Distributed caching for multiple databases

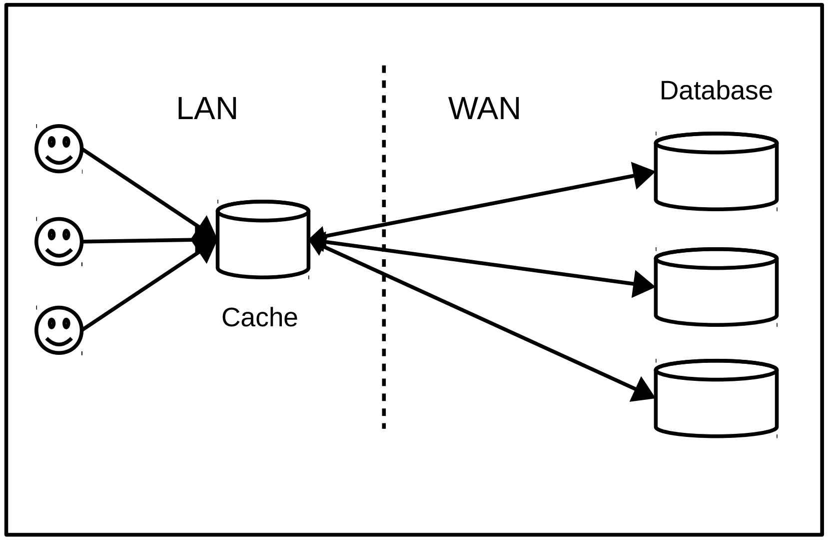
K. V. Santhilata,
Post Graduate Research Student,
Department of Informatics,
School of Natural and Mathematical Sciences,
King's College London, London, U.K

Road map

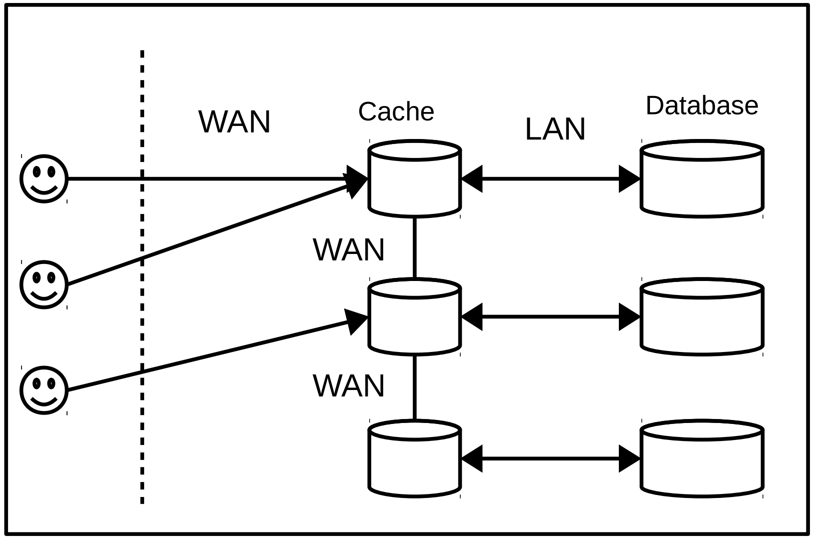
- Introduction
- Problem statement
- Solution
- Example
- Evaluation
- Summary & future work

Introduction - Data caching

Cache at client-side

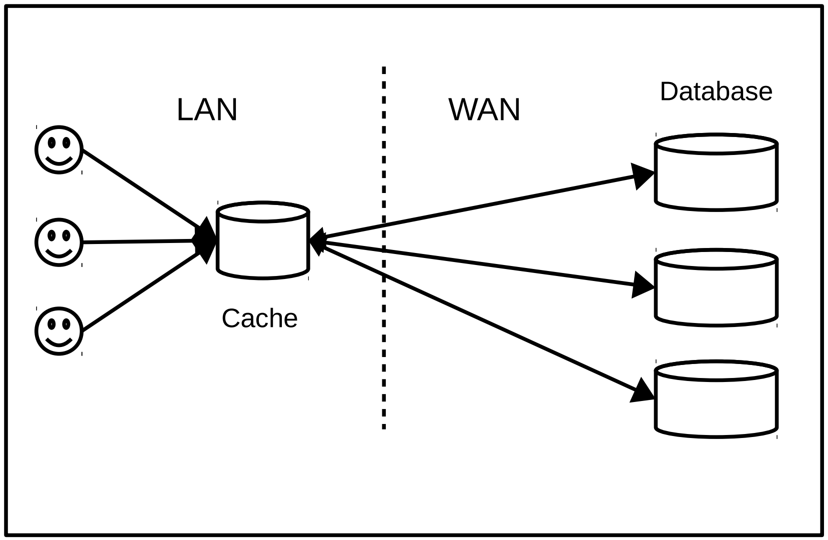


Cache at server-side

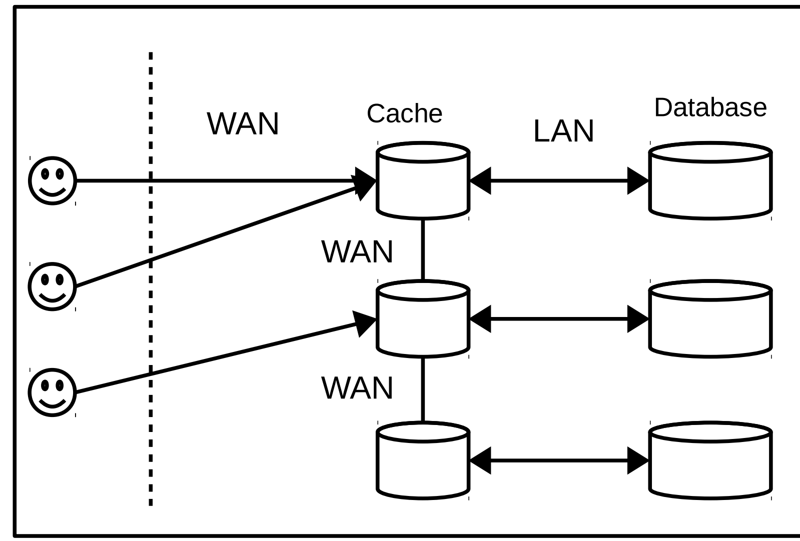


Introduction - Data caching

Cache at client-side



Cache at server-side

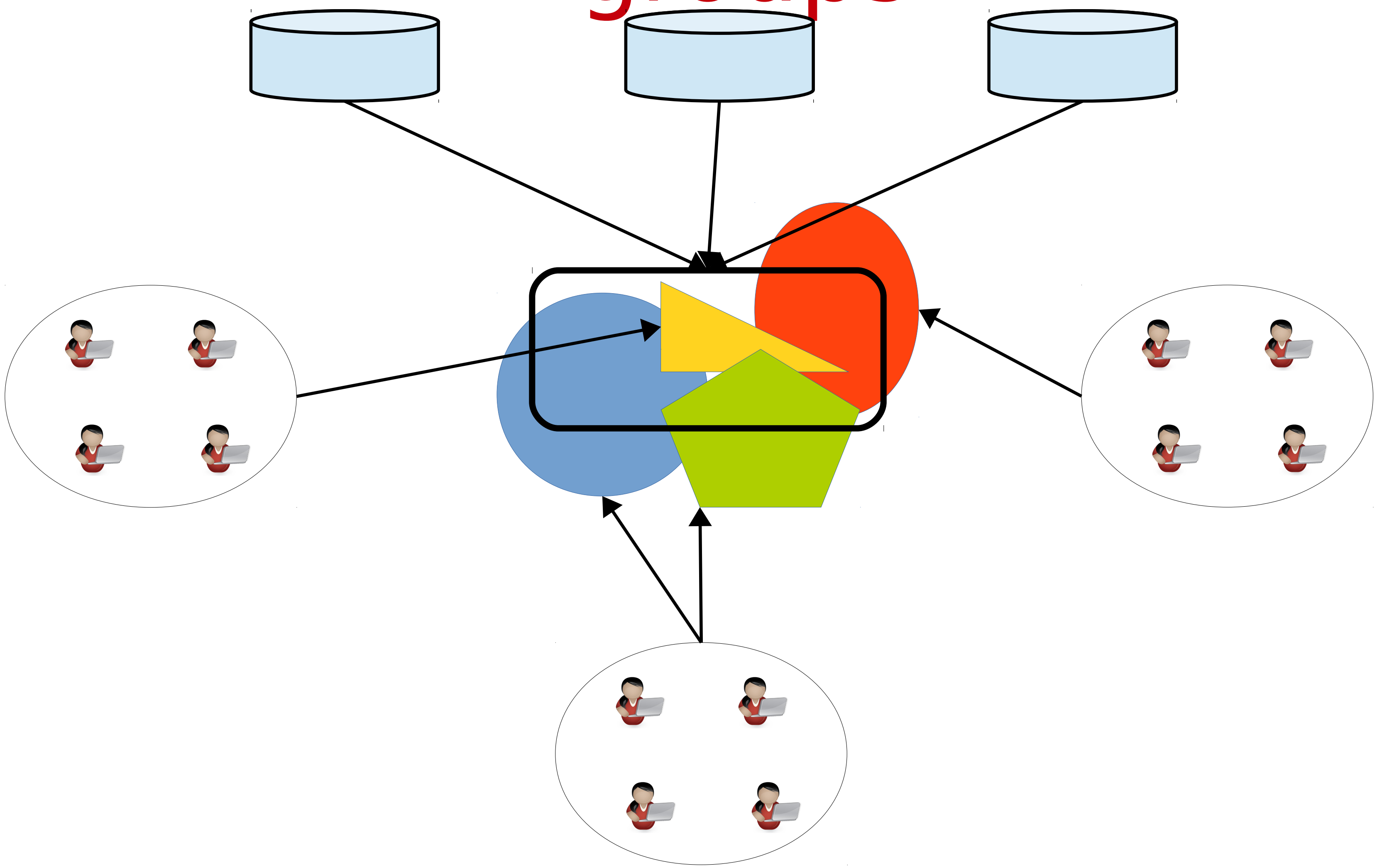


- Reduced data transfers
- Reduced response time
- Reduced resource utilization

What to cache?

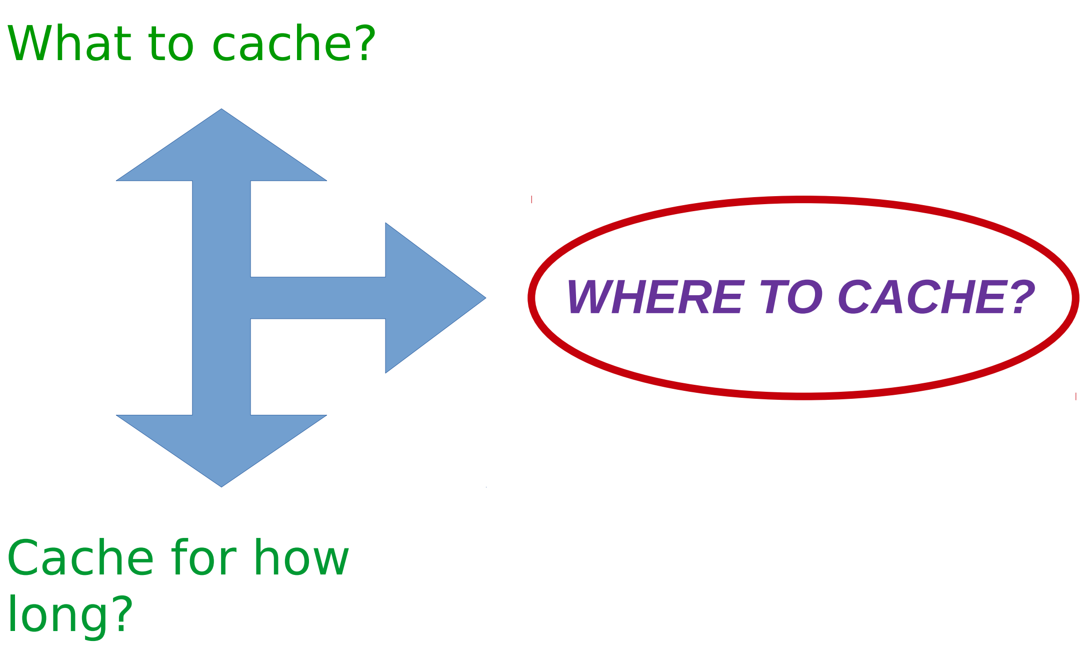
Cache for how long?

Caching needs among work groups

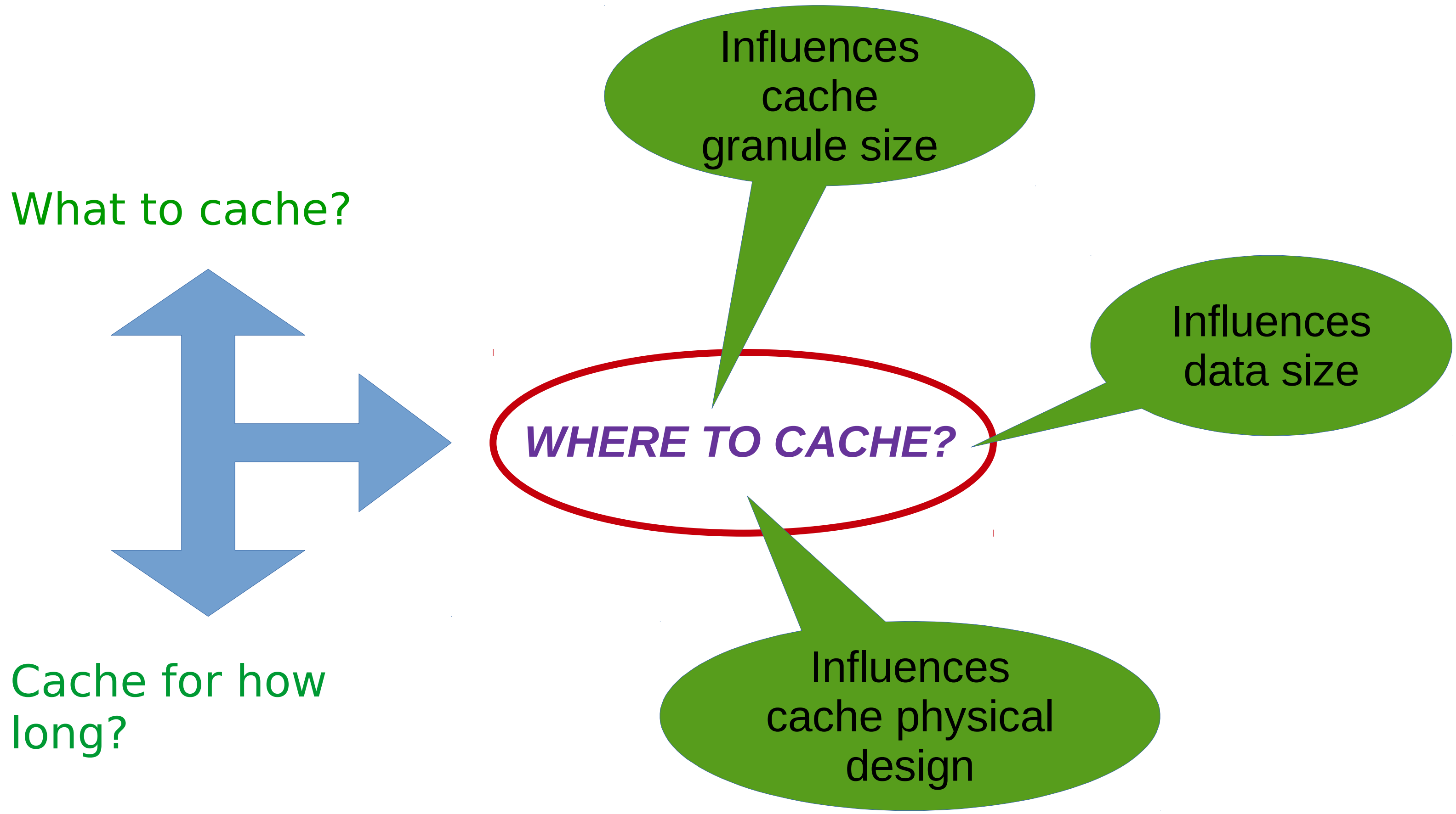


How to cache in the dynamic work flow environment among work groups while querying distributed databases?

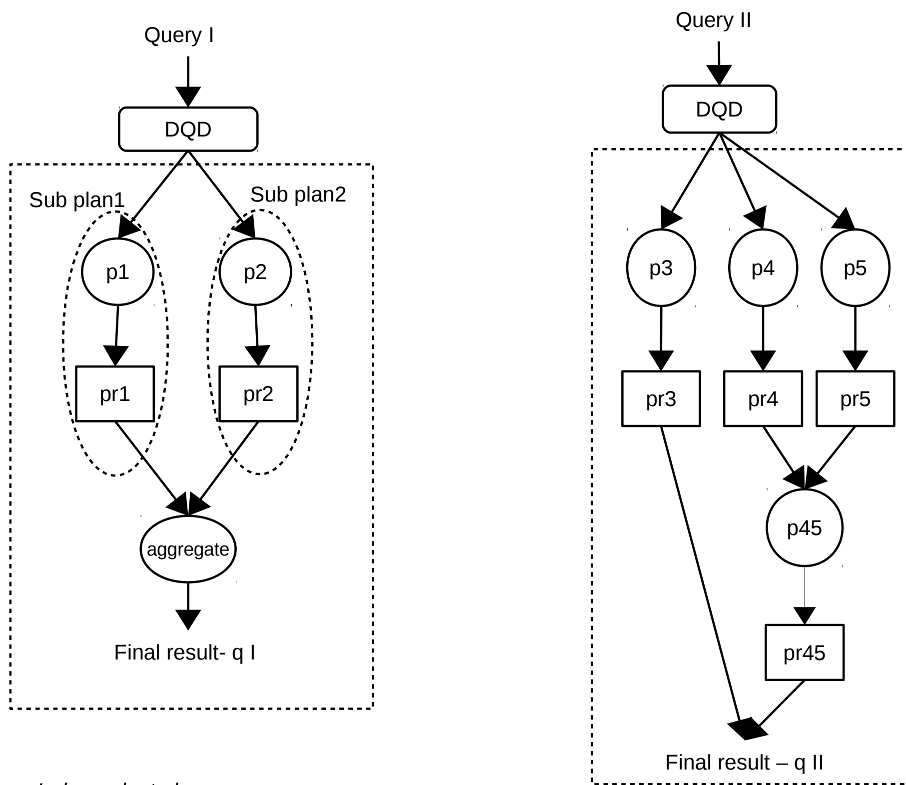
Solution: Adaptive mobile co-operative cache

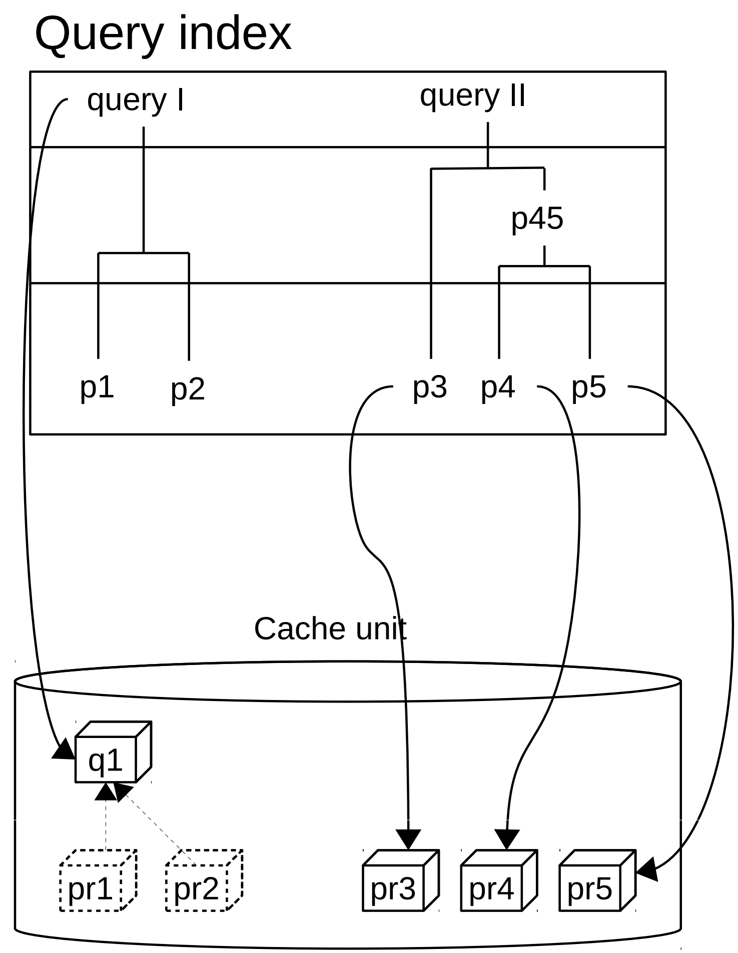


Solution: Adaptive mobile co-operative cache



Cache design - 1



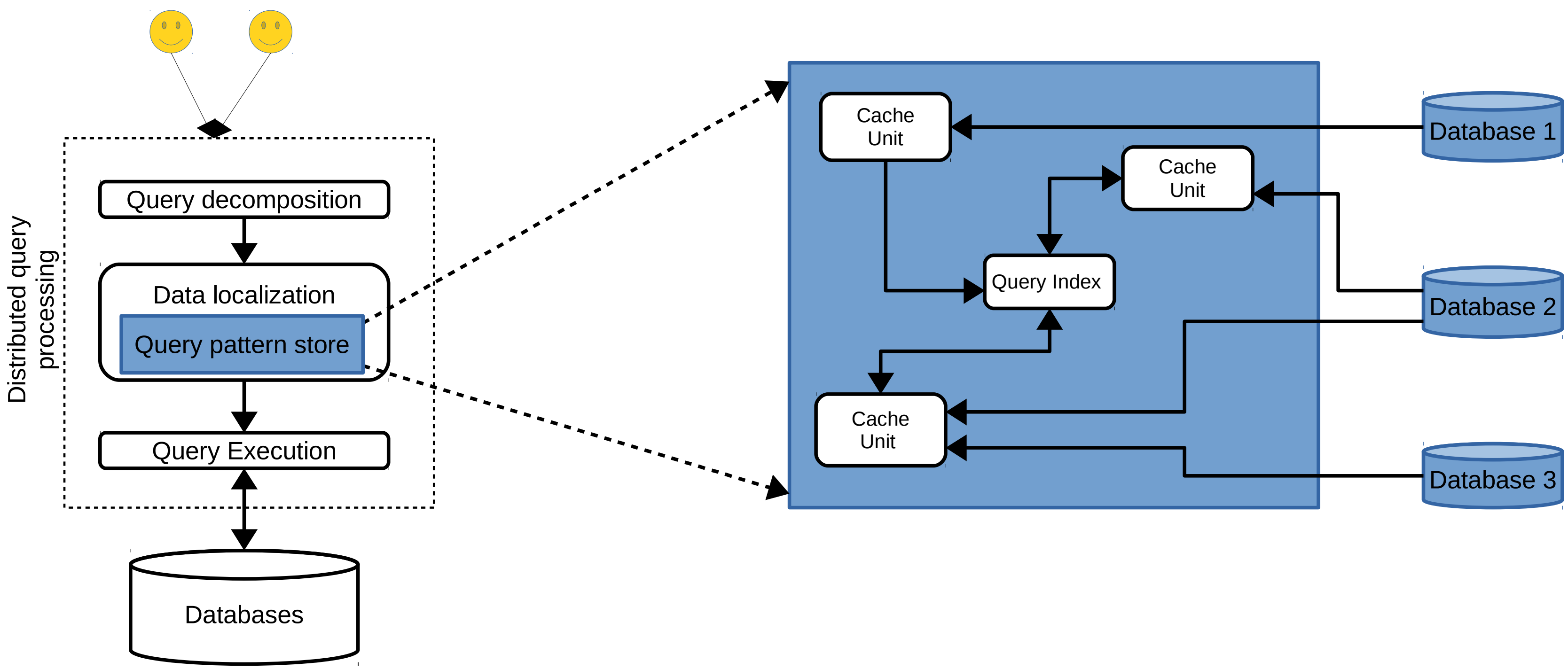


Independent plans:
Sub Plan1 : p1->pr1
Sub Plan2 : p2->pr2

Algorithm - cache granule creation

```
For each new query{
 -  Fragment query into sub queries
 -  Cache data as a smallest data object
 -  Cluster / de-cluster sub queries using association rules
 -  Store information in the knowledge pattern base for future use
}
```

Cache design - 2

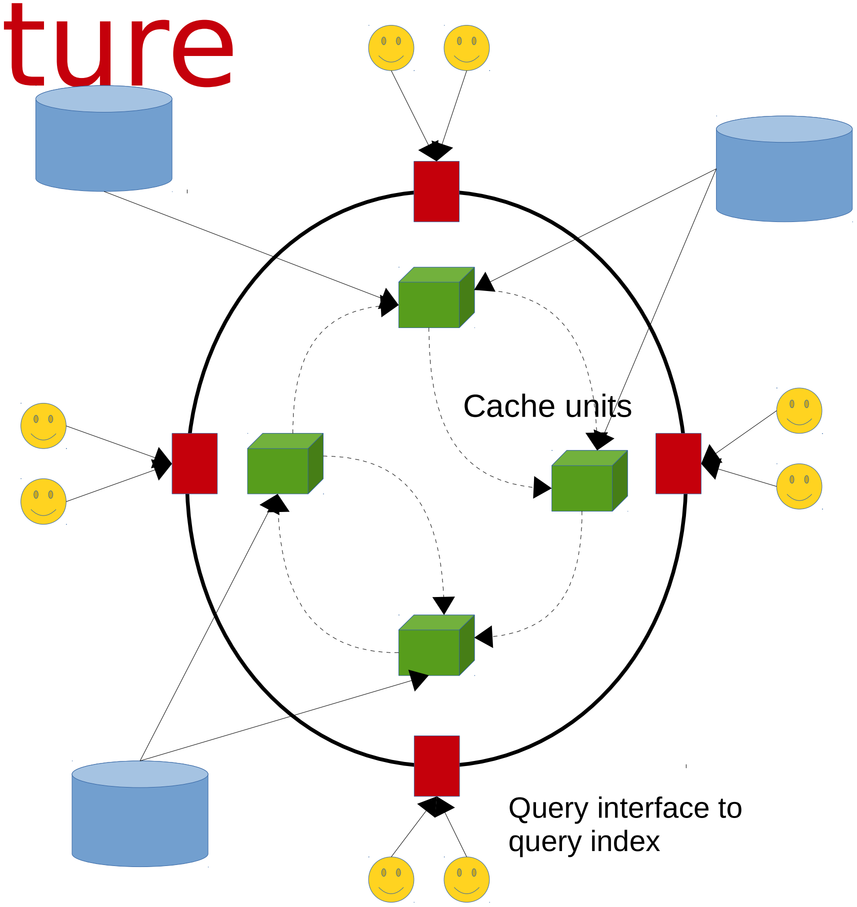


Algorithm - cache granule selection at each node

```
For each new query{
 -  Top-down search query index for the biggest granule matching
 -  Update query index for the frequency of updates and location of the query generation
 -  Estimate data size
 -  Cluster / de-cluster sub queries using geographical information
 -  Store information in the knowledge pattern base for future use
}
```

Mobile co-operative cache architecture

- Common query index
- Each cache unit stores portable data objects
- Cache refreshment is a heuristic based on the sub query frequency, time span and data size
- Assesses need for the data on location basis and performs cache data transfers




```
Algorithm: cache location selection
For a given time epoch{
   For each of the cache node{
    • Identify the data usage at the current cache node
    • Calculate and estimate the need for the data for other nodes (generate the popularity of the data needed)
    • Calculate data transfer costs
    • Store information in the knowledge pattern base for future use
   }
}
```

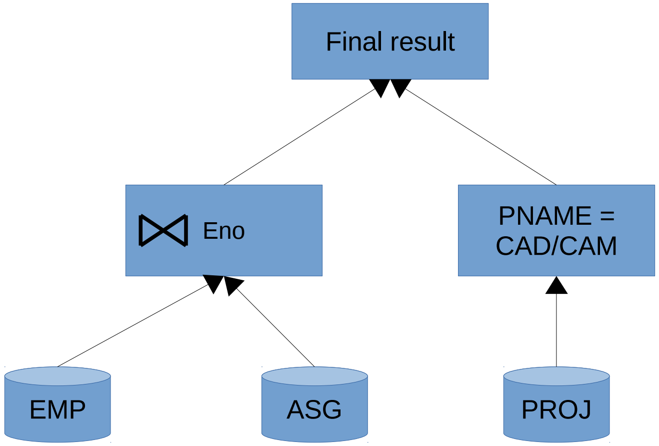
Example

Query trace (semantic query)

- Q1_location1: Names of employees working on CAD/CAM project
- Q2_location2: Names of employees working on CAD/CAM project at manager level
- Q3_location1: Find the names of employees who are managing a project

Query trace - SQL

- Q1_location1: **SELECT** *EMP.ENAME* **FROM** *EMP, ASG, PROJ* **WHERE** *EMP.ENO = ASG.ENO* **AND** *ASG.PNO=PROJ.PNO* **AND** *PNAME="CAD/CAM"*
- Q2_location2: **SELECT** *EMP.ENAME* **FROM** *EMP, ASG, PROJ* **WHERE** *EMP.ENO = ASG.ENO* **AND** *ASG.PNO=PROJ.PNO* **AND** *PNAME="CAD/CAM" AND ASG.RESP = "MANAGER"*
- Q3_location1: **SELECT** *ENAME* **FROM** *EMP,ASG* **WHERE** *EMP.ENO = ASG.ENO* **AND** *ASG.RESP = "MANAGER"*



Cost parameters & Evaluation

- Types of queries
  - Partial query hits from the cached data
  - Query predictability and data ordering patterns
- Query prioritization & balanced cost functions
  - Profiling for the resource utilization
  - Priority variation for shorter and real-time, time shared applications
  - Data transfer costs
- Response time
  - Estimation using entropy measurement
  - Queries that need single data source / multiple data sources
- Cache
  - Queries from varied local area groups and cache unit personalization
  - Cache training time

Pros and Cons

Advantages

- Allows query indexing for faster data object location
- Query indexing proves to be ideal with slow changing work patterns and data loads
- Handles nested queries easily as well as joins from multiple databases
- Improved cache hit ratio even with random queries
- Sub queries identify the most frequently needed data object, hence unused data fragment in the cache can be easily evicted

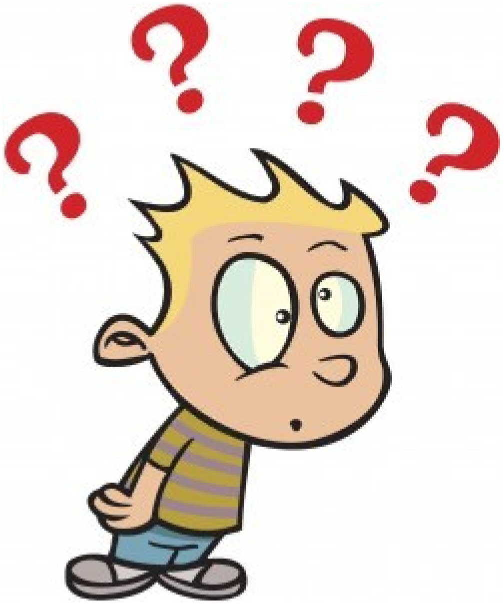
Efficiency is affected by

- Dependency on the query optimizer's efficiency for initial queries
- Complexity of the sub query generation / distribution algorithm

Summary & Future work

Picture copied from: https://markarmstrongillustration.files.wordpress.com/2010/07/eggbaskets.jpg

- Cached query mobility based on restrictions on the data size of the cache unit
- Query approximation



Comparison between random queries and 30% overlapped sub queries

Response time