Mining Data Streams

JIAWEI HAN
COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

APRIL 4, 2016

Mining Data Streams

- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary

Data Streams and Their Characteristics

- Data Streams
  - Features: Continuous, ordered, changing, fast, huge volume
  - Contrast with traditional DBMS (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 low-level and multi-dimensional in nature, needs multi-level and multi-dimensional processing

Streaming Data 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 vs. DSMS (Data Stream Management Systems)

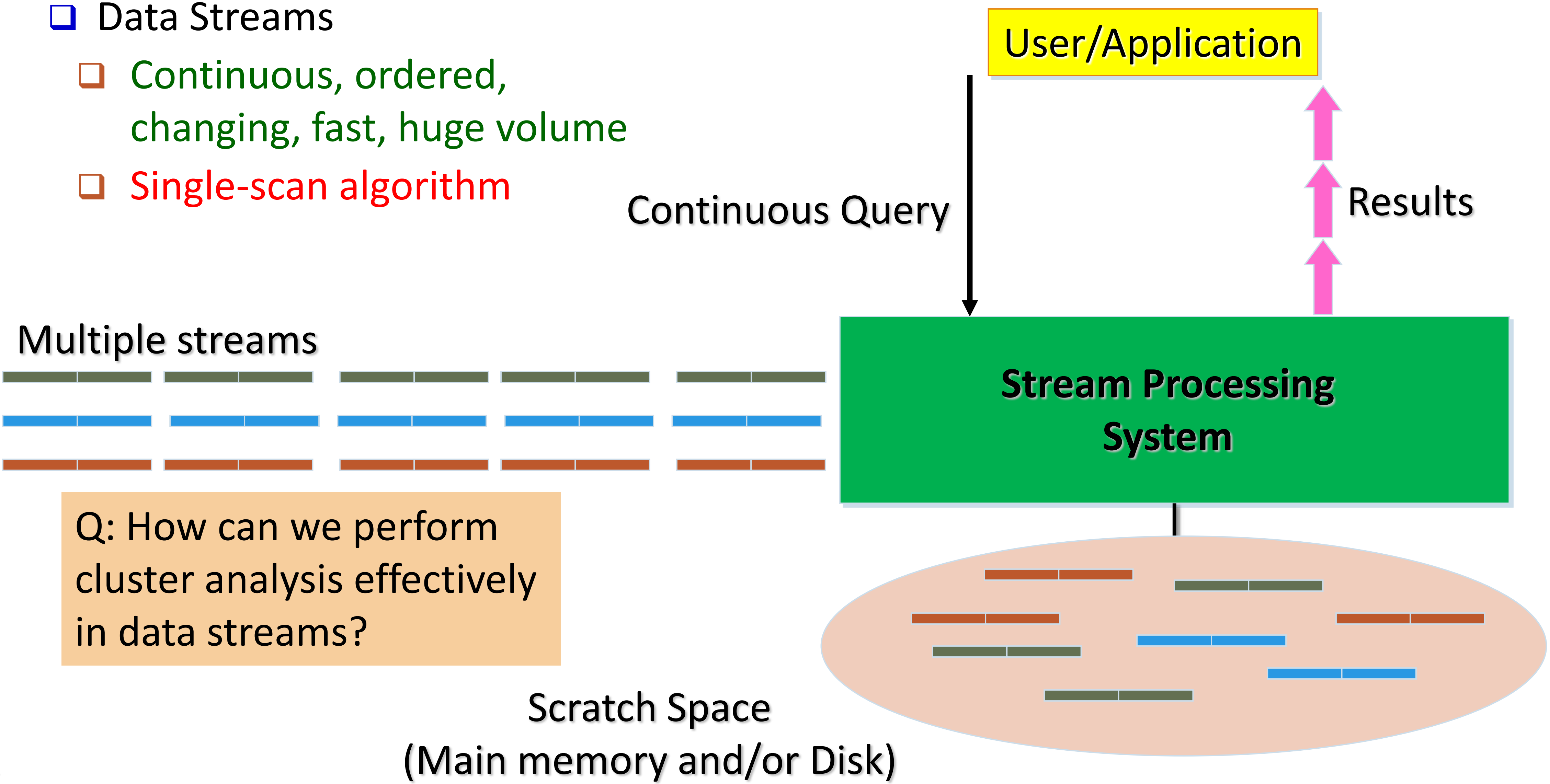
- Persistent relations
- One-time queries
- Random access
- “Unbounded” disk store
- Only current state matters
- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

- 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’04 tutorial slides

Stream Data Processing: An Architecture

- Data Streams
  - Continuous, ordered, changing, fast, huge volume
  - Single-scan algorithm



Q: How can we perform cluster analysis effectively in data streams?

Challenges of Stream Query 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 query processing
  - Most stream data are at low-level or multi-dimensional in nature

Stream Data Mining Tasks

- Stream mining vs. stream querying
  - Stream mining shares many difficulties with stream querying
    - E.g., single-scan, fast response, dynamic, …
    - But often requires less “precision”, e.g., no join, grouping, sorting
  - Patterns are hidden and more general than querying
- Stream data mining tasks
  - Multi-dimensional on-line analysis of streams
  - Pattern mining in data streams
  - Classification of stream data
  - Clustering data streams
  - Mining outliers and anomalies in stream data

Challenges of 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?

Mining Data Streams

- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary

Multi-Dimensional Stream Analysis: Examples

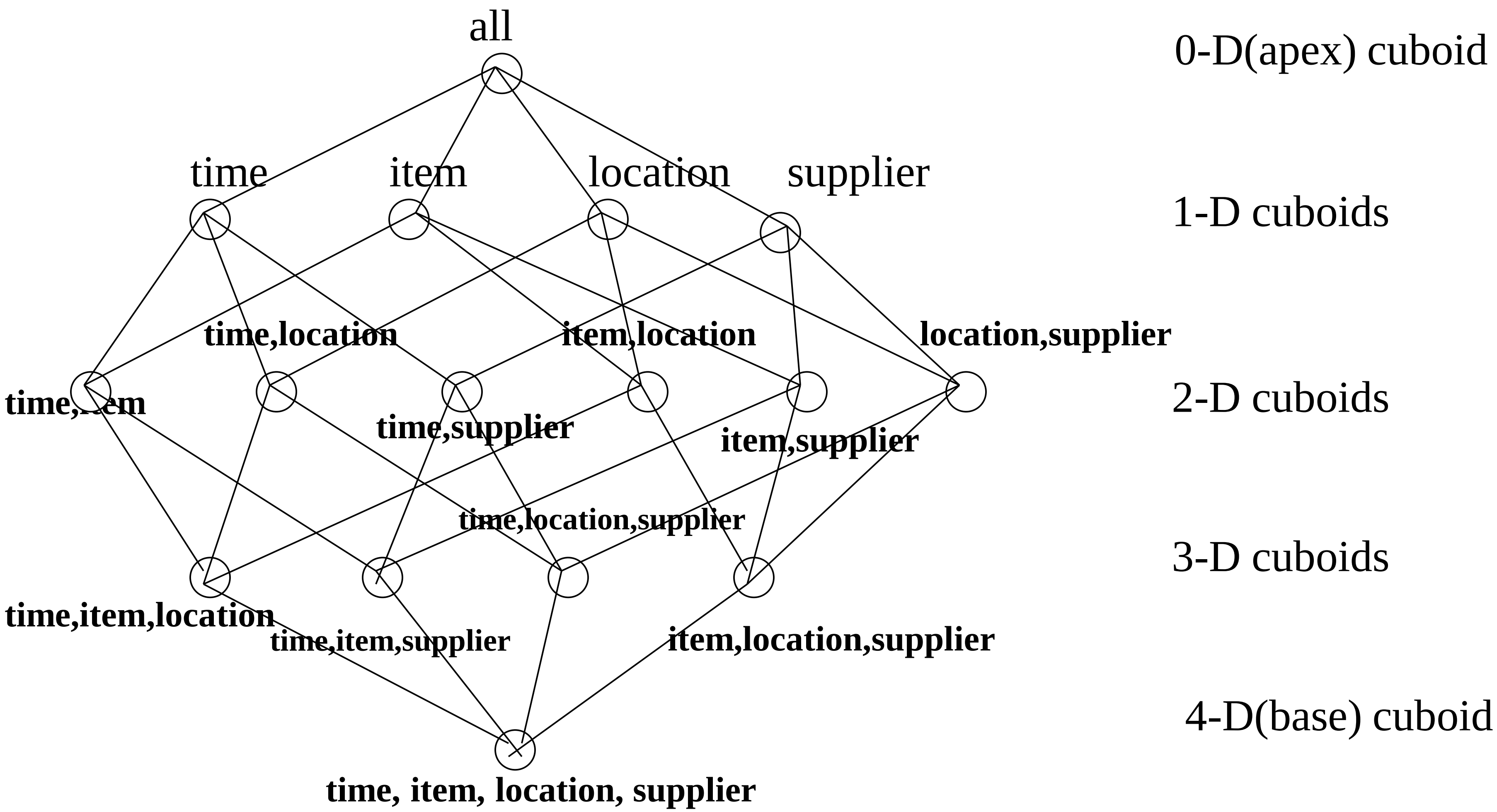
- Analysis of Web click streams
  - Raw data at low levels: seconds, web page addresses, user IP addresses, …
  - Analysts want: changes, trends, unusual patterns, at reasonable levels of details
  - E.g., *Average clicking traffic in North America on sports in the last 15 minutes is 40% higher than 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*

A Stream Cube Architecture

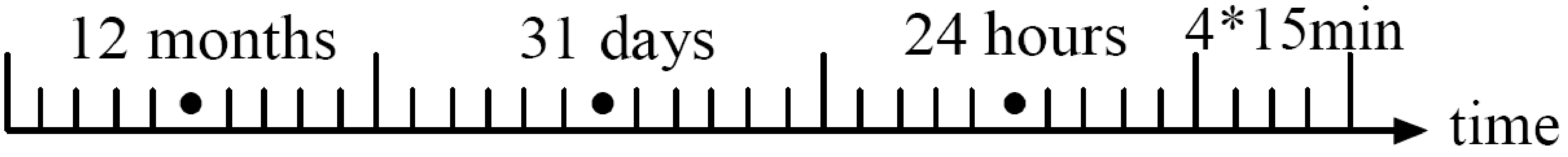
- A tilted time frame
  - Different time granularities
    - second, minute, quarter, hour, day, week, …
- Critical layers
  - Minimum interest layer (m-layer)
  - Observation layer (o-layer)
  - User: watches at o-layer and occasionally needs to drill-down down to m-layer
- Partial materialization of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

Cube: A Lattice of Cuboids

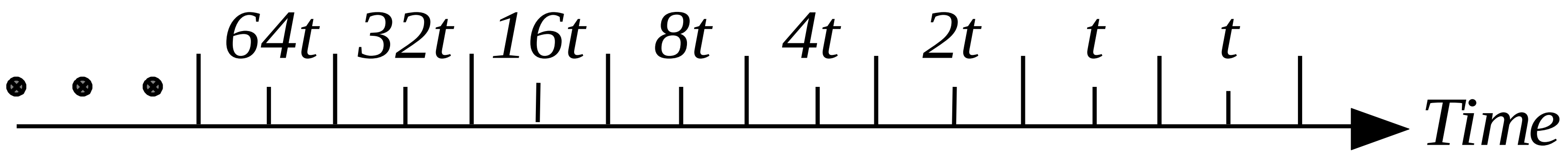


Time Dimension: A Tilted Time Model

- **Tilted time frames:** A trade-off between space and granularity of time
  - Decide at what moments the snapshots of the statistical information are stored
- **Design**: *Natural, logarithmic* and *pyramidal* tilted time frames
  - **Natural tilted time frame:**
    - Ex: Minimal: 15min, then 4 * 15mins → 1 hour, 24 hours → day, …

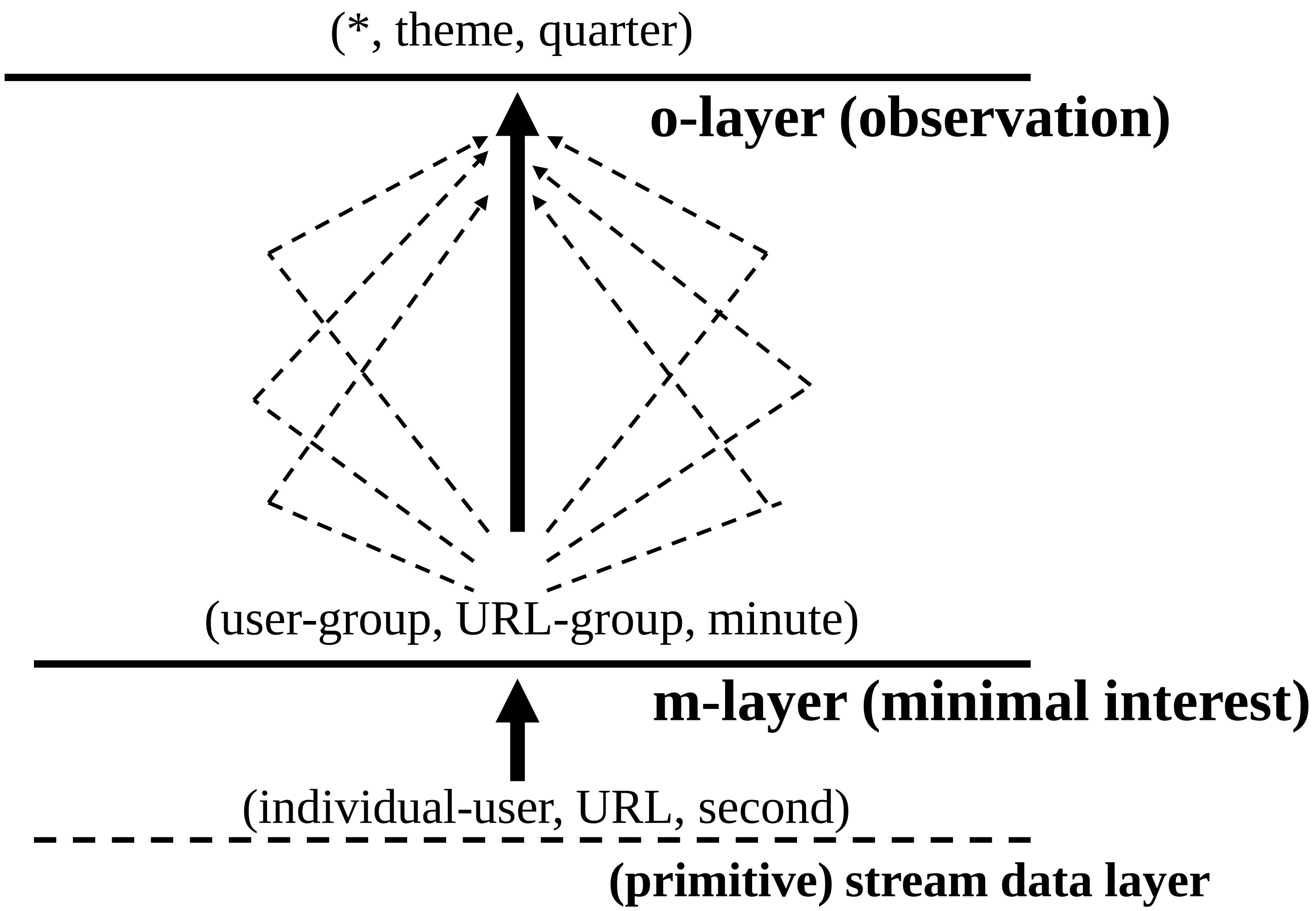


  - **Logarithmic tilted time frame:**
    - Ex. Minimal: 1 minute, then 1, 2, 4, 8, 16, 32, …



Two Critical Generalized Layers in the Stream Cube

- Raw data stream sits at the "primitive" stream data layer
- Stream data is generalized to m-layer (minimal interest layer) and "stored" to facilitate flexible drilling
- Stream data should be constantly summarized and presented at the o-layer (observation layer) for constant observation

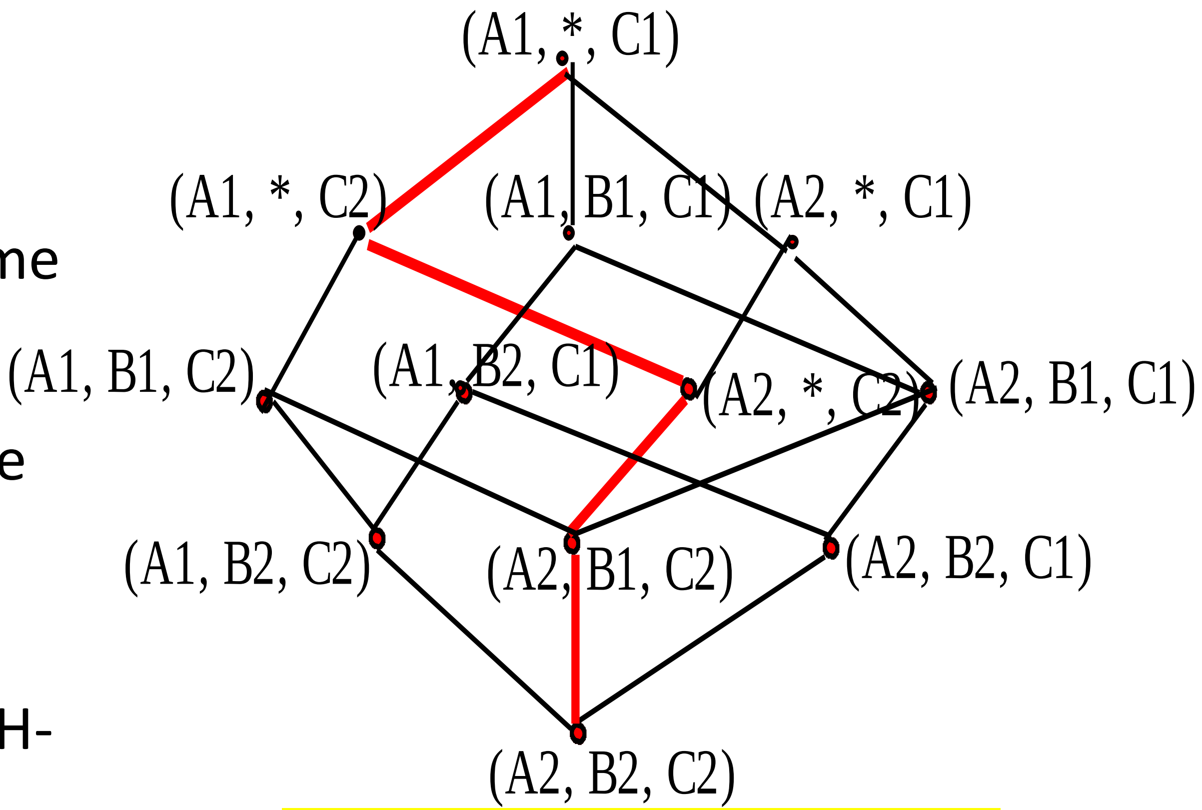


OLAP Operation and Cube Materialization

- OLAP(Online Analytical Processing) operations:
  - Roll up (drill-up): summarize data
    - *by climbing up hierarchy or by dimension reduction*
  - Drill down (roll down): reverse of roll-up
    - *from higher level summary to lower level summary or detailed data, or introducing new dimensions*
  - Slice and dice: *project and select*
  - Pivot (rotate): *reorient the cube, visualization, 3D to series of 2D planes*
- Cube partial materialization
  - Store *some* pre-computed cuboids for fast online processing

On-Line Partial Materialization

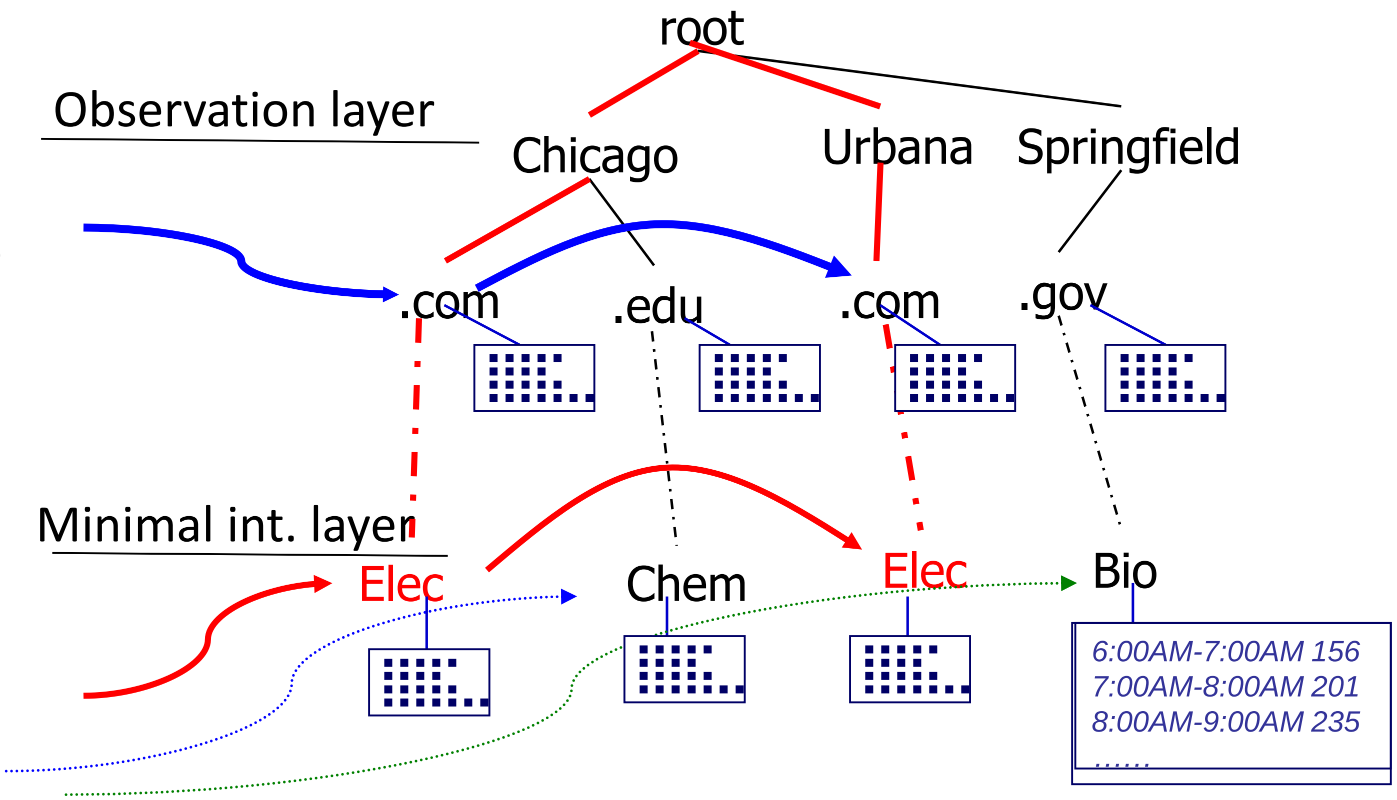
- Materialization takes precious space and time
  - Only incremental materialization (with tilted time frame)
- Only materialize "cuboids" of the critical layers?
  - Online computation may take too much time
- Preferred solution:
  - *Popular-path* approach: Materializing those along the popular drilling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



Materialization on Popular Path

OLAP Processing Using Stream Cubes

- Online aggregation vs. query-based computation
  - Online computing while streaming: aggregating stream cubes
  - Query-based computation: Using computed cuboids
- An H-tree cubing Structure (Ref.: Han, et al., SIGMOD'01)
  - 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*



Mining Data Streams

- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary

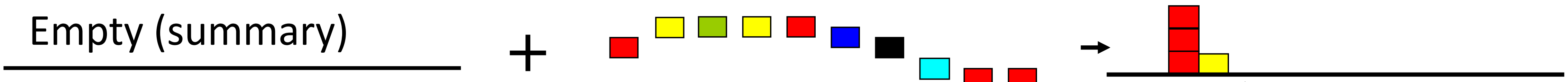
Mining Approximate Frequent Patterns

- Mining precise frequent patterns in stream data: Unrealistic
  - Cannot even store them in a compressed form (e.g., FPtree)
- Approximate answers are often sufficient for pattern analysis
  - Ex.: 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 to keep the items with very low support count
    - Advantage: Guaranteed error bound
    - Disadvantage: Keeping a large set of traces

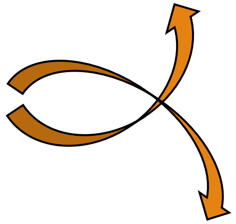
Lossy Counting for Frequent Single Items

Divide stream into ‘buckets’ (bucket size is 1/ ε = 1000)

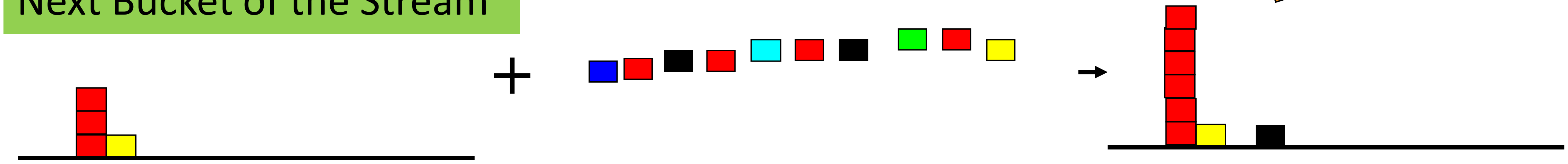
First Bucket of the Stream



At bucket boundary, decrease all counters by 1



Next Bucket of the Stream



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 = $1/\varepsilon$

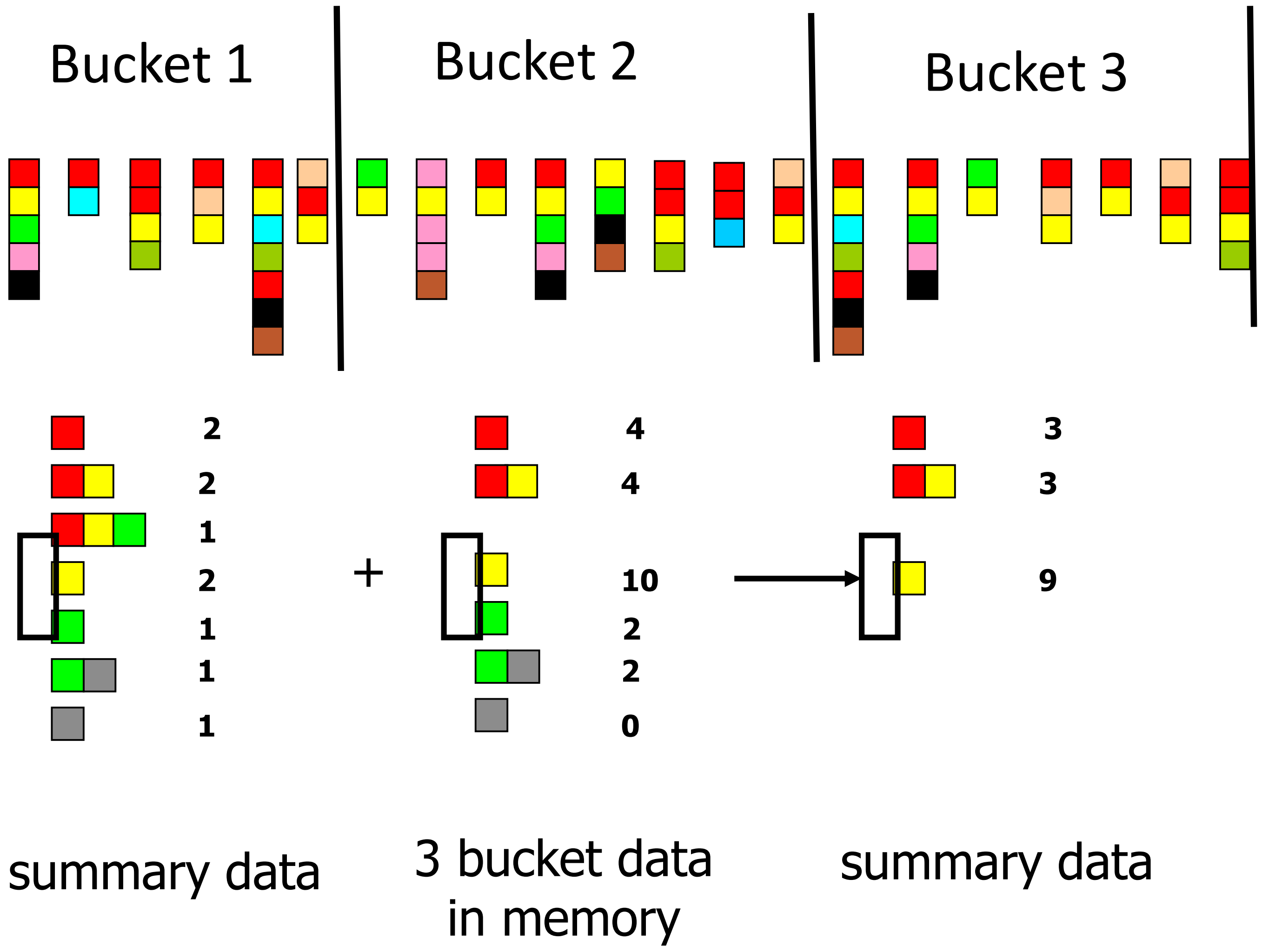
then frequency count error $\leq$ # of buckets

$= N/\text{bucket-size} = N/(1/\varepsilon) = \varepsilon N$

- Approximation guarantee
  - No false negatives
  - False positives have true frequency count at least $(\sigma - \varepsilon)N$
  - Frequency count underestimated by at most εN

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, then decrease each frequency count by 3
- Update summary data structure
  - Itemset ( ) is deleted. That's why we choose a large number of buckets–delete more
- Pruning Itemsets – Apriori Rule
  - If we find itemset ( ) is not frequent, we needn't consider its superset



Other Issues and Recommended Readings

- Other issues on pattern discovery in data streams
  - Space-saving computation of frequent and top-*k* elements (Metwally, Agrawal, and El Abbadi, ICDT'05)
  - Mining approximate frequent *k*-itemsets in data streams
  - Mining sequential patterns in data streams
- Recommended Readings
  - G. Manku and 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

Mining Data Streams

- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary

Classification for Dynamic Data Streams

- Decision tree induction for stream data classification
  - VFDT (Very Fast Decision Tree)/CVFDT (Domingos, Hulten, Spencer, KDD00/KDD01)
- 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)
    - Classifying skewed stream data (Gao, Fan, and Han, SDM'07)
- Evolution modeling: Tilted time framework, incremental updating, dynamic maintenance, and model construction
  - Comparing of models to find changes

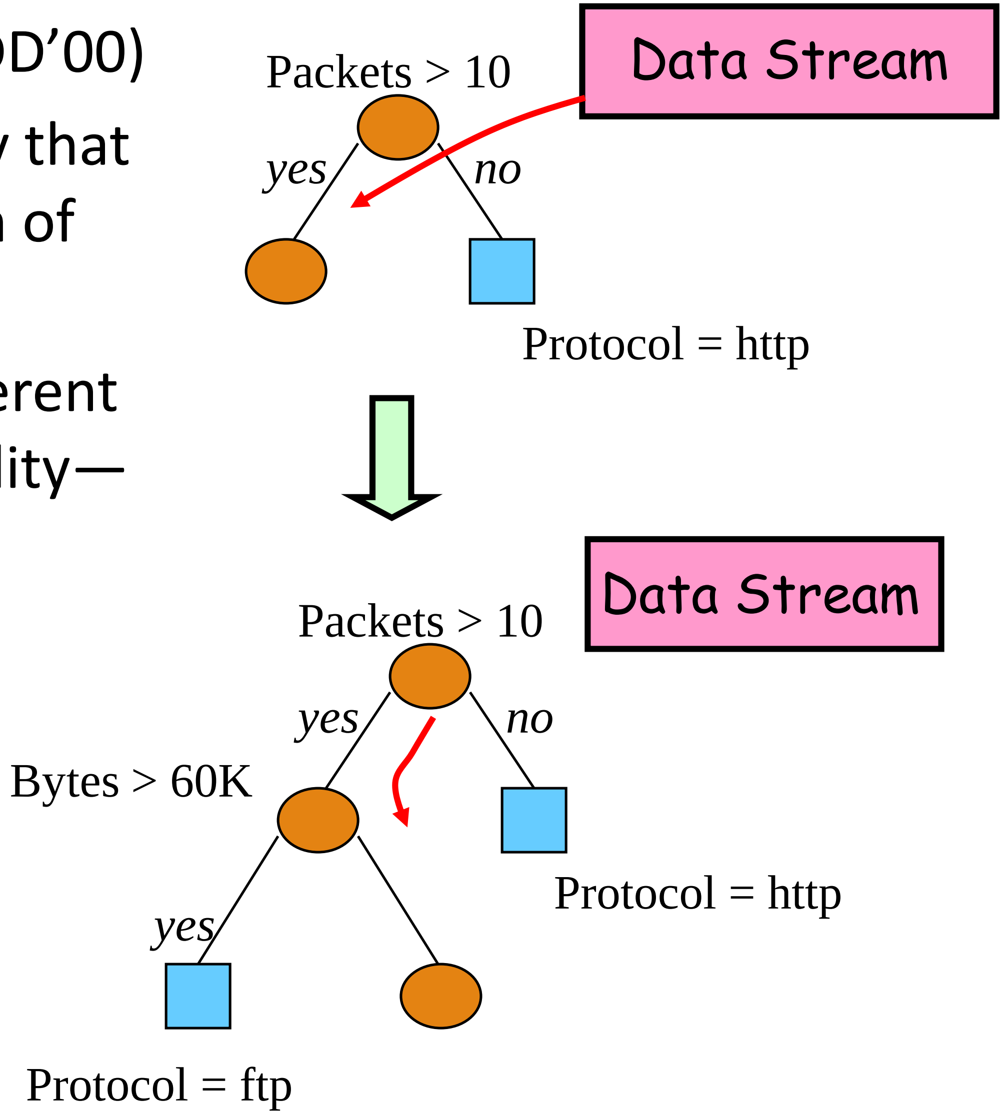
Very Fast Decision Tree for Data Streams

- Very Fast Decision Trees(VFDT) (Domingos, et al., KDD'00)
- Hoeffding's inequality: A result in probability theory that gives an upper bound on the probability for the sum of random variables to deviate from its expected value
- Based on Hoeffding Bound principle, classifying different samples leads to the same model with high probability—can use a small set of samples
- Hoeffding Bound (Additive Chernoff Bound)
  - Given: r: random variable, R: range of r, N: # independent observations
  - True mean of r is at least $r_{avg} - \varepsilon$,

    with probability $1 - \delta$

    (where δ is user-specified)

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



Ack. From Gehrke's SIGMOD tutorial slides

Hoeffding Tree: How to Handle Concept Drifts?

- Hoeffding Tree: strengths and weakness
  - 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
  - Weakness
    - Could spend a lot of time with ties
    - Memory used with tree expansion
    - Number of candidate attributes
- Concept Drift
  - Time-changing data streams
  - Incorporate new and eliminate old
- CVFDT (Concept-adapting VFDT)
  - Increments count with new example
  - Decrement old example
    - Sliding window
    - Nodes assigned monotonically increasing IDs
  - Grows alternate subtrees
    - When alternate more accurate: Replace the old one
  - O(w) better runtime than VFDT-window

Ensemble of Classifiers

- Ensemble is a better way to handle concept drift than single trees
  - 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

Issues in Stream Classification

- Descriptive model vs. generative model
  - Generative models assume data follows some distribution while descriptive models make no assumptions
  - Distribution of stream data is unknown and may evolve, so descriptive model is better
- Label prediction vs. probability estimation
  - Classify test examples into one class or estimate $P(y|x)$ for each y
  - Probability estimation is better:
    - Stream applications may be stochastic (an example could be assigned to several classes with different probabilities)
    - Probability estimates provide confidence information and could be used in post processing

Classifying Data Streams with Skewed Distribution

- Problems of typical classification methods on skewed data:
  - Tend to ignore positive examples due to the small number
  - The cost of misclassifying positive examples is usually huge, e.g., misclassifying credit card fraud as normal
- Classify data stream with skewed distribution (i.e., rare events)
  - Employ both **biased sampling** and **ensemble** techniques
  - Reduce classification errors on the minority class

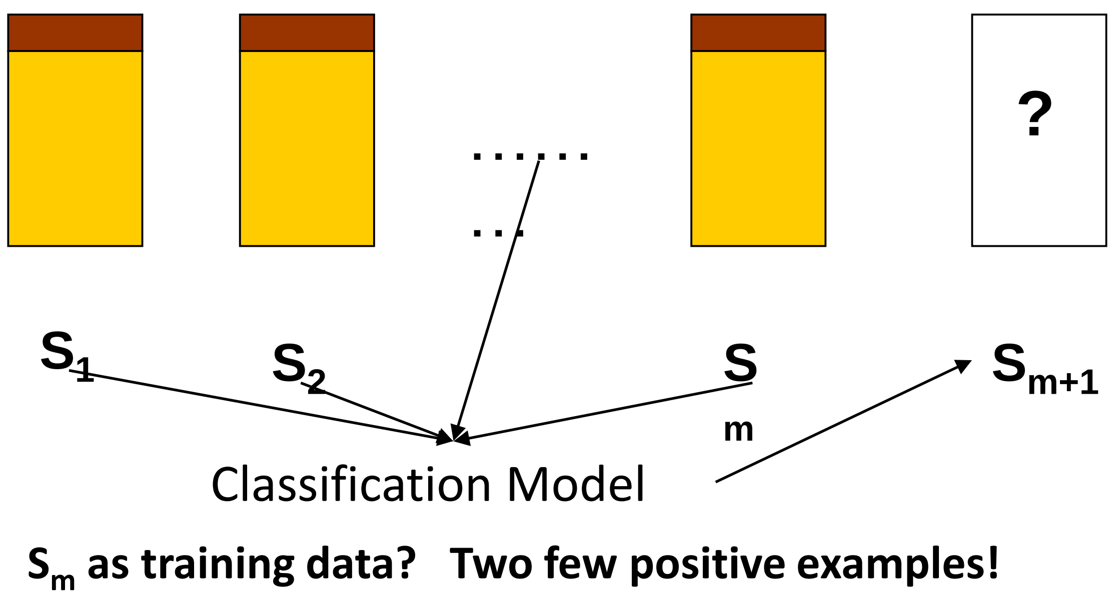
Concept Drifts

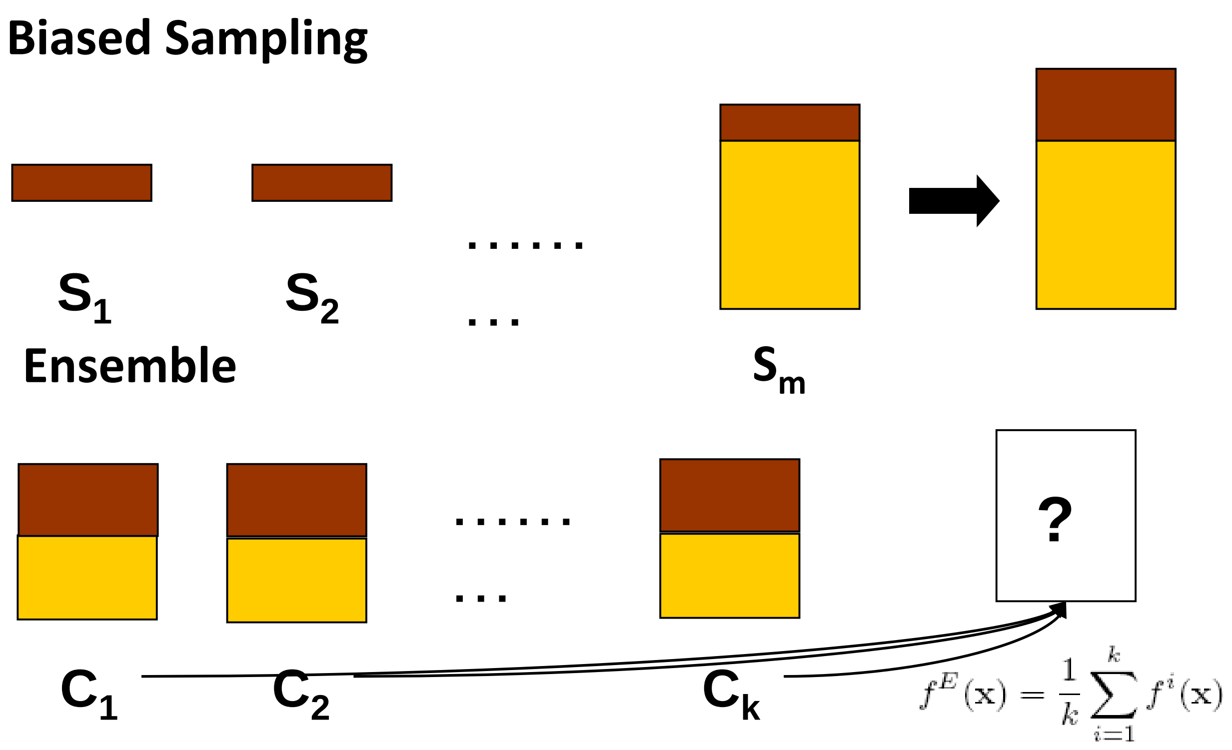
- Changes in P(x, y) x-feature vector y-class label $P(x,y) = P(y|x)P(x)$
- Four possibilities:
  - No change: $P(y|x)$, $P(x)$ remain unchanged
  - Feature change: only $P(x)$ changes
  - Conditional change: only $P(y|x)$ changes
  - Dual change: both $P(y|x)$ and $P(x)$ changes
- Expected error:

$$Err = \int_{(\mathrm{x},y)\in\mathcal{P}(\mathrm{x},y)} \mathcal{P}(\mathrm{x})(1 - \mathcal{P}(y_p|\mathrm{x}))\mathrm{dx}$$

- No matter how concept changes, the expected error could increase, decrease, or remain unchanged
- Training on the most recent data could help reduce expected error

Stream Ensemble Approach





- Ideas of *Stream Ensemble*
  - **Biased sampling:** Save only the positive examples in the streams
  - **Ensemble:** Partition negative examples of S_m into k portions to build k classifiers

Experiments: Mean Squared Error on Synthetic & Real Data

- Test on concept-drift streams (synthetic data)

Table 2: Mean Squared Error on Deterministic Stream Data

Changes	Decision Trees			Naive Bayes			Logistic Regression		
	SE	NS	SS	SE	NS	SS	SE	NS	SS
Feature	0.1275	0.9637	0.6446	**0.0577**	0.8693	0.4328	0.1501	0.8117	0.5411
Conditional	0.0943	0.9805	0.5500	**0.0476**	0.8830	0.4380	0.1301	0.8944	0.5729
Dual	0.0854	0.9521	0.5174	**0.0664**	0.8596	0.4650	0.1413	0.8371	0.5525

Table 3: Mean Squared Error on Stochastic Stream Data

Changes	Decision Trees			Naive Bayes			Logistic Regression		
	SE	NS	SS	SE	NS	SS	SE	NS	SS
Feature	0.0847	0.6823	0.4639	**0.0314**	0.5371	0.2236	0.0974	0.5311	0.3217
Conditional	0.0552	0.6421	0.4463	**0.0299**	0.5675	0.2449	0.1029	0.6578	0.4151
Dual	0.0684	0.6758	0.4107	**0.0301**	0.5981	0.2556	0.0887	0.6388	0.4075

- Test on real data

Table 4: Mean Squared Error on Real Data

Data Set	Decision Trees			Naive Bayes			Logistic Regression		
	SE	NS	SS	SE	NS	SS	SE	NS	SS
Thyroid1	**0.0000**	0.0799	0.0003	0.0057	0.0655	0.0366	0.0105	0.2001	0.0634
Thyroid2	**0.0001**	0.0266	0.0047	0.0505	0.4774	0.2323	0.0044	0.2443	0.0391
Opt	0.0147	0.1160	0.0414	0.0106	0.0972	0.0106	**0.0025**	0.1131	0.0225
Letter	**0.0191**	0.2290	0.0653	0.1024	0.2459	0.1567	0.0595	0.4091	0.2061
Covtype	0.0003	0.2500	0.0834	**0.0000**	0.4496e-7	0.0001e-7	0.0008	0.0835	0.0417

Stream Ensemble always has lower error rate

Experiments: Model Accuracy and Training Efficiency

- Model accuracy

Table 5: Decision Tree as Base Learner

	SE	NS	SS
Synthetic1	**0.9464**	0.5175	0.6944
Synthetic2	**0.9337**	0.4840	0.6611
Thyroid1	**1.0000**	0.9999	0.9999
Thyroid2	**0.9998**	**0.9998**	0.9996
Opt	**0.9942**	0.9495	0.9777
Letter	**0.9931**	0.9467	0.9782
Covtype	**1.0000**	**1.0000**	0.9999

Table 6: Naive Bayes as Base Learner

	SE	NS	SS
Synthetic1	**0.9532**	0.8220	0.9525
Synthetic2	**0.9558**	0.8355	0.9556
Thyroid1	**0.9982**	0.9979	**0.9982**
Thyroid2	**0.9551**	0.9054	0.9145
Opt	**0.9926**	0.9722	0.9898
Letter	**0.9395**	0.9389	0.9389
Covtype	**0.9997**	0.9995	**0.9997**

- Training time

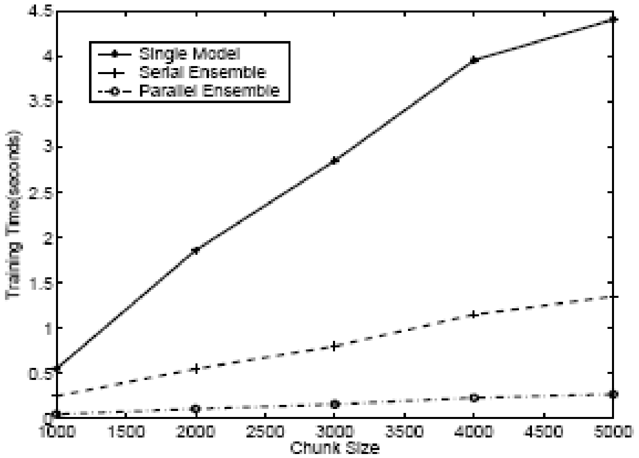


Figure 4: Training Time

Table 7: Logistic Regression as Base Learner

	SE	NS	SS
Synthetic1	**0.8801**	0.8363	0.8737
Synthetic2	**0.8992**	0.8102	0.8854
Thyroid1	**0.9977**	0.9774	0.9909
Thyroid2	**0.9949**	0.9593	0.9930
Opt	**0.9971**	0.9940	0.9953
Letter	**0.9545**	0.9448	0.9517
Covtype	**0.9995**	0.9989	0.9994

Mining Data Streams

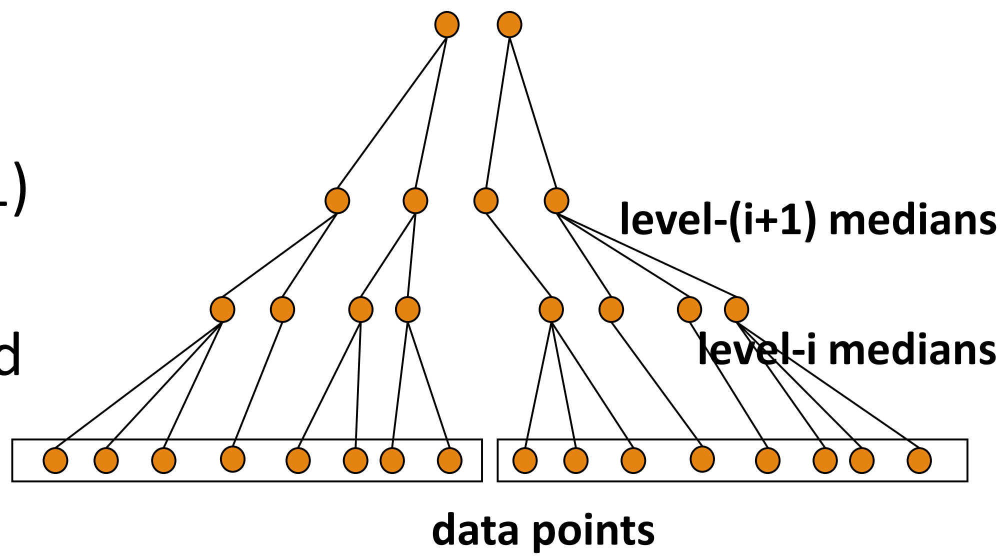
- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary

Stream Clustering: A K-Median Approach

- O'Callaghan et al. Streaming-Data Algorithms for High-Quality Clustering (ICDE'02)
- Base on the *k-median* method
  - Data stream points are from metric space
  - Find *k* clusters in the stream such that the sum of distances from data points to their closest centers is minimized
- A constant factor approximation algorithm
  - In small space, a simple two-step algorithm
  - 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
  - Let S' be centers for $S_1, \ldots, S_l$ with each center weighted by the number of points assigned to it
    - Cluster S' to find *k* centers

Hierarchical Clustering Tree

- **Hierarchical Clustering Tree 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



- Concerns:
  - Quality will suffer for evolving data streams (maintaining only *m* level-*i* medians)
  - Limited functionality in discovering and exploring clusters over different portions of the stream over time

CluStream: A Framework for Clustering Evolving Data Streams

- C. Aggarwal, J. Han, J. Wang, P. S. Yu, A Framework for Clustering Data Streams, VLDB'03
- Design goal of CluStream
  - High quality for clustering evolving data streams with rich functionality
  - Stream mining: One-pass over the stream data, limited space usage, high efficiency
- The CluStream Methodology
  - **Tilted time frame work**: otherwise, will lose dynamic changes
  - **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

Pyramidal Tilted Time Frame Adopted by CluStream

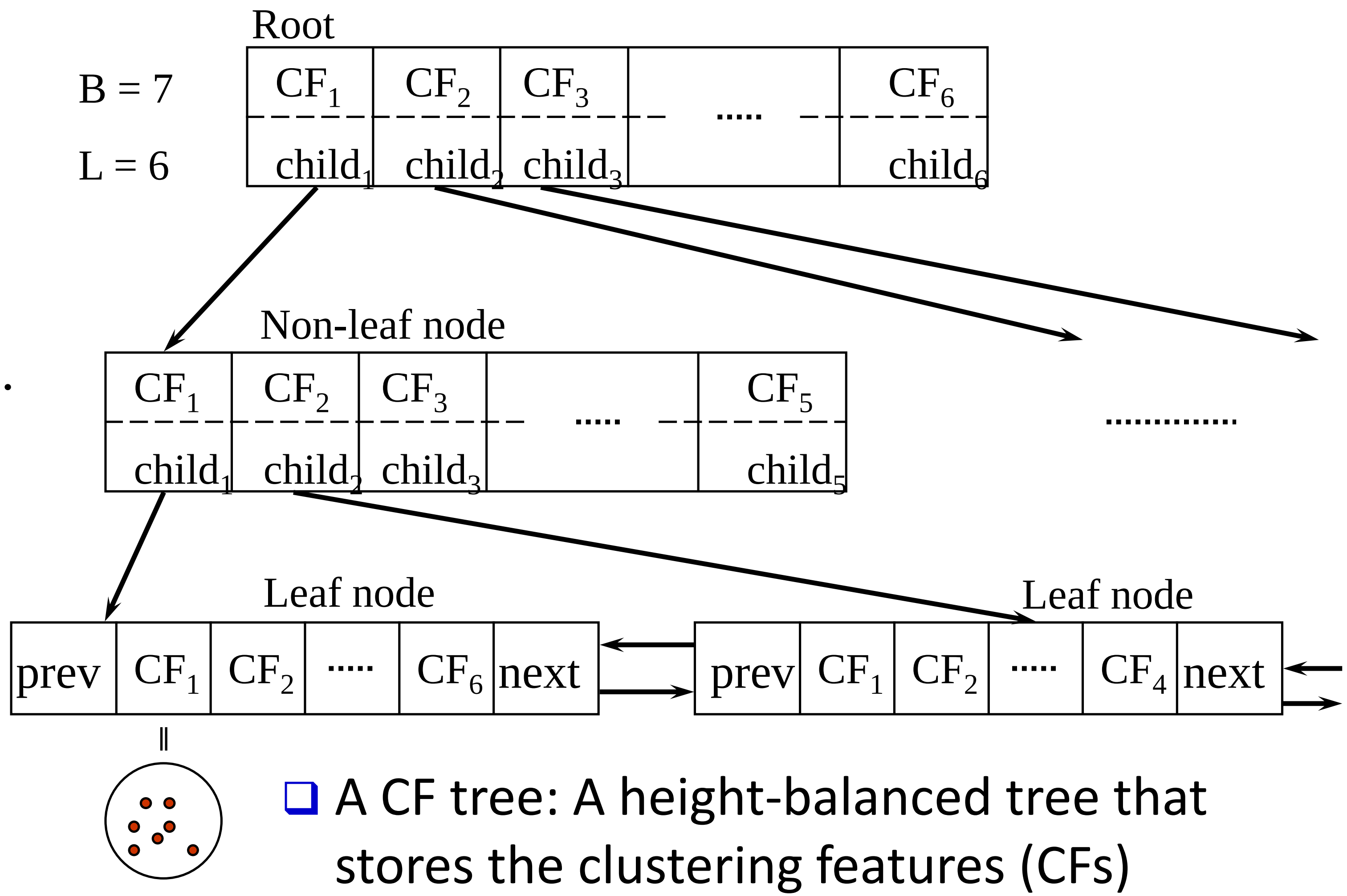
- **Pyramidal tilted time frame:**
  - Example: Suppose there are six frames (d = 5) and each takes a maximal of three snapshots
  - Given a snapshot number *N*
    - If *N* mod $2^d = 0$, insert into the frame number *d*
    - If there are more than three snapshots, eliminate the oldest one

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

- 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 (α + 1) snapshots are stored

The CluStream Framework: A Micro-Clustering Approach Using the BIRCH CF-Tree Structure

- Micro-clusters stored in CF-Tree
  - Statistical information about data locality
  - Temporal extension of the *cluster-feature vector* $\bar{X}_1 \dots \bar{X}_k \dots$
  - Multi-dimensional points with time stamps $T_1 \dots T_k \dots$
  - Each point contains *d* dimensions, i.e., $\bar{X}_i = \left(x_i^1 \dots x_i^d\right)$
  - A micro-cluster for *n* points is defined as a (2*d* + 3) tuple $\left(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n\right)$



- A CF tree: A height-balanced tree that stores the clustering features (CFs)
- The non-leaf nodes store sums of the CFs of their children

CluStream: Clustering Evolving On-Line Data Streams

- Divide the clustering process into *online* and *offline* components
  - **Online component (micro-cluster maintenance)**
    - Periodically store summary statistics about the stream data
      - Initially, create *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
        - Otherwise, create a new cluster
        - May delete obsolete micro-clusters or merge two closest ones
- **Offline component (query-based macro-clustering)**
  - Answers various user questions based on the stored summary statistics
  - Based on a user-specified time-horizon *h* and the number of macro-clusters *k*, compute macro-clusters using the *k*-means algorithm

Mining Data Streams

- What is stream data? stream data management systems? and stream data mining?
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Summary

Summary: Stream Data Mining

- Stream data mining and stream OLAP analysis:
  - Real life problem: Effectiveness, efficiency and scalability
- Stream OLAP
  - 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
- Stream data mining
  - Mining frequent patterns
  - Stream classification
  - Stream cluster analysis

References on Stream Data Mining (I)

- C. Aggarwal, J. Han, J. Wang, P. S. Yu, "A Framework for Clustering Data Streams", VLDB'03
- C. Aggarwal, J. Han, J. Wang, 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.
- 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, and R. Rastogi, "Processing Complex Aggregate Queries over Data Streams", SIGMOD'02
- J. Gehrke, F. Korn, and D. Srivastava, "On computing correlated aggregates over continuous data streams", SIGMOD'01
- J. Gao, W. Fan, and J. Han, "A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions", SDM'07

References on Stream Data Mining (II)

- 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’01
- S. Madden, M. Shah, J. Hellerstein, V. Raman, “Continuously Adaptive Continuous Queries over Streams”, SIGMOD’02
- 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”, Proc 2003 ACM-SIAM Symp. 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