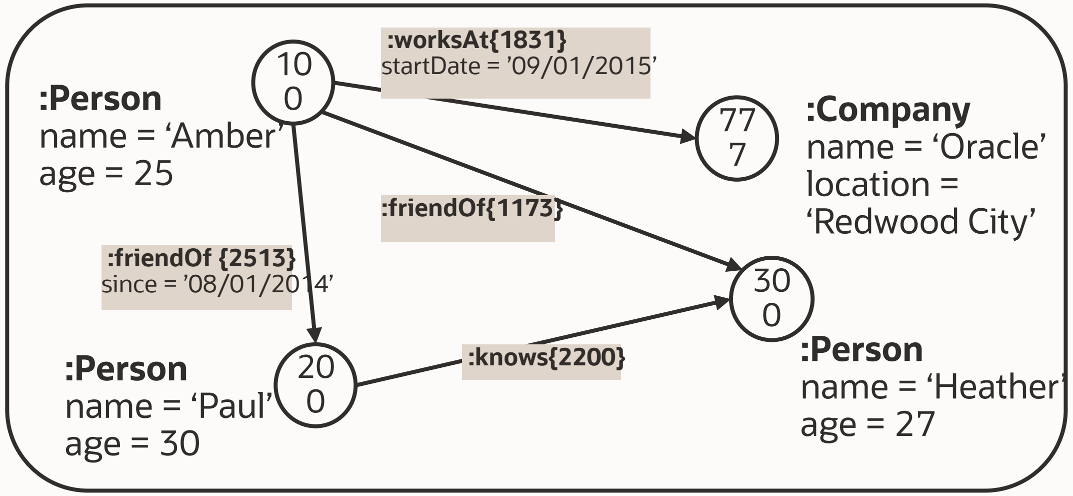
# EFFICIENT PROPERTY PROJECTIONS OF GRAPH QUERIES OVER RELATIONAL DATA

Mikael Morales (Oracle Labs), Vlad Ioan Haprian (Oracle Labs), Srinivas Karthik (EPFL), **Danica Porobic** (Oracle), Laurent Daynés (Oracle Labs), Anastasia Ailamaki (EPFL & RAW Labs)

# What is a property graph?

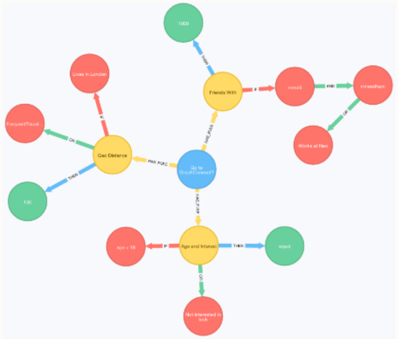
- A set of vertices and edges
- Edges connect vertices
- Vertices and edges can have
  - labels (one or more)
    - A label is an identifier
    - that also provides typing information
  - properties/attributes (zero or more)
    - By virtue of properties being associated with labels.
- A property is a typed key/value pair.



# Full materialization

```
SELECT * FROM GRAPH_TABLE(MY_GRAPH
MATCH (a is Person) → (b is Car)
COLUMNS (a.age, b.brand) ) T;
```

Graph Runtime



**Paths**

| a.id | b.id |
|---|---|
| 0 | 4 |
| 1 | 0 |
| 3 | 3 |

**Projection**

| a.age | b.brand |
|---|---|
| 32 | Mercedes |
| 43 | Ferrari |
| 56 | BMW |

**Array lookups by ids**

In-Memory storage

Person Table

| id | age | name | city |
|---|---|---|---|
| 0 | 32 | Joe | Lausanne |
| 1 | 43 | Bob | New York |
| 2 | 23 | Alice | Geneva |
| 3 | 56 | John | Zurich |

Car Table

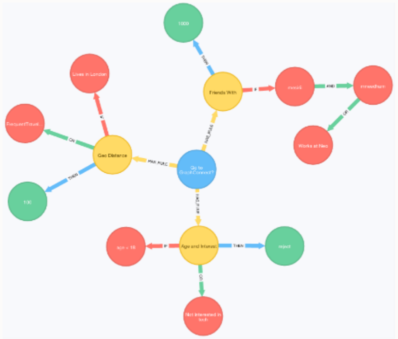
| id | brand |
|---|---|
| 0 | Ferrari |
| 1 | Renault |
| 2 | Toyota |
| 3 | BMW |
| 4 | Mercedes |

**Low execution times but very high memory footprint!**

# Batch on-demand projections

SELECT * FROM GRAPH_TABLE(MY_GRAPH
MATCH (a is Person) → (b is Car)
COLUMNS (a.age, b.brand) ) T;

Query Runtime



**Paths**

| a.id | b.id |
|---|---|
| 0 | 4 |
| 1 | 0 |
| 3 | 3 |

**Projection**

| a.age | b.brand |
|---|---|
| 32 | Mercedes |
| 43 | Ferrari |
| 56 | BMW |

Table scans for each batch of paths

Persistent storage

Person Table

| id | age | name | city |
|---|---|---|---|
| **0** | **32** | Joe | Lausanne |
| **1** | **43** | Bob | New York |
| 2 | 23 | Alice | Geneva |
| **3** | **56** | John | Zurich |

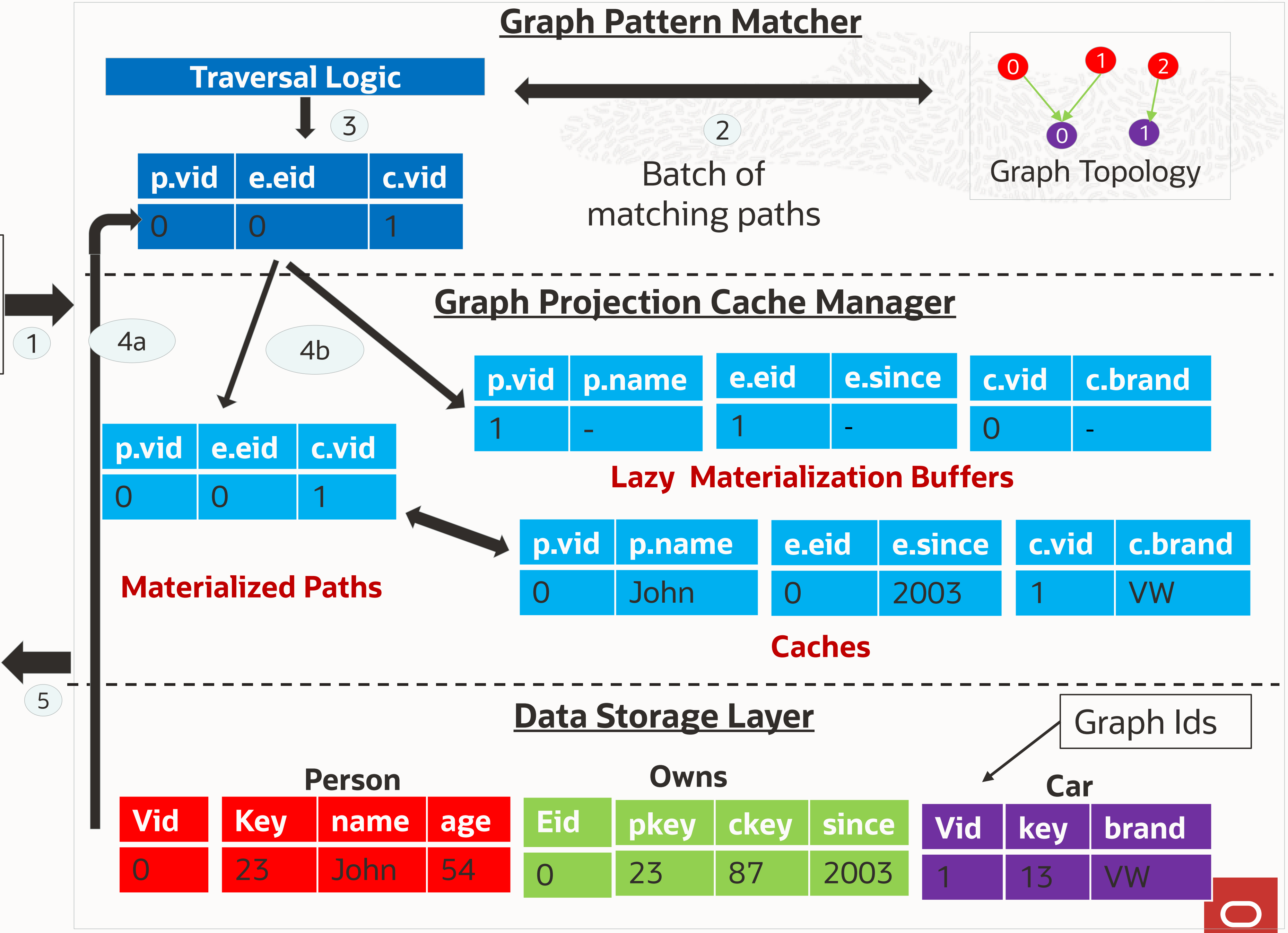
Car Table

| id | brand |
|---|---|
| **0** | **Ferrari** |
| 1 | Renault |
| 2 | Toyota |
| **3** | **BMW** |
| **4** | **Mercedes** |

**Too slow as the number of matching paths quickly explodes**

# Graph Cache Manager: Overview

**SELECT** p.name, e.since, c.brand
**FROM** G **MATCH**
(p: person)->[e:owns]->(c:car)
**WHERE** p.name = ‘John'



| p.vid | e.eid | c.vid |
|---|---|---|
| 0 | 0 | 1 |

**Lazy Materialization Buffers**

| p.vid | p.name |
|---|---|
| 1 | - |

| e.eid | e.since |
|---|---|
| 1 | - |

| c.vid | c.brand |
|---|---|
| 0 | - |

**Materialized Paths**

| p.vid | e.eid | c.vid |
|---|---|---|
| 0 | 0 | 1 |

**Caches**

| p.vid | p.name |
|---|---|
| 0 | John |

| e.eid | e.since |
|---|---|
| 0 | 2003 |

| c.vid | c.brand |
|---|---|
| 1 | VW |

**Person**

| Vid | Key | name | age |
|---|---|---|---|
| 0 | 23 | John | 54 |

**Owns**

| Eid | pkey | ckey | since |
|---|---|---|---|
| 0 | 23 | 87 | 2003 |

**Car**

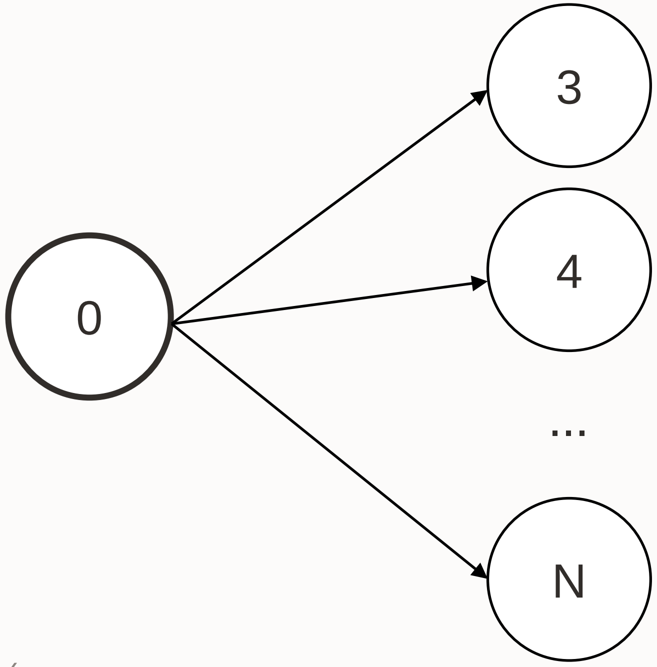
| Vid | key | brand |
|---|---|---|
| 1 | 13 | VW |

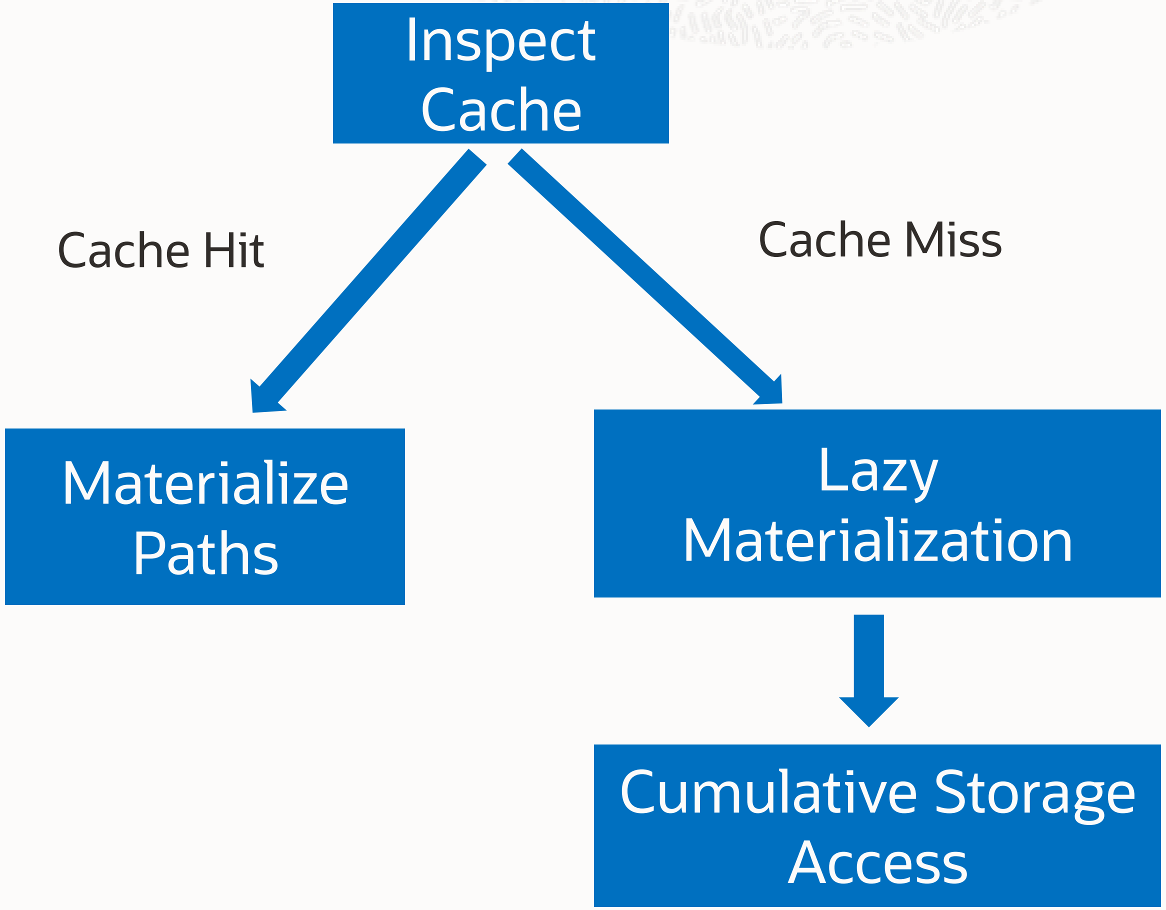
| p.name | e.since | c.brand |
|---|---|---|
| John | 2003 | VW |

# 1. Caching

## Key Idea

Highly Connected vertices are likely to appear in multiple paths => cache them



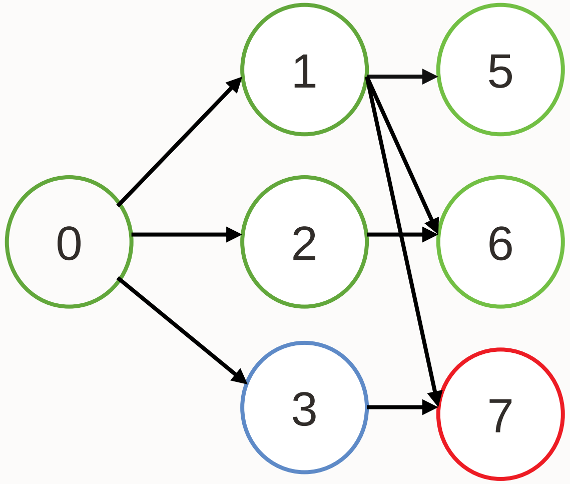


# 2. Prefetching

## Key Idea

At level $i$, siblings of the current vertex will appear subsequently => prefetch using graph topology

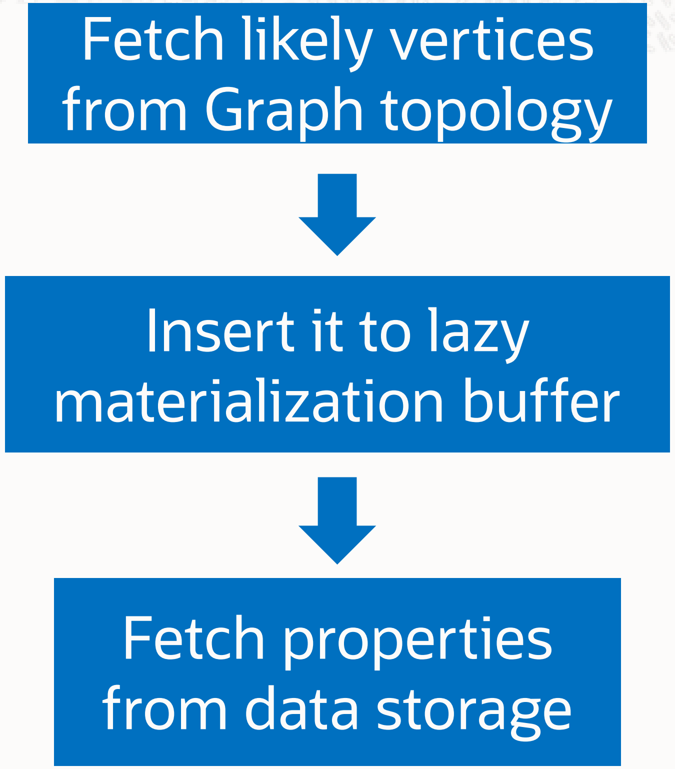
**Path pattern:** (a) → (b) → (c)



🟩 **Processed**

🟦 **Prefetched for level *b***

🟥 **Prefetched for level *c***


Fetch likely vertices from Graph topology

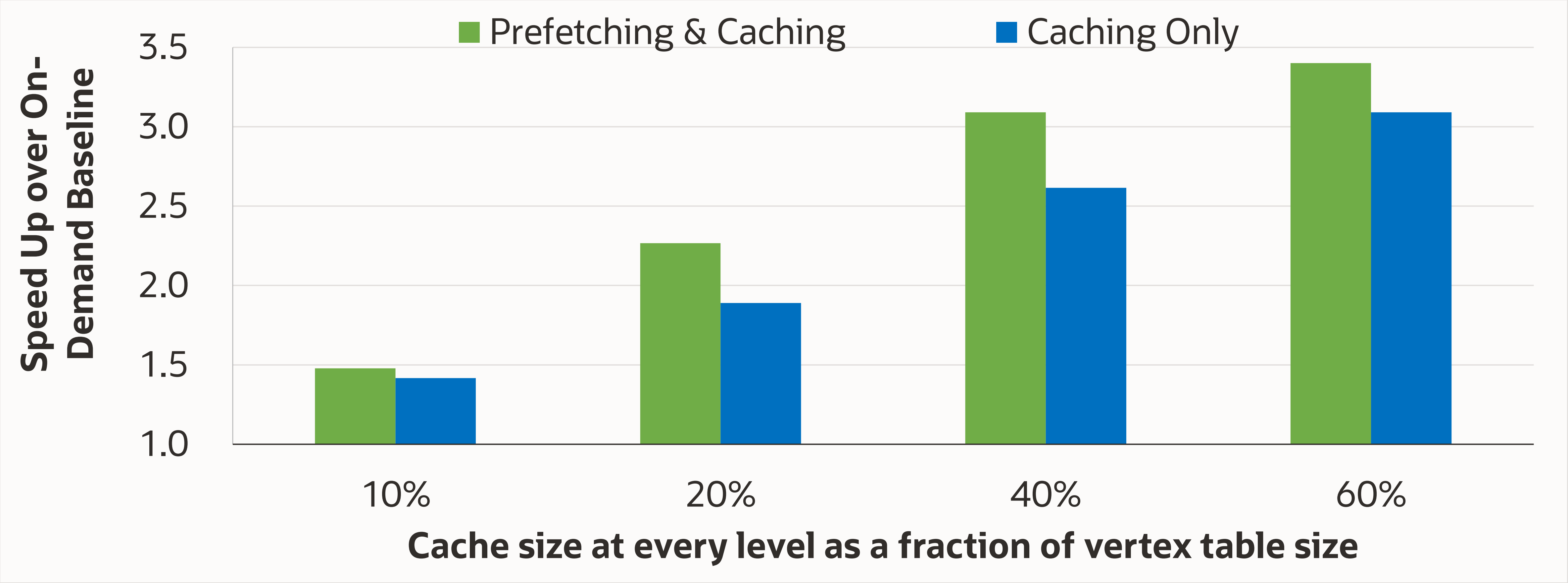
Insert it to lazy materialization buffer

Fetch properties from data storage

# Experimental Evaluation: Effect of Caching and Prefetching

Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz,
512GB DDR4 RAM @ 2133MHz
Single Threaded Execution

**Graph:** **LDBC 4GB**
**Projection:** T.agea, T.ageb, T.agec, T.aged
**Path pattern:** (a) → (b) → (c) → (d)
**Output Size:** **5 Billion Paths**



**Good improvement even with small caches**

# Conclusion

- First step toward enabling efficient property projections with controlled memory footprint
- Takes advantage of the structure of the graph
- Leverage efficient caching mechanisms

**Thank You**