Δομές Κατακερματισμού (Hashing)

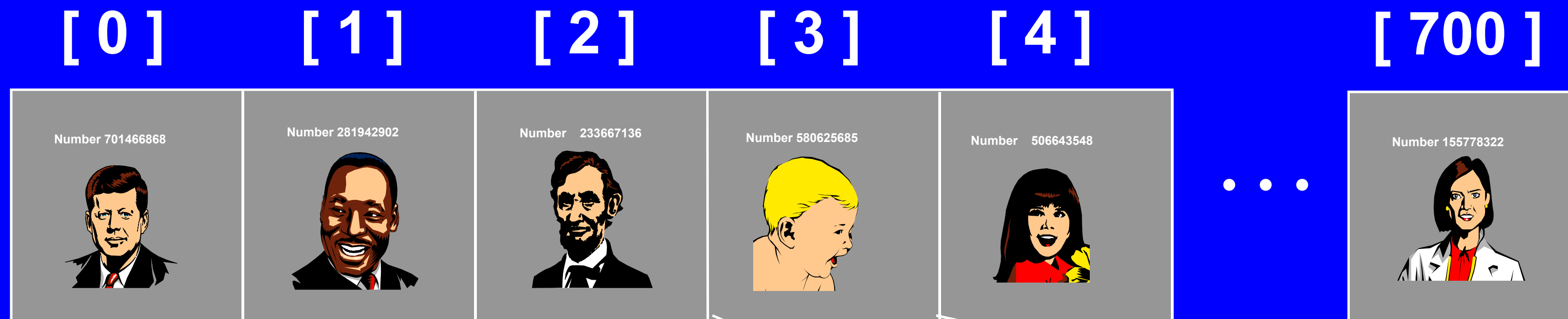
Σ. ΣΙΟΥΤΑΣ
Καθηγητής
Τμήμα Μηχανικών Η/Υ && Πληροφορικής
Πανεπιστήμιο Πατρών

2018-2019

Πρόβλημα

- Δίδεται ένας πίνακας (table) και ένα σύνολο από εγγραφές (records).
- Κάθε εγγραφή σχετίζεται με ένα κλειδί (associated key).
- Επιζητούμε αποδοτική εύρεση/ένθεση/διαγραφή εγγραφής σε/από θέση του πίνακα που σχετίζεται με συγκεκριμένο κλειδί.

Εύρεση



- Κάθε εγγραφή στη λίστα έχει ένα associated key.
- Στο παράδειγμα αυτό, τα κλειδιά είναι τα ID numbers.
- Δεδομένου ότι γνωρίζουμε το key, πως μπορούμε να ανακτήσουμε αποδοτικά την σχετική εγγραφή ?

Μπορούμε κάτι καλύτερο από $O(\log_2 n)$?

- Πολυπλοκότητα μέσης και χειρότερης περίπτωσης για σειριακή αναζήτηση = $O(n)$
- Πολυπλοκότητα μέσης και χειρότερης περίπτωσης για δυαδική αναζήτηση = $O(\log_2 n)$

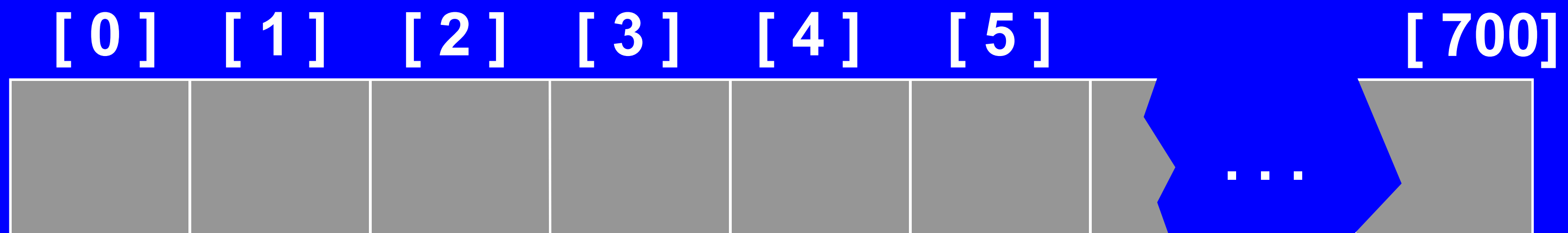
- Μπορούμε να επιτύχουμε κάτι καλύτερο από αυτό?

ΝΑΙ. Χρησιμοποιώντας πίνακες κατακερματισμού (Hash Tables)!

Τι είναι Πίνακας Κατακερματισμού (Hash Table) ?

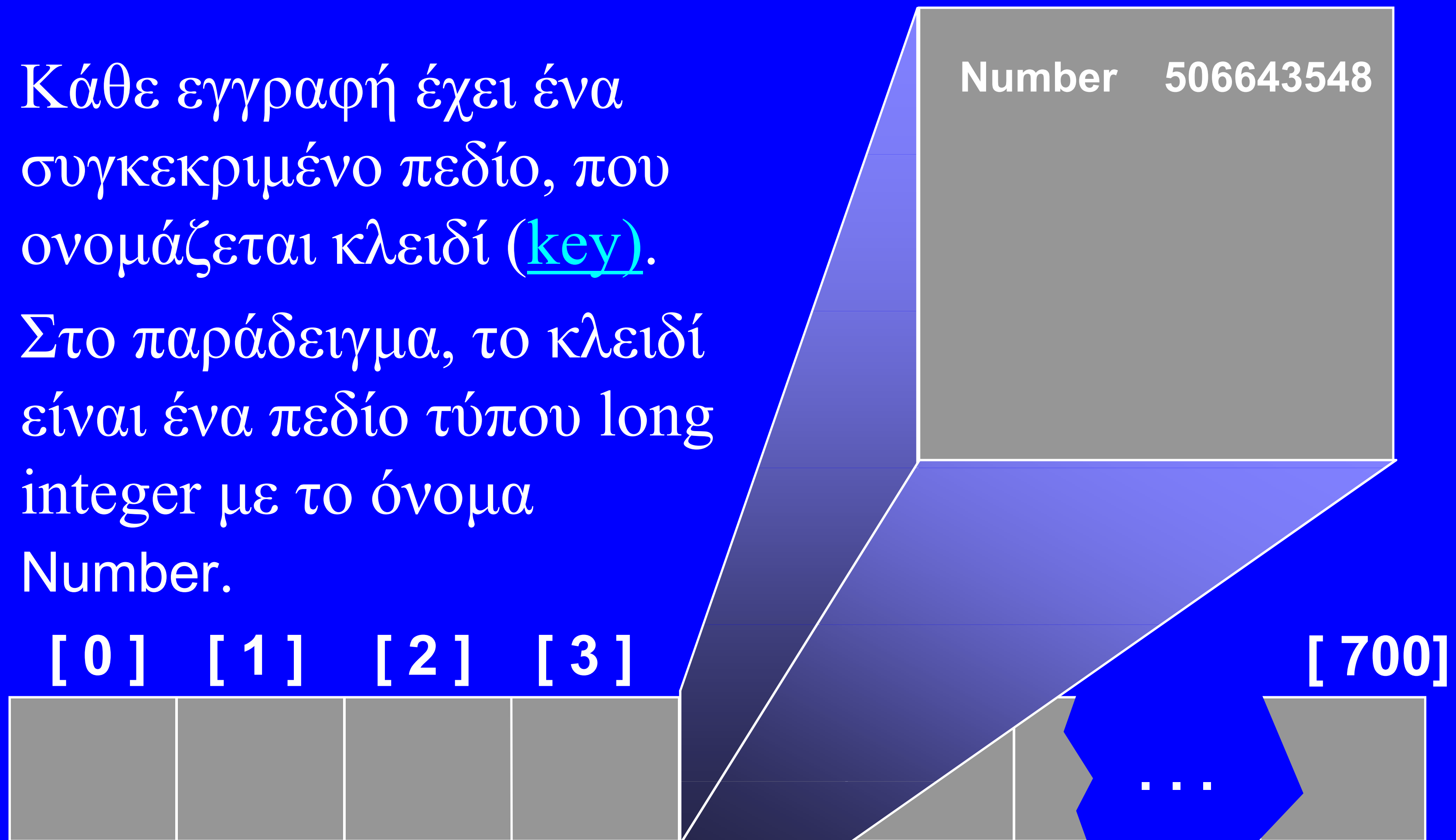
- Η πιο απλή μορφή ενός πίνακα κατακερματισμού είναι ένας πίνακας (array) εγγραφών (ή και αντικειμένων).

- Στο παράδειγμα έχουμε 701 εγγραφές.



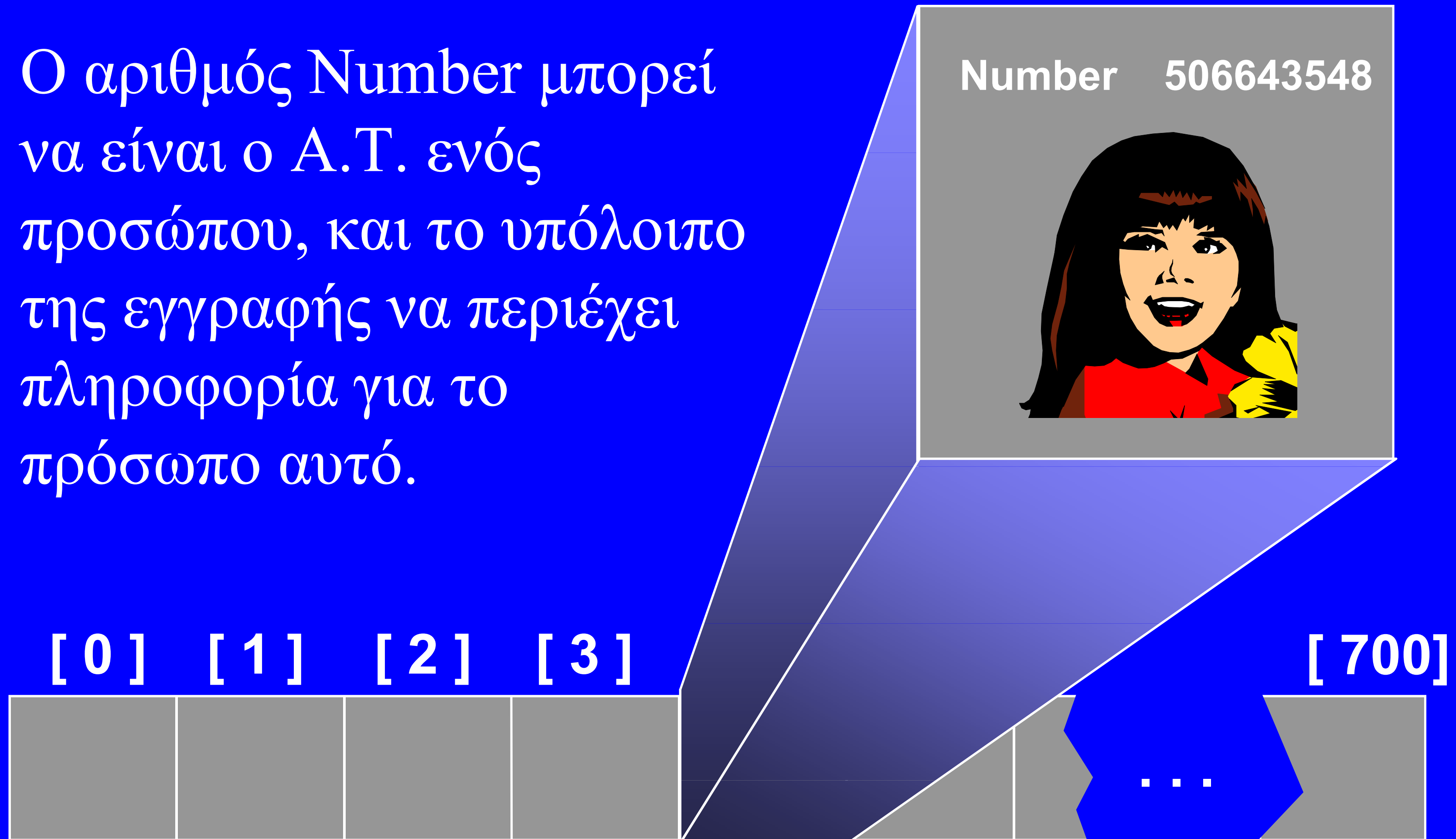
Πίνακας Κατακερματισμού

- Κάθε εγγραφή έχει ένα συγκεκριμένο πεδίο, που ονομάζεται κλειδί (key).
- Στο παράδειγμα, το κλειδί είναι ένα πεδίο τύπου long integer με το όνομα Number.



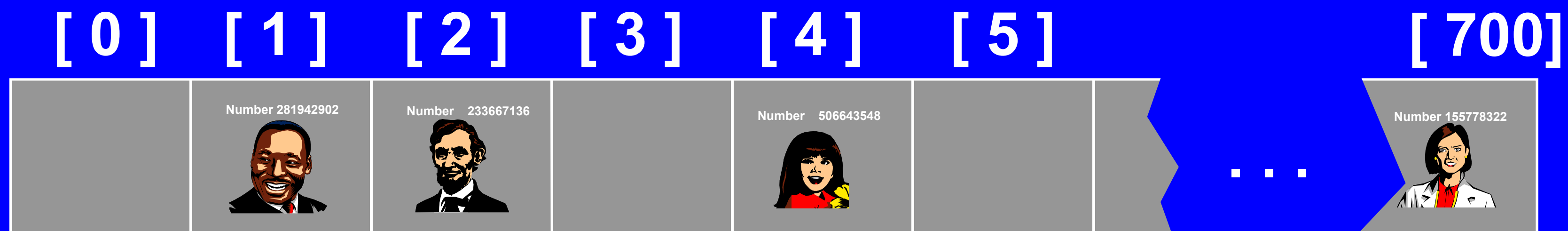
Πίνακας Κατακερματισμού

- Ο αριθμός Number μπορεί να είναι ο Α.Τ. ενός προσώπου, και το υπόλοιπο της εγγραφής να περιέχει πληροφορία για το πρόσωπο αυτό.



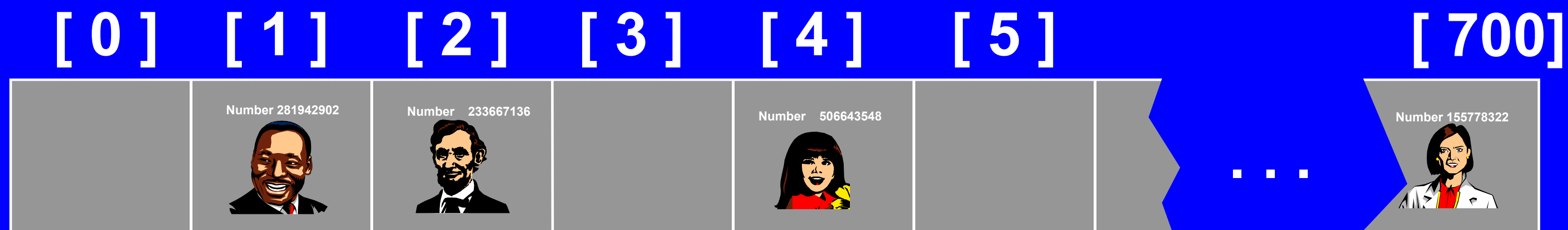
Πίνακας Κατακερματισμού

- Όταν χρησιμοποιούμε πίνακα κατακερματισμού, κάποιες θέσεις περιέχουν έγκυρες εγγραφές (valid records), και κάποιες άλλες είναι "άδειες" ("empty").



Κατακερματισμός Ανοιχτής Διευθυνσιοδότησης (Open Address Hashing)

- Για να ενθέσουμε (insert) μία νέα εγγραφή, το **κλειδί** πρέπει κάπως να **μετατραπεί σε** θέση πίνακα (**array index**).
- Η θέση αυτή ονομάζεται τιμή κατακερματισμού (**hash value)** του κλειδιού.

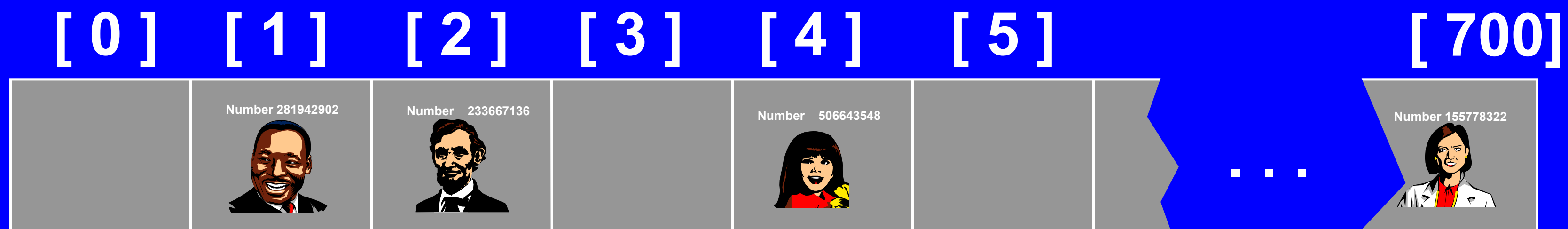


Ενθέτοντας μία νέα εγγραφή

- Ο τυπικός τρόπος να δημιουργήσουμε μία τιμή κατακερματισμού (hash value) είναι:

  (Number mod 701)

Τι σημαίνει (580625685 % 701) ?

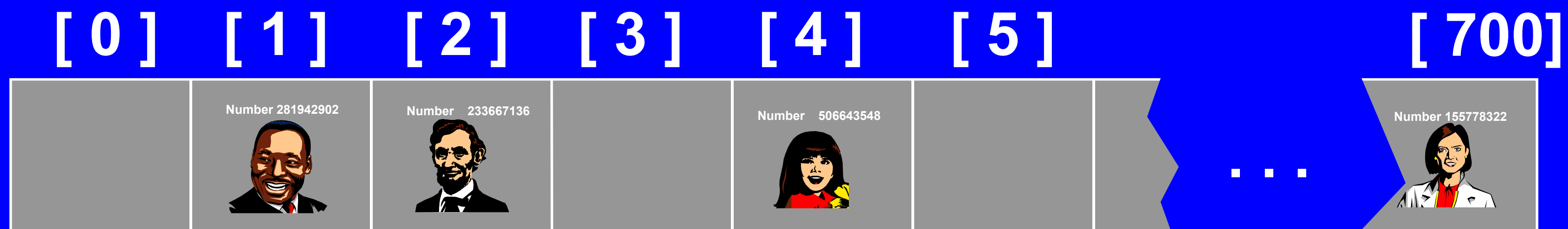


Ενθέτοντας μία νέα εγγραφή

- Τυπικά δημιουργούμε μία hash value:

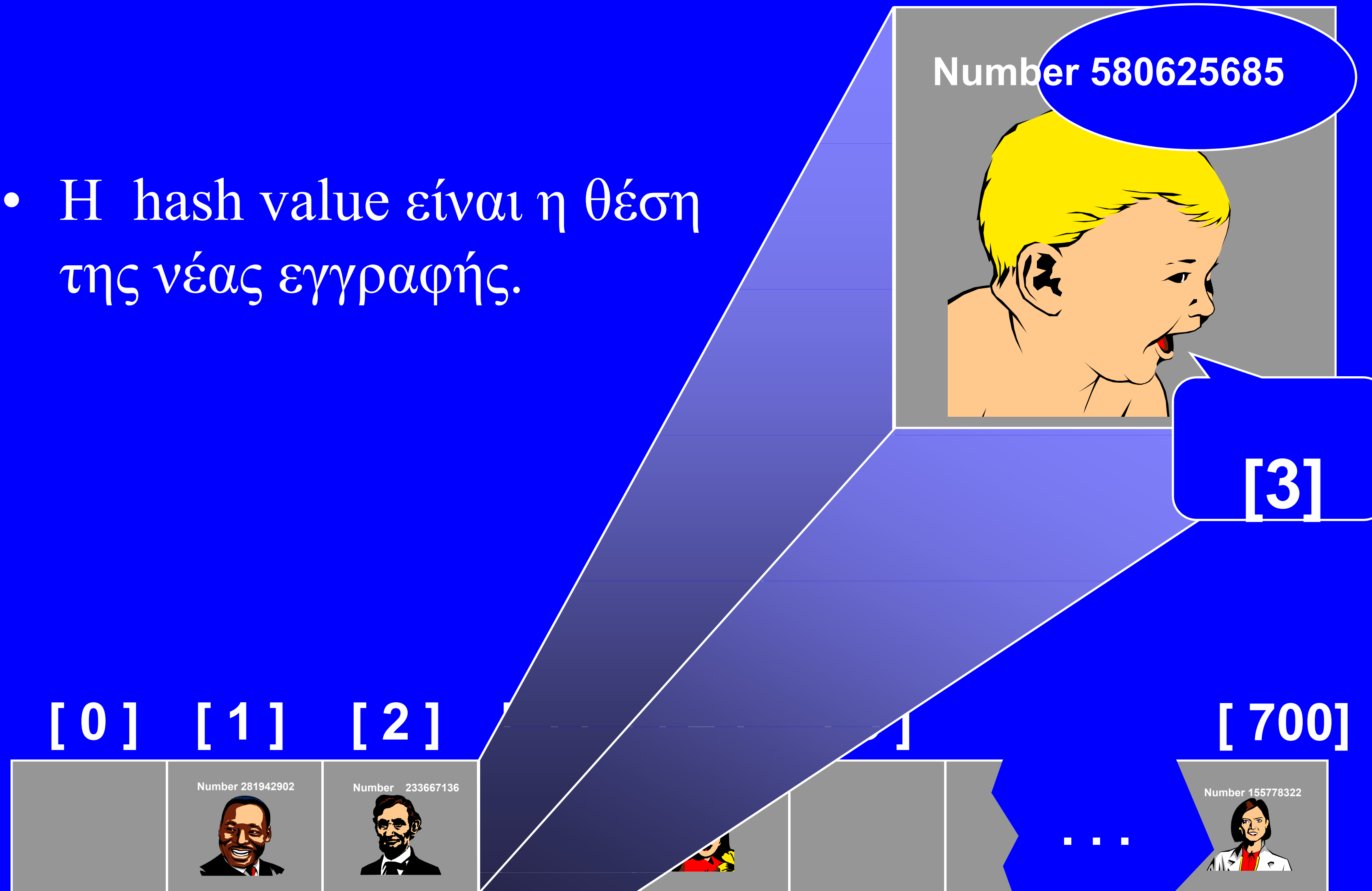
  (Number mod 701)

Ποιο είναι το (580625685 % 701) ?



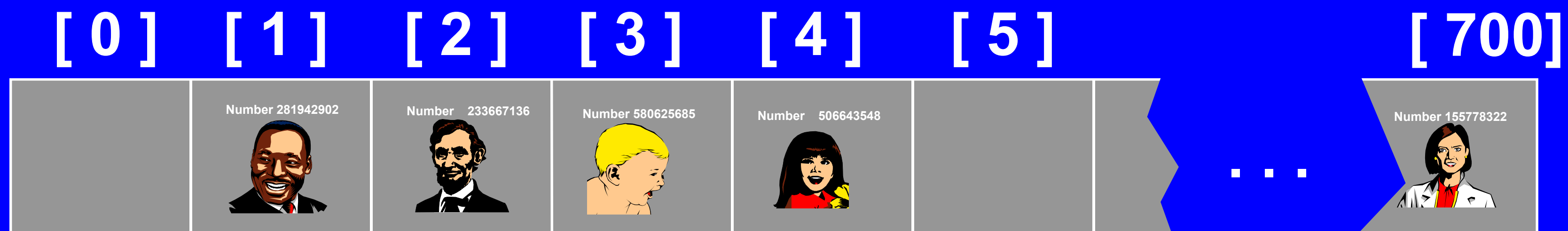
Ενθέτοντας μία νέα εγγραφή

- Η hash value είναι η θέση της νέας εγγραφής.



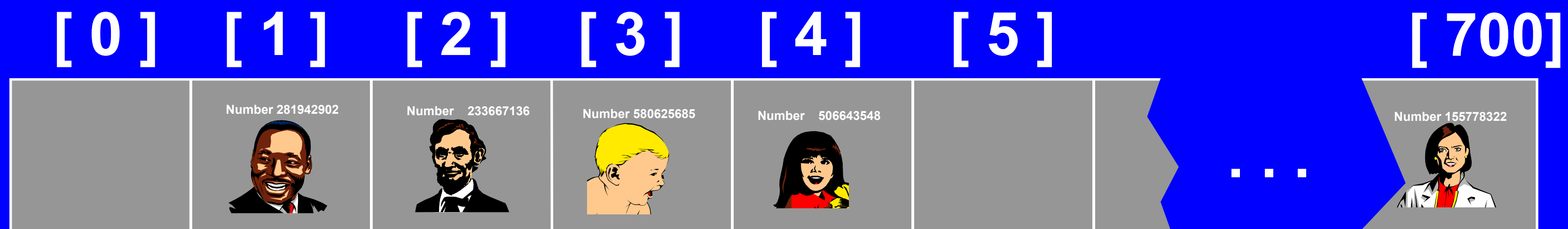
Ενθέτοντας μία νέα εγγραφή

- Η hash value είναι η θέση της νέας εγγραφής.



Συγκρούσεις (Collisions)

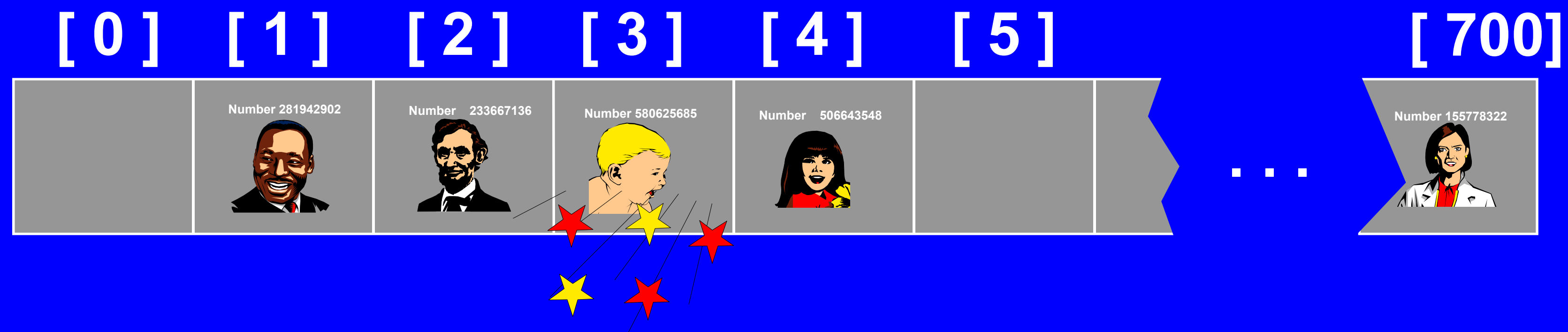
- Θέλουμε να ενθέσουμε μία νέα εγγραφή με hash value=2.



Συγκρούσεις

- Αυτό ονομάζεται **σύγκρουση**, γιατί υπάρχει μία άλλη έγκυρη εγγραφή στη θέση [2].

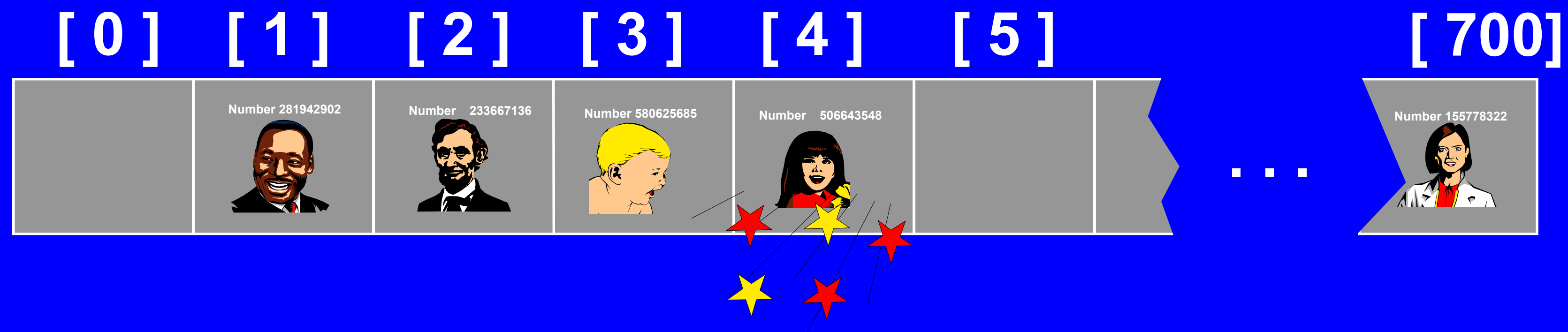
Όταν συμβαίνει σύγκρουση μετακινούμαστε προς τα εμπρός μέχρι να βρούμε άδεια θέση.



Συγκρούσεις

- Αυτό ονομάζεται **σύγκρουση**, γιατί υπάρχει μία άλλη έγκυρη εγγραφή στη θέση [2].

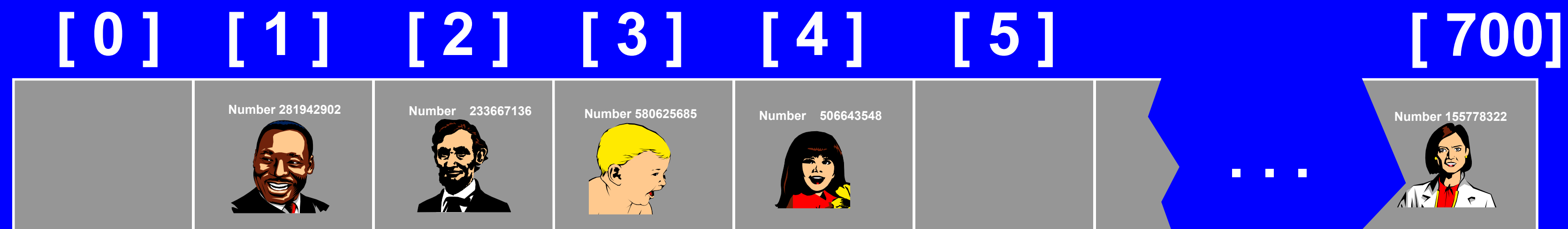
Όταν συμβαίνει σύγκρουση μετακινούμαστε προς τα εμπρός μέχρι να βρούμε άδεια θέση.



Συγκρούσεις

- Αυτό ονομάζεται **σύγκρουση**, γιατί υπάρχει μία άλλη έγκυρη εγγραφή στη θέση [2].

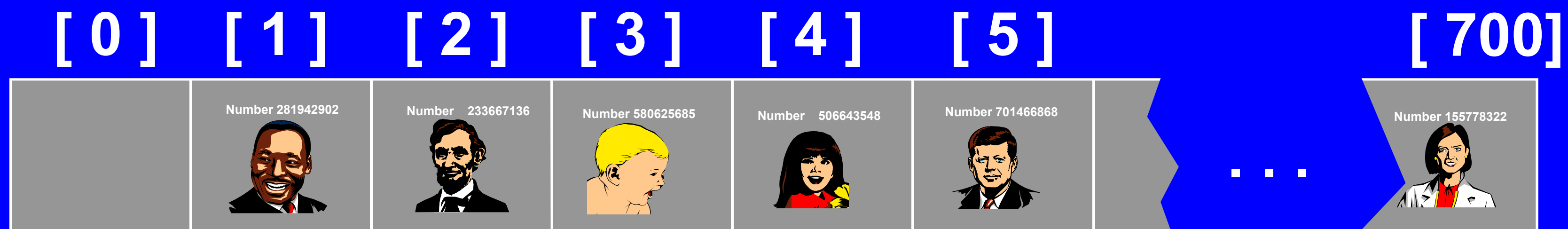
Όταν συμβαίνει σύγκρουση μετακινούμαστε προς τα εμπρός μέχρι να βρούμε άδεια θέση.



Συγκρούσεις

- Αυτό ονομάζεται **σύγκρουση**, γιατί υπάρχει μία άλλη έγκυρη εγγραφή στη θέση [2].

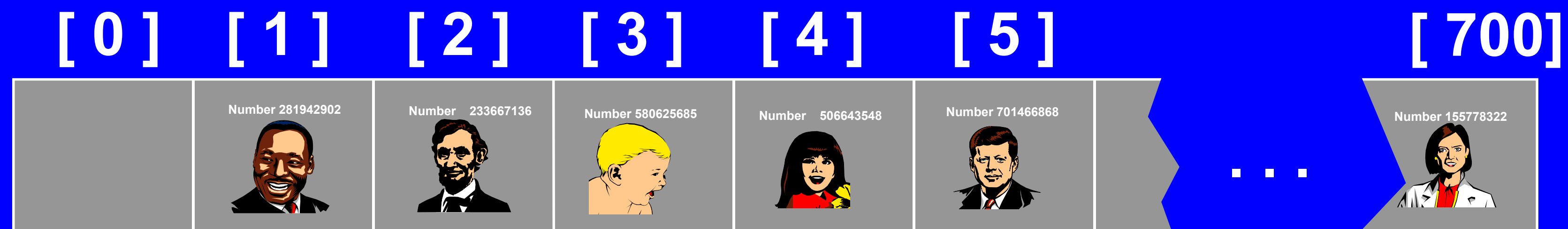
Η νέα εγγραφή αποθηκεύεται στην κενή θέση



Αναζητώντας το Key

- Τα δεδομένα που σχετίζονται με το key πρέπει να ανακτηθούν αρκετά γρήγορα.

Number 701466868



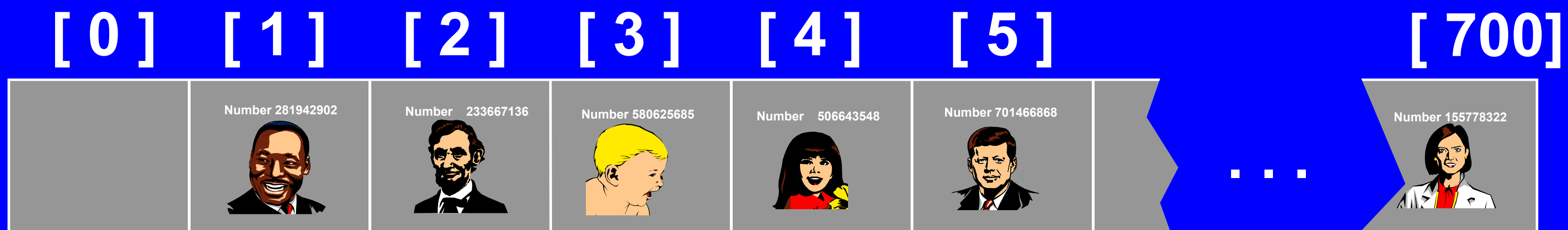
Αναζητώντας το Key

- Υπολόγισε τη hash value.
- Έλεγξε αυτή τη θέση του πίνακα για το κλειδί.

Number 701466868

My hash value is [2].

Not me.



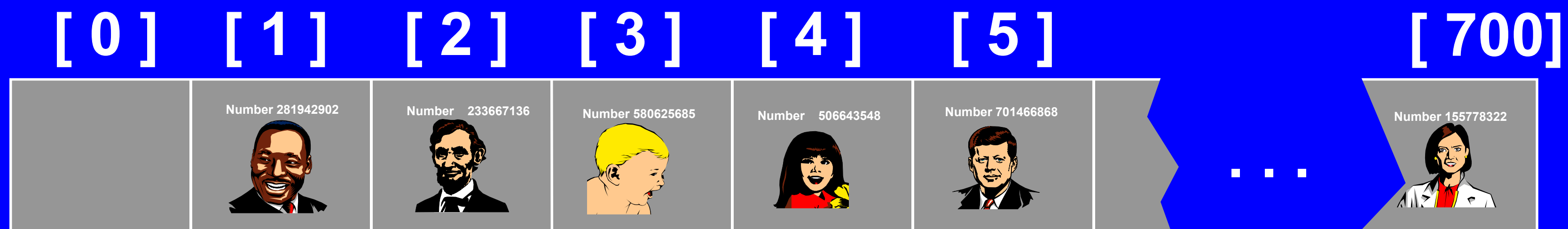
Αναζητώντας το Key

- Συνέχισε να μετακινείσαι προς τα εμπρός (forward) μέχρι να βρεις το key, ή να φτάσεις σε άδειο κελί.

Number 701466868

My hash value is [2].

Not me.



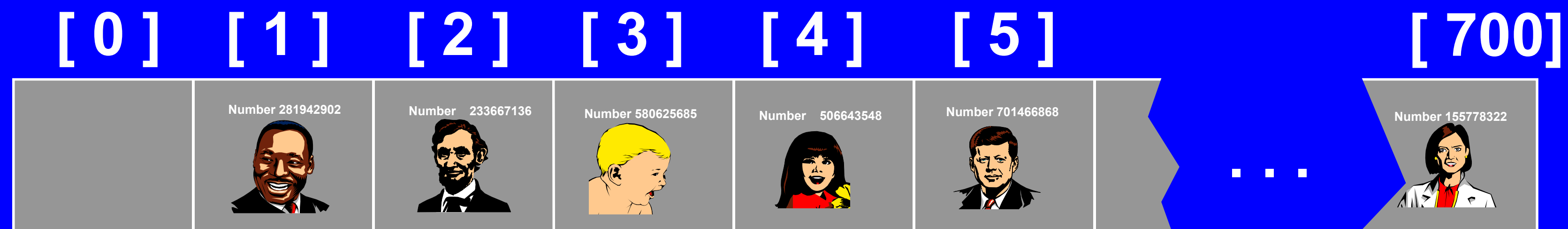
Αναζητώντας το Key

- Συνέχισε να μετακινείσαι προς τα εμπρός (forward) μέχρι να βρεις το key, ή να φτάσεις σε άδειο κελί.

Number 701466868

My hash value is [2].

Not me.



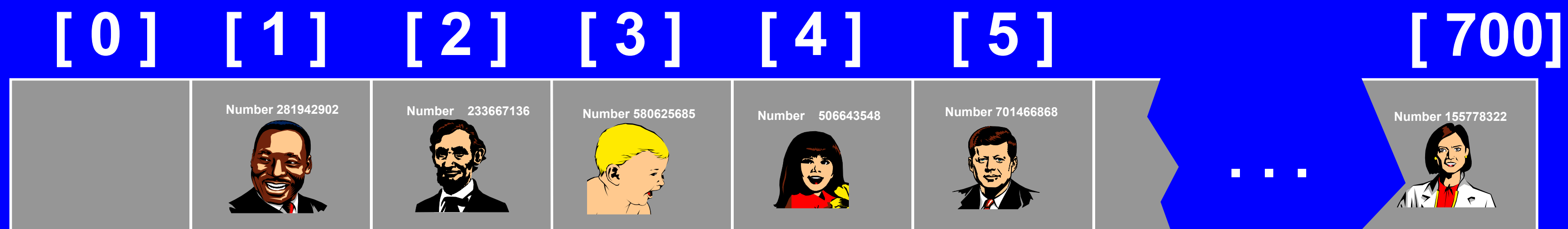
Αναζητώντας το Key

- Συνέχισε να μετακινείσαι προς τα εμπρός (forward) μέχρι να βρεις το key, ή να φτάσεις σε άδειο κελί.

Number 701466868

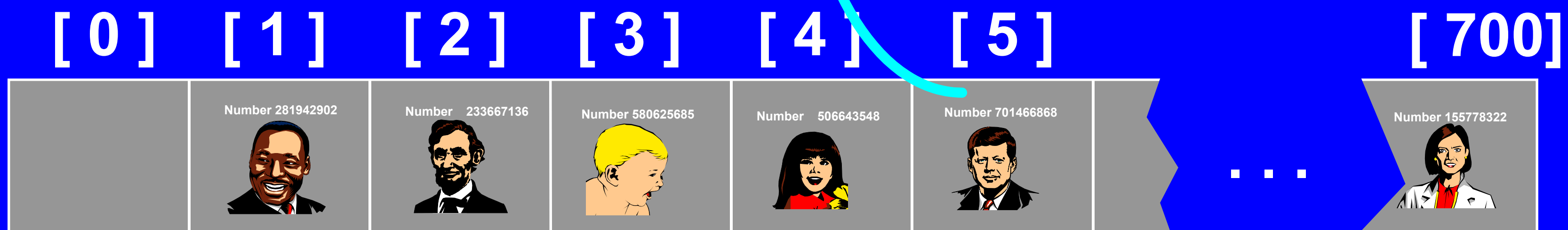
My hash value is [2].

Yes!



Αναζητώντας το Key

- Όταν εντοπιστεί η εγγραφή, η πληροφορία ανακτάται και αντιγράφεται στην κατάλληλη θέση μνήμης.



ΕΞΑΣΚΗΣΗ

$h(k) = k\%11$

0	1001
1	9537
2	3016
3	
4	
5	
6	
7	9874
8	2009
9	9875
10	

1. Τι θα συμβεί αν η επόμενη εγγραφή έχει hash value 0?
   → Πήγαινε στη θέση 3

Το ίδιο ισχύει για τις εγγραφές με hash value 1 ή 2!
Μόνο η εγγραφή με hash value 3 θα παραμείνει εκεί.

⇒ $p = 4/11$ η πιθανότητα η επόμενη εγγραφή να πάει στο στη θέση 3

2. Ομοίως, εγγραφές που κατακερματίζονται στις θέσεις 7,8,9 θα καταλήξουν στη θέση 10
3. Μόνο εγγραφές που κατακερματίζονται στην 4 θα καταλήξουν στην 4 ($p=1/11$); Το ίδιο ισχύει για τις θέσεις 5 και 6.

insert 1052 (h.v.=7)

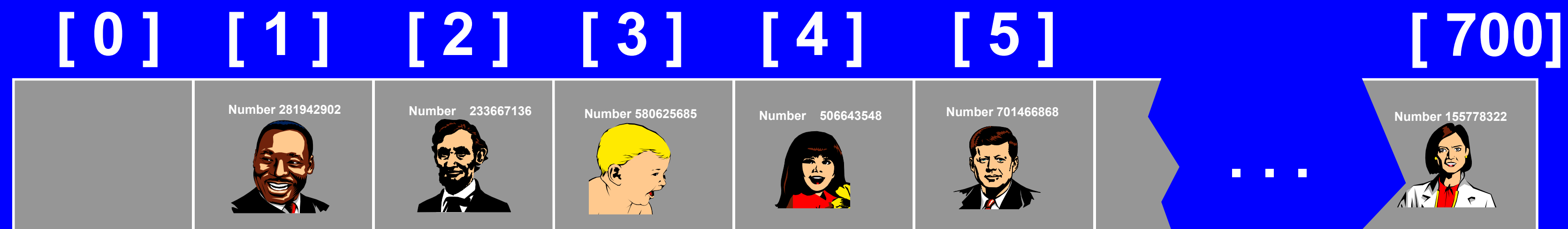
0	1001
1	9537
2	3016
3	
4	
5	
6	
7	9874
8	2009
9	9875
10	1052

Επόμενη εγγραφή στη θέση 3 με $p = 8/11$

Διαγράφοντας μία εγγραφή

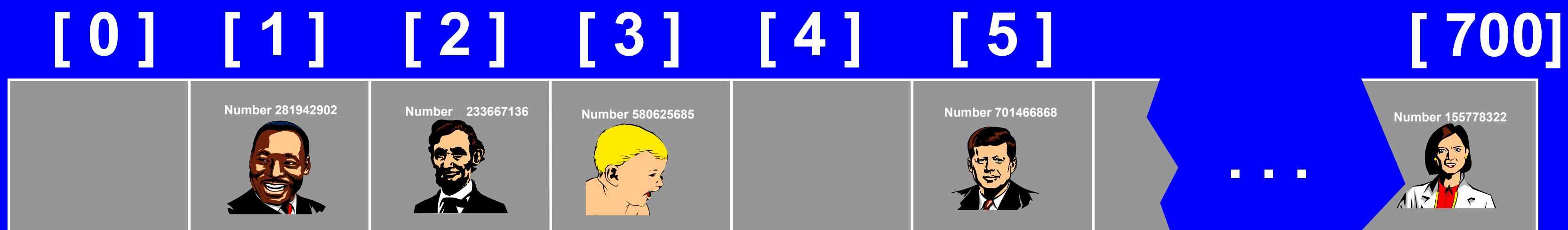
- Εγγραφές μπορούν επίσης να διαγράφονται από έναν πίνακα κατακερματισμού.

Please delete me.



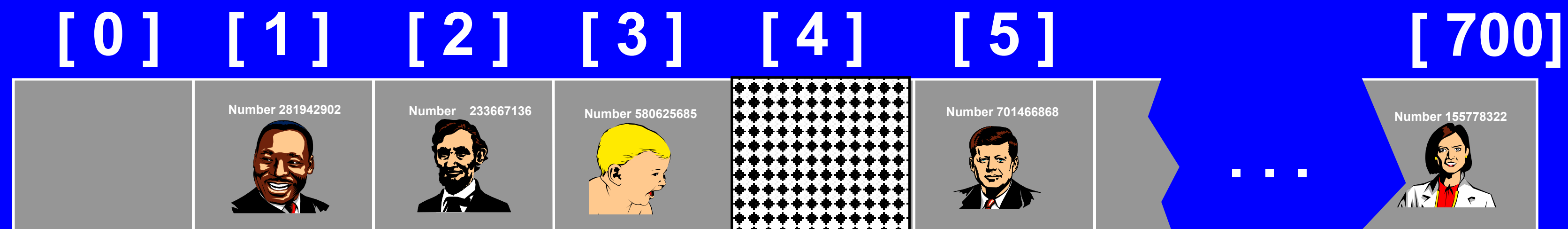
Διαγράφοντας μία εγγραφή

- Εγγραφές μπορούν επίσης να διαγράφονται από έναν πίνακα κατακερματισμού.
- Σε αυτή την περίπτωση η άδεια θέση ΔΕΝ θα πρέπει να θεωρηθεί "κενό κελί", γιατί κάτι τέτοιο θα "μπέρδευε" τις μελλοντικές αναζητήσεις.



Διαγράφοντας μία εγγραφή

- Εγγραφές μπορούν επίσης να διαγράφονται από έναν πίνακα κατακερματισμού.
- Σε αυτή την περίπτωση η άδεια θέση ΔΕΝ θα πρέπει να θεωρηθεί "κενό κελί", γιατί κάτι τέτοιο θα "μπέρδευε" τις μελλοντικές αναζητήσεις.
- Αντιθέτως, θα πρέπει να μαρκαριστεί με κάποιον συγκεκριμένο τρόπο ώστε πιθανή επόμενη αναζήτηση να μπορεί να αντιληφθεί ότι στη θέση αυτή κάτι προϋπήρχε και να μην τερματιστεί απρόσμενα!!!



Κατακερματισμός

- Οι πίνακες κατακερματισμού αποθηκεύουν συλλογές εγγραφών με keys.
- Η θέση της εγγραφής εξαρτάται από τη hash value του κλειδιού key.
- Ανοιχτή Διευθυνσιοδότηση:
  - Όταν συμβαίνει σύγκρουση, η επόμενη διαθέσιμη θέση χρησιμοποιείται.
  - Γενικά, η αναζήτηση για συγκεκριμένο κλειδί γίνεται ταχύτατα.
  - Όταν μία εγγραφή διαγράφεται , η θέση μαρκάρεται με ειδικό τρόπο, ώστε οι διαδικασίες αναζήτησης να γνωρίζουν ότι η θέση αυτή κάποτε χρησιμοποιούνταν, και να ΜΗΝ τερματίζουν απρόσμενα!!!

Ανοιχτή Διευθυνσιοδότηση

- Για να μειώσουμε το φαινόμενο των “συγκρούσεων”, η χωρητικότητα του πίνακα κατακερματισμού (CAPACITY) θα πρέπει να είναι πρώτος αριθμός, π.χ. της μορφής 4k+3
  - Συναρτήσεις Κατακερματισμού:
    - Division hash function: key % CAPACITY
    - Mid-square function: (key*key) % CAPACITY
    - Multiplicative hash function: Το κλειδί key πολλαπλασιάζεται από μία θετική σταθερά μικρότερη της μονάδας. Η Hash συνάρτηση επιστρέφει κάποια από τα πρώτα ψηφία του κλασματικού αποτελέσματος.

Εφαρμογή Συνάρτησης Κατακερματισμού σε συμβολοσειρές

- Folding Method:

```
int h(String x, int D)
{
int i, sum;
for (sum=0, i=0; i<x.length(); i++)
   sum+= (int)x.charAt(i);
return (sum%D);
}
```

Αθροίζει τις ASCII τιμές των γραμμάτων στη συμβολοσειρά (string).

- Υπολογίζει το modulo του αθροίσματος με την χωρητικότητα του πίνακα κατακερματισμού D.

Εφαρμογή Συνάρτησης Κατακερματισμού σε συμβολοσειρές

- Much better: Cyclic Shift

```
static long hashCode(String key, int D)
{
   int h=0;
   for (int i=0, i<key.length(); i++){
    h = (h << 4) | ( h >> 27);
    h += (int) key.charAt(i);
    }
   return h%D;
}
```

Δημιουργία Συστάδων (Clustering)

- Στη μέθοδο κατακερματισμού που περιγράψαμε, όταν κατά την ένθεση έχουμε σύγκρουση, μετακινούμαστε στον πίνακα προς τα εμπρός, μέχρι να βρεθεί κενή θέση. Η τεχνική αυτή είναι γνωστή και ως: *linear probing*.

- ***Πρόβλημα***: όταν αρκετά διαφορετικά κλειδιά κατακερματίζονται στην ίδια θέση, γειτονικές (συνεχόμενες) θέσεις του πίνακα ΓΕΜΙΖΟΥΝ. Αυτό οδηγεί στο πρόβλημα του *clustering (δημιουργία συνεχόμενων καταλυμένων θέσεων)*.

- Καθώς οι θέσεις του πίνακα που γεμίζουν πλησιάζουν στη μέγιστη χωρητικότητά του (capacity), τα παραπάνω clusters τείνουν να συγχωνευτούν.
- Η τεχνική του linear probing προσπαθεί να εντοπίσει κενή θέση, αλλά ΑΔΥΝΑΤΕΙ!!! Αυτό επιβραδύνει χρονικά την πράξη της ένθεσης κατά πολύ!!!

Διπλός Κατακερματισμός (Double Hashing)

- Μία συνηθισμένη τεχνική για να αποφύγουμε τα clusters είναι το *double hashing*.
- Έστω *hash1 η αρχική (original) συνάρτηση κατακερματισμού.*
- Ορίζουμε και μία δεύτερη συνάρτηση κατακερματισμού *hash2.*

Double hashing algorithm:

1. *Όταν ενθέτουμε ένα item (εγγραφή (struct) ή αντικείμενο (class)), χρησιμοποιούμε την hash1(key) για να προσδιορίσουμε τη θέση ένθεσης i στον πίνακα.*
2. *Αν έχουμε σύγκρουση, χρησιμοποιούμε την hash2(key) για να υπολογίσουμε ΠΟΣΟ ΜΑΚΡΙΑ πρέπει να μετακινηθούμε στον πίνακα προς τα μπρος, ώστε να εντοπίσουμε κενή θέση:*

next location = (i + hash2(key)) % CAPACITY

Διπλός Κατακερματισμός

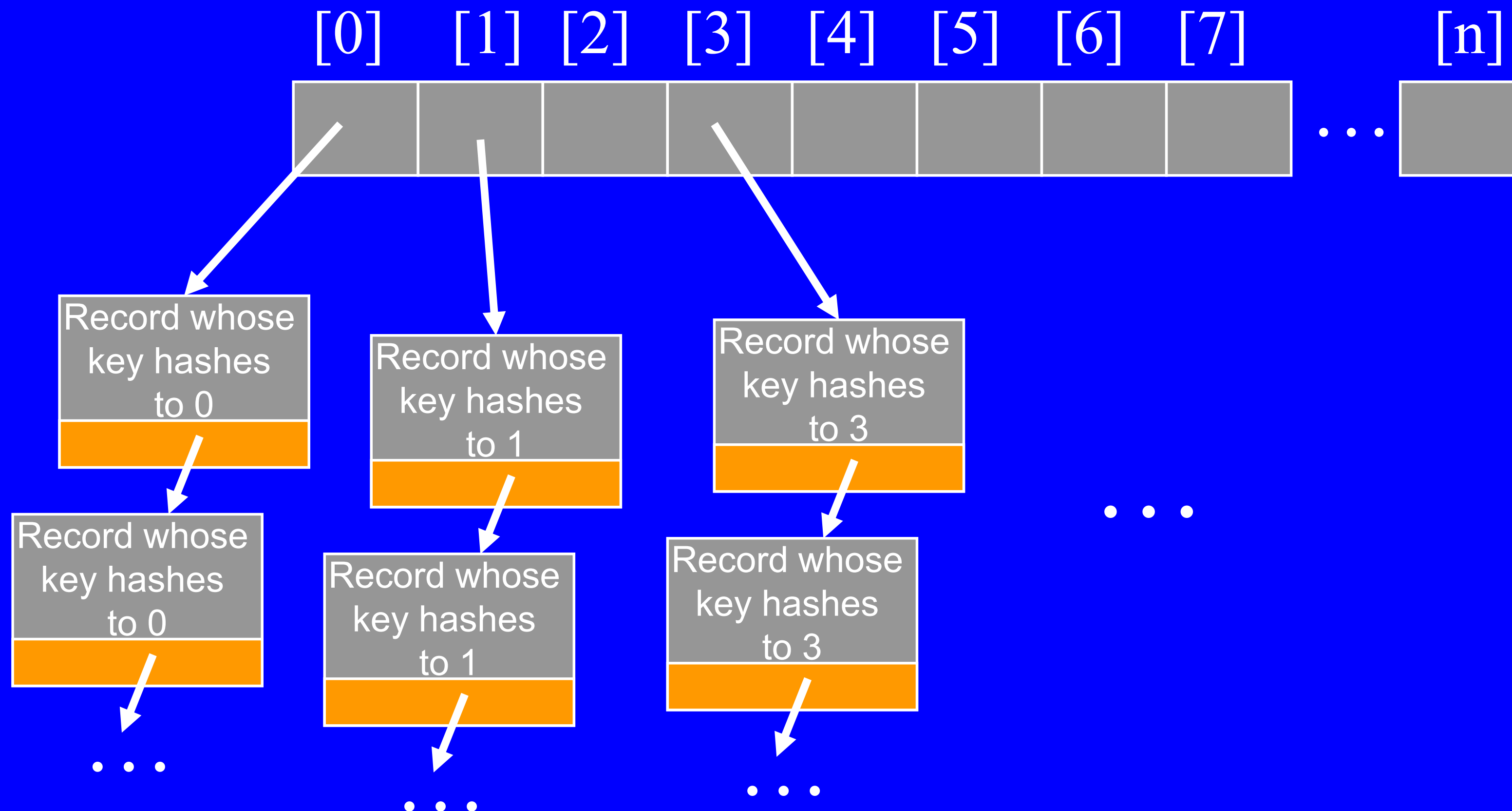
- Το φαινόμενο του Clustering τείνει να μειωθεί, γιατί η hash2() έχει διαφορετικές τιμές για τα keys από αυτές που αρχικά χαρτογραφήθηκαν σε ίδιες θέσεις μέσω της συνάρτησης hash1().
- Αυτό έρχεται σε αντίθεση με τη μέθοδο κατακερματισμού με *linear probing.*
- Και οι δύο ανήκουν στην κατηγορία *open address hashing*, διότι ψάχνουν να εντοπίσουν στον πίνακα την επόμενη κενή θέση.
- Στον κατακερματισμό με linear probing:
  - hash2(key) = (i+1)%CAPACITY
- Στον κατακερματισμό με double hashing η συνάρτηση hash2() μπορεί να είναι μία γενική συνάρτηση της μορφής:
  - hash2(key) = (i+f(key))%CAPACITY

Κατακερματισμός με αλυσίδες (Chained Hashing)

- Στη μέθοδο open address hashing, διαχειριζόμαστε τις συγκρούσεις διερευνώντας στον πίνακα για την επόμενη κενή θέση.
- Όταν ο πίνακας γεμίσει, δεν μπορούμε να προσθέσουμε νέες εγγραφές.
- Μία λύση είναι να επανεκτιμήσουμε το μέγεθος (resizing) του πίνακα → Dynamic Hashing
- Μία άλλη εναλλακτική λύση: Chained Hashing.

Κατακερματισμός με αλυσίδες

- Στην τεχνική αλυσιδωτού κατακερματισμού , κάθε θέση του πίνακα κατακερματισμού δεικτοδοτεί μία λίστα από εγγραφές των οποίων τα κλειδιά κατακερματίζονται στη θέση αυτή:



Κατακερματισμός με αλυσίδες

- Ευελπιστούμε οι αριθμοί των εγγραφών ανά θέση να είναι αυστηρά ισομεγέθεις, ώστε οι λίστες να έχουν μικρό μήκος.

Μέση Ανάλυση Χρόνου:

- *Υποθέτουμε ότι έχουμε D* θέσεις και *n* εγγραφές. Άρα, κατά μέσο όρο έχουμε *n/D* εγγραφές ανά θέση.
- *Οι πράξεις insert, search, delete* απαιτούν O(1+n/D) χρόνο έκαστη.
- Αν επιλέξουμε το D να είναι περίπου n, ο συνολικός χρόνος που απαιτείται είναι O(1).
- Υποθέτοντας ότι κάθε κλειδί κατακερματίζεται σε οποιοδήποτε θέση με την ίδια πιθανότητα, το μέγεθος της λίστας για κάθε θέση γίνεται σταθερό, δηλαδή έχουμε ισομεγέθεις αλυσίδες σταθερού μήκους και ο συνολικός χρόνος στη μέση περίπτωση γίνεται O(1).

Χρονική Ανάλυση Κατακερματισμού

- Χειρότερη Περίπτωση: κάθε κλειδί κατακερματίζεται στην ίδια θέση του πίνακα! O(n) search!!
- Ευτυχώς, η μέση περίπτωση είναι ΠΙΟ ΠΟΛΛΑ ΥΠΟΣΧΟΜΕΝΗ.
- Ορίζουμε ως παράγοντα φόρτου (*load factor) το παρακάτω κλάσμα:*

$$\alpha = \frac{\textit{number of occupied table locations}}{\textit{size of table's array}}$$

Μέση Χρονική Πολυπλοκότητα Αναζήτησης

Στην τεχνική open addressing με linear probing, ο μέσος αριθμός των θέσεων του πίνακα που πρέπει να εξετάσουμε σε μία επιτυχή αναζήτηση είναι κατά προσέγγιση:

$$\frac{1}{2}\left(1+ 1/(1-\alpha)\right)$$

Double hashing: $-\ln(1-\alpha)/\alpha$

Chained hashing: $1+\alpha/2$

Ανάλυση Πολυπλοκότητας Τεχνικής με Linear Probing

Υπόθεση. Η ακολουθία διερεύνησης (*Probe sequence) για το κλειδί k μπορεί να είναι με ίδια πιθανότητα οποιοσδήποτε συνδυασμός από 0,…,D – 1*

Θεώρημα. *Μέσος Αριθμός Διερευνήσεων* ≤1/(1-α)

Απόδειξη.

X := μέσος αριθμός διερευνήσεων (μέχρι και την τελευταία μη-επιτυχή)

Pr [1η μη-επιτυχής διερεύνηση] = n/D=α

Pr [2η μη-επιτυχής διερεύνηση / 1η μη-επιτυχής διερεύνηση] = (n-1)/(D-1) < n/D=α

.

.

Pr [(i-1)th μη-επιτυχής διερεύνηση / όλες οι προηγούμενες (i-2) ήταν μη-επιτυχείς] =
= (n-i+2)/(D-i+2) ≤ n/D=α

$$\Pr[X \geq i] \leq a^{i-1}$$

$$E[X] = \sum_{i=1}^{\infty} \Pr[X \geq i] \leq \sum_{i=1}^{\infty} a^{i-1} = \sum_{i=0}^{\infty} a^{i} = \frac{1}{1-a}$$

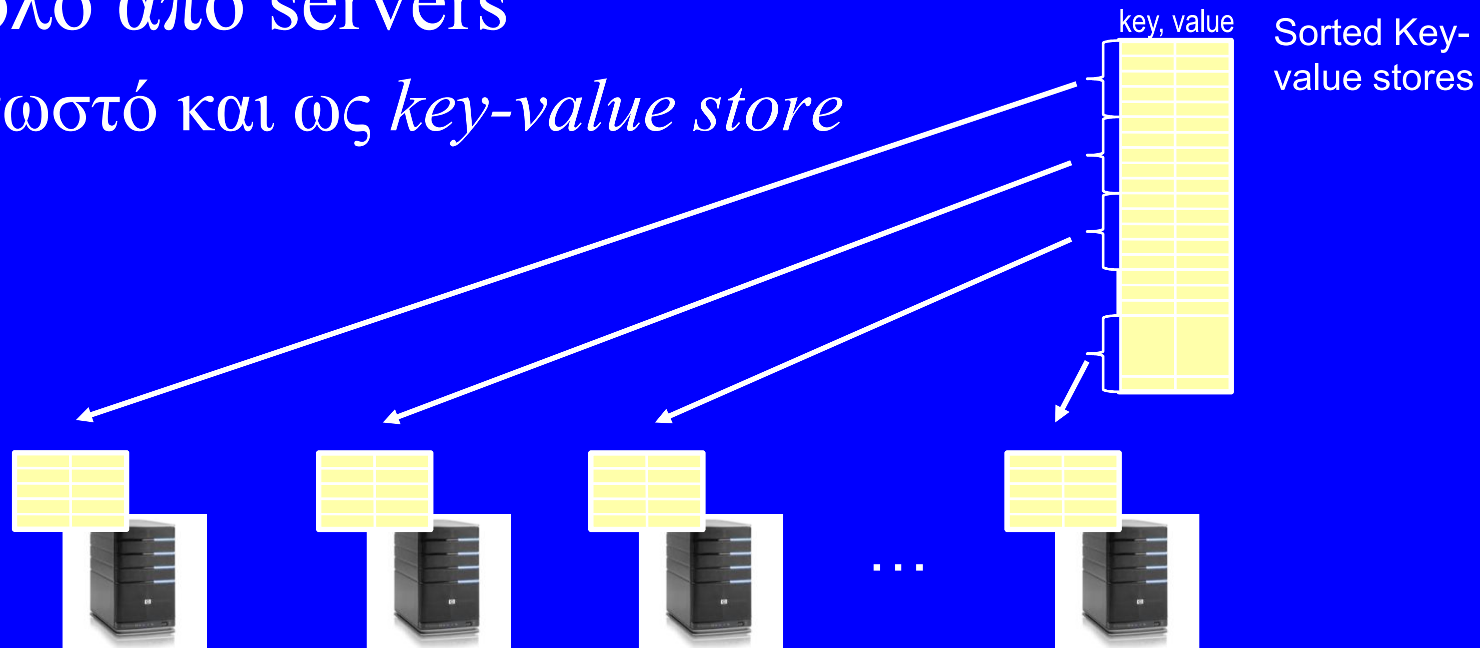
Συνολικά: ½ (1+1/1-α)

Μέσος αριθμός προσπελάσεων στον πίνακα κατακερματισμού κατά τη διάρκεια Επιτυχούς Αναζήτησης (Successful Search)

Load factor(α)	Open addressing, linear probing ½ (1+1/(1-α))	Open addressing double hashing -ln(1-α)/α	Chained hashing 1+α/2
0.5	1.50	1.39	1.25
0.6	1.75	1.53	1.30
0.7	2.17	1.72	1.35
0.8	3.00	2.01	1.40
0.9	5.50	2.56	1.45
1.0	Δεν εφαρμόζεται	Δεν εφαρμόζεται	1.50
2.0	Δεν εφαρμόζεται	Δεν εφαρμόζεται	2.00
3.0	Δεν εφαρμόζεται	Δεν εφαρμόζεται	2.50

Κατανεμημένοι Πίνακες Κατακερματισμού (DHTs)

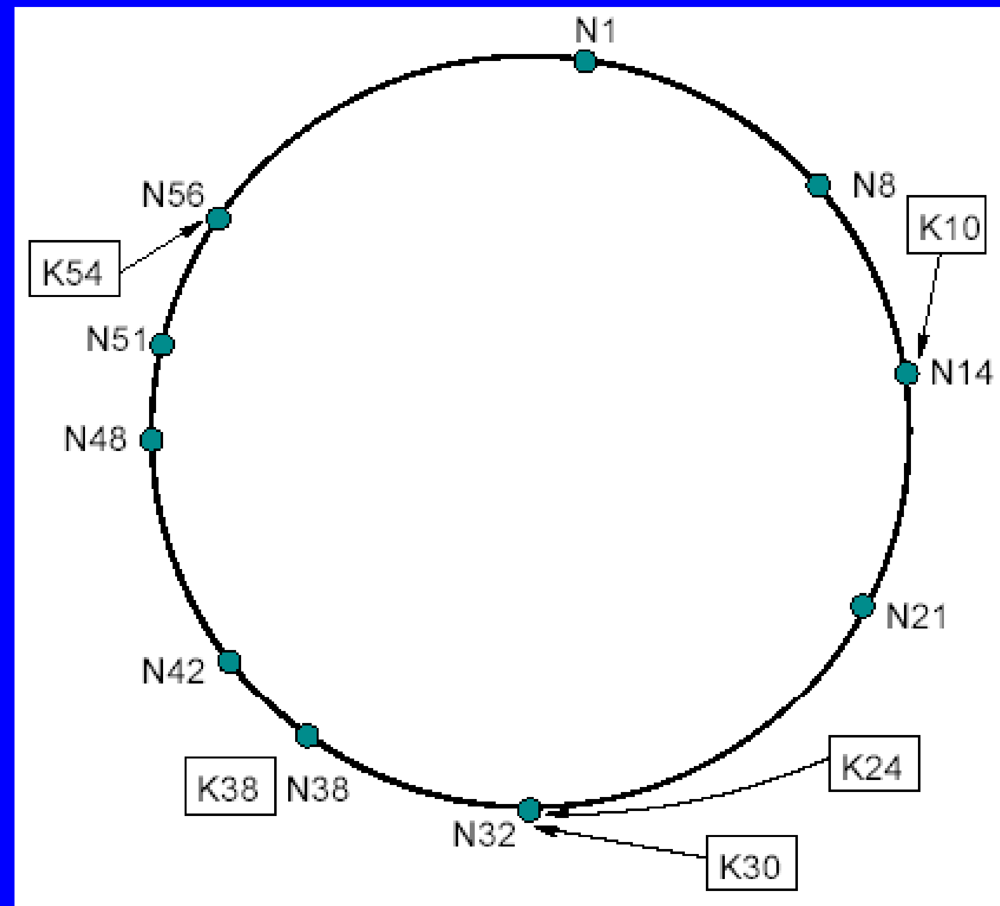
- Διαμέρισε τον πίνακα κατακερματισμού σε ένα σύνολο από servers
  - Γνωστό και ως *key-value store*



- **SHA-1:Secured Hash Standard**
  Παράγει κλειδιά των m-bits, με πολύ μικρή πιθανότητα συγκρούσεων
- **Key_ID = SHA-1(key) mod 2^m**
- **Node_ID = SHA-1(IP address) mod 2^m**
- **Κάθε κλειδί Key_ID αντιστοιχίζεται στον κόμβο με το μικρότερο Node_ID τ.ω.:**
  Node_ID >= Key_ID → Consistent Hashing

m-bit κλειδιά (Key_IDs και Node_IDs) πάνω στον δακτύλιο Chord

- Key_ID = SHA-1(key) mod 2^m
- Node_ID = SHA-1(IP address) mod 2^m
- K10→ N14
- K24, K30→N32
- K38→ N38
- K54→N56



Ανακεφαλαίωση

- Γραμμική Αναζήτηση: O(n) στη μέση περίπτωση
- Δυαδική Αναζήτηση: $O(\log_2 n)$ στη μέση περίπτωση
- Κατακερματισμός
  - Ανοιχτή Διευθυνσιοδότηση
    - Γραμμική Διερεύνηση (Linear probing)
    - Διπλός Κατακερματισμός (Double hashing)
  - Κατακερματισμός με αλυσίδες (Chained hashing)
  - Ο Μέσος αριθμός διερευνήσεων στον πίνακα κατά τη διάρκεια Επιτυχούς Αναζήτησης (Successful Search) είναι συνάρτηση του παράγοντα φόρτου α.