Πίνακες Κατακερματισμού

Hash Tables

Δομές αναζήτησης … μέχρι στιγμής

Δομή	Αναζήτηση
Table	O(N)
Sorted Table	O(log N)
List	O(N)
Sorted List	O(N)
Balanced Binary Tree	O(log N)

Υπάρχει κάτι καλύτερο;

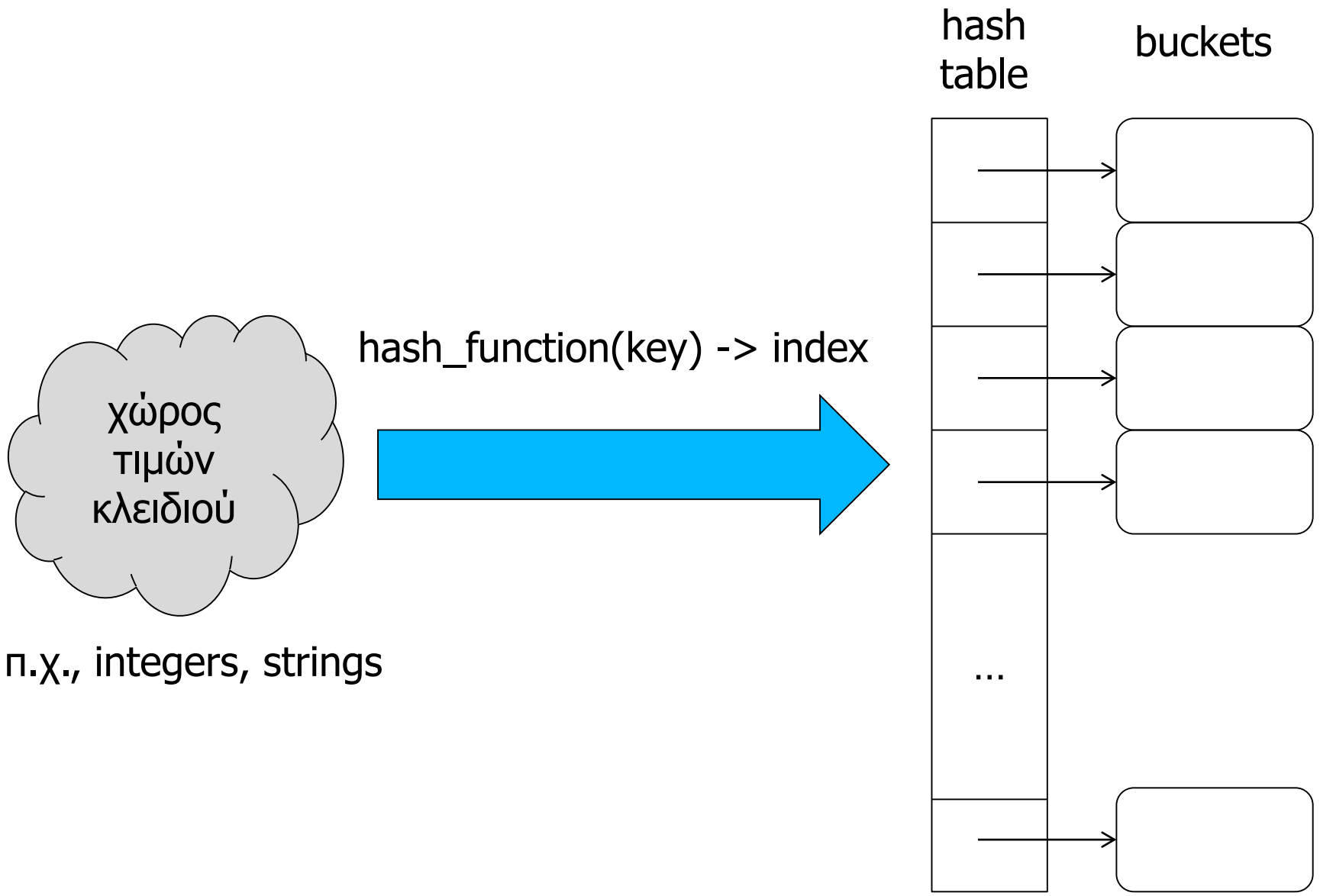
- Ένα ιδανικό ευρετήριο θα υποστήριζε όλες τις πράξεις (προσθήκης, αναζήτησης, απομάκρυνσης) με σταθερό **κόστος** που είναι **ανεξάρτητο** του αριθμού των στοιχείων που περιέχει το ερευτήριο
  - της τάξης $O(1)$
- Μια τέτοια ιδιότητα φαίνεται αδύνατη
- Με κατάλληλα «μαγικά», είναι εφικτή
- **Πίνακες κατακερματισμού (hash tables)**

Δομές αναζήτησης

Δομή	Αναζήτηση
Table	$O(N)$
Sorted Table	$O(\log N)$
List	$O(N)$
Sorted List	$O(N)$
Balanced Binary Tree	$O(\log N)$
Hash table	$O(1)$

Πίνακας κατακερματισμού

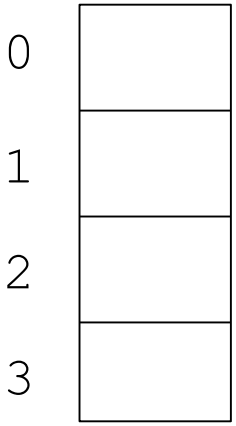
- Κάθε θέση του πίνακα δείχνει σε ένα ξεχωριστό **δοχείο** (bucket) που περιέχει τις επιμέρους εγγραφές
  - πίνακας Ν θέσεων: Ν ξεχωριστά δοχεία
- Η αντιστοίχιση των εγγραφών στα δοχεία του πίνακα γίνεται χρησιμοποιώντας ως «κλειδί» την τιμή από ένα ή περισσότερα πεδία της εγγραφής
- Το κλειδί **μετασχηματίζεται** μέσα από μια **συνάρτηση κατακερματισμού** (hash function)
  - επιστρέφεται ένα hash value/tag, που καθορίζει την θέση του πίνακα/δοχείου που θα τοποθετηθεί η εγγραφή



π.χ., integers, strings

hash_function:

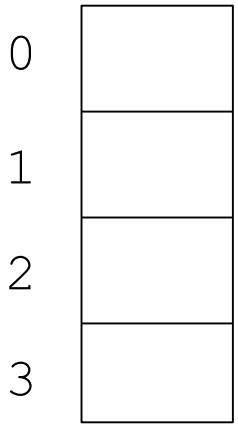
```
str[0] % 4
```

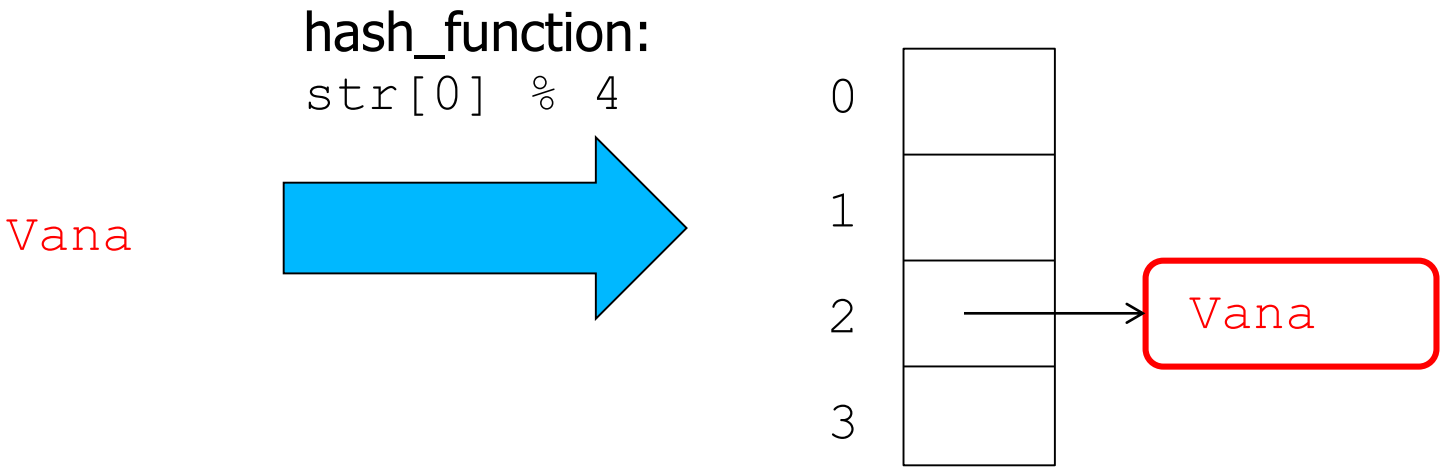


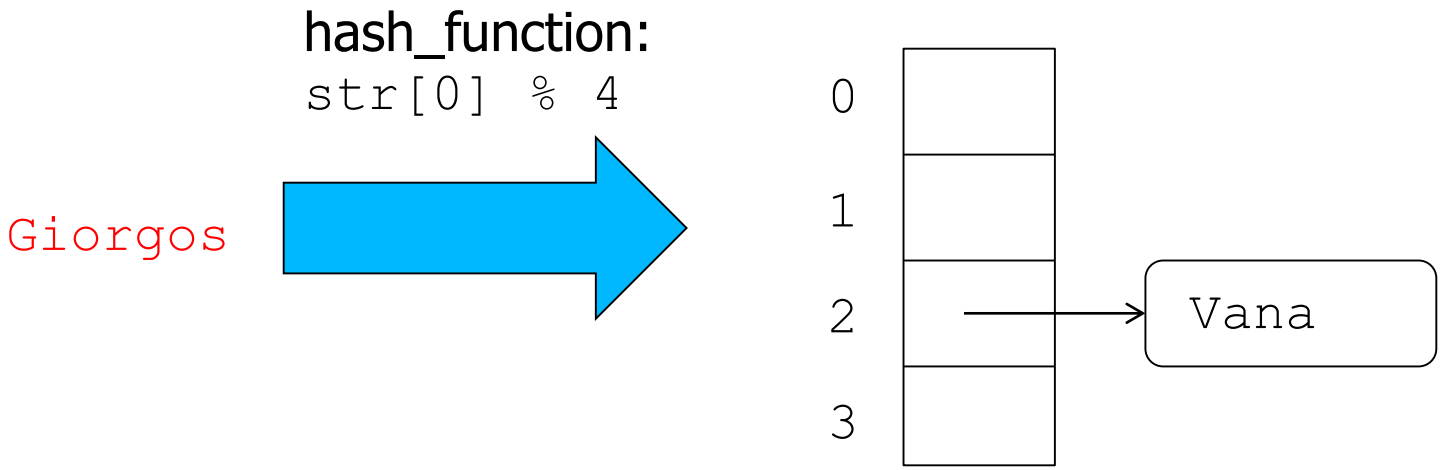
hash_function:

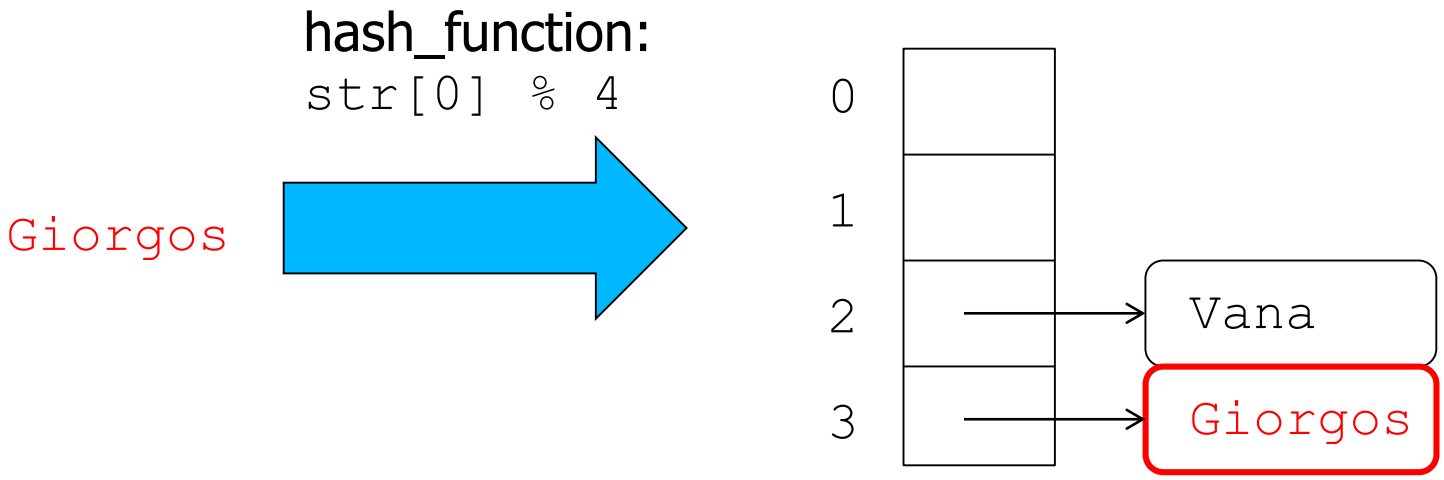
`str[0] % 4`

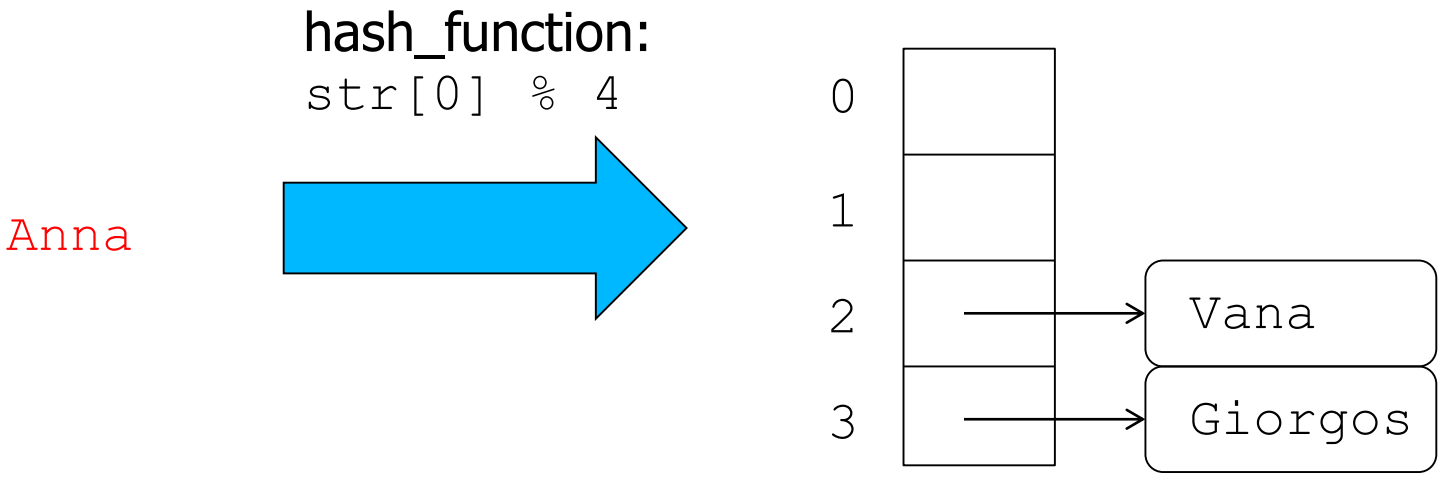
Vana

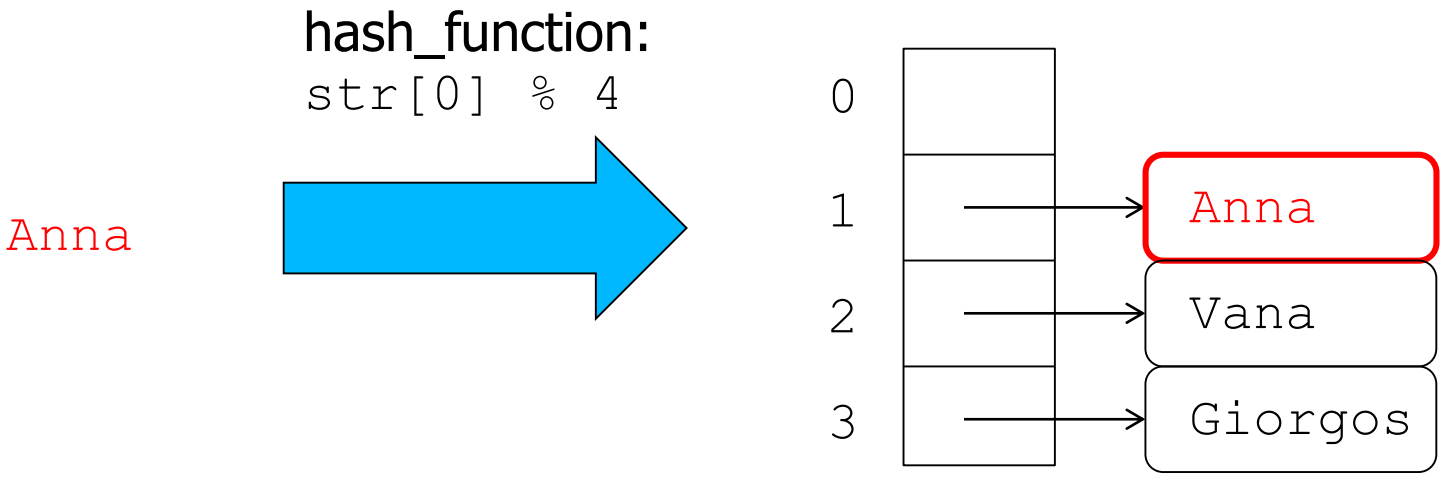


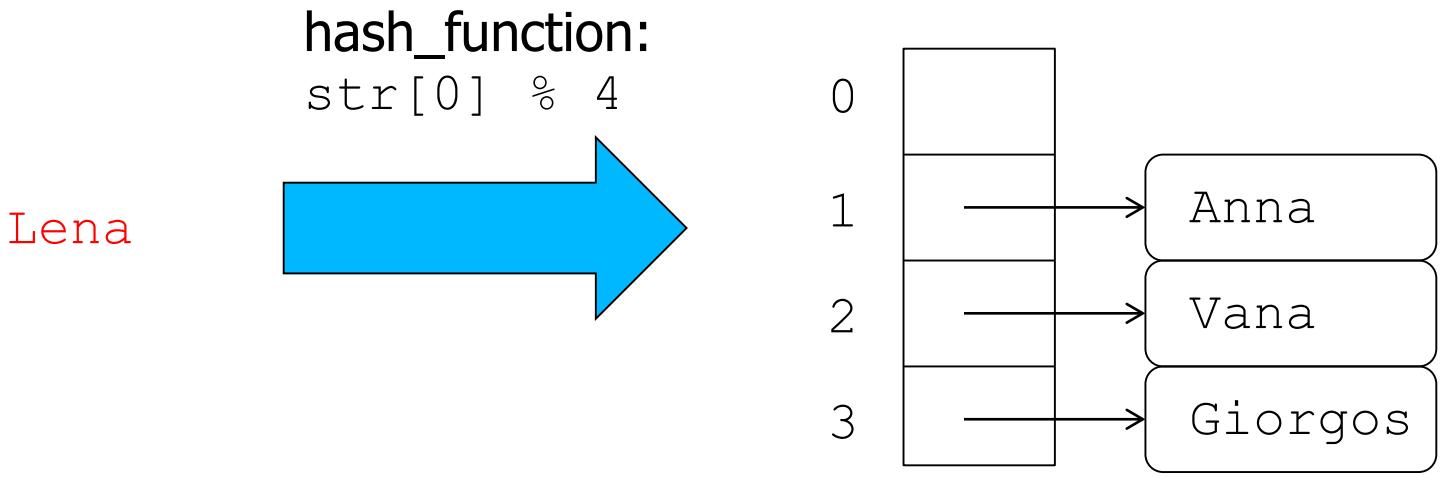

hash_function:
str[0] % 4
Vana
0
1
2
3
Vana

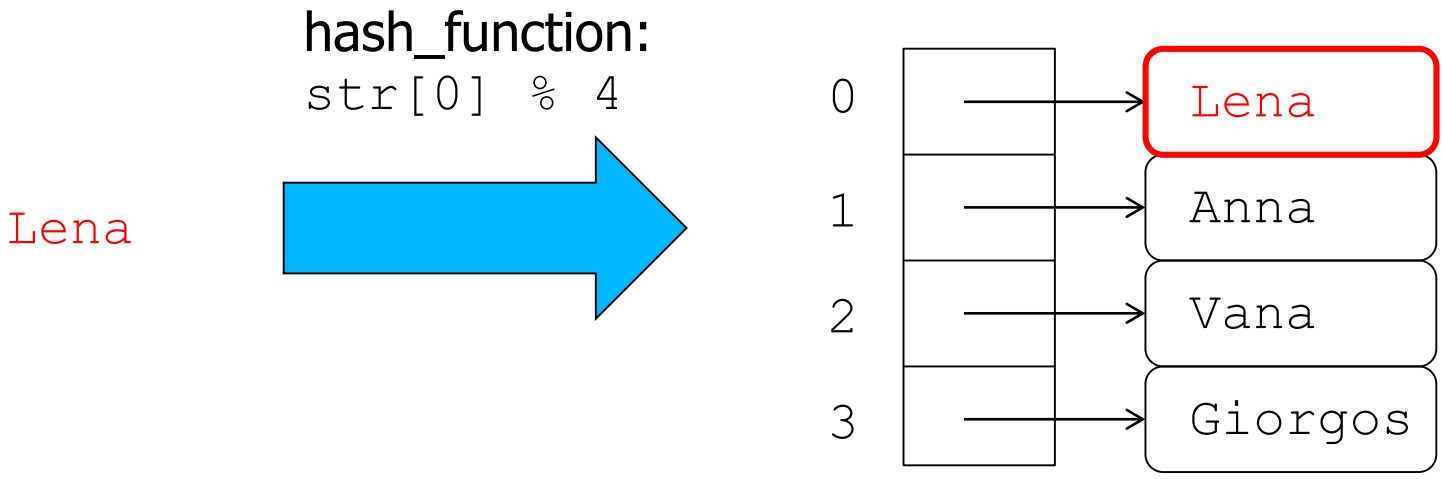

hash_function:
str[0] % 4
Giorgos
0
1
2
3
Vana


hash_function:
str[0] % 4
Giorgos
0
1
2
3
Vana
Giorgos


hash_function:
str[0] % 4
Anna
0
1
2
3
Vana
Giorgos


hash_function:
str[0] % 4
Anna
0
1
2
3
Anna
Vana
Giorgos


hash_function:
str[0] % 4
Lena
0
1
2
3
Anna
Vana
Giorgos


hash_function:
str[0] % 4
Lena
0
1
2
3
Lena
Anna
Vana
Giorgos

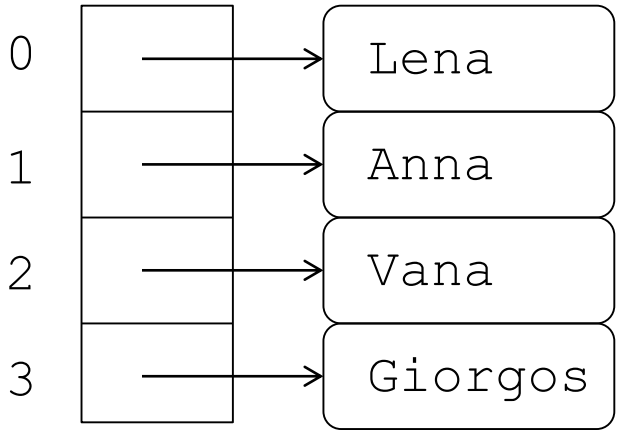
Συγκρούσεις

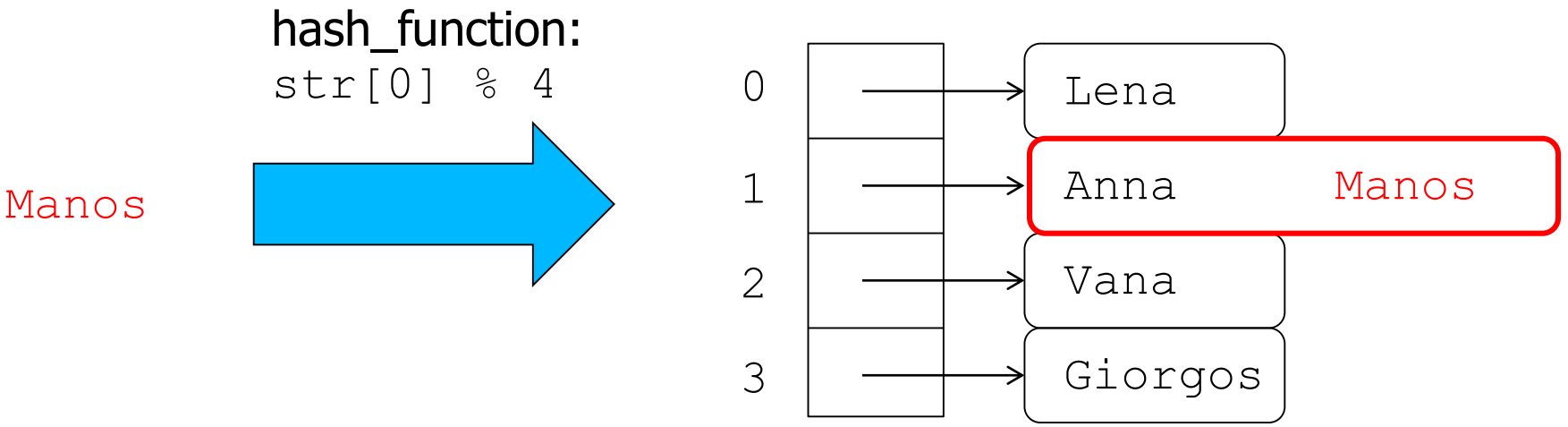
- Η συνάρτηση κατακερματισμού μπορεί να επιστρέψει την **ίδια τιμή** για **διαφορετικές** τιμές κλειδιών
- Αυτό ονομάζεται **σύγκρουση** (collision)
- Οι εγγραφές που έχουν τα ίδια hash values τοποθετούνται στο **ίδιο** δοχείο
  - παρότι έχουν διαφορετικές τιμές κλειδιά
- Πρέπει να υποστηρίζεται **αναζήτηση** εγγραφών **μέσα σε ένα δοχείο** με βάση την τιμή του κλειδιού
  - κατάλληλη δομή αναζήτησης
  - π.χ. ταξινομημένη λίστα (chained buckets)

Manos

hash_function:

```
str[0] % 4
```

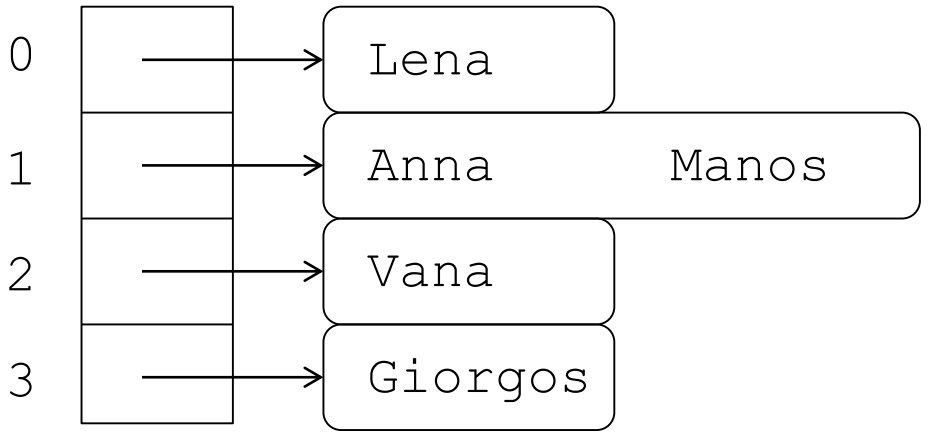


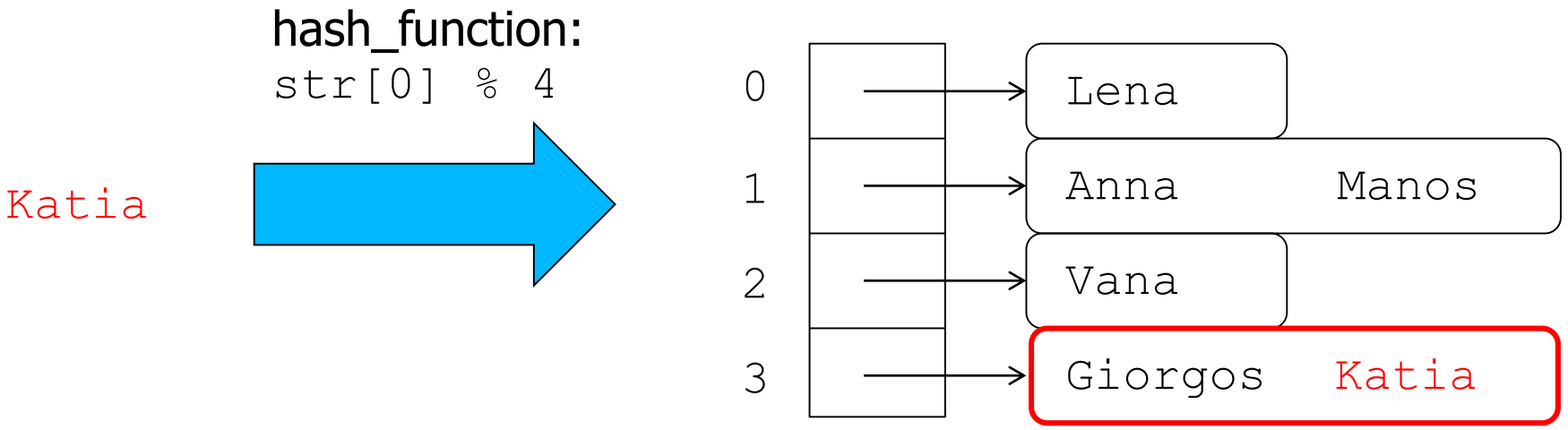

hash_function:
str[0] % 4
Manos
0
1
2
3
Lena
Anna
Manos
Vana
Giorgos

hash_function:

```
str[0] % 4
```

Katia

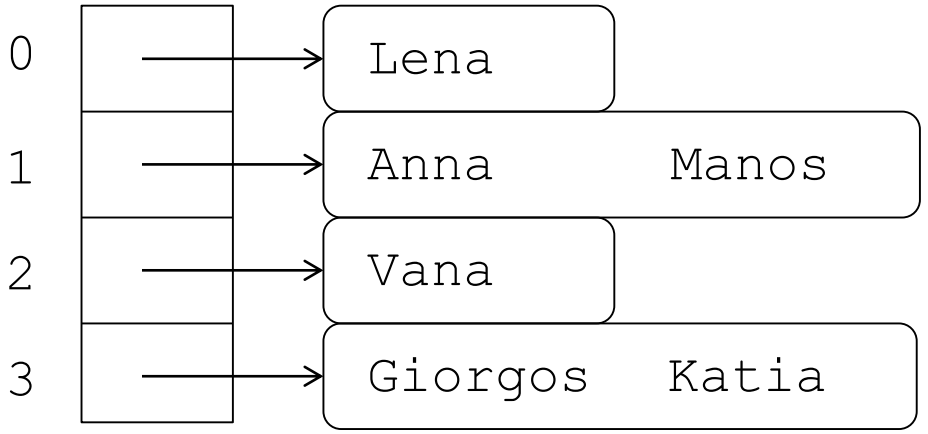


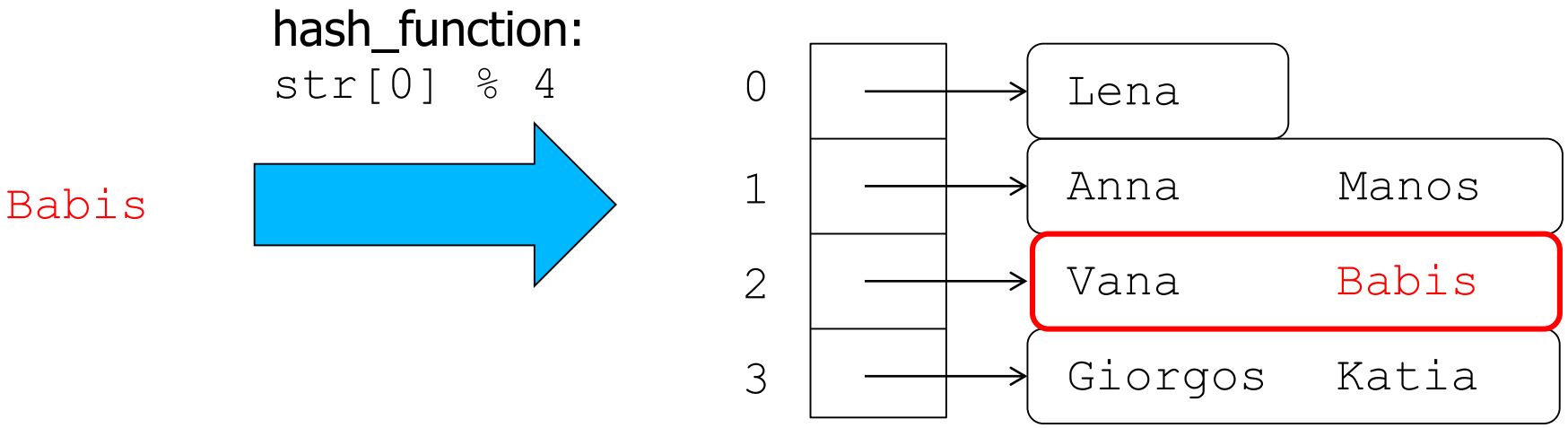

hash_function:
str[0] % 4
Katia
0
1
2
3
Lena
Anna
Manos
Vana
Giorgos
Katia

hash_function:

```
str[0] % 4
```

Babis

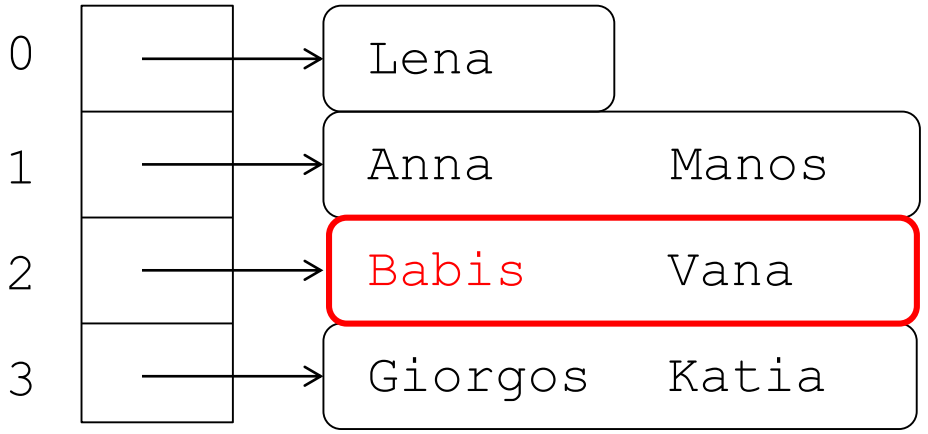



hash_function:
str[0] % 4
Babis
0
1
2
3
Lena
Anna
Manos
Vana
Babis
Giorgos
Katia

hash_function:

```
str[0] % 4
```

Babis



Απόδοση

- Επιλογή **καλής** συνάρτησης κατακερματισμού
  - ομοιόμορφη κατανομή των τιμών των κλειδιών στα δοχεία

- Ιδανικά, αν ο πίνακας είναι αρκετά μεγάλος για να χωρέσει (σχεδόν όλες) τις εγγραφές, οι πράξεις προσθήκης, αναζήτησης, διαγραφής είναι $O(1)$

- Όσο πιο **ανομοιόμορφα** γεμίζουν τα δοχεία (τόσο περισσότερες εγγραφές καταλήγουν στα ίδια δοχεία), τόσο περισσότερο η απόδοση του πίνακα κατακερματισμού τείνει προς στην **απόδοση της δομής αναζήτησης μέσα στα δοχεία**

Μετρικές

- Μέγεθος του πίνακα (N)
- Αριθμός εισαγωγών (E)
- Μέγεθος δοχείων (B)

- **Load factor:** E/N
  - ~1 => ο πίνακας είναι υπερ-φορτωμένος
  - <<1 => ο πίνακας είναι υπο-φορτωμένος

- **Max/mean/average bucket:** μέγιστη τιμή B
  - ~1 => γρήγορη αναζήτηση μέσα στα δοχεία
  - >>1 => αργή αναζήτηση μέσα στα δοχεία

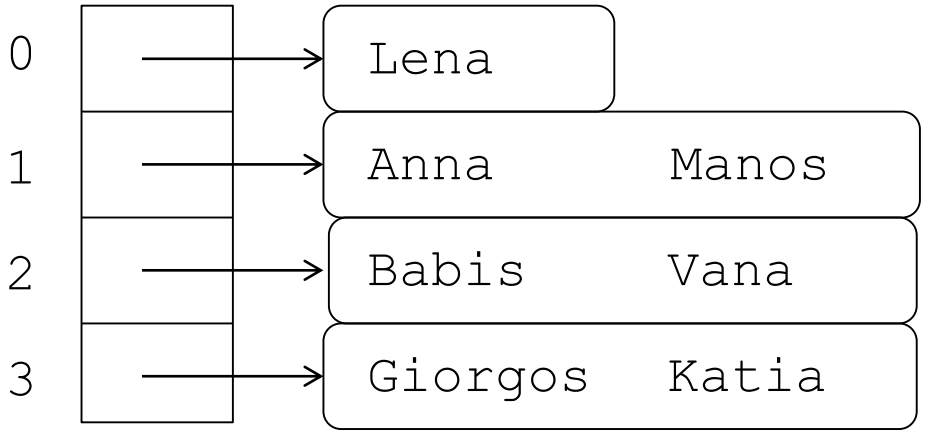
Τακτικές βελτιώση απόδοσης

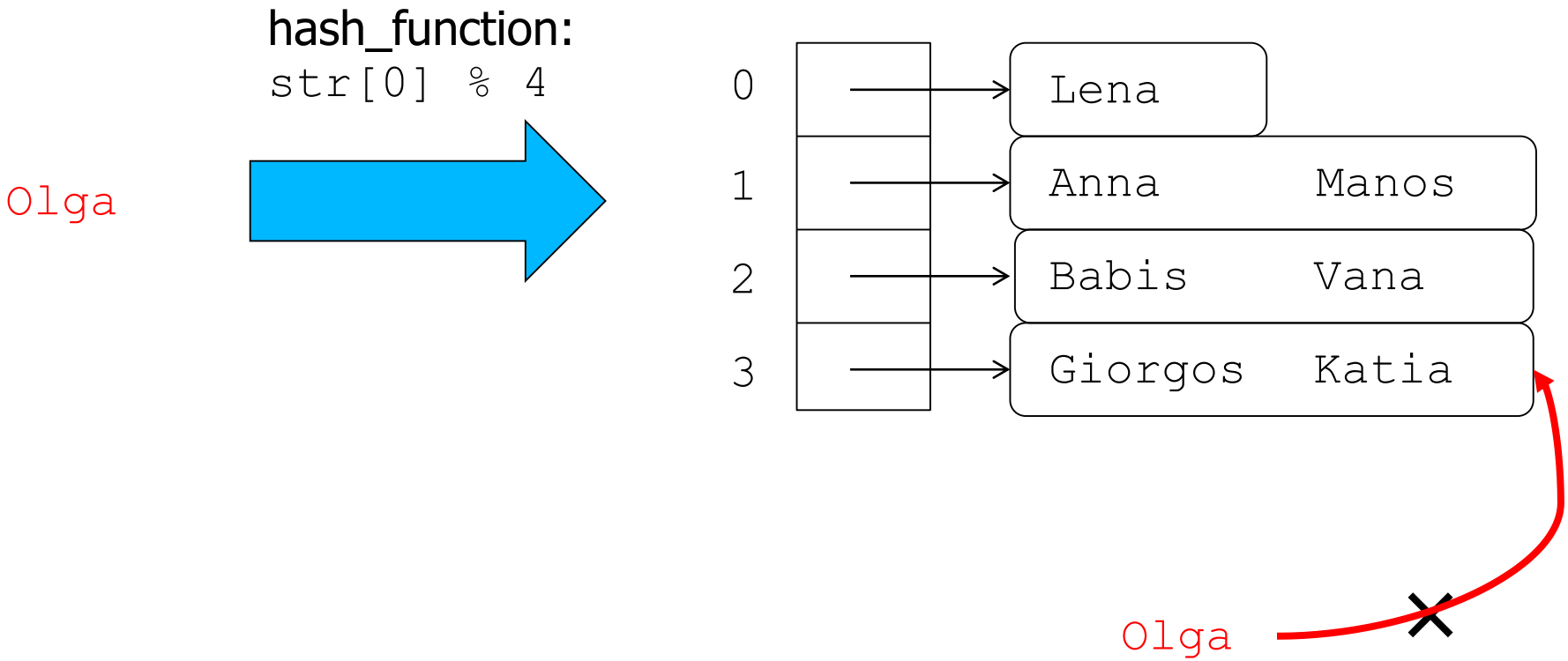
- Αλλαγή/προσαρμογή της συνάρτησης κατακερματισμού
  - όταν Max bucket >> 1 παρότι Load factor <<1
- Δυναμική αύξηση του μεγέθους του πίνακα
  - όταν Load factor ~1
  - οι συγκρούσεις είναι «νομοτελειακά» αναπόφευκτες, όσο καλή και να είναι η συνάρτηση κατακερματισμού
- Δυναμική μείωση του μεγέθους του πίνακα
  - όταν Load factor << 1
  - σπατάλη μνήμης

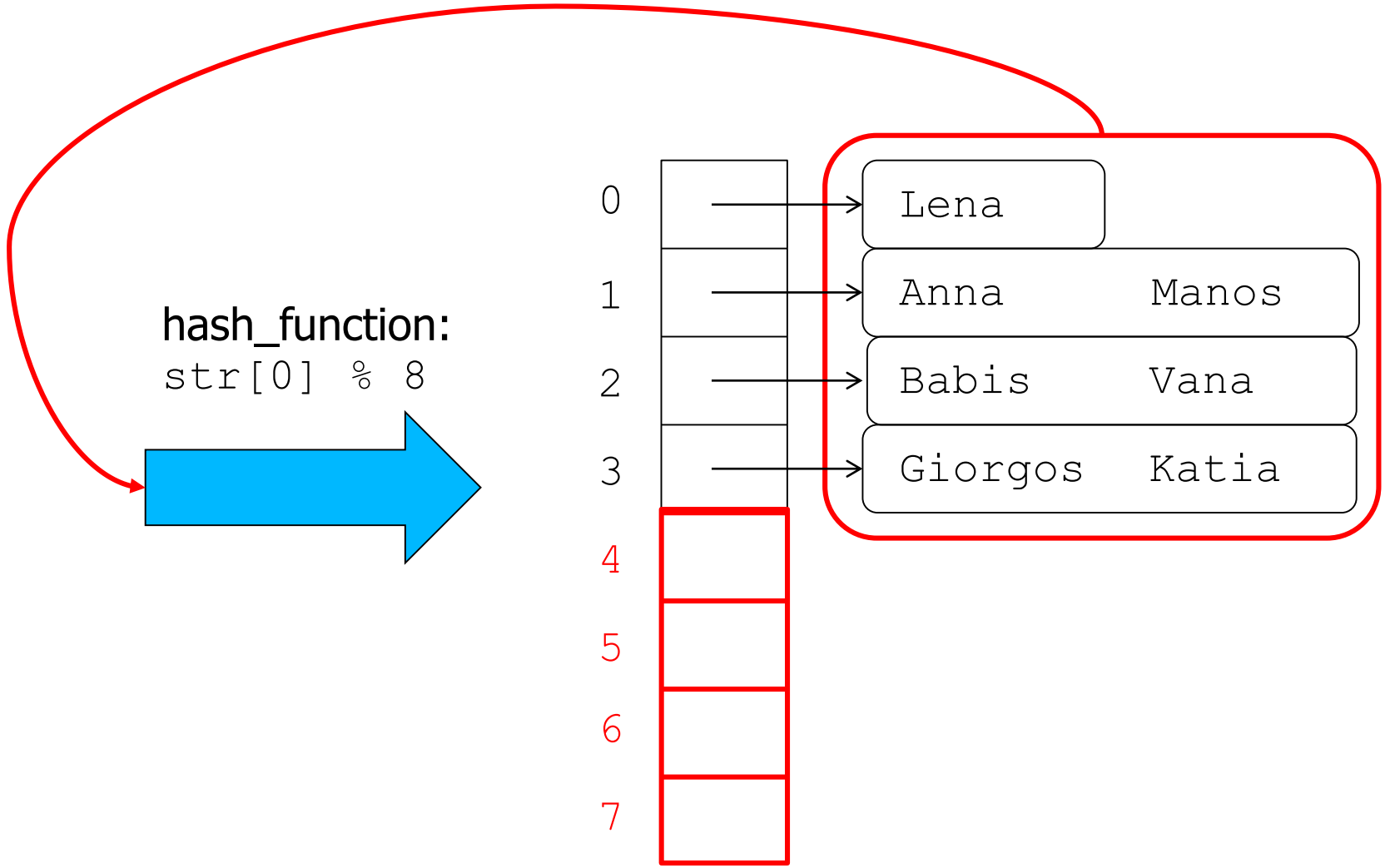
Olga

hash_function:

```
str[0] % 4
```

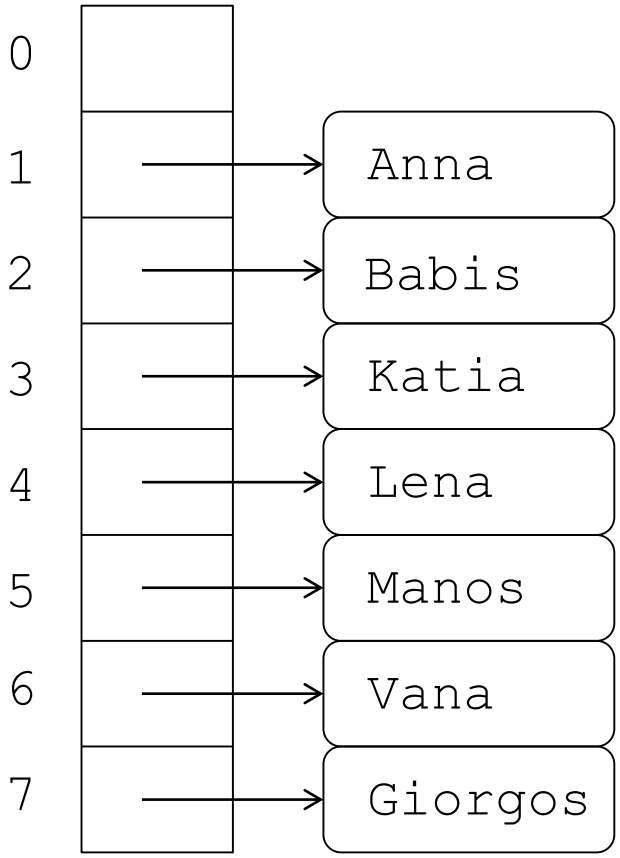


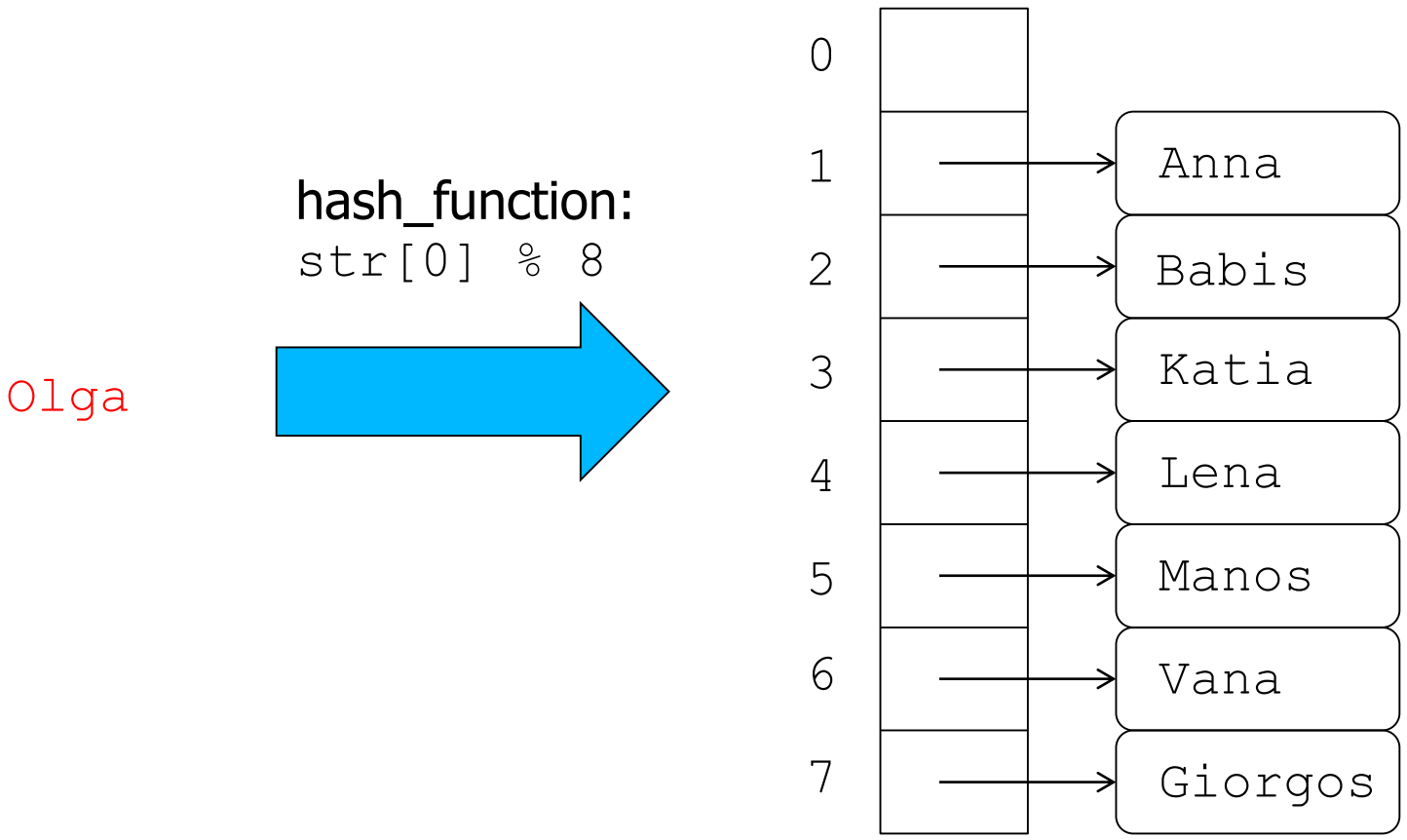

hash_function:
str[0] % 4
Olga
0
1
2
3
Lena
Anna
Manos
Babis
Vana
Giorgos
Katia
Olga

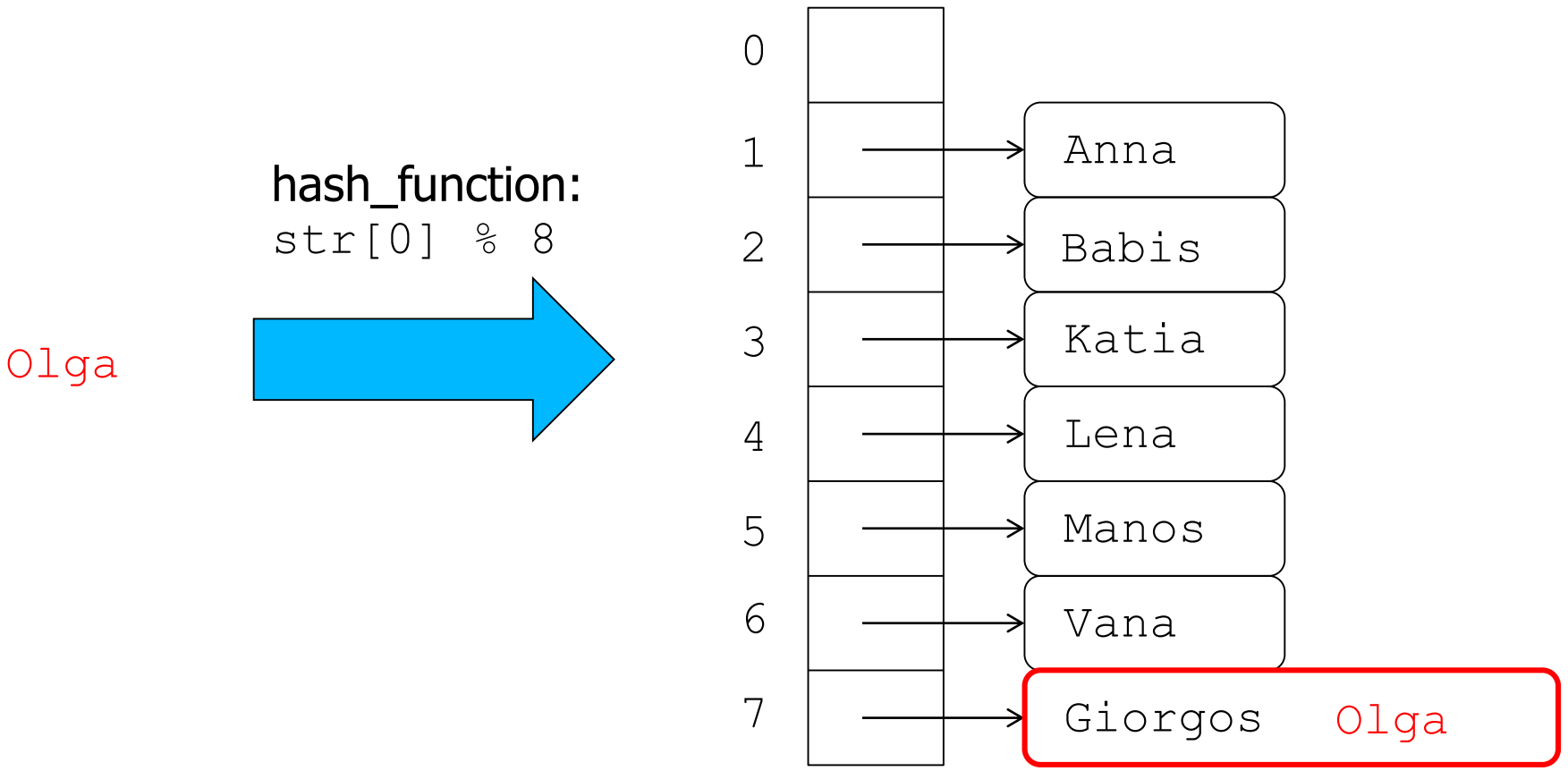

hash_function:
str[0] % 8
0
1
2
3
4
5
6
7
Lena
Anna
Manos
Babis
Vana
Giorgos
Katia

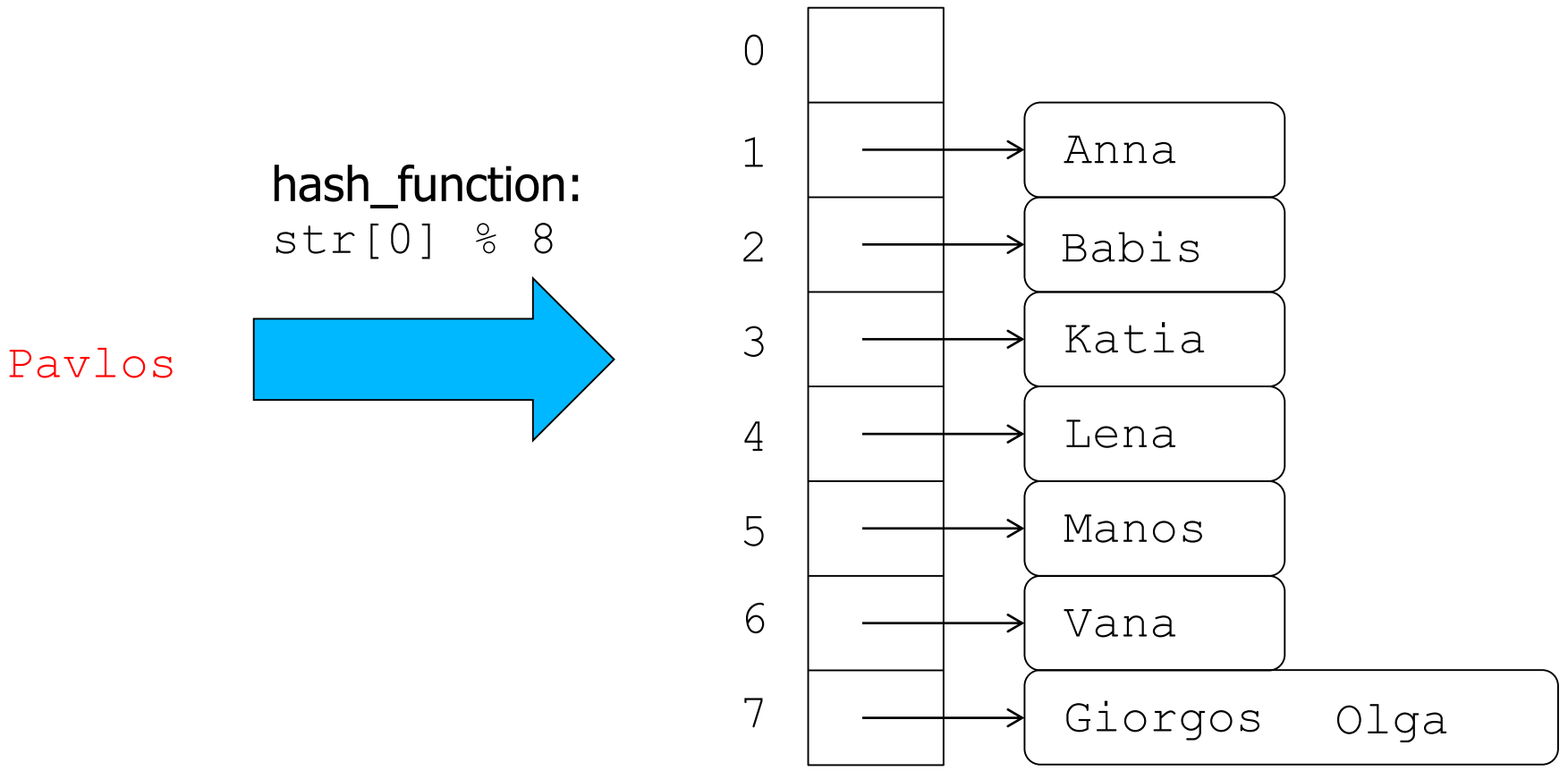
hash_function:

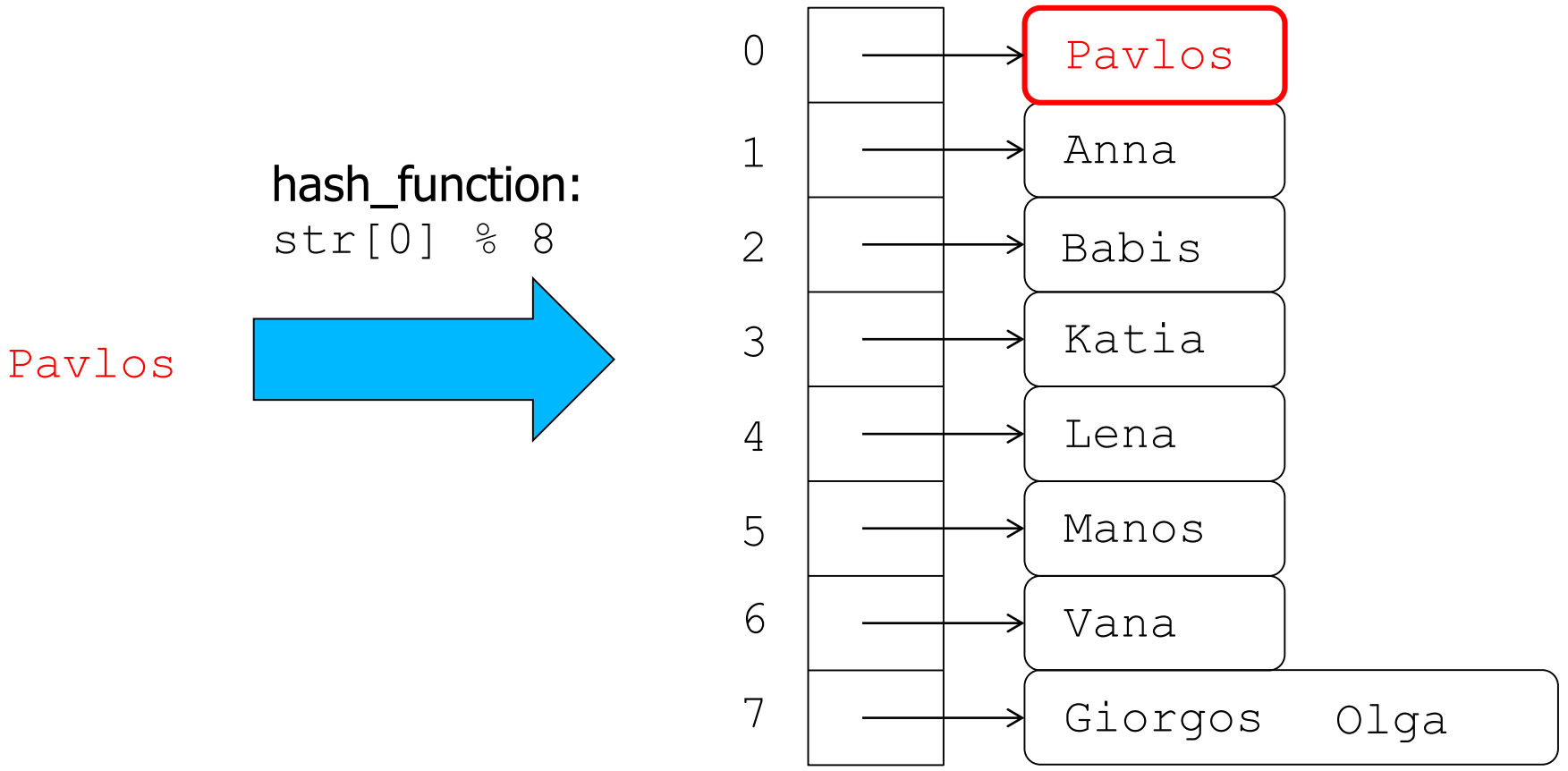
```
str[0] % 8
```

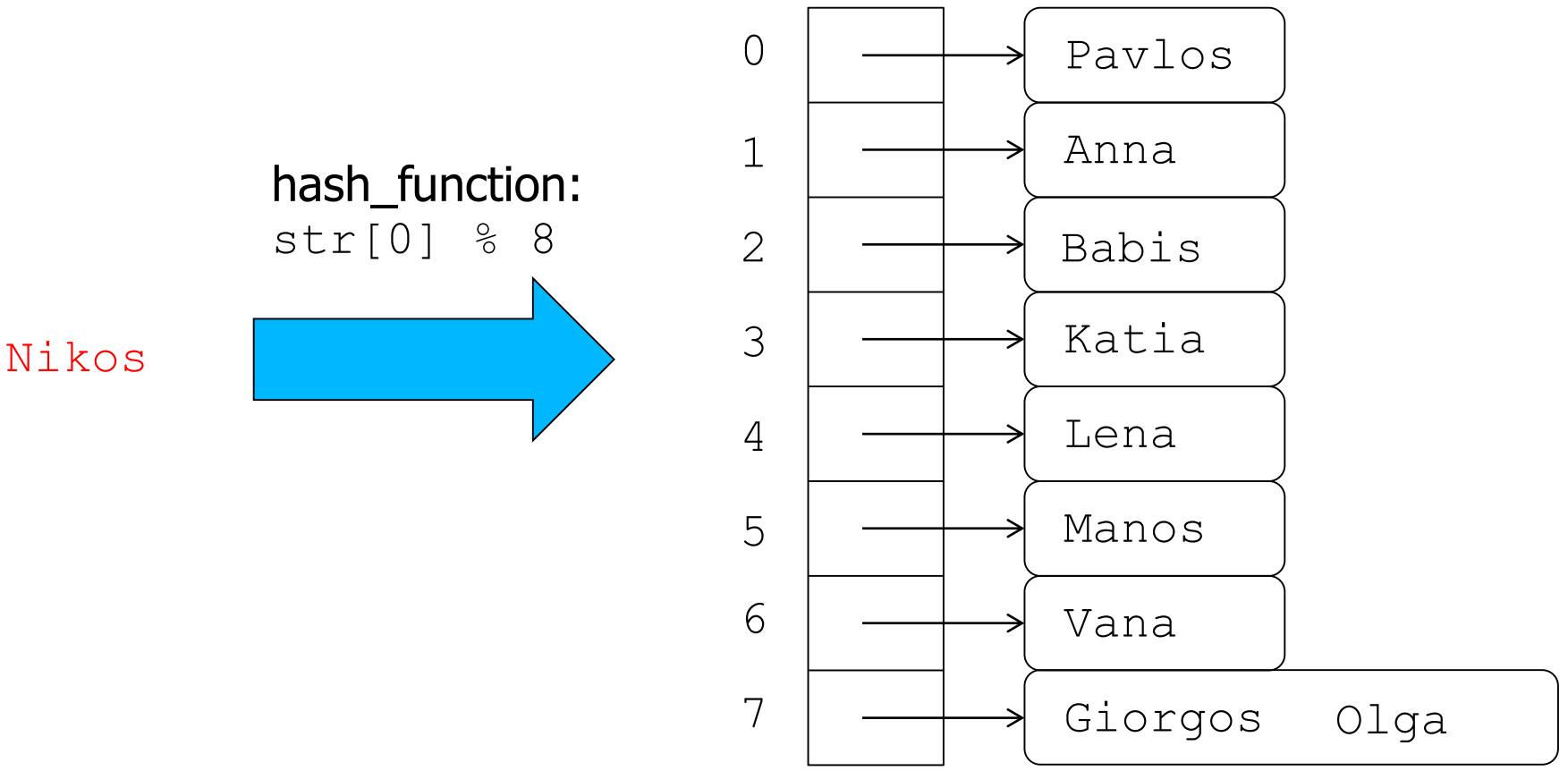



Olga
hash_function:
str[0] % 8
0
1
2
3
4
5
6
7
Anna
Babis
Katia
Lena
Manos
Vana
Giorgos


Olga
hash_function:
str[0] % 8
0
1
2
3
4
5
6
7
Anna
Babis
Katia
Lena
Manos
Vana
Giorgos
Olga

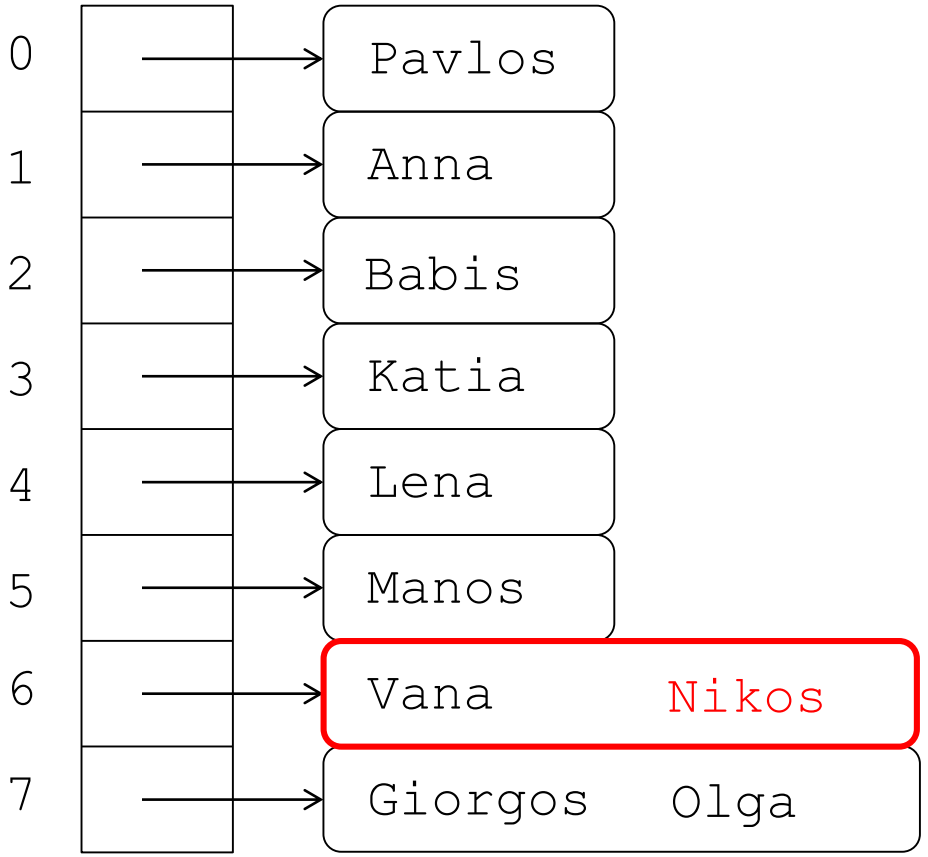

hash_function:
str[0] % 8
Pavlos
0
1
2
3
4
5
6
7
Anna
Babis
Katia
Lena
Manos
Vana
Giorgos
Olga

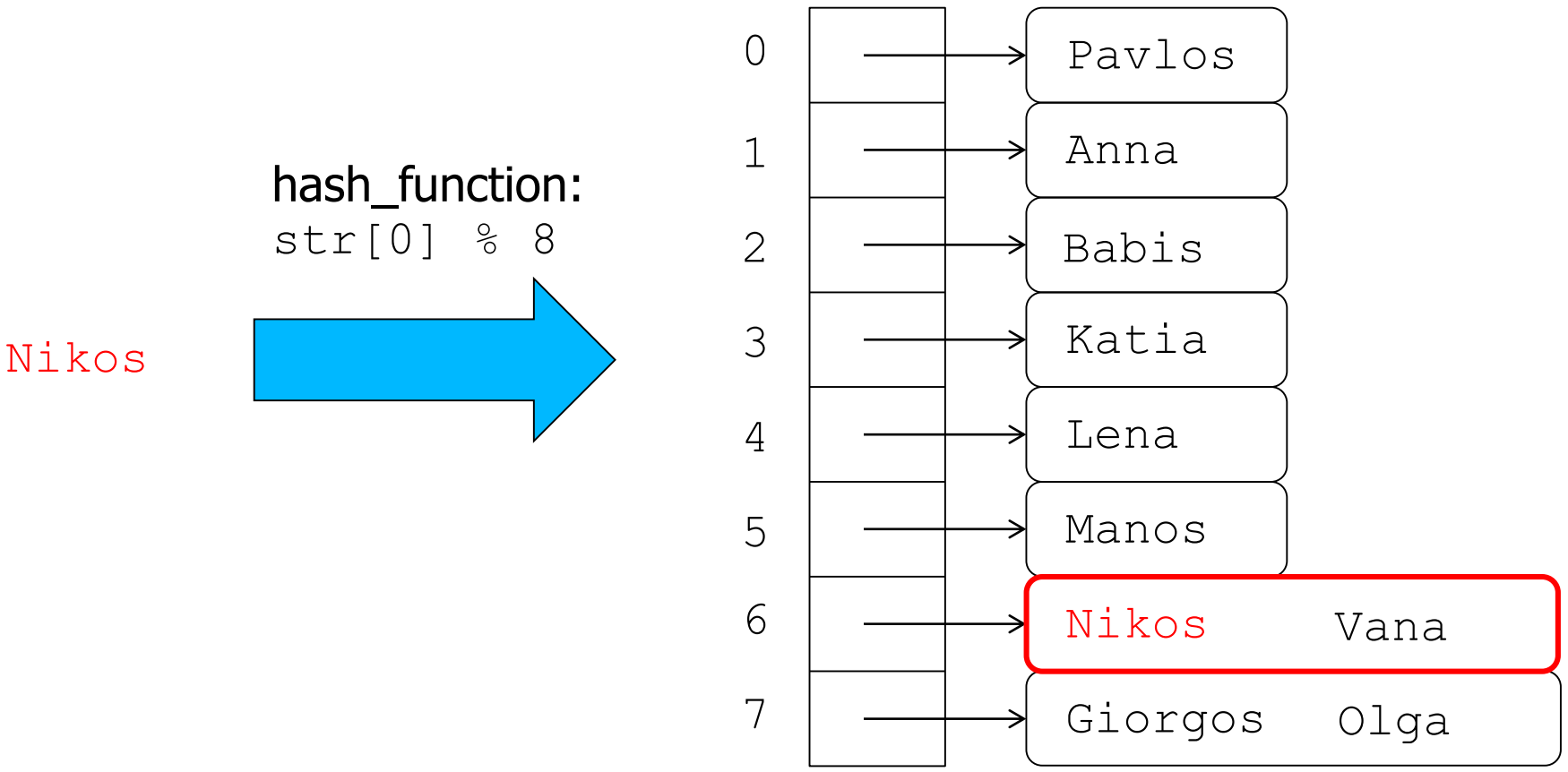

hash_function:
str[0] % 8
Pavlos
0
1
2
3
4
5
6
7
Pavlos
Anna
Babis
Katia
Lena
Manos
Vana
Giorgos
Olga


hash_function:
str[0] % 8
Nikos
0
1
2
3
4
5
6
7
Pavlos
Anna
Babis
Katia
Lena
Manos
Vana
Giorgos
Olga

Nikos

hash_function:
`str[0] % 8`




hash_function:
str[0] % 8
Nikos
0
1
2
3
4
5
6
7
Pavlos
Anna
Babis
Katia
Lena
Manos
Nikos
Vana
Giorgos
Olga

hash_function:

```
str[0] % 8
```