Lock-Free Concurrent Data Structures

Danny Hendler
Ben-Gurion University

Key synchronization alternatives

1. Lock-based synchronization

2. Nonblocking algorithms

3. Transactional memory

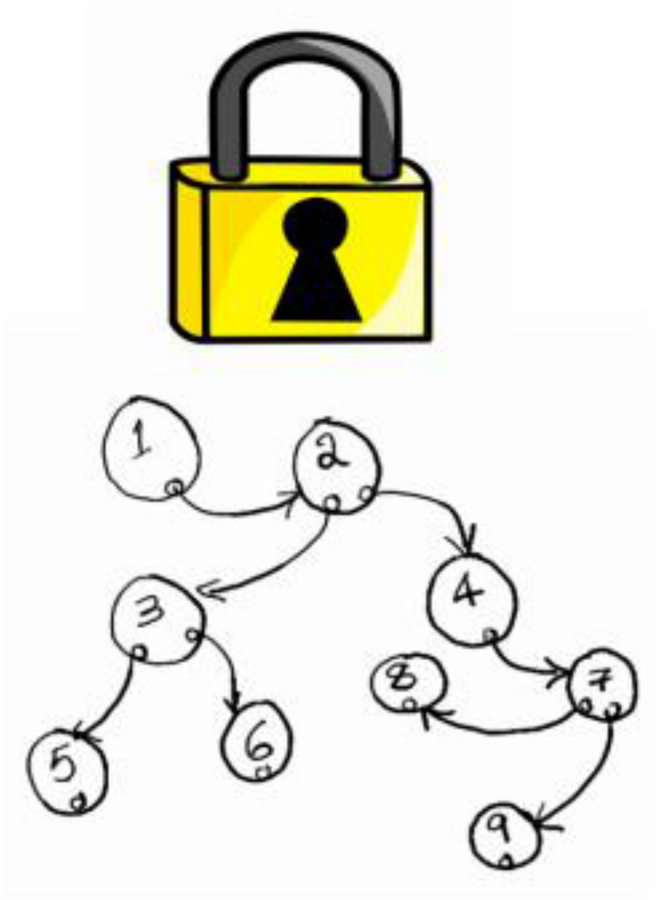
Coarse-grained locks

Pros

□ Easy to program

Cons

□ Sequential



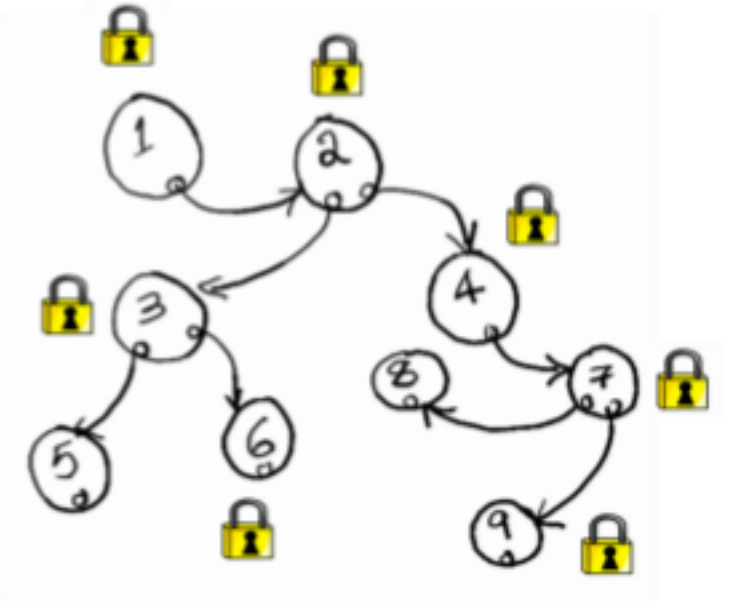
Fine-grained locks

Pros

- ❑ Potentially scalable

Cons

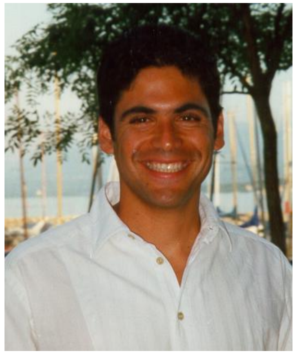
- ❑ Not robust against failures
- ❑ Susceptible to:
  - ○ Deadlocks
  - ○ Priority inversion
  - ○ Convoying
- ❑ Locks do not compose



Nonblocking synchronization

Wait-freedom

Every thread is guaranteed to complete its operation after performing a sufficient number of steps.



Lock-freedom

Some thread is guaranteed to complete its operation after a sufficient number of steps by threads is taken.

Obstruction-freedom

A thread is guaranteed to complete its operation after performing a sufficient number of steps ***when running solo***.

Lock-free algorithms

❑ Ensure global progress

❑ Avoid lock-based programming weaknesses

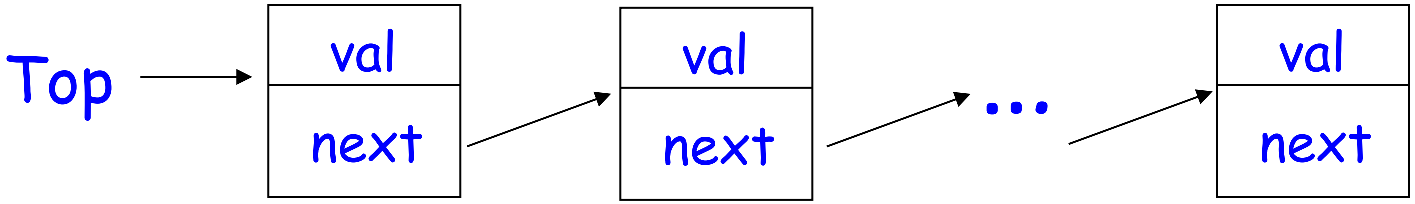
❑ Often require strong synchronization operations
  - Compare-and-swap (CAS)
  - Fetch-and-add
  - Swap
  - …

❑ Often difficult to devise and prove correct

Talk Outline

- Preliminaries
- A simple lock-free stack algorithm
  - Linearizability
- Michael & Scott queue algorithm
- The Harris-Michael linked list algorithm
- Elimination-based stack
- Discussion & conclusions

Treiber/IBM's stack algorithm

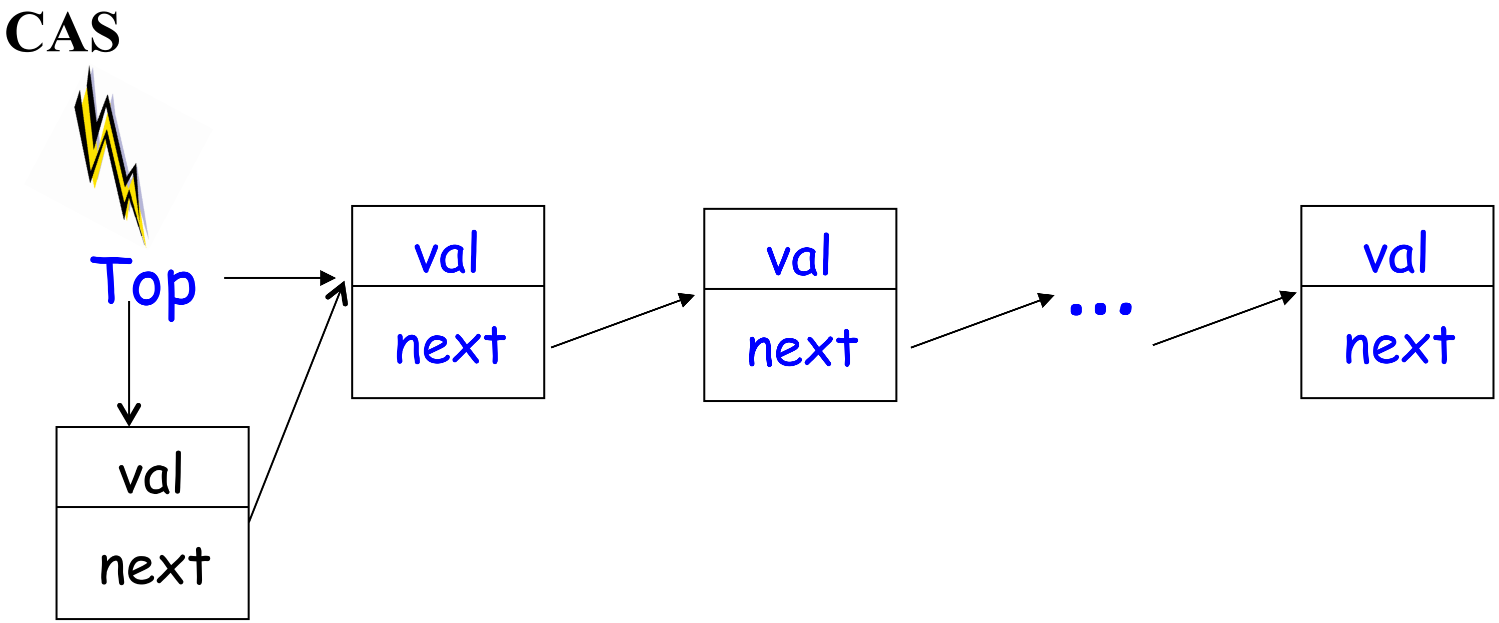


- ❑ Stack represented as linked list
- ❑ Top pointer manipulated by compare-and-swap (CAS) operations

Compare&swap(var,expected,new)

```
atomically
  t ⟵ read from var
  if (var = expected) {
    var ⟵ new
    return success
    }
  else
    return failure;
```

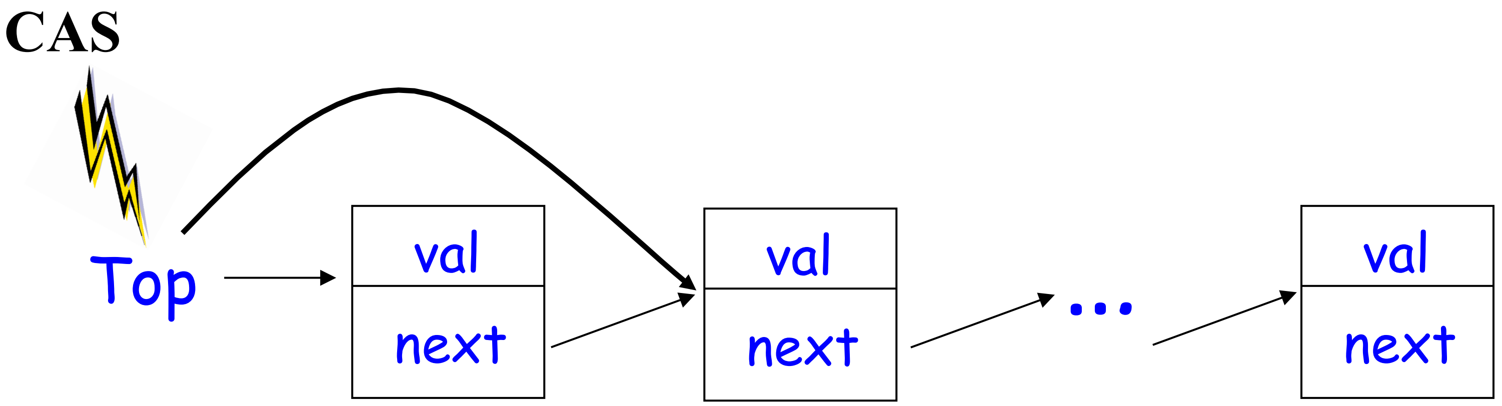
Treiber/IBM: Push



Treiber/IBM: Push

```
Push(int v, Stack S)
n := new NODE ;create node for new stack item
n.val := v          ;write item value
do forever         ;repeat until success
    node top := S.top
    n.next := top         ;next points to current top (LIFO order)
    if compare&swap(S, top, n) ; try to add new item
        return                          ; return if succeeded
end do
```

Treiber/IBM: Pop



Treiber/IBM: Pop

```
Pop(Stack S)
do forever
    top := S.top
    if top = null
        return empty
    if compare&swap(S, top, top.next)
        return-val=top.val
        free top?
        return return-val
end do
```

Why is the algorithm lock-free?

Is the algorithm “correct”?

What does it mean for a concurrent algorithm to be correct?

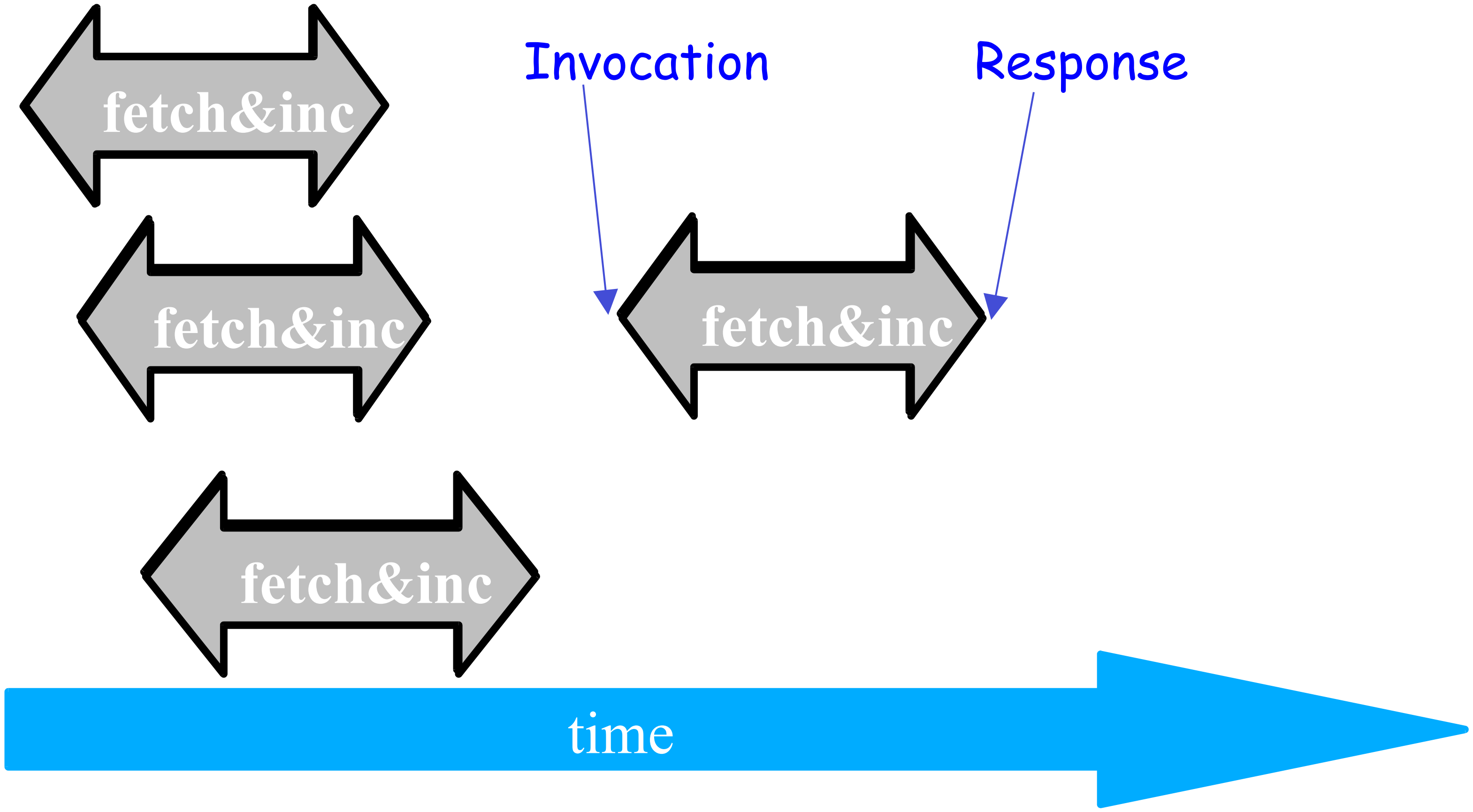
Correctness of sequential counter

- fetch&increment, applied to a counter with value v, returns v and increments the counter’s value to (v+1).

- Values returned by consecutive operations:
  0, 1, 2, …

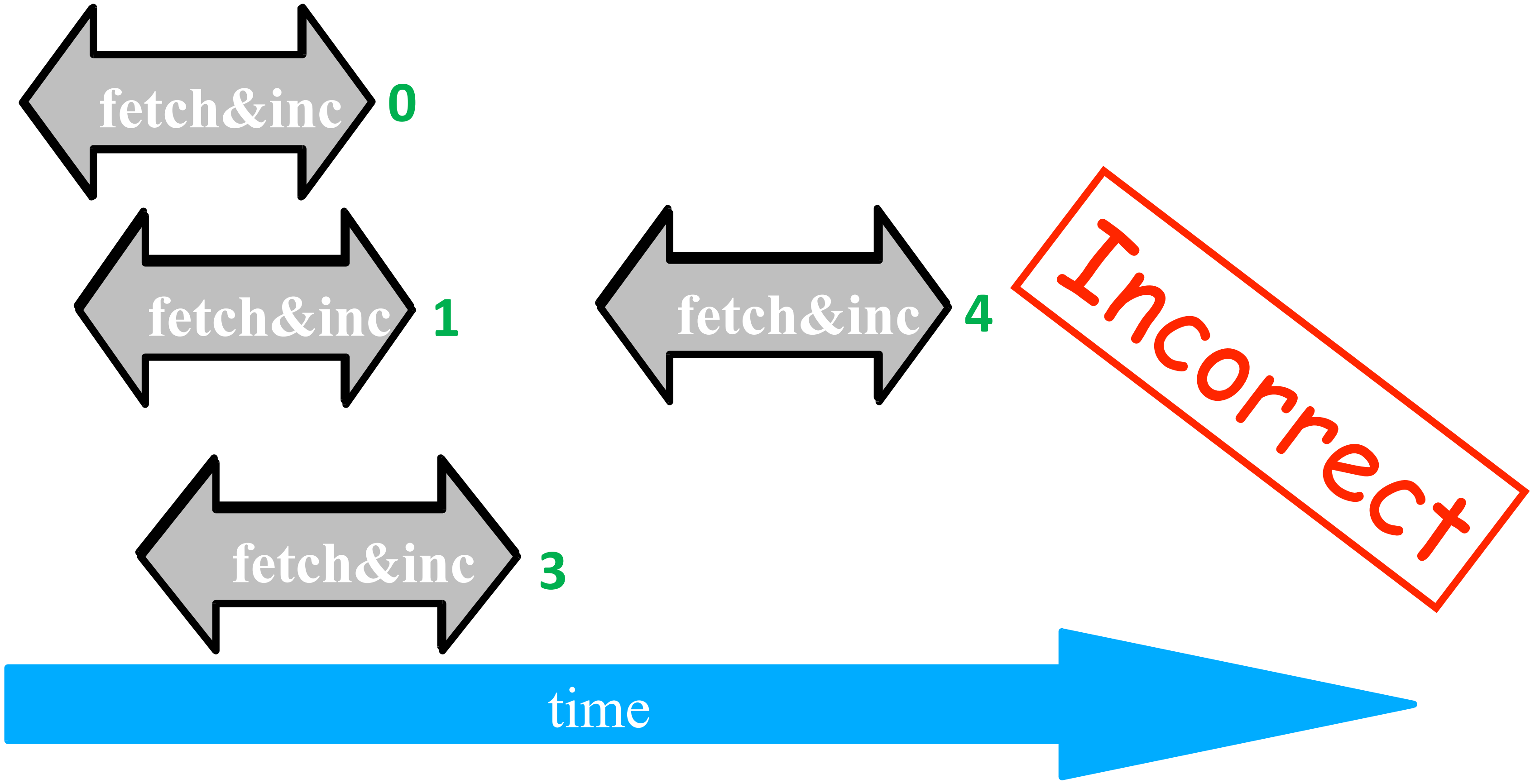
How should we define the correctness of a shared counter?

Correctness of concurrent counter?



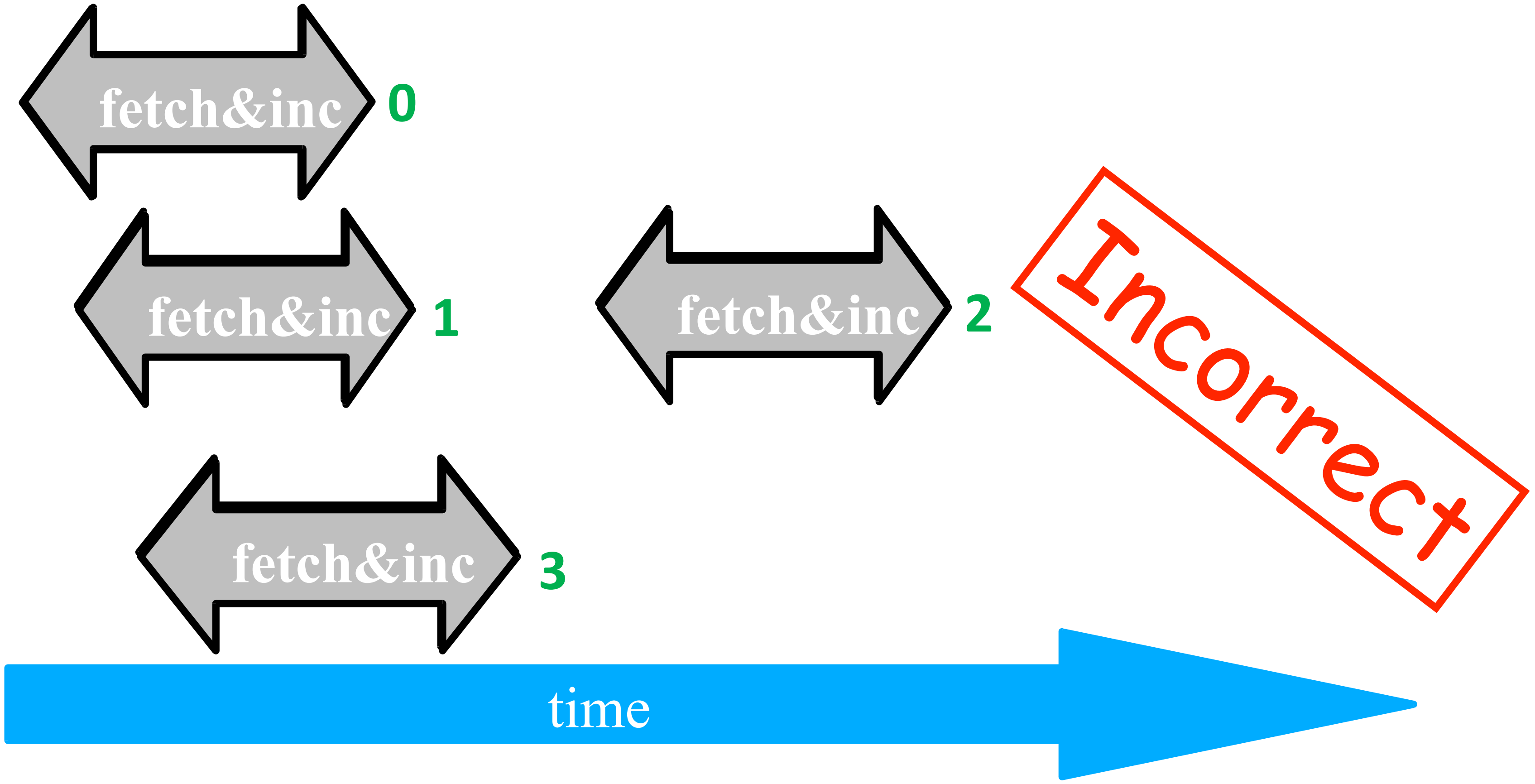
There is only a partial order between operations!

Correctness of concurrent counter?

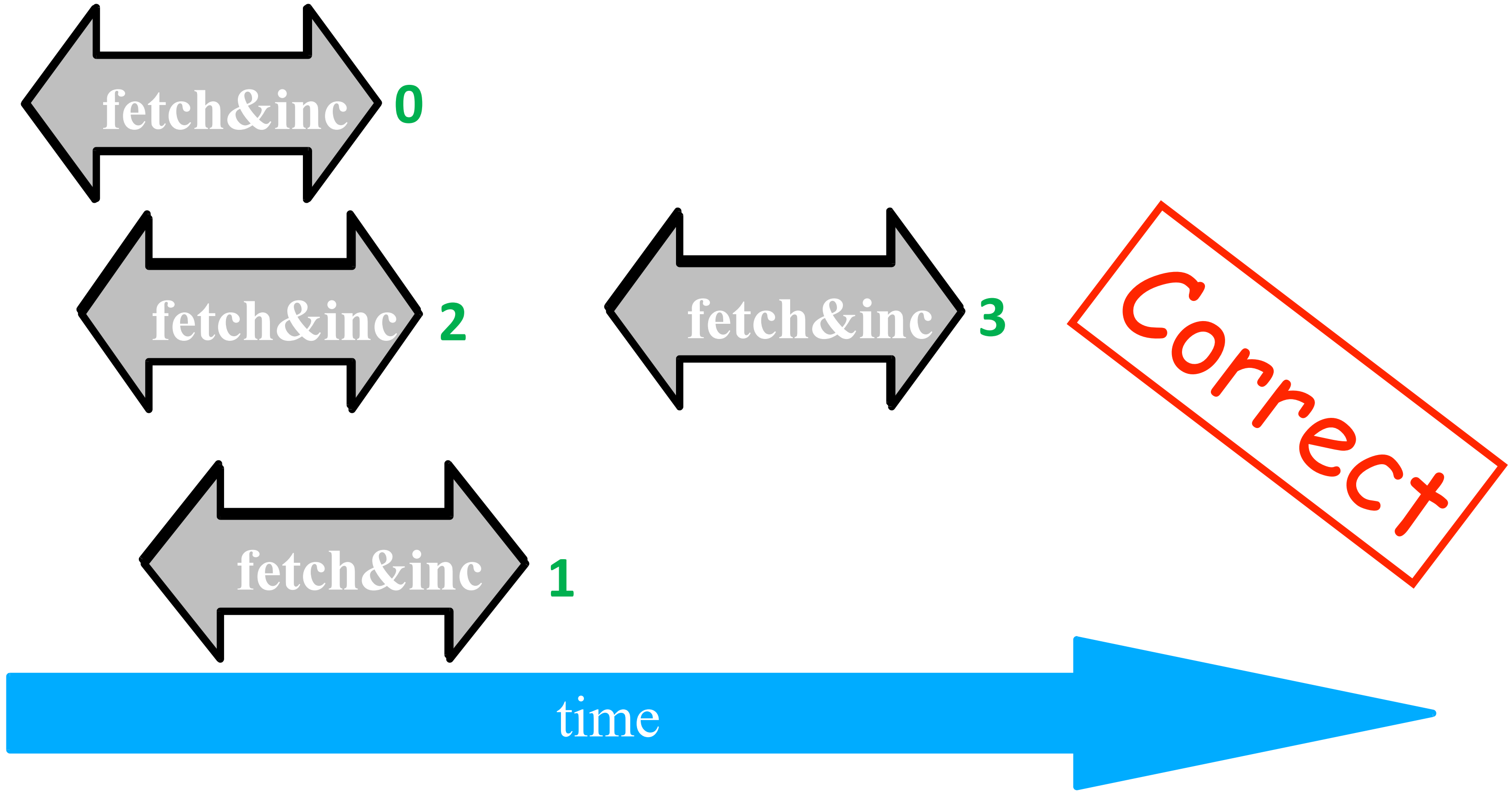


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

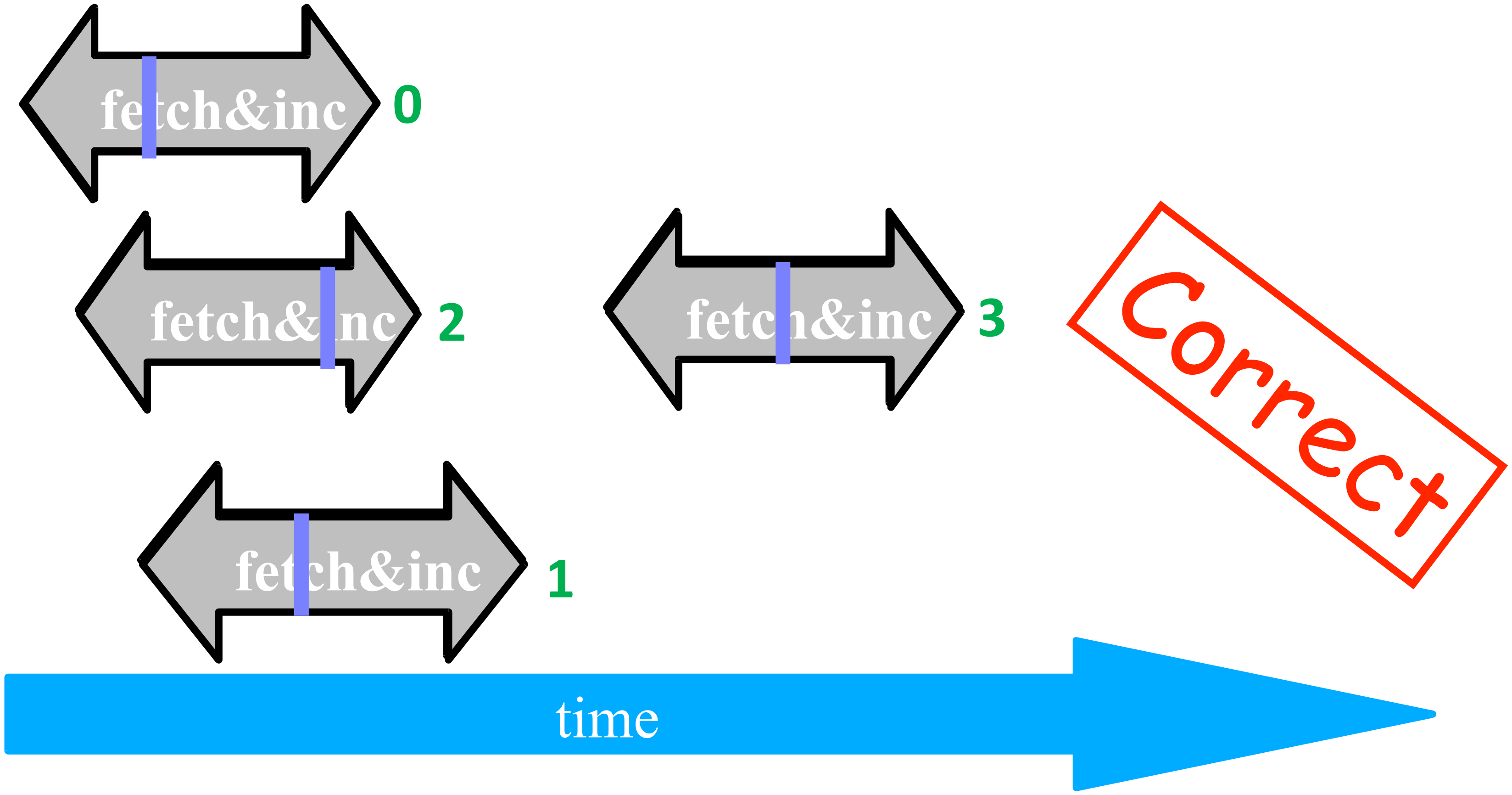
Correctness of concurrent counter?



Correctness of concurrent counter?



Correctness of concurrent counter?



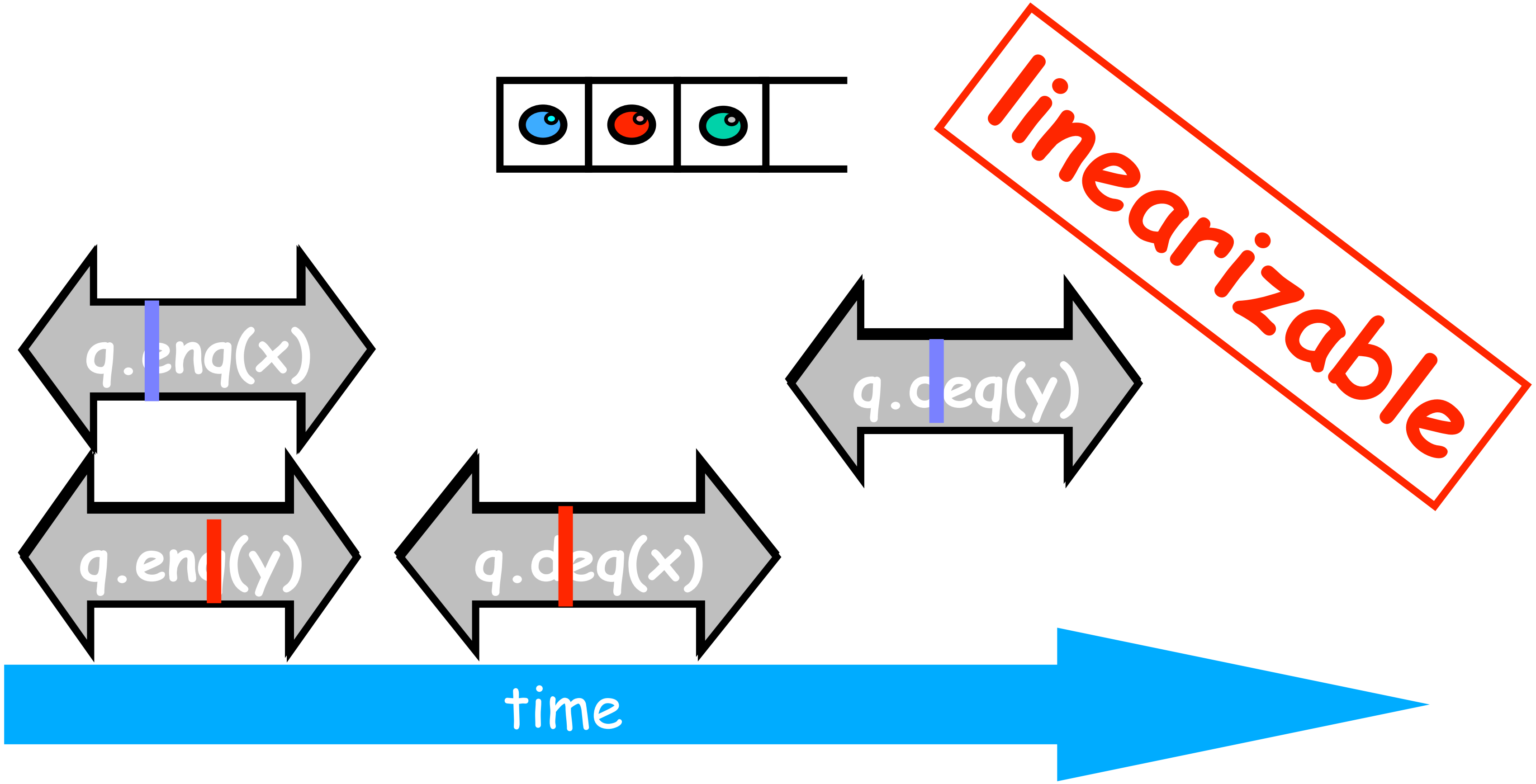
Linearizability definition

Linearizability

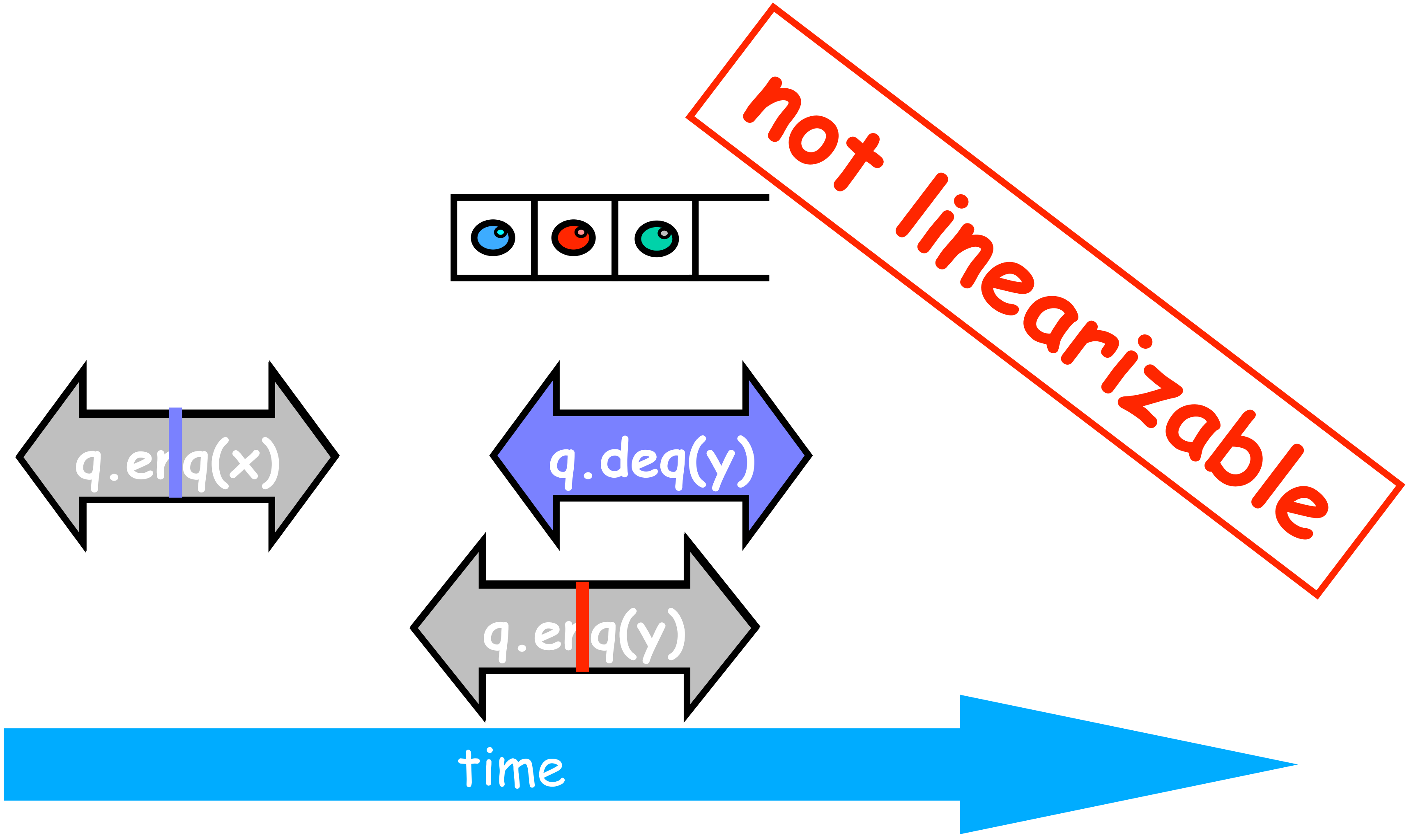
An execution is linearizable if there exists a permutation of the operations on each object o, π, such that:

- π is a sequential history of o
- π preserves the partial order of the execution.

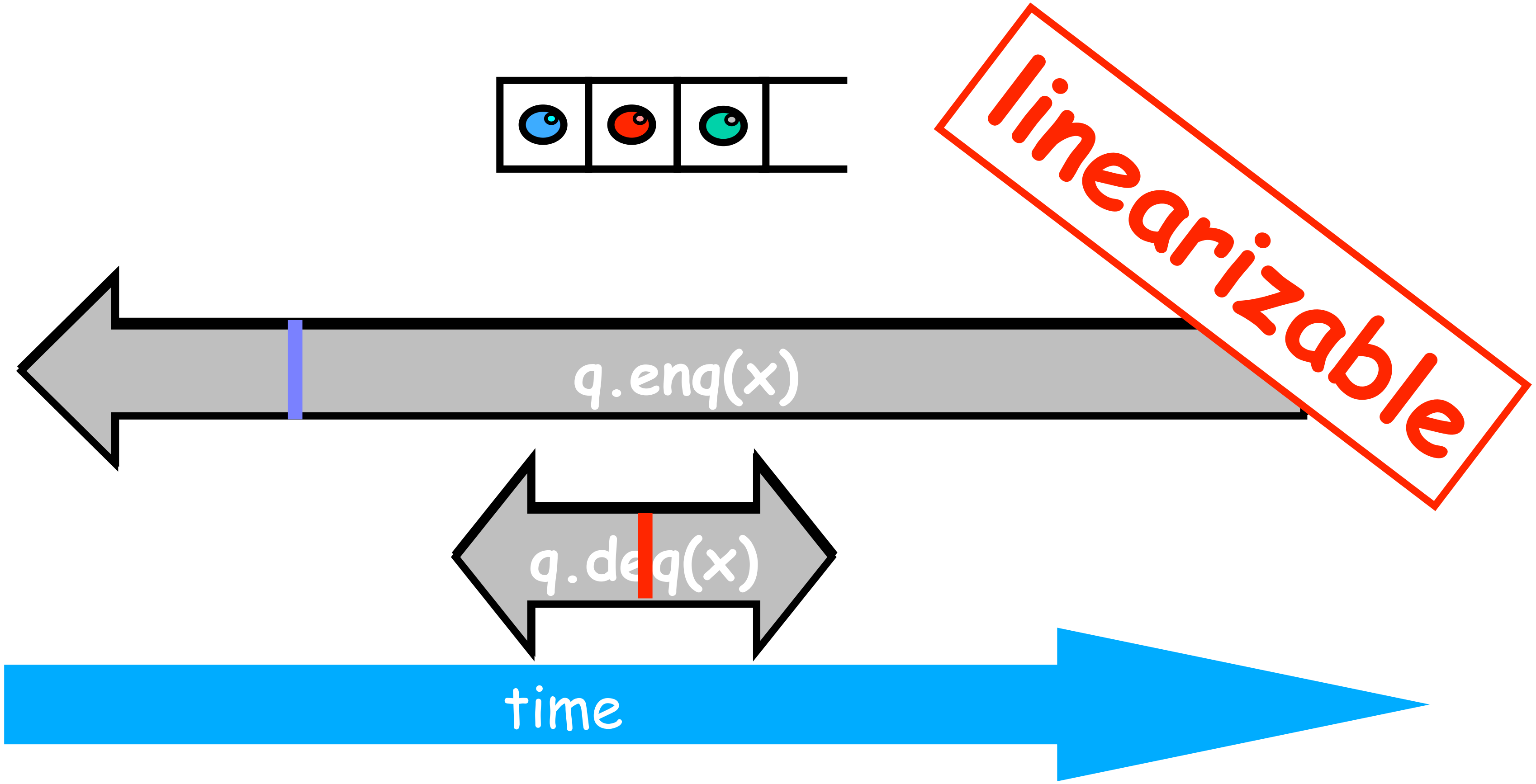
Linearizability: more examples



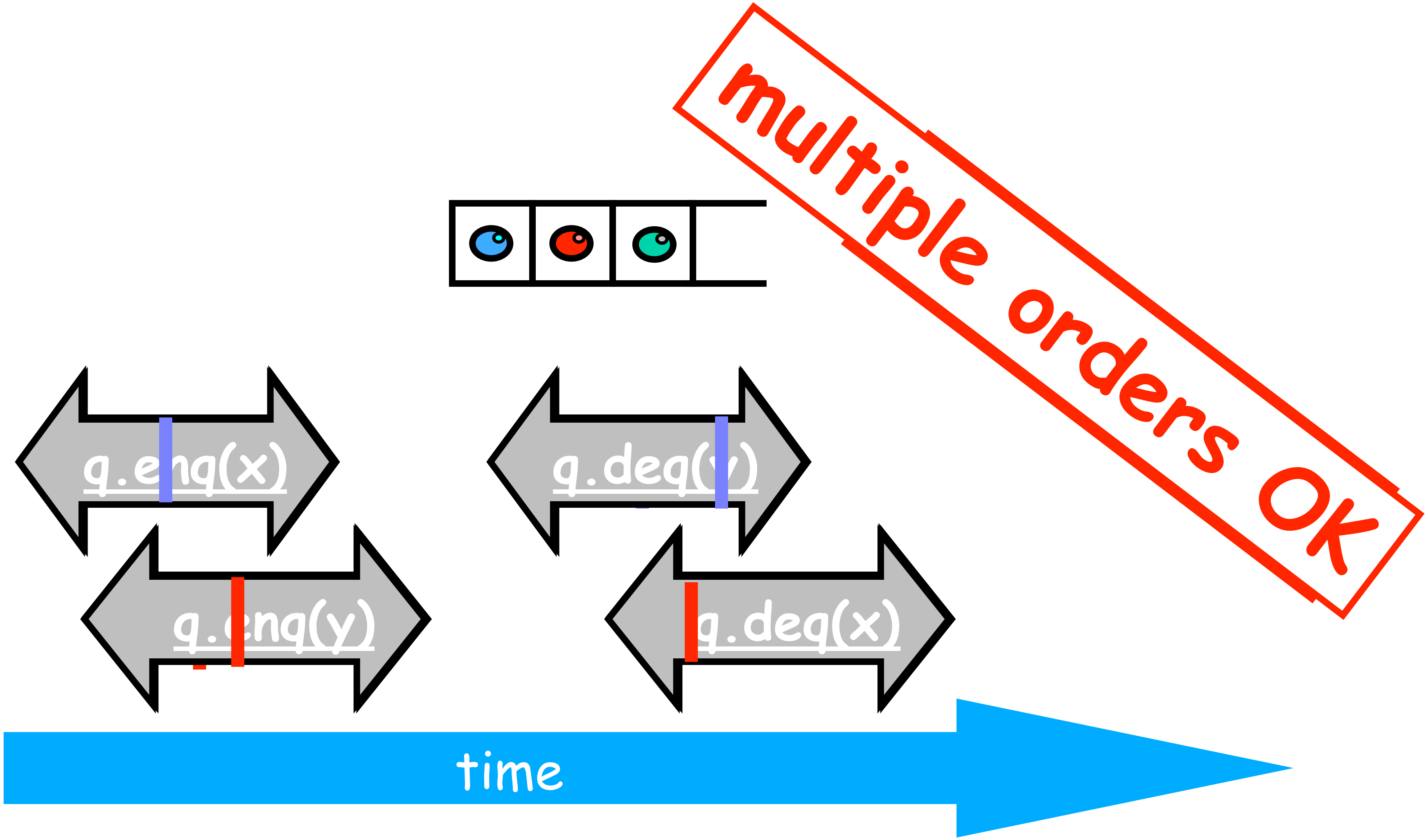
Linearizability: more examples



Linearizability: more examples



Linearizability: more examples



Back to Trieber’s stack algorithm
Push linearization points

Upon success

```
Push(int v, Stack S)
n := new NODE ;create node for new stack item
n.val := v          ;write item value
do forever        ;repeat until success
   node top := S.top
   n.next := top         ;next points to current (LIFO order)
   if compare&swap(S, top, n) ; try to add new item
       return                         ; return if succeeded
end do
```

Back to Trieber's stack algorithm
Pop linearization points

When empty

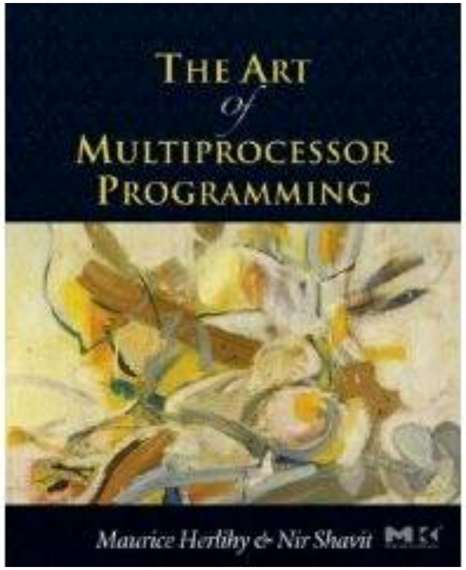
Upon success

```
Pop(Stack S)
do forever
    top := S.top
    if top = null
        return empty
    if compare&swap(S, top, top.next)
        return-val=top.val
return return-val
end do
```

Talk Outline

- Preliminaries
- A simple lock-free stack algorithm
  - Linearizability
- Michael & Scott queue algorithm
- The Harris-Michael linked list algorithm
- Elimination-based stack
- Discussion & conclusions

The art of multiprocessor programming

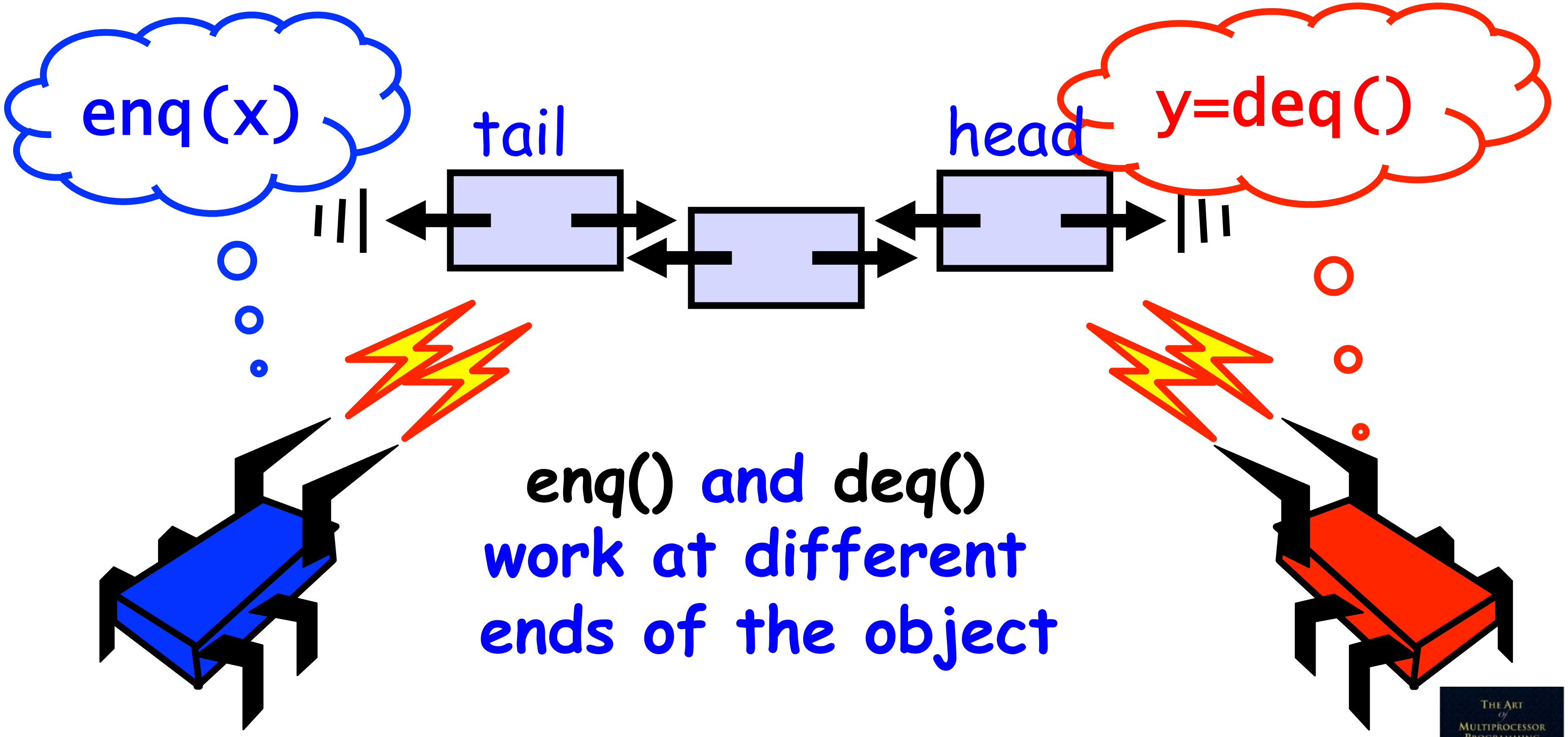


- Companion slides for
- The Art of Multiprocessor Programming
- by Maurice Herlihy & Nir Shavit

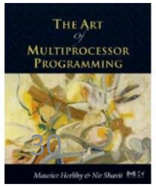
Queue interface

- ❑ Pool of items
- ❑ First-in-first-out
- ❑ Methods
  - – **enq(x)** adds **x** at the end of the queue
  - – **deq** returns the item at the head of the queue or an empty indication

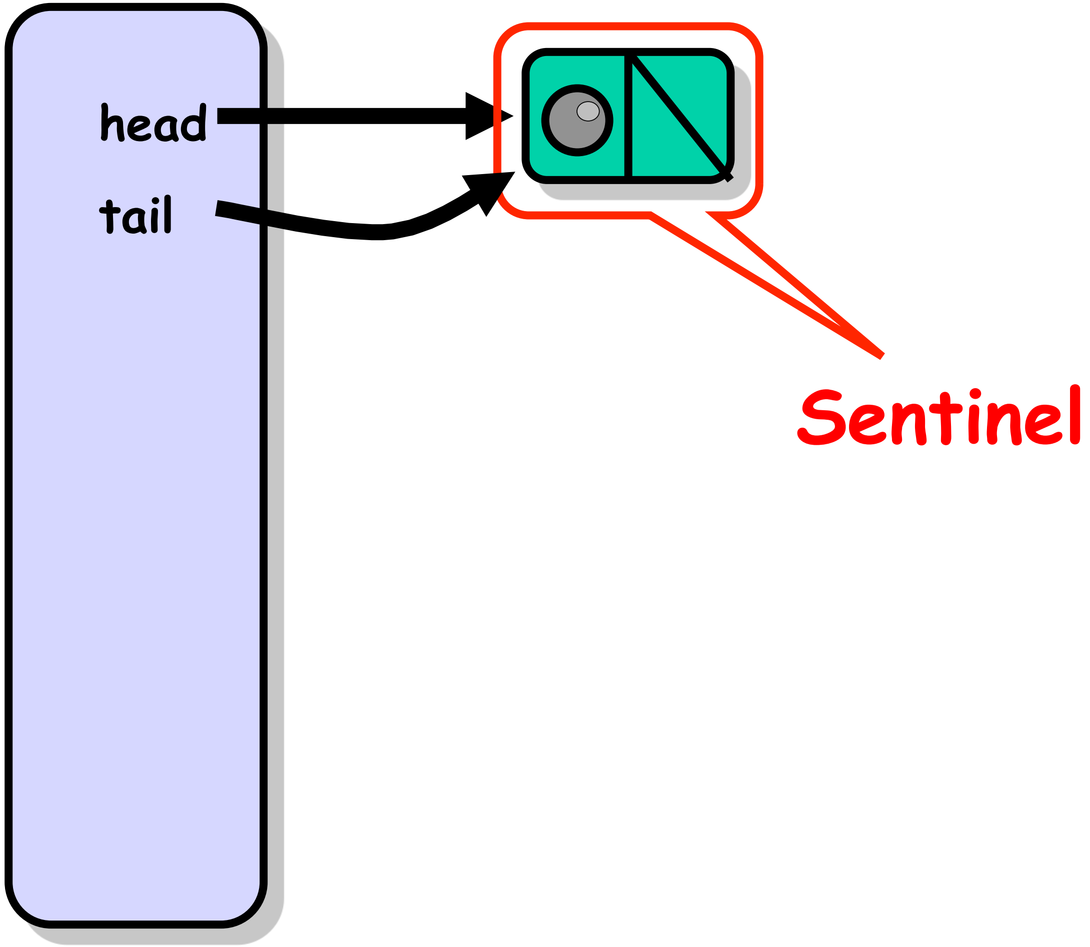
Queue: concurrency



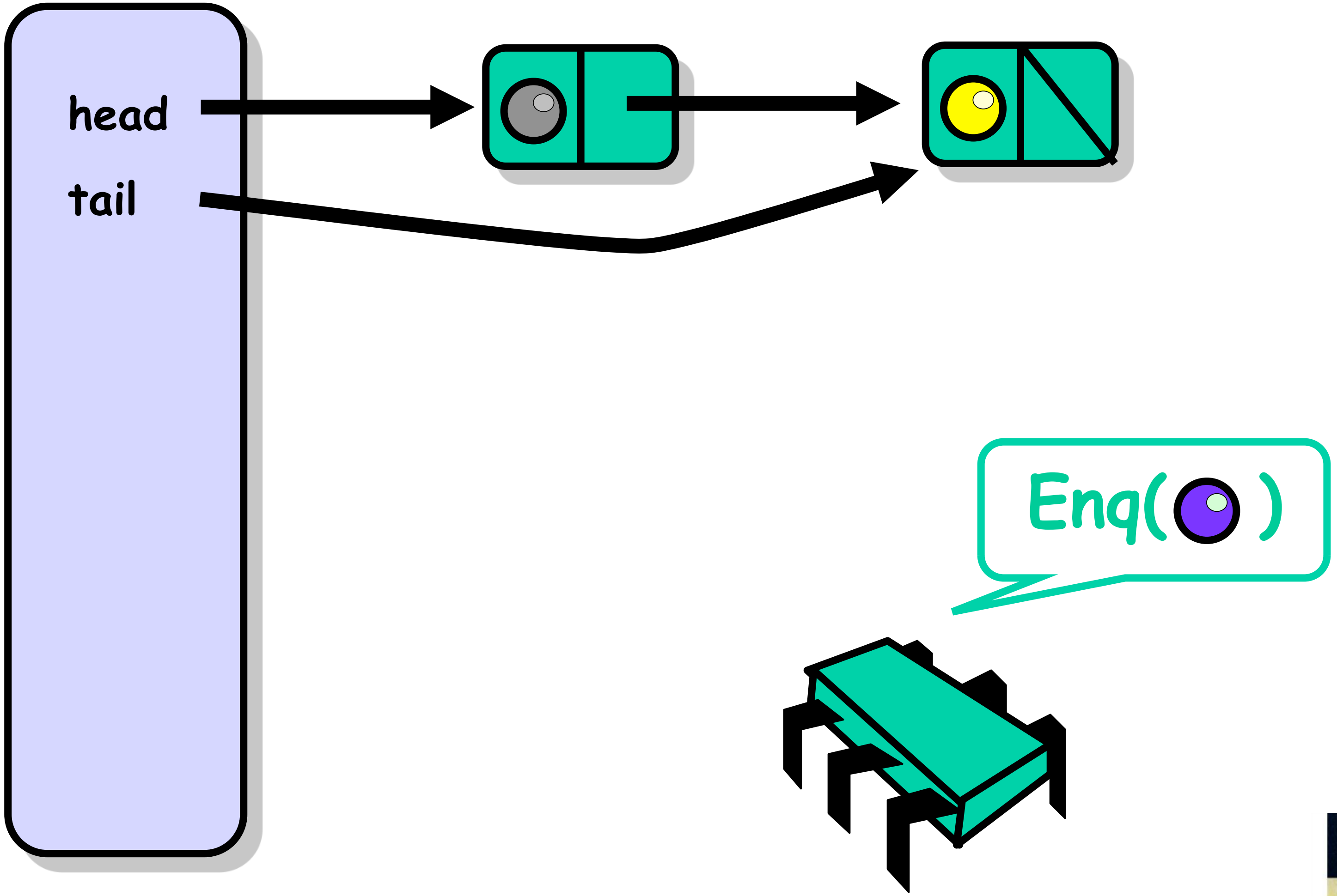
enq() and deq()
work at different
ends of the object

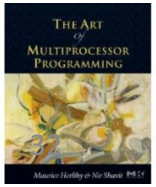


Michael & Scott queue
Sentinel

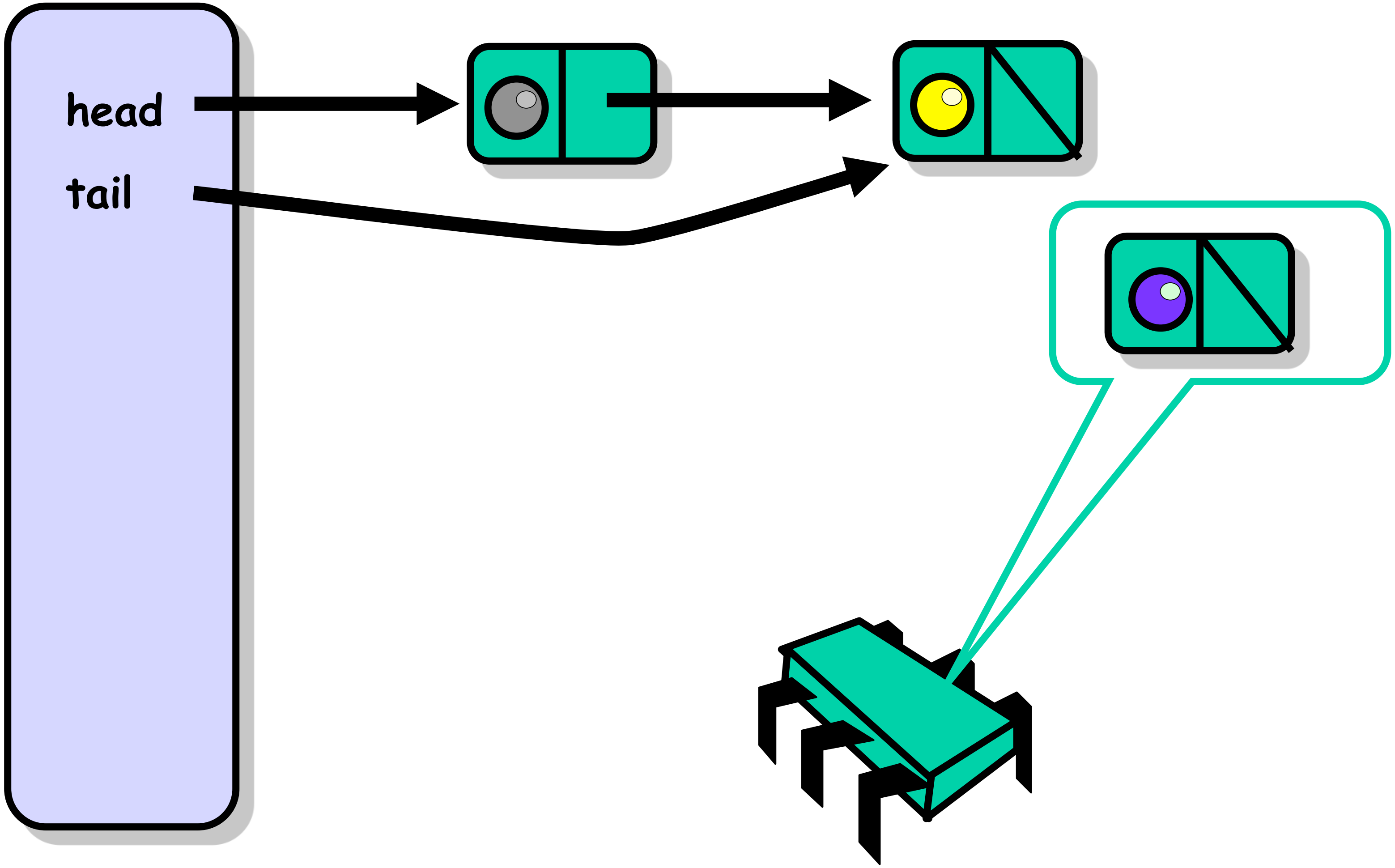


Michael & Scott queue Enq

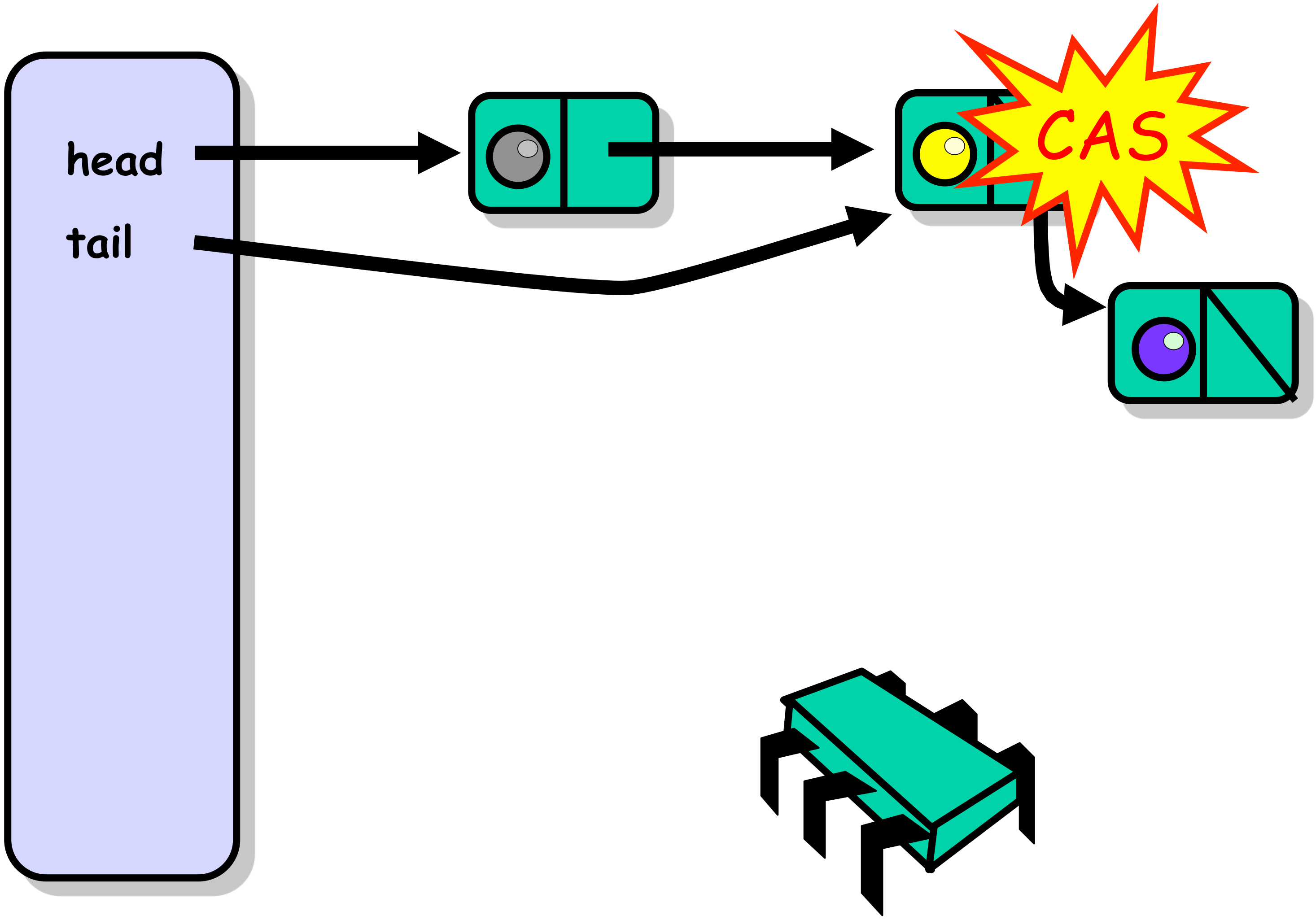




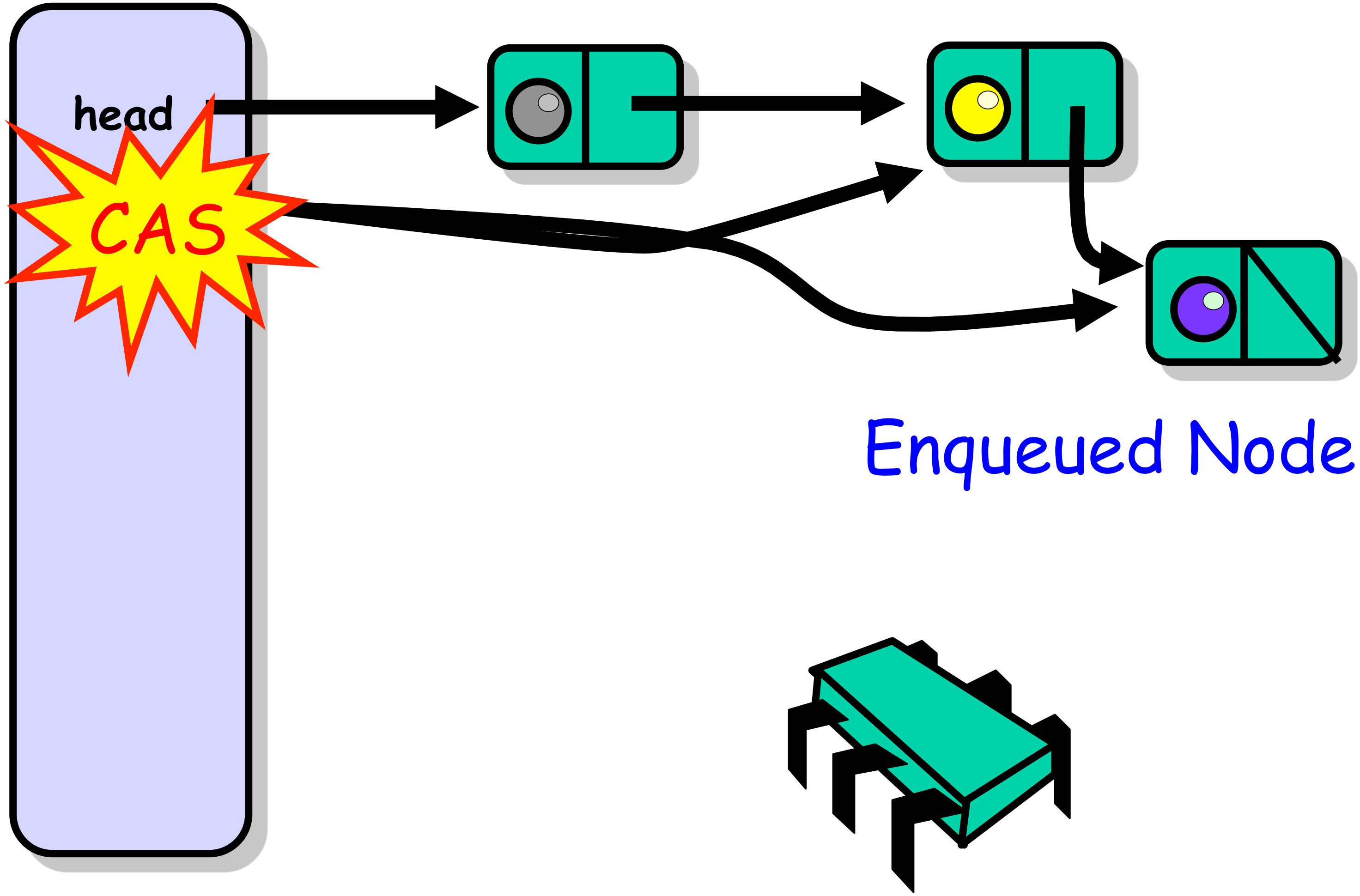
Michael & Scott queue
Enq



Michael & Scott queue
Enq: first CAS



Michael & Scott queue
Enq: second CAS



Michael & Scott queue Enq

- ❑ Two CAS operations (not atomic)

- ❑ Tail references either:
  - ○ Actual last node
  - ○ One-before-last node (needs to be fixed!)

⇩

If tail has non-null *next* reference, CAS tail to tail.*next*

AtomicReference
Atomically update reference

- AtomicReference *class*
  - Java.util.concurrent.atomic *package*

```
Public object get();
```

```
Public boolean
  compareAndSet (T expected, T new);
```

AtomicReference
Atomically update reference

- AtomicReference class
  - Java.util.concurrent.atomic package

```
Public object get();
```

```
Public boolean
   compareAndSet (T expected, T new);
```

Returns current reference

AtomicReference
Atomically update reference

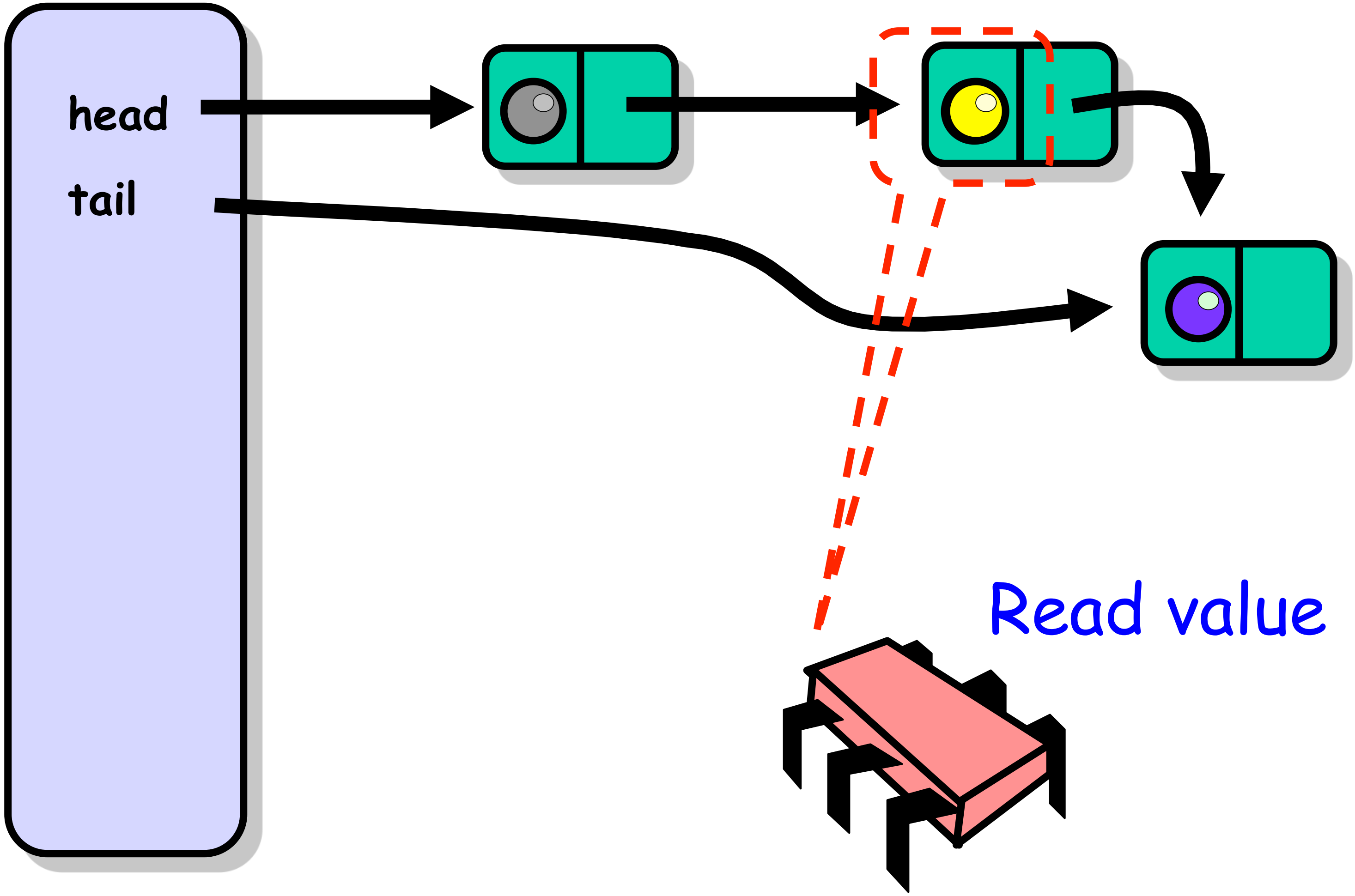
- AtomicReference class
  - Java.util.concurrent.atomic package

```
Public object get();
```

```
Public boolean
  compareAndSet (T expected, T new);
```

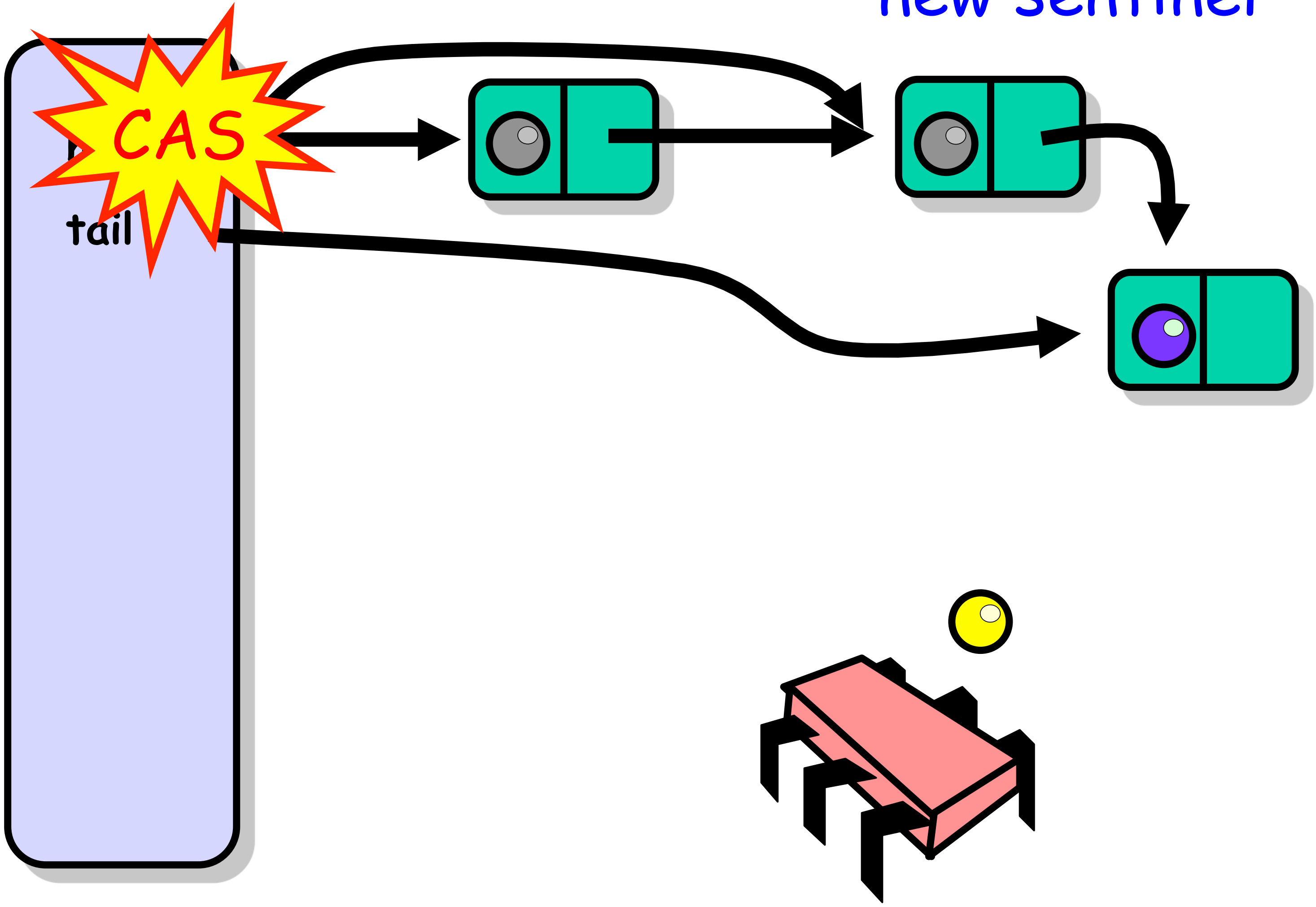
Apply CAS: if expected value, change to new

Michael & Scott queue
Deq



Michael & Scott queue Deq

Make first Node new sentinel



Michael & Scott queue
Queue node

```
public class Node {
 public T value;
 public AtomicReference<Node> next;
 public Node(T value) {
     this.value=value;
     next=new AtomicReference<Node>(null);
     }
}
```

Michael & Scott queue
Queue node

```
public class Node {
 public T value;
 public AtomicReference<Node> next;
 public Node(T value) {
     this.value=value;
     next=new AtomicReference<Node>(null);
     }
}
```

Value stored by node

Michael & Scott queue
Queue node

```
public class Node {
 public T value;
 public AtomicReference<Node> next;
 public Node(T value) {
     this.value=value;
     next=new AtomicReference<Node>(null);
     }
}
```

Reference to next queue node

Michael & Scott queue
Queue node

```
public class Node {
 public T value;
 public AtomicReference<Node> next;
 public Node(T value) {
     this.value=value;
     next=new AtomicReference<Node>(null);
     }
}
```

New node created with null 'next'

Michael & Scott queue
Enq pseudo-code

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
       if (next == null) {
           if (last.next.compareAndSet(null,node) {
               tail.compareAndSet(last,node);
               return;
           }
        } else {
           tail.compareAndSet(last,next);
        }
      }
  }
}
```

Michael & Scott queue
Enq pseudo-code

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
       if (next == null) {
           if (last.next.compareAndSet(null,node) {
               tail.compareAndSet(last,node);
               return;
           }
        } else {
           tail.compareAndSet(last,next);
        }
      }
  }
}
```

Create new node

Michael & Scott queue
Enq pseudo-code

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
       if (next == null) {
           if (last.next.compareAndSet(null,node) {
               tail.compareAndSet(last,node);
               return;
           }
        } else {
           tail.compareAndSet(last,next);
        }
      }
  }
}
```

Repeat until successful

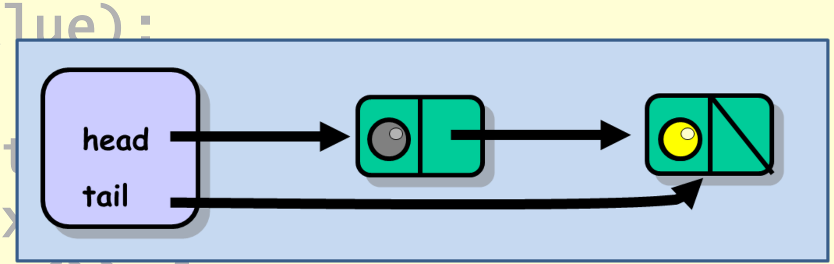
Michael & Scott queue Enq pseudo-code

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
      if (next == null) {
        if (last.next.compareAndSet(null,node) {
          tail.compareAndSet(last,node);
          return;
        }
      } else {
        tail.compareAndSet(last,next);
      }
    }
  }
}
```

Read tail and its next reference

Michael & Scott queue
Enq pseudo-code

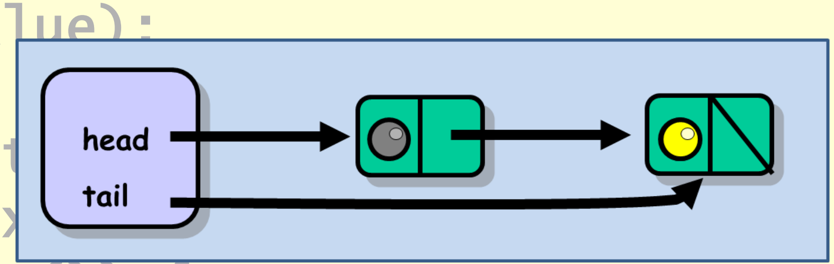
```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get
    Node next = last.nex
    if (last == tail.get()) {
      if (next == null) {
        if (last.next.compareAndSet(null,node) {
          tail.compareAndSet(last,node);
          return;
        }
      } else {
        tail.compareAndSet(last,next);
      }
    }
  }
}
```



If no need to fix tail, CAS last.next

Michael & Scott queue
Enq pseudo-code

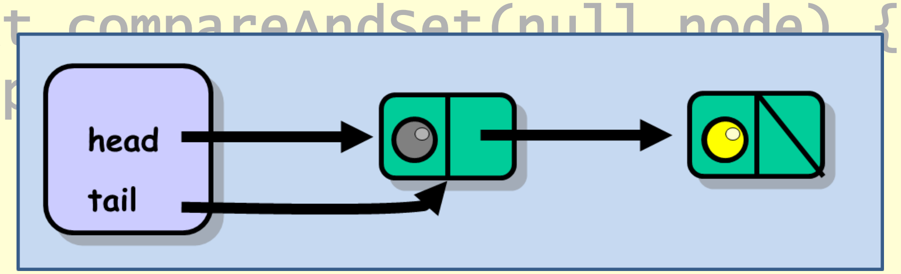
```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get
    Node next = last.nex
    if (last == tail.get()) {
       if (next == null) {
           if (last.next.compareAndSet(null,node) {
               tail.compareAndSet(last,node);
               return;
           }
       } else {
           tail.compareAndSet(last,next);
       }
     }
  }
}
```



If successful, try to fix tail

Michael & Scott queue
Enq pseudo-code

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
       if (next == null) {
          if (last.next.compareAndSet(null,node)) {
             tail.comp
             return;
          }
       } else {
          tail.compareAndSet(last,next);
       }
     }
  }
}
```



Try to fix tail

Michael & Scott queue
Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Michael & Scott queue
Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Return value or throw EmptyException

Michael & Scott queue
Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Repeat until completed

Michael & Scott queue
Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

If head and tail are same node…

Michael & Scott queue Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

If queue contains only sentinel, it is empty

Michael & Scott queue
Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Otherwise, tail should be fixed

Michael & Scott queue
Deq pseudo-code

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Try to dequeue from first node

Michael & Scott queue
Enq linearization points

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
       if (next == null) {
           if (last.next.compareAndSet(null,node) {
               tail.compareAndSet(last,node);
               return;
           }
        } else {
           tail.compareAndSet(last,next);
        }
      }
  }
}
```

Michael & Scott queue Enq linearization points

```
public boolean enq(T value) {
  Node node=new Node(value);
  while (true) {
    Node last = tail.get();
    Node next = last.next.get();
    if (last == tail.get()) {
       if (next == null) {
           if (last.next.compareAndSet(null,node) {
               tail.compareAndSet(last,node);
               return;
           }
       } else {
           tail.compareAndSet(last,next);
       }
     }
  }
}
```

Upon success

Michael & Scott queue
Deq linearization points

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Michael & Scott queue
Deq linearization points

```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Upon success

Michael & Scott queue Deq linearization points

When empty

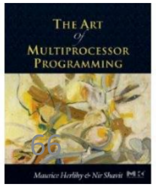
```
public T deq() throws EmptyException{
  while (true) {
    Node first = head.get();
    Node last = tail.get();
    Node next = first.next.get();
    if (first == last) {
       if (next == null) {
           throw new EmptyException();
           }
           tail.compareAndSet(last,next);
        } else {
           T value = next.value;
           if (head.compareAndSet(first,next))
              return value;
        }
     }
  }
}
```

Talk Outline

- Preliminaries
- A simple lock-free stack algorithm
  - Linearizability
- Michael & Scott queue algorithm
- The Harris-Michael linked list algorithm
- Elimination-based stack
- Discussion & conclusions

Set interface

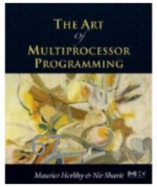
- ❑ Unordered collection of items
- ❑ No duplicates
- ❑ Methods
  - **add(x)** put **x** in set
  - **remove(x)** take **x** out of set
  - **contains(x)** tests if **x** in set



List-based sets

```
public interface Set<T> {
 public boolean add(T x);
 public boolean remove(T x);
 public boolean contains(T x);
}
```

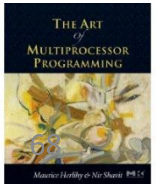
Add item to set



List-based sets

```
public interface Set<T> {
 public boolean add(T x);
 public boolean remove(T x);
 public boolean contains(T x);
}
```

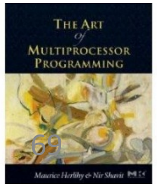
Remove item from set



List-based sets

```
public interface Set<T> {
 public boolean add(T x);
 public boolean remove(T x);
 public boolean contains(T x);
}
```

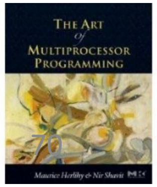
Is item in set?



The Art of Multiprocessor Programming

List Node

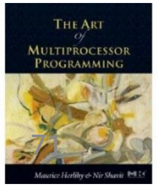
```
public class Node {
 public T item;
 public int key;
 public Node next;
}
```


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

List Node

```
public class Node {
 public T item;
 public int key;
 public Node next;
}
```

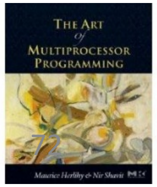
item of interest


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

List Node

```
public class Node {
 public T item;
 public int key;
 public Node next;
}
```

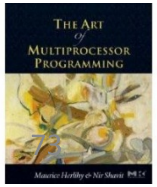
Usually hash code



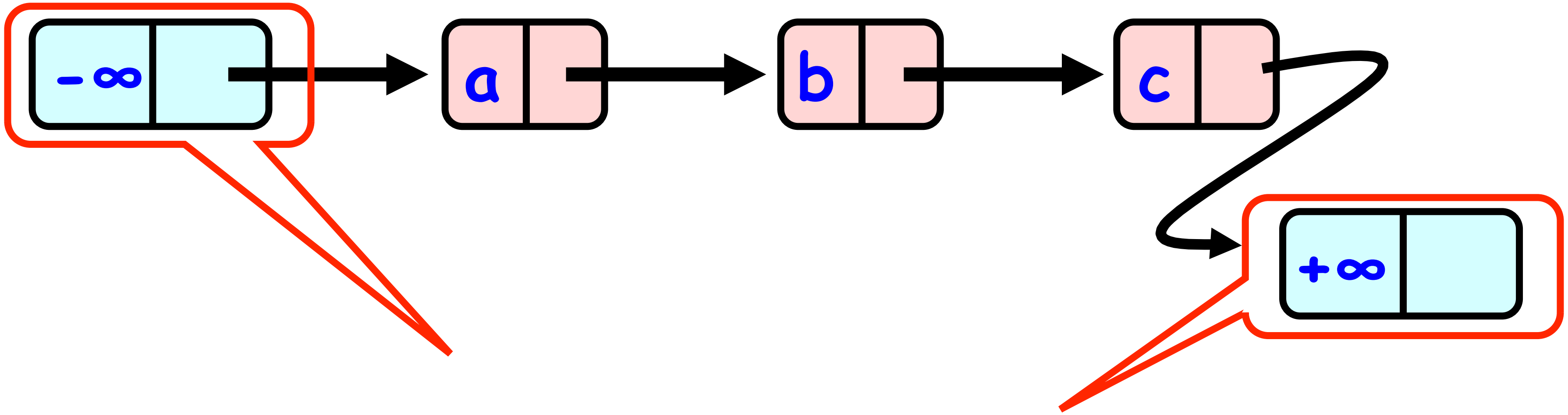
List Node

```
public class Node {
 public T item;
 public int key;
 public Node next;
}
```

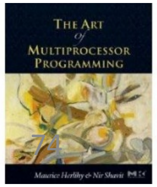
Reference to next node



The List-Based Set

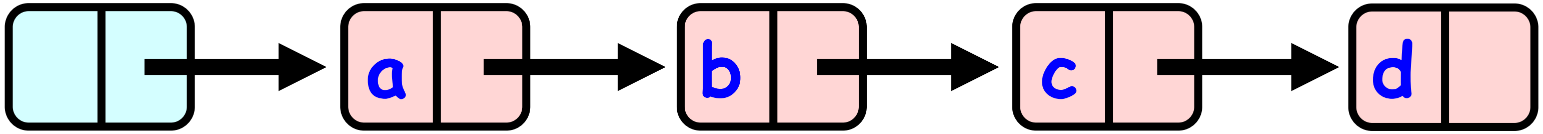


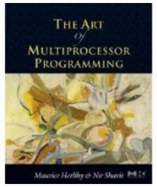
Sorted with Sentinel nodes (min & max possible keys)


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

The List-Based Set
Why synchronization is required

- □ Scan list from left to right, apply operation `at the right place’
- □ Not so simple…

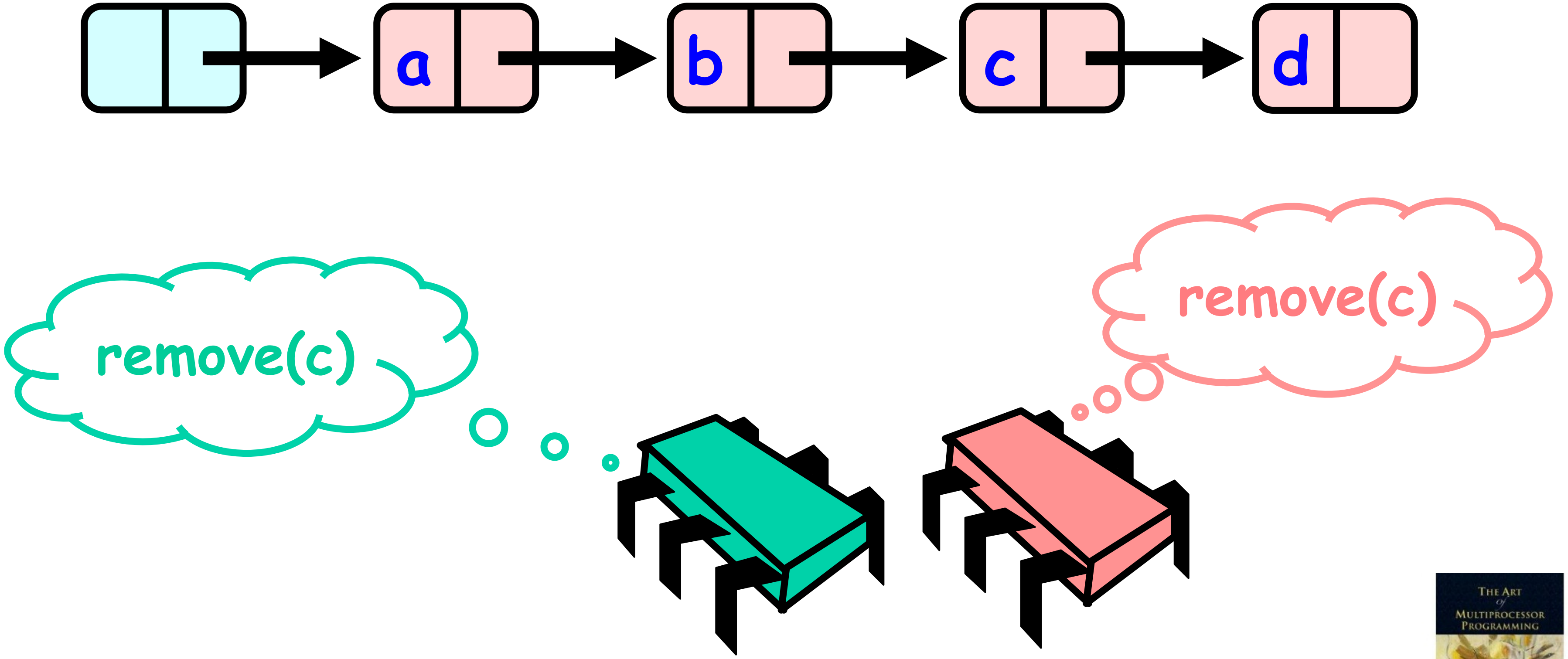



The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

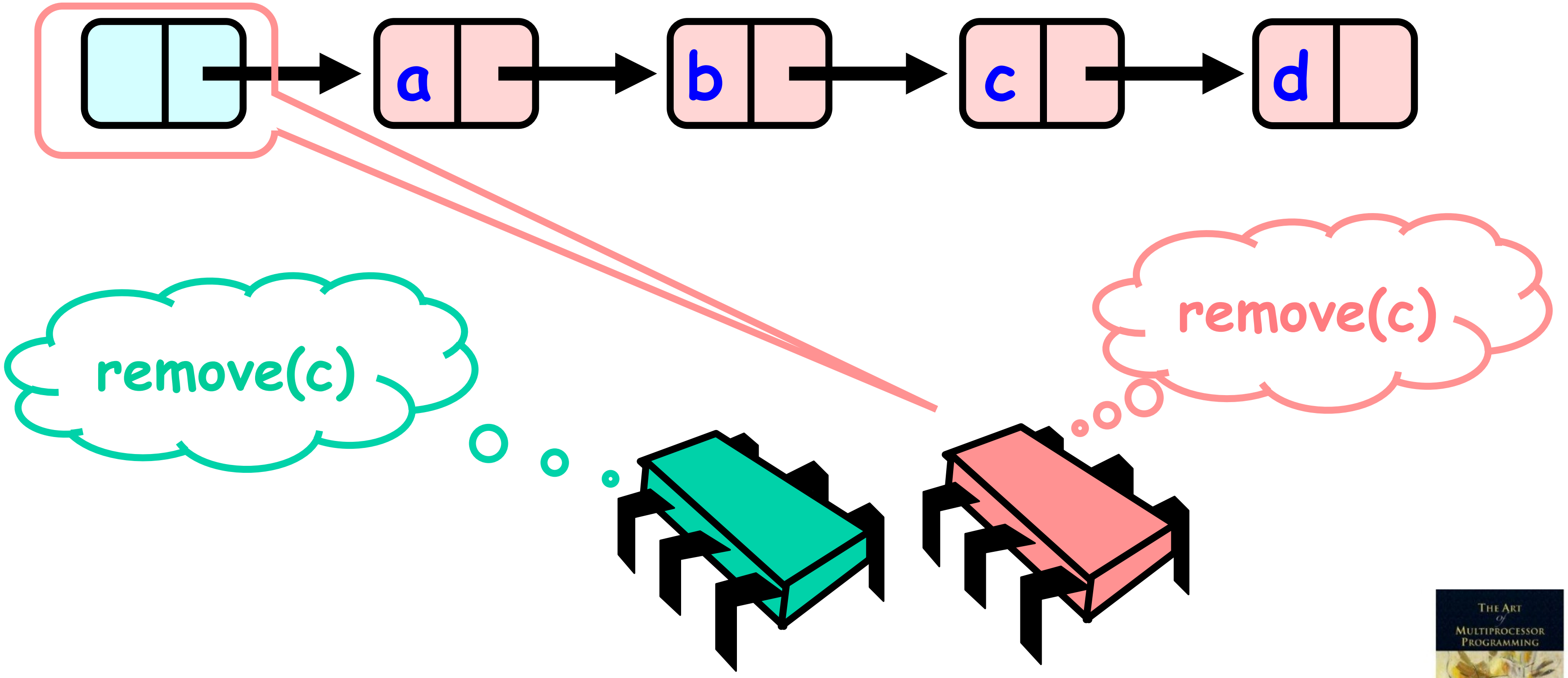
- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…



Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

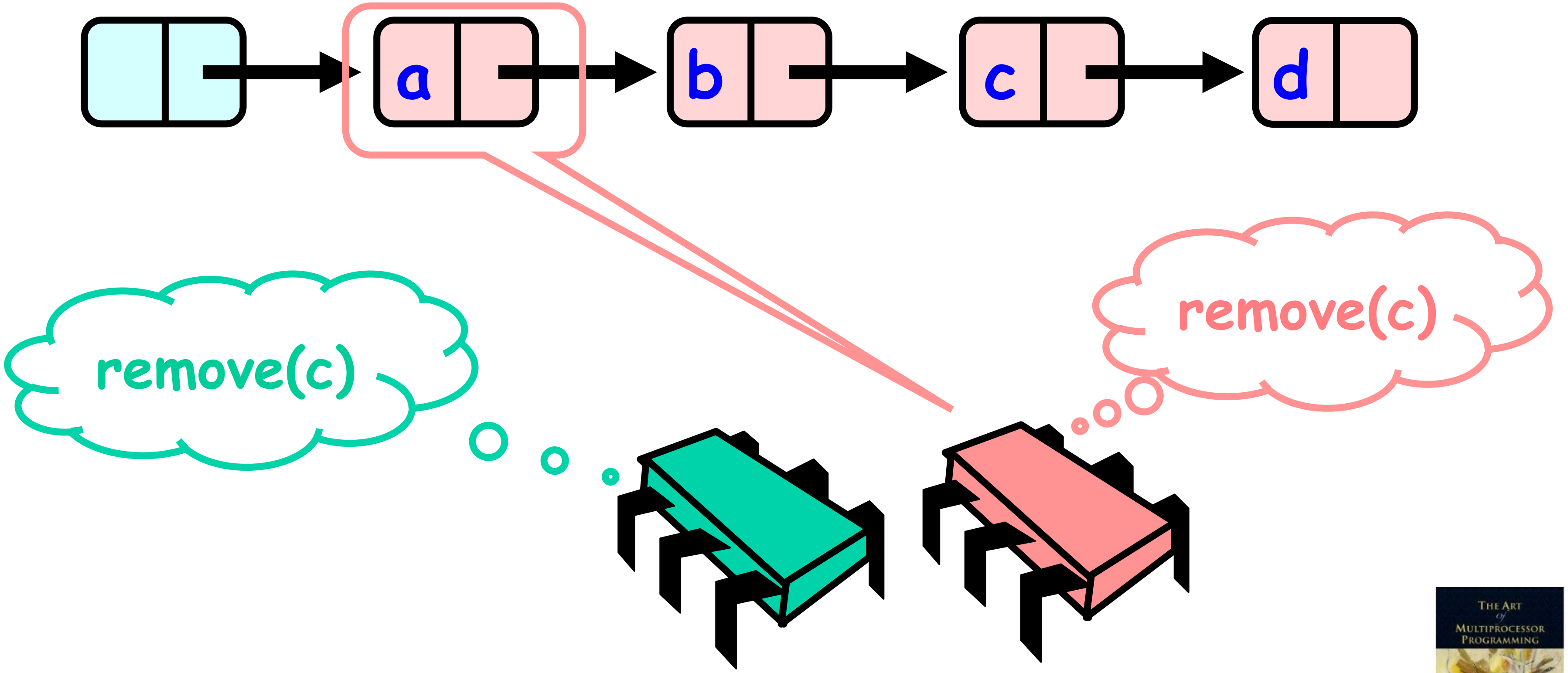
- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…

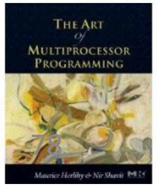


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…

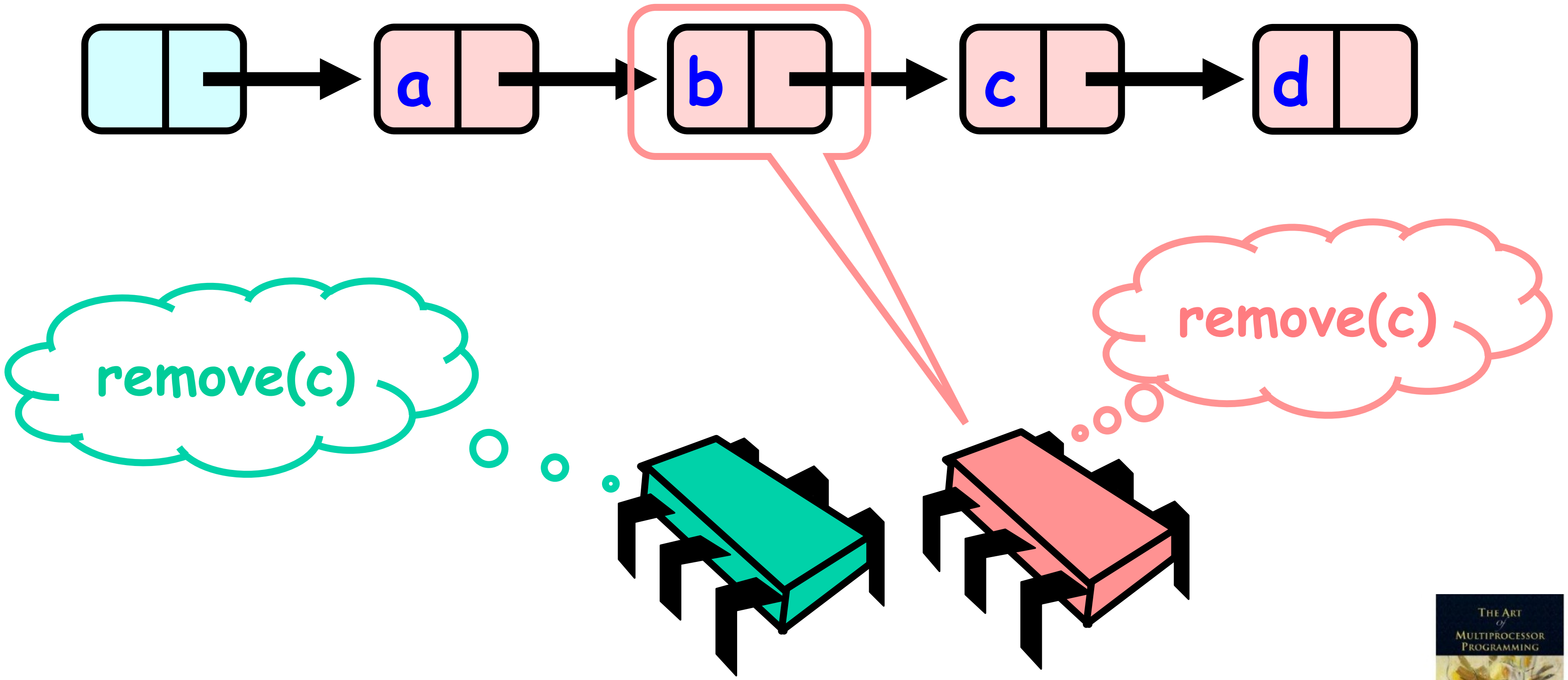


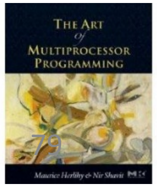


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…

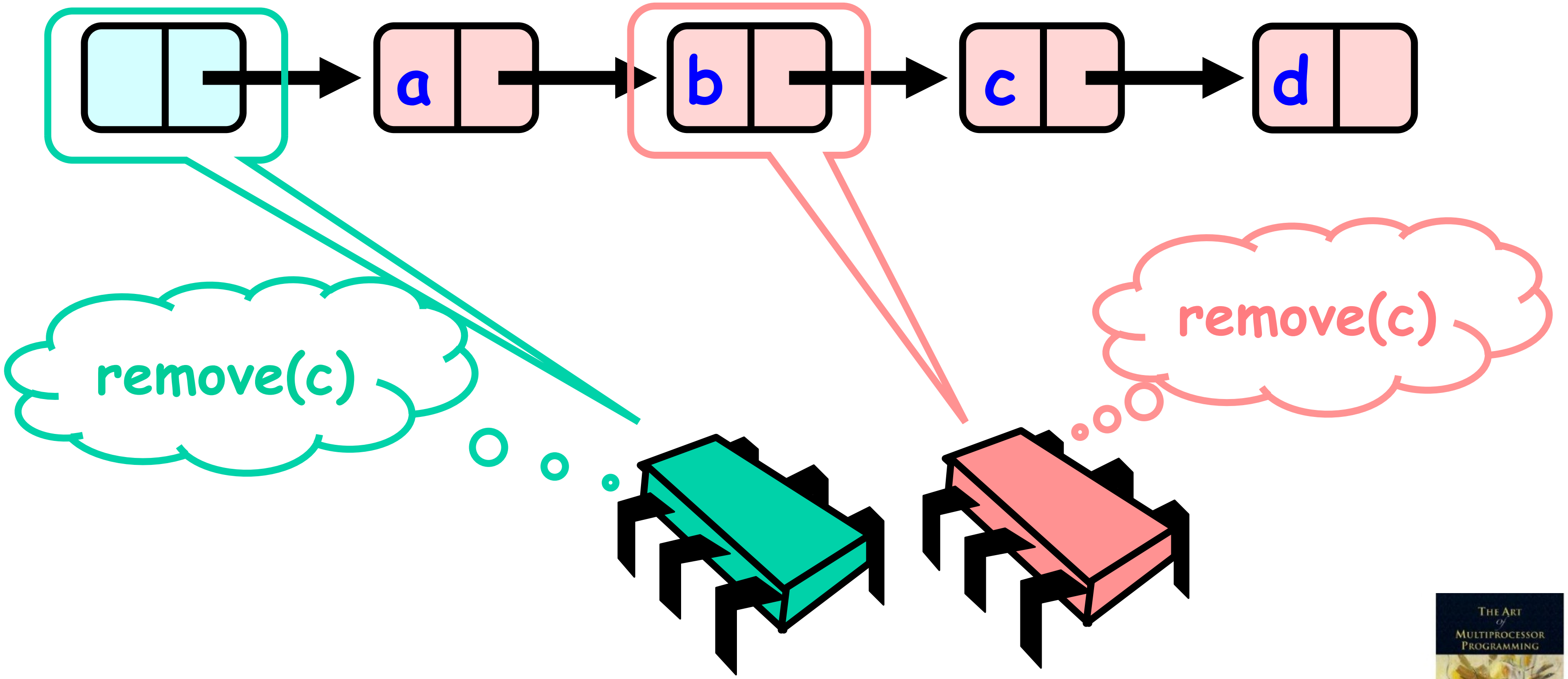


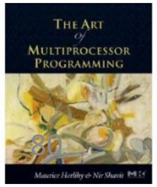


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

- ❑ Scan list from left to right, apply operation `at the right place'
- ❑ Not so simple…

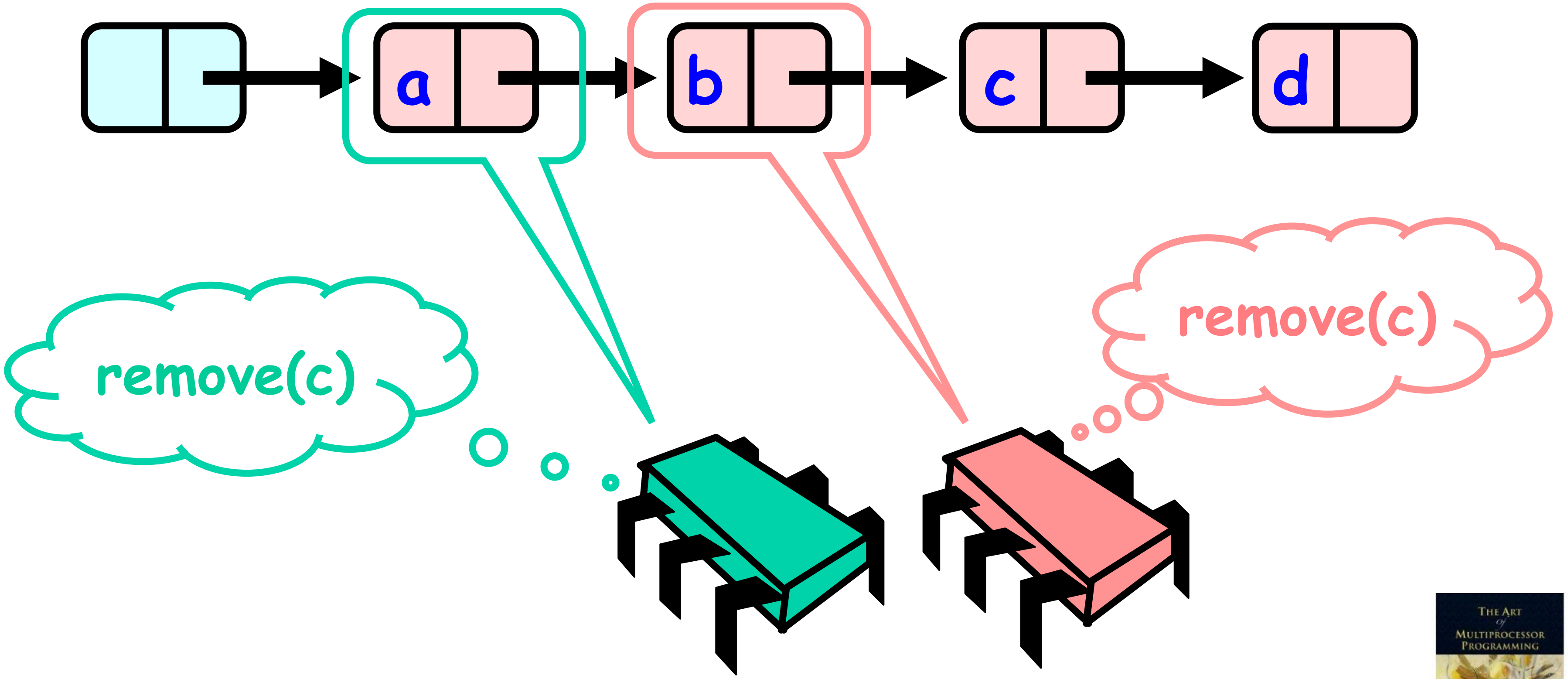


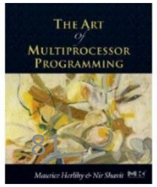


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

- Scan list from left to right, apply operation `at the right place’
- Not so simple…

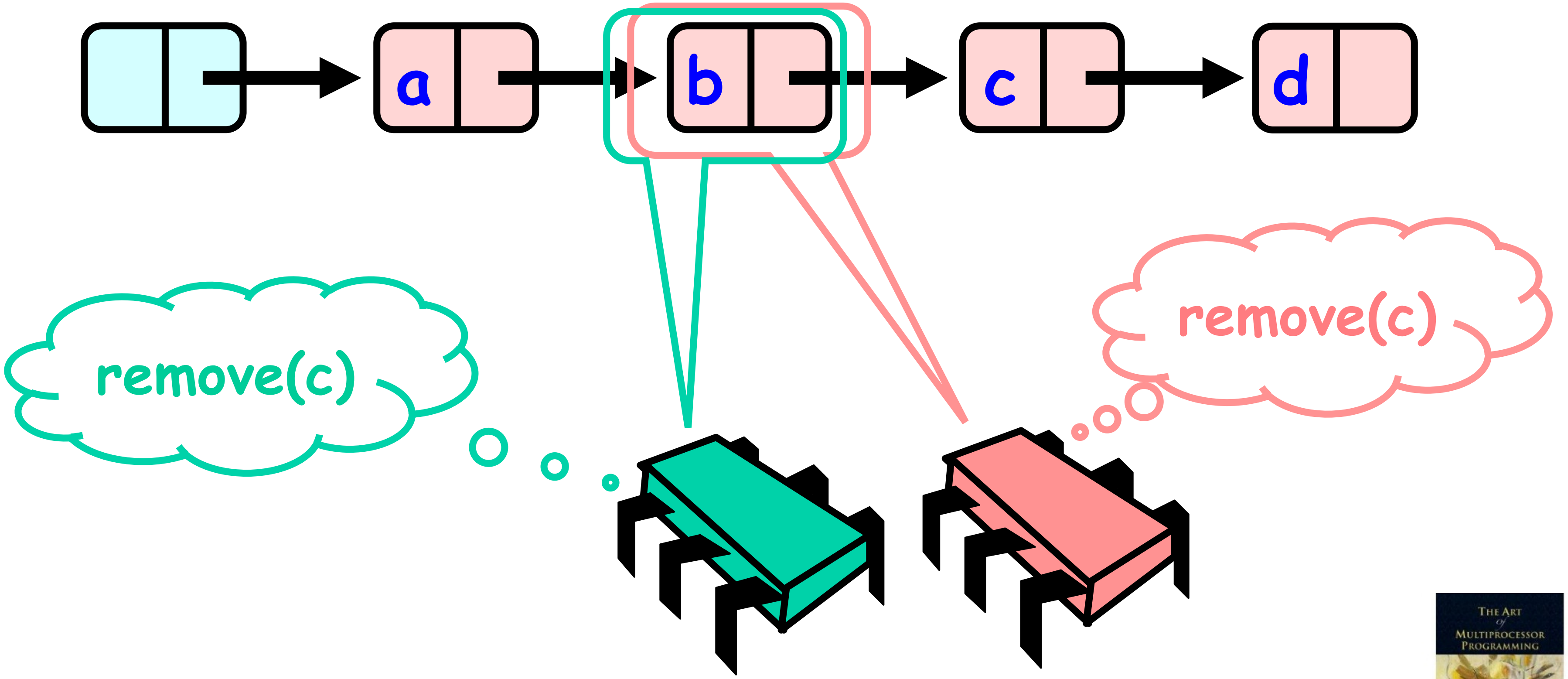


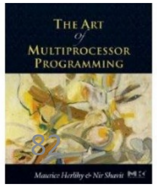


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…

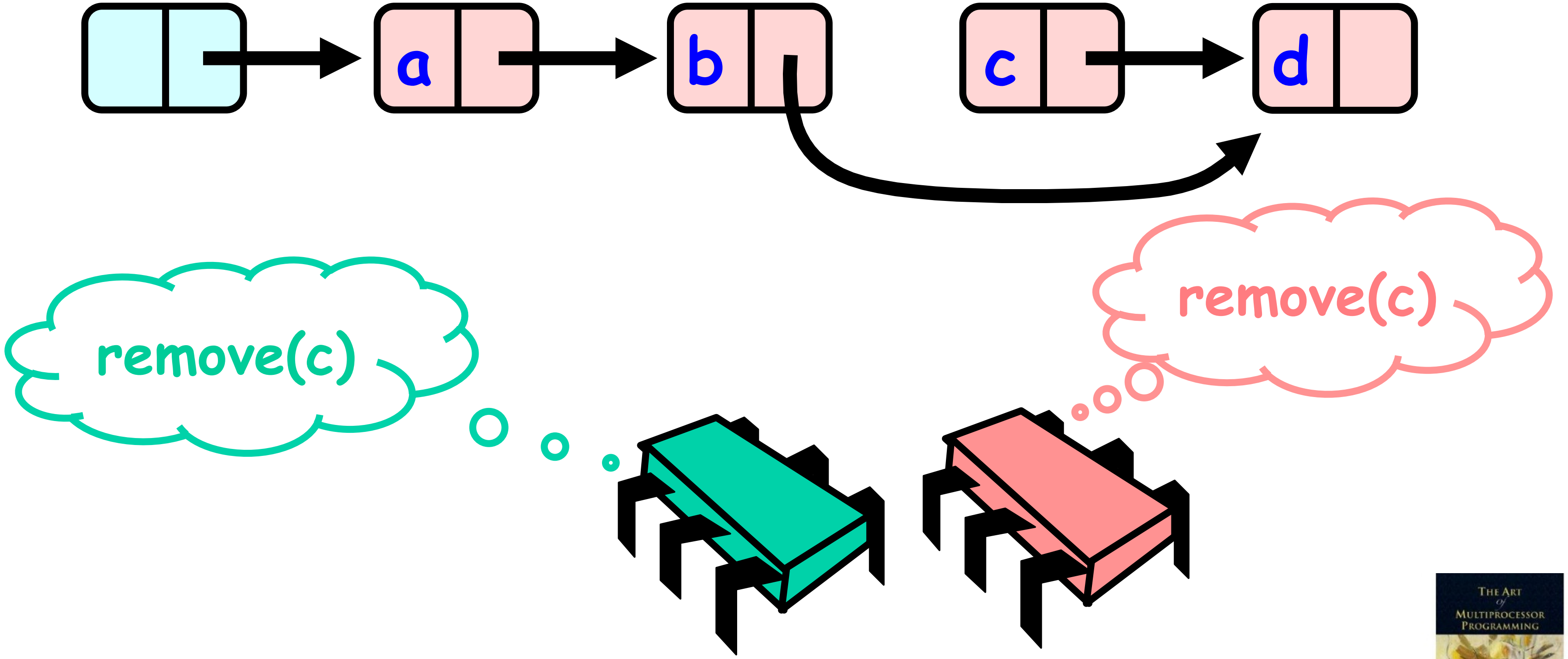


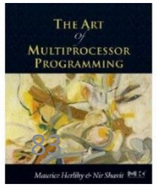


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…

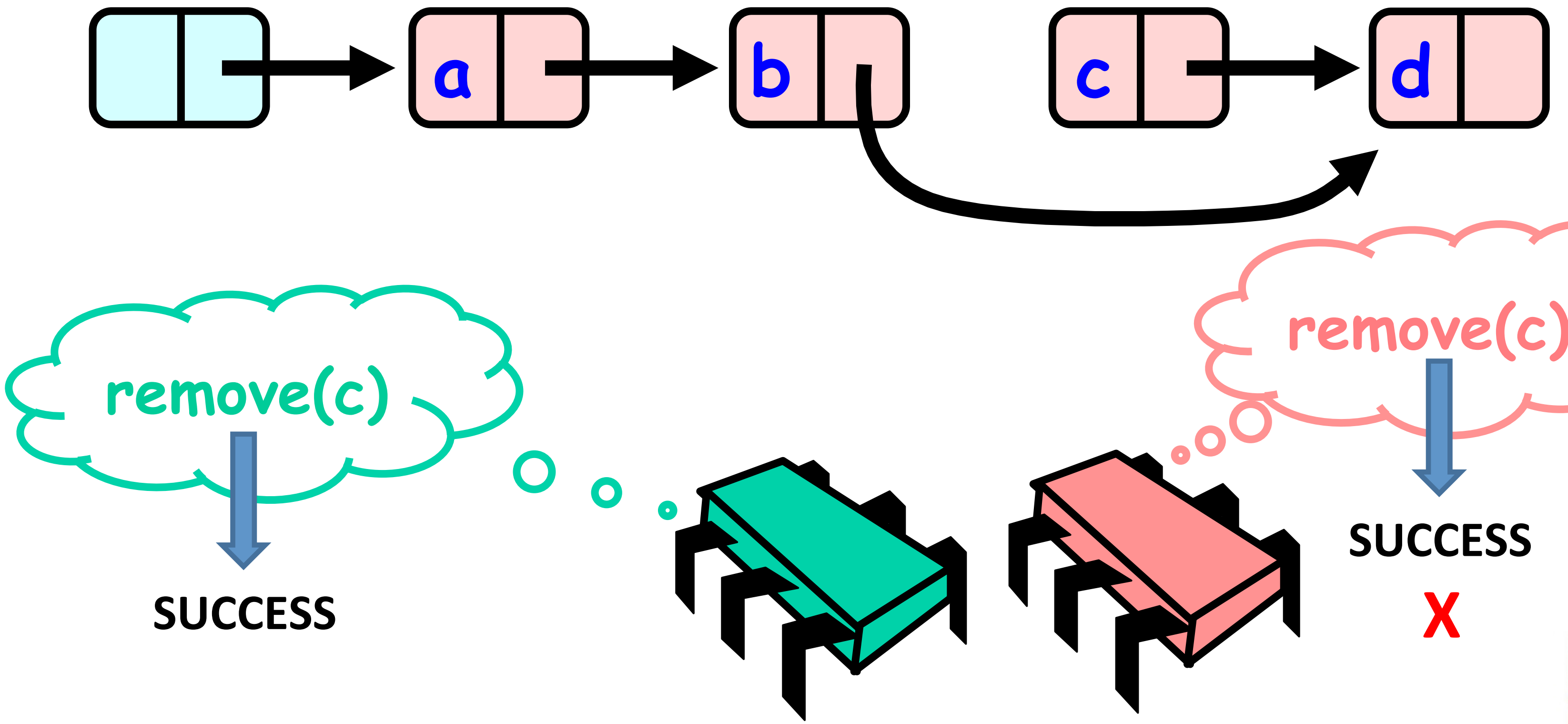


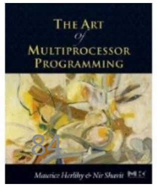


Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Why synchronization is required

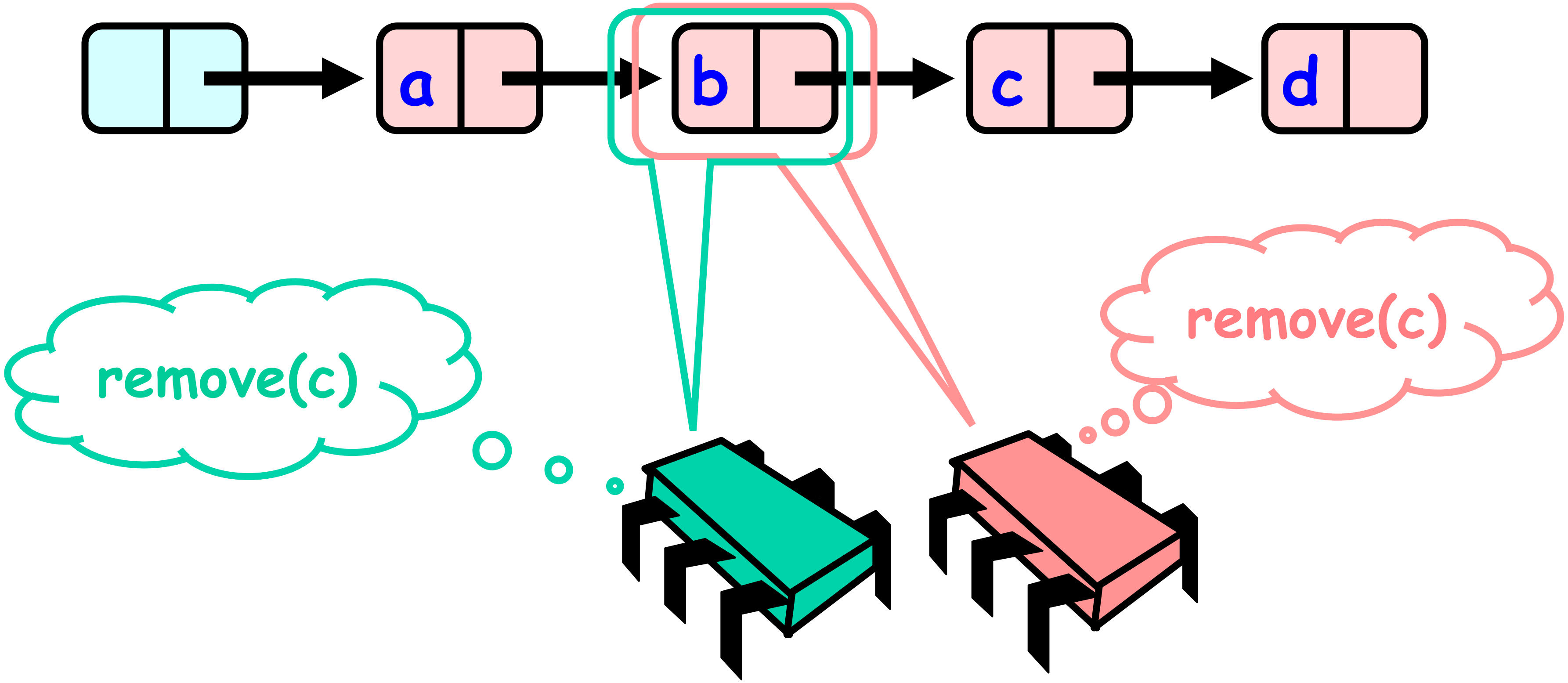
- ❑ Scan list from left to right, apply operation `at the right place’
- ❑ Not so simple…





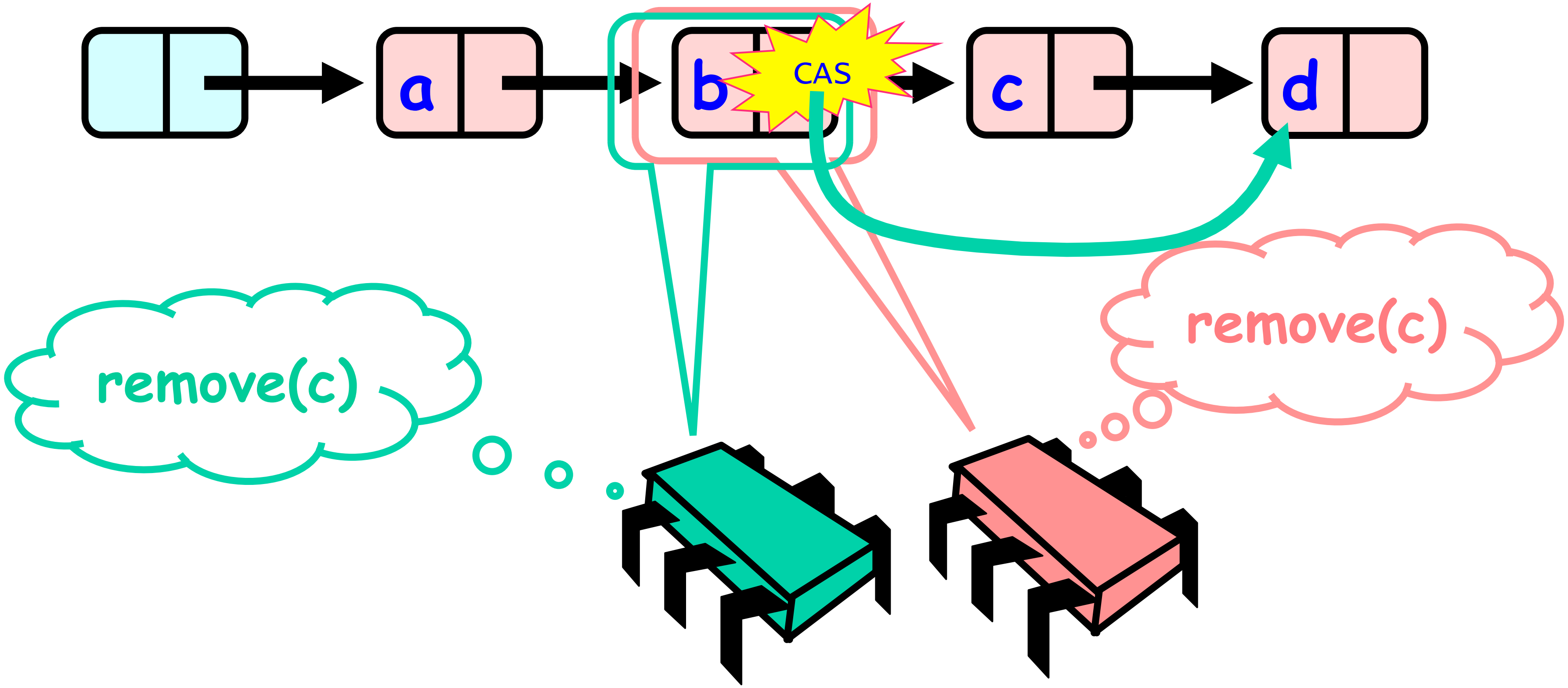
Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Use compare-and-swap (CAS)!



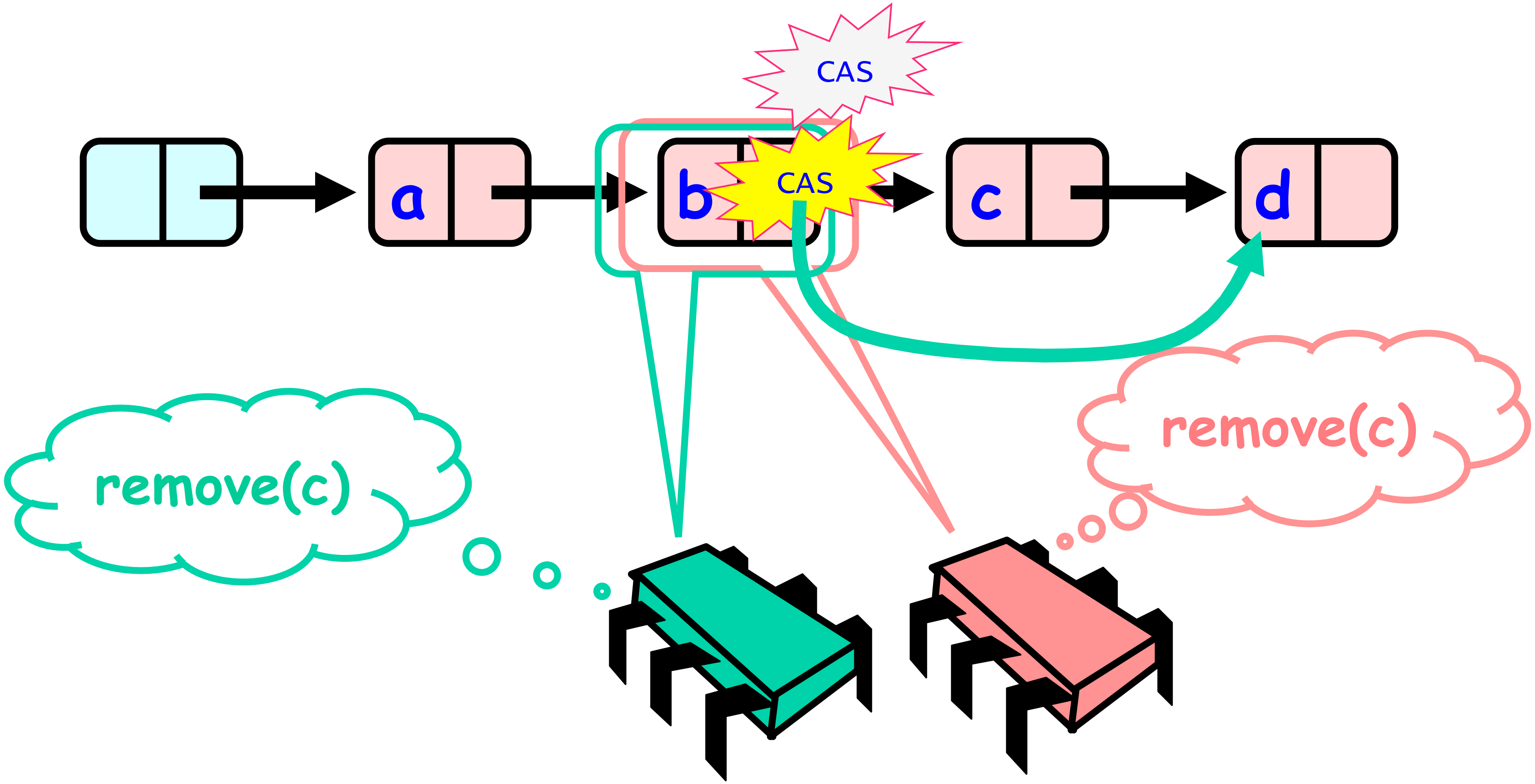
Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

The List-Based Set
Use compare-and-swap (CAS)!



Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

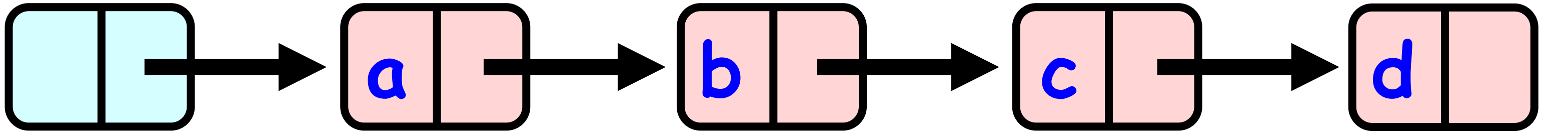
The List-Based Set
Use compare-and-swap (CAS)!



Danny Hendler, SPTCC summer school, Saint-Petersburg, 2017

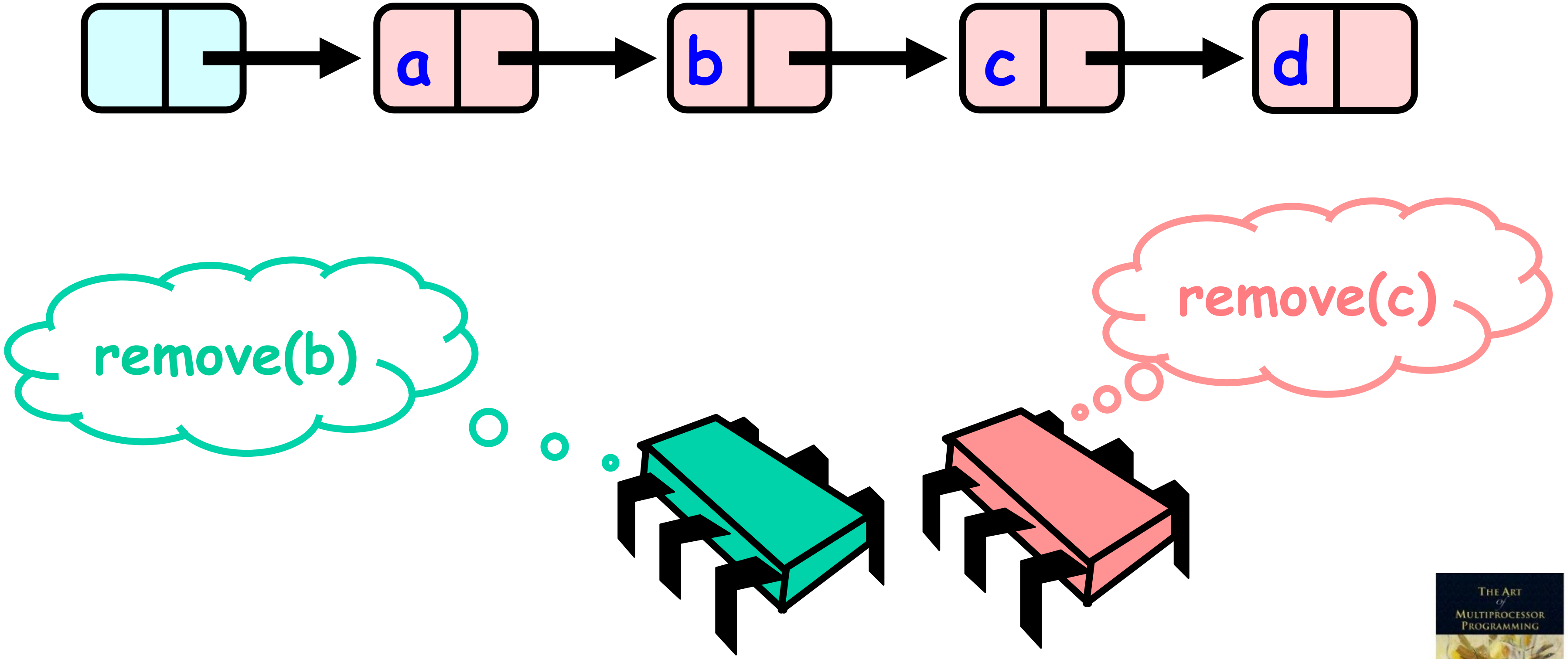
The List-Based Set
Why synchronization is required (2)

- ❑ Apply operation `at the right place’ **using CAS**
- ❑ Not so simple…



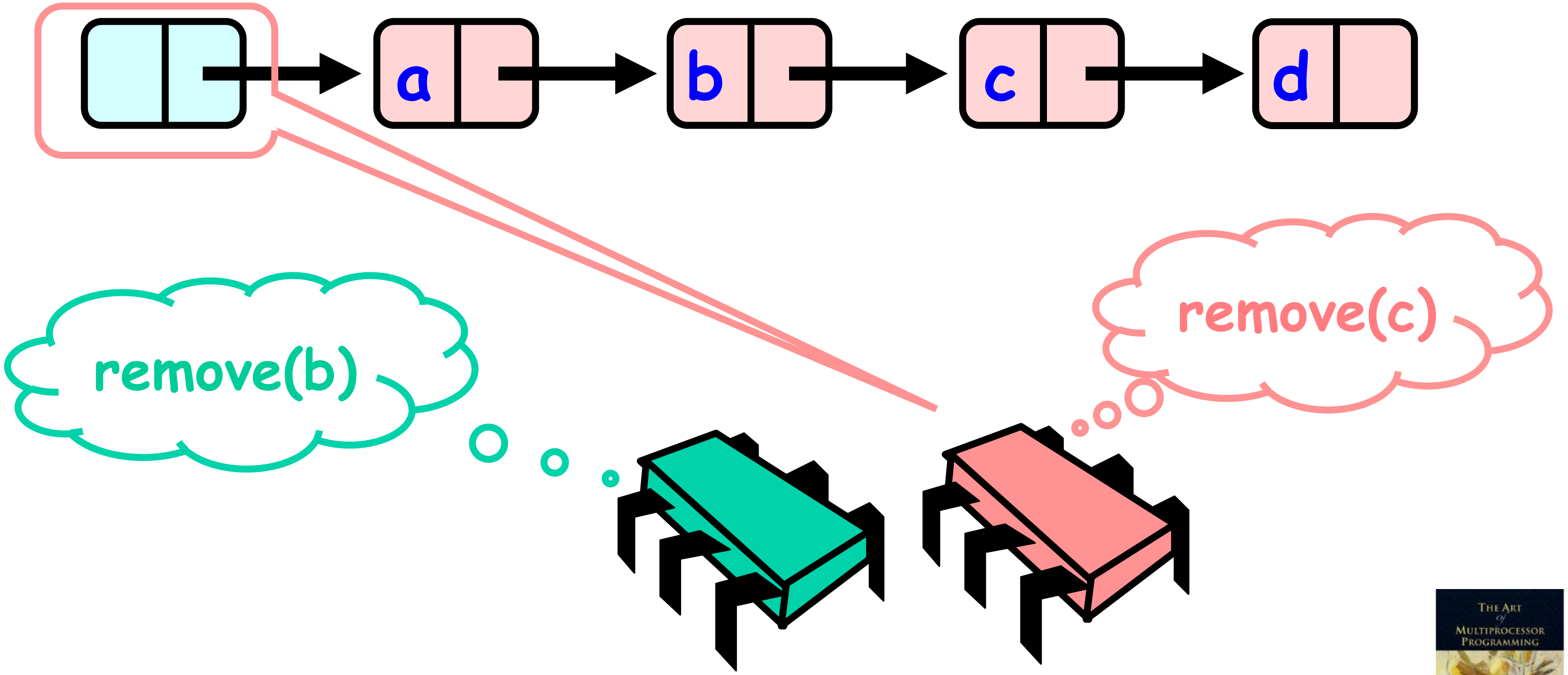
The List-Based Set
Why synchronization is required (2)

- Apply operation `at the right place’ **using CAS**
- Not so simple…



The List-Based Set
Why synchronization is required (2)

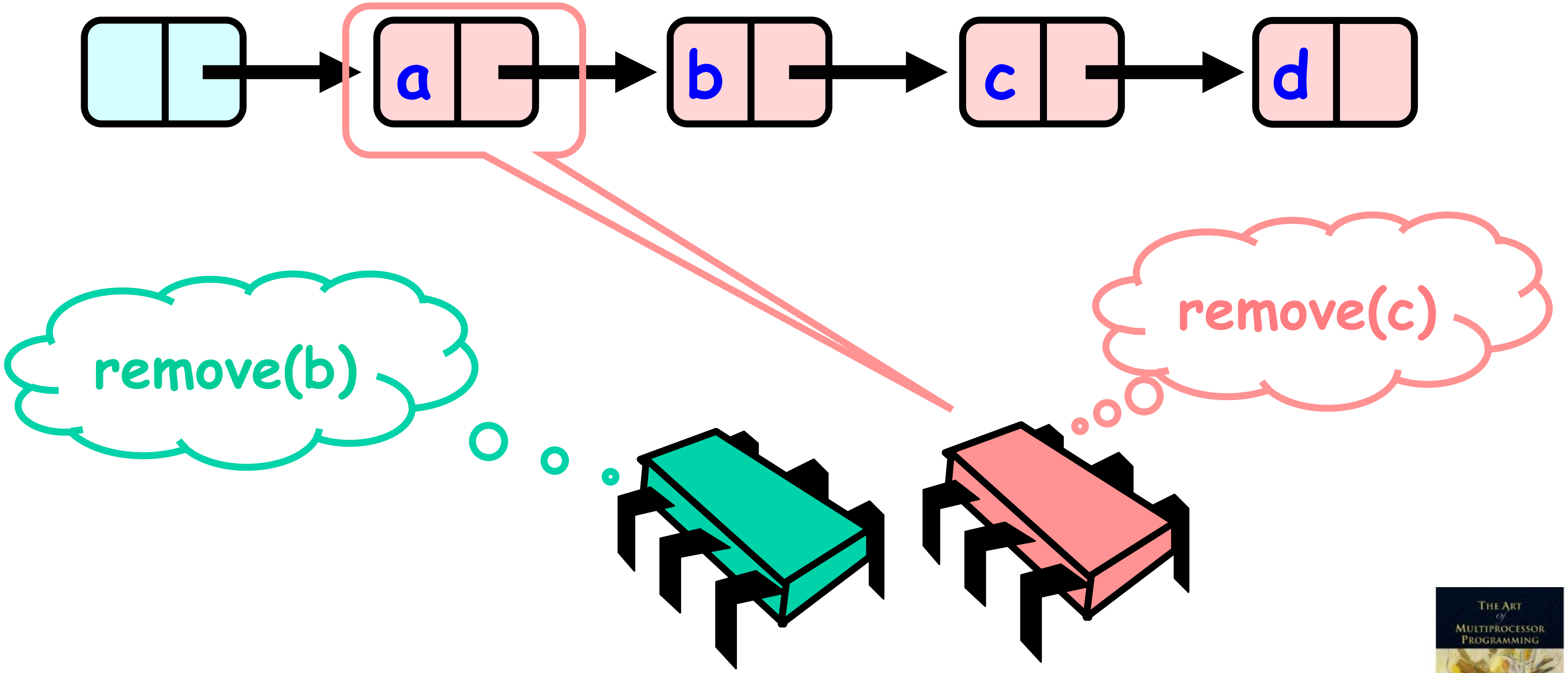
- ❑ Apply operation `at the right place’ **using CAS**
- ❑ Not so simple…



The List-Based Set
Why synchronization is required (2)

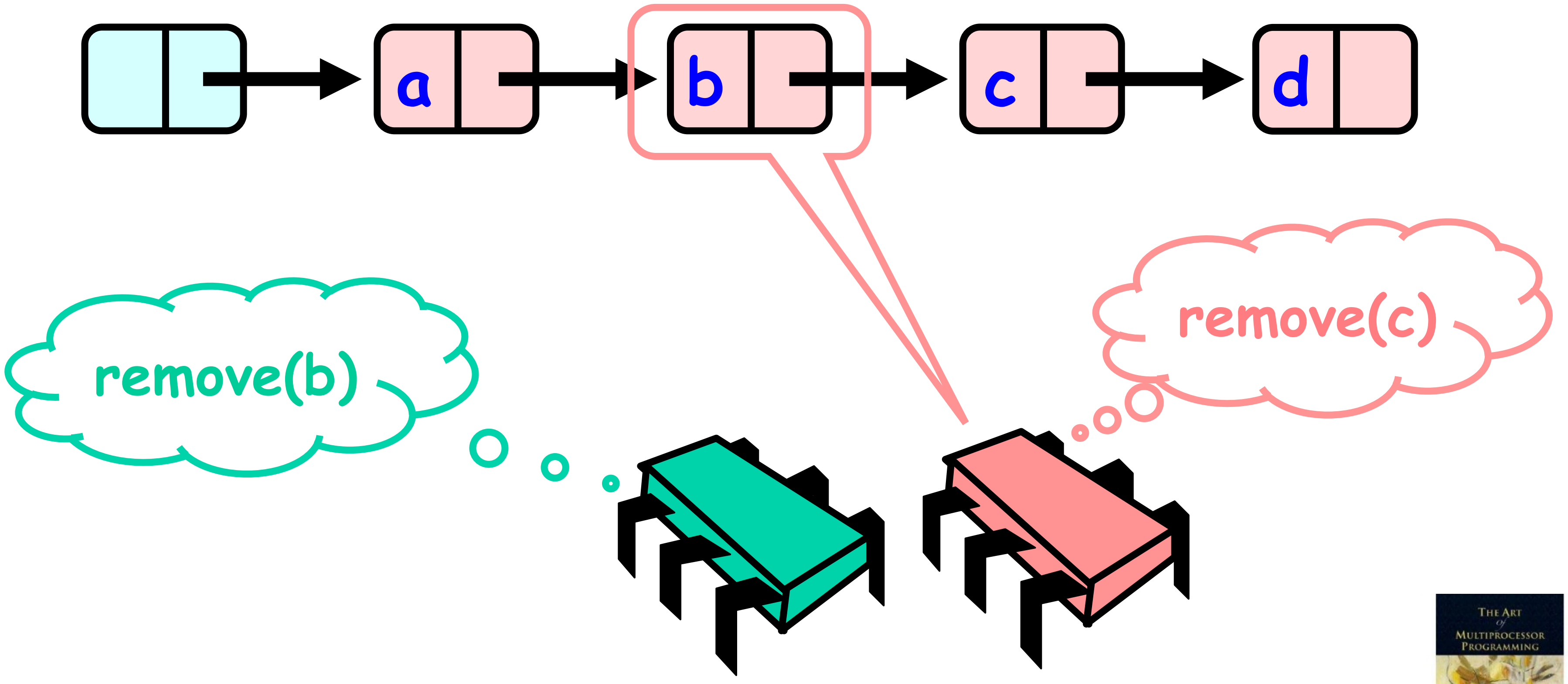
❑ Apply operation `at the right place’ **using CAS**

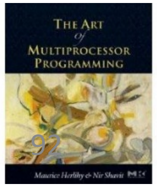
❑ Not so simple…



The List-Based Set
Why synchronization is required (2)

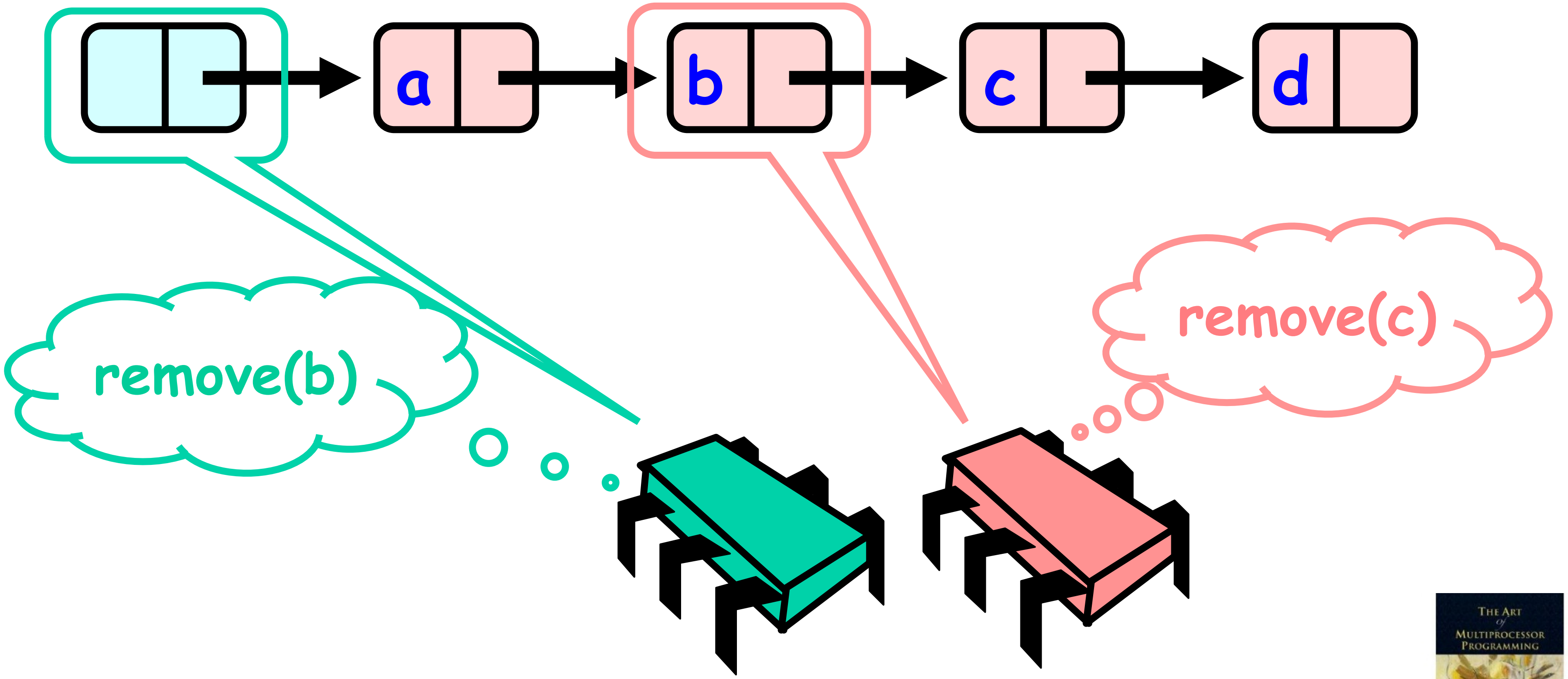
- ❑ Apply operation `at the right place’ **using CAS**
- ❑ Not so simple…

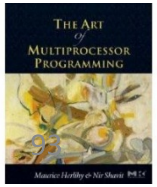




The List-Based Set
Why synchronization is required (2)

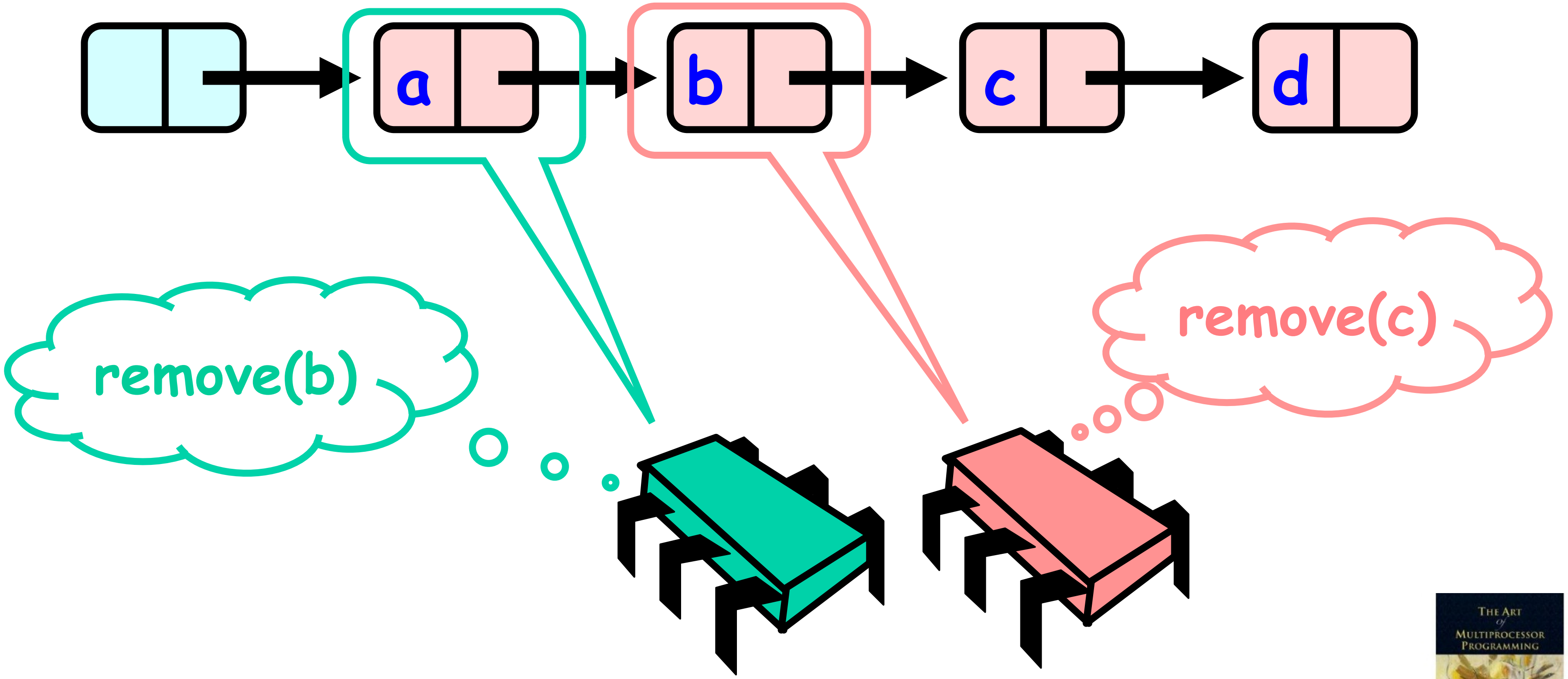
- ❑ Apply operation `at the right place' **using CAS**
- ❑ Not so simple…

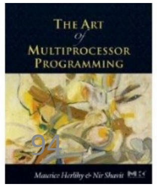




The List-Based Set
Why synchronization is required (2)

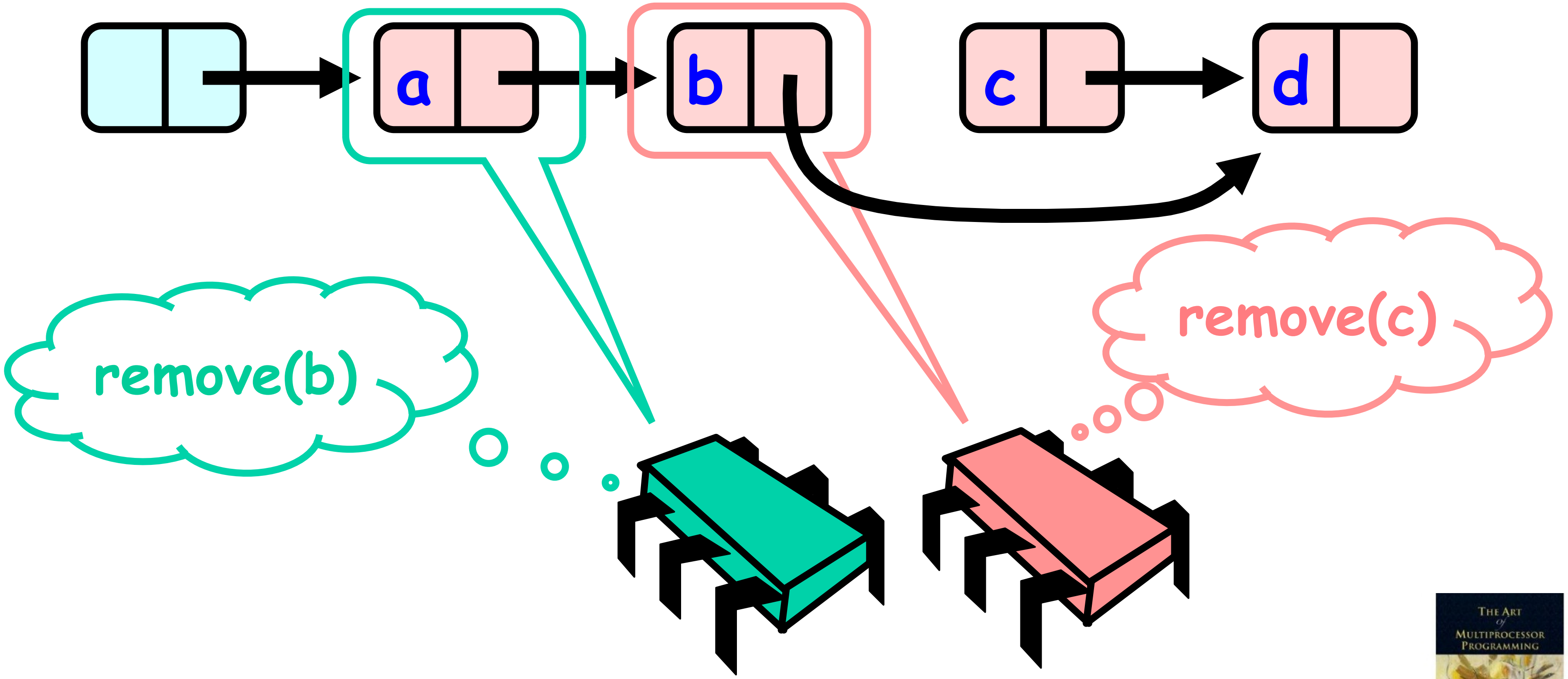
- ❑ Apply operation `at the right place’ **using CAS**
- ❑ Not so simple…

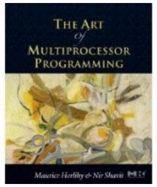




The List-Based Set
Why synchronization is required (2)

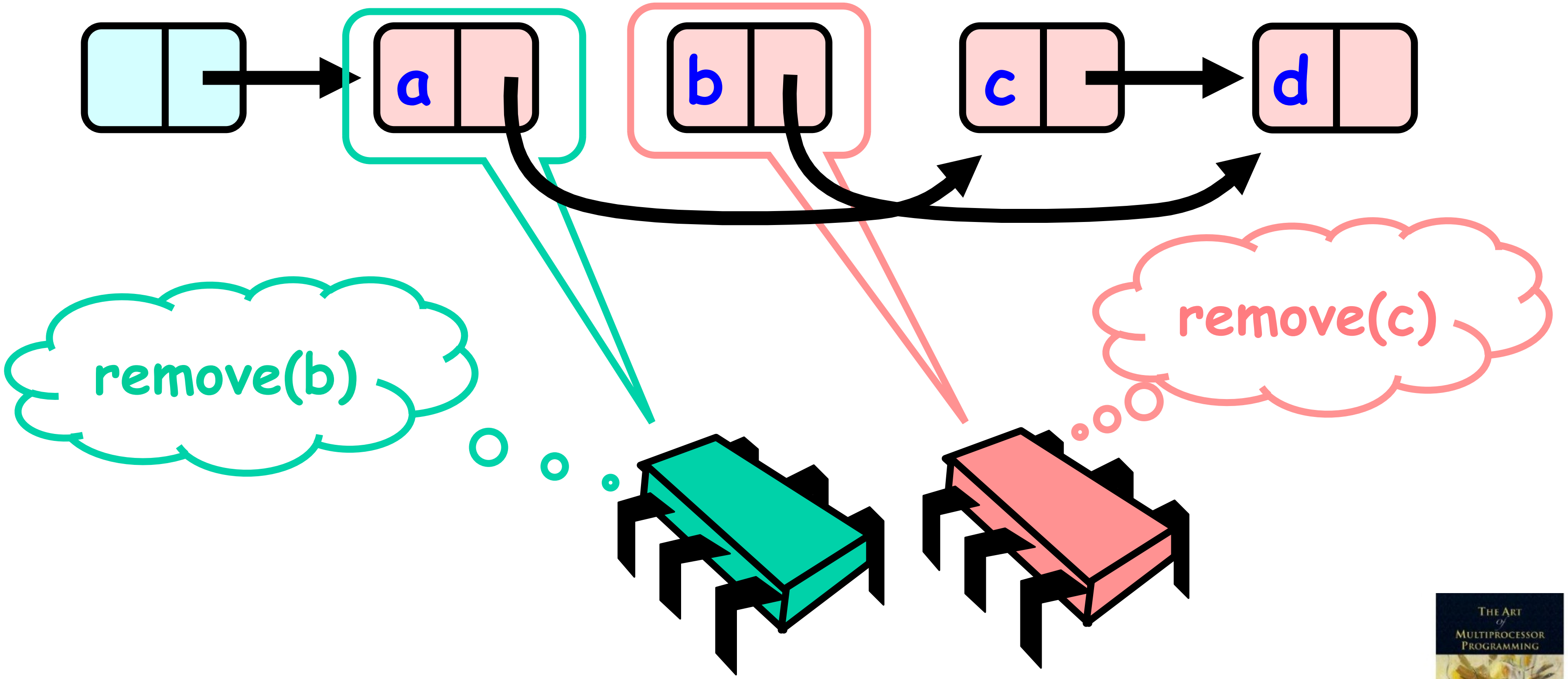
- ❑ Apply operation `at the right place' **using CAS**
- ❑ Not so simple…

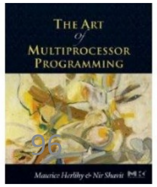




The List-Based Set
Why synchronization is required (2)

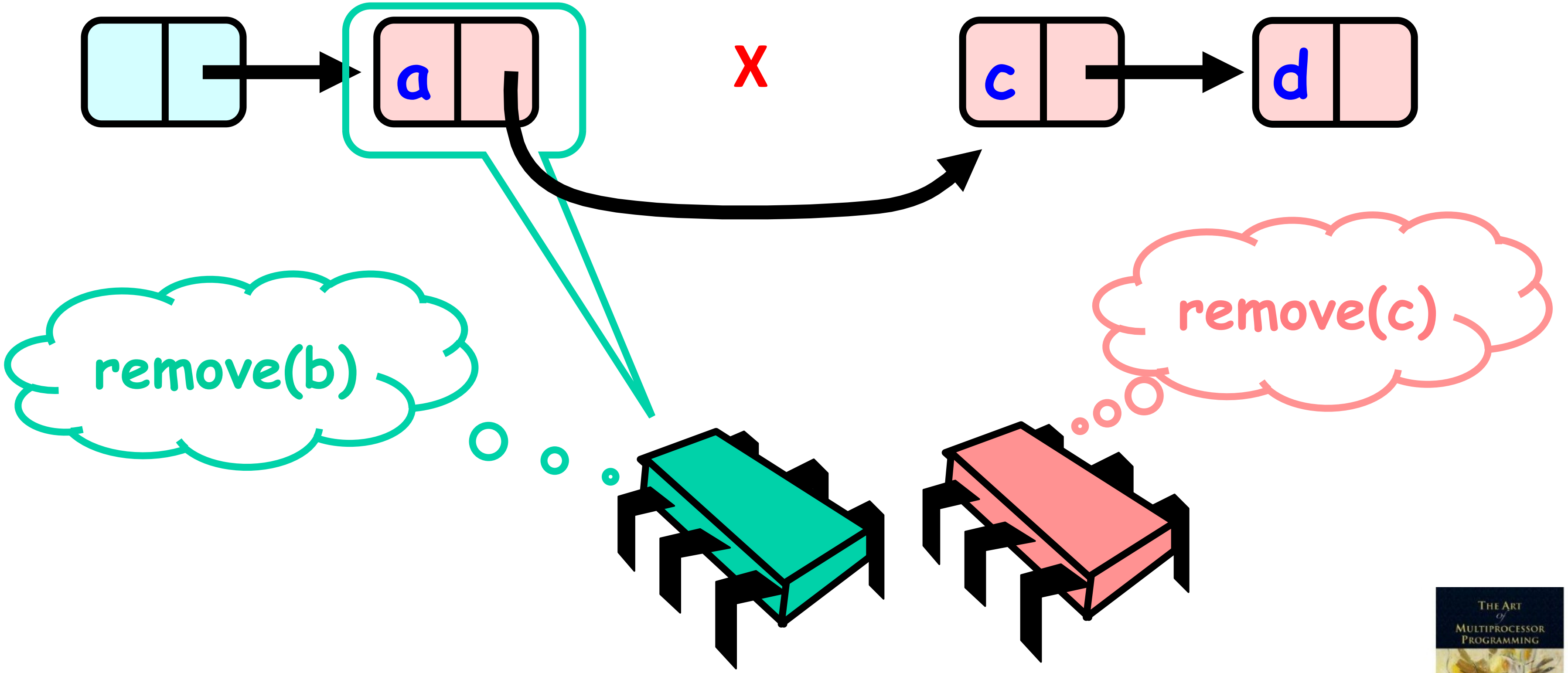
- ❑ Apply operation `at the right place’ **using CAS**
- ❑ Not so simple…




The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

The List-Based Set
Why synchronization is required (2)

- ❑ Apply operation `at the right place’ **using CAS**
- ❑ Not so simple…



The List-Based Set
Logical remove, then physical remove

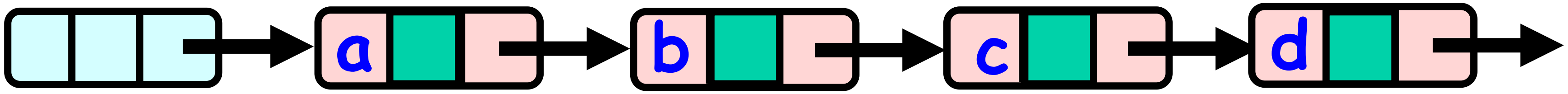
❑ Scan list from left to right

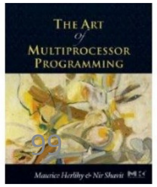
❑ Apply modifications using CAS

❑ Separate removal to two steps

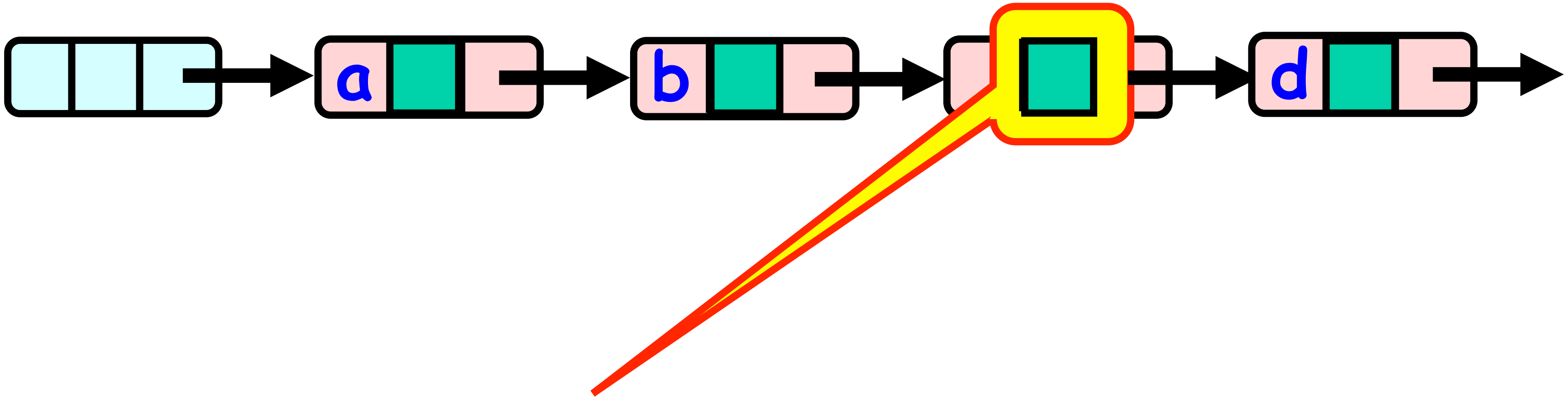
- Logical removal: mark node to be deleted
- Physical removal: change predecessor's *next* reference

The List-Based Set
Logical remove, then physical remove

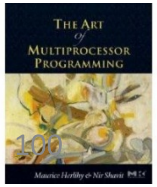



The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

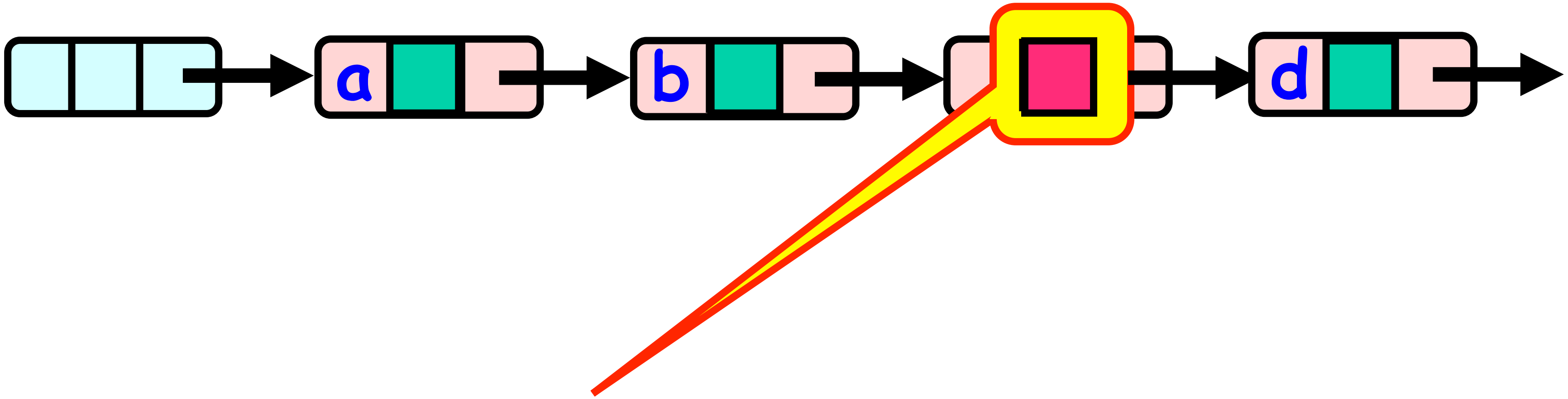
The List-Based Set
Logical remove, then physical remove



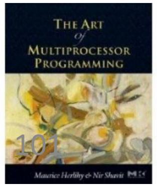
Present in list


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

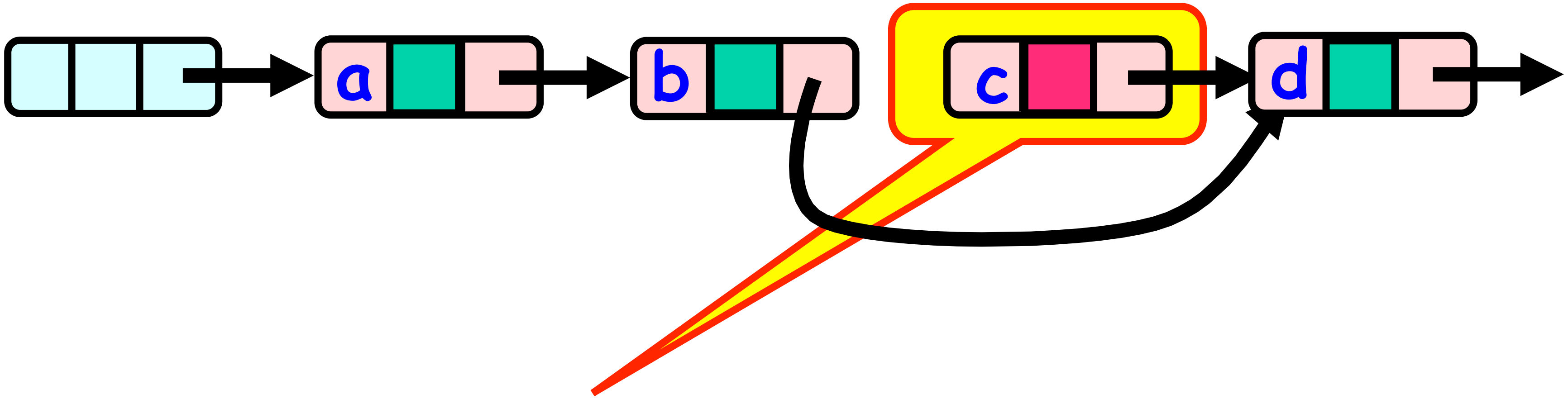
The List-Based Set
Logical remove, then physical remove



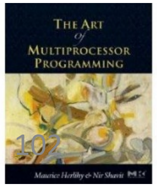
Logically deleted



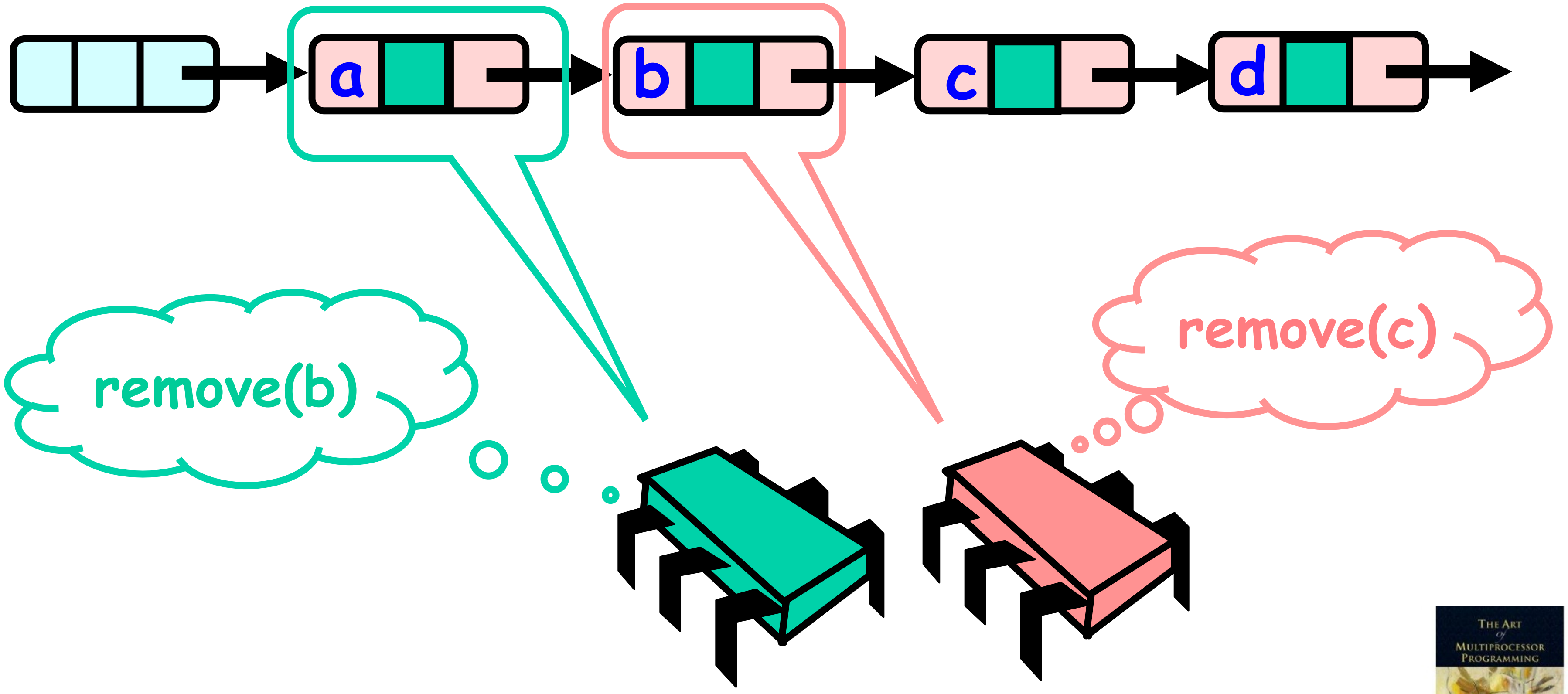
The List-Based Set
Logical remove, then physical remove



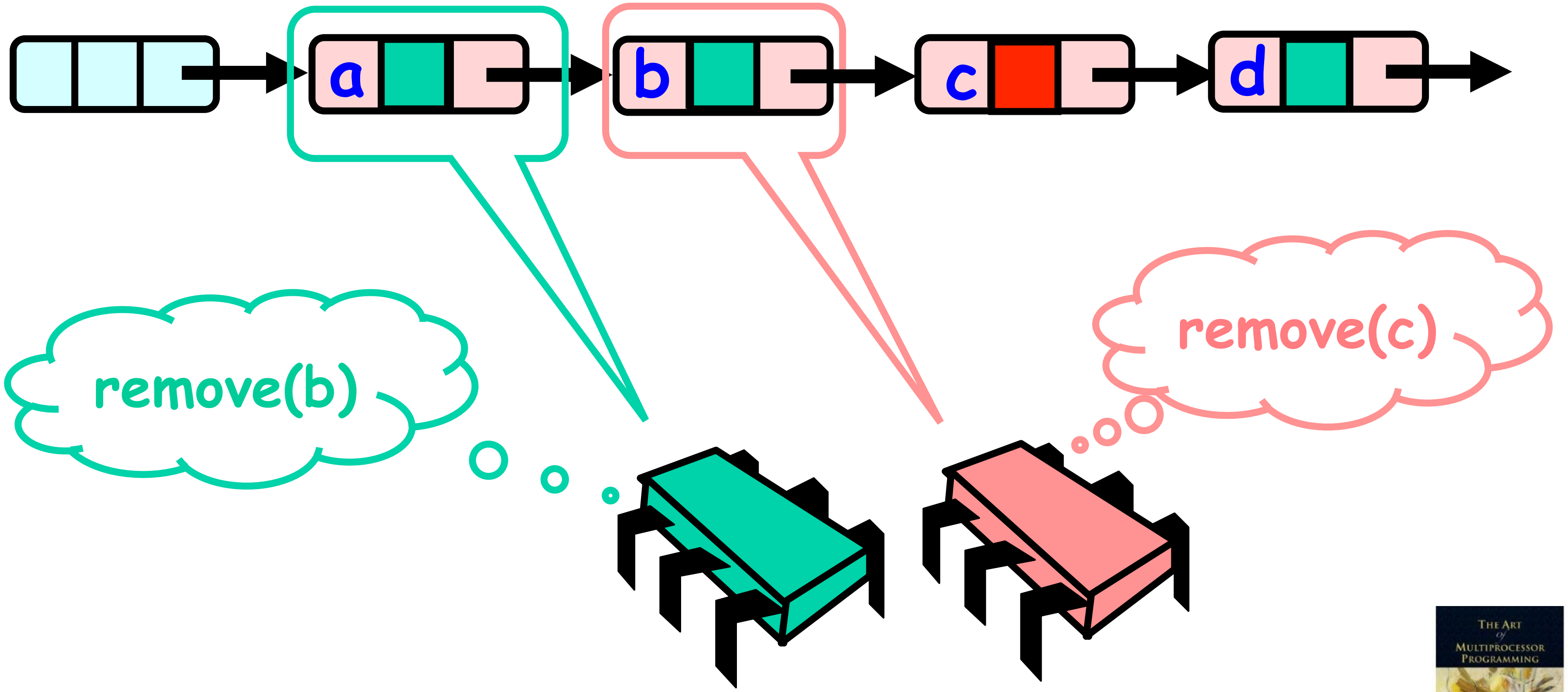
Physically deleted

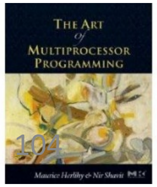


The List-Based Set
Logical remove, then physical remove

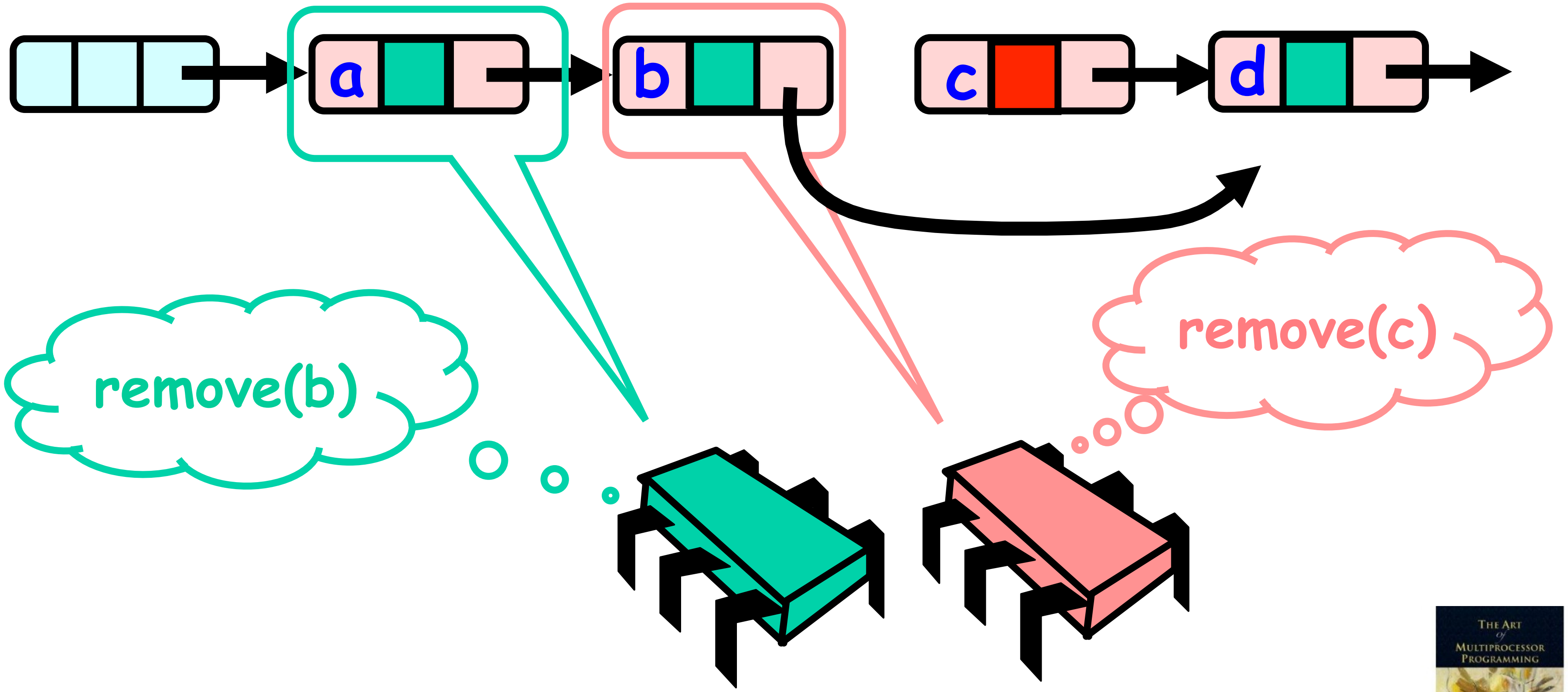


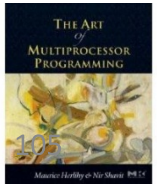
The List-Based Set
Logical remove, then physical remove



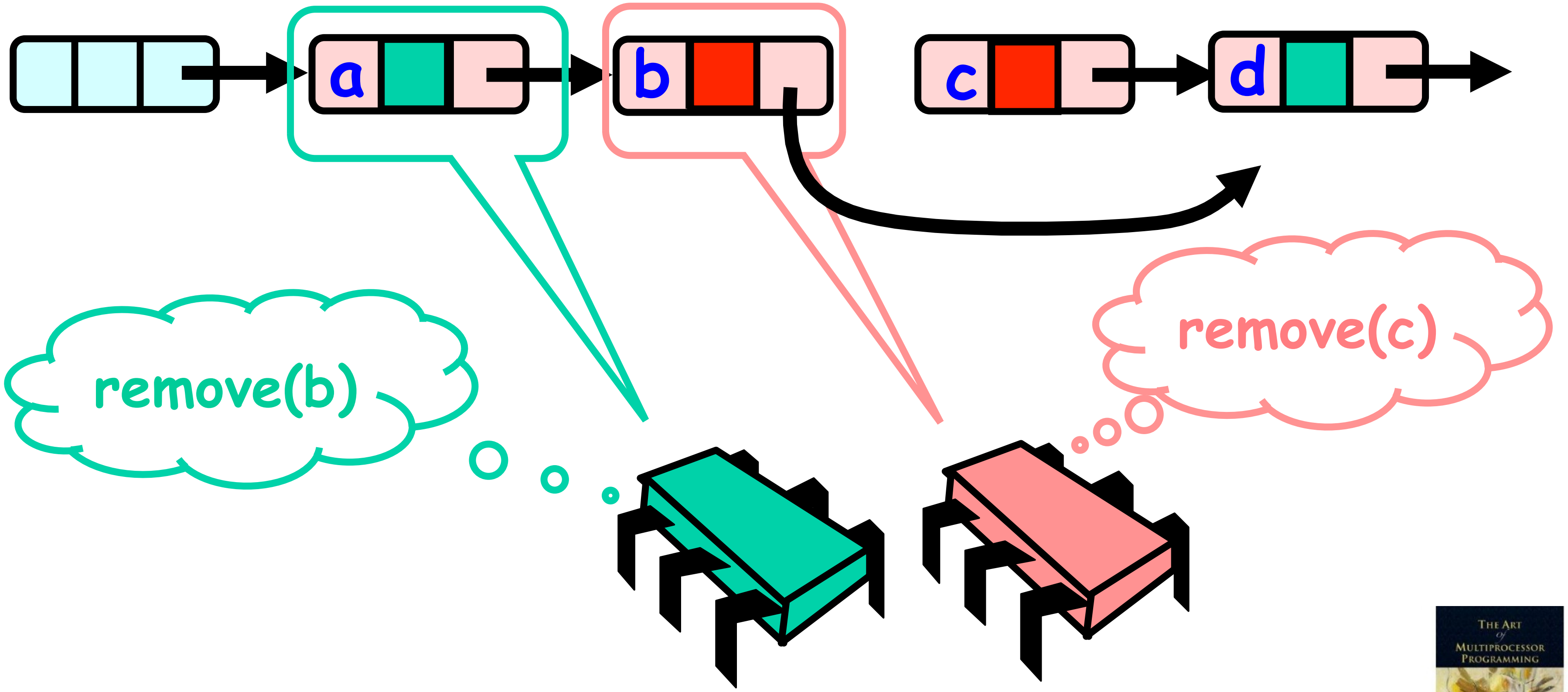


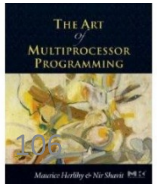
The List-Based Set
Logical remove, then physical remove



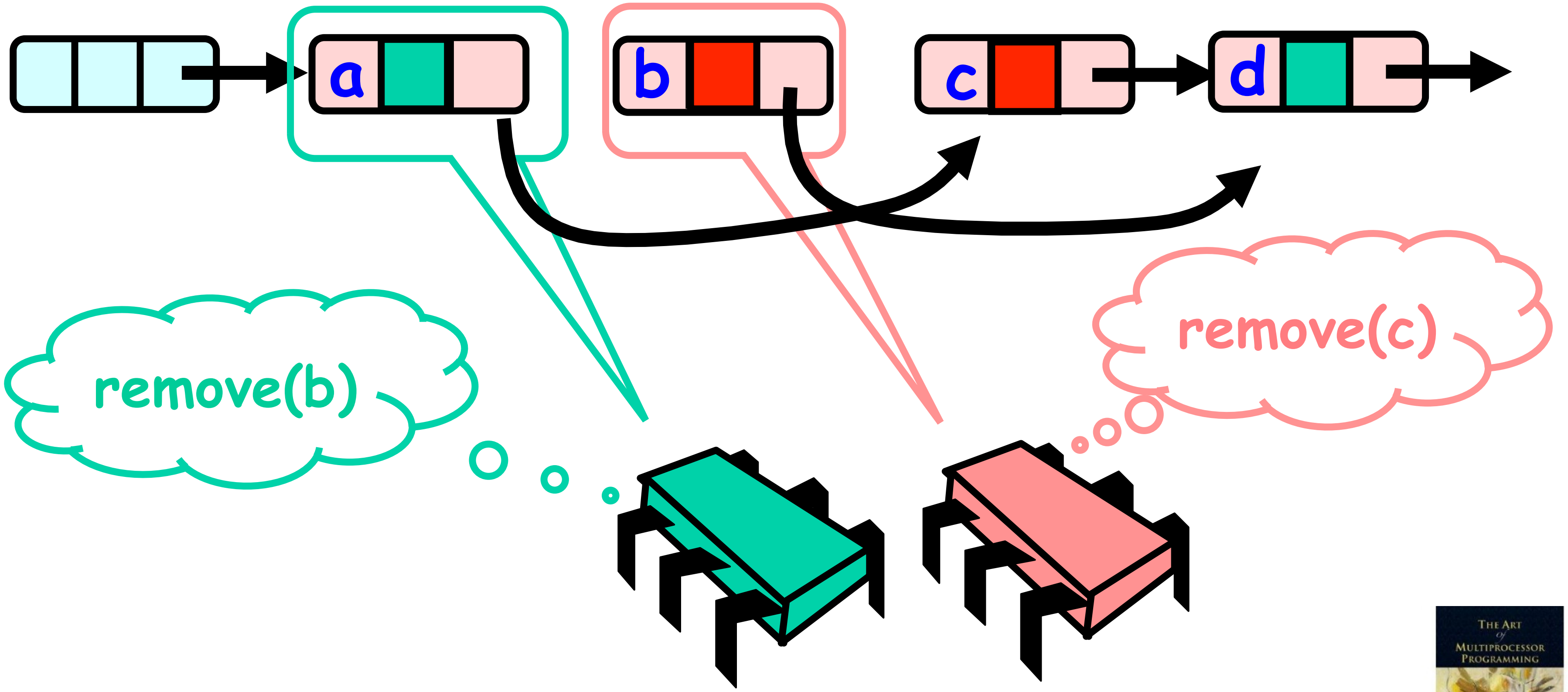


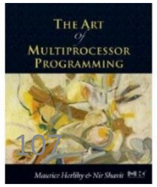
The List-Based Set
Logical remove, then physical remove



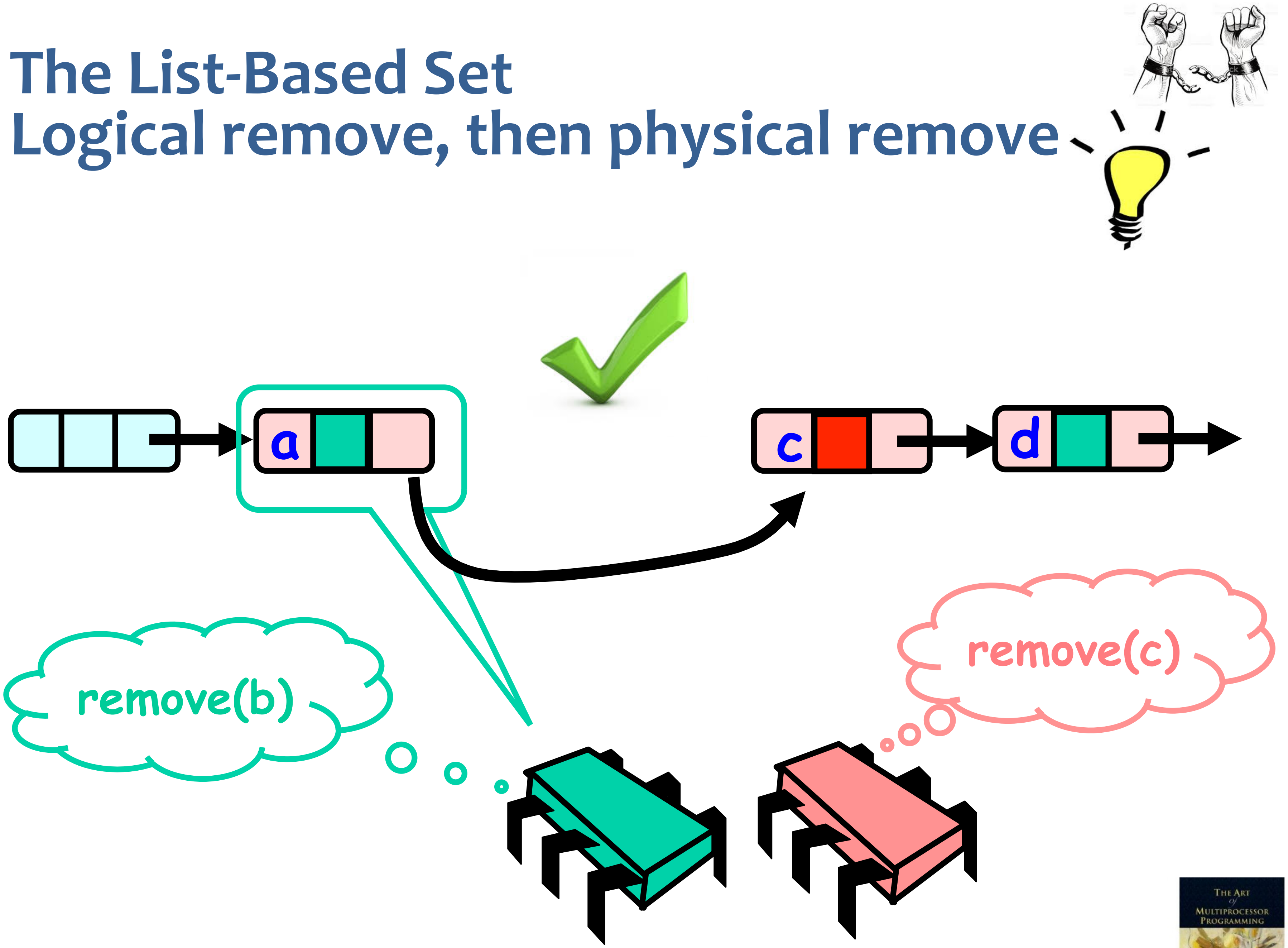


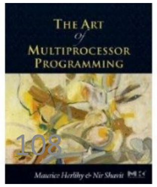
The List-Based Set
Logical remove, then physical remove





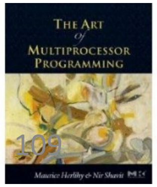
The List-Based Set
Logical remove, then physical remove





The List-Based Set
Logical remove, then physical remove

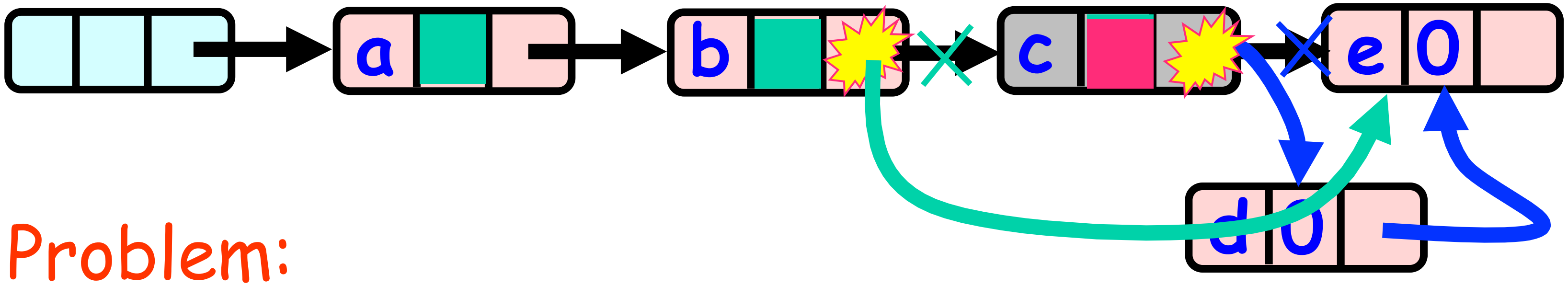
Still not enough!


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

The List-Based Set
Logical remove, then physical remove

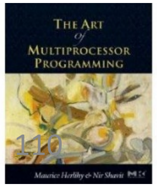
Still not enough!

Logical Removal =
Set Mark Bit



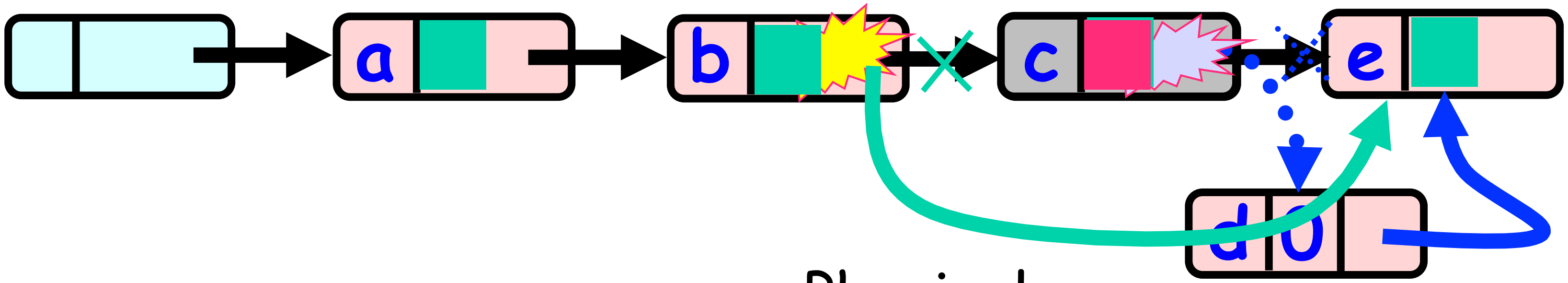
Problem:
d not added to list…
Must Prevent
manipulation of
removed node's pointer

Node added
Before
Physical
Removal CAS



AtomicMarkableRereference Combine bit and pointer (Harris)

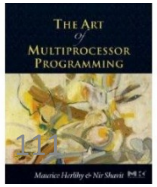
Logical Removal = Set Mark Bit



Physical Removal CAS

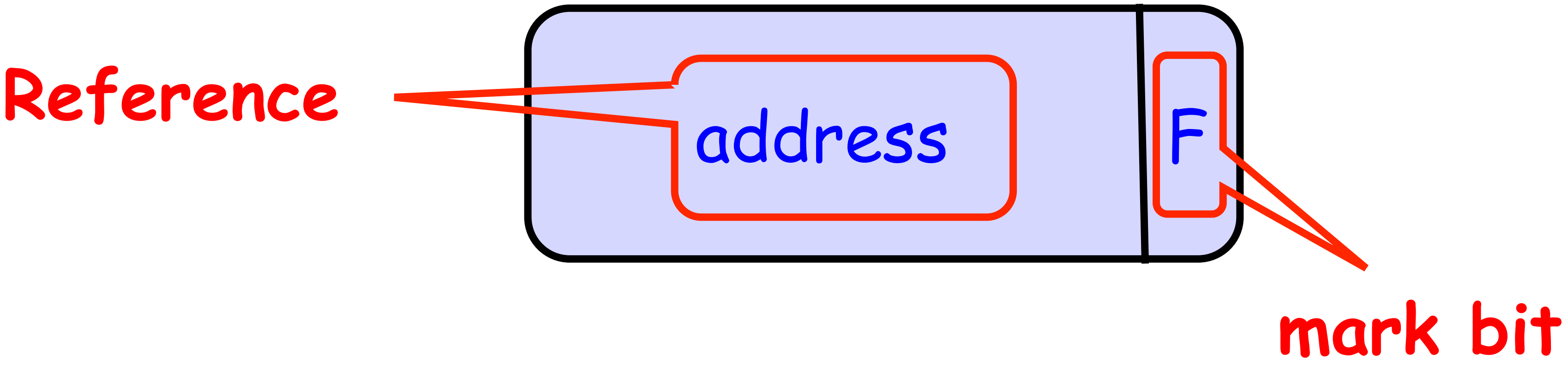
Fail CAS: Node not added after logical Removal

Mark-Bit and Pointer are CASed together (AtomicMarkableReference)


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

AtomicMarkableRereference
Marking a node

- AtomicMarkableReference class
  - Java.util.concurrent.atomic package

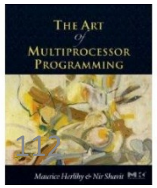


Reference

address

F

mark bit

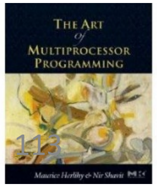


AtomicMarkableReference
Extracting reference & mark

```
Public Object get(boolean[] marked);
```

Returns reference

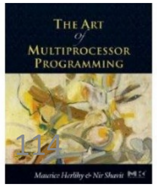
Returns mark at array index 0!



AtomicMarkableReference Extracting reference only

```
public object getReference();
```

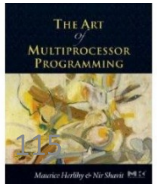
Value of reference



AtomicMarkableReference Extracting mark only

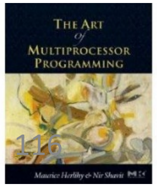
```
public boolean isMarked();
```

Value of mark



AtomicMarkableReference
Changing state

```
Public boolean compareAndSet(
  Object expectedRef,
  Object updateRef,
  boolean expectedMark,
  boolean updateMark);
```

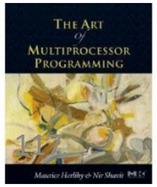

The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

AtomicMarkableReference
Changing state

If this is the current reference …

```
Public boolean compareAndSet(
   Object expectedRef,
   Object updateRef,
   boolean expectedMark,
   boolean updateMark);
```

And this is the current mark …

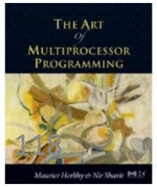

The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

AtomicMarkableReference
Changing state

…then change to this new reference …

```
Public boolean compareAndSet(
  Object expectedRef,
  Object updateRef,
  boolean expectedMark,
  boolean updateMark);
```

… and this new mark


THE ART of MULTIPROCESSOR PROGRAMMING
Maurice Herlihy & Nir Shavit

The List-Based Set
Key ideas

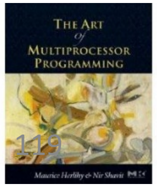
❑ Scan list from left to right

❑ Apply modifications using CAS

❑ Separate removal to two steps

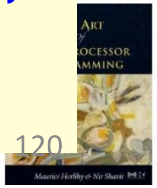
- – Logical removal: mark node to be deleted
  - <u>Once done, *next* reference cannot be changed</u>
- – Physical removal: change predecessor's *next* reference

❑ When finding a logically-deleted node, finish the job


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Remove pseudo-code

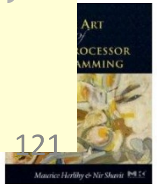
```
public boolean remove(T item) {
Boolean snip;
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
  Node succ = curr.next.getReference();
  snip = curr.next.compareAndSet(succ, succ, false
true);
  if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



Remove pseudo-code

```
public boolean remove(T item) {
Boolean snip;
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.compareAndSet (succ, succ, false,
true);
  if (!snip) continue;
  pred.next.compareAndSet(curr, succ, false, false);
  return true;
}}}
```

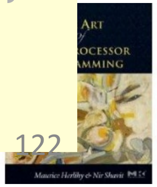
Keep trying



Remove pseudo-code

```
public boolean remove(T item) {
Boolean snip;
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.compareAndSet (succ, succ, false, 
true);
  if (!snip) continue;
  pred.next.compareAndSet(curr, succ, false, false);
  return true;
}}}
```

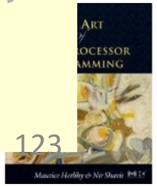
Find neighbors



Remove pseudo-code

```
public boolean remove(T item) {
Boolean snip;
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.compareAndSet(succ, succ, false,
true);
  if (!snip) continue;
  pred.next.compareAndSet(curr, succ, false, false);
  return true;
}}}
```

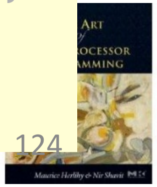
She's not there …



Remove pseudo-code

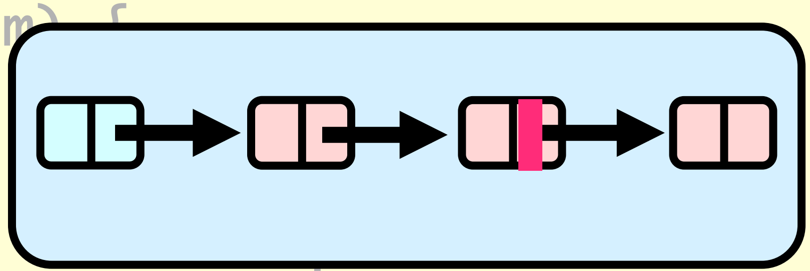
```
public boolean remove(T item) {
Boolean snip;
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
  Node succ = curr.next.getReference();
  snip = curr.next.compareAndSet(succ, succ, false,
true);
 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

Try to mark node as deleted

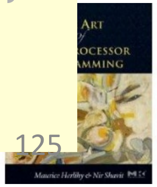


Remove pseudo-code

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

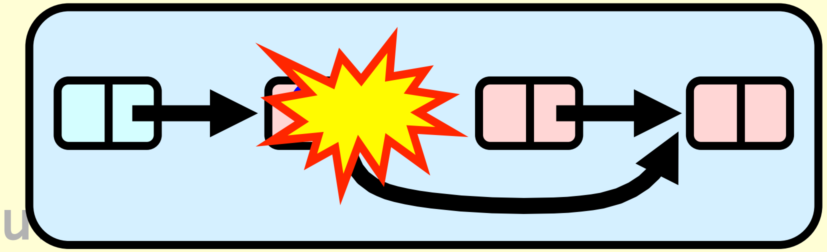


If it doesn't work, just retry, if it does, job essentially done



Remove pseudo-code

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

Remove linearization points

```
public boolean remove(T item) {
Boolean snip;
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.compareAndSet(succ, succ, false
true);
  if (!snip) continue;
   pred.next.compareAndSet(curr, succ, false,
false);
     return true;
}}}
```

Remove linearization points

Upon success

```
public boolean remove(T item) {
Boolean snip;
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.compareAndSet(succ, succ, false
true);
  if (!snip) continue;
   pred.next.compareAndSet(curr, succ, false,
false);
     return true;
}}}
```

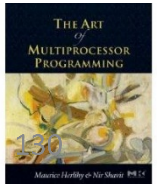
Remove linearization points

```
public boolean remove(T item) {
Boolean snip;
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.compareAndSet(succ, succ, false
true);
  if (!snip) continue;
   pred.next.compareAndSet(curr, succ, false,
false);
     return true;
}}}
```

When
returning
false

Add pseudo-code

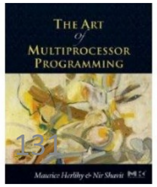
```
public boolean add(T item) {
 boolean splice;
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred, curr = window.curr;
   if (curr.key == key) {
      return false;
   } else {
   Node node = new Node(item);
   node.next = new AtomicMarkableRef(curr, false);
   if (pred.next.compareAndSet(curr,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


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Add pseudo-code

```
public boolean add(T item) {
 boolean splice;
 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;}
}}}
```

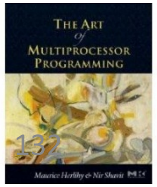
Keep trying



Add pseudo-code

```
public boolean add(T item) {
 boolean splice;
 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;}
}}}
```

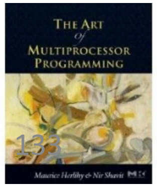
Find neighbors



Add pseudo-code

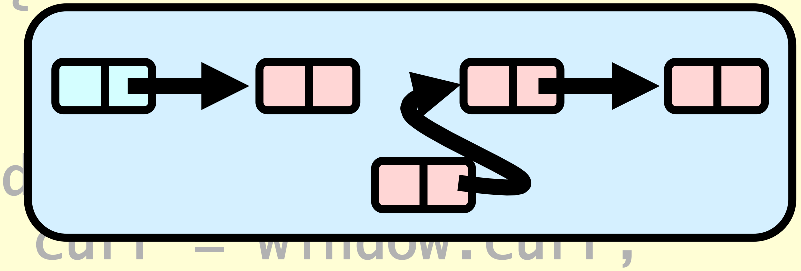
```
public boolean add(T item) {
 boolean splice;
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred, curr = window.curr;
   if (curr.key == key) {
      return false;
   } else {
   Node node = new Node(item);
   node.next = new AtomicMarkableRef(curr, false);
   if (pred.next.compareAndSet(curr,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

Item already there.

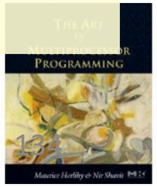


Add pseudo-code

```
public boolean add(T item) {
 boolean splice;
 while (true) {
   Window window = find(head
   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;}
}}}
```

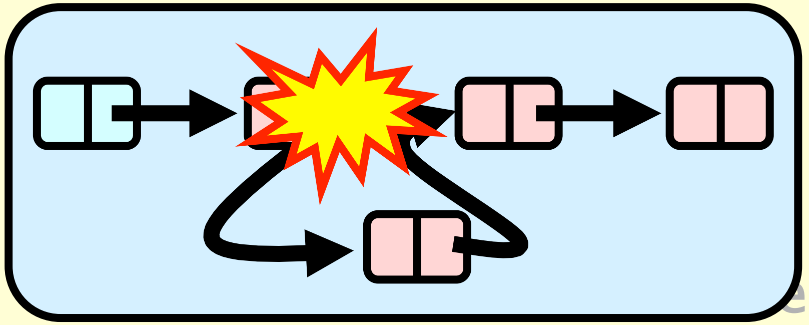


create new node

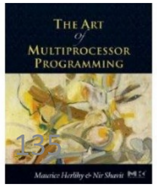


Add pseudo-code

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



Install new node, else retry loop



Add linearization points

```
public boolean add(T item) {
 boolean splice;
 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 linearization points

Upon success

```
public boolean add(T item) {
 boolean splice;
 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;}
}}}
```

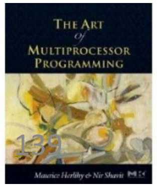
Add linearization points

```
public boolean add(T item) {
 boolean splice;
 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;}
}}}
```

When returning false

Contains pseudo-code

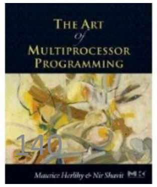
```
public boolean contains(T item) {
    boolean marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
  }
```


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Contains pseudo-code

```
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])
  }
```

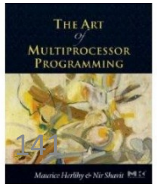
Start at the head



Contains pseudo-code

```
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])
  }
```

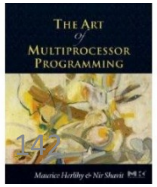
Search key range



Contains pseudo-code

```
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])
  }
```

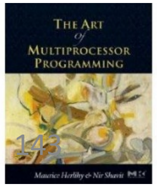
Traverse



Contains pseudo-code

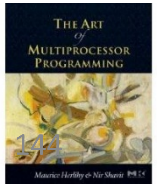
```
public boolean contains(T item) {
    boolean marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
  }
```

Return true if value found in a non-marked node



Contains linearization point

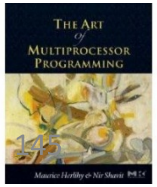
```
public boolean contains(T item) {
    boolean marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
  }
```


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Contains linearization point

When returning true

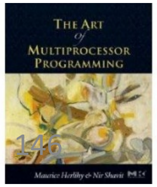
```
public boolean contains(T item) {
    boolean marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
  }
```



Contains linearization point

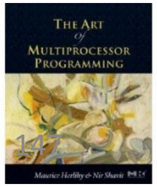
```
public boolean contains(T item) {
    boolean marked;
    int key = item.hashCode();
    Node curr = this.head;
    while (curr.key < key)
      curr = curr.next;
    Node succ = curr.next.get(marked);
    return (curr.key == key && !marked[0])
  }
```

Linearization more complicated when returning false


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Find pseudo-code

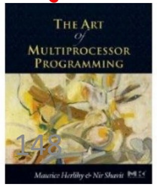
```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
    succ = curr.next.get(marked);
    while (marked[0]) {
    …
    }
    if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



Find pseudo-code

```
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;
     }
}}
```

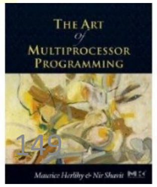
Start search for key
at the head



Find pseudo-code

```
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;
    }
}}
```

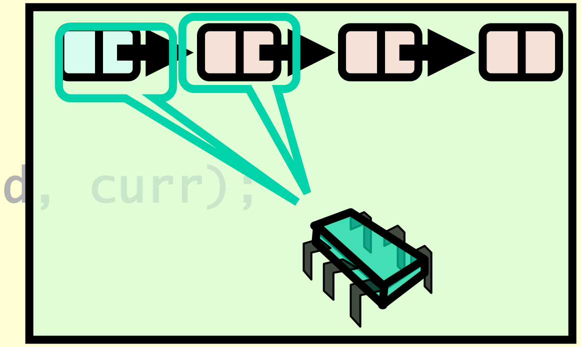
If list changes while traversed, start over. Lock-Free because we start over only if someone else makes progress

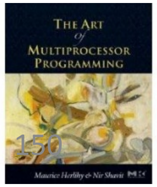


Find pseudo-code

```
public Window find(Node head, int key) {
 Node pred = nul
 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

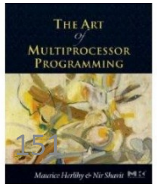




Find pseudo-code

```
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;
    }
}}
```

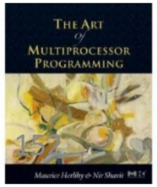
Move down the list



Find pseudo-code

```
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;
    }
}}
```

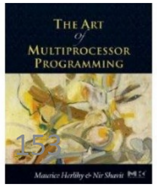
Get ref to successor and
current deleted bit


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Find pseudo-code

```
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;
```

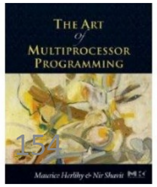
Try to remove deleted nodes in path…code details soon



Find pseudo-code

```
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;
     }
}}
```

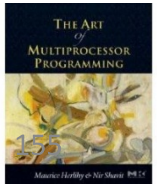
If curr key that is greater or equal, return pred and curr



Find pseudo-code

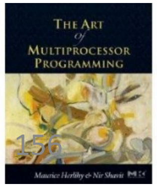
```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
    succ = curr.next.get(marked);
    while (marked[0]) {
     …
    }
   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

Otherwise advance window and loop again



Find pseudo-code

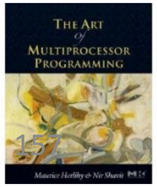
```
retry: while (true) {
   …
   while (marked[0]) {
     snip = pred.next.compareAndSet(curr,
                                    succ, false, false);
     if (!snip) continue retry;
     curr = succ;
     succ = curr.next.get(marked);
   }
…
```



Find pseudo-code

If current node is marked

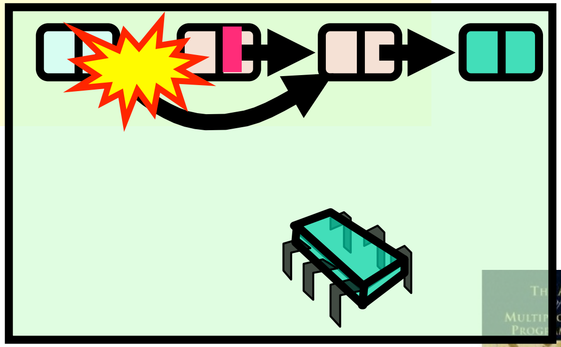
```
retry: while (true) {
   …
   while (marked[0]) {
      snip = pred.next.compareAndSet(curr,
                                     succ, false, false);
      if (!snip) continue retry;
      curr = succ;
      succ = curr.next.get(marked);
   }
…
```


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Find pseudo-code

Try to snip out node

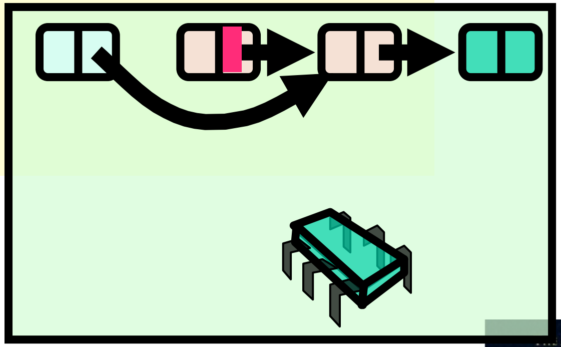
```
retry: while (true) {
   …
   while (marked[0]) {
     snip = pred.next.compareAndSet(curr,
                                    succ, false, false);
     if (!snip) continue retry;
     curr = succ;
     succ = curr.next.get(marked);
   }
…
```

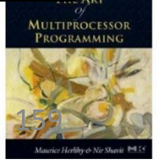


Find pseudo-code

if predecessor's next field changed, retry whole traversal

```
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);
   }
…
```

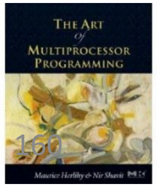




Find pseudo-code

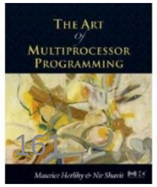
Otherwise move on to check if next node deleted

```
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);
   }
…
```



Find linearization points

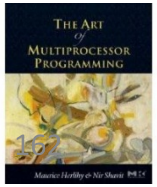
```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 retry: while (true) {
   pred = head;
   curr = pred.next.getReference();
   while (true) {
    succ = curr.next.get(marked);
    while (marked[0]) {
    …
    }
    if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


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Find linearization points

```
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;
     }
}}
```

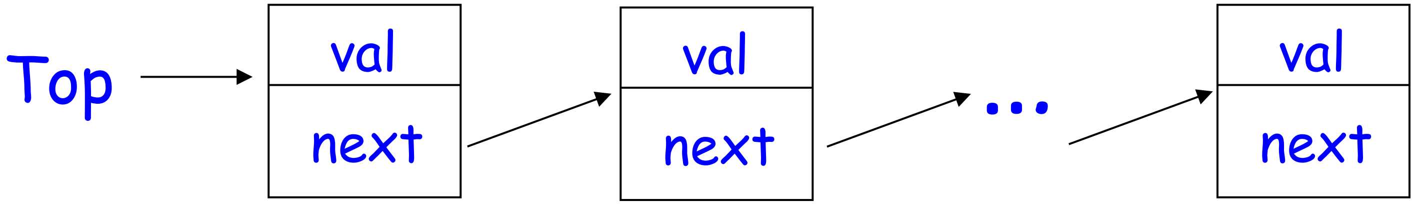
Last read of non-marked node

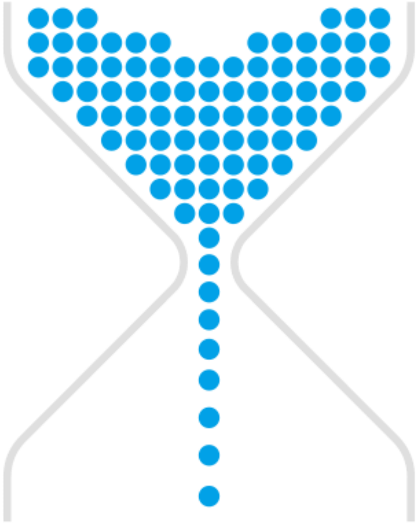


Talk Outline

- Preliminaries
- A simple lock-free stack algorithm
    - Linearizability
- Michael & Scott queue algorithm
- The Harris-Michael linked list algorithm
- Elimination-based stack
- Discussion & conclusions

IBM/Treiber algorithm's disadvantage





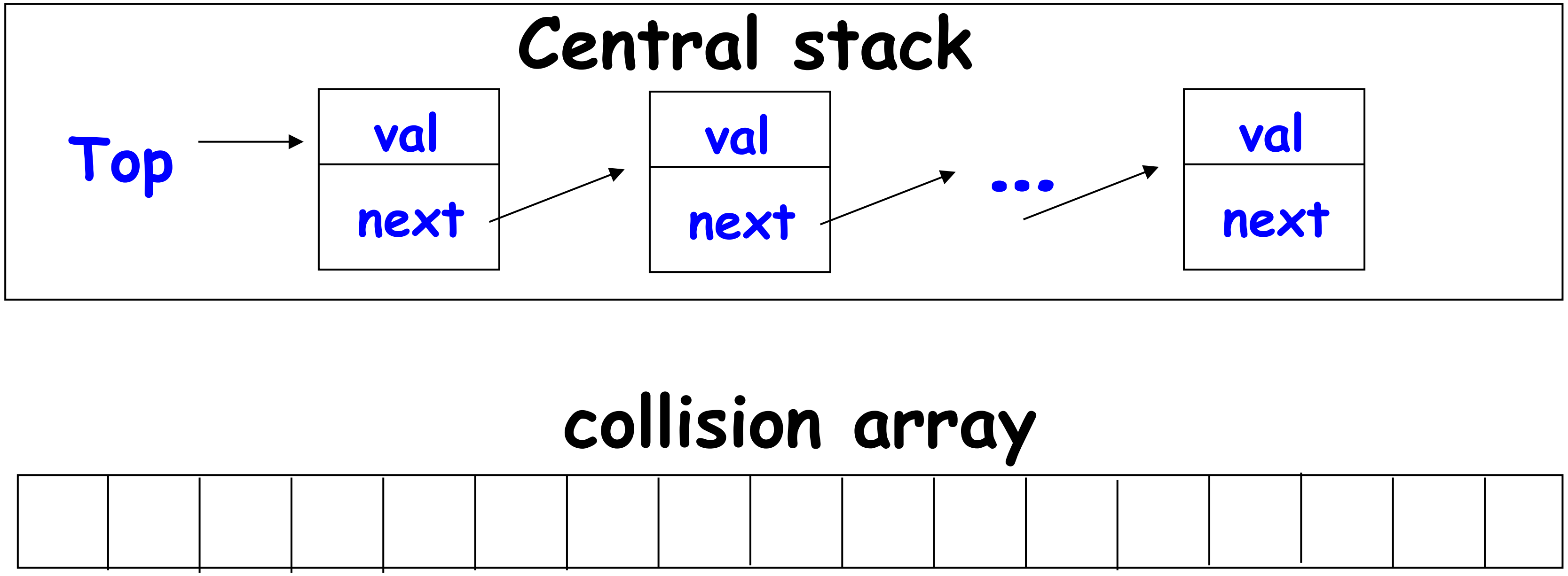
Has a sequential bottleneck

Is this inherent?

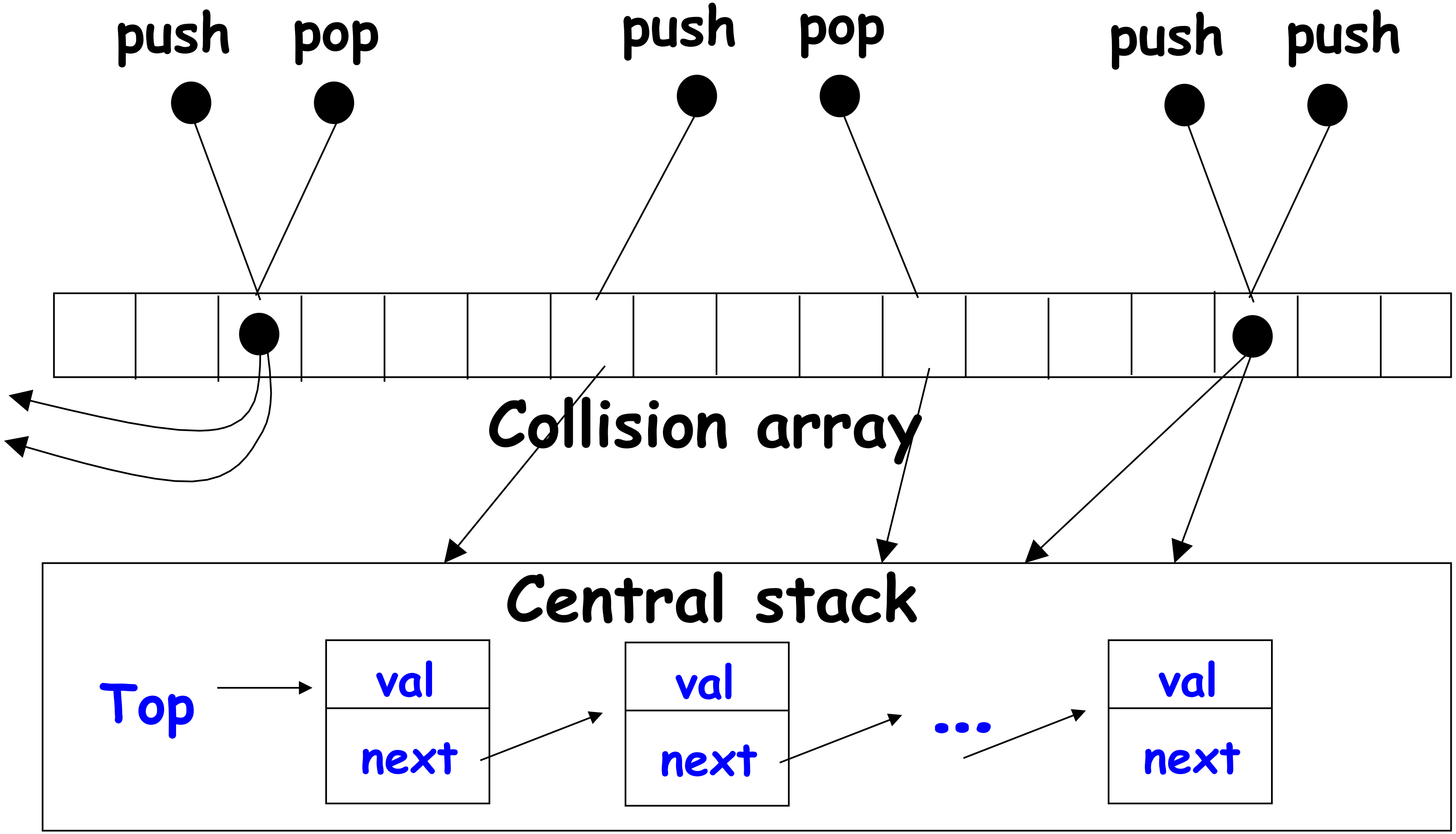
An elimination-backoff stack (Hendler, Shavit & Yerushalmi, 2004)

Key idea:

pairs of push/pop operations may collide and eliminate each other without accessing a central stack.

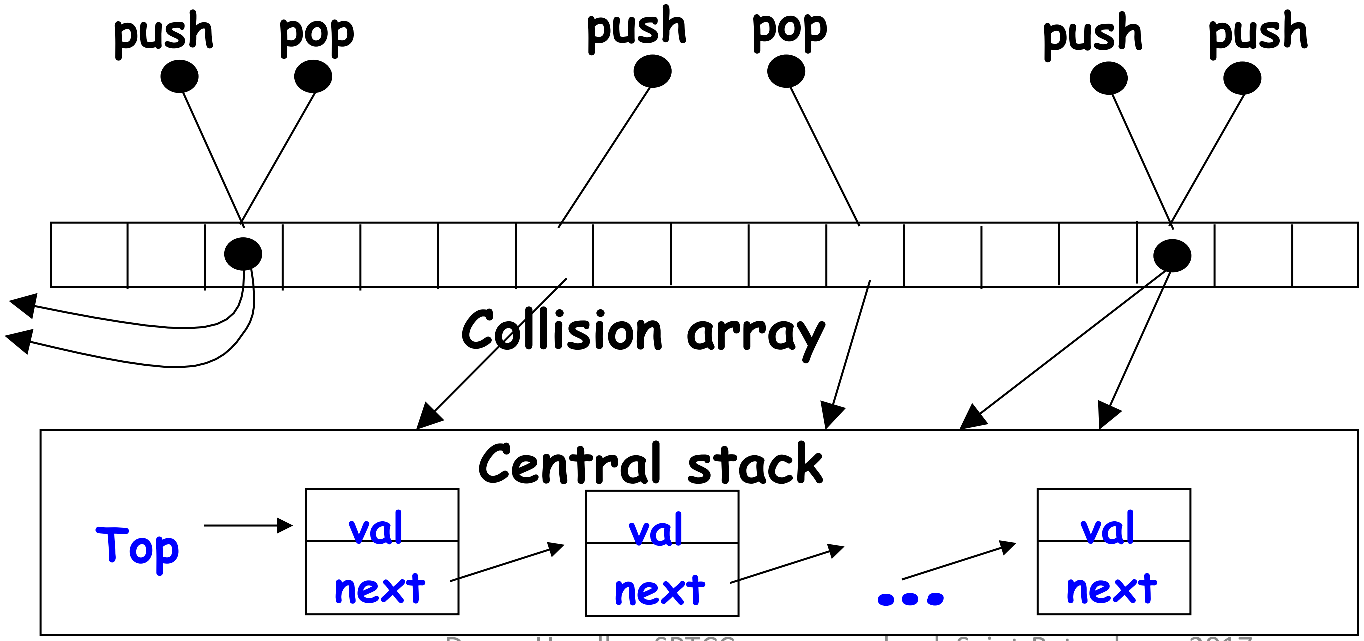


An elimination-backoff stack
Collision scenarios



An elimination-backoff stack
Elimination challenges

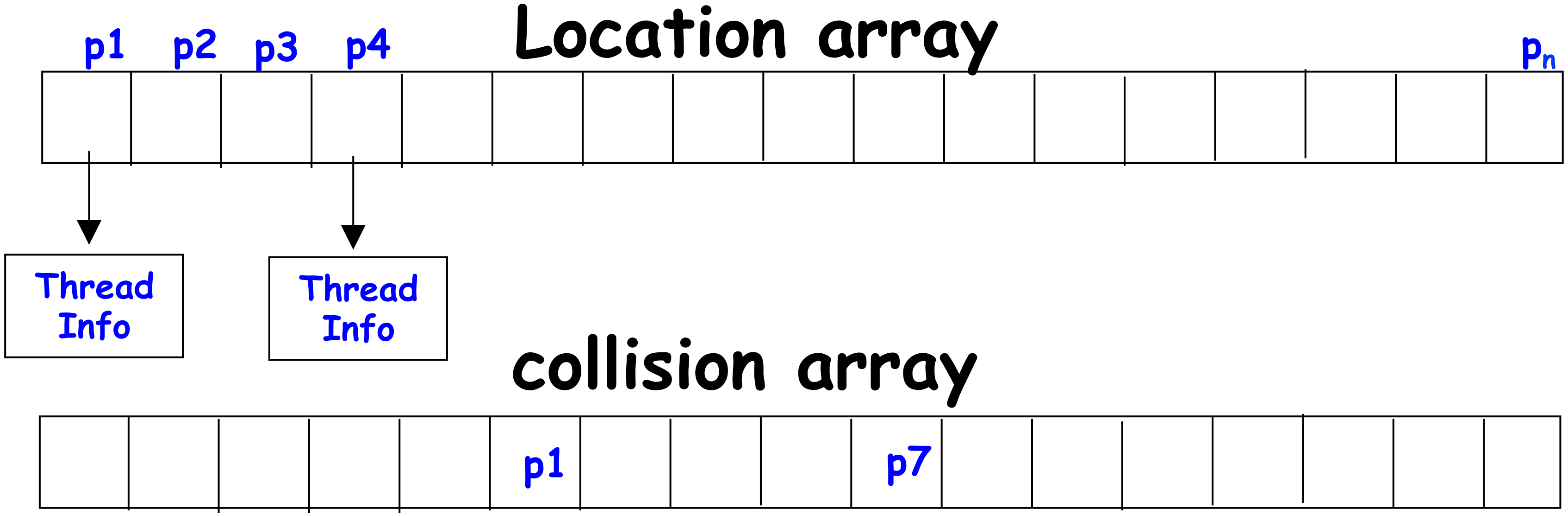
- ❑ Prevent elimination chains: e.g., A collides with B, which collides with C…
- ❑ Prevent race conditions: e.g., A collides with B, which is already gone…



Data structures

Each stack operation is represented by a ThreadInfo structure

```
struct ThreadInfo {
        id          ;the identifier of the thread performing the operation
        op          ;a PUSH/POP opcode
        cell        ;a cell structure
        spin        ; duration to spin
        }
Struct cell { ;a representation of stack item as in Treiber
        pnext       ;pointer to the next cell
        pdata       ;stack item}
```



Pseudo-code: main loop

```
void EStack(ThreadInfo *p)
  Do forever
    stack:  if (TryPerformStackOp(p)==TRUE) return ;Aapply op to central stack
    location[mypid]=p ;announce arrival
    pos=GetPosition(p) ;get a random position at the collision array
    him=collision[pos] ;read current value of that position
    while (!compare&swap(&collision[pos],him,mypid);try to write own ID
      him=collision[pos]                              ;continue till success
    if (him != empty) ;if read an ID of another thread
      q=location[him] ;read a pointer to the other thread's info
      if (q!=NULL &&  q->id=him && q->op != p->op) ;if may collide
        if (compare&swap(&location[mypid],p,NULL) ; prevent unwanted collisions
          if (TryCollision(p,q)==true) ;if collided successfully
            return ;return code is already at ThreadInfo structure
          else goto stack ;try to apply operation to central stack
        else FinishCollision(p), return ;extract information and finish
    delay (p->spin) ;Wait for other thread to collide with me
    if (!compare&swap(&location[mypid],p,NULL) ;if someone collided with me
      FinishCollision(p), return;Extract information and finish
```

Pseudo-code: TryCollision,FinishCollision

```
void TryCollision(ThreadInfo* p, ThreadInfo *q)
    if (p->op==PUSH)
        if (compare&swap(&location[him],q,p)) ;give my record to other thread
            return TRUE
        else
            return FALSE
    else
        if (compare&swap(&location[him],q,NULL))
            p->cell=q->cell ;get pointer to PUSH operation's cell
            return TRUE
        else
            return FALSE
```

```
void FinishCollision(ThreadInfo* p)
    if (p->op==POP)
        p->pcell=location[mypid]->pcell
        location[mypid]=NULL
```

Linearization points

If operation completed on central stack, same as Treiber

<u>Otherwise:</u>

Colliding operations-pair linearized together - push before pop.

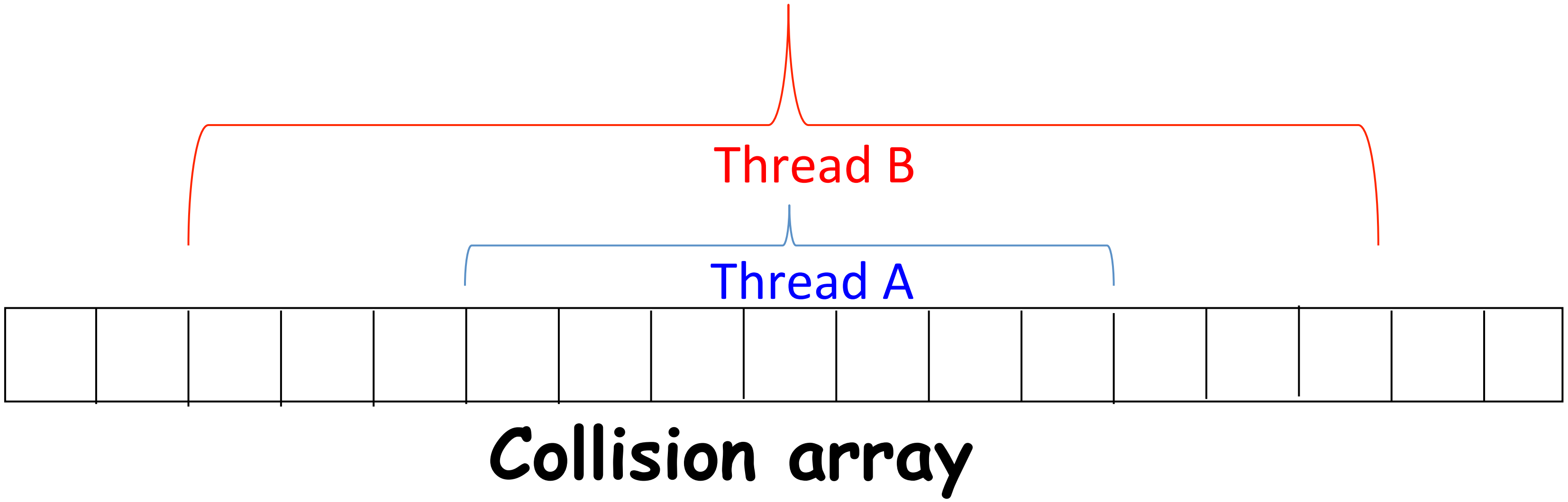
Upon success

Upon success

```
void TryCollision(ThreadInfo* p, ThreadInfo *q)
    if (p->op==PUSH)
        if (compare&swap(&location[him],q,p)) ;give my record to other thread
            return TRUE
        else
            return FALSE
    else
        if (compare&swap(&location[him],q,NULL))
            p->cell=q->cell ;get pointer to PUSH operation's cell
            return TRUE
        else
            return FALSE
```

Adaptive elimination backoff

- ❑ Handle load by backoff in space and time
  - o E.g., exponential backoff
- ❑ Decisions made locally, per thread
- ❑ Array-width/waiting-period decreased when:
  - o Many `no-show' unsuccessful collision attempts
- ❑ Array-width/waiting-period increased when:
  - o Many `high-contention' unsuccessful collision attempts



Talk Outline

- Preliminaries
- A simple lock-free stack algorithm
  - Linearizability
- Michael & Scott queue algorithm
- The Harris-Michael linked list algorithm
- Elimination-based stack
- Discussion & conclusions

The notion of helping

❑ Lock-free algorithms may be made wait-free using the notion of *helping*

❑ Used for wait-free data-structures and universal constructions

❑ Formal definitions attempted only recently

Censor-Hillel, Petrank and Timnat, PODC, 2015
Attiya, Castañeda and Hendler , OPODIS, 2015

- Used for proving complexity & impossibility results

Informal notions of `helping' (1)

Informal notions of `helping' (2)

Conclusions

❑ Lock-free algorithms may be often wait-free in practice

❑ Require strong synchronization operations

❑ Often difficult to devise

❑ Guarantee global progress in the face of thread failures

Any
Questions

Exercise formulation
The swap and fetch-and-inc operations

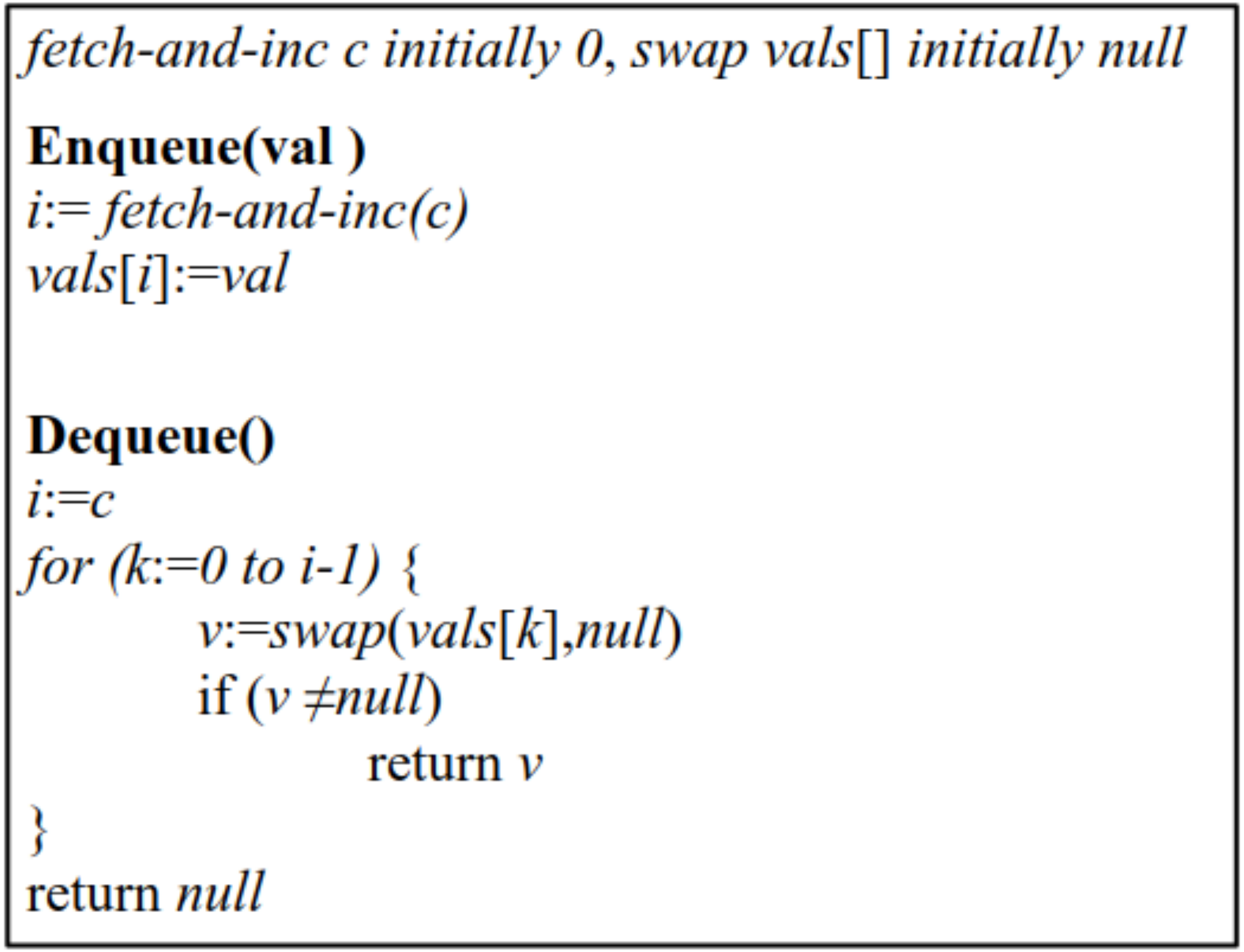
fetch-and-inc(c)

```
atomically
  t ⟵ read from c
  c ⟵ c + 1
  return t
```

swap(var,new)

```
atomically
  t ⟵ read from var
  var ⟵ new
  return t
```

Exercise formulation
A lock-free queue algorithm

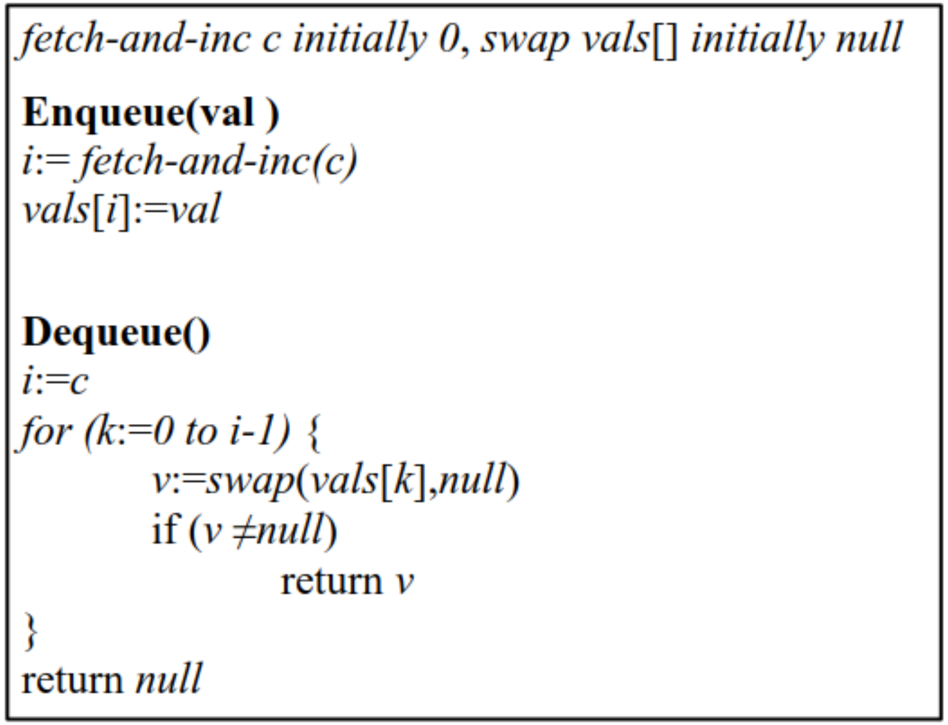


```
fetch-and-inc c initially 0, swap vals[] initially null

Enqueue(val )
i:= fetch-and-inc(c)
vals[i]:=val


Dequeue()
i:=c
for (k:=0 to i-1) {
        v:=swap(vals[k],null)
        if (v ≠null)
                return v
}
return null
```

Exercise formulation
The questions

a. Describe a detailed execution showing that the algorithm is not linearizable.

b. Present a small change to the algorithm to make it linearizable (and still lock-free).



```
fetch-and-inc c initially 0, swap vals[] initially null

Enqueue(val )
i:= fetch-and-inc(c)
vals[i]:=val


Dequeue()
i:=c
for (k:=0 to i-1) {
          v:=swap(vals[k],null)
          if (v ≠null)
                    return v
}
return null
```