# Curs SDA 5:
# Structuri de date avansate pentru multimi
## B-Trees. Structuri de date pentru multimi disjuncte

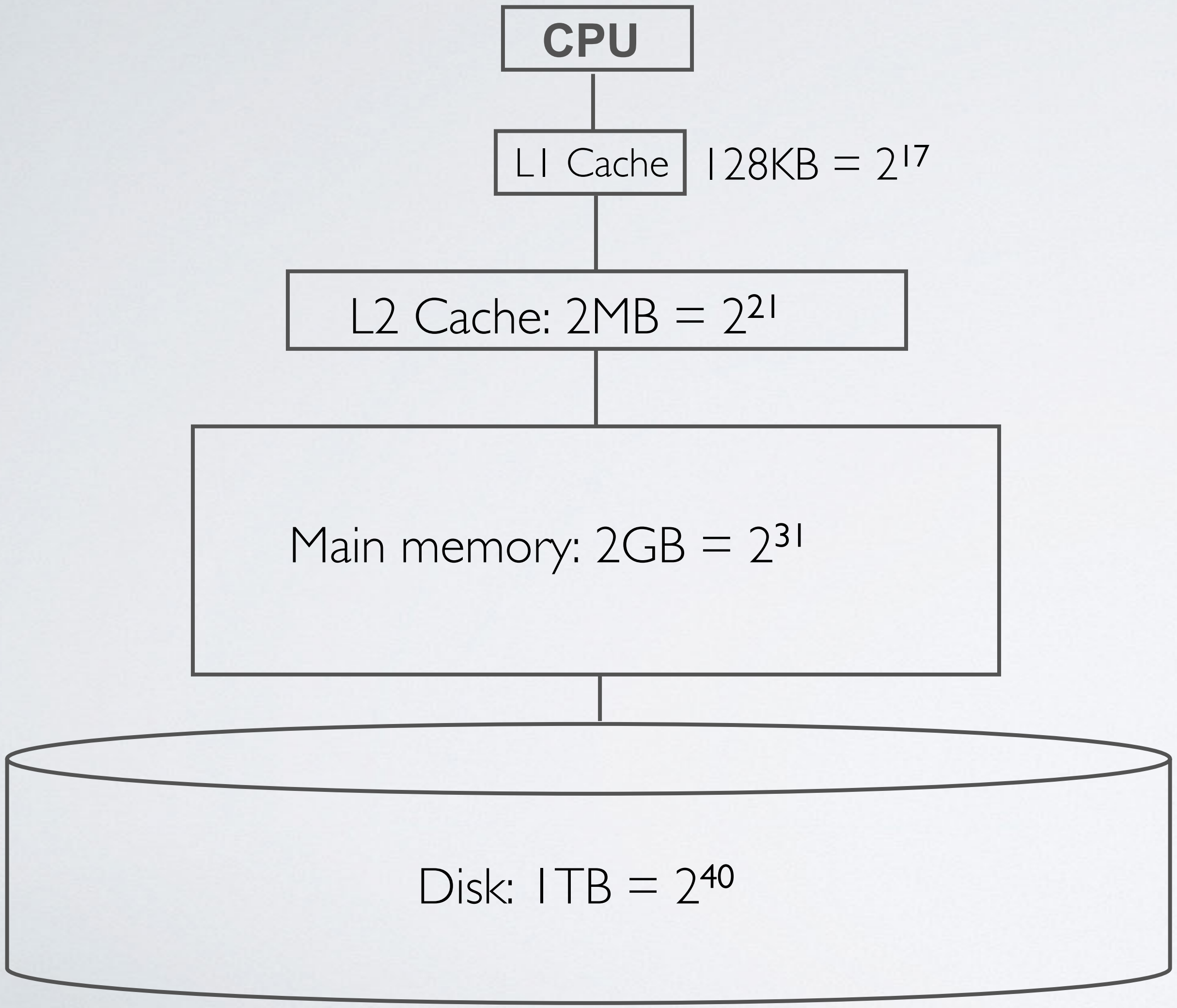
# DIN CURSUL TRECUT … ARBORI BINARI DE CAUTARE ECHILIBRATI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- daca se utilizeaza frecvent operatii bazate pe ordinea sortata a elementelor -> se prefera un ABC echilibrat
- … dar daca avem nevoie sa stocam un dictionar foarte mare (1GB = $2^{30}$ bytes)?
  - Problema este ca nu mai putem presupune *"fiecare operatie de acces la memorie (read/write) ia O(1)"*, pentru ca ...

# IERARHIA DE MEMORIE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA




instructiuni, operanzii in registri (e.g., `add`, `mov`, etc): $2^{30}$/sec

aducerea datelor in L1: $2^{29}$/sec

aducerea datelor in L2: $2^{25}$/sec

aducerea datelor in memoria principala: $2^{22}$/sec

aducerea datelor din locatie aleatoare de pe disk: $2^{7}$/sec

"streamed": $2^{18}$/sec

https://courses.cs.washington.edu/courses/cse332/10sp/lectures/lecture8.pdf

# MORALA…

- Este mult mai rapid sa executi … decat:

| | |
|---|---|
| 5 mil. op. aritmetice | 1 op. de acces la disc |
| 2500 op. de acces la L2 cache | 1 op. de acces la disc |
| 400 op. de acces la memoria principala | 1 op. de acces la disc |

- De ce se utilizeaza modelul acesta de ierarhie?
  - Compromis dimensiune/viteza/cost
    - "You can have either two ☺"
  - HDD-urile devin mai mari, nu mai rapide
  - Cresterea vitezei la nivele superioare -> cele inferioare *par* (relativ) mai incete

# DE REGULA – "FORGET ABOUT IT"

- Pentru ca …
- Hardware-ul muta automat datele in memoria *cache* din *memoria principala*
  - Inlocuind ceea ce e deja acolo
  - Algoritmii sunt mai rapizi in realitate daca datele "incap" in cache (de regula incap)
- Accesul la *disc* se face prin intermediul sistemului de operare
- Deci, de regula nu trebuie sa consideram ierarhia de memorie explicit cand proiectam algoritmi si structuri de date
- Cu cateva exceptii, cand datele nu incap in memoria principala (disc ->) / cache (memoria principala ->)
  - LATENTA!
    - Prin urmare, se trimit mai multe date decat doar "referinta" ceruta, pentru ca …
      - E usor
      - Probabil sa fie utilizata in curand (e.g. vectori, campuri din structura) ~ principiul localitatii

# LEGATURA CU STRUCTURILE DE DATE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Datele sunt mutate intre nivelele ierarhiei in “grup”:
  - Disc ↔ memorie: *bloc (pagina)*
  - Memorie principala ↔ cache: *linie*
  - *Dimensiunea nu este sub controlul programului!*
- Structurile de date
  - Implementarile secventiale (vector) exploateaza acest lucru, cele inlantuite nu
- Pp. coada cu $2^{23}$ obiecte, fiecare $2^7$ bytes (pe disc), dimensiune bloc = $2^{10}$ bytes
  - Implementarea secventiala – $2^{20}$ accese la disc
    - daca e “streamed” perfect: > 4 secunde – $2^{18}$/sec
    - la locatii aleatoare pe disc: > 8000 sec (> 2h) – $2^7$/sec
  - Implementare inlantuita, caz defavorabil – $2^{23}$ accese aleatoare la disc (>16h) – $2^7$/sec
- Nota: vector nu e neaparat “bine” (e.g. heap, unde acesele se fac in “salt”, nu liniar)

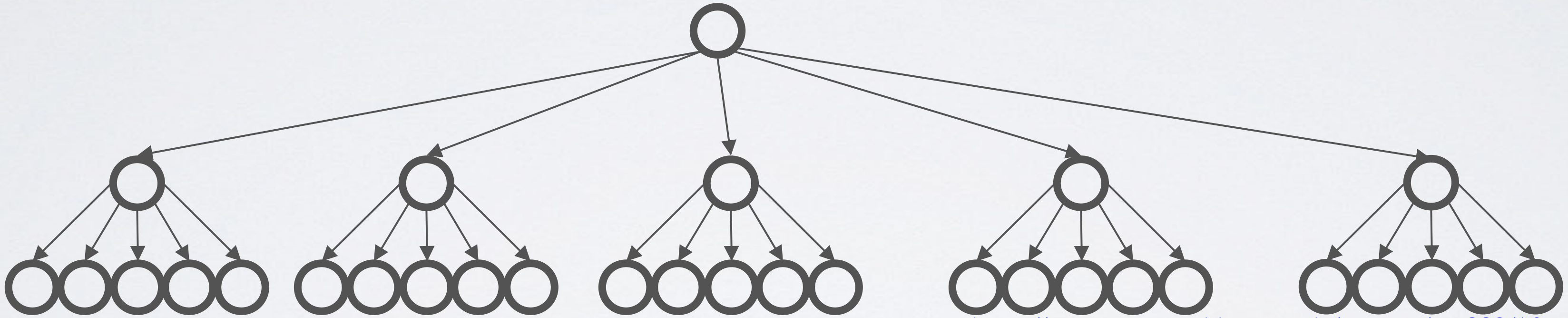
# REVENIND LA ARBORI….

- AVL de inaltime = 40, dimensiunea unui nod = b Bytes => minim ~ 240215253 noduri => ~ 230 * b MB (e.g. b=4 => dimensiune arbore > 1GB)
- Ce urmarim …
  - Arbore de cautare echilibrat
  - Inaltime mai mica decat a AVL, pentru a minimiza nr. de accese la disc
  - Sa exploatam dimensiunea blocului (unitatea de transfer disc ↔ memorie)
- Idee
  - Crestem factorul de ramificare (en. *branching factor*)

# B-TREES: ARBORE DE CAUTARE MULTICAI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cerinte
  - Vector de M copii, ***sortat***, la fiecare nod (Node* [])
  - Sa alegem M astfel incat vectorul sa incapa intr-un bloc de disc (1 acces)



https://courses.cs.washington.edu/courses/cse332/10sp/lectures/lecture8.pdf

  - Arbore perfect de inaltime h va avea $(M^{h+1}-1)/(M-1)$ noduri
  - Numarul de accese la cautare, daca e echilibrat?
    - $\log_M n$
    - E.g. $M=256$, $n=2^{40}$ (5 accese in loc de 40)
  - Complexitatea op. cautare, daca e echilibrat? (revenim la asta putin mai incolo)

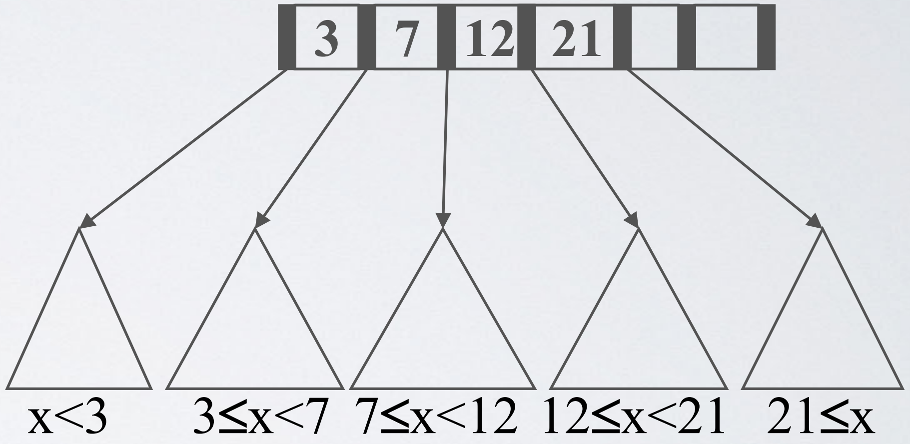
# B-TREES

1. Cum ordonam cheile?
2. Datele utile nu sunt utilizate la cautare (doar cheile!)
3. Cum cautam?

1. Relatia de ordine: sub-arborele aflat intre cheile x si y contine doar chei $\geq x$ si $< y$
2. Datele se stocheaza doar la frunze (in exemple vom ignora datele – reprezentam doar cheile coresp.)
3. *Nod intern*: cautare binara pe cele $\leq$ M-1 chei;
4. *Frunza*: cautare binara pe cele $\leq$ L obiecte

   Complexitate?

   $O(\log_2 M \log_M n)$




https://courses.cs.washington.edu/courses/cse332/10sp/lectures/lecture8.pdf

# B-TREES

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Timp logaritmic *doar daca avem o* conditie de echilibru
- Structura B-TREE

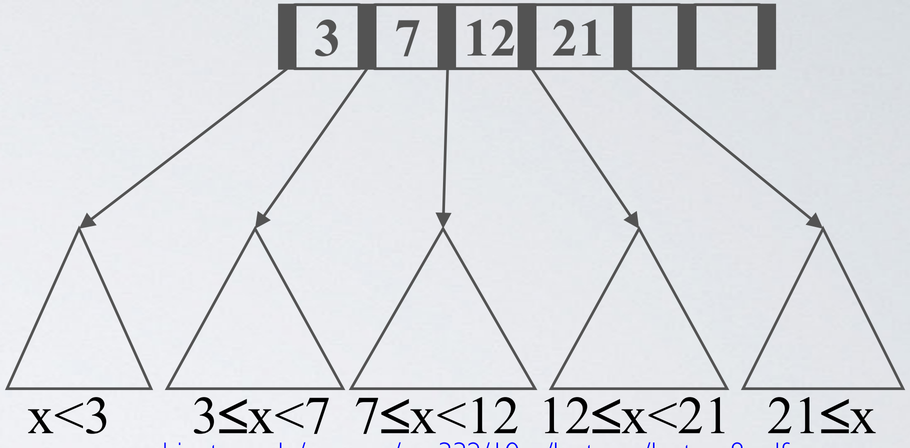
  1. Radacina:

  Daca arborele are mai putin de L elemente atunci radacina este frunza; (caz special)

  Altfel, radacina este nod intern si are intre 2 si M copii

  2. Noduri interne ("noduri indicatoare"):
     - intre $\left\lceil \frac{M}{2} \right\rceil$ si $M$ copii (sunt cel putin pe jumatate pline)

  3. Frunze ("noduri de date"):
     - Toate frunzele la aceeasi adancime
     - intre $\left\lceil \frac{L}{2} \right\rceil$ si $L$ itemi de date (cel putin pe jumatate pline)
- Reiese relativ usor ca: $n \geq \underbrace{2\lceil M/2 \rceil^{h-1}}_{\text{Nr. minim de frunze}} \underbrace{\lceil L/2 \rceil}_{\text{Nr. minim de itemi de date pe frunza}}$
- n – nr total de itemi de date




https://courses.cs.washington.edu/courses/cse332/10sp/lectures/lecture8.pdf

M=7; fiecare nod intern are intre 4 si 7 copii (3 si 6 chei)

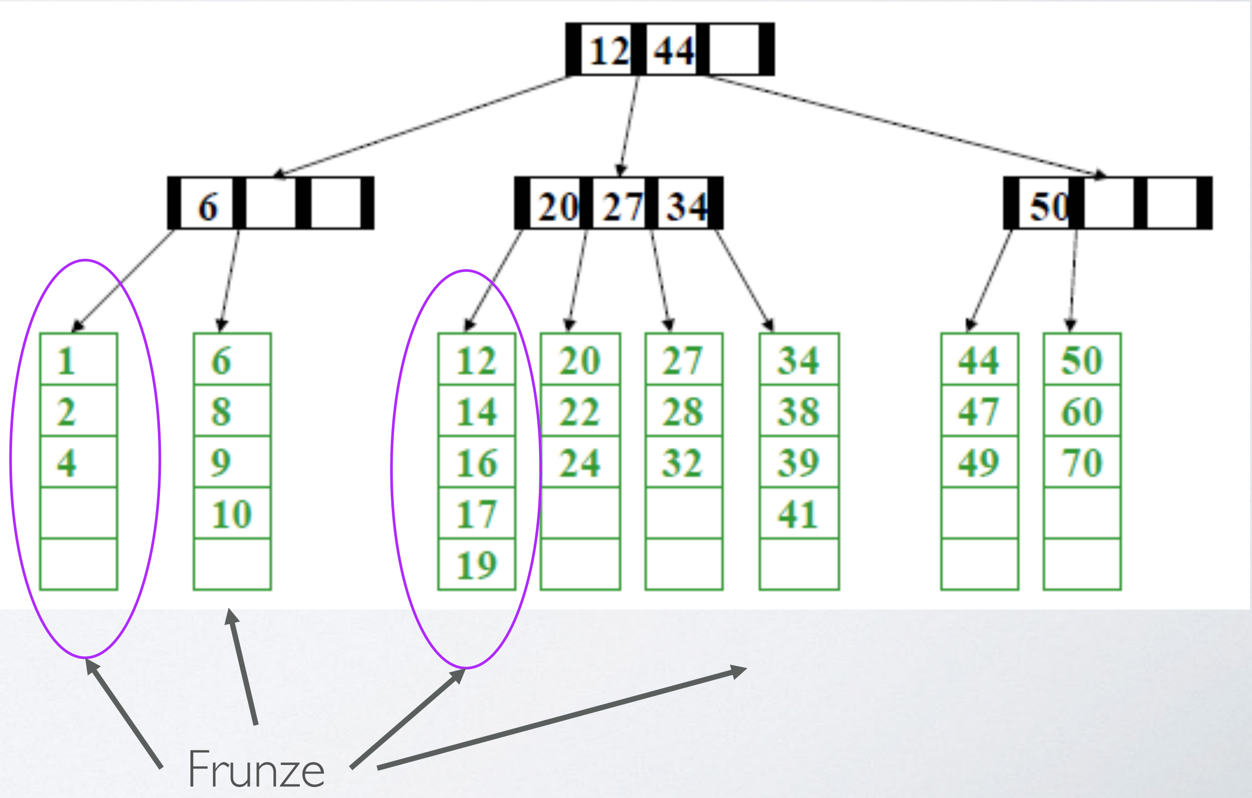
# B-TREE EXEMPLU

Se da M = 4 – numar maxim de copii

Se da L = 5 – numar maxim de itemi intr-o frunza

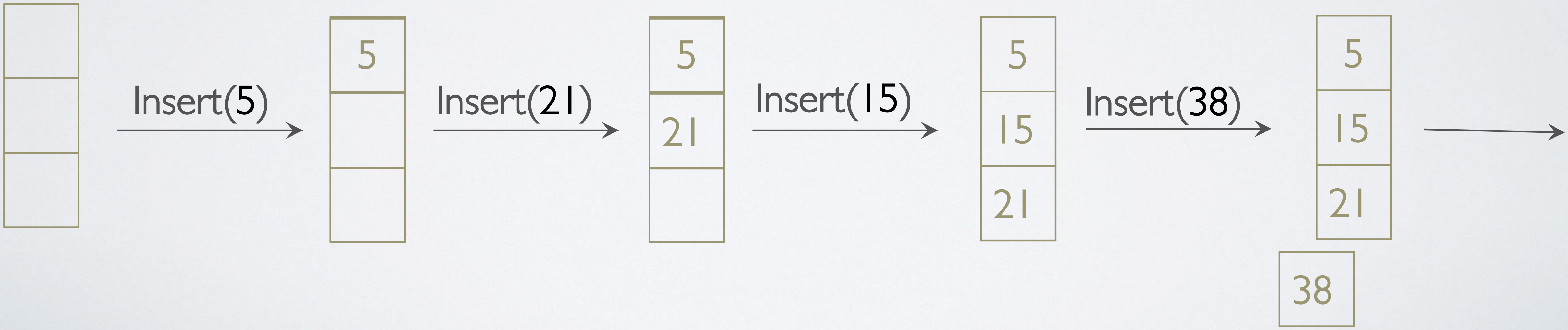
Rezulta:

- Toate nodurile interne au intre 2 si 4 copii
- Toate frunzele au intre 3 si 5 itemi.
- Toate frunzele sunt la aceeasi inaltime



# B-TREES – INSERARE

- Cum inseram?
- Ce probleme pot aparea la inserare?
  - *Overflow* la frunza corespunzatoare (L elemente)
- E.g.: M=3 => nodurile interne au intre $\left\lceil \frac{3}{2} \right\rceil$ = 2 si 3 copii
- L=3 => cate obiecte / itemi au frunzele ? (intre 2 si 3)

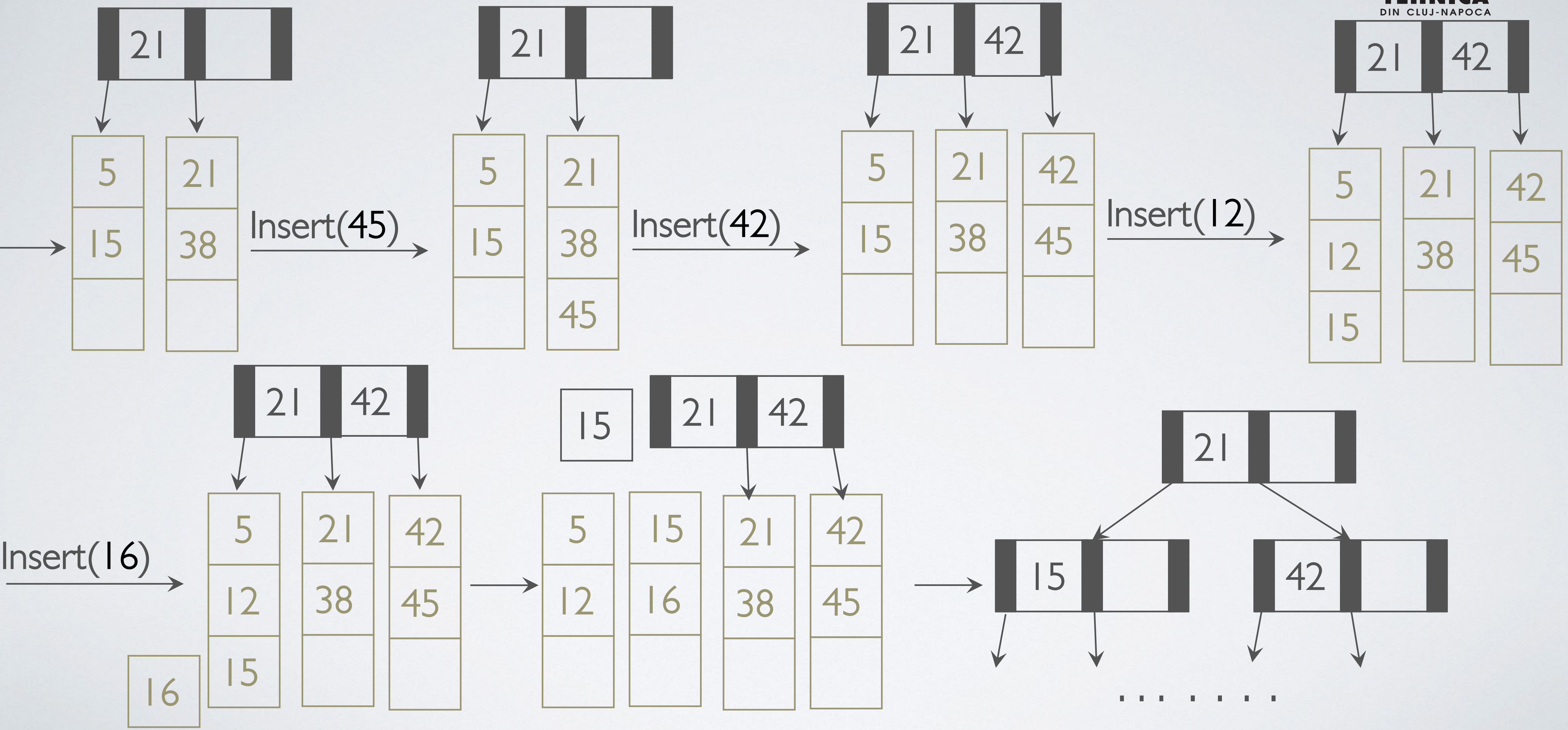



https://courses.cs.washington.edu/courses/cse332/10sp/lectures/lecture8.pdf

M=3, L=3

# B-TREES – INSERARE (cont'd)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA



# B-TREES – ALGORITM INSERARE

1. Insereaza elementul in frunza corespunzatoare, in ordine
2. Daca acum nodul frunza are L+1 elemente, *overflow*!
   a. Se divide nodul frunza in 2 noduri
      - Frunza initiala ramane cu cele $\lceil (L+1)/2 \rceil$ elemente mai mici
      - Noua frunza cu $\lfloor (L+1)/2 \rfloor = \lceil L/2 \rceil$ elemente mai mari
   b. Se adauga cheia noua (mediana) in parinte, in ordine
3. Daca pasul 2 a cauzat *overflow* in parinte (M+1 copii)
   a. Se divide nodul intern in 2 noduri
      - Nodul initial ramane cu $\lceil (M+1)/2 \rceil$ chei mai mici
      - Noul nod cu $\lfloor (M+1)/2 \rfloor = \lceil M/2 \rceil$ elemente mai mari
   b. Se adauga cheia noua (mediana) in parinte, in ordine
4. Daca pasul 3 a cauzat *overflow* in parinte, se repeta pasul 3 la parinte(i), pana nu mai avem *overflow*
   - Daca avem *overflow* la radacina, creem o radacina noua, cu 2 copii (doar asa creste inaltimea!)

# B-TREES – EFICIENTA ALGORITM INSERARE

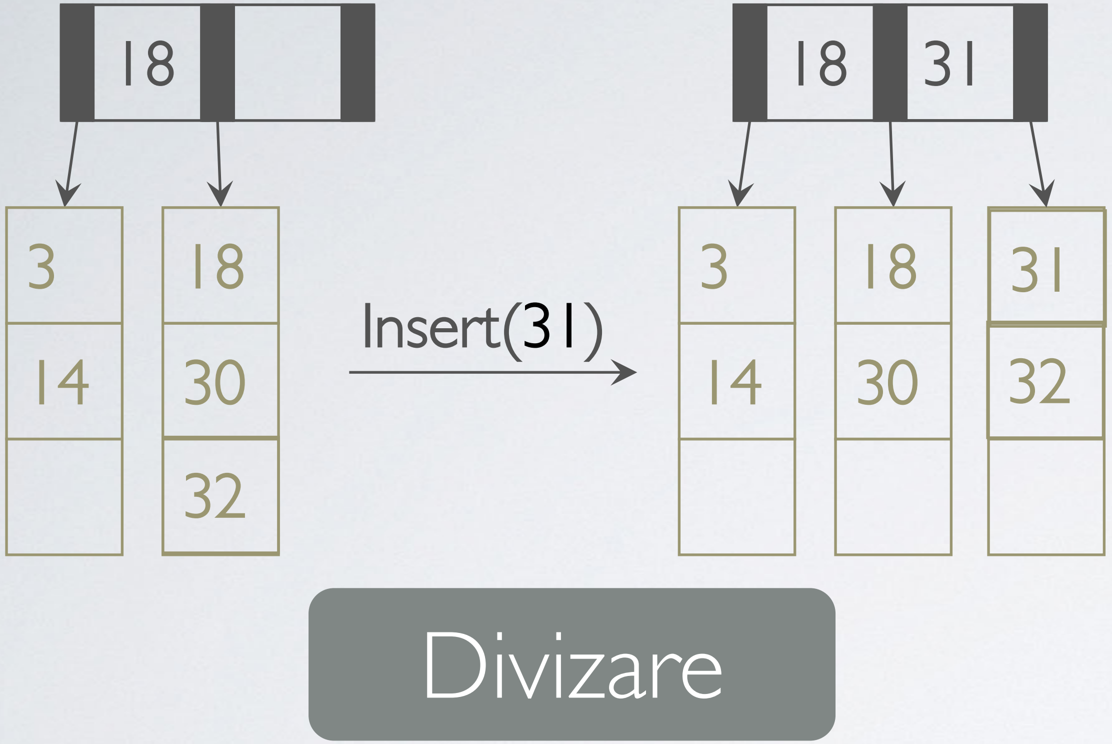
1. Gasirea frunzei corecte: $O(\log_2 M \log_M n)$
2. Inserare in frunza: $O(L)$
3. Divizarea frunzei: $O(L)$
4. Divizarea nodurilor parinte, pana la radacina: $O(M \log_M n)$

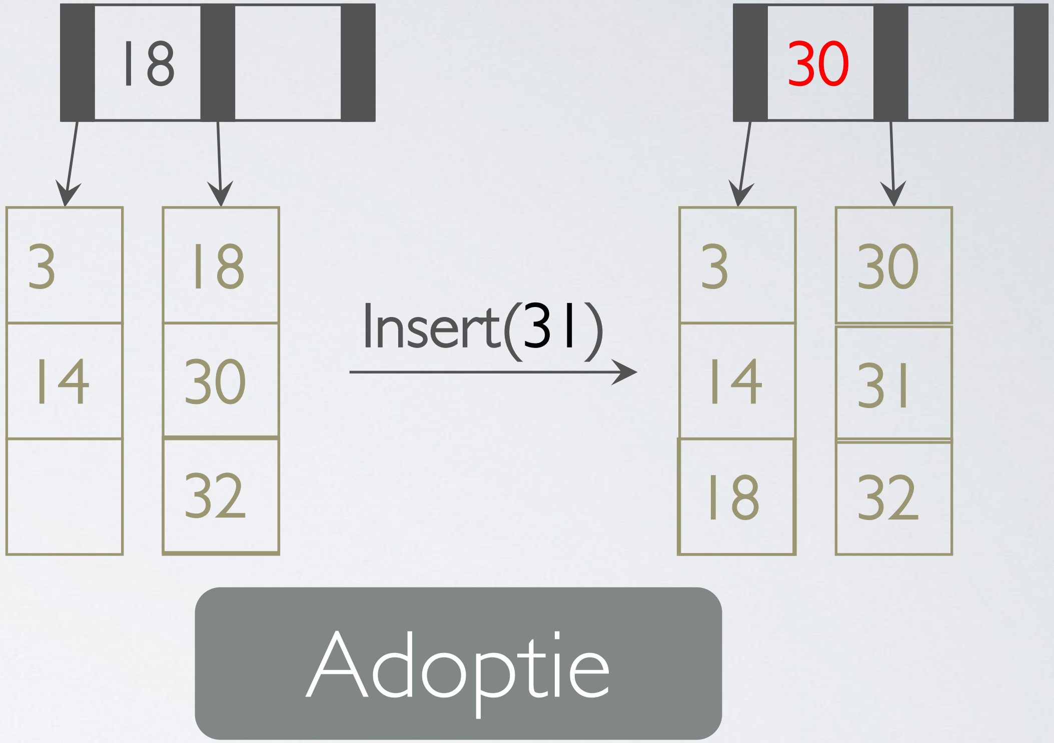
Total: $O(L + M \log_M n)$

DAR, e chiar bine, pentru ca:

- Divizarile sunt rare (M si L de regula sunt mari, dupa o divizare, noduri pe jumatate pline)
- Divizarea radacinii, prin urmare, este extrem de rara
- Reducerea nr. de accese la disc este prioritatea principala: inserarea ia deci $O(\log_M n)$ in medie

# B-TREES – INSERARE CU ADOPTIE



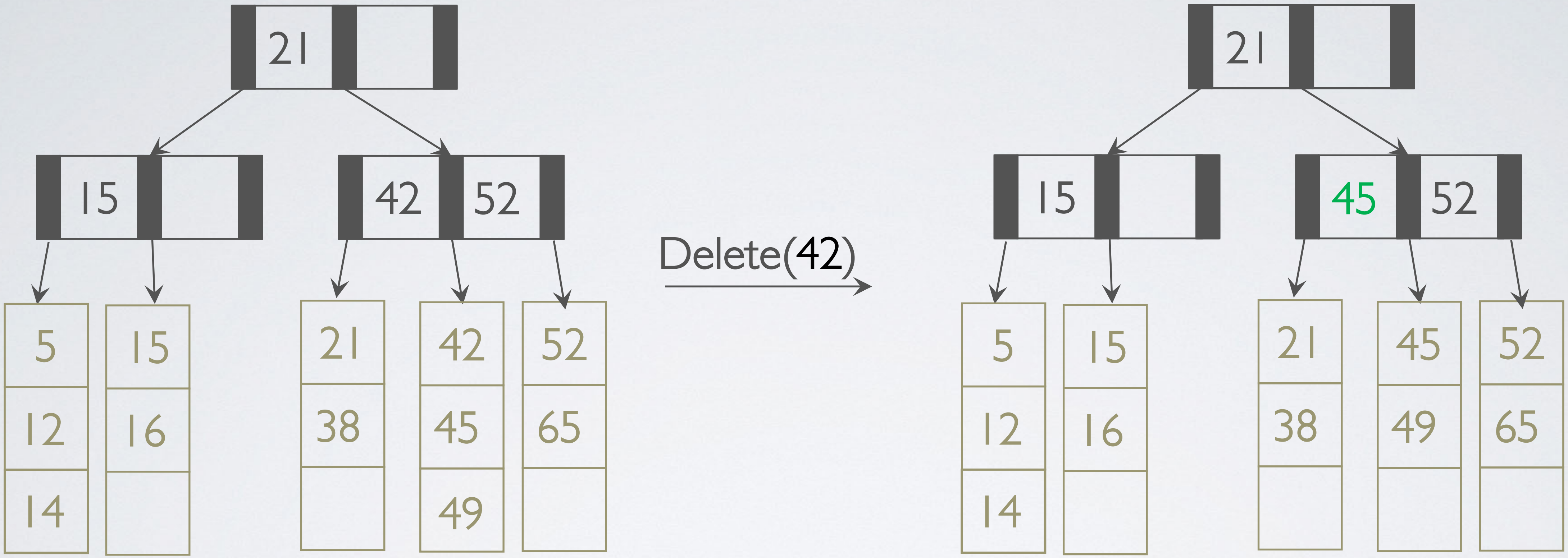




- Divizarea se poate uneori evita, prin *adoptie*
- Aceasta idee se aplica si la stergere, dar invers

M=3, L=3

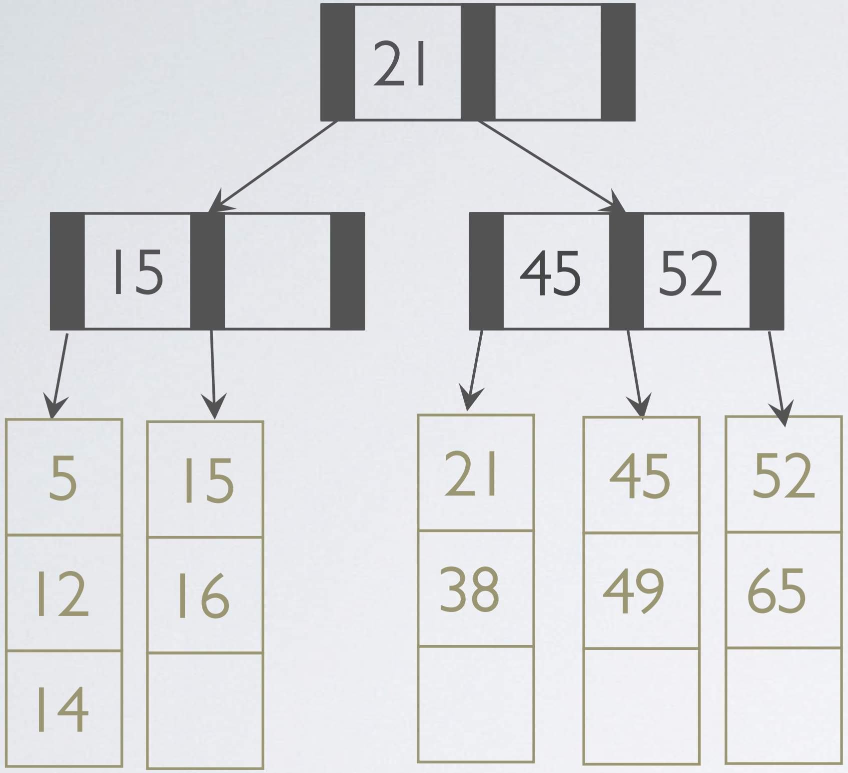
# B-TREES – STERGERE



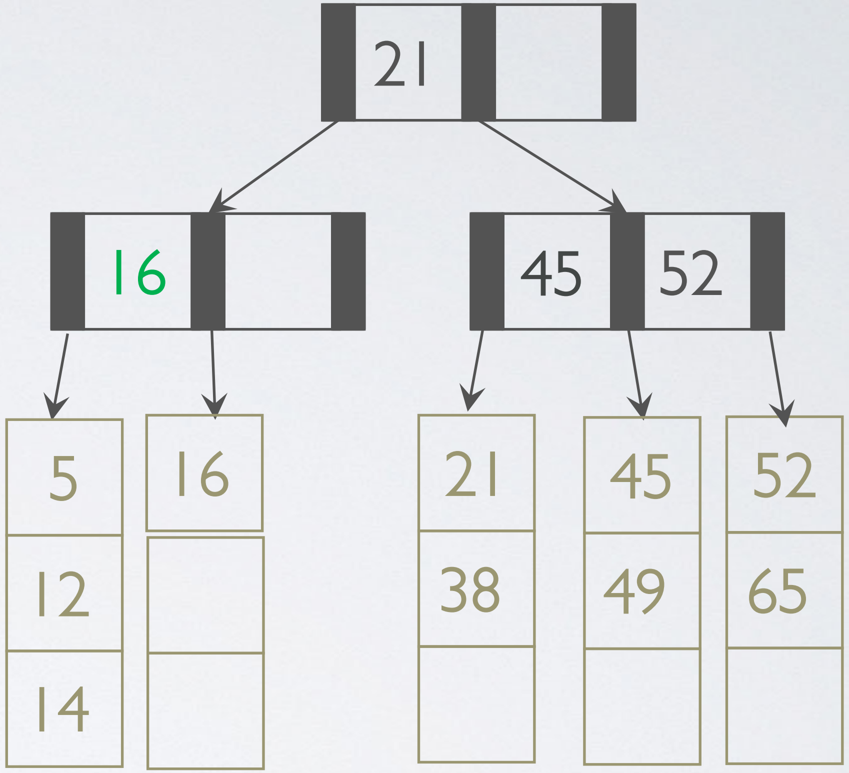


# B-TREES – STERGERE

M=3, L=3



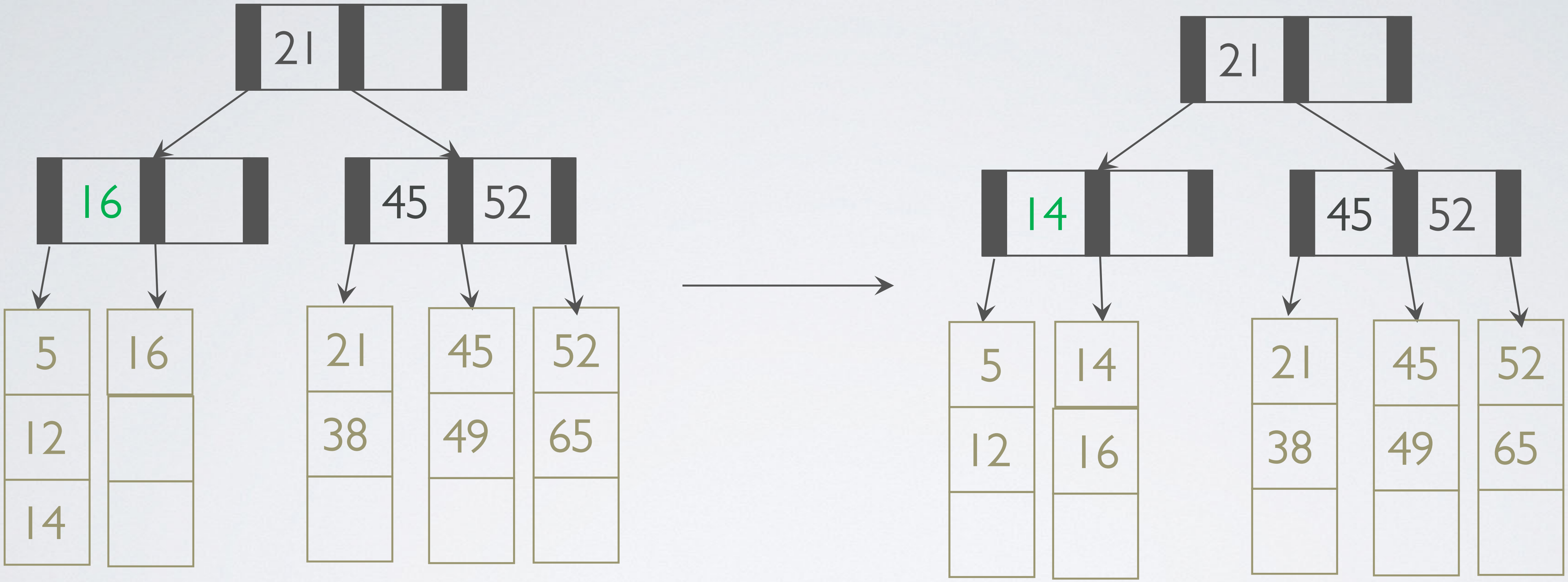

Delete(15)




Underflow la frunza! Adopta!

# B-TREES – STERGERE

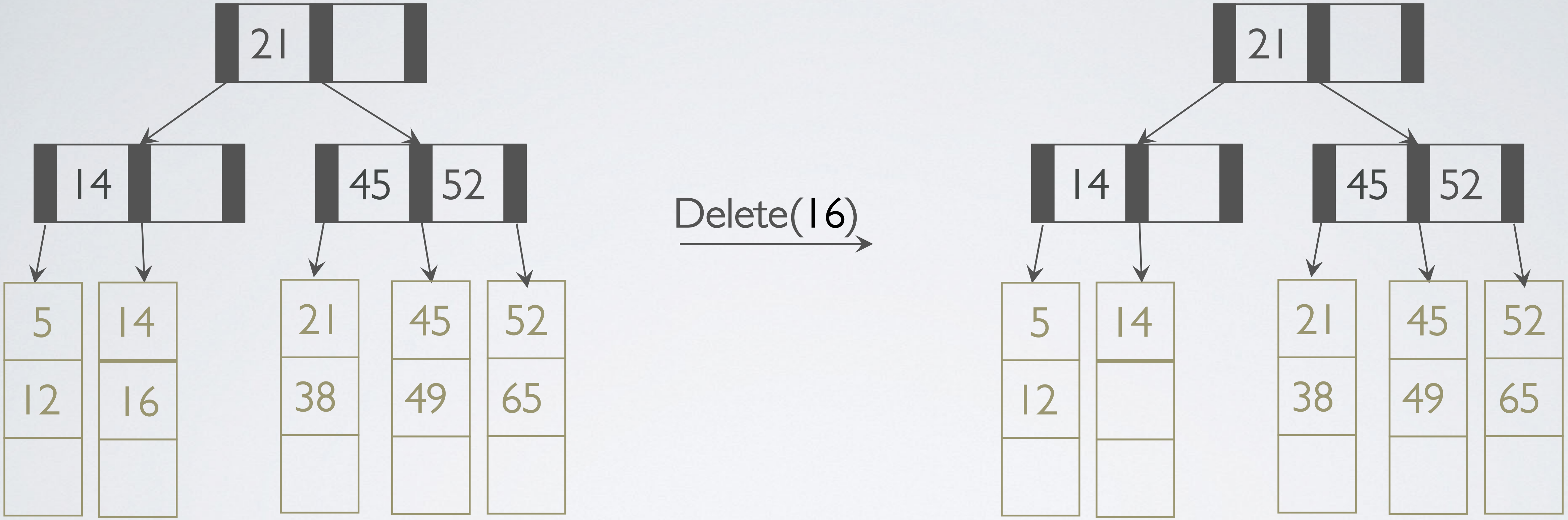
M=3, L=3




Underflow la frunza! Adopta!

M=3, L=3

# B-TREES – STERGERE

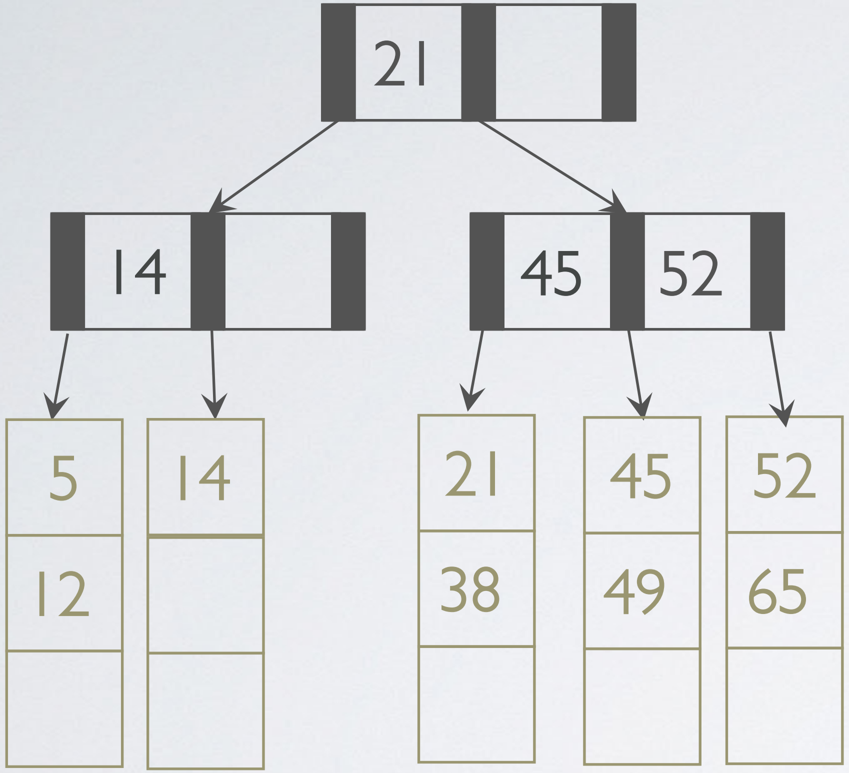



Underflow la frunza! Vecinii la minim!

M=3, L=3

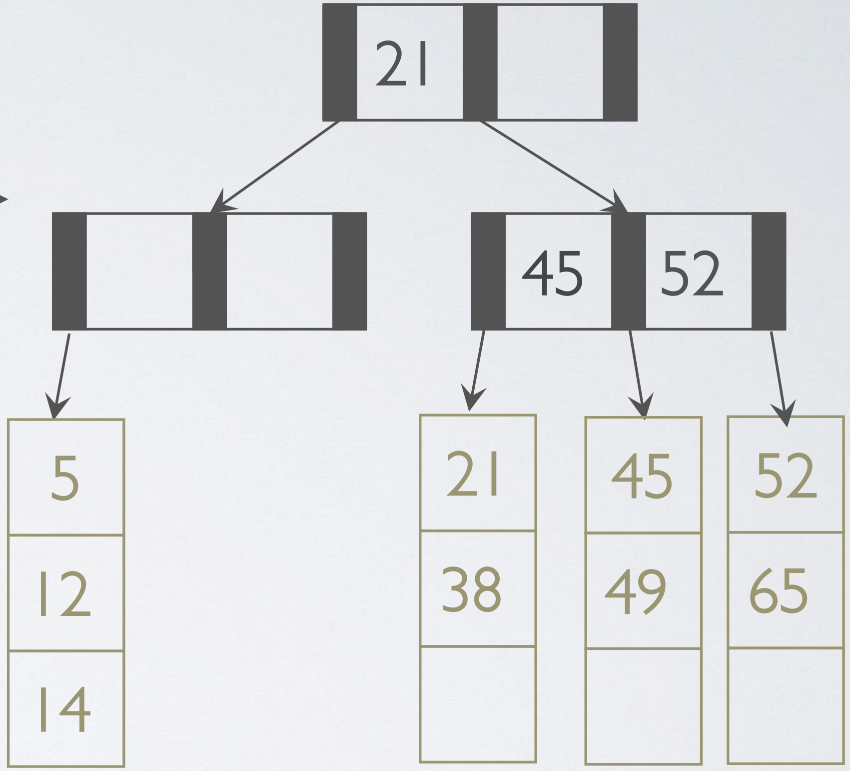
# B-TREES – STERGERE






Nodurile se unesc

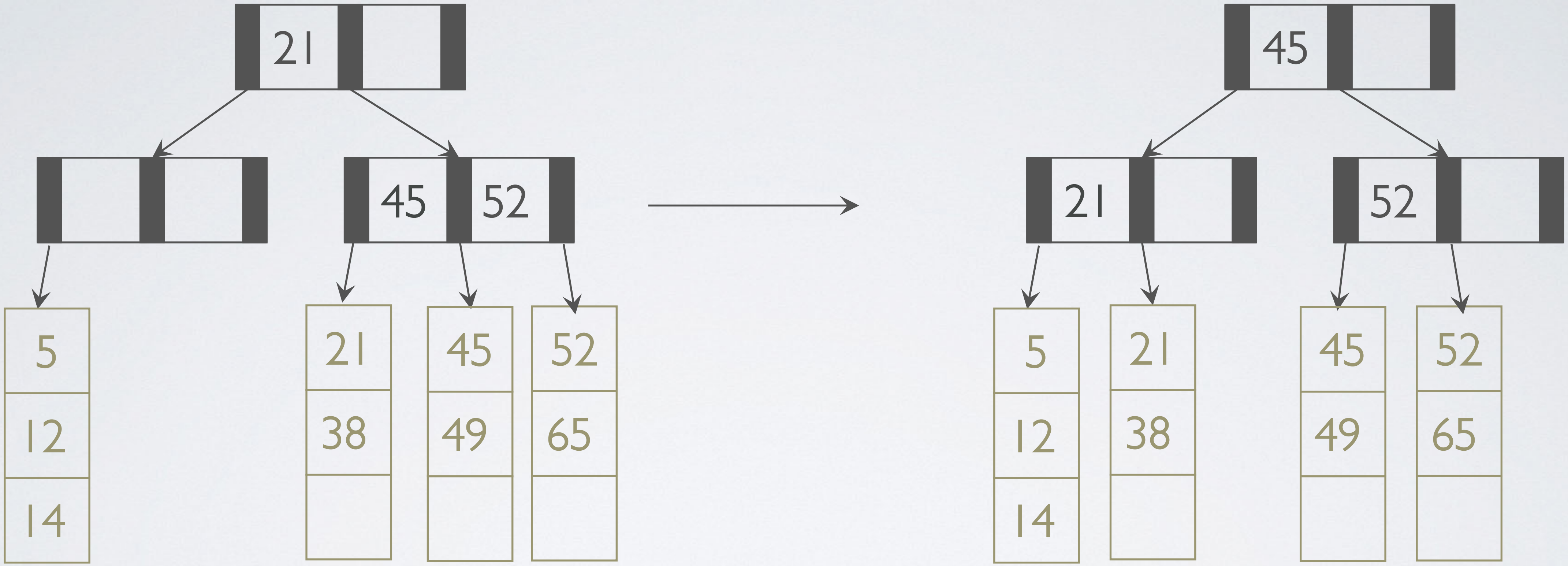
Se elimina o frunza




Acum putem avea underflow la parinte; are vecini

M=3, L=3

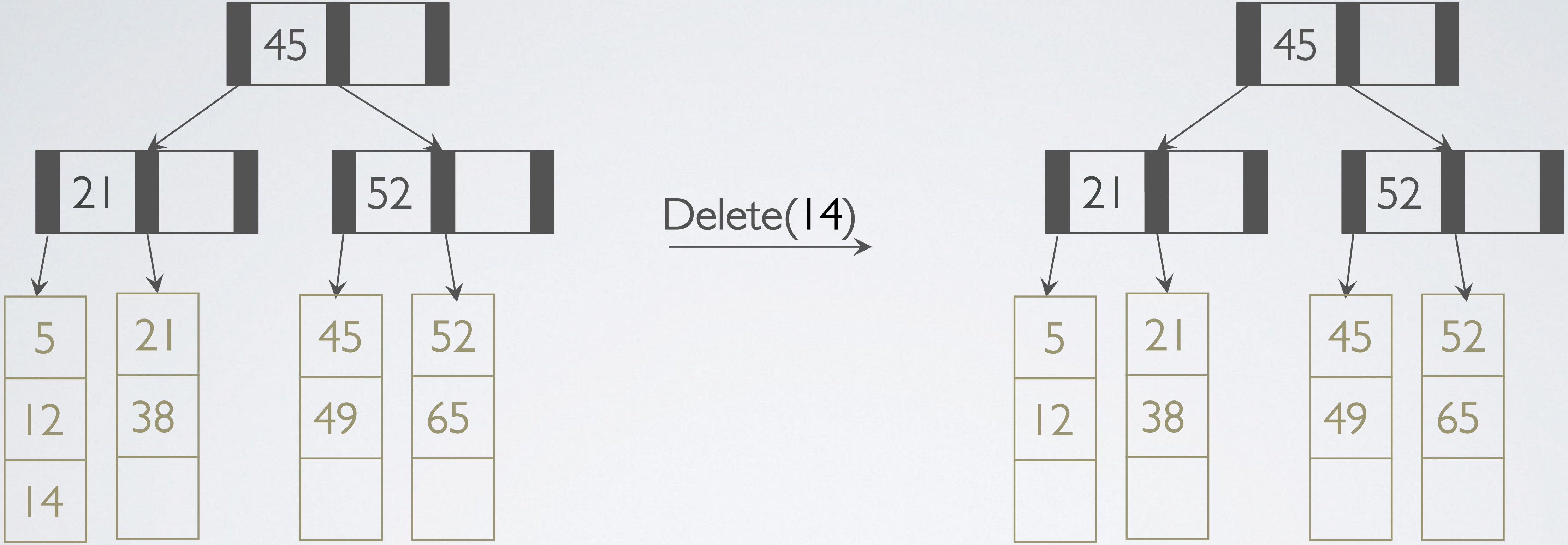
# B-TREES – STERGERE



M=3, L=3

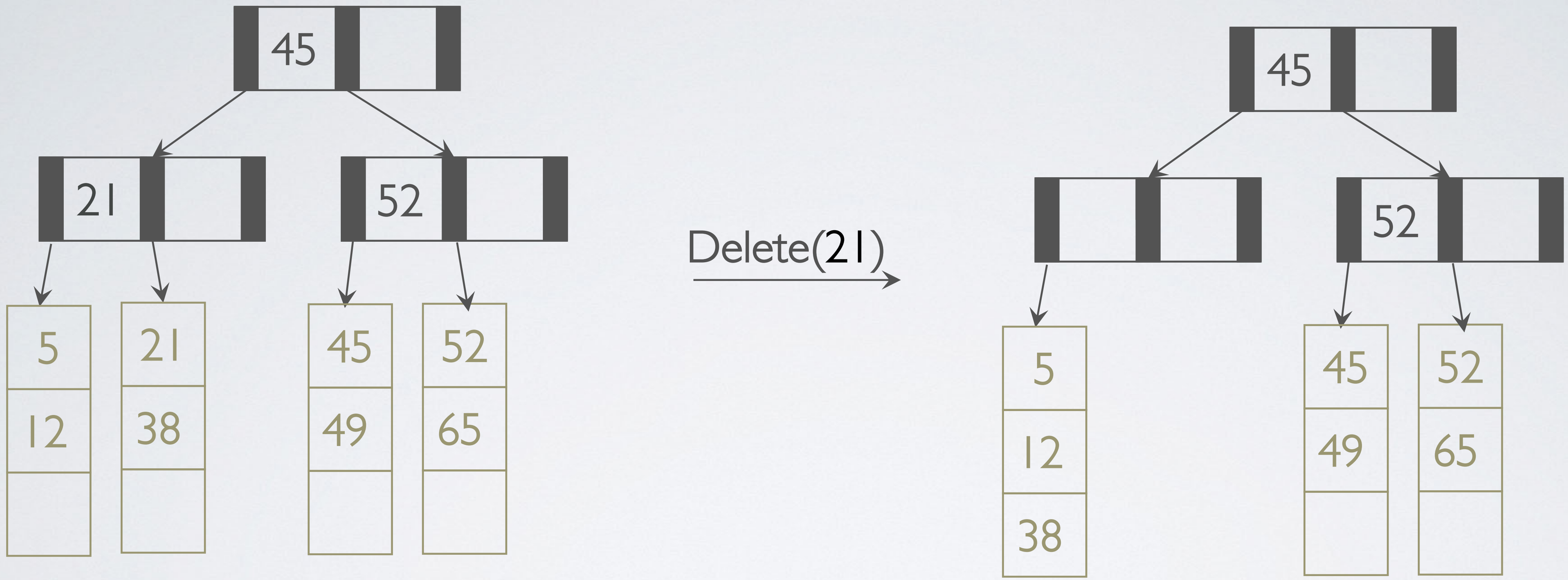
# B-TREES – STERGERE





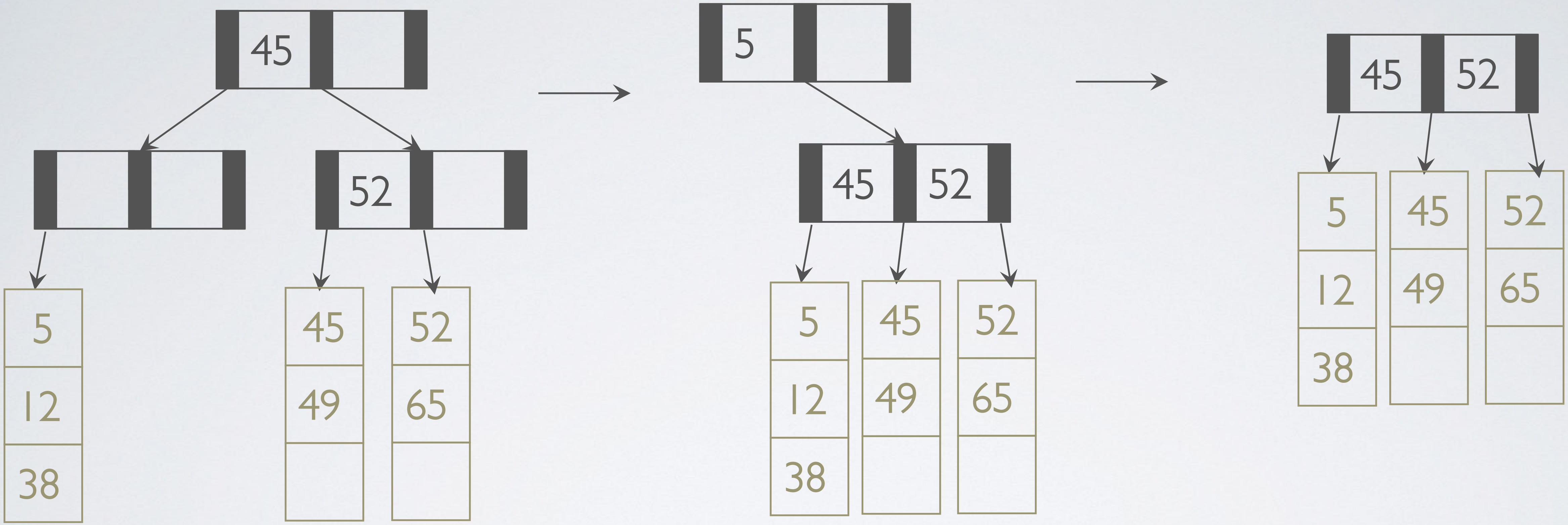
# B-TREES – STERGERE

M=3, L=3



M=3, L=3

# B-TREES – STERGERE



# B-TREES – ALGORITM STERGERE

1. Sterge elementul din frunza corespunzatoare
2. Daca acum nodul frunza are $\lceil L/2 \rceil - 1$ elemente, *underflow*!
   a. Daca un vecin are $> \lceil L/2 \rceil$ elemente, adopta si updateaza parintele
   b. Altfel combina frunza cu cea vecina
      - Avem garantat numar legal de elemente
      - Parintele are acum cu $1$ nod mai putin
3. Daca pasul 2 a cauzat *underflow* in parinte (are $\lceil M/2 \rceil - 1$ copii)
   a. Daca un vecin are $> \lceil M/2 \rceil$ chei, adopta si updateaza parintele
   b. Altfel combina nodul cu vecinul
      - Parintele are acum cu $1$ nod mai putin, e posibil sa fie nevoie sa continuam in sus, posibil pana la radacina

- Daca radacina a ajuns sa aiba doar $1$ copil, stergem radacina, copilul devine radacina
  - Doar asa scade inaltimea arborelui!

# B-TREES – EFICIENTA ALGORITM STERGERE

| | | |
|---|---|---|
| 1. | Gasirea frunzei corecte: | 1. $O(\log_2 M \log_M n)$ |
| 2. | Stergerea frunzei: | 2. $O(L)$ |
| 3. | Adoptie/combinare cu vecinul: | 3. $O(L)$ |
| 4. | Adoptie/combinare pe noduri parinte, pana la radacina: | 4. $O(M \log_M n)$ |

Total: $O(L + M \log_M n)$

DAR, nu e asa rau, pentru ca:

- Combinarile nu sunt asa de comune
- Ca si la inserare, reducerea nr. de accese la disc este prioritatea principala: $O(\log_M n)$

# B-TREES – MORALA

De ce sunt potriviti pentru date stocate extern (pe disc)?

- Se stocheaza multe chei la un nod
  - Toate sunt aduse in memorie intr-un singur acces la disc
  - Face cautarea binara peste *M*-1 chei sa fie intr-adevar eficienta
  - *M* trebuie ales potrivit (e.g. dim. bloc = 1KB, dim. cheie = 4B, dim. pointer = 4B, M=?)

- Nodurile interne contin *doar chei (B+ trees)*
  - In orice cautare acceseaza doar un item de date
  - Deci se aduce doar continutul unei singure frunze in memorie
  - Dimensiunea datelor nu afecteaza valoarea lui *M*

# STRUCTURI DE DATE PENTRU MULTIMI DISJUNCTE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- A.k.a. *union-find*, *merge-find*
- Structura de date pentru a reprezenta o multime de elemente partitionata intr-un numar de sub-multimi disjuncte (intersectia a oricare 2 multimi e vida)
  - Exemple utilizare
    - Algoritmul de determinare a componentelor conexe ale unui graf
    - Algoritmul lui Kruskal – Arbore de Acoperire Minim
    - Lowest Common Ancestor, offline (se da multimea de perechi de noduri)
    - Analiza alias-urilor in teoria compilatoarelor

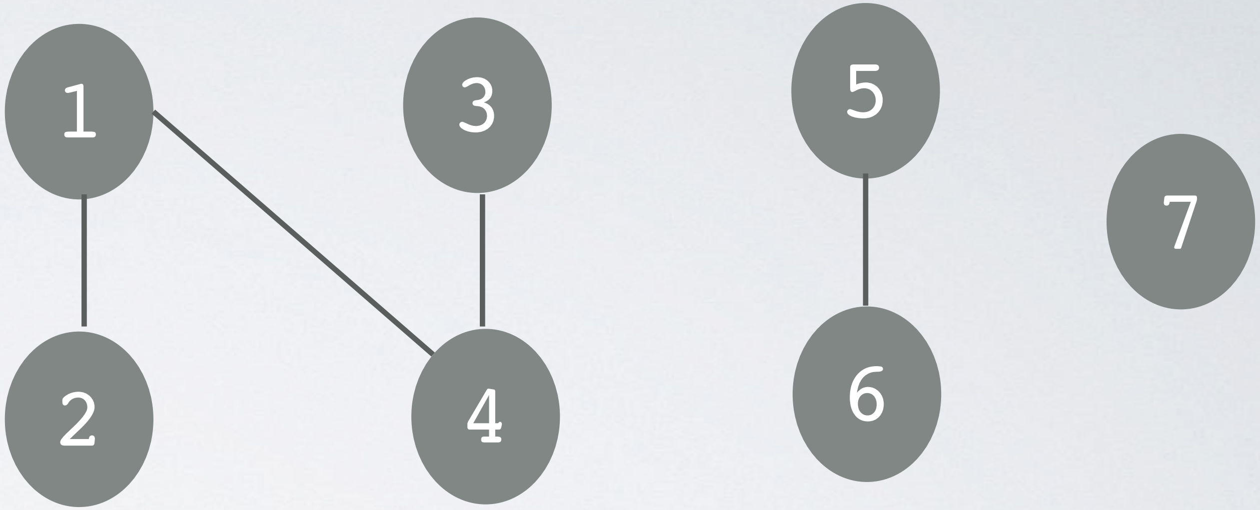
# STRUCTURI DE DATE PENTRU MULTIMI DISJUNCTE

- Clase de echivalenta
  - Daca multimea S are o *relatie de echivalenta* definita pe elementele ei (reflexiva, tranzitiva, simetrica), atunci ea se poate partitiona in multimile $S_1$, $S_2$, …$S_n$, a.i.
    - $\forall i, j, S_i \cap S_j = \emptyset \; si \;\; \bigcup_k S_k = S$
- Problema de echivalenta
  - se da S si o secventa de relatii de forma: $a \equiv b$
  - se proceseaza relatiile in ordine, astfel incat la orice moment sa se poata specifica clasa de echivalenta de care apartine un anumit element

# MULTIMI DISJUNCTE – EXEMPLU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- S={1,2,…,7}
- 1 ≡ 2, 5 ≡ 6, 3 ≡ 4, 1 ≡ 4
- 1 ≡ 2: {1,2}{3}{4}{5}{6}{7}
- 5 ≡ 6: {1,2}{3}{4}{5,6}{7}
- 3 ≡ 4: {1,2}{3,4}{5,6}{7}
- 1 ≡ 4: {1,2,3,4}{5,6}{7}

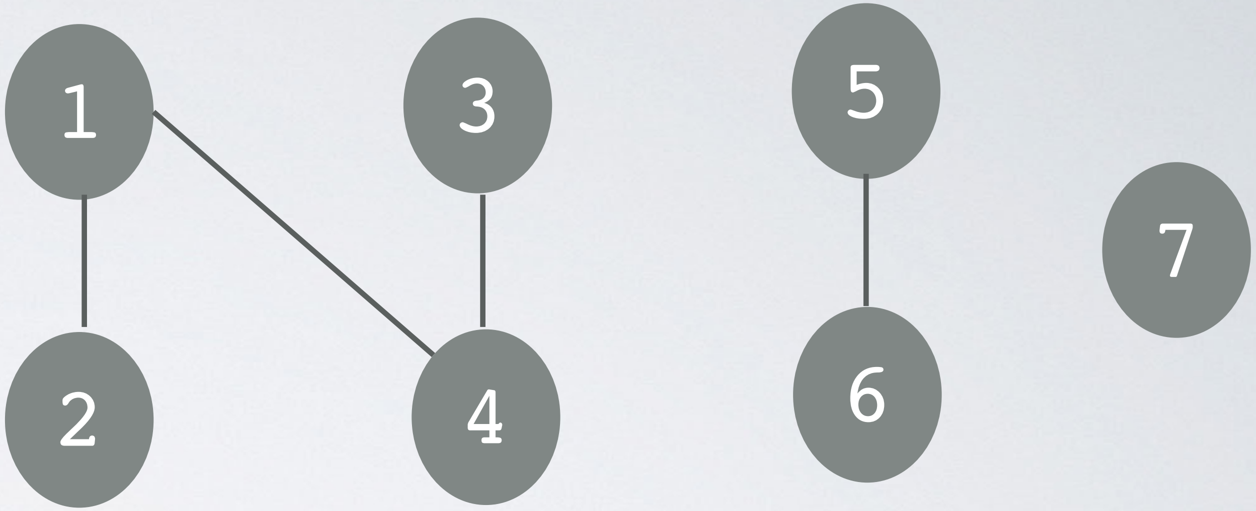



Determinarea componentelor conexe

# MULTIMI DISJUNCTE: OPERATII

- <u>UNION(*A, B*)</u> genereaza reuniunea multimilor A si B; rezultatul este stocat fie in A, fie in B; cealalta multime dispare
- <u>FIND-SET(*x*)</u>, returneaza multimea de care apartine x (nume, referinta, etc)
- <u>MAKE-SET(*A, x*)</u> creeaza multimea *A* care contine elementul x
- Fiecare multime are un <u>*element reprezentativ*</u>, prin care se identifica multimea

# MULTIMI DISJUNCTE – EXEMPLU

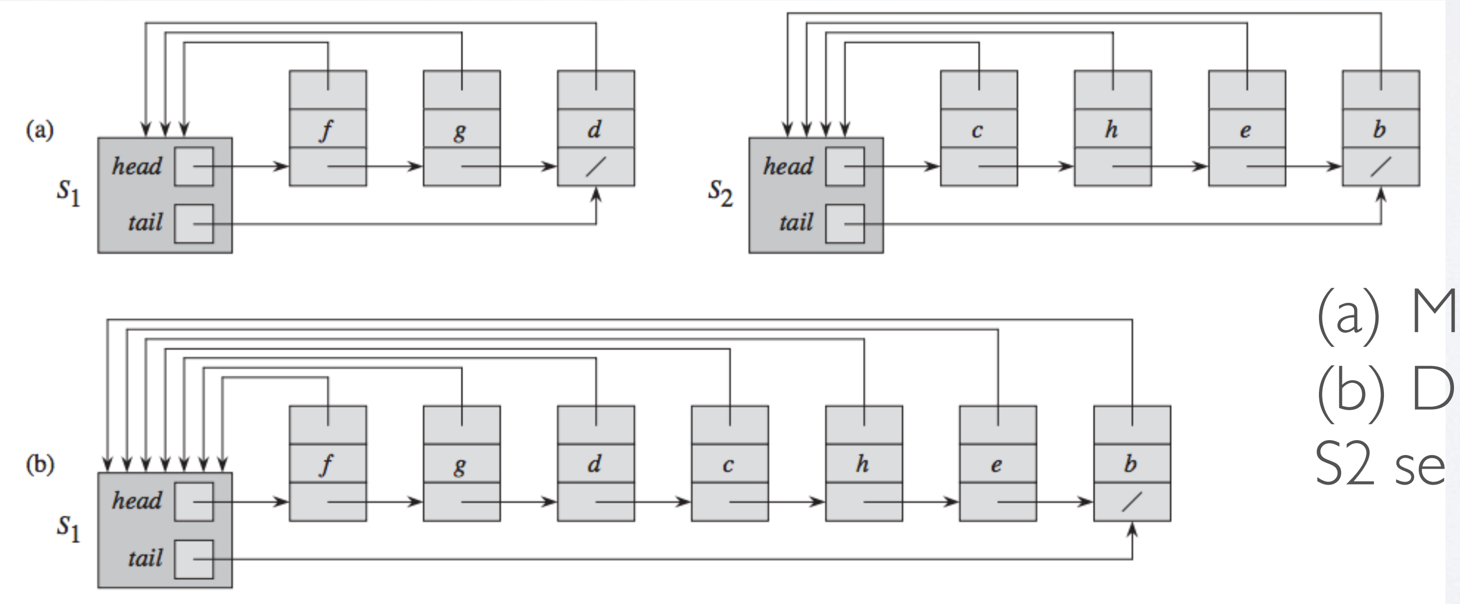



```
CONNECTED-COMPONENTS(G)
for each vertex in G.V
    MAKE-SET(v)
for each edge u in G.E
    if FIND-SET(u) != FIND-SET(v)
        UNION(u,v)
```

Determinarea componentelor conexe

# MULTIMI DISJUNCTE: IMPLEMENTAREA CU LISTE

- Multimea - lista inlantuita (pointer la first, eventual si la last)
- Fiecare nod - obiect de multime; contine elementul, adresa urmatorului element si referinta la reprezentantul multimii (obiectul de tip lista)
- **`MAKE-SET(x), FIND-SET(x), UNION(x,y)?`**




(a) Multimile $S_1$ si $S_2$, ca liste
(b) Dupa union, multimea rezultata e $S_1$, S2 se stergre

*Th. Cormen, "Introduction to Algorithms"*

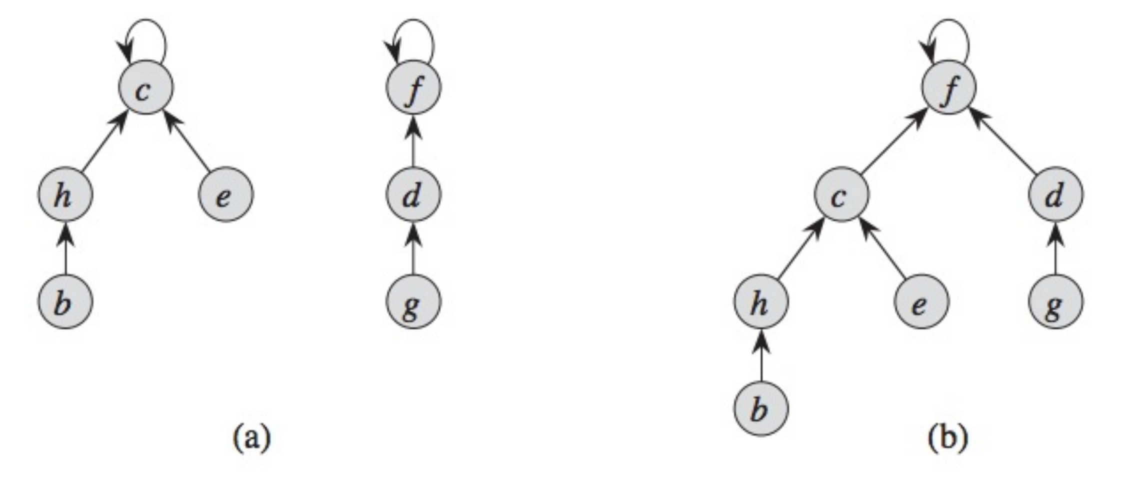
# MULTIMI DISJUNCTE: IMPLEMENTAREA CU ARBORI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Multimea - arbore, radacina arborelui identifica multimea
- Fiecare nod - obiect de multime; contine elementul si referinta la un nod parinte
- Radacina - identificatorul multimii; legatura parinte pointeaza catre acelasi nod (self-reference)
- Se poate utiliza reprezentarea de vectori de parinti pentru arbore: parent[i]
- **`FIND-SET(x)`** returneaza radacina arborelui care contine elementul x
- **`UNION(x,y)`** combina arborii care

contin elementele x si y

(a) Multimile $S_1$ si $S_2$, reprezentarea cu arbori
(b) Dupa union, reprezentantul unei multimi va pointa (legatura parinte) catre reprezentantul celeilalte multimi



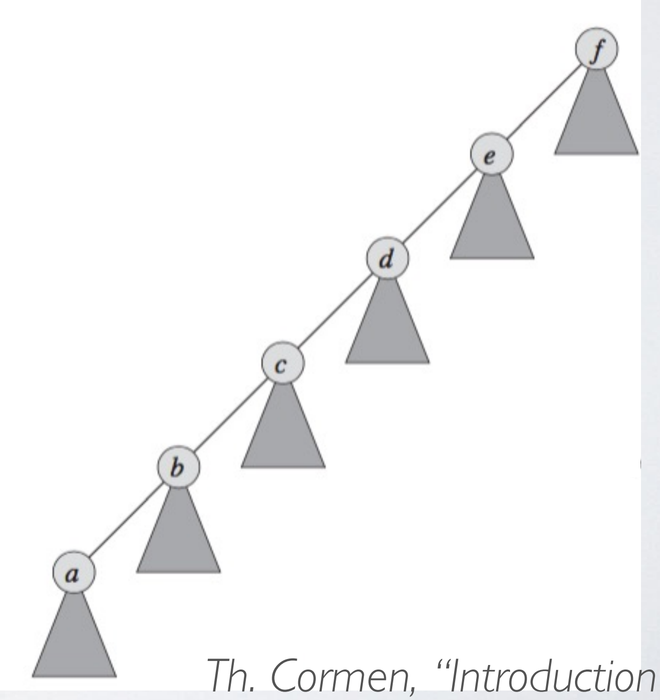

*Th. Cormen, "Introduction to Algorithms"*

# MULTIMI DISJUNCTE: IMPLEMENTAREA CU ARBORI

```
MAKE-SET(x)
    x.p = x

FIND-SET(x)
    crt <- x
    while parent[crt] != crt
        crt <- parent[crt]
    return crt
```

```
UNION(x,y)
    root1 <- FIND-SET(x)
    root2 <- FIND-SET(y)
    if root1 != root2 then
        parent[root2] <- root1
```




Th. Cormen, "Introduction to Algorithms"

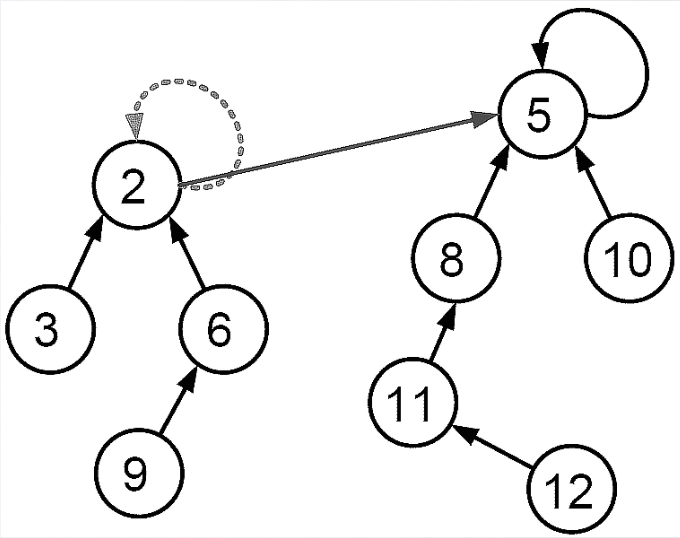
Dupa mai multe op. de union, arborele ar putea degenera

# MULTIMI DISJUNCTE: IMPLEMENTAREA CU ARBORI. IMBUNATATIRI (I)

- Stocam in fiecare nod informatie legata de limita superioara a inaltimii - *rang*
- *Union by rank:*
  - La o operatie de union, devine radacina arborele cu rangul maxim

```
UNION(x,y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x,y)
    if x.rank > y.rank
        y.p <- x
    else
         x.p <- y
    if x.rank = y.rank
        y.rank++
```

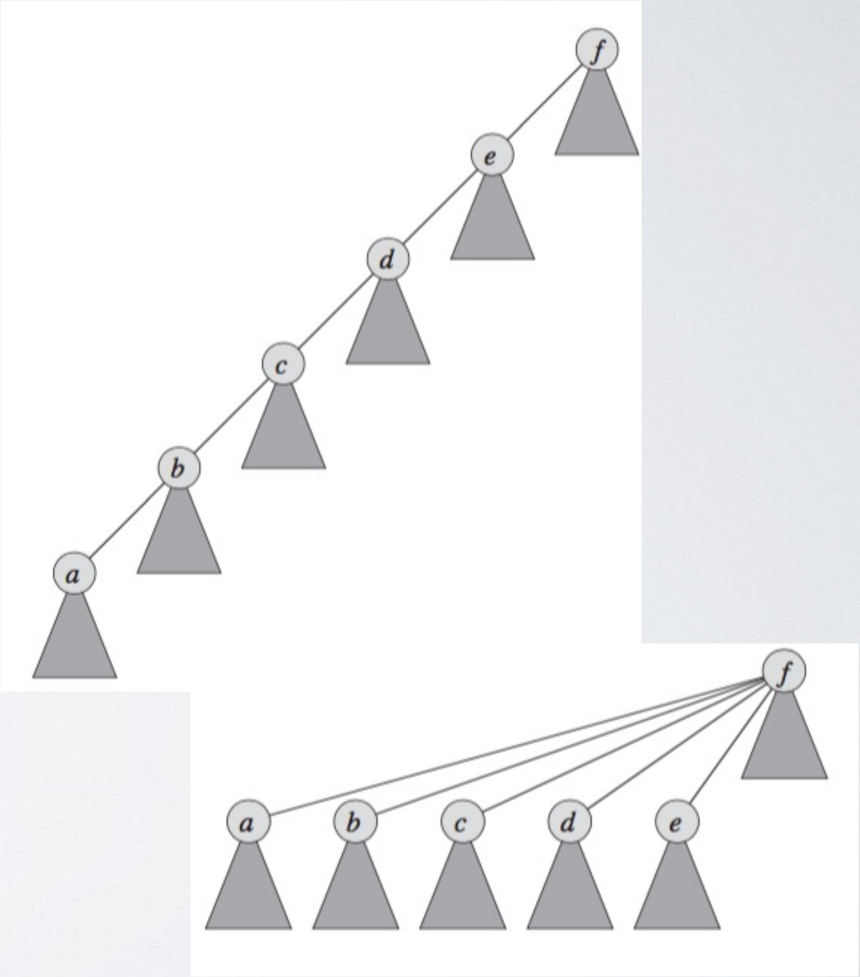


# MULTIMI DISJUNCTE: IMPLEMENTAREA CU ARBORI. IMBUNATATIRI (I)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- *Path compression*
  - dupa efectuarea unui find, se compreseaza toti pointerii de pe calea traversata astfel incat sa pointeze direct catre radacina
  - Functie in 2 treceri
    - Up (forward): gasirea radacinii
    - Down (backward): updatarea legaturilor

```
FIND-SET(x)
    if x != x.p
        x.p <- FIND-SET(x.p)
    return x.p
```



# EURISTICI DE IMBUNATATIRE A TIMPULUI: ANALIZA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Analiza amortizata
- Considerand
  - n - numarul de operatii MAKE-SET
  - m - numarul total de operatii (MAKE-SET, FIND-SET, UNION)
- Union by rank: implica timp $O(n \log n)$ pentru $n$ operatii union-find:
  - de fiecare cand urmam un pointer parinte, trecem intr-un sub-arbore de dimensiune cel putin dubla fata de sub-arborele initial
  - prin urmare, o operatie de *find* parcurge cel mult $\boldsymbol{O(\log n)}$ pointeri
- Path compression: implica imp $O(n \log n)$ pentru $n$ operatii union-find

- $O(m\alpha(n))$, $\alpha(n)$ o functie care creste f incet

# BIBLIOGRAFIE


- Curs B-trees, U. Washington: https://courses.cs.washington.edu/courses/cse332/10sp/
- CLR – capitolul 18 (B-Trees)
- CLR - capitolul 21 (Data Structures for Disjoint Sets)

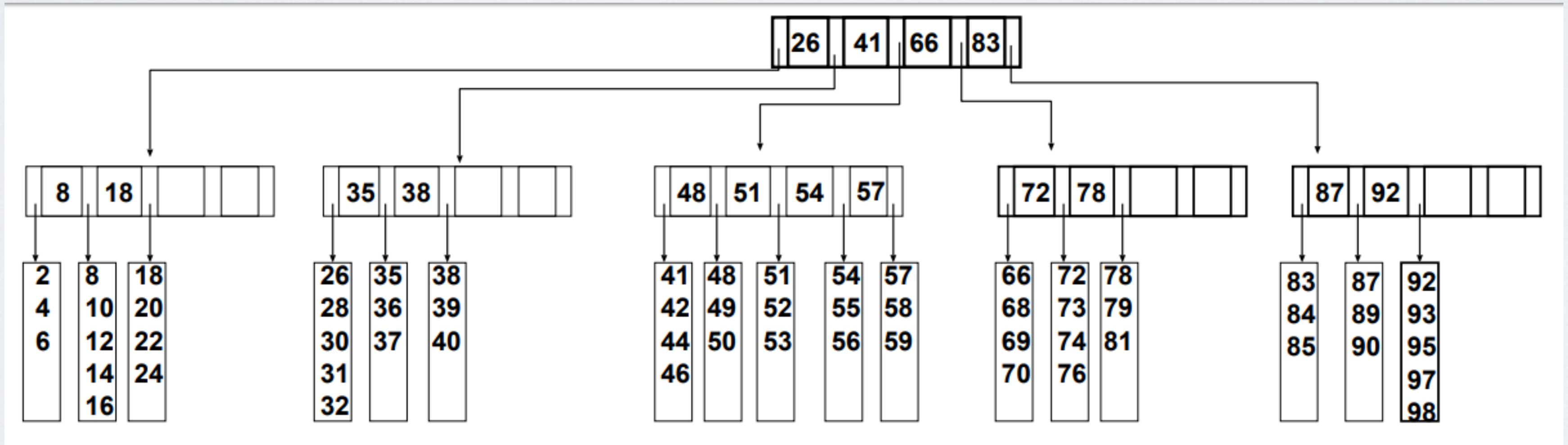
# EXERCITII B-TREES

1. Se da b-arborele din figura.

a) Sa se calculeze M si L

b) Care este costul (numar de accese la disc) necesar sa se acceseze orice informatie din arbore?

c) Desenati cum arata arborele dupa fiecare din operatiile: insert (1), insert (33), insert (17) delete(81)



# EXERCITII MULTIMI DISJUNCTE



Multimi disjuncte

1. Se da o matrice care contine valori de 0 si de 1. Presupunem ca toate valorile de 1 care sunt conectate (au cel putin un vecin egal cu 1) reprezinta un obiect. Sa se determine cate obiecte sunt in imagine folosind functiile de la multimi disjuncte.
2. Aplicatii practice in procesare de imagini:
    1. Cate linii contine un document scanat
    2. Cate semnaturi sunt pe o pagina