OGO : Object Graph Programming

Aditya Thimmaiah
University of Texas at Austin

Leonidas Lampropoulos
University of Maryland

Christopher Rossbach
University of Texas at Austin

Milos Gligoric
University of Texas at Austin

TEXAS
The University of Texas at Austin

UNIVERSITY OF MARYLAND

NSF
CCF-2107206
CCF-2107291
CCF-2217696

Motivation

- Imperative programming is still the dominant paradigm (eg : Python and Java)
- Many tasks are easier to express using declarative paradigm (like LINQ)
- How do we enable programming by arbitrarily mixing imperative and declarative paradigms?
  - Create an instance of a class in Java: query(“CREATE (:`java.util.LinkedList`)”)
  - Find all instances of a class: query(“MATCH (n:`java.util.LinkedList`) RETURN n”)
  - Enforce Singleton invariant: query(“MATCH (n:`java.util.LinkedList`) RETURN COUNT(n) = 1”)
  - Check acyclicity of B-Trees: query(“MATCH (n:`Node`)-[:left|right*0..]->(m:`Node`) WHERE n.value = m.value RETURN TRUE”)

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified
- Implement two prototypes of OGO

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified
- Implement two prototypes of OGO
- Introduce optimizations to improve efficiency of query execution

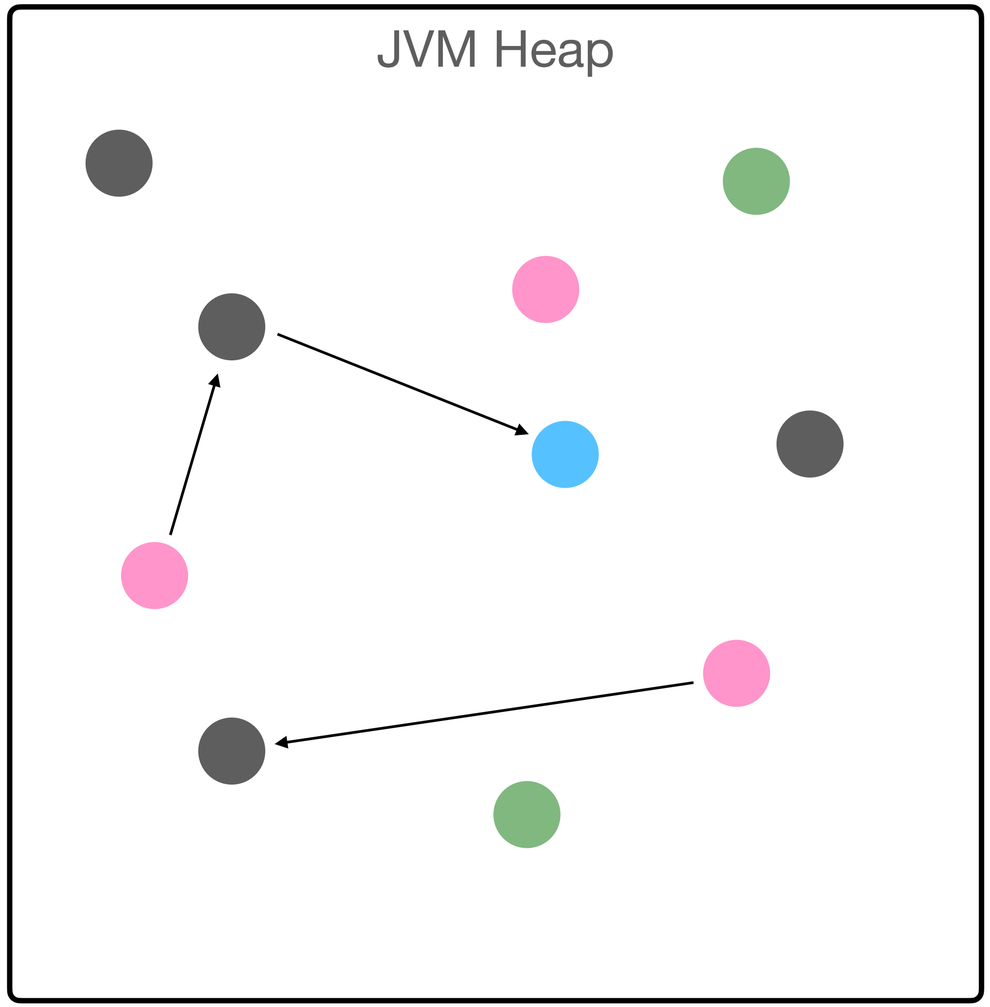
Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified
- Implement two prototypes of OGO
- Introduce optimizations to improve efficiency of query execution
- Evaluation
  - Demonstrate increase in expressivity by re-implementing imperative Java methods with OGO
  - Demonstrate robustness of OGO by re-implementing dozens of assertions in open-source projects with OGO

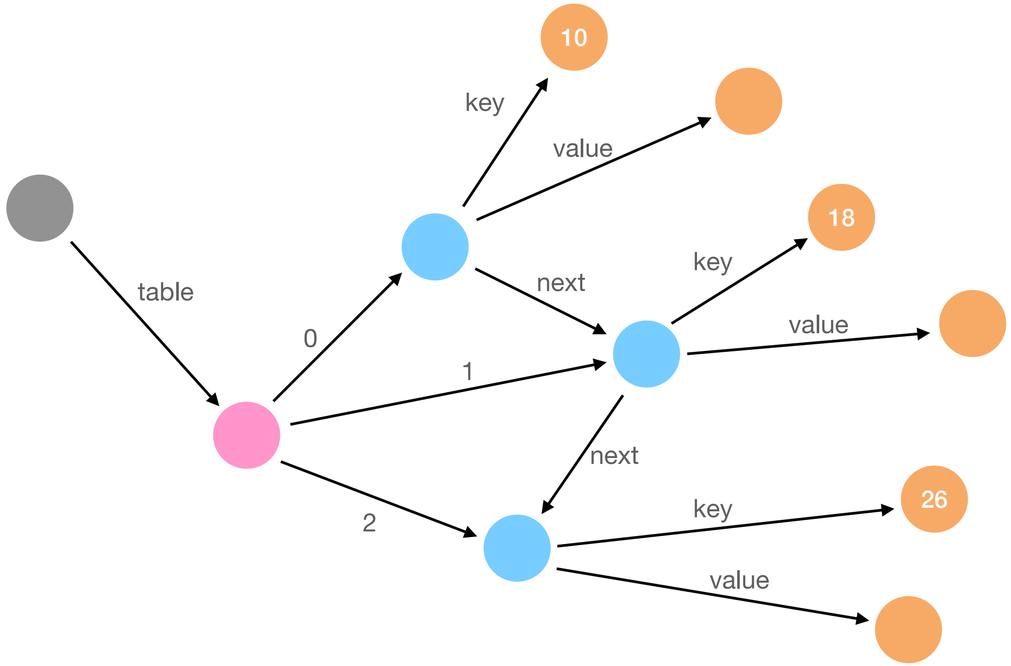
Insight behind OGO

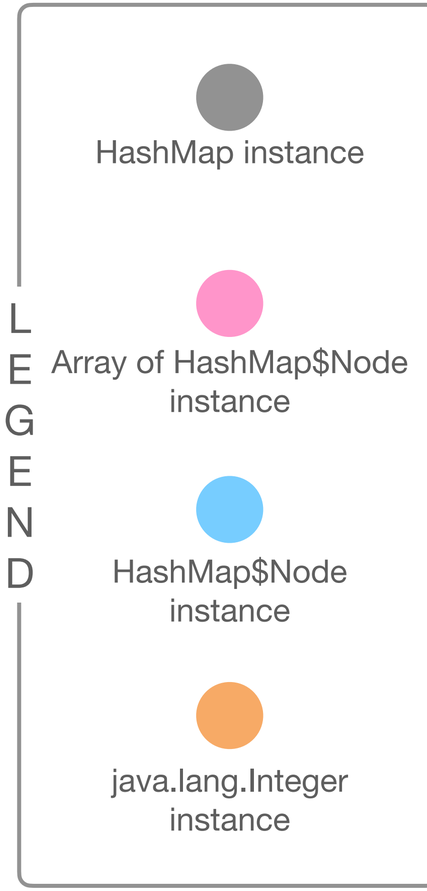
- We see the heap of the JVM as a graph database (eg Neo4j) with
  - Objects as nodes
  - References as edges
- We use expressive declarative graph query language Cypher to query and program the heap
- Example:

```
1.  ArrayList<Long> list = new ArrayList<Long>();
2.  list.add(10L);
3.  list.add(20L);
4.  list.add(30L);
5.  query("MATCH ({$1})-[:elementData]->()"
6.        +"-[]->({value : 20}) RETURN TRUE"
7.  );
```



Example | containsKey using OGO

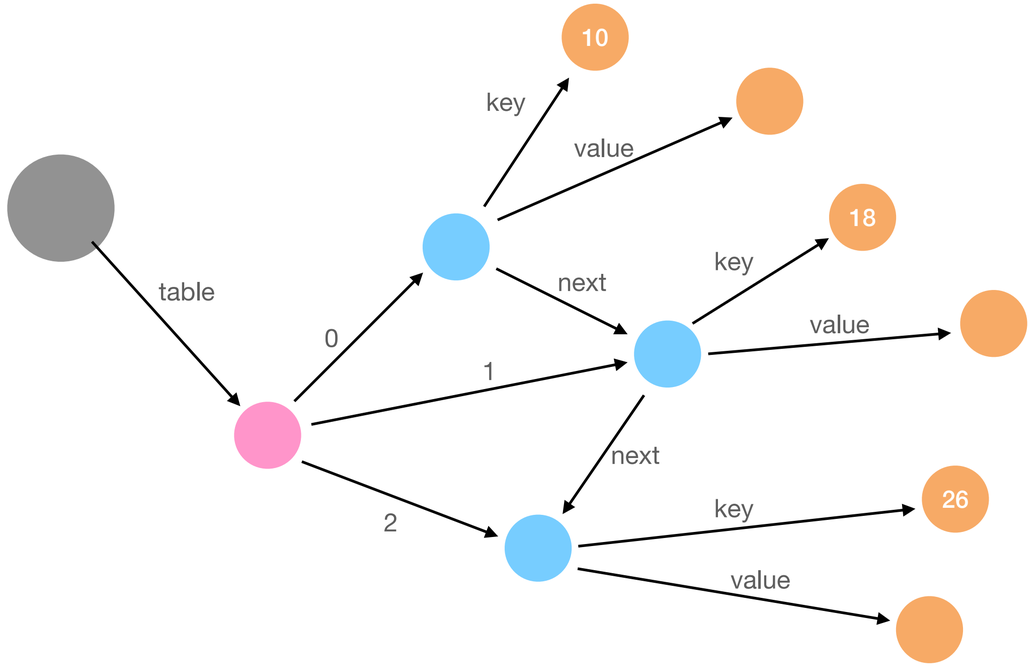


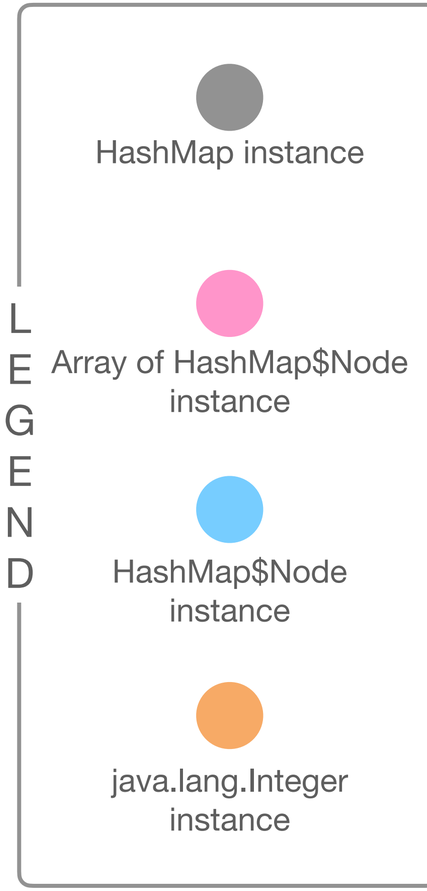



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

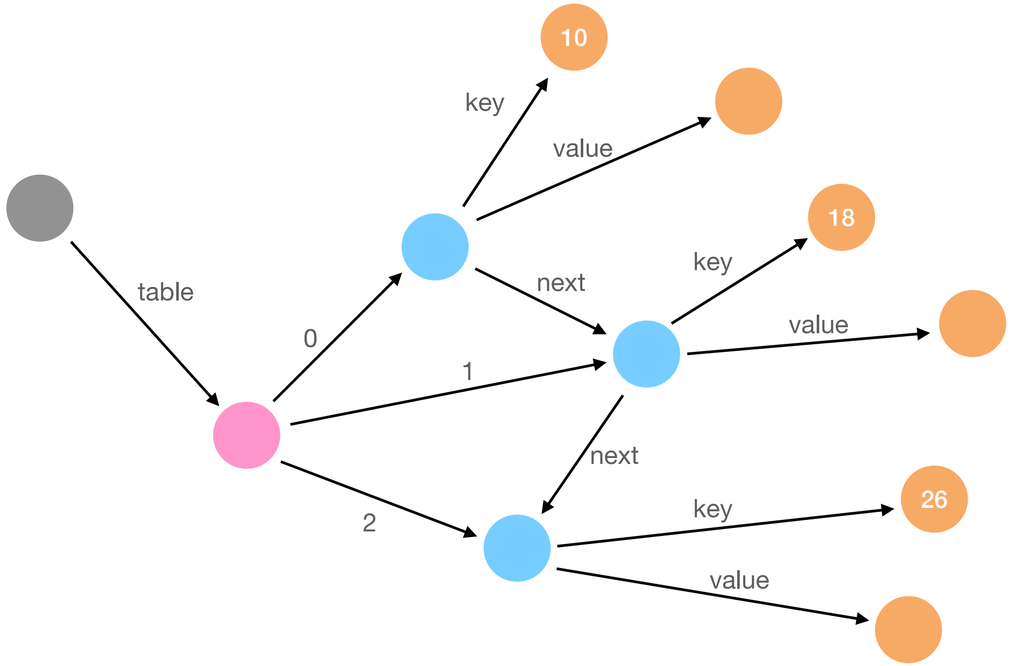


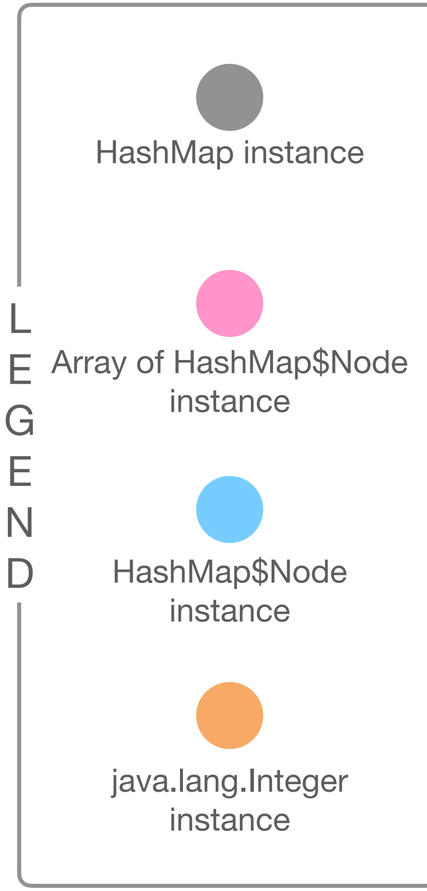



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                               implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

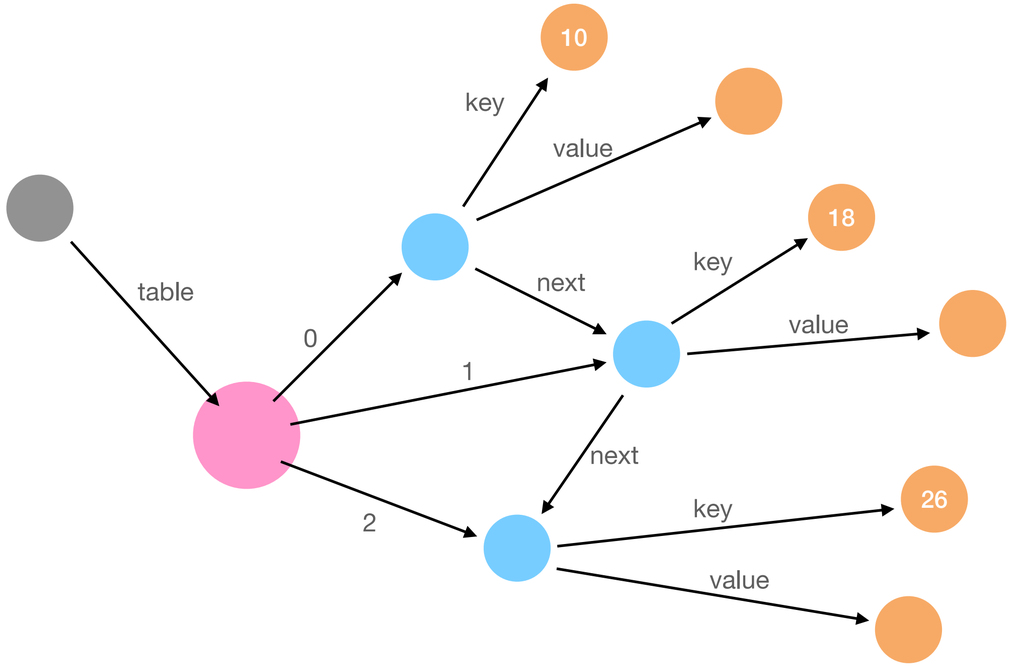


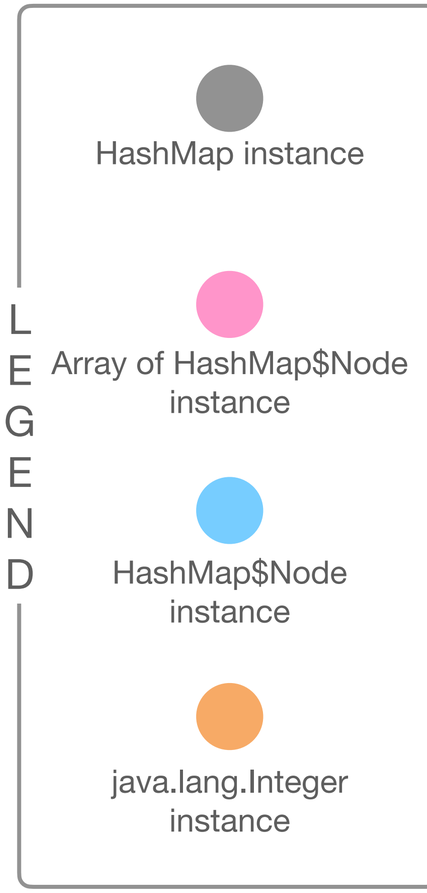



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

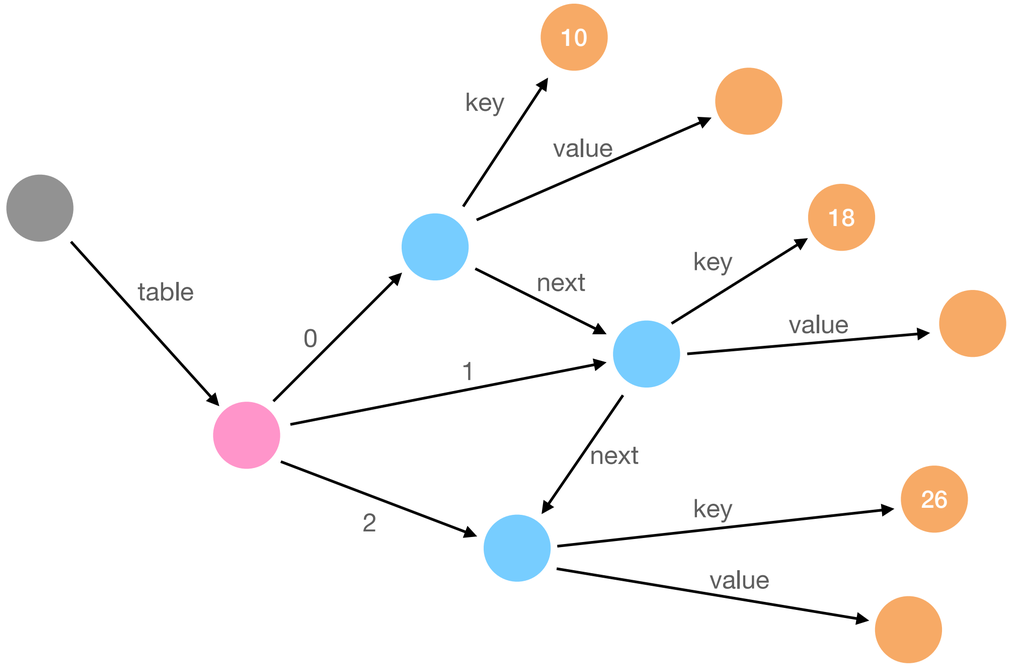


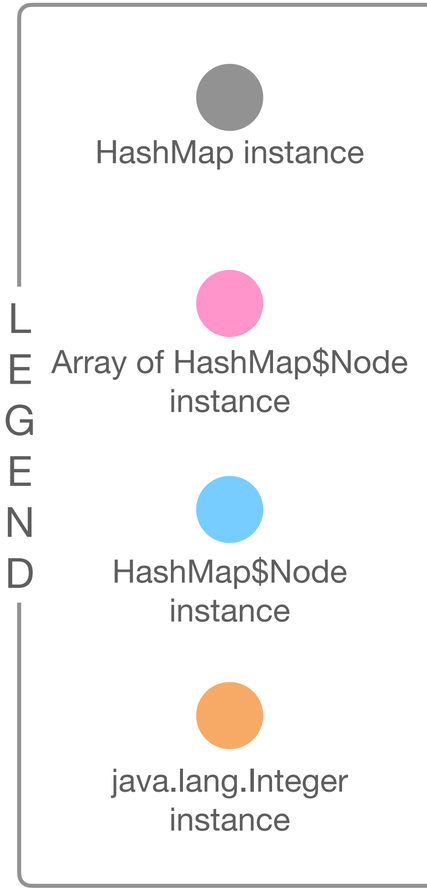



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

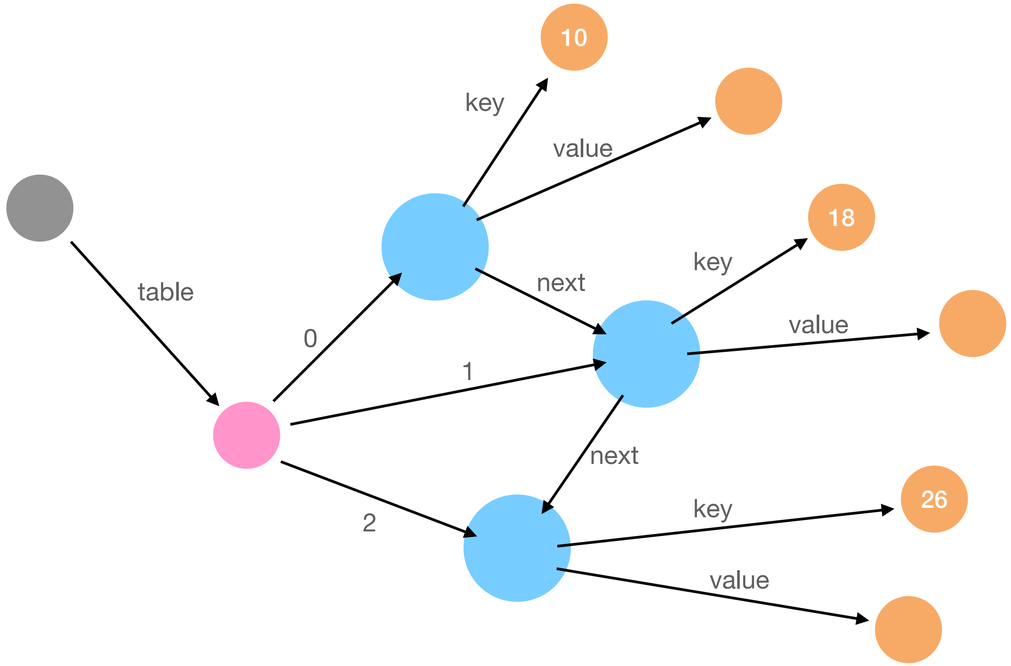


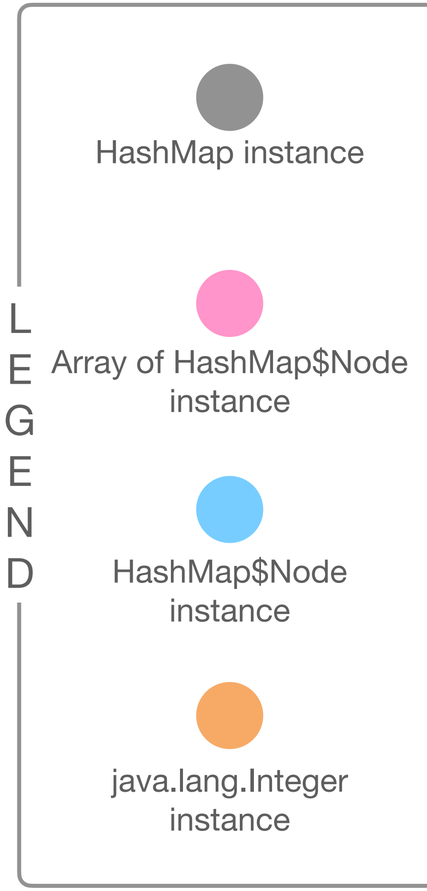



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

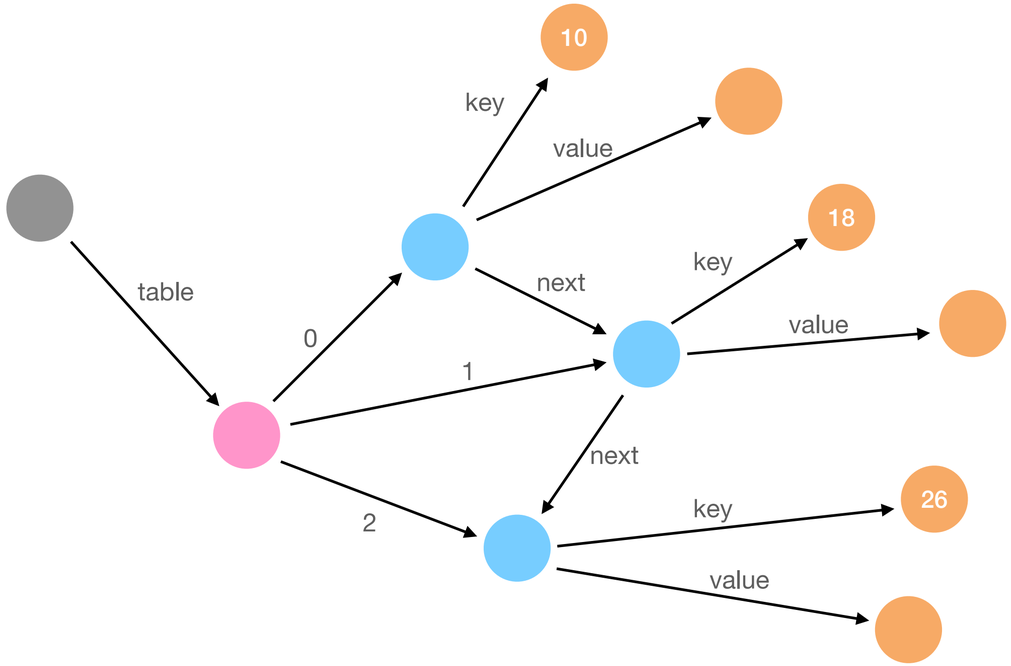


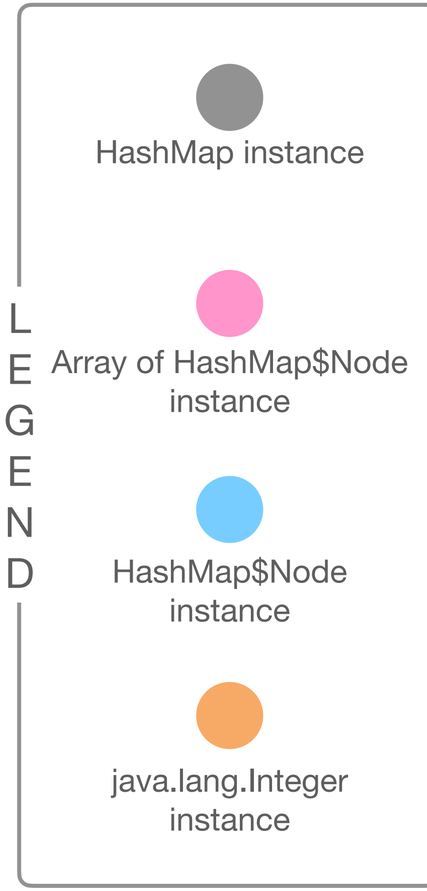



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

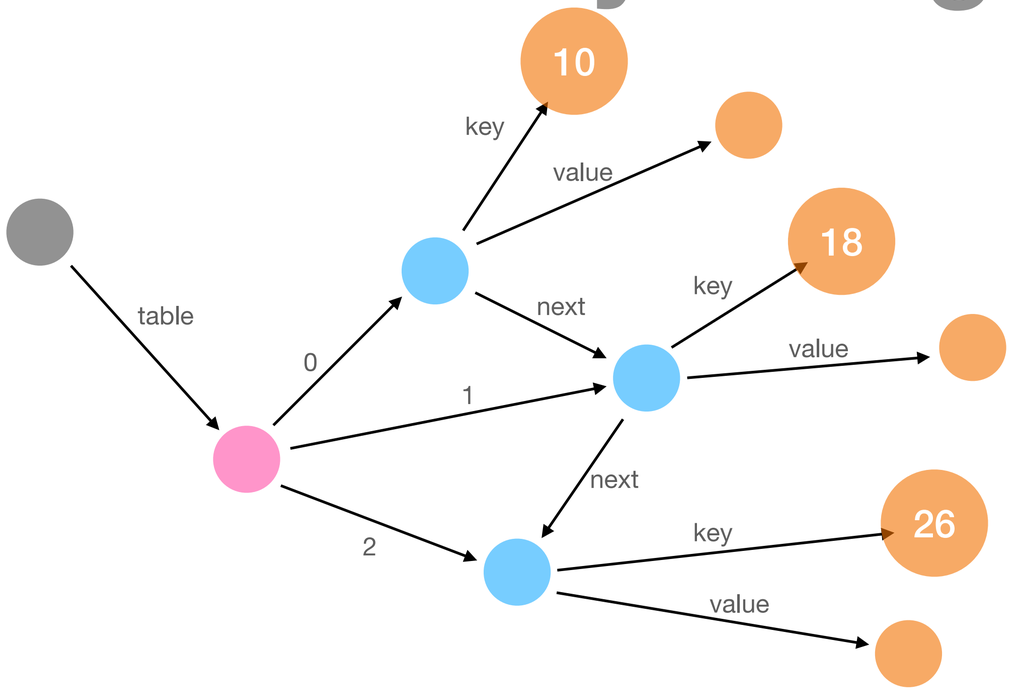


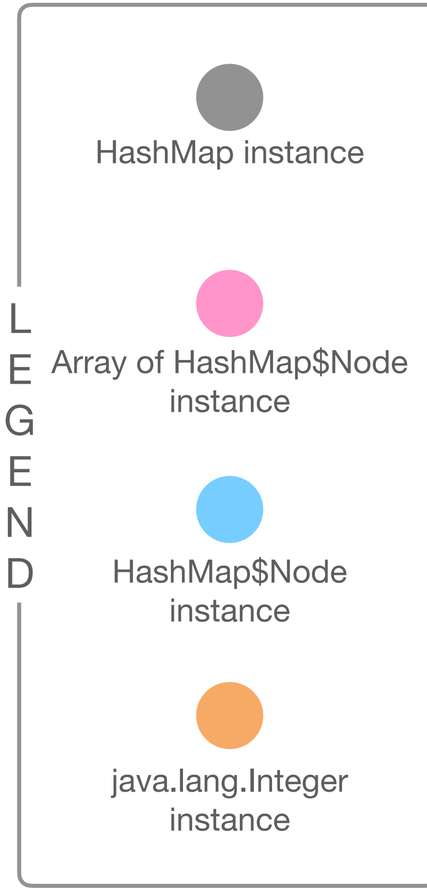



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

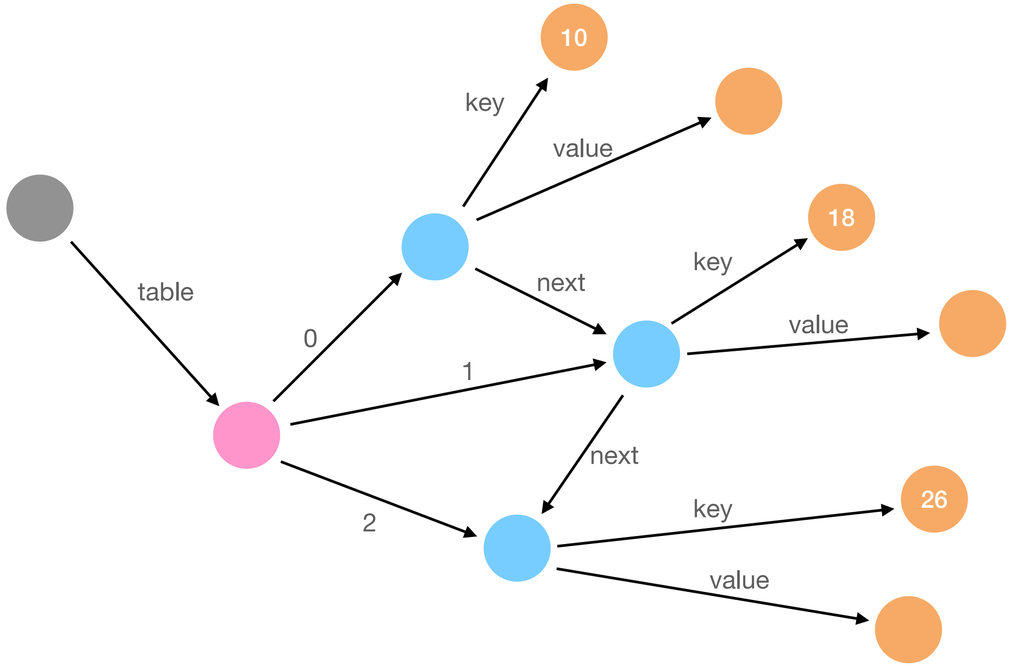


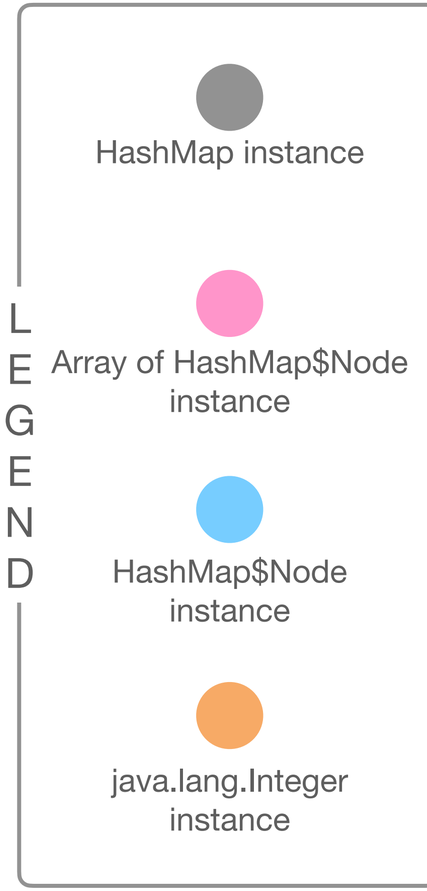



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

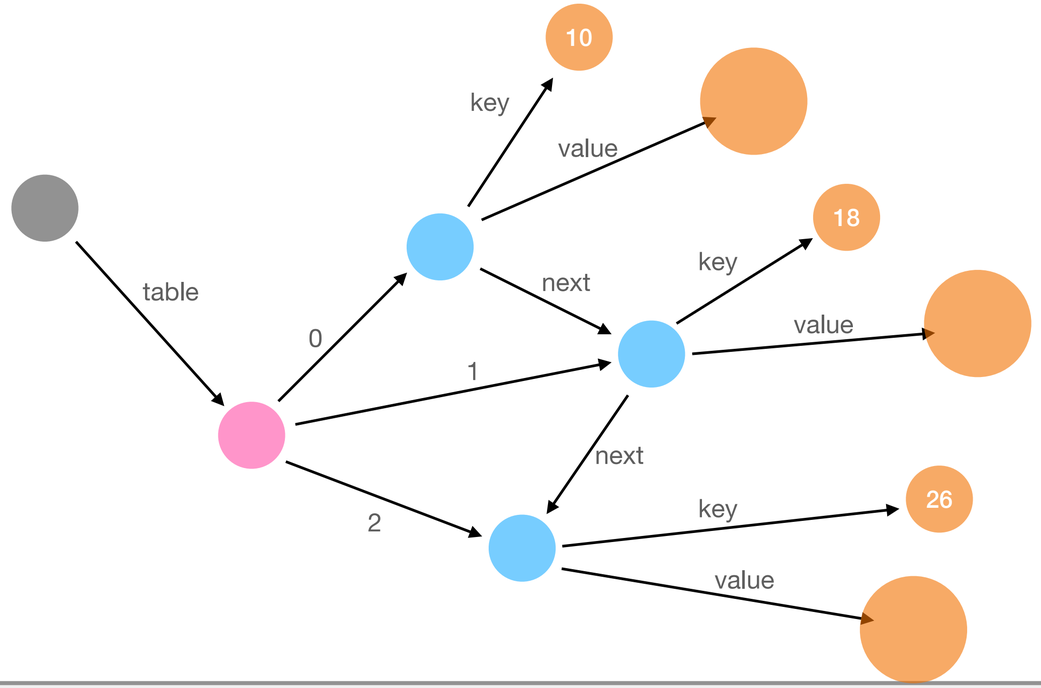


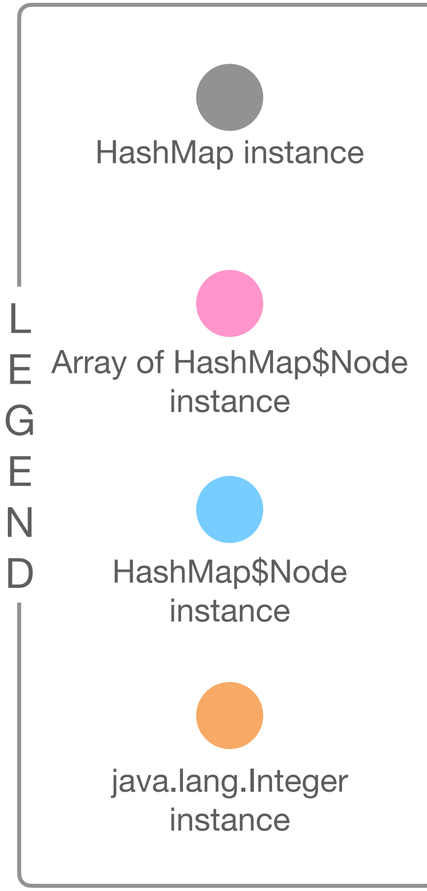



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

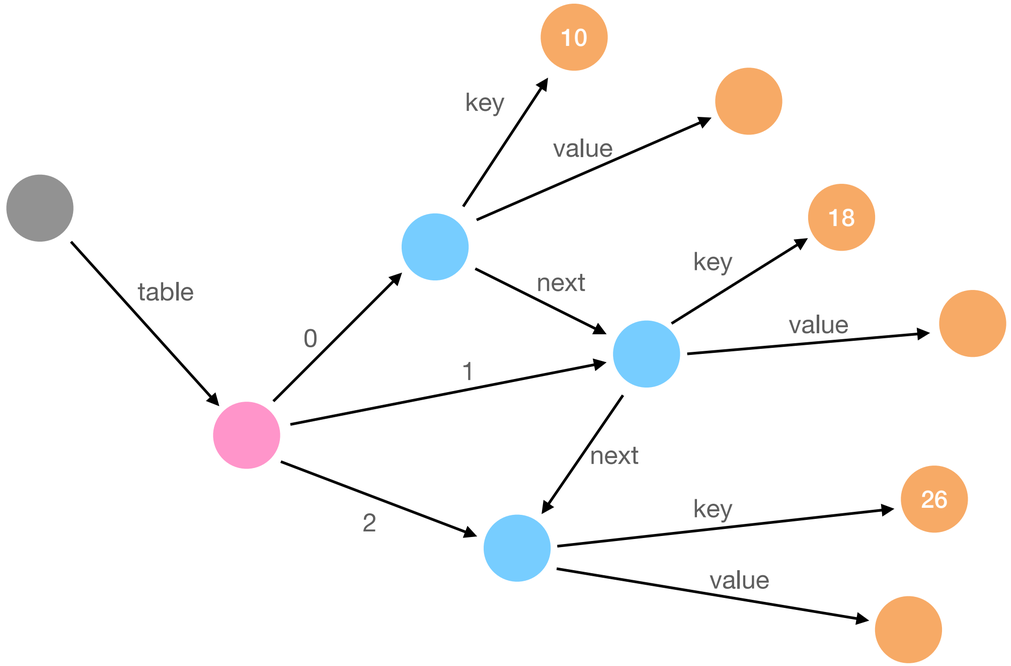


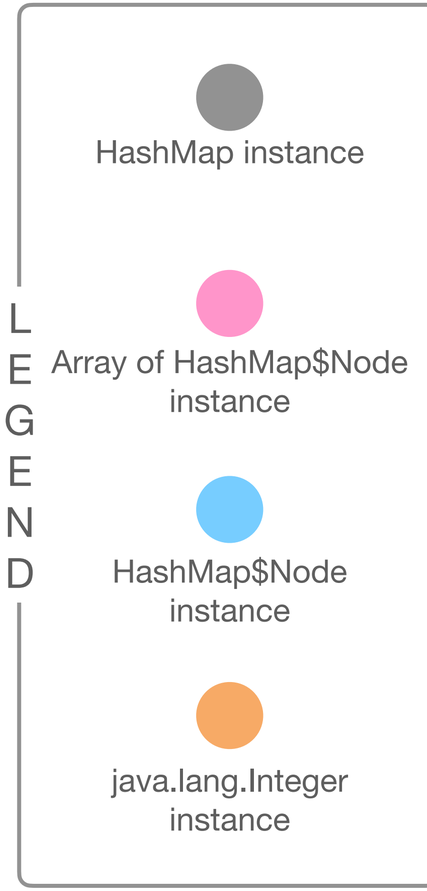



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

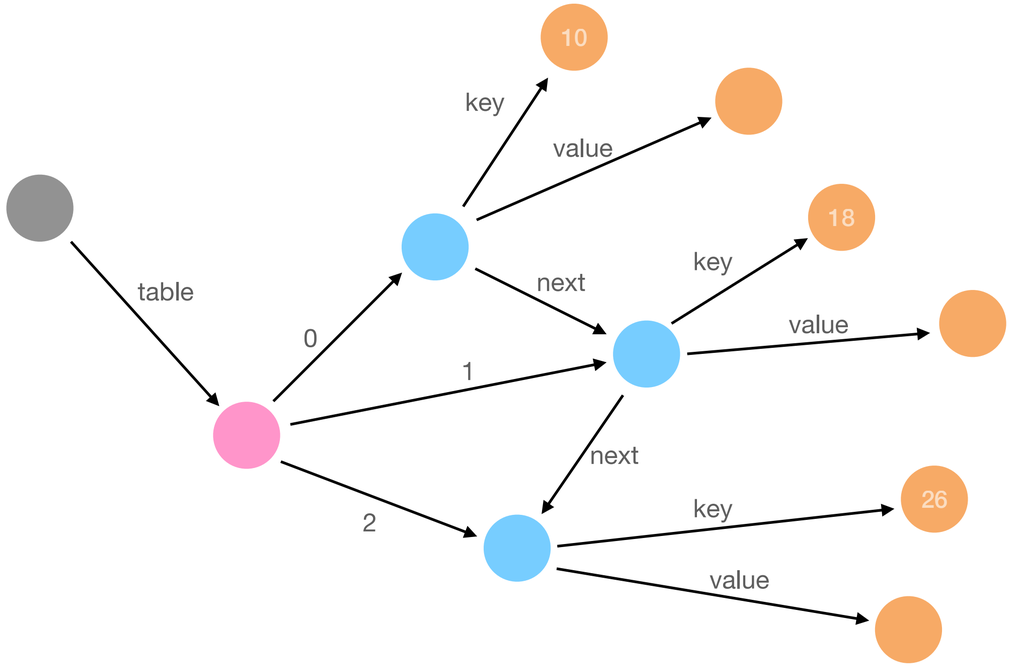


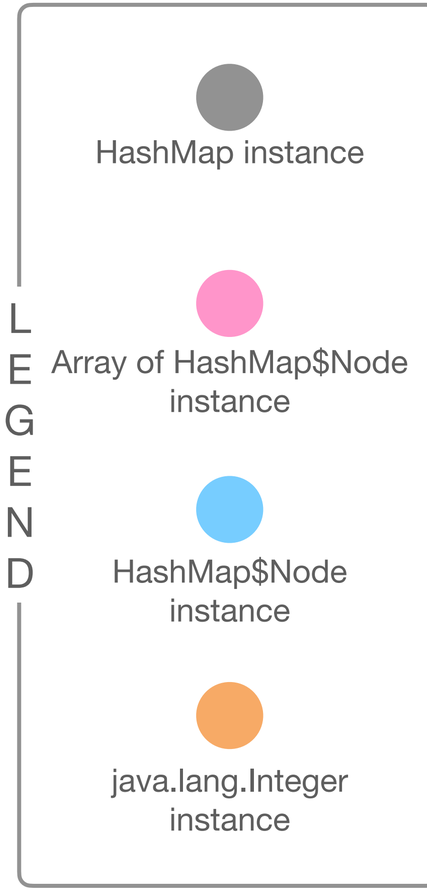



```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO

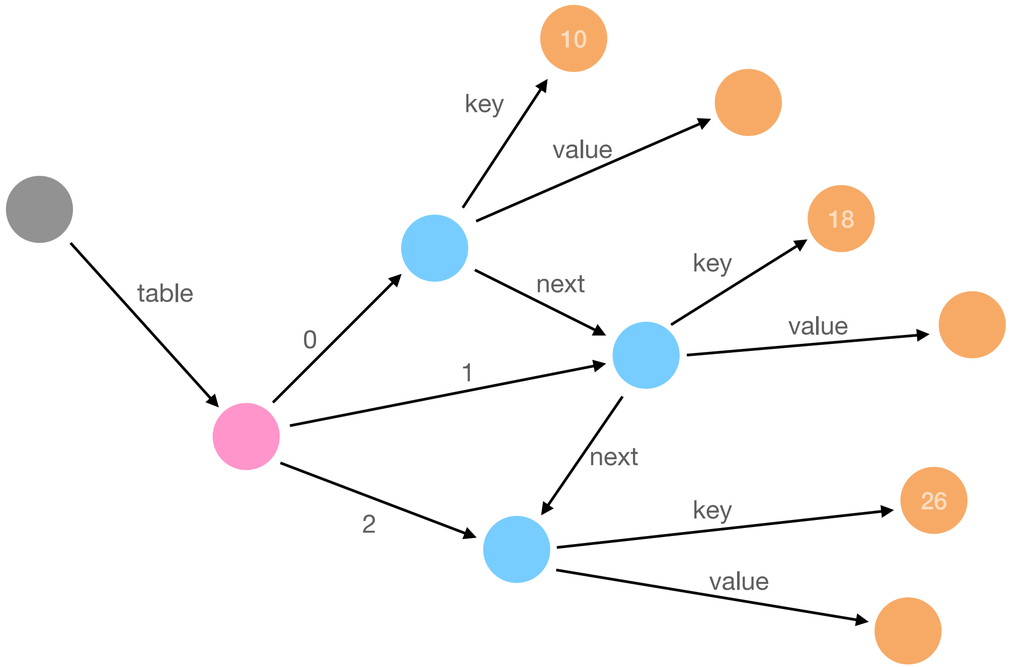





```
1.  public class HashMap<K,V> extends AbstractMap<K,V>
2.                                implements Map<K,V>,Cloneable,Serializable {
3.      transient Node<K,V>[] table;
4.      static class Node<K,V> implements Map.Entry<K,V> {
5.          final int hash; final K key; V value; Node<K,V> next;
6.      }
7.  }
```

Snippet of java.util.HashMap class definition

Example | containsKey using OGO



LEGEND

HashMap instance

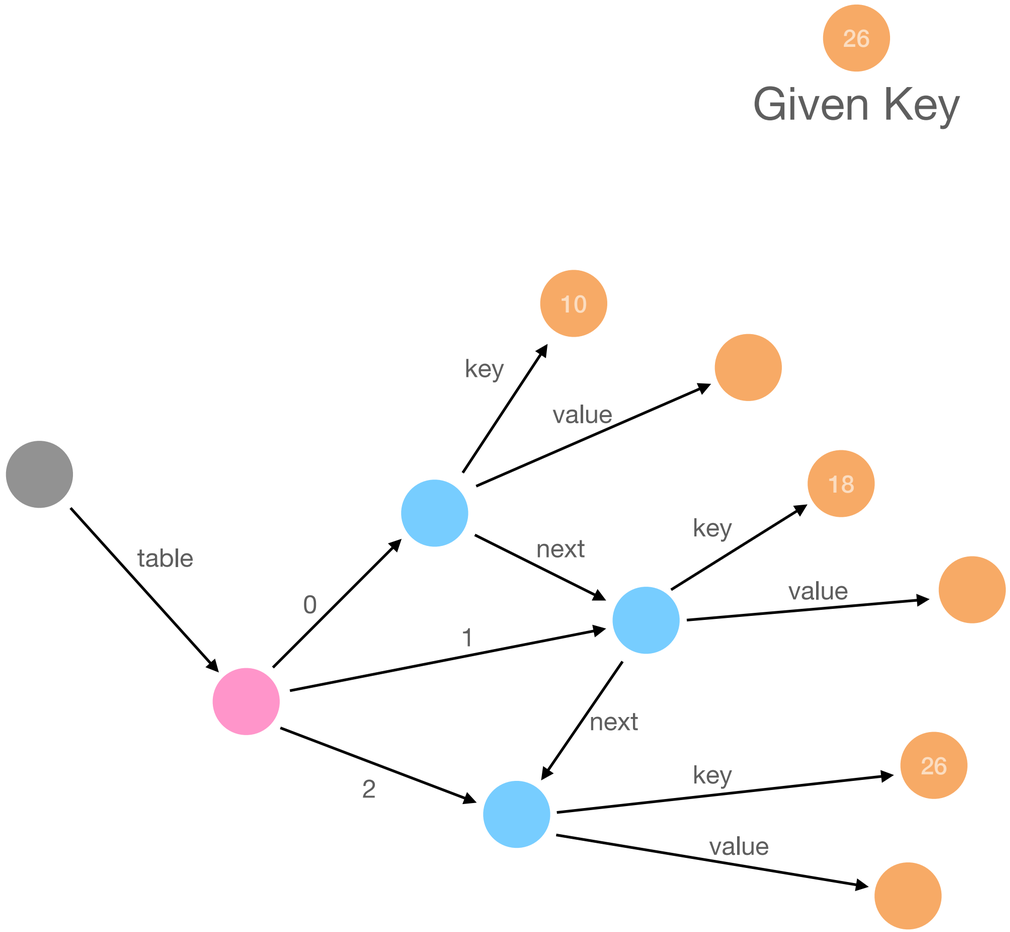
Array of HashMap$Node instance

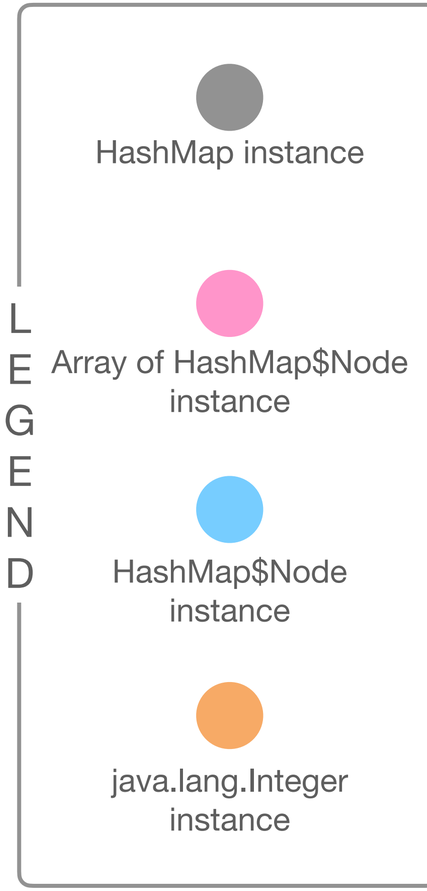
HashMap$Node instance

java.lang.Integer instance

Example | containsKey using OGO

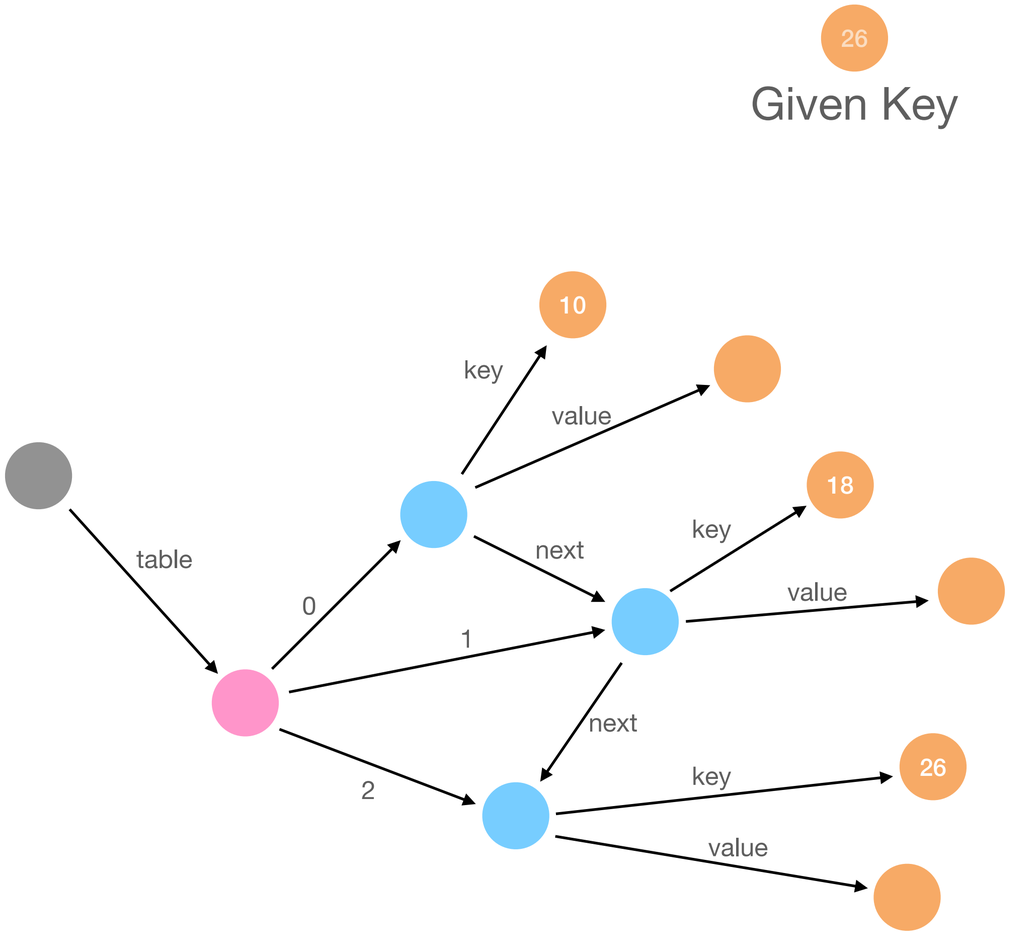
- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key

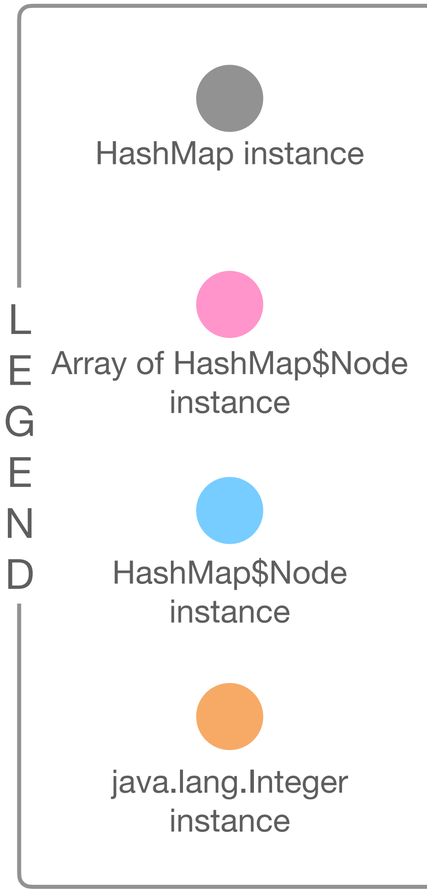




Example | containsKey using OGO

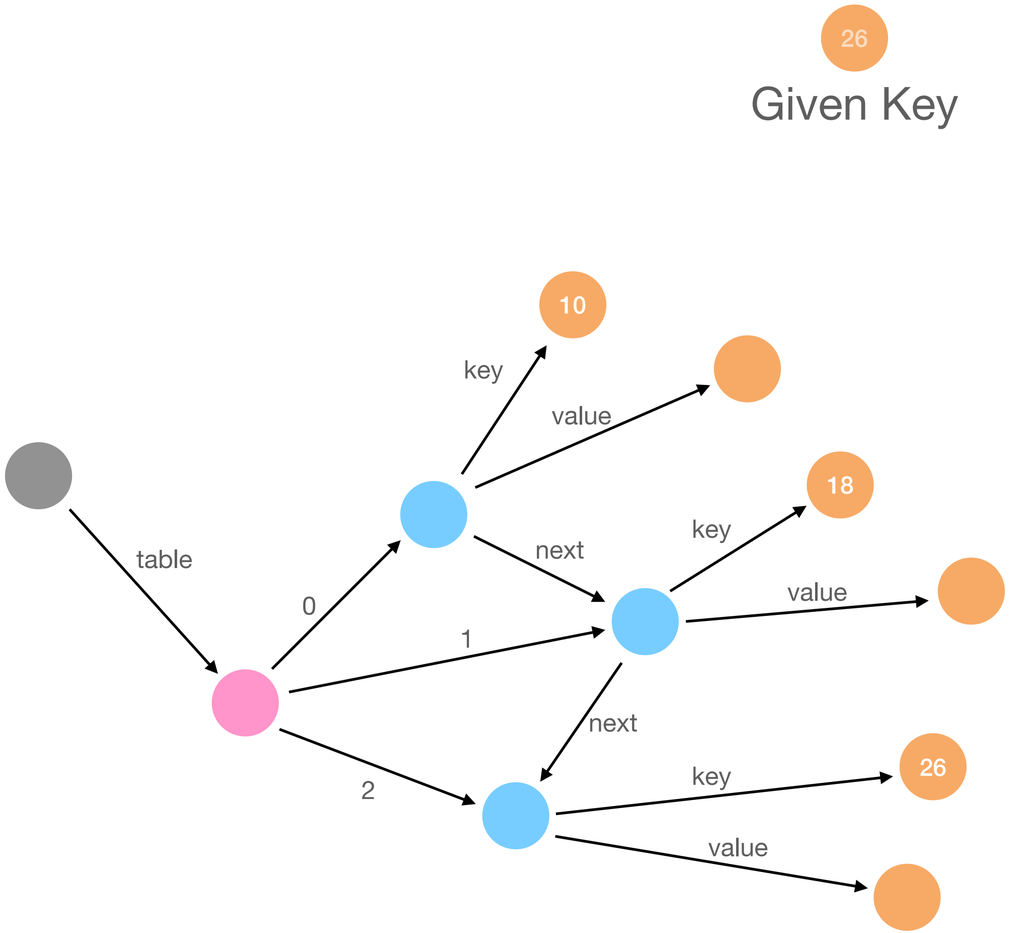
- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key



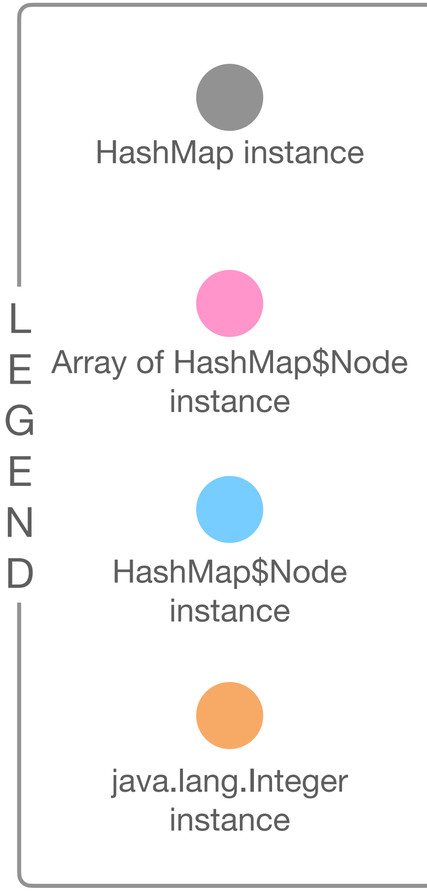


Example | containsKey using OGO

- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key

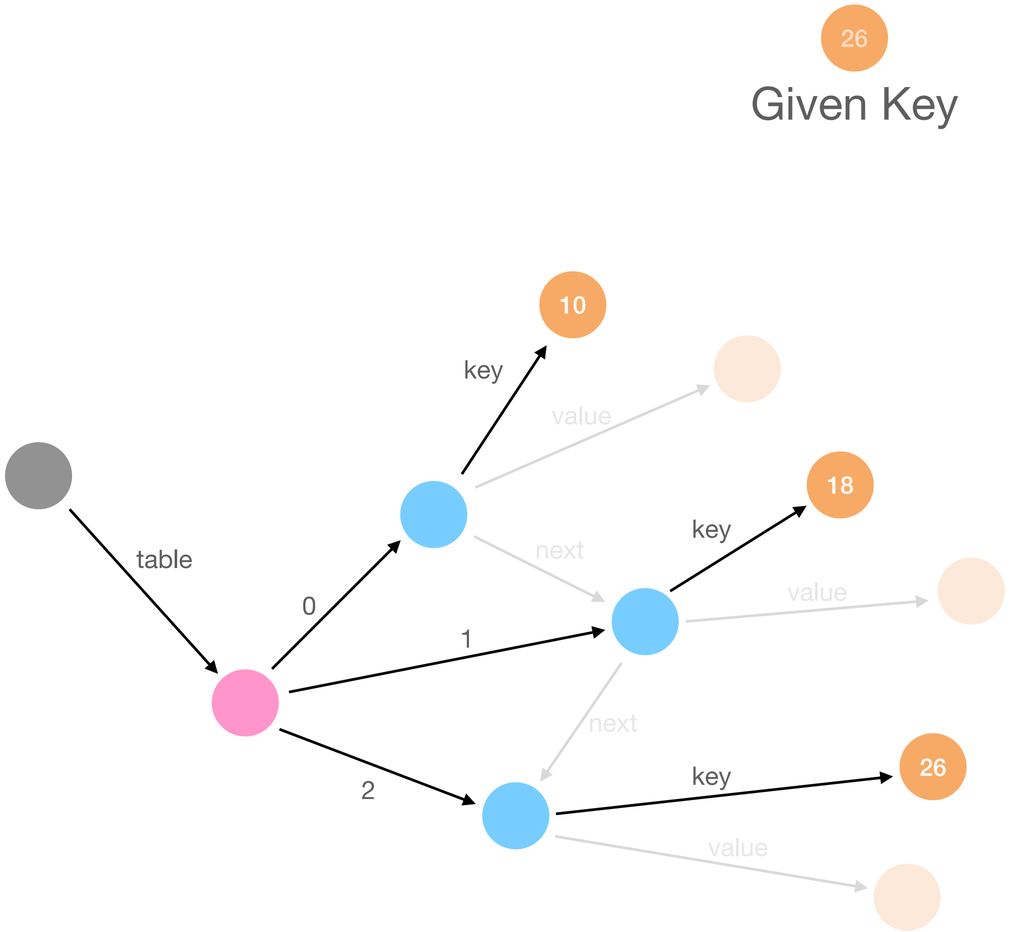



```
query(“
   MATCH  ({$1})-[:table]->()-[]->()-[:key]->(n)
   MATCH  (m {$2})
   WHERE n.`equals`(m)
   RETURN COUNT(n) <> 0”, map, givenKey
)
```



Example | containsKey using OGO

- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key




```
query(“
   MATCH  ({$1})-[:table]->()-[]->()-[:key]->(n)
   MATCH  (m {$2})
   WHERE n.`equals`(m)
   RETURN COUNT(n) <> 0”, map, givenKey
)
```

LEGEND

HashMap instance

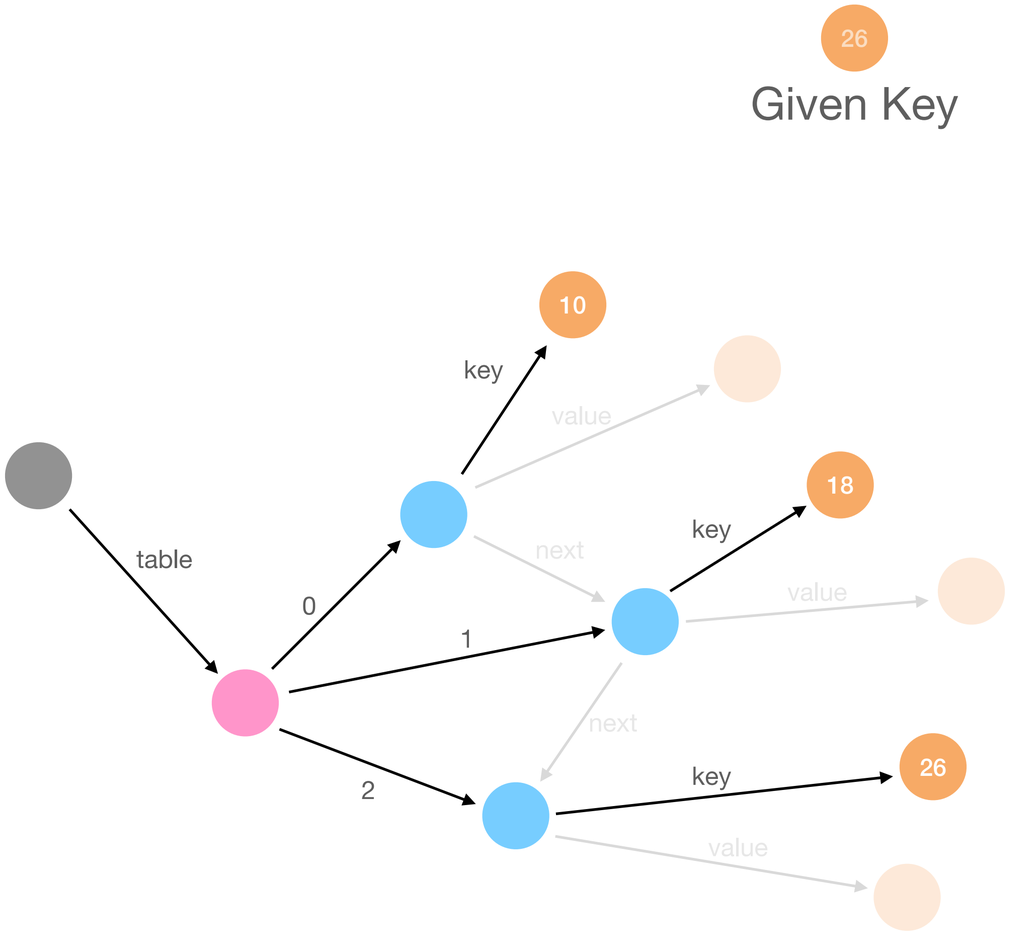
Array of HashMap$Node instance

HashMap$Node instance

java.lang.Integer instance

Example | containsKey using OGO

- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key




```
query(“
   MATCH  ({$1})-[:table]->()-[]->()-[:key]->(n)
   MATCH  (m {$2})
   WHERE n.`equals`(m)
   RETURN COUNT(n) <> 0”, map, givenKey
)
```

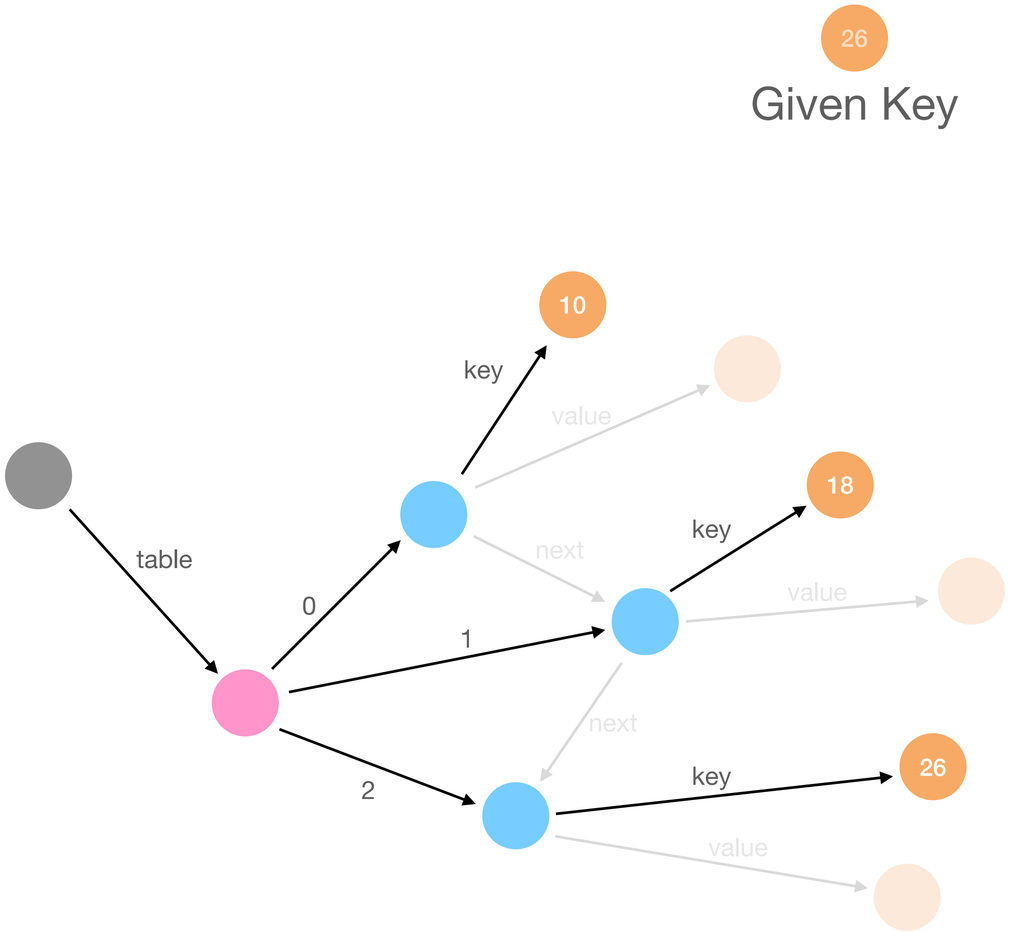
n = { 10 18 26 }

LEGEND

- HashMap instance
- Array of HashMap$Node instance
- HashMap$Node instance
- java.lang.Integer instance

Example | containsKey using OGO

- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key




```
query(“
   MATCH  ({$1})-[:table]->()-[]->()-[:key]->(n)
   MATCH  (m {$2})
   WHERE n.`equals`(m)
   RETURN COUNT(n) <> 0”, map, givenKey
)
```

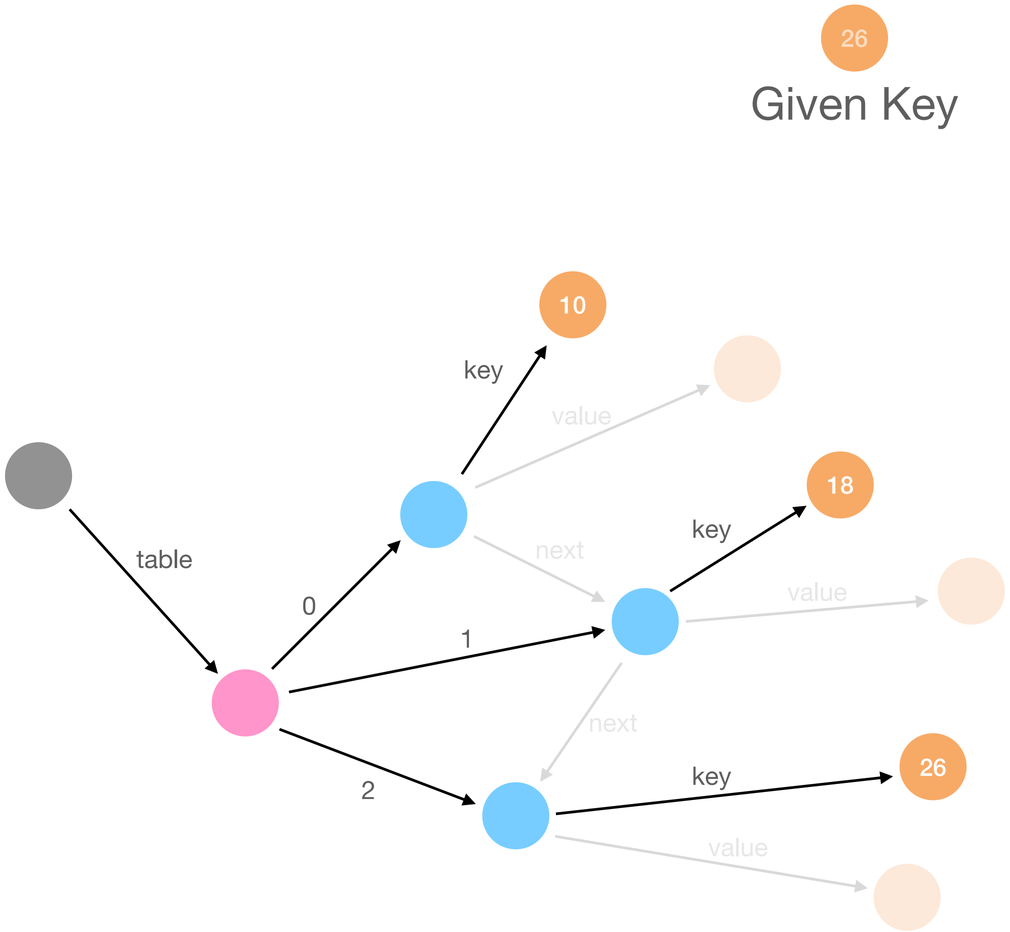
n = { 10 18 26 }

LEGEND

- HashMap instance
- Array of HashMap$Node instance
- HashMap$Node instance
- java.lang.Integer instance

Example | containsKey using OGO

- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key




```
query(“
   MATCH  ({$1})-[:table]->()-[]->()-[:key]->(n)
   MATCH  (m {$2})
   WHERE n.`equals`(m)
   RETURN COUNT(n) <> 0”, map, givenKey
)
```

n = { 10 18 26 }

m = { 26 Given Key }

LEGEND

HashMap instance

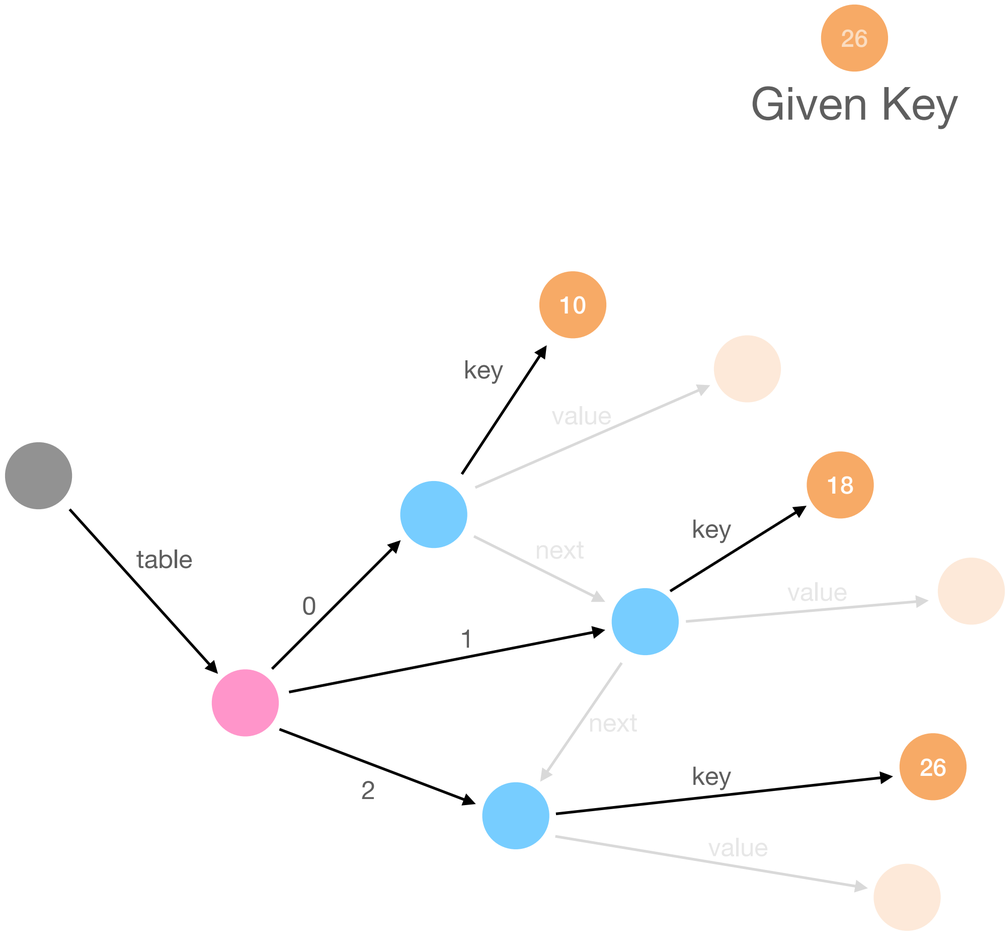
Array of HashMap$Node instance

HashMap$Node instance

java.lang.Integer instance

Example | containsKey using OGO

- The containsKey method of java.util.HashMap class checks if the hash-map contains the given key




```
query(“
   MATCH  ({$1})-[:table]->()-[]->()-[:key]->(n)
   MATCH  (m {$2})
   WHERE n.`equals`(m)
   RETURN COUNT(n) <> 0”, map, givenKey
)
```

n = { 26 }

m = { 26 Given Key }

LEGEND

HashMap instance

Array of HashMap$Node instance

HashMap$Node instance

java.lang.Integer instance

Example | containsKey using OGO

- Imperative implementation is **more** verbose than its OGO counterpart

Imperative implementation

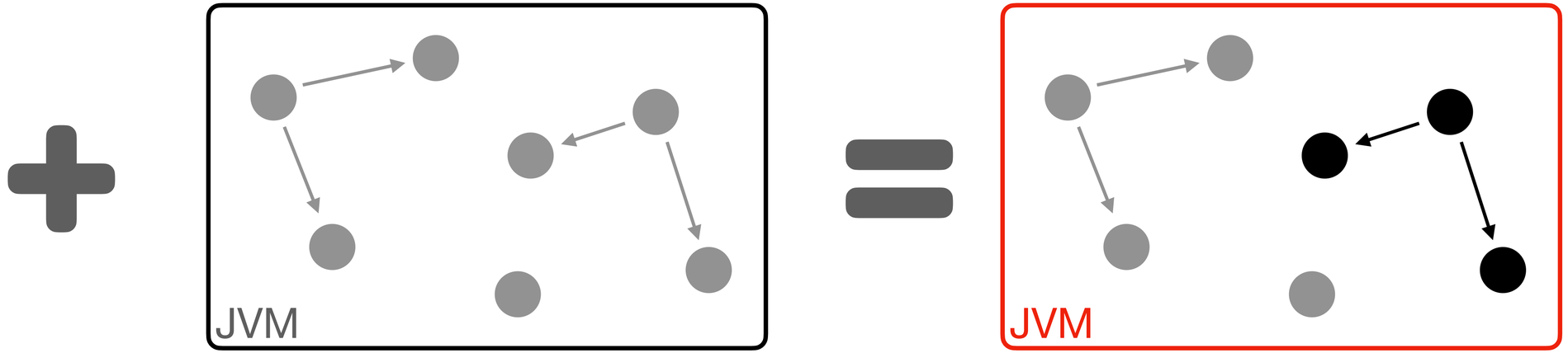
```
1.  public boolean containsKey(Object key) {
2.      return getNode(hash(key), key) != null;
3.  }
4.  final Node<K,V> getNode(int hash, Object key) {
5.      Node<K,V>[] tab; Node<K,V> first, e; int n; K k;
6.      if ((tab = table) != null && (n = tab.length) > 0 &&
7.              (first = tab[(n - 1) & hash]) != null) {
8.          if (first.hash == hash && // always check first node
9.              ((k = first.key) == key || (key != null && key.equals(k))))
10.             return first;
11.         if ((e = first.next) != null) {
12.             if (first instanceof TreeNode)
13.                 return ((TreeNode<K,V>)first).getTreeNode(hash, key);
14.             do {
15.                 if (e.hash == hash && ((k = e.key) == key ||
16.                      (key != null && key.equals(k)))) return e;
17.             } while ((e = e.next) != null);}
18.         } return null;}
```

OGO implementation

```
1.  public boolean containsKey(Object key) {
2.      return queryBool(
3.          "MATCH ({$1})-[:table]->()-[*]->()-[:key]->(n) MATCH (m {$2})"
4.         +"WHERE n.`equals`(m) RETURN COUNT(n) <> 0", this, key);
5.  }
```

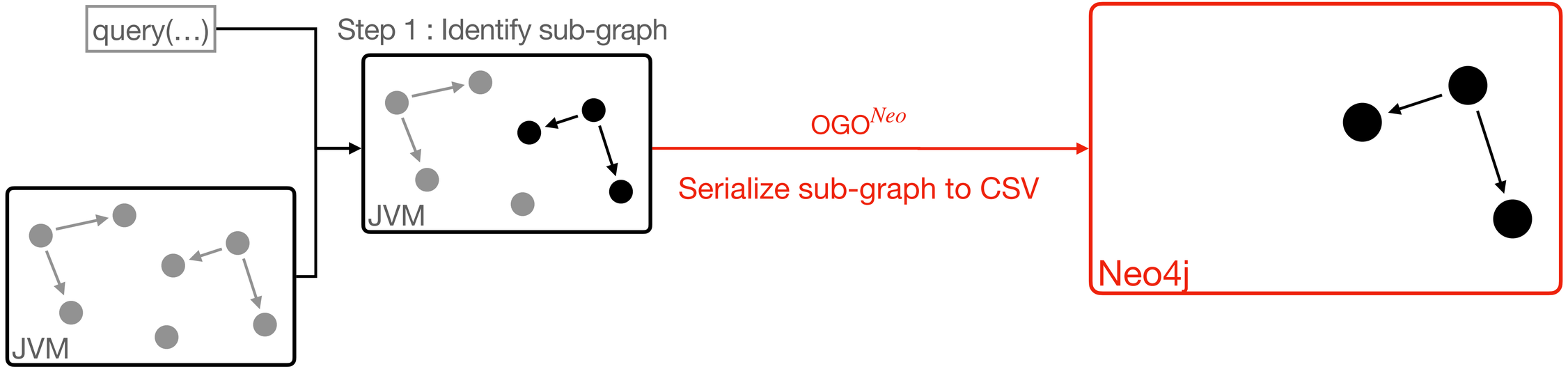
OGO Architecture

Step 1 : Identify sub-graph

query(…) 

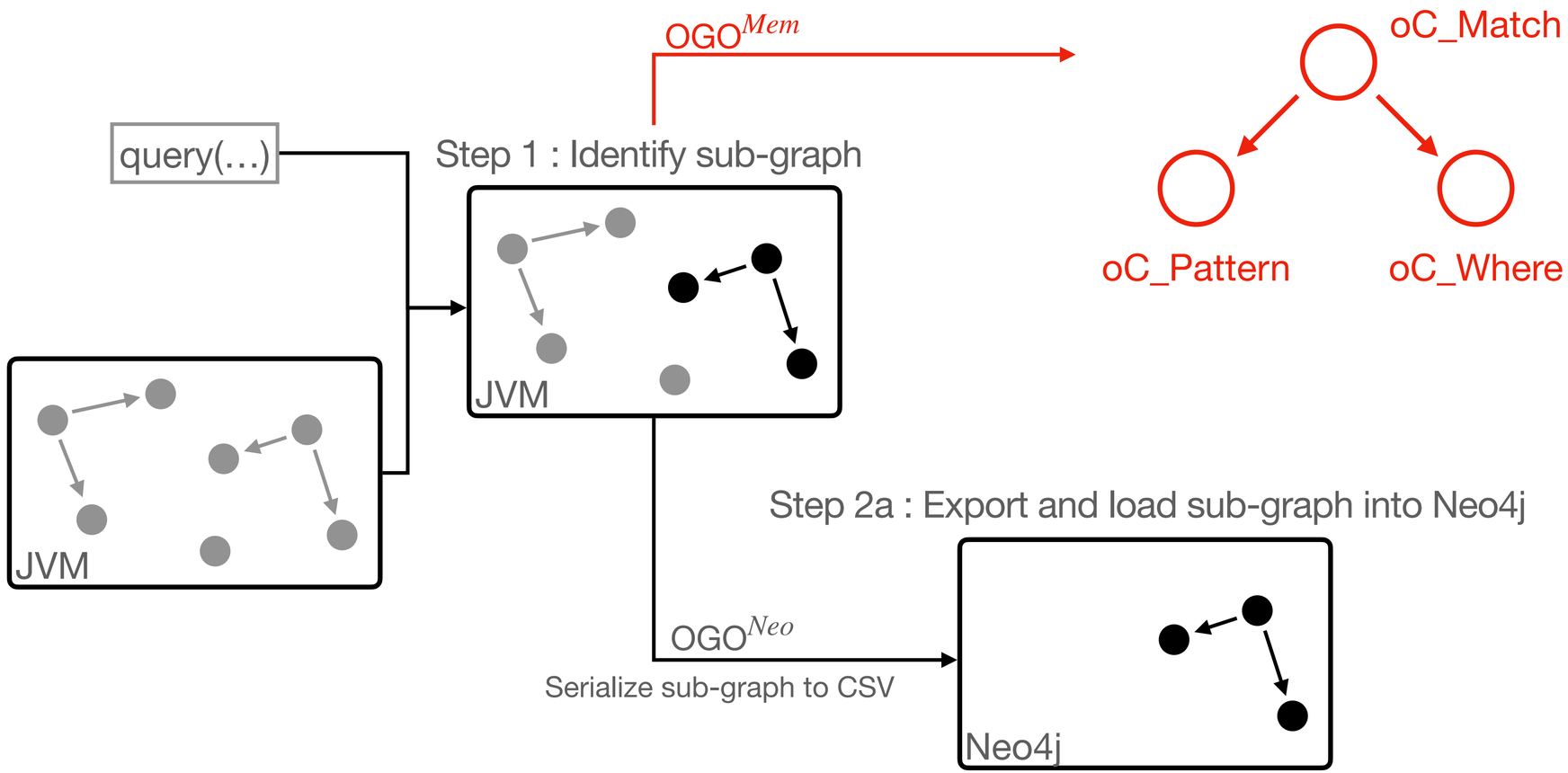
OGO Architecture

Step 2a : Export and load sub-graph into Neo4j (OGO^{Neo})



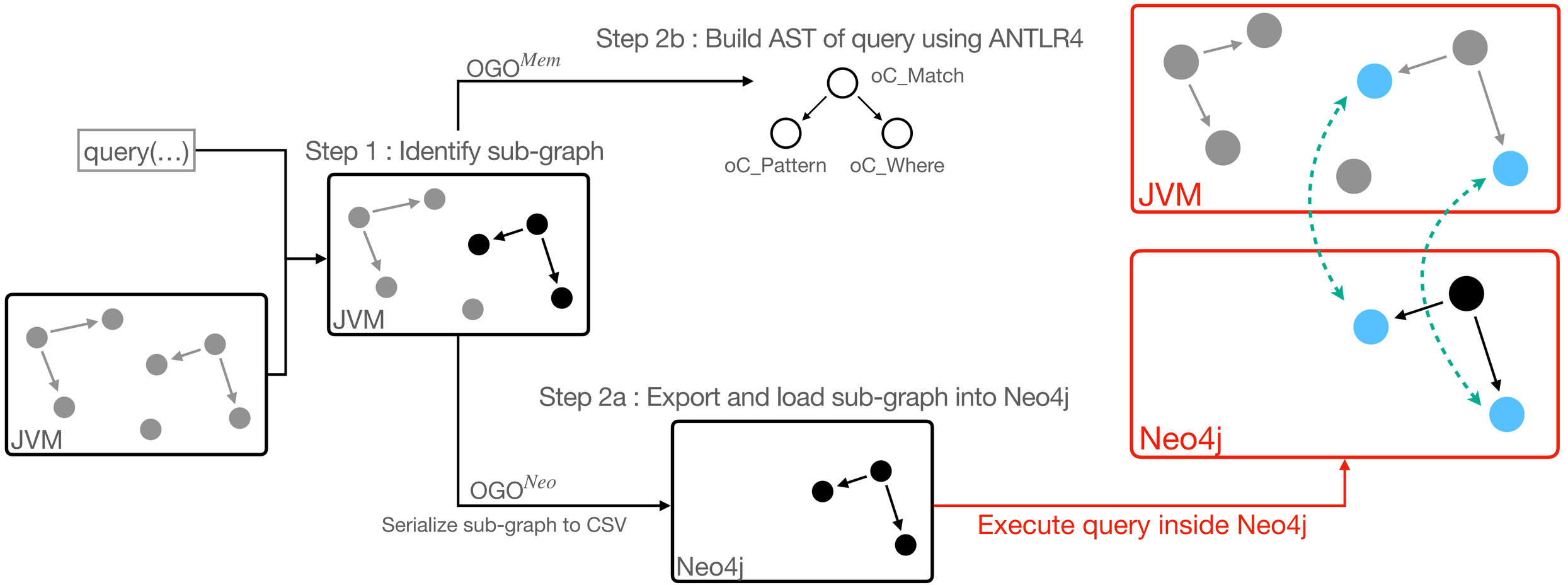
OGO Architecture

Step 2b : Build AST of query using ANTLR4 (OGO^{Mem})



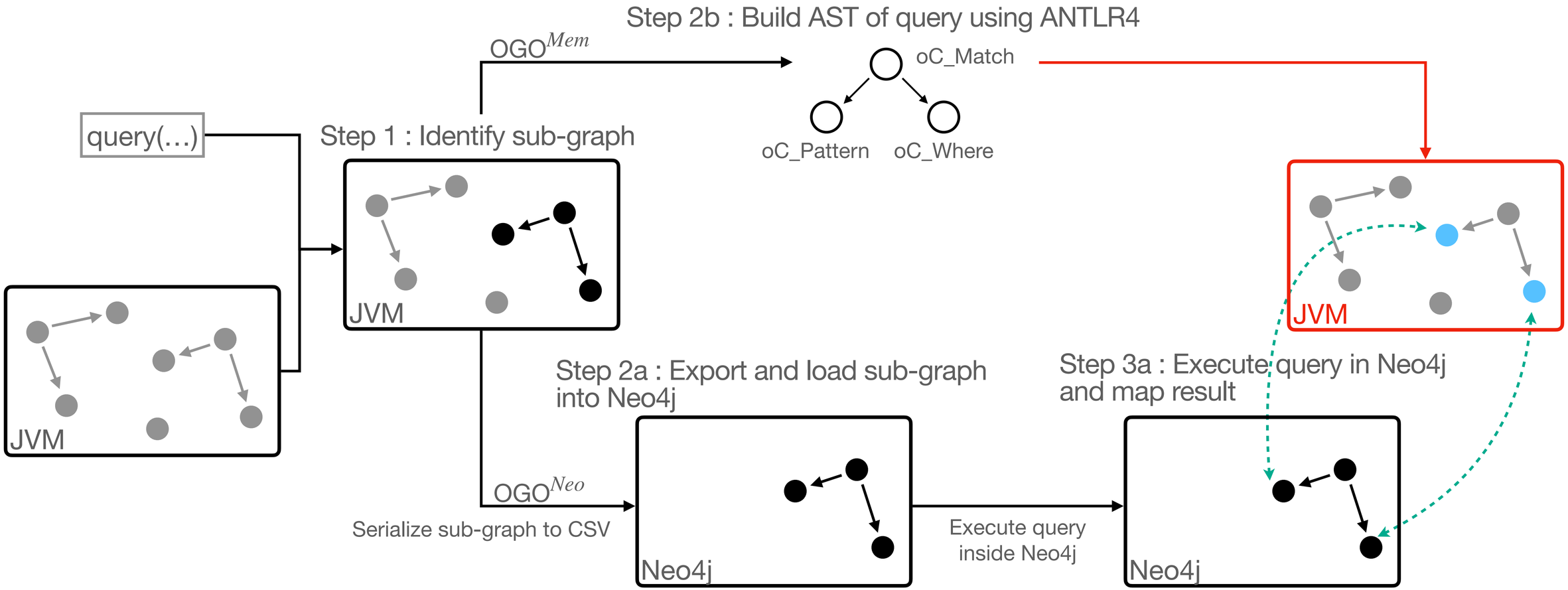
OGO Architecture

Step 3a : Execute query in Neo4j and map result



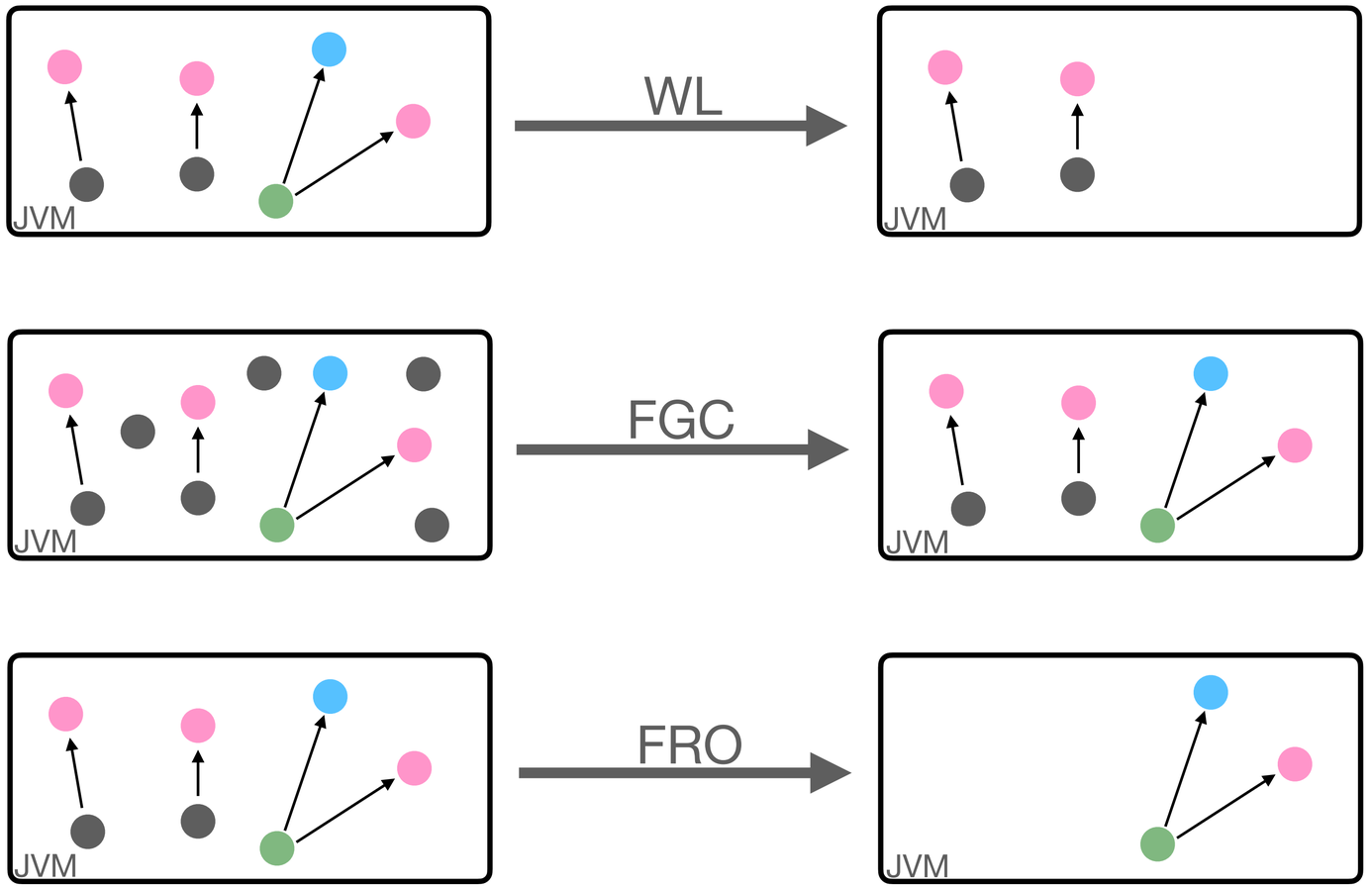
OGO Architecture

Step 3b : Execute query in-memory



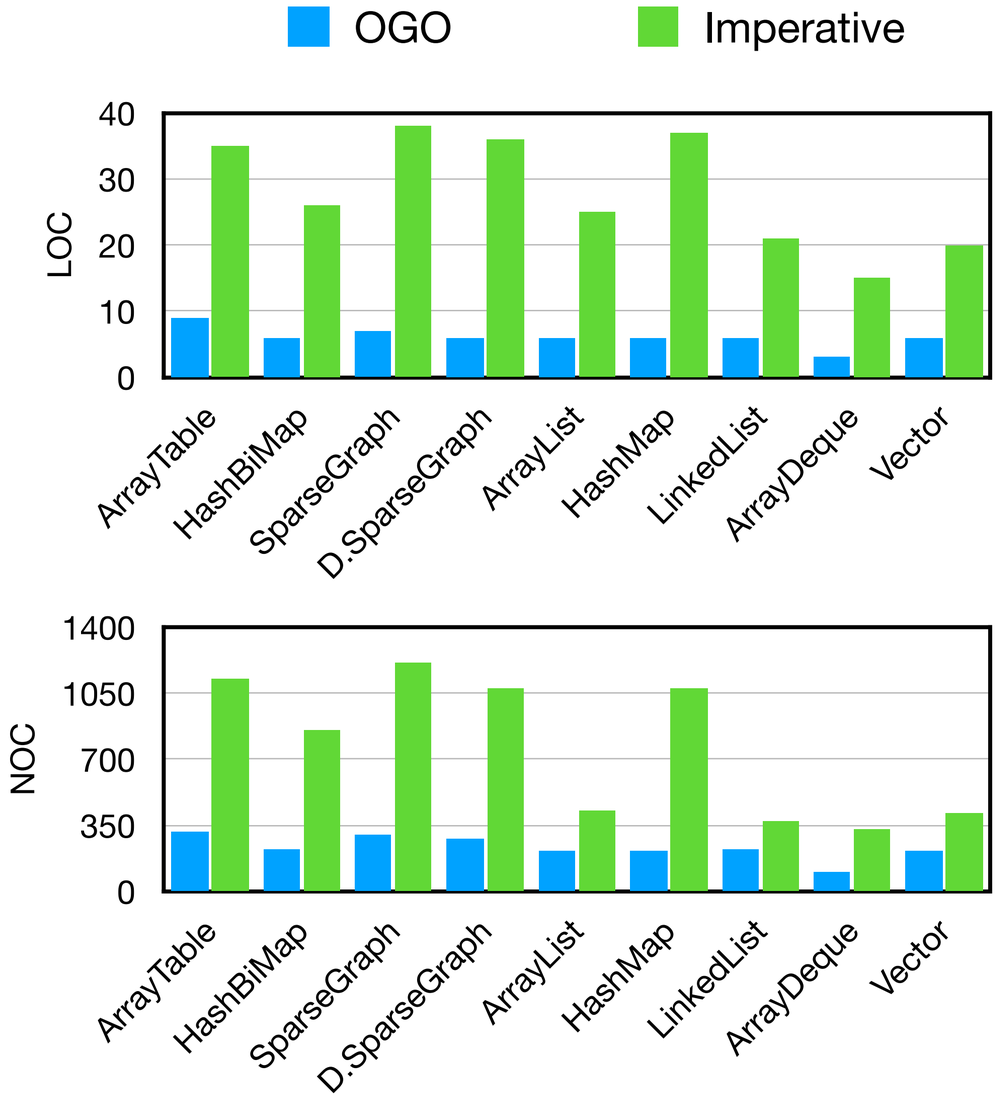
Optimizations

- We introduced 3 optimizations to improve execution speed of queries
  - **Whitelist (WL)** : definitively include instances of user specified classes into sub-graph.
  - **Force Garbage Collection (FGC) :** force garbage collection before sub-graph identification.
  - **Fix Root Object (FRO)** : limit root of sub-graph to user specified objects.



Results | Reimplementing Java Methods

- We reimplemented imperative Java methods using OGO for data structures in libraries such as:
  - Google Guava
  - Java Universal Network/Graph Framework (JUNG)
  - Java Collections Framework (JCF)
- We quantify expressivity as a measure of Lines of code (LOC) and Number of characters (NOC)
- Expressivity of code **increases** with **decrease** in LOC and NOC (Conciseness Conjecture[1])
- Imperative implementation on average requires:
  - **5x more** LOC than OGO
  - **3.6x more** NOC than OGO



1. M. Felleisen. On the expressive power of programming languages. In *Science of Computer Programming*, pages 134-151. Springer-Verlag, 1990

Results | Reimplementing methods

- We compared the end-to-end query execution times for the different modes and optimizations of OGO
- We see that OGO^{Mem} with **+WL+FRO** performs best and on par with **+WL+FRO+FGC** with an average execution time of **241ms**

Class	OGO^{Neo}				OGO^{Mem}			
	+WL				+WL			
	-	+FRO	+FGC	+FRO+FGC	-	+FRO	+FGC	+FRO+FGC
Vector	1125	794	1254	823	873	316	980	290
U.SparseGraph	7808	2287	24986	8499	1032	186	1045	293
HashBiMap	2470	856	2635	782	519	196	526	179
HashMap	2693	1771	2997	1529	1166	315	1326	289
SparseGraph	13533	2954	16945	2666	1363	188	1444	167
ArrayDeque	1563	1139	1442	1047	1151	320	1104	292
ArrayList	1391	1152	1476	1131	1080	312	1128	292
ArrayTable	5278	1313	5786	1332	624	193	652	178
LinkedList	838	354	3101	827	492	190	638	257
D.SparseGraph	3371	3160	10549	3240	1107	188	1215	177
Avg.	4007	1578	7117	2188	941	240	1006	241

WL : Whitelist, **FRO** : Fix root objects, **FGC** : Force garbage collection

Results | Reimplementing methods

- We compared the end-to-end query execution times for the different modes and optimizations of OGO
- We see that OGO^{Mem} with **+WL+FRO** performs best and on par with **+WL+FRO+FGC** with an average execution time of **241ms**

Class	OGO^{Neo}				OGO^{Mem}			
	+WL				+WL			
	-	+FRO	+FGC	+FRO+FGC	-	+FRO	+FGC	+FRO+FGC
Vector	1125	794	1254	823	873	316	980	290
U.SparseGraph	7808	2287	24986	8499	1032	186	1045	293
HashBiMap	2470	856	2635	782	519	196	526	179
HashMap	2693	1771	2997	1529	1166	315	1326	289
SparseGraph	13533	2954	16945	2666	1363	188	1444	167
ArrayDeque	1563	1139	1442	1047	1151	320	1104	292
ArrayList	1391	1152	1476	1131	1080	312	1128	292
ArrayTable	5278	1313	5786	1332	624	193	652	178
LinkedList	838	354	3101	827	492	190	638	257
D.SparseGraph	3371	3160	10549	3240	1107	188	1215	177
Avg.	4007	1578	7117	2188	941	240	1006	241

WL : Whitelist, **FRO** : Fix root objects, **FGC** : Force garbage collection

Conclusions

- We introduced a new view of the runtime state of a program as an object graph that can be queried like a graph database

Contact : auditt@utexas.edu

Conclusions

- We introduced a new view of the runtime state of a program as an object graph that can be queried like a graph database
- We introduced OGO which allows programming the heap with declarative queries

Contact : auditt@utexas.edu

Conclusions

- We introduced a new view of the runtime state of a program as an object graph that can be queried like a graph database
- We introduced OGO which allows programming the heap with declarative queries
- We introduced two prototypes of OGO and demonstrated that the in-memory execution engine is faster

Contact : auditt@utexas.edu

Conclusions

- We introduced a new view of the runtime state of a program as an object graph that can be queried like a graph database
- We introduced OGO which allows programming the heap with declarative queries
- We introduced two prototypes of OGO and demonstrated that the in-memory execution engine is faster
- We showed that methods implemented using OGO is more expressive than their imperative counterparts

Contact : auditt@utexas.edu

Conclusions

- We introduced a new view of the runtime state of a program as an object graph that can be queried like a graph database
- We introduced OGO which allows programming the heap with declarative queries
- We introduced two prototypes of OGO and demonstrated that the in-memory execution engine is faster
- We showed that methods implemented using OGO is more expressive than their imperative counterparts
- We demonstrated the robustness and versatility of OGO by reimplementing assertions and methods in large scale open-source projects

Contact : auditt@utexas.edu

Conclusions

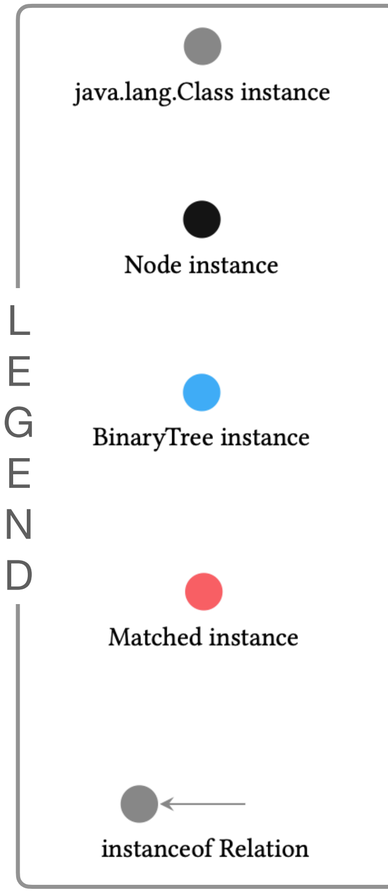
- We introduced a new view of the runtime state of a program as an object graph that can be queried like a graph database
- We introduced OGO which allows programming the heap with declarative queries
- We introduced two prototypes of OGO and demonstrated that the in-memory execution engine is faster
- We showed that methods implemented using OGO is more expressive than their imperative counterparts
- We demonstrated the robustness and versatility of OGO by reimplementing assertions and methods in large scale open-source projects
- A lot more in the paper (API design, extensive evaluation, formalization)

Contact : auditt@utexas.edu

Supplementary

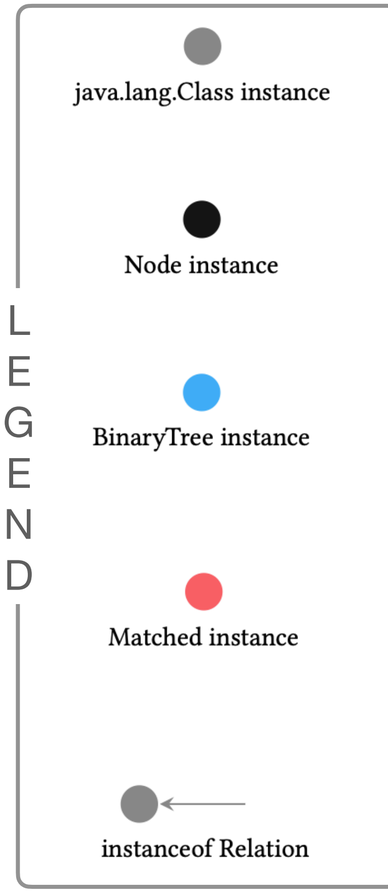
Example

```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

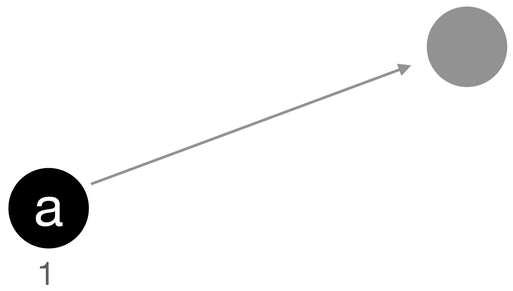


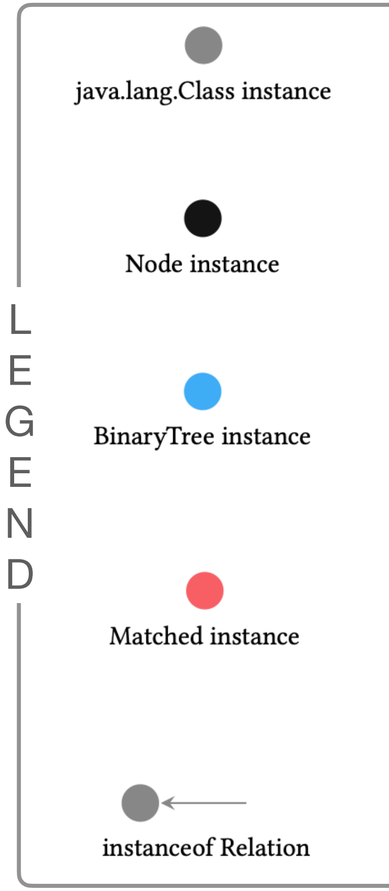
Example

```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```



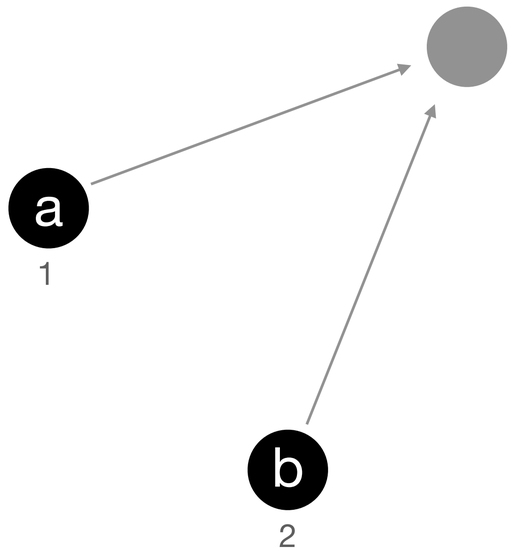
Example

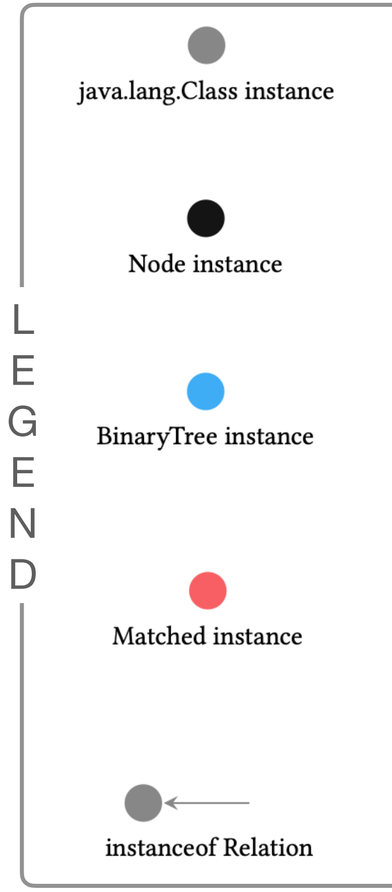





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

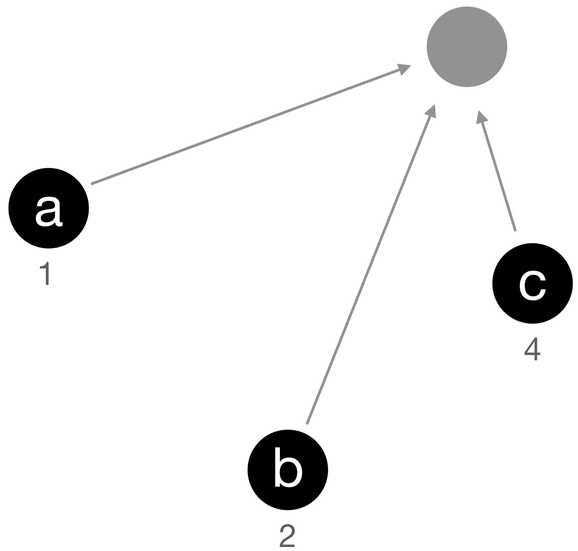
Example

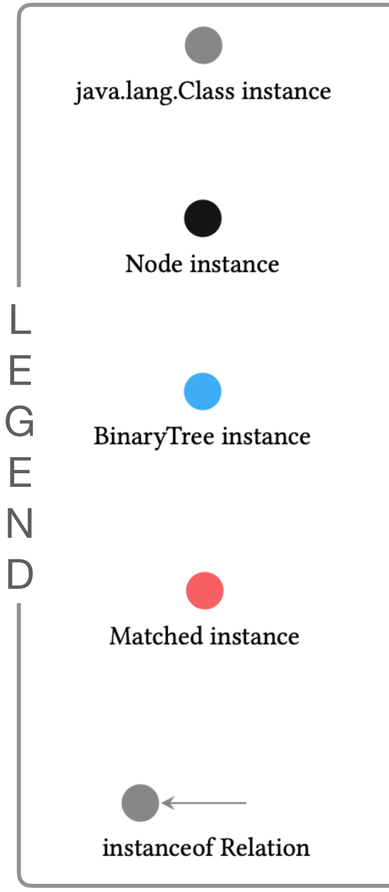





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

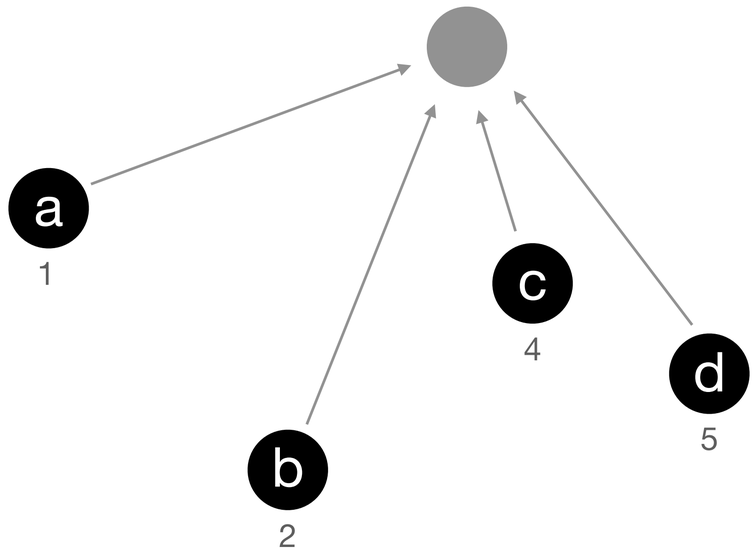
Example

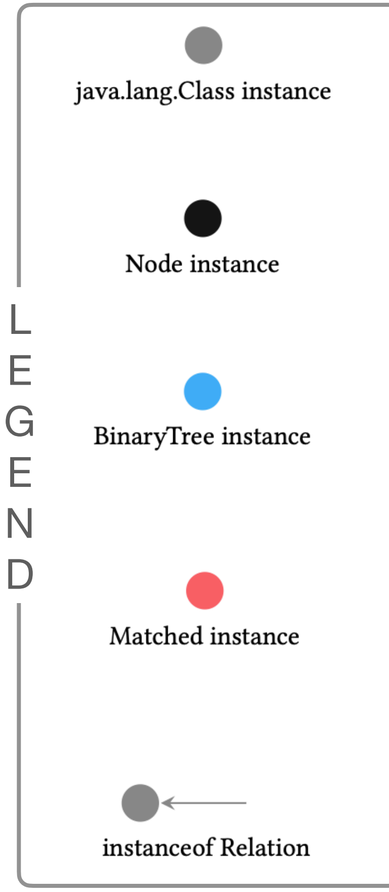





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

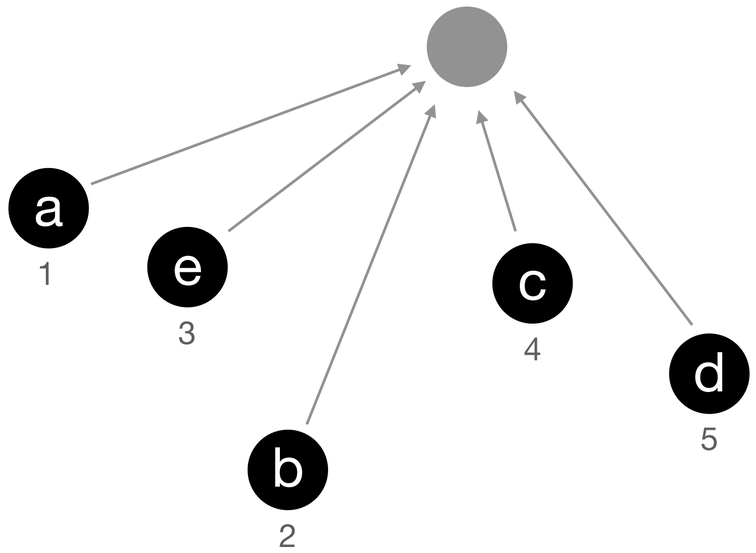
Example

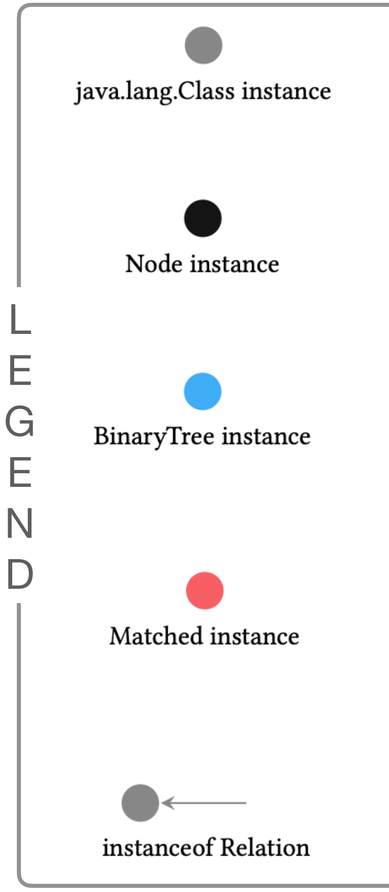





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

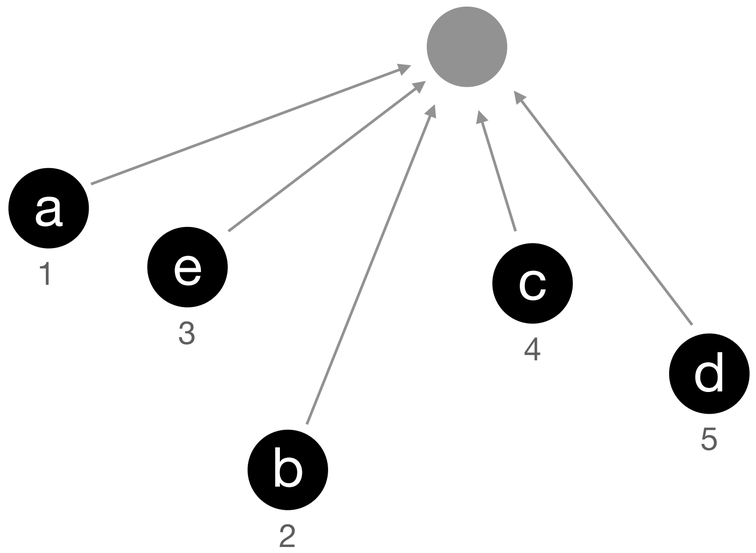
Example

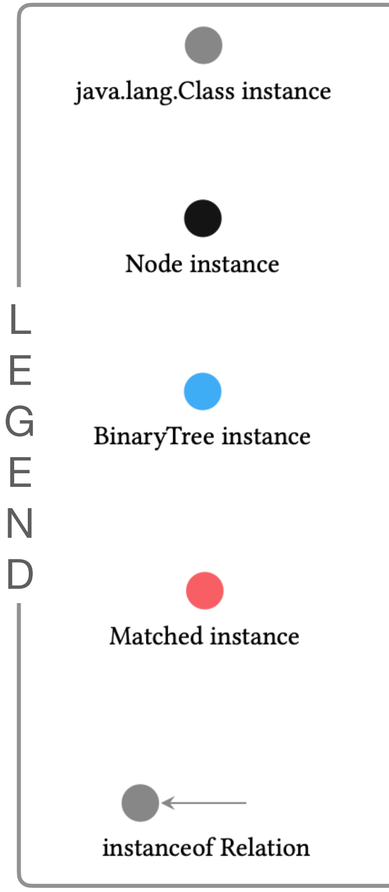





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

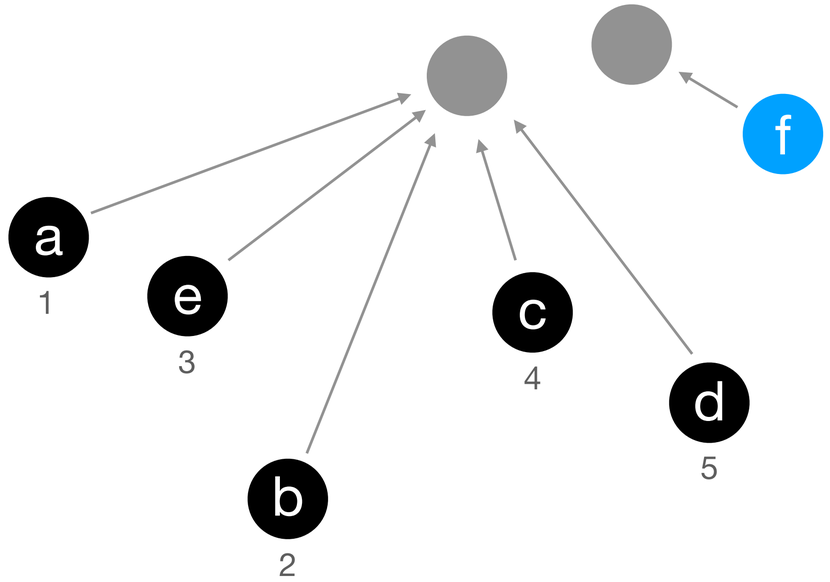
Example

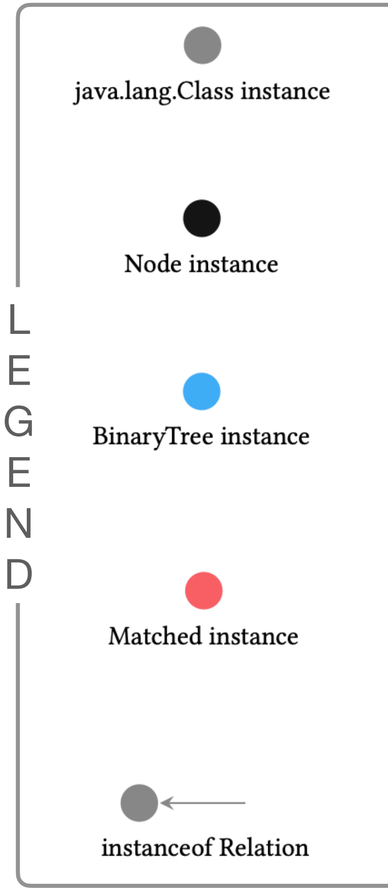





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

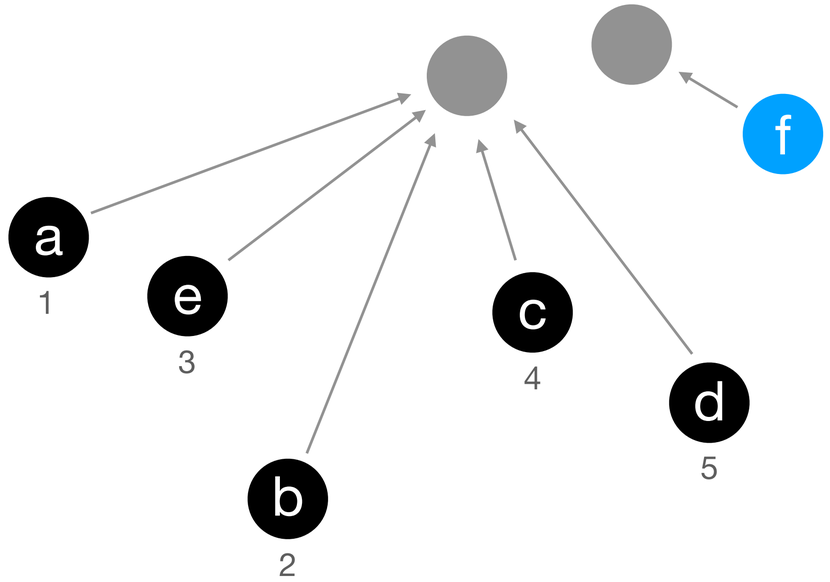
Example

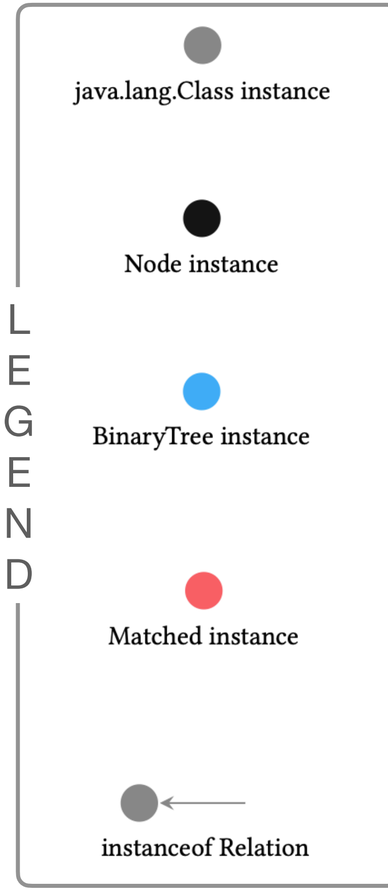





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

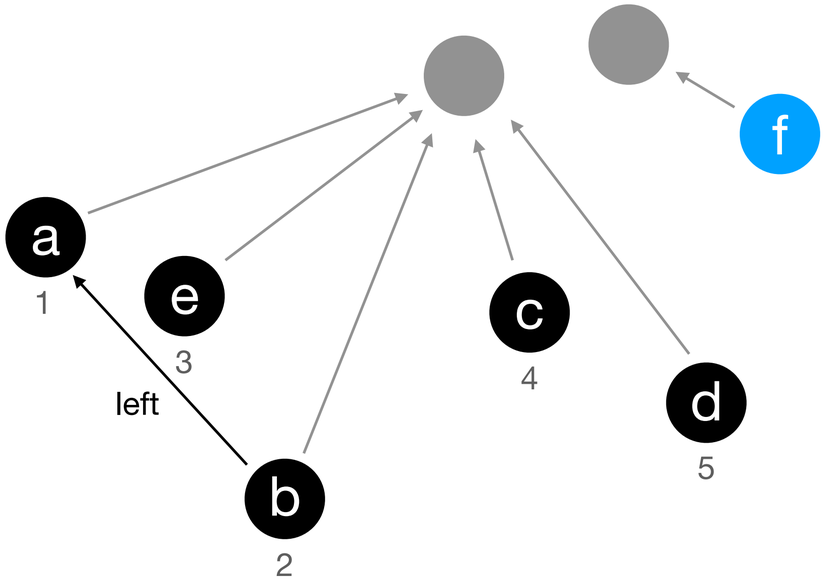
Example

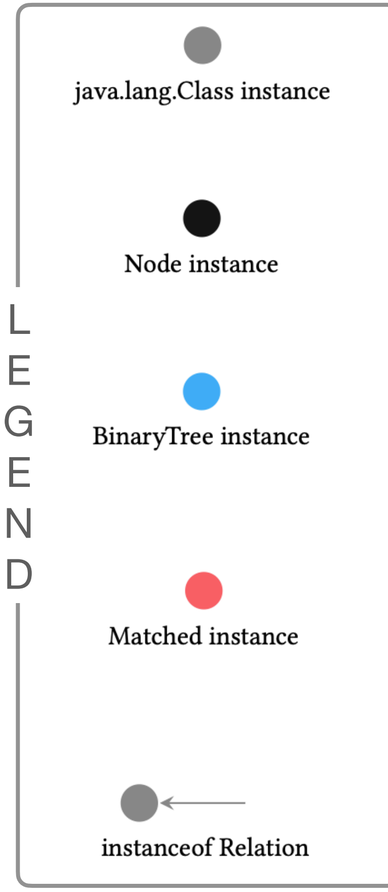





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

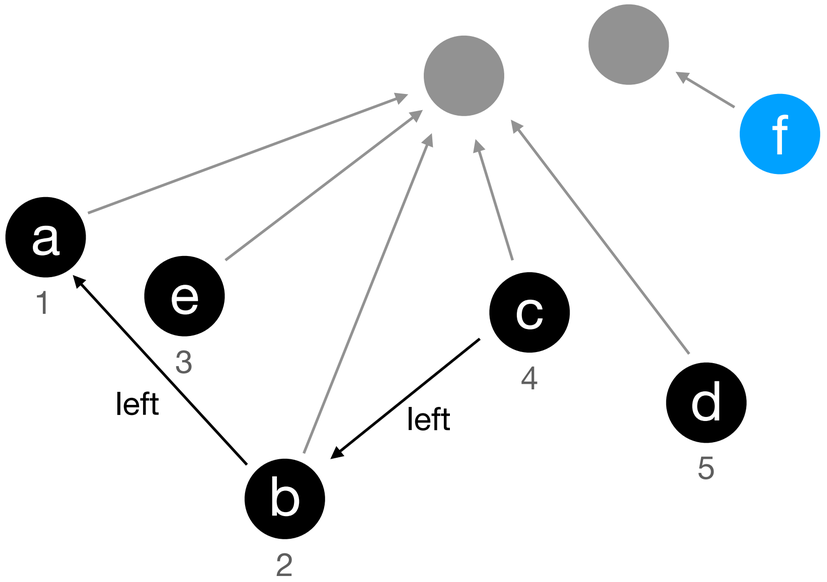
Example

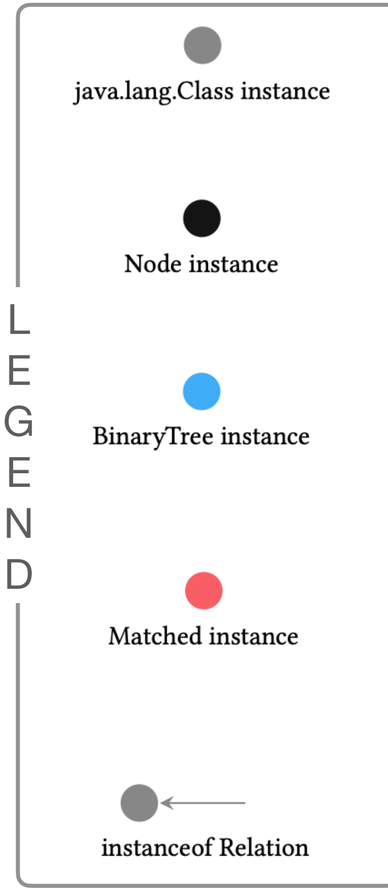





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

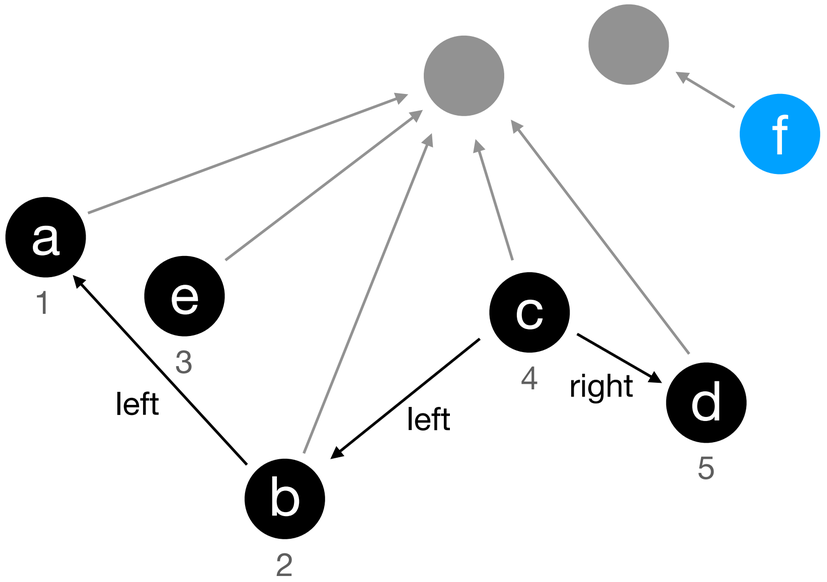
Example

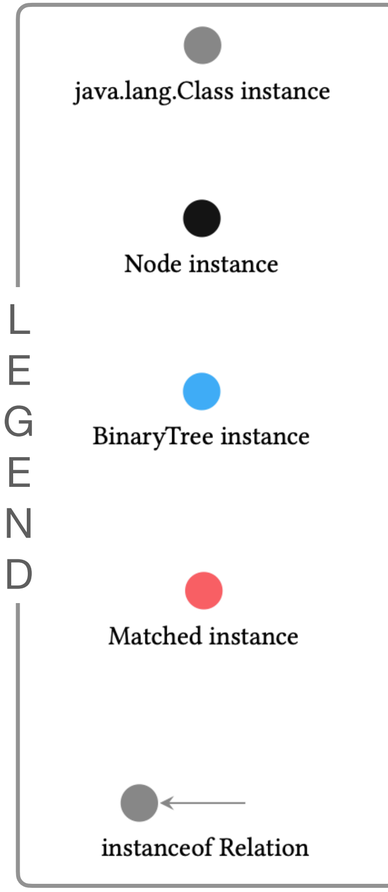





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

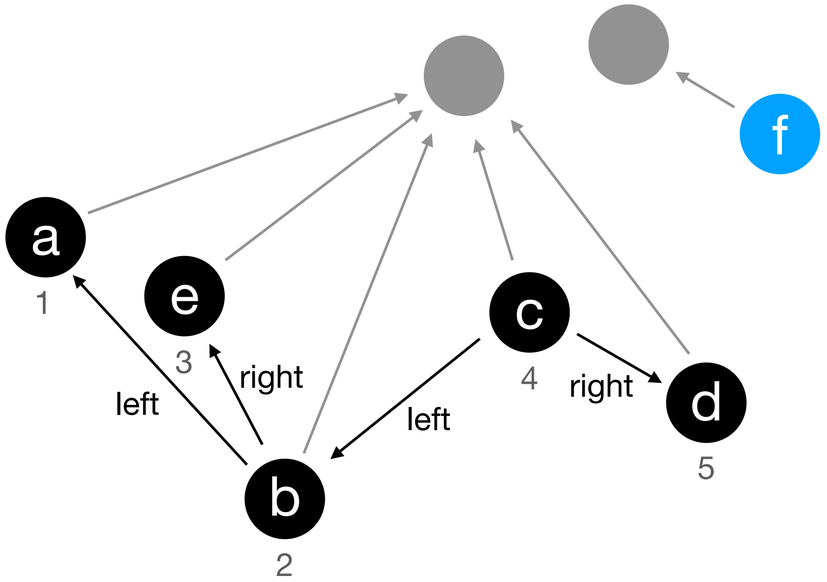
Example

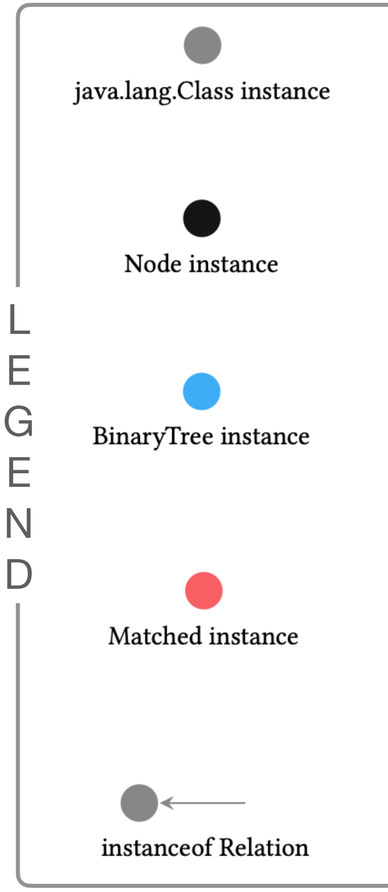





```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

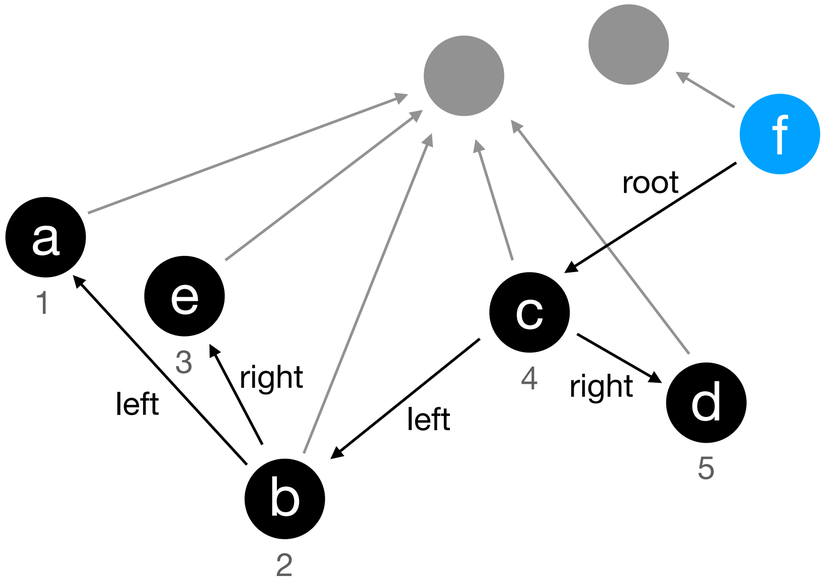
Example



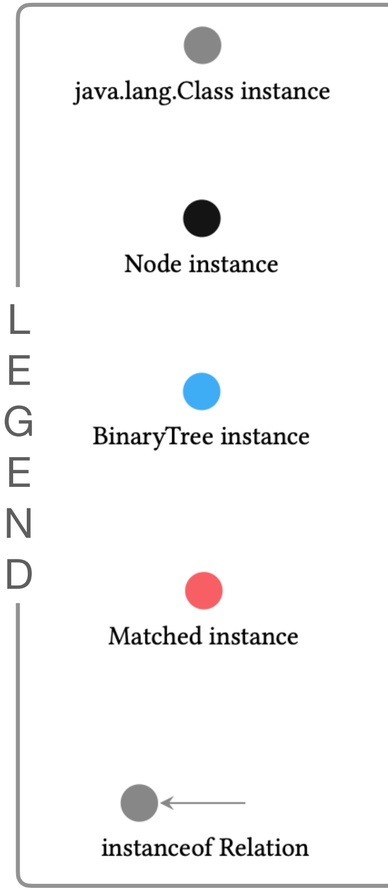



```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```

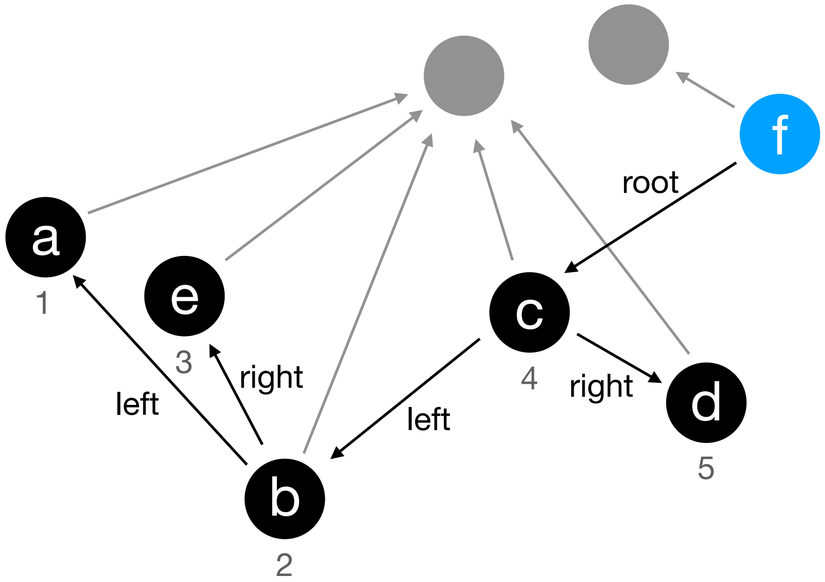
Example

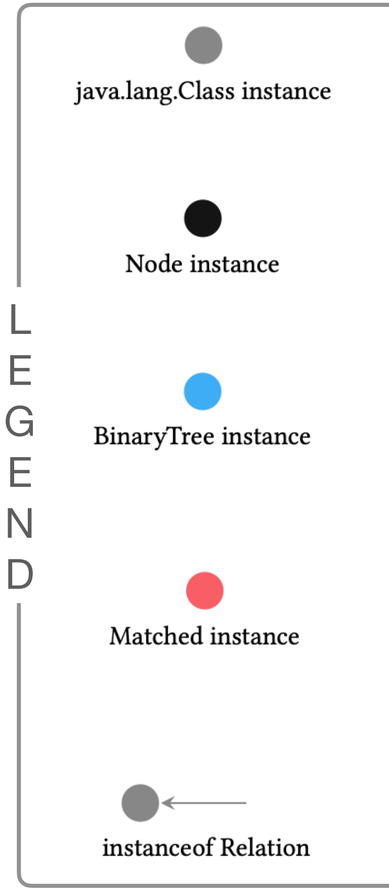



```
1 BinaryTree bTree = (BinaryTree) queryObject(
2 "CREATE (a:@1 {value:1}), (b:@1 {value:2}), (c:@1 {value:4}),
3  (d:@1 {value:5}), (e:@1 {value:3})
4  CREATE (f:@2 {size:5})
5  MERGE (a)<-[:left]-(b)<-[:left]-(c)-[:right]->(d) MERGE (b)-[:right]->(e)
6  MERGE (f)-[:root]->(c)
7  RETURN f", Node.class, BinaryTree.class);
```



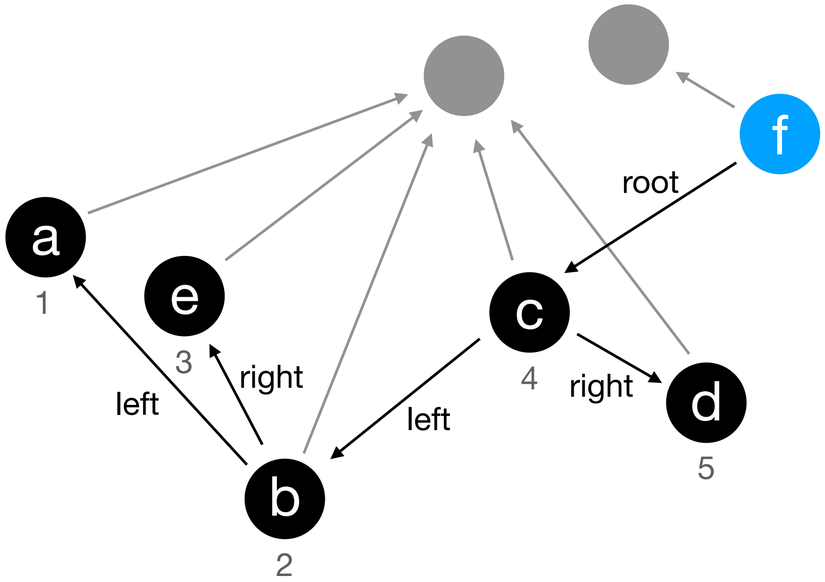
Example

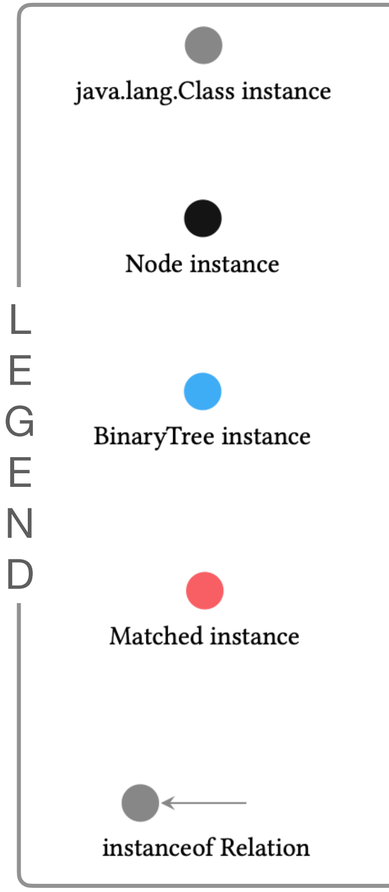





```
1 ResultSet matchedNodes = query(
2 bTree.root,
3 "MATCH (n {$1})-[*]-(m)
4  RETURN m",bTree.root);
```

Example

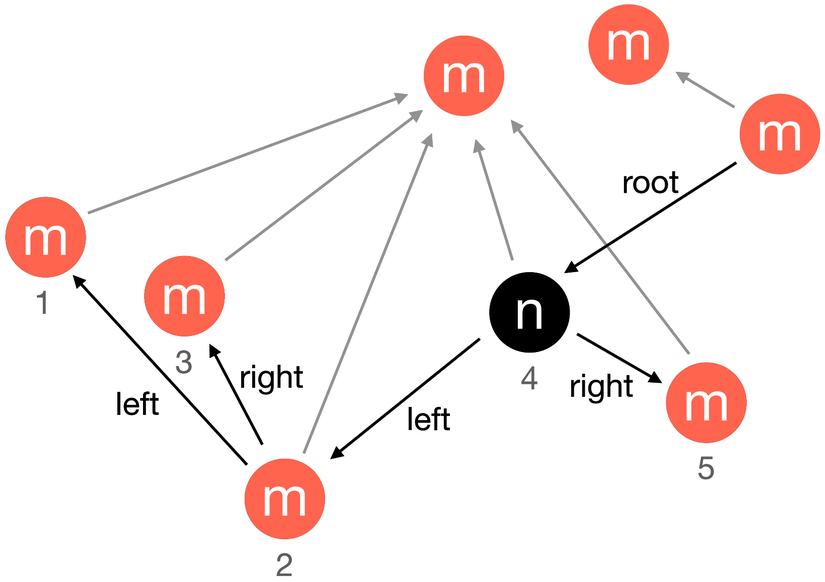




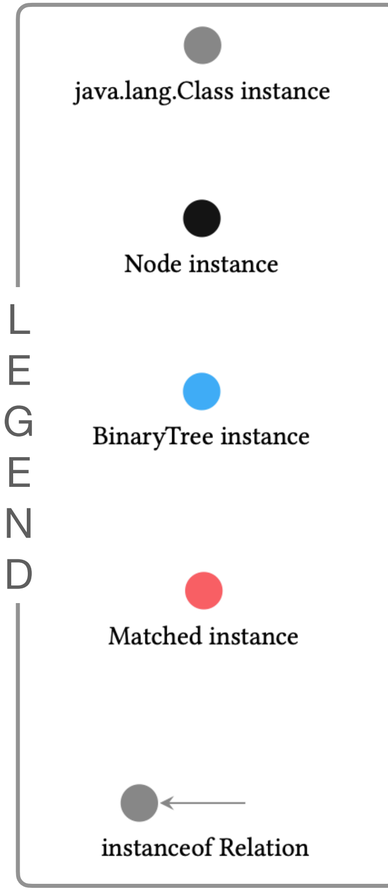

```
1 ResultSet matchedNodes = query(
2 bTree.root,
3 "MATCH (n {$1})-[*]-(m)
4  RETURN m",bTree.root);
```

Example

Useful for detecting confinement[1] and aliasing[2,3] issues




```
1 ResultSet matchedNodes = query(
2 bTree.root,
3 "MATCH (n {$1})-[*]-(m)
4  RETURN m",bTree.root);
```



1. Anindya Banerjee and David A. Naumann. Ownership confinement ensures representation independence for object-oriented programs. pages 894–960, J. ACM, 2005
2. John Hogg. Islands: Aliasing Protection In Object-Oriented Languages. pages 271–285, OOPSLA, 1990
3. Dave Clarke, James Noble and Tobias Wrigstad. Aliasing in Object-Oriented Programming. LNCS, Springer, 2013

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified
- Implement two prototypes of OGO:
  - OGO^{Neo} : Use external 3^{rd} party Cypher query engine (*Neo4j*)
  - OGO^{Mem} : Use custom-built in-memory Cypher query engine

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified
- Implement two prototypes of OGO:
  - OGO^{Neo} : Use external 3^{rd} party Cypher query engine (*Neo4j*)
  - OGO^{Mem} : Use custom-built in-memory Cypher query engine
- Introduce optimizations to improve efficiency of query execution

Our Contributions

- OGO : Object Graph Programming that enables mixing imperative and declarative styles
- Introduce a new view of the runtime state of a program
- Introduce a novel way by which runtime state can be queried and modified
- Implement two prototypes of OGO:
  - OGO^{Neo} : Use external 3^{rd} party Cypher query engine (*Neo4j*)
  - OGO^{Mem} : Use custom-built in-memory Cypher query engine
- Introduce optimizations to improve efficiency of query execution
- Evaluation
  - Demonstrate increase in expressivity by re-implementing imperative Java methods with OGO
  - Demonstrate robustness of OGO by re-implementing assertions in open-source projects with OGO

Language Paradigms

Imperative

- Focuses on the *how*
- Eg languages : Java, C/C++, Python

```java
public boolean containsKey(Object key) {
    return getNode(hash(key), key) != null;
}
final Node<K,V> getNode(int hash, Object key) {
    Node<K,V>[] tab; Node<K,V> first, e; int n; K k;
    if ((tab = table) != null && (n = tab.length) > 0 &&
            (first = tab[(n - 1) & hash]) != null) {
        if (first.hash == hash && // always check first node
            ((k = first.key) == key || (key != null && key.equals(k))))
            return first;
        if ((e = first.next) != null) {
            if (first instanceof TreeNode)
                return ((TreeNode<K,V>)first).getTreeNode(hash, key);
            do {
                if (e.hash == hash && ((k = e.key) == key ||
                     (key != null && key.equals(k)))) return e;
            } while ((e = e.next) != null);}
        } return null;}
```

Declarative

- Focuses on the *what*
- Eg languages : SQL, Cypher

```java
public boolean containsKey(Object key) {
    return queryBool(
       "MATCH ({$1})-[:table]->()-[*]->()-[:key]->(n) MATCH (m {$2})
        WHERE n.`equals`(m) = true RETURN COUNT(n) <> 0", this, key);
}
```

Results - Assertions

- We reimplemented assertions in several popular open source projects using *OGO*
- We compared the end-to-end execution times for the query executions for the different modes of *OGO* and for different optimizations
- We see that OGO^{Mem} with **+WL+FGC** performs best with an average execution time of **164ms**

Project	SHA	LOC	#Assert	OGO^{Neo}					OGO^{Mem}			Redn.
					+WL					+WL		
				Naive	-	+FRO	+FGC	+FRO+FGC	Naive	-	+FGC	[%]
Assertj-db	8aefa0f	78555	9	832652	7063	7092	4799	4861	851	860	248	95
Cli	0d06c4b	9256	27	99798	1987	1982	1900	1880	224	226	160	91
Csv	1c551d9	9910	12	448403	2867	2713	2291	2171	319	318	178	92
Lang	dded8fd	86960	23	255390	1718	1724	1510	1437	287	286	148	90
Geometry	6c6d046	79996	95	OOM	1407	1397	1321	1267	258	257	145	89
Guava	0ca124d	367861	5	175658	2804	2784	2458	2408	217	218	130	95
Jfreechart	7cdbfbe	134399	10	383694	1811	1824	1625	1548	256	234	134	91
Json-java	31110b5	14829	10	99798	1456	1423	1483	1400	155	155	126	91
Tcases	2fccf74	482581	26	323006	39736	39713	2120	2068	768	796	199	90
Zip4j	fc3a258	15525	18	79658	3528	3548	3322	3157	244	242	174	94
Avg.	-	-	-	269806	6419	6420	2283	2219	358	359	164	-
Σ				2698057	64197	64200	22829	22197	3579	3592	1642	