Data Stream Processing Tools

Dr. Sonali Agarwal
Associate Professor, Department of IT
IIIT Allahabad

Adaptive Machine Learning

From Batch Processing to Real-Time Intelligence

Exploring the evolution from ***traditional batch learning*** to ***sophisticated adaptive systems that learn continuously from streaming data*** and changing environments.

Batch Learning: The Foundation

Complete Dataset Training

Model trains on entire dataset at once, requiring all data before learning begins

Static After Training

Updates require complete retraining from scratch with high computational overhead

Fixed Environment

Best suited for stable datasets like MNIST, CIFAR-10, or annual credit scoring snapshots

Online Learning Revolution

Incremental Instance Updates

Models update incrementally as each data instance arrives, eliminating need for data storage

Lightweight Architecture

Minimal memory footprint enables real-time processing for time-sensitive applications

Key Challenges

- Sensitivity to noisy updates
- Risk of **catastrophic forgetting**
- Requires careful algorithm design

Stream Learning: Handling Infinite Data

01

Continuous Processing

Handles infinite, fast-arriving data streams with real-time constraints

02

Memory Management

Operates under strict memory limitations without storing complete history

03

Concept Drift Adaptation

Adapts to evolving statistical properties and changing data distributions

Concept Drift: The Moving Target Problem

1

Abrupt Drift

Sudden environmental changes like system failures or policy updates

2

Gradual Drift

Slow transitions such as climate trends or evolving user preferences

3

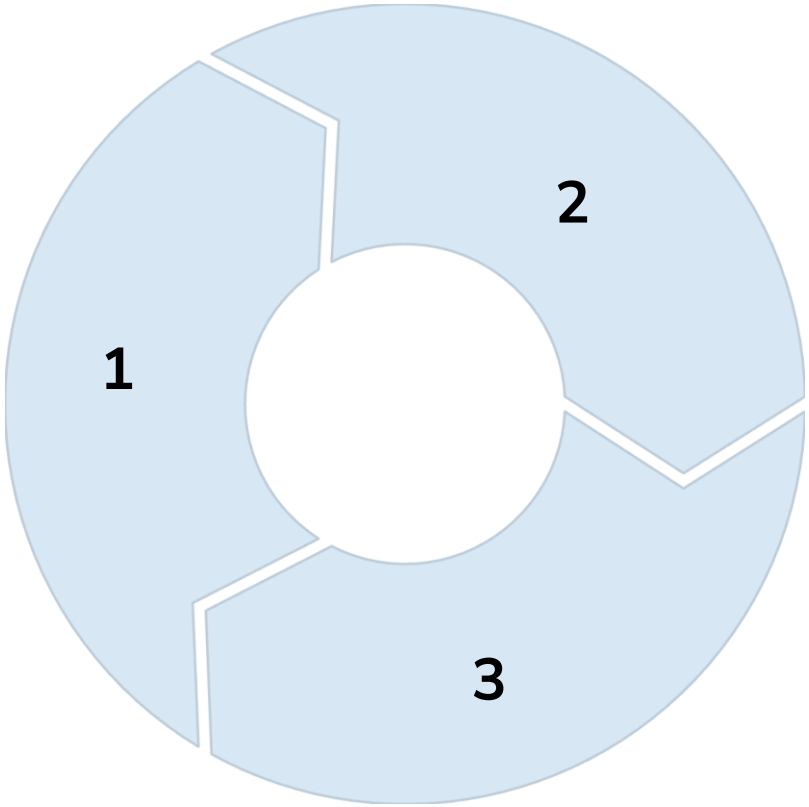
Recurring Drift

Cyclical patterns like seasonal sales or daily traffic variations

Continual Learning: Beyond Single Tasks

Sequential Tasks

Learn multiple environments over time



Plasticity vs Stability

Balance new learning with memory retention

Anti-Forgetting

Prevent catastrophic loss of previous knowledge

Critical for personal assistants and robotic systems that must accumulate skills without losing previous capabilities.

Advanced Techniques

1

Replay Methods

Strategic rehearsal using subsets of historical data to maintain performance

2

Regularization Approaches

Learning without Forgetting (LwF) preserve important parameters

3

Dynamic Architectures

Progressive networks that grow and adapt structure for new tasks

Specialized Learning Paradigms

Semi-Supervised Streams

Leverage limited labeled data with abundant unlabeled streams for cost-effective learning

Applications

- Medical monitoring systems
- Social media content analysis
- Cybersecurity threat detection

Deep Stream Learning Integration

1

Hybrid Pipelines

Combine classical efficiency with deep learning representation power

2

Online Fine-Tuning

Adapt pre-trained models incrementally for streaming applications

3

Real-Time Intelligence

Enable sophisticated video analytics and adaptive speech recognition

The Future is Adaptive

From IoT sensor networks to fraud detection systems, adaptive learning transforms how machines understand our evolving world.

24/7

Continuous Learning

Always-on intelligence

∞

Infinite Scalability

Unbounded data streams

1ms

Real-Time Response

Ultra-low latency decisions

Introduction to Pyspark

Basics of Pyspark and how it differs from other frameworks.

What is PySpark?

- PySpark is an interface for Apache Spark in Python.
- PySpark provides **Py4j library,** with the help of this library, Python can be easily integrated with Apache Spark.
- PySpark plays an essential role when it needs to work with a vast dataset or analyze them.
- With PySpark, you can write Python and SQL-like commands to manipulate and analyze data in a distributed processing environment.

Difference between Spark and Pyspark

Spark	Pyspark
1. It is a processing framework.	1. It is a collaboration of spark framework to work with python language.
2. Spark is a general-purpose and open-source engine for processing huge amounts of data.	2. Pyspark is an API gateway to apache spark using python language.
3. Originally written in scala.	3. Originally written in python.
4. Uses RPC server to accept API from other programming languages.	4. Pyspark API is converted internally to scala to trigger spark session.
5. Spark uses various library imports to carry out specific functionality.	5. Py4j library is used to facilitate working with spark.
6. Spark is developed to tame the need for huge processing of data in distributed systems.	6. PySpark is developed to make it easy for python developers to communicate with the Spark framework

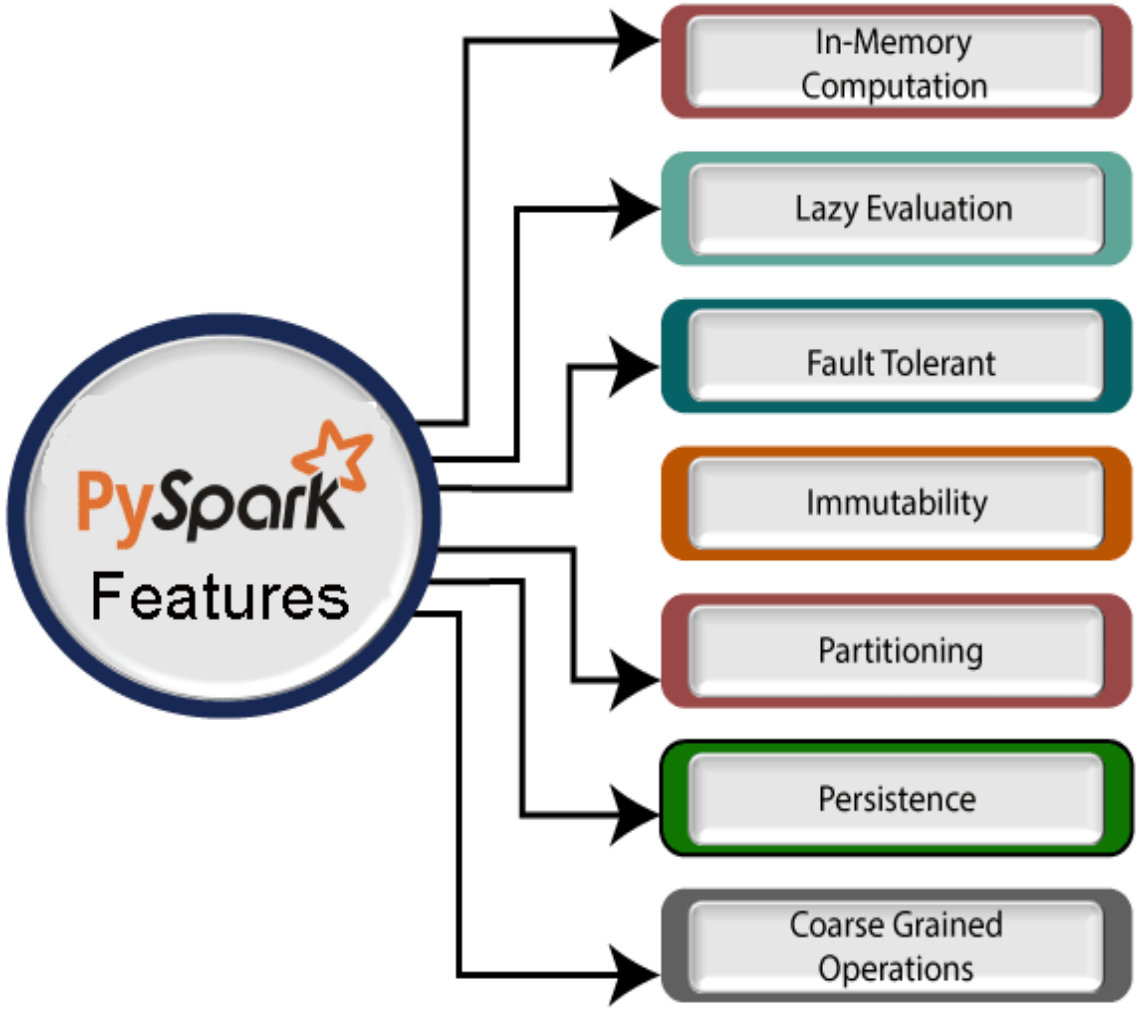
Key features of PySpark

Real-time Computation

- PySpark provides real-time computation on a large amount of data because it focuses on in-memory processing. It shows the low latency.

Support Multiple Language

- PySpark framework is suited with various programming languages like Scala, Java, Python, and R. Its compatibility makes it the preferable frameworks for processing huge datasets.



Key features of PySpark

Caching and disk consistency

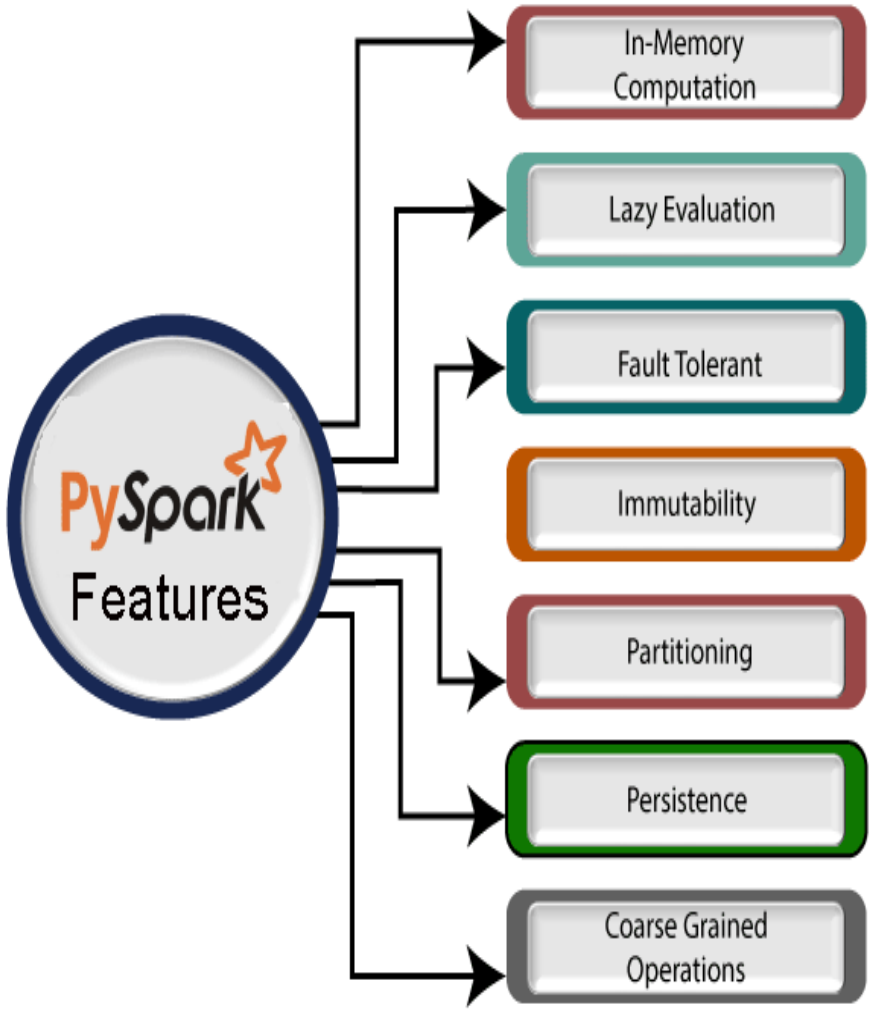
- PySpark framework provides powerful caching and good disk consistency.

Swift Processing

- PySpark allows us to achieve a high data processing speed, which is about 100 times faster in memory and 10 times faster on the disk.

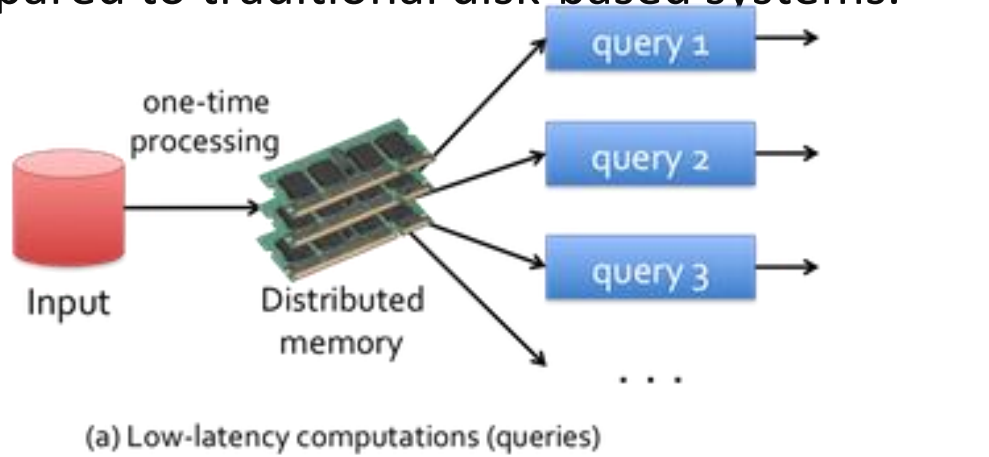
Works well with RDD

- Python programming language is dynamically typed, which helps when working with RDD.

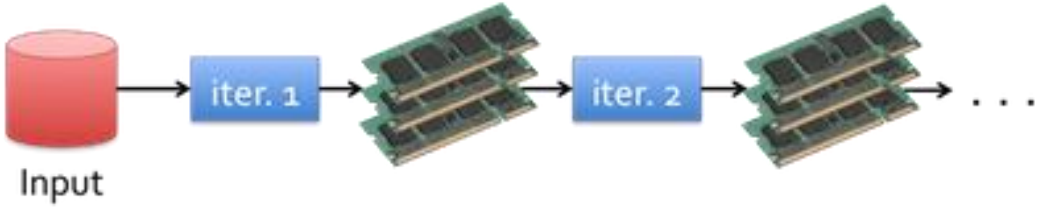


In memory computation

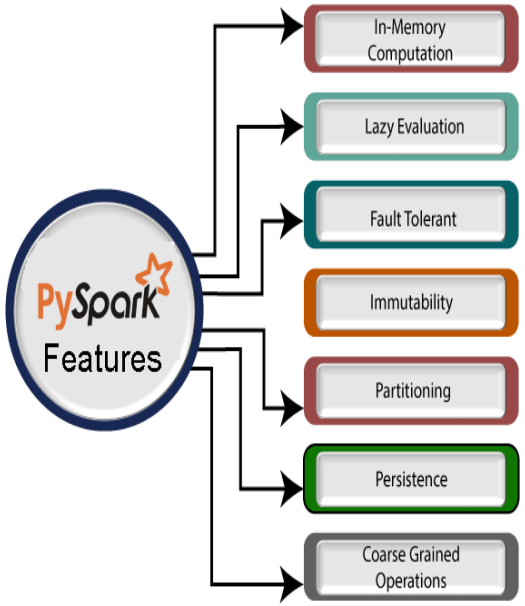
- Instead of relying heavily on disk-based I/O operations, Spark keeps intermediate data in memory as much as possible, which significantly speeds up processing compared to traditional disk-based systems.



(a) Low-latency computations (queries)



(b) Iterative computations



When we use cache() method, all the RDD stores in-memory.

Lazy Evaluation

- In Spark, transformations (like map, filter, flatMap) are not executed immediately.
- Instead, Spark builds a logical execution plan (DAG: Directed Acyclic Graph).
- Actual computation only happens when an action (like collect, count, saveAsTextFile) is called.

Why Lazy Evaluation?

- **Optimization**: Spark can optimize the whole chain of transformations before execution (e.g., pipeline transformations, reduce shuffles).
- **Efficiency**: Avoids unnecessary computations by only materializing data when needed.
- **Fault Tolerance**: DAG allows recomputation of lost partitions without re-running the entire pipeline.

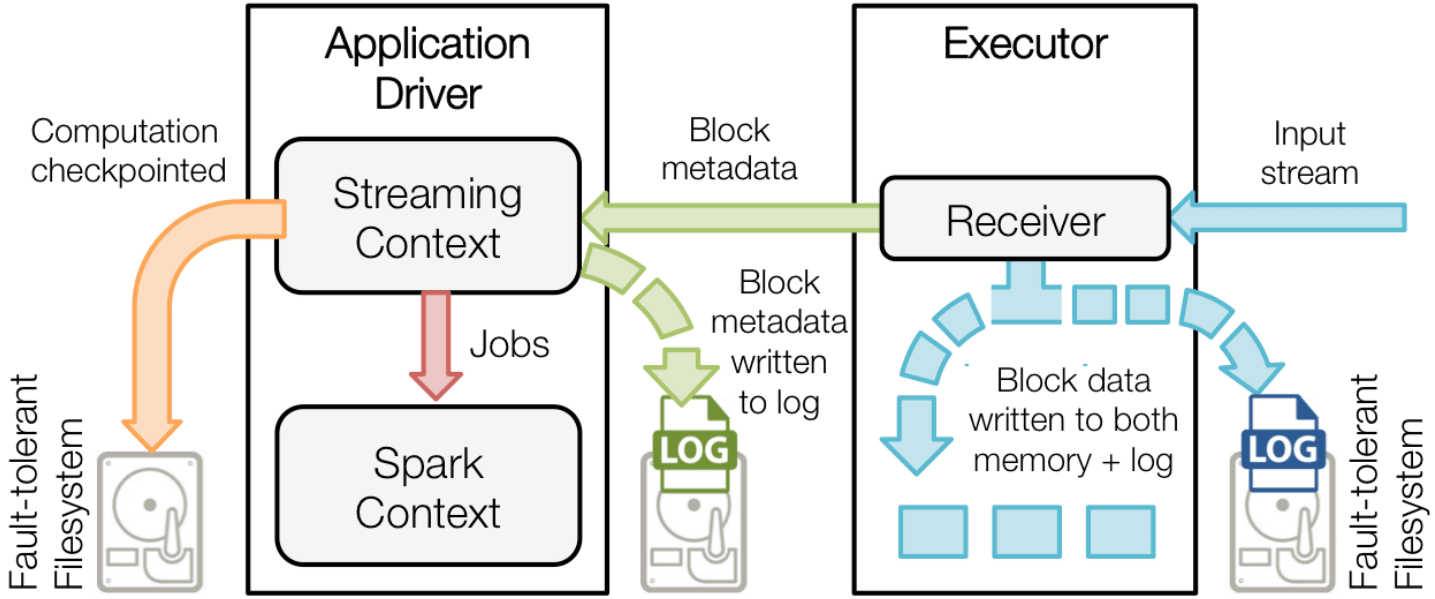
Fault tolerant

Spark Fault Tolerance is the ability of a system to continue to function properly even if some of its components fail (or have one or more faults within them).

- **Worker Node Failure**
  - The worker node (or slave node) executes the application code on the Spark cluster. The application will bear the loss of in-memory in case of any worker nodes operating stop functioning. The loss of buffer data will occur if a receiver is used over crashed nodes.
- **Driver Node Failure**
  - The driver node’s purpose is to run the Spark Streaming application; however, if it fails, the SparkContent will be lost, and the executors will be unable to access any in-memory data.

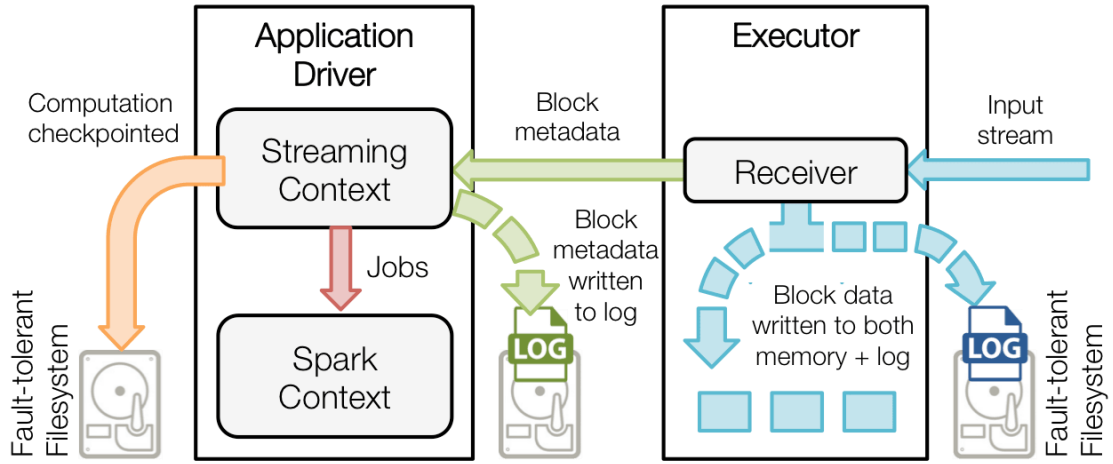
Fault tolerant (In normal case)

- Receiving data (blue arrows) -
- Notifying driver (green arrows) –
- Processing the data (red arrow) -.
- Checkpointing the computation (orange arrow)



Fault tolerant (in abnormal case)

- Recover computation (orange arrow) –
- Recover block metadata (green arrow) –
- Re-generate incomplete jobs (red arrow) –
- Read the block saved in the logs (blue arrow) –

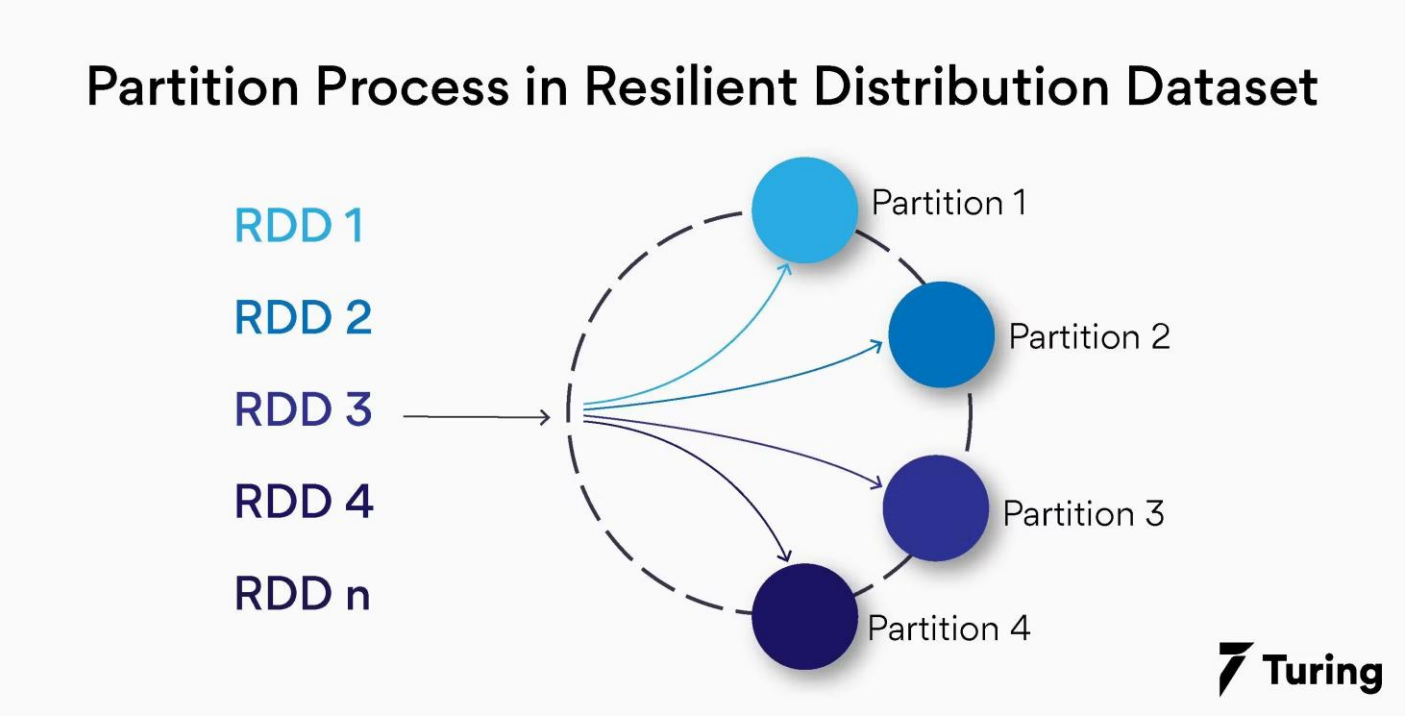


Immutability

- Immutability means that once an RDD or **DataFrame is created, its contents cannot be changed.**
- This is because RDDs are immutable in Spark, which means that any operation that modifies an RDD actually creates a new RDD with the modified data.
- This is a fundamental characteristic of RDDs in Spark that enables their scalability, parallelism, and fault tolerance.

Partition

- The resilient distributed datasets are a collection of massive volumes of data items that cannot fit into a single node. It is due to this reason that they need to be partitioned across multiple nodes.



Course Gained Operation

- Spark RDD offers two types of grained operations namely coarse-grained and fine-grained.
- The **coarse-grained operation** allows us to transform the whole dataset while the fine-grained operation allows us to transform individual elements in the dataset.
- Coarse-grained transformations work on entire RDD partitions at once, processing data in bulk. They are powerful and efficient for parallel processing. Examples include map, filter, and more.
- Fine-grained transformations operate on individual elements within RDD partitions. While precise, they can be less efficient for parallel processing. Examples include flatMap, mapPartitions, and others.

Coarse-grained transformations

```
# Coarse-grained transformation: Using 'map' to multiply each element by 2
result = rdd.map(lambda x: x * 2)

# Read data from a directory of text files
rdd = sparkContext.textFile("path/to/text/files")

 # Read data from a SequenceFile
 rdd = sparkContext.sequenceFile("path/to/sequence/file")

 # Read binary files and get (filename, content) pairs
 rdd = sparkContext.binaryFiles("path/to/binary/files")

 # Save an RDD as text files (coarse-grained)
 rdd.saveAsTextFile("path/to/output/directory")

 # Save an RDD as a SequenceFile (coarse-grained)
 rdd.saveAsSequenceFile("path/to/output/sequencefile")

 # Save an RDD as serialized Java objects (coarse-grained)
 rdd.saveAsObjectFile("path/to/output/objectfile")

# Save an RDD as a Parquet file (coarse-grained)
rdd.saveAsParquetFile("path/to/output/parquetfile")

 # Save an RDD as a Hadoop dataset (coarse-grained)
 rdd.saveAsHadoopDataset(conf)
```

Fine-Grained Transformations:

```
# Fine-grained transformation: Using 'flatMap' to create pairs (element, ele
result = rdd.flatMap(lambda x: [(x, x*x)])

# Fine-grained read using filter
filtered_rdd = original_rdd.filter(lambda x: x % 2 == 0)

# Fine-grained read using map
mapped_rdd = original_rdd.map(lambda x: x * 2)

# Fine-grained read using sample
sampled_rdd = original_rdd.sample(False, 0.2)

# Fine-grained read using take
first_elements = original_rdd.take(5)

# Fine-grained read using takeSample
sampled_elements = original_rdd.takeSample(False, 5)

# Fine-grained read using collect (caution with large datasets)
all_elements = original_rdd.collect()
```

Reasons to choose PySpark over pandas

- PySpark is a general-purpose, in-memory, distributed processing engine that allows you to process data efficiently in a distributed fashion.
- Applications running on PySpark are 100x faster than traditional systems.
- Using PySpark we can process data from Hadoop HDFS, AWS S3, and many file systems.
- PySpark is also used to process real-time data using Streaming and Kafka.
- PySpark natively has machine learning and graph libraries.
- PySpark provides a variety of methods for data processing along with methods to convert PySpark DataFrame to Pandas Dataframe and vice-versa.

River: Online Machine Learning in Python

A unified, open-source library for streaming machine learning that handles continuous data streams with incremental learning capabilities.

The Challenge of Real-Time Learning

Batch Learning Limits

Traditional ML requires complete datasets and periodic retraining, creating delays and resource constraints.

Continuous Data Streams

Real-world applications generate endless streams of data that need immediate processing and adaptation.

Online Learning Need

Systems must learn incrementally from each new sample without storing entire datasets in memory.

Introducing River

What is River?

Open-source Python library for online machine learning, born from the merger of Creme and scikit-multiflow.

- Classification and regression
- Clustering and forecasting
- Anomaly detection
- Community-driven development

Mission: Become the go-to library for streaming machine learning across academia and industry.

Core Features

Incremental Learning

Process data instance by instance or in mini-batches, adapting models continuously without full retraining.

Dictionary Structures

Efficient data handling with O(1) operations for heterogeneous and evolving feature spaces.

Transformers & Pipelines

Chain operations seamlessly with modular components for reproducible ML workflows.

Cython Performance

Optimized core implementation delivers high-speed processing for real-time applications.

Cython = A superset of Python that allows you to add static type declarations and compile Python code into C extensions.It bridges the ease of Python with the speed of C.

Architecture Design

01

Specialized Mixins

Models extend base classes with core methods: learn_one, predict_one, predict_proba_one, score_one

02

Transformer Interface

Consistent transform_one method for data preprocessing and feature engineering

03

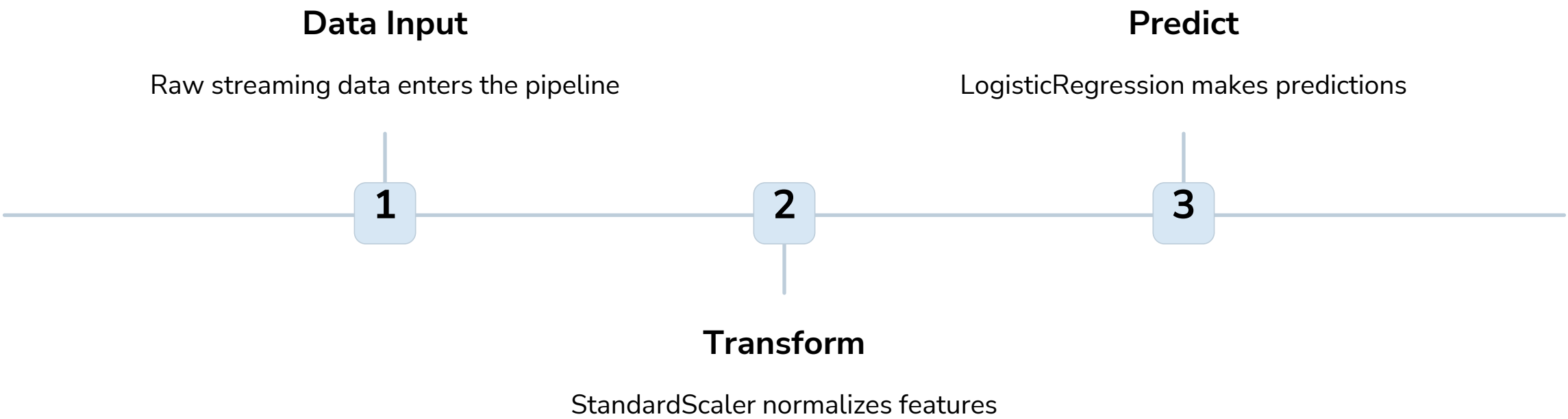
Pipeline Chaining

Use | operator to chain operations: StandardScaler | LogisticRegression

Pipeline Architecture

> `StandardScaler | LogisticRegression`

Pipelines create sequences of transformers and estimators, ensuring reproducibility and modularity in machine learning workflows.



Learning Modes

Instance Incremental

Process one sample at a time with learn_one method for true online learning.

Batch Incremental

Handle mini-batches using learn_many with pandas DataFrame support.

Comprehensive Algorithm Suite

Linear Models

Optimized linear algorithms for classification and regression

Clustering

Unsupervised learning with online clustering algorithms

Tree Methods

Decision trees, random forests, and ensemble methods

Anomaly Detection

Identify outliers and concept drift in streaming data

Time Series

Forecasting, bandits, and recommender systems

Key Contributions

1

Unified Library

Single solution for all online ML needs

2

Flexible Architecture

Easy-to-use, efficient design patterns

3

Superior Performance

Comparable or better than alternatives

4

Open Source

Community-driven development model

River bridges the gap between academic research and industry applications, providing a robust foundation for streaming machine learning.

Get Started with River

Resources & Links

- GitHub Repository
- Documentation and tutorials
- Community support and contributions
- Citation: arXiv:2012.04740

Related Projects

Built upon the foundations of Creme, scikit-multiflow, and MOA frameworks.

Ready to stream? Join the River community and start building real-time ML applications today.

CapyMOA: Efficient Machine Learning for Data Streams in Python

A high-performance stream learning library that bridges the gap between efficiency and accessibility

The Stream Learning Challenge

Continuous Data

Real-world data arrives as endless streams, requiring algorithms that can adapt and learn incrementally without storing historical data.

Concept Drift

Data distributions evolve over time, making previously learned patterns obsolete and demanding continuous model adaptation.

Resource Constraints

Limited memory and real-time processing requirements force efficient algorithms that balance accuracy with computational speed.

Traditional batch learning approaches fail when faced with these dynamic, resource-constrained environments.

Introducing CapyMOA

Open-Source Foundation

Built on BSD 3-Clause License, ensuring accessibility and community-driven development for researchers and practitioners alike.

Hybrid Architecture

Combines Java's computational efficiency with Python's ease of use, delivering the best of both worlds for stream learning.

Comprehensive Toolkit

- Classification and regression algorithms
- Anomaly detection and clustering
- Drift detection mechanisms
- Semi-supervised learning
- AutoML capabilities
- Online pipeline orchestration

Competitive Landscape Analysis

1

MOA

Strengths: High efficiency, mature algorithms

Limitations: Java-only, steep learning curve

2

River

Strengths: Python-native, user-friendly API

Limitations: Performance bottlenecks

3

Scikit-Multiflow

Strengths: MOA integration attempts

Limitations: Largely superseded by River

4

CapyMOA

Innovation: Optimal balance of efficiency and accessibility

Core Architecture Strengths

1

Efficiency First

Leverages optimized Java algorithms with minimal memory overhead, ensuring high-performance stream processing without computational waste.

2

Seamless Integration

Native interoperability with MOA's extensive algorithm library and PyTorch's deep learning capabilities for comprehensive ML workflows.

3

Python Accessibility

Intuitive, Pythonic API design that abstracts complexity while maintaining full control over advanced stream learning configurations.

Robust Data Architecture

Schema + Instance Model	Multi-Format Support	Abstraction Layer
Structured approach enforces feature consistency across streaming data, preventing common errors that plague dictionary-based systems.	Handles CSV, ARFF files, and synthetic data streams with automatic format detection and conversion for seamless data ingestion.	Hides Java implementation details behind clean Python interfaces, enabling focus on algorithm development rather than technical overhead.

Modular Pipeline Architecture

01

Transformers

Feature scaling, selection, and dimensionality reduction components that preprocess streaming data for optimal algorithm performance.

02

Drift Detectors

Integrated monitoring systems that identify concept drift in real-time, triggering adaptive responses to maintain model accuracy.

03

Learners

Classification and regression algorithms that continuously update their knowledge as new data arrives in the stream.

04

Custom Elements

Extensible PipelineElement class enables researchers to implement novel algorithms while maintaining framework compatibility.

Dynamic adaptation mechanisms automatically respond to drift signals, ensuring continuous learning effectiveness.

Advanced Concept Drift Management

DriftStream Simulation

Comprehensive module for generating both abrupt and gradual concept drifts, enabling thorough algorithm testing under realistic conditions.

Integrated Detection

Drift detectors seamlessly embedded within processing pipelines provide real-time alerts and trigger adaptive responses.

Continuous Evaluation

Prequential evaluation framework tracks performance metrics continuously, providing insights beyond simple accuracy measurements.

1 Detection

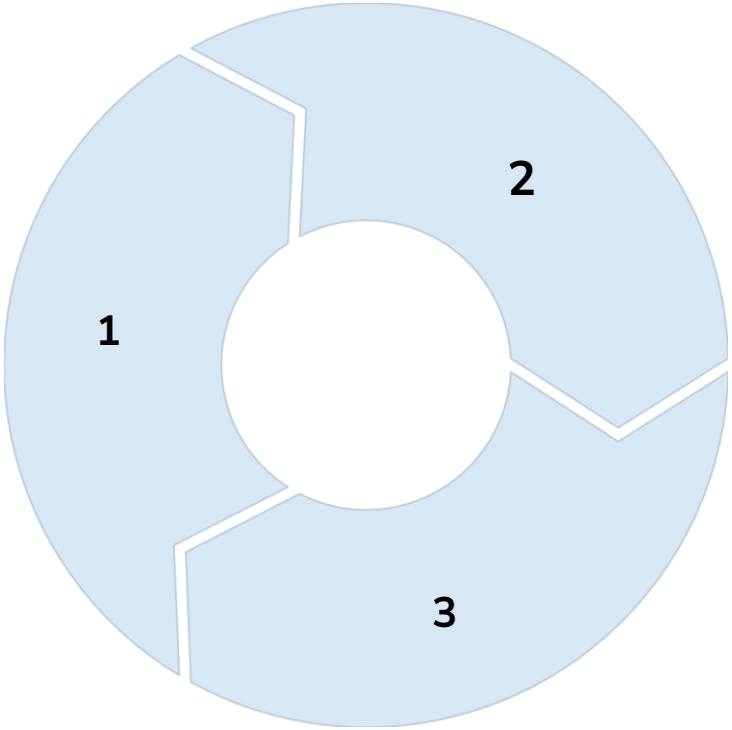
Monitor distribution changes

2 Analysis

Assess drift magnitude

3 Adaptation

Update model parameters



Comprehensive Evaluation Framework

1

Cumulative Metrics

Track performance across all instances processed since stream initialization, providing overall algorithm effectiveness insights.

2

Windowed Assessment

Focus on recent performance within sliding windows, capturing algorithm adaptability to evolving data patterns and concept drift.

3

Unified Interface

Consistent evaluation approach across classification, regression, and anomaly detection tasks with standardized metric reporting.

4

Clustering Specialization

Dedicated evaluation methods for unsupervised learning scenarios with specialized clustering quality assessment metrics.

Future Directions

CapyMOA Achievement

Successfully delivers an efficient, accessible, and extensible framework that bridges the performance gap between Java and Python stream learning libraries.

Outperforms existing Python frameworks while approaching MOA's computational efficiency.

Research Roadmap

- **Online Continual Learning (OCL)** integration for lifelong learning scenarios
- **Enhanced drift evaluation** with advanced statistical methods
- **Deep learning framework** integration beyond PyTorch
- **Advanced clustering evaluation** metrics and methodologies

> Empowering the next generation of stream learning research with performance that doesn't compromise on accessibility.

Complex Event Processing for Real Time Applications

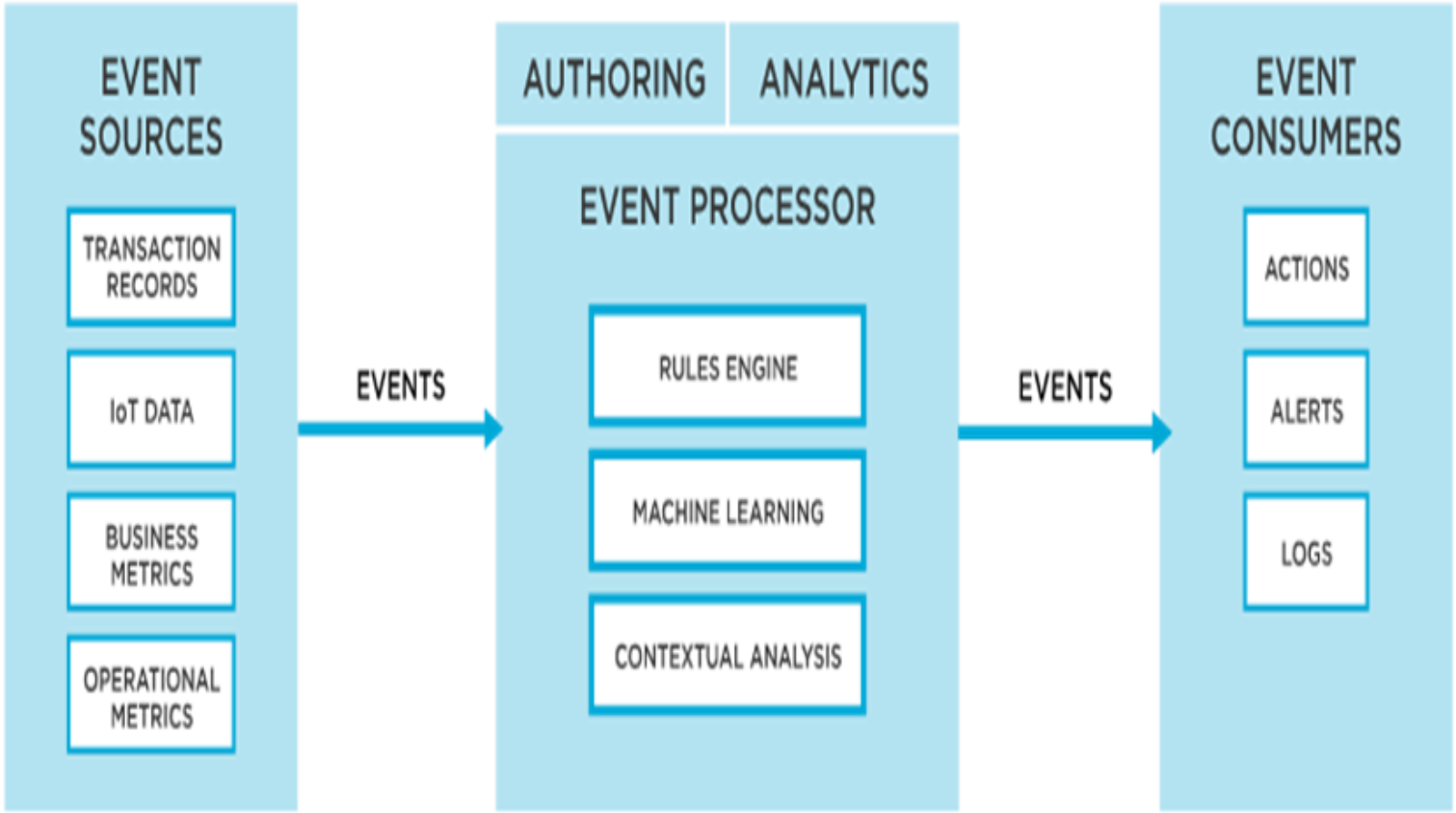
Dr. Sonali Agarwal
Associate Professor, Dept. Of IT
IIIT Allahabad

Outline

- Introduction to Event Processing
- Real Time Analytics
- Complex Event Processing
- Use cases of Complex Event Processing
- Tools Used
- Flink And Kafka
- Rules and Complex Events
- Use Cases
- Conclusion and Future work

What is Event Processing?

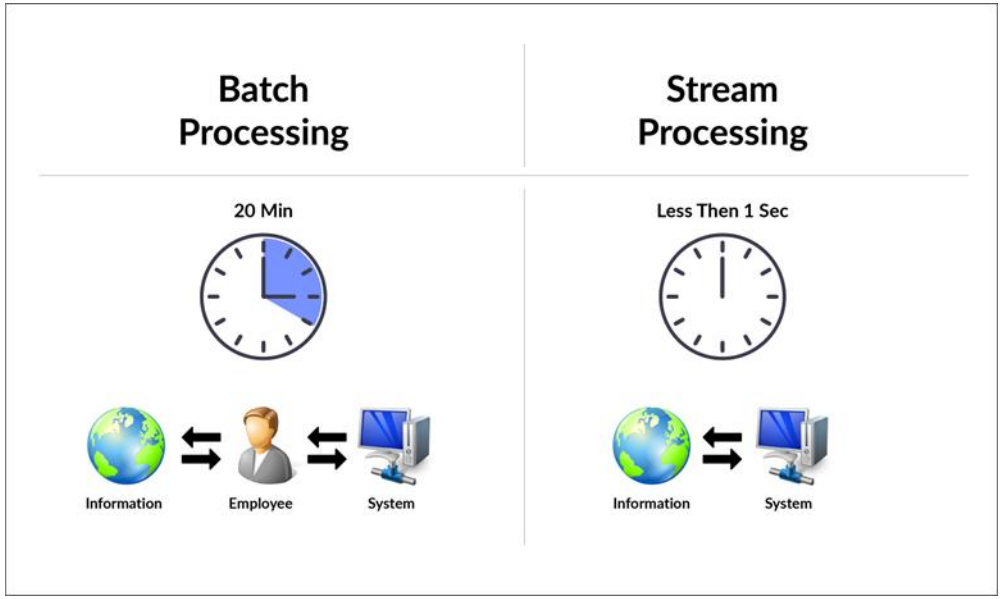
- Event processing is a process that takes events or streams of events, analyzes them and takes an appropriate action.
- Analysis can be based on predefined decision tables or more sophisticated machine learning algorithms.



Event Processing

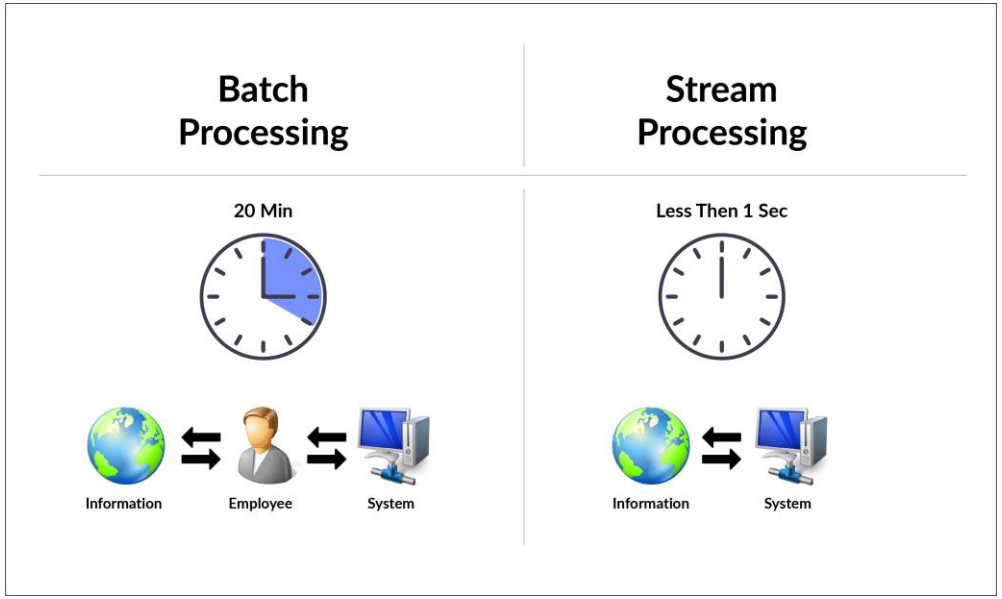
Real Time Analytics

- Real time analytics refers to the process of preparing and measuring data as soon as it enters the database. In other words, users get insights or can draw conclusions immediately (or very rapidly after) the data enters their system.
- Real-time analytics allows businesses to react without delay. They can seize opportunities or prevent problems before they happen.



Real Time Analytics

- By comparison, batch-style analytics may take hours or even days to yield results. Consequently, batch analytical applications often yield only “after the fact” insights (lagging indicators).



Who Uses real time analytics

Examples of real-time applications includes:

- Real time credit scoring, helping financial institutions to decide immediately whether to extend credit.
- Customer relationship management (CRM), maximizing satisfaction and business results during each interaction with the customer.
- Fraud detection at points of sale.
- Targeting individual customers in retail outlets with promotions and incentives, while the customers are in the store and next to the merchandise.

Real-Time Analytics Challenges

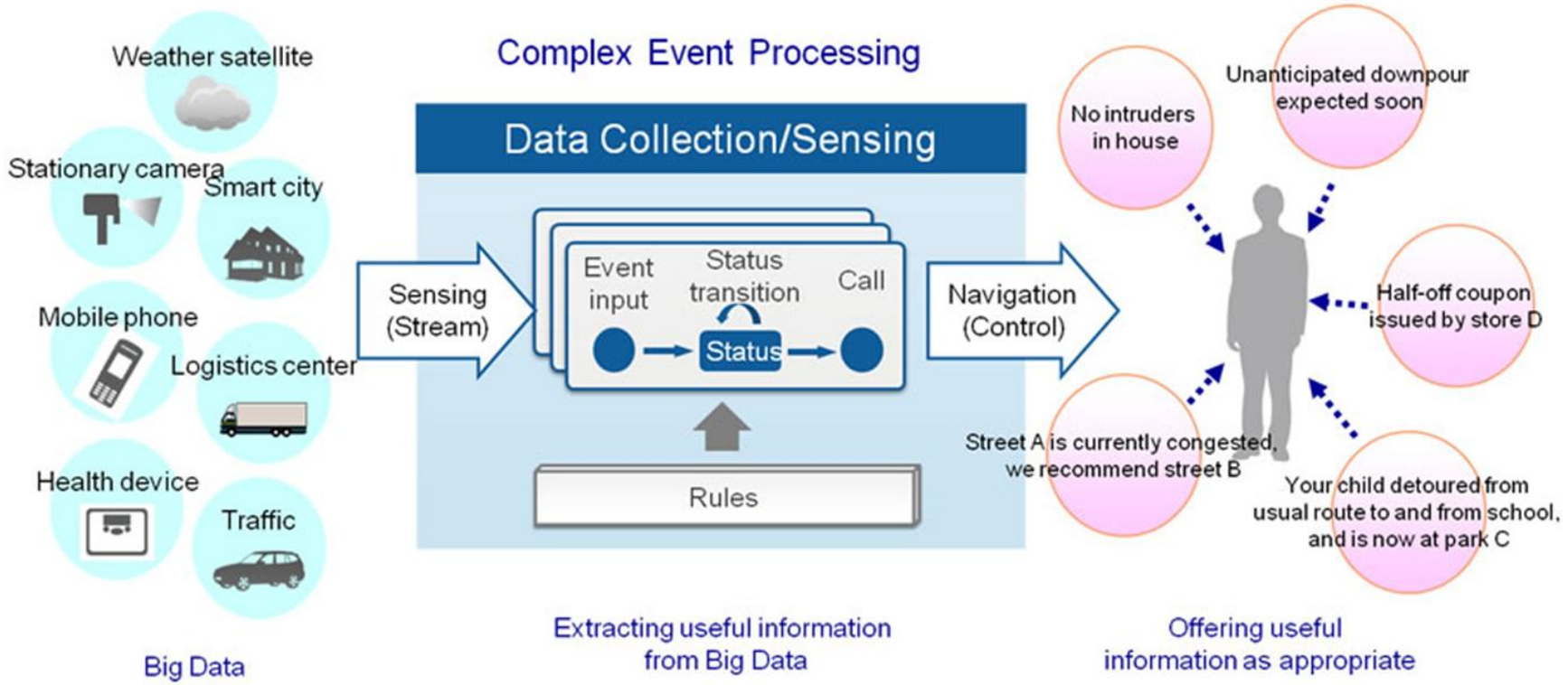
- To be immediately useful, real-time analytics applications should have high availability and low response times. They should also be able to handle large amounts of data, up to and including terabytes. Yet they should still return answers to queries within just seconds.
- Real-time also means handling changing data sources, which may spring up as market and business factors change. In short, they should also handle big data. Real-time big data analytics are already used in financial trading.
- They use data from financial databases, social media, and satellite weather stations to instantly inform buying and selling decisions.

Real Time Analytics

Real Time Analytics

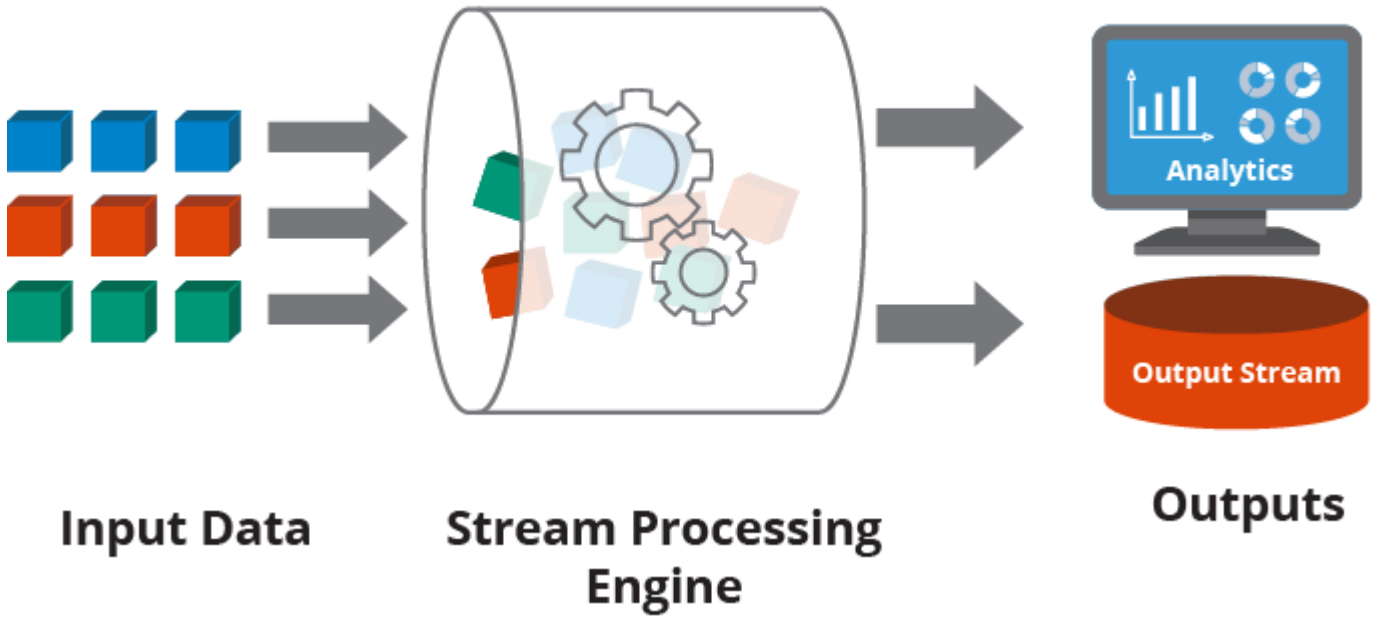
What is Complex Event Processing?

- Complex event processing (CEP) is the use of technology to predict high-level events likely to result from specific sets of low-level factors.
- By identifying and analyzing cause-and-effect relationships among events in real time, CEP allows personnel to take effective actions in response to specific opportunities or threats.



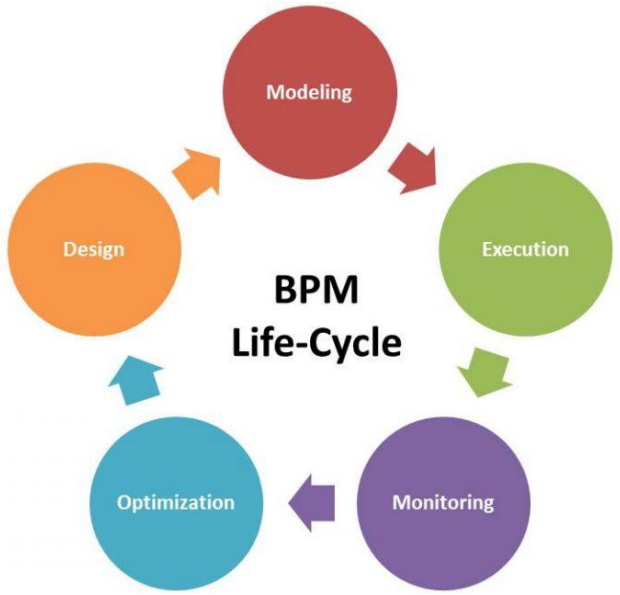
What is Complex Event Processing?

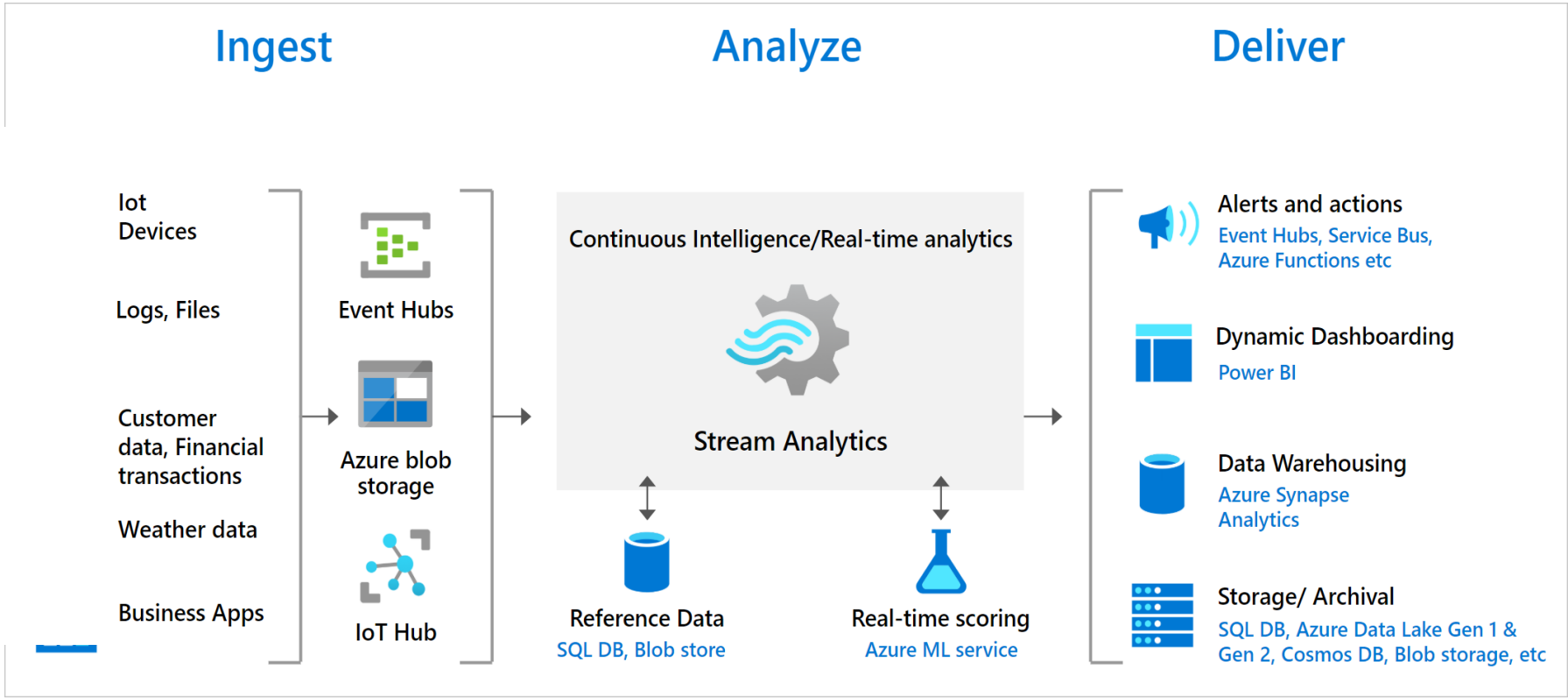
- Likewise, CEP has been somewhat overshadowed by event stream processing and streaming analytics, two approaches for processing and analyzing streams of data that implement some, but not all, of the core ideas of CEP.



What is Complex Event Processing?

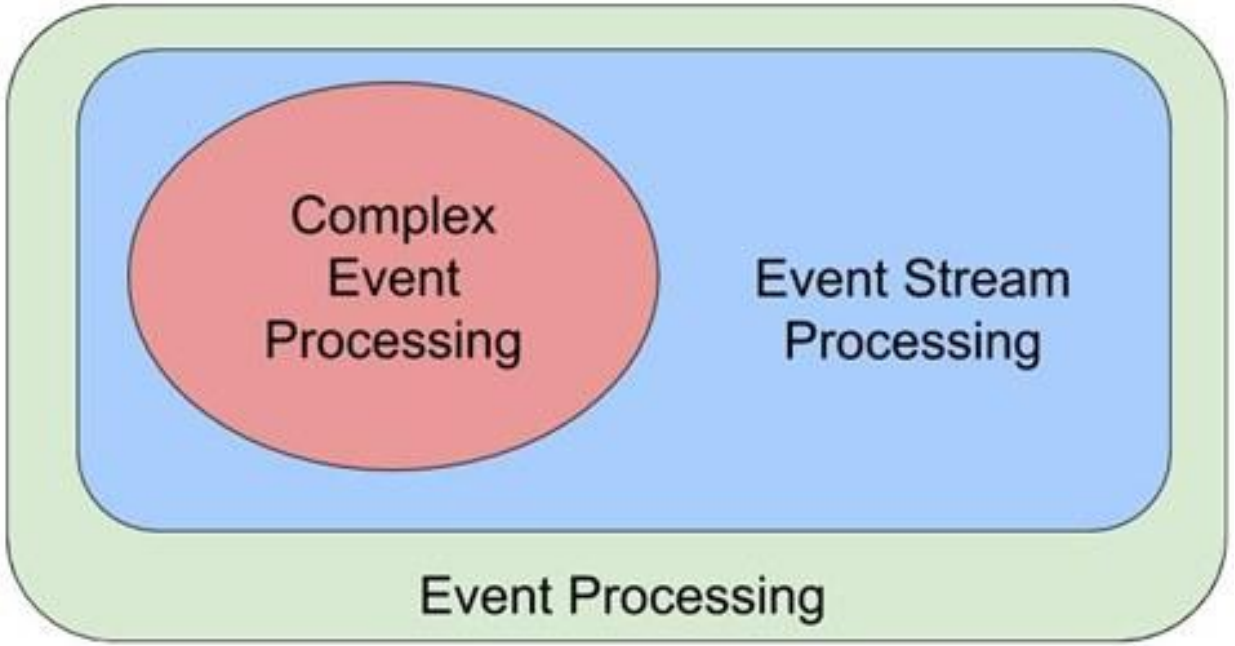
- Despite the fact that different terminology is often used today for processing and analyzing real-time events, experts believe that CEP remains relevant for improving enterprise architectures, and, in particular, for its application in the discipline of business process management, which aims to improve business processes end to end.





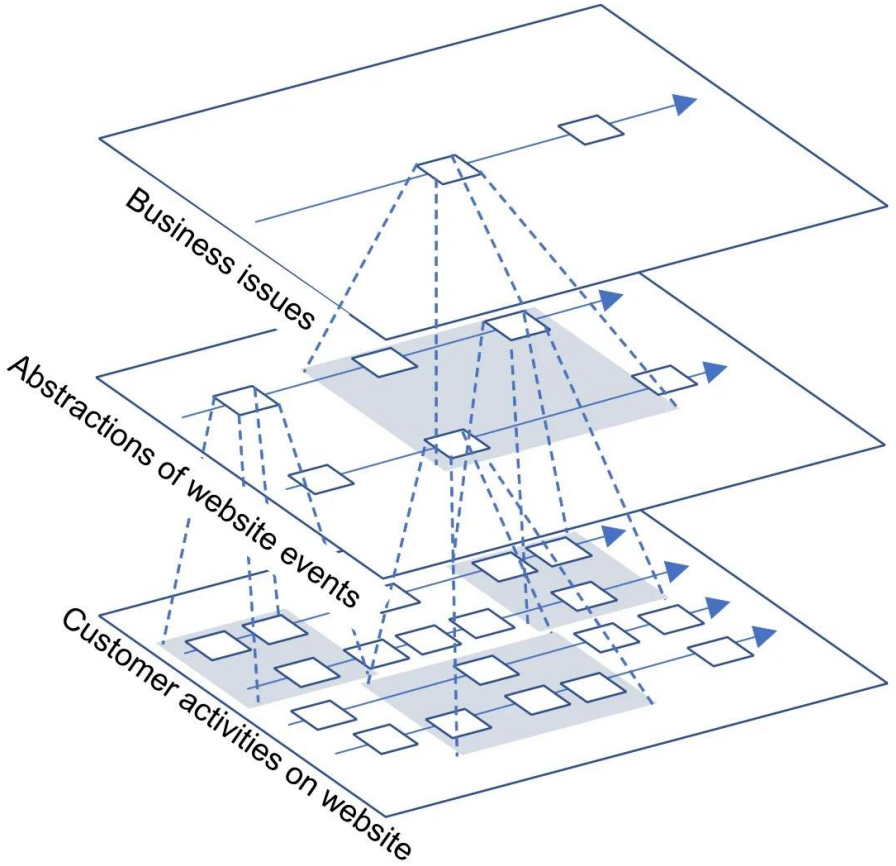
Components of Event Processing

- Stream (i.e., event) processing supports many kinds of continuous analytics such as filter, aggregation, enrichment, classification, joining, etc.
- The CEP uses patterns over sequences of simple events to detect and report composite events.



Conceptual Hierarchies in CEP

- One of the distinguishing features of CEP is the use of conceptual hierarchies.
- We all know that events in the real world come at different levels of abstraction.
- At one level, we can talk about a customer's intentions and feelings. At a lower level, we can talk about her GPS trail or the actions of her mouse.



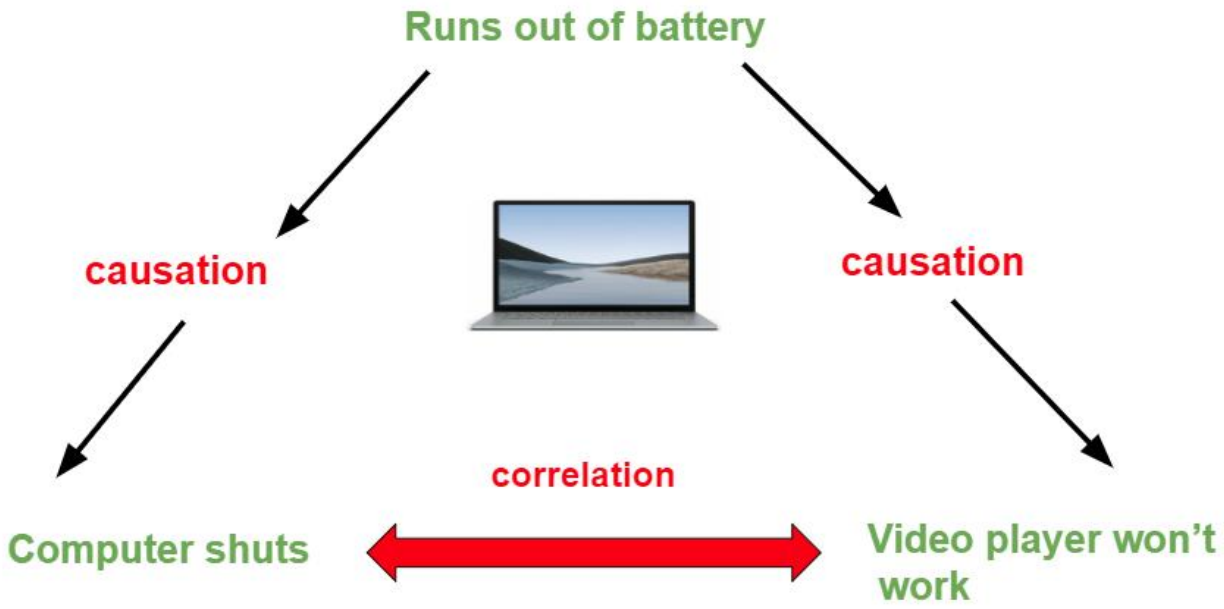
Abstractions computed from intermediate events: Unsuccessful search, product sales statistics

Abstractions computed from base events: Login, create customer profile, add credit card, product lookup...

Input (base) events received from website: search action, add item to cart, enter customer name, click on a link, resize image, enter credit card type, enter credit card number, enter credit card security code....

Causal Relationships in CEP

- Another distinguishing feature of CEP is the use of causal relationships.
- It is very common to talk about events causing other events.
- Causation means one thing will cause other thing



Event Pattern detection in CEP

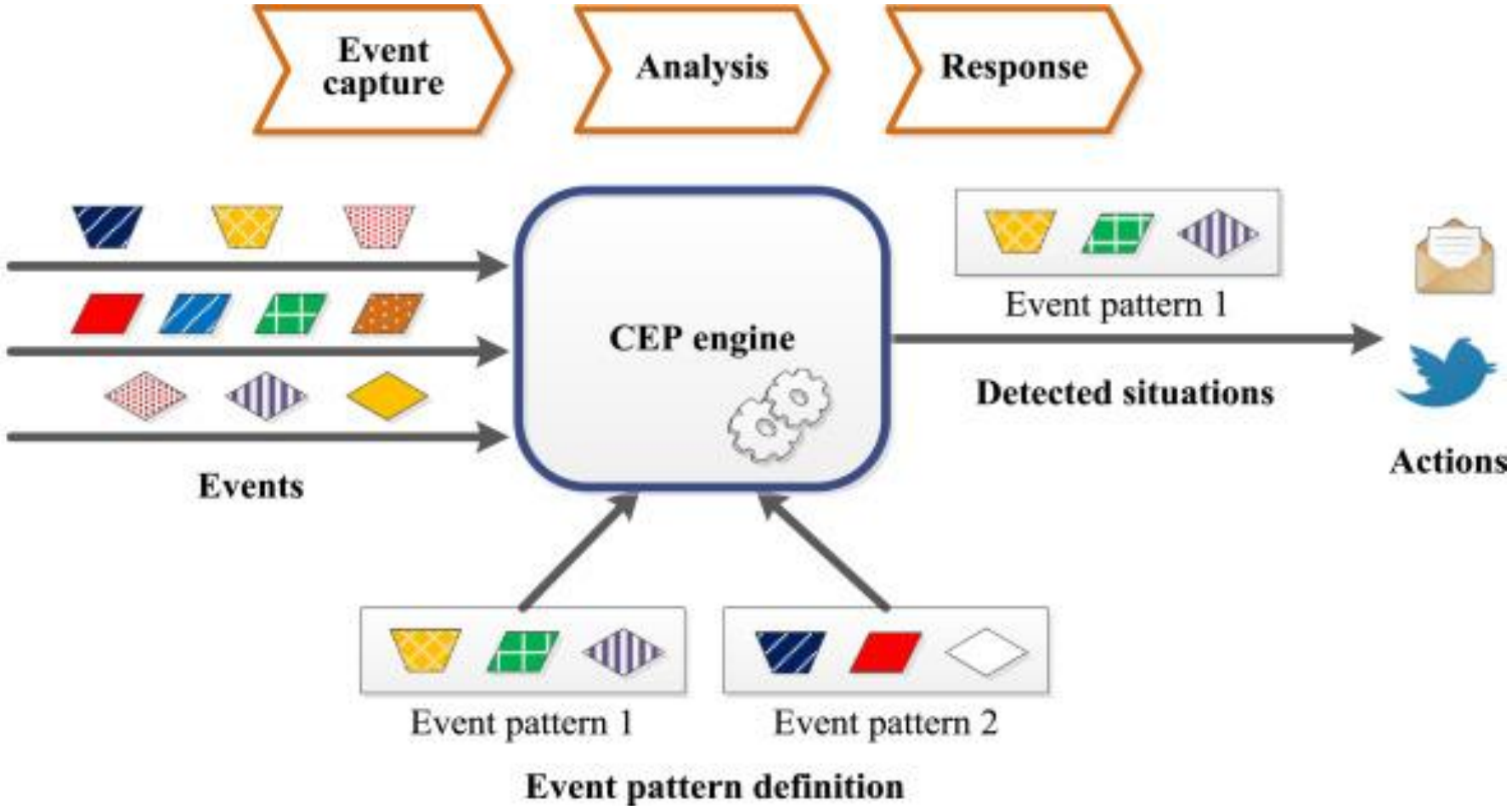
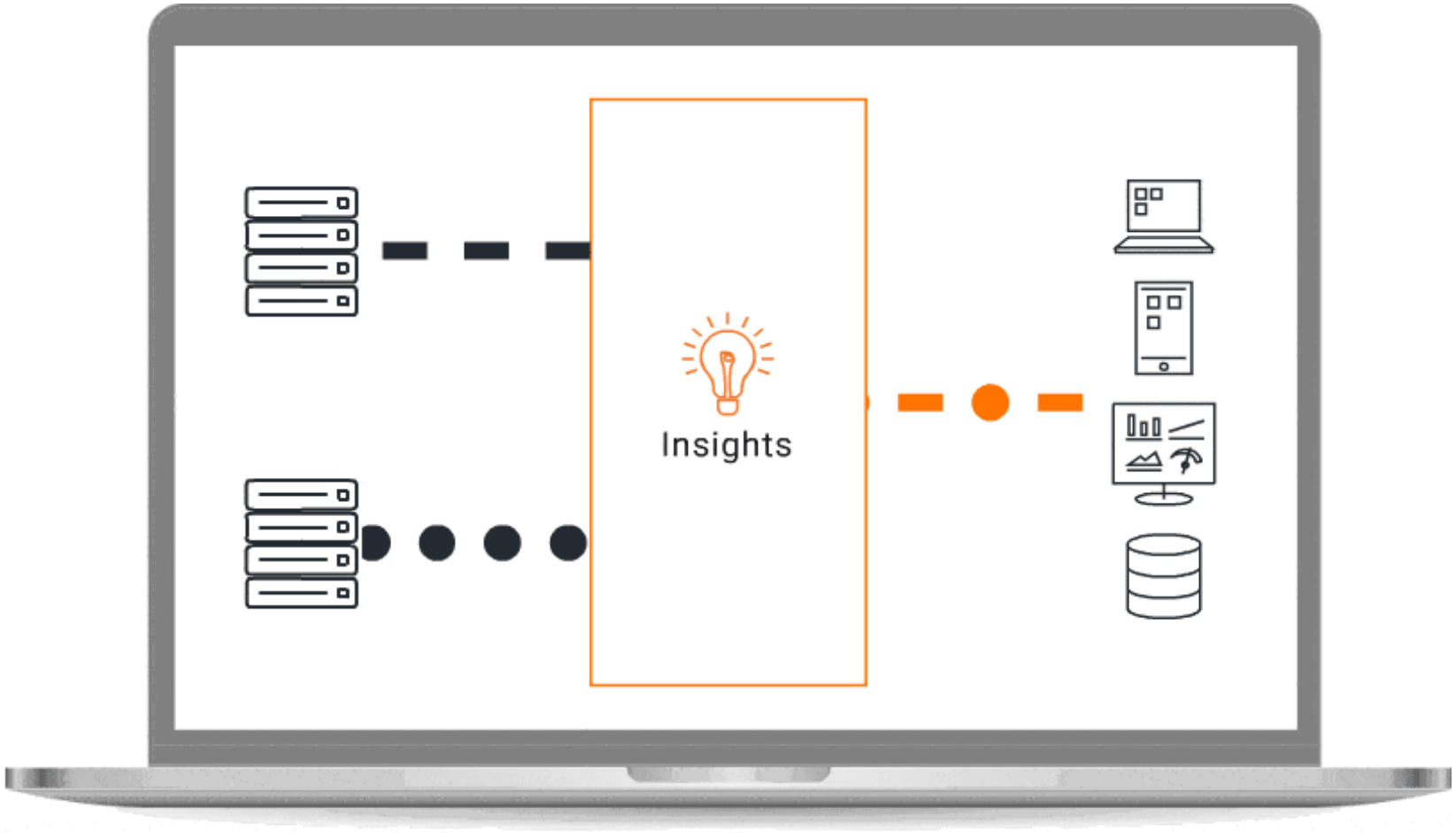


Fig.3- Pattern Detection

Processing of Event Streams



Processing of Event Streams

Benefits of complex event processing

CEP's ability to detect complex patterns in multiple sources of data provides many benefits, including the following:

- Makes it easier to understand the relationship between events.
- Helps to connect individual events into more complex chains.
- Simplifies the development and tuning of business logic.
- Can be embedded into fraud detections, logistics and IoT applications and helps to build more accurate simulations, models and predictive analytics.

CEP Use Cases

- **Fraud Prevention and Detection:** Banks can use CEP to inspect and identify fraudulent transactions by tracking real-time events against various patterns. A login from a new device can be combined with a password change and other account activity to create a complex event that flags the possibility of fraud.
- **Real-Time Marketing:** E-commerce retailers can use CEP to offer personalized recommendations based on a combination of GPS data, social network activity, holidays, and previous shopping habits. The ability to combine different data sources along with historical data is one of the key strengths of CEP.

CEP Use Cases

- **IoT:** By combining information across various sources, CEP has a transformative effect by collecting IoT sensor streams for real-time monitoring, analytics, and troubleshooting. For example, by combining distributed data from lighting, alarms, and other devices with real-time weather, date, and time, a smart building can predict the behavior of its occupants and optimize the use of lights and heating while providing automated services to occupants.

CEP Use Cases

- **Predictive Analytics:** By combining events generated by pharmacy sales, social networking sites, twitter, and GPS streams, we can predict the emergence of new coronavirus clusters. Almost all forms of predictions rely on finding complex patterns in massive amounts of data from numerous sources, so CEP is a natural part of the predictive analytics landscape.

Tools Used for Complex Event Processing

- Apache Spark Streaming used by Data bricks
- Apache Flink used by data Artisans
- Apache Samza used by LinkedIn
- Apache Storm used by Twitter.
- Hadoop/Map Reduce.
- Amazon Kinesis Analytics.
- Microsoft Azure Stream Analytics, Stream Insight.
- Fujitsu Software Interstage Big Data Complex Event Processing Server.

Why Apache Flink?

- Apache Flink reduces the complexity that has been faced by other distributed data-driven engines.
- It achieves this feature by integrating query optimization, concepts from database systems and efficient parallel in-memory and out-of-core algorithms, with the Map Reduce framework.

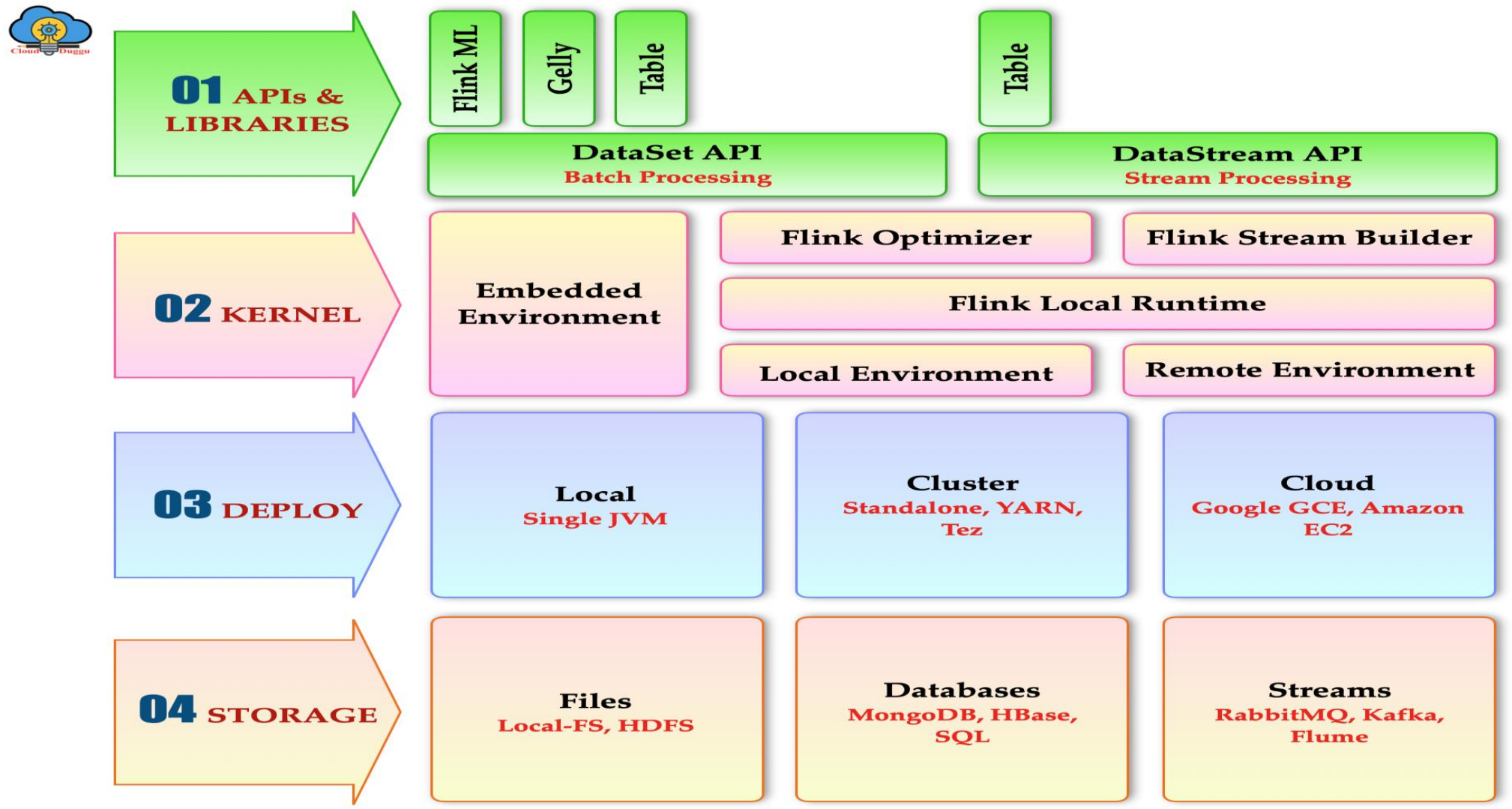
Apache Flink Features

- **True streaming engine:**
  - Infrastructure based on streaming data not only enables a new type of latency-critical applications and give more actual operational insights through more updated views of the processes.
  - They are more potential in making classical data warehousing setups radically more simple and flexible.
- **Custom Memory Manager:**
  - Flink implements its own memory management inside the JVM. Its features are C++ style memory management inside the JVM.
  - User data stored in serialized bytes array in JVM. Memory is allocated, de-allocated and used strictly using on internal buffer pool implementation.

Apache Flink Features

- **Unified Framework:**
  - Big data is getting matured with Flink and Flink is a unified framework which allows building a single data workflow that holds streaming, batch, SQL and Machine learning. Flink can process graph using its own Gelly library and use Machine learning algorithm from its own FlinkML library.
- **Automatic cost-based optimizer:**
  - In Flink, Batch programs are automatically optimized to exploit situations where expensive operations (like shuffles and sorts) can be avoided, and when intermediate data should be cached.
- **Easy to Use**
  - The Flink APIs make it easier to use than programming for Map Reduce and it is easier to test as compared to Hadoop.

Flink Ecosystem



Components of Flink System

Storage / Streaming: Flink is only a computation engine that does not have any storage system. It reads and writes data from different storage systems as well as can consume data from streaming systems.

- **HDFS –** Hadoop Distributed File System
- **Local-FS** – Local File System
- **HBase** – NoSQL Database in the Hadoop ecosystem
- **MongoDB** – NoSQL Database
- **RDBMS** – Any relational database
- **S3** – Simple Storage Service from Amazon
- **Kafka** – Distributed Messaging Queue
- **Flume** – Data Collection and Aggregation Tool
- **RabbitMQ** – Messaging Queue

Deployment of Flink System

Deployment :-Deployment of resource management is the second layer in Flink ecosystem. One can deploy Flink in the following modes:

- Local mode – On a single node, in single JVM
- Cluster – On a multi-node cluster, with following resource manager
- Standalone – This is the default resource manager of Flink.
- YARN – This is a very popular resource manager, it is part of Hadoop, introduced in Hadoop 2.x
- Mesos – This is a generalized resource manager.
- Cloud – on Amazon or Google cloud.

Components of Flink System

- **Flink Kernel:-**The third layer is Runtime the Distributed Streaming Dataflow, which is also called as the kernel of Apache Flink. This layer provides distributed processing, fault tolerance, reliability, native iterative processing capability, etc.
- **APIs and Library:-**These Flink APIs and Libraries provide a diverse capability to Flink as listed below.
  - Dataset API:-This API handles the data at the rest that is generated from various sources, for example by reading text or CSV files or from local collections and allows user to implement transformations like mapping, filtering, joining, grouping, etc.
  - Some best practices to follow while working in Dataset API are
    - **print() –** Use it for fast printing a dataset
    - **collect() –** Use for fast retrieval of dataset
    - **name() –** Use it on an operator for easy searching in logs

Components of Flink System

- **Table API:-**Table API enables users to perform ad-hoc analysis through languages like SQL for relational stream and batch processing. It can be embedded in Data Set and DataStream APIs of Flink in both Java and Scala.
- Actually, it allows users to run SQL queries on the top of Flink that saves them from writing complex code for data processing.
- The key concept of the Table API is a Table that represents a table with relational schema. Using Data Set or DataStream one can create Tables. It converts into a Dataset or DataStream or registers in a table catalog using a Table Environment.
- A Table is always bound to a specific Table Environment. It is not possible to combine Tables of different Table Environments.

Components of Flink System

- **Gelly**

- It is the graph processing engine which allows users to run operations for creating, transforming and processing the graph. It also provides the library of the algorithm to simplify the development of graph applications.
- Gelly can transform and modify graph using high-level functions that are similar to those provided by batch processing APIs. Its APIs are available in both Java and Scala. Scala methods are implemented as wrappers on top of Java operations.
- A dataset of vertices and edges represents a graph in Gelly. The unique ID of DataSet defines a vertex and a value while source ID defines an edge, target ID, and value.
- Transformations and Utilities are the methods of Graph class that include transformations, graph metrics, mutations and neighborhood aggregations.

Components of Flink System

- **Flink ML –** Machine Learning for Flink in Scala, which provides intuitive APIs and efficient algorithm to handle machine learning applications in Apache Flink.
  - So, as we know machine learning algorithms are iterative in nature, Flink provides native support for iterative algorithm to handle the same quite effectively and efficiently.
  - It supports Supervised learning(Optimization framework, Multiple linear regression and SVM algorithms) and Unsupervised learning (k-Nearest Neighbors join) along with data preprocessing. One of the key concepts of FlinkML is its scikit-learn inspired pipelining mechanism that allows building complex data analysis pipelines.
- **Flink CEP:-**It allows easy detection of complex event patterns in streams of data that is useful in finding matching sequences to get insights of data. For comparing and finding matching events the pattern **API** easily define the complex event patterns. So, the patterns have different states in which the user can define conditions for events.

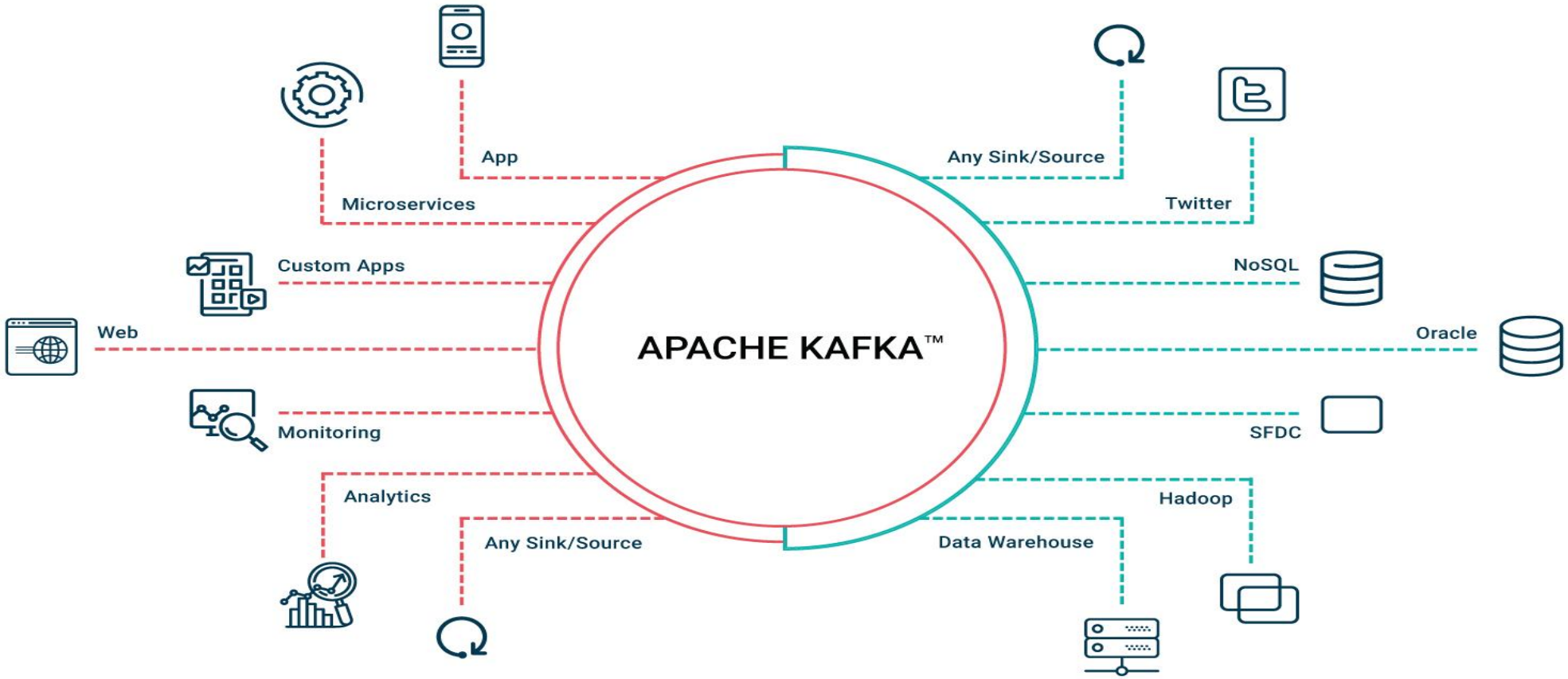
Apache Kafka For Flink

- Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on premise as well as cloud environments.
- Kafka is run as a cluster of one or more servers that can span multiple datacenters or cloud regions. Some of these servers form the storage layer, called the brokers. Other servers run Kafka connect to continuously import and export data as event streams to integrate Kafka with your existing systems such as relational databases as well as other Kafka clusters.
- They allow you to write distributed applications and micro services that read, write, and process streams of events in parallel, at scale, and in a fault-tolerant manner even in the case of network problems or machine failures.

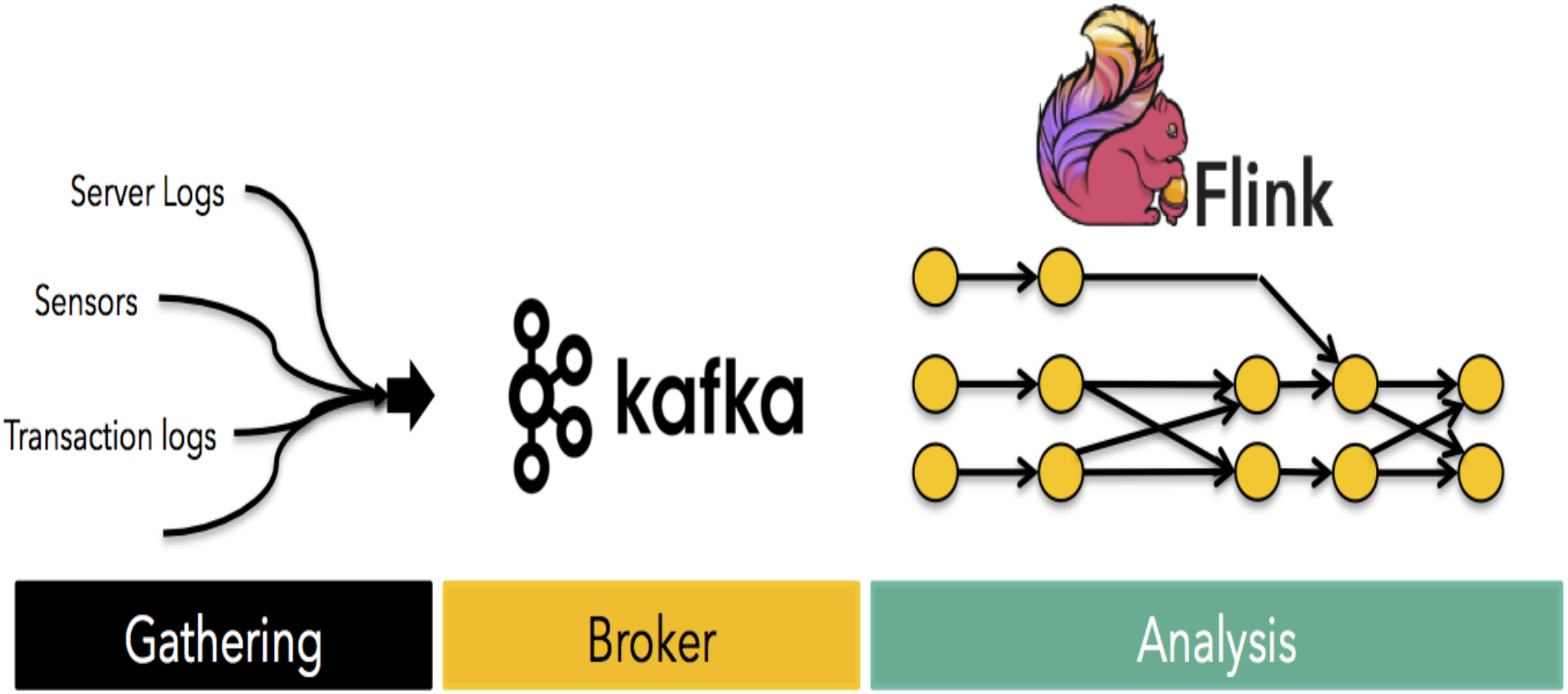
Apache Kafka For Flink

- An event records the fact that "something happened" in the world or in your business. It is also called record or message in the documentation. When you read or write data to Kafka, you do this in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers. Here's an example event:
  - Event key: "Alice"
  - Event value: "Made a payment of $200 to Bob"
  - Event timestamp: "Jun. 25, 2020 at 2:06 p.m.”
- **Producers** are those client applications that publish (write) events to Kafka, and **consumers** are those that subscribe to these events. In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability that Kafka is known for. For example, producers never need to wait for consumers. Kafka provides various guarantees such as the ability to process events exactly-once.

An Overview of Kafka Processing



Kafka and Flink Integration



Kafka and Flink Integration

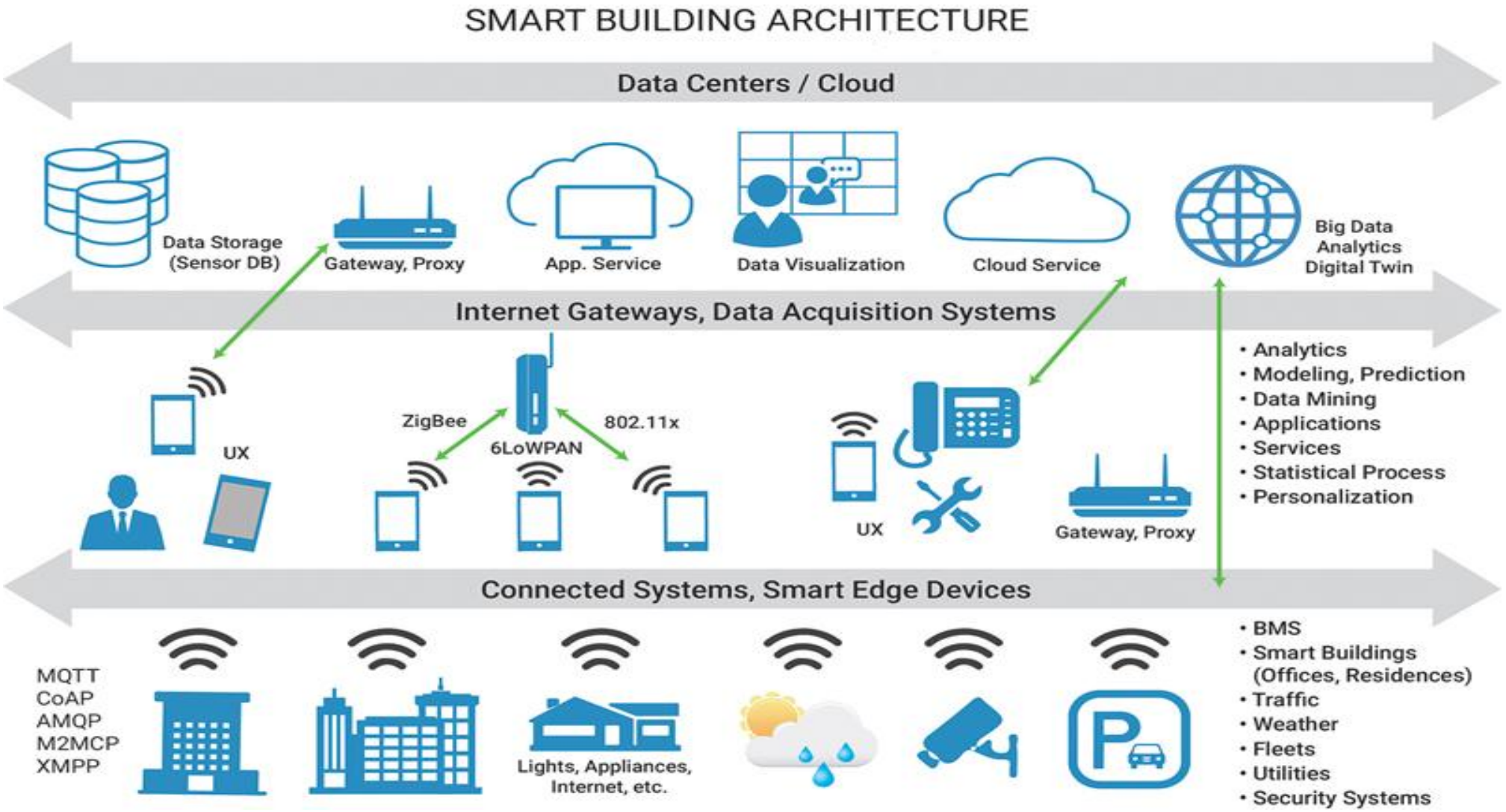
Introduction to Smart Building

- These are the buildings that have a smart infrastructure that is in charge of obtaining information to automate different processes such as lighting, access, air conditioning, video surveillance, parking, or security.
- Thanks to the data obtained through IoT (Internet of Things) devices and their interconnection, a building can automatically manage all the resources optimizing their performance.
- It can also provide valuable information for an individual to make better decisions to further improve the livability and efficiency of the building. The functionalities offered by smart buildings go beyond home automation systems.
- A smart building is one that uses technology to enable efficient and economical use of resources, while creating a safe and comfortable environment for occupants.

Benefits of Smart Building

- Smart buildings generate a large volume of valuable building data about how they are being utilized. Analyzing this data can give you insight regarding usage patterns and trends, so that you can make informed decisions on how to optimize your building, bringing the following advantages:
  - Increased productivity
  - Reduce energy consumption
  - Reduce Energy Consumption.
  - Automation
  - Provide Valuable Building Insights
  - Real-Time Data
  - Reduce Overall Costs
  - Improve Occupant Comfort & Wellbeing
  - Improve Work Efficiency

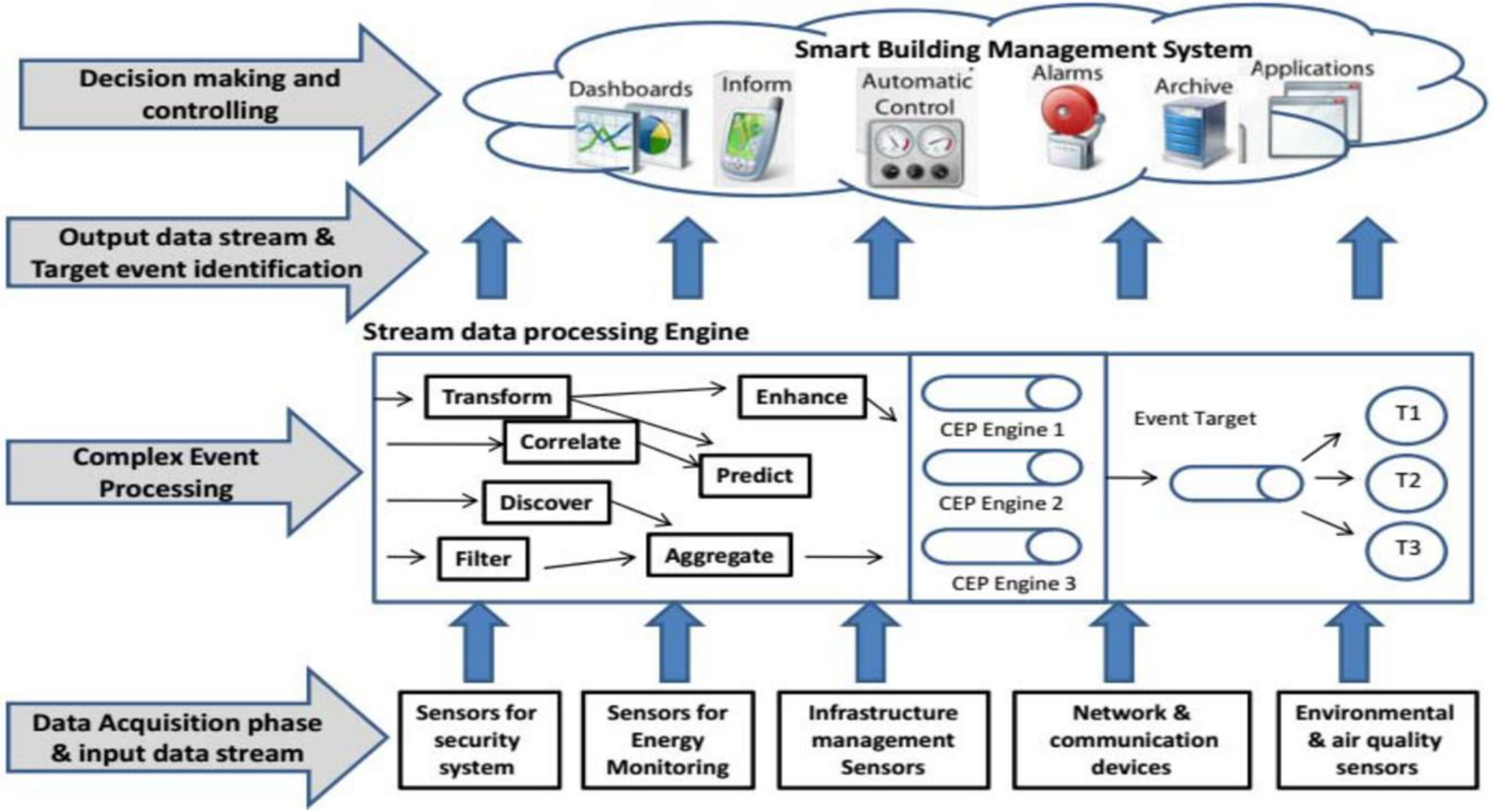
A Smart Building Prototype



Smart Buildings as CEP Application

- Smart buildings equipped with sensors and electronic devices as a Cyber Physical Systems (CPS) offer great research perspective to explore communication, computation and controlling of physical devices by using real time Complex Event Processing (CEP) and analytics.
- CPS like Smart-Building involves the integration of several types of equipment interoperability, maintainability, signaling, bandwidth, reliability, security, privacy, authentications, data storage, heterogeneity and cost effectiveness are the critical issues to be addressed.
- In addition, interactivity and dynamic feedback, context correlation with respect to streaming events and uncertainties in event composition are also identified as major challenging issues.
- To handle all these challenges integration of smart buildings with CEP and real-time data analytics are emerging as a new area of research.

CEP for Smart Building



CEP for Smart Building

- Presently buildings are equipped with lighting, heating, air conditioning, elevator control, security systems along with fire and smoke detection and other necessities.
- All these systems are manually driven and operating at the cost of building spaces, resources, energy and cost effectiveness.
- Smart buildings aim to enhance efficiency, through the interconnection of all these systems in a cost-effective way with upgraded hygiene, safety, productivity, and living standard of those who live or work inside the walls.
- Integration, monitoring and controlling several building assets by distributed sensing, communication and perception in the background of high-speed data analytics.

Smart Building Operations

➢Different operations in Smart Buildings can be followed as:-

- ❖ Energy Consumption Management
- ❖ Motion Detection System
- ❖ Lightning control System
- ❖ Air Quality Monitoring System
- ❖ Temperature Monitoring System
- ❖ Heating and Ventilation System
- ❖ Elevator Management System
- ❖ Occupancy Detection System
- ❖ Remote Control System
- ❖ Security System

Smart Building Operations

- **Energy Consumption Management:-**Smart home devices like thermostats and appliances are highly efficient because they can automatically adjust to changing energy needs. Plus, they can detect inefficiencies so electricity, water and gas waste can be minimized. Your energy consumption can be reduced exponentially when smart devices run your home.
- **Motion Detection System:-** motion sensor, or motion detector, is an electronic device that uses a sensor to detect nearby people or objects. Motion sensors are an important component of any security system. When a sensor detects motion, it will send an alert to your security system, and with newer systems, right to your mobile phone.
- **Lightning control System:-**Having an automatic home light control system will let you turn the lights on and off with just a tap of a button. You can also choose to turn on all the lights or just select lights in an area. Some residential lighting control systems could even let you control outdoor lighting.

Smart Building Operations

- **Air Quality Monitoring System:-**Smarter Technologies' smart air quality monitoring solution is able to detect carbon dioxide levels, noxious gases and pollutants, sending real-time data to a central management dashboard. It provides enhanced visibility, improved situational awareness and earlier indications of pollution hotspots.
- **Temperature Monitoring System:-**Room alert provides far greater visibility into temperatures throughout the building by making it easy to deploy sensors throughout a building. This can help identify inefficiencies with a building HVAC and possible energy wasted over heating or cooling an area in the building.
- **Heating and Ventilation System:-** Heating, ventilation and air conditioning (HVAC) systems have a significant impact on both comfort and costs in any building. Modern buildings require smart HVAC systems that create comfortable, healthy and safe environments for the occupants, while minimizing energy consumption and increasing sustainability.

Smart Building Operations

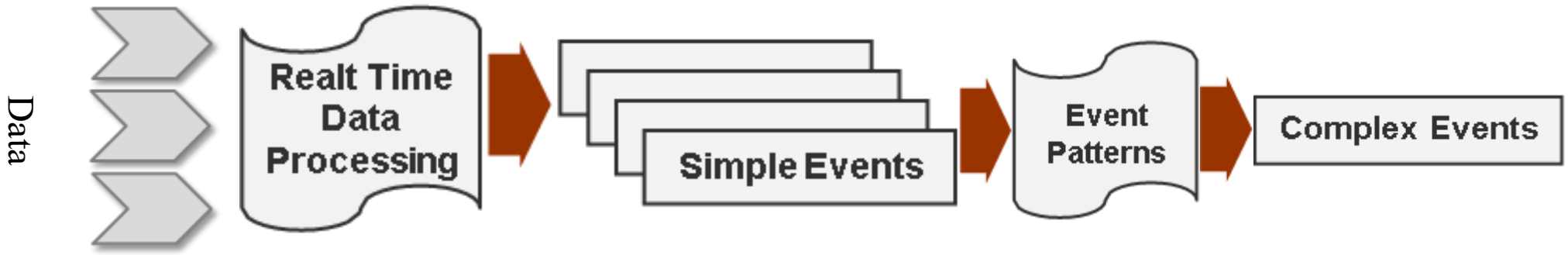
- **Elevator Management System:-** It guides the lift in what order to stop at Floors, when to close and open the door. This plc based smart elevator control system project consists of three sensors, on each floor these three sensors senses the load in elevator. On pressing the start button twice, the elevator moves till the second floor.
- **Occupancy Detection System:-** Occupancy monitoring lets you know the presence of people in your workplaces in real time. Sensors can be placed on desks and in doorways to allow you to see usage and monitor space and equipment.
- **Smart Security System**:-Smart security is a close ally of intelligent buildings. It uses the same technologies wireless networks and the internet of things that give users the ability to customize and monitor their security systems in the home and manage security over a wider area and respond to threat in the workplace.

Complex Events Vs Atomic Events

- **Complex Events:-** These are composed (composite event) or derived (derived event) from occurred atomic or other complex event instances, e.g. according to the operators of an event algebra or as a result of applying an algorithmic function or process to one or more other events
  - Included events are called components while the resulting complex event is the parent event
  - First event instance contributing to the detection of a complex event is called initiator, where the last is the terminator; all others are called interiors
- **Atomic Event :-**An atomic event (also raw event or primitive event) is defined as an instantaneous (at a specific point in time), significant (relates to a context), indivisible (cannot be further decomposed and happens completely or not at all) occurrence of a happening.

From Simple to Complex Events

Event Pattern :-An event pattern (also event class, event definition, event schema, or event type) describes the structure and properties of an (atomic or complex) even it describes on an abstract level the essential factors that uniquely identify the occurrence of an event of that type, i.e. its detection condition(s) and its properties with respect to instantiation, selection and consumption.



From Simple to Complex Events

Example Event Algebra Operators:

- **Sequence Operator (;):** *(E1;E2)*
- **Disjunction Operator (**Ú**):** *(E1* Ú *E2)* , at least one
- **Conjunction Operator (**Ù**):** (E1 Ù E2)
- **Simultaneous Operator (=):** *(E1 = E2)*
- **Negation Operator (¬):** *(E1* Ù Ø E2 *)*
- **Quantification (Any):** Any(n) E1, when n events of type E1 occurs
- **Aperiodic Operator (Ap):** *Ap(E2, E1, E3)* , E2 Within E1 & E3
- **Periodic Operator (Per):** Per(t, E1, E2) , every t time-steps in between E1 and E2

CFP for occupancy detection

➢States and Events Definition for Occupancy Detection

- ❖ Among thousand of incoming events a monitoring system may for instance receive the following same sources.
- ❖ If occupancy is detected(A) and Temperature is high(B) and Light status is On(C) Then It may be inferred from the events that is it a complex event based on the event coming form streaming data we may infer useful decision.

➢ The above state of rules may be expressed as following:

- ❖ ce1(A,B,C)^(A='Occupancy status',B='Temperature status', Light Status(C)) Then maintain the temperature.
- ❖ To Enable complete home security a couple of rules is needed to infer the states and make a meaningful decision. For this we need a complex events that will be enabled to define the following situation.

CFP for occupancy detection

- Ce1(A,B,C) : motion detected(A) ^ light intensity(B) ^face recognition(C) then
  - Then the ECA(event condition and action) rules can be written to assign reactions for situations like:
    - ce1(A,B,C)^(A='yes',B='low',C='authorized')then turn_on_light and unlock_door.
    - ce1(A,B,C)^(A='yes',B='low',C='unauthorized')then turn_on_light and notify_user.
    - ce1(A,B,C)^(A='yes',B='high',C='authorized') then unlock_door.
- **States and Event Definition for Gas/Fire Detection**
  - List of States: home status(A):A state which specifies whether house is empty or occupied.
  - gas_conc(B): A state saying whether the concentration of inflammable gases is high or not temp_change(C): A state saying whether the temperature is gradually increasing and above threshold value.

CFP for occupancy detection

➢ List of Actions:

- ❖ open_windows: It is an atomic action to open windows.
- ❖ alarm(home_status(A)):- It is a complex action to trigger an alarm when the home is occupied and not when empty.
- ❖ notify user: It is an atomic action to send a push notification to the user device.
- ❖ fire_sprinkler: It is atomic action to turn on fire sprinkler.

➢ States and Event Definition for Humidity

- ❖ List of States:
  - Humidity Ratio(A):A state which specifies about room moisture condition. A{ Yes, No)
  - CO_2(B): A state saying whether the CO_2 level is B{High or Low}
  - user_decision(C): A state saying whether to turn on HVAC let say C{Yes, No}

CFP for occupancy detection

- **Based On state of events we can infer the complex events.**
- **List of Events:**
  - HVAC (status(I)): This event occurs when the heating and ventilation level is at{ high or low}
  - List of Actions:
    - Notify_user: It is an atomic action to send a push notification to the user device.
    - Start heating_ventilation (user_decision(C)): It is a complex action to start if users decision is yes and do noting if user decision is no.
    - Stop heating and ventilation(user_decision (C)) It is an atomic event based on list of states we can infer the appropriate decision.
    - State Rules: When heating and ventilation i.e. low or high and the humidity level is low and High (as per user decision) as well in that case system is in critical state and system will not generate user notification on hourly basis until ventilation is not reached on a threshold value.

CFP for occupancy detection

- ce3(A,B,C):- Heating (A)^CO_2(B)^user_decision (C)
- Then the ECA(event condition action) rule can be written to assign reaction for situations like:
- ce3(A,B,C) ^ (A='low',B='no',C='yes')then notify_user about the low Heating and Ventilation.
- ce3(A,B,C)^(A='low',B='yes',C='no') then notify_user about the high Heating and Ventilation.
- ce3(A,B,C)^(A='low',B='yes',C='yes') then notify_user about the high humidity.

States and Events Definition for motion detection

- List of States:
  - motion_detected(A): A state specifying whether there is motion in front of the camera. **A{Yes,no)**.
  - Motion status (B): Specifies whether the motion status is **B{Yes or No**}.
  - Motion_times_tamp (C): A state specifying the time at motion is detected.

CFP for occupancy detection

- **List of Events:**
  - break-in(status(I)): An event which is triggered when there is a motion detected into the building premises.

    Let say(yes, no)
- **List of Actions:**
  - unlock_door: It is an atomic action to unlock the door.
  - notify_user: It is an atomic action to send push notification and image of an unknown person or notify.
  - **State Rules**: When someone tries to forcefully enter into your house then a break-in event is generated and system goes into critical state and will send an alert notification to user devices and set alarm on for a possible motion detected along appropriate location detected.

USE CASE 1

- **Objective :-To monitor the CO2 levels for tracking occupancy and predict anomalies using the Streaming data.**
  - Possible anomalies are : overcrowding, fire hazard, etc.
  - CO2 levels are numerical data which correspond to two levels : safe & unsafe
  - The dataset used is **UCI Occupancy Detection Dataset from Kaggle**
  - The system will collect the data from different sensors and collect at a location for event processing at regular intervals and create a summary at a dashboard.
  - If the CO2 levels are very high, it means there is a potential fire in the area. If it is slightly higher, It corresponds to overcrowding/AC malfunction. Hence we need to learn the boundary that separates the two conditions of emergency.

Dataset Description

```
"date","Temperature","Humidity","Light","CO2","HumidityRatio","Occupancy"
"1","2015-02-04 17:51:00",23.18,27.272,426,721.25,0.00479298817650529,1
"2","2015-02-04 17:51:59",23.15,27.2675,429.5,714,0.00478344094931065,1
"3","2015-02-04 17:53:00",23.15,27.245,426,713.5,0.00477946352442199,1
"4","2015-02-04 17:54:00",23.15,27.2,426,708.25,0.00477150882608175,1
"5","2015-02-04 17:55:00",23.1,27.2,426,704.5,0.00475699293331518,1
"6","2015-02-04 17:55:59",23.1,27.2,419,701,0.00475699293331518,1
"7","2015-02-04 17:57:00",23.1,27.2,419,701.666666666667,0.00475699293331518,1
"8","2015-02-04 17:57:59",23.1,27.2,419,699,0.00475699293331518,1
```

Learning the Threshold for Event Definition

The value of threshold is learned using the Support Vector Machine which is a great tool to identify decision boundary for classification problems.

We fit a hyperplane with parameters **w** and **b** .

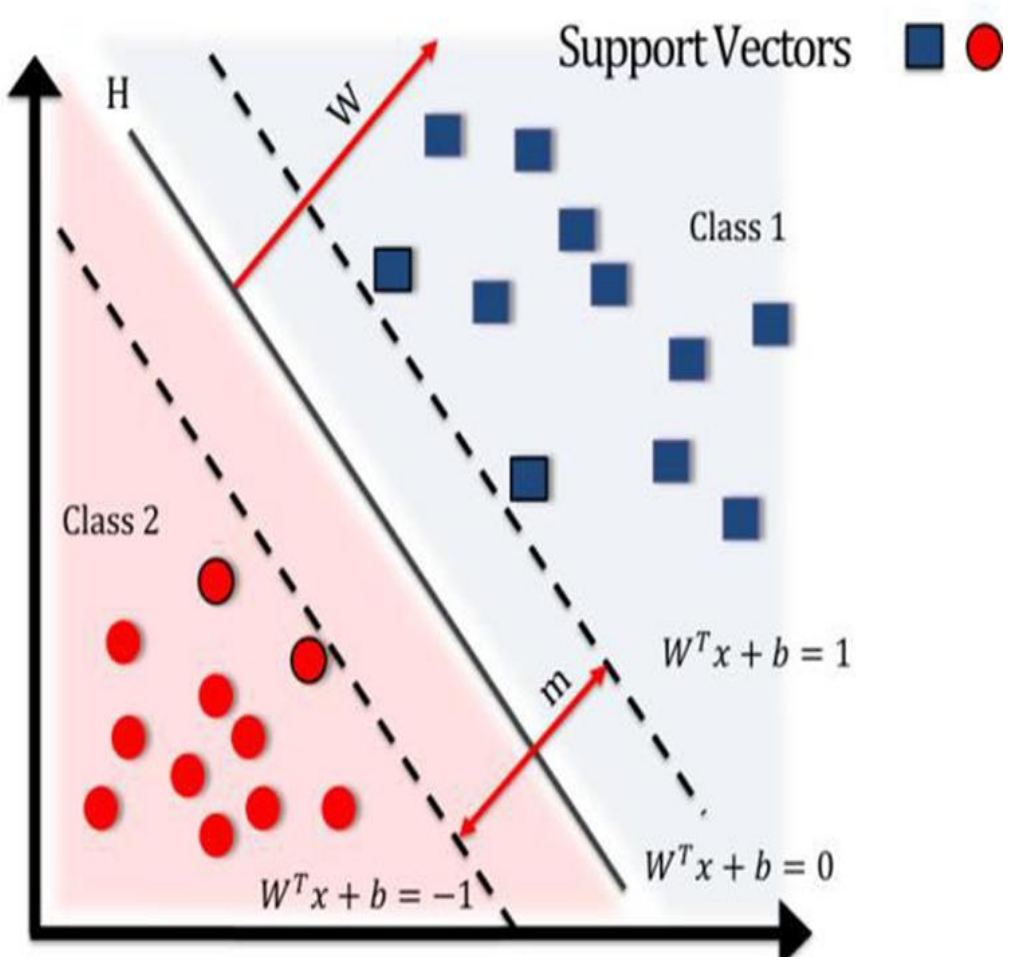
Thus if x is an input, then :

If w*x + b < 0 then x belongs to class 2 . (unoccupied)

If w*x + b >= 0 then x belongs to class 1. (occupied)

We obtained the value b = -12.86
And w = 0.0169

The input features are CO2 levels.
The prediction variable is 0 & 1

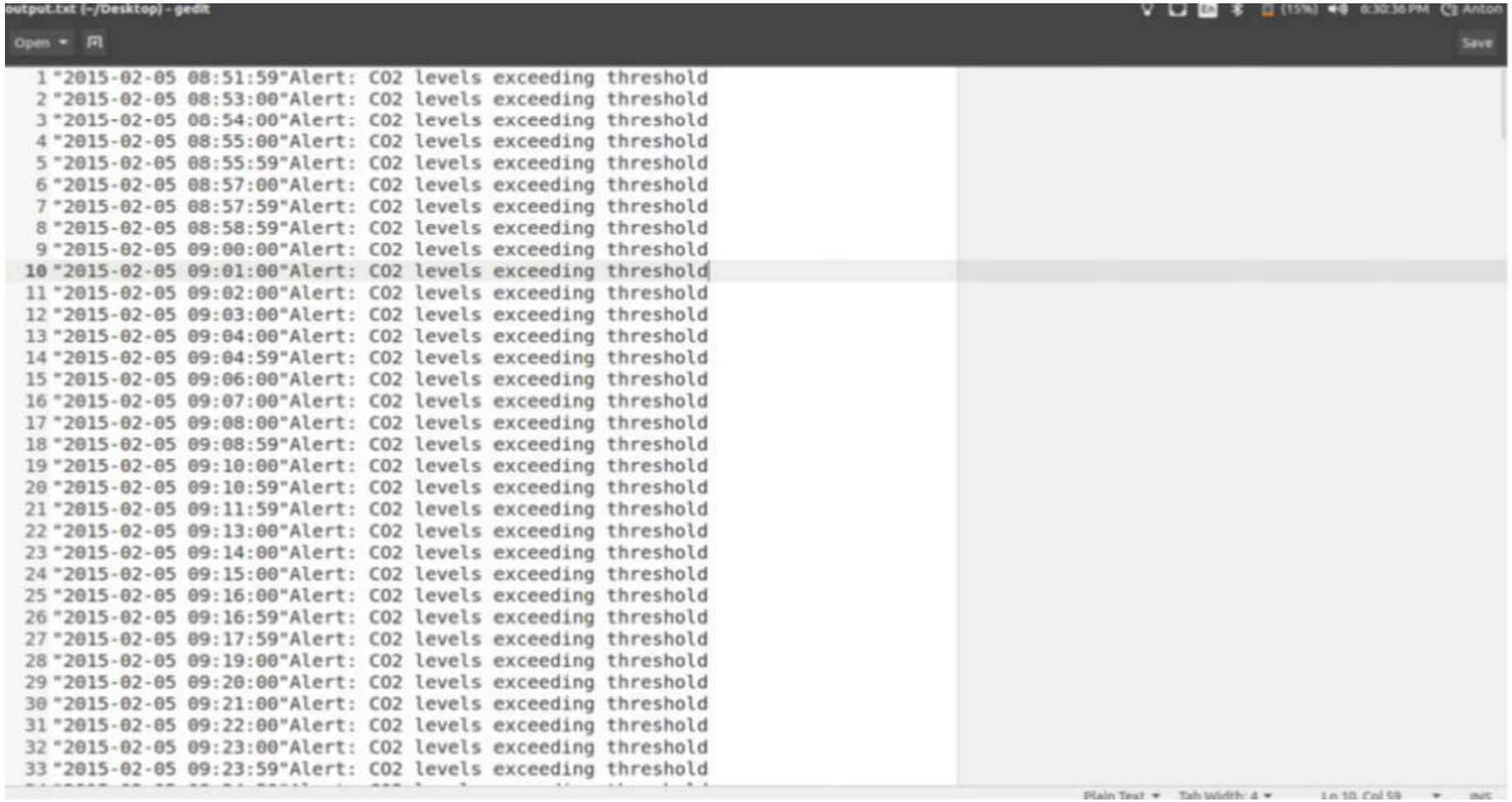


Defining Threshold Using SVM

Implementation Details

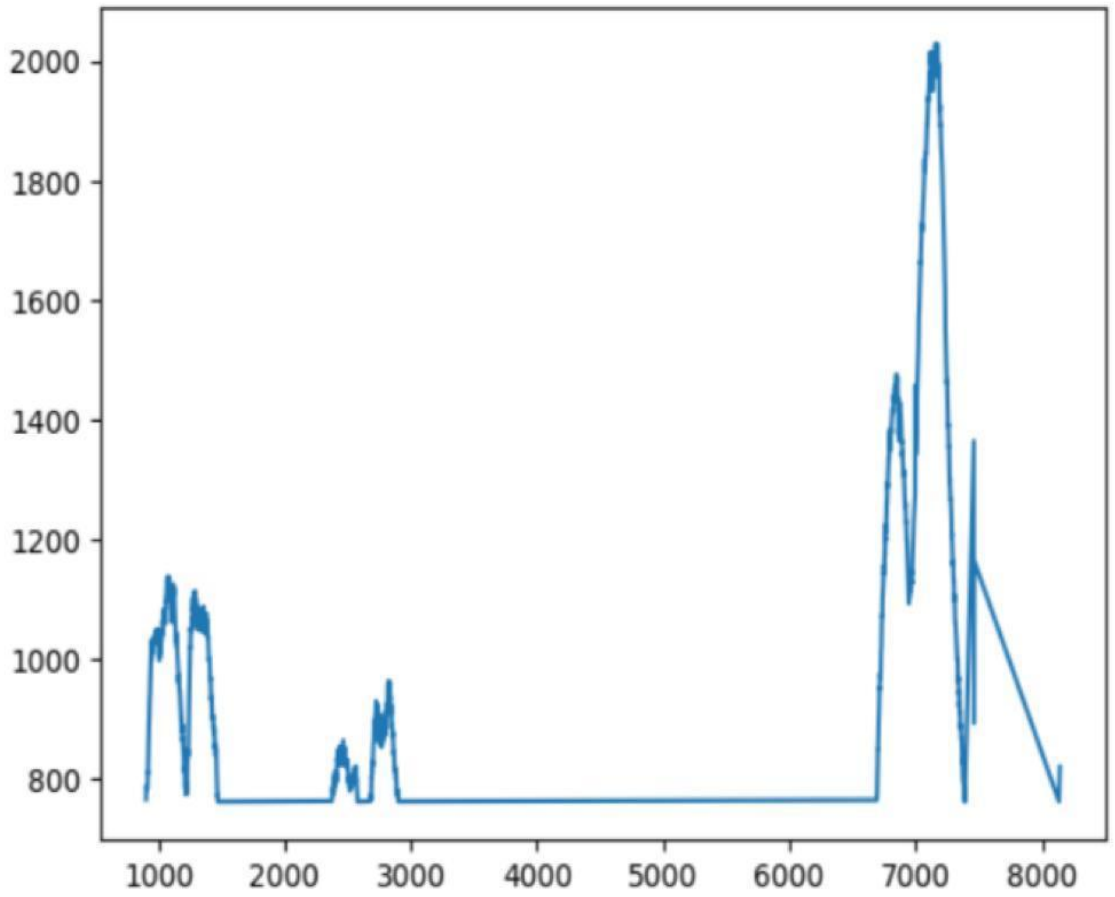
- Initialization of system for stream processing.
- Apache Fink, Apache Kafka and zookeeper has been initialized with Hadoop installation.
- Dataset is preprocessed for streaming.
- As there is a single source of data generation, data aggregation step is omitted.
- The dashboard is showing the alarm generation from CO2 level. Hence end users are alerted with this.
- Rule set is learned for alert generation using the learned value from dataset.
- The boundary for the above threshold depend on the CO2 levels. This threshold from the dataset is around 800 ppm.
- Based on the inference from our model, well ventilated or unoccupied buildings have a CO2 levels between 200 to 300 ppm.
- To predict we used an SVM model that predict this threshold value equal to 761.211 ppm.

Alert Generation with threshold Value



Results

- After initialization of code and running the script, we get a real time plot of CO2 level on the y axis vs the time on the x axis.
- The final output after data generation is shown :
- As soon as we cross the threshold, there is a alert generation as shown on the DataStream.
- We introduced a delay of time in the signal generation from the dataset to model the real time data generation.

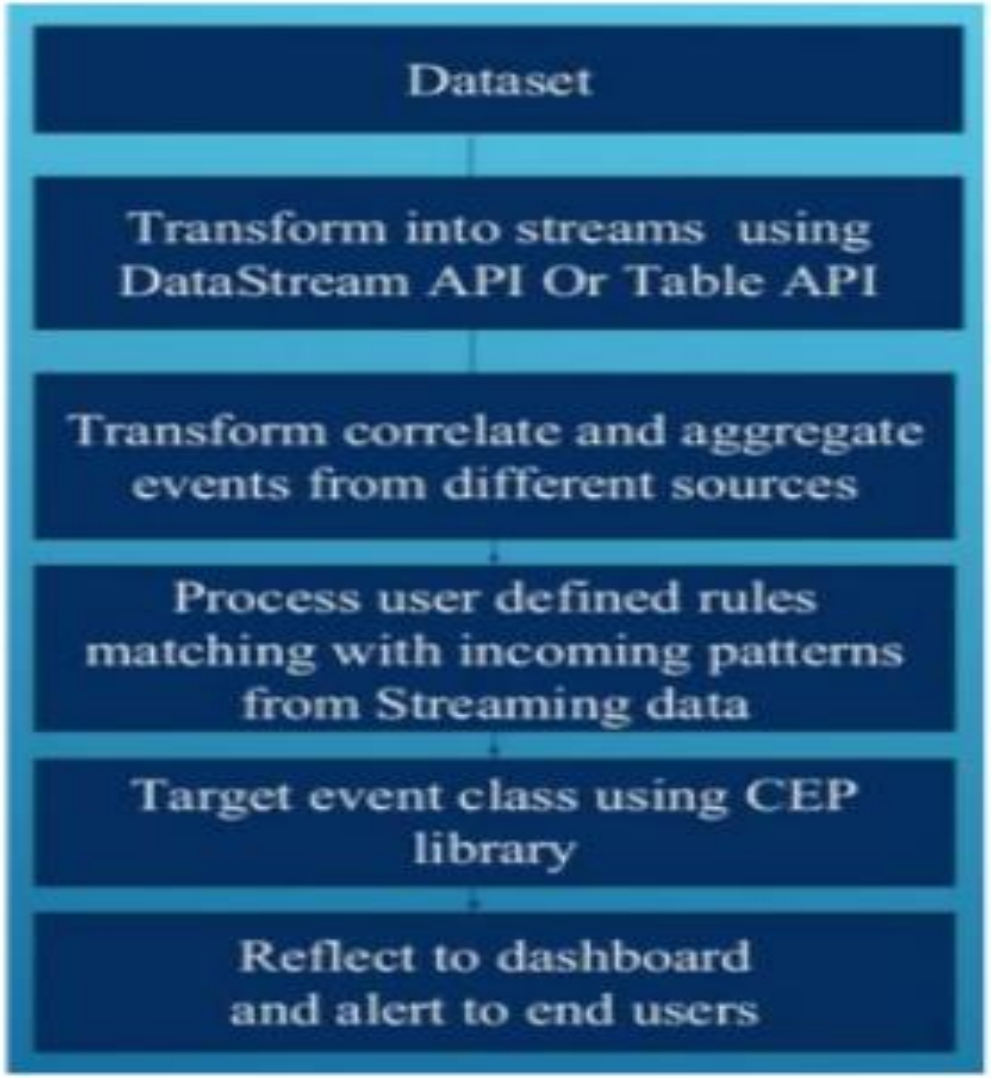


Co2 Vs Time

USE CASE 2

- **Objective:-Design a CEP system and real time implementation based on humidity and humidity ratio predict whether a particular premises is occupied or not and send an alert to occupant about the soaring high humidity ratio if detected.**
  - Predicting whether a particular premises is occupied or not and sending an alert to the occupant about the soaring high humidity ratio using CEP & our aim is to get higher accuracy while predicting the humidity ratio.
  - Represent Temperature, Occupancy and time data of particular location as DataStream using kafka.
  - Construct a Complex event processing system in flink which processes streaming data and alerts the occupant about the soaring high humidity ratio.

Flow Chart Of Implementation



Preprocessed data

	humidity	ratio	occupancy
0	23.7000	0.004764	1
1	23.7180	0.004773	1
2	23.7300	0.004765	1
3	23.7225	0.004744	1
4	23.7540	0.004767	1

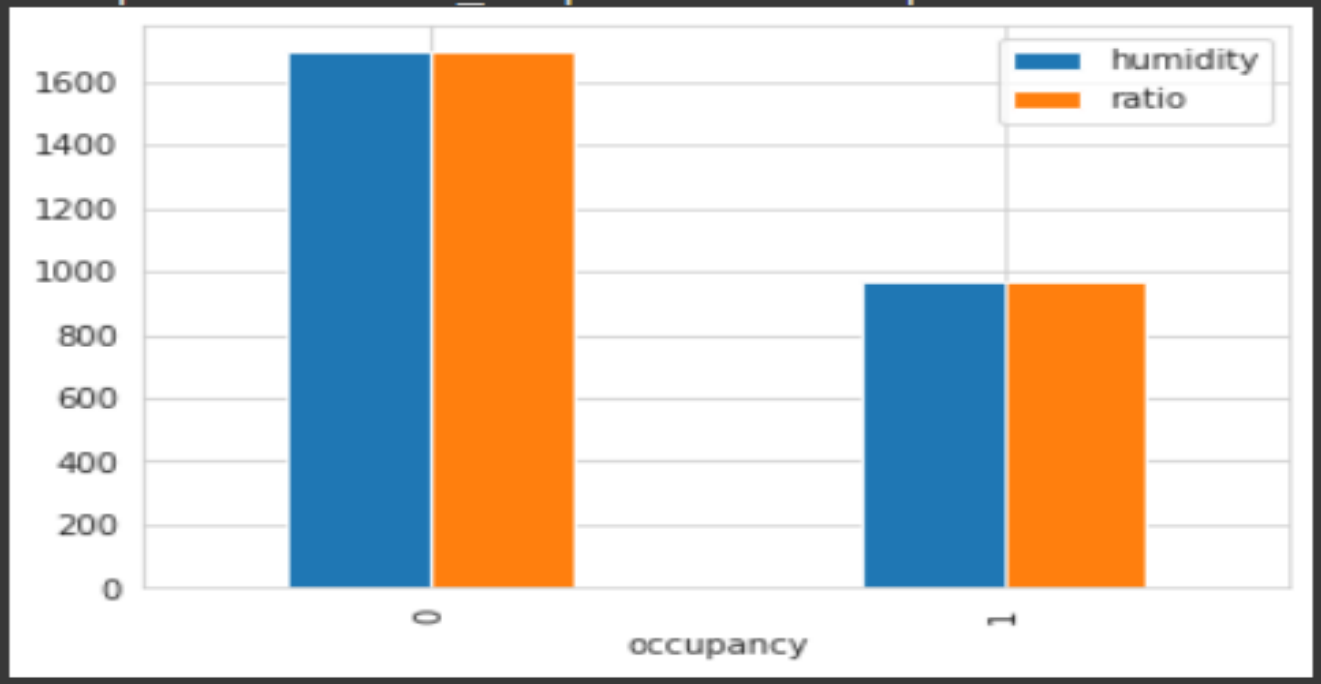
Steps for Implementation

- Preprocessing the dataset.
- Transforming data into streams using DataStream API
- Transform Correlate and aggregate events from different sources
- Process user defined rules matching with incoming patterns from streaming data
- Target event class using CEP library
- Reflect to dashboard and alert to end user.

Distribution of the occupancy 0 and occupancy 1 data

```
df.groupby('occupancy').count().plot.bar()
```

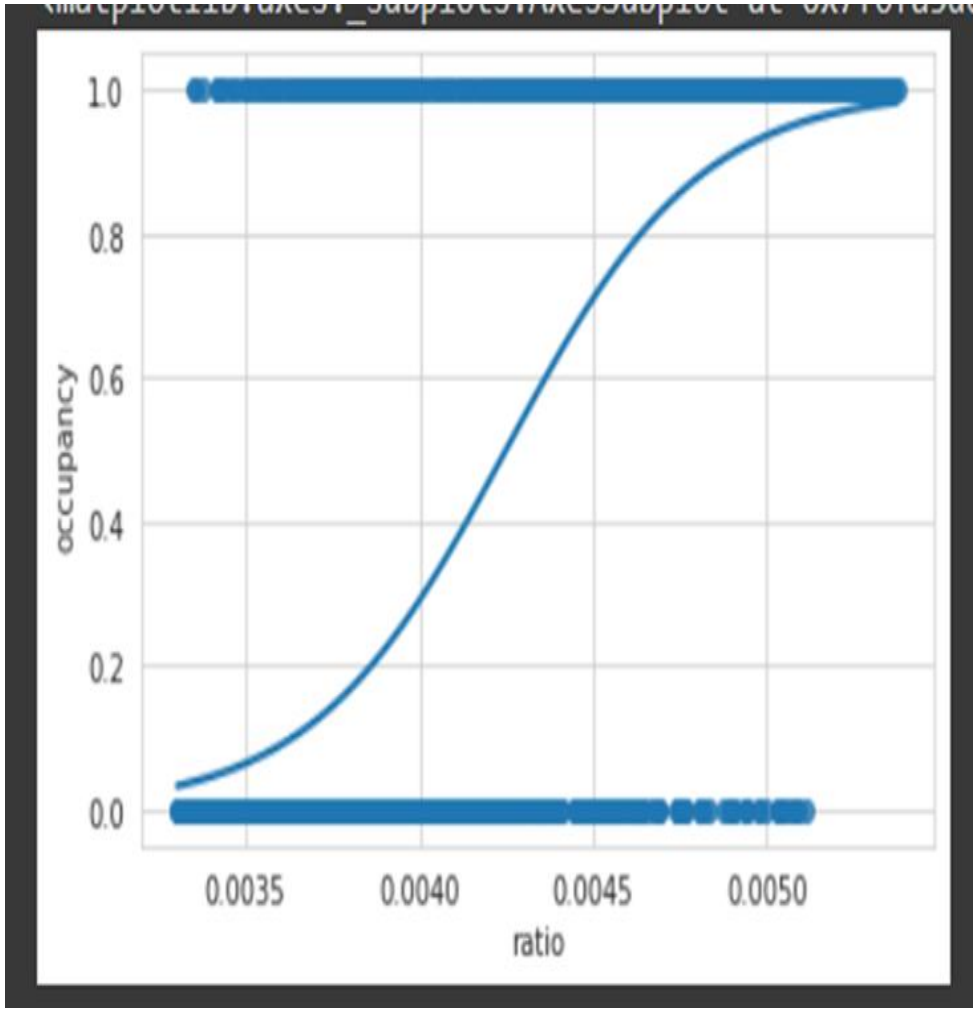
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f967a7e90>
```



Distribution of the occupancy 0 and occupancy 1 data.

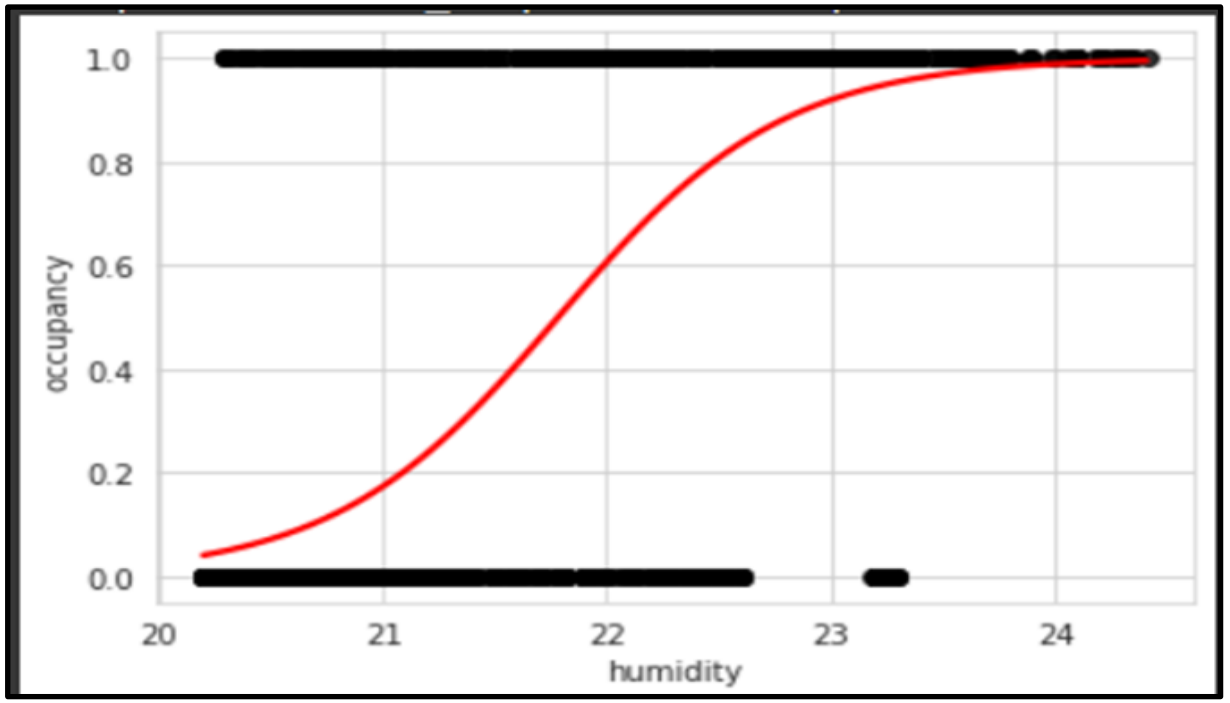
Establishing Relationship between humidity ratio and occupancy.

- Different models were tried out using the scipy library to find the model that gave the best accuracy.
- Data was managed using the pandas library.
- 90% of the data was used for training while 10% was kept to make up the test set.
- Logistic regression model was trained using 90% percent data the features were humidity and humidity ratio while the dependent variable was occupancy.
- The model performed well predicting correct occupancy values 85% of the time.



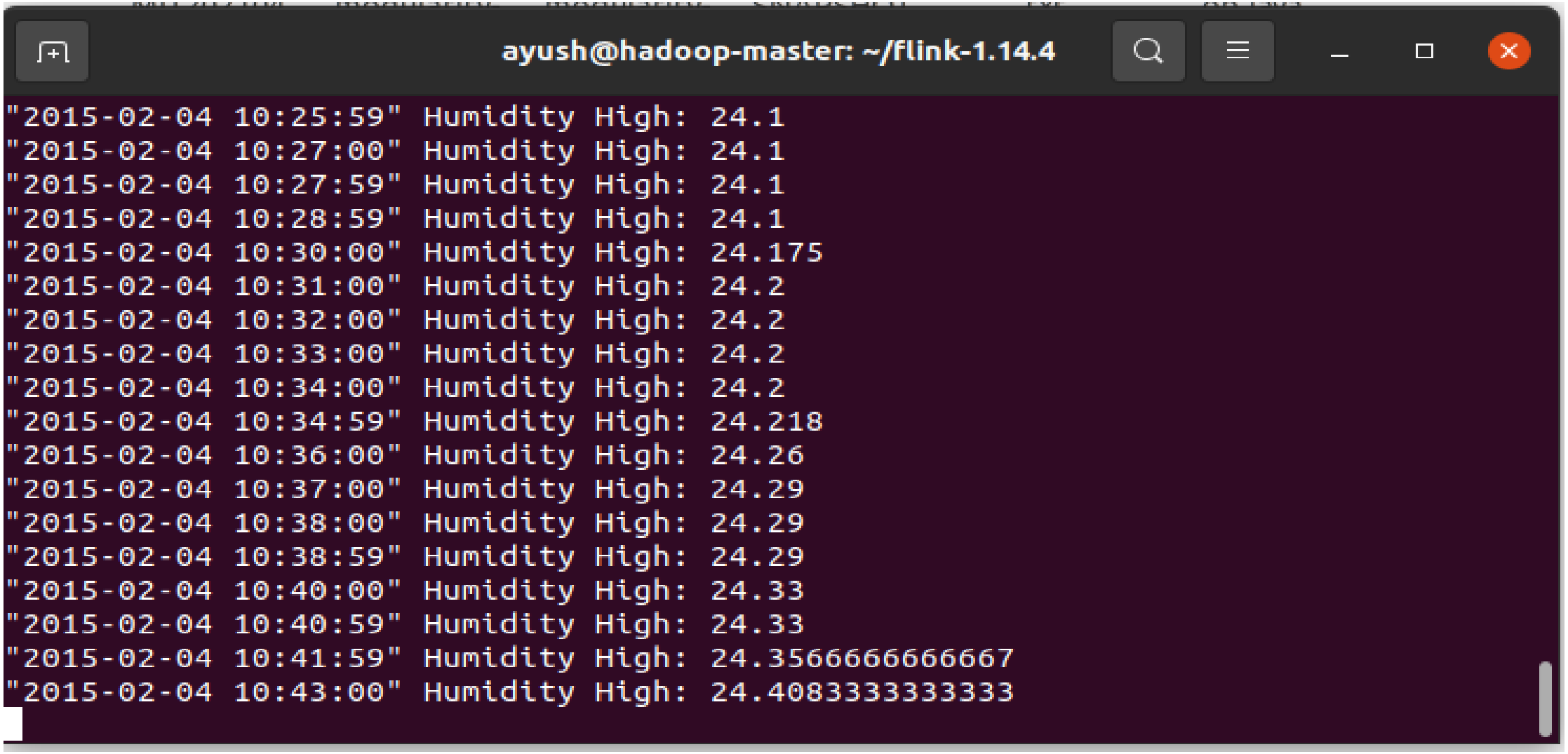
Humidity ratio vs occupancy

Distribution of the occupancy vs humidity



Distribution of the occupancy vs Humidity.

Streaming Result



Result of Prediction

```
[15] new_pred_class = logreg.predict(x_test)
```

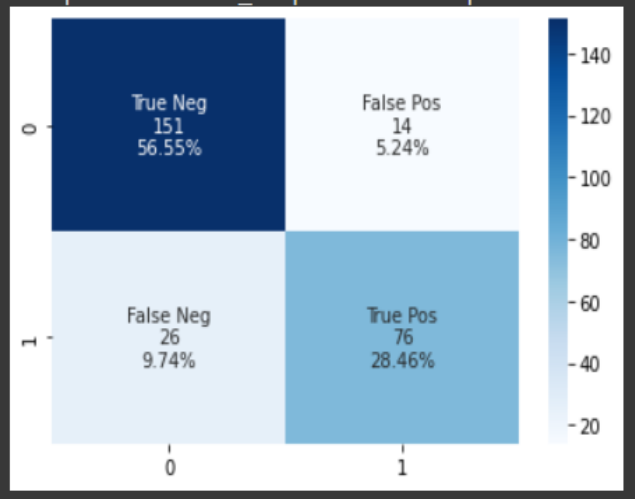
```
[16] from sklearn.metrics import accuracy_score
     score = accuracy_score(y_test, new_pred_class)
```

```
[17] score * 100
```

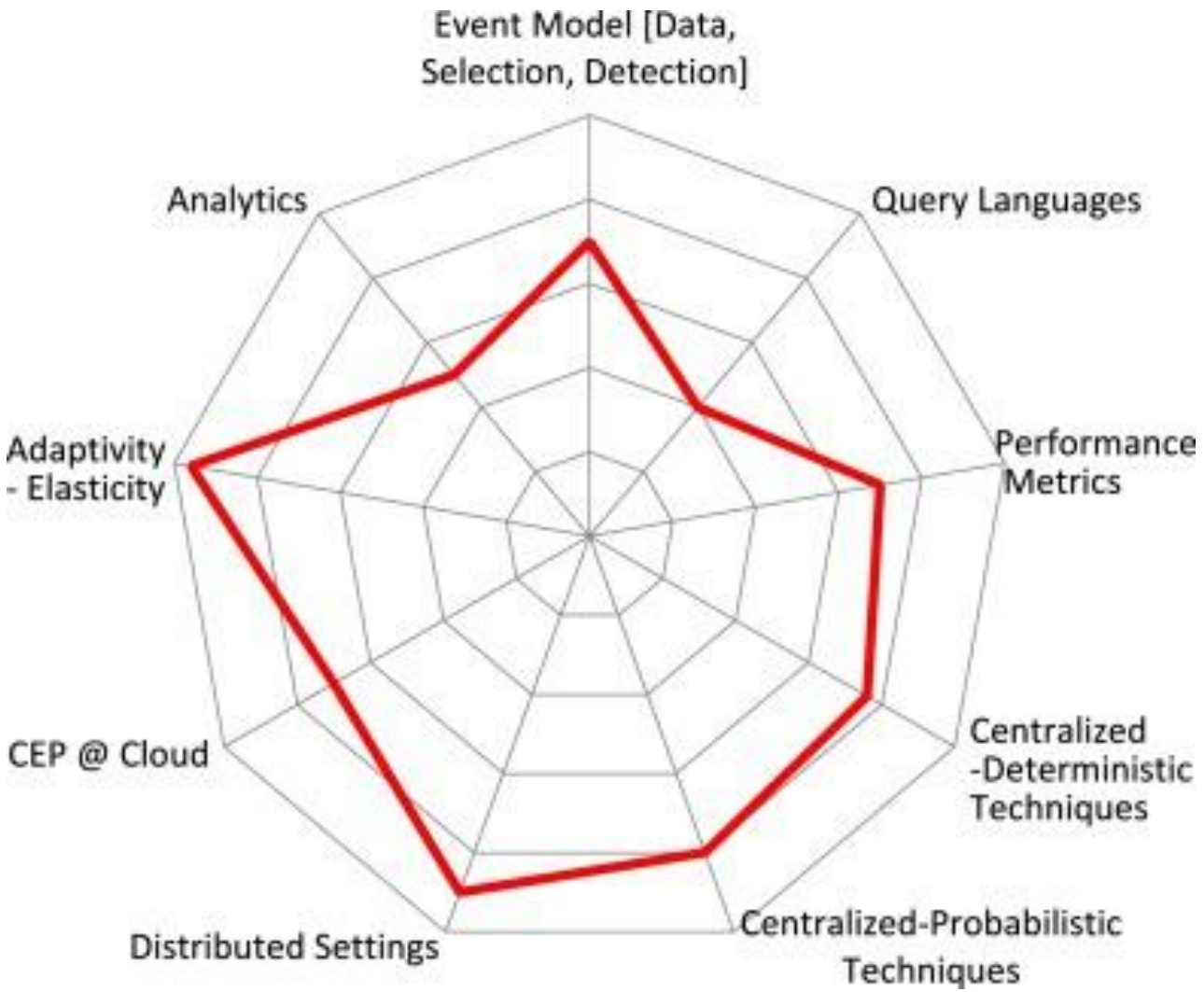
```
85.0187265917603
```

```
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fb8e8e990>
```



Challenges and Research directions of CEP



References

- **Advanced Analytics with Spark** by S. Ryza, U. Laserson, S. Owen and J. Wills
- **Apache Spark documentation**
  - http://spark.apache.org/
  - http://spark.apache.org/docs/latest/programming-guide.html
- **Pyspark**
  - http://spark.apache.org/docs/latest/api/python/pyspark.html
- **Resilient Distributed Dataset: A Fault-tolerant Abstraction for in-Memory Cluster Computing.** M. Zaharia et al.
  - https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf