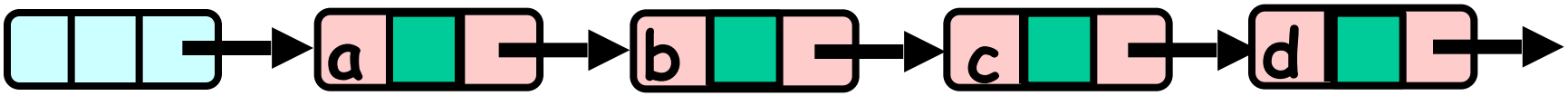
Implémentation paresseuse de Liste

- Comme optimiste, sauf que
  - On traverse une seule fois pour add et remove (mais restent avec des verrous)
  - contains(x) sans verrous …
- Comment?
  - Détruite les noeuds posent des problèmes
  - On le fait "paresseusement"

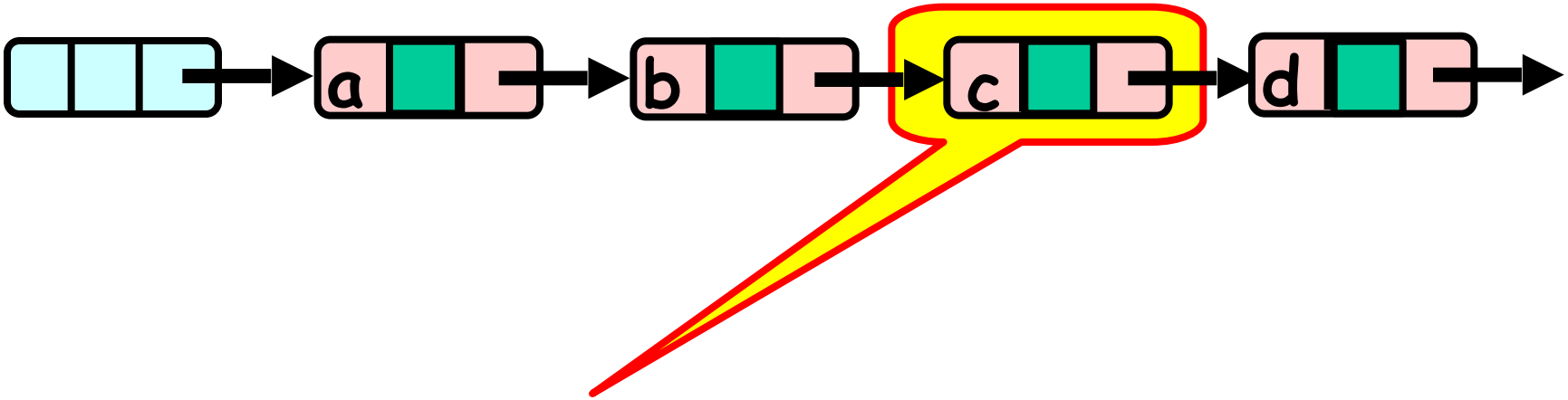
Liste paresseuse

- remove()
  - Parcourir la liste (comme avant)
  - Verrouiller prédécesseur & courant (comme avant)
- Destruction logique
  - Marqué le noeud courant comme enlevé (nouveau!)
- Destruction physique
  - Redirige le suivant de prédécesseur (comme avant)

Liste paresseuse

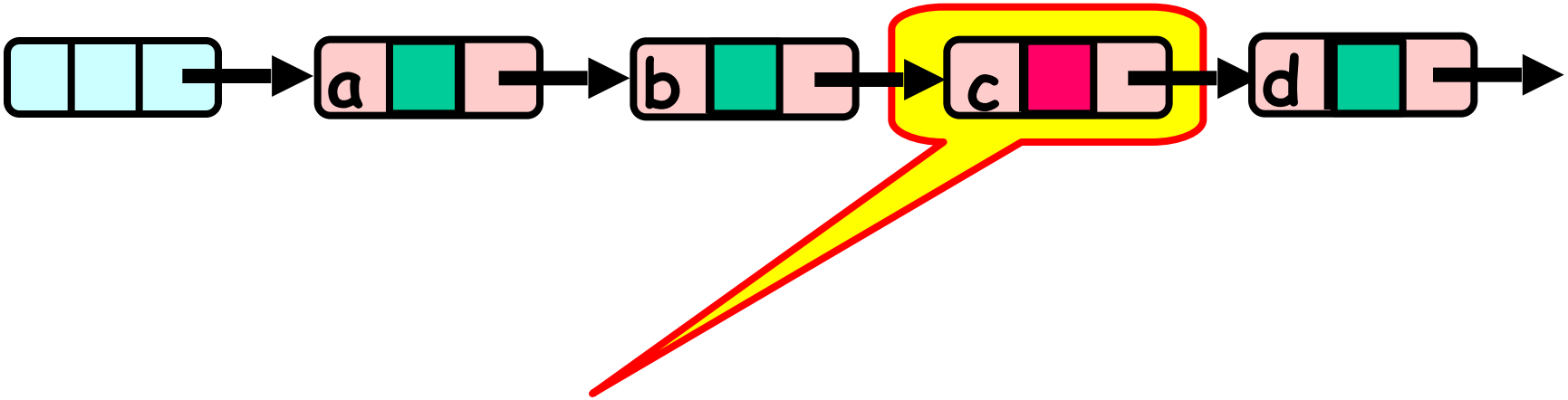


Liste paresseuse



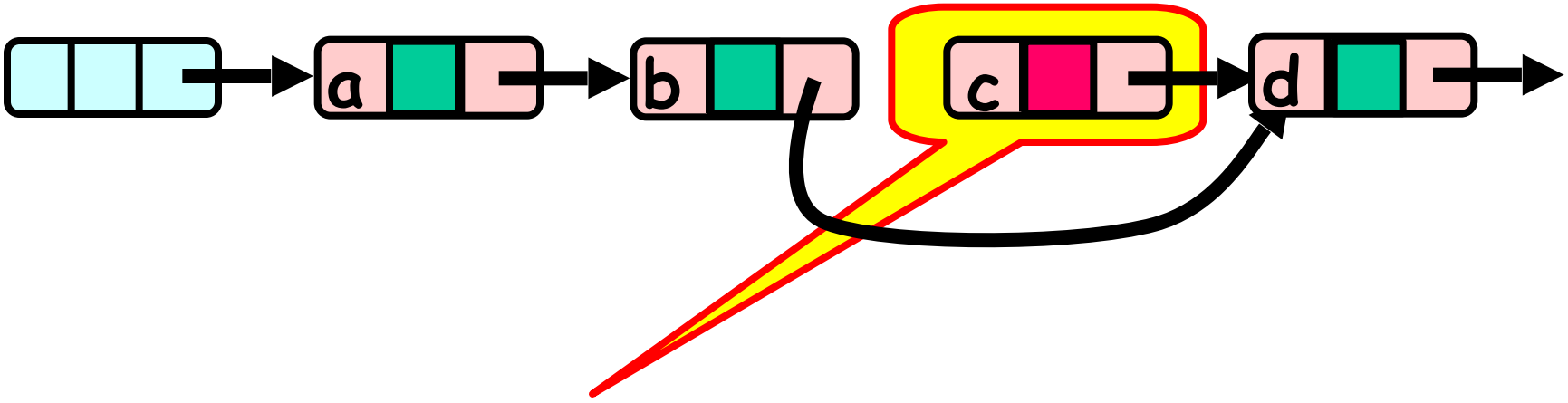
Présent dans la liste

Liste paresseuse



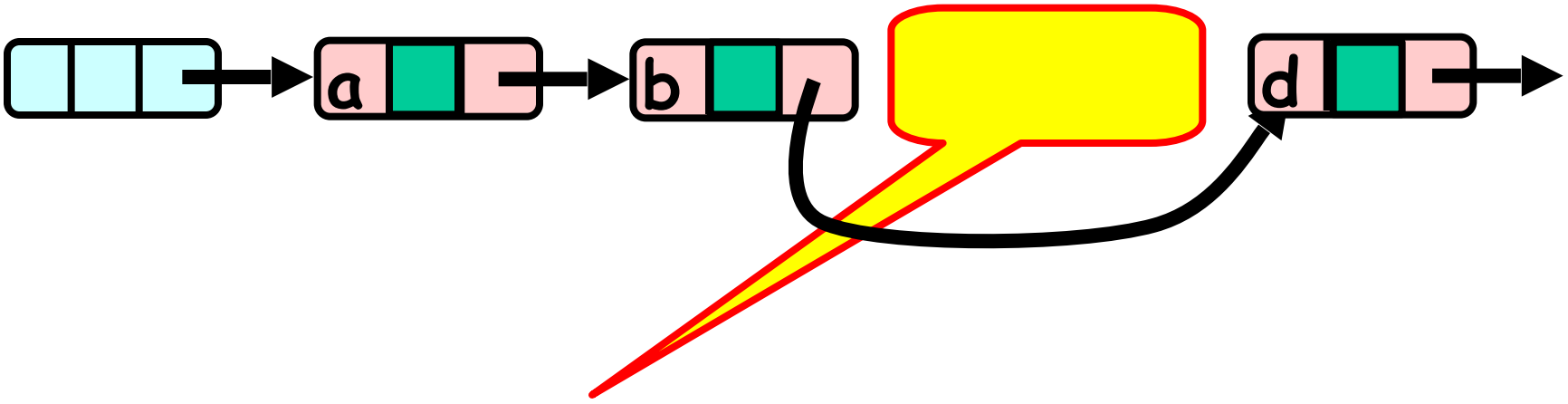
Destruction logique

Liste paresseuse



Destruction physique

Liste paresseuse



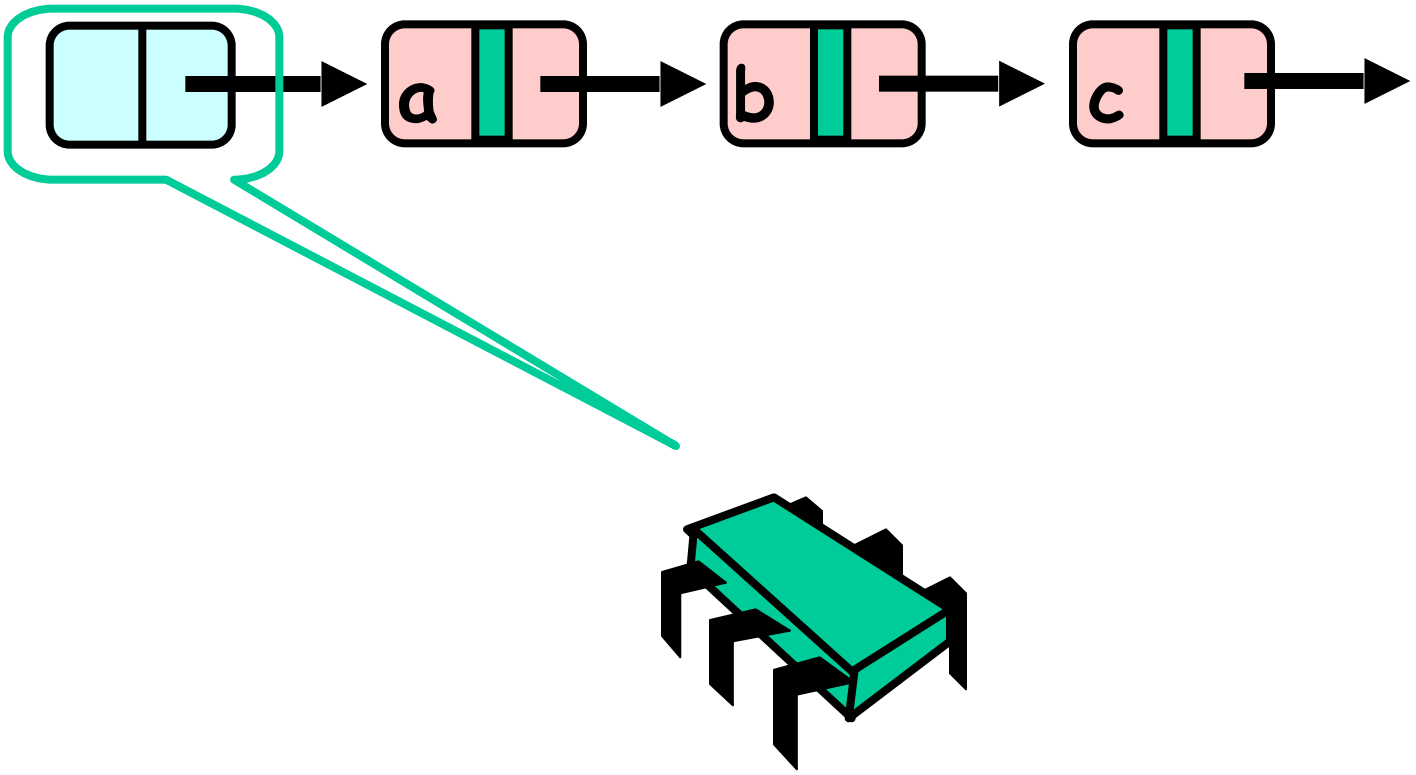
Destruction physique

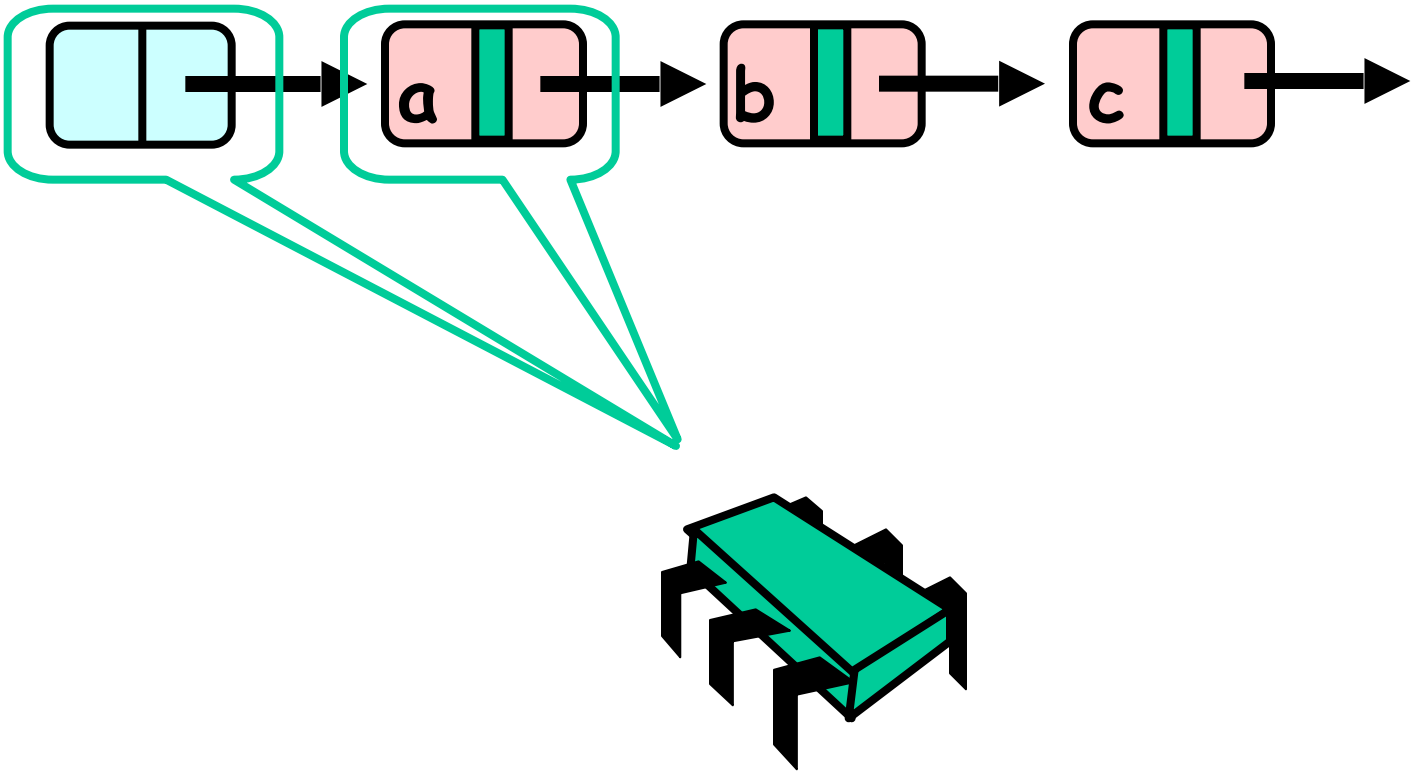
Destruction physique

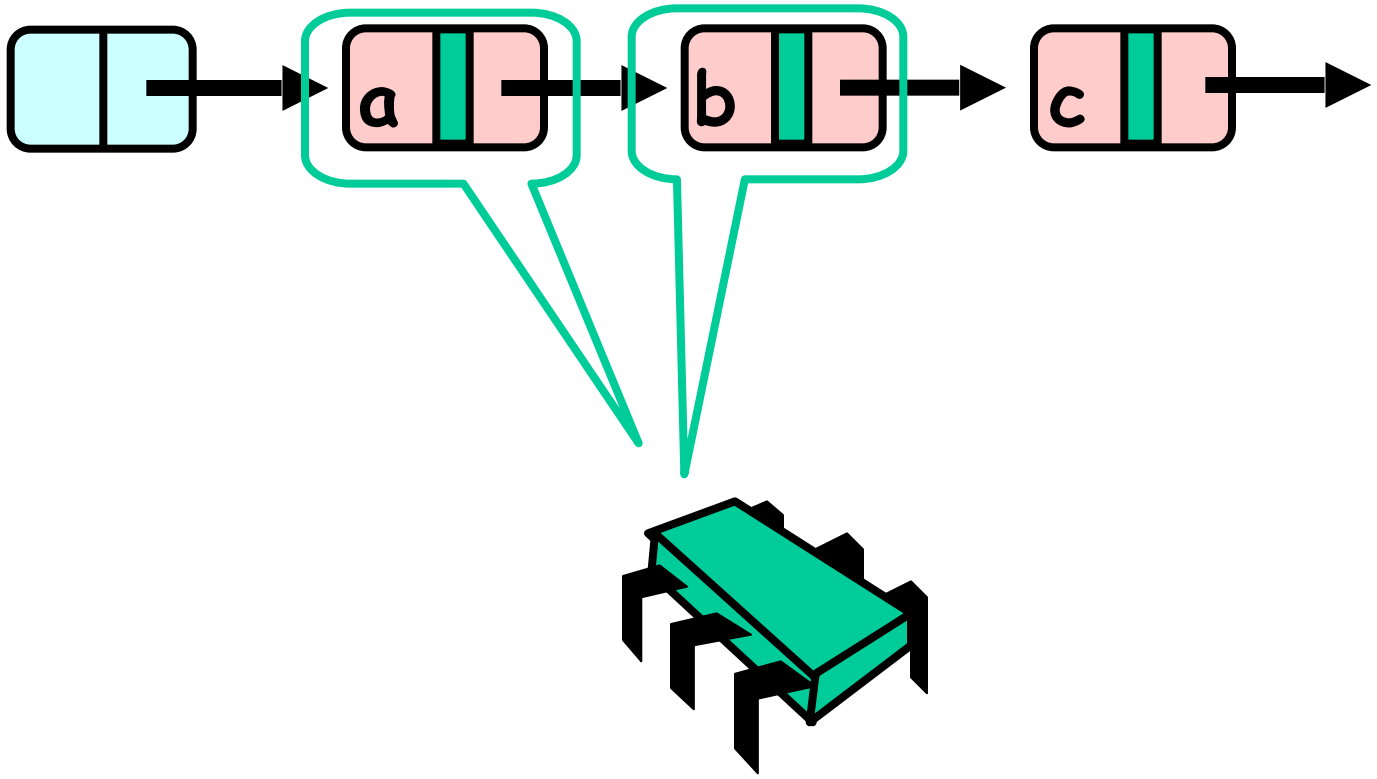
- Toutes les méthodes
  - Parcours sans verrou parmi les noeuds verrouillés et marqués
  - La destruction d'un noeud ne ralentit pas les autres appels…
- On doit encore verrouiller pred et curr

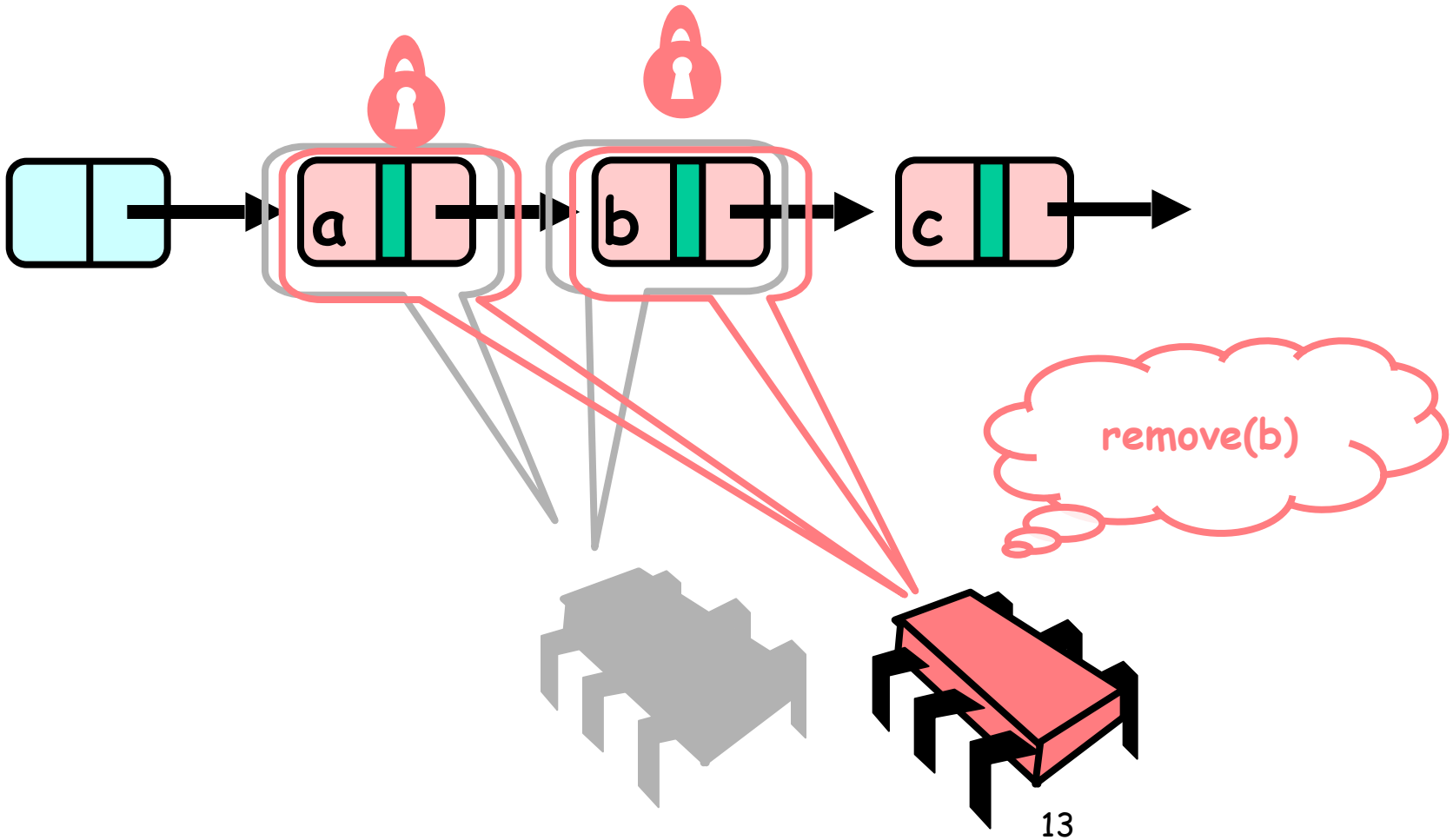
Validation

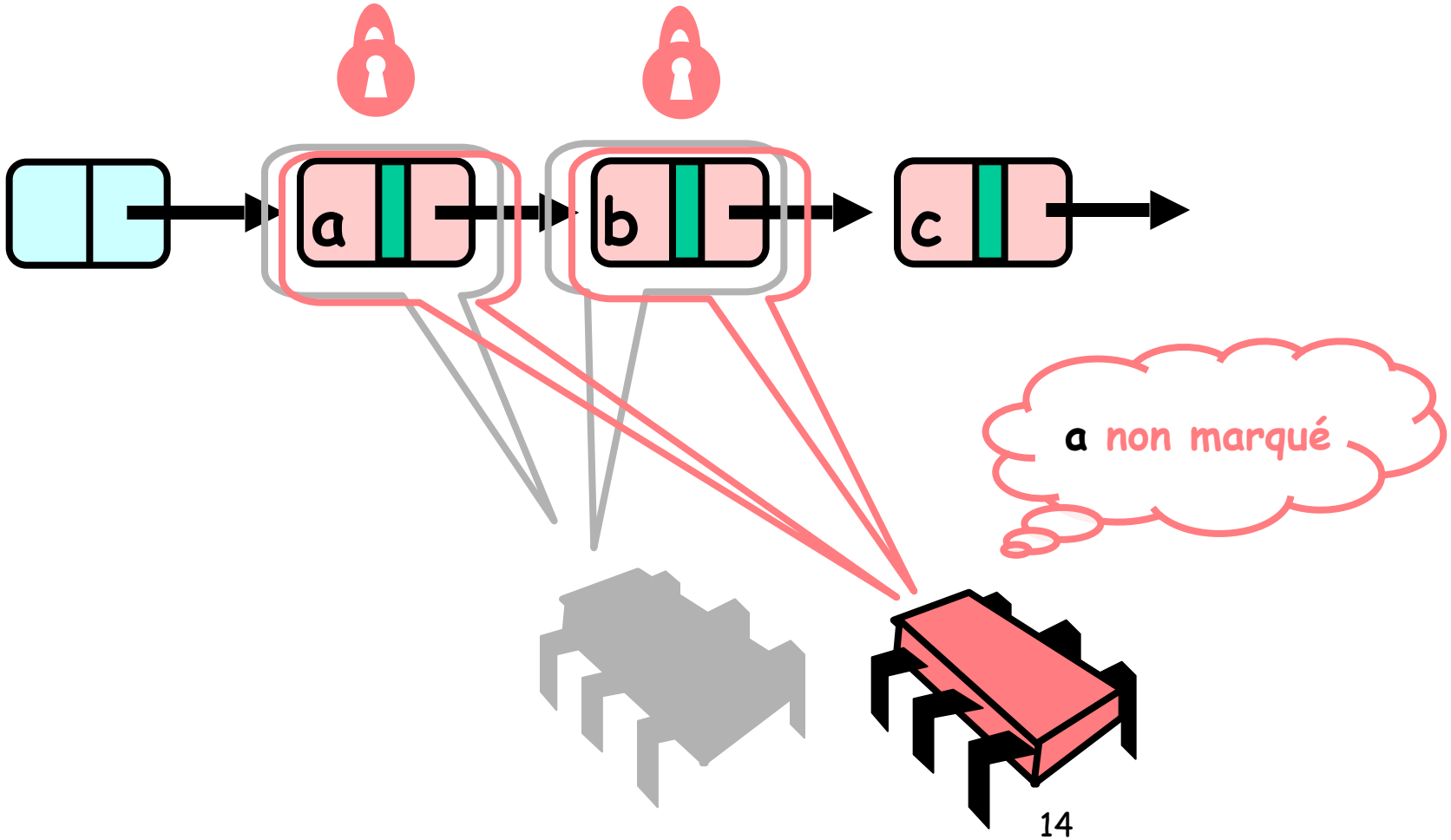
- Pas besoin de reparcourir la liste!
- Vérifie que pred n'est pas marqué
- Vérifie que curr n'est pas marqué
- Vérifie que le successeur de pred est curr

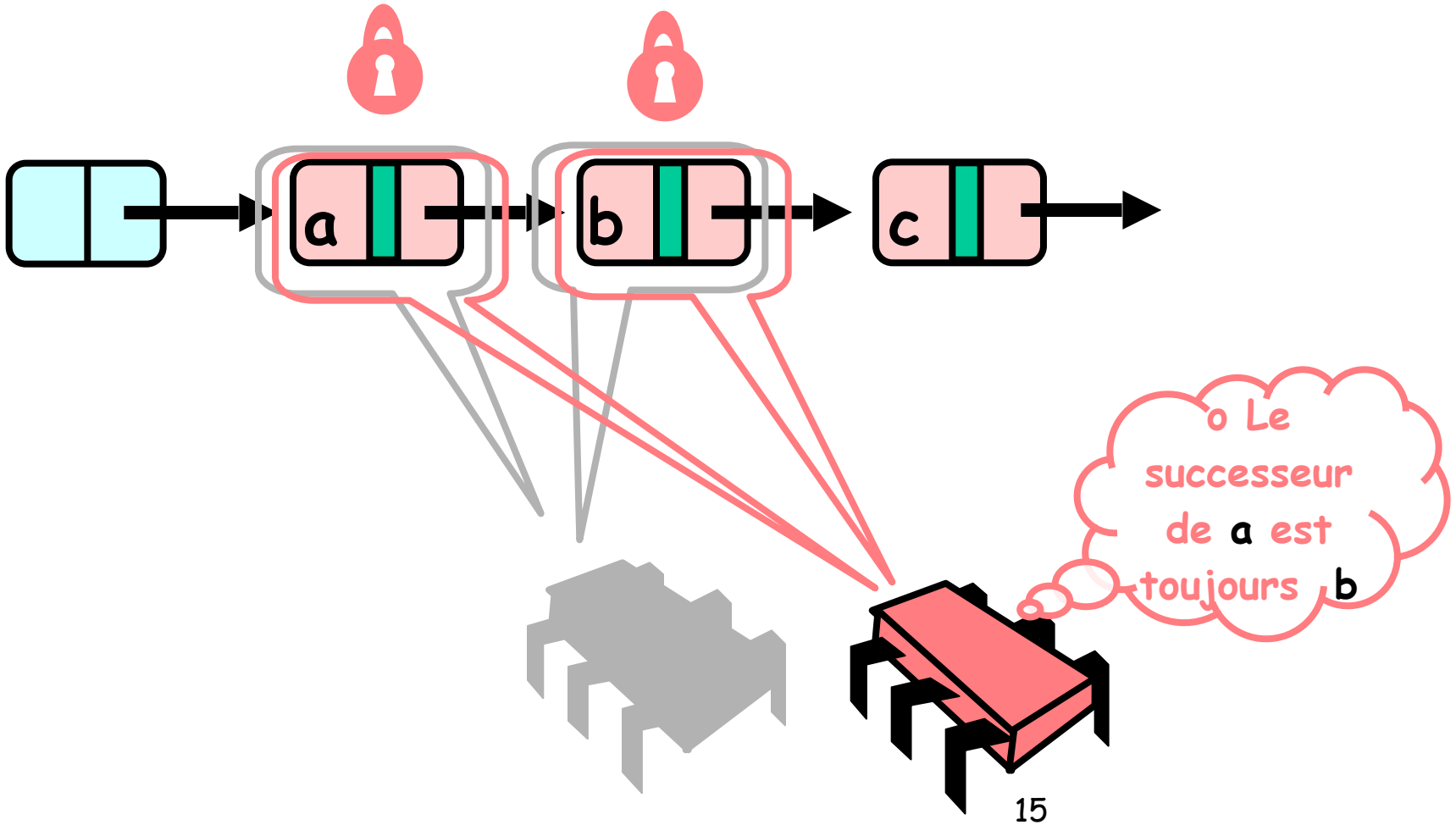

a
b
c

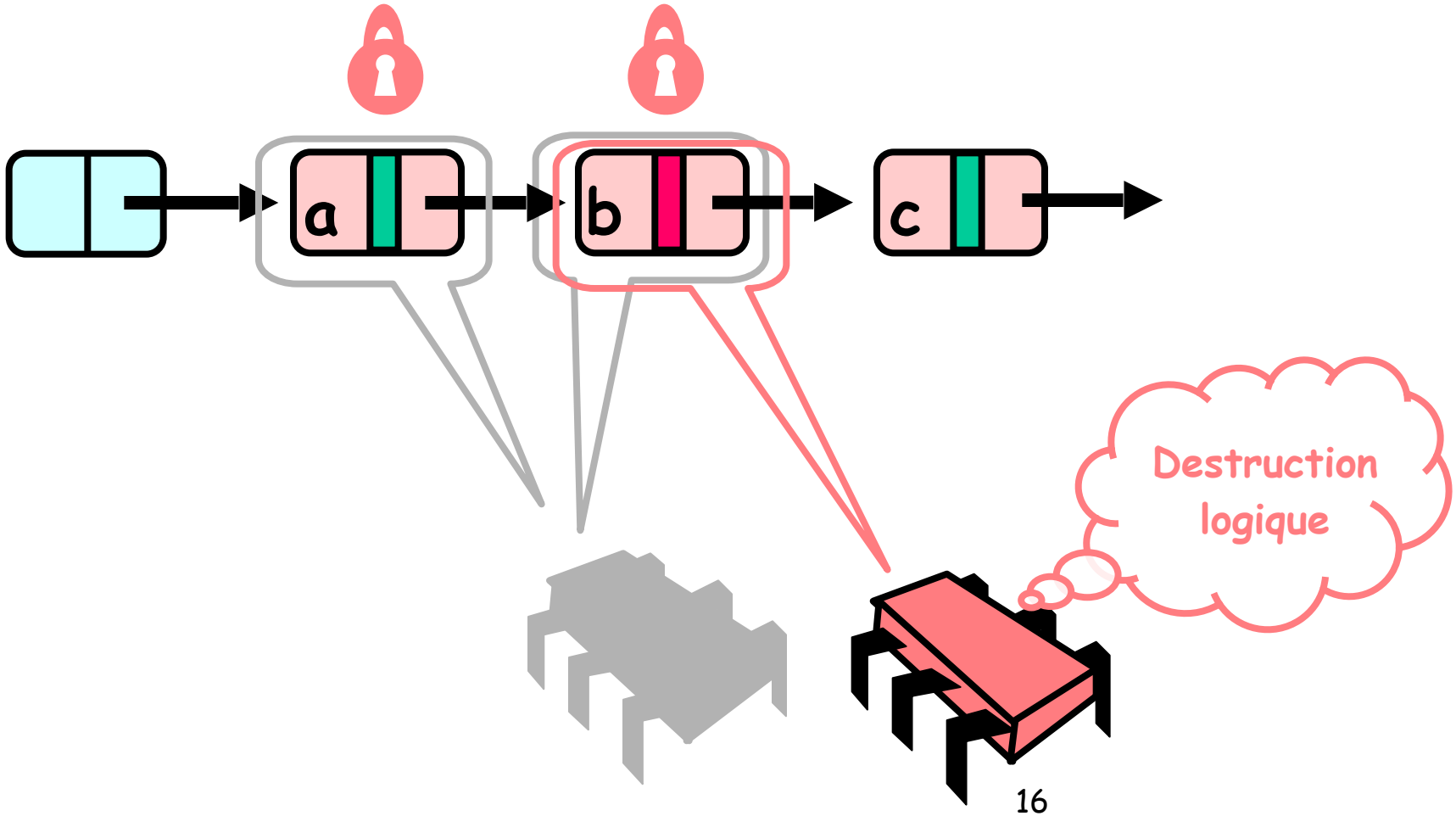

a
b
c

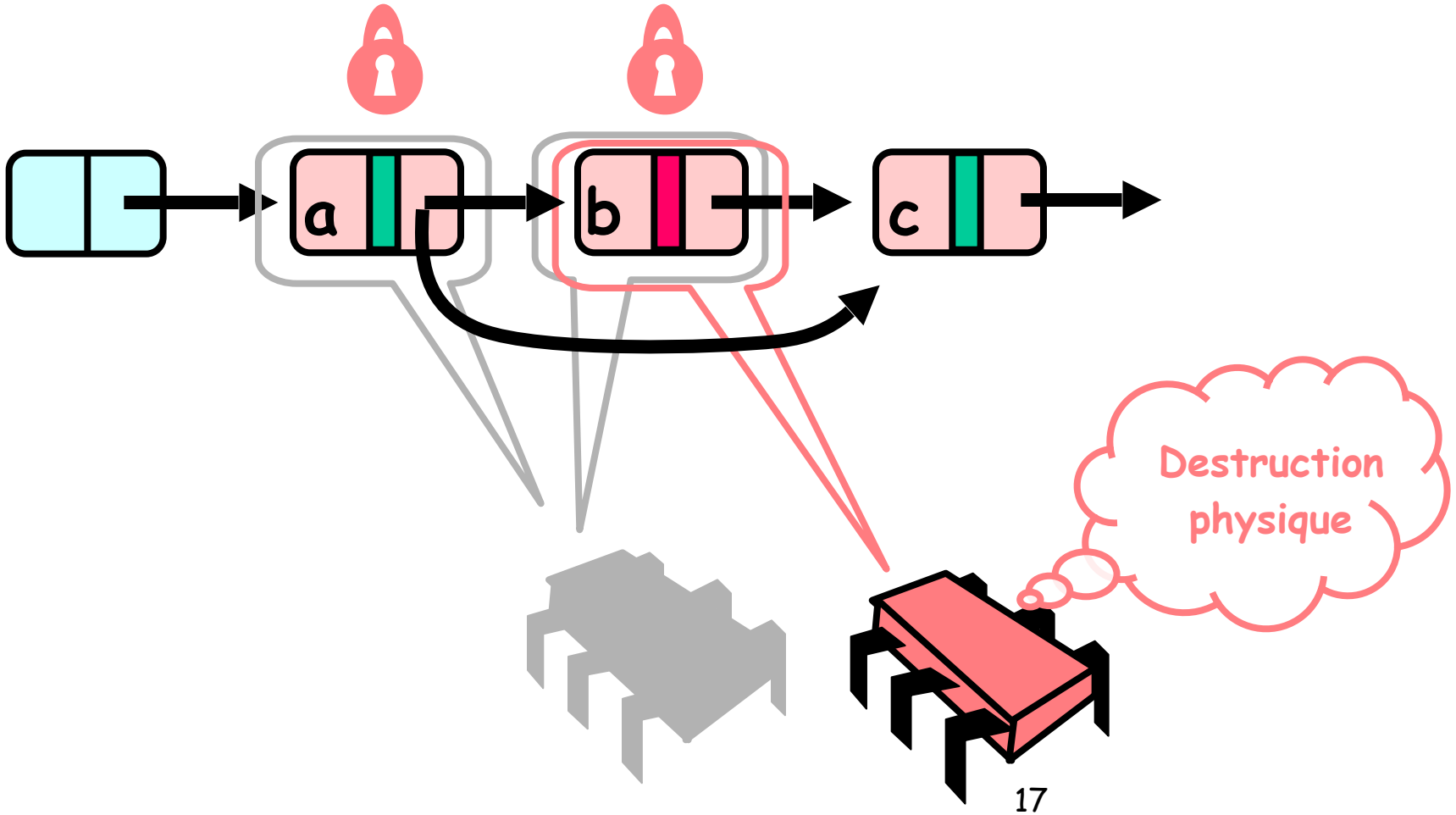

a
b
c

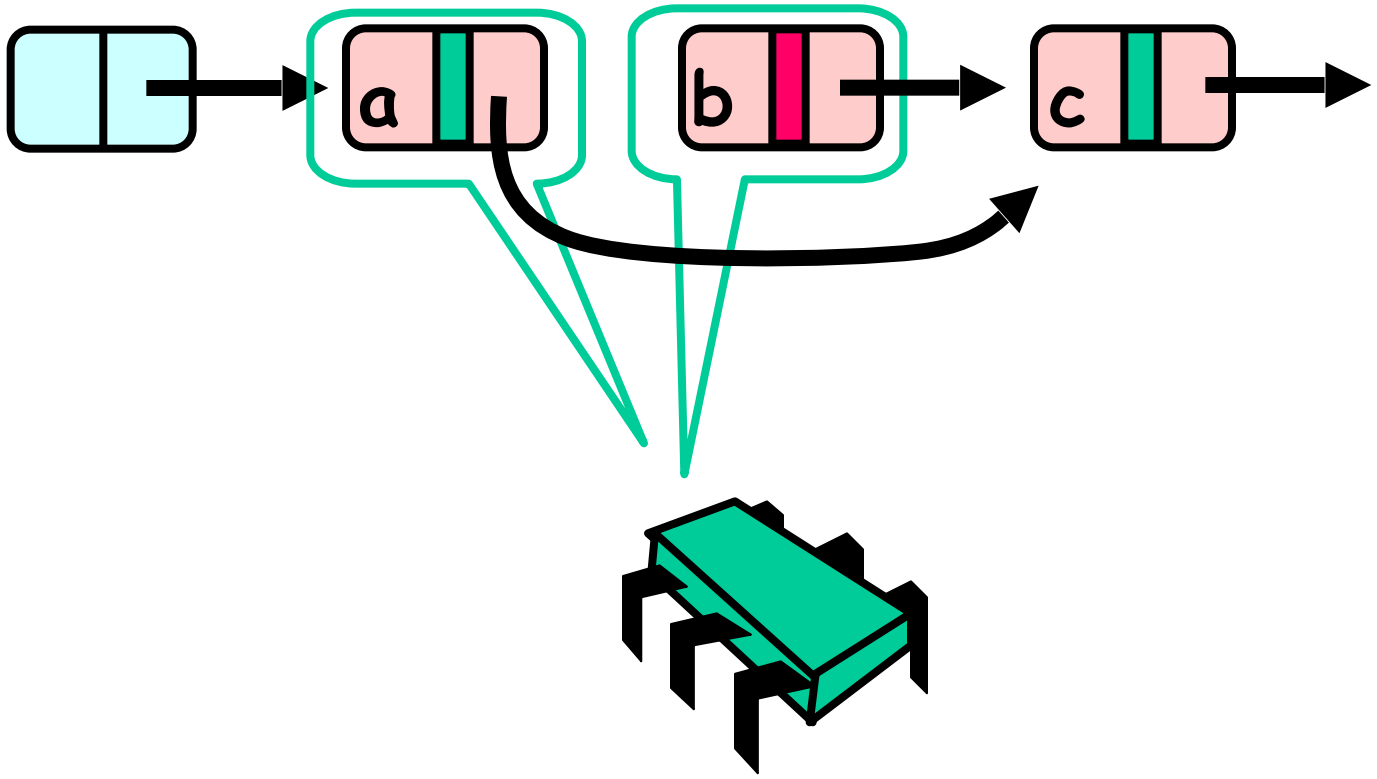

a
b
c
remove(b)


a
b
c
a non marqué


a
b
c
Le successeur de a est toujours b


a
b
c
Destruction
logique


a
b
c
Destruction
physique


a
b
c

Nouvelle application

- S(head) =
  - { x | il existe un noeud a tel que
    - a atteignable de head et
    - a.item = x et
    - a n'est pas marqué
  - }

Invariant

- Si non marqué l'item est dans set
- Et atteignable depuis head
- Et si la traversée n'est pas fini atteignable de pred

Validation

```
private boolean
  validate(Node pred, Node curr) {
 return
  !pred.marked &&
  !curr.marked &&
  pred.next == curr);
  }
```

Validation

```
private boolean
  validate(Node pred, Node curr) {
 return
  !pred.marked &&
  !curr.marked &&
  pred.next == curr);
  }
```

Pred pas détruit logiquement

Validation

```
private boolean
  validate(Node pred, Node curr) {
 return
  !pred.marked &&
  !curr.marked &&
  pred.next == curr);
  }
```

Curr pas détruit logiquement

Validation

```
private boolean
  validate(Node pred, Node curr) {
 return
  !pred.marked &&
  !curr.marked &&
  pred.next == curr);
 }
```

Le suivant de pred est toujours curr

Remove

```
public boolean remove(Item item) {
 int key = item.hashCode();
while (true) {
   Node pred = head;
   Node curr = pred.next;
   while (curr.key < key) {
         pred = curr; curr = curr.next;}
```

Remove

```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
   if (curr.key == key) {
    curr.marked = true;
    pred.next = curr.next;
    return true;
   } else {
    return false;
   }}} finally {
          pred.unlock();
          curr.unlock();
   }}}
```

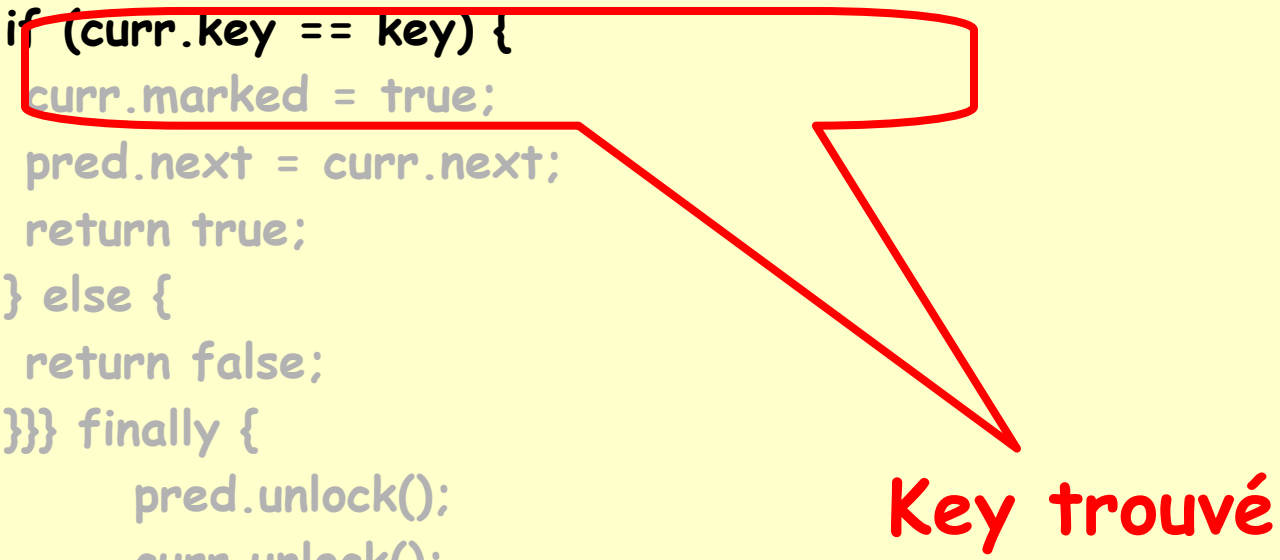
Remove

```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
    if (curr.key == key) {
      curr.marked = true;
      pred.next = curr.next;
      return true;
    } else {
      return false;
    }}} finally {
          pred.unlock();
          curr.unlock();
    }}}
```

Validate comme avant

Remove

```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
   if (curr.key == key) {
    curr.marked = true;
    pred.next = curr.next;
    return true;
   } else {
    return false;
   }}} finally {
         pred.unlock();
         curr.unlock();
   }}}
```



Remove

```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
   if (curr.key == key) {
    curr.marked = true;
    pred.next = curr.next;
    return true;
   } else {
    return false;
   }}} finally {
         pred.unlock();
         curr.unlock();
   }}}
```

Destruction logique

Remove

```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
   if (curr.key == key) {
    curr.marked = true;
    pred.next = curr.next;
    return true;
   } else {
    return false;
   }}} finally {
         pred.unlock();
         curr.unlock();
   }}}
```

Destruction physique

Remove

```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
   if (curr.key == key) {
    curr.marked = true;
    pred.next = curr.next;
    return true;
   } else {
    return false;
   }}} finally {
         pred.unlock();
         curr.unlock();
   }}}
```

On recommence si validate faux

Add

```
public boolean add(Item item) {
 int key = item.hashCode();
while (true) {
   Node pred = head;
   Node curr = pred.next;
   while (curr.key < key) {
         pred = curr; curr = curr.next;}
```

Add

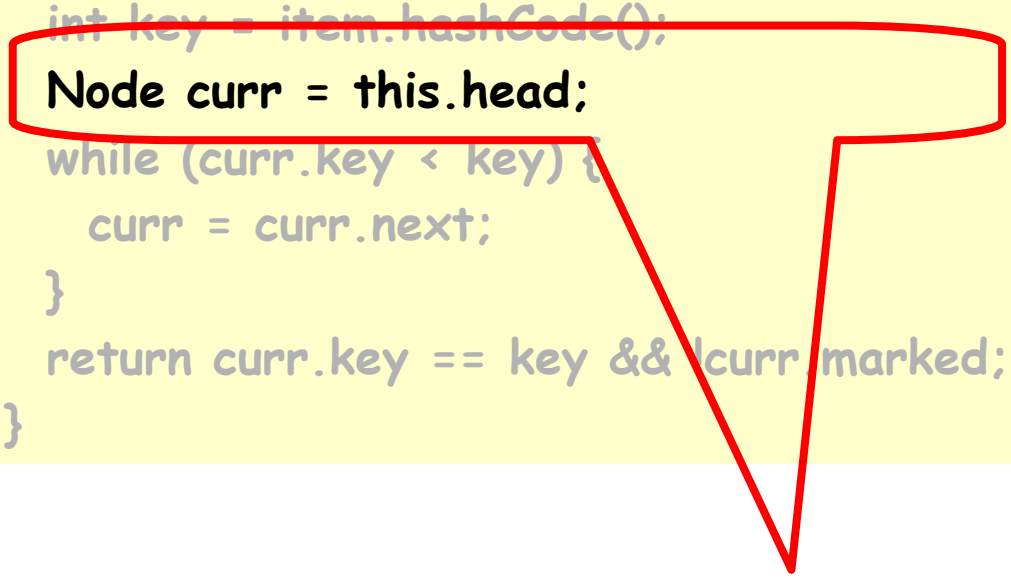
```
try {
  pred.lock(); curr.lock();
  if (validate(pred,curr) {
   if (curr.key != key) {
     Node node=new Node(item);
    Node.next=curr;
    pred.next = node;
    return true;
   } else {
    return false;
   }}} finally {
          pred.unlock();
          curr.unlock();
   }}}
```

Contains

```
public boolean contains(Item item) {
  int key = item.hashCode();
  Node curr = this.head;
  while (curr.key < key) {
    curr = curr.next;
  }
  return curr.key == key && !curr.marked;
}
```

Contains

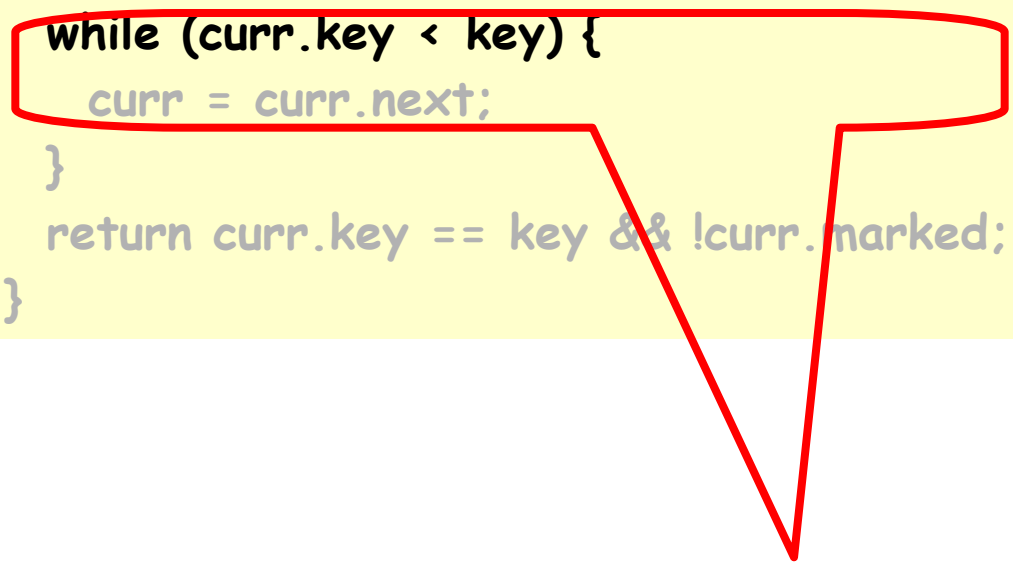
```
public boolean contains(Item item) {
  int key = item.hashCode();
  Node curr = this.head;
  while (curr.key < key) {
    curr = curr.next;
  }
  return curr.key == key && !curr.marked;
}
```



On commence à head

Contains

```
public boolean contains(Item item) {
  int key = item.hashCode();
  Node curr = this.head;
  while (curr.key < key) {
    curr = curr.next;
  }
  return curr.key == key && !curr.marked;
}
```



Recherche par la clef

Contains

```
public boolean contains(Item item) {
  int key = item.hashCode();
  Node curr = this.head;
  while (curr.key < key) {
    curr = curr.next;
  }
  return curr.key == key && !curr.marked;
}
```

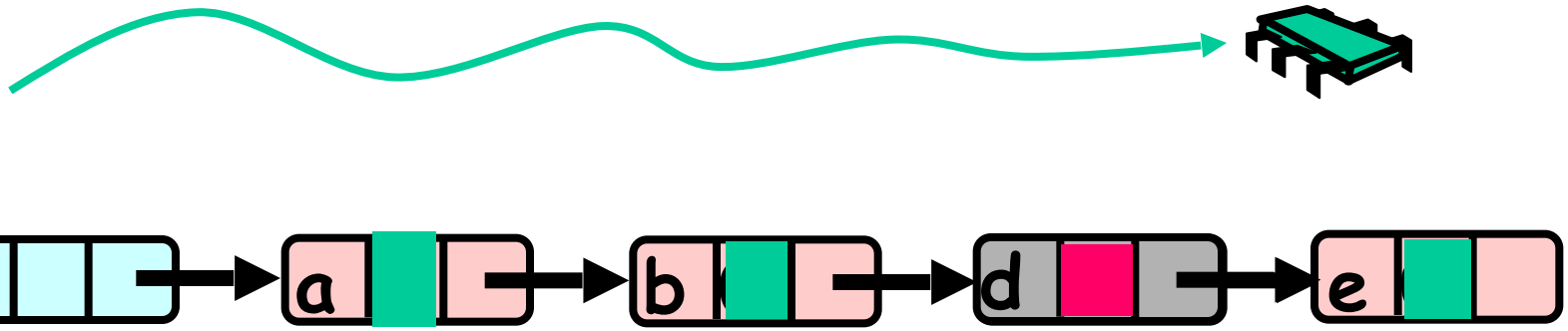
Traverse sans verrou
(Les noeuds peuvent avoir été enlevés)

Contains

```
public boolean contains(Item item) {
  int key = item.hashCode();
  Node curr = this.head;
  while (curr.key < key) {
    curr = curr.next;
  }
  return curr.key == key && !curr.marked;
}
```

Present et non enlevé?

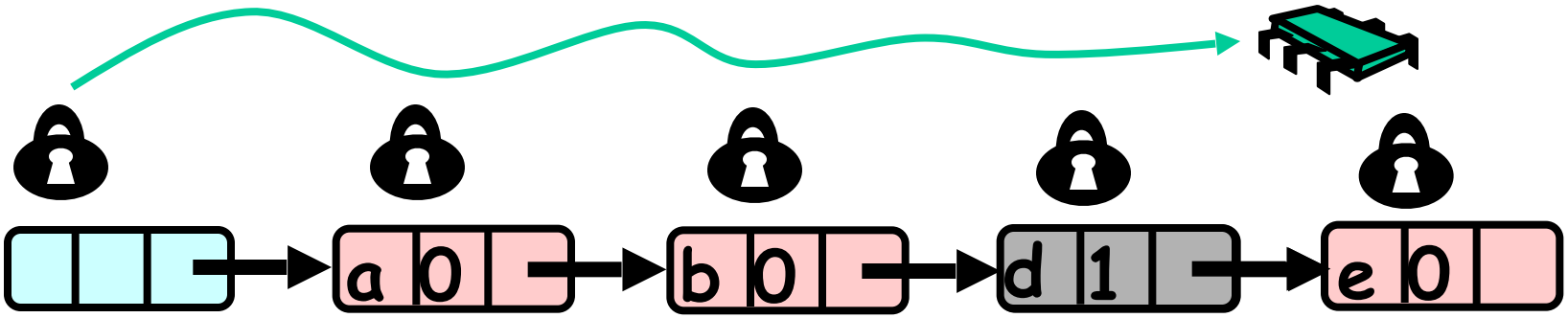
Résumé: Wait-free Contains



Utilise un bit indiquant le noeud marqué et le fait que la liste soit ordonnée

1. Pas marqué →dans set
2. Marqué ou absent → pas dans set

Paresseux



add() et remove() paresseux (blocking) + Wait-free contains()

Evaluation

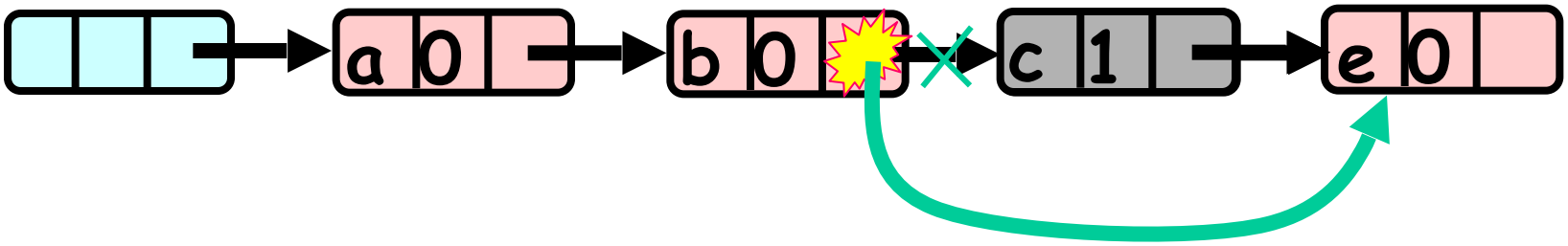
- Bon:
  - contains() sans lock (wait-free!)
  - Bon car souvent beaucoup d'appel à contains()
  - Pas de retraversée en cas de succès
- Mauvais
  - Contention sur add() et remove(), on peut être amener à retraverser
  - Une thread lente ralentit l'appel de ces méthodes

Implémentation sans verrou

- Elimine entièrement les verrous
- contains() wait-free ; add() et remove() non blocking
- Utilise compareAndSet()
- Que peut-il se passer?

Remove en utilisant CAS

Remove logique =
Positionne un bit

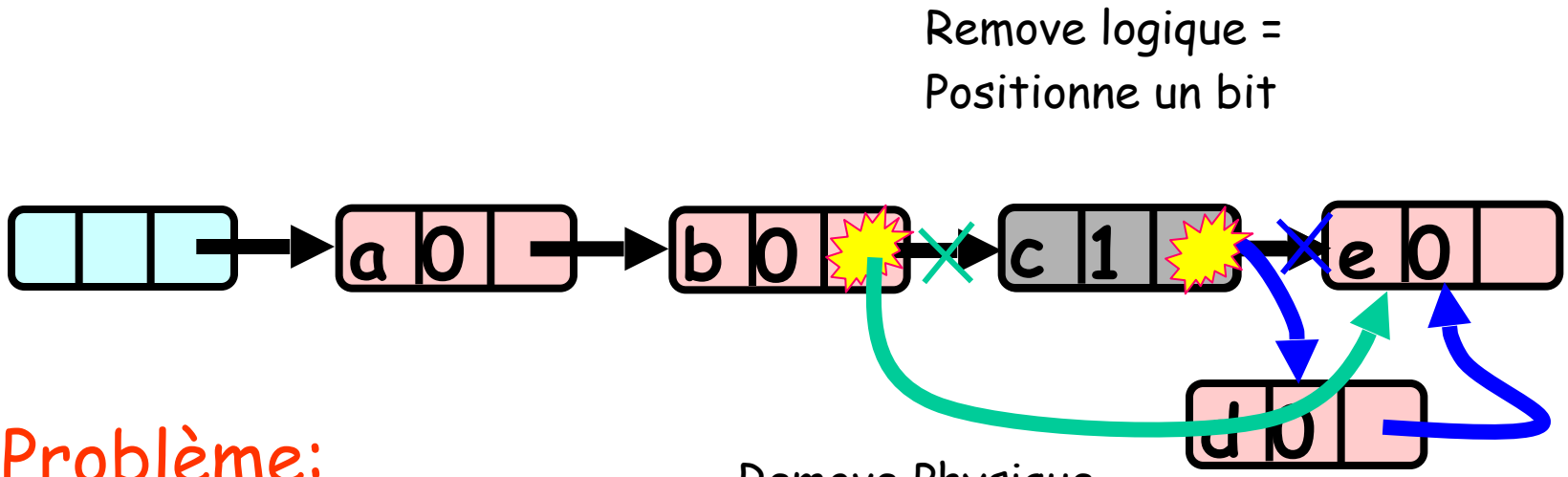


Utiliser CAS pour modifier le champ next

Remove physique
modification du
pointeur CAS

Insuffisant

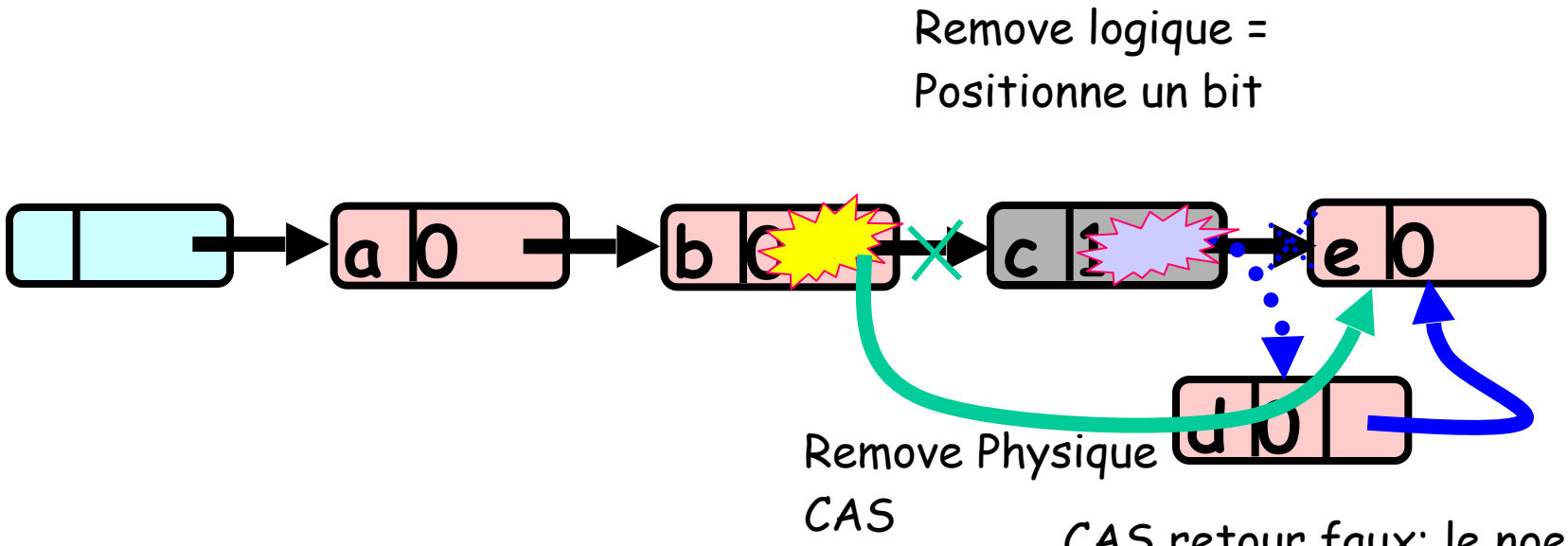
Problème…



Problème:
d n'a pas été ajouté
à la liste…

Faire attention au
pointeur des
noeuds enlevés

Solution: Combiner Bit et CAS



Bit de marque et le pointeur sont ensemble dans le CAS (AtomicMarkableReference)

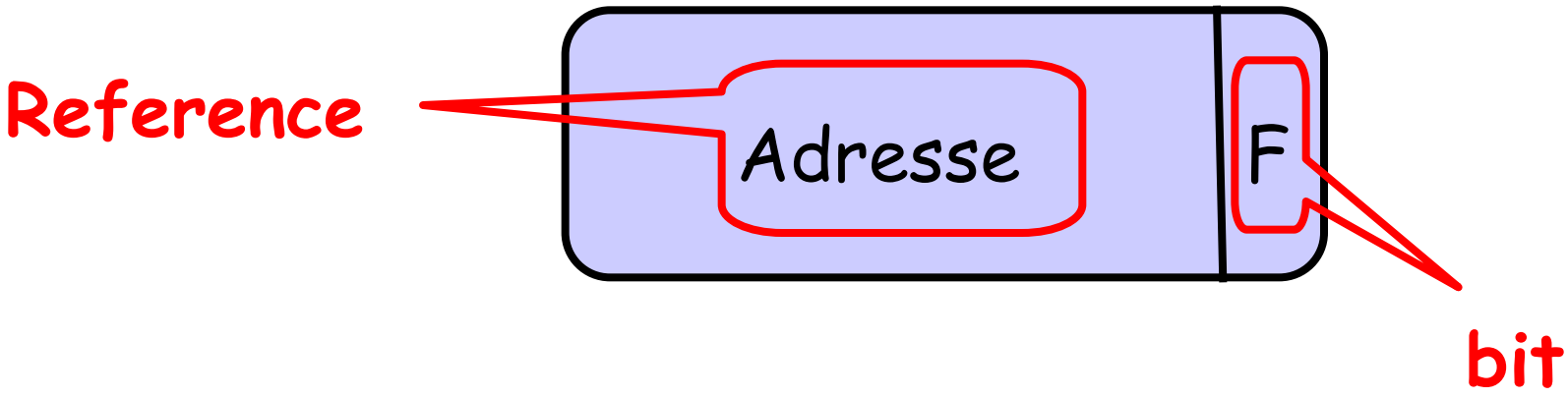
Solution

- Utilise AtomicMarkableReference
- Atomiquement
  - Modifie la référence et
  - Met à jour la marque
- Remove en deux étapes
  - Mettre la marque dans le champ next
  - Rediriger le pointeur du prédécesseur

Toute mise à jour du champ next quand la marque est posée échouera

Marquer un noeud

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



Extraire Reference & Marque

Public Object get(boolean[] marked);

Extraire Reference & Marque

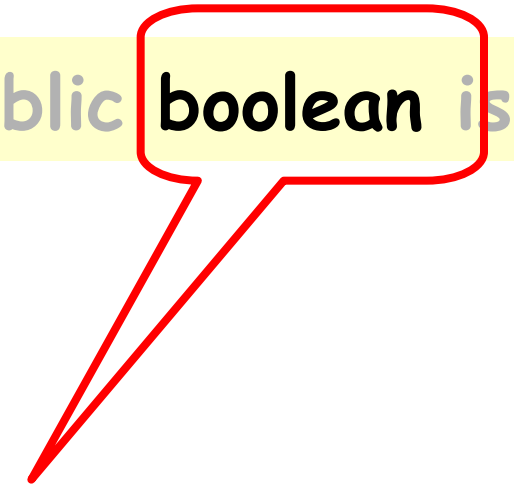
```
Public Object get(boolean[] marked);
```

Retourne la reference

Retourne la marque dans un tableau de boolean à l'indice 0 !!!

Extraire Reference & Marque

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



Valeur de la marque

Changement d'état

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

Changement d'état

Si cette référence est la référence courante

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

Et si cette marque est la marque courante

Changement d'état

…alors changer pour cette nouvelle référence …

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

… et cette nouvelle marque

Changement d'état

```
public boolean attemptMark(
  Object expectedRef,
  boolean updateMark);
```

Changement d'état

```
public boolean attemptMark(
  Object expectedRef,
  boolean updateMark);
```

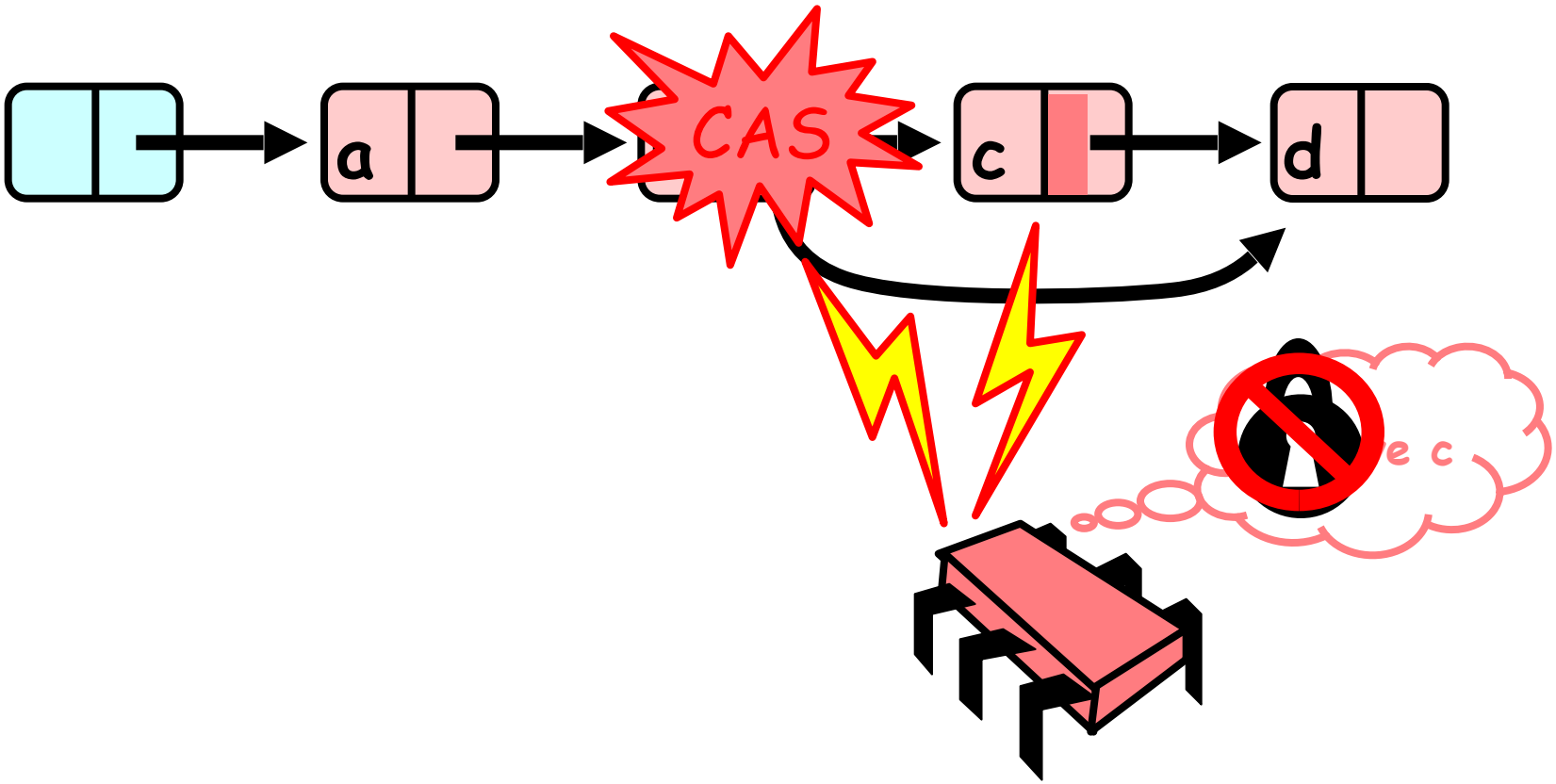
Si cette reference est la reference courante …

Changement d'état

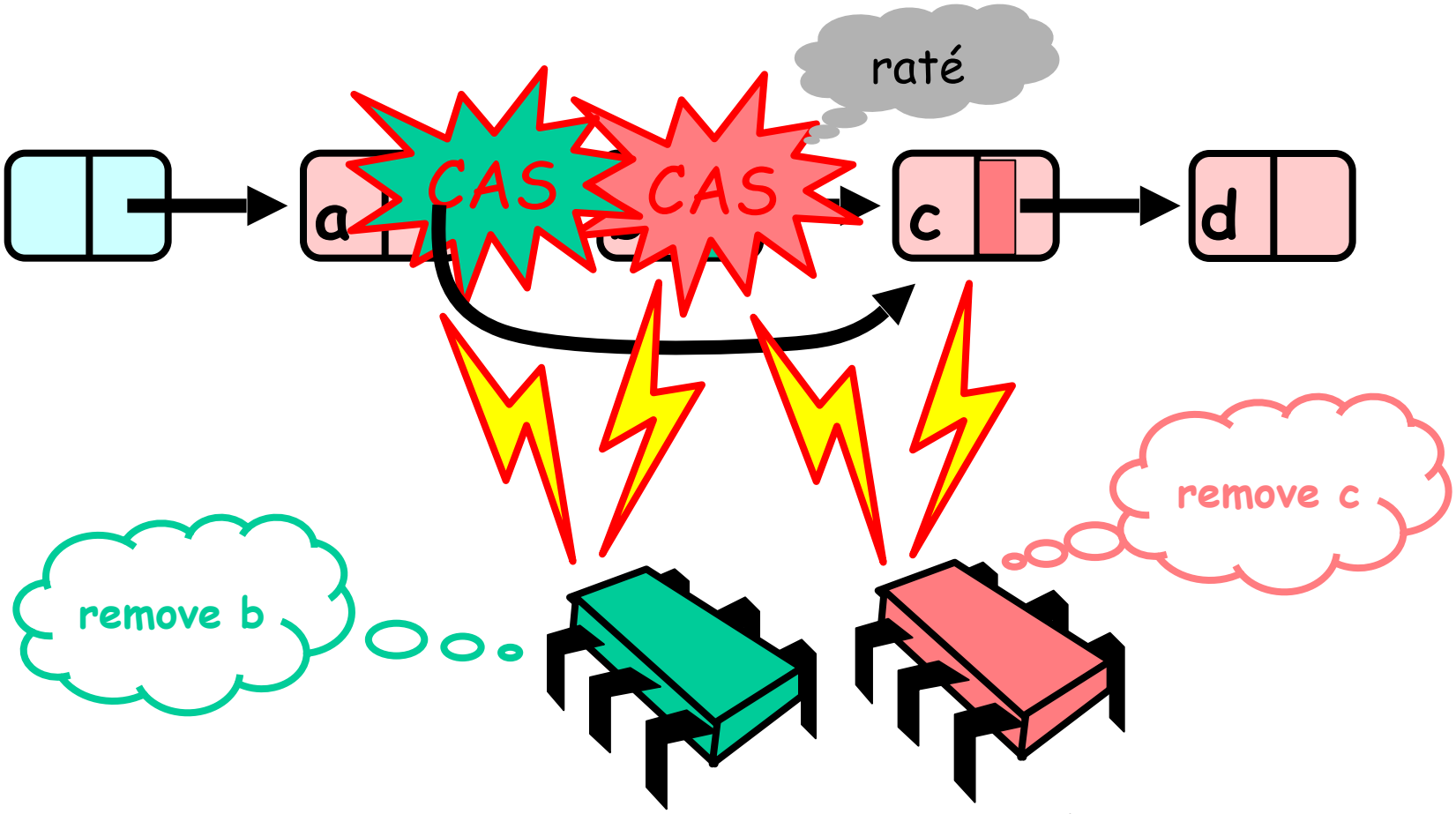
```
public boolean attemptMark(
  Object expectedRef,
  boolean updateMark);
```

.. alors changer pour cette nouvelle marque.

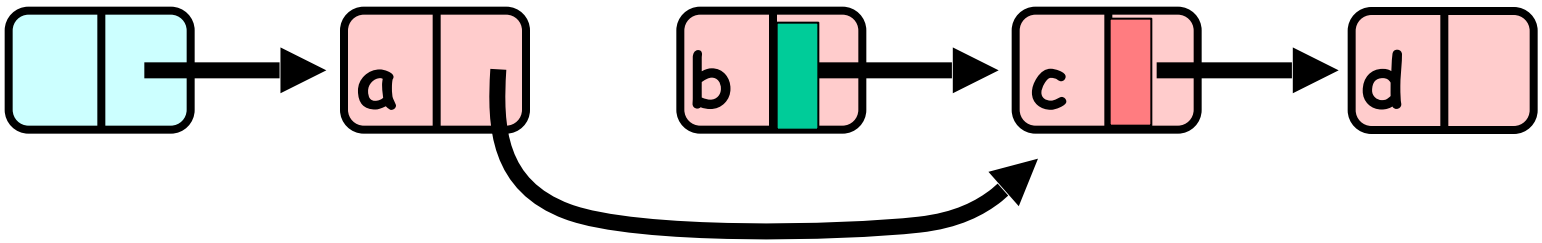
Enlever un noeud

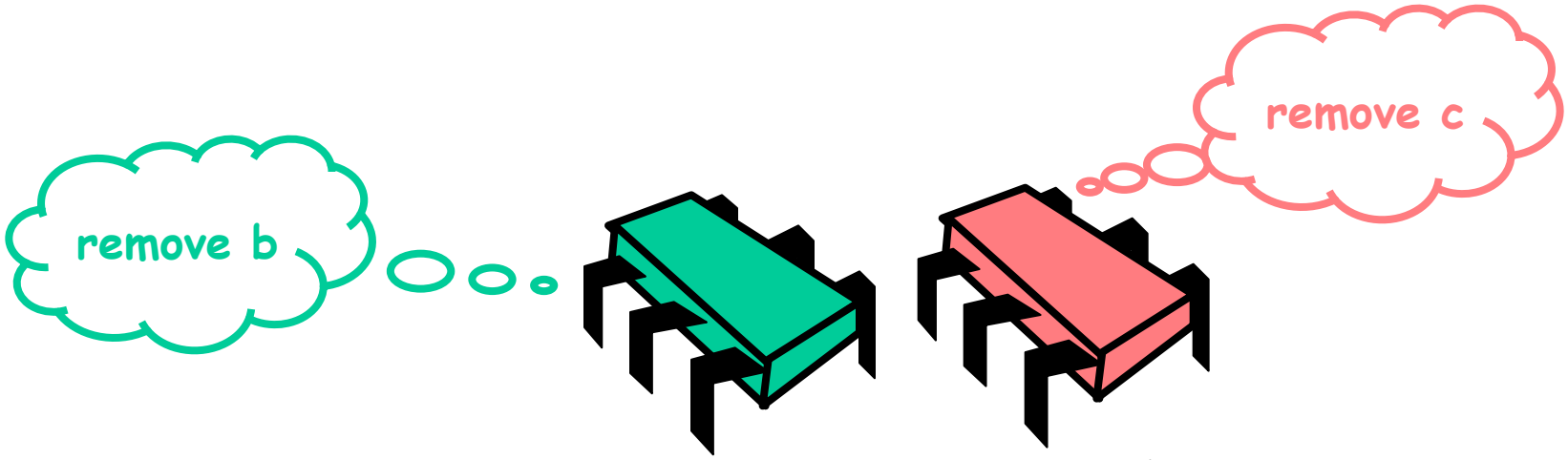


Enlever un noeud

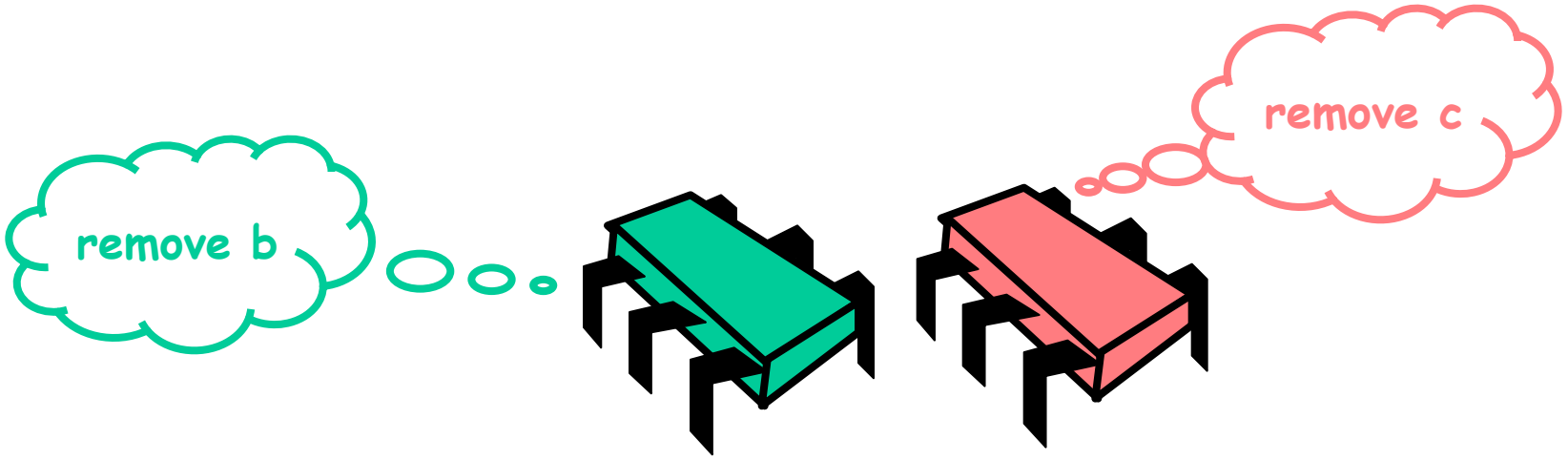


Enlever un noeud





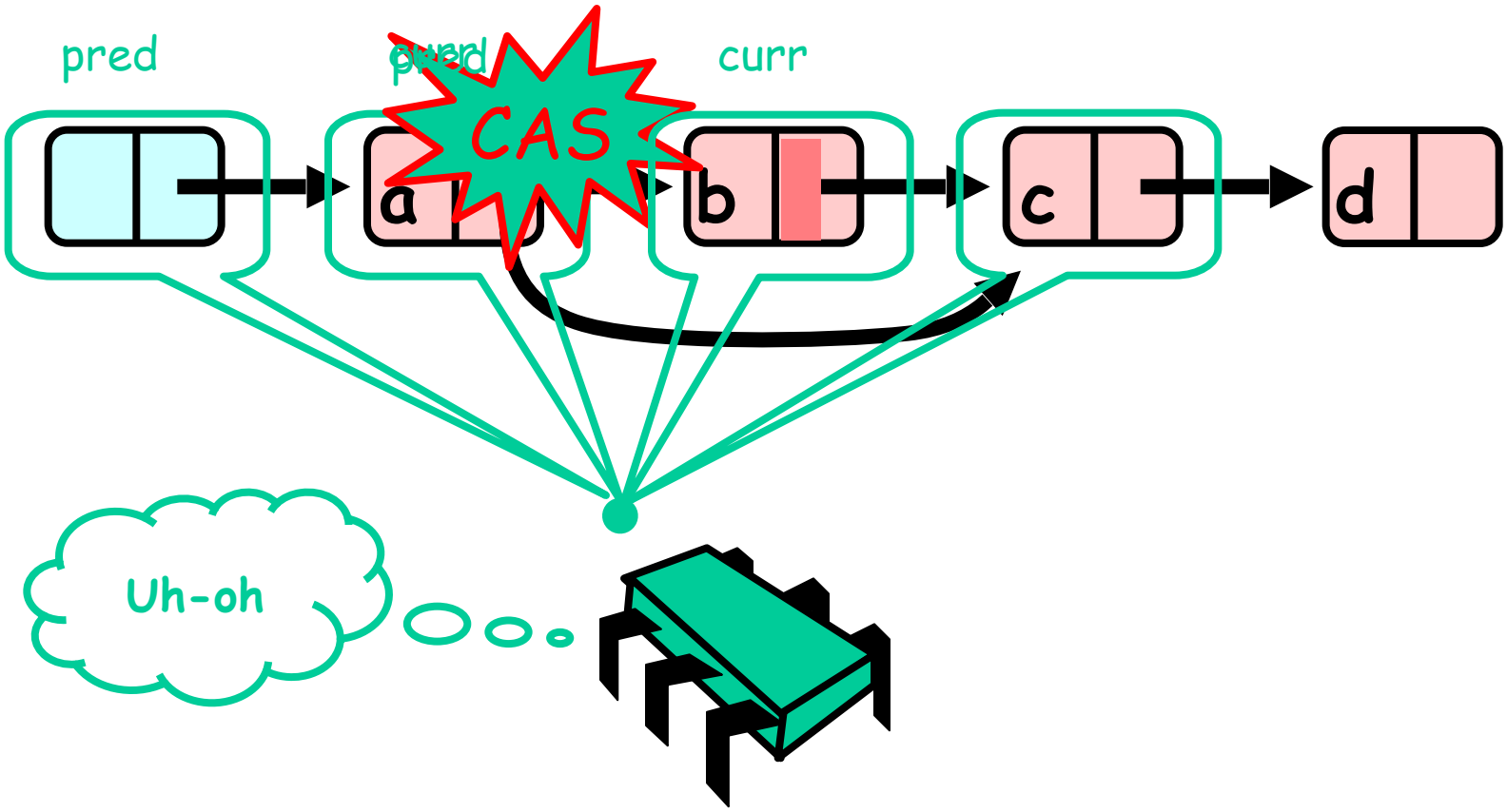
Enlever un noeud



Traverser la liste

- Q: que faire quand on trouve un noeud logiquement enlevé sur le chemin?
- R: finir le travail.
  - Modifier par CAS le champ next du prédécesseur
  - Et repeter si besoin

Traversée sans verrou (seulement Add et Remove)

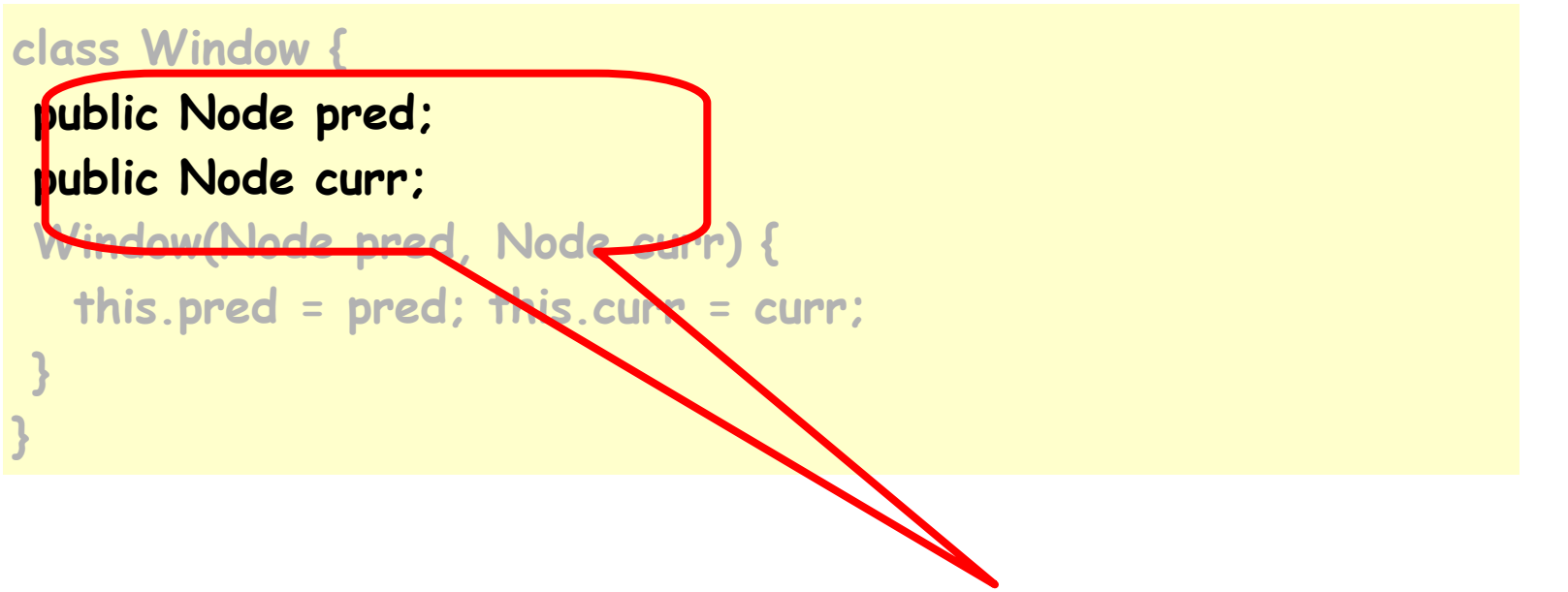


Window Class

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

Window Class

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



Regroupe pred et curr

Méthode Find

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

Méthode Find

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

Find retourne window

Méthode Find

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

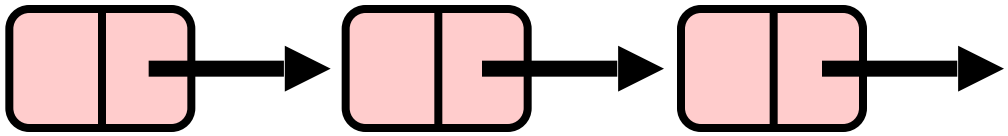
Extraction de pred et curr

Méthode Find

```
Window window = find(node,item);
```

dans la liste commencant a node

Effet de bord: fait les destructions physiques si besoin



pred

La plus grande clef inférieur à item

curr

La plus petite clef Supérieure ou égale à item

Remove

```
public boolean remove(T item) {
Boolean snip; int key = item.hashCode();
while (true) {
 Window window = find(head, key);
 Node pred = window.pred; Node 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

```
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;
}}}
```

On continue tant qu'on a pas reussi

Remove

```
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;
}}}
```

Trouver les voisins

Remove

```
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;
}}}
```

L'item n'est pas dans la liste

Remove

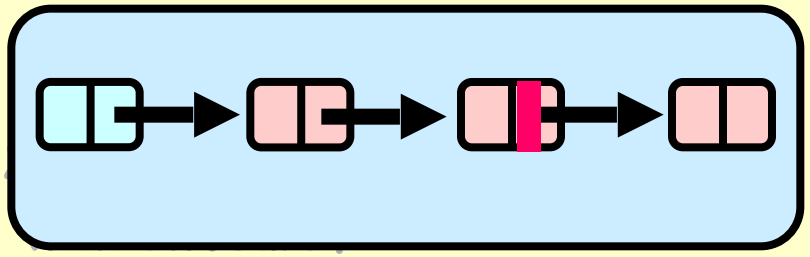
```
public boolean remove(T item) {
Boolean snip;
while (true) {
 Window window = find(head, key);
 Node pred = window.pred, curr = window.curr;
  if (curr.key != key) {
     return false;
  } else {
   Node succ = curr.next.getReference();
   snip = curr.next.attemptMark(succ, true);
   if (!snip) continue;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

On essaie de marquer le noeud comme enlevé

Remove

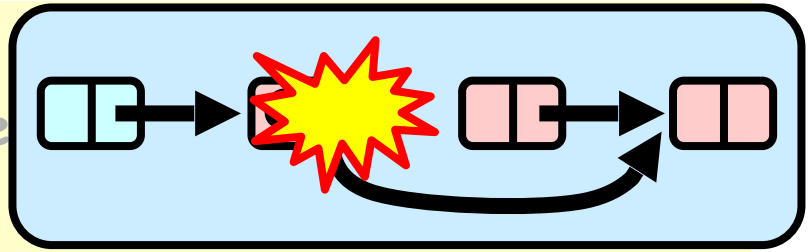
```
public boolean remove(T item) {
Boolean snip;
while (true) {
  Window window = find(head, key);
  Node pred = window.pred, curr =
  if (curr.key != key) {
    return false;
  } else {
    Node succ = curr.next.getReference();
    snip = curr.next.attemptMark(succ, true);
    if (!snip) continue;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

Si ca n'a pas marché on réessaie, sinon le travail est pratiquement fait



Remove

```
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;
}}}
```



On essaie d'avancer la reference (si retour faux c'est que quelqu'un d'autre l'a fait ou le fera).

Add

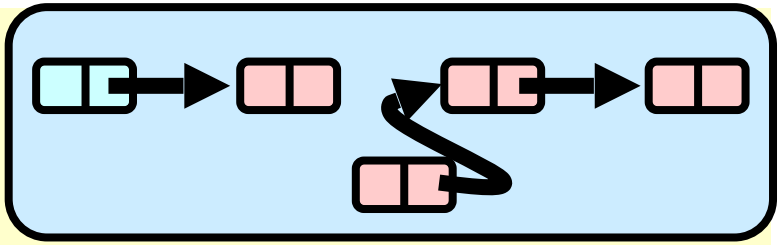
```
public boolean add(T item) {
 int key = item.hashCode();
 while (true) {
   Window window = find(head, key);
   Node pred = window.pred; Node curr = window.curr;
   if (curr.key == key) {
      return false;
   } else {
   Node node = new Node(item);
   node.next = new AtomicMarkableRef(curr, false);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

Add

```
public boolean add(T item) {
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 déjà la.

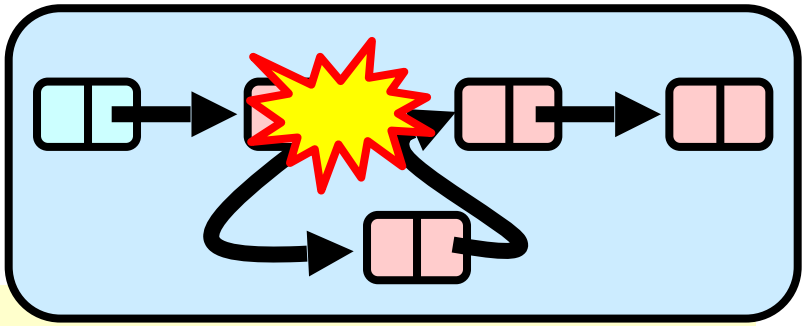
Add



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

Crée le nouveau noeud

Add



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

Installe le nouveau noeud sinon recommence

Wait-free Contains

```
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])
  }
```

Wait-free Contains

```
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]);
  }
```

Seule difference: pour tester si curr est marqué on appel curr.next.get et on vérifie que marked[0] est vrai

Find

```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null,t=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

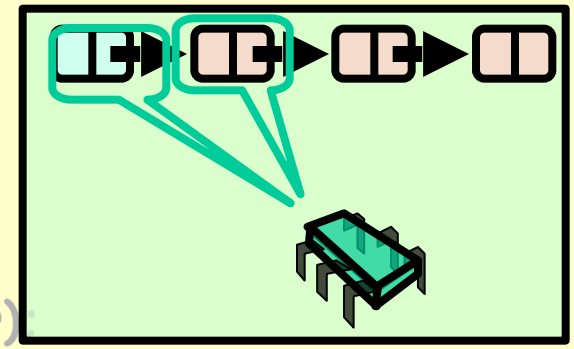
```
public Window find(Node head, int key) {
 Node pred = null, curr = null, succ = null;
 boolean[] marked = {false}; boolean snip;
 while (true) {
    pred = head;
    curr = pred.next.getReference();
    while (true) {
     succ = curr.next.get(marked);
     while (marked[0]) {
     …
     }
    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

Si la liste a change pendant le parcours , on recomence Lock-Free car on recommence si quelqu'un d'autre a progressé

Find

```
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;
    }
```

On commence head



Find

```
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;
    }
}
```

On parcours la liste

Find

```
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;
    }
```

ref du successeur et bit de marque

Find

```
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;
   }
```

On tente d'enlever les noeuds « a enlever » au passaage

Find

```
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;
    }
```

si curr key est >= retourne pred et curr

Find

```
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;
    }
```

Sinon avancer la fenêtre et refaire la boucle

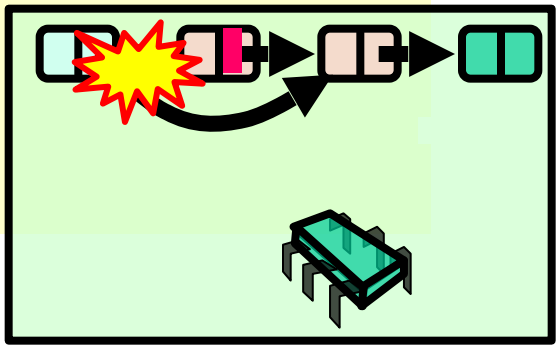
Find

```
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

On essaie d'enlever le noeud

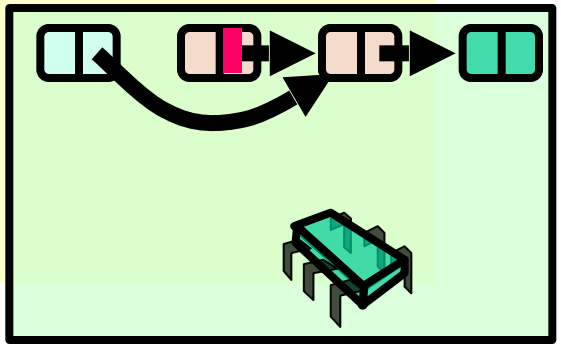
```
retry: while (true) {
    …
    while (marked[0]) {
      snip = pred.next.compareAndSet(curr, succ, false,
false);
      if (!snip) continue retry;
      curr = succ;
      succ = curr.next.get(marked);
    }
…
```



Find

Si le champ suivant du predecesseur a changé on doit refaire tout le parcours

```
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

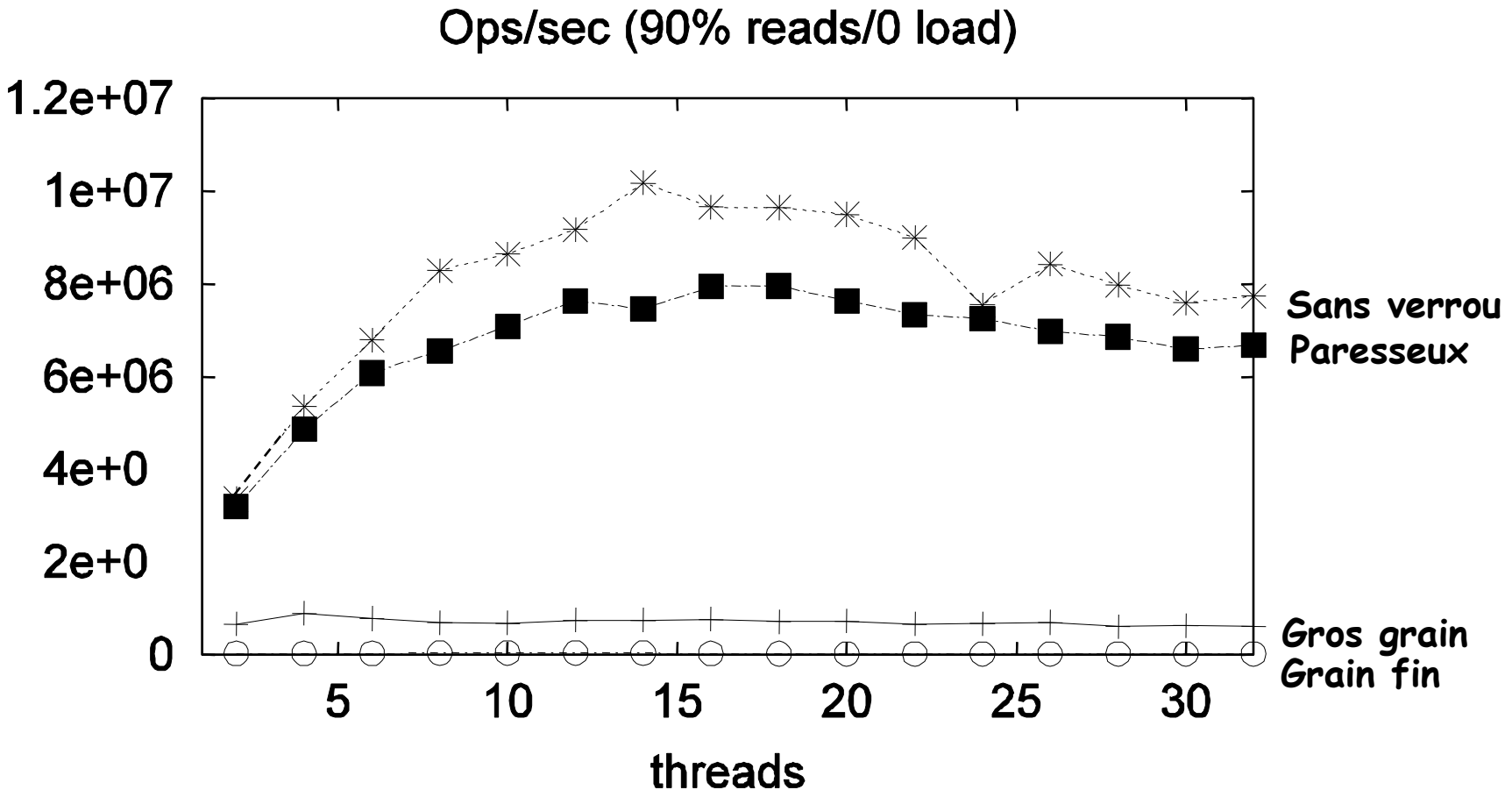
Sinon on avance pour verifier si le noeud suivant est detruit

```
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);
   }
…
```

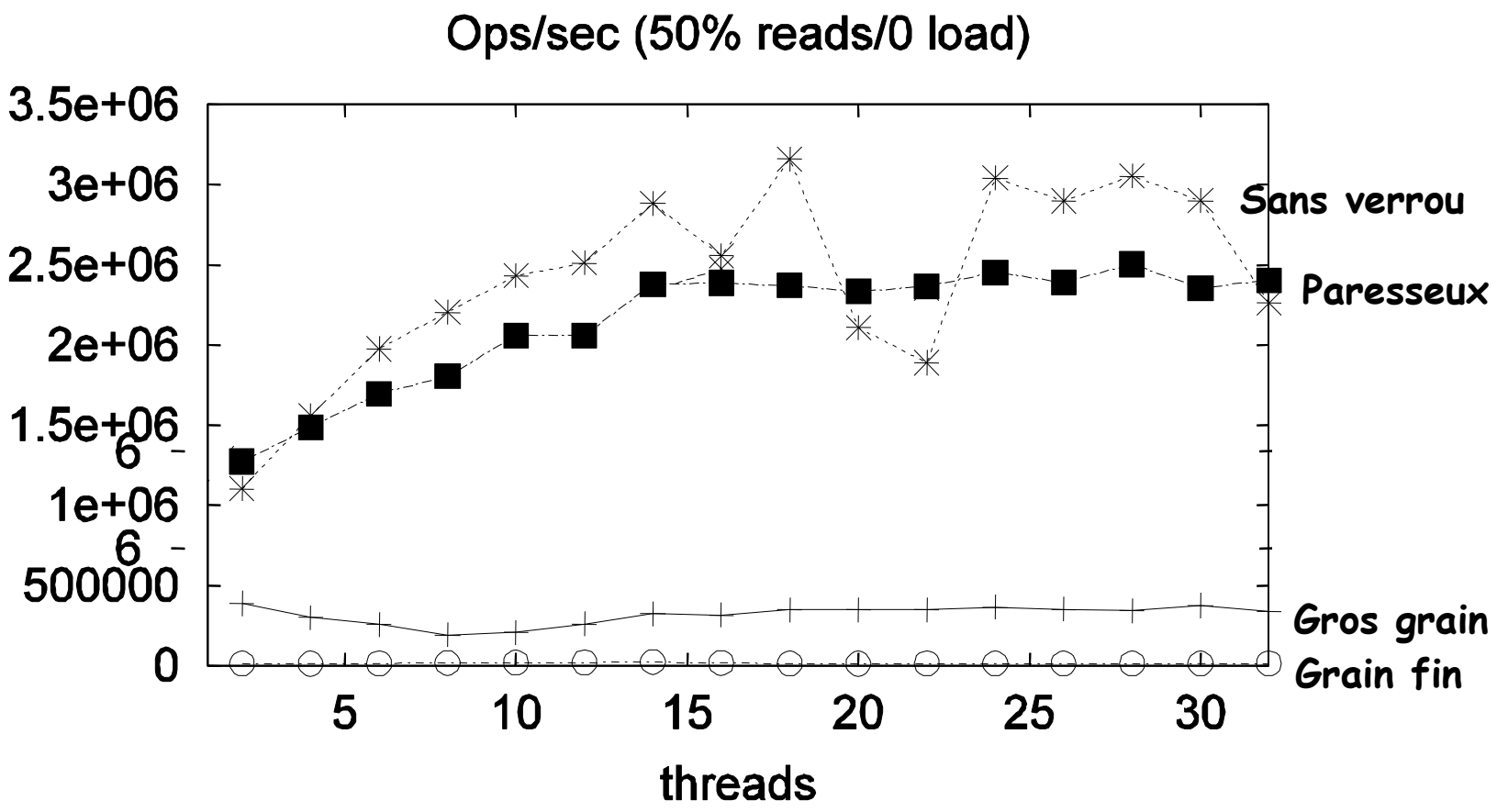
Implémentation sans verrou

- Elimine entièrement les verrous
- contains() wait-free ; add() et remove() non blocking
- Utilise compareAndSet()

Avec un fort taux de Contains



Avec un faible taux de contains



""Avec ou sans Verrou""

- Blocking vs. Non-blocking: points de vue extrémistes des deux côtés
- Réponse: tenter le compromis, allie verrouillage et non-blocage

Exemple : la liste paresseuse combine le blocage add() et remove() et un wait-free contains()
N'oubliez pas : le blocage/non-blocage est une propriété d'une méthode