Concepts and Models of Parallel and Data-centric Programming

Shared Memory VIII

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

Outline

Granularity / five patterns of synchronization

Motivation

- We learned: we should not write our own synchronization constructs
- We learned: locks can be expensive for use with many threads

- BUT: adding threads should not lower throughput
  - Lock everything to be sure things are correct?
  - Lock (almost) nothing to get good scalability?
- Goal: examine five patterns
  - Bag of tricks …
  - Methods that work more than once …
- Illustrate these patterns by implementing a list-based Set
  - Common application and/or building block for other apps

Set Interface

- Unordered collection of items
  - No duplicates
- Methods
  - **add(x)** put **x** in set
  - **remove(x)** take **x** out of set
  - **contains(x)** tests if **x** in set

```
template<typename T> class Set {
public:
  bool add(T x);
  bool remove(T x);
  bool contains(T x);
};
```

List-based Set

- List node:

```
template<typename T> class Node {
public:
    T item;
    int key;
    Node* next;
};
```

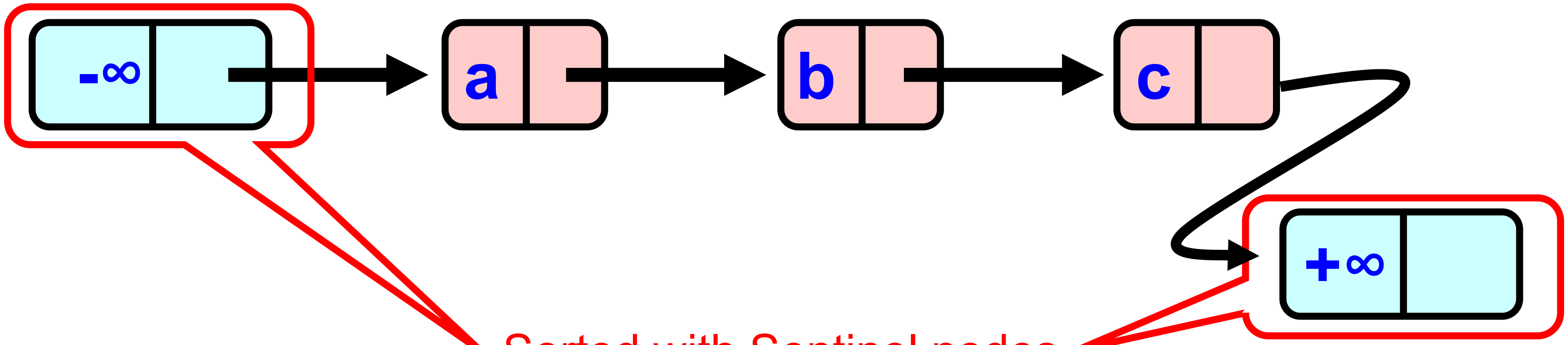
Item of interest
Hash
Reference to next node

List-based Set

- List node:

```
template<typename T> class Node {
public:
    T item;
    int key;
    Node* next;
};
```

Item of interest
Hash
Reference to next node



Sorted with Sentinel nodes
(min & max possible keys)

Concurrent Objects

- Invariant: (an object's) property that always holds

Concurrent Objects

- Invariant: (an object’s) property that always holds

- Simple formalization for the Set:
  - S(head) = { x | there exists a such that
    - a reachable from head and
    - a.item = x
      }

Concurrent Objects

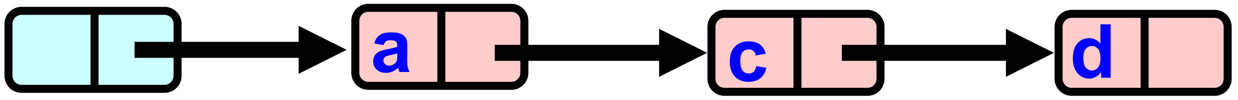
- Invariant: (an object's) property that always holds

- Simple formalization for the Set:
  - S(head) = { x | there exists a such that
    - a reachable from head and
    - a.item = x
    }

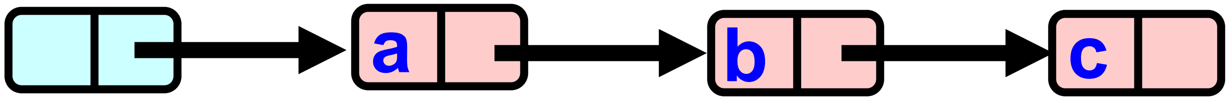
- Established because
  - True when object is **created**
  - Truth **preserved** by each method
  - Methods considered are the only modifiers (encapsulation)

Sequential List-based Set / 1

Add()

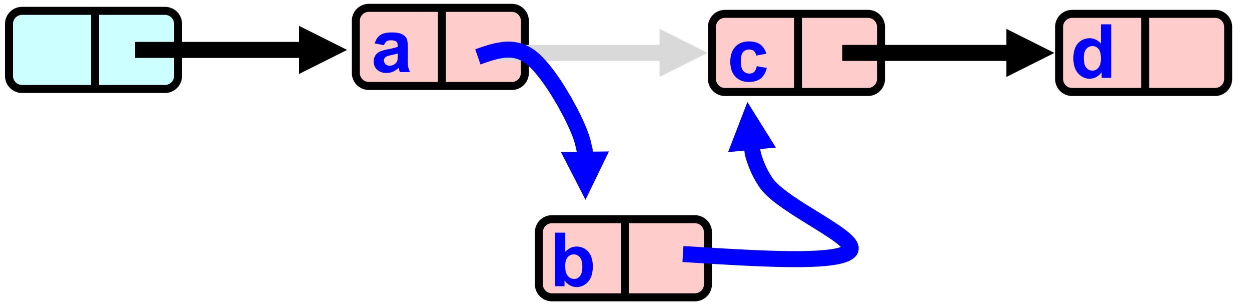


Remove()



Sequential List-based Set / 2

Add()



Remove()