CPT-S 415

Big Data

Yinghui Wu

EME B45

CPT-S 415
Big Data

Data stream processing

- ✓ Data stream management: Overview
- ✓ Data synopsis in data stream processing
- ✓ Incremental query processing

Streams – A Brave New World

- Traditional DBMS: data stored in *finite, persistent data sets*
- Data Streams: distributed, continuous, unbounded, rapid, time varying, noisy, . . .
- Data-Stream Management: variety of modern applications
  - Network monitoring and traffic engineering
  - Sensor networks
  - Telecom call-detail records
  - Network security
  - Financial applications
  - Manufacturing processes
  - Web logs and clickstreams
  - Other massive data sets…

Data stream processing and DSMS

The DSMS Research Field

- ✓ Active research field (~ 15 years) derived from the database community
  - Stream algorithms
  - Application and database perspective
- ✓ Two syllabus articles:
  - Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Jennifer Widom: "Models and issues in data stream systems"
  - Lukasz Golab, M. Tamer Ozsu: "Issues in data stream management”
- ✓ Future: Complex Event Processing (CEP)

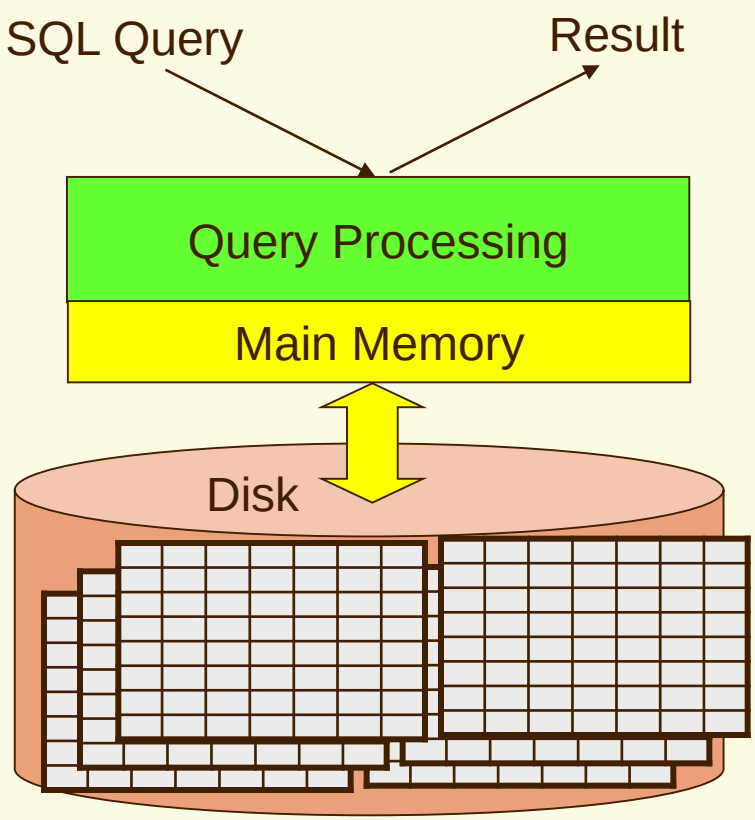
Data Streams - Terms

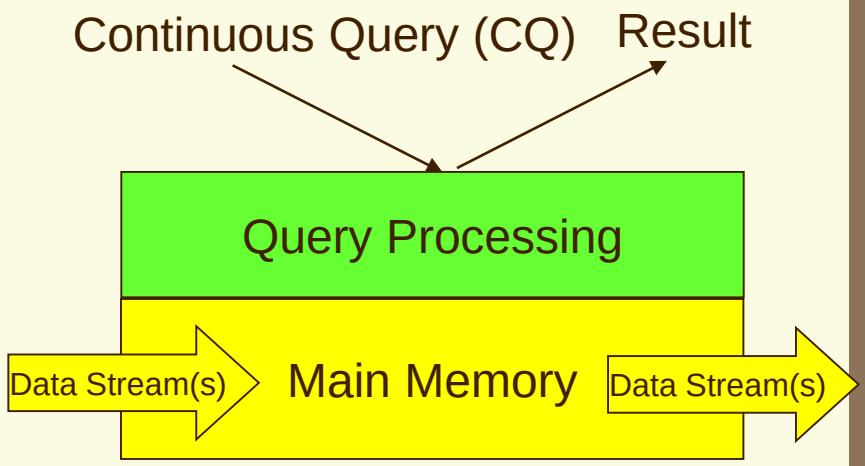
- A **data stream** is a (potentially unbounded) sequence of tuples
- Each tuple consist of a set of attributes, similar to a row in database table
- **Transactional** data streams: log interactions between entities
  - Credit card: purchases by consumers from merchants
  - Telecommunications: phone calls by callers to dialed parties
  - Web: accesses by clients of resources at servers
- **Measurement** data streams: monitor evolution of entity states
  - Sensor networks: physical phenomena, road traffic
  - IP network: traffic at router interfaces
  - Earth climate: temperature, moisture at weather stations

Motivation

- Massive data sets:
  - Huge numbers of users, e.g.,
    - AT&T long-distance: ~ 300M calls/day
    - AT&T IP backbone: ~ 10B IP flows/day
  - Highly detailed measurements, e.g.,
    - NOAA: satellite-based measurements of earth geodetics
  - Huge number of measurement points, e.g.,
    - Sensor networks with huge number of sensors
- Near real-time analysis
  - ISP: controlling service levels
  - NOAA: tornado detection using weather radar
  - Hospital: Patient monitoring
- Traditional data feeds
  - Simple queries (e.g., value lookup) needed in real-time
  - Complex queries (e.g., trend analyses) performed off-line

DBMS vs. DSMS #1





DBMS vs. DSMS #2

- Traditional DBMS:
  - stored sets of relatively static records with no pre-defined notion of time
  - good for applications that require persistent data storage and complex querying

DSMS:

support on-line analysis of rapidly changing data streams

data stream: real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items, too large to store entirely, not ending

continuous queries

DBMS vs. DSMS #3

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

- ✓ Transient streams
- ✓ Continuous queries
- ✓ Sequential access
- ✓ Bounded main memory
- ✓ History/arrival-order is critical
- ✓ Active stores
- ✓ Possibly multi-GB arrival rate
- ✓ Real-time requirements
- ✓ Data stale/imprecise
- ✓ Unpredictable/variable data arrival and characteristics

DSMS Applications

- Network security (e.g., iPolicy, NetForensics/Cisco, Niksun)
  - Network packet streams, user session information
  - URL filtering, detecting intrusions & DOS attacks & viruses

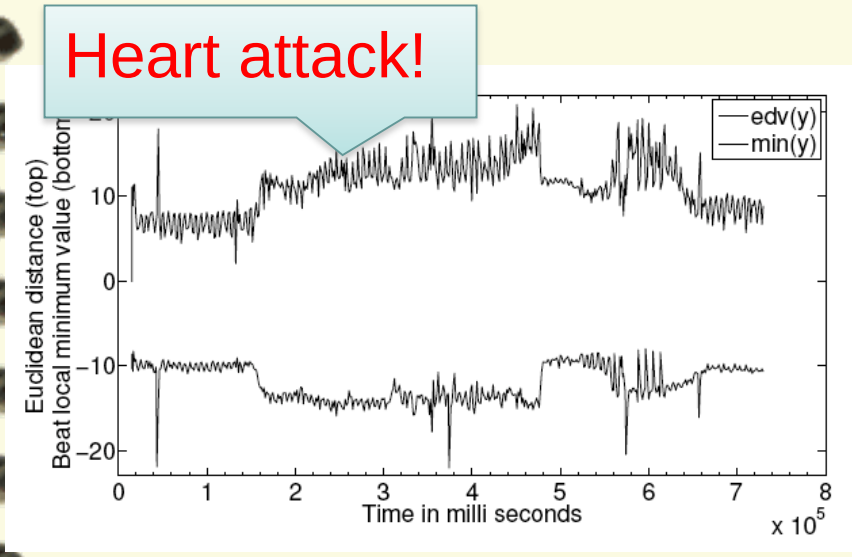
Pull-based

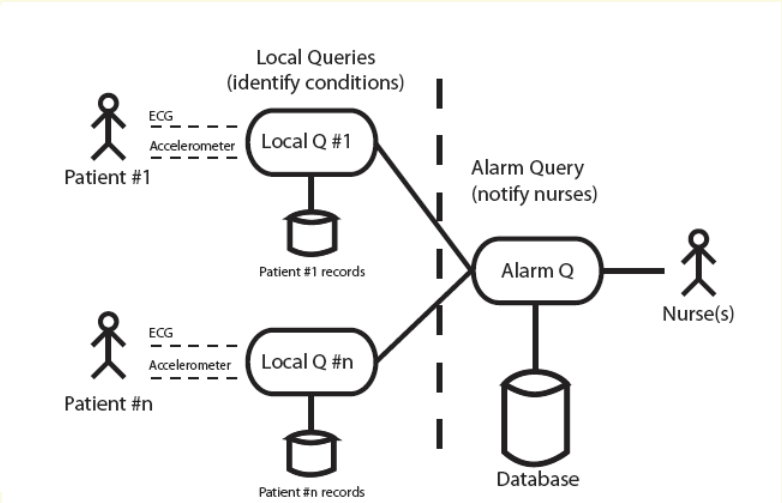
- Sensor Networks
  - Network Traffic Analysis
  - Real time analysis of Internet traffic. E.g., Traffic statistics and critical condition detection.
- Financial Tickers
  - On-line analysis of stock prices, discover correlations, identify trends.
- Transaction Log Analysis
  - e.g. Web click streams and telephone calls

Push-based

Application

- ✓ Stig Støa, Morten Lindeberg and Vera Goebel. *Online Analysis of Myocardial Ischemia From Medical Sensor Data Streams with Esper,* 2009
- ✓ Queries over sensor traces from surgical procedures on pigs performed at IVS, Rikshospitalet, running a open source java system called Esper
- ✓ Successful identification of occlusion to the heart (heart attack)






```
SELECT y, timestamp
FROM Accelerometer.win:ext_timed(t, 5 s)
HAVING count(y) BETWEEN 2 AND 200
```

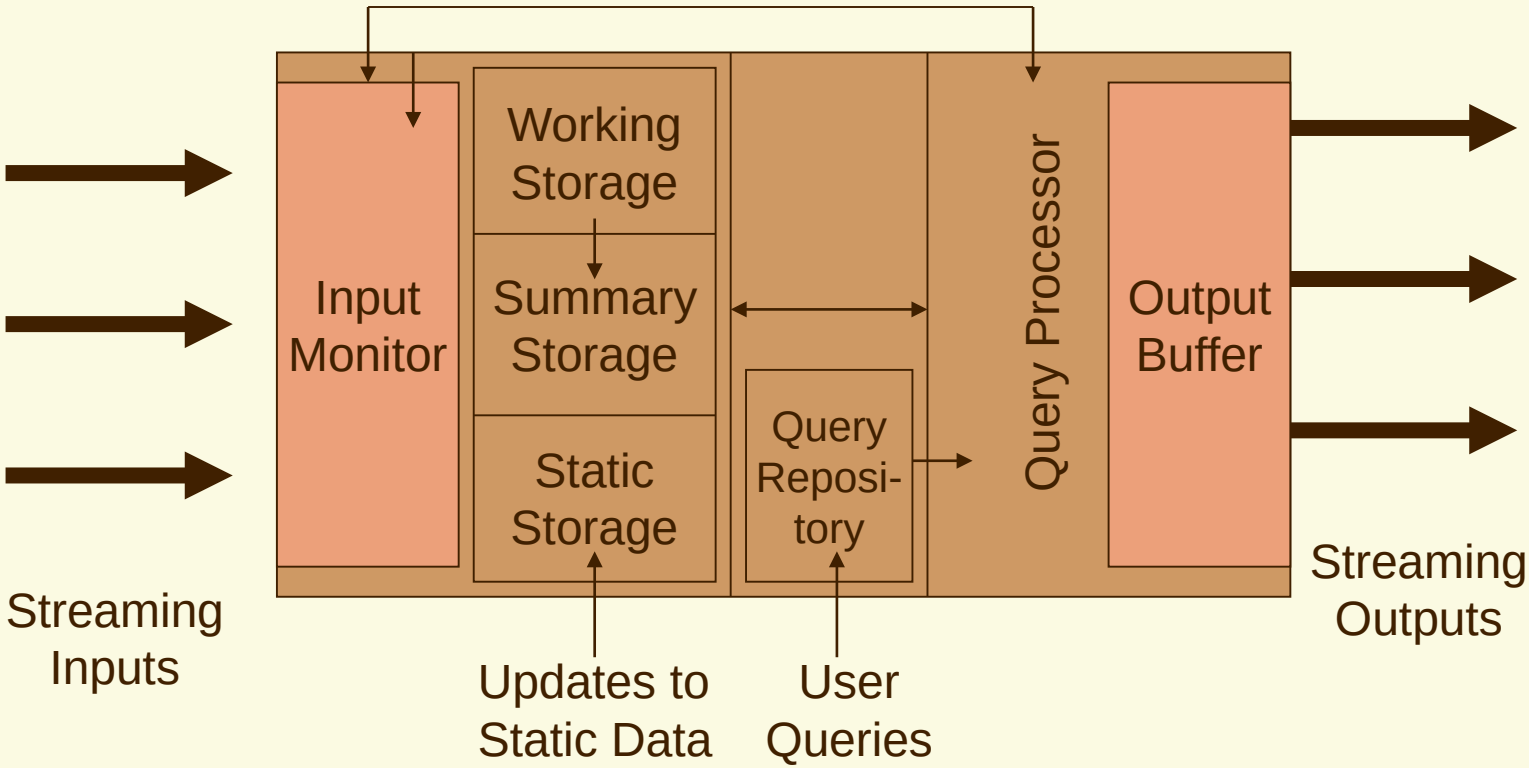
What do we need?

- ✓ Data model and query semantics: order- and time-based operations
  - Selection
  - Nested aggregation
  - Multiplexing and demultiplexing
  - Frequent item queries
  - Joins
  - Windowed queries
- ✓ Query processing:
  - Streaming query plans must use non-blocking operators
  - Only single-pass algorithms over data streams
- ✓ Data reduction: approximate summary structures
  - Synopses, digests => no exact answers
- ✓ Real-time reactions for monitoring applications => active mechanisms
- ✓ Long-running queries: variable system conditions
- ✓ Scalability: shared execution of many continuous queries, monitoring multiple streams

Concepts and issues: Architecture

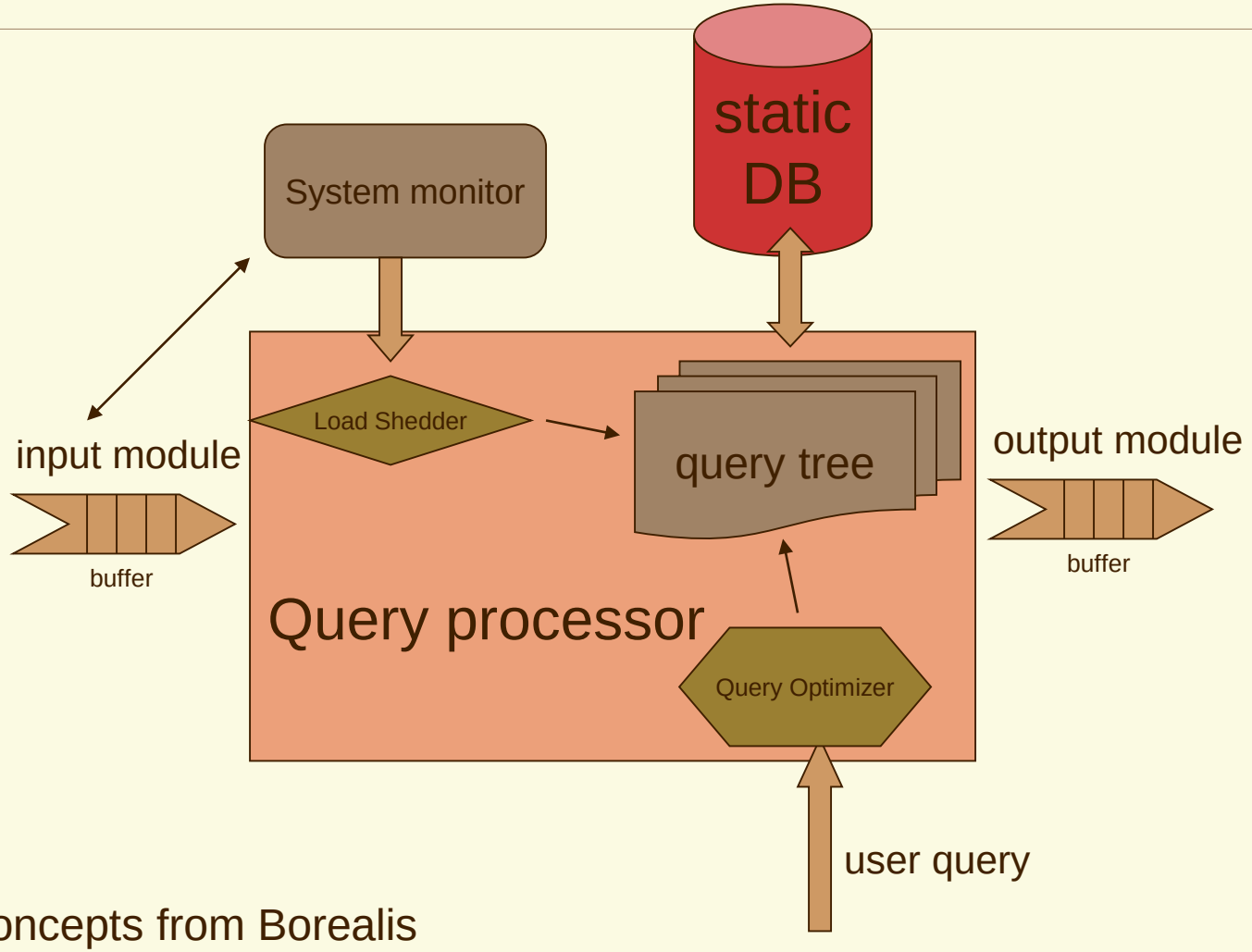
Courtesy (slides adapted) from Vera Goebel

Generic DSMS Architecture



[Golab & Özsu 2003]

A closer look at stream query engine

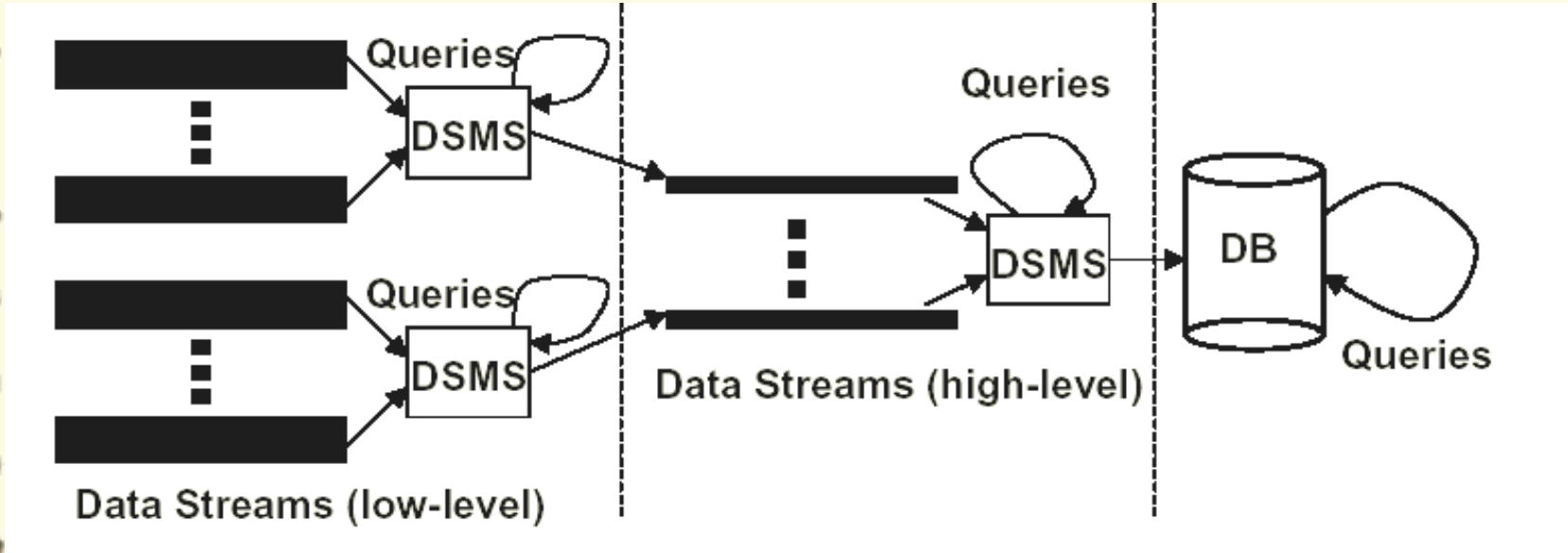


Concepts from Borealis

"Load Shedding techniques for data stream systems, B.Babcock, et.al 2003"

3-Level Architecture

- ✓ Reduce tuples through several layered operations (several DSMSs)
- ✓ Store results in static DB for later analysis



Data models

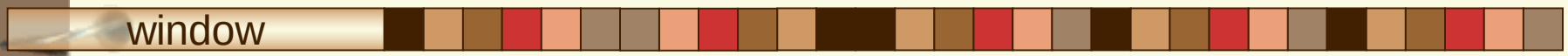
Data Models

- ✓ Real-time data stream: sequence of items that arrive in some order and may only be seen once.
- ✓ Stream items: like relational tuples
  - Relation-based: e.g., STREAM, TelegraphCQ and Borealis
  - Object-based: e.g., COUGAR, Tribecca
- ✓ Window models
  - Direction of movements of the endpoints: fixed window, sliding window, landmark window
  - Time-based vs. Tuple-based
  - Update interval: eager (for each new arriving), lazy (batch processing), non-overlapping tumbling windows.

More on Windows

- ✓ Mechanism for extracting a finite relation from an infinite stream
- ✓ Desirable query semantic; data freshness; blocking operators..

Sliding:



Jumping:

Overlapping

(adapted from Jarle Søberg)

Timestamps

- ✓ Used for tuple ordering and by the DSMS for defining window sizes (time-based)
- ✓ Useful for the user to know when the the tuple originated
- ✓ Explicit: set by the source of data
- ✓ Implicit: set by DSMS, when it has arrived
- ✓ Ordering is an issue
- ✓ Distributed systems: no exact notion of time

Stream Queries

Queries: DBMS vs DSMS

- ✓ DBMS: one-time (transient) queries
  DSMS: continuous (persistent) queries
- ✓ DBMS: Arbitrary data access
  DSMS: one-pass algorithms
- ✓ DBMS: Unbounded memory requirements
  DSMS: Limited memory/Cache; window techniques
- ✓ DBMS: (mostly) exact query answer
  DSMS: (mostly) approximate query answer
  - We introduced approximate query processing
    - sampling, synopses, sketches, wavelets, histograms, …
    - apply over data streams?
- ✓ DBMS: fixed query plans optimized at beginning
  DSMS: adaptive query operators

Query Optimization

- DBMS: table based cardinalities used in query optimization
  => Problematic in a streaming environment
- Cost metrics and statistics: accuracy and reporting delay vs. memory usage, output rate, power usage
- Query optimization: query rewriting to minimize cost metric, adaptive query plans, due to changing processing time of operators, selectivity of predicates, and stream arrival rates
- Query optimization techniques
  - stream rate based, resource based, QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability

Selections and Projections

- Selections, (duplicate preserving) projections are straightforward
  - Local, per-element operators
  - Duplicate eliminating projection is like grouping
- Projection needs to include ordering attribute
  - No restriction for position ordered streams

```
SELECT sourceIP, time
FROM Traffic
WHERE length > 512
```

Joins

- General case of join operators problematic on streams
  - May need to join arbitrarily far apart stream tuples
  - Equijoin on stream ordering attributes is tractable
- Majority of work focuses on joins between streams with windows specified on each stream

```
SELECT A.sourceIP, B.sourceIP
FROM Traffic1 A [window T1], Traffic2 B [window T2]
WHERE A.destIP = B.destIP
```

Aggregations

- General form:
  - **select** G, F1 **from** S **where** P **group by** G **having** F2 op ϑ
  - G: grouping attributes, F1,F2: aggregate expressions
  - Window techniques are needed

- Aggregate expressions:
  - distributive: sum, count, min, max
  - algebraic: avg
  - holistic: count-distinct, median

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: hierarchical decomposition
- ✓ Histograms: approximate frequency of element values in stream

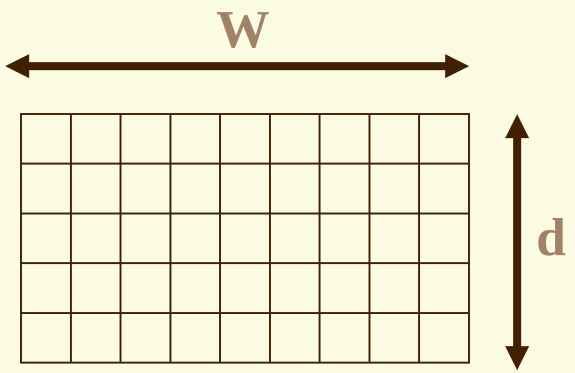
Holistic Aggregates: a case study

Holistic Aggregates

- Holistic aggregates need the whole input to compute (no summary suffices)
  - E.g., `count distinct`, need to remember all distinct items to tell if new item is distinct or not
- So focus on approximating aggregates
  - Adopt ideas from sampling, data reduction, streams etc.
- Aggregate approximation:
  - Sketch summaries
  - Other mergable summaries
  - Building uniform samples, etc…

CM Sketch

- ✓ Count-Min Sketch: can be used for point queries, range queries, quantiles, join size estimation.
- ✓ Model input as a vector x_i of dimension U (U is large)
- ✓ Creates a small summary as an array of $w \times d$ in size
- ✓ Use d hash function to map vector entries to $[1..w]$



Data stream model

Consider the vector $\vec{a}(t) = (a_1(t), \ldots, a_i(t), \ldots a_n(t))$

initially $a_i(0) = 0 \quad \forall i$

The tth update (i_t, c_t)

$$a_{i'}(t) = a_{i'}(t-1) \quad \forall i' \neq i_t$$

$$a_{i_t}(t) = a_{i_t}(t-1) + c_t$$

Count-Min Sketch

A Count-Min (CM) Sketch with parameters (ε, δ) is represented by a two-dimensional array counts with width w and depth $d : count[1,1] \ldots count[d,$

Given parameters (ε, δ), set $w = \left\lceil \frac{e}{\varepsilon} \right\rceil$ and $d = \left\lceil \ln \frac{1}{\delta} \right\rceil$.

Each entry of the array is initially zero.

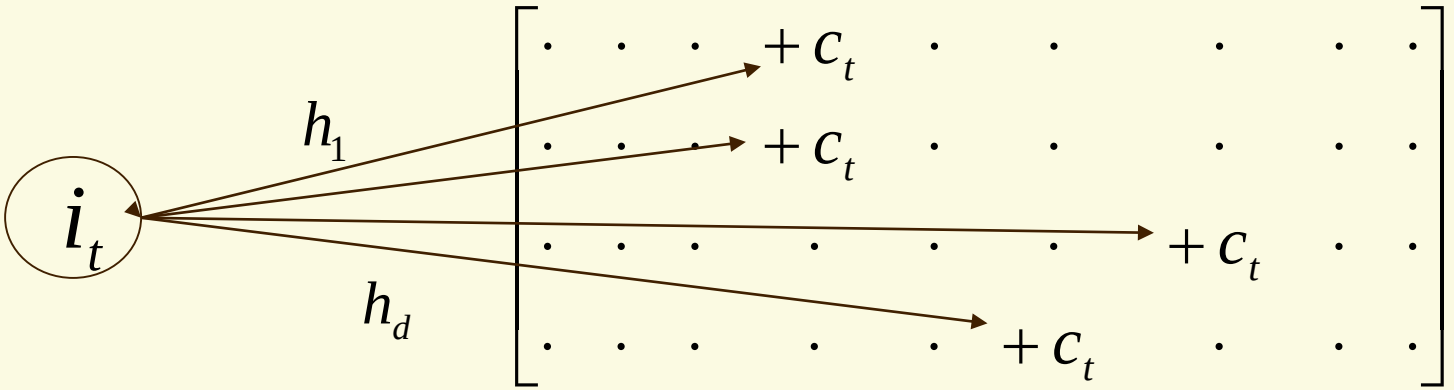
d hash functions are chosen uniformly at random from a pairwise independent family

$$h_1, \ldots, h_d : \{1 \ldots n\} \rightarrow \{1 \ldots w\}$$

Update procedure

When (i_t, c_t) arrives, set $\forall 1 \le j \le d$

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t$$



Approximate Query Answering Using CM Sketches

- point query $Q(i)$ approx. $\Rightarrow$ a_i

- range queries $Q(l,r)$ approx. $\Rightarrow$ $\sum_{i=l}^{r} a_i$

- inner product queries $Q(\vec{a},\vec{b})$ approx. $\Rightarrow$ $\vec{a}\cdot\vec{b} = \sum_{i=1}^{n} a_i b_i$

Approximate Query Answering Using CM Sketches

- point query $Q(i)$ approx. a_i

$$\hat{a}_i = \min_j count[j, h_j(i)] \quad \text{and} \quad P[\hat{a}_i > a_i + \varepsilon \|\vec{a}\|_1] \leq \delta$$

- range queries $Q(l, r)$ approx. $\sum_{i=l}^{r} a_i$

$$= \hat{a}[l, r] \text{ from } 2\log_2 n \text{ point queries}$$

- inner product queries $Q(\vec{a}, \vec{b})$ approx. $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i$

$$(\vec{a} \cdot \vec{b}) = \min_j (\vec{a} \cdot \vec{b})_j \quad \text{where} \quad (\vec{a} \cdot \vec{b})_j = \sum_{k=1}^{w} count_{\vec{a}}[j,k] * count_{\vec{b}}[j,k]$$

Sketch Summary

Sketches	Space	Update Time	Query Time
Count-Min	1/ε ; O(K)	1	1
Bloom Filter	m ; Constant	k	k
Count Bloom Filter	mC; O(m)	k	k
Multi Counting Bloom filter	mC; O(m)	k	k
FM	ML O(M)	M	M

C = Number of Bits in the Counter in Bloom Filter
M = Number of Bit Maps used in FM Sketch
L = Number of Bits in FM Sketch
All Notations Described Earlier

Incremental query processing

40

Incremental query answering

- ✓ Real-life data is dynamic – constant changes
- ✓ Re-compute $Q(D \oplus \Delta D)$ starting from scratch?
- ✓ Changes ΔD are typically small

5%/week in Web graphs

Compute $Q(D)$ once, and then incrementally maintain it

Incremental query processing:

- ✓ Input: Q, D, $Q(D)$, ΔD
- ✓ Output: ΔM such that $Q(D \oplus \Delta D) = Q(D) \oplus \Delta M$

Old output

Changes to the input

New output

Changes to the output

When changes ΔD to the data D are small, typically so are the changes ΔM to the output $Q(D \oplus \Delta D)$

Minimizing unnecessary recomputation

Why study incremental query answering?

Incremental query answering

- ✓ Input: Q, D, $Q(D)$, ΔD
- ✓ Output: ΔM such that $Q(D \oplus \Delta D) = Q(D) \oplus \Delta M$

- ✓ E-commerce systems: a fixed set of (parameterized) queries
  - – Repeatedly invoked and evaluated
  - ✓ View maintenance: in response to changes to the underlying graph
  - ✓ Data synopsis: maintenance in the presence of changes
  - ✓ Indexing structure: 2-hop covers; landmarks…

An important issue for querying big data

Complexity of incremental problems

Incremental query answering

- ✓ Input: Q, D, Q(D), ΔD
- ✓ Output: ΔM such that $Q(D \oplus \Delta D) = Q(D) \oplus \Delta M$

G. Ramalingam, Thomas W. Reps: *On the Computational Complexity of Dynamic Graph Problems*. TCS 158(1&2), 1996

The cost of query processing: a function of |D| and |Q|

- ✓ incremental algorithms: |CHANGED|, the size of changes in
  - the input: ΔG, and
  - the output: ΔM

The updating cost that is inherent to the incremental problem itself

- ✓ Bounded: the cost is expressible as f(|CHANGED|, |Q|)?
- ✓ Optimal: in O(|CHANGED| + |Q|)?

The amount of work absolutely necessary to perform

Complexity analysis in terms of the size of changes

Incremental graph pattern matching

Complexity of incremental algorithms

- ✓ Result graphs
  - Union of isomorphic subgraphs for subgraph isomorphism
  - A graph Gr = (Vr, Er) for (bounded) simulation
    - Vr : the nodes in G matching pattern nodes in Gp
    - Er: the paths in G matching edges in Gp
- ✓ Affected Area (AFF)
  - the difference between Gr and the result graph of G and
  - respectively.
  - |CHANGED| = |ΔG| + |AFF|
- ✓ Optimal, bounded and unbounded problem
  - expressible by f(|CHANGED|)?

Measure the complexity with the size of changes

Incremental simulation matching

- ✓ Input: Q, G, $Q(G)$, ΔG
- ✓ Output: ΔM such that $Q(G \oplus \Delta G) = Q(G) \oplus \Delta M$
- ✓ Incremental simulation is in

Batch updates

$$O(|\Delta G|(|Q||AFF| + |AFF|^2)) \text{ time}$$

in $O(|AFF|)$ time

unbounded

- ✓ Optimal for
  - single-edge deletions and general patterns
  - single-edge insertions and DAG patterns

General patterns and graphs; batch updates

2 times faster than its batch counterpart for changes up to 10%

Semi-boundedness

- ✓ Semi-bounded: the cost is a PTME function f(|CHANGED|, |Q|)
  - ✓ Incremental simulation is in

| Q | is small

$O(|\Delta G|(|Q||AFF| + |AFF|^2))$ time

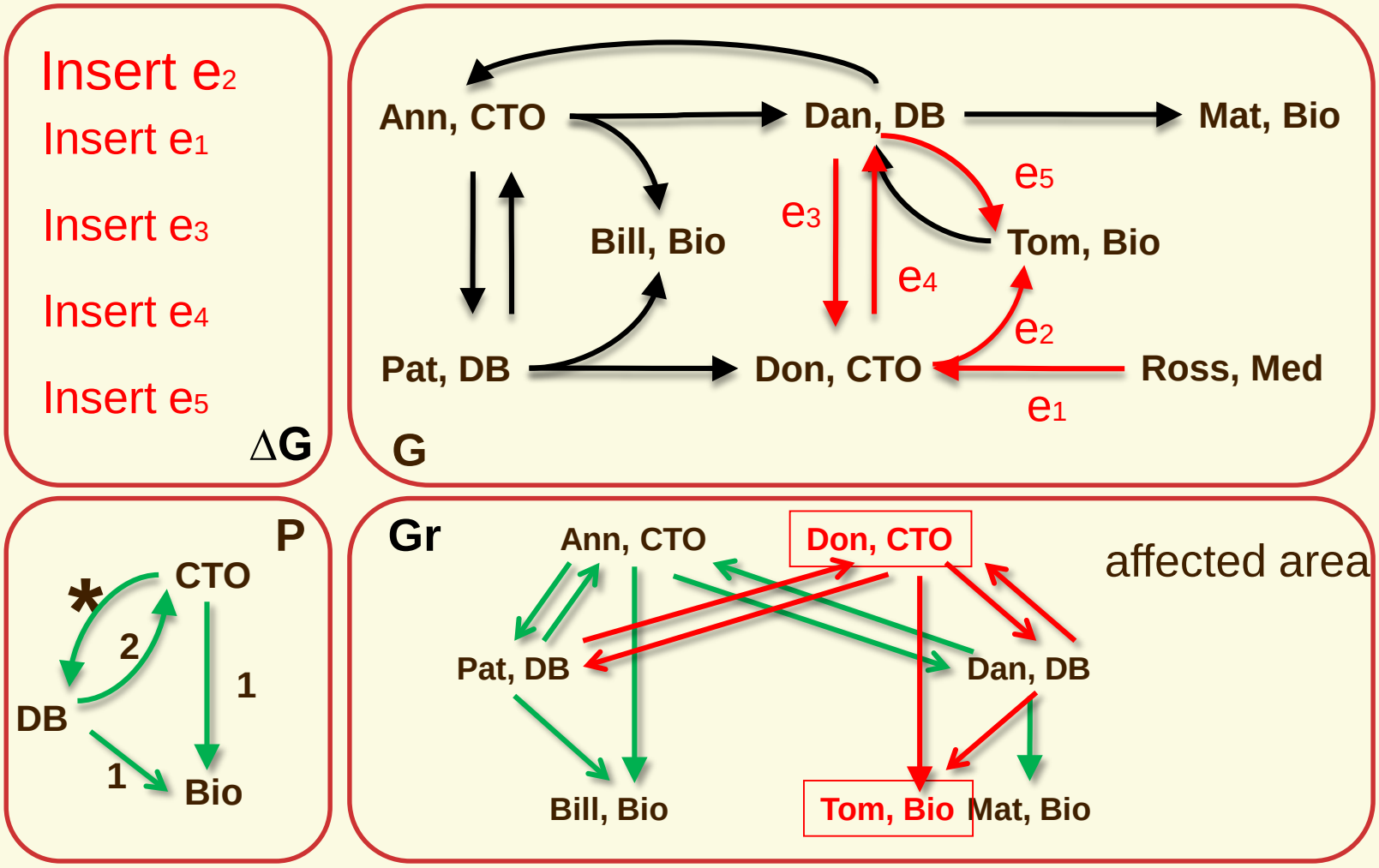
for batch updates and general patterns

Independent of | G |

Independent of | G |

Semi-boundedness is good enough!

Complexity of incremental algorithms (cont)



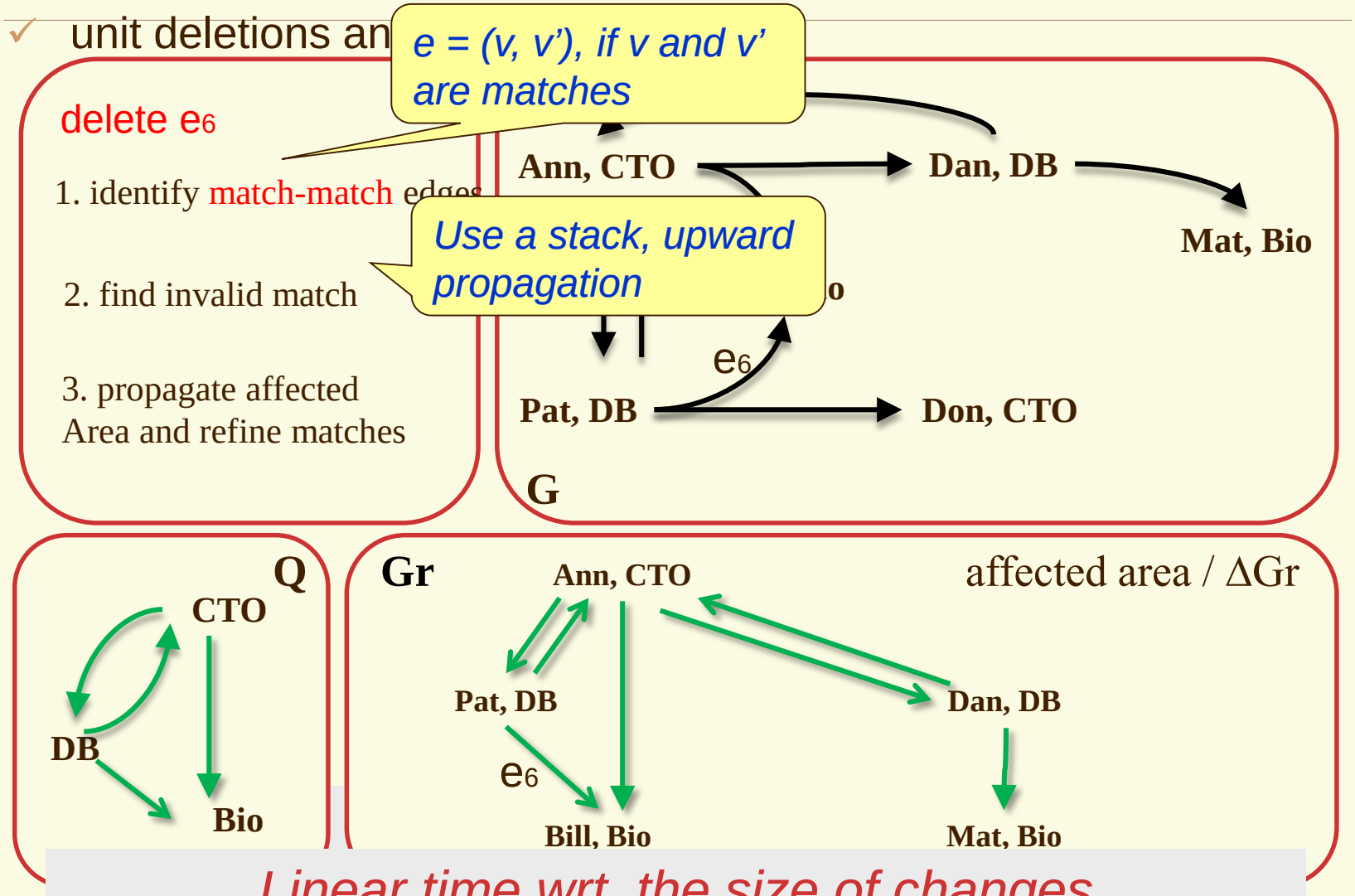
Incremental Simulation matching

- ✓ Problem statement
  - Input: Gp, G, Gr, ΔG
  - Output: ΔGr, the updates to Gr s.t. $M_{sim}(G \oplus \Delta G) = M(Gp,G) \oplus \Delta M$
- ✓ Complexity
  - unbounded even for unit updates and general patterns
  - bounded for single-edge deletions and general patterns
  - bounded for single-edge insertions and DAG patterns, within optimal time O(|AFF|)
  - In $O(|\Delta G|(|Gp||AFF| + |AFF|^2))$ for batch updates and general patterns

Measure the complexity with the size of changes

Incremental Simulation: optimal results

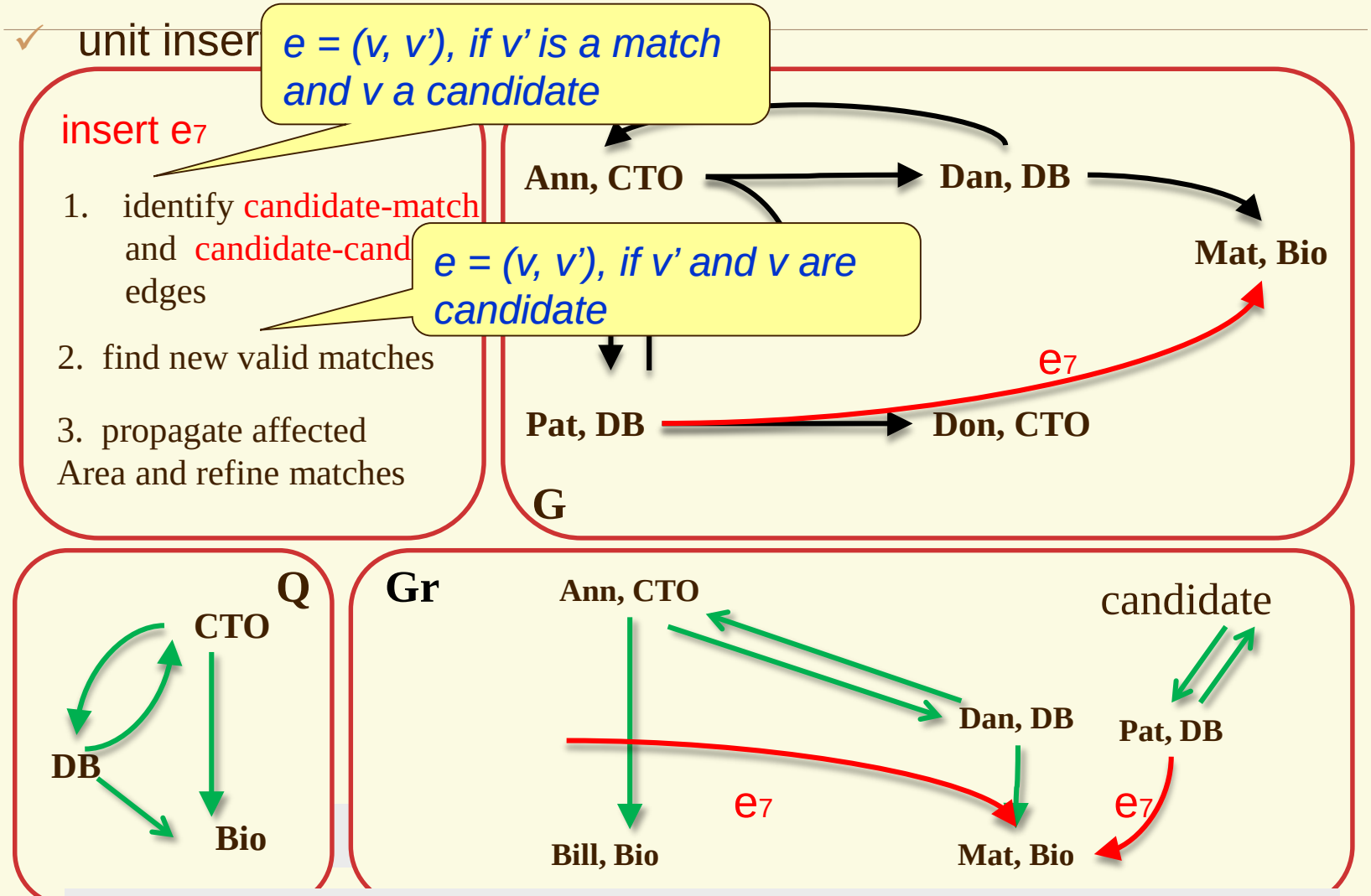
- ✓ unit deletions an



Linear time wrt. the size of changes

Incremental Simulation: optimal results

- unit inser



Linear time wrt. the size of changes

Incremental subgraph isomorphism

- ✓ Incremental subgraph isomorphism matching:
  - **Input: Gp, G, Gr, ∆G**
  - **Output: ∆Gr, the updates to Gr s.t.**

    Miso(G⊕∆G) = M_{iso}(Gp,G)⊕∆M
- ✓ Incremental subgraph isomorphism:
  - **Input: Gp, G, Gr, ∆G**
  - **Output: true if there is a subgraph in G⊕∆G that is isomorphi**

    = M_{iso}(Gp,G)⊕∆M
- ✓ Complexity
  - IncIsoMatch is unbounded even for unit updates over DAG graphs for path patterns
  - IncIso is NP-complete even for path pattern and unit update

Incremental subgraph isomorphism

- ✓ Input: Q, G, $M_{iso}(Q, G)$, ΔG
- ✓ Output: ΔM such that $M_{iso}(Q, G \oplus \Delta G) = M_{iso}(Q, G) \oplus \Delta M$
- ✓ Boundedness and complexity
  - Incremental matching via subgraph isomorphism is unbounded even for unit updates over DAG graphs for path patterns
  - Incremental subgraph isomorphism is NP-complete even when G is fixed

not semi-bounded unless P = NP

- ✓ Input: Q, G, $M(Q, G)$, ΔG
- ✓ Question: whether there exists a subgraph in $G \oplus \Delta G$ that is isomorphic to Q

What should we do?

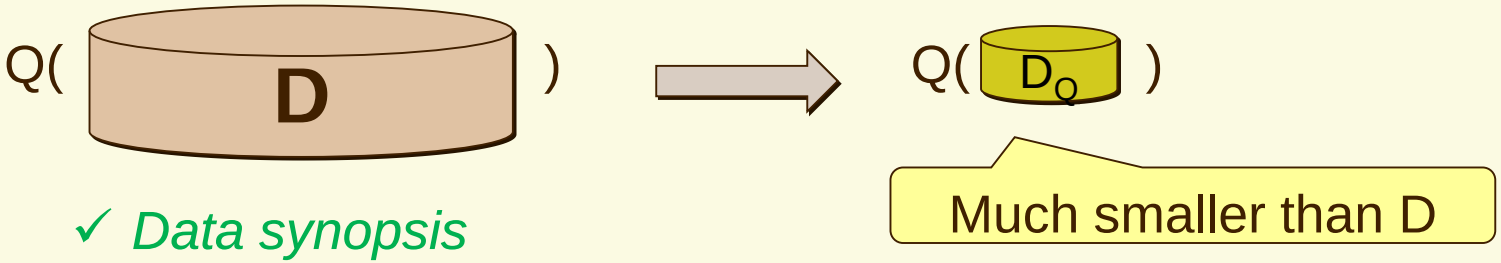
Neither bounded nor semi-bounded

Summary

- ✓ Data stream processing: Overview
- ✓ Incremental query processing
- ✓ Data synopsis in data stream processing

How to make big data small

- Input: A class $\mathcal{Q}$ of queries
- Question: Can we effectively find, given queries $Q \in \mathcal{Q}$ and any (possibly big) data D, a small D_Q such that
  - $Q(D) = Q(D_Q)$?



- ✓ *Data synopsis*
- ✓ *Boundedly evaluable queries*
- ✓ *Query answering using views*
- ✓ *Incremental evaluation*
- ✓ *Distributed query processing*

Effective methods for making big data small