POLITECNICO DI MILANO

Mining Data Streams

Data Mining and Text Mining (UIC 583 @ Politecnico di Milano)

Prof. Pier Luca Lanzi (Spring 2009)

Lecture outline

- What is stream data?
- Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

- Reference: Jiawei Han and Micheline Kamber, "Data Mining: Concepts and Techniques", The Morgan Kaufmann Series in Data Management Systems (Second Edition)
  Chapter 8, part 1

Data Streams

Characteristics of Data Streams

- Data Streams vs DBMS
  - Data streams
    continuous, ordered, changing, fast, huge amount
  - Traditional DBMS
    data stored in finite, persistent data sets
- Characteristics
  - Huge volumes of continuous data, possibly infinite
  - Fast changing and requires fast, real-time response
  - Data stream captures nicely our data processing needs of today
  - Random access is expensive
    single scan algorithm (can only have one look)
  - Store only the summary of the data seen thus far
  - Most stream data are at pretty low-level or multi-dimensional in nature, needs multi-level and multi-dimensional processing

What are the Applications?

- Telecommunication calling records
- Business: credit card transaction flows
- Network monitoring and traffic engineering
- Financial market: stock exchange
- Engineering & industrial processes: power supply & manufacturing
- Sensor, monitoring & surveillance: video streams, RFIDs
- Security monitoring
- Web logs and Web page click streams
- Massive data sets (even saved but random access is too expensive)

DBMS versus DSMS

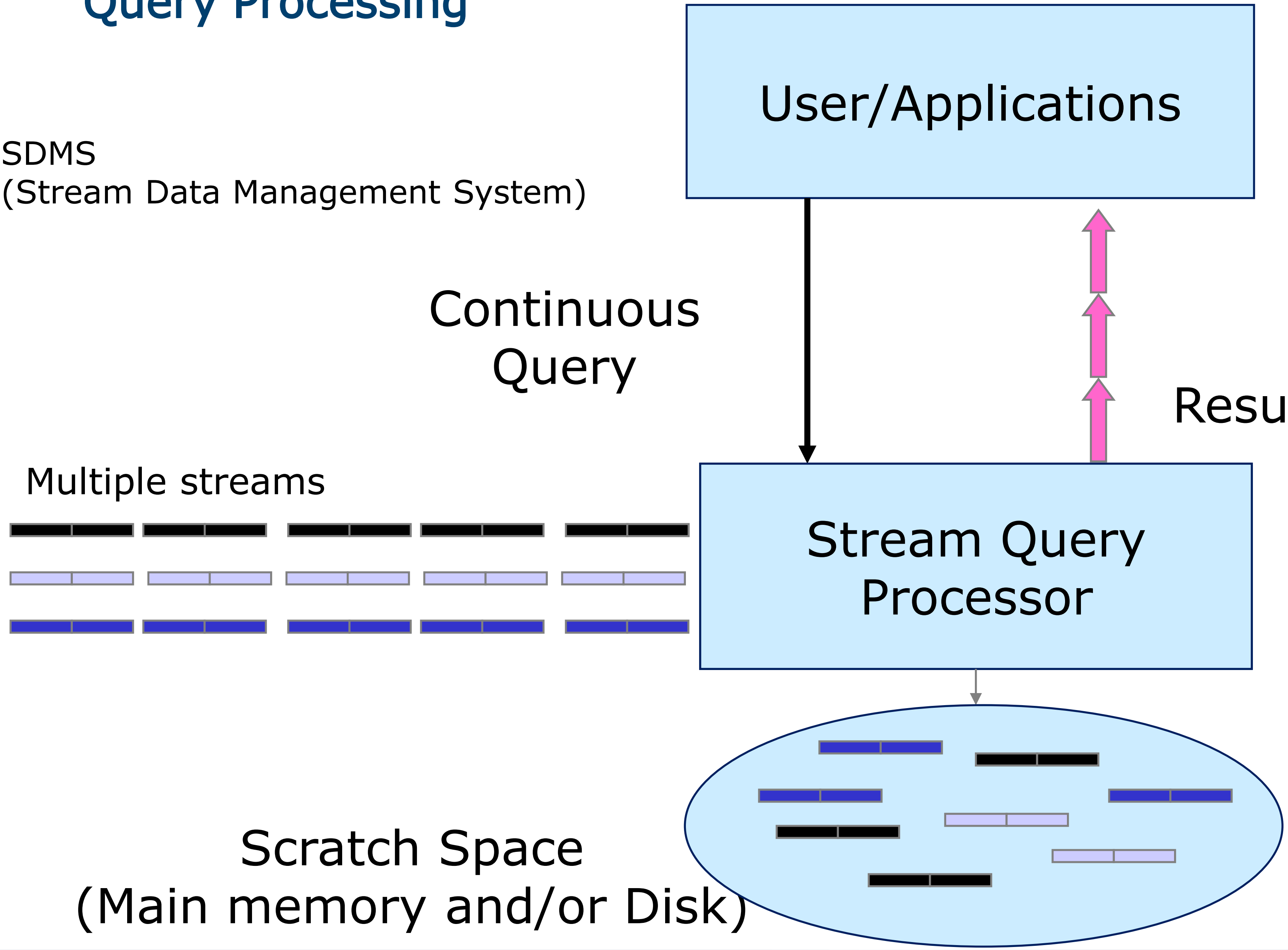
- Persistent relations
- One-time queries
- Random access
- "Unbounded" disk store
- Only current state matters
- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

- Transient streams
- Continuous queries
- Sequential access
- Bounded main memory
- Historical data is important
- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics

Ack. From Motwani's PODS tutorial slides

The Architecture of Stream Query Processing

SDMS
(Stream Data Management System)



What are Challenges of Stream Data Processing?

- Multiple, continuous, rapid, time-varying, ordered streams
- Main memory computations
- Queries are often continuous
  - Evaluated continuously as stream data arrives
  - Answer updated over time
- Queries are often complex
  - Beyond element-at-a-time processing
  - Beyond stream-at-a-time processing
  - Beyond relational queries (scientific, data mining, OLAP)
- Multi-level/multi-dimensional processing and data mining
  - Most stream data are at low-level or multi-dimensional in nature

Processing Stream Queries

- Query types
  - One-time query vs. continuous query (being evaluated continuously as stream continues to arrive)
  - Predefined query vs. ad-hoc query (issued on-line)
- Unbounded memory requirements
  - For real-time response, main memory algorithm should be used
  - Memory requirement is unbounded if one will join future tuples
- Approximate query answering
  - With bounded memory, it is not always possible to produce exact answers
  - High-quality approximate answers are desired
  - Data reduction and synopsis construction methods: Sketches, random sampling, histograms, wavelets, etc.

What the Methodologies for Stream Data Processing?

- Major challenges
  - Keep track of a large universe, e.g., pairs of IP address, not ages
- Methodology
  - Synopses (trade-off between accuracy and storage)
  - Use synopsis data structure, much smaller ($O(\log^k N)$ space) than their base data set ($O(N)$ space)
  - Compute an approximate answer within a small error range (factor ε of the actual answer)
- Major methods
  - Random sampling
  - Histograms
  - Sliding windows
  - Multi-resolution model
  - Sketches
  - Radomized algorithms

Stream Data Processing Methods (1)

- Random sampling (but without knowing the total length in advance)
  - Reservoir sampling: maintain a set of s candidates in the reservoir, which form a true random sample of the element seen so far in the stream. As the data stream flow, every new element has a certain probability (s/N) of replacing an old element in the reservoir.
- Sliding windows
  - Make decisions based only on recent data of sliding window size w
  - An element arriving at time t expires at time t + w
- Histograms
  - Approximate the frequency distribution of element values in a stream
  - Partition data into a set of contiguous buckets
  - Equal-width (equal value range for buckets) vs. V-optimal (minimizing frequency variance within each bucket)
- Multi-resolution models
  - Popular models: balanced binary trees, micro-clusters, and wavelets

Approximate Query Answering in Streams

- Sliding windows
  - Only over sliding windows of recent stream data
  - Approximation but often more desirable in applications
- Batched processing, sampling and synopses
  - Batched if update is fast but computing is slow
    - Compute periodically, not very timely
  - Sampling if update is slow but computing is fast
    - Compute using sample data, but not good for joins, etc.
  - Synopsis data structures
    - Maintain a small synopsis or sketch of data
    - Good for querying historical data
- Blocking operators, e.g., sorting, avg, min, etc.
  - Blocking if unable to produce the first output until seeing the entire input

Projects on DSMS (Data Stream Management System)

- Research projects and system prototypes
  - STREAM (Stanford): A general-purpose DSMS
  - Cougar (Cornell): sensors
  - Aurora (Brown/MIT): sensor monitoring, dataflow
  - Hancock (AT&T): telecom streams
  - Niagara (OGI/Wisconsin): Internet XML databases
  - OpenCQ (Georgia Tech): triggers, incr. view maintenance
  - Tapestry (Xerox): pub/sub content-based filtering
  - Telegraph (Berkeley): adaptive engine for sensors
  - Tradebot (www.tradebot.com): stock tickers & streams
  - Tribeca (Bellcore): network monitoring
  - MAIDS (UIUC/NCSA): Mining Alarming Incidents in Data Streams

Stream Data Mining vs. Stream Querying

- Stream mining—A more challenging task in many cases
  - It shares most of the difficulties with stream querying
  - But often requires less "precision", e.g., no join, grouping, sorting
  - Patterns are hidden and more general than querying
  - It may require exploratory analysis, not necessarily continuous queries
- Stream data mining tasks
  - Multi-dimensional on-line analysis of streams
  - Mining outliers and unusual patterns in stream data
  - Clustering data streams
  - Classification of stream data

Challenges for Mining Dynamics in Data Streams

- Most stream data are at pretty low-level or multi-dimensional in nature: needs ML/MD processing

- Analysis requirements
  - Multi-dimensional trends and unusual patterns
  - Capturing important changes at multi-dimensions/levels
  - Fast, real-time detection and response
  - Comparing with data cube: Similarity and differences

- Stream (data) cube or stream OLAP: Is this feasible?
  - Can we implement it efficiently?

Multi-Dimensional Stream Analysis: Examples

- Analysis of Web click streams
  - Raw data at low levels: seconds, web page addresses, user IP addresses, …
  - Analysts want: changes, trends, unusual patterns, at reasonable levels of details
  - E.g., Average clicking traffic in North America on sports in the last 15 minutes is 40% higher than that in the last 24 hours.”
- Analysis of power consumption streams
  - Raw data: power consumption flow for every household, every minute
  - Patterns one may find: average hourly power consumption surges up 30% for manufacturing companies in Chicago in the last 2 hours today than that of the same day a week ago

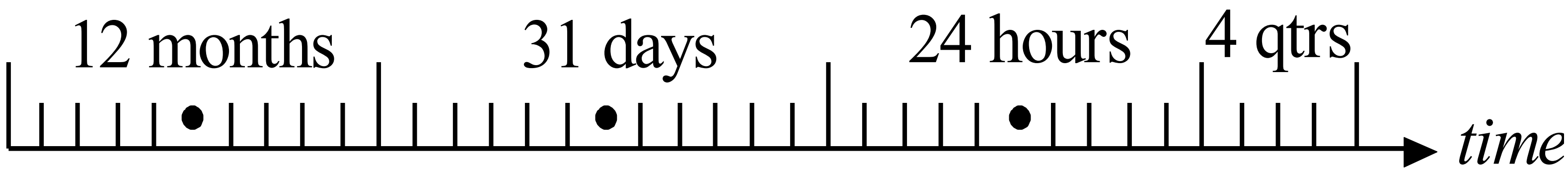
Architectures

A Stream Cube Architecture

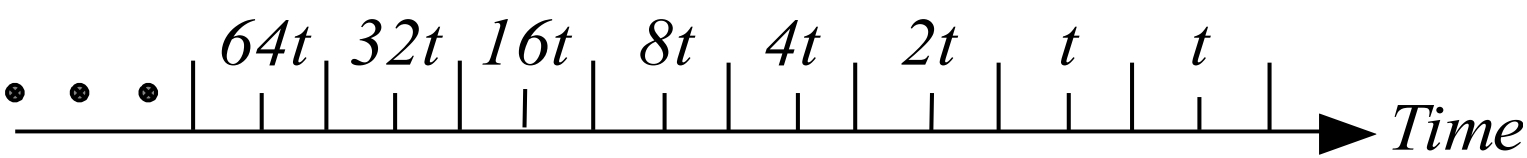
- A tilted time frame
  - Different time granularities:
    second, minute, quarter, hour, day, week, …
- Critical layers
  - Minimum interest layer (m-layer)
  - Observation layer (o-layer)
  - User: watches at o-layer and occasionally needs to drill-down down to m-layer
- Partial materialization of stream cubes
  - Full materialization: too space and time consuming
  - No materialization: slow response at query time
  - Partial materialization: what do we mean "partial"?

A Titled Time Model (1)

- Natural tilted time frame:
  - Example: Minimal: quarter, then 4 quarters → 1 hour, 24 hours → day, …



- Logarithmic tilted time frame:
  - Example: Minimal: 1 minute, then 1, 2, 4, 8, 16, 32, …

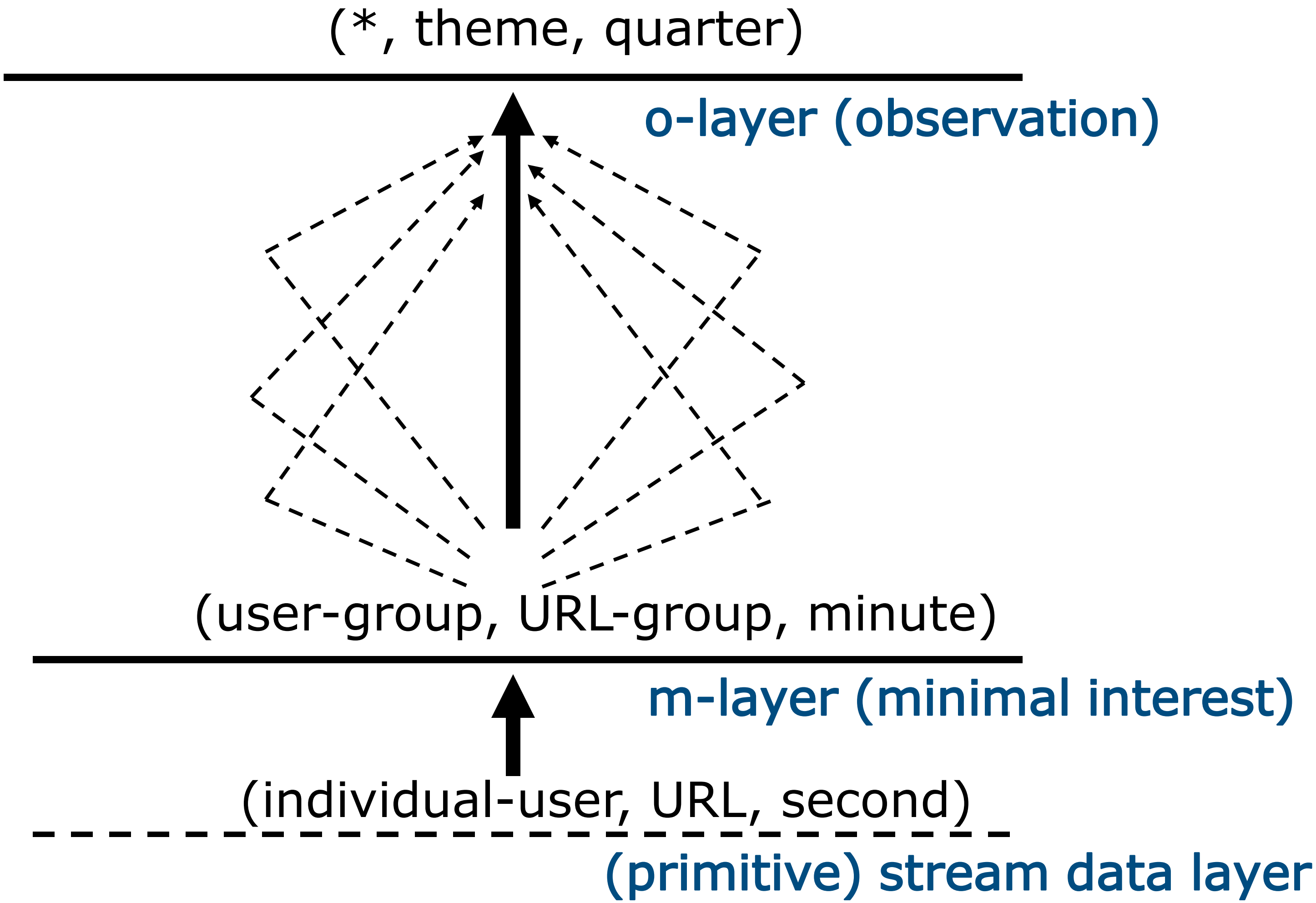


A Titled Time Model (2)

- Pyramidal tilted time frame
  - Example: Suppose there are 5 frames and each takes maximal 3 snapshots
  - Given a snapshot number N, if N mod 2d = 0, insert into the frame number d. If there are more than 3 snapshots, “kick out” the oldest one.

Frame no.	Snapshots (by clock time)
0	69 67 65
1	70 66 62
2	68 60 52
3	56 40 24
4	48 16
5	64 32

Two Critical Layers in the Stream Cube

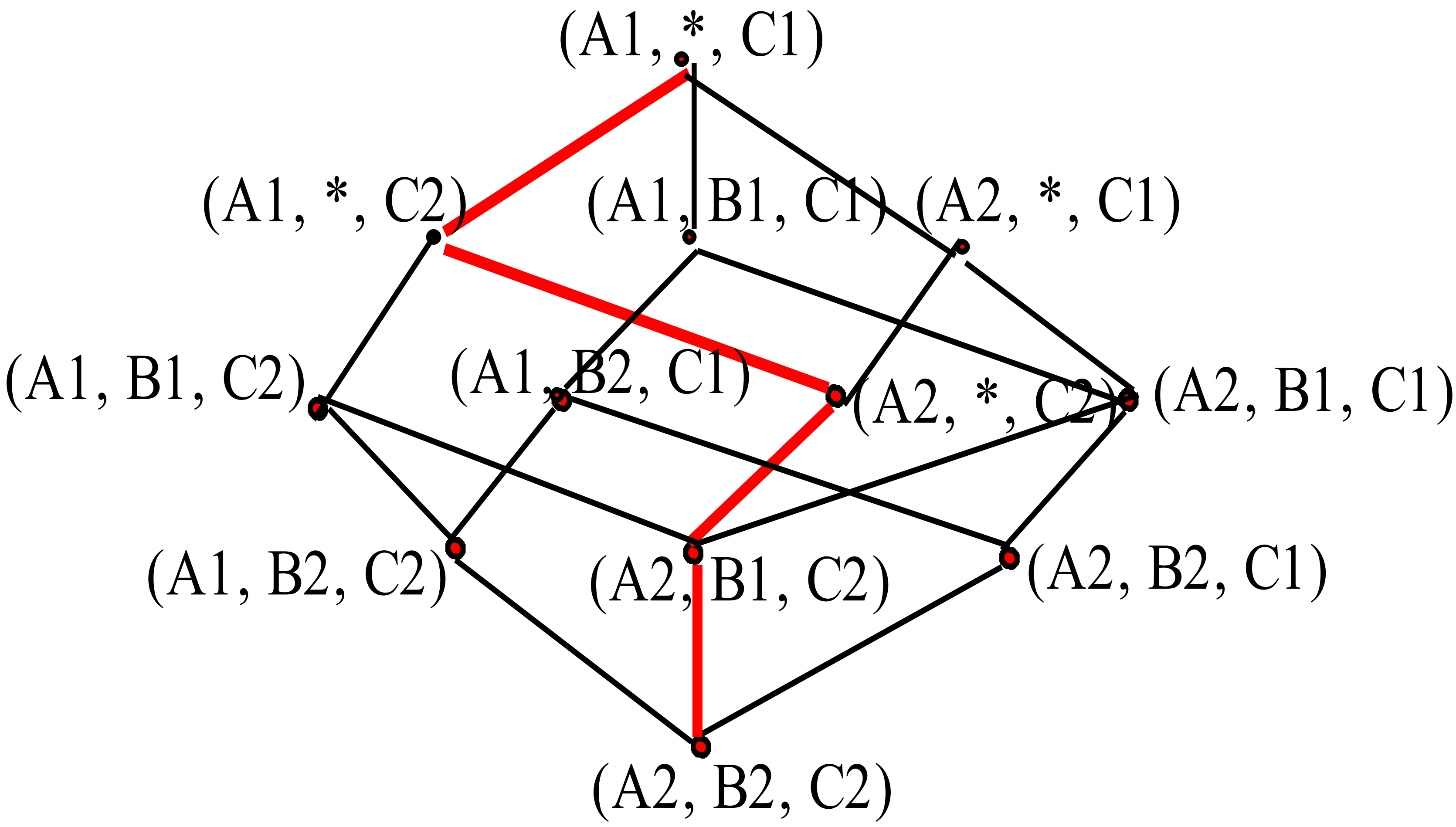


On-Line Partial Materialization vs. OLAP Processing

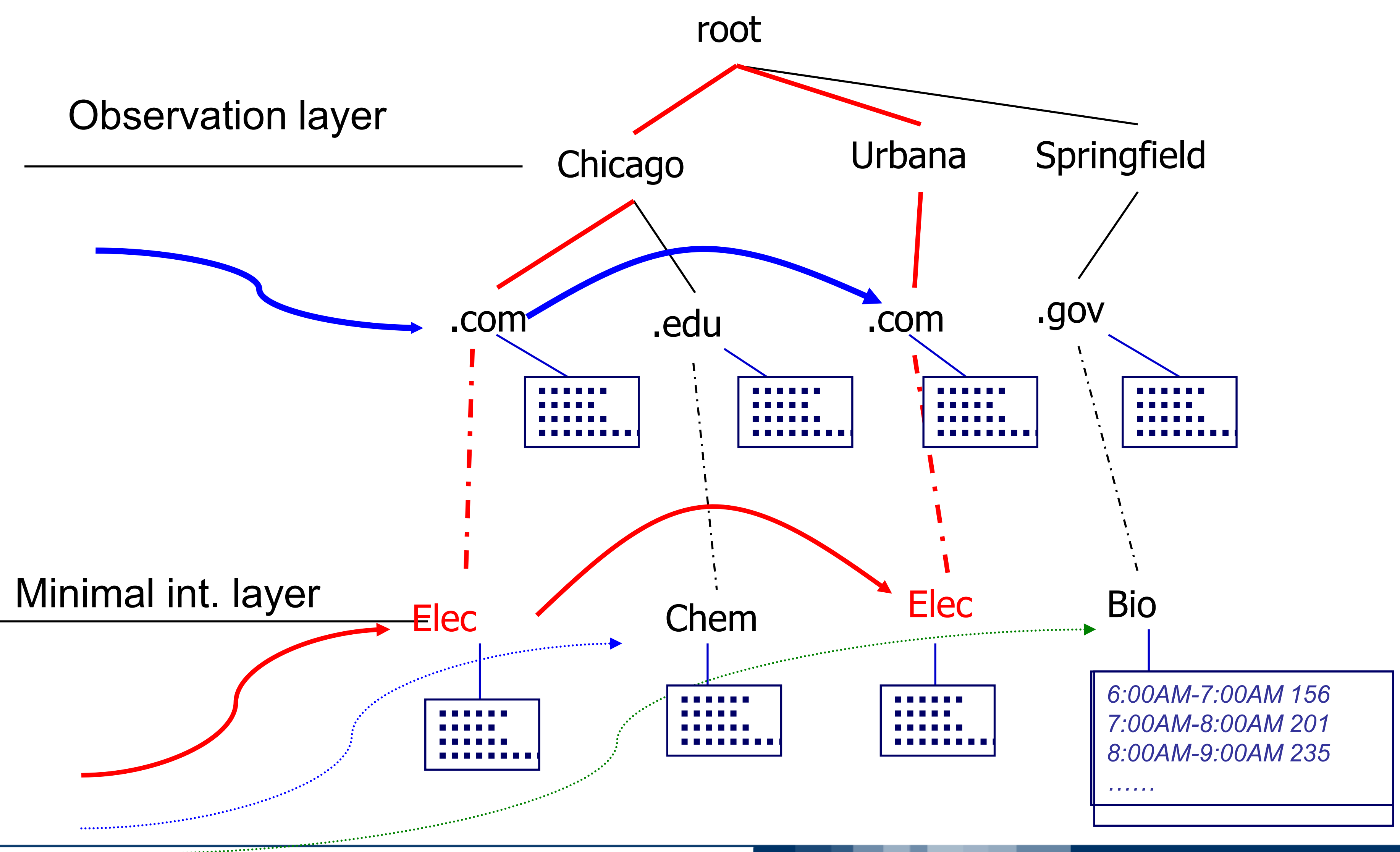
- On-line materialization
  - Materialization takes precious space and time
    - Only incremental materialization (with tilted time frame)
  - Only materialize “cuboids” of the critical layers?
    - Online computation may take too much time
  - Preferred solution:
    - popular-path approach: Materializing those along the popular drilling paths
    - H-tree structure: Such cuboids can be computed and stored efficiently using the H-tree structure

- Online aggregation vs. query-based computation
  - Online computing while streaming: aggregating stream cubes
  - Query-based computation: using computed cuboids

Stream Cube Structure: From m-layer to o-layer



An H-Tree Cubing Structure



Benefits of H-Tree and H-Cubing

- H-tree and H-cubing
  - Developed for computing data cubes and ice-berg cubes
  - Fast cubing, space preserving in cube computation
- Using H-tree for stream cubing
  - Space preserving: intermediate aggregates can be computed incrementally and saved in tree nodes
  - Facilitate computing other cells and multi-dimensional analysis
  - H-tree with computed cells can be viewed as stream cube

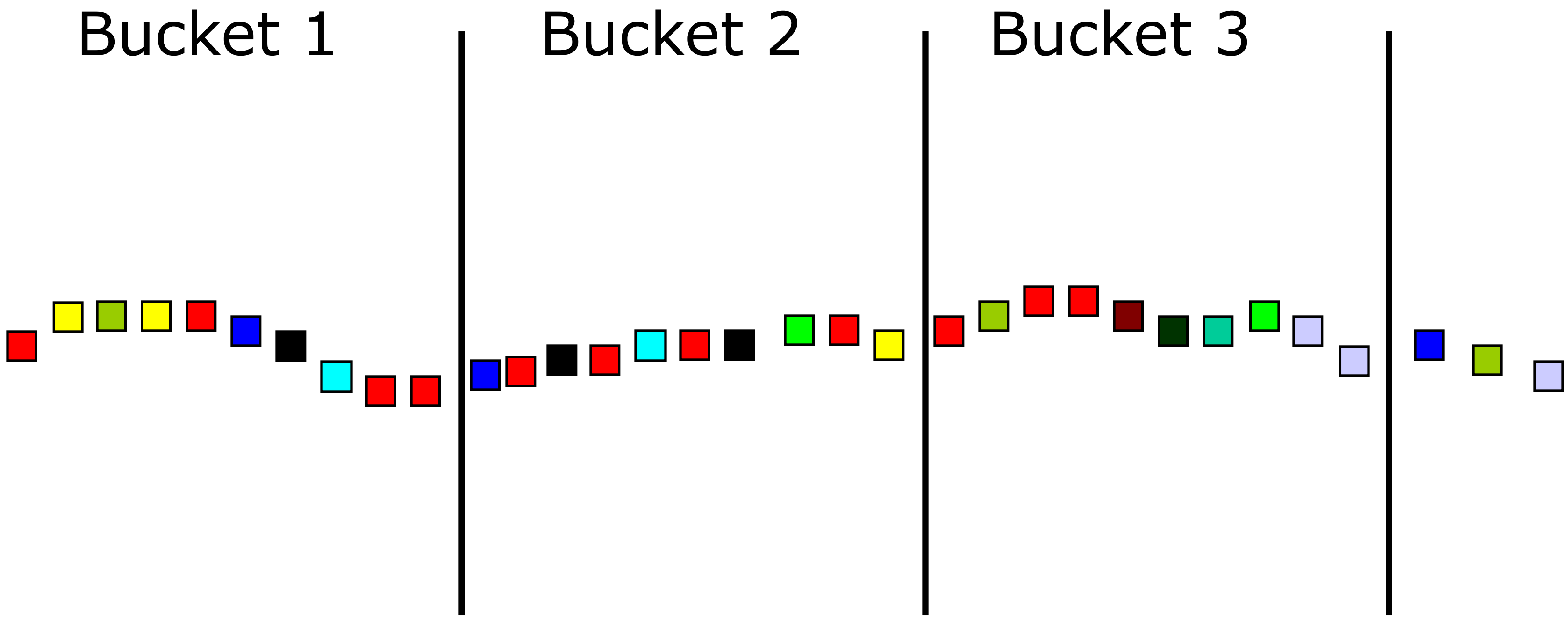
Frequent patterns

Frequent Patterns for Stream Data

- Frequent pattern mining is valuable in stream applications
  - e.g., network intrusion mining
- Mining precise freq. patterns in stream data: unrealistic
  - Even store them in a compressed form, such as FPtree
- How to mine frequent patterns with good approximation?
  - Approximate frequent patterns (Manku & Motwani VLDB'02)
  - Keep only current frequent patterns? No changes can be detected
- Mining evolution freq. patterns (C. Giannella, J. Han, X. Yan, P.S. Yu, 2003)
  - Use tilted time window frame
  - Mining evolution and dramatic changes of frequent patterns
- Space-saving computation of frequent and top-k elements (Metwally, Agrawal, and El Abbadi, ICDT'05)

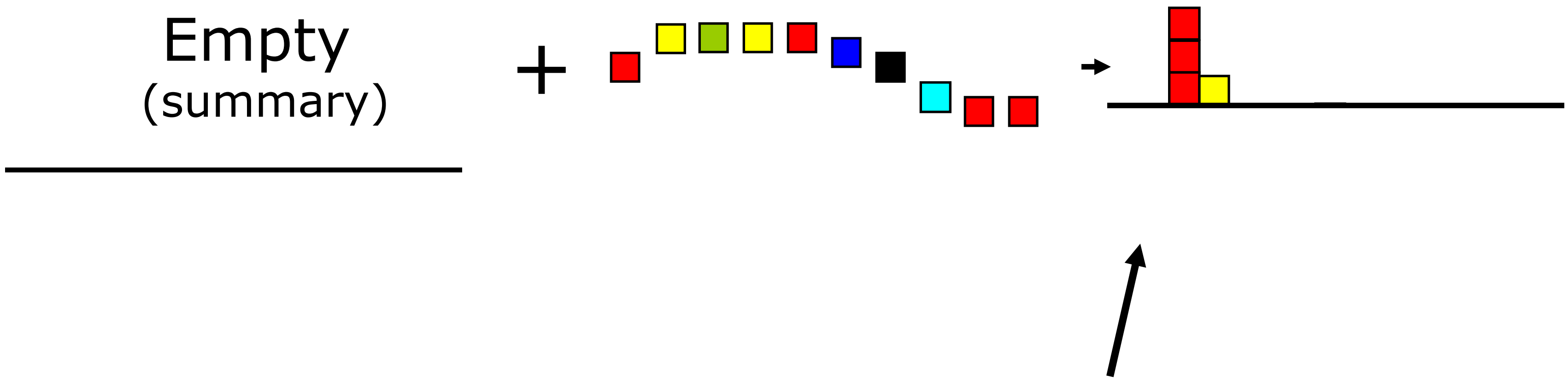
Mining Approximate Frequent Patterns

- Mining precise freq. patterns in stream data: unrealistic
  - Even store them in a compressed form, such as FPtree
- Approximate answers are often sufficient (e.g., trend/pattern analysis)
- Example: a router is interested in all flows:
  - whose frequency is at least 1% (σ) of the entire traffic stream seen so far
  - and feels that 1/10 of σ (ε = 0.1%) error is comfortable
- How to mine frequent patterns with good approximation?
  - Lossy Counting Algorithm (Manku & Motwani, VLDB’02)
  - Major ideas: not tracing items until it becomes frequent
  - Adv: guaranteed error bound
  - Disadv: keep a large set of traces



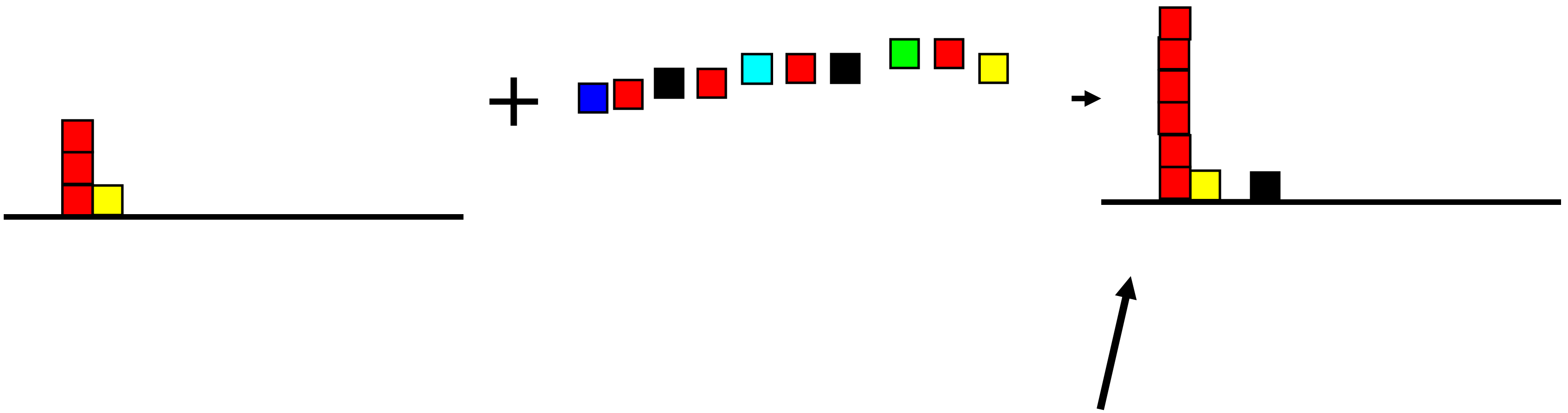
Divide Stream into 'Buckets' (bucket size is $1/\varepsilon = 1000$)

First Bucket of Stream



At bucket boundary, decrease all counters by 1

Next Bucket of Stream



At bucket boundary, decrease all counters by 1

Approximation Guarantee

- Given
  - (1) support threshold: σ,
  - (2) error threshold: ε, and
  - (3) stream length N

- Output: items with frequency counts exceeding $(\sigma - \varepsilon)\,N$

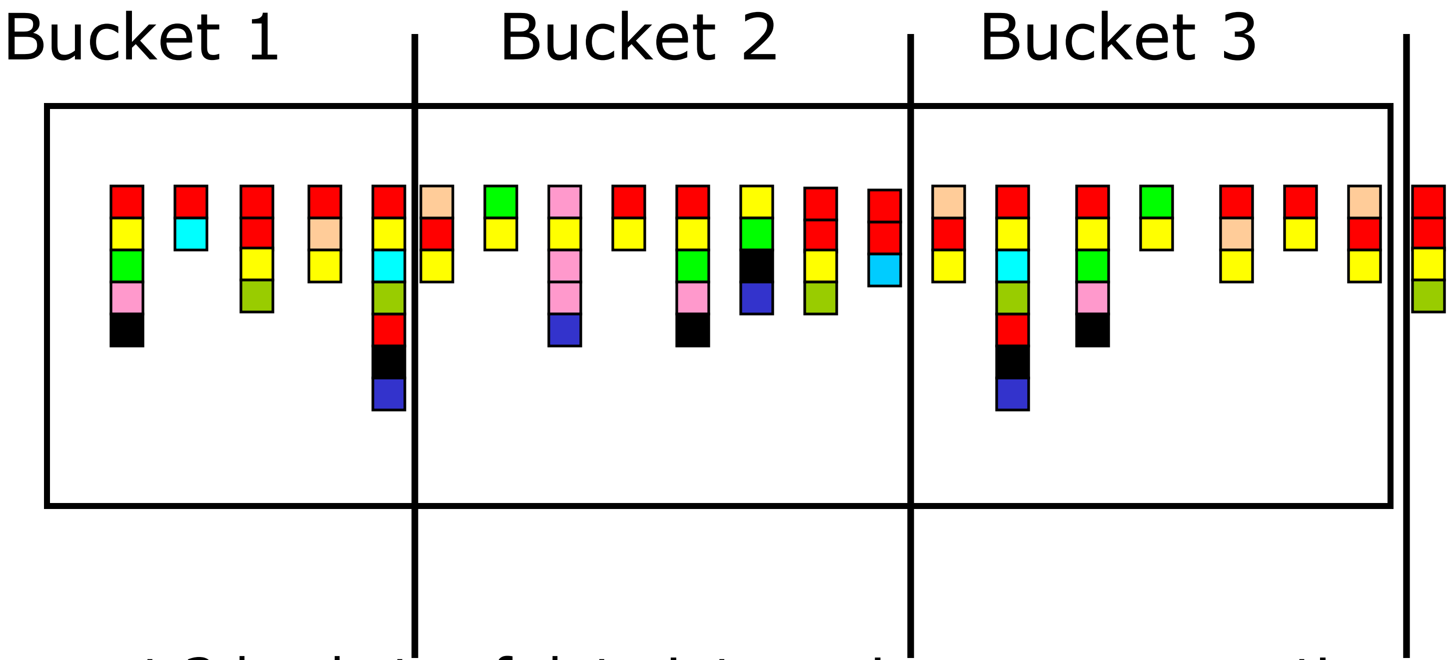
- How much do we undercount?
  - If (stream length seen so far = N) and (bucket-size = $1/\varepsilon$) then frequency count error $\leq$ #buckets $= \varepsilon N$

- Approximation guarantee
  - No false negatives
  - False positives have true frequency count at least $(\sigma-\varepsilon)N$
  - Frequency count underestimated by at most εN
  - The space requirement is limited to $1/\varepsilon \log(\varepsilon N)$

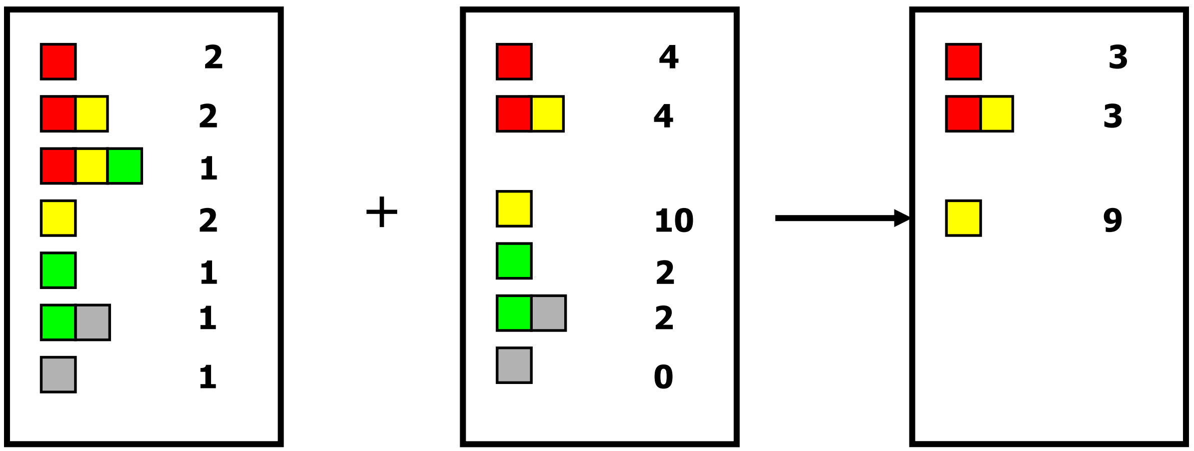
Lossy Counting For Frequent Itemsets

Divide Stream into 'Buckets' as for frequent items

But fill as many buckets as possible in main memory one time



If we put 3 buckets of data into main memory one time,
Then decrease each frequency count by 3



summary data

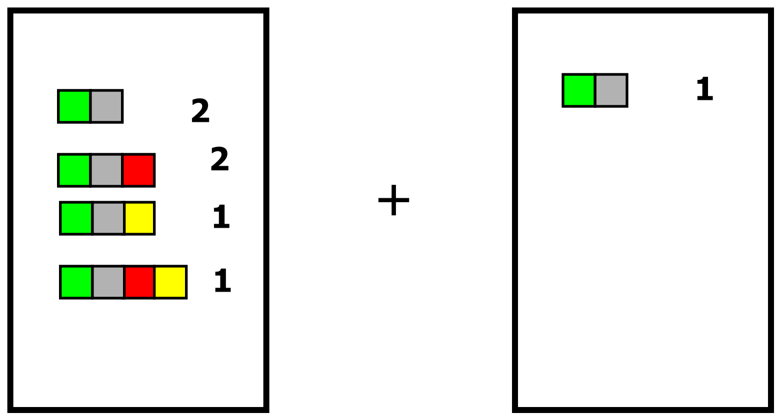
3 bucket data in memory

summary data

Itemset () is deleted.

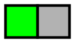
That's why we choose a large number of buckets – delete more

Pruning Itemsets – Apriori Rule



summary data

3 bucket data in memory

If we find itemset () is not frequent itemset,
Then we needn’t consider its superset

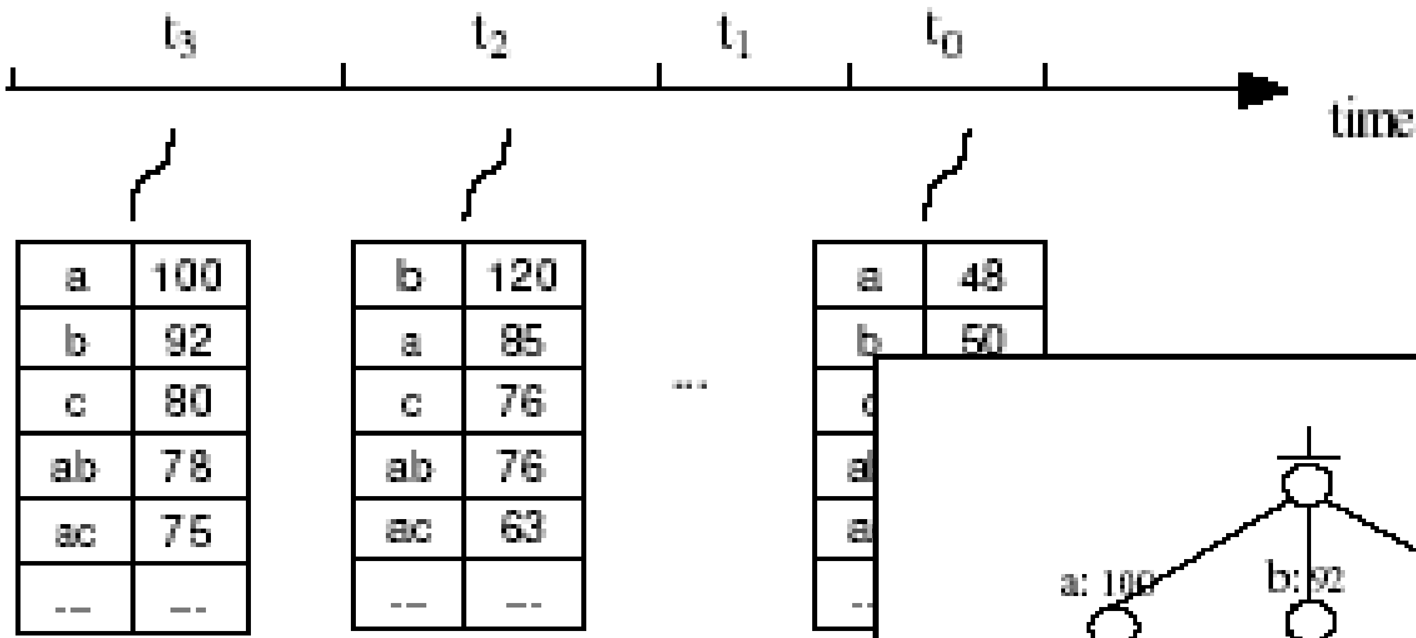
Summary of Lossy Counting

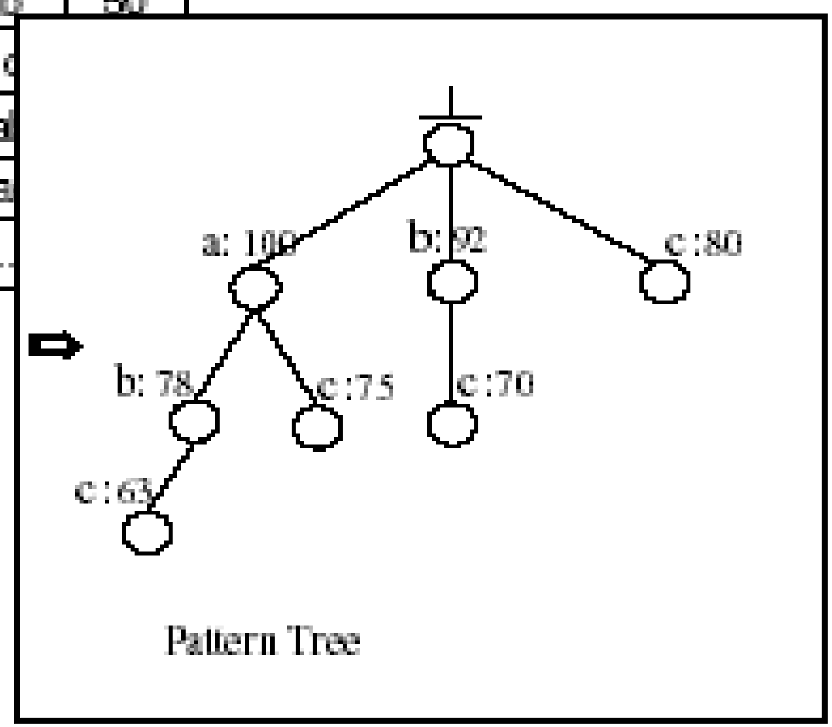
- Strength
  - A simple idea
  - Can be extended to frequent itemsets
- Weakness:
  - Space Bound is not good
  - For frequent itemsets, they do scan each record many times
  - The output is based on all previous data. But sometimes, we are only interested in recent data
- A space-saving method for stream frequent item mining
  - Metwally, Agrawal and El Abbadi, ICDT'05

Mining Evolution of Frequent Patterns for Stream Data

- Approximate frequent patterns (Manku & Motwani VLDB'02)
  - Keep only current frequent patterns:
    No changes can be detected
- Mining evolution and dramatic changes of frequent patterns (Giannella, Han, Yan, Yu, 2003)
  - Use tilted time window frame
  - Use compressed form to store significant (approximate) frequent patterns and their time-dependent traces
- Note: To mine precise counts, one has to trace/keep a fixed (and small) set of items

Two Structures for Mining Frequent Patterns with Tilted-Time Window (1)

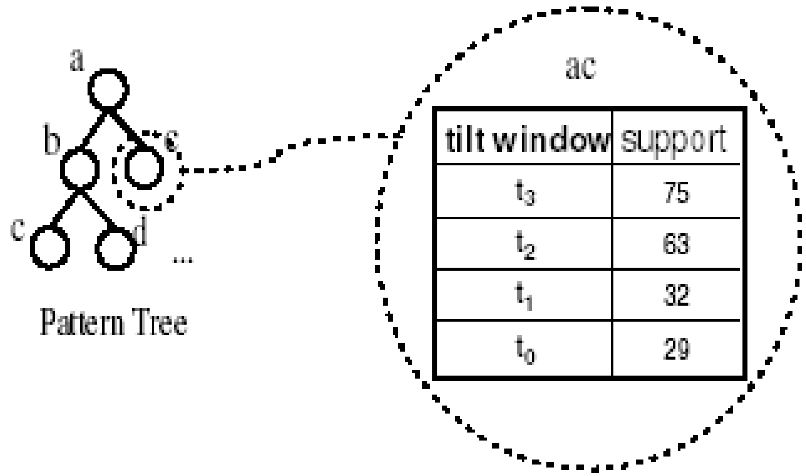




- FP-Trees store Frequent Patterns, rather than Transactions
- Tilted-time major: An FP-tree for each tilted time frame

Two Structures for Mining Frequent Patterns with Tilted-Time Window (2)

- The second data structure:
  - Observation: FP-Trees of different time units are similar
  - Pattern-tree major: each node is associated with a tilted-time window



Pattern Tree

tilt window	support
t_3	75
t_2	63
t_1	32
t_0	29

Tilted-time Window Table

Classification

Classification in Data Streams
What are the issues?

- It is impossible to store the whole data set, as traditional classification algorithms require
- It is usually not possible to perform multiple scans of the input data
- Data streams are time-varying! There is concept drift.

- Decision tree induction for stream data classification
  - VFDT (Very Fast Decision Tree)/CVFDT
- Is decision-tree good for modeling fast changing data, e.g., stock market analysis?
- Other stream classification methods
  - Instead of decision-trees, consider other models:
    Naïve Bayesian,
    Ensemble (Wang, Fan, Yu, Han. KDD'03)
    K-nearest neighbors (Aggarwal, Han, Wang, Yu. KDD'04)
  - Tilted time framework, incremental updating, dynamic maintenance, and model construction
  - Comparing of models to find changes

Hoeffding Tree

- Initially introduced to analyze click-streams
- With high probability, classifies tuples the same
- Only uses small sample
  - Based on Hoeffding Bound principle

- Hoeffding Bound (Additive Chernoff Bound)
  - r: random variable representing the attribute selection method
  - R: range of r
  - n: # independent observations
  - Mean of r is at least $r_{avg} - \varepsilon$, with probability $1 - \delta$

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$$

Hoeffding Tree Algorithm

- The algorithm uses the bound to determin, with high probability the smallest number N of examples needed at a node to select the splitting attribute

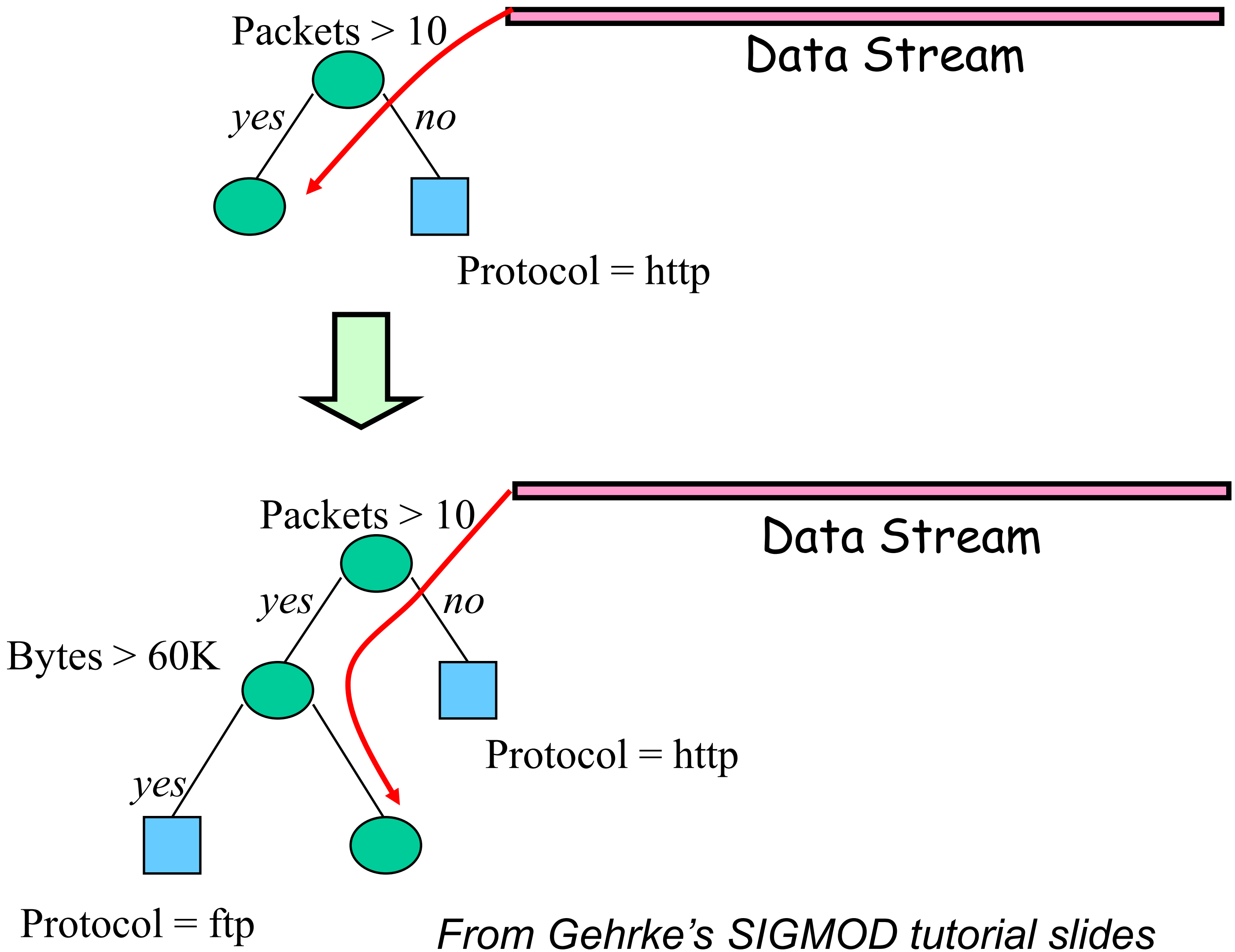
Hoeffding Tree Algorithm

- Hoeffding Tree Input
  - S: sequence of examples
  - X: attributes
  - G( ): evaluation function
  - δ: desired accuracy

```
for each example in S
   retrieve G(Xa) and G(Xb)   //two highest G(Xi)
   if ( G(Xa) – G(Xb) > ε )
    split on Xa
    recurse to next node
    break
```

- Complexity is O(ldvc) where l is the depth, d is the number of attributes, v is the maximum number of attributes, c is the number of classes

Decision-Tree Induction with Data Streams



From Gehrke's SIGMOD tutorial slides

Hoeffding Tree:
Strengths and Weaknesses

Strengths

- Scales better than traditional methods
  - Sublinear with sampling
  - Very small memory utilization
- Incremental
  - Make class predictions in parallel
  - New examples are added as they come

Weaknesses

- Could spend a lot of time with ties
- Memory used with tree expansion
- Number of candidate attributes

VFDT (Very Fast Decision Tree)

- Modifications to Hoeffding Tree
  - Near-ties broken more aggressively
  - G computed every nmin
  - Deactivates certain leaves to save memory
  - Poor attributes dropped
  - Initialize with traditional learner (helps learning curve)
- Compare to Hoeffding Tree: Better time and memory
- Compare to traditional decision tree
  - Similar accuracy
  - Better runtime with 1.61 million examples
    - 21 minutes for VFDT
    - 24 hours for C4.5
- Still does not handle concept drift

CVFDT (Concept-adapting VFDT)

- Concept Drift
  - Time-changing data streams
  - Incorporate new and eliminate old
- CVFDT
  - Increments count with new example
  - Decrement old example
    - Sliding window
    - Nodes assigned monotonically increasing IDs
  - Grows alternate subtrees
  - When alternate more accurate, then replace old
  - O(w) better runtime than VFDT-window

Ensemble of Classifiers Algorithm

- H. Wang, W. Fan, P. S. Yu, and J. Han, “Mining Concept-Drifting Data Streams using Ensemble Classifiers”, KDD'03.
- Method (derived from the ensemble idea in classification)

```
train K classifiers from K chunks
for each subsequent chunk
     train a new classifier
     test other classifiers against the chunk
     assign weight to each classifier
     select top K classifiers
```

Clustering

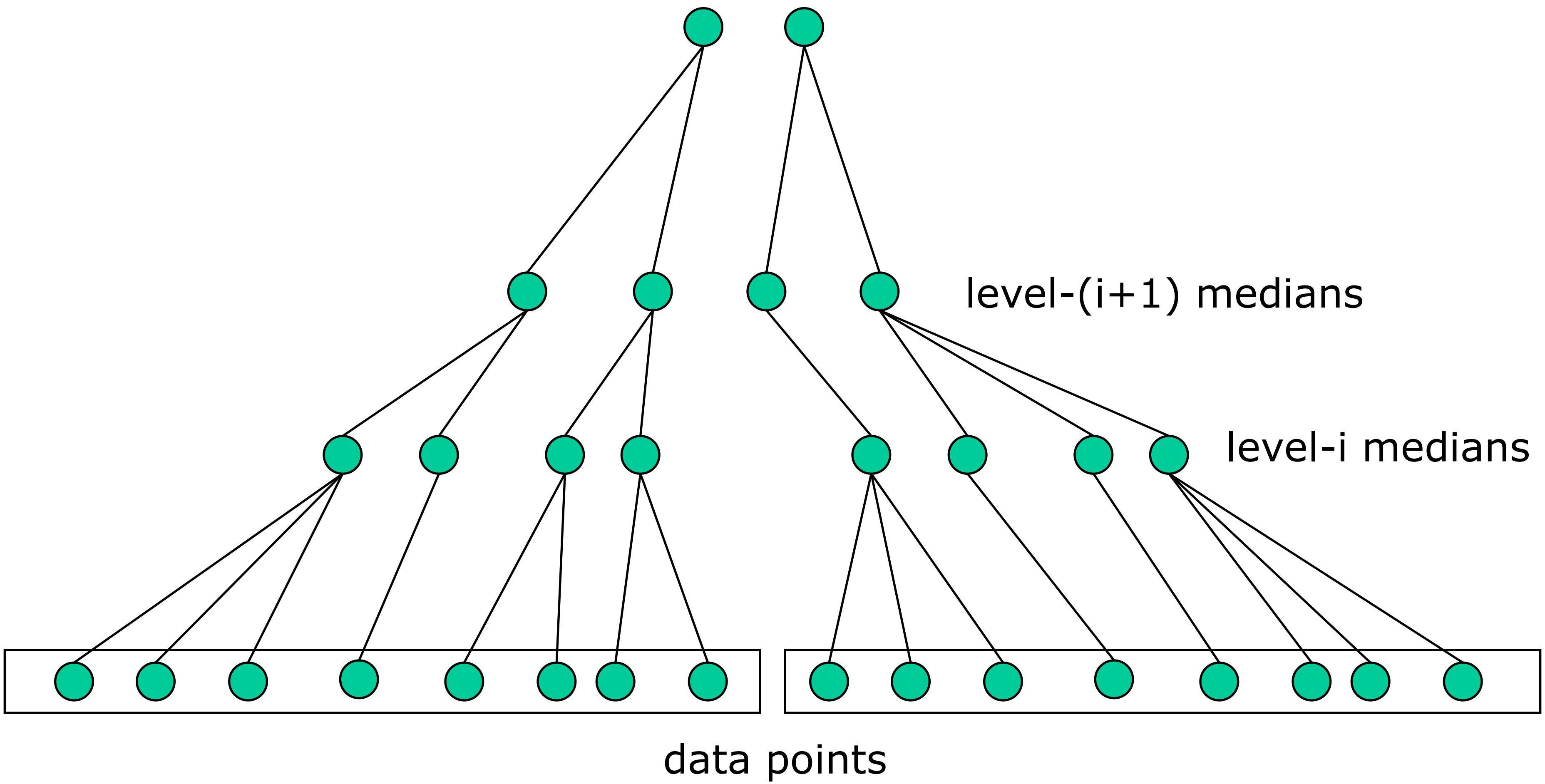
Clustering Evolving Data Streams
What methodologies?

- Compute and store summaries of past data
- Apply a divide-and-conquer strategy
- Incremental clustering of incoming data streams
- Perform microclustering as well as macroclustering anlysis
- Explore multiple time granularity for the analysis of cluster evolution
- Divide stream clustering into on-line and off-line processes

- Base on the k-median method
  - Data stream points from metric space
  - Find k clusters in the stream s.t. the sum of distances from data points to their closest center is minimized
- Constant factor approximation algorithm
  In small space, a simple two step algorithm:

  1. For each set of M records, S_i,
     find O(k) centers in $S_1, \ldots, S_l$
     Local clustering: Assign each point in S_i to its closest center
  2. Let S' be centers for $S_1, \ldots, S_l$ with each
     center weighted by number of points assigned to it
     Cluster S' to find k centers

Hierarchical Clustering Tree



Hierarchical Tree and Drawbacks

- Method
  - Maintain at most m level-i medians
  - On seeing m of them, generate O(k) level-(i+1) medians of weight equal to the sum of the weights of the intermediate medians assigned to them
- Drawbacks
  - Low quality for evolving data streams (register only k centers)
  - Limited functionality in discovering and exploring clusters over different portions of the stream over time

- Network intrusion detection: one example
  - Detect bursts of activities or abrupt changes in real time—by on-line clustering
- The methodology
  by C. Agarwal, J. Han, J. Wang, P.S. Yu, VLDB'03
  - Tilted time frame work:
    o.w. dynamic changes cannot be found
  - Micro-clustering: better quality than k-means/k-median
    - incremental, online processing and maintenance)
  - Two stages: micro-clustering and macro-clustering
  - With limited "overhead" to achieve high efficiency, scalability, quality of results and power of evolution/change detection

CluStream: A Framework for Clustering Evolving Data Streams

- Design goal
  - High quality for clustering evolving data streams with greater functionality
  - While keep the stream mining requirement in mind
    - One-pass over the original stream data
    - Limited space usage and high efficiency
- CluStream: A framework for clustering evolving data streams
  - Divide the clustering process into online and offline components
  - Online component: periodically stores summary statistics about the stream data
  - Offline component: answers various user questions based on the stored summary statistics

The CluStream Framework

- Micro-cluster
  - Statistical information about data locality
  - Temporal extension of the cluster-feature vector
    - Multi-dimensional points $X_1 \ldots X_k \ldots$ with time stamps $T_1 \ldots T_k \ldots$
    - Each point contains d dimensions, i.e., $X = (x^1 \ldots x^d)$
    - A micro-cluster for n points is defined as a (2.d + 3) tuple

$$\left(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n\right)$$

- Pyramidal time frame
  - Decide at what moments the snapshots of the statistical information are stored away on disk

CluStream: Pyramidal Time Frame

- Snapshots of a set of micro-clusters are stored following the pyramidal pattern
- They are stored at differing levels of granularity depending on the recency
- Snapshots are classified into different orders varying from 1 to log(T)
  - The i-th order snapshots occur at intervals of α^i where $\alpha \geq 1$
  - Only the last $(\alpha + 1)$ snapshots are stored

CluStream: Clustering On-line Streams

- Online micro-cluster maintenance
  - Initial creation of q micro-clusters
    - q is usually significantly larger than the number of natural clusters
  - Online incremental update of micro-clusters
    - If new point is within max-boundary, insert into the micro-cluster
    - O.w., create a new cluster
    - May delete obsolete micro-cluster or merge two closest ones
- Query-based macro-clustering
  - Based on a user-specified time-horizon h and the number of macro-clusters K, compute macroclusters using the k-means algorithm

Stream Data Mining: What are the Research Issues?

- Mining sequential patterns in data streams
- Mining partial periodicity in data streams
- Mining notable gradients in data streams
- Mining outliers and unusual patterns in data streams

- Stream clustering
  - Multi-dimensional clustering analysis?
    Cluster not confined to 2-D metric space, how to incorporate other features, especially non-numerical properties
  - Stream clustering with other clustering approaches?
  - Constraint-based cluster analysis with data streams?

Summary

Summary: Stream Data Mining

- Stream Data Mining is a rich and on-going research field
- Current research focus in database community:
  - DSMS system architecture
  - Continuous query processing
  - Supporting mechanisms
- Stream data mining and stream OLAP analysis
  - Powerful tools for finding general and unusual patterns
  - Effectiveness, efficiency and scalability: lots of open problems
- Philosophy on stream data analysis and mining
  - A multi-dimensional stream analysis framework
  - Time is a special dimension: Tilted time frame
  - What to compute and what to save?—Critical layers
  - Partial materialization and precomputation
  - Mining dynamics of stream data

References on Stream Data Mining (1)

- C. Aggarwal, J. Han, J. Wang, P. S. Yu. A Framework for Clustering Data Streams, VLDB'03
- C. C. Aggarwal, J. Han, J. Wang and P. S. Yu. On-Demand Classification of Evolving Data Streams, KDD'04
- C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Projected Clustering of High Dimensional Data Streams, VLDB'04
- S. Babu and J. Widom. Continuous Queries over Data Streams. SIGMOD Record, Sept. 2001
- B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom. Models and Issues in Data Stream Systems", PODS'02. (Conference tutorial)
- Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. "Multi-Dimensional Regression Analysis of Time-Series Data Streams, VLDB'02
- P. Domingos and G. Hulten, "Mining high-speed data streams", KDD'00
- A. Dobra, M. N. Garofalakis, J. Gehrke, R. Rastogi. Processing Complex Aggregate Queries over Data Streams, SIGMOD'02
- J. Gehrke, F. Korn, D. Srivastava. On computing correlated aggregates over continuous data streams. SIGMOD'01
- C. Giannella, J. Han, J. Pei, X. Yan and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities, Kargupta, et al. (eds.), Next Generation Data Mining'04

References on Stream Data Mining (2)

- S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams, FOCS'00
- G. Hulten, L. Spencer and P. Domingos: Mining time-changing data streams. KDD 2001
- S. Madden, M. Shah, J. Hellerstein, V. Raman, Continuously Adaptive Continuous Queries over Streams, SIGMOD02
- G. Manku, R. Motwani. Approximate Frequency Counts over Data Streams, VLDB’02
- A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. ICDT'05
- S. Muthukrishnan, Data streams: algorithms and applications, Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, 2003
- R. Motwani and P. Raghavan, Randomized Algorithms, Cambridge Univ. Press, 1995
- S. Viglas and J. Naughton, Rate-Based Query Optimization for Streaming Information Sources, SIGMOD’02
- Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, VLDB’02
- H. Wang, W. Fan, P. S. Yu, and J. Han, Mining Concept-Drifting Data Streams using Ensemble Classifiers, KDD'03