

SPARK

1. Install java

```
$ yum install java-1.8.0-openjdk
```

2. Install pyspark and pyarrow

3. Download and install Spark

```
$ mv spark-3.1.1-bin-hadoop3.2 /usr/local/spark
$ vim ~/.bashrc  modify and $ source ~/.bashrc
    export PATH = PATH:/usr/local/spark/bin
```

4. Spark Master Configuration

```
$ cd /usr/local/spark/conf
$ cp spark-env.sh.template spark-env.sh
$ vim spark-env.sh
   export SPARK_MASTER_HOST='<MASTER-IP>'
   export  JAVA_HOME=<JAVA-PATH>
```

5. Workers Configuration

```
$ vim slaves
    master
    slave01
    slave02
    slave03
```

KAFKA

1. Download and install Kafka

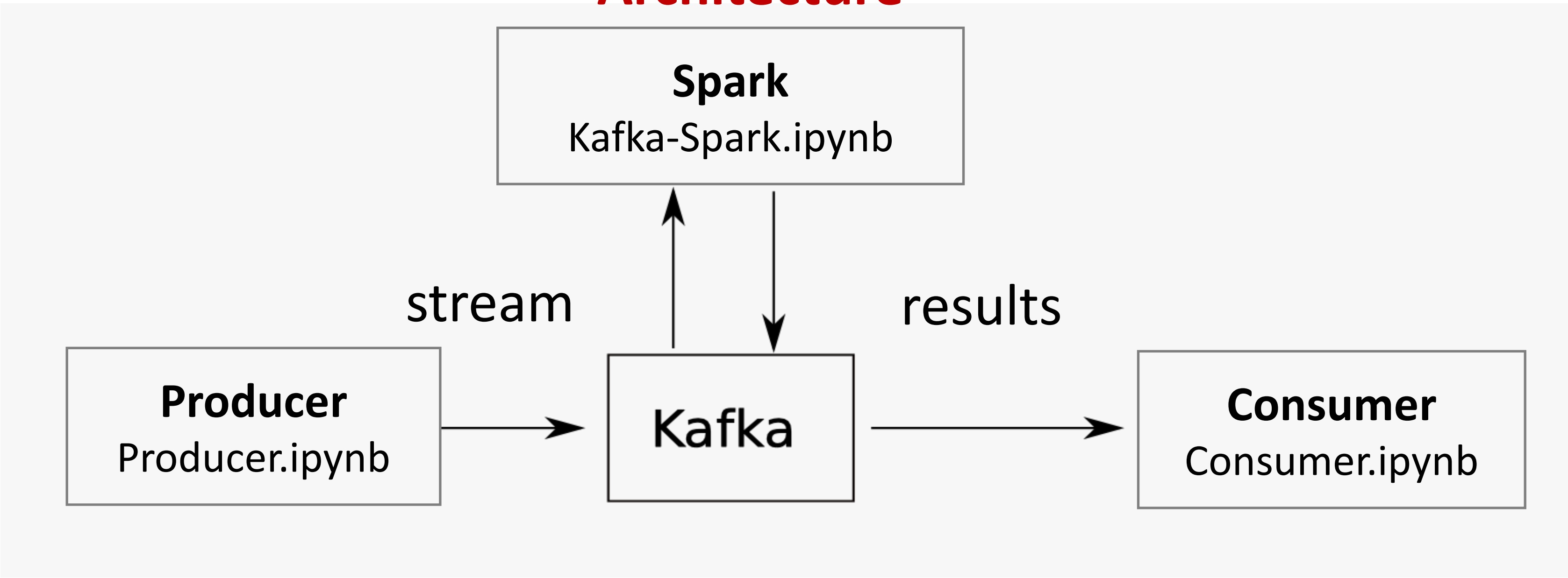
kafka_2.13-2.7.0

2. Configuration

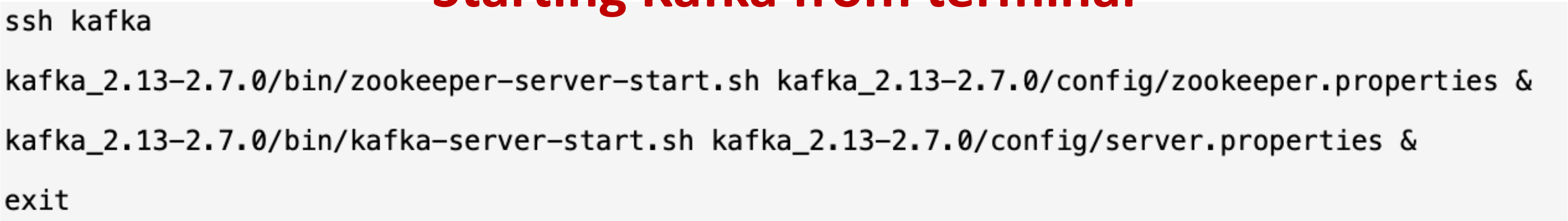
```
$ vim config/zookeeper.properties
$ vim config/server.properties
```

Kafka

Architecture



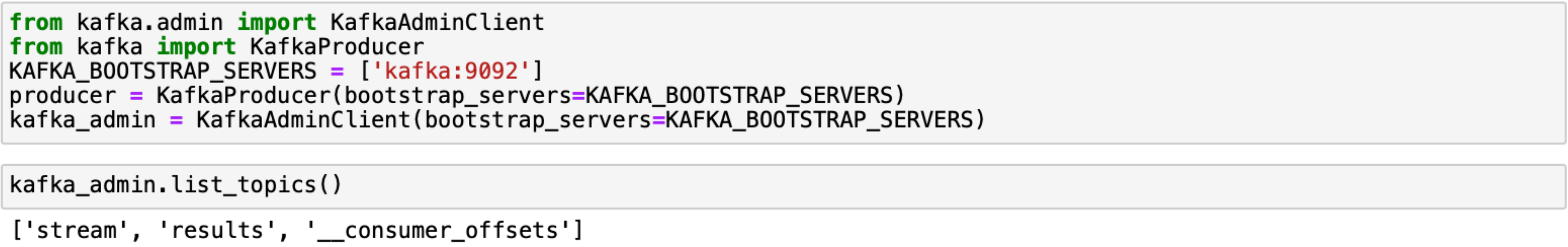
Starting Kafka from terminal



```
ssh kafka

kafka_2.13-2.7.0/bin/zookeeper-server-start.sh kafka_2.13-2.7.0/config/zookeeper.properties &

kafka_2.13-2.7.0/bin/kafka-server-start.sh kafka_2.13-2.7.0/config/server.properties &

exit
```

Kafka Producer

Producer

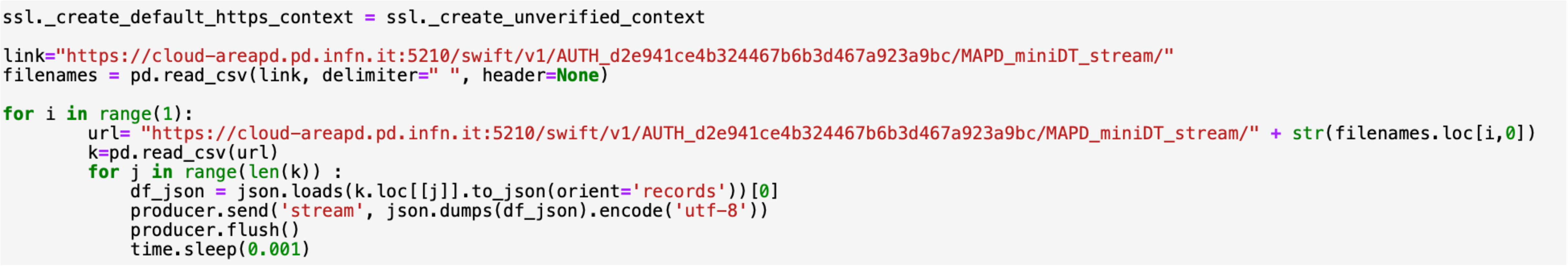


```
from kafka.admin import KafkaAdminClient
from kafka import KafkaProducer
KAFKA_BOOTSTRAP_SERVERS = ['kafka:9092']
producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS)
kafka_admin = KafkaAdminClient(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS)
```

```
kafka_admin.list_topics()
```

```
['stream', 'results', '__consumer_offsets']
```

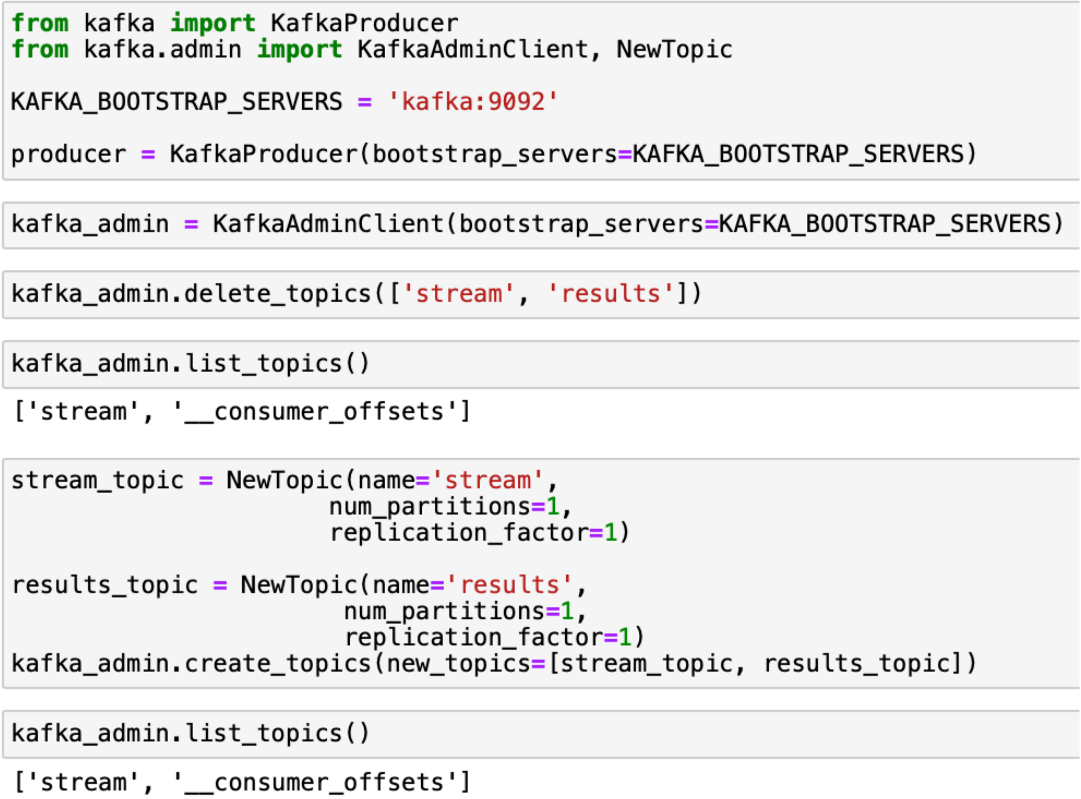
Import and sending data



```
ssl._create_default_https_context = ssl._create_unverified_context

link="https://cloud-areapd.pd.infn.it:5210/swift/v1/AUTH_d2e941ce4b324467b6b3d467a923a9bc/MAPD_miniDT_stream/"
filenames = pd.read_csv(link, delimiter=" ", header=None)

for i in range(1):
        url= "https://cloud-areapd.pd.infn.it:5210/swift/v1/AUTH_d2e941ce4b324467b6b3d467a923a9bc/MAPD_miniDT_stream/" + str(filenames.loc[i,0])
        k=pd.read_csv(url)
        for j in range(len(k)) :
            df_json = json.loads(k.loc[[j]].to_json(orient='records'))[0]
            producer.send('stream', json.dumps(df_json).encode('utf-8'))
            producer.flush()
            time.sleep(0.001)
```

Kafka - Spark

Kafka topics



```
from kafka import KafkaProducer
from kafka.admin import KafkaAdminClient, NewTopic

KAFKA_BOOTSTRAP_SERVERS = 'kafka:9092'

producer = KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS)
```

```
kafka_admin = KafkaAdminClient(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS)
```

```
kafka_admin.delete_topics(['stream', 'results'])
```

```
kafka_admin.list_topics()
```

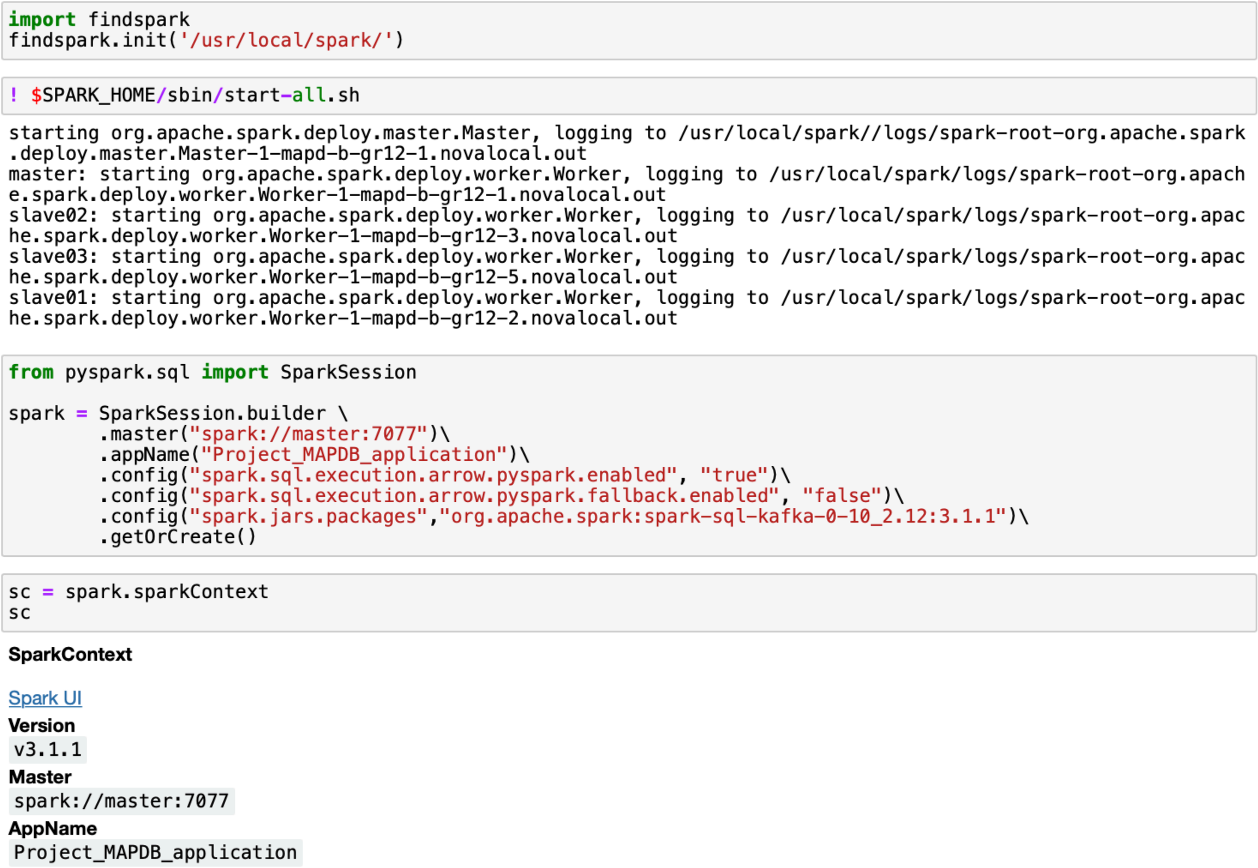
```
['stream', '__consumer_offsets']
```

```
stream_topic = NewTopic(name='stream',
                        num_partitions=1,
                        replication_factor=1)

results_topic = NewTopic(name='results',
                         num_partitions=1,
                         replication_factor=1)
kafka_admin.create_topics(new_topics=[stream_topic, results_topic])
```

```
kafka_admin.list_topics()
```

```
['stream', '__consumer_offsets']
```

Starting Spark session



```
import findspark
findspark.init('/usr/local/spark/')
```

```
! $SPARK_HOME/sbin/start-all.sh
```

```
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark//logs/spark-root-org.apache.spark
.deploy.master.Master-1-mapd-b-gr12-1.novalocal.out
master: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-root-org.apach
e.spark.deploy.worker.Worker-1-mapd-b-gr12-1.novalocal.out
slave02: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-root-org.apac
he.spark.deploy.worker.Worker-1-mapd-b-gr12-3.novalocal.out
slave03: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-root-org.apac
he.spark.deploy.worker.Worker-1-mapd-b-gr12-5.novalocal.out
slave01: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-root-org.apac
he.spark.deploy.worker.Worker-1-mapd-b-gr12-2.novalocal.out
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
        .master("spark://master:7077")\
        .appName("Project_MAPDB_application")\
        .config("spark.sql.execution.arrow.pyspark.enabled", "true")\
        .config("spark.sql.execution.arrow.pyspark.fallback.enabled", "false")\
        .config("spark.jars.packages","org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.1")\
        .getOrCreate()
```

```
sc = spark.sparkContext
sc
```

SparkContext

Spark UI

Version

`v3.1.1`

Master

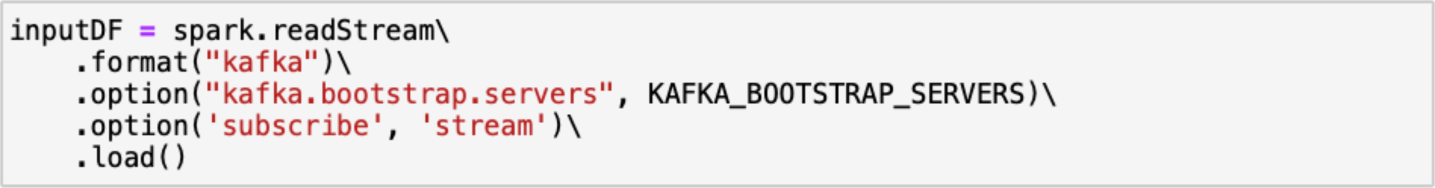
`spark://master:7077`

AppName

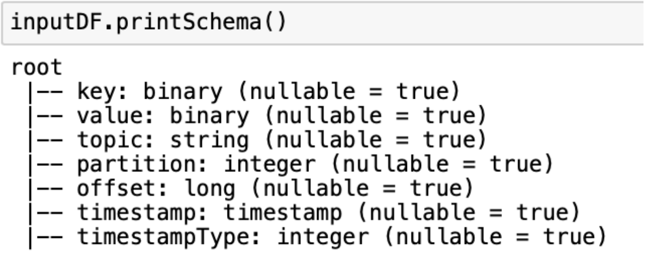
`Project_MAPDB_application`

DEGLI STUDI DI PADOVA

Kafka - Spark

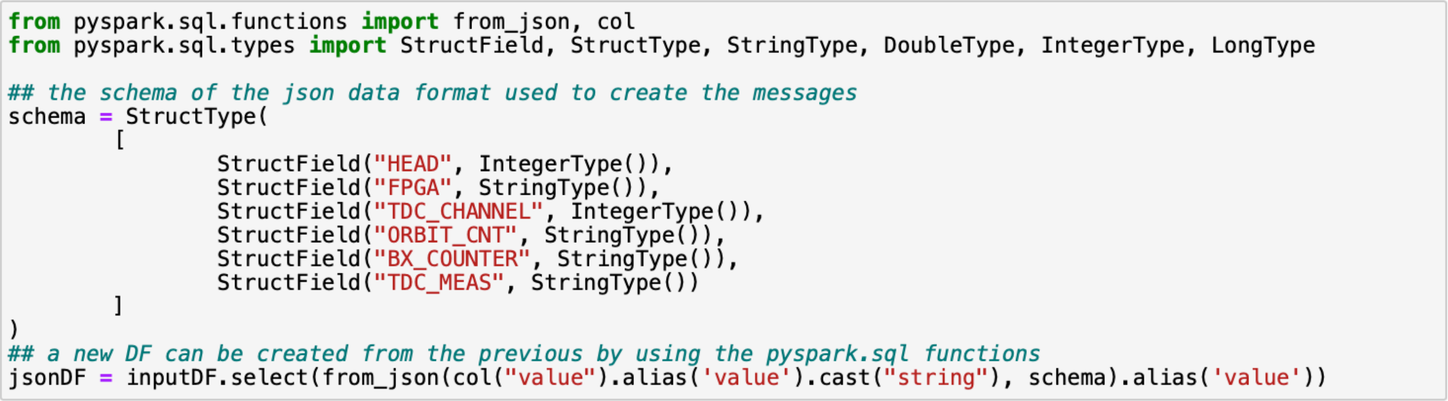


```
inputDF = spark.readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVERS)\
    .option('subscribe', 'stream')\
    .load()
```

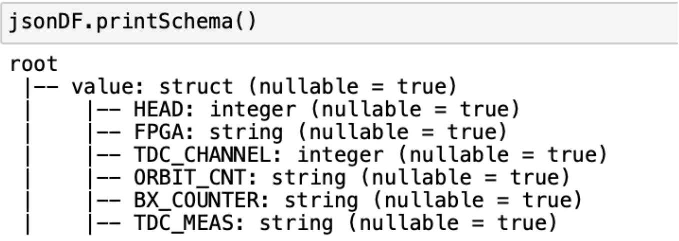


```
inputDF.printSchema()
```

```
root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

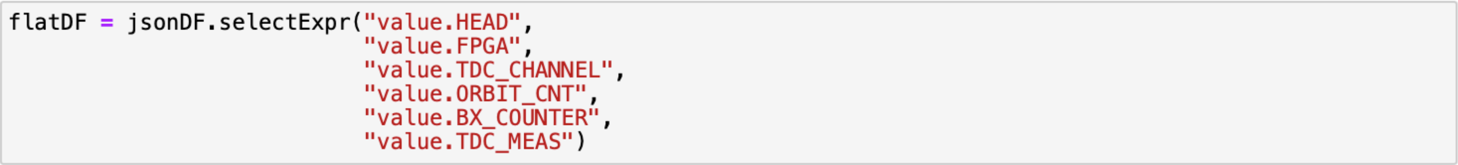


```
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructField, StructType, StringType, DoubleType, IntegerType, LongType

## the schema of the json data format used to create the messages
schema = StructType(
        [
                StructField("HEAD", IntegerType()),
                StructField("FPGA", StringType()),
                StructField("TDC_CHANNEL", IntegerType()),
                StructField("ORBIT_CNT", StringType()),
                StructField("BX_COUNTER", StringType()),
                StructField("TDC_MEAS", StringType())
        ]
)
## a new DF can be created from the previous by using the pyspark.sql functions
jsonDF = inputDF.select(from_json(col("value").alias('value').cast("string"), schema).alias('value'))
```

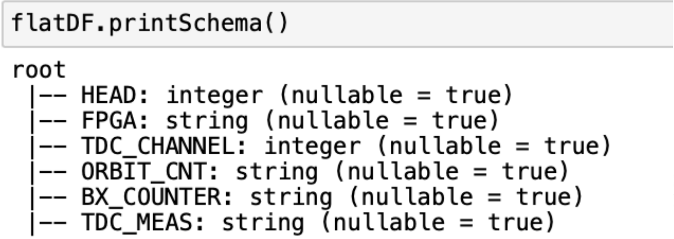


```
jsonDF.printSchema()
```

```
root
 |-- value: struct (nullable = true)
 |    |-- HEAD: integer (nullable = true)
 |    |-- FPGA: string (nullable = true)
 |    |-- TDC_CHANNEL: integer (nullable = true)
 |    |-- ORBIT_CNT: string (nullable = true)
 |    |-- BX_COUNTER: string (nullable = true)
 |    |-- TDC_MEAS: string (nullable = true)
```



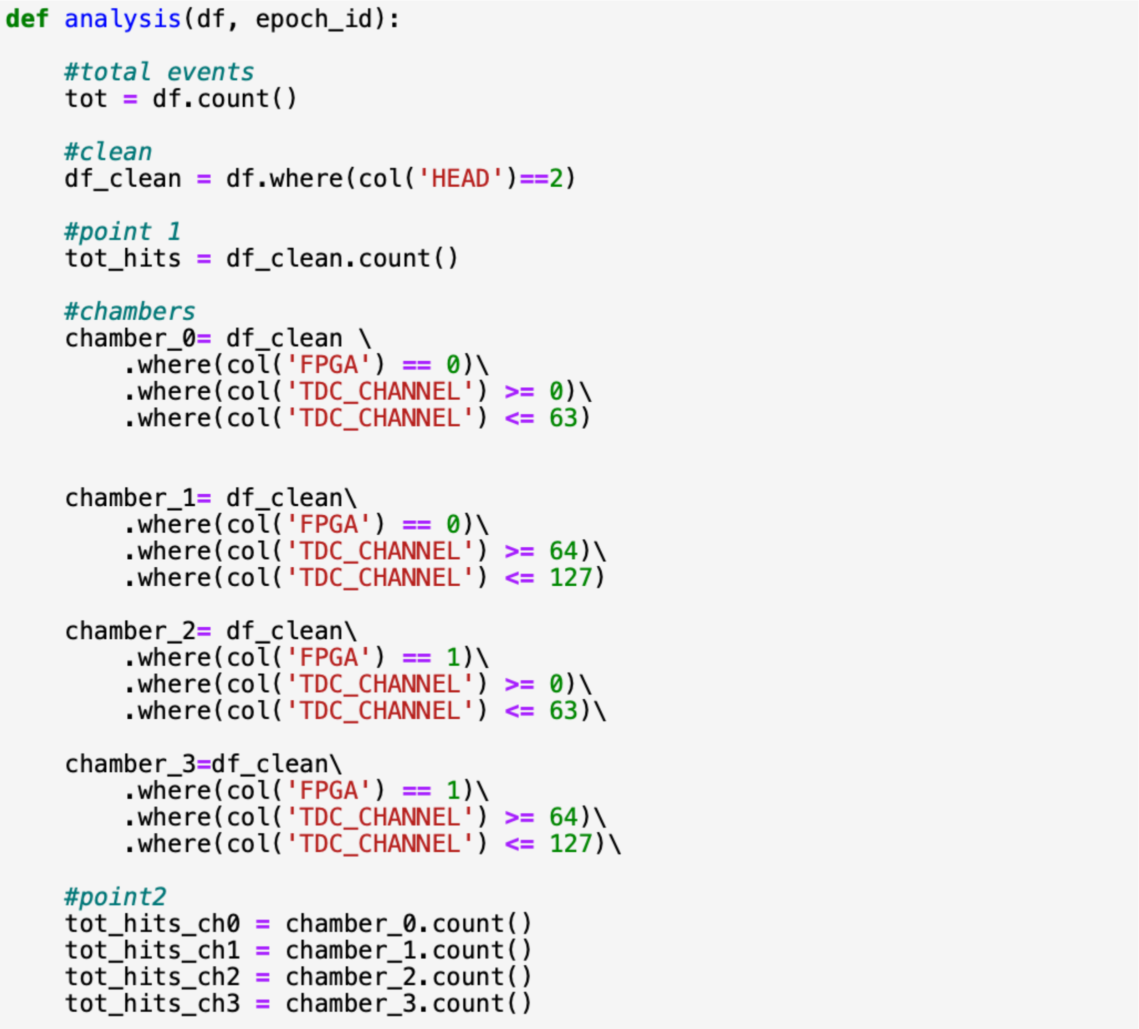
```
flatDF = jsonDF.selectExpr("value.HEAD",
                           "value.FPGA",
                           "value.TDC_CHANNEL",
                           "value.ORBIT_CNT",
                           "value.BX_COUNTER",
                           "value.TDC_MEAS")
```



```
flatDF.printSchema()
```

```
root
 |-- HEAD: integer (nullable = true)
 |-- FPGA: string (nullable = true)
 |-- TDC_CHANNEL: integer (nullable = true)
 |-- ORBIT_CNT: string (nullable = true)
 |-- BX_COUNTER: string (nullable = true)
 |-- TDC_MEAS: string (nullable = true)
```

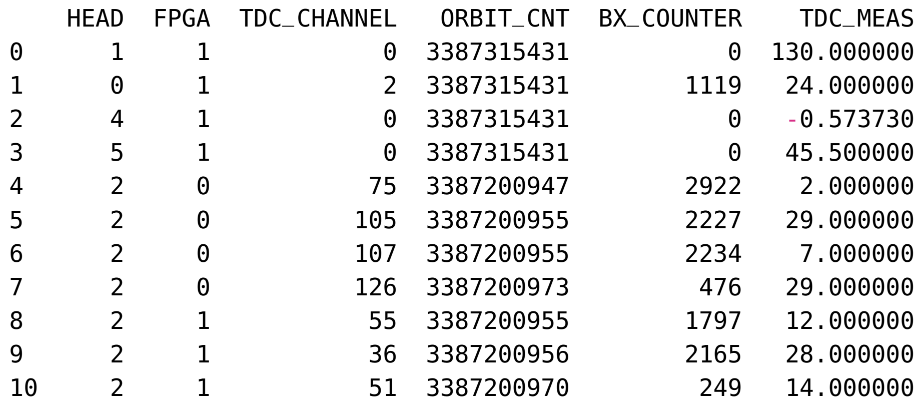
Data processing in Spark



```
def analysis(df, epoch_id):

    #total events
    tot = df.count()

    #clean
    df_clean = df.where(col('HEAD')==2)

    #point 1
    tot_hits = df_clean.count()

    #chambers
    chamber_0= df_clean \
        .where(col('FPGA') == 0)\
        .where(col('TDC_CHANNEL') >= 0)\
        .where(col('TDC_CHANNEL') <= 63)


    chamber_1= df_clean\
        .where(col('FPGA') == 0)\
        .where(col('TDC_CHANNEL') >= 64)\
        .where(col('TDC_CHANNEL') <= 127)

    chamber_2= df_clean\
        .where(col('FPGA') == 1)\
        .where(col('TDC_CHANNEL') >= 0)\
        .where(col('TDC_CHANNEL') <= 63)\

    chamber_3=df_clean\
        .where(col('FPGA') == 1)\
        .where(col('TDC_CHANNEL') >= 64)\
        .where(col('TDC_CHANNEL') <= 127)\

    #point2
    tot_hits_ch0 = chamber_0.count()
    tot_hits_ch1 = chamber_1.count()
    tot_hits_ch2 = chamber_2.count()
    tot_hits_ch3 = chamber_3.count()
```

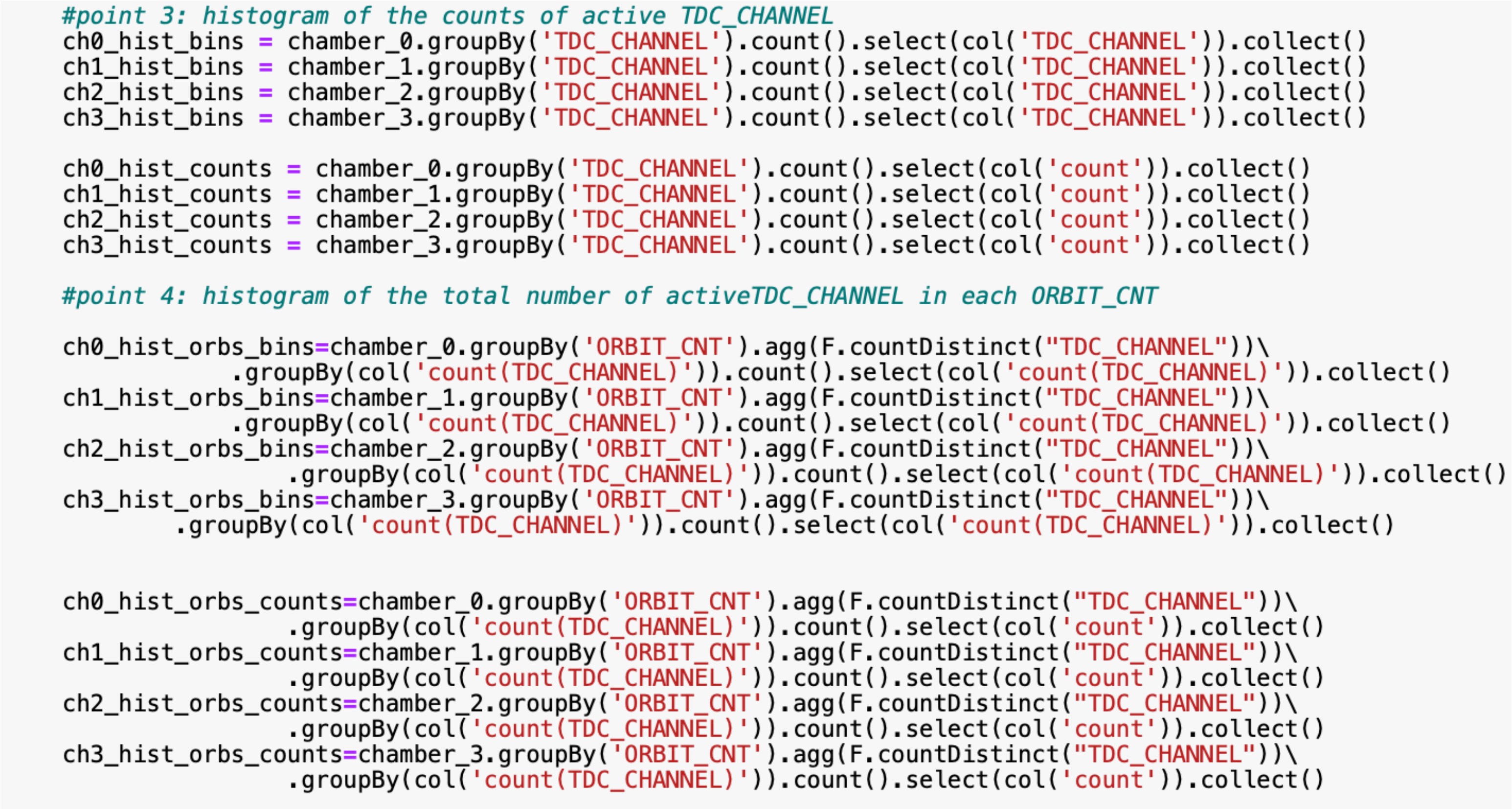
Data processing

- Filtering with condition HEAD==2
- Compute total number of processed hits, post-cleansing
- Map into chambers
- Compute total number of processed hits, post-cleansing, per chamber



```
    HEAD  FPGA  TDC_CHANNEL   ORBIT_CNT  BX_COUNTER    TDC_MEAS
0      1     1            0  3387315431           0  130.000000
1      0     1            2  3387315431        1119   24.000000
2      4     1            0  3387315431           0   -0.573730
3      5     1            0  3387315431           0   45.500000
4      2     0           75  3387200947        2922    2.000000
5      2     0          105  3387200955        2227   29.000000
6      2     0          107  3387200955        2234    7.000000
7      2     0          126  3387200973         476   29.000000
8      2     1           55  3387200955        1797   12.000000
9      2     1           36  3387200956        2165   28.000000
10     2     1           51  3387200970         249   14.000000
```

Data processing in Spark

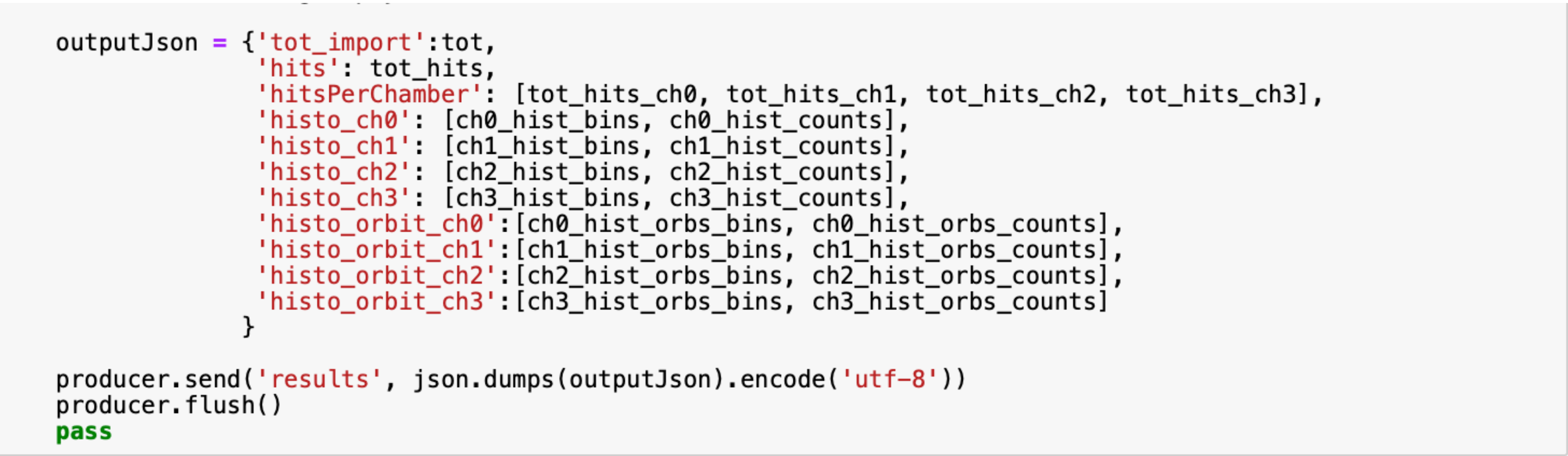


```
#point 3: histogram of the counts of active TDC_CHANNEL
ch0_hist_bins = chamber_0.groupBy('TDC_CHANNEL').count().select(col('TDC_CHANNEL')).collect()
ch1_hist_bins = chamber_1.groupBy('TDC_CHANNEL').count().select(col('TDC_CHANNEL')).collect()
ch2_hist_bins = chamber_2.groupBy('TDC_CHANNEL').count().select(col('TDC_CHANNEL')).collect()
ch3_hist_bins = chamber_3.groupBy('TDC_CHANNEL').count().select(col('TDC_CHANNEL')).collect()

ch0_hist_counts = chamber_0.groupBy('TDC_CHANNEL').count().select(col('count')).collect()
ch1_hist_counts = chamber_1.groupBy('TDC_CHANNEL').count().select(col('count')).collect()
ch2_hist_counts = chamber_2.groupBy('TDC_CHANNEL').count().select(col('count')).collect()
ch3_hist_counts = chamber_3.groupBy('TDC_CHANNEL').count().select(col('count')).collect()

#point 4: histogram of the total number of activeTDC_CHANNEL in each ORBIT_CNT

ch0_hist_orbs_bins=chamber_0.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
            .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count(TDC_CHANNEL)')).collect()
ch1_hist_orbs_bins=chamber_1.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
            .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count(TDC_CHANNEL)')).collect()
ch2_hist_orbs_bins=chamber_2.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
                .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count(TDC_CHANNEL)')).collect()
ch3_hist_orbs_bins=chamber_3.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
        .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count(TDC_CHANNEL)')).collect()


ch0_hist_orbs_counts=chamber_0.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
                .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count')).collect()
ch1_hist_orbs_counts=chamber_1.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
                .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count')).collect()
ch2_hist_orbs_counts=chamber_2.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
                .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count')).collect()
ch3_hist_orbs_counts=chamber_3.groupBy('ORBIT_CNT').agg(F.countDistinct("TDC_CHANNEL"))\
                .groupBy(col('count(TDC_CHANNEL)')).count().select(col('count')).collect()
```

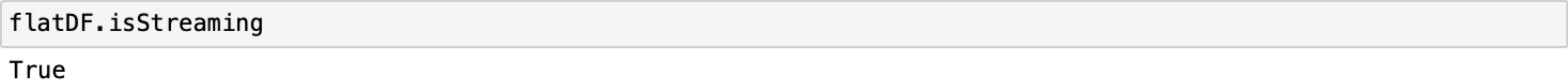
Data processing

- Histogram of the counts of active TDC_CHANNEL per chamber
- Histogram of the total number of active TDC_CHANNEL per chamber in each ORBIT_CNT

Streaming of results

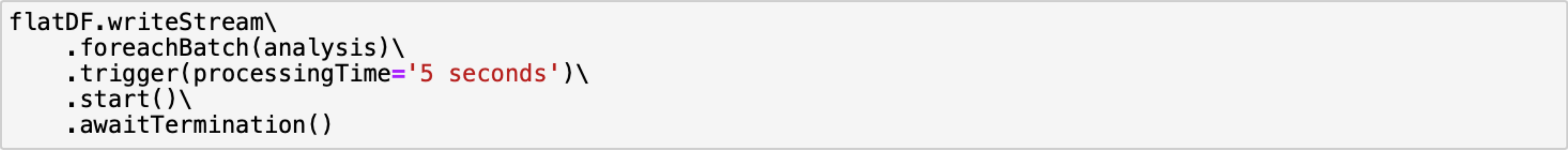


```
    outputJson = {'tot_import':tot,
                  'hits': tot_hits,
                  'hitsPerChamber': [tot_hits_ch0, tot_hits_ch1, tot_hits_ch2, tot_hits_ch3],
                  'histo_ch0': [ch0_hist_bins, ch0_hist_counts],
                  'histo_ch1': [ch1_hist_bins, ch1_hist_counts],
                  'histo_ch2': [ch2_hist_bins, ch2_hist_counts],
                  'histo_ch3': [ch3_hist_bins, ch3_hist_counts],
                  'histo_orbit_ch0':[ch0_hist_orbs_bins, ch0_hist_orbs_counts],
                  'histo_orbit_ch1':[ch1_hist_orbs_bins, ch1_hist_orbs_counts],
                  'histo_orbit_ch2':[ch2_hist_orbs_bins, ch2_hist_orbs_counts],
                  'histo_orbit_ch3':[ch3_hist_orbs_bins, ch3_hist_orbs_counts]
                 }

    producer.send('results', json.dumps(outputJson).encode('utf-8'))
    producer.flush()
    pass
```



```
flatDF.isStreaming
```

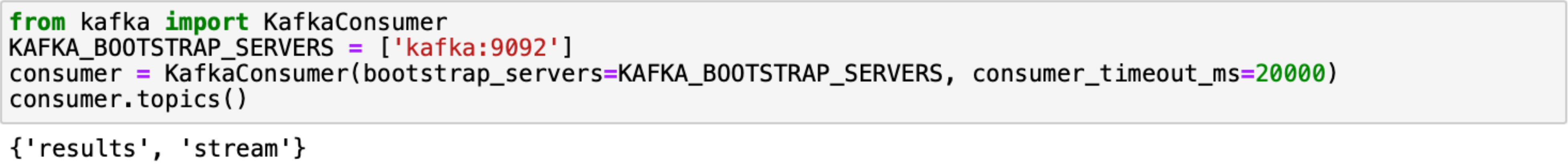
```
True
```



```
flatDF.writeStream\
    .foreachBatch(analysis)\
    .trigger(processingTime='5 seconds')\
    .start()\
    .awaitTermination()
```

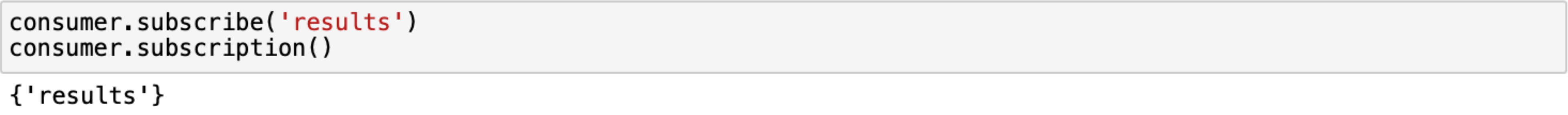
Kafka Consumer

Kafka consumer



```
from kafka import KafkaConsumer
KAFKA_BOOTSTRAP_SERVERS = ['kafka:9092']
consumer = KafkaConsumer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS, consumer_timeout_ms=20000)
consumer.topics()
```

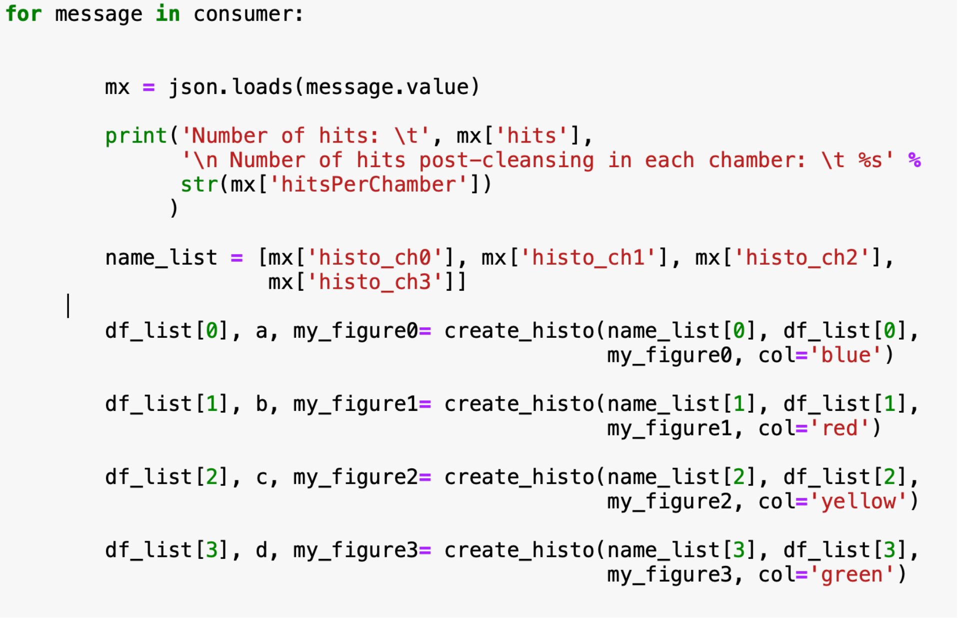
```
{'results', 'stream'}
```



```
consumer.subscribe('results')
consumer.subscription()
```

```
{'results'}
```

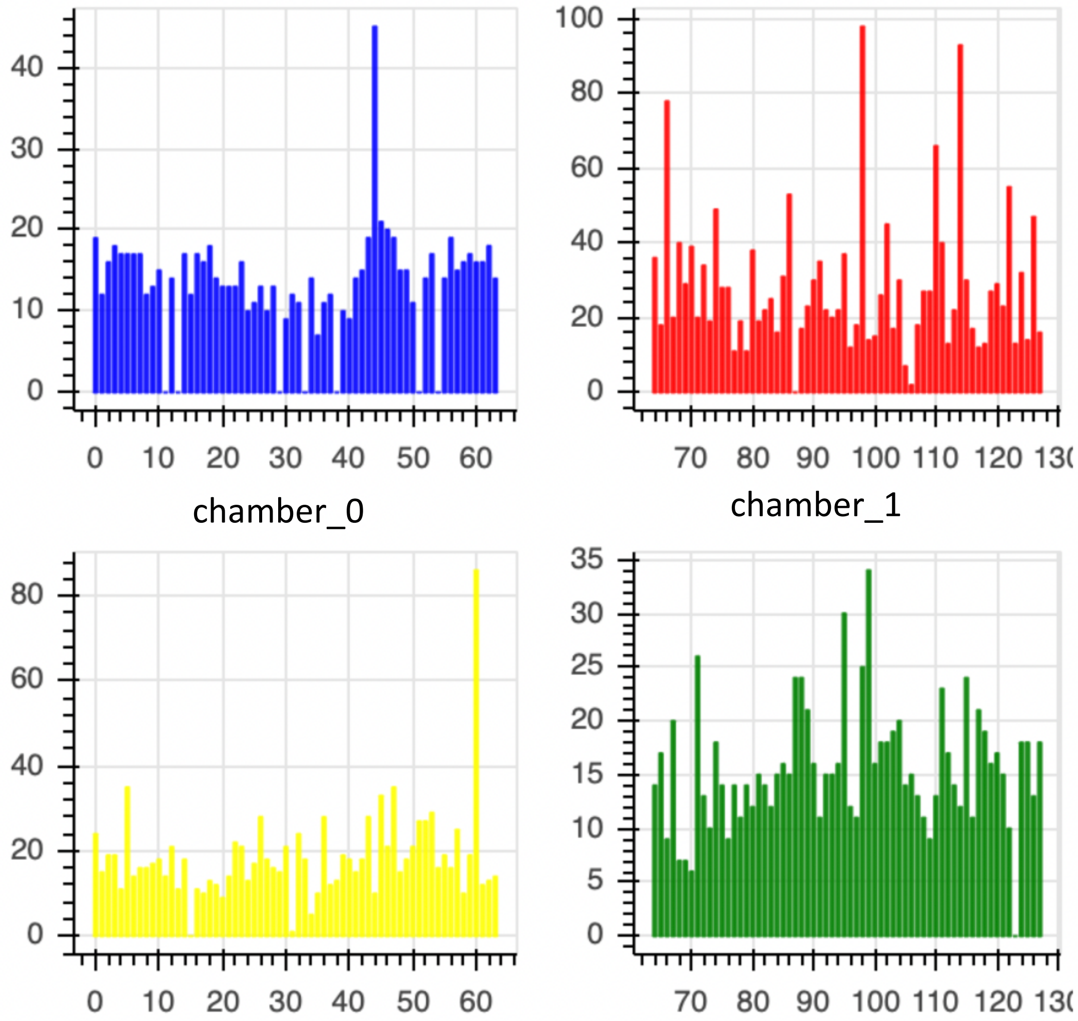
Reading results and plots



```
for message in consumer:


        mx = json.loads(message.value)

        print('Number of hits: \t', mx['hits'],
              '\n Number of hits post-cleansing in each chamber: \t %s' %
              str(mx['hitsPerChamber'])
             )

        name_list = [mx['histo_ch0'], mx['histo_ch1'], mx['histo_ch2'],
                     mx['histo_ch3']]

        df_list[0], a, my_figure0= create_histo(name_list[0], df_list[0],
                                                my_figure0, col='blue')

        df_list[1], b, my_figure1= create_histo(name_list[1], df_list[1],
                                                my_figure1, col='red')

        df_list[2], c, my_figure2= create_histo(name_list[2], df_list[2],
                                                my_figure2, col='yellow')

        df_list[3], d, my_figure3= create_histo(name_list[3], df_list[3],
                                                my_figure3, col='green')
```

Dashboard

Plots of the counts of active TDC_CHANNEL



chamber_0

chamber_1

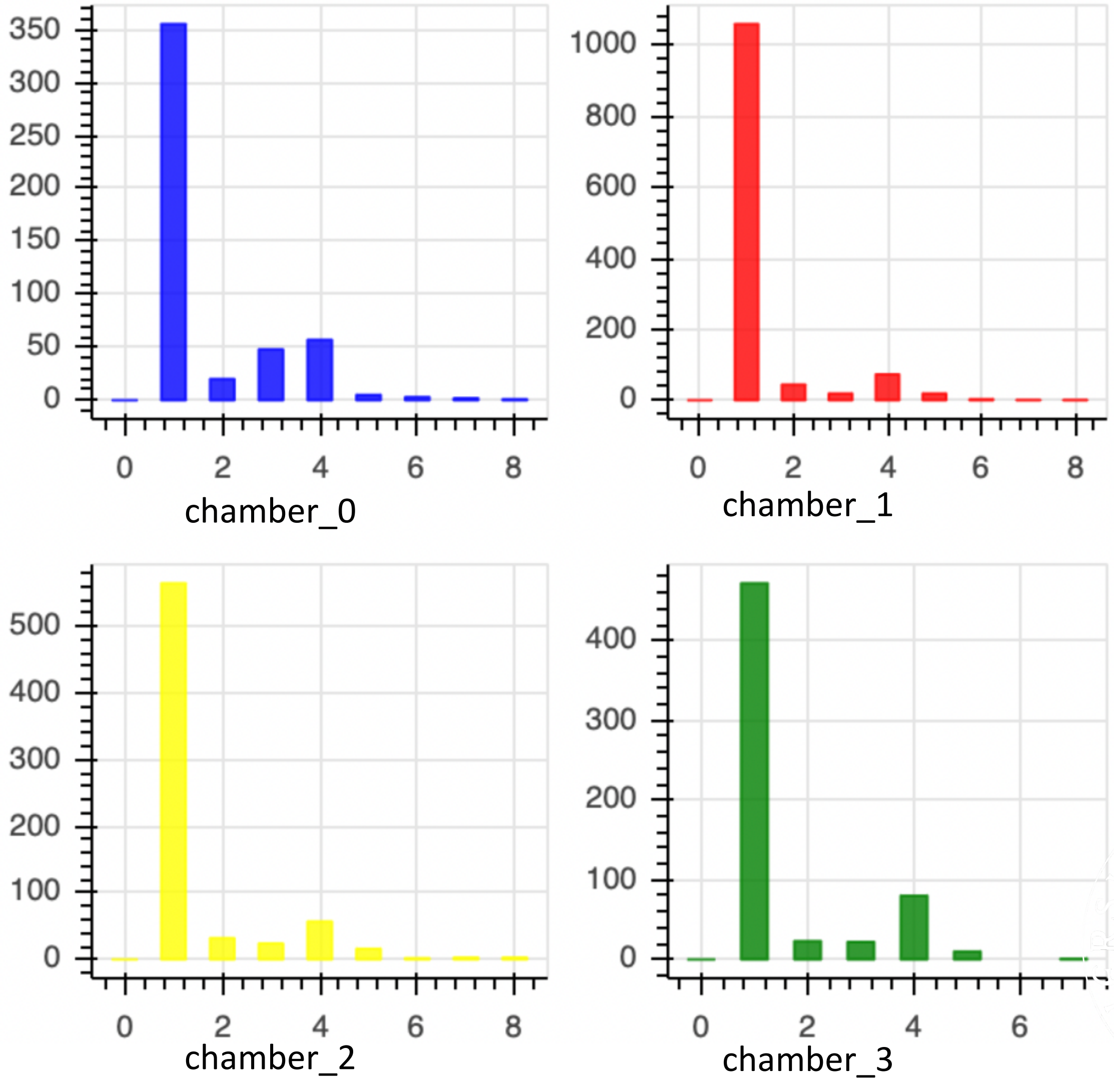
chamber_2

chamber_3

Università degli Studi di Padova

Dashboard

Plots of the total number of active TDC_CHANNEL in each ORBIT_CNT



chamber_0

chamber_1

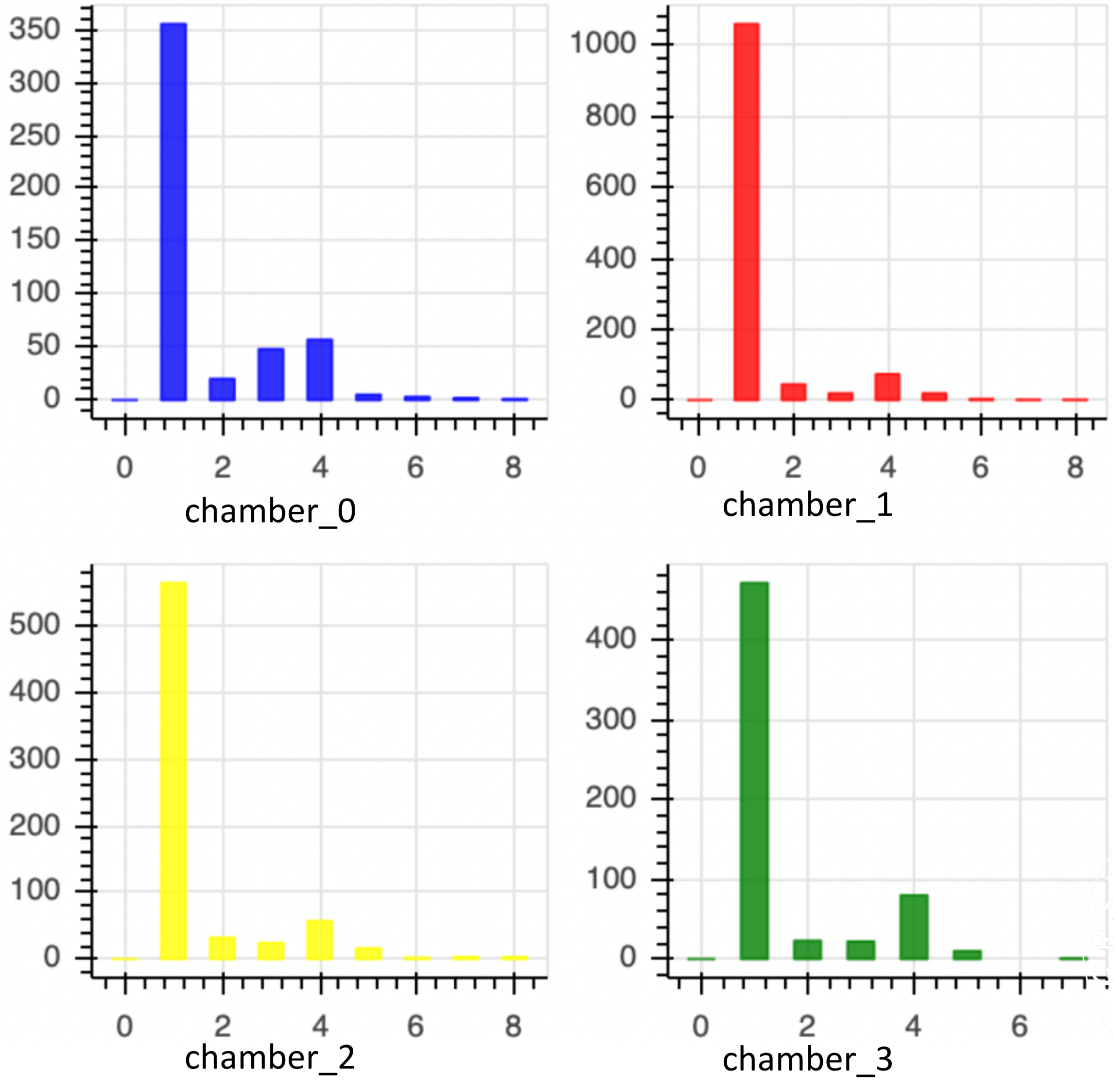
chamber_2

chamber_3

Università degli Studi di Padova

Dashboard

Plots of the total number of active TDC_CHANNEL in each ORBIT_CNT



chamber_0

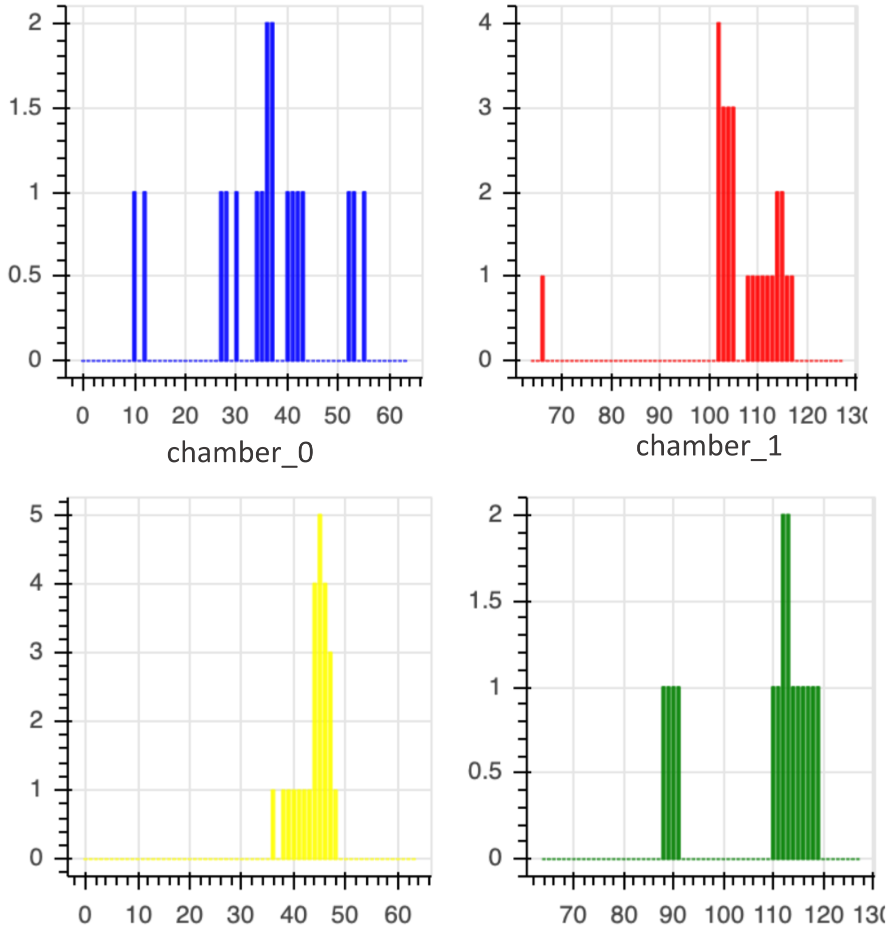
chamber_1

chamber_2

chamber_3

Dashboard

Plots of the counts of active TDC_CHANNEL
only for those orbits with at least one scintillator signal in it



chamber_0

chamber_1

chamber_2

chamber_3

Università
degli Studi
di Padova

Conclusion

To sum up:

- We set up a Spark and Kafka cluster with four workers and one broker on a multi-node VM cluster
- We implemented a streaming application using Kafka and Spark Structured Streaming
- We processed the data of the Drift Tubes detectors extracting the information of interest
- We were able to plot our results in a live updating dashboard using Bokeh

Thanks for the attention!