Data Streaming

Raja CHIKY
raja.chiky@isep.fr
2015-2016

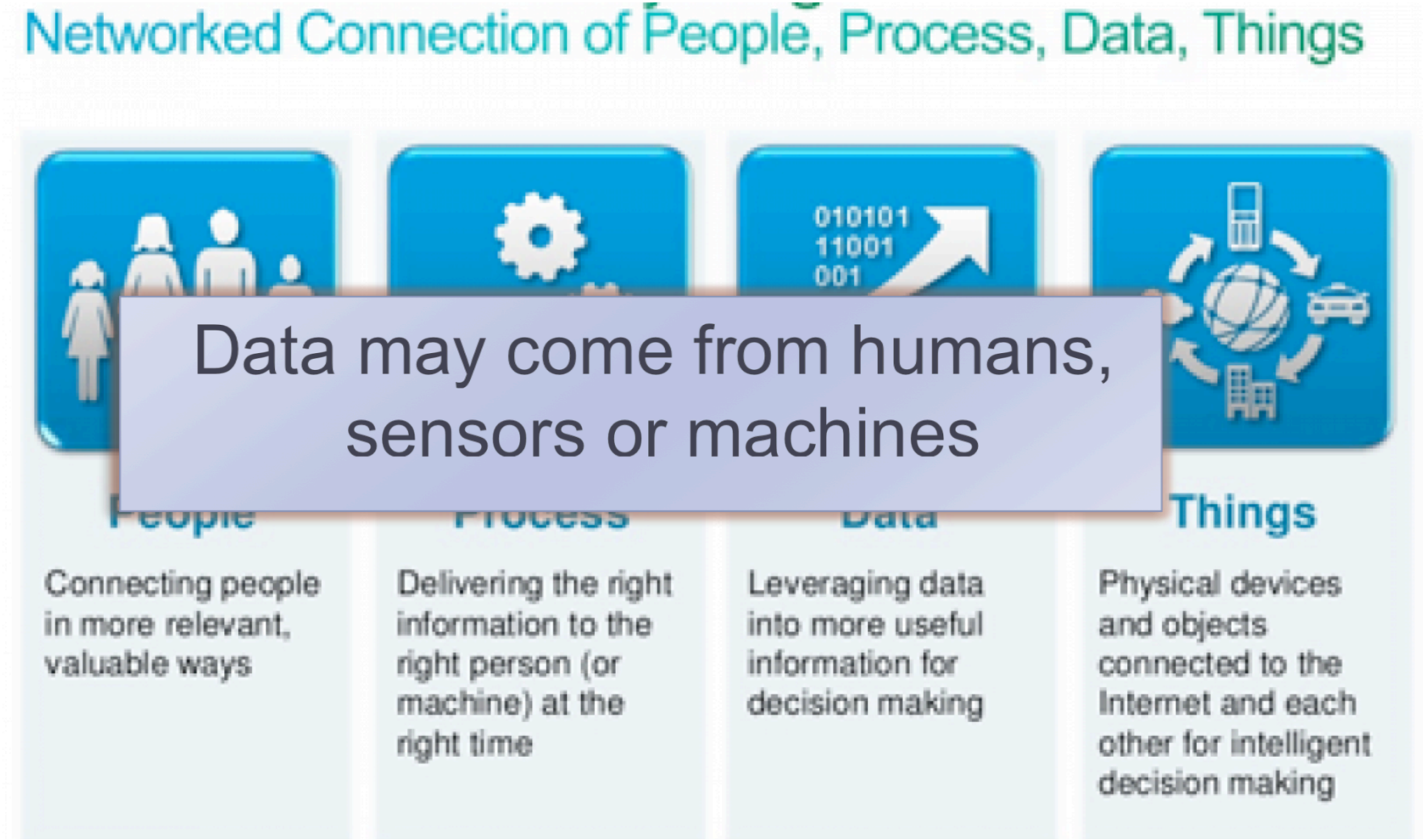
OUTLINE

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

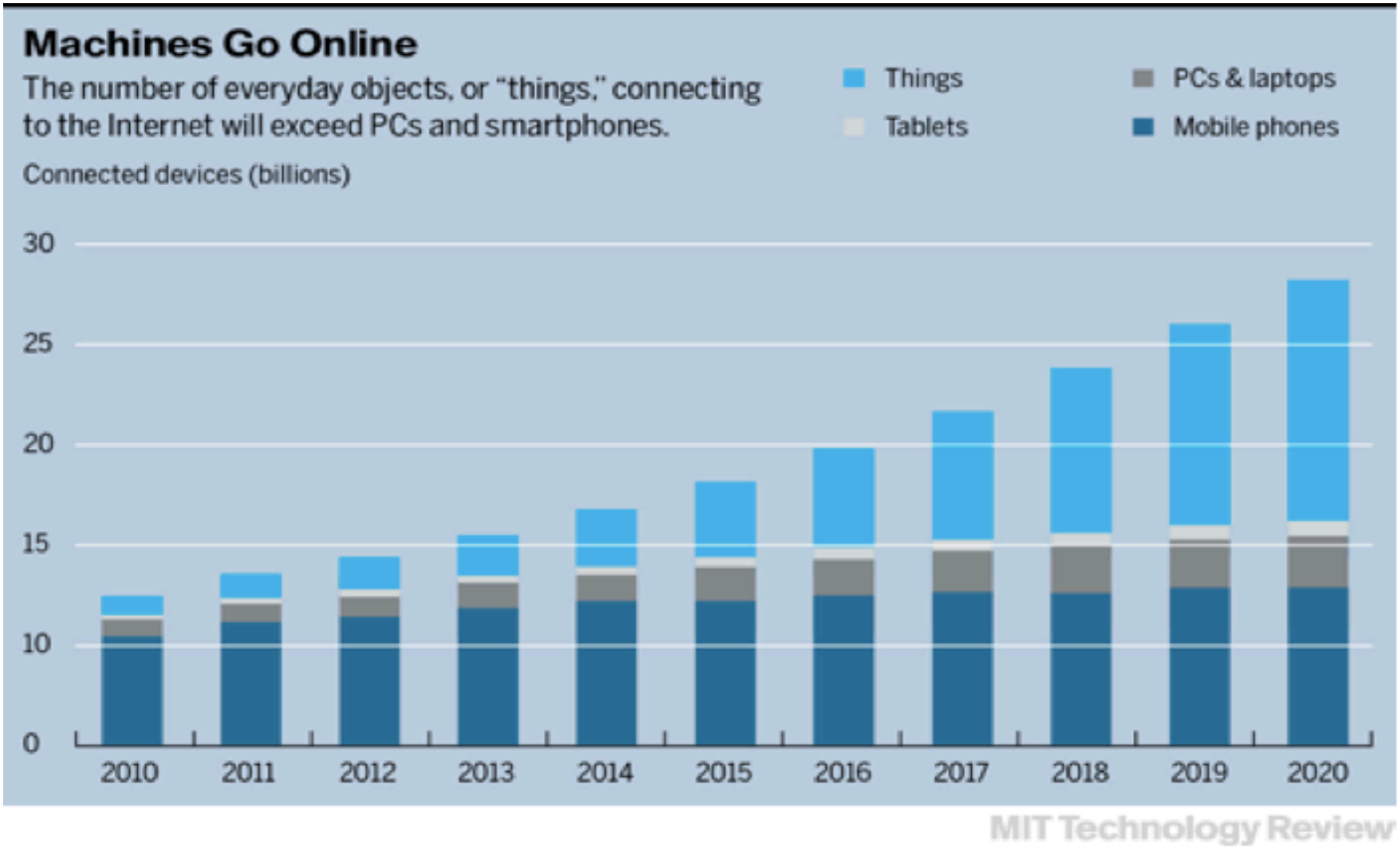
New era


IN 60 SECONDS
1 NEW
1,600+ READS ON Scribd
13,000+ MUSIC STREAMING ON PANDORA
12,000+ NEW ADS POSTED ON craigslist
370,000+ MINUTES VOICE CALLS ON skype
98,000+ TWEETS
20,000+ NEW POSTS ON tumblr.
13,000+ iPhone APPLICATIONS DOWNLOADED
320+ NEW twitter ACCOUNTS
100+ NEW LinkedIn ACCOUNTS
1 NEW ARTICLE IS PUBLISHED
THE WORLD'S LARGEST COMMUNITY CREATED CONTENT
QUESTIONS ASKED ON THE INTERNET...
100+
40+
600+ NEW VIDEOS
6,600+ NEW flickr
50+ WordPress DOWNLOADS
25+ HOURS TOTAL DURATION
70+ DOMAINS REGISTERED
60+ NEW BLOGS
1,500+ BLOG POSTS
168 MILLION EMAILS ARE SENT
694,445 SEARCH QUERIES
1,700+ Firefox DOWNLOADS
695,000+ facebook STATUS UPDATES
125+ PLUGIN DOWNLOADS
79,364 WALL POSTS
$10,040 COMMENTS
Google
GO-Globe.com

Where is all this data coming from?



More and More connected Things



Output

User Interaction

Store

Gathering Information

Data sources

Static data

Ad-hoc queries
Analytics

Data Warehouse

ETL
Batch
processing

databases

Output

User Interaction

Store

Gathering Information

Data sources

Static data

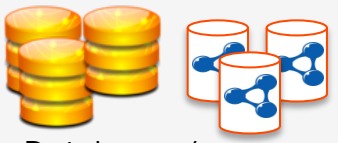
Ad-hoc queries
Analytics

Data Warehouse

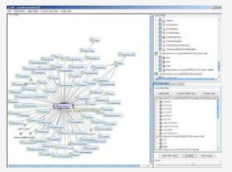
ETL
Batch
processing

databases

Stream (big) data



Databases/
Triplestores
(synopsis)



Real time
visual-analytics

Knowledge
enrichment

Continuous
queries/
Business rules

Retro-action

Semantic ETL
stream
processing

Load shedding

Data stream

Static data

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**
  - Process queries (compute statistics, activate alarms)
  - Apply data mining algorithms
- **Requirements**
  - Real-time processing
  - One-pass processing
  - Bounded storage (no complete storage of streams)
  - Possibly consider several streams

Applications of data stream processing

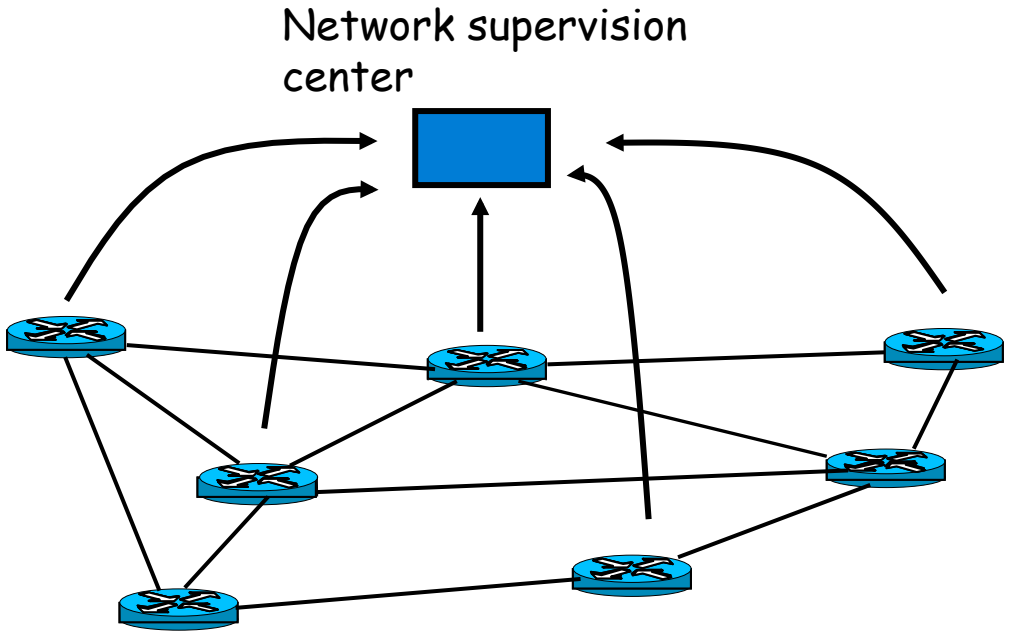
- **Let's go deeper into some examples**
  - Network management
  - Stock monitoring

Network management

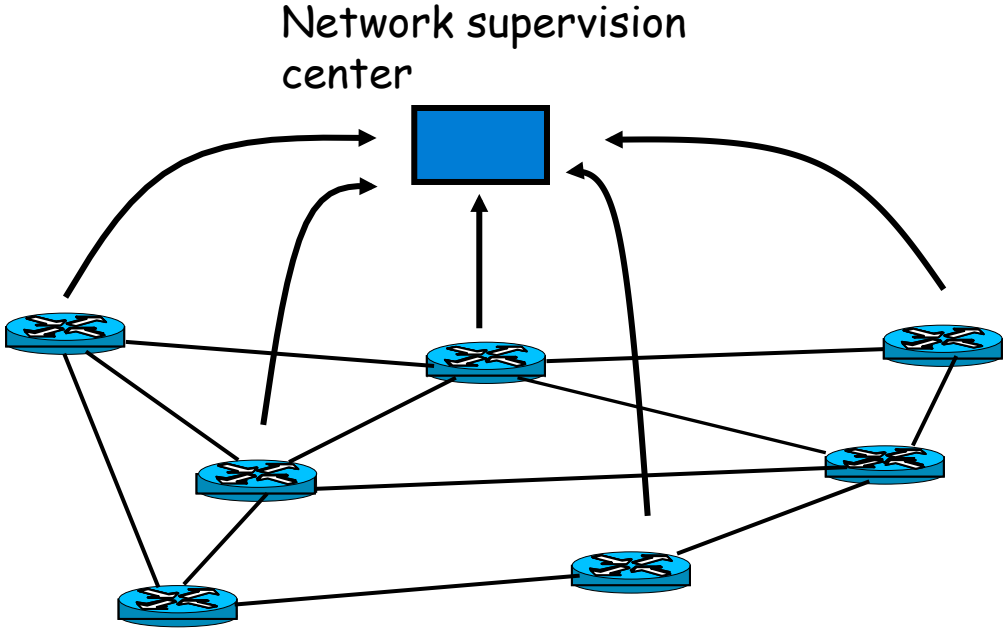
- **Supervision of a computer network**
- **Improvement of network configuration (hardware, software, architecture)**
- **Detection of attacks**
- **Measurements made on routers**

Huge volume of data

High rate of arrivals

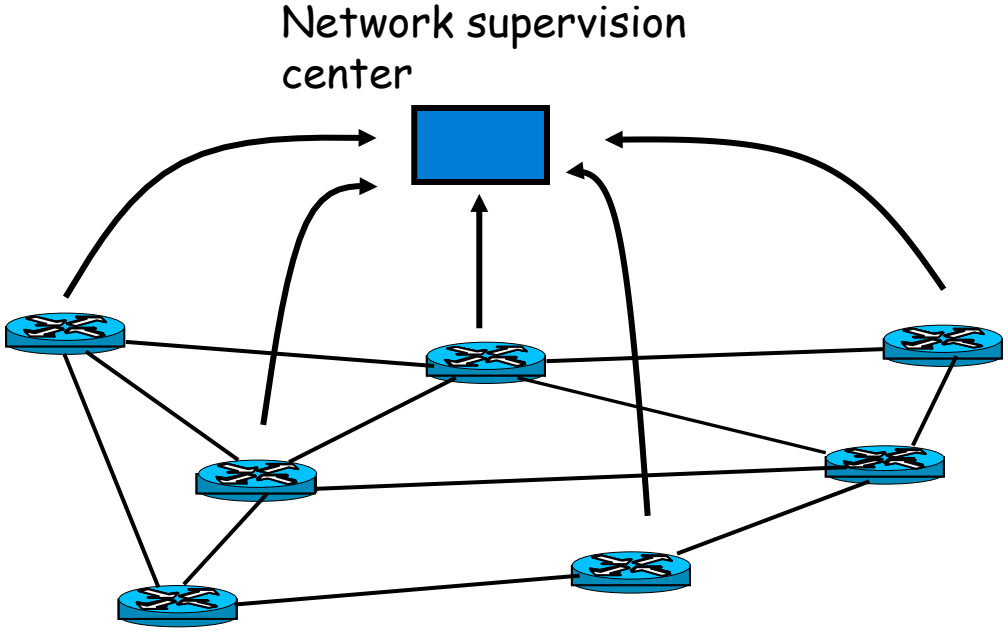


Network management



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



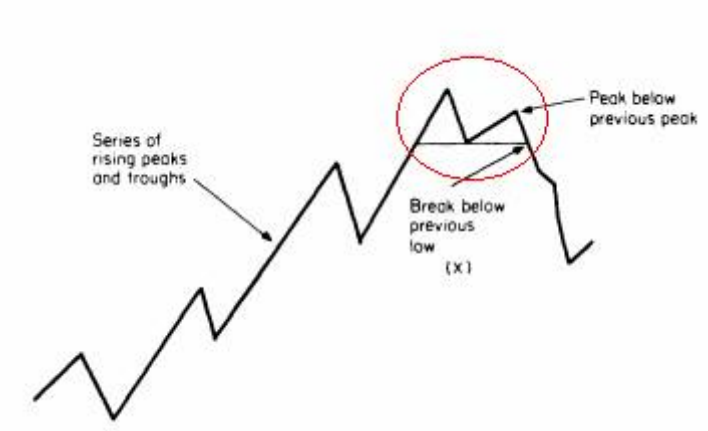
Typical queries:

- 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**

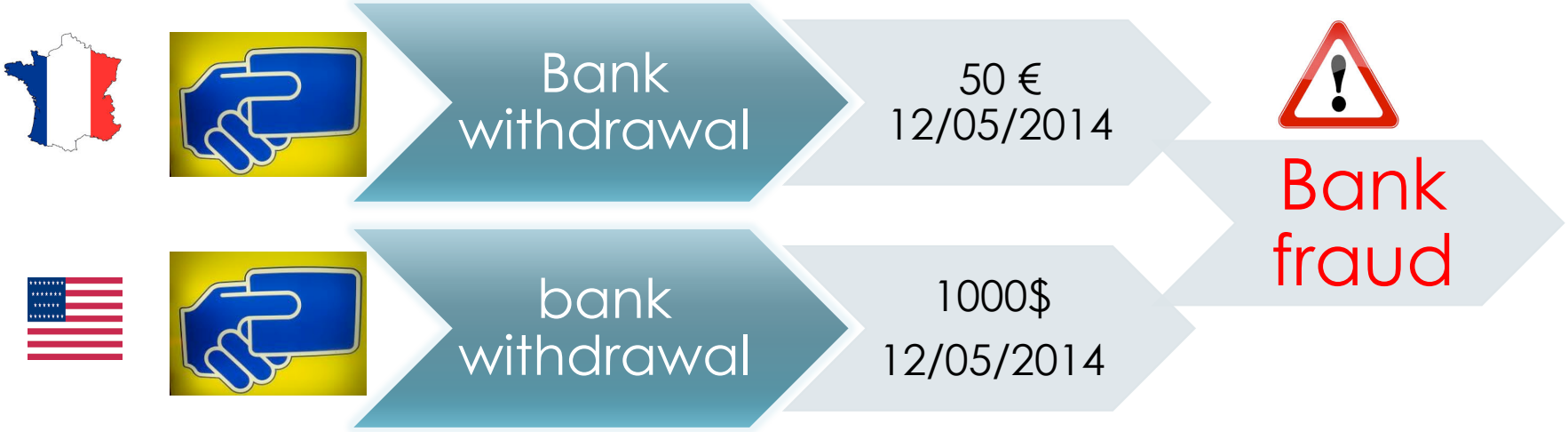
- *Notify me when the price of IBM is above $83, and the first MSFT price afterwards is below $27.*
- *Notify me when some stock goes up by at least 5% from one transaction to the next.*
- *Notify me when the price of any stock increases monotonically for ≥30 min.*
- *Notify me when the difference between 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:**

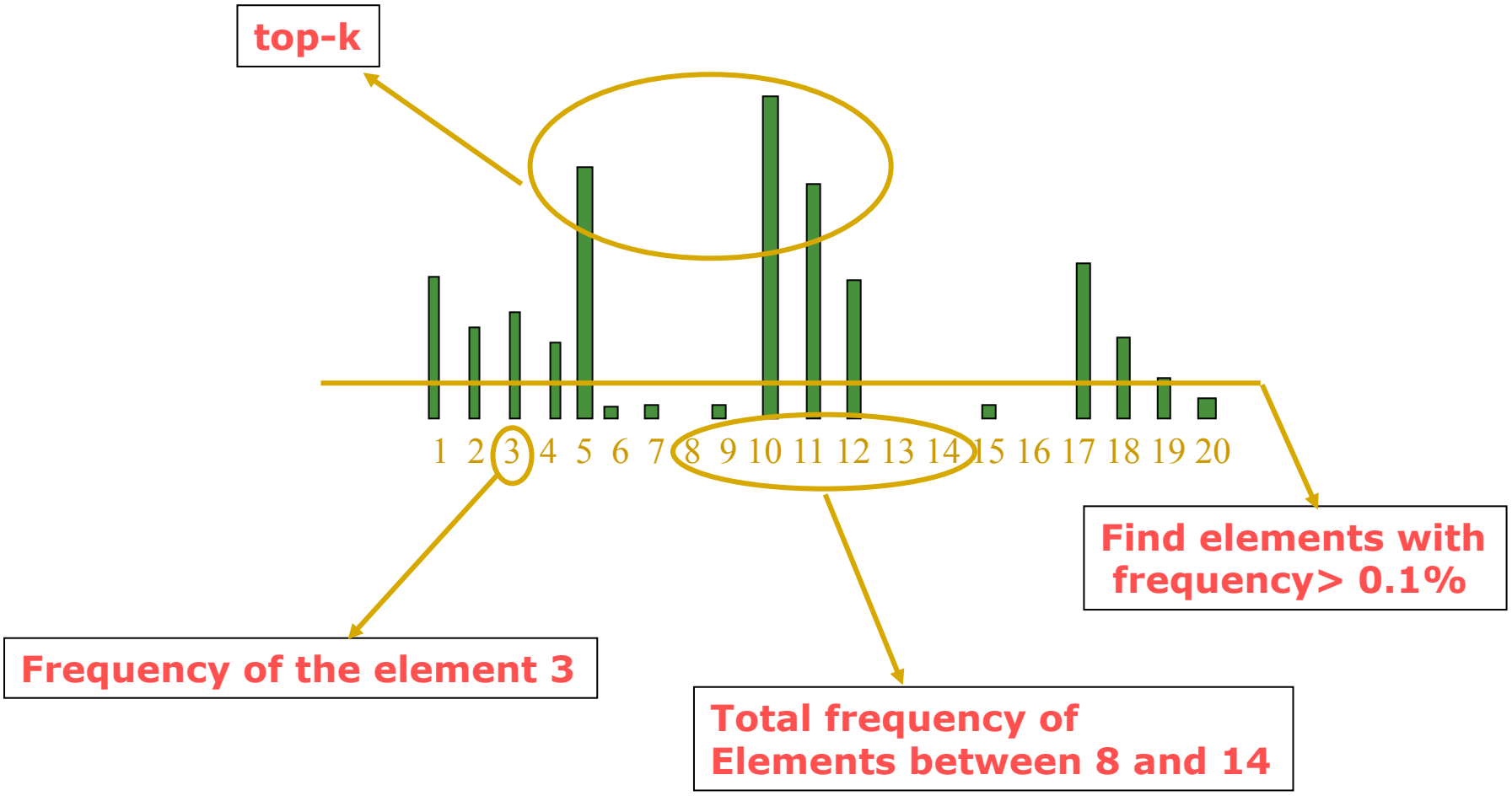


  - Join between several streams
  - Join between stream data and customer database
- Generic tools for processing streams
- Avoid the ‘Store’, ‘Compute’, ‘Delete’ approach
- Solution: incremental computation and definition of temporal windows for joins

Where is the problem? (2/2)

- **Example:**
  - *100 most frequent @S IP adresses 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

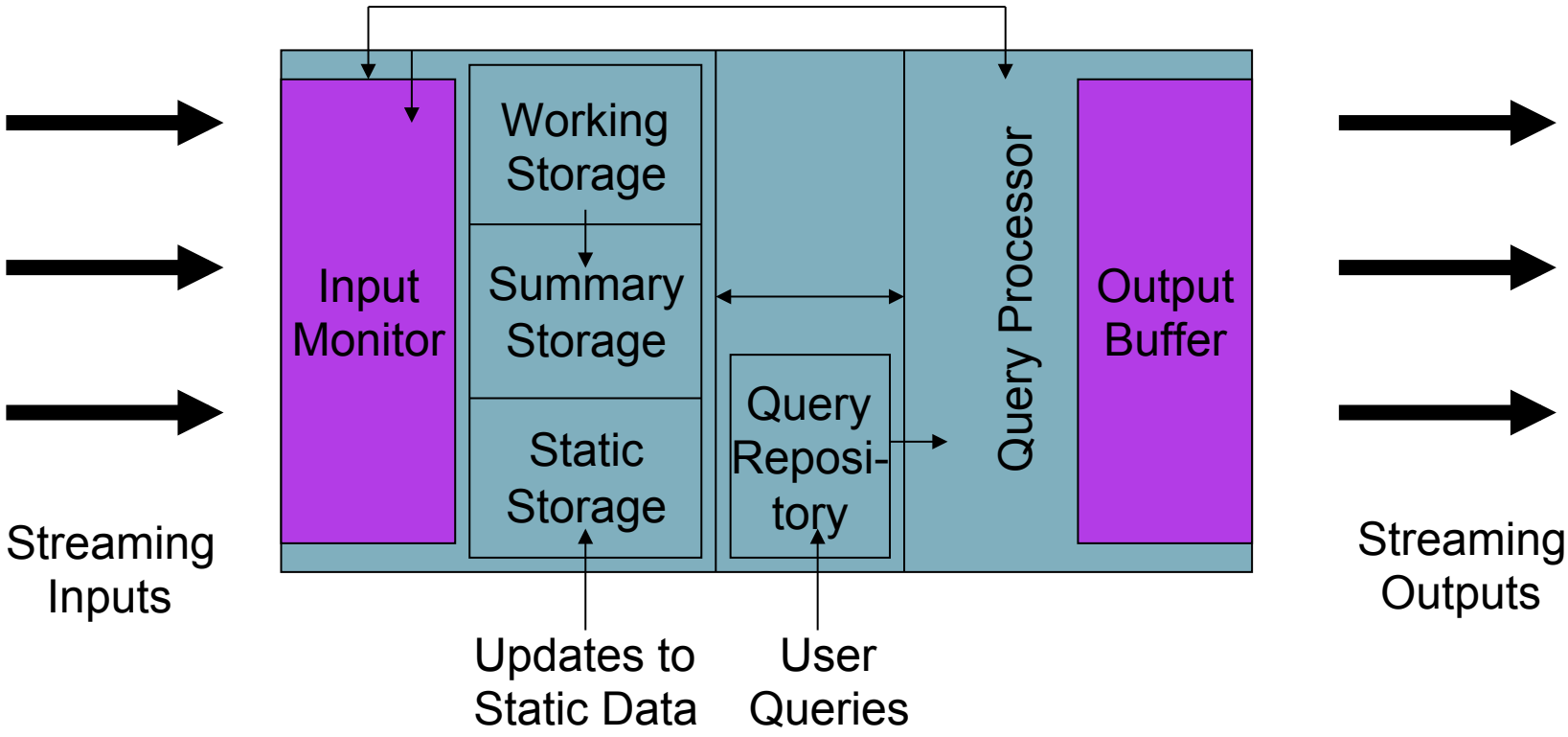
Examples of queries



Data Stream Management Systems

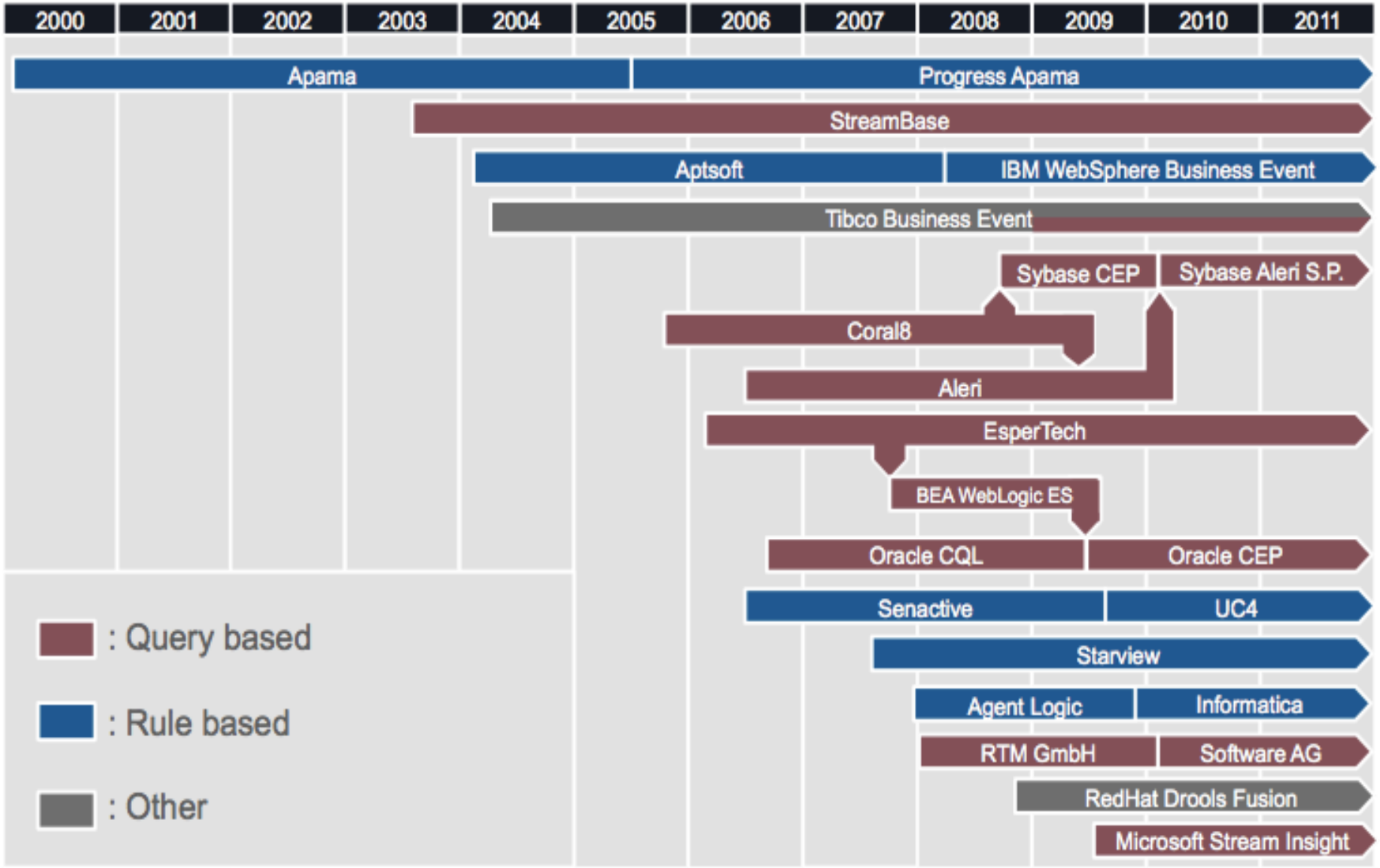
	DBMS	DSMS
Data model	Permanent updatable relations	Streams and permanent updatable relations
Storage	Data is stored on disk	Permanent relations are stored on disk Streams are processed on the fly
Query	SQL language Creating structures Inserting/updating/deleting data Retrieving data (one-time query)	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

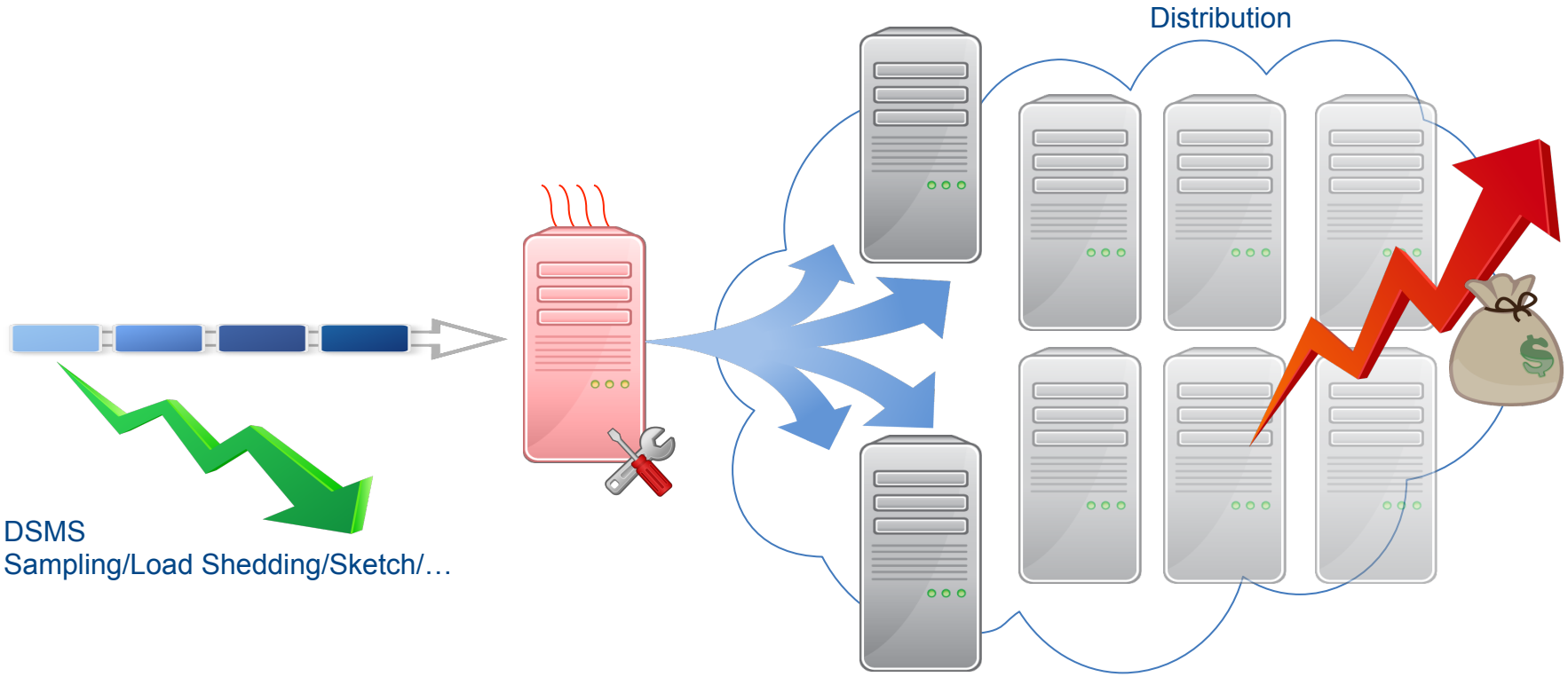


Golab & Oszu (2003)

Existing DSMS



How to deal with Big Data Streams



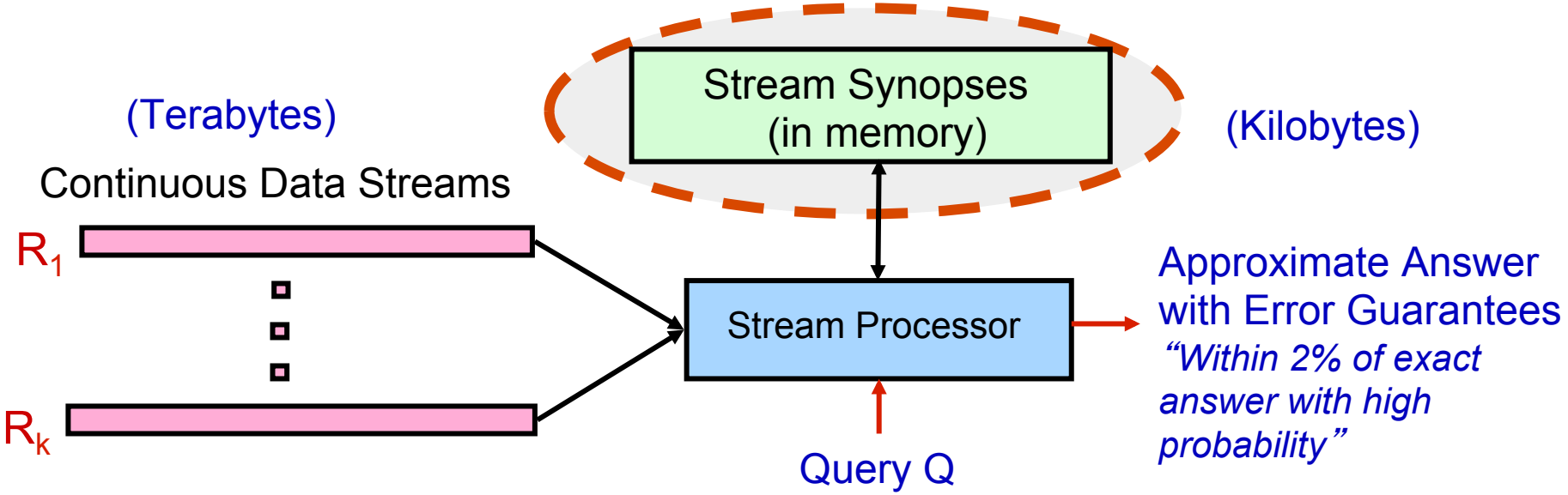
Approximate answers to queries

- **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

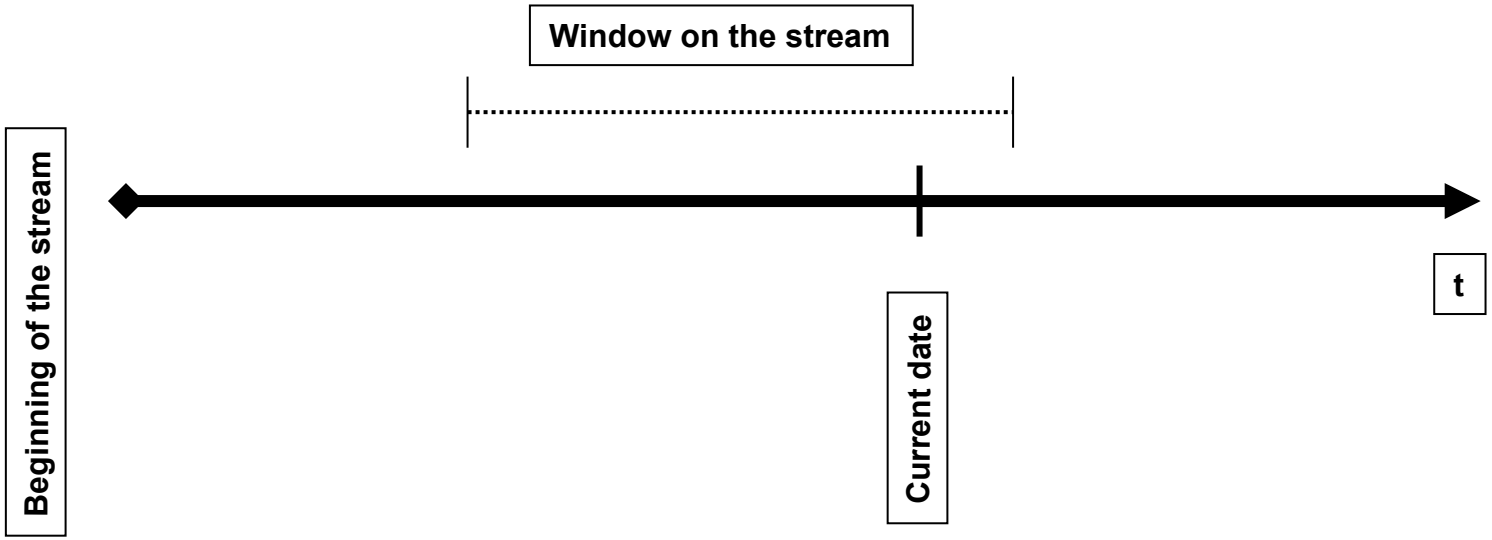
Data-Stream Algorithmics Model



- **Approximate answers– e.g. trend analysis, anomaly detection**
- **Requirements for stream synopses**
- **Single Pass: Each record is examined at most once**
- **Small Space: Log or polylog in data stream size**
- **Small-time: Low per-record processing time (maintain synopses)**

Windowing

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



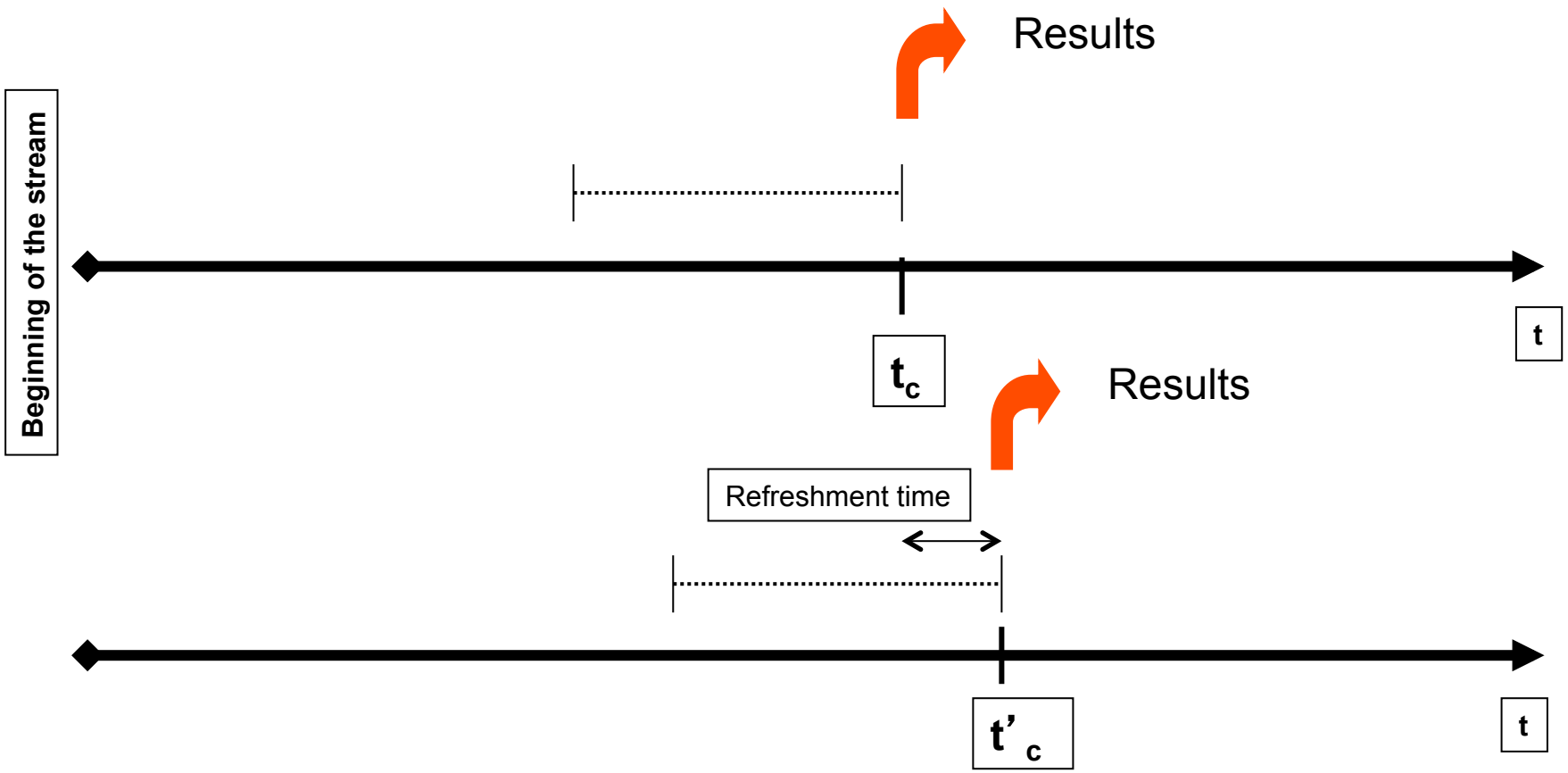
Windowing

- **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

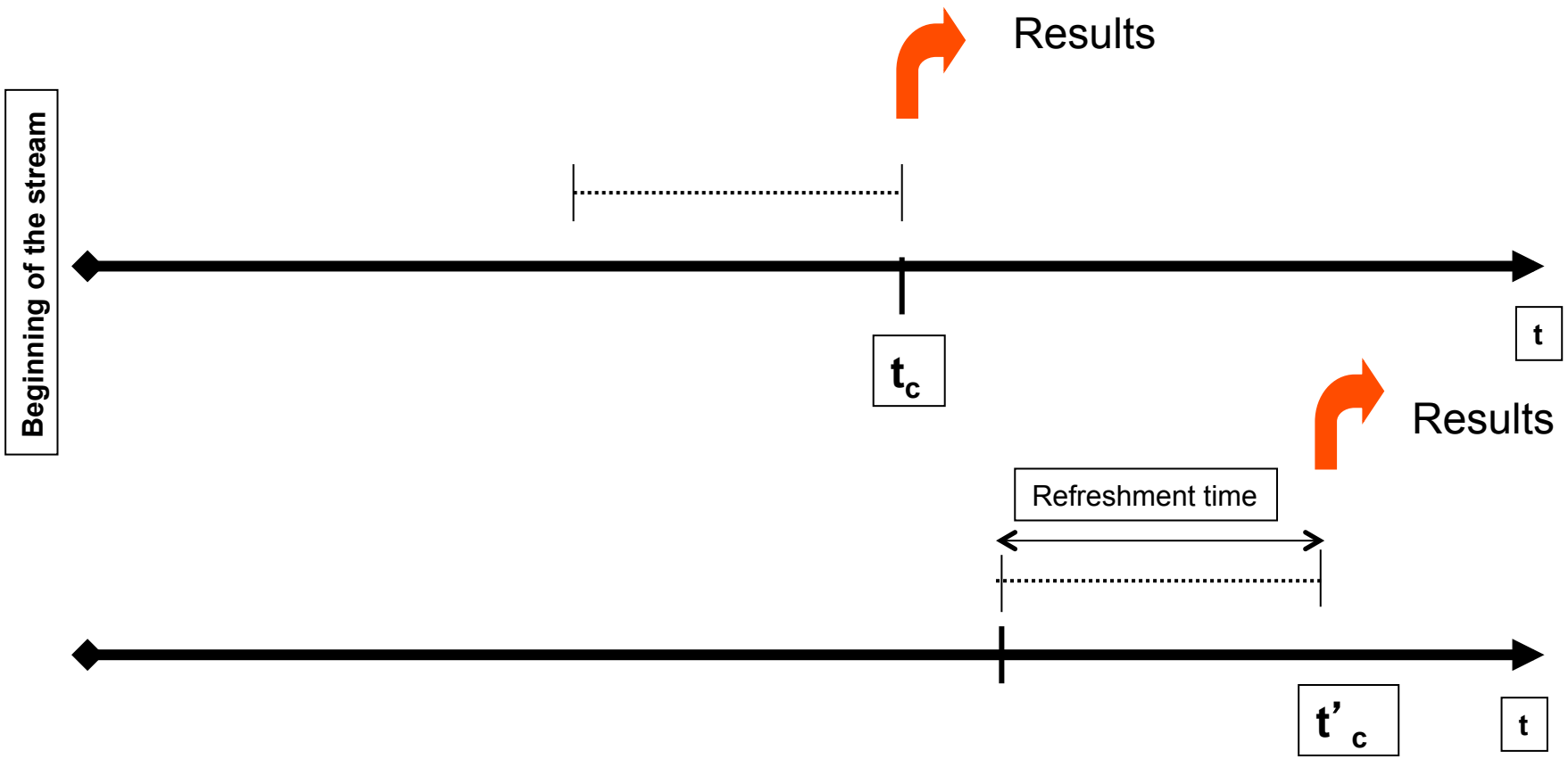
- **Refreshing rate**
  - Rate of producing results (every item, every 10 items, every minute, …)

Sliding window



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

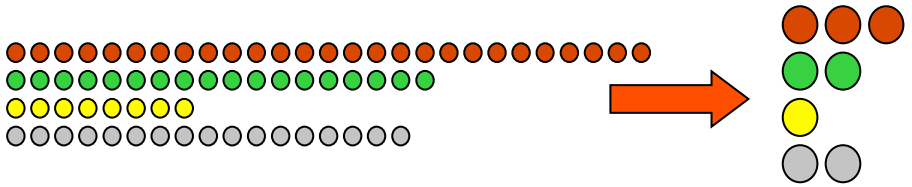
Sliding window vs. Tumbling window



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 >> 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?



The general sampling problem

- **Let h(x) be a hash function that maps x to an integer**
- **Define an identifier field for each tuple**
- **Define the proportion of tuples to keep= n/k**
- **For each tuple x**
  - Compute h(x) (in k buckets)
  - If (h(x)<=n) keep x in the sample

- **What if we want to vary the sample size?**

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.

A simple, Unsatisfying Approach

- Choose a random subset X={x1, …,xk}, X⊂{0,1,…,n-1}
- The sample always consists of the non-expired elements whose indexes are equal to x1, …,xk (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

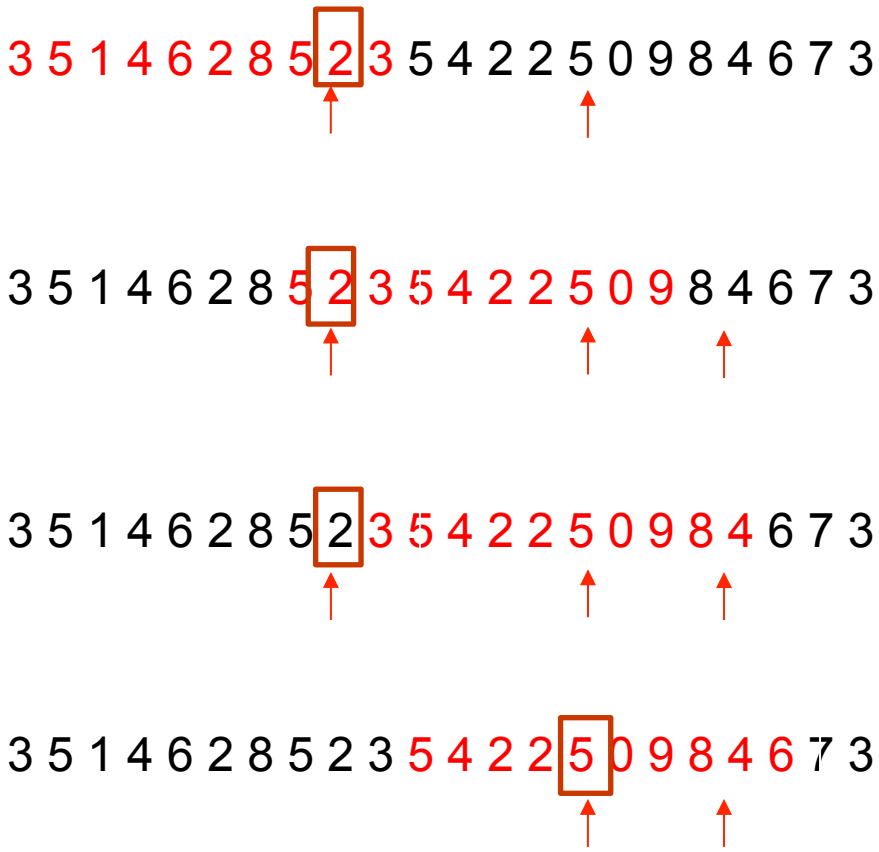
Reservoir sampling

- **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 ith 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 *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…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 choses 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

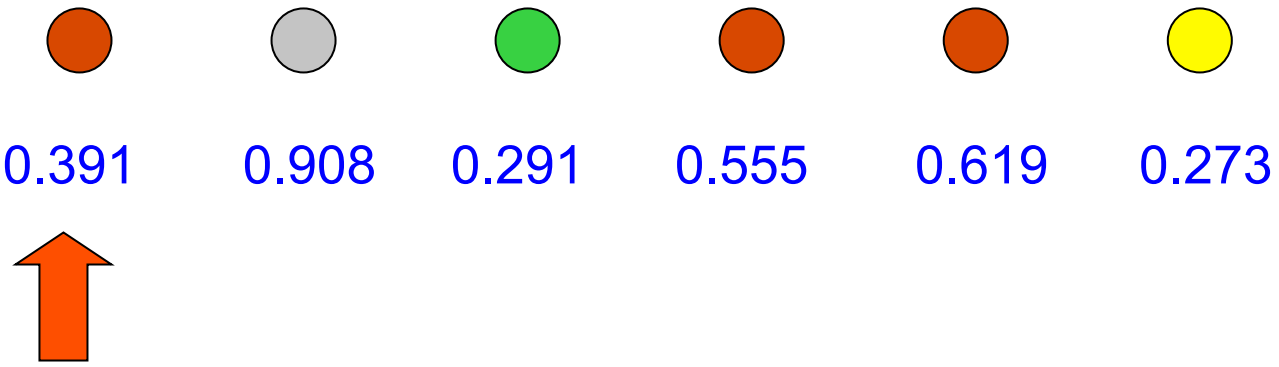


Another Simple Approach: oversample

- 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

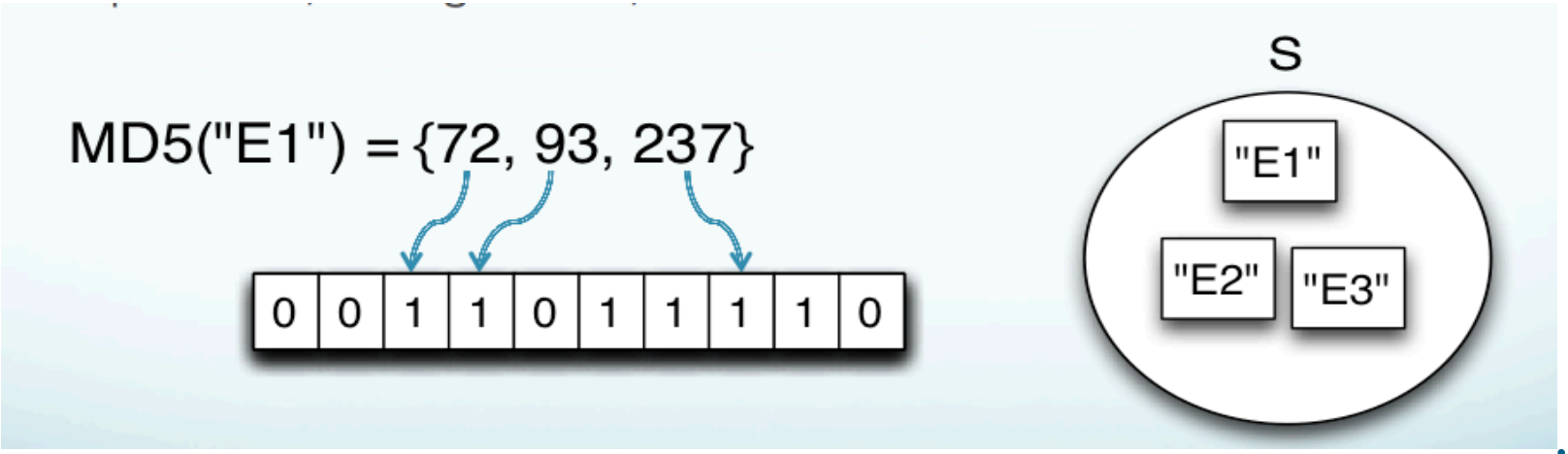
Hash function - Revision

- **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

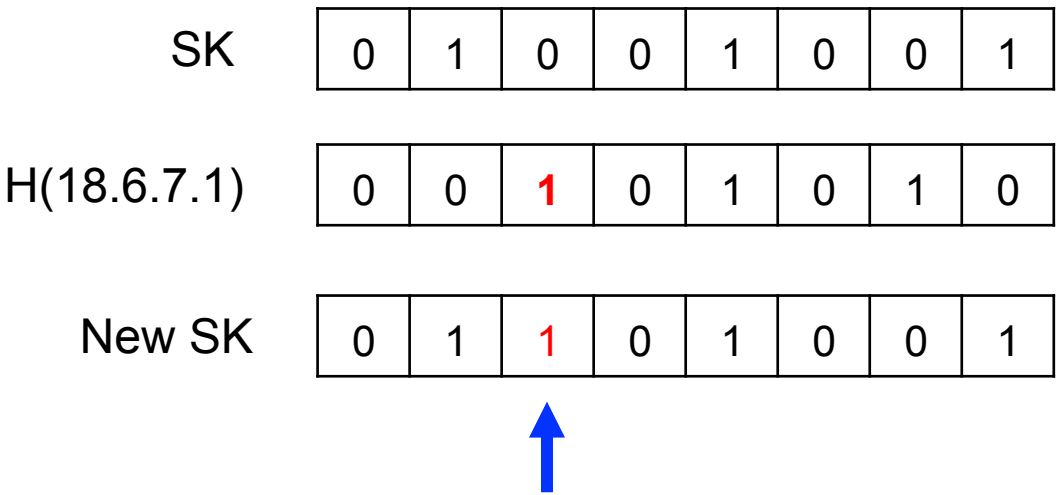
18.6.7.1 →

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

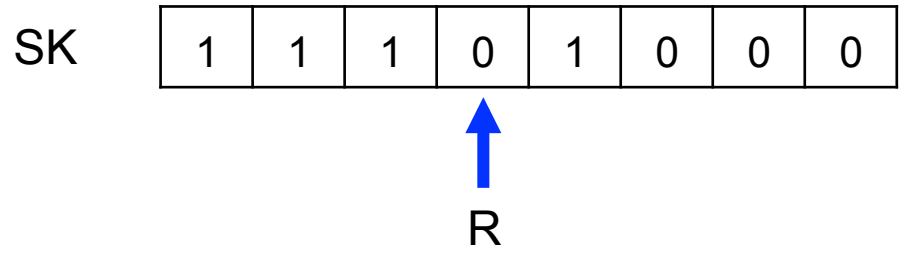


Sketches

- **Result**
  - Select the position R (0…L-1) of the leftmost 0 in SK
  - E(R) = log2 (φ*N) with φ = 0.77351…

➔ Estimate the count by **2^R/φ**

  - σ(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 b	Hash Function 1 $M[\cdot]$	Hash Function 2 b	Hash Function 2 $M[\cdot]$	Hash Function 3 b	Hash Function 3 $M[\cdot]$
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**

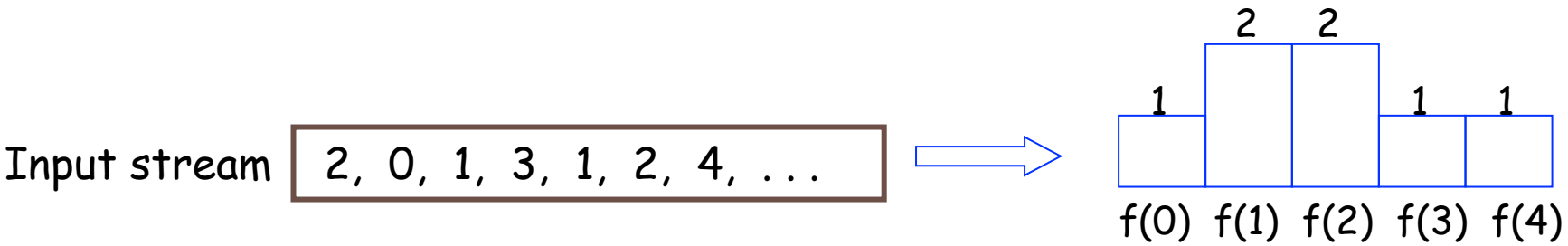
- **Hash functions:**
  - h(x)=(2x+1)mod32
  - h(x)=(3x+7)mod32
  - h(x)=(4x)mod32

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

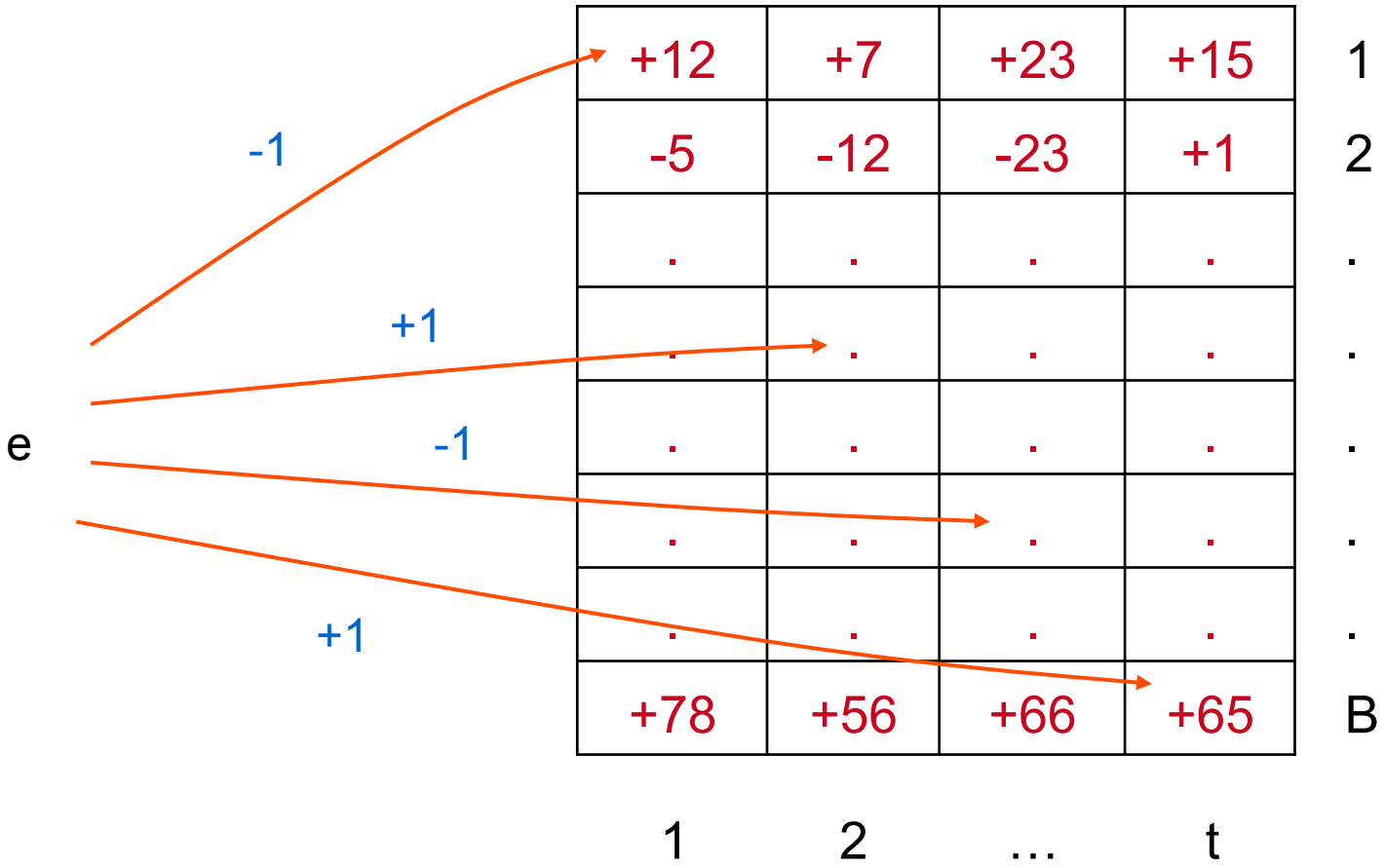
  - Tail length of each element,
  - Estimate the number of elements

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, … , N-1] to [0, 1, … , B]
  - s : hash function from [0, … , N-1] to {+1, -1}
  - Array of B counters: $C_1, \ldots, C_B$ (with B << 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:

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

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

Sketches

- **Algorithm**
  - Maintenance of a list (e_1, e_2, …, 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

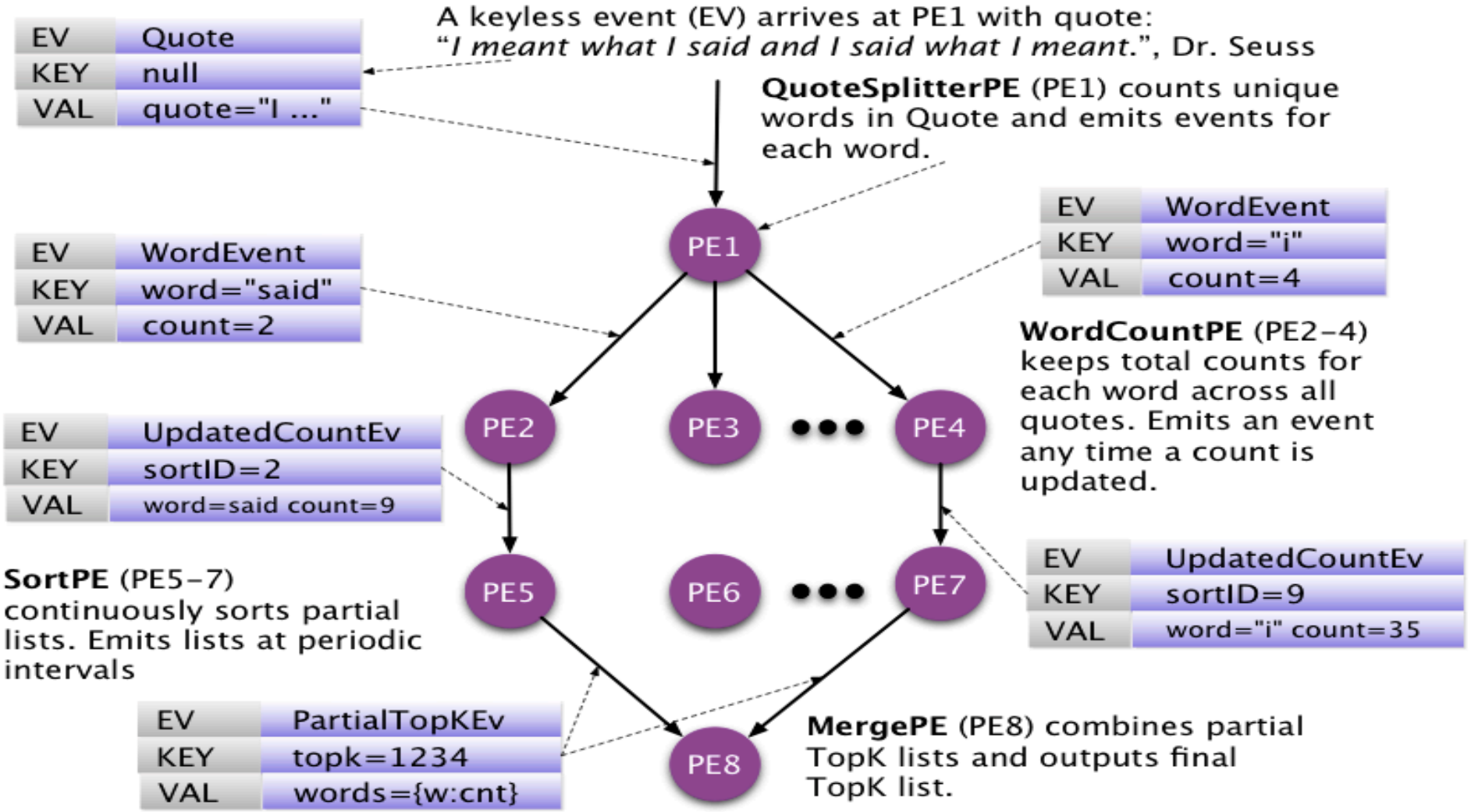
Answering today Big Data Needs

S4, Storm, SAMOA, …

Yahoo S4

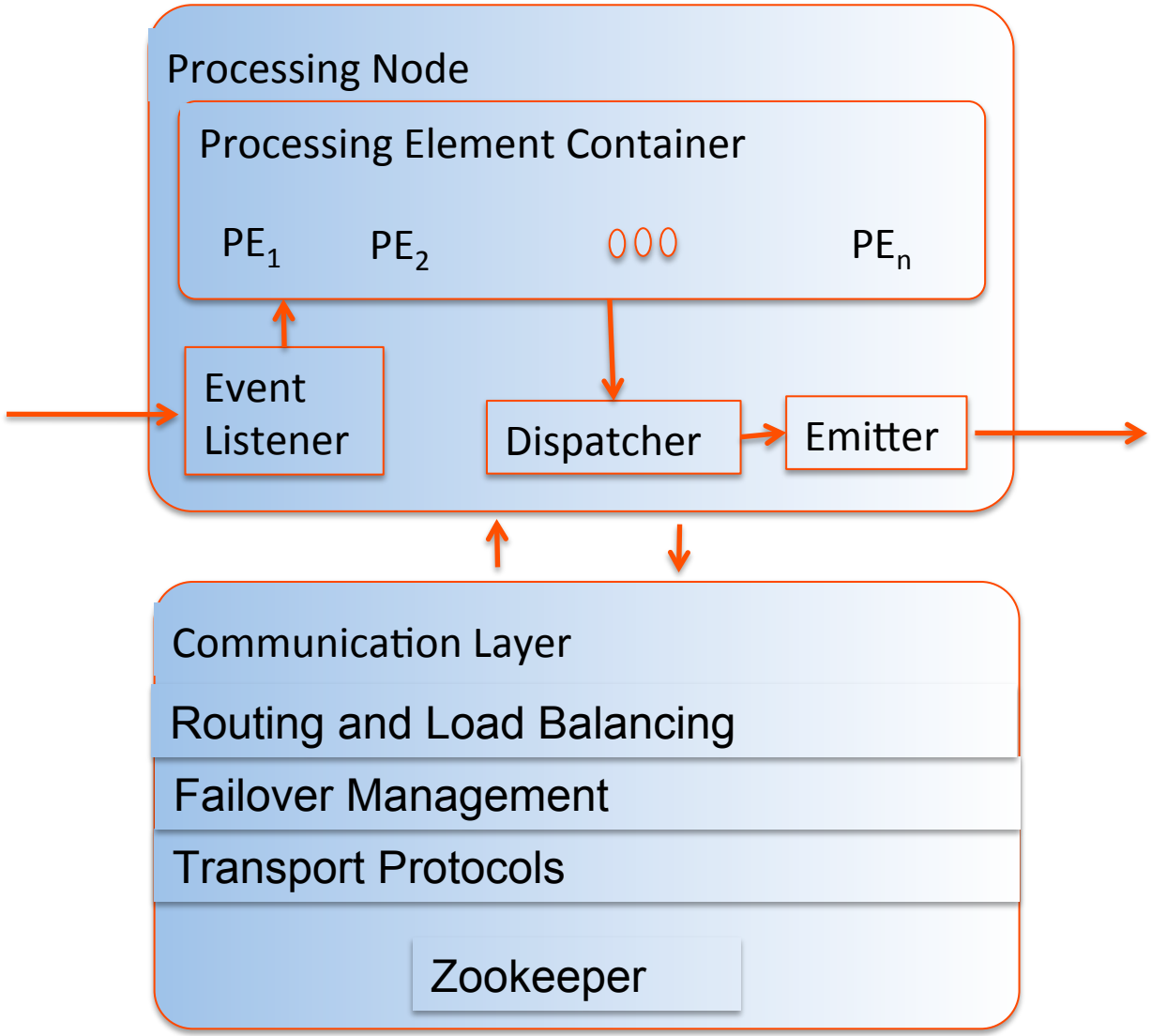
- **Simple Scalable Streaming System**
  - Use a decentralized and symmetric architecture
    - All nodes are similar, no master node
- **Based on Processing Elements**
- **Each PE has 4 components:**
  1. Functionality defined by a PE class and associated configuration
     - input event handler *processEvent*()
     - 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 atomatically deployed on the Processing Nodes (machines):**
  - Ensures load balancing of events,
  - Notifies appropriate PEs when an event comes in.



PE ID	PE Name	Key Tuple
PE1	QuoteSplitterPE	null
PE2	WordCountPE	word="said"
PE4	WordCountPE	word="i"
PE5	SortPE	sortID=2
PE7	SortPE	sortID=9
PE8	MergePE	topK=1234

S4 architecture

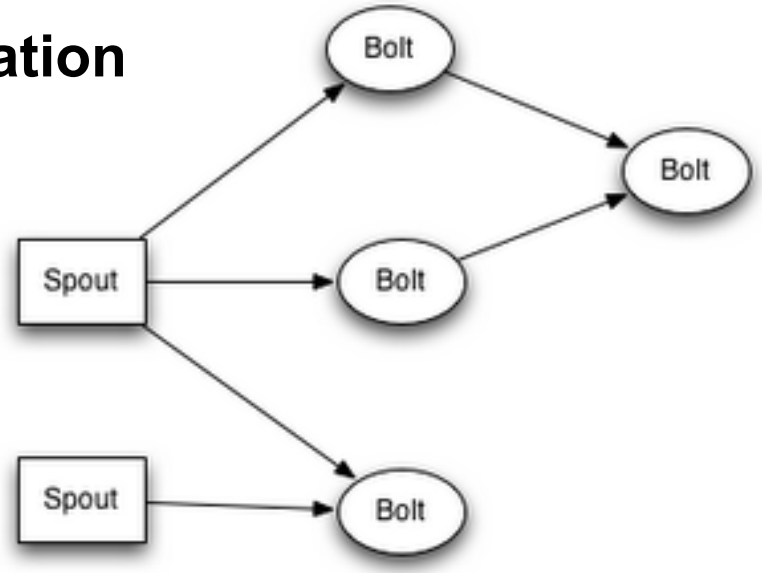


Source: http://fr.slideshare.net/alekbr/s4-stream-computing-platform

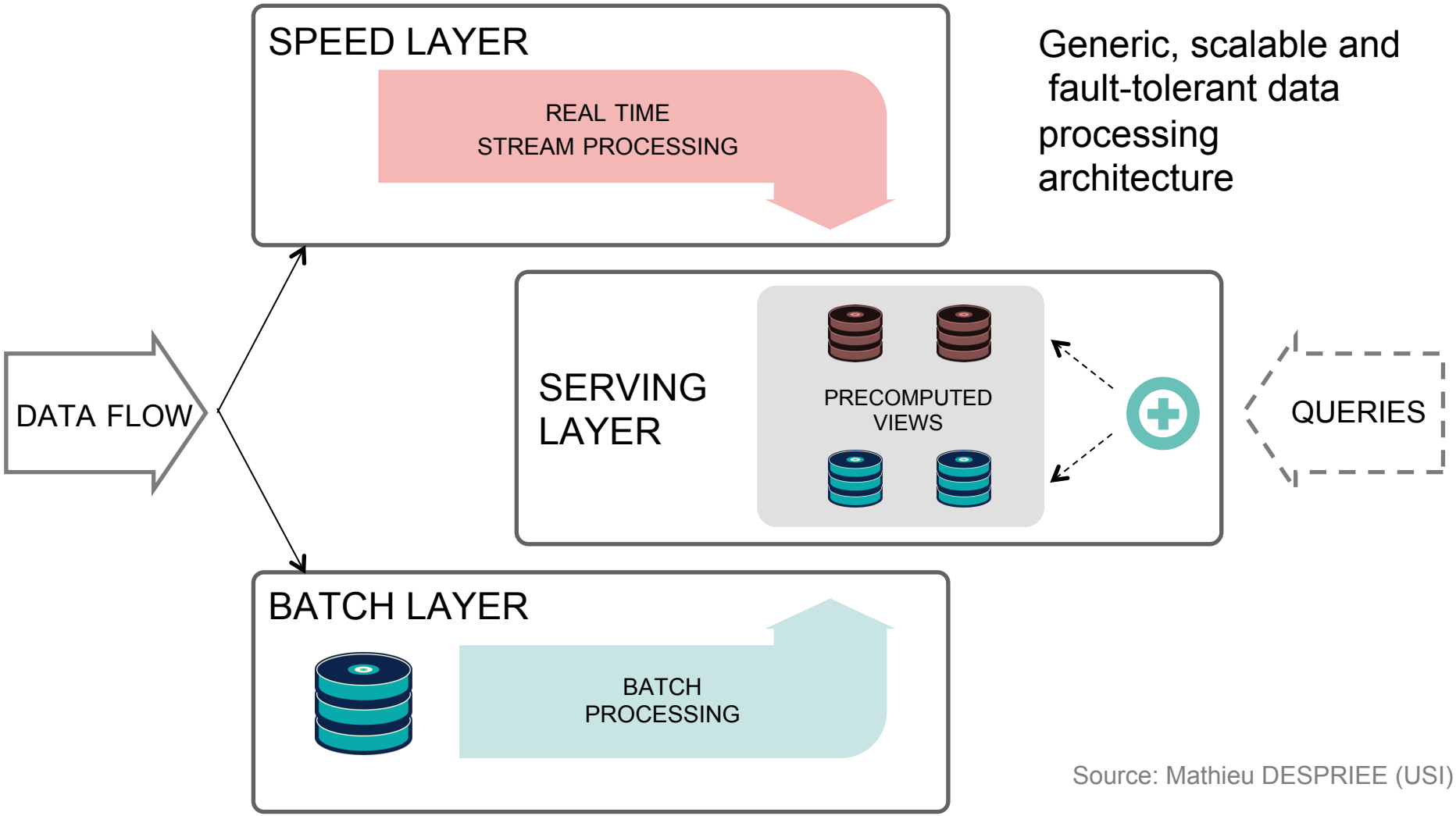
Twitter Storm

- **Same princple 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)



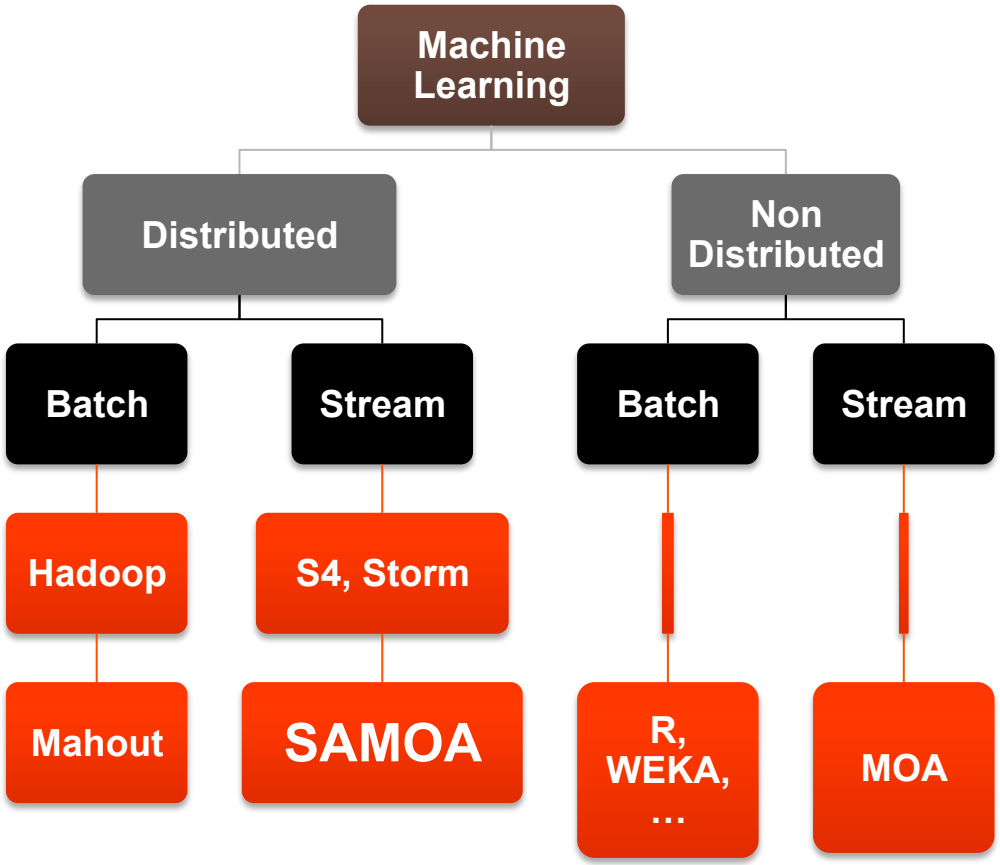
Generic, scalable and fault-tolerant data processing architecture

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

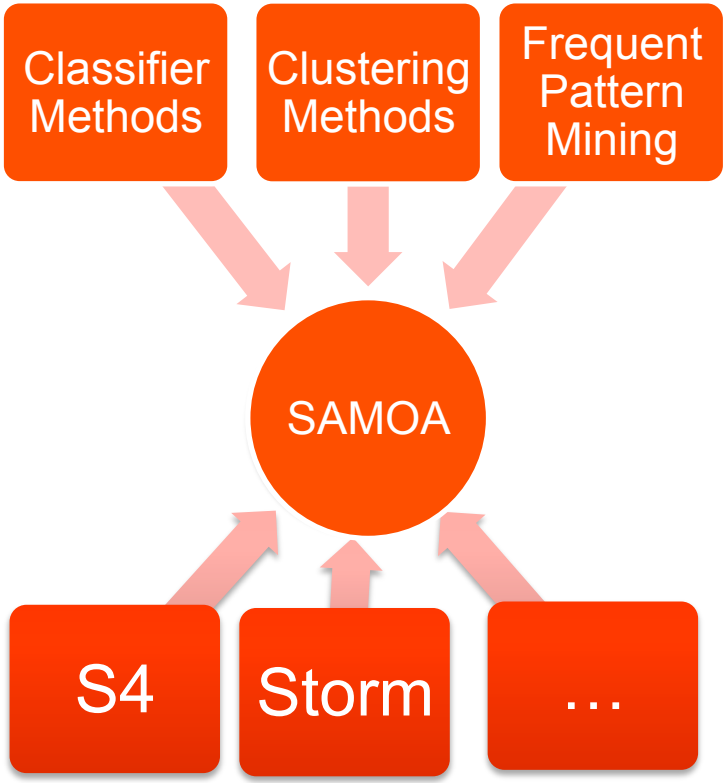


What is SAMOA?

- **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



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

TELECOM Evolution

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

isep

Data Reduction Techniques

- Aggregation: approximations e.g., mean or median
- Load Shedding: drop random tuples
- Sampling: only consider samples from the stream (e.g., random selection). Used in sensor networks.
- Sketches: summaries of stream that occupy small amount of memory, e.g., randomized sketching
- Wavelets: hierachical decomposition
- Histograms: approximate frequency of element values in stream