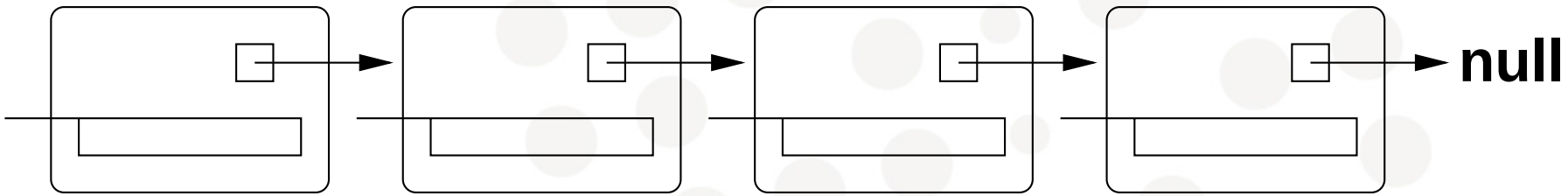
INF1010—Rekursive metoder, binære søketrær

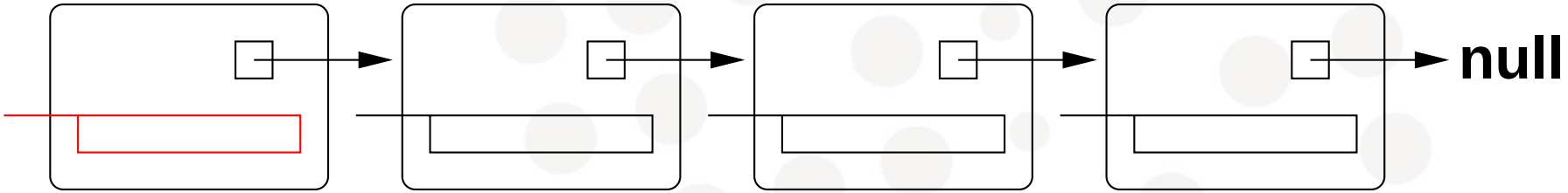
Algoritmer: Mer om rekursive kall mellom objekter
Ny datastruktur: binært tre

UNIVERSITETET I OSLO

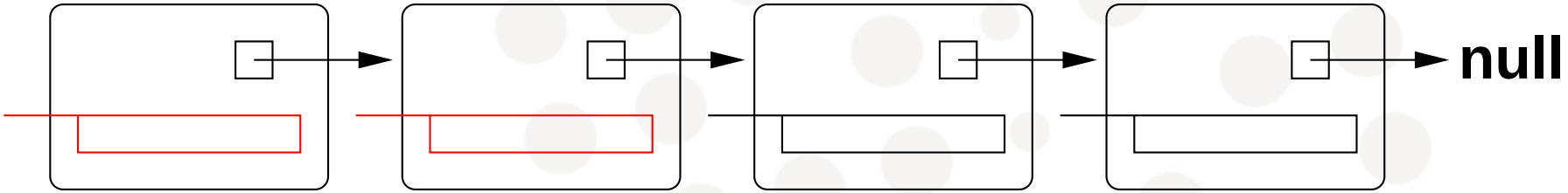
```
public void skrivUtMeg() {
    System.out.println(navn +
        "␣er␣venn␣med␣" + minBesteVennHeter());
}

public void skrivUtAllt() {
    skrivUtMeg();
    if (hentBestevenn() != null) hentBestevenn().skrivUtAllt();
}
```



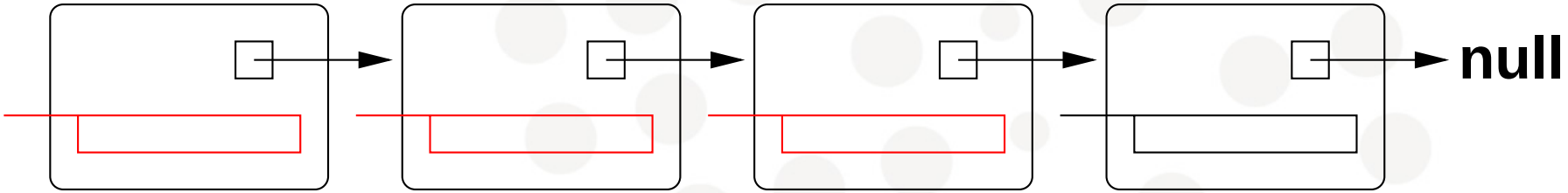

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



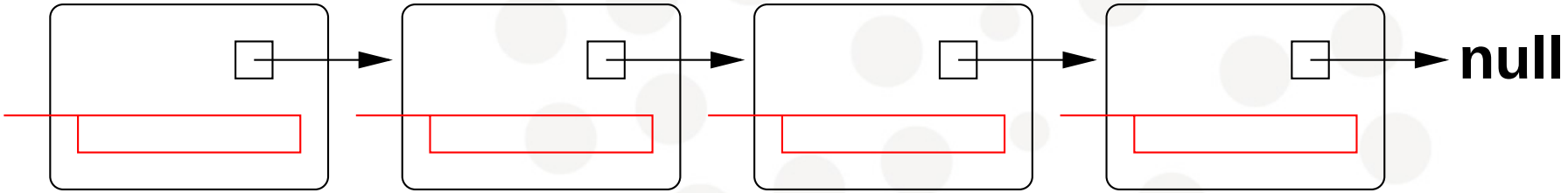

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



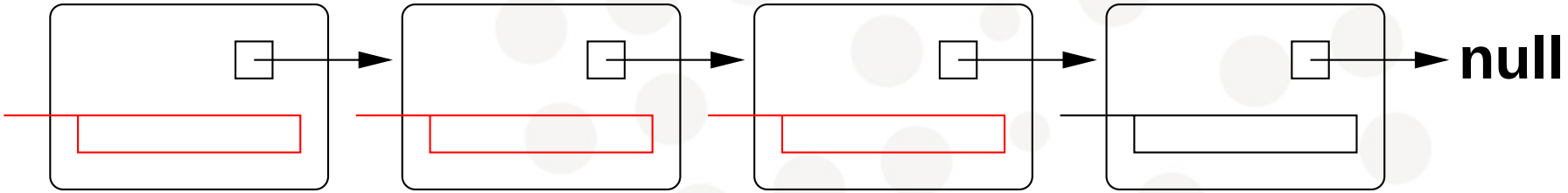

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



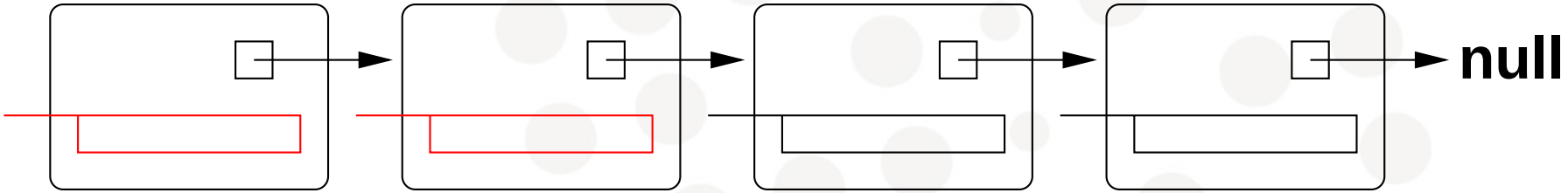

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



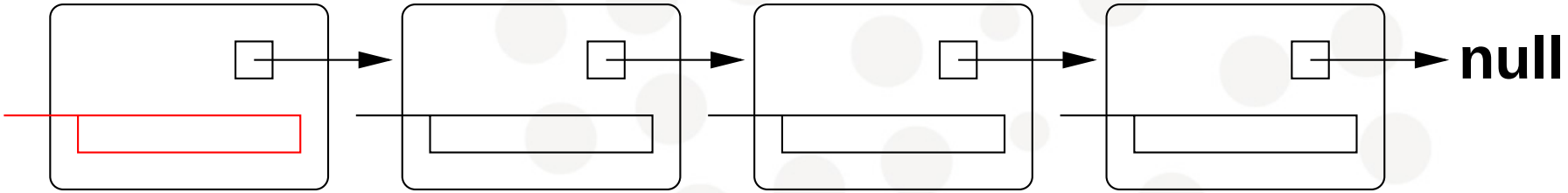

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



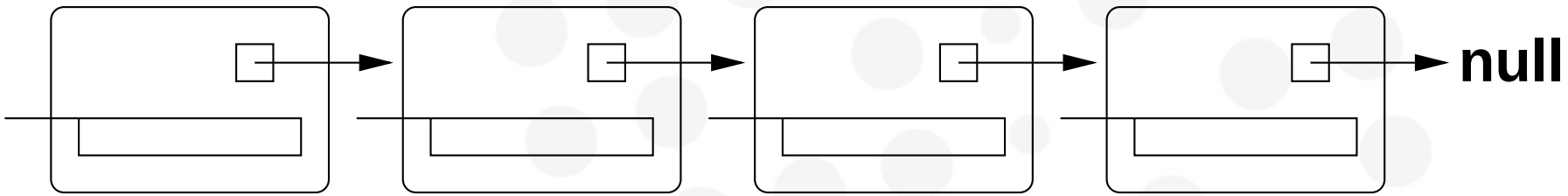

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



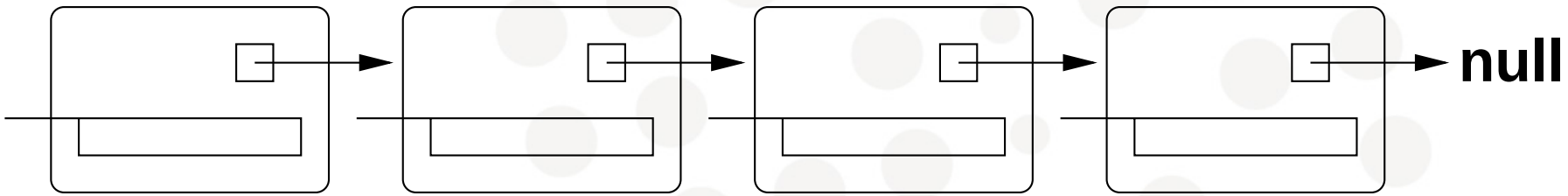

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



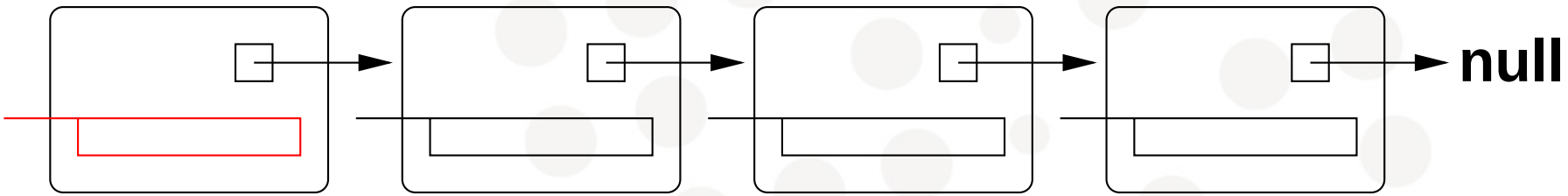

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



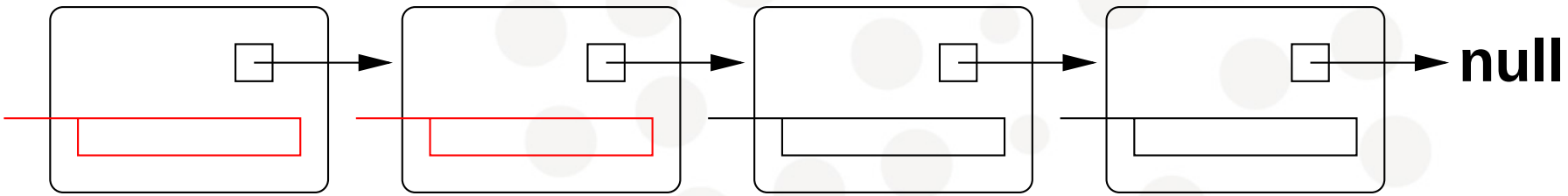

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      skrivUtMeg();
      if (neste != null) neste.skrivUtMegOgResten();
    }

    ...
}
```



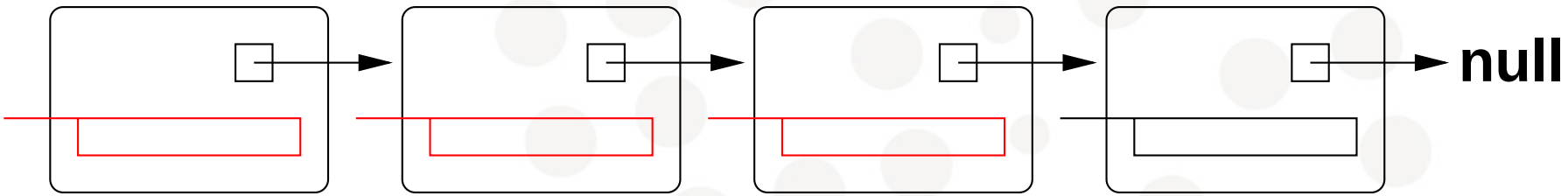

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```



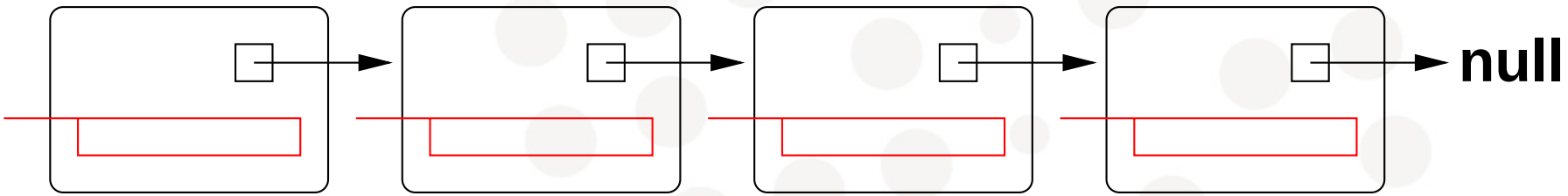

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```



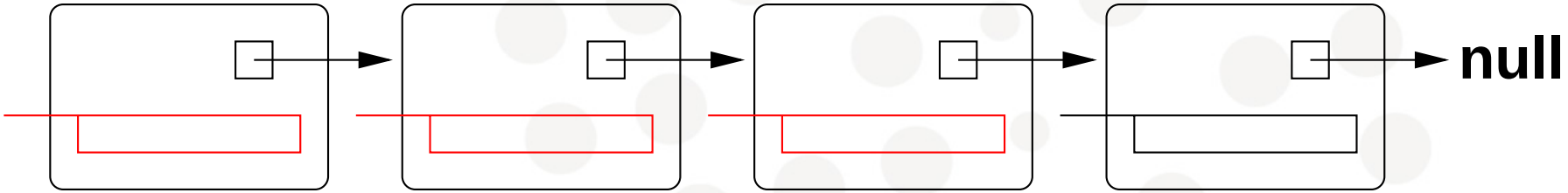

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```



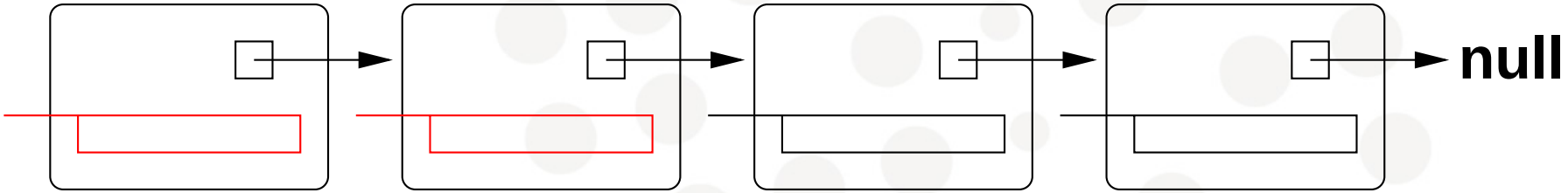

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```



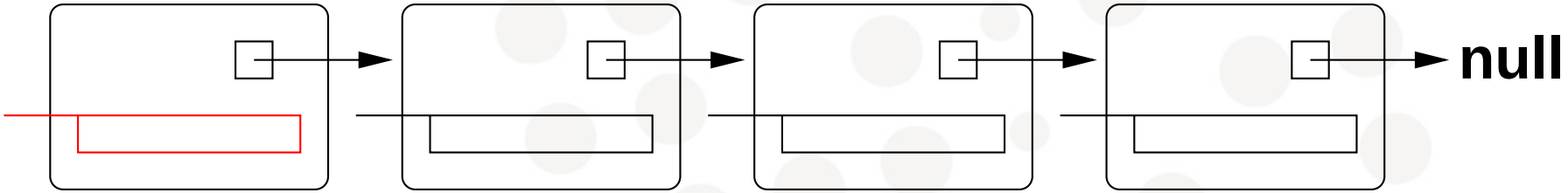

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```



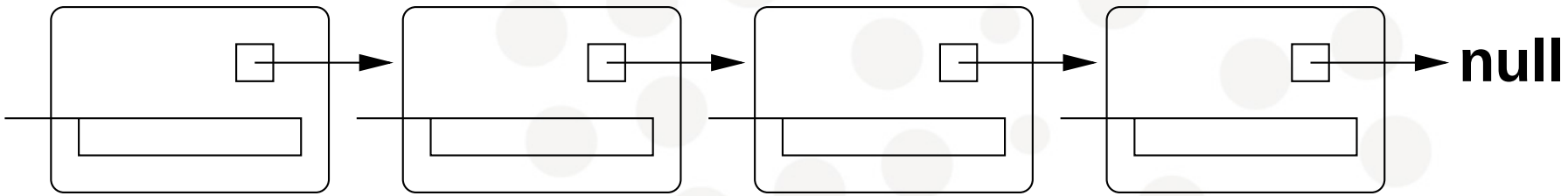

```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```




```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```




```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```




```
class ListeElement {
    Object data;
    ListeElement neste;

    public void skrivUtMegOgResten() {
      if (neste != null) neste.skrivUtMegOgResten();
      skrivUtMeg();
    }

    ...
}
```

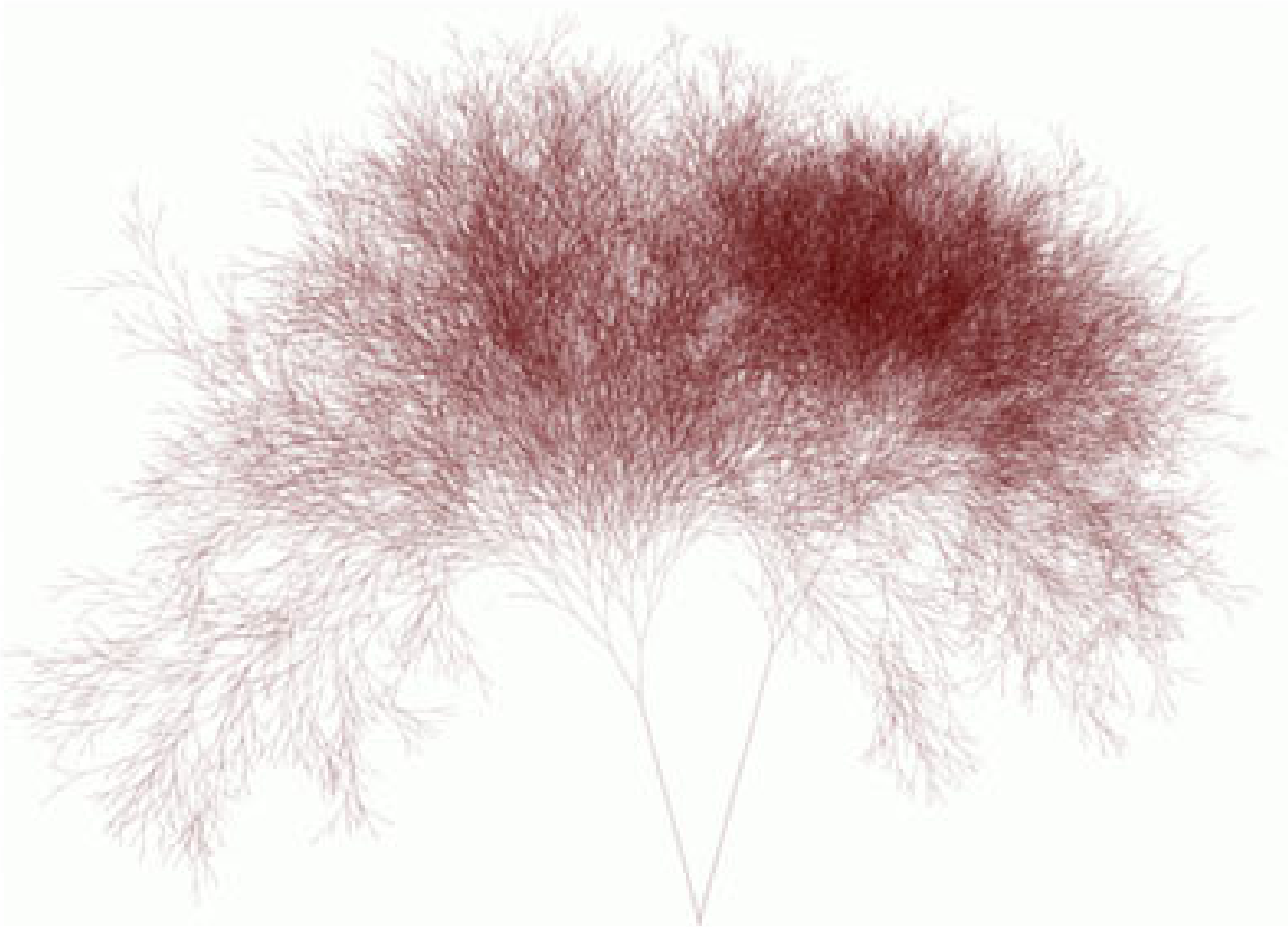
```
class ListeElement {
   Object data;
   ListeElement neste;

   public void settInnSist(ListeElement ny) {
     if (neste == null) neste = ny;
     else neste.settInnSist(ny);
   }
}
```

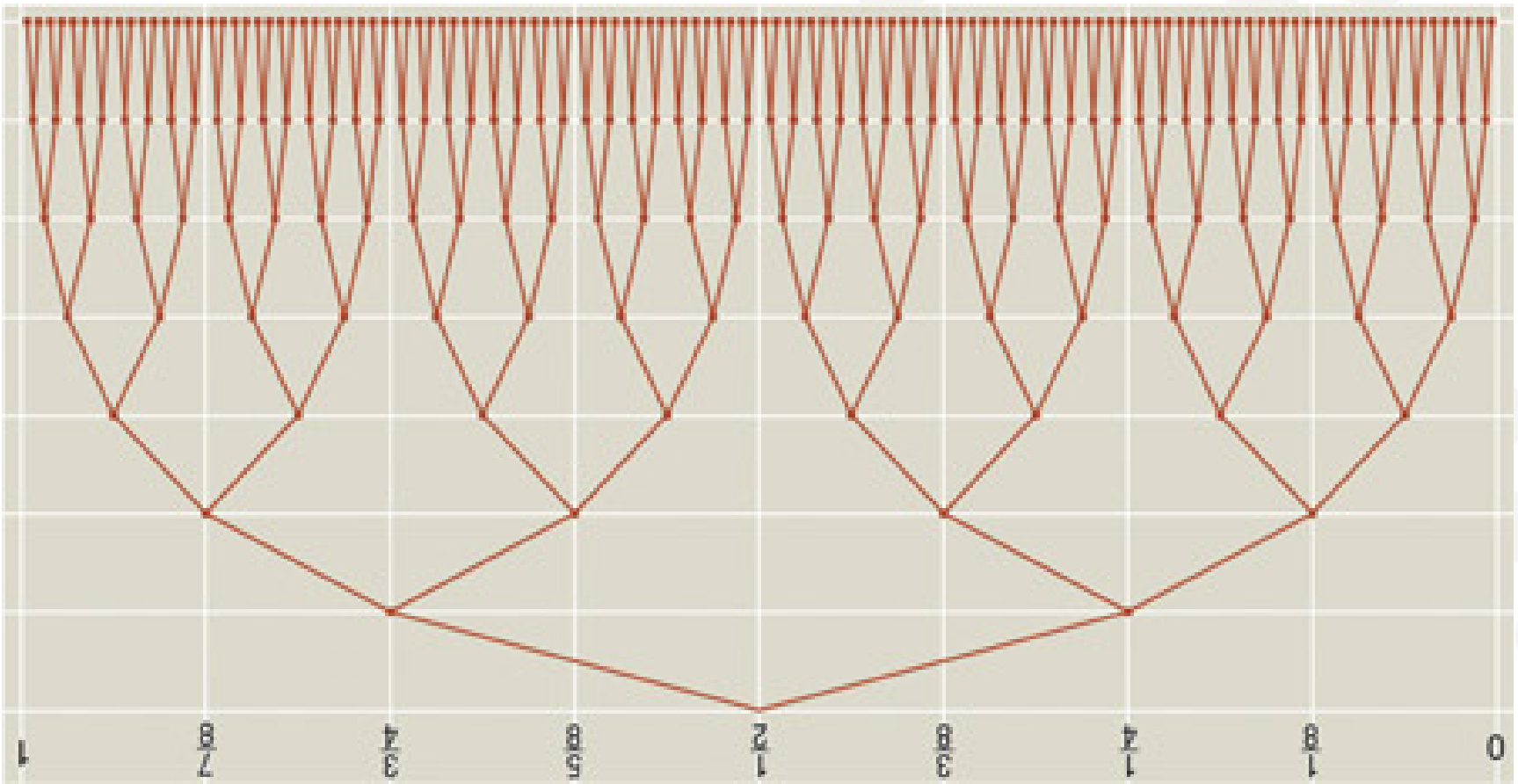
Eksempler på enkle rekursive metoder:

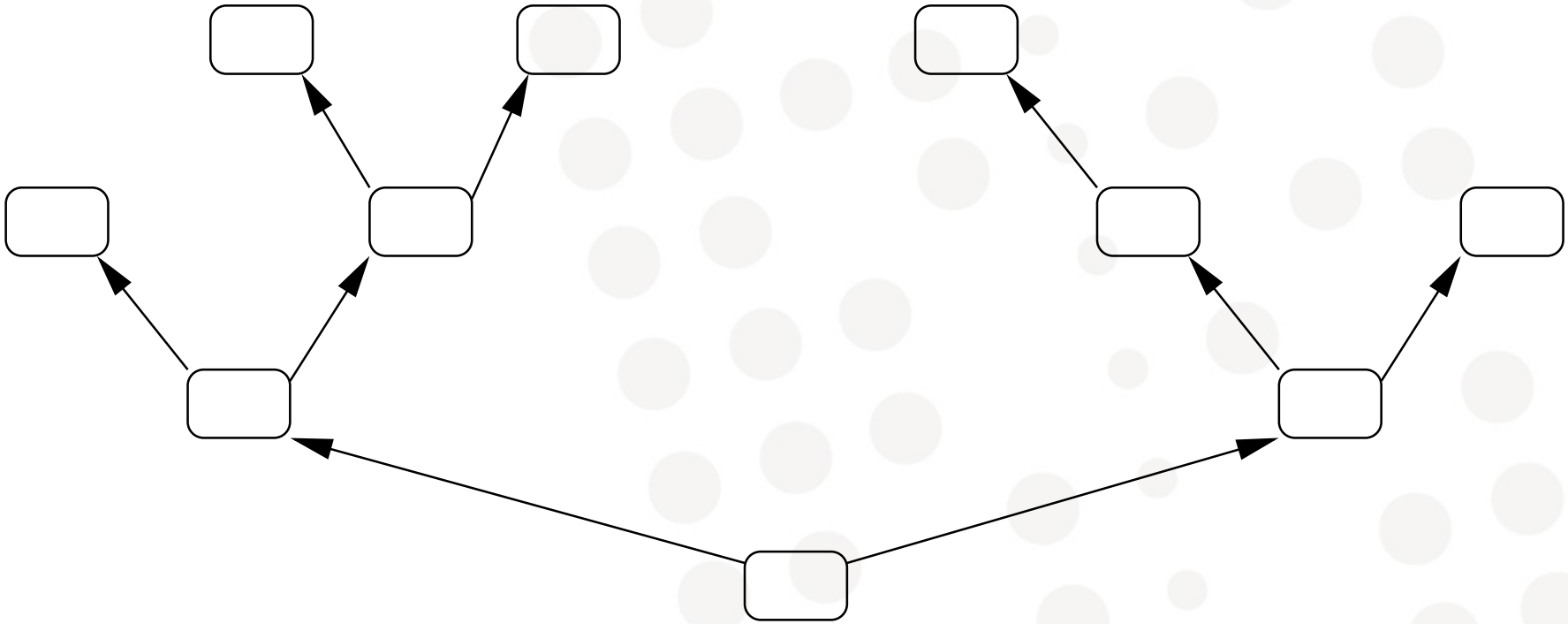
- Finne maks- eller minimumsverdier
- skrive ut / traversere hele lista
- gjøre endringer i hele datastrukturen for utvalgte objekter

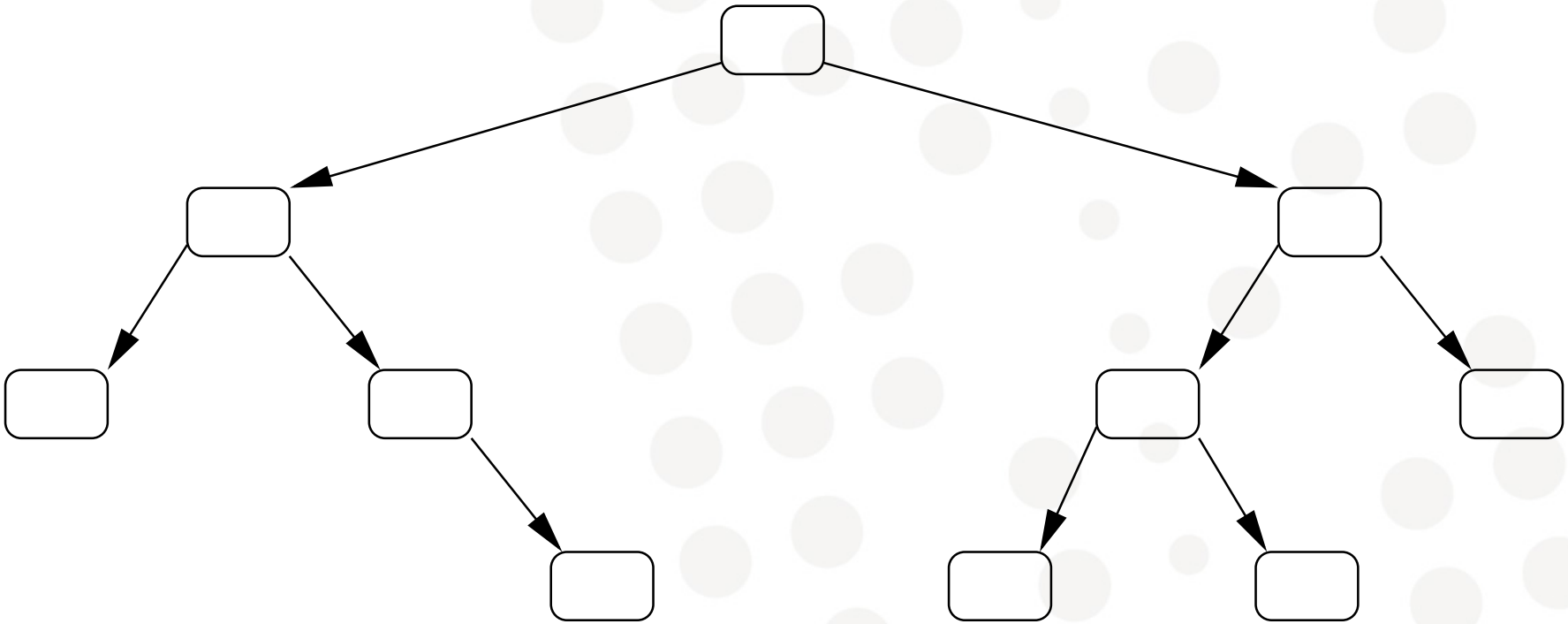
- hvorfor er lenkelister en rekursiv datastruktur?
- er sudokubrettet en lenkeliste?
- hvilke invariante tilstandspåstander i en lenkeliste må holde for at vi kan programmere rekursivt? Forslag:
  - alle listeelementer har en og bare en nestepeker
  - ingen nestepekere peker på førsteelementet
  - alle andre elementer blir pekt på av en og bare en nestepeker
  - en og bare en nestepeker er null (i siste element)

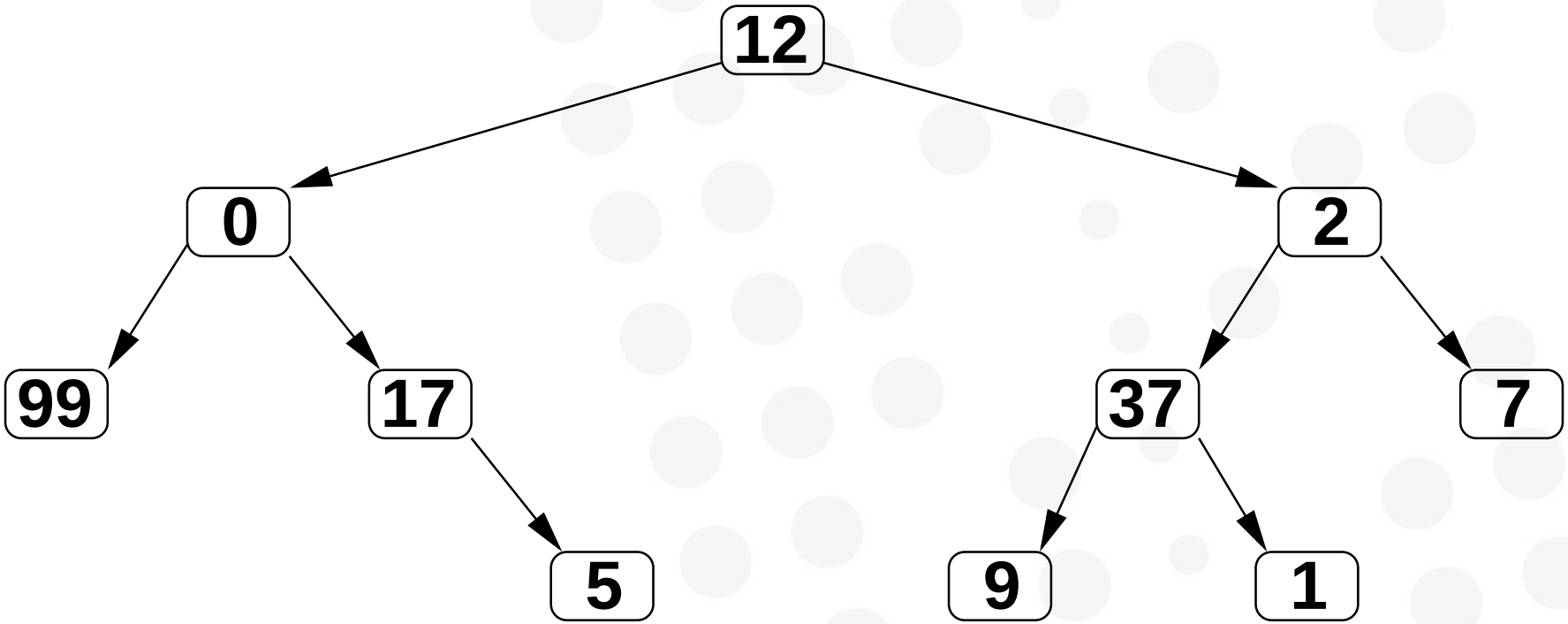
Hva er et binærtre?

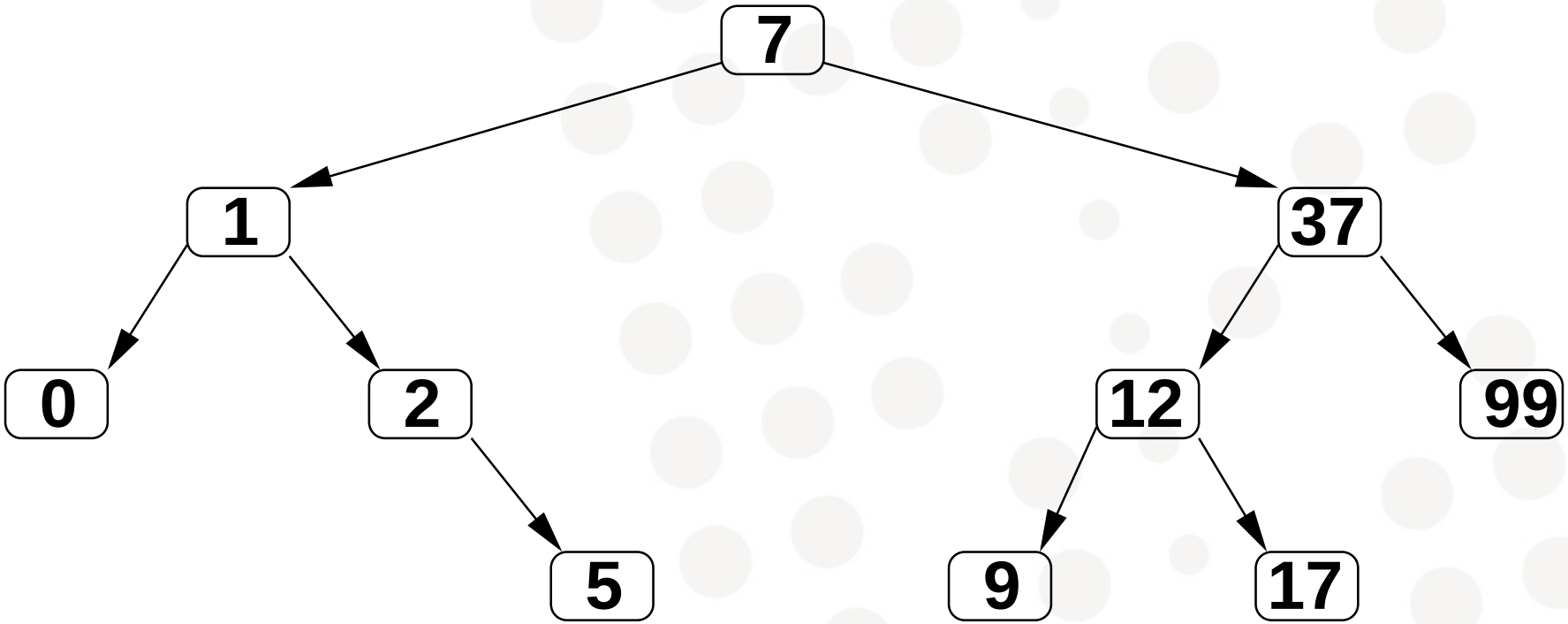


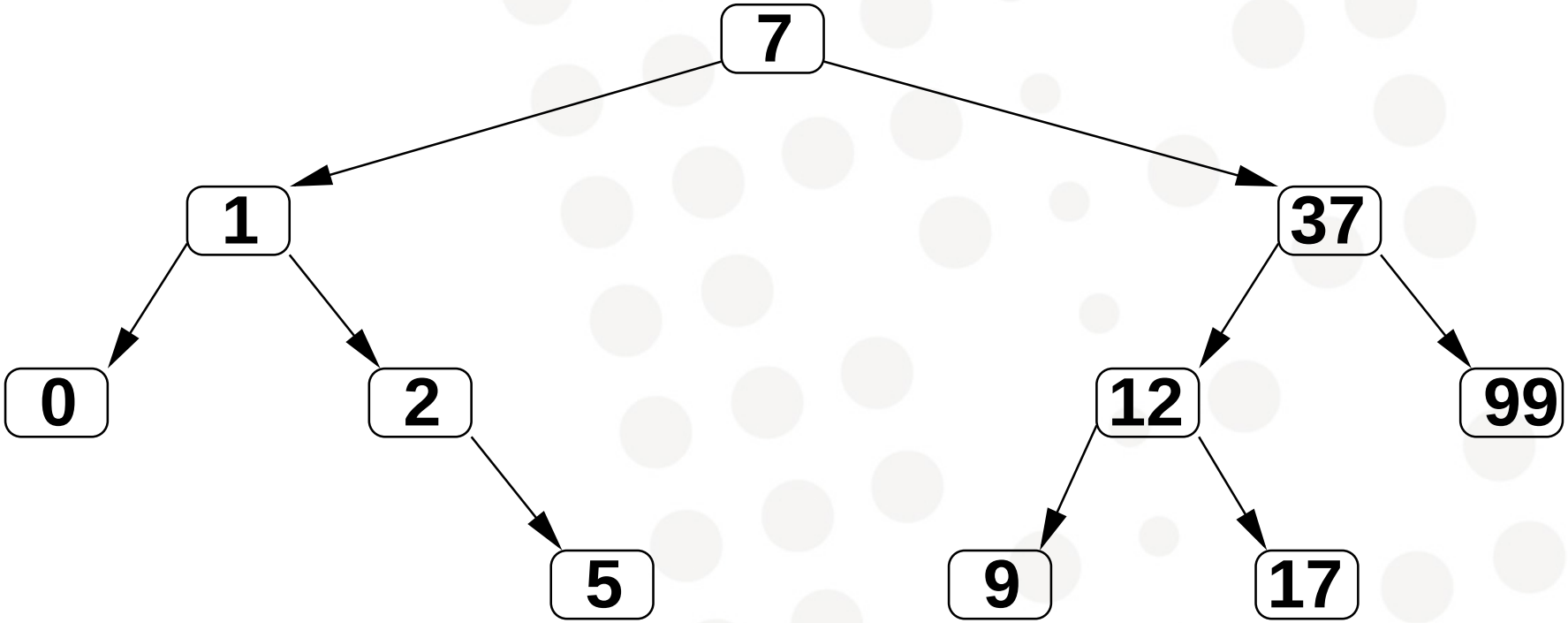




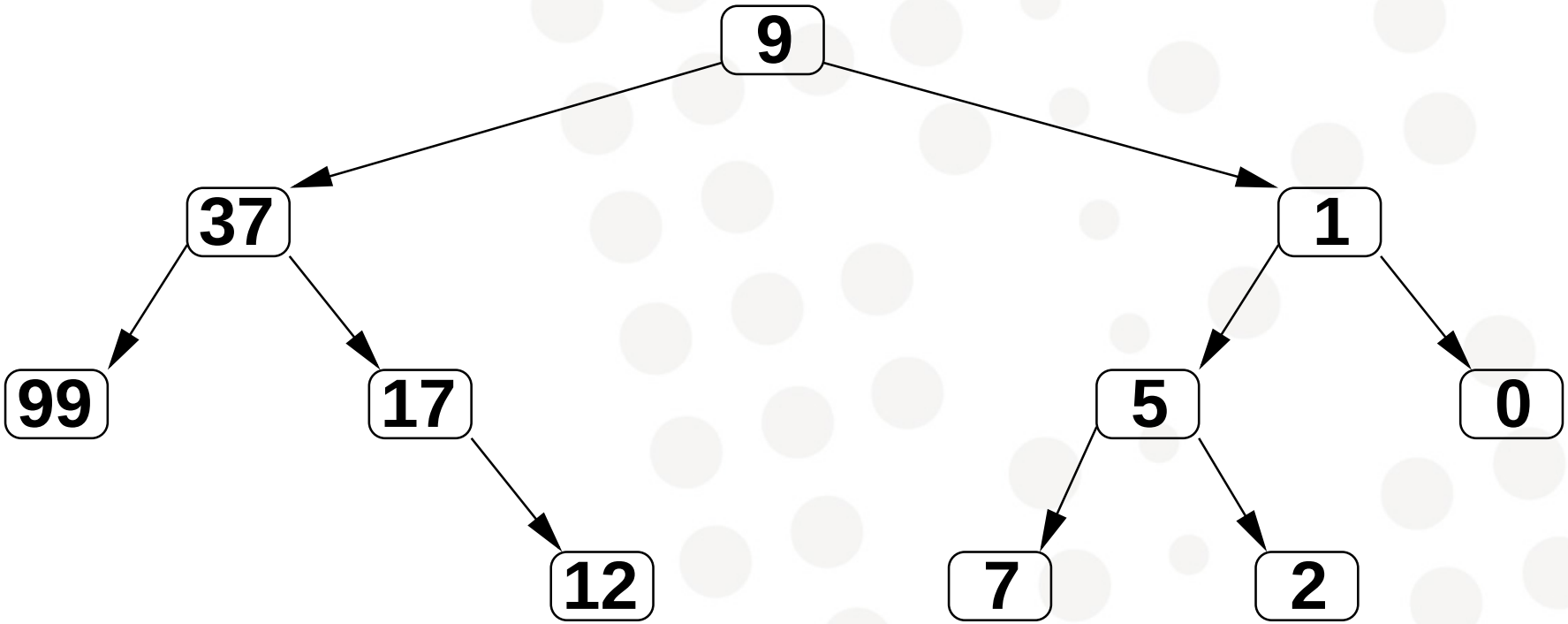


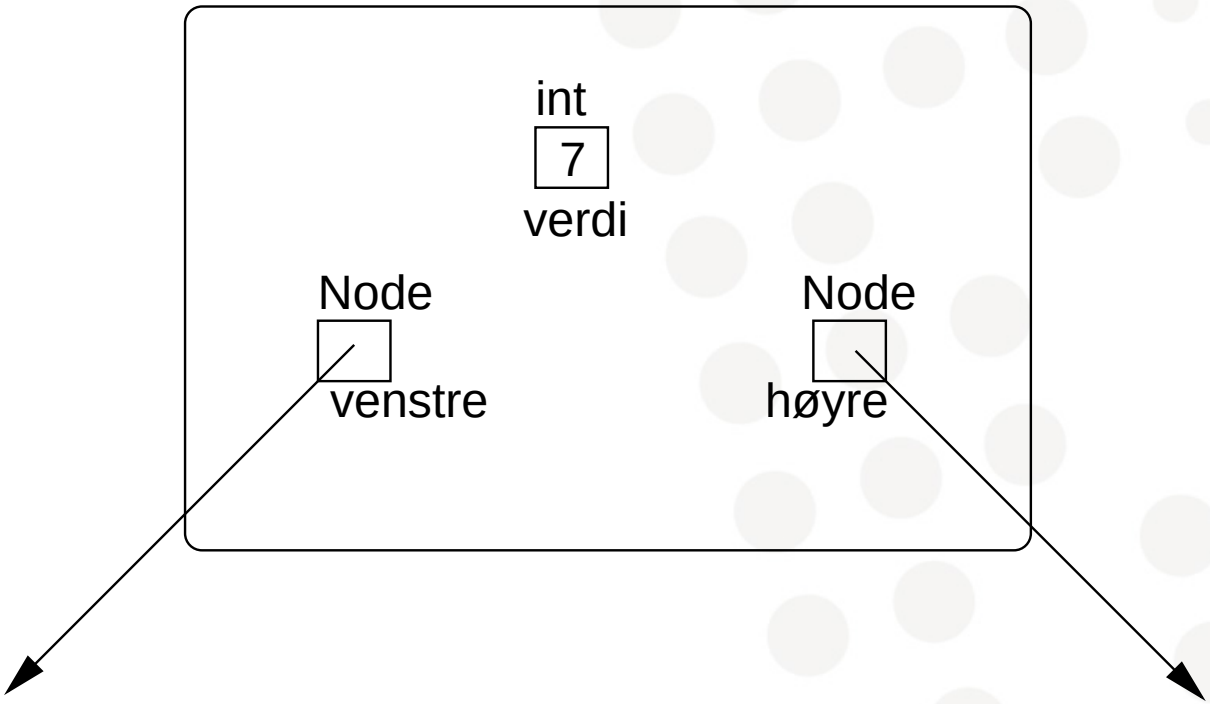

12
0
2
99
17
37
7
5
9
1

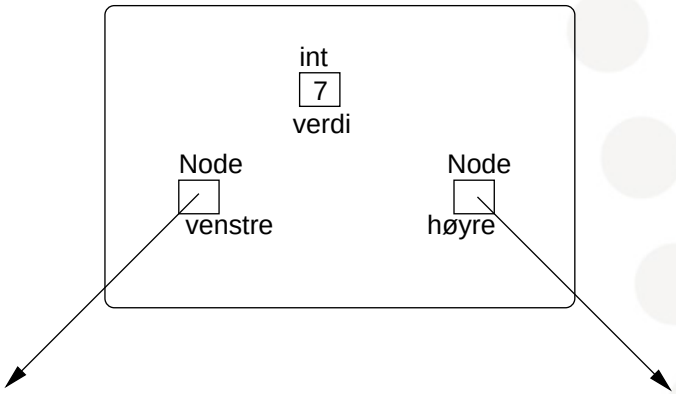




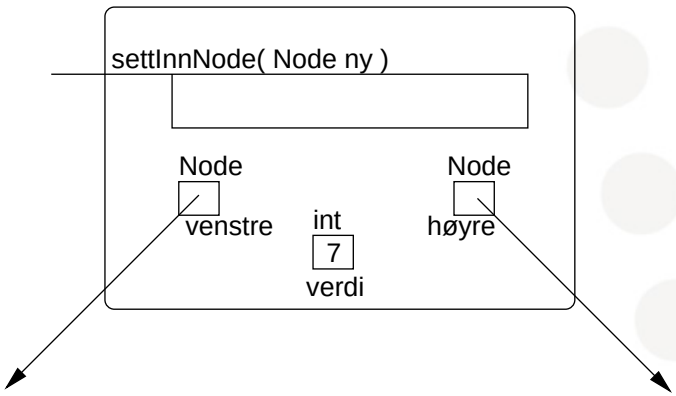
- rot
- subtre (barn)
- dybde (høyde)


9
37
1
99
17
5
0
12
7
2

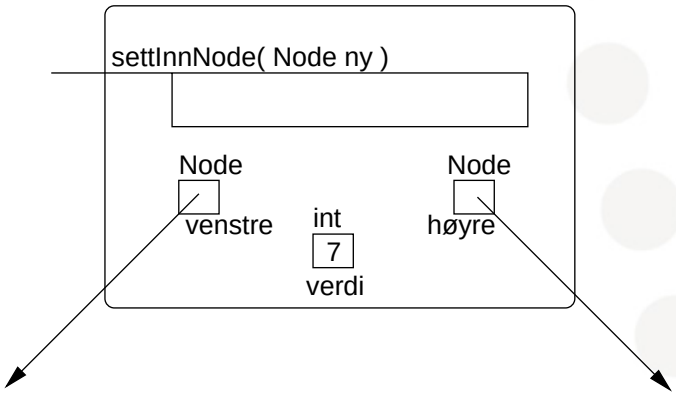

int
7
verdi
Node
venstre
Node
høyre



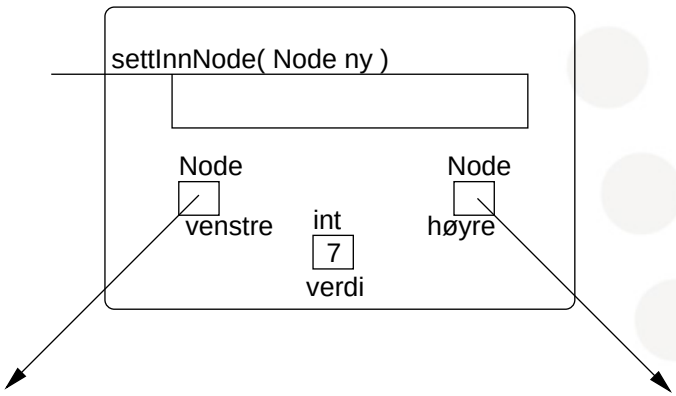

```
class Node {
  int verdi; // representerer sorteringskriteriet

  Node venstre; // peker til subtreet til venstre
  Node høyre; // peker til subtreet til høyre

  Node (int i) { verdi = i; }
}
```



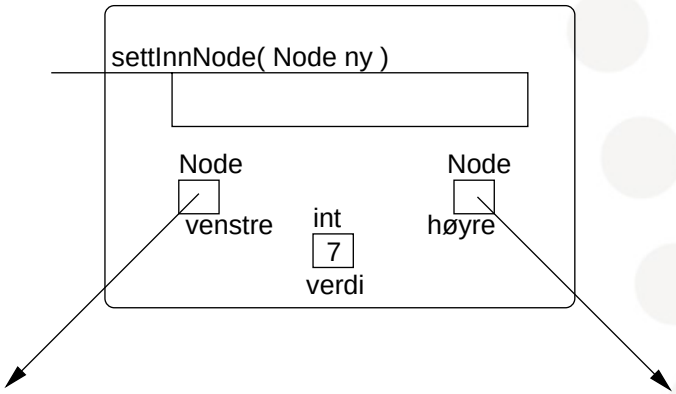

```
class Node {
  int verdi;
  Node venstre, høyre;
  ...

  public void settInnNode( Node ny ) {
    if (venstre == null) venstre = ny;
    else venstre.settInnNode(ny);
  }
}
```



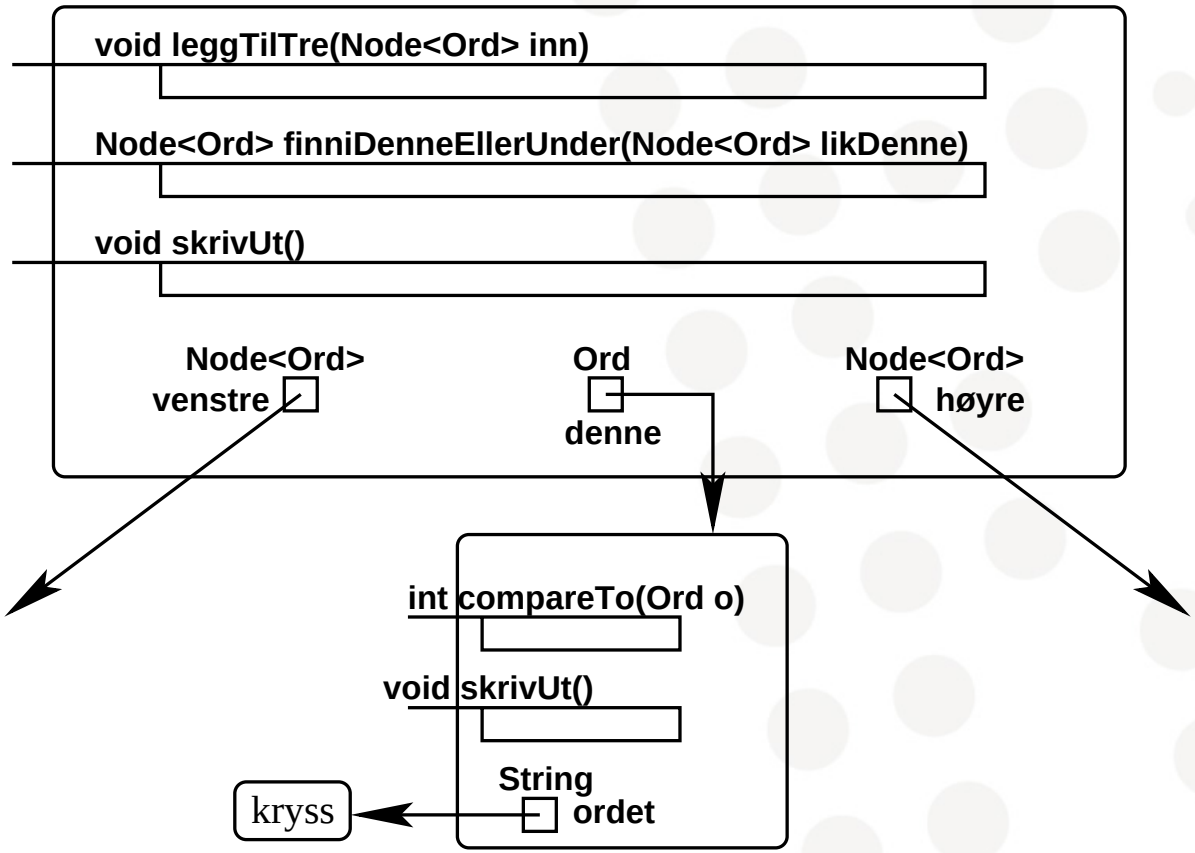

```
class Node {
  int verdi;
  Node venstre, høyre;
  ...

  public void settInnNode( Node ny ) {
    if (venstre == null) venstre = ny;
    else if (høyre == null) høyre = ny;
    else venstre.settInnNode(ny);
  }
}
```



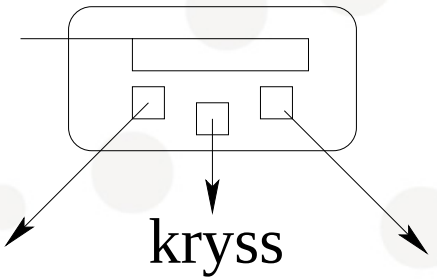

```
class Node {
  int verdi;
  Node venstre, høyre;
  ...
  public void settInnNode( Node ny ) {
    if (ny.verdi > verdi)
        <sett inn til høyre>
    else <sett inn til venstre>
  }
}
```




```
public void settInnNode( Node ny ) {
  if (ny.verdi > verdi)
     if (høyre == null) høyre = ny;
     else høyre.settInnNode(ny);
  else
     if (venstre == null) venstre = ny;
     else venstre.settInnNode(ny);
}
```



Sterkt forenklet:



Klassen Node<T>

```
class Node <T extends BTNodeRolle <T>> {
    Node<T> venstre, høyre;
    T denne;

    Node (T t){denne = t;}

    void leggTilTre(Node <T> inn)
    Node<T> finniDenneEllerUnder (Node <T> likDenne)
    void skrivUt()
}
```

```
interface  BTNodeRolle <T>{
      int compareTo(T e);
      void skrivUt();
}

class Node <T extends BTNodeRolle <T>> {
    Node<T> venstre, høyre;
    T denne;
    Node (T t){denne = t;}

    void leggTilTre(Node <T> inn)
    Node<T> finniDenneEllerUnder (Node <T> likDenne)
    void skrivUt()
}
```

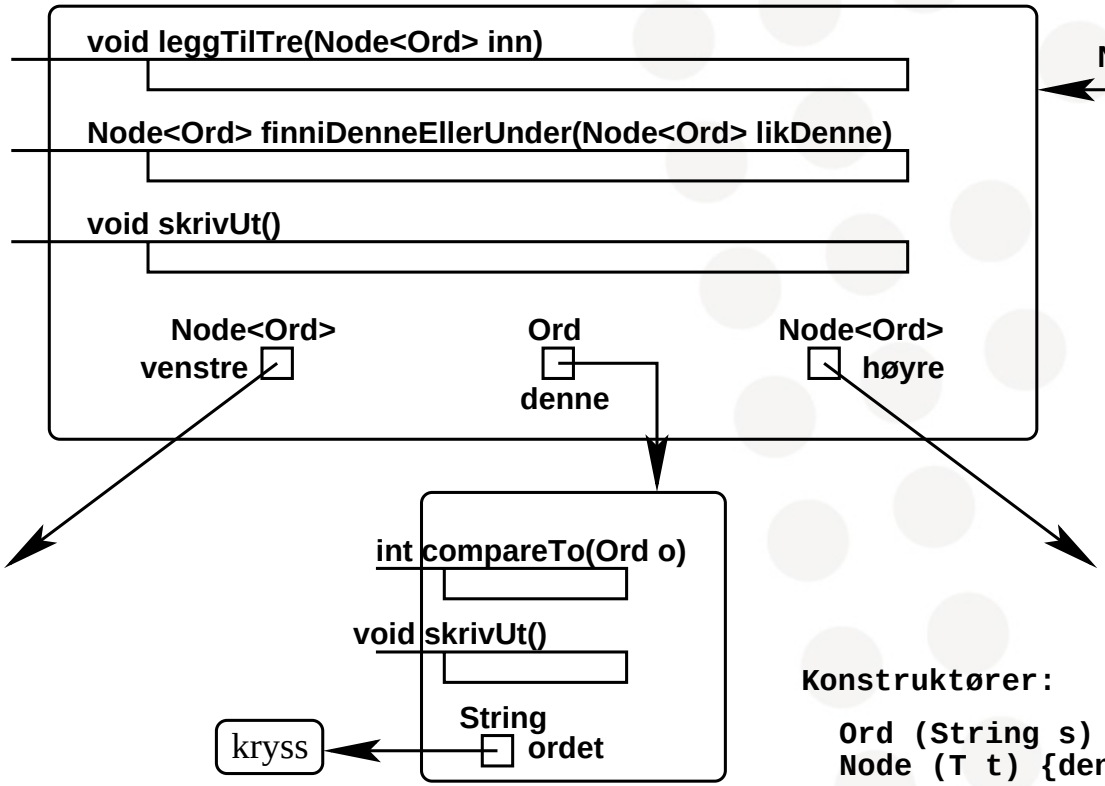
```
class Ord implements BTNodeRolle <Ord> {
    private String ordet;
    Ord (String s) { ordet = s; }

    public int compareTo(Ord o)
    public void skrivUt()
}

public class Eksempel1{
    public static void main( String[] args ) {
    Node<Ord> rot = new Node<Ord> (new Ord(``kryss''));
}
```

Klassen Ord

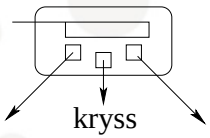
```
class Ord implements BTNodeRolle <Ord> {
    private String ordet;
    Ord (String s) { ordet = s; }

    public int compareTo(Ord o){
        return ordet.compareToIgnoreCase(o.ordet);
    }

    public void skrivUt() {
        System.out.print(ordet+"␣");
    }
}
```

Et Node<Ord>-objekt

Tilstand i programmet etter setningen
Node<Ord> rot = new Node<Ord> (new Ord(kryss"));



Sterkt forenklet:




```
Konstruktører:

  Ord (String s) {ordet = s;}
  Node (T t) {denne = t;}
```

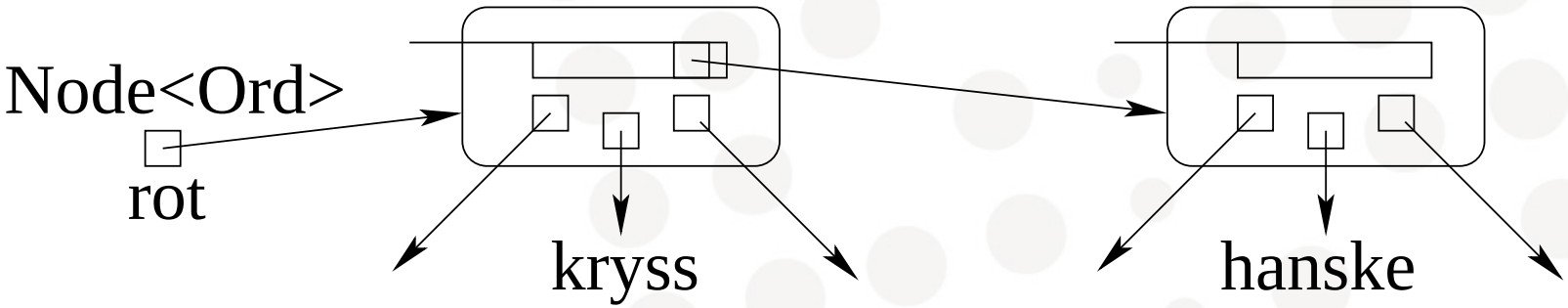
Fra main-metoden

```
Node<Ord> rot = new Node<Ord> (new Ord("kryss"));
rot.leggTilTre(new Node<Ord> (new Ord("hanske")));
rot.leggTilTre(new Node<Ord> (new Ord("angre")));
rot.leggTilTre(new Node<Ord> (new Ord("nikkel")));
rot.leggTilTre(new Node<Ord> (new Ord("ansvar")));
rot.leggTilTre(new Node<Ord> (new Ord("juletre")));
rot.leggTilTre(new Node<Ord> (new Ord("trøffel")));
rot.leggTilTre(new Node<Ord> (new Ord("hylle")));
rot.leggTilTre(new Node<Ord> (new Ord("adrenalin")));
rot.leggTilTre(new Node<Ord> (new Ord("laser")));
```

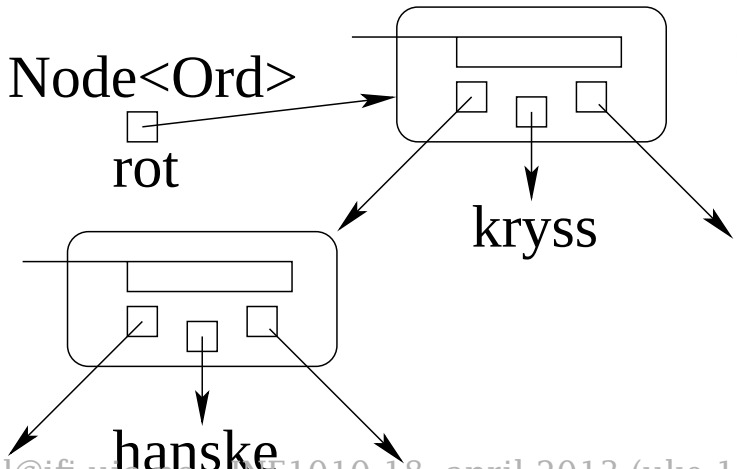
Settinnmetoden i Node<T>

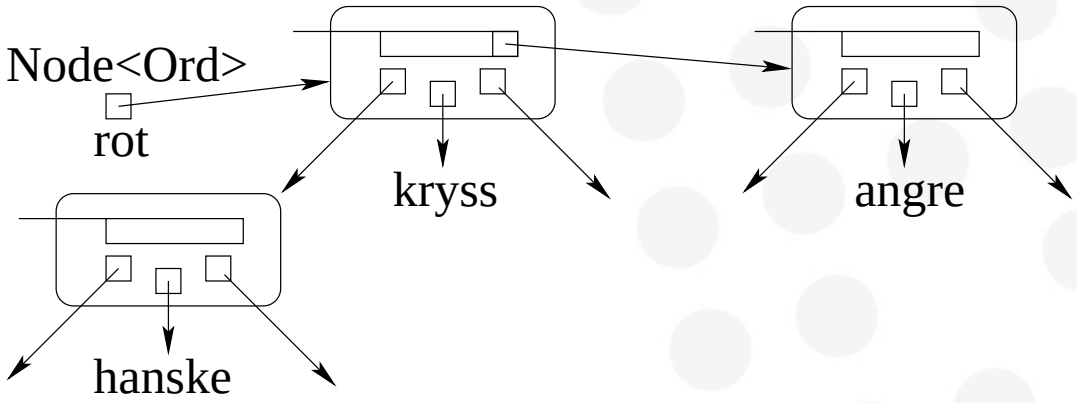
```
void leggTilTre(Node <T> inn) {
    int smnlgn = denne.compareTo( inn.denne );
    if ( smnlgn < 0 )
        if (høyre == null) høyre = inn;
        else høyre.leggTilTre(inn);
    else
        if (venstre == null) venstre = inn;
        else venstre.leggTilTre(inn);
}
```

Hva skjer med like objekter?

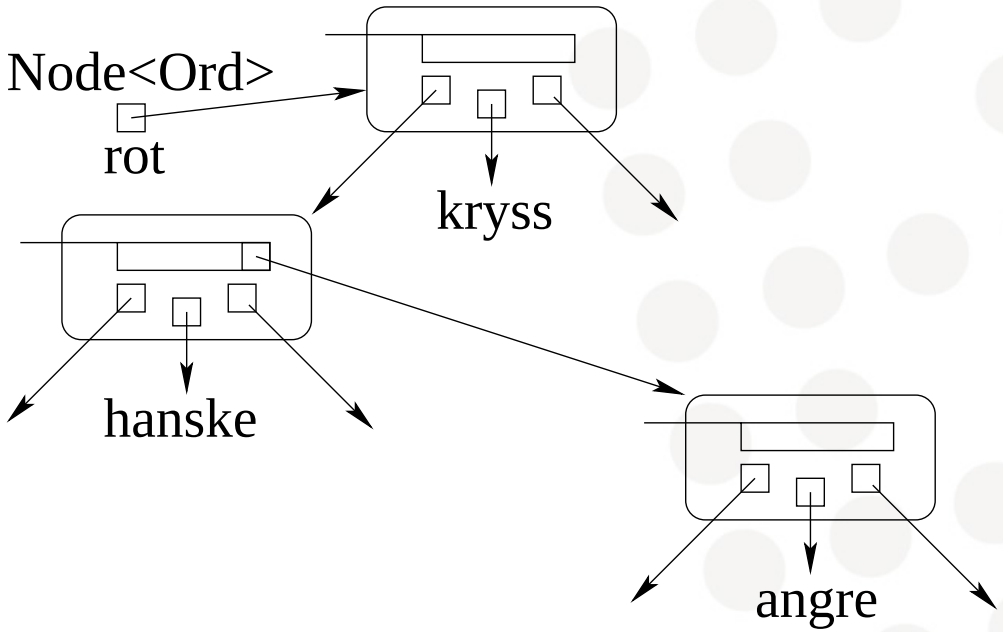


Datastruktur ved kall på settinn-metoden (over) og resultat (under)

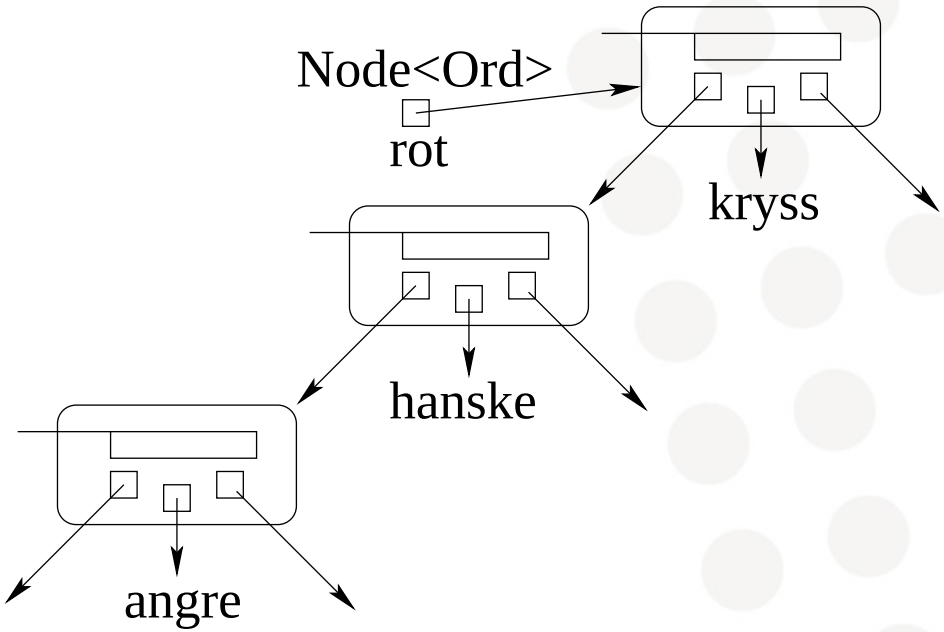




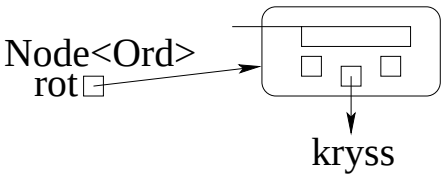

```
void leggTilTre(Node <T> inn) {
    int smnlgn = denne.compareTo( inn.denne );
    if ( smnlgn < 0 )
        if (høyre == null) høyre = inn;
        else høyre.leggTilTre(inn);
    else
        if (venstre == null) venstre = inn;
        else venstre.leggTilTre(inn);
}
```



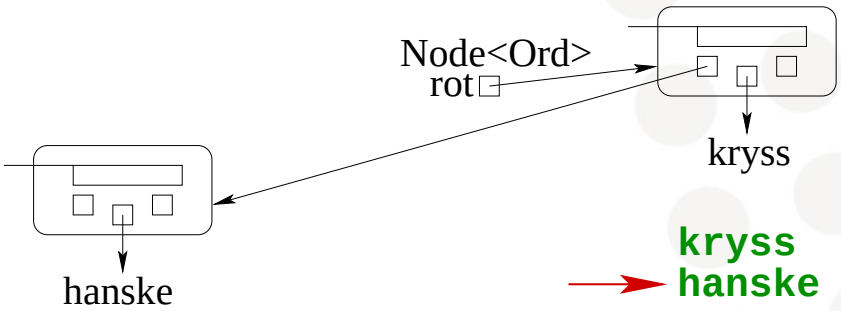

```
void leggTilTre(Node <T> inn) {
    int smnlgn = denne.compareTo( inn.denne );
    if ( smnlgn < 0 )
        if (høyre == null) høyre = inn;
        else høyre.leggTilTre(inn);
    else
        if (venstre == null) venstre = inn;
        else venstre.leggTilTre(inn);
}
```



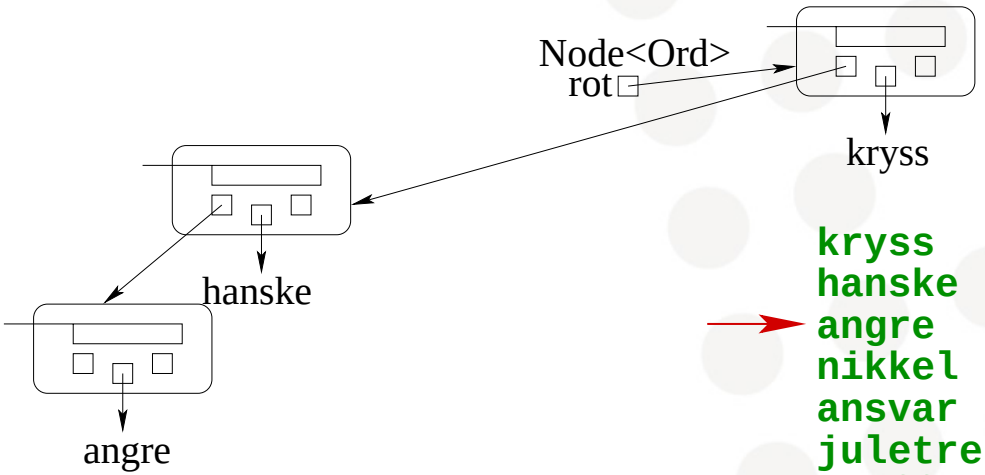

```
void leggTilTre(Node <T> inn) {
    int smnlgn = denne.compareTo( inn.denne );
    if ( smnlgn < 0 )
        if (høyre == null) høyre = inn;
        else høyre.leggTilTre(inn);
    else
        if (venstre == null) venstre = inn;
        else venstre.leggTilTre(inn);
}
```



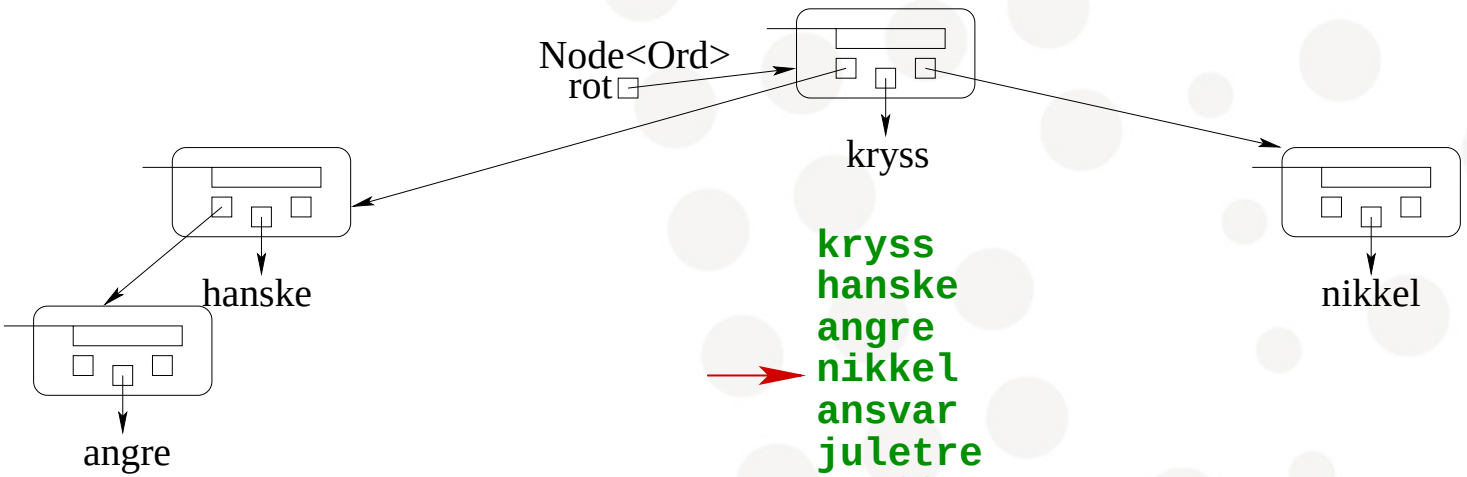

```
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser
```



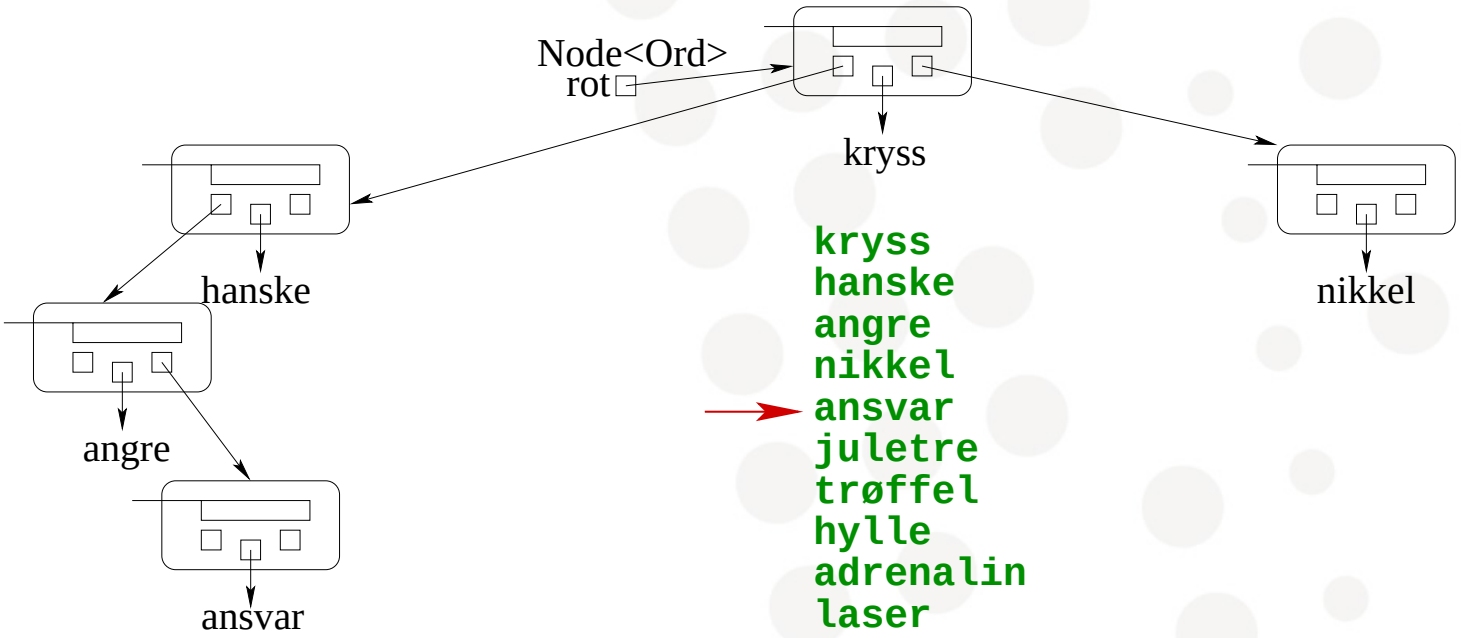

```
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser
```

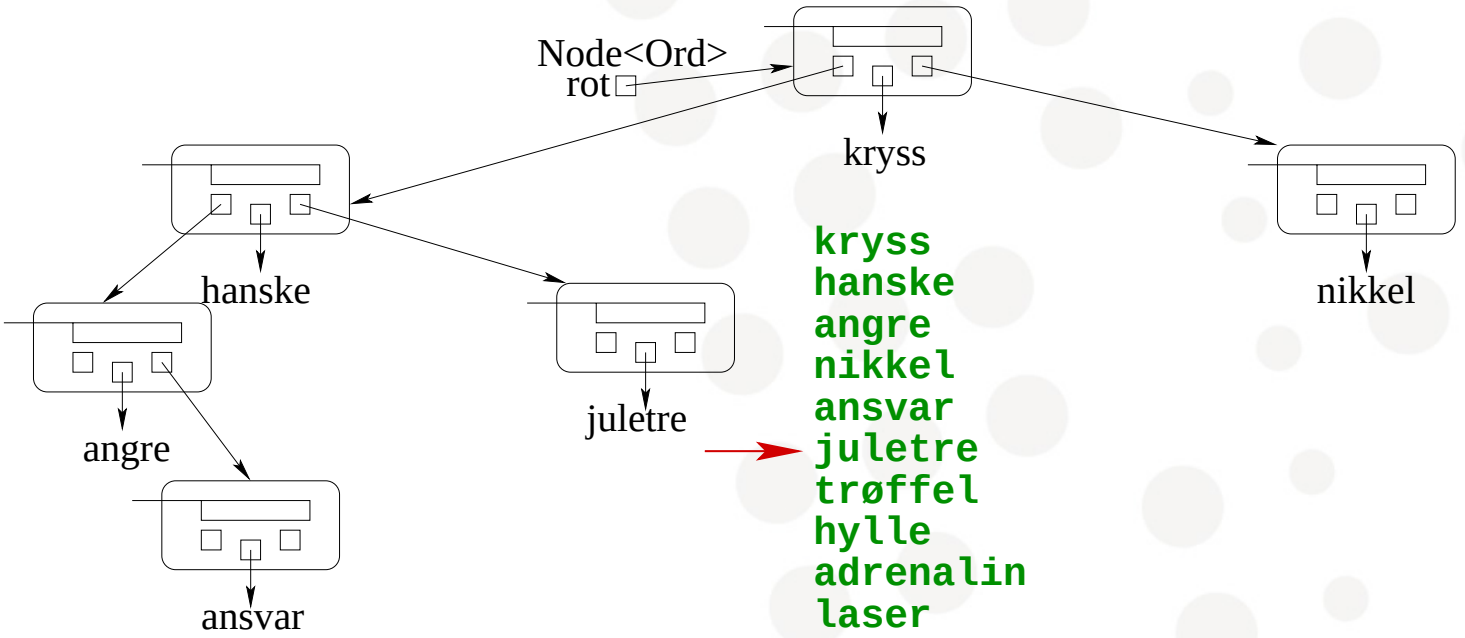


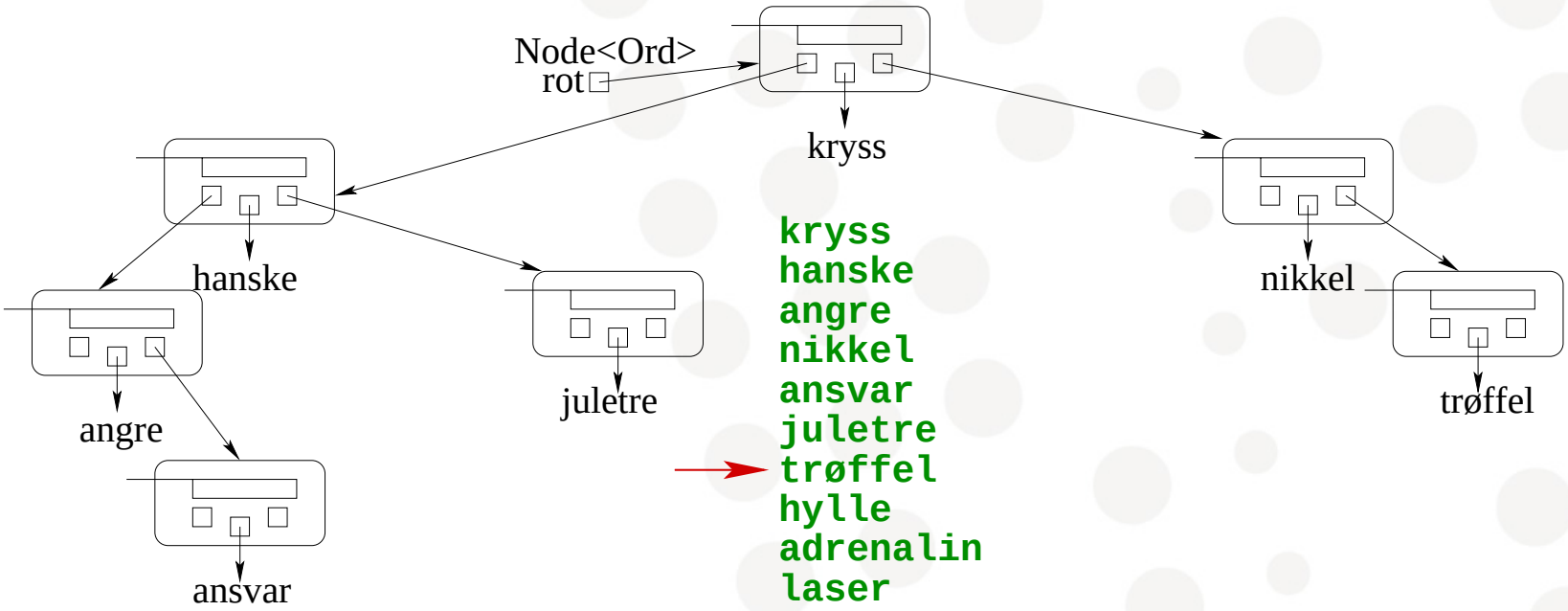

```
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser
```

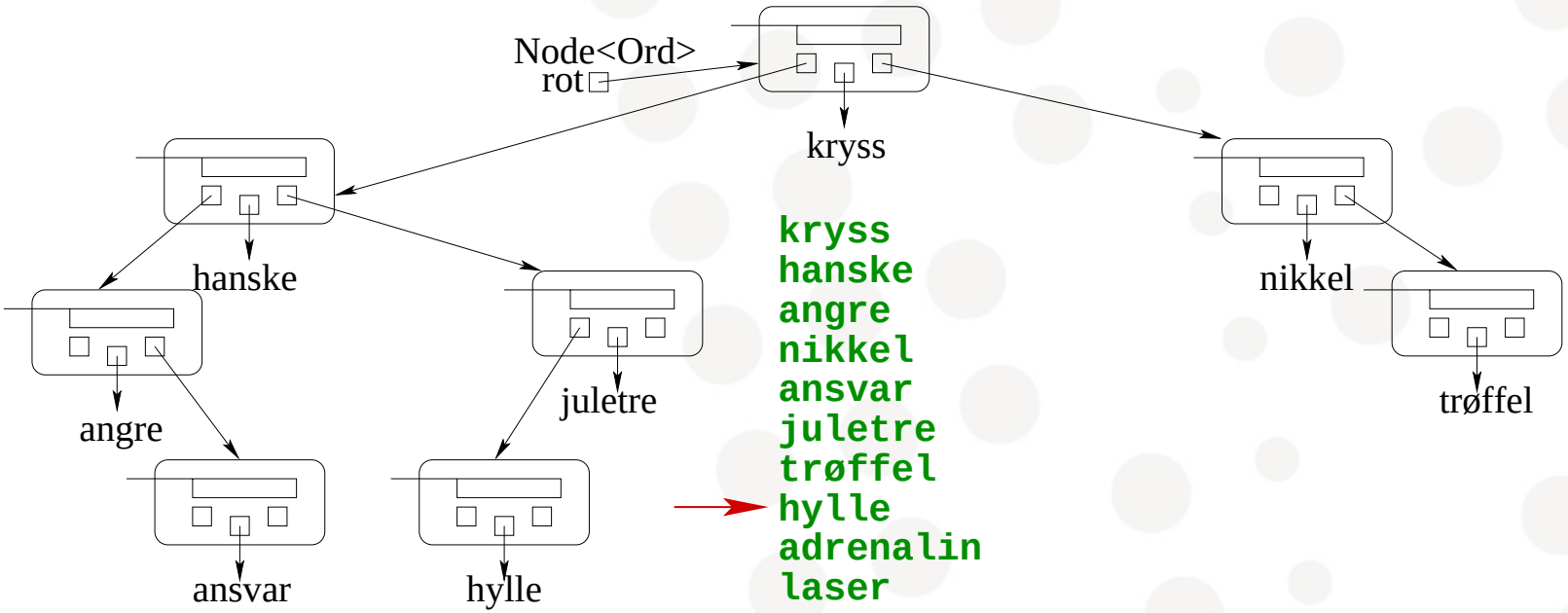


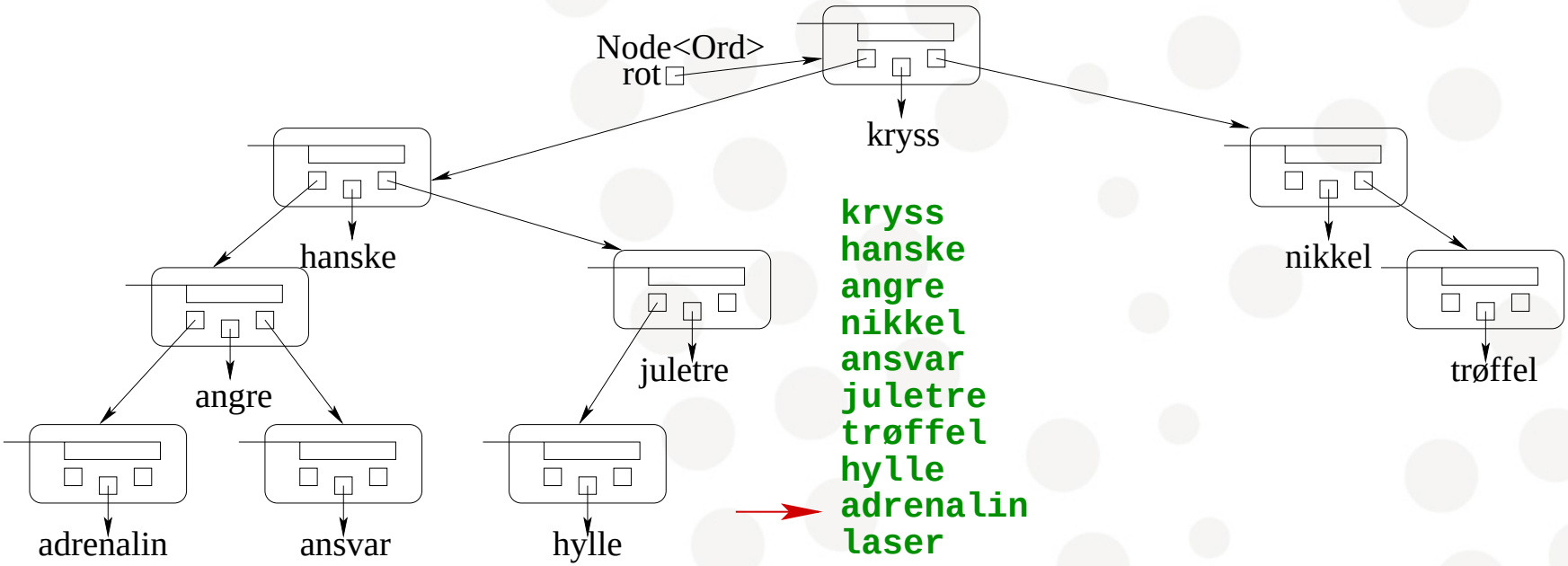

```
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser
```

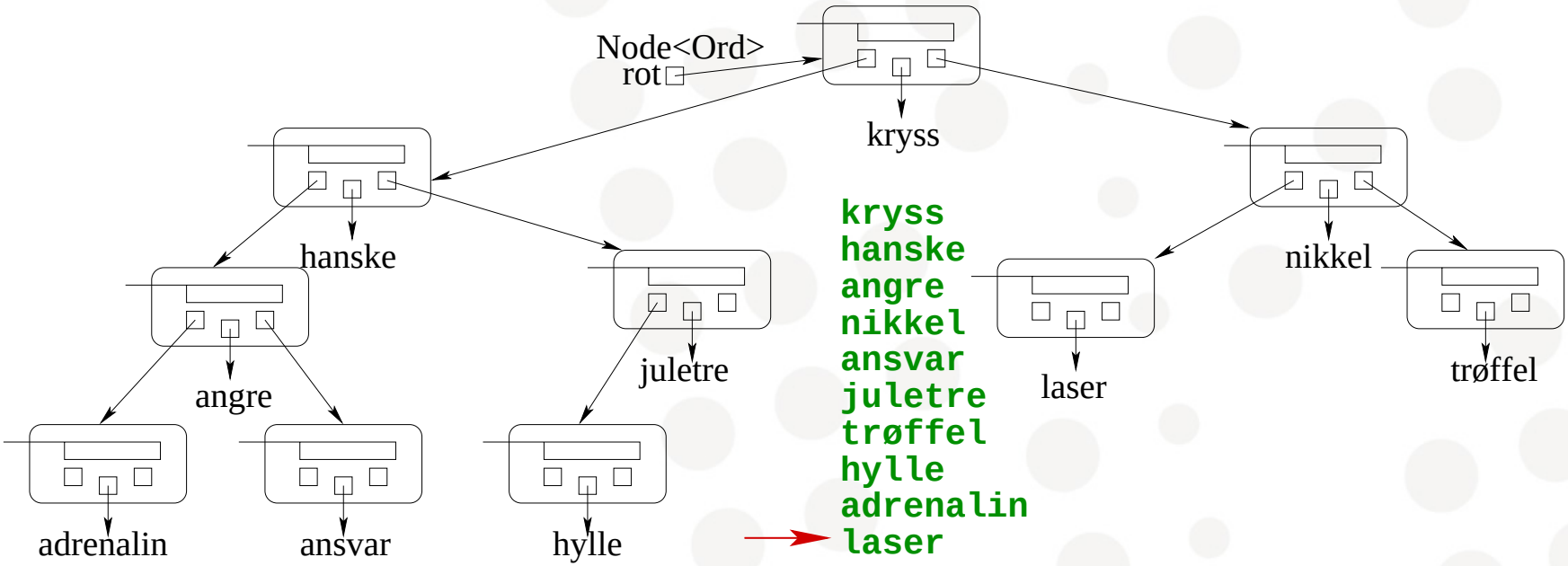

Node<Ord>
rot
kryss
hanske
angre
ansvar
nikkel
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser


Node<Ord>
rot
kryss
hanske
nikkel
angre
juletre
ansvar
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser



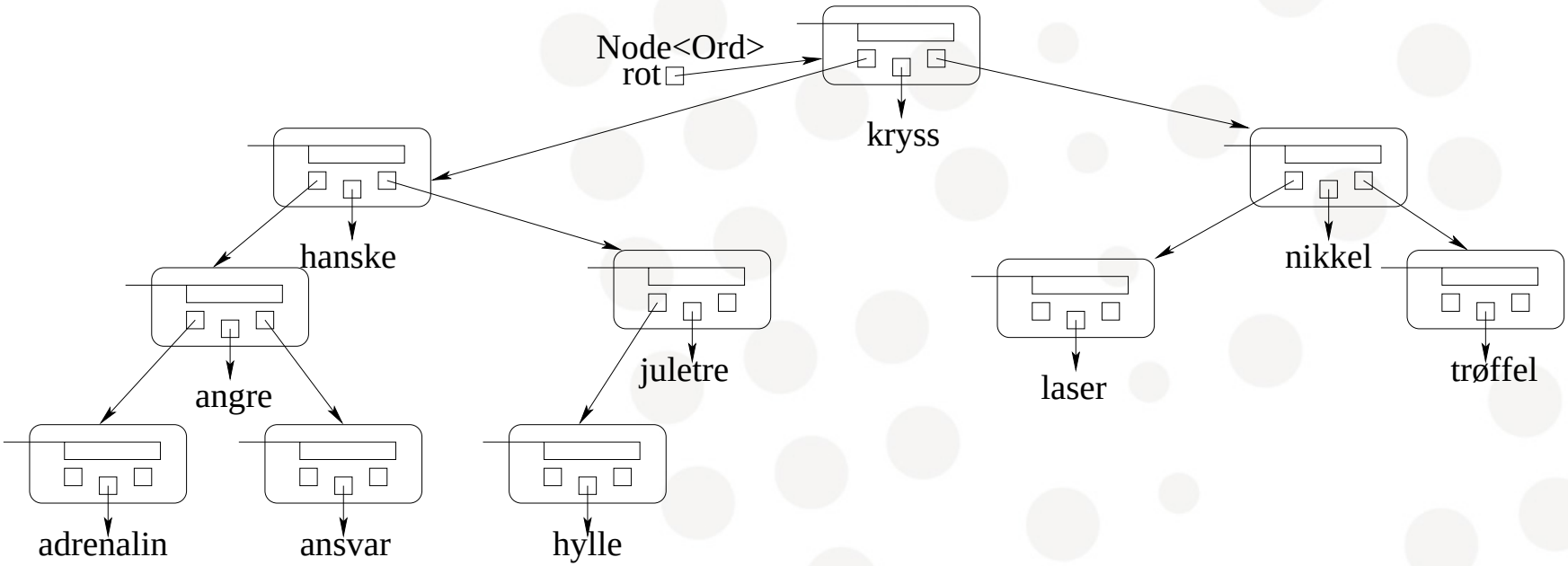

Node<Ord>
rot
kryss
hanske
nikkel
angre
juletre
trøffel
ansvar
hylle
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser



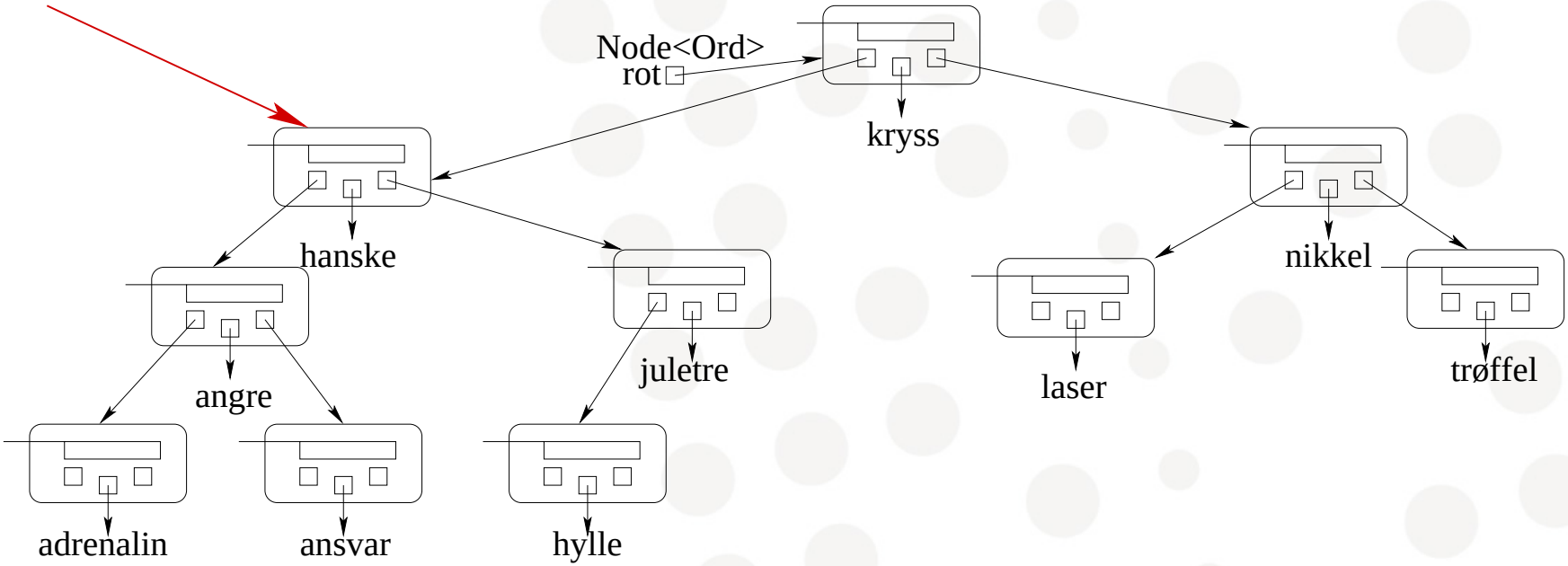

Node<Ord>
rot
kryss
hanske
nikkel
angre
juletre
laser
trøffel
adrenalin
ansvar
hylle
kryss
hanske
angre
nikkel
ansvar
juletre
trøffel
hylle
adrenalin
laser

Finnmetoden i Node<T>

```
Node<T> finniDenneEllerUnder (Node <T> likDenne){
    Node<T> fantDette;
    int smnlgn = denne.compareTo( likDenne.denne );

    if ( smnlgn < 0 )
        if (høyre == null) fantDette = null;
        else fantDette = høyre.finniDenneEllerUnder(likDenne);
    else if ( smnlgn > 0 )
        if (venstre == null) fantDette = null;
        else fantDette = venstre.finniDenneEllerUnder(likDenne);
    else // smnlgn == 0, dvs. dette er noden det letes etter
        fantDette = this;

    return fantDette;
}
```




```
Node<Ord> funnet =
    rot.finniDenneEllerUnder(new Node<Ord> (new Ord("hanske")));
```




```
funnet.skrivUt();
```

Skrivutmetoden i Node<T>

```
void skrivUt(){
    if (venstre != null) venstre.skrivUt();
    denne.skrivUt();
    if (høyre != null) høyre.skrivUt();
}
```

NB! Rekkefølgen på skrivUt-kallene vil forandre rekkefølgen. Metoden som er vist skriver ut objektene i alfabetisk stigende rekkefølge. Hvorfor?

Hva må du kunne om binære søketrær i INF1010

Lage programmer som

1. bygger opp et binært søketre (sette inn objekter)
2. finner igjen et objekt i et binært søketre
3. traverserer hele treet (f.eks. skriver ut alle objektene)
4. gjør om et binært søketre til en sortert liste
5. gjør om en usortert liste til et binært søketre (samme som 1.)

Du skal også kunne gjøre rede for tilstandspåstandene som ligger til grunn for at en datastruktur skal være et binært søketre.

Vil du vite mer om binære søketrær, kan denne wikipediaartikkelen være et greit utgangspunkt: http://en.wikipedia.org/wiki/Binary_search_tree

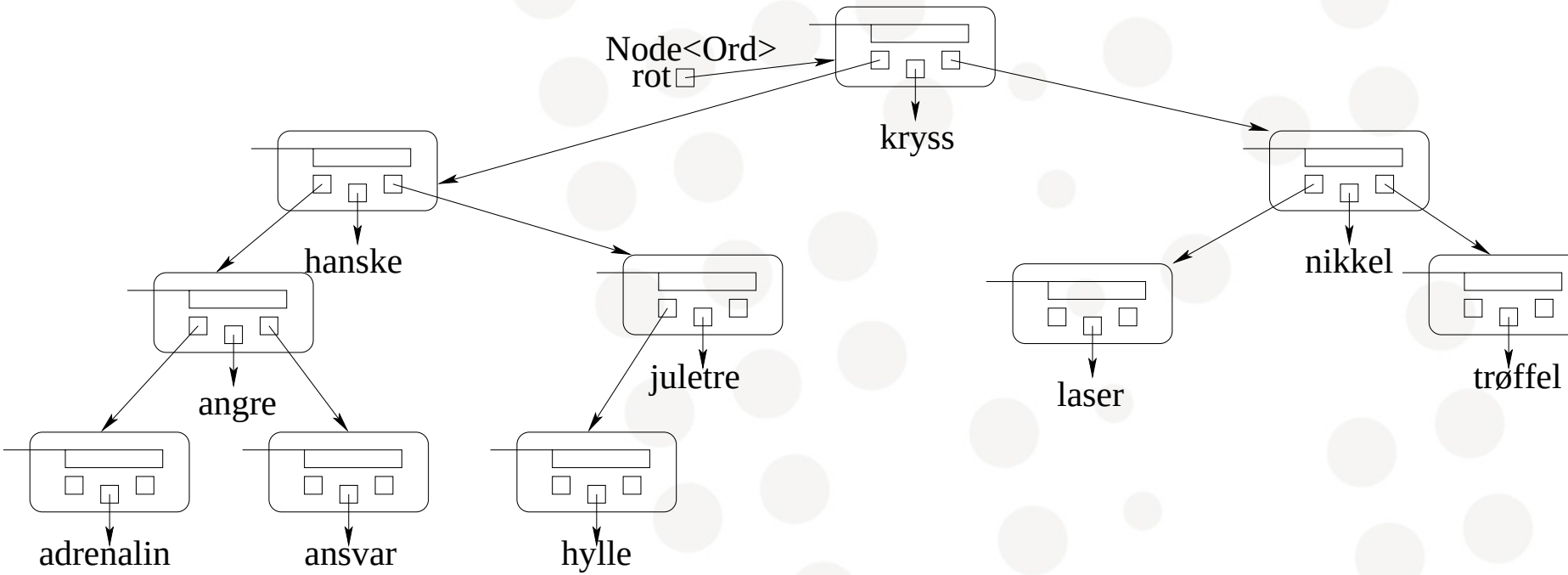
Tilstandspåstander

- et binært søketre (BST) består av 0 til n noder (noder)

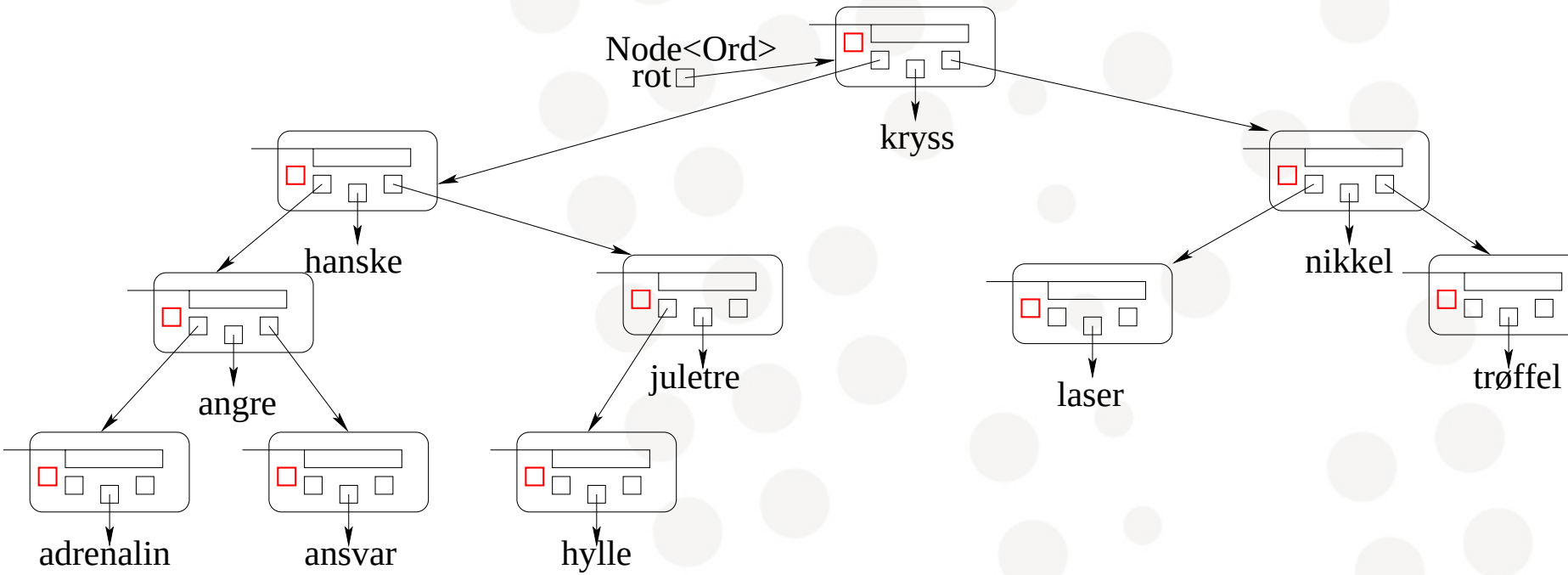
For alle noder i et BST gjelder:

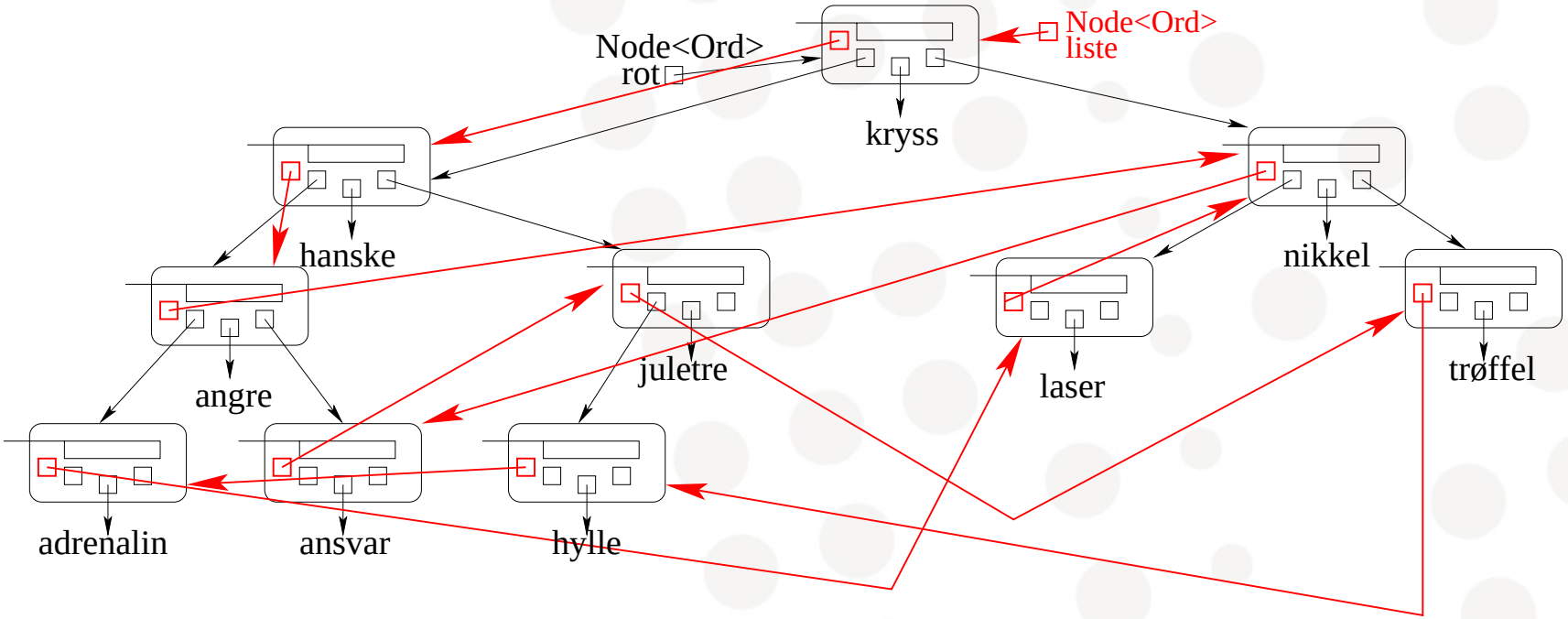
- en node har en konstant verdi og en compareTo-metode som er veldefinert når noden sammenlignes med andre noder
- en node har pekere til to BST (subtrær), v og h.
- alle nodene i venstre subtre (hvis ikke tomt) har verdier som er mindre enn eller lik nodens verdi basert på returverdien til nodens compareTo-metode.
- alle nodene i høyre subtre har verdier som er større enn eller lik nodens verdi basert på returverdien til nodens compareTo-metode.

Fra binærtre til liste



Liste er en enklere struktur å lage en iterator over.


Node<Ord>
rot
kryss
hanske
nikkel
angre
juletre
laser
trøffel
adrenalin
ansvar
hylle


Node<Ord>
rot
Node<Ord>
liste
kryss
hanske
nikkel
angre
juletre
laser
trøffel
adrenalin
ansvar
hylle