CoopSC
A Cooperative Database Caching Architecture

Andrei Vancea

Department of Informatics IFI, Communication Systems Group CSG, University of Zürich

ifi

Background

- Databases
  - Client / server architecture
  - Client : asks a query (SQL)
  - Server : returns the result (tuples)

- Data shipping architectures
  - Most of the query processing is performed on the client side
  - Data is sent from the server to the client at processing time
  - Client-side caching
    - Response time
    - Server load

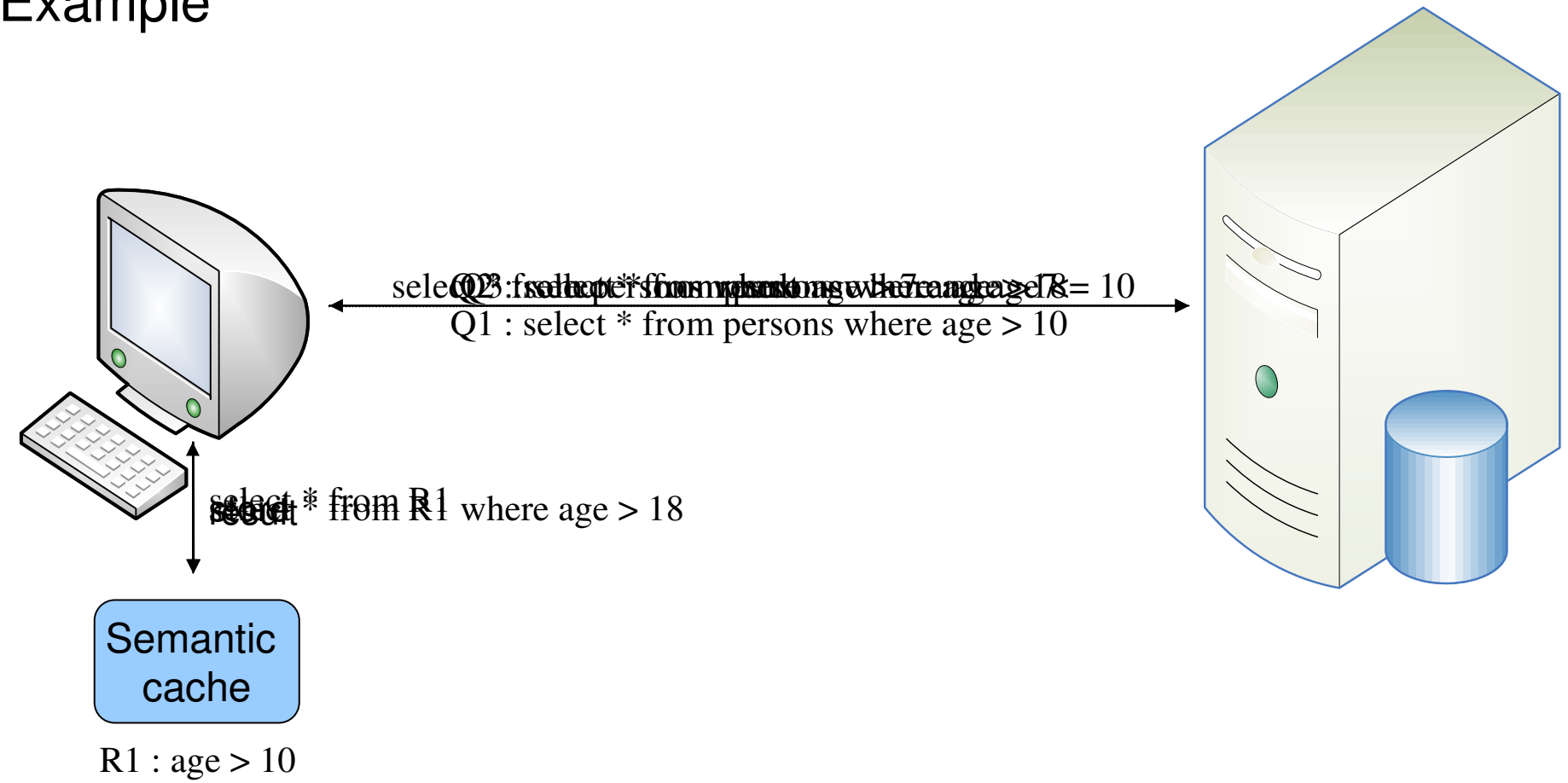
Background

- Client-side caching
  - Page caching
  - Tuple caching
  - **Semantic caching**

- Semantic caching
  - Clients store the results of old queries, together with their descriptions
  - Old query results are used when answering new queries

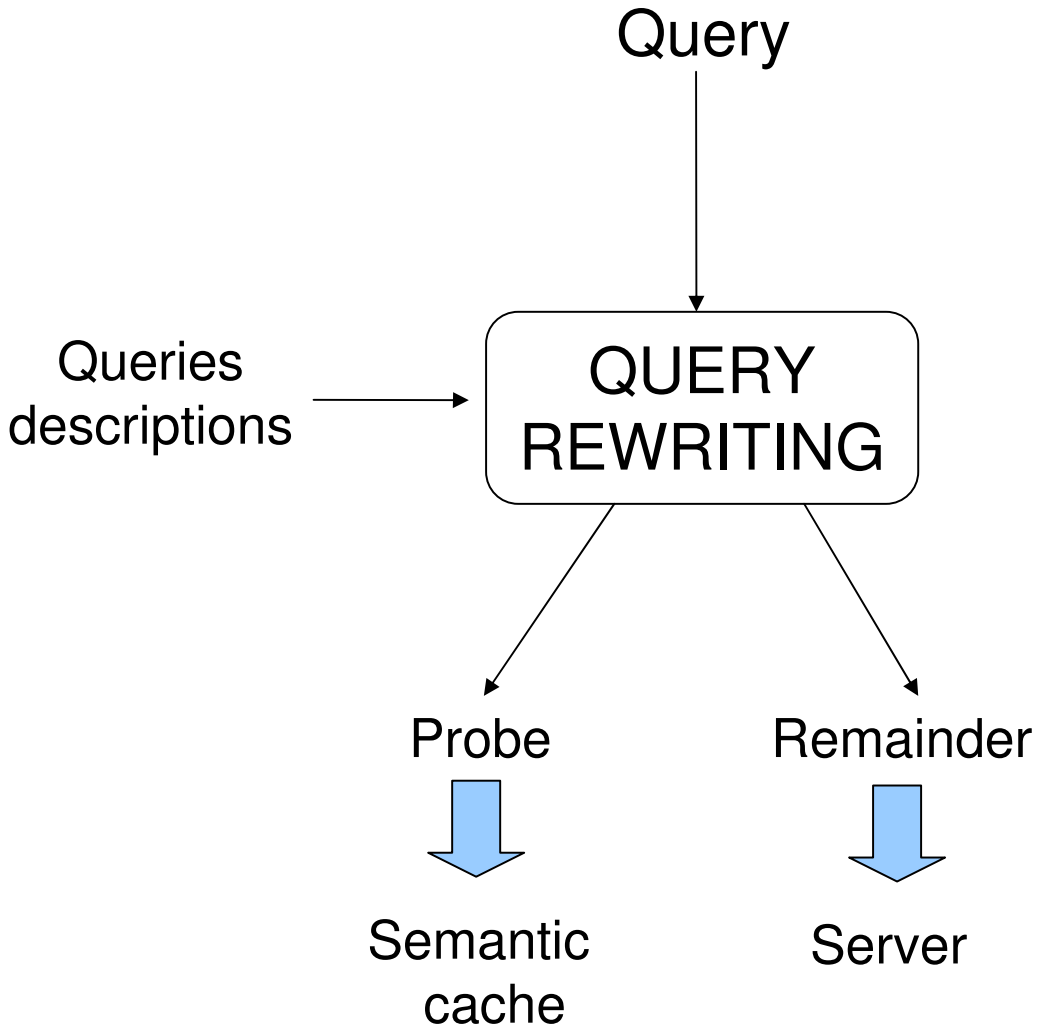
Background - Semantic Caching

Example



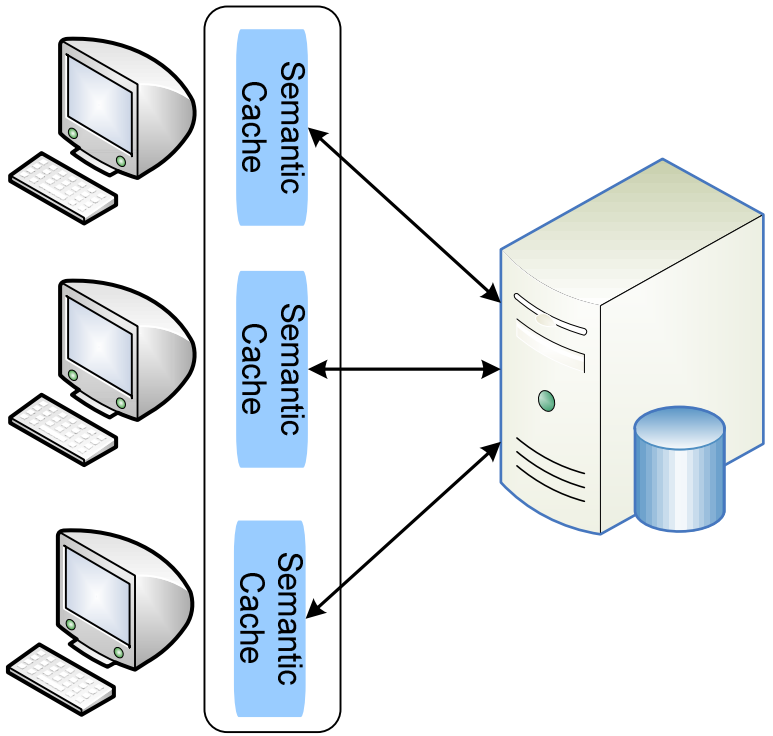
Background - Semantic Caching

- Cache entry
  - Query description
  - Result set
- Query rewriting
  - Probe
  - Remainder

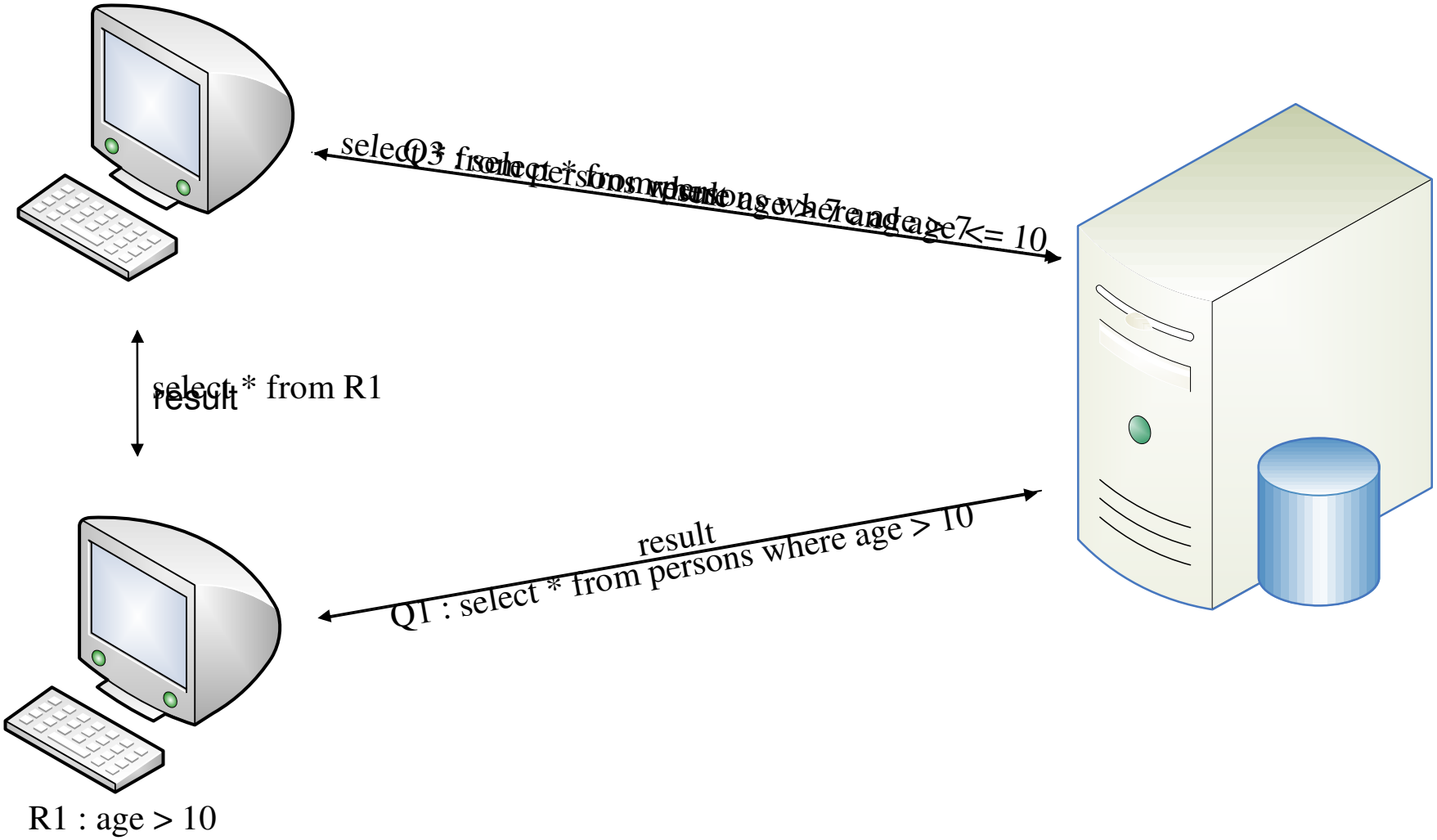


Cooperative Semantic Caching

- Share the local semantic caches between clients in a cooperative matter
- When answering a query, check if there are other clients that have useful semantic cache entries
- Why?
  - Reduce the load of the database
  - Decrease response time

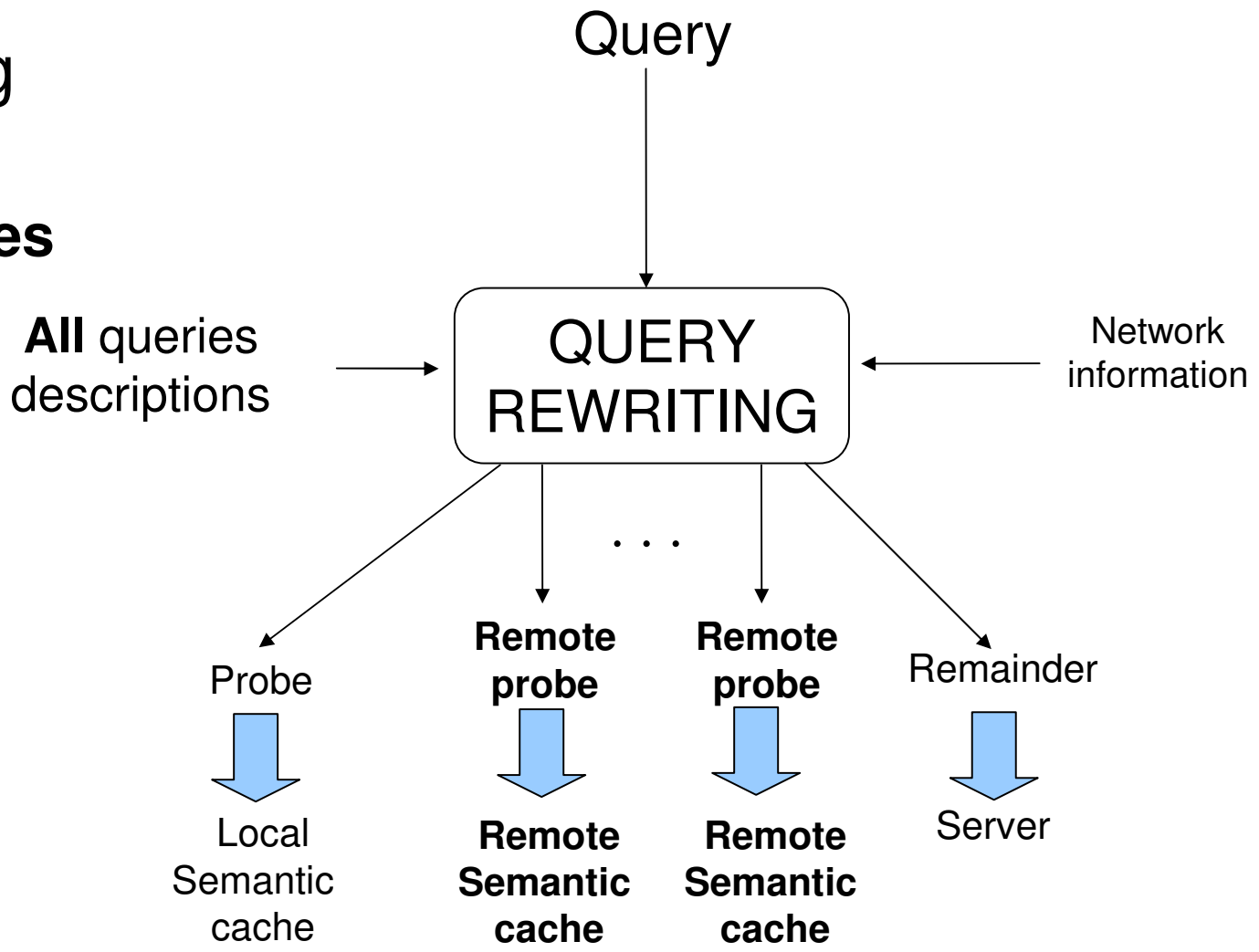


Cooperative Semantic Caching



Query Rewriting

- Query rewriting
  - Probe
  - **Remote probes**
  - Remainder



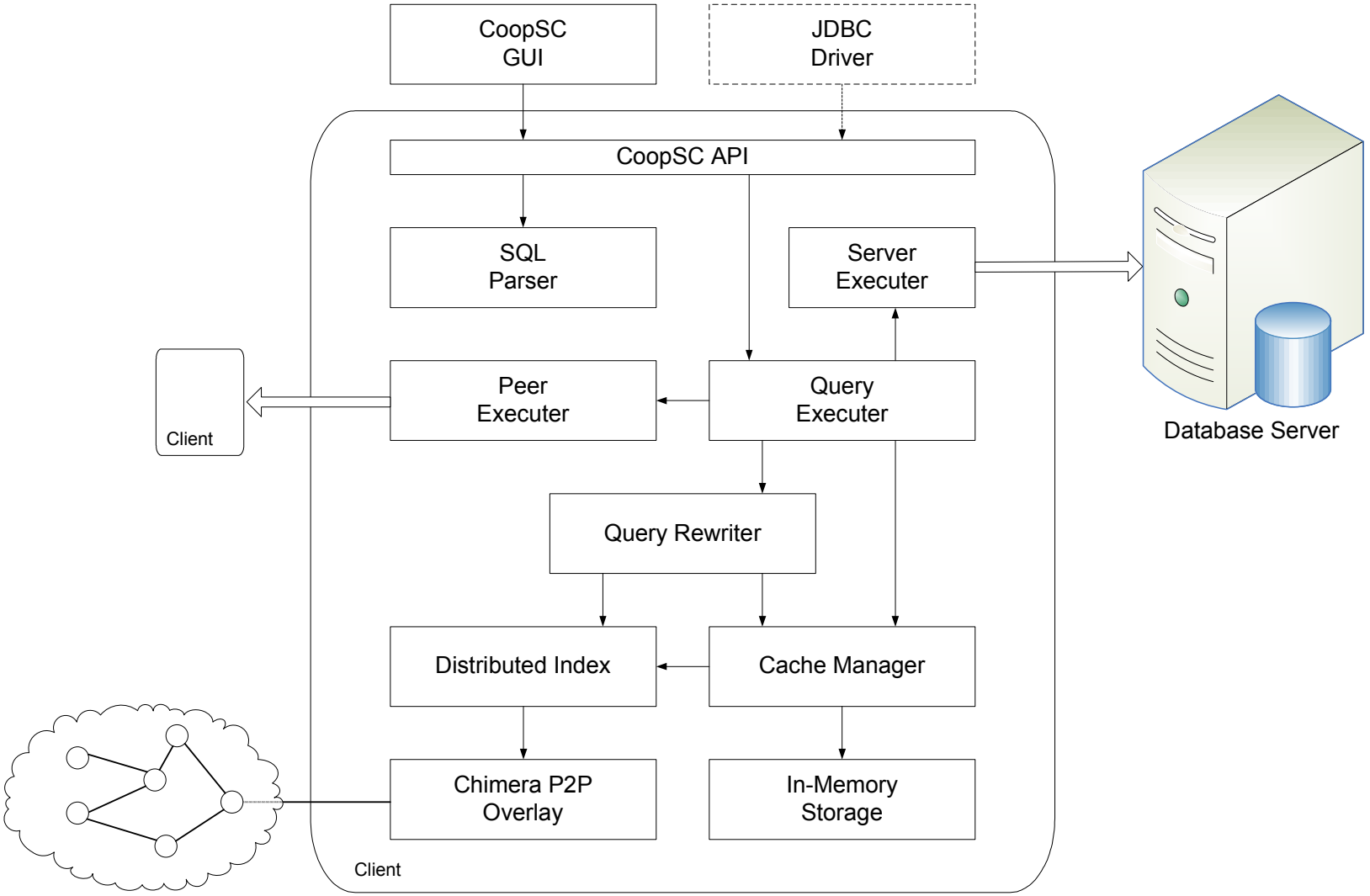
CoopSC

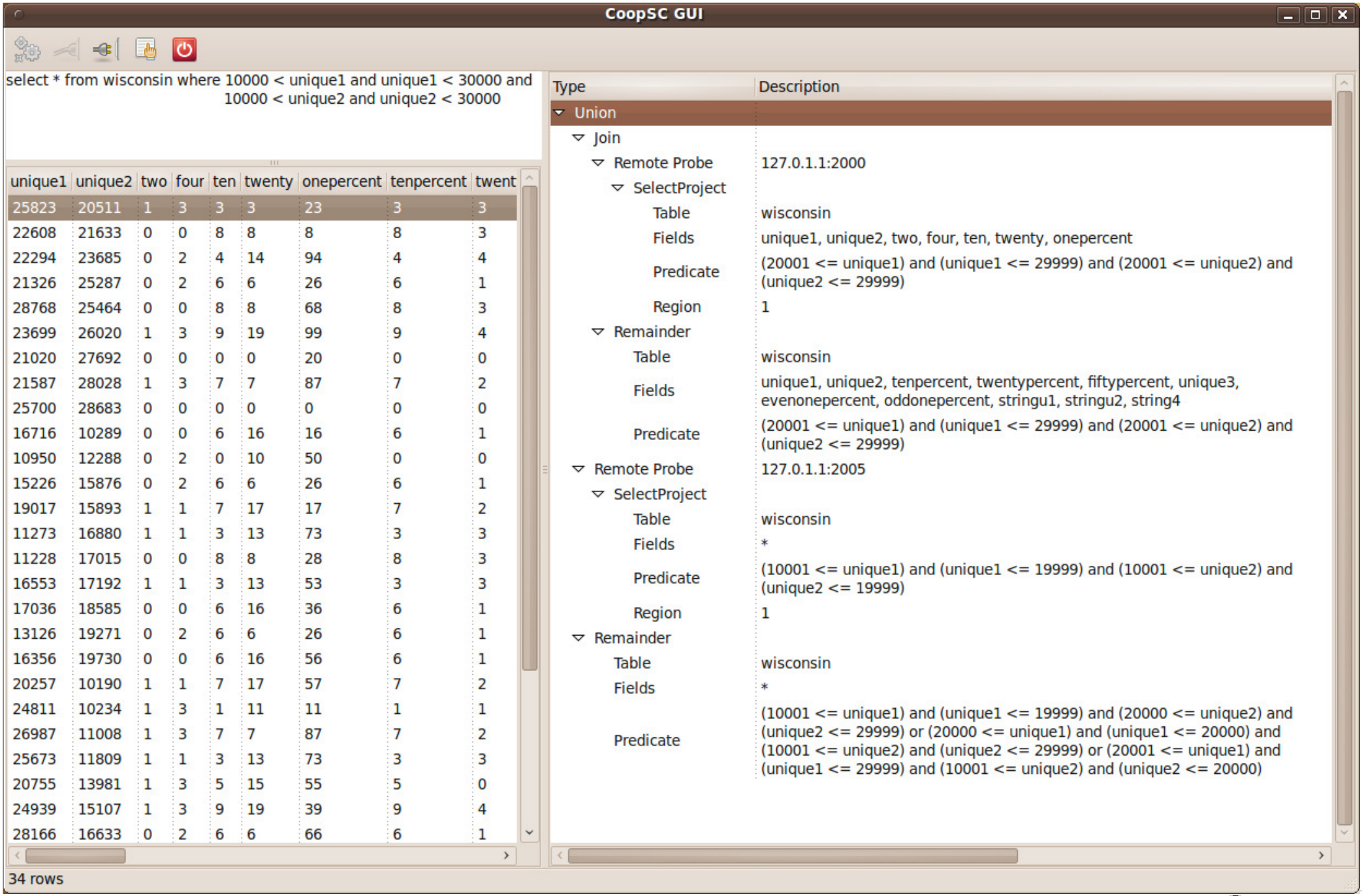
- **Coop**erative **S**emantic **C**aching
- Back-end : PostgreSQL
- Query Types
  - Selection (range predicates)
  - Projection
  - *select id, name, age from persons where 20 < age and age < 30*
- Cache organization
  - Semantic regions – stored locally by clients
  - Distributed index – built on top of a P2P overlay
- LRU (Least Recently Used) replacement policy

CoopSC

- Semantic regions
  - n-Tuples
  - Query description
    - Fields
    - Predicate
- Distributed Index
  - Indexes the semantic regions of all clients
  - Used for query rewriting
  - Built on top of Chimera P2P overlay

CoopSC - Architecture




CoopSC GUI
select * from wisconsin where 10000 < unique1 and unique1 < 30000 and
10000 < unique2 and unique2 < 30000
unique1 unique2 two four ten twenty onepercent tenpercent twent
25823 20511 1 3 3 3 23 3 3
22608 21633 0 0 8 8 8 8 3
22294 23685 0 2 4 14 94 4 4
21326 25287 0 2 6 6 26 6 1
28768 25464 0 0 8 8 68 8 3
23699 26020 1 3 9 19 99 9 4
21020 27692 0 0 0 0 20 0 0
21587 28028 1 3 7 7 87 7 2
25700 28683 0 0 0 0 0 0 0
16716 10289 0 0 6 16 16 6 1
10950 12288 0 2 0 10 50 0 0
15226 15876 0 2 6 6 26 6 1
19017 15893 1 1 7 17 17 7 2
11273 16880 1 1 3 13 73 3 3
11228 17015 0 0 8 8 28 8 3
16553 17192 1 1 3 13 53 3 3
17036 18585 0 0 6 16 36 6 1
13126 19271 0 2 6 6 26 6 1
16356 19730 0 0 6 16 56 6 1
20257 10190 1 1 7 17 57 7 2
24811 10234 1 3 1 11 11 1 1
26987 11008 1 3 7 7 87 7 2
25673 11809 1 1 3 13 73 3 3
20755 13981 1 3 5 15 55 5 0
24939 15107 1 3 9 19 39 9 4
28166 16633 0 2 6 6 66 6 1
34 rows
Type
Description
Union
Join
Remote Probe
127.0.1.1:2000
SelectProject
Table
wisconsin
Fields
unique1, unique2, two, four, ten, twenty, onepercent
Predicate
(20001 <= unique1) and (unique1 <= 29999) and (20001 <= unique2) and (unique2 <= 29999)
Region
1
Remainder
Table
wisconsin
Fields
unique1, unique2, tenpercent, twentypercent, fiftypercent, unique3, evenonepercent, oddonepercent, stringu1, stringu2, string4
Predicate
(20001 <= unique1) and (unique1 <= 29999) and (20001 <= unique2) and (unique2 <= 29999)
Remote Probe
127.0.1.1:2005
SelectProject
Table
wisconsin
Fields
*
Predicate
(10001 <= unique1) and (unique1 <= 19999) and (10001 <= unique2) and (unique2 <= 19999)
Region
1
Remainder
Table
wisconsin
Fields
*
Predicate
(10001 <= unique1) and (unique1 <= 19999) and (20000 <= unique2) and (unique2 <= 29999) or (20000 <= unique1) and (unique1 <= 20000) and (10001 <= unique2) and (unique2 <= 29999) or (20001 <= unique1) and (unique1 <= 29999) and (10001 <= unique2) and (unique2 <= 20000)

Conclusion

- P2P approach used for reducing the load of the database server
- CoopSC Architecture