

Data Streaming

Raja CHIKY
raja.chiky@isep.fr
2014-2015





OUTLINE

- **Context: Big Data**
 - What is a data stream ?**
 - Data stream management systems**
 - Basic approximate algorithms**
 - Big Data: Distributed Systems**
 - Semantic Data Streaming**
 - Conclusion**



New era



Where is all this data coming from?

Networked Connection of People, Process, Data, Things



People

Connecting people in more relevant, valuable ways



Process

Delivering the right information to the right person (or machine) at the right time



Data

Leveraging data into more useful information for decision making



Things

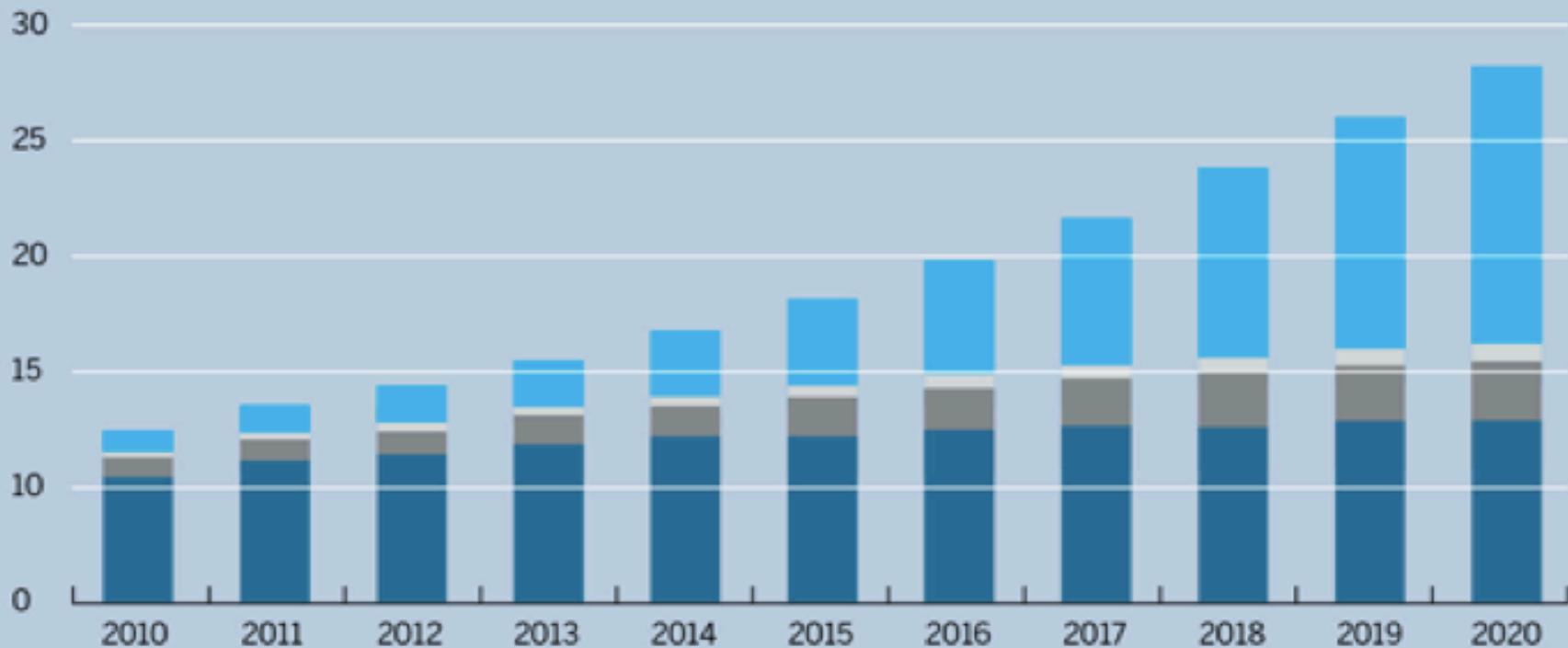
Physical devices and objects connected to the Internet and each other for intelligent decision making

More and More connected Things

Machines Go Online

The number of everyday objects, or “things,” connecting to the Internet will exceed PCs and smartphones.

Connected devices (billions)



MIT Technology Review

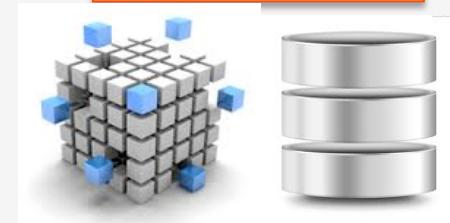
C



Static data



Ad-hoc queries Analytics



Data Warehouse

ETL Batch processing



databases

Output

User
Interaction

Store

Gathering
Information

Data
sources

Big Data : Velocity

Website logs



Network monitoring



Financial services



Weather forecasting



Too much data streams

eCommerce



Traffic control



Power consumption



What is a data stream?

- Golab & Oszu (2003): “*A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety.*”
- Massive volumes of data, items arrive at a high rate.



Applications of data stream processing

- **Data stream processing**

- **Data stream processing**
 - Process queries (compute statistics, activate alarms)
 - Apply data mining algorithms

- **Requirements**

- **Requirements**
 - Real-time processing
 - One-pass processing
 - Bounded storage (no complete storage of streams)
 - Possibly consider several streams

Applications of data stream processing

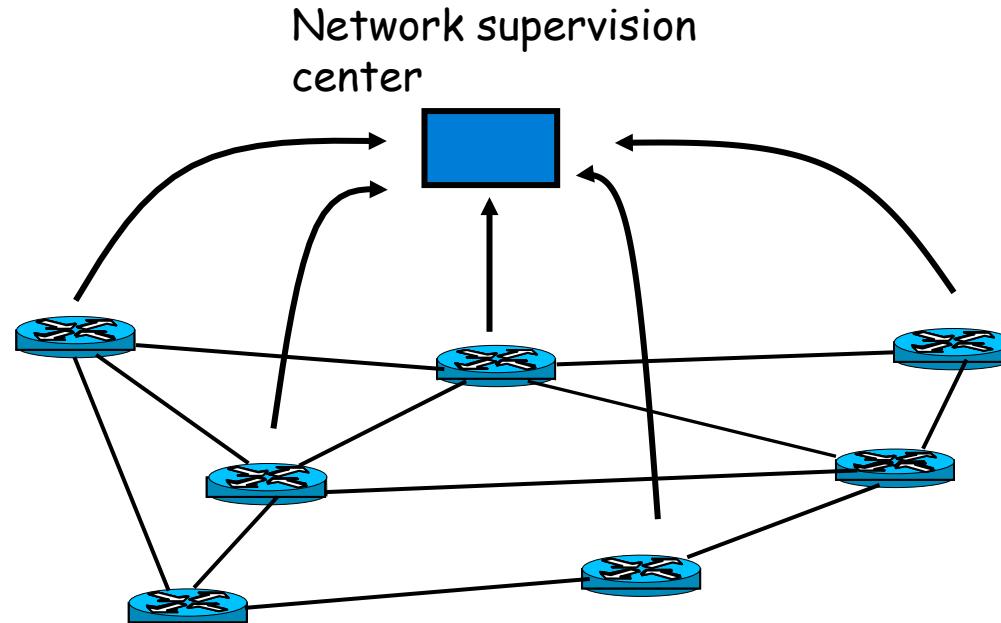
- Let's go deeper into some examples
 - Network management
 - Stock monitoring

Network management

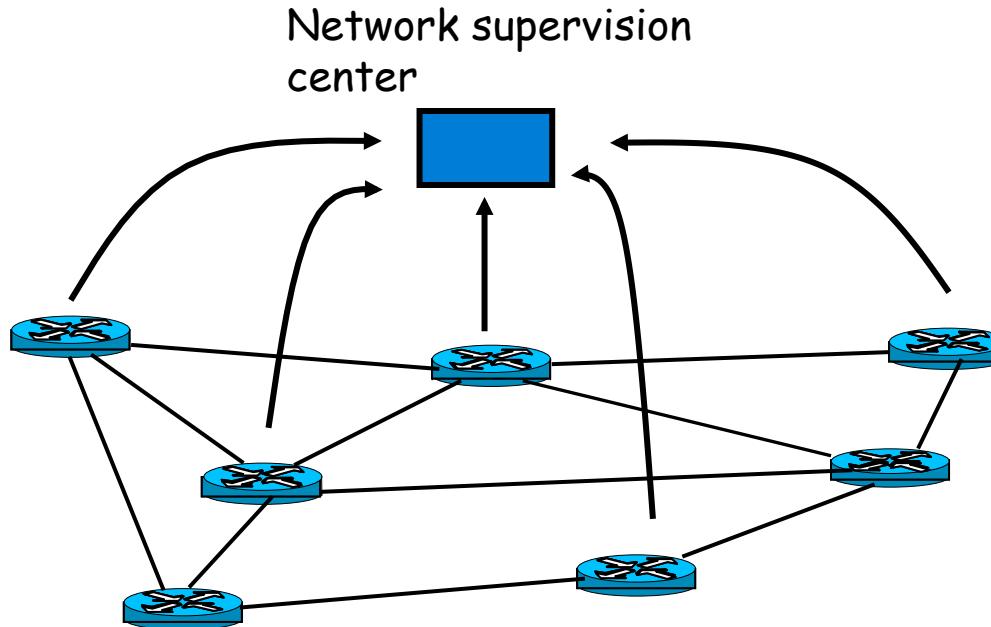
- **Supervision of a computer network**
- **Improvement of network configuration (hardware, software, architecture)**
- **Detection of attacks**
- **Measurements made on routers**

Huge volume of data

High rate of arrivals

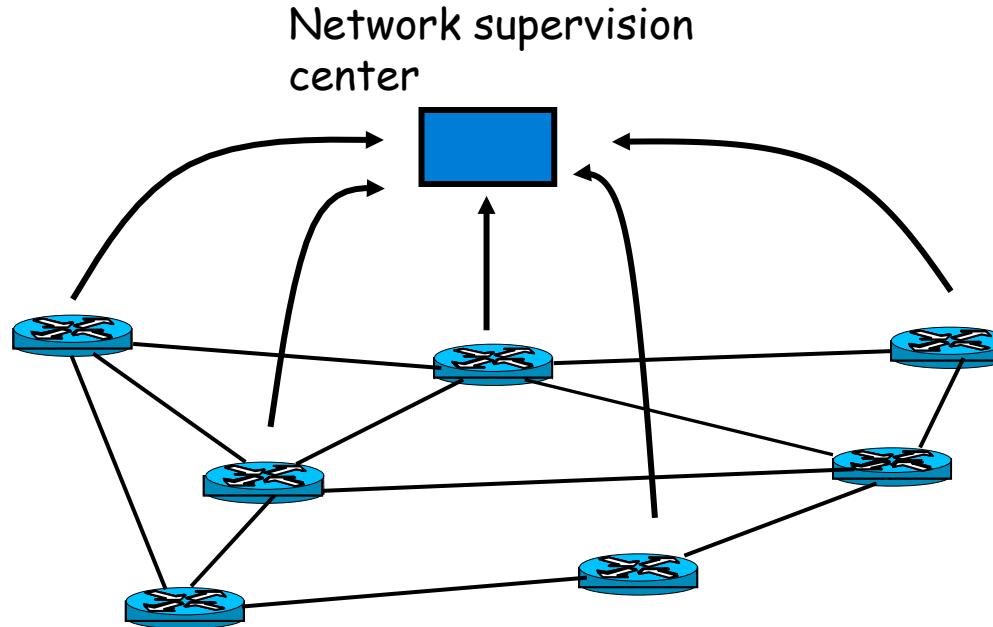


Network management



| Timestamp | Source | Destination | Duration | Bytes | Protocol |
|-----------|----------|-------------|----------|-------|----------|
| ... | ... | ... | ... | ... | ... |
| 12342 | 10.1.0.2 | 16.2.3.7 | 12 | 20K | http |
| 12343 | 18.6.7.1 | 12.4.0.3 | 16 | 24K | http |
| 12344 | 12.4.3.8 | 14.8.7.4 | 26 | 58K | http |
| 12345 | 19.7.1.2 | 16.5.5.8 | 18 | 80K | ftp |
| ... | ... | ... | ... | ... | ... |

Network management



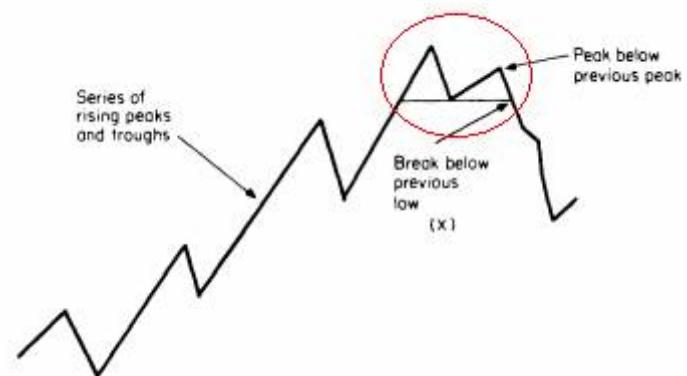
Typical queries:

- 100 most frequent (@S, @D) on router R1 ...
- How many different (@S, @D) seen on R1 but not R2 ...
- ... during last month, last week, last day, last hour ?

Stock monitoring

- Stream of price and sales volume of stocks over time
- Technical analysis/charting for stock investors
- Support trading decisions

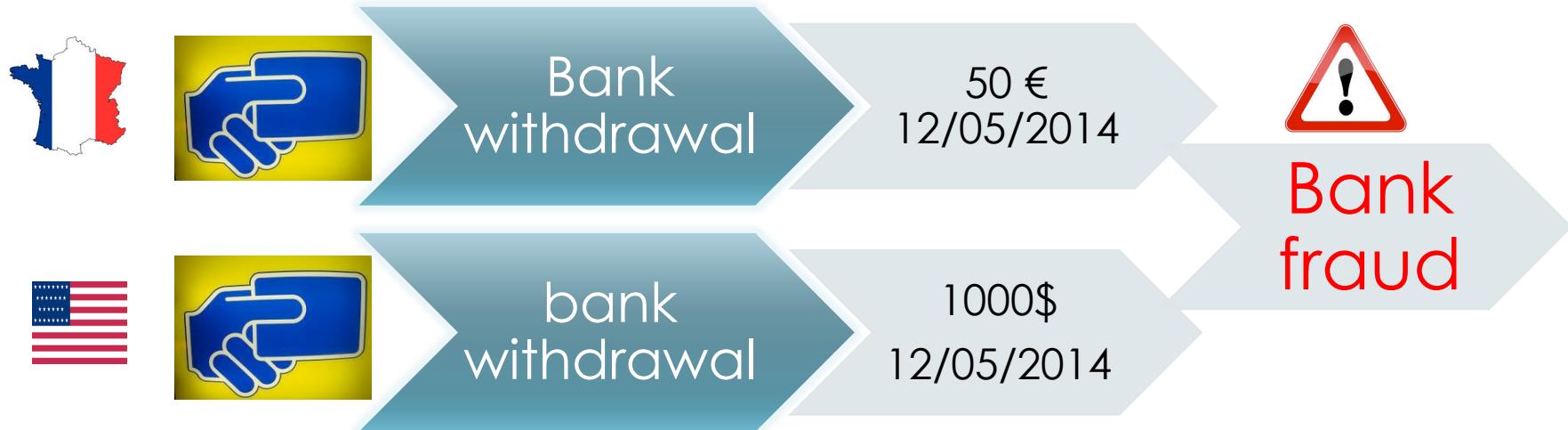
- *Notify me when the price of IBM is above \$83, and the first MSFT price afterwards is below \$27.*
- *Notify me when some stock goes up by at least 5% from one transaction to the next.*
- *Notify me when the price of any stock increases monotonically for ≥ 30 min.*
- *Notify me when the difference between the current price of a stock and its 10 day moving average is greater than some threshold value*



Source: Gehrke 07 and Cayuga application scenarios (Cornell University)

Where is the problem? (1/2)

- Example:



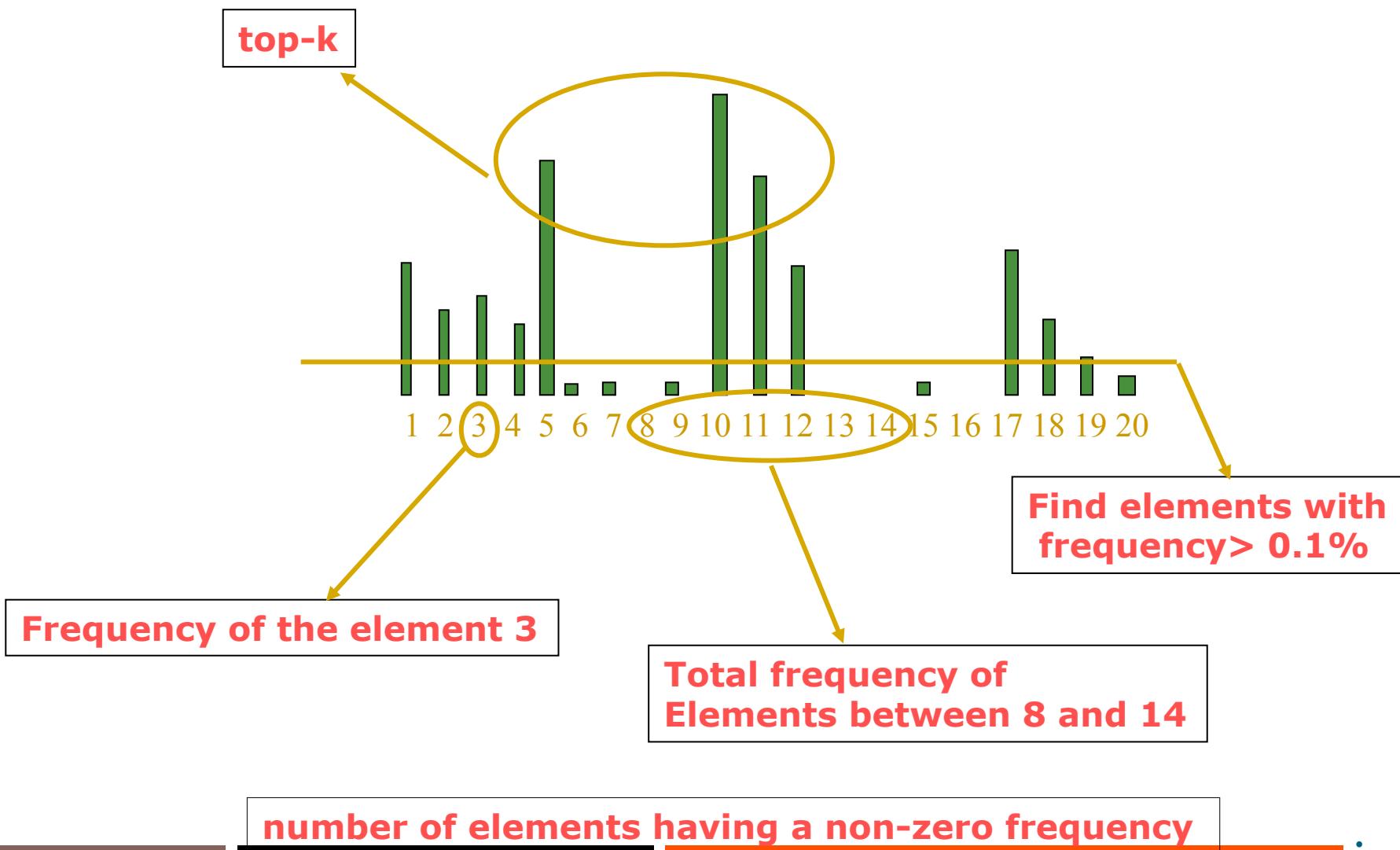
- Join between several streams
- Join between stream data and customer database
- Generic tools for processing streams
- Avoid the ‘Store’, ‘Compute’, ‘Delete’ approach
- Solution: incremental computation and definition of temporal windows for joins

Where is the problem? (2/2)

- **Example:**

- *100 most frequent @S IP addresses on a router*
 - Maintain a table of IP addresses with frequencies ?
 - Sampling the stream ?
- Face high (and varying) rate of arrivals
- Exact versus approximate answers

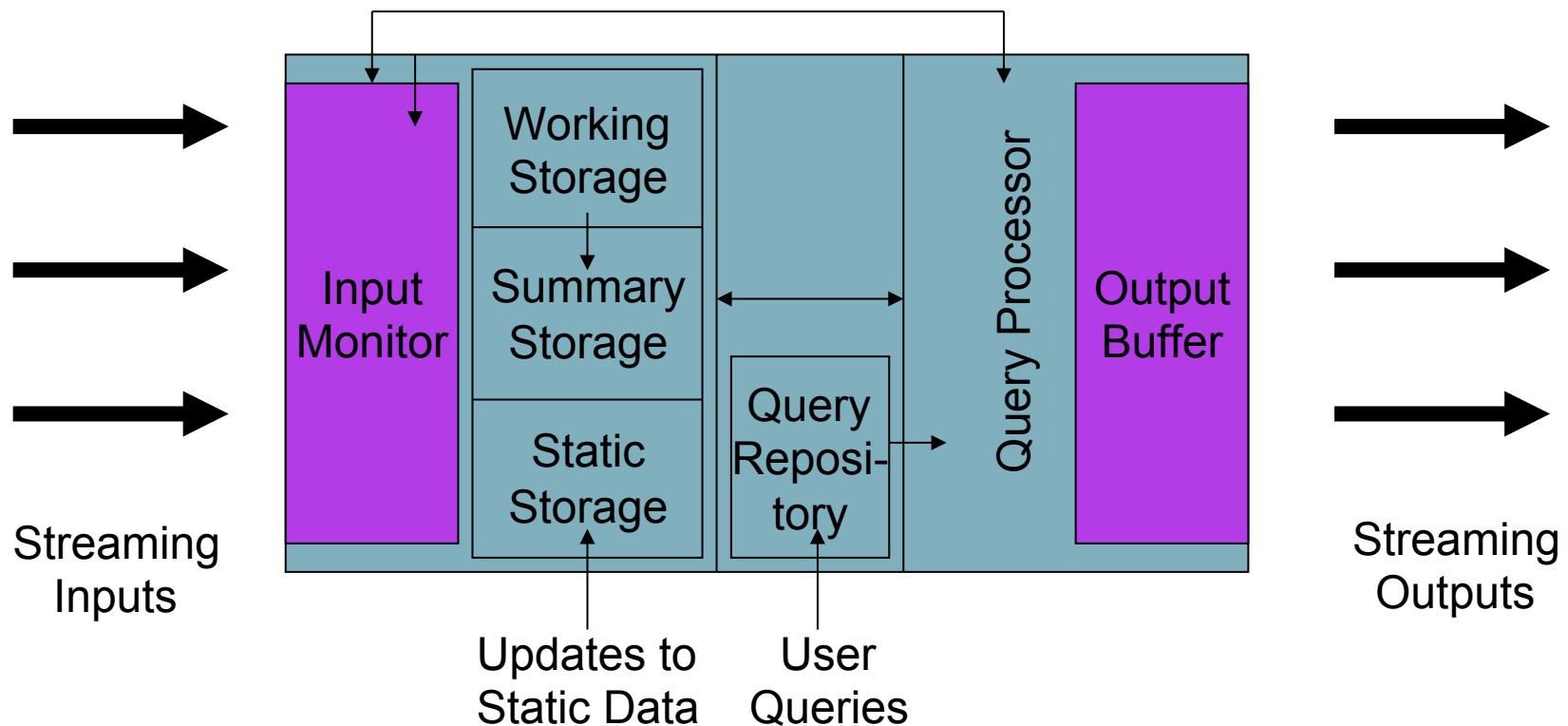
Examples of queries



Data Stream Management Systems

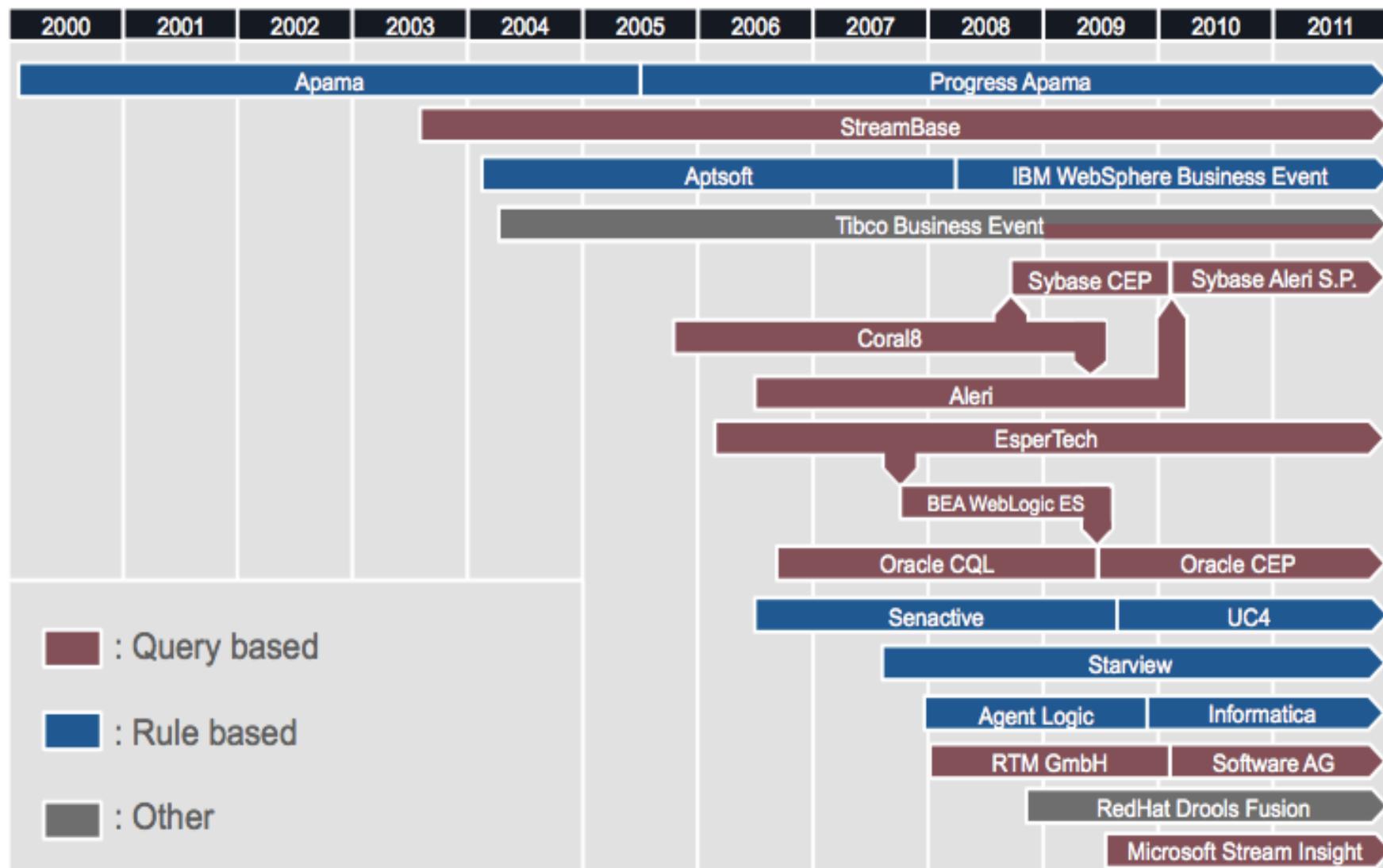
| | DBMS | DSMS |
|-------------|---|---|
| Data model | Permanent updatable relations | Streams and permanent updatable relations |
| Storage | Data is stored on disk | Permanent relations are stored on disk Streams are processed on the fly |
| Query | SQL language Creating structures Inserting/updating/deleting data Retrieving data (one-time query) | SQL-like query language Standard SQL on permanent relations Extended SQL on streams with windowing Continuous queries |
| Performance | Large volumes of data | Optimization of computer resources to deal with Several streams Several queries Ability to face variations in arrival rates without crash |

Generic DSMS architecture

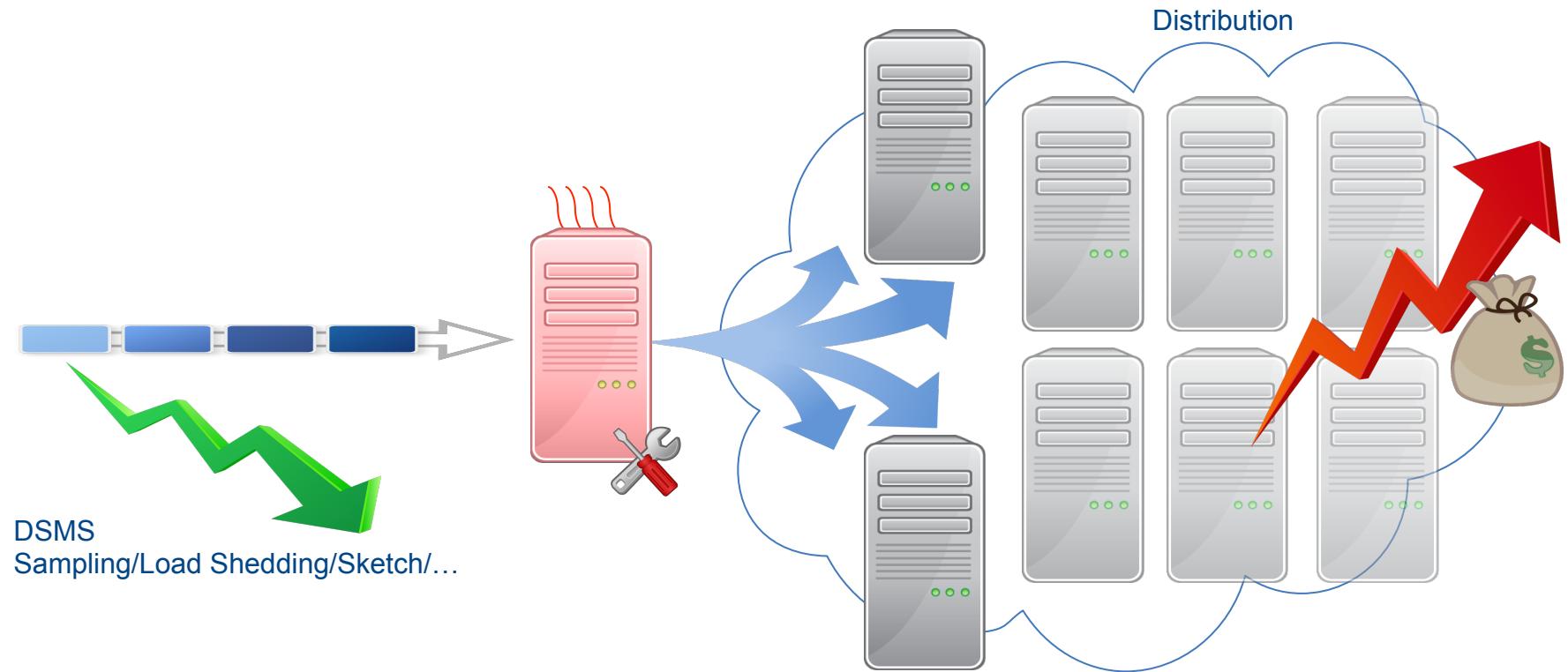


Golab & Oszu (2003)

Existing DSMS



How to deal with Big Data Streams



Approximate answers to queries

- **When ?**

- Queries needing unbounded memory
- Too much queries/too rapid streams/too high response time requirements
 - CPU limit
 - Memory limit

- **Solution : approximate answers to queries**

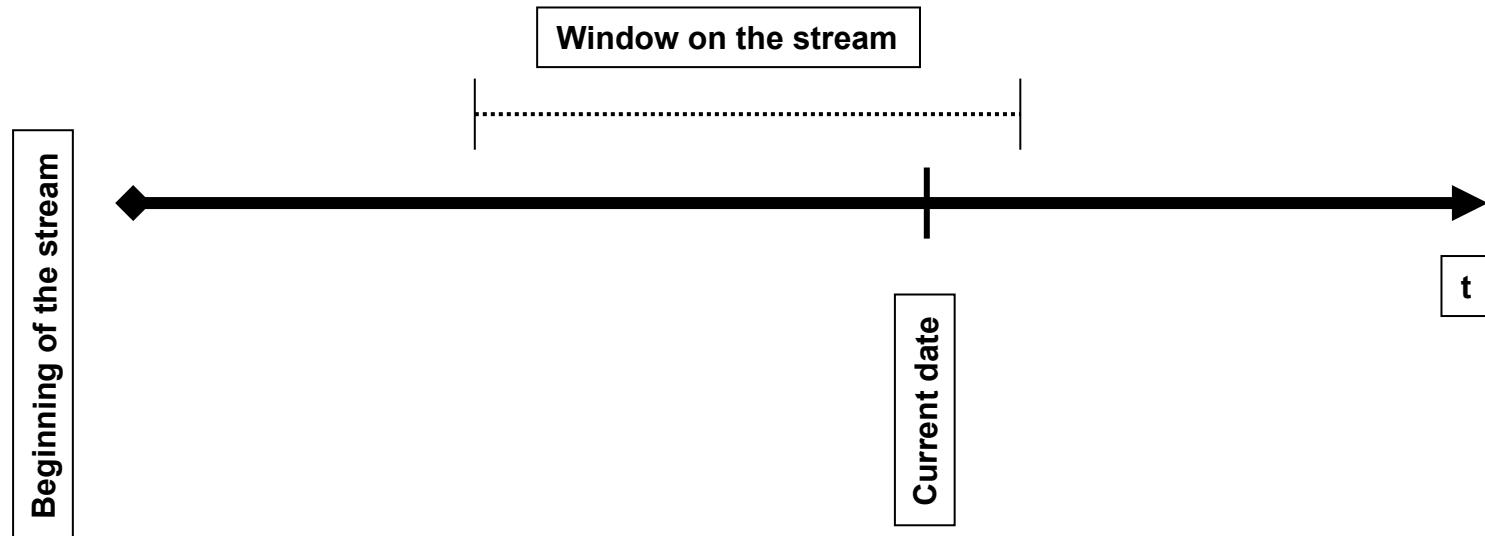
- Sliding windows
- Sampling and load shedding
- Definition of synopsis

Approaches

- **Two approaches for handling such streams**
 - Use a time window, and query the window as a static table
 - When you can't store collected data, or to keep track of historical data
 - Sampling
 - Filtering
 - Counting

Windowing

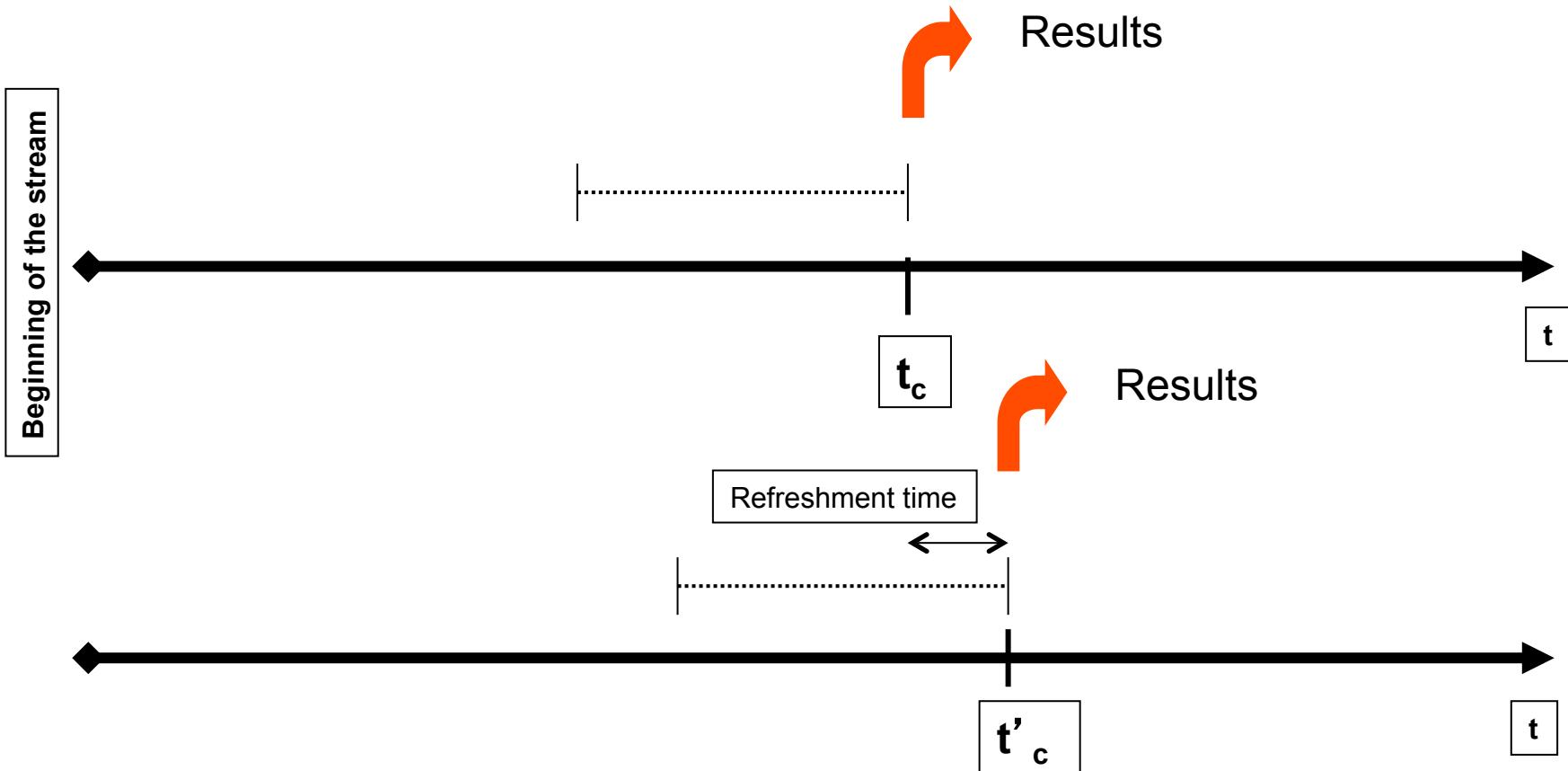
- Applying queries/mining tasks to the whole stream (from beginning to current time)
- Applying queries/mining to a portion of the stream



Windowing

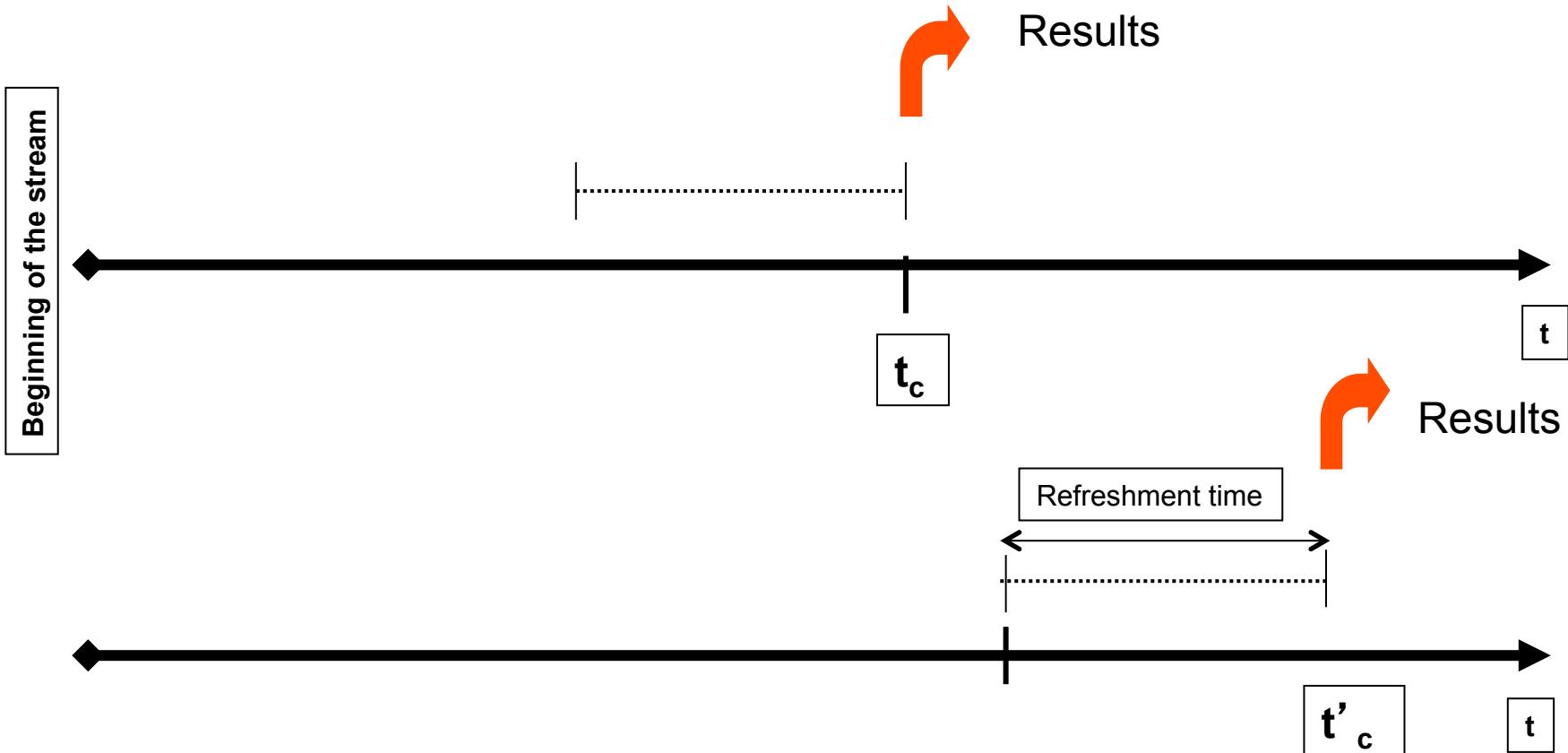
- **Definition of windows of interest on streams**
 - Fixed windows: September 2014
 - Sliding windows: last 3 hours
 - Landmark windows: from September 1st, 2014
- **Window specification**
 - Physical : last 3 hours
 - Logical : last 1000 items
- **Refreshing rate**
 - Rate of producing results (every item, every 10 items, every minute, ...)

Sliding window



*Give me the last room where Axel has been in the last **10 minutes**, updating results **every minute***

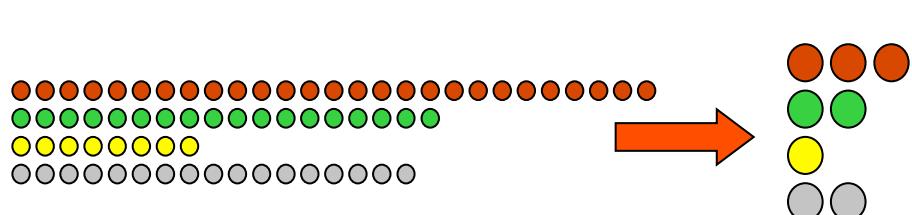
Sliding window vs. Tumbling window



*Give me the last room where Axel has been in the last **10 minutes**, updating results **every 10 minutes***

Sampling from data stream

- **Inputs:**
 - Sample size k
 - Window size $n \gg k$ (alternatively, time duration m) (optionnaly)
 - Stream of data elements that arrive online
- **Output:**
 - k elements chosen uniformly at random from the last n elements (alternatively, from all elements that have arrived in the last m time units)
- **Goal:**
 - maintain a data structure that can produce the desired output at any time upon request
- **Challenge:**
 - don't know how long stream is
 - So when/how often to sample?



A simple, Unsatisfying Approach

- Choose a random subset $X=\{x_1, \dots, x_k\}$, $X \subset \{0, 1, \dots, n-1\}$
- The sample always consists of the non-expired elements whose indexes are equal to x_1, \dots, x_k (modulo n)
- Only uses $O(k)$ memory
- Technically produces a uniform random sample of each window, but unsatisfying because the sample is highly periodic
- Unsuitable for many real applications, particularly those with periodicity in the data

Reservoir sampling

- **Classic online algorithm due to Vitter (1985)**
- **Maintains a fixed-size uniform random sample**
 - Size of the data stream need not be known in advance
- **Data structure: “reservoir” of k data elements**
- **As the i th data element arrives:**
 - Add it to the reservoir with probability $p = k/i$, discarding a randomly chosen data element from the reservoir to make room

Chain Sampling

- ❖ Chain Sampling method is for sequence based windows.
- ❖ In this type of sampling when the i^{th} element arrives it is chosen to become the sample with probability $\text{Min}(i,n)/n$.
- ❖ If the i^{th} element is chosen as the sample, the algorithm also selects the index of the element that will replace it when expires (assuming that it is still present in the sample when it expires). This index is picked uniformly at random from the range $i+1 \dots i+n$, representing the range of indexes of the elements that will be active when the i^{th} element expires.
- ❖ When the element with the selected index arrives, the algorithm stores it in the memory and chooses the index of the element that will replace it when it expires etc., building a chain of elements to use in case of the expiration of the current element in the sample.



Chain Sampling

3 5 1 4 6 2 8 5 **2** 3 5 4 2 2 5 0 9 8 4 6 7 3



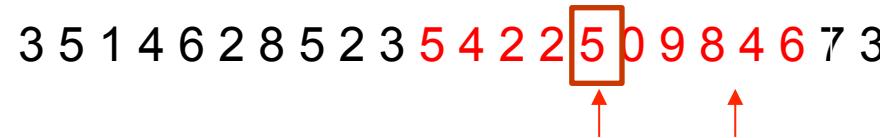
3 5 1 4 6 2 8 5 **2** 3 5 4 2 2 5 0 9 8 4 6 7 3



3 5 1 4 6 2 8 5 **2** 3 5 4 2 2 5 0 9 8 4 6 7 3



3 5 1 4 6 2 8 5 2 3 5 4 2 2 **5** 0 9 8 4 6 7 3



Another Simple Approach: oversample

- As each element arrives remember it with probability $p = ck/n \log n$; otherwise discard it
- Discard elements when they expire
- When asked to produce a sample, choose k elements at random from the set in memory
- Expected memory usage of $O(k \log n)$
- The algorithm can fail if less than k elements from a window are remembered; however with high probability this will not happen

Min-wise Sampling

- For each item, pick a random fraction between 0 and 1
- Store item(s) with the smallest random tag (Nath et al. 2004)



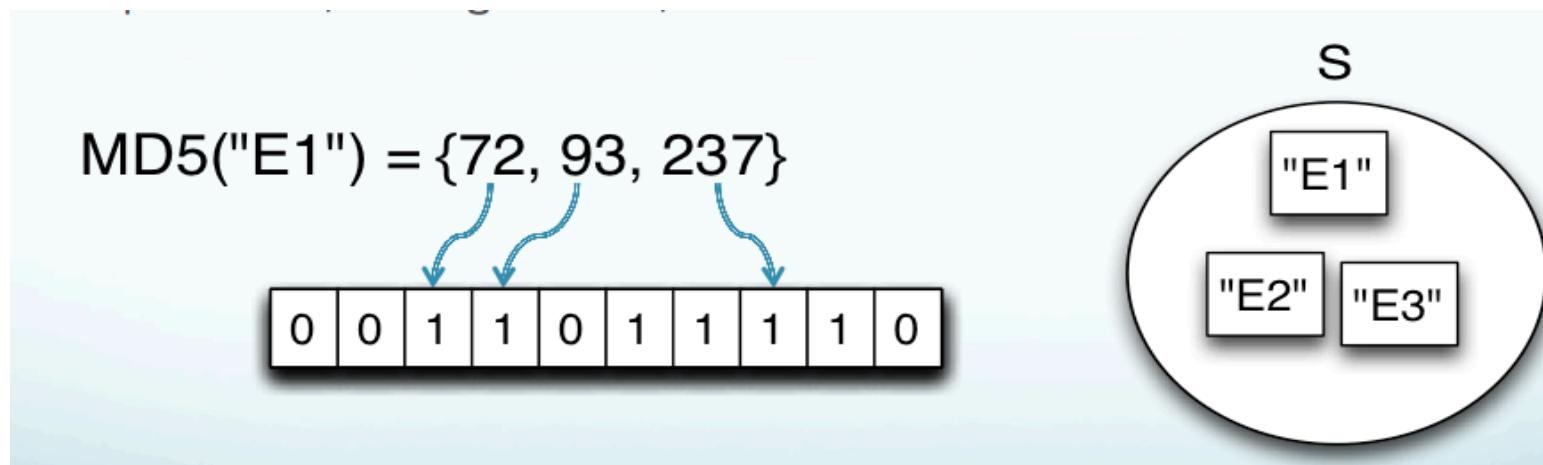
Each item has same chance of least tag, so uniform
Can run on multiple streams separately, then merge

Hash function - Revision

- **Hash function:** Any algorithm that maps data of arbitrary length to data of a fixed length. The values returned by a hash function are called hash values, hash codes, hash sums, checksums or simply hashes.

Stream filtering

- **Compact data structure, by B.H. Bloom, 1970:**
 - A bit array of size m,
 - A H family of k hash functions (MD5, SHA256, Murmur),
 - A set S of n items
 - False positive probability $f=(1-e^{-kn/m})^k$
- **Example M=10, hash=MD5, k=3**



Sketches

- **Sketch**
 - Synopsis structure taking advantage of high volumes of data
 - Provides an approximate result with probabilistic bounds
 - Random projections on smaller spaces (hash functions)
- **Many sketch structures: usually dedicated to a specialized task**
- **Examples of sketch structures**
 - COUNT (Flajolet 85)
 - COUNT SKETCH (Charikar et al. 04)

Difference between Sampling and Sketching



- **Sample sees only those items which were selected to be in the sample; whereas the sketch sees the entire input, but is restricted to retain only a small summary of it**
- **Not every problem can be solved with sampling**
 - Example: counting how many distinct items in the stream
 - If a large fraction of items aren't sampled, don't know if they are all same or all different

Sketches: COUNT (Flajolet Martin algorithm)

- **Goal**

- Number N of **distinct values** in a stream (for large N)
- Example: number of distinct IP addresses going through a router

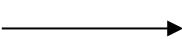
- **Sketch structure**

- SK: L bits initialized to 0

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

- H: hashing function transforming an element of the stream into L bits

18.6.7.1



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

- H distributes uniformly elements of the stream on the 2^L possibilities

Sketches

- **Method**

- Maintenance and update of SK
- For each new element e
- Compute $H(e)$
- Select the position of the leftmost 1 in $H(e)$
- Force to 1 this position in SK

| | | | | | | | | | |
|----|--|---|---|---|---|---|---|---|---|
| SK | <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table> | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | | |

| | | | | | | | | | |
|---------------|---|----------|---|----------|---|---|---|---|---|
| $H(18.6.7.1)$ | <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table> | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | |

| | | | | | | | | | |
|--------|---|----------|---|----------|---|---|---|---|---|
| New SK | <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table> | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | | |



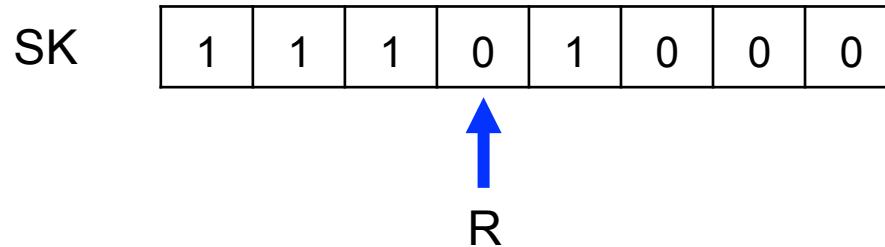
Sketches

- **Result**

- Select the position R ($0 \dots L-1$) of the leftmost 0 in SK
- $E(R) = \log_2 (\varphi^* N)$ with $\varphi = 0.77351\dots$

→ Estimate the count by $2^R/\varphi$

- $\sigma(R) = 1.12$



To improve accuracy of this approximation algorithm, use multiple hash functions and use the average R instead.

Example

| Item value v | Hash Function 1 | | Hash Function 2 | | Hash Function 3 | |
|----------------|-----------------|------------|-----------------|------------|-----------------|------------|
| | b | $M[\cdot]$ | b | $M[\cdot]$ | b | $M[\cdot]$ |
| 15 | 1 | 00000010 | 1 | 00000010 | 0 | 00000001 |
| 36 | 0 | 00000011 | 1 | 00000010 | 0 | 00000001 |
| 4 | 0 | 00000011 | 0 | 00000011 | 0 | 00000001 |
| 29 | 0 | 00000011 | 2 | 00000111 | 1 | 00000011 |
| 9 | 3 | 00001011 | 0 | 00000111 | 0 | 00000011 |
| 36 | 0 | 00001011 | 1 | 00000111 | 0 | 00000011 |
| 14 | 1 | 00001011 | 0 | 00000111 | 1 | 00000011 |
| 4 | 0 | 00001011 | 0 | 00000111 | 0 | 00000011 |

$Z = 2$ $Z = 3$ $Z = 2$

$$\text{Estimate } \hat{F}_0 = \left\lfloor \frac{2^{(2+3+2)/3}}{.77351} \right\rfloor = 6$$

Exercise

- **Stream 3,1,4,1,5,9,2,6,5**

- **Hash functions:**

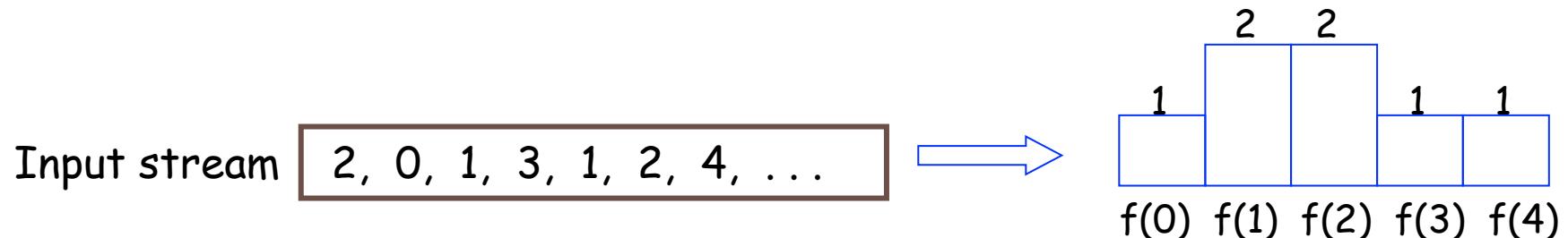
- $h(x) = (2x+1) \bmod 32$
- $h(x) = (3x+7) \bmod 32$
- $h(x) = (4x) \bmod 32$

Result is a binary 5 bit integer. Determine for each hash function:

- Tail length of each element,
- Estimate the number of elements

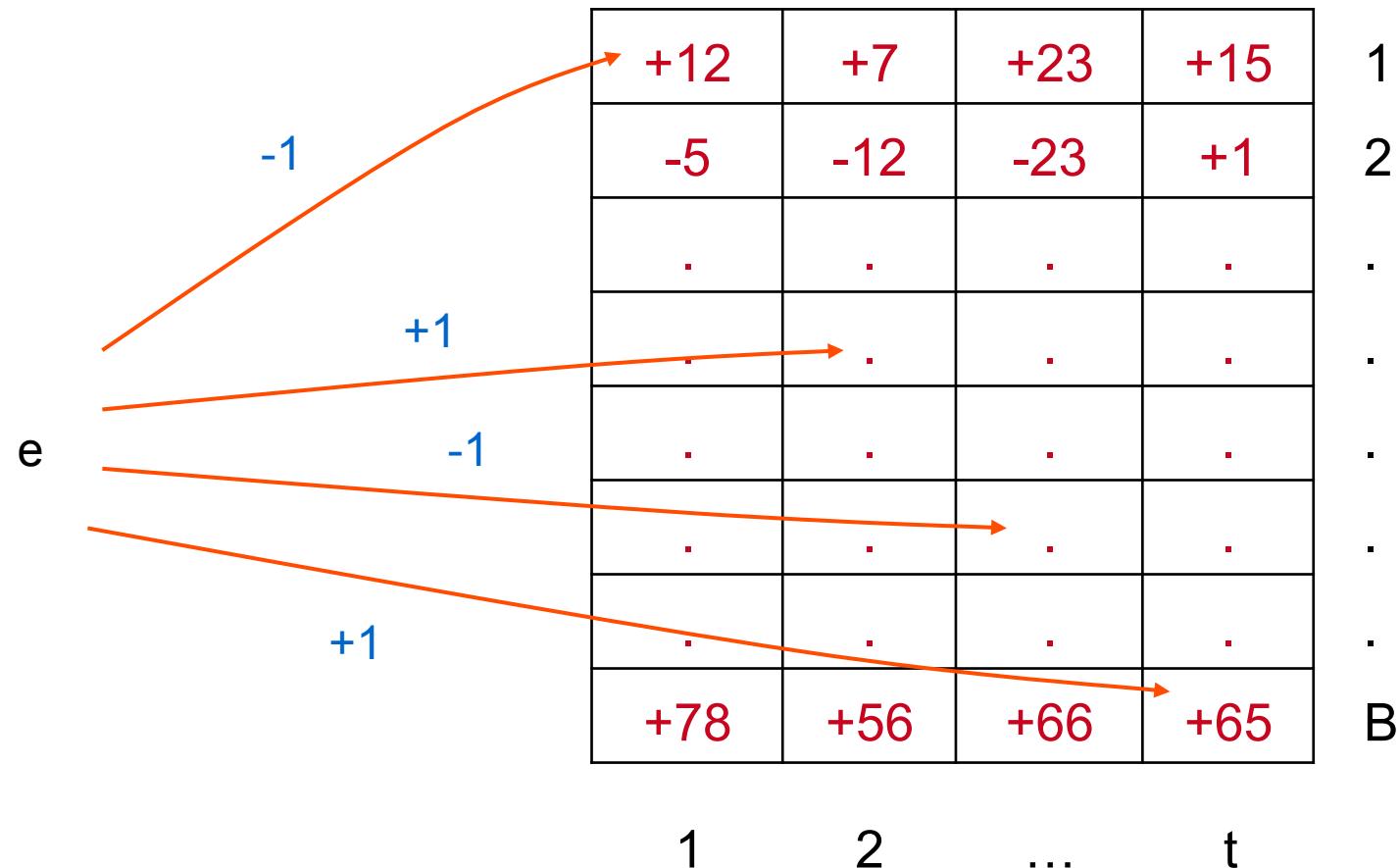
Sketches

- **COUNT SKETCH ALGORITHM (Charikar et al. 2004)**
- **Goal**
 - k most frequent elements in a stream (for large number N of distinct values)
 - Ex. 100 most frequent IP addresses going through a router





Sketches



Sketches

- **Sketch structure**

- h : hash function from $[0, \dots, N-1]$ to $[0, 1, \dots, B]$
- s : hash function from $[0, \dots, N-1]$ to $\{+1, -1\}$
- Array of B counters: C_1, \dots, C_B (with $B \ll N$)

- **Sketch maintenance**

when e arrives: $C_{h(e)} += s(e)$

- **Use of sketch**

- Estimation of frequency of object e : $n_e \approx C_{h(e)} \cdot s(e)$
- Actually t hash function h and t hash function s :

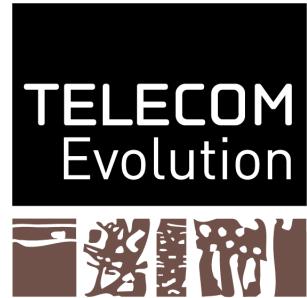
$$n_e \approx \text{median}_{j \in [1 \dots t]} (C_{hj(e)} \cdot s_j(e))$$

- Theoretical results on error depending on N , t and B .

Sketches

▪ Algorithm

- Maintenance of a list (e_1, e_2, \dots, e_k) of the current k most frequent elements
- For a new arriving element e
 - Add e to the sketch structure
 - Estimate frequency of e from the sketch structure
 - If $f(e) > f(e_k)$, remove e_k and insert e into the list



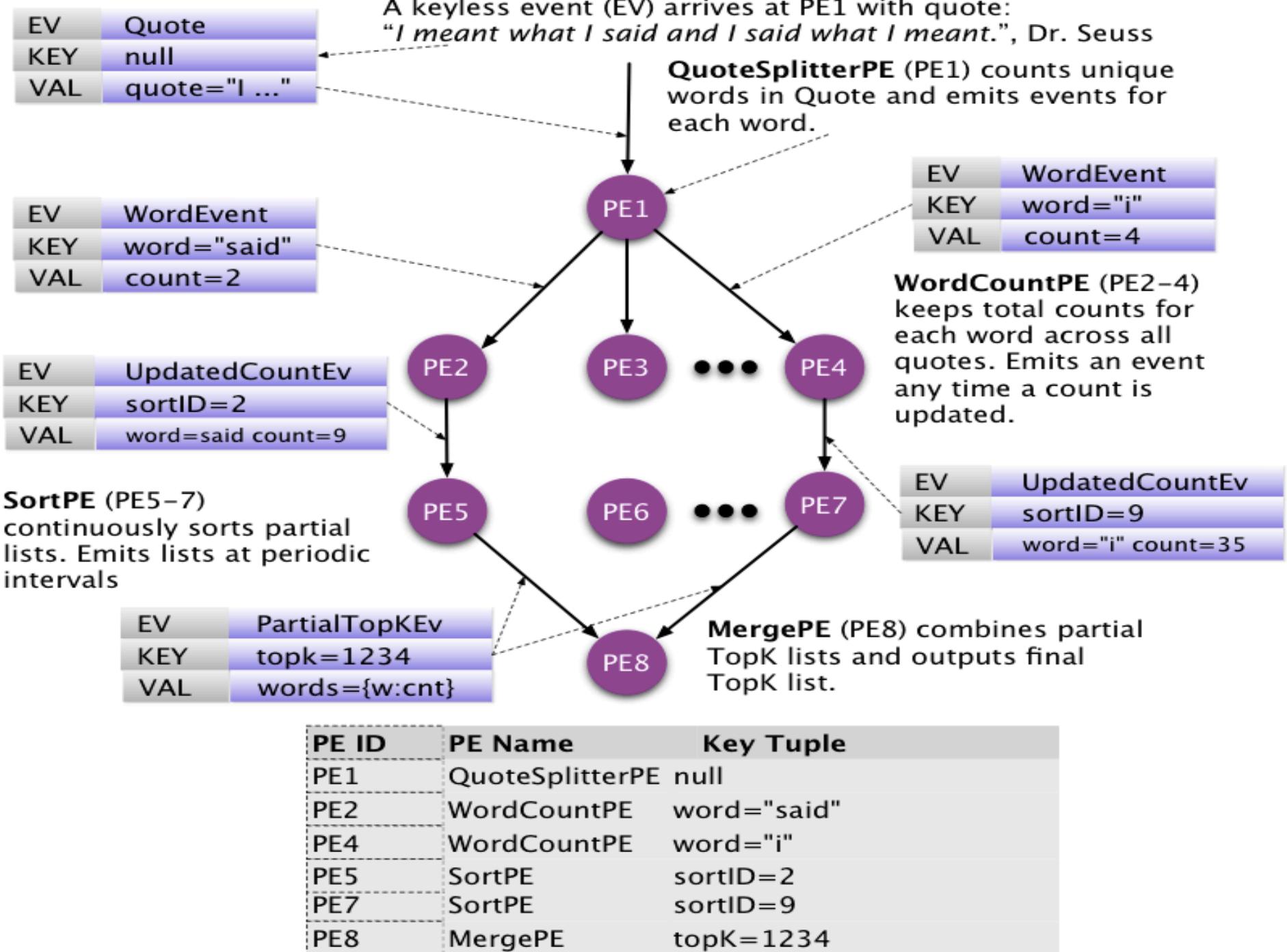
Distributed Systems

Answering today Big Data Needs
S4, Storm, SAMOA, ...



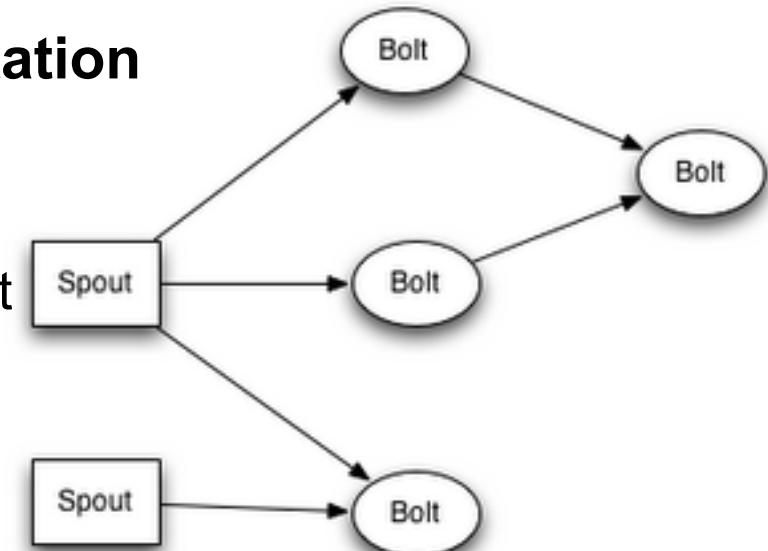
Yahoo S4

- **Simple Scalable Streaming System**
 - Use a decentralized and symmetric architecture
 - All nodes are similar, no master node
- **Based on Processing Elements**
- **Each PE has 4 components:**
 1. Functionality defined by a PE class and associated configuration
 - input event handler *processEvent()*
 - output mechanism *output()*
 2. the types of events that it consumes
 3. the keyed attribute in those events
 4. the value of the keyed attribute in events which it consumes
- **Several PEs are available for standard tasks such as count, aggregate, join ...**
 - Custom PEs can easily be programmed
- **The PEs are automatically deployed on the Processing Nodes (machines):**
 - Ensures load balancing of events,
 - Notifies appropriate PEs when an event comes in.

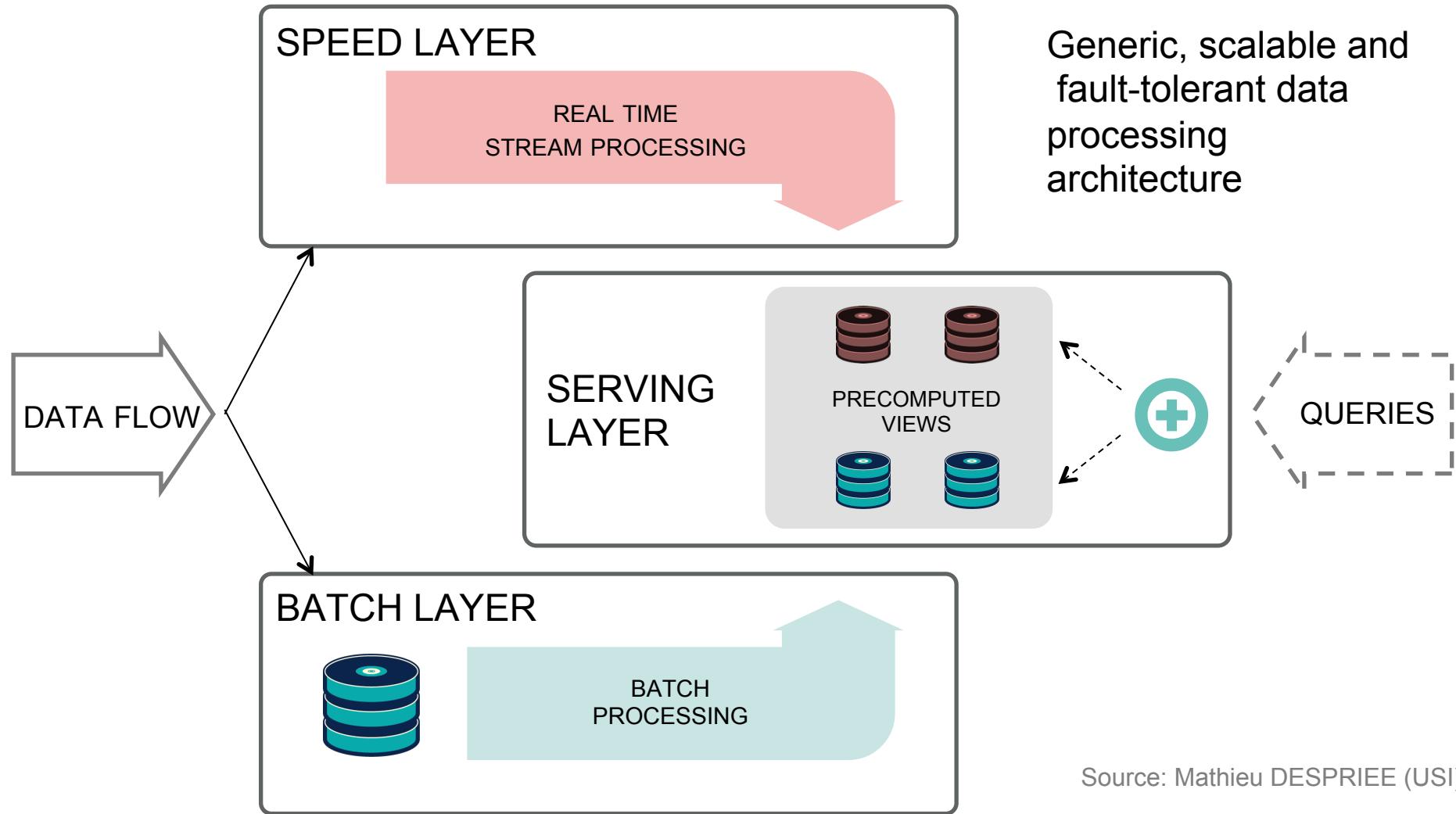


Twitter Storm

- Same principle as S4, with a simplified programming model
- Storm provides realtime computation
 - Scalable
 - Guarantees no data loss
 - Extremely robust and fault-tolerant
- Concepts
 - Streams
 - Spouts
 - Bolts
 - Topologies



Lambda architecture (By Nathan Marz)

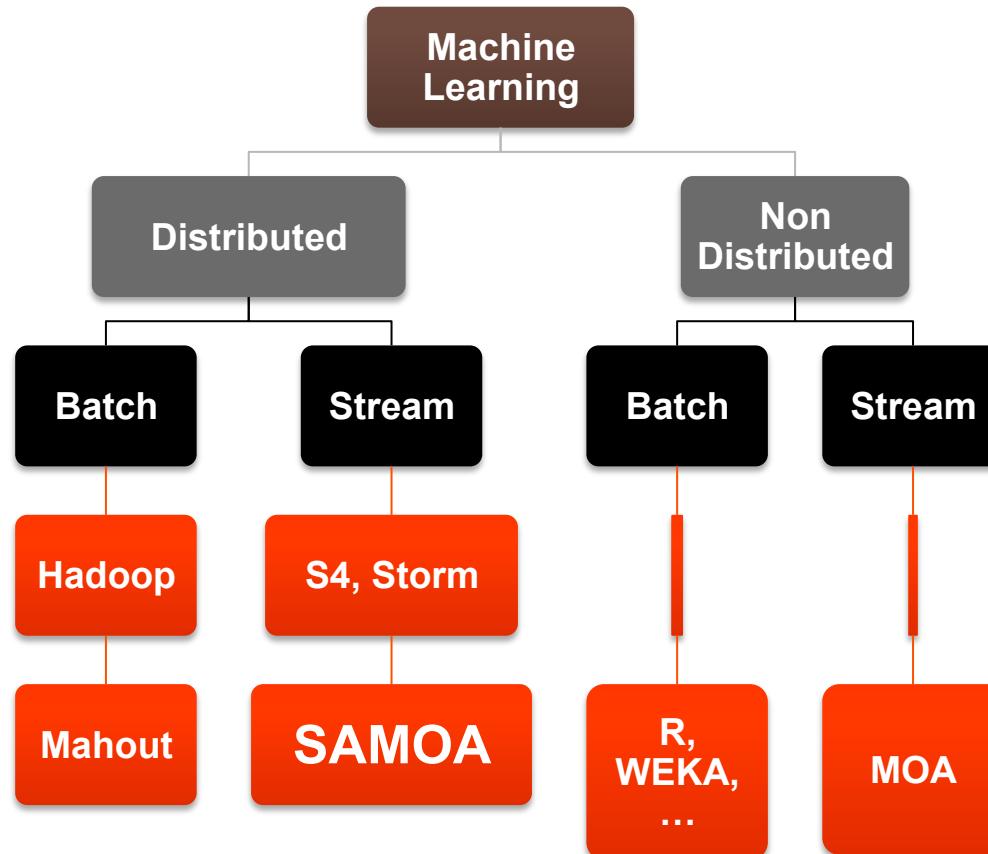


Source: Mathieu DESPRIEE (USI)

Lambda architecture

1. All data entering the system is dispatched to both the batch layer and the speed layer for processing.
2. The batch layer has two functions: (i) managing the master dataset (an immutable, append-only set of raw data), and (ii) to pre-compute the batch views.
3. The serving layer indexes the batch views so that they can be queried in ad-hoc way.
4. The speed layer compensates for the high latency of updates to the serving layer and deals with recent data only.
5. Any incoming query can be answered by merging results from batch views and real-time views.

Big Data Stream Mining



What is SAMOA?

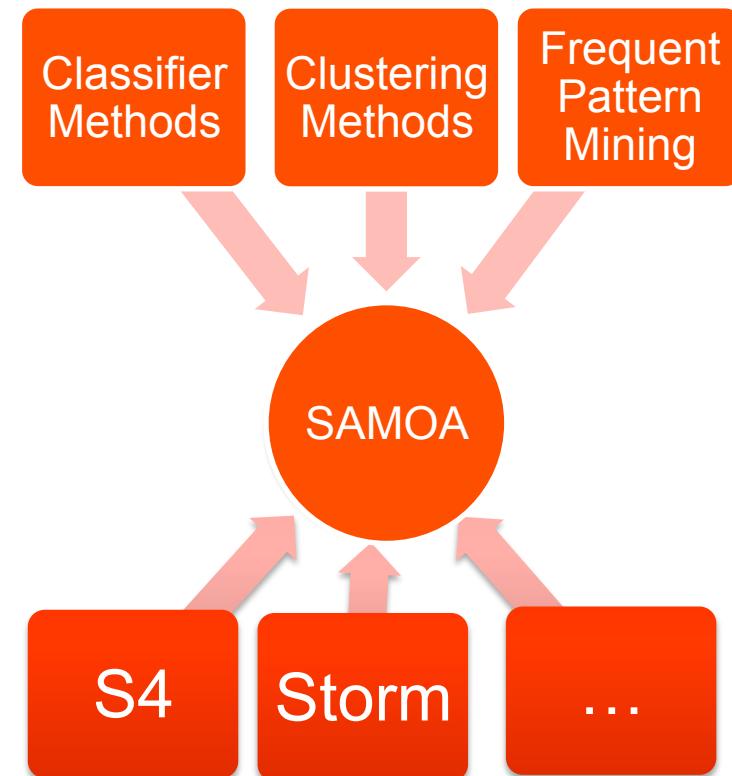


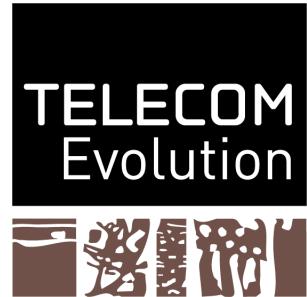
The screenshot shows the homepage of the SAMOA Scalable Advanced Massive Online Analysis website. The header features three colored squares (brown, black, orange) followed by the title "What is SAMOA?". The main content area has a white background with a dark header bar containing the text "WELCOME TO SAMOA!". Below this is a search bar with a magnifying glass icon and the word "Search". A large, scenic photograph of a tropical beach with palm trees and clear blue water is centered on the page. To the right of the photo is a sidebar with the text: "Upcoming **BIG DATA** Mining Project! The goal of **SAMOA** is to provide a framework for mining data streams using a cluster/cloud environment. In particular, we are interested in using Storm (<http://storm-project.net>) and S4 (<http://incubator.apache.org/s4/>) as the underlying computational framework."

- **NEW Software framework for mining distributed data streams**
- **Big Data mining for evolving streams in REAL-TIME**

SAMOA architecture

- ▶ Use S4, Storm, or other distributed stream processing platform
- ▶ Use MOA, or other streaming machine learning library
- ▶ Easy to extend through PACKAGES

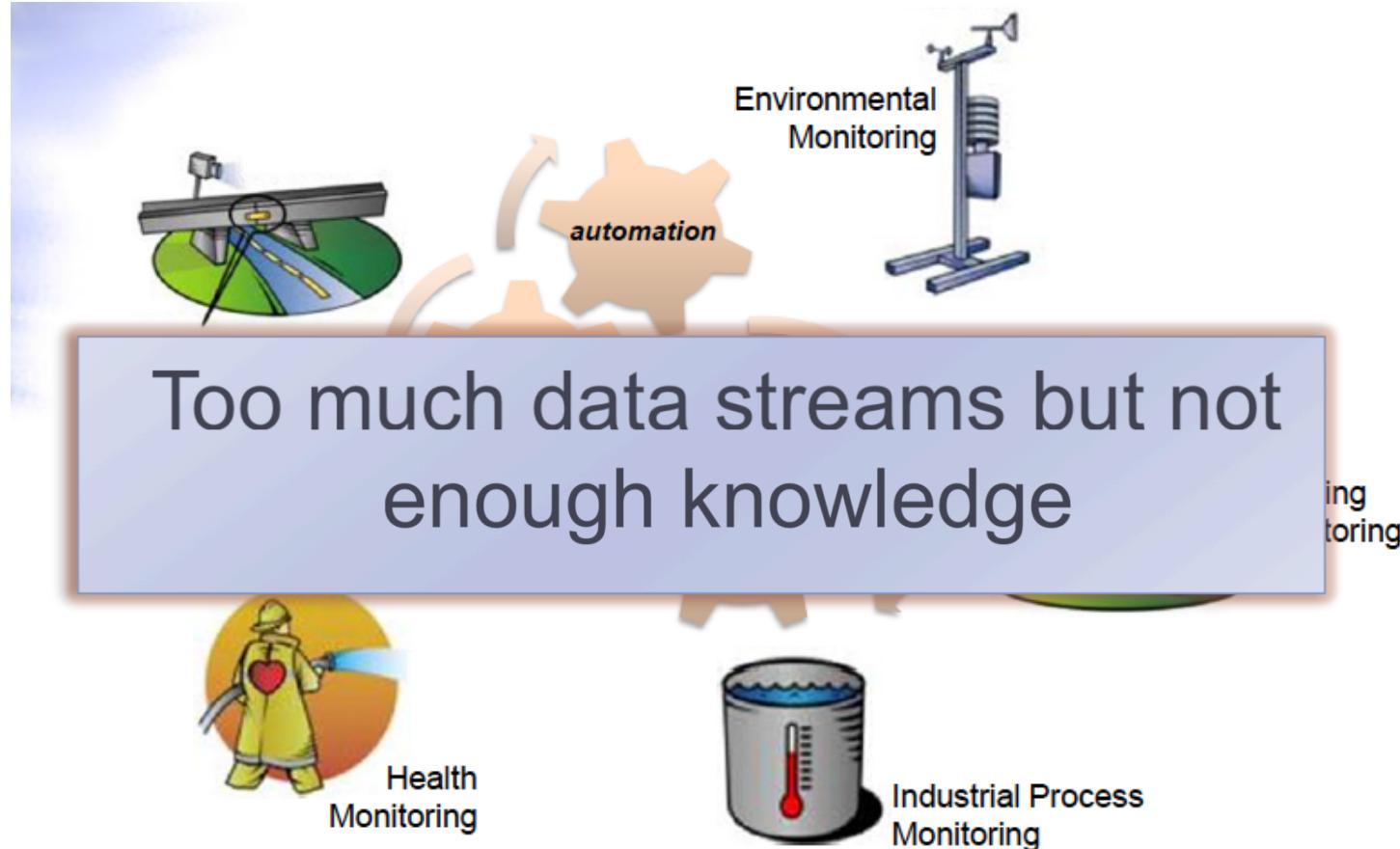




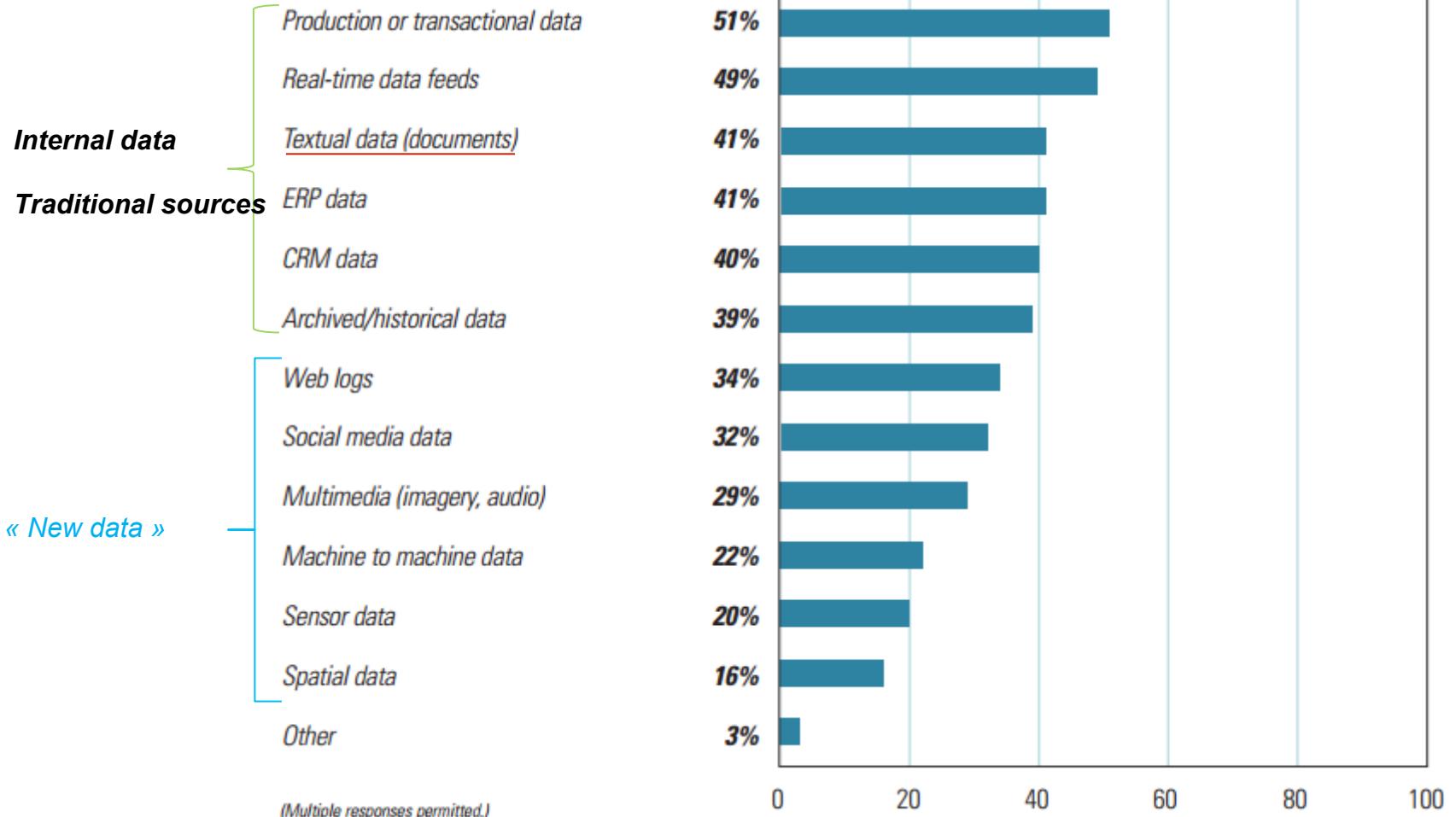
Semantic data streaming



Too much data streams

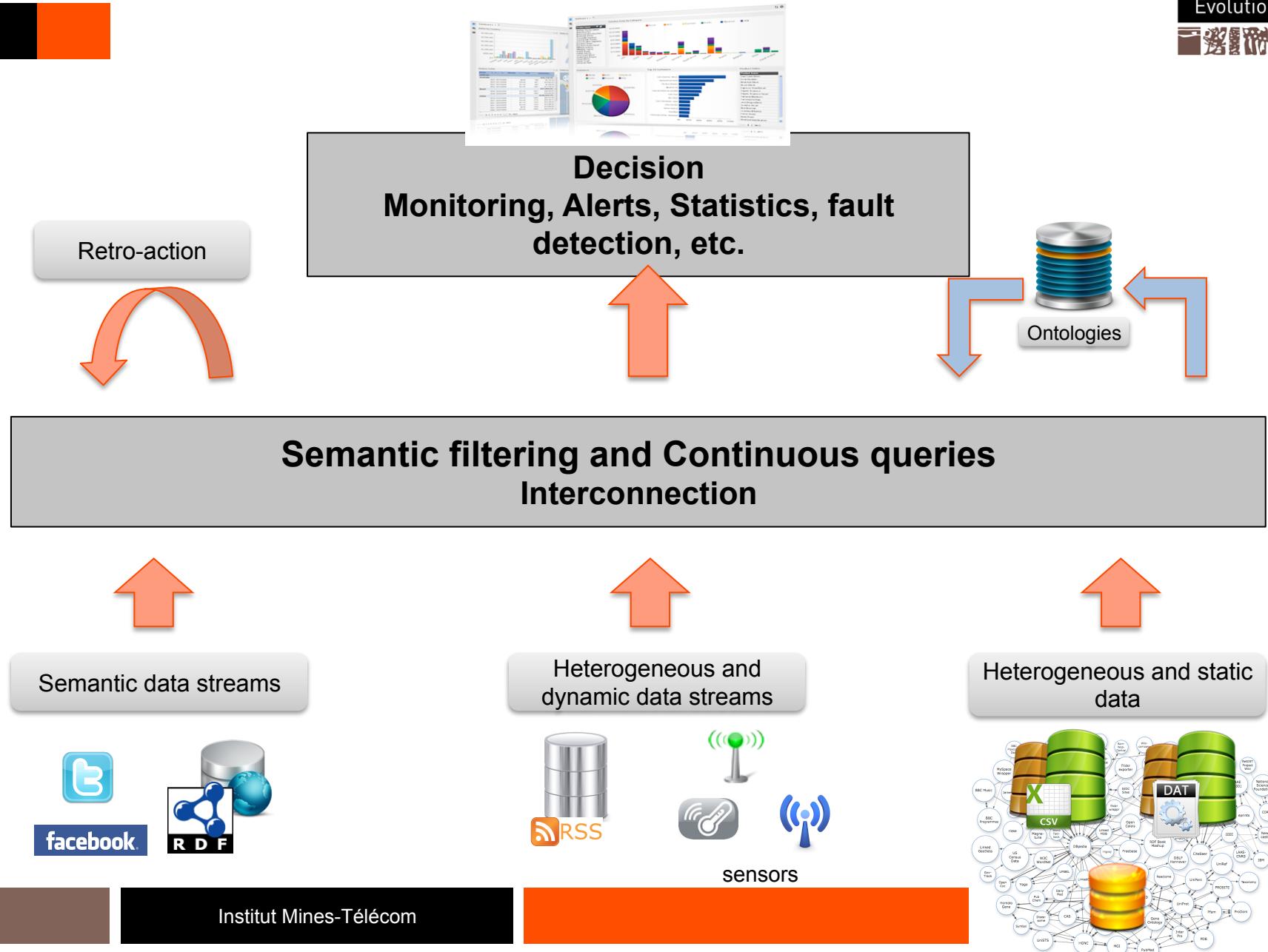


Type of data used in Big Data initiatives



Source: Big Data opportunities survey, Unisphere / SAP, May 2013.

BI: New Generation





Semantic Web technologies for data stream



- Annotate stream data **with semantic metadata**
- Apply **Linked Data principles** to publish streaming data
- Interlink streaming data with **existing datasets**
- Integrate data stream processing + **reasoning**

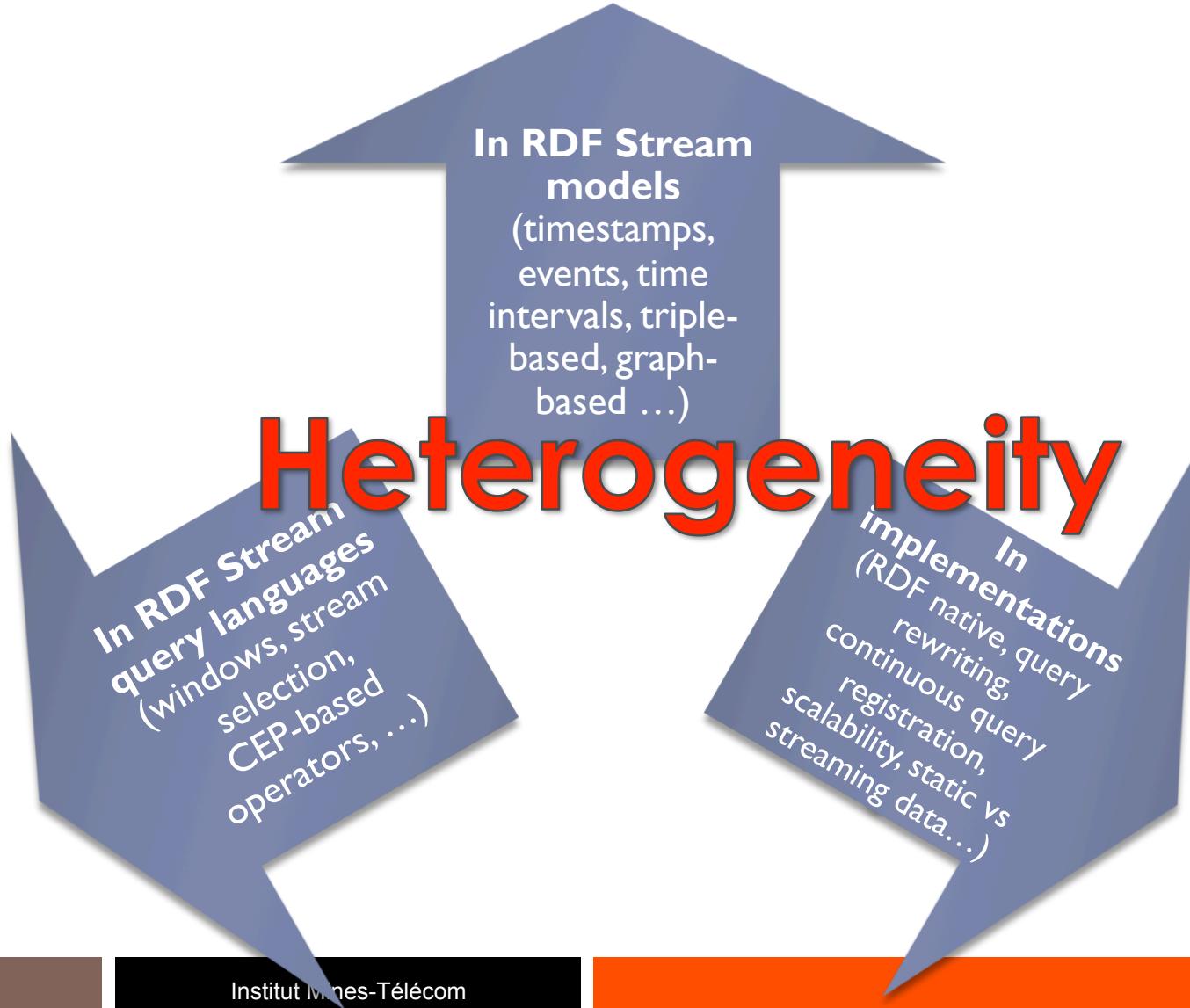
- **Objectives :** interoperability, automation, enrichment



Existing prototypes



Approaches



Example: SRBench

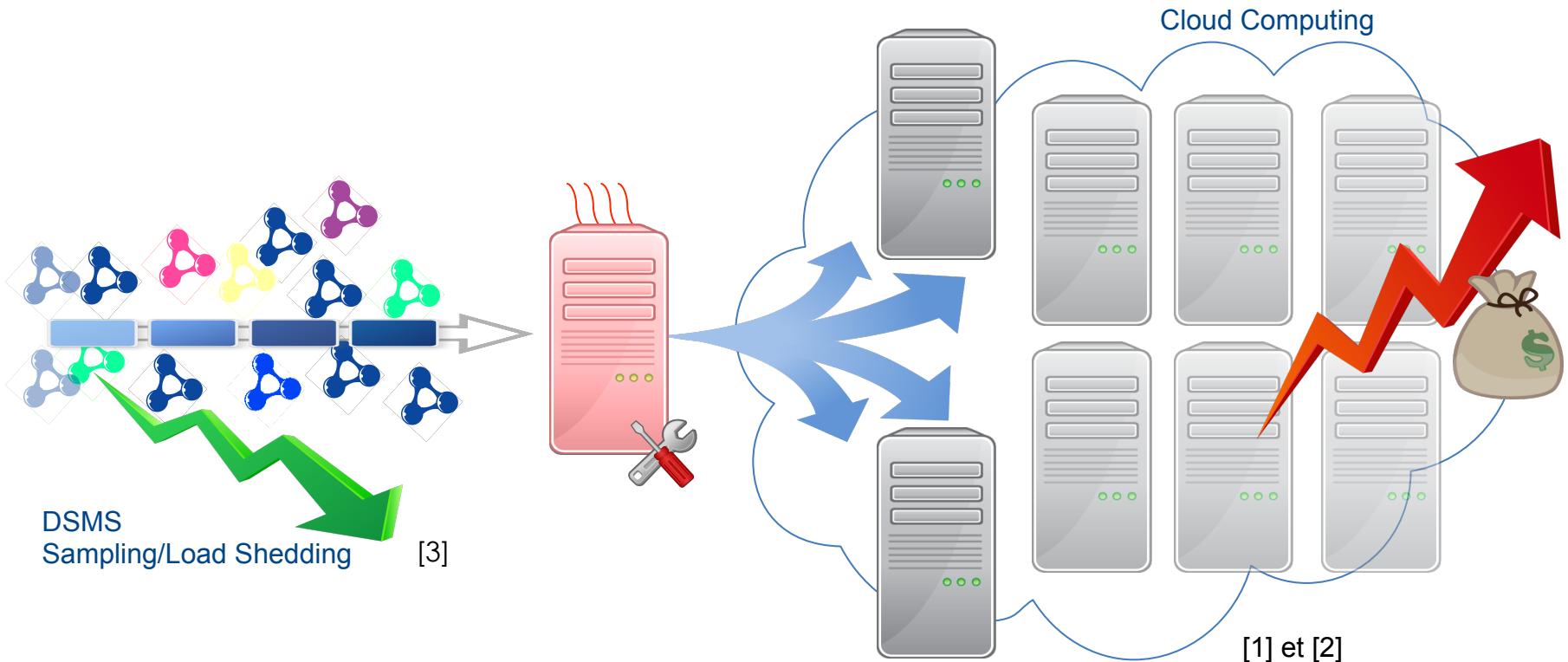
Q: Detect if a hurricane has been observed

“A hurricane has a sustained wind (for more than 3 hours) of at least 33 metres per second or 74 miles per hour (119 km/h).”

```
ASK
WHERE {
  STREAM <http://www.cwi.nl/SRBench/observations> [RANGE 10800s SLIDE 600s]
    {?observation om-owl:procedure ?sensor ;
     om-owl:observedProperty weather:WindSpeed ;
     om-owl:result [ om-owl:floatValue ?value ] .}
  }
GROUP BY ?sensor
HAVING ( AVG(?value) >= "74"^^xsd:float )
```

```
ASK
FROM STREAM <http://www.cwi.nl/SRBench/observations> [RANGE 3h STEP 10m]
WHERE {
  ?observation om-owl:procedure ?sensor ;
  om-owl:observedProperty weather:WindSpeed ;
  om-owl:result [ om-owl:floatValue ?value ] .}
GROUP BY ?sensor
HAVING ( AVG(?value) >= "74"^^xsd:float )
```

How to deal with Big Data Streams



- [1] J. Hoeksema & S. Kotoulas : High-performance Distributed Stream Reasoning using S4 (ISWC 2011)
- [2] D. L. Phuoc & al : Elastic and Scalable Processing of Linked Stream Data in the Cloud (ISWC 2013)

Sampling Extensions for continuous SPARQL



```
PREFIX vocab: http://data-gov.tw.rpi.edu/vocab/p/8/
SELECT ?val
WHERE {
  STREAM <C:/CQELS/streams/data-8.stream> [NOW] [UNISAMPLING %80]
  { ?rawData vocab:ozone_8hr_daily_max ?val }
  GRAPH <C:/CQELS/test.rdf> { ?user sioc:account_of ?person }
}
```

Operators: [**UNISAMPLING** %*{Sampling Percentage}*]
[**RESSAMPLING** %*{Reservoir Size}*]
[**CHNSAMPLING** %*{Window Size}*]

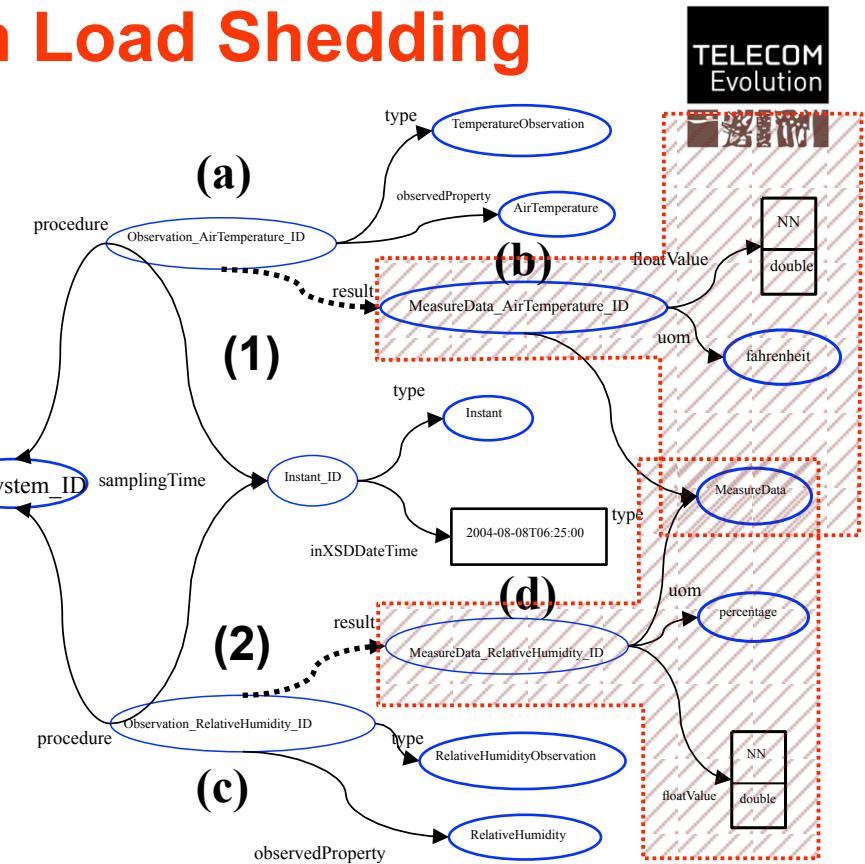
N. Jain & al : Sampling Semantic Data Stream: Resolving Overload and Limited Storage Issues. (DaEng 2013)

Semantic Data Stream Load Shedding (1/2)

RDF Triple approach :

Effect of deleting the two triples below (1) and (2) (in dotted line) in the graph :

- The deletion of the first RDF triple destroys the link connecting node (a) to node (b)
 - The deletion of the second RDF triple destroys the link connecting node (c) to node (d)
- => This has the effect of making nodes (b), (d) and all those connected to them unreachables, in spite of the presence of their data in memory. Which represents 6 RDF stripes among 18 i.e. **33% of unusable data**

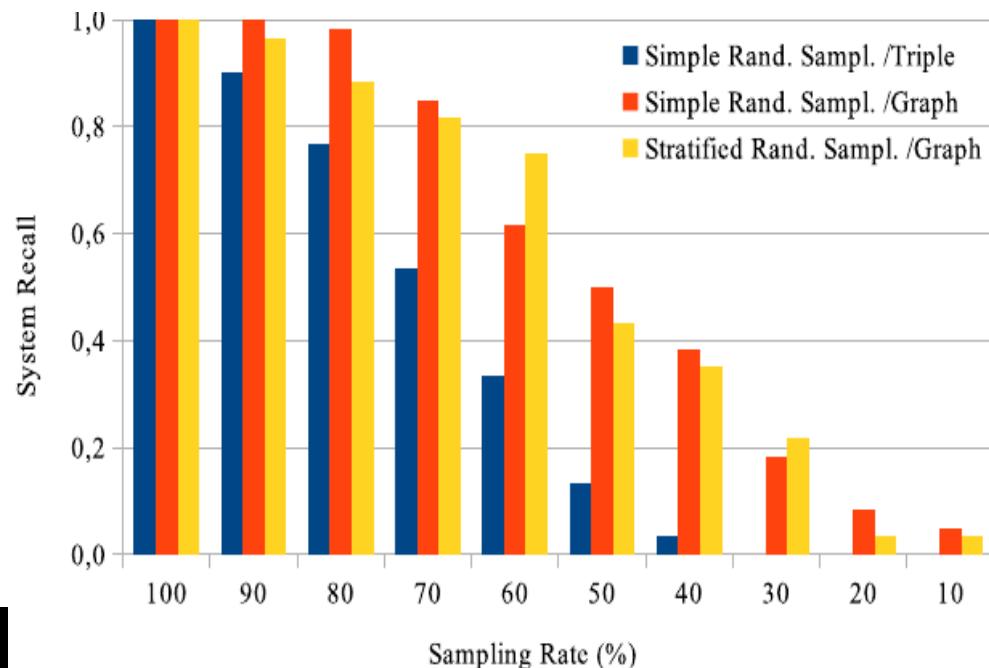


Semantic Data Stream Load Shedding (2/2)

RDF Graph approach :

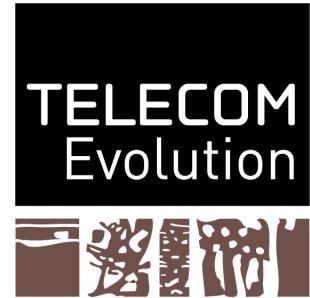
Effect of deleting sub-graphs such as those formed by nodes (b) or (d) and all nodes to which they are directly connected.

- => Preserving the semantic level of the information
- => Protecting the data consistency of the whole graph
- => Enhancing the Semantic Data Stream systems



Conclusion: Big Data Stream challenges

- **Semantic Information aggregation**
 - Information aggregation: “too much data to assimilate but not enough knowledge to act”
- **Distributed and real-time processing**
 - Design of real-time and distributed algorithms for stream processing and information aggregation
 - Distribution and parallelization of data mining algorithms
- **Visual analytics and user modeling**
 - Dynamic user model
 - Novel visualizations for very large datasets



Thanks to
Zakia Kazi Aoul, ISEP
Marie-Aude Aufaure, ECP
Fethi Belghaouti, ISEP-INT
Georges Hébrail, EDF R&D
Sylvain Lefebvre, ISEP
Youstra Chabchoub, ISEP

