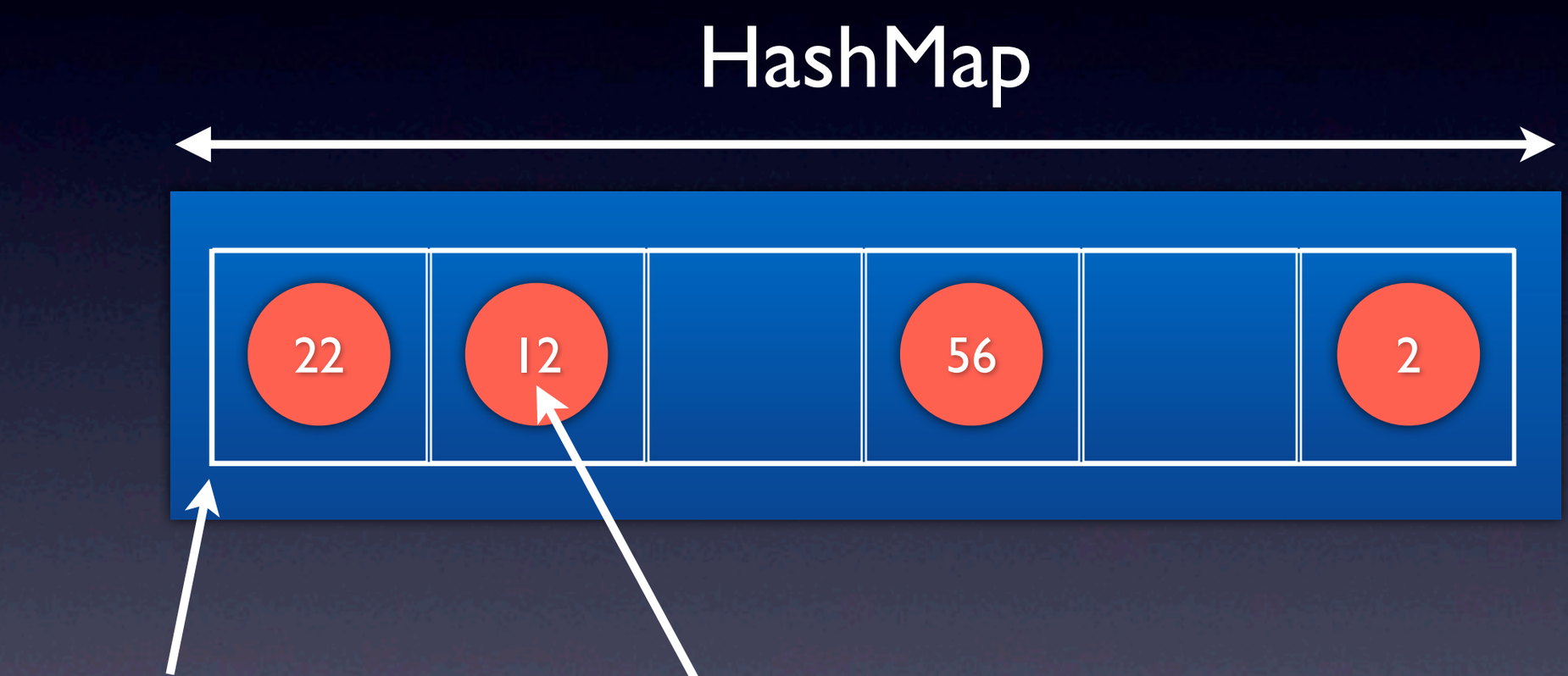
HashMap / Dictionary

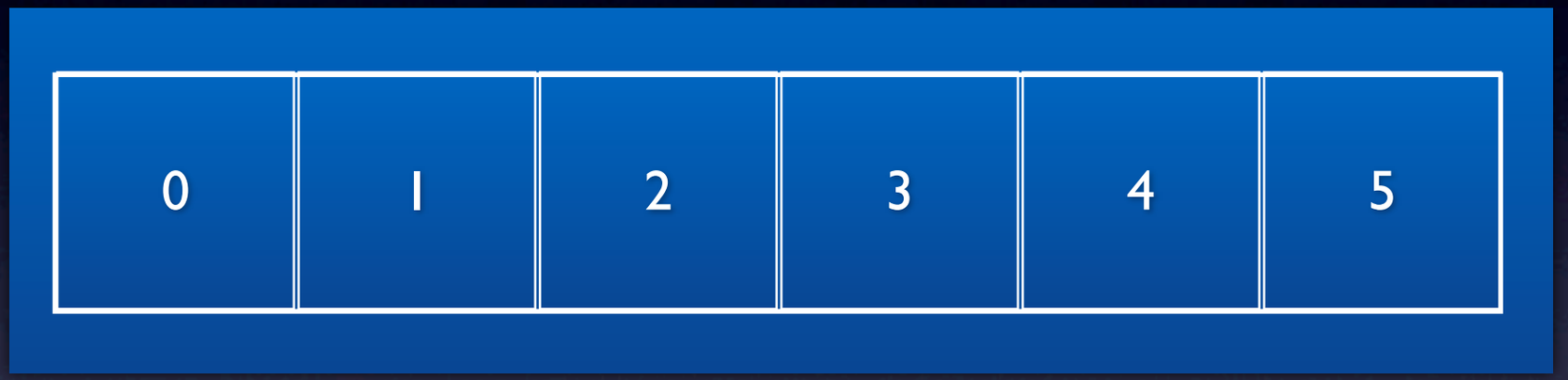
Aufbau



Motivation

- Effizienter Key-Value-Store
- Schnelles Einfügen -> O(1) (AVG)
- Schnelle Suche -> O(1) (AVG)

Einfügen eines Elements

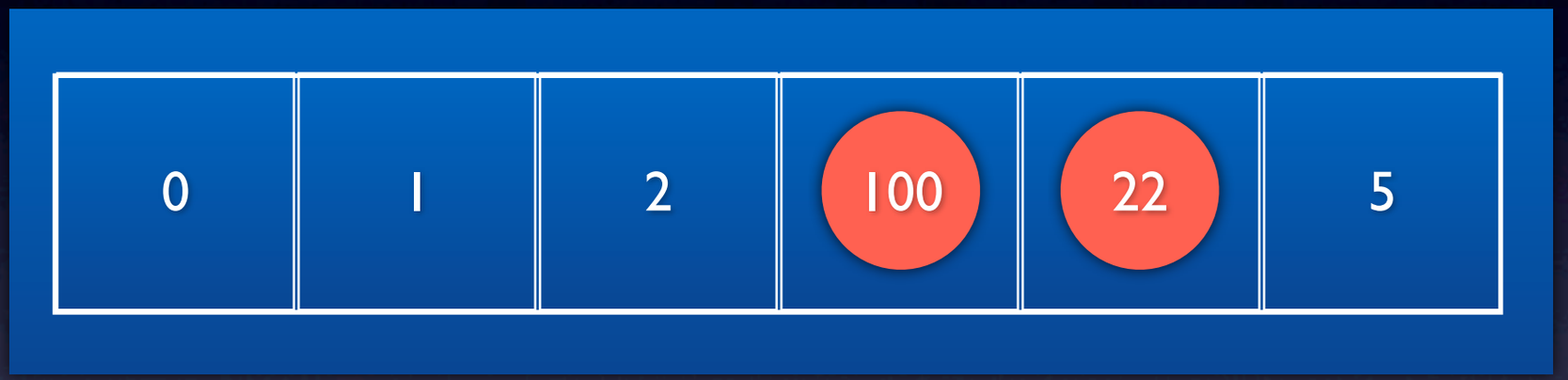


22 key: “test”
value: 22

100 key: “bla”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

Einfügen eines Elements

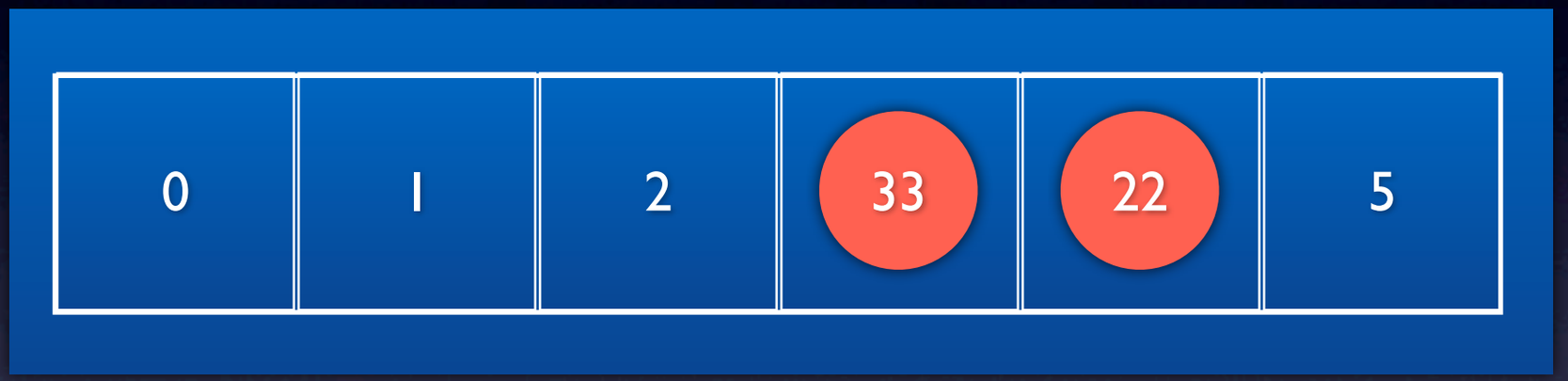


22 key: “test”
value: 22

100 key: “bla”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

Problem: Hash Funktion

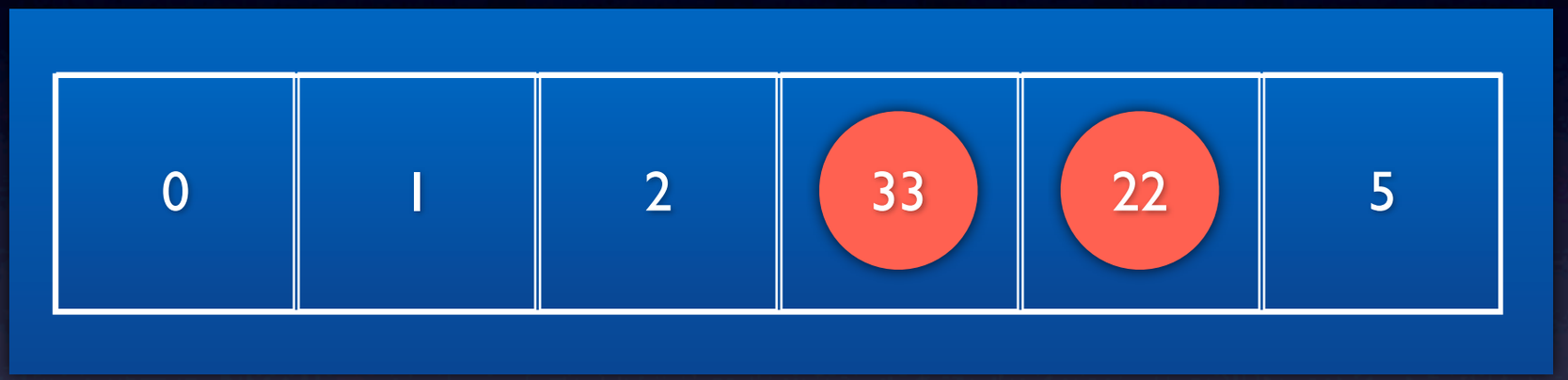


12

key: “einzulangerkey”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

Lösung: Modulo

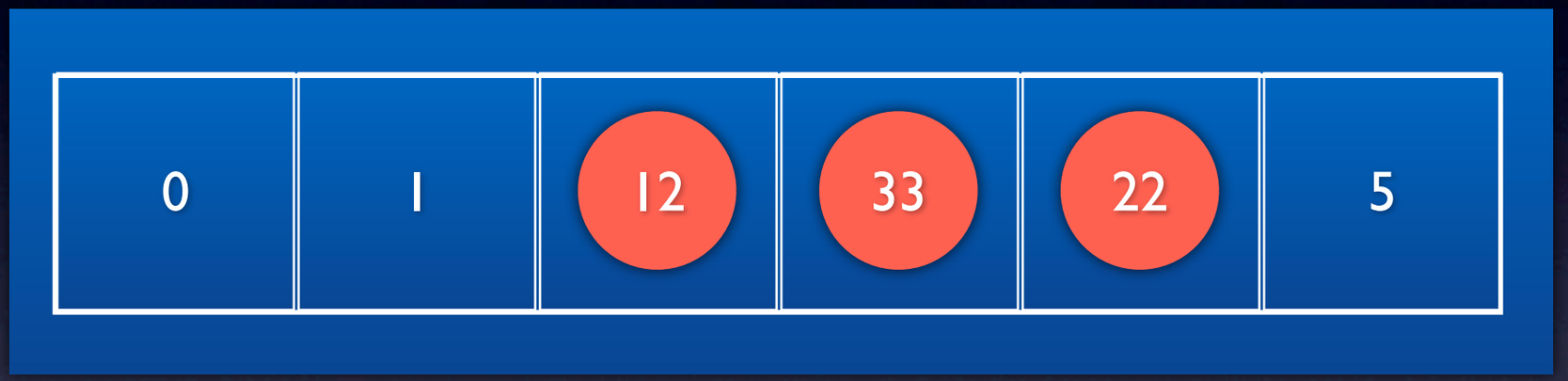


12

key: “einzulangerkey”
value: 22

```
int pos = Math.abs(key.hashCode()) % data.length;
```

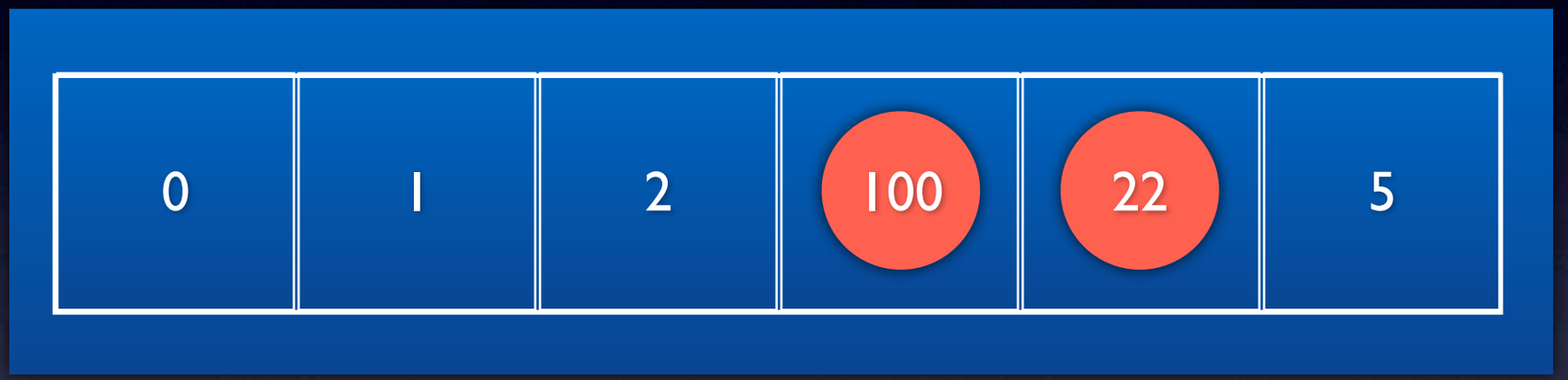
Lösung: Modulo



12 key: “einzulangerkey”
value: 22

```
int pos = Math.abs(key.hashCode()) % data.length;
```

Problem: Kollision

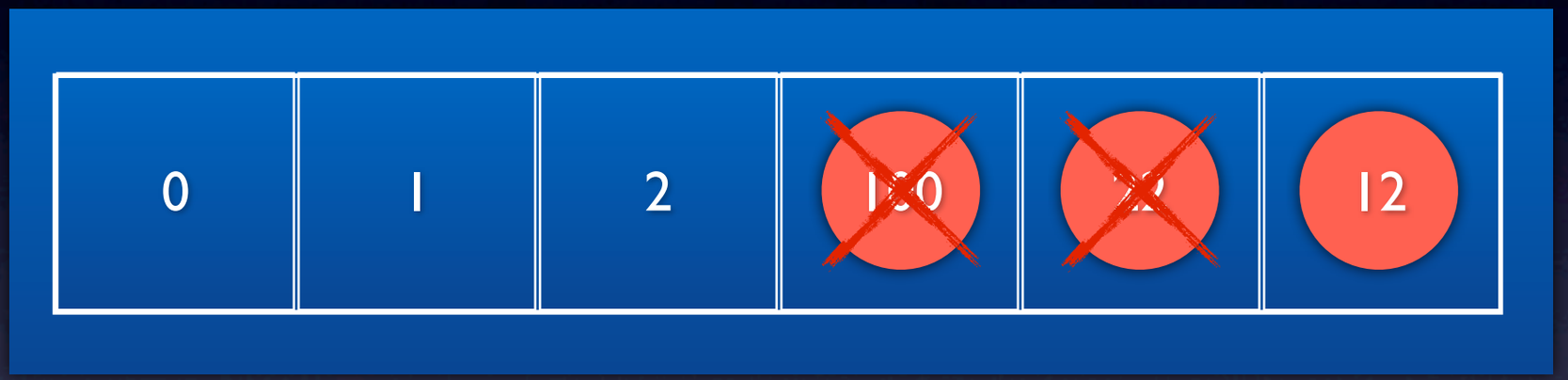


12

key: “bob”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

Lösung: verschieben

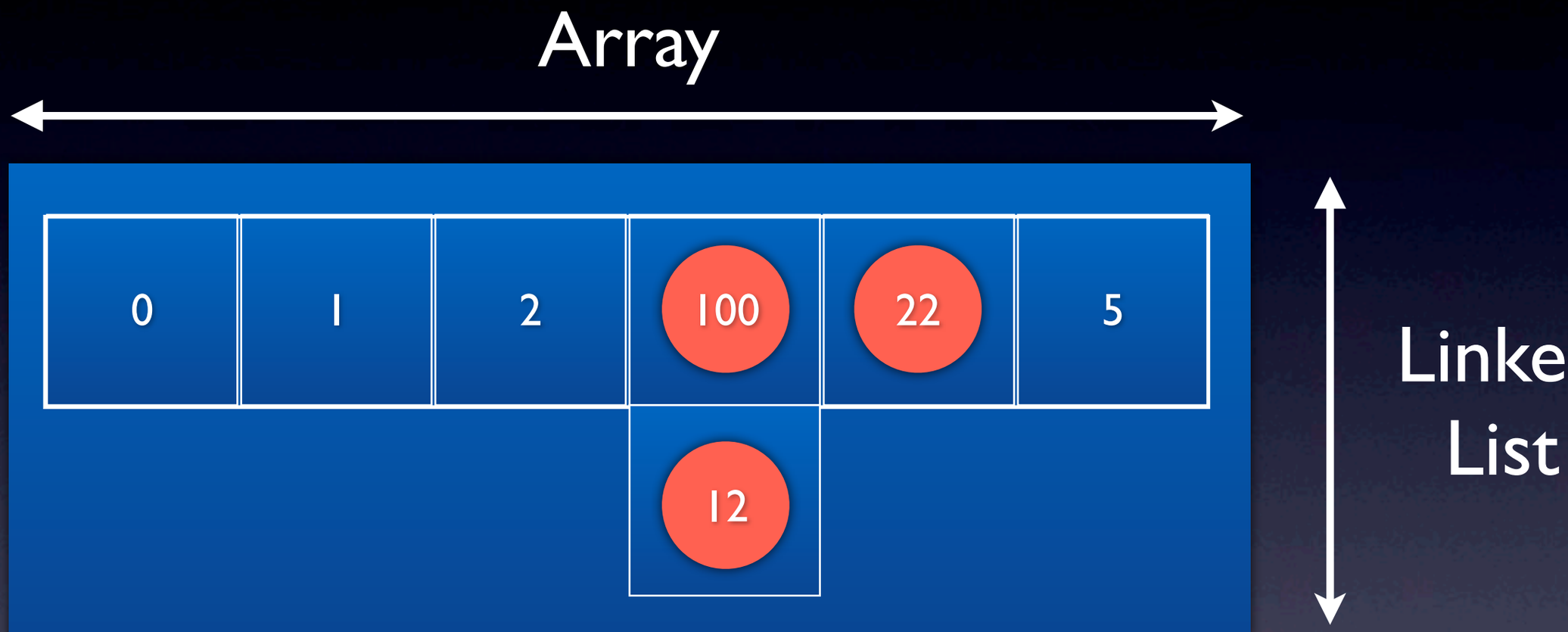


12

key: “bob”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

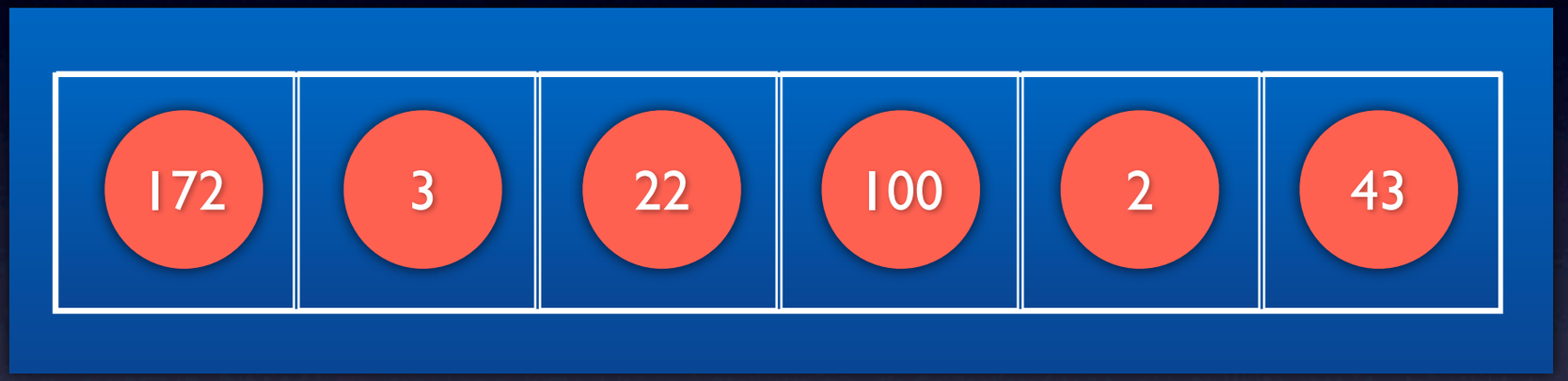
Lösung: verketten



12 key: “bob”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

Problem: Array überfüllt



12

key: “bob”
value: 22

```
public int hashCode(String s) {
    return s.length;
}
```

Lösung: Array verdoppeln

- Neues Array anlegen
  - Doppelte Größe
- Alle Werte des alten Arrays übertragen
  - Für jeden Wert den Hash neu berechnen!

Beispiel

22

key: “testing”
value: 22

0	1	2	3	4	5

0	1	2	3	4	5	6	7

Beispiel

22

key: “testing”
value: 22