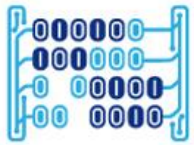


MACC
Matemáticas Aplicadas y
Ciencias de la Computación

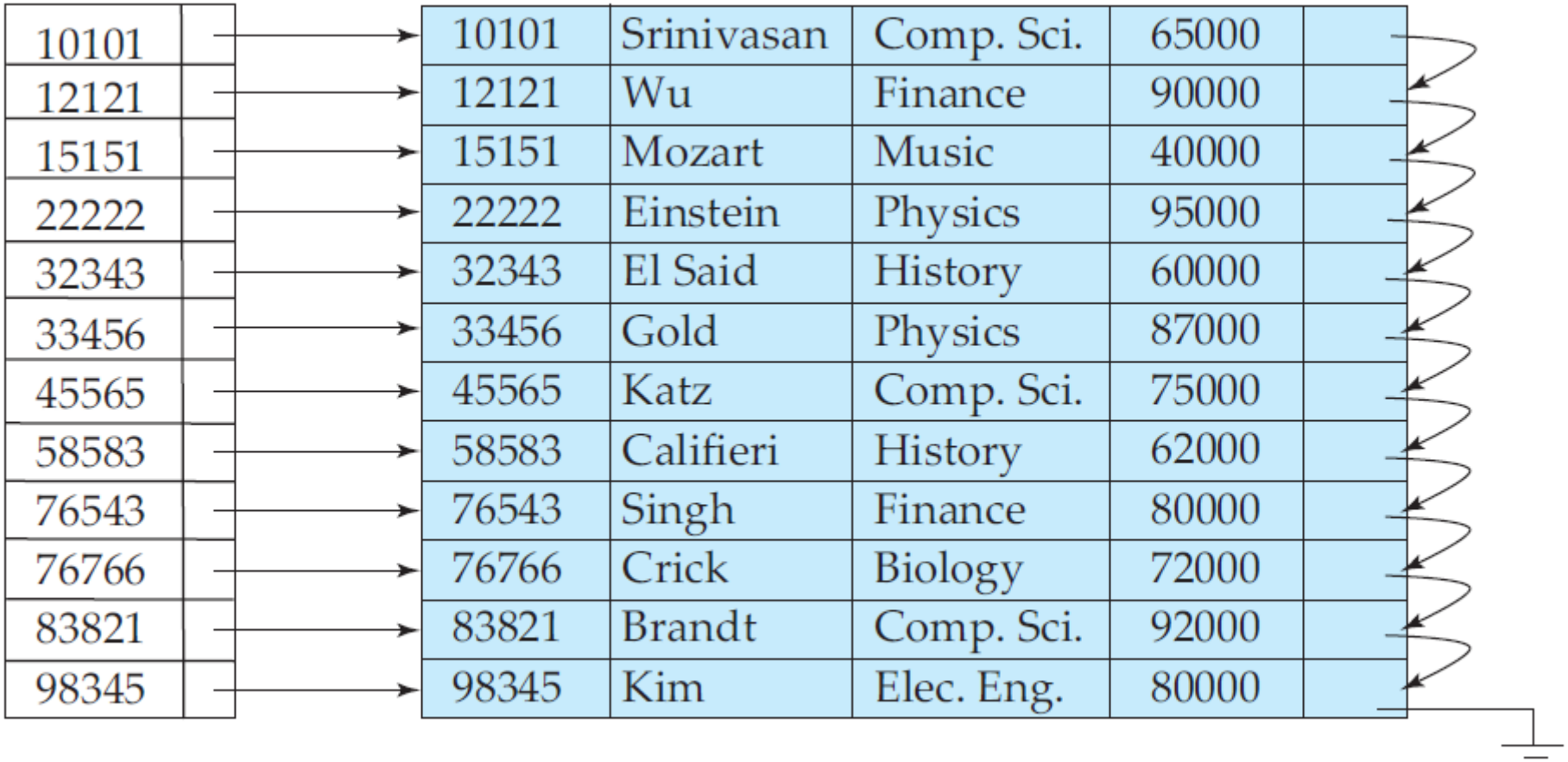
Índices

Manejo de Bases de Datos

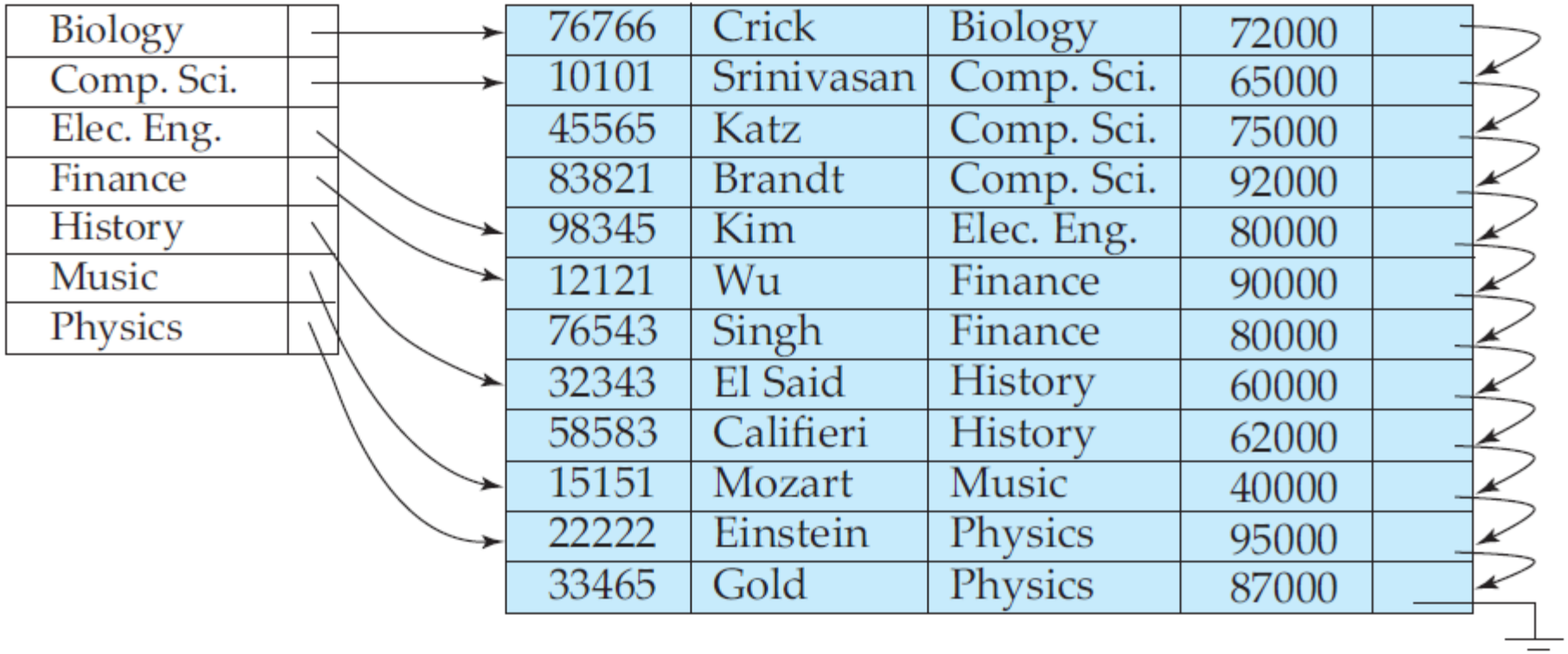
Juan F. Pérez
Departamento de Matemáticas Aplicadas
y Ciencias de la Computación

2019 - 1

Índices

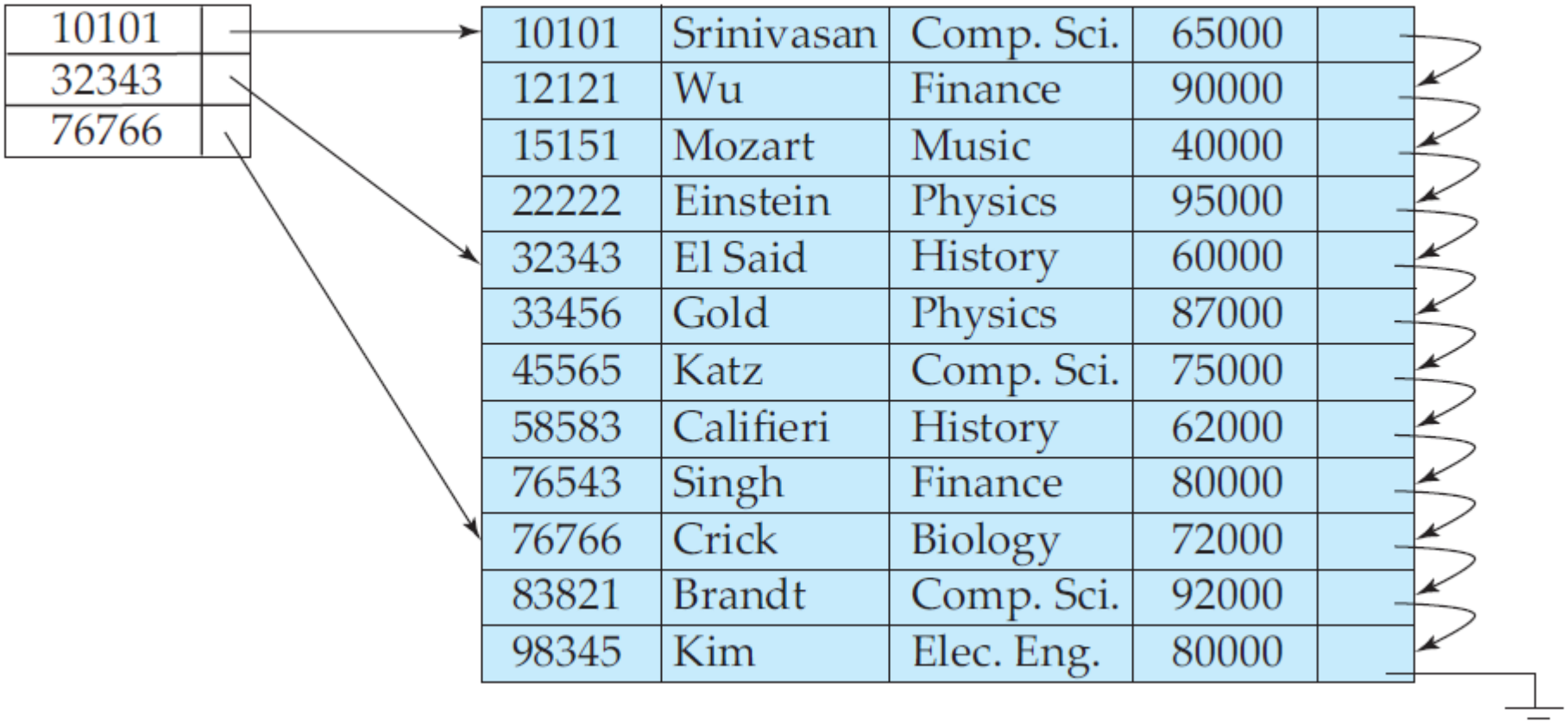

10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345
10101 Srinivasan Comp. Sci. 65000
12121 Wu Finance 90000
15151 Mozart Music 40000
22222 Einstein Physics 95000
32343 El Said History 60000
33456 Gold Physics 87000
45565 Katz Comp. Sci. 75000
58583 Califieri History 62000
76543 Singh Finance 80000
76766 Crick Biology 72000
83821 Brandt Comp. Sci. 92000
98345 Kim Elec. Eng. 80000

Índices


Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics
76766 Crick Biology 72000
10101 Srinivasan Comp. Sci. 65000
45565 Katz Comp. Sci. 75000
83821 Brandt Comp. Sci. 92000
98345 Kim Elec. Eng. 80000
12121 Wu Finance 90000
76543 Singh Finance 80000
32343 El Said History 60000
58583 Califieri History 62000
15151 Mozart Music 40000
22222 Einstein Physics 95000
33465 Gold Physics 87000

Índices

Disperso

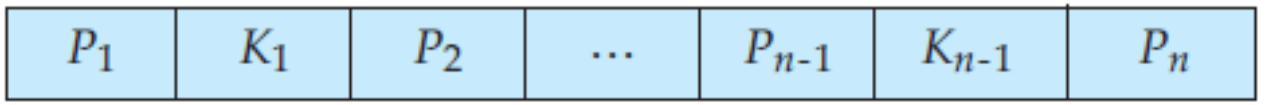


Índices y Árboles B+

Nodo

P: apuntador

K: valor de llave de búsqueda



n: parámetro

Al menos ceil(n/2) apuntadores

A lo sumo n apuntadores

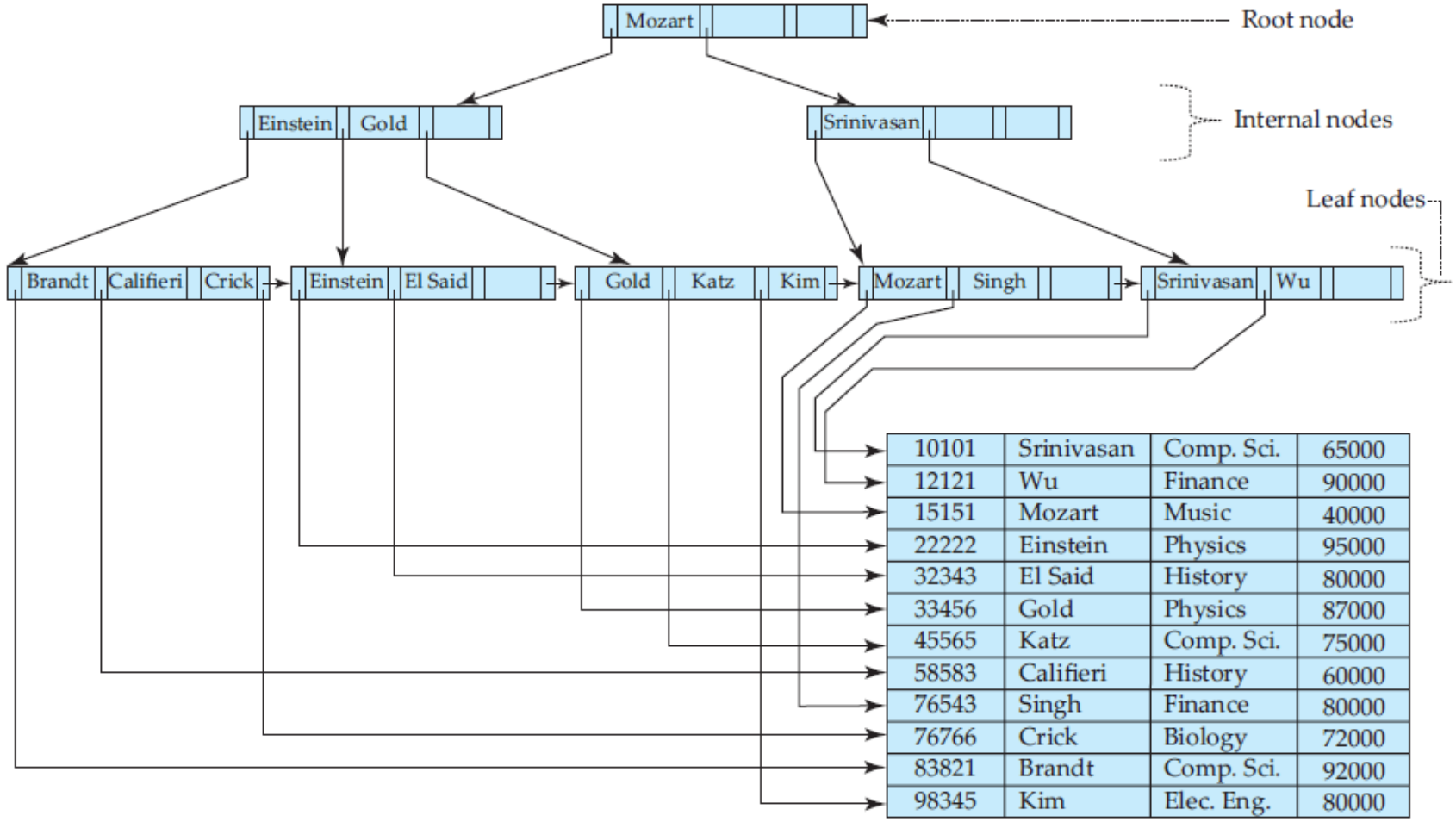
Índices y Árboles B+

Nodos hoja (n=4):
apuntadores a registros

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

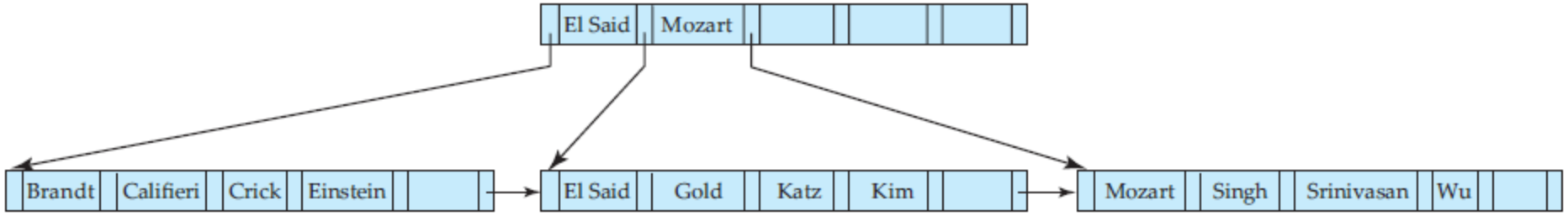
Índices y Árboles B+

n=4



Índices y Árboles B+

n=6



Índices y Árboles B+

Búsqueda

```
function find(value V)
/* Returns leaf node C and index i such that C.P_i points to first record
* with search key value V */
    Set C = root node
    while (C is not a leaf node) begin
        Let i = smallest number such that V ≤ C.K_i
        if there is no such number i then begin
            Let P_m = last non-null pointer in the node
            Set C = C.P_m
        end
        else if (V = C.K_i)
            then Set C = C.P_{i+1}
        else C = C.P_i /* V < C.K_i */
    end
    /* C is a leaf node */
    Let i be the least value such that K_i = V
    if there is such a value i
        then return (C, i)
        else return null ; /* No record with key value V exists*/
```

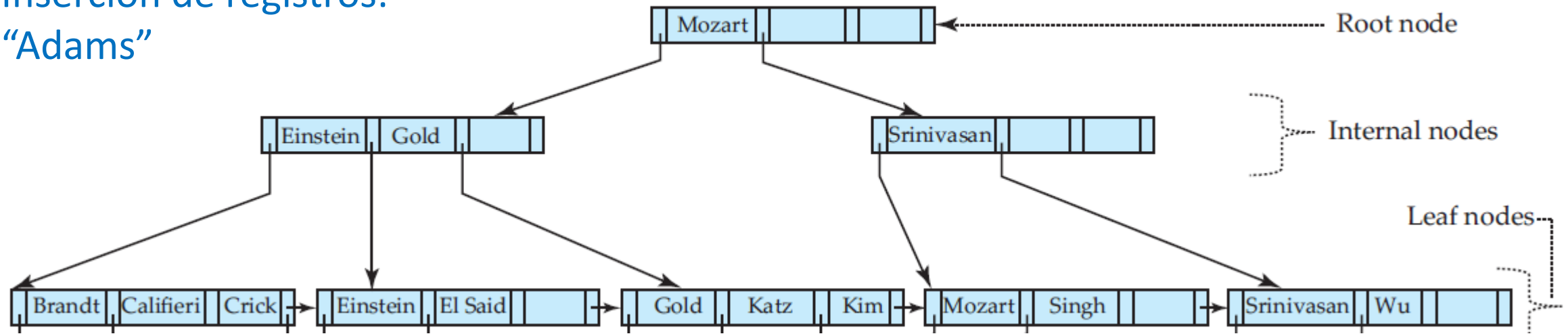
Índices y Árboles B+

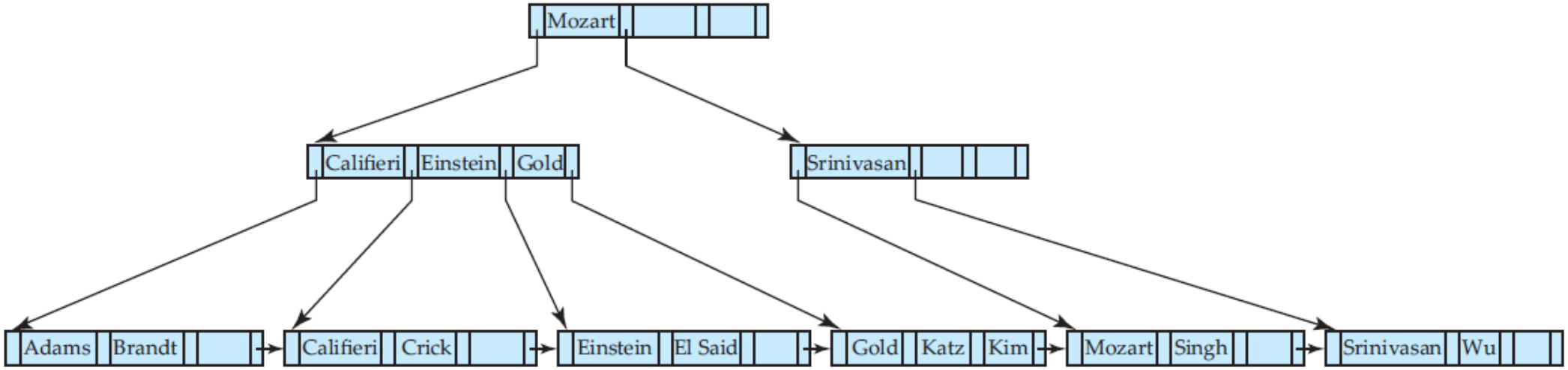
Búsqueda varios

```
procedure printAll(value V)
/* prints all records with search key value V */
    Set done = false;
    Set (L, i) = find(V);
    if ((L, i) is null) return
    repeat
        repeat
            Print record pointed to by L.P_i
            Set i = i + 1
        until (i > number of keys in L or L.K_i > V)
        if (i > number of keys in L)
            then L = L.P_n
            else Set done = true;
    until (done or L is null)
```

Índices y Árboles B+

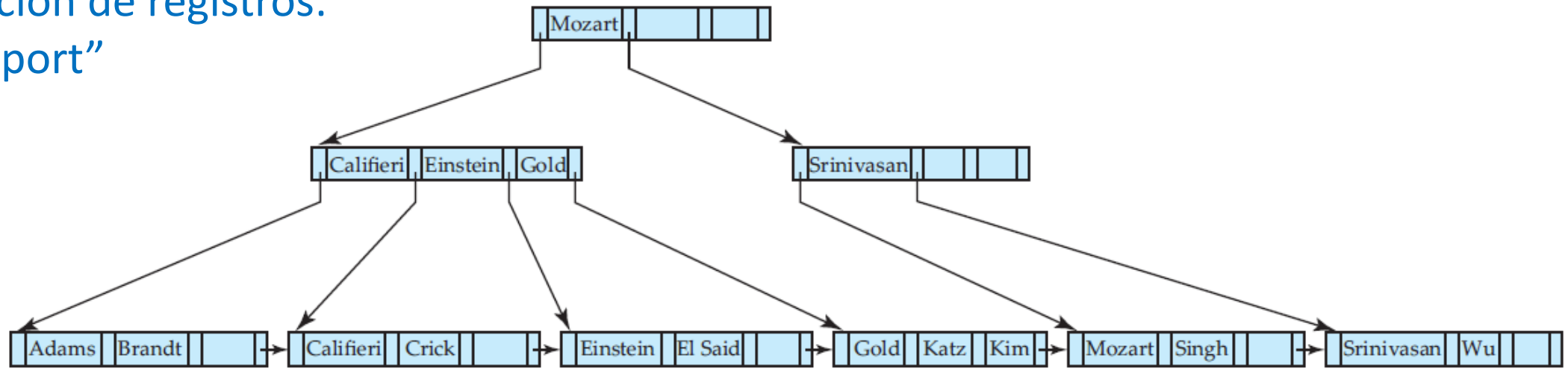
Inserción de registros:
"Adams"

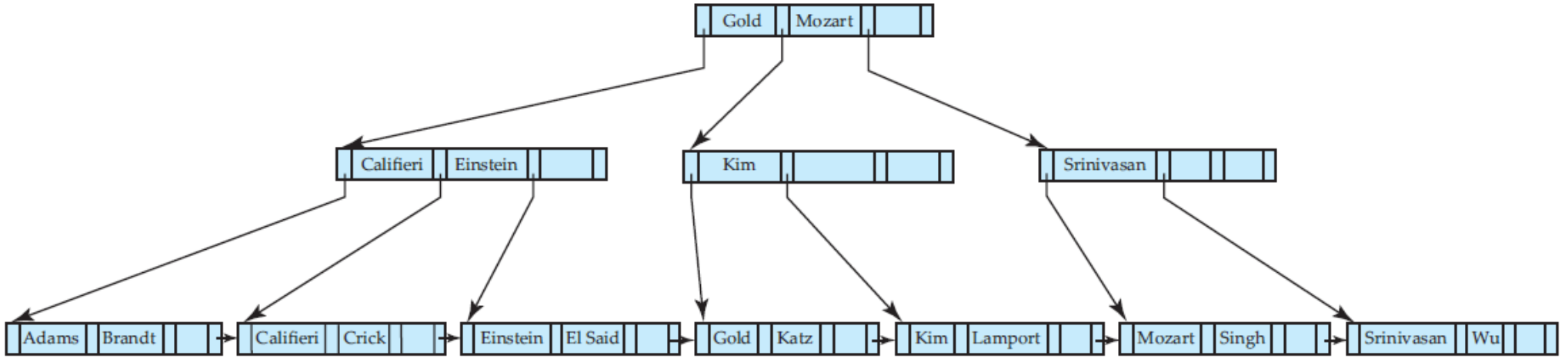




Índices y Árboles B+

Inserción de registros:
"Lamport"





Índices y Árboles B+

Inserción de registros

```
procedure insert(value K, pointer P)
    if (tree is empty) create an empty leaf node L, which is also the root
    else Find the leaf node L that should contain key value K
    if (L has less than n − 1 key values)
        then insert_in_leaf (L, K, P)
        else begin /* L has n − 1 key values already, split it */
            Create node L'
            Copy L.P1 ... L.Kn−1 to a block of memory T that can
                hold n (pointer, key-value) pairs
            insert_in_leaf (T, K, P)
            Set L'.Pn = L.Pn; Set L.Pn = L'
            Erase L.P1 through L.Kn−1 from L
            Copy T.P1 through T.K⌈n/2⌉ from T into L starting at L.P1
            Copy T.P⌈n/2⌉+1 through T.Kn from T into L' starting at L'.P1
            Let K' be the smallest key-value in L'
            insert_in_parent(L, K', L')
        end
```

Índices y Árboles B+

Inserción de registros

```
procedure insert_in_leaf (node L, value K, pointer P)
    if (K < L.K_1)
        then insert P, K into L just before L.P_1
        else begin
            Let K_i be the highest value in L that is less than K
            Insert P, K into L just after T.K_i
        end
```

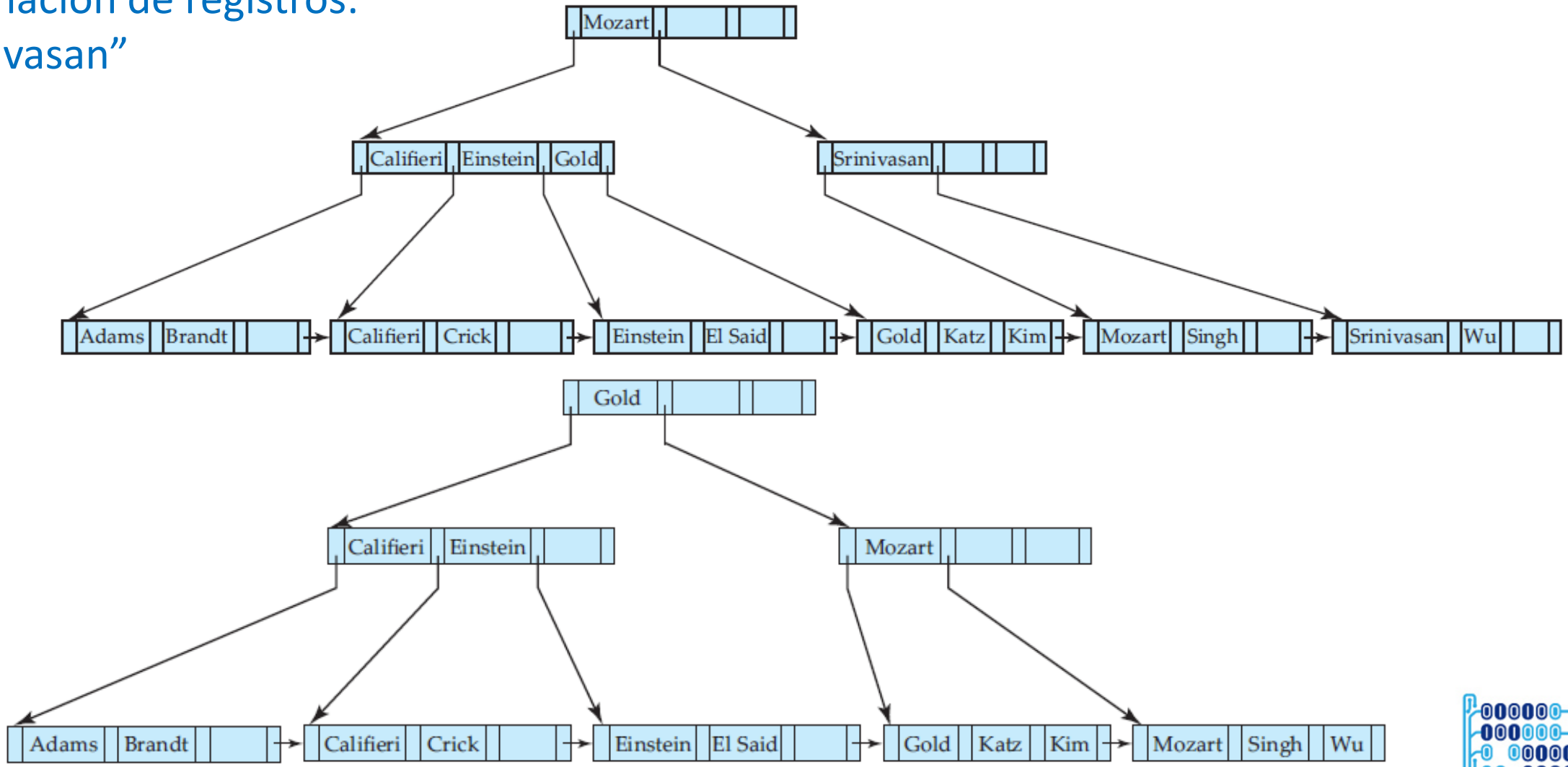
Índices y Árboles B+

Inserción de registros

```
procedure insert_in_parent(node N, value K', node N')
    if (N is the root of the tree)
        then begin
            Create a new node R containing N, K', N'    /* N and N' are pointers */
            Make R the root of the tree
            return
        end
    Let P = parent (N)
    if (P has less than n pointers)
        then insert (K', N') in P just after N
        else begin /* Split P */
            Copy P to a block of memory T that can hold P and (K', N')
            Insert (K', N') into T just after N
            Erase all entries from P; Create node P'
            Copy T.P_1 ... T.P_⌈n/2⌉ into P
            Let K'' = T.K_⌈n/2⌉
            Copy T.P_⌈n/2⌉+1 ... T.P_n+1 into P'
            insert_in_parent(P, K'', P')
        end
```

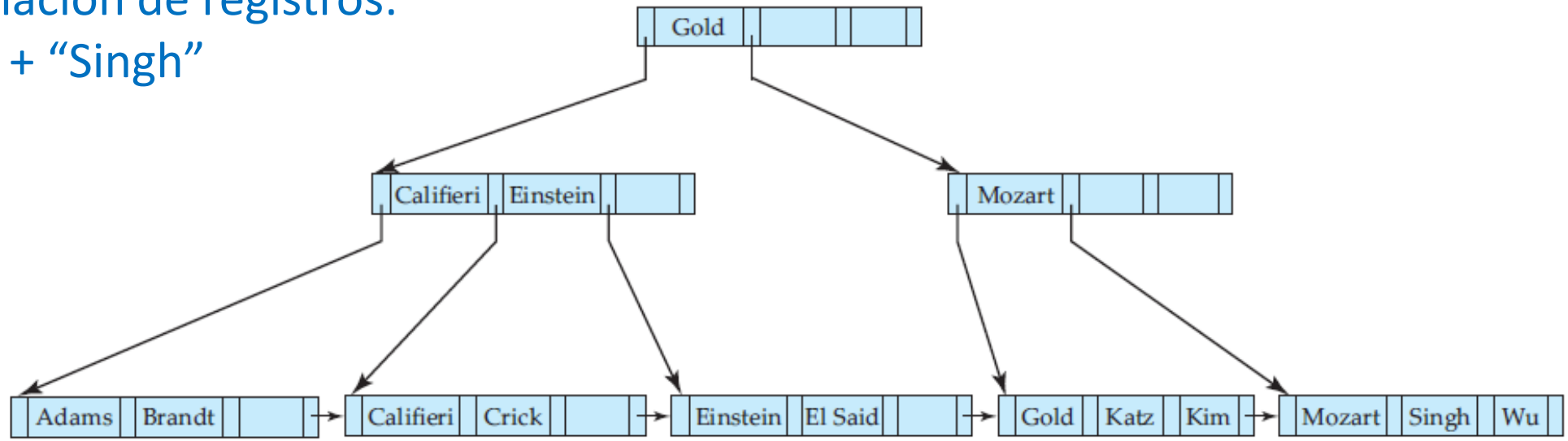
Índices y Árboles B+

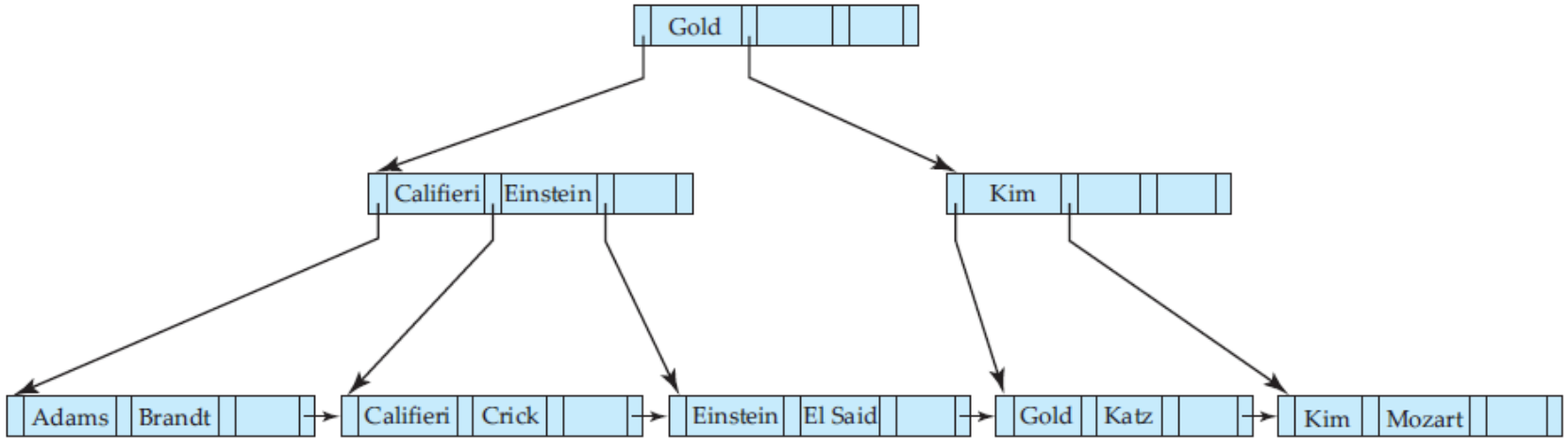
Eliminación de registros:
"Srinivasan"



Índices y Árboles B+

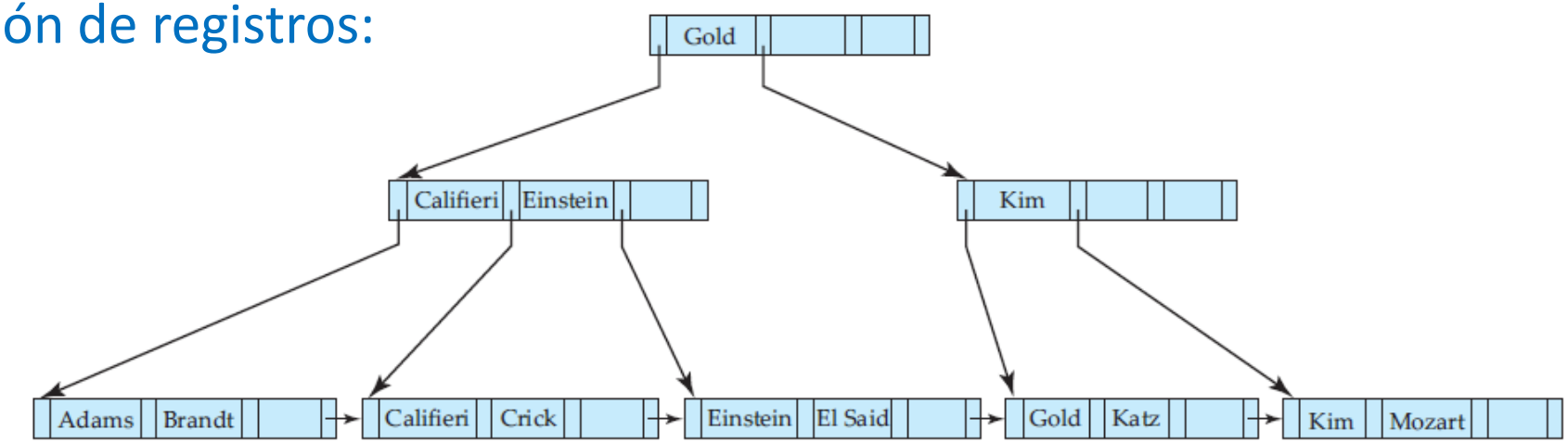
Eliminación de registros:
"Wu" + "Singh"

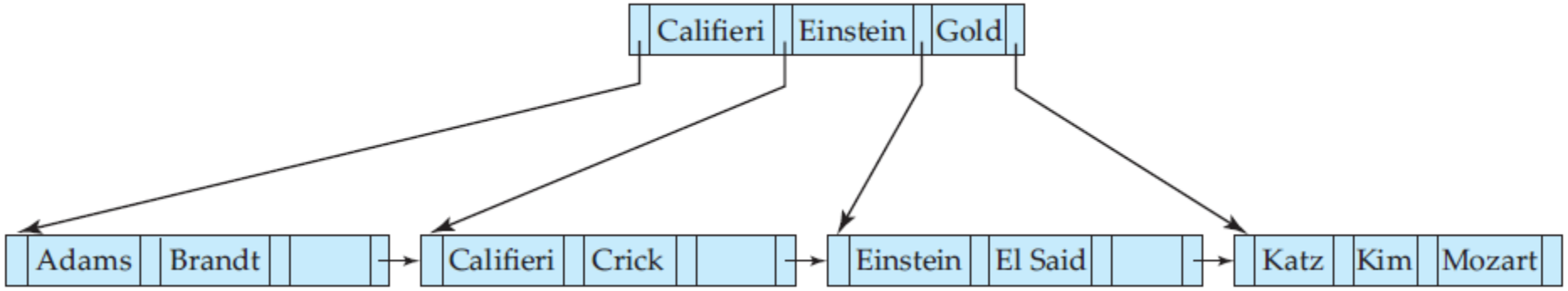




Índices y Árboles B+

Eliminación de registros:
"Gold"





Índices y Árboles B+

Eliminación de registros

```
procedure delete(value K, pointer P)
    find the leaf node L that contains (K, P)
    delete_entry(L, K, P)
```

Índices y Árboles B+

Eliminación de registros

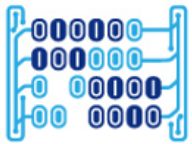
```
procedure delete_entry(node N, value K, pointer P)
    delete (K, P) from N
    if (N is the root and N has only one remaining child)
    then make the child of N the new root of the tree and delete N
    else if (N has too few values/pointers) then begin
        Let N' be the previous or next child of parent(N)
        Let K' be the value between pointers N and N' in parent(N)
        if (entries in N and N' can fit in a single node)
            then begin /* Coalesce nodes */
                if (N is a predecessor of N') then swap_variables(N, N')
                if (N is not a leaf)
                    then append K' and all pointers and values in N to N'
                    else append all (K_i, P_i) pairs in N to N'; set N'.P_n = N.P_n
                delete_entry(parent(N), K', N); delete node N
            end
```

Índices y Árboles B+

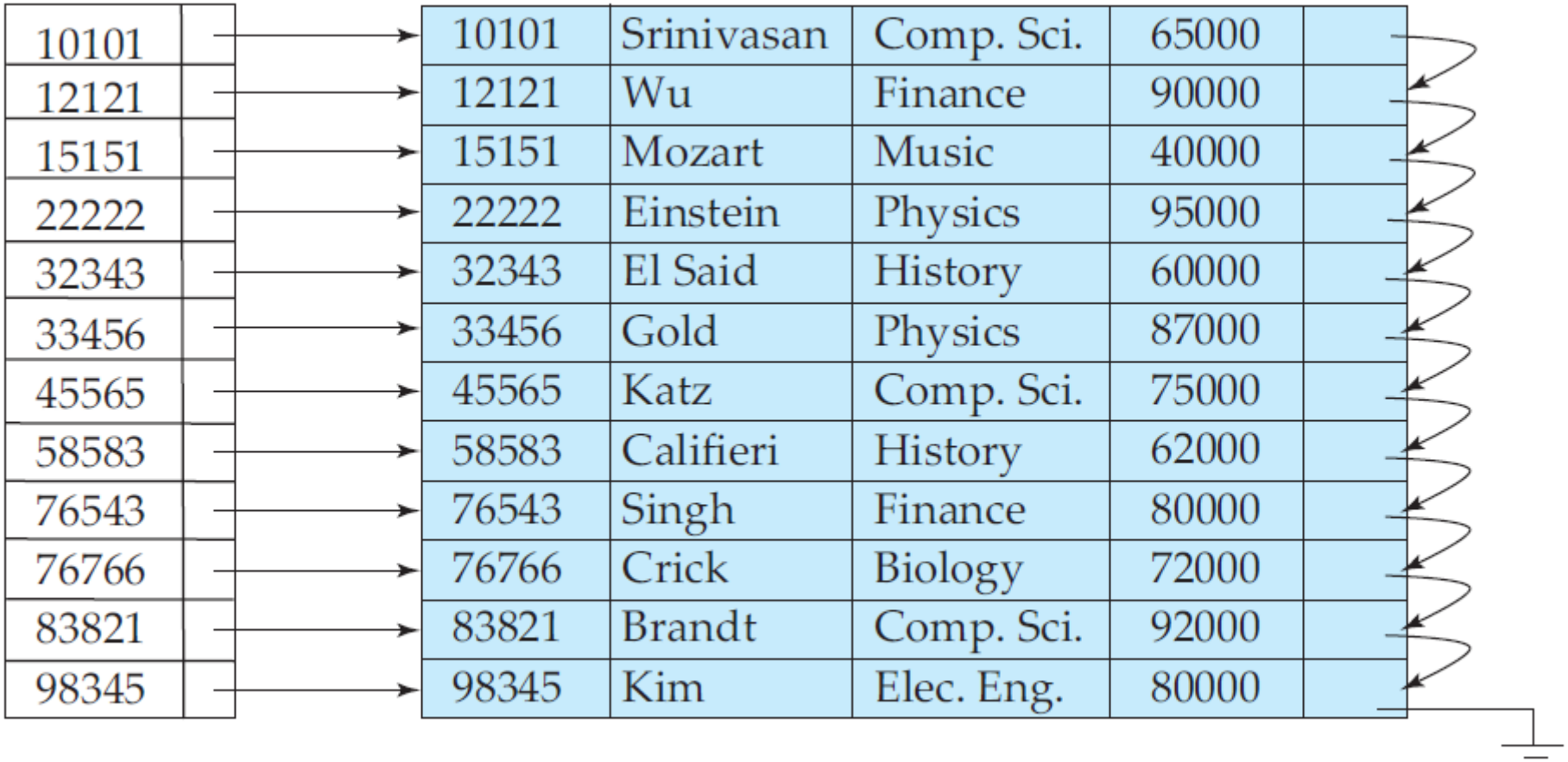
Eliminación de registros

```
else begin /* Redistribution: borrow an entry from N' */
    if (N' is a predecessor of N) then begin
        if (N is a nonleaf node) then begin
            let m be such that N'.P_m is the last pointer in N'
            remove (N'.K_{m-1}, N'.P_m) from N'
            insert (N'.P_m, K') as the first pointer and value in N,
                by shifting other pointers and values right
            replace K' in parent(N) by N'.K_{m-1}
        end
        else begin
            let m be such that (N'.P_m, N'.K_m) is the last pointer/value
                pair in N'
            remove (N'.P_m, N'.K_m) from N'
            insert (N'.P_m, N'.K_m) as the first pointer and value in N,
                by shifting other pointers and values right
            replace K' in parent(N) by N'.K_m
        end
    end
    else ... symmetric to the then case ...
end
end
```

Índices y Tablas de Hash



Índices y Tablas de Hash



10101	10101	Srinivasan	Comp. Sci.	65000
12121	12121	Wu	Finance	90000
15151	15151	Mozart	Music	40000
22222	22222	Einstein	Physics	95000
32343	32343	El Said	History	60000
33456	33456	Gold	Physics	87000
45565	45565	Katz	Comp. Sci.	75000
58583	58583	Califieri	History	62000
76543	76543	Singh	Finance	80000
76766	76766	Crick	Biology	72000
83821	83821	Brandt	Comp. Sci.	92000
98345	98345	Kim	Elec. Eng.	80000

Índices y Tablas de Hash

Tabla de hash para el índice

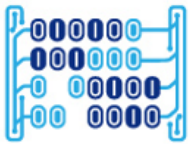
Almacenar índices (llave) y apuntadores a registros (valor) en cajas

Cada caja almacena varios registros (página)

La caja en la que se almacena un registro depende de una función de hash aplicada al registro (id)

Hashing estático:

- Desde el principio se fija el número de cajas
- Si una caja se llena se crea una nueva caja de y se enlaza como lista enlazada (de cajas)



Índices y Tablas de Hash

Hashing dinámico

Se inicia con un número de cajas que se incrementa en la medida que se necesita

Cambiar la función de hash es costoso: re-asignación global de registros a cajas (re-hashing)

Solución: usar parcialmente (en bits) el resultado de la función de hash para emplear solo las cajas que sean necesarias (1 bit: hasta 2 cajas, 2 bits: hasta 4 cajas, etc.)

Solo requiere reasignar registros de una caja llena a dos cajas nuevas que surgen de dividir la asignación usando un bit adicional