Практические построения на списках

Roman Elizarov[1] Nikita Koval[2]

[1]Kotlin Team Lead, JetBrains
elizarov@gmail.com

[2]Researcher, JetBrains
PhD student, IST Austria
ndkoval@ya.ru

ИТМО 2018

План

1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

План

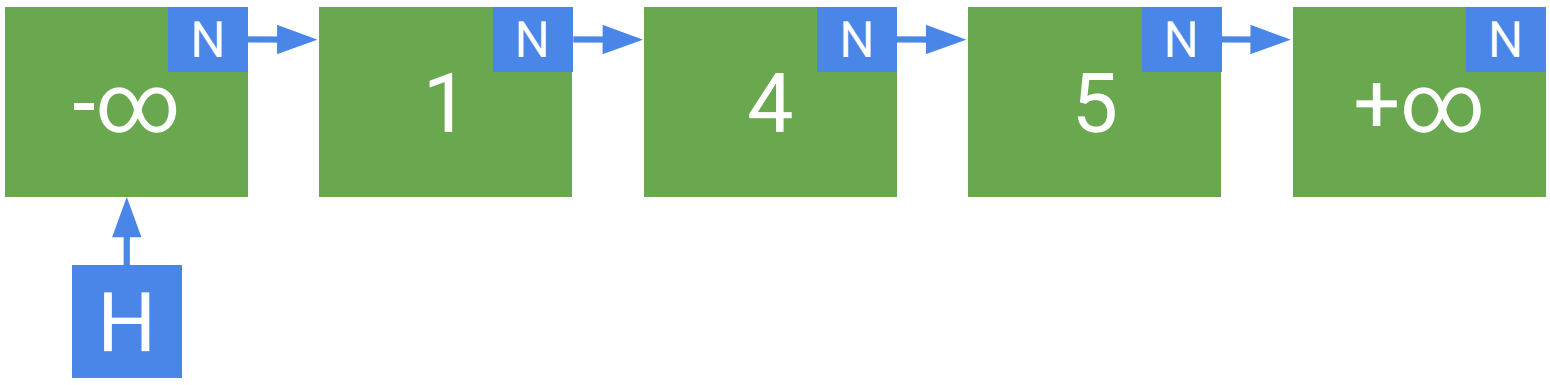
1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

Множество

```
interface Set {
    fun add(key: Int)
    fun contains(key: Int): Boolean
    fun remove(key: Int)
}
```

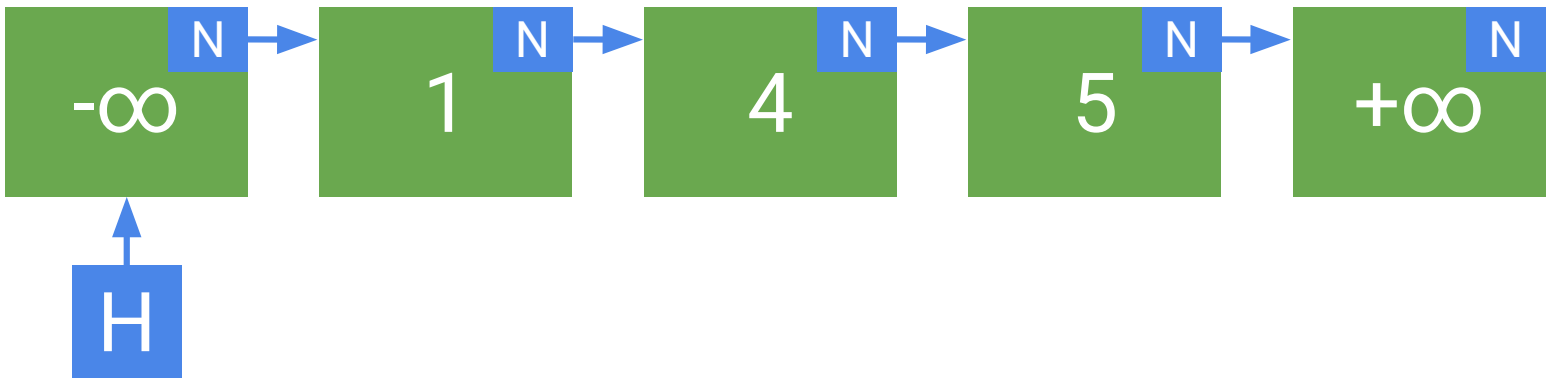
Односвязный список

Элементы упорядочены по возрастанию

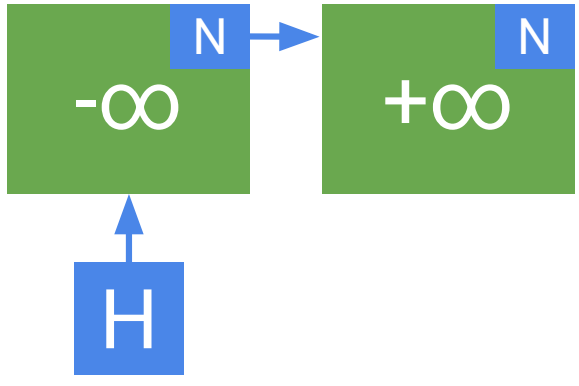


Односвязный список

Элементы упорядочены по возрастанию



Пустой список состоит из двух граничных элементов



Односвязный список: алгоритм

- Элементы упорядочены по возрастанию
- Ищем окно $(\texttt{cur}, \texttt{next})$, что $\texttt{cur.KEY} < k \leq \texttt{next.KEY}$ и $\texttt{cur}.N = \texttt{next}$
- Искомый элемент будет в `next`
- Новый элемент добавляем между `cur` и `next`

Односвязный список: псевдокод (1)

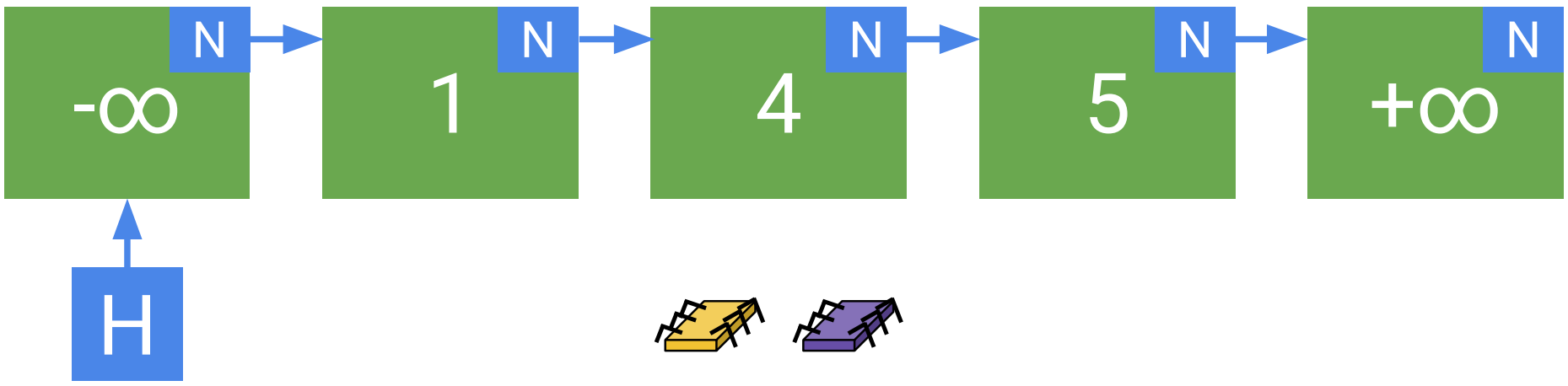
```
class Node(var N: Node, val key: Int)

val head = Node(−∞, Node(∞, null))

fun findWindow(key: Int): (Node, Node) {
  cur := head, next := cur.N
  while (next.key < key) {
    cur = next
    next = cur.N
  }
  return (cur, next)
}
```

Односвязный список: псевдокод (2)

```
fun contains(key: Int): Boolean {
  (cur, next) := findWindow(key)
  return next.key == key
}

fun add(key: Int) {
  (cur, next) := findWindow(key)
  if (next.key != key)
    cur.N = Node(key, next)
}

fun remove(key: Int) {
  (cur, next) := findWindow(key)
  if (next.key == key)
    cur.N = next.N
}
```

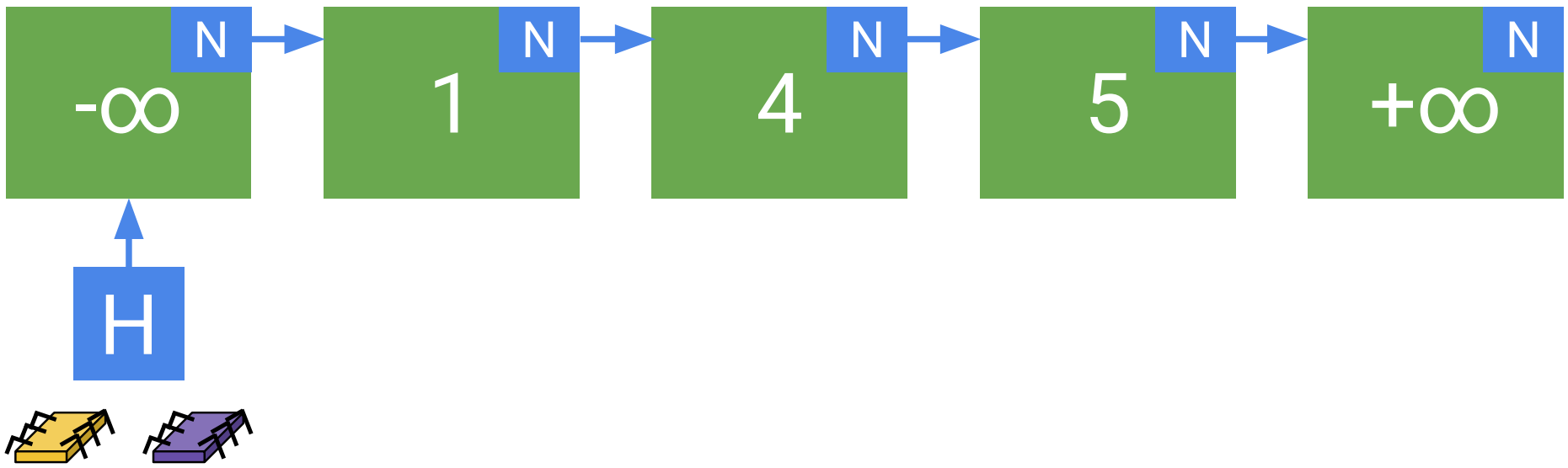
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



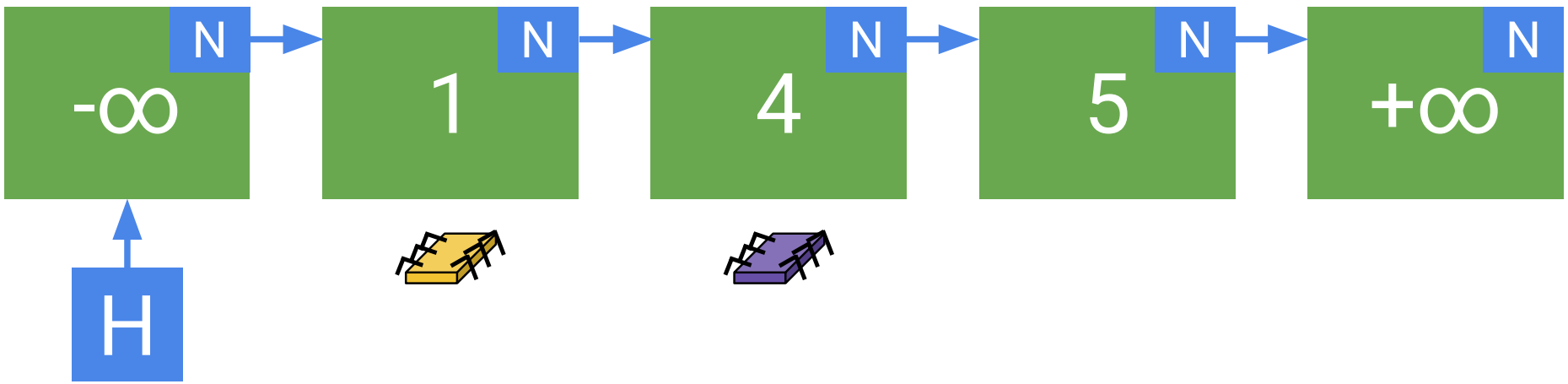
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



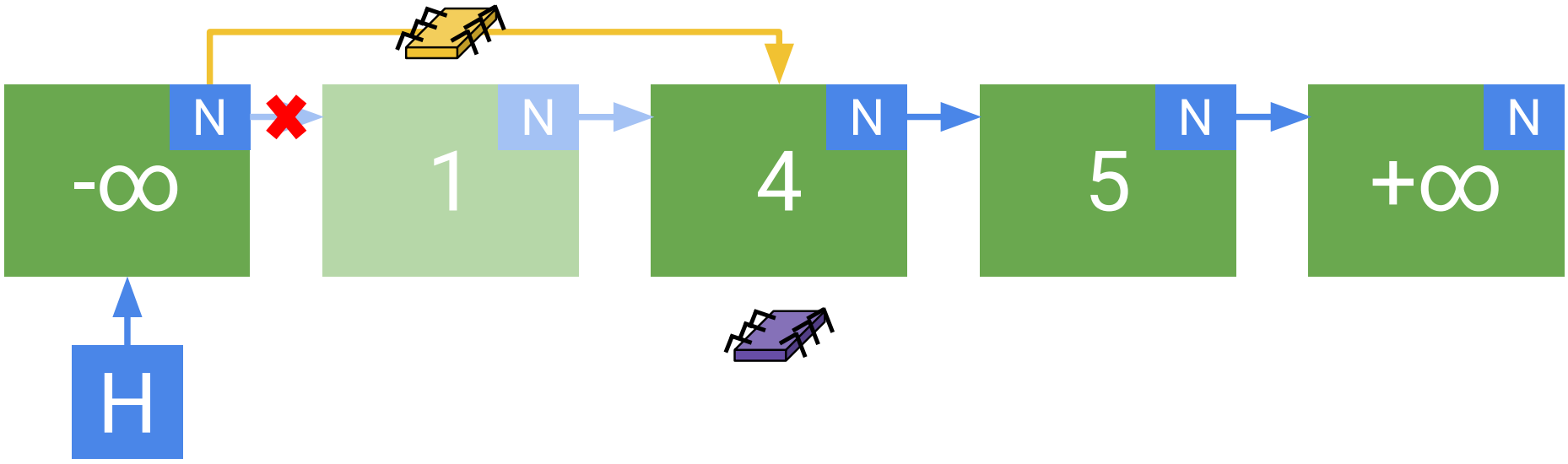
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



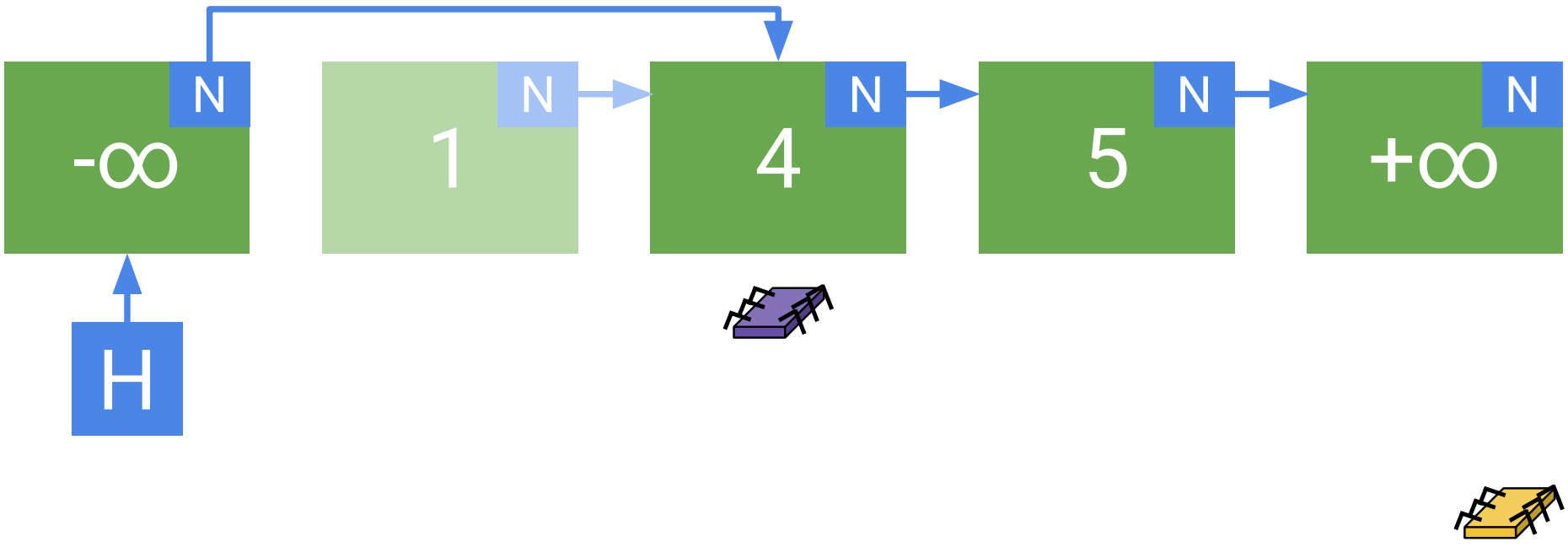
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



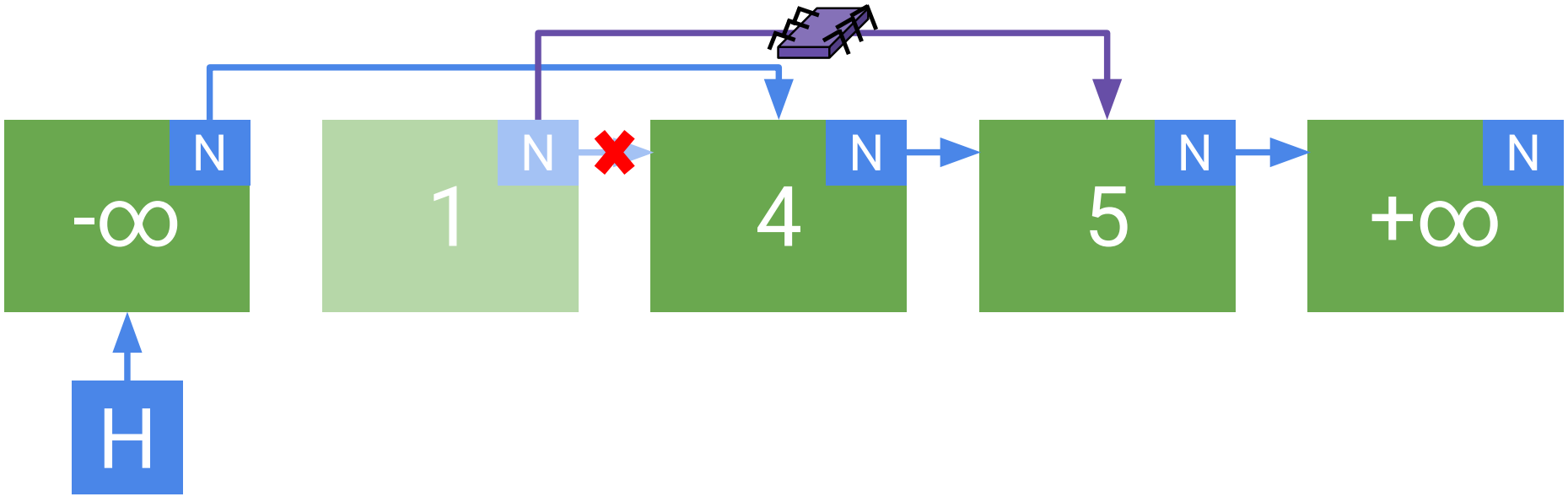
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



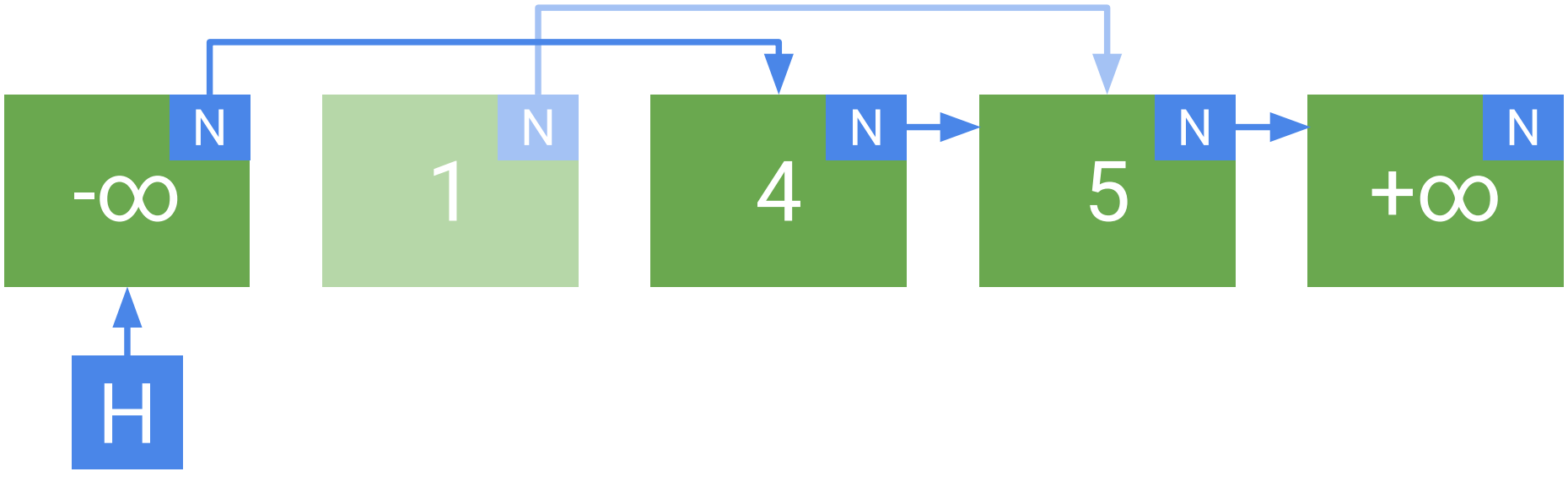
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



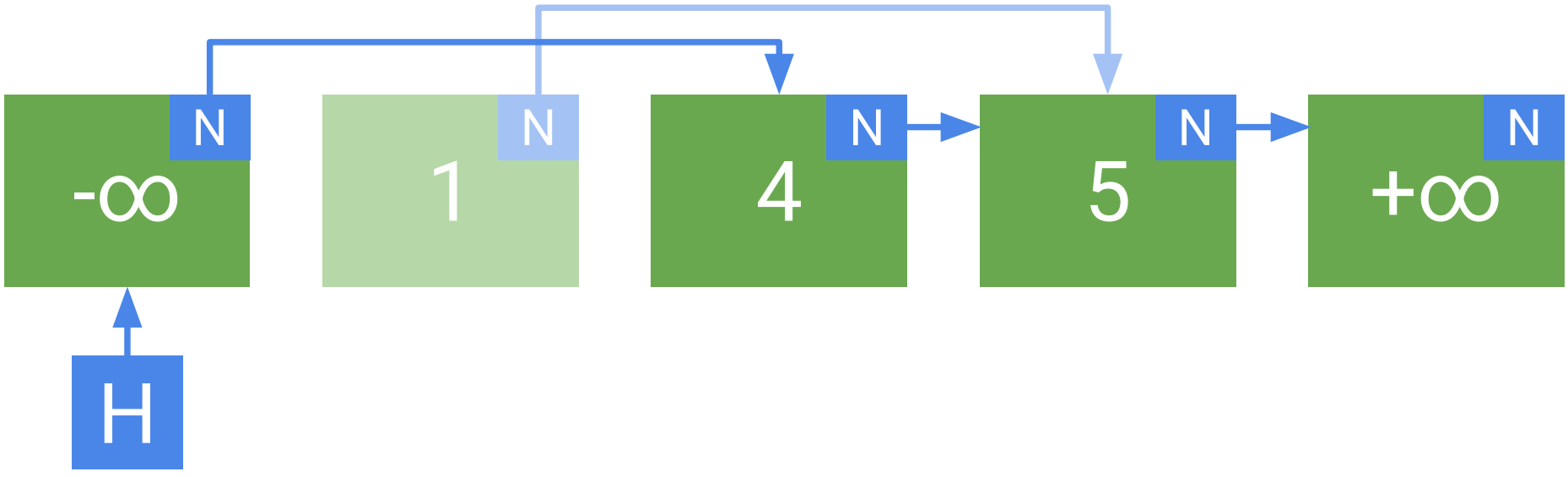
Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



Проблема

Жёлтый удаляет «1», фиолетовый удаляет «4»



У фиолетового ничего не вышло!

План

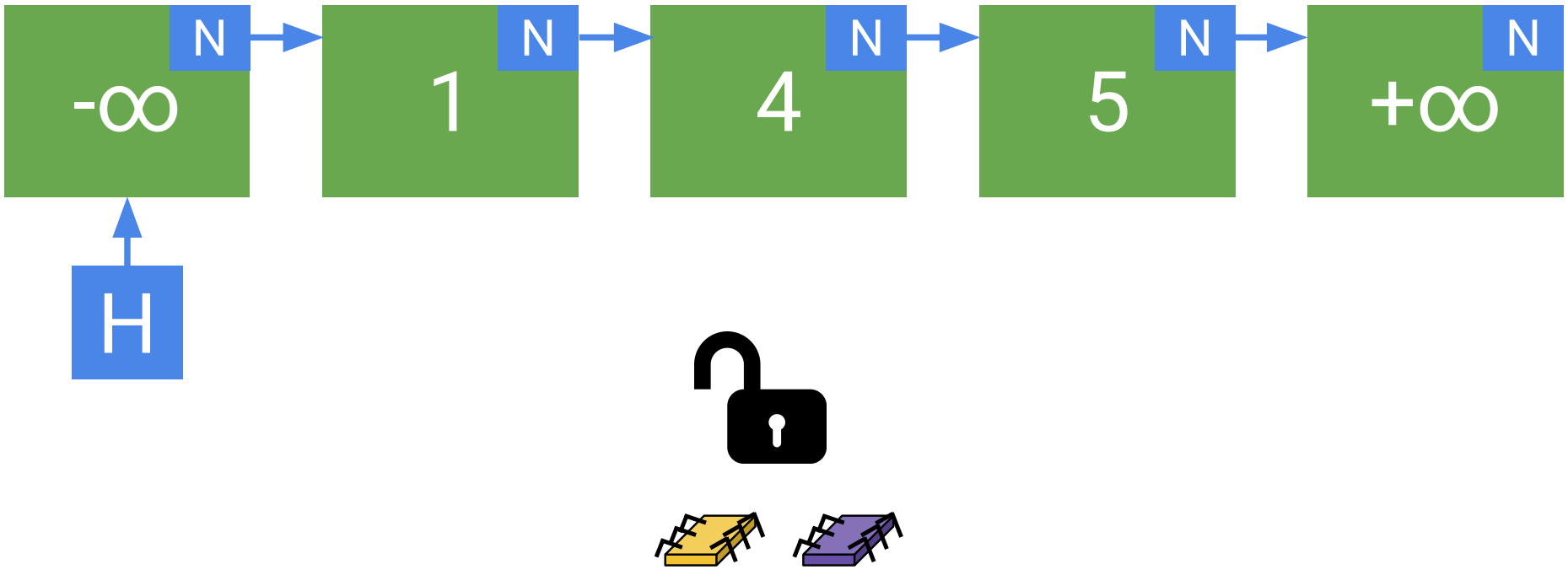
1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

Грубая синхронизация

- Coarse-grained locking
- Используем общую блокировку для всех операций
- $\Rightarrow$ обеспечиваем последовательное исполнение
- В Java для этого можно использовать $\mathrm{synchronized}$ или $\mathrm{j.u.c.locks.ReentrantLock}$

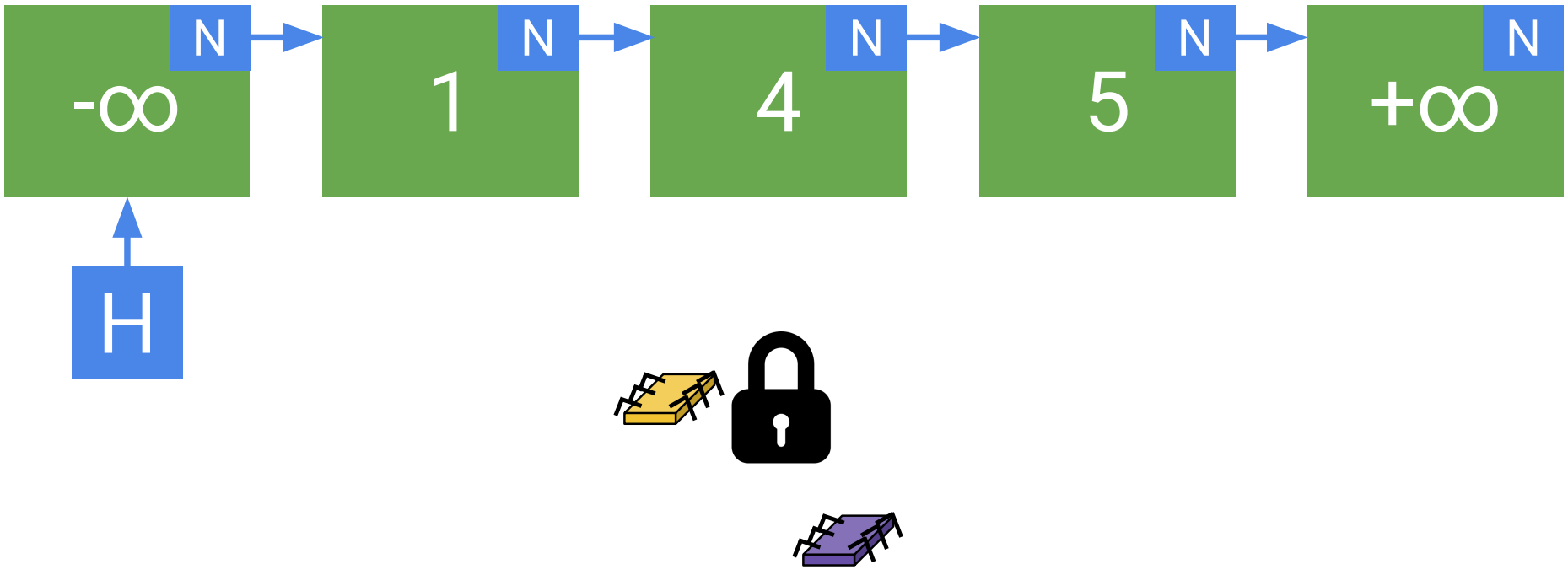
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



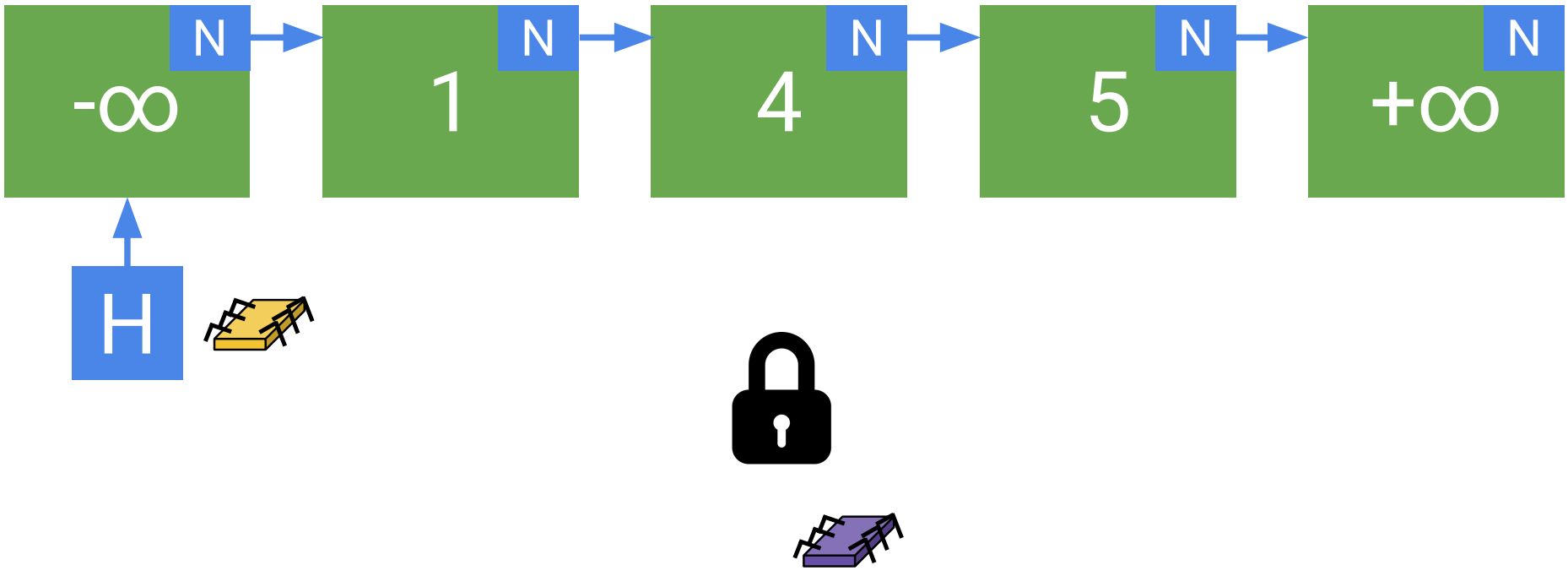
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



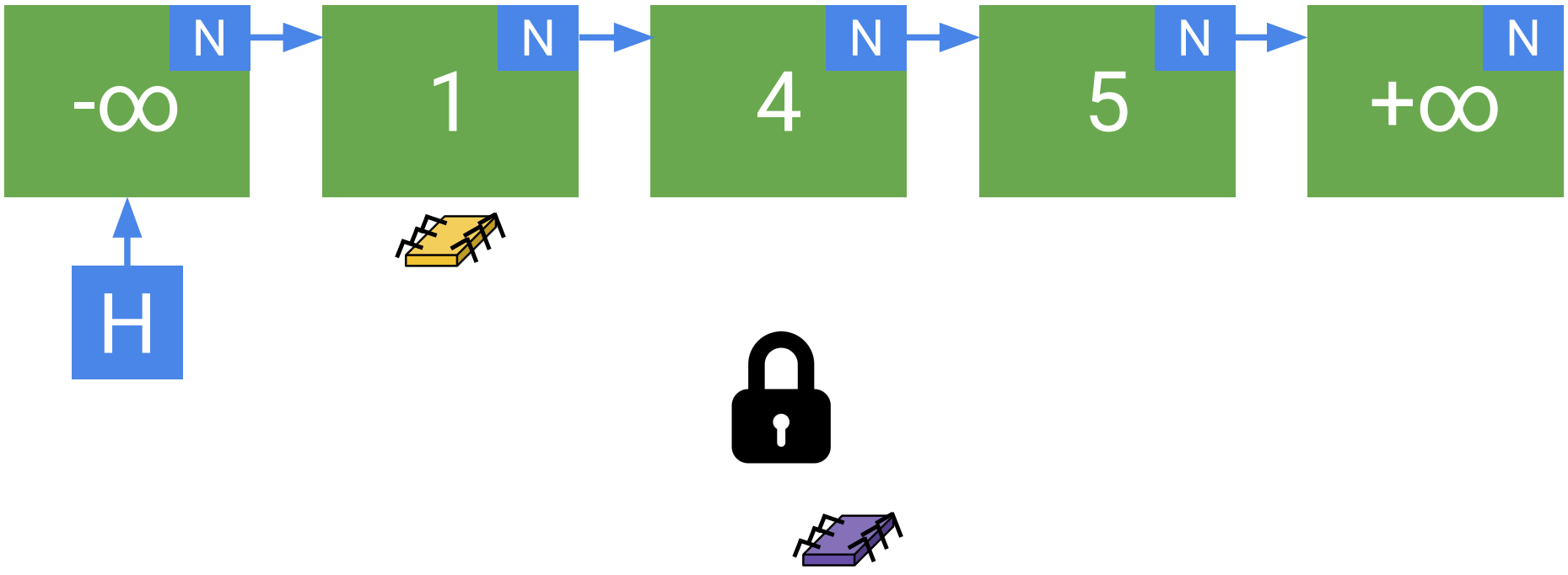
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



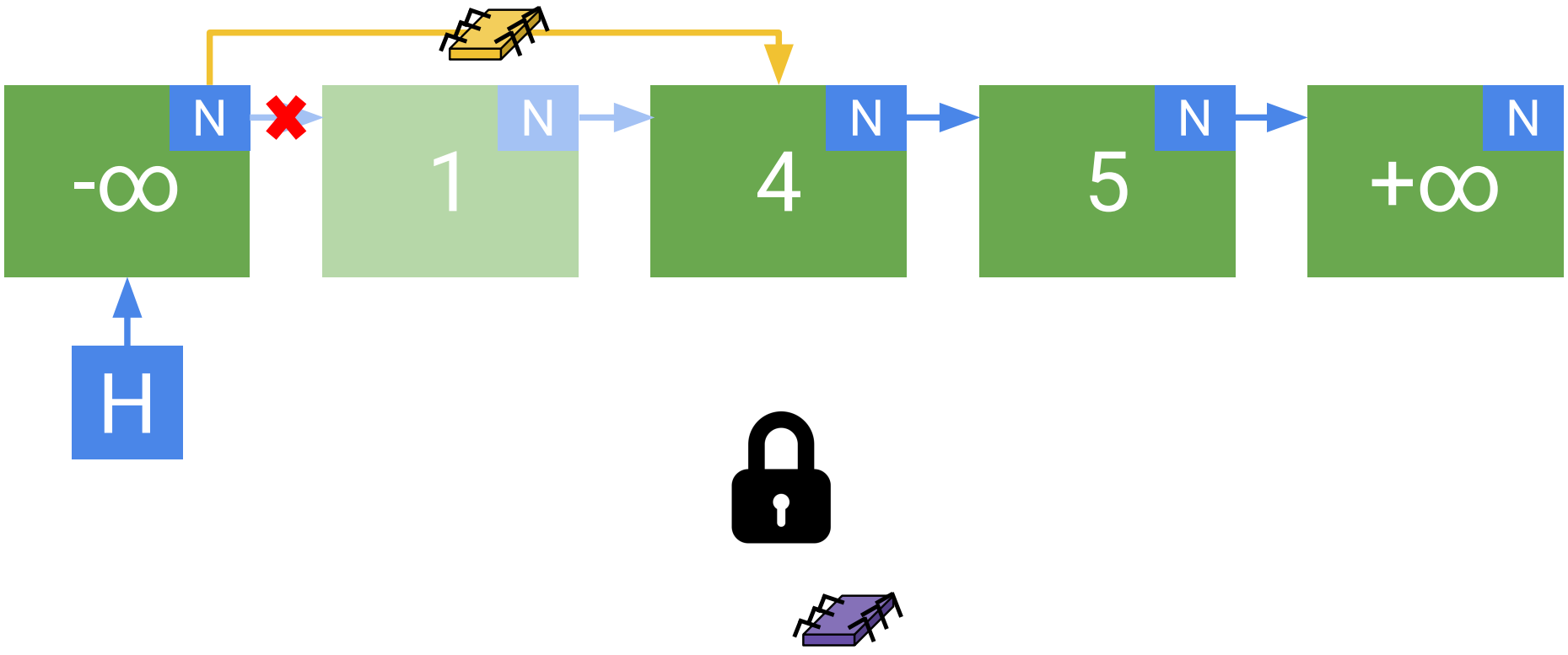
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



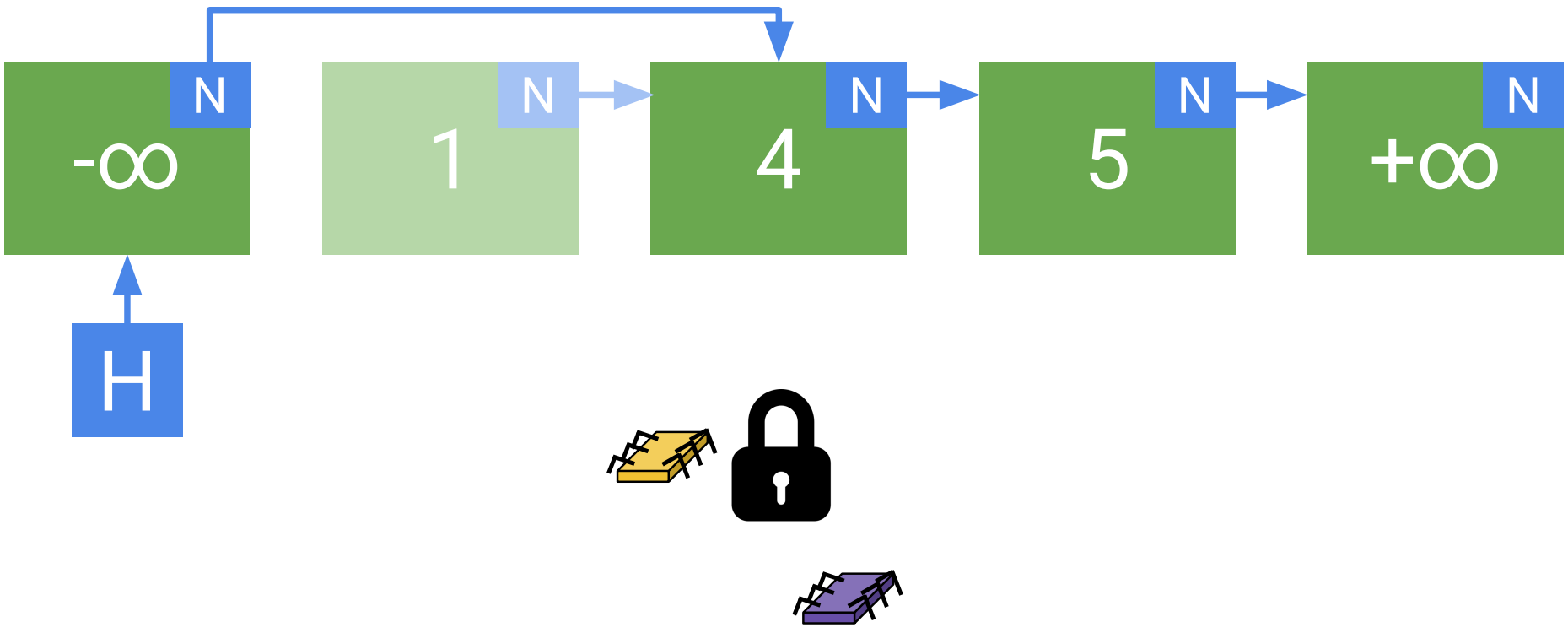
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



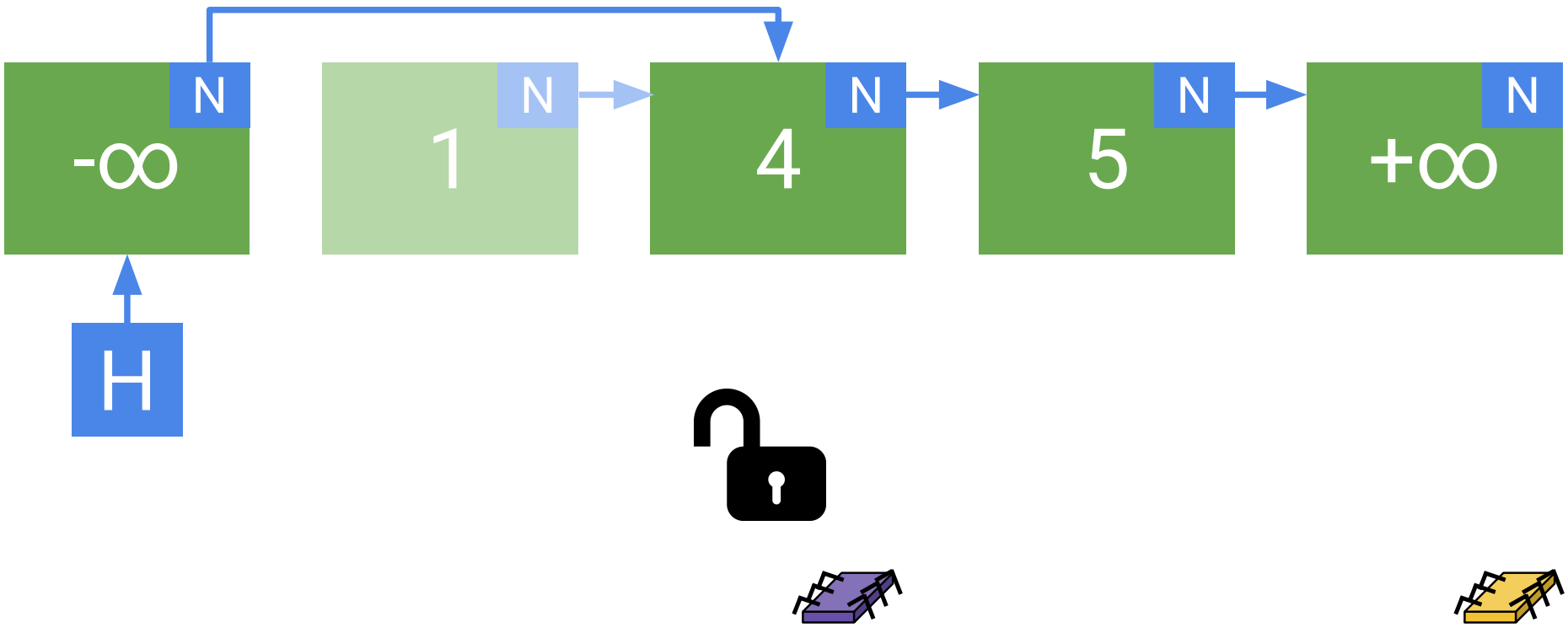
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



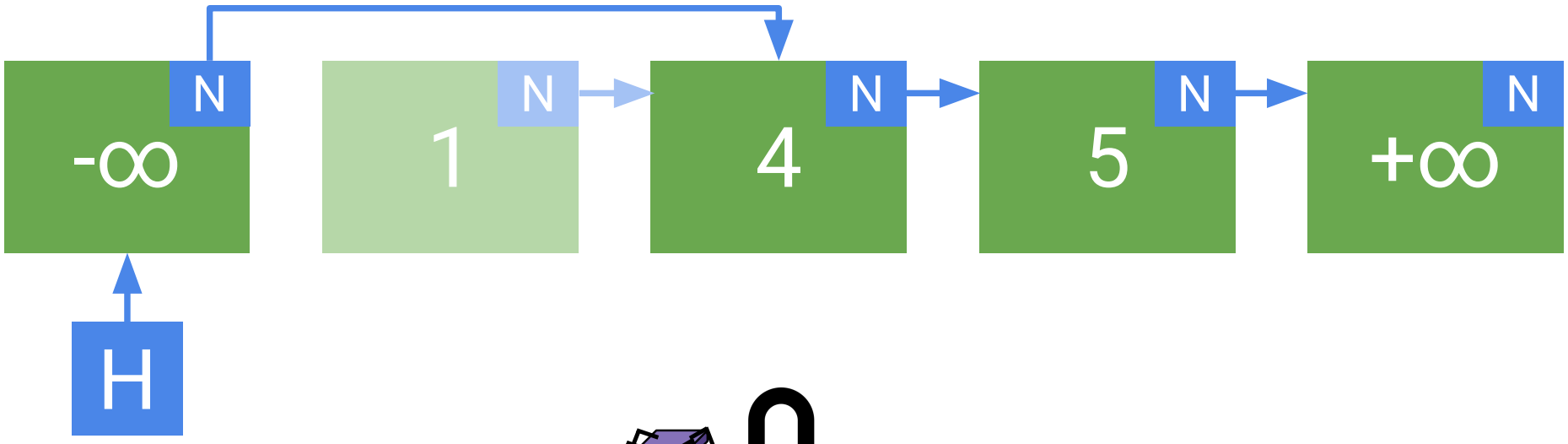
Пример

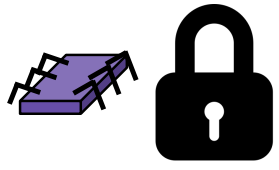
Жёлтый удаляет «1», фиолетовый удаляет «4»



Пример

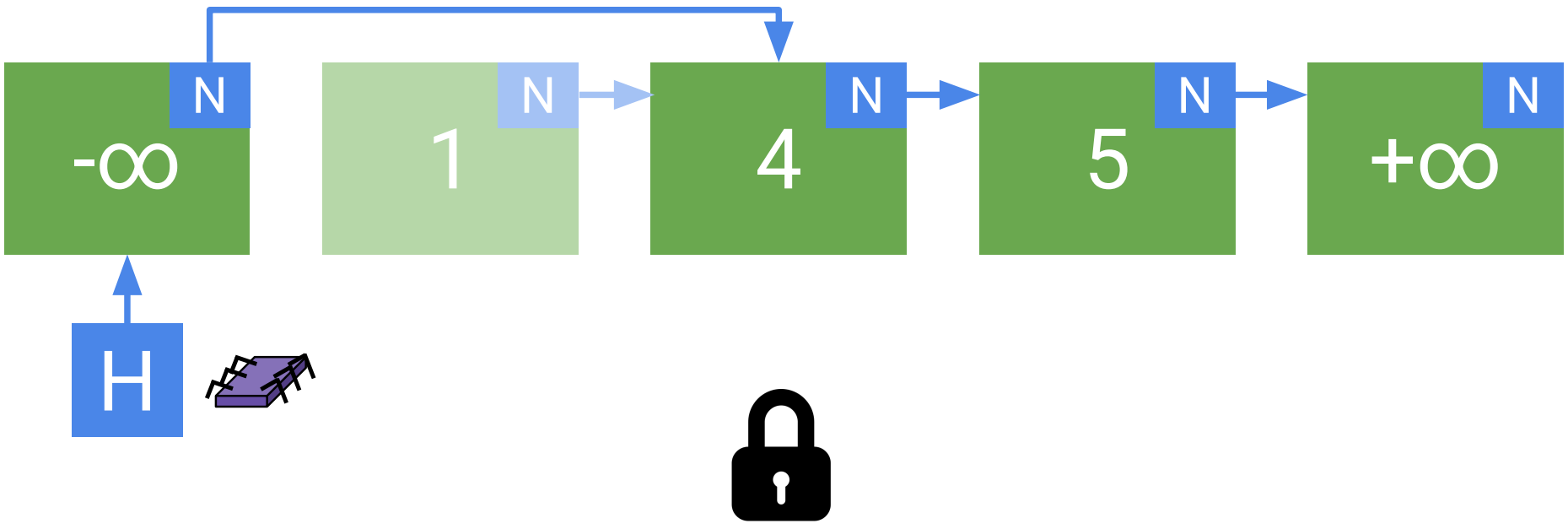
Жёлтый удаляет «1», фиолетовый удаляет «4»





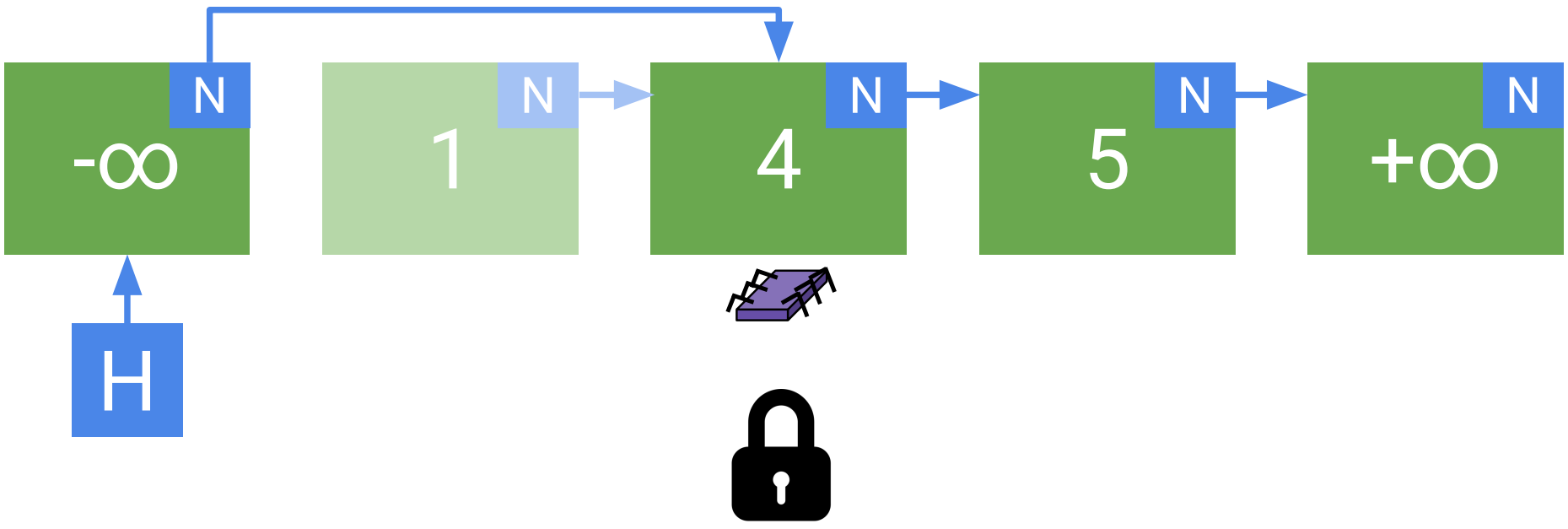
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



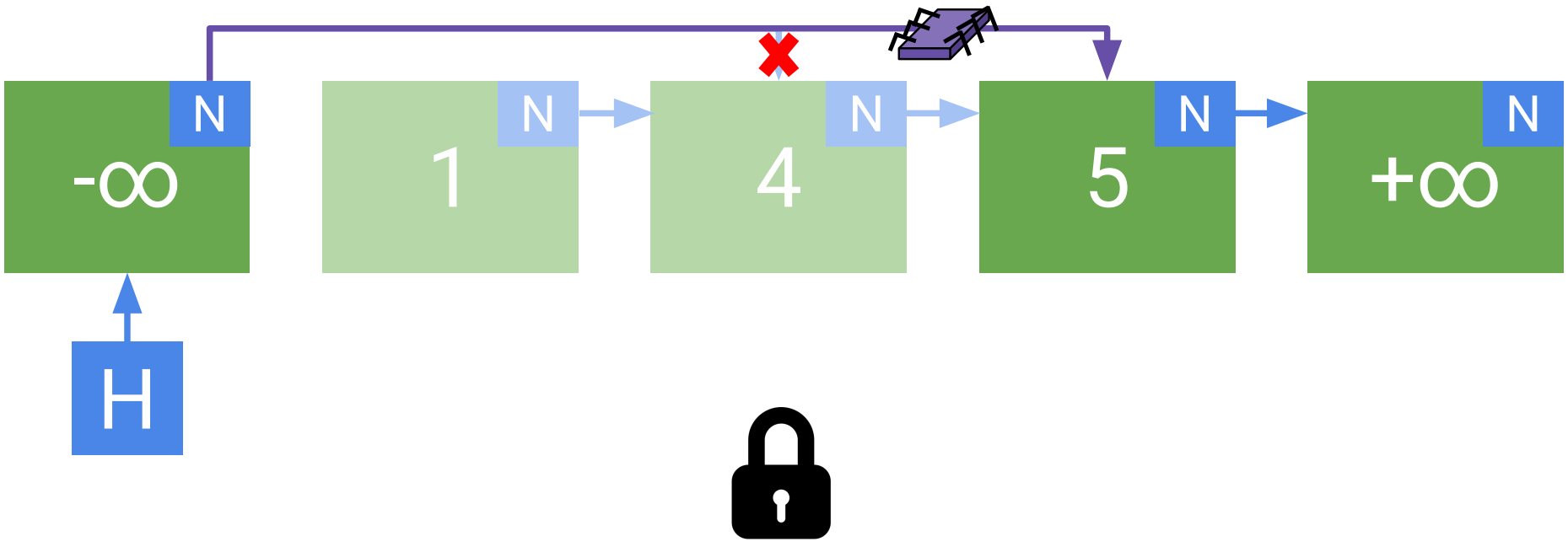
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



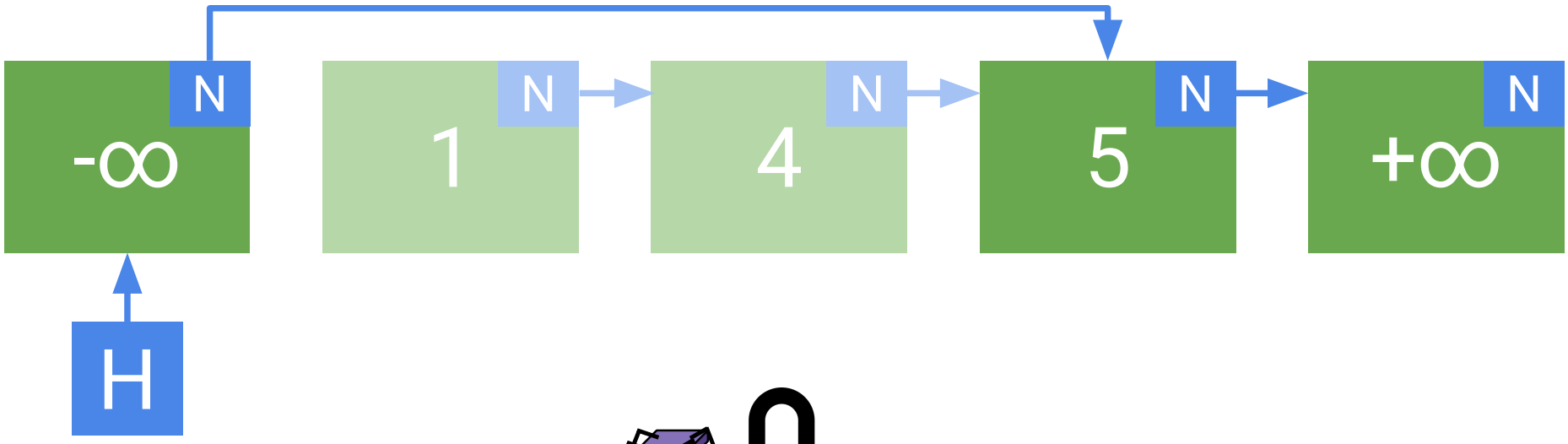
Пример

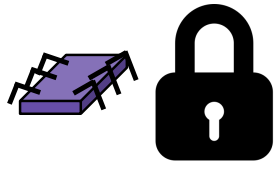
Жёлтый удаляет «1», фиолетовый удаляет «4»



Пример

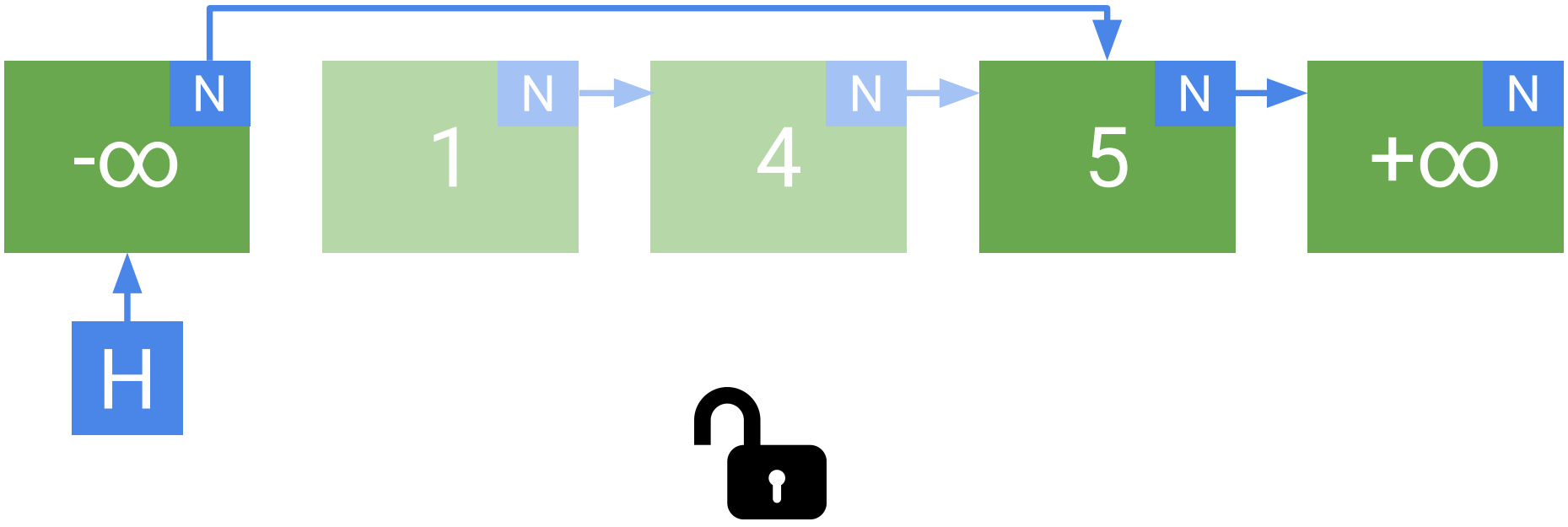
Жёлтый удаляет «1», фиолетовый удаляет «4»





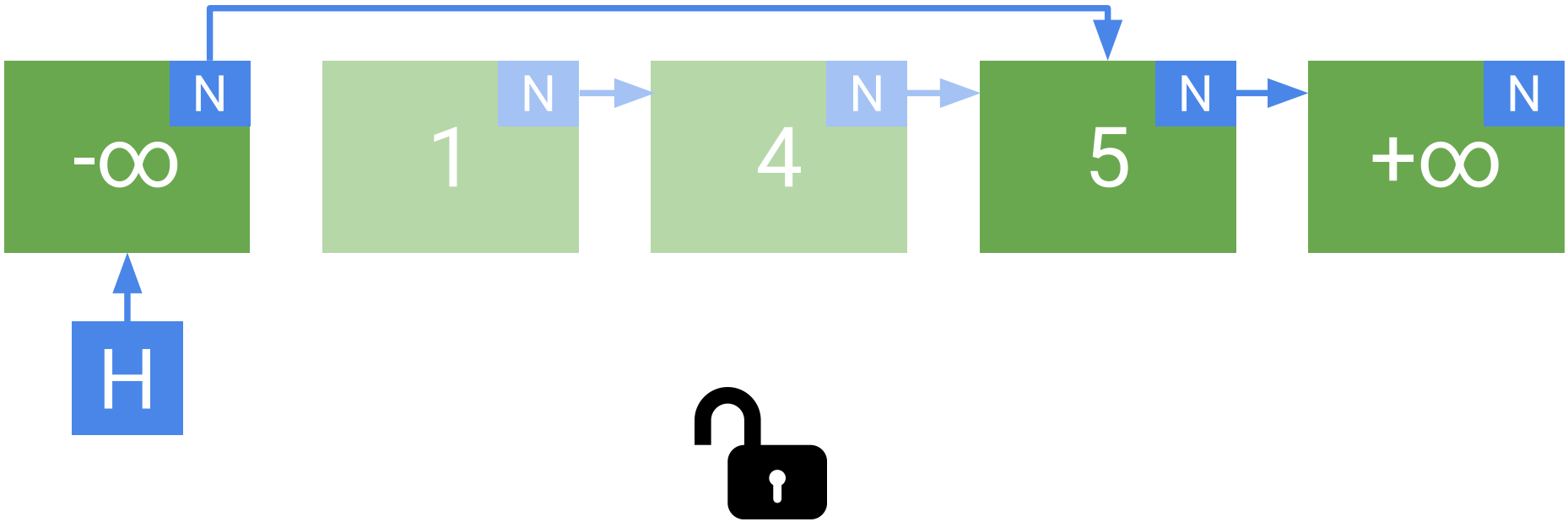
Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



Пример

Жёлтый удаляет «1», фиолетовый удаляет «4»



На этот раз удалили оба элемента!

Грубая синхронизация: псевдокод

```
fun contains(key: Int): Boolean = synchronized {
  (cur, next) := findWindow(key)
  return next.key == key
}

fun add(key: Int) = synchronized {
  (cur, next) := findWindow(key)
  if (next.key != key)
    cur.N = Node(key, next)
}

fun remove(key: Int) = synchronized {
  (cur, next) := findWindow(key)
  if (next.key == key)
    cur.N = next.N
}
```

План

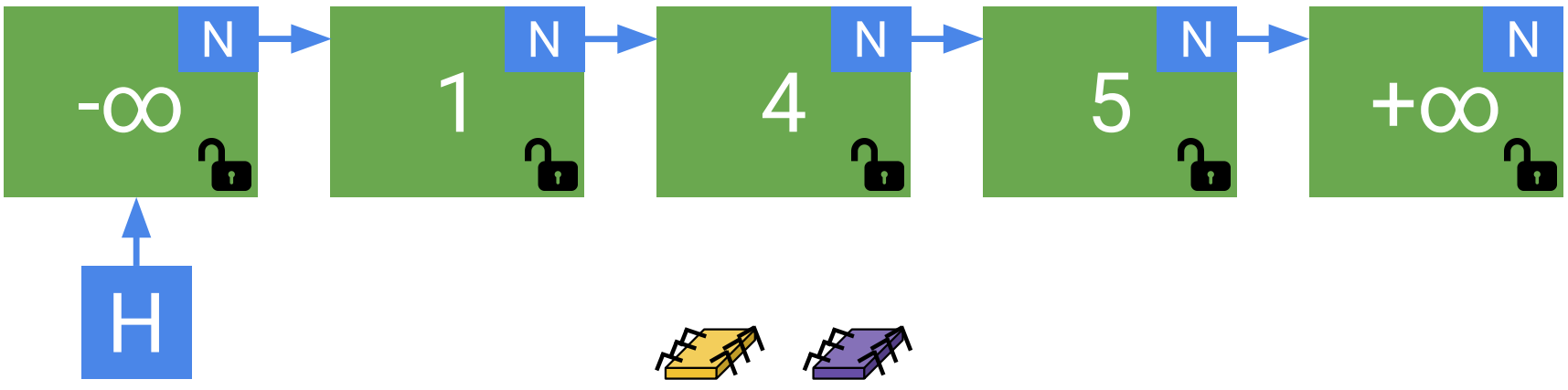
1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

Тонкая синхронизация

- Fine-Grained locking
- Своя блокировка на каждый элемент
- При поиске окна держим блокировку на текущий и следующий элементы

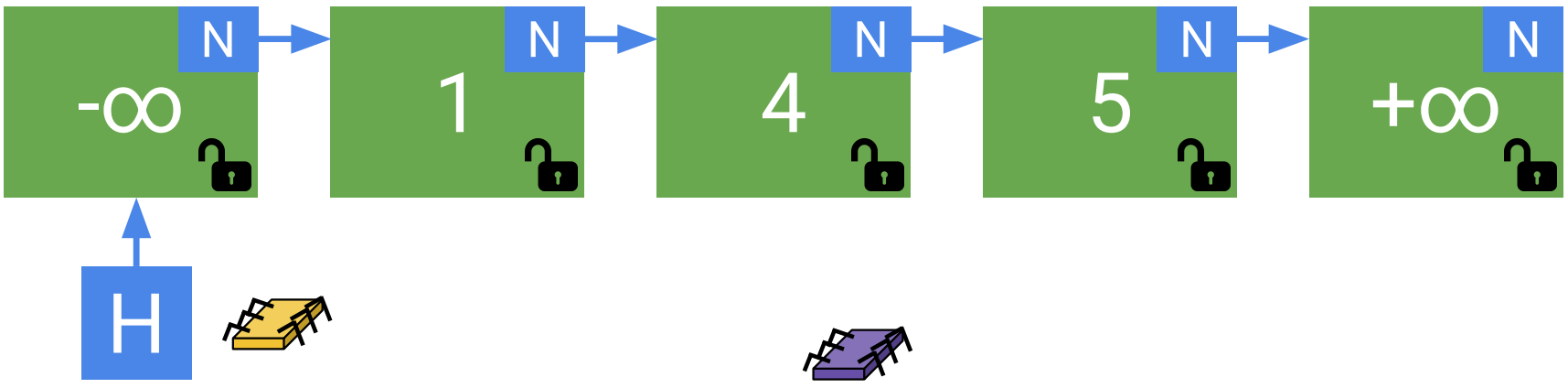
Пример

Жёлтый удаляет «4», фиолетовый удаляет «5»



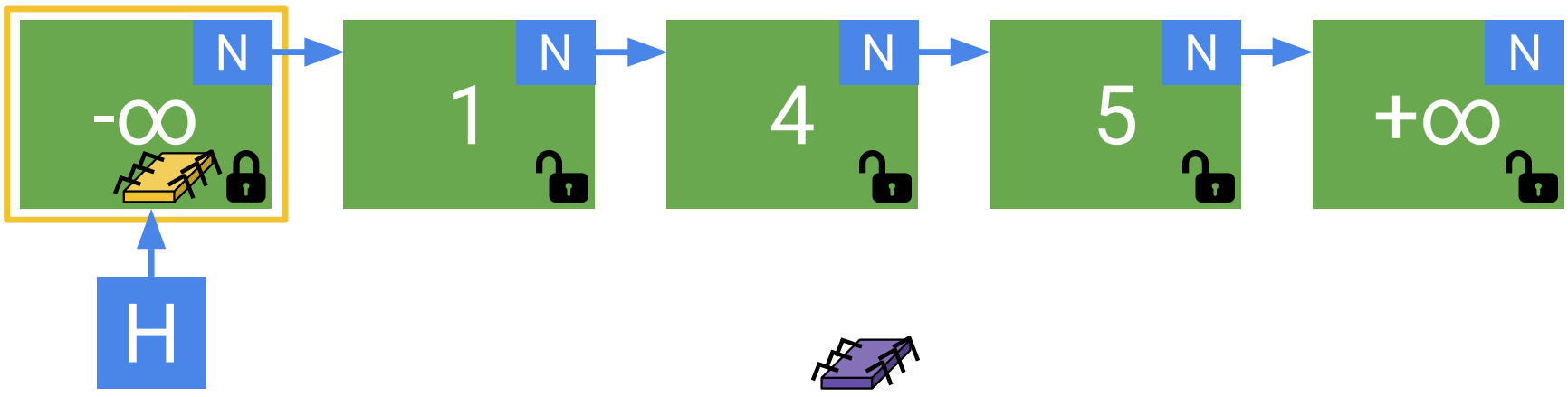
Пример

Жёлтый удаляет «4», фиолетовый удаляет «5»



Пример

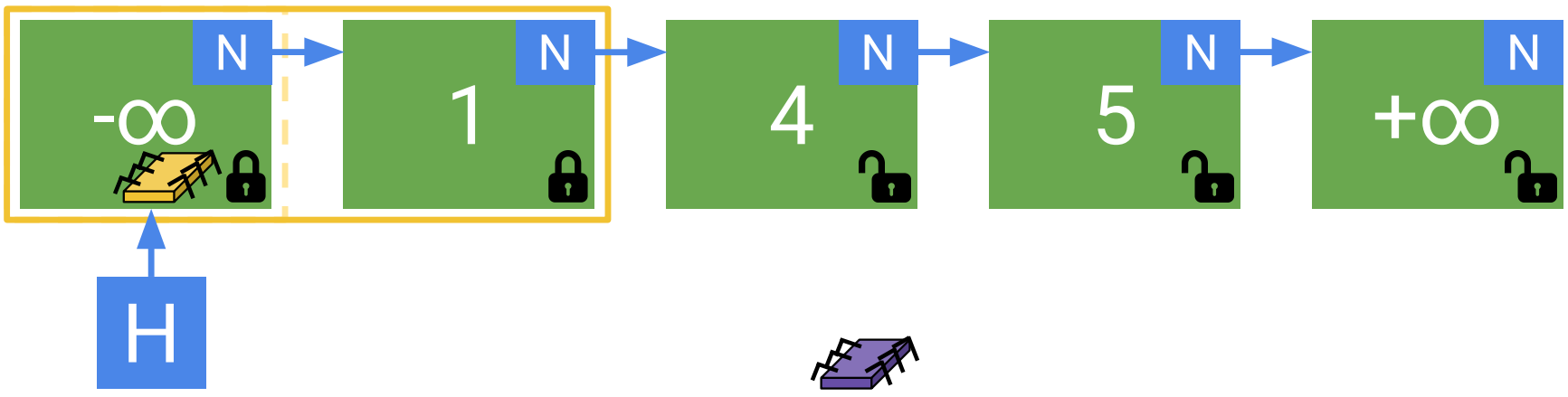
Жёлтый удаляет «4», фиолетовый удаляет «5»



Жёлтый берёт блокировку на голову списка ...

Пример

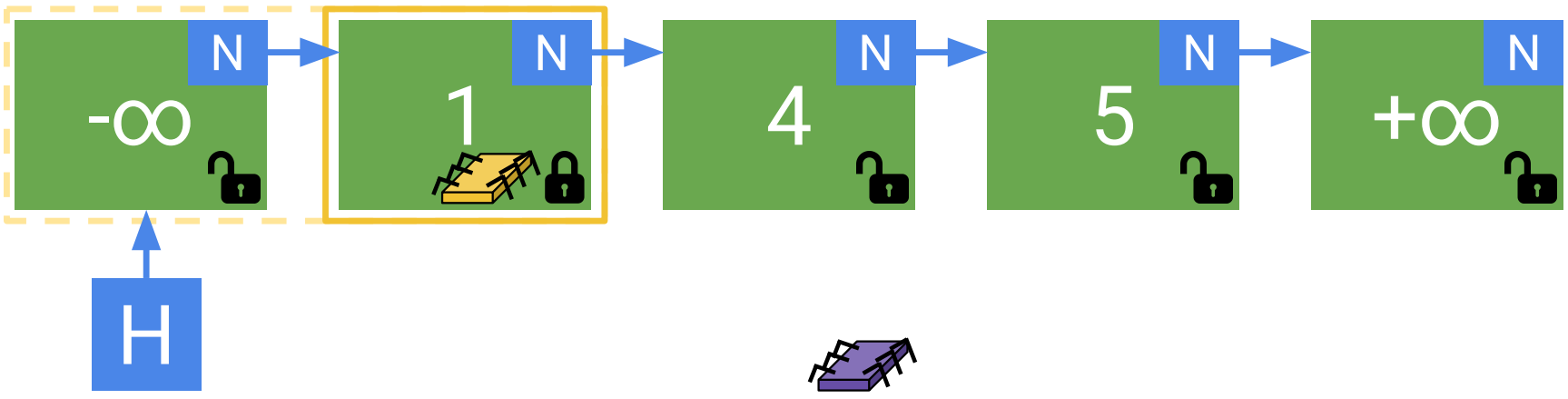
Жёлтый удаляет «4», фиолетовый удаляет «5»



... и на следующий элемент

Пример

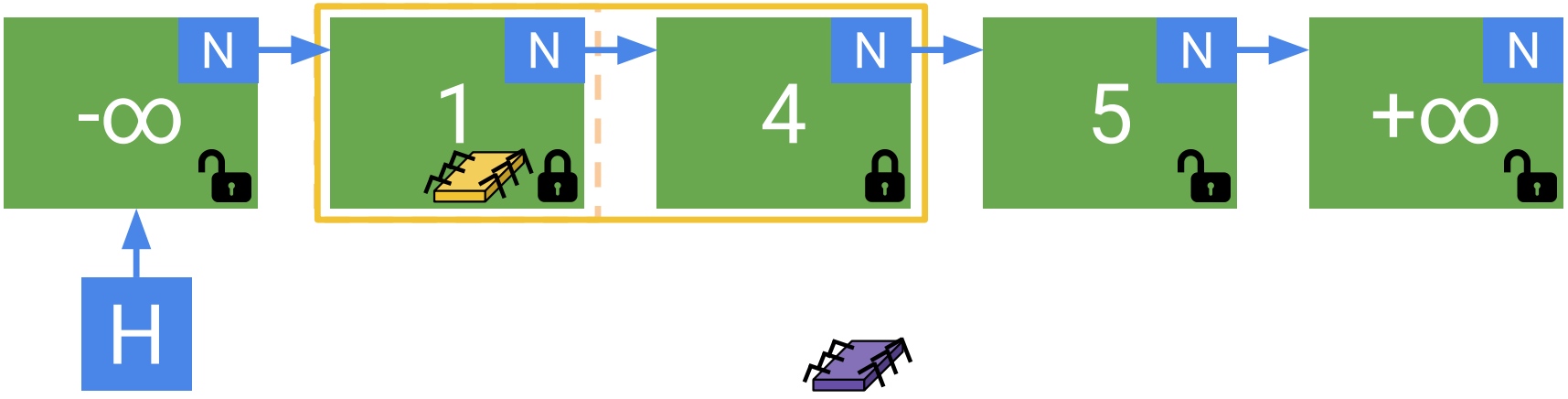
Жёлтый удаляет «4», фиолетовый удаляет «5»



Жёлтый отпускает блокировку на голову списка ...

Пример

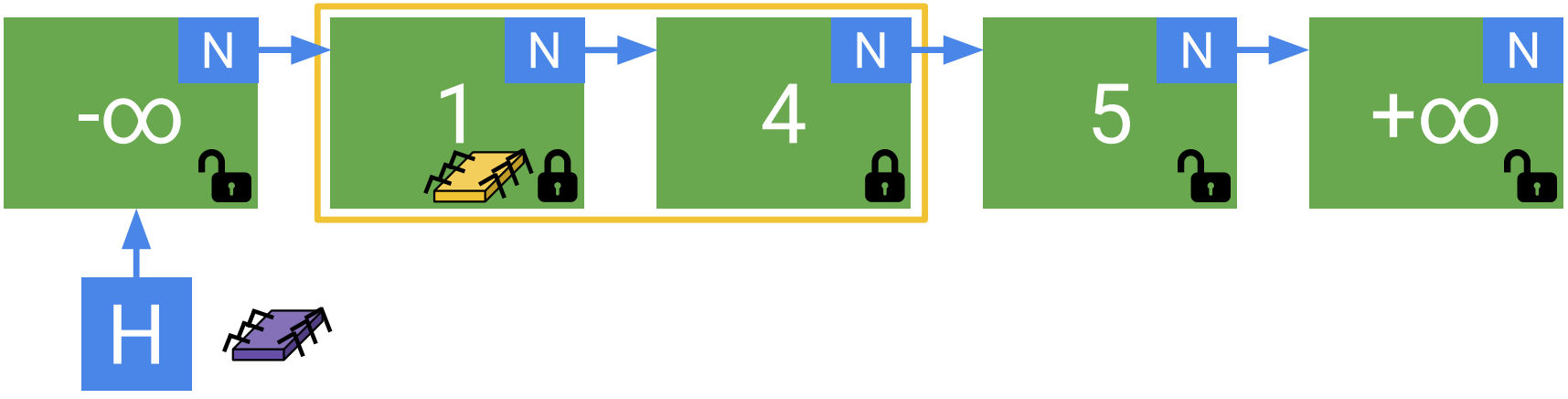
Жёлтый удаляет «4», фиолетовый удаляет «5»



... и берёт блокировку на «4», нашёл окно

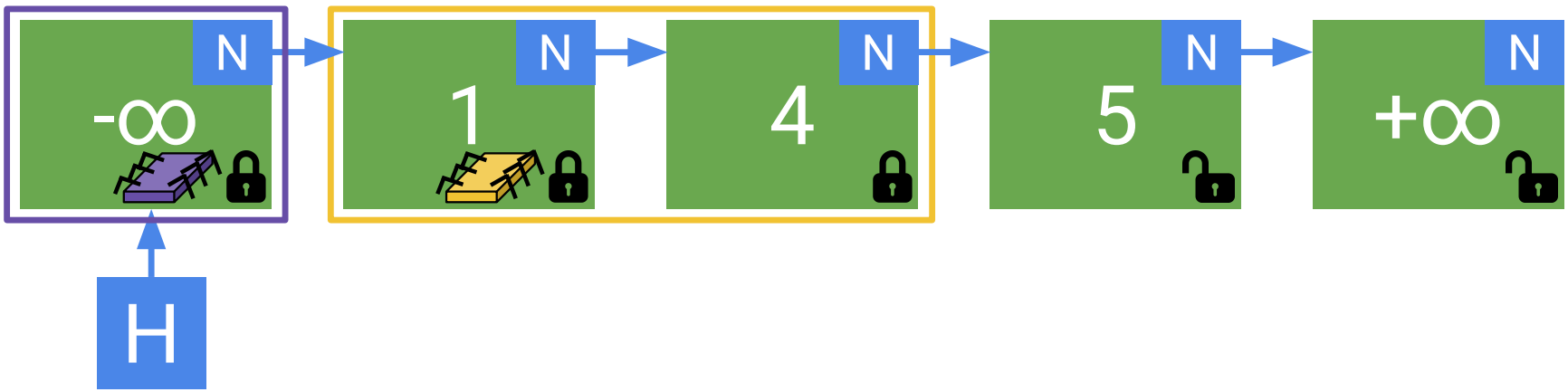
Пример

Жёлтый удаляет «4», фиолетовый удаляет «5»



Пример

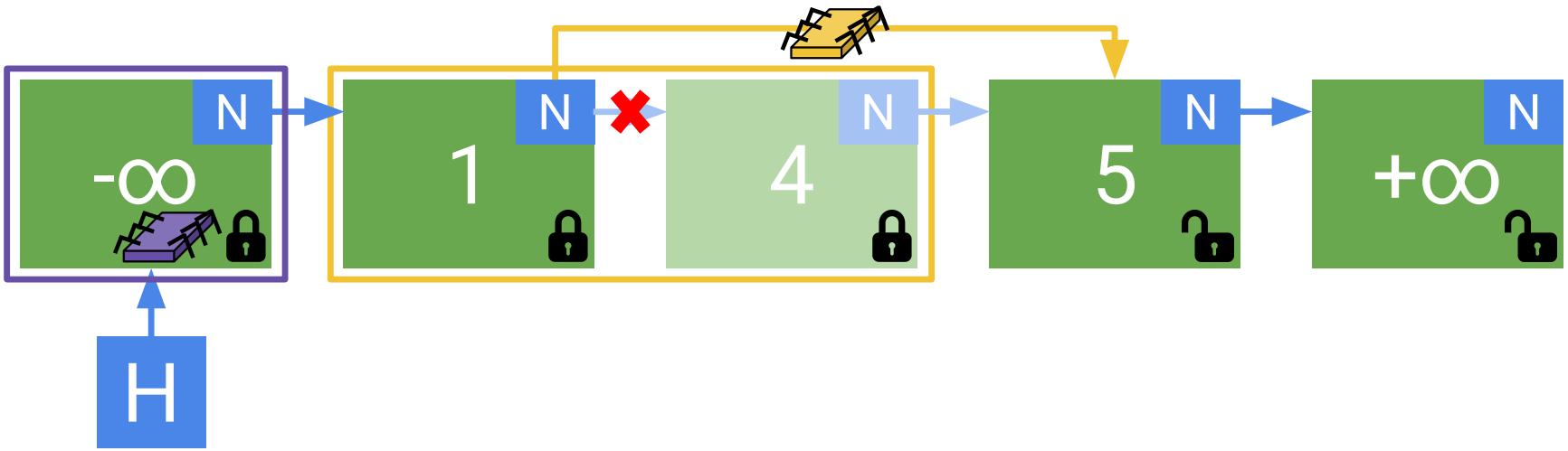
Жёлтый удаляет «4», фиолетовый удаляет «5»



Фиолетовый берёт блокировку на голову списка и ждет жёлтого

Пример

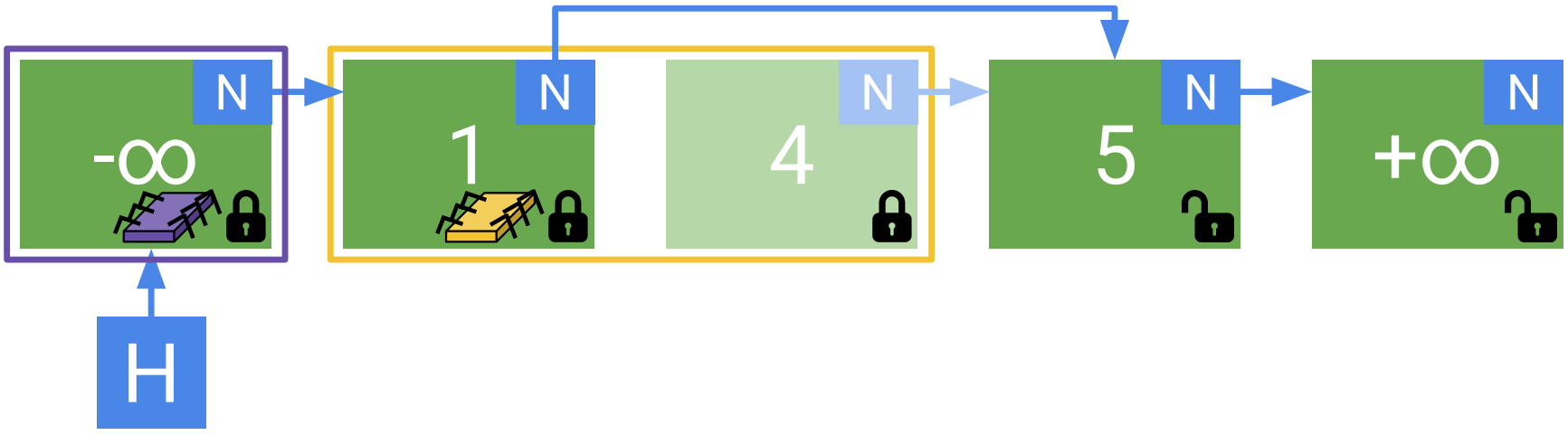
Жёлтый удаляет «4», фиолетовый удаляет «5»



Жёлтый удаляет «4»

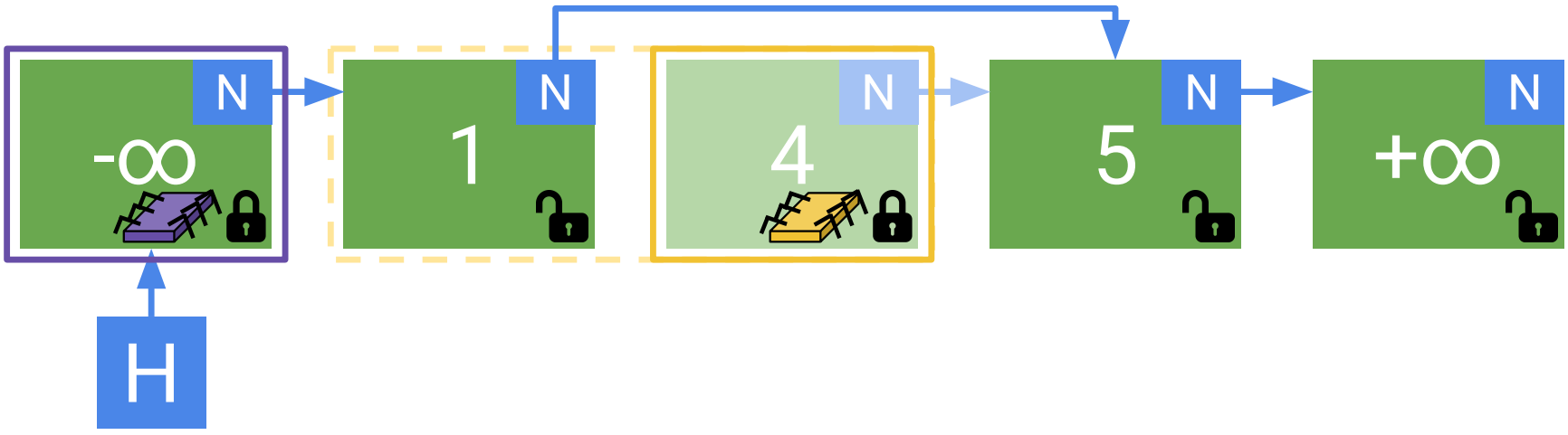
Пример

Жёлтый удаляет «4», фиолетовый удаляет «5»



Пример

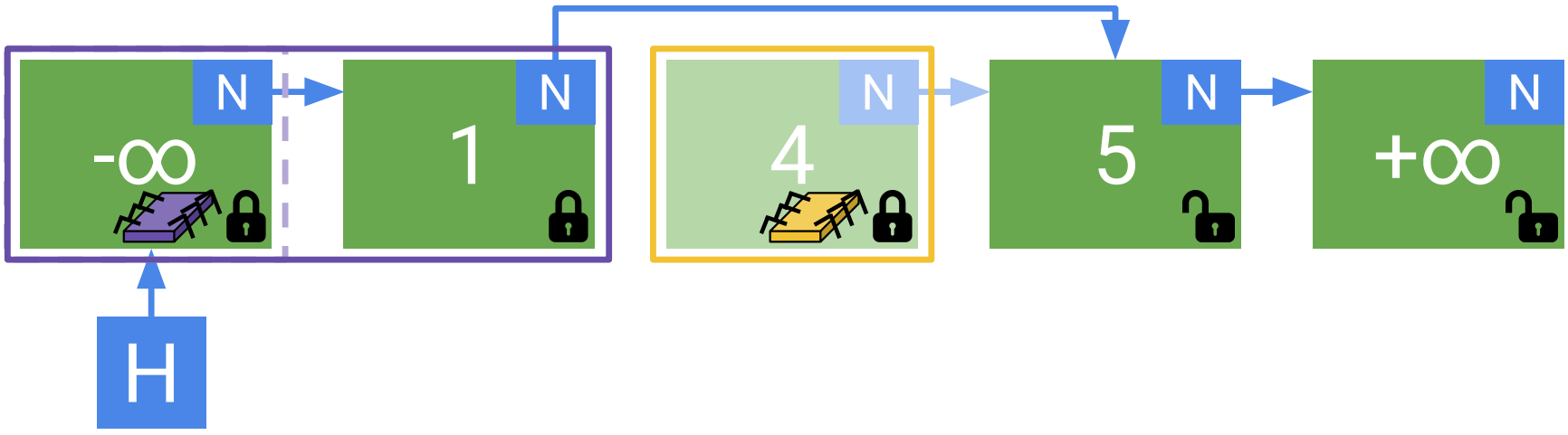
Жёлтый удаляет «4», фиолетовый удаляет «5»



Жёлтый отпускает блокировку на «1»

Пример

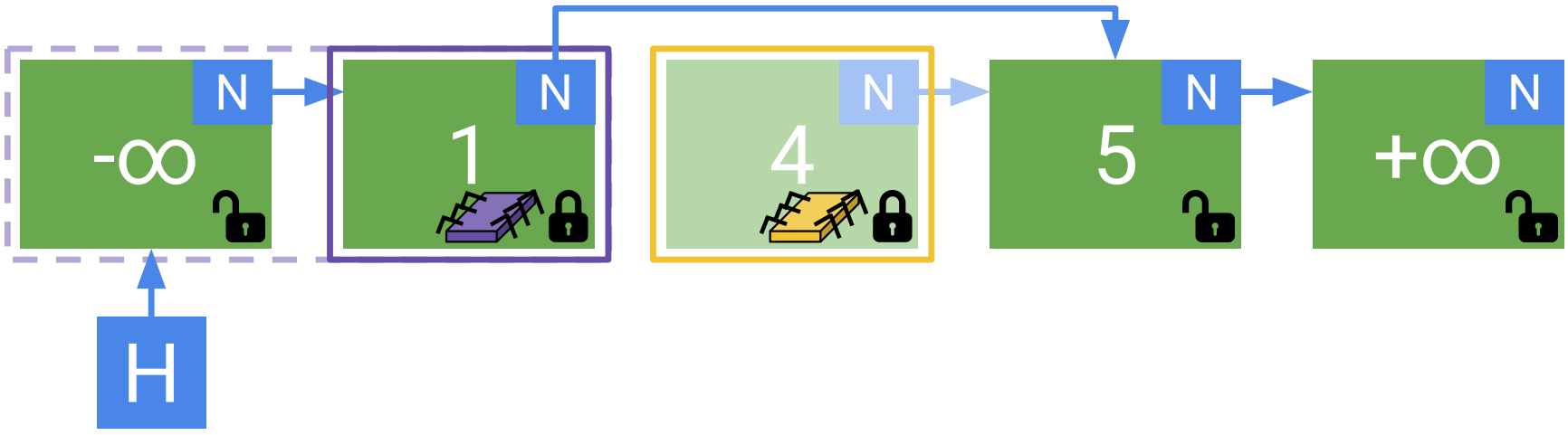
Жёлтый удаляет «4», фиолетовый удаляет «5»



Фиолетовый берёт блокировку на «1» ...

Пример

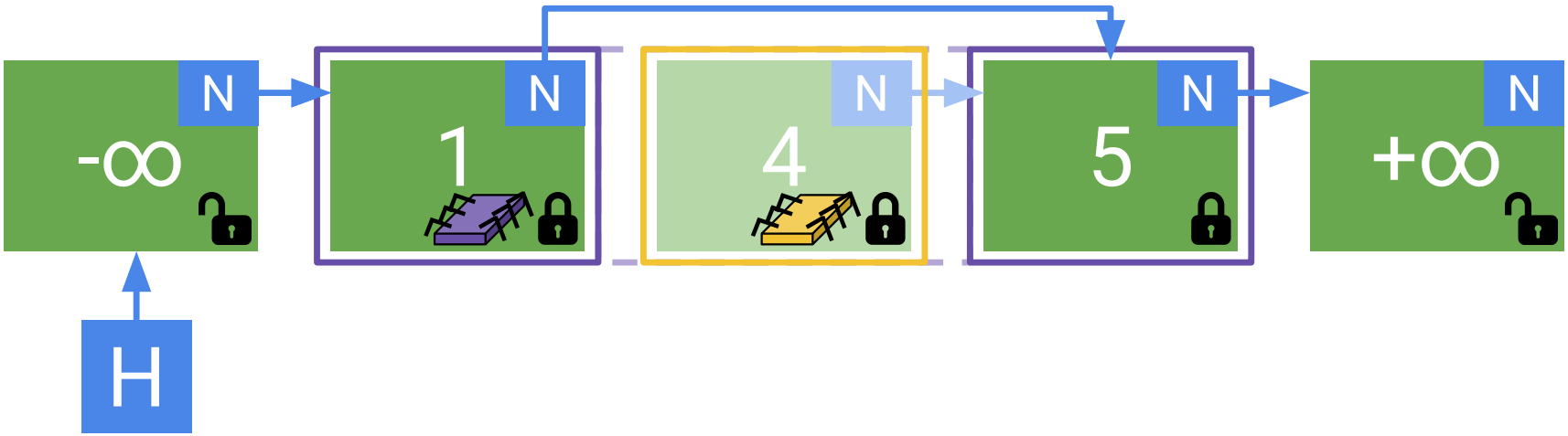
Жёлтый удаляет «4», фиолетовый удаляет «5»



... и отпускает блокировку на голову списка

Пример

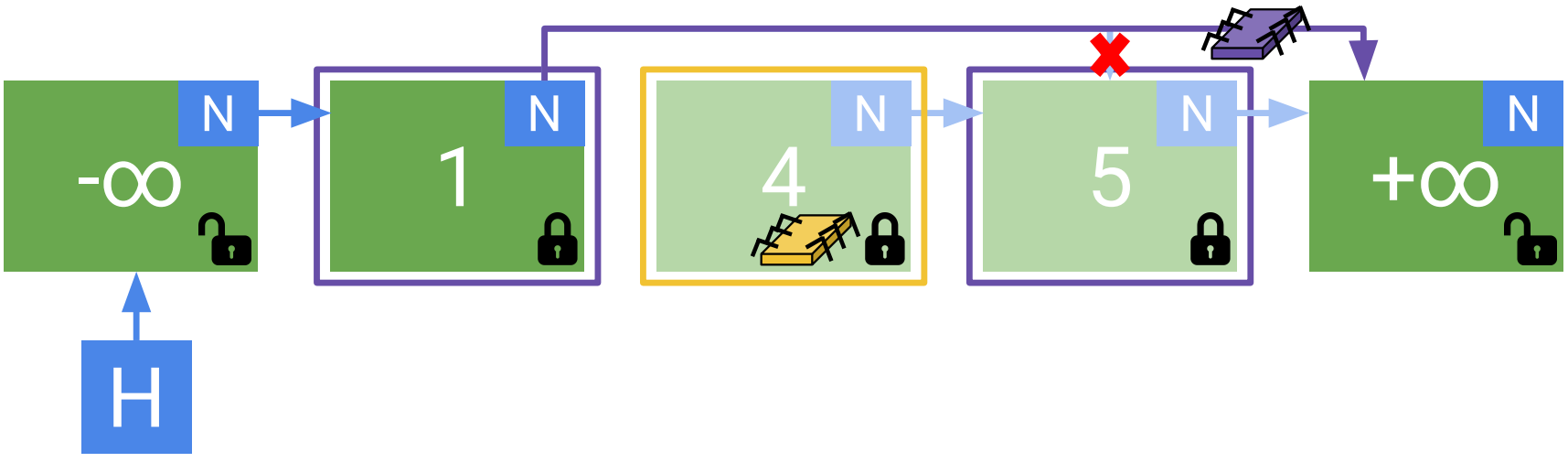
Жёлтый удаляет «4», фиолетовый удаляет «5»



Фиолетовый берёт блокировку на «5», нашёл окно.

Пример

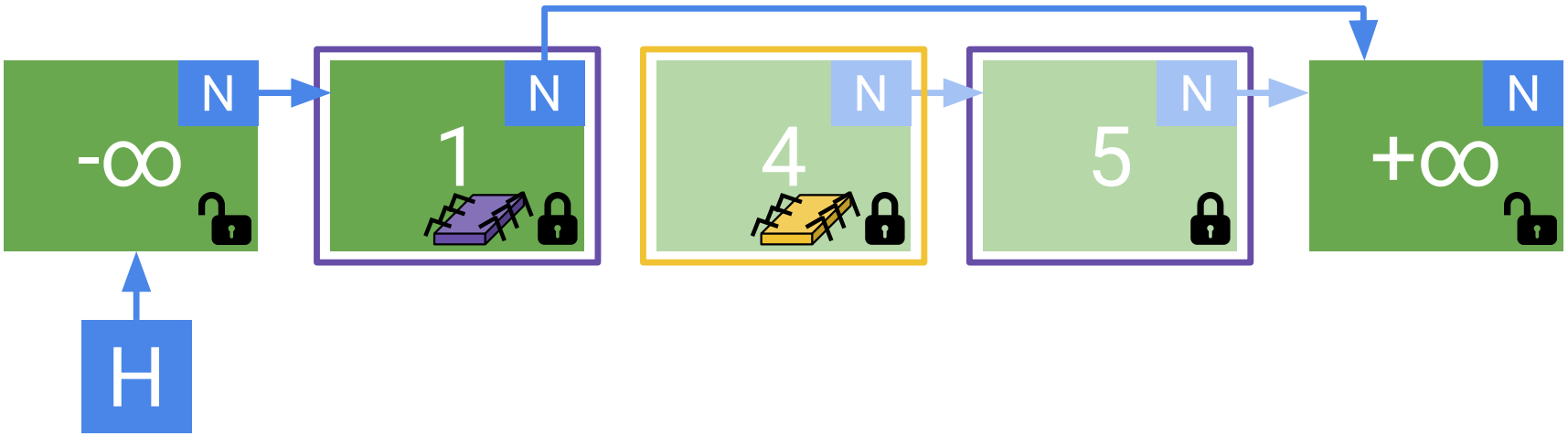
Жёлтый удаляет «4», фиолетовый удаляет «5»



Фиолетовый удаляет «5» ...

Пример

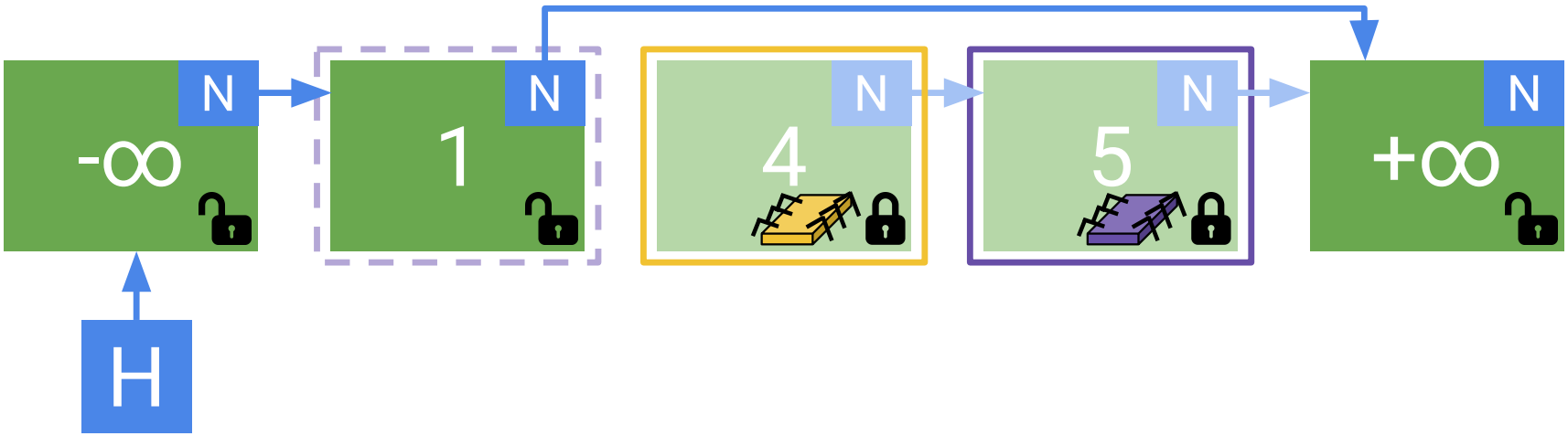
Жёлтый удаляет «4», фиолетовый удаляет «5»



Фиолетовый удаляет «5» ...

Пример

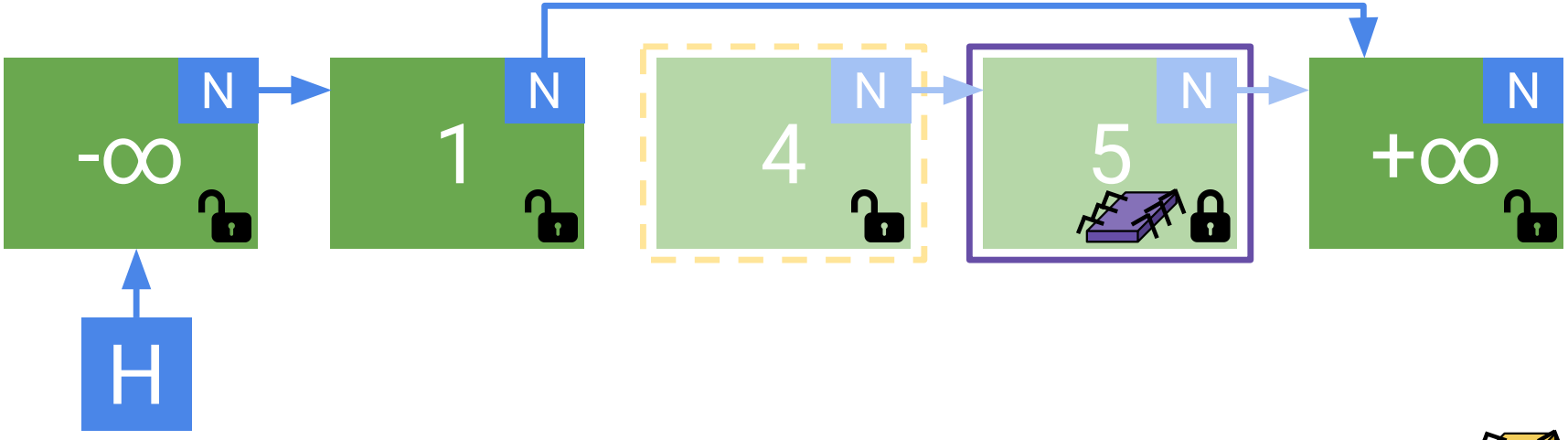
Жёлтый удаляет «4», фиолетовый удаляет «5»



... и отпускает блокировку на «1»

Пример

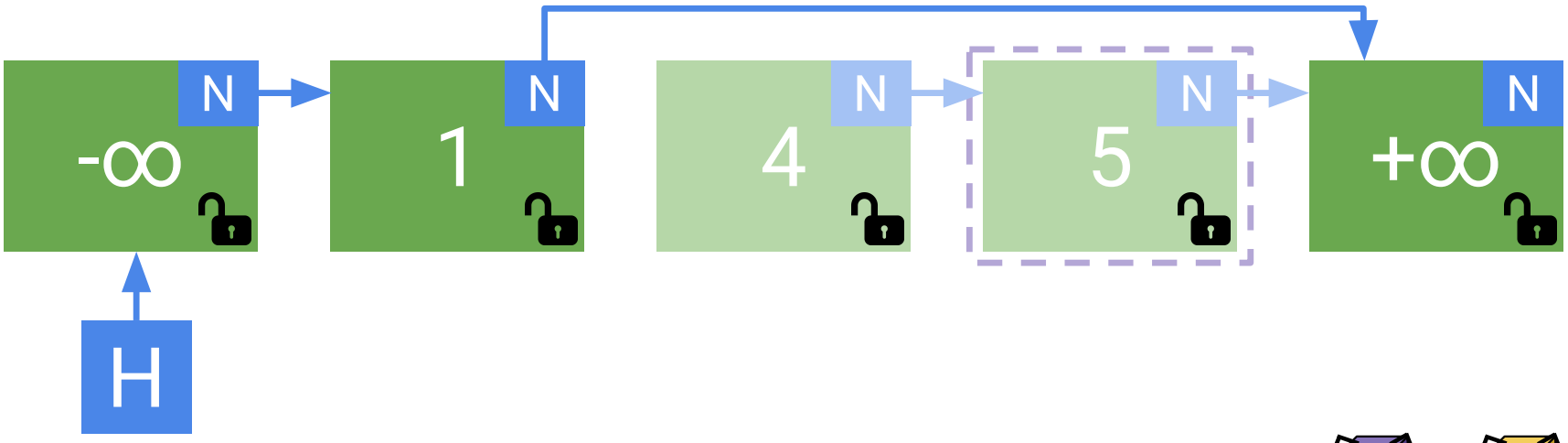
Жёлтый удаляет «4», фиолетовый удаляет «5»



Жёлтый отпускает блокировку на «4»

Пример

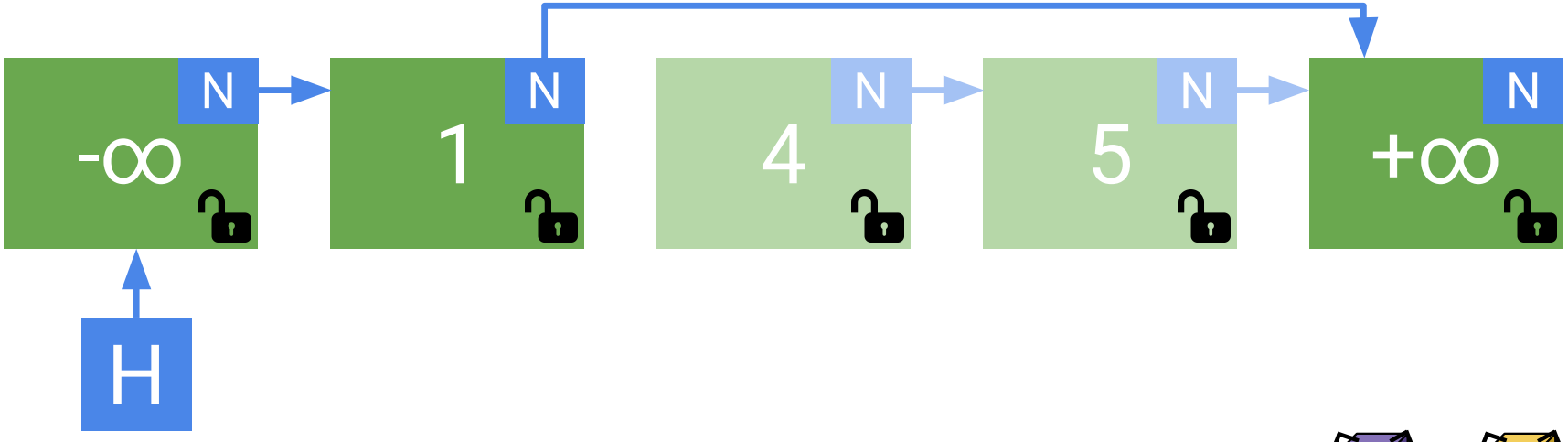
Жёлтый удаляет «4», фиолетовый удаляет «5»



Фиолетовый отпускает блокировку на «5»

Пример

Жёлтый удаляет «4», фиолетовый удаляет «5»



Оба элемента удалены корректно

Тонкая синхронизация: псевдокод

```
fun findWindow(key: Int): (Node, Node) {
    cur := head; cur.lock()
    next := cur.N; next.lock()
    while (next.key < key) {
        cur.unlock(); cur = next
        next = cur.N; next.lock()
    }
    return (cur, next)
}

fun contains(key: Int): Boolean {
    (cur, next) := findWindow(key)
    res := next.key == key
    cur.unlock(); next.unlock()
    return res
}
```

Корректность

- Поиск окна: запись и чтение $\texttt{cur}.N$ не могут происходить параллельно
- Модификация: во время изменения окно защищено блокировками $\Rightarrow$ атомарно
- $\forall k$: операции с ключом k линеаризуемы $\Rightarrow$ всё исполнение линеаризуемо
- Операции с ключом k упорядочены взятием соответствующей блокировки

План

1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

Алгоритм абстрактной операции

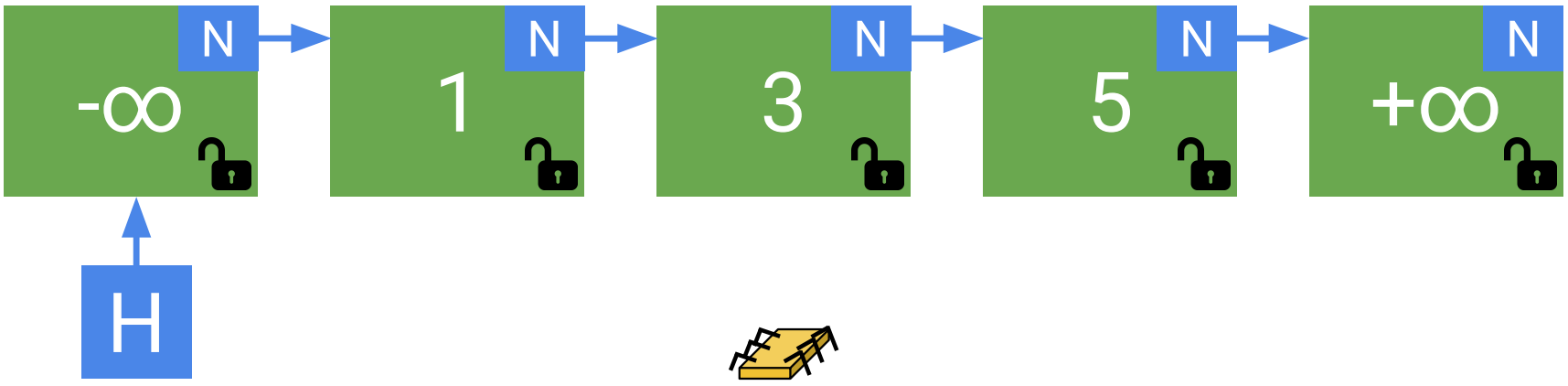
1. Найти окно $(\texttt{cur}, \texttt{next})$ без синхронизации
2. Взять блокировки на $\texttt{cur}$ и $\texttt{next}$
3. Проверить инвариант $\texttt{cur}.N = \texttt{next}$
4. Проверить, что $\texttt{cur}$ не удалён
5. Выполнить операцию (добавить, удалить, ...)
6. При любой ошибке начать заново

Проверка, что узел не удалён

- Как проверить, что `cur` не удален?
- Держим блокировку на `cur` и `cur` удален $\Rightarrow$ не увидим `cur` при проходе
- Попробуем найти `cur` ещё раз за $O(n)$ и проверим, что $\texttt{cur}.N = \texttt{next}$

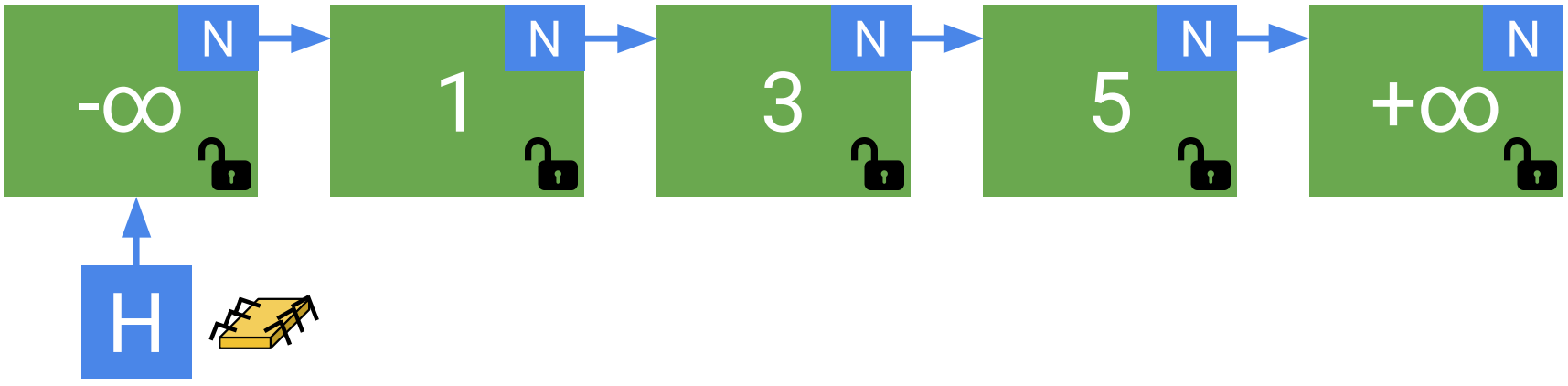
Пример успешной операции

Жёлтый добавляет «4»



Пример успешной операции

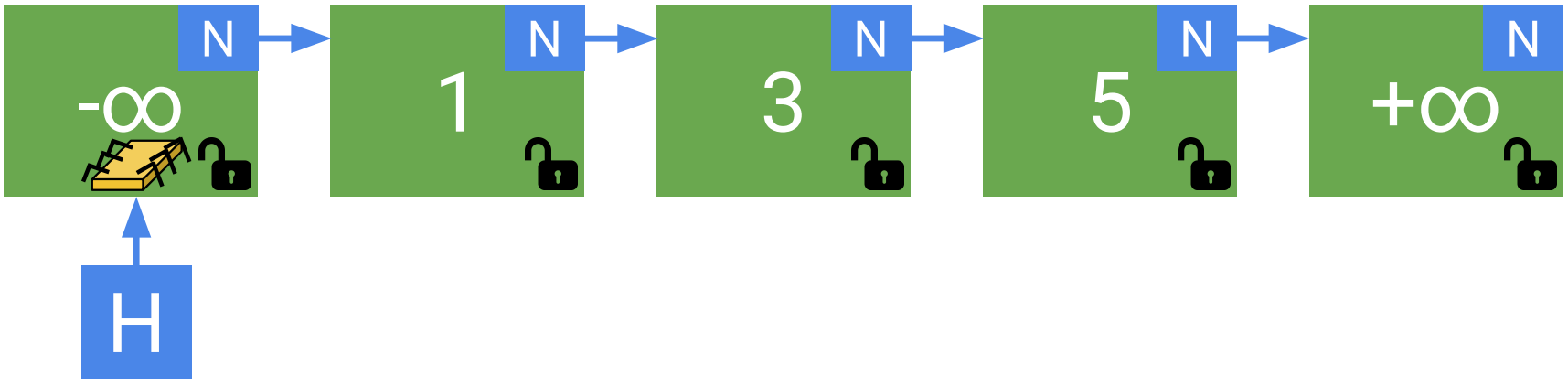
Жёлтый добавляет «4»



Жёлтый ищет окно

Пример успешной операции

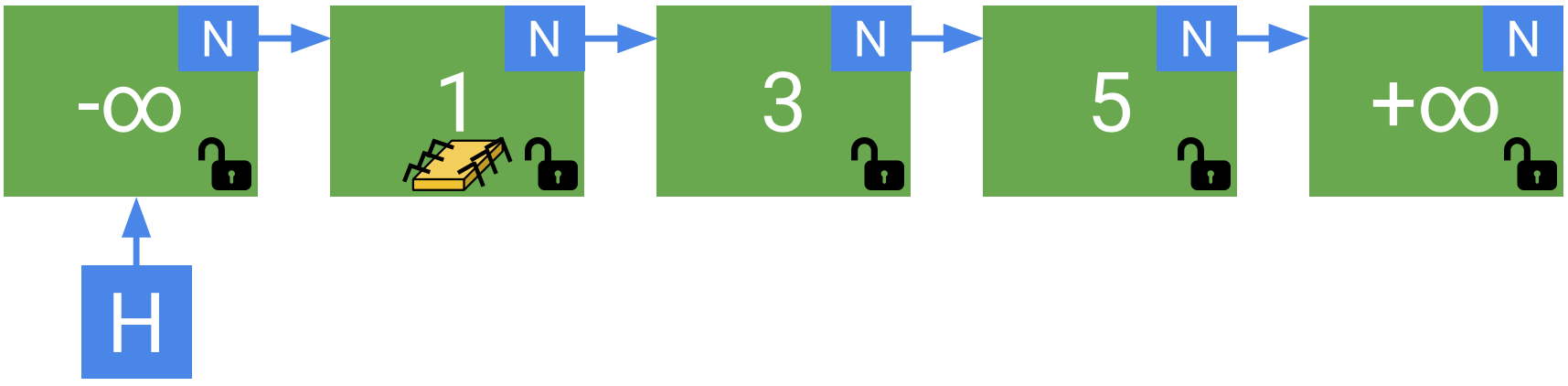
Жёлтый добавляет «4»



Жёлтый ищет окно

Пример успешной операции

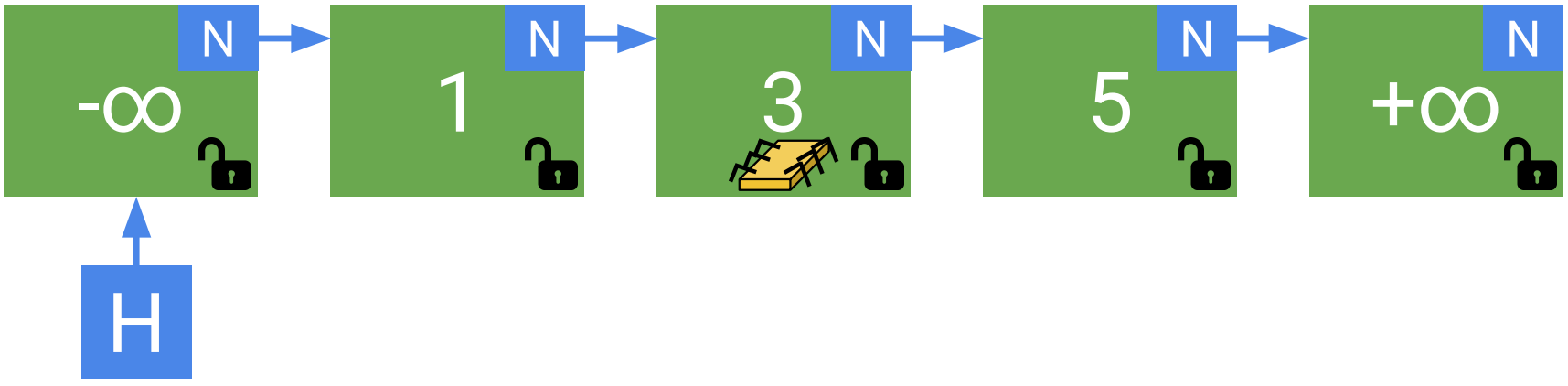
Жёлтый добавляет «4»



Жёлтый ищет окно

Пример успешной операции

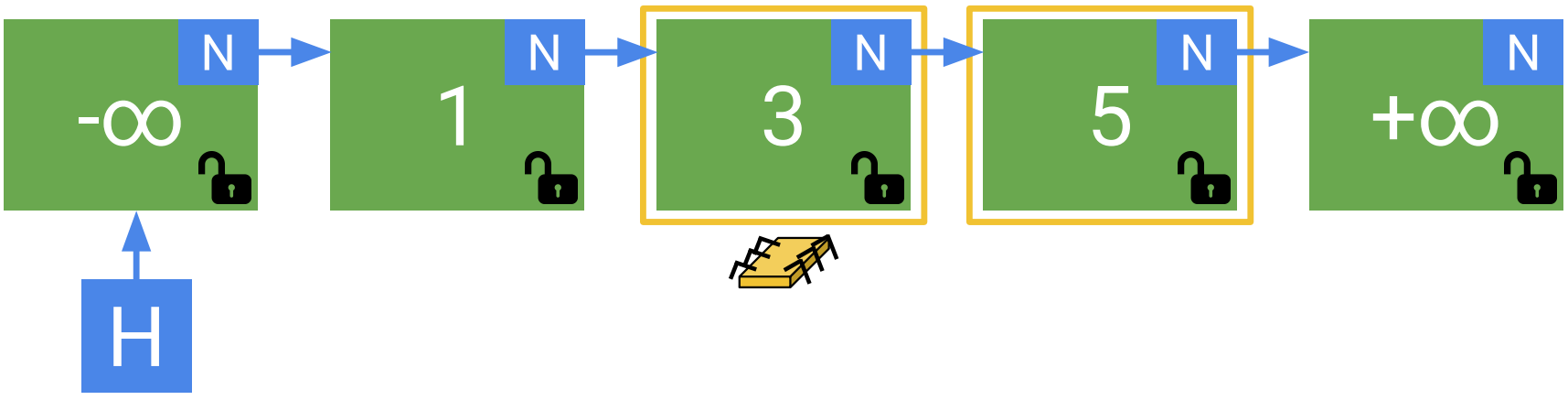
Жёлтый добавляет «4»



Жёлтый ищет окно

Пример успешной операции

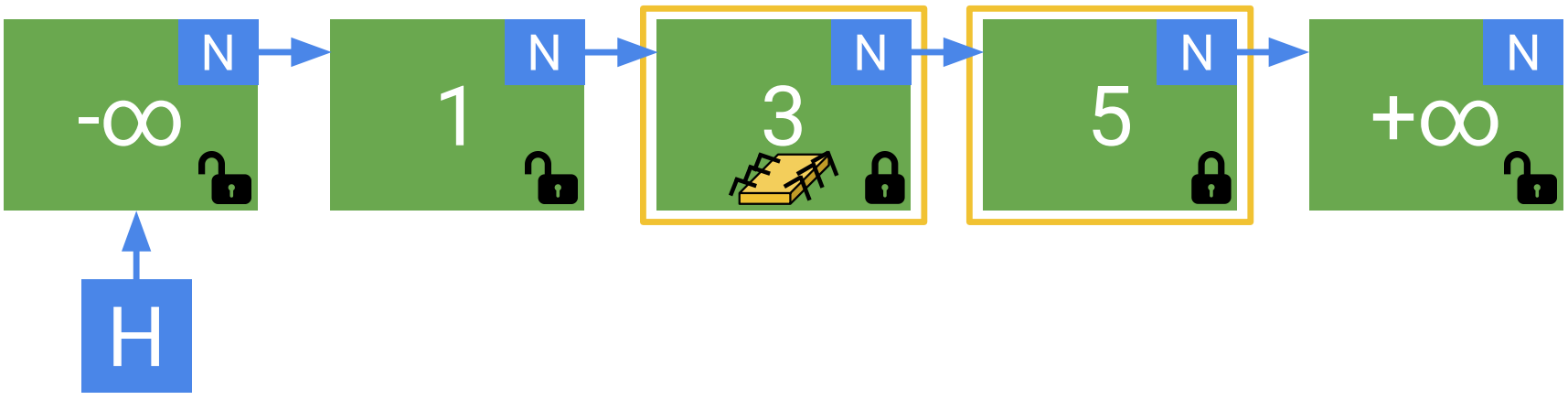
Жёлтый добавляет «4»



Нашёл окно

Пример успешной операции

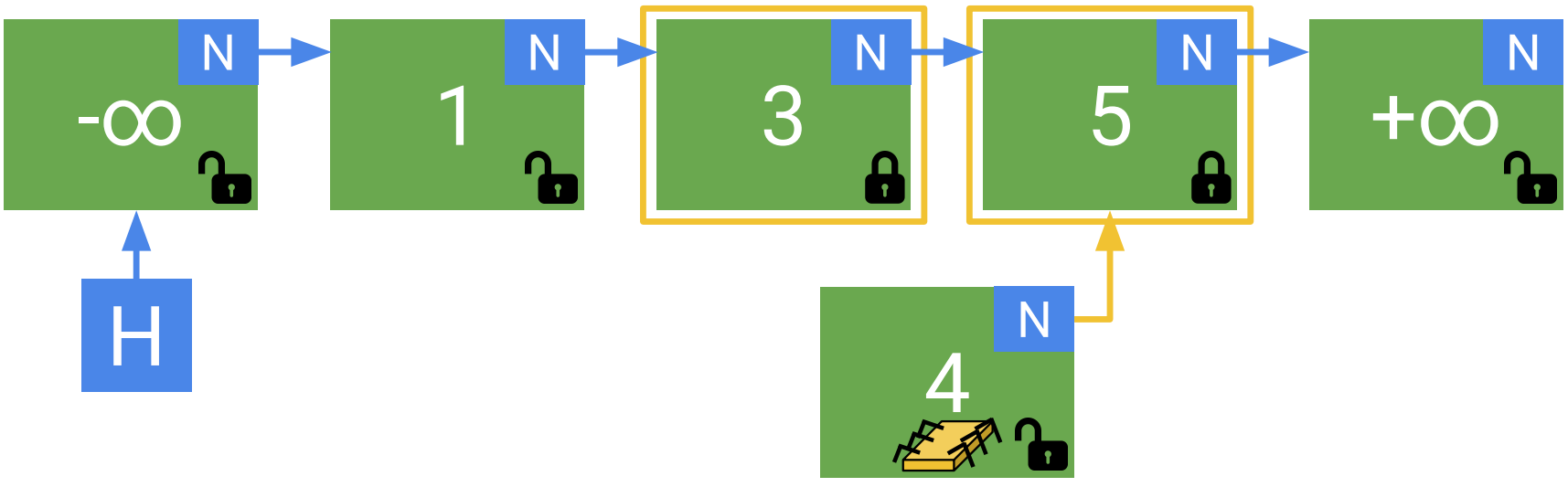
Жёлтый добавляет «4»



Берёт блокировки

Пример успешной операции

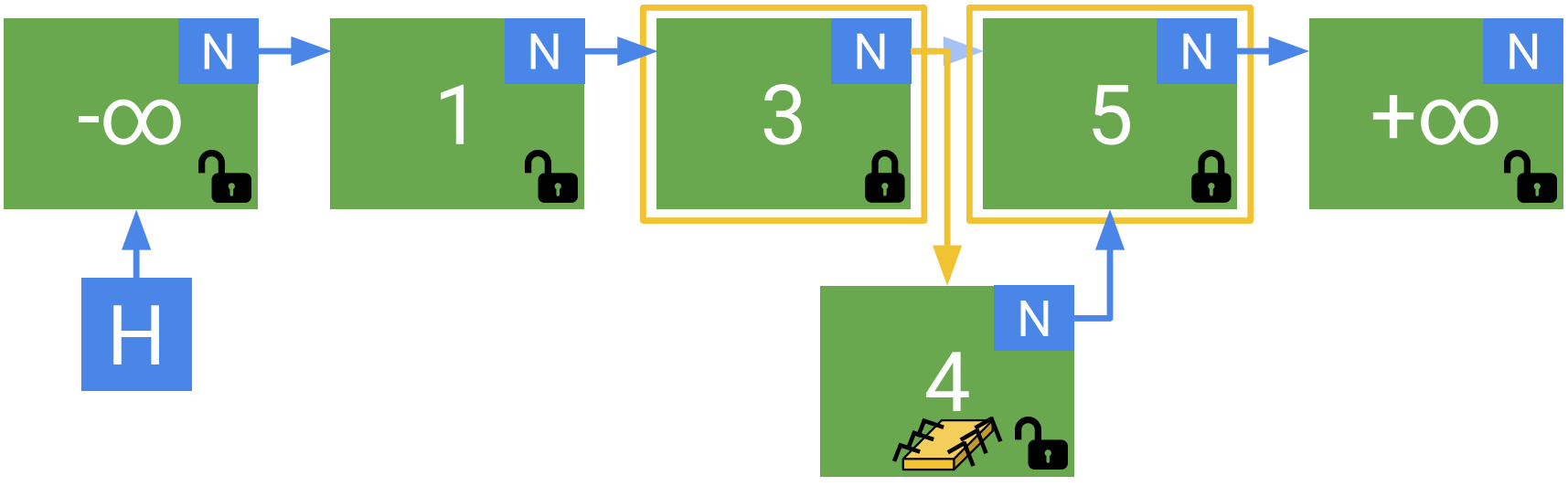
Жёлтый добавляет «4»



Добавляет узел «4»

Пример успешной операции

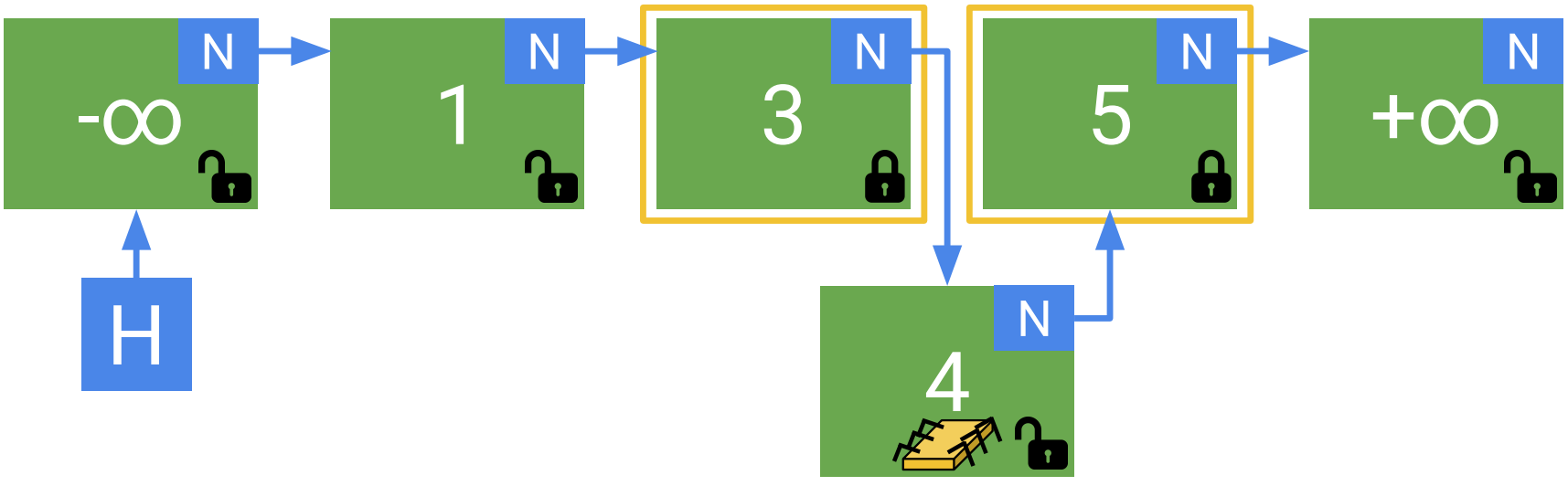
Жёлтый добавляет «4»



Добавляет узел «4»

Пример успешной операции

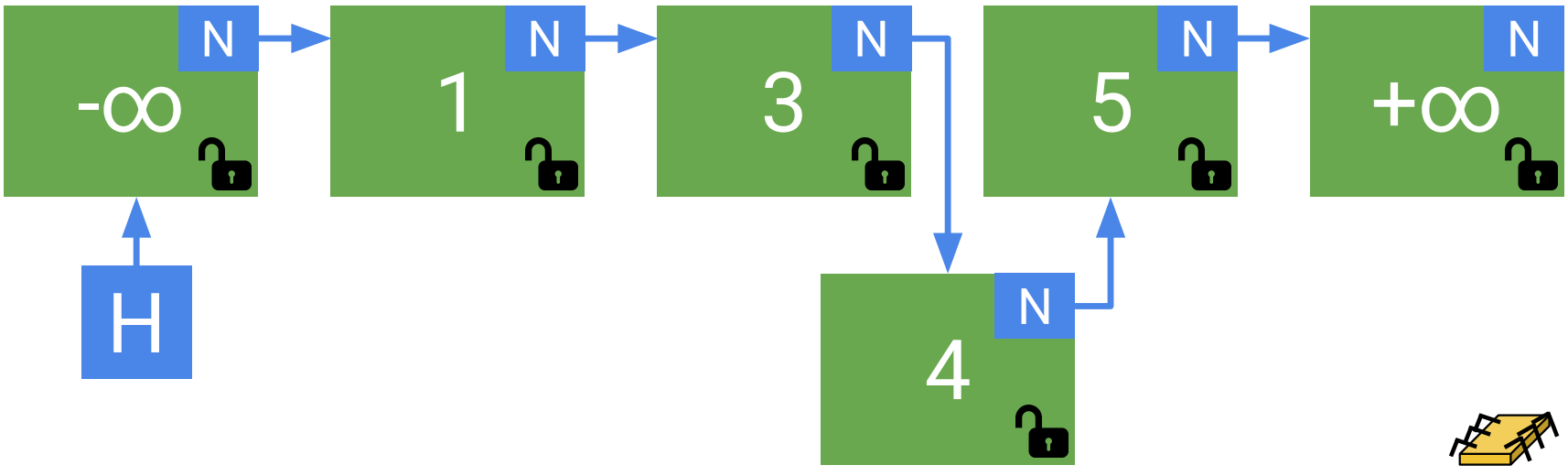
Жёлтый добавляет «4»



Добавляет узел «4»

Пример успешной операции

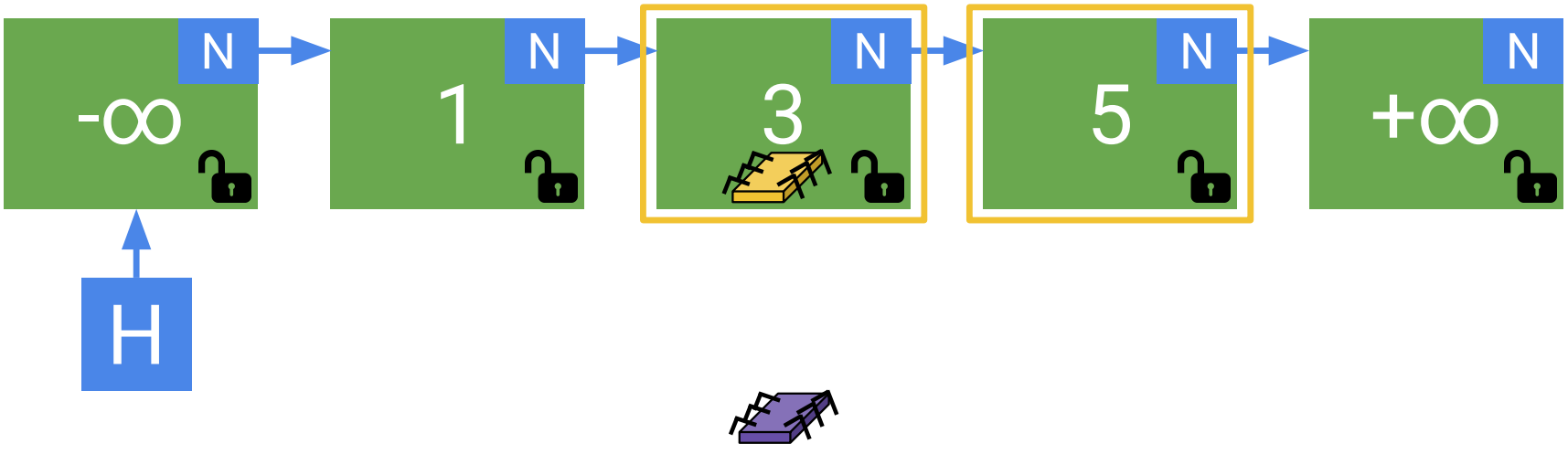
Жёлтый добавляет «4»



Отпускает блокировки и уходит

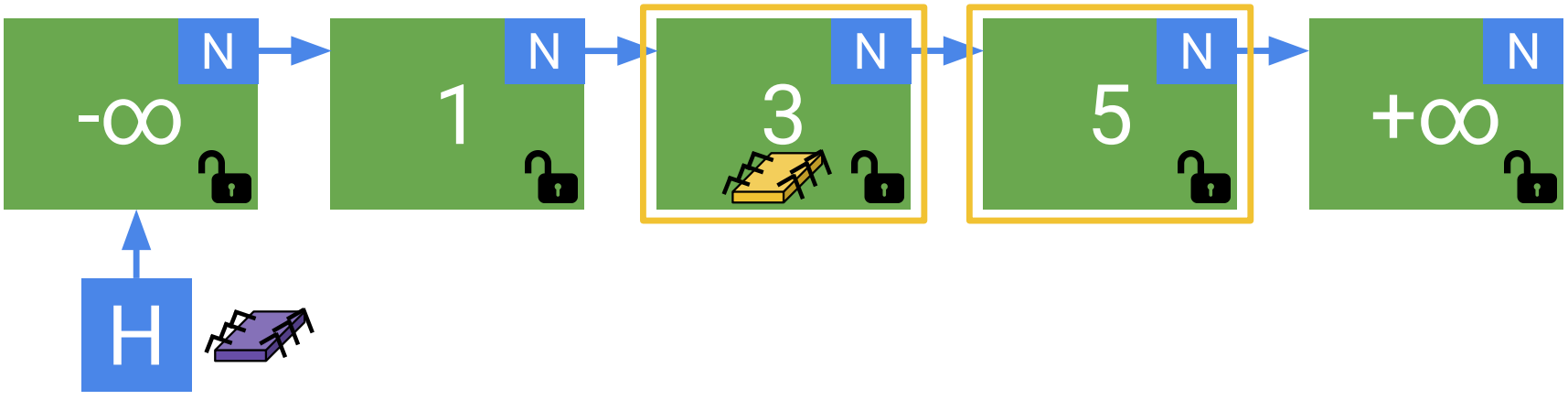
Проблема: cur.N != next

Оба добавляют «4»



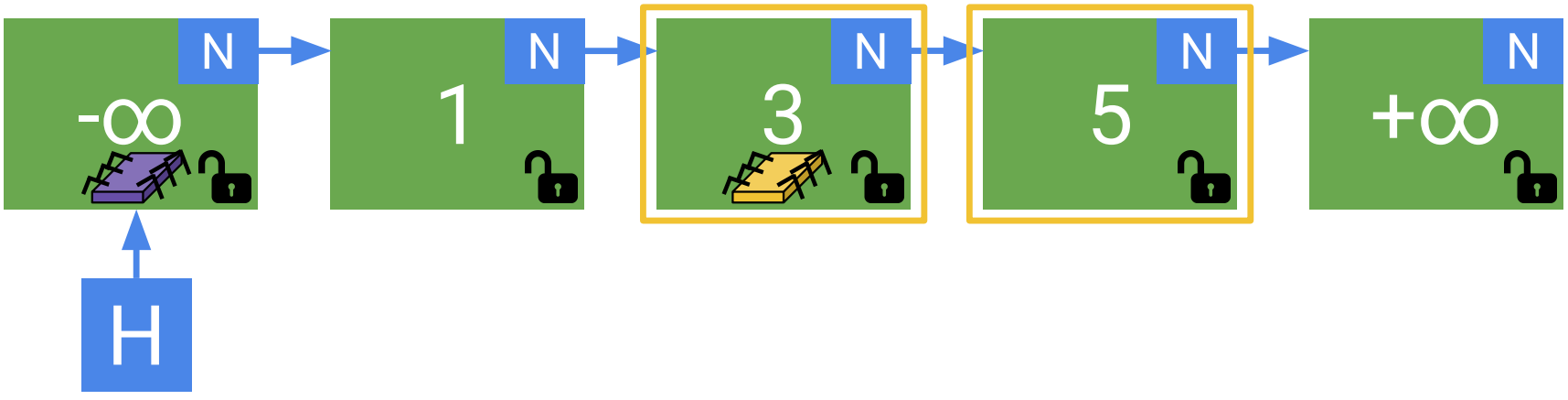
Проблема: cur.N != next

Оба добавляют «4»



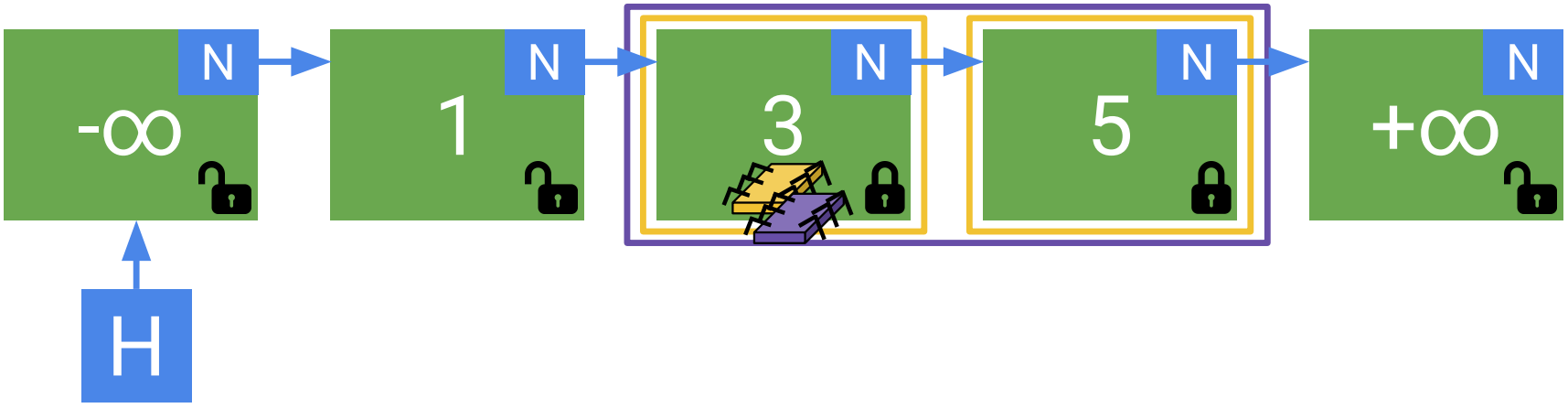
Проблема: cur.N != next

Оба добавляют «4»



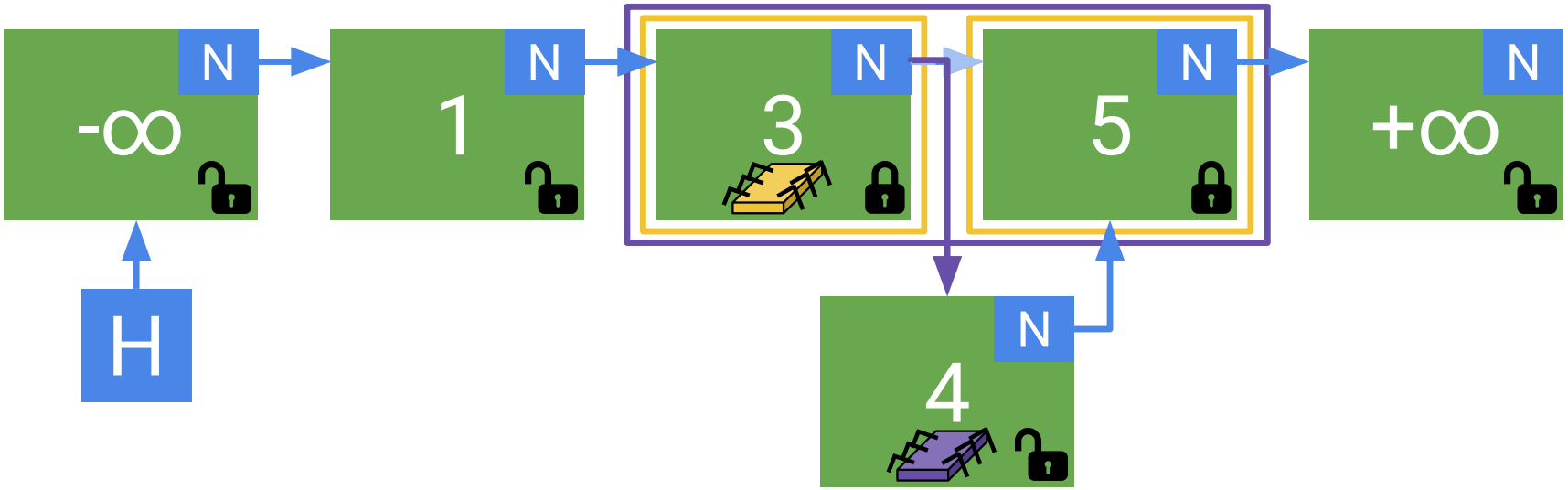
Проблема: cur.N != next

Оба добавляют «4»



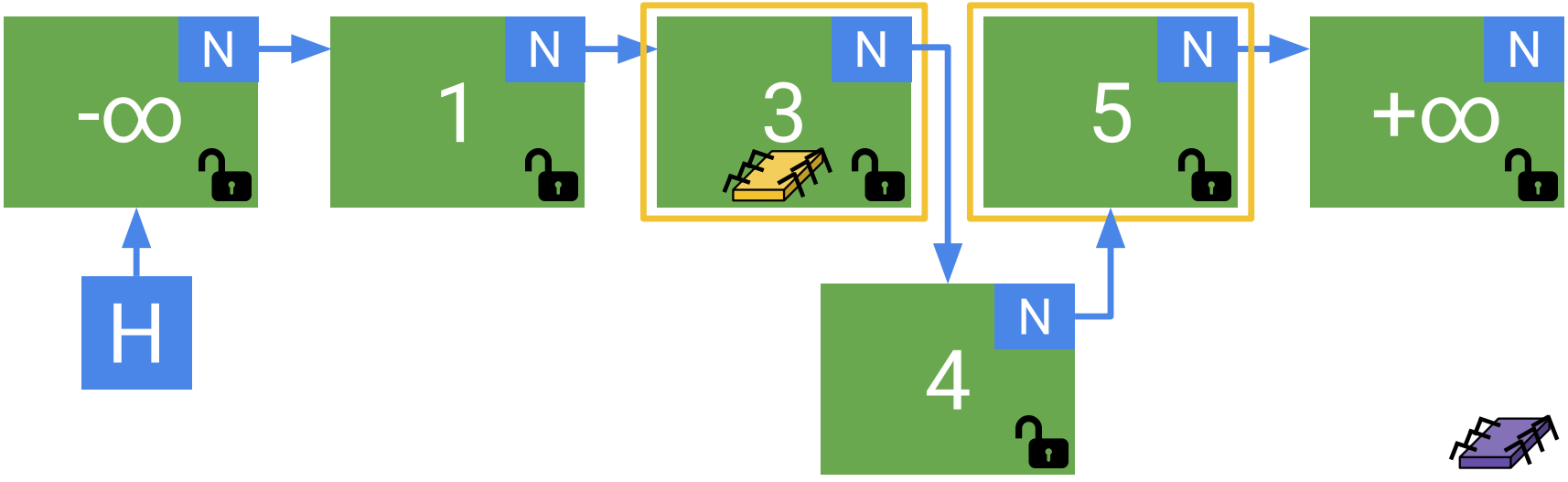
Проблема: cur.N != next

Оба добавляют «4»



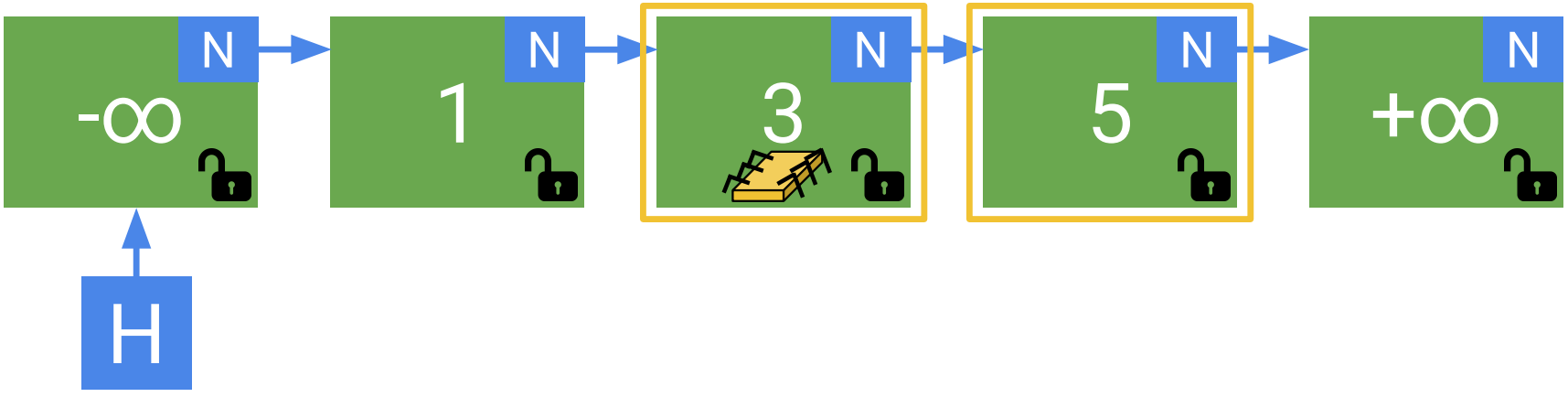
Проблема: cur.N != next

Оба добавляют «4»



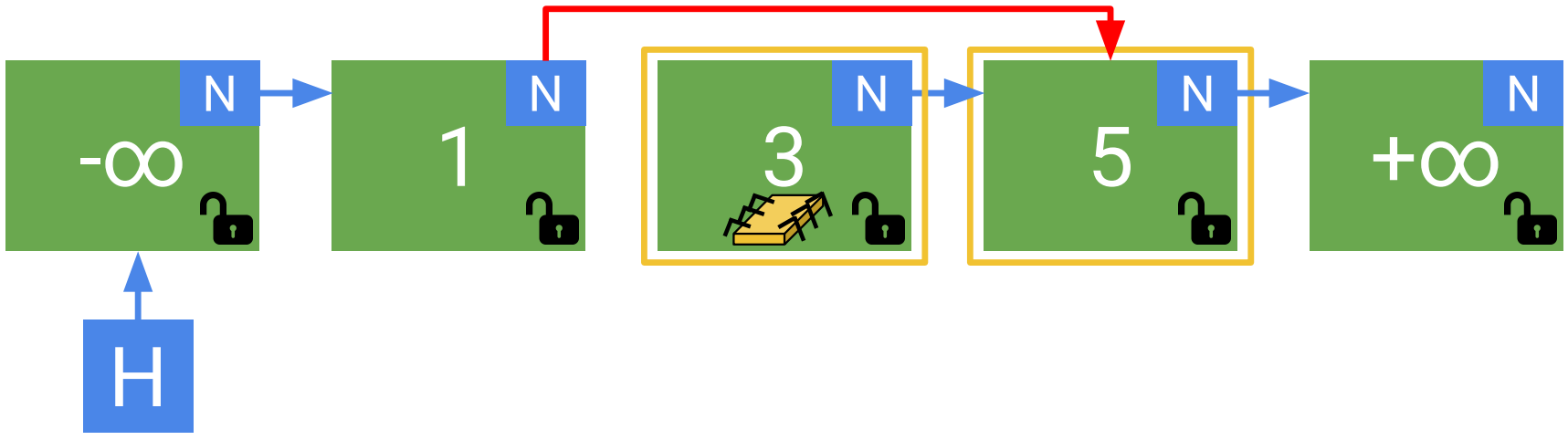
Проблема: cur уже удалили

Пока жёлтый «тормозил», `cur` уже удалили



Проблема: cur уже удалили

Пока жёлтый «тормозил», `cur` уже удалили



Оптимистичный поиск: псевдокод

```
fun findWindow(key: Int): (Node, Node) {
    // Without locks
}

fun contains(key): Boolean = while (true) {
    (cur, next) := findWindow(key)
    cur.lock(); next.lock()
    if (!validate(cur, next))
        (cur, next).unlock(); continue
    res := next.key == key
    (cur, next).unlock()
    return res
}
```

Оптимистичный поиск: валидация окна

```
fun validate(cur: Node, next: Node): Boolean {
  node := head
  while (node.key < cur.key)
    node = node.N
  return (cur, next) == (node, node.N)
}
```

Корректность

- Поиск: запись и чтение $\texttt{cur}.N$ связанны отношением «произошло до»
- Можем говорить о линеаризуемости операций над одинаковыми ключами
- Точка линеаризации - взятие блокировки на cur

План

1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

Ленивое удаление: идея

- Добавим в Node поле $\mathrm{removed}$ типа $\mathrm{Boolean}$
- Удаление в две фазы:
  1. $\mathrm{node.removed} = \mathrm{true}$ - логическое удаление
  2. Физическое удаление из списка

Ленивое удаление: идея

- Добавим в Node поле $\mathrm{removed}$ типа $\mathrm{Boolean}$
- Удаление в две фазы:
  1. $\mathrm{node.removed} = \mathrm{true}$ - логическое удаление
  2. Физическое удаление из списка
- Инвариант: все неудаленные вершины в списке
- $\Rightarrow$ теперь не надо проходить по списку в $\mathrm{validate}()$

Ленивое удаление: псевдокод

```
fun validate(cur: Node, next: Node): Boolean {
    return !cur.removed &&
           !next.removed &&
           cur.N == next
}
```

План

1. Множество на односвязном списке
2. Грубая синхронизация
3. Тонкая синхронизация
4. Оптимистичная синхронизация
5. Ленивая синхронизация
6. Неблокирующая синхронизация

Неблокирующий поиск

- На момент чтения поля N видим состояние на момент записи N или новее
- $\Rightarrow$ можем не брать блокировку при поиске

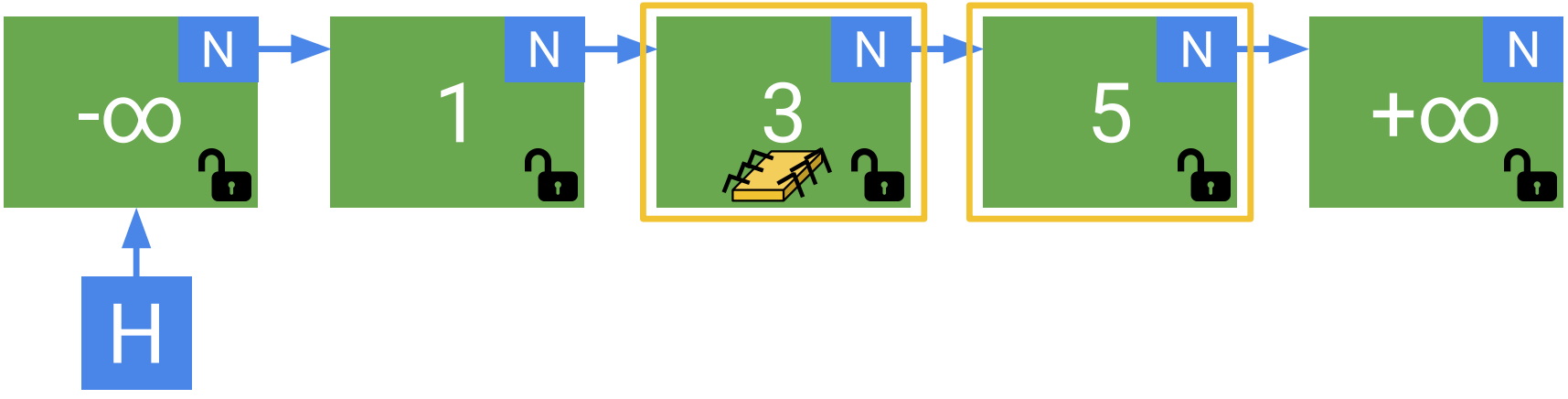
Неблокирующий поиск

- На момент чтения поля N видим состояние на момент записи N или новее
- $\Rightarrow$ можем не брать блокировку при поиске

```
fun contains(key: Int): Boolean {
    (cur, next) = findWindow(key) // No locks here
    return next.key == key
}
```

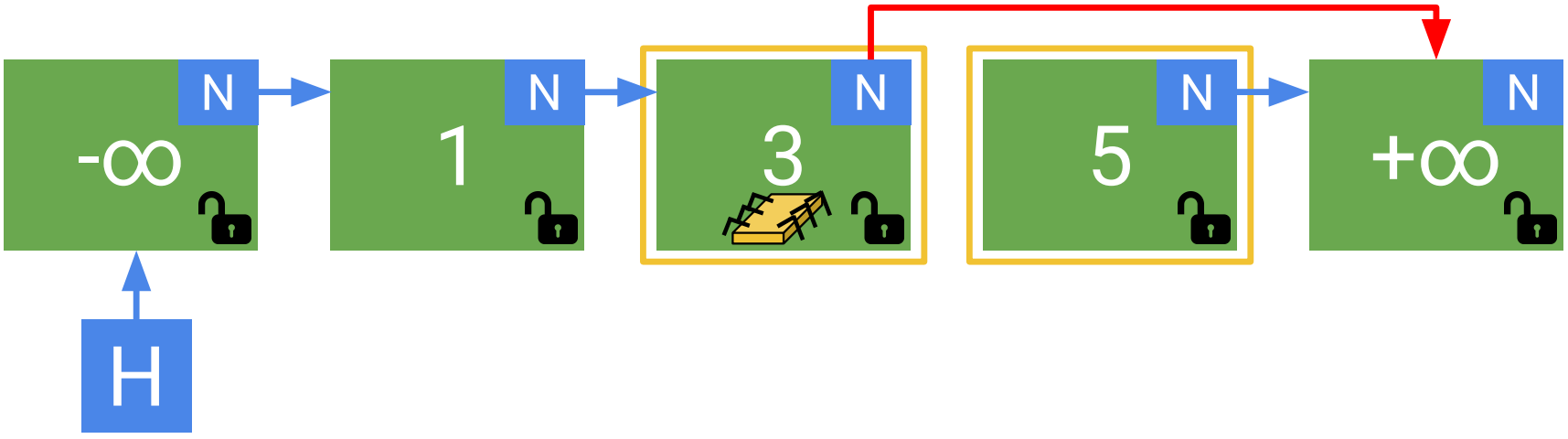
Пример

Жёлтый ищет «5», но его удаляют параллельно



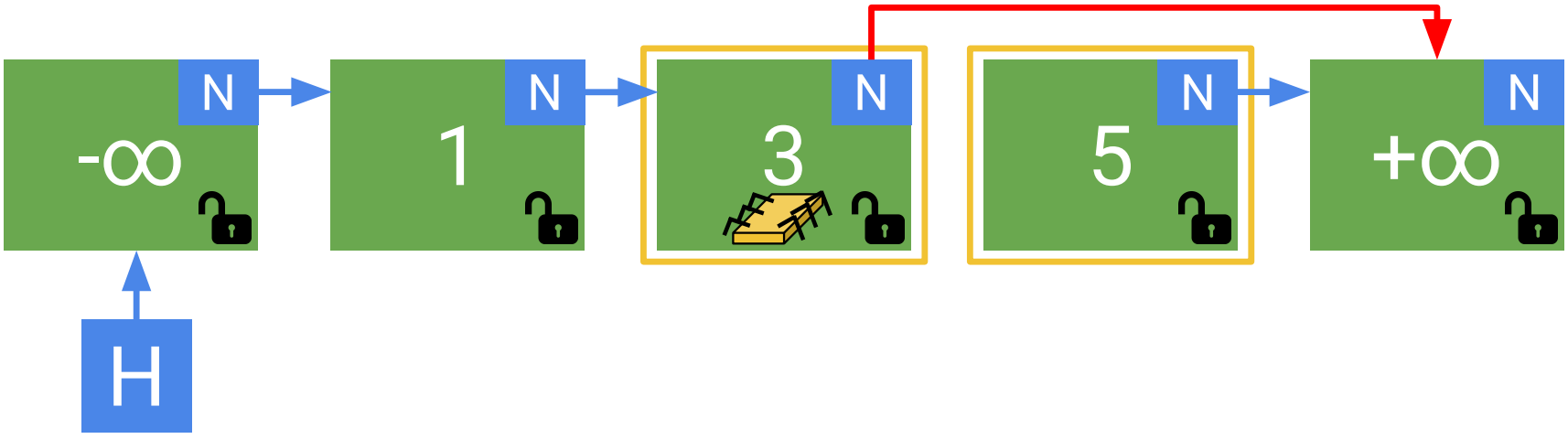
Пример

Жёлтый ищет «5», но его удаляют параллельно



Пример

Жёлтый ищет «5», но его удаляют параллельно



Выполняются параллельно

$\Rightarrow$ можем упорядочить как угодно

Compare-and-set

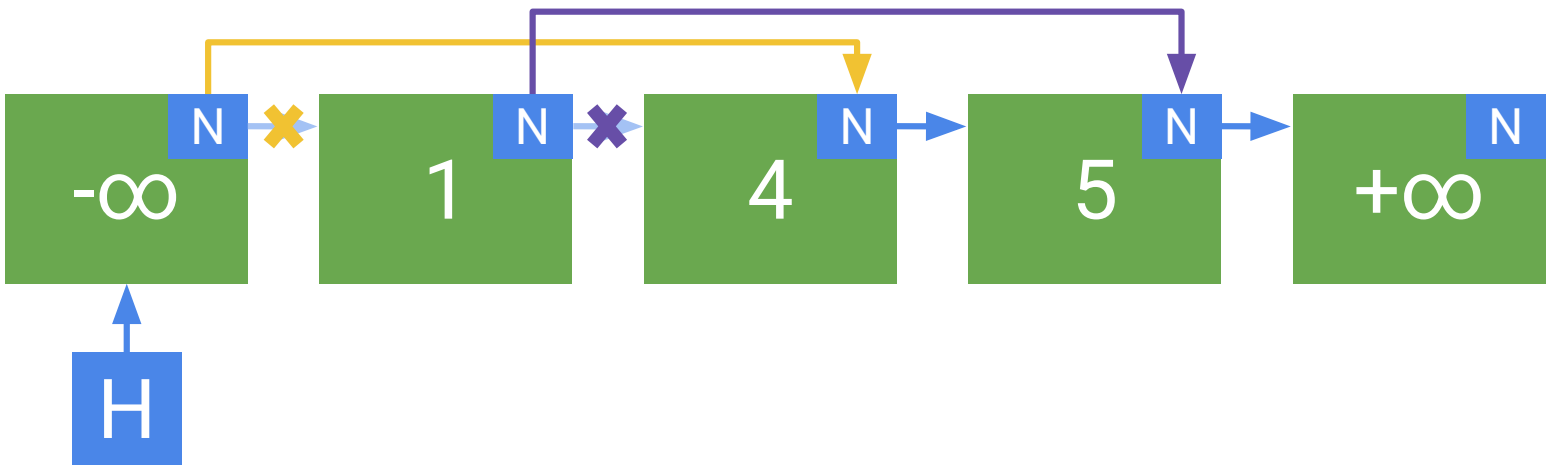
```
class AtomicReg<T> {
  var x: T

  CAS(expected: T, value: T): Boolean {
    do atomically {
      val old = x
      if (old == expected):
        x = value
        return true
      return false
    }
  }
}
```

Неблокирующая модификация

- Вернёмся к множеству на односвязном списке

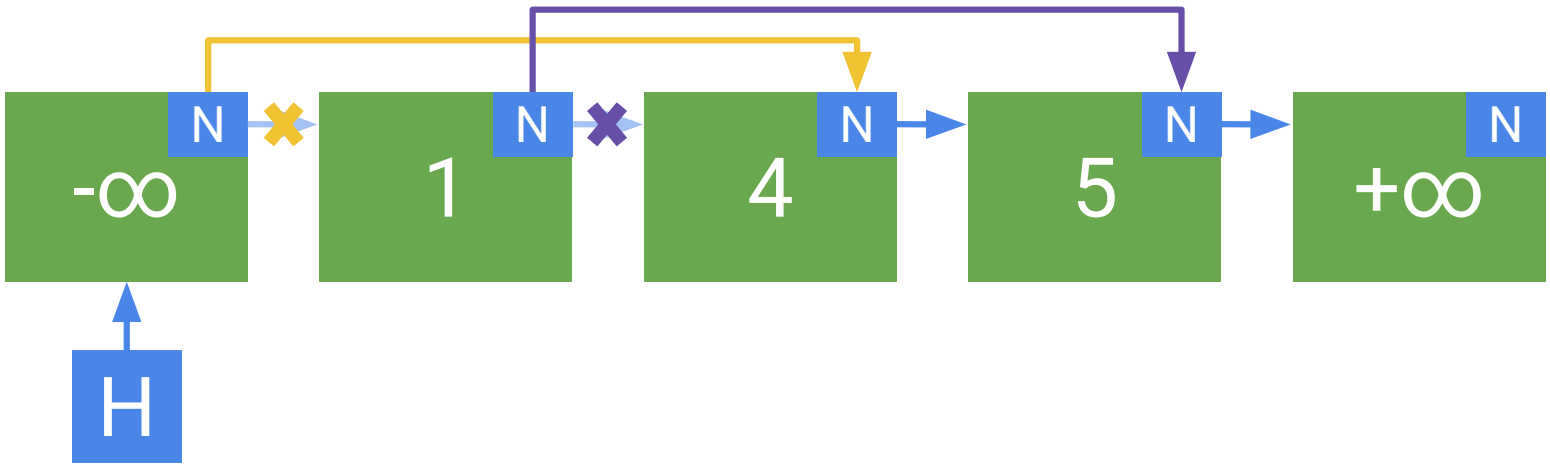
Неблокирующая модификация

- Вернёмся к множеству на односвязном списке
- Просто CAS недостаточно, не работает remove



Неблокирующая модификация

- Вернёмся к множеству на односвязном списке
- Просто CAS недостаточно, не работает remove



Всё оттого, что мы не знали, что «1» уже удалили

Неблокирующая модификация

- Объединим N и $\mathrm{removed}$ в одну переменную, пару $(\mathrm{N}, \mathrm{removed})$
- Будем менять $(\mathrm{N}, \mathrm{removed})$ атомарно
- Каждая операция модификации будет выполняться одним успешным CAS-ом
- В Java для этого есть $\mathrm{AtomicMarkableReference}$

Неблокирующая модификация

- Объединим N и removed в одну переменную, пару (N, removed)
- Будем менять (N, removed) атомарно
- Каждая операция модификации будет выполняться одним успешным CAS-ом
- В Java для этого есть AtomicMarkableReference

```
class Node (
  var NR: (Node, Boolean),
  val key: Int
)
```

Поиск окна

```
fun findWindow(key: Int): (Node, Node) = while(true) {
    var cur := head, next := cur.N
    while (next.key < key):
        (node, removed) := next.NR
        if (removed) { {
            // remove from the list
            if (!cur.NR.CAS( (next, false), (node, false) ) continue
            next = node
        } else {
            cur = next
            next = cur.N
        }
    // TODO: check if `next` is not removed
    return (cur, next)
}
```

Поиск

```
fun contains(key: Int): Boolean {
    (cur, next) := findWindow(key)
    return next.key == key
}
```

Поиск может не удалять узлы физически

Добавление

```
fun add(key: Int) = while(true) {
    (cur, next) := findWindow(key)
    if (next.key == key) return
    node := Node(key, (next, false))
    if (cur.NR.CAS( (next, false), (node, false) )) return
}
```

Удаление

```
fun remove(key: Int) = while(true) {
    (cur, next) := findWindow(key)
    if (next.key != key) return // not found
    (nextNext, _) := next.NR
    if (next.NR.CAS( (nextNext, false), (nextNext, true) )) {
        // help `findWindow` to remove `next`
        cur.NR.CAS( (next, false), (nextNext, false) )
        return // removed
    }
}
```

Спасибо за внимание!

Практические построения на списках

Roman Elizarov[1] Nikita Koval[2]

[1]Kotlin Team Lead, JetBrains
elizarov@gmail.com

[2]Researcher, JetBrains
PhD student, IST Austria
ndkoval@ya.ru

ИТМО 2018