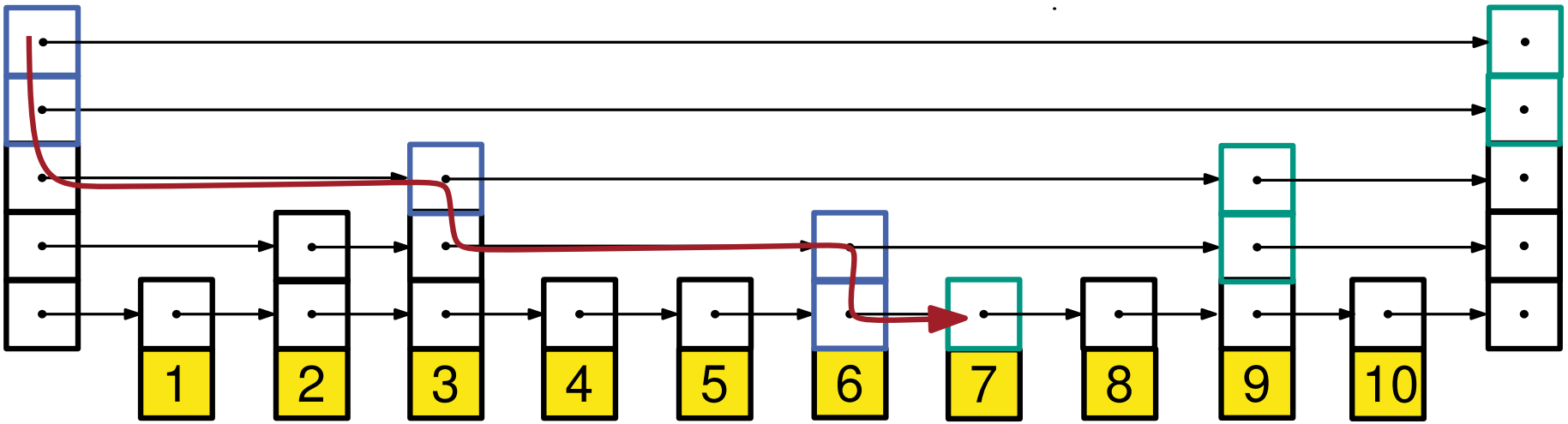
Karlsruhe Institute of Technology

Concurrent Skiplists: A Probabilistic Alternative to Balanced Trees

A Practical Course · Efficient Parallel C++ · March 3, 2022

Manuel Haag

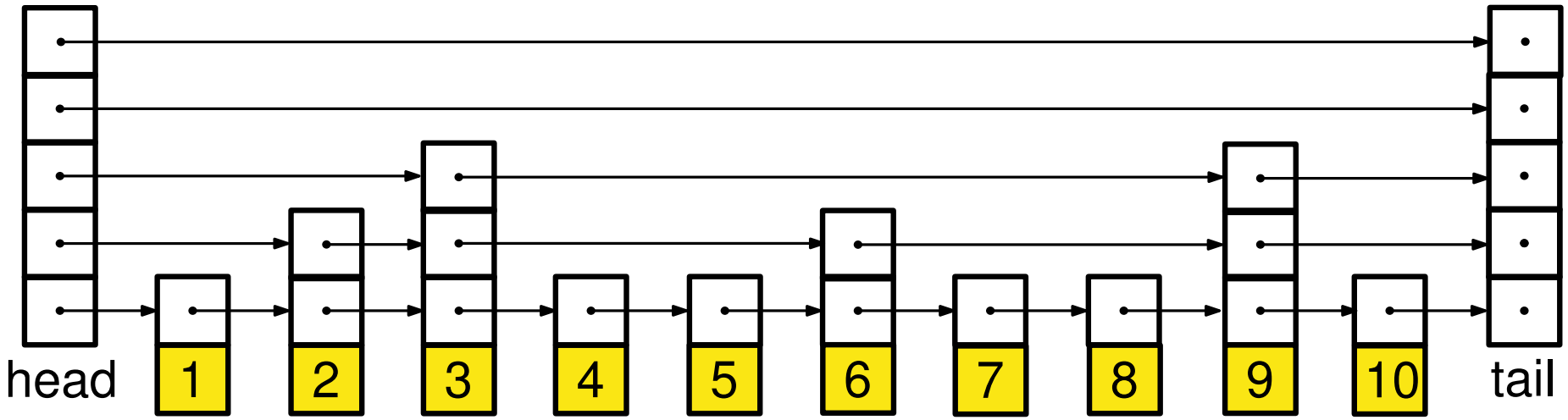
INSTITUTE OF THEORETICAL INFORMATICS · ALGORITHMICS GROUP



What is a Skiplist?

Goal: dynamic ordered set

Operations: `search, insert, remove`



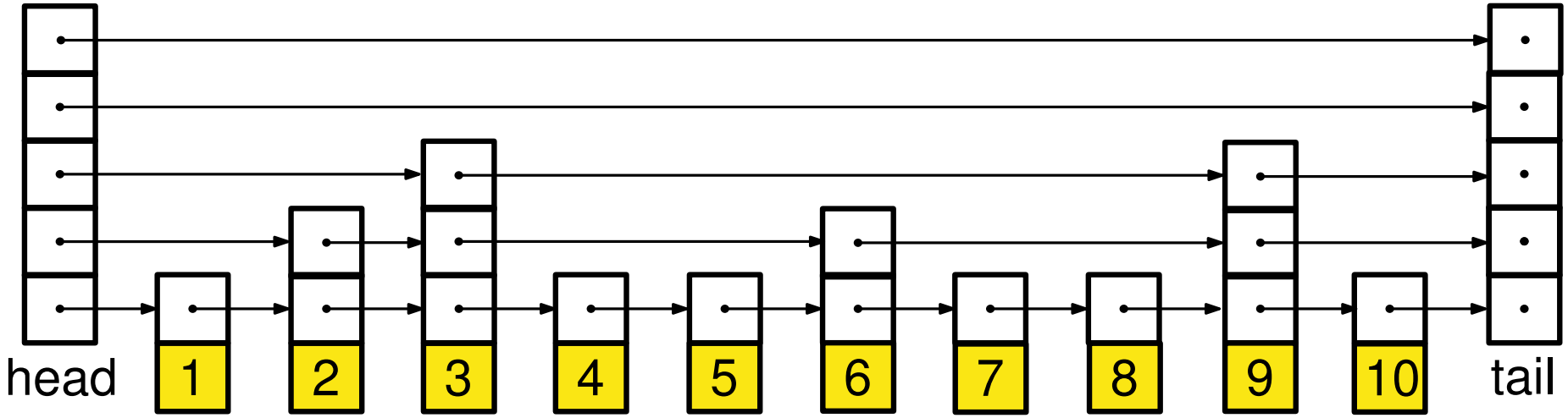
Institute of Theoretical Informatics
Algorithmics Group

What is a Skiplist?

Goal: dynamic ordered set

Operations: `search, insert, remove`

- bottom layer is linked list
- create nodes with random heights $\rightarrow$ “fast lane” allows skipping nodes
- $P(\text{increase height by } k) = p^k(1-p) \sim Geo(1-p)$



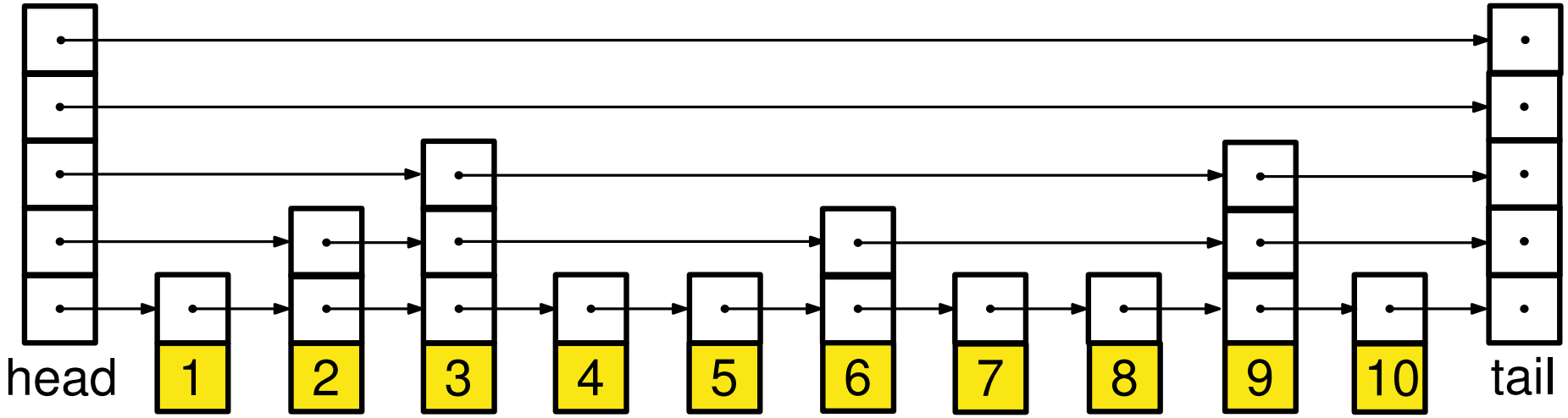
Institute of Theoretical Informatics
Algorithmics Group

What is a Skiplist?

Goal: dynamic ordered set

Operations: `search, insert, remove`

- bottom layer is linked list
- create nodes with random heights $\rightarrow$ “fast lane” allows skipping nodes
- $P(\text{increase height by } k) = p^k(1-p) \sim Geo(1-p)$



Complexity: $\mathbb{E}[\text{cost}] = \frac{1}{p}\log_{\frac{1}{p}}(n) \in \mathcal{O}(\log n)$

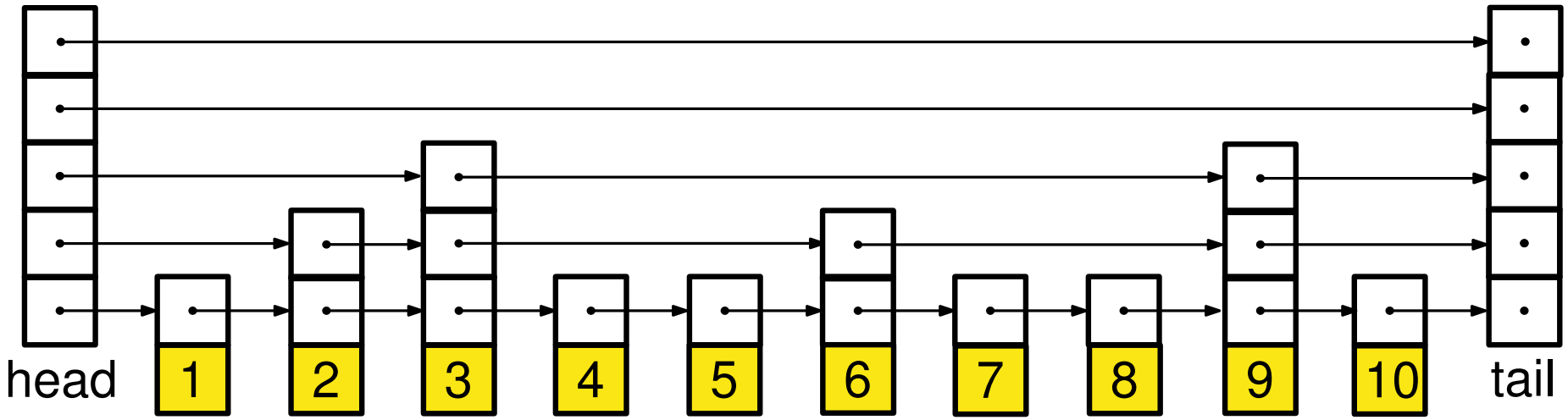
Storage: $\mathbb{E}[\text{height}] = \frac{1}{1-p} \in \mathcal{O}(n)$

Institute of Theoretical Informatics
Algorithmics Group

Extension to Indexable Skiplist

Goal: dynamic ordered set

Operations: `search, insert, remove , index, rank`



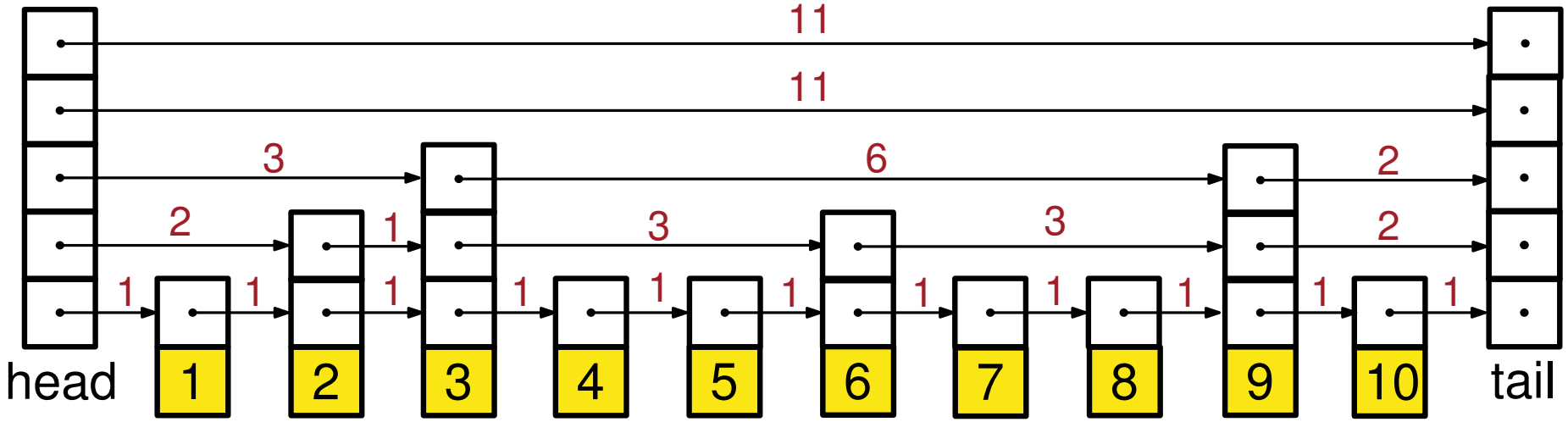
Institute of Theoretical Informatics
Algorithmics Group

Extension to Indexable Skiplist

Goal: dynamic ordered set

Operations: `search, insert, remove , index, rank`

- store length of pointers



Motivation

- up to 5 supported operations
- probabilistic balancing
- easier to parallize than balanced trees

Institute of Theoretical Informatics
Algorithmics Group

Motivation

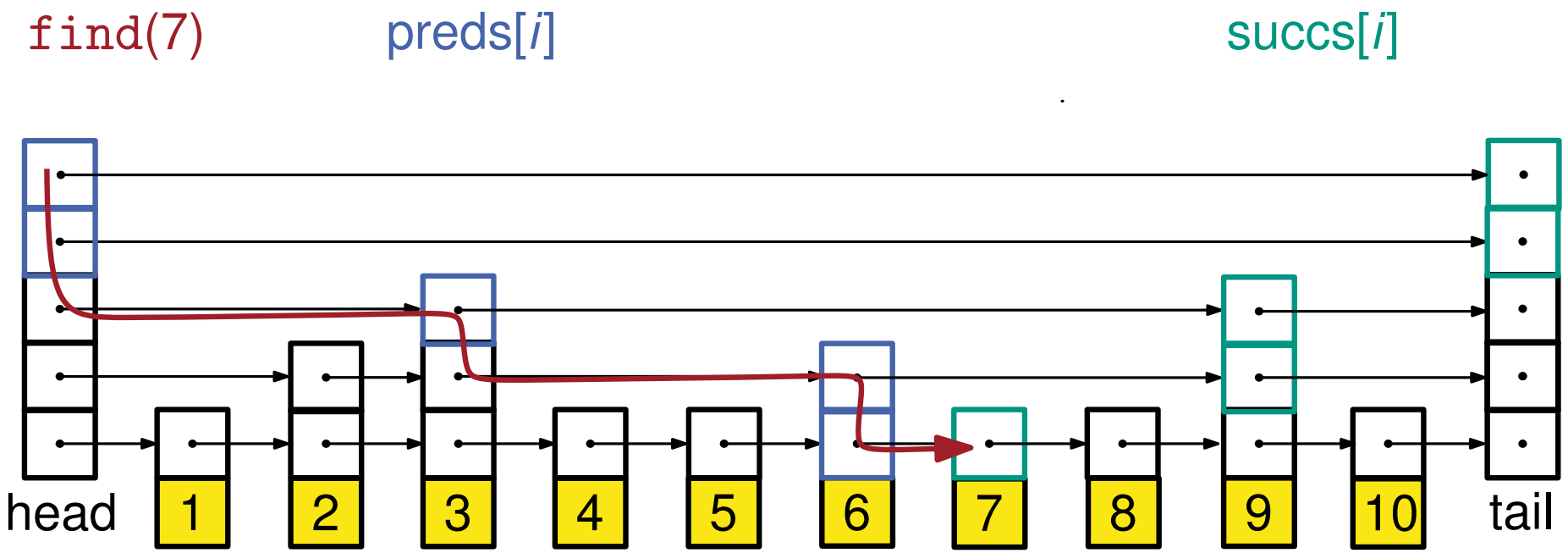
- up to 5 supported operations
- probabilistic balancing
- easier to parallize than balanced trees

Exercise: Threadsafe Skiplist

- lockbased
- lockfree
- lockbased/lockfree + indexable
- evaluation

Institute of Theoretical Informatics
Algorithmics Group

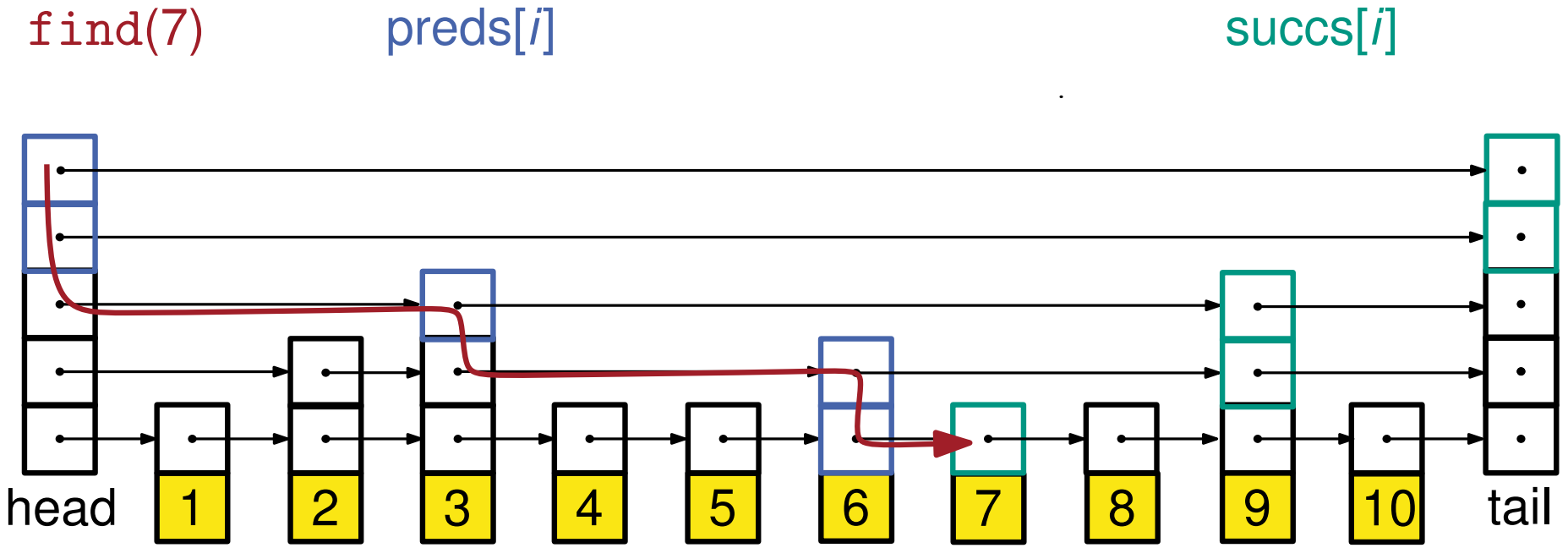
Central Operation: find



Institute of Theoretical Informatics
Algorithmics Group

Central Operation: find

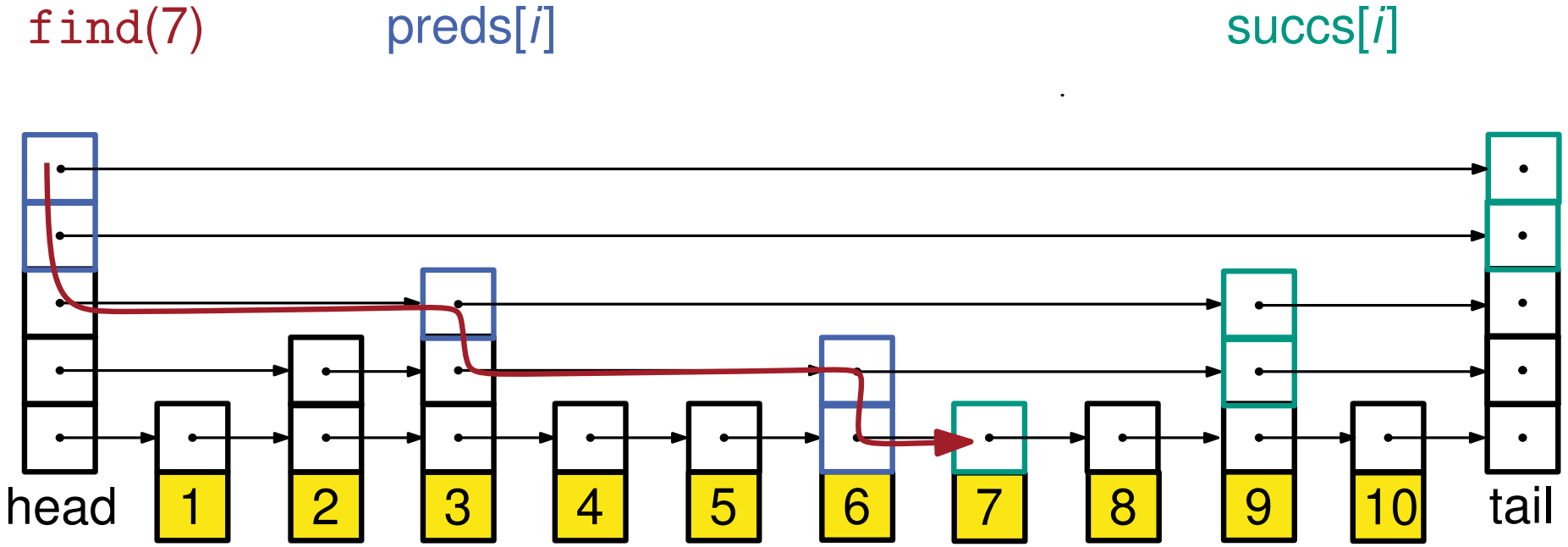
- `find`: traverse skiplist and document preds and succs ($\neq$ `search`)
- basis for other operations



Institute of Theoretical Informatics
Algorithmics Group

Central Operation: find

- `find`: traverse skiplist and document preds and succs ($\neq$ `search`)
- basis for other operations



- no locks are acquired $\rightarrow$ no contention
- leads to inconsistencies $\rightarrow$ validate and retry

Implementation Highlights: Lock-Variant

- validate: preds and succs consistent and are not removed
- flags: being deleted, fully linked
- use of Spinlocks (faster than Mutex in preliminary experiments)

Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Lock-Variant

- validate: preds and succs consistent and are not removed
- flags: being deleted, fully linked
- use of Spinlocks (faster than Mutex in preliminary experiments)

```
auto try_insert_at = [&](int j) {
    if(validate(j)) {
        preds[j] -> lock.lock();
        if(validate(j)) {
            new_node -> next[j] = succs[j];
            preds[j] -> next[j] = new_node;
            preds[j] -> lock.unlock();
            return true;
            }
        preds[j] -> lock.unlock();
    }
    find(preds, succs, insert_key); //get new preds and succs
    return false;
};
```


Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Lock-Variant

- validate: preds and succs consistent and are not removed
- flags: being deleted, fully linked
- use of Spinlocks (faster than Mutex in preliminary experiments)

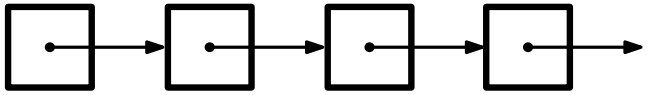
```
auto try_insert_at = [&](int j) {
    if(validate(j)) {
        preds[j] -> lock.lock();
        if(validate(j)) {
            new_node -> next[j] = succs[j];
            preds[j] -> next[j] = new_node;
            preds[j] -> lock.unlock();
            return true;
            }
        preds[j] -> lock.unlock();
    }
    find(preds, succs, insert_key); //get new preds and succs
    return false;
};
```

- thread to succeed at first level links remaining levels
- new_node accessible after first linking $\rightarrow$ lock until the end
- remove similar

Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Lockfree-Variant

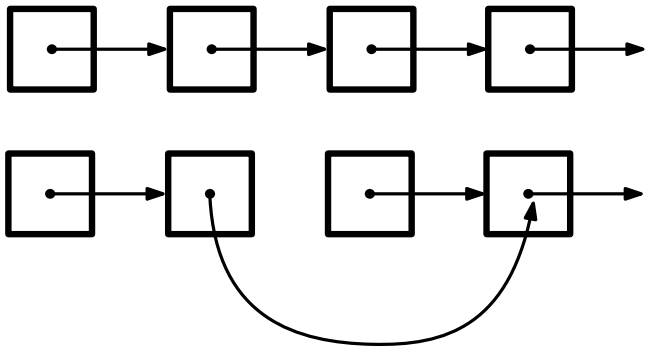
Problem: no validation and can only change one pointer atomically



Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Lockfree-Variant

Problem: no validation and can only change one pointer atomically

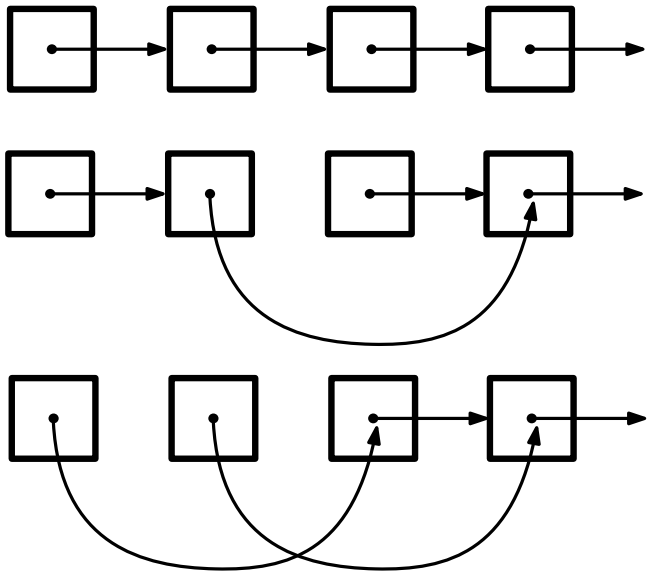


3. removed

Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Lockfree-Variant

Problem: no validation and can only change one pointer atomically

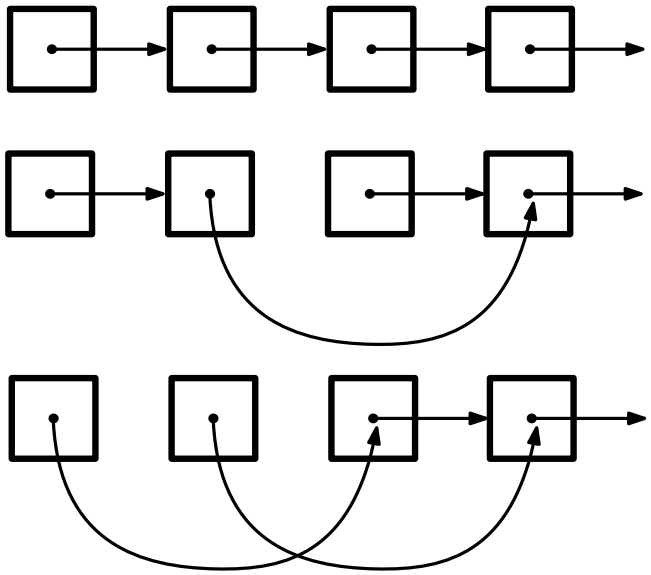


3. removed

2. removed, but 3. inserted again!

Implementation Highlights: Lockfree-Variant

Problem: no validation and can only change one pointer atomically



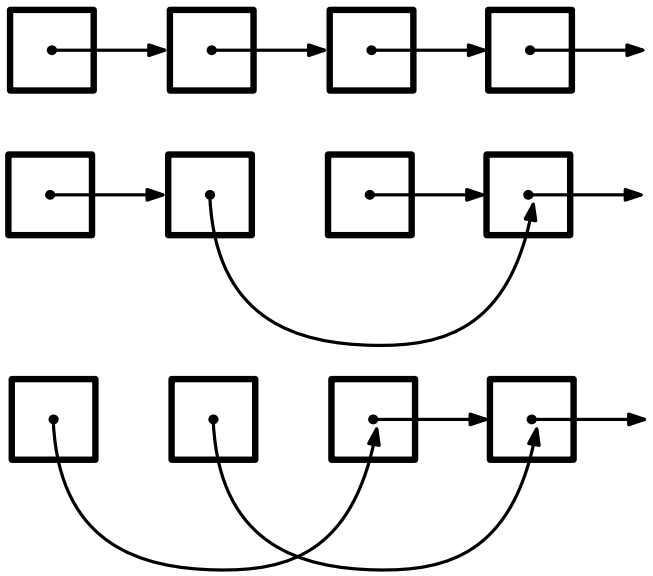
3. removed

2. removed, but 3. inserted again!

Solution: only remove from left to right

Implementation Highlights: Lockfree-Variant

Problem: no validation and can only change one pointer atomically



Solution: only remove from left to right

- mark outgoing pointers of removed nodes
- unlink nodes lazy in `find`

Implementation Highlights: Lockfree-Variant

Trick: last 16 bits of 64bit-pointer are not used

$\rightarrow$ store bool in pointer address

$\rightarrow$ allows atomically setting pointer and marked flag

Implementation Highlights: Lockfree-Variant

Trick: last 16 bits of 64bit-pointer are not used

→ store bool in pointer address

→ allows atomically setting pointer and marked flag

```
template<class T>
class MarkableReference
{
private:
    uintptr_t val;
    static const uintptr_t mask = 1;
public:
    MarkableReference(T* ref = NULL, bool mark = false) {
        val = ((uintptr_t)ref & ~mask) | (mark ? 1 : 0);
    }
    T* getRef() const { return (T*)(val & ~mask); }
    bool getMark() const { return (val & mask); }
    T *operator->() const { return (T*)(val & ~mask); }
};
```


Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Lockfree-Variant

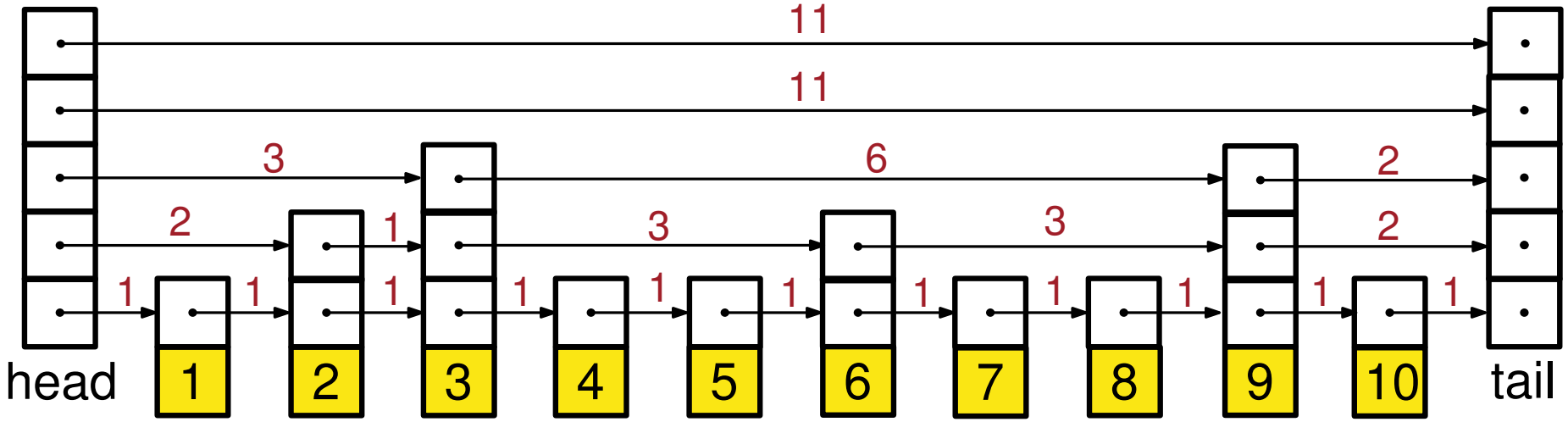
unlink in `find`:

```
// pred -> cur -> succ
while(succ.getMark()) {
    MarkPtr expect = {cur.getRef(), false};
    snip = (pred -> next[i].compare_exchange_strong(expect, {succ.getRef(), false}));
    if(!snip) {goto retry;} //restart find
    cur = pred -> next[i];
    succ = cur -> next[i];
}
```


Institute of Theoretical Informatics
Algorithmics Group

Implementation Highlights: Indexable-Variant

- hard to maintain lengths, since all length above change
- supoptimal solution: recompute lengths before `rank` and `index` operation



Garbage Collection

- shared_ptr allows dynamic deallocation

  $\rightarrow$ 10 times slower due to counting references

Garbage Collection

- shared_ptr allows dynamic deallocation

  $\rightarrow$ 10 times slower due to counting references

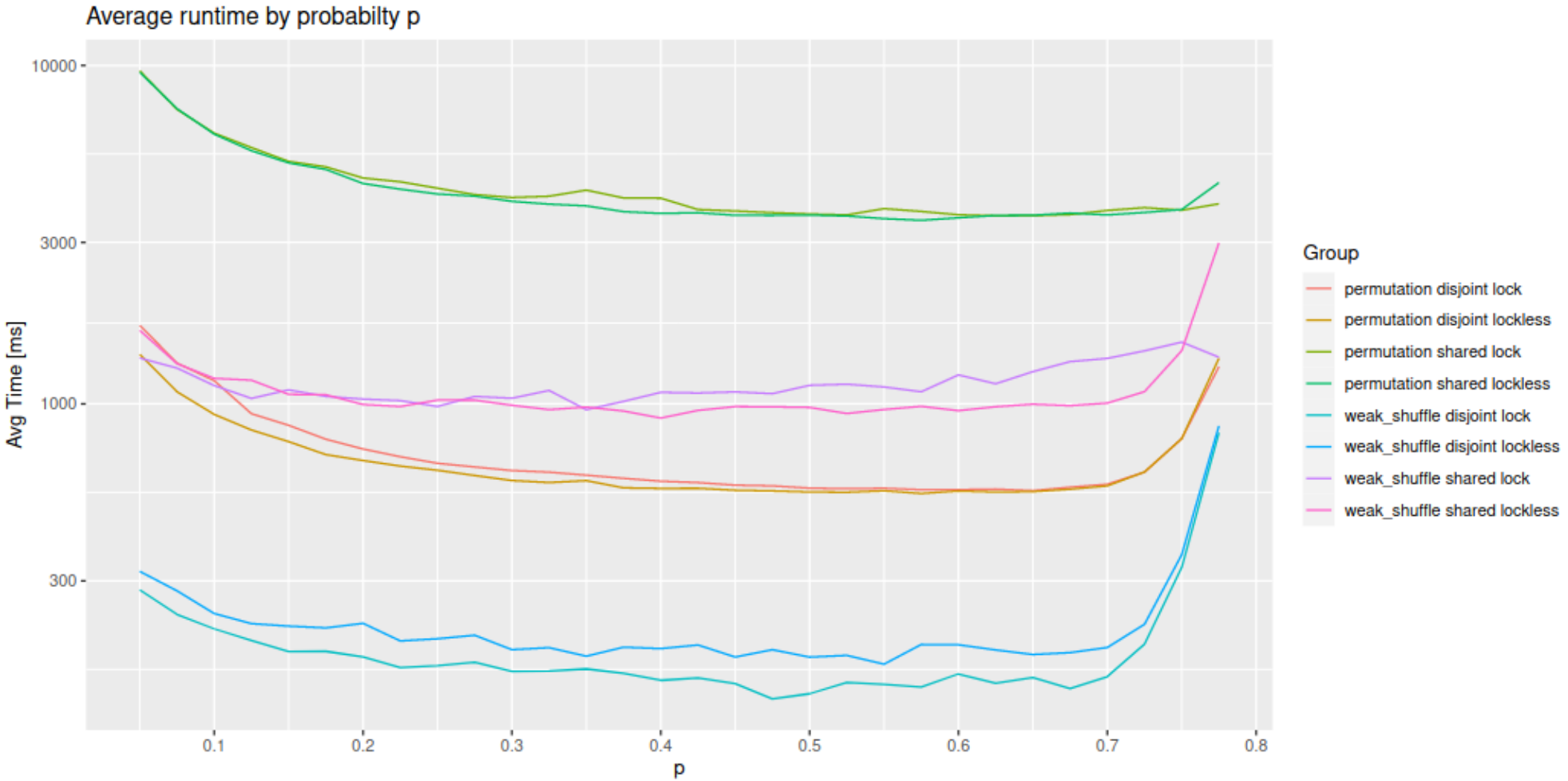
- simple solution: store removed nodes in garbage queues and clean up at the end

  $\rightarrow$ memory will go out for long time usage

  $\rightarrow$ ok for short time usage

Evaluation

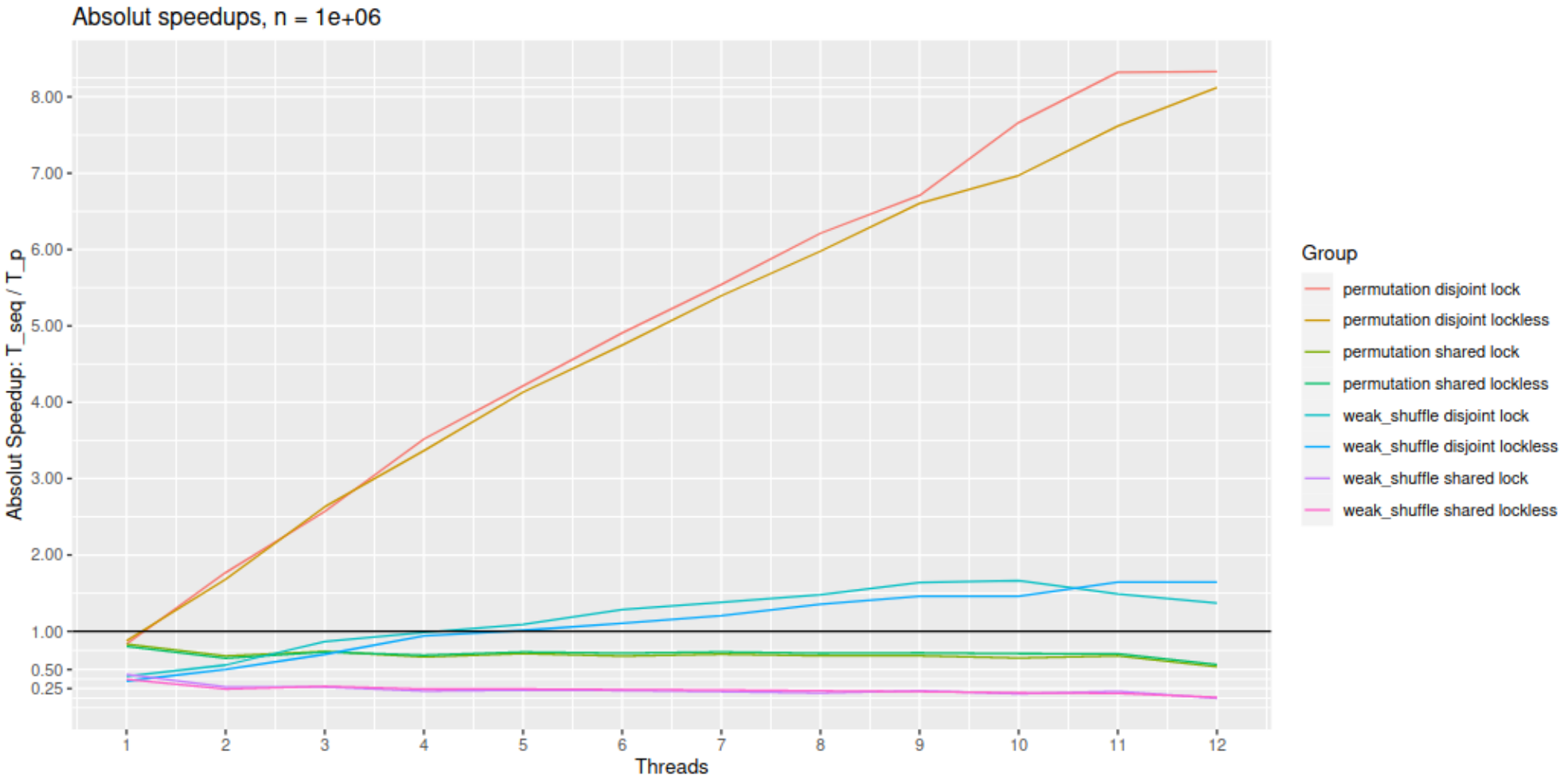
Benchmark on 6 core laptop: 5 runs, 20% insert, 20% remove, 60% search, Key = int



Institute of Theoretical Informatics
Algorithmics Group

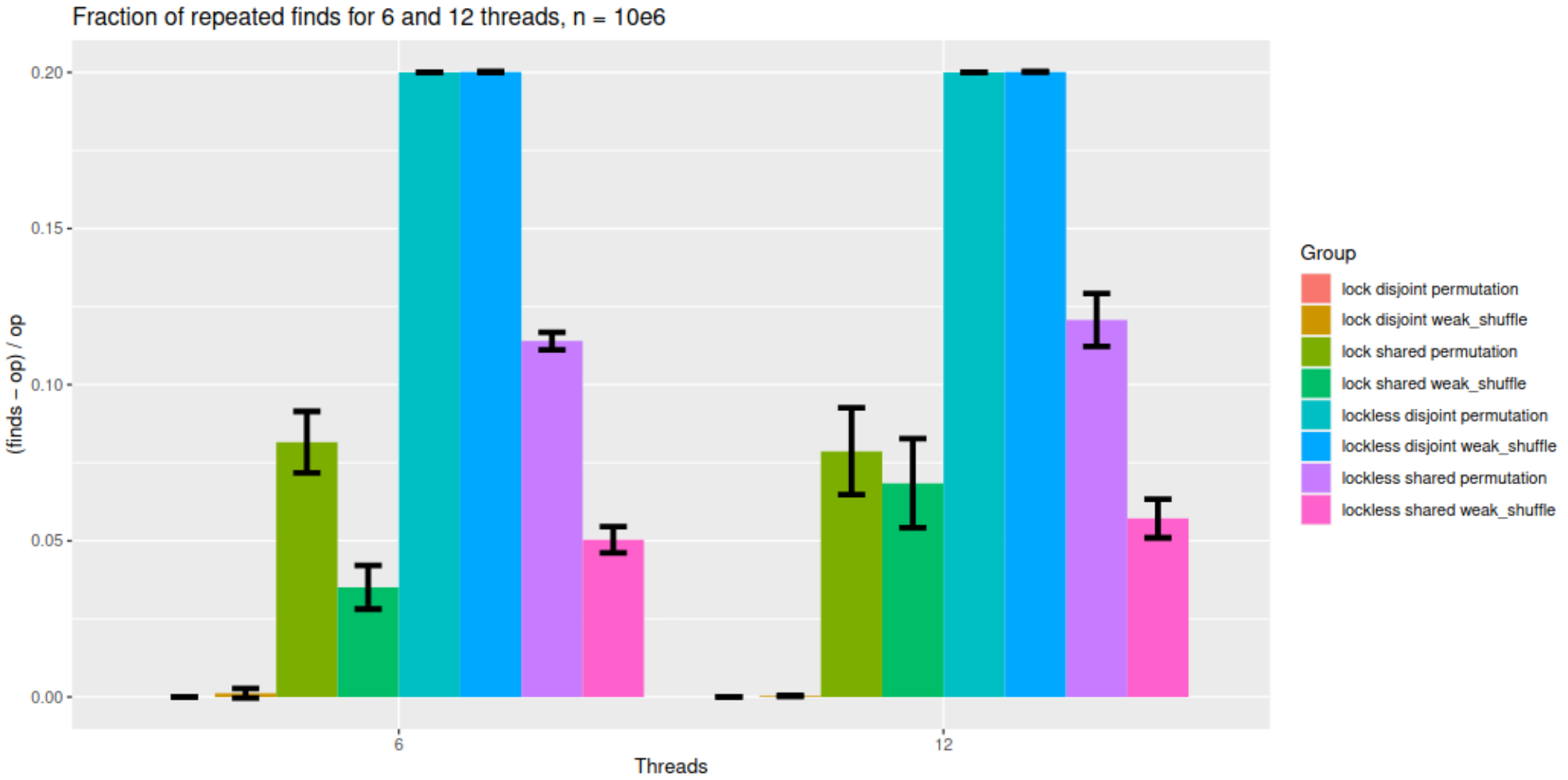
Evaluation

Benchmark on 6 core laptop: 5 runs, 20% insert, 20% remove, 60% search, Key = int



Evaluation

Benchmark on 6 core laptop: 5 runs, 20% insert, 20% remove, 60% search, Key = int



References

[1] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. ACM Transactions on Database Systems, 33(6):668–676, 1990.

[2] Y. Lev, M. Herlihy, V. Luchangco, and N. Shavit. A Simple Optimistic Skiplist Algorithm. Fourteenth Colloquium on structural information and communication complexity (SIROCCO) 2007 pp. 124–138, June 5–8, 2007, Castiglioncello (LI), Italy.

[3] Herlihy, Y. Lev, and N. Shavit. A lock-free concurrent skiplist with wait-free search. Unpublished Manuscript, Sun Microsystems Laboratories, Burlington, Massachusetts, 2007.

[4] The Art of Multiprocessor Programming, Maurice Herlihy, Nir Shavit.

Summary

Lockbased

- lockfree `find` + validation
- lock levels individually
- first successfull thread inserts/removes node completly

Lockfree

- markable pointer
- delay remove to `find`

Indexable

- recompute lengths before `rank` and `index`

Garbage Collection

- shared_ptr slow, thus dealloaction of all deleted nodes at the end