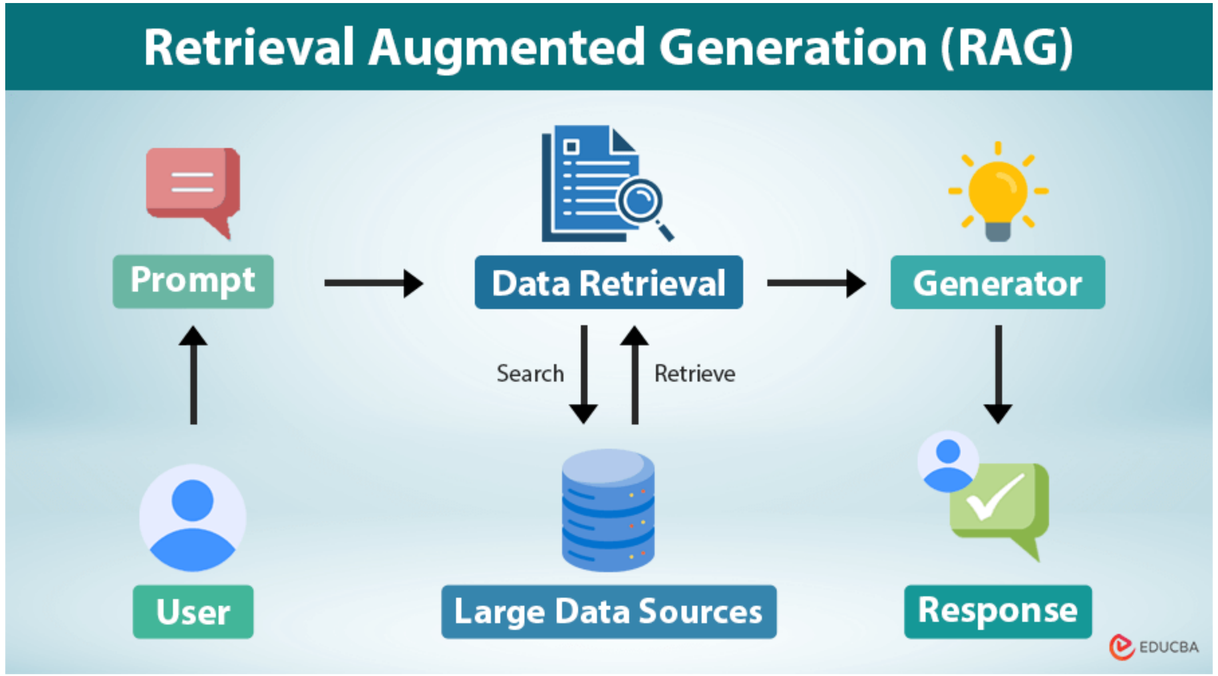
Project Catapult

A project proposal - followup on Proximity

Mathis Randl for SaCS, 22/8/25

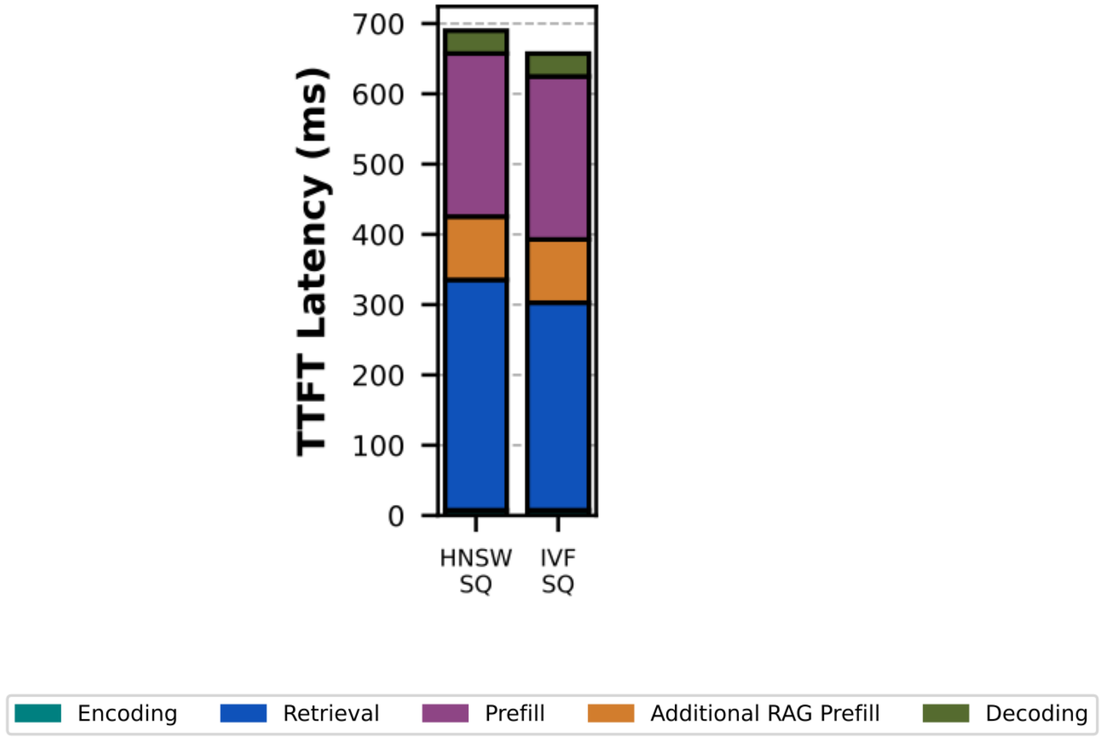
Motivation

Vector databases are cool



Motivation

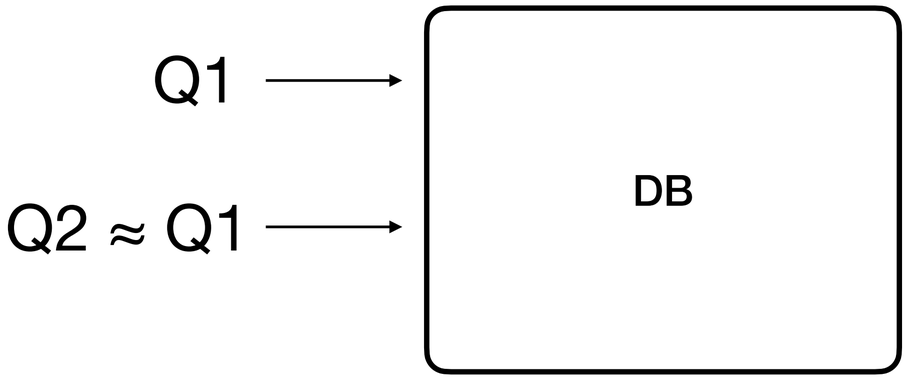
… but vector databases are slow



- In a typical setup (logarithmic approximate search, 9B LLM)
- **RAG document retrieval takes ~50% of the initial latency!** Plus some extra inference cost
- Shen et al. [1], we also observed similar numbers in our experiments

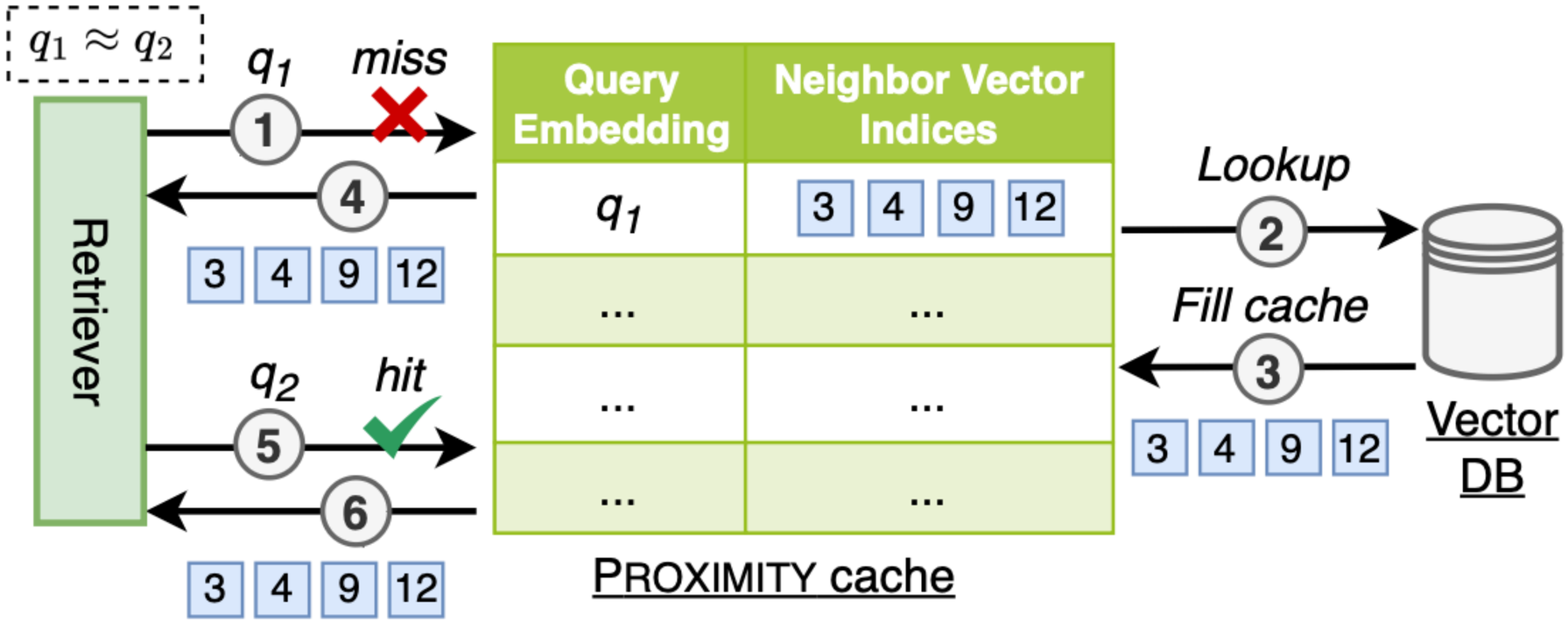
[1] https://arxiv.org/pdf/2412.11854v1

Root cause: discarding query patterns



Memory-less: Doing twice the work!

Proximity


$q_1 \approx q_2$
Retriever
q_1
miss
1
4
3 4 9 12
q_2
hit
5
6
3 4 9 12
Query Embedding
Neighbor Vector Indices
q_1
3 4 9 12
...
...
...
...
...
...
PROXIMITY cache
Lookup
2
Fill cache
3
3 4 9 12
Vector DB

Proximity has problems

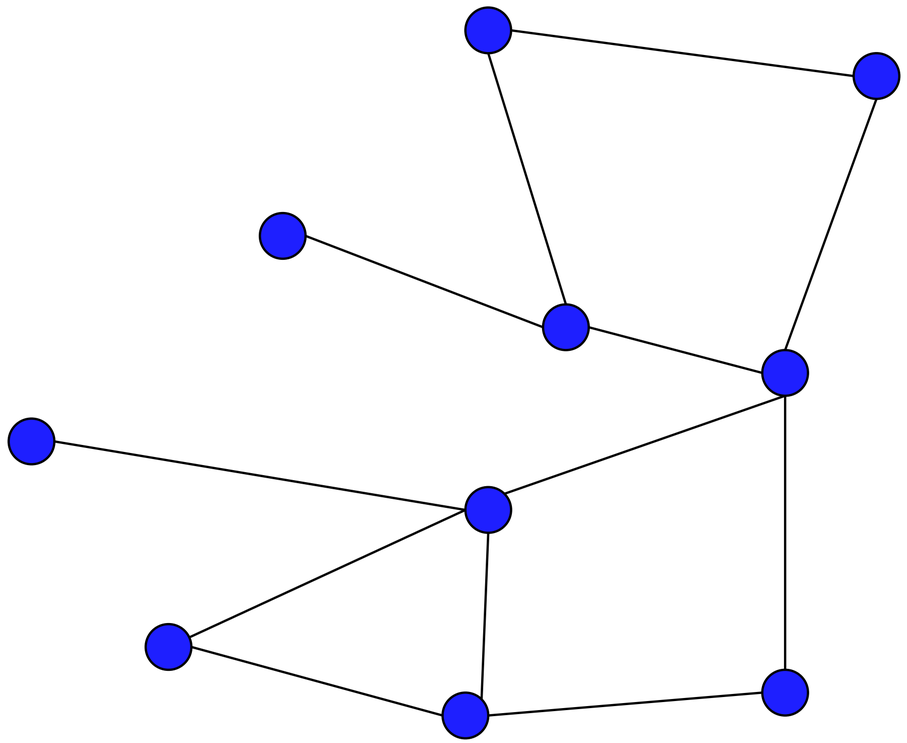
- Tolerance is **arbitrary**
- **Eviction** issues
- **Zero** quality guarantees
- Mildly similar queries **not** mildly useful to each other

Proximity has problems

- Tolerance is **arbitrary**
- **Eviction** issues
- **Zero** quality guarantees
- Mildly similar queries **not** mildly useful to each other

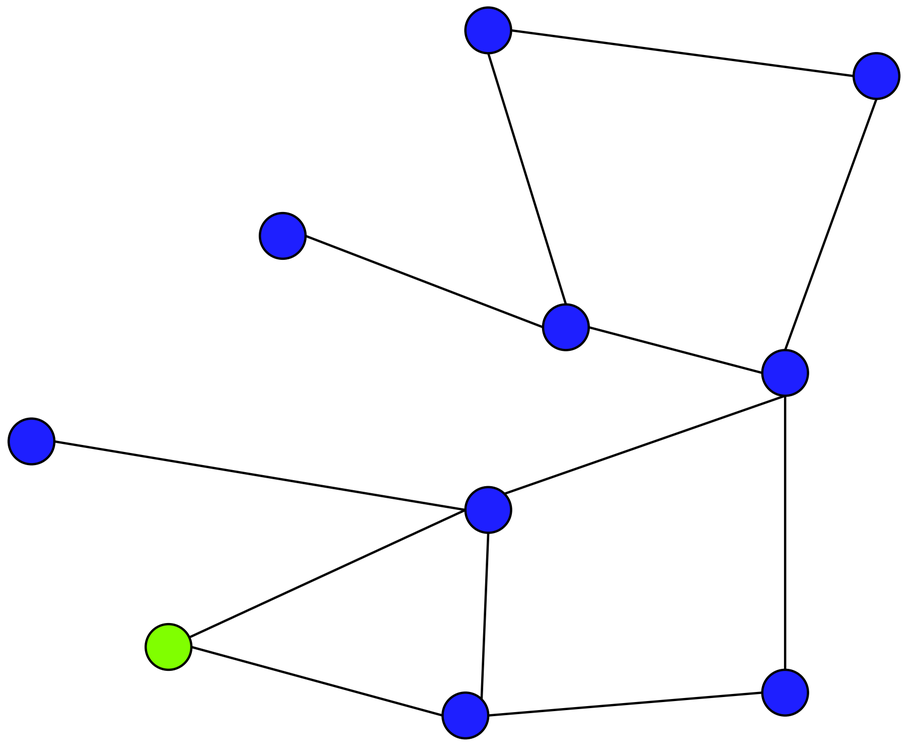
Remove the cache, improve graph

Solution: look at graph database



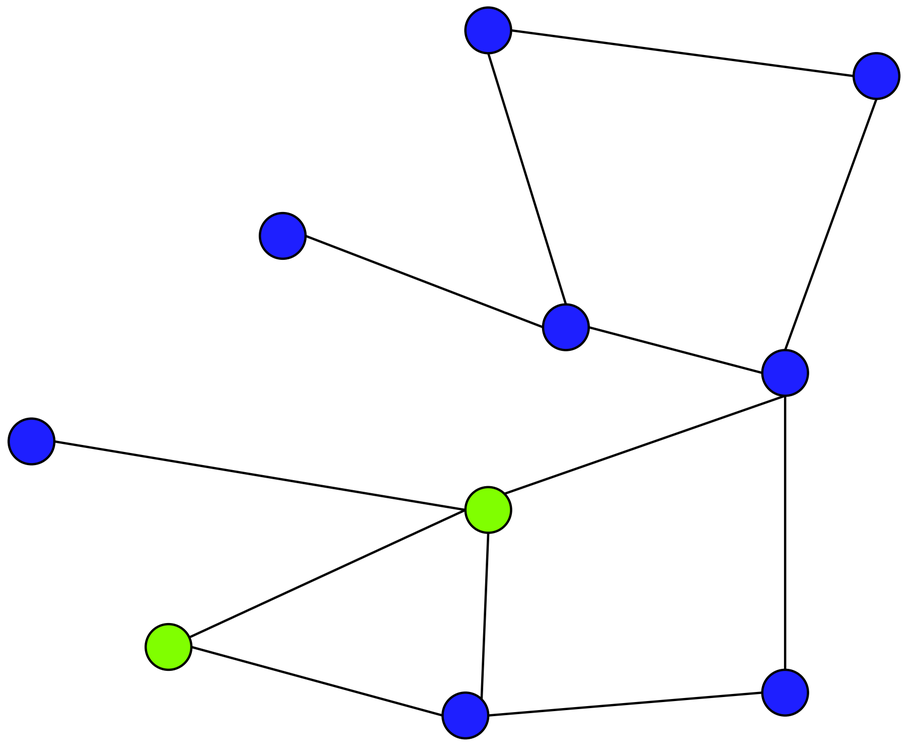
✕ Q1

Solution: look at graph database



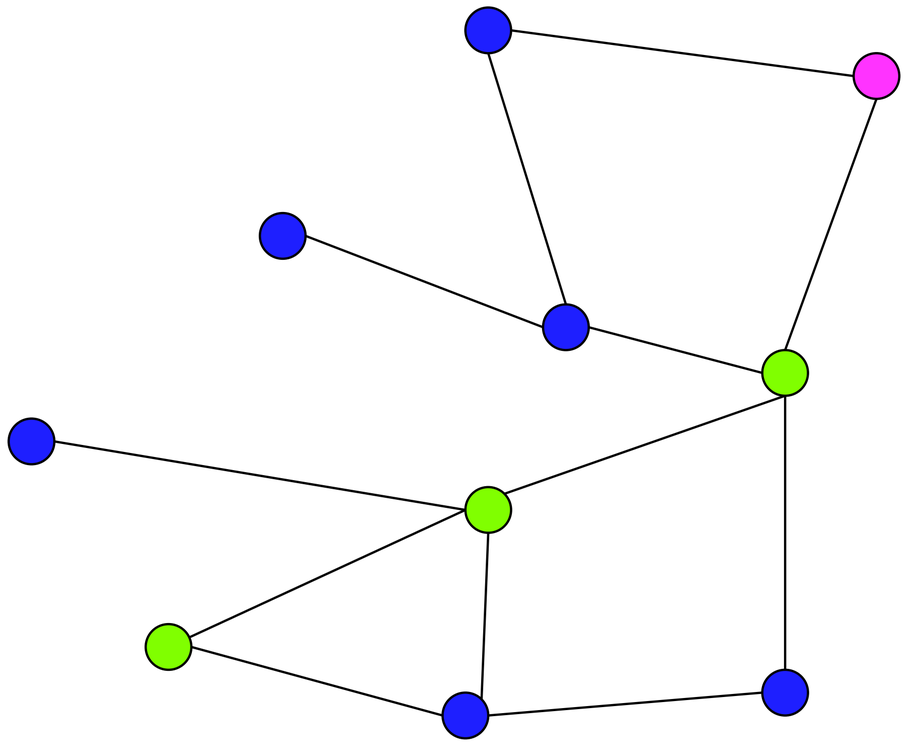
✕ Q1

Solution: look at graph database



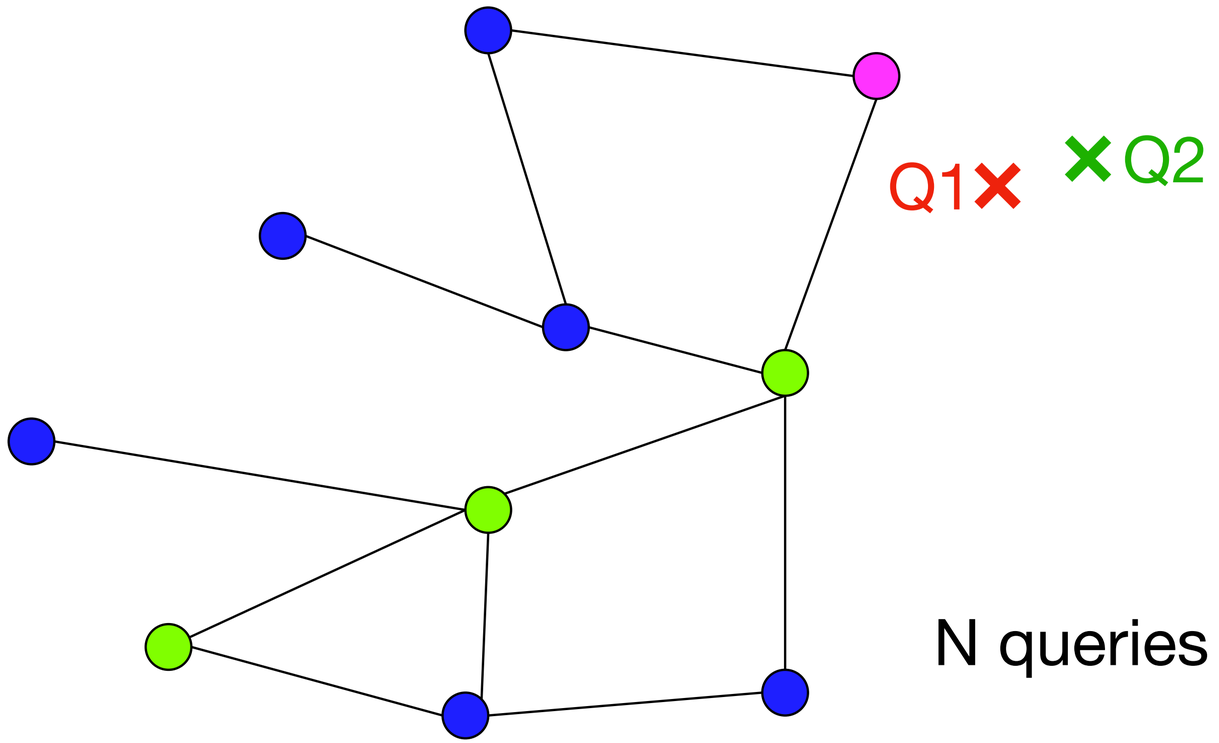
✕ Q1

Solution: look at graph database



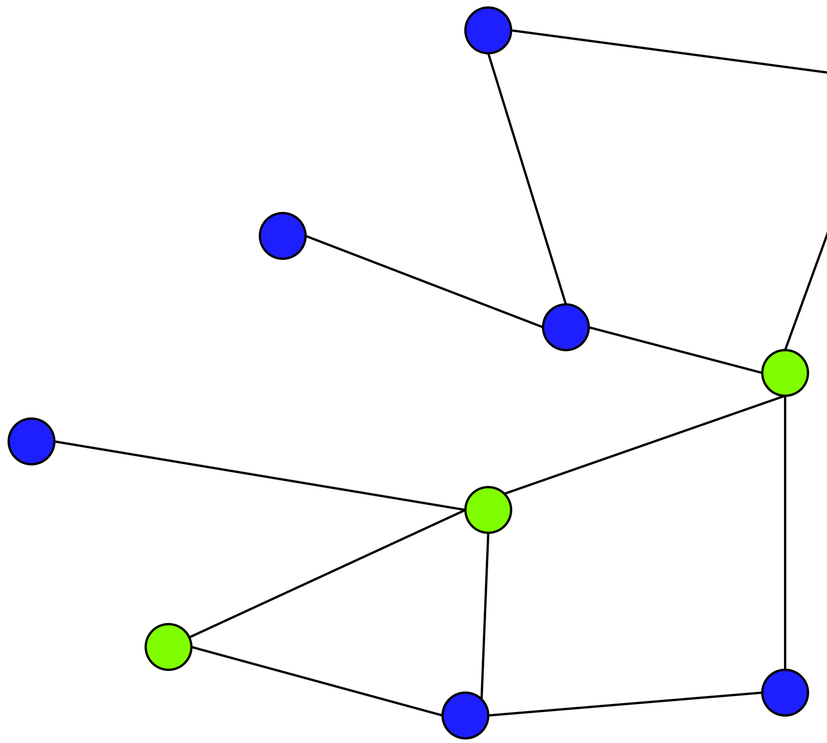
✗ Q1

Solution: look at graph database



N queries = N*time (bad)

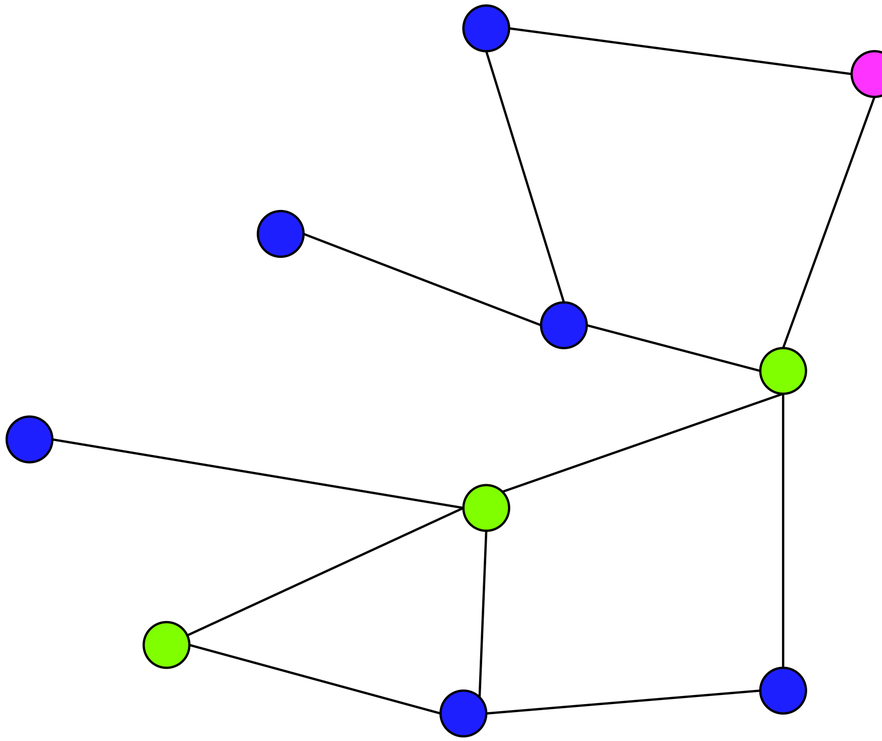
What if we had a memory oracle?



Q1✕

Q1: as usual

What if we had a memory oracle?



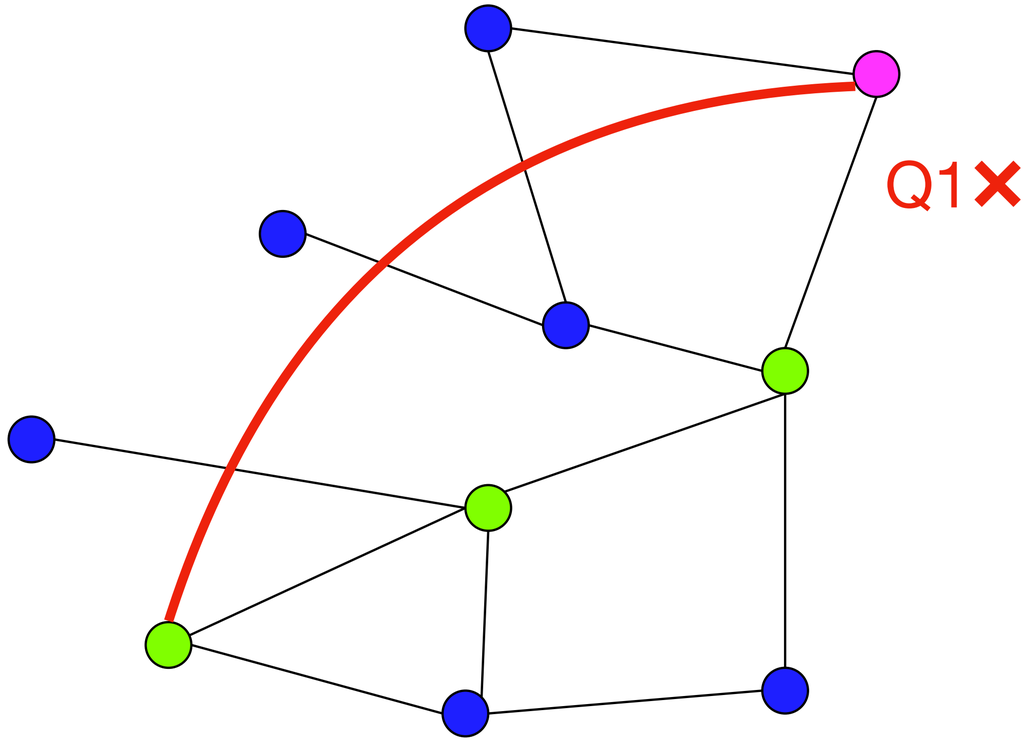
Q1 ✗ ✗ Q2

Q1: as usual

Q2: start at pink node

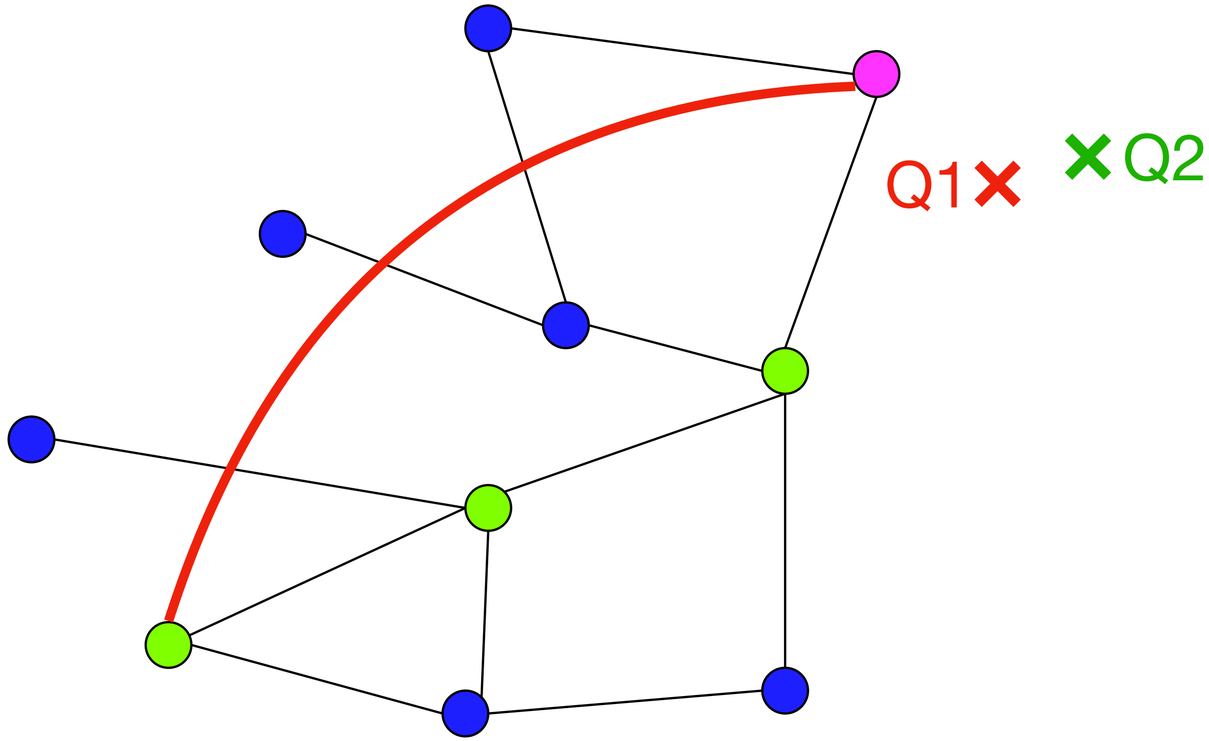
Implementing the oracle

Introducing catapults



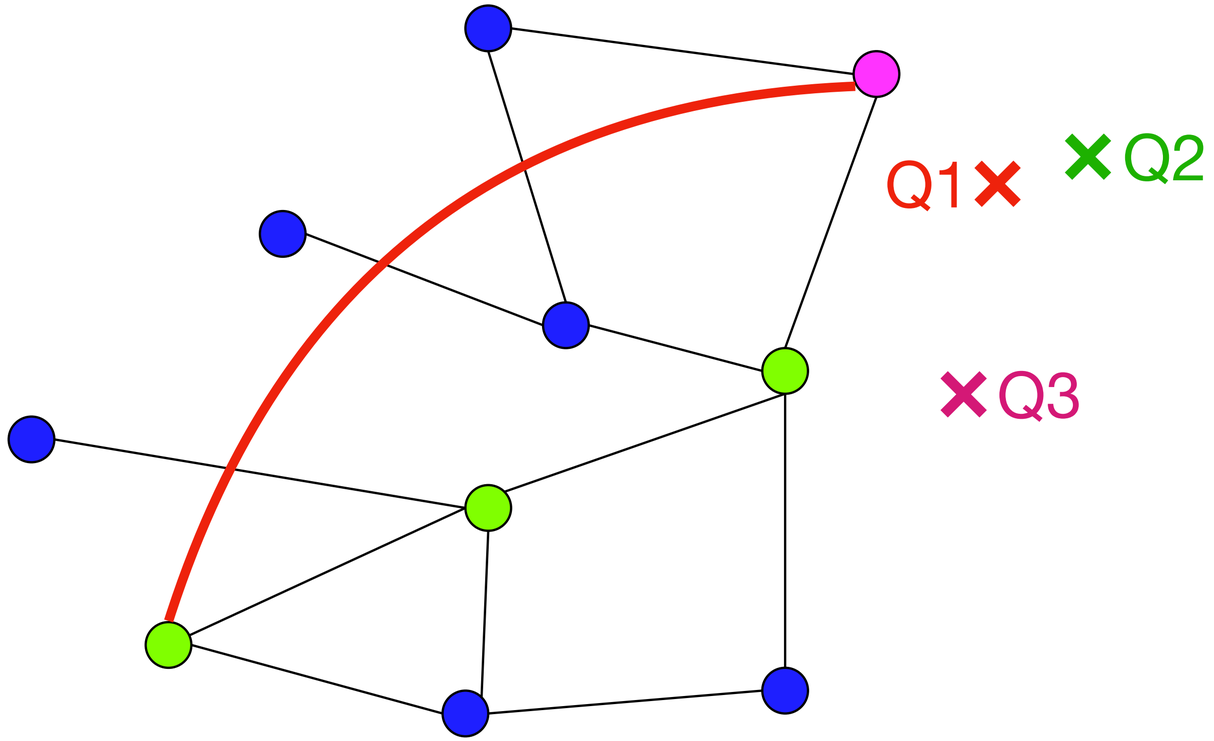
Implementing the oracle

Introducing catapults



Implementing the oracle

Introducing catapults



General algorithm

- Pick starting point
- Greedily follow graph until locally best node
- NEW: add catapult from starting point to finish point
- Return as usual

Proximity problems

- Tolerance -> handled implicitly by the catapult
- Cache thrashing/eviction issues -> not relevant anymore
- Zero guarantees -> local optimum now guaranteed
- Mildly similar queries not mildly useful to each other -> not true anymore

Open questions

- Catapult eviction / catapult proliferation
- Picking entry point -> yearns for LSH
- Static portion of graph ?
- Multithreading

Conclusion

- Proposed (afaik) new method for exploiting query bias in databases
- Based on mutable/dynamic graphs as opposed to traditionally static ones
- Solves a lot of headaches of Proximity w/ hopefully similar benefits

Paper timeline