# Breaking the Top-K Barrier of Hidden Web Databases

**Saravanan Thirumuruganathan**
**Dr.Gautam Das**
**UT Arlington and QCRI**

**Dr.Nan Zhang**

**George Washington University**

ICDE 2013

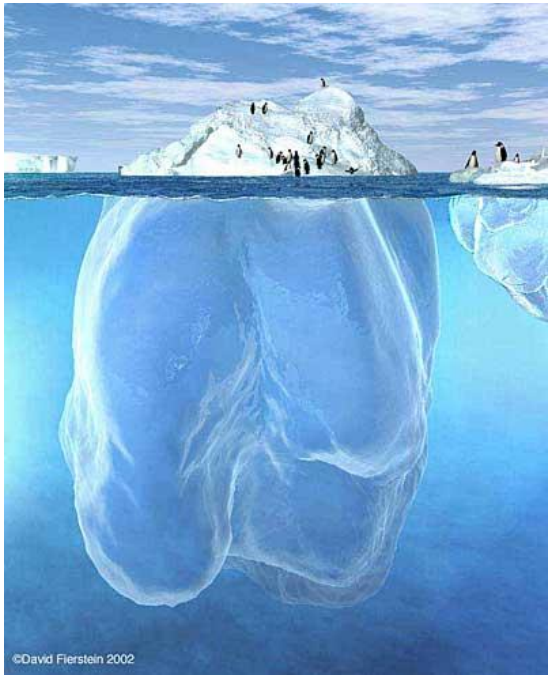
Web Data

# Outline

- Background and Motivation
- GetNext operator
- Candidate generation
- Candidate testing
- Extensions
- Experimental Results
- Conclusions

# The Deep Web

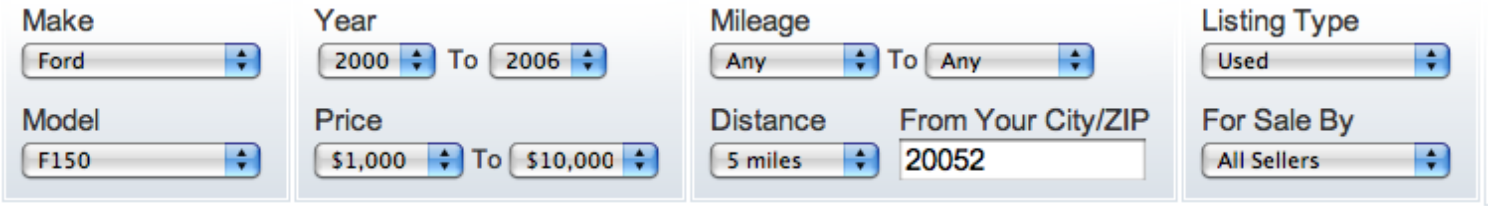
## Deep Web vs Surface Web




Amazon
Ebay
NSF Award Search
PubMed
and many many more

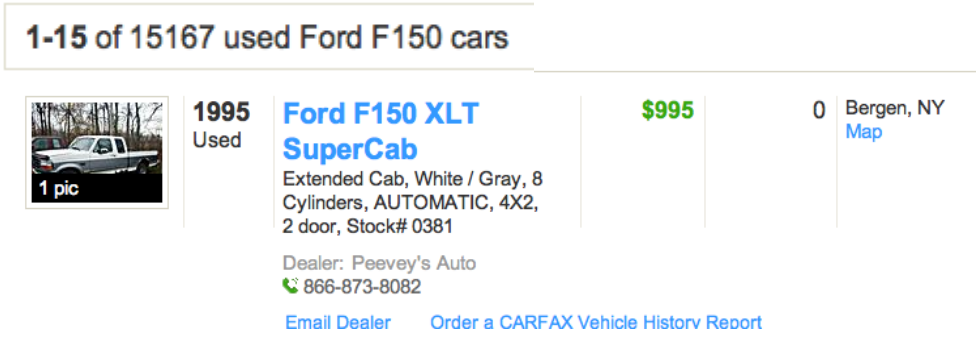
# Hidden Web Databases

Form-like interface




Return top-*k* tuples

Query limits



# Motivation

- *k* is arbitrarily set
- Ranking function : proprietary or pre-chosen
- Limits innovative third-party services
  - Web mashups
  - Web aggregators

# Motivation

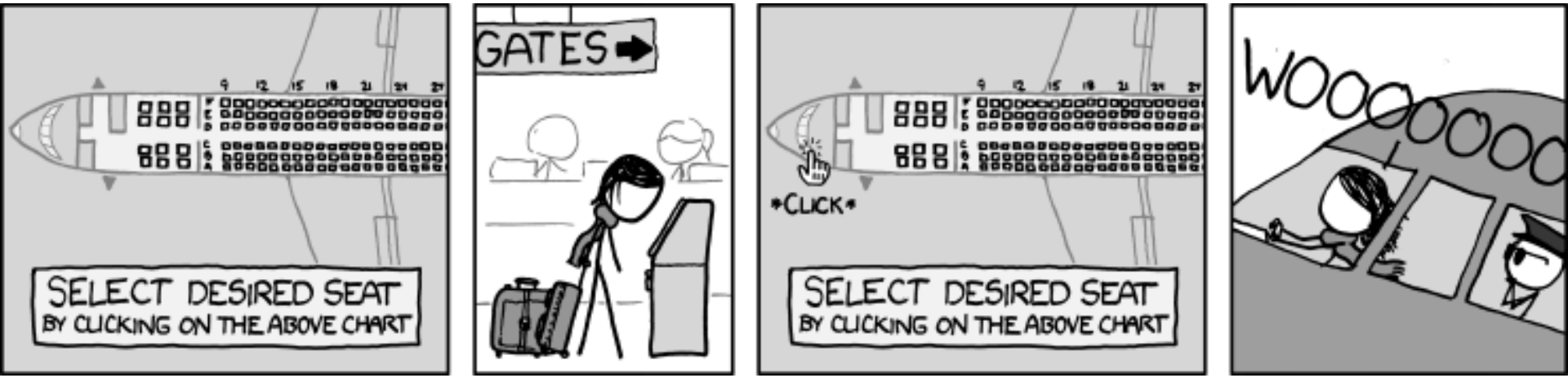



Image Credit : xkcd.com/726/

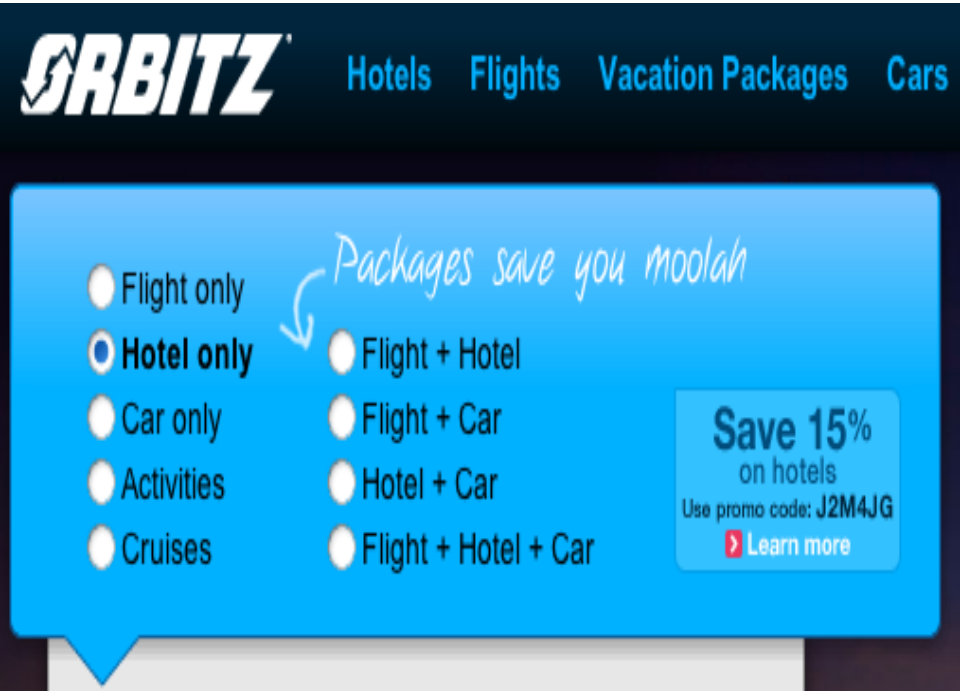
# Motivation


**Hidden Web Databases to query :**

- Airline websites (Ticket details)
- SeatGuru (Leg room)
- FlightStats (On-time record)
- US DOT (Lost luggage record) and many more

Image Credit : intrip.com

# Motivation





**Aim :** Find top-10 cheapest hotels from Orbitz with excellent TripAdvisor ratings.

Image Credit : orbitz.com and tripadvisor.com

# Breaking the top-*k* barrier

- One solution to solve all problems
  - Break the top-*k* barrier

- What information does every top-*k* query provide?
  - Total ordering information of top-*k* matched tuples
  - Ordering information between matched top-*k* and non top-*k* tuples

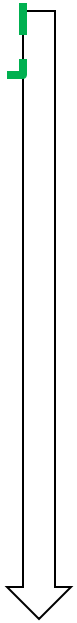
# Breaking the top-*k* barrier

- Idea for breaking top-*k* barrier:
  - Ask slightly different but related queries
  - Infer next ranked tuple from query results

- GetNext operator for hidden databases
  - Given top-*h* tuples retrieve tuple ranked *h+1*
  - Challenging for hidden databases

# Running Example

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| t1 | No | Bad | Bad | No | Good | $500 |
| t2 | No | Bad | Bad | Yes | Good | $525 |
| t3 | No | Bad | Good | No | Good | $550 |
| t4 | No | Good | Good | Yes | Good | $575 |
| t5 | Yes | Good | Good | No | Good | $600 |
| t6 | Yes | Good | Good | Yes | Good | $625 |
| t7 | No | Bad | Bad | No | Bad | $700 |



**K = 3**

**Query Structure :** SELECT * FROM D [WHERE predicates except price]

# Tuple domination

- Comparing a pair of tuples
  - To identify tuple with higher rank
  - Issue most specific query
  - Infer higher ranked tuple from results

- Higher ranked tuple is said to *directly dominate* the lower ranked one

# Comparing Tuples

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| **t1** | **No** | **Bad** | **Bad** | **No** | **Good** | **$500** |
| t2 | No | Bad | Bad | Yes | Good | $525 |
| **t3** | **No** | **Bad** | **Good** | **No** | **Good** | **$550** |
| t4 | No | Good | Good | Yes | Good | $575 |
| **t5** | **Yes** | **Good** | **Good** | **No** | **Good** | **$600** |
| t6 | Yes | Good | Good | Yes | Good | $625 |
| t7 | No | Bad | Bad | No | Bad | $700 |

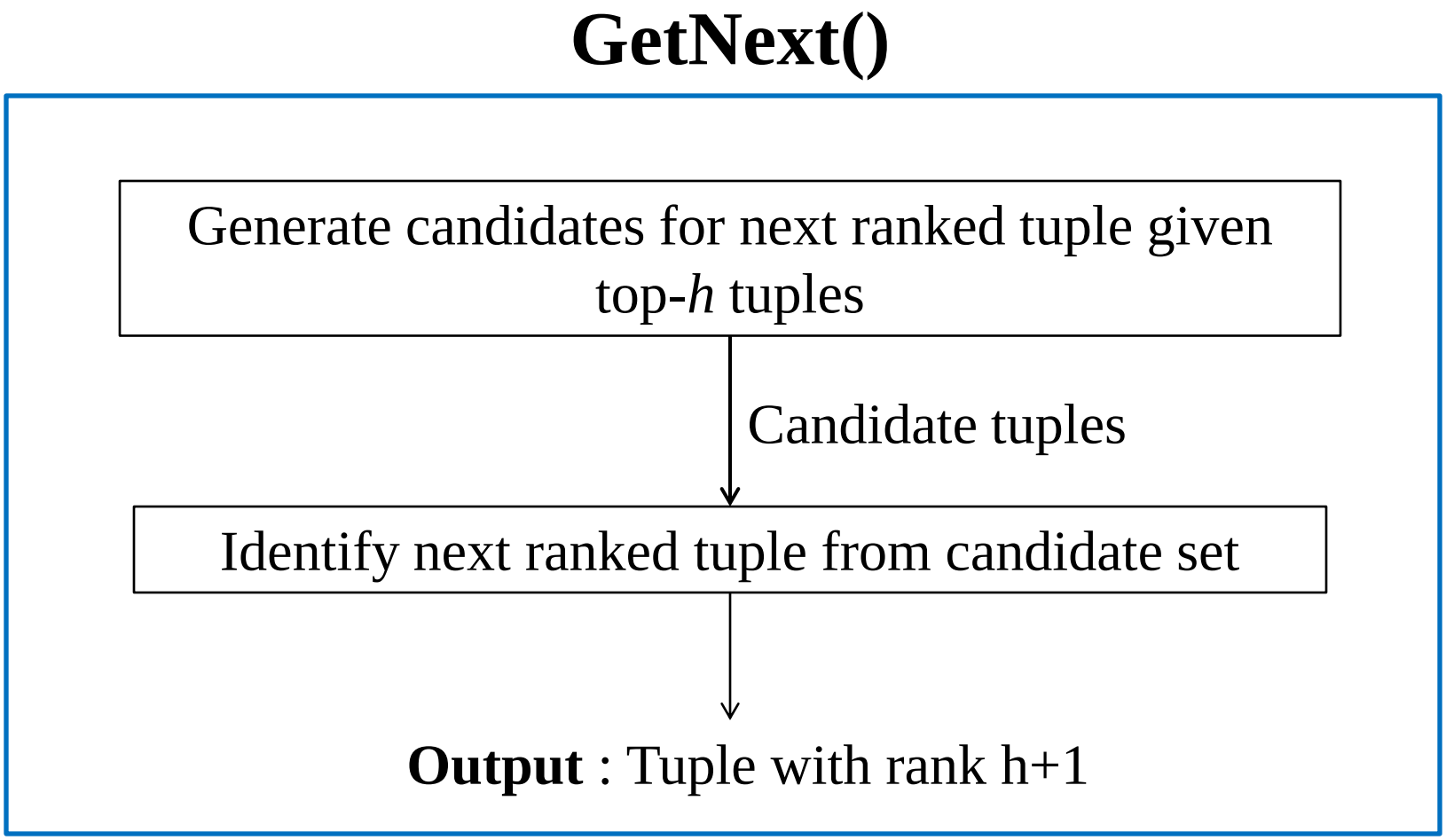
- Tuples t5 and t7 are directly comparable
  - Most specific query : SELECT * FROM D WHERE Wifi = No

# Comparing Tuples

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| **t1** | **No** | **Bad** | **Bad** | **No** | **Good** | **$500** |
| **t2** | **No** | **Bad** | **Bad** | **Yes** | **Good** | **$525** |
| **t3** | **No** | **Bad** | **Good** | **No** | **Good** | **$550** |
| t4 | No | Good | Good | Yes | Good | $575 |
| t5 | Yes | Good | Good | No | Good | $600 |
| t6 | Yes | Good | Good | Yes | Good | $625 |
| t7 | No | Bad | Bad | No | Bad | $700 |

- Tuples t6 and t7 are not directly comparable
  - Most specific query : SELECT * FROM D

# Meta Algorithm



# Candidate Generation

- Given top-*h* tuples, produce a set of candidates for next ranked tuple.

- Candidate set must be
  - Minimal
  - Complete

# Candidate Generation

**K=3, SELECT * FROM D** **Aim :** Get tuple with rank 4

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| t1 | No | Bad | Bad | No | Good | $500 |
| t2 | No | Bad | Bad | Yes | Good | $525 |
| t3 | No | Bad | Good | No | Good | $550 |

**Idea :** Select a set of queries that

- Each match less that *k* tuples
- Jointly covers all tuples in database

All top ranked unseen tuples form candidate set.

# Candidate Generation

**K=3, SELECT * FROM D**

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| t1 | No | Bad | Bad | No | Good | $500 |
| t2 | No | Bad | Bad | Yes | Good | $525 |
| t3 | No | Bad | Good | No | Good | $550 |

**Queries :**

SELECT * FROM D WHERE Legroom = Bad {t7}

SELECT * FROM D WHERE Legroom = Good{t4, t5}

**Candidates :** {t4, t7}

# Candidate Generation

**K=3, SELECT * FROM D**

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| t1 | No | Bad | Bad | No | Good | $500 |
| t2 | No | Bad | Bad | Yes | Good | $525 |
| t3 | No | Bad | Good | No | Good | $550 |

**Queries :**

SELECT * FROM D WHERE Legroom = Bad and Wifi = No

SELECT * FROM D WHERE Legroom = Bad and Wifi = Yes

SELECT * FROM D WHERE Legroom = Good and Wifi = No

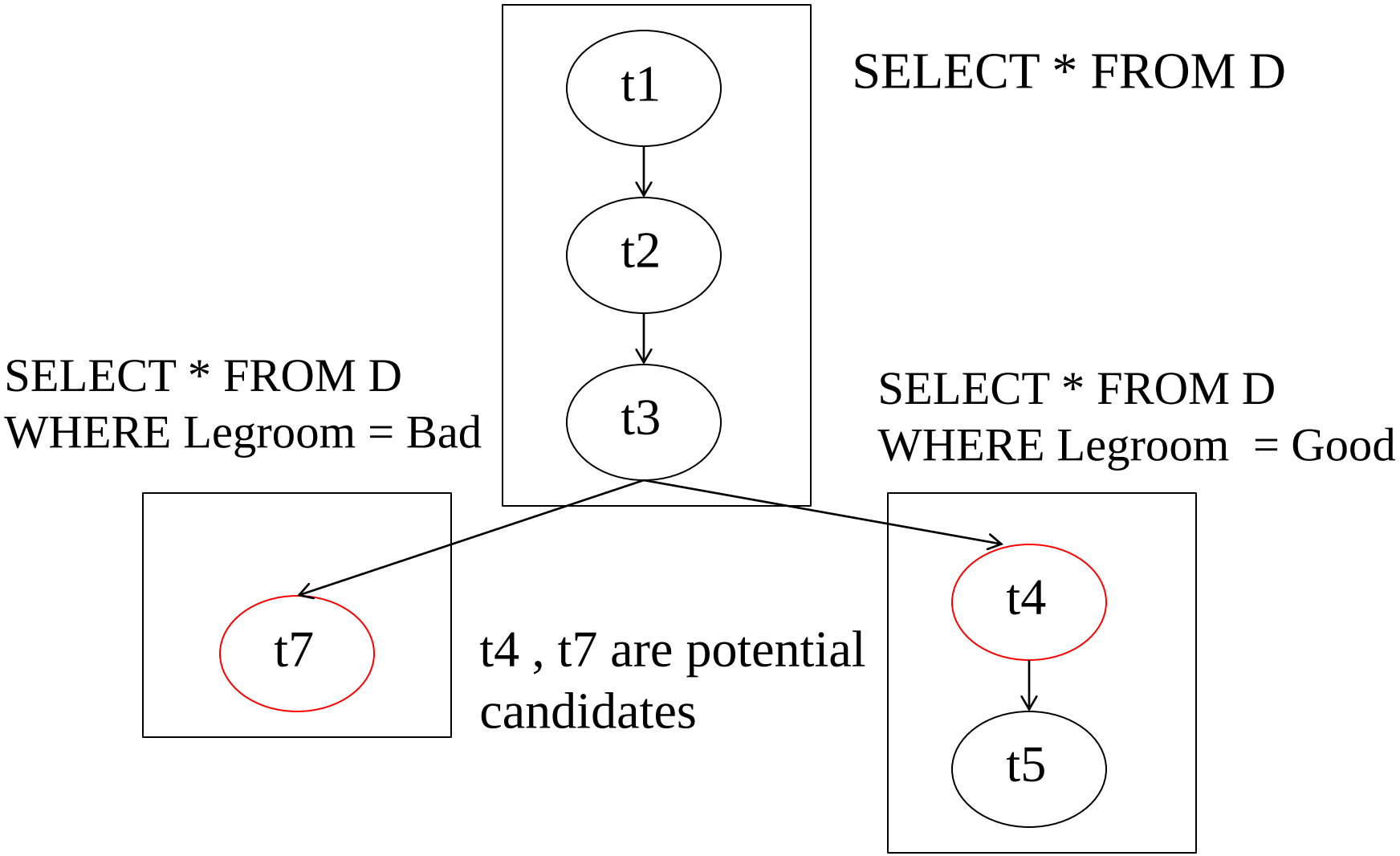
SELECT * FROM D WHERE Legroom = Good and Wifi = Yes

# Candidate Generation

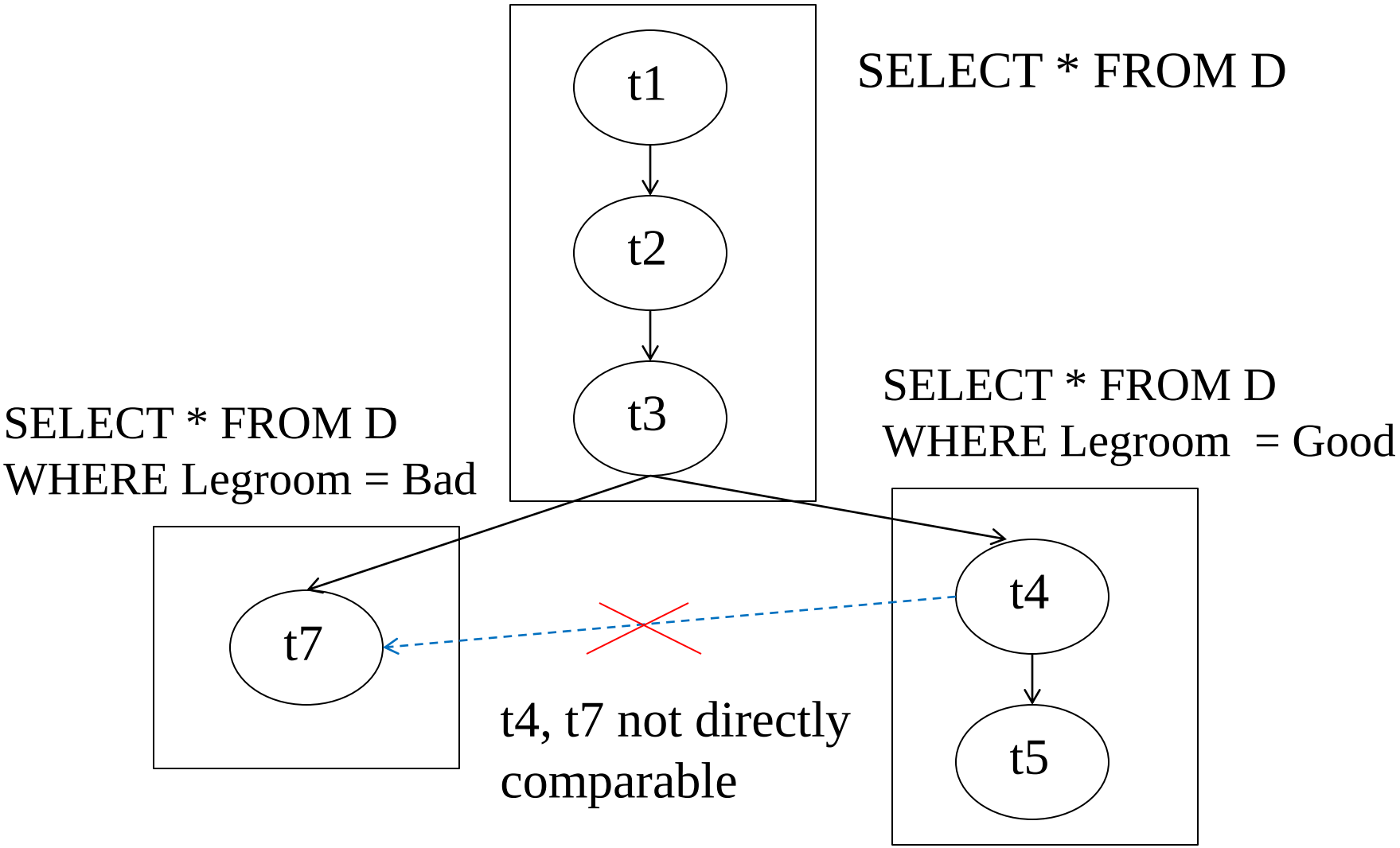
**We can do better !**

- Use all available information from results
- Retrieve multiple tuples in a single GetNext call
- **Idea :** Dominance graph

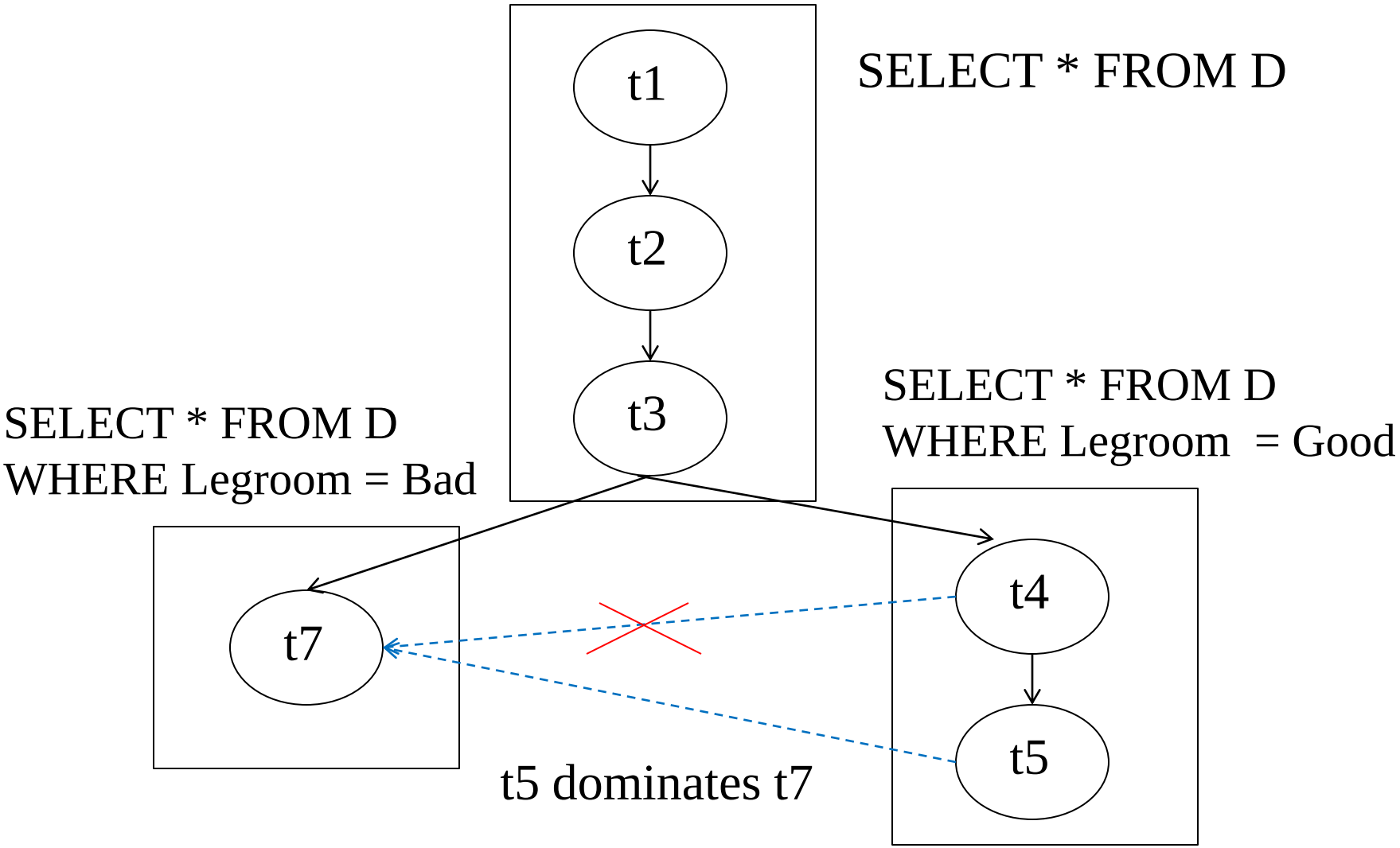
# Candidate Generation



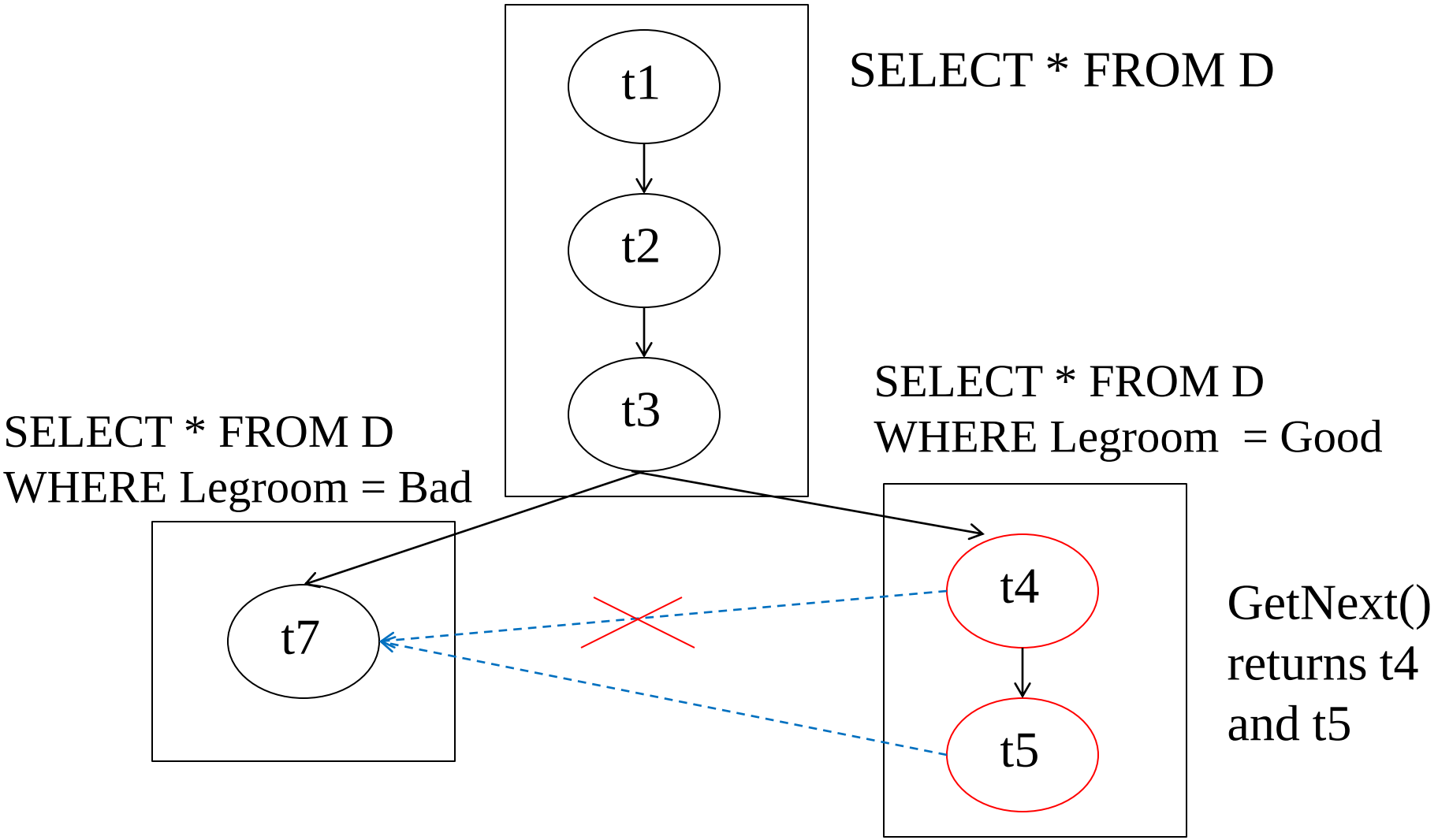
# Candidate Generation



# Candidate Generation



# Candidate Generation



# Candidate Generation

- Identify a set of queries that partition top-*h* tuples and jointly cover entire database
- Execute all queries
- Top tuple from each query that is not dominated form the candidate set

# Candidate Testing

- What we have?
  - Set of candidate tuples one which is guaranteed to be the next ranked tuple
- Simple cases :
  - Only one tuple in candidate set
  - Ranking function is known and can be computed locally
- **Aim :**
  - Identify the next ranked tuple from candidate set

# Candidate Testing

- Bad solution :
  - Crawl entire database and pick next ranked tuple

- **Observation :** Only top-*h* tuples can dominate tuple ranked *h+1*.

# Better Idea - I

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| t7 | No | Bad | Bad | No | Bad | $700 |

- Execute all 2^m queries matching tuple t7 (m = queryable #attributes)
- Verify that t7 is not dominated by non top-*h* tuples.

Single predicate queries matching t7

SELECT * FROM D WHERE Free-Luggage = No

SELECT * FROM D WHERE Luggage-Record = Bad

….

Two predicate queries matching t7

SELECT * FROM D WHERE Free-Luggage = No AND Luggage-Record = Bad

….

# History Inference

| Rank | Free Luggage? | Luggage record | Legroom | Wifi | On-time Record | Price |
|---|---|---|---|---|---|---|
| t7 | No | Bad | Bad | No | Bad | $700 |

Q1 : SELECT * FROM D WHERE Free-Luggage = No AND Luggage-Record = Bad *{t1, t2, t3}*

Q2 : SELECT * FROM D WHERE Free-Luggage = No AND Luggage-Record = Bad AND Legroom = Bad *{t1, t2, **t7**}*

- Do not issue any query returning only tuples from top-*h*
- Ignore Q1 in favor of Q2

# Better Idea - II

- **Beyond-h minimal queries:**
  - Must return at-least one non top-*h* tuple
  - Any subset of its predicates must only return top-*h* tuples

- Mapping between beyond-*h* minimal queries and minimal infrequent itemsets with threshold k/h
- Requires much less than 2^m queries

# Better Idea - III

- **Query Ordering:**
  - Consider beyond-*h* queries of all candidate tuples simultaneously
  - Reorder queries based on their elimination potential
    - #candidate tuples matched
    - #expected tuples in Database matched

# Algorithm

- **GetNext:**
  - Generate candidates given top-*h* tuples
  - Collect beyond-*h* queries for all candidates and reorder them
  - For each query :
    - Reject any candidate tuple dominated by other candidates or non top-*h* tuple
    - Repeat till only one tuple left

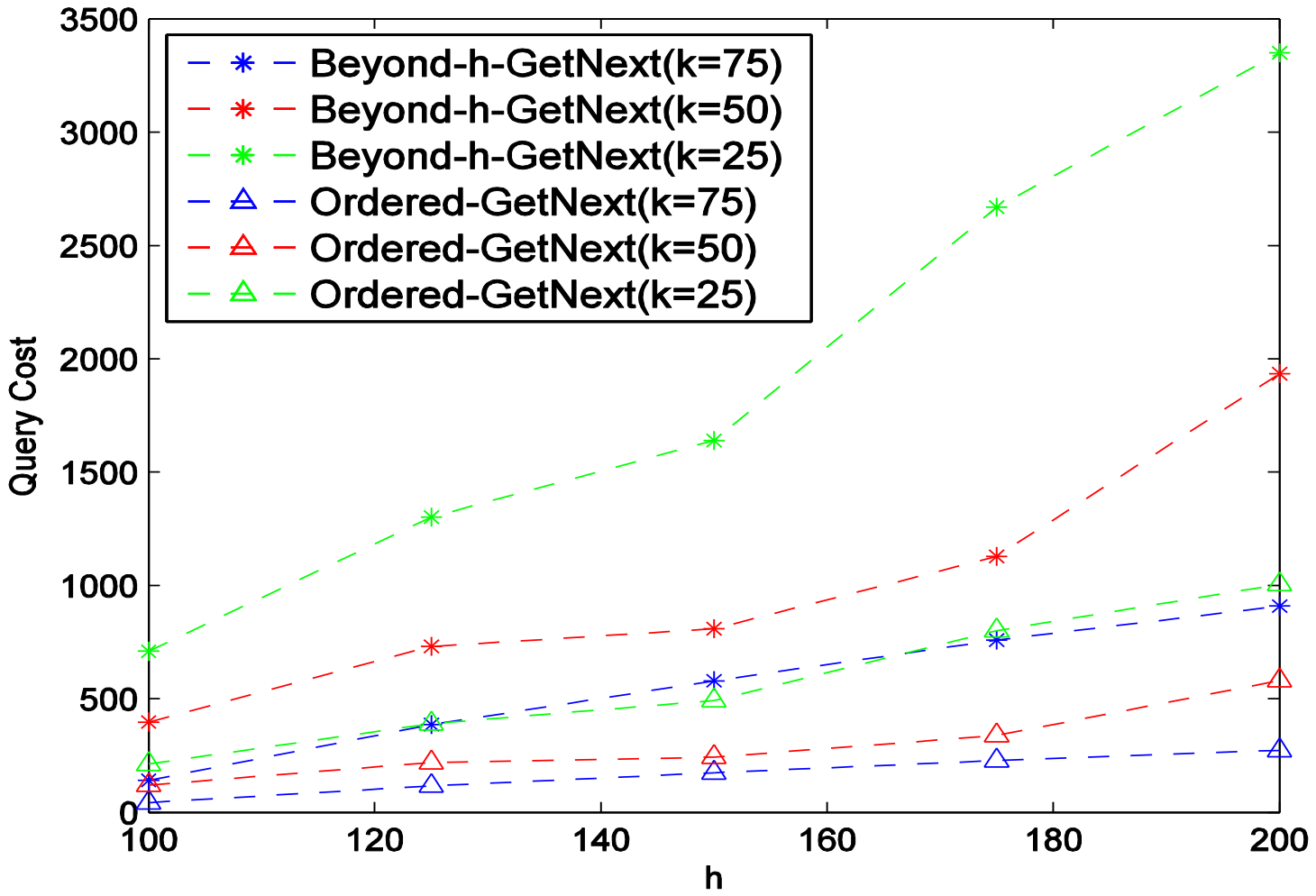
# Extensions

- Absence of total order in top ranked tuple
  - Affects candidate testing only
  - Randomly generate some valid total order from inferred partial order
- GetNext with selectivity constraints
  - Prefix condition to each queries
  - Some tuples may not be directly comparable anymore

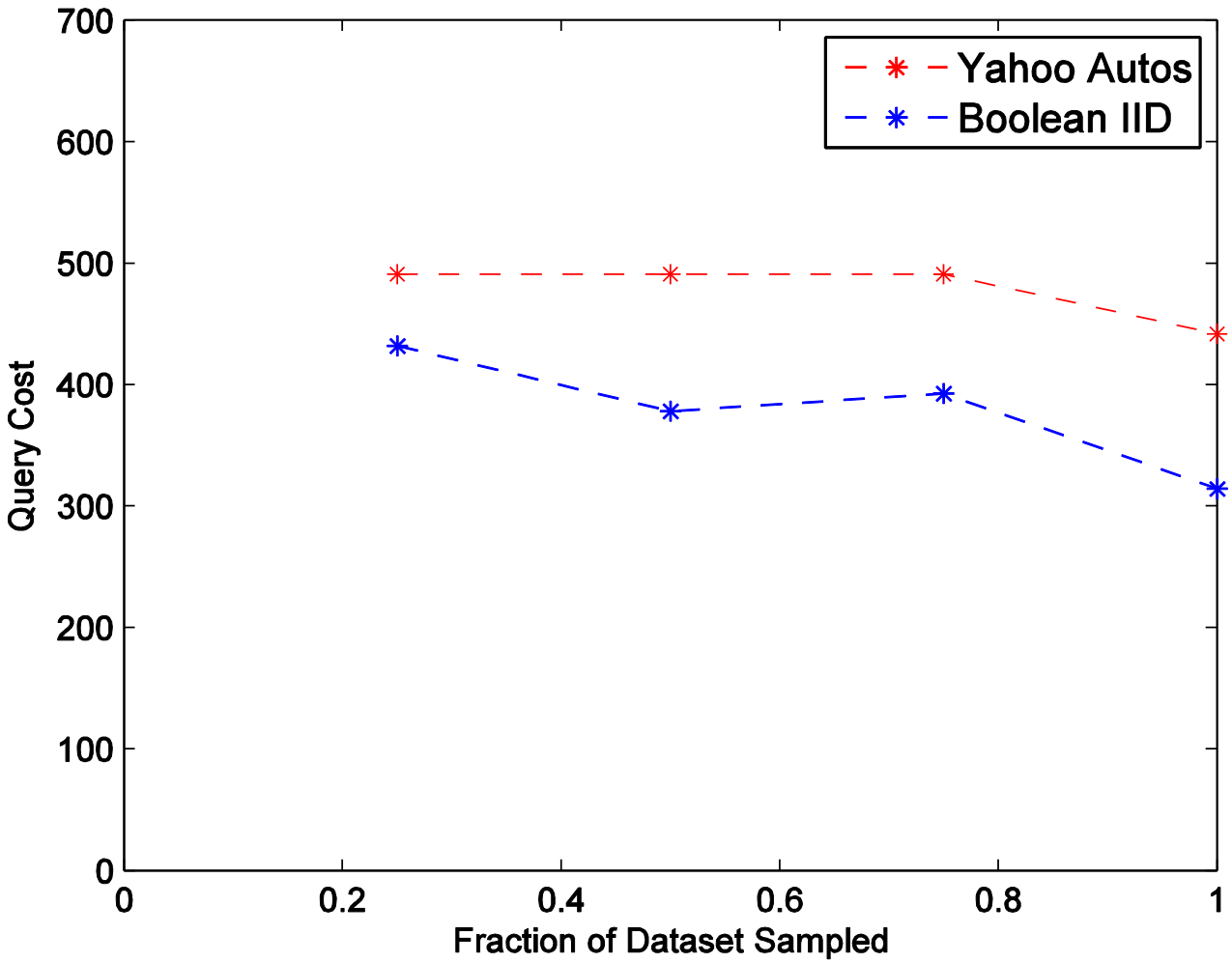
# Experiments

- Datasets
  - Synthetic boolean iid dataset with 200K tuples and 80 attributes
  - Yahoo Autos dataset with 200K tuples and 38 attributes (domain size 2 to 16)
  - Ranking : attribute based/ random permutation

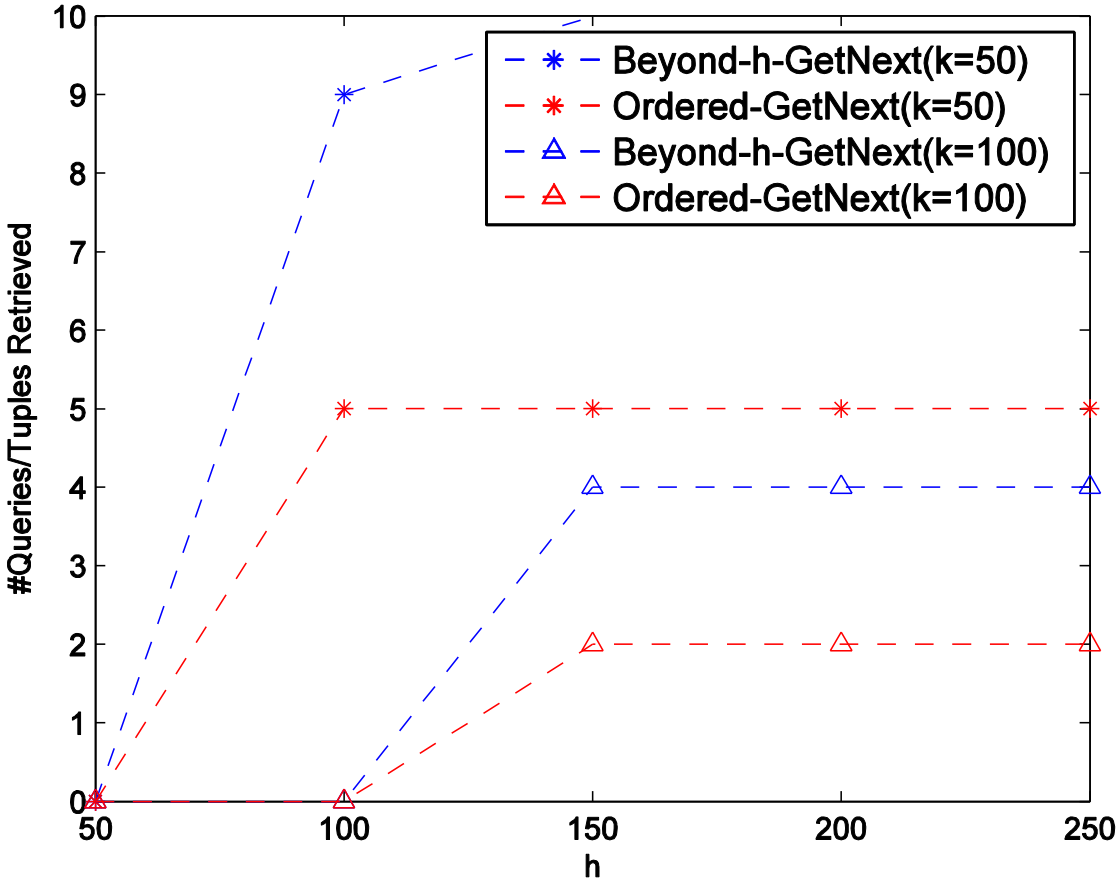
# Experiments



# Experiments



# Online Experiment over Amazon



# Conclusion

- Introduced GetNext operator for hidden databases
- Ordered crawling of hidden databases
- Enables a number of innovative third party applications
- Generate and test framework
- Comprehensive set of experiments