PCDS2024

The 1st International symposium on
Parallel Computing and Distributed Systems

Sept. 21-22 | 2024 Singapore

TRENT UNIVERSITY

Paper ID: PD104

New Parallel Order Maintenance Data Structure

Bin Guo (binguo@trentu.ca, Trent University, ON Canada)
Emil Sekerinski (emil@mcmaster.ca, McMaster University, ON Canada)

Parallel Order Maintenance (OM) Data Structure

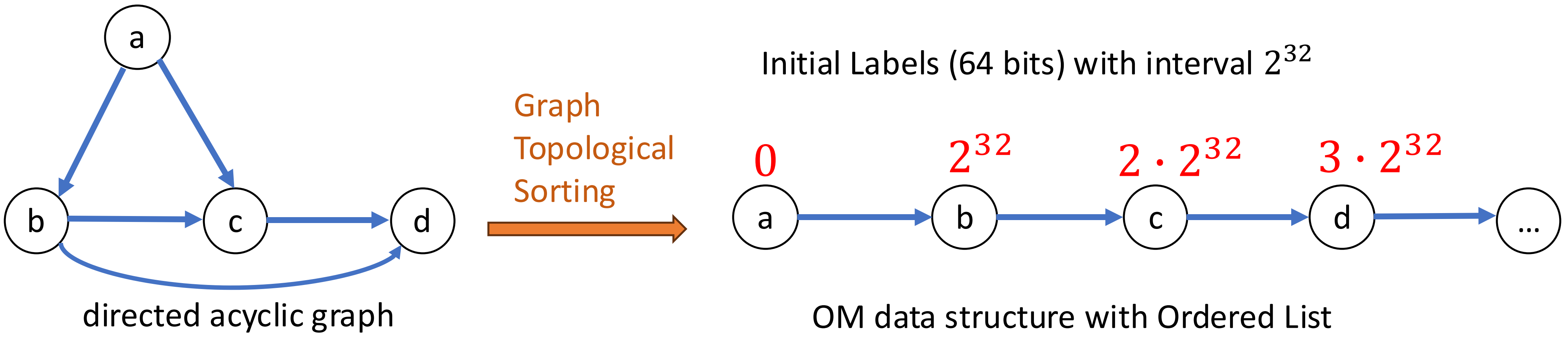
- Maintain a total order of unique items in a list, denoted by $\mathbb{O}$
- Three operations
  - **Order**(**x**, **y**): if **x** precedes **y** in the order list $\mathbb{O}$
  - **Insert**(**x**, **y**): insert **y** after **x** in $\mathbb{O}$
  - **Delete**(**x**): delete **x** from $\mathbb{O}$
- The naïve implementation is to use Balanced Binary Search Tree
- Dietz et al. propose the OM data structure [1,2]
  - use labels to comparing

Compare The time complexities

	Naïve Balance Binary Search Tree	**OM data Structure [1,2]**
Order	$O(\log N)$	$O(1)$
Insert	$O(\log N)$	Amortized $O(1)$
Delete	$O(\log N)$	$O(1)$

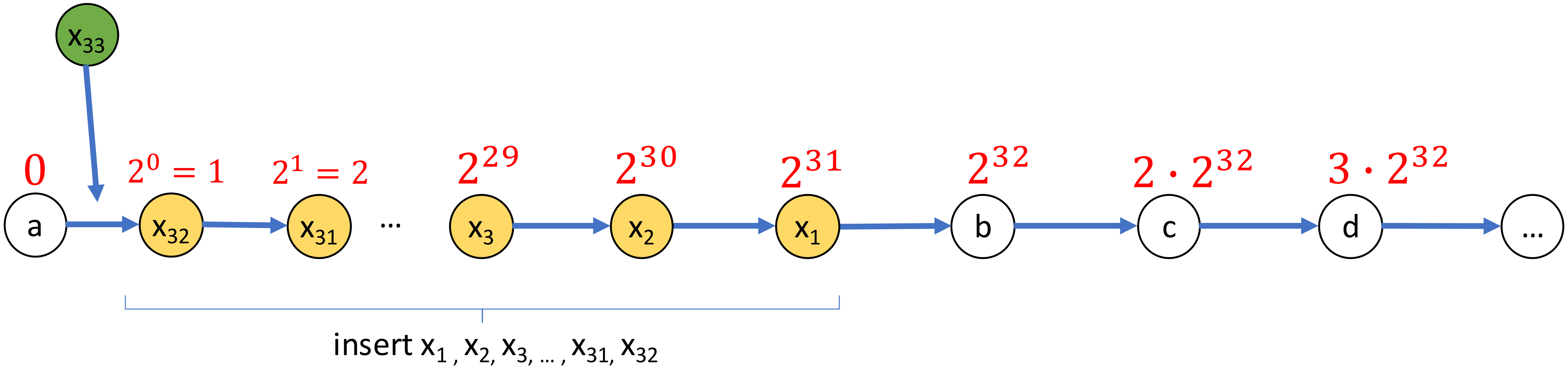
For N items in total

Examples: Order and Delete



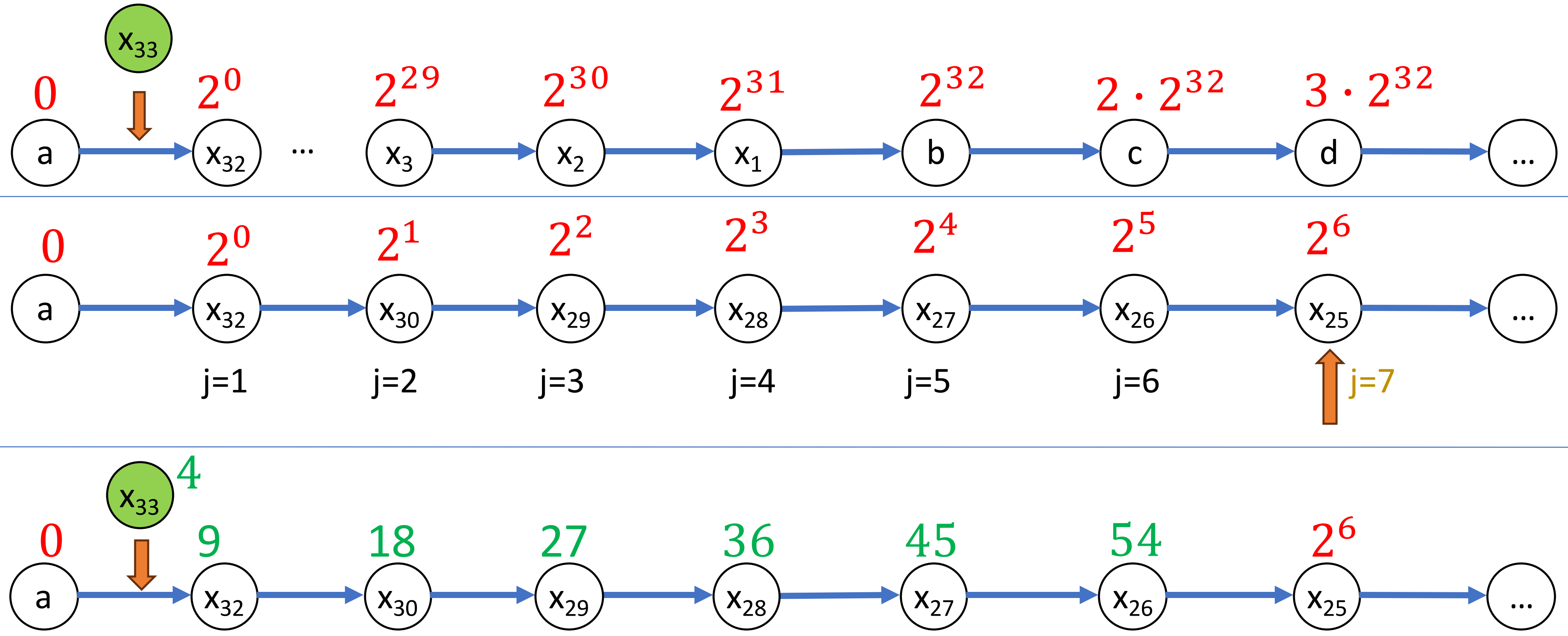
- Labels indicate the order of vertices
- **Order**(**a, b**) by comparing labels, $0 < 2^{32}$, so **a** is ahead **b**
- **Delete**(**b**) with not affect the labels

Examples: Insert



- **Insert**(**a**, **x**): **x** is in the middle between **a** and **a.next**
- At most 32 items can **Insert** after **a**, without changing labels
- It will trigger the **Relabel** operation when insert x_{33}

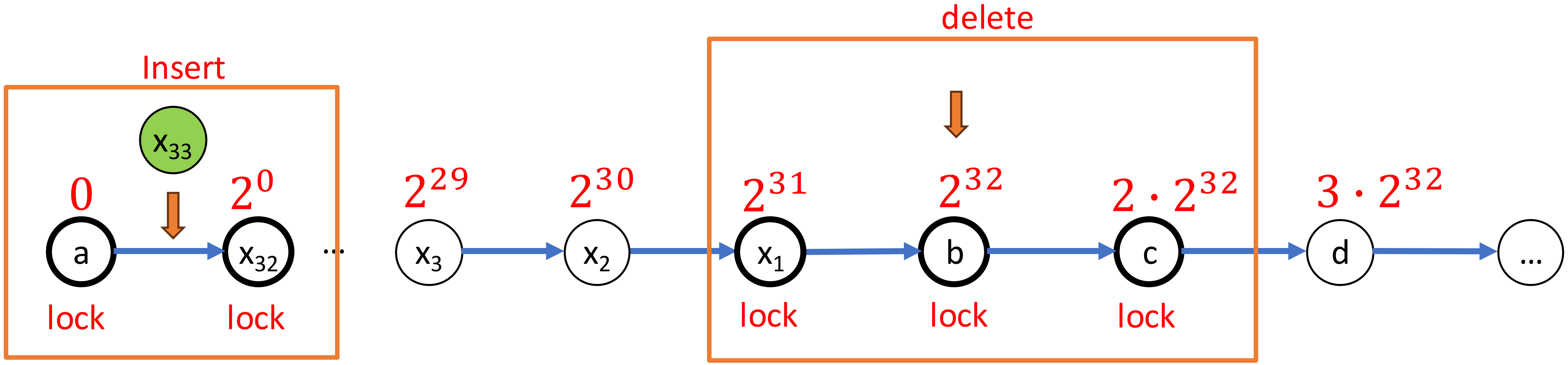
Examples: Relabel



- **Relabel(a)**: start from a, find the gap that is $L(x_n) - L(a) > j^2$ for traversing ***j*** items
  - Find x_{25} with j = 7, so that $2^6 - 0 = 64 > 7^2 = 49$
- **Relabel** from x_{31} to x_{26}, then insert x_{33} with label 4
- The amortized running time is $O(\log N)$. Can be reduce to amortized $O(1)$ by using groups (details in my paper)

Our Contribution: Parallel OM data structure

- **Parallel-Delete** and **Parallel-Insert**
  - In the double-linked list, we lock the related items

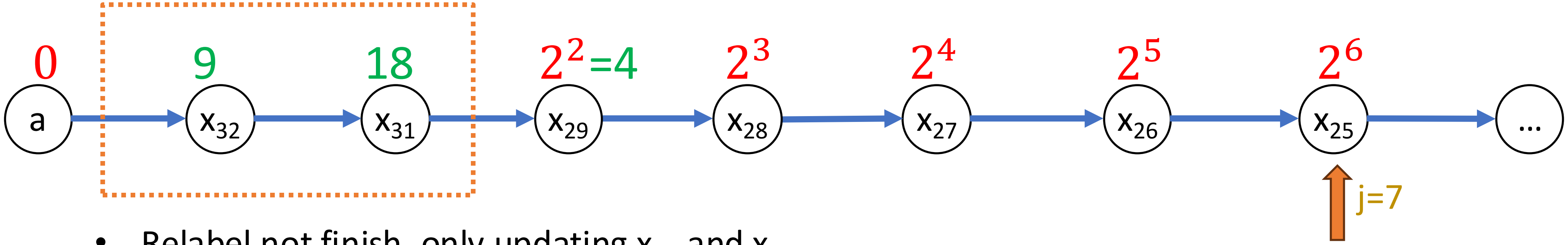


- Both lock **a** and **x_{32}** when inserting **x_{33}**
- Assign **x_{33}** a new label
- The relabel process is triggered, which also need to lock related vertices

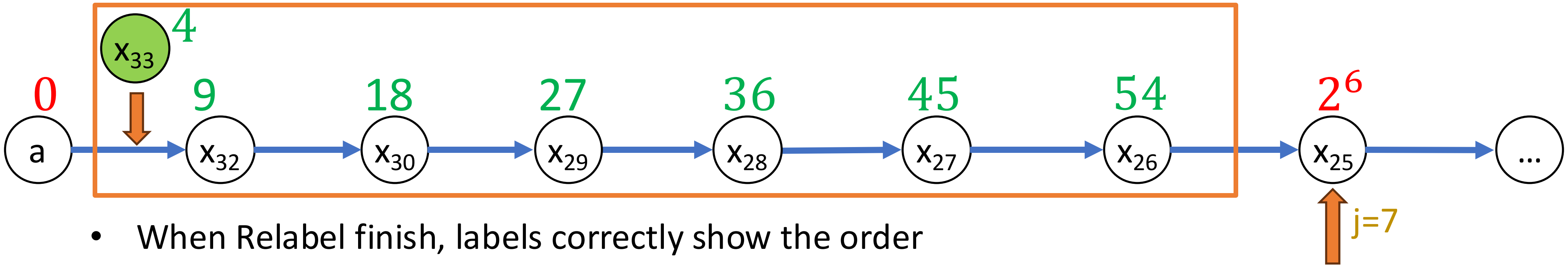
- We lock **x_1** , **b** and **c** when deleting **b**
- The labels are not affected

Our Contribution: Parallel OM data structure (2)

- We desire lock-free **Parallel-Order** Operation
  - The **Relabel** may create labels that not correctly represent the order.



- Relabel not finish, only updating x_{31} and $x_{30,}$
- The labels are incorrect to show order



- When Relabel finish, labels correctly show the order
- For parallel **Order** and **Insert** operations, the labels must correctly show the order at any time

Our Contribution: Parallel OM data structure (3)

- We propose a new **Relabel** operation
- The **Relabel** with reverse order from the later item

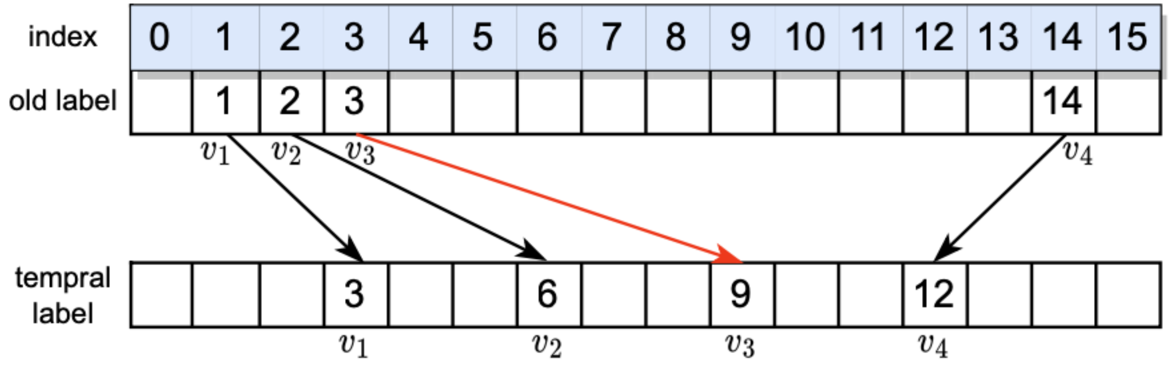


Figure 4. An example of the `AssignLabel` procedure.

- All labels can be correctly indicating the order at any time snap
- The **Parallel Order** is **lock free**

Application: Core Maintenance

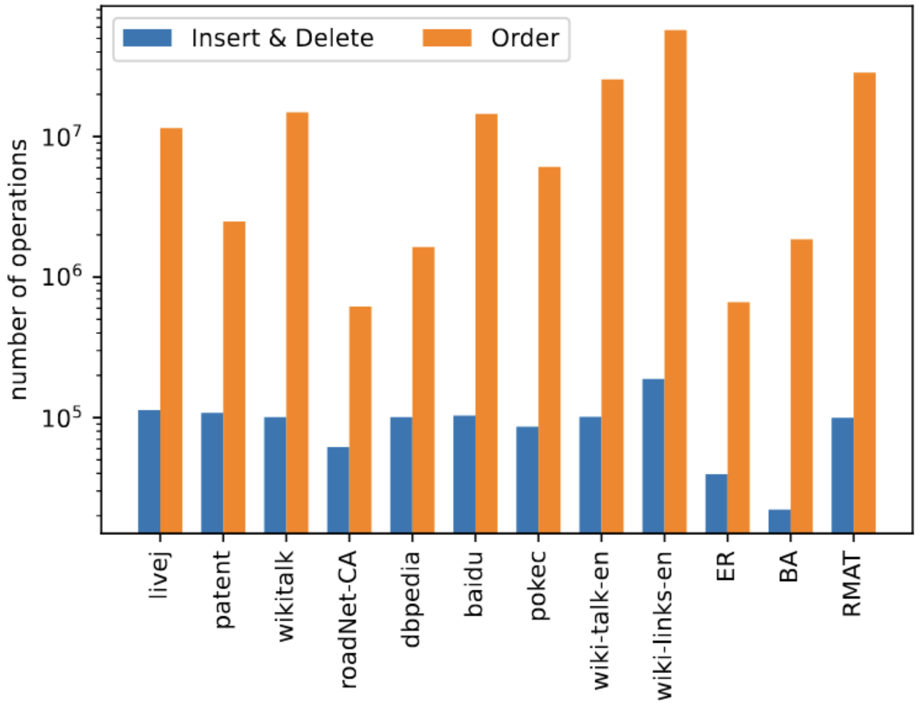


Figure 2. The number of OM operations for core maintenance by inserting 100, 000 random edges into each graph.

- Typically, a large portion (more than 90%) is **Order** operations
- Only small portion (less than 10%) are **Insert** and **Delete** Operations

- This is why lock-free **Order** is meaningful
- It is a break-through for real applications

Experiments with 64 workers

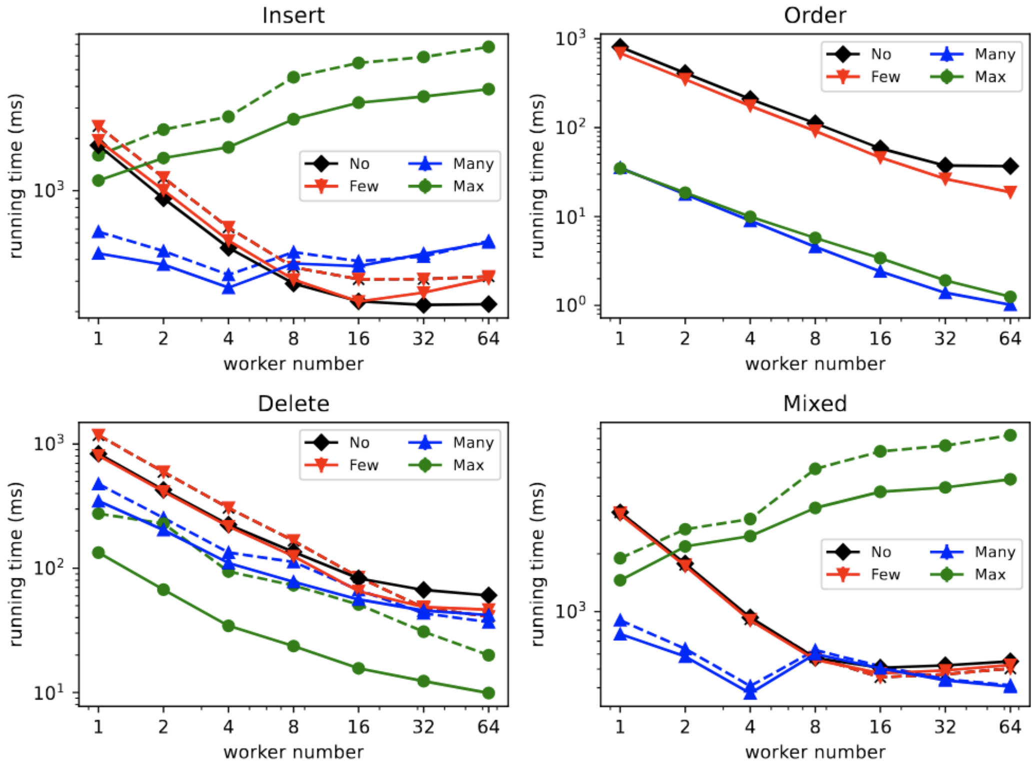


Fig. 1: The running times for *NO*, *FEW*, *MANY*, and *MAX* cases. We have that x-axis shows the number of workers, and the y-axis displays the execution time (in milliseconds), both on an exponential scale.

- **Insert**: insert 10 million items into O.
- **Order**: compare its order of 10 million time.
- **Delete**: delete all inserted items, a total of 10 million
- **Mixed**: insert 10 million items, mixed with 100 million Order operations (simulate in applications)

- **No relabel case**: insert 10 million items into 10 million positions
- **Few relabel case**: insert 10 million items into 1 million positions
- **Many relabel case**: insert 10 million items into 1000 positions
- **Max relabel case**: insert 10 million items into 1 positions

Conclusion

- The parallel **Order** operations achieve the best speedups

- In future, we attempt to make **Insert** and **Delete** as lock-free
  - By using Muti-CAS
- Also, apply parallel OM data structure to many other applications
  - like Ordered Set
  - UML

Reference

- [1] Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In Proceedings of the nineteenth annual ACM symposium on Theory of computing, pages 365–372, 1987.
- [2] Michael A Bender, Richard Cole, Erik D Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In European Symposium on Algorithms, pages 152–164. Springer, 2002.