IIT Hyderabad

# Non-blocking Dynamic Unbounded Graphs with Wait-Free Snapshot

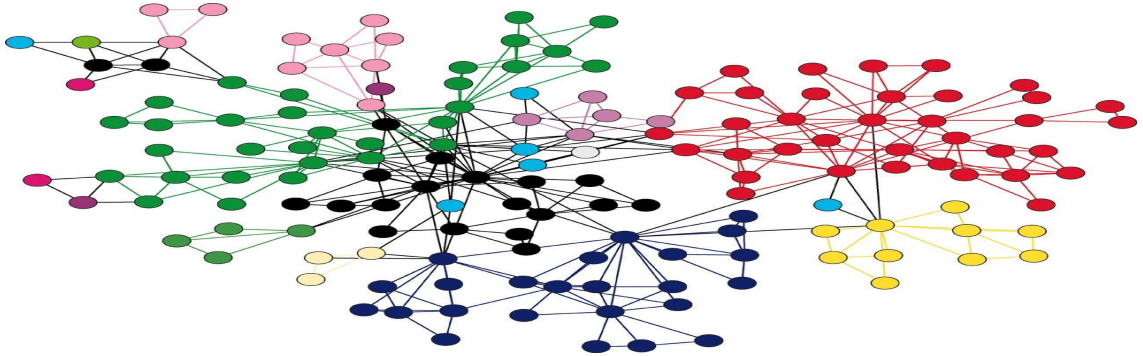
Gaurav Bhardwaj, Sathya Peri, Pratik Shetty

**Presented by: Dr. Sathya Peri**

# Contents

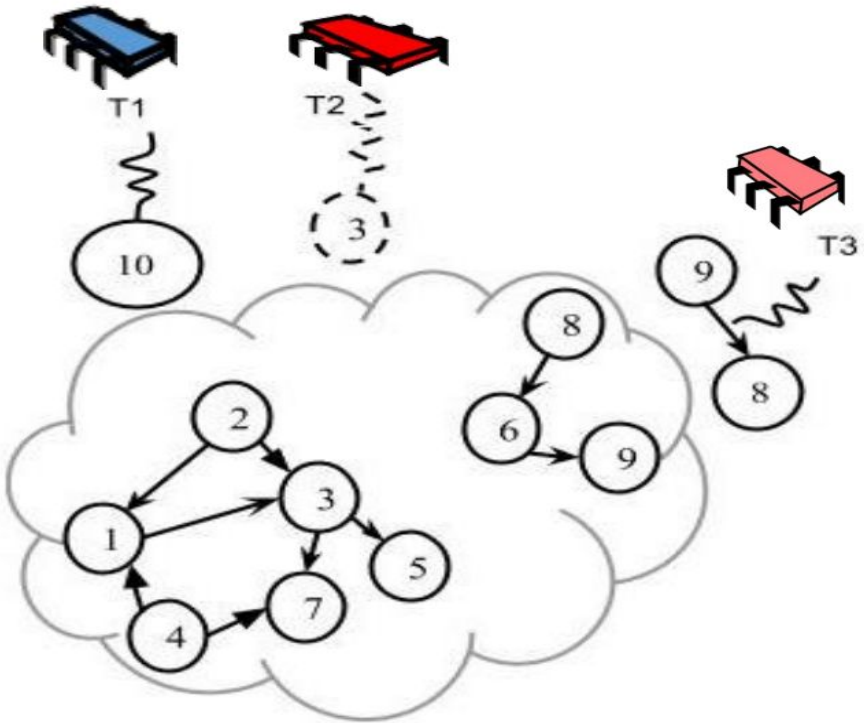
# Graphs

- Graphs provide a visual representation of relationships and connections in data, making complex structures easier to understand.
- Graphs are ideal for modeling various relationships such as social networks, web links, and dependencies between tasks.
- Graphs allow for flexible data representation, enabling the modeling of diverse scenarios and real-world interactions.



# Concurrent Graphs:

- In applications like social media analytics and network monitoring, concurrent graphs enable real-time analysis of dynamic data.
- Concurrent graphs are essential in large-scale systems, ensuring scalability and efficient processing of data in parallel.
- Concurrent graphs optimize resource utilization in multi-core processors, enhancing the performance of graph-based algorithms.



# Concurrent Graph Operation

- Add Vertex / Add Edge
- Del Vertex / Del Edge
- Lookup Vertex / Lookup Edge
- Graph Snapshot
  - **Betweenness Centrality**
  - **Diameter**
  - All Pair Shortest Path
  - Page Rank

# Snapshot

- To perform graph analytics operations such as BFS, getpath, SSSP etc. in a concurrent setting consistent snapshot of the dynamic graph is required.
- Once, we have a consistent snapshot of the graph, one can perform analytics on the snapshot.
- Without consistent snapshot, the analytics may not useful

# Correctness (Consistency)

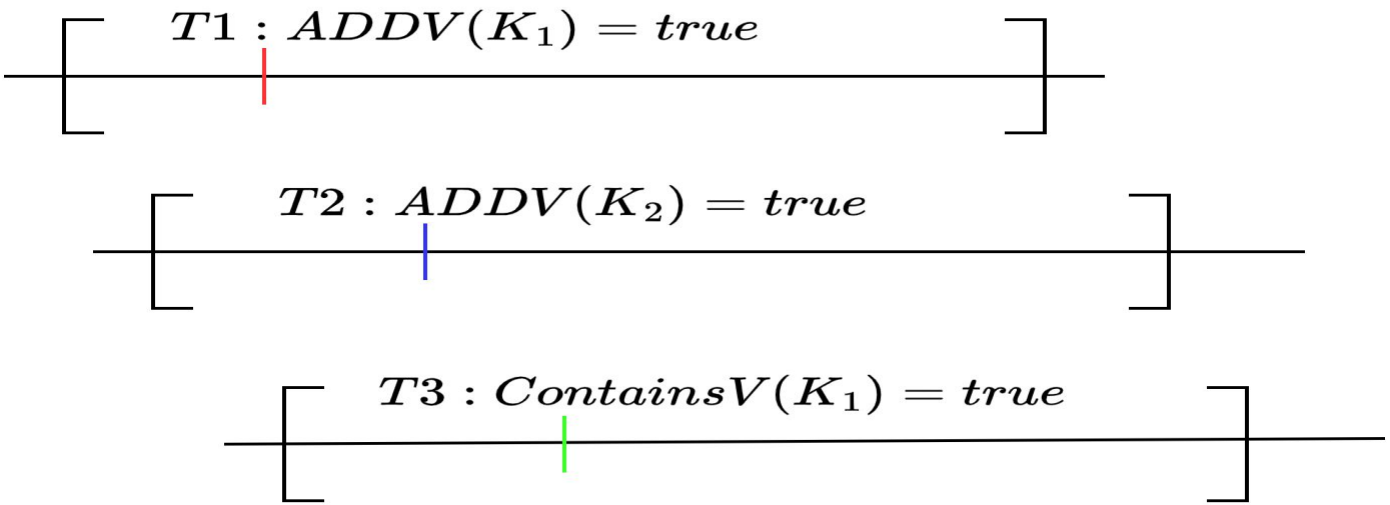
- The ADT operations implemented by the data structure are represented by their invocation and return steps.
- For an arbitrary concurrent execution of a set of ADT operations should satisfy the consistency framework **Linearizability** [a] .

[a] **"Linearizability: a correctness condition for concurrent objects."** Herlihy, M.P., Wing, J.M.: ACM Trans. Program. Lang. Syst. (TOPLAS) 12(3), 463–492 (1990)

# Linearizability

- Assign an atomic step as a linearization point (LP) inside the execution interval of each of the operations and show that the data structure invariants are maintained across the LPs.
- An arbitrary concurrent execution is equivalent to a valid sequential execution obtained by ordering the operations by their LPs.
- A concurrent object is correct if all its executions are linearizable.

# Linearizability Example



# Non-Blocking Progress Condition

An execution is said to be Non-Blocking if it doesn't blocks the execution of other threads.

- **Obstruction Free:** A thread is guaranteed to finish in a finite number of steps in isolation.
- **Lock-Freedom:** At least one thread should be able to finish in the finite number of steps.
- **Wait-Freedom:** All threads should be able to finish in a finite number of steps.

# Literature Work

- Chatterjee et al.[b]
  - Proposed a Lock-Free dynamic unbounded unweighted graph.
  - Used Lock-Free Linked Lists to store vertices and edges.
  - Used obstruction free method to perform the partial snapshot on the graph.

- Chatterjee et al.[c]
  - Extended the previous work for speed up.
  - Proposed a Lock-Free dynamic unbounded weighted graph.
  - Used Lock-Free Hash Tables to store vertices and Binary Search tree to store edges.
  - Used obstruction free method to perform the partial snapshot on the graph.

[b] **"A Simple and Practical Concurrent Non-blocking Unbounded Graph with Reachability Queries",** Bapi Chatterjee, Sathya Peri, Muktikanta Sa, Nandini Singhal in the 20th International Conference on Distributed Computing and Networking (ICDCN), Bangalore, India, January 2019.

[c] **"Non-blocking dynamic unbounded graphs with worst-case amortized bounds",** Bapi Chatterjee, Sathya Peri, Muktikanta Sa, Manogana In: International Conference on Principles of Distributed Systems (2021)

# Drawback:

- Current Literature supports obstruction free snapshot.
- As a result, a snapshot operation may continuously get interrupted
  - Hence may never terminate.

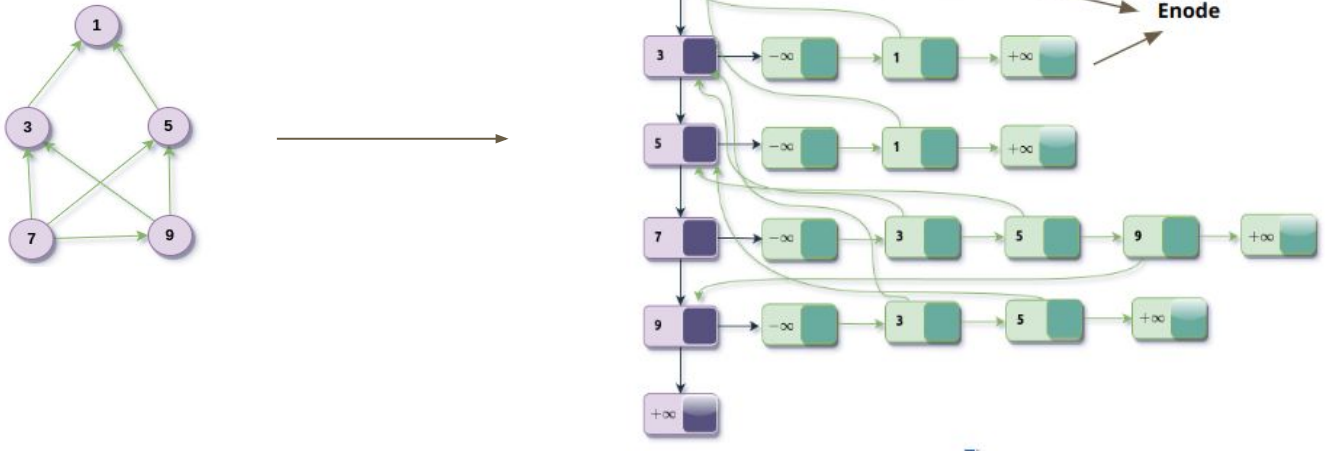
IIT Hyderabad

# Motivation

- To Construct a **wait-free** Snapshot for an Unbounded graph while allowing concurrent non-blocking (lock-free) *update* methods.

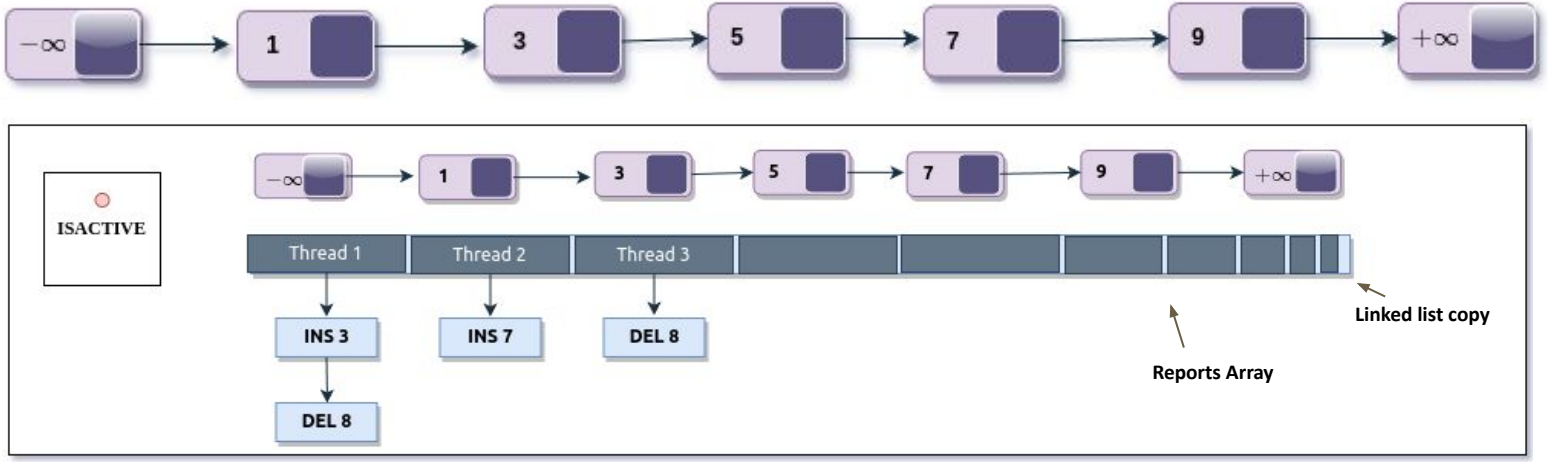
# Unbounded Concurrent Graph

- Part of Chatterjee et al's concurrent framework for unbounded graphs.
- The vertices of graph are stored as linked List of *Vnodes* ordered by their key.
- Similarly the edges associated with each vertex are stored as linked list of *Enodes*.



# Snapshot of Linked List using Iterator [d]

- Iterates through the data structure and makes a copy.
- Collects reports of operation running concurrently.
- Multiple iterator can run concurrently.




[d] **"Lock-free data-structure iterators."** Petrank, E., Timnat, S.: In: Afek, Y. (ed.) DISC 2013. LNCS, vol. 8205, pp. 224–238. Springer, Heidelberg (2013).

# Snapshot of Graph using Iterator

- After creating the *SnapCollector* Object We start by copying all the *Vnodes* of the graph by creating the corresponding object *SnapVnode*.

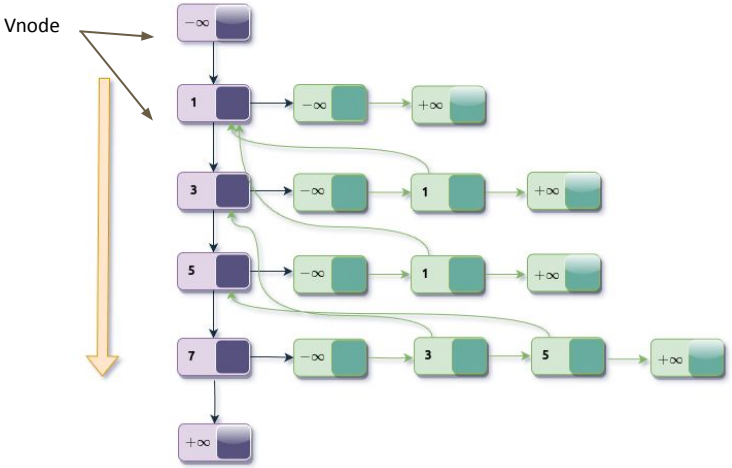



Fig. Graph Data Structure

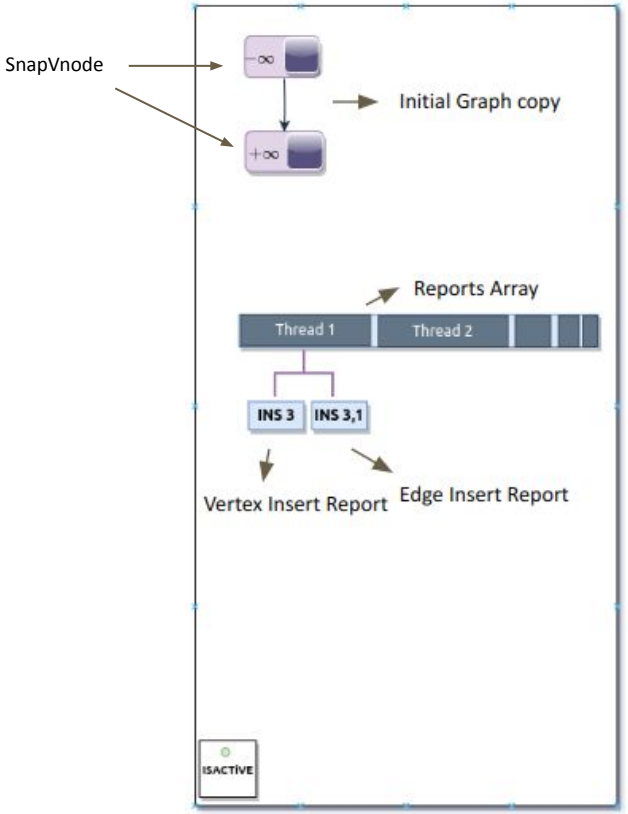
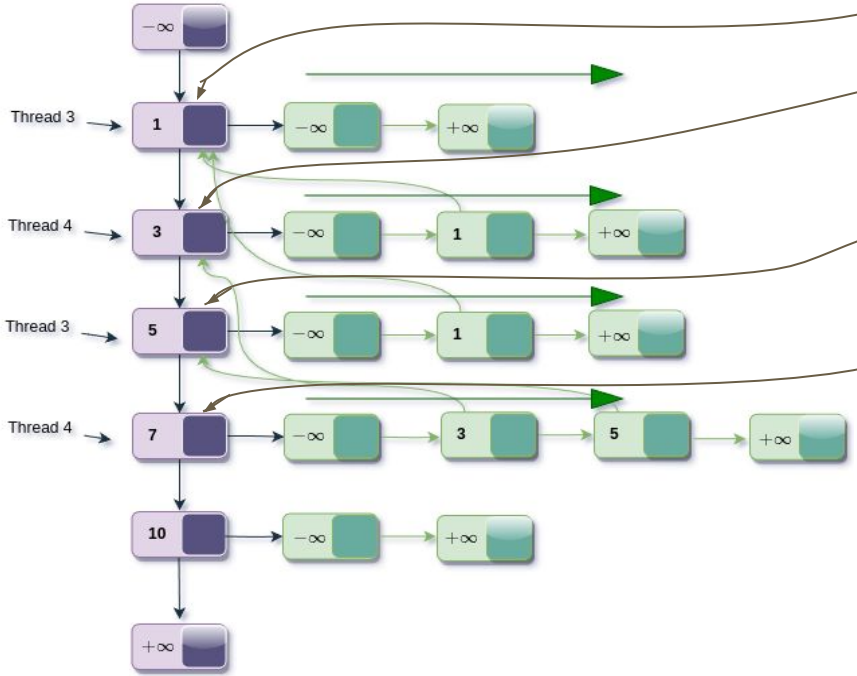
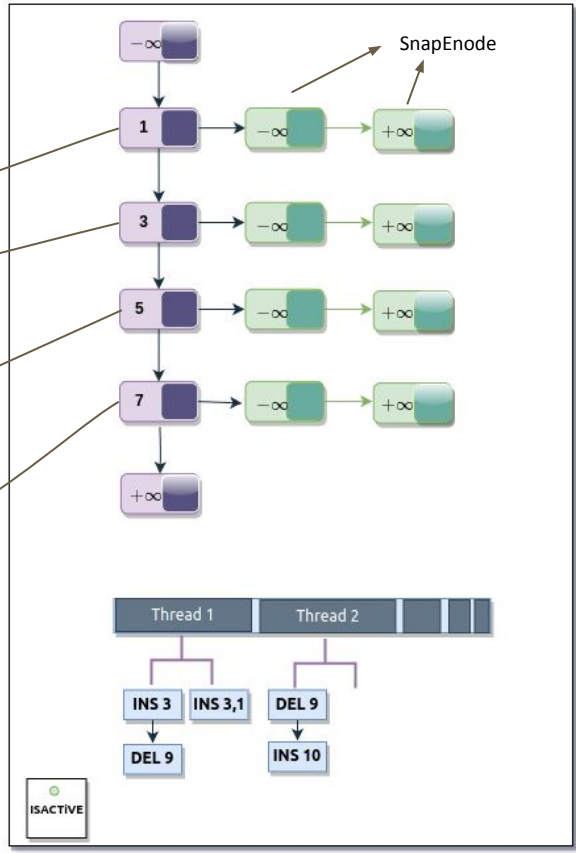


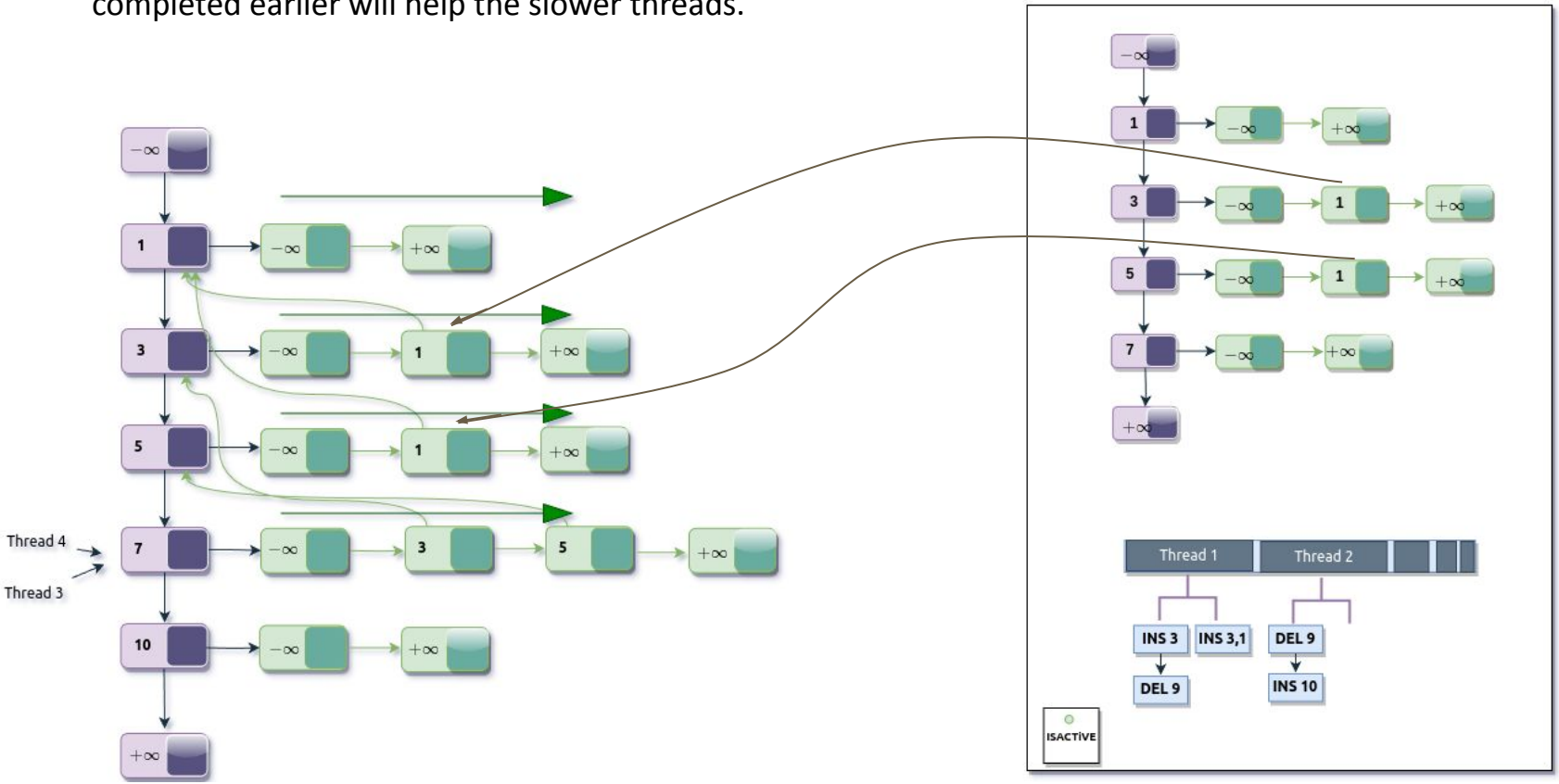

Fig. SnapCollector Object

- The *Vnodes* collected in one pass are then assigned to threads that are performing snapshot concurrently.
- Any Vnodes added later will be part of the report and will be added during *reconstruction*. Eg. **Thread 2 -> INS 10**
- Each thread will iterate through the *Enodes* and create the corresponding *SnapEnode* in *SnapCollector* Object.

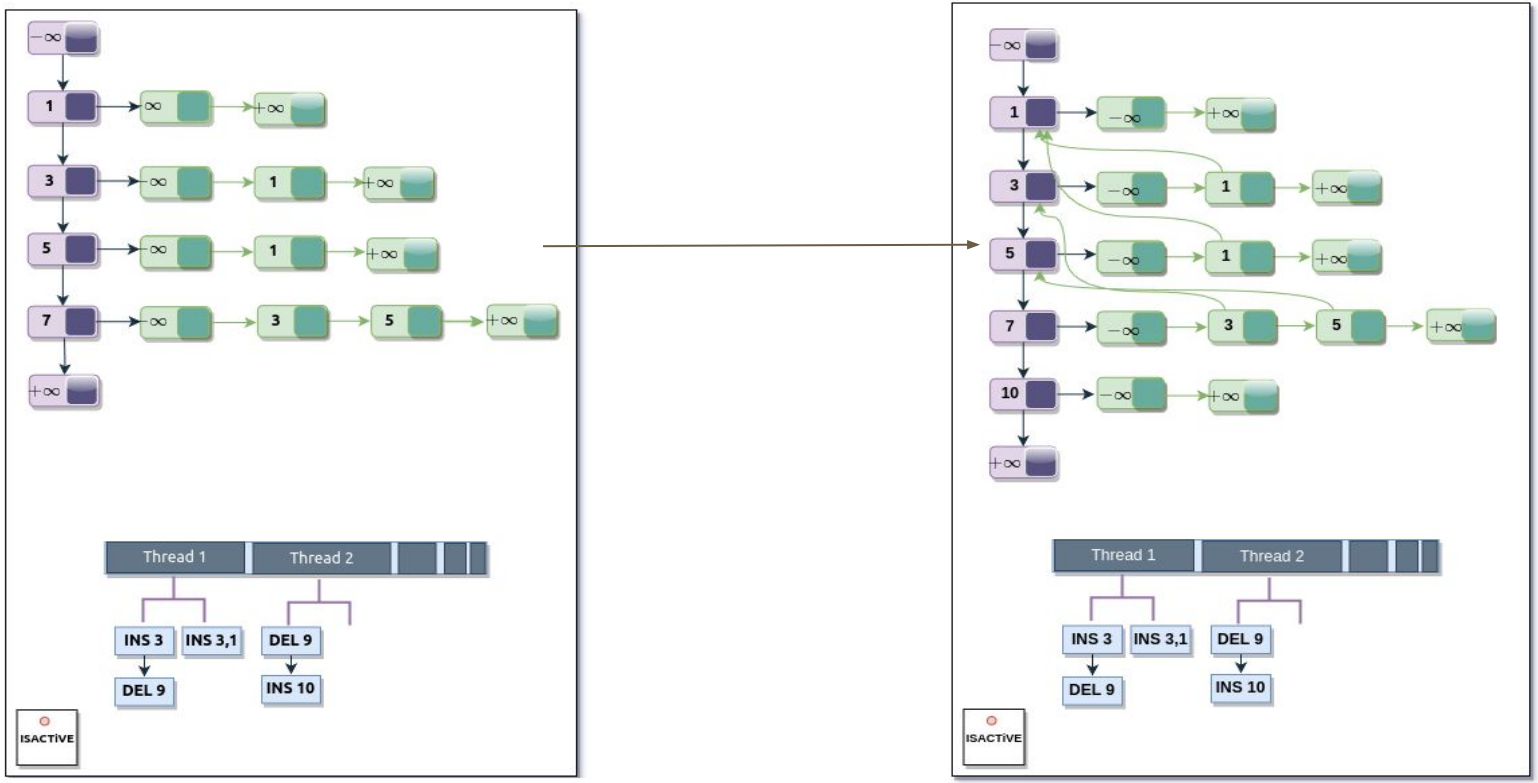


- If any thread is slow. Then the threads that completed earlier will help the slower threads.



- Once a partial copy of the graph is created the *SnapCollector* is deactivated using flag *IsActive.*
- The updates performed by other concurrent threads is recorded using report.
- These reports are used to reconstruct the graph.



# Results and Analysis :

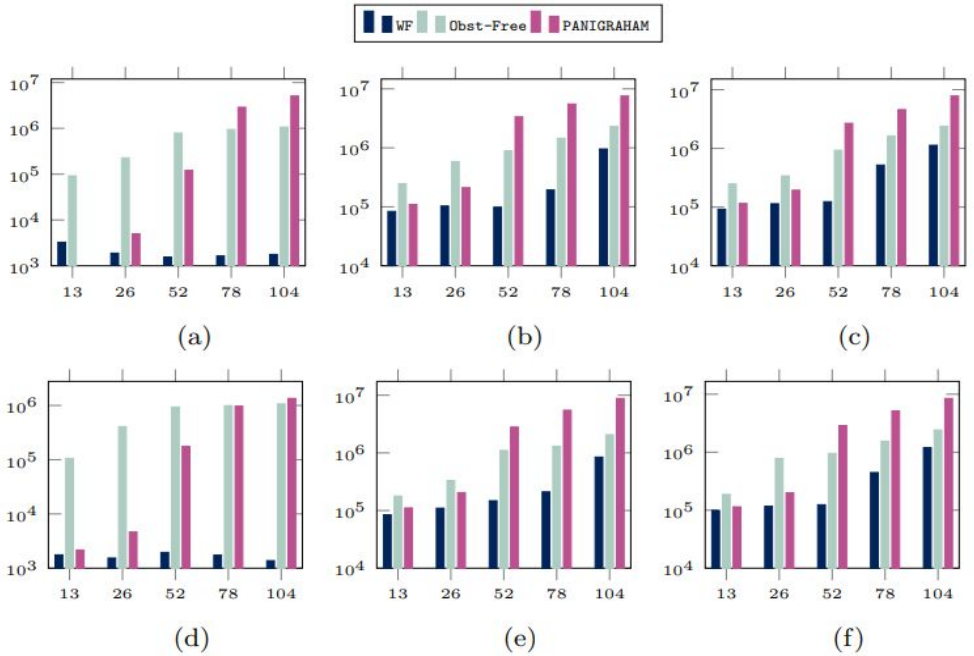
- The evaluation metric used is *Average Time.* We measure *Average time* w.r.t
  - Increasing spawned threads
  - Increasing the workload to critical operations(Snapshot/Diameter/Betweenness Centrality)
  - Multiple Snap graph Datasets.

- **Workload Distribution :** The distribution is over the following ordered set of Operations (AddVertex, RemoveVertex, ContainsVertex, AddEdge, RemoveEdge, ContainsEdge, and Critical Operation(Snapshot/Diameter/Betweenness Centrality).
  - Read Heavy Workload : 3%, 2%, 45%, 3%, 2%, 45% , 2%
  - Update Heavy Workload: 12%, 13%, 25%, 13%, 12%, 25% , 2%

- **Algorithms :** We compare our implementation to Obstruction-free implementations of same operations using Chatterjee et al. namely
  - **Obst-Free: "A Simple and Practical Concurrent Non-blocking Unbounded Graph with Reachability Queries",** Bapi Chatterjee, Sathya Peri, Muktikanta Sa, Nandini Singhal in the 20th International Conference on Distributed Computing and Networking (ICDCN), Bangalore, India, January 2019.
  - **PANIGRAHAM: "Non-blocking dynamic unbounded graphs with worst-case amortized bounds",** Bapi Chatterjee, Sathya Peri, Muktikanta Sa, Manogana In: International Conference on Principles of Distributed Systems (2021)

# Results and Analysis :

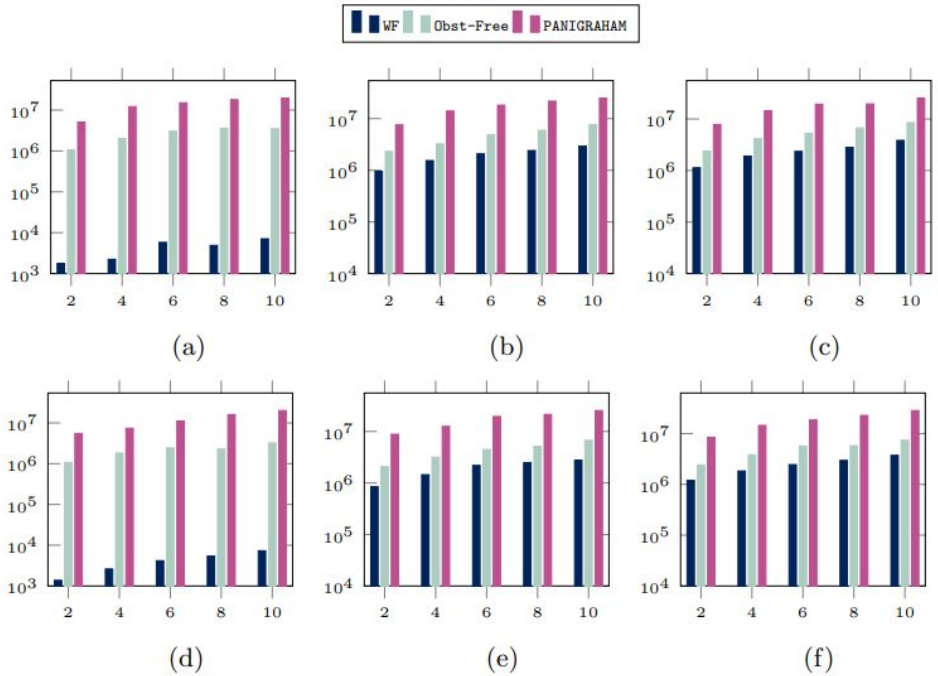
- Read heavy and Update heavy performance of our implementation to its counterparts
  - X-axis : No of threads ,
  - Y-axis : Average time taken(µs)




(a) Read Heavy workload with snapshot, (b) Read Heavy workload with Diameter, (c) Read Heavy workload with Betweenness Centrality, (d) Update Heavy with snapshot, (e) Update Heavy with Diameter, (f) Update Heavy with Betweenness Centrality.

# Results and Analysis :

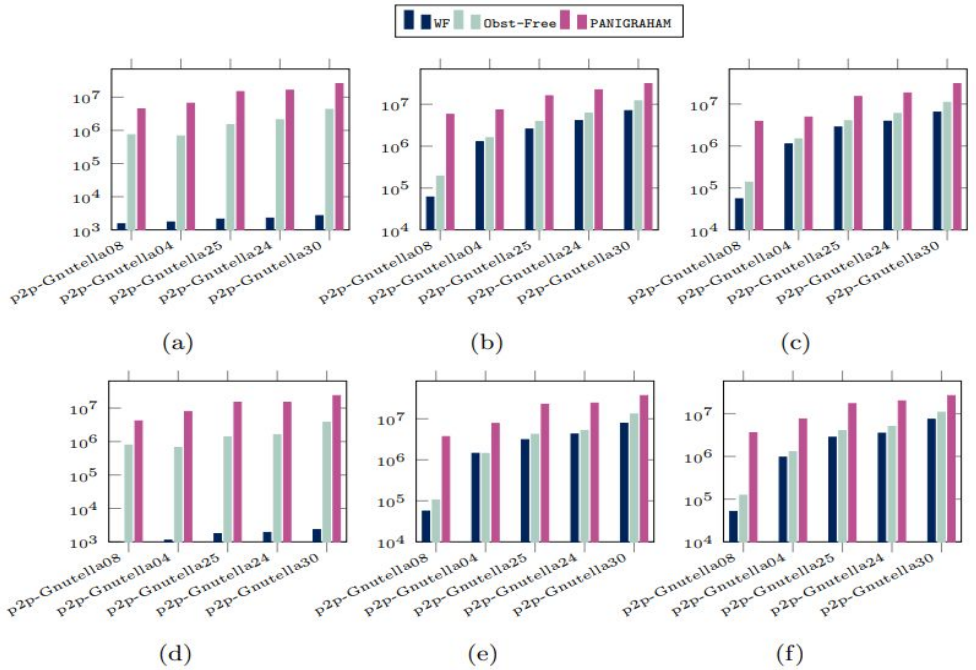
- X-axis : Percentage of Graph Analytics Operations
  Y-axis : Average time taken(µs)




(a)Read Heavy workload with snapshot, (b) Read Heavy workload with Diameter, (c) Read Heavy workload with Betweenness Centrality, (d) update Heavy with snapshot, (e) Update Heavy with Diameter, (f) Update Heavy with Betweenness Centrality.

# Results and Analysis :

- X-axis : Snap graph Datasets.
  Y-axis : Average time taken(µs)




(a)Read Heavy workload with snapshot, (b) Read Heavy workload with Diameter, (c) Read Heavy workload with Betweenness Centrality, (d) update Heavy with snapshot, (e) Update Heavy with Diameter, (f) Update Heavy with Betweenness Centrality.

# Conclusion and Future Work

**Conclusion:**

- Our proposed solution is the first of its kind that supports the concurrent wait-free snapshot.
- Our solution outperforms the current existing solutions for graph analytics operations such as
  - Snapshot, Diameter and Betweenness Centrality.

**Future Work:**

- Partial Snapshots are relevant for several graph analytics such as SSSP, BFS and are efficient.
  - The solution proposed takes a complete snapshot
  - How can the current solution be extended to obtain the partial snapshot.
- Update operations on the graph can be modified to get the maximum progress guarantee i.e. wait-freedom.