NOVA SCHOOL OF SCIENCE & TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

Alternative Synchronization Strategies — Lock-Free Algorithms (2) —

lecture 20 (2021-05-24)

Master in Computer Science and Engineering

— Concurrency and Parallelism / 2020-21 —

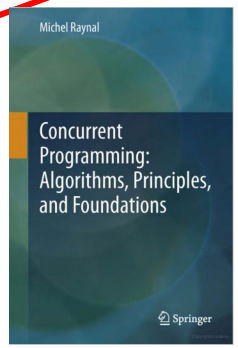
João Lourenço <joao.lourenco@fct.unl.pt>

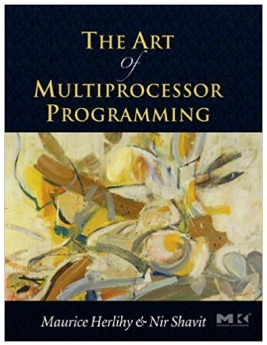
Alternative Synchronization Strategies

- Contents:
  - Liveness: Types of Progress
  - Coarse-Grained Synchronization
  - Fine-Grained Synchronization
  - Optimistic Synchronization
  - Lazy Synchronization
  - Lock-Free Synchronization

Past lectures

Today





- Reading list:
  - chapter 5 of the Textbook
  - Chapter 9 of "The Art of Multiprocessor Programming" by Maurice Herlihy & Nir Shavit *(available at clip)*

The Window Class

```
class Window {
 public Node pred;
 public Node curr;
 Window(Node pred, Node curr) {
   this.pred = pred; this.curr = curr;
 }
}
```

The Window Class

```
class Window {
 public Node pred;
 public Node curr;
 Window(Node pred, Node curr) {
   this.pred = pred; this.curr = curr;
 }
}
```

A container for pred and current values

Using the Find Method

```
Window window = find(head, key);
Node pred = window.pred;
Node curr = window.curr;
```

Using the Find Method

```
Window window = find(head, key);
Node pred = window.pred;
Node curr = window.curr;
```

Find returns window

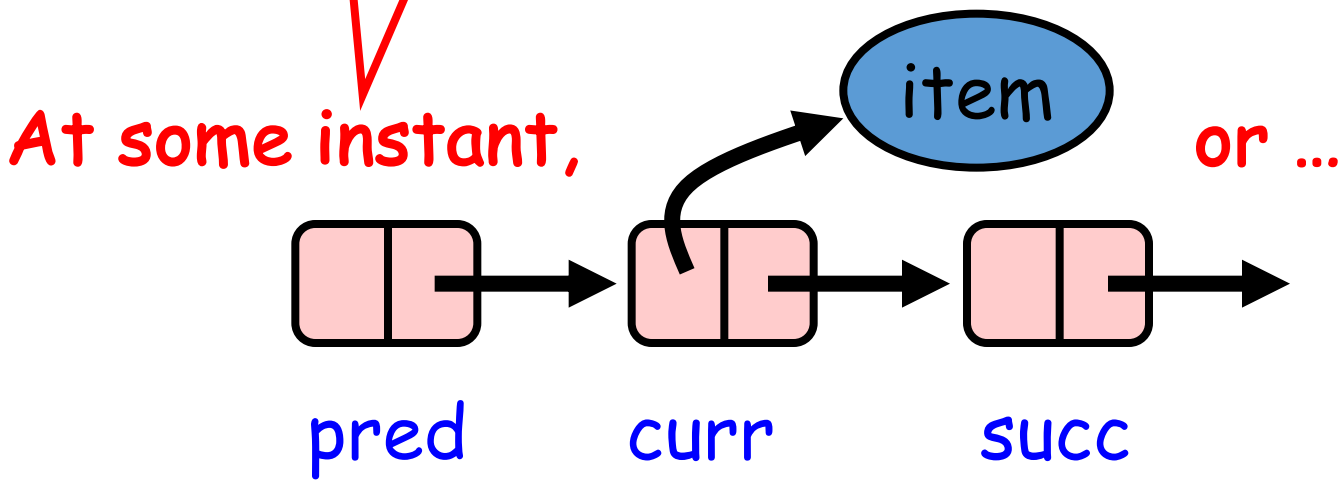
Using the Find Method

```
Window window = find(head, key);
Node pred = window.pred;
Node curr = window.curr;
```

Extract pred and curr

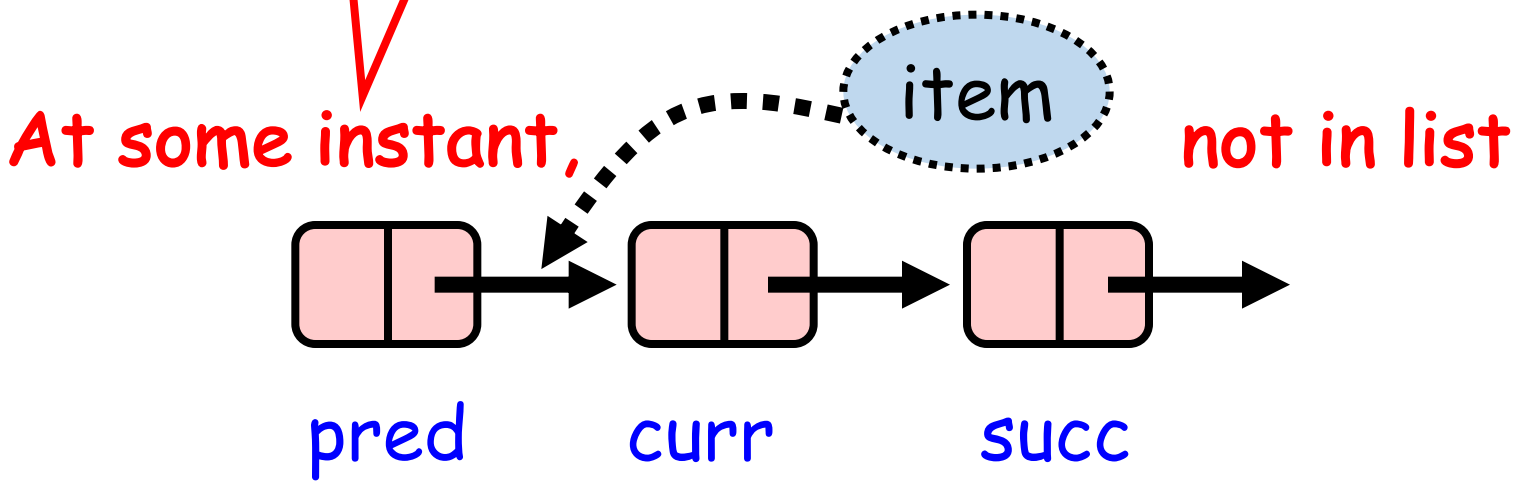
The Find Method

```
Window window = find(head, key);
```

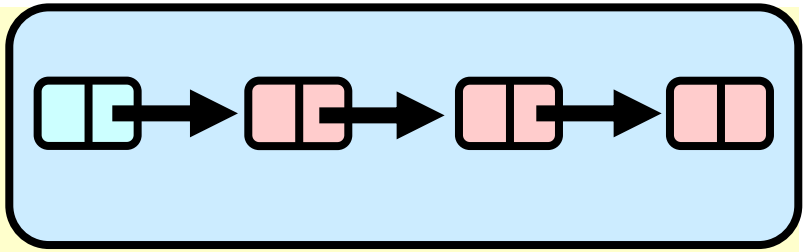


The Find Method

```
Window window = find(head, key);
```



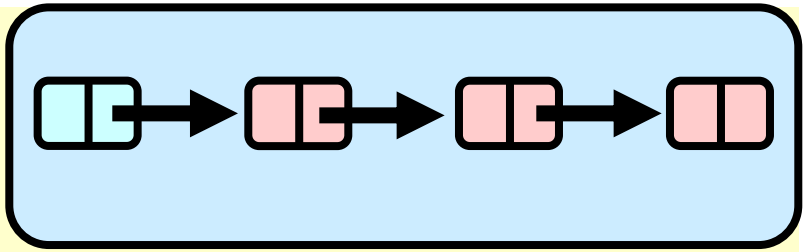
Remove (lock-free)



```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
     Node succ = curr.next.getReference();
     snip = curr.next.attemptMark(succ, true);
     if (!snip) continue;
     pred.next.compareAndSet(curr, succ, false, false);
     return true;
}}}
```

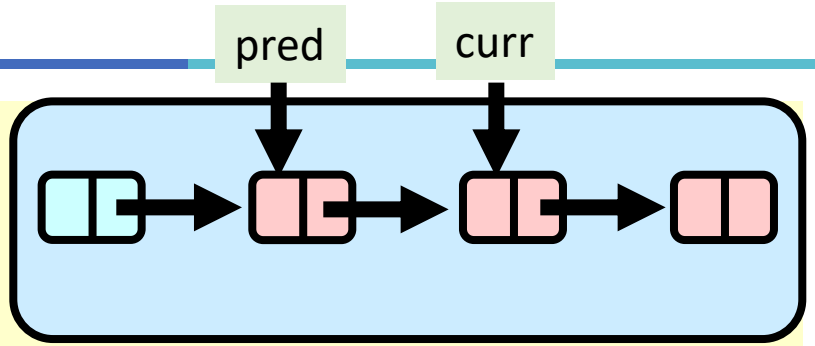
Remove (lock-free)

```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
     Node succ = curr.next.getReference();
     snip = curr.next.attemptMark(succ, true);
     if (!snip) continue;
     pred.next.compareAndSet(curr, succ, false, false);
     return true;
}}}
```



Keep trying

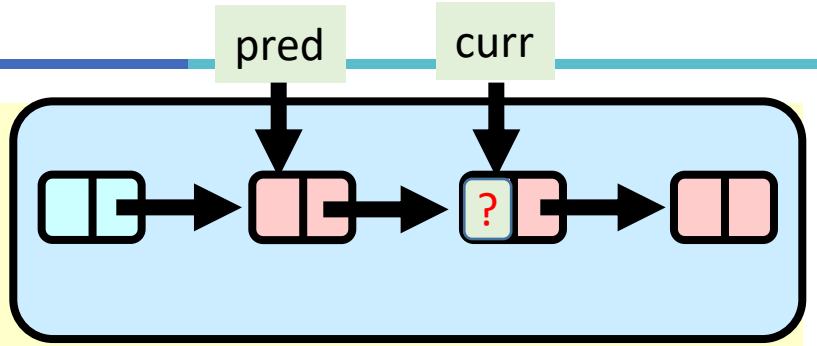
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
     Node succ = curr.next.getReference();
     snip = curr.next.attemptMark(succ, true);
     if (!snip) continue;
     pred.next.compareAndSet(curr, succ, false, false);
     return true;
}}}
```

Find neighbors

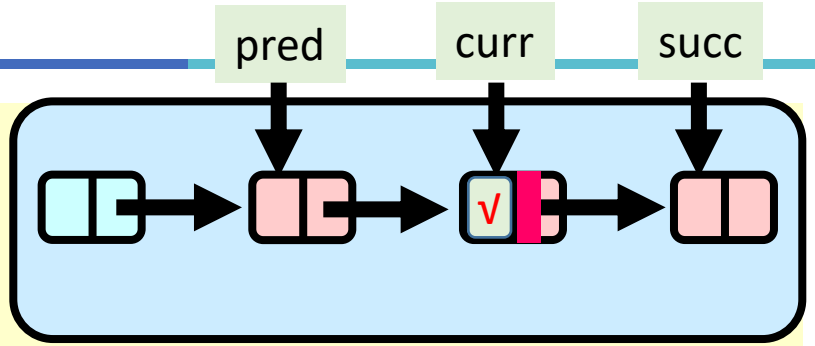
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
     Node succ = curr.next.getReference();
     snip = curr.next.attemptMark(succ, true);
     if (!snip) continue;
     pred.next.compareAndSet(curr, succ, false, false);
     return true;
}}}
```

It is not there …

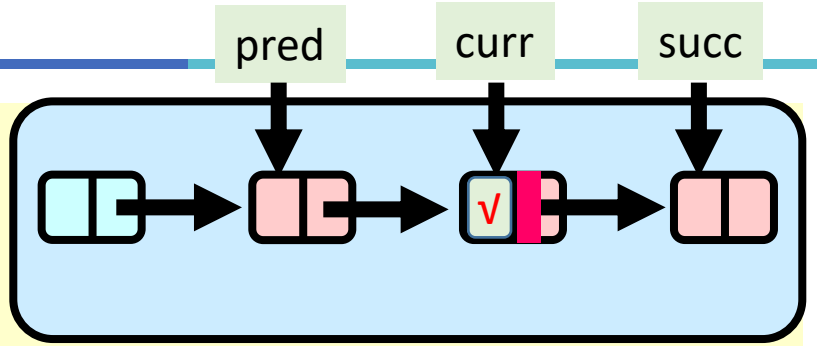
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
     Node succ = curr.next.getReference();
     snip = curr.next.attemptMark(succ, true);
     if (!snip) continue;
     pred.next.compareAndSet(curr, succ, false, false);
     return true;
}}}
```

Try to mark node as deleted

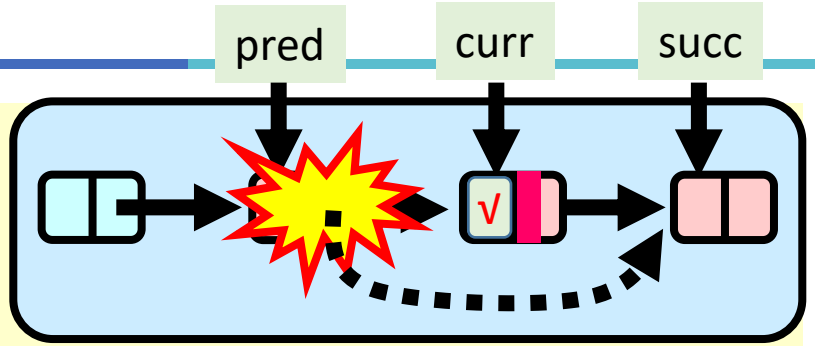
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
  Window window = find(head, key);
  Node pred = window.pred, curr = window.curr;
  if (curr.key == key) {
    return false;
  } else {
    Node succ = curr.next.getReference();
    snip = curr.next.attemptMark(succ, true);
    if (!snip) continue;
    pred.next.compareAndSet(curr, succ, false, false);
    return true;
}}}
```

If it doesn't work, just retry, if it does, job essentially done

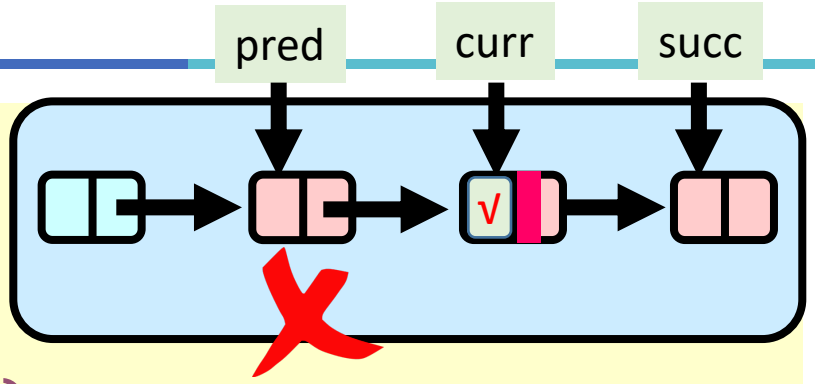
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
    return false;
  } else {
    Node succ = curr.next.getReference();
    snip = curr.next.attemptMark(succ, true);
    if (!snip) continue;
    pred.next.compareAndSet(curr, succ, false, false);
    return true;
}}}
```

Try to advance reference
(if we don't succeed, someone else did or will do).

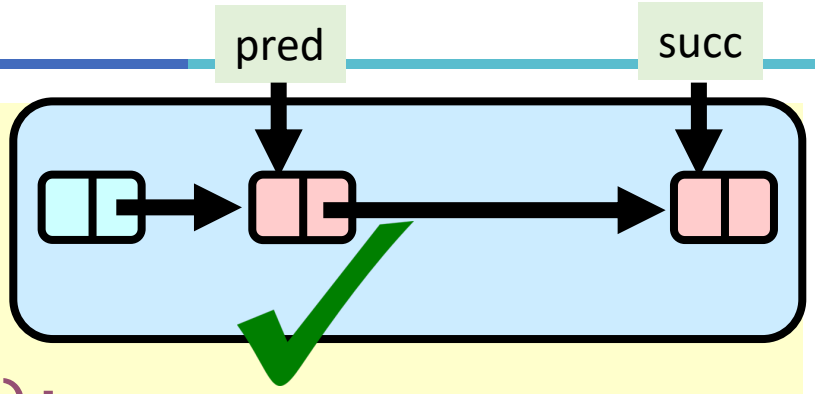
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
    return false;
  } else {
    Node succ = curr.next.getReference();
    snip = curr.next.attemptMark(succ, true);
    if (!snip) continue;
    pred.next.compareAndSet(curr, succ, false, false);
    return true;
}}}
```

Try to advance reference
(if we don't succeed, someone else did or will do).

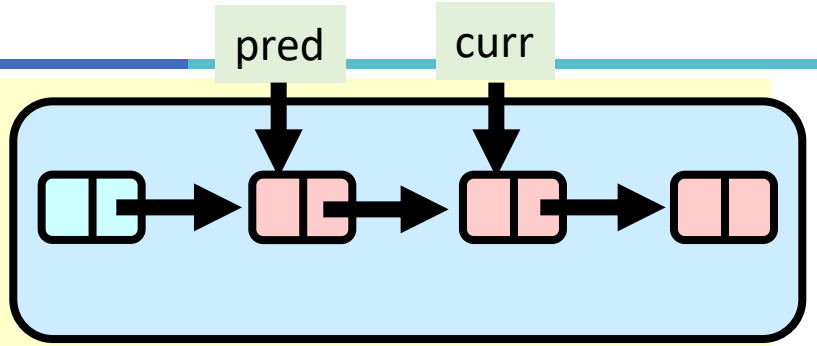
Remove (lock-free)




```
public boolean remove(T item) {
Boolean snip;
int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
    return false;
  } else {
    Node succ = curr.next.getReference();
    snip = curr.next.attemptMark(succ, true);
    if (!snip) continue;
    pred.next.compareAndSet(curr, succ, false, false);
    return true;
}}}
```

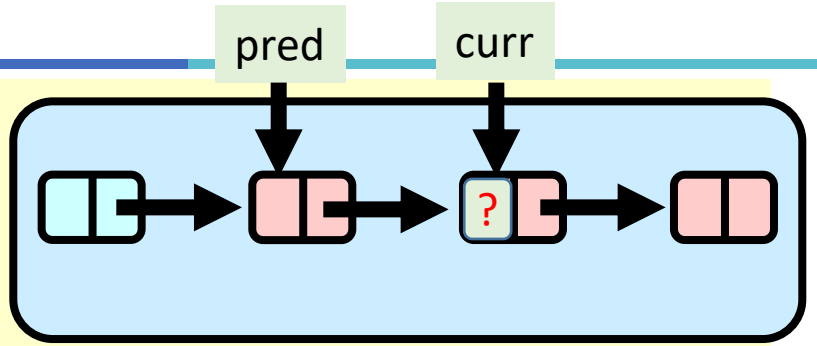
Try to advance reference
(if we don't succeed, someone else did or will do).

Add (lock-free)




```
public boolean add(T item) {
 boolean snip;
 int key = item.hashCode();
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred, curr = window.curr;
   if (curr.key == key) {
      return false;
   } else {
      Node node = new Node(item);
      node.next = new AtomicMarkableRef(curr, false);
      if (pred.next.compareAndSet(curr, node,
                          false, false)) {return true;}
}}}
```

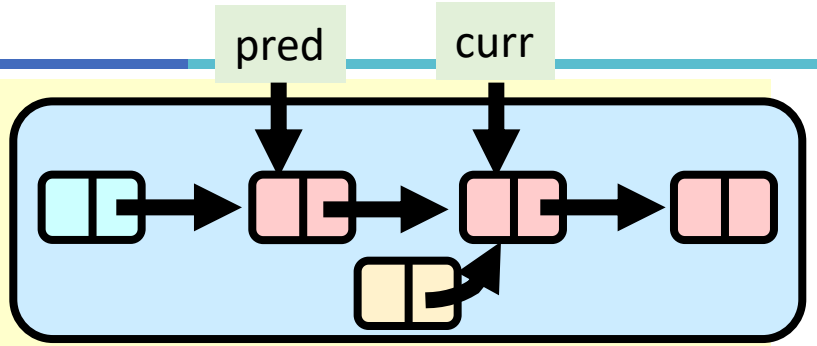
Add (lock-free)




```
public boolean add(T item) {
 boolean snip;
 int key = item.hashCode();
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred, curr = window.curr;
   if (curr.key == key) {
      return false;
   } else {
      Node node = new Node(item);
      node.next = new AtomicMarkableRef(curr, false);
      if (pred.next.compareAndSet(curr, node,
                                  false, false)) {return true;}
}}}
```

Item already there.

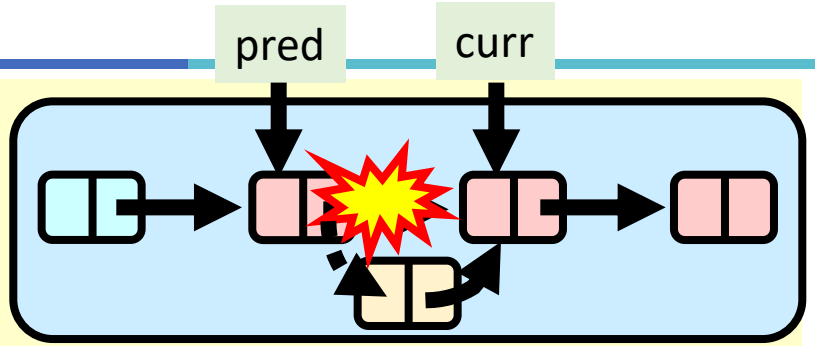
Add (lock-free)




```
public boolean add(T item) {
 boolean snip;
 int key = item.hashCode();
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred, curr = window.curr;
   if (curr.key == key) {
      return false;
   } else {
      Node node = new Node(item);
      node.next = new AtomicMarkableRef(curr, false);
      if (pred.next.compareAndSet(curr, node,
                                  false, false)) {return true;}
}}}
```

create new node

Add (lock-free)




```
public boolean add(T item) {
 boolean snip;
 int key = item.hashCode();
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred, curr = window.curr;
   if (curr.key == key) {
     return false;
   } else {
     Node node = new Node(item);
     node.next = new AtomicMarkableRef(curr, false);
     if (pred.next.compareAndSet(curr, node,
                                 false, false)) {return true;}
}}}
```

Install new node,
else retry loop

Contains (wait-free)

```
public boolean contains(T item) {
    boolean[] marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
}
```

Contains (wait-free)

```
public boolean contains(T item) {
    boolean[] marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
  }
```

Only diff is that we get and check marked

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
     curr = succ;
   }
}}
```

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
    pred = head;
    curr = pred.next.getReference();
    while (true) {
      succ = curr.next.get(marked);
      while (marked[0]) {
        …
      }
      if (curr.key >= key)
            return new Window(pred, curr);
      pred = curr;
      curr = succ;
    }
}}
```

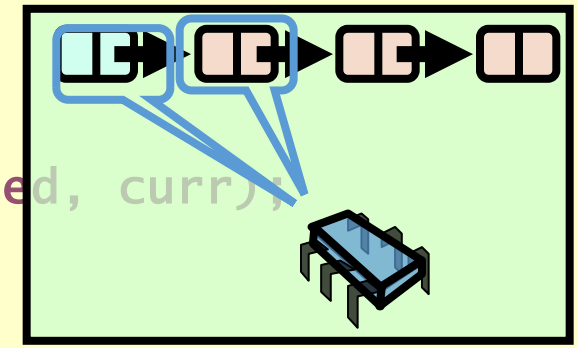
If list changes while traversed, start over

Lock-Free because we start over only if someone else makes progress

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
     curr = succ;
   }
}}
```

Start looking from head



Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
     curr = succ;
   }
}}
```

Move down the list

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
     curr = succ;
   }
}}
```

Get ref to successor and current deleted bit

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
```

Try to remove deleted nodes in path… code details soon

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
     curr = succ;
   }
}}
```

If curr key is greater or equal, return pred and curr

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
         return new Window(pred, curr);
     pred = curr;
     curr = succ;
   }
}}
```

Otherwise advance window and loop again

Find (lock-free)

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
       …
     }
     if (curr.key >= key)
           return new Window(pred, curr);
     pred = curr;
```

Try to remove deleted nodes in path… Let's see how it works!

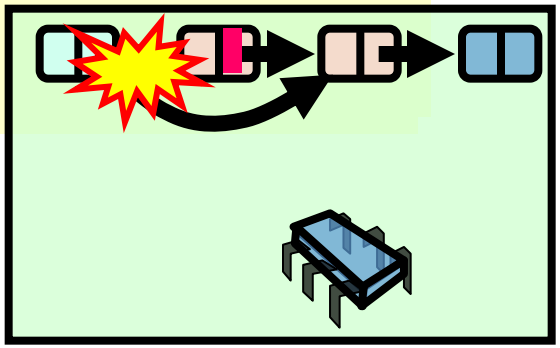
Find (lock-free)

```
retry: while (true) {
    …
    while (marked[0]) {
      snip = pred.next.compareAndSet(curr,
                                succ, false, false);
      if (!snip) continue retry;
      curr = succ;
      succ = curr.next.get(marked);
    }
    …
```

Find (lock-free)

Try to snip out node

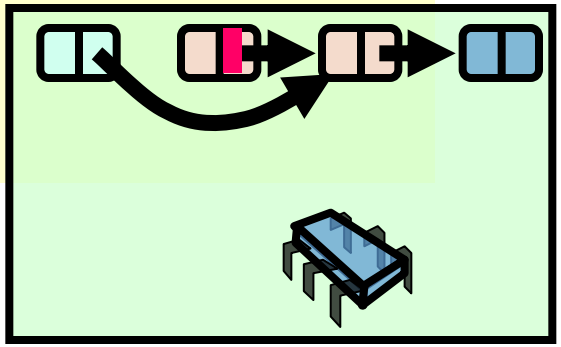
```
retry: while (true) {
    …
    while (marked[0]) {
      snip = pred.next.compareAndSet(curr,
                                     succ, false, false);
      if (!snip) continue retry;
      curr = succ;
      succ = curr.next.get(marked);
    }
    …
```



Find (lock-free)

if predecessor's next field changed must retry whole

```
retry: while (true) {
    …
    while (marked[0]) {
      snip = pred.next.compareAndSet(curr,
                                     succ, false, false);
      if (!snip) continue retry;
      curr = succ;
      succ = curr.next.get(marked);
    }
    …
```



Find (lock-free)

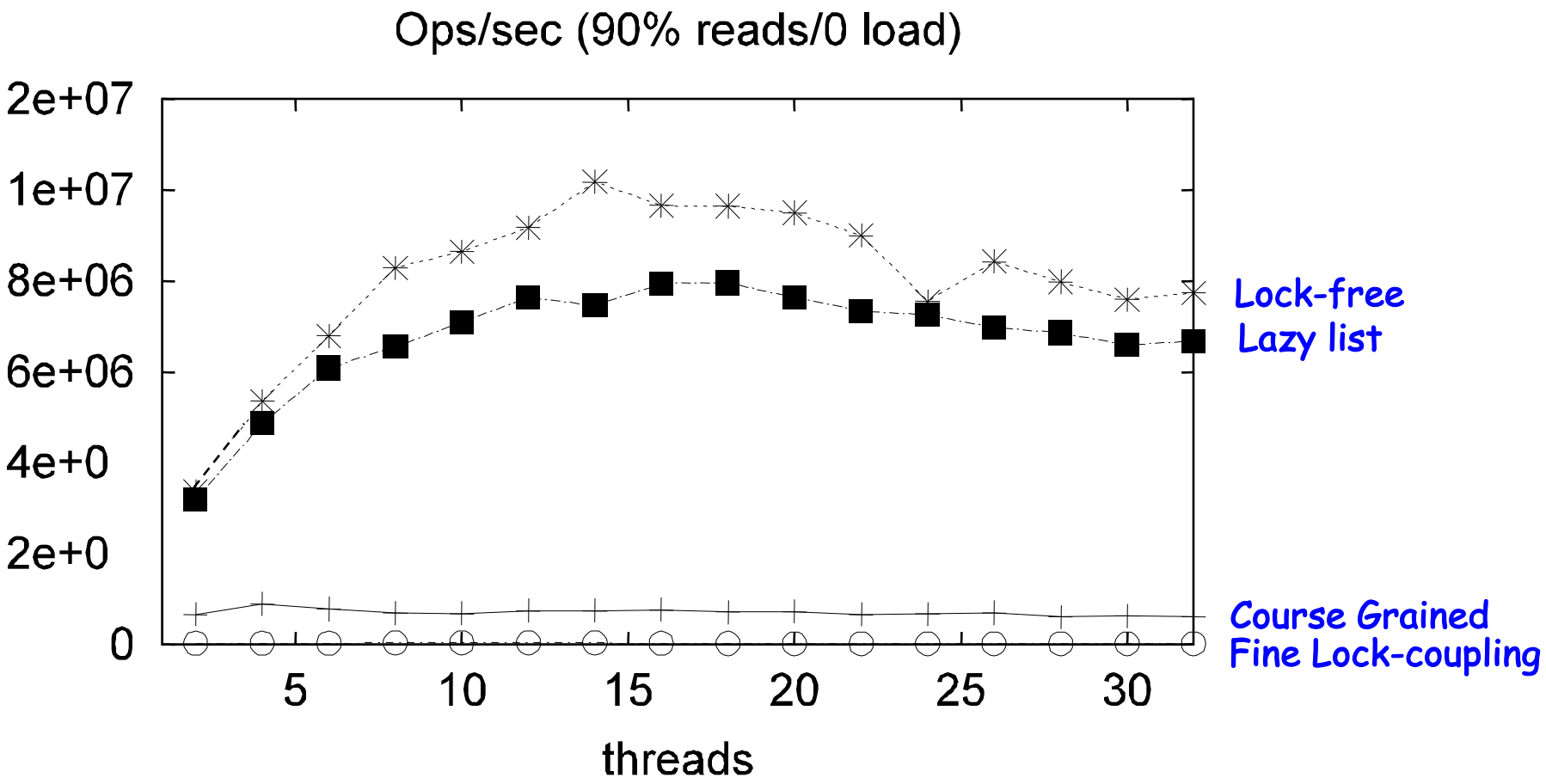
Otherwise move on to check if next node deleted

```
retry: while (true) {
    …
    while (marked[0]) {
      snip = pred.next.compareAndSet(curr,
                                     succ, false, false);
      if (!snip) continue retry;
      curr = succ;
      succ = curr.next.get(marked);
    }
    …
```

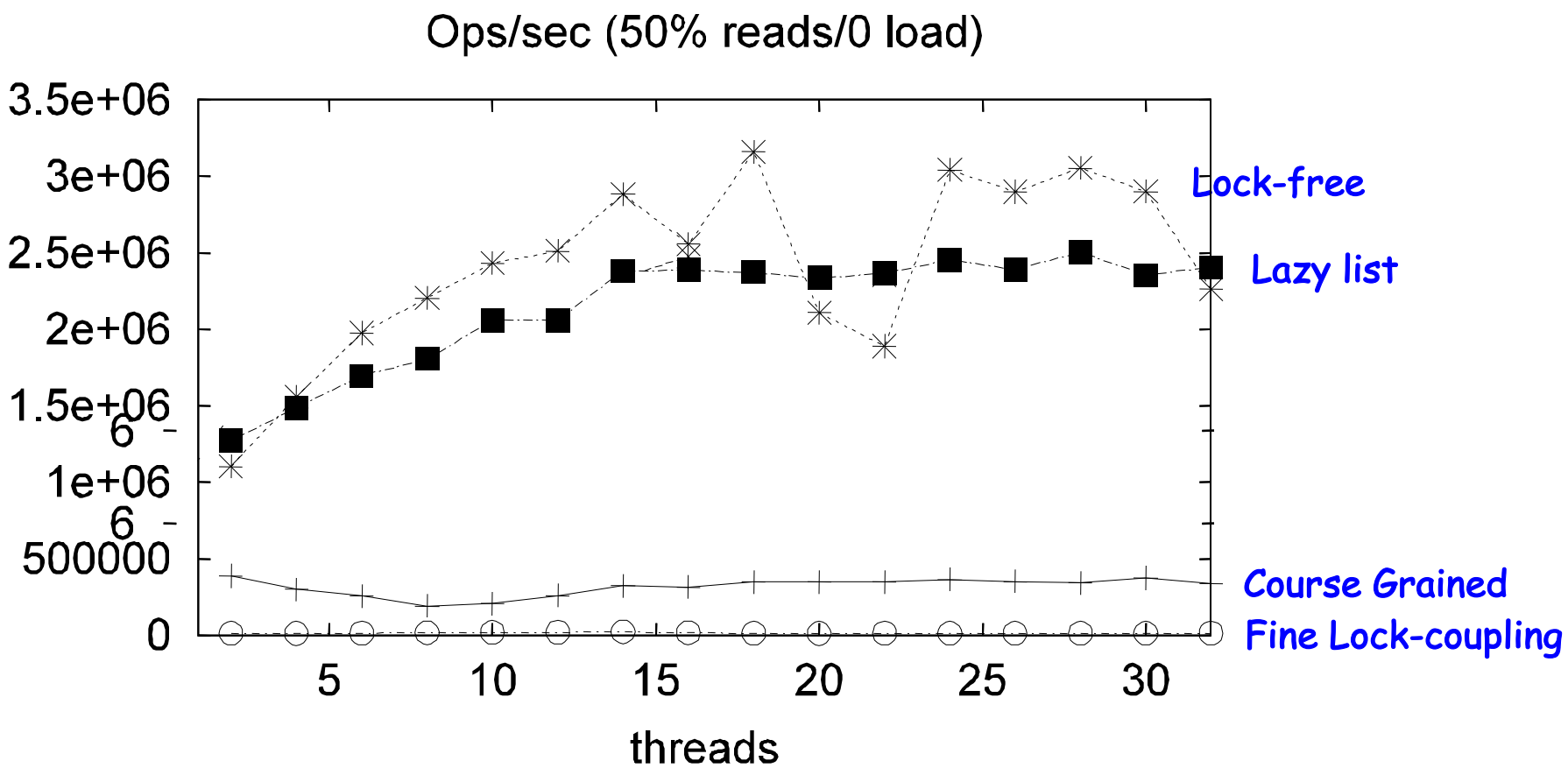
Performance

- On 16 node shared memory machine
- Benchmark throughput of Java List-based Set
- Algs. vary % of *Contains()* method Calls

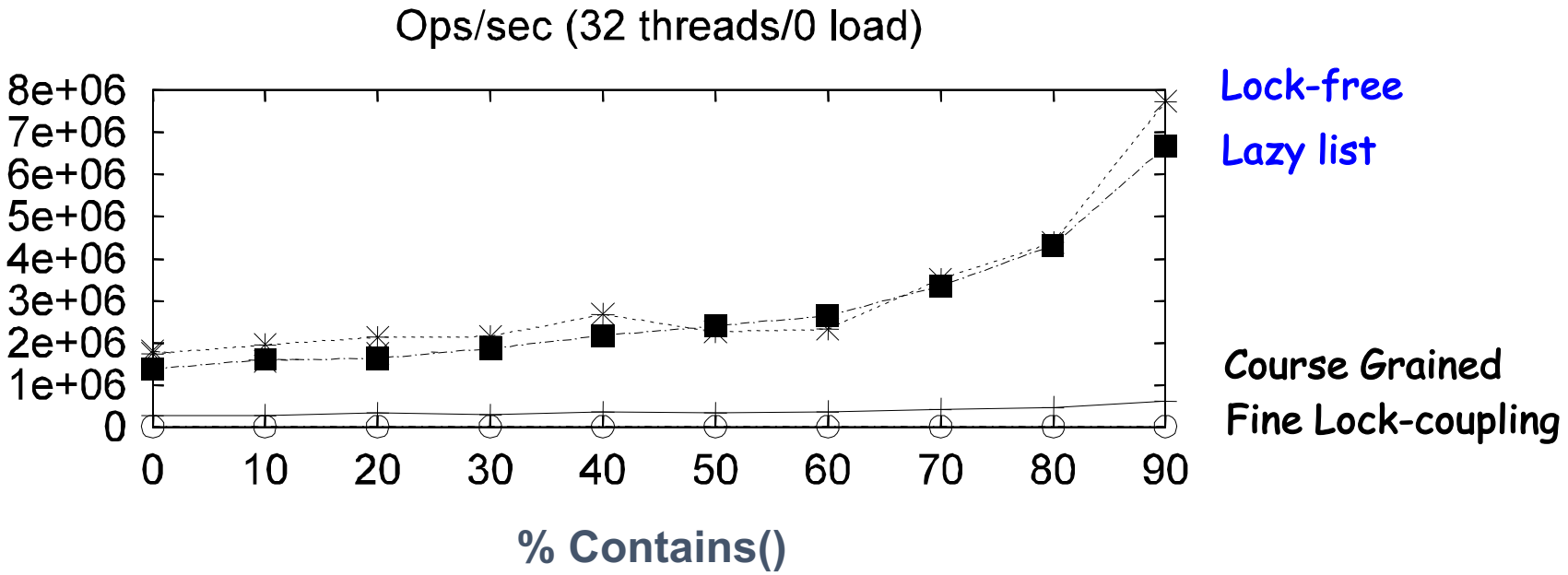
High Contains Ratio



Low Contains Ratio



As Contains Ratio Increases



Summary

- Contents:
  - Coarse-Grained Synchronization
  - Fine-Grained Synchronization
  - Optimistic Synchronization
  - Lazy Synchronization
  - Lock-Free Synchronization

- Reading list:
  - chapter 5 of the Textbook
  - Chapter 9 of "The Art of Multiprocessor Programming" by Maurice Herlihy & Nir Shavit *(available at clip)*

“To Lock or Not to Lock”

- Locking vs. Non-blocking: Extremist views on both sides
- The answer: nobler to compromise, combine locking and non-blocking
  - Example: Lazy list combines blocking add() and remove() and a wait-free contains()
  - Remember: Blocking/non-blocking is a property of a method

The END