HÁSKÓLINN | REYKJAVÍK
REYKJAVIK UNIVERSITY

Practical Considerations for Semantic Cache Management: Performance and Overhead

Björn Þór Jónsson
Reykjavík University

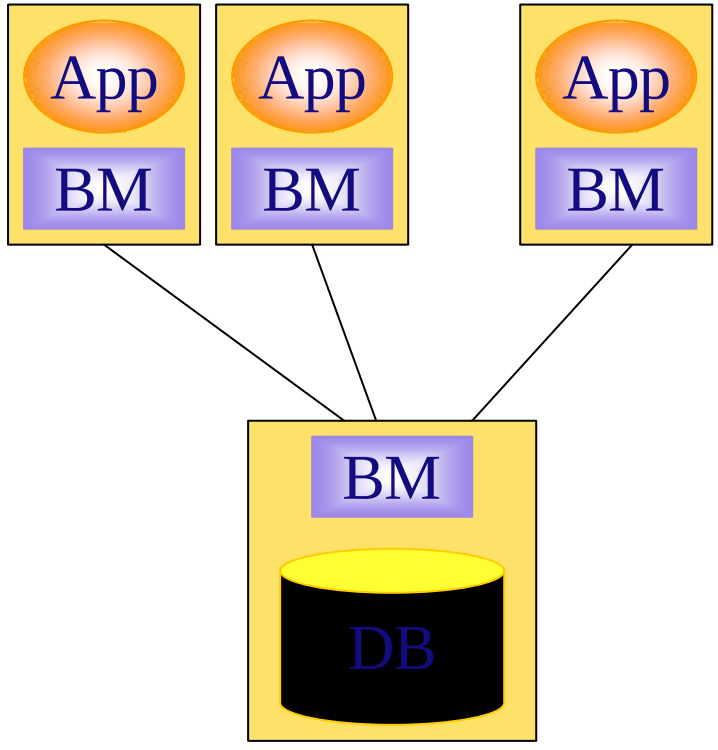
Michael J. Franklin
Divesh Srivastava

Database Caching

- Going on for more than a decade
- Many different (lines of) architectures
- Motivated by applications

Engineering Applications

- CAD/CAM/CASE applications
  - Object-oriented features
  - Short interactions
  - Long transactions
  - Strict consistency – rare conflicts



- Physical Caching
  - Distributed buffer management

Physical Caching Work

- "The DeWitt paper"
  - Object-server vs. Page-server
  - Hybrid caching (Kossmann & Kemper)
- Carey, Franklin & co
  - Consistency protocols, disk caching, global caching, ...
- Systems
  - O_2, GemStone, Exodus, ...

Data Warehousing

- OLAP type applications
  - Relational features
  - Long queries
  - Controlled updates
  - Relaxed consistency
- Logical caching
  - Materialized views
  - Cube-slice caching

Logical Caching Work

- Stanford DW project
- Berkeley DW project
- Maryland DW project
- IYFU DW project
- IYFU MiddleWare project
- Research systems, commercial systems

Early Web

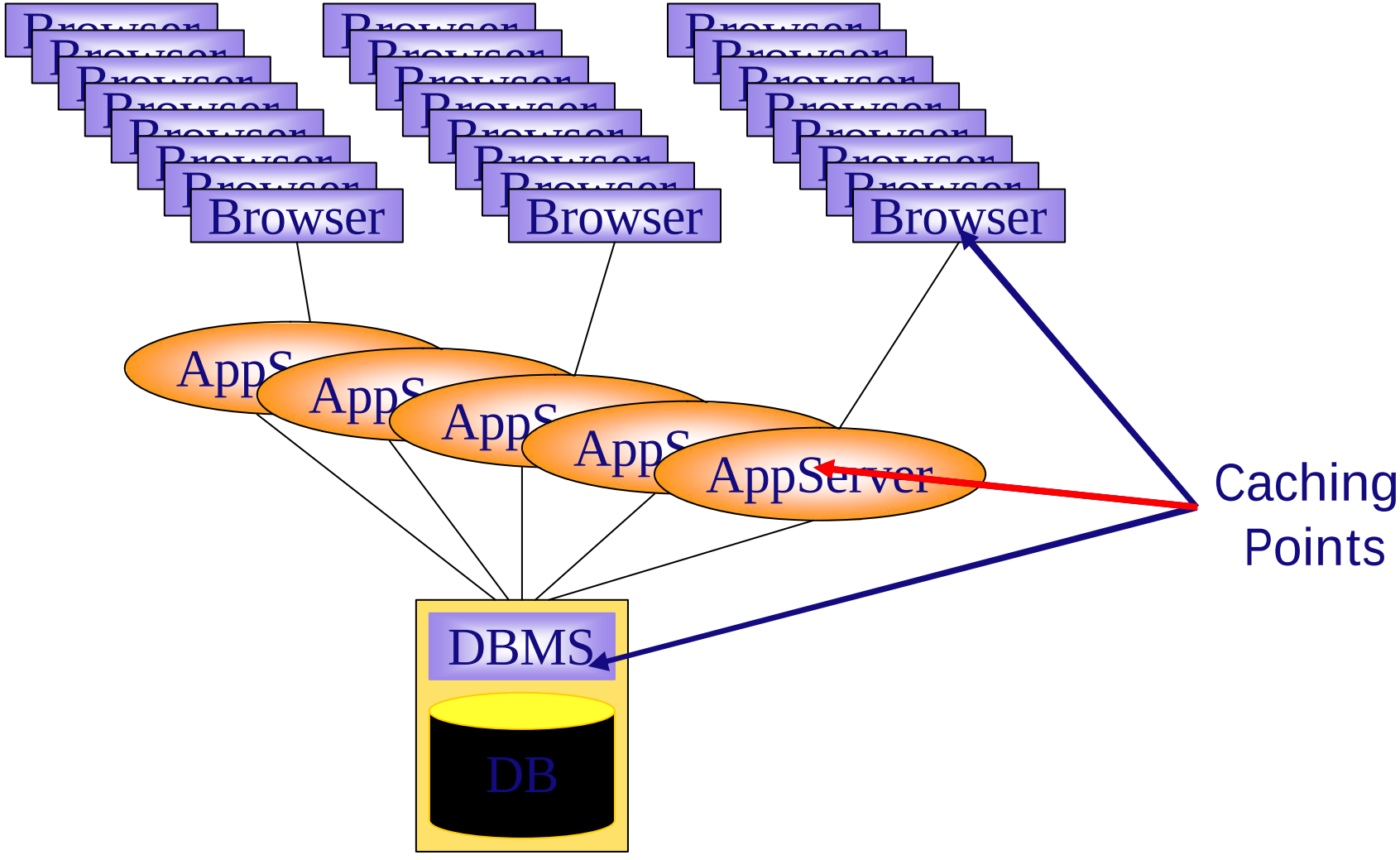
- Web browsing
  - Hyperlinks to URLs
  - “Point” queries
  - Very relaxed consistency

- Proxy caching
  - Similar to physical caching (URL ≈ Page ID)
  - Less interesting workloads

Recent Web

- Query-based On-line “Stuff”
  - News services, search engines, rental services, bibliography servers, e-commerce!
- The useful part of the Web

Architecture



Application Servers

- Reside on top of “legacy” servers
- Fairly simple queries (mostly)
- Moderate (varied) consistency
- Very high peak loads
- Potentially a great place to cache data

AppServer Caching

HÁSKÓLINN | REYKJAVÍK
REYKJAVIK UNIVERSITY

- Tables
  - Replication (?)
  - IBM, Oracle
- Web-page fragments
  - Oracle
- Static table fragments
  - TimesTen
- Semantic caching
  - TBD

Semantic Caching

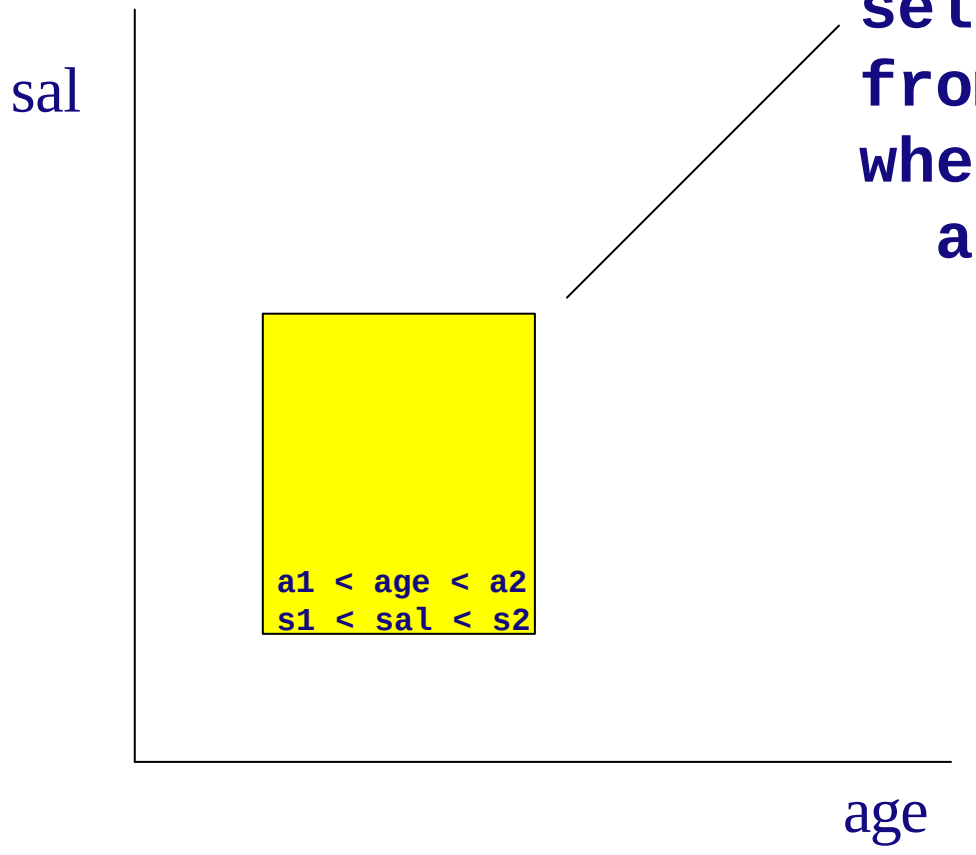
- First described in VLDB ’96 paper
  - Simulation model
- Selection predicates “in-cache”
  - Bring in data
  - Manage cache contents
- Logical caching variant
  - No overlap of data or cache description
- Semantic regions
  - Granularity of cache management
  - Application oriented replacement policies

Overview

- Motivation
- Model of semantic caching
- Prototype implementation
- Nice workloads
- Nasty workloads
- Conclusions and future work

Cache Contents

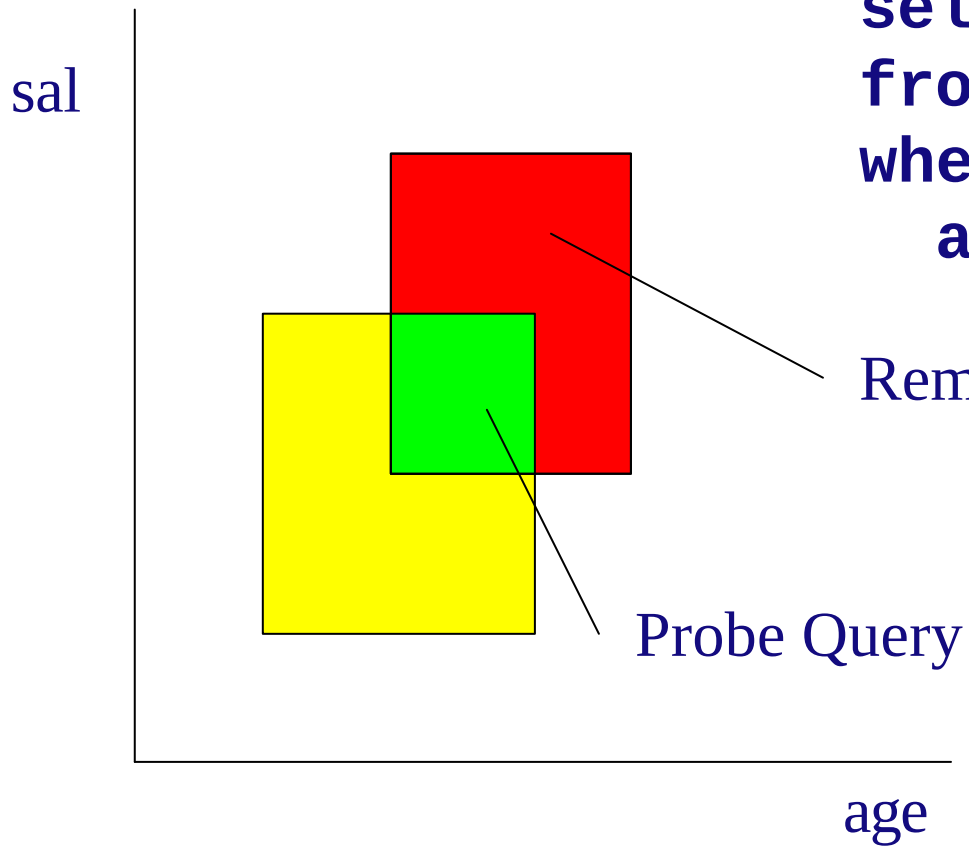
- Cache contents organized by queries




```
select *
from   R
where  a1 < age < a2
  and  s1 < sal < s2
```

Remainder Queries

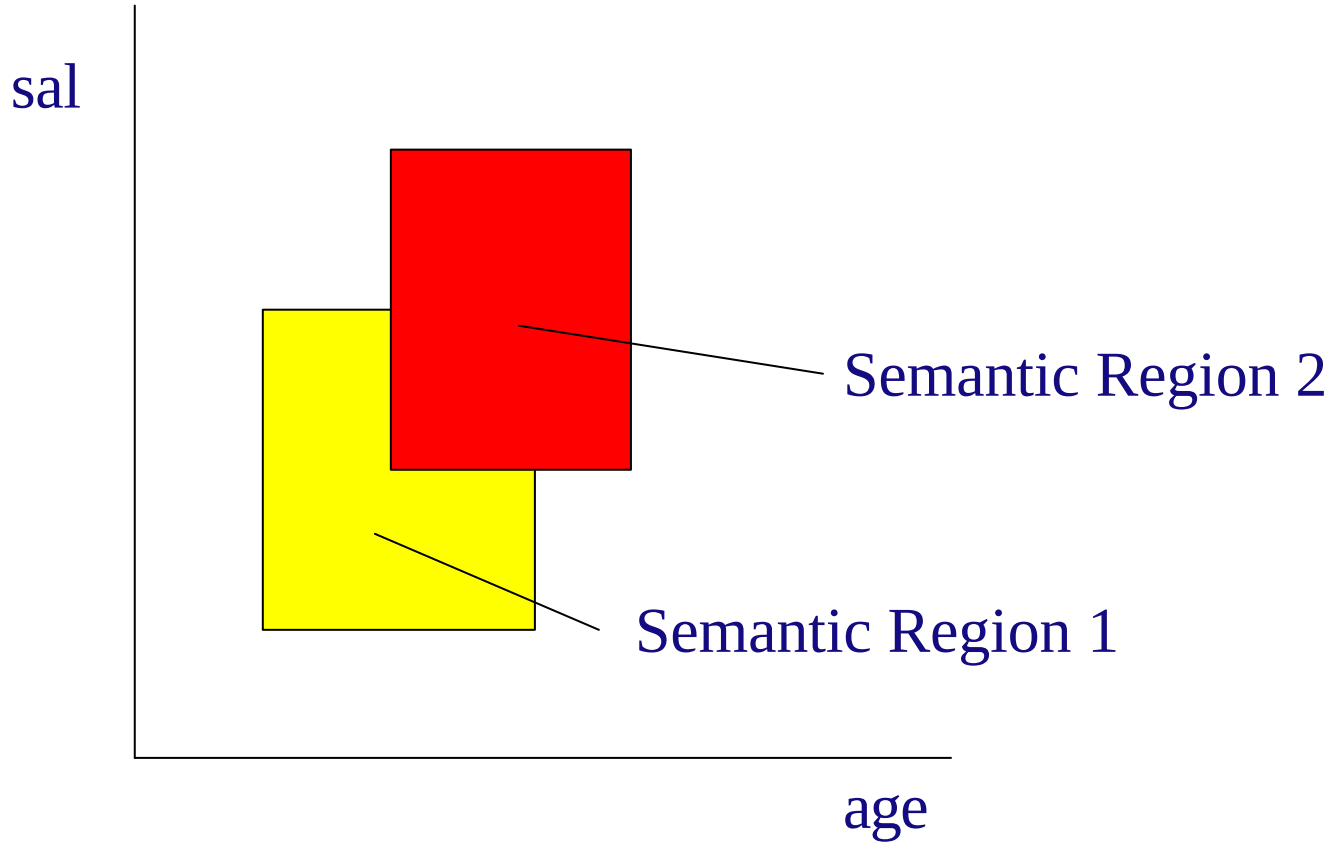
- Only missing data fetched from server




```
select *
from   R
where  a3 < age < a4
  and  s3 < sal < s4
```

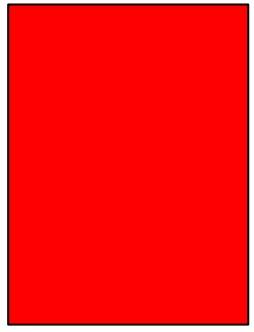
Cache Granularity

- Cache is organized by semantic regions



Semantic Regions

- Granularity of replacement
- Aggregate cache overhead
  - Variable sized
  - May shrink over time
- Described by a predicate
  - May be used to formulate a new replacement policy



Selection Predicates

- Constraint formulas handled “in=cache”
  - Formula domain closed under conjuction, disjunction and negation
  - Prototype deals with range queries
- For query Q, cache contents V
  - Probe query = $Q \wedge (V)$
  - Remainder query = $Q \wedge (\neg V)$

Prototype

- Written in C++
- Connects to relational servers via ODBC
- Deals with range queries

The competition

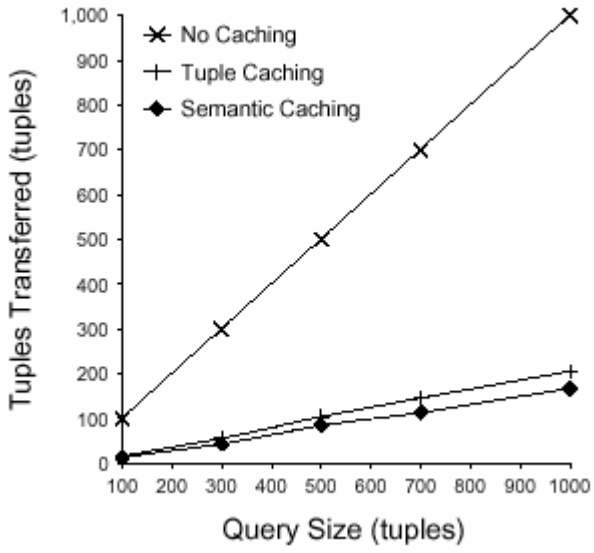
- Simulation study
  - Page Caching ≈ Page Server
  - Tuple Caching ≈ Object Server
  - Neither works well against a relational server
- Tuple Caching´
  - Fake OIDs using a key
  - Use knowledge of DB to simulate indexes
- No Caching

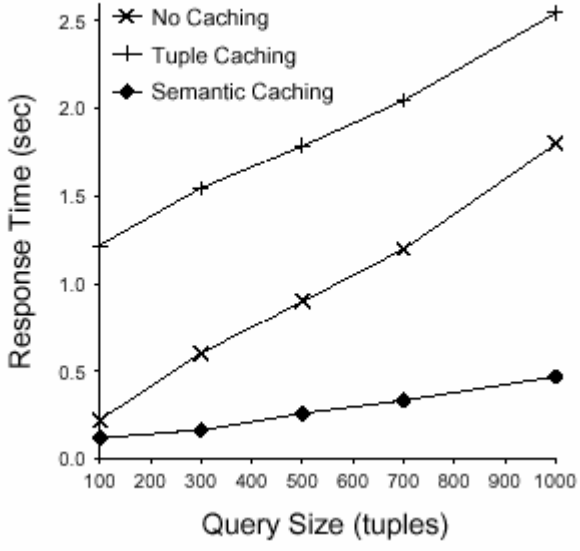
Nice Workloads

- Use 1M Wisconsin relation
- Range selections of 100-1,000 tuples
- Hot spot of 10,000 tuples
  - 90% of queries have “center” in hot spot
- Selections on various attributes
  - Clustered
  - Unclustered
  - Unindexed
  - Clustered & unindexed attributes

Clustered Index

- Tuples Transferred
  - SC contains hot spot
  - TC has slightly higher overhead
- Response Time
  - SC relatively slower due to query opt
  - Poor query optimization for TC





Two Attributes I

HÁSKÓLINN | REYKJAVÍK
REYKJAVIK UNIVERSITY

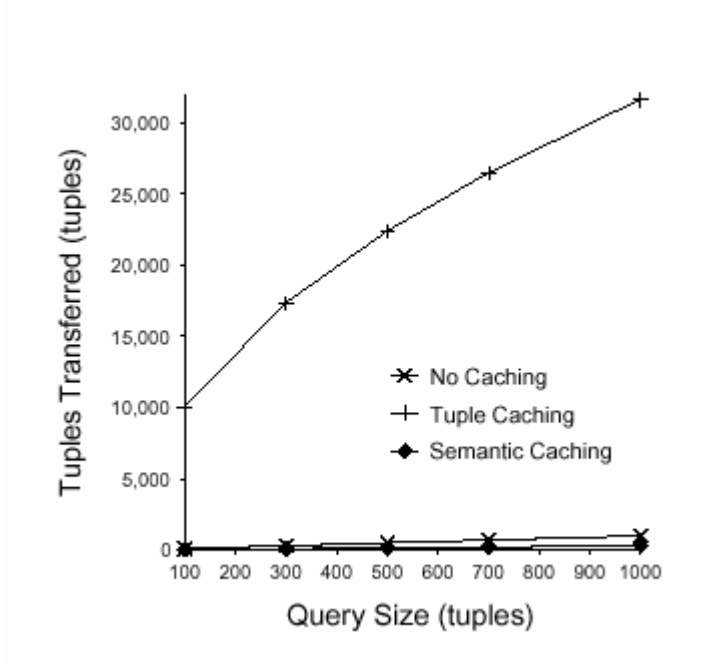
One attribute

```
SELECT *
FROM Wisc1M
WHERE Unique2 > 4999 AND Unique2 < 6000;
```

Two attributes

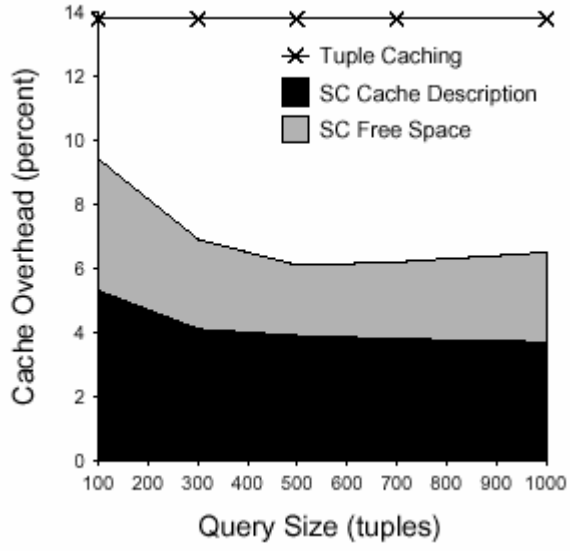
```
SELECT *
FROM Wisc1M
WHERE Unique2 > 39999 AND Unique2 < 50000
  AND Unique3 > 24999 AND Unique3 < 35000;
```

- Tuples transferred
  - TC ships all tuples satisfying constraint on Unique2
  - SC performs as before



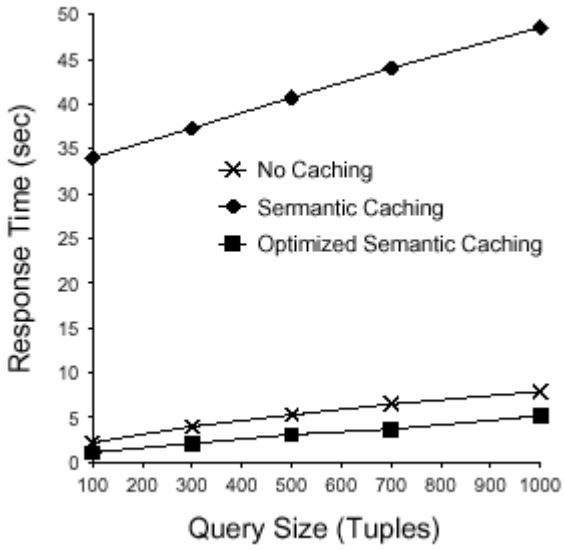
Two Attributes II

- Cache Overhead
  - TC has fixed overhead
  - SC has variable overhead
    - Regions vary in size
    - Discarded region may leave empty space
  - SC has lower overhead
    - Semantic regions aggregate overhead
    - Even lower for one attribute



Two Attributes III

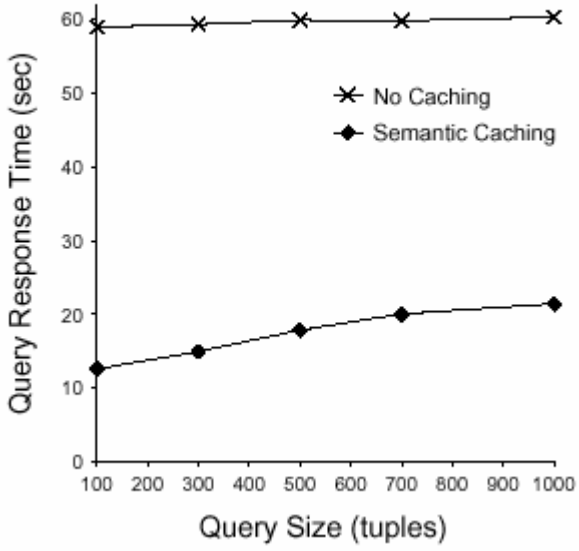
- Response Time
  - No more TC
  - Optimizer has problems with complex remainder queries
  - Client side optimization helps
    - Run as is
    - Individual factors
    - Bounding box




```
SELECT *
FROM Wisc1M
WHERE ((Unique2 > 145019 AND Unique2 < 156588) AND
       (Unique3 > 140709 AND Unique3 < 155615))
   OR ((Unique2 > 156587 AND Unique2 < 159297) AND
       (Unique3 > 140709 AND Unique3 < 155438))
   OR ((Unique2 > 159296 AND Unique2 < 170628) AND
       (Unique3 > 140709 AND Unique3 < 143918))
   OR ((Unique2 > 170627 AND Unique2 < 170965) AND
       (Unique3 > 140709 AND Unique3 < 141582));
```

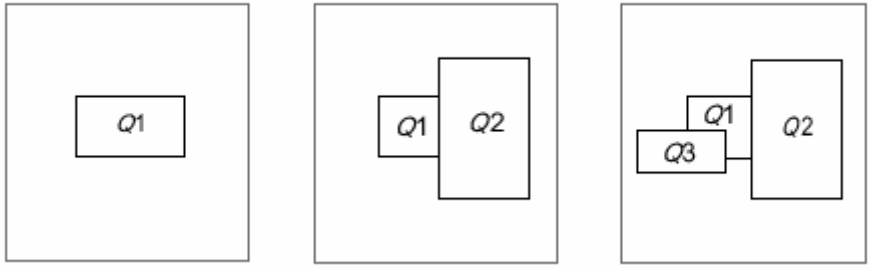
No Index

- Tuples transferred
  - As before
- Response time
  - For each query, a sequential scan
  - SC can avoid contacting server when entire result is in cache

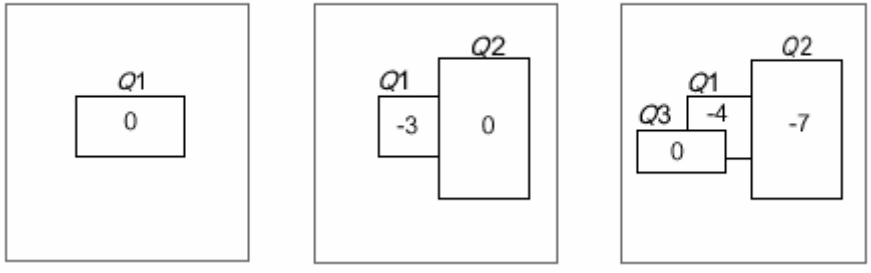


Replacement Policies I

- With LRU
  - Q1 is discarded

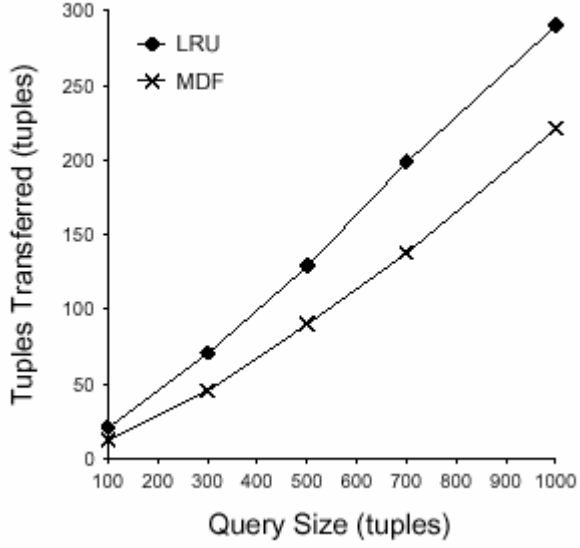


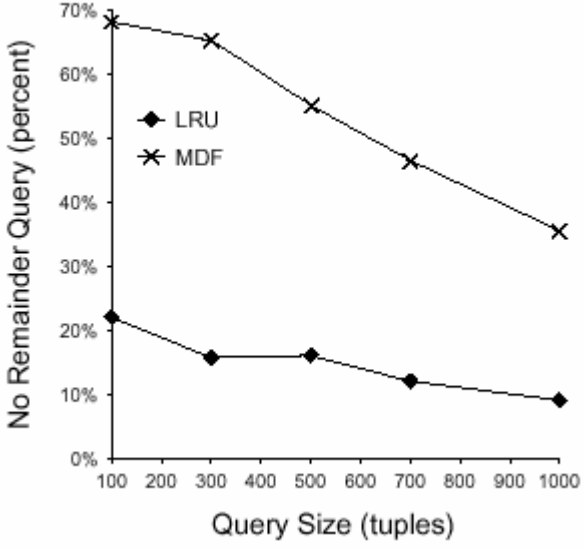
- With Manhattan distance
  - Q2 is discarded



Replacement Policies II

- Tuples Transferred
  - MDF works well for this workload
  - Throws out cold regions early
- No Remainder Query
  - MDF also avoids “gaps” in hot spot
  - Therefore more queries completely answered





Summary

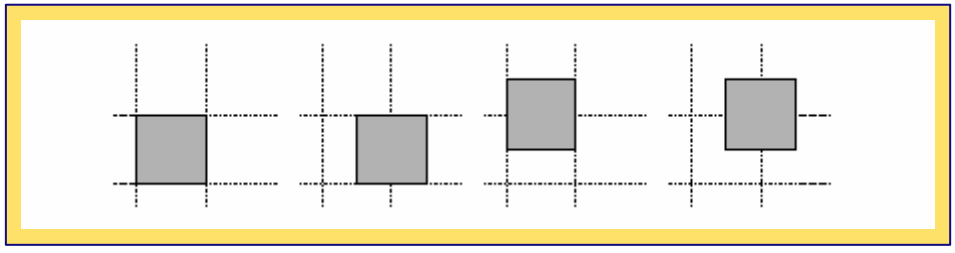
- The good:
  - Low cache overhead
  - Insensitivity to clustering
  - Reduced network traffic
  - Answer queries w/o talking to server
  - Application specific replacement policies
- The bad:
  - Complex remainder queries
  - Increased overhead going from one to two attributes

Nasty Workloads

- Add more attributes to queries
  - From 2 to 6 attributes
  - 1,000 tuple queries → make ranges wider
- Observe effects on
  - Server performance
  - Cache management overhead

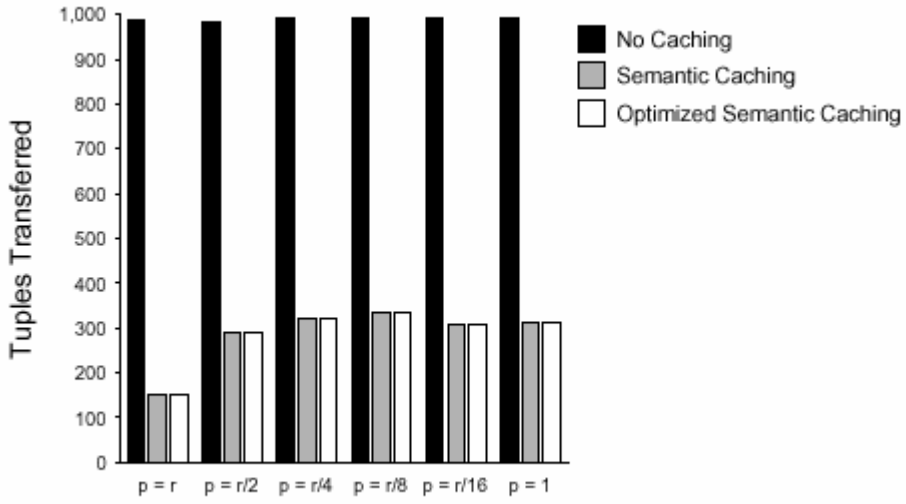
Overlap of Queries

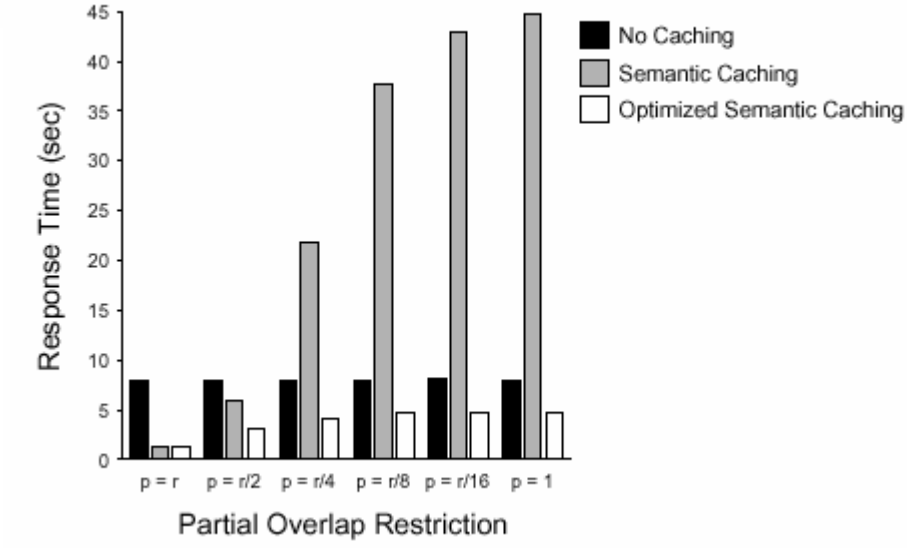
- Defined “partial overlap restriction”
  - p = r, r/2, r/4, r/8, r/16 and r/r = 1
  - p = r ➔ no partial overlap
  - p = 1 ➔ arbitray overlap
  - p = r/2 ➔ very restricted overlaps



Two Dimensions

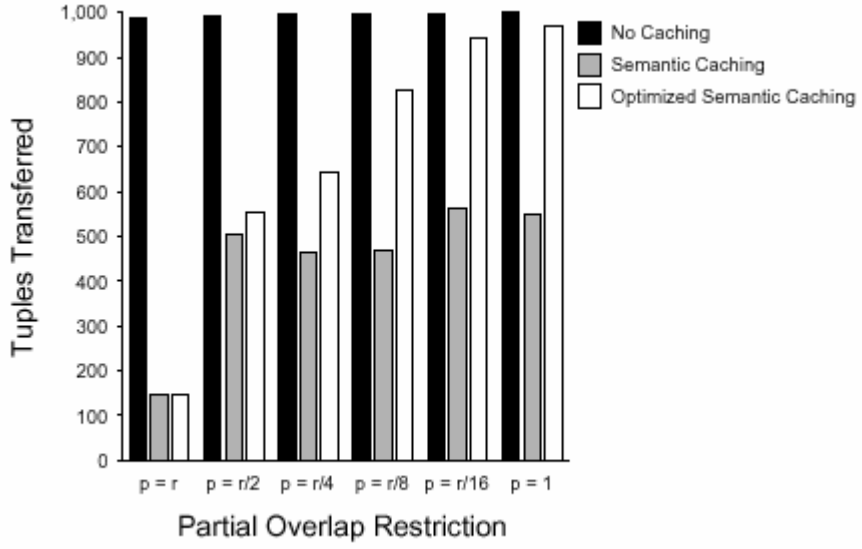
- Tuples Transferred
  - Best with no overlap
  - p = 1 is “nice” wl
- Response time
  - NoOpt → poor performance
  - Opt → similar to tuples transferred

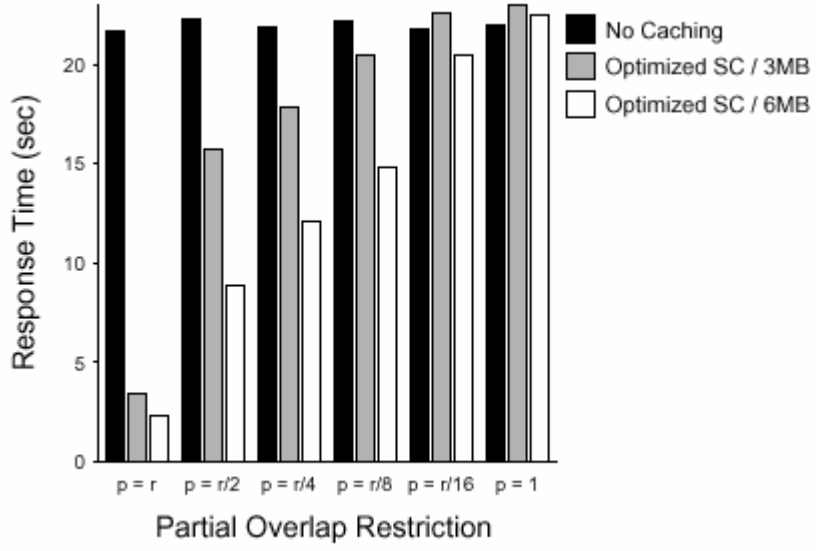




Three Dimensions

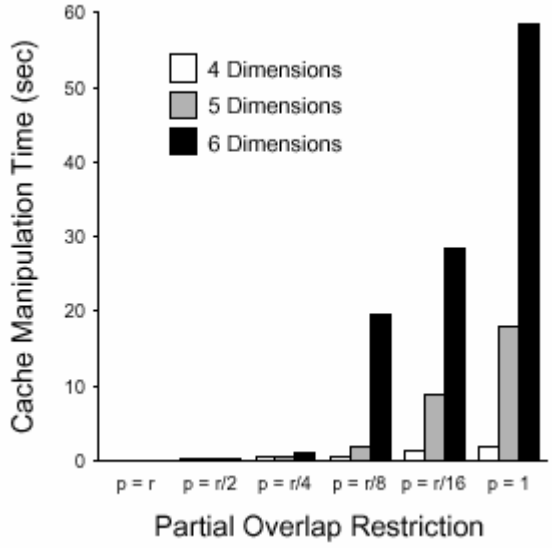
- Tuples Transferred
  - Smaller improvements
  - Better for larger cache
- Response time
  - Benefits disappear with increased overlap
  - No overlap still great





Four+ Dimensions

- Cache Manipulation Time
  - Very small for 3– dimensions
  - Grows exponentially with many dimensions & overlap
- Response time
  - No overlap ➔ stays same
  - No benefits otherwise



Summary

- Always good for non-overlapping workloads
- With many dimensions, response time gets worse
- Tuples transferred metric is better

Conclusions

- Results of study
  - Semantic caching model
  - Prototype implementation
  - Very good for simpler workloads
  - Performance degrades with workload complexity
- Future Work
  - Study e-commerce / on-line services applications
  - Extend prototype functionality