Многопоточный алгоритм для задачи о динамической связности

Федоров А. И.

Научный руководитель: к.ф.-м.н. Д. Н. Москвин

Научный консультант: Н. Д. Коваль

Санкт-Петербургская школа физико-математических и компьютерных наук

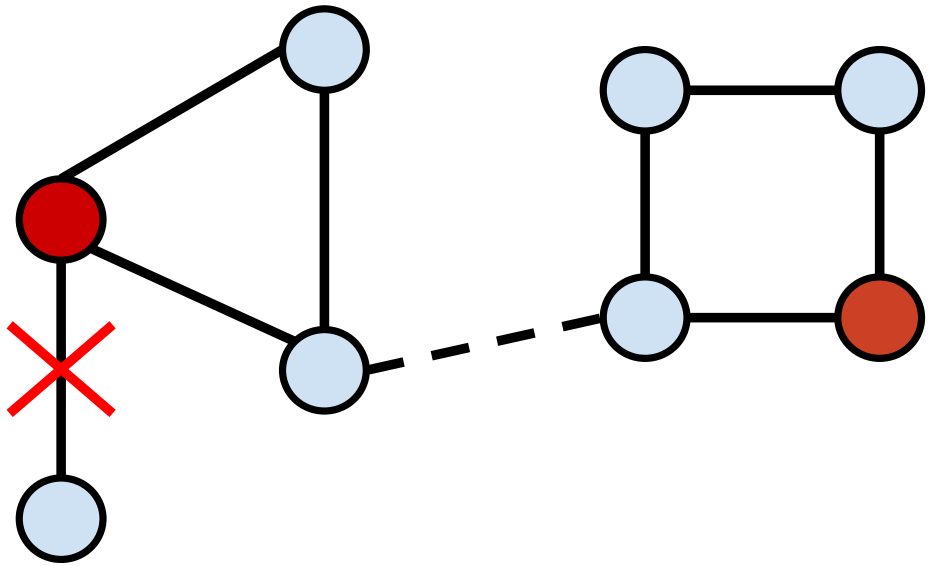
НИУ ВШЭ – Санкт-Петербург

2020

Задача о динамической связности

Дан неориентированный граф.
Требуется поддерживать следующие операции:

- `add_edge(u, v)` – добавить ребро (u, v) в граф;
- `remove_edge(u, v)` – удалить существующее ребро (u, v) из графа;
- `connected(u, v)` – проверить, находятся ли u и v в одной компоненте связности.



Анализ решений: однопоточные

- Первый алгоритм с полилогарифмическим временем работы на операцию[1]
- Классический детермированный алгоритм с средним $\mathcal{O}(\log^2 n)$ на операцию, где n – число вершин графа[2]
- Лучший детерминированный алгоритм: $\mathcal{O}(\frac{\log^2 n}{\log \log n})$ на операцию[3]

---

1 Monika Rauch Henzinger и Valerie King. "Randomized Dynamic Graph Algorithms with Polylogarithmic Time per Operation". B: STOC '95. 1995.

2 Jacob Holm, Kristian de Lichtenberg и Mikkel Thorup. "Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity". B: *J. ACM* (2001).

3 Christian Wulff-Nilsen. "Faster Deterministic Fully-Dynamic Graph Connectivity". B: SODA '13. 2013.

Анализ решений: многопоточные

Основной критерий корректности многопоточных структур данных – линеаризуемость.

Многопоточные алгоритмы:

- Неблокирующий многопоточный алгоритм, умеющий находить пути между вершинами[4]
  - проверка связности асимптотически гораздо медленнее, чем оптимум.
- Алгоритм с техникой parallel combining[5]
  - масштабируется при большом числе проверок связности.
  - мало применим при малом количестве чтений.

---

[4] Bapi Chatterjee и др. "A simple and practical concurrent non-blocking unbounded graph with linearizable reachability qeries". В: ICDCN '19. 2019.

[5] Vitaly Aksenov, Petr Kuznetsov и Anatoly Shalyto. "Parallel Combining: Benefits of Explicit Synchronization". В: *OPODIS*. 2018.

Цели и задачи

Цель

Построение эффективного и линеаризуемого многопоточного алгоритма для задачи о динамической связности.

Задачи

- Выбрать эффективный однопоточный алгоритм с целью последующего его обобщения на многопоточный случай.
- Оптимизировать алгоритм для увеличения параллелизации операций.
- Оценить полученный алгоритм на серии различных сценариев и сравнить с существующими решениями.

Выбор базового алгоритма

Для многопоточного обобщения выбран классический алгоритм Холма и др.[6]

- Может быть улучшен до лучшего известного алгоритма.
- Основан на тех же идеях, что и рандомизированные алгоритмы.
- Близок к оптимуму.
- Среднее время операции важнее времени в худшем случае для практического алгоритма.

[6] Jacob Holm, Kristian de Lichtenberg и Mikkel Thorup. "Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity". В: *J. ACM* (2001).

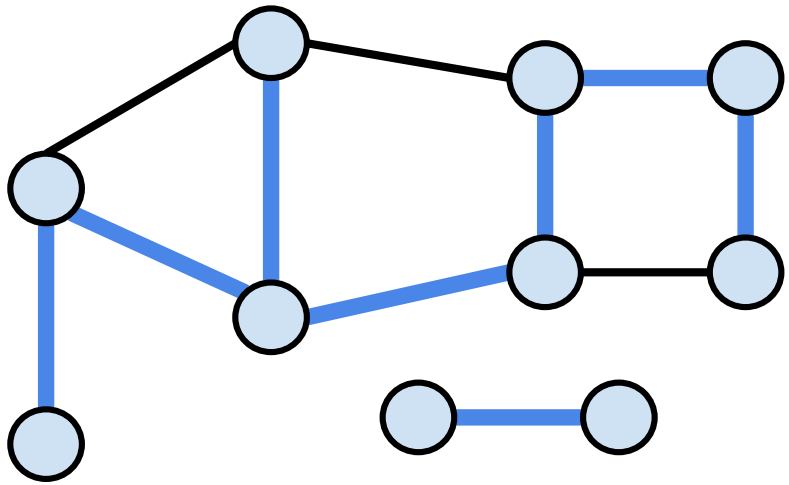
Однопоточный алгоритм[7]

Поддерживает остовный лес.

Деревья Эйлерова обхода – структура данных, которая умеет:

- объединять деревья ребро;
- удалять ребро (дерево разделяется на два);
- проверять, лежат ли две вершины в одном дереве.

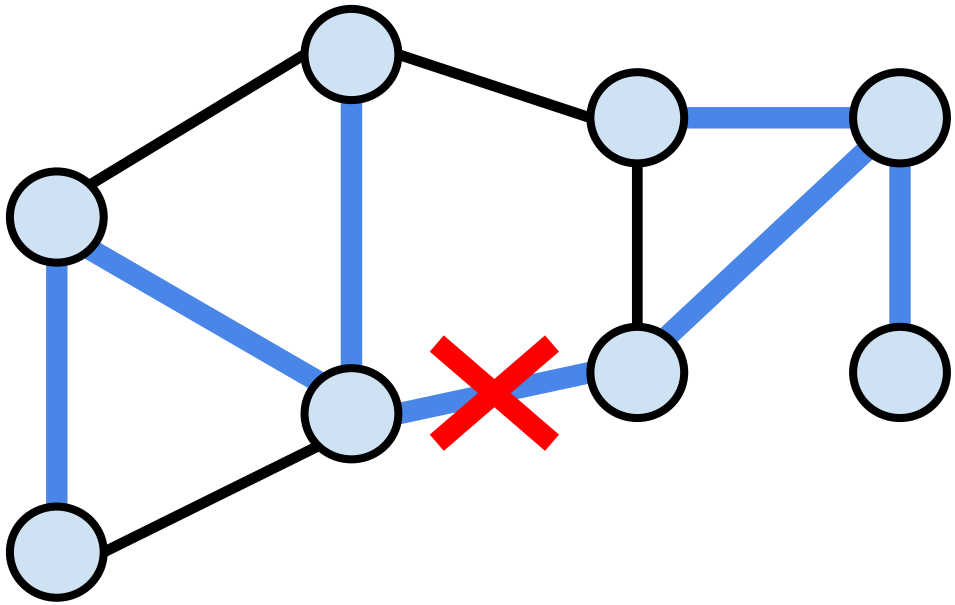
Хранит эйлеровы обходы для каждой компоненты связности в сбалансированном дереве.



[7] Jacob Holm, Kristian de Lichtenberg и Mikkel Thorup. "Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity". В: *J. ACM* (2001).

Общая схема алгоритма

- `connected(u, v)` – проверяет, лежат ли u и v в одном остовном дереве.
- `add_edge(u, v)` – добавляет ребро в деревья эйлерова обхода, если раньше u и v не были в одной компоненте.
- `remove_edge(u, v)` – если ребро было остовным, то нужно искать замену...
  - Холм, Личтенберг и Торуп вводят уровневую структуру, чтобы амортизированное время стало $\mathcal{O}(\log^2 n)$



Алгоритм с грубой блокировкой

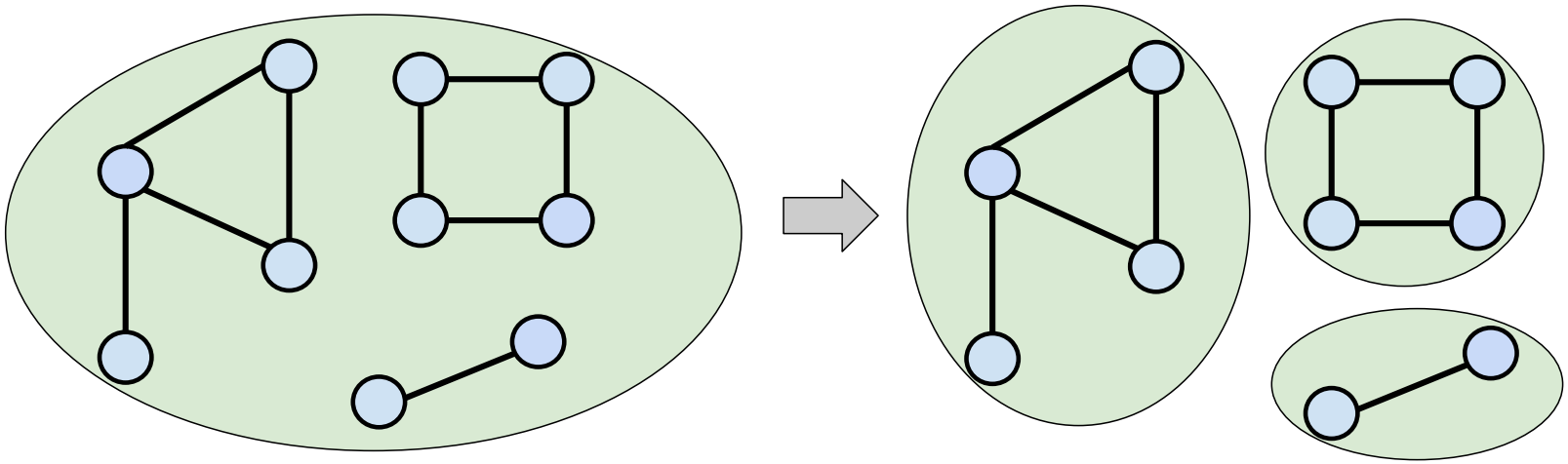
Корректный многопоточный алгоритм с грубой блокировкой:

- Одна глобальная блокировка на всю структуру данных.
- Блокировка берется перед операцией, отпускается – после.
- Нет параллелизации операций $\Rightarrow$ нельзя получить ускорение по сравнению с однопоточным алгоритмом.

Три оптимизации улучшают этот алгоритм с помощью уменьшения количества работы с блокировками.

Тонкая блокировка на компонентах связности

Замена грубой блокировки на тонкую на компонентах связности:



Позволяет естественным образом распараллелить работу над разными компонентами.

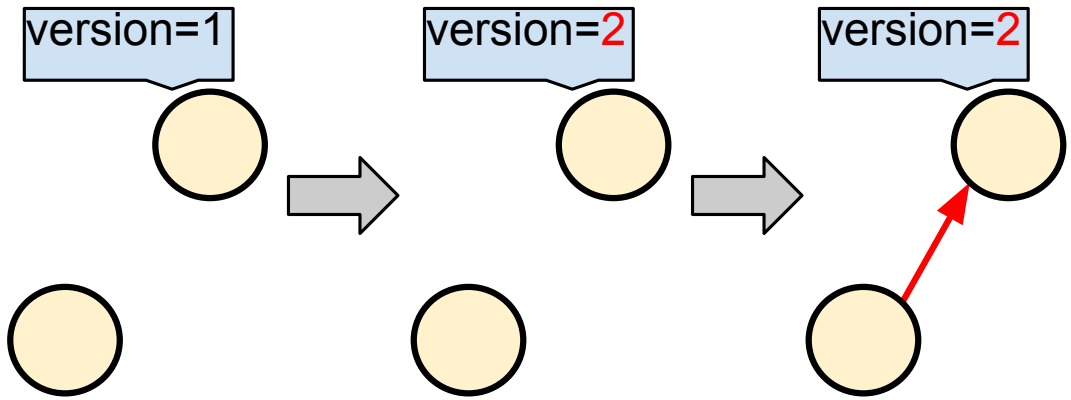
Блокировка на корнях деревьев с последующей проверкой.

У объединения и разделения деревьев выделяются логические и фактические части.

Неблокирующие чтения

Версионирование корней деревьев.

Изменение версий и деревьев происходит неатомарно. Сначала меняется версия, затем – дерево.

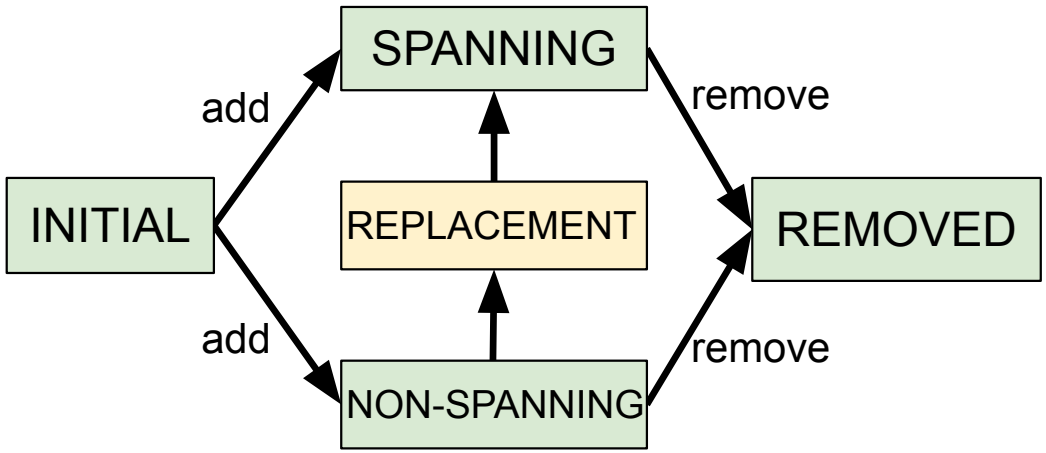


Версия может опережать модификацию дерева, но на не более, чем на одно изменение.

Предложен линеаризуемый неблокирующий (lock-free) алгоритм проверки связности на основе такого версионирования.

Неблокирующие изменения неостовных ребер

Для корректности используется система статусов ребер.

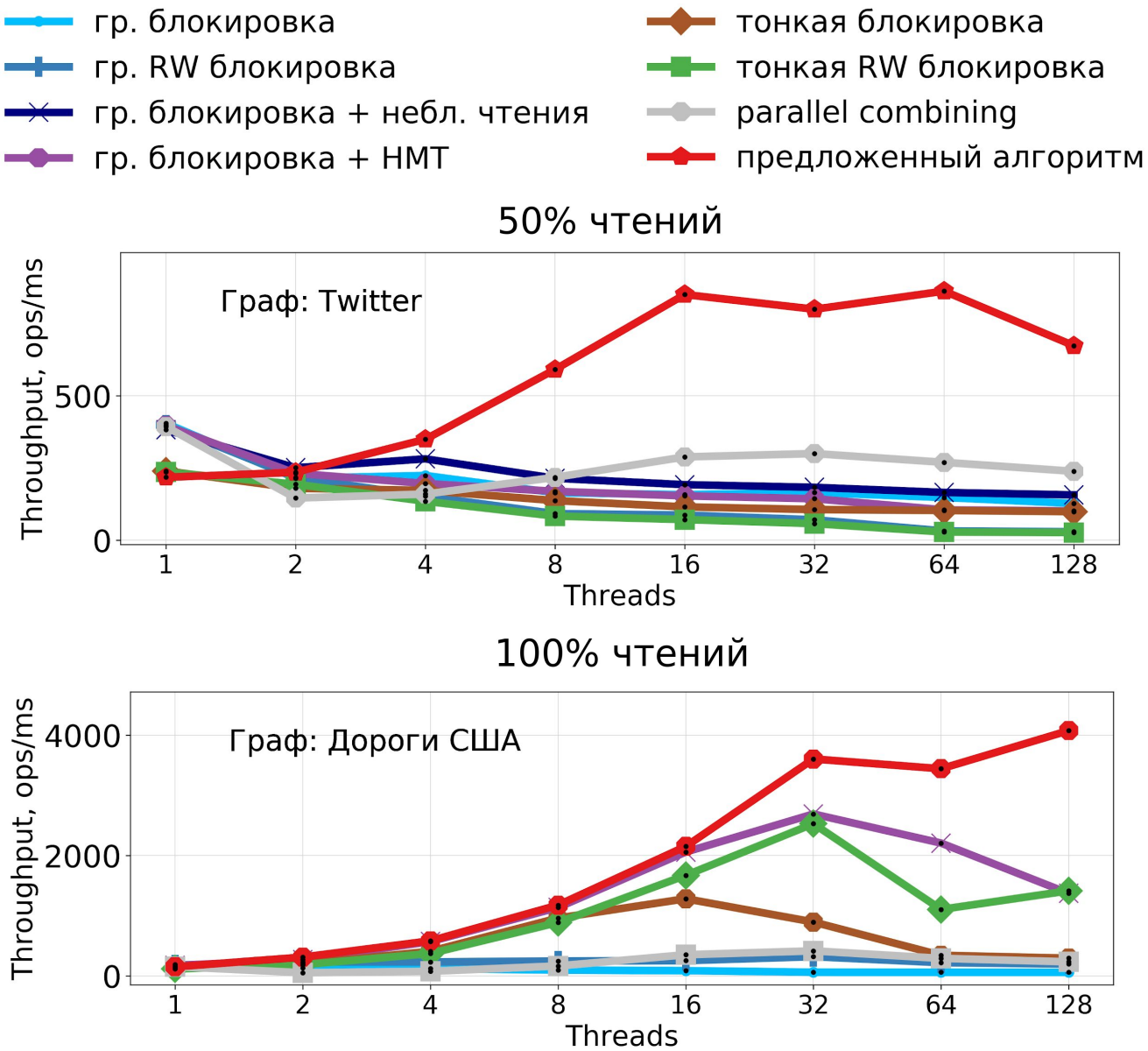


Удаление неостовного: `status.CAS(NON-SPANNING, REMOVED)`.

Операция блокирующего удаления публикуют информацию о себе в корне дерева и помогает добавлять обнаруженные еще не добавленные ребра.

Неблокирующее добавление может предложить ребро в качестве замены для параллельной операции удаления.

Эксперименты



Результаты

Создан многопоточный корректный и эффективный алгоритм на основании алгоритма Холма и др.

Предложены три оптимизации:

- Тонкая блокировка для компонент связности.
- Алгоритм неблокирующей проверки связности.
- Операции, не меняюшие остовный лес, без взятия блокировок.

Экспериментально показана практическая эффективность алгоритма в целом и каждой оптимизации в частности.

Результаты данной работы будут представлены на SPAA 2021 (ACM Symposium on Parallel Algorithms and Architectures).

Реализации и бенчмарки:
https://github.com/alefedor/concurrent-dynamic-connectivity

Практическая оценка

Алгоритмы для сравнения:

1. Грубая блокировка
   - базовый
   - с lock-elision
   - с неблокирующими чтениями на предложенной структуре данных для деревьев
   - readers-writer блокировка
2. Тонкая блокировка
   - базовый
   - readers-writer блокировки
   - с неблокирующими чтениями на предложенной структуре
3. Parallel combining
4. Предложенный алгоритм

Сценарии

Сценарии:

- Случайный сценарий – случайные операции
- Инкрементальный – добавить все ребра графа
- Декрементальный – удалить все ребра графа
- Двухуровневый

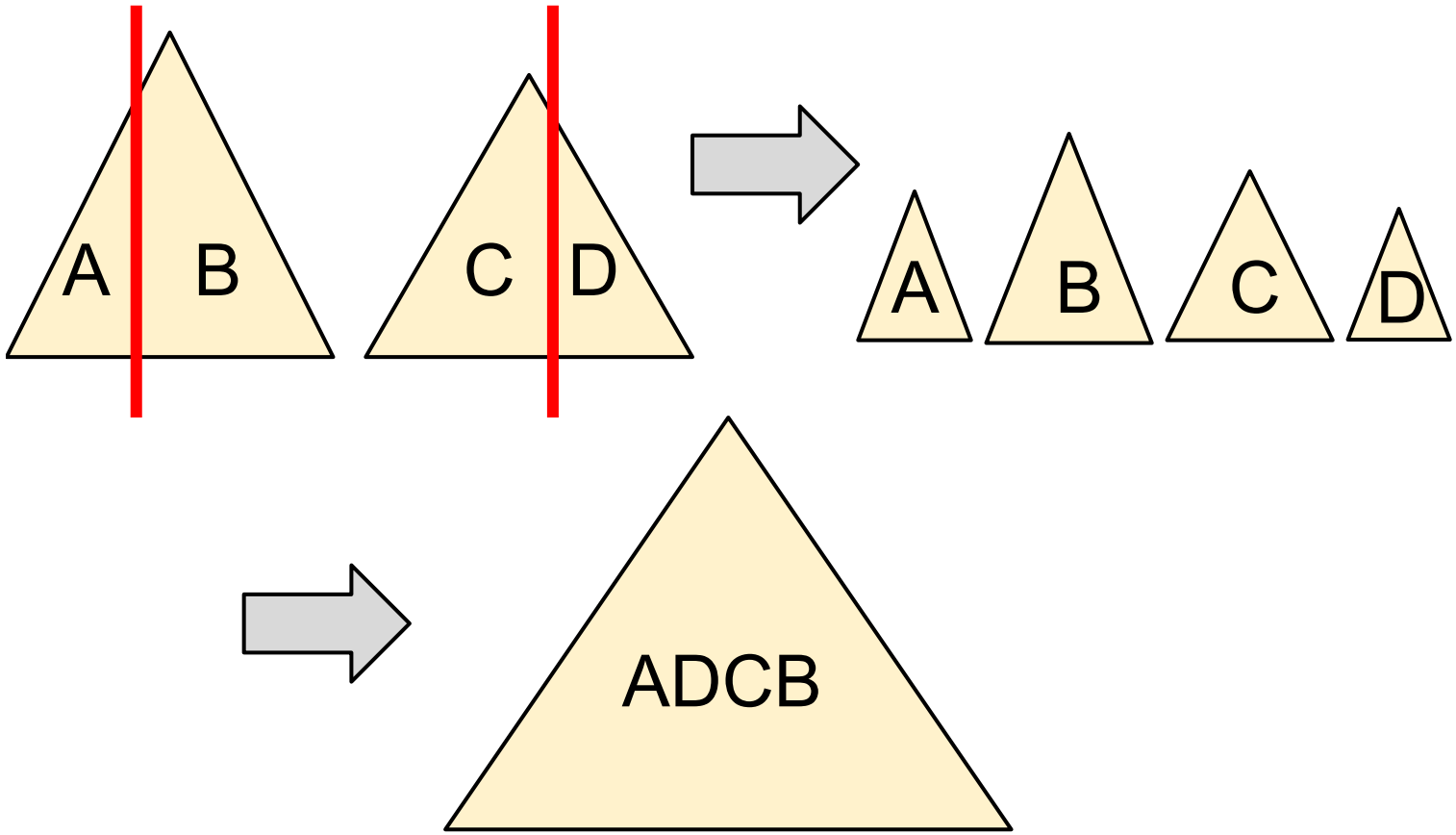
Графы:

- Реалистичные (граф дорог, веб-графы)
- Случайные (с разными плотностями ребер)

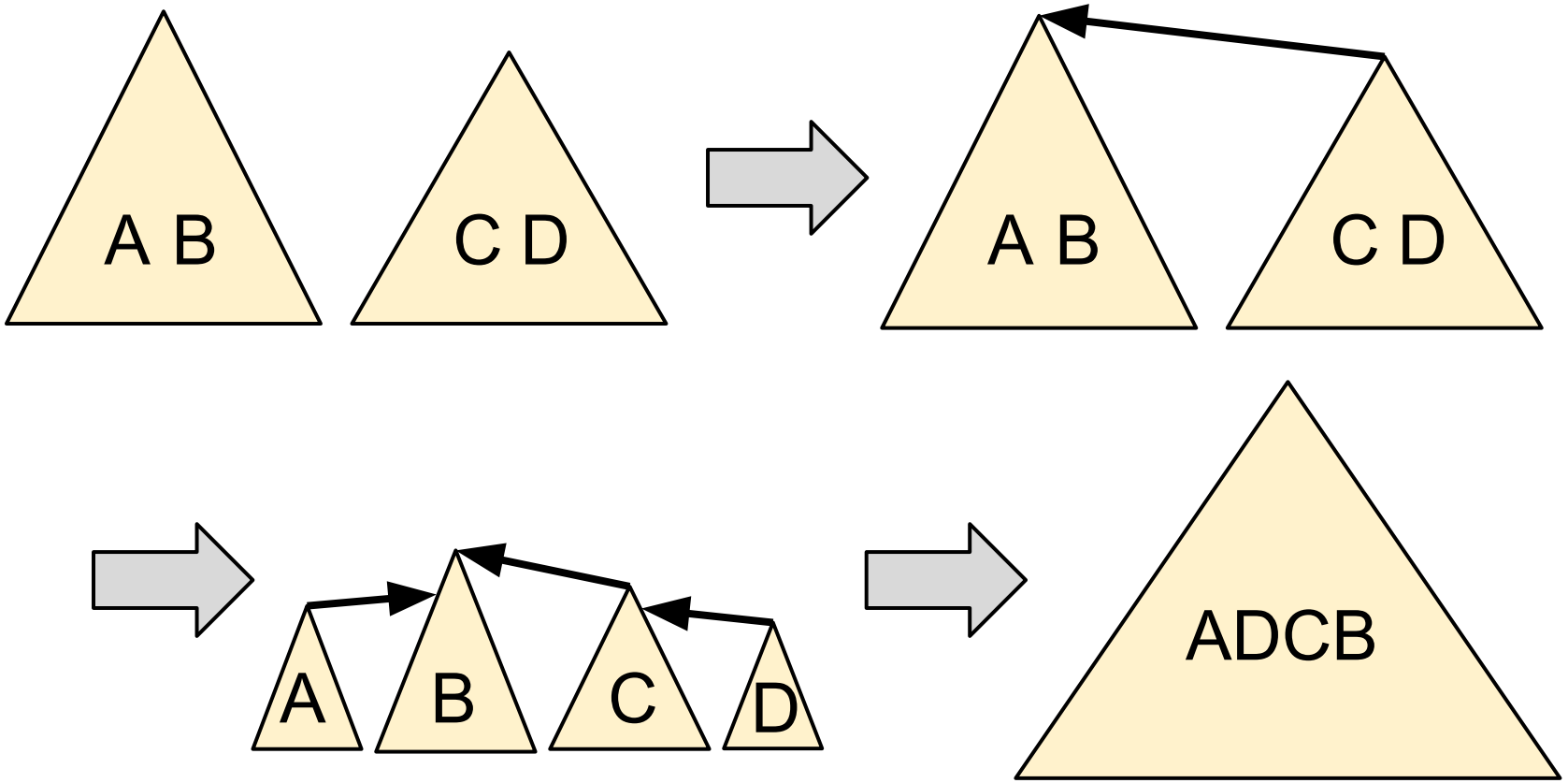
Тонкая блокировка: проблема

Операции с деревьями не атомарные.

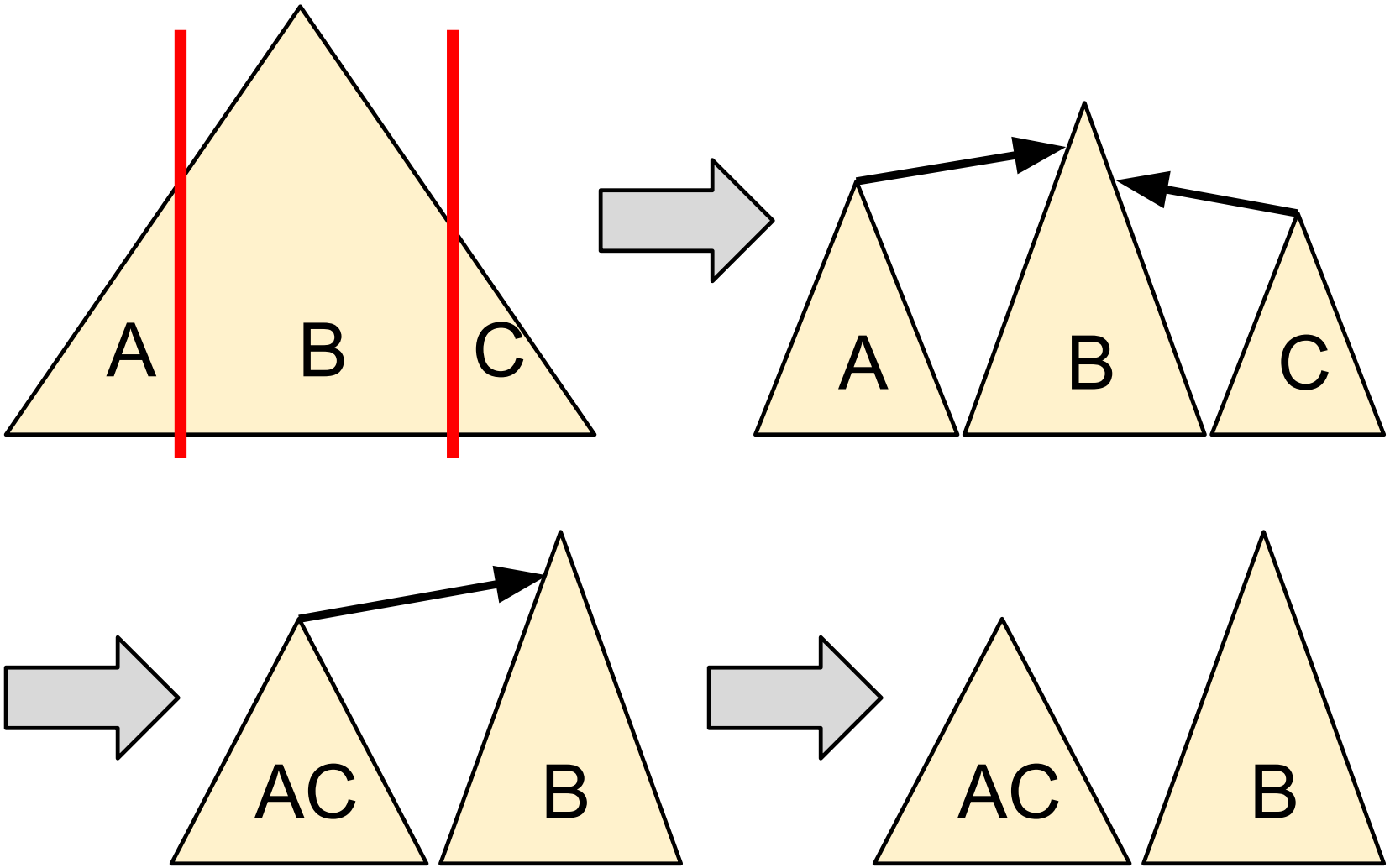
Объединение деревьев:



Линеаризуемое объединение деревьев



Линеаризуемое разделение деревьев



Software

- Java or Kotlin
- OpenJDK 11.0.4
- Ubuntu
- JMH (Java Microbenchmark Harness) library to avoid JIT-related problems and reproducibility
- LinCheck for linearizability checking