
ORACLE®

ASGraph

A Mutable Multi-Versioned Graph Container with High Analytical Performance

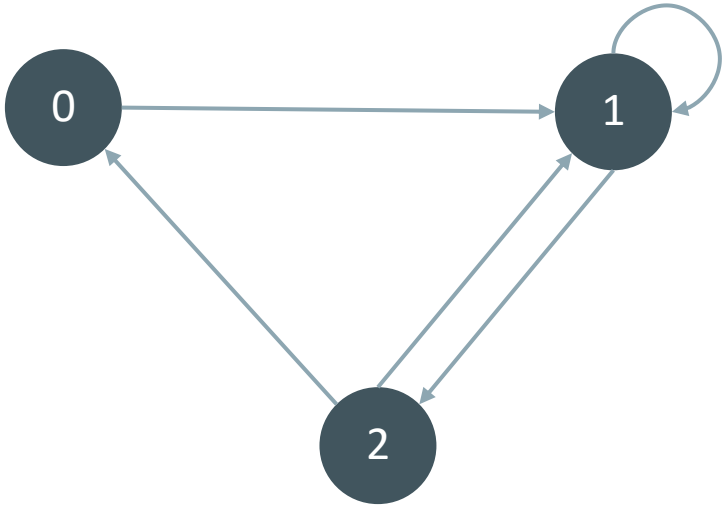
Michael Haubenschild
Manuel Then
Sungpack Hong
Hassan Chafi

Oracle Labs
June 24, 2016

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

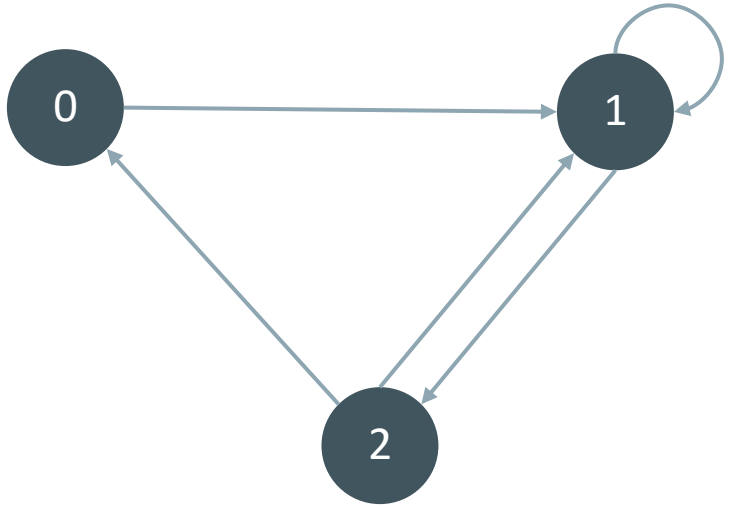
How do we store graphs?



How do we store graphs?

Adjacency matrix

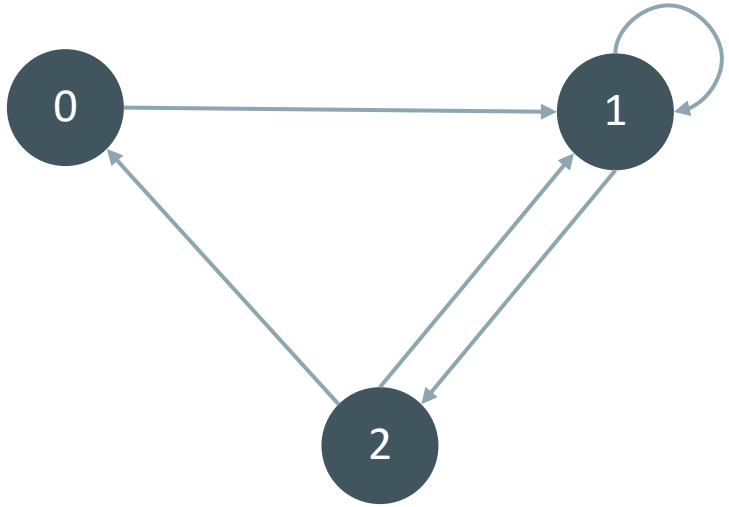
	A	B	C
A		x	
B		x	x
C	x	x	



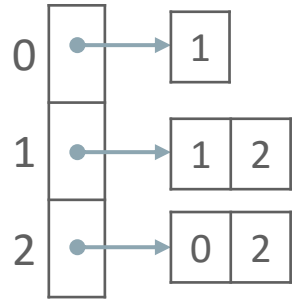
How do we store graphs?

Adjacency matrix

	A	B	C
A		x	
B		x	x
C	x	x	



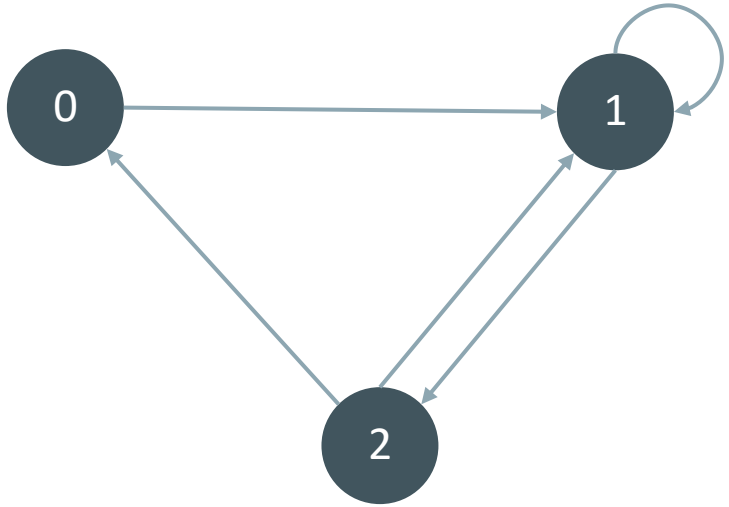
Adjacency Lists



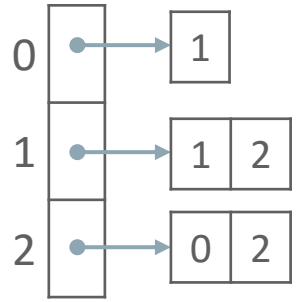
How do we store graphs?

Adjacency matrix

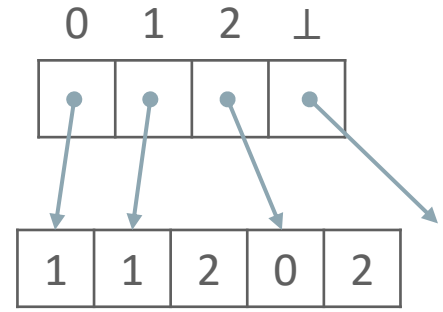
	A	B	C
A		x	
B		x	x
C	x	x	



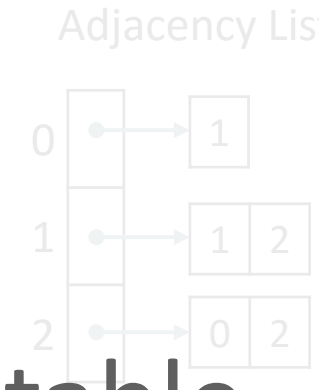
Adjacency Lists



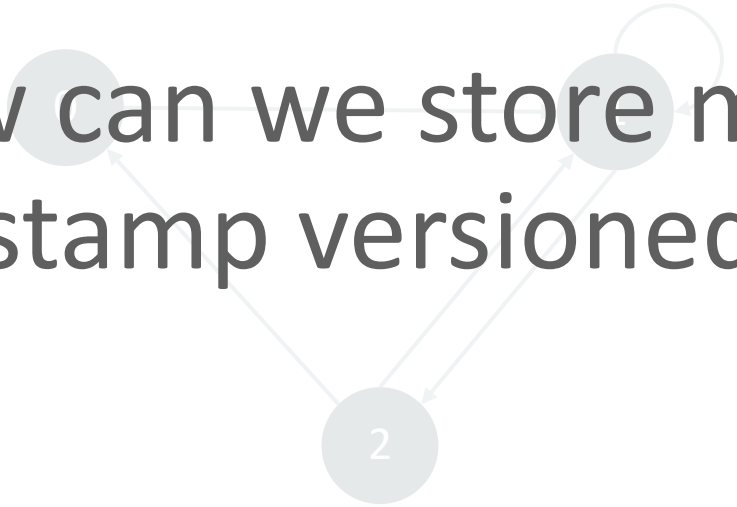
Compressed Sparse Row



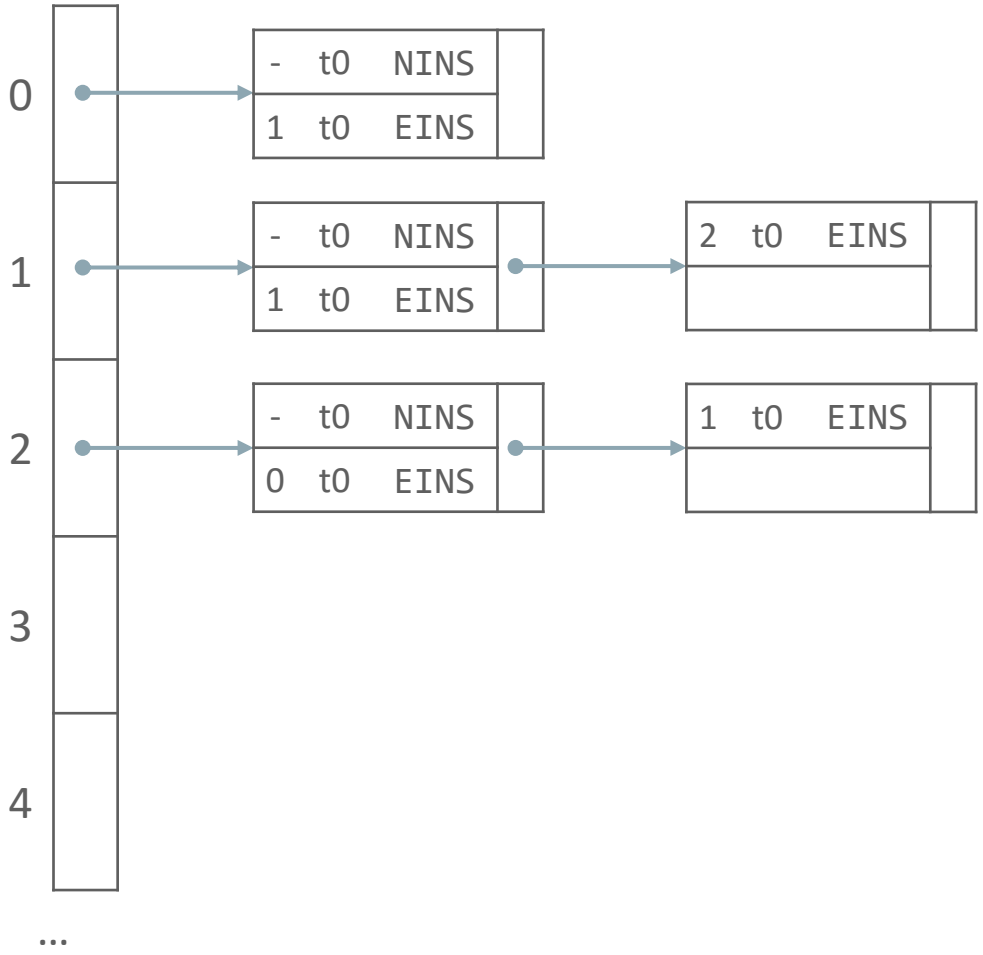
How do we store graphs?

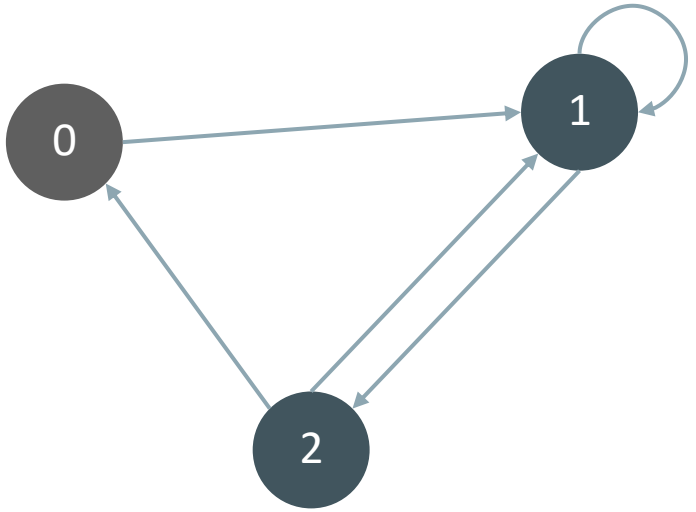


How can we store mutable, timestamp versioned graphs?



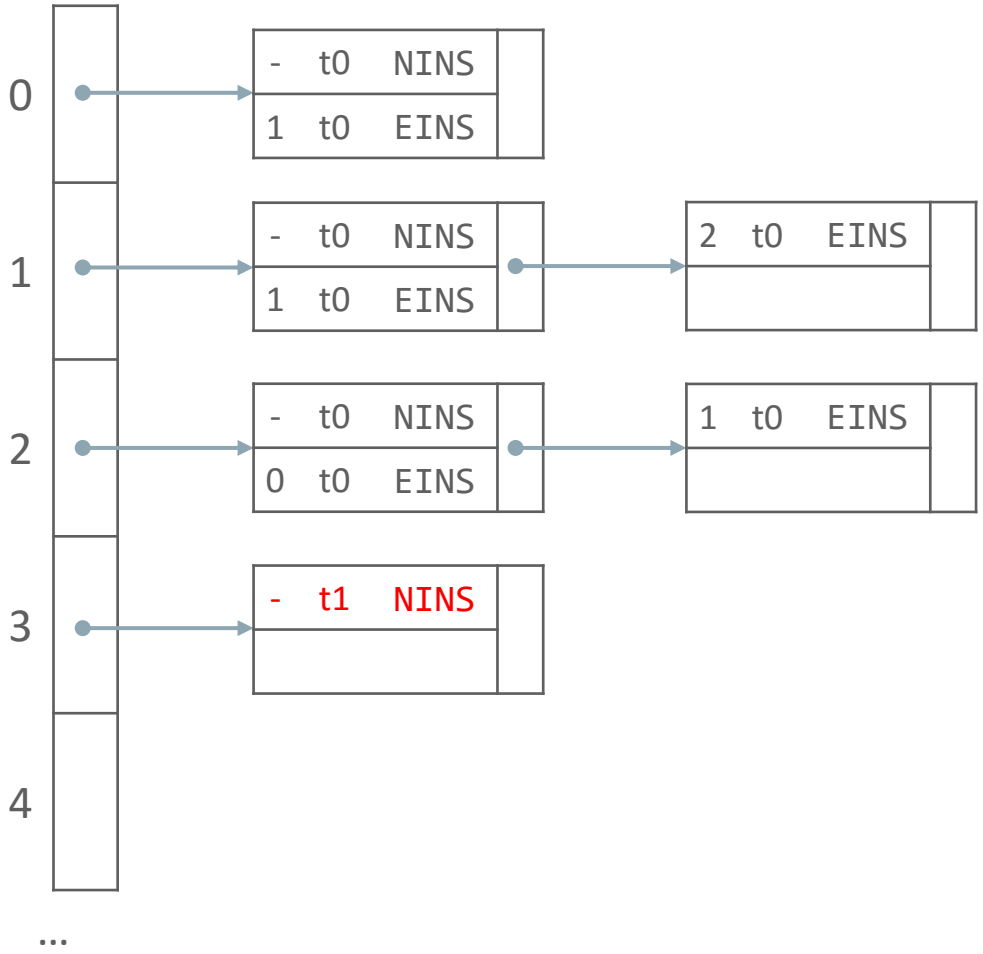
ASGraph (=Analytical Snapshot Graph)

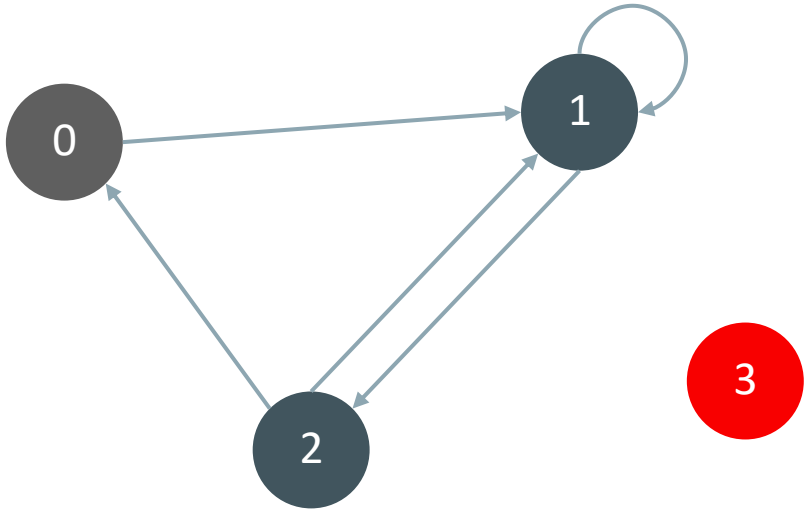




Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

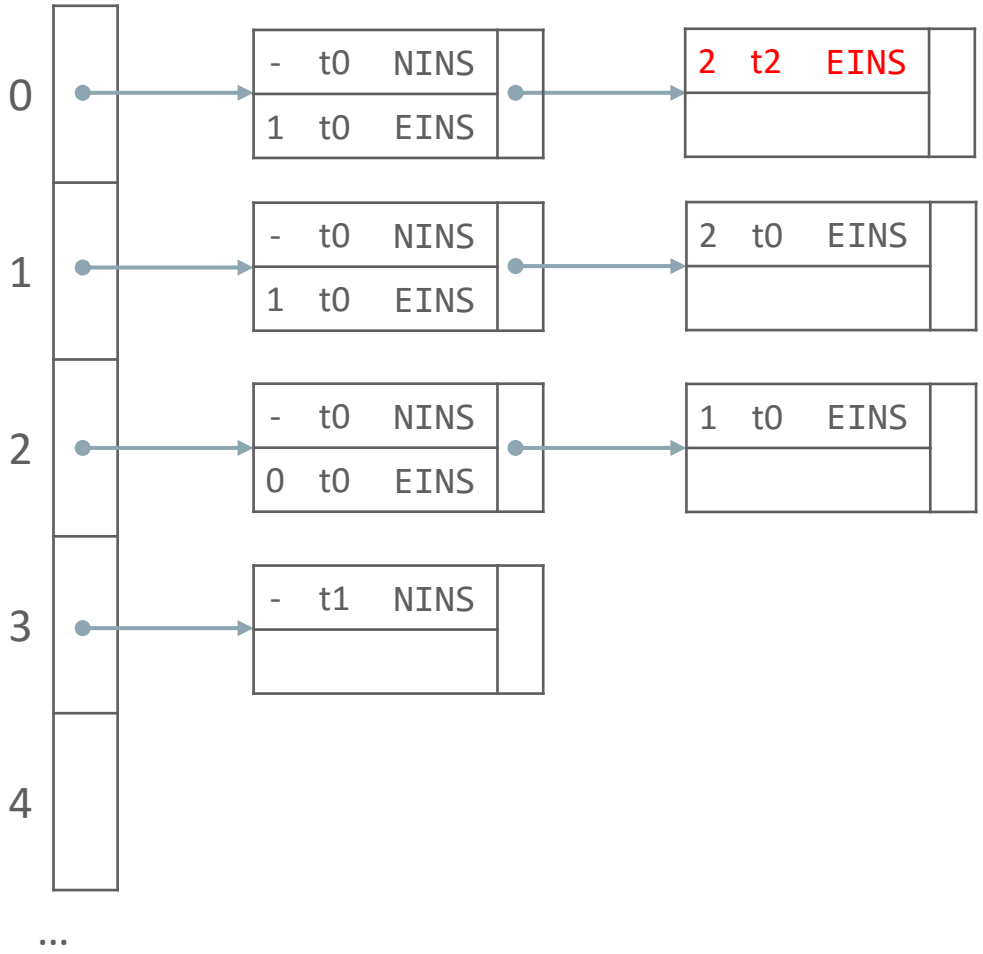
ASGraph

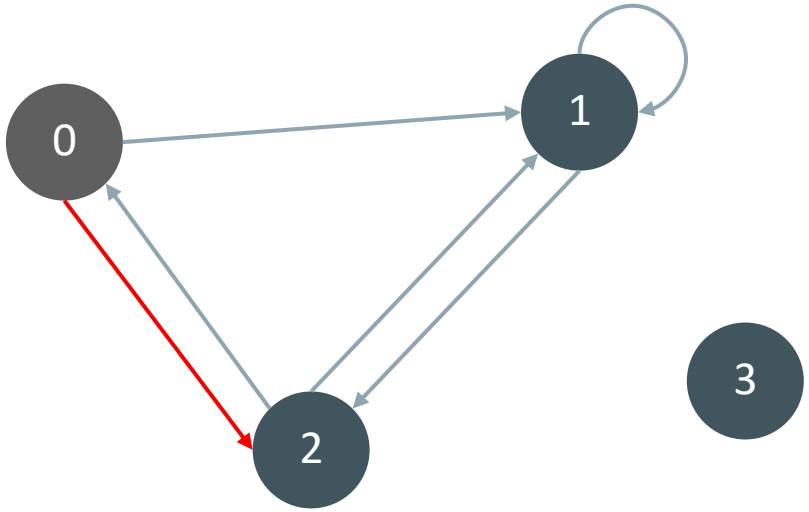




Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

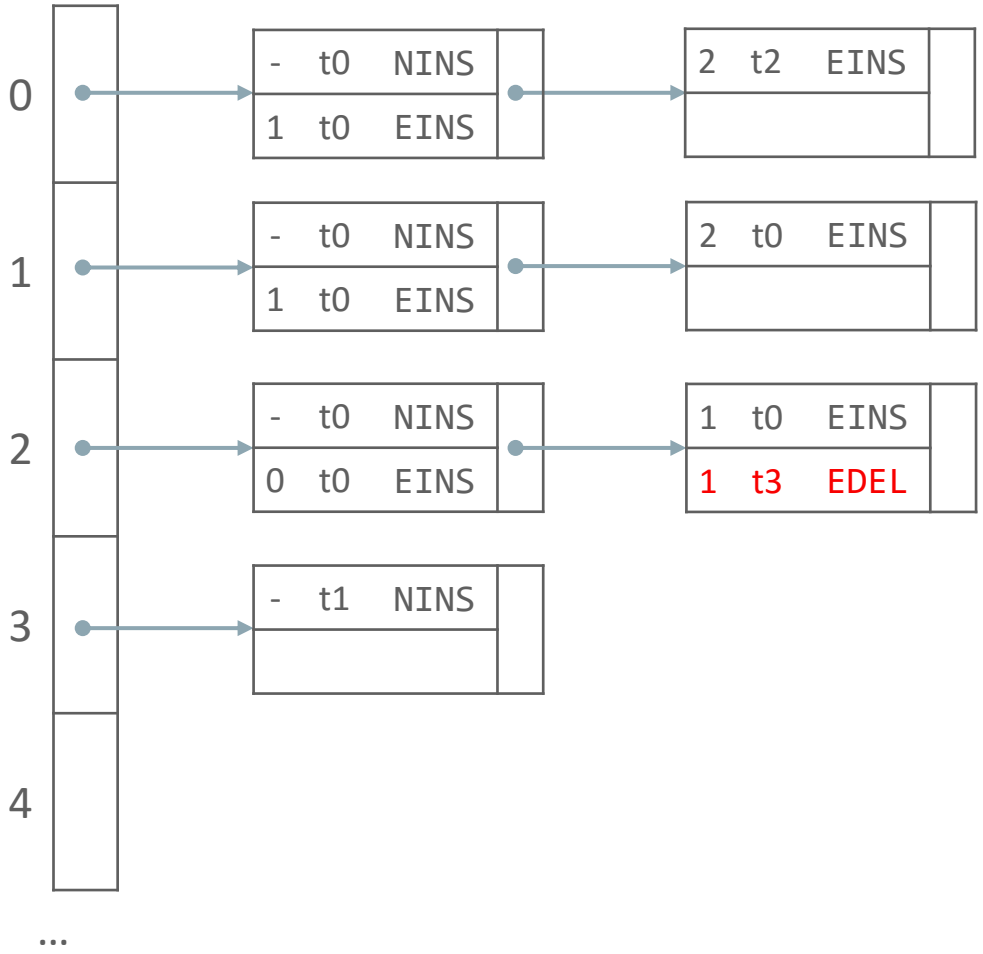
ASGraph

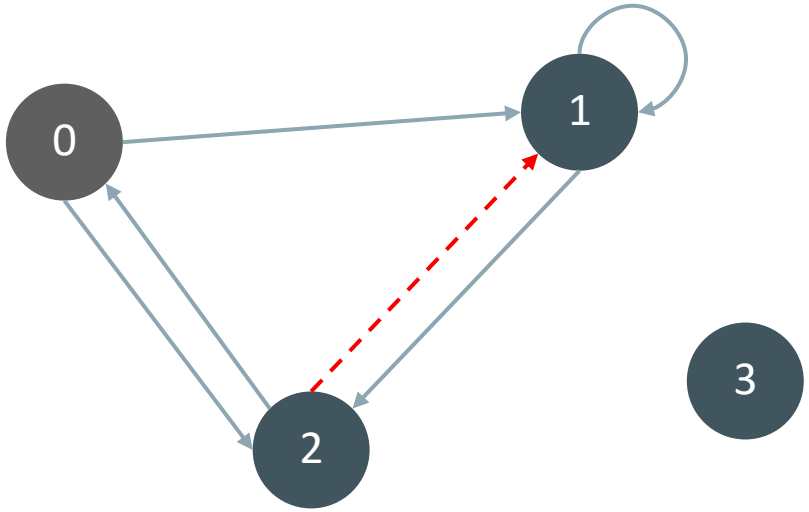




Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

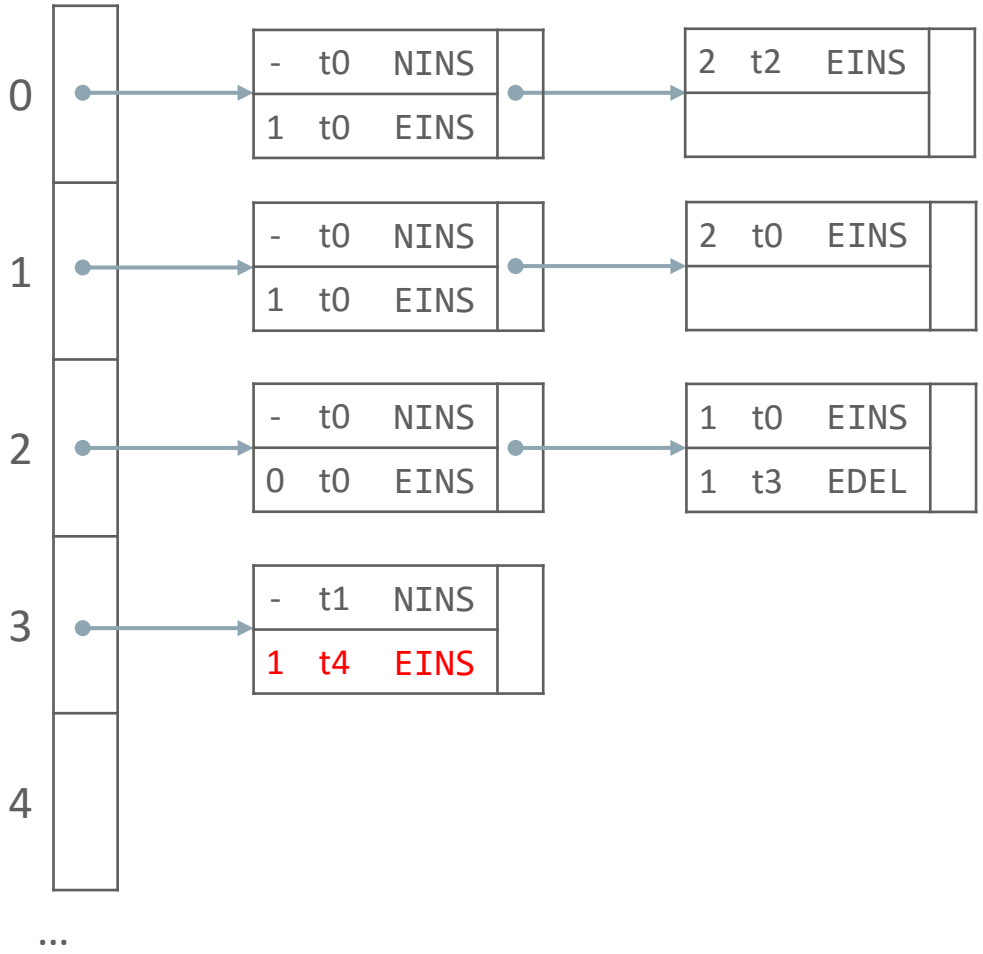
ASGraph

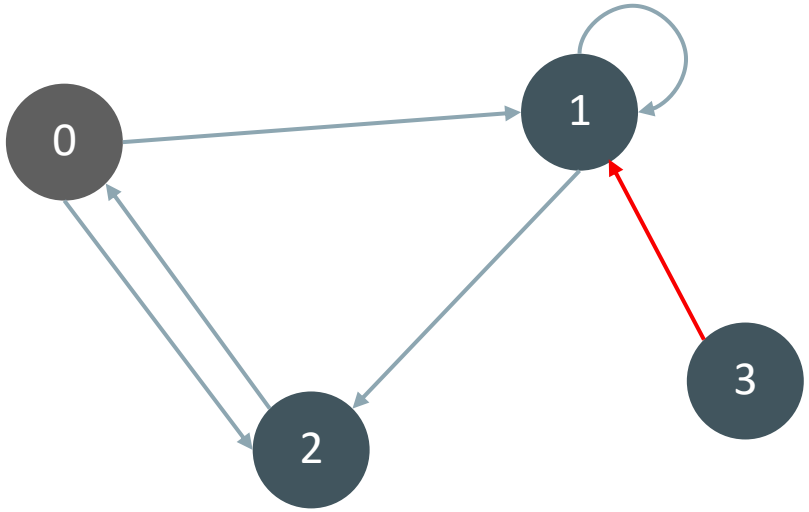




Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

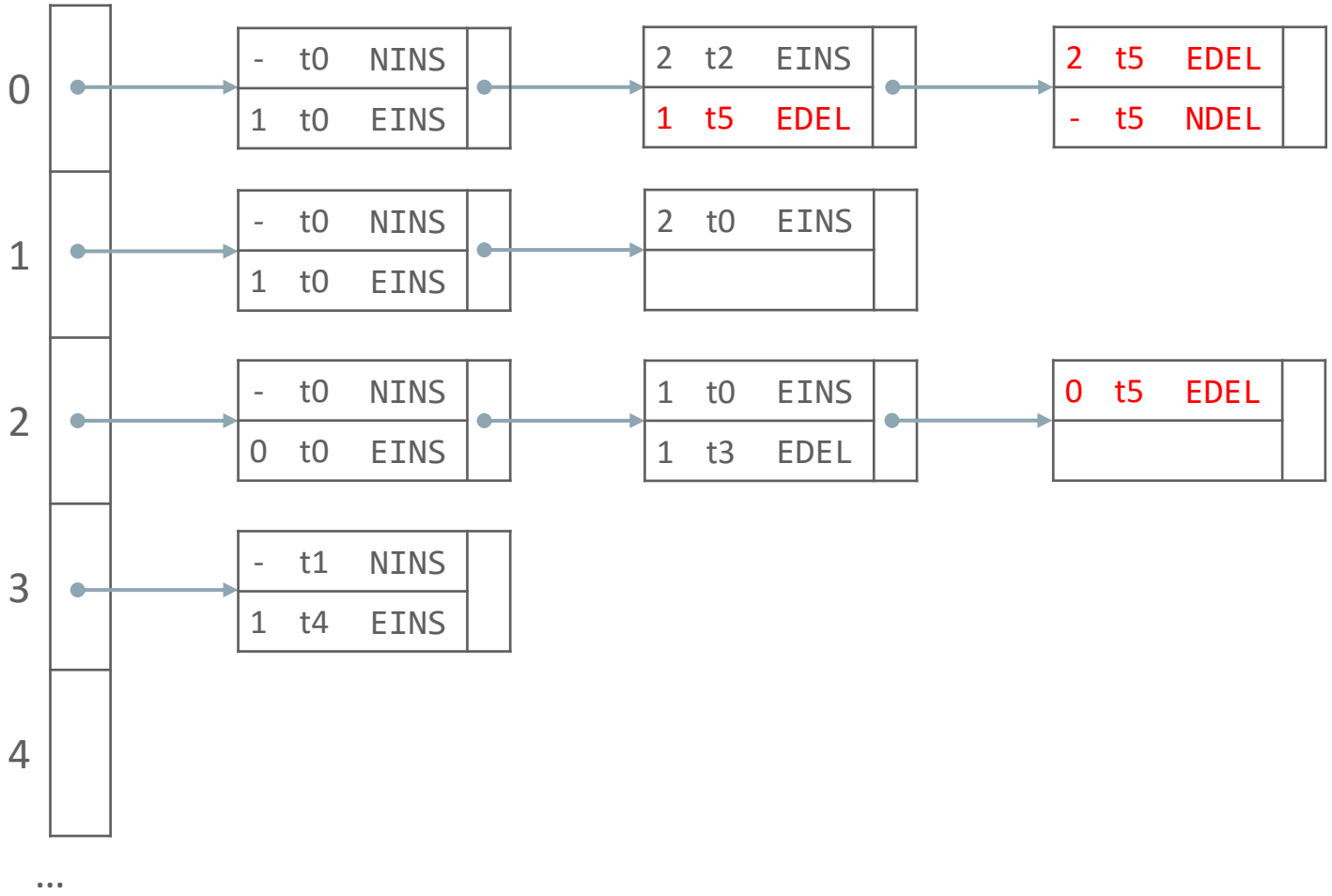
ASGraph

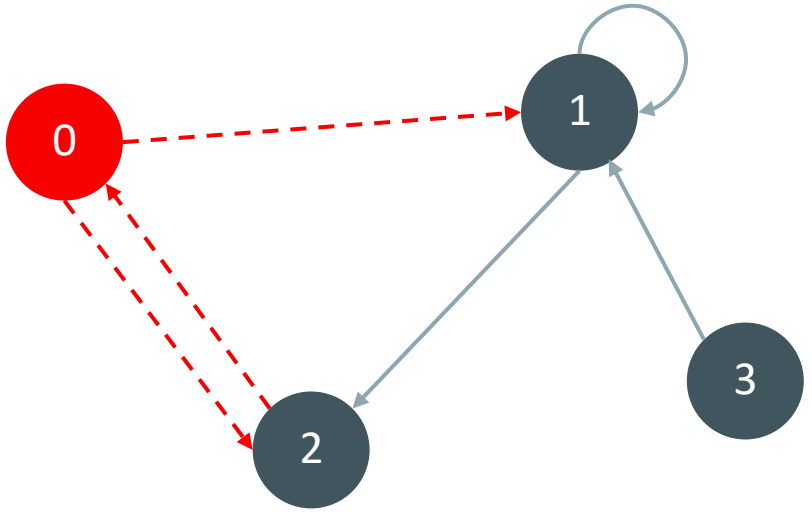




Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

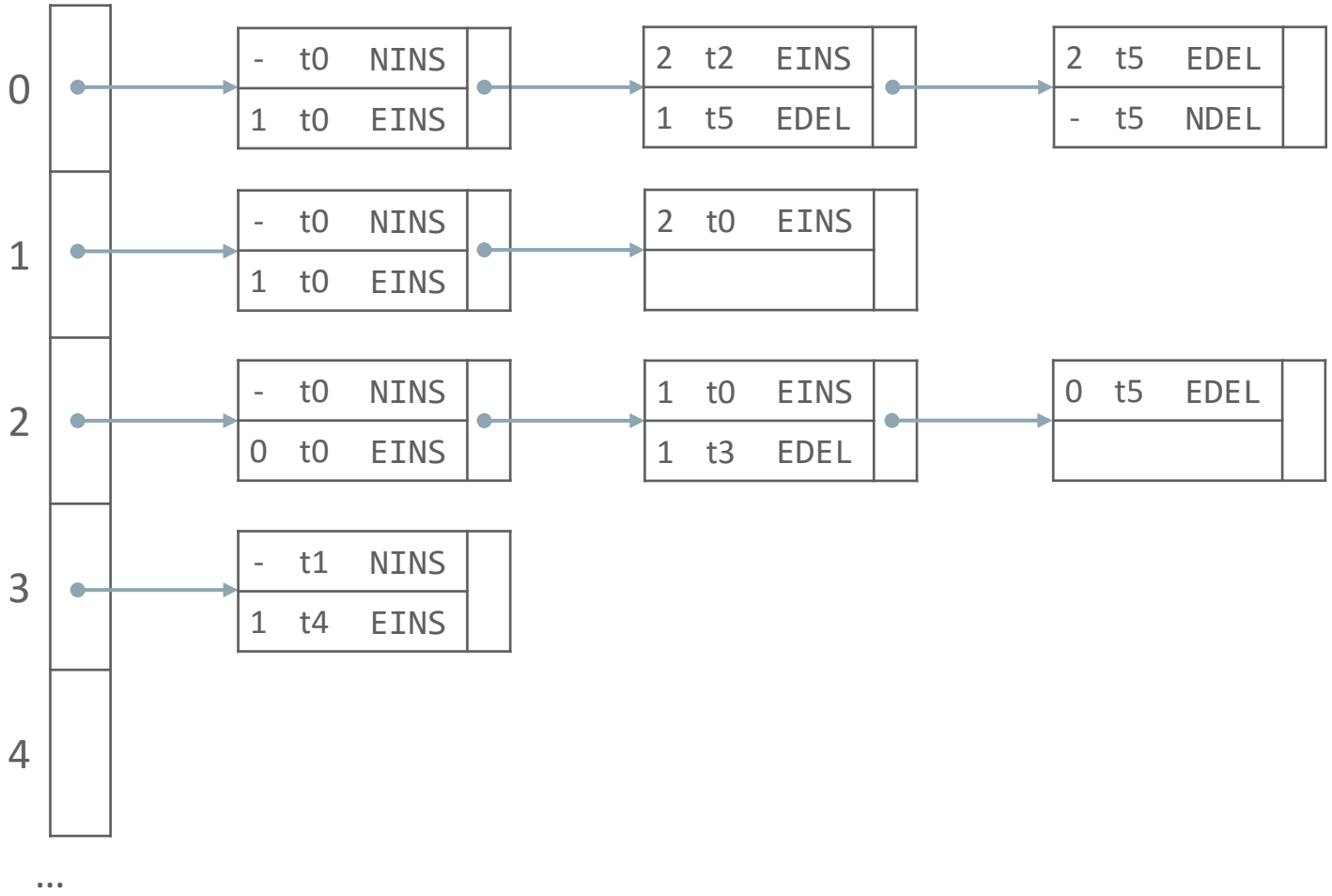
ASGraph

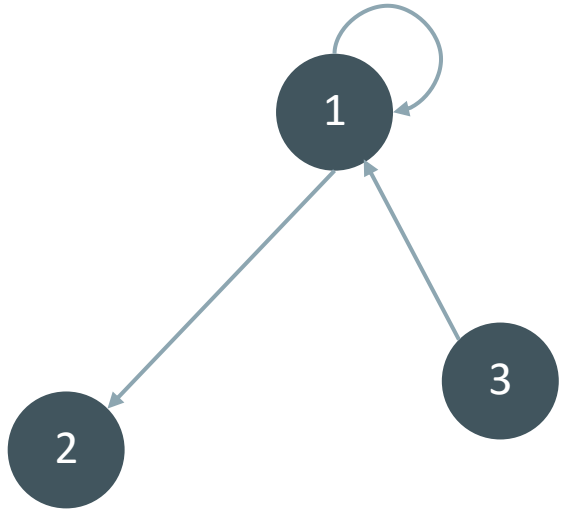




Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

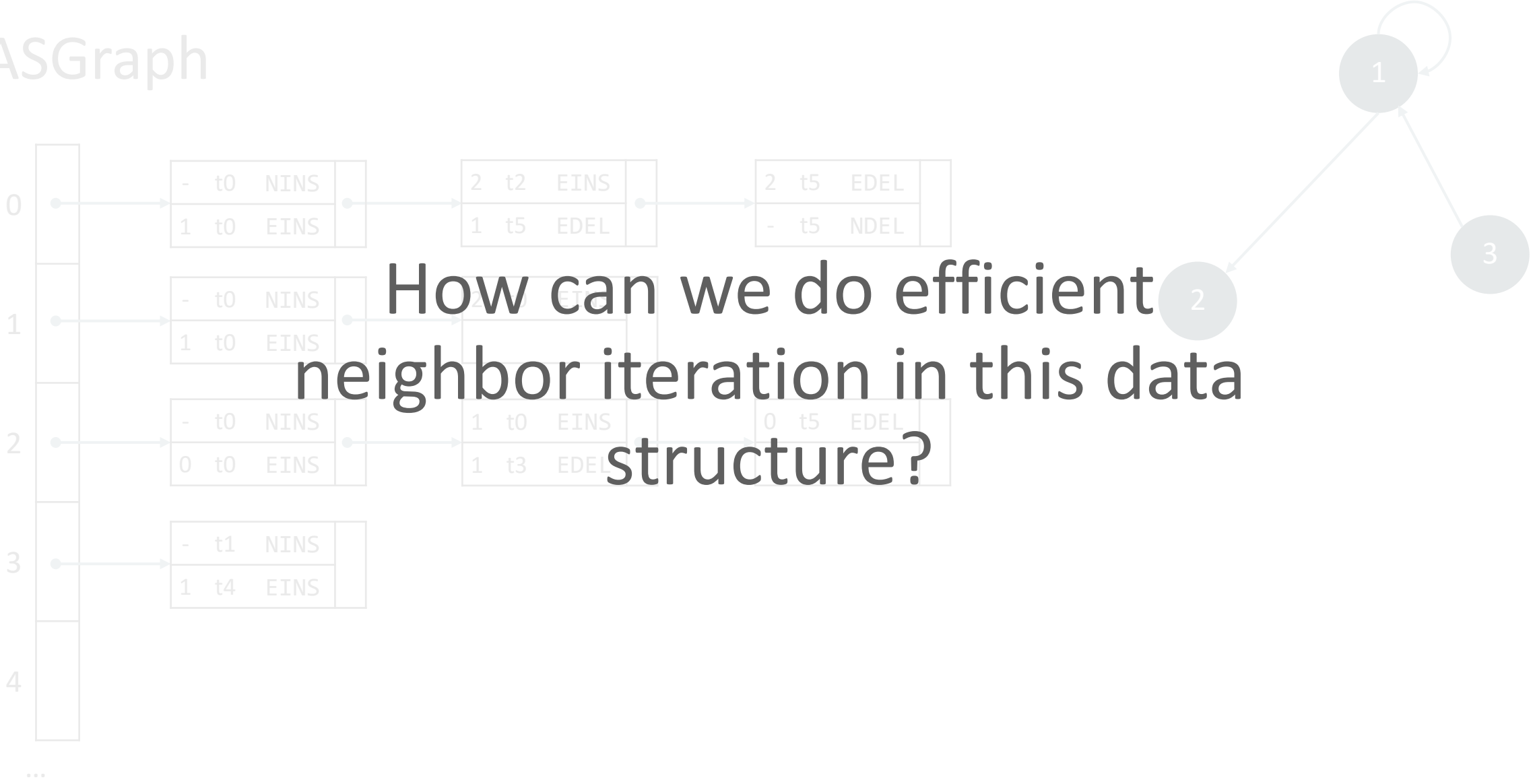
ASGraph





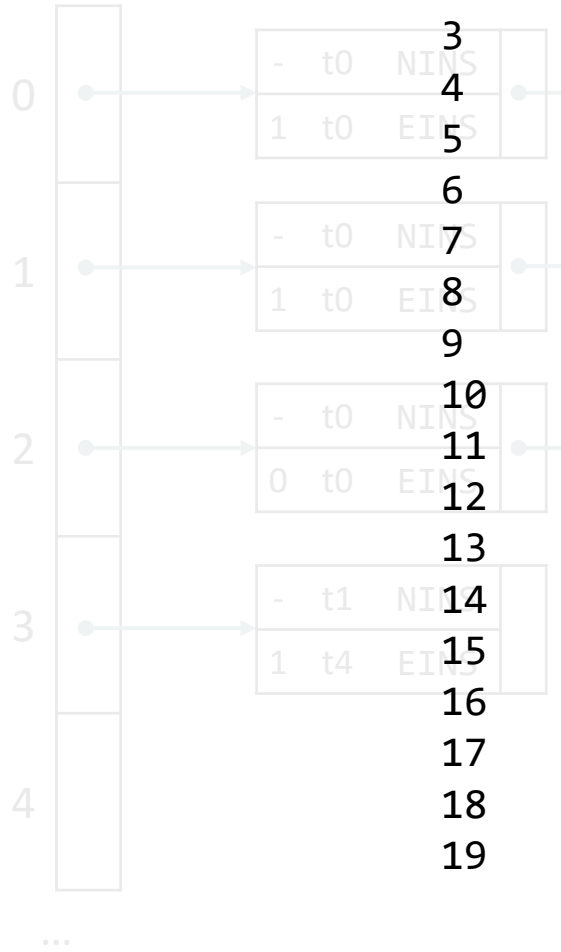
Operation Types	
NINS	Node Insert
NDEL	Node Delete
EINS	Edge Insert
EDEL	Edge Delete

ASGraph

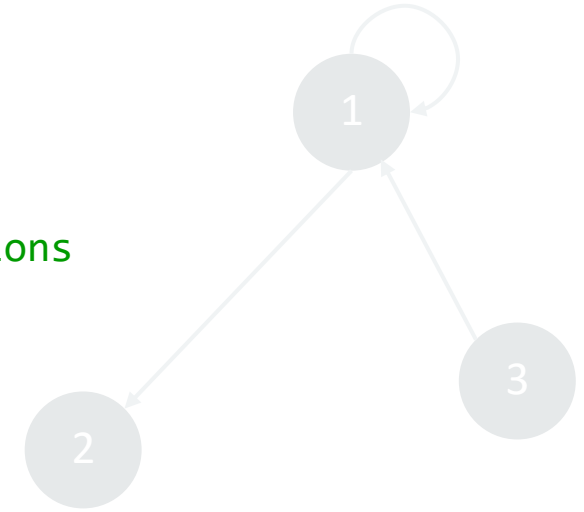


How can we do efficient neighbor iteration in this data structure?

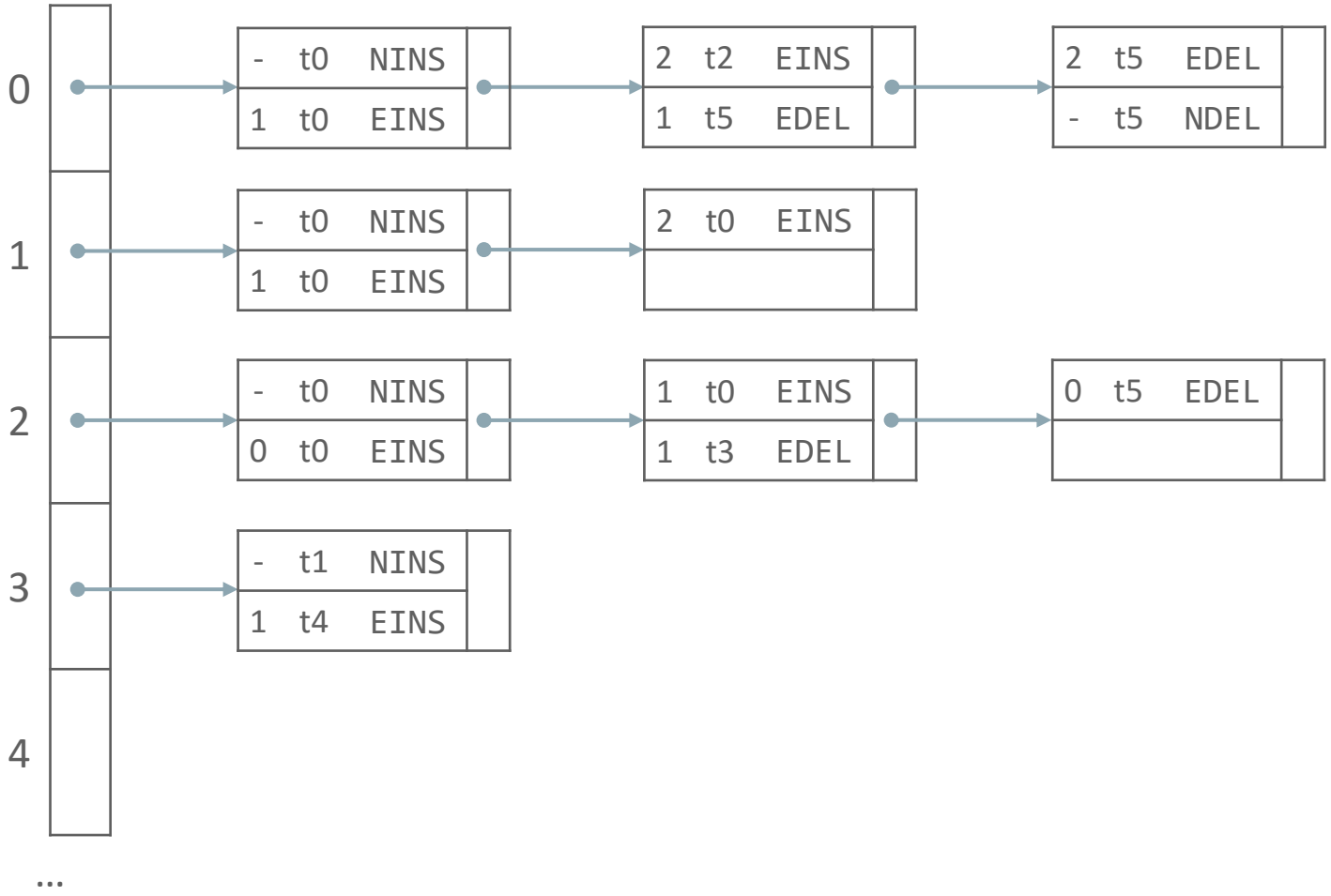
ASGraph



```
0 createSnapshot(TS t)
1   for n : nodes
2      var CAND := List[operation] //edge candidates
3      var REST := List[operation] //remaining operations
4      var DEL := Map[nodeId->timestamp]
5      for op : n.operations
6         if(op.type == EINS && op.timestamp <= t)
7            CAND.append(op)
8         else
9            REST.append(op)
10           if(op.type == EDEL)
11              DEL[op.to] = op.timestamp
12     for c : CAND
13        if(c.timestamp <= DEL[c.to])
14        //move c to REST
15        var tmp = CAND.remove(c)
16        REST.append(tmp)
17        //OPTIONALLY: sort CAND at this point
18        //Replace bucket content with reordered operations
19     n.operations <- concat(CAND,[EOS],REST)
```

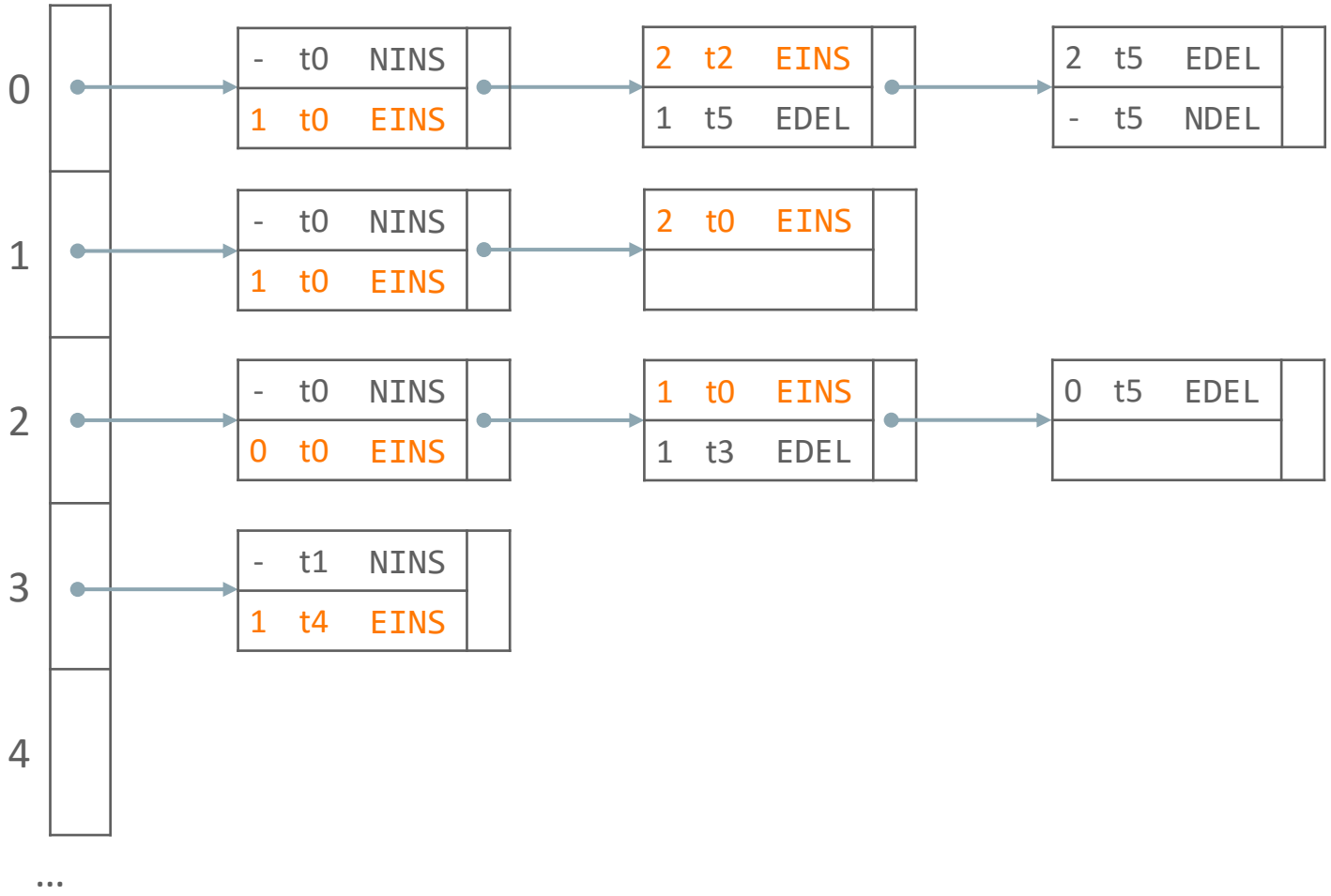


createSnapshot(t5)



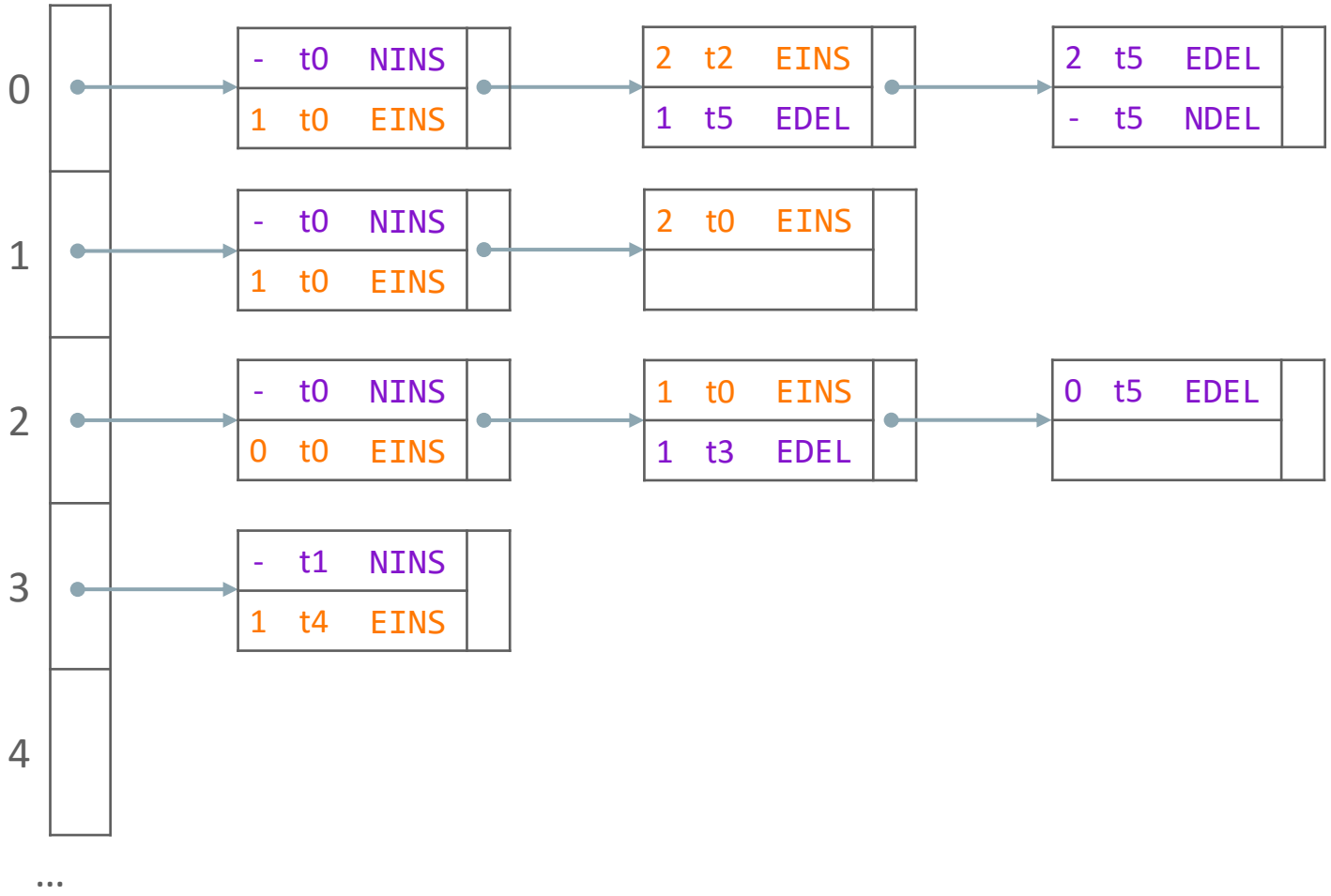

```
1 for n : nodes
2    var CAND := List[operation]
3    var REST := List[operation]
4    var DEL := Map[nodeId->timestamp]
5    for op : n.operations
6       if(op.type == EINS && op.timestamp <= t)
7          CAND.append(op)
8       else
9          REST.append(op)
10         if(op.type == EDEL)
11            DEL[op.to] = op.timestamp
12   for c : CAND
13      if(c.timestamp <= DEL[c.to])
14         var tmp = CAND.remove(c)
15         REST.append(tmp)
16   n.operations <- concat(CAND,[EOS],REST)
```

createSnapshot(t5)



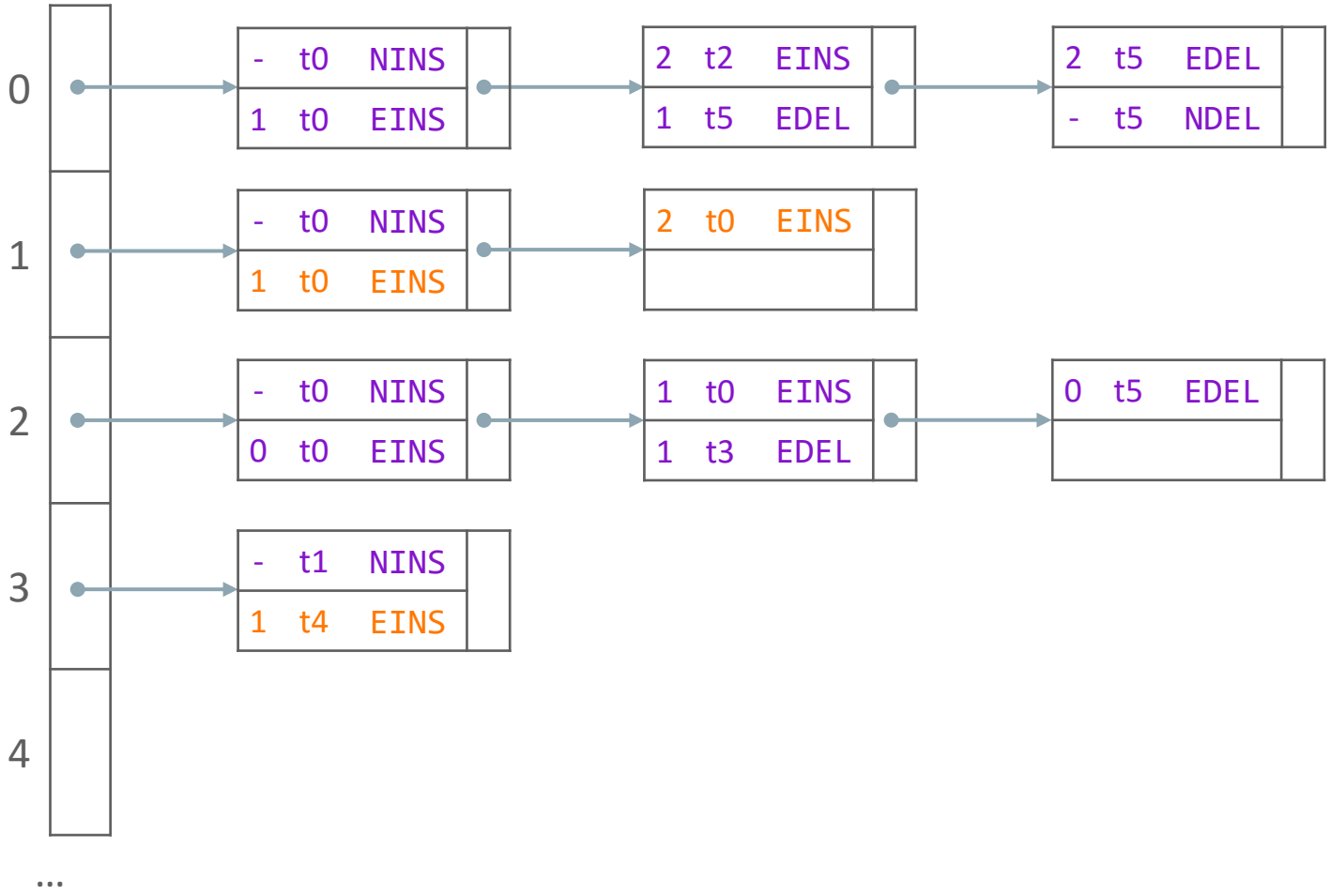

```
1 for n : nodes
2    var CAND := List[operation]
3    var REST := List[operation]
4    var DEL := Map[nodeId->timestamp]
5    for op : n.operations
6       if(op.type == EINS && op.timestamp <= t)
7          CAND.append(op)
8       else
9          REST.append(op)
10         if(op.type == EDEL)
11            DEL[op.to] = op.timestamp
12   for c : CAND
13      if(c.timestamp <= DEL[c.to])
14         var tmp = CAND.remove(c)
15         REST.append(tmp)
16   n.operations <- concat(CAND,[EOS],REST)
```

createSnapshot(t5)



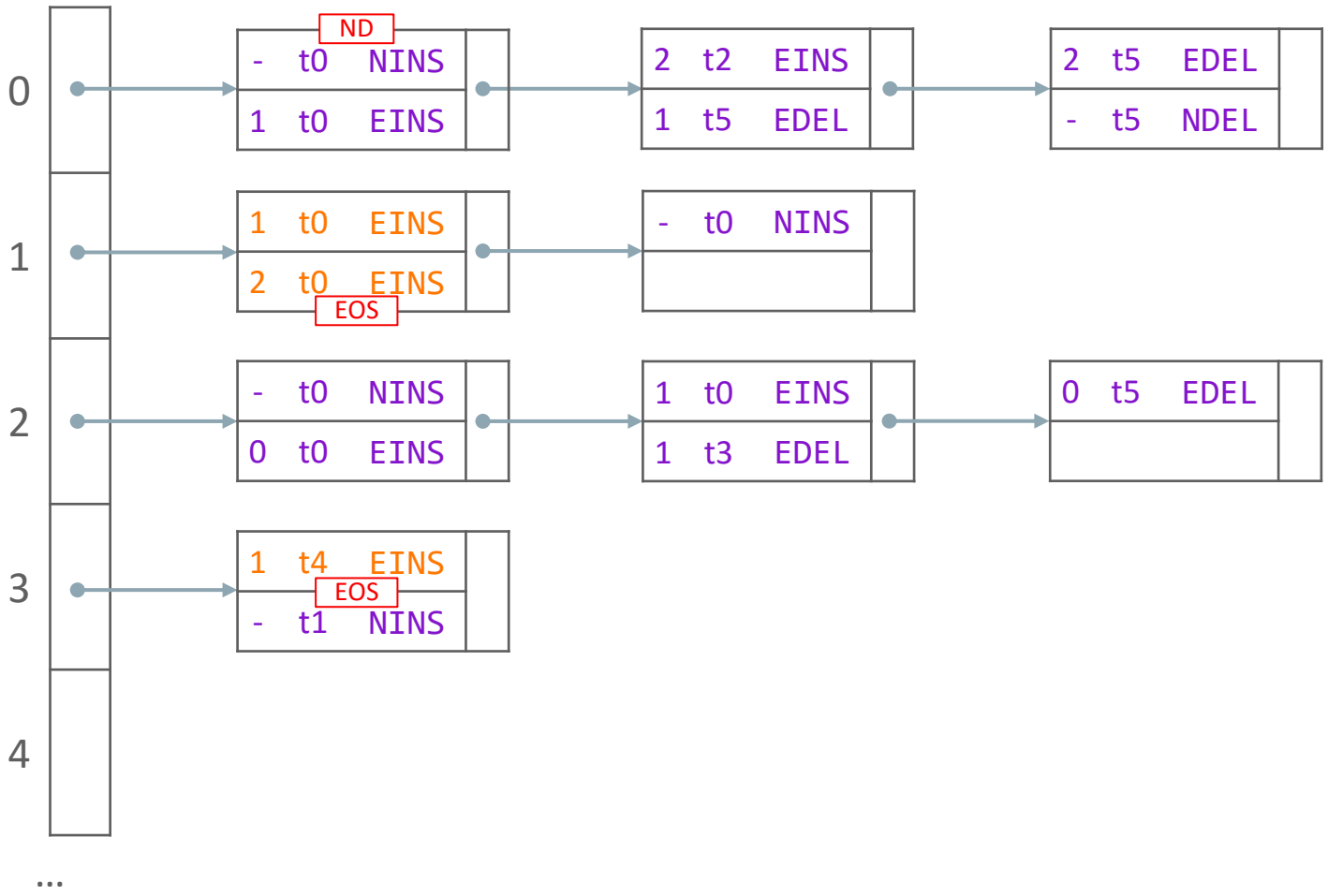

```
1 for n : nodes
2    var CAND := List[operation]
3    var REST := List[operation]
4    var DEL := Map[nodeId->timestamp]
5    for op : n.operations
6       if(op.type == EINS && op.timestamp <= t)
7          CAND.append(op)
8       else
9          REST.append(op)
10         if(op.type == EDEL)
11            DEL[op.to] = op.timestamp
12   for c : CAND
13      if(c.timestamp <= DEL[c.to])
14         var tmp = CAND.remove(c)
15         REST.append(tmp)
16   n.operations <- concat(CAND,[EOS],REST)
```

createSnapshot(t5)




```
1 for n : nodes
2    var CAND := List[operation]
3    var REST := List[operation]
4    var DEL := Map[nodeId->timestamp]
5    for op : n.operations
6       if(op.type == EINS && op.timestamp <= t)
7          CAND.append(op)
8       else
9          REST.append(op)
10         if(op.type == EDEL)
11            DEL[op.to] = op.timestamp
12   for c : CAND
13      if(c.timestamp <= DEL[c.to])
14         var tmp = CAND.remove(c)
15         REST.append(tmp)
16   n.operations <- concat(CAND,[EOS],REST)
```

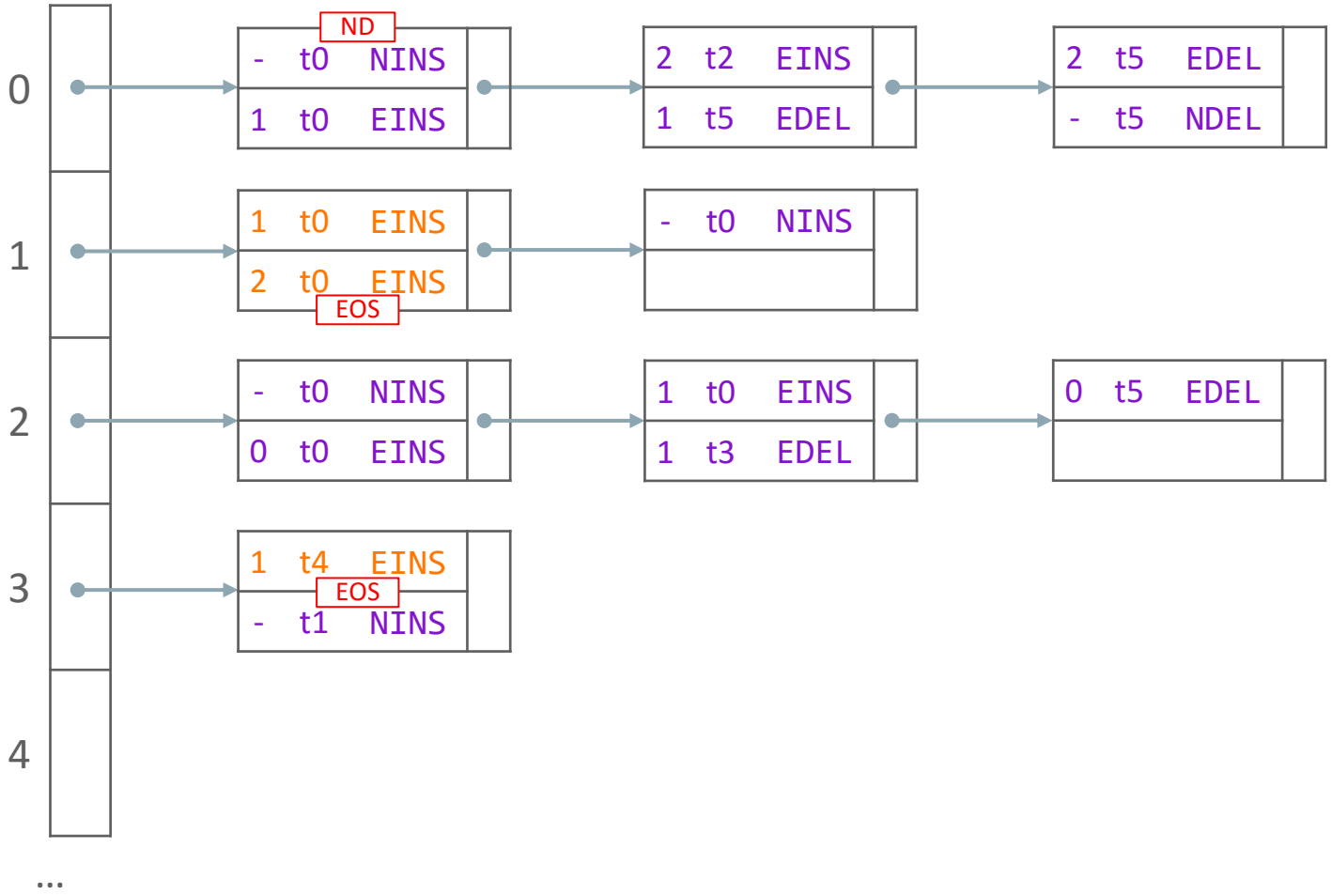
createSnapshot(t5)

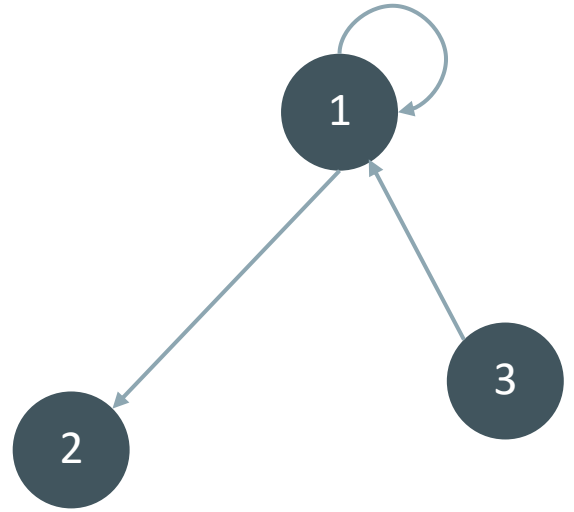


Marker Types	
[EOS]	End of Snapshot
[ND]	Node Deleted

```
1 for n : nodes
2    var CAND := List[operation]
3    var REST := List[operation]
4    var DEL := Map[nodeId->timestamp]
5    for op : n.operations
6       if(op.type == EINS && op.timestamp <= t)
7          CAND.append(op)
8       else
9          REST.append(op)
10         if(op.type == EDEL)
11            DEL[op.to] = op.timestamp
12   for c : CAND
13      if(c.timestamp <= DEL[c.to])
14         var tmp = CAND.remove(c)
15         REST.append(tmp)
16   n.operations <- concat(CAND,[EOS],REST)
```

createSnapshot(t5)






```
1 for n : nodes
2    var CAND := List[operation]
3    var REST := List[operation]
4    var DEL := Map[nodeId->timestamp]
5    for op : n.operations
6       if(op.type == EINS && op.timestamp <= t)
7          CAND.append(op)
8       else
9          REST.append(op)
10         if(op.type == EDEL)
11            DEL[op.to] = op.timestamp
12   for c : CAND
13      if(c.timestamp <= DEL[c.to])
14         var tmp = CAND.remove(c)
15         REST.append(tmp)
16   n.operations <- concat(CAND,[EOS],REST)
```

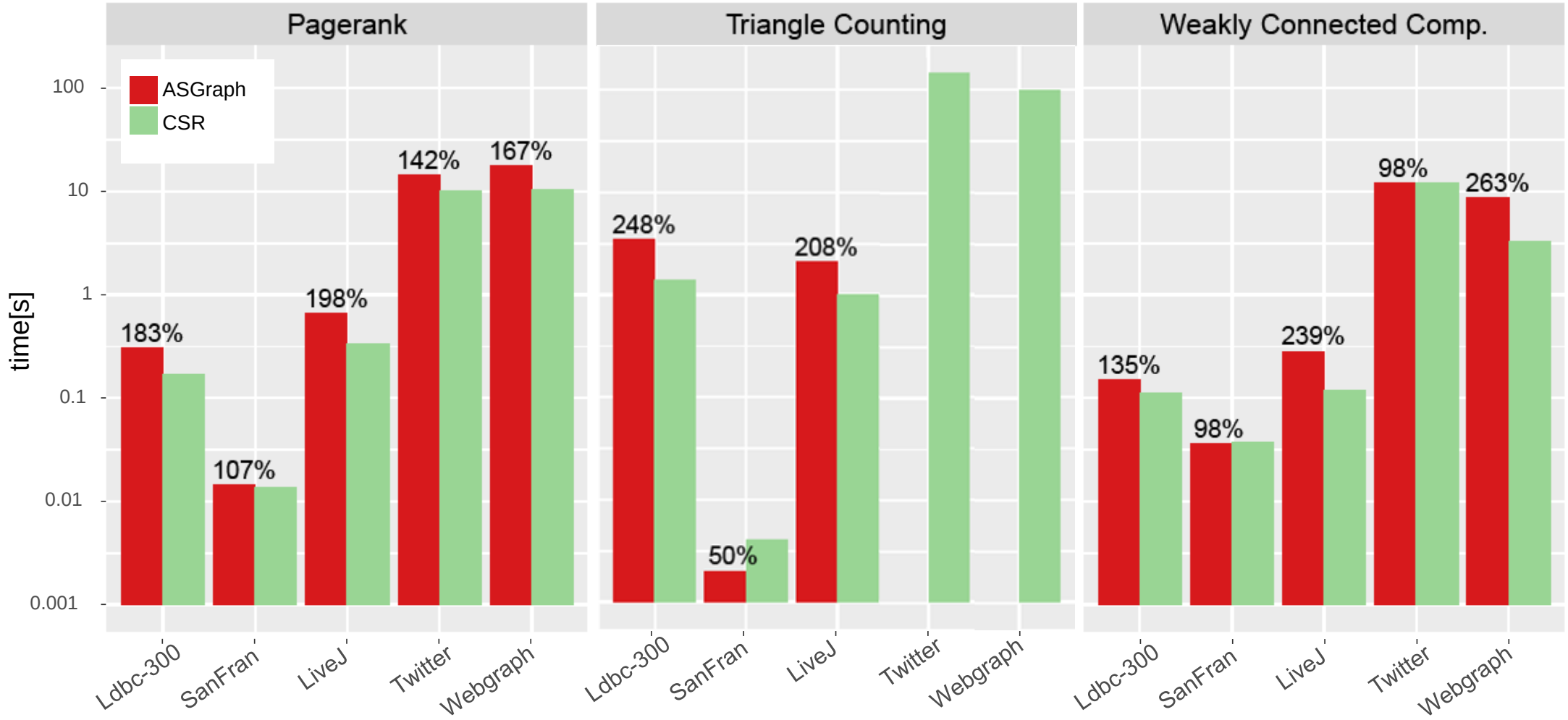
Advantages of this approach

- Keep the iteration logic simple
  - ➔ better code generation & branch prediction
- Prefetch next buckets explicitly
  - ➔mitigate effects of cache misses
- Concurrent updates are possible
  - ➔We do not touch memory beyond the last entry at the time of snapshot creation

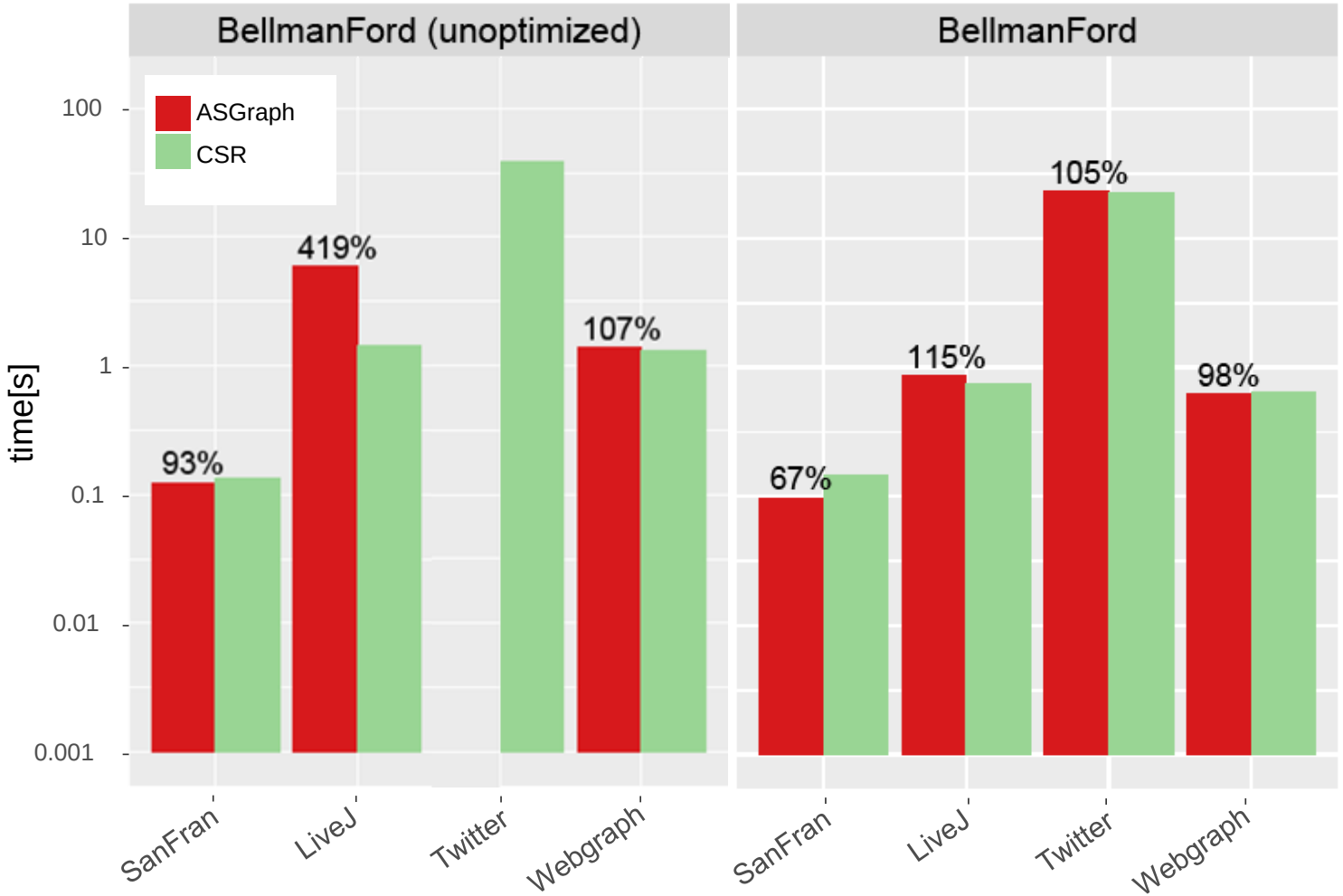
Disadvantages

- Neighbors can be accessed only via forward iteration, no random access
  ➔ Common neighbor iteration profits heavily from random access
- High memory consumption
  - Additional header data per bucket
  - Fill factor of the buckets
  - Timestamps and OpTypes

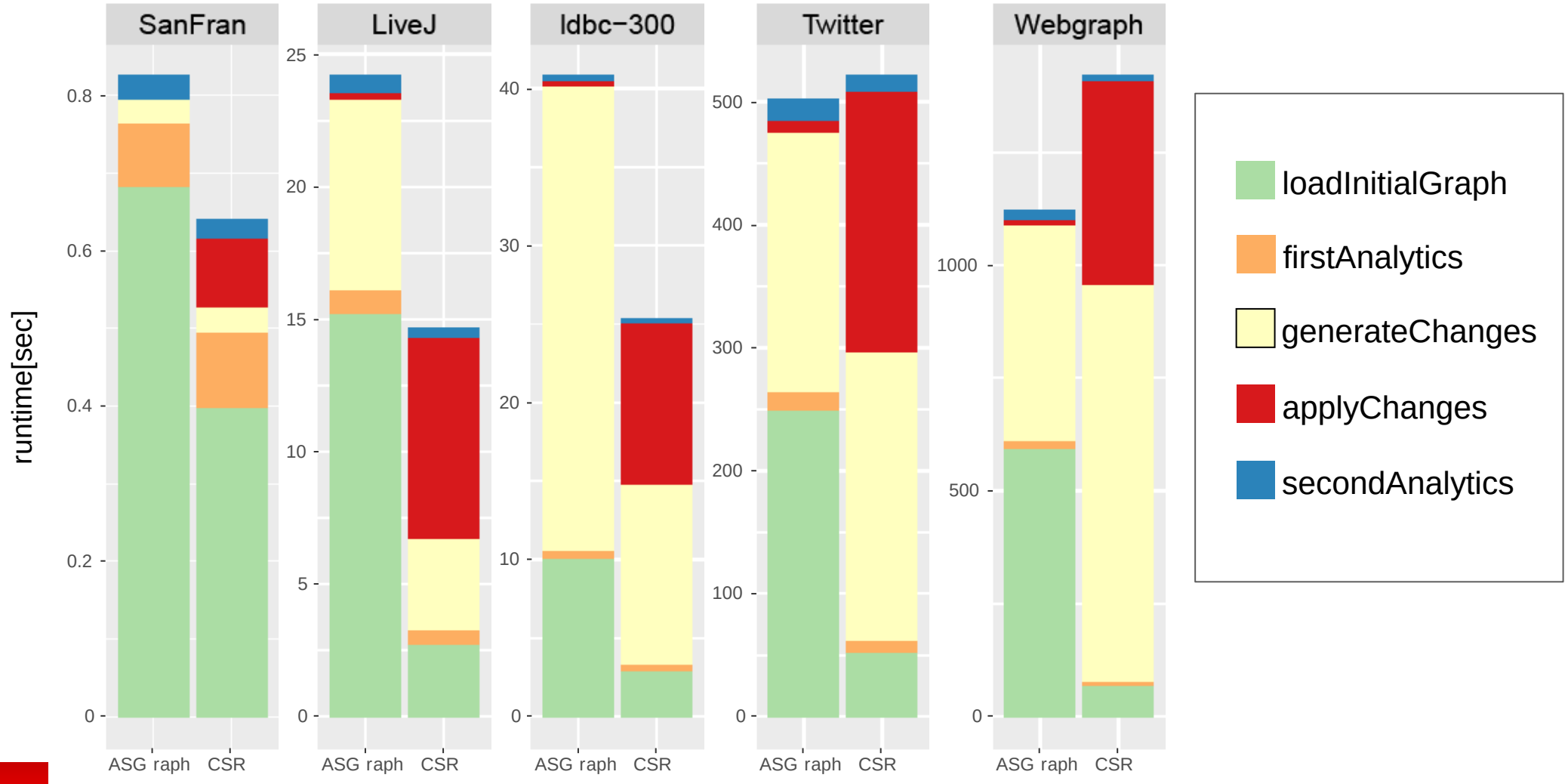
Evaluation – Algorithm Comparison



Evaluation – Algorithm Comparison



Evaluation – Complex Scenario



Summary

- ASGraph has performance in the same order of magnitude as CSR
- Performance independent on #snapshots
- Concurrent updates during analysis
- Outperforms CSR when graph is updated between multiple analysis

Future Work

- Resizing of the node directory
- Concurrent access to multiple versions
- Re-Mapping of vertex ids to a dense range
- Distributed version of ASGraph
- Reverse bucket lists for stable update performance

ORACLE

Questions?


ORACLE®