CASCADE: AN ATTACK-RESISTANT DHT WITH MINIMAL HARD STATE

Alexander Mohr

Mayank Mishra

State University of New York at Stony Brook

Motivation

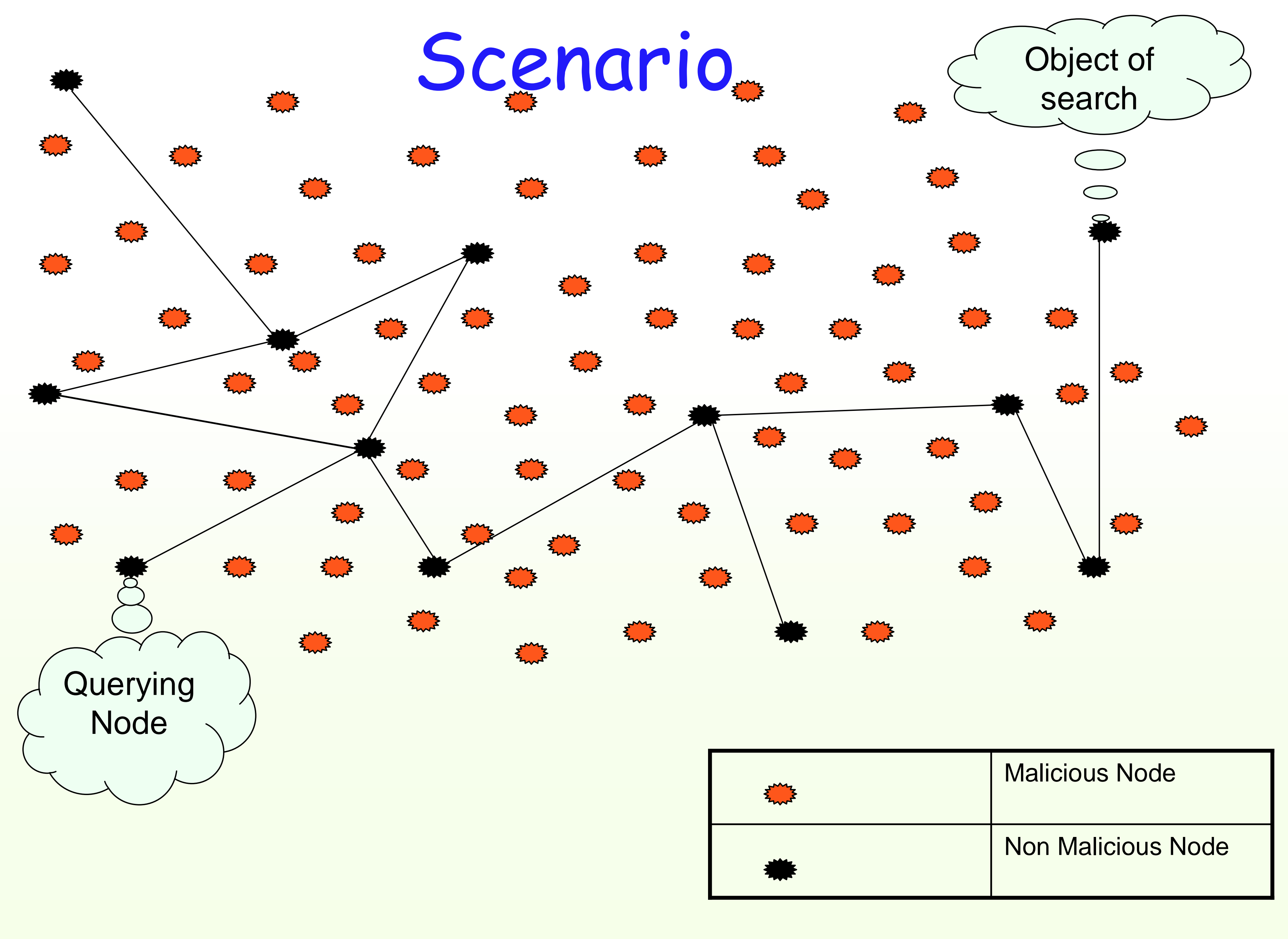
- Many P2P networks are not designed with attack-resistance in mind (Gnutella, Shareaza, eDonkey2k, Chord, CAN, Pastry, etc).
- Those that are attack-resistant generally are not as efficient (Freenet, RON, etc).

Let's try for both in a sloppy DHT!

Goals

- Guarantee that a resource in the network can be located (even if 90-95% peers are malicious).
- Make searches efficient with extensive caching.
- Empower users to have control over their searches.

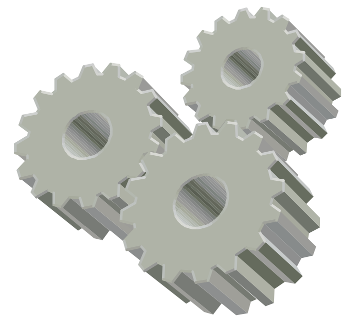
Scenario



Threat Model

- Underlying network is well-behaved.
- Nodes can be malicious or non-malicious.
- Malicious peers are Byzantine.
  - Co-ordinate amongst themselves.
  - May delay communication between non-malicious peers.

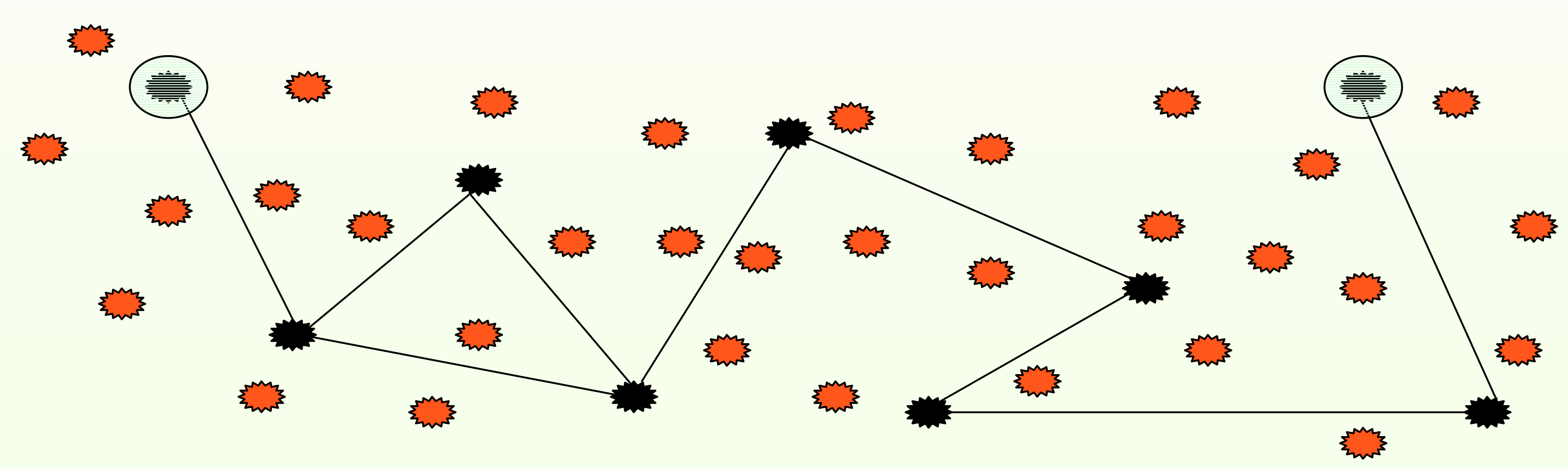
System Description



- Each node stores :
  - The keys that the node itself has inserted into the DHT (its "published keys").
  - A subset of alive peers (its "neighbors").
- When queried for a key, a node :
  - Consults its list of published keys,
  - Responds with the associated value if it was present,
  - Returns its list of neighbors.
- Searching the network is an iterative breadth-first search.

Claim

If there exists any non-malicious path from a query originator to a peer publishing the search key, the search will eventually succeed!



Claim

If there exists any non-malicious path from a query originator to a peer publishing the search key, the search will eventually succeed!

But, we’ve said nothing about whether such a path is likely to exist!

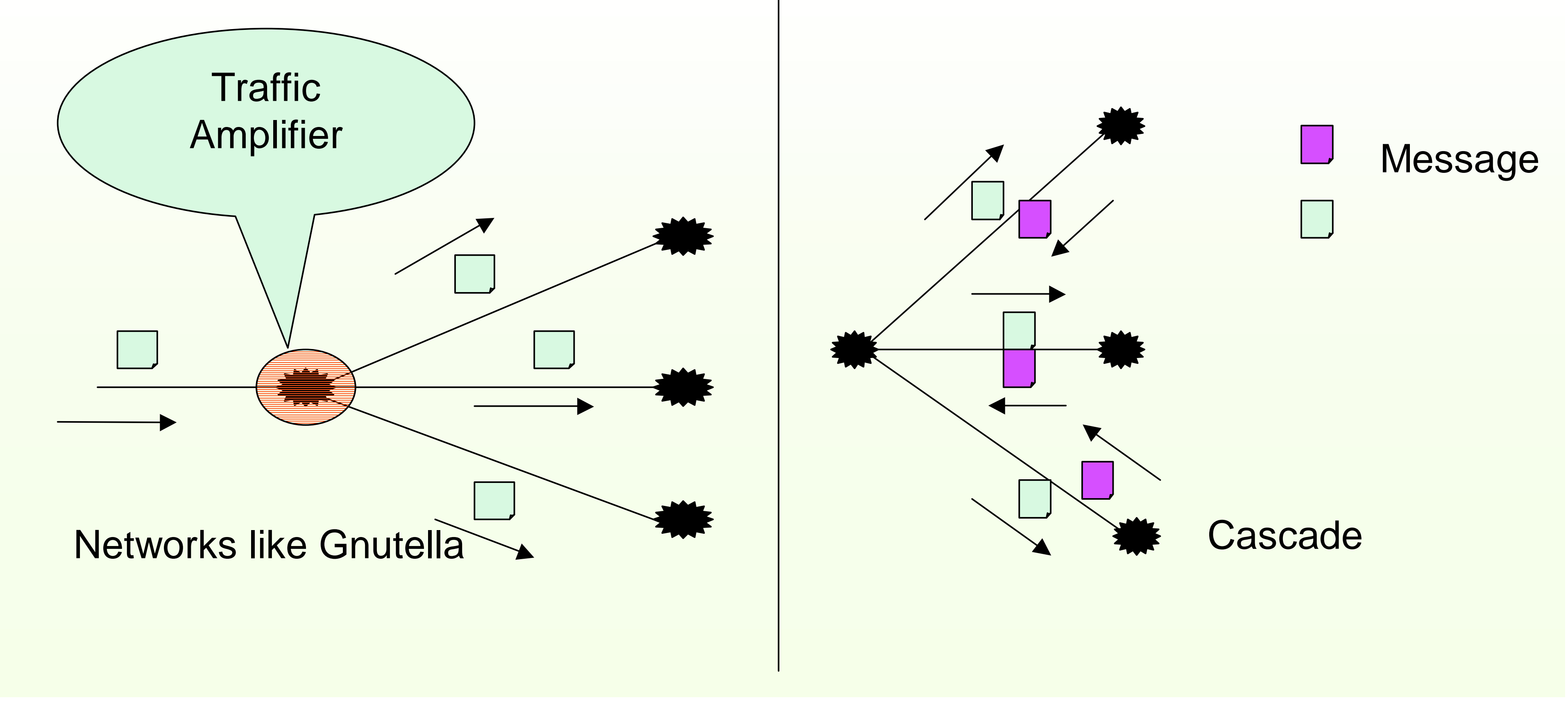
Open Question

- Can we guarantee that a non-malicious path will exist?

- *Maybe*: we're not yet sure how feasible it is.
  - Secure Routing [Castro *et al.*, 2002]
  - When choosing a new neighbor:
    - Flood the network to obtain a list of all peers.
    - Pick one at random.

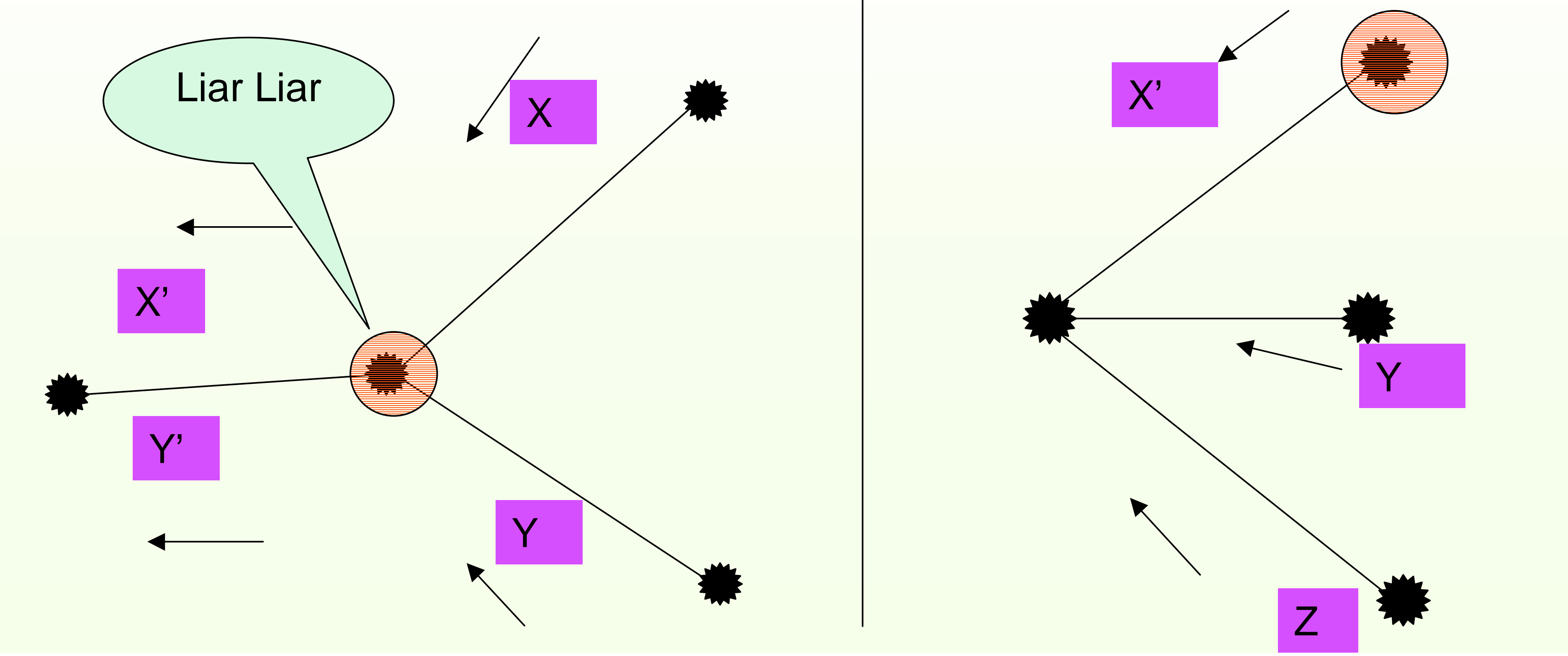
Traffic Amplification Attacks

- Iterative search prevents traffic amplification.
- More effort to search, but that might be good:



Man-in-the-Middle Attacks

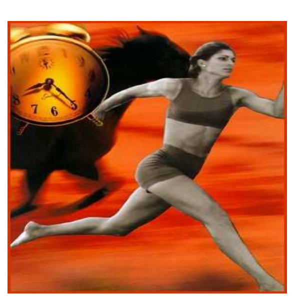
- There is no man in the middle.
  - Don't have to trust what others say on someone else's behalf.



State Exhaustion Attacks

- All per-query state is located on the querying node itself.
- No per-query state is maintained by the network.

Caching and Performance



- Goal #2: Efficient search.
- Add passive caching:
  - Known-peers cache.
  - Results cache.
  - Query cache.

Caches are hints and are not required for correct operation!!

Known Peers Cache

- Whenever you discover a peer, store:
  - Whom they were.
  - When you saw them.

- Save this cache between program runs to bootstrap.
- With directed searches, get there faster.

Results Cache

- Store the results of your own searches:
  - What you found.
  - Where it was.
  - When you saw it.

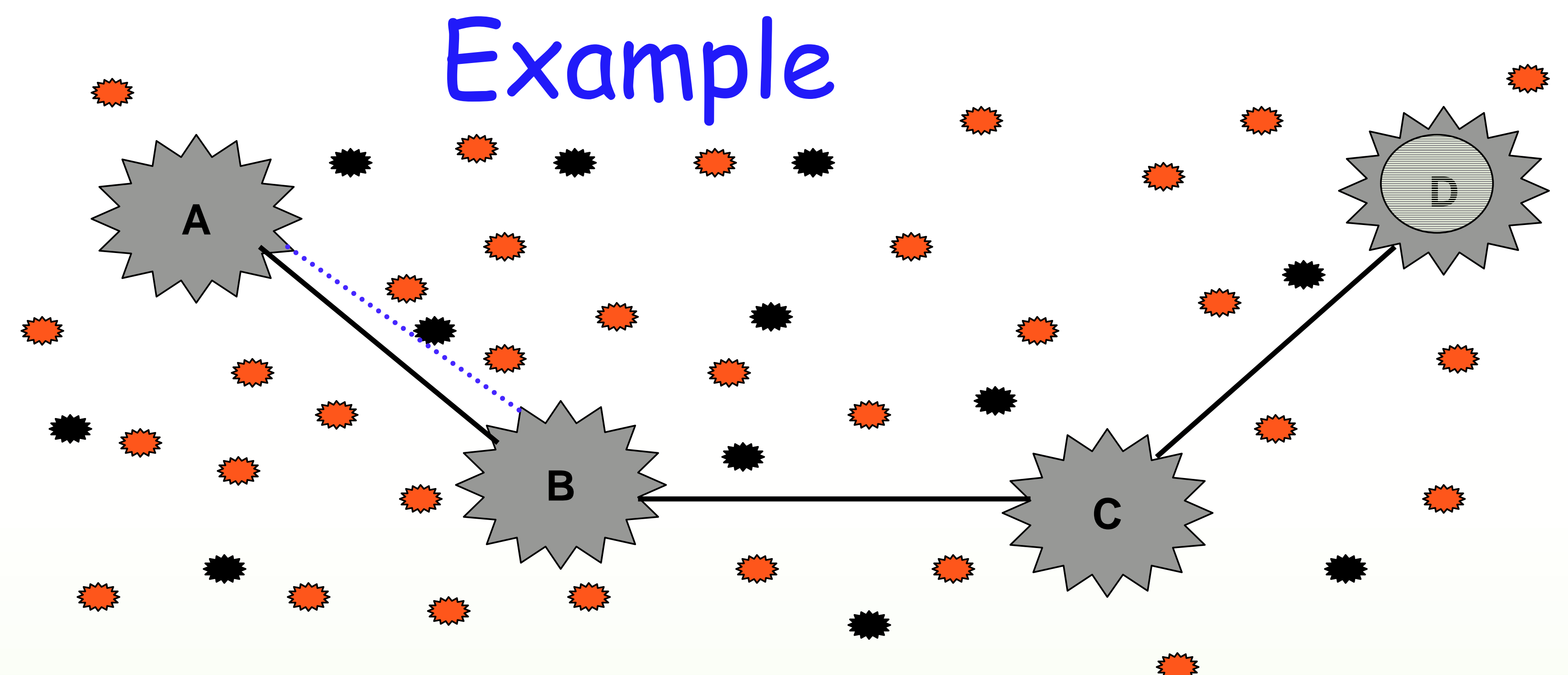
- When a node asks you for a key that you previously found, tell it where and when!

Query Cache

- When someone else queries you for a key, remember:
  - What they queried for.
  - Whom they were.
  - When they queried you.
- Also: tell them if anyone else is looking for the same key and when they were looking!

Like path-based replication, but passive!

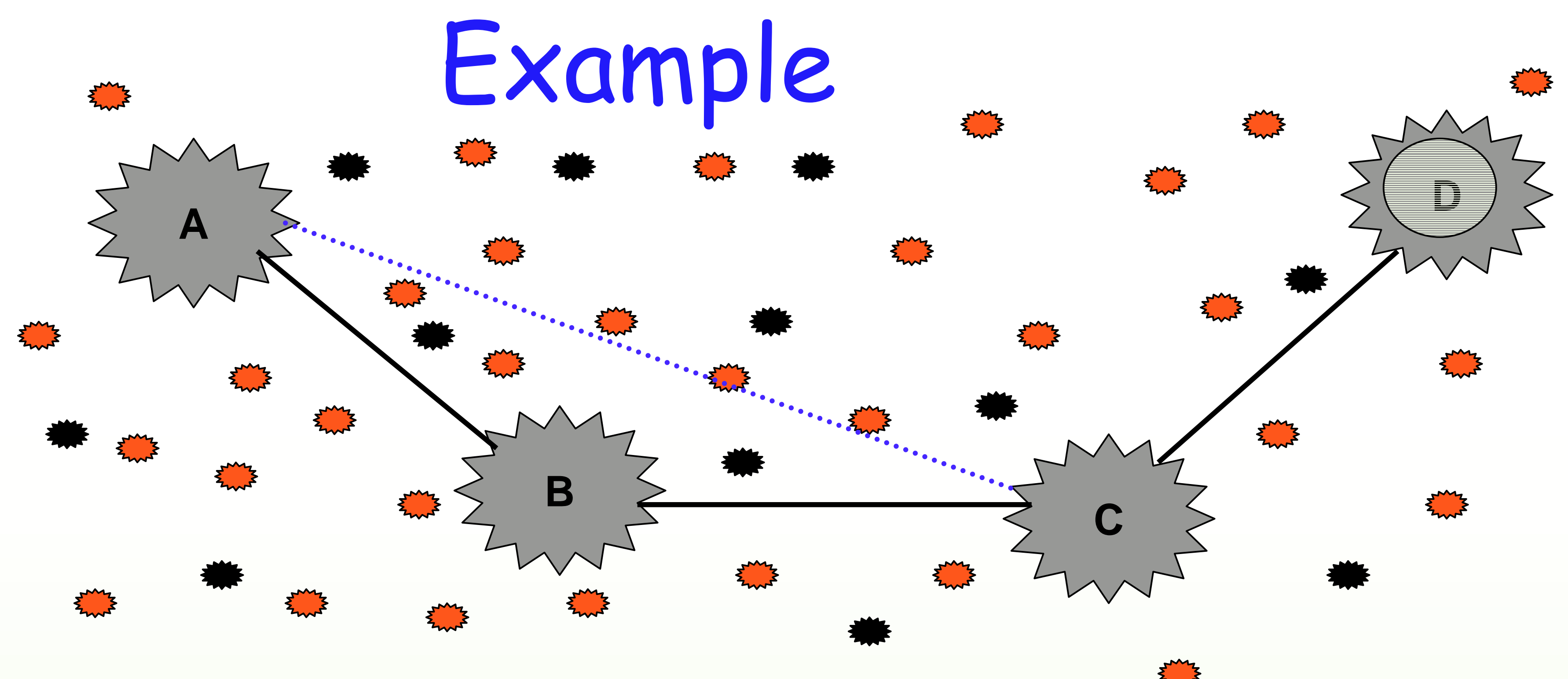
Example



A queries for key x which is located at D.

NODE	Query Cache	Results Cache
A	-	-
B	A	-
C	-	-
D	-	-

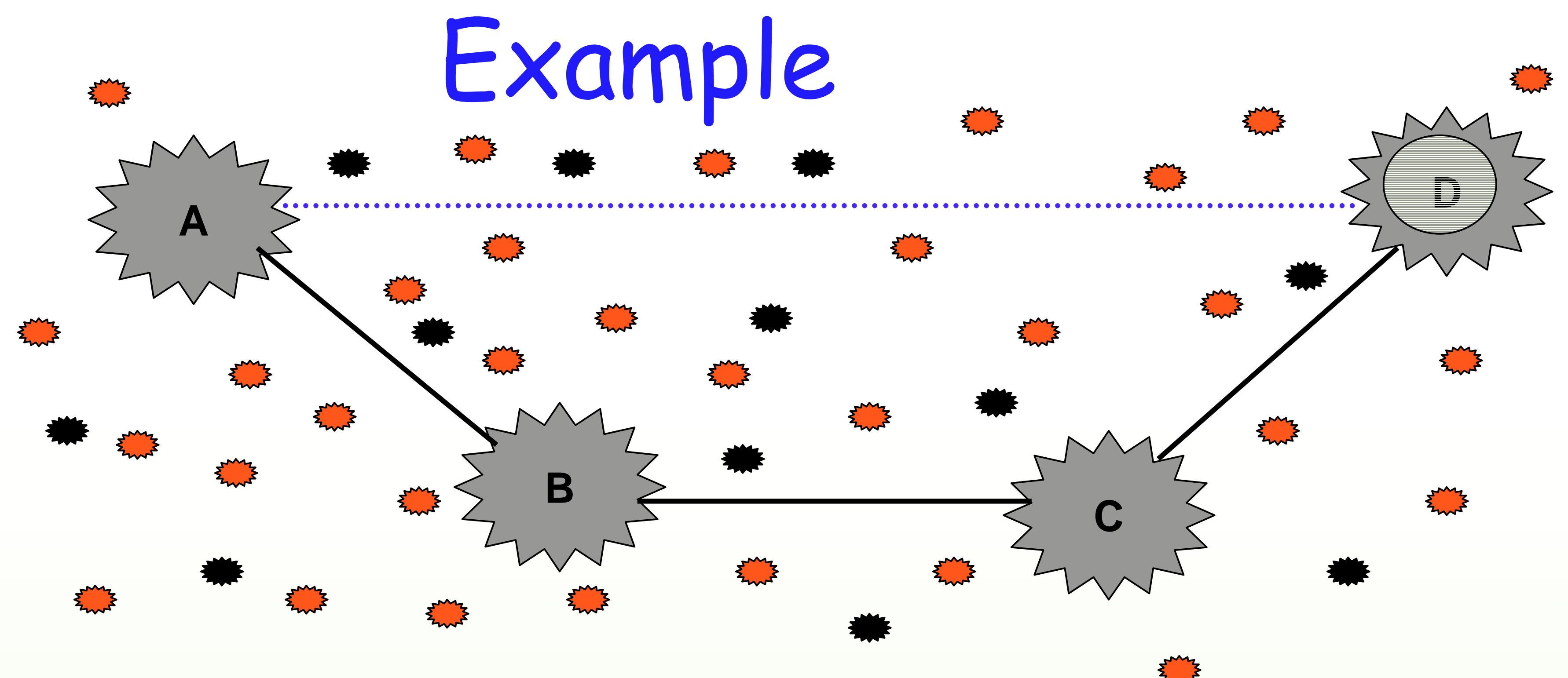
Example



A queries for key x which is located at D.

NODE	Query Cache	Results Cache
A	-	-
B	A	-
C	A	-
D	-	-

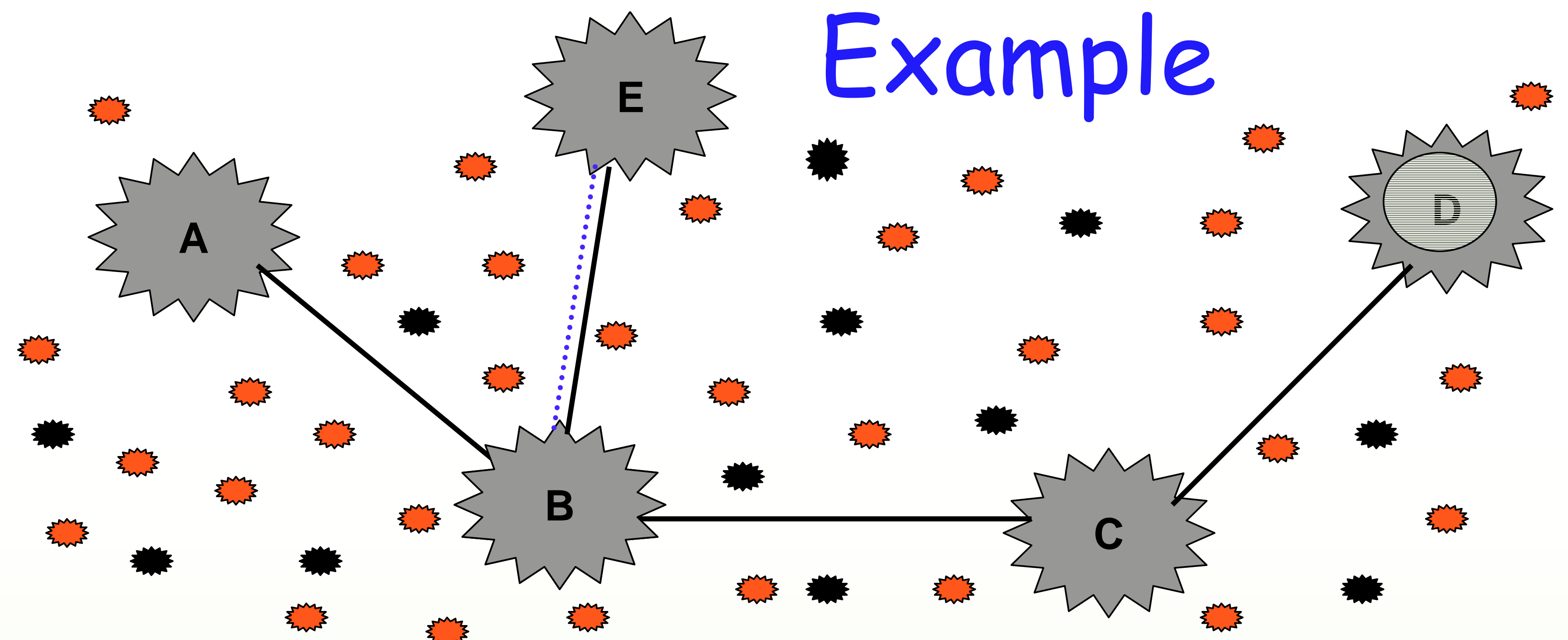
Example



A queries for key x which is located at D.

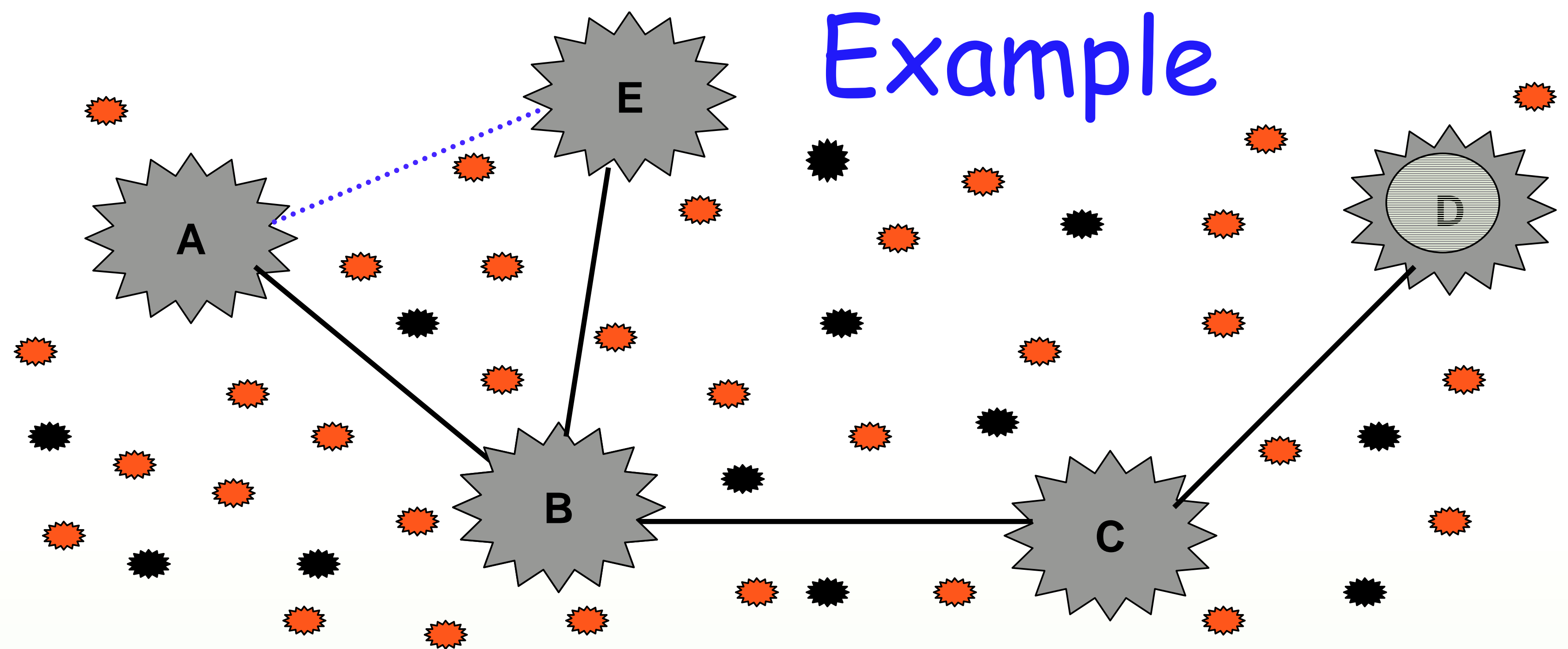
NODE	Query Cache	Results Cache
A	-	D
B	A	-
C	A	-
D	A	-

Example



Now E searches for key x .

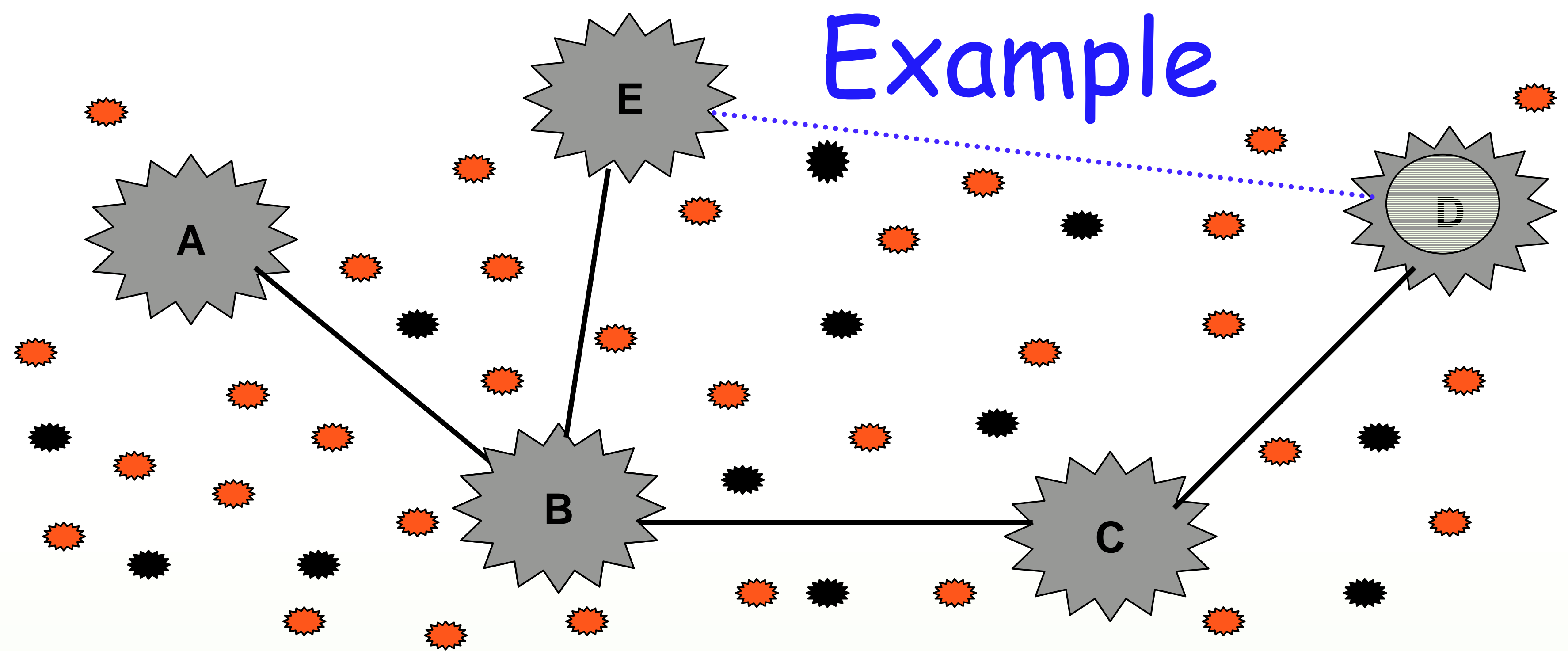
NODE	Query Cache	Results Cache
A	-	D
B	A	-
C	A	-
D	A	-
E	-	-



E follows B's query cache hint to A.

NODE	Query Cache	Results Cache
A	-	D
B	A,E	-
C	A	-
D	A	-
E	-	-

Example



E follows A’s result cache hint to D.

NODE	Query Cache	Results Cache
A	-	D
B	A,E	-
C	A	-
D	A,E	-
E	-	D

Soft Structure

- It's easy to add Chord-like structure!
- Responsibility cache:
  - Key-value pairs that are nearby in identifier space.
- Structured neighbor list:
  - In addition to random neighbors, add structured neighbors.

Flexibility and Control

- The user is in control of the search process!!
- Flexibility:
  - The user may choose to trust a node and use its cached information (Fast Search).
  - The user may NOT trust a peer's cache and instead use a BFS (Reliable Search).
  - Hybrids..

Conclusion

- In the best case, Chord-like structure and caches allow very efficient search.

- In the worst case, a node can search more if it really cares about search results!

Dumb network, smart end-hosts!

Future Work

- Ensure that non-malicious paths are likely to exist.
- Prevent other attacks on the system.
  - *What are they?*
- Quantify benefits of our caching schemes.