CPT-S 415

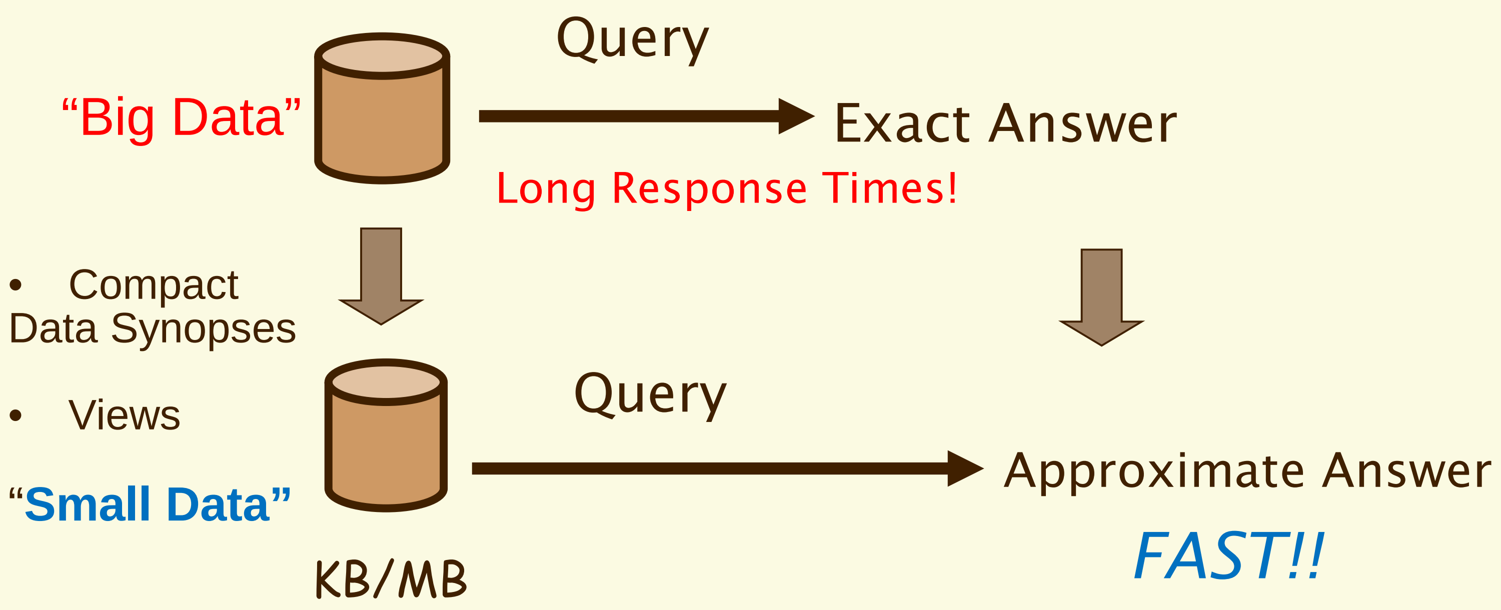
Big Data

Yinghui Wu

EME B45

Answering queries using views

Data-driven Approximate Query Processing



"Big Data"

Query

Exact Answer

Long Response Times!

- Compact Data Synopses
- Views

"**Small Data**"

KB/MB

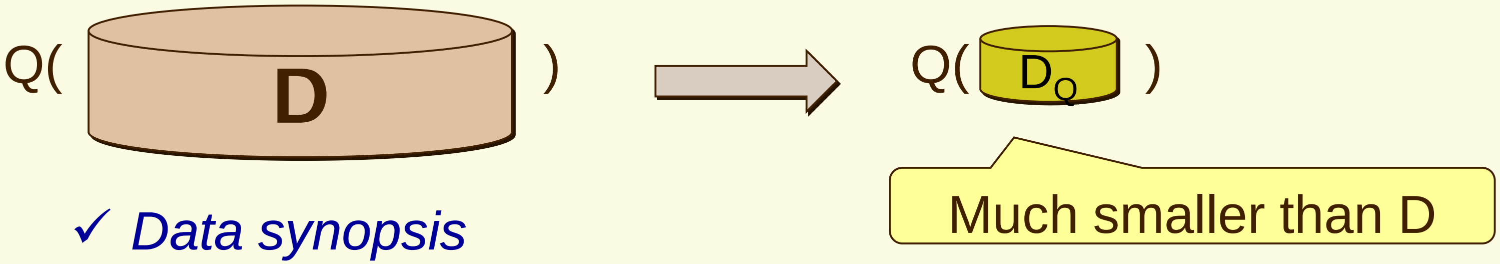
Query

Approximate Answer

FAST!!

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

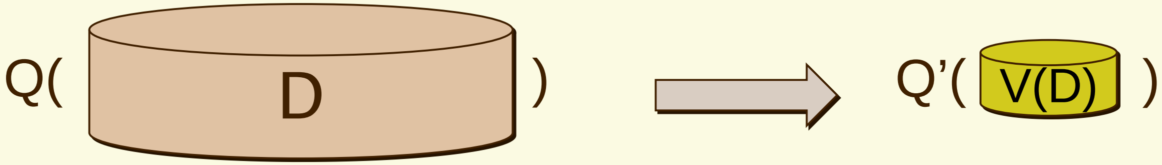
Answering queries using views

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

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

- ✓ Q and Q' are *equivalent*
- ✓ Q' only accesses $\mathcal{V}$(D)

for any G, Q(G) = Q'(G)

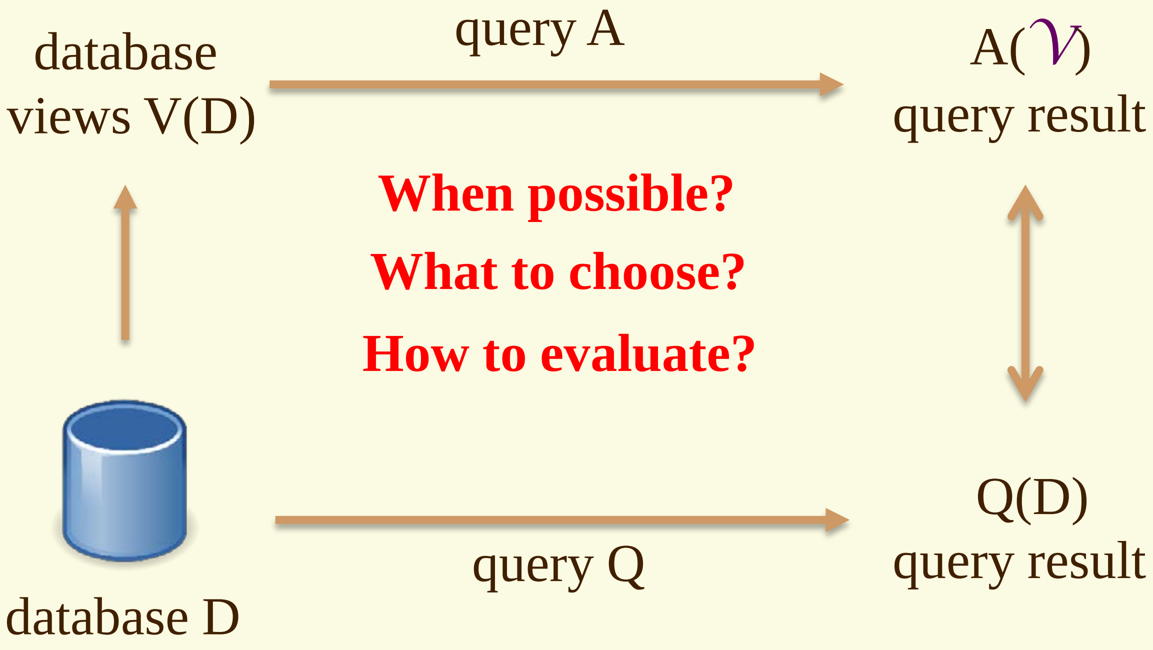


Answering pattern queries on big data:

- ✓ Regardless of how big D is – the cost is "independent" of D
- ✓ $\mathcal{V}$(D) is often much smaller than D (4% -- 12% on real-life data)

The complexity is no longer a function of |D|

Answering query using views



database views V(D)

query A

$A(\mathcal{V})$ query result

When possible?

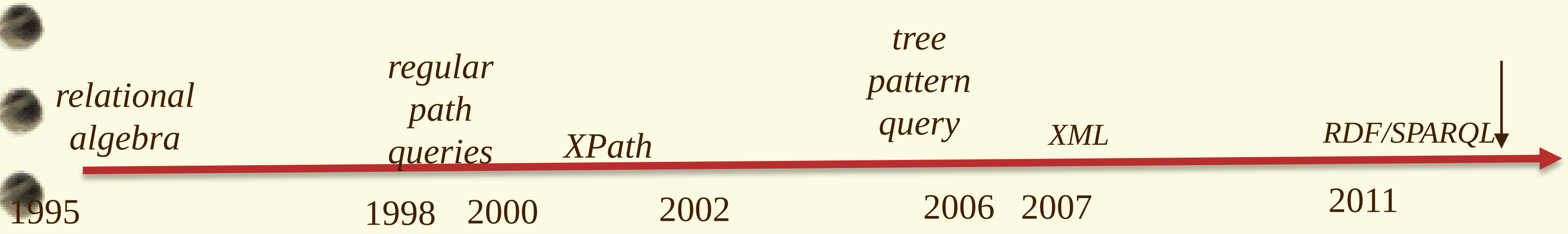
What to choose?

How to evaluate?

database D

query Q

Q(D) query result



relational algebra 1995

regular path queries 1998

2000

XPath 2002

tree pattern query 2006

XML 2007

RDF/SPARQL 2011

A classical techniques

Views revisited for Relational Data

- Views are relations, except that they are not physically stored.
  - a *logical* or *virtual table* based on a query.
  - useful to think of a *view* as a stored query.
  - Views are created through use of a CREATE VIEW
  - command that incorporates use of the SELECT statement.
  - Views are queried just like tables.
- For presenting different information to different users
- Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = “Development”
```

- Payroll has access to Employee, others only to Developers

Types of Views

- ✓ Virtual views:
  - – Used in databases
  - – Computed only on-demand – slow*er* at runtime
  - – Always up to date

- ✓ Materialized views
  - – A view whose tuples are stored in the database is said to be materialized

  - – Provides fast access, like a (very high-level) cache.

  - – Need to maintain the view as the underlying tables change.
  - – Ideally, incremental view maintenance algorithms.
  - – Used in data warehouses
  - – Precomputed offline – faster at runtime

A View Definition

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW  Pullman-view  AS

        SELECT  buyer, seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = “Pullman”    AND
                Person.name = Purchase.buyer
```

We have a new virtual table:
Pullman-view(buyer, seller, product, store)

Application: Querying the WWW

- ✓ Assume a virtual schema, e.g.,
  - Course(number, university, title, prof, quarter)
- ✓ Every data source on the web contains the answer to a view over the virtual schema:

```
WSU database:   SELECT  number, title, prof
                FROM    Course
                WHERE   univ='WSU' AND quarter='09/15'
Stanford database: SELECT  number, title, prof, quarter
                   FROM    Course
                   WHERE   univ='Stanford'
```

User query: find all professors who teach "database systems"

Answering Queries Using Views

- ✓ What if we want to *use* a set of views to answer a query?
  - – Given a query Q and a set of view definitions V1,…,Vn:
  - – Is it possible to answer Q using only the V's?

- ✓ Why?
  - – The obvious reason…
  - – Answering queries vs big variety.
  - – Data integration and knowledge integration

- ✓ How? Query Rewriting and Query Answering
  - – Query rewriting based on schema information
    - Query containment and minimization
  - – Query answering without schema information

Query answering using views: what can go wrong?

- ✓ I still have **only** the result of PullmanView:

```
SELECT  buyer, seller, product, store
FROM    Person, Purchase
WHERE   Person.city = ‘Pullman’    AND
        Person.per-name = Purchase.buyer
```

- ✓ but I want to answer the query

```
SELECT  buyer, seller
FROM    Person, Purchase
WHERE   Person.city = ‘Pullman’    AND
        Person.per-name = Purchase.buyer AND
        Person.Phone  LIKE ‘206 543 %’.
```

Another example

- ✓ Query: q(X,Z) :- r(X,Y), s(Y,Z), t(X,Z), Y > 5.
- ✓ What can go wrong?

- V1(A,B) :- r(A,C), s(C1,B) (join predicate not applied)
- V2(A,B) :- r(A,C), s(C,B), C > 1 (predicate too weak).
- V3(A,B) :- r(A,B), r1(A,B) (irrelevant condition).
- V4(A) :- r(A,B), s(B,C), t(A,C), B > 5:
- needed argument is projected out. Can be recovered
- if we have a functional dependency t: A --> C.

Dimensions of Query Rewriting Problem

- ✓ View definition language
- ✓ Query language
- ✓ **Equivalent or maximally contained rewriting (When?)**
- ✓ **Query evaluation (How?)**
- ✓ **Selection of views (What to select?)**
- ✓ Completeness/soundness of the views
- ✓ Output: query execution plan or logical plan.

Query Containment and Equivalence: Definitions

- ✓ Query Q_1 *contained in* query Q_2 if for **every** database *D*

  $$Q_1(D) \supseteq Q_2(D)$$

- ✓ Query Q_1 is equivalent to query Q_2 if $Q_1(D) \supseteq Q_2(D)$ and $Q_2(D) \supseteq Q_1(D)$

Original query:

```
SELECT  buyer, seller
FROM    Person, Purchase
WHERE   Person.city = 'Pullman'
AND Person.per-name = Purchase.buyer
AND Purchase.product='gizmo'
```

Rewritten query:

```
SELECT  buyer, seller
FROM      PullmanView
WHERE   product= 'gizmo'
```

Note: properties of the queries, not of the database!

Query Rewriting: issues

- ✓ Given a query Q and a set of view definitions V1,…,Vn
- ✓ Q’ is a rewriting of the query using V’s if it refers only to the views or to interpreted predicates.
- ✓ Q’ is an equivalent rewriting of Q using the V’s if Q’ is equivalent to Q.
- ✓ Q’ is a maximally-contained rewriting of Q w.r.t. L using the V’s if there is no other Q’’ such that: Q’’ strictly contains Q’, and Q’’ is contained in Q.

The rewriting problem is NP-hard.

Certain Answers

✓ **Given:** A query Q, View definitions $V_1,\ldots V_n$,
Extensions of the views: $v_1,\ldots v_n$ i.e. materialized views

✓ Consider the set of databases ***D*** that are consistent with $V_1,\ldots V_n$ and $v_1,\ldots v_n$.

✓ The tuple *t* is a certain answer to Q if it would be an answer in every database in ***D.***

✓ Note: an equivalent rewriting provides all certain answers.

Schema: friends(X,Y)

T1: select X from friends (X,Y): extension: {HarryPotter} → {HarryPotter, RonWeasley}

T2: select Y from friends (X,Y): extension: {RonWeasley}

Query: select (x,y) from friends (X,Y)

Finding All Answers from Views

- ✓ If a rewriting is equivalent: you definitely get all answers

- ✓ So what is the complexity of finding all the answers?

  - [Abiteboul & Duschka, PODS-98],
  - [Grahne and Mendelzon, ICDT-99]: *surprisingly hard!*

- ✓ **Certain answers:**
- ✓ Given specific extensions $v_1,\ldots v_n$ to the view, is the tuple *t* is an answer in *every* database *D* that is consistent with the extensions $v_1,\ldots,v_n$?

Why & When is it Hard?

Sources can be:

- sound (open world assumption)
- complete
- sound and complete (closed-world assumption)

- If sources are either all sound or all complete, then maximally-contained rewriting exists.

- If sources are sound and complete, the problem is NP-complete.

Schema: friends (X,Y)

T1: select X from friends (X,Y): extension: {a}

T2: select Y from friends (X,Y): extension: {b}

→ {a, b} ?

Query: select (x,y) from friends (X,Y)

Query Rewriting Using Views

Original query:

```
SELECT  buyer, seller
FROM    Person, Purchase
WHERE   Person.city = ‘Pullman’    AND
         Person.per-name = Purchase.buyer  AND
         Purchase.product=‘gizmo’.
```

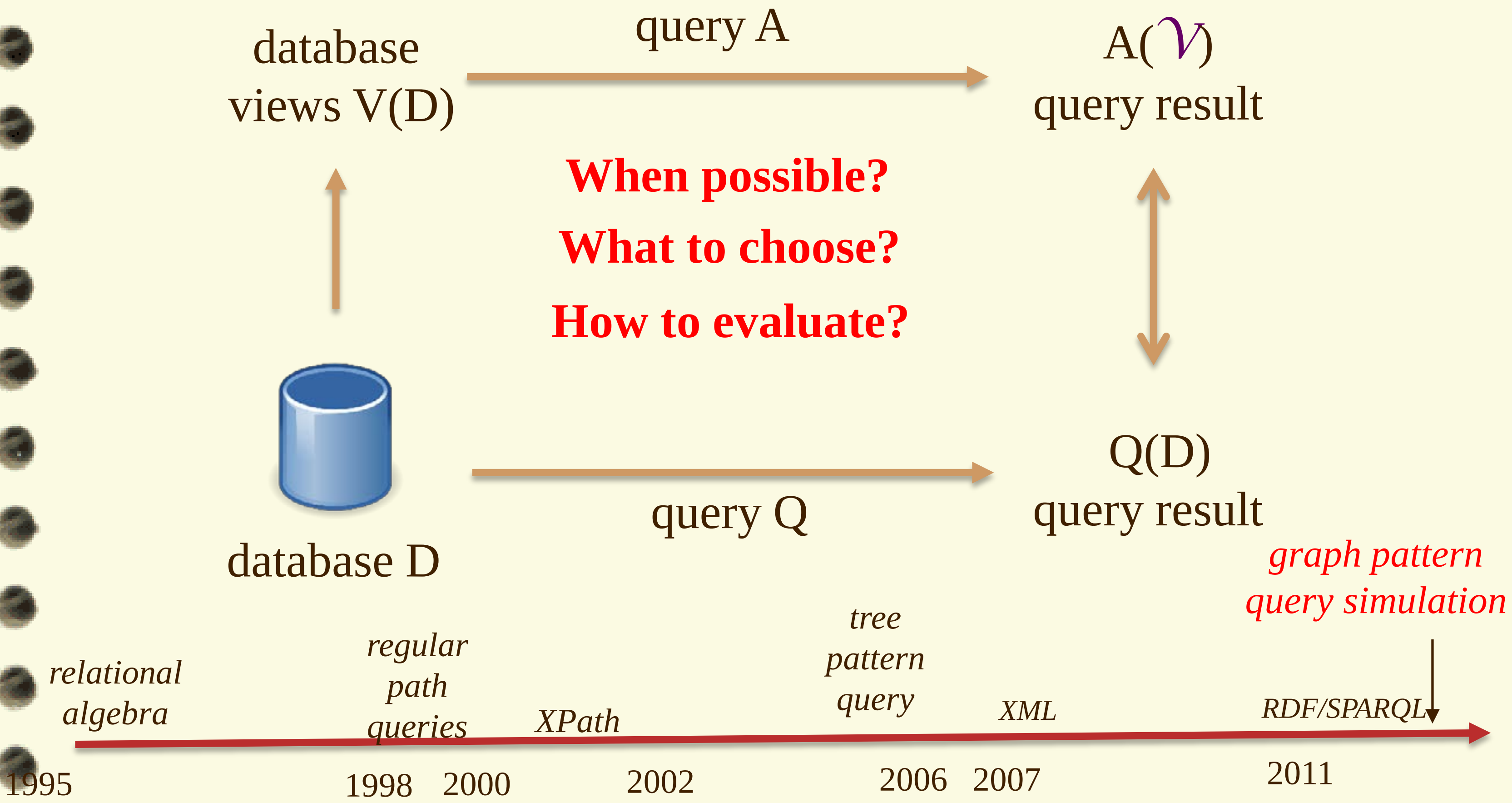
Rewritten query:

```
SELECT  buyer, seller
FROM     PullmanView
WHERE   product= ‘gizmo’
```

Pullman-view(buyer, seller, product, store)

Graph pattern matching using views

Answering query using views



database views V(D)

query A

A($\mathcal{V}$) query result

When possible?

What to choose?

How to evaluate?

database D

query Q

Q(D) query result

graph pattern query simulation

relational algebra 1995

regular path queries 1998

2000

XPath 2002

tree pattern query 2006

XML 2007

RDF/SPARQL 2011

A classical techniques, but still in their infancy for graphs

Graph pattern matching by graph simulation

- ✓ Input: A directed graph G, and a graph pattern Q
- ✓ Output: the maximum simulation relation R

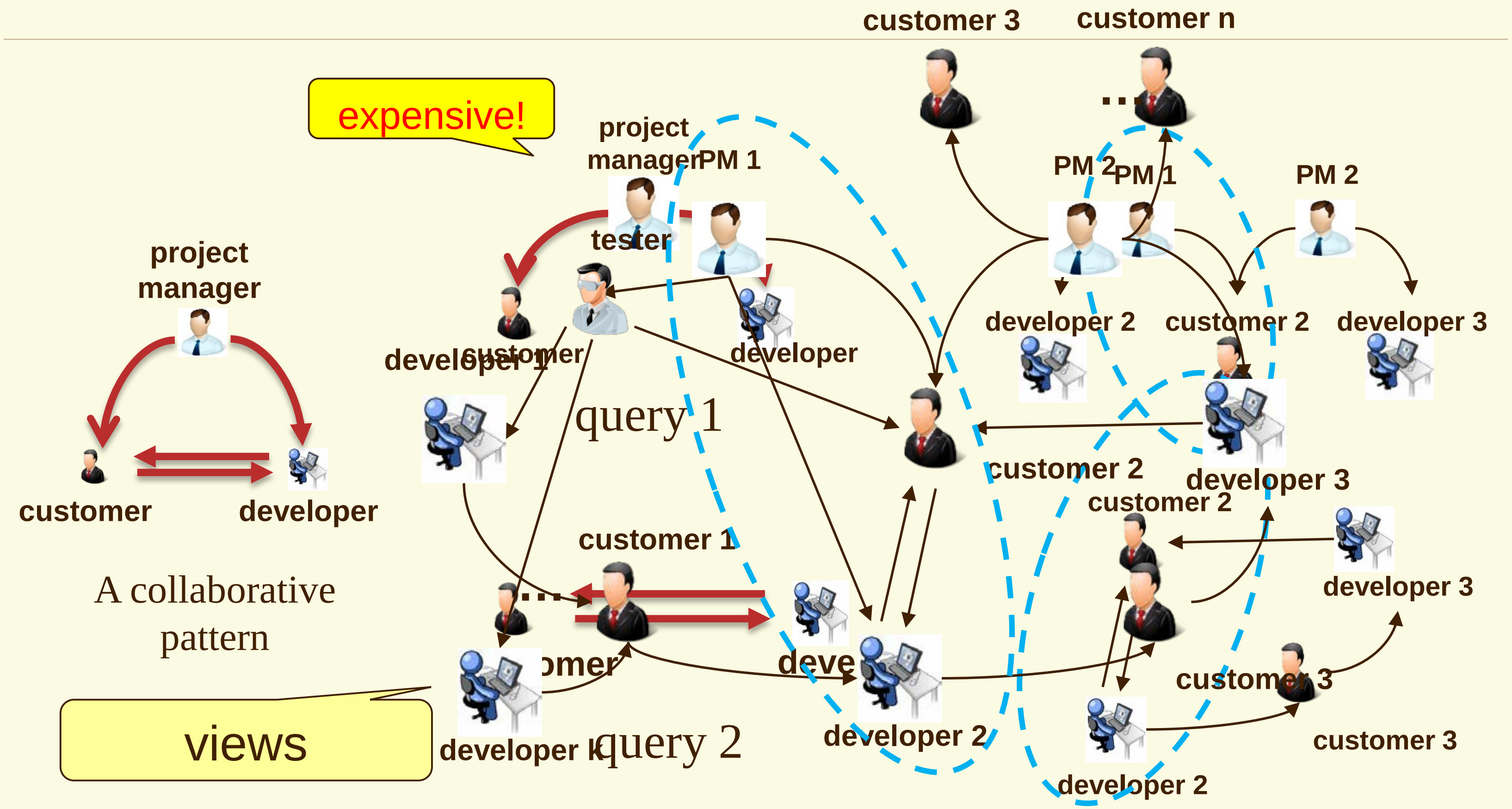
- ✓ Maximum simulation relation: always exists and is unique
  - If a match relation exists, then there exists a maximum one
  - Otherwise, it is the empty set – still maximum

- ✓ Complexity: $O((|V| + |V_Q|)(|E| + |E_Q|))$

- ✓ The output is a unique relation, possibly of size |Q||V|

Using views? Incremental?

Querying collaborative network



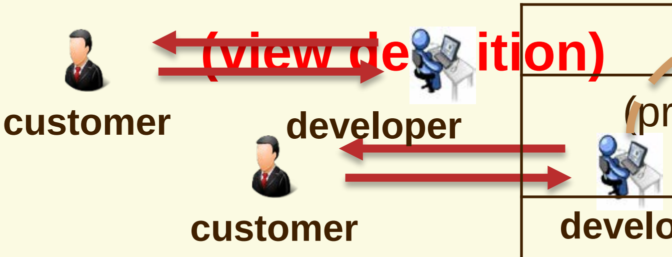
A collaborative pattern

A collaborative (chat) network

Amrit Chintan et al, Information System management, 2010

Detecting Coordination Problems in Collaborative Software Development Environments,

patterns and views



(view definition)

view 1

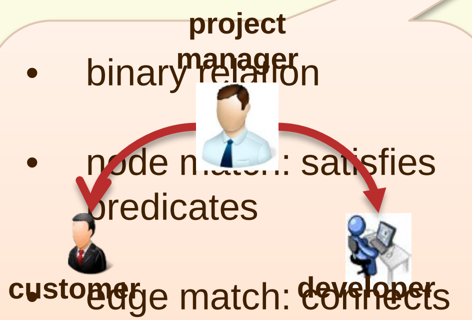
edges	matches
(project manager, developer)	{(PM 1, developer 2), (PM 2, developer 3)}
(project manager, customer)	{(PM 1, customer 2), (PM 2, customer 2),

pattern query

view definition 1

view extension 1

query result



- binary relation
- node match: satisfies predicates
- edge match: connects two node matches

view definition 2

view 2

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

(view extension)

view extension 2

When a pattern can be matched using views

Given Q and a set $\mathcal{V} = \{V_1, \ldots V_n\}$, Q is contained in $\mathcal{V}$, denoted by $Q \sqsubseteq \mathcal{V}$ if there exists a mapping λ from query edge to view edges, such that for all graphs G

- ✓ the edge matches of Q are contained in the edge matches of those specified by λ.

Q can be answered using views $\mathcal{V}$
if and only if $Q \sqsubseteq \mathcal{V}$

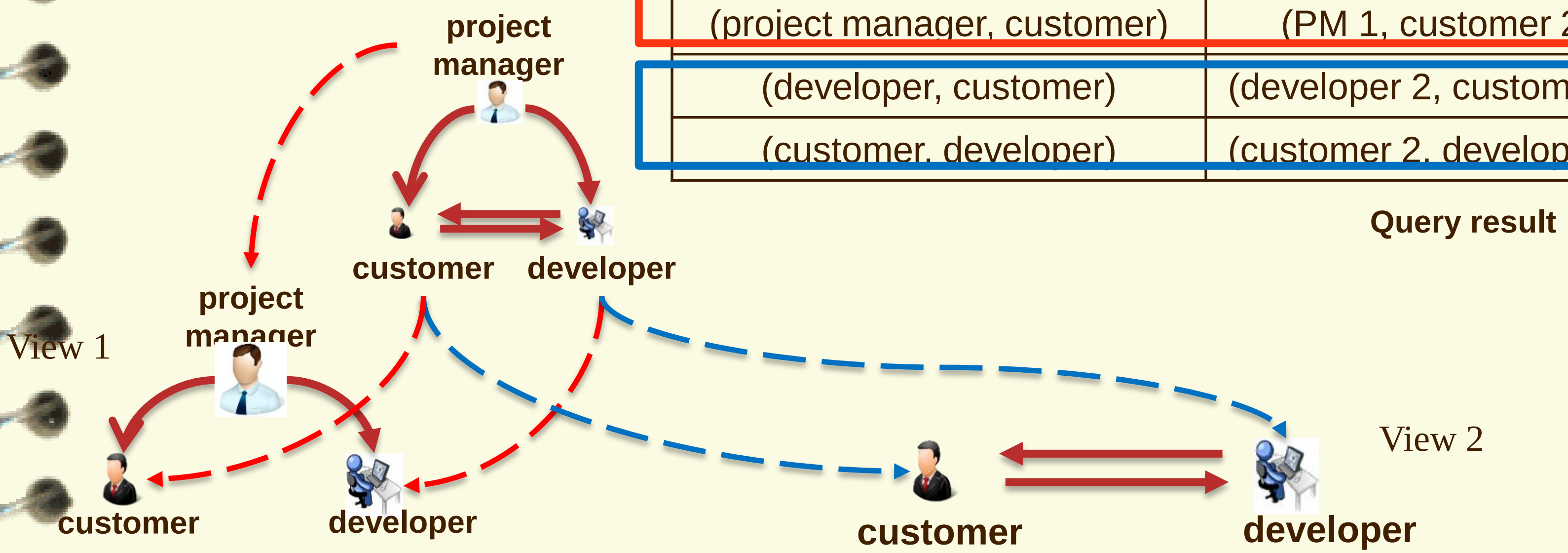
A necessary and sufficient condition

Pattern containment: a characterization

Pattern containment

(project manager, developer)	(PM 1, developer 2)
(project manager, customer)	(PM 1, customer 2)
(developer, customer)	(developer 2, customer 2)
(customer, developer)	(customer 2, developer 2)

Query result



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

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

How to determine the existence of λ?

Determining Pattern containment

NP-complete for relational conjunctive queries, undecidable for relational algebra

Given Q and a set $\mathcal{V} = \{V_1, \ldots V_n\}$,

Determine whether $Q \sqsubseteq \mathcal{V}$ is in PTIME

Idea: matching views to queries (as "a canonical data graph")

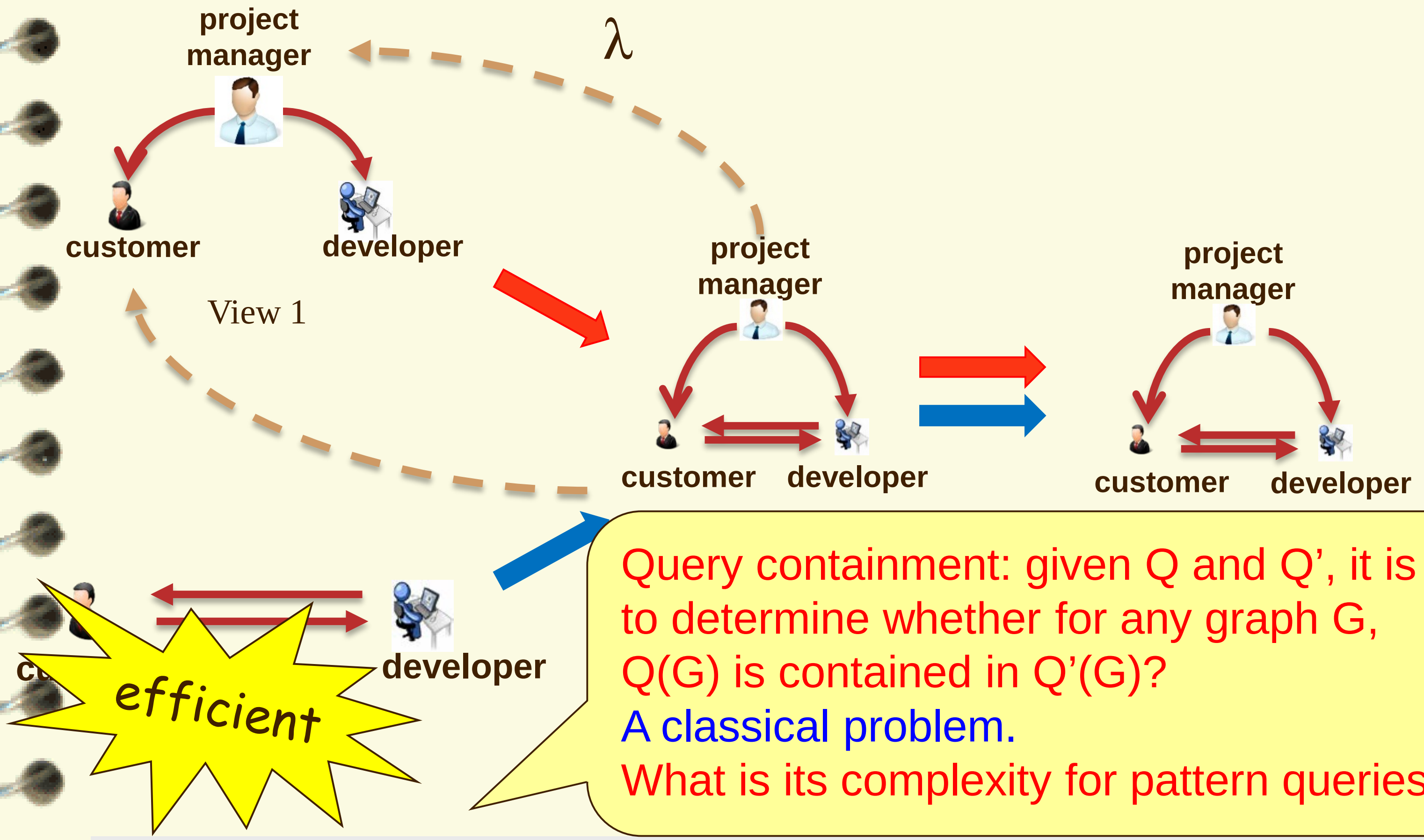
- view match M_V^Q: edge matches of a view V in a query Q
- $Q \sqsubseteq \mathcal{V}$ if and only if the union of all view matches M_V^Q is E_p, the query edge set

Algorithm

- Compute the edge matches of each view (treat Q as data graph)
- Check if the union of all edge matches is E_p
- Induce λ if $Q \sqsubseteq \mathcal{V}$

A practical characterization: patterns are small in practice

Pattern 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

Query evaluation using views

✓ Input: pattern query Q, graph G, a set of views $\mathcal{V}$ and extensions in G, and a mapping λ

✓ Output: Find the query result Q(G)

Algorithm

- Collect edge matches for each query edge e and λ(e)
- Iteratively remove non-matches until no change happens
- Return Q(G)

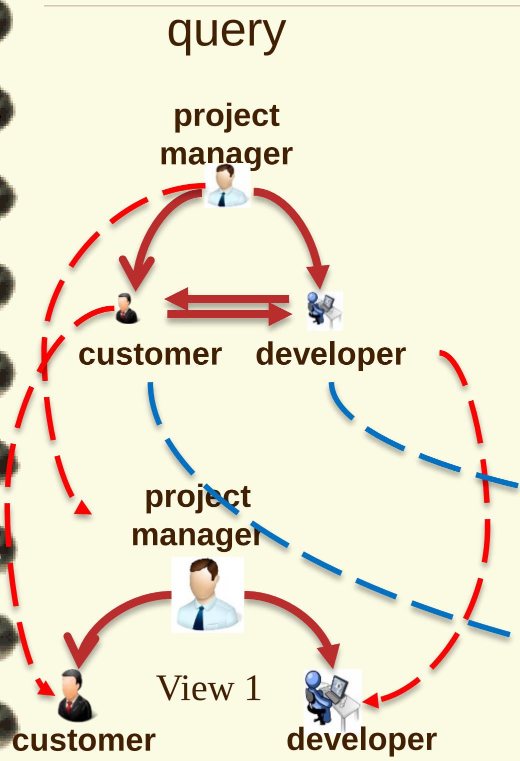
Recall simulation algorithm

If Q ⊑ $\mathcal{V}$

More efficient.

Q(G) can be evaluated in $O(|Q||\mathcal{V}(G)| + |\mathcal{V}(G)|^2)$ time

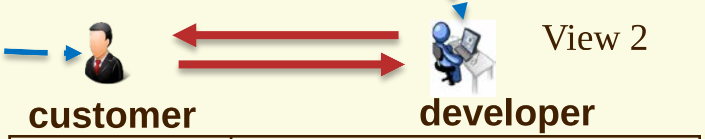
Query evaluation using views



(project manager, developer)	2), ~~(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	



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

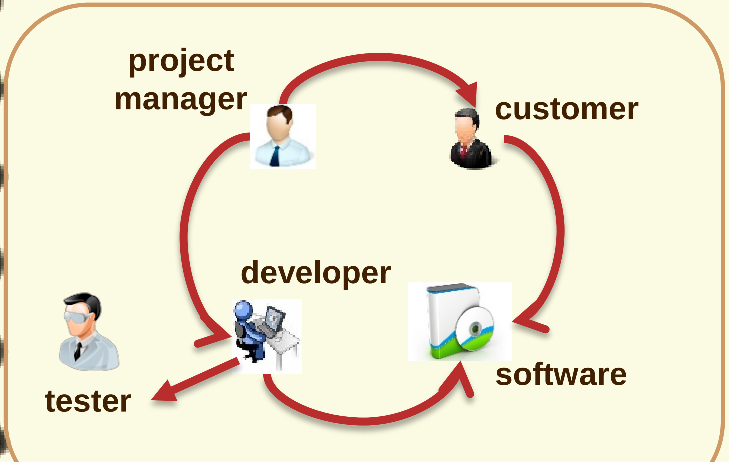
“bottom-up” strategy

4% -- 12% of G

Are we done yet?

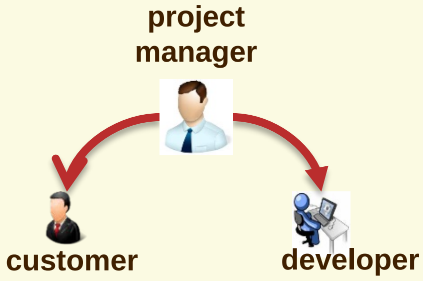
Without accessing the underlying big graph G

What views to choose?

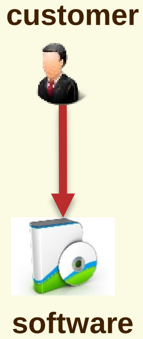


query

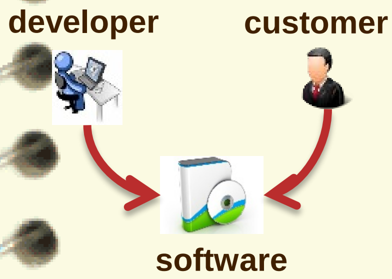
choose all?



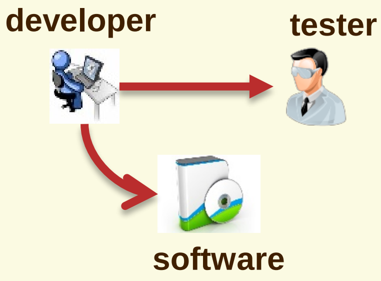
view 1



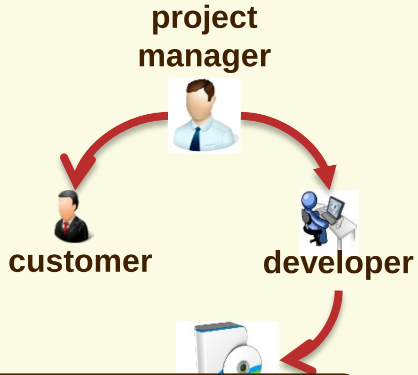
view 2

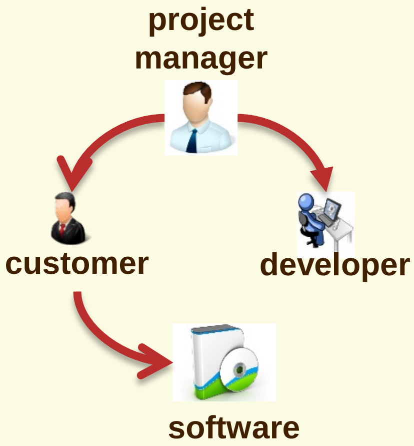


view 3



view 4





view 6

efficiency

Why do we care?

Minimum containment

Given Q and a set of views $\mathcal{V}$, find a subset of $\mathcal{V}'$ such that

- $Q \sqsubseteq \mathcal{V}$
- no view set with less views can contain Q

Minimum containment is NP-complete

- APX-hard as optimization
- - no PTIME algorithm that gives approx. ratio within arbitrarily given constant factor

two options

What can we do?

An log|Ep|-approximation

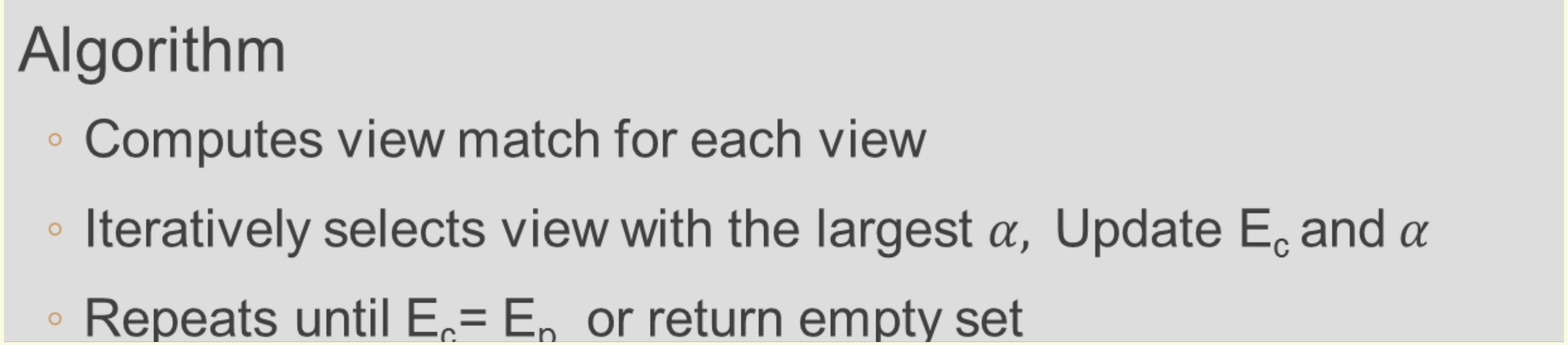
Idea: greedily select views $\mathcal{V}$ that “cover” more query edges

$$\alpha(V) = \frac{|MV^Q \setminus \mathrm{Ec}|}{|Ep|}$$

E_c: already covered

MV^Q : new addition

To decide whether to include a particular view V



Algorithm

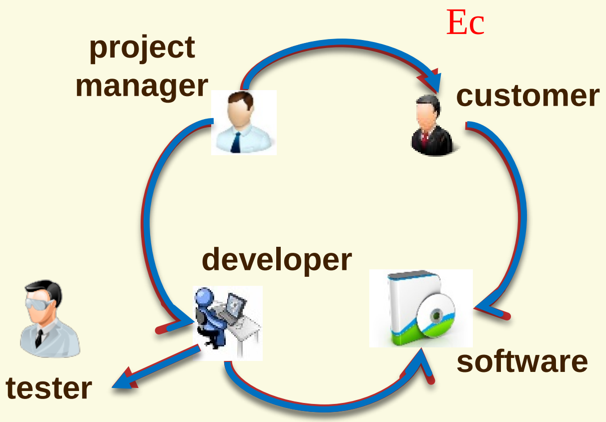
- Computes view match for each view
- Iteratively selects view with the largest α, Update E_c and α
- Repeats until $E_c = E_p$ or return empty set

Approximation: performance guarantees

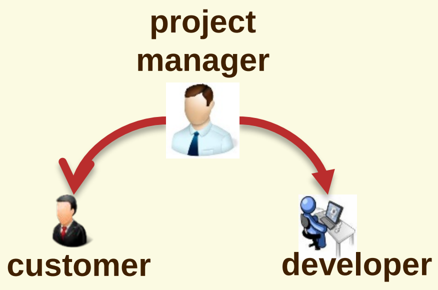
Minimum containment: example

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

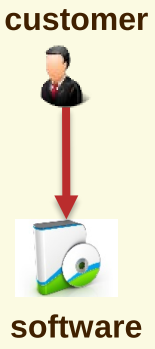
$\alpha(\text{view } 1) = 0$



query



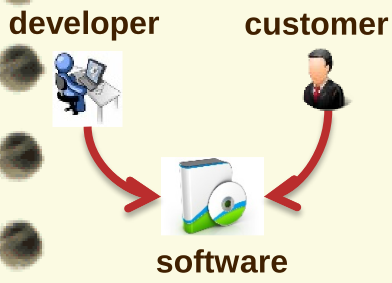
view 1



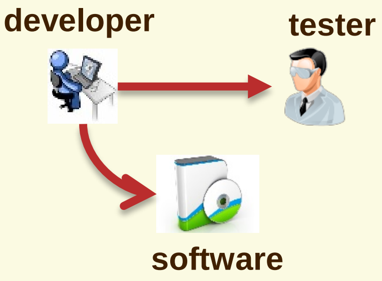
view 2

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

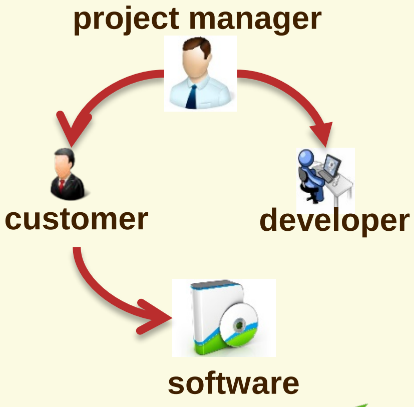
$\alpha(\text{view } 4) = 0.4$



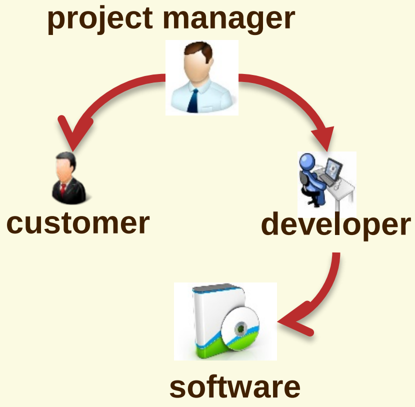
view 3



view 4 ✓



view 5 ✓



view 6

Greedy: based on the metric

Minimal containment

Given Q and a set of views $\mathcal{V}$, find a subset of V' such that

- $Q \sqsubseteq V$
- no views as subset of V' can contain Q

$O(|Q|^2 \, card(\mathcal{V}) + |\mathcal{V}|^2 + |Q| \, |\mathcal{V}|)$ time

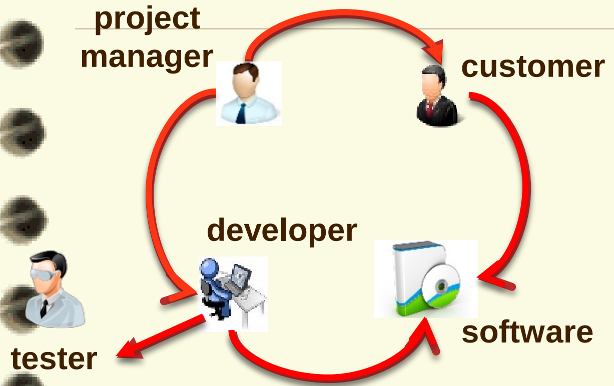
Algorithm

- Computes view match for each view
- Iteratively selects a view that extends E_c
- Repeats until Ec= Ep or return empty set

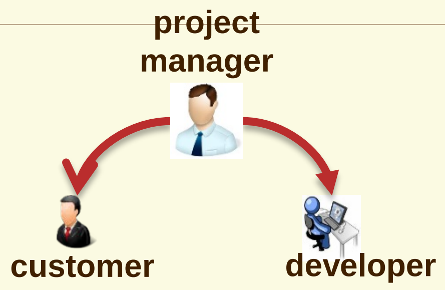
new addition

Minimal containment is in PTIME

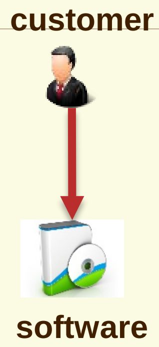
Minimal containment: example



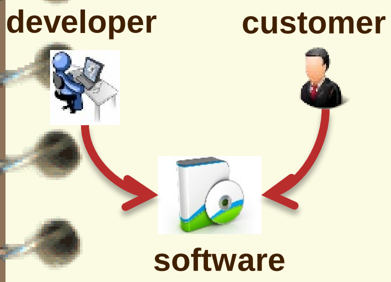
query



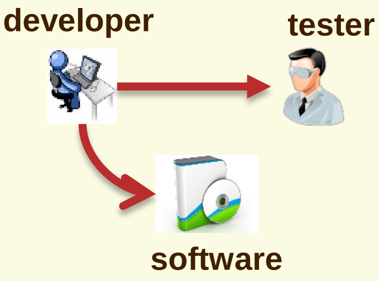
view 1 ✓



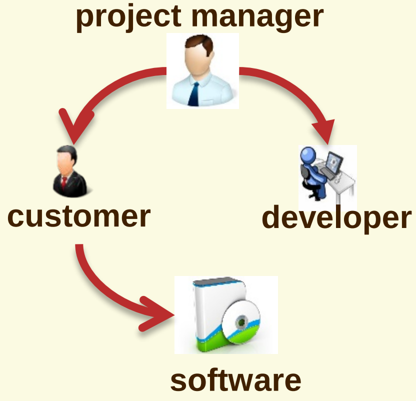
view 2 ✗



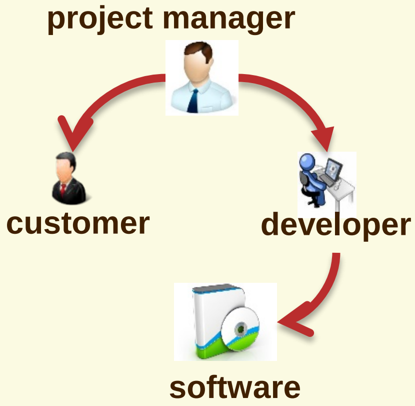
view 3 ✓



view 4 ✓



view 5



Eliminate redundant views

Putting together

Problem	Complexity	Algorithm
containment	PTIME	$O(card(V)\|Q\|^2+\|V\|^2+\|Q\|\|V\|)$
minimum containment	NP-c/APX-hard	$\log\|E_p\|$-approximable $O(card(V)\|Q\|^2+\|V\|^2+\|Q\|\|V\|+\|Q\|card(V)^{3/2})$
minimal containment	PTIME	$O(card(V)\|Q\|^2+\|V\|^2+\|Q\|\|V\|)$
evaluation	PTIME	$O(\|Q\|\|V(G)\| + \|V(G)\|^2)$

✓ characterization: sufficient and necessary condition for deciding whether a query can be answered using a set of views

✓ evaluation: how to evaluate queries using views

✓ vie e f

Subgraph isomorphism?

View maintenance?

The study is still in its infancy for graph queries

Summing up

Making big data small

Yes, it is doable!

- ✓Approximate query models *(query-driven approximation)*
- ✓Data synopsis: property preserving *(data-driven approximation)*
- ✓Bounded evaluable queries: dynamic reduction *(principled search scheme)*
- ✓Query answering using views: *(make big data small)*
  - ✓ *query evaluation by only accessing views*
- ✓Incremental query answering: *(coping with dynamic data; next lecture)*
  - ✓ depending on the size of the changes rather than the size of the original big data
- ✓. . .

Combinations of these are more effective

Summary and review

- ✓ What is query answering using views?
- ✓ What is query containment? What is the complexity of deciding query containment for relations? For XML? Graph pattern queries via graph simulation?
- ✓ What questions do we have to answer for answering graph queries using views?
- ✓ What is incremental query evaluation? What are the benefits?
- ✓ What is a unit update? Batch updates?
- ✓ When can we say that an incremental problem is bounded? Semi-bounded?
- ✓ How to show that an incremental problem is bounded? How to disprove it?

Papers for you to review

- W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In WWW, 2011.

  http://www.cs.fsu.edu/~lifeifei/papers/rdfview.pdf
- D. Saha. An incremental bisimulation algorithm. In FSTTCS, 2007 http://cs.famaf.unc.edu.ar/~rfervari/sites/all/files/readings/incremental-bis-07.pdf
- S. K. Shukla, E. K. Shukla, D. J. Rosenkrantz, H. B. H. Iii, and R. E. Stearns. The polynomial time decidability of simulation relations for finite state processes: A HORNSAT based approach. In DIMACS Ser. Discrete, 1997. (search Google Scholar)
- W. Fan, X. Wang, and Y. Wu. Answering Graph Pattern Queries using Views, ICDE 2014. (query answering using views)
- W. Fan, X. Wang, and Y. Wu. Incremental Graph Pattern Matching, TODS 38(3), 2013. (bounded incremental query answering)

Papers to read

- Shoubridge P., Kraetzl M., Wallis W. D., Bunke H. Detection of Abnormal Change in a Time Series of Graphs. Journal of Interconnection Networks (JOIN) 3(1-2):85-101, 2002.
- Shoubridge, P. Kraetzl, M. Ray, D. Detection of abnormal change in dynamic networks. Information, Decision and Control, 1999.
- Kelly Marie Kapsabelis, Peter John Dickinson, Kutluyil Dogancay. Investigation of graph edit distance cost functions for detection of network anomalies. ANZIAM J. 48 (CTAC2006) pp.436–449, 2007.
- Panagiotis Papadimitriou, Ali Dasdan, Hector Garcia-Molina. Web graph similarity for anomaly detection. J. Internet Services and Applications (JISA) 1(1):19-30 (2010)
- B. Pincombe. Anomaly Detection in Time Series of Graphs using ARMA Processes. ASOR BULLETIN, 24(4):2, 2005.
- Gao, Xinbo and Xiao, Bing and Tao, Dacheng and Li, Xuelong. A survey of graph edit distance. Pattern Anal. and App.s 13 (1), pp. 113-129. 2010.
- Horst Bunke, Peter J. Dickinson, Andreas Humm, Christophe Irniger, Miro Kraetzl: Computer Network Monitoring and Abnormal Event Detection Using Graph Matching and Multidimensional Scaling. Industrial Conference on Data Mining 2006:576-590

Papers to read

- C.E. Priebe, J.M. Conroy, D.J. Marchette, and Y. Park. Scan Statistics on Enron Graphs. Computational & Mathematical Organization Theory, 11(3):229–247, 2005.
- Ide, T. and Kashima, H., Eigenspace-Based Anomaly Detection in Computer Systems. KDD, 2004.
- L. Akoglu, M. McGlohon, C. Faloutsos. Event Detection in Time Series of Mobile Communication Graphs. Army Science Conference, 2010.
- Sun, Jimeng and Xie, Yinglian and Zhang, Hui and Faloutsos, Christos. Less is more: Compact matrix representation of large sparse graphs. ICDM 2007.
- Sun, Jimeng and Tao, Dacheng and Faloutsos, Christos. Beyond streams and graphs: dynamic tensor analysis. KDD 2006: 374-383
- Sun J., Faloutsos C., Papadimitriou S., Yu P. S. GraphScope: parameter-free mining of large time-evolving graphs. KDD, 2007.
- R. Rossi, B. Gallagher, J. Neville, and K. Henderson. Role-Dynamics: Fast Mining of Large Dynamic Networks. 1st Workshop on Large Scale Network Analysis, WWW, 2012.
- Cemal Cagatay Bilgin , Bülent Yener . Dynamic Network Evolution: Models, Clustering, Anomaly Detection. Survey, 2008.