Data Analytics (IT-3006)

Kalinga Institute of Industrial Technology Deemed to be University Bhubaneswar-751024

School of Computer Engineering

3 Credit

Lecture Note – Unit 3

Course Contents

Sr #	Major and Detailed Coverage Area	Hrs
2	**Mining Data Streams**	10
	Introduction to Mining Data Streams, Data Stream Management Systems, Data Stream Mining, Examples of Data Stream Applications, Stream Queries, Issues in Data Stream Query, Sampling in Data Streams, Filtering Streams, Counting Distinct Elements in a Stream, Estimating Moments, ~~Querying on Windows – Counting Ones in a Window, Decaying Windows~~, Real-Time Analytics Platform (RTAP).	

Data Stream

Data Stream – Large data volume, likely unstructured and structured arriving at a very high rate, which requires real time/near real time analysis for effective decision making.

- It is basically continuously generated data and arrives in a stream (sequence of data elements made available over time). It is generally time-stamped and geo-tagged (in the form of latitude and longitude).
- Stream is composed of synchronized sequence of elements or events.
- If it is not processed immediately, then it is lost forever.
- In general, such data is generated as part of application logs, events, or collected from a large pool of devices continuously generating events such as ATM or PoS.

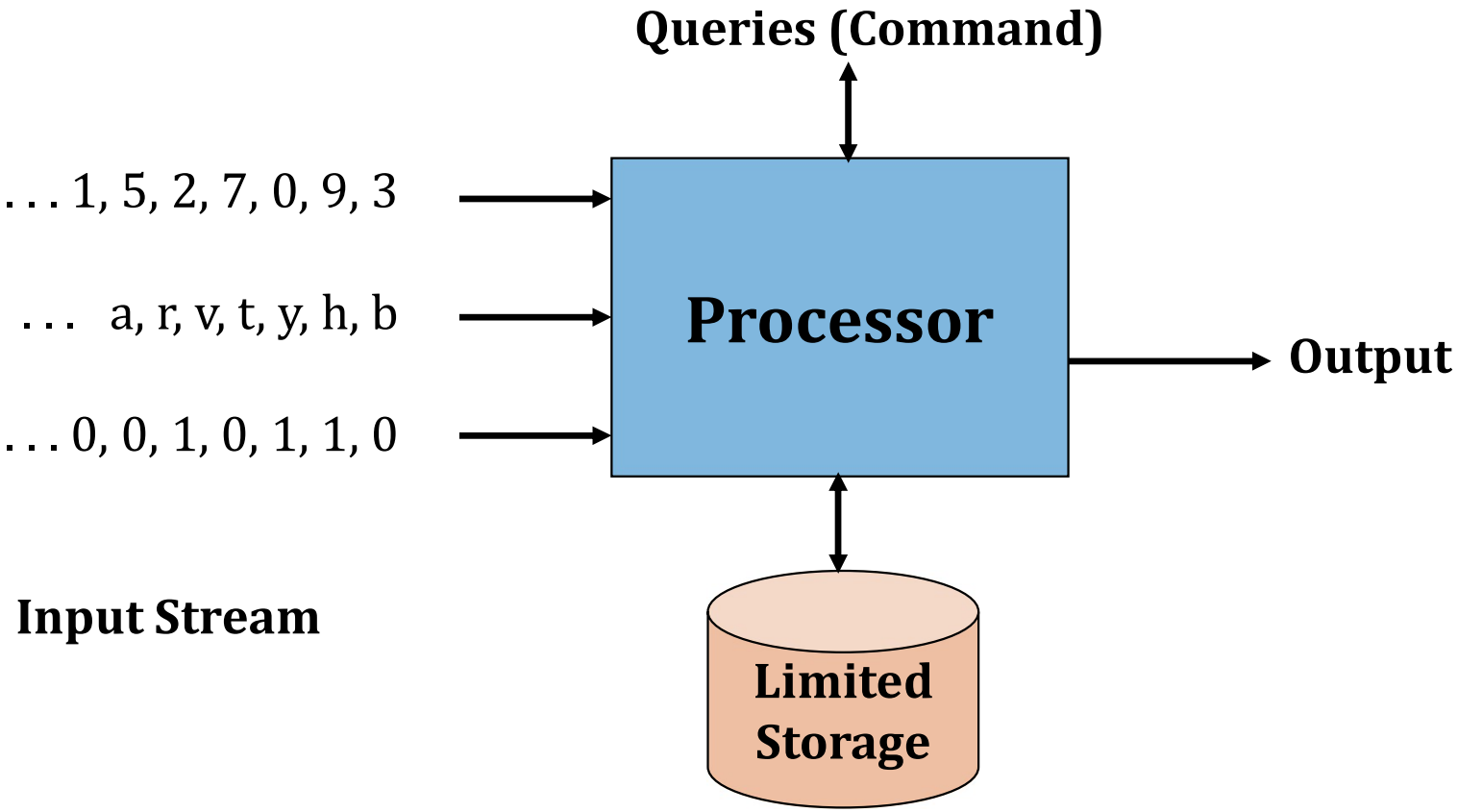
Example:

Data Center: Large network deployment of a data center with hundreds of servers, switches, routers and other devices in the network. The event logs from all these devices at real time create a stream of data. This data can be used to prevent failures in the data center and automate triggers so that the complete data center is fault tolerant.

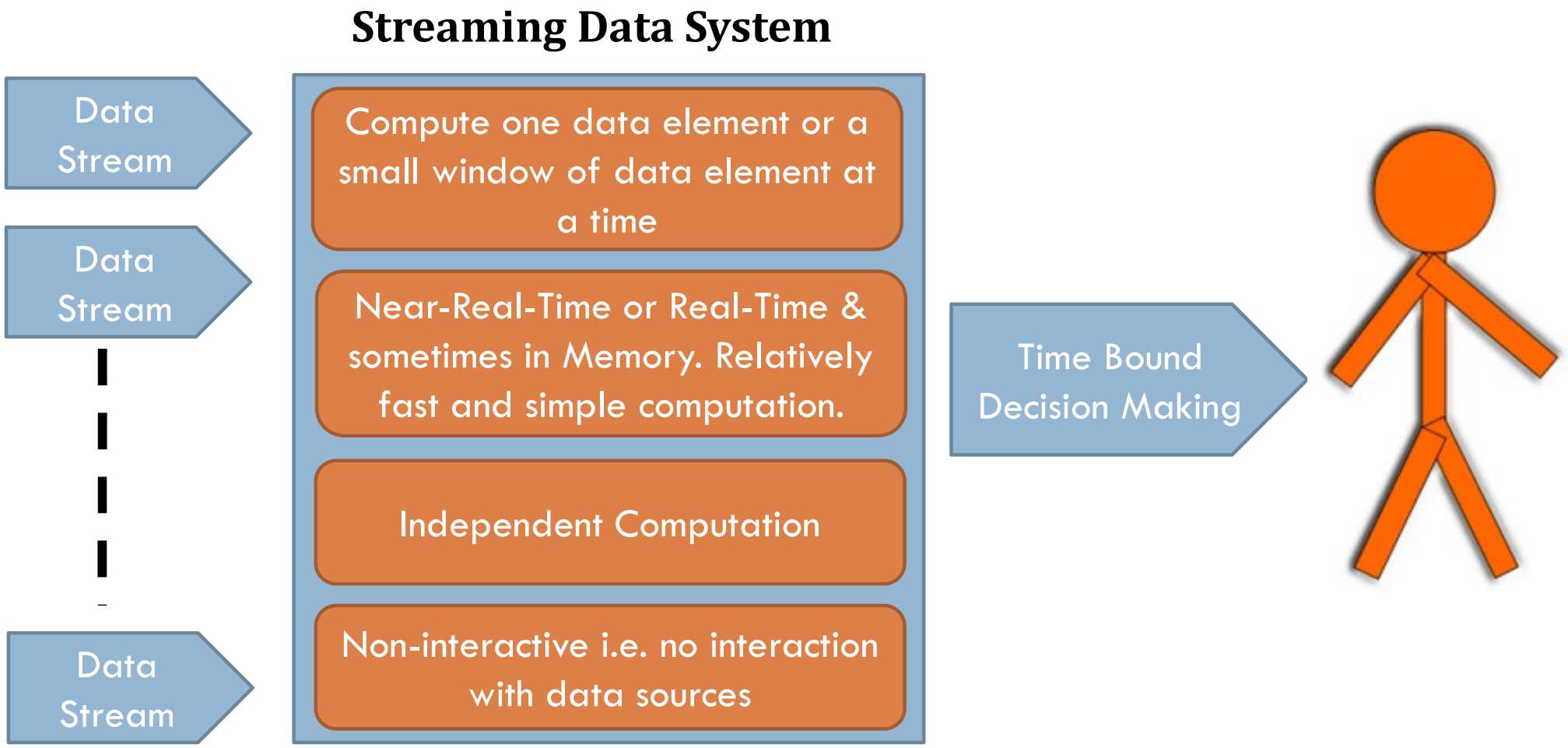
Stock Market: The data generated here is a stream of data where a lot of events are happening in real-time. The price of stock are continuously varying. These are large continuous data streams which needs analysis in real-time for better decisions on trading.

Basic Model of Stream data

- Input data rapidly and streams needn't have the same data rates or data types.
- The system cannot store the data entirely.
- Queries tends to ask information about recent data.
- The scan never turn back.



Streaming Data System



Data-at-Rest vs. Data-in-Motion

- **Data-at-rest** - This refers to data that has been collected from various sources and is then analyzed after the event occurs. The point where the data is analyzed and the point where action is taken on it occur at two separate times. For example, a retailer analyzes a previous month's sales data and uses it to make strategic decisions about the present month's business activities. The action takes place after the data-creating event has occurred. For data at rest, a batch processing method would be most likely.
- **Data-in-motion** - The collection process for data in motion is similar to that of data at rest; however, the difference lies in the analytics. In this case, the analytics occur in real-time as the event happens. For example – sensor data from self-driving vehicles. For data in motion, you'd want to utilize a real-time processing method.

Data-at-Rest vs. Data-in-Motion Infrastructure Option

Data-at-rest

Public Cloud

Public cloud can be an ideal infrastructure choice in such scenario from a cost standpoint, since virtual machines can easily be spun up as needed to analyze the data and spun down when finished.

Data-in-motion

Bare-Metal Cloud

Bare-Metal cloud can be an preferable infrastructure choice. It involves the use of dedicated servers that offers cloud-like features without the use of virtualization.

Streaming Data Changes over Time

Change can be periodic or sporadic

Periodic: evening, weekends etc

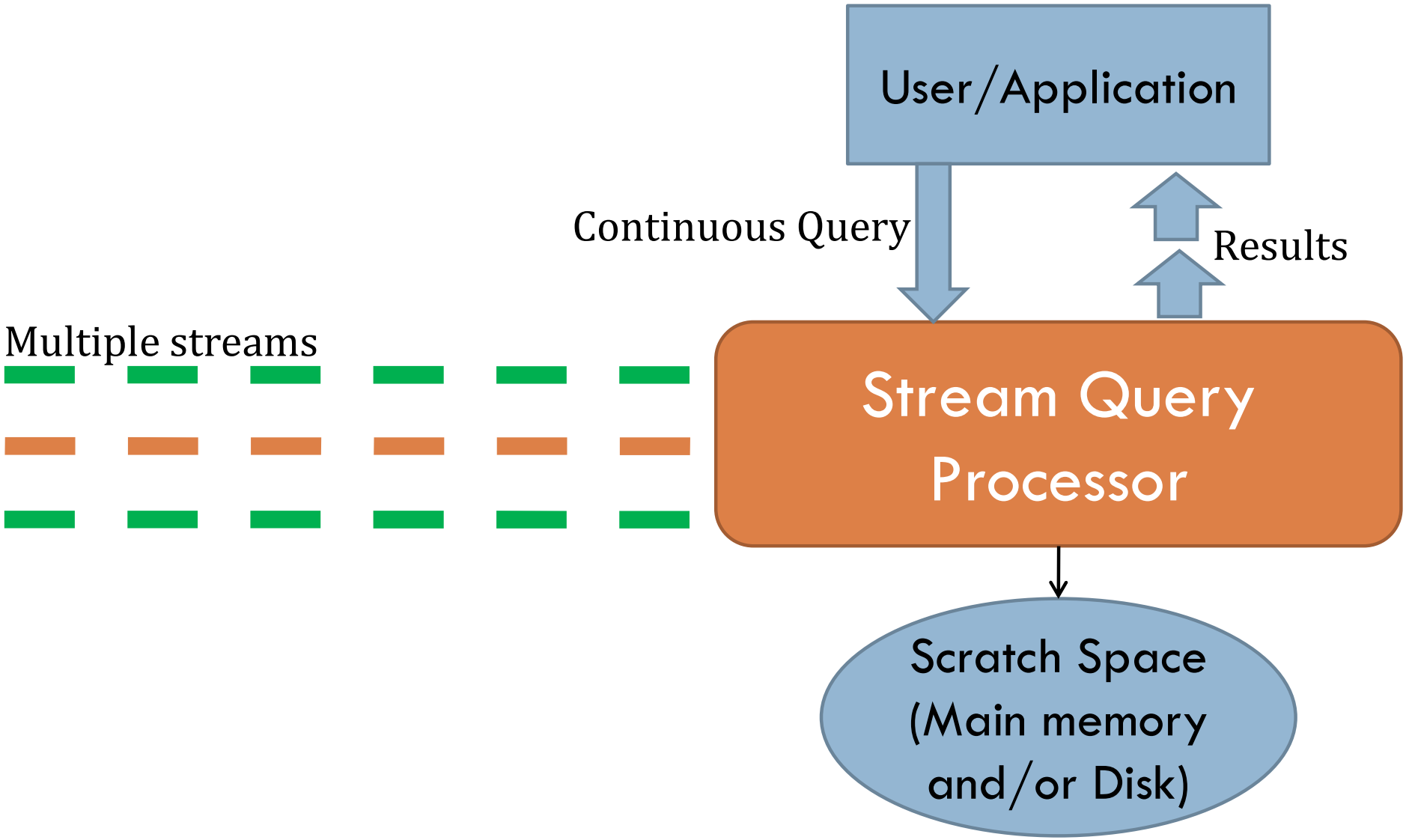
People post Facebook messages more in the evening in comparison to during day, working hours.

Sporadic: major events

In summary, streaming data:

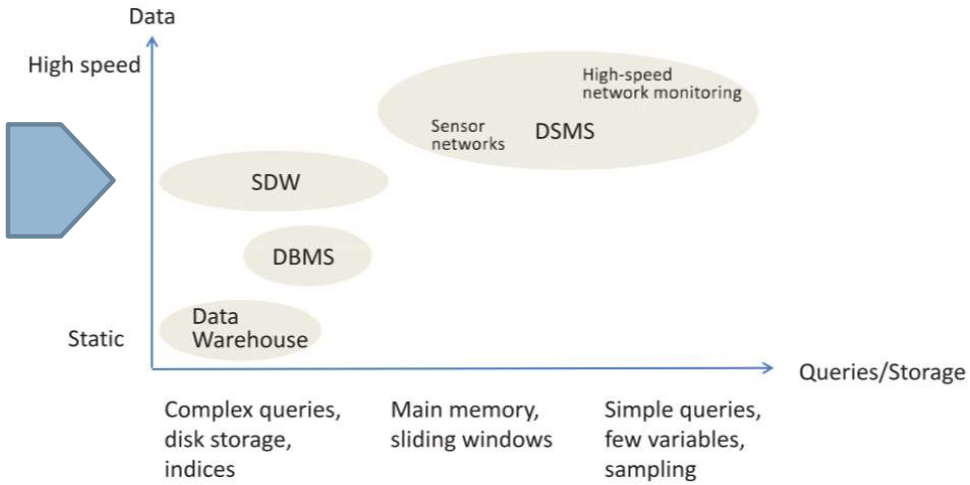
- Size is unbounded i.e. it continually generated and can't process all at once
- Size and Frequency is unpredictable due to human behavior
- Processing is must be relatively fast and simple

Architecture: Stream Query Processing



Data Stream Management Systems

- Traditional relational databases store and retrieve records of data that are static in nature and do not perceive a notation of time unless time is added as an attribute during the schema design.
- The model is adequate for legacy applications and older repositories of information, but many current and emerging application require support for online analysis of rapidly arriving and changing data streams.
- This has resulted in data stream management system (DSMS) with an emphasis on continuous query languages and query evaluation.
- There are two complementary techniques for end-to-end stream processing: Data Stream Management Systems (DSMSs) and Streaming.
- Comparison of DSMS and SDW with traditional database and warehouse systems, wherein data rates are on the y-axis, and query complexity and available storage on the x-axis.



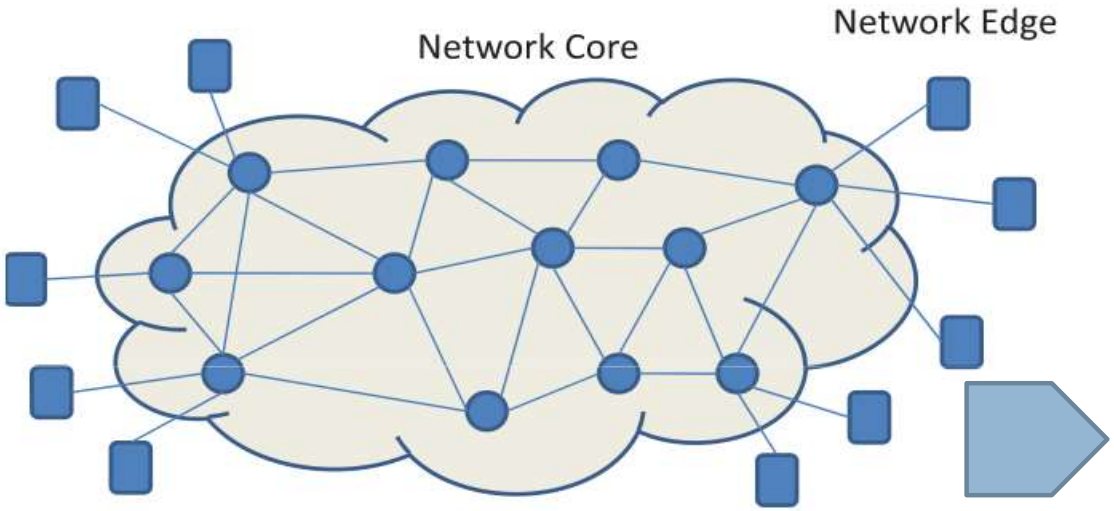
Summary difference between DBMS and DSMS

	DBMS	DSMS
Data	Persistent relations	Streams, time windows
Data access	Random	Sequential, One-pass
Updates	Arbitrary	Append-only
Update Rates	Relatively Low	High, bursty
Processing model	Query driven (pull-based)	Data driven (push-based)
Queries	One-time	Continuous
Query Plans	Fixed	Adaptive
Query Optimization	One query	Multi-query
Query Answers	Exact	Exact or approximate
Latency	Relatively high	Low

Summary difference between Traditional data warehouse and SDW

	Traditional data warehouse	SDW
Update frequency	Low	High
Update propagation	Synchronous	Asynchronous
Data	Historical	Recent and historical
ETL Process	Complex	Fast and light-weight

Network monitoring - A use case



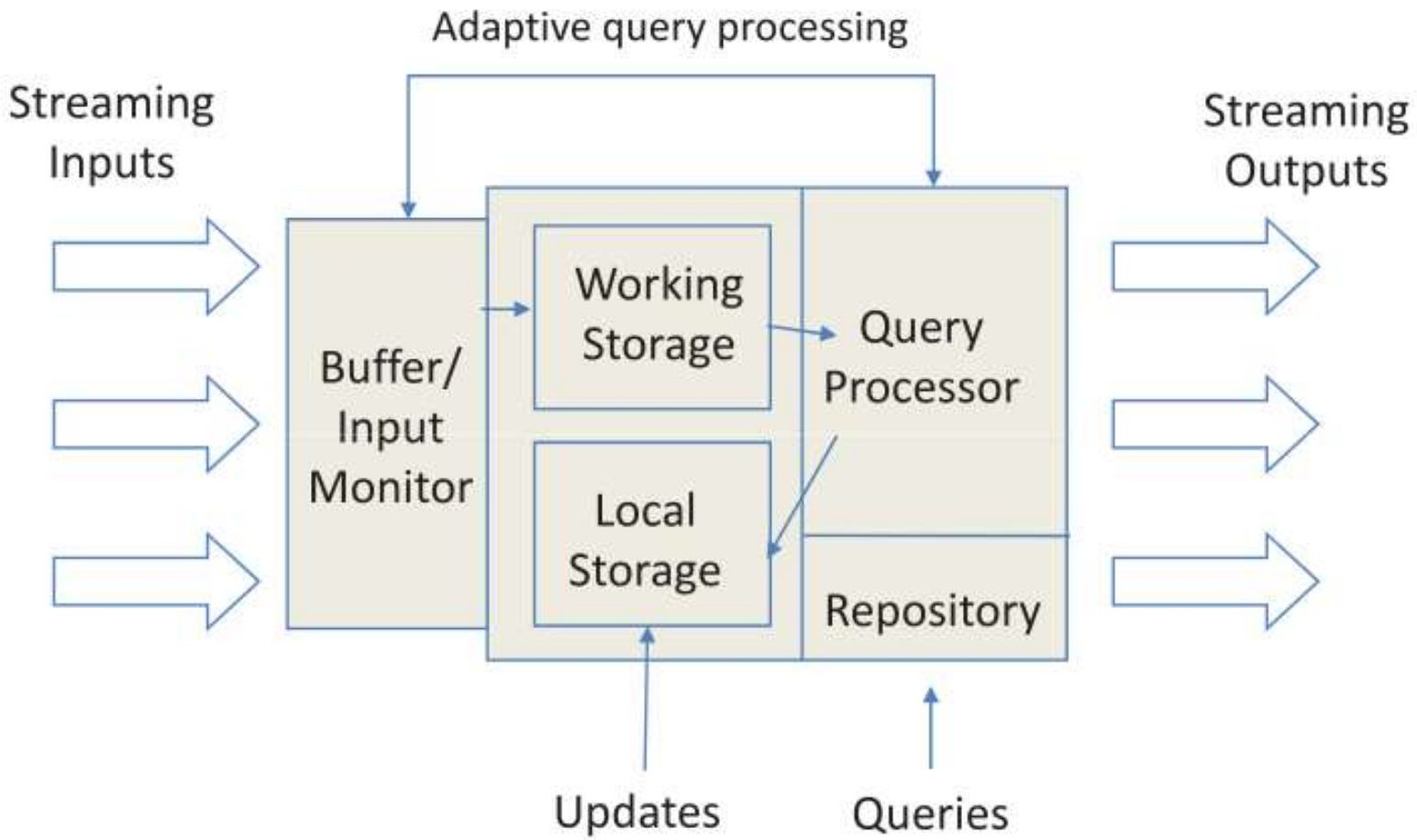
The network monitoring illustrates a simple IP network with high-speed routers and links in the core, and hosts (clients and servers) at the edge. A large network contains thousands of routers and links, and its core links may carry many thousands of packets per second; in fact, optical links in the Internet backbone can reach speeds of over 100 million packets per second.

- The traffic flowing through the network is itself a high-speed data stream, with each data packet containing fields such as a timestamp, the source and destination IP addresses, and ports.
- Other network monitoring data streams include real-time system and alert logs produced by routers, routing and configuration updates, and periodic performance measurements.
- However, it is not feasible to perform complex operations on high-speed streams or to keep transmitting terabytes of raw data to a data management system.
- Instead, there is a need of scalable and flexible end-to-end data stream management solutions, ranging from real-time low-latency alerting and monitoring, ad-hoc analysis and early data reduction on raw streaming data, to long-term analysis of processed data.

Network monitoring – DBMS, DSMS, SDW

- Database Management Systems (DBMSs) handle somewhat more dynamic workloads, consisting of ad-hoc queries and data manipulation statements, i.e., insertions, updates and deletions of a single row or groups of rows. On the other hand, DSMSs lie in the top right corner as they evaluate continuous queries on data streams that accumulate over time.
- SDWs, also known as Active Data Warehouses, combine the real-time response of a DSMS (by attempting to load and propagate new data across materialized views as soon as they arrive) with a data warehouse's ability to manage Terabytes of historical data on secondary storage.
- In applications such as troubleshooting a live network, the data rates may be so high that only the simplest continuous queries that require very little working memory and per-tuple processing are feasible, such as simple filters and simple aggregates over non-overlapping windows.
- In network monitoring, an SDW may store traffic streams that have been pre-aggregated or otherwise pre-processed by a DSMS, as well as various network performance and configuration feeds that arrive with a wide range of inter-arrival times, e.g., once a minute to once a day.

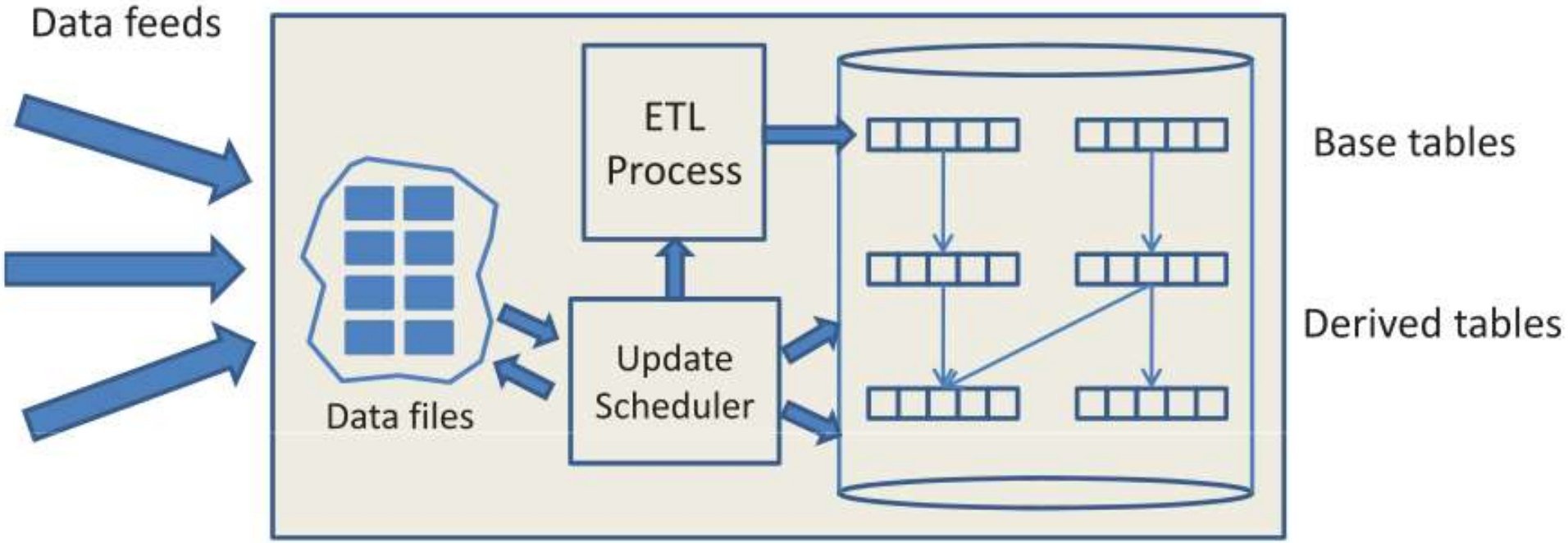
Reference architecture of a DSMS



Reference architecture of a DSMS cont...

- The input buffer captures the streaming inputs. Optionally, an input monitor may collect various statistics such as inter-arrival times or drop some incoming data in a controlled fashion (e.g., via random sampling) if the system cannot keep up.
- The working storage component temporarily stores recent portions of the stream and/or various summary data structures needed by queries. Depending on the arrival rates, this ranges from a small number of counters in fast RAM to memory-resident sliding windows.
- Local storage may be used for metadata such as foreign key mappings, e.g., translation from numeric device IDs that accompany router performance data to more user-friendly router names. Users may directly update the metadata in the local storage, but the working storage is used only for query processing.
- Continuous queries are registered in the query repository and converted into execution plans; similar queries may be grouped for shared processing. While superficially similar to relational query plans, continuous query plans also require buffers, inter-operator queues and scheduling algorithms to handle continuously streaming data. Conceptually, each operator consumes a data stream and returns a modified stream for consumption by the next operator in the pipeline.
- The query processor may communicate with the input monitor and may change the query plans in response to changes in the workload and the input rates.
- Finally, results may streamed to users, to alerting or event-processing applications, or to a SDW for permanent storage and further analysis.

Reference architecture of a SDW



Reference architecture of a SDW cont...

- Data streams or feeds arrive periodically from various sources, often in the form of text or zipped files.
- An update scheduler decides which file or batch of files to load next.
- The data then pass through an ETL process, as in traditional data warehouses. Examples of ETL tasks include unzipping compressed files, and simple data cleaning and standardization (e.g., converting strings to lower or upper case or converting timestamps to GMT).
- Base tables are sourced directly from the raw files, while derived tables correspond to materialized views (over base or other derived tables).
- Base and derived tables are usually partitioned by time so that arrivals of new data only affect the most recent partitions.

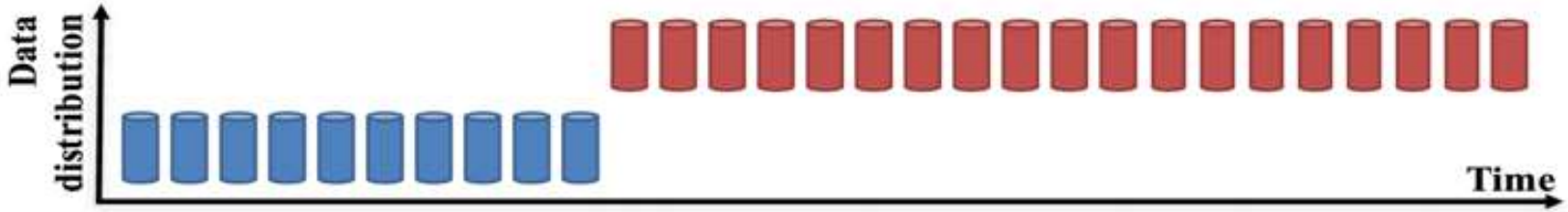
Mining Big Data Stream

- Mining big data streams faces three principal challenges: volume, velocity, and volatility.
- Volume and velocity require a high volume of data to be processed in limited time. Starting from the first arriving instance, the amount of available data constantly increases from zero to potentially infinity. This requires incremental approaches that incorporate information as it becomes available, and online processing if not all data can be kept.
- Volatility corresponds to a dynamic environment with ever-changing patterns. Here, old data is of limited use, even if it could be saved and processed again later.
- This can affect the data mining models in multiple ways:
  - Change of the target variable.
  - Change in the available feature information.
  - Drift.

Mining Big Data Stream cont…

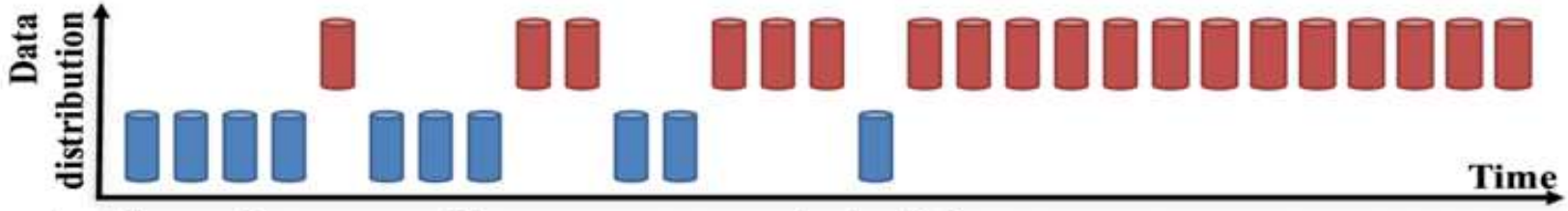
- Changes of the target variable occur for example in credit scoring, when the definition of the classification target "default" versus "non-default" changes due to business or regulatory requirements.
- Changes in the available feature information arise when new features become available, e.g. due to a new sensor or instrument. Similarly, existing features might need to be excluded due to regulatory requirements, or a feature might change in its scale, if data from a more precise instrument becomes available.
- Finally, drift is a phenomenon that occurs when the distributions of features x and target variables y change in time, e.g. sudden changes in the popularity of movie several days after its release due to good reviews from those who watched it and also due to changes in the price of the movie.

Types of drift



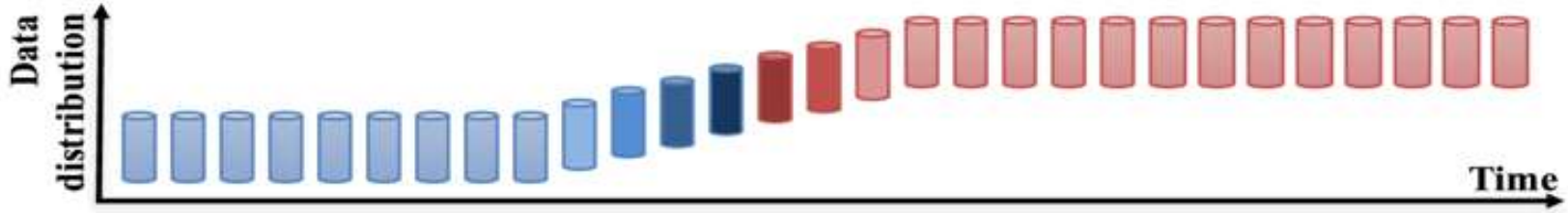
Sudden Drift:

A new concept occurs within a short time.



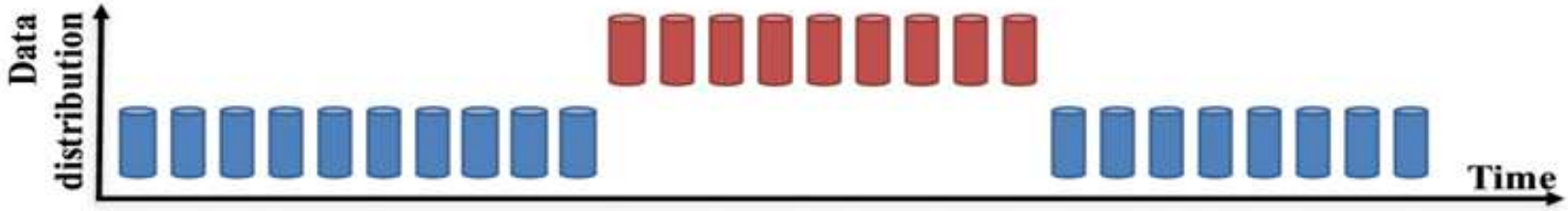
Gradual Drift:

A new concept gradually replaces an old one over a period of time.



Incremental Drift:

An old concept incrementally changes to a new concept over a period of time.



Reoccurring Drift:

An old concept may reoccur after some time.

Mining Big Data Stream cont…

Mining big data stream is challenging in the two aspects.

- First, random access to fast and large data streams may be impossible. Thus multi-pass algorithm (ones that load data into main memory multiple times) are often infeasible.
- Second, the exact answers from data streams are often too expensive.

The most common data stream mining tasks are:

- Clustering.
- Classification.
- Frequent pattern mining.

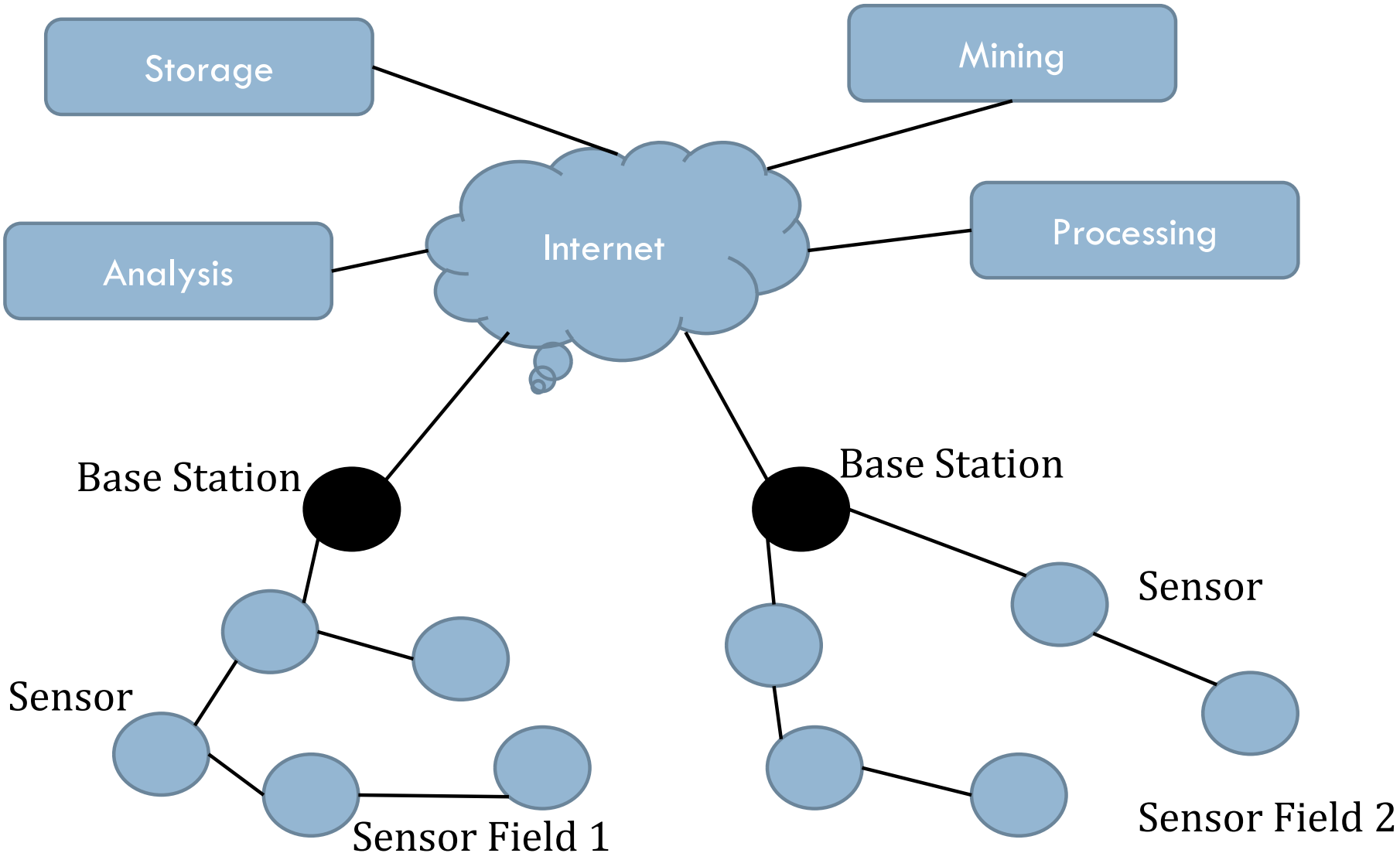
Examples of Data Stream Applications

- Sensor Networks
- Network Traffic Analysis
- Financial Applications
- Transaction Log Analysis

Sensor Networks

- Sensor networks (SNs) consist of spatially distributed devices communicating through wireless radio and cooperatively sensing physical or environmental conditions to provide a high degree of visibility into the environmental physical processes. These are notably useful in emergency scenarios, such as floods, fires, volcanoes, battlefields, where human participation is too dangerous and infrastructure networks are either impossible or too expensive.
- SNs are a huge source of data occurring in streams. They require constant monitoring of several variables, based on which important decisions are made.
- In many cases, alerts and alarms may be generated as a response to the information received from a series of sensors. To perform such analysis, aggregations and joins over multiple data stream corresponding to various sensors are required.
- Examples of queries involves:
  - Perform join of several data stream like temperature streams, ocean streams etc at weather stations to give alerts or warning of disasters like cyclones and tsunami.
  - Constant monitoring of a stream of power usage stastics to a power station, group them by locations, user types etc to manage power distribution efficiently.

Sensor networks



Network traffic analysis

Network traffic analysis (NTA) is a method of monitoring network availability and activity to identify anomalies, including security and operational issues. Common use cases for NTA include:

- Collecting a real-time and historical record of what's happening on the network.
- Detecting malware such as ransom ware activity.
- Detecting the use of vulnerable protocols and ciphers.
- Troubleshooting a slow network.
- Improving internal visibility and eliminating blind spots.

Implementing a data stream solution can continuously monitor network traffic gives the insight to optimize network performance, minimize attack surface, enhance security, and improve the management of resources. Examples queries include:

- Check whether a current stream of actions over a time window are similar to previous identified intrusions on the network.
- Check if several routes over which traffic is moving has several common intermediate nodes which may potential indicate a congestion on that route.

Financial Applications

Online analysis of stock prices and making hold or sell decisions requires quickly identification of correlations and fast changing trends and to an extent forecasting future valuations as data is constantly arriving from several sources like news, current stock movement etc. Typical queries include:

- Find the stocks priced between $1 and $200, which is showing very large buying in the last one hour based on some federal bank news about tax rates for a particular industry.
- Find all the stocks trading above their 100 day moving average by more than 10% and also with volume exceeding a million shares.

Transactional Log Analysis

Online mining of web usage logs, telephone call records and ATM are the examples of data streams since they continuously output data and are potentially infinite. The goal is to find interesting customer behavior patterns, identifying suspicious spending behavior that could indicate fraud etc. Typical queries include:

- Examine current buying pattern of users at a website and potentially plan advertising campaigns and recommendations.
- Continuously monitor location, average spends etc of credit card customers and identify potential frauds.

Stream Queries

- Streams Queries are similar to SQL in that one can specify which data like to include in the stream, any conditions that the data has to match, etc.
- Streams queries are composed in the following format: **SELECT** <select criteria> **WHERE** <where criteria> **HAVING** <having criteria>
- Two types of queries can be identified as typical over data streams. The first distinction is between one-time queries and continuous queries.
  - **One-time queries:** These are the queries that are evaluated once over a point-in-time snapshot of the dataset, with the answers returned to the users. For example, a stock price checker may alert the user when a stock price crosses a particular price point.
  - **Continuous queries:** These are evaluated continuously as data streams continue to arrive. The answer to a continuous query is produced over time, always reflecting the stream data so far. It may be stored and updated as new data arrives, or they may produced as data streams themselves. Typically, aggregation queries such as finding maximum, average, count etc. are the continuous queries where values are stored.

Stream Queries cont…

The second distinction is between predefined queries and ad hoc queries:

- **Predefined query:** A predefined query is one that is supplied to the data stream management system before any relevant data has arrived. Predefined queries are generally continuous queries, although scheduled one-time queries can also be predefined.
- **Ad hoc queries:** Ad hoc queries, are issued online after the data streams have already begun. Ad hoc queries can be either one-time queries or continuous queries. Ad hoc queries complicate the design of a data stream management system, both because they are not known in advance for the purposes of query optimization, identification of common sub expressions across queries, etc., and more importantly because the correct answer to an ad hoc query may require referencing data elements that have already arrived on the data streams.

Issues in data stream queries

Query processing in the data stream model comes with its own unique challenges and are:

- Unbounded Memory Requirements
- Approximate Query Answering
- Sliding Windows
- Batch Processing, Sampling and Synopses
- Blocking Operators

Unbounded Memory Requirements

- Data streams are potentially unbounded (i.e. it continually generated and can't process all at once) in size, and the amount of storage required to compute an exact answer to a data stream query may also grow without bound.
- The continuous data stream model is most applicable to problems where timely query responses are important and there are large volumes of data that are being continually produced at a high rate over time.
- New data is constantly arriving even as the old data is being processed; the amount of computation time per data element must be low, or else the latency of the computation will be too high and the algorithm will not be able to keep pace with the data stream.
- For this reason, the interest is in algorithms that are able to confine themselves to main memory without accessing disk.

Approximate Query Answering

- When we are limited to a bounded amount of memory it is not always possible to produce exact answers for data stream queries; however, high-quality approximate answers are often acceptable in lieu of exact answers.
- Approximation algorithms for problems defined over data streams has been a fruitful research area in the algorithms community in recent years, which has led to some general techniques for data reduction and synopsis construction, including: sketches, random sampling, histograms, and wavelets.

Note:

Data reduction: It is the transformation of numerical or alphabetical digital information derived empirically or experimentally into a corrected, ordered, and simplified form.

Synopsis construction: Creating synopsis of data refers to the process of applying summarization techniques that are capable of summarizing the incoming stream for further analysis. Since synopsis of data does not represent all the characteristics of the dataset, approximate answers are produced when using data synopsis. Synopses structures can be used both for answering queries and applying data mining algorithms to streams.

Sliding Windows

- One technique for producing an approximate answer to a data stream query is to evaluate the query not over the entire past history of the data streams, but rather only over sliding windows of recent data from the streams. For example, only data from the last week could be considered in producing query answers, with data older than one week being discarded.
- Imposing sliding windows on data streams is a natural method for approximation that has several attractive properties. It is well defined and easily understood: the semantics of the approximation are clear, so that users of the system can be confident that they understand what is given up in producing the approximate answer.
- Most importantly, it emphasizes recent data, which in the majority of real-world applications is more important and relevant than old data: if one is trying in real-time to make sense of network traffic patterns, or phone call or transaction records, or scientific sensor data, then in general insights based on the recent past will be more informative and useful than insights based on stale data.
- In fact, for many such applications, sliding windows can be thought of not as an approximation technique reluctantly imposed due to the infeasibility of computing over all historical data, but rather as part of the desired query semantics explicitly expressed as part of the user's query.

Sliding Windows Example

Consider the demo stream **STREAM_001** with the schema as:

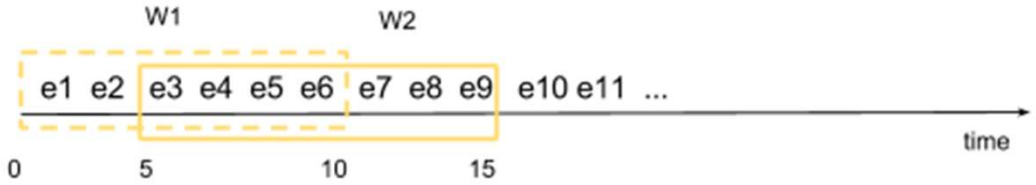
```
(TICKER_SYMBOL VARCHAR(4),
  SECTOR varchar(16),
  CHANGE REAL,
  PRICE REAL)
```

- Suppose a application to compute aggregates using a sliding 1-minute window i.e. for each new record that appears on the stream, the application to emit an output by applying aggregates on records in the preceding 1-minute window.
- The following time-based windowed query can be used. The query uses the WINDOW clause to define the 1-minute range interval. The PARTITION BY in the WINDOW clause groups records by ticker values within the sliding window.

```
 SELECT STREAM ticker_symbol,
     MIN(Price) OVER W1 AS Min_Price,
     MAX(Price) OVER W1 AS Max_Price,
     AVG(Price) OVER W1 AS Avg_Price
 FROM   “STREAM_001"
WINDOW W1 AS (
PARTITION BY ticker_symbol
RANGE INTERVAL '1' MINUTE PRECEDING);
```

Sliding Windows and Tumbling Windows

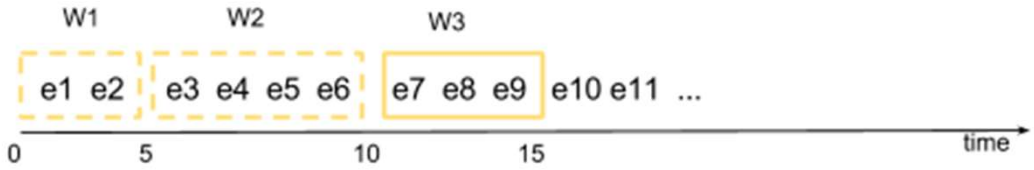
- In a sliding window, tuples are grouped within a window that slides across the data stream according to a specified interval. A time-based sliding window with a length of ten seconds and a sliding interval of five seconds contains tuples that arrive within a ten-second window. The set of tuples within the window are evaluated every five seconds. Sliding windows can contain overlapping data; an event can belong to more than one sliding window.
- In the following image, the first window (w1, in the box with dashed lines) contains events that arrived between the zero th and ten th seconds. The second window (w2, in the box with solid lines) contains events that arrived between the fifth and fifteenth seconds. Note that events e3 through e6 are in both windows. When window w2 is evaluated at time t = 15 seconds, events e1 and e2 are dropped from the event queue.



An example would be to compute the moving average of a stock price across the last five minutes, triggered every second.

Sliding Windows and Tumbling Windows cont…

- In a tumbling window, tuples are grouped in a single window based on time or count. A tuple belongs to only one window.
- For example, consider a time-based tumbling window with a length of five seconds. The first window (w1) contains events that arrived between the zero th and fifth seconds. The second window (w2) contains events that arrived between the fifth and tenth seconds, and the third window (w3) contains events that arrived between tenth and fifteenth seconds. The tumbling window is evaluated every five seconds, and none of the windows overlap; each segment represents a distinct time segment.



An example would be to compute the average price of a stock over the last five minutes, computed every five minutes.

Batch Processing, Sampling and Synopses

Another class of techniques for producing approximate answers is to give up on processing every data element as it arrives, resorting to some sort of sampling or batch processing technique to speed up query execution.

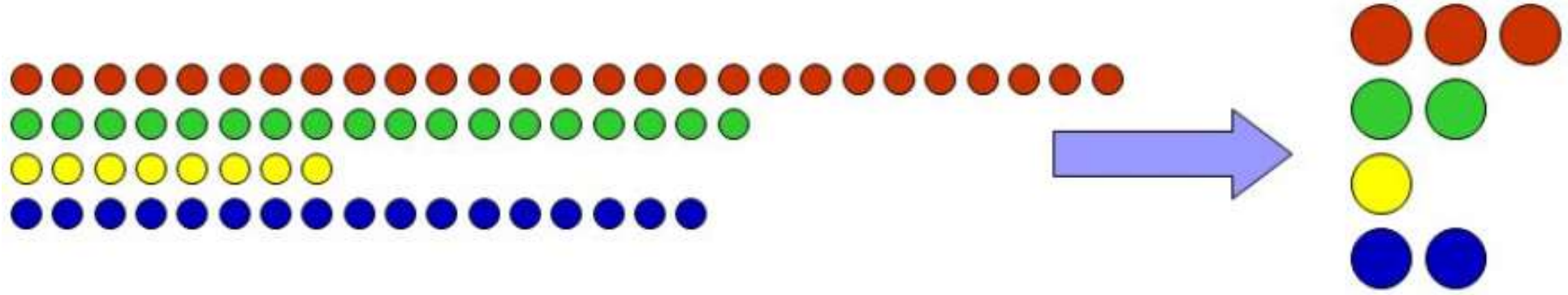
- In **batch processing**, rather than producing a continually up-to-date answer, the data elements are buffered as they arrive, and the answer to the query is computed periodically as time permits. The query answer may be considered approximate in the sense that it is not timely, i.e., it represents the exact answer at a point in the recent past rather than the exact answer at the present moment. This approach of approximation through batch processing is attractive because it does not cause any uncertainty about the accuracy of the answer, sacrificing timeliness instead. It is also a good approach when data streams are bursty.
- **Sampling** is based on the principle that it is futile to attempt to make use of all the data when computing an answer, because data arrives faster than it can be processed. Instead, some tuples must be skipped altogether, so that the query is evaluated over a sample of the data stream rather than over the entire data stream.
- For classes of data stream queries where no exact data structure with the desired properties exists, one can often design an approximate data structure that maintains a small **synopsis** or sketch of the data rather than an exact representation, and therefore is able to keep computation per data element to a minimum.

Blocking Operators

- A blocking query operator is a query operator that is unable to produce the first tuple of its output until it has seen its entire input. Sorting is an example of a blocking operator, as are aggregation operators such as SUM, COUNT, MIN, MAX, and AVG.
- If one thinks about evaluating continuous stream queries using a traditional tree of query operators, where data streams enter at the leaves and final query answers are produced at the root, then the incorporation of blocking operators into the query tree poses problems.
- Since continuous data streams may be infinite, a blocking operator that has a data stream as one of its inputs will never see its entire input, and therefore it will never be able to produce any output.
- Clearly, blocking operators are not very suitable to the data stream computation model, but aggregate queries are extremely common, and sorted data is easier to work with and can often be processed more efficiently than unsorted data.
- Doing away with blocking operators altogether would be problematic, but dealing with them effectively is one of the more challenging aspects of data stream computation.

Sampling in Data Streams

Sampling is a common practice for selecting a subset of data to be analyzed. Instead of dealing with an entire data stream, the instances at periodic intervals are selected. Sampling is used to compute statistics (expected values) of the stream. While sampling methods reduce the amount of data to process, and, by consequence, the computational costs, they can also be a source of errors. The main problem is to obtain a representative sample, a subset of data that has approximately the same properties of the original data.



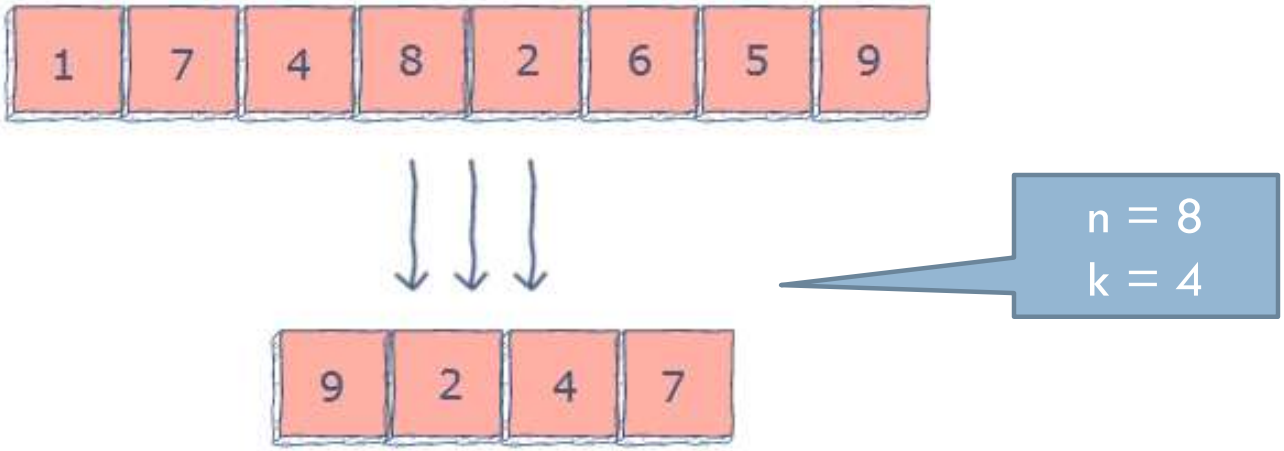
Need & how of sampling - System cannot store the entire stream conveniently, so

- how to make critical calculations about the stream using a limited amount of (primary or secondary) memory?
- Don't know how long the stream is, so when and how often to sample?

Three solutions namely (i) Reservoir , (ii) Biased Reservoir , and (iii) Concise sampling

Reservoir Sampling

Reservoir sampling is a randomized algorithm that is used to select k out of n samples; n is usually very large or unknown. For example, it can be used to obtain a sample of size k from a population of people with brown hair. This algorithm takes O(n) to select k elements with uniform probability.



The key idea behind reservoir sampling is to create a 'reservoir' from a big ocean of data. Each element of the population has an equal probability of being present in the sample and that probability is (k/n). With this key idea, a subsample to be created. It has to be noted, when a sample is created, the distributions should be identical not only row-wise but also column-wise, wherein columns are the features.

Reservoir Sampling Algorithm

0. Start
1. Create an array reservoir[0..k-1] and copy first k items of stream[] to it.
2. Iterate from k to n−1. In each iteration i:
   2.1. Generate a random number from 0 to i. Let the generated random number is j.
   2.2. If j is in range 0 to k-1, replace reservoir[j] with arr[i]
3. Stop

Illustration

Input:
The list of integer stream: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}, and the value of k = 6
Output:
k-selected items in the given array: 8 2 7 9 12 6

Biased Reservoir Sampling

In many cases, the stream data may evolve over time, and the corresponding data mining or query results may also change over time. Thus, the results of a query over a more recent window may be quite different from the results of a query over a more distant window. Similarly, the entire history of the data stream may not relevant for use in a repetitive data mining application such as classification. The simple reservoir sampling algorithm can be adapted to a sample from a moving window over data streams. This is useful in many data stream applications where a small amount of recent history is more relevant than the entire previous stream. However, this can sometimes be an extreme solution, since for some applications we may need to sample from varying lengths of the stream history. While recent queries may be more frequent, it is also not possible to completely disregard queries over more distant horizons in the data stream. **Biased reservoir sampling** is a bias function to regulate the sampling from the stream. This bias gives a higher probability of selecting data points from recent parts of the stream as compared to distant past. This bias function is quite effective since it regulates the sampling in a smooth way so that the queries over recent horizons are more accurately resolved.

Concise sampling

Many a time, the size of the reservoir is sometimes restricted by the available main memory. It is desirable to increase the sample size within the available main memory restrictions. For this purpose, the technique of concise sampling is quite effective. Concise sampling exploits the fact that the number of distinct values of an attribute is often significantly smaller than the size of the data stream. In many applications, sampling is performed based on a single attribute in multi-dimensional data. For example, customer data in an e-commerce site sampling may be done based on only customer ids. The number of distinct customer ids is definitely much smaller than “n” the size of the entire stream.

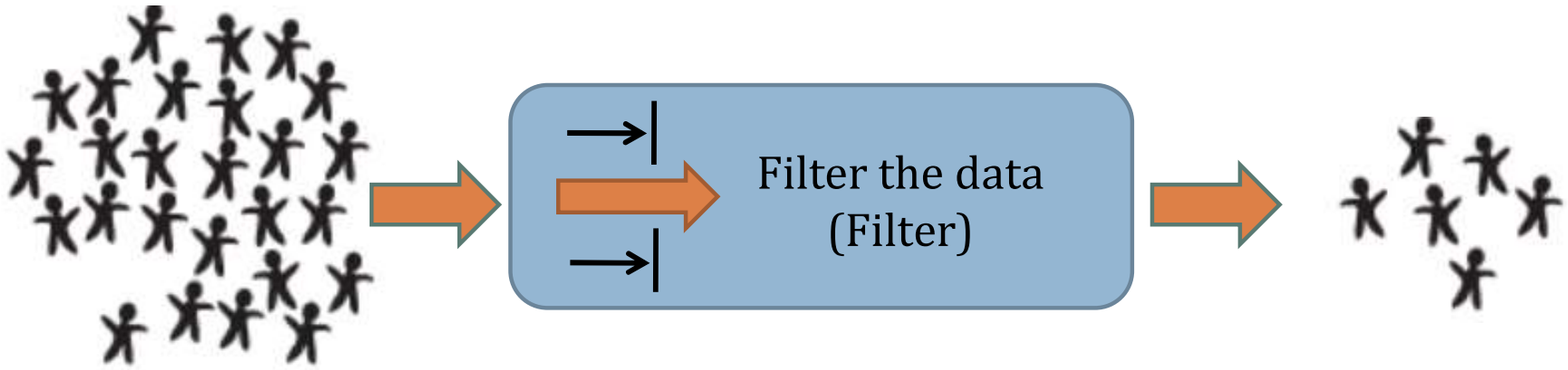
The repeated occurrence of the same value can be exploited in order to increase the sample size beyond the relevant space restrictions. We note that when the number of distinct values in the stream is smaller than the main memory limitations, the entire stream can be maintained in main memory, and therefore, sampling may not even be necessary. For current systems in which the memory sizes maybe the order of several gigabytes, very large sample sizes can be main memory resident as long as the number of distinct values do not exceed the memory constraints.

Other Sampling Techniques

- Inverse Sampling
- Weighted Sampling
- Biased Sampling
- Priority Sampling
- Dynamic Sampling
- Chain Sampling

Filtering Stream

A filter is a is a program or section of code that is designed to examine each input or output stream for certain qualifying criteria and then process or forward it accordingly by producing another stream.



All stock transaction
(Input)

KIIT stock transaction
(Output)

In this example, the streams processing application needs to filter the stock transaction data for KIIT transaction records.

Bloom Filter

Bloom filter is a space-efficient probabilistic data structure conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. **False positive** matches are possible, but **False Negatives are not –** in other words, a query returns either "**possibly in set**" or "**definitely not in set**"

False Positive = "**possibly in set**" or "**definitely not in set**"

False Negative = "**possibly not in set**" or "**definitely in set**"

Overview

x : An element

S: A set of elements

Input: x, S Output:

-TRUE if x in S

-FALSE if x not in S

Bloom Filter cont…

Suppose you are creating an account on Facebook, you want to enter a cool username, you entered it and got a message, “Username is already taken”. You added your birth date along username, still no luck. Now you have added your university roll number also, still got “Username is already taken”. It’s really frustrating, isn’t it?

But have you ever thought how quickly Facebook check availability of username by searching millions of username registered with it. There are many ways to do this job –

Linear search : Bad idea!

Binary Search : There must be something better!!

Bloom Filter is a data structure that can do this job.

Bloom Filter cont…

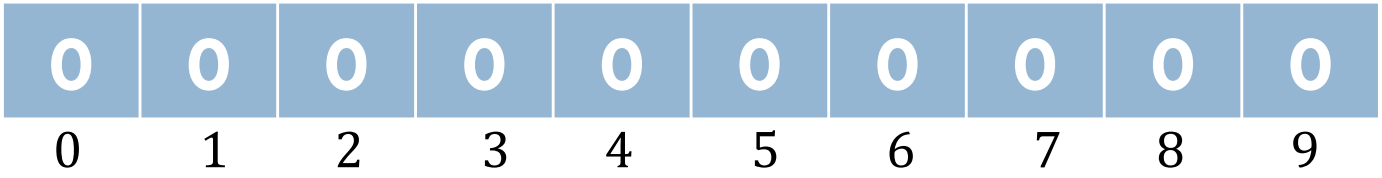
For understanding bloom filters, one must know hashing. A hash function takes input and outputs a unique identifier of fixed length which is used for identification of input.

Properties of Bloom Filter

- Unlike a standard hash table, a Bloom filter of a fixed size can represent a set with an arbitrarily large number of elements.
- Adding an element never fails. However, the false positive rate increases steadily as elements are added until all bits in the filter are set to 1, at which point all queries yield a positive result.
- Bloom filters never generate false negative result, i.e., telling you that a username doesn't exist when it actually exists.
- Deleting elements from filter is not possible because, if we delete a single element by clearing bits at indices generated by k hash functions, it might cause deletion of few other elements.

Working of Bloom Filter

A empty bloom filter is a bit array of n bits, all set to zero, like below:



We need k number of hash functions to calculate the hashes for a given input. When we want to add an item in the filter, the bits at k indices $h_1(x)$, $h_2(x)$, ... $h_k(x)$ are set, where indices are calculated using hash functions.

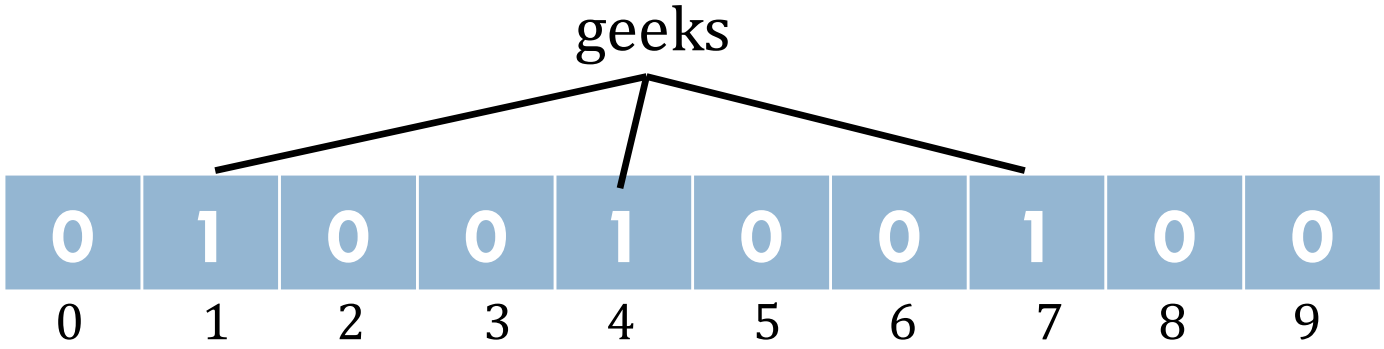
Example – Suppose we want to enter "geeks" in the filter, we are using 3 hash functions and a bit array of length 10, all set to 0 initially. First we'll calculate the hashes as following :

h_1("geeks") % 10 = 1, h_2("geeks") % 10 = 4, and h_3("geeks") % 10 = 7

Note: *These outputs are random for explanation only.*

Working of Bloom Filter cont...

Now we will set the bits at indices 1, 4 and 7 to 1



Again we want to enter "nerd", similarly we'll calculate hashes

h_1("nerd") % 10 = 3

h_2("nerd") % 10 = 5

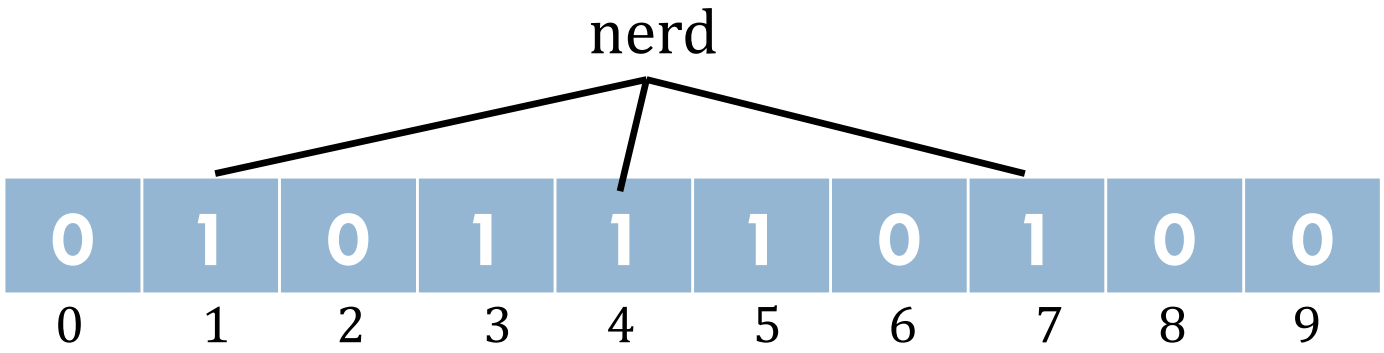
h_3("nerd") % 10 = 4

Note: *These outputs are random for explanation only.*

Set the bits at indices 3, 5 and 4 to 1

Working of Bloom Filter cont...

Now we will set the bits at indices 3, 5 and 4 to 1



Now if we want to check “geeks” is present in filter or not. We’ll do the same process but this time in reverse order. We calculate respective hashes using h_1, h_2 and h_3 and check if all these indices are set to 1 in the bit array. If all the bits are set then we can say that “geeks” is probably present. If any of the bit at these indices are 0 then “geeks” is definitely not present.

False Positive in Bloom Filters

The question is why we said "probably present", why this uncertainty. Let's understand this with an example. Suppose we want to check whether "cat" is present or not. We'll calculate hashes using h_1, h_2 and h_3

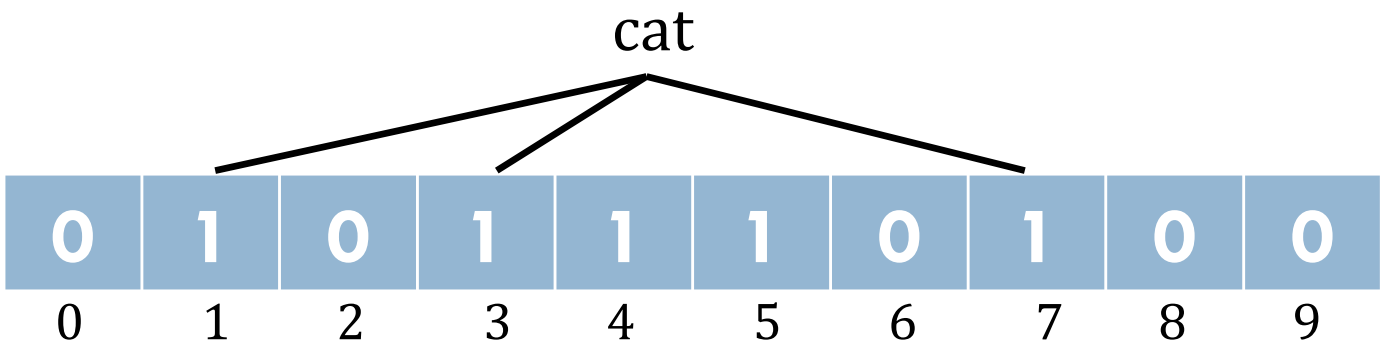
h_1("cat") % 10 = 1

h_2("cat") % 10 = 3

h_3("cat") % 10 = 7

If we check the bit array, bits at these indices are set to 1 but we know that "cat" was never added to the filter. Bit at index 1 and 7 was set when we added "geeks" and bit 3 was set we added "nerd".

False Positive in Bloom Filters cont...



So, because bits at calculated indices are already set by some other item, bloom filter erroneously claim that "cat" is present and generating a false positive result. Depending on the application, it could be huge downside or relatively okay.

We can control the probability of getting a false positive by controlling the size of the Bloom filter. More space means fewer false positives. If we want decrease probability of false positive result, we have to use more number of hash functions and larger bit array. This would add latency in addition of item and checking membership.

Bloom Filter Algorithm

Insertion

Data: e is the element to insert into the Bloom filter.

```
insert(e)
begin
 /* Loop all hash functions k */
 for j : 1 . . . k do
  m ← h_j(e) //apply the hash function on e
  Bm ← bf[m] //retrive val at mth pos from Bloom filter bf
  if Bm == 0 then
  /* Bloom filter had zero bit at index m */
   Bm ← 1;
  end if
 end for
end
```

Lookup

Data: x is the element for which membership is tested.

```
bool isMember(x) /* returns true or false to the membership test */
begin
 t ← 1
 j ← 1
  while t == 1 and j ≤ k do
   m ← hj(x)
   Bm ← bf[m]
    if Bm == 0 then
     t ← 0
    end
    j ← j + 1;
   end while
 return (bool) t
end
```

Bloom Filter Performance

A Bloom filter requires space O(n) and can answer membership queries in O(1) time where n is number item inserted in the filter. Although the asymptotic space complexity of a Bloom filter is the same as a hash map, O(n), a Bloom filter is more space efficient.

Class Exercise

- A empty bloom filter is of size 11 with 4 hash functions namely
  - $h_1(x) = (3x+ 3) \bmod 6$
  - $h_2(x) = (2x+ 9) \bmod 2$
  - $h_3(x) = (3x+ 7) \bmod 8$
  - $h_4(x) = (2x+ 3) \bmod 5$

  Illustrate bloom filter insertion with 7 and then 8.

  Perform bloom filter lookup/membership test with 10 and 48

False Positive in Bloom Filters cont…

0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11

K=4 (# of Hash Function)

Note: *These outputs are random for explanation only.*

INSERT (x_1), $x_1=7$

$h_1(x_1) = 0$

$h_2(x_1) = 1$

$h_3(x_1) = 4$

$h_4(x_1) = 2$

INSERT (x_2), $x_2 = 8$

$h_1(x_2) = 3$

$h_2(x_2) = 1$

$h_3(x_2) = 7$

$h_4(x_2) = 4$

State of bloom filter post to the insertion of x_1 and x_2

1	1	1	1	1	0	0	1	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11

False Positive in Bloom Filters cont'd

LOOKUP(x_3), x_3 = 10
$h_1(x_3) = 3$
$h_2(x_3) = 1$
$h_3(x_3) = 5$

X_3 doesn't exist

LOOKUP(x_4) , x_4 = 48
$h_1(x_4) = 3$
$h_2(x_4) = 1$
$h_3(x_4) = 7$
$h_4(x_4) = 4$

X_4 - Case of FALSE POSITIVE

Optimum number of hash functions

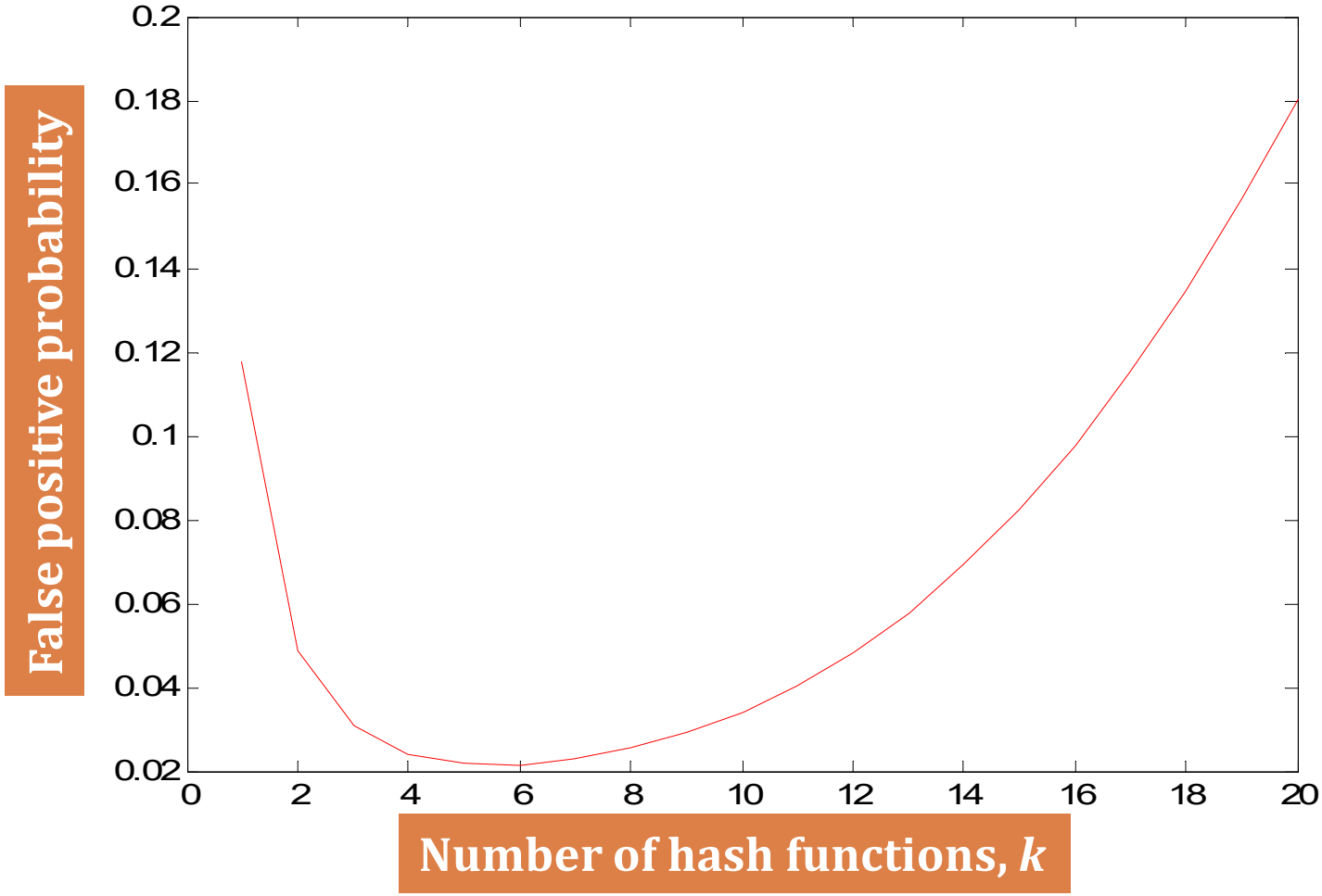
The number of hash functions k must be a positive integer. If n is size of bit array and m is number of elements to be inserted, then k can be calculated as :

$$k = \frac{n}{m} \ln(2)$$

Class Exercise

Calculate the optimal number of hash functions for 10 bit length bloom filter having 3 numbers of input elements.

What happens to increasing k?



Calculating probability of False Positives

- Probability that a slot is hashed = 1/n
- Probability that a slot is not hashed = 1 – (1/n)
- Probability that a slot is not hashed after insertion of an element for all the k hash function is:

$$\left(1 - \frac{1}{n}\right)^{k}$$

- Probability that a slot is not set to 1 after insertion of m element is:

$$\left(1 - \frac{1}{n}\right)^{km}$$

- Probability that a slot is set to 1 after insertion of m element is:

$$1 - \left(1 - \frac{1}{n}\right)^{km}$$

Calculating probability of False Positives cont'd

Let n be the size of bit array, k be the number of hash functions and m be the number of expected elements to be inserted in the filter, then the probability of false positive p can be calculated as:

$$\left(1-\left(\frac{1}{e}\right)^{\frac{km}{n}}\right)^{k}$$

Class Exercise

Calculate the probability of false positives with table size 10 and no. of items to be inserted are 3.

Counting distinct elements in a stream – Approach 1

(1, 2, 2, 1, 3, 1, 5, 1, 3, 3, 3, 2, 2)
Number of distinct elements = 4

How to calculate?

1. Initialize the hashtable (large binary array) of size n with all zeros.
2. Choose the hash function h_i : i ∈ {1, ..., k}
3. For each flow label f ∈ {1, ..., m} , compute h(f) and mark that position in the hashtable with 1
4. Count the number of positions in the hashtable with 1 and call it c.
5. The number of distinct items is m* ln (m / (m-c))

Class Exercise

Count the distinct elements in a data stream of elements {1, 2, 2, 1, 3, 1, 5, 1, 3, 3, 3, 2, 2} with the hash function h(x) = (5x+1) mod 6 of size 11.

Counting distinct elements in a stream using Flajolet-Martin algorithm – Approach 2

Count the distinct elements in a data stream of elements {6,8,4,6,3,4} with hash function h(x) = (5x+1) mod 6 of size 11.

How to calculate?

Step 1:

Apply Hash function(s) to the data stream and compute the slots.

h(6)=1, h(8)=5, h(4)=3, h(6)=1, h(3)=4, h(4)=3.

The slot numbers obtained are:{1,5,3,1,4,3}

Step 2: Convert the numbers to binary

h(6)=1=001, h(8)=5=101, h(4)=3=001, h(6)=1=001, h(3)=4=100, h(4)=3=011

Step 3: Calculate the maximum trailing zeros

TZ ={0, 0, 0, 0, 2, 0} /* TZ stands for Trailing Zeros */

R = MAX(TZ) = MAX(0, 0, 0, 0, 2, 0) = 2

Step 4: Estimate the distinct elements with the formula 2^R

Number of distinct elements = $2^R = 2^2 = 4$

Bloom Filter Use Cases

- Bitcoin uses Bloom filters to speed up wallet synchronization and also to improve Bitcoin wallet security
- Google Chrome uses the Bloom filter to identify malicious URLs - it keeps a local Bloom filter as the first check for Spam URL
- Google BigTable and Apache Cassandra use Bloom filters to reduce disk lookups for non-existent rows or columns
- The Squid Web Proxy Cache uses Bloom filters for cache digests - proxies periodically exchange Bloom filters for avoiding ICP messages
- Genomics community uses Bloom filter for classification of DNA sequences and efficient counting of k-mers in DNA sequences
- Used for preventing weak password choices using a dictionary of easily guessable passwords as bloom filter
- Used to implement spell checker using a predefined dictionary with large number of words

Other Types of Bloom Filter

- **Compressed Bloom Filter** - Using a larger but sparser Bloom Filter can yield the same false positive rate with a smaller number of transmitted bits.
- **Scalable Bloom Filter** - A Scalable Bloom Filters consist of two or more Standard Bloom Filters, allowing arbitrary growth of the set being represented.
- **Generalized Bloom Filter** - Generalized Bloom Filter uses hash functions that can set as well as reset bits.
- **Stable Bloom Filter** - This variant of Bloom Filter is particularly useful in data streaming applications.

Estimating Moments

- Assume a stream A of length N which is composed of m different types of items $a_1,...,a_m$ each of which repeats itself $n_1,...,n_m$ times (in arbitrary order). The frequency moments of order k, f_k is defined as

$$f_k = \sum_{i=1}^{m} n_i^k$$

- The 0^{th} order moment i.e. f_0 is the number of distinct elements in the stream.
- The 1^{st} order moment i.e. f_1 is the length of the stream.
- The 2^{nd} order moment f_2 is an important quantity which represent show "skewed" the distribution of the elements in stream is.

Example

Consider the stream a, b, c, b, d, a, c, d, a, b, d, c, a, a, b wherein $n_a = 5$, $n_b = 4$, $n_c = 3$ and $n_d = 3$. In this case:

- f_0 = number of distinct elements = 4
- f_1 = length of the stream = $n_a + n_b + n_c + n_d = 5 + 4 + 3 + 3 = 15$
- $f_2 = n_a^2 + n_b^2 + n_c^2 + n_d^2 = 5^2 + 4^2 + 3^2 + 3^2 = 59$

Real-Time Analytics Platform (RTAP)

- Real-time analytics makes use of all available data and resources when they are needed, and it consists of dynamic analysis and reporting, based on data entered into a system before the actual time of use.
- Real-time denotes the ability to process data as it arrives, rather than storing the data and retrieving it at some point in the future.
- For example, consider an e-merchant like Flipkart or Snapdeal; real time means the time elapsed from the time a customer enters the website to the time the customer logs out. Any analytics procedure, like providing the customer with recommendations or offering a discount based on current value in the shopping car, etc., will have to be done within this timeframe which may be a about 15 minutes to an hour.
- But from the point of view of a military application where there is constant monitoring say of the air space, time needed to analyze a potential threat pattern and make decision maybe a few milliseconds.

RTAP cont…

Real-Time Analytics is thus discovering meaningful patterns in data for something urgent. There are two specific and useful types of real-time analytics i.e. On-Demand and Continuous.

- **On-Demand Real-Time Analytics** is reactive because it waits for users to request a query and then delivers the analytics. This is used when someone within a company needs to take a pulse on what is happening right this minute. For instance, a movie producer may want to monitor the tweets and identify sentiments about his movie on the first day first show and be prepared for the outcome.
- **Continuous Real-Time Analytics** is more proactive and alerts users with continuous updates in real time. The best example could be monitoring the stock market trends and provide analytics to help users make a decision to buy or sell all in real time.


THANK
YOU!