Mining of data stream

Examples of Stream Sources

1 Sensor Data
To learn something about ocean behavior, we might want to deploy a million sensors, each sending back a stream, at the rate of ten per second.

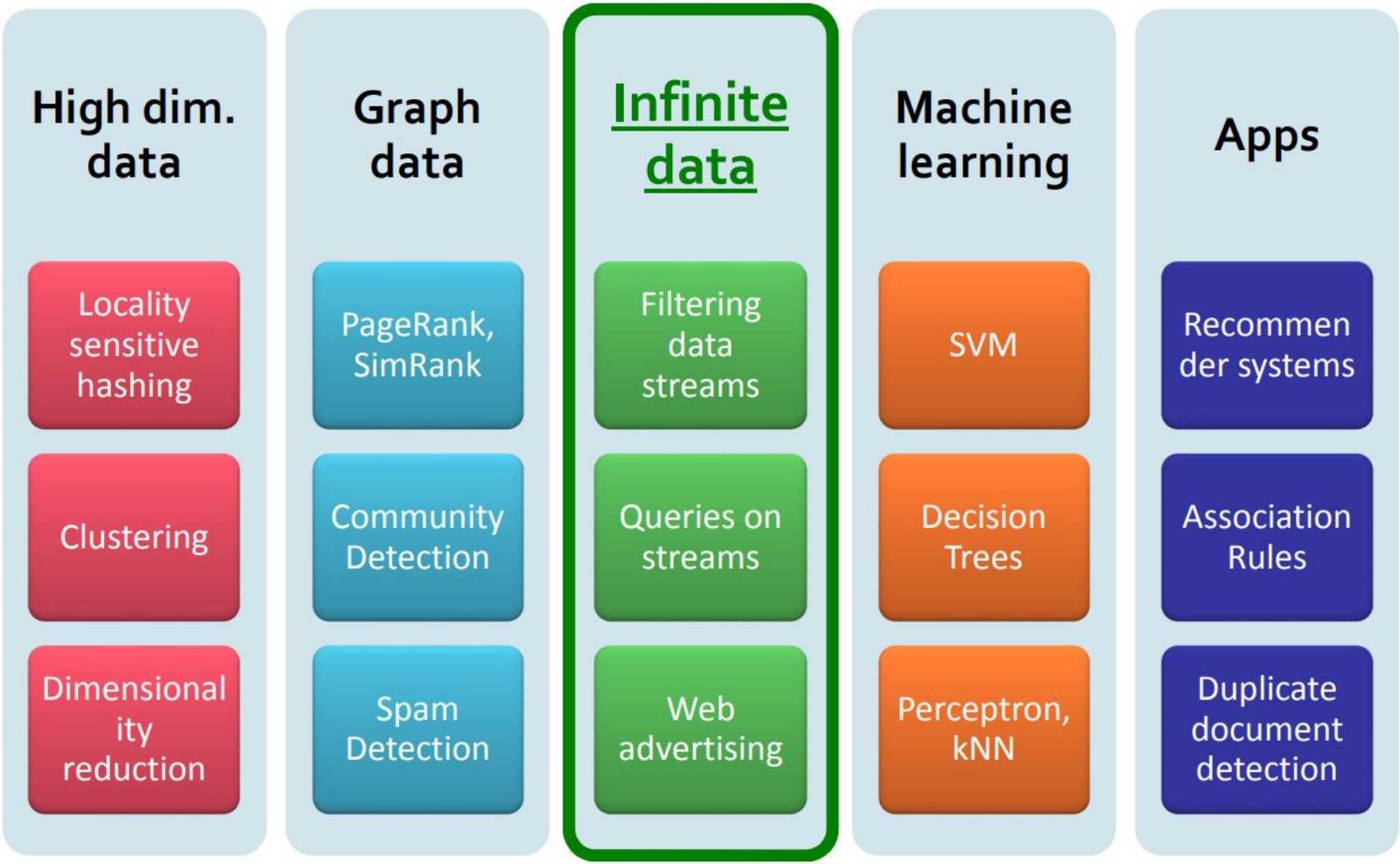
2 Image Data
Satellites often send down to earth streams consisting of many terabytes of images per day.

.

Examples of Stream Sources

3 Internet and Web Traffic

A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs.
Google receives several hundred million search queries per day. Yahoo! accepts billions of “clicks” per day on its various sites. Many interesting things can be learned from these streams. For example, an increase in queries like “sore throat” enables us to track the spread of viruses. A sudden increase in the click rate for a link could indicate some news connected to that page, or it could mean that the link is broken and needs to be repaired


High dim. data
Locality sensitive hashing
Clustering
Dimensionality reduction
Graph data
PageRank, SimRank
Community Detection
Spam Detection
Infinite data
Filtering data streams
Queries on streams
Web advertising
Machine learning
SVM
Decision Trees
Perceptron, kNN
Apps
Recommender systems
Association Rules
Duplicate document detection

Data Streams

- **In many data mining situations, we do not know the entire data set in advance**

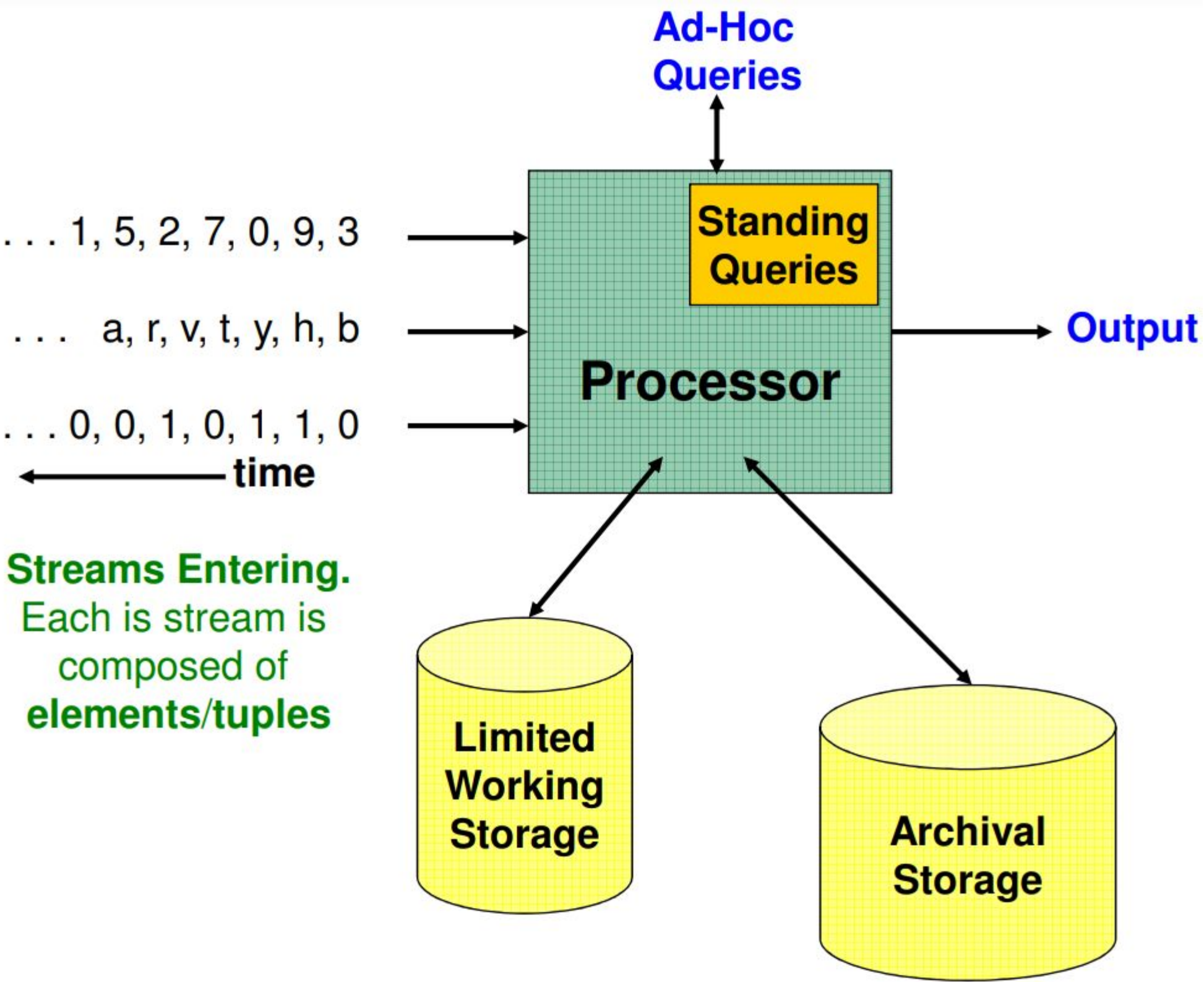
- **Stream Management** is important when the input rate is controlled **externally:**
  - Google queries
  - Twitter or Facebook status updates
- We can think of the **data** as **infinite** and **non-stationary** (the distribution changes over time)

Stream Model

- Input **elements** enter at a rapid rate, at one or more input ports (i.e., **streams**)
  - **We call elements of the stream tuples**
- **The system cannot store the entire stream accessibly**
- **Q: How do you make critical calculations about the stream using a limited amount of (secondary) memory?**

Problems of Data Stream

- **Sampling data from a stream**
  - Construct a random sample
- **Queries over sliding windows**
  - Number of items of type ***x*** in the last ***k*** elements of the stream



Streams Entering.
Each is stream is composed of **elements/tuples**

Stream Processor

Any number of streams can enter the system.

Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform.

The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system.

Limited storage vs. archival storage

Streams may be archived in a large archival store, but it is not possible to answer queries from the archival store.
It requires time-consuming retrieval processes.

There is also a working store, into which summaries or parts of streams may be placed, and which can be used for answering queries.
The working store might be disk, or it might be main memory, depending on how fast we need to process queries.
But either way, it is of sufficiently limited capacity that it cannot store all the data from all the streams.

standing queries

- Standing queries are placed within the processor. These queries are permanently executing, and produce outputs at appropriate times.
- Example are
- The stream produced by the ocean-surface-temperature sensor might have a standing query to output an alert whenever the temperature exceeds 25 degrees centigrade.
- The maximum temperature ever recorded by that sensor

Ad-hoc queries

A question asked once about the current state of a stream or streams.
If we do not store all streams in their entirety, as normally we can not, then we cannot expect to answer arbitrary queries about streams.
A common approach is to store a sliding window. sliding window can be the most recent n elements of a stream, for some n ,or it can be all the elements that arrived within the last t time units
The stream -management system must keep the window fresh, deleting the oldest elements as new ones come in.

Issues in Stream Processing

- Streams often deliver elements very rapidly.
- We must process elements in real time, or we lose the opportunity to process them at all.
- Requires very large main memory
- Requires the invention of new techniques in order to execute them at a realistic rate on a machine of realistic size.
- Generalizations about stream algorithms worth bearing in mind, it is much more efficient to get an approximate answer to our problem than an exact solution

Data Management Vs. Stream Management

- In a DBMS, input is under the control of the programming staff.
  - SQL INSERT commands or bulk loaders.
- Stream Management is important when the input rate is controlled externally.
  - Example: Google search queries.

Sampling Data in a Stream

Selecting a subset of a stream so that we can ask queries about the selected subset and have the answers be statistically representative of the stream as a whole. Search engine receives a stream of queries, and it would like to study the behavior of typical users. Let us assume the stream consists of tuples (user,query, time). We wish to store only 1/10th of the stream elements.
We use the hash function as a random-number generator,with the important property that, when applied to the same user several times, we always get the same “random” number.

When Sampling Doesn't Work

- Suppose Google would like to examine its stream of search queries for the past month to find out what fraction of them were unique – asked only once.
- But to save time, we are only going to sample $1/10^{th}$ of the stream.
- The fraction of unique queries in the sample will not be the fraction for the stream as a whole.
  - In fact, we can't even adjust the sample's fraction to give the correct answer.

Example: Salary Ranges

- Data = tuples of the form (EmpID, Dept, Salary).
- Query: What is the average range of salaries within a department?
- Key = Dept.
- Value = (EmpID, Salary).
- Sample picks some departments, has salaries for all employees of that department, including its min and max salaries.

We want to work with let us say 10% of those tuples. Picking randomly any tuple will not work. For a given department we are likely to miss both of the employees with the minimum or maximum salary in that department. So we are biased because of poor sampling strategy. The right way to sample is to treat only the department component of the tuple as a key and employee ID, salary as a value.

Types of queries one wants on answer on a data stream:

- **Filtering a data stream**
  - Select elements with property ***x*** from the stream
- **Counting distinct elements**
  - Number of distinct elements in the last ***k*** elements of the stream
- **Estimating moments**
  - Estimate avg./std. dev. of last ***k*** elements
- **Finding frequent elements**

- **Mining query streams**
  - Google wants to know what queries are more frequent today than yesterday
- **Mining click streams**
  - Yahoo wants to know which of its pages are getting an unusual number of hits in the past hour
- **Mining social network news feeds**
  - E.g., look for trending topics on Twitter, Facebook

Flajolet Martin Algorithm

To count distinct elements in a stream

Determine the distinct element in the stream using FM.

- **I**nput stream of integers x = 1,3,2,1,2,3,4,3,1,2,3,1 .

Hash Function, $h(x) = 6x + 1 \bmod 5$

For the given input stream 1,3,2,1,2,3,4,3,1,2,3,1

$h(x) = 6x + 1 \mod 5$

$h(1) = 2$	$h(4) = 0$
$h(3) = 4$	$h(3) = 4$
$h(2) = 3$	$h(1) = 2$
$h(1) = 2$	$h(2) = 3$
$h(2) = 3$	$h(3) = 3$
$h(3) = 4$	$h(1) = 2$

- For the given input stream 1,3,2,1,2,3,4,3,1,2,3,1.
- For every hash function calculated, write the binary equivalent for the same.

h(1) = 2 = 010	h(4) = 0 = 000
h(3) = 4 = 100	h(3) = 4 = 100
h(2) = 3 = 011	h(1) = 2 = 010
h(1) = 2 = 010	h(2) = 3 = 011
h(2) = 3 = 011	h(3) = 4 = 100
h(3) = 4 = 100	h(1) = 2 = 010

- From the binary equivalent trailing zero values, write the value of maximum number of trailing zeros.
- The value of $r = 2$
- The distinct value $R = 2^r$
- Therefore $R = 2^2 = 4$
- Hence , there are 4 distinct elements as 1,3,2,4