Big Data: Theory and Practice II

CPT-S 415 Big Data

Big Data: Theory and Practice

- ✓ Theory
  - Tractability revisited for querying big data
  - Parallel scalability
  - Bounded evaluability
- ✓ Techniques
  - Parallel algorithms
  - Bounded evaluability and access constraints
  - Query-preserving compression
  - Query answering using views
  - Bounded incremental query processing

Techniques for querying big data: review

A general approach to querying big data

Given a query Q, an access schema $\mathcal{A}$ and a big dataset D

1. Decide whether Q is effectively bounded under $\mathcal{A}$
2. If so, generate a bounded query plan for Q
3. Otherwise, do one of the following:
   - ① Ext… ake Q
   - ②
   - ③

- ✓ *77% of conjunctive queries are boundedly evaluable*
- ✓ *Efficiency: 9 seconds vs. 14 hours of MySQL*
- ✓ *60% of graph pattern queries are boundedly evaluable (via subgraph isomorphism)*
- ✓ *Improvement: 4 orders of magnitudes*

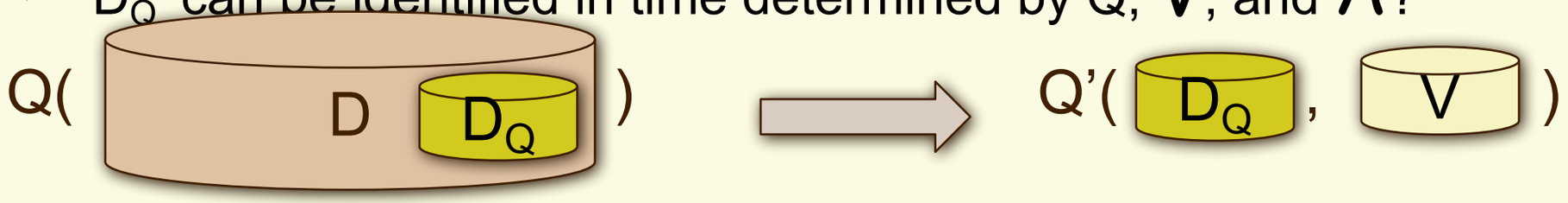
Very effective for conjunctive queries

Strategy 1: make best use of views

Bounded evaluability using views

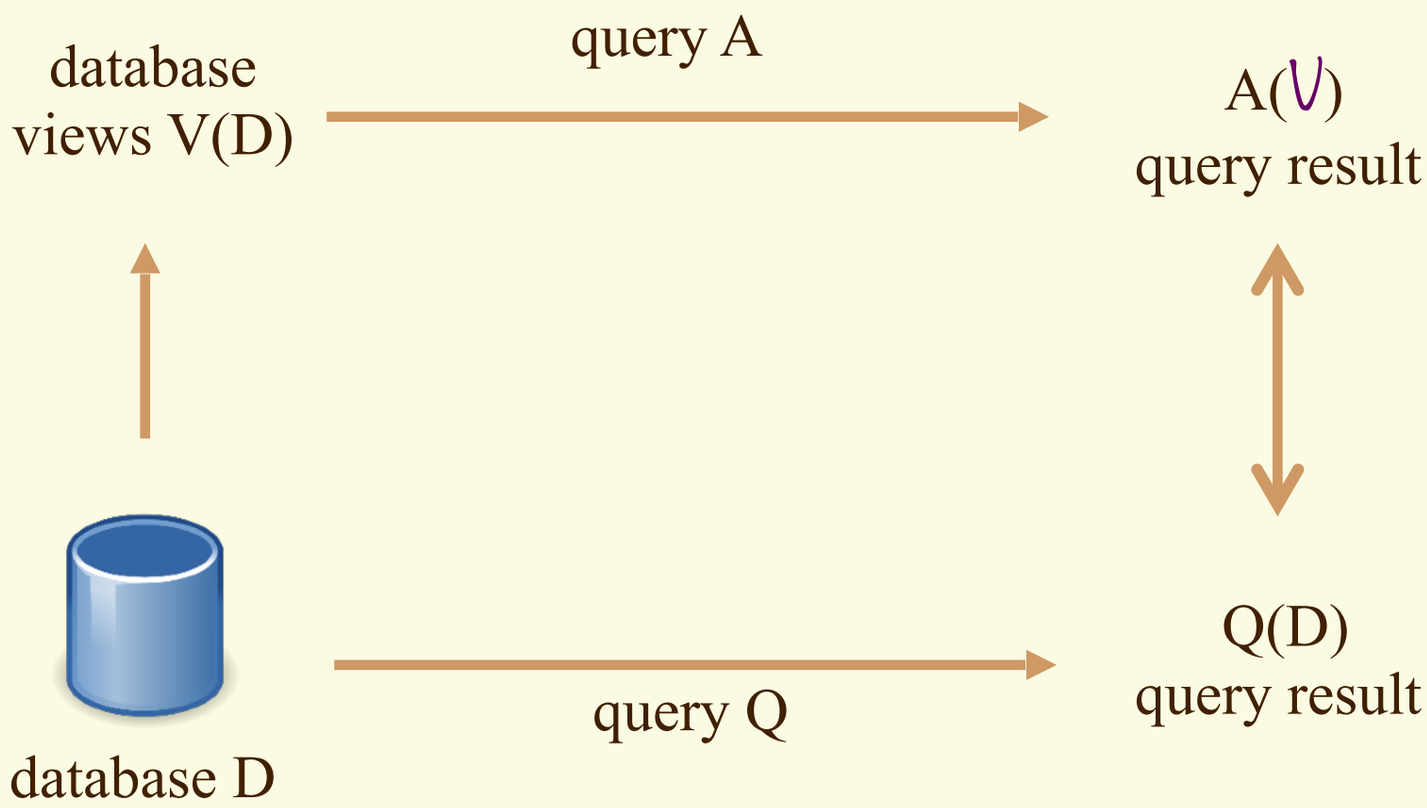
- Input: A class $\mathcal{Q}$ of queries, a set of views $\mathcal{V}$, an access schema $\mathcal{A}$
- Question: Can we find by using $\mathcal{A}$, for any query $Q \in \mathcal{Q}$ and any (possibly big) dataset D, a fraction D_Q of D such that
  - $|D_Q| \leq M$,
  - a rewriting Q' of Q using $\mathcal{V}$,
  - $Q(D) = Q'(D_Q, \mathcal{V}(D))$, and
  - D_Q can be identified in time determined by Q, $\mathcal{V}$, and $\mathcal{A}$?

access views, and additionally a bounded amount of data



Query Q may not be boundedly evaluable, but may be boundedly evaluable with views!

Answering query using views

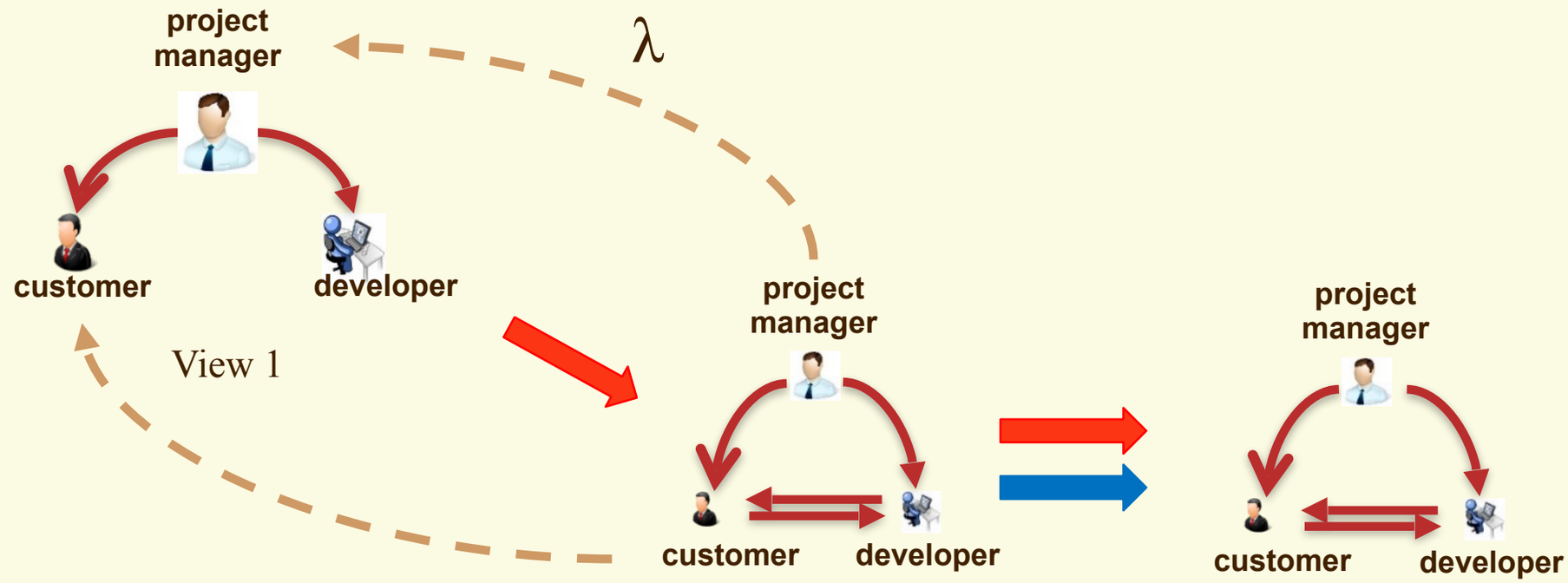


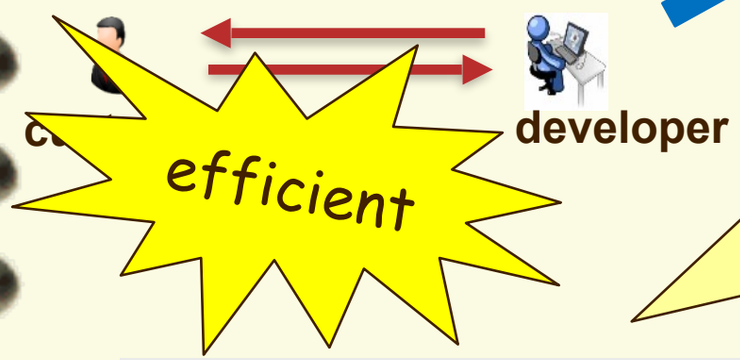
When possible?
Query containment

What to choose?
Maximal containment
Minimum containment

How to evaluate?
Query rewriting

Query containment: example





Query containment: given Q and Q', it is to determine whether for any graph G, Q(G) is contained in Q'(G)?
A classical problem.
What is its complexity for pattern queries?

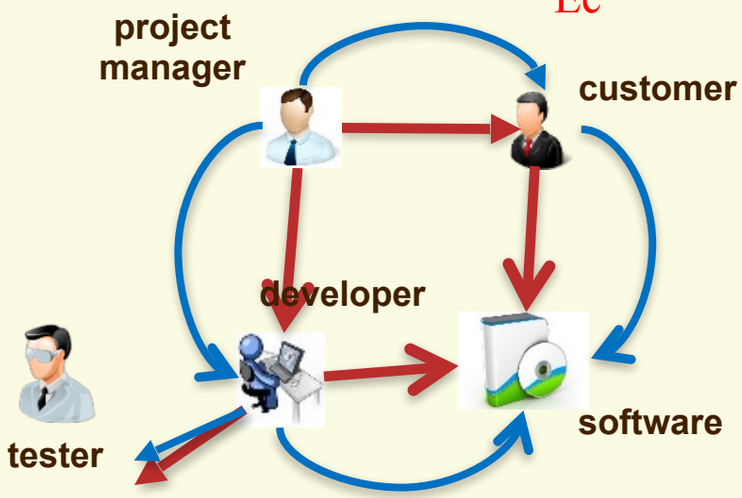
$Q \sqsubseteq \mathcal{V}$ *can be determined in* $O(card(\mathcal{V})|Q|^2 + |\mathcal{V}|^2 + |Q||\mathcal{V}|)$ *time*

Minimum containment: example

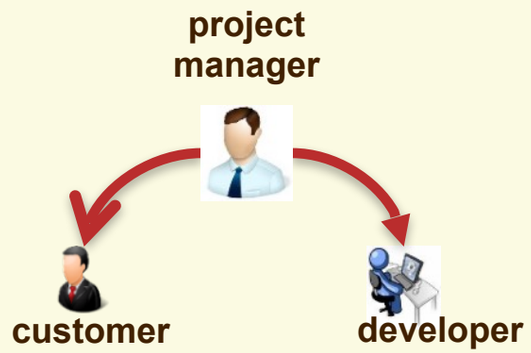
$\alpha(\text{view } 1) = 0.4$

$\alpha(\text{view } 1) = 0$

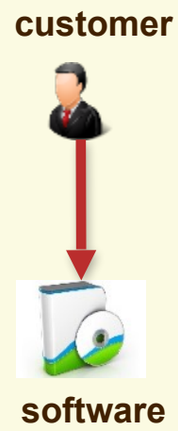
Ec



query

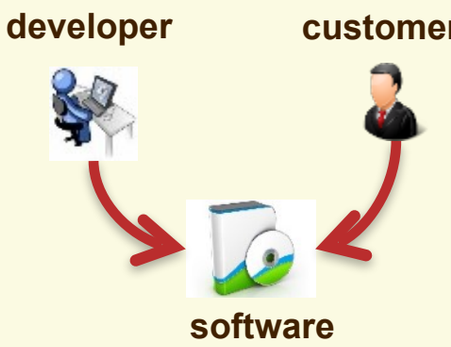


view 1

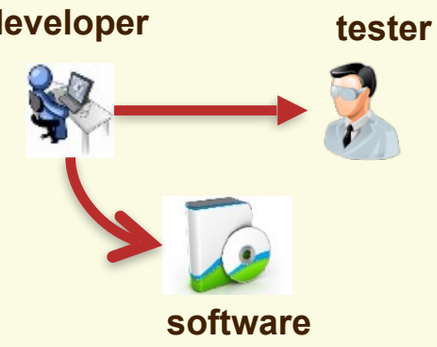


view 2

$\alpha(\text{view } 4) = 0.4$

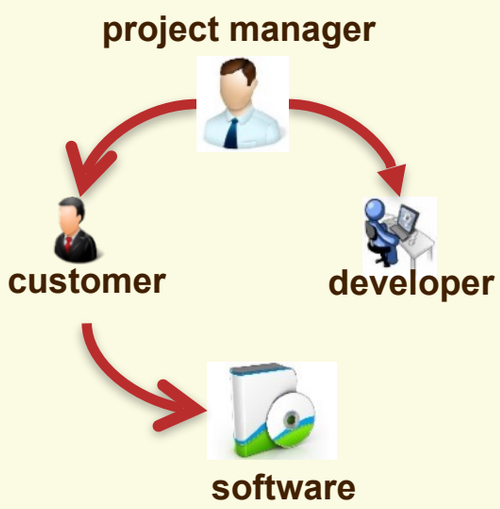


view 3

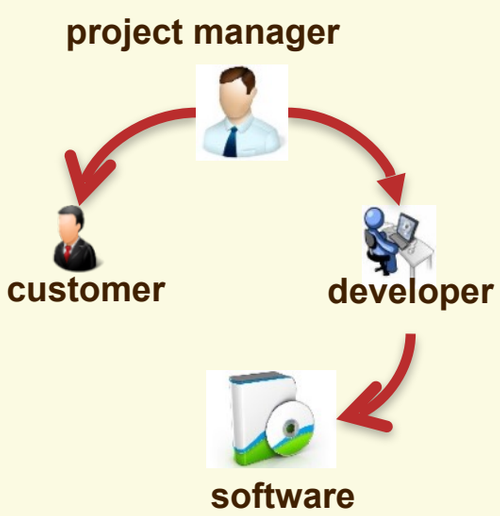


view 4 ✓

$\alpha(\text{view } 6) = 0.6$



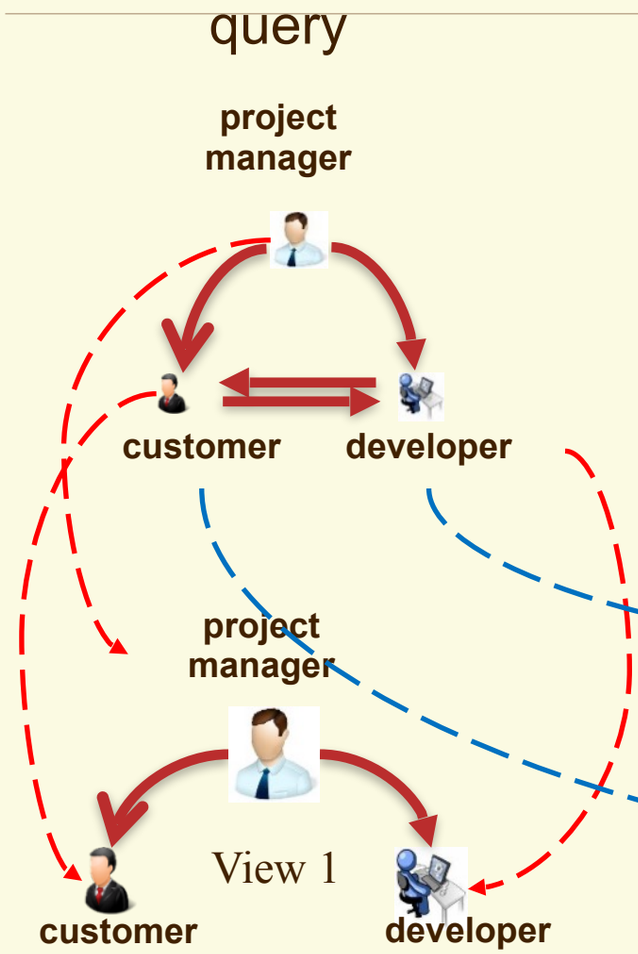
view 5 ✓



view 6

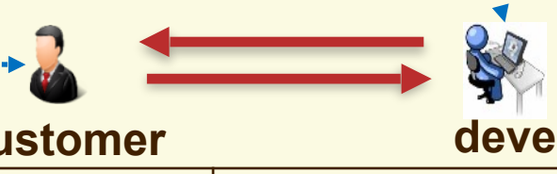
Greedy: based on the metric

Query evaluation using views: example



(project manager, developer)	"bottom-up" strategy (PM 2, developer 3)}
(project manager, customer)	{(PM 1, customer 2), (PM 2, customer 2)}
(developer, customer)	{(developer 2, customer 2), (developer 2, customer 3), (developer 3, customer 2)}
(customer, developer)	{(customer 2, developer 2), (customer 3, developer 3)}

Query result



(project manager, developer)	{(PM 1, developer 2), (PM 2, developer 3)}
(project manager,	

(customer, developer)	{(customer 2, developer 2), (customer 3, developer 3)}
	{(developer 2, customer 2), (developer 2, customer 3),

4% -- 12% of G

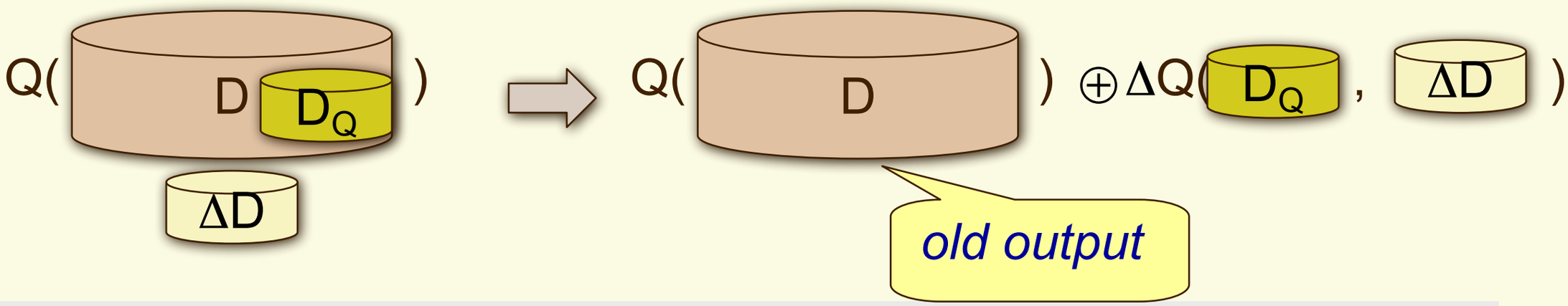
Without accessing the underlying big graph G

Strategy 2: do only necessary computation for data updates

Incremental bounded evaluability

- Input: A class $\mathcal{Q}$ of queries, an access schema $\mathcal{A}$
- Question: Can we find by using $\mathcal{A}$, for any query $Q \in \mathcal{Q}$, any dataset D, and any changes ΔD to D, a fraction D_Q of D such that
  - $|D_Q| \leq M$,
  - $Q(D \oplus \Delta D) = Q(D) \oplus \Delta Q(\Delta D, D_Q)$, and
  - D_Q can be identified in time determined by Q and $\mathcal{A}$?

access an additional bounded amount of data



Query Q may not be boundedly evaluable, but may be incrementally boundedly evaluable!

Incremental pattern queries

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

- ✓ Incremental pattern matching (simulation) is in

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

AFF: D_Q

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

in $O(|AFF|)$ time

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

Incremental Bounded Evaluability: Example

Insert e_2

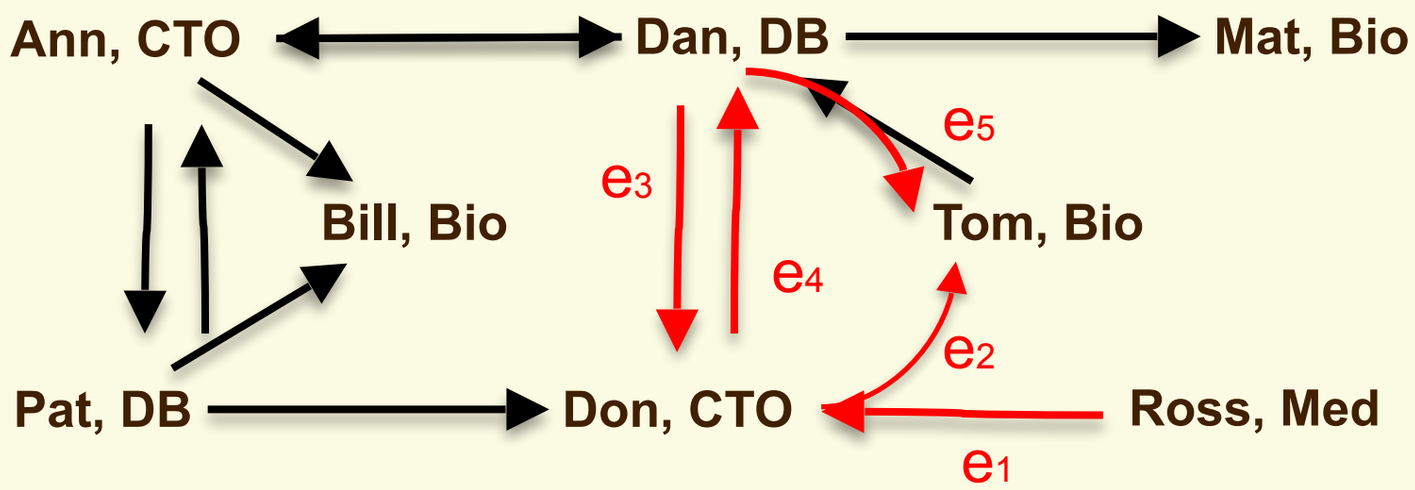
Insert e_1

Insert e_3

Insert e_4

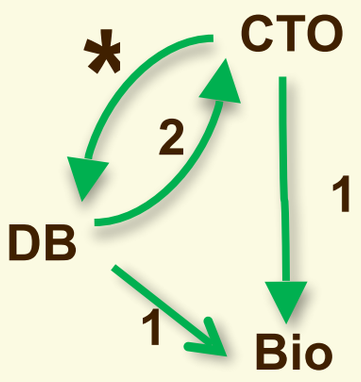
Insert e_5

ΔG

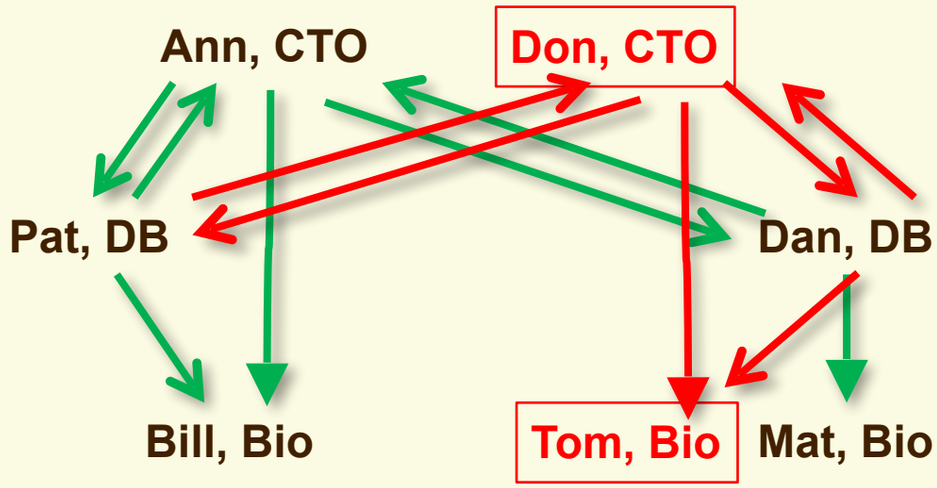


G

P



Gr



affected area

Strategy 3: make Big Data small with more machines

Parallel query processing

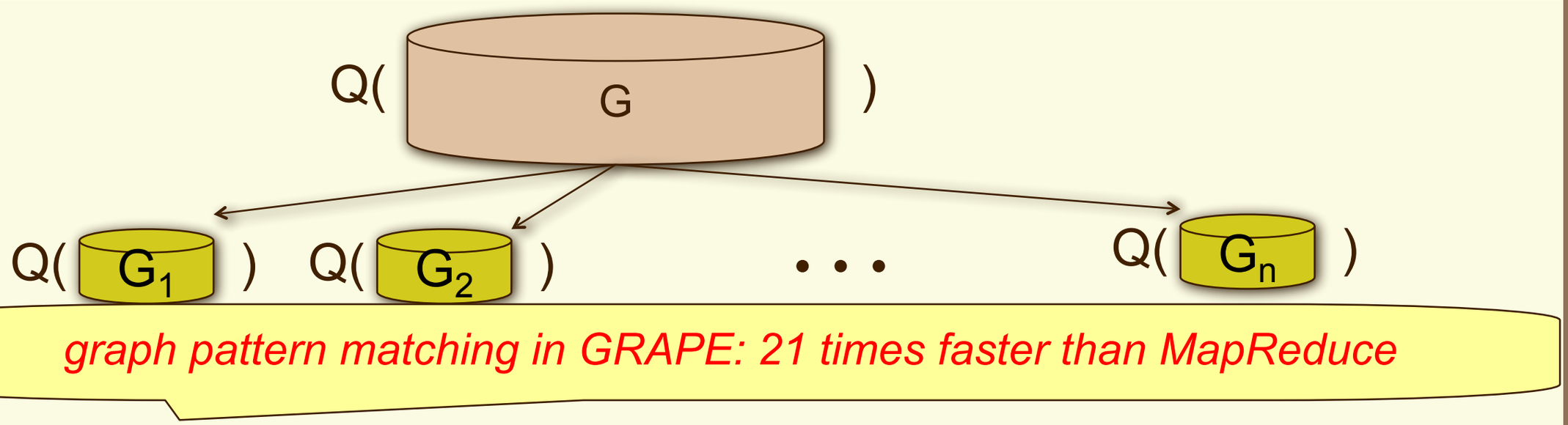
Divide and conquer

manageable sizes

- ✓ partition G into fragments $(G_1, \ldots, G_n)$, distributed to various sites
- ✓ upon receiving a query Q,

evaluate Q on smaller G_i

  - evaluate $Q(G_i)$ *in parallel*
  - collect partial answers at a coordinator site, and assemble them to find the answer $Q(G)$ in the entire G



Parallel processing = Partial evaluation + message passing

Example: Coordinator in GRAPE

Each machine (site) Si is either

- ✓a coordinator
- ✓a worker: conduct local computation and produce partial answers

Coordinator: receive/post queries, control termination, and assemble answers

- ✓ Upon receiving a query Q
  - post Q to all workers
  - Initialize a status flag for each worker, mutable by the worker
- ✓ Terminate the computation when all flags are true
  - Assemble partial answers from workers, and produce the final answer Q(G)

Termination, partial answer assembling

Example: Workers in GRAPE

Worker: conduct local computation and produce partial answers

✓upon receiving a query Q,

- evaluate Q(Gi) *in parallel*
- send messages to request data for “border nodes”

use local data Gi only

With edges to other fragments

✓ Incremental comp… …v messages M

- evaluate Q(Gi + M) *in parallel*

Incremental computation

✓ set its flag true if no more changes to partial results, and send the partial answer to the coordinator

This step repeats until the partial answer at site Si is ready

Local computation, partial evaluation, recursion, partial answers

Strategy 4: Data compression

Query preserving compression

The cost of query processing: **f**(|G|, |Q|)

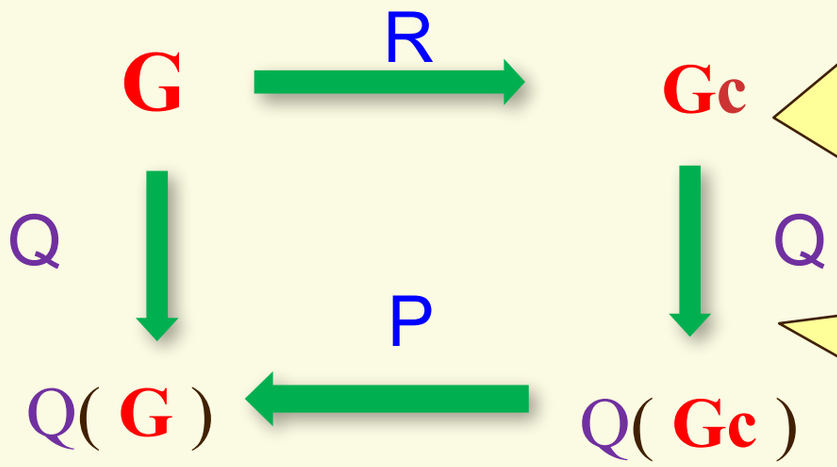
reduce the parameter?

Query preserving compression <R, P> for a class L of queries

- ✓ For any data collection G, $G_C = R(G)$
- ✓ For any Q in L, $Q(G) = P(Q, G_c)$

Compressing

Post-processing



In contrast to lossless compression, retain only relevant information for answering queries in L

No need to restore the original graph G.

Better compression ratio!

18 times faster on average for reachability queries

Reachability queries

- ✓ Reachability
  - Input: A directed graph G, and a pair of nodes s and t in G
  - Question: Does there exist a path from s to t in G?

O(|V| + |E|) time

- ✓ Equivalence relation:
  - reachability relation R_e: a node pair $(u,v) \in R_e$ iff they have the same set of ancestors and descendants in G.
  - for any graph G, there is a unique maximum R_e, i.e., the reachability equivalence relation of G

Compress G by leveraging the equivalence relation

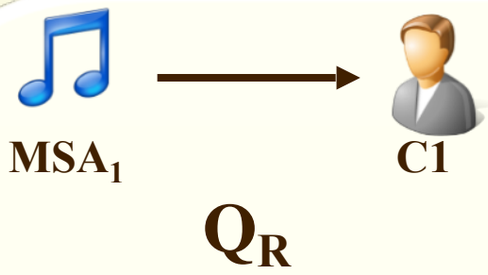
Reachability preserving compression

- A reachability preserving compression R for G
  - R maps each node in G to its reachability equivalence class in G_C, and each edge to an edge between two equivalence classes

Nodes in G_C: equivalence classes

- Correctness:
  - For any query $Q_R(v,w)$ over G, v can reach w iff R(v) can reach R(w) in G_C
  - Compression R is in quadratic time
  - no post-processing function P is required.

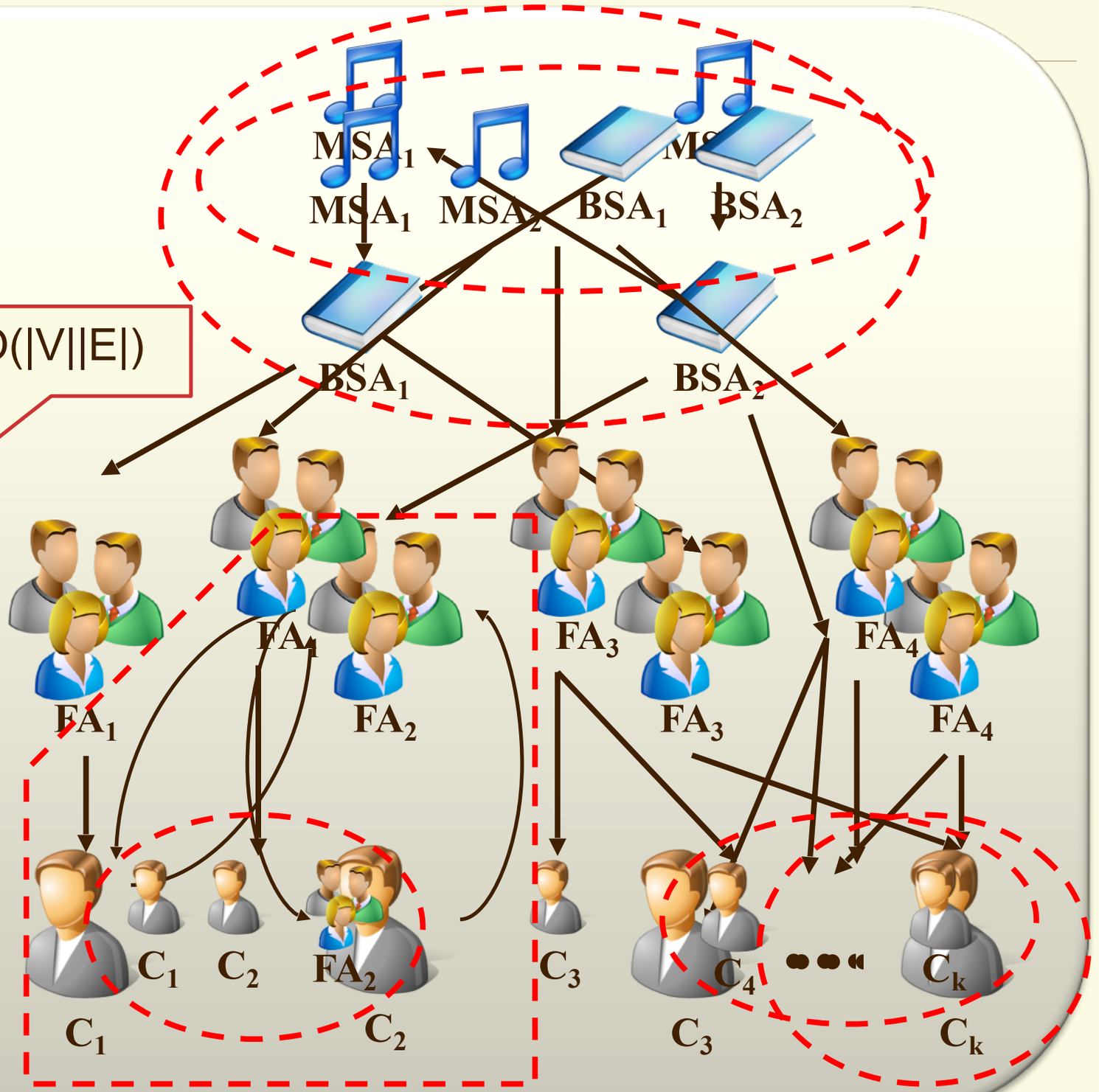
Algorithm and example



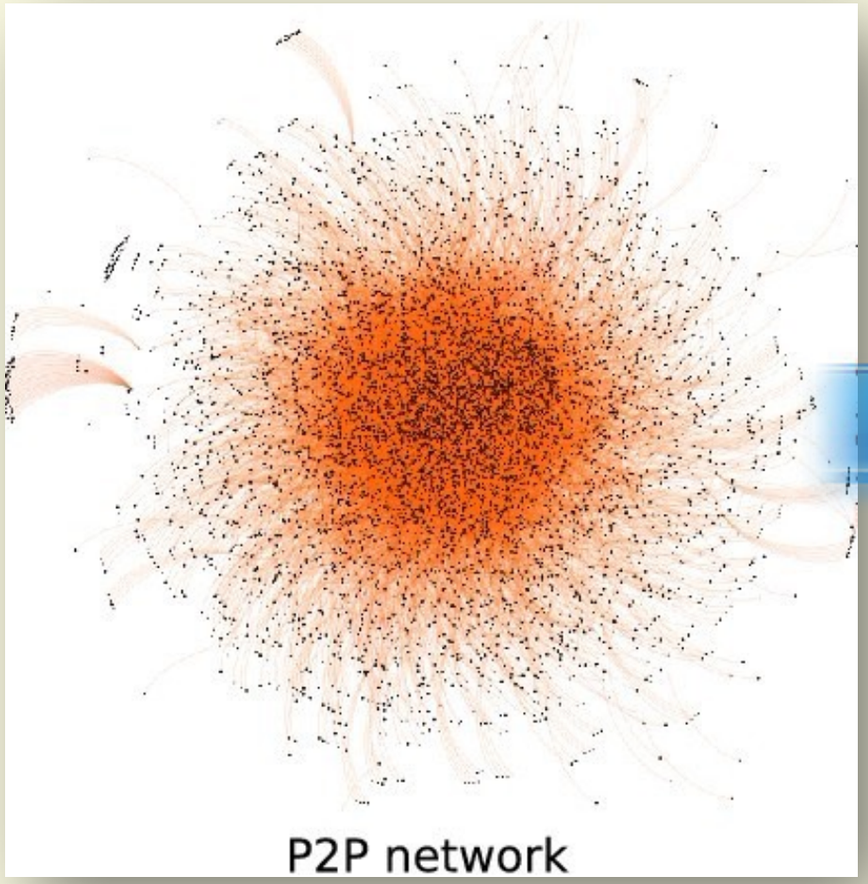
Q_R

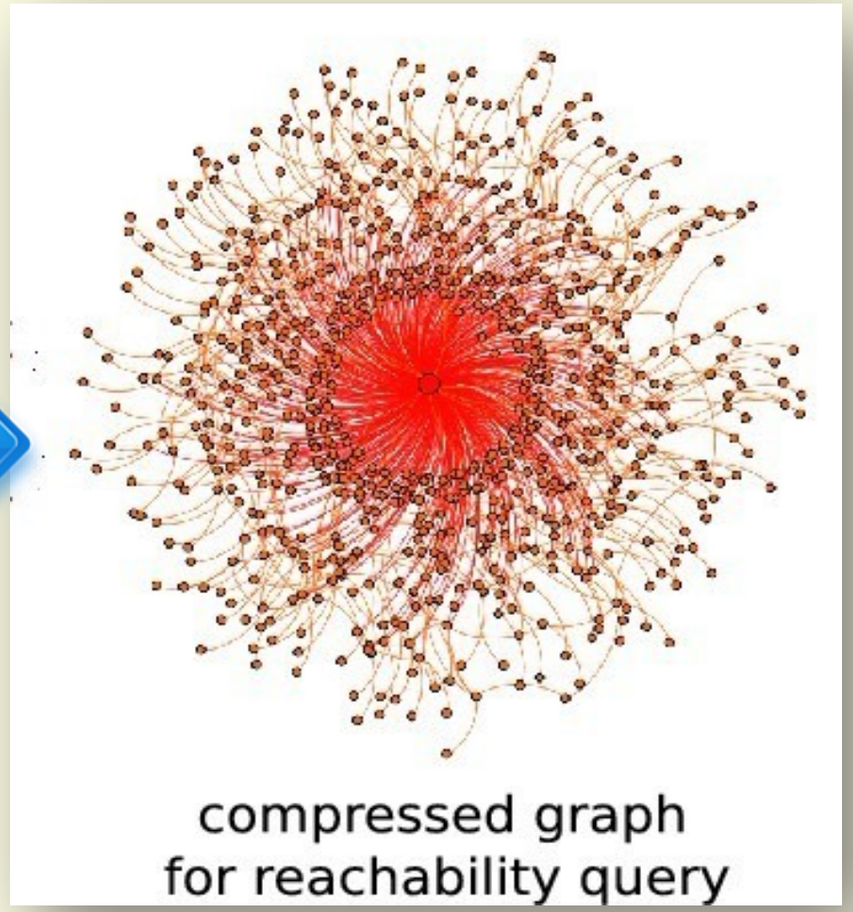
1. Compute Re and its equivalence classes
2. Construct a node for each node set in the equivalence class
3. Construct G_C

O(|V||E|)



What does it look like in real life?





Reduction: 95% in average for reachability queries

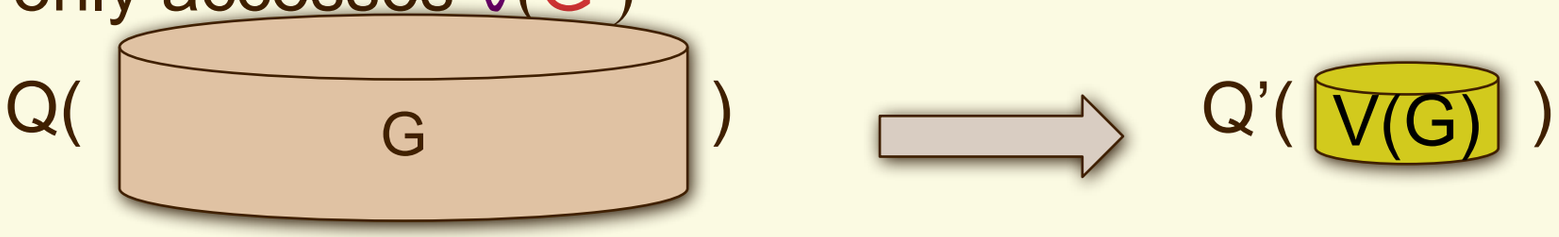
18 times faster on average for reachability queries

Answering queries using views

The cost of query processing: **f**(|G|, |Q|)

Query answering using views: given a query Q in a language L and a set V views, find another query Q' such that

- ✓ Q and Q' are *equivalent* (for any G, Q(G) = Q'(G))
- ✓ Q' only accesses V(G)



V(G) is often much smaller than G (4% -- 12% on real-life data)

Improvement: 31 times faster for graph pattern matching

The complexity is no longer a function of |G|

Incremental query answering

- ✓ Real-life data is dynamic – constant
- ✓ Re-compute Q(G⊕ΔG) starting from scratch?
- ✓ Changes ΔG are typically small

5%/week in Web graphs

Compute Q(G) once, and then incrementally maintain it

Incremental process

- ✓ Input: Q, G, Q(G), ΔG
- ✓ Output: ΔM such that Q(G⊕ΔG) = Q(G) ⊕ ΔM

Old output

Changes to the input

New output

Changes to the output

When changes ΔG to the data G are small, typically so are the

At least twice as fast for pattern matching for changes up to 10%

Minimizing unnecessary recomputation

A principled approach: Making big data small

- ✓ Bounded evaluable queries
- ✓ Parallel query processing (MapReduce, GRAPE, etc)
- ✓ Query preserving compression: convert big data to small data
- ✓ Query answering using views: make big data small
- ✓ Bounded incremental query answering: depending on the size of the changes rather than the size of the original big data
- ✓ . . .

Yes, MapReduce is useful, but it is not the only way!

Combinations of these can do much better than MapReduce!

The journey just starts…

- How does these strategies apply to analytical/mining/learning approaches?
- How to trade off speed with accuracy for mining/learning and other AI operators?
- A declarative language and system to enable tunable computing environment (e.g., speed vs. precision)
  - MIT/UC Berkeley Blink DB; http://blinkdb.org/
- A self-adjust system to automatically find best setting?
  - Big Data ← ? → AI

…

Summary and Review

- ✓ What is BD-tractability? Why do we care about it?
- ✓ What is parallel scalability? Name a few parallel scalable algorithms
- ✓ What is bounded evaluability? Why do we want to study it?
- ✓ How to make big data “small”?
- ✓ Is MapReduce the only way for querying big data? Can we do better than it?
- ✓ What is query preserving data compression? Query answering using views? Bounded incremental query answering?
- ✓ If a class of queries is known not to be BD-tractable, how can we process the queries in the context of big data?