Efficient Support for Range Queries and Range Updates Using Contention Adapting Search Trees

Kostis Sagonas
Kjell Winblad

Department of Information Technology
Uppsala University, Sweden

Sept 9, 2015

UPPSALA
UNIVERSITET

Background

- Multicores are everywhere
- Scalable Concurrent Data Structures
  - Ordered Sets, Key-Value Stores, etc.

UPPSALA
UNIVERSITET

Background

- Multicores are everywhere
- Scalable Concurrent Data Structures
  - Ordered Sets, Key-Value Stores, etc.
- Lock-based, Lock-free, Transactional Memory

UPPSALA
UNIVERSITET

Background

- Multicores are everywhere
- Scalable Concurrent Data Structures
  - Ordered Sets, Key-Value Stores, etc.
- Lock-based, Lock-free, Transactional Memory
- Many support single key operations
  - insert, remove, lookup

UPPSALA
UNIVERSITET

Background

- Multicores are everywhere
- Scalable Concurrent Data Structures
  - Ordered Sets, Key-Value Stores, etc.
- Lock-based, Lock-free, Transactional Memory
- Many support single key operations
  - insert, remove, lookup
- Few support more complex operations
  - range queries, bulk updates, etc.

UPPSALA
UNIVERSITET

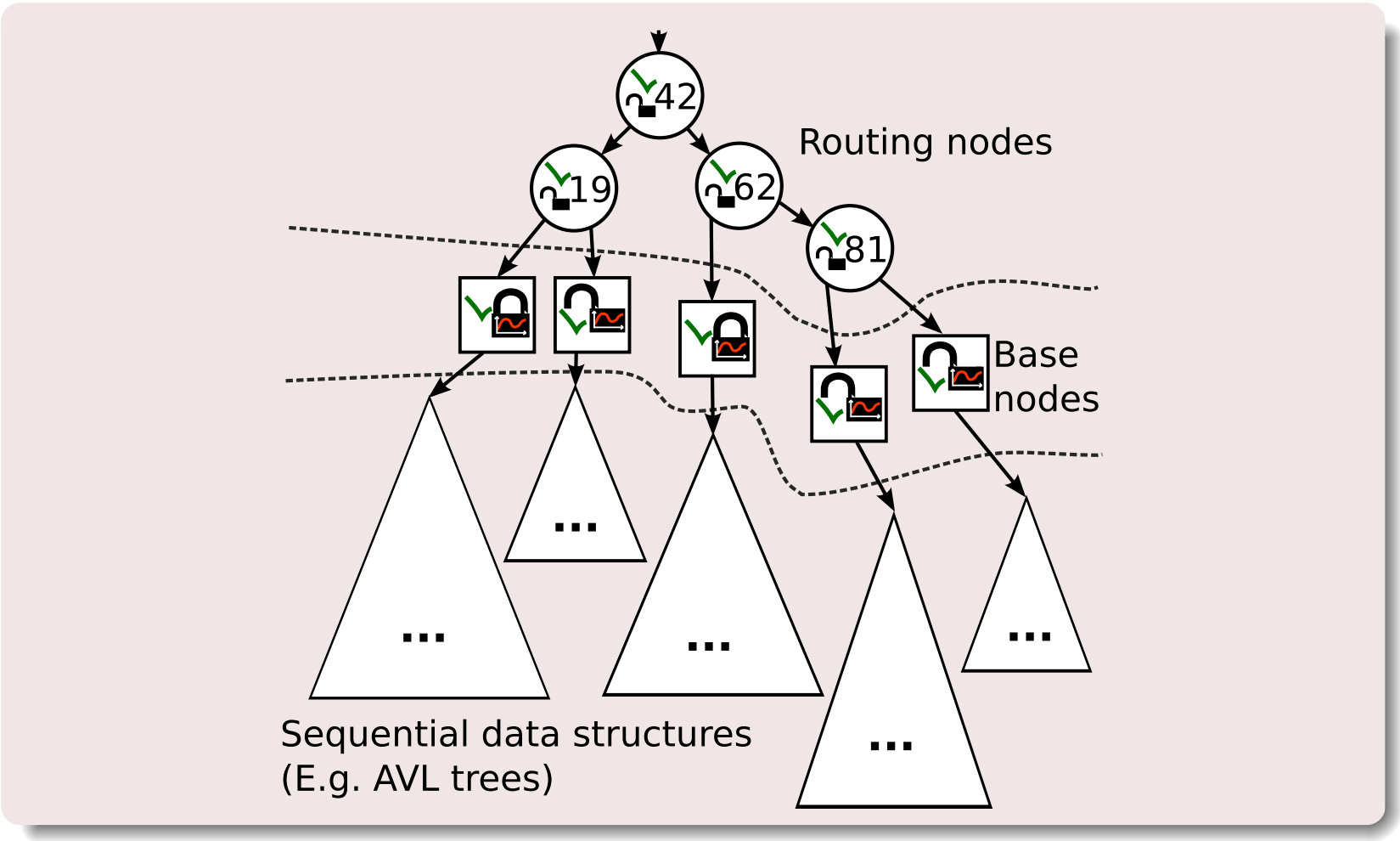
Background

- Multicores are everywhere
- Scalable Concurrent Data Structures
  - Ordered Sets, Key-Value Stores, etc.
- Lock-based, Lock-free, Transactional Memory
- Many support single key operations
  - insert, remove, lookup
- Few support more complex operations
  - range queries, bulk updates, etc.
- **Fixed synchronisation granularity**

Can we do better by dynamically adapting?

- Contention Adapting Search Trees (CA trees)
  - Lock-based
  - Adapt synchronisation granularity to fit workload
  - Support for complex operations
    - Bulk operations
    - Range operations

CA Tree Structure



Contention Statistics Collecting Lock

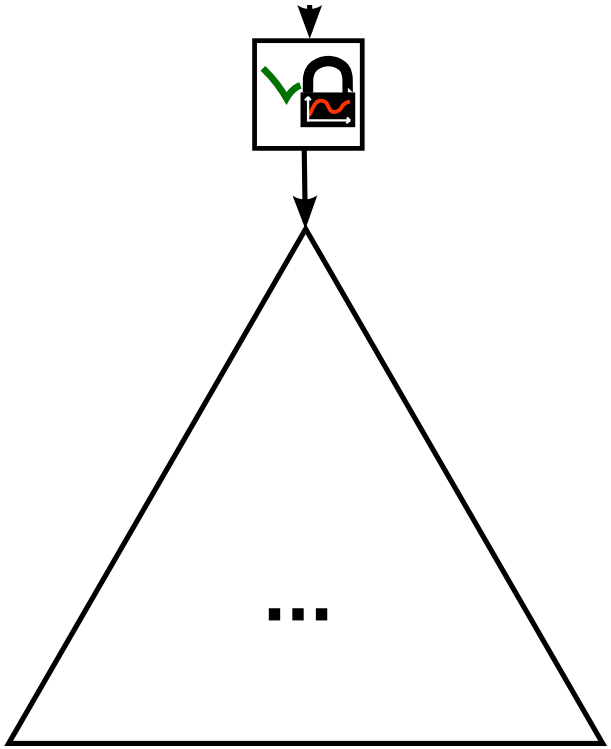
```
struct StatLock {
  Lock lock;
  int statistics;
}

void statLock(StatLock slock) {
  if (tryLock(slock.lock)) {
    slock.statistics -= SUCCESS_CONTRIBUTION;
    return;
  }
  lock(slock.lock);
  slock.statistics += FAIL_CONTRIBUTION;
}
```

UPPSALA
UNIVERSITET

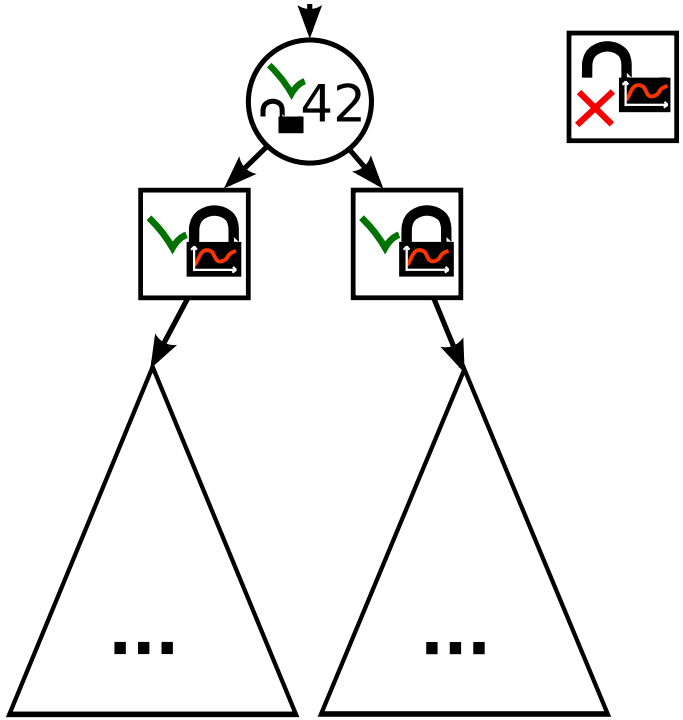
Contention Adaptation

```
...

performOperation(base.root, parameters...);

if (base.lock.statistics > MAX_CONTENTION) {
  highContentionSplit(tree, base, prevNode);
} else if (base.lock.statistics < MIN_CONTENTION) {
  lowContentionJoin(tree, base, prevNode);
}
statUnlock(base.lock)
```

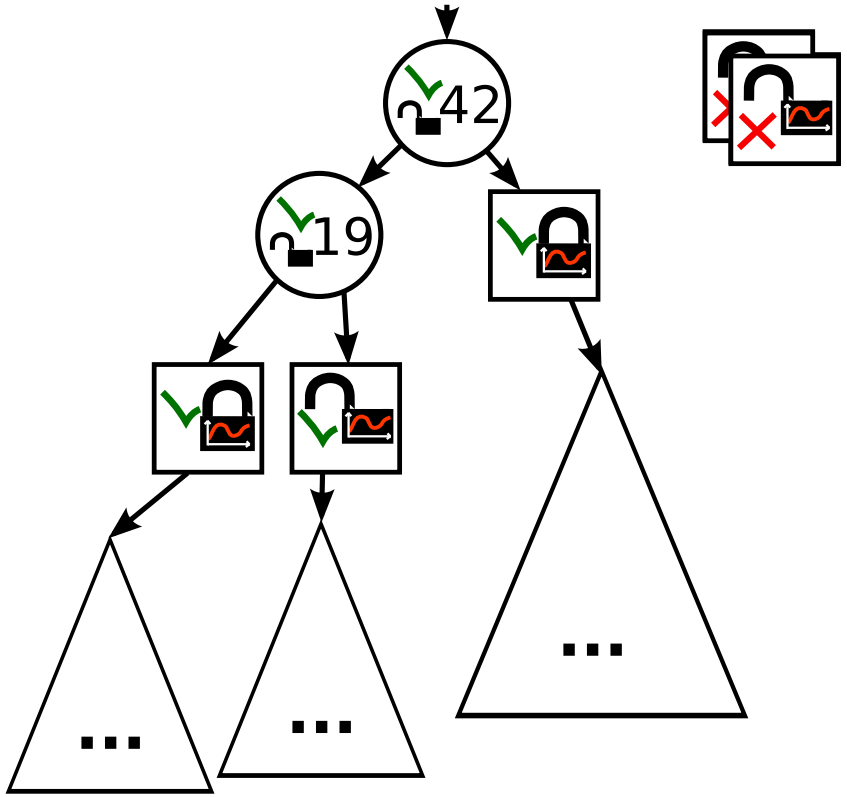
CA Tree Animation



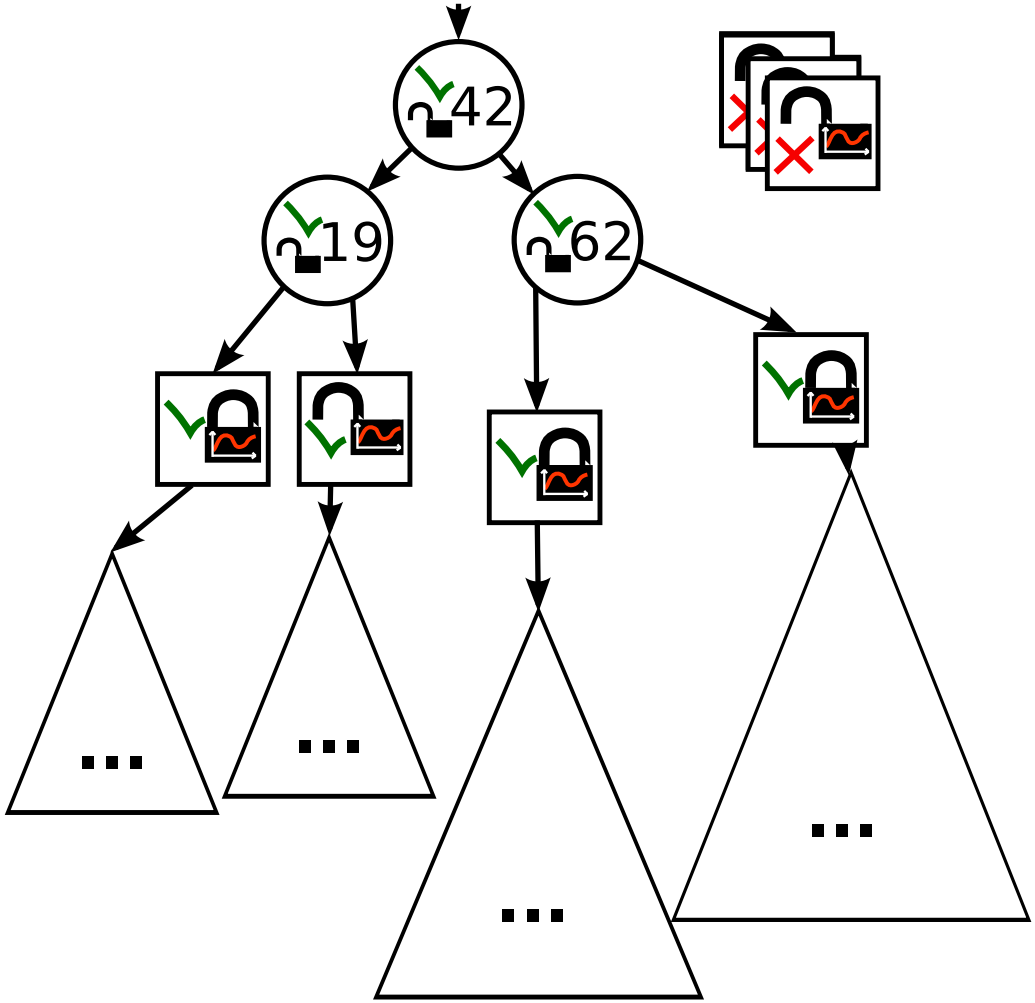
CA Tree Animation



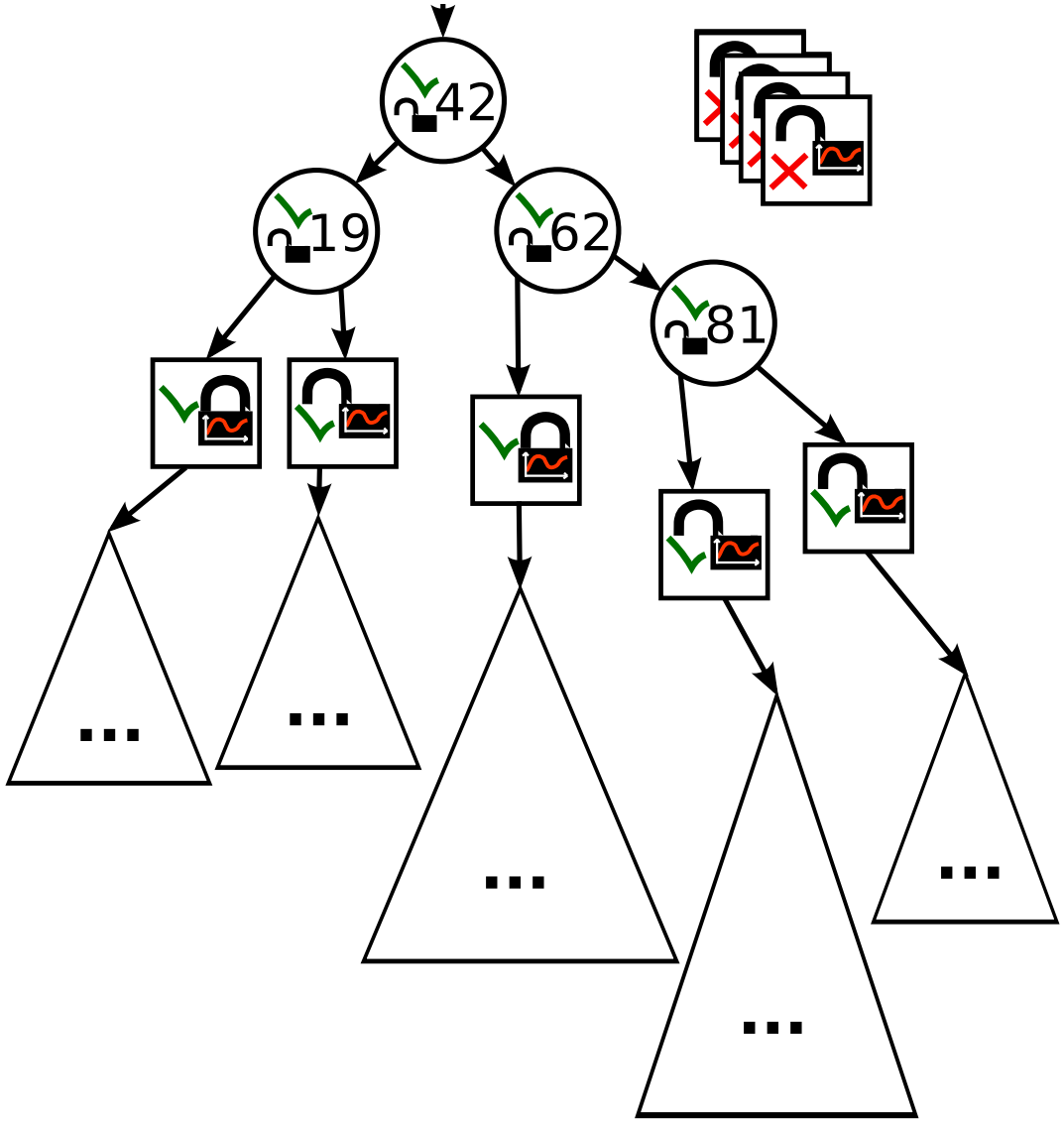
CA Tree Animation



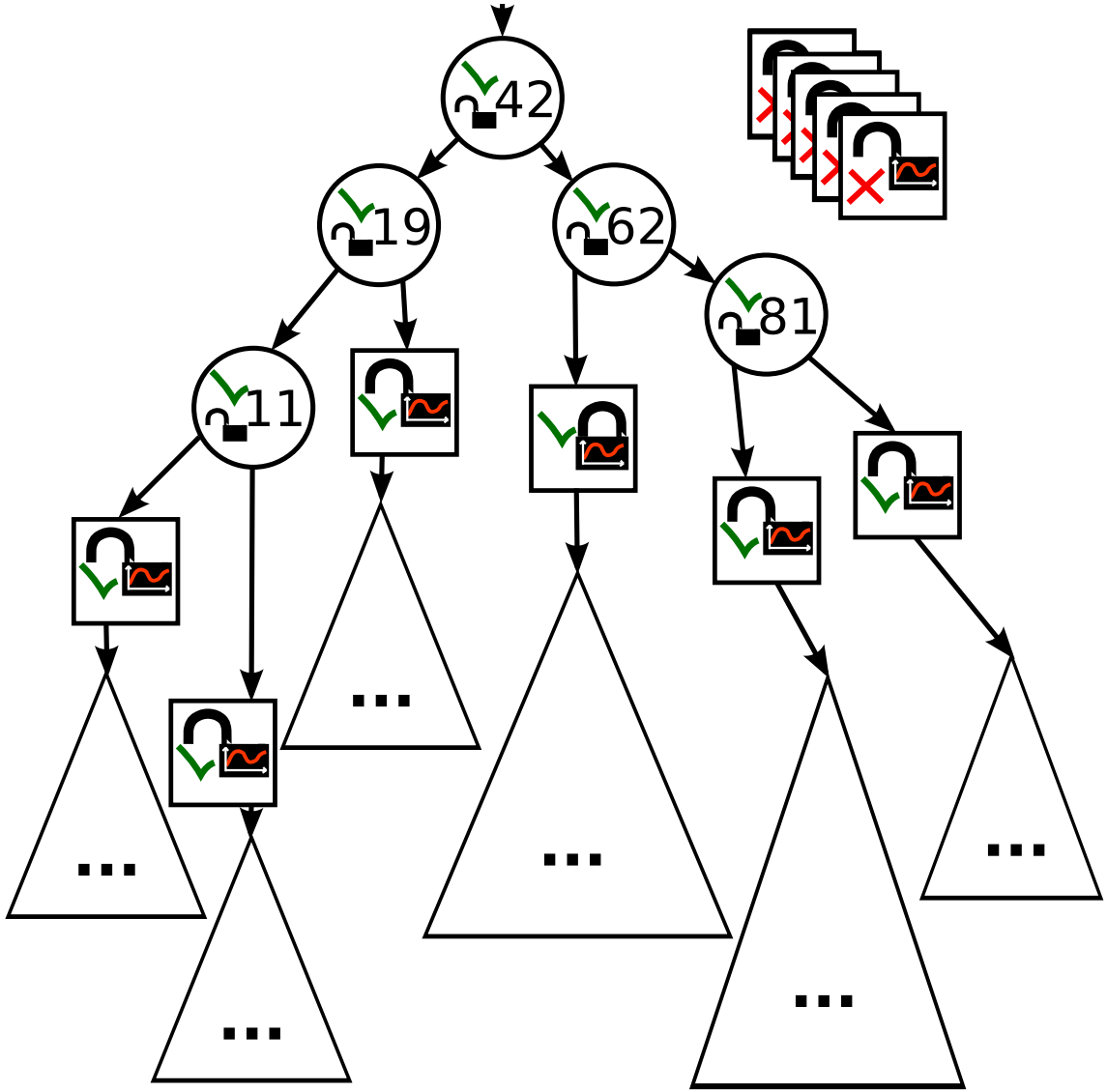
CA Tree Animation



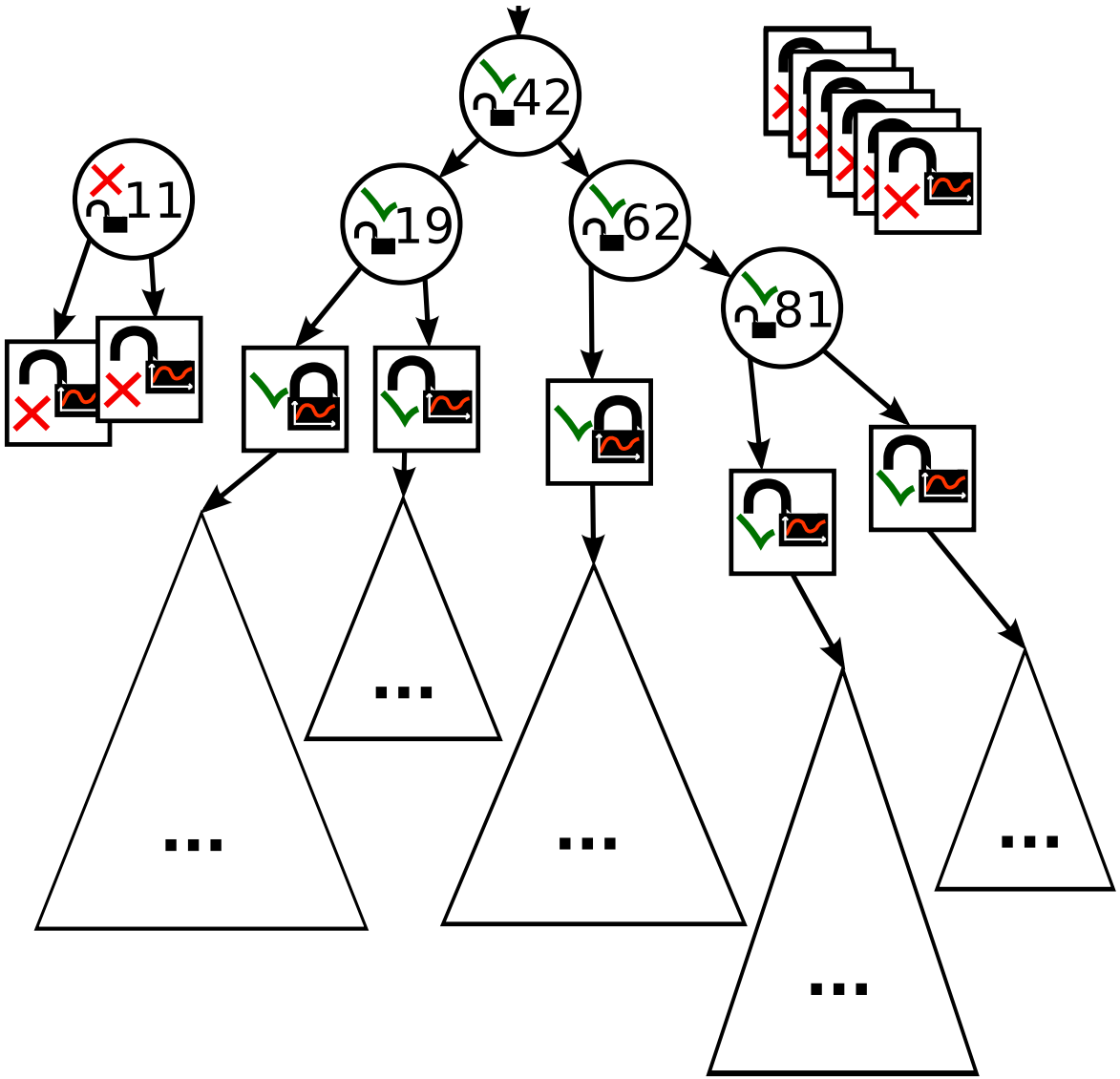
CA Tree Animation



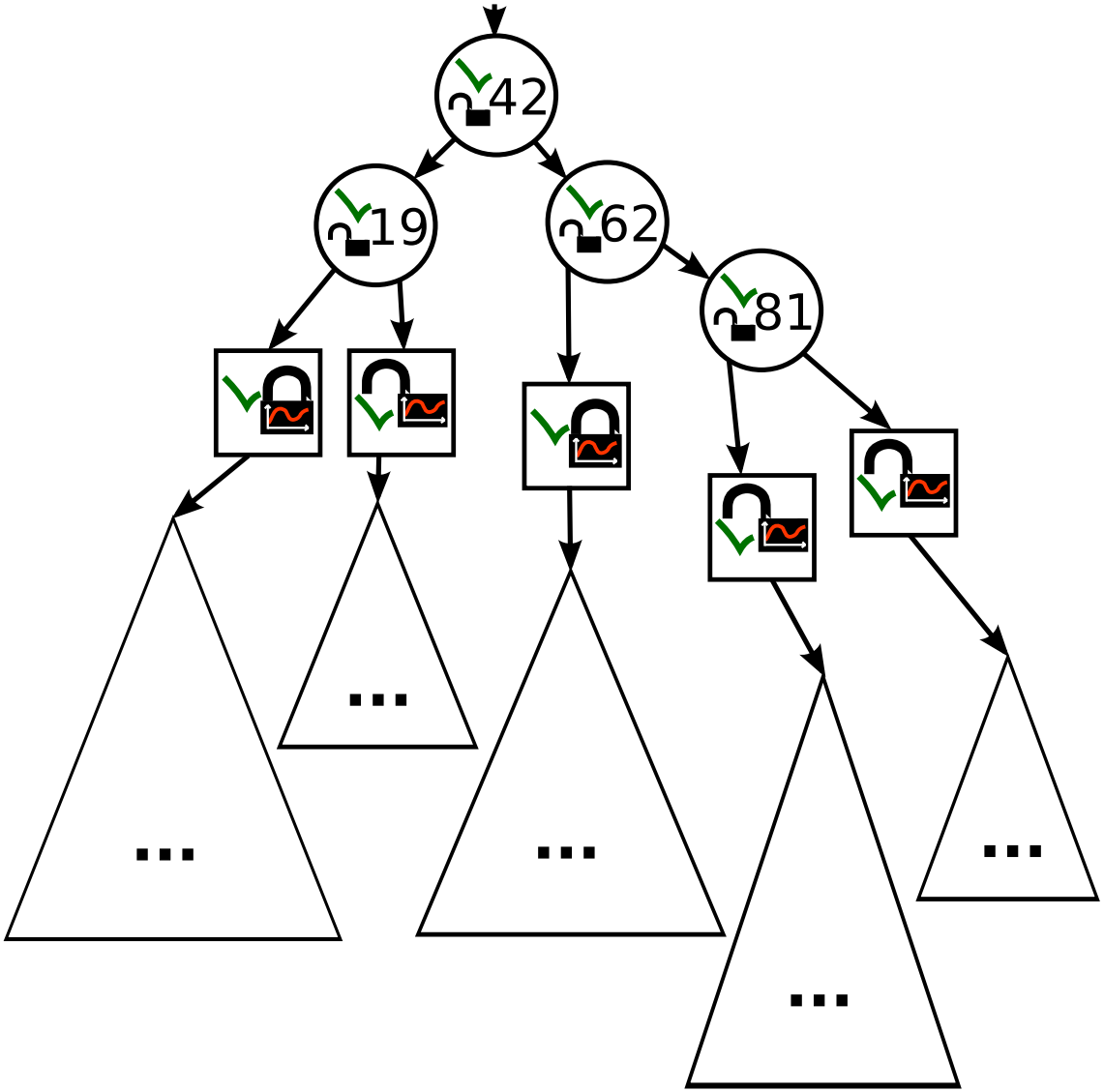
CA Tree Animation



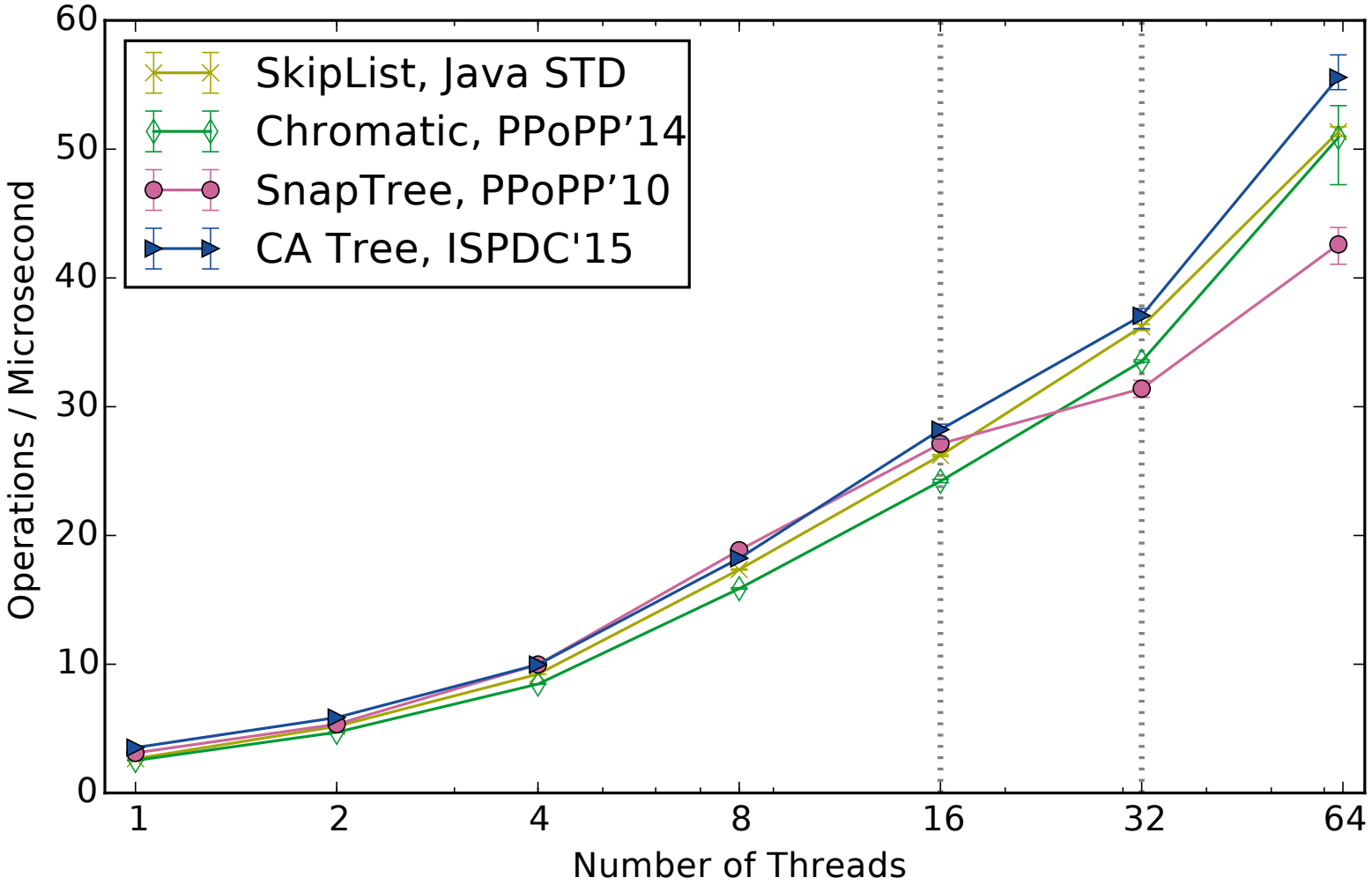
CA Tree Animation



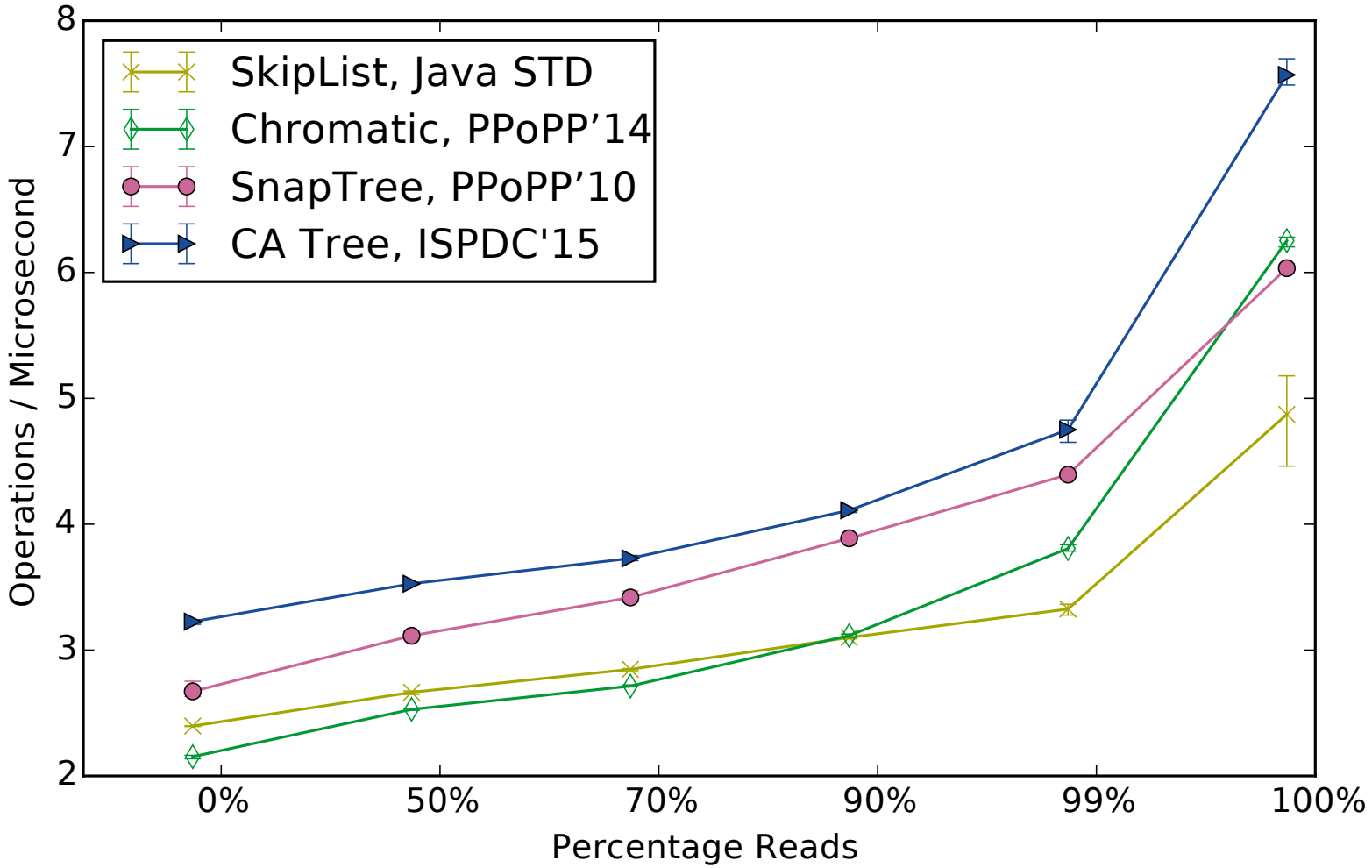
CA Tree Animation



Size 10000, 50% Update, 50% Lookup



Size 10000, Sequential Throughput



UPPSALA UNIVERSITET

Multi-key Operations

Example Operations

- Bulk insert
- Bulk remove
- Swap
- ...

Algorithm Ideas

- Sort keys (to prevent deadlock)
- Find and lock base nodes

Range Operations

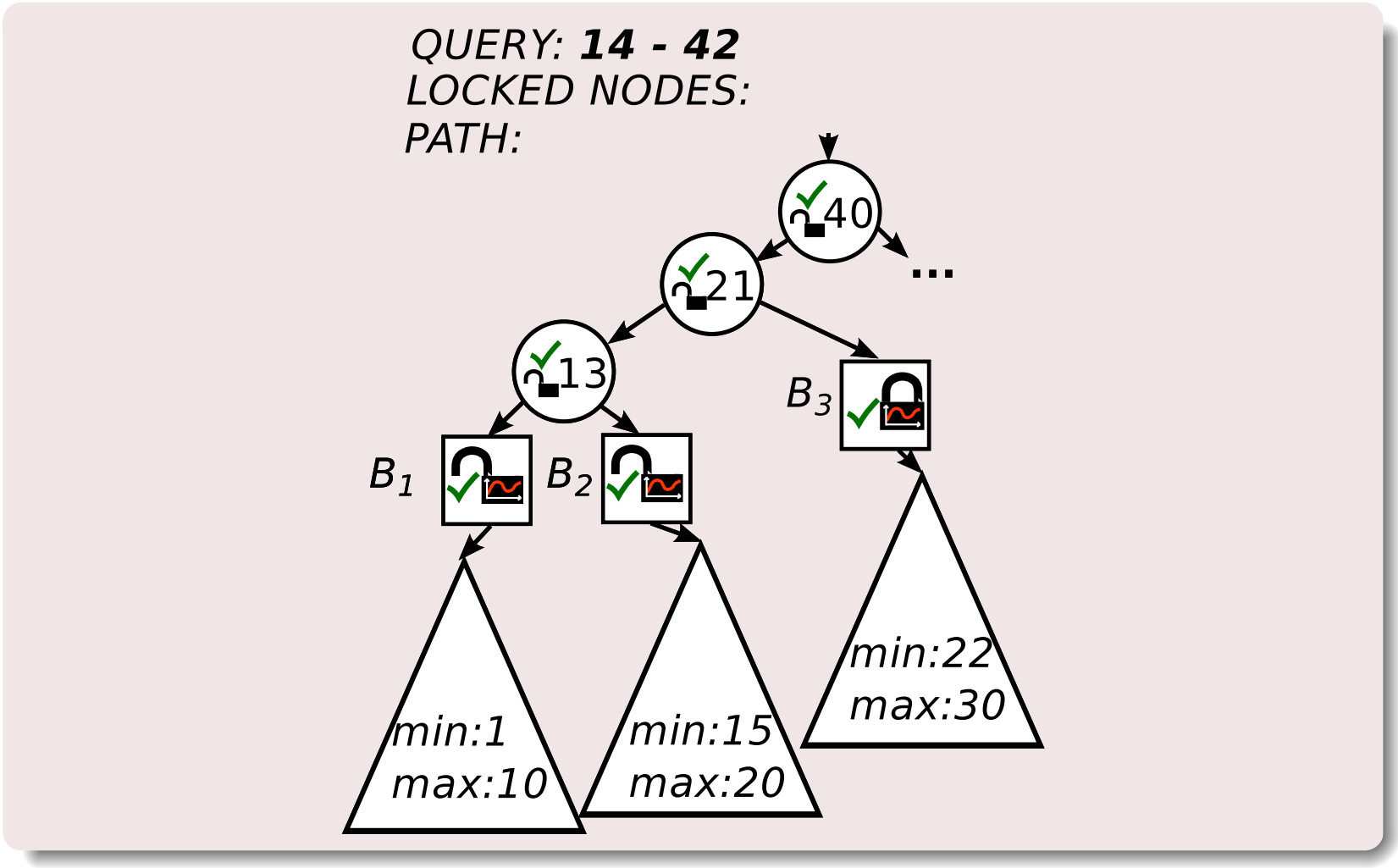
Range query atomic snapshot of all entries with keys in a range

Range update atomic update to all values with keys in a range

Range query

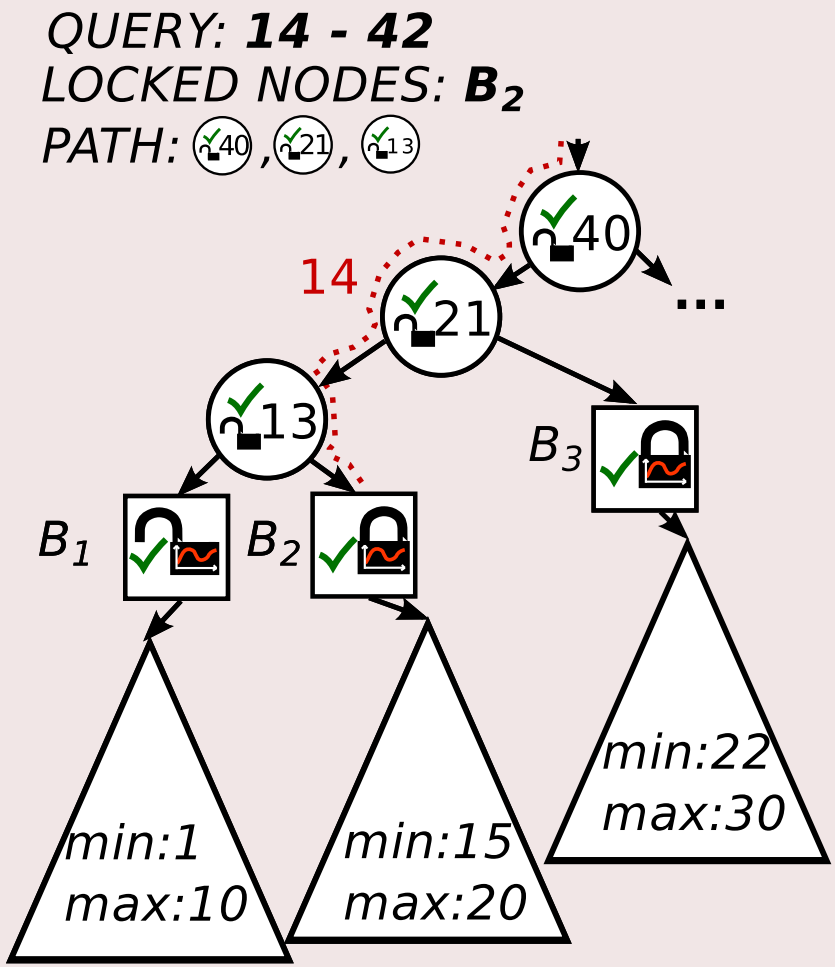
- Sequence lock optimistic attempt
- Non-optimistically if optimistic attempt fail

Range Query Example



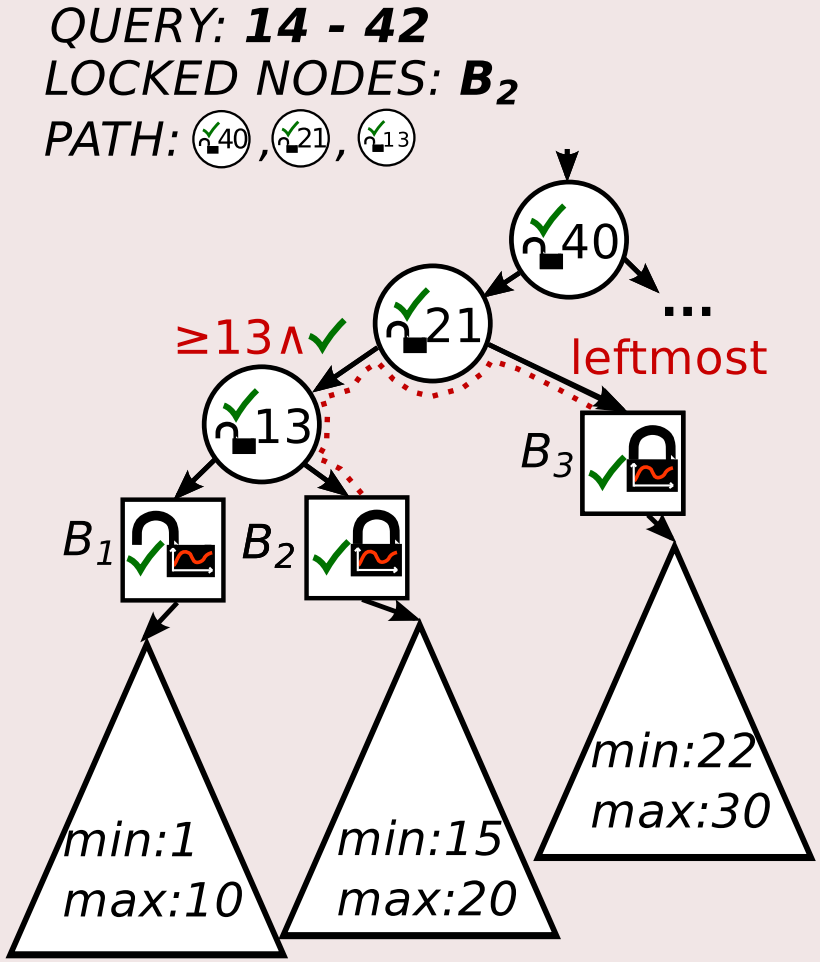
Range Query Example

Step 1: Search for the first key



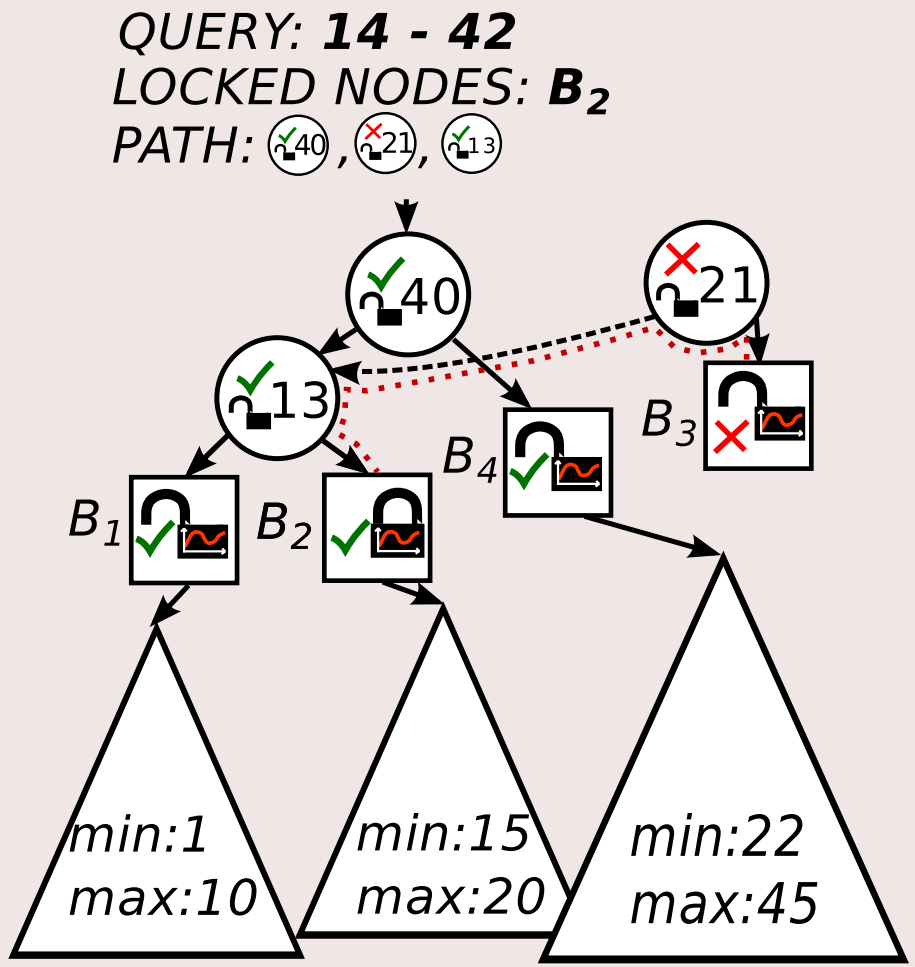
Range Query Example

Step 2: Search backwards until condition, then leftmost



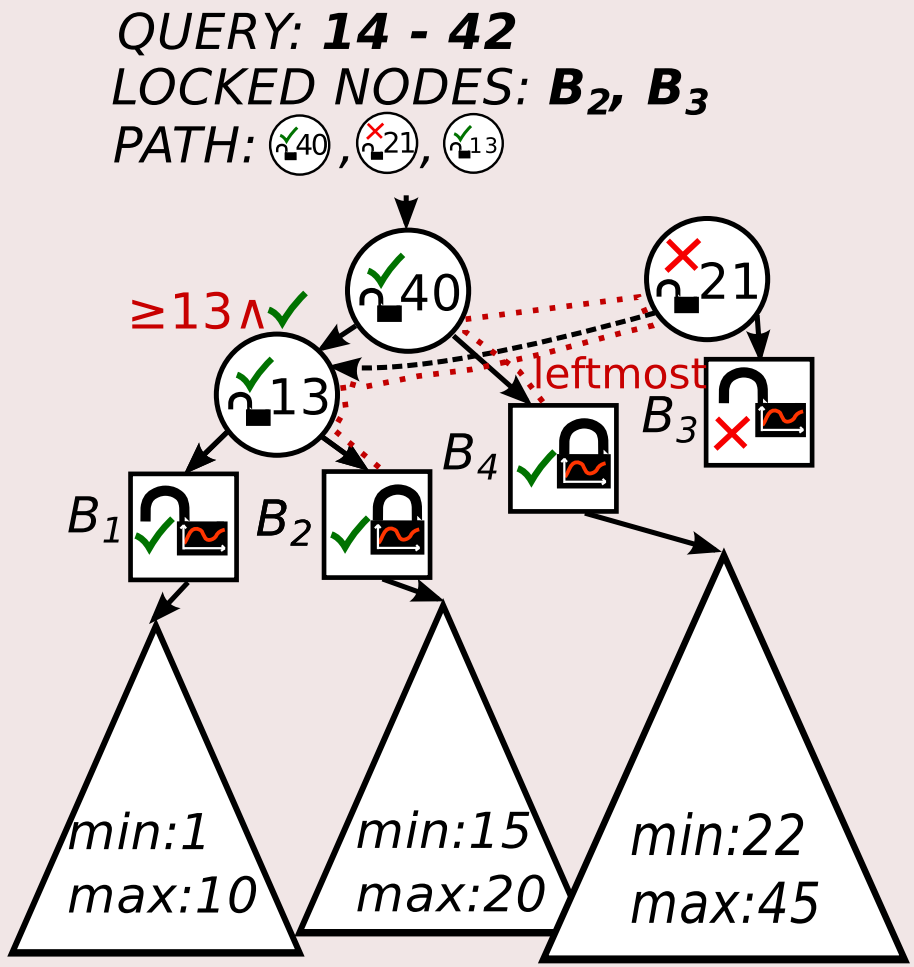
Range Query Example

Step 3: Retry when invalid node is found



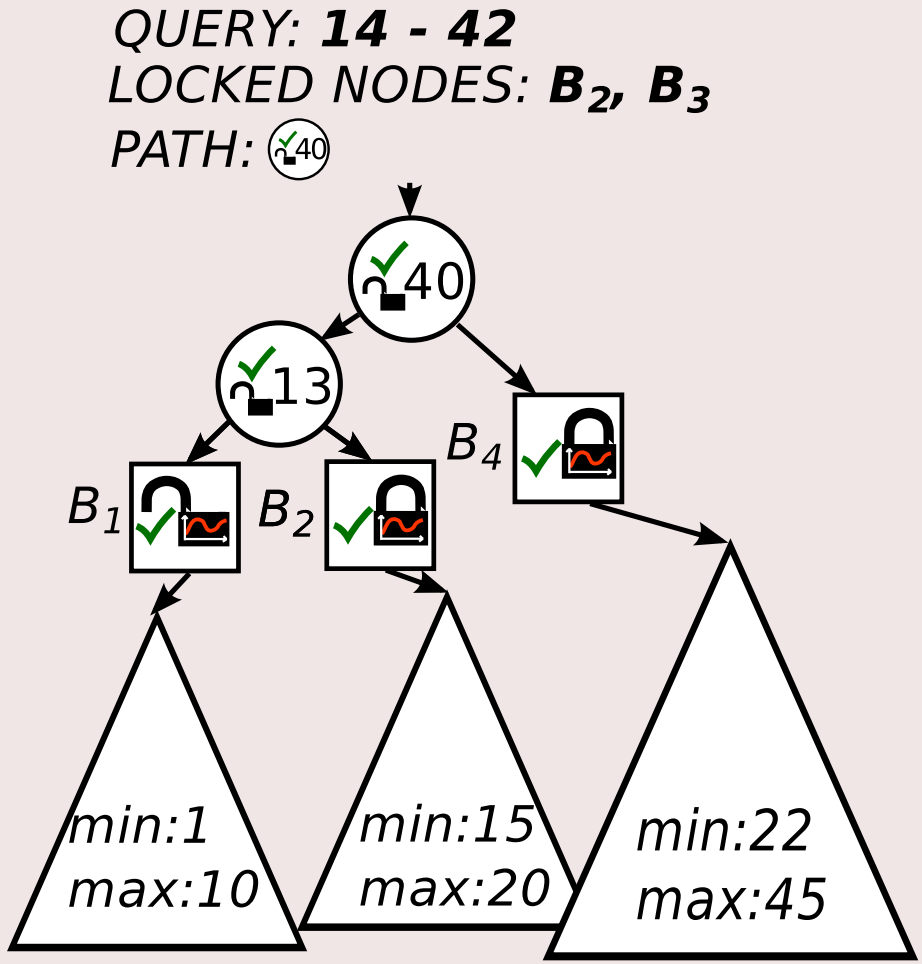
Range Query Example

Step 4: Retry



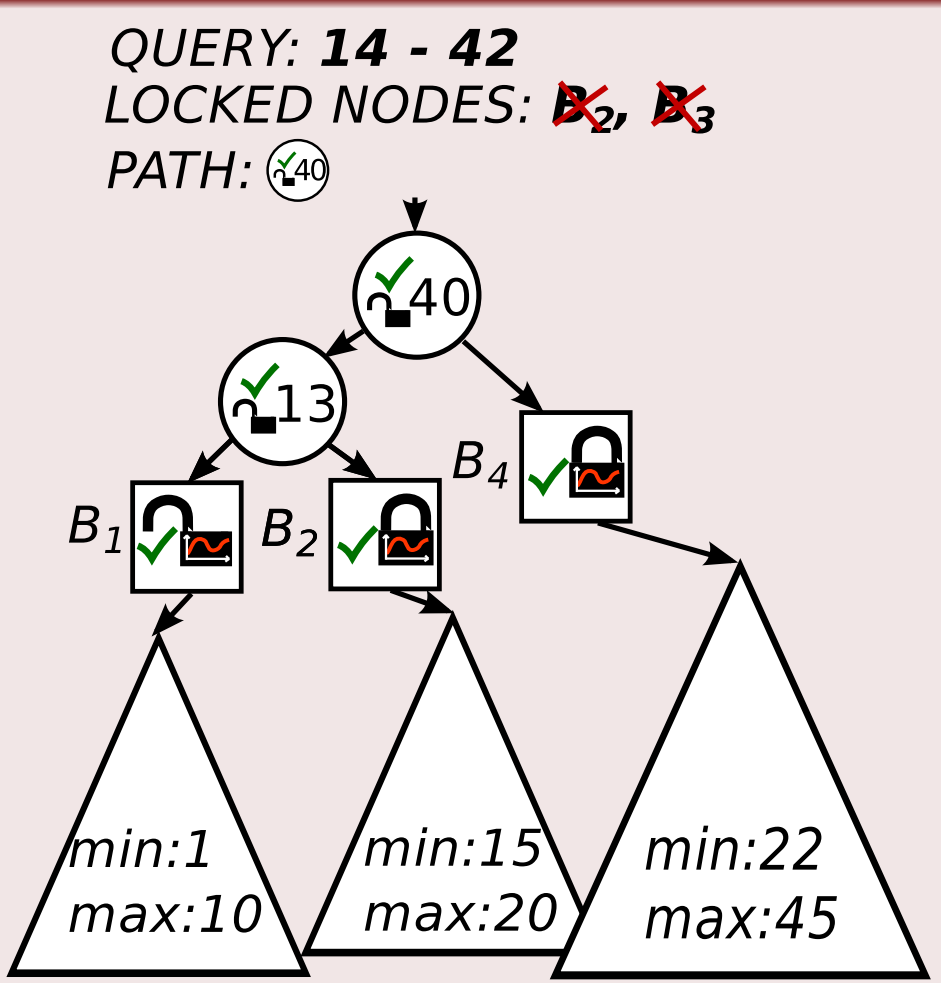
Range Query Example

Step 5: Path updated to last locked base node



Range Query Example

Step 6: Perform operation on base nodes and unlock



UPPSALA UNIVERSITET

Contention Statistics: multi-key operations

Contention Statistics

- One base node
  - As for single key operation
- Several base nodes
  - Decrease contention statistics
- Successful optimistic read attempt
  - Don't change contention statistics

Adaptation

- On the last unlocked base node
  - In the same way as single key operations

UPPSALA
UNIVERSITET

Properties (see paper)

- Linearizability
- Deadlock freedom
- Livelock freedom

Evaluation

- ***CA Tree-AVL*** – CA tree with **AVL trees** in base nodes
- ***CA Tree-FatSL*** – CA tree with **fat node skip lists** in base nodes
- **SkipList** – Lock-free skip list (**no atomic range queries**)
  - From Java Standard Library (D. Lea)
- **SnapTree** – O(1) snapshot by copy on write [PPoPP'10]
  - N. G. Bronson, J. Casper, H. Chafi, and K. Olukotun
- **k-ary** – Lock-free search tree with k elements in nodes
  - T. Brown and H. Avni [PODC'11]

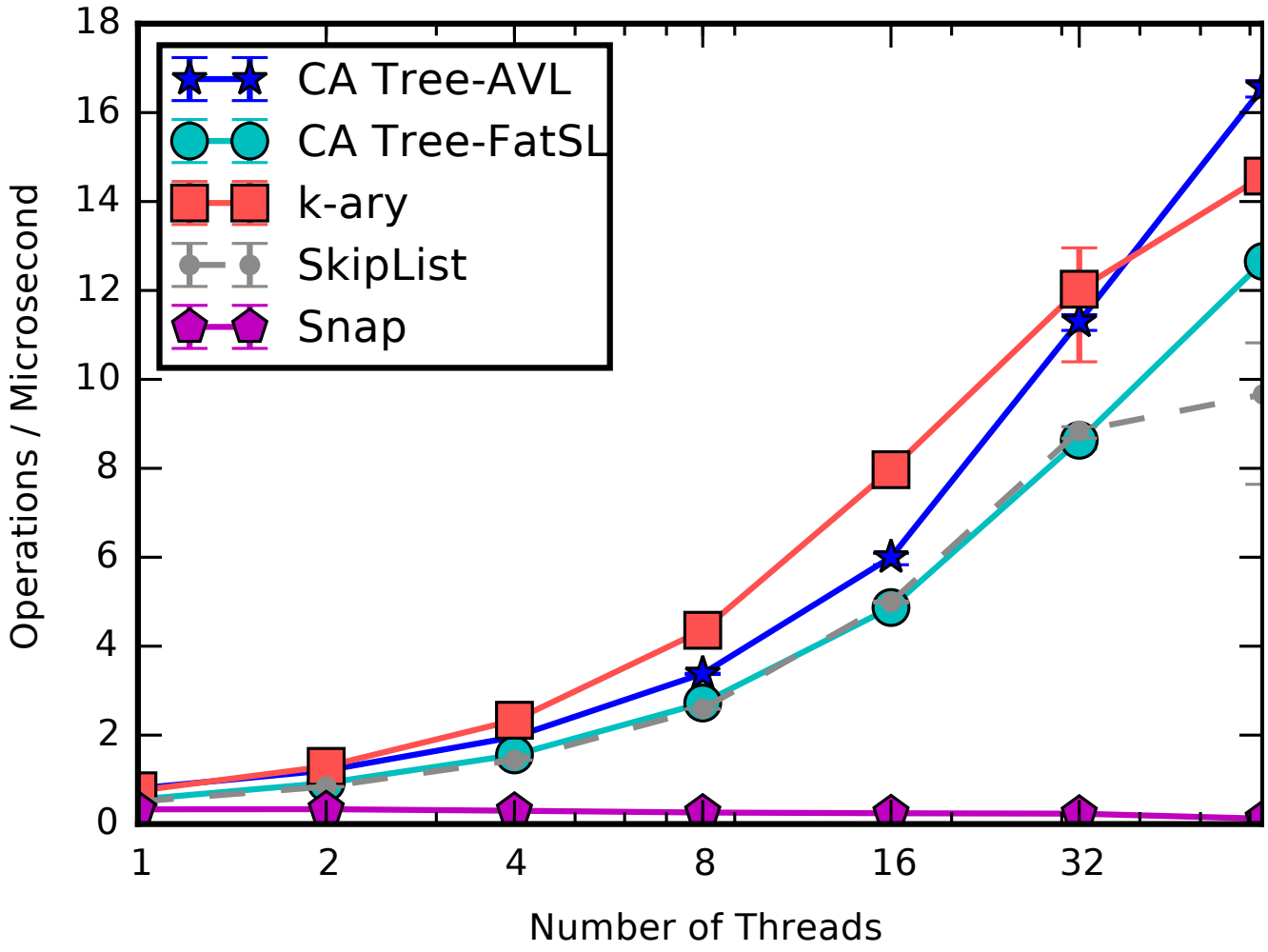
UPPSALA UNIVERSITET

Evaluation

- Benchmark with a mix of
  - Inserts
  - Removes
  - Lookups
  - **Range Queries of size up to** *max*
  - **Range Updates of size up to** *max*
- Platform
  - NUMA with four AMD Opteron 6276 (2.3 GHz)
    16 cores each = **64 physical cores**
- Implementation in Java

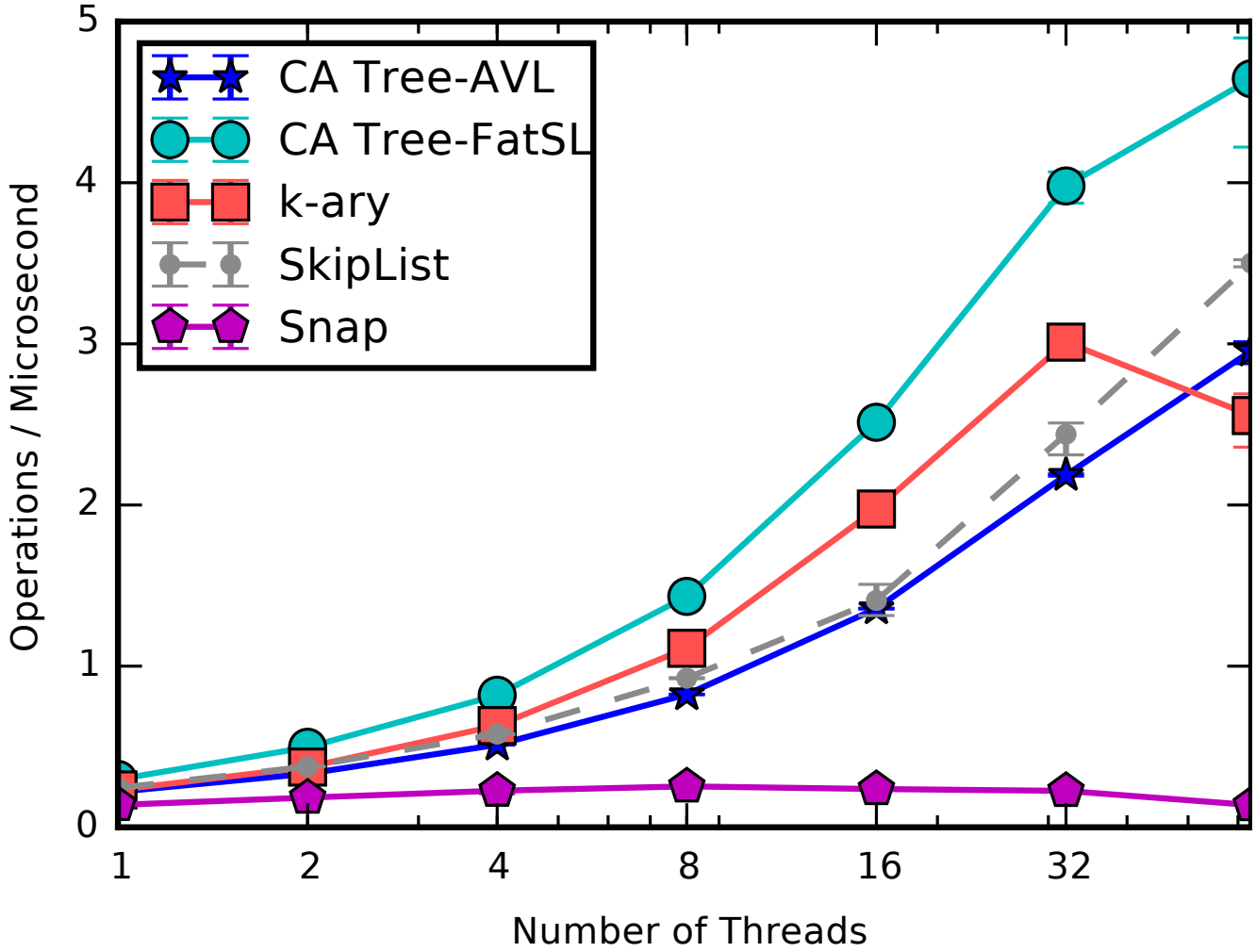
Small Range Queries

Inserts:10%, Removes:10%, Lookups:55%,
Queries:25%-max:10



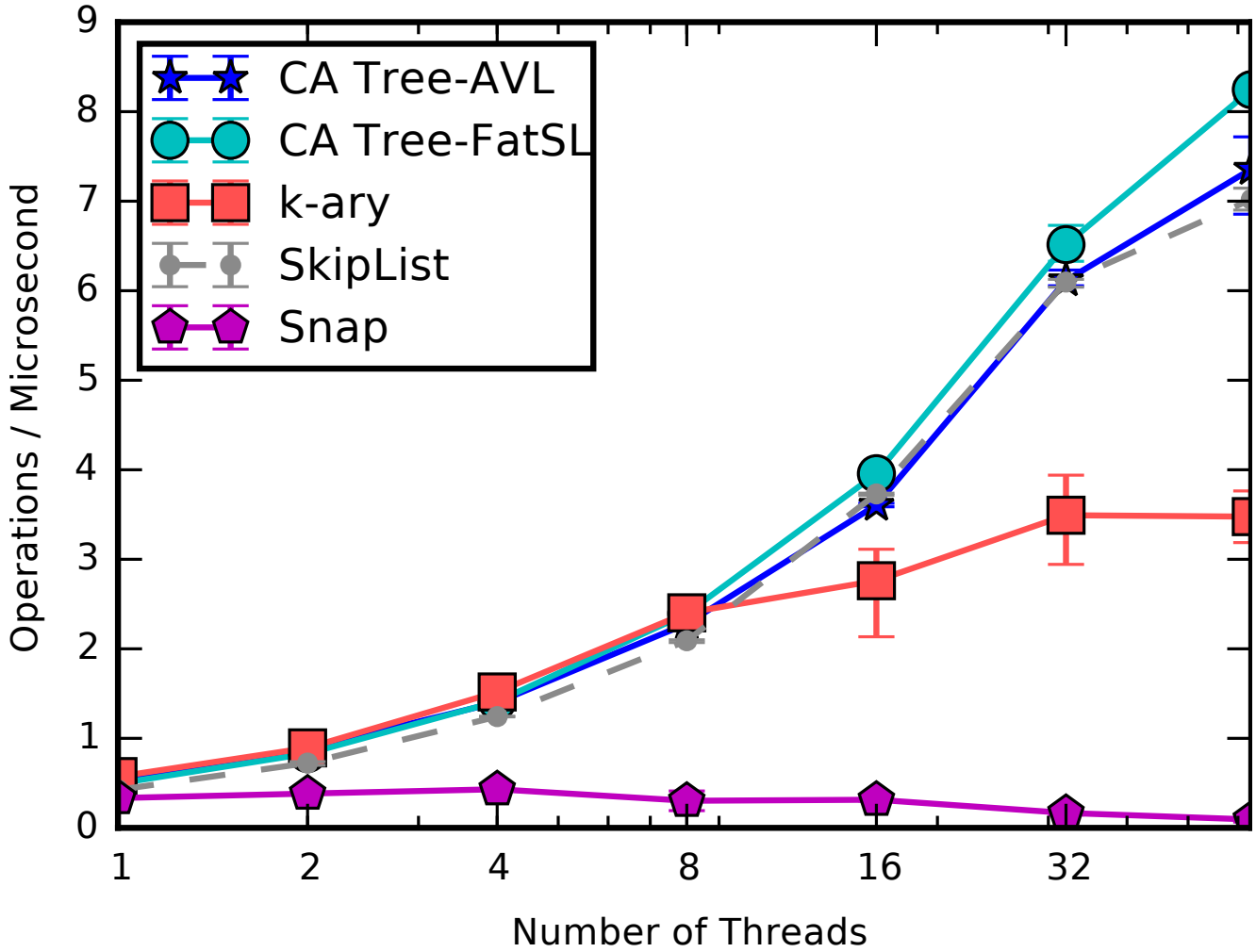
Large Range Queries

Inserts:10%, Removes:10%, Lookups:55%,
Queries:25%-max:1000



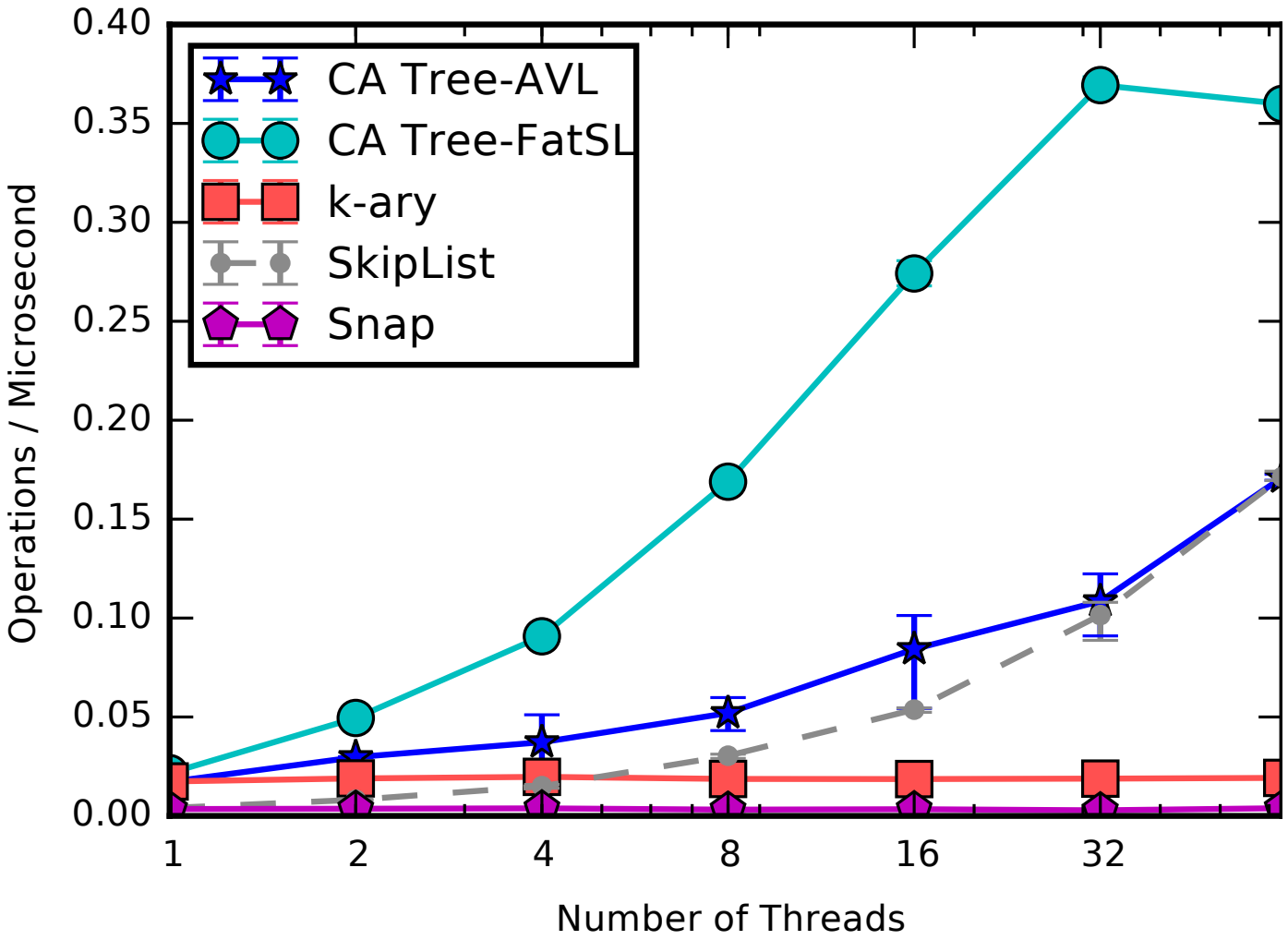
Few Range Updates

Inserts:2.5%, Removes:2.5%, Lookups:44%,
Queries:50%-max:100, Updates:1%-max:100



Large Range Updates and Queries

Inserts:1.5%, Removes:1.5%, Lookups:27%,
Queries:50%-max:10000, Updates:20%-max:10000



What to take home?

- CA trees scale well for range operations
- Key advantage: Naturally adapt to fit workload
- Plug-and-play: sequential data structure
  - Different performance characteristics