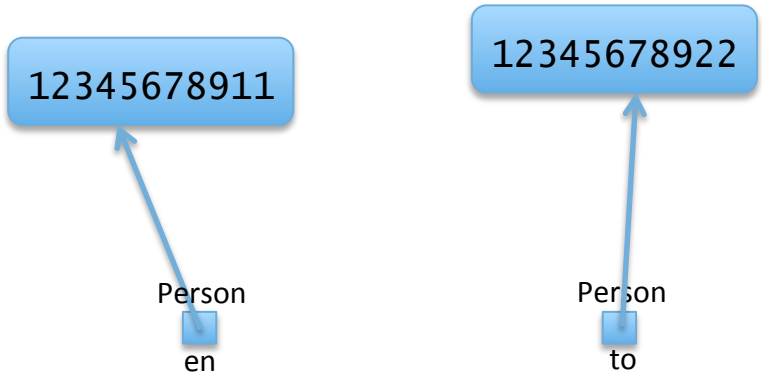
```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;         // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;        // this < enAnnen
       else        return  0;        // this == enAnnen
    }
}
```

```
interface Sammenlignbar<E> {
   int sammenlignMed(E e);
}
```

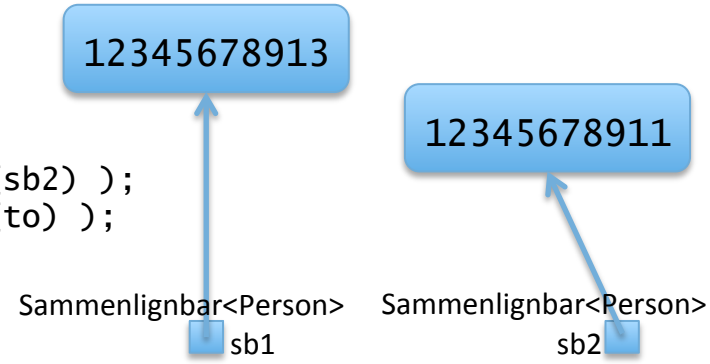
```
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```

```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
```

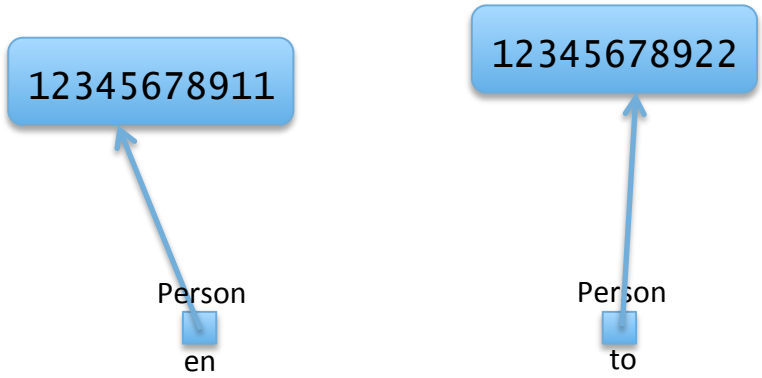


En datastrukturtegning beskriver en tilstand i maskinen når programmet kjører

```
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```



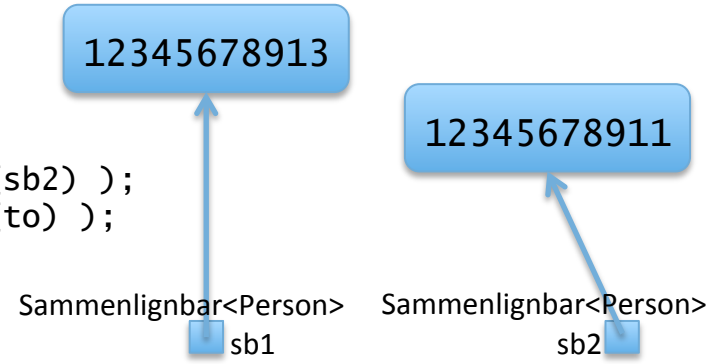
```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
```



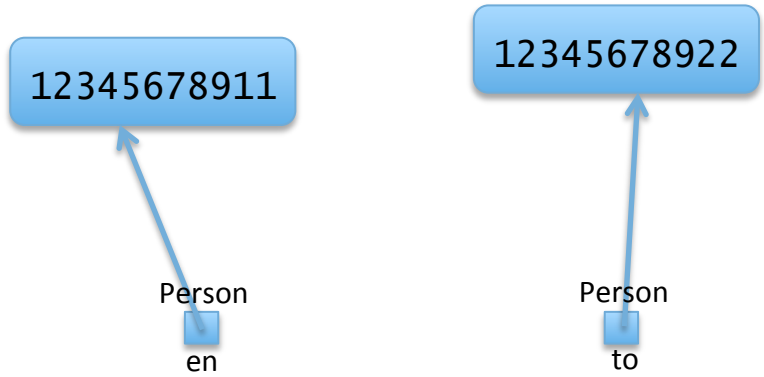
En datastrukturtegning beskriver en tilstand i maskinen når programmet kjører

Tilstandspåstand

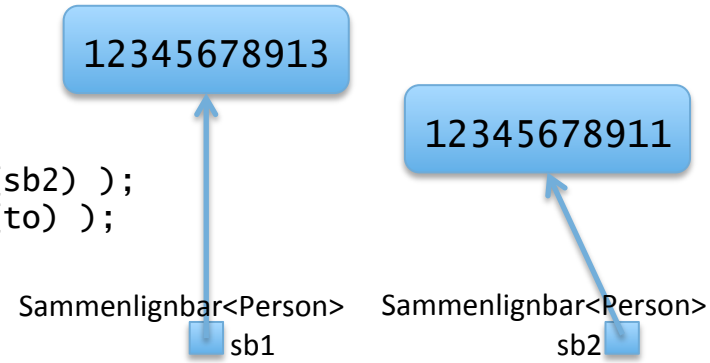
```
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```



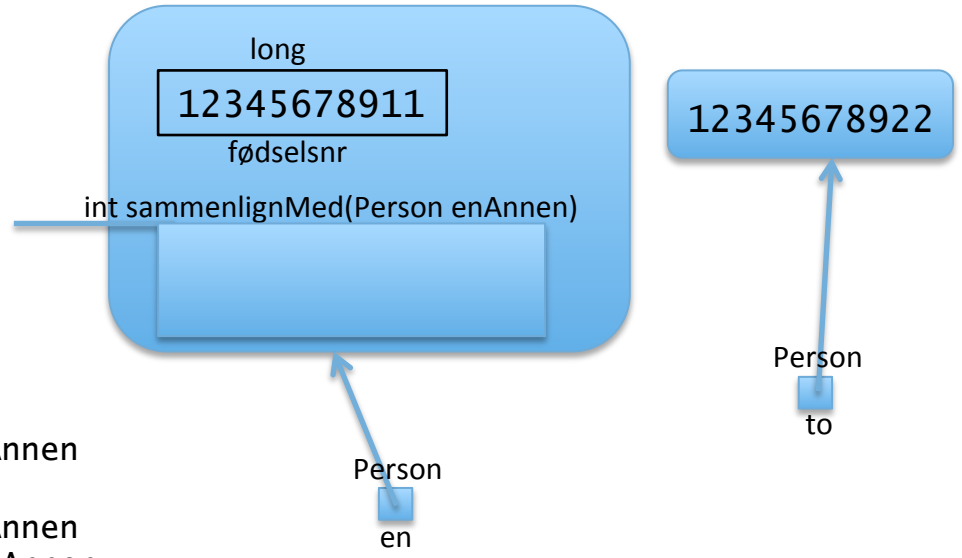
```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```



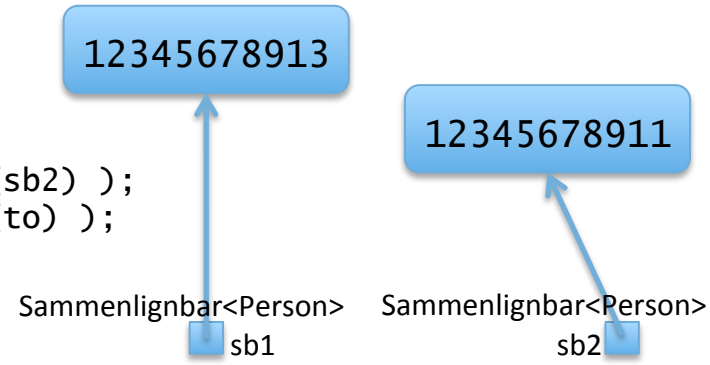

```
> java Sammenlign
123456789inf1010
en <= to
true
false
```



```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```




```
> java Sammenlign
123456789inf1010
en <= to
true
false
```



```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
```

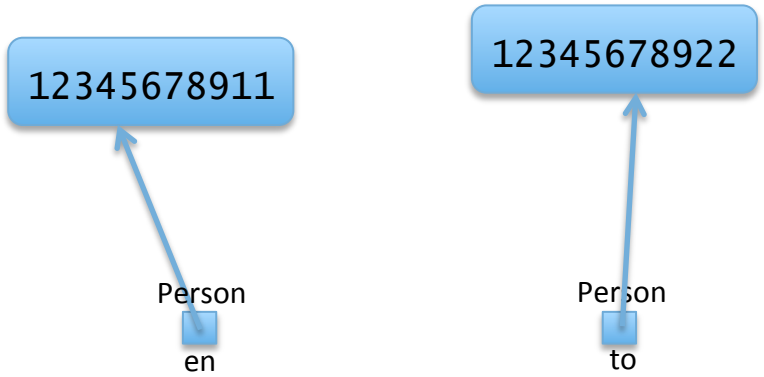
```
interface Sammenlignbar<E> {
   int sammenlignMed(E e);
}
```

```
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     // System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```

```
> javac Sammenlign.java
Sammenlign.java:19: error: incompatible types:
Sammenlignbar<Person> cannot be converted to Person
System.out.println
( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
                                            ^
```

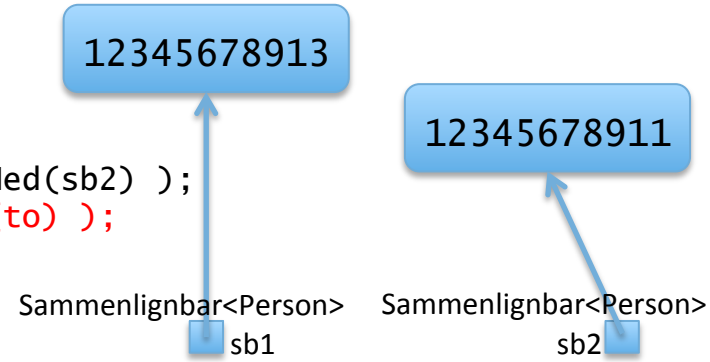
Kompileringsfeil, fordi aktuell parameter (sb2) *ikke har samme type som formell parameter*

```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     // System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```

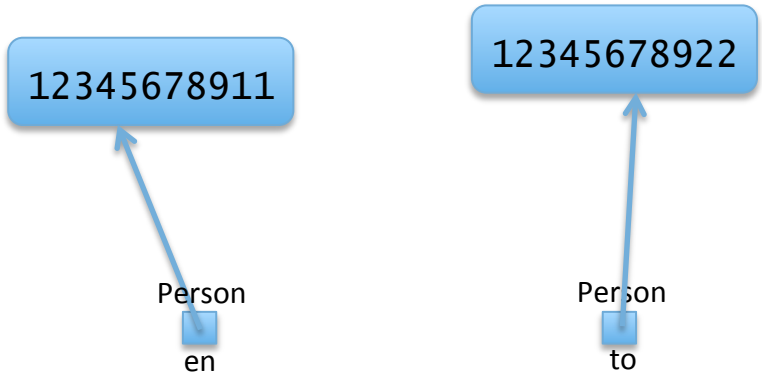



```
> java Sammenlign
123456789inf1010
en <= to
true
false
```

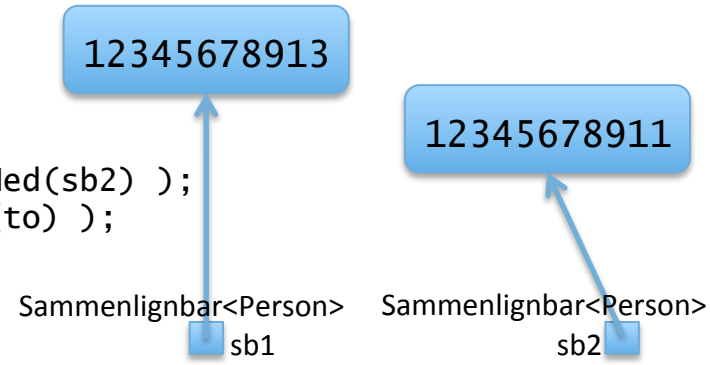
(1 > -1)

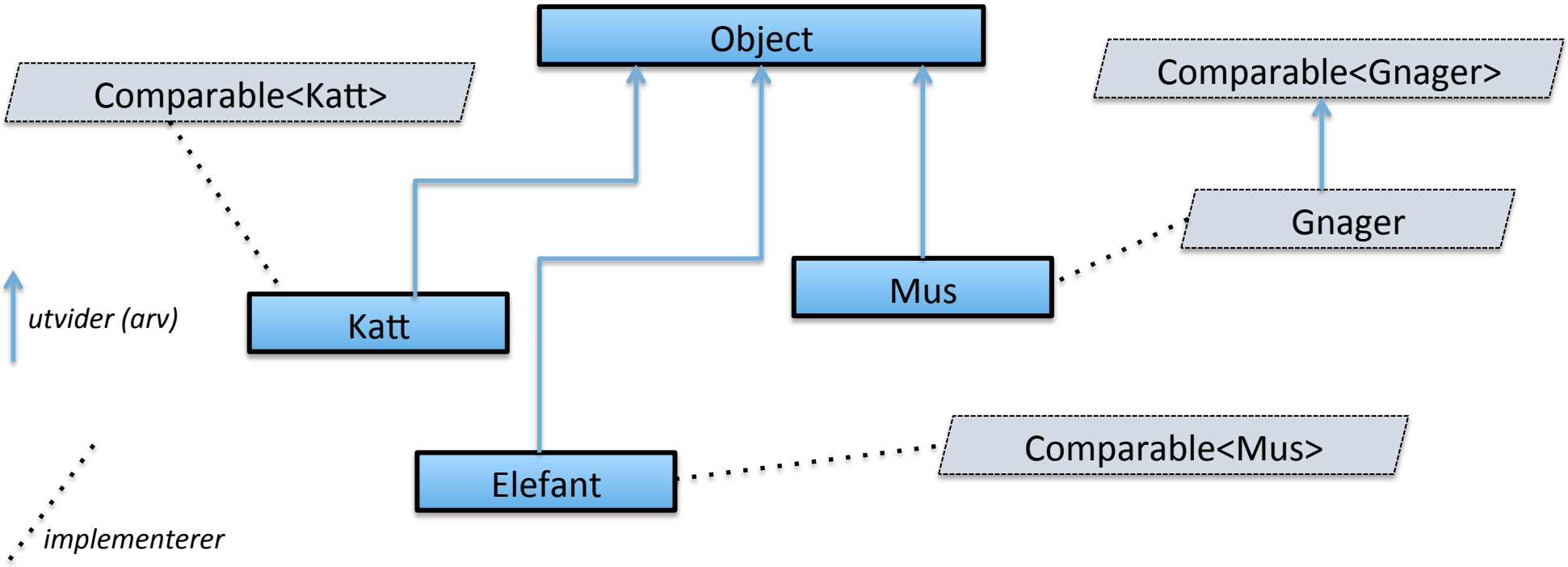


```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("en <= to");

     // System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```

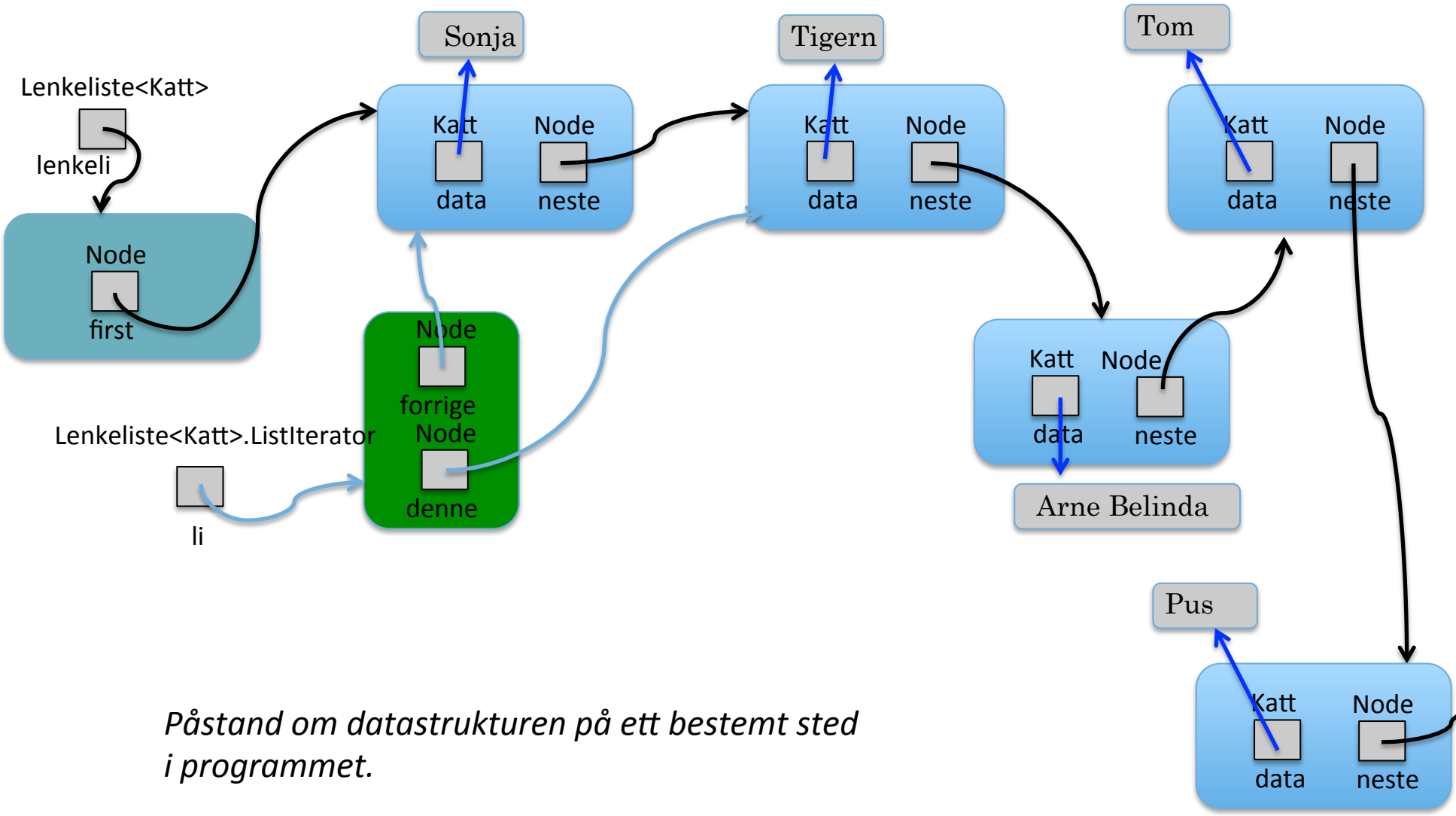



```
> java Sammenlign
123456789inf1010
en <= to
true
false
```





Et klassehierarki sier noe om strukturen i programmet (koden), og ingenting om datastrukturen under kjøring.



Påstand om datastrukturen på ett bestemt sted i programmet.

I dag

Begrensende typeparametre

Tilstandspåstander

Iteratorer

LIFO/FIFO

```
interface Gnager {
    public boolean lever();
    public int alder();
}
class Rotte implements Gnager { }
class Mus implements Gnager { }
class HusMus extends Mus { }

class Bol<T extends Gnager> {  }

Bol<Gnager> bol = new Bol<Gnager>();
```

I dette bolet kan vi også plassere husmus !

```
interface Gnager {
    public boolean lever();
    public int alder();
}
class Rotte implements Gnager { }
class Mus implements Gnager { }
class HusMus extends Mus { }

class Bol<T extends Gnager> {  }

Bol<HusMus> bol = new Bol<HusMus>();
```

I dette bolet kan vi **bare** *plassere husmus !*

public final class String extends Object implements Serializable, Comparable<String>, CharSequence { …. }

class Lenkeliste <T extends Comparable<T> > { }

```
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();
lenkel04.settInnForan(new String("INF1010"));

Comparable<String> cs = "Et objekt som kan sammenlignes med et String-objekt";
lenkel04.settInnForan( ((String) cs) );
```

```
interface Gnager extends Comparable<Gnager> {
   public boolean lever();
   public int alder();
}
```

```
class Mus implements Gnager {
   int alder;

   public boolean lever() {return alder() > 37;}
   public int alder() {return alder;}
   public int compareTo(Gnager g) { return 1; }
   public int compareTo(Mus m) { return 7; }

}
```

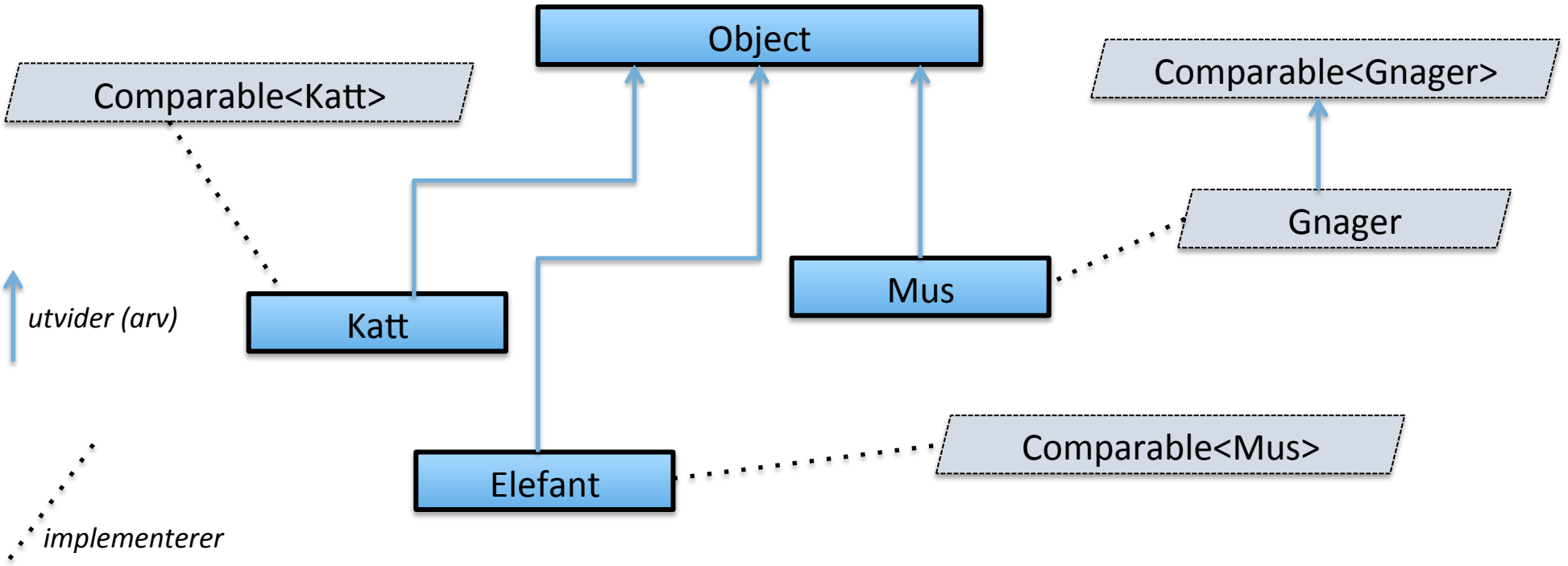
class Lenkeliste <T extends Comparable<T> > { }

```
Lenkeliste<Mus> lenkel03 = new Lenkeliste<Mus>();
lenkel03.settInnForan(new Mus());
```

```
Lenkeliste<Object> lenkel01 = new Lenkeliste<Object>();
Lenkeliste<Katt>   lenkel02 = new Lenkeliste<Katt>();
Lenkeliste<Gnager> lenkel03 = new Lenkeliste<Gnager>();
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();

Lenkeliste<Mus> lenkel05 = new Lenkeliste<Mus>();
Lenkeliste<Elefant> lenkel06 = new Lenkeliste<Elefant>();
```

class Lenkeliste <T extends Comparable<T> > { … }

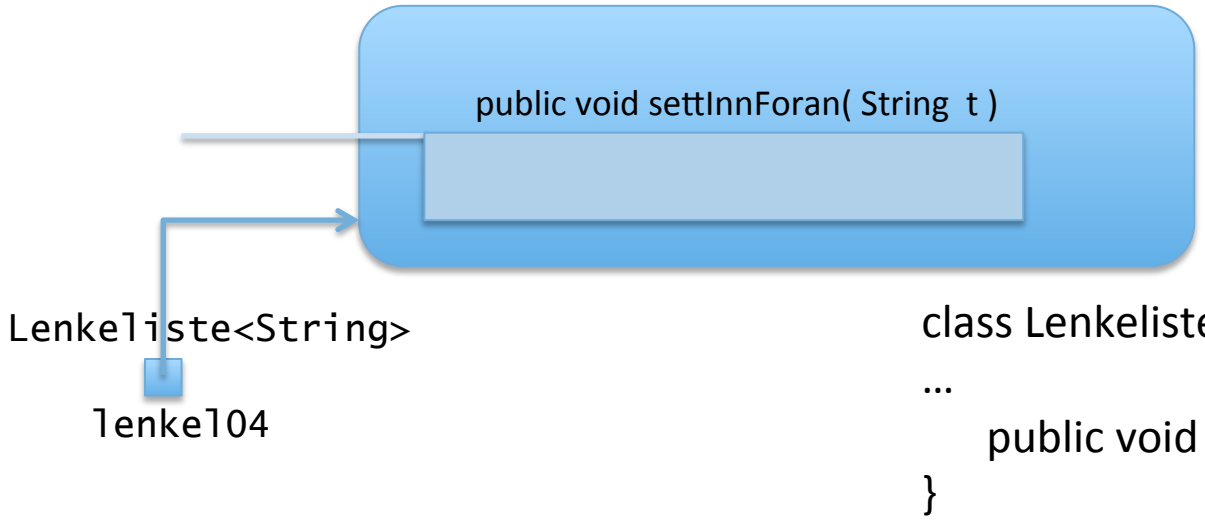
```
Lenkeliste<Object> lenkel01 = new Lenkeliste<Object>();
Lenkeliste<Katt>   lenkel02 = new Lenkeliste<Katt>();
Lenkeliste<Gnager> lenkel03 = new Lenkeliste<Gnager>();
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();

Lenkeliste<Mus> lenkel05 = new Lenkeliste<Mus>();
Lenkeliste<Elefant> lenkel06 = new Lenkeliste<Elefant>();
```



```
Lenkeliste<Object> lenkel01 = new Lenkeliste<Object>();
Lenkeliste<Katt>   lenkel02 = new Lenkeliste<Katt>();
Lenkeliste<Gnager> lenkel03 = new Lenkeliste<Gnager>();
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();

Comparable<String> cs = ”Objekt som kan sammenlignes med et String-objekt";
lenkel04.settInnForan( cs );

Lenkeliste<Mus> lenkel05 = new Lenkeliste<Mus>();
Lenkeliste<Elefant> lenkel06 = new Lenkeliste<Elefant>();
```

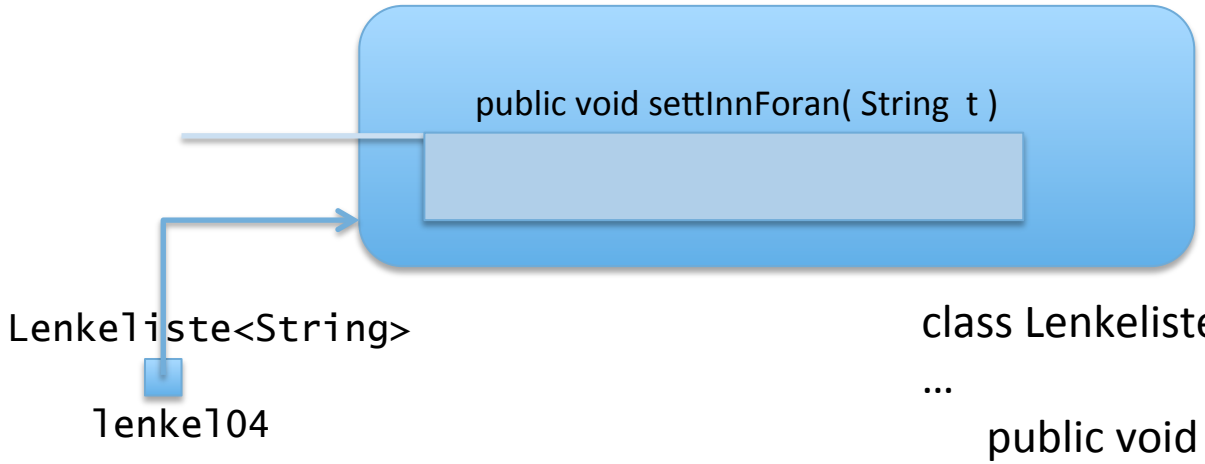
```
class Lenkeliste< T extends Comparable<T> > {
…
    public void settInnForan( T  t ) { ...}
}
```

```
Lenkeliste<Object> lenkel01 = new Lenkeliste<Object>();
Lenkeliste<Katt>   lenkel02 = new Lenkeliste<Katt>();
Lenkeliste<Gnager> lenkel03 = new Lenkeliste<Gnager>();
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();

Comparable<String> cs = ”Objekt som kan sammenlignes med et String-objekt";
lenkel04.settInnForan( cs );

Lenkeliste<Mus> lenkel05 = new Lenkeliste<Mus>();
Lenkeliste<Elefant> lenkel06 = new Lenkeliste<Elefant>();
```



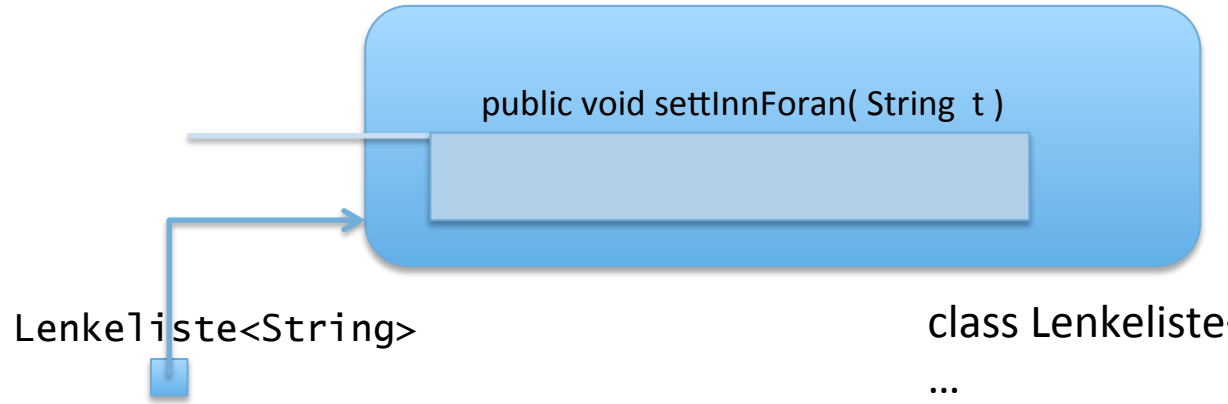

```
class Lenkeliste< T extends Comparable<T> > {
...
    public void settInnForan( T t ) { ...}
}
```

```
Lenkeliste<Object> lenkel01 = new Lenkeliste<Object>();
Lenkeliste<Katt>   lenkel02 = new Lenkeliste<Katt>();
Lenkeliste<Gnager> lenkel03 = new Lenkeliste<Gnager>();
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();

Comparable<String> cs = ”Objekt som kan sammenlignes med et String-objekt";
lenkel04.settInnForan( cs );

Lenkeliste<Mus> lenkel05 = new Lenkeliste<Mus>();
Lenkeliste<Elefant> lenkel06 = new Lenkeliste<Elefant>();
```




```
class Lenkeliste< T extends Comparable<T> > {
...
    public void settInnForan( T t ) { ...}
}
```

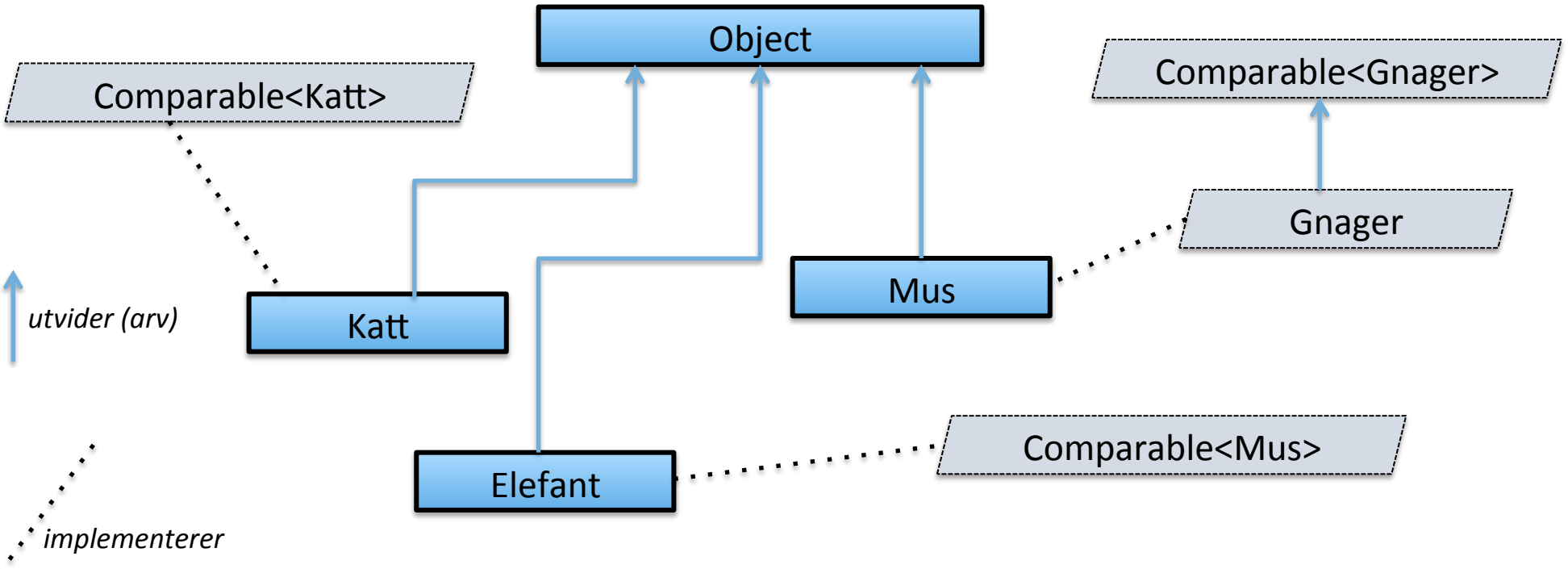
```
Lenkeliste<Object> lenkel01 = new Lenkeliste<Object>();
Lenkeliste<Katt>   lenkel02 = new Lenkeliste<Katt>();
Lenkeliste<Gnager> lenkel03 = new Lenkeliste<Gnager>();
Lenkeliste<String> lenkel04 = new Lenkeliste<String>();

Comparable<String> cs = ”Objekt som kan sammenlignes med et String-objekt";
lenkel04.settInnForan( ((String) cs) );

Lenkeliste<Mus> lenkel05 = new Lenkeliste<Mus>();
Lenkeliste<Elefant> lenkel06 = new Lenkeliste<Elefant>();
```

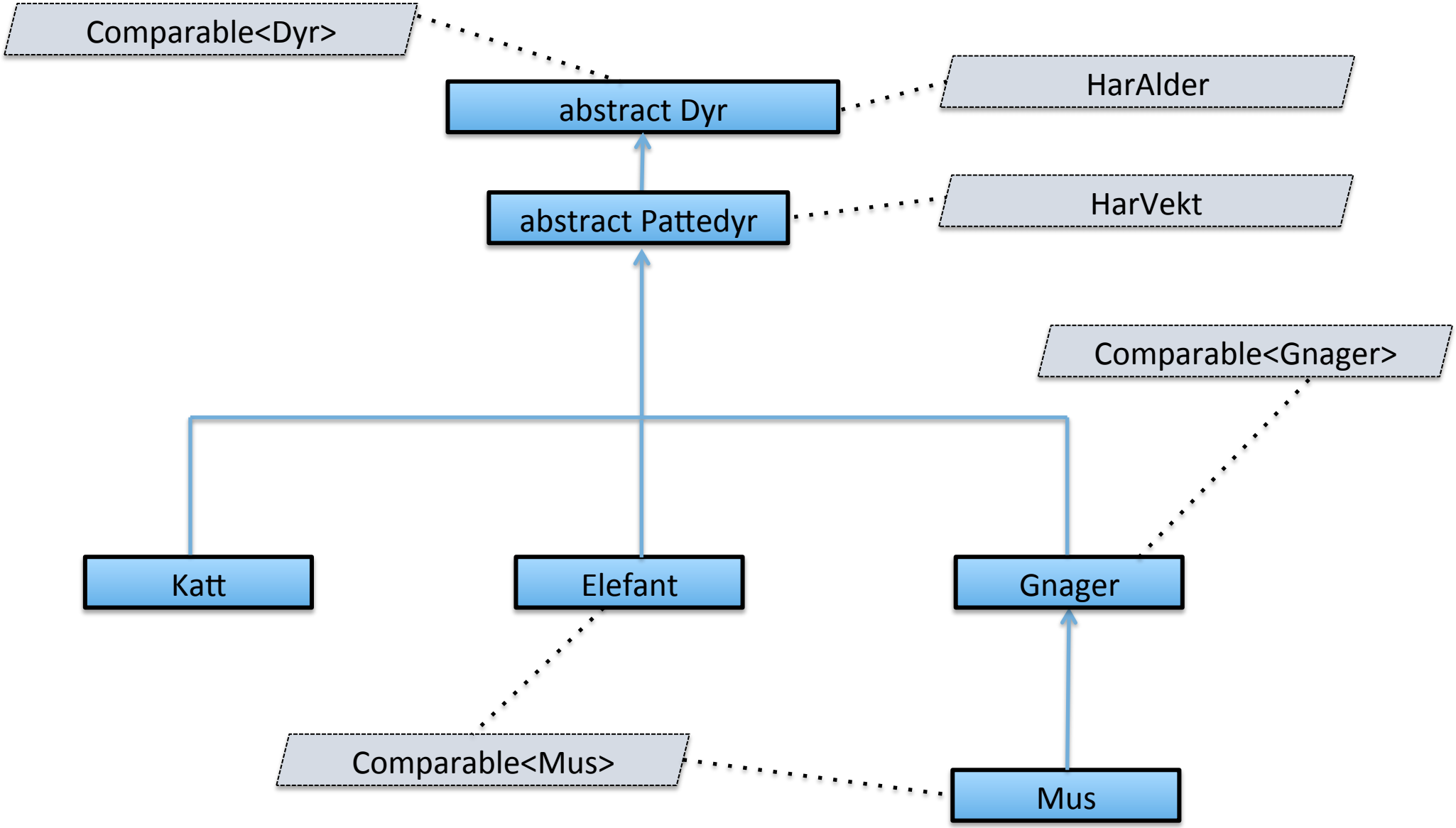



```
class Lenkeliste< T extends Comparable<T> > {
...
    public void settInnForan( T t ) { ...}
}
```

```
class Katt implements Comparable<Katt> {
   String navn;
   Katt (String n) { navn = n; }

   public int compareTo(Katt k) { return 1; }
}

interface Gnager extends Comparable<Gnager> {
   public boolean lever();
   public int alder();
}
```

```
class Mus implements Gnager {
   int alder;

   public boolean lever() {return alder() > 37;}
   public int alder() {return alder;}
   public int compareTo(Gnager g) { return 1; }
}

class Elefant implements Comparable<Mus> {
   public int compareTo(Mus m) { return -1; }
}
```




Comparable<Dyr>
abstract Dyr
HarAlder
abstract Pattedyr
HarVekt
Comparable<Gnager>
Katt
Elefant
Gnager
Comparable<Mus>
Mus

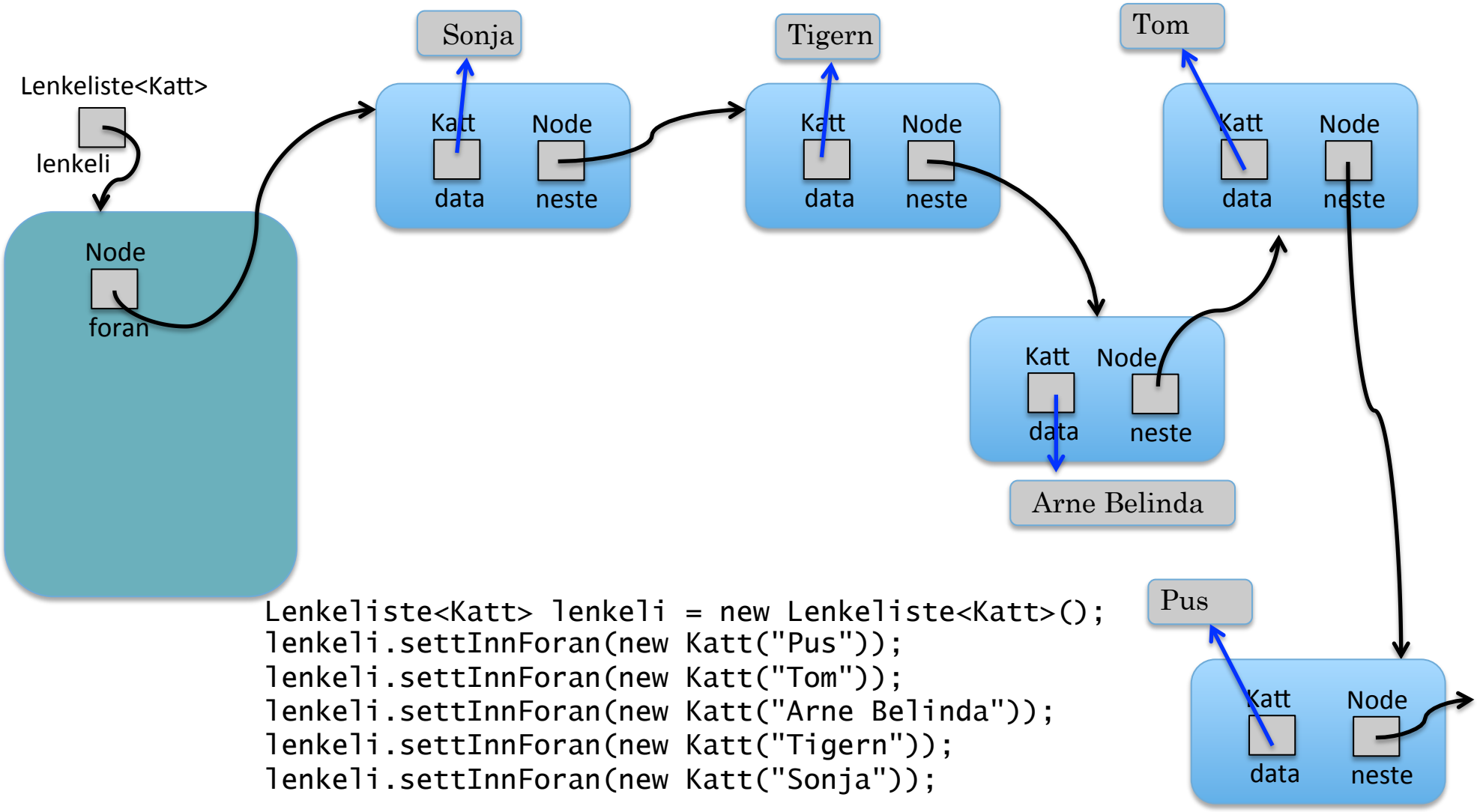
I dag

Begrensende typeparametre

Tilstandspåstander

Iteratorer

LIFO/FIFO




```
Lenkeliste<Katt> lenkeli = new Lenkeliste<Katt>();
lenkeli.settInnForan(new Katt("Pus"));
lenkeli.settInnForan(new Katt("Tom"));
lenkeli.settInnForan(new Katt("Arne Belinda"));
lenkeli.settInnForan(new Katt("Tigern"));
lenkeli.settInnForan(new Katt("Sonja"));
```

I dag

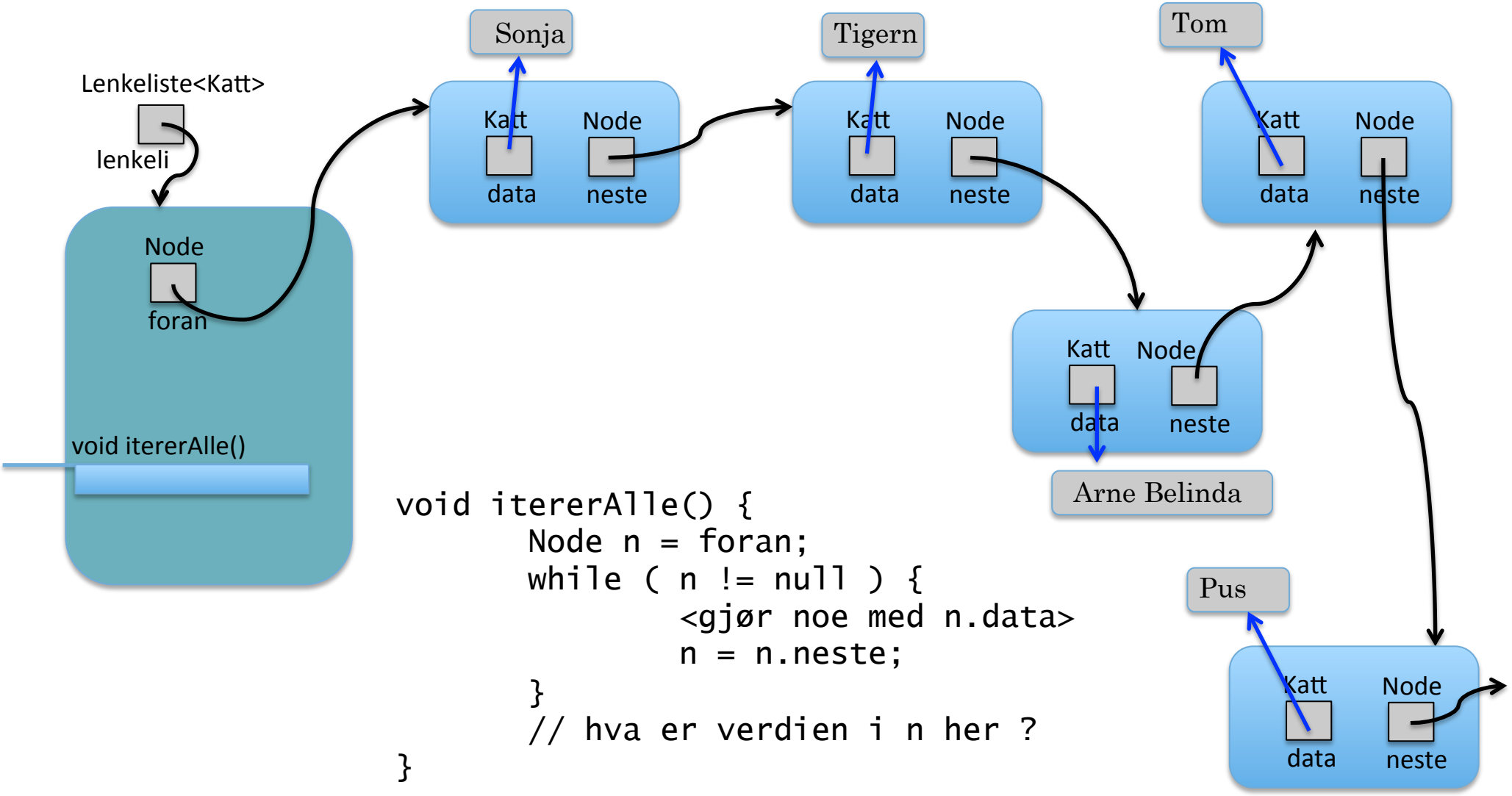
Begrensende typeparametre

Tilstandspåstander

Iteratorer

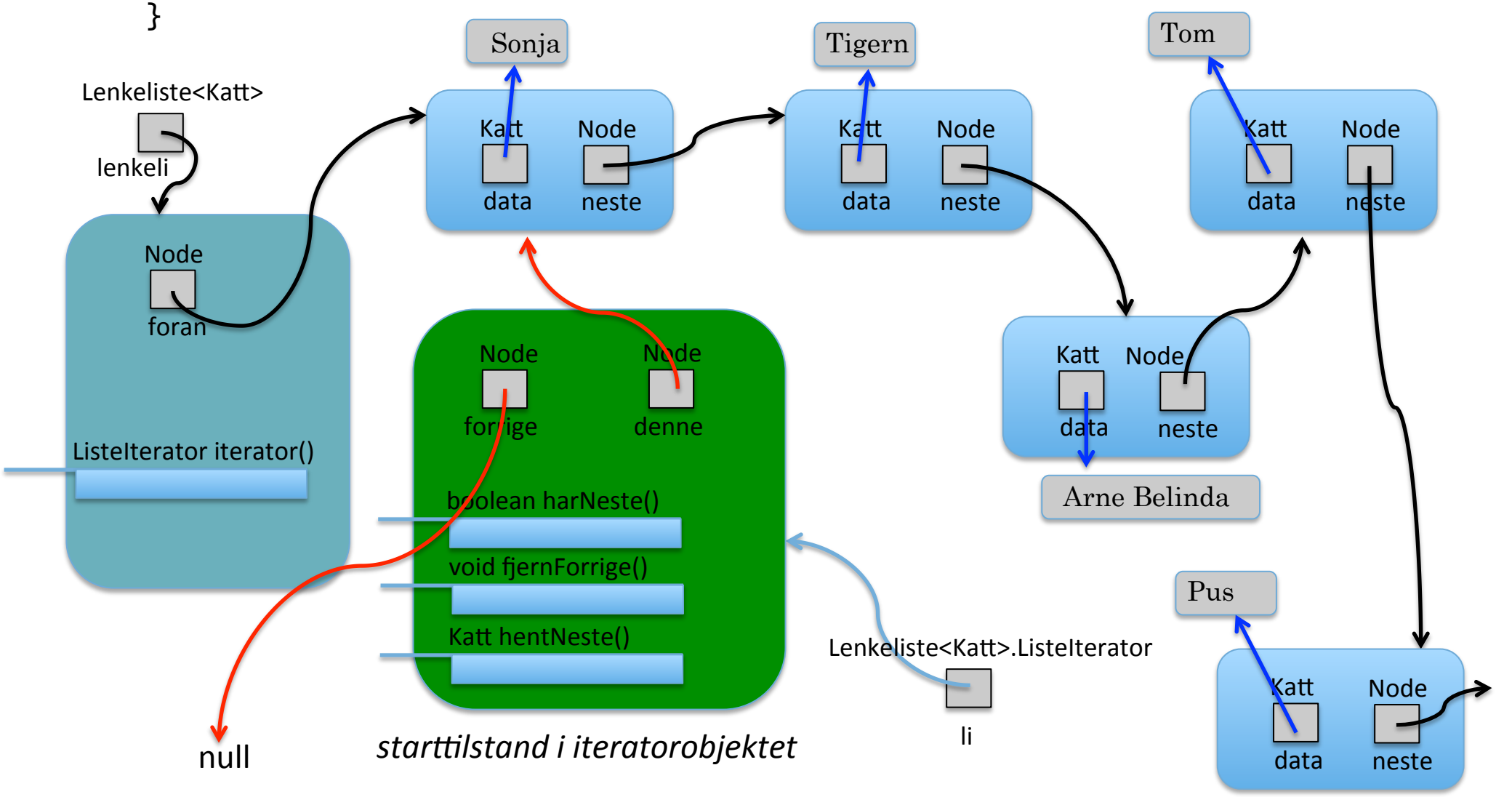
LIFO/FIFO

```
Lenkeliste<Katt> lenkeli = new Lenkeliste<Katt>();
lenkeli.settInnForan(new Katt("Pus"));
lenkeli.settInnForan(new Katt("Tom"));
lenkeli.settInnForan(new Katt("Arne Belinda"));
lenkeli.settInnForan(new Katt("Tigern"));
lenkeli.settInnForan(new Katt("Sonja"));
```



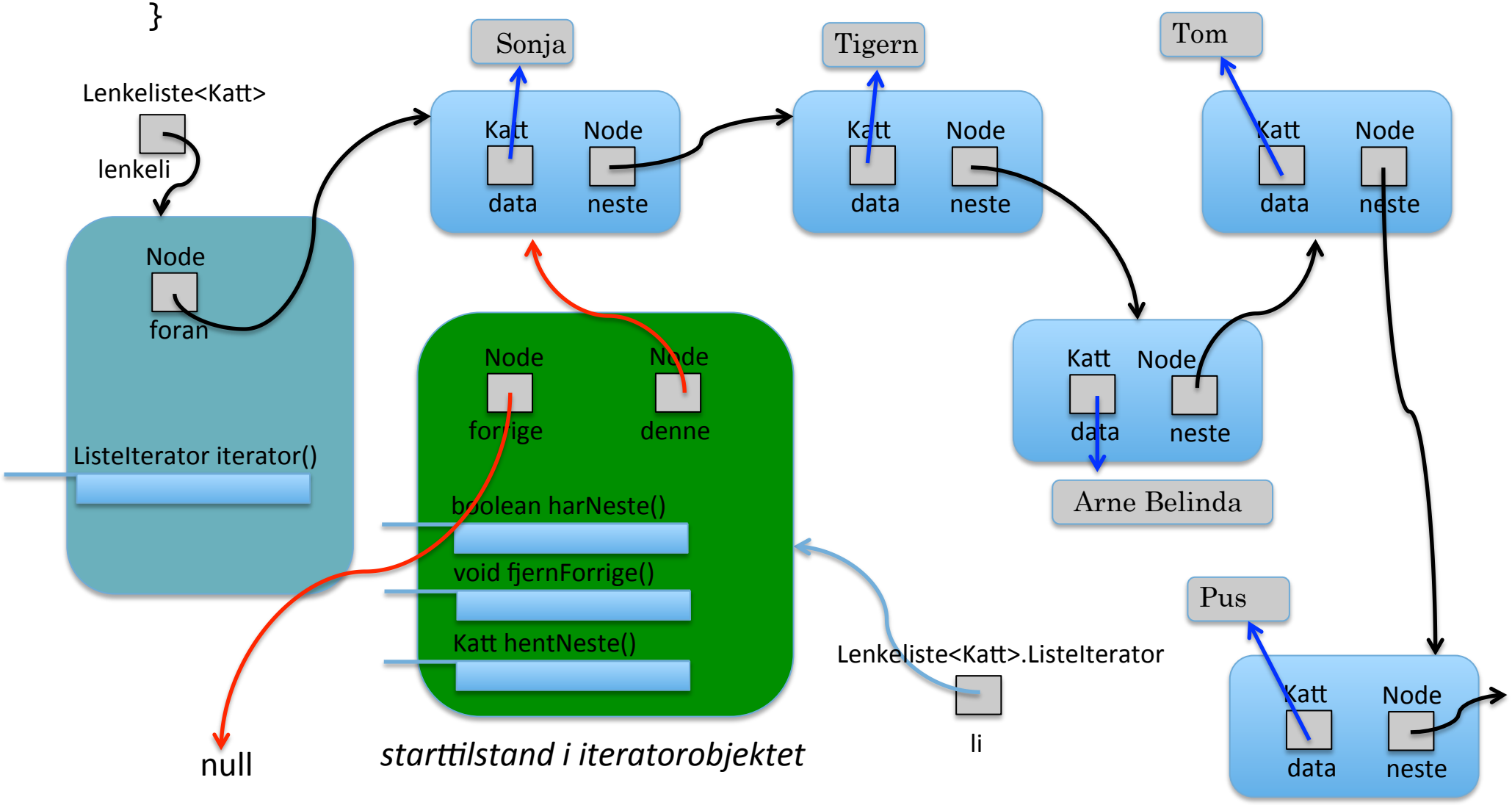

```
void itererAlle() {
        Node n = foran;
        while ( n != null ) {
                <gjør noe med n.data>
                n = n.neste;
        }
        // hva er verdien i n her ?
}
```

```
class ListeIterator {
    Node denne = foran;
    Node forrige = null;

    ...
}
```



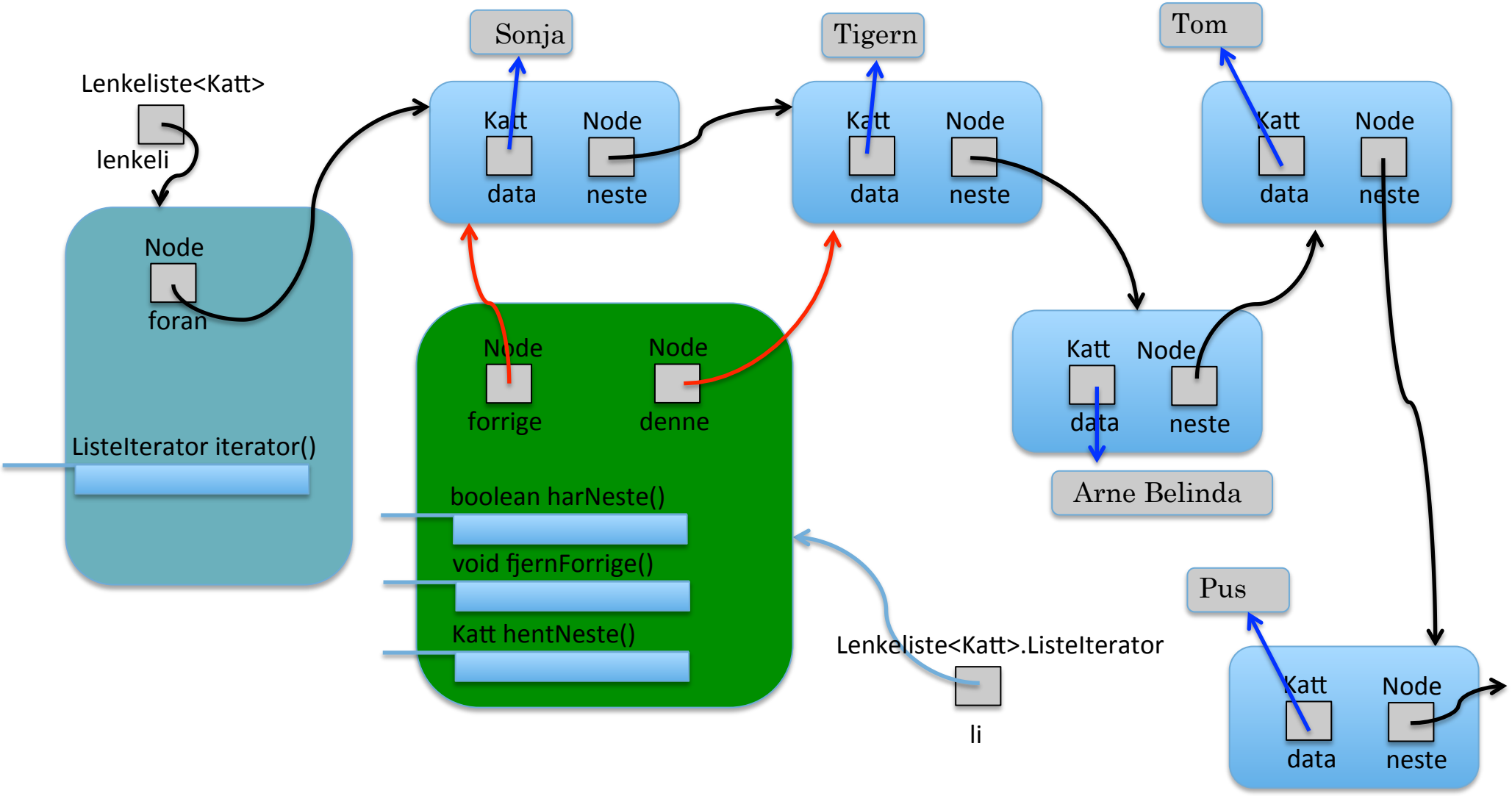
```
class ListeIterator {
    Node denne = foran;
    Node forrige = null;

    ...
}
```

Diskuter hva metodene skal gjøre

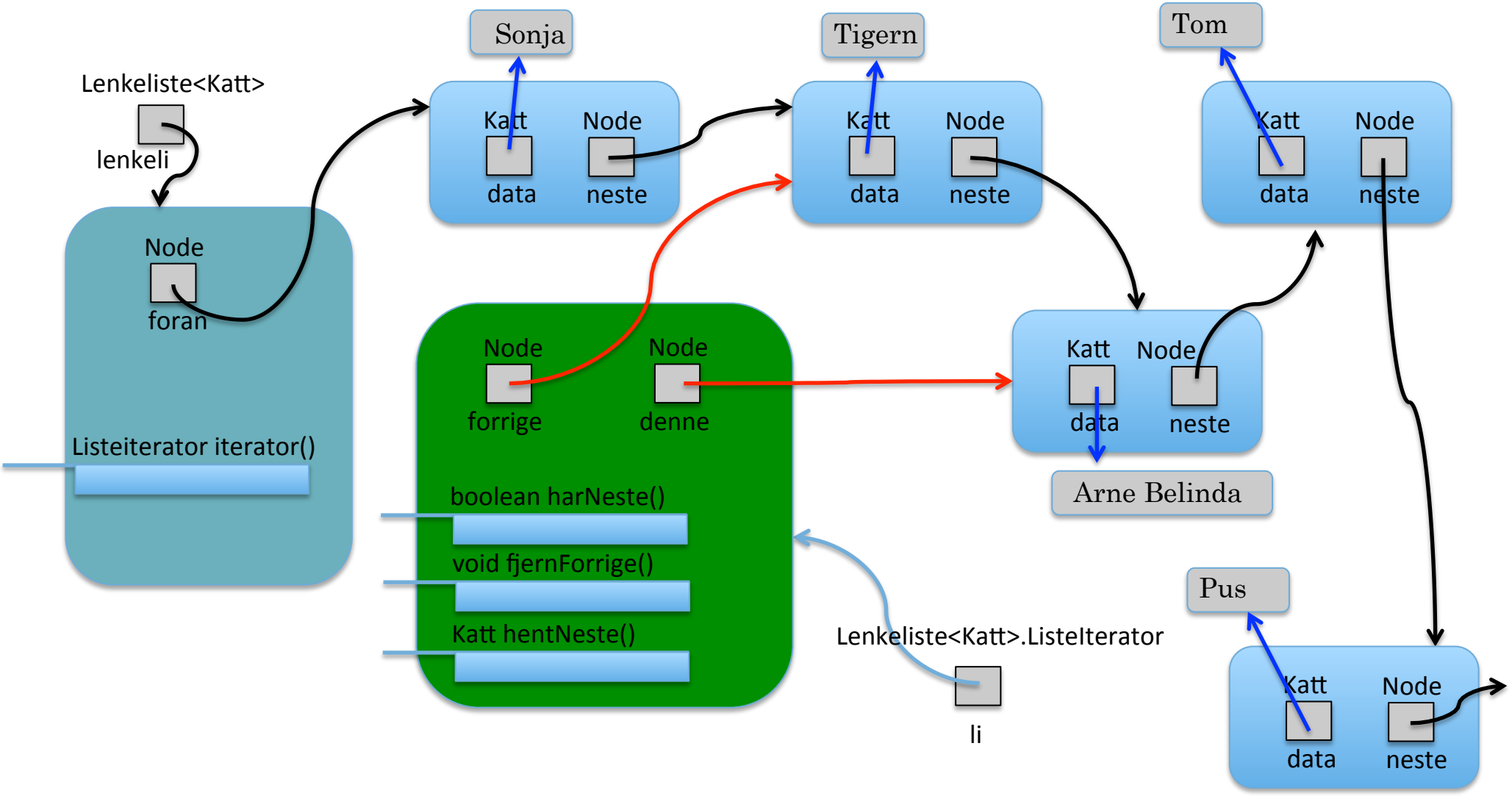


starttilstand i iteratorobjektet

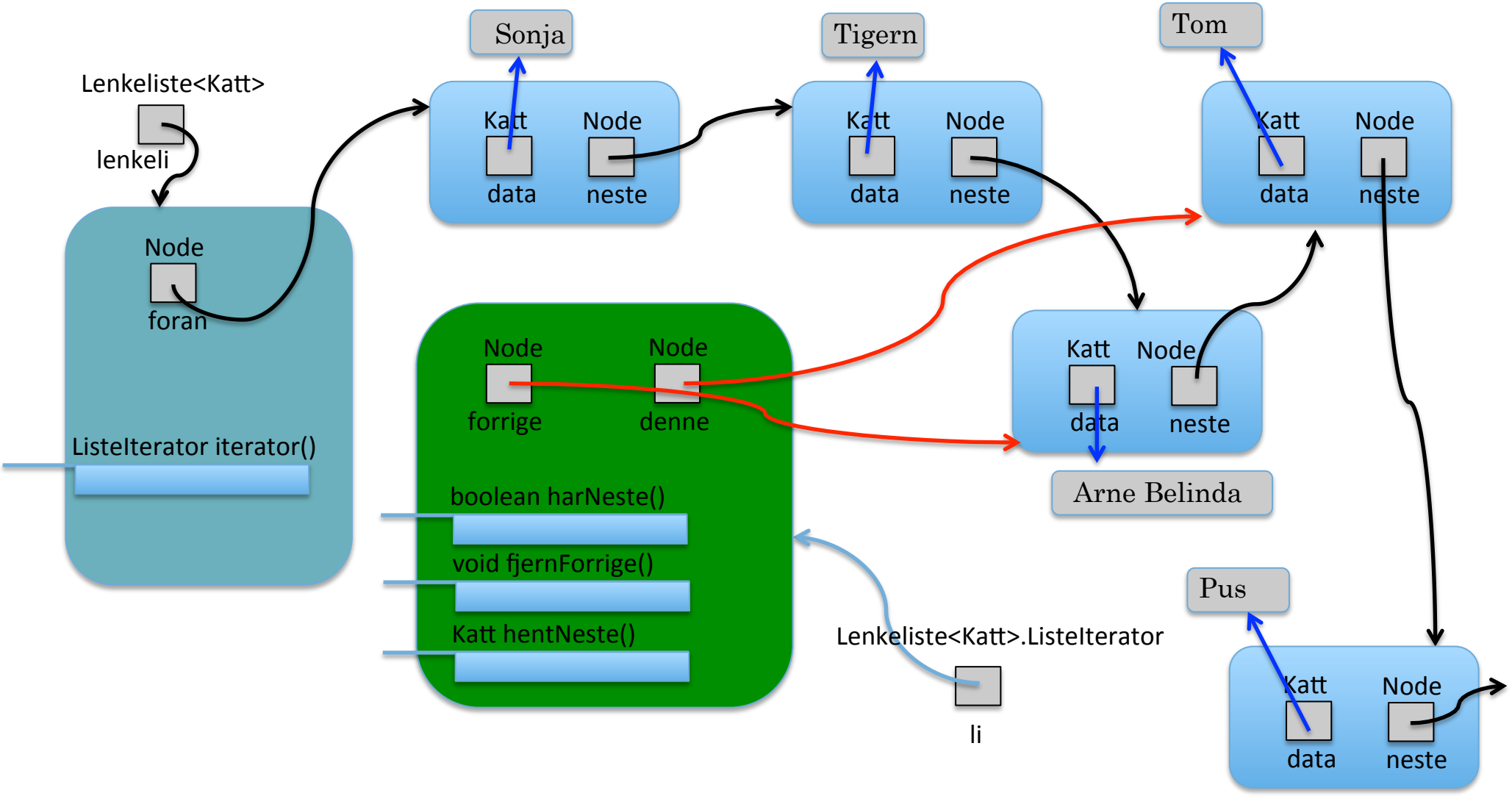
li.hentNeste();



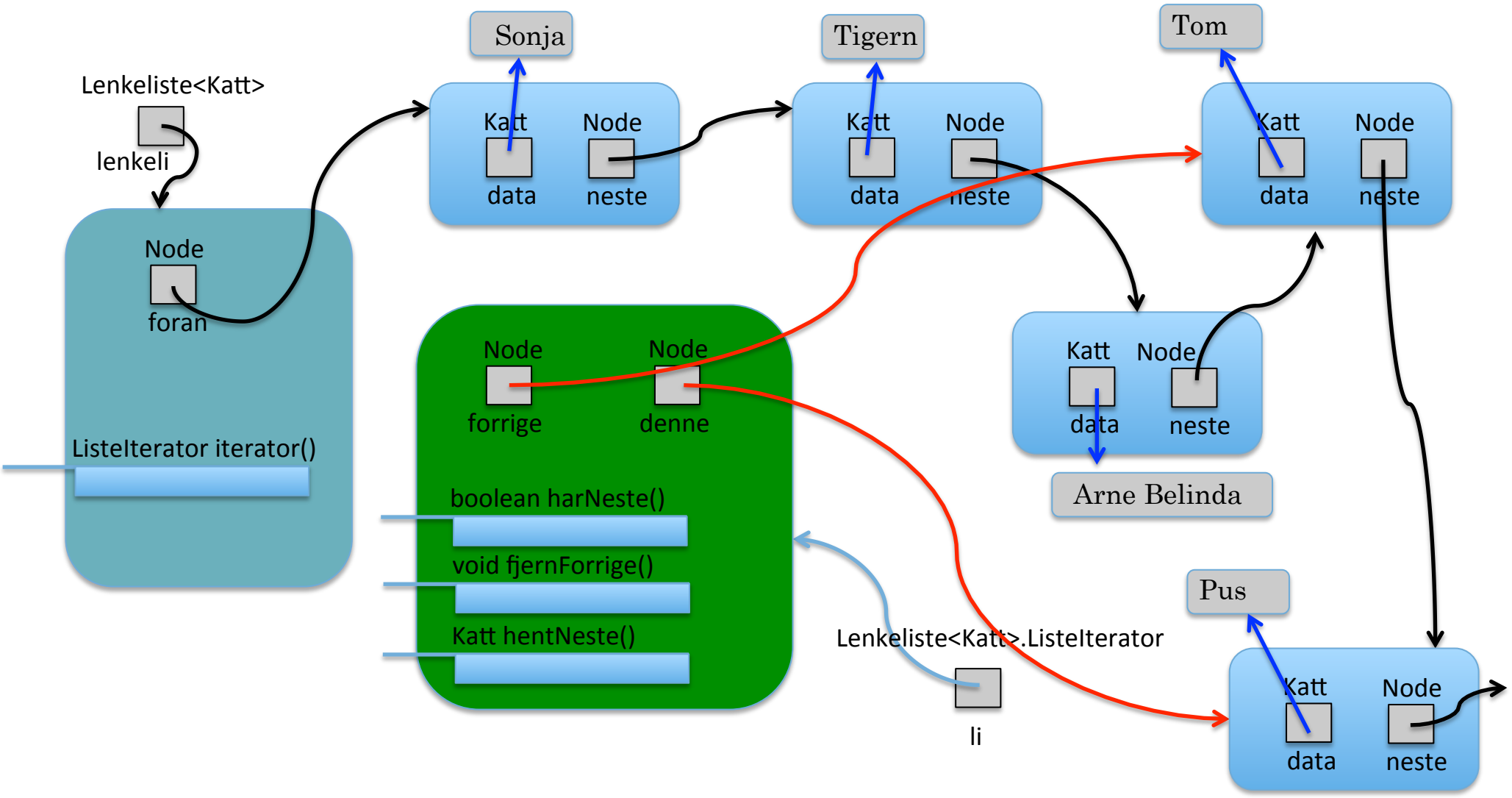
li.hentNeste();



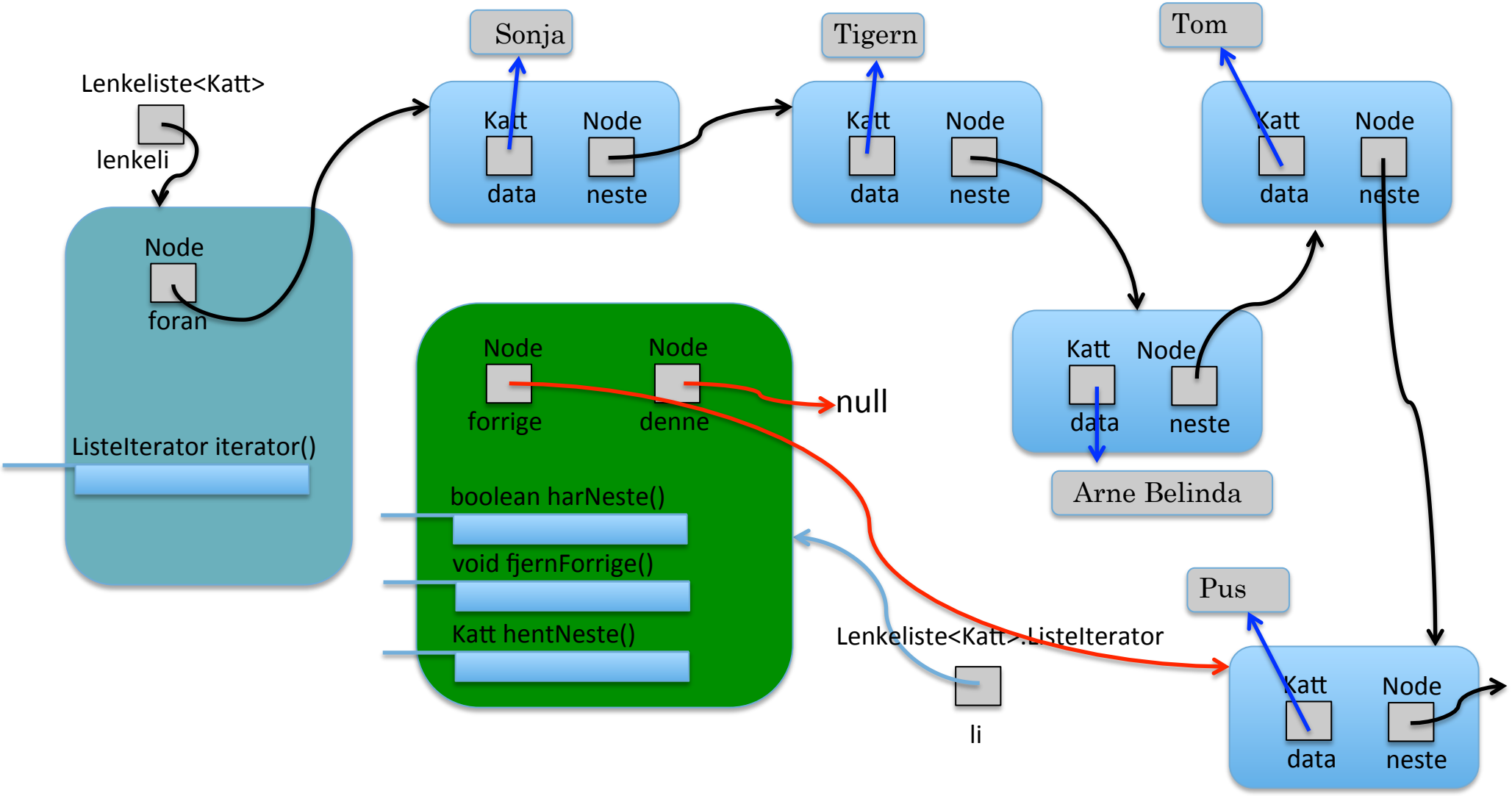
li.hentNeste();



li.hentNeste();

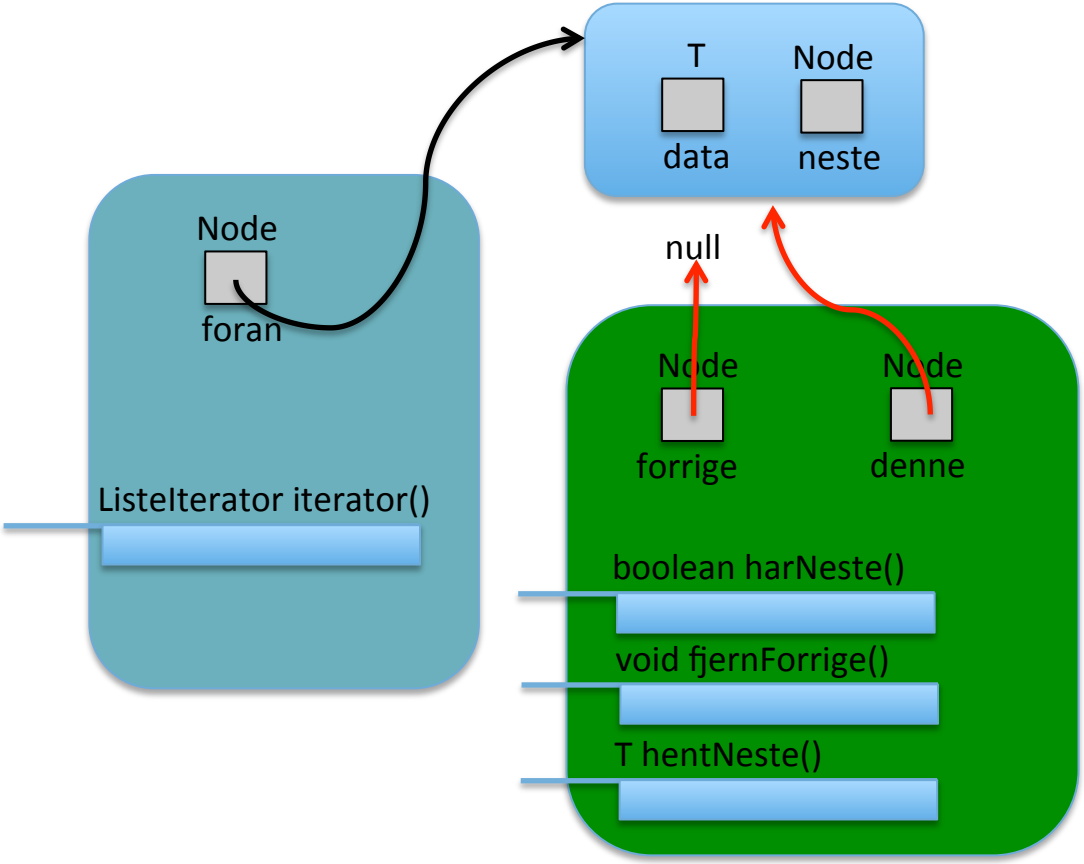


li.hentNeste();



Invariant tilstandspåstand:

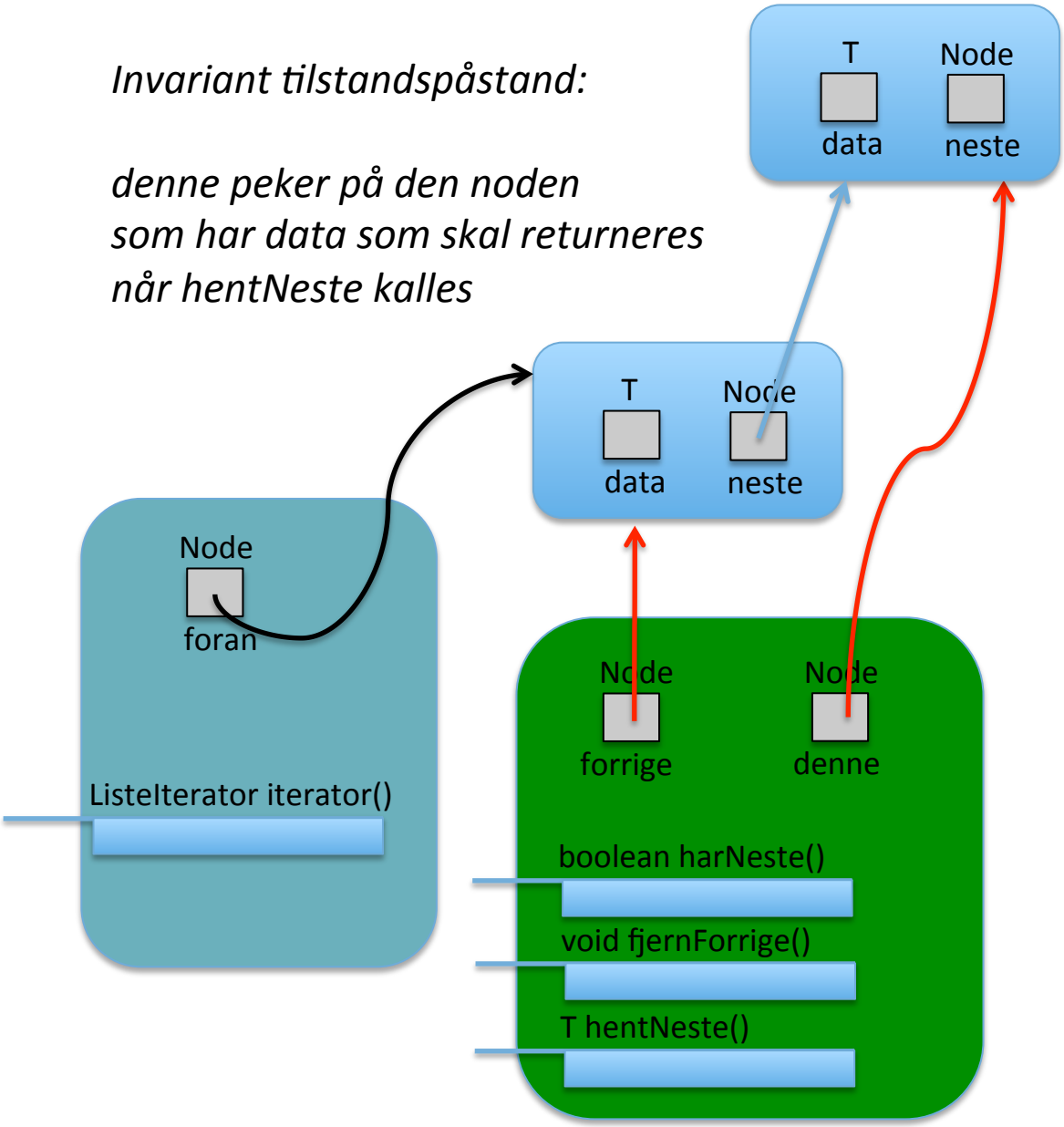
denne peker på den noden
som har data som skal returneres
når hentNeste kalles




```
class Lenkeliste <T> {
    private Node foran = null ;

    private class Node {
    T data ;
    Node neste = null ;

        Node (T s) {
            data = s;
        }
    }

    class ListeIterator {
        Node denne = foran;
        Node forrige = null;

        public boolean harNeste() {}
        public void fjernForrige() {}
        public T hentNeste() { }
    }

    public ListeIterator iterator() {
        return new ListeIterator ();
    }
}
```

Invariant tilstandspåstand:

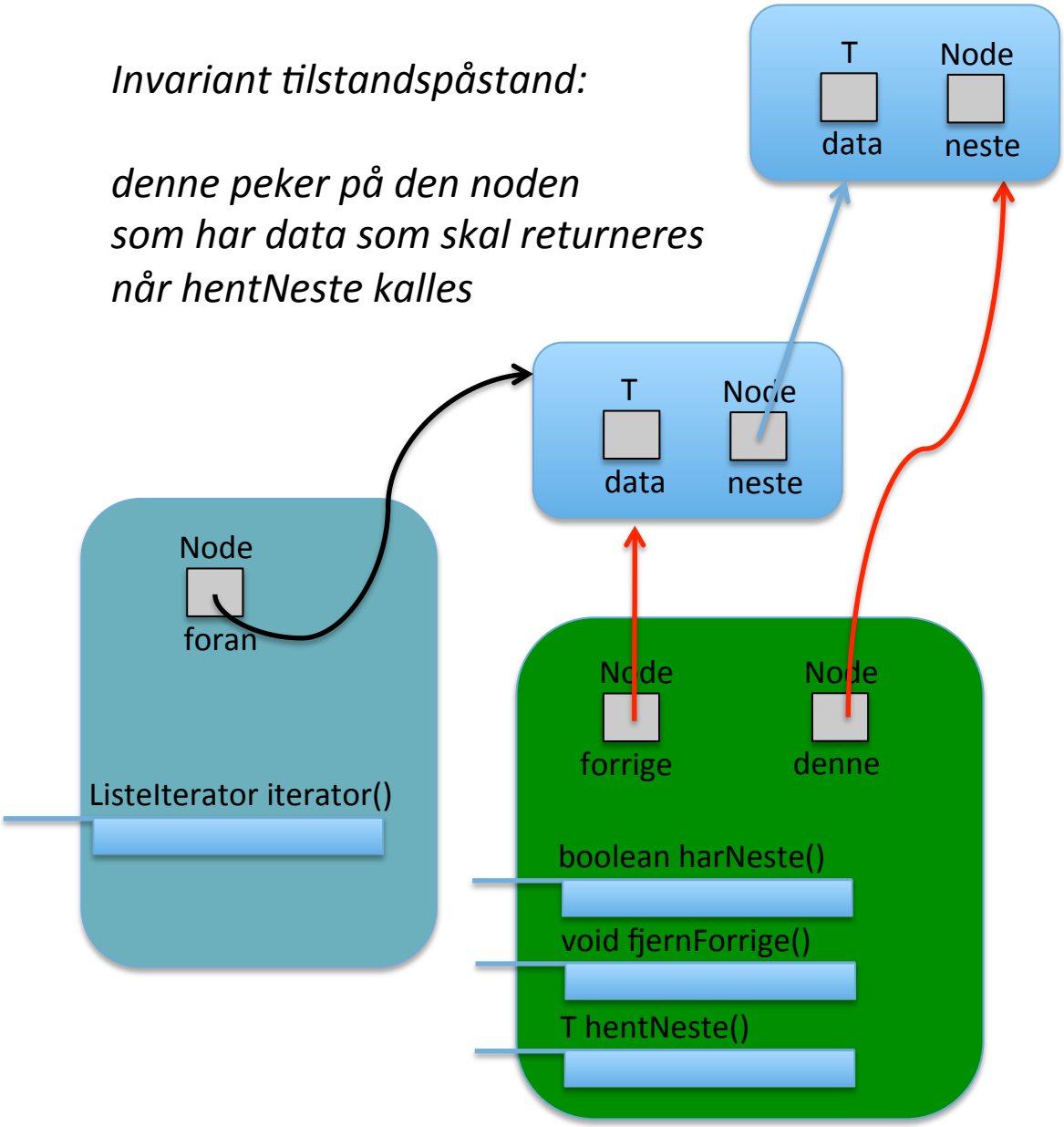
denne peker på den noden
som har data som skal returneres
når hentNeste kalles




```
class ListeIterator {
        Node denne = foran;
        Node forrige = null;

        public boolean harNeste() {
              return denne != null;
        }

        public void fjernForrige() {}

        public T hentNeste() {
              T returnerDenne = denne.data;
              forrige = denne;
             denne = denne.neste;
        }
 }
```

Invariant tilstandspåstand:

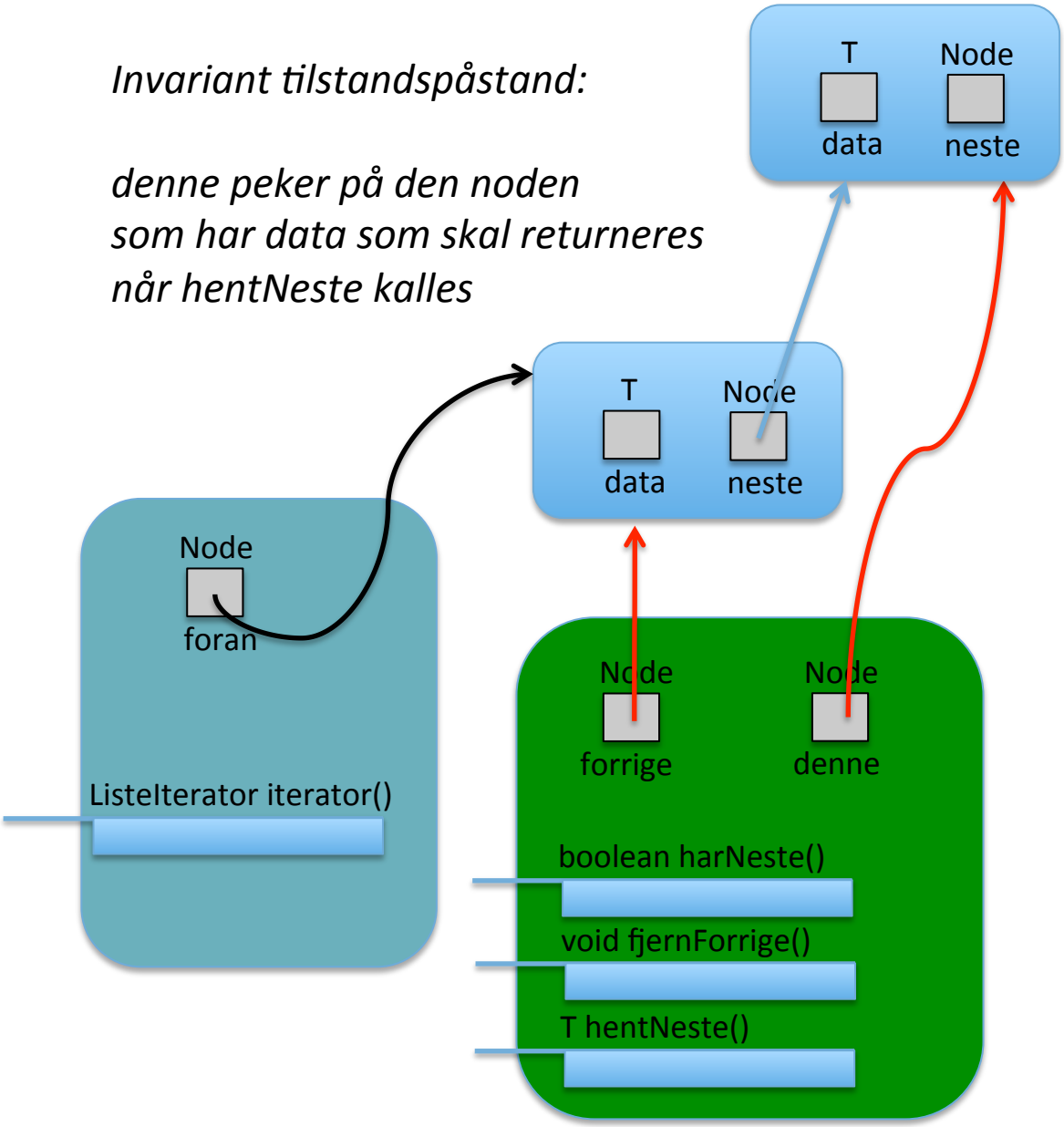
denne peker på den noden
som har data som skal returneres
når hentNeste kalles




```
class ListeIterator {
        Node denne = foran;
        Node forrige = null;

        public boolean harNeste() {
              return denne != null;
        }

        public void fjernForrige() {}

        public T hentNeste() {
             T returnerDenne = denne.data;
              forrige = denne;
             denne = denne.neste;
             return returnerDenne;
        }
 }
```

Invariant tilstandspåstand:

denne peker på den noden
som har data som skal returneres
når hentNeste kalles




```
class ListeIterator {
        Node denne = foran;
        Node forrige = null;

        public boolean harNeste() {
                return denne != null;
        }

        public void fjernForrige() {}

        public T hentNeste() {
                forrige = denne;
                denne = denne.neste;
                return forrige.data;
        }
}
```

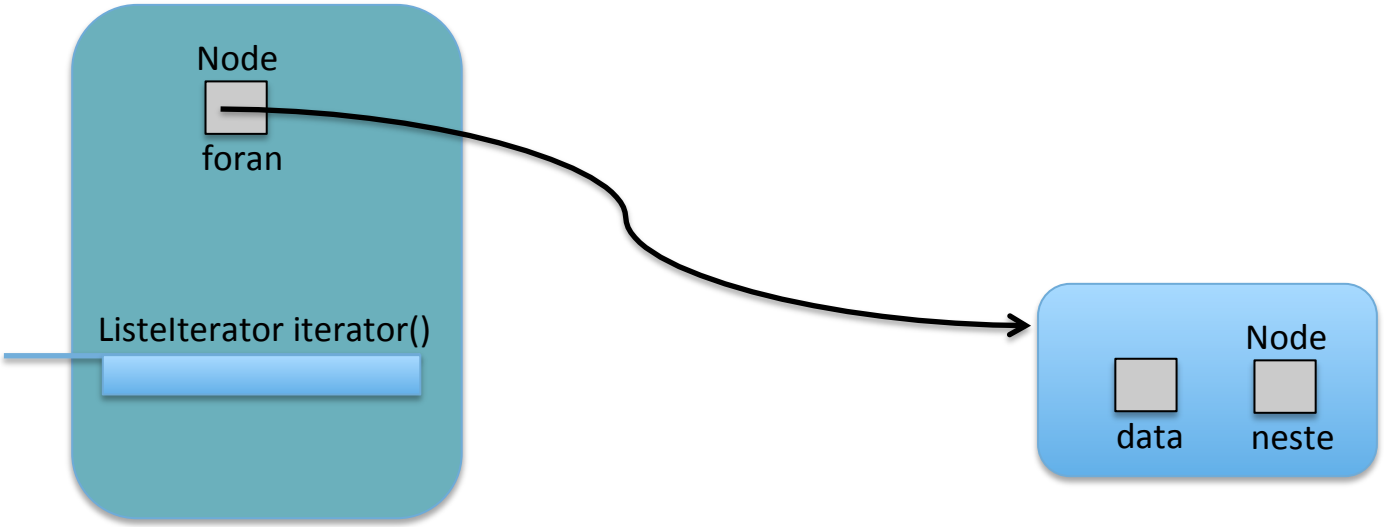
Regler for bruk av iteratoren

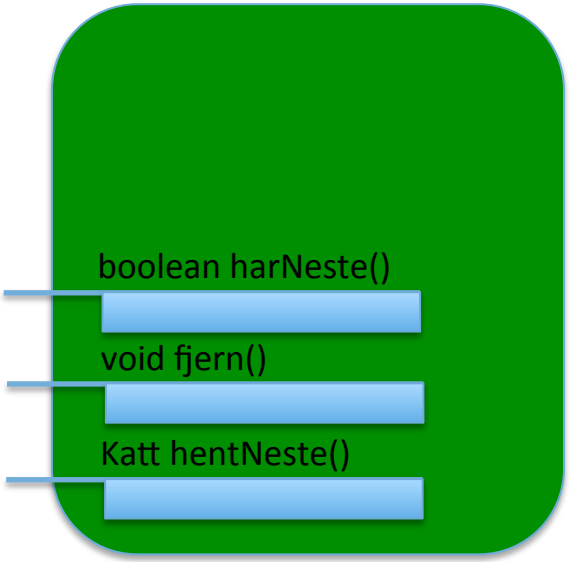
1. hentNeste() forutsetter at harNeste() er true
2. fjernForrige() kan ikke kalles før hentNeste()
3. fjernForrige() kan ikke kalles to ganger uten hentNeste() i mellom

2 og 3 håndteres ved at det skrives feilmelding ved brudd. Bedre å kaste unntak.

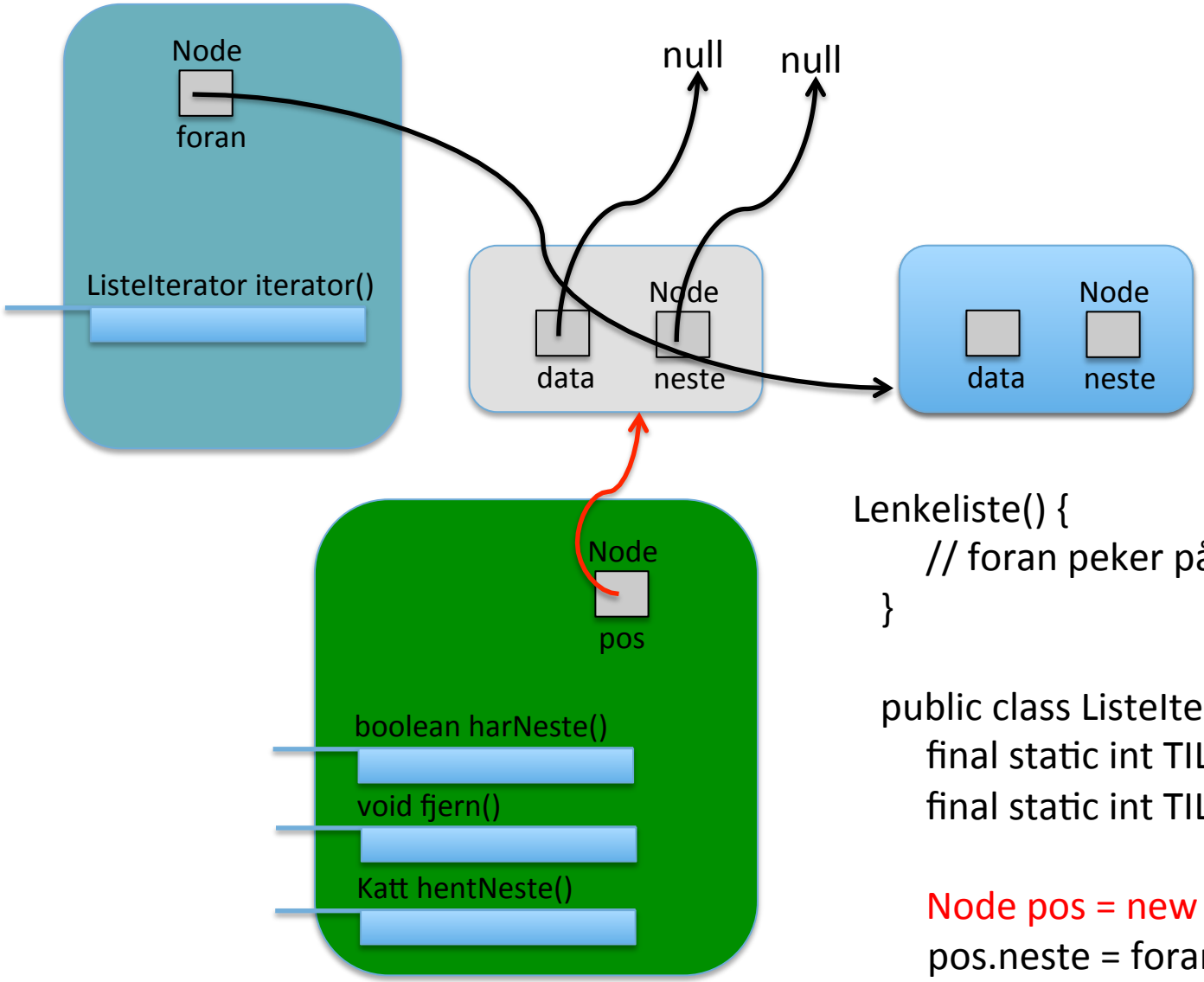
1 sjekkes ikke. Kunne vært lagt inn i hentNeste().

```
class ListeIterator {
   Node denne = foran;
   Node forrige = null;

   public boolean harNeste() {
                return denne != null;
         }

         public void fjernForrige() {}

         public T hentNeste() {
                if ( !harNeste() ) {
                      throw new NoSuchElementException();
                }
                forrige = denne;
                denne = denne.neste;
                return forrige.data;
         }
 }
```

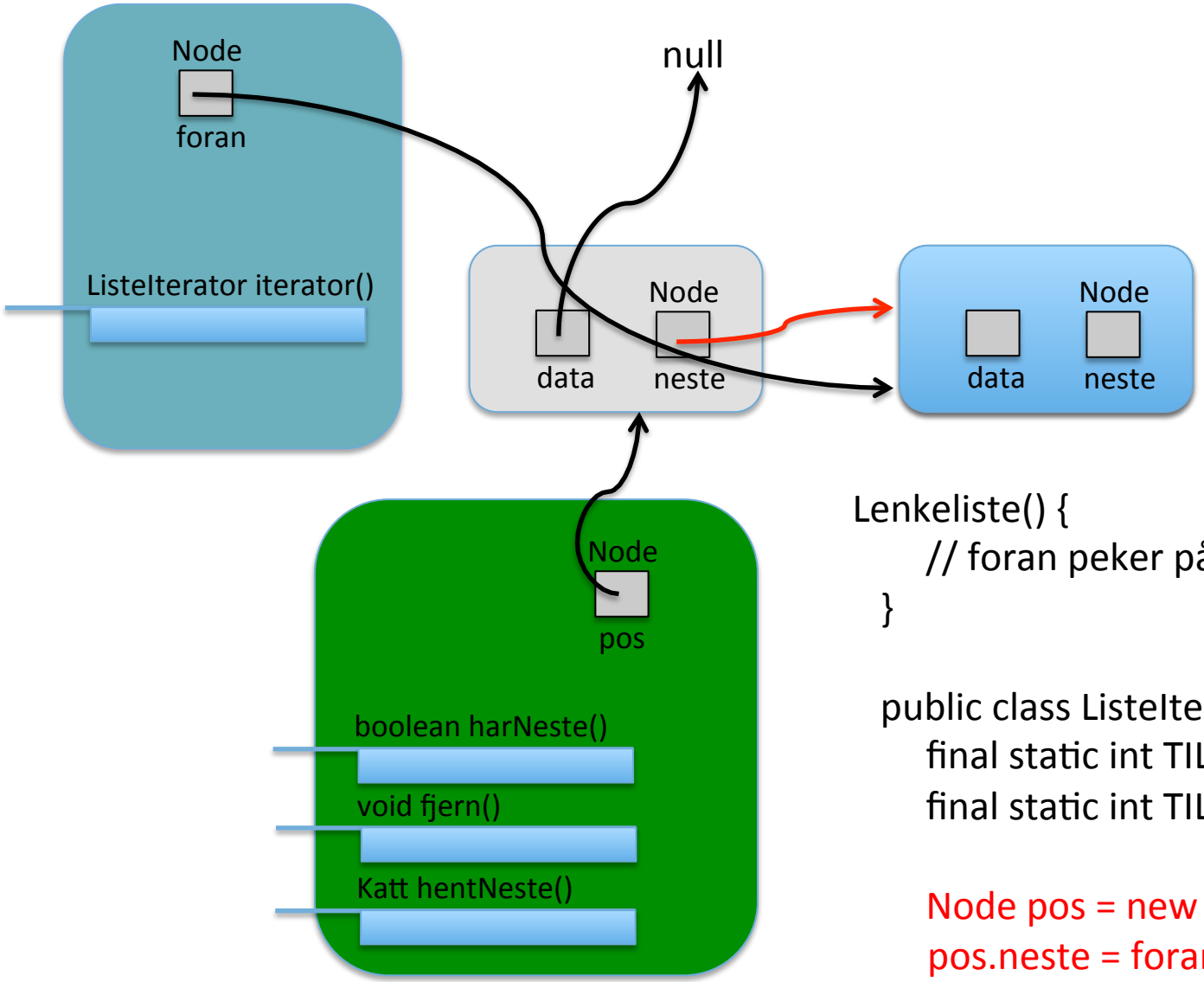





```
Lenkeliste() {
      // foran peker på første node
  }

  public class ListeIterator {
      final static int TILSTAND_A = 0;
      final static int TILSTAND_B = 1;

      Node pos = new Node(null); // listehode
      pos.neste = foran;
      int tilstand = TILSTAND_A;
      …
  }
```

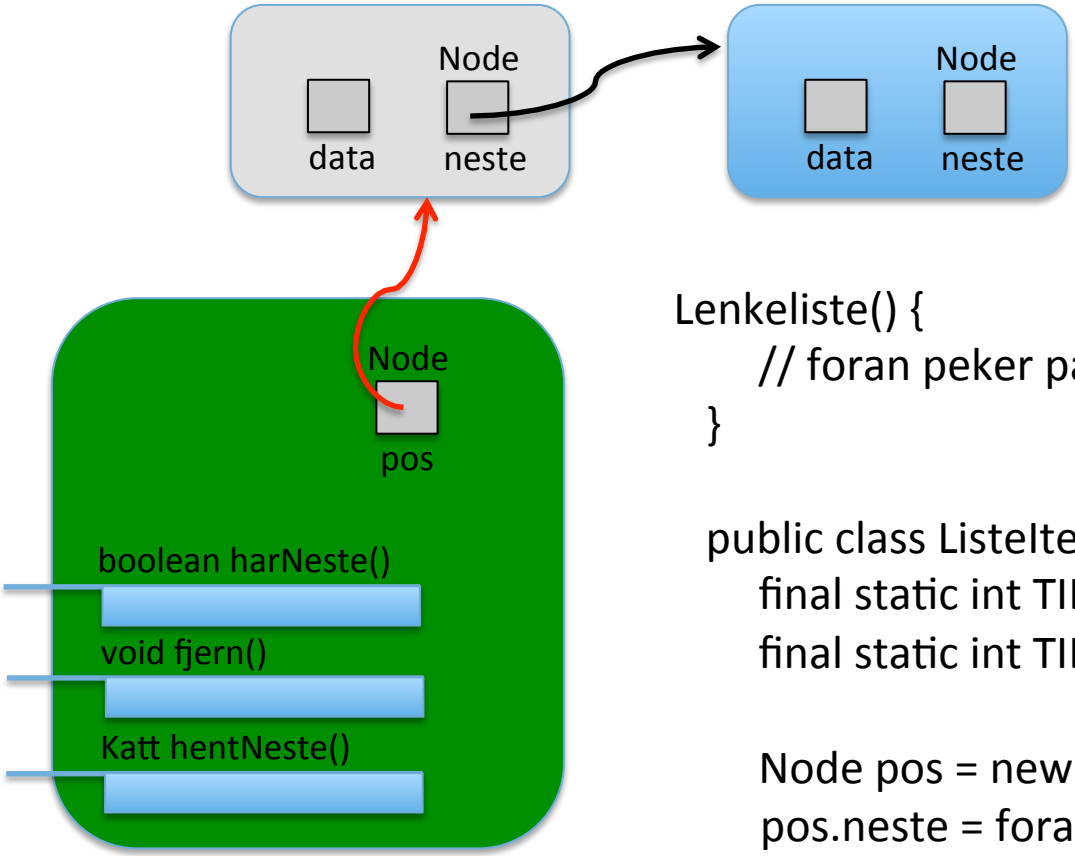



```
Lenkeliste() {
    // foran peker på første node
}

public class ListeIterator {
    final static int TILSTAND_A = 0;
    final static int TILSTAND_B = 1;

    Node pos = new Node(null); // listehode
    pos.neste = foran;
    int tilstand = TILSTAND_A;
    …
}
```




```
Lenkeliste() {
    // foran peker på første node
}

public class ListeIterator {
    final static int TILSTAND_A = 0;
    final static int TILSTAND_B = 1;

    Node pos = new Node(null); // listehode
    pos.neste = foran;
    int tilstand = TILSTAND_A;
    …
}
```

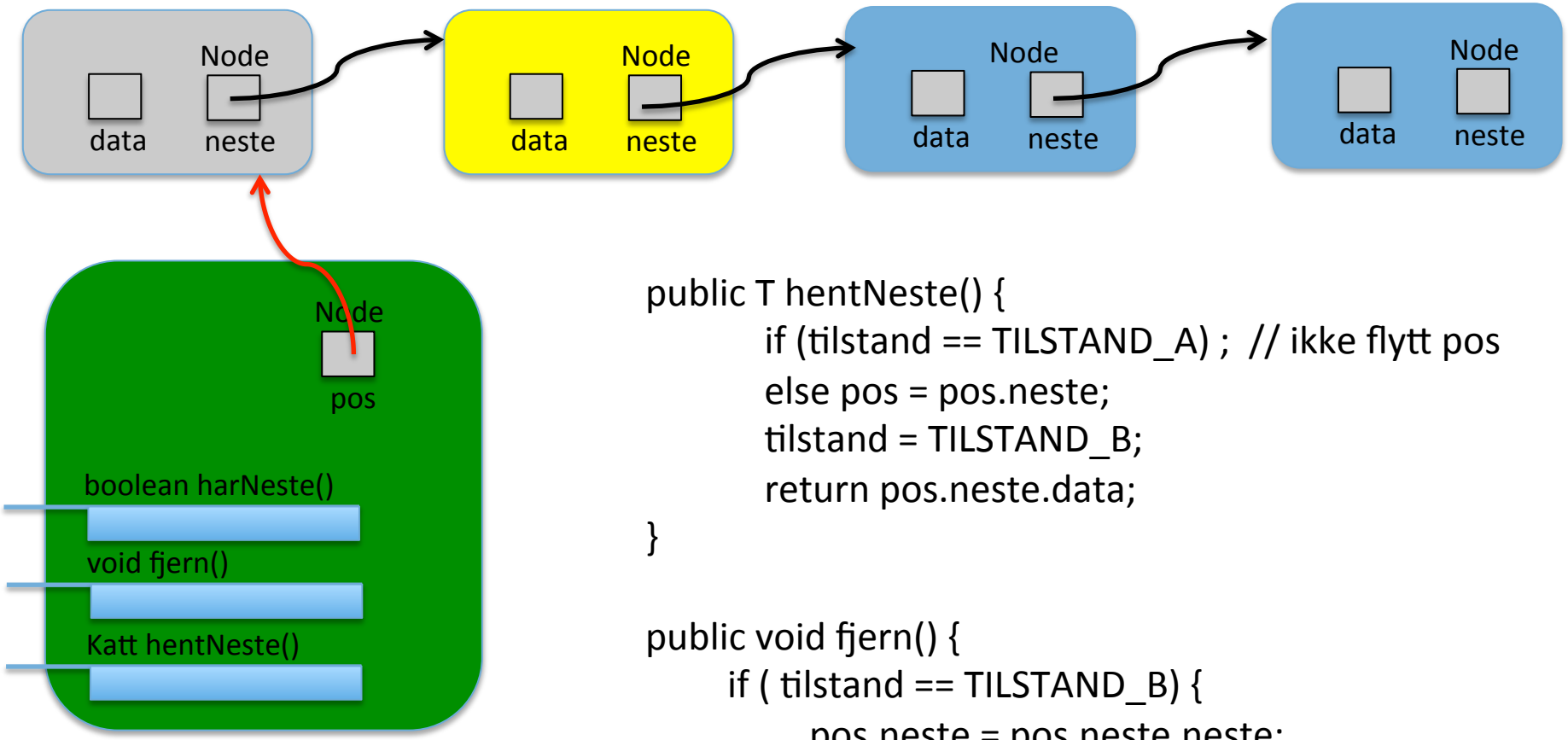
Tilstand A er initialtilstand




```
Lenkeliste() {
      // foran peker på første node
  }

  public class ListeIterator {
      final static int TILSTAND_A = 0;
      final static int TILSTAND_B = 1;

      Node pos = new Node(null); // listehode
      pos.neste = foran;
      Node pos = lh;
      int tilstand = TILSTAND_A;
      ...
  }
```

Tilstand A

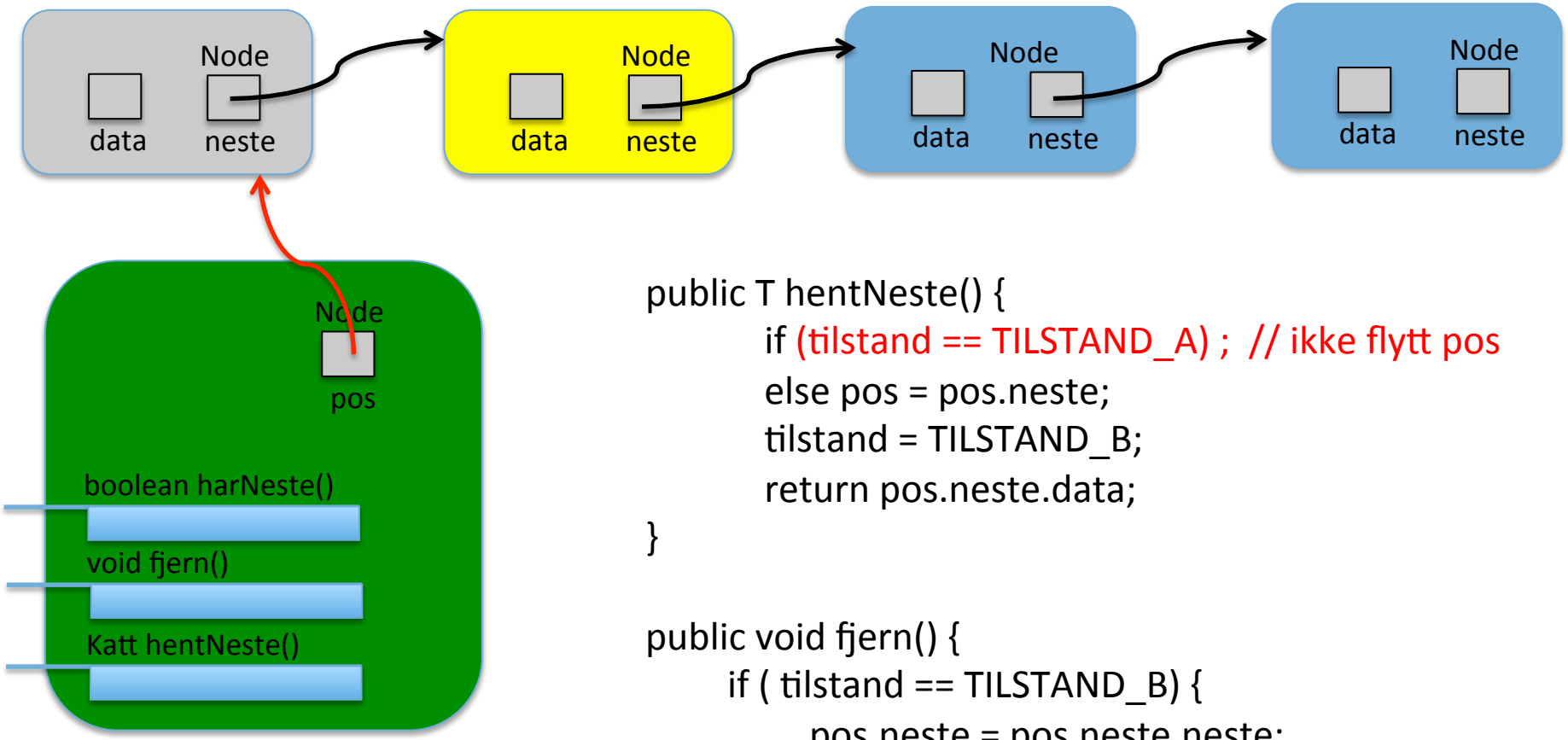
fjern() kan ikke kalles
pos.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

Tilstand A

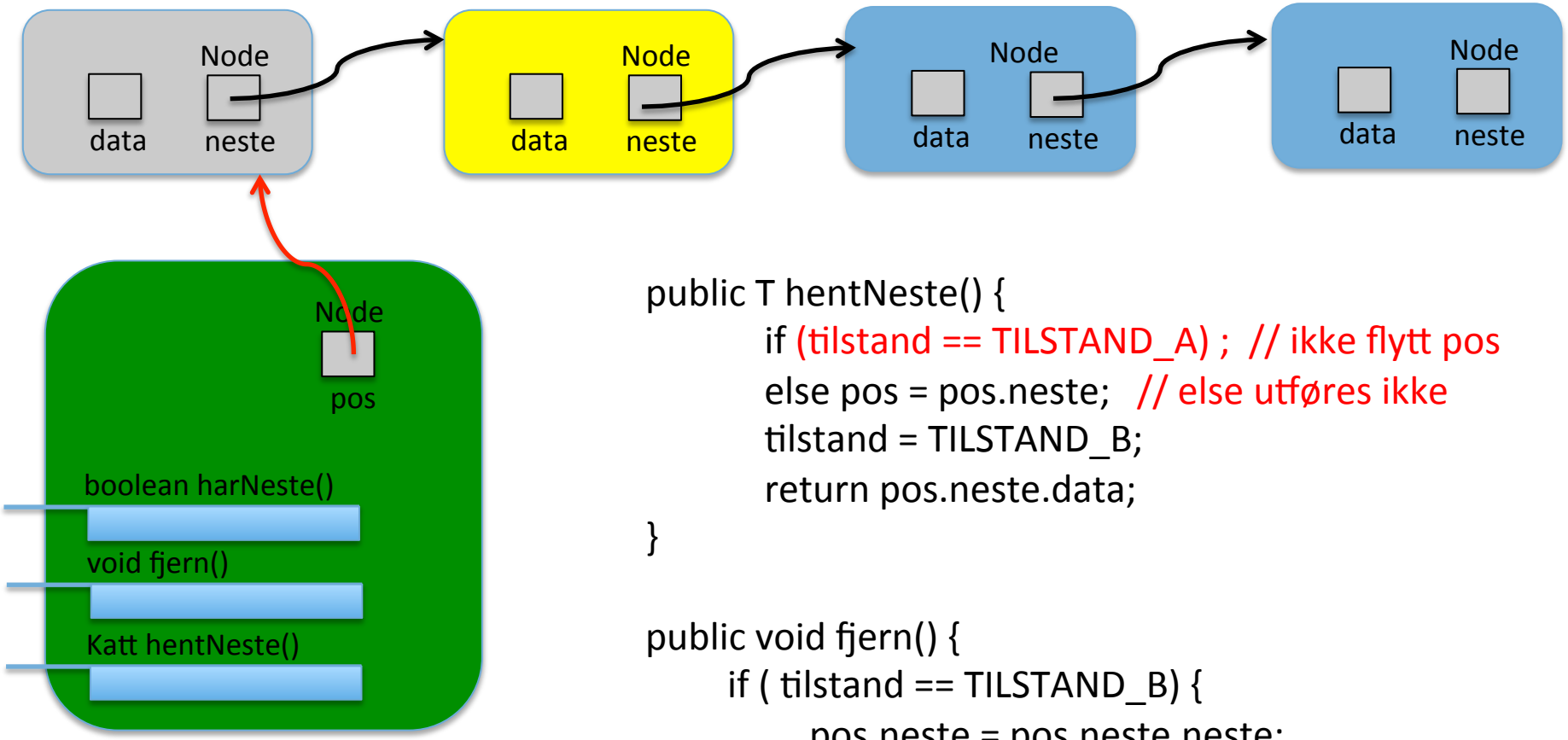
fjern() kan ikke kalles
pos.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

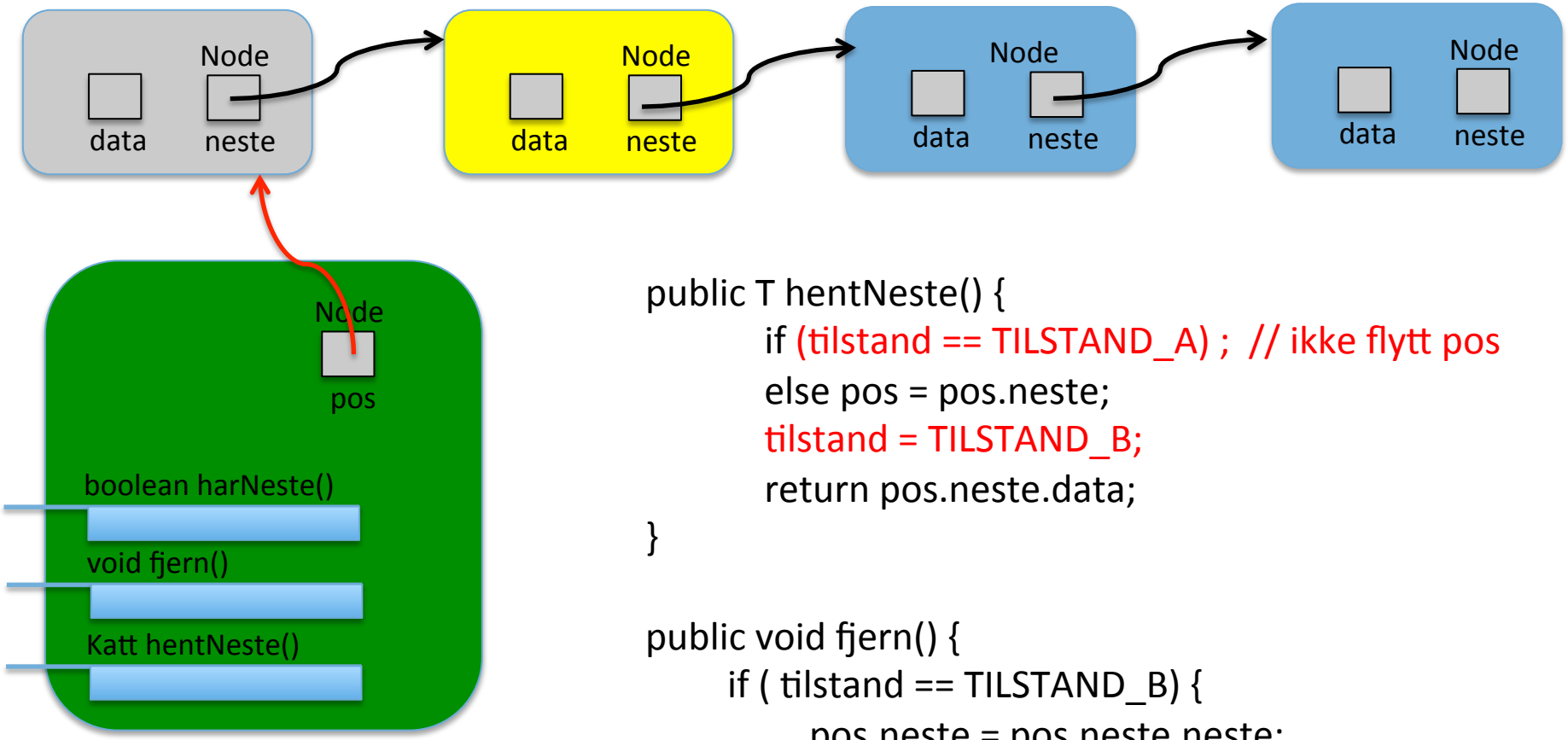
Tilstand A

fjern() kan ikke kalles
pos.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;   // else utføres ikke
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

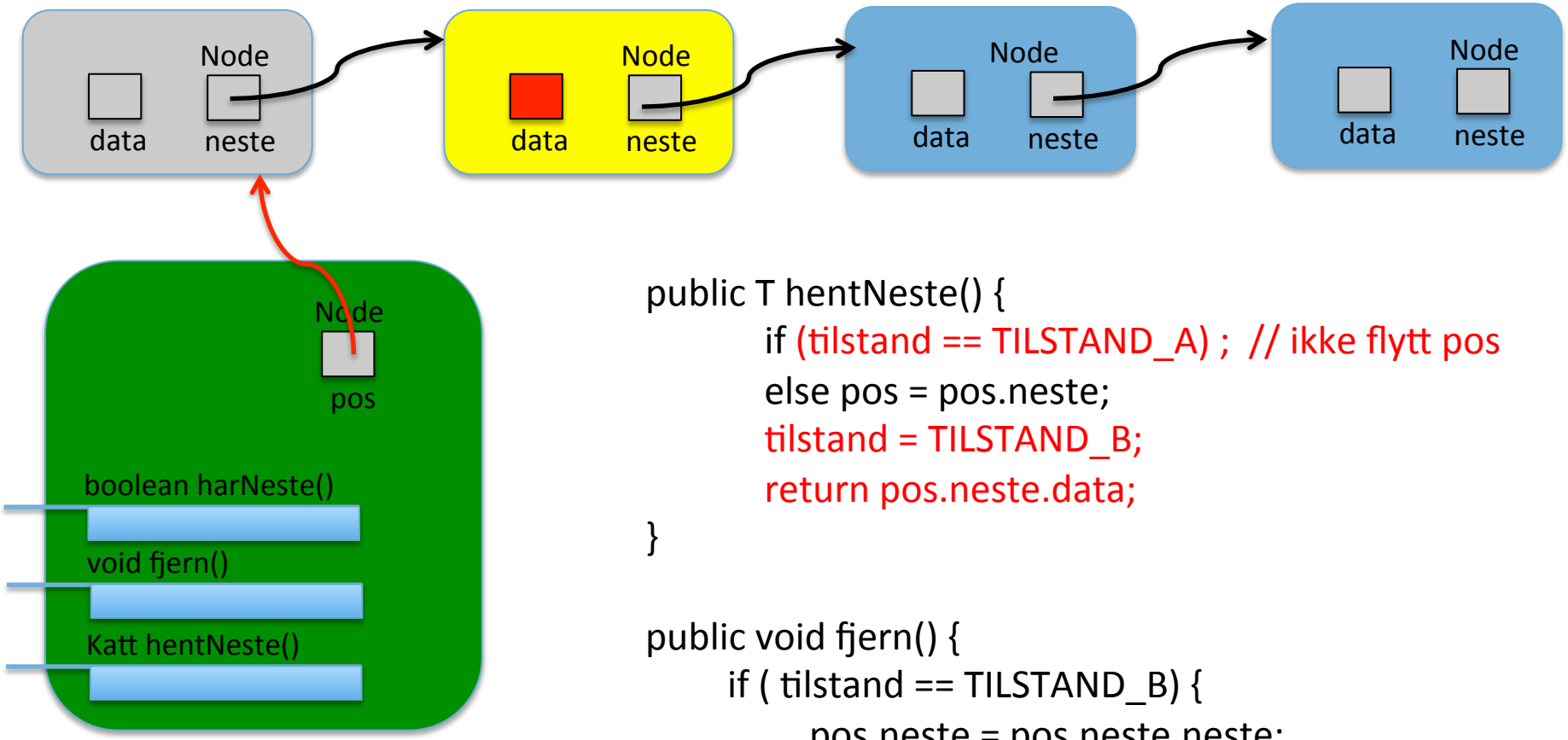
Tilstand B




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

Tilstand B

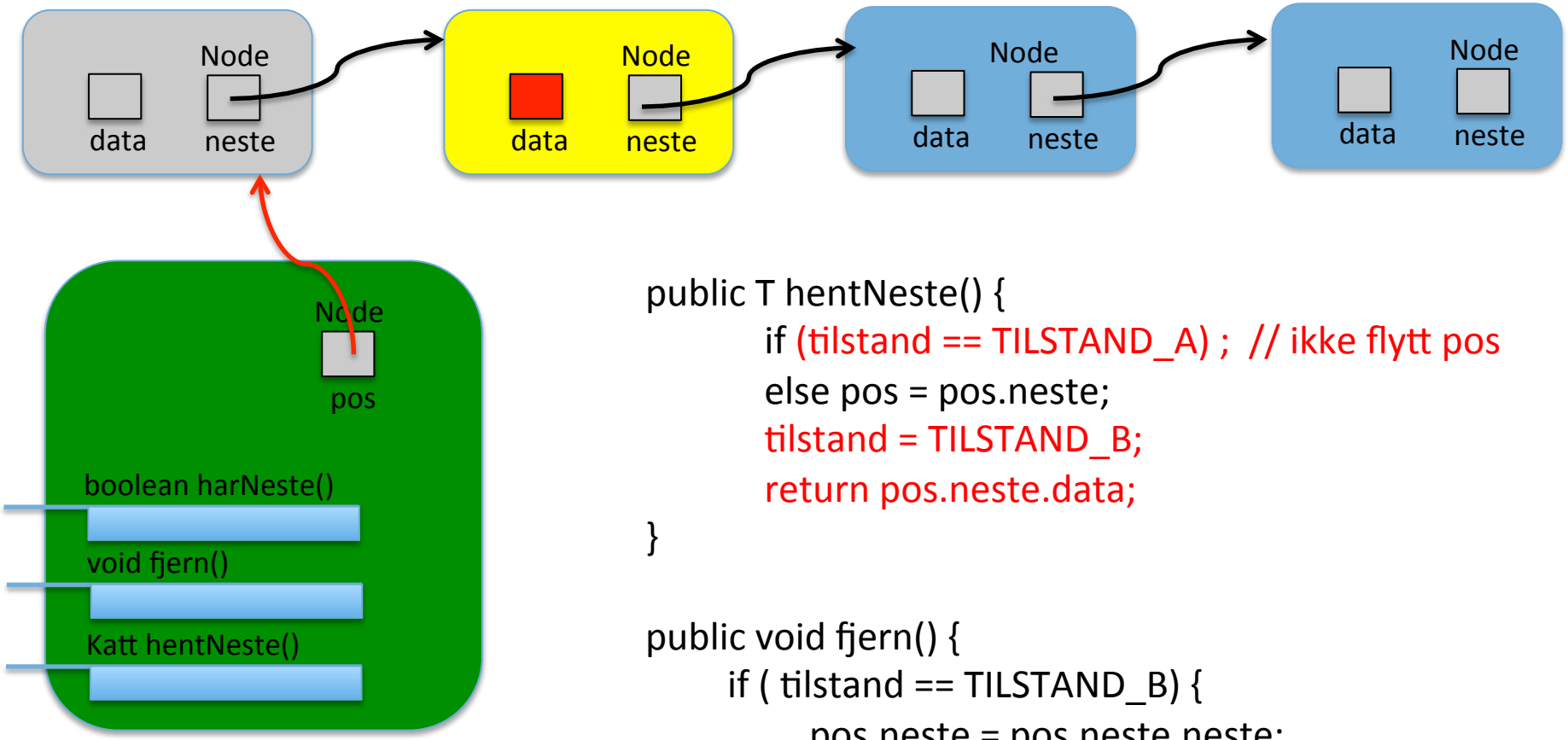
pos.neste er den som skal fjernes hvis fjern() kalles




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

Tilstand B

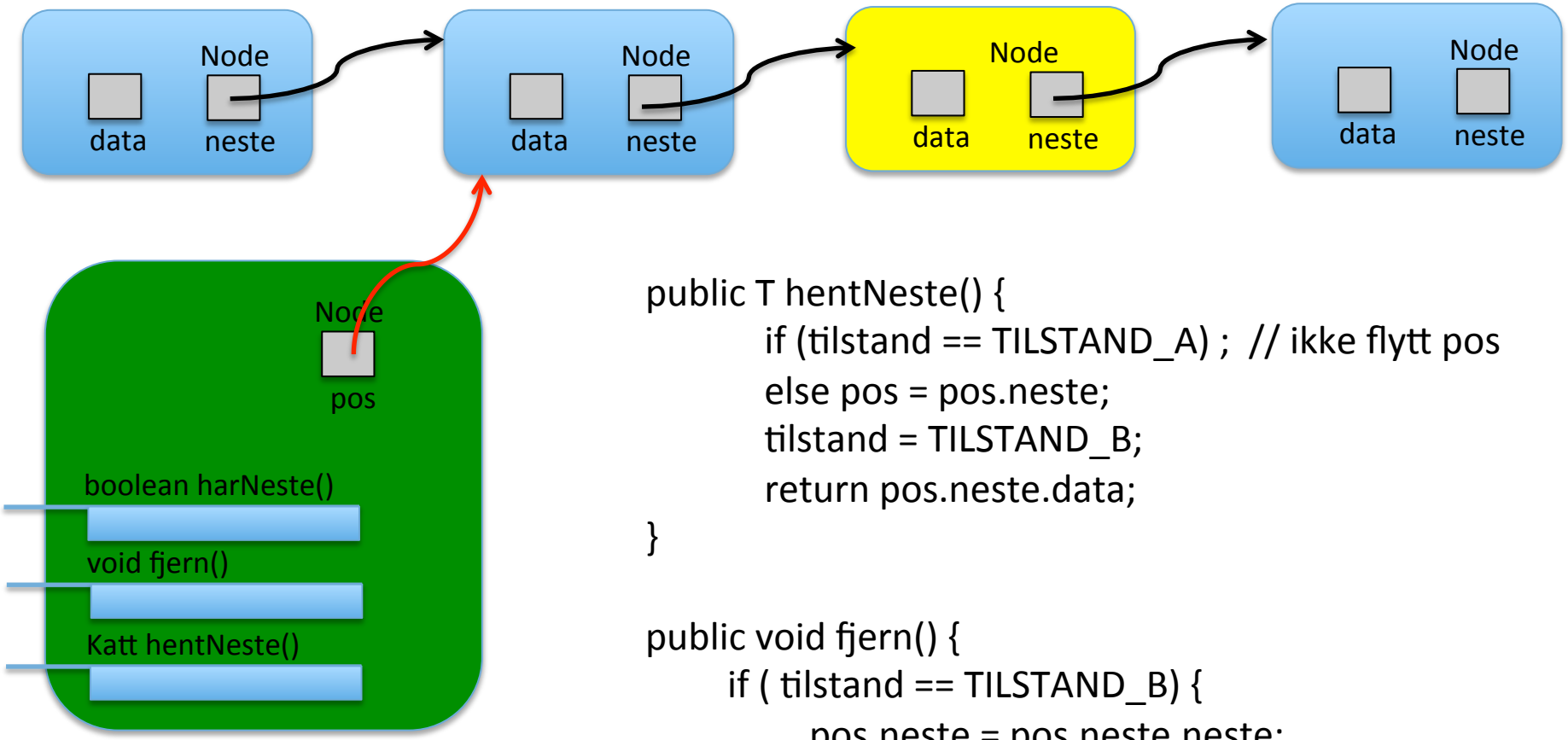
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

Tilstand B

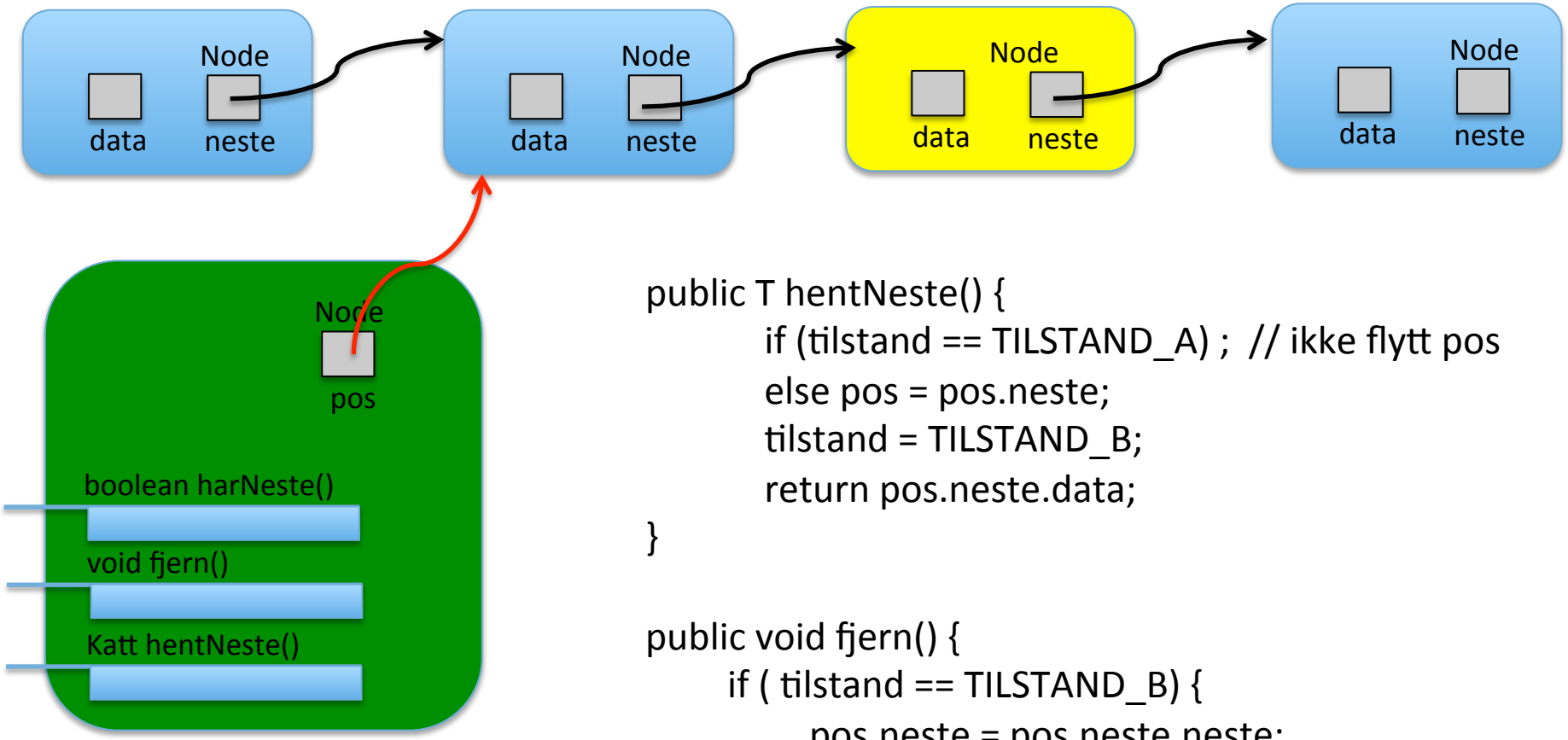
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else System.out.println("Feil kall på fjern()");
}
```

Tilstand B

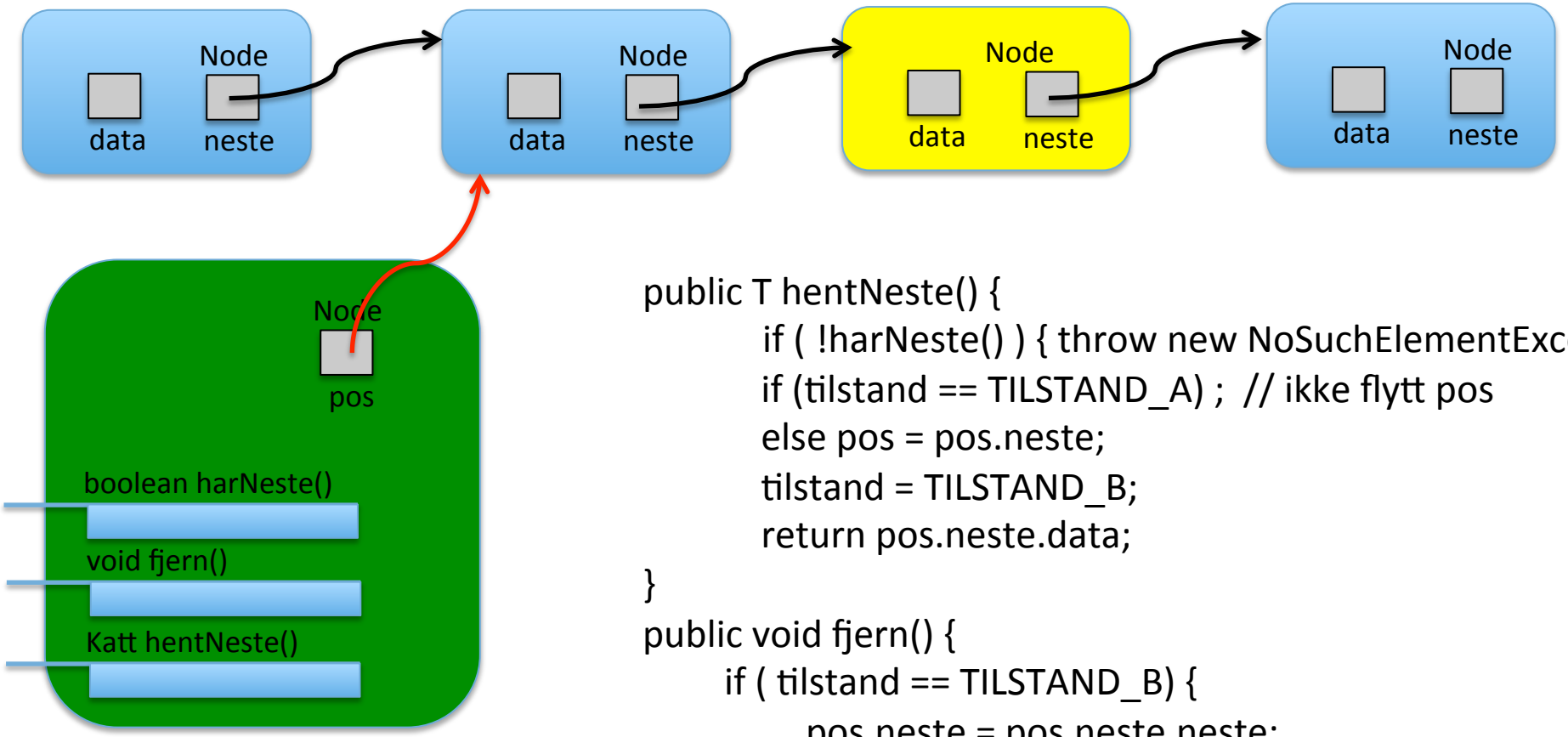
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}

public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B

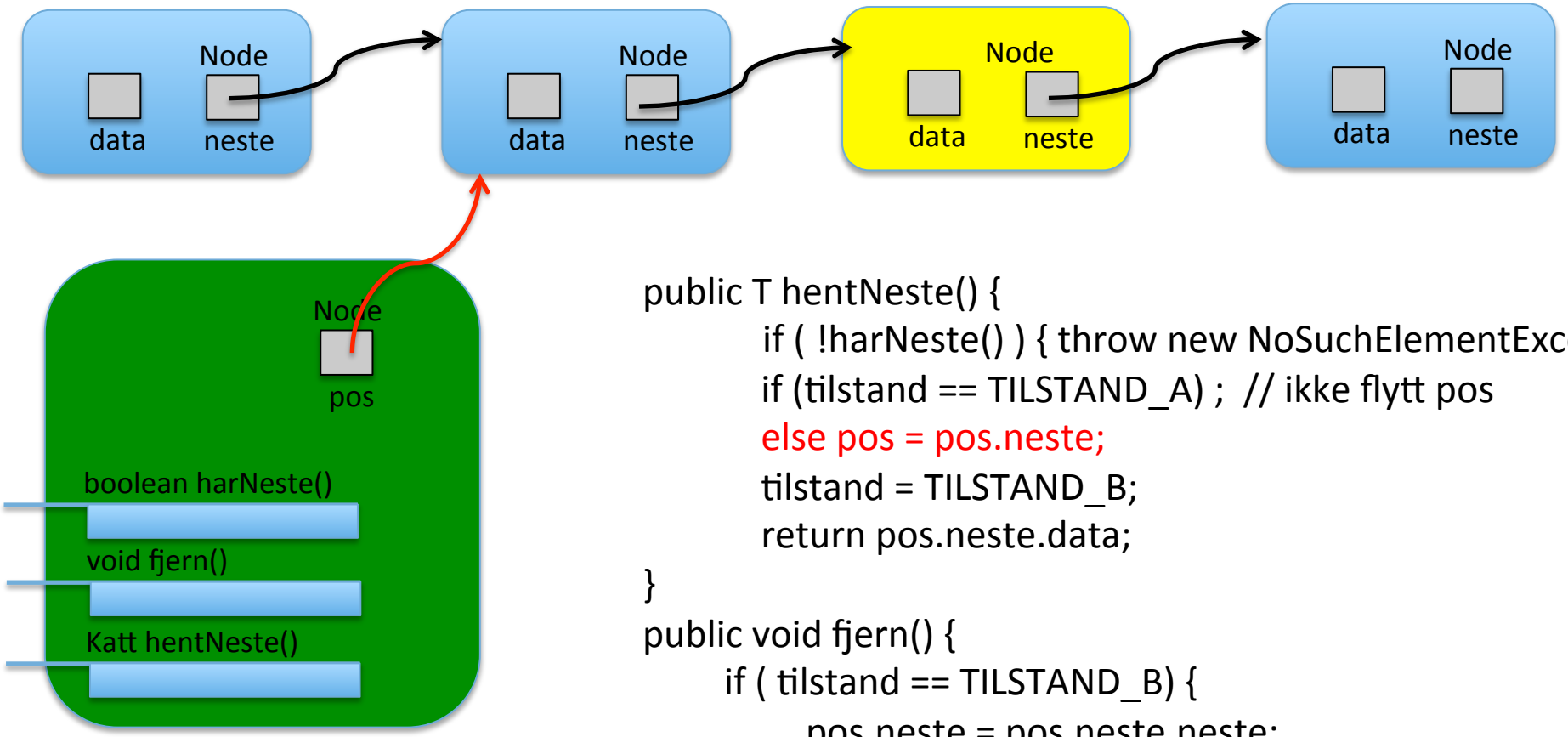
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B

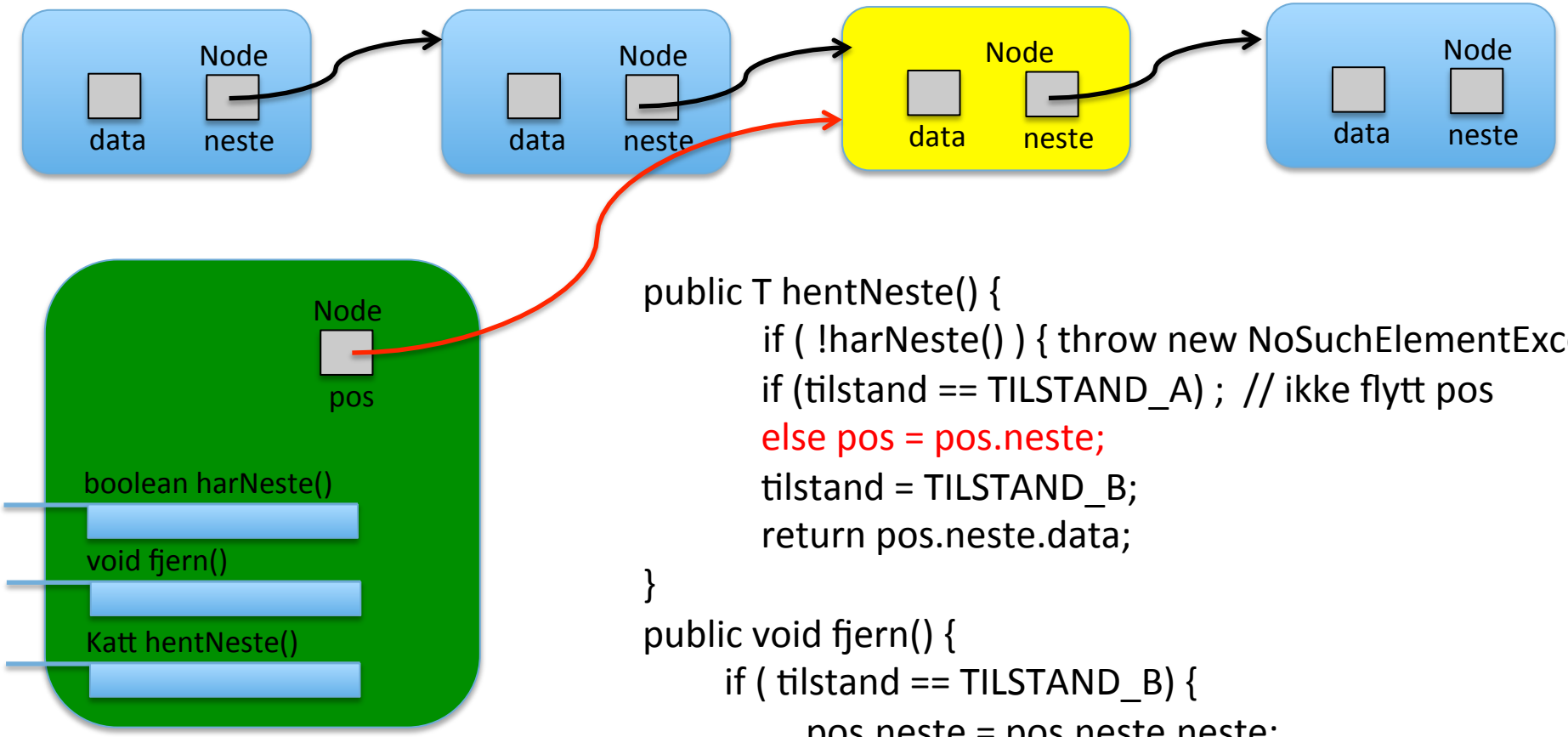
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B

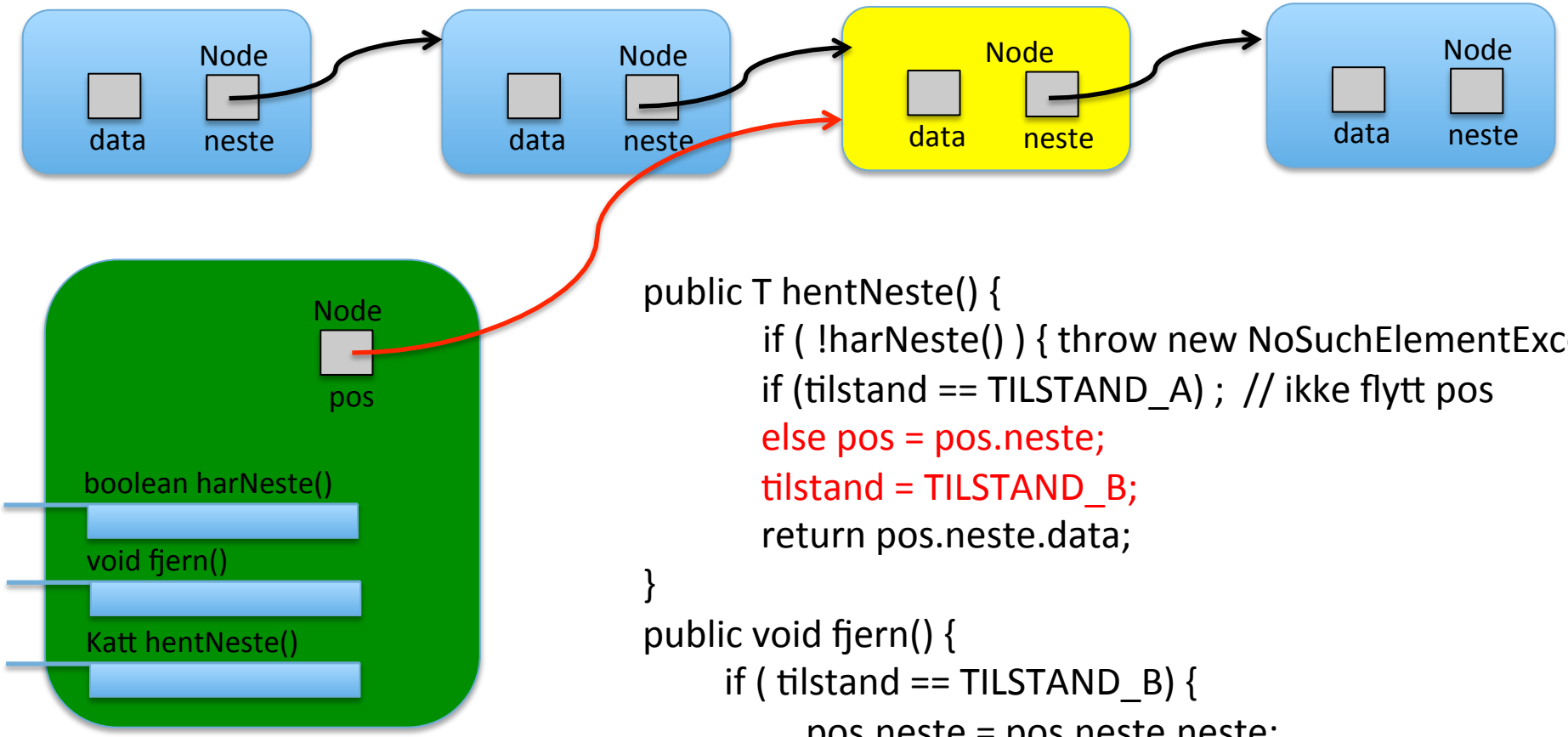
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B

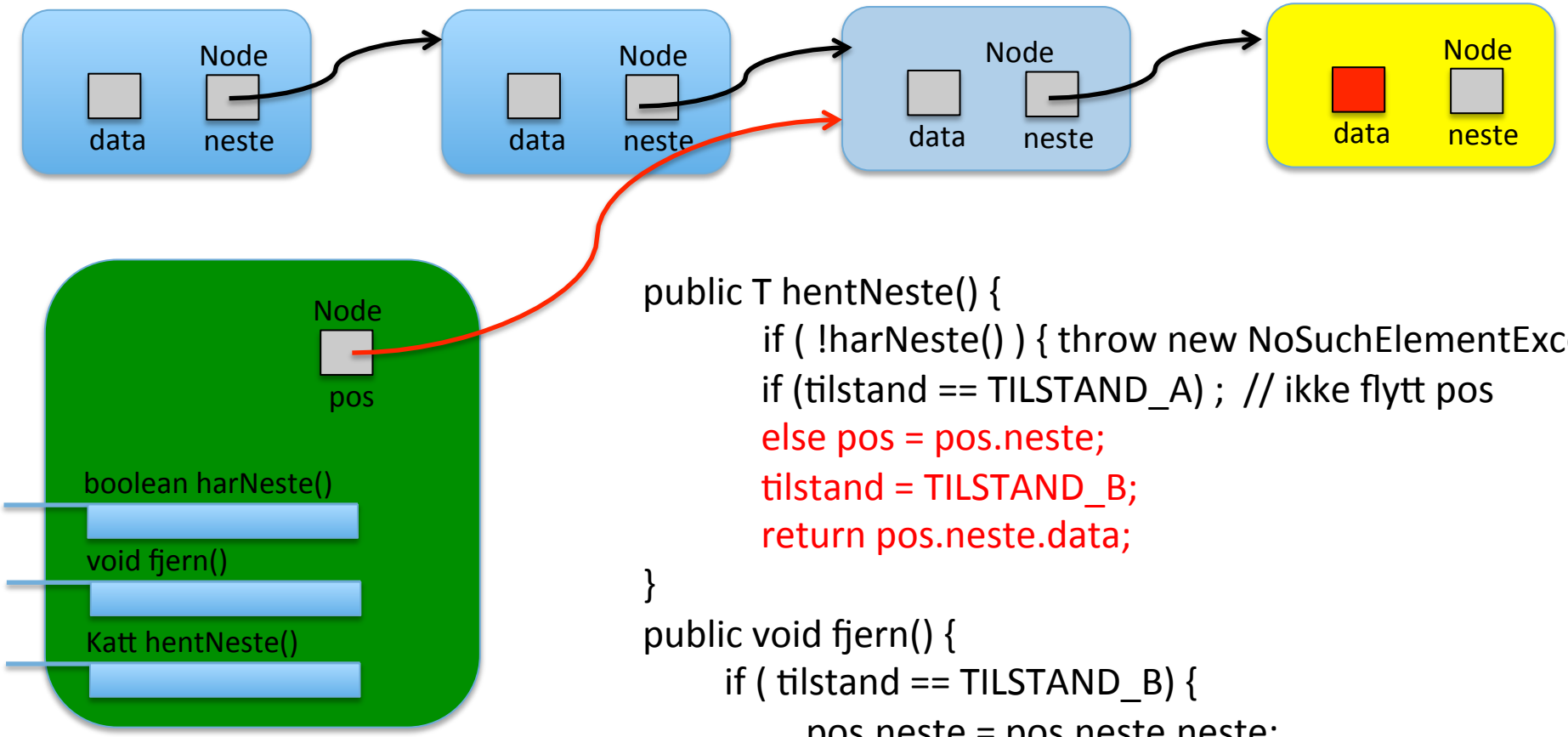
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B

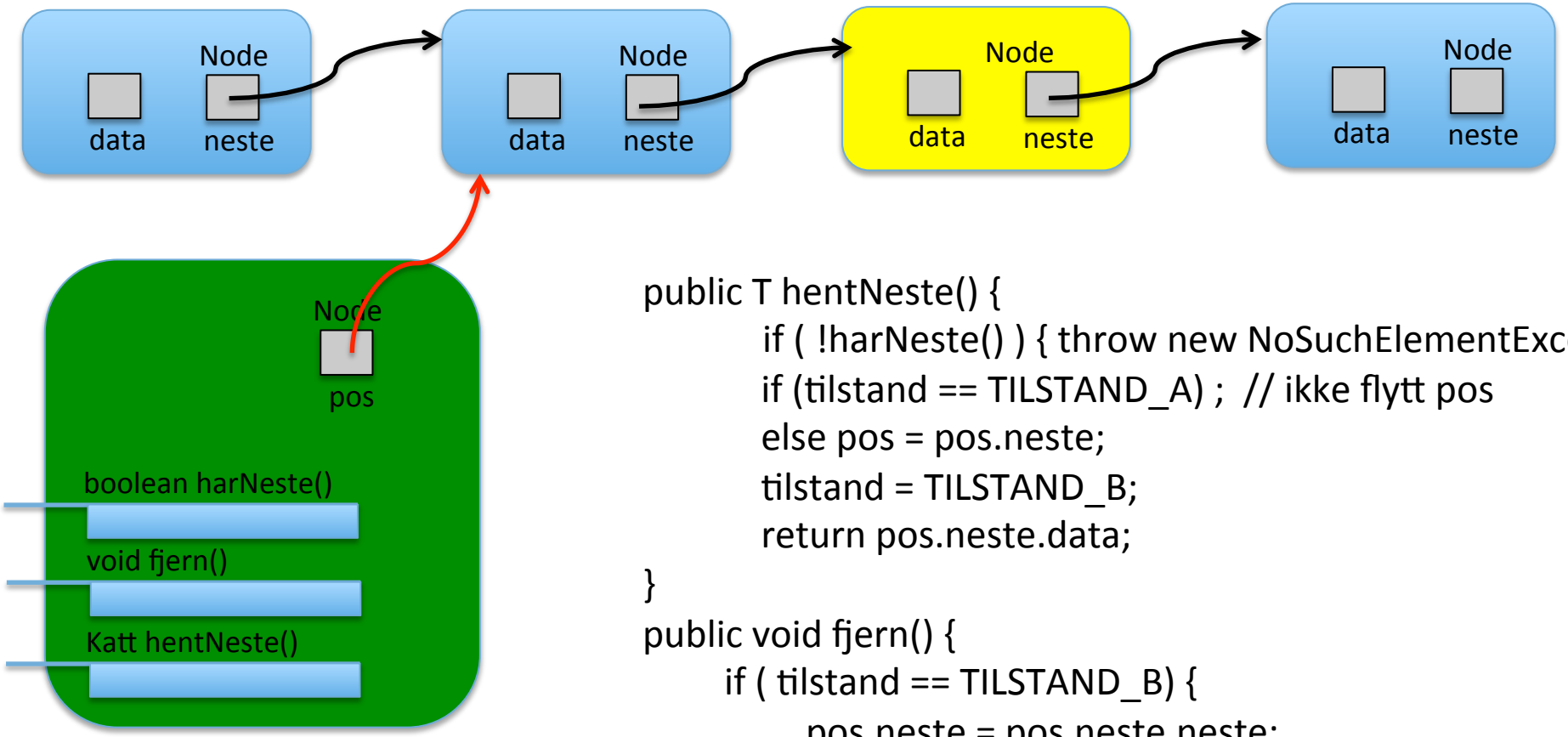
pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

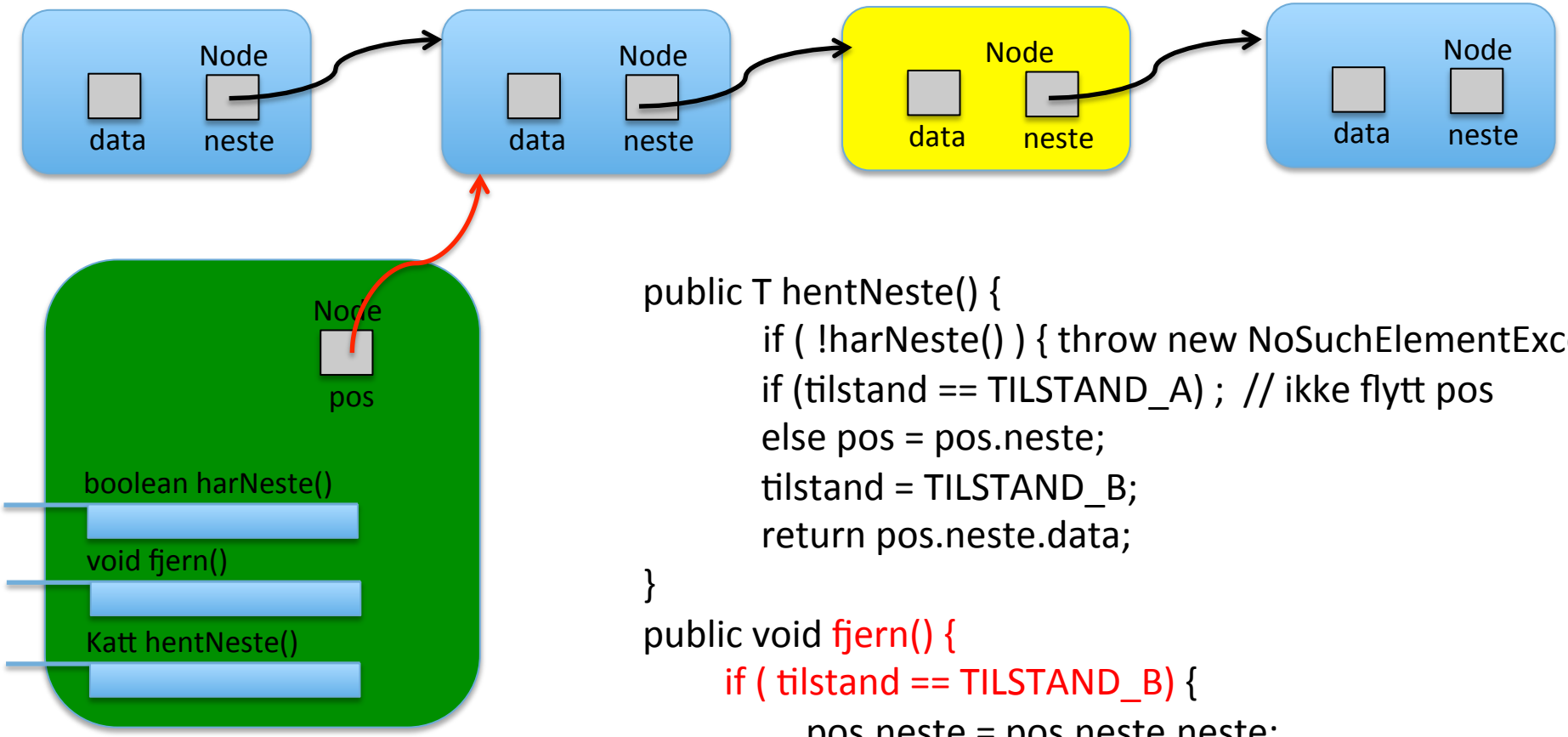
Tilstand B

pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()



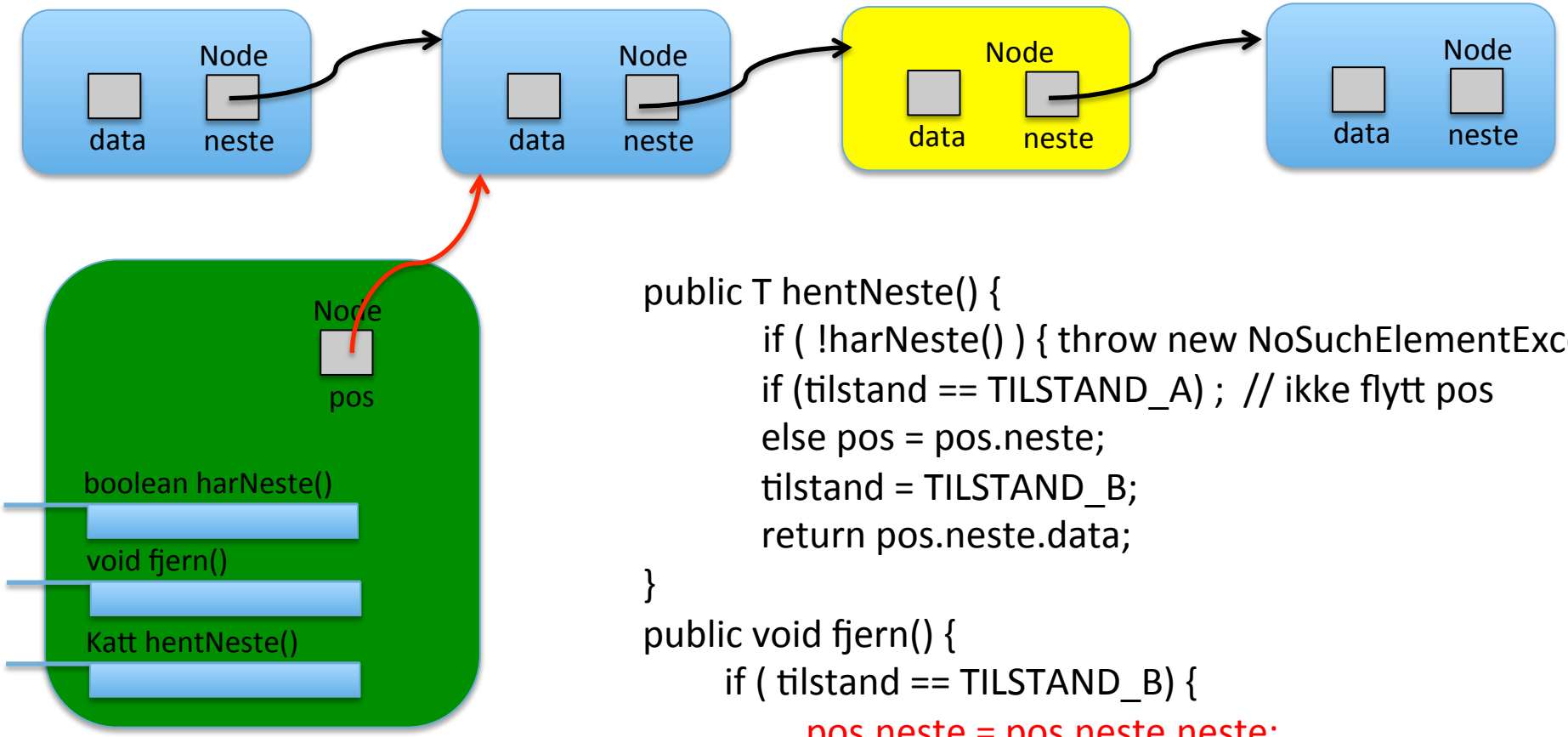

```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B



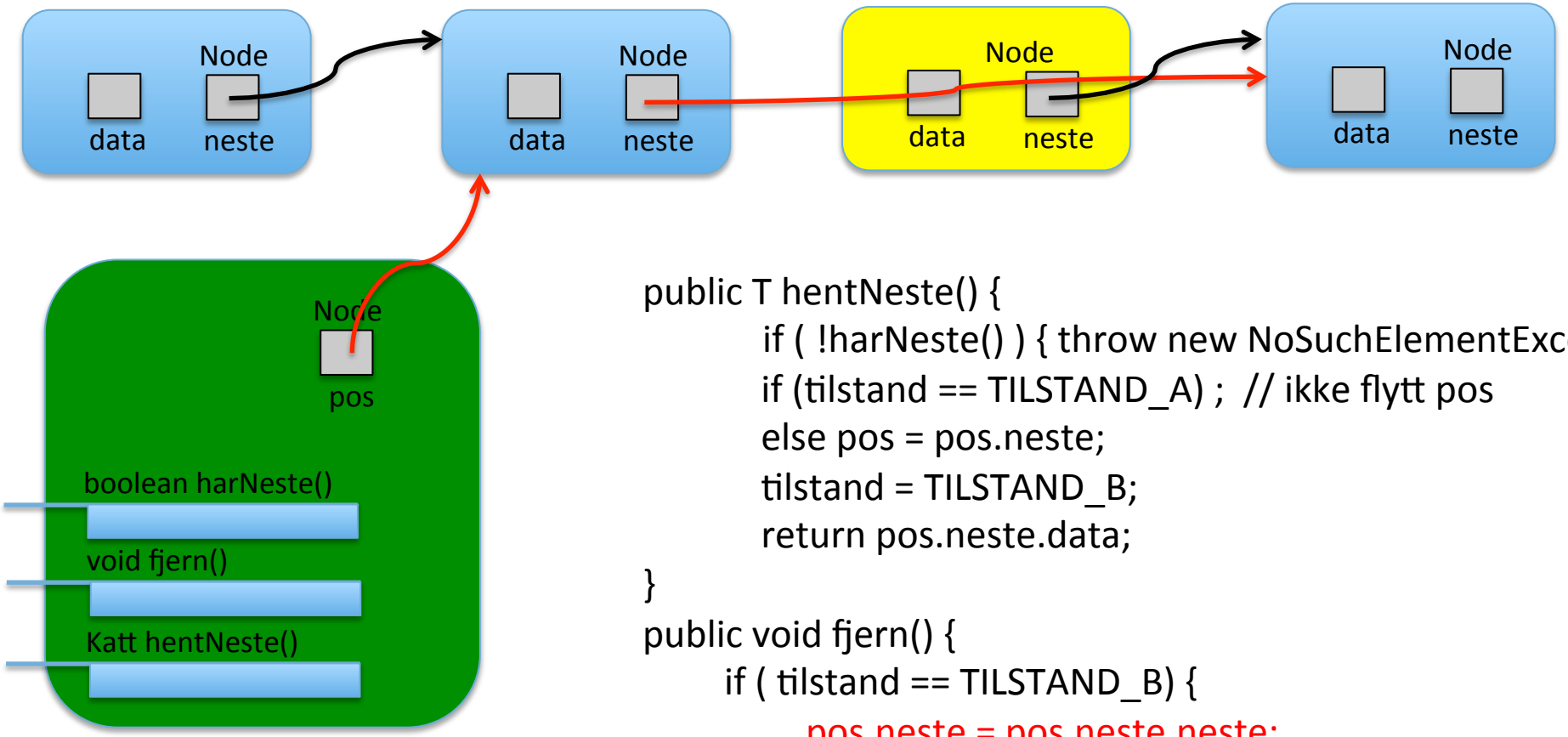

```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B



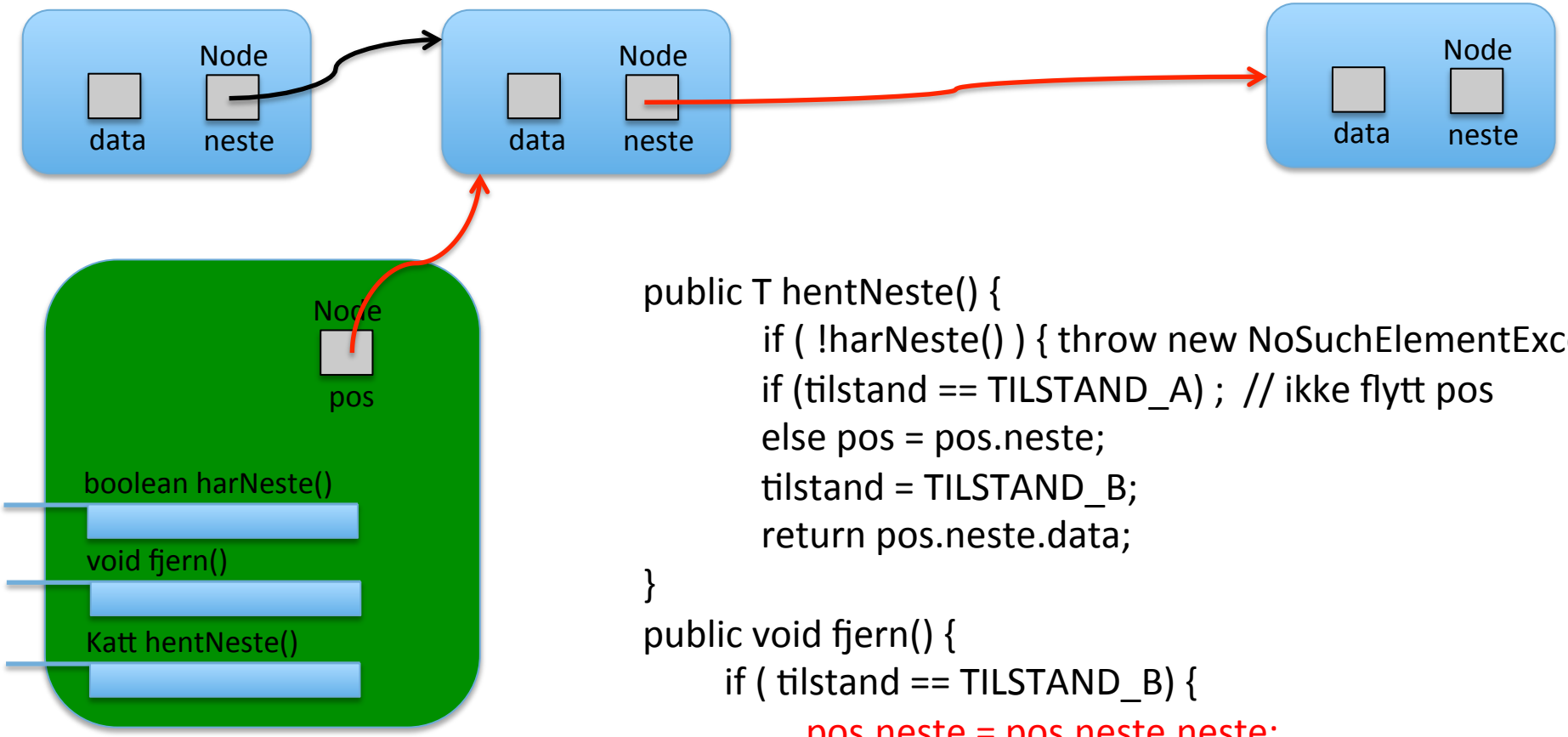

```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B



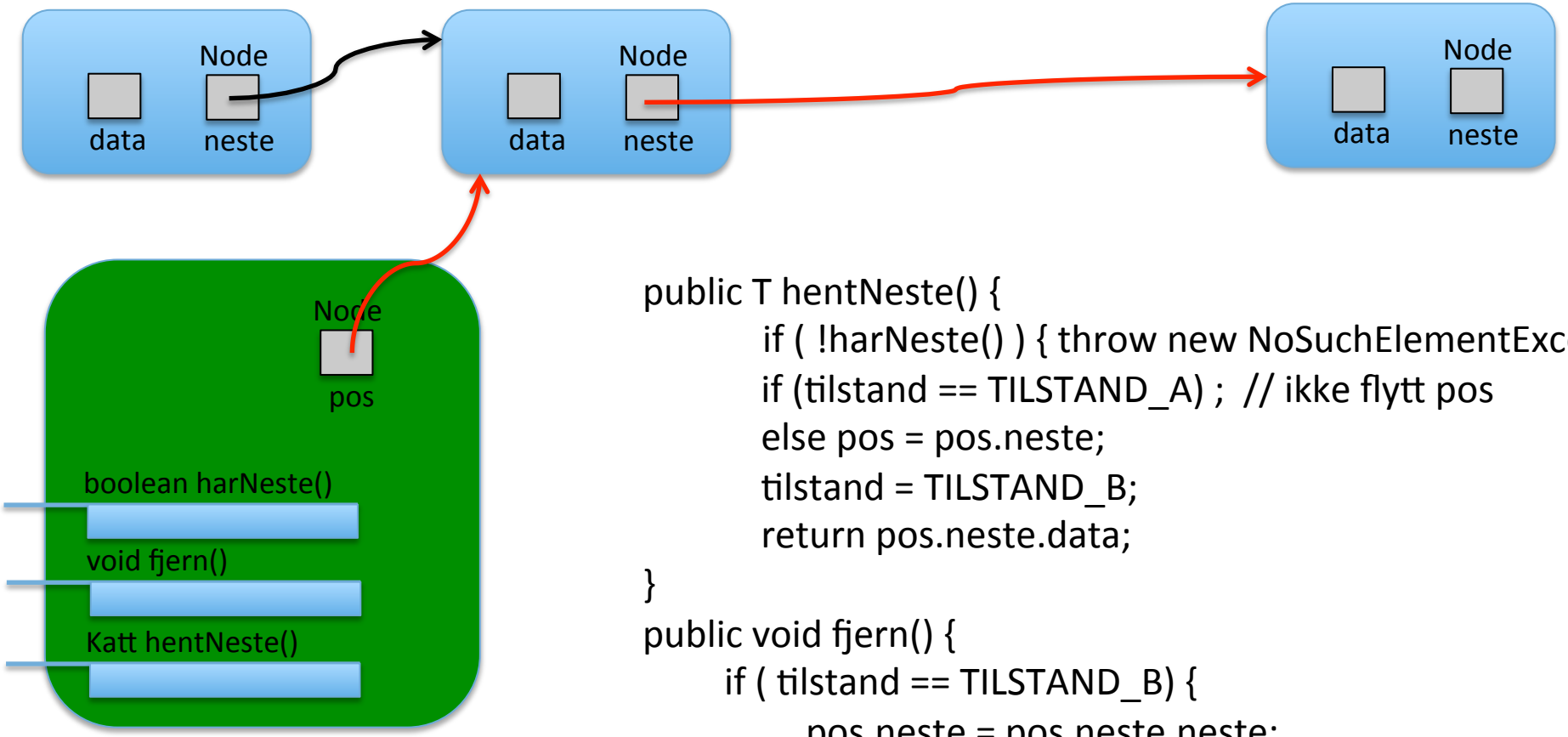

```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

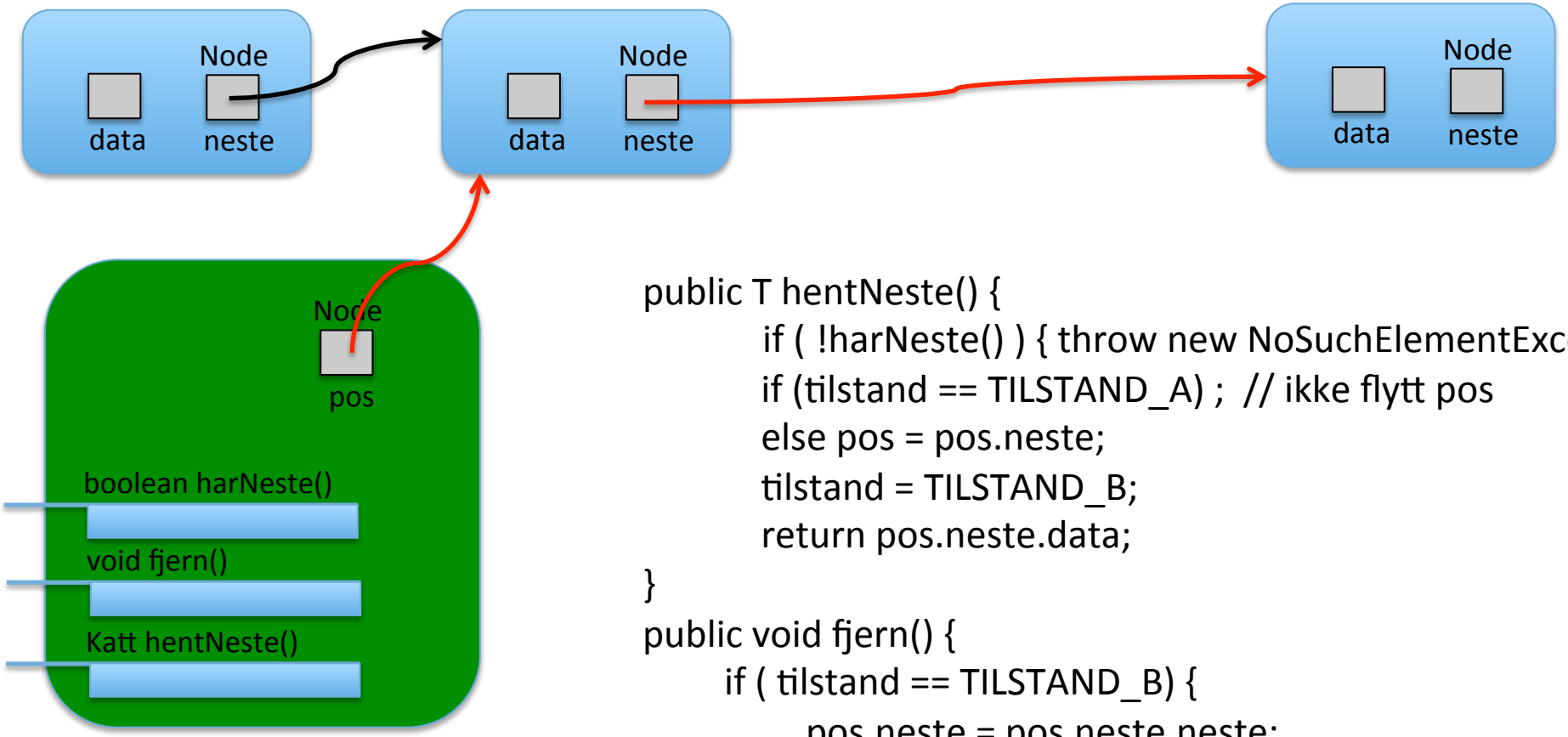
Tilstand A




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand A

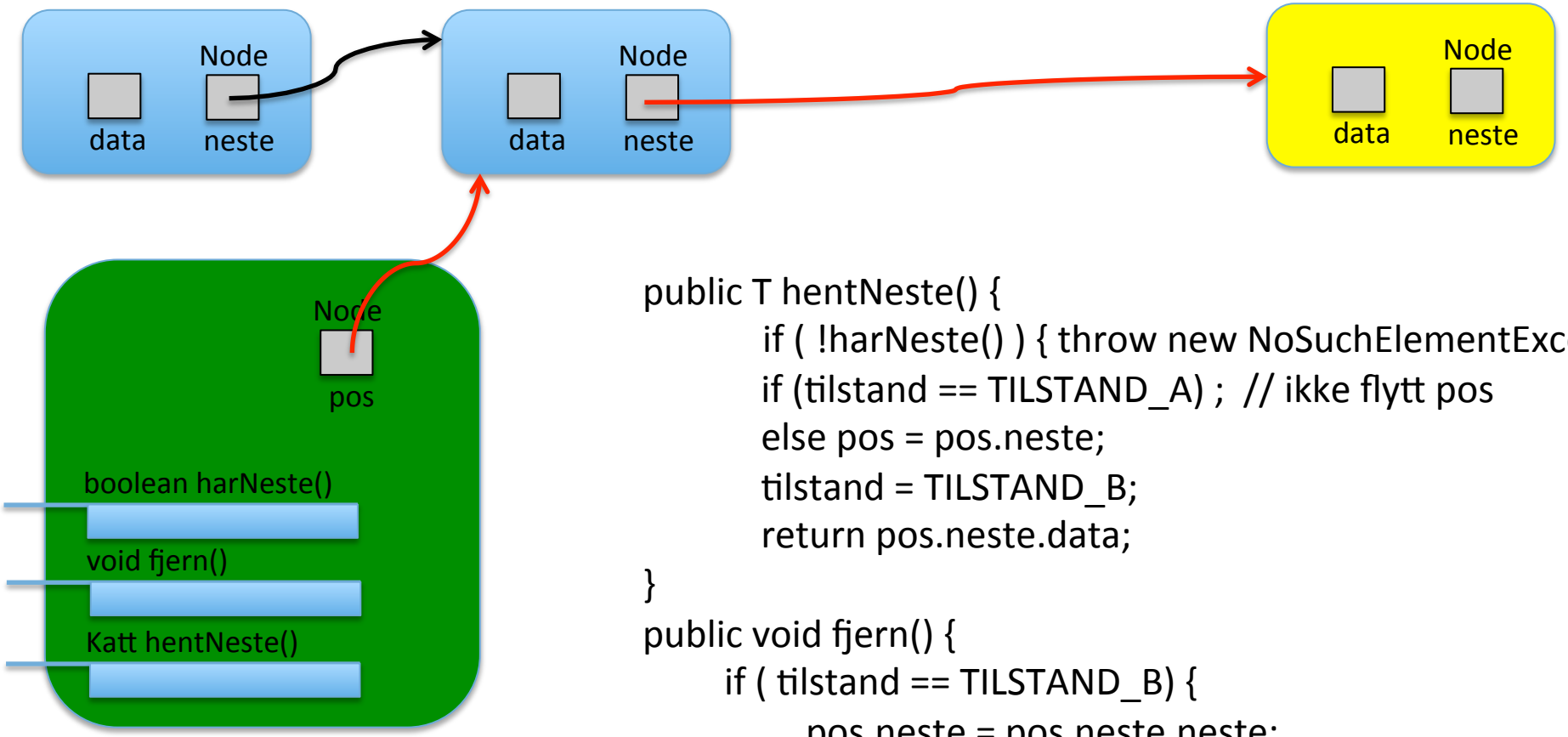
fjern() kan ikke kalles
pos.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand A

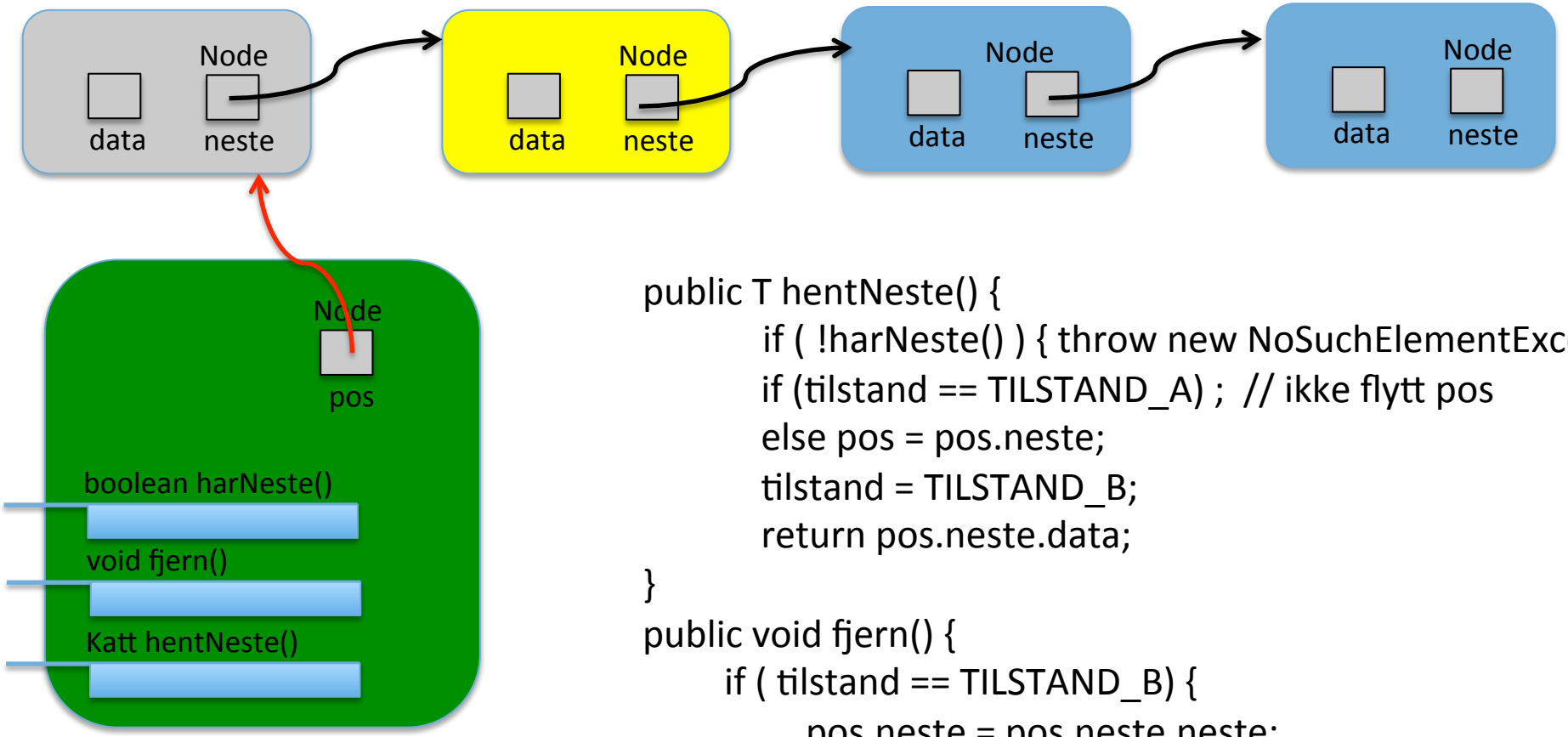
fjern() kan ikke kalles
pos.neste.data er neste som skal returneres av hentNeste()




```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand A

fjern() kan ikke kalles
pos.neste.data er neste som skal returneres av hentNeste()

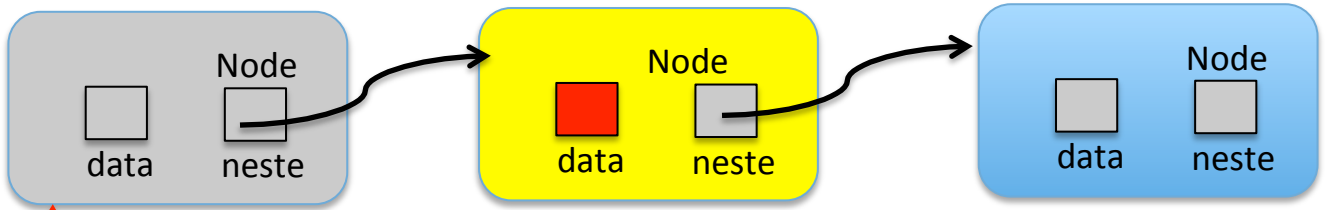


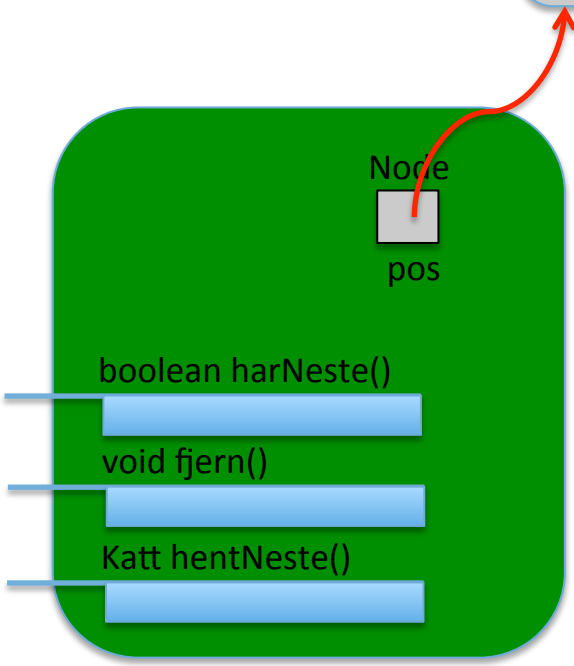

```
public T hentNeste() {
        if ( !harNeste() ) { throw new NoSuchElementException(); }
        if (tilstand == TILSTAND_A) ;  // ikke flytt pos
        else pos = pos.neste;
        tilstand = TILSTAND_B;
        return pos.neste.data;
}
public void fjern() {
      if ( tilstand == TILSTAND_B) {
            pos.neste = pos.neste.neste;
            tilstand = TILSTAND_A;
      } else  { throw new IllegalStateException ; }
}
```

Tilstand B

pos.neste er den som skal fjernes hvis fjern() kalles
pos.neste.neste.data er neste som skal returneres av hentNeste()

Spesialtilstand:
`foran == pos.neste`
(også initialtilstand)






```
public void fjern() {
        if ( tilstand == TILSTAND_B) {
                pos.neste = pos.neste.neste;
                tilstand = TILSTAND_A;
        } else  { throw new IllegalStateException ; }
}
```

1. Hvis foran *peker på objektet som skal fjernes (gul node) må vi før fjerningen huske å oppdatere* foran*. Dette må i såfall sjekkes ved hvert kall på fjern(). Implementer dette.*

2. Vi kan beholde fjern() uforandret hvis vi gjør endringer i datastrukturen i lenkelsista. Foreslå slike endringer.

```
java.util
interface Iterator<E> {
        boolean     hasNext();
        // Returns true if the iteration has more elements.

        E next();
        // Returns the next element in the iteration.

        void remove();
        //Removes from the underlying collection the
        //last element returned by this iterator (optional operation).
}
```

```
class Lenkeliste<T> {

    private class ListeIterator implements java.util.Iterator<T> {
      public boolean hasNext() { ….}
      public T next() {….}
      public void remove() { ….}
    }

    public java.util.Iterator<T> iterator() {
      return new ListeIterator ();
    }

}
```

```
java.lang

interface Iterable<T> {
    java.util.Iterator<T> iterator();
    // Returns an iterator over a set of elements of type T.
}
```

```
class Lenkeliste<T> implements Iterable<T> {

    private class ListeIterator implements java.util.Iterator<T> {
      public boolean hasNext() { ….}
      public T next() {….}
      public void remove() { ….}
    }

    public java.util.Iterator<T> iterator() {
      return new ListeIterator ();
    }

}
```

```
Lenkeliste<Hund> kennel = new Lenkeliste<Hund>();

kennel.settInnForan(new Hund("Trofast"));
…

for (Hund h : kennel) {
      h.vaksinerMot( new Vaksine("kennelhoste") ) ;
}
```

```
class Lenkeliste<T> implements Iterable<T> {

     private class ListeIterator implements java.util.Iterator<T> {
        public boolean hasNext() { ….}
        public T next() {….}
        public void remove() { ….}
     }

     public java.util.Iterator<T> iterator() {
        return new ListeIterator ();
     }

}
```

```
import java.util.*;

class Lenkeliste<T> implements Iterable<T> {

     private class ListeIterator implements Iterator<T> {
        public boolean hasNext() { ….}
        public T next() {….}
        public void remove() { ….}
     }

     public Iterator<T> iterator() {
        return new ListeIterator ();
     }
}
```

```
Lenkeliste<Katt> lenkeli = new Lenkeliste<Katt>();

Iterator<Katt> li = lenkeli.iterator();

for (Katt kk : lenkeli) {
          System.out.println(kk.toString() );
}
```