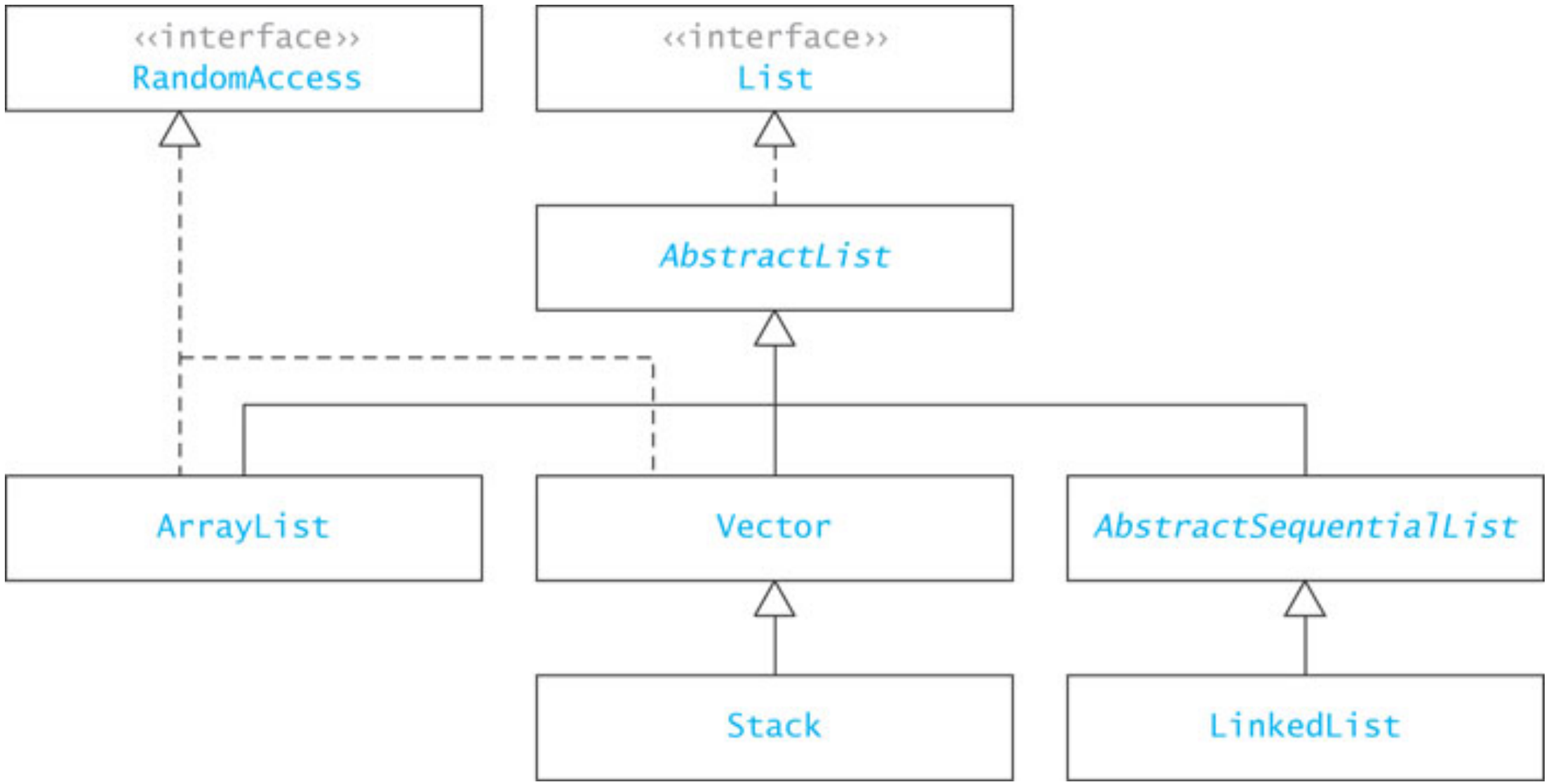
Listor

Koffman & Wolfgang

- kapitel 2, avsnitt 2.1–2.3, 2.5–2.6 och 2.9

Figur 2.1, sid 63

java.util.List, med dess implementeringar



List och primitiva typer

En array kan innehålla primitiva typer:

- int[], double[], etc.

Övriga container-klasser kan ***inte*** det, utan måste ha objekt som element

Alltså måste vi använda objekt-motsvarigheterna Integer, Double, etc:

- ArrayList<Integer>
- LinkedList<Double>

Men för det mesta så översätter Java automagiskt mellan primitiva typer och motsvarande objekt

Tabell 2.1, sid 68

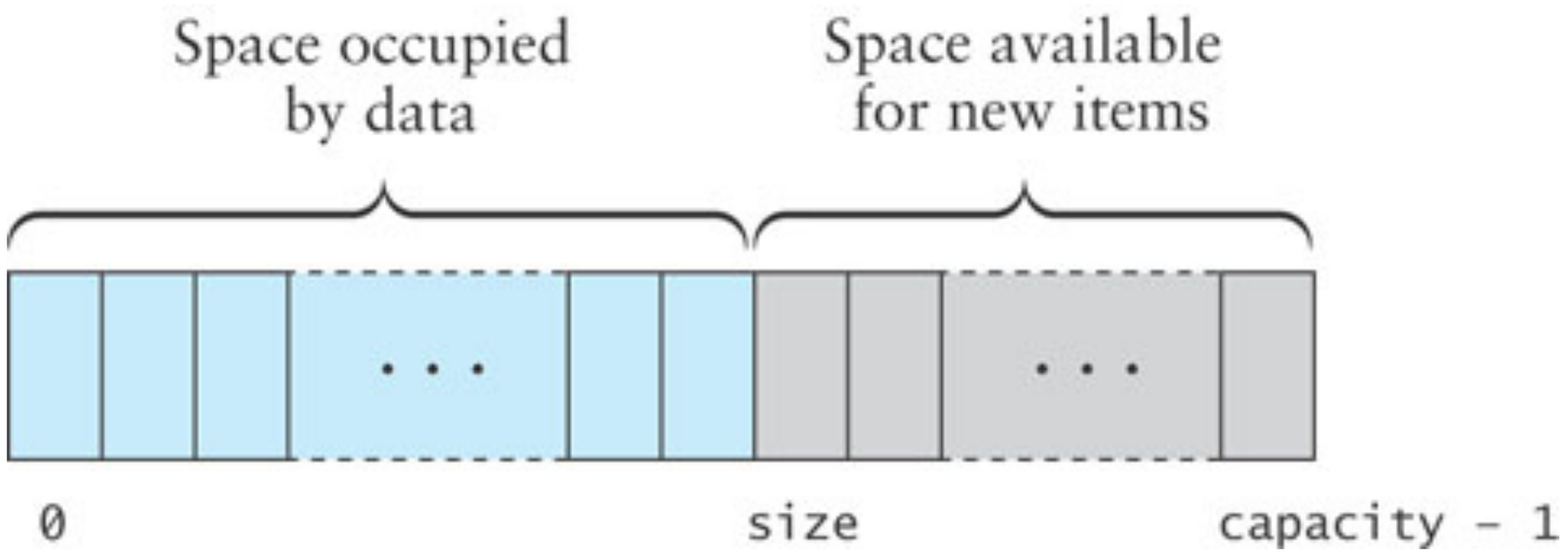
Utvalda metoder från List<E> interface:

Method	Behavior
`public E get(int index)`	Returns a reference to the element at position `index`.
`public E set(int index, E anEntry)`	Sets the element at position `index` to reference `anEntry`. Returns the previous value.
`public int size()`	Gets the current size of the `ArrayList`.
`public boolean add(E anEntry)`	Adds a reference to `anEntry` at the end of the `ArrayList`. Always returns `true`.
`public void add(int index, E anEntry)`	Adds a reference to `anEntry`, inserting it before the item at position `index`.
`int indexOf(E target)`	Searches for `target` and returns the position of the first occurrence, or –1 if it is not in the `ArrayList`.
`public E remove(int index)`	Returns and removes the item at position `index` and shifts the items that follow it to fill the vacated space.

Figur 2.3, sid 71

Hur KWArrayList fungerar:

- en array theData
- två variabler, size och capacity



Att skapa ett fält av typen E[]

```
public KWArrayList () {
        capacity = INITIAL_CAPACITY;
        theData = (E[]) new Object[capacity];
}
```

Att skapa ett fält av typen `E[]`

```
public KWArrayList () {
        capacity = INITIAL_CAPACITY;
        theData = (E[]) new Object[capacity];
}
```

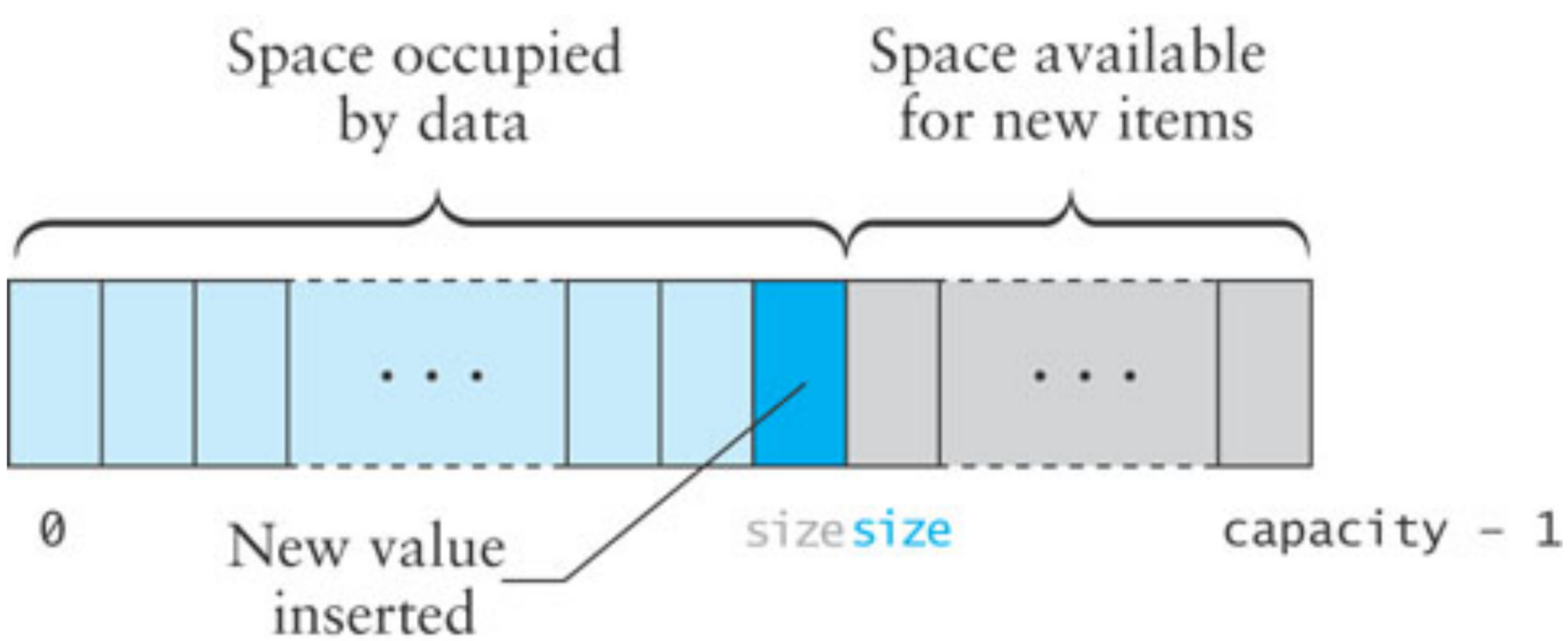
Denna sats allokerar plats för ett fält av typen `Object[]` och kastar det sedan till rätt typ `E[]`.

Detta beror på en bugg i Java-specifikationen, och är förhoppningsvis enda gången ni ska behöva använda type casting,

Figur 2.4, sid 73

För att lägga till ett element stoppar vi bara in det på sista-positionen, och ökar sista-positionen med 1

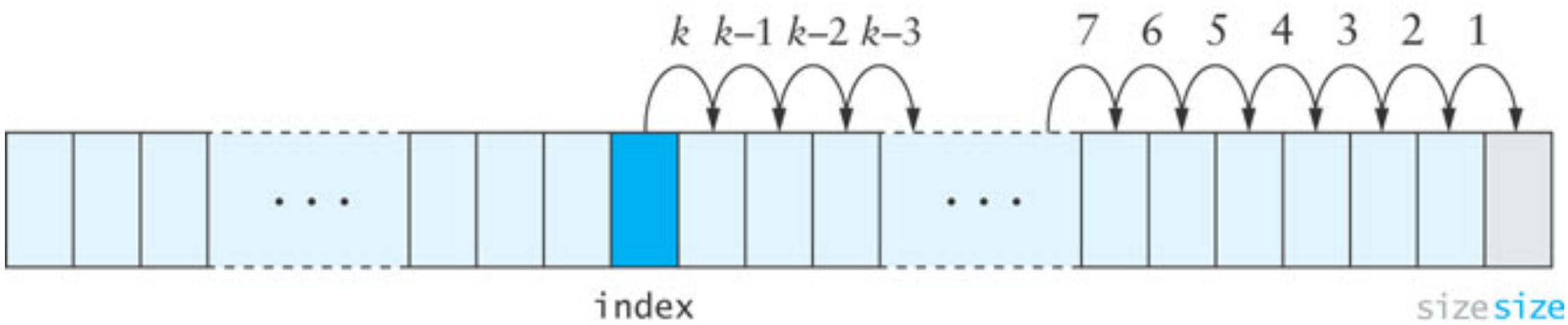
- under förutsättning att vi har plats kvar i det interna fältet



Figur 2.5, sid 74

För att lägga till något i mitten av fältet, behöver vi flytta fram de efterföljande elementen an plats

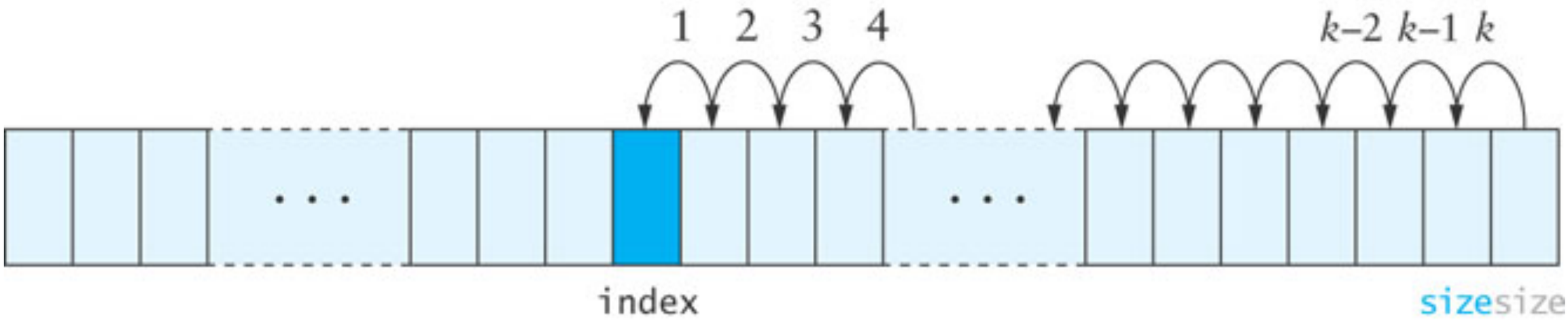
- vi måste börja flytta det sista elementet, och sedan gå baklänges genom fältet



Figur 2.6, sid 76

För att ta bort ett element, måste vi flytta efterföljarna ett steg framåt i fältet

- vi måste börja med det närmaste elementet och sedan fortsätta framåt i fältet till slutet



Om-allokering

När det interna fältet blir för stort skapar vi ett nytt större och kopierar alla gamla data

```
private void reallocate () {
    capacity *= 2;
    theData = Arrays.copyOf(theData, capacity);
}
```

Det nya fältet är dubbelt så stort som det gamla

- vi dubblerar istället för att öka med en konstant
- då får vi konstant *amorterad* tidskomplexitet
- dvs, i *medeltal* blir tidsåtgången konstant (inte beroende av storleken på fältet)

Enkellänkade listor

Att lägga till och ta bort element inuti en ArrayList tar linjär tid O(n)

- vilket är oändligt långsamt i vissa applikationer

En länkad lista kan lägga till och ta bort varsomhelt i en lista i konstant tid, O(1)

- å andra sidan tar det linjär tid O(n) att gå till en specifik position i en länkad lista, vilket är konstant i en ArrayList

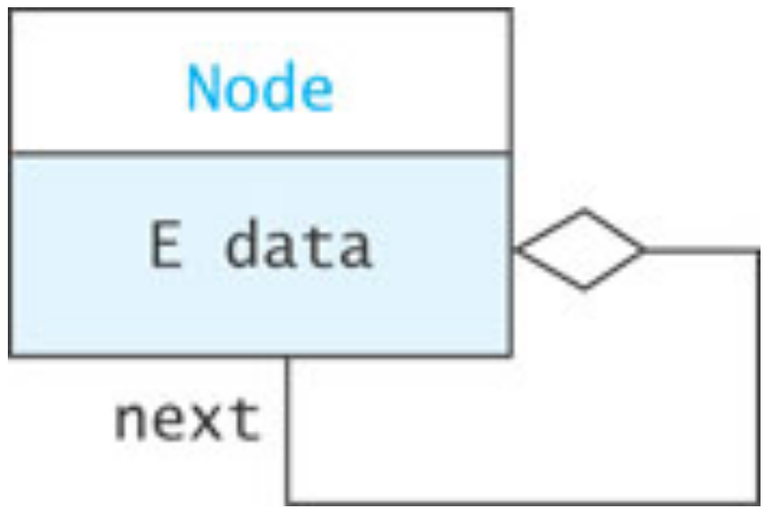
I en länkad lista så är varje element (”nod”) länkad till det efterföljande elementet

Figur 2.15, sid 90

En enkellänkad listnod innehåller

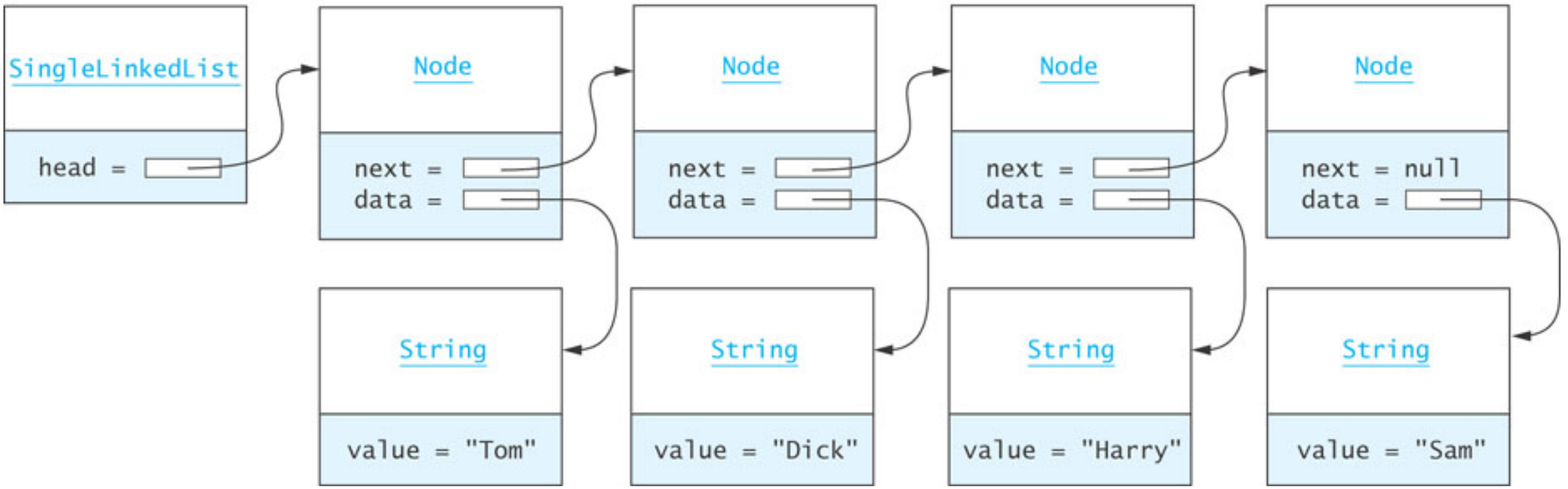
- noddata, och
- en länk till efterföljaren

```
private static class Node<E> {
        private E data;
        private Node<E> next;
}
```

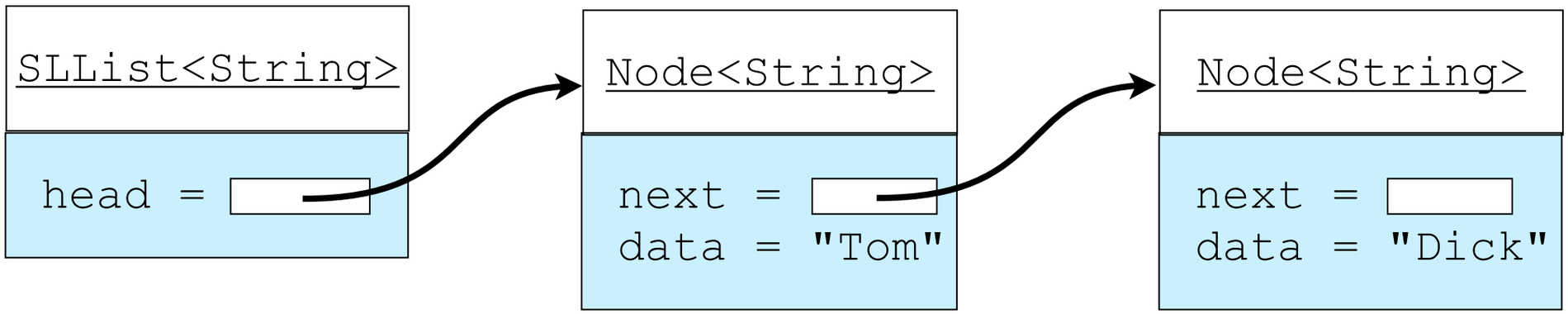


Figur 2.16, sid 90

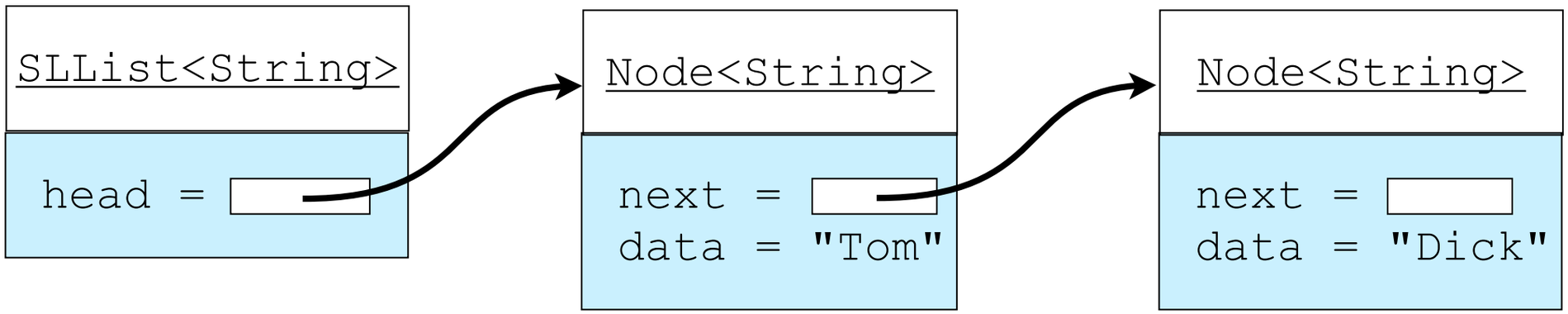
Länkad representation av listan
{”Tom”, ”Dick”, ”Harry”, ”Sam”}



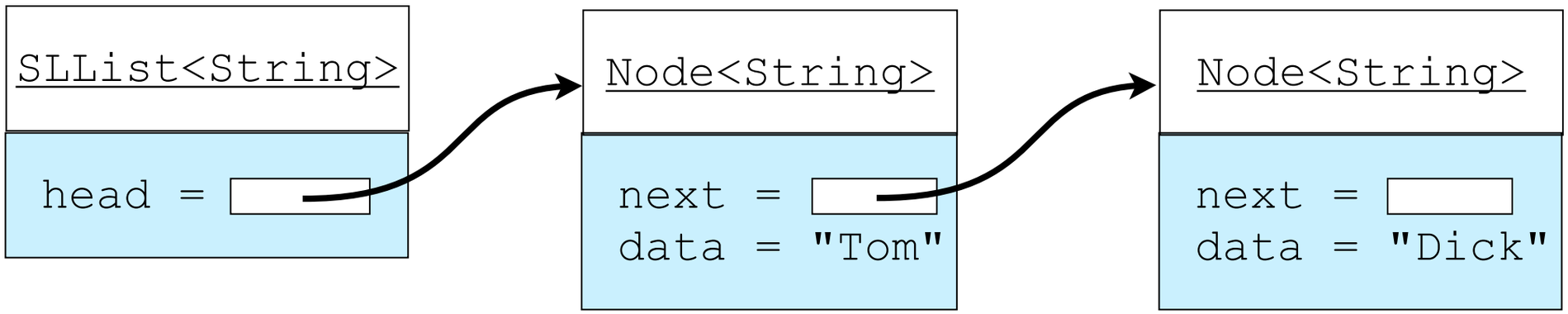
Exempellista

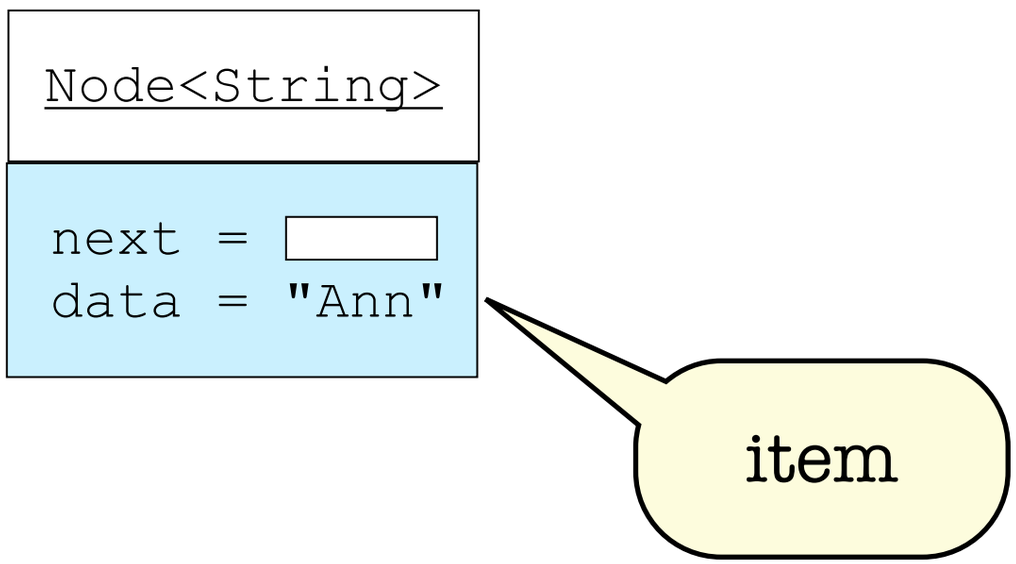


Metoden addFirst(E item)

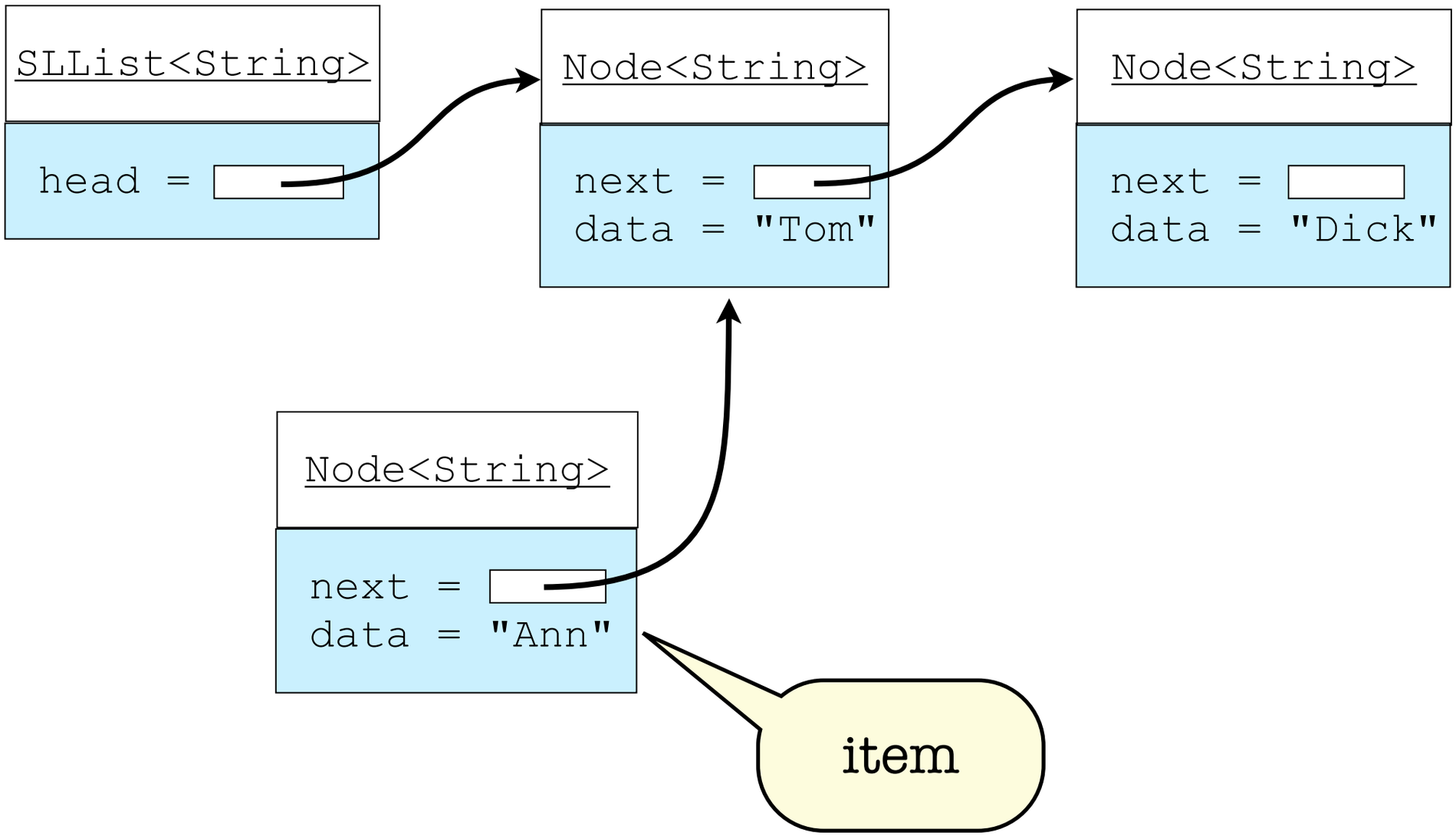


Metoden addFirst(E item)

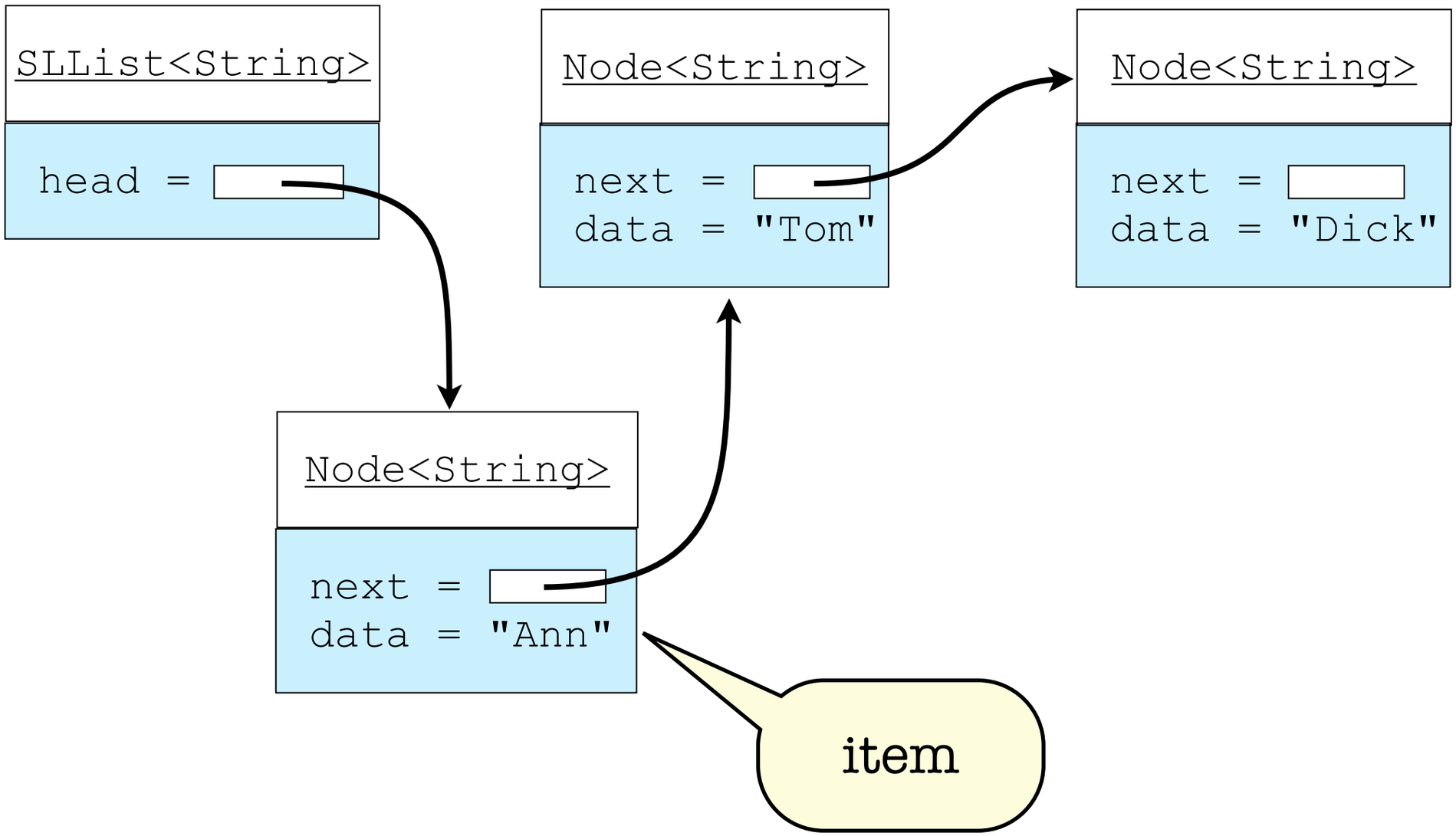




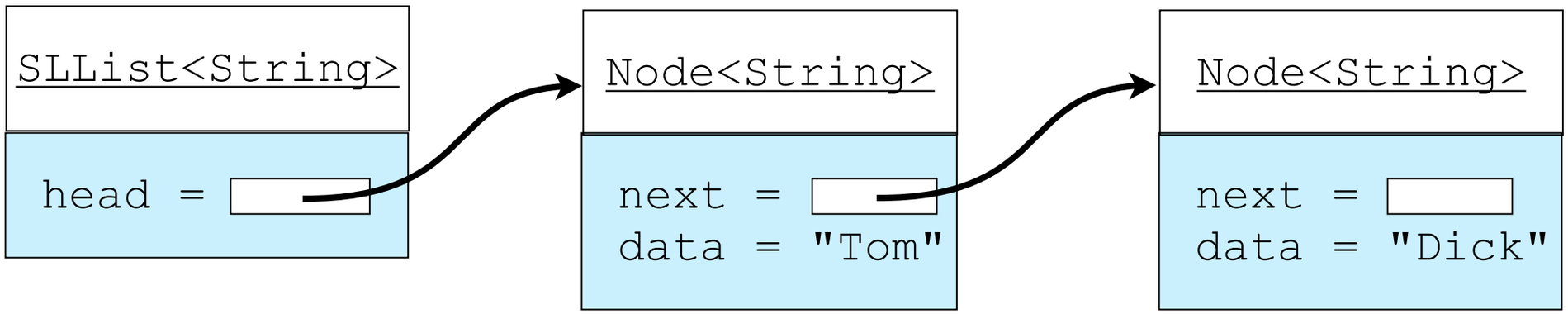
Metoden addFirst(E item)



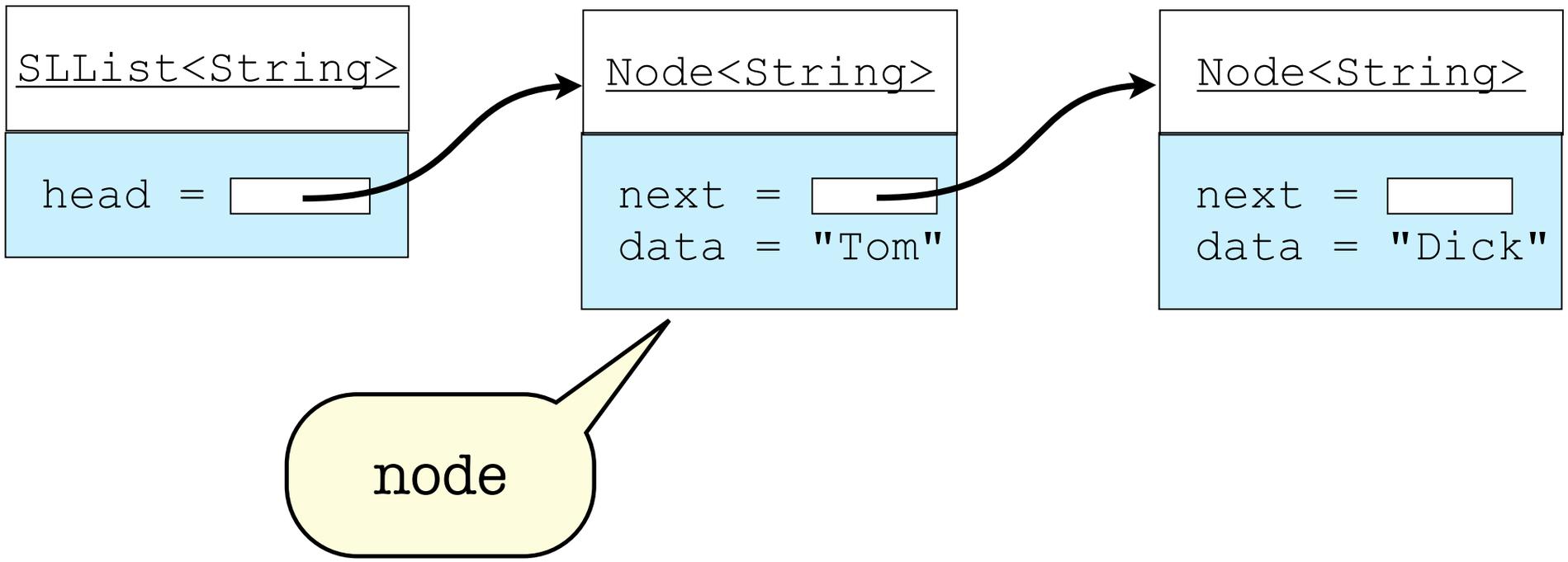
Metoden addFirst(E item)



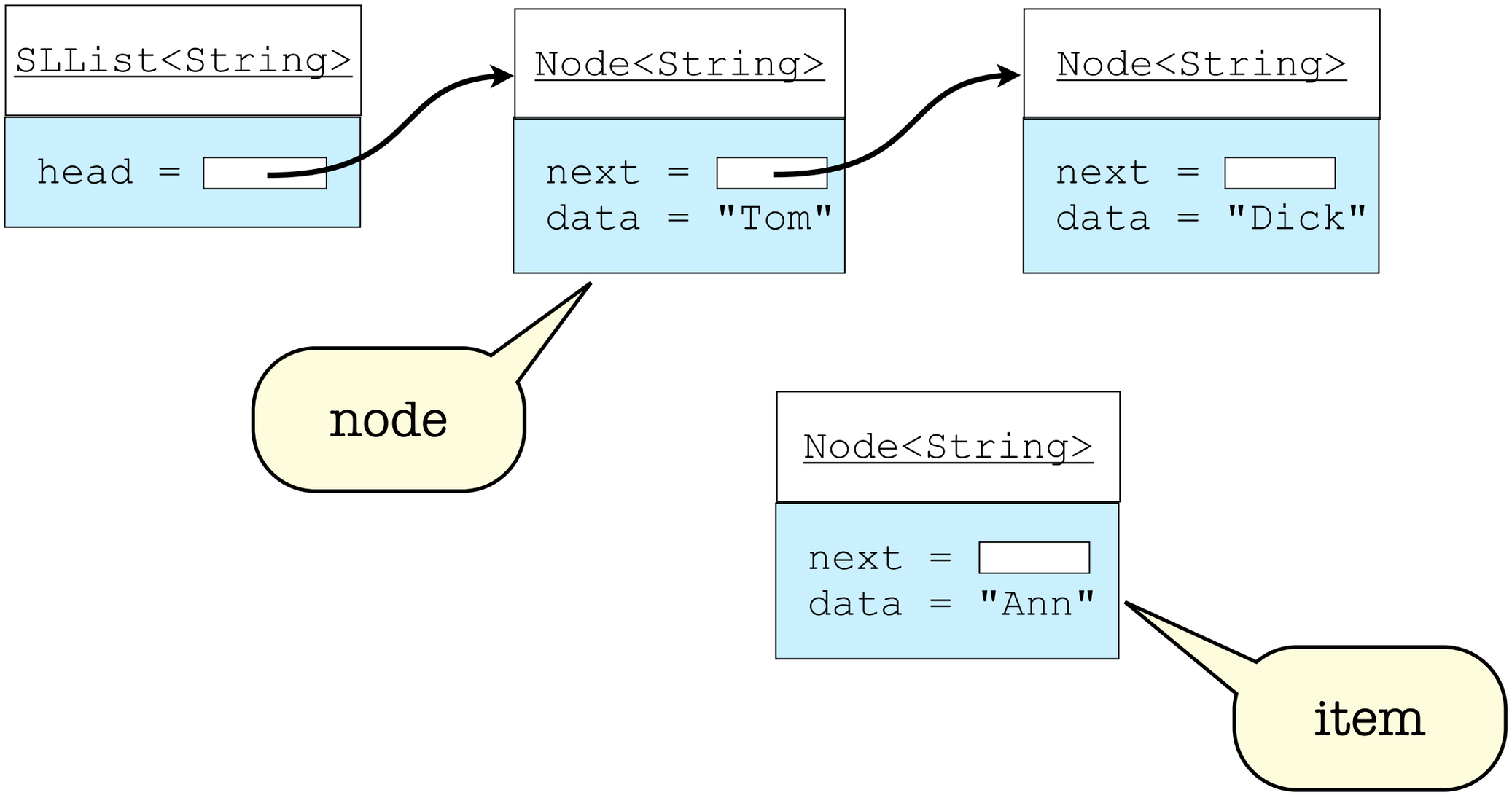
addAfter(Node<E> node, E item)



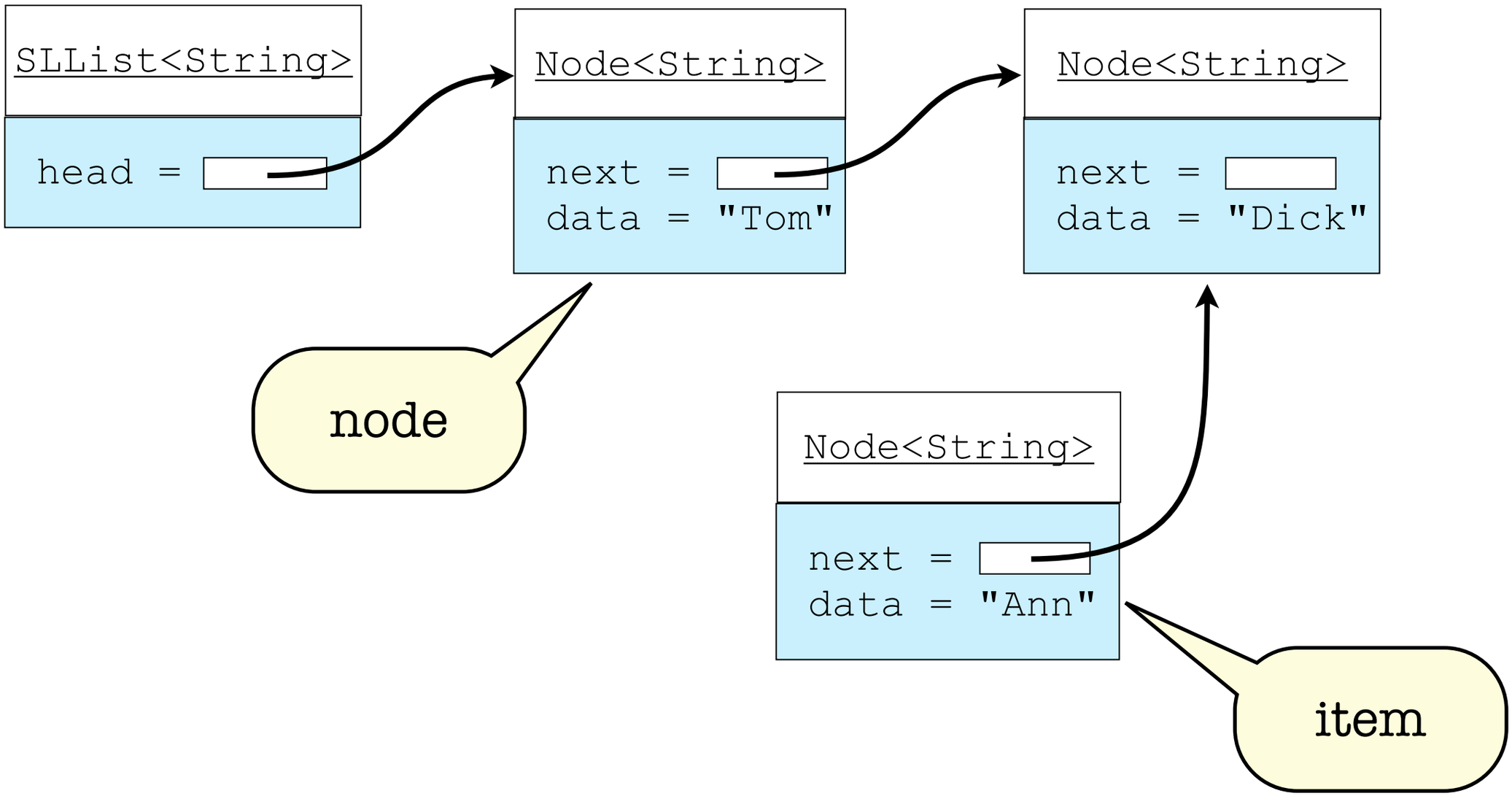
addAfter(Node<E> node, E item)



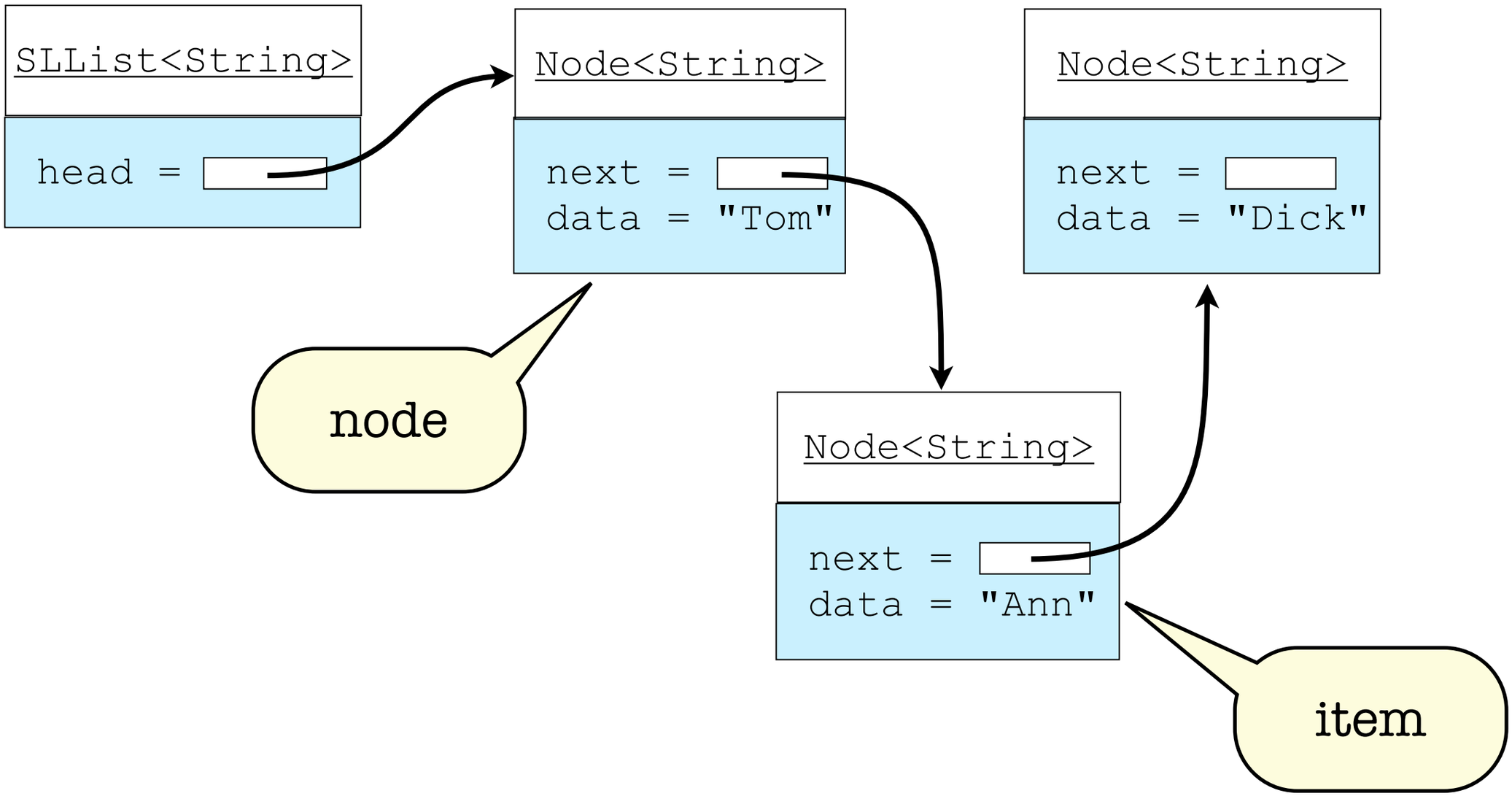
addAfter(Node<E> node, E item)



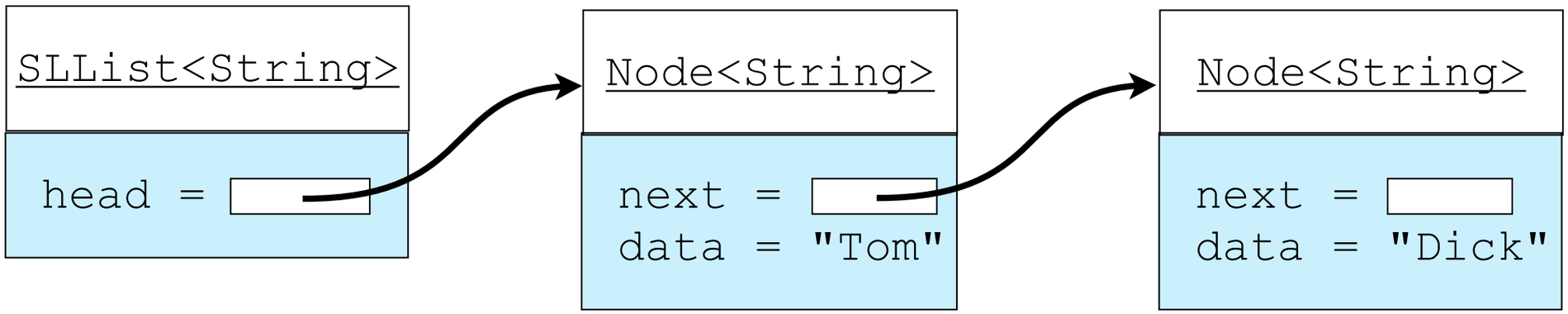
addAfter(Node<E> node, E item)



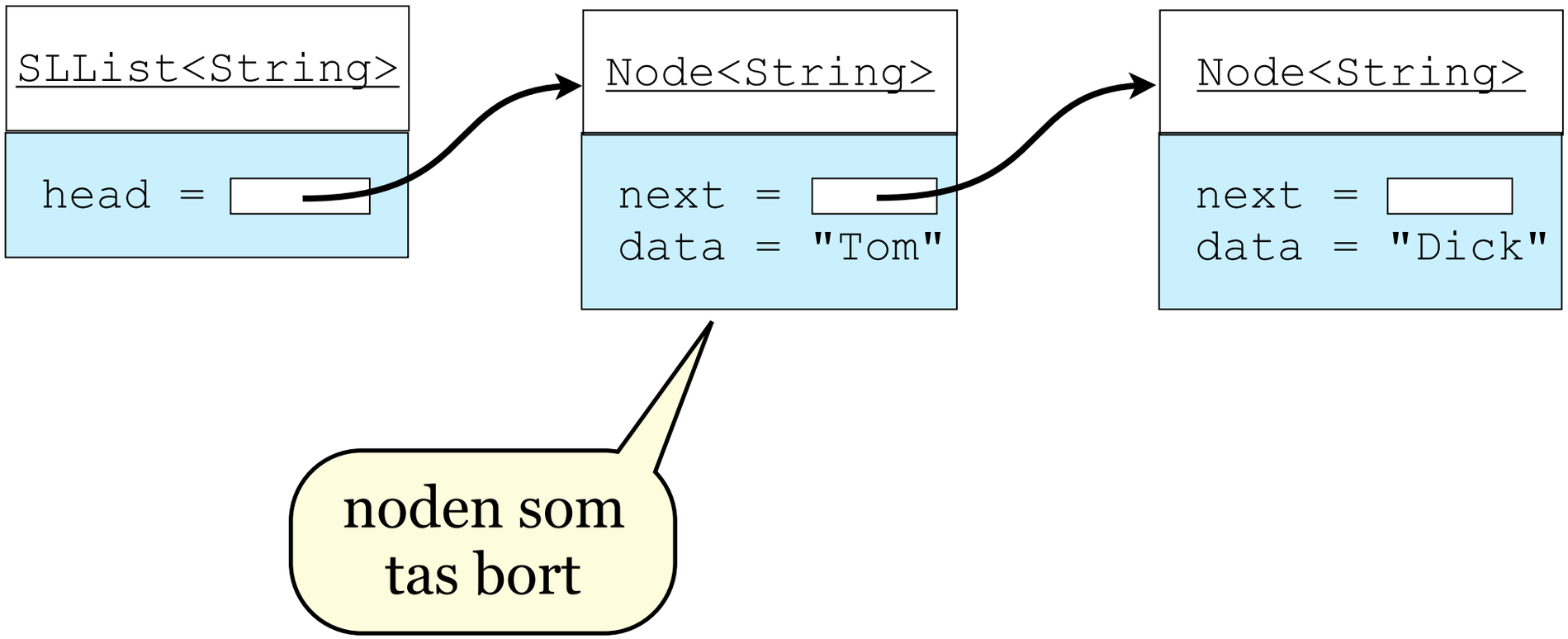
addAfter(Node<E> node, E item)



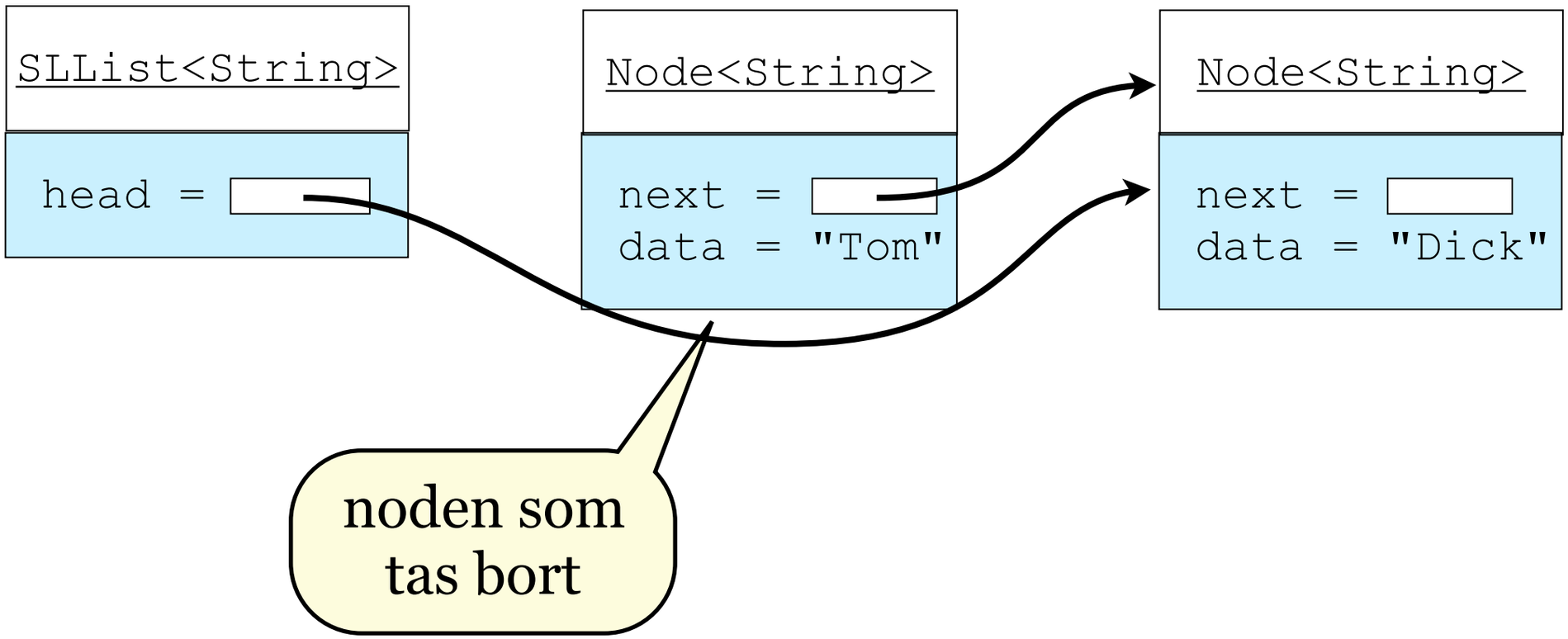
removeFirst()



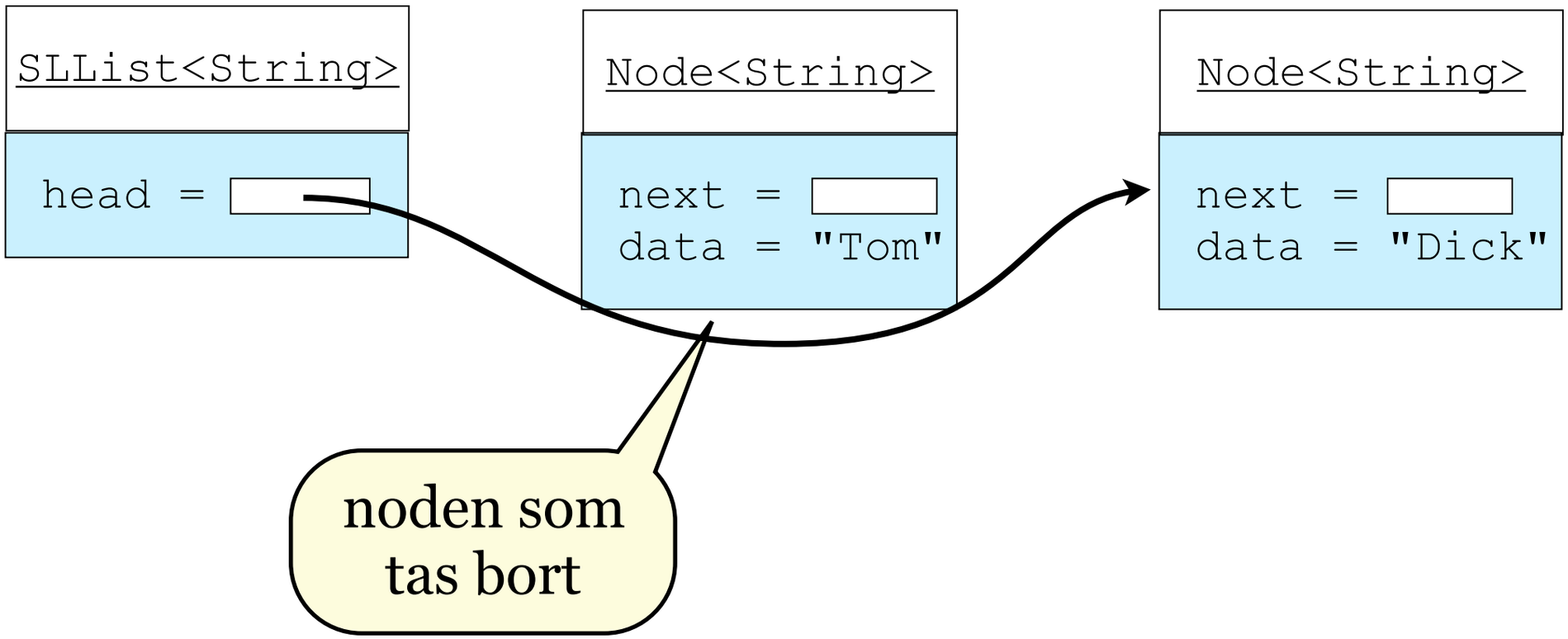
removeFirst()



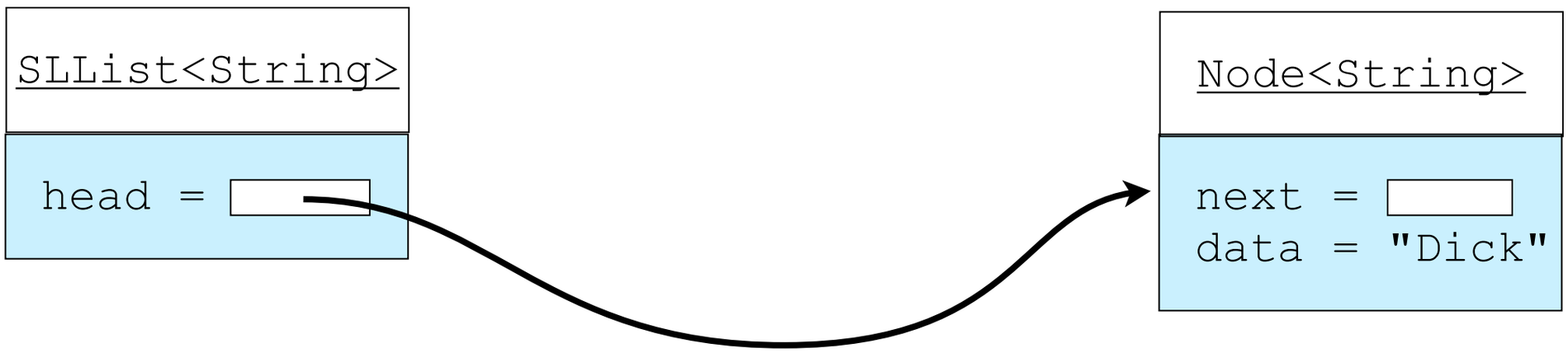
removeFirst()



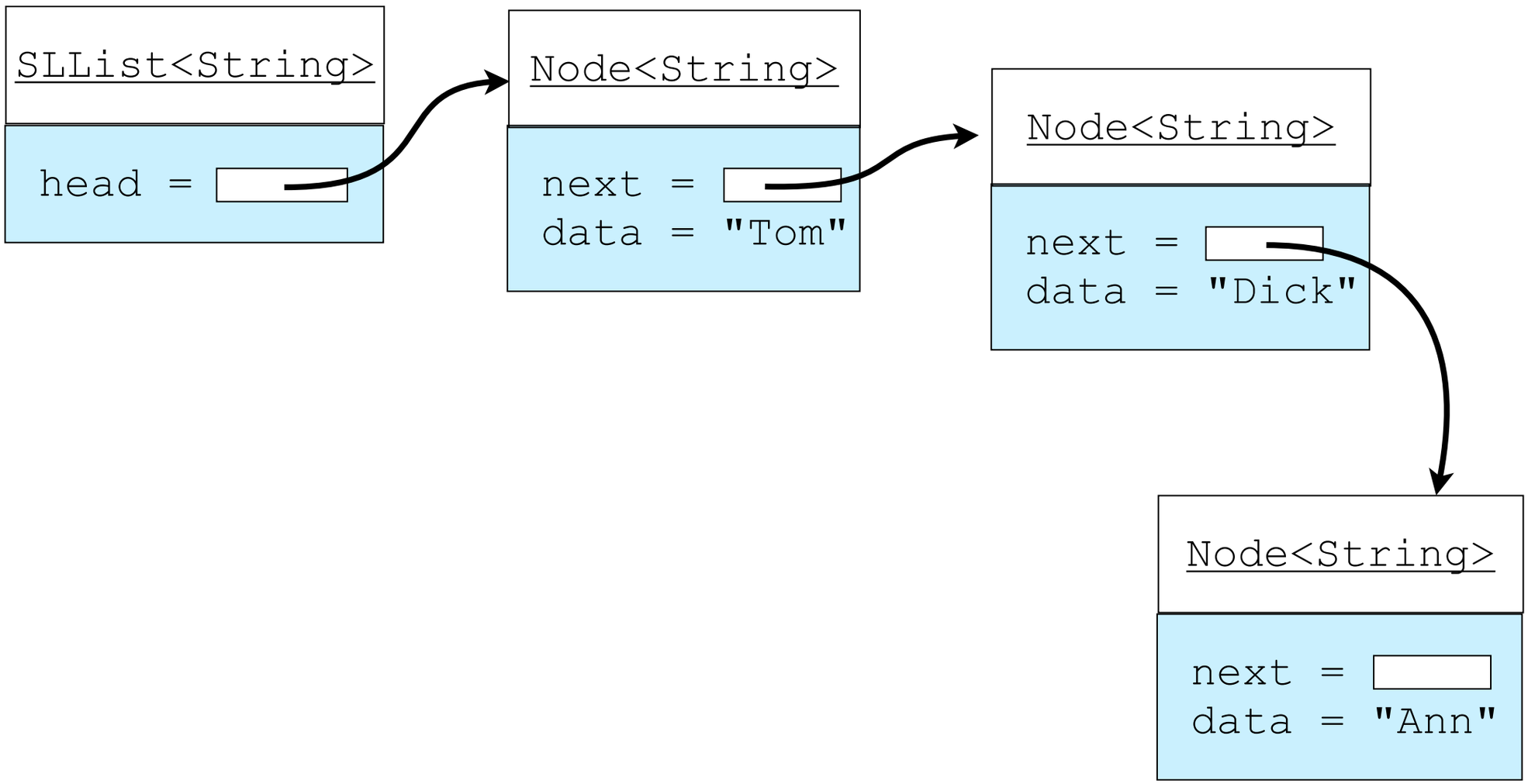
removeFirst()



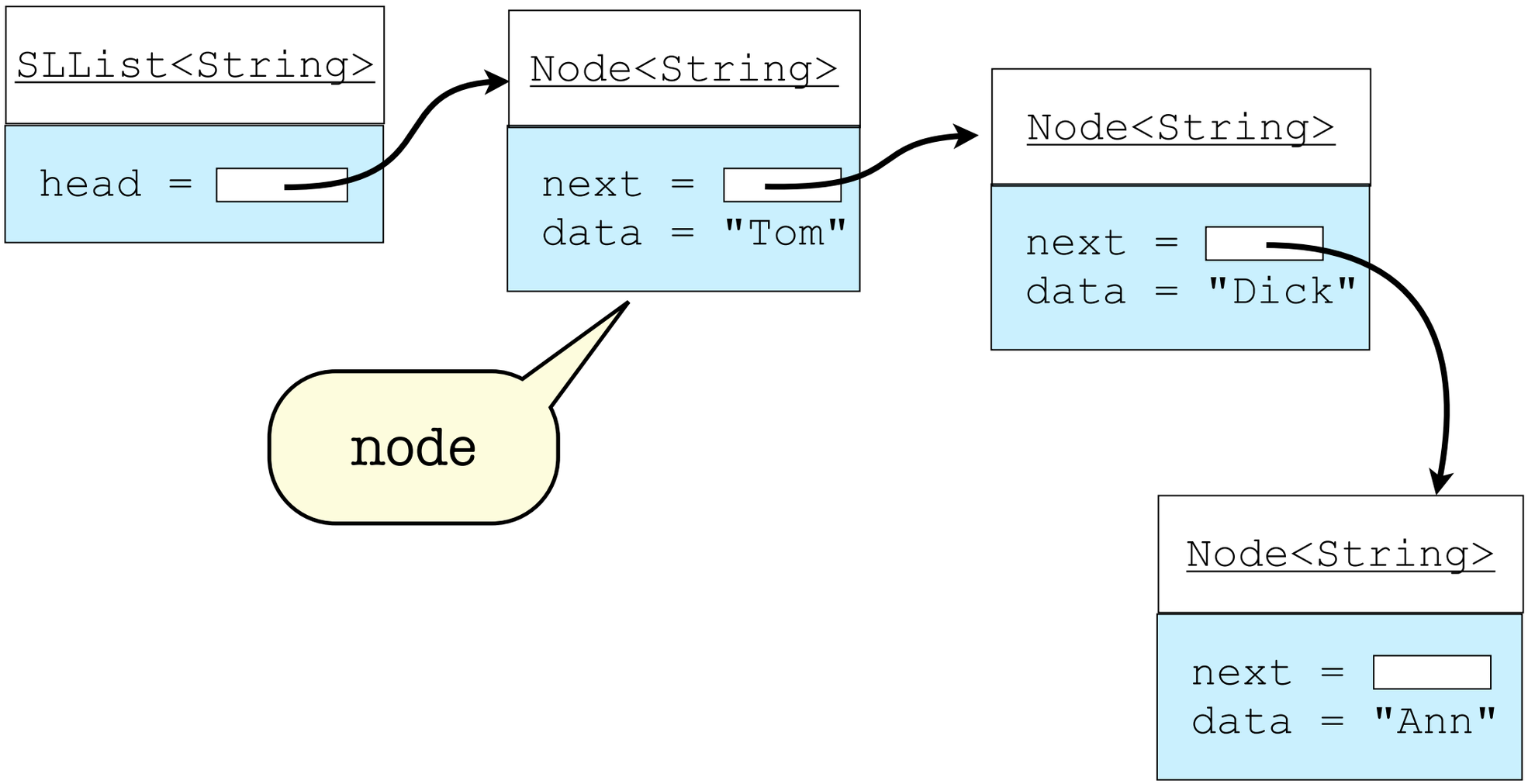
removeFirst()



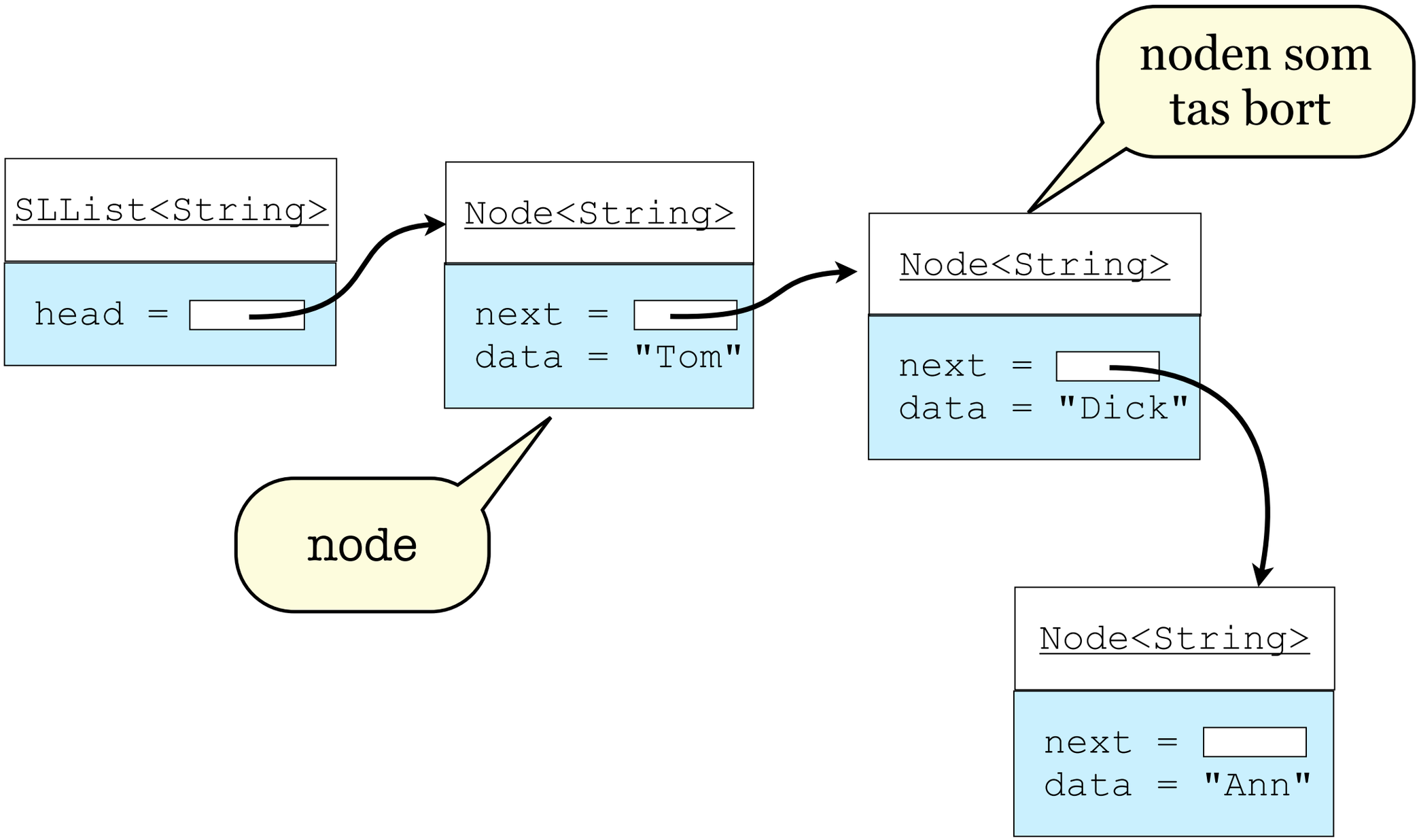
removeAfter(Node<E> node)



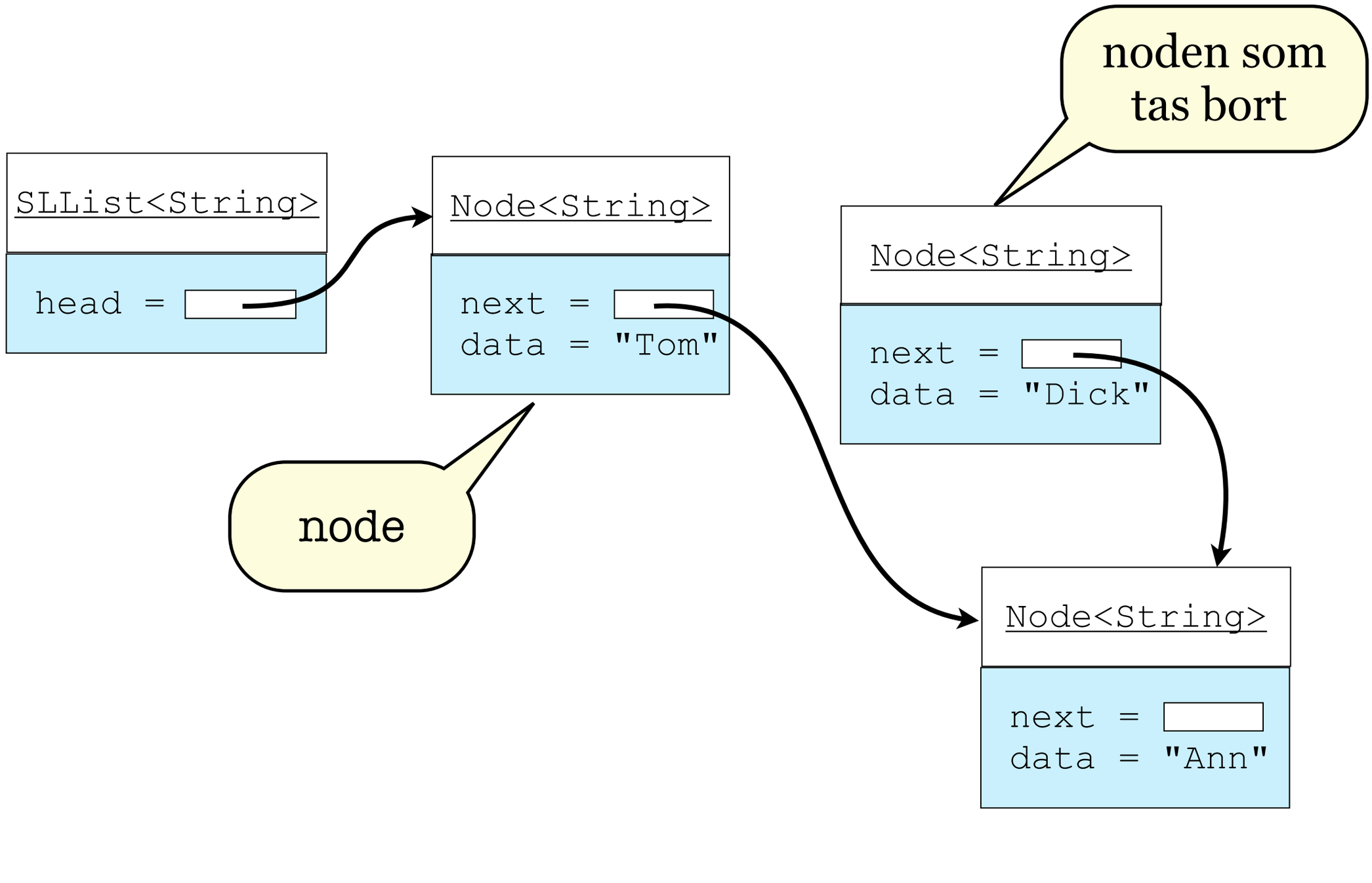
removeAfter(Node<E> node)



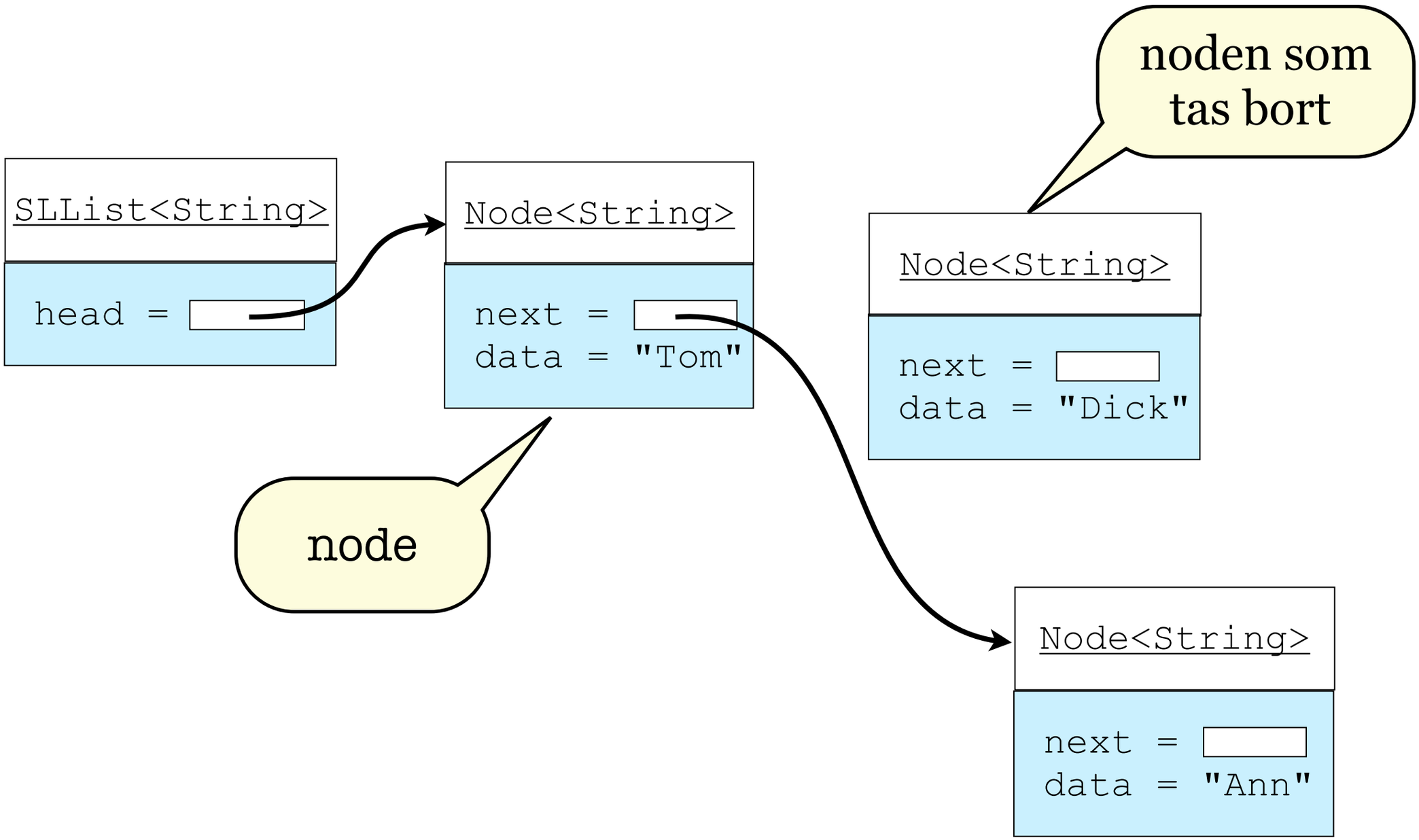
removeAfter(Node<E> node)



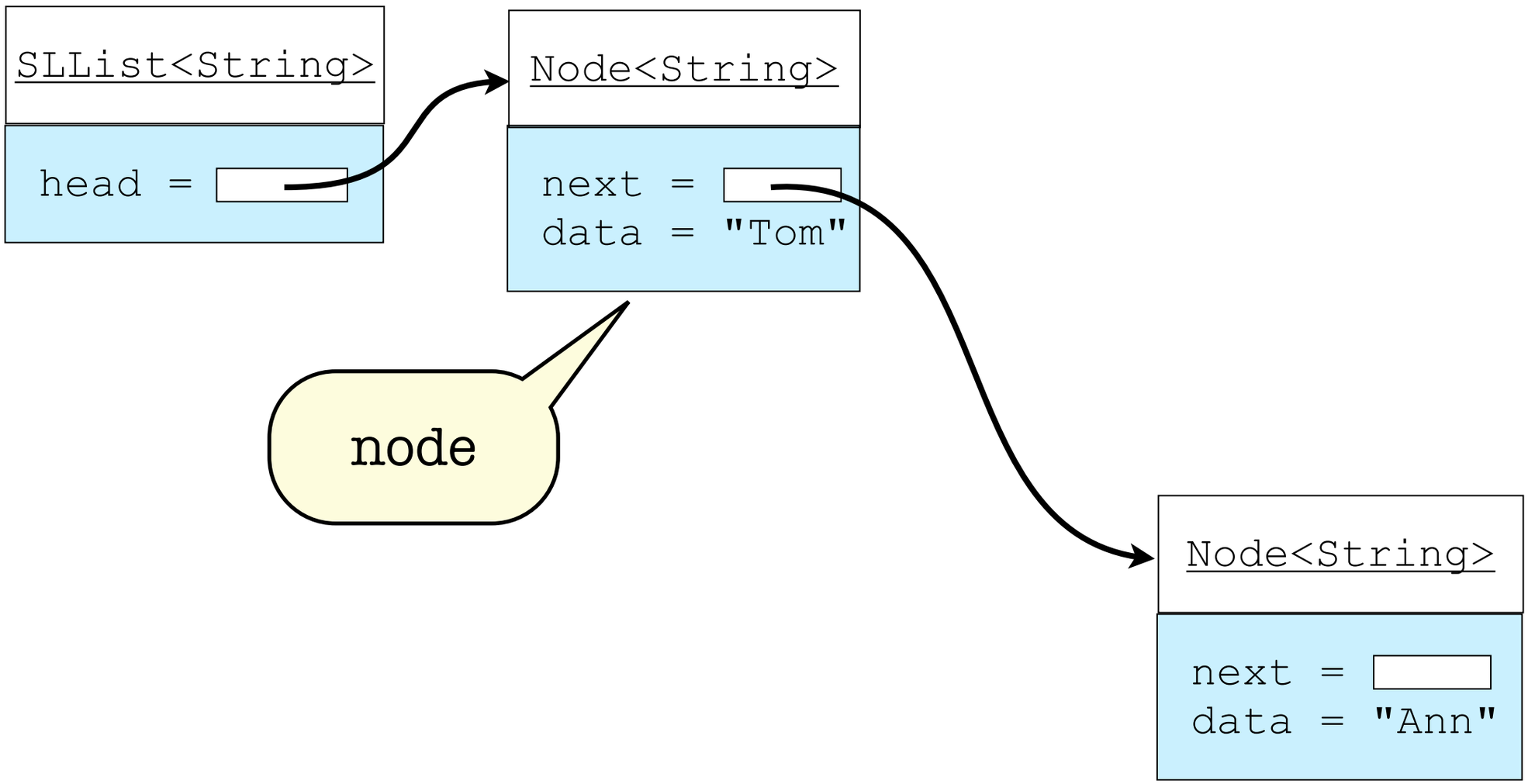
removeAfter(Node<E> node)



removeAfter(Node<E> node)



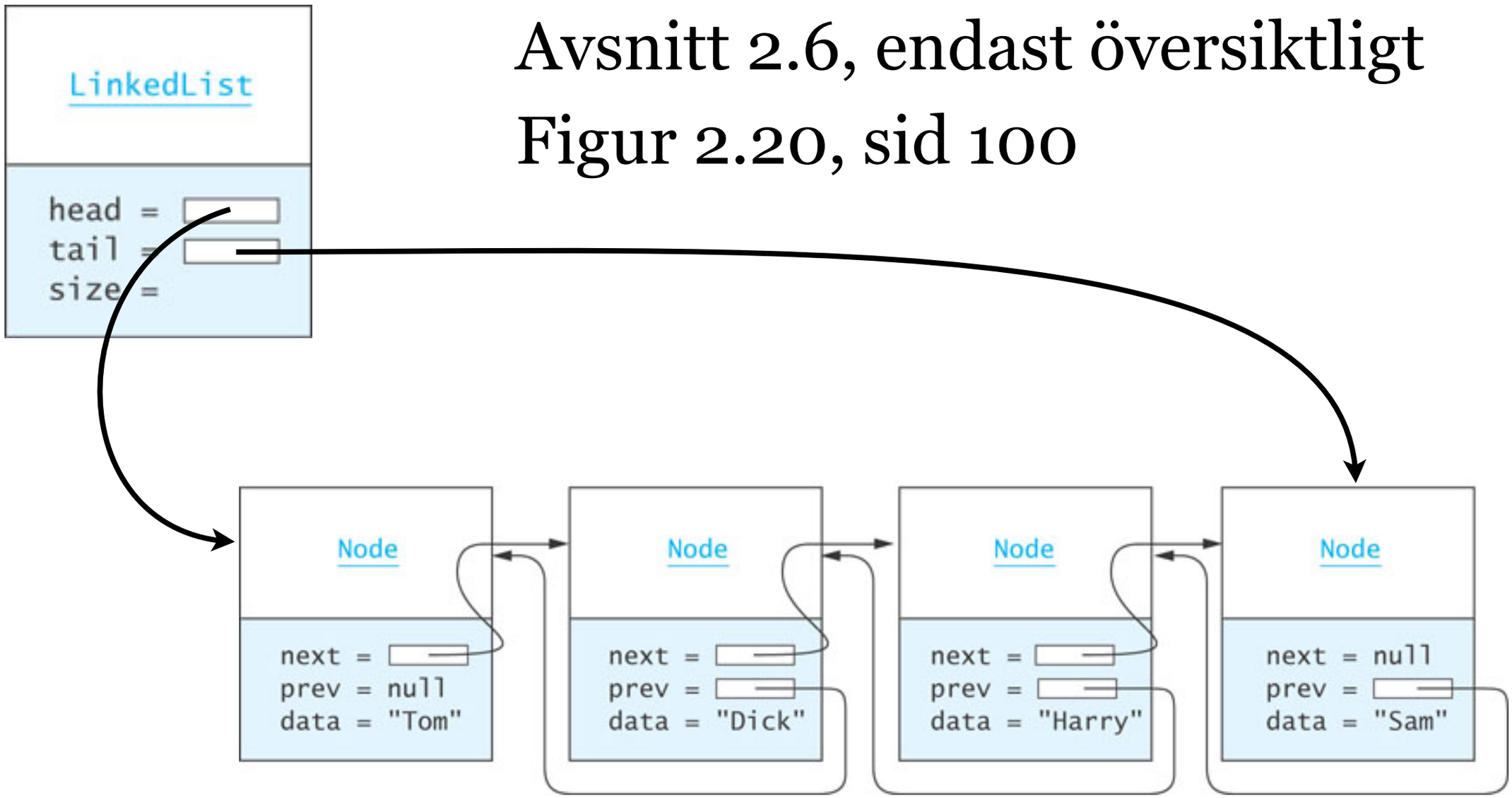
removeAfter(Node<E> node)



Dubbellänkade listor

Avsnitt 2.6, endast översiktligt

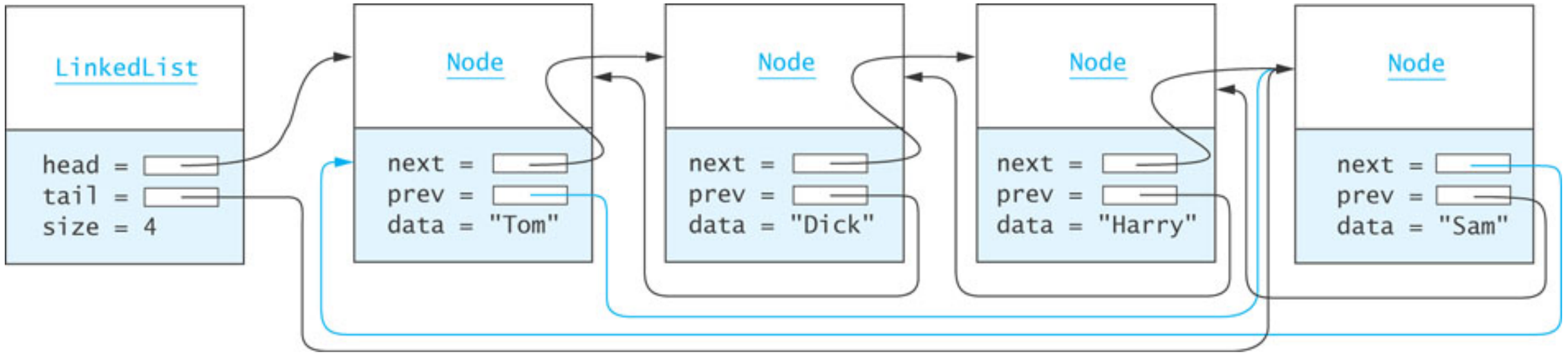
Figur 2.20, sid 100



Cirkulära dubbellänkade listor

Avsnitt 2.6, endast översiktligt

Figur 2.25, sid 103



Klassen LinkedList

java.util.LinkedList<E>

Method	Behavior
`public void add(int index, E obj)`	Inserts object `obj` into the list at position `index`.
`public void addFirst(E obj)`	Inserts object `obj` as the first element of the list.
`public void addLast(E obj)`	Adds object `obj` to the end of the list.
`public E get(int index)`	Returns the item at position `index`.
`public E getFirst()`	Gets the first element in the list. Throws `NoSuchElementException` if the list is empty.
`public E getLast()`	Gets the last element in the list. Throws `NoSuchElementException` if the list is empty.
`public boolean remove(E obj)`	Removes the first occurrence of object `obj` from the list. Returns `true` if the list contained object `obj`; otherwise, returns `false`.
`public int size()`	Returns the number of objects contained in the list.

Collection interface

Specificerar en delmängd av metoderna i List interface

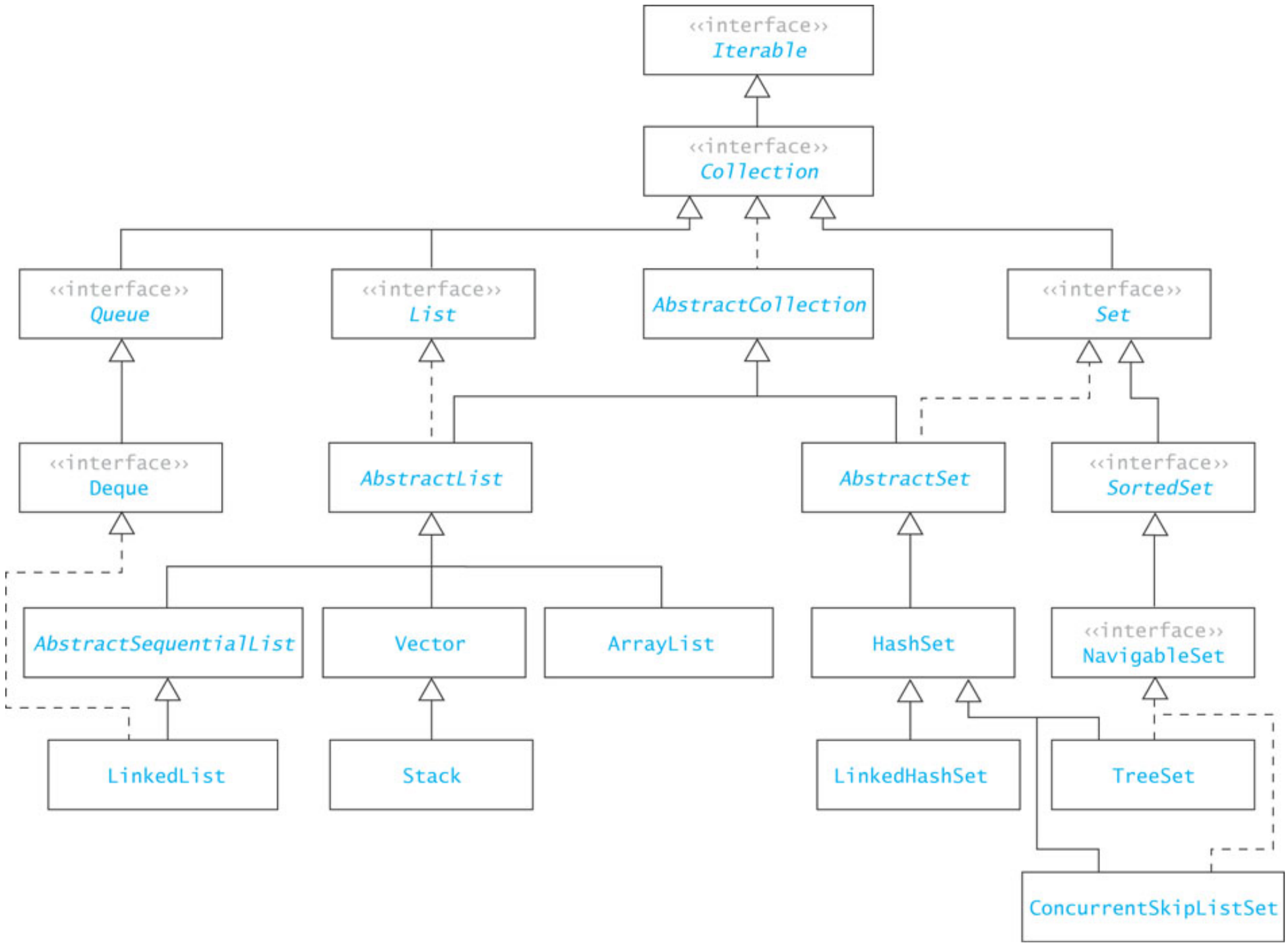
Bland annat:

- `add(E)`
- `remove(Object)`
- `iterator()`

Men inte:

- `add(int, E)`
- `get(int)`
- `remove(int)`
- `set(int, E)`

Java Collection Framework



Tabell 2.12, sid 123

java.util.Collection<E>

Method	Behavior
`boolean add(E obj)`	Ensures that the collection contains the object `obj`. Returns `true` if the collection was modified.
`boolean contains(E obj)`	Returns `true` if the collection contains the object `obj`.
`Iterator<E> iterator()`	Returns an `Iterator` to the collection.
`int size()`	Returns the size of the collection.

- ordningen mellan element är ***inte*** specificerad
- ett elements position är ***inte*** specificerad
- om ett element redan finns i samlingen, så är det ***inte*** säkert att det läggs till igen

AbstractCollection, AbstractList, och AbstractSequentialList

AbstractCollection<E>

- det räcker att implementera
  add(E), size(), iterator()

AbstractList<E>

- det räcker att implementera
  add(int, E), get(int), remove(int), set(int, E), size()

```
public class KWArrayList<E>
        extends AbstractList<E>  implements List<E>
```

AbstractSequentialList<E>

- det räcker att implementera
  listIterator(), size()

```
public class KWLinkedList<E>
        extends AbstractSequentialList<E>  implements List<E>
```