UPPSALA
UNIVERSITET

Programmering för Språkteknologer II

Föreläsning 6

Hashtabeller

Sara Stymne

Baserat på bilder av Markus Saers

UPPSALA
UNIVERSITET

Idag

- Hashtabeller
  - De här slidesen
- Komplexitet
  - Övning
- Objektorientering
  - Övning på objektorienterad analys/design

UPPSALA
UNIVERSITET

Innehåll

- Associativa datastrukturer
  - Hashtabeller
  - Sökträd
- Implementationsdetaljer för att använda associativa datastrukturer

UPPSALA
UNIVERSITET

Innehåll

- Associativa datastrukturer
  - Hashtabeller
  - Sökträd
- Implementationsdetaljer för att använda associativa datastrukturer

UPPSALA
UNIVERSITET

Associativa datastrukturer

- Låter oss associera godtyckliga nycklar med godtyckliga värden
  - Vektor: associerar heltal med godtyckliga värden
- Behöver inte vara ”tät”
  - Vektorer: måste ha plats för alla nycklar från 0 till den högsta
- Nycklar måste vara unika
  - En nyckel får inte associeras med flera värden
  - Utgör en ”mängd”
    - Vektorer: mängden giltiga index

UPPSALA UNIVERSITET

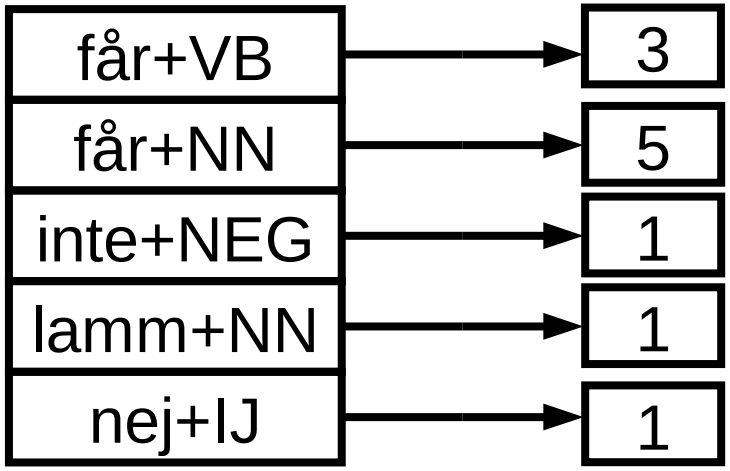
Associativa datastrukturer

- Två vanliga implementationer
  - Hashtabeller
    - (Jätte) snabba
      - O(1) – i medelfallet
    - Osorterade
    - Nästan täta
  - Sökträd
    - Snabba
      - O(log n) (om balanserade)
    - Sorterade
    - Helt täta

UPPSALA UNIVERSITET

Hashtabeller

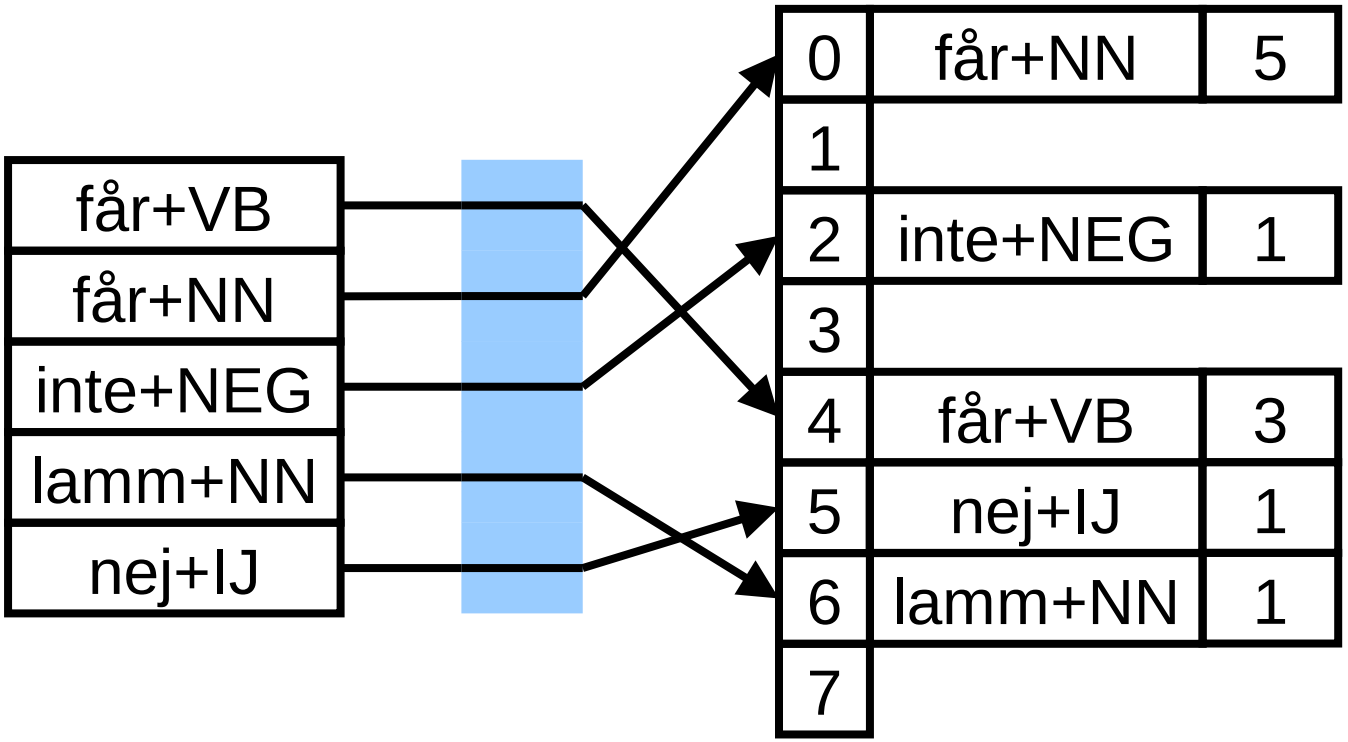
- Nyckel–värde par



UPPSALA UNIVERSITET

Hashtabeller

- Hashfunktion
  - Omvandlar nycklar till giltiga index
  - Internt lagras värdena i en array



UPPSALA
UNIVERSITET

Hashtabeller

- Insättning
  - Tillämpa hashfunktionen
  - Sätt in värdet på indexets plats
- Sökning
  - Tillämpa hashfunktionen
  - Slå upp det erhållna indexet

UPPSALA UNIVERSITET

Hashfunktion och vektorstorlek

- Hashfunktionen returnerar en godtycklig int, x
- Vektorn har en given storlek, n
- Hur gör vi om värdet så det är begränsat till vektorns storlek?
  - 0 <= x <= n-1

UPPSALA
UNIVERSITET

Hashfunktion och vektorstorlek

- Hashfunktionen returnerar en godtycklig int, x
- Vektorn har en given storlek, n
- Hur gör vi om värdet så det är begränsat till vektorns storlek?
  - 0 <= x <= n-1
- Modulo!
- key.hashCode() % n
  - Garanterar att värdet ligger i intervallet 0-n

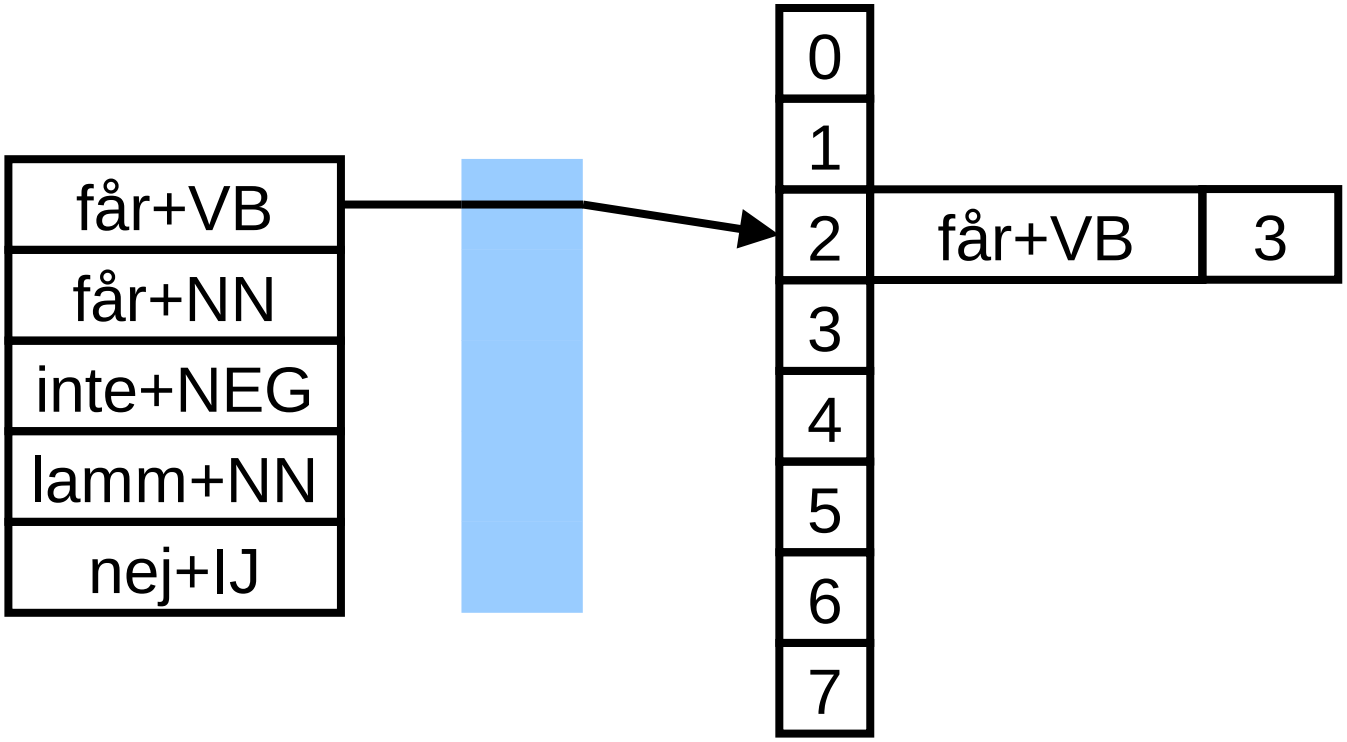
UPPSALA UNIVERSITET

Krockhantering

- Insättning
  - Tillämpa hashfunktionen
  - Sätt in värdet på indexet plats
- Sökning
  - Tillämpa hashfunktionen
  - Slå upp det erhållna indexet
- **Krockhantering**
  - Vad händer om två värden får samma index?

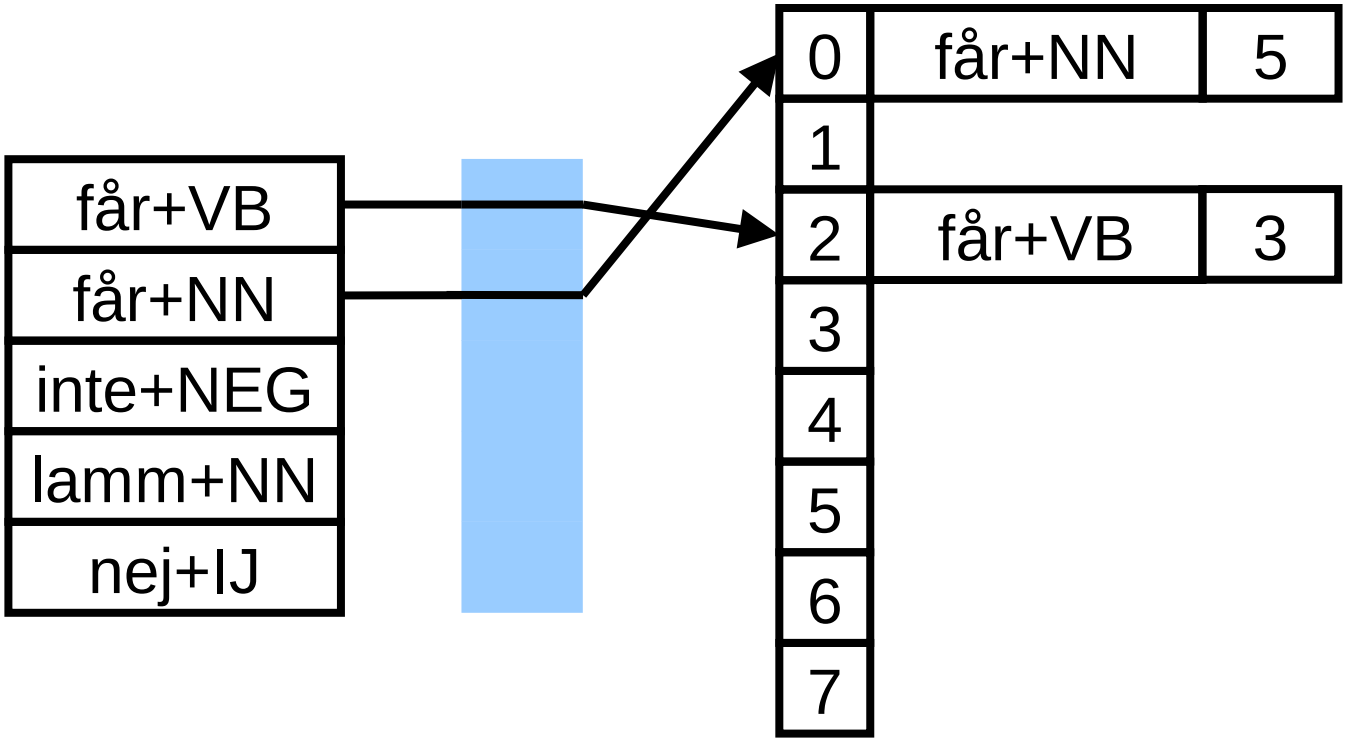
UPPSALA UNIVERSITET

Hashtabeller: krockhantering



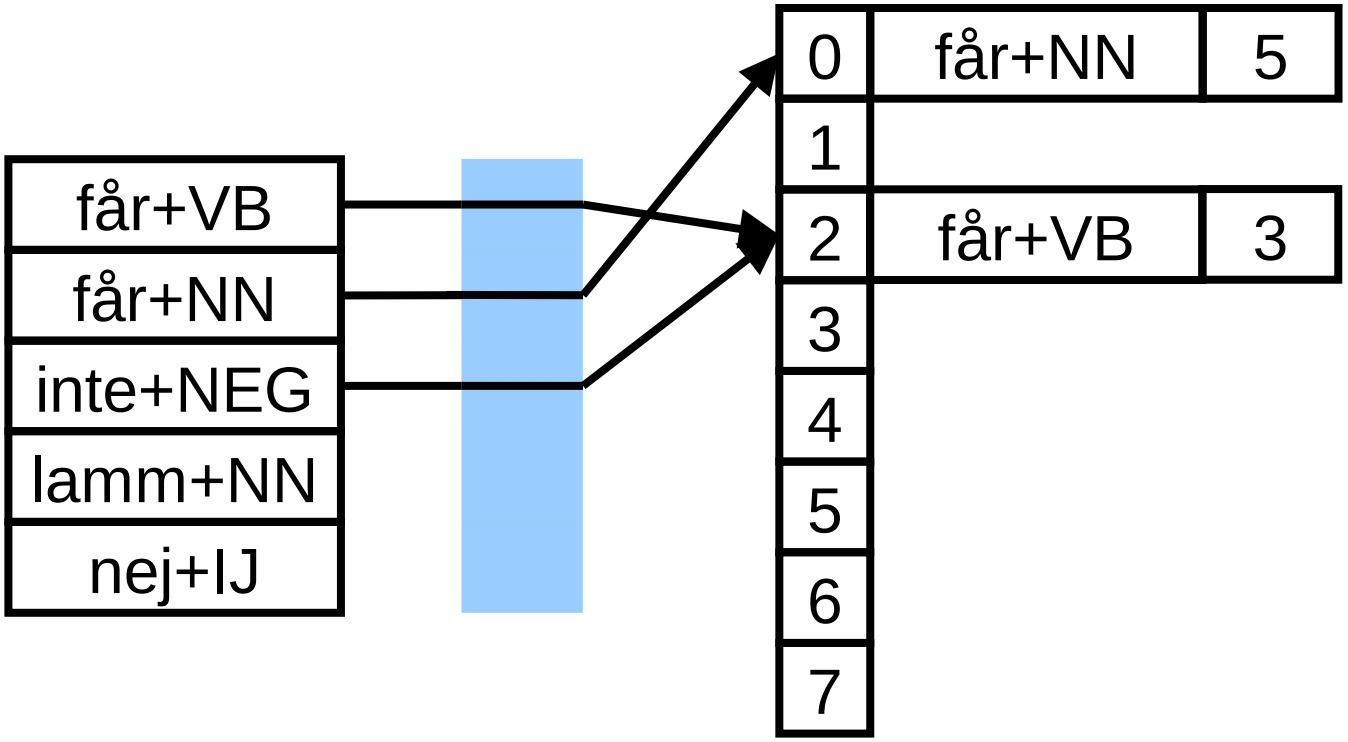
UPPSALA UNIVERSITET

Hashtabeller: krockhantering



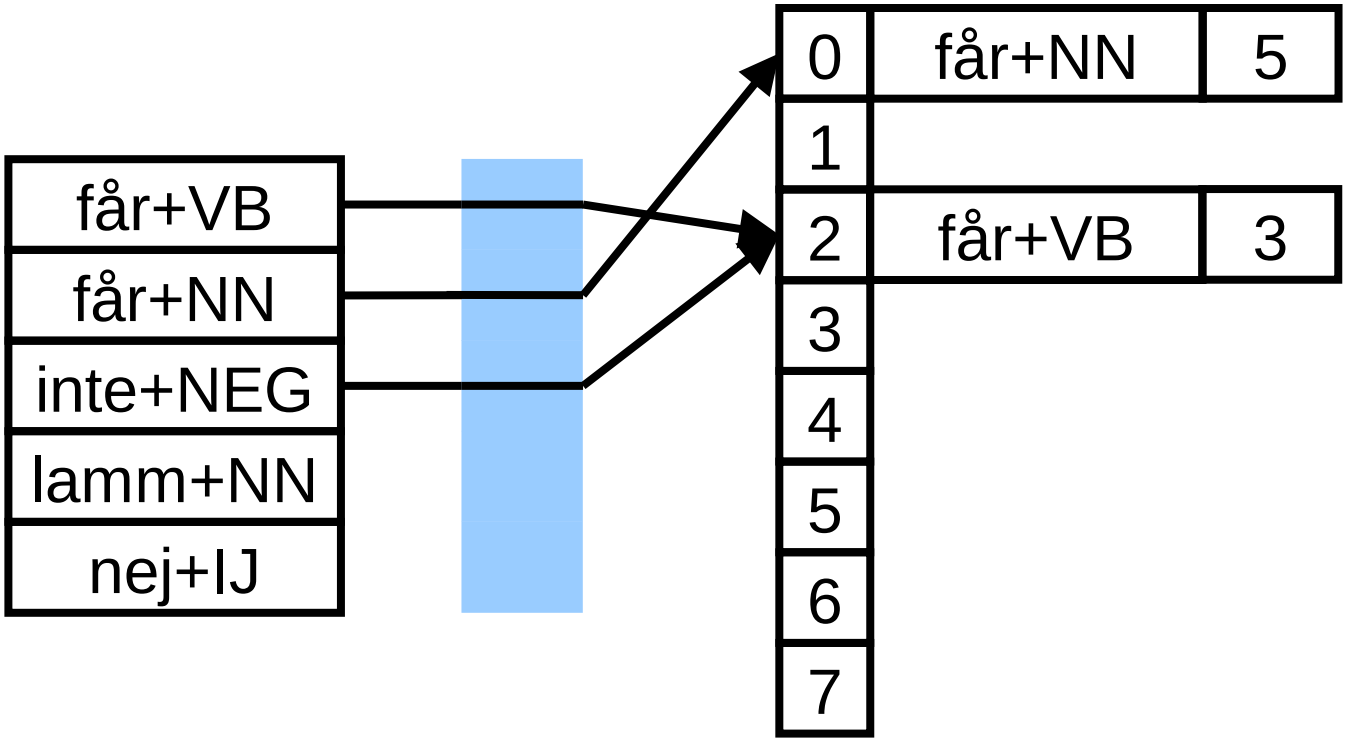
UPPSALA UNIVERSITET

Hashtabeller: krockhantering



UPPSALA UNIVERSITET

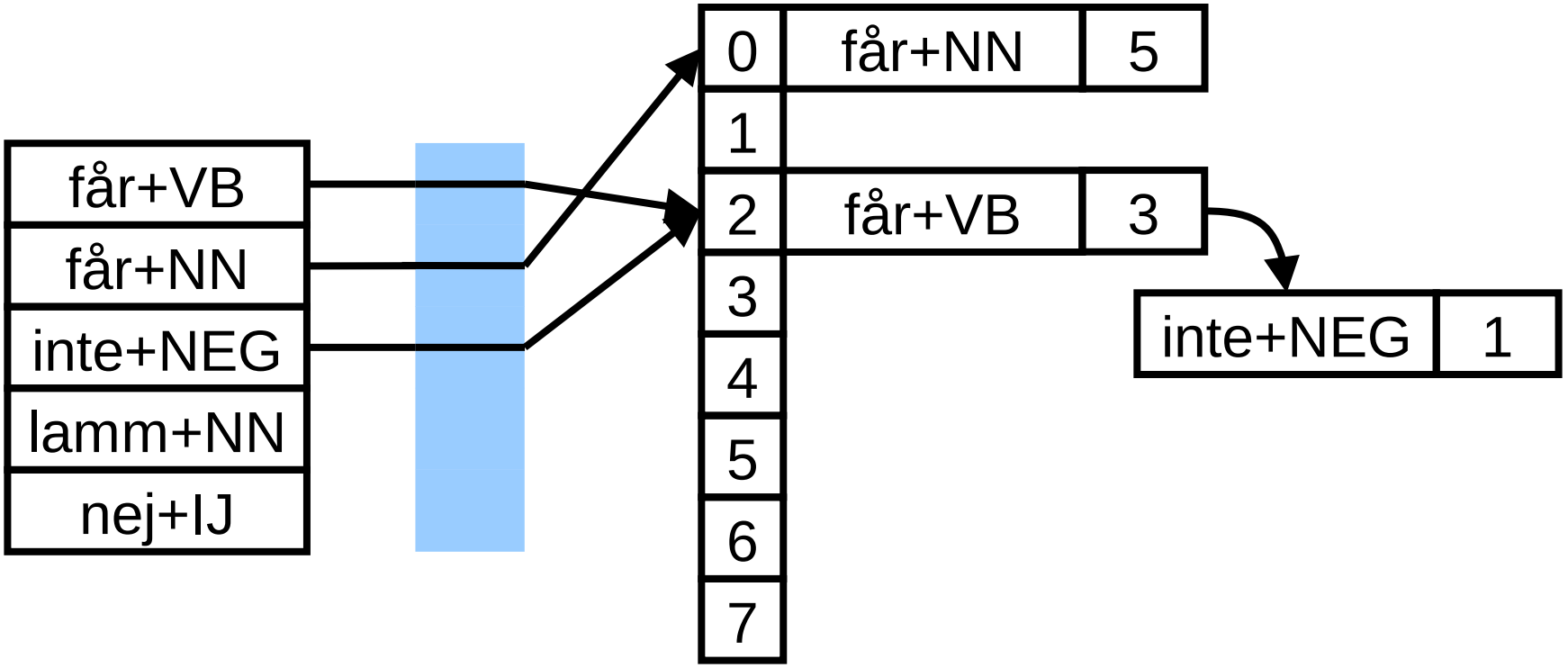
Hashtabeller: krockhantering



UPPSALA UNIVERSITET

Hashtabeller: krockhantering

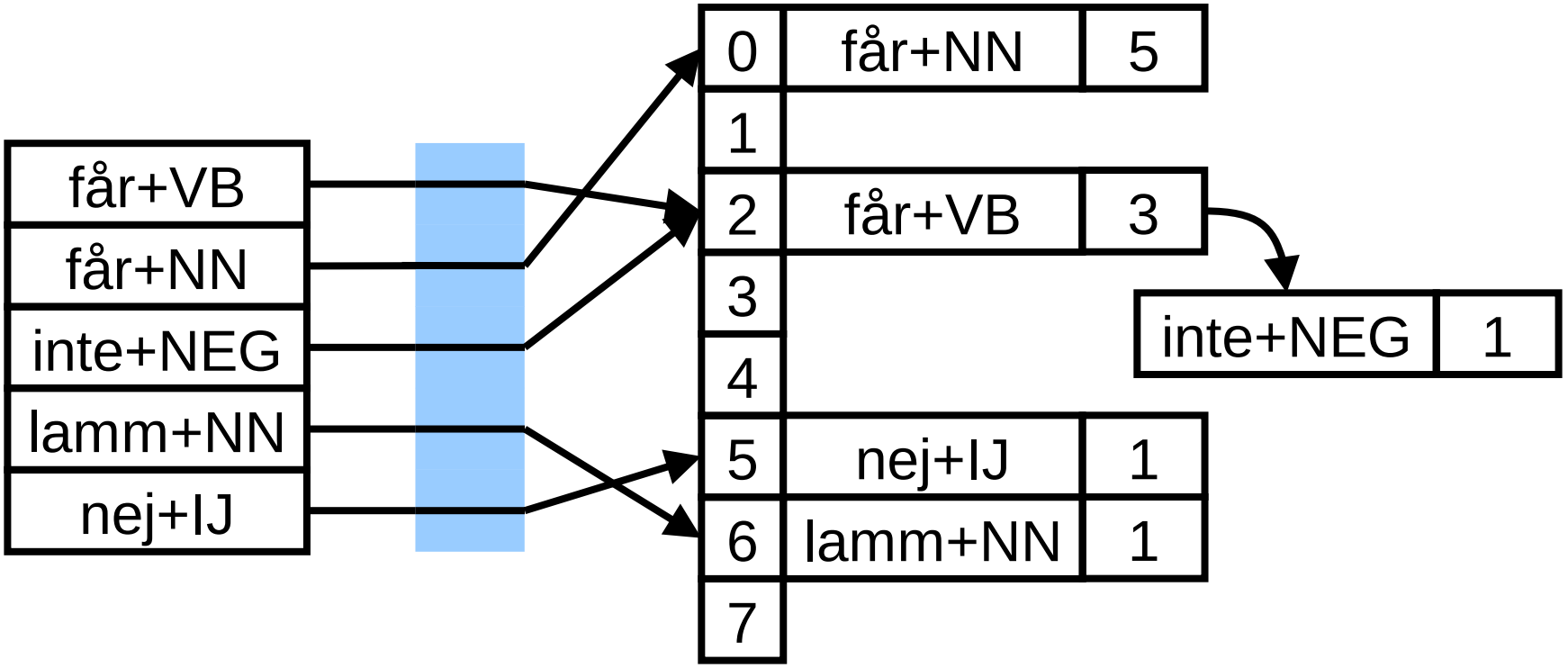
Länkning



UPPSALA UNIVERSITET

Hashtabeller: krockhantering

Länkning



UPPSALA UNIVERSITET

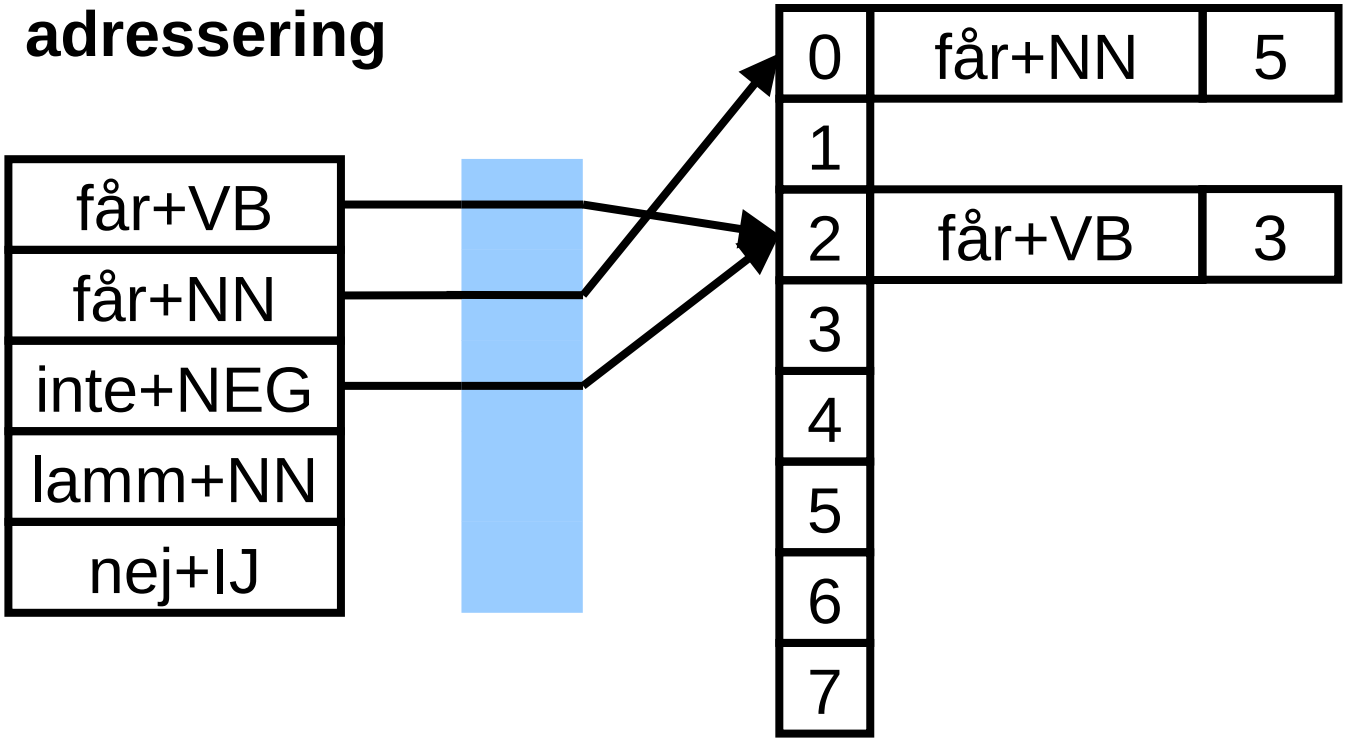
Krockhantering: Länkning

- Varje plats i vektorn är en länkad lista med nyckel–värde-par
- Sökning
  - Applicera hashfunktionen på nyckeln
  - Löp igenom listan tills
    - Nyckeln hittas (medlem i nyckelmängden)
    - Slutet på listan nås (ej medlem i nyckelmängden)

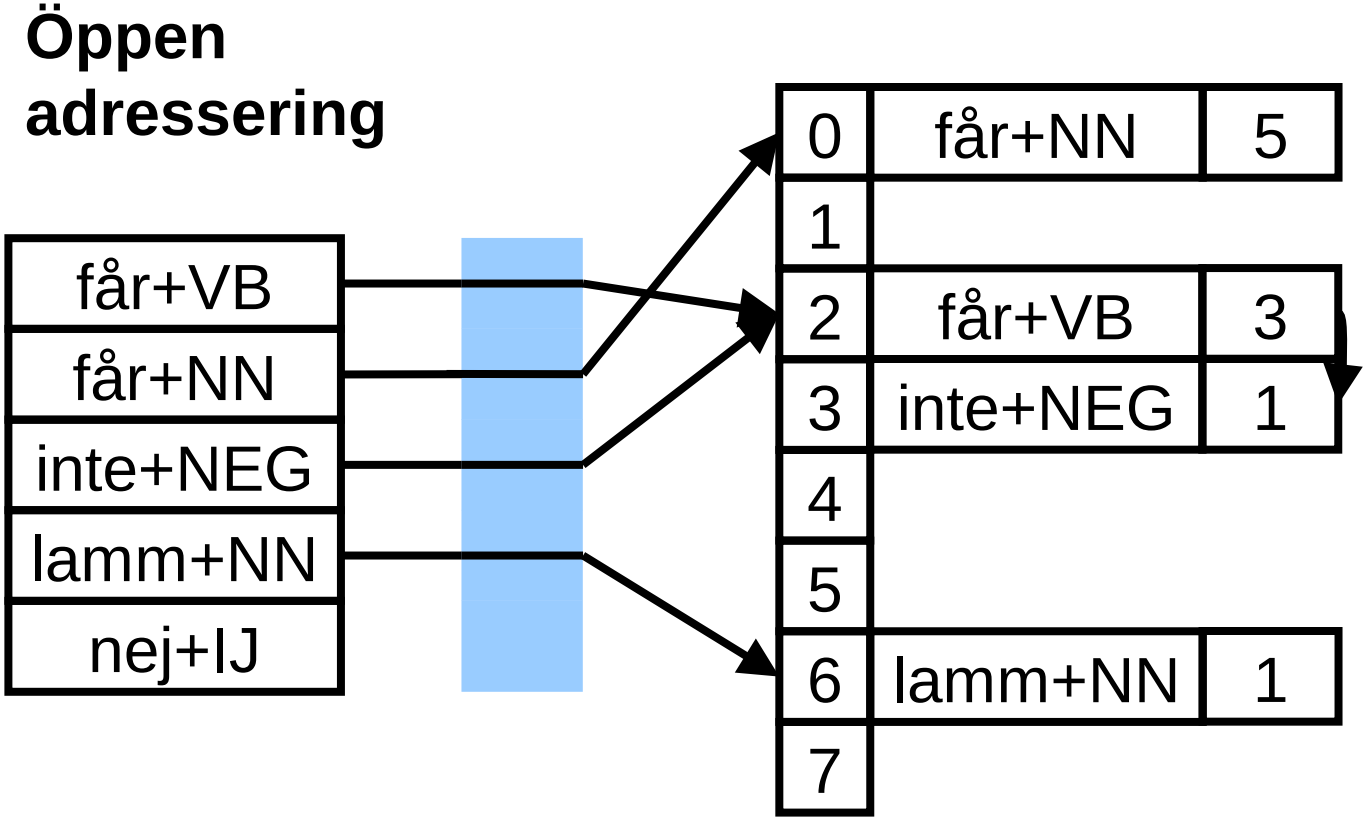
UPPSALA UNIVERSITET

Hashtabeller: krockhantering

Öppen adressering

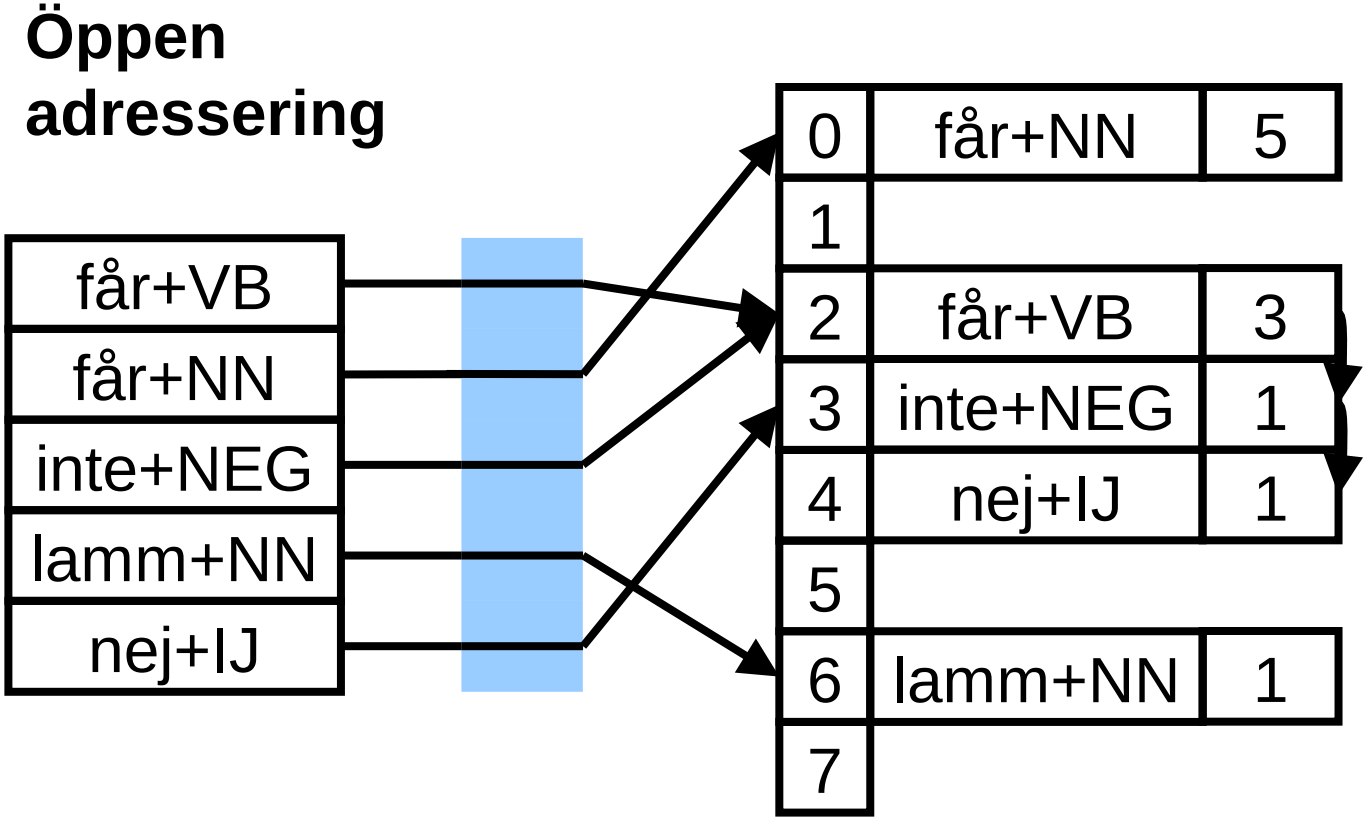


Hashtabeller: krockhantering
Öppen adressering



UPPSALA UNIVERSITET

Hashtabeller: krockhantering
Öppen adressering



UPPSALA UNIVERSITET

Krockhantering: Öppen adressering

- Sökning
  - Applicera hashfunktionen på nyckeln
  - Om platsen är upptagen av en annan nyckel
    - Gå till en annan plats i vektorn
    - Repetera tills nyckeln eller en tom plats hittas
- Olika strategier
  - Gå ett steg i taget
    - Linjär sökning
  - Öka steglängden med antalet steg som tagits
    - Kvadratisk sökning (1,2,4,...
  - Låt en hashfunktion avgöra hur många steg som ska tas
    - Dubbelhashning

UPPSALA UNIVERSITET

Täthet

- Ju fler nycklar som finns i en hashtabell desto större är risken för krockar
- Ju fler krockar desto längre tid tar insättning/sökning
- Trade-off mellan tid och utrymme
- Täthet: antal nycklar/plats
  - 0,75 bra riktvärde
  - Men det beror på data/applikationen

UPPSALA
UNIVERSITET

I Java

- Klasserna **`HashMap`** och **`HashSet`**
  - En mängd kan modelleras genom att man ignorerar värdena och bara bryr sig om nyckelmängden i den associativa datastrukturen
  - Använder länkning som krockhantering
- Kräver en implementation av metoderna **`int hashCode()`** och **`boolean equals(Obejct other)`**
  - Avgör vilket värde som ska in i hashfunktionen
  - Avgör ifall två nycklar är lika
  - Finns definierad i **`Object`**

UPPSALA
UNIVERSITET

Konstruktorer i **`HashMap`** och **`HashSet`**

- **`HashXXX()`**
  - Sätter initialCapacity till 16
  - Sätter loadFactor till 0,75
- **`HashXXX(int initialCapacity)`**
  - Sätter loadFactor till 0,75
- **`HashXXX(int initialCapacity, float loadFactor)`**
- När antalet element överstiger kapaciteten gånger täthetsfaktorn ökas utrymmet
  - Alla värden måste hashas om
  - Tar tid, inte bra att göra ofta
- Täthetsfaktorn kan användas för att balansera effektivitet mot utrymme

UPPSALA
UNIVERSITET

Krockhantering

- Javas implementation av **`HashMap`** och **`HashSet`** hanterar krockar med länkning
- Metoden **`int hashCode()`**
  - Avgör vilken plats en nyckel ska in på
- Metoden **`boolean equals(Object o)`**
  - Avgör om två nycklar är lika i länkningen
- Båda bör implementeras för att objekt av en klass ska kunna hashas
  - En grundimplementation finns i Object

UPPSALA
UNIVERSITET

Idén med **`hashCode()`**

- Whenever it is invoked on the same object more than once during an execution of a Java application, the **`hashCode`** method must consistently return the same integer, provided no information used in **`equals`** comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

UPPSALA
UNIVERSITET

Idén med **`hashCode()`**

- If two objects are equal according to the **`equals(Object)`** method, then calling the **`hashCode`** method on each of the two objects must produce the same integer result.

UPPSALA
UNIVERSITET

Idén med **`hashCode()`**

- It is *not* required that if two objects are unequal according to the **`equals(java.lang.Object)`** method, then calling the **`hashCode`** method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

UPPSALA
UNIVERSITET

Idén med **`hashCode()`**

- Två objekt som är lika enligt **`equals`**
  ska ge samma **`hashCode`**
- Två objekt som inte är lika enligt **`equals`**
  bör ge olika **`hashCode`**
- Ett oförändrat objekt ska ge ett oförändrat **`hashCode`**-värde

UPPSALA
UNIVERSITET

Idén med `equals(Object o)`

- Ta reda på om två objekt är identiska
  - Varför inte `==`?
  - Skillnaden mellan objekt och värden
- Steg för steg
  - Om det är samma objekt är de lika
  - Om de är kompatibla:
    - Gör en explicit typomvandling
    - Kontrollera ifall attributen är identiska
  - Annars är de olika

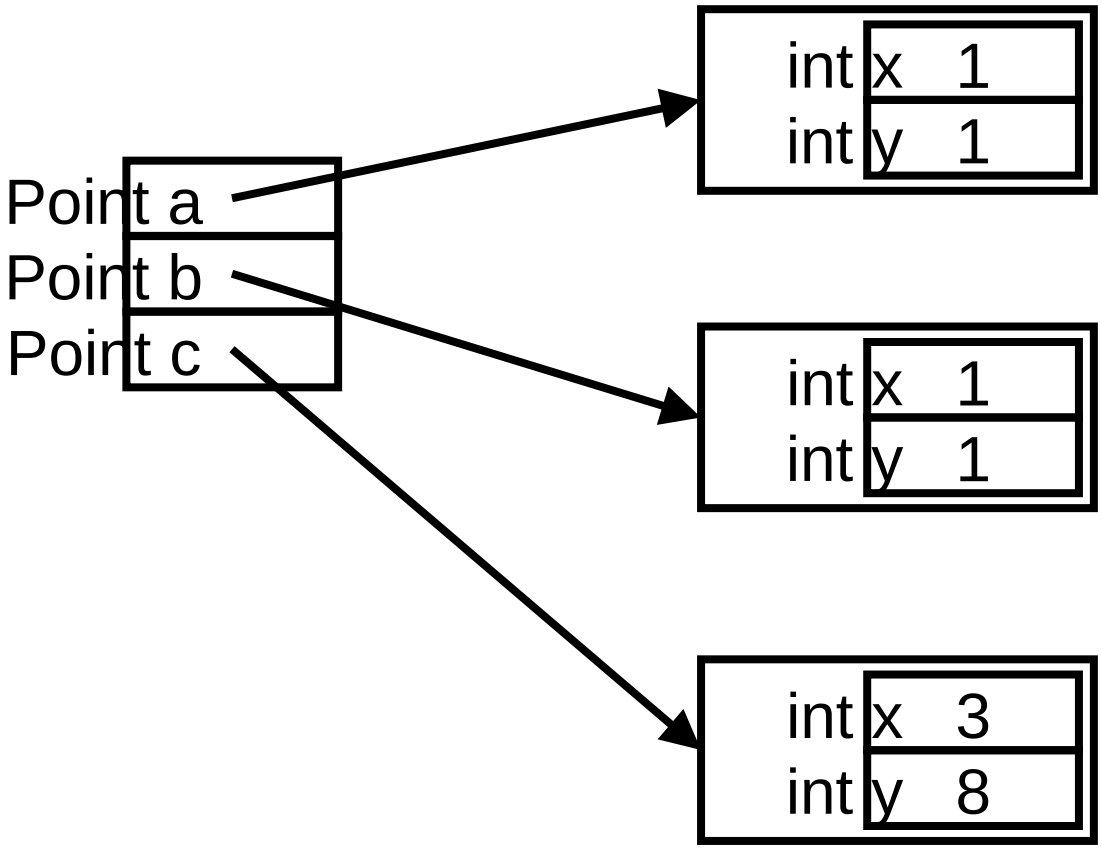
UPPSALA
UNIVERSITET

== och **`equals`**

int	a	12
int	b	12
int	c	48

UPPSALA UNIVERSITET

== och **equals**



UPPSALA
UNIVERSITET

`equals`-metod för `Point`

```
public class Point {
   public boolean equals(Object o) {
      if (this == o) {
         return true;
      }
      if (o instanceof Point) {
         Point oPoint = (Point)o;
         return x == oPoint.x && y == oPoint.y;
      } else {
         return false;
      }
   }
}
```

UPPSALA
UNIVERSITET

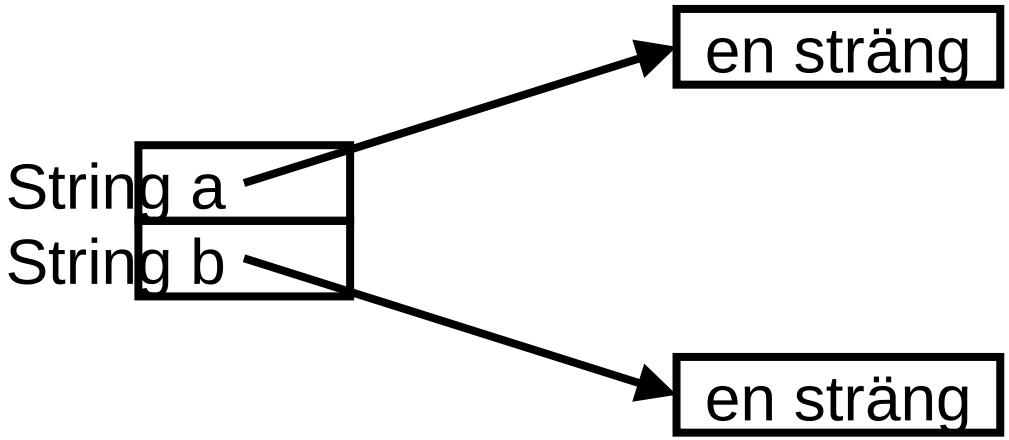
`equals`-metod för `Point`

```
public class Point {
    public boolean equals(Object o) {
       if (this == o) {
          return true;
       }
       if (!o instanceof Point)
          return false;
       Point oPoint = (Point)o;
       return x == oPoint.x && y == oPoint.y;
    }
}
```

UPPSALA UNIVERSITET

Men `String` då?

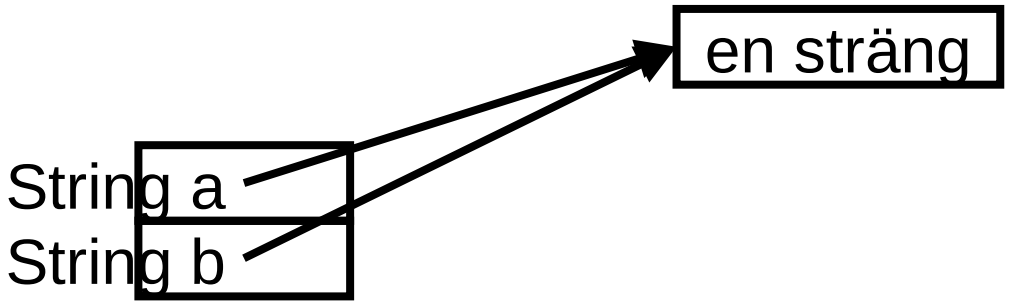
```
public class Test {
   public static void main(String[] args) {
      String a = new String(”en sträng”);
      String b = new String(”en sträng”);
      System.out.println(a == b);
      System.out.println(a.equals(b));
   }
}
```



UPPSALA UNIVERSITET

Men `String` då?

```
public class Test {
   public static void main(String[] args) {
      String a = new String(”en sträng”);
      String b = new String(”en sträng”);
      System.out.println(a == b);
      System.out.println(a.equals(b));
   }
}
```



UPPSALA
UNIVERSITET

Men `String` då?

- För att spara plats existerar varje unik sträng bara på ett ställe
- Alla pilar till identiska strängar pekas om så att de pekar ut samma sträng-objekt
- Fungerar eftersom String är mutable – ”det den pekar på kan ej ändras, bara vad den pekar på”
- Inga garantier för att det fungerar i framtiden – använd **`equals()`**!

UPPSALA
UNIVERSITET

Komplexitet

	Medelfallet	Värsta fallet
Insättning	O(1)	O(n)
Sökning	O(1)	O(n)
Borttagning	O(1)	O(n)

- Vad påverkar värsta fallet?
  - Hashfunktionen!
  - Tätheten
    - Tradeoff tid/plats