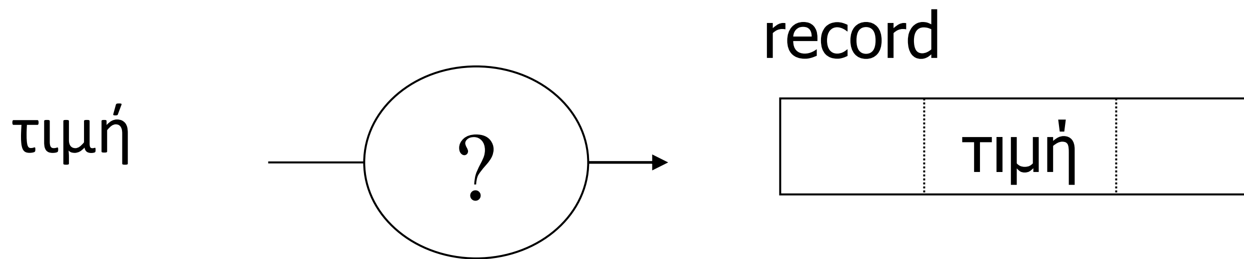
Οργάνωση στο Δίσκο και Ευρετήρια

Γιάννης Κωτίδης

Γενική εικόνα

Δενδρικά ευρετήρια & ευρετήρια κατακερματισμού

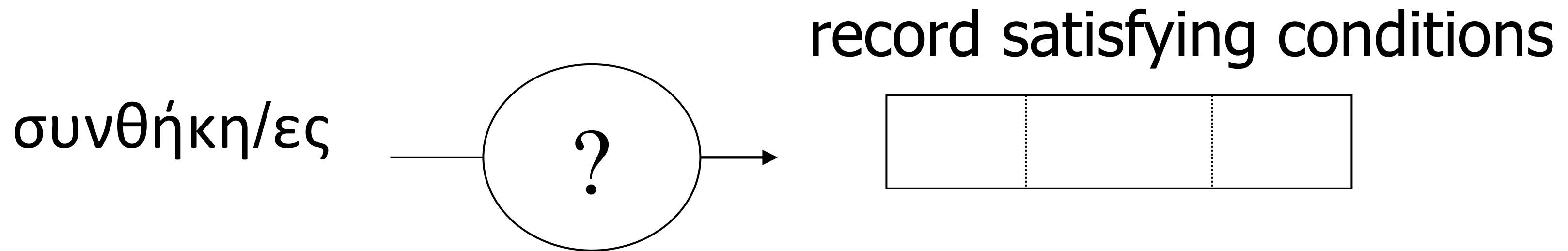


- SELECT * FROM R WHERE a=11

- SELECT student_name FROM STUDENTS WHERE grade =10

Γενική εικόνα

Ποιο σύνθετα ερωτήματα



- SELECT student_name from STUDENTS WHERE grade >=5
- SELECT * FROM R WHERE a≥0 and b<42
- SELECT director from Movies where MATCH(title) AGAINST (“Star Wars”)

Παράδειγμα

- Έστω η σχέση
  - Employee (ID, firstName, lastName, Dept, Salary)
  - 10 M tuples (εγγραφές)
- Επερώτηση:

```
SELECT lastName, Dept, Salary
FROM Employee
WHERE firstName = "Bob"
```

Εκτέλεση χωρίς ευρετήριο (TableScan)

- Ας υποθέσουμε ότι η σχέση είναι αποθηκευμένη στο δίσκο σε συνεχόμενες σελίδες
- **Φυσικό Πλάνο** εκτέλεσης:
  - TableScan(Employee)
    - …ανάγνωση όλης της σχέσης, σελίδα-σελίδα
  - Output tuples* with firstName = "Bob"
    - …για κάθε εγγραφή που διαβάζεις, έλεγξε την τιμή του γνωρίσματος firstName, επέστρεψε όσες εγγραφές ικανοποιούν τη συνθήκη

* tulpe = πλειάδα

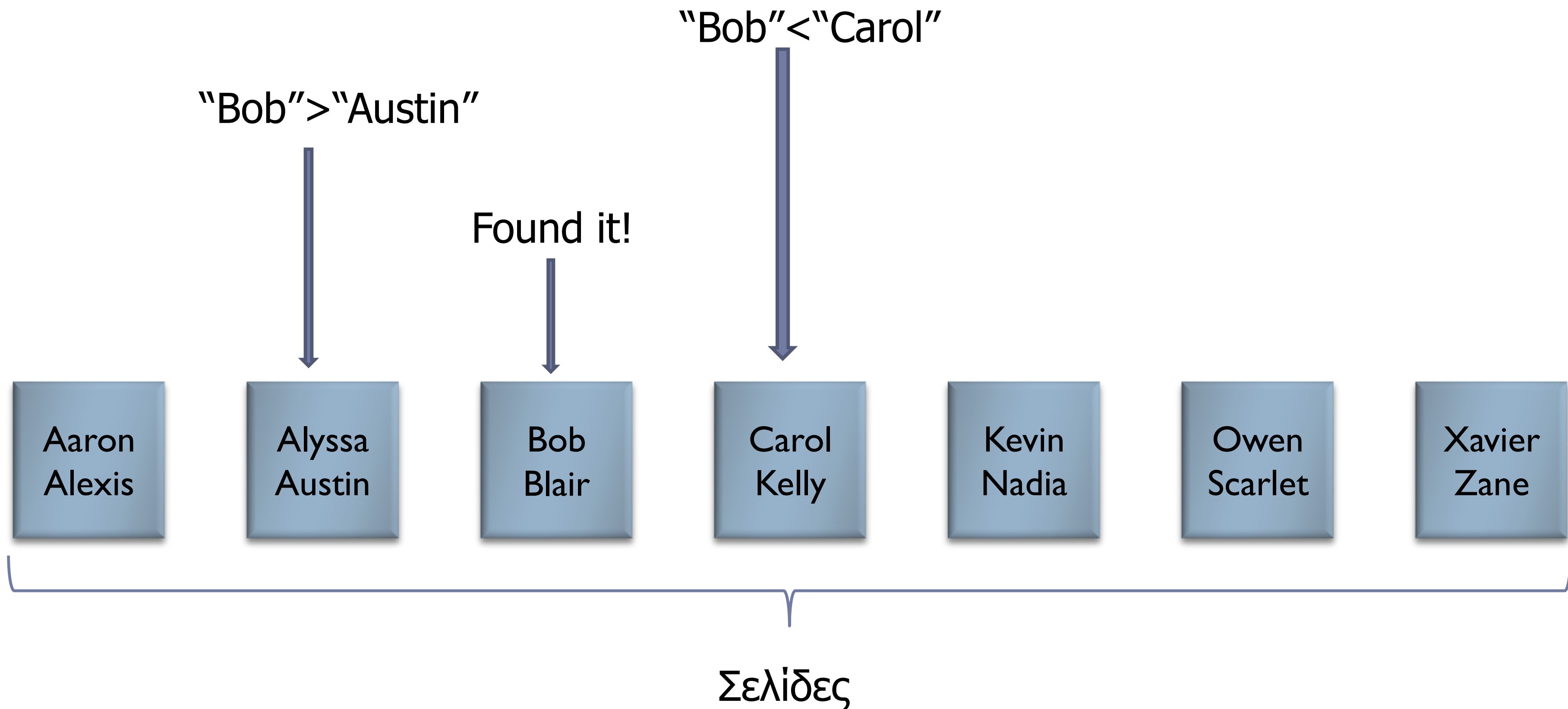
Εκτέλεση χωρίς ευρετήριο

- Χρόνος εκτέλεσης (best case):
  - Υποθέτω οι σελίδες της σχέσεις είναι αποθηκευμένες σειριακά στο δίσκο και χρησιμοποιώ μόνο την ταχύτητα σειριακής εκτέλεσης στους υπολογισμούς
    - αγνοώ μικρο-καθυστερήσεις, όπως όταν αλλάζουμε κύλινδρο
    - στους παρακάτω υπολογισμούς κανονικά θα έπρεπε να εκφράζω το μέγεθος της σχέσης σε σελίδες (βλ. επόμενη διαφάνεια)
  - Μέγεθος εγγραφής = 100 bytes
  - Μέγεθος σχέσης = 10 Million x 100 =1000000000 bytes
  - Ταχύτητα ανάγνωσης δίσκου (Sequential Read) = 40 MB/s
  - Time @ 40 MB/s = 1000000000/40/1024/1024 = 23.8 secs

Εναλλακτικό Πλάνο Εκτέλεσης

- Υποθέτω ότι οι εγγραφές της σχέσης είναι *ταξινομημένες* με βάση το γνώρισμα firstName
- Ιδέα: ας κάνουμε **δυαδική αναζήτηση**

Παράδειγμα Δυαδικής Αναζήτησης



Δυαδική αναζήτηση

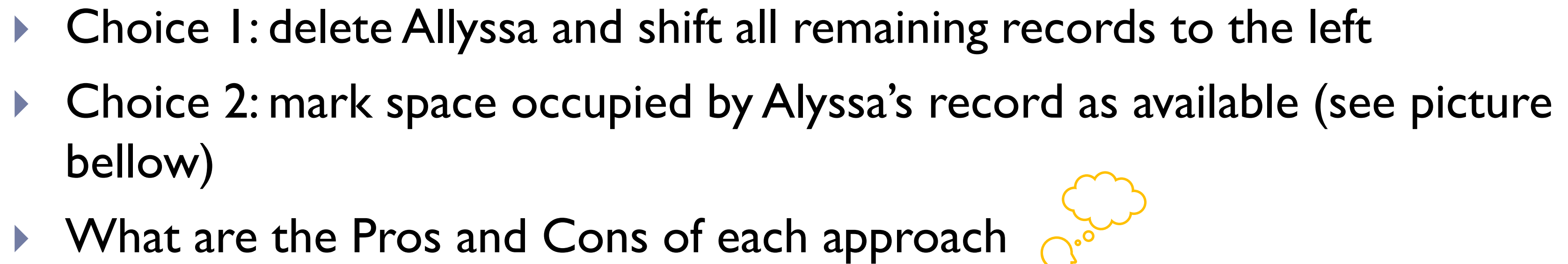
- Μέγεθος σελίδας: 1024 bytes
- Εγγραφές ανά σελίδα: 1024 /100 =10
  - Υποθέτω ότι κάθε εγγραφή πρέπει να αποθηκευτεί ολόκληρη στην ίδια σελίδα. Αυτή η υλοποίηση είναι απλούστερη αλλά αφήνει λίγο αχρησιμοποίητο χώρο σε κάθε σελίδα.
- Αριθμός σελίδων: 10 Million / 10 = 1 Million σελίδες
- Σελίδες που θα διαβάσω: $\lceil \log_2(1M) \rceil = 20$ (στρογγυλοποίηση προς τα πάνω)
- Έστω χρόνος για ένα random I/O: 20ms
- Χρόνος εκτέλεσης: 20 ms x 20 = 400 ms = 0,4 secs

Δουλεύει πάντα?

- Πώς κρατάω τις εγγραφές της σχέσης ταξινομημένες όταν έχω εισαγωγές (inserts), διαγραφές (deletes) και ενημερώσεις (updates);

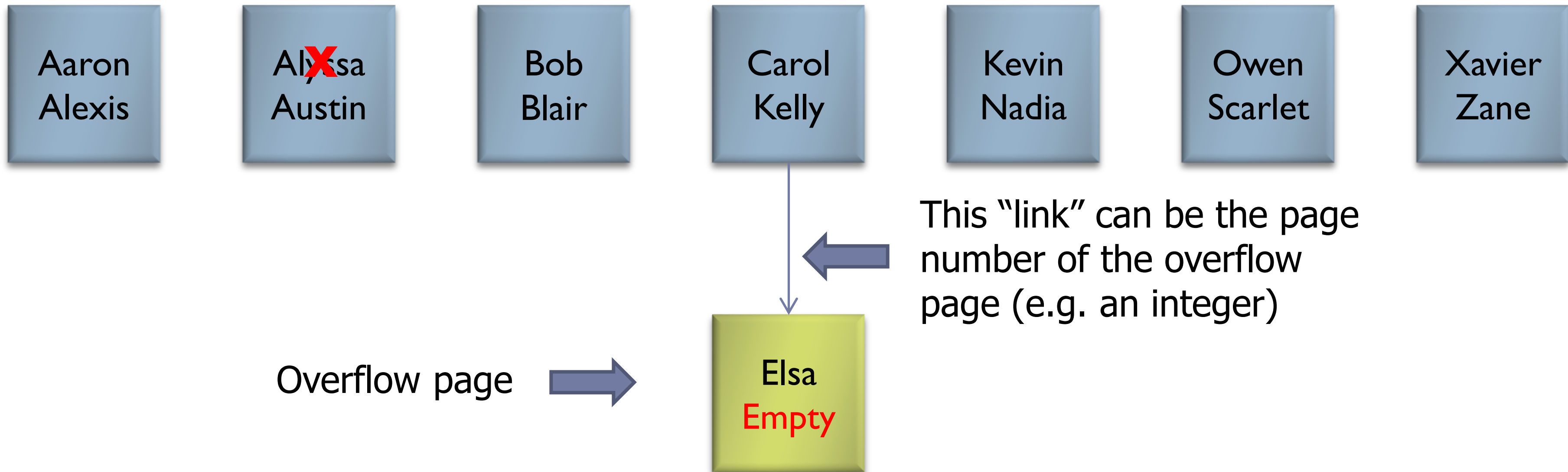
Deletion (Allysa)

- Choice 1: delete Allyssa and shift all remaining records to the left
- Choice 2: mark space occupied by Alyssa's record as available (see picture bellow)
- What are the Pros and Cons of each approach



Insertion (Elsa)

- Choice 1: shift records to the right in order to create space
- Choice 2: create an **overflow** page linked from the page that would normally store Elsa if there were enough space
- What are the Pros and Cons of each approach?



Recap

- Keeping records sorted on a particular attribute has some advantages at query time
  - consider queries that require a specific ordering (list all employees ordered by their firstName)
- But requires extra work when the relation is modified (insert/delete/update)
- We will discuss lateron that physical ordering based on some attribute(s) values can be achieved using a clustering index

Προσοχή

- Ακόμα και αν διατηρήσουμε τη σχέση ταξινομημένη ως προς πχ το όνομα, αυτό δεν θα μας είναι χρήσιμο σε πολλά άλλα ερωτήματα
- Πχ συνθήκες σε διαφορετικά γνωρίσματα από το πεδίο διάταξης:

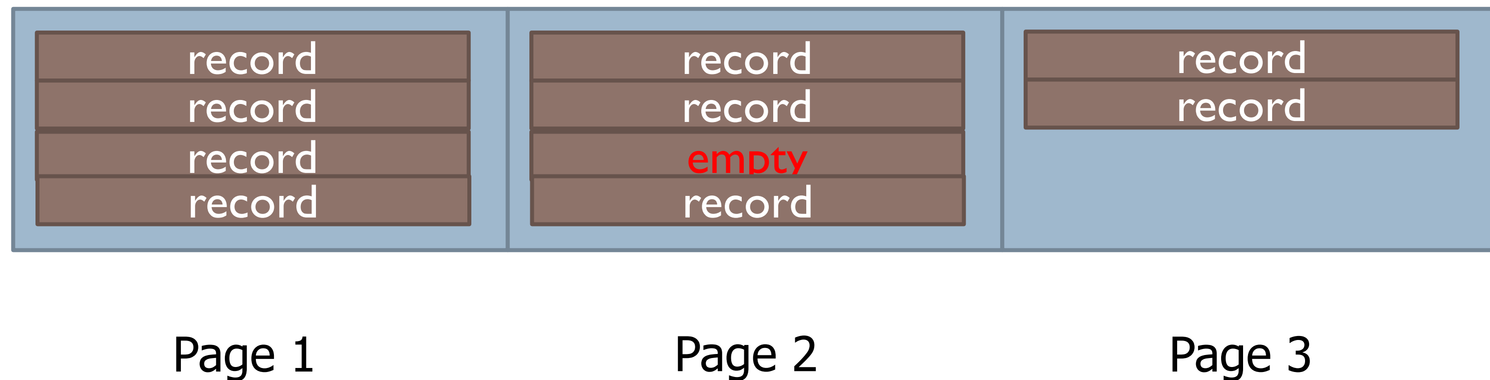
```
SELECT *
FROM Employee
WHERE Dept = “Sales”
```

ή

```
SELECT *
FROM Employee
WHERE Salary > 10000
```

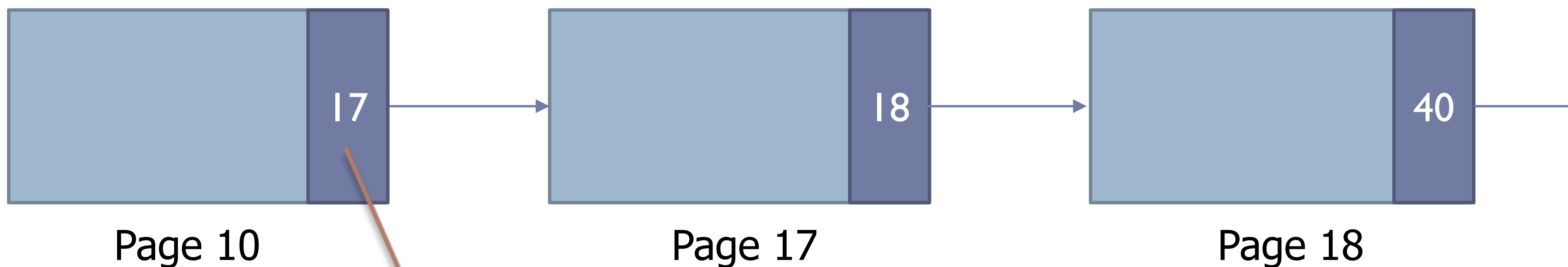
Εναλλακτικοί τρόποι οργάνωσης εγγραφών σε ένα DBMS

- Heap (random order) files
  - Οι εγγραφές της σχέσης αποθηκεύονται μέσα σε σελίδες με ευθύνη του DBMS, χωρίς κάποια προκαθορισμένη σειρά.
  - Απλούστερη υλοποίηση. Μικρότερες ανάγκες για αναδιάταξη σε εισαγωγές, διαγραφές, τροποποιήσεις.
- Παράδειγμα (1): συνεχόμενες σελίδες, εγγραφές ίδιου μεγέθους



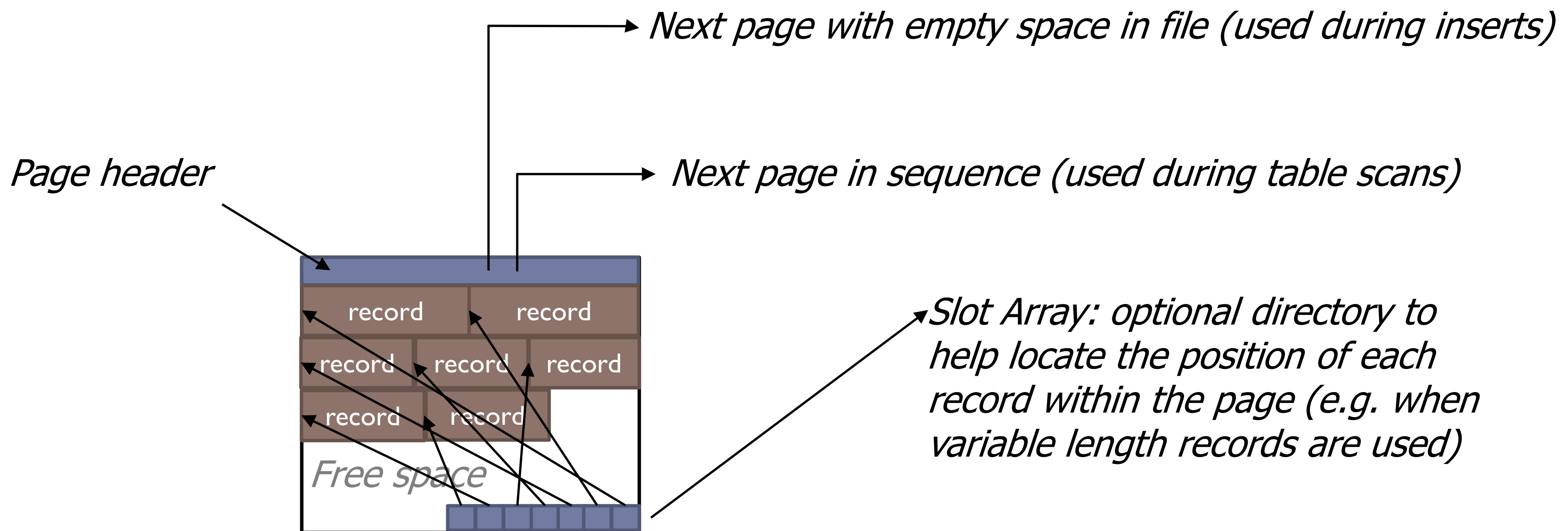
Σημείωση

- Οι σελίδες μιας σχέσης μπορεί να είναι πραγματικά συνεχόμενες στο δίσκο
  - επιθυμητό αλλά δύσκολο να επιτευχθεί 100% για δυναμικά δεδομένα (fragmentation)
  - ...ή συνδεδεμένες (ως λίστα) μέσω αναφορών (pointers)
    - Αναφορά/pointer: αριθμός επόμενης σελίδας



Παράδειγμα 2: heap file υλοποιημένο ως συνδεδεμένη λίστα σελίδων για εγγραφές μεταβλητού μεγέθους

- Το Page header μπορεί να περιέχει
  - Αναφορά στη σχέση τις εγγραφές της οποίας περιέχει η σελίδα αυτή
  - Αριθμός εγγραφών στη σελίδα
  - Τον αριθμό της επόμενης σελίδας (εφόσον υπάρχει)
  - Αναφορά στην επόμενη σελίδα με κενό χώρο



Εντοπισμός μιας εγγραφής

- Ανάλογα με την υλοποίηση μπορούμε να εντοπίσουμε σε μία εγγραφή στο δίσκο χρησιμοποιώντας
  - Τον αριθμό σελίδας που την περιέχει (ανατρέχοντας στη συνέχεια μέσα στη σελίδα)
  - Τον αριθμό σελίδας και τη θέση της (offset) μέσα στη σελίδα (βλ. προηγούμενο παράδειγμα με το slot array)
  - Τον αριθμό εγγραφής, αν η θέση της στο δίσκο μπορεί να προκύψει έμμεσα από αυτόν
    - Πχ αν έχουμε εγγραφές σταθερού μεγέθους (1ο παράδειγμα)
  - …

Εναλλακτικοί τρόποι οργάνωσης εγγραφών σε ένα DBMS

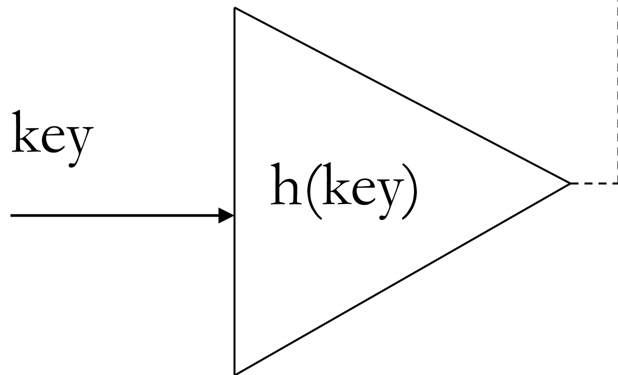
- Heap (random order) files
  - Οι εγγραφές της σχέσης αποθηκεύονται μέσα σε σελίδες με ευθύνη του DBMS, χωρίς κάποια προκαθορισμένη σειρά.
  - Απλούστερη υλοποίηση. Μικρότερες ανάγκες για αναδιάταξη σε εισαγωγές, διαγραφές, τροποποιήσεις.
- Sorted/sequential files
  - Κατάλληλα όταν οι εγγραφές πρέπει να ανακτηθούν σε μια προκαθορισμένη σειρά (πχ ORDER BY) ή όταν έχουμε *αναζητήσεις εύρους*.
  - Ανάγκη για αναδιάταξη όταν έχουμε αλλαγές.
  - Συχνά υλοποιούνται μέσω B-tree ευρετηρίων που θα συζητήσουμε
- Indexes
  - Αρχεία δενδροειδή ή οργανωμένα με βάση τον κατακερματισμό τα οποία επιτρέπουν τη γρήγορη ανάκτηση εγγραφών με κάποιο κριτήριο αναζήτησης.

Ευρετήρια

- Βοηθητικές δομές δεδομένων που μας επιτρέπουν να ανακτούμε γρήγορα εγγραφές μιας σχέσης που ικανοποιούν μία λογική συνθήκη.
- Η λογική συνθήκη (predicate) μπορεί
  - Να αναφέρεται σε ένα ή περισσότερα γνωρίσματα
  - Να περιέχει συνθήκες ισότητας, ανισότητας, συναρτήσεις οριζόμενες από το χρήστη (UDFs)
  - Να αναφέρεται σε περίπλοκες συνθήκες
    - Χωρικές Βάσεις Δεδομένων
      - Βρες όλα τα εστιατόρια σε ακτίνα 2km (Range Query)
    - Κοντινότεροι γείτονες (Nearest Neighbor Queries)
      - Βρες τους 10 χρήστες που ακούν παρόμοια μουσική με τον John
    - Αναζήτηση μέσω keywords (Information Retrieval)
      - Βρες όλα τα άρθρα που περιέχουν τις φράσεις “Data Science”, “Big Data” ή άλλες συναφείς με αυτές

Main Memory example: hash table

Example: organize all customer orders in a searchable static hash table using attribute orderId as hash key



h(key) = key % 8

buckets

Pointer to record of order with orderId=32

0 → 32 → 48 → (null)
1
2 → 10 → (null)
3 → 27 → 75 → (null)
4
5 → 21 → (null)
6
7 → 55 → (null)

Ευρετήρια & ΣΔΒΔ

- Χρειαζόμαστε δομές κατάλληλες για αποθήκευση στο δίσκο
  - Όταν οι εγγραφές έρχονται στη μνήμη το ΣΔΒΔ μπορεί να τις οργανώνει αποδοτικά στη μνήμη για ταχύτερη προσπέλαση
    - Αυτό το κεφάλαιο όμως δε θα μας απασχολήσει ιδιαίτερα στο μάθημα!

- Ανάγκη για ενημέρωση των ευρετηρίων όταν αλλάζει το περιεχόμενο της σχέσης (updates)
  - Πιθανώς αυξημένο κόστος αν έχουμε συχνά ενημερώσεις στο index key
  - Όμως κερδίζω σε ταχύτητα επερωτήσεων
  - Ιδέες:
    - καθυστερημένα (defer,lazy) updates
    - μαζικά (bulk) updates

Πότε φτιάχνω ευρετήρια;

- C_u : χρόνος ενημέρωσης ευρετηρίου
- F_u : συχνότητα ενημέρωσης
- C_{qi} : χρόνος επερώτησης με ευρετήριο
- C_q : χρόνος επερώτησης χωρίς ευρετήριο
  - θεωρούμε την περίπτωση όπου το ευρετήριο βοηθά: $C_q > C_{qi}$
- F_q : συχνότητα επερώτησης
- Θα πρέπει $F_q * (C_q - C_{qi}) > F_u * C_u$

- …δηλαδή όταν το κέρδος λόγω της επιτάχυνσης των ερωτημάτων υπερκαλύπτει το κόστος ενημέρωσης

Ορολογία

- Ένα ευρετήριο (index) είναι μια βοηθητική δομή αρχείου που επιτρέπει την αποδοτική αναζήτηση μιας εγγραφής σε μία σχέση
- Το ευρετήριο καθορίζεται σε ένα ή περισσότερα γνώρισμα της σχέσης που καλείται πεδίο/α ευρετηριοποίησης (indexing field/indexing attribute/index key)

Είδη Ευρετηρίου

- **Πρωτεύον ευρετήριο** (*primary index*) όταν εφαρμόζεται στο κλειδί της σχέσης
  - Συνήθως δημιουργείται αυτόματα
    - Γιατί πιστεύετε ότι είναι απαραίτητο?
- **Δευτερεύον ευρετήριο** (*secondary index*) όταν ορίζεται σε πεδίο διαφορετικό από το κλειδί
  - Πχ Employee.firstName
- **Ευρετήριο συστάδων** (*clustering/clustered index*) όταν η τιμή του γνωρίσματος ευρετηριοποίησης καθορίζει τη θέση της εγγραφής
  - Συνήθως ένα primary index είναι και clustered index

Indexes with Included Columns

- Σε κάποια συστήματα μπορούμε να προσθέσουμε στο ευρετήριο πεδία που δεν είναι μέρος του πεδίου αναζήτησης ως Included Columns

candidate included column

Select Employee.firstName

From Employees

Where Employee.lastName = 'Jordan'

candidate index key

- Η τιμή του γνωρίσματος firstName αποθηκεύεται στο ευρετήριο μαζί με το lastName
  - Δε χρειάζεται πλέον να ανατρέξουμε στη σχέση (πχ heap file) για να απαντήσουμε το ερώτημα!
  - Όμως το μέγεθος του ευρετηρίου αυξάνει...

Ποια πληροφορία μας δίνει ένα ευρετήριο?

- Μας λέει σε ποιο σημείο (στο δίσκο) είναι αποθηκευμένες οι εγγραφές που ψάχνουμε
- Για τη συζήτηση μας είναι ποιο εύκολο να σκεφτόμαστε ότι το ευρετήριο μας επιστρέψει δείκτες (pointers) προς τις εγγραφές αυτές
- Ένας τέτοιος pointer (ανάλογα την υλοποίηση όπως έχουμε ήδη συζητήσει) μπορεί να είναι:
  - ο αριθμός της σελίδας που περιέχει την εγγραφή, σελίδα+offset
- Επίσης μπορεί να είναι η τιμή του κλειδιού της εγγραφής
  - Σε αυτή την περίπτωση η εγγραφή μπορεί να ανακτηθεί κάνοντας ένα clustered index seek/key lookup στο πρωτεύων ευρετήριο
    - σε clustered ευρετήρια οι σελίδες με τις εγγραφές της σχέσης συχνά είναι ενσωματωμένες στη δομή του ευρετηρίου (πχ ως φύλλα σε ένα B-tree)

Θέματα βελτιστοποίησης

- Ας θεωρήσουμε την επερώτηση:

```
SELECT *
FROM Employee
WHERE firstName = "Bob" AND Salary>700
```

- Η σχέση είναι οργανωμένη ως heap file
- Έχω ευρετήριο στο firstName και ένα δεύτερο στο Salary
  - Ποιο από τα δύο να χρησιμοποιήσω;
  - Μπορώ και τα δύο;
  - Υπάρχει καλύτερη λύση;

Ιδέες (1,2,3)

- Χρησιμοποίησε το ευρετήριο στο firstName. Για κάθε εγγραφή που θα τραβήξεις με αυτό το ευρετήριο, έλεγξε αν Salary>700

- Χρησιμοποίησε το ευρετήριο στο Salary ψάχνοντας για Salary>700. Για κάθε εγγραφή που θα τραβήξεις με αυτό το ευρετήριο, έλεγξε αν firstName="Bob"

- Πάρε τις λίστες των "pointers" που γυρίζει κάθε ευρετήριο και βρες την τομή τους. Επέστρεψε όλα τα records στη τομή των λιστών. Δε χρειάζεται κάποιος επιπλέον έλεγχος (γιατί?)

Ιδέες (4,5)

- Φτιάξε ένα νέο ευρετήριο στο *σύνθετο γνώρισμα* (firstName,Salary)
- Φτιάξε ένα νέο ευρετήριο στο (Salary, firstName)
    - Ποια η διαφορά από το παραπάνω;

Ευρετήριο ενός γνωρίσματος (Single-Attribute Index)

- Σχέση ταξινομημένη στο κλειδί του ευρετηρίου (index-key)
- Ας φτιάξουμε ένα ευρετήριο όπως σε ένα βιβλίο...

Sequential File

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

Τιμή κλειδιού

Υπόλοιπα γνωρίσματα εγγραφής

Σελίδα

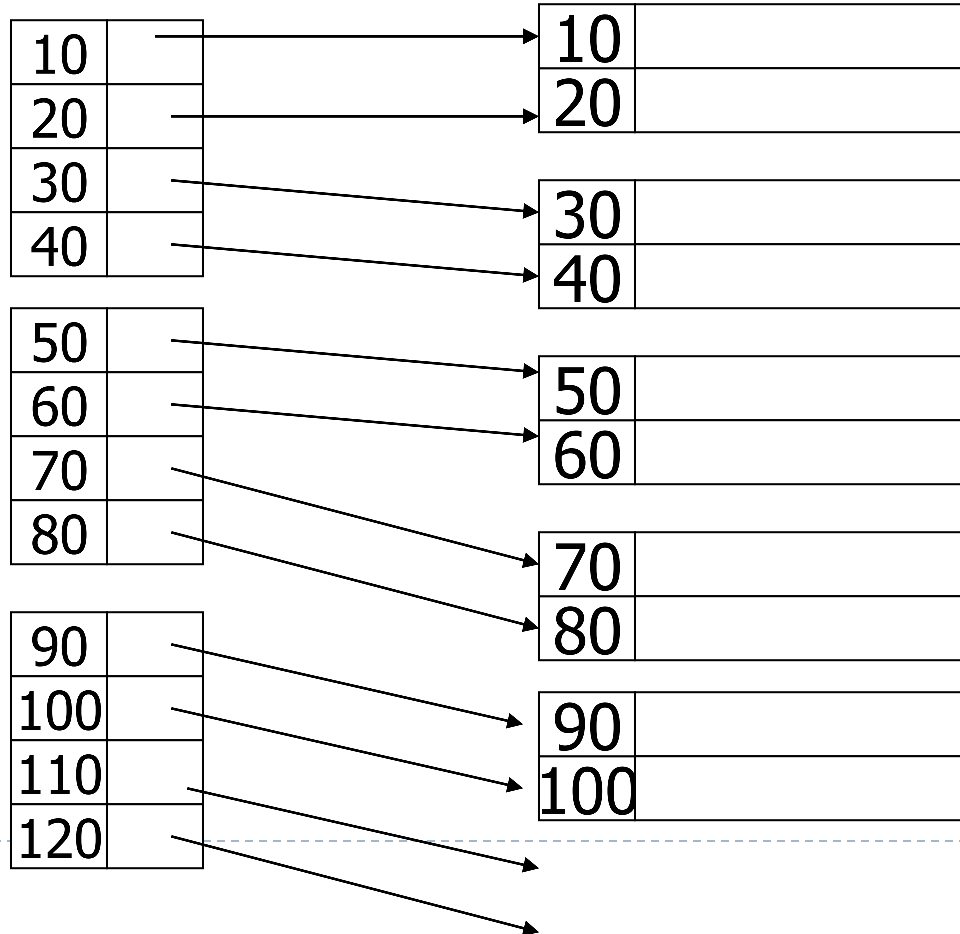
Αυτό μας κάνει?

Χρήσιμο εφόσον το ευρετήριο είναι σημαντικά μικρότερο από τη σχέση

Πυκνό ευρετήριο (Dense Index):

Στο ευρετήριο εμφανίζονται όλες οι τιμές του γνωρίσματος που υπάρχουν στη σχέση

Πως θα το χρησιμοποιήσω σε μία επερώτηση?



Παράδειγμα 1

- Έστω σχέση Empoyee(id, fname, lname, work-phone, home-phone, **ssn**, dept, position) με 100 χιλιάδες εγγραφές
- Κάθε εγγραφή καταλαμβάνει 80 bytes και αποθηκεύεται σε σελίδες μεγέθους 8KB.
  - 8192/80 = 102,4. Υποθέτω για το παράδειγμα ότι μια εγγραφή δε διασπάται κατά την αποθήκευση της σε παραπάνω από μία σελίδες
    - Επομένως κάθε σελίδα «χωράει» 102 εγγραφές αφήνοντας λίγο κενό χώρο
  - Η σχέση καταλαμβάνει $\lceil 100000/102 \rceil$ = 981 σελίδες στο δίσκο
- Έστω ευρετήριο πάνω στο γνώρισμα ssn
  - Κάθε εγγραφή του ευρετηρίου καταλαμβάνει 12 bytes
    - Κάθε σελίδα του ευρετηρίου χωράει 8192/12 δηλαδή 682 εγγραφές
  - Το ευρετήριο καταλαμβάνει $\lceil 100000/682 \rceil$ = 147 σελίδες στο δίσκο

Παράδειγμα 2

- Διατεταγμένη σχέση R με 30000 εγγραφές
- Μέγεθος σελίδας1KB, μέγεθος εγγραφής 120bytes
- Δίσκος: 20msec για random I/O, 30MB/sec για sequential read

- Κόστος αναζήτησης μίας εγγραφής με συγκεκριμένη τιμή στο πεδίο διάταξης (πρωτεύον κλειδί);

Δυαδική αναζήτηση χωρίς ευρετήριο

- Κάθε σελίδα περιέχει 1024/120 δηλαδή 8 εγγραφές
- Η σχέση καταλαμβάνει $B(R)=\lceil 30000/8 \rceil=3750$ σελίδες
- Η δυαδική αναζήτηση θα χρειαστεί $\lceil \log_2(3750) \rceil = 12$ I/O
- Έστω 20msec για κάθε random I/O
  - Συνολικά 0,020*12 = 0,24secs

Αναζήτηση με ευρετήριο

- Έστω ότι κάθε εγγραφή στο ευρετήριο καταλαμβάνει 4 bytes για την τιμή και 4 bytes για τον pointer
- Κάθε σελίδα του ευρετηρίου περιέχει $\lceil 1024/8 \rceil$=128 εγγραφές
- Το ευρετήριο καταλαμβάνει $\lceil 30000/128 \rceil$=235 σελίδες
- Η δυαδική αναζήτηση στο ευρετήριο θα χρειαστεί $\lceil \log_2(235) \rceil$ = 8 I/O
- Χρειάζομαι και 1 I/O όταν ακολουθώ τον pointer για να ανακτήσω την εγγραφή από το αρχείο της σχέσης, άρα συνολικά 9 I/O
  - Χρόνος = 9*0,02=0,18 secs

Ερώτηση: αν δεν κάνω δυαδική αναζήτηση?

- Ας υπολογίσουμε τον μέσο χρόνο…
- Χωρίς ευρετήριο
  - ½*3750 σελίδες * 1KB/σελίδα = (περίπου) 1.8 MB
  - 1.8 MB σε 0,06sec @ 30MB/sec
  - Προσθέτω 1 seek για την πρώτη σελίδα της σχέσης (+0,02secs)
  - Σύνολο 0,08secs

Ευρετήριο (χωρίς δυαδική αναζήτηση)

- Διαβάζω (κατά μέσο όρο) το μισό ευρετήριο
  - ½ * 235 σελίδες = 117.5
  - Άρα θα διαβάσω σελίδες = 118 σελίδες = 118KB
  - 118KB σε 0,004secs @ 30MB/sec
  - 1 seek για την αρχή του ευρετηρίου και 1 seek για να βρω τη σελίδα της σχέσης που περιέχει την εγγραφή
  - Σύνολο = 0,044secs

Σύνοψη

	Δυαδική Αναζήτηση	Σειριακή ανάγνωση (μέσος χρόνος)
Με ευρετήριο	0,18	0,044
Χωρίς ευρετήριο	0,24	0,08

Προτεινόμενη άσκηση: Πως αλλάζουν οι συσχετισμοί αν πχ η σχέση ήταν 1000 φορές ποιο μεγάλη?

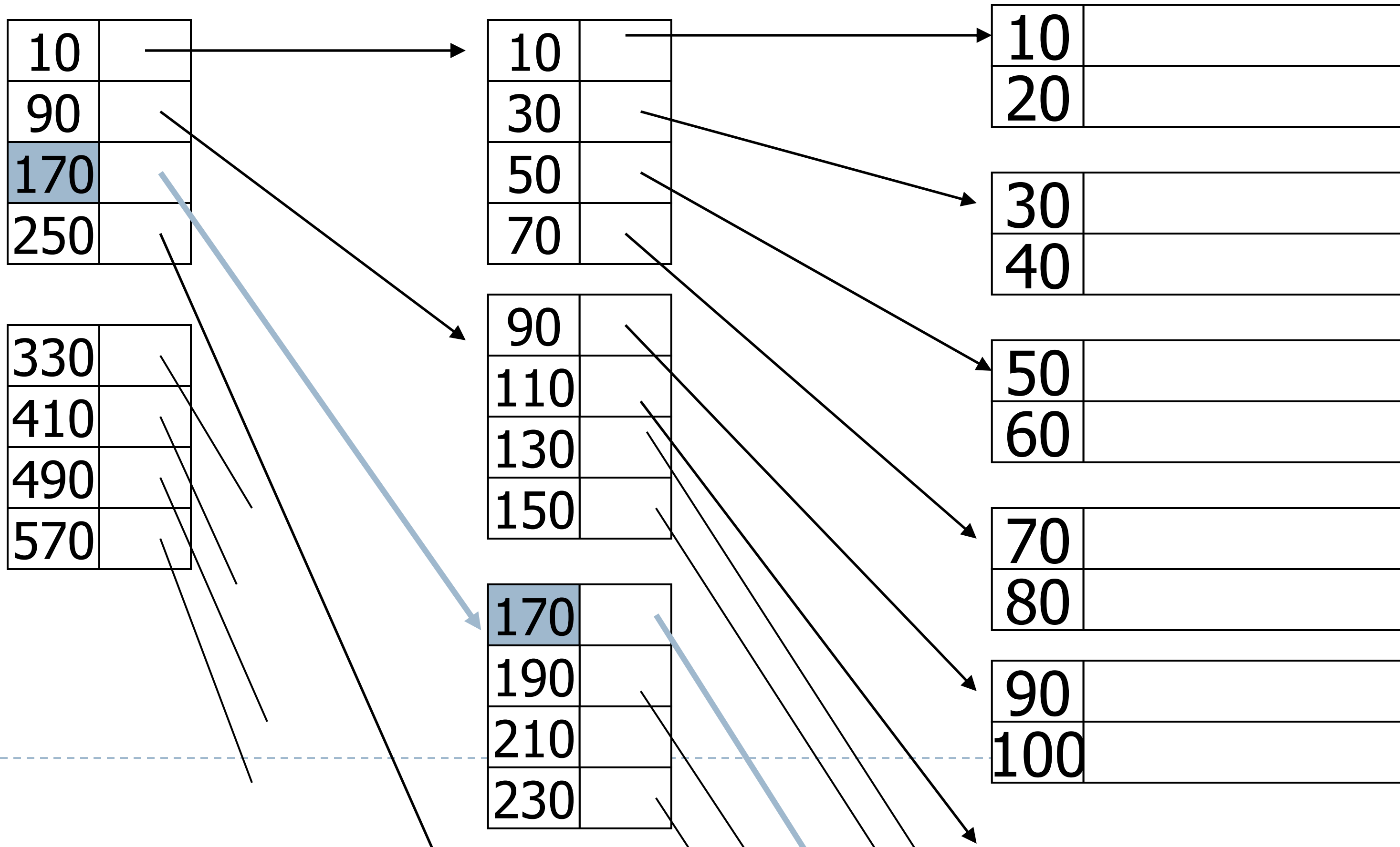
Αραιό ευρετήριο (sparse index)

- Καταγράφω την **πρώτη** τιμή του κλειδιού σε κάθε σελίδα
- Το ευρετήριο είναι σημαντικά μικρότερο
- Τι χάνω;
- Μπορεί να υλοποιηθεί πάντα;

Αραιό ευρετήριο (sparse index)

- Καταγράφω την **πρώτη** τιμή του κλειδιού σε κάθε σελίδα
- Το ευρετήριο είναι σημαντικά μικρότερο
- Αναζήτηση για το 55?

Ας συνεχίσουμε στο ίδιο μοτίβο (multi-level index)



Ερώτηση

- Μπορούμε να φτιάξουμε ένα πυκνό δεύτερο επίπεδο σε ένα πυκνό ευρετήριο πρώτου επιπέδου;

Sparse Index vs. Dense Index

- <u>Sparse:</u> Μικρότερο μέγεθος, μπορώ να κρατήσω μεγαλύτερο τμήμα του στη μνήμη
- <u>Dense</u>: Μπορώ να απαντήσω αν η τιμή του γνωρίσματος που ψάχνω υπάρχει ή όχι στη σχέση χωρίς να ανατρέξω στη σχέση

Επίσης:

- sparse ταχύτερο σε εισαγωγές/ενημερώσεις (γιατί?)
- dense απαραίτητο για secondary indexes

Αραιό ευρετήριο (sparse index)

- Αλλαγή (update) του 60 σε 62?
- Σε αυτό το παράδειγμα η ενημέρωση της εγγραφής δεν αλλάζει κάτι στο ευρετήριο

Sparse Index

10	
30	
50	
70	

90	
110	
130	
150	

170	
190	
210	
230	

Sequential File

10	
20	

30	
40	

50	
60 → 62	

70	
80	

90	
100	

Διπλοεγγραφές: το index/sequence-key δεν είναι πρωτεύων κλειδί

10	
10	

10	
20	

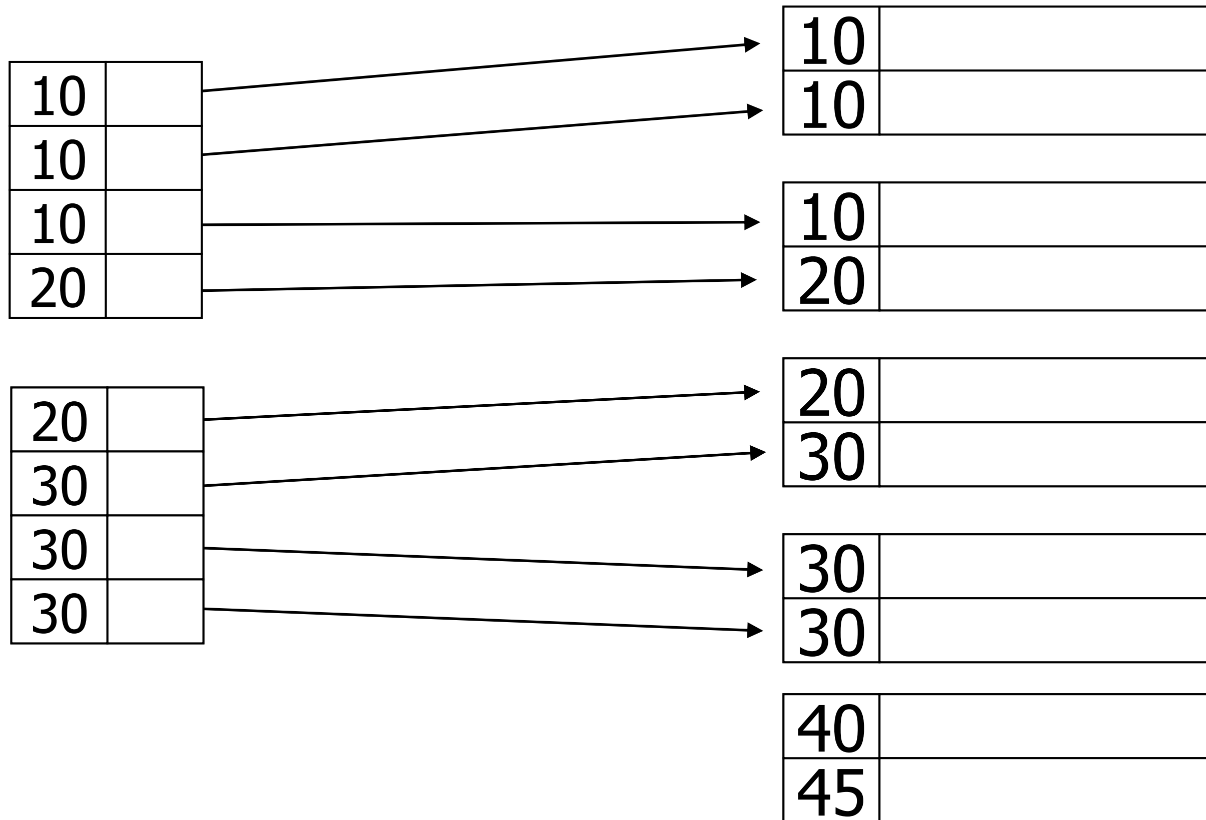
20	
30	

30	
30	

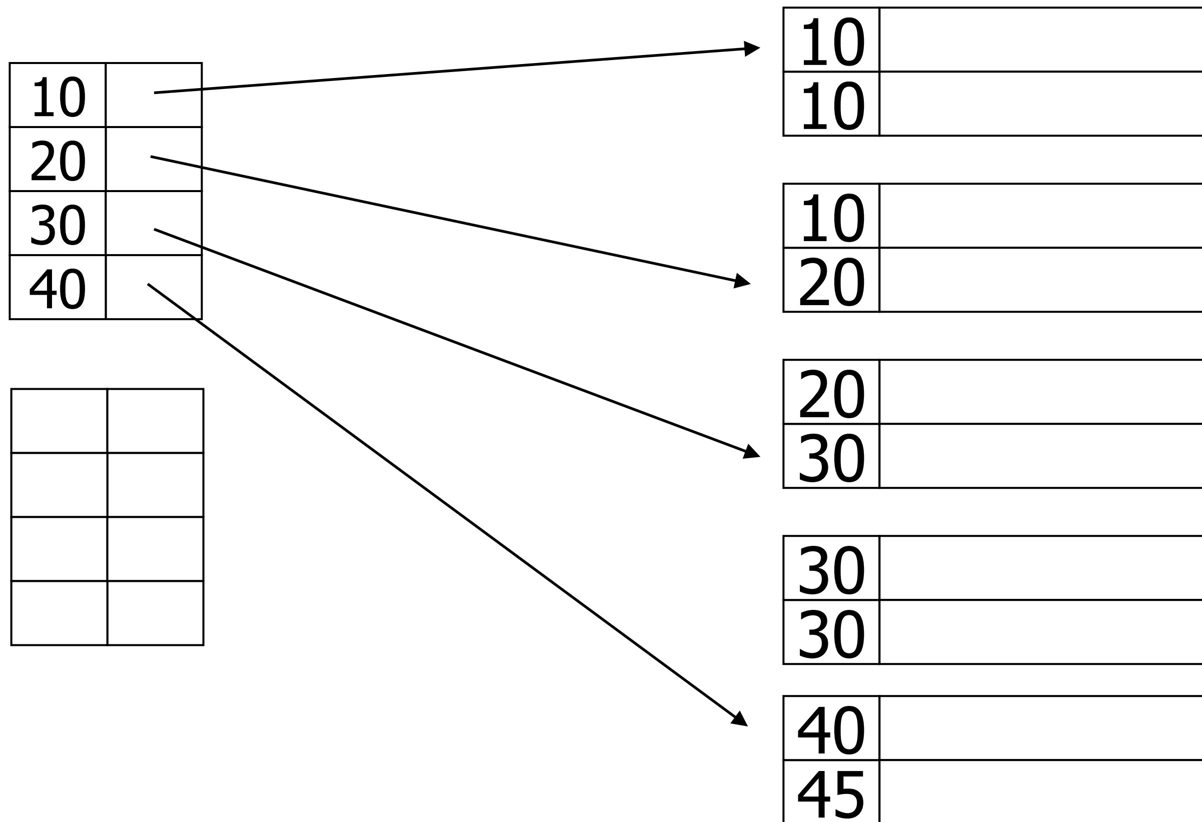
40	
45	

Διπλοεγγραφές

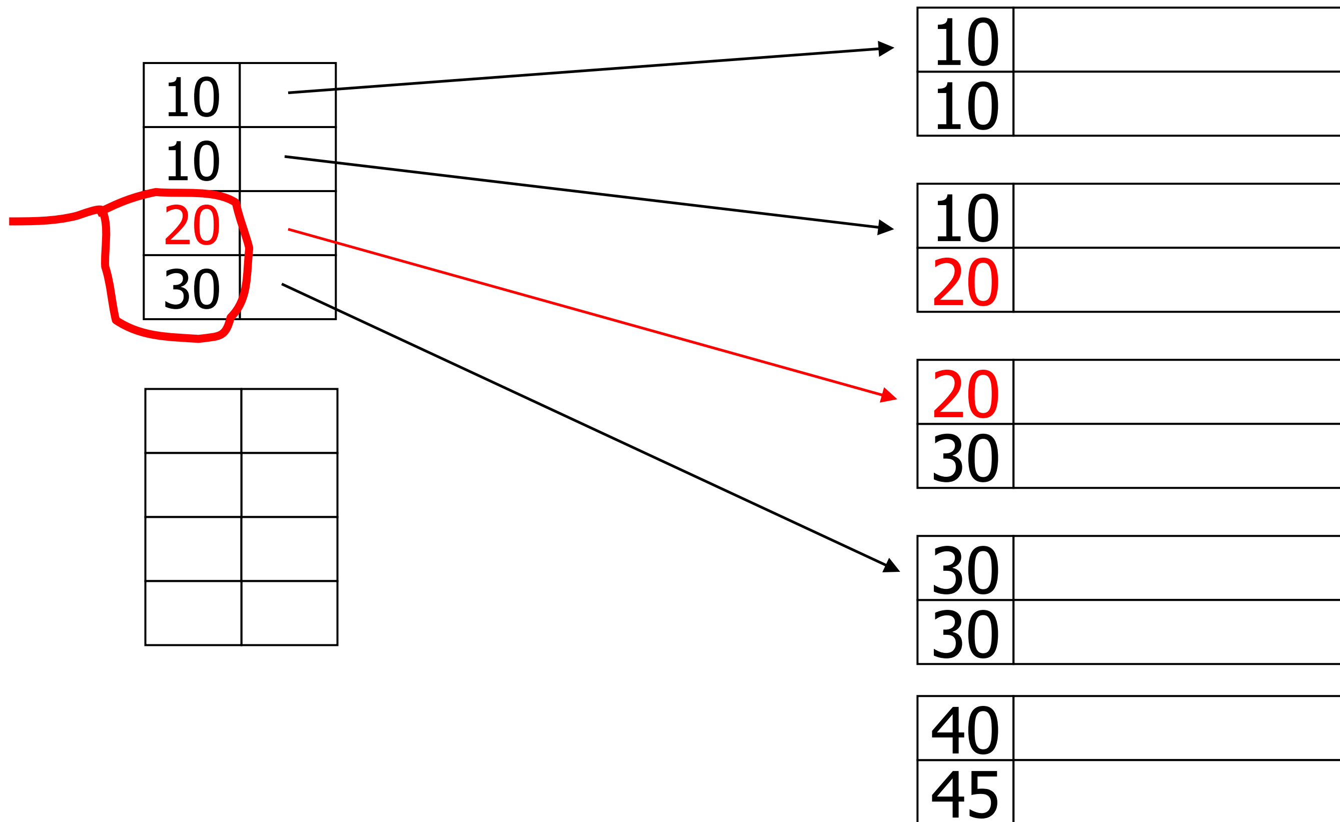
Dense index: μία λύση



Καλύτερη λύση;



Διπλοεγγραφές σε Sparse Index?



Duplicate keys

Sparse index, another way?

– place first new key from block

should this be 40?

10	
20	
30	
30	

10	
10	

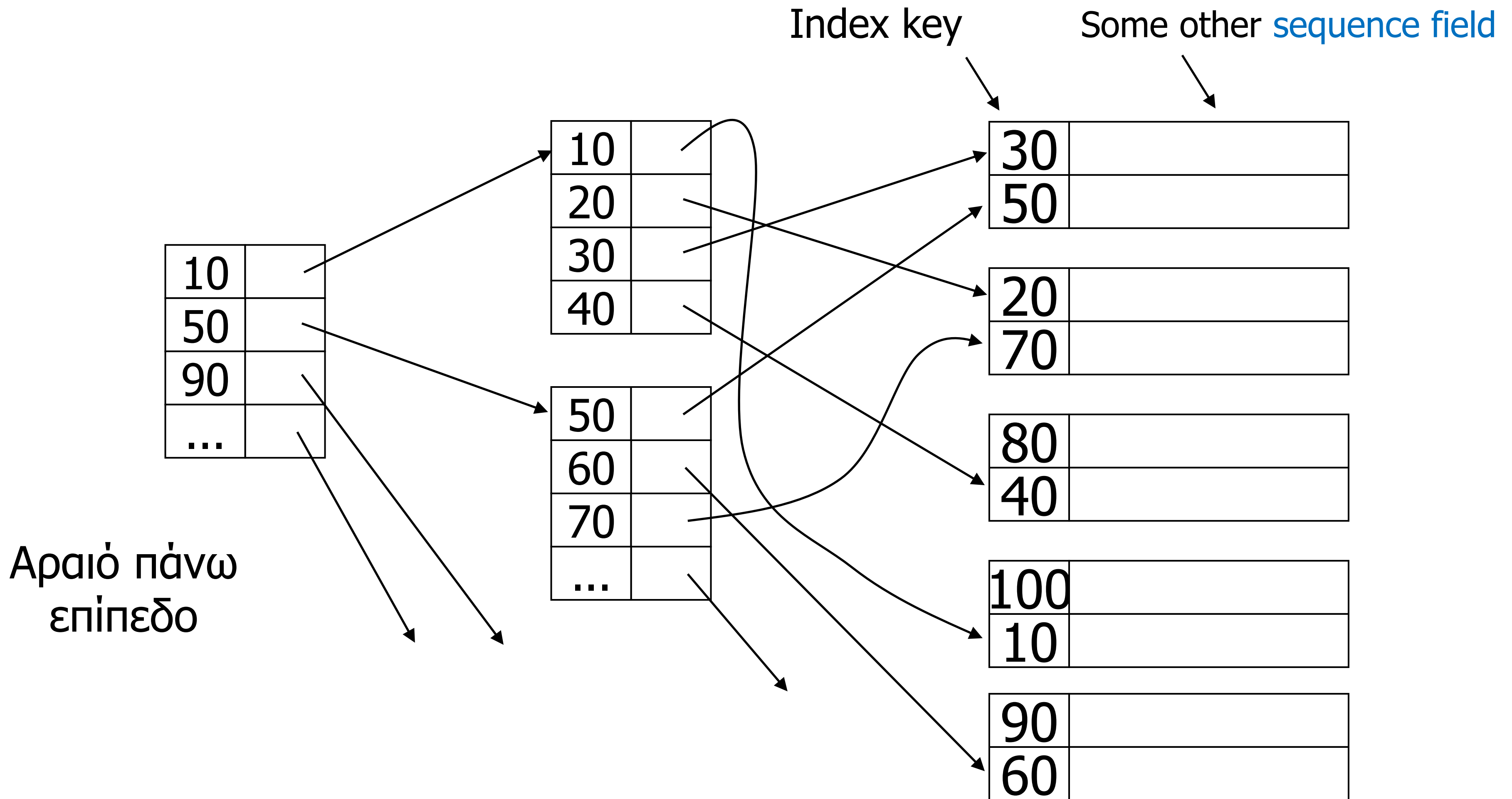
10	
20	

20	
30	

30	
30	

40	
45	

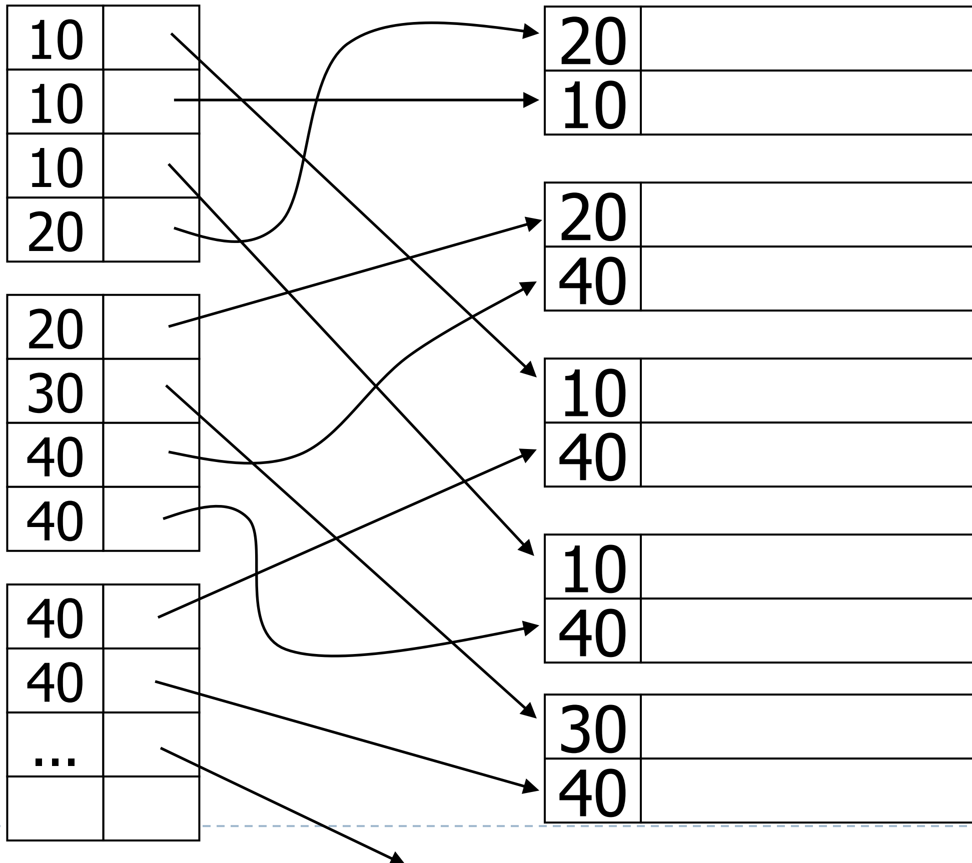
Secondary index
(Κατώτερο επίπεδο πάντα πυκνό)



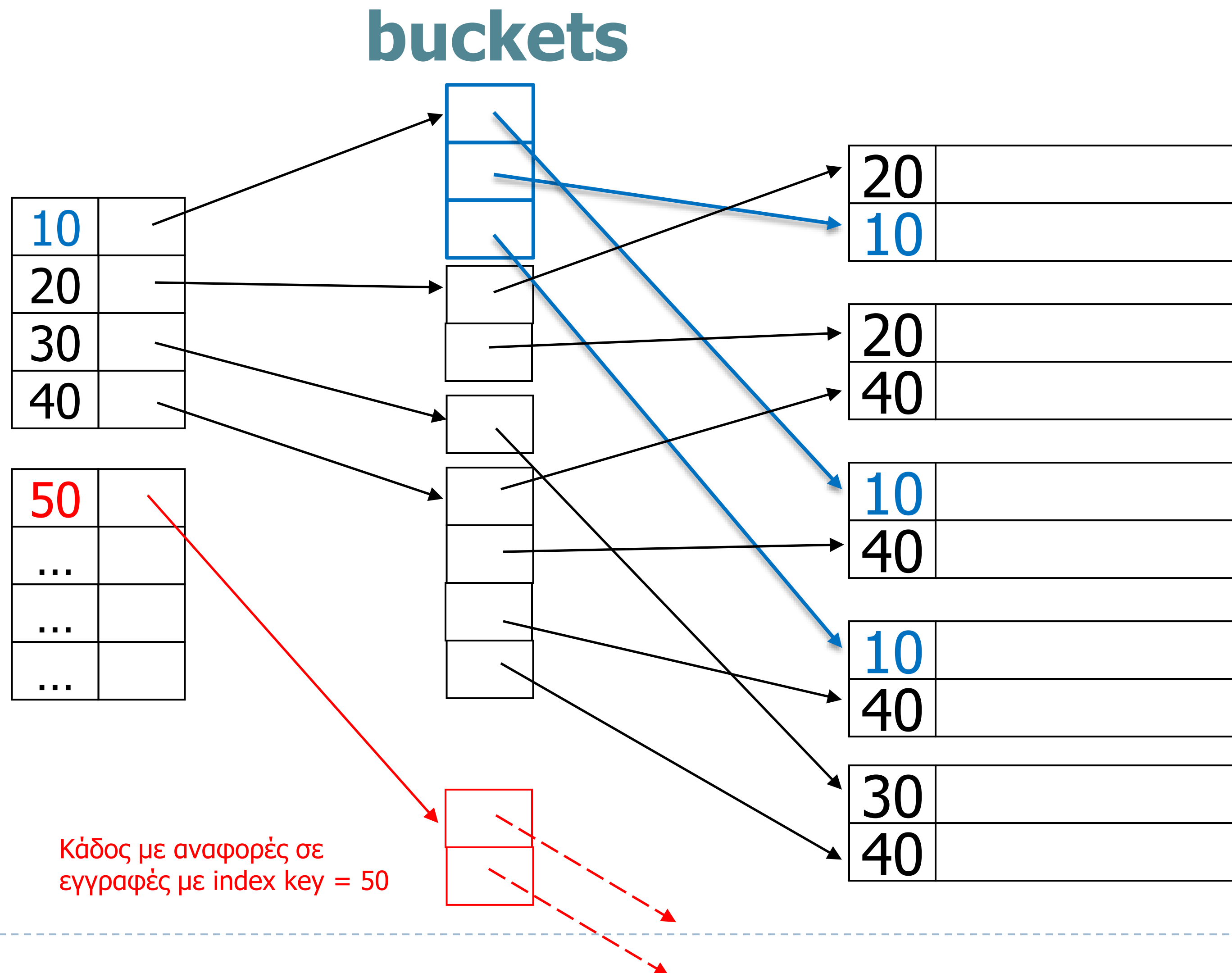
Διπλοεγγραφές & secondary indexes

Μία επιλογή:

Σπατάλη χώρου!



Ιδέα των Buckets (κάδοι)

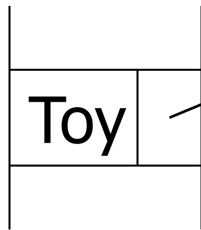


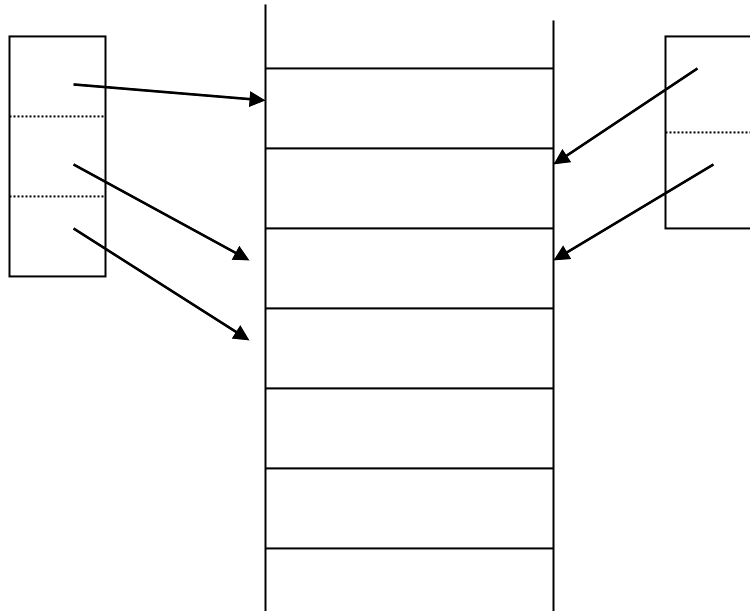
Παράδειγμα

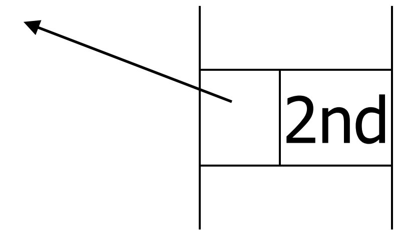
Indexes	Records
empID: primary	EMP (empID,name,dept,floor,...)
Dept: secondary	
Floor: secondary	

Επερώτηση: Βρες υπαλλήλους με

(Toy Dept) $\wedge$ (2nd floor)

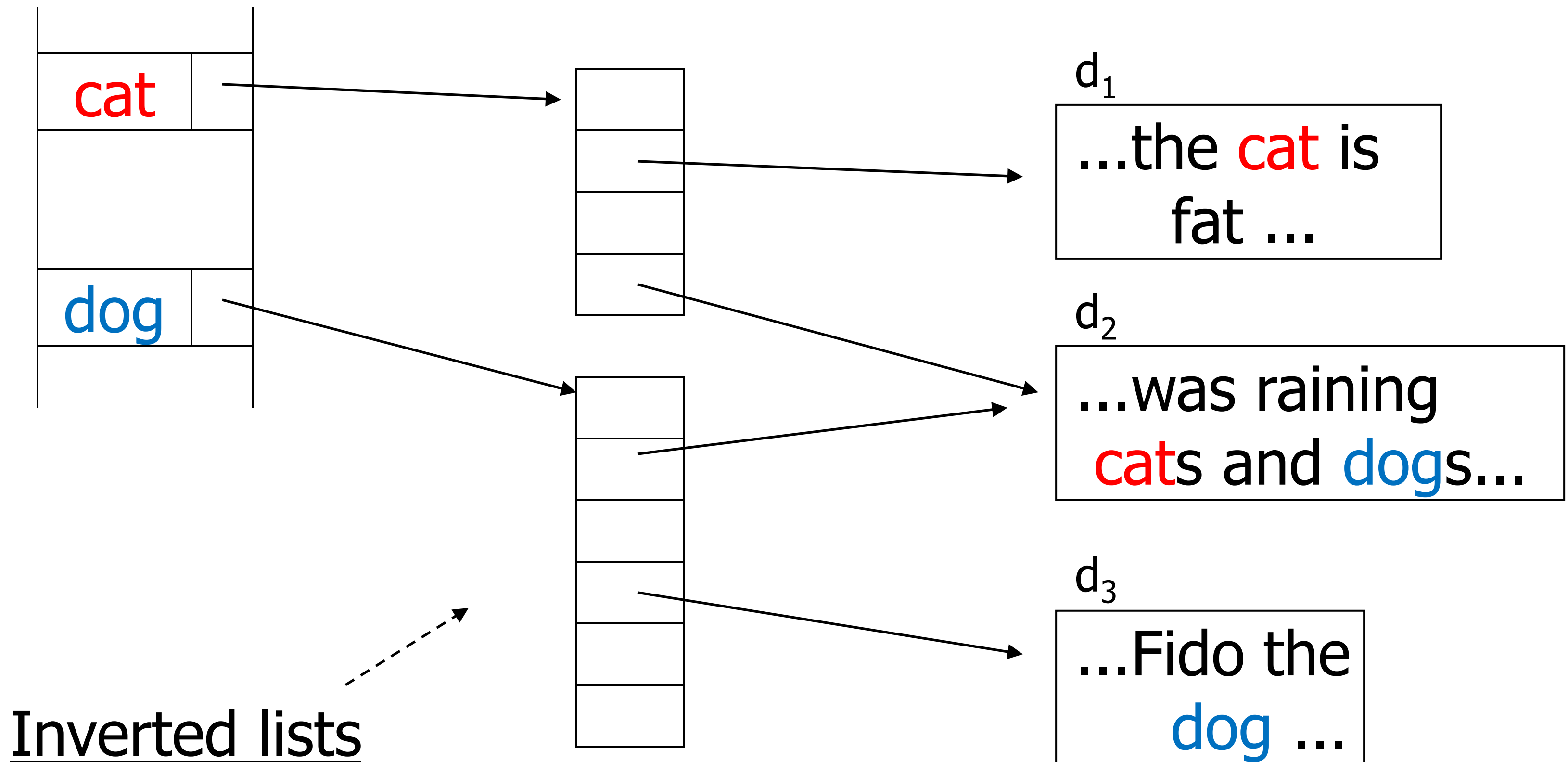






→ Πάρε την τομή του bucket για "Toy" και του bucket για "2nd Floor"

Text Information Retrieval (IR)

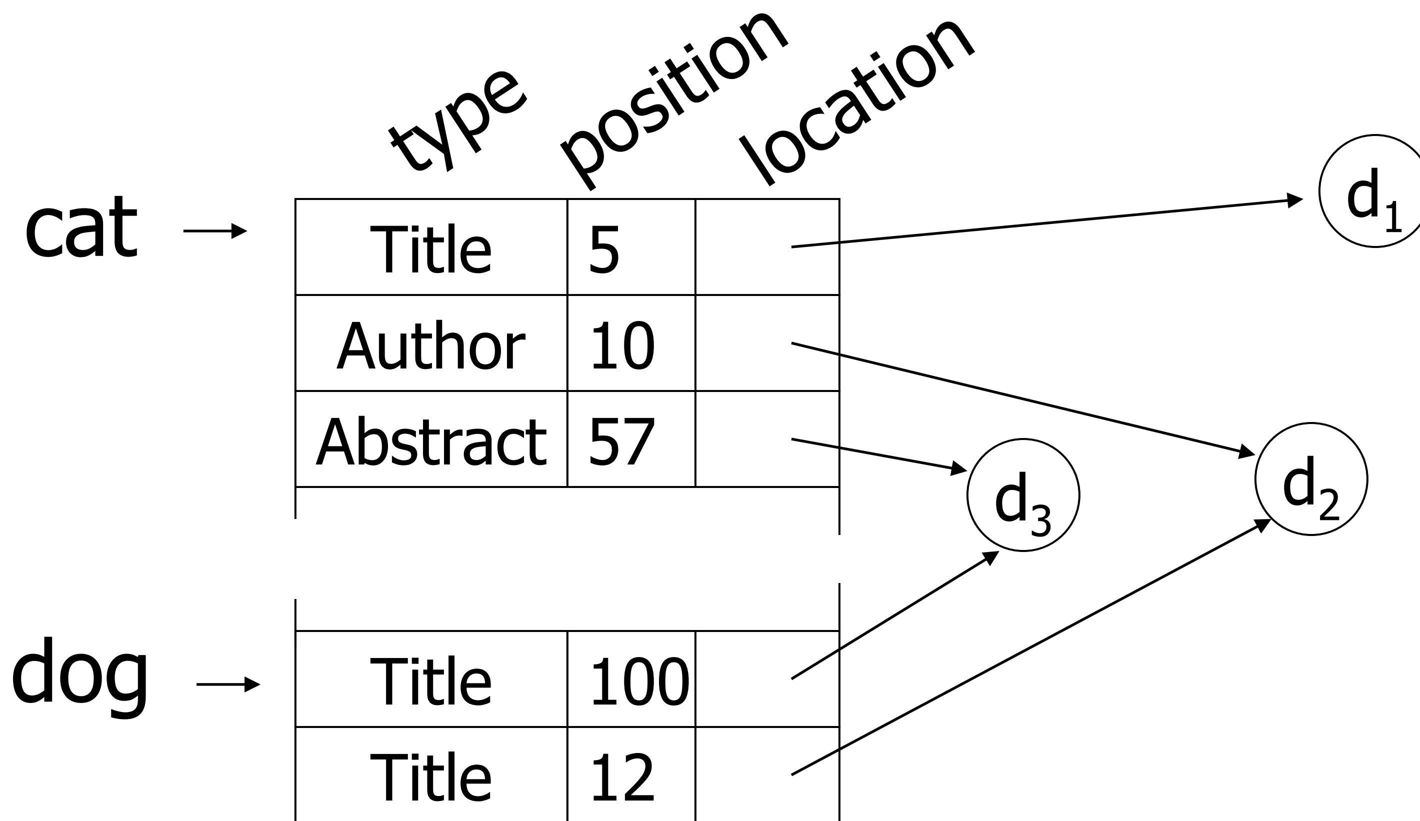


"Pointers" inside buckets may be document-ids or combinations of (doc-id, pos)

IR QUERIES

- Βρες κείμενα για “cat” και “dog”
- Βρες κείμενα για “cat” ή “dog”
- Βρες κείμενα για “cat” και όχι “dog”
- Βρες κείμενα για “cat” στον τίτλο
- Βρες κείμενα για “cat” και “dog” όπου οι 2 λέξεις αναφέρονται σε κοντινή απόσταση (πχ μικρότερη από 5 λέξεις)

Επιπλέον πληροφορία στις ανεστραμμένες λίστες



Μία άλλη λύση: Vector space model

cat dog

w1 w2 w3 w4 w5 w6 w7 …

DOC = <1 0 0 1 1 0 0 …>

Query= <0 0 1 1 0 0 0 …>

DOT PRODUCT = 1 + …… = score

Conventional indexes

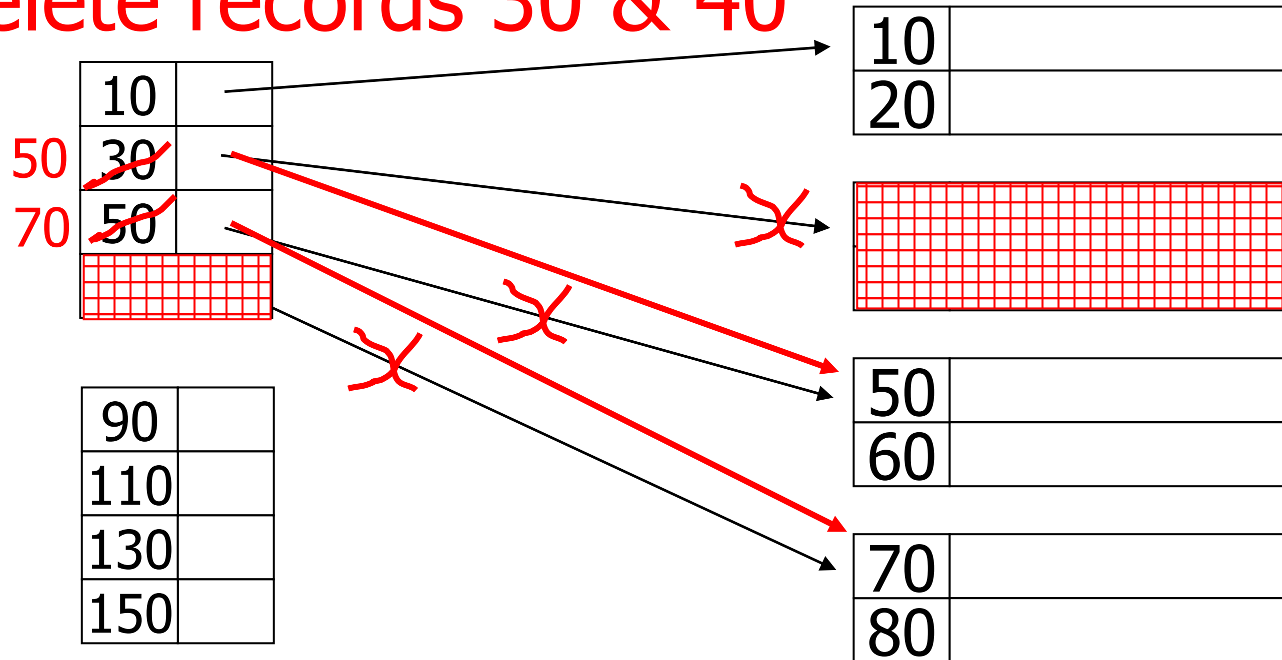
Πλεονεκτήματα:

- Απλές δομές
- Index is sequential file (αρχική κατασκευή)

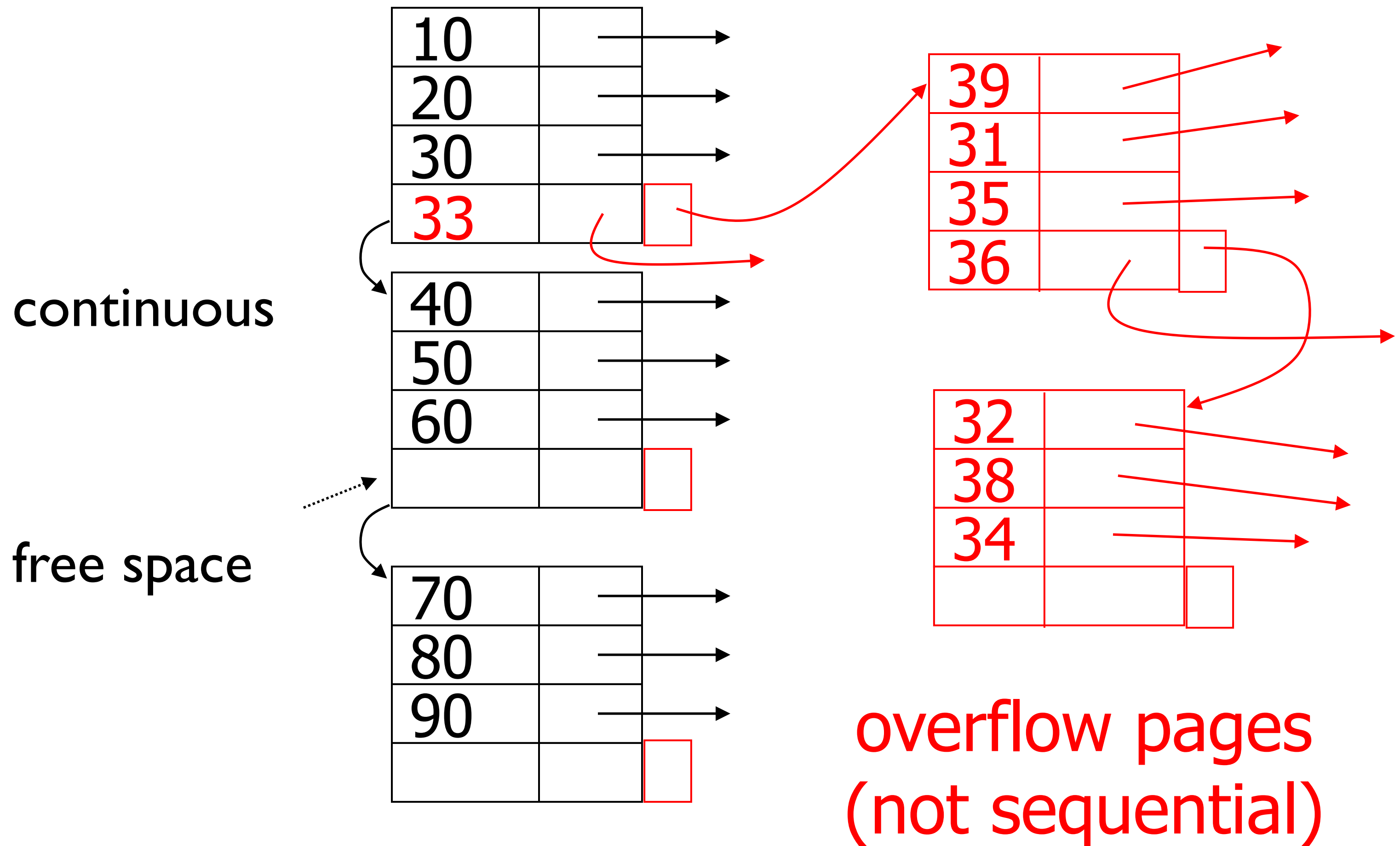
Μειονεκτήματα:

- Επιπλοκές σε inserts/deletes
- Lose sequentiality & balance

Διαγραφές σε αραιό ευρετήριο



Εισαγωγές



Που είμαστε

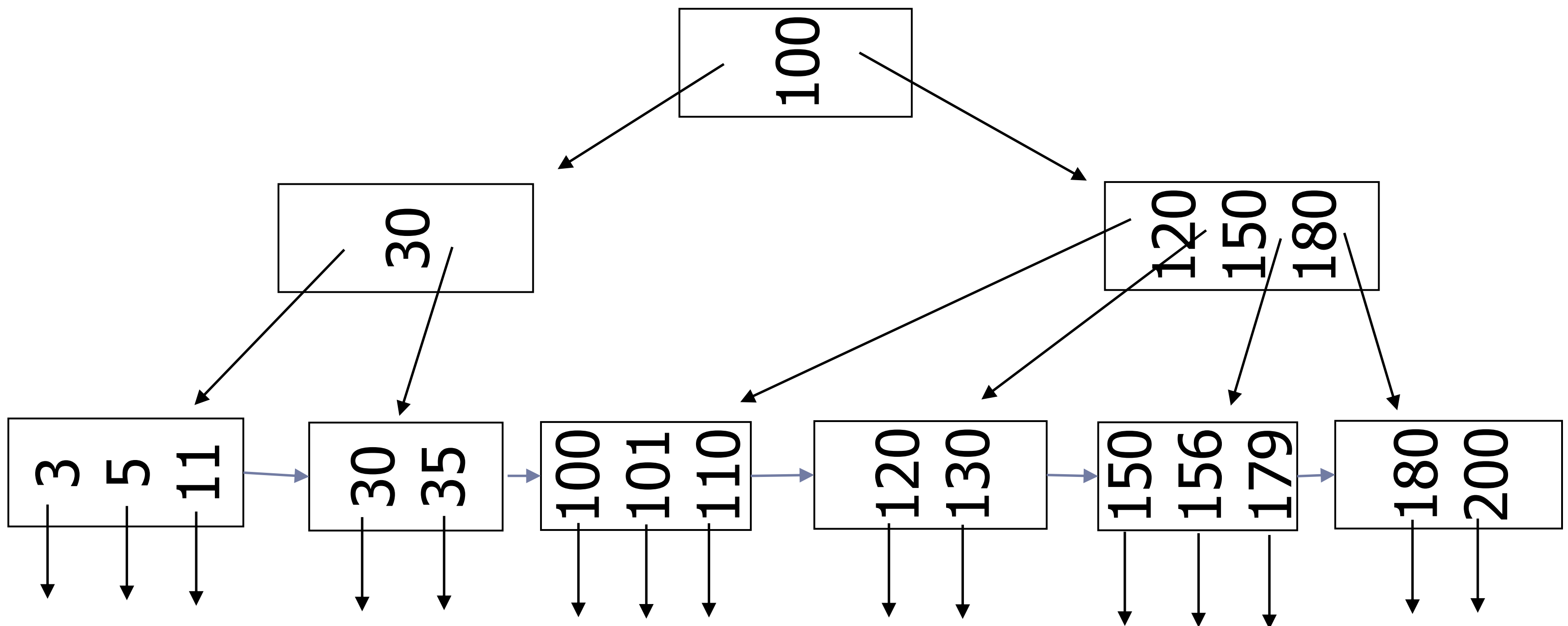
- Conventional indexes
- B-Trees $\Rightarrow$ Επόμενο θέμα
- Hashing schemes

B-trees (...για την ακρίβεια B$^+$-trees)

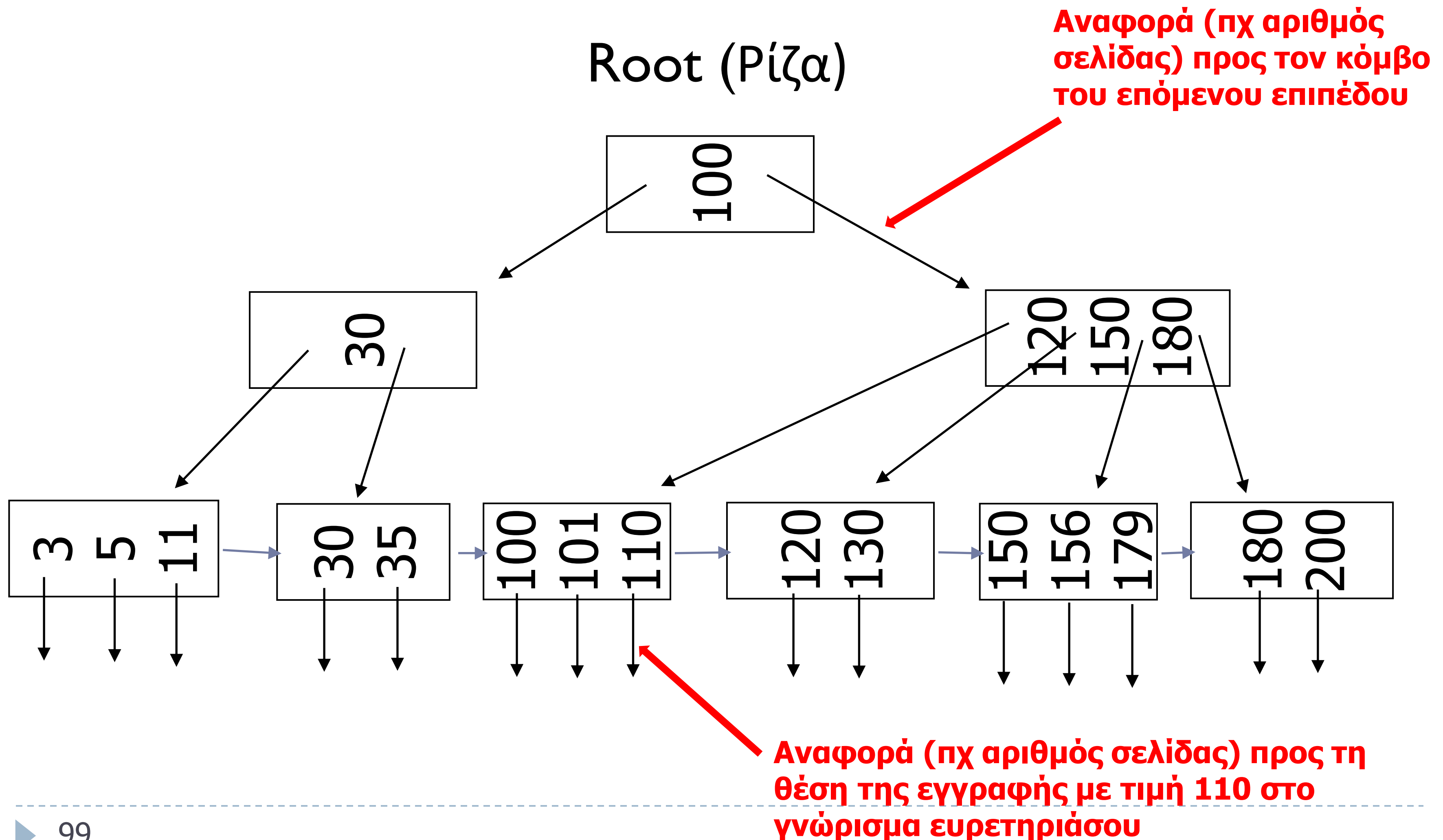
- Μία διαφορετική προσέγγιση
  - Δεν επιδιώκουμε το ευρετήριο να είναι σειριακό
    - Δύσκολο να επιτευχθεί σε σχέσεις με εισαγωγές/διαγραφές
  - Χρησιμοποιούμε ιεραρχική οργάνωση
  - Στόχος το δέντρο που προκύπτει να είναι ισορροπημένο (balanced): όλα τα φύλλα στην ίδια απόσταση από τη ρίζα
    - Ισορροπημένα δέντρα παρέχουν εγγυήσεις κατά την αναζήτηση
  - Added bonus: οι τιμές του γνωρίσματος ευρετηριοποίησης είναι διατεταγμένες στα φύλλα του δέντρου
    - Μπορεί να αξιοποιηθεί σε ερωτήματα που επιθυμούν διάταξη (πχ ORDER BY)

Θυμάμαι!

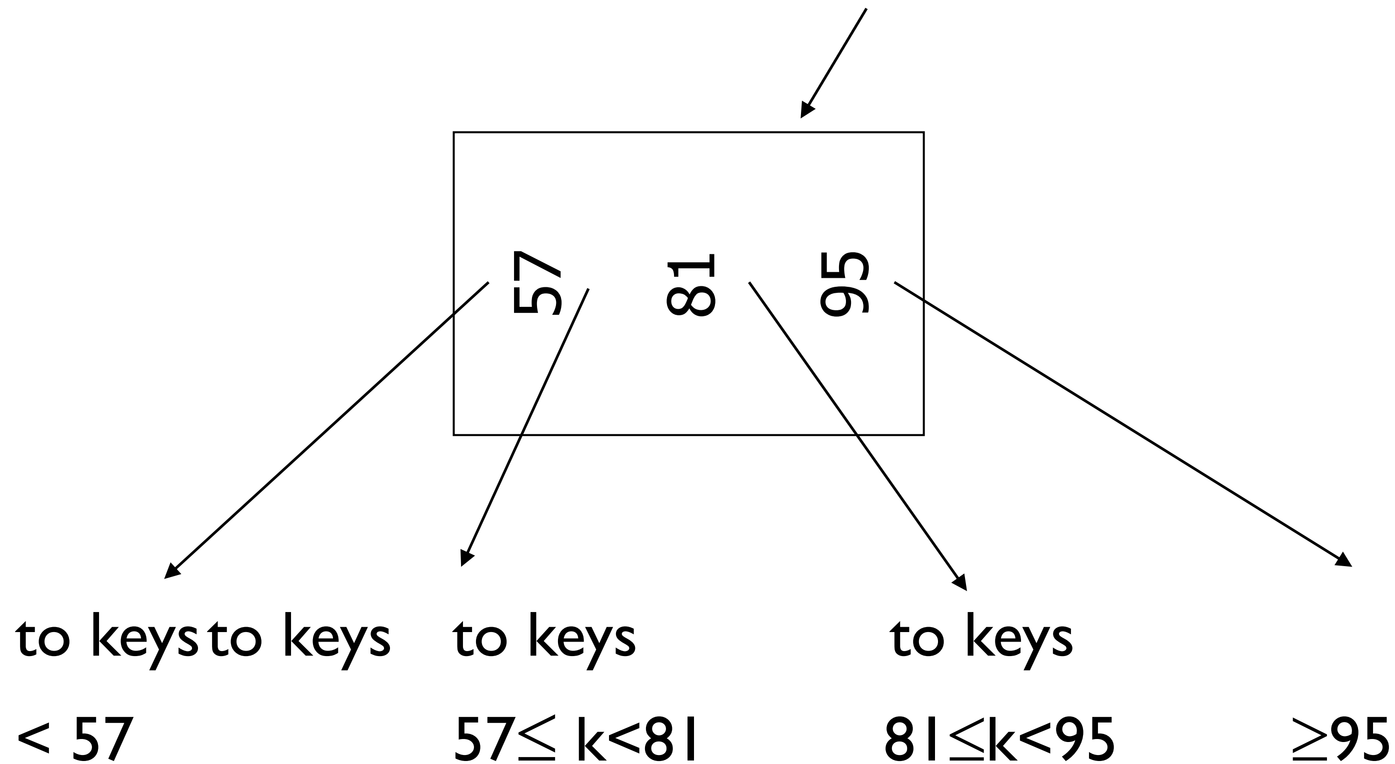
- Αναφερόμαστε σε δομές που είναι κατάλληλες για αποθήκευση στο δίσκο
  - Κάθε κόμβος του δέντρου αντιστοιχεί σε μια σελίδα στο δίσκο



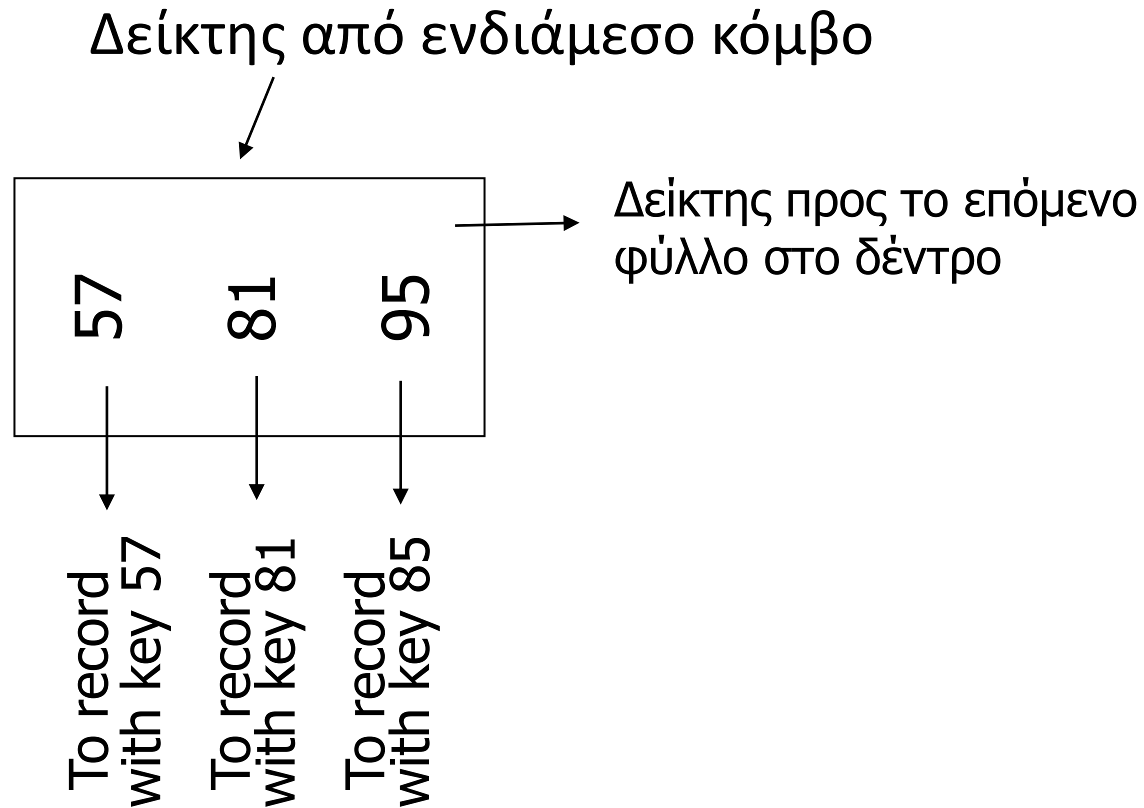
Παράδειγμα: B+-Tree (n=3)



Παράδειγμα ενδιάμεσου κόμβου (non-leaf node)



Παράδειγμα φύλλου (leaf node)



Μέγεθος κόμβων δέντρου (n)

- n+1 pointers
- n keys

Δε θέλουμε οι κόμβοι να είναι πολύ άδειοι (...σπατάλη χώρου)

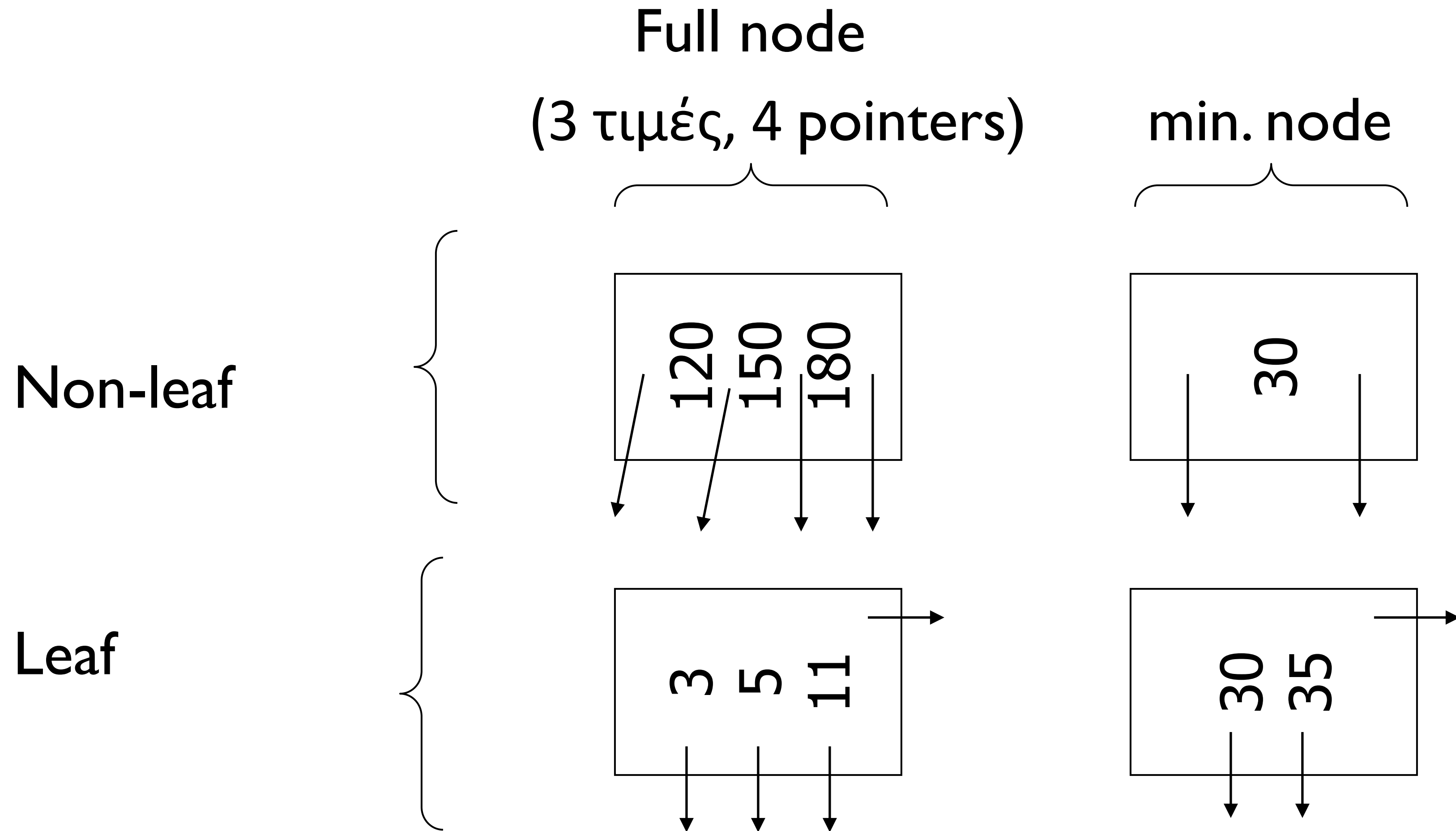
- Ένας κόμβος πρέπει να περιέχει (ανάλογα με τον τύπο του) τουλάχιστον

Non-leaf: $\lceil (n+1)/2 \rceil$ pointers **Ελάχιστος αριθμός κόμβων παιδιών**

Leaf: $\lfloor (n+1)/2 \rfloor$ pointers to data **Ελάχιστος αριθμός τιμών γνωρίσματος**

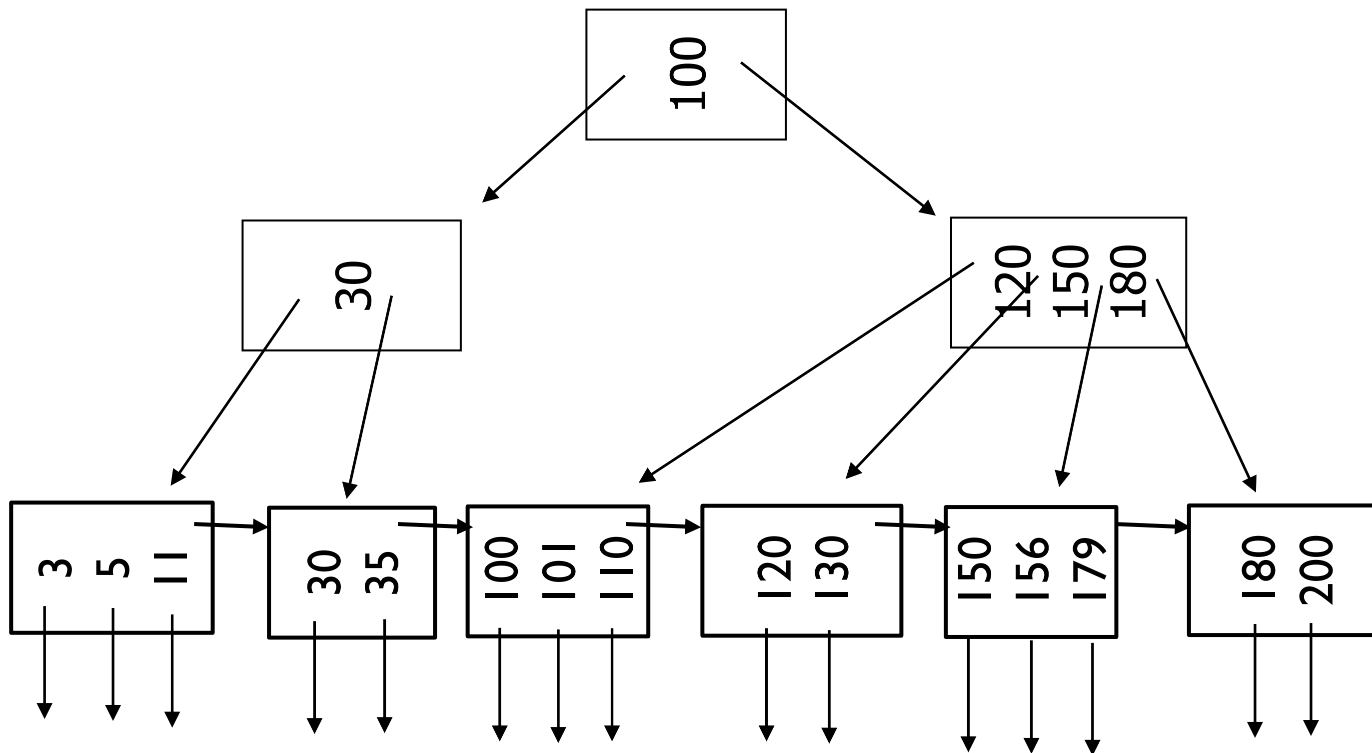
(η ρίζα εξαιρείται από τον κανόνα)

Παράδειγμα: n=3



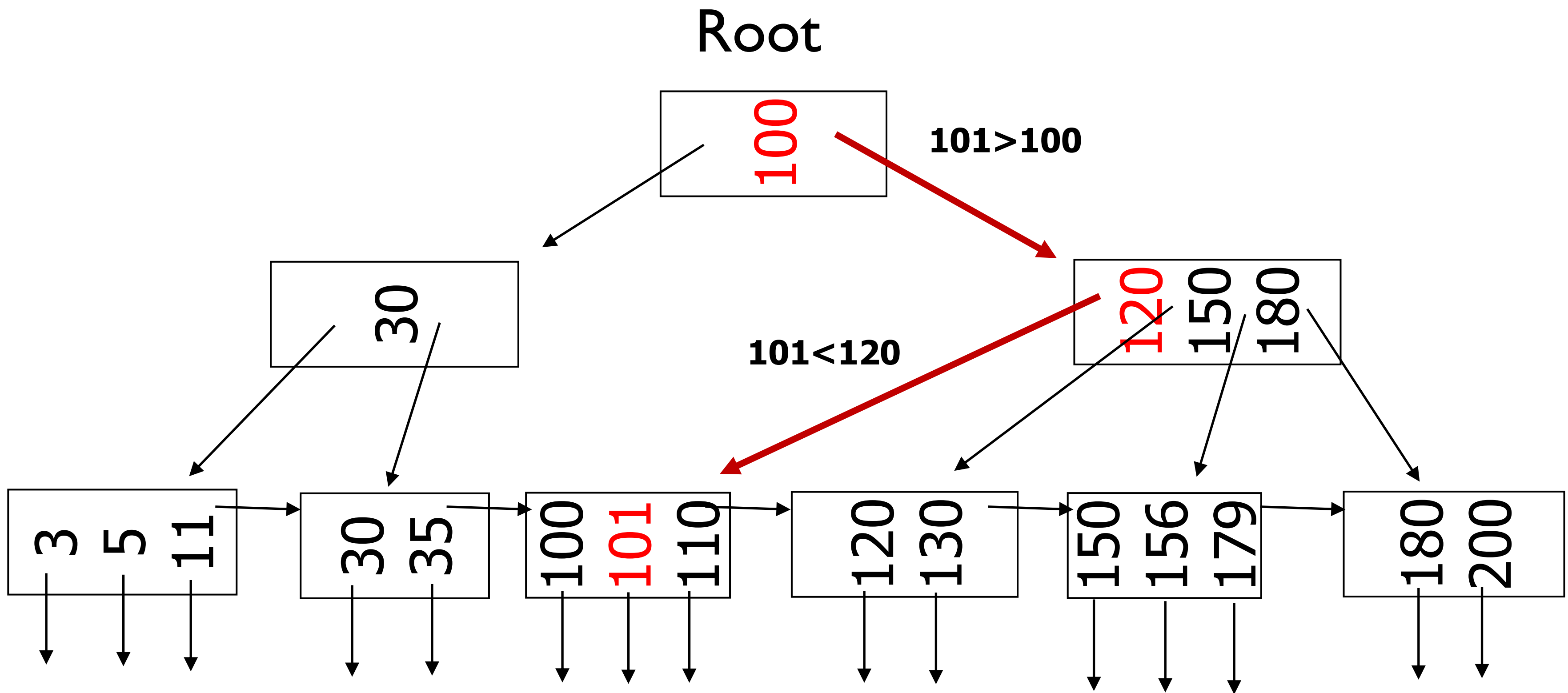
Επίσης

- Όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο (balanced tree)



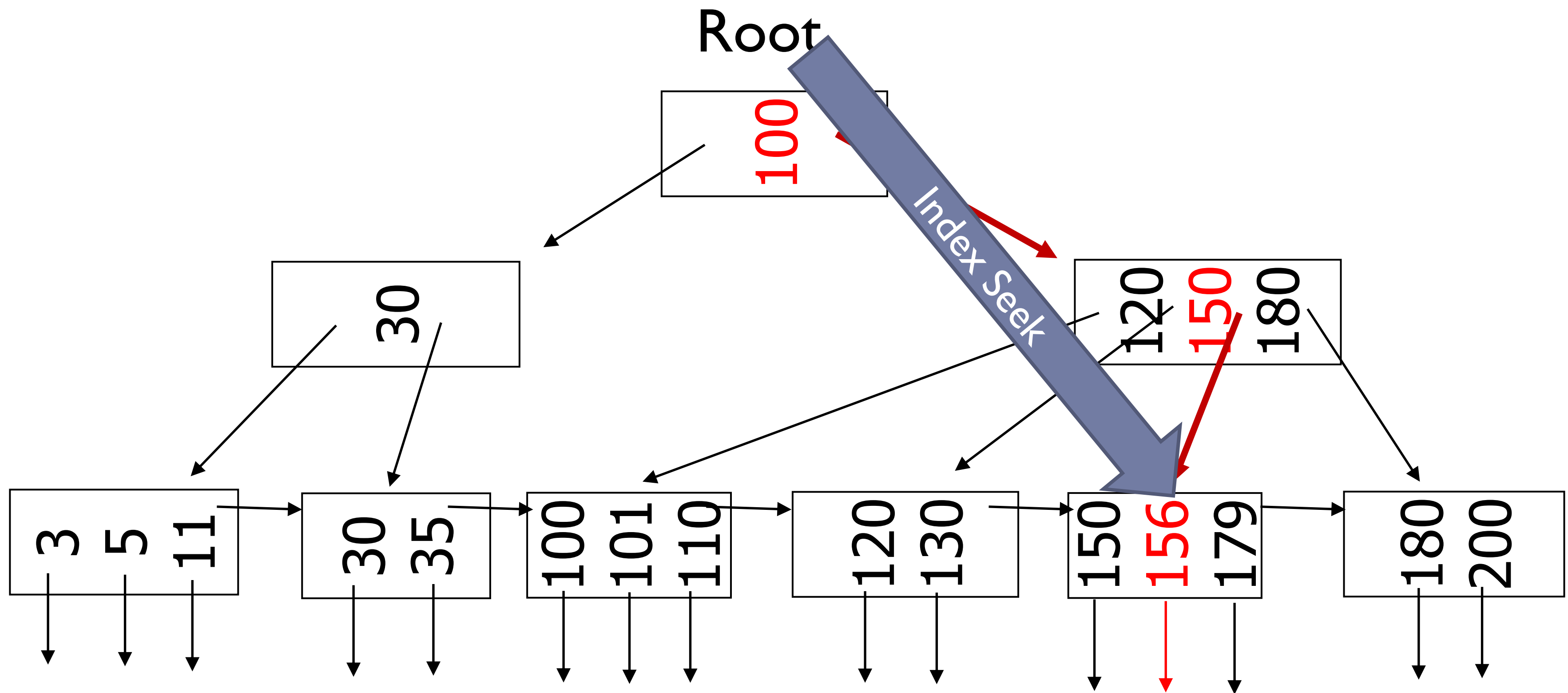
Αναζήτηση (Exact Match/Point Queries)

Search key=101



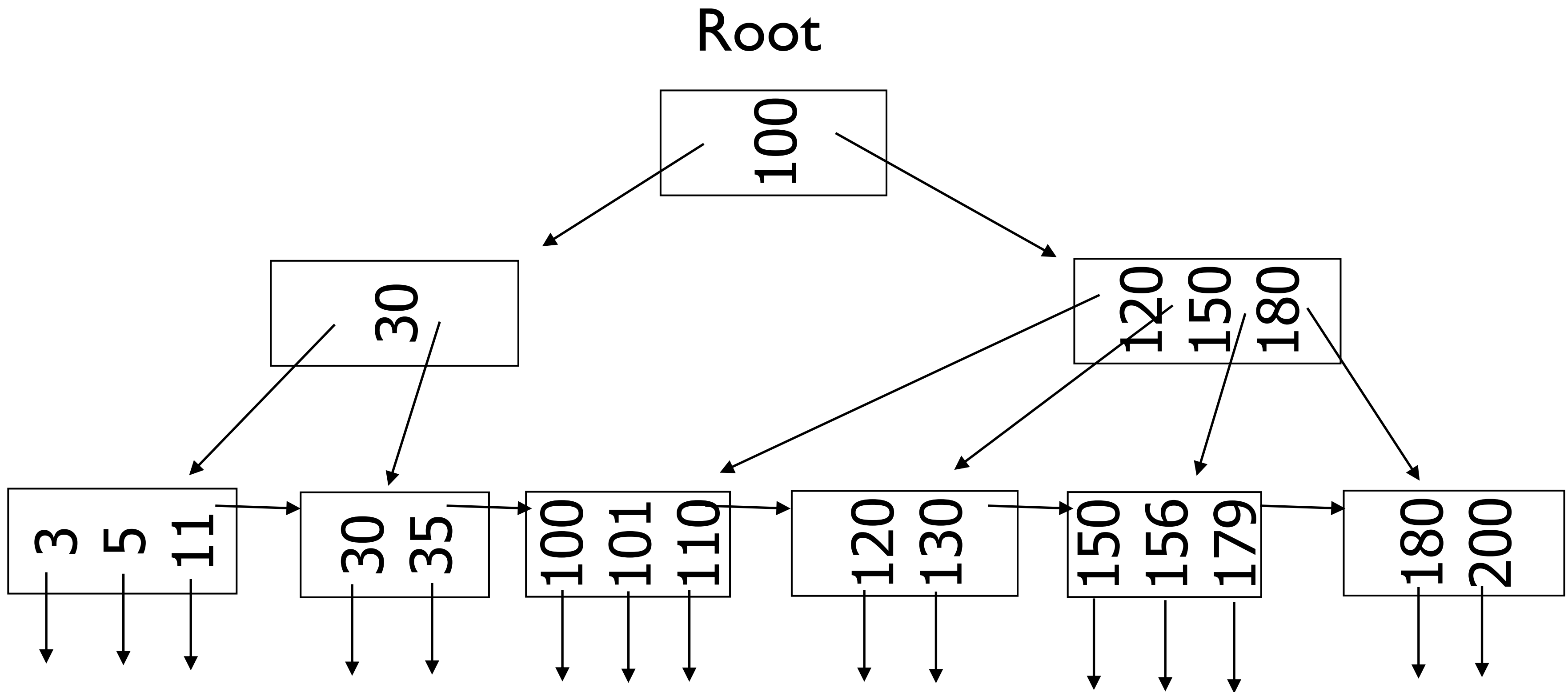
Index Seek

Search key=156



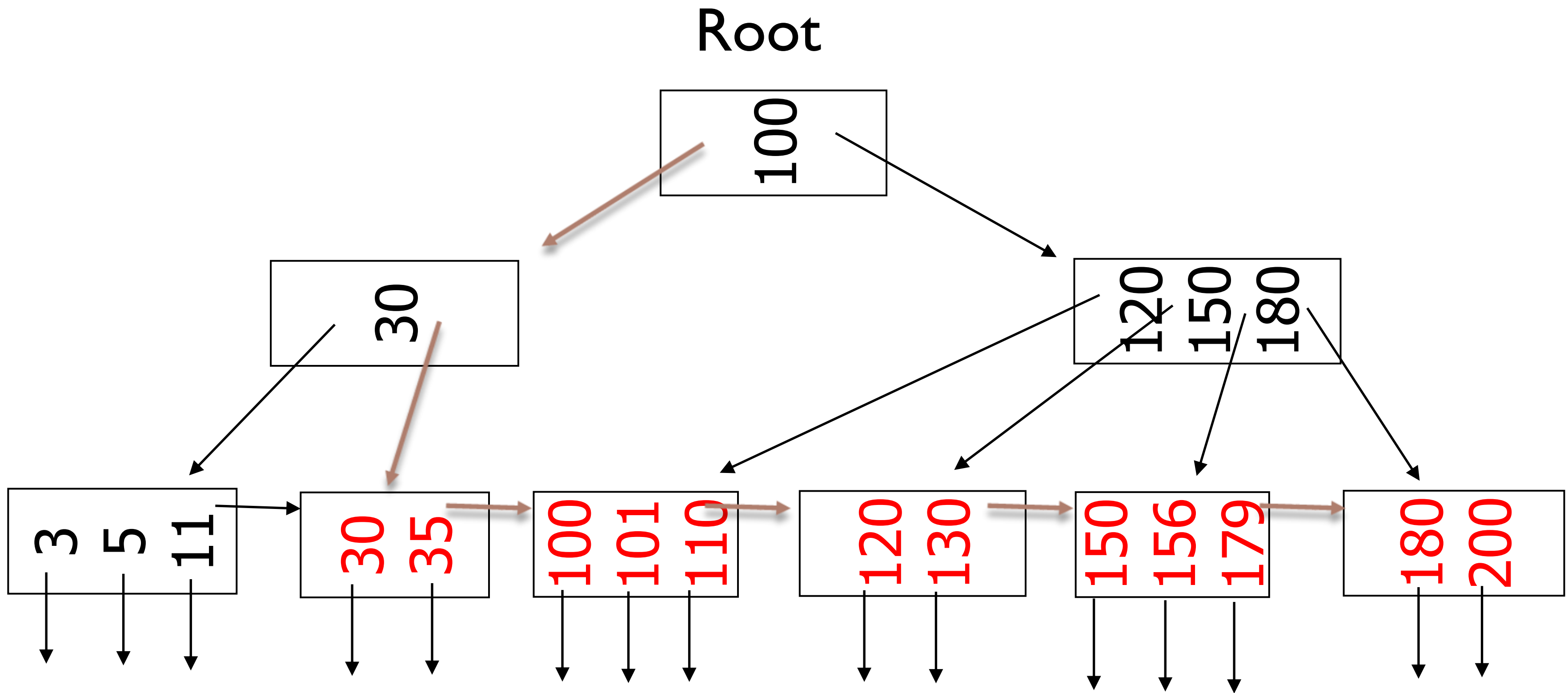
Range Queries

- Search key≥30 ?
- 115 ≤ Search key ≤ 155 ?



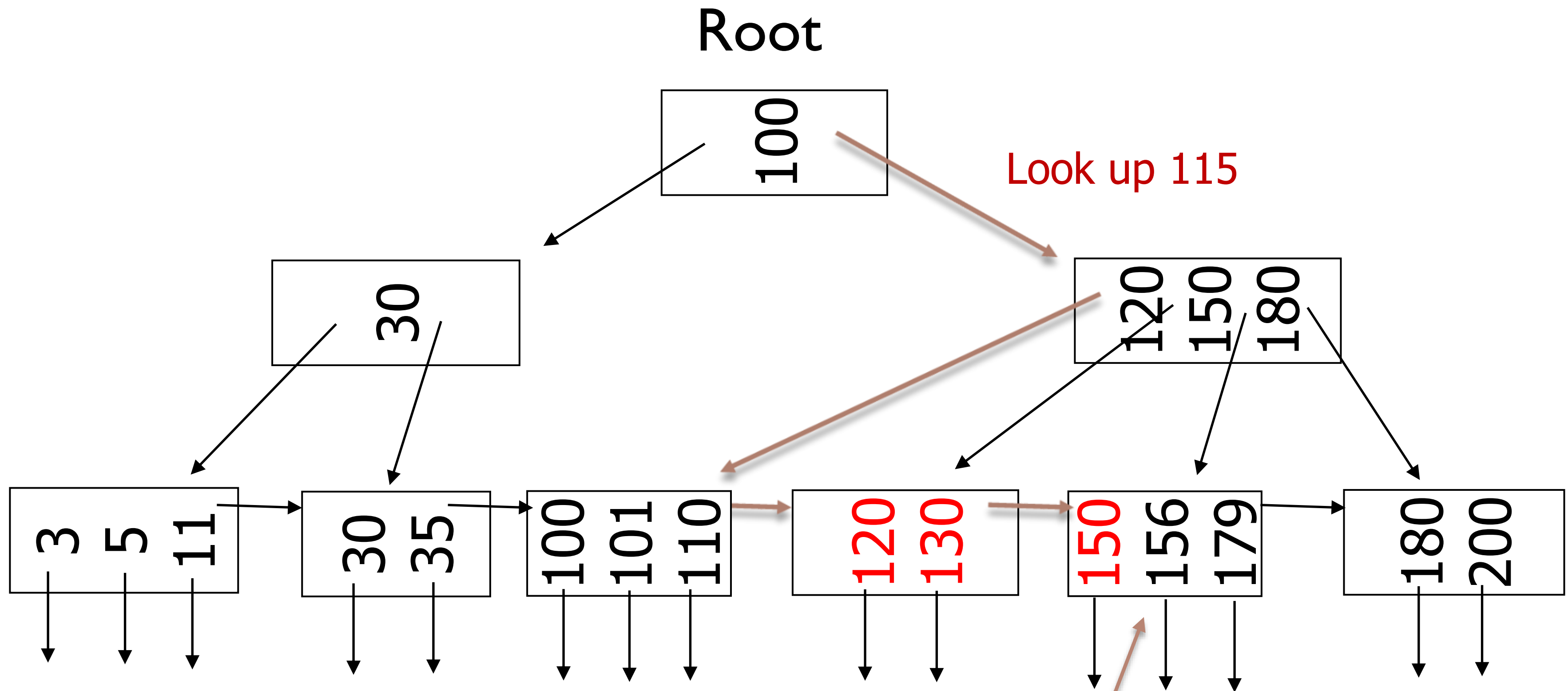
Range Queries

- Search key≥30 ?



• 115<=Search key ≤155

Range Queries

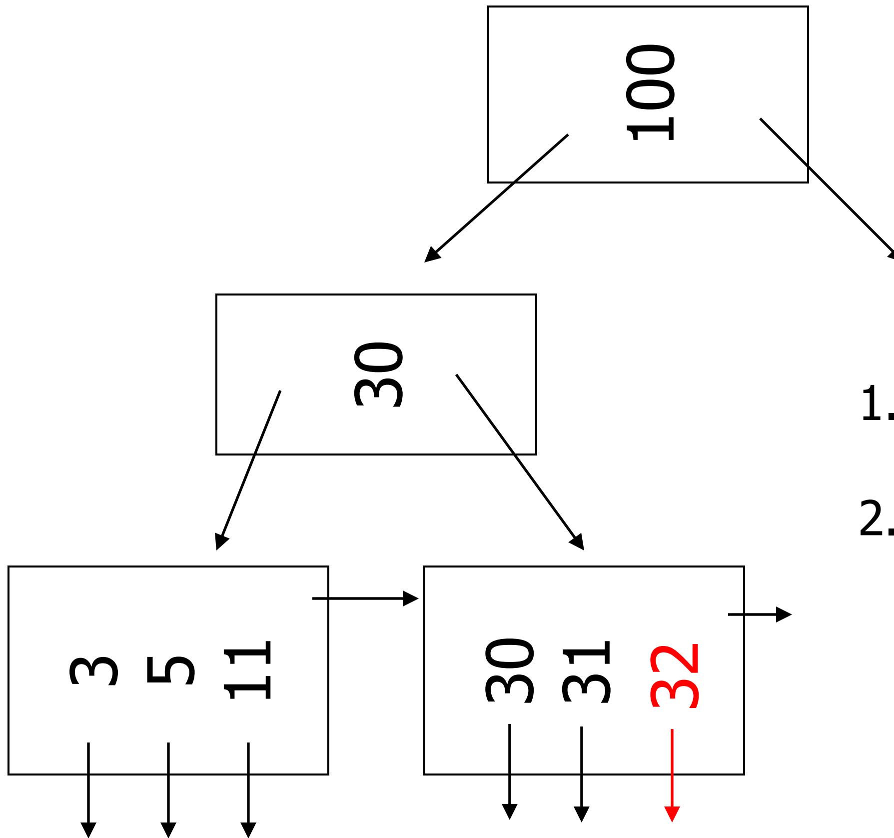


B+tree: Εισαγωγή

- Κάνουμε πρώτα αναζήτηση της τιμής που θα εισάγουμε
- Εύκολη περίπτωση
  - (a) Υπάρχει χώρος στο φύλλο
- Ποιο δύσκολη: Υπερχείλιση (overflow)
  - (b) leaf overflow
  - (c) non-leaf overflow
  - (d) new root

(a) Insert key = 32

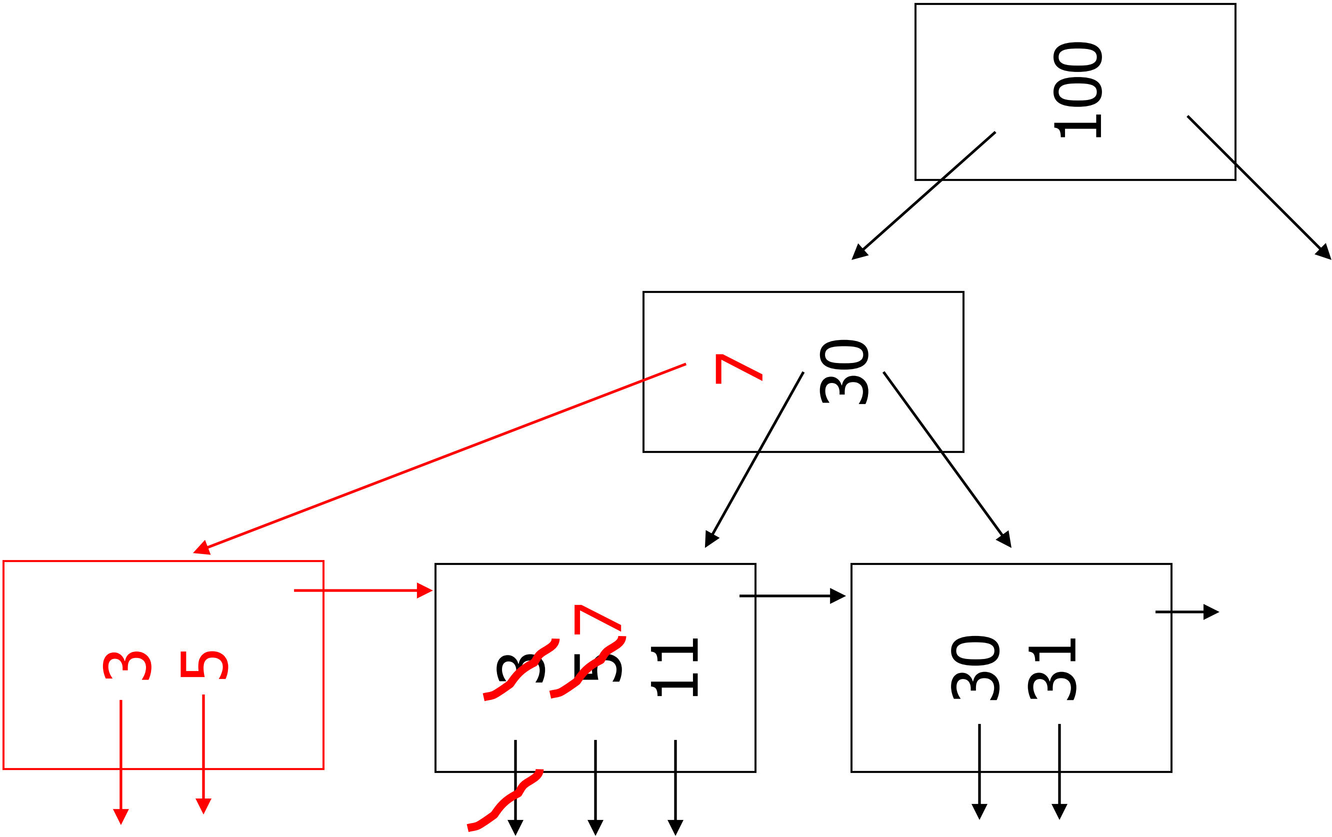
n=3



1. Κάνουμε αναζήτηση με τη τιμή 32 που θέλουμε να εισάγουμε
2. Εισάγουμε την τιμή και τον δείκτη στο φύλλο που καταλήξαμε

(b) Insert key = 7

n=3



(c) Insert key = 160

n=3

100 160

120 150 180

180

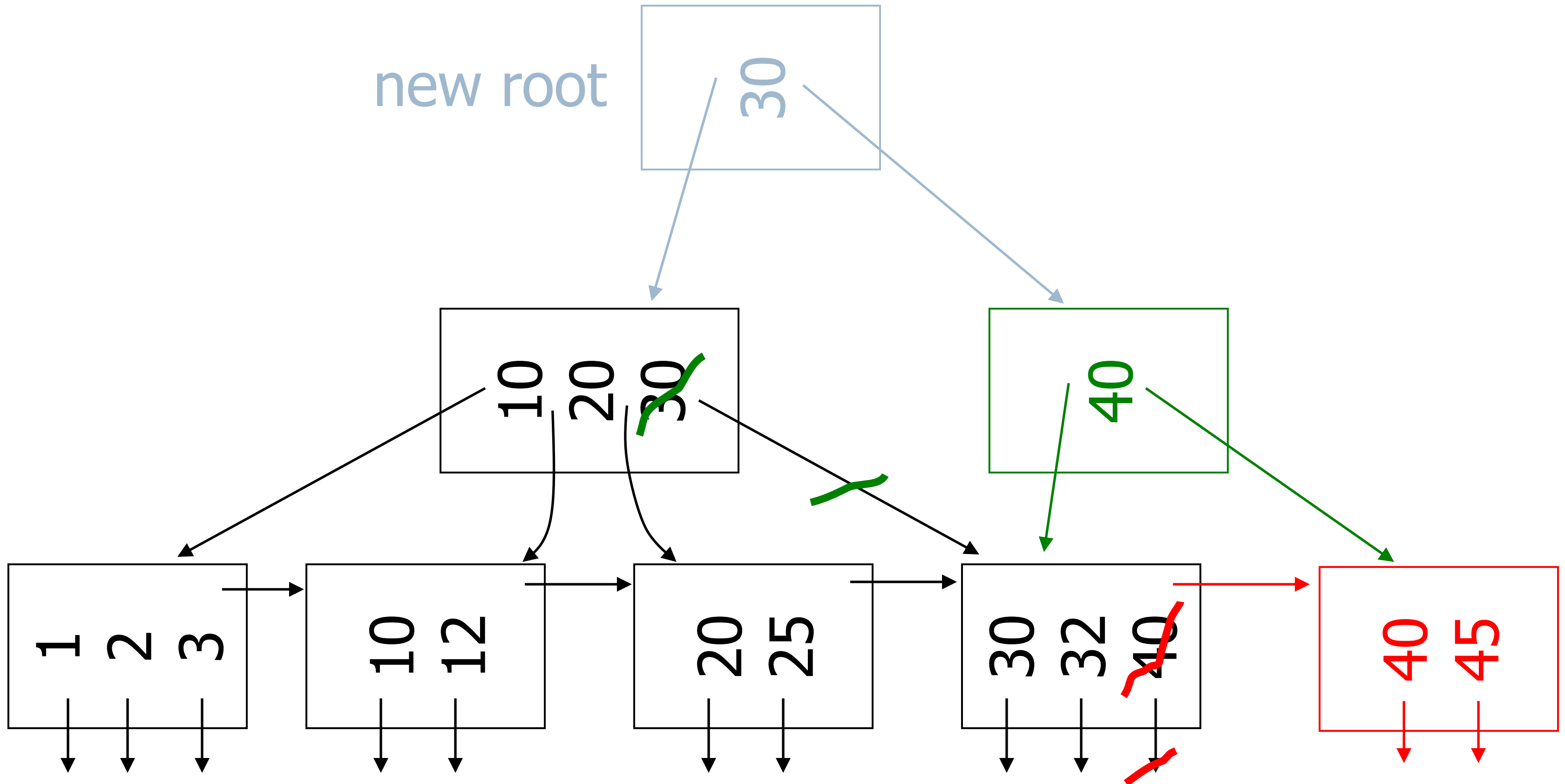
150 156 179

160 179

180 200

(d) New root, insert 45

n=3



Deletion from B-tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

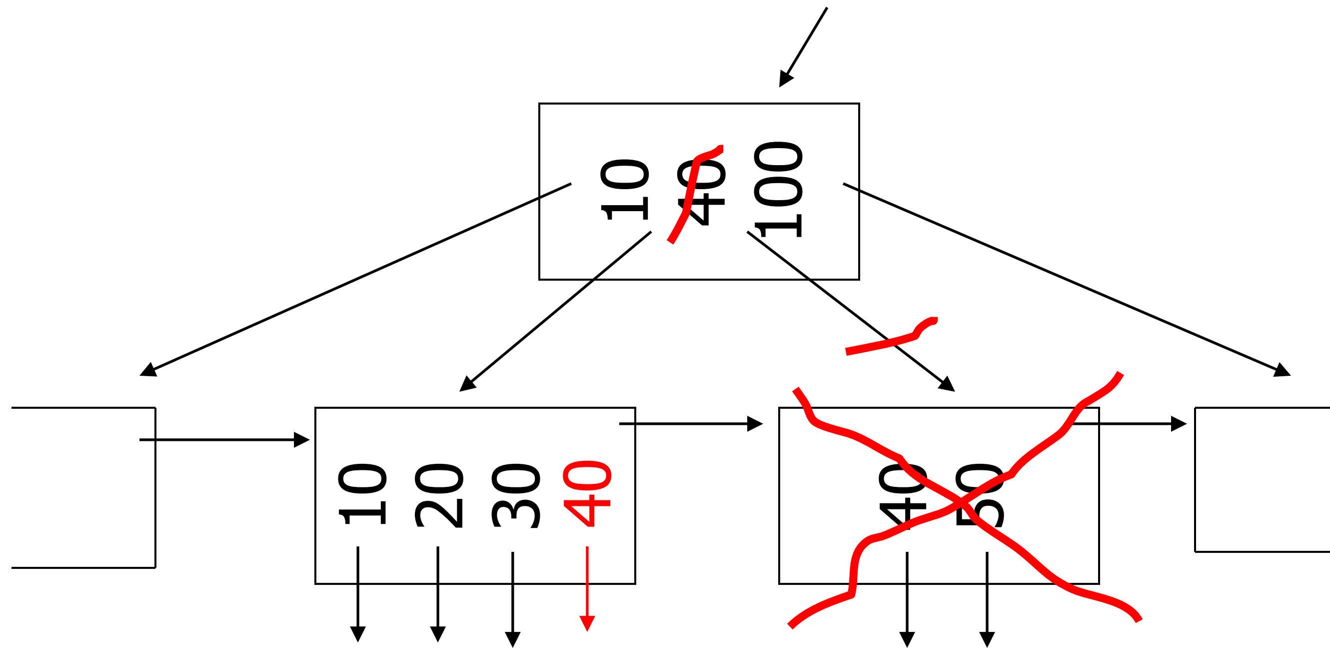
(c) Re-distribute keys

(d) Cases (b) or (c) at non-leaf

(b) Coalesce with sibling

n=4

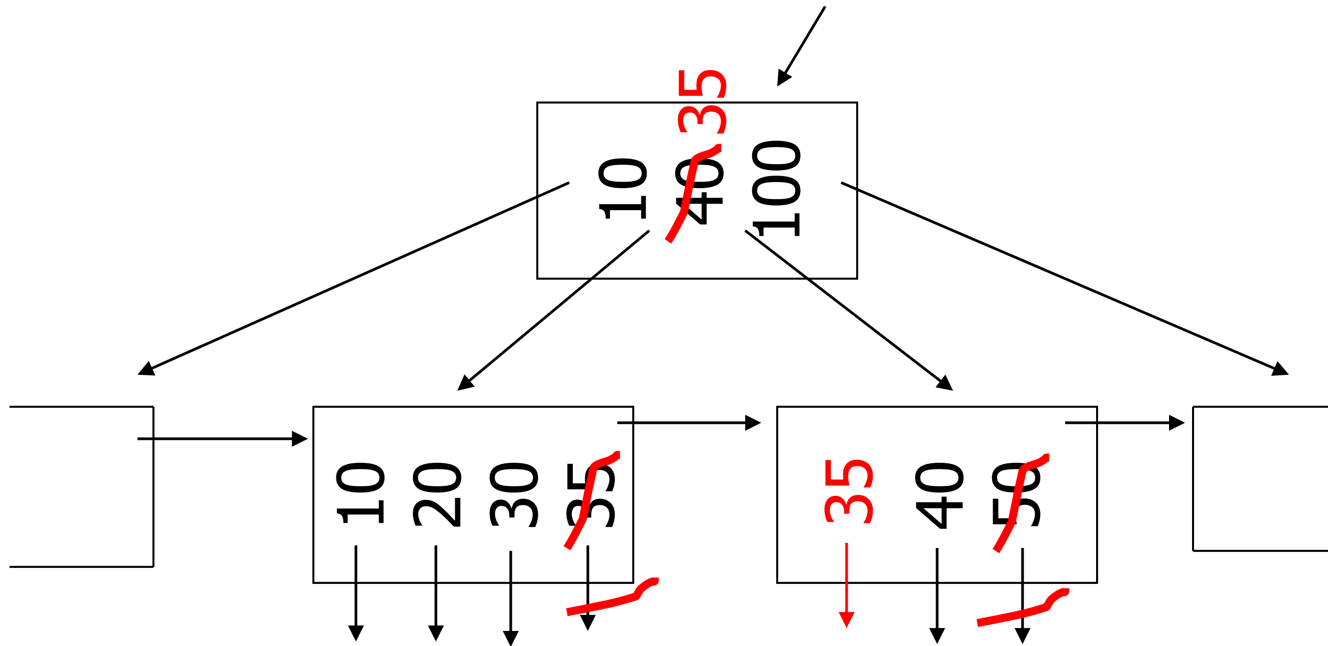
- Delete 50



(c) Redistribute keys

n=4

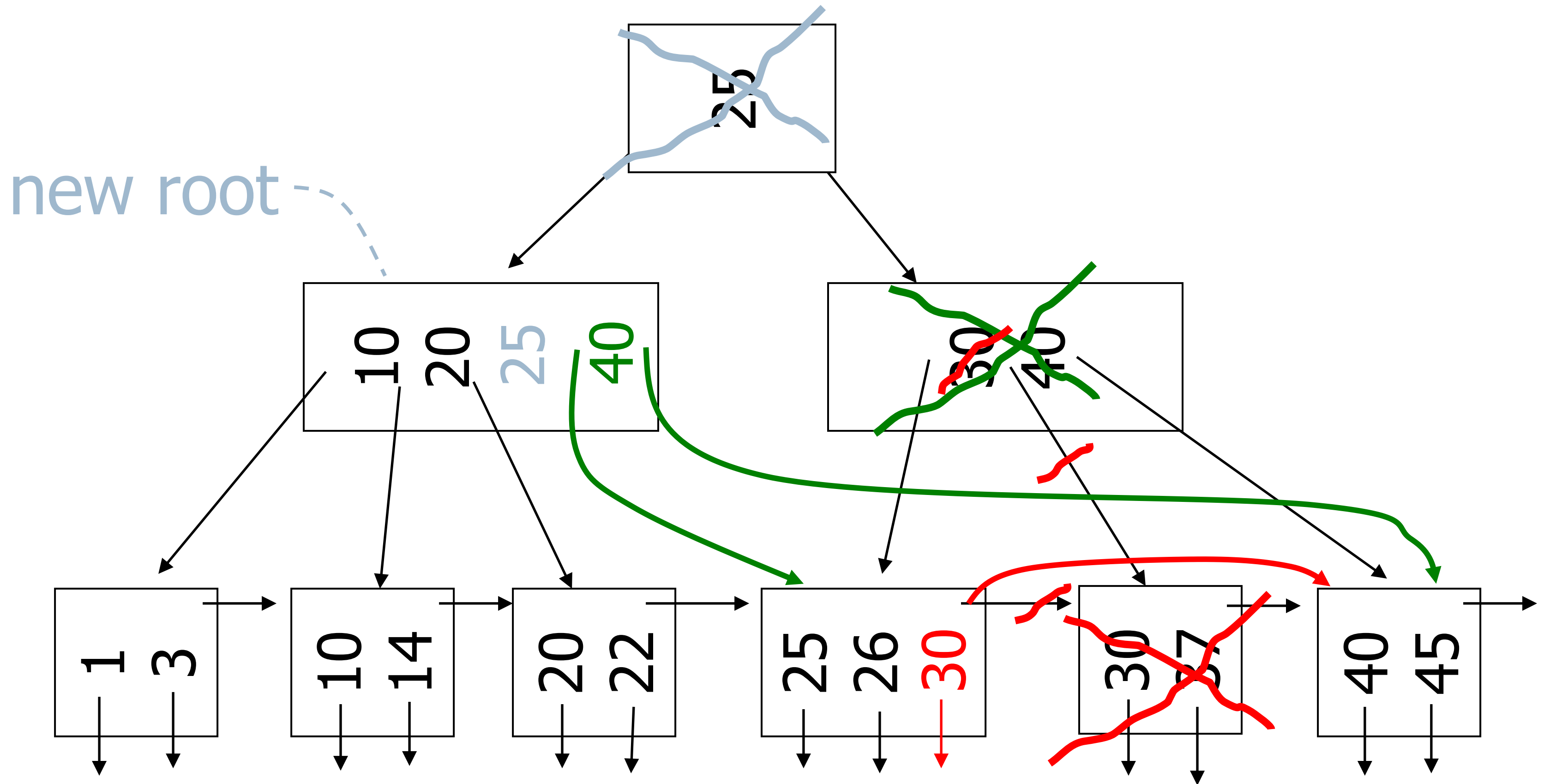
- Delete 50



(d) Non-leaf coalese

- Delete 37

n=4



B-tree deletions in practice

- Often, coalescing is <u>not</u> implemented
  - Too hard and not worth it!
  - Consider a delete followed by an insert
  - Same with updates

Creating index: practical examples

- Assume table emp(<u>eid</u>, ssn, fname, lname,city,salary)
- Want to retrieve employee data based on their last name

CREATE INDEX emp_ename ON emp(lname)

index name *table & index key*

- Also want to perform quick lookups based on the social security number of an employee
  - Notice that duplicate ssn values are not allowed

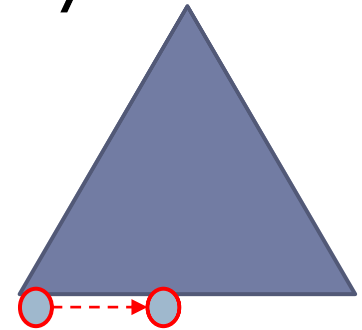
CREATE UNIQUE INDEX *ssn ON emp(ssn)*

Updates

- Σε ένα ΣΔΒΣ οι ενημερώσεις στο B-tree γίνονται γιατί το περιεχόμενο από κάποιο base table έχει τροποποιηθεί και η αλλαγή επηρεάζει το index key.
- Πχ index on emp(salary)
- Update 1: change employ city from "Athens" to "Patras" does not trigger an index update
- Update 2: give 10% increase to employ "John" triggers an index update

The Halloween Problem
(System R – 1976)

- Assume table Employees(EmdId, Name, Salary)
  - Have a B-tree index on Salary
- Want to give a 10% raise to all low-income employees
  - Update Employees Set Salary = Salary * 1.1
    Where Salary < 25000;



- Query plan:
  - Find all employees with Salary < 25000 using the B-tree
    - Start from left-most leaf, move to the right
  - Follow pointers, give10% raise
    - Each new salary value triggers a delete followed by an insert with the updated value in the B-tree. This "pushes" the employee further to the right.

Halloween Problem

- What is the outcome of this process?
- Possible solution: when the optimizer is making a plan for processing an update statement, it doesn't use an index that is based on the field that is being updated.

Is LRU a good policy for B-tree buffers?

$\rightarrow$ Of course not!

$\rightarrow$ Should try to keep root in memory at all times

(and perhaps some nodes from second level)

Σκεφτείτε το παρακάτω

- Δίνεται αρχείο EMPLOYEES.TXT με στοιχεία (κωδικός υπαλλήλου, όνομα, επώνυμο, τμήμα) για 100000 υπαλλήλους.
- Θέλετε να φορτώσετε τις εγγραφές του αρχείου στη σχέση EMPLOYEES(eid,fname,lname,deptid) που θα δημιουργήσετε και να φτιάξετε ευρετήριο στο deptid (κωδικός τμήματος).
- Ποια από τις παρακάτω ακολουθίες εντολών, A ή B, ενδέχεται να εκτελεστεί ταχύτερα και γιατί;

A:

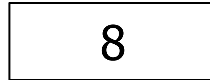
```
create table EMPLOYEES(eid int, fname varchar(40), lname varchar(40), deptid int);
create index I on EMPLOYEES(deptid);
load data local infile "EMPLOYEES.txt" into table EMPLOYEES;
```

B:

```
create table EMPLOYEES(eid int, fname varchar(40), lname varchar(40), deptid int);
load data local infile "EMPLOYEES.txt" into table EMPLOYEES;
create index I on EMPLOYEES(deptid);
```

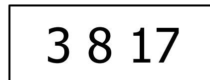
One at a time updates (insertions) n=3

- Insert 8

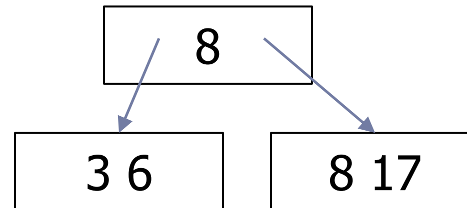


- Insert 3

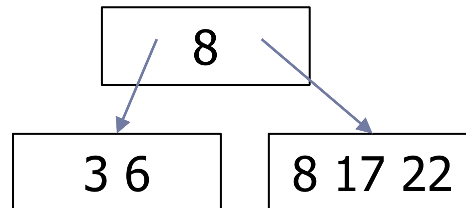
- Insert 17



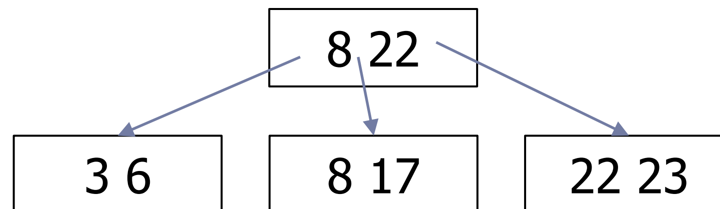
- Insert 6



- Insert 22



- Insert 23



- Observations:
  - Cost of reorganizations (per new item)
    - Leaves + intermediate nodes
  - Many leaf nodes are half-full!
    - Waisted space

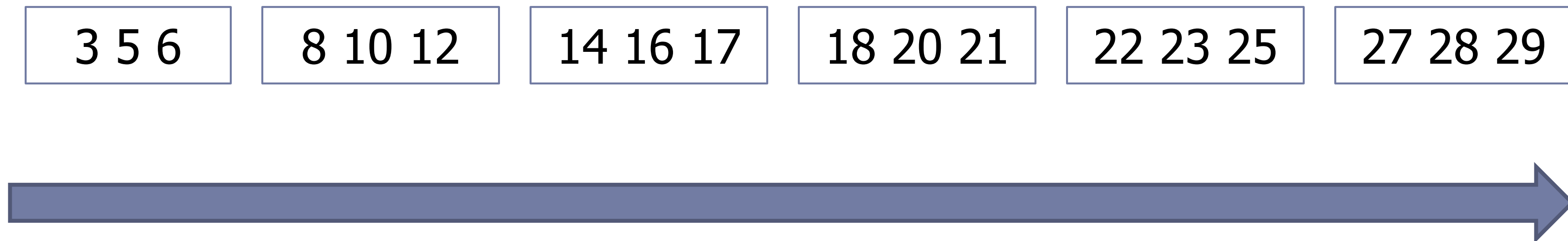
Bulk Loading a B^+-tree (n=3)

Read and **sort** all index keys in the relation

3 5 6   8 10 12   14 16 17   18 20 21   22 23 25   27 28 29

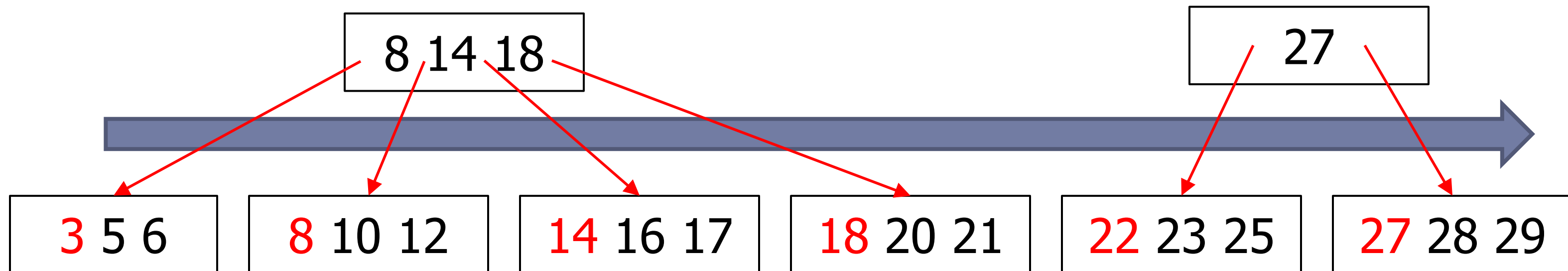
Bulk Loading a B^+-tree (n=3)

Scan sorted list of keys and create leaves



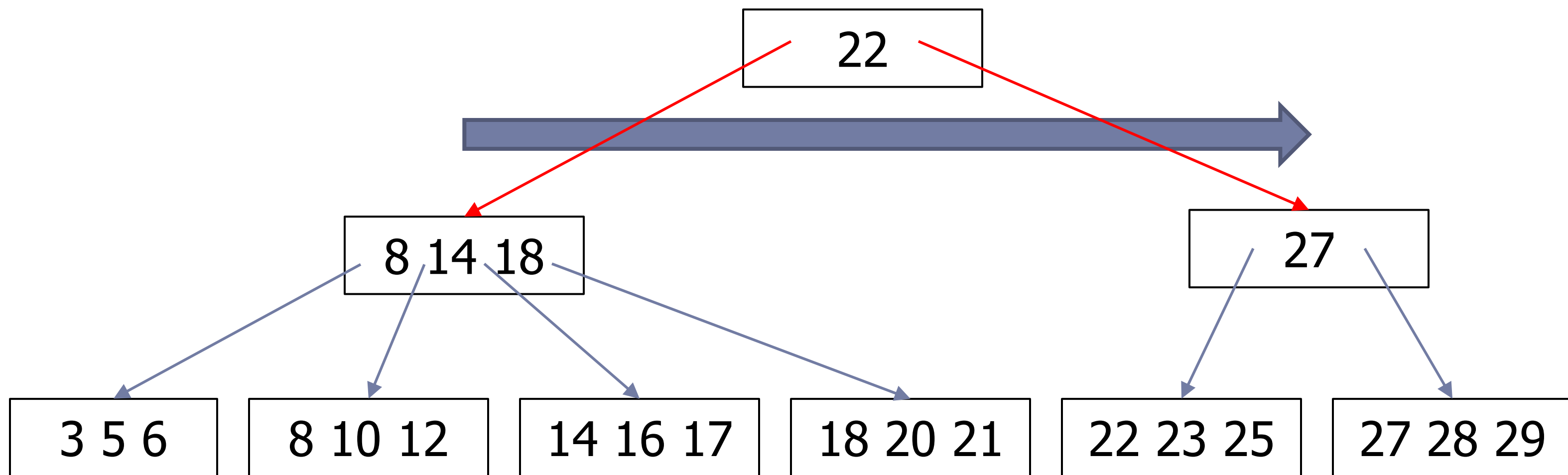
Bulk Loading a B^+-tree (n=3)

- Recursively create intermediate nodes by passing upwards nodes (page number + min key in the subtree) from the previous level

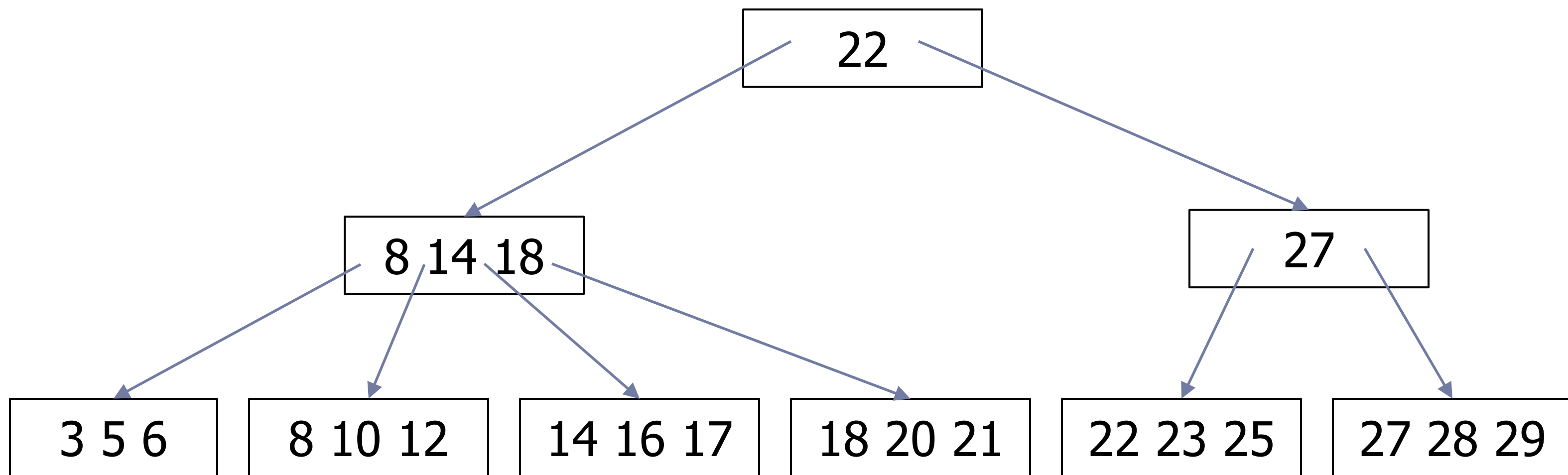


Bulk Loading a B^+-tree (n=3)

Last iteration creates the Root page



Bulk Loading a B^+-tree (n=3)



Ευρετήριο σε σύνθετο κλειδί

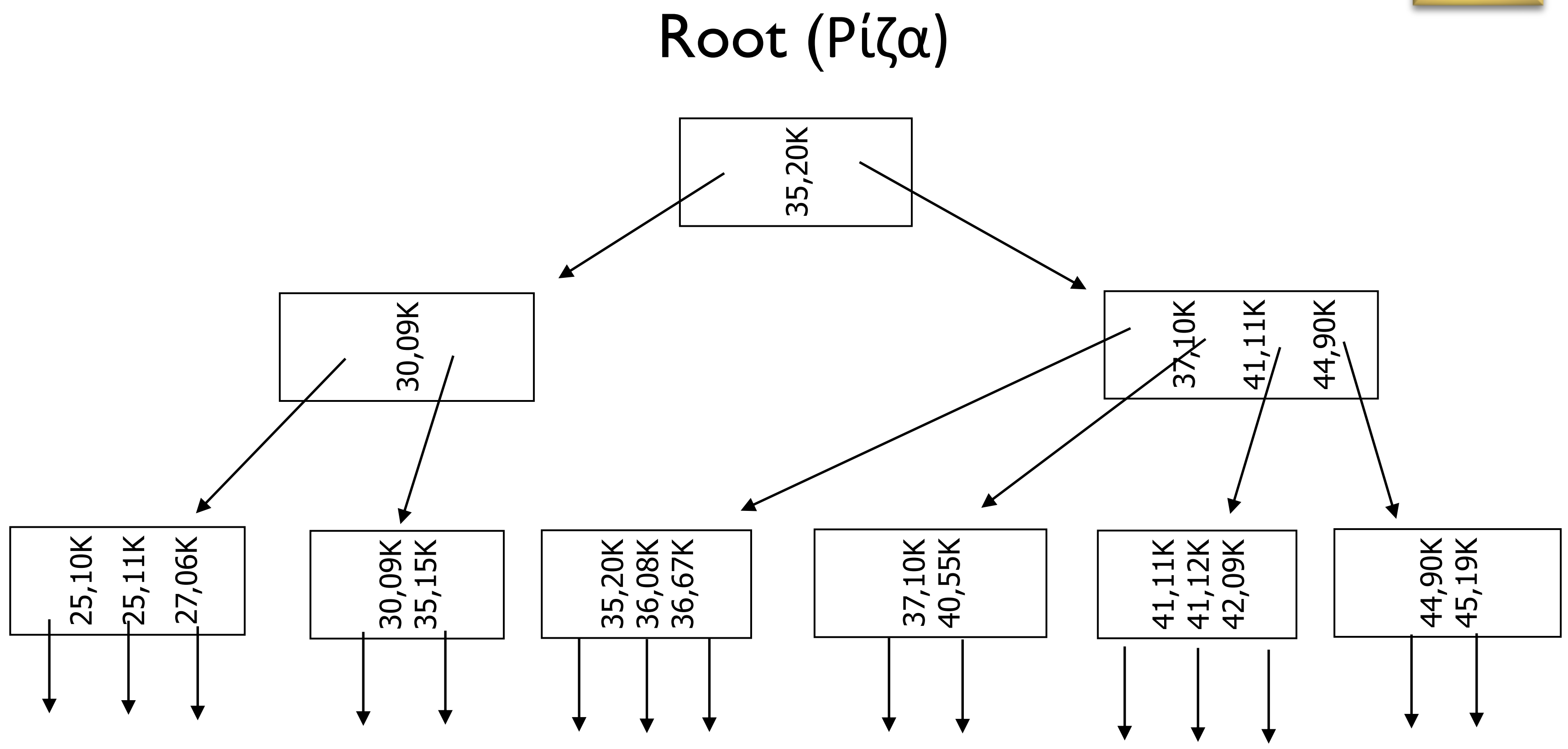
- Πίνακας Employee(eId, eName, eAge, eSalary)
  - Περιορισμοί: 18 ≤ eAge ≤ 65
    00K ≤ eSal ≤ 99K
- Πχ (1001,"Jim L",36,08K)
- Create index on composite key (eAge,eSalary):
  - CREATE INDEX MyIndex ON Employee(eAge,eSalary)

Note

- How to order composite values, e.g. (eAge, eSalary)

- First, order by eAge
- In case of a tie, order by eSalary

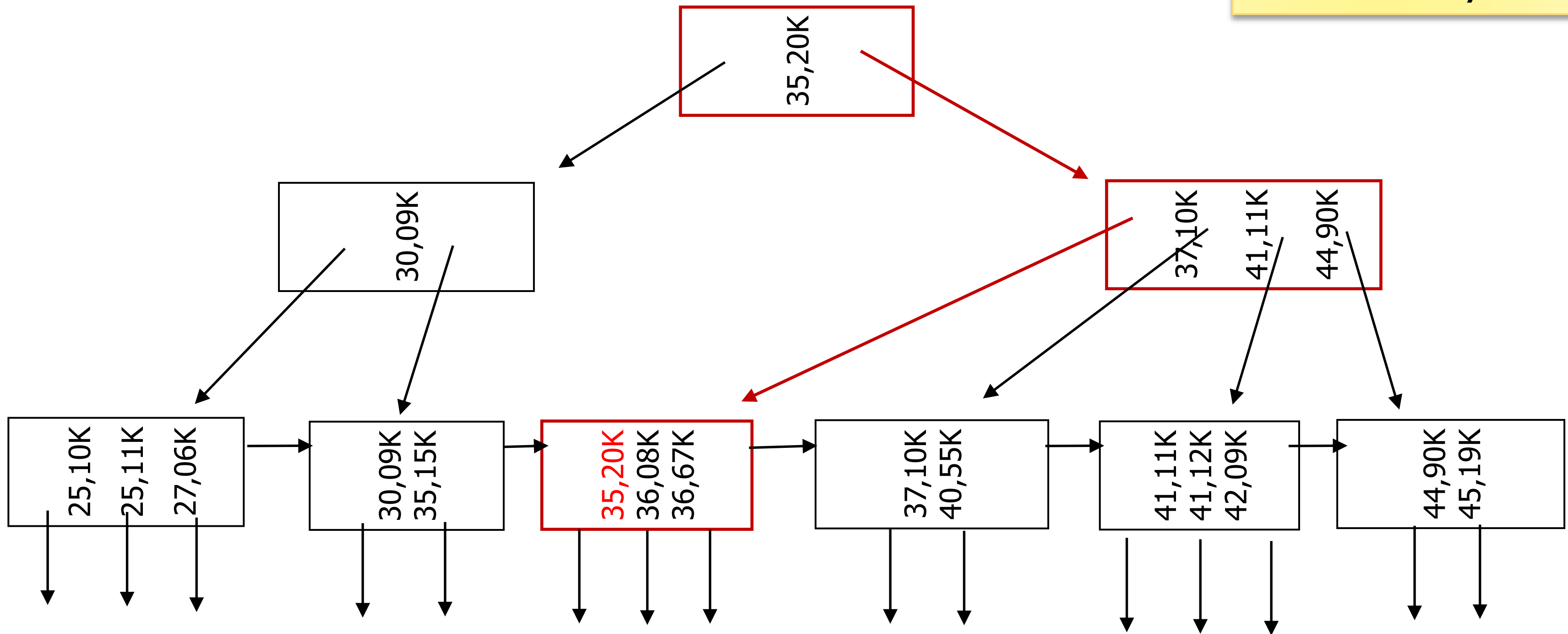
Υλοποίηση με B+-tree



Αναζήτηση:Υπάλληλοι 35,20K?

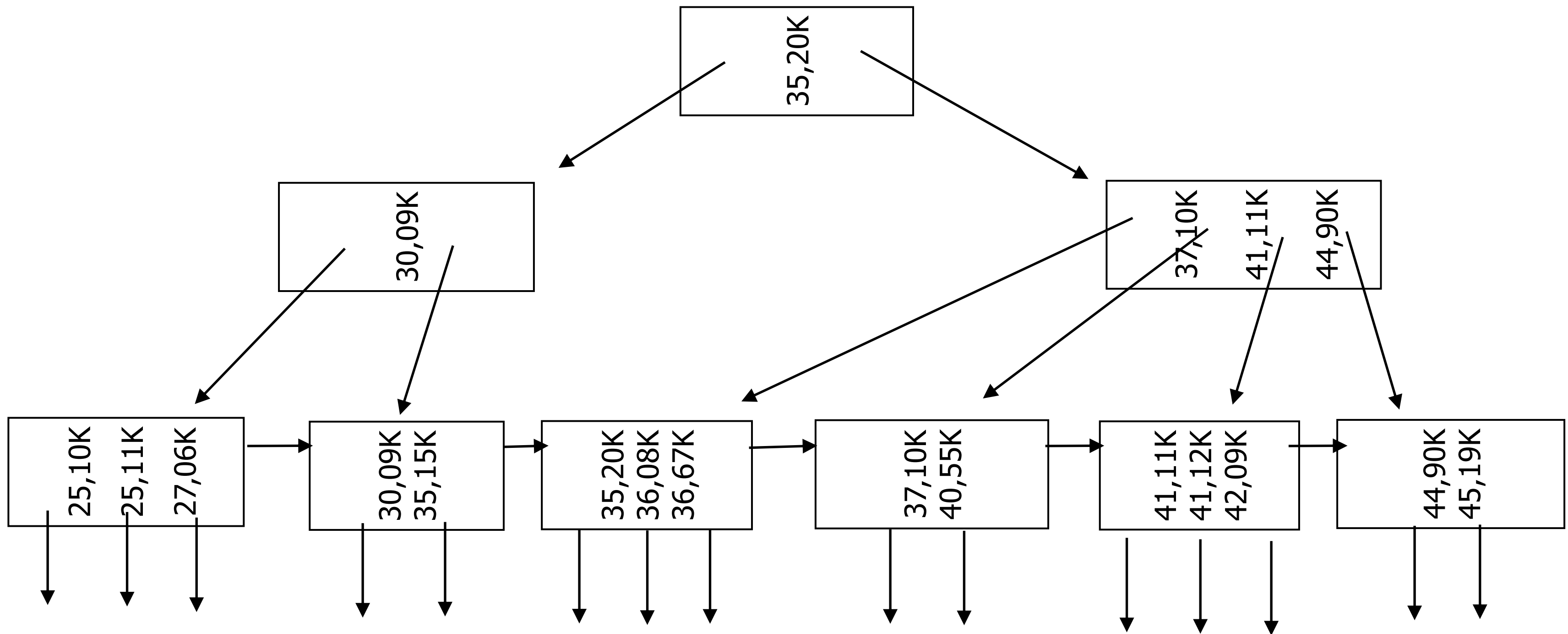
Πόσες σελίδες θα διαβάσω?

SELECT eName
FROM Employee
WHERE eAge=35
AND eSalary=20K



Αναζήτηση:Υπάλληλοι από 30-35?

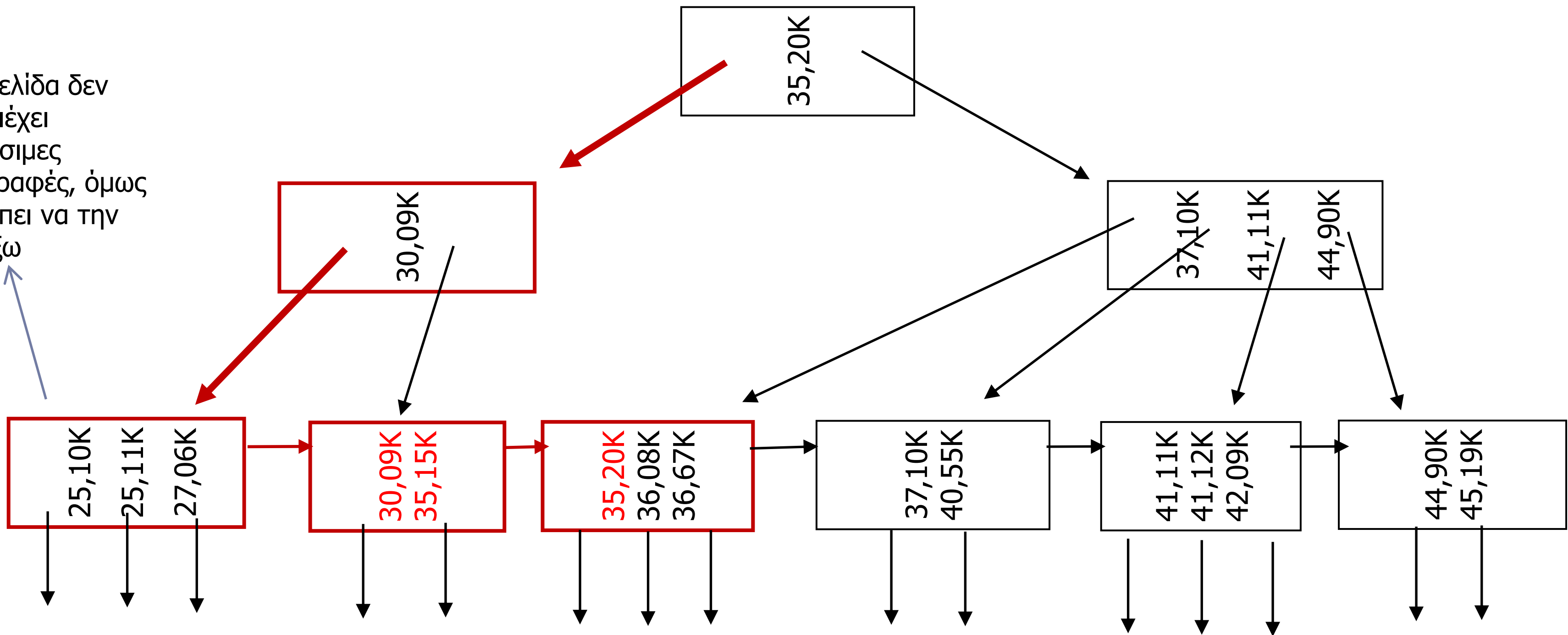
Πόσες σελίδες θα διαβάσω?



Αναζήτηση:Υπάλληλοι από 30-35?

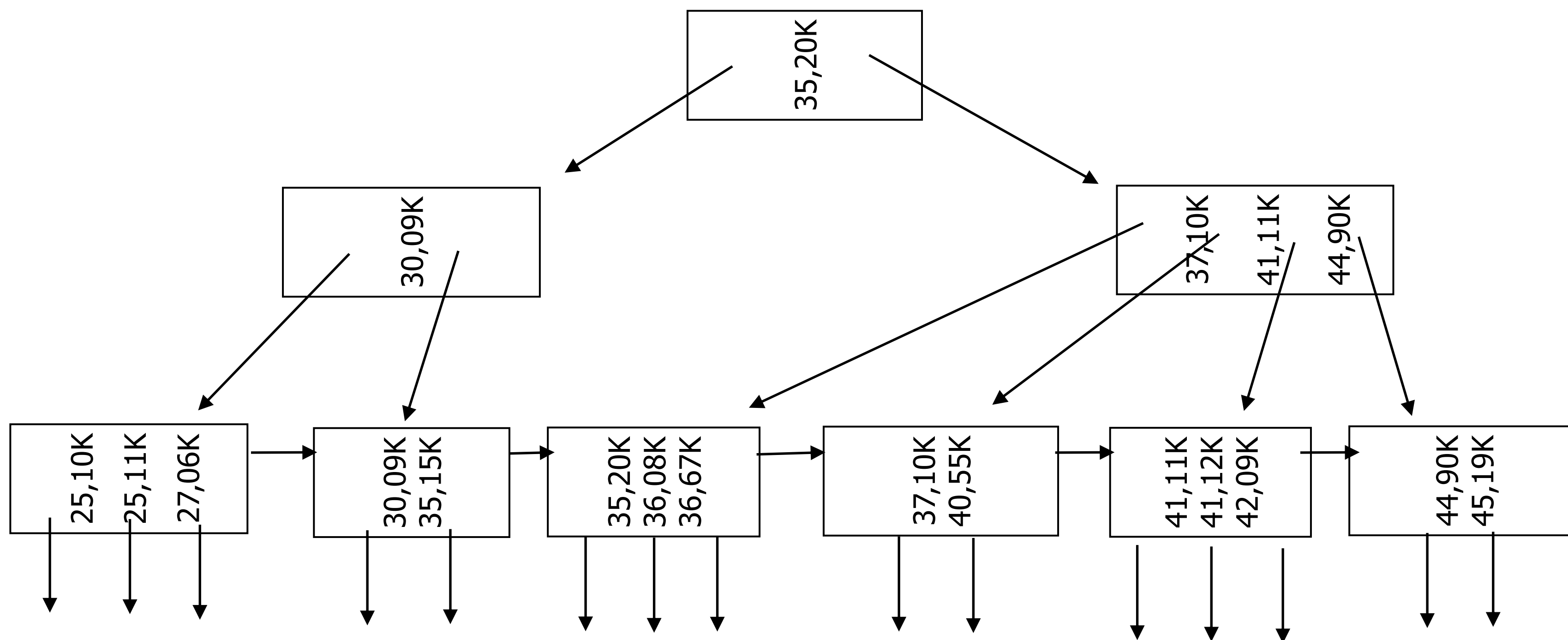
Αναζητώ από (30,00K) έως (35,99K)

Root (Ρίζα)



Αναζήτηση: Υπάλληλοι από 30-35 και 10-20K?

Πόσες σελίδες θα διαβάσω?



Αναζήτηση: Υπάλληλοι από 30-35 και 10-20Κ?

Αναζητώ από (30,10Κ) έως (35,20Κ)

Root (Ρίζα)

35,20K

30,09K

37,10K 41,11K 44,90K

25,10K 25,11K 27,06K

30,09K 35,15K

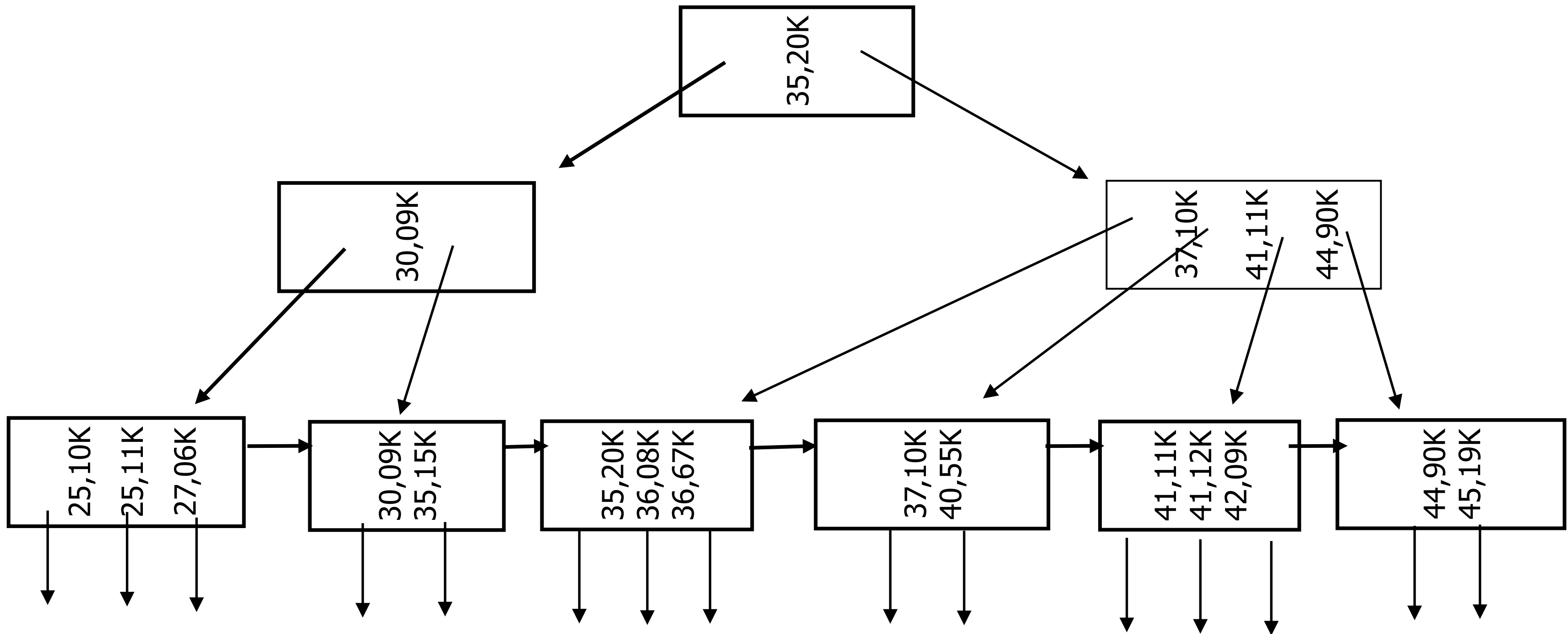
35,20K 36,08K 36,67K

37,10K 40,55K

41,11K 41,12K 42,09K

44,90K 45,19K

Αναζήτηση:Υπάλληλοι από 20K-25K?



Αναζήτηση:Υπάλληλοι από 20K-25K?

Πόσες σελίδες θα διαβάσω?
Είναι χρήσιμο το ευρετήριο?
Πολύ-λίγο-καθόλου?