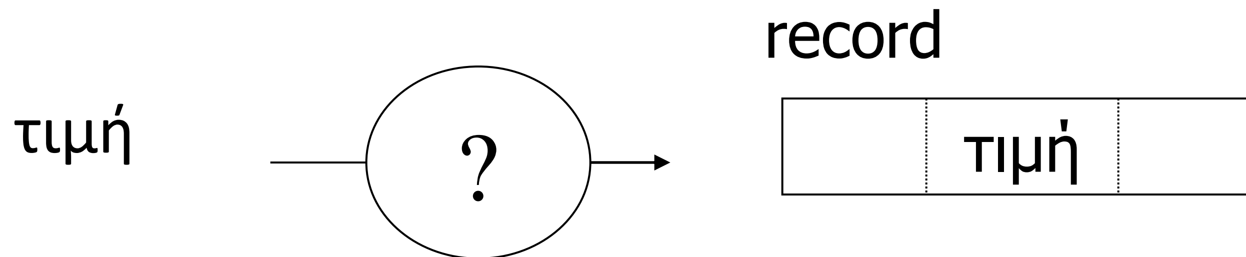
Ευρετήρια

Γιάννης Κωτίδης

Γενική εικόνα

Ευρετήρια & Κατακερματισμός



Πχ

SELECT * FROM R WHERE a=11
SELECT * FROM R WHERE 0<= b and b<42

Παράδειγμα

- Έστω η σχέση
  - Employee (ID, Name, Dept, …)
  - 10 M tuples (εγγραφές)
- Επερώτηση:

```
SELECT *
FROM Employee
WHERE Name = “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 Name = “Bob”
    - ...για κάθε εγγραφή έλεγξε την τιμή του γνωρίσματος, επέστρεψε όσα ικανοποιούν τη συνθήκη

Εκτέλεση χωρίς ευρετήριο

- Χρόνος εκτέλεσης:
  - Μέγεθος εγγραφής = 100 bytes
  - Μέγεθος σχέσης= 10 M x 100 = 1 GB
  - Ταχύτητα ανάγνωσης δίσκου (Sequential Read) = 40 MB/s
  - Time @ 40 MB/s = 1024/40 = 25.6 secs

Εναλλακτικό Πλάνο Εκτέλεσης

- Υποθέτω ότι οι εγγραφές της σχέσης είναι ταξινομημένες με βάση το γνώρισμα Name
- Ιδέα: ας κάνουμε **δυαδική αναζήτηση**
  - Μέγεθος σελίδας: 1024 bytes
  - Εγγραφές ανά σελίδα: 1024 /100=10
  - Αριθμός σελίδων: 10 M / 10= 1 M σελίδες
  - Σελίδες που θα διαβάσω : $\log_2(1M)=20$
  - 1 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);
- Συνθήκες σε διαφορετικά γνωρίσματα:

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

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
    - Κοντινότεροι γείτονες (Nearest Neighbor Queries)
    - Αναζήτηση μέσω keywords

Ευρετήρια & ΣΔΒΔ

- Χρειαζόμαστε δομές κατάλληλες για αποθήκευση στο δίσκο
- Ανάγκη για ενημέρωση όταν αλλάζει η σχέση
  - Κόστος αν έχουμε συχνά ενημερώσεις (updates)
  - 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 κάνει πιο αποδοτική την 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)

Είδη Ευρετηρίου

- **Πρωτεύον ευρετήριο** (primary index) όταν εφαρμόζεται στο κλειδί διάταξης του αρχείου
- **Ευρετήριο συστάδων** (clustering index) όταν ορίζεται στο πεδίο διάταξης το οποίο όμως δεν είναι κλειδί
- **Δευτερεύον ευρετήριο** (secondary index) όταν ορίζεται σε πεδίο διαφορετικό από το πεδίο διάταξης

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
  - Αυτό δεν ισχύει πάντα. Η θέση της εγγραφής μπορεί να προκύπτει έμμεσα από τη πληροφορία που μας δίνει το ευρετήριο
  - Σε κάποιες περιπτώσεις το ευρετήριο είναι ενσωματωμένο στη δομή που αποθηκεύει τις εγγραφές της σχέσης στο δίσκο

Θέματα βελτιστοποίησης

- Ας θεωρήσουμε την επερώτηση:

```
SELECT *
FROM Employee
WHERE Name = “Bob” AND Sal>700
```

- Έχω ευρετήριο στο Name και ένα δεύτερο στο Sal
  - Ποιο από τα δύο να χρησιμοποιήσω;
  - Μπορώ και τα δύο;

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	

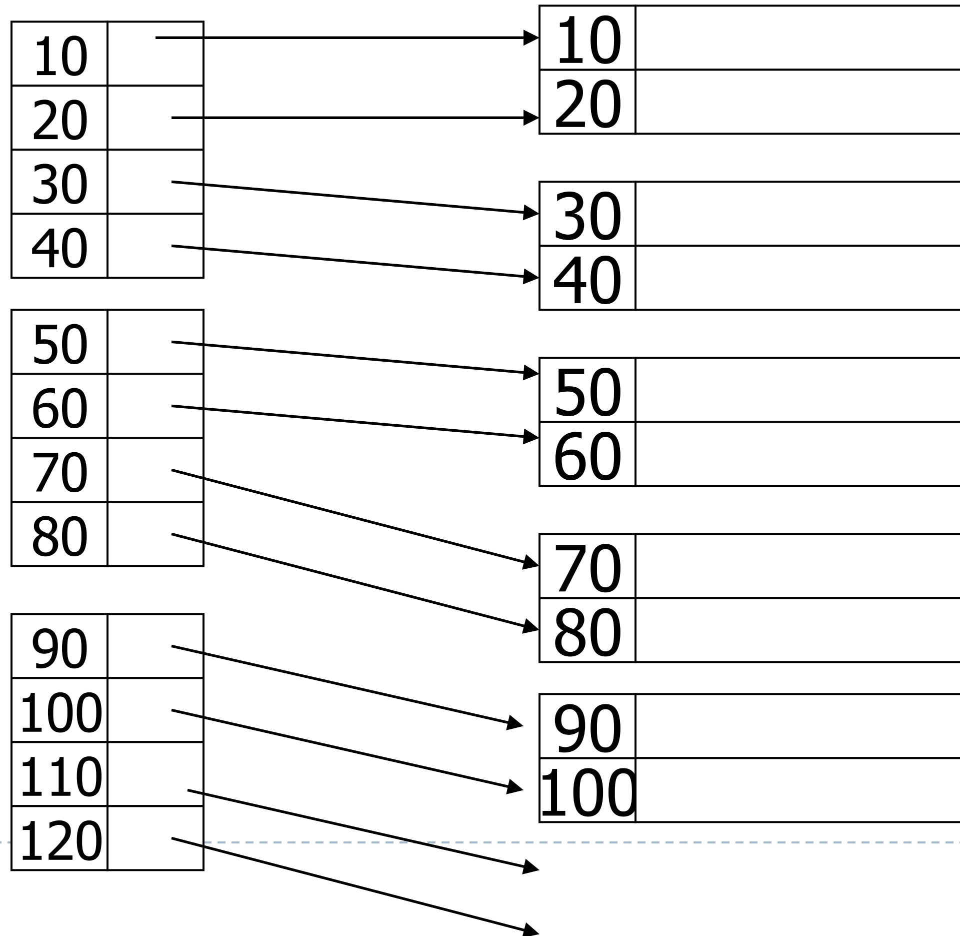
Αυτό μας κάνει?

Χρήσιμο εφόσον το ευρετήριο είναι σημαντικά μικρότερο από τη σχέση

Πυκνό ευρετήριο (Dense Index):

Εμφανίζονται όλες οι τιμές του γνωρίσματος που υπάρχουν στη σχέση

Πως θα το χρησιμοποιήσω σε μία επερώτηση?



Παράδειγμα

- Διατεταγμένη σχέση R με 30000 εγγραφές
- Μέγεθος σελίδας 1KB, μέγεθος εγγραφής 100 bytes
- Δίσκος: 20msec για random I/O (μετράω μόνο το seek time), 30MB/sec για sequential read

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

Δυαδική αναζήτηση χωρίς ευρετήριο

- Κάθε σελίδα περιέχει 1024/100 δηλαδή 10 εγγραφές
- Η σχέση καταλαμβάνει $B(R)=\lceil 30000/10 \rceil=3000$ σελίδες
- Η δυαδική αναζήτηση θα χρειαστεί $\lceil \log_2(3000) \rceil = 12$ I/O
- Έστω 20msec για κάθε random I/O
  - Συνολικά 0,24secs

Αναζήτηση με ευρετήριο

- Έστω ότι κάθε εγγραφή στο ευρετήριο καταλαμβάνει 8 bytes για την τιμή και 4 bytes για τον pointer
- Κάθε σελίδα του ευρετηρίου περιέχει $\lceil 1024/12 \rceil$=85 εγγραφές
- Το ευρετήριο καταλαμβάνει $\lceil 30000/85 \rceil$=36 σελίδες
- Η δυαδική αναζήτηση στο ευρετήριο θα χρειαστεί $\lceil \log_2(36) \rceil$ = 6 I/O
- Χρειάζομαι και 1 I/O όταν ακολουθώ τον pointer προς την εγγραφή άρα συνολικά 7 I/O αντί για 12
  - Χρόνος = 7*0,02=0,14 secs

Ερώτηση: αν δεν κάνω δυαδική αναζήτηση?

- Ας υπολογίσουμε τον μέσο χρόνο…
- Χωρίς ευρετήριο
  - ½*3000 σελίδες * 1KB/σελίδα = (περίπου) 1.5 MB
  - 1,5 MB σε 0,05sec @ 30MB/sec
  - Προσθέτω 1 seek για την πρώτη σελίδα της σχέσης (+0,02secs)
  - Σύνολο 0,07secs

Ευρετήριο (χωρίς δυαδική αναζήτηση)

- Διαβάζω (κατά μέσο όρο) το μισό ευρετήριο
  - ½ * 36 σελίδες = 18 σελίδες = 18KB
  - 18KB σε 0,006secs @ 30MB/sec
  - I seek για την αρχή του ευρετηρίου και 1 seek για να βρω τη σελίδα της σχέσης που περιέχει την εγγραφή
  - Σύνολο = 0,046secs

Σύνοψη

	Δυαδική Αναζήτηση	**Σειριακή ανάγνωση (μέσος χρόνος)**
Με ευρετήριο	0,14	0,046
Χωρίς ευρετήριο	0,24	0,07

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

Σημείωση

- Οι σελίδες της σχέσης και του ευρετηρίου μπορεί να είναι πραγματικά συνεχόμενες στο δίσκο
  - ...ή συνδεδεμένες (σαν λίστα) μέσω pointers

Αραιό ευρετήριο (sparse index)

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

Sparse Index

10	
30	
50	
70	

90	
110	
130	
150	

170	
190	
210	
230	

Sequential File

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

Ας συνεχίσουμε στο ίδιο μοτίβο (multi-level index)

Sparse 2nd level

Sequential File

10	
90	
170	
250	

330	
410	
490	
570	

10	
30	
50	
70	

90	
110	
130	
150	

170	
190	
210	
230	

10	
20	

30	
40	

50	
60	

70	
80	

90	
100	

Ερώτηση

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

Sparse Index vs. Dense Index

- **Sparse:** Μικρότερο μέγεθος, μπορώ να κρατήσω μεγαλύτερο τμήμα του στη μνήμη
- **Dense:** Μπορώ να απαντήσω αν για η τιμή του γνωρίσματος υπάρχει ή όχι στη σχέση χωρίς να ανατρέξω στη σχέση

Επίσης:

- sparse καλύτερο για εισαγωγές (γιατί?)
- dense απαραίτητο για secondary indexes

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

10	
10	

10	
20	

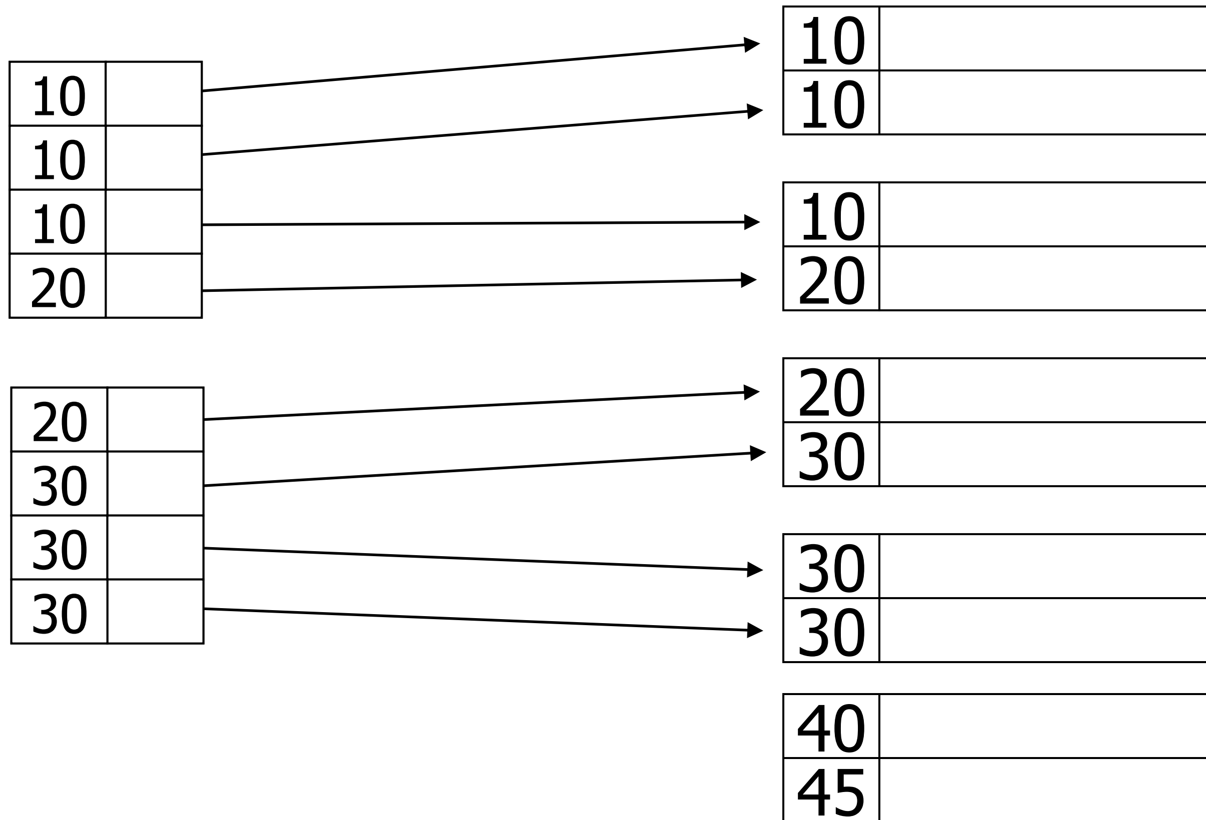
20	
30	

30	
30	

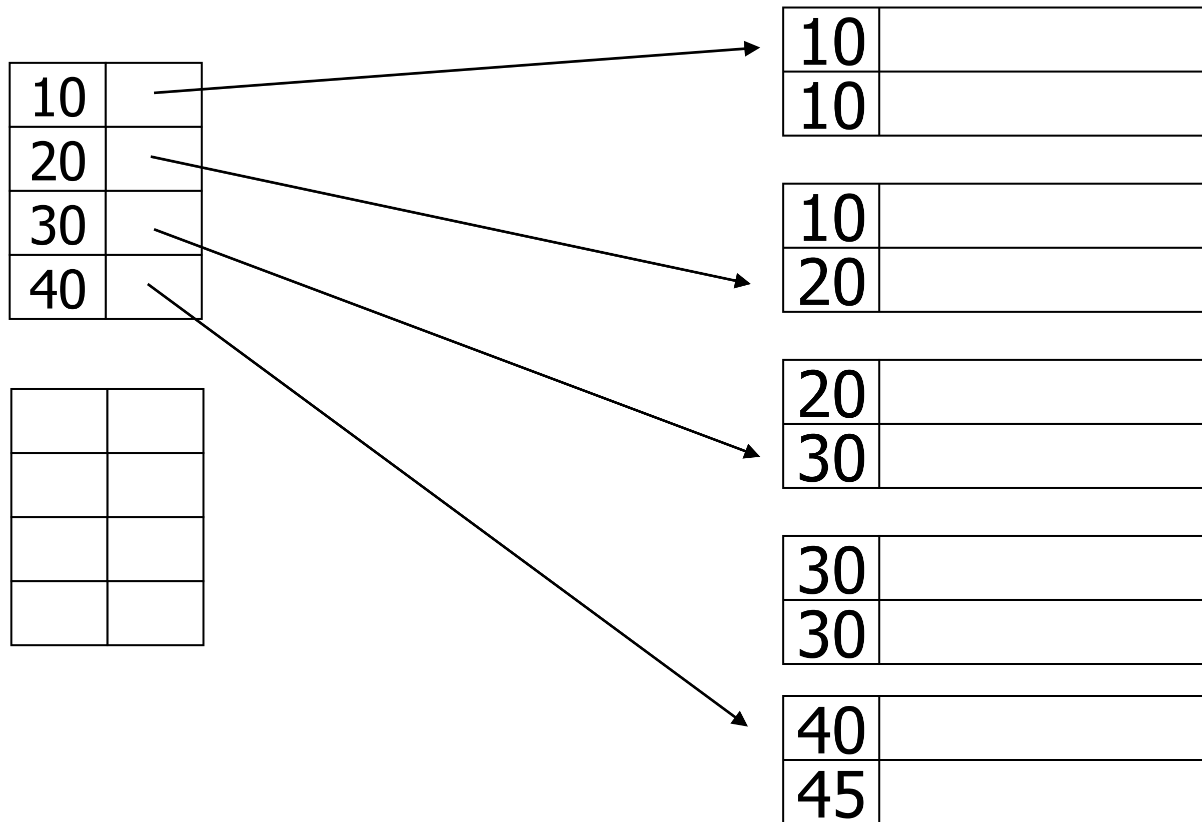
40	
45	

Διπλοεγγραφές

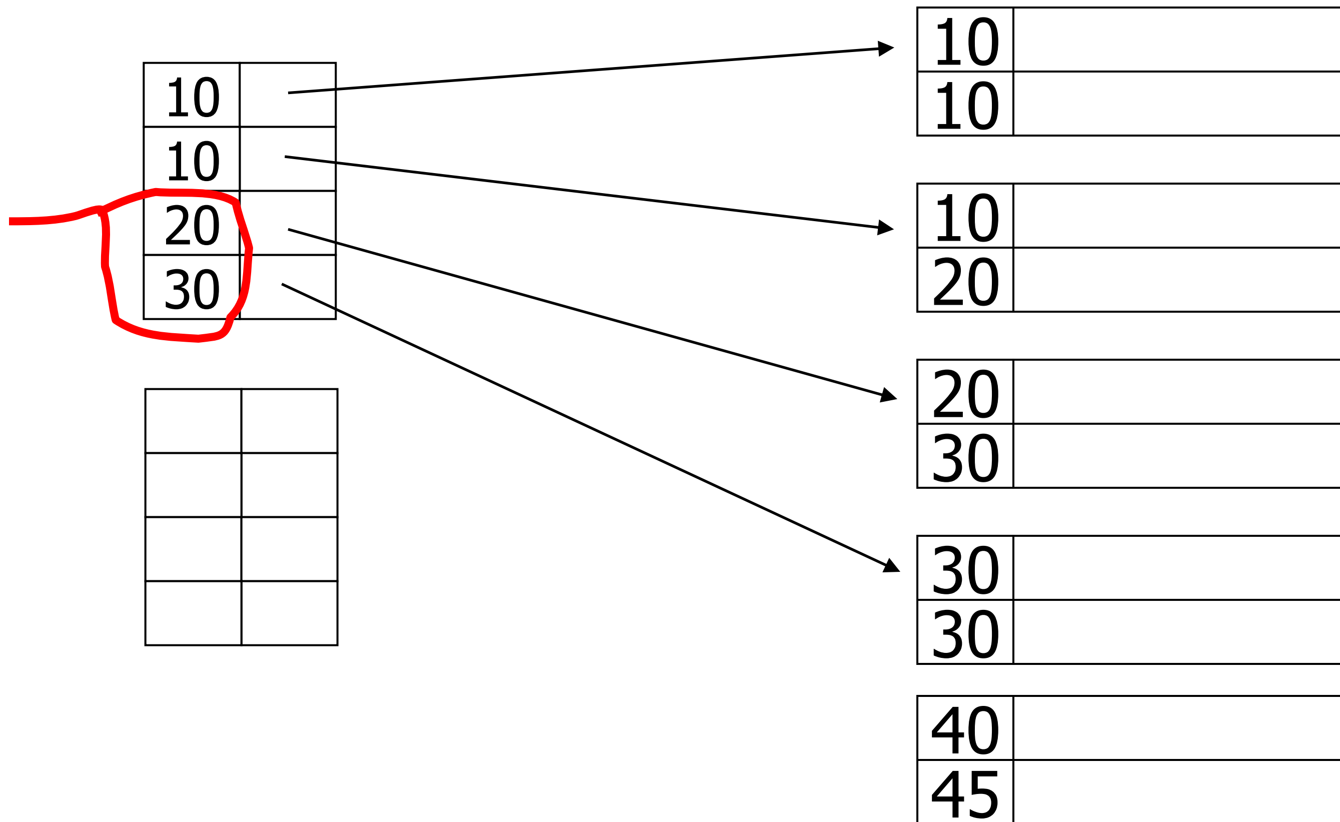
Dense index: μία λύση



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	

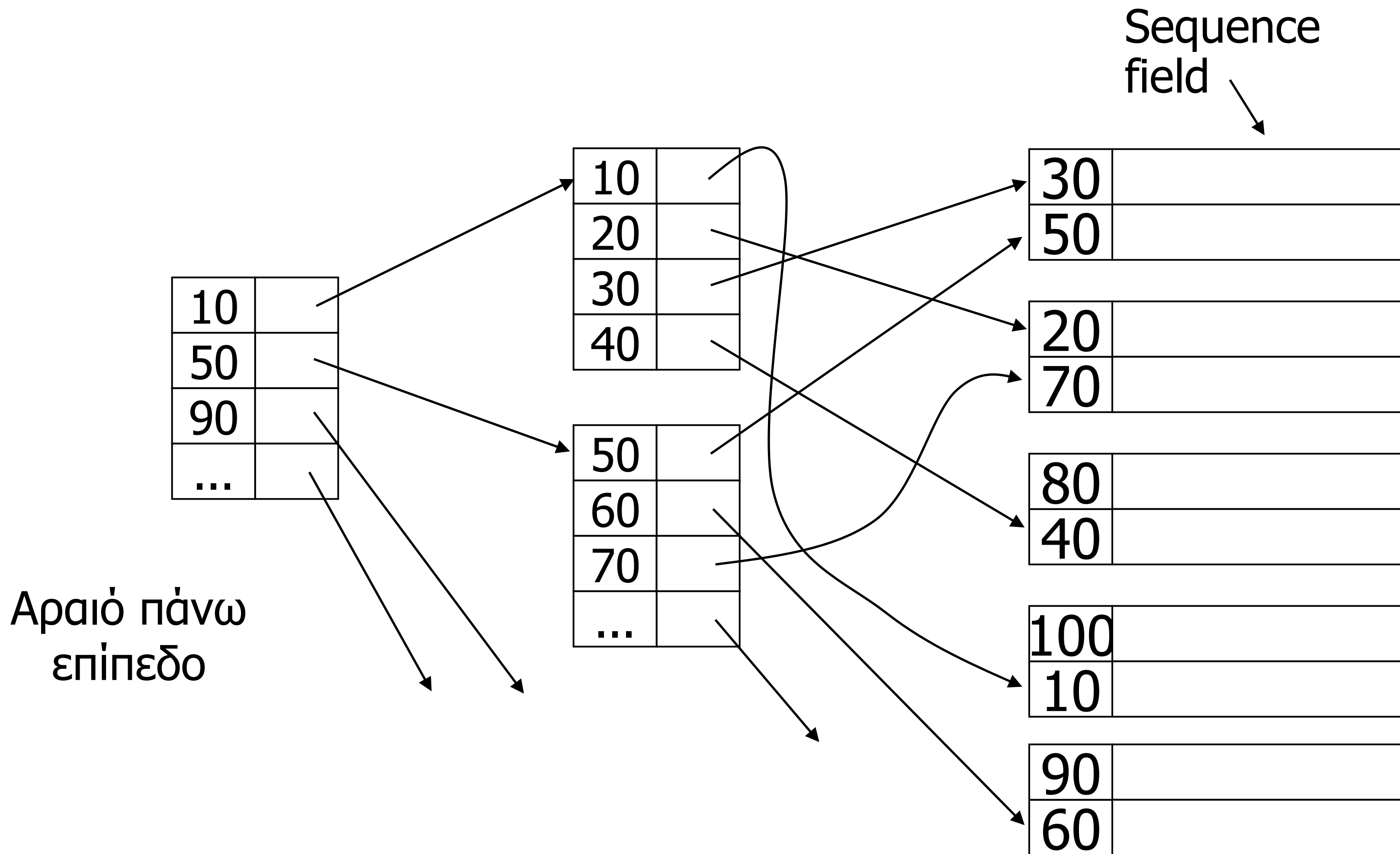
10	
20	

20	
30	

30	
30	

40	
45	

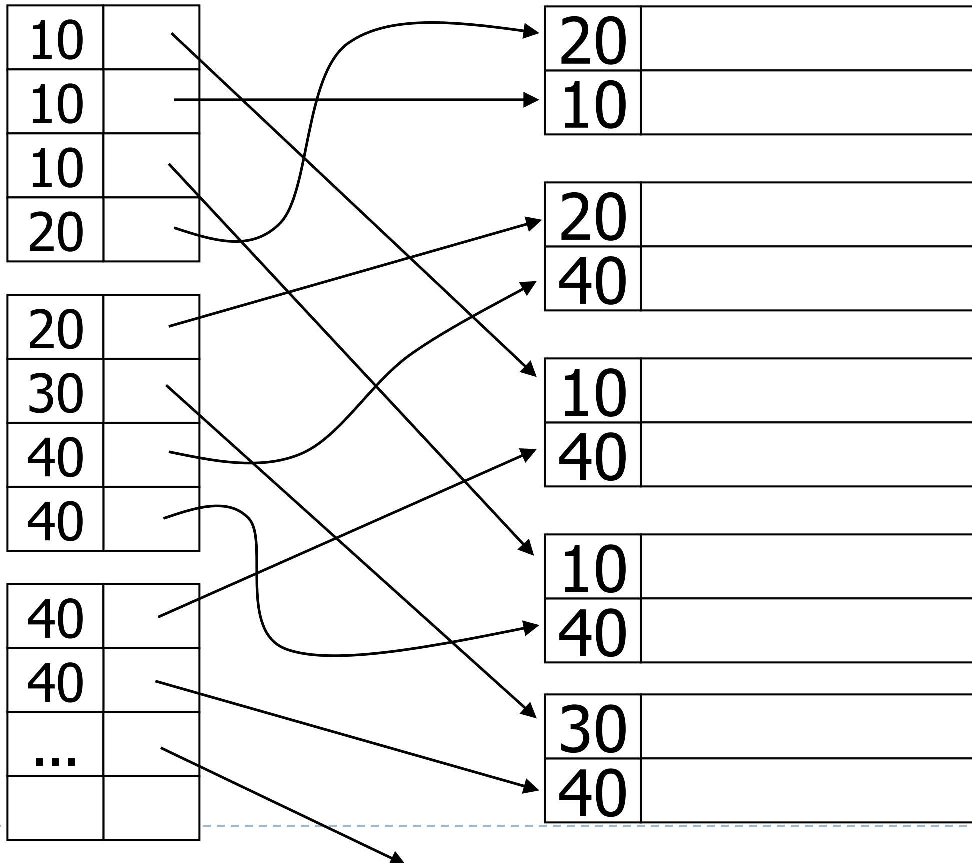
Secondary index
(Κατώτερο επίπεδο πάντα πυκνό)



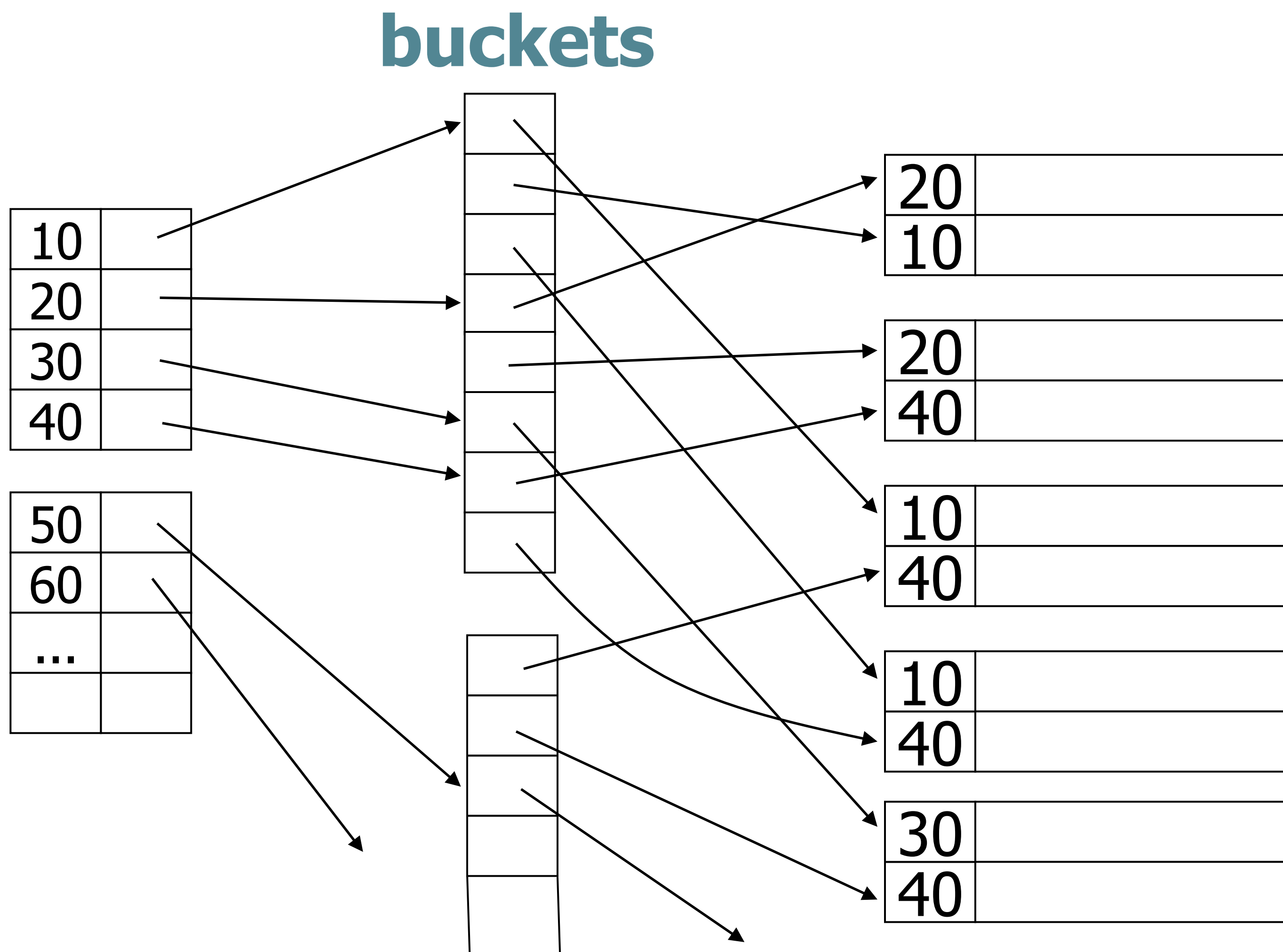
Διπλοεγγραφές & secondary indexes

Μία επιλογή:

Σπατάλη χώρου!



Ιδέα των Buckets (κάδοι)

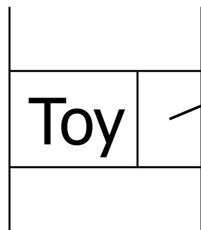


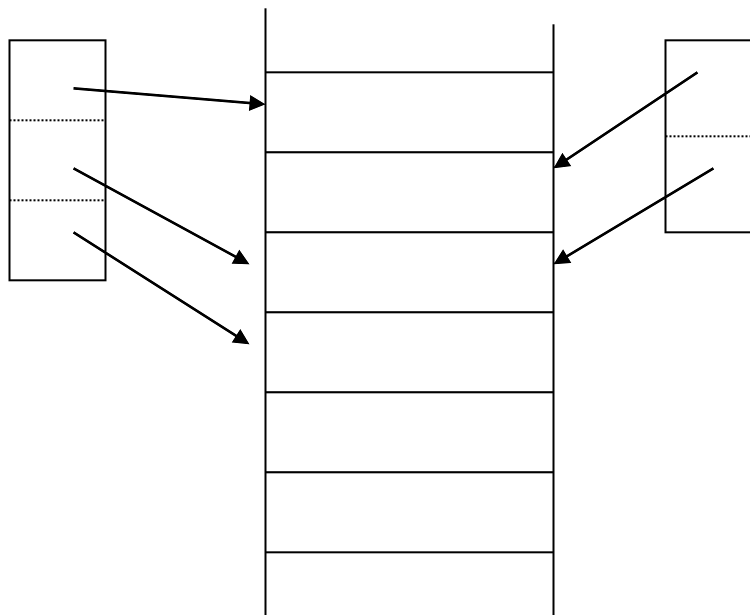
Παράδειγμα

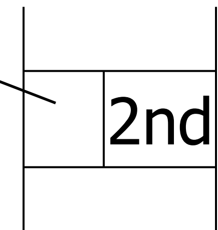
Indexes	Records
Name: primary	EMP (name,dept,floor,...)
Dept: secondary	
Floor: secondary	

Επερώτηση: Βρες υπαλλήλους με

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

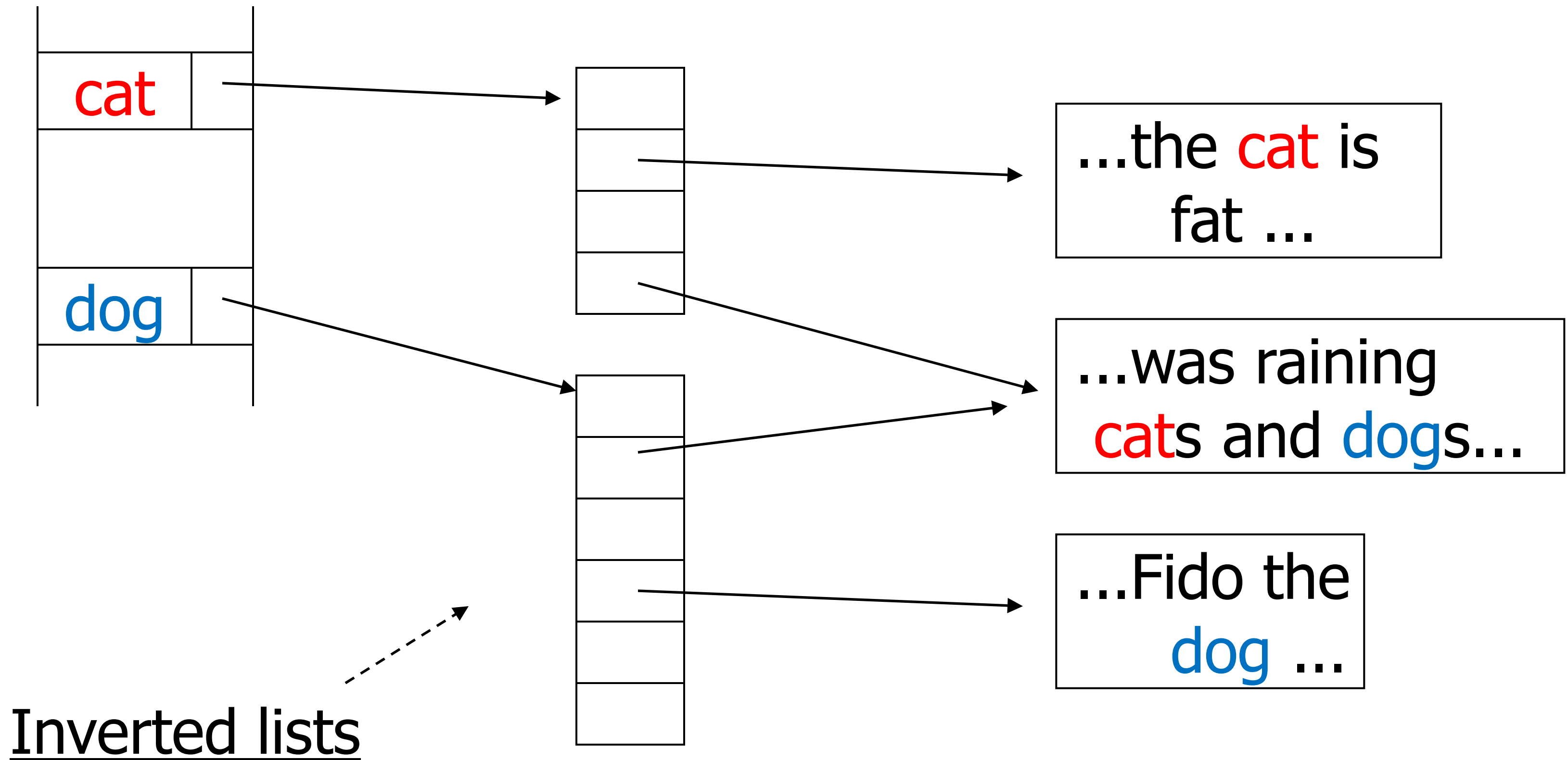






→ Πάρε την τομή του bucket για "Toy" και του bucket για "2nd Floor"

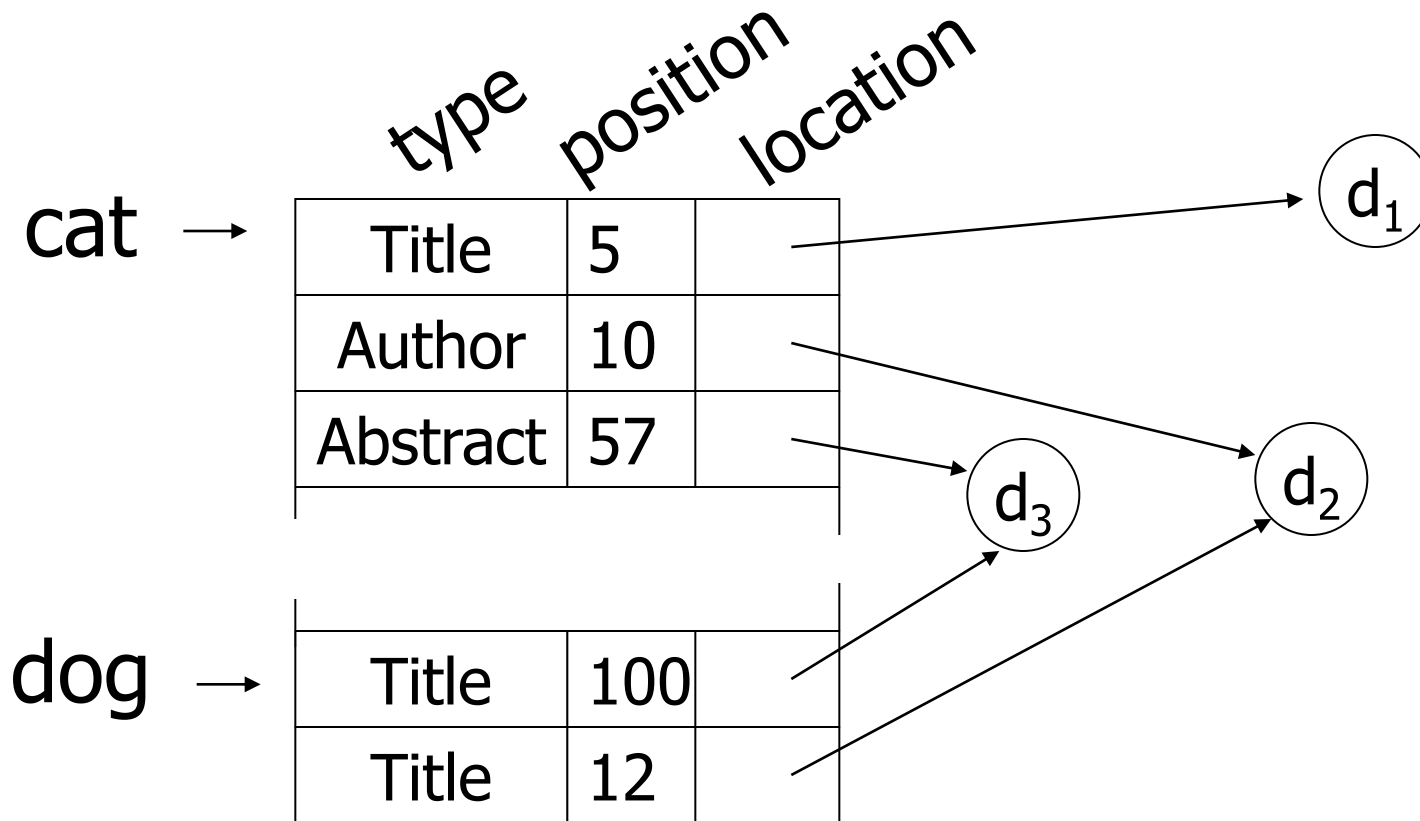
Text Information Retrieval (IR)



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

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

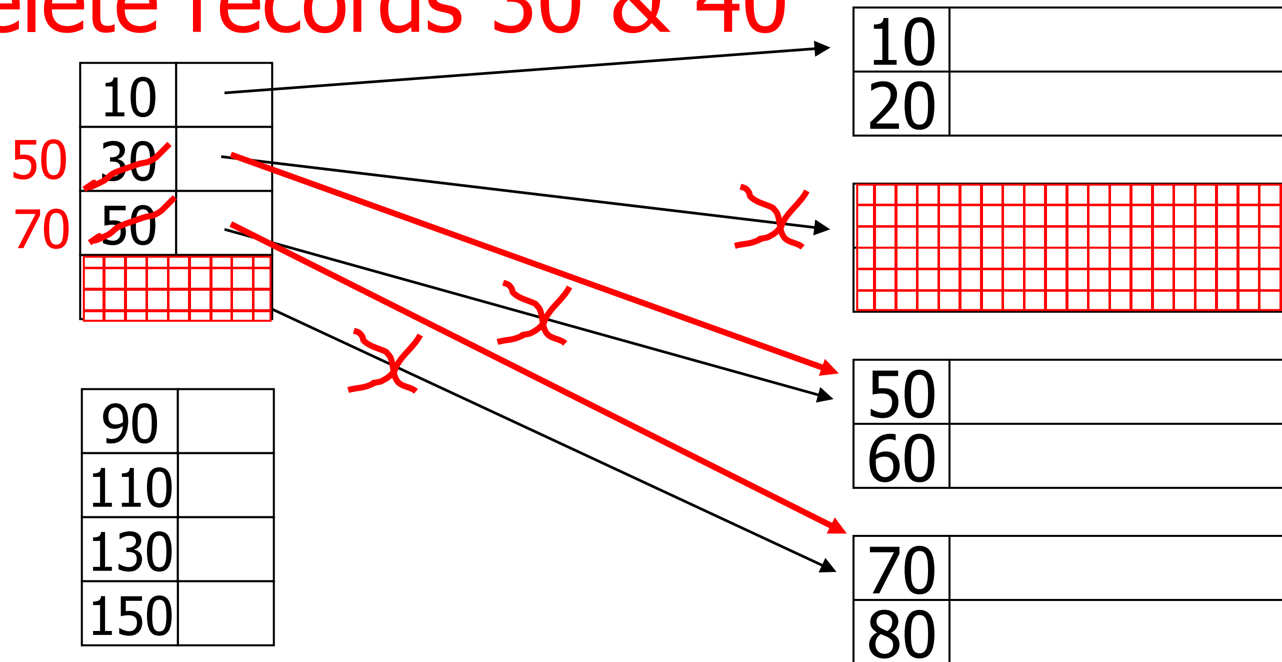
Πλεονεκτήματα:

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

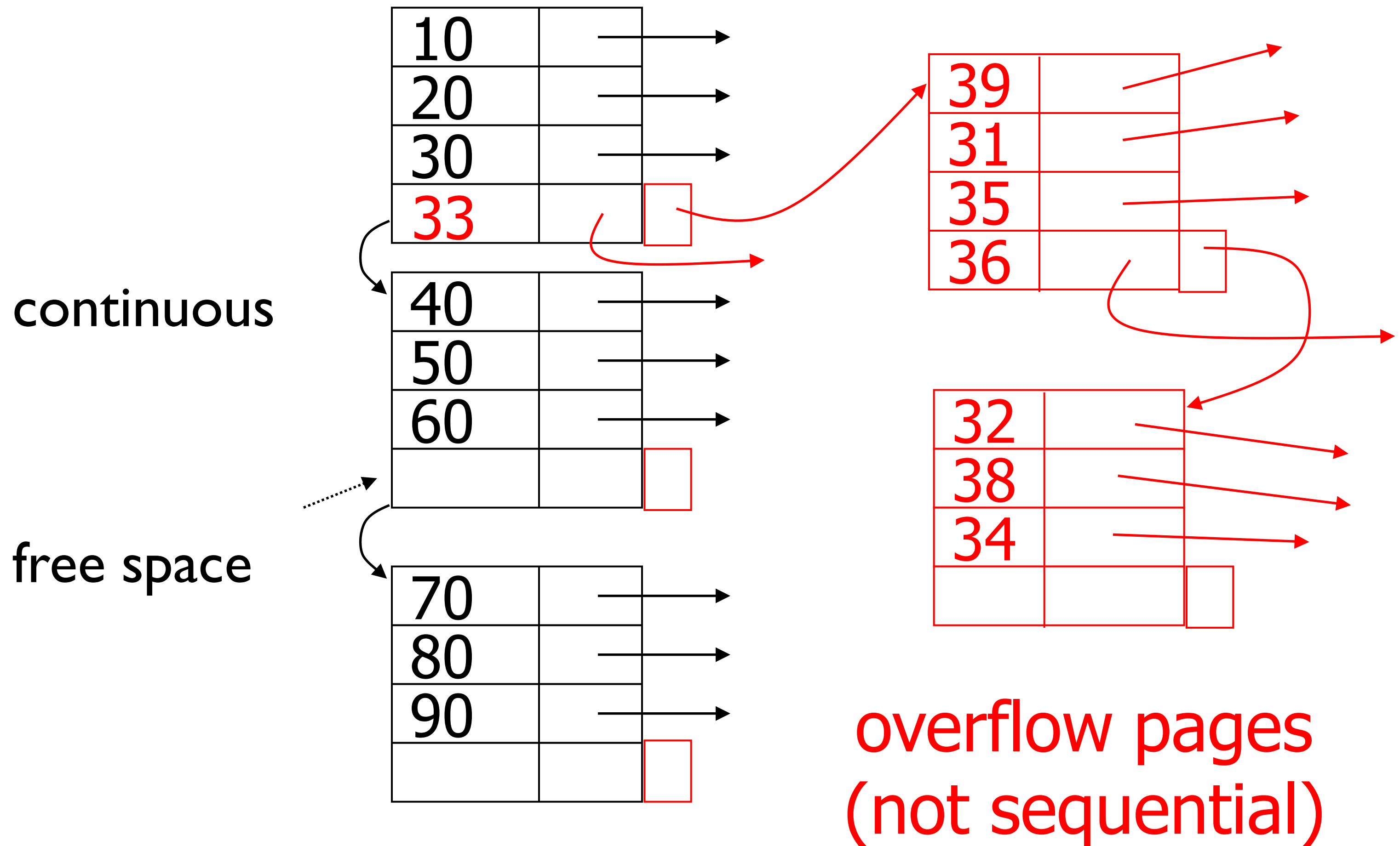
Μειονεκτήματα:

- Επιπλοκές σε inserts/deletes
- Lose sequentiality & balance

Διαγραφές σε αραιό ευρετήριο



Εισαγωγές



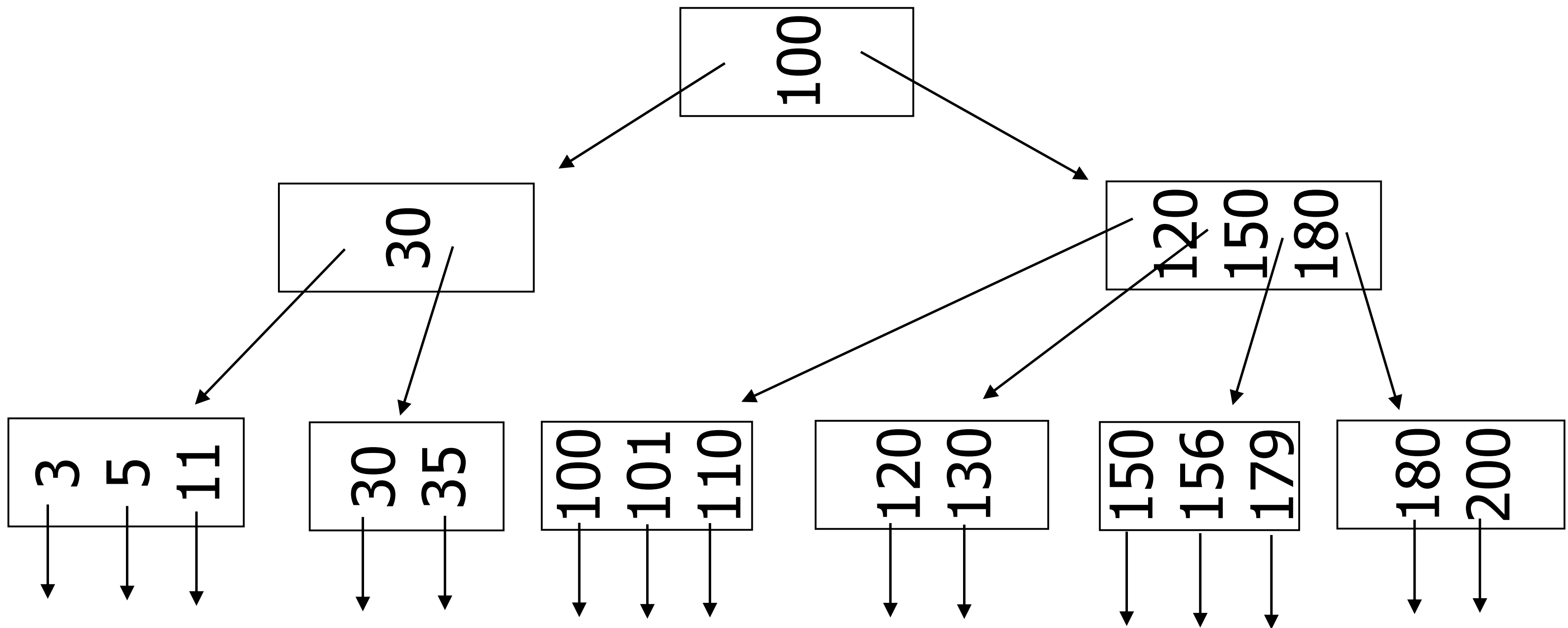
Που είμαστε

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

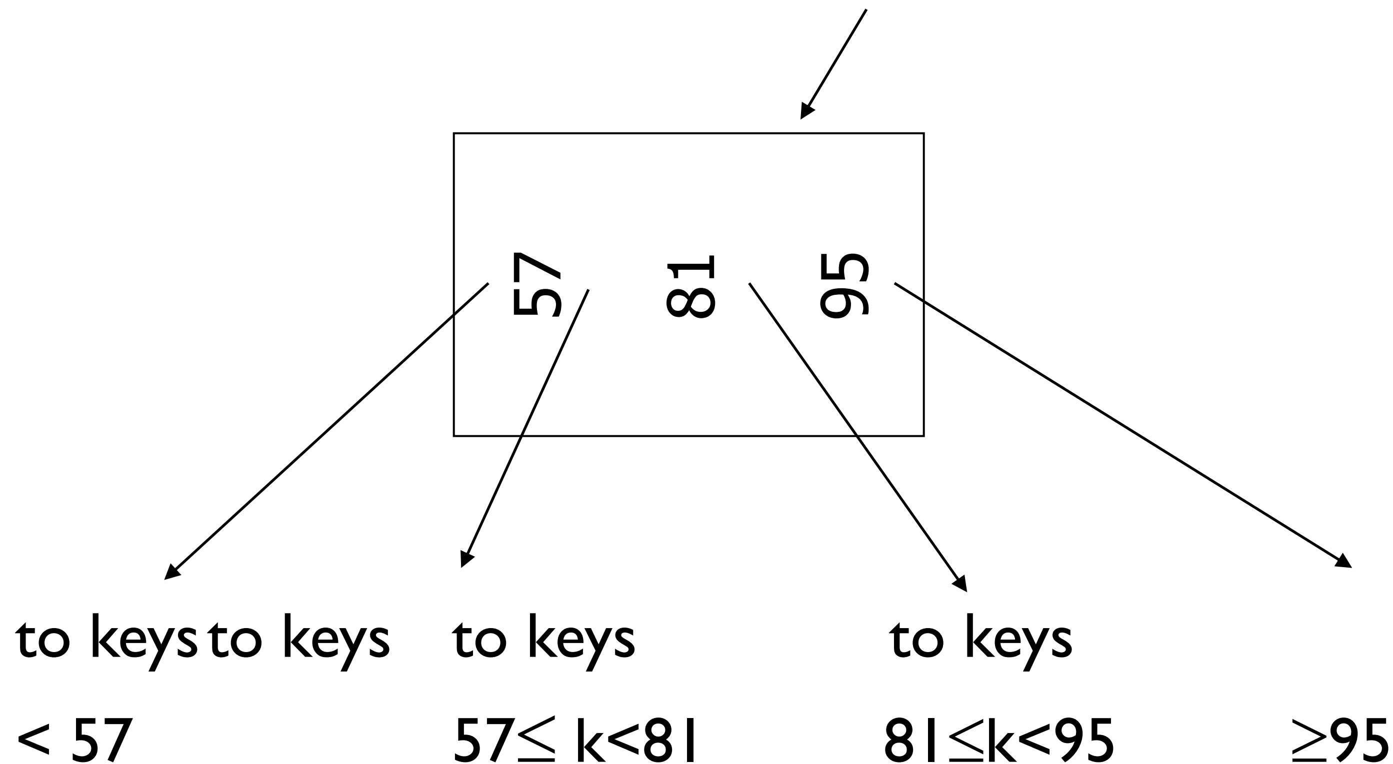
B-trees
(...για την ακρίβεια B^+-trees)

- Μία διαφορετική προσέγγιση
  - Δεν προσπαθούμε το ευρετήριο να είναι σειριακό
  - 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

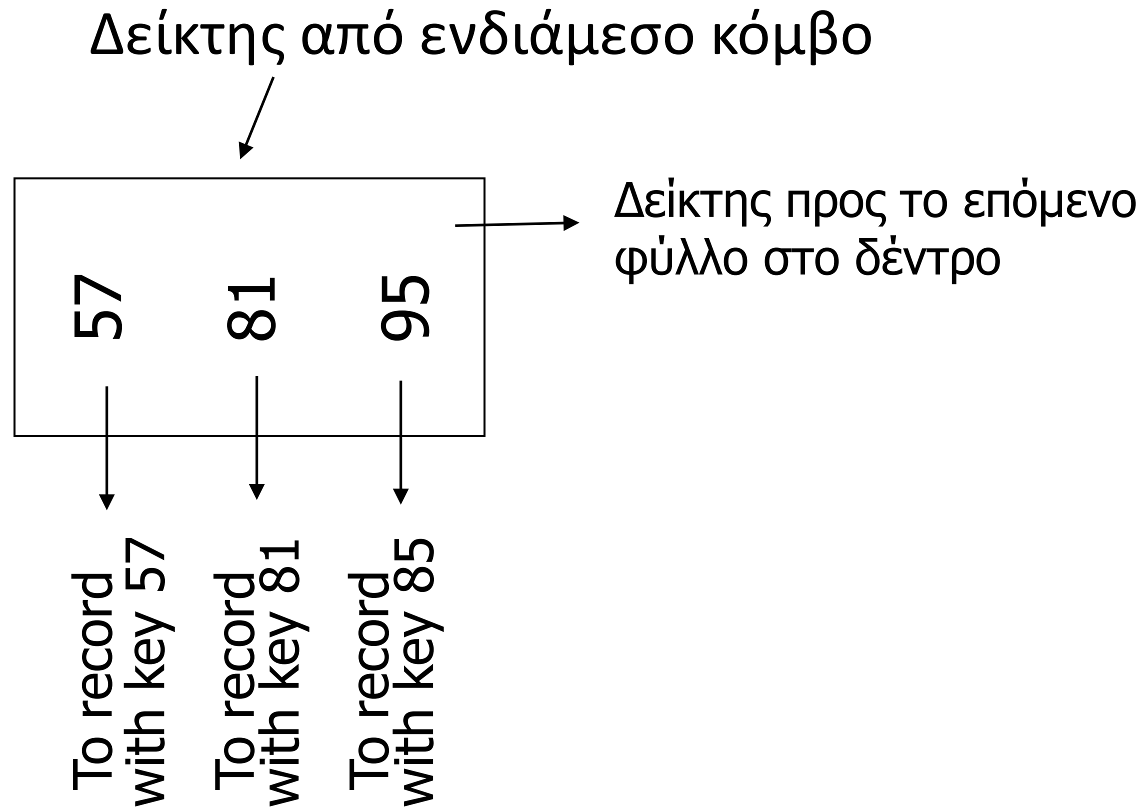
Παράδειγμα: B-Tree



Παράδειγμα ενδιάμεσου κόμβου (non-leaf node)



Παράδειγμα φύλλου (leaf node)



Μέγεθος κόμβων δέντρου

- n+1 pointers
- n keys

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

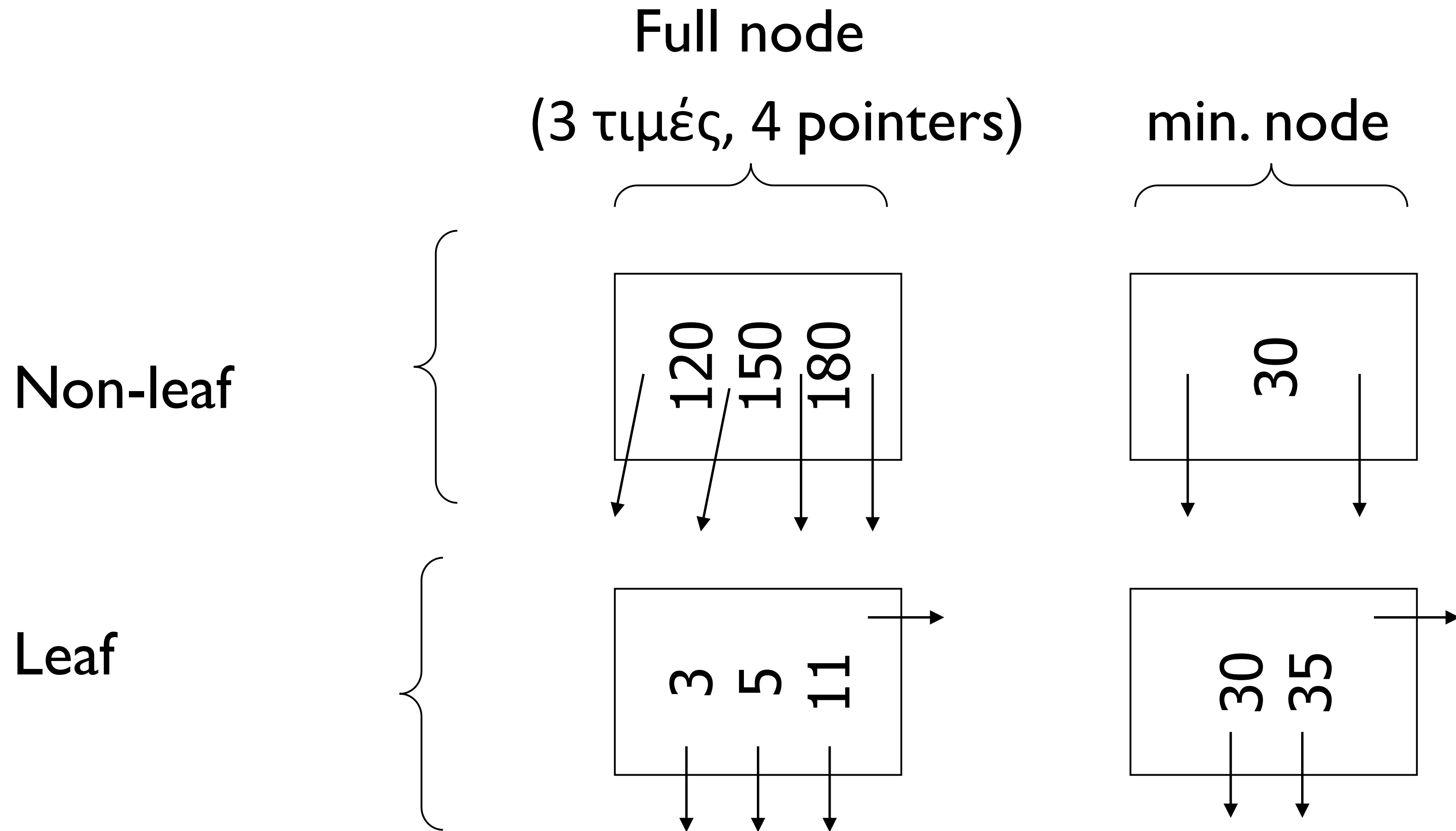
- Ένας κόμβος πρέπει να περιέχει (ανάλογα με τον τύπο του) τουλάχιστον

Non-leaf: $\lceil (n+1)/2 \rceil$ pointers

Leaf: $\lfloor (n+1)/2 \rfloor$ pointers to data

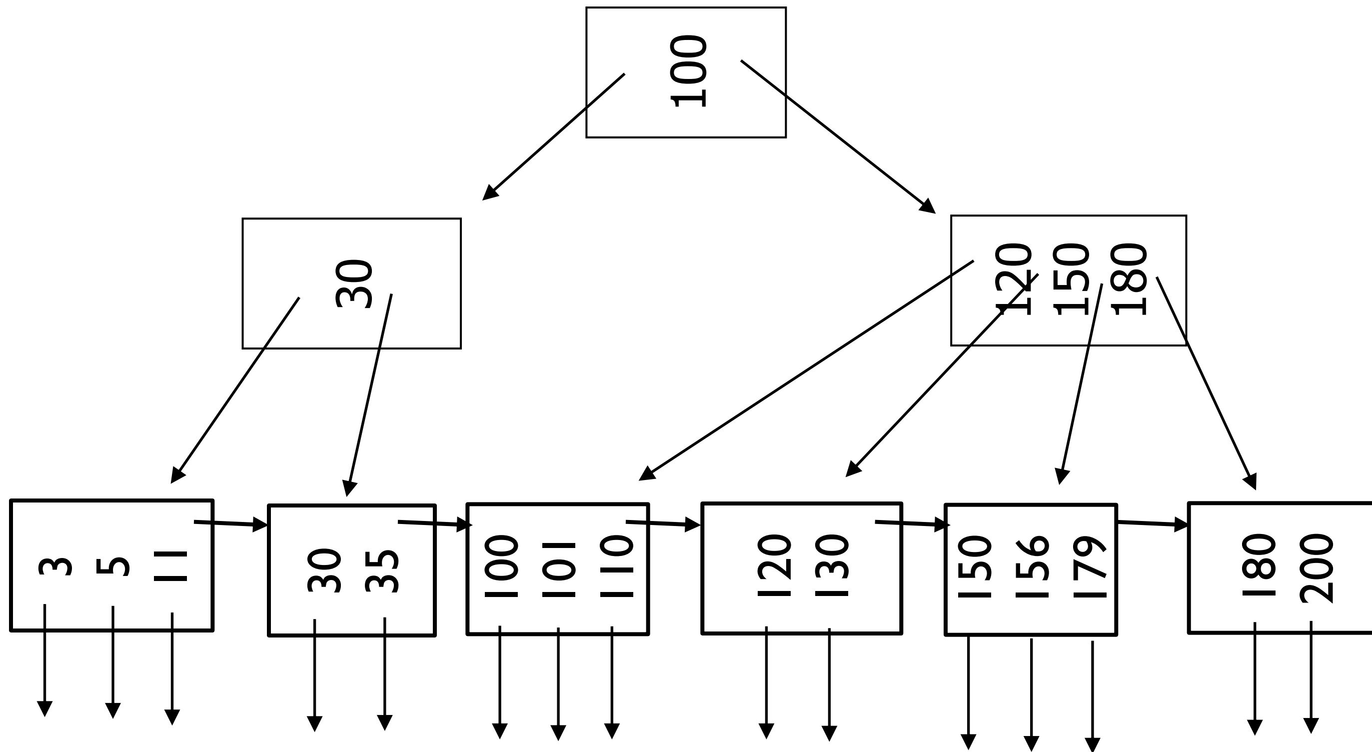
(η ρίζα εξαιρείται από τον κανόνα)

Παράδειγμα: n=3

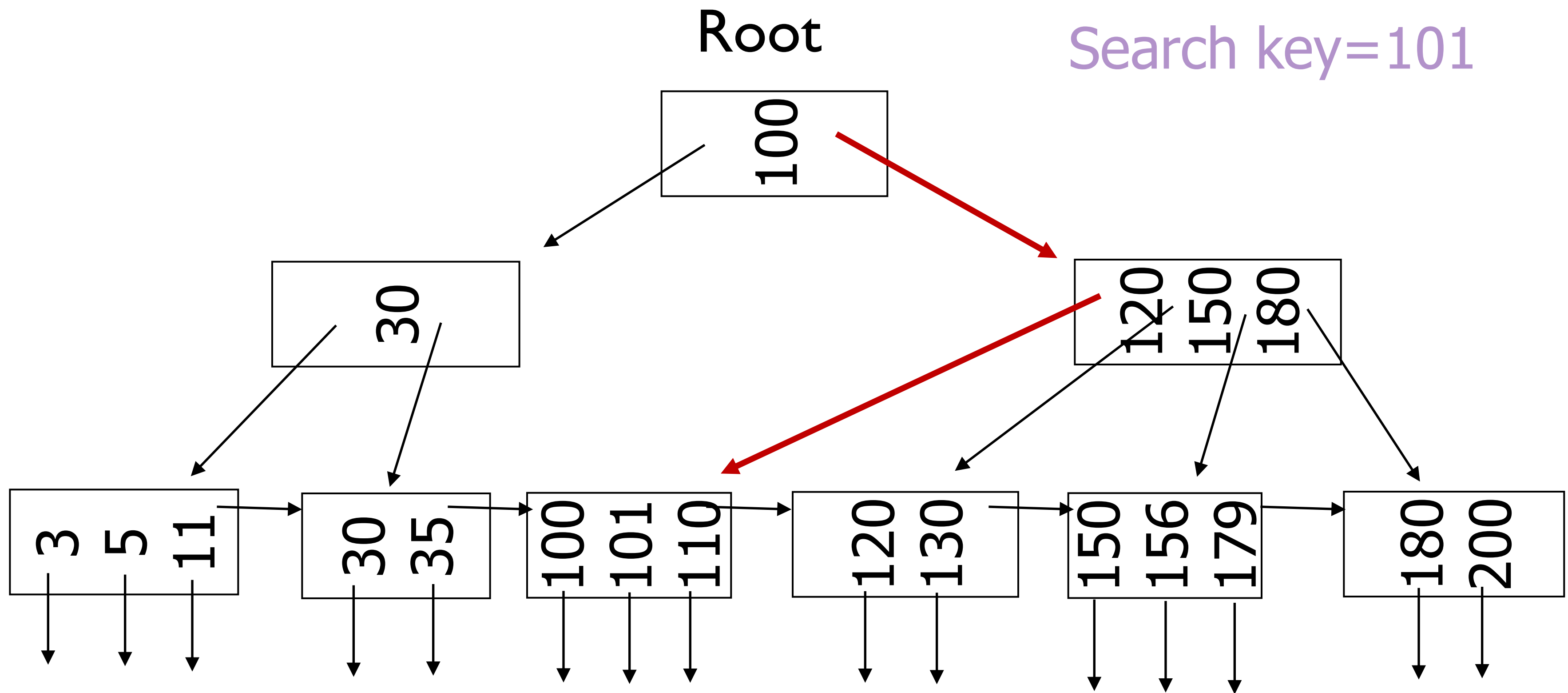


Επίσης

- Όλα τα φύλλα στο ίδιο επίπεδο (balanced tree)

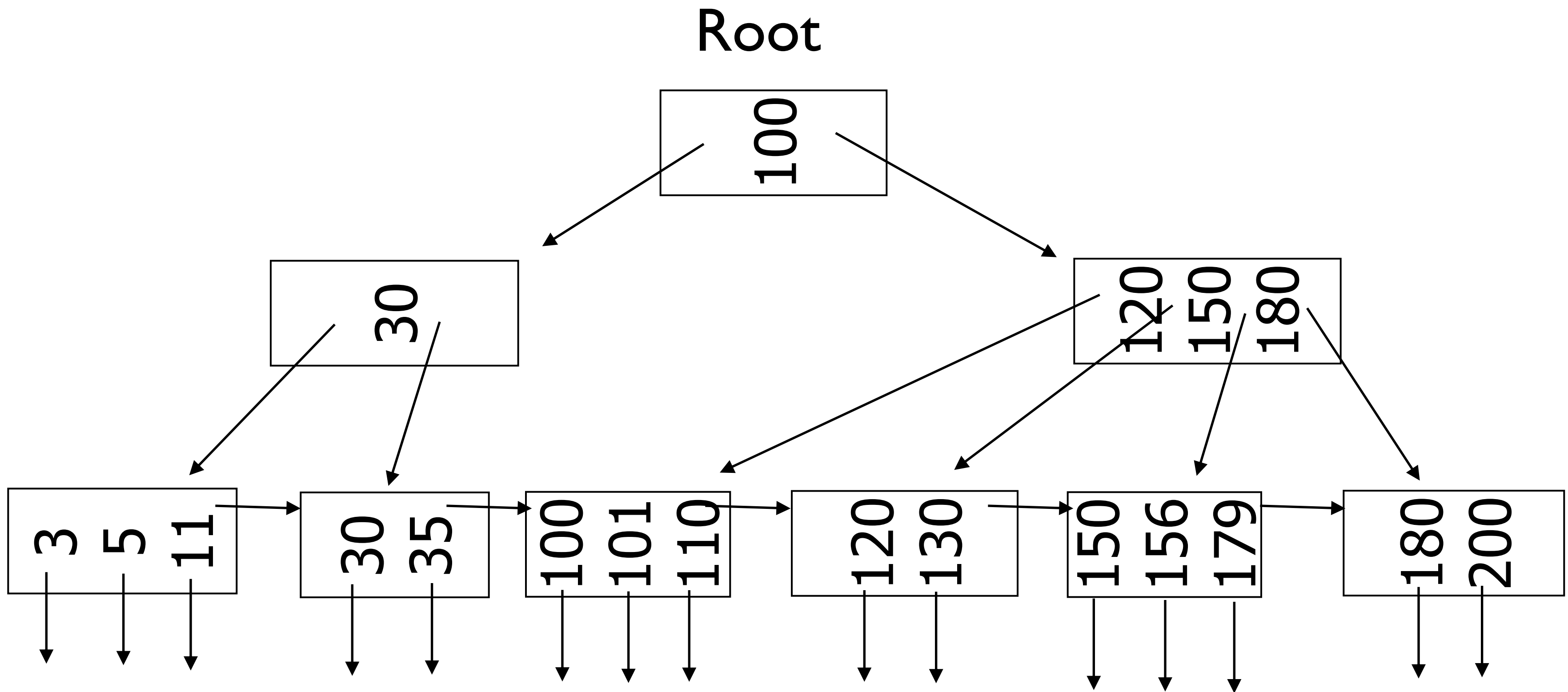


Αναζήτηση (“Point Queries”)



Range Queries

- Search key>=30 ?
- 101<=Search key <=155



B+tree: Εισαγωγή

Εύκολη περίπτωση

(a) Υπάρχει χώρος στο φύλλο

Ποιο δύσκολη: Υπερχείλιση (overflow)

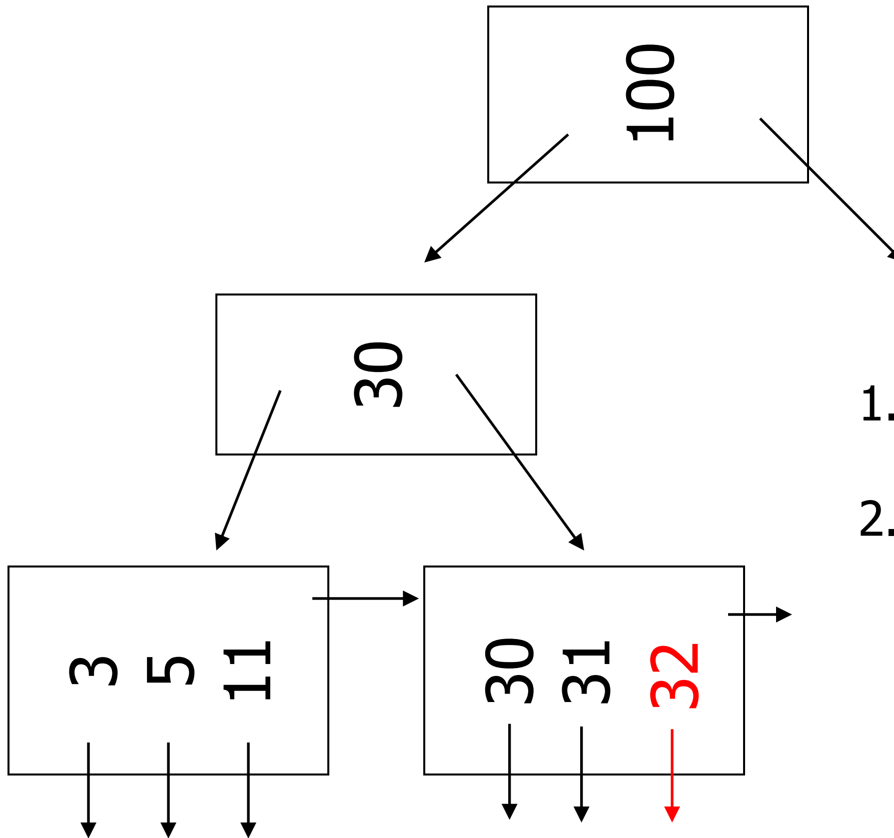
(b) leaf overflow

(c) non-leaf overflow

(d) new root

(a) Insert key = 32

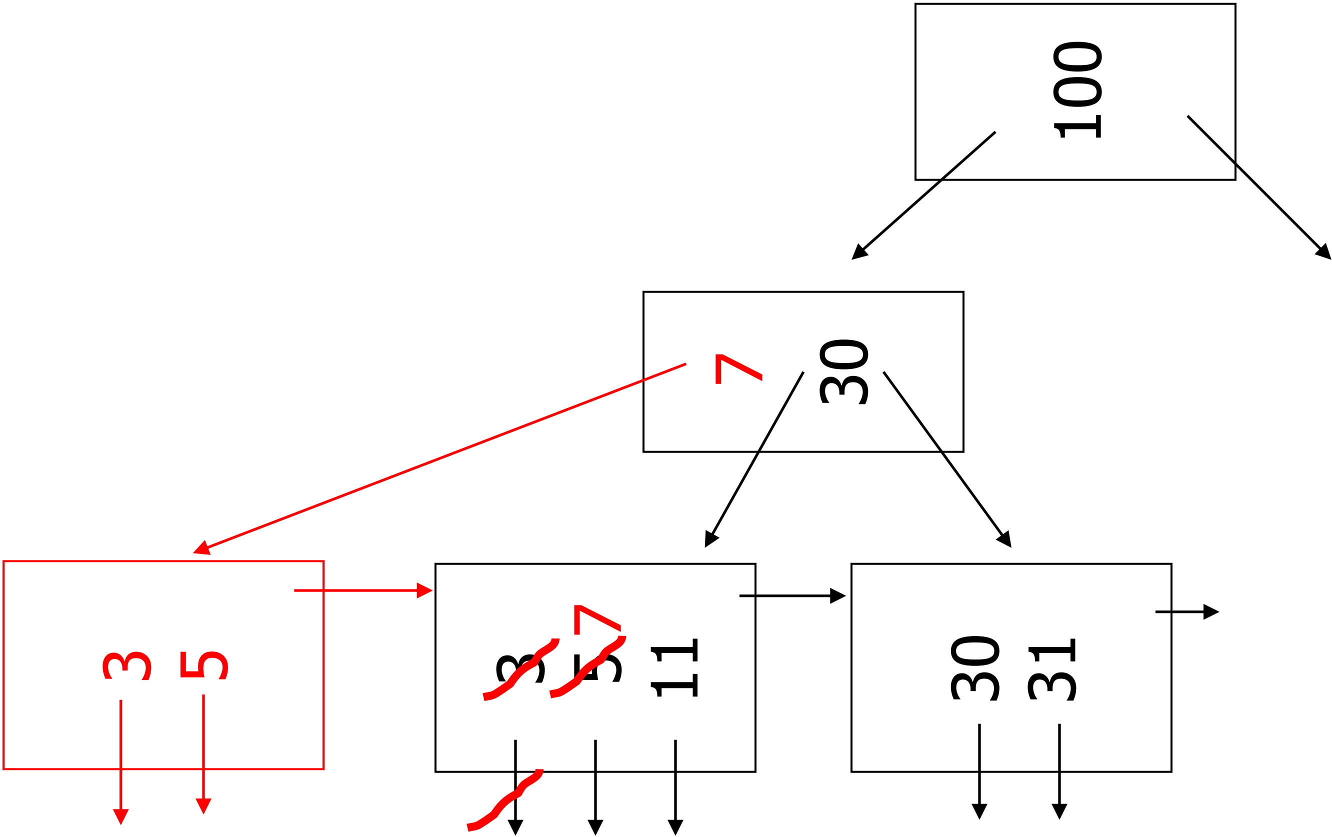
n=3



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

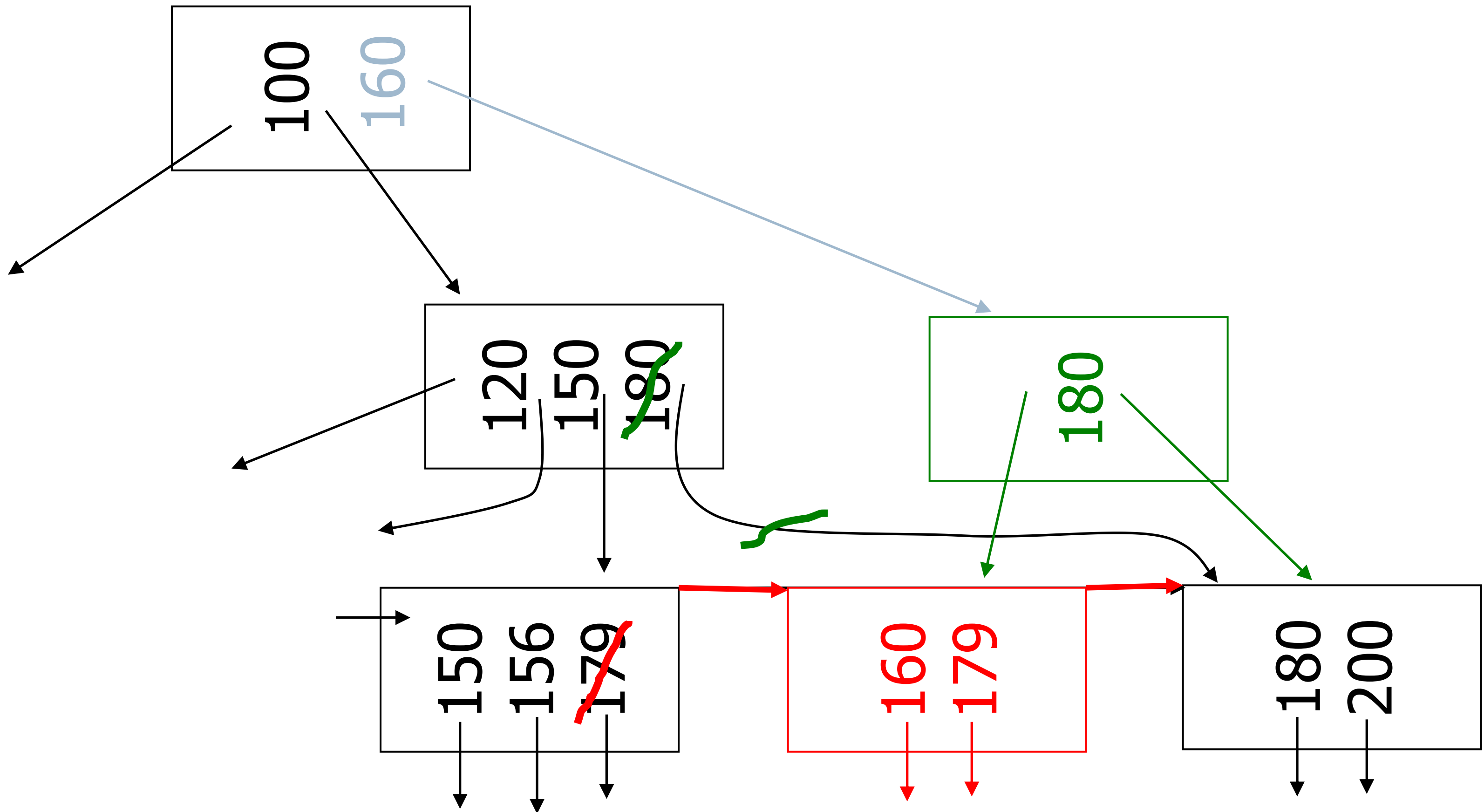
(b) Insert key = 7

n=3



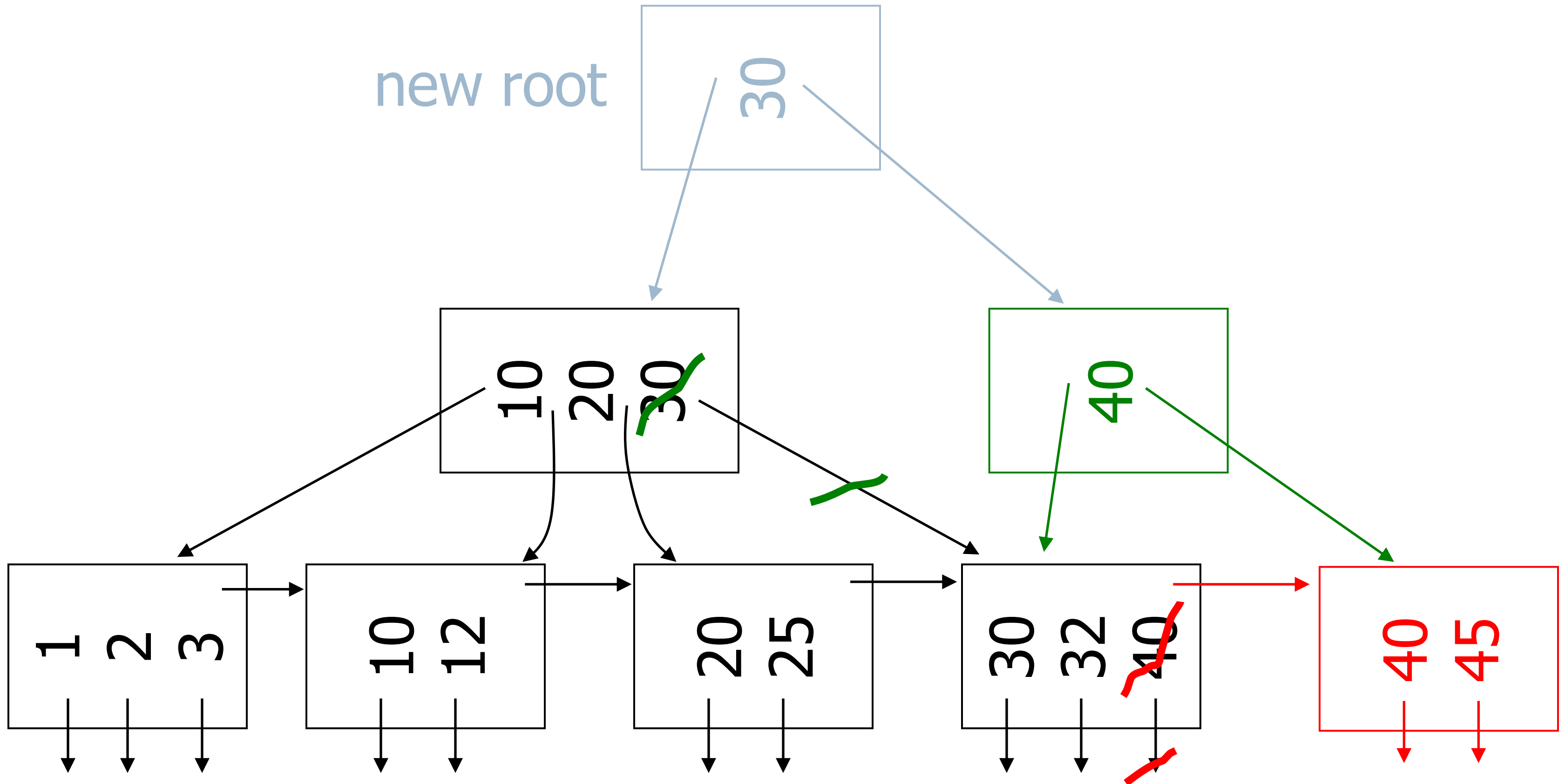
(c) Insert key = 160

n=3



(d) New root, insert 45

n=3



Deletion from B-tree

(a) Simple case - no example

(b) Coalesce with neighbor (sibling)

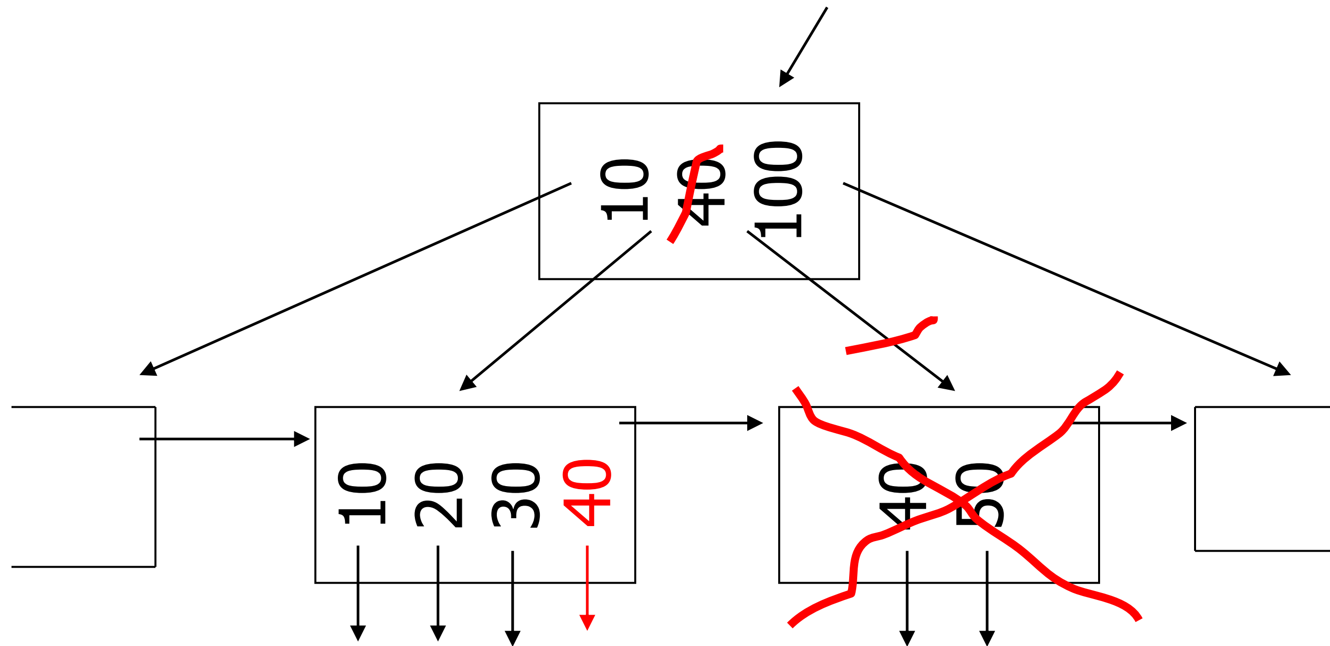
(c) Re-distribute keys

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

(b) Coalesce with sibling

n=4

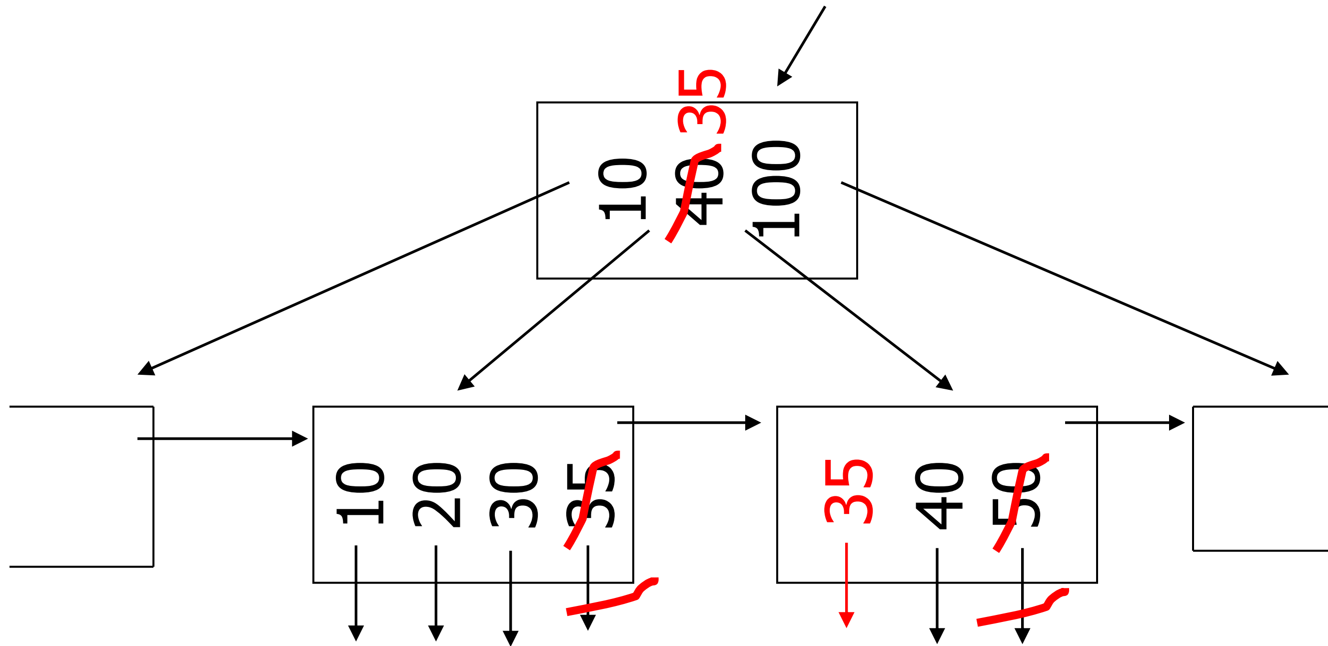
- Delete 50



(c) Redistribute keys

n=4

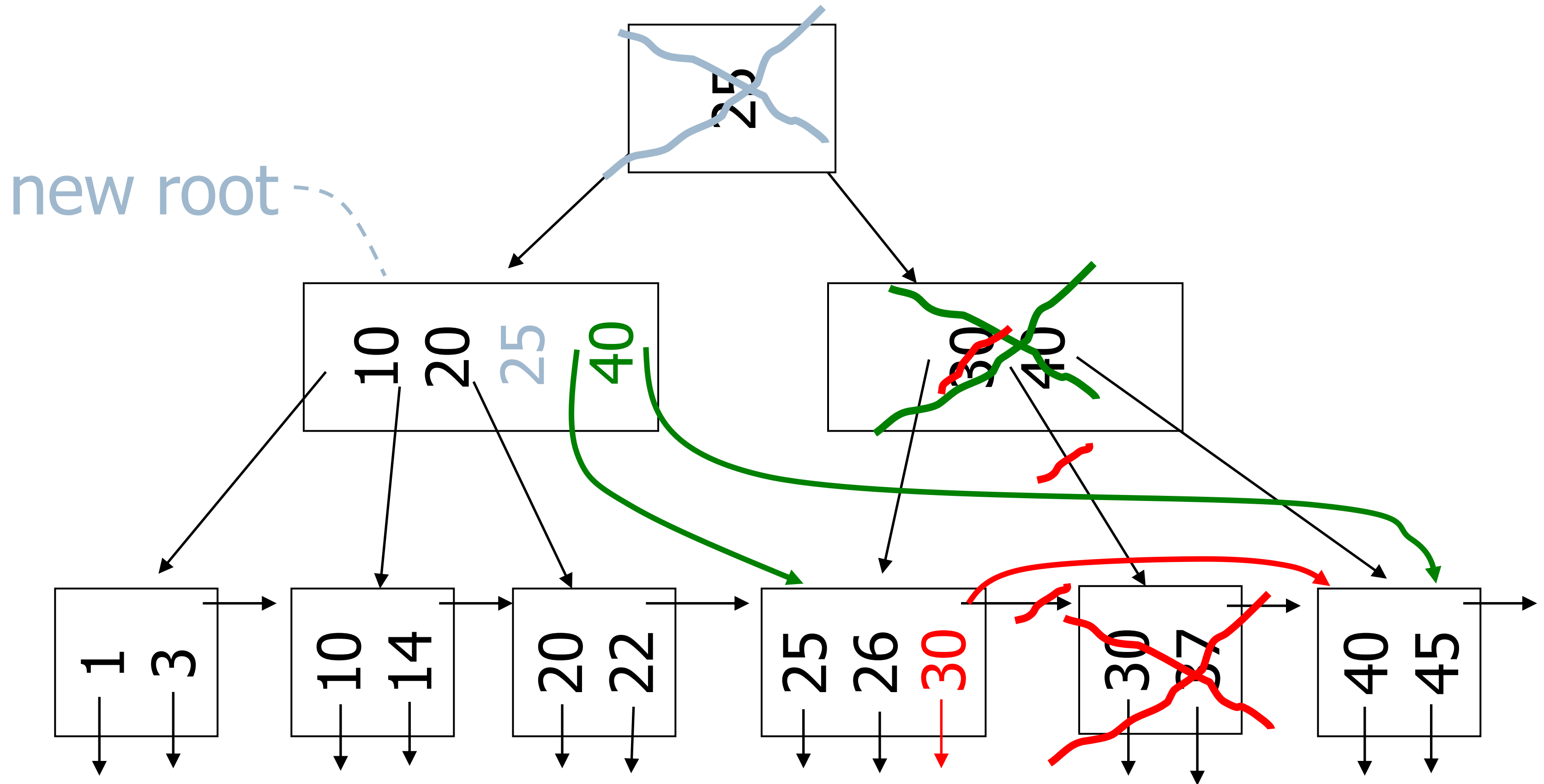
- Delete 50



(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!

Is LRU a good policy for B-tree buffers?

→ Of course not!

→ 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, διεύθυνση) για 100000 υπαλλήλους.
- Θέλετε να φορτώσετε τις εγγραφές του αρχείου στη σχέση EMPLOYEES(eid,name,address) που θα δημιουργήσετε και να φτιάξετε ευρετήριο στο name.
- Ποια από τις παρακάτω ακολουθίες εντολών, A ή B, θα εκτελεστεί ταχύτερα και γιατί;

A:

```
create table EMPLOYEES(eid int, name varchar(100), address varchar(200));
create index I on EMPLOYEES(name);
load data local infile "EMPLOYEES.txt" into table EMPLOYEES;
```

B:

```
create table EMPLOYEES(eid int, name varchar(100), address varchar(200));
load data local infile "EMPLOYEES.txt" into table EMPLOYEES;
create index I on EMPLOYEES(name);
```