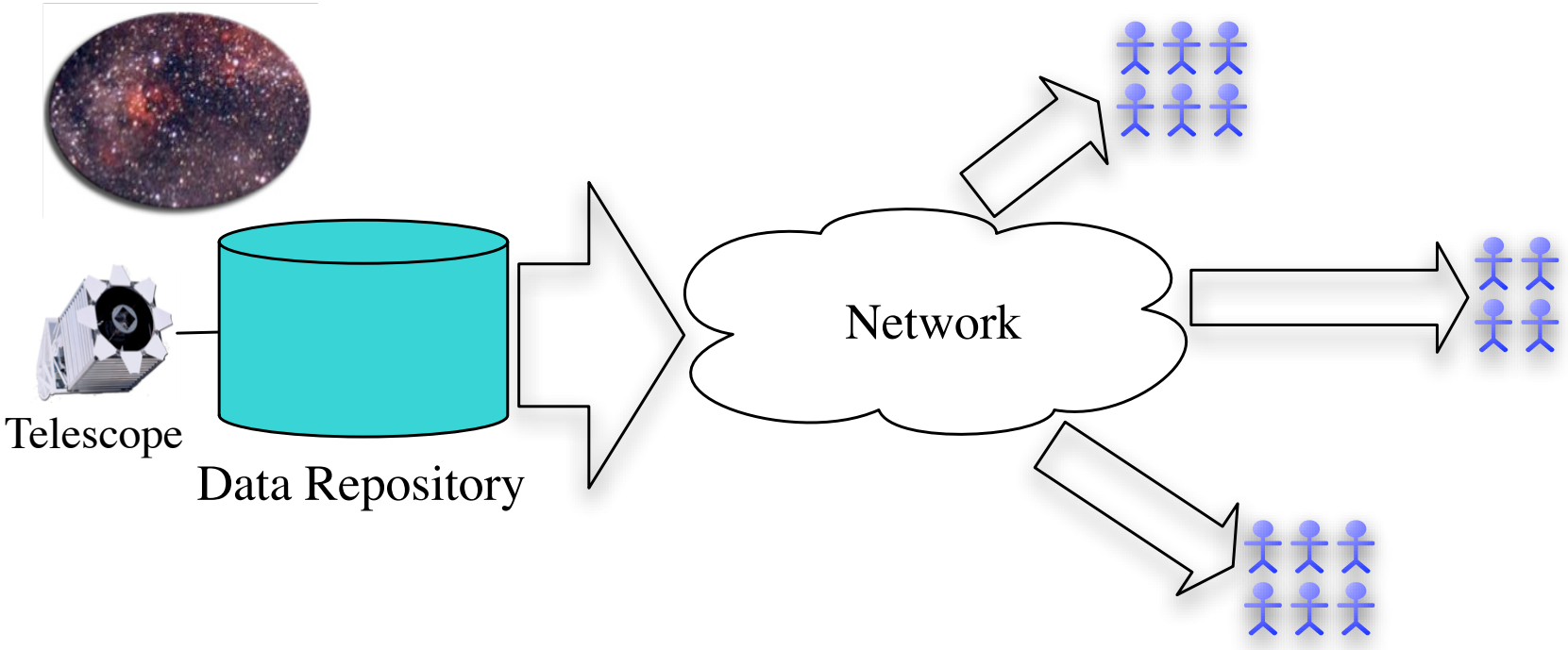
Caching for Data Intensive Scientific Repositories

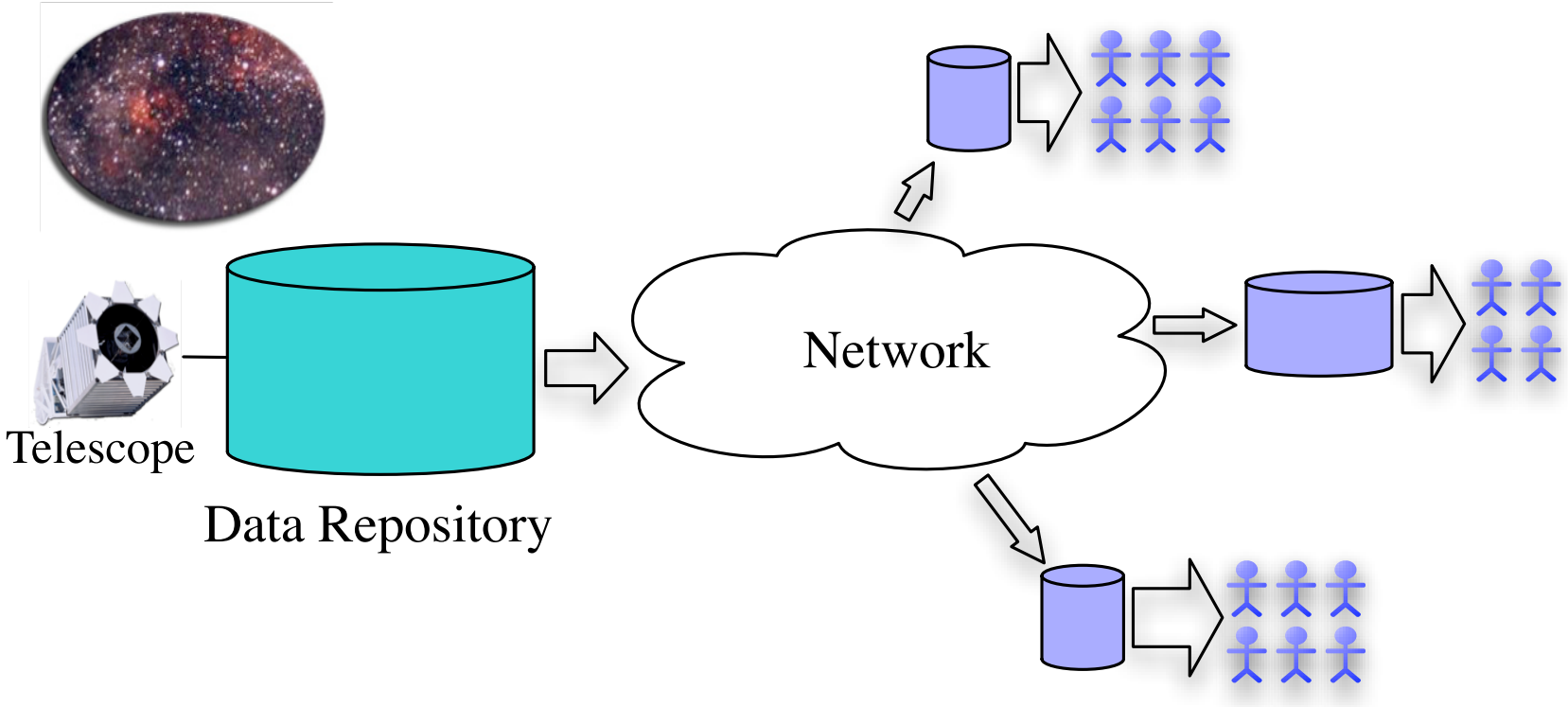
Ani Thakar, Dan Wang, Tanu Malik, Philip Little, Amitabh Chaudhary

Scientific repositories can have a large “network footprint”



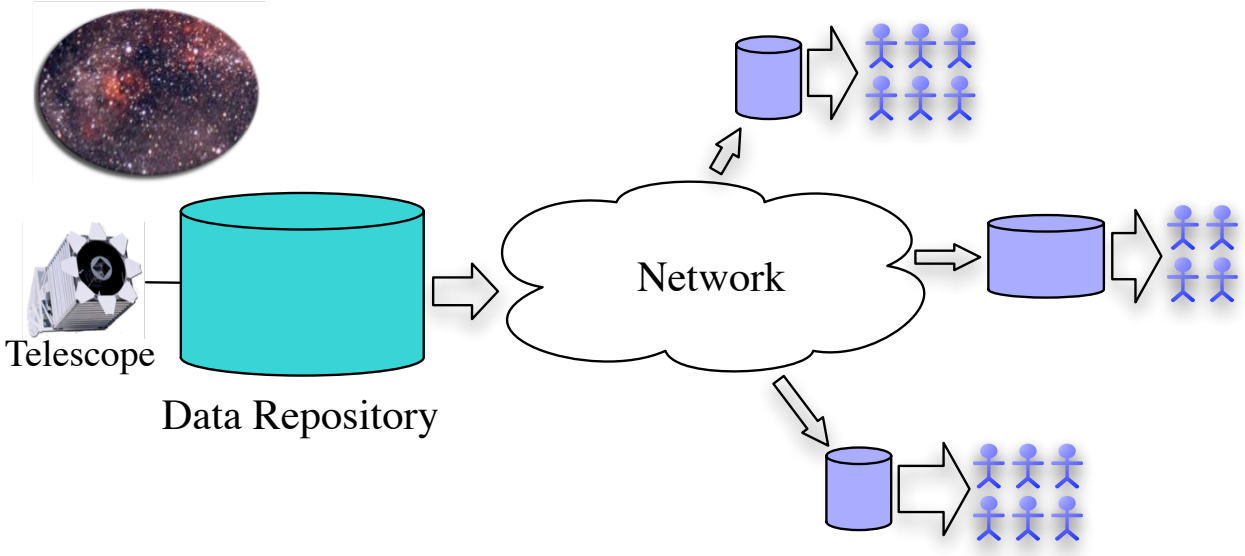
Pan-STARRS is expected to service over 10 TB of query results each day.
LSST will be a 150 times the size of Pan-STARRS.

Well-designed proxy caches can help reduce the network footprint



In simulations on SDSS (static data), traffic reduced to one-fifth.

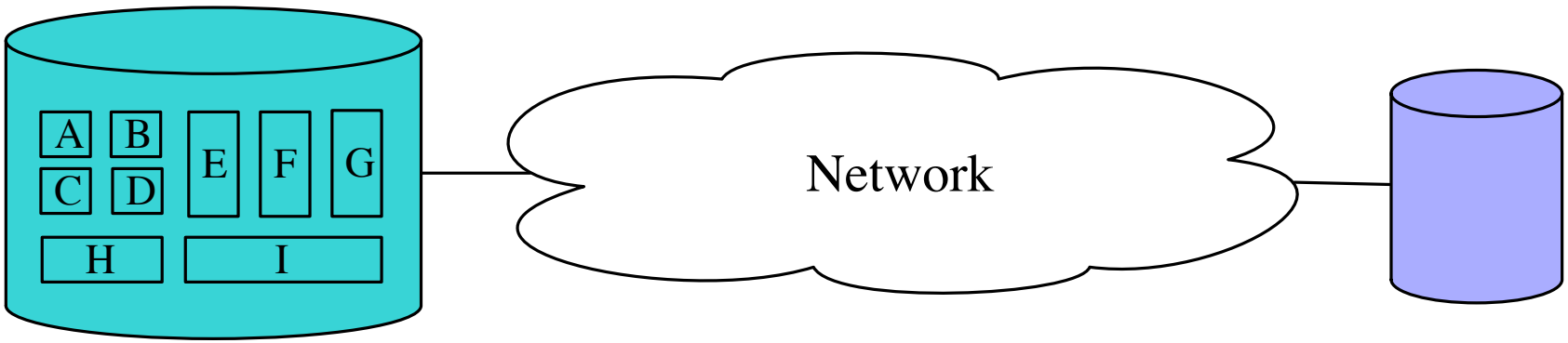
Well-designed proxy caches are hard to design



Three challenges —

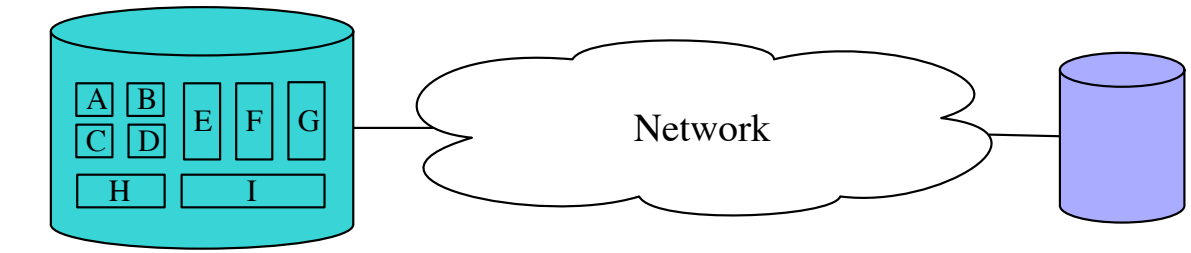
- How do we adaptively choose the best objects to cache?
- How do we process queries on transient objects?
- How do we move large data objects?

Cache objects have varying sizes and varying load costs



Objects can be relations, columns, horizontal partitions, vertical partitions, etc.

Caching decisions are not limited to loading and evicting objects



U_1
INSERT INTO A
VALUES (2.3, 30, ...)
VALUES (4.5, 25, ...)

U_2
UPDATE G
SET G.z = G.z+3.34
WHERE G.z ≤ 9.6

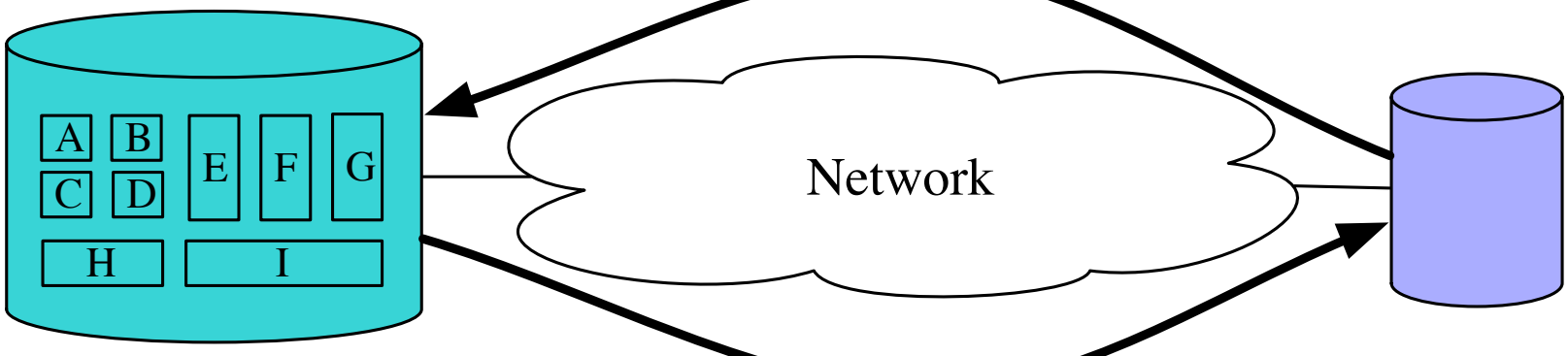
Q_1
SELECT MAX(A.x)
FROM A, B
WHERE A.x = B.y

Q_2
SELECT G.z
FROM A, G
WHERE A.x G.z

Three types of data communication —

- Query shipping
- Object loading
- Update shipping

Query shipping is for answering queries without using the cache contents



Q_1
SELECT MAX(A.x)
FROM A, B
WHERE A.x = B.y

Q_2
SELECT G.z
FROM A, G
WHERE A.x G.z

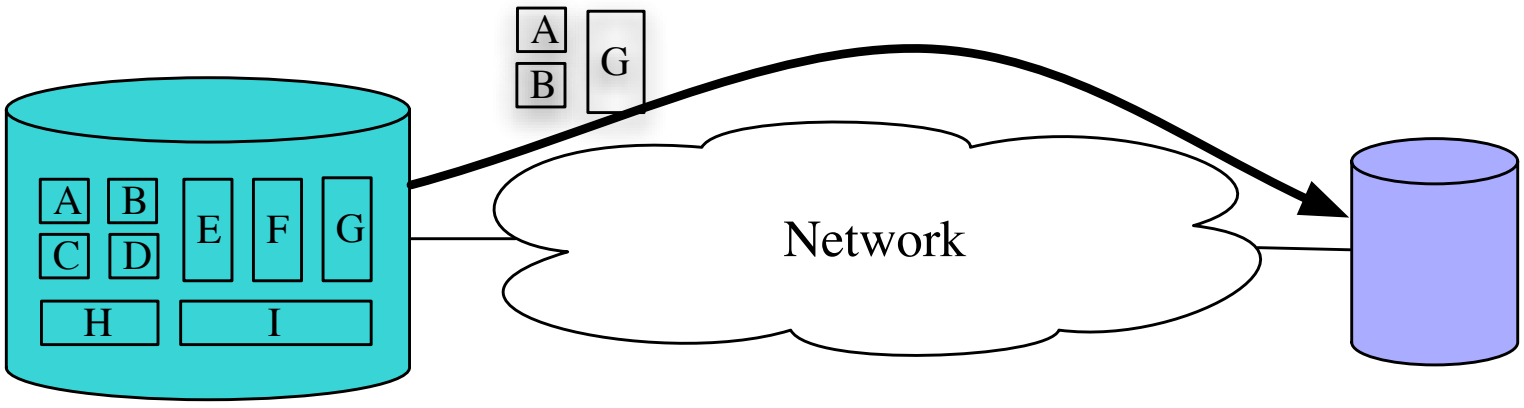
U_1
INSERT INTO A
VALUES (2.3, 30, ...)
VALUES (4.5, 25, ...)

Q_1 Result
4.5

Q_2 Result
2.3, 4.5, 7.9, 2.1, ...

U_2
UPDATE G
SET G.z = G.z+3.34
WHERE G.z $\leq$ 9.6

Loading is for moving frequently accessed objects



U_1

INSERT INTO A
VALUES (2.3, 30, ...)
VALUES (4.5, 25, ...)

U_2

UPDATE G
SET G.z = G.z+3.34
WHERE G.z ≤ 9.6

Q_1

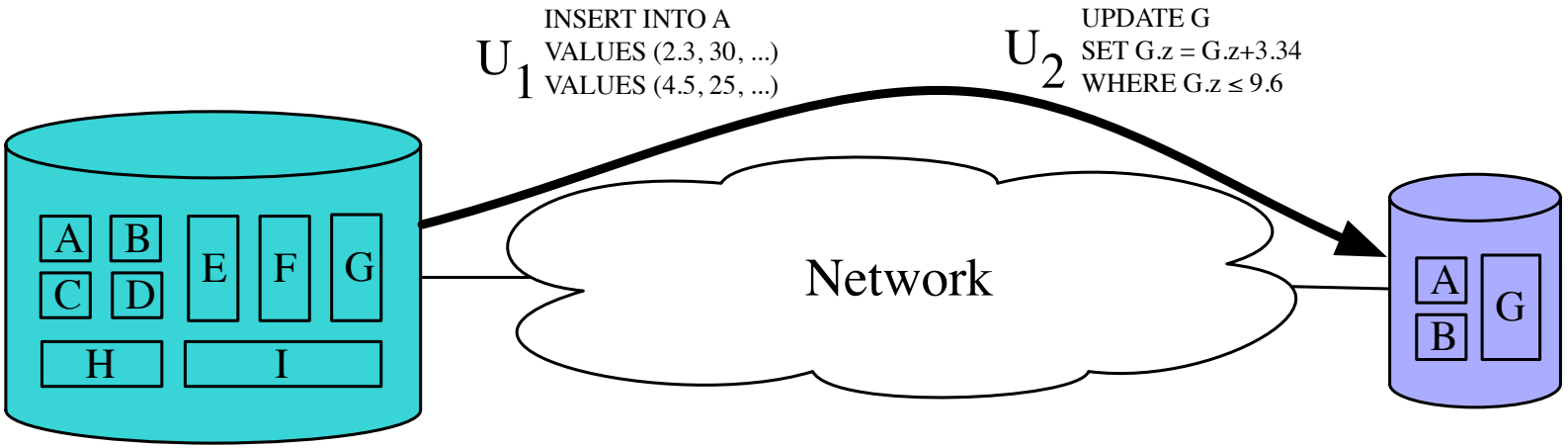
SELECT MAX(A.x)
FROM A, B
WHERE A.x = B.y

Q_2

SELECT G.z
FROM A, G
WHERE A.x ≤ G.z

This may require evicting other objects from the cache.

Update shipping is for keeping objects up-to-date



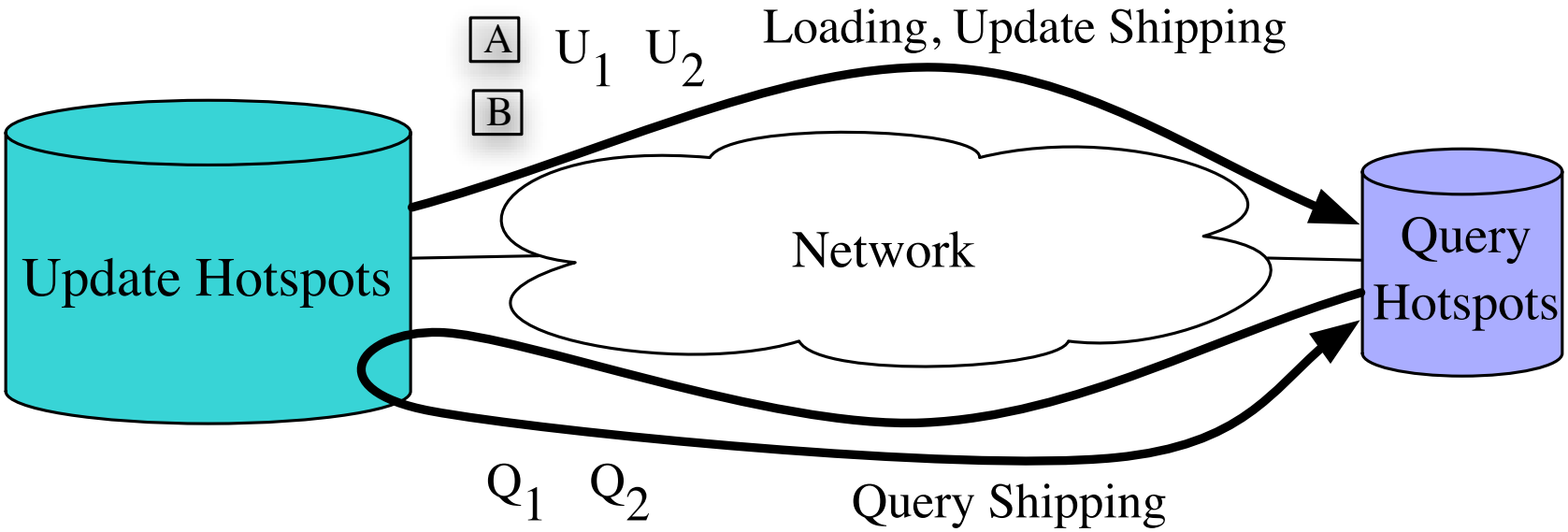
Q_1

```
SELECT MAX(A.x)
FROM A, B
WHERE A.x = B.y
```

Q_2

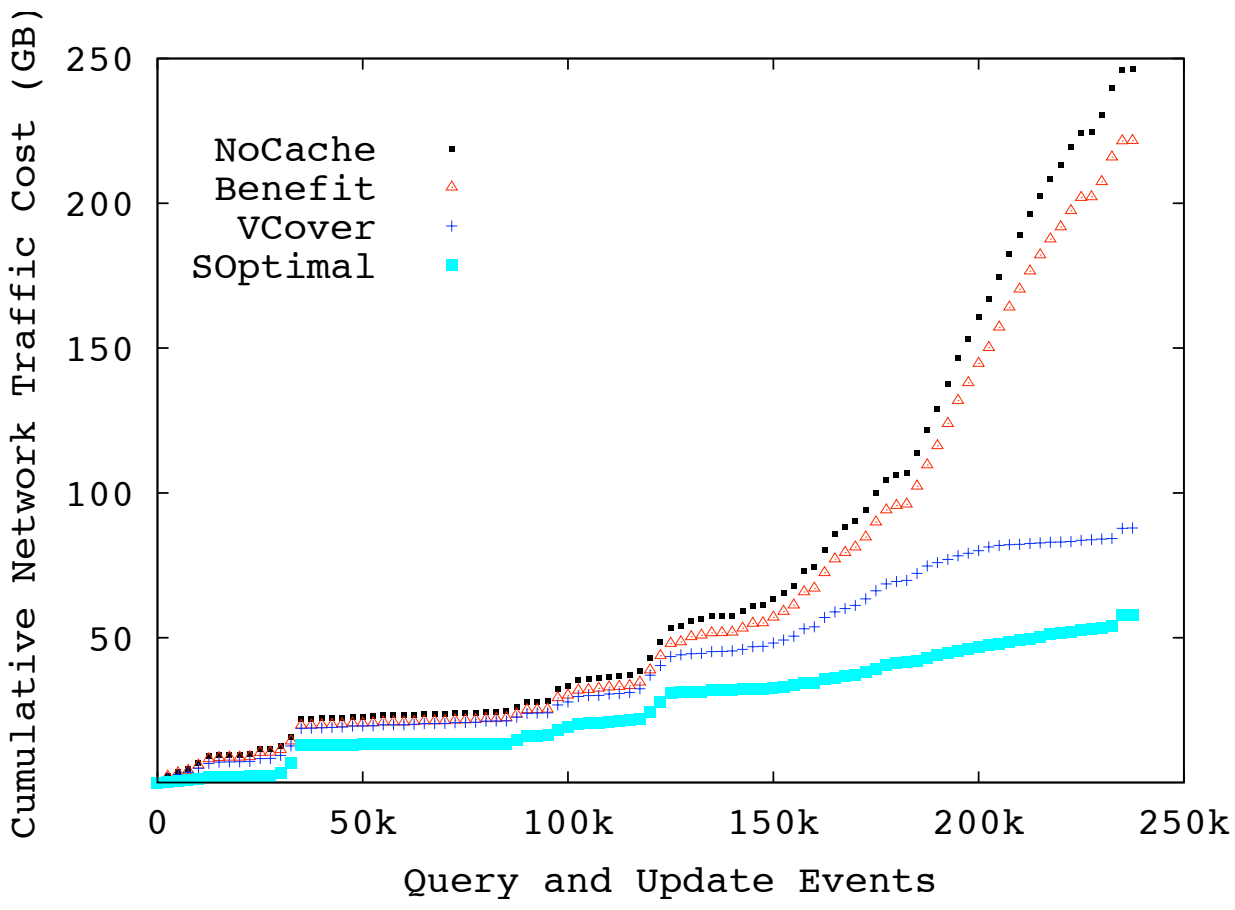
```
SELECT G.z
FROM A, G
WHERE A.x ≤ G.z
```

The objective is to keep the heavily queried objects in cache, and the heavily updated objects out of it — adaptively



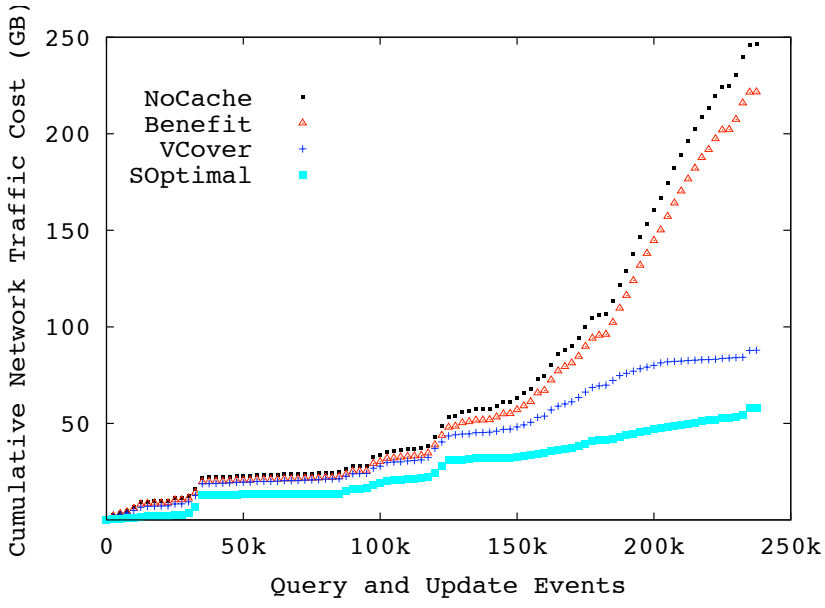
The interdependencies between objects makes this even harder.

Algorithm Benefit learns from the past window, but is hard to tune



It greedily loads objects by the benefit of keeping them in cache.

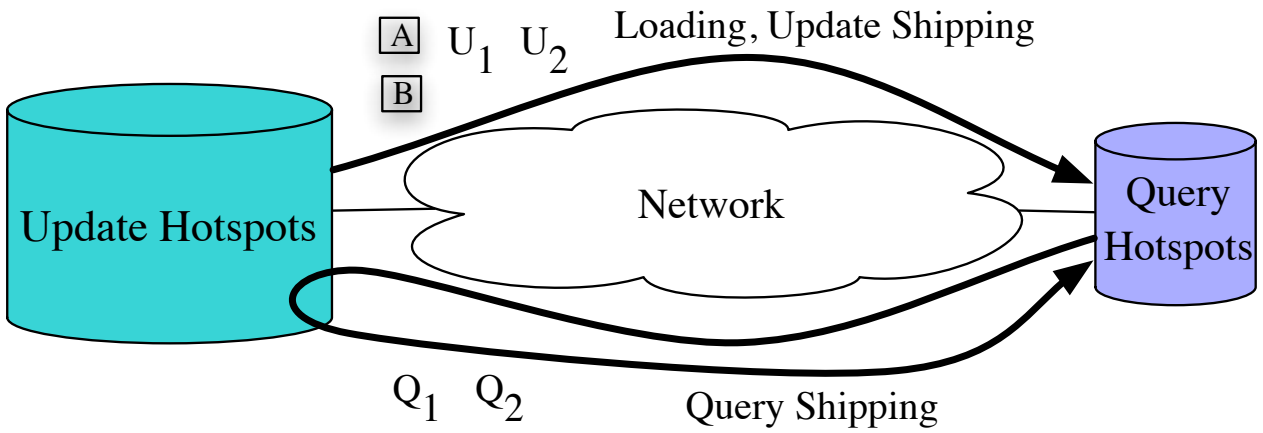
Algorithm VCover is conservative but performs close to the offline (static) optimal



Characteristics —

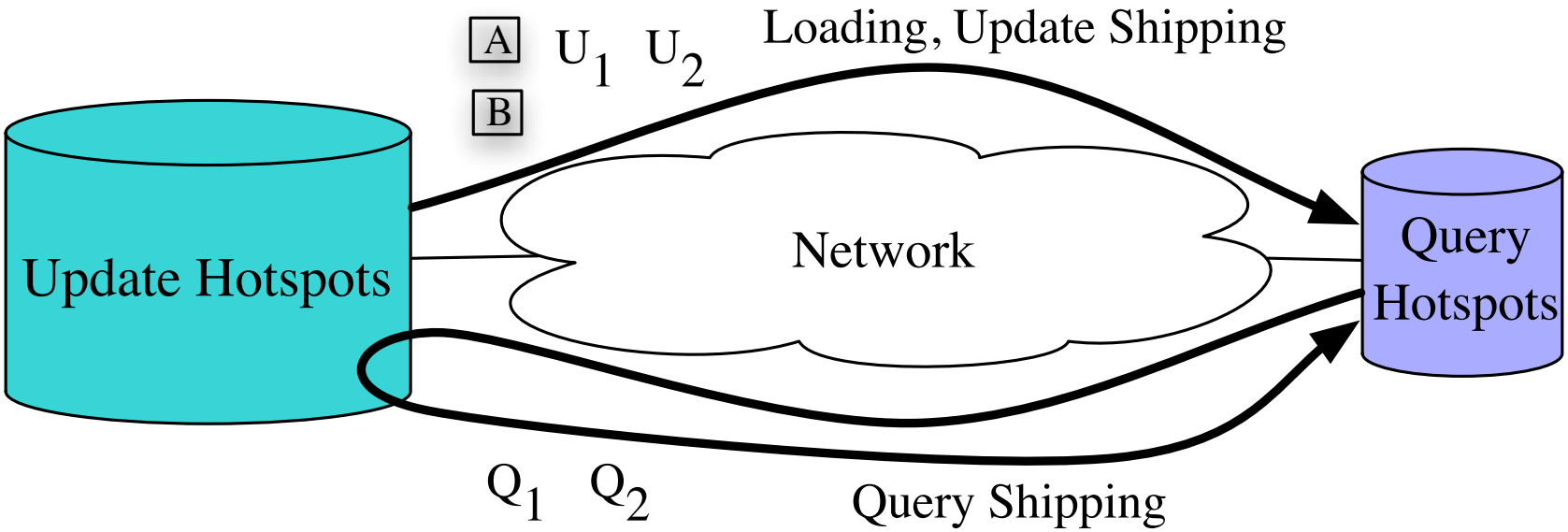
- It is based on online algorithms for caching.
- It incorporates a rent-versus-buy approach.
- It captures query-update interactions in a bi-partite graph (the minimum weighted vertex cover of which is the optimal solution).

Several open questions remain in creating an effective database cache



- Can we reduce the size of VCover data structures?
- Are there better caching algorithms?
- What is the best granularity for a data object?
- Should we be caching query results rather than data objects?
- How do we re-write queries for transient data objects?
- Can we use indices on transient data objects?

In summary, clever algorithms can help build effective caching solutions for data intensive repositories, but much remains to be done



Questions?