Lock-free Contention Adapting Search Trees

Kjell Winblad, Konstantinos Sagonas and Bengt Jonsson

Department of Information Technology

Uppsala University, Sweden

UPPSALA
UNIVERSITET

Introduction

Our contribution

- A new lock-free data structure
  - The lock-free contention adapting search tree (LFCA tree)

UPPSALA
UNIVERSITET

Introduction

Our contribution

- A new lock-free data structure
  - The lock-free contention adapting search tree (LFCA tree)
- Ordered sets and key-value stores

UPPSALA
UNIVERSITET

Introduction

Our contribution

- A new lock-free data structure
  - The lock-free contention adapting search tree (LFCA tree)
- Ordered sets and key-value stores
- Insert and Remove (lock-free and linearizable)

UPPSALA
UNIVERSITET

Introduction

Our contribution

- A new lock-free data structure
  - The lock-free contention adapting search tree (LFCA tree)
- Ordered sets and key-value stores
- Insert and Remove (lock-free and linearizable)
- Lookup (wait-free and linearizable)

UPPSALA
UNIVERSITET

Introduction

Our contribution

- A new lock-free data structure
  - The lock-free contention adapting search tree (LFCA tree)
- Ordered sets and key-value stores
- Insert and Remove (lock-free and linearizable)
- Lookup (wait-free and linearizable)
- Range query (lock-free and linearizable)
  - atomic snapshot of all items inside a range

UPPSALA UNIVERSITET

Introduction

Our contribution

- A new lock-free data structure
  - The lock-free contention adapting search tree (LFCA tree)
- Ordered sets and key-value stores
- Insert and Remove (lock-free and linearizable)
- Lookup (wait-free and linearizable)
- Range query (lock-free and linearizable)
  - atomic snapshot of all items inside a range
- **Adapts its synchronization granularity to the workload**
  - Attempts to get the best of:
    - Coarse-grained synchronization
    - Fine-grained synchronization

UPPSALA
UNIVERSITET

Introduction

Concurrent ordered sets with support for range queries

- Important for:
  - Big scale databases and data processing platforms
    - Fast updates to store incoming data
    - Concurrent range queries for analytics

UPPSALA
UNIVERSITET

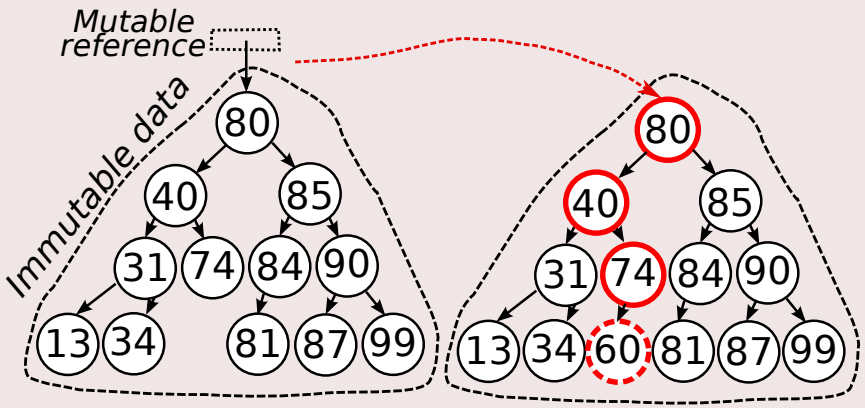
Introduction

Concurrent ordered sets with support for range queries

- Important for:
  - Big scale databases and data processing platforms
    - Fast updates to store incoming data
    - Concurrent range queries for analytics
- Challenging
  - Large range queries + updates = **many collisions**

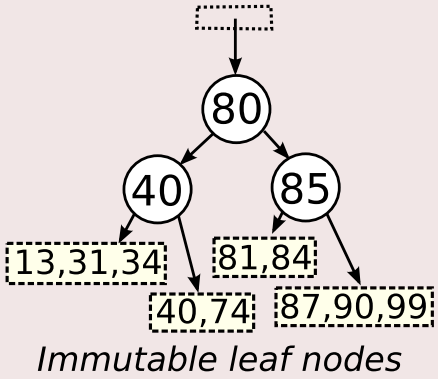
Introduction

Coarse-grained – Mutable reference to immutable data



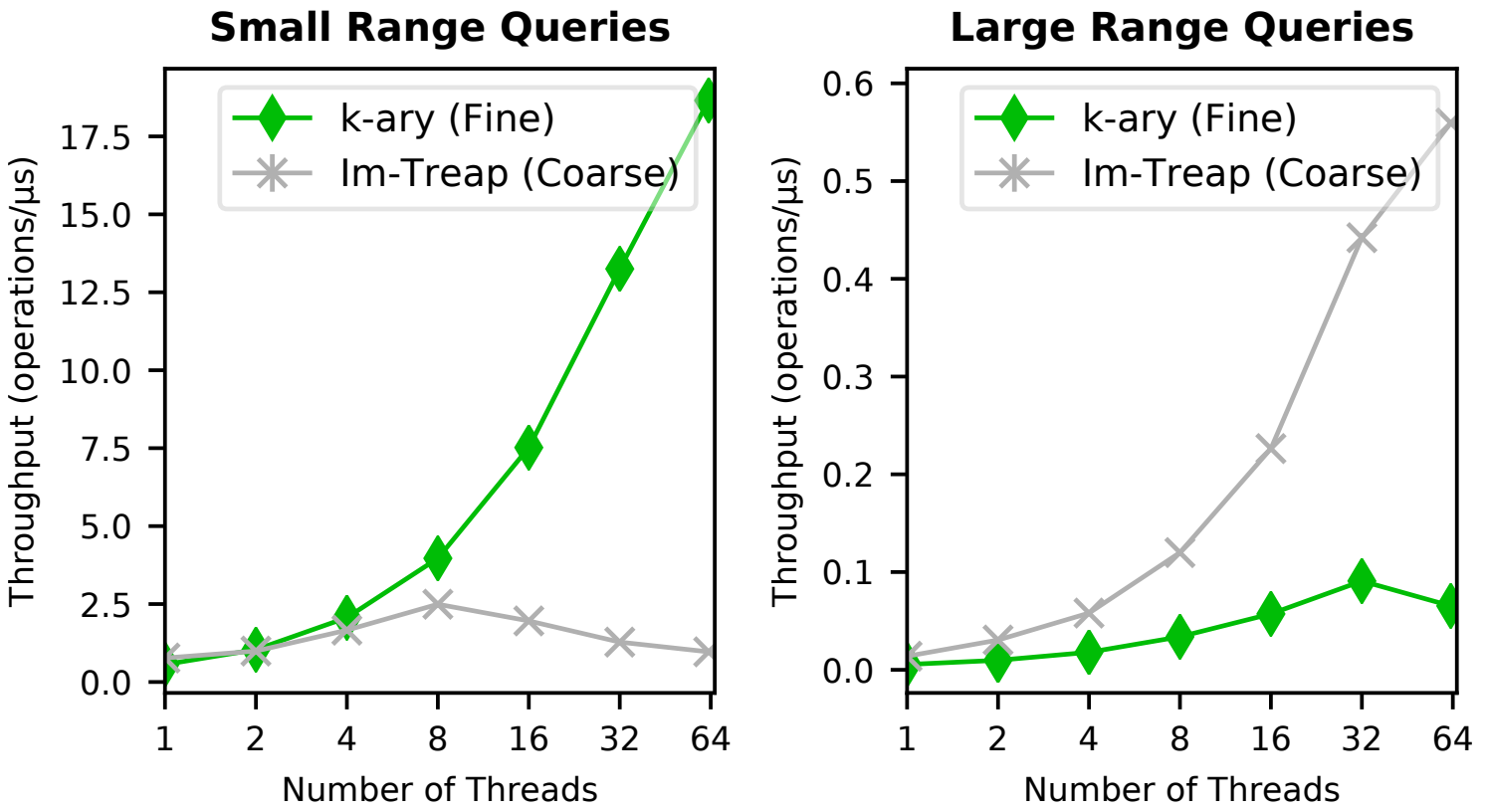
- Im-Treap
  Herlihy [PPoPP'90]
  (general method)

Fine-grained – Immutable leaf nodes



- k-ary [OPODIS'12]

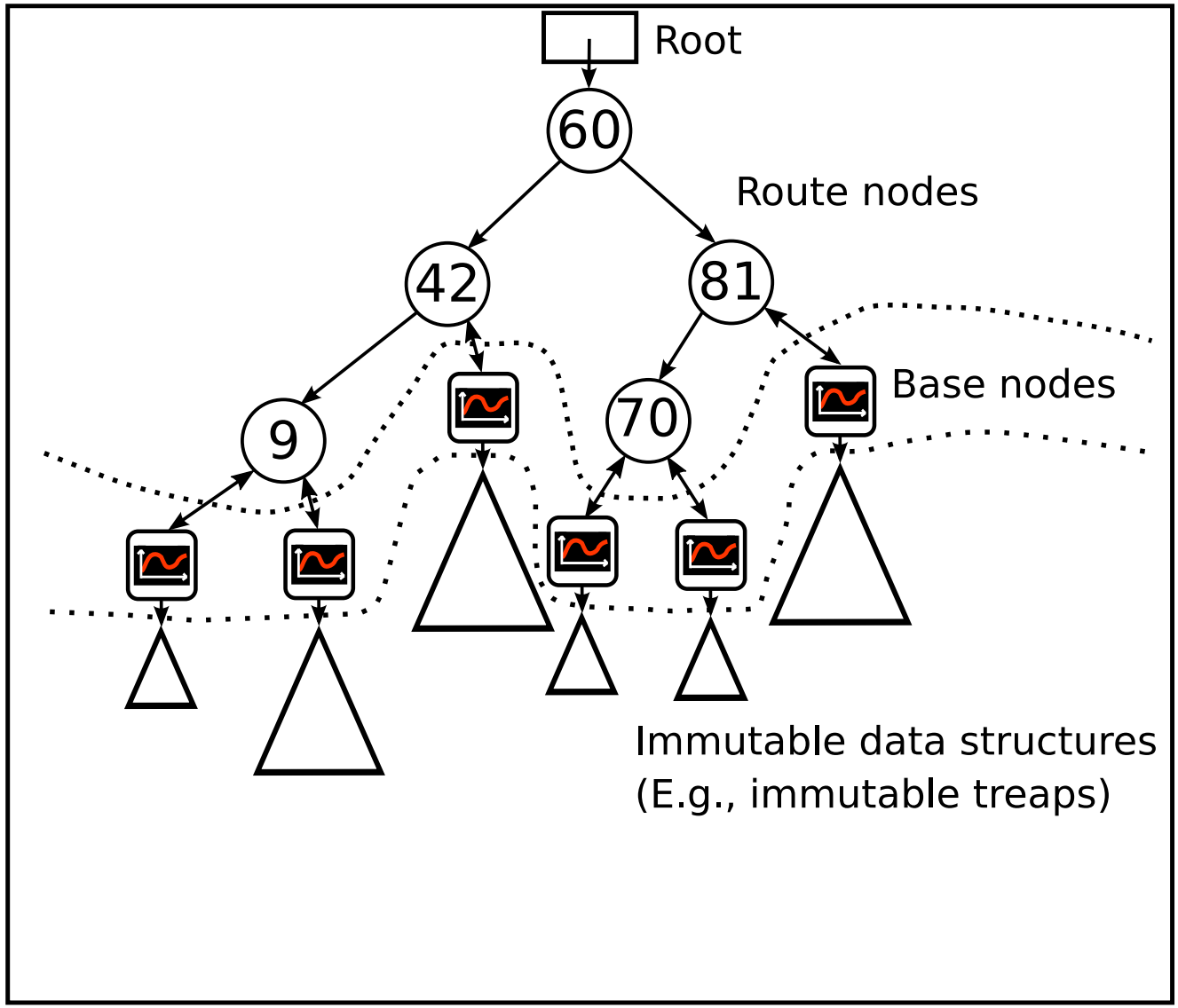
Coarse-grained vs Fine-grained – Trade-offs



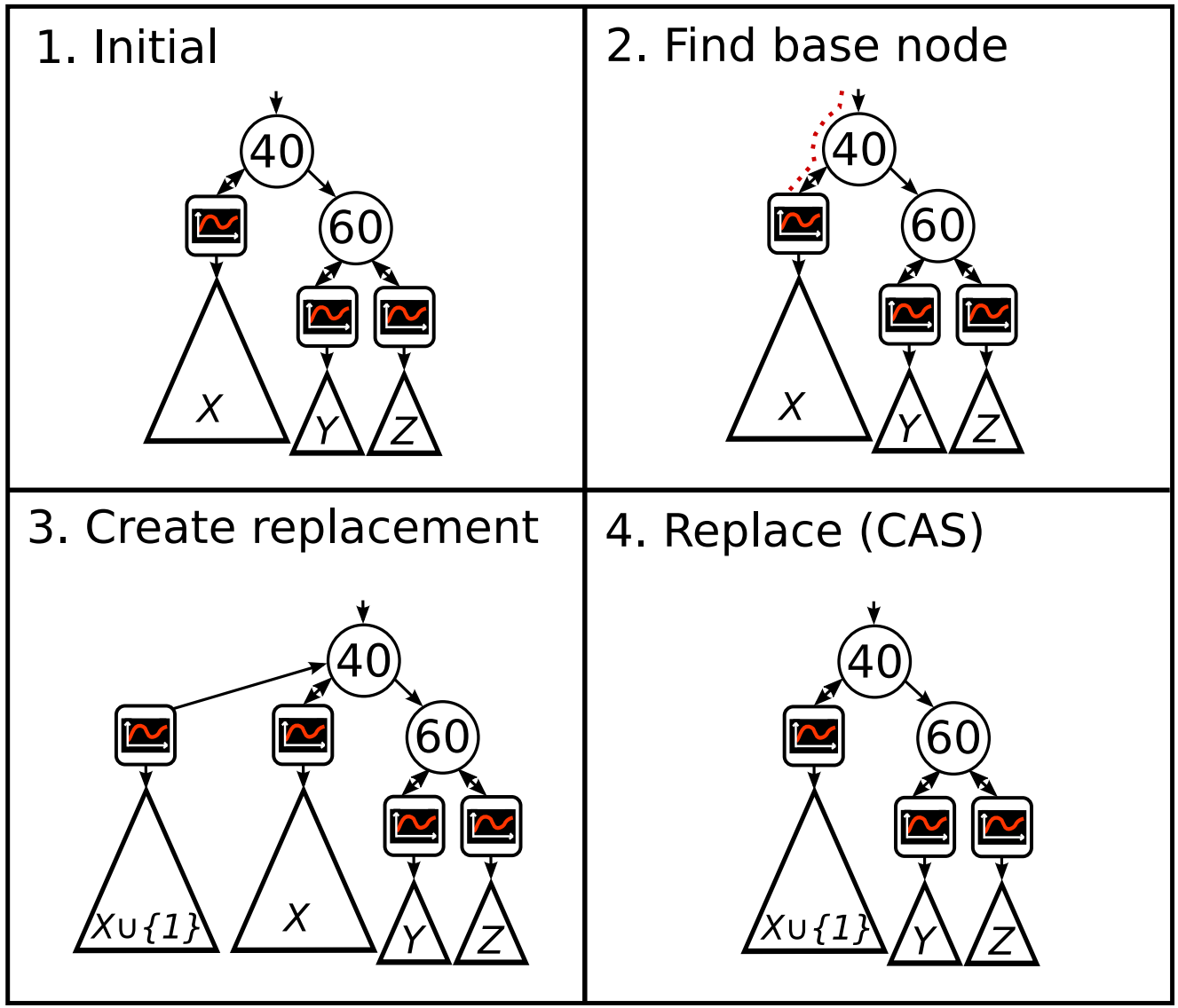
(a) Average #items/query ≈ 5 (b) Average #items/query ≈ 50K

Figure: 10% Inserts, 10% Removes, 55% Lookups, 25% Range queries, ≈ 500K items

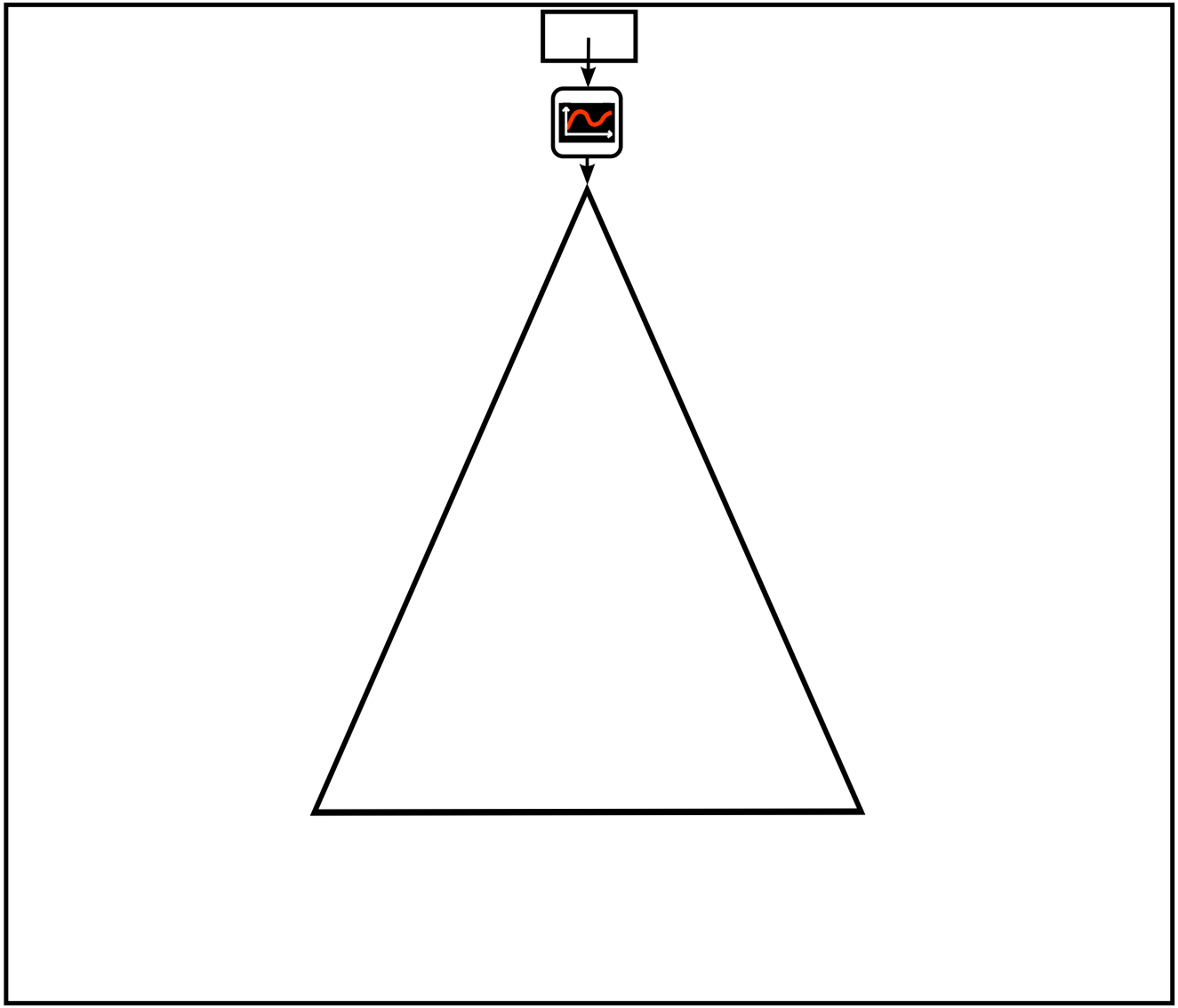
LFCA tree


Root
60
Route nodes
42
81
9
70
Base nodes
Immutable data structures
(E.g., immutable treaps)

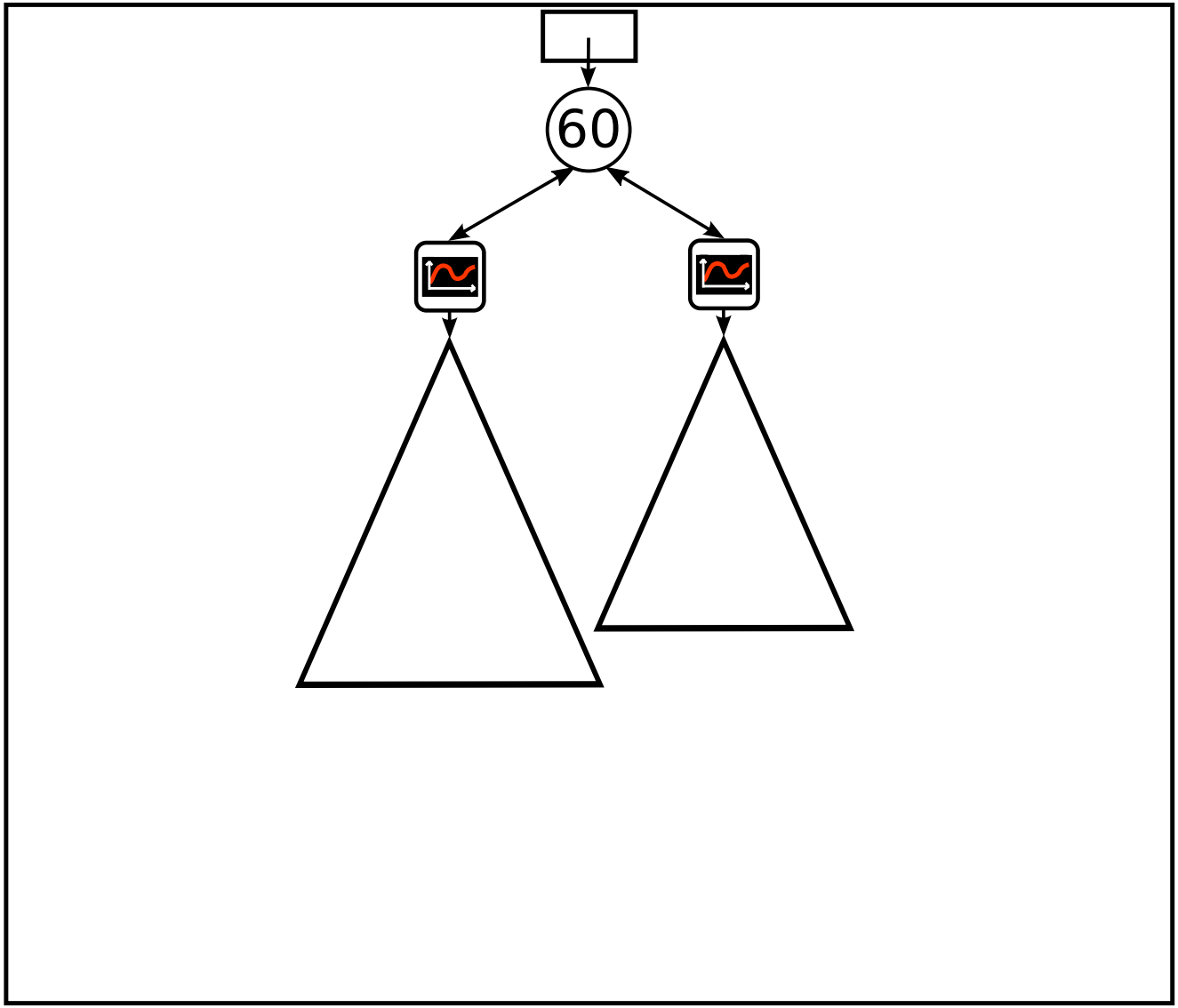
LFCA tree (Insert 1)



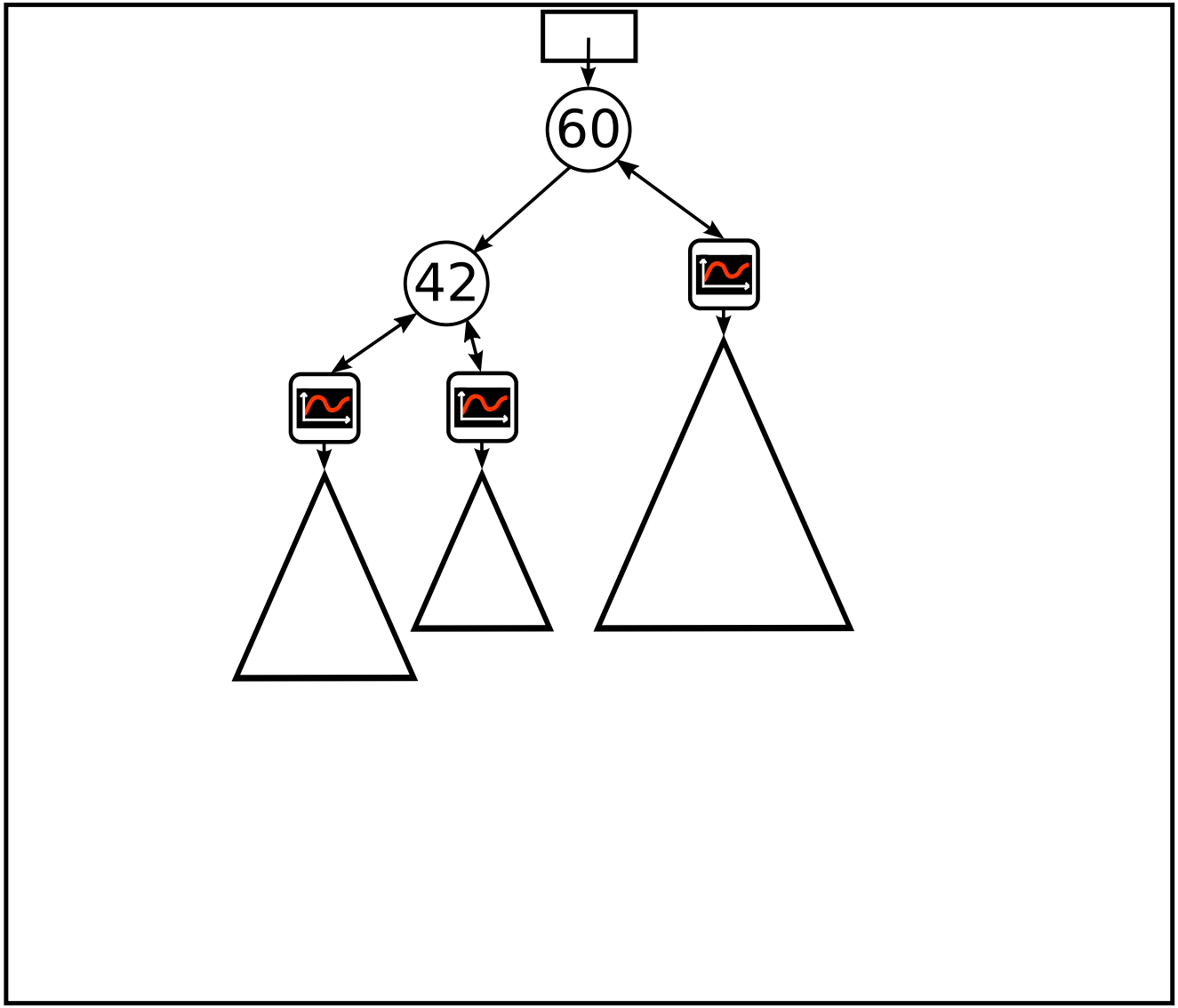
LFCA tree (Adaptations)



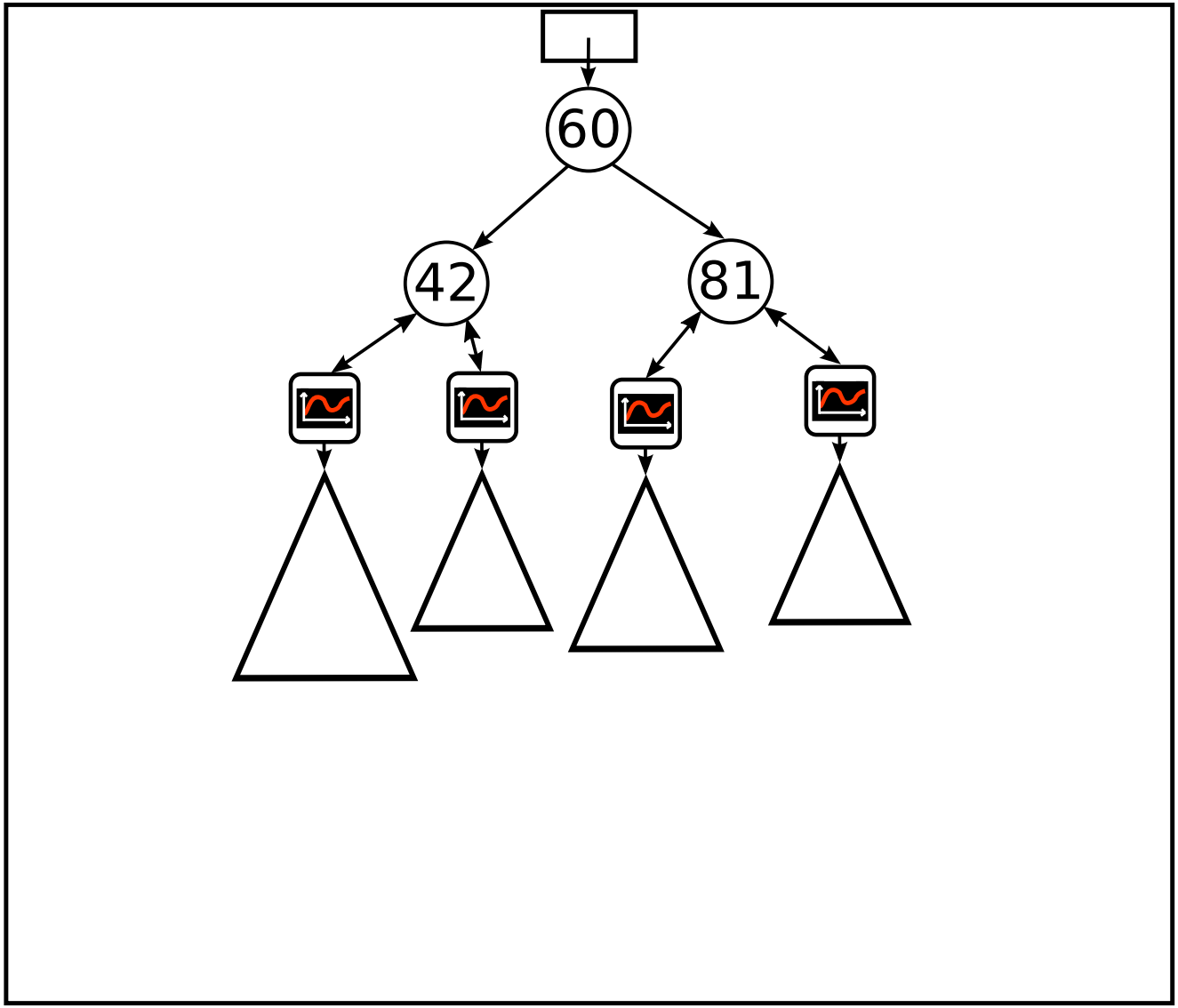
LFCA tree (Adaptations)



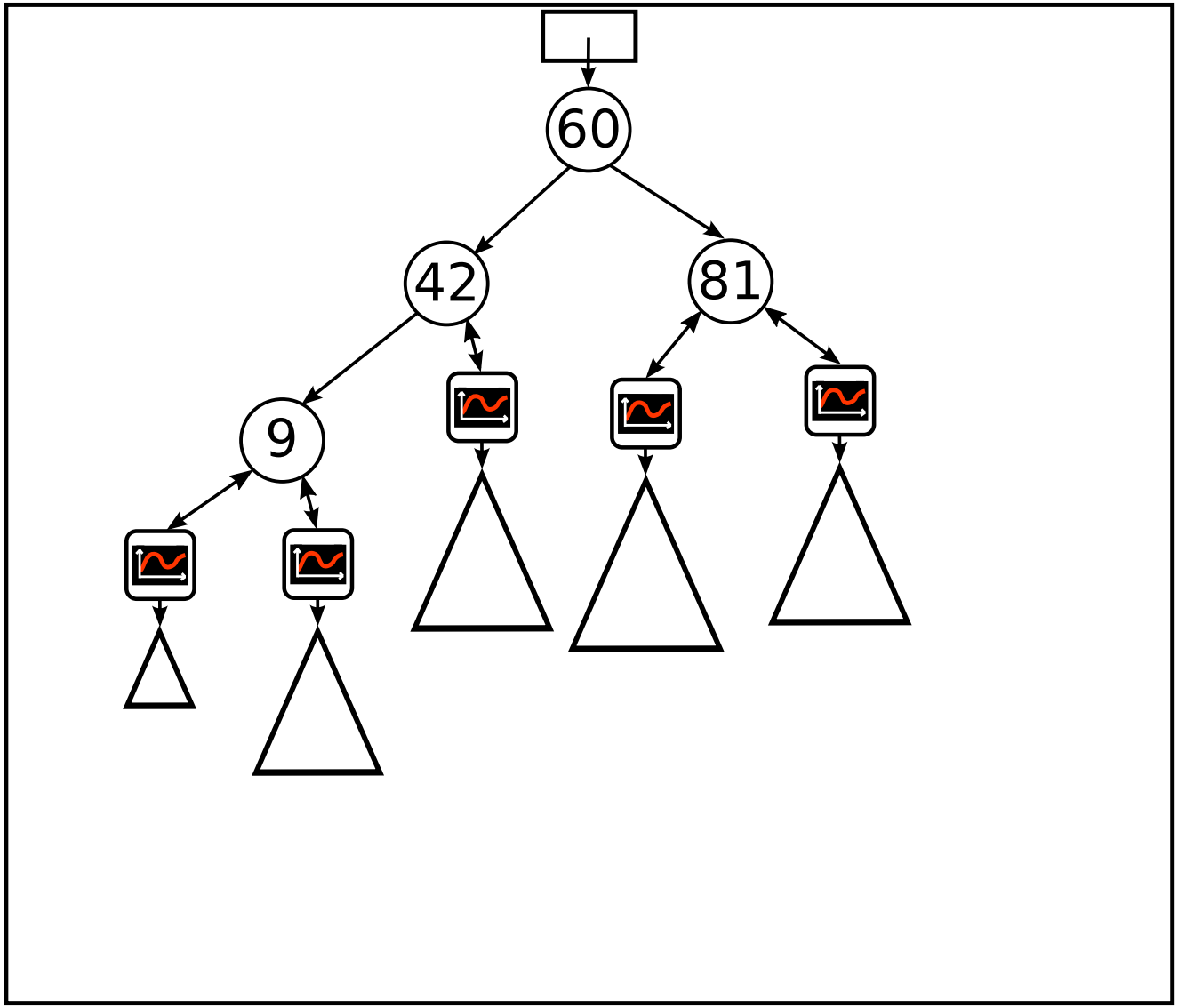
LFCA tree (Adaptations)



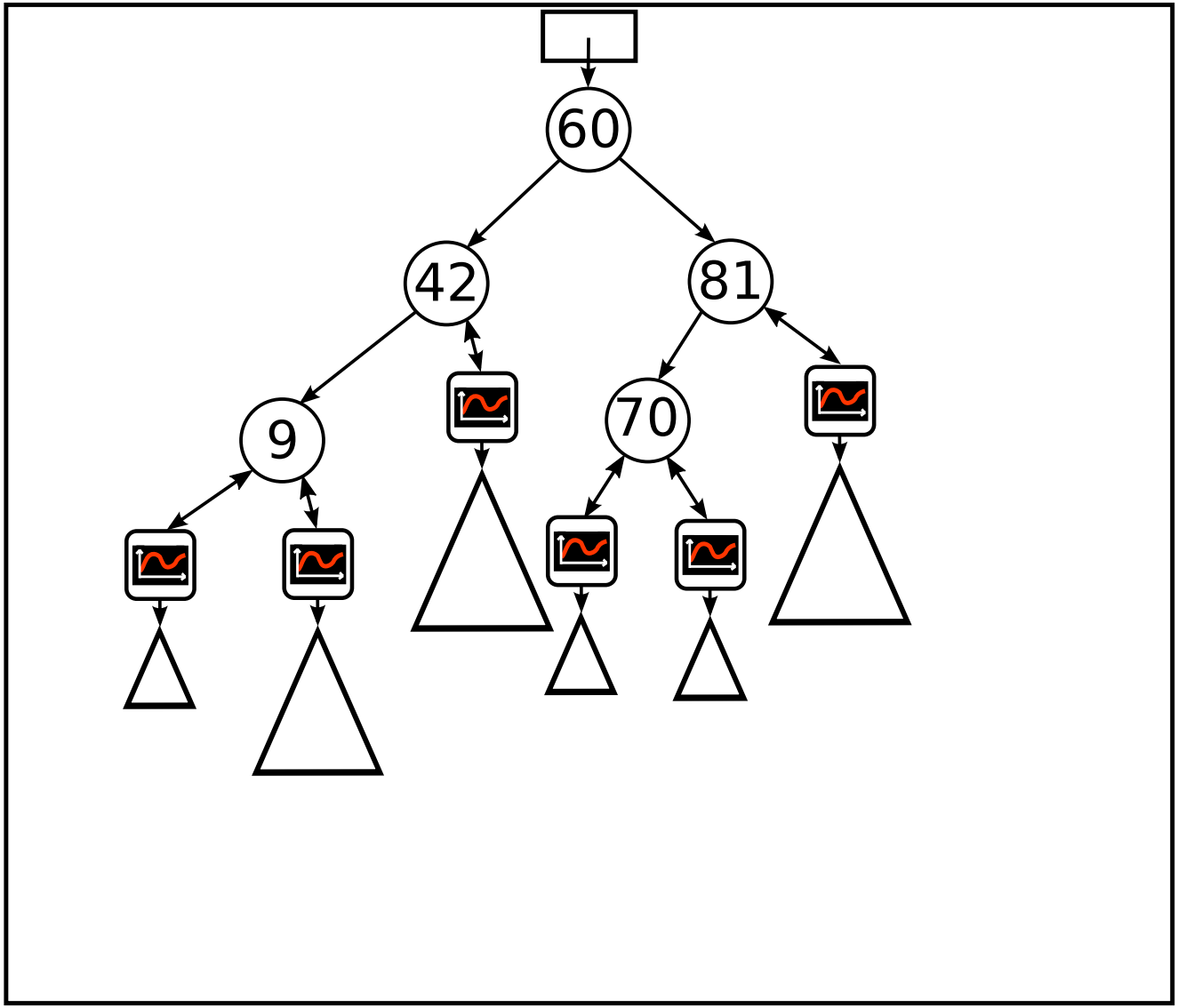
LFCA tree (Adaptations)



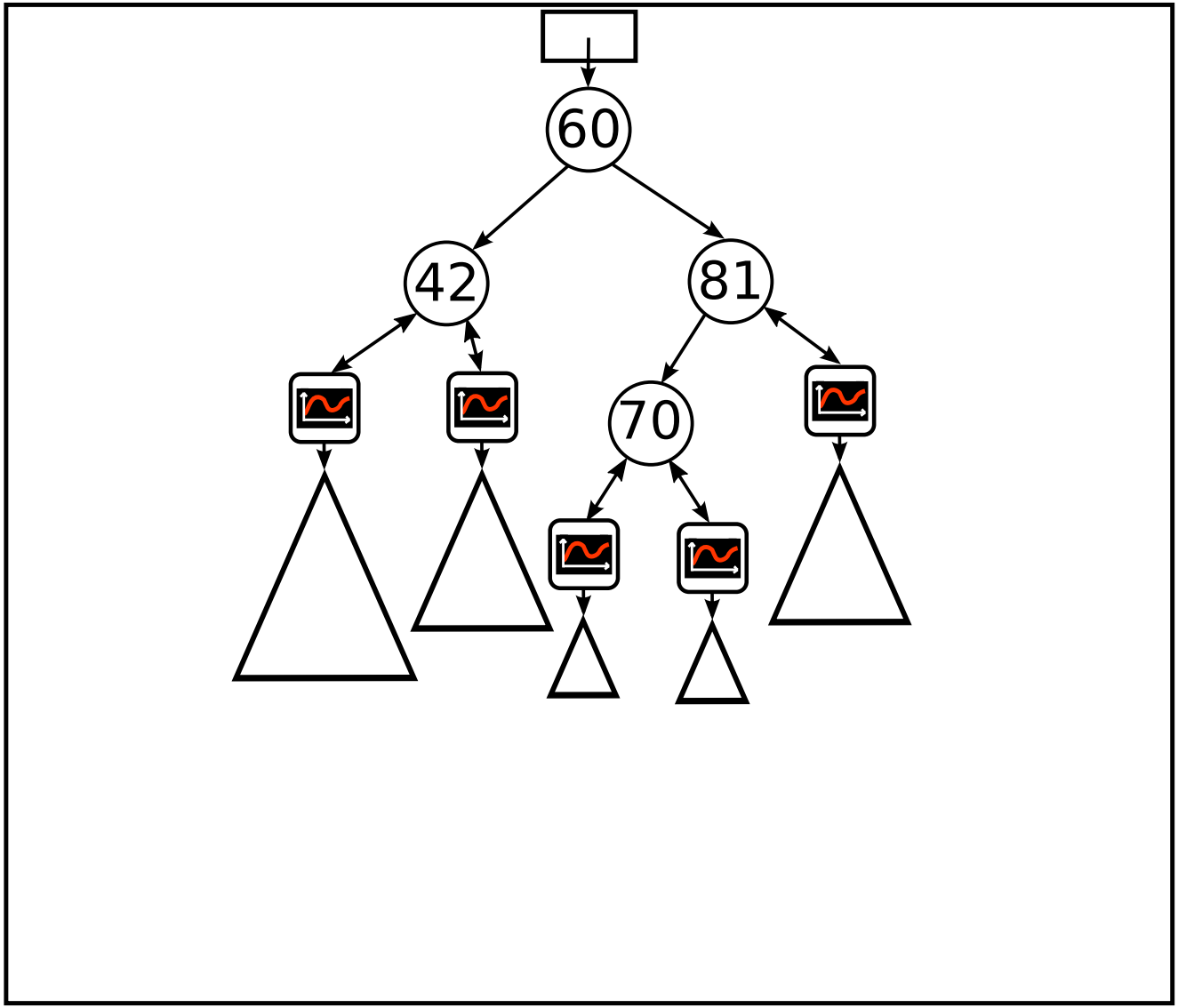
LFCA tree (Adaptations)



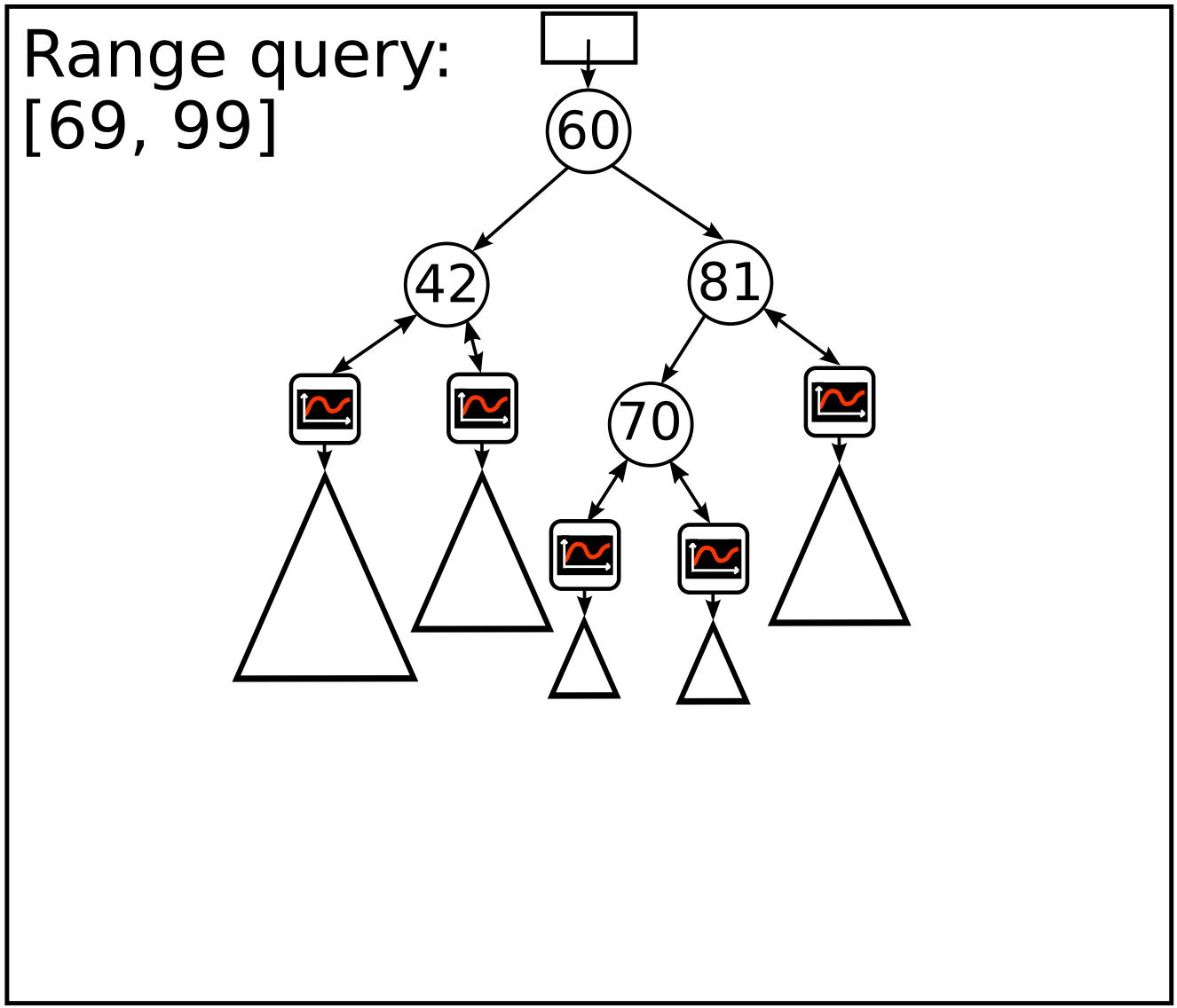
LFCA tree (Adaptations)



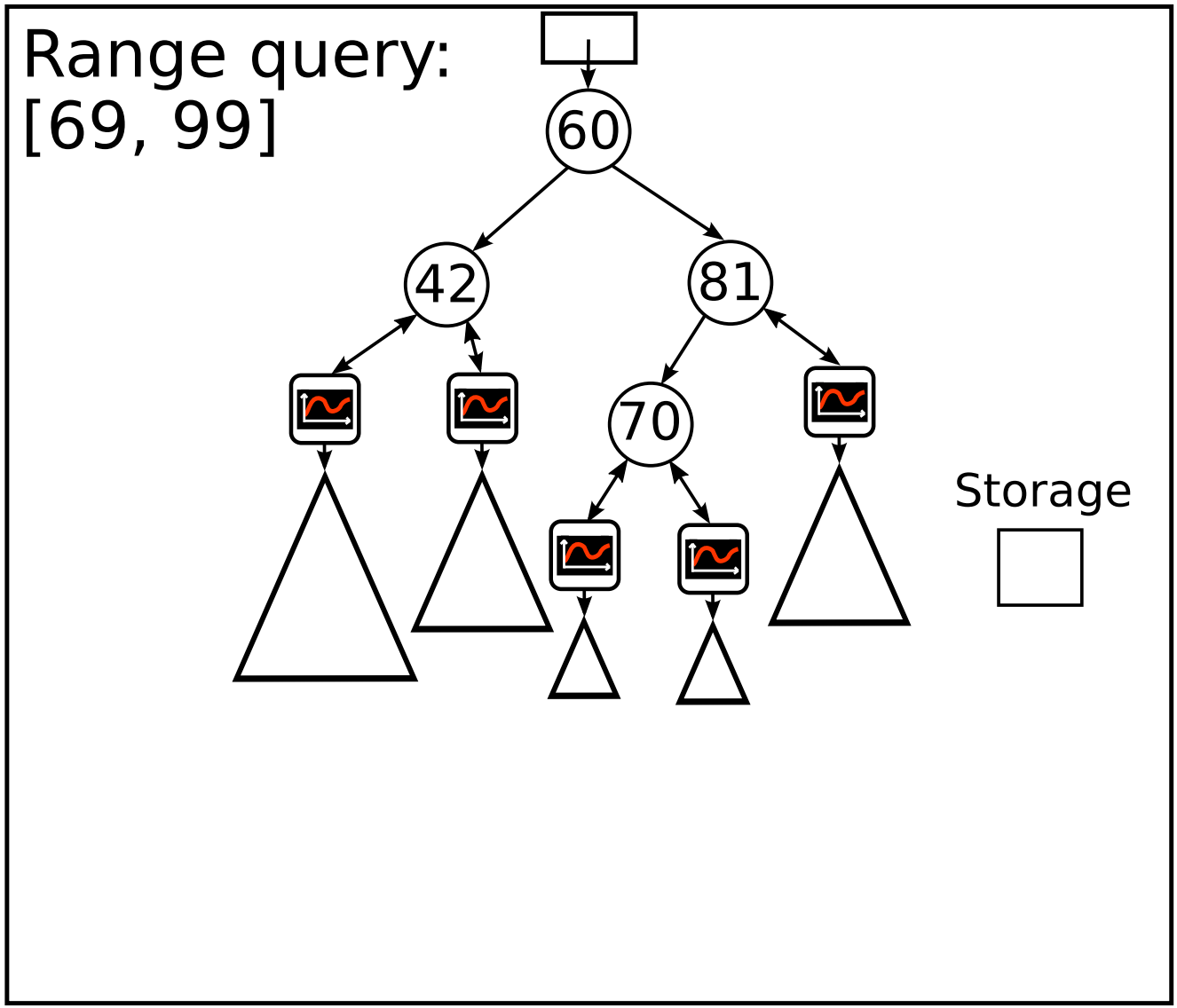
LFCA tree (Adaptations)



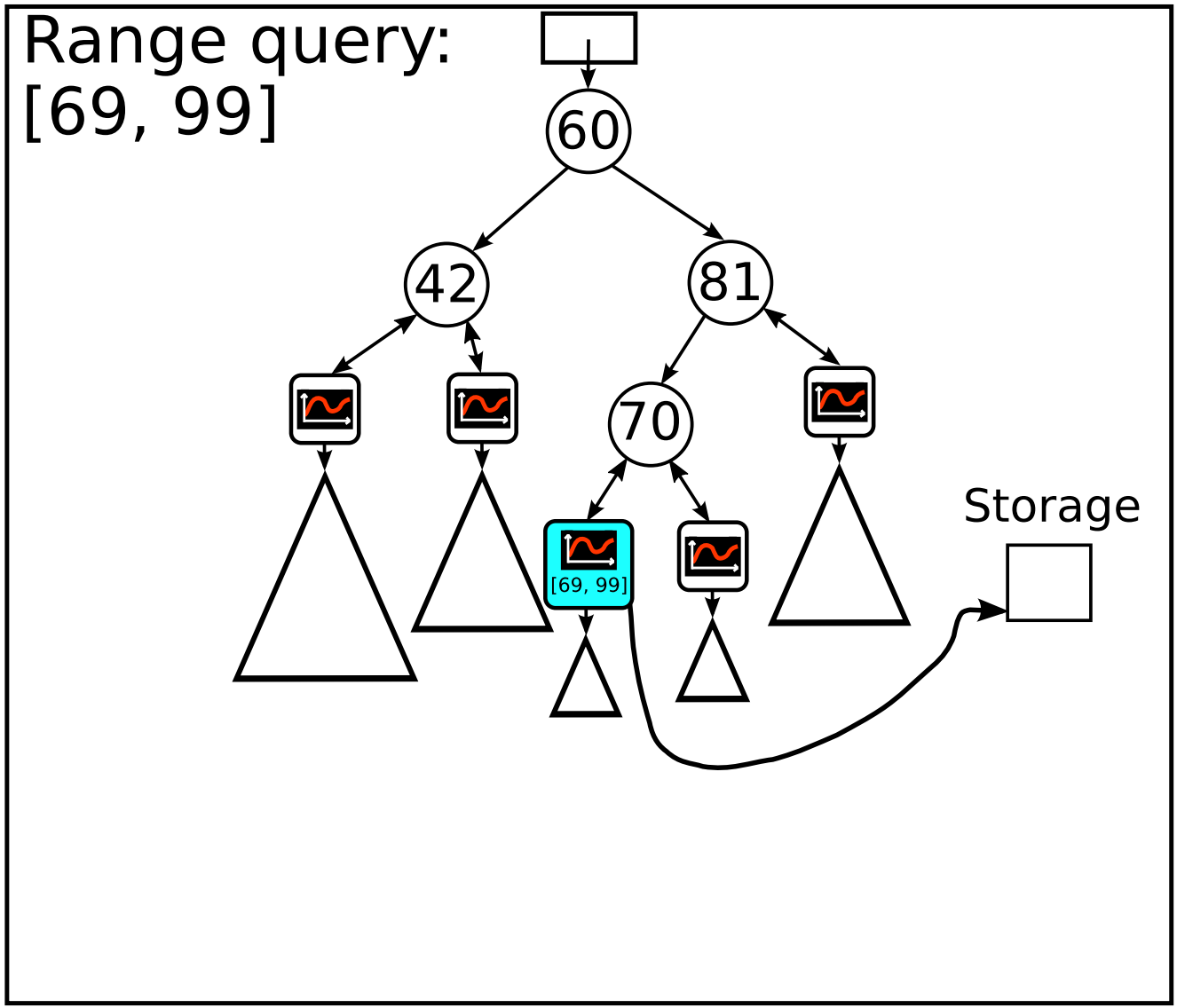
LFCA tree



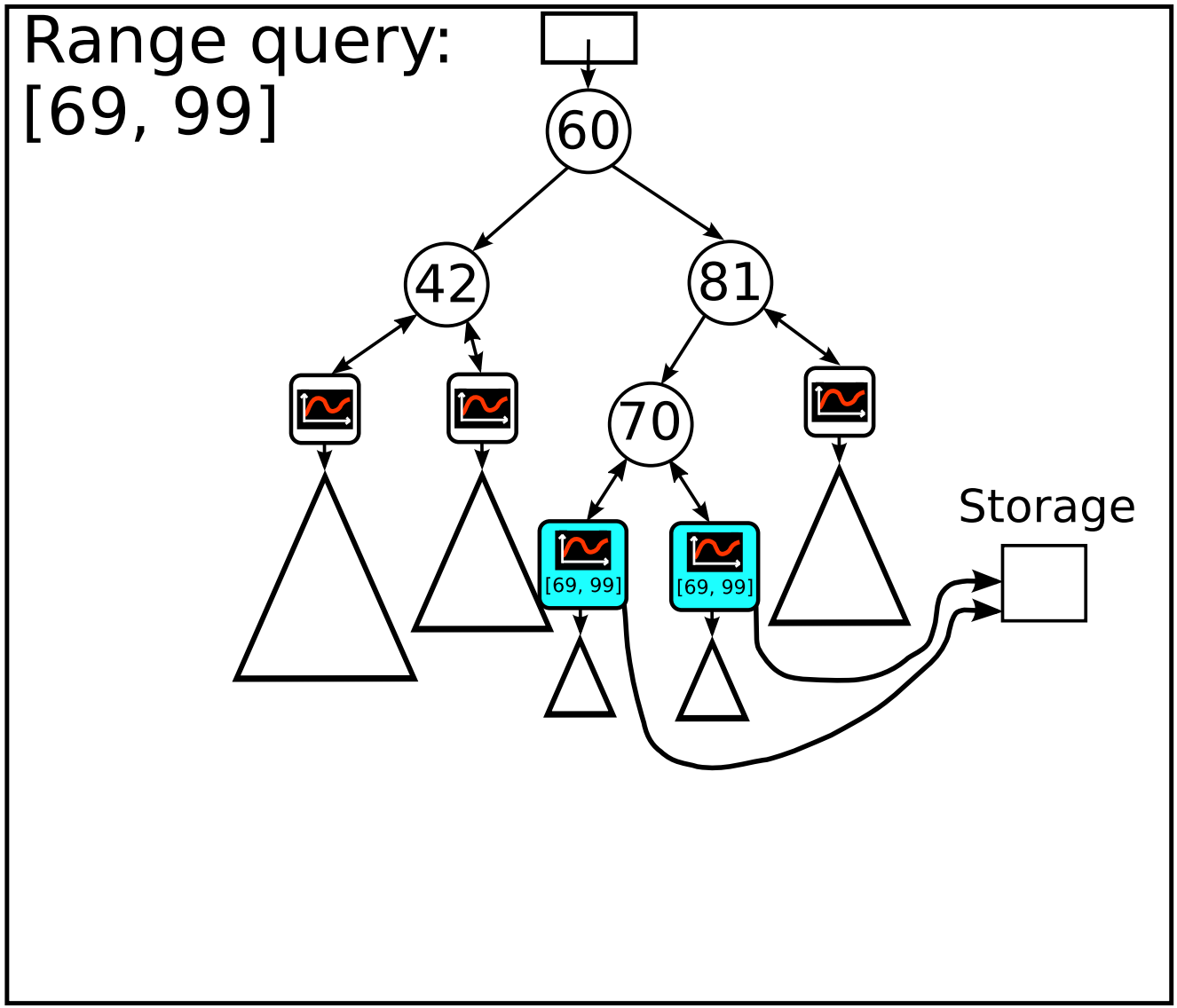
LFCA tree



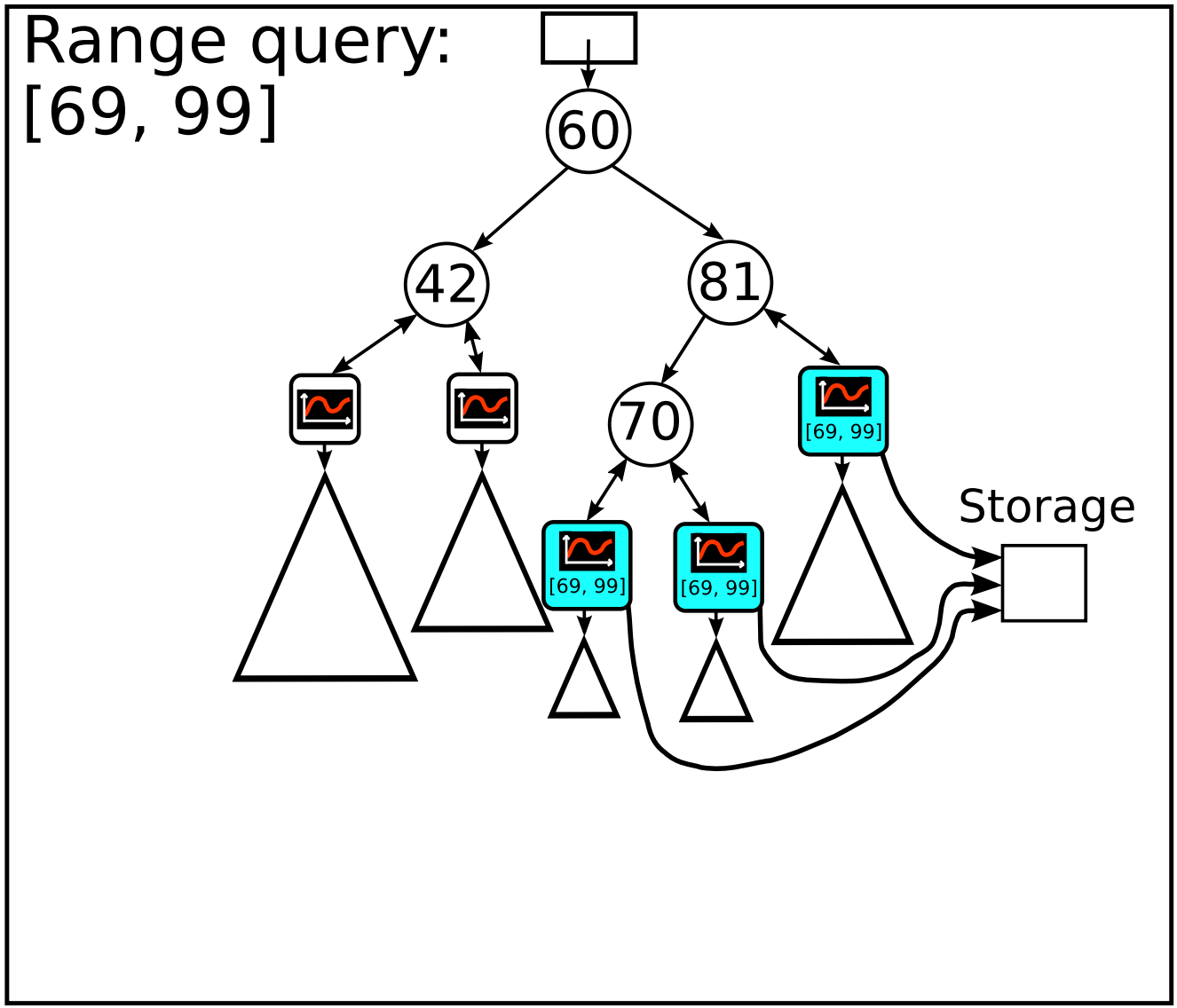
LFCA tree



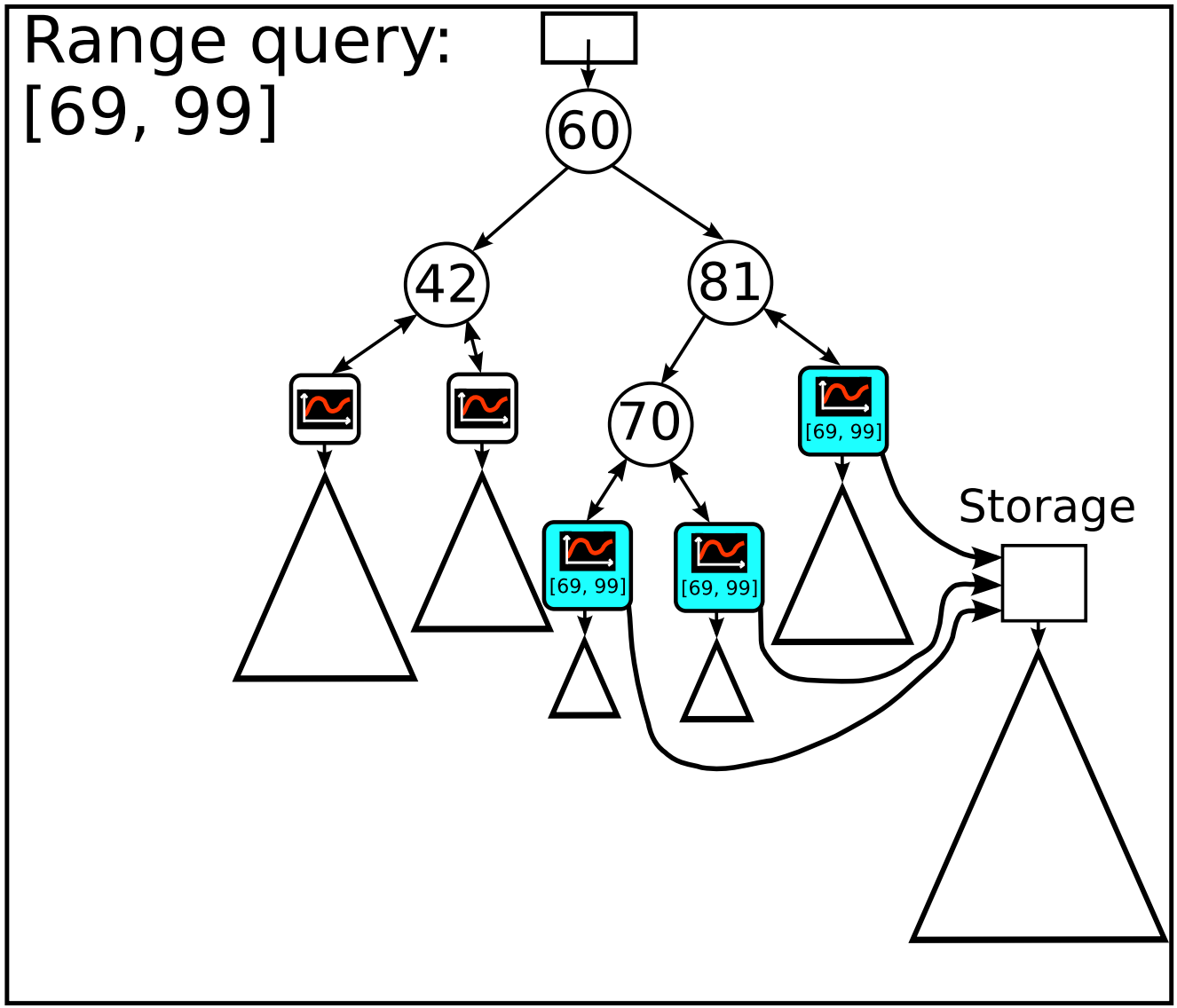
LFCA tree



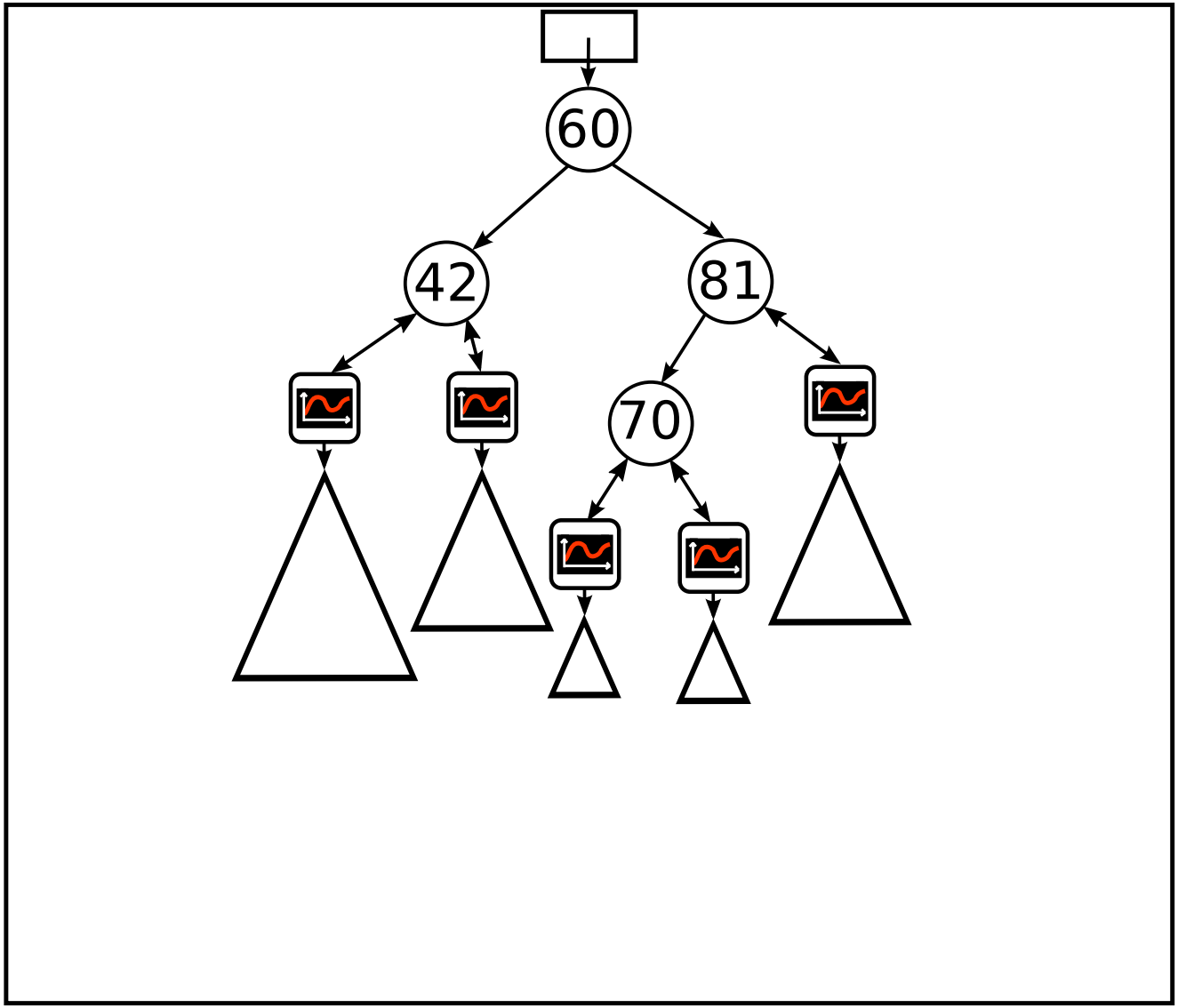
LFCA tree



LFCA tree



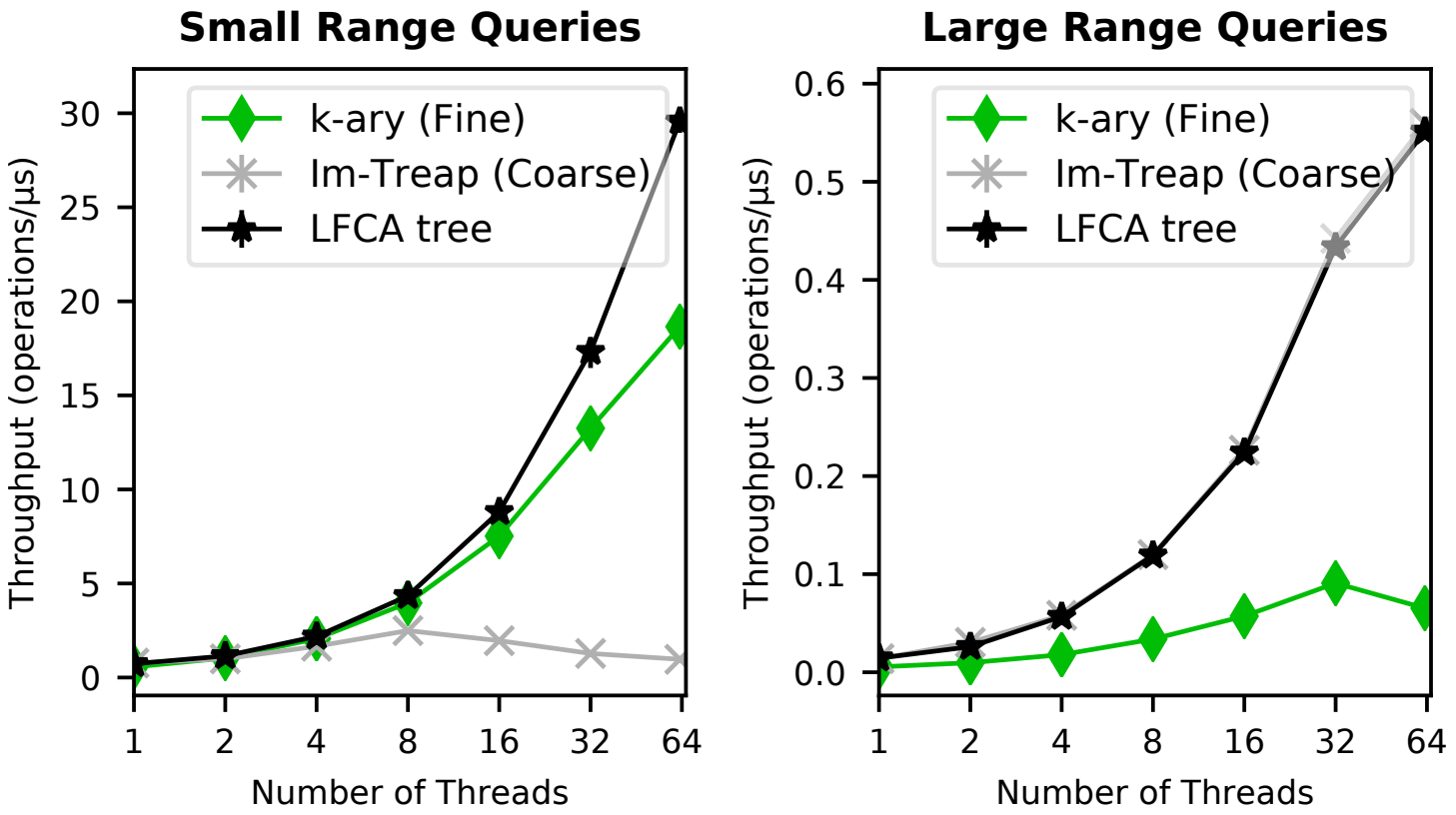
LFCA tree



Implementation Details

- Immutable treap (balanced binary search tree)
- Items stored in fat leaf nodes
  - Up to 64 items in arrays
  - Improves cache locality

How do LFCA trees perform?



(a) Average #items/query ≈ 5 (b) Average #items/query ≈ 50K

Figure: 10% Inserts, 10% Removes, 55% Lookups, 25% Range queries, ≈ 500K items

Evaluation

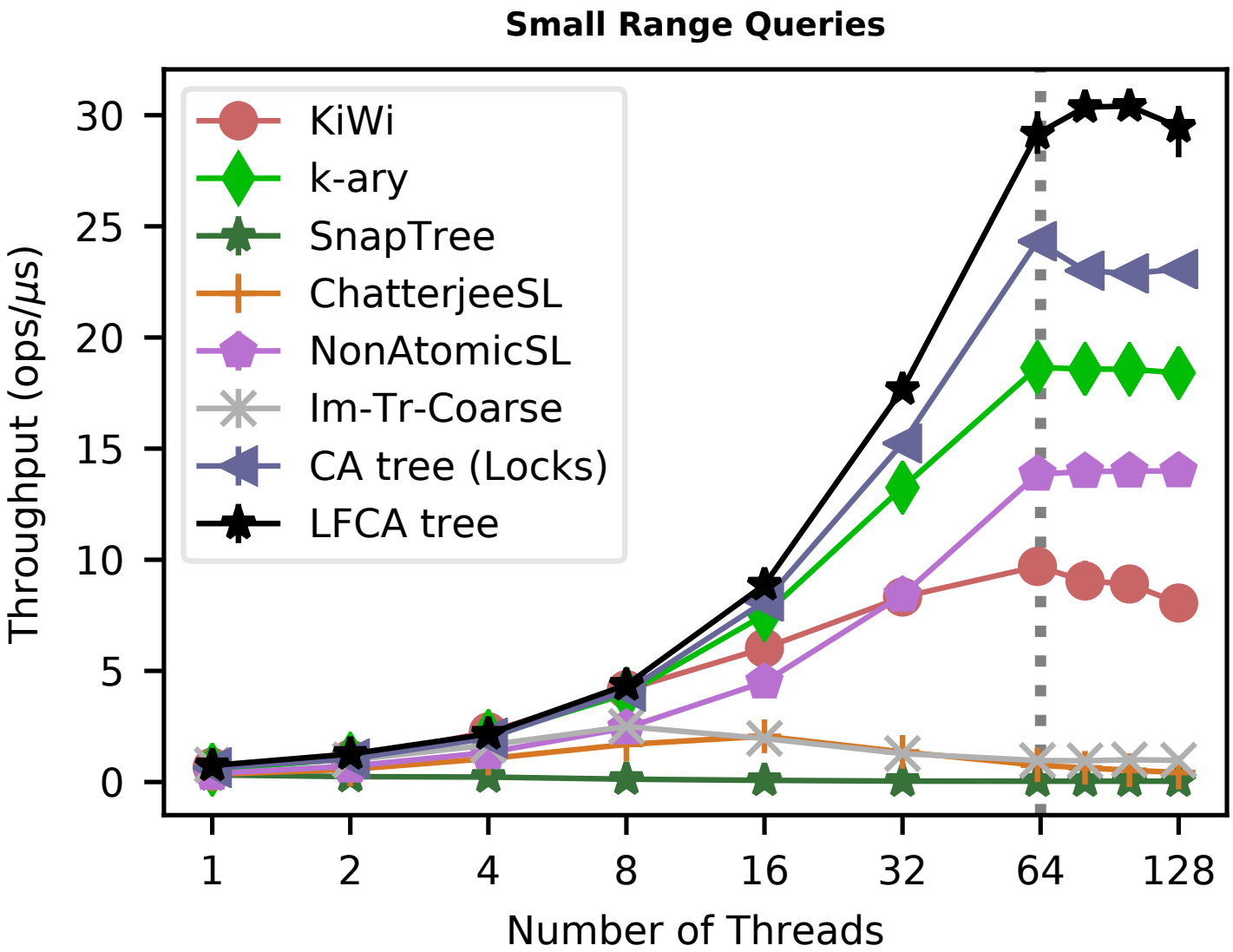
- Benchmark with a mix of
  - Inserts
  - Removes
  - Lookups
  - **Range Queries of size up to** *max*
- Platform
  - NUMA with four Intel Xeon E5-4650 CPUs (2.70GHz)
    8 cores each with hyperthreading **= 64 logical cores**
- Implementation in Java
- See paper for other benchmarks...

Data structures

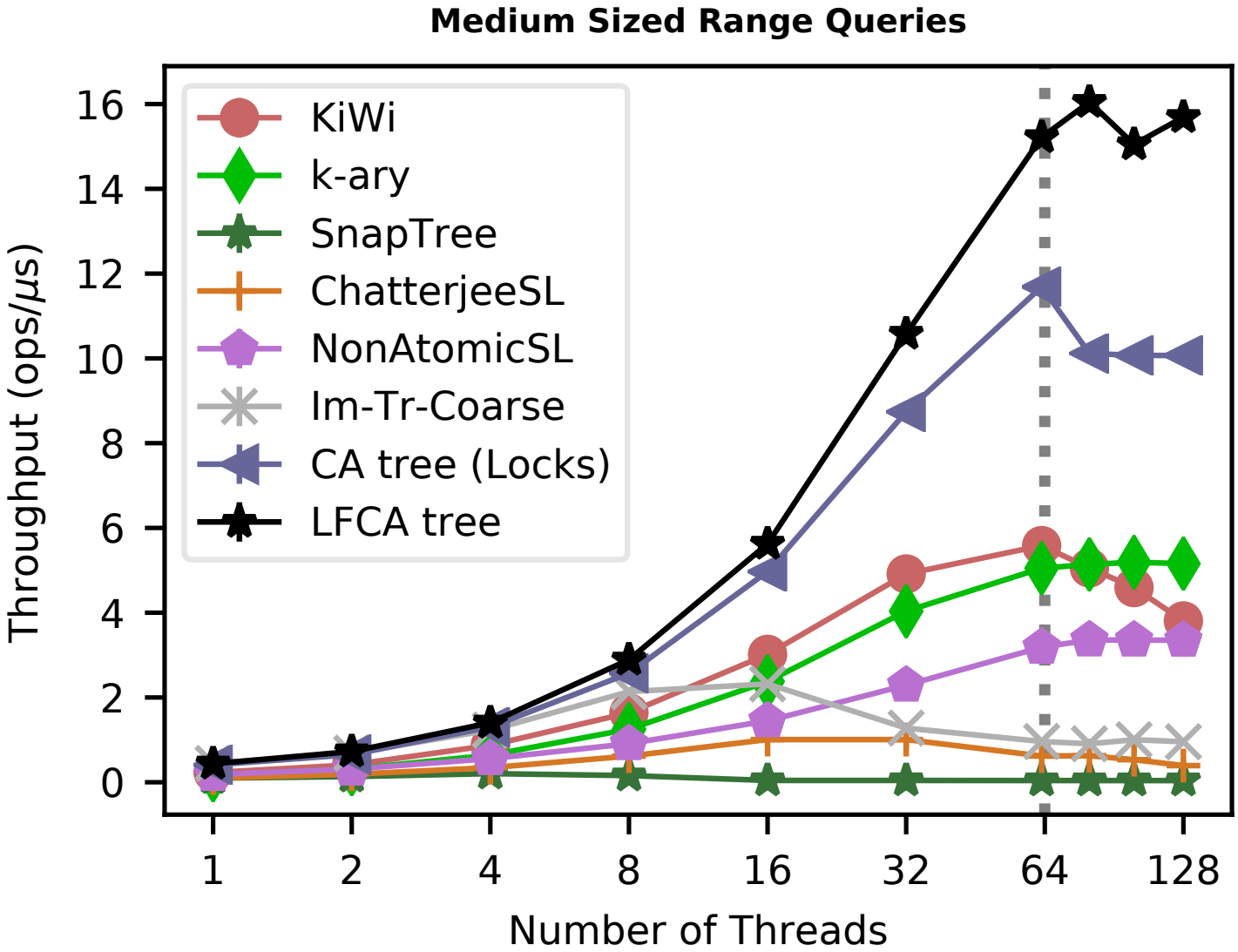
- **KiWi** – Global version number counter for range queries
  Basin et al. [PPoPP'17]
- **ChatterjeeSL** – General method for range queries
  Chatterjee [ICDCN'17]
- **k-ary** – up to k items in immutable leaf nodes
  Brown and Avni [OPODIS'12]
- **Lock-Based CA tree**
  Sagonas and Winblad [JPDC'18], Winblad [ICCSW'17]
- **SnapTree** – Fast snapshots by letting updates copy nodes
  Bronson et al. [PPoPP'10]

More...

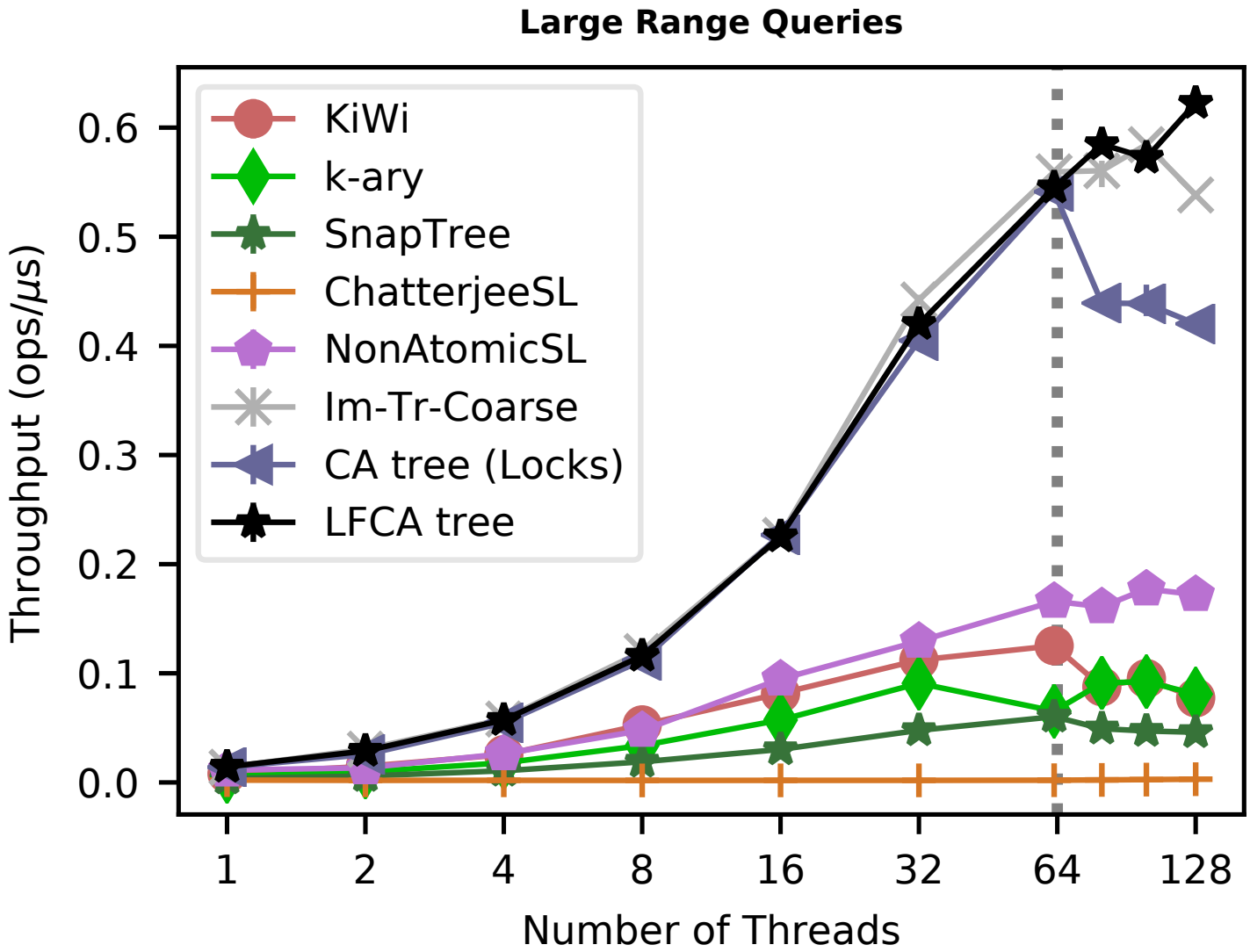
- See paper for a more detailed discussion...



≈ 500K items, Inserts:10%, Removes:10%, Lookups:55%,
Queries:25%-max:10



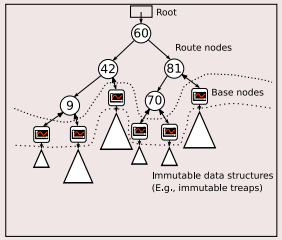
≈ 500K items, Inserts:10%, Removes:10%, Lookups:55%,
Queries:25%-max:1000



≈ 500K items, Inserts:10%, Removes:10%, Lookups:55%,
Queries:25%-max:100000

Final Remarks

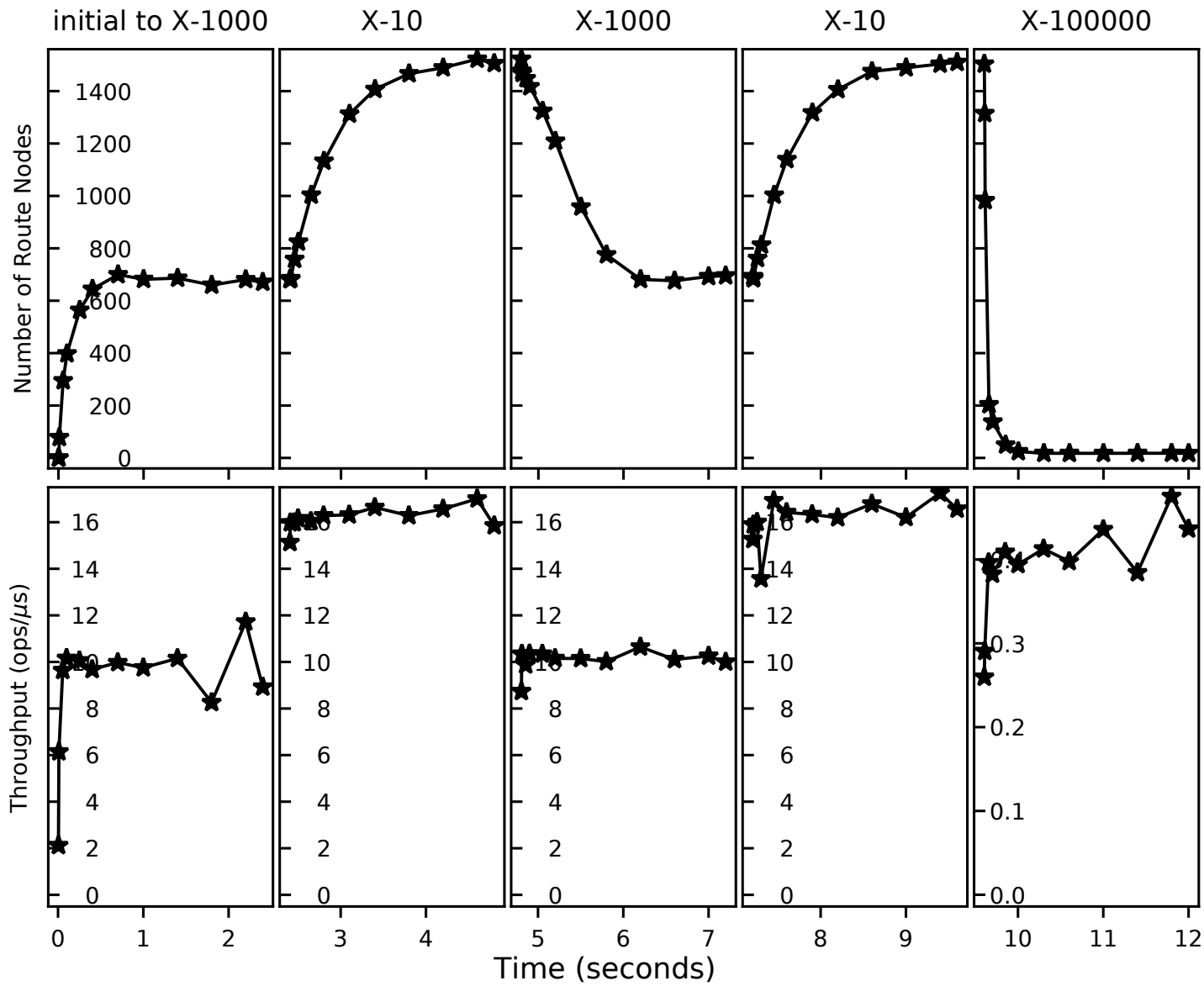
LFCA tree



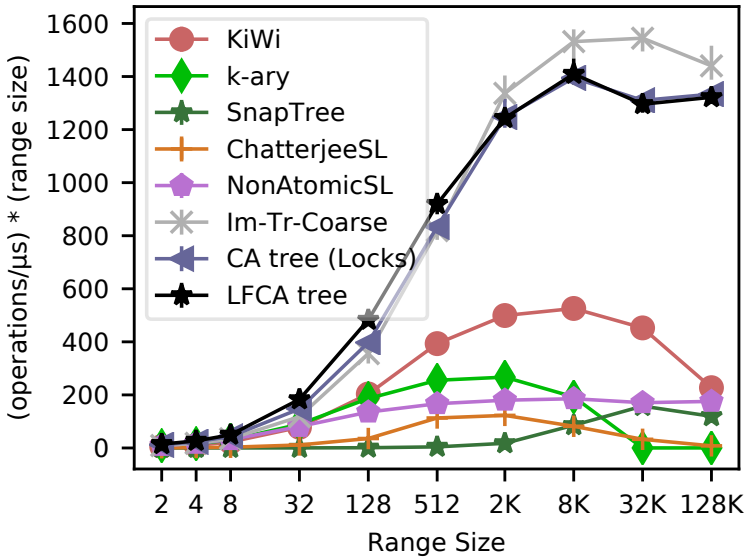
- Lock-free search tree with range query support that adapts to contention
- Automatically balances the trade-offs of coarse-grained and fine-grained synchronization

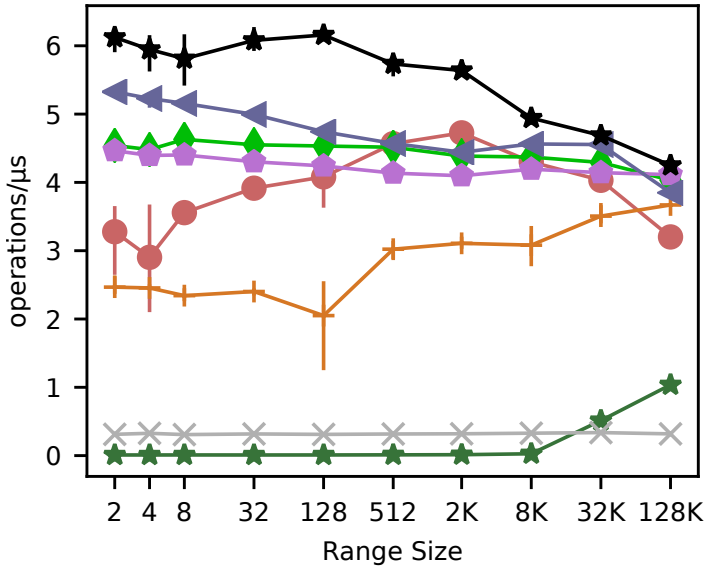


- Scales better than related concurrent data structures over a wide range of scenarios


initial to X-1000
X-10
X-1000
X-10
X-100000
Number of Route Nodes
Throughput (ops/μs)
Time (seconds)

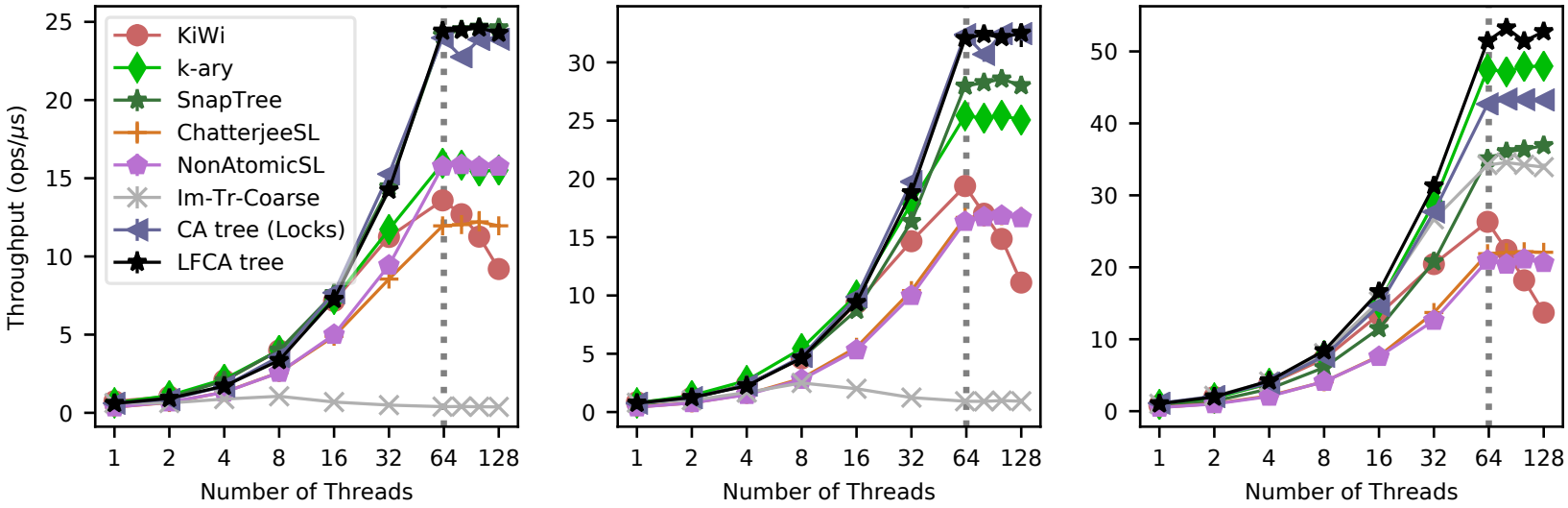
X = w:20% r:55% q:25%





(a) Range queries (parallel updates) (b) Updates (parallel range queries)

Figure: On the left, throughput for the 16 range query threads, and on the right, throughput for the threads doing only inserts and removes.



(a) w:50% r:50% (b) w:20% r:80% (c) w:1% r:99%

Figure: Single-item operations only. Throughput (operations/μs) on the y-axis and thread count on the x-axis. The sub-figures are ordered in increasing amount of lookups.