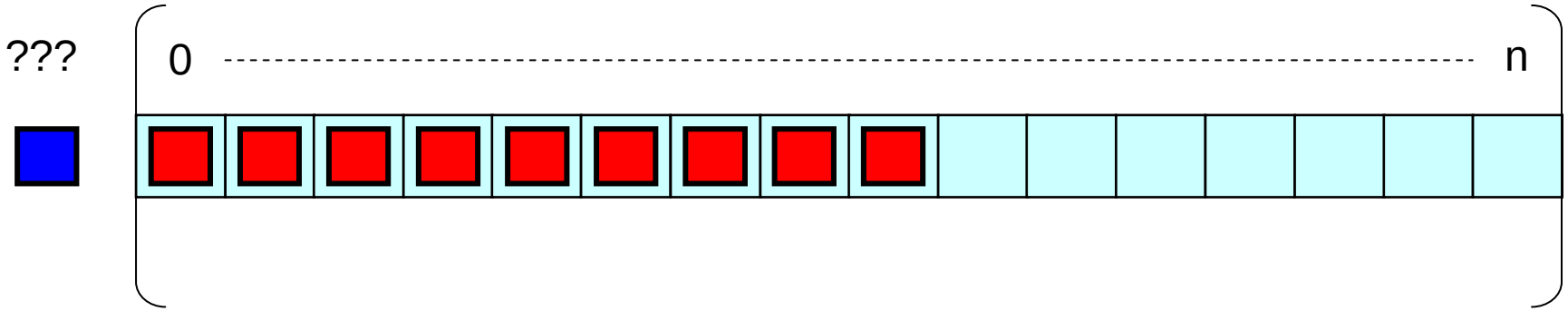
Algoritmer och data strukturer
-Länkade listor, kap 17-

För utveckling av verksamhet, produkter och livskvalitet.

HÖGSKOLAN HALMSTAD

Statisk minnesallokering

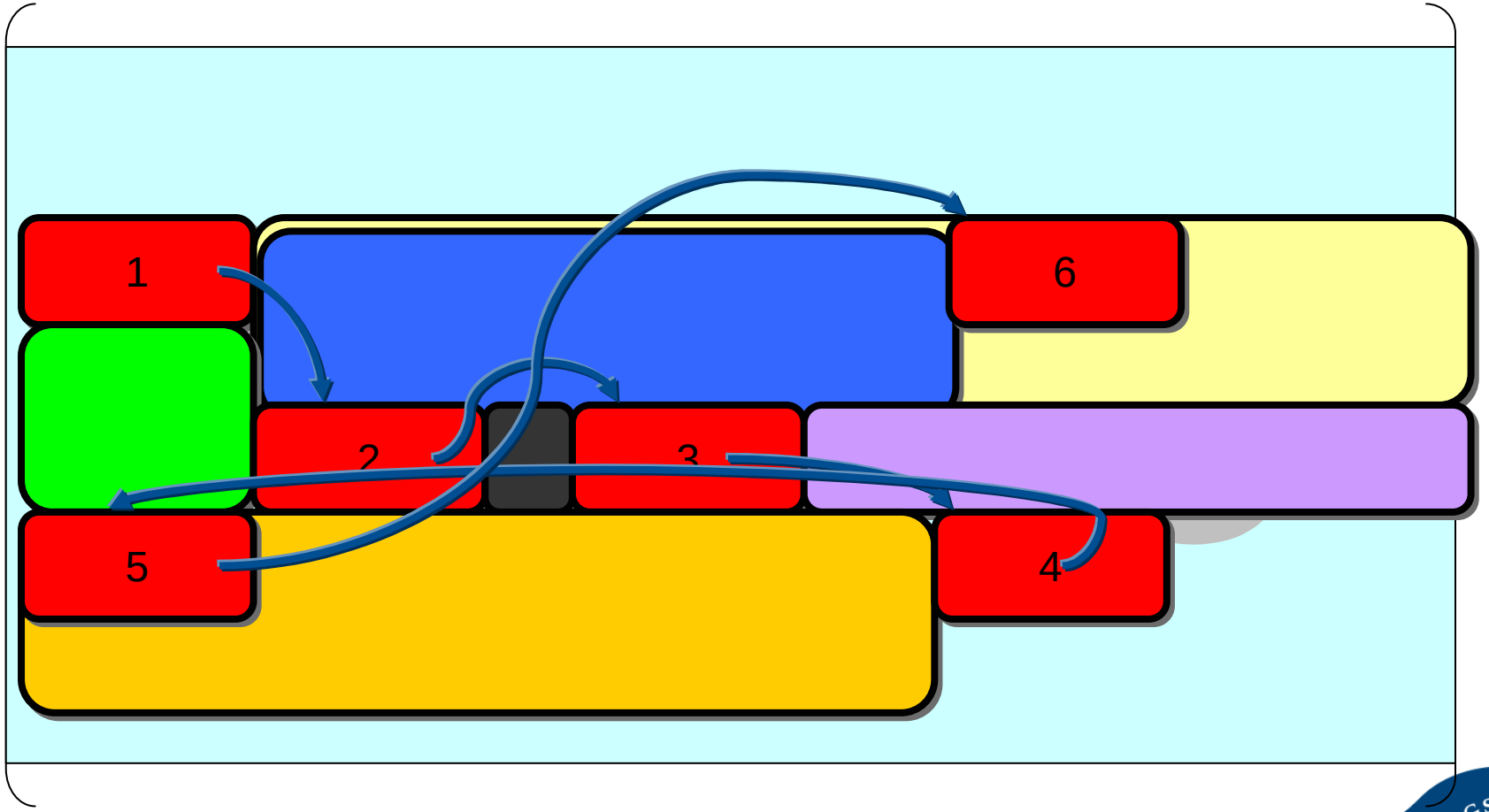
- Statiska minnesstrukturer (Arrayer)
  - För stora / För små
  - ”Jobbiga” *Insert(index i)* / *Remove(index i)* operationer



Dynamisk minnesallokering

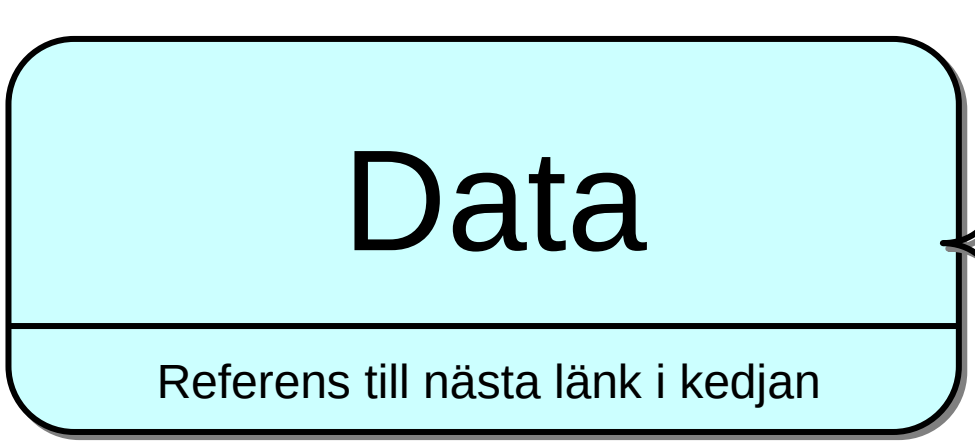
- Separata *”noder”*
  - Allokeras vid behov
  - Avallokeras när de inte längre behövs
  - Håller information om:
    - Nodens data
    - Positionsinformation relativt den totala datastrukturen

En kedja av data



En länk (nod) i kedjan

- Olika sorters listor
  - *Enkellänkade* listor
  - *Dubbellänkade* listor




```
public class ListNode
{
   public Object data;
   public ListNode next;
}
```

- Första länken är MYCKET viktig!

Klassen ListNode / av Object

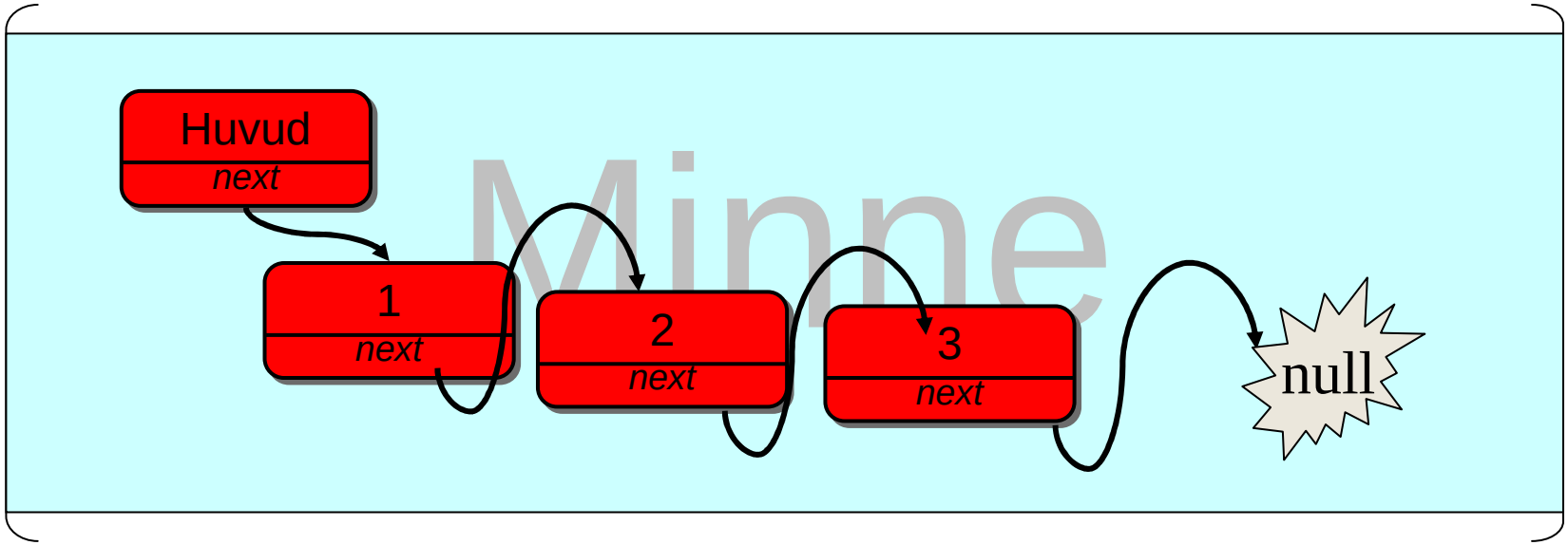
```
class ListNode
{
    public Object   element;
    public ListNode next;

      // Konstruktor

   public ListNode( Object theElement )
    {
        this( theElement, null );
    }

    public ListNode( Object theElement, ListNode n )
    {
        element = theElement;
        next    = n;
    }

}
```

Klassen LinkedList

```
class LinkedList
{
 private ListNode header;

 public LinkedList ( )
    {
        header = new ListNode( null );
    }
// andra metoder
```

Enkellänkade listor

- Varje länk har en referens till nästa länk i listan
- Kräver kännedom om *”föregående länk”* vid insert() / remove() operationer



Huvud / Svans

- När är listan slut?
  - Sätt sista länkens next = `null`
  - Testa på `null` under traverseringen

```
public void print()
{
    ListNode temp=header.next;
    while( temp!=null ){
    System.out.println(temp.element);
    temp=temp.next;

}
```

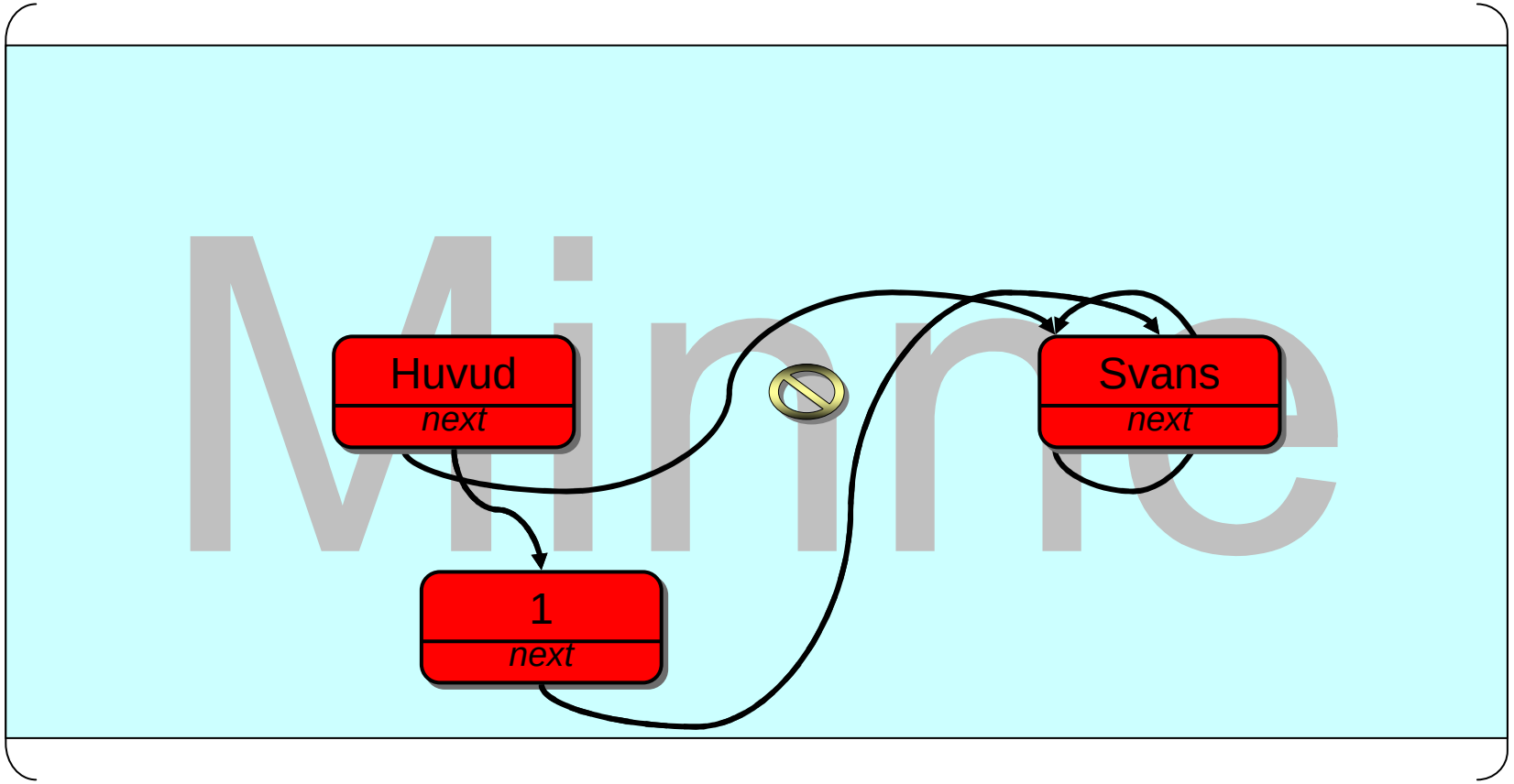
- Bättre sätt – Huvud / Svans!
  - *”Dummy”*-länkar som aldrig tas bort eller bär värden

List traversering

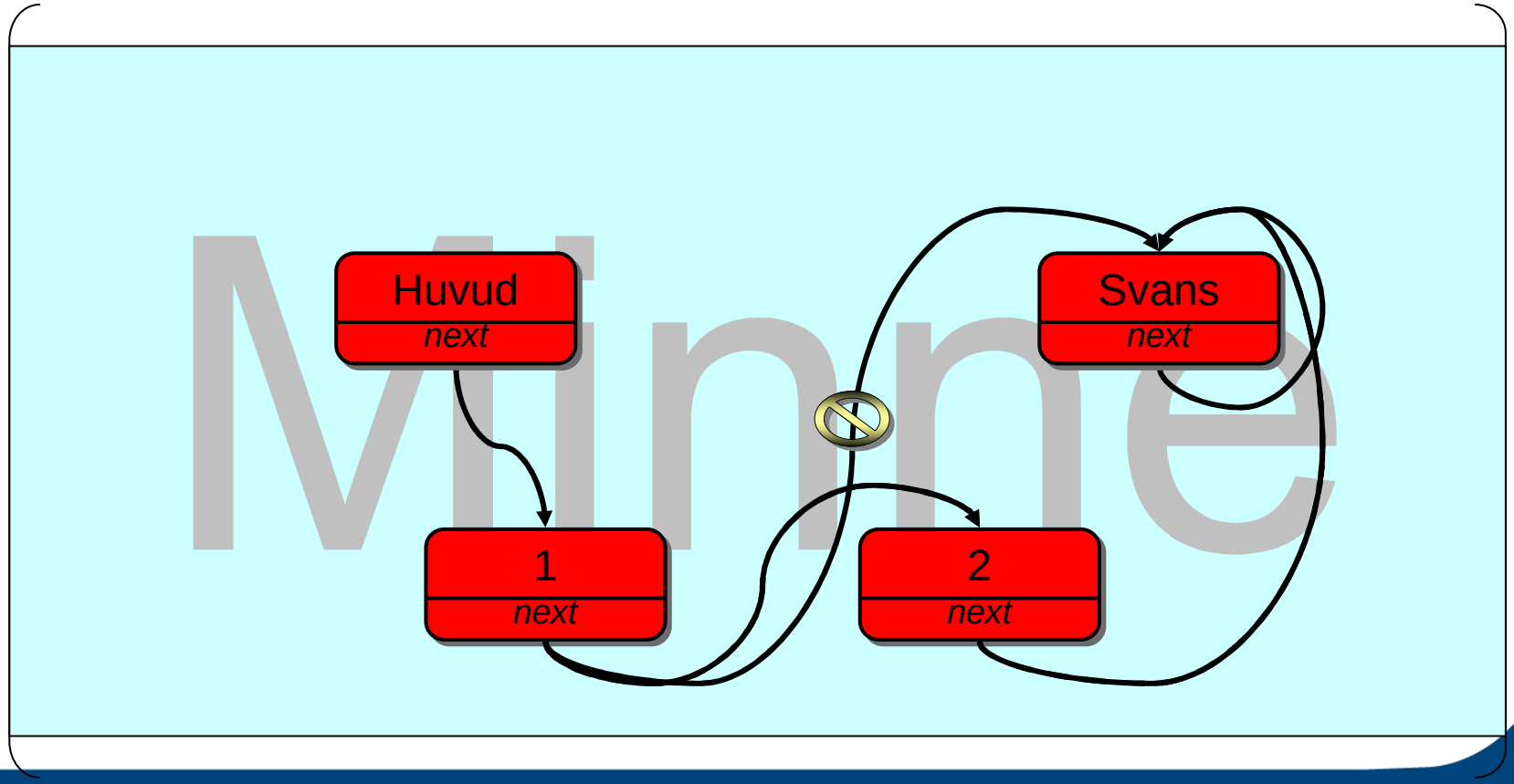
- Ingen möjlighet till indexering som hos statiska arrayer!
- *”Länk för länk”* med start på första länken tills rätt position hittad!

```
public ListNode getNode(Object x)
{
  ListNode temp = header;

  while(temp.element != x)
    temp = temp.next;

  return temp;
}
```

Huvud / Svans



OBS! Först länka mot svansen sedan bryt länken från huvud



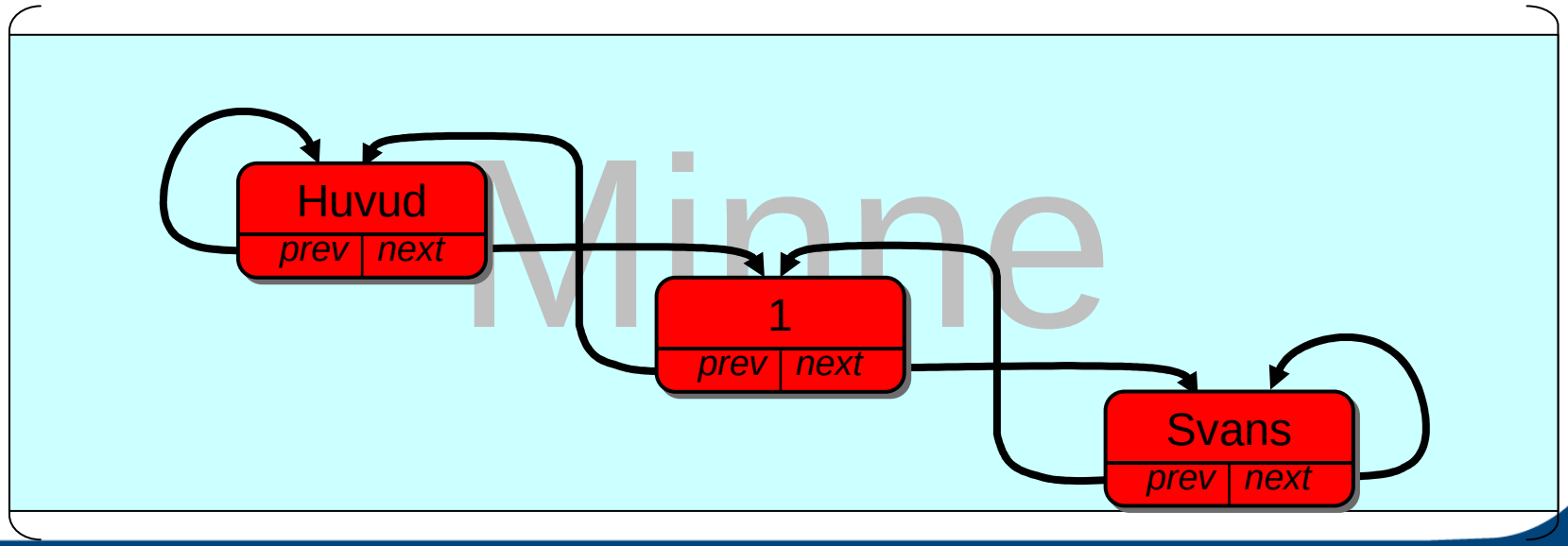
Vanliga operationer på länkade listor

- *insert*()
- *remove*()
- *find*()
- *get*()
- *iterator*()
- *sort*()

I java gäller operationerna från List interface

Dubbellänkade listor

- Varje länk har en referens till nästa länk OCH en referens till *föregående* länk i listan.



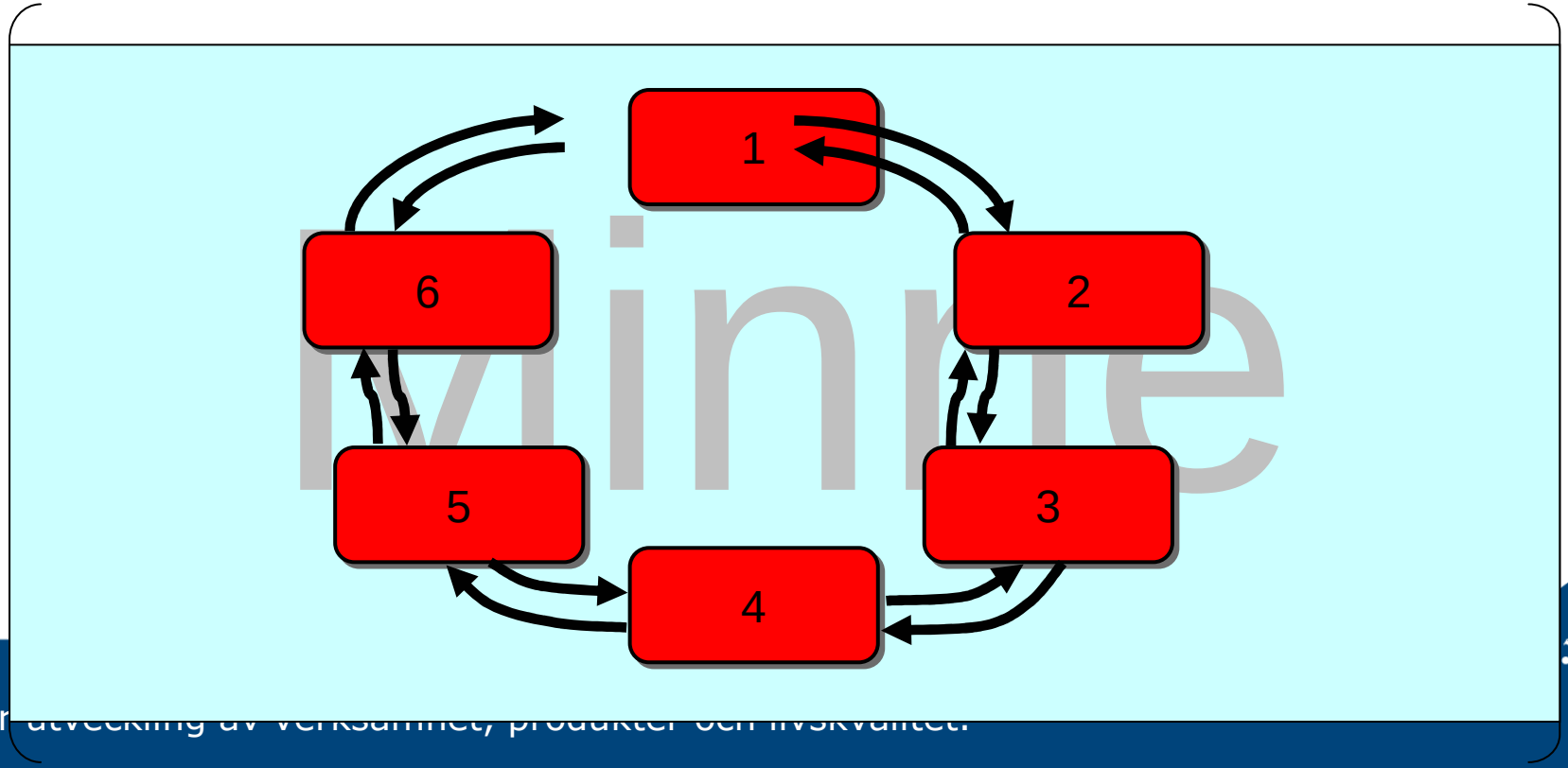
Dubbellänkade listor

- Förenklar listmanipulationer
  - *insert*()
  - *remove*()
- Medför FLER referens-manipulationer för varje operation

```
public void
removeNode(ListNode l)
{
  l.prev.next = l.next;
  l.next.prev = l.prev;

  l = null;
}
```

Cirkulära listor

- Ta bort svansen och huvudet och knyt ihop liständarna



Sortering av länkad listor

- Dirigera om pekarna istället för själva länkinnehållet
  - Betydligt mycket mer effektivt vid stora datamängder.
  - Många referensmanipulationer