Internet of Things

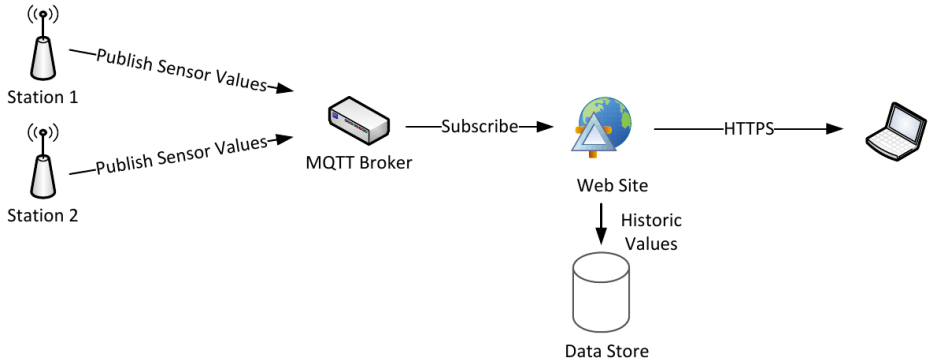
IoT Data Analytics

Ioannis Chatzigiannakis

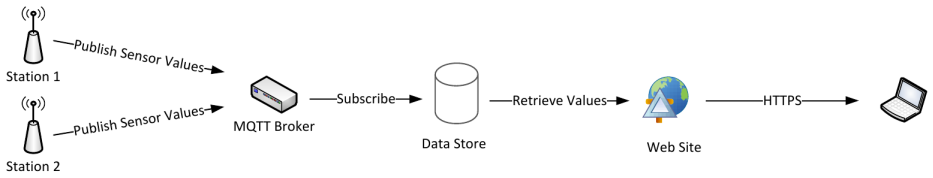
Sapienza University of Rome
Department of Computer, Control, and Management Engineering (DIAG)

Lecture 8:
IoT Data Analytics

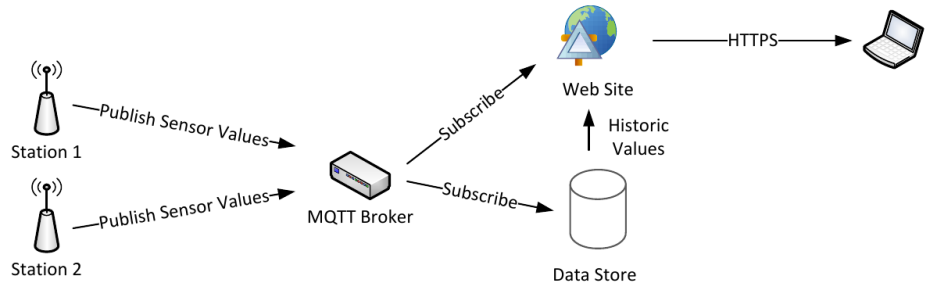
Cloud-based Architecture of 1st Assignment



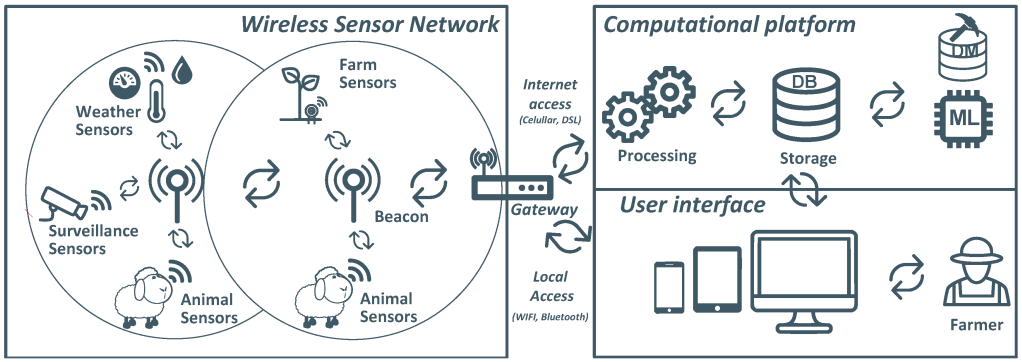
Cloud-based Architecture of 1st Assignment



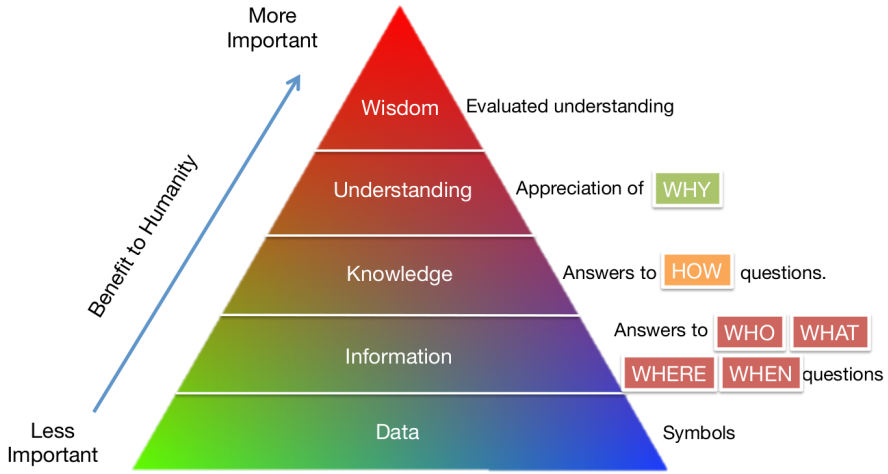
Cloud-based Architecture of 1st Assignment



Cloud-Based Architecture – Main Compoments



The value of IoT

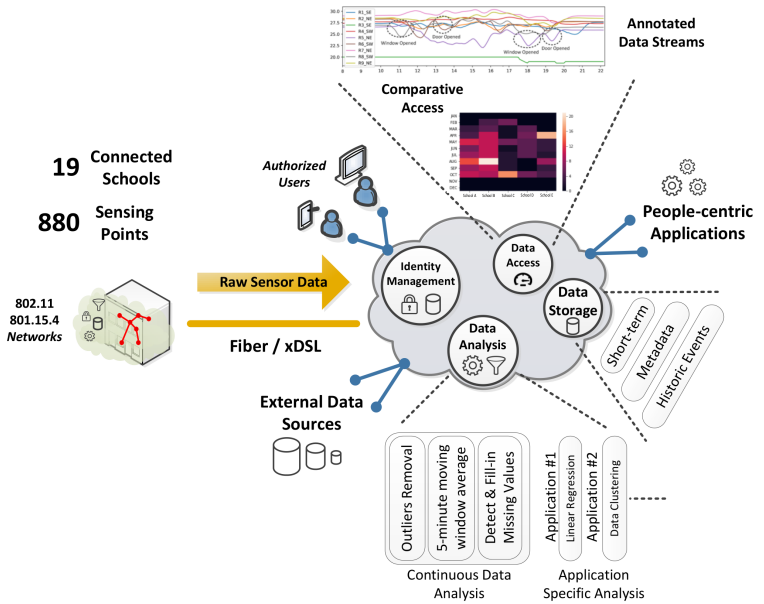


It's all in the Cloud

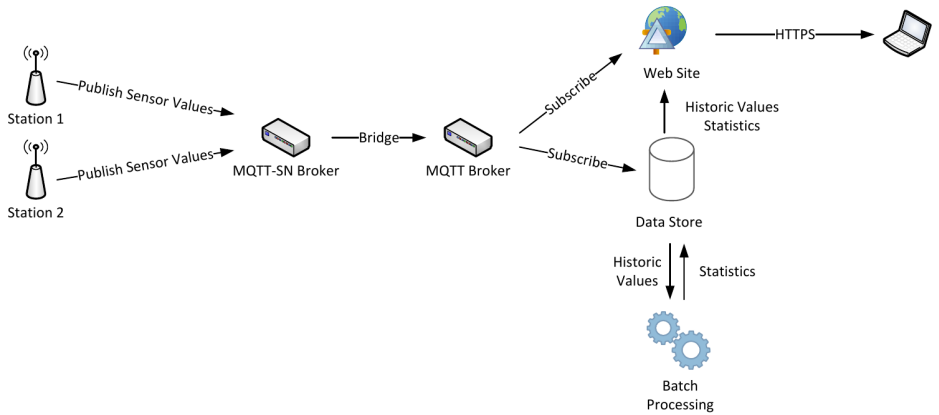
Data Processing in a Cloud-based Architecture

- We wish to process the data arriving from the sensors.
- Produce statistics for predefine period of time:
  - Every Hour
  - Every Day
  - Every Week
  - ...
- Carry out various data mining tasks:
  - Identify anomalies
  - Identify seasonality of values
  - Identify corellation between values
  - ...

GAIA: Cloud-based Smart School Architecture



Batch-based Data Processing



Batch-based Data Processing

Batch Processing

All Input

Hadoop, Spark, Flink

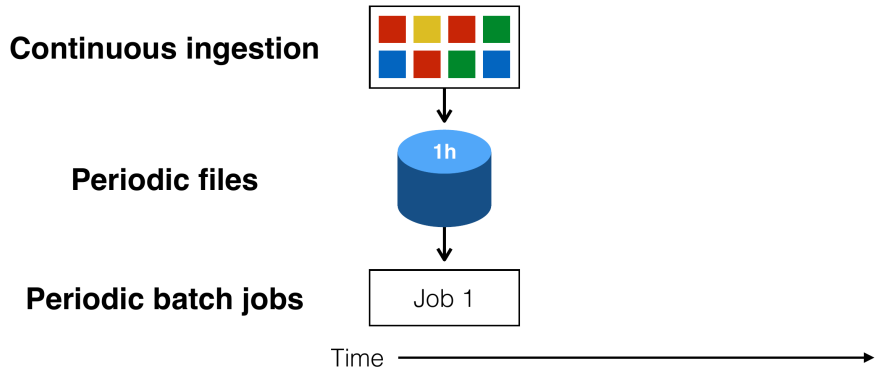
All Output

A Code Example

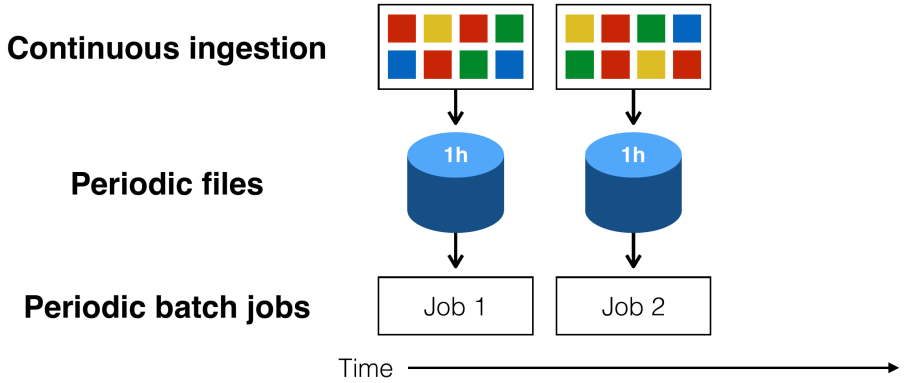
```
DataSet<ColorEvent> counts = env
  .readFile("MM-dd.csv")
  .groupBy("color")
  .count();
```

- Several frameworks exist for batch-based processing.
- Generic code example following Spark/Flink style.
- Here data is assumed to arrive as a Data Set in CSV format.
- Alternative: retrieve data from database using a query.
- We carry out various transformations (group, filter).
- We compute various aggregates (count, min, max . . .).

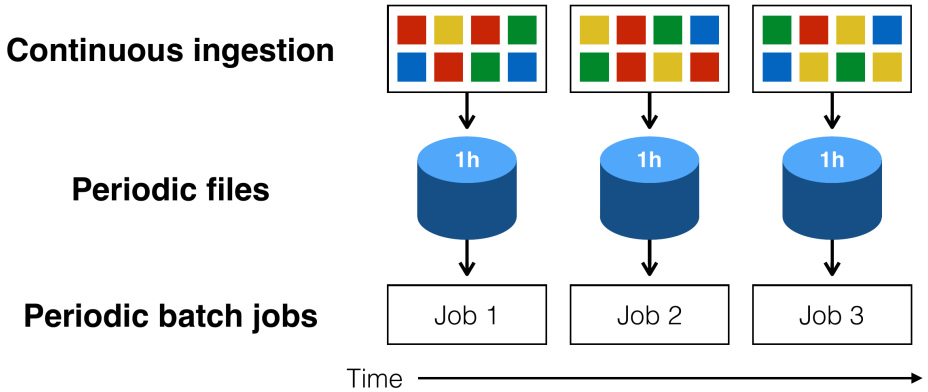
Continuous Counting



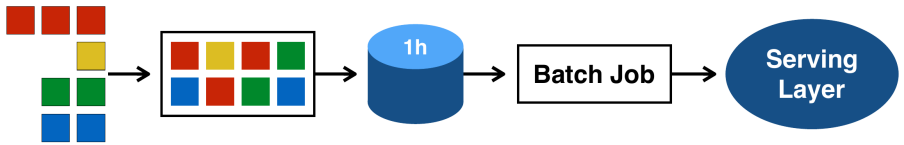
Continuous Counting



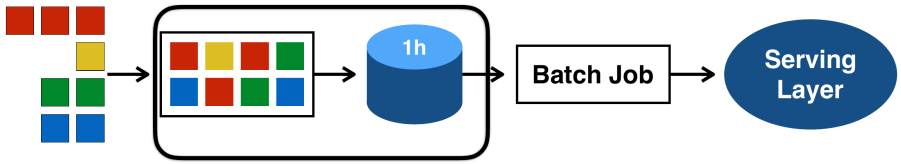
Continuous Counting



Moving Parts

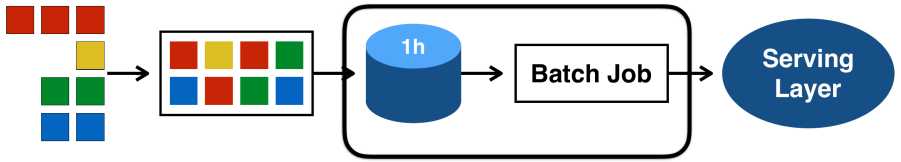


Moving Parts



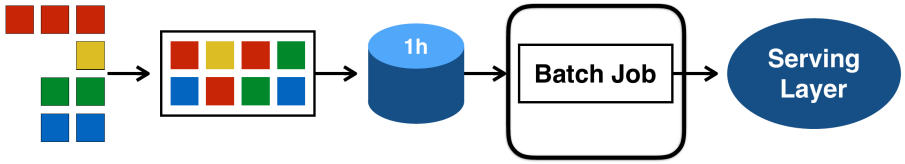
Data loading into HDFS
(e.g. Flume)

Moving Parts



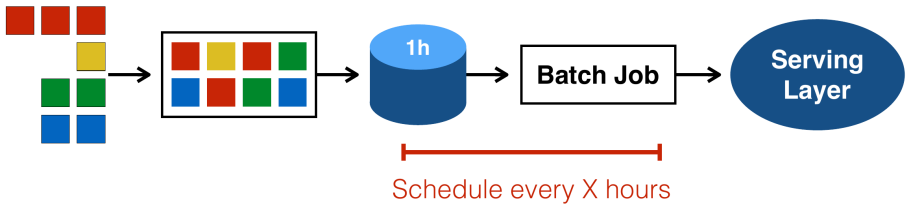
Periodic job scheduler
(e.g. Oozie)

Moving Parts



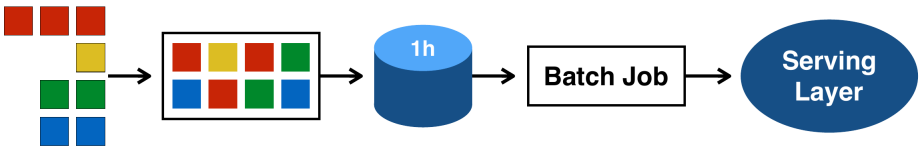
Batch processor
(e.g. Hadoop, Spark, Flink)

High Latency



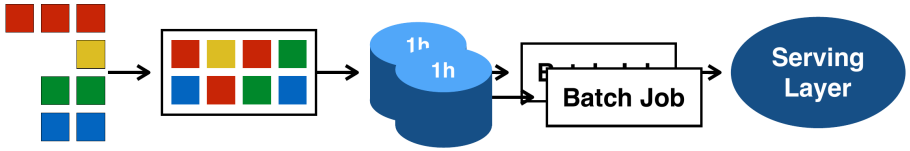
- Latency from event to serving layer usually in the range of hours.

Implicity Treatment of Time



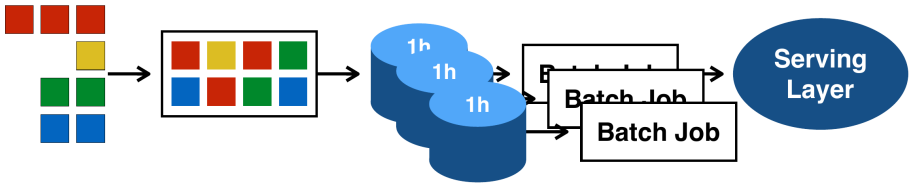
- Time is treated outside our application.
- Part of administrative tasks.

Implicity Treatment of Time



- Time is treated outside our application.
- Part of administrative tasks.

Implicity Treatment of Time



- Time is treated outside our application.
- Part of administrative tasks.

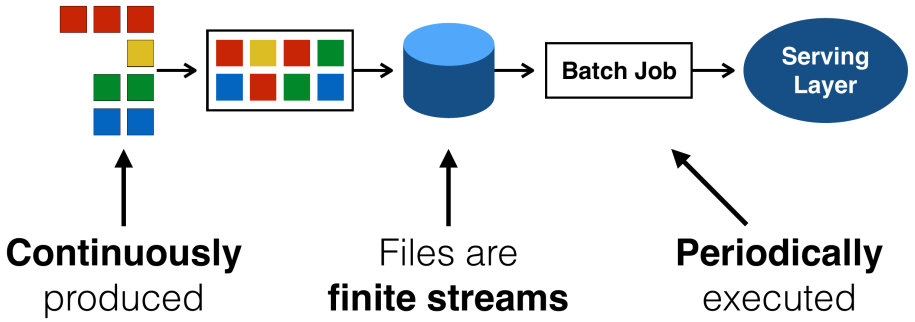
Implicity Treatment of Time

```
DataSet<ColorEvent> counts = env
  .readFile("MM-dd.csv")
  .groupBy("color")
  .count();
```

Time is **implicit** in input file

- Time is treated outside our application.
- Part of administrative tasks.

Streaming over Batch



Stream-based Data Processing

- Until now, stream processors were less mature than their batch counterparts. This led to:
  - in-house solutions,
  - abuse of batch processors,
  - Lambda architectures
- This is no longer needed with new generation stream processors like Flink, Spark ....
- Stream-based processing is enabling the obvious: continuous processing on data that is continuously produced.

Why Streaming?

- Monitor data and react in real time.
- Implement robust continuous applications.
- Adopt a decentralized architecture.
- Consolidate analytics infrastructure.

Continuous Analytics

- A production data application that needs to be live 24/7 feeding other systems (perhaps customer-facing)
- Need to be efficient, consistent, correct, and manageable
- Stream processing is a great way to implement continuous applications robustly

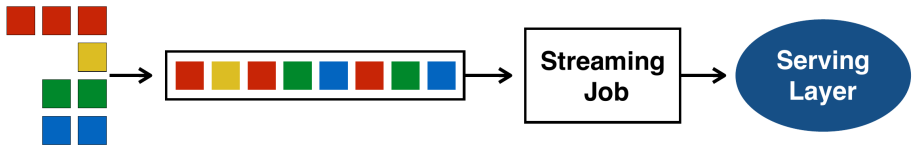
Streaming vs Real-time

- Streaming != Real-time
- E.g., streaming that is not real time: continuous applications with large windows
- E.g., real-time that is not streaming: very fast data warehousing queries
- However: streaming applications can be fast

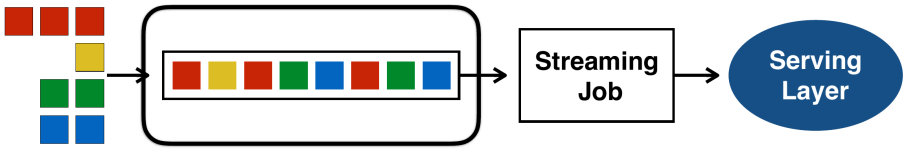
When and why does this matter?

- Immediate reaction to life
  - E.g., generate alerts on anomaly/pattern/special event
- Avoid unnecessary tradeoffs
  - Even if application is not latency-critical
  - With Flink you do not pay a price for latency!

Streaming all the Way



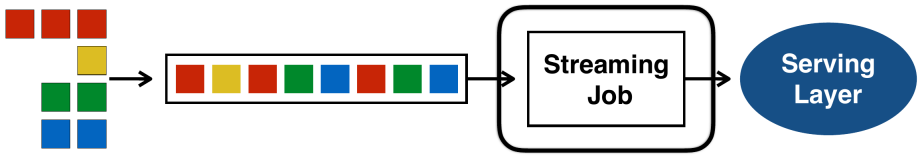
Streaming all the Way



Message Queue
(e.g. Apache Kafka)

Durability and Replay

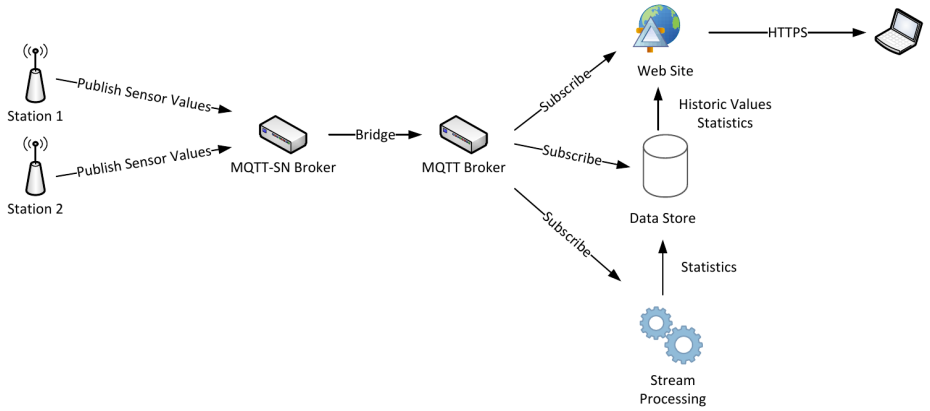
Streaming all the Way



Stream Processor
(e.g. Apache Flink)

Consistent Processing

Stream-based Data Processing



Windowing

Aggregates on **streams**
are scoped by **windows**

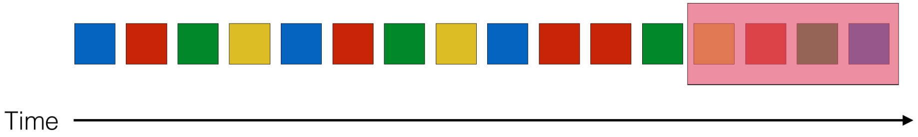
Time-driven
e.g. last X minutes

Data-driven
e.g. last X records

Tumbling Windows (No Overlap)

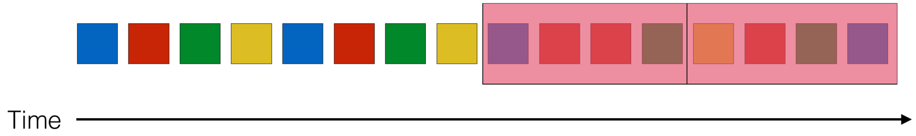
- Example: Average value over the last 5 minutes.
- Maximum value over the last 100 readings.

Tumbling Windows (No Overlap)



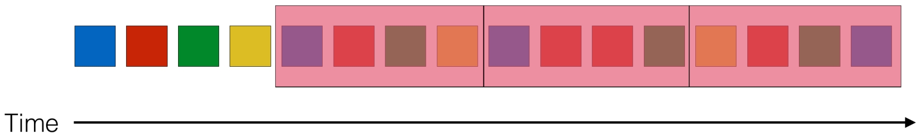
- Example: Average value over the last 5 minutes.
- Maximum value over the last 100 readings.

Tumbling Windows (No Overlap)



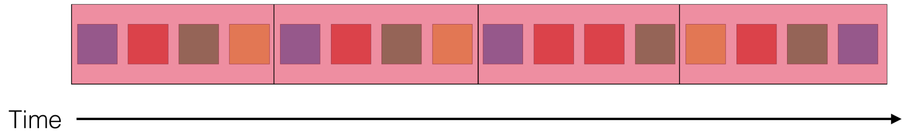
- Example: Average value over the last 5 minutes.
- Maximum value over the last 100 readings.

Tumbling Windows (No Overlap)



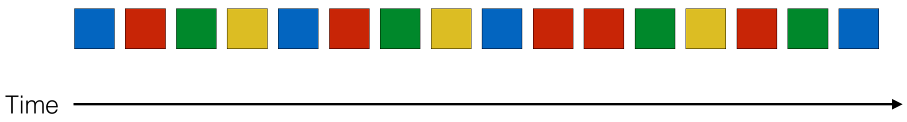
- Example: Average value over the last 5 minutes.
- Maximum value over the last 100 readings.

Tumbling Windows (No Overlap)



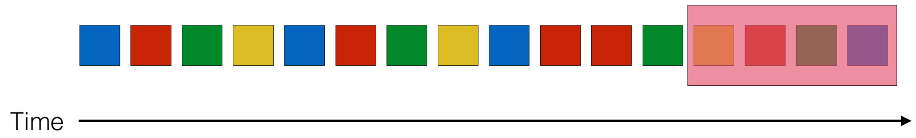
- Example: Average value over the last 5 minutes.
- Maximum value over the last 100 readings.

Sliding Windows (With Overlap)



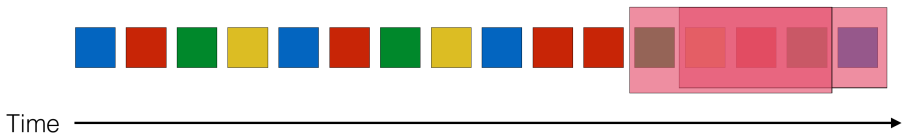
- Example: Average value over the last 5 minutes, updated each minute.
- Maximum value over the last 100 readings, updated every 10 readings.

Sliding Windows (With Overlap)



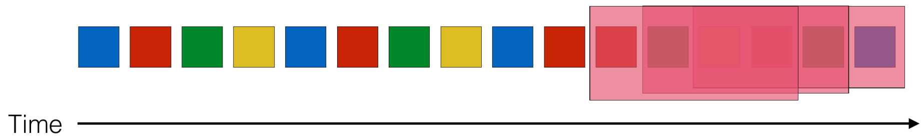
- Example: Average value over the last 5 minutes, updated each minute.
- Maximum value over the last 100 readings, updated every 10 readings.

Sliding Windows (With Overlap)



- Example: Average value over the last 5 minutes, updated each minute.
- Maximum value over the last 100 readings, updated every 10 readings.

Sliding Windows (With Overlap)



- Example: Average value over the last 5 minutes, updated each minute.
- Maximum value over the last 100 readings, updated every 10 readings.

Sliding Windows (With Overlap)

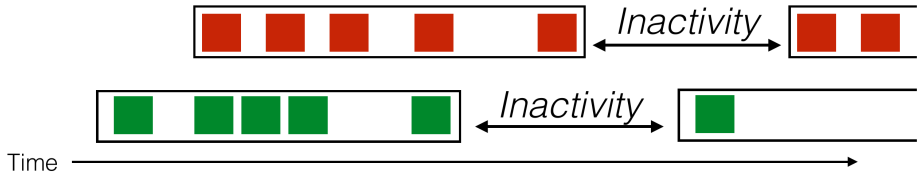
- Example: Average value over the last 5 minutes, updated each minute.
- Maximum value over the last 100 readings, updated every 10 readings.

Explicity Handling of Time

```
DataStream<ColorEvent> counts = env
  .addSource(new KafkaConsumer(…))
  .keyBy("color")
  .timeWindow(Time.minutes(60))
  .apply(new CountPerWindow());
```

Time is **explicit** in **your program**

Session Windows



- Sessions close after period of inactivity.
- Example: Compute Average from first value until connection time-out or last value.

Session Windows

```
DataStream<ColorEvent> counts = env
  .addSource(new KafkaConsumer(…))
  .keyBy("color")
  .window(EventTimeSessionWindows
    .withGap(Time.minutes(10))
  .apply(new CountPerWindow());
```

Notions of Time

Event Time

Time when event happened.

12:23 am

Notions of Time

Event Time

Time when event happened.

12:23 am

1:37 pm

Processing Time

Time measured by system clock

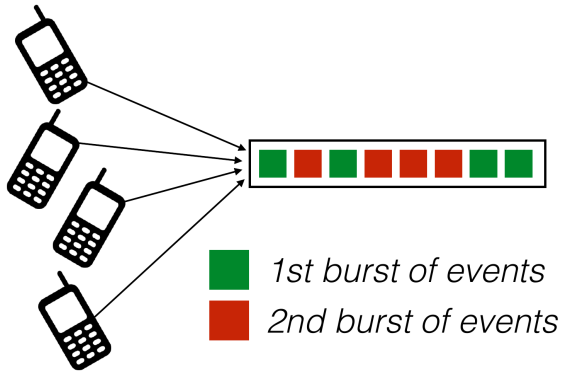
Out of Order Events

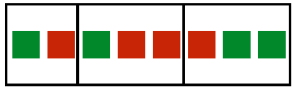
Event Time

Episode IV	Episode V	Episode VI	Episode I	Episode II	Episode III	Episode VII
1977	1980	1983	1999	2002	2005	2015

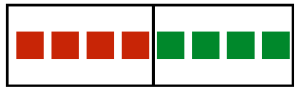
Processing Time

Out of Order Events





Processing Time Windows



Event Time Windows

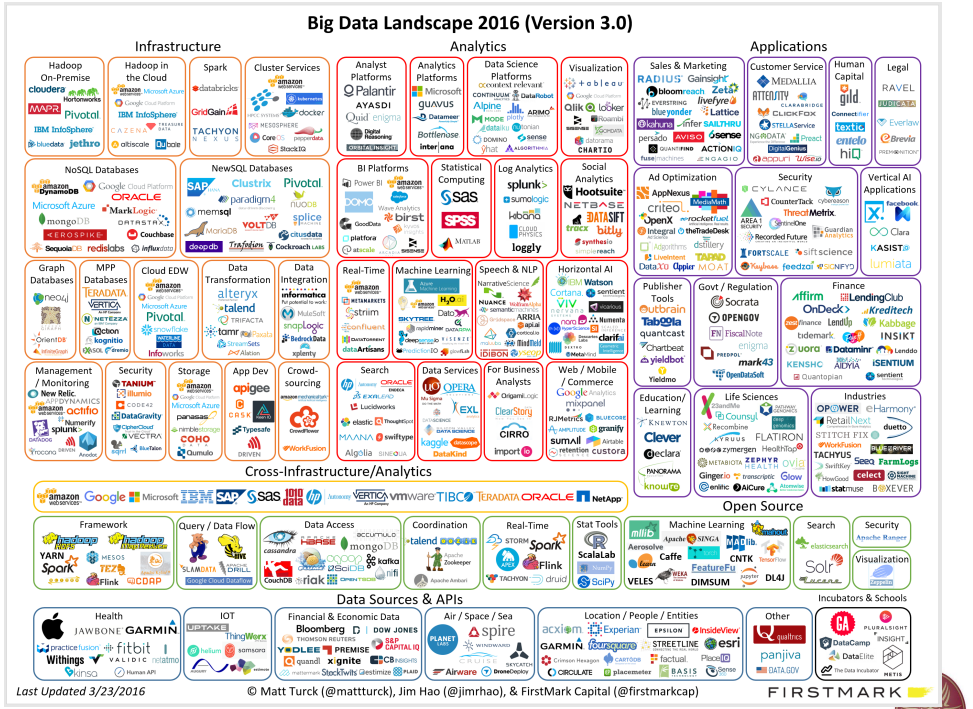
Notions of Time

```
env.setStreamTimeCharacteristic(
  TimeCharacteristic.EventTime);

DataStream<ColorEvent> counts = env
  ...
  .timeWindow(Time.minutes(60))
  .apply(new CountPerWindow());
```

Apache Flink

- Open source.
- Started in 2009 in Berlin.
- In Apache incubator since 2014.
- Fast, general purpose distributed data processing system.
- Supports batch and stream processing.
- Ready to use.

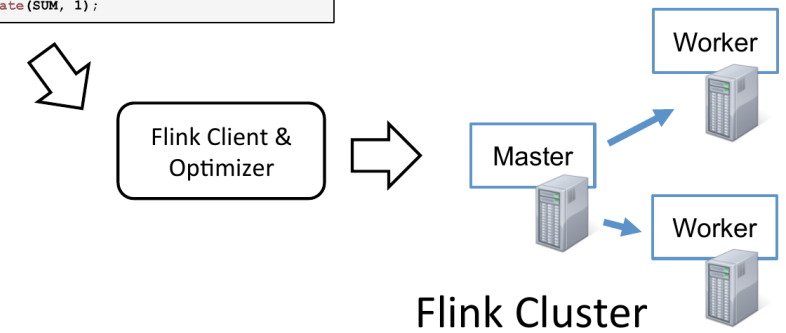


An Example

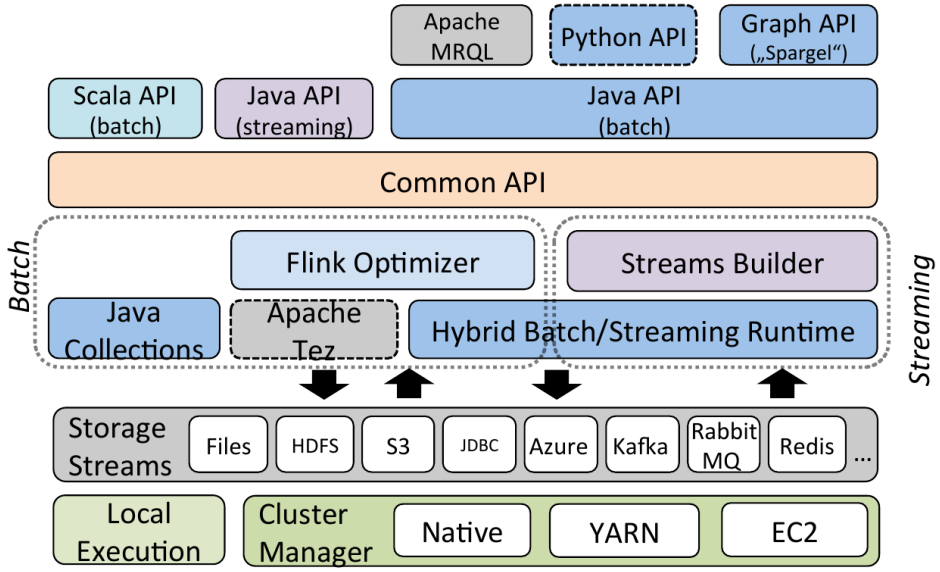
Analytical Program

```
DataSet<String> text = env.readTextFile(input);

DataSet<Tuple2<String, Integer>> result = text
        .flatMap((str, out) -> {
            for (String token : value.split("\\W")) {
                out.collect(new Tuple2<>(token, 1));
            }
        })
        .groupBy(0)
        .aggregate(SUM, 1);
```



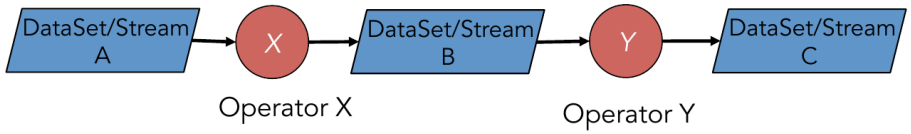
Architecture



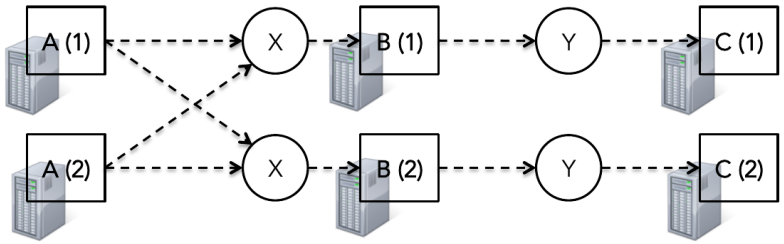
Map-Reduce: Parallel Processing Paradigm

Data abstractions: Data Set, Data Stream

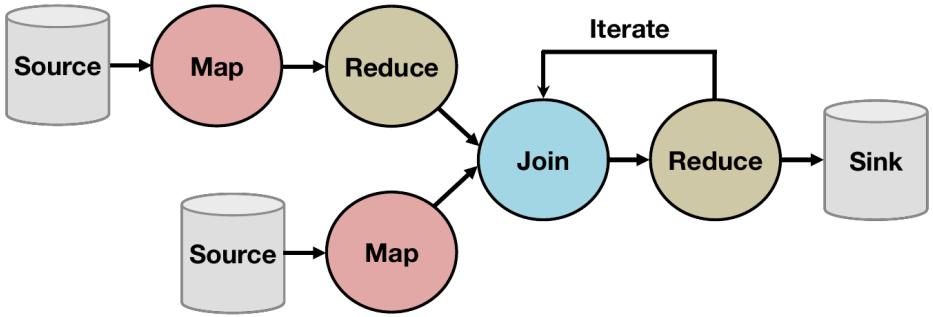
Program



Parallel Execution

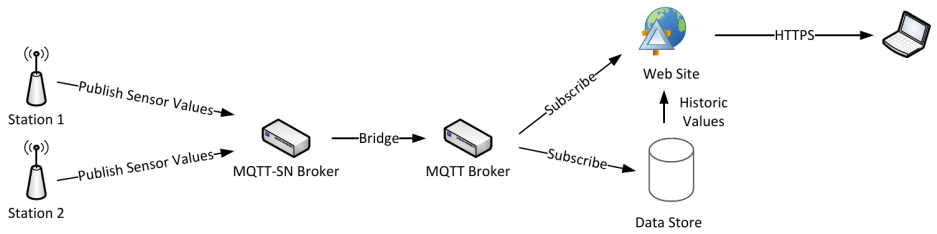


Map-Reduce Pipelines

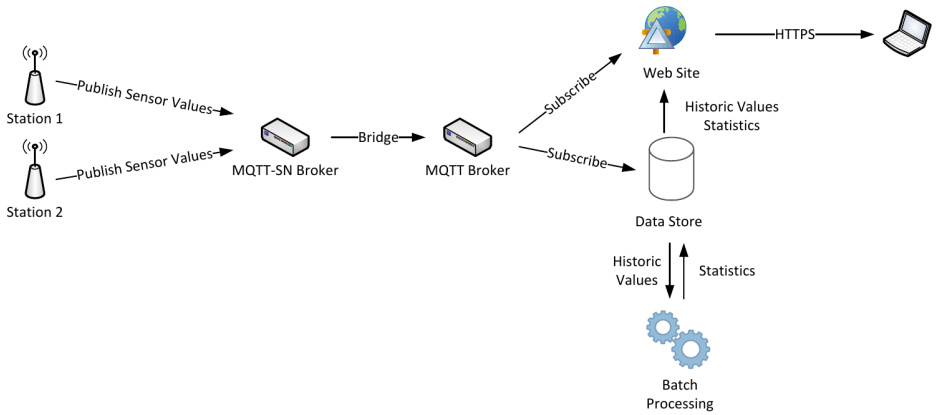


Map, FlatMap, MapPartition, Filter, Project, Reduce, ReduceGroup, Aggregate, Distinct, Join, CoGoup, Cross, Iterate, Iterate Delta, Iterate-Vertex-Centric, Windowing

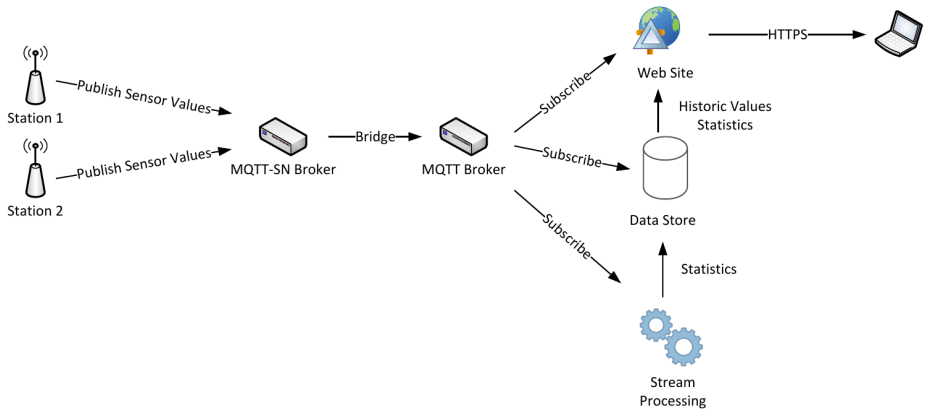
Cloud-based Architecture of 2nd Assignment



Cloud-based Architecture of 2nd Assignment



Cloud-based Architecture of 2nd Assignment



Data arriving to the Cloud

What Makes a Smart City?

Multiple Applications Create Big Data

Connected Plane
40 TB per day (0.1% transmitted)

Connected Factory
1 PB per day (0.2% transmitted)

Public Safety
50 PB per day (<0.1% transmitted)

Weather Sensors
10 MB per day (5% transmitted)

Intelligent Building
275 GB per day (1% transmitted)

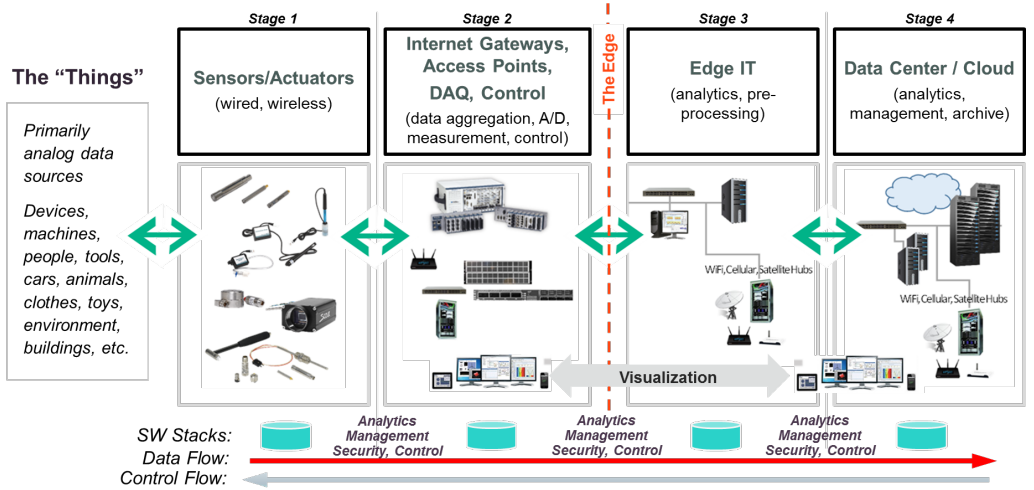
Smart Hospital
5 TB per day (0.1% transmitted)

Smart Car
70 GB per day (0.1% transmitted)

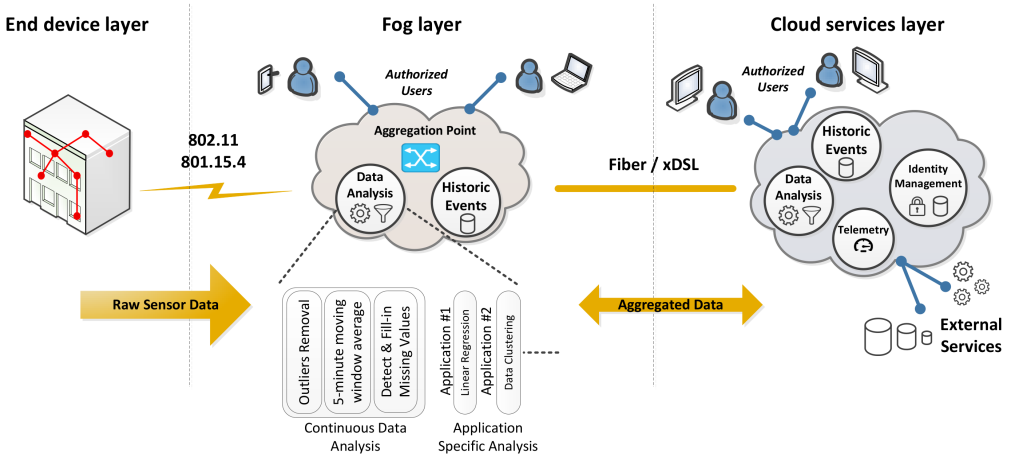
Smart Grid
5 GB per day (1% transmitted)

Source: Cisco Global Cloud Index, 2015–2020

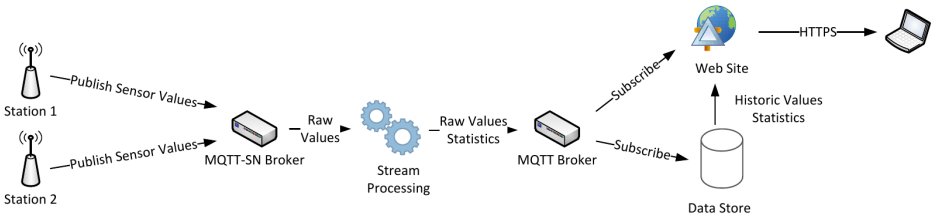
Edge-based Processing Stages



GAIA: Edge-based Smart School Architecture



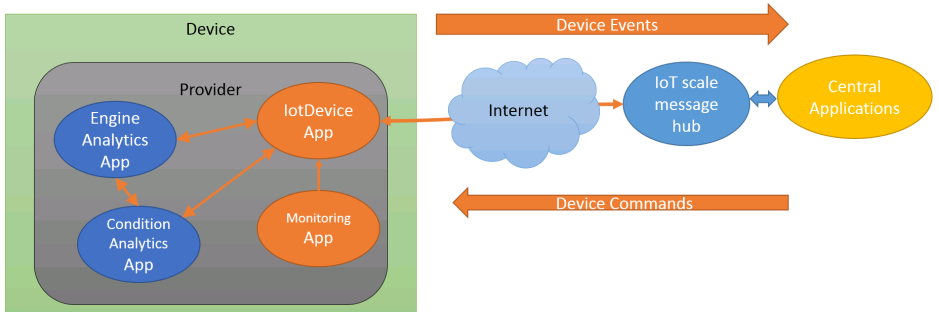
Edge-based Data Processing



Apache Edgent

- Light-weight stream processing.
- Provides a micro-kernel runtime to execute Edgent applications.
- Executes a data flow graph consisting of oplets connected by streams.
- A stream is an endless sequence of tuples or data items.

Apache Edgent Scenario



- Two analytic Edgent applications communicating with each other and the system IoTDevice application.
- IoTDevice application responsible for communicating with the message hub.