Ευρετήρια και Κατακερματισμός

Α’ μέρος

ΣΥΝΟΨΗ ΕΝΟΤΗΤΑΣ

- Ορισμοί - Βασικές έννοιες
- Διατεταγμένα Ευρετήρια
- B+-Tree Δομές Ευρετηρίων

Βασικές έννοιες

- Ένα ***ευρετήριο (index)*** είναι μια βοηθητική δομή αρχείου που κάνει πιο αποδοτική την αναζήτηση μιας εγγραφής σε ένα αρχείο
- **Κλειδί αναζήτησης** – γνώρισμα ή σύνολο γνωρισμάτων που χρησιμοποιείται για αναζήτηση
- Μια καταχώρηση / εγγραφή στο Ευρετήριο έχει την μορφή

Κλειδί αναζήτησης	**Δείκτης στο μπλοκ εγγραφής**

- Η δομή ευρετηρίου καταλαμβάνει μικρότερο χώρο από το ίδιο το αρχείο ***(οι καταχωρήσεις είναι μικρότερες και λιγότερες)***
- 2 Βασικές κατηγορίες:
  - **Διατεταγμένο ευρετήριο:** Ταξινομημένα κλειδιά αναζήτησης
  - **Ευρετήριο κατακερματισμού:** Κλειδιά αναζήτησης ομοιόμορφα κατανεμημένα σε “buckets” με χρήση συνάρτησης hash.

Μετρικές αποτίμησης ευρετηρίων

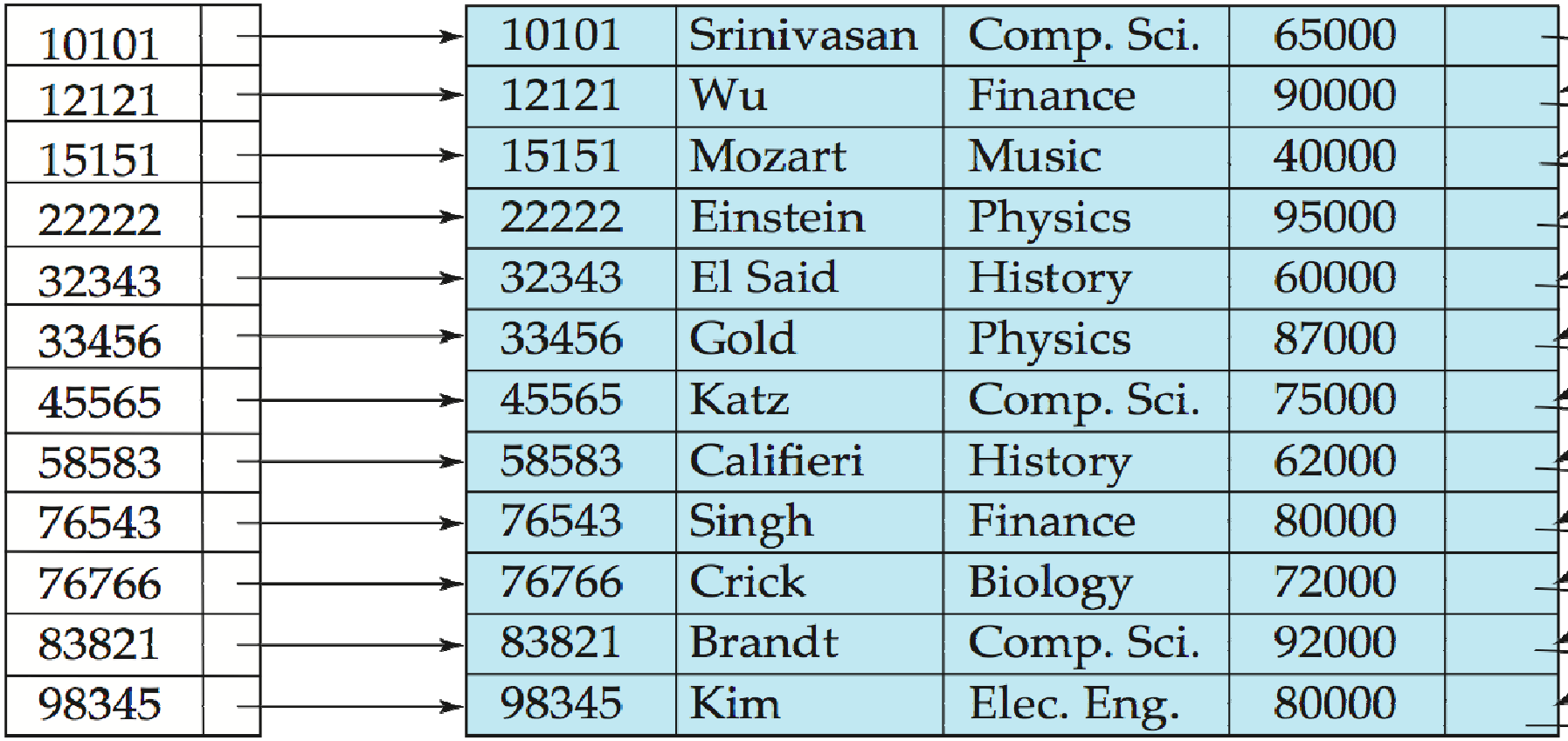
- Τύποι πρόσβασης
  - Εγγραφές με συγκεκριμένη τιμή για ένα γνώρισμα (point queries)
  - Εγγραφές με τιμές γνωρισμάτων σε συγκεκριμένο εύρος
- Χρόνος πρόσβασης
- Χρόνος εισαγωγής
- Χρόνος διαγραφής
- Κόστος σε χώρο

Διατεταγμένα Ευρετήρια

- Σε ένα **διατεταγμένο ευρετήριο,** οι δείκτες αποθηκεύονται ταξινομημένοι ως προς ένα κλειδί αναζήτησης
- **Πρωτεύον ευρετήριο:** σε ένα σειριακά ταξινομημένο αρχείο το κλειδί αναζήτησης ορίζει τη σειριακή ταξινόμηση του αρχείου
  - Λέγεται και **clustering index (ευρετήριο συμπλέγματος)**
  - Το κλειδί αναζήτησης είναι συνήθως, αλλά όχι απαραίτητα, το πρωτεύον κλειδί
- **Δευτερεύον ευρετήριο**: το κλειδί αναζήτησης ορίζει διαφορετική σειρά από τη σειριακή ταξινόμηση του αρχείου.
  - Λέγεται και non-clustering index.
- Αρχεία σειριακού ευρετηρίου**:** αρχείο με πρωτεύον ευρετήριο στο κλειδί αναζήτησης

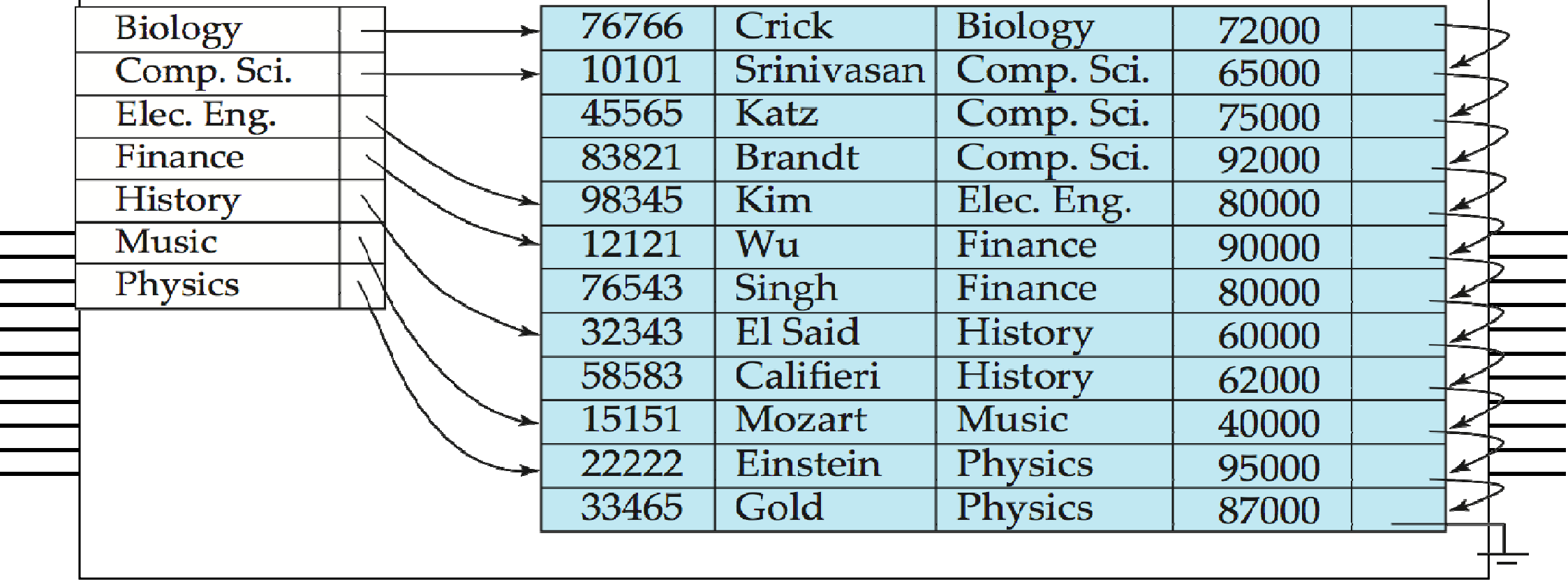
Πυκνά ευρετήρια

- Ένα στοιχείο ευρετηρίου για κάθε τιμή κλειδιού αναζήτησης του αρχείου
- E.g. index on *ID* attribute of *instructor* relation



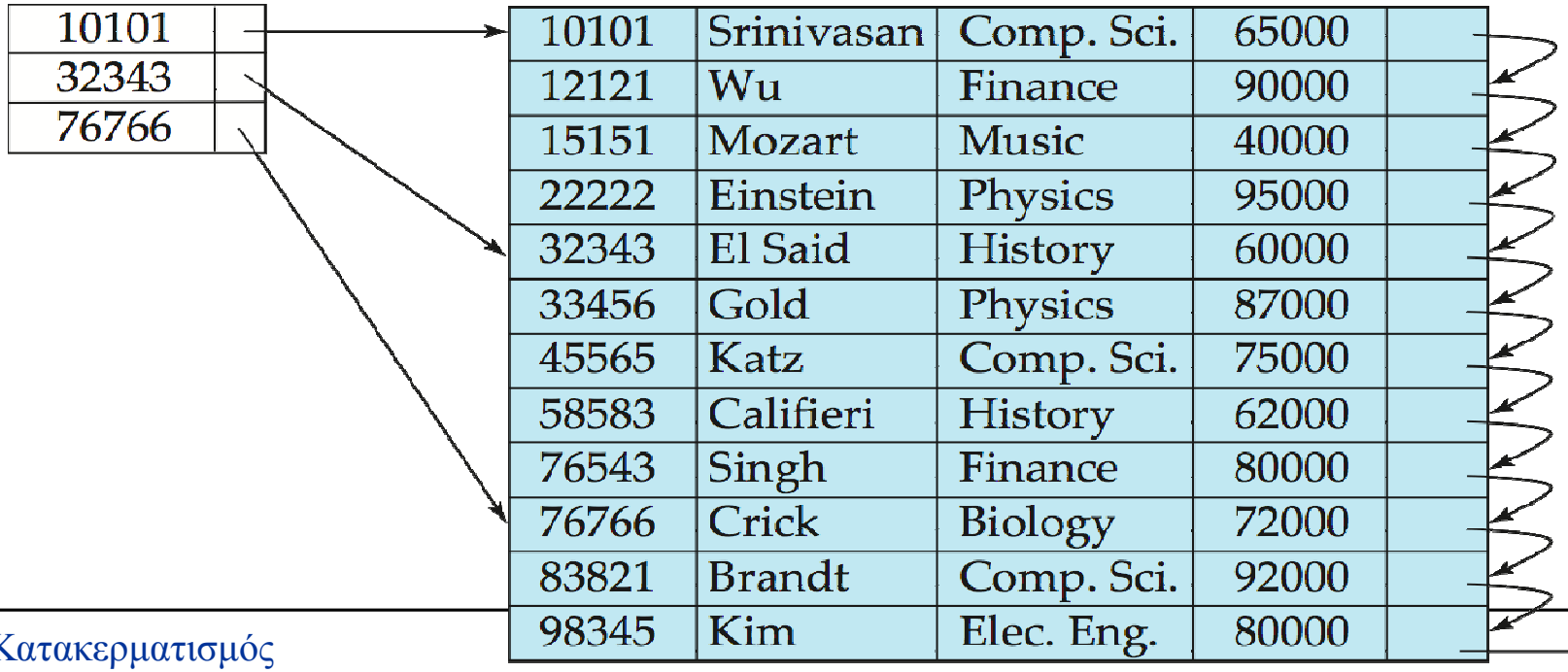
Πυκνά ευρετήρια

- Πυκνό ευρετήριο στο *dept_name*, με το αρχείο *instructor* διατεταγμένο ως προς *dept_name*



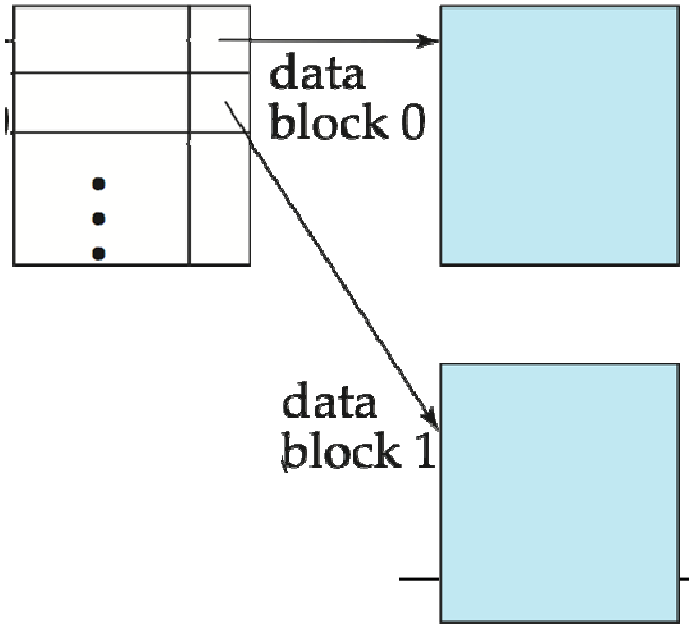
Αραιά Ευρετήρια

- Περιέχουν στοιχεία ευρετηρίου για μερικές μόνο τιμές του κλειδιού αναζήτησης
  - Μόνο για σειριακά διατεταγμένες εγγραφές ως προς το κλειδί αναζήτησης
- Για εύρεση εγγραφών με κλειδί αναζήτησης K :
  - Βρες στοιχείο με την μεγαλύτερη τιμή κλειδιού αναζήτησης < K
  - Σειριακή αναζήτηση ξεκινώντας από την εγγραφή που δείχνει το στοιχείο



Αραιά Ευρετήρια

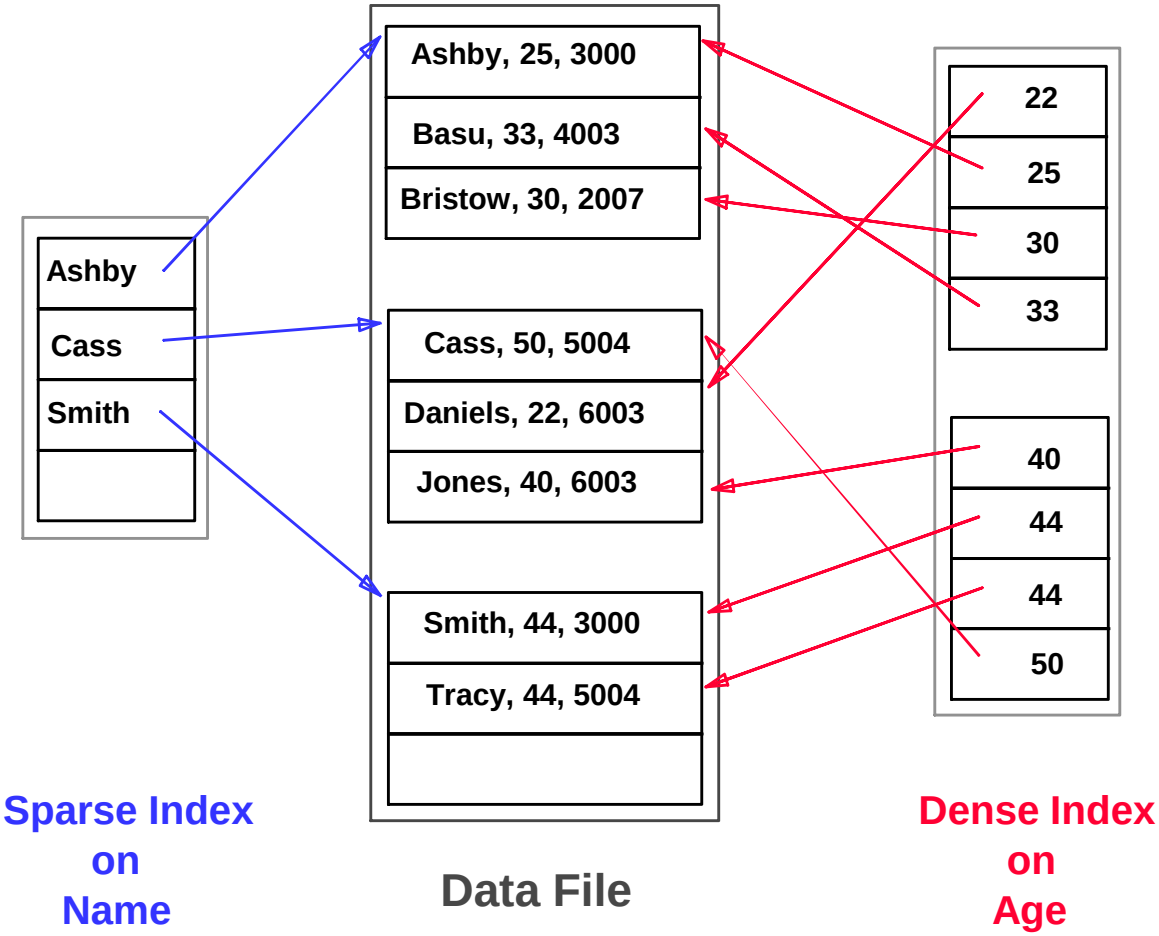
- Σε σύγκριση με τα πυκνά ευρετήρια:
  - Λιγότερος χώρος
  - Μικρότερο κόστος συντήρησης για εισαγωγές και διαγραφές
  - Πιο αργός εντοπισμός εγγραφών
- **Καλός συμβιβασμός**: αραιό ευρετήριο με ένα στοιχείο ευρετηρίου ανά μπλοκ
  - Βασικό κόστος = χρόνος να έρθει το μπλοκ στη μνήμη



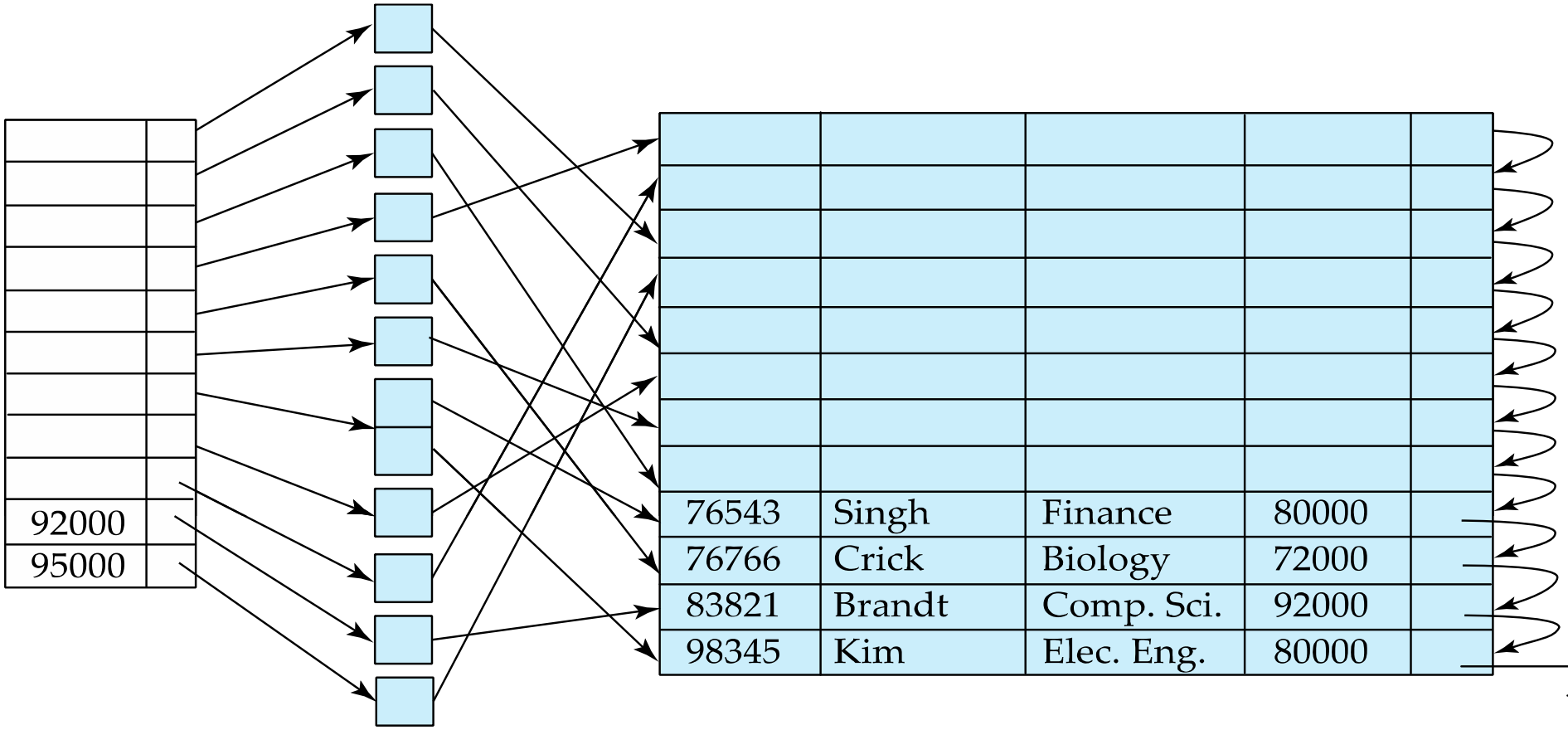
Δευτερεύοντα ευρετήρια

- Συχνά θέλουμε να βρούμε εγγραφές με τιμές σε συγκεκριμένο γνώρισμα που δεν είναι το κλειδί αναζήτησης του πρωτεύοντος ευρετηρίου
  - Παράδειγμα 1: Σχέση *instructor* αποθηκευμένη σειριακά ως προς ID: θέλουμε να βρούμε καθηγητές συγκεκριμένου department
  - Παράδειγμα 2: Σχέση *instructor* αποθηκευμένη σειριακά ως προς ID: θέλουμε να βρούμε καθηγητές συγκεκριμένου εύρους salary
- We can have a secondary index with an index record for each search-key value

Πυκνά και αραιά ευρετήρια



Παράδειγμα δευτερεύοντος ευρετηρίου



Secondary index on *salary* field of *instructor*

- Αν κλειδί αναζήτησης είναι υποψήφιο κλειδί τότε όπως πυκνά πρωτεύοντα ευρετήρια
- Αλλιώς επιπλέον επίπεδο έμμεσων δεικτών: bucket με δείκτες σε όλες τις εγγραφές για ένα κλειδί αναζήτησης
- Πρέπει να είναι πυκνά

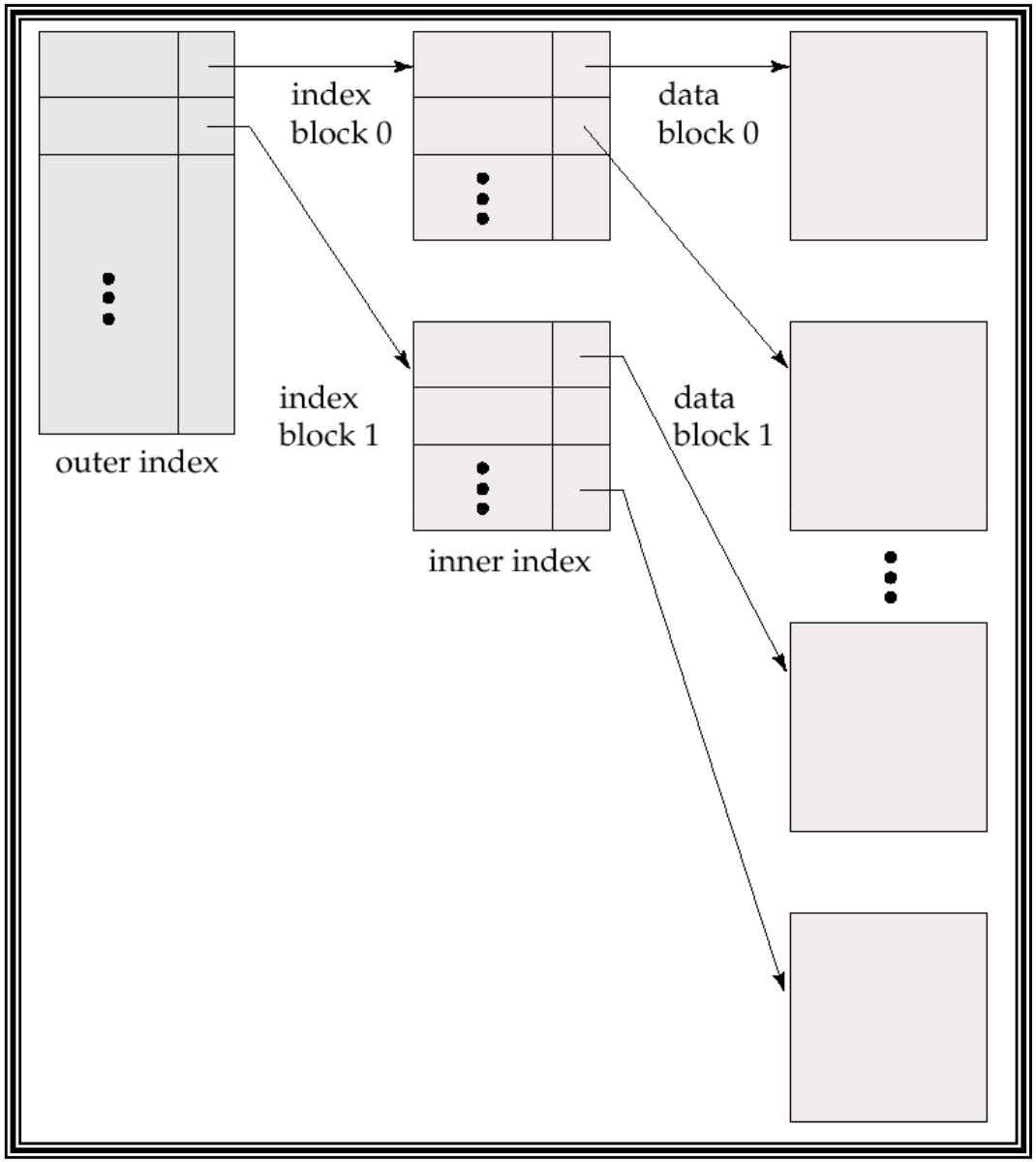
Πρωτεύοντα και δευτερεύοντα ευρετήρια

- Τα ευρετήρια επιταχύνουν την αναζήτηση εγγραφών
- Αλλά: Η ενημέρωση των ευρετηρίων σημαίνει επιπλέον κόστος – όταν ενημερώνεται μια εγγραφή πρέπει να ενημερωθεί και κάθε δείκτης
- Σειριακή αναζήτηση με πρωτεύον ευρετήριο είναι αποδοτική, αλλά δευτερεύον ακριβή
  - Κάθε προσπέλαση εγγραφής μπορεί να φέρει καινούριο μπλοκ απτό τον δίσκο
  - ~5 - 10 milliseconds vs 100 nanoseconds για προσπέλαση μνήμης

Ευρετήρια Πολλών Επιπέδων

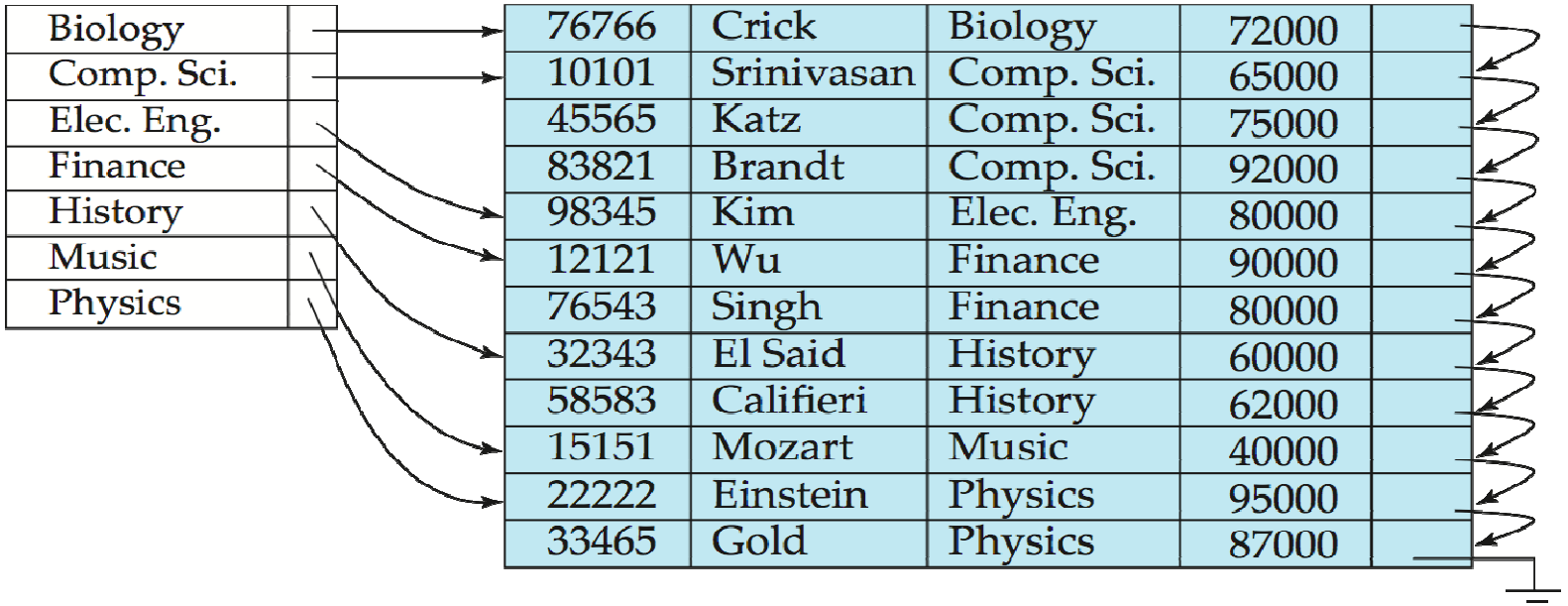
- Αν το πρωτεύον ευρετήριο δε χωρά στη μνήμη τότε γράφεται σε αρχεία, άρα η προσπέλαση είναι ακριβή
- **Τα Αρχεία Ευρετηρίων είναι απλά Αρχεία, άρα και σε αυτά μπορούν να οριστούν Ευρετήρια**
- Καταλήγουμε λοιπόν σε μια **ιεραρχία δομών ευρετηρίων** (πρώτο επίπεδο, δεύτερο επίπεδο, κλπ.)
- Κάθε επίπεδο του ευρετηρίου είναι ένα **διατεταγμένο αρχείο**, συνεπώς, Εισαγωγές/Διαγραφές εγγραφών απαιτούν επιπλέον δουλειά
- Ένα πολύ-επίπεδο ευρετήριο αποτελεί ένα **Δέντρο Αναζήτησης** με την υπόθεση ότι το πρώτο επίπεδο **χωρά σε ένα Μπλοκ**

Ευρετήρια Πολλών Επιπέδων

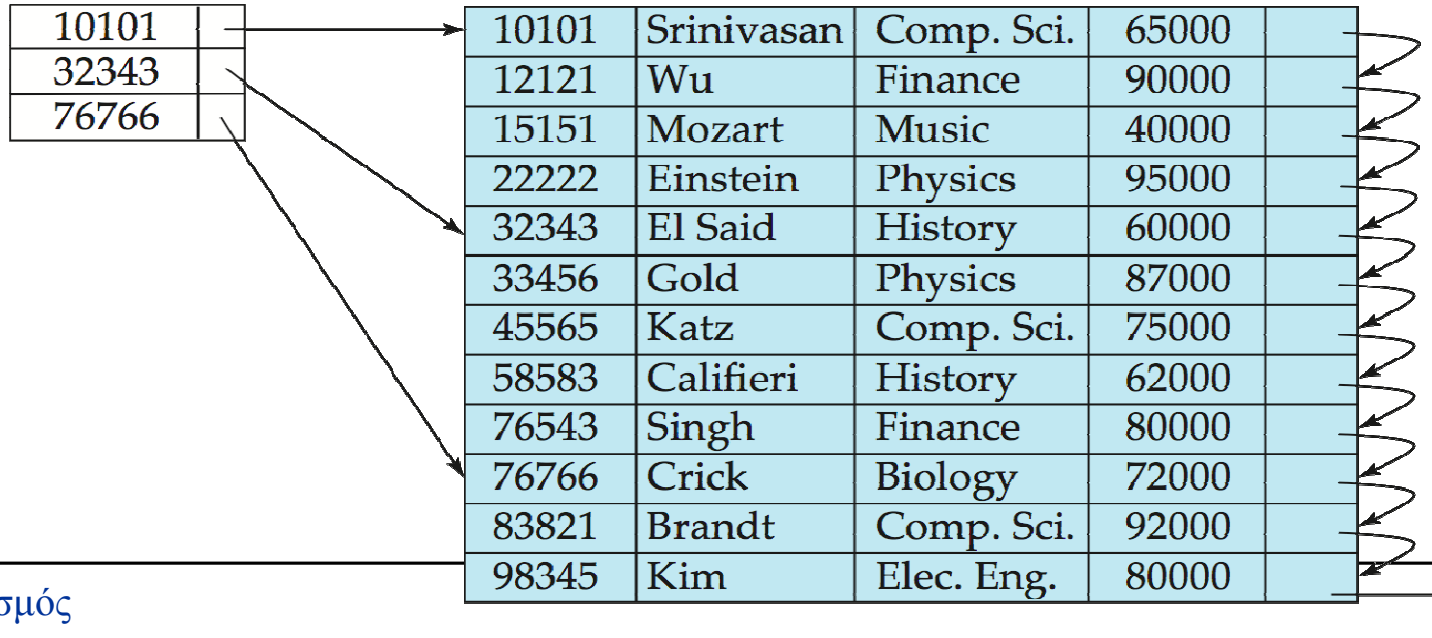


Ενημέρωση ευρετηρίου: Διαγραφή

– **Πυκνά**



– **Αραιά**

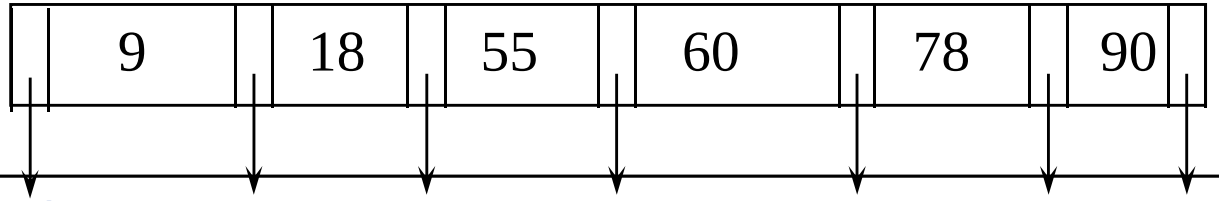


Ενημέρωση ευρετηρίου: Εισαγωγή

- **Ενός επιπέδου**
  - **Πυκνά**– Αν το κλειδί αναζήτησης δεν υπάρχει στο ευρετήριο το εισάγουμε στην κατάλληλη θέση
  - **Αραιά**– αν αποθηκεύεται μια καταχώρηση για κάθε μπλοκ αρχείου δεν αλλάζει κάτι αν δε δημιουργηθεί νέο μπλοκ
    - » Αν δημιουργηθεί νέο μπλοκ τότε το πρώτο κλειδί αναζήτησης στο block δημιουργεί καινούρια καταχώρηση στο ευρετήριο
- **Πολλαπλών επιπέδων:** επέκταση των αλγορίθμων ενός επιπέδου

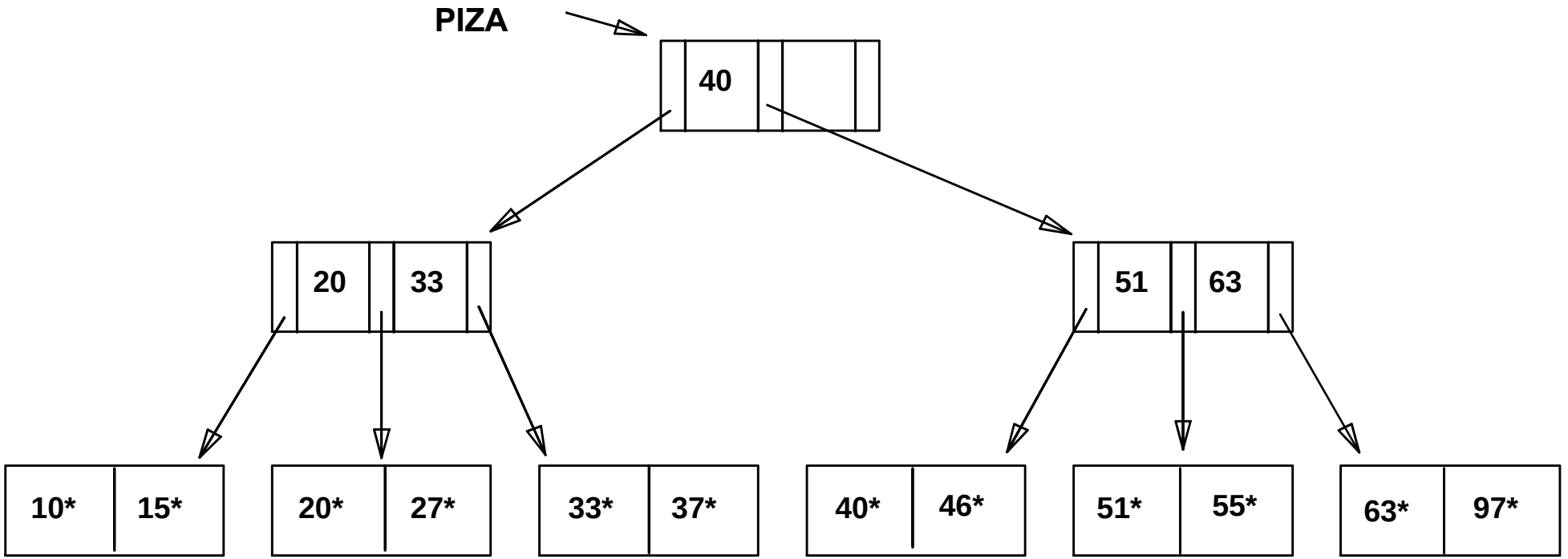
Indexed Sequential Access Method (ISAM)

- Το ISAM είναι μια πολύ-επίπεδη **Δομή Ευρετηρίου (Δέντρο)** για ***άμεση πρόσβαση*** σε εγγραφές αρχείου **διατεταγμένου στο Κλειδί**
- Κάθε κόμβος του Δέντρου είναι **ένα Μπλοκ** στο Δίσκο
- Οι κόμβοι του Δέντρου κρατούν **<τιμή κλειδιού, δείκτης>** ζεύγη, ταξινομημένα στην **τιμή κλειδιού.** Οι εσωτερικοί κόμβοι ¨δείχνουν¨ σε χαμηλότερου επιπέδου κόμβους ενώ τα φύλλα-κόμβοι ¨δείχνουν¨ σε Μπλοκ του αρχείου (Σχέσης). Ένας Δείκτης ¨δείχνει¨ ένα υπό-δέντρο με τιμές κλειδιού ΜΕΓΑΛΥΤΕΡΕΣ ή ΙΣΕΣ της αντίστοιχης τιμής κλειδιού και ΜΙΚΡΟΤΕΡΕΣ της τιμής Κλειδιού του επόμενου Δείκτη



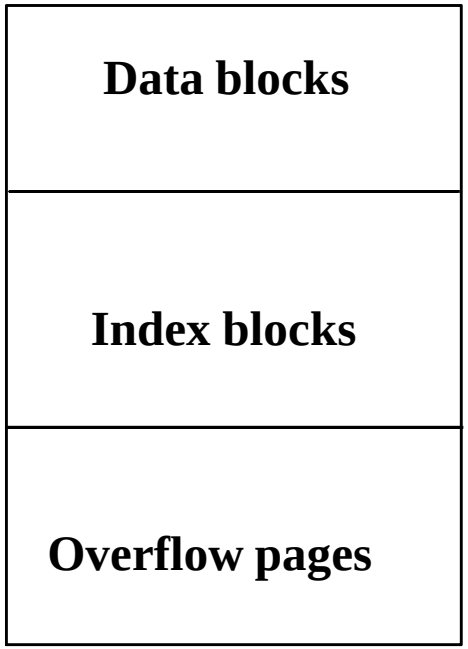
ISAM Δέντρο -- Παράδειγμα

- Κάθε κόμβος μπορεί να έχει 2 καταχωρήσεις



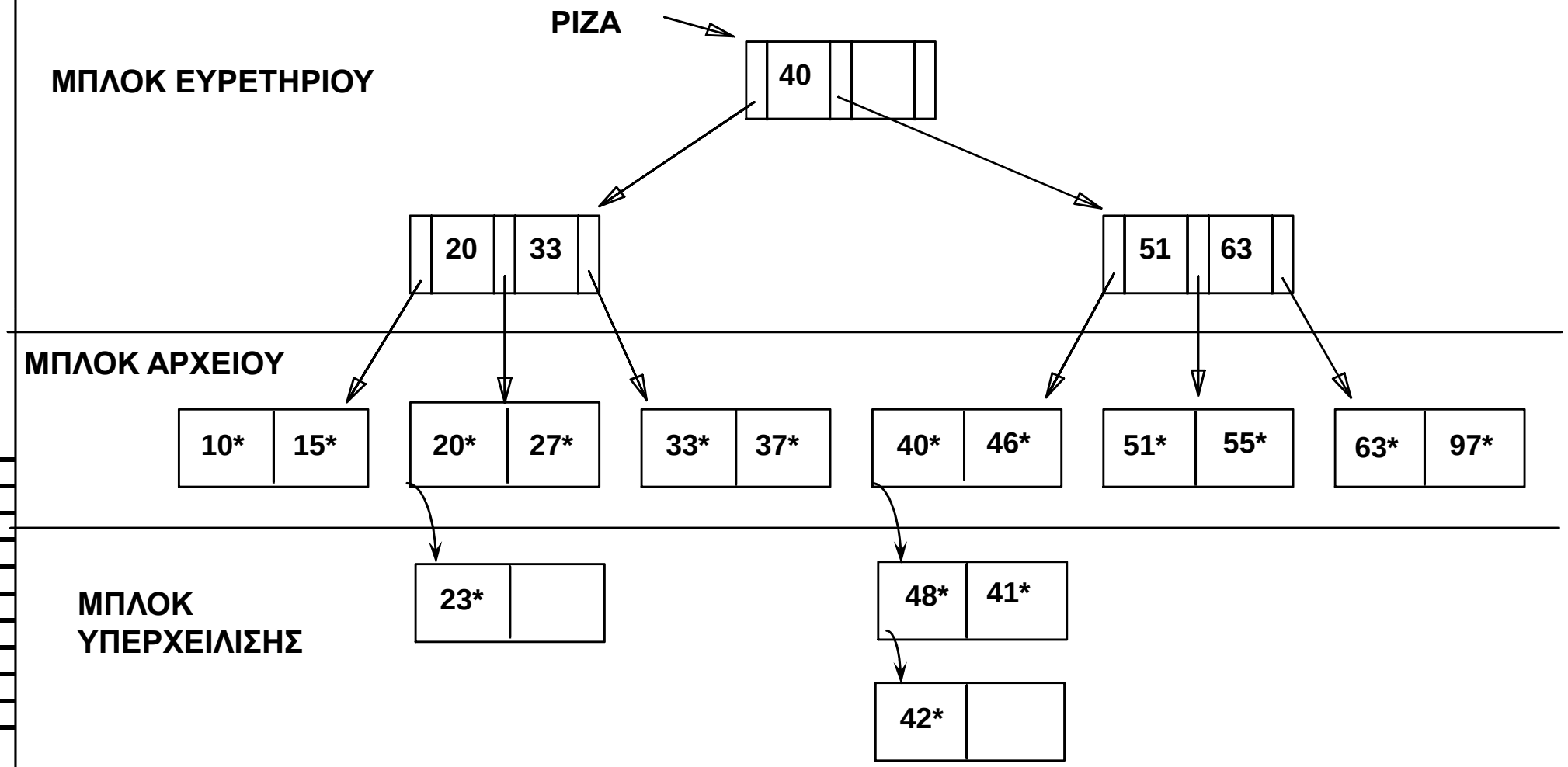
Βασικές λειτουργίες

- *Δημιουργία*: 1. Φύλλα (μπλοκς δεδομένων) σειριακά, ταξινομημένα με βάση το κλειδί αναζήτησης
  2. Μπλοκς ευρετηρίου
  3. Χώρος για overflow blocks
- *Καταχωρήσεις ευρετηρίου*: <search key value, block id>; Και μετά άμεση αναζήτηση δεδομένων που βρίσκονται στα φύλλα
- *Αναζήτηση*: Ξεκινάμε από τη ρίζα; Συγκρίσεις κλειδιών για να φτάσουμε σε φύλλο.
- *Εισαγωγή*: Βρες το φύλλο που ανήκει η νέα τιμή (Μπορεί να χρειαστεί overflow block).
- *Διαγραφή*: Βρες και διάγραψε τιμή από το φύλλο; Αν αδειάσει ένα overflow block αποδέσμευσε χώρο

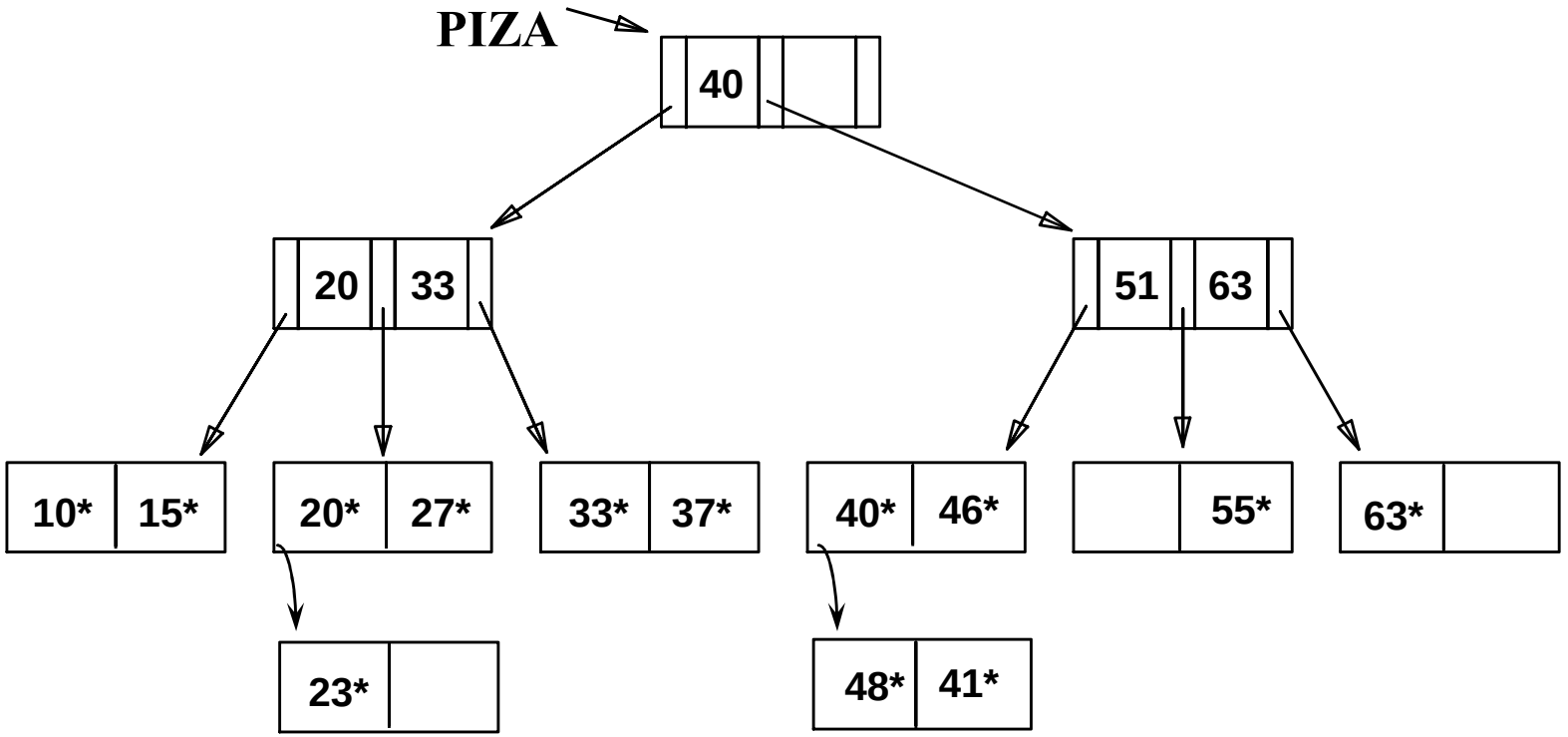


✉ **Static tree structure:** ***inserts/deletes affect only leaf pages.***

Μετά την εισαγωγή των 23*, 48*, 41*, 42* ...



Μετά την διαγραφή των 42*, 51*, 97*



Παρατηρείστε ότι το 51* είναι στο ευρετήριο, αλλά όχι στο Αρχείο!

Η Επίδοση του ISAM

- **ΕΠΙΔΟΣΗ (PERFORMANCE).**

  Έστω ότι έχουμε ***D blocks*** για Δεδομένα (φύλλα) και ***k*** δείκτες για κάθε κόμβο

  ΔΙΑΔΙΚΑΣΙΑ ΑΝΑΖΗΤΗΣΗΣ:

Σειριακή Σάρωση (scan):	D
Δυαδική Αναζήτηση Σχέσης:	$\log_2 D$
Δυαδική Αναζήτηση (μονό επίπεδο):	$\log_2 (D/k)$
Διάσχιση του ISAM Δέντρου:	$\log_k D$

ISAM -- ΣΧΟΛΙΑ

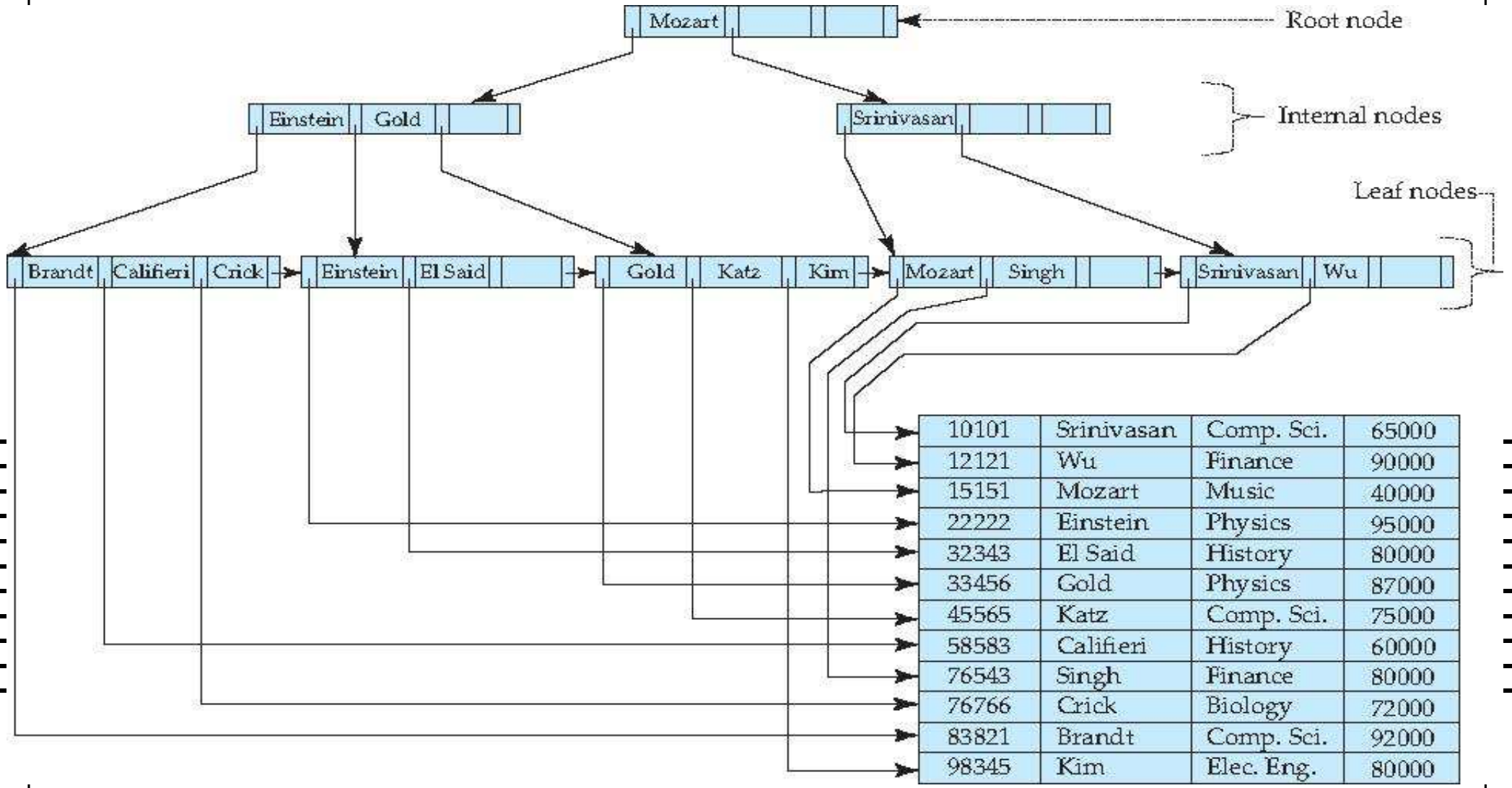
- **ΠΛΕΟΝΕΚΤΗΜΑΤΑ:**
  - Παρέχει έναν **ταξινομημένο κατάλογο** για το Αρχείο (ή Σχέση)
  - Εξαιρετική δομή για **ακριβείς ερωτήσεις (exact queries)** π.χ., *Salary = 400000*
  - Το ISAM διευκολύνει την εκτέλεση των **ερωτήσεων διακύμανσης** (**range queries).** π.χ., *Salary μεταξύ 350000 και 600000*

- **ΜΕΙΟΝΕΚΤΗΜΑΤΑ**
  - Είναι μια **ΣΤΑΤΙΚΗ** δομή που εύκολα χάνει την ¨ισορροπία¨ της (**unbalanced) -** *ισορροπία ύψους και πληρότητας των κόμβων*
  - Αν στο Αρχείο γίνονται πολλές ενημερώσεις (volatile δεδομένα), τότε το Αρχείο μπορεί να χάσει την ταξινόμησή του.
  - Το Ευρετήριο απαιτεί μεγάλο χώρο το Δίσκο

B^+ - Δέντρα

- Το **B^+-Δέντρο** είναι μια πολύ-επίπεδη δομή ευρετηρίου για ένα διατεταγμένο αρχείο
- Οι κόμβοι-φύλλα **περιέχουν ταξινομημένες εγγραφές (πλειάδες)**, οι άλλοι (εσωτερικοί) κόμβοι έχουν **ειδική μορφή**
- Ένας κόμβος αντιστοιχεί σε ένα **Μπλοκ**
- Κάθε κόμβος κρατιέται **κάτι μεταξύ γεμάτου και μισό-γεμάτου**
- Οι Εισαγωγές σε κόμβους, που **δεν είναι γεμάτοι, γίνονται αποδοτικά**. Αν ένας κόμβος είναι γεμάτος, τότε έχουμε **διάσπαση**
- Οι Διαγραφές γίνονται **πολύ αποδοτικά** αν ο κόμβος **δεν καθίσταται** λιγότερο από μισό-γεμάτος (ώστε να απαιτήσει ένωση)
- Η δομή Δέντρου παραμένει ανά πάσα στιγμή **ΙΣΟΡΡΟΠΗΜΕΝΗ**

Παράδειγμα B+- Δέντρα



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

B^+ - Δέντρα - Σύνοψη

A B^+-tree is a rooted tree satisfying the following properties:

- Όλα τα μονοπάτια από τη ρίζα στα φύλλα είναι ίδιου μήκους
- Κάθε κόμβος που δεν είναι ρίζα ούτε φύλλο έχει από $\lceil n/2 \rceil$ έως και n παιδιά
- Κάθε κόμβος φύλλο έχει μεταξύ $\lceil (n-1)/2 \rceil$ και $n-1$ τιμές
- Ειδικές περιπτώσεις:
  - Αν η ρίζα δεν είναι φύλλο, έχει τουλάχιστον 2 παιδιά.
  - Αν η ρίζα είναι φύλλο (αν δλδ δεν υπάρχουν άλλοι κόμβοι στο δέντρο), έχει από 0 έως και $(n-1)$ τιμές.

B^+ - Δέντρα - Σύνοψη

- Μορφή Εσωτερικών Κόμβων:

P_1	K_1	P_2	K_2	...	...	P_{n-1}	K_{n-1}	P_n

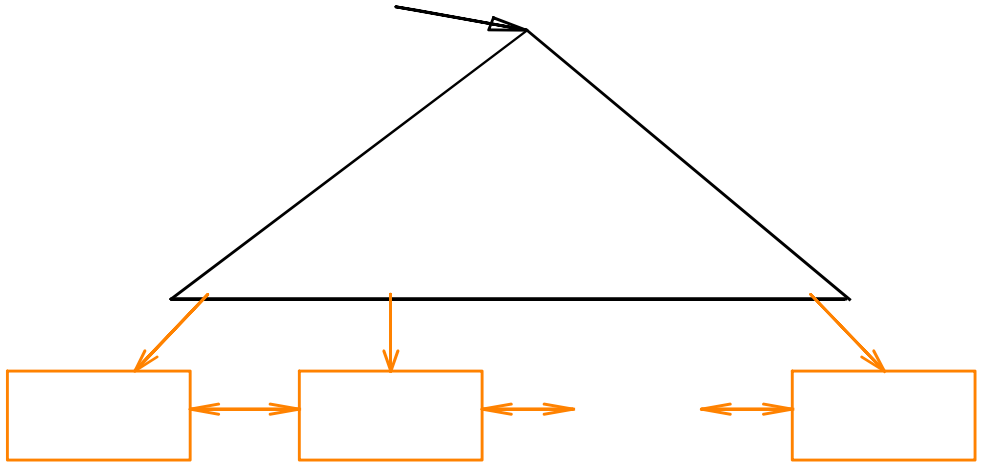
$K_1 < K_2 < ... K_{n-1}$

Το P_1 δείχνει ένα κόμβο που περιέχει τιμές κλειδιού n, $n < K_1$
Το P_2 δείχνει ένα κόμβο που περιέχει τιμές κλειδιού n, $K_1 <= n < K_2$

- **Παραλλαγές των B^+- Δέντρων:**
  - B-Δέντρα : *Σαν τα B^+-Δέντρα, αλλά οι εσωτερικοί κόμβοι περιέχουν επιπλέον δείκτες σε δεδομένα. Είναι συνήθως πιο μεγάλα και είναι δύσκολο να υλοποιηθούν, ενίοτε όμως, είναι ταχύτερα*
  - B*-Δέντρα : *Σαν τα B^+-Δέντρα, αλλά κρατούν κάθε κόμβο γεμάτο (τουλάχιστον) κατά τα 2/3. Μικρότερα και ταχύτερα δέντρα, αλλά πολύ χειρότερα για Εισαγωγές /Διαγραφές*

B^+ - Δέντρα – Τα πλέον δημοφιλή Ευρετήρια

- Εισαγωγή / Διαγραφή με κόστος $\log_F N$ - κρατούν το Δέντρο σε *ισορροπημένη μορφή*. (F = εξάπλωση, N = αριθμός των φύλλων)
- Ελάχιστη πληρότητα 50% (εκτός της Ρίζας). Κάθε κόμβος περιέχει **d <= *m*** **<= 2d** καταχωρήσεις. Το **d** ονομάζεται ***Τάξη*** *του Δέντρου*.
- Εξαιρετική δομή ΚΑΙ για exact queries ΚΑΙ για range queries.

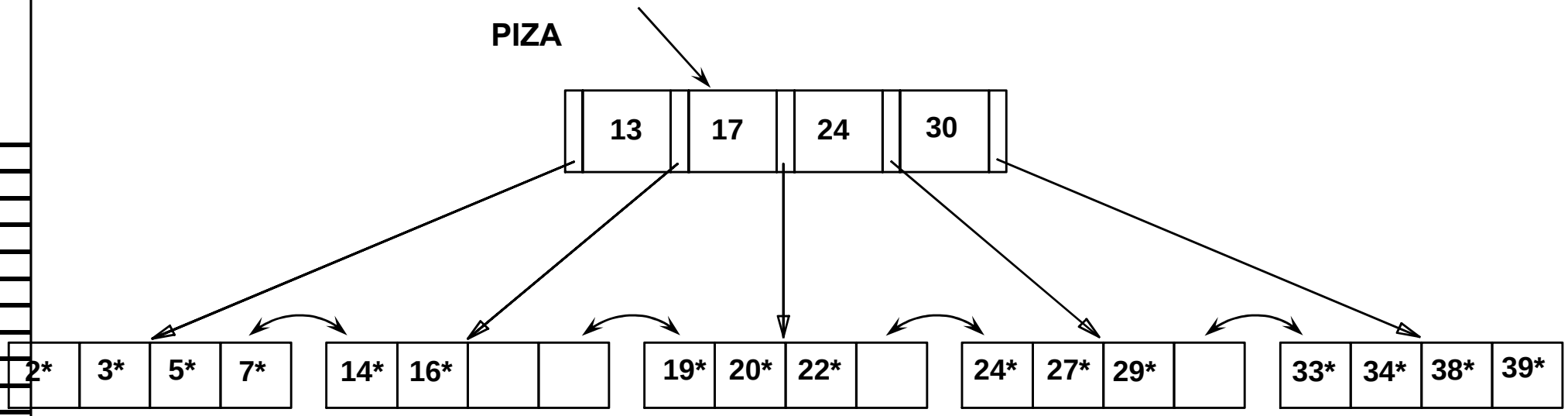


Καταχωρήσεις Ευρετηρίου
(Άμεση Αναζήτηση)

Καταχωρήσεις Δεδομένων
(«Σύνολο ακολουθίας»)

Αναζήτηση B+ Δέντρου

- Η αναζήτηση ξεκινά από τη Ρίζα, και οι συγκρίσεις των κλειδιών μας οδηγούν στα φύλλα (όπως στο ISAM).
- Αναζήτηση για τα 5*, 15*, όλες οι καταχωρήσεις >= 24* ...



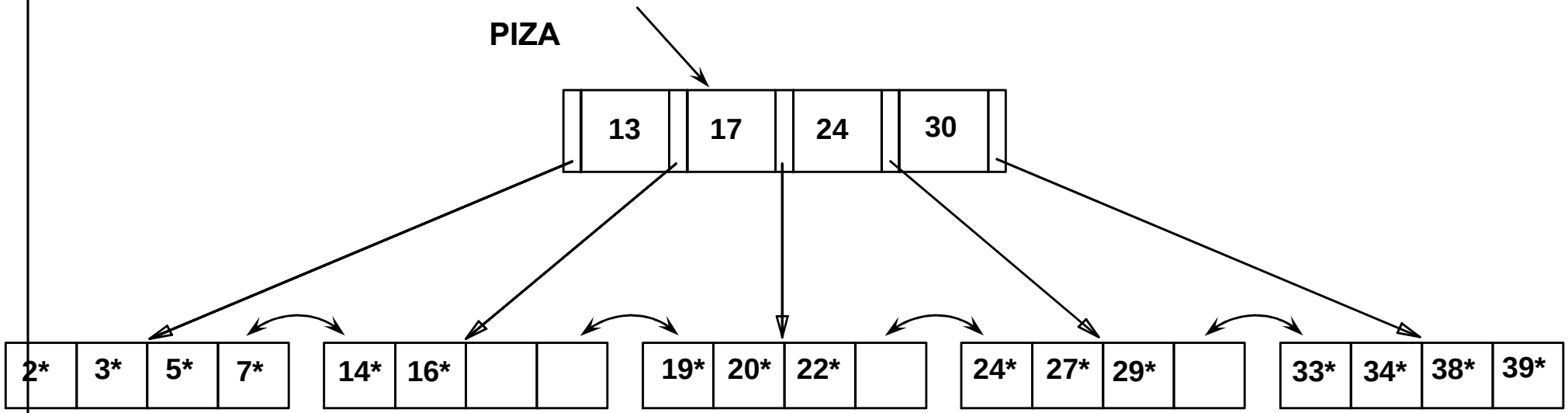
Αναζήτηση B+ Δέντρου

- Αν υπάρχουν K κλειδιά αναζήτησης στο αρχείο, το ύψος του δέντρου δεν είναι μεγαλύτερο από $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$.
- Ένα κόμβος = ένα block, τυπικά 4 kilobytes
  - Και n είναι συνήθως100 (40 bytes per index entry).
- Με K=1 εκατομμύριο κλειδιά αναζήτησης και n = 100
  - at most $log_{50}(1{,}000{,}000) = 4$ κόμβοι για κάθε αναζήτηση
- Δυαδικό δέντρο με 1 εκατομμύριο κλειδιά αναζήτησης — περίπου 20 κόμβοι για 1 αναζήτηση
  - Μεγάλη διαφορά σε disk I/O, άρα και σε χρόνο

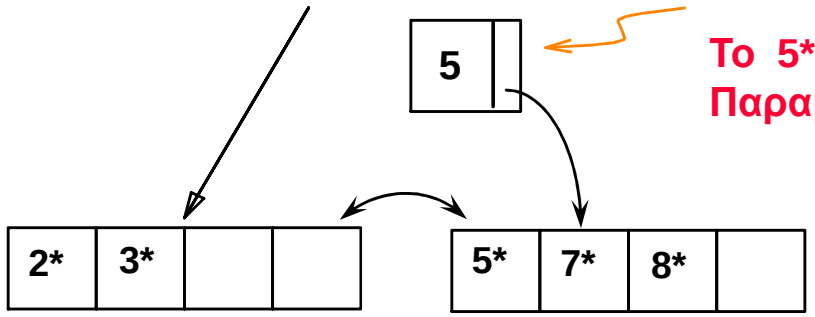
Η Εισαγωγή μιας καταχώρησης δεδομένων (εγγραφής)

- Βρες το σωστό Φύλλο *L*.
- Βάλε την καταχώρηση στο *L*.
  - Αν το *L* έχει αρκετό χώρο, *τελείωσες*!
  - Αλλιώς, πρέπει να *διασπαστεί* το *L (στο L και ένα νέο κόμβο L2)*
    - » Κάνε ισόρροπη ανακατανομή των καταχωρήσεων *(split key)*
    - » Βάλε νέο δείκτη ευρετηρίου στον πατέρα του *L* να δείχνει στο *L2*
    - » Η παραπάνω διαδικασία μπορεί να χρειάζεται να γίνει **αναδρομικά**
  - *Σε διάσπαση εσωτερικών κόμβων*, κάνε το ίδιο, αλλά *σπρώξε πάνω* το split key (διαφοροποίηση από τα φύλλα)
- Οι διασπάσεις κόμβων κάνουν το δέντρο πιο πλατύ
- Η διάσπαση της ρίζας κάνει το δέντρο πιο ψηλό

Η Εισαγωγή της καταχώρησης 8*

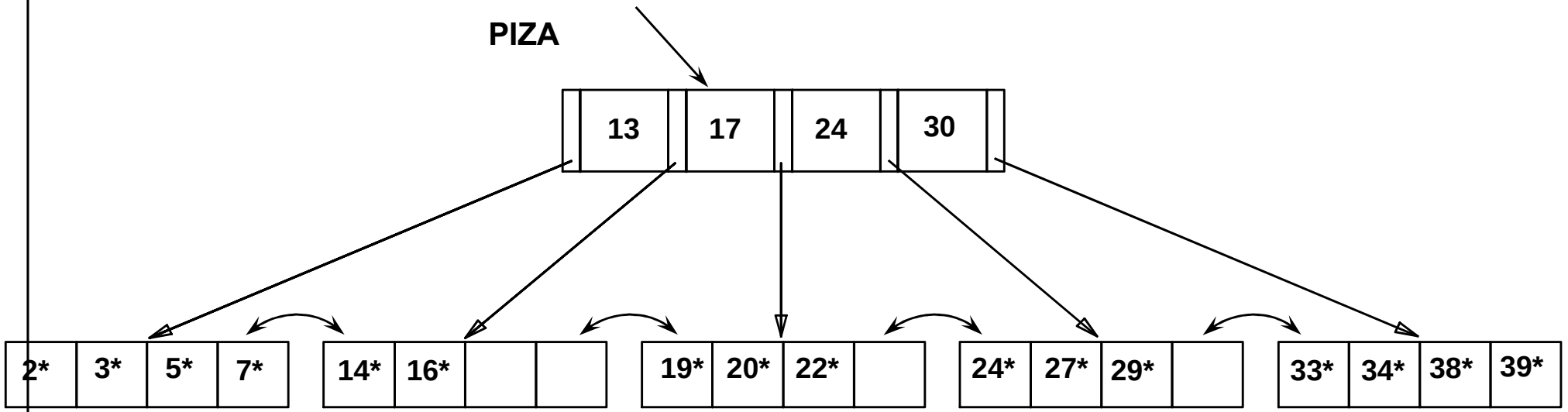


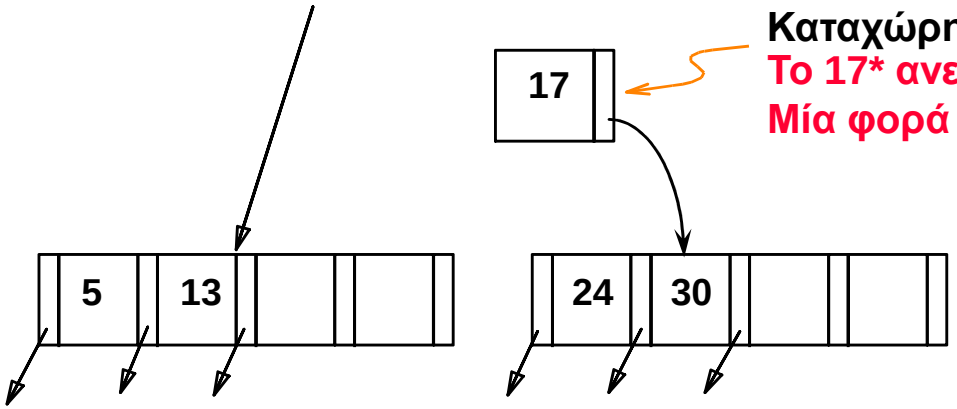
Καταχώρηση στον πατέρα κόμβο.



Το 5* ανεβαίνει πάνω, αλλά
Παραμένει και στο φύλλο

Η Εισαγωγή της καταχώρησης 8*



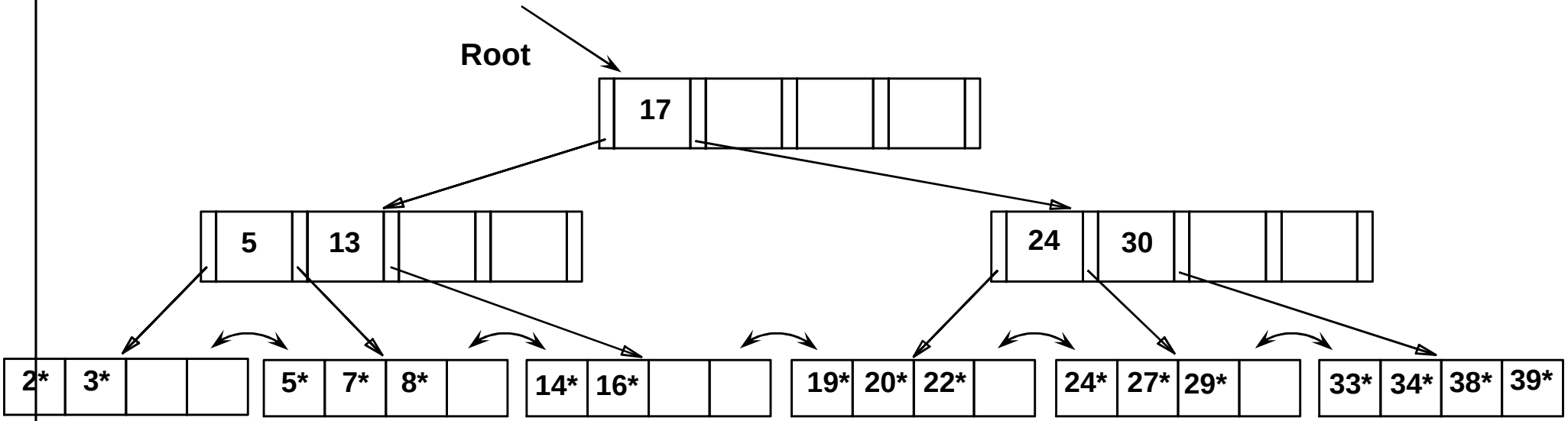


Καταχώρηση στον Πατέρα Κόμβο

Το 17* ανεβαίνει πάνω και παρουσιάζεται μόνο

Μία φορά στο Ευρετήριο (σε αντίθεση με Φύλλα)

Τελικό B+ Δέντρο Μετά την Εισαγωγή του 8*

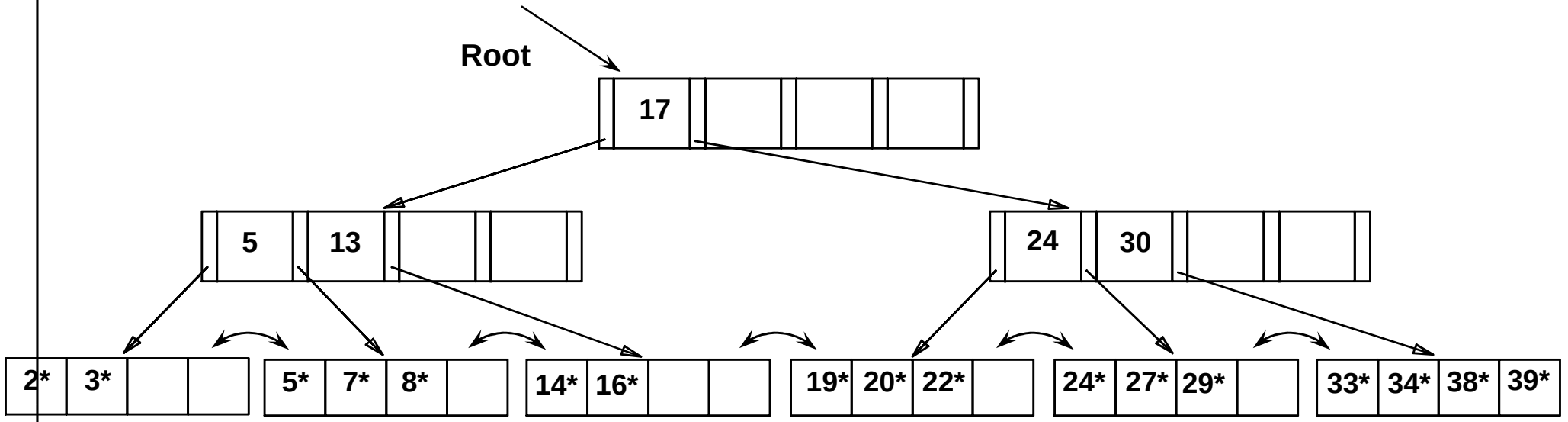


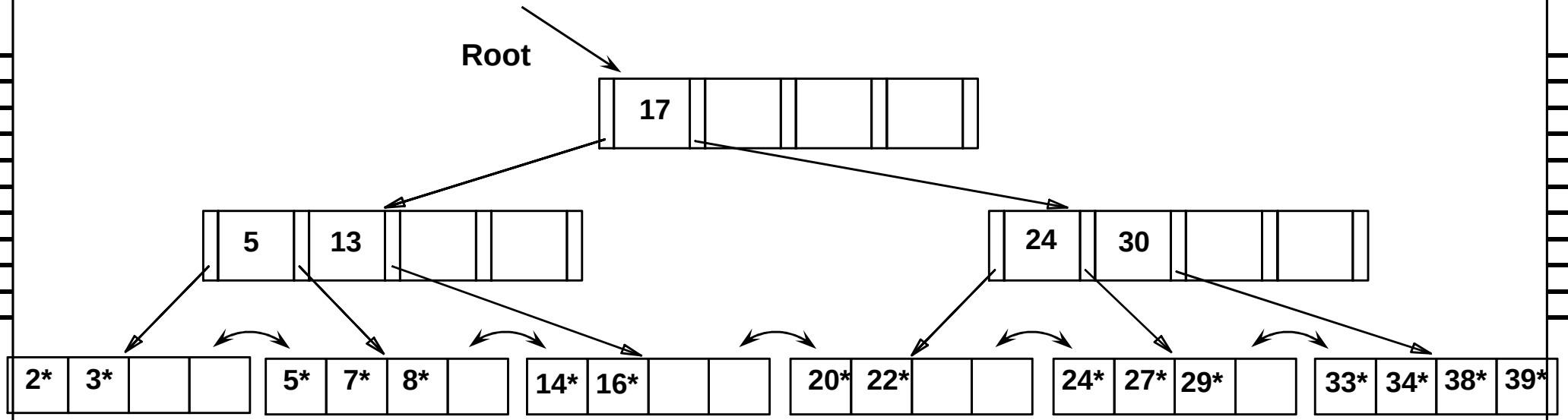
- Η PIZA διασπάστηκε οδηγώντας σε αύξηση του ύψους.

Η Διαγραφή μιας καταχώρησης δεδομένων (εγγραφής)

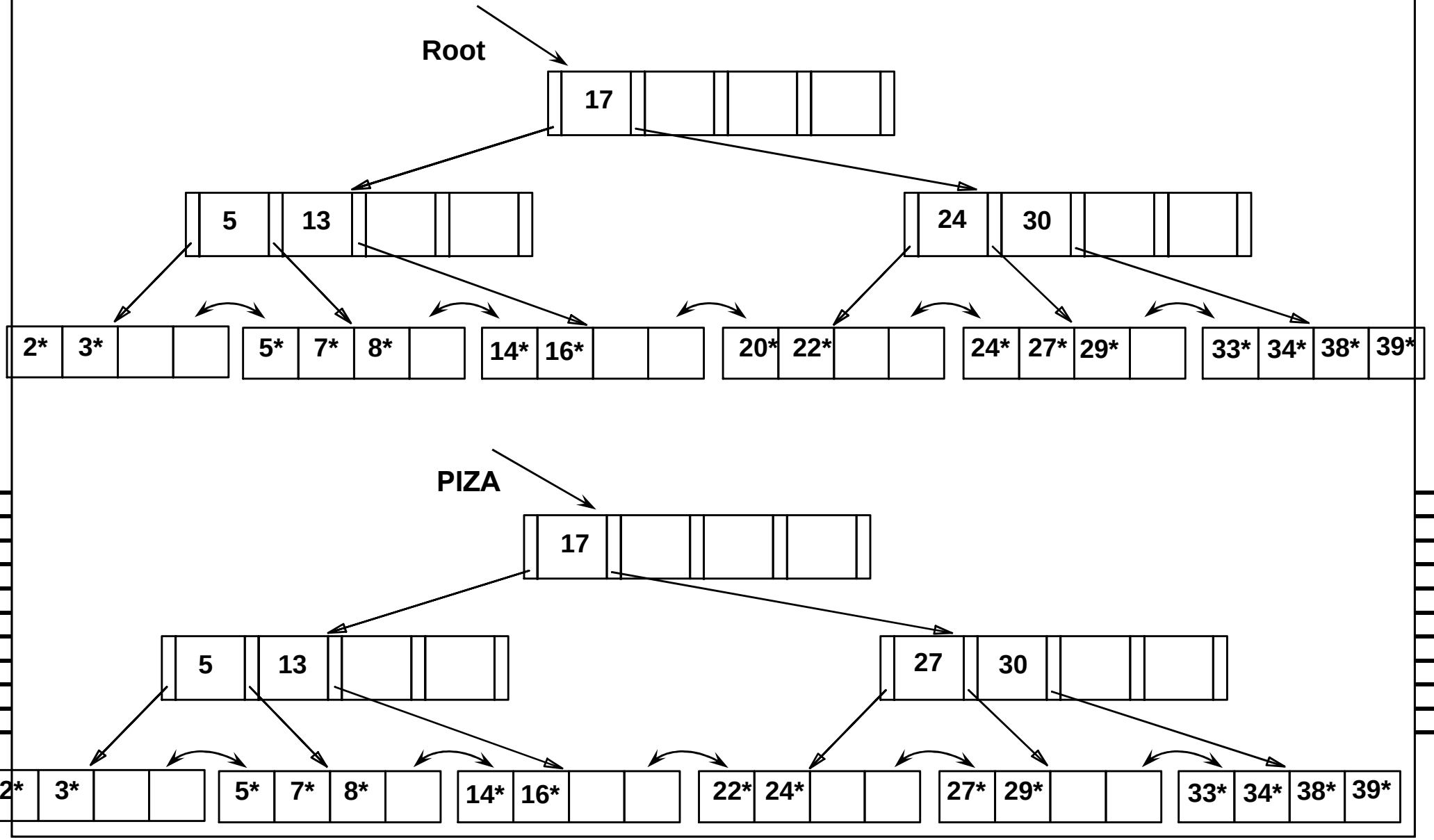
- Αρχίζοντας από τη Ρίζα, βρες το φύλλο *L* όπου ανήκει η καταχώρηση
- Διάγραψε την καταχώρηση.
  - Αν το L είναι τουλάχιστον μισό-γεμάτο, *τελείωσες!*
  - Αν το L έχει λιγότερες από τις μισές καταχωρήσεις,
    - » Προσπάθησε να κάνεις ανακατανομή με *γειτονικό κόμβο (κόμβο με τον ίδιο πατέρα του L)* και άλλαξε τα κλειδιά των προγόνων όπου χρειάζεται
    - » Αν η ανακατανομή αποτύχει, συγχώνευσε το *L* και τον γειτονικό κόμβο και αφαίρεσε το split key από τον πατέρα
- Για εσωτερικούς κόμβους «κατεβάζω» το split key του πατέρα
- Η συγχώνευση μπορεί να φτάσει στη Ρίζα, μειώνοντας το ύψος του Δέντρου.

Διαγραφή του 19*

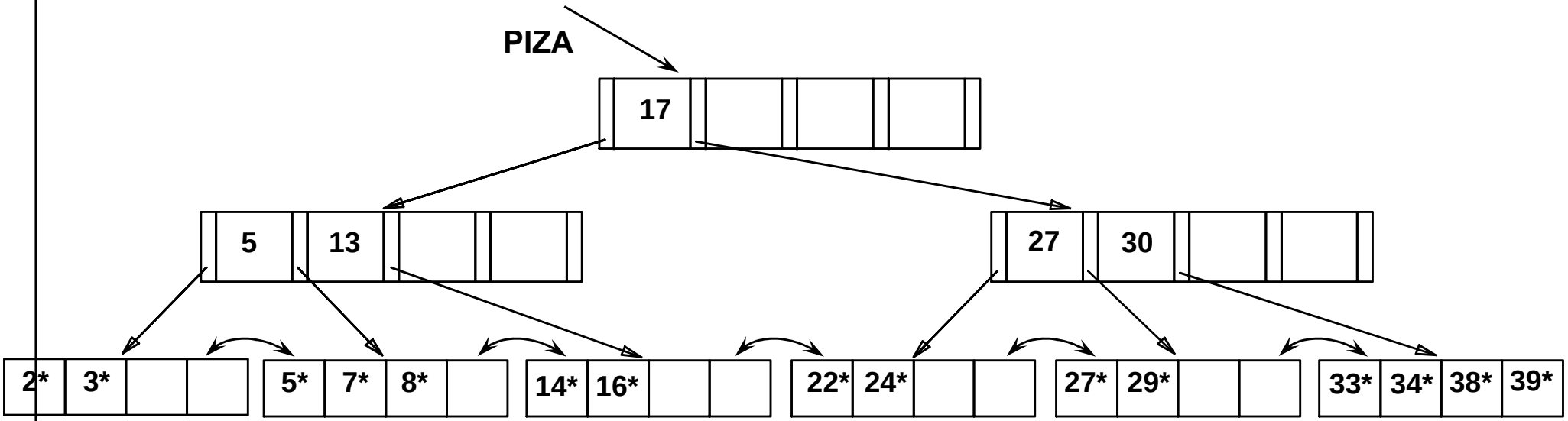




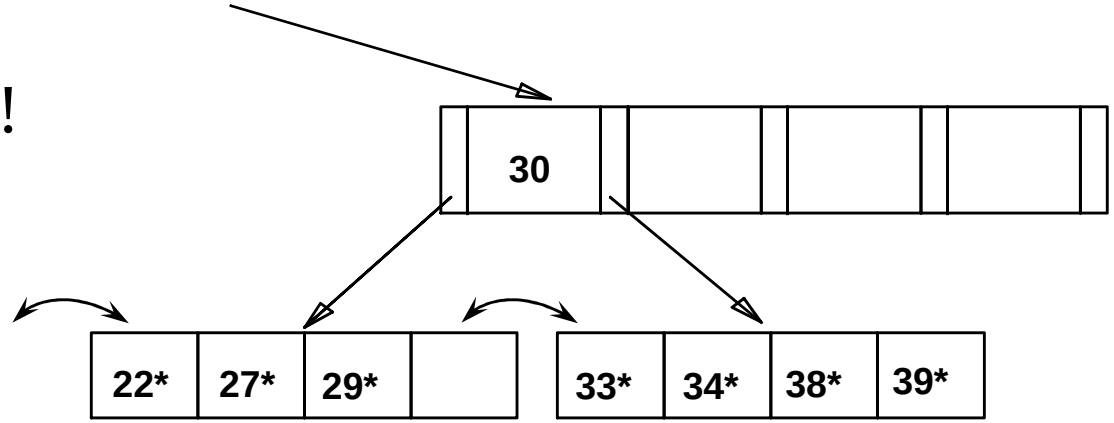
Διαγραφή του 20* - Ανακατανομή



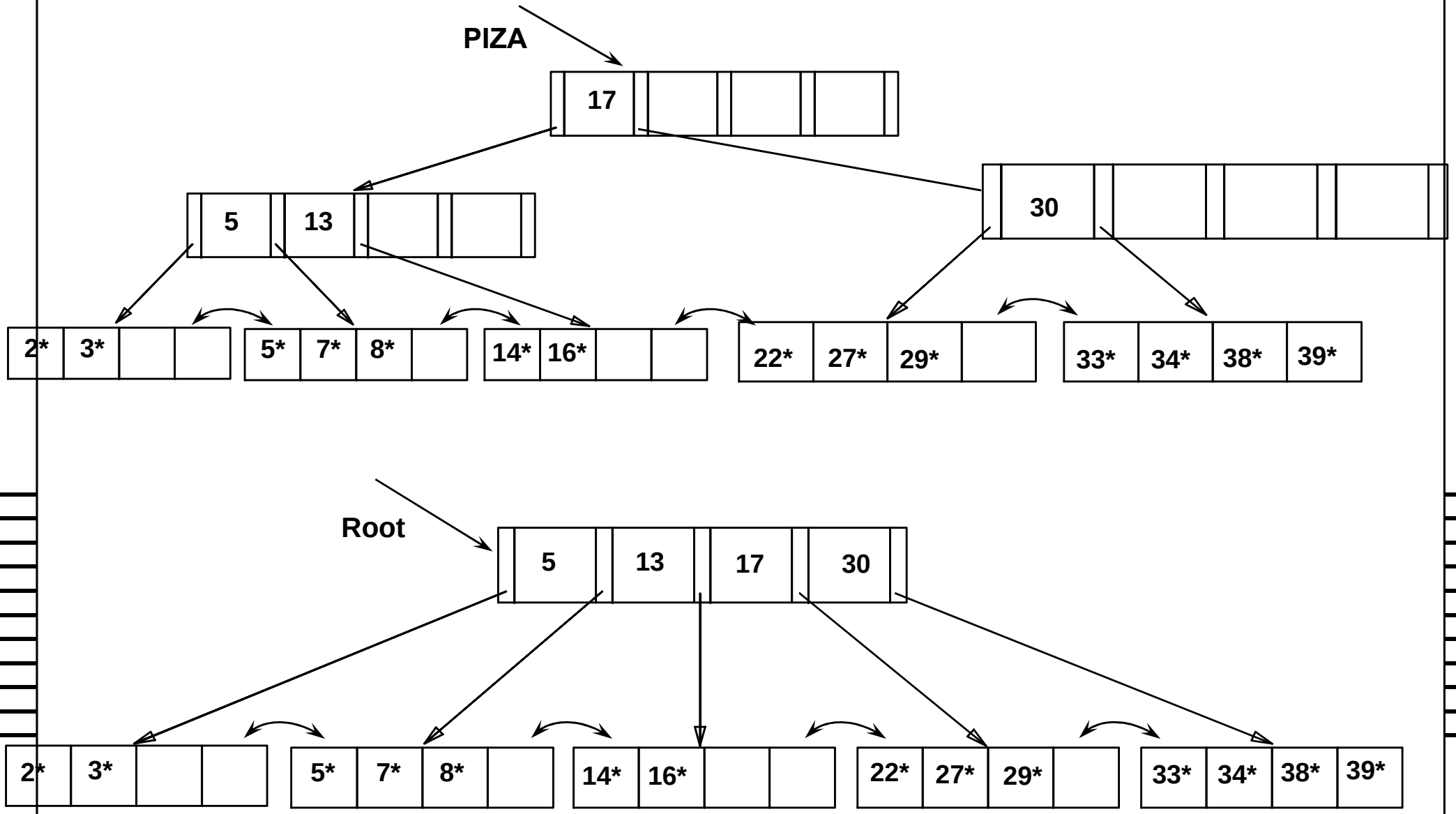
Διαγραφή του 24* (1)



- Απαιτείται ΣΥΓΧΩΝΕΥΣΗ!



Διαγραφή του 24* (2)



Δεικτοδότηση Strings

- Κλειδιά με strings μεταβλητού μεγέθους
  - Μεταβλητός αριθμός δεικτών ανά κόμβο
  - Χρησιμοποίηση χώρου ως κριτίριο για split
- **Prefix compression**
  - Στους εσωτερικούς κόμβους τα κλειδιά αναζήτησης μπορούν να είναι προθέματα (prefixes) του string
    - » Κρατάω αρκετούς χαρακτήρες για να διαχωρίζονται οι τιμές
      - E.g. “Silas” and “Silberschatz” can be separated by “Silb”
  - Τα κλειδιά στα φύλλα συμπιέζονται μοιραζόμενα κοινά προθέματα

Bulk Loading and Bottom-Up Build

- Αν έχουμε πολλές εγγραφές και θέλουμε να δημιουργήσουμε ένα B+ Δέντρο σε κάποιο γνώρισμα, χρησιμοποιώντας τις παραπάνω μεθόδους (εγγραφή προς εγγραφή) θα έχουμε μεγάλη καθυστέρηση
- *To Bulk Loading* είναι εξαιρετικά πιο αποδοτικό
- Efficient alternative 1:
  - Ταξινόμησε όλες τις καταχωρήσεις
  - Εισαγωγή με ταξινομημένη σειρά
    - » Εισαγωγή σε block ήδη στη μνήμη (or cause a split)
    - » Καλύτερο I/O
- Efficient alternative 2: **Bottom-up B^+-tree construction**
  - Ταξινόμησε όλες τις καταχωρήσεις
  - Φτιάξε το δέντρο layer-by-layer, ξεκινώντας από τα φύλλα

Σύνοψη του Bulk Loading

- Εναλλακτικός Τρόπος 1: Πολλαπλές Εισαγωγές
  - Αργός
  - Δεν καταλήγει σε σειριακή τοποθέτηση των φύλλων
- Εναλλακτικός Τρόπος 1 : *Bulk Loading*
  - Πλεονεκτήματα στη Λειτουργία (π.χ., έλεγχο συνδρομικότητας)
  - Μικρότερος αριθμός από I/Os κατά την διάρκεια εισαγωγής
  - Σειριακή τοποθέτηση των φύλλων στο Δίσκο
  - Ελέγχει καλύτερα τον παράγοντα πληρότητας

Multiple-Key Access

- Χρήση πολλαπλών δεικτών για κάποια queries.
- Example:

  select *ID*

  from *instructor*

  where *dept_name* = “Finance” **and** *salary* = 80000
- Πιθανές στρατηγικές:
  1. Ευρετήριο για *dept_name* για να βρούμε καθηγητές στο Finance; Διαλέγω εγγραφές με *salary = 80000*
  2. Ευρετήριο για *salary* για να βρούμε καθηγητές με μισθό $80000; Διαλέγω εγγραφές με *dept_name* = “Finance”.
  3. Ευρετήριο *dept_name* για να βρω καθηγητές στο “Finance”. Ευρετήριο *salary* για να βρούμε καθηγητές με μισθό $80000. Παίρνουμε την τομή

Δείκτες σε πολλαπλά κλειδιά

- **Composite search keys** are search keys containing more than one attribute
  - E.g. (*dept_name, salary*)
- Lexicographic ordering: $(a_1, a_2) < (b_1, b_2)$ if either
  - $a_1 < b_1$, or
  - $a_1 = b_1$ and $a_2 < b_2$

Δείκτες σε πολλαπλά κλειδιά

Αν έχουμε ευρετήριο στο
(*dept_name, salary*).

- With the **where** clause
  where *dept_name* = “Finance” **and** *salary* = 80000
  τότε το ευρετήριο (*dept_name, salary*) μας δίνει τις εγγραφές που ικανοποιούν και τα 2 κριτήρια
  - Using separate indices in less efficient — we may fetch many records (or pointers) that satisfy only one of the conditions.
- Can also efficiently handle
  where *dept_name* = “Finance” **and** *salary* < 80000
- But cannot efficiently handle
  where *dept_name* < “Finance” **and** *balance* = 80000
  - May fetch many records that satisfy the first but not the second condition