Database System Concepts

Algoritmos de Update em Árvores B^+

Departamento de Engenharia Informática
Instituto Superior Técnico

1^{st} Semester
2009/2010

Outline

Outline

Procedure *Insert*

procedure insert(value K, pointer P)
  Find the leaf node L that should contain K
  if L has less than $n-1$ values **then**
    insert-in-leaf(L, K, P)
  else begin
    Create node L'
    Copy $L.P_1 \ldots L.K_{n-1}$ to T (an auxiliary node)
    insert-in-leaf(T, K, P)
    Set $L'.P_n = L.P_n$; Set $L.P_n = L'$
    Erase $L.P_1 \ldots L.K_{n-1}$ from L
    Copy $T.P_1 \ldots T.K_{\lceil n/2 \rceil}$ to $L.P_1 \ldots$
    Copy $T.P_{\lceil n/2 \rceil + 1} \ldots T.K_n$ to $L'.P_1 \ldots$
    Let K' be the smallest key in L'
    insert-in-parent(L, K', L')
  end

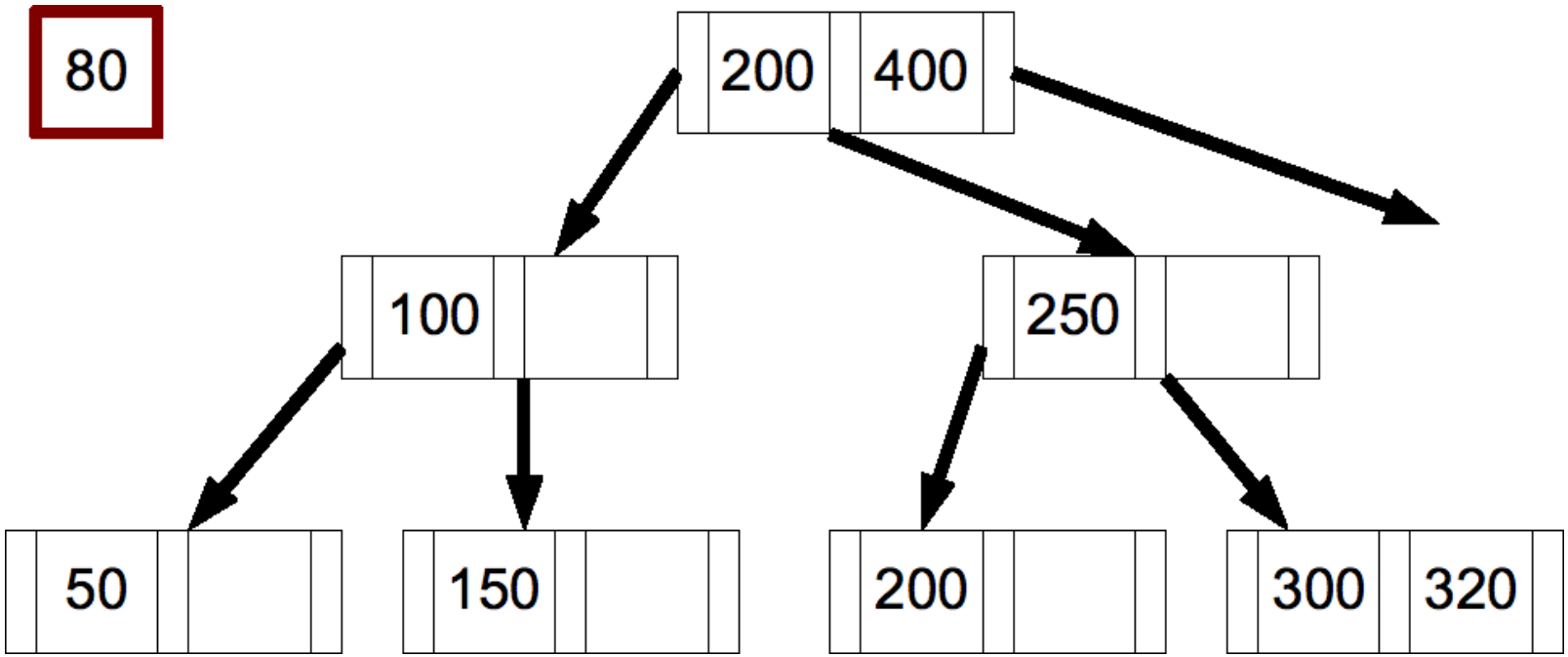
Procedure *Insert-in-leaf*

procedure insert-in-leaf(node L, value K, pointer P)
  if $K < L.K_1$ **then**
    Insert P, K into L before $L.P_1$
  else begin
    Let K_i be the least value in L that is less than K
    Insert P, K into L after $T.K_i$
  end

Procedure *Insert-in-parent*

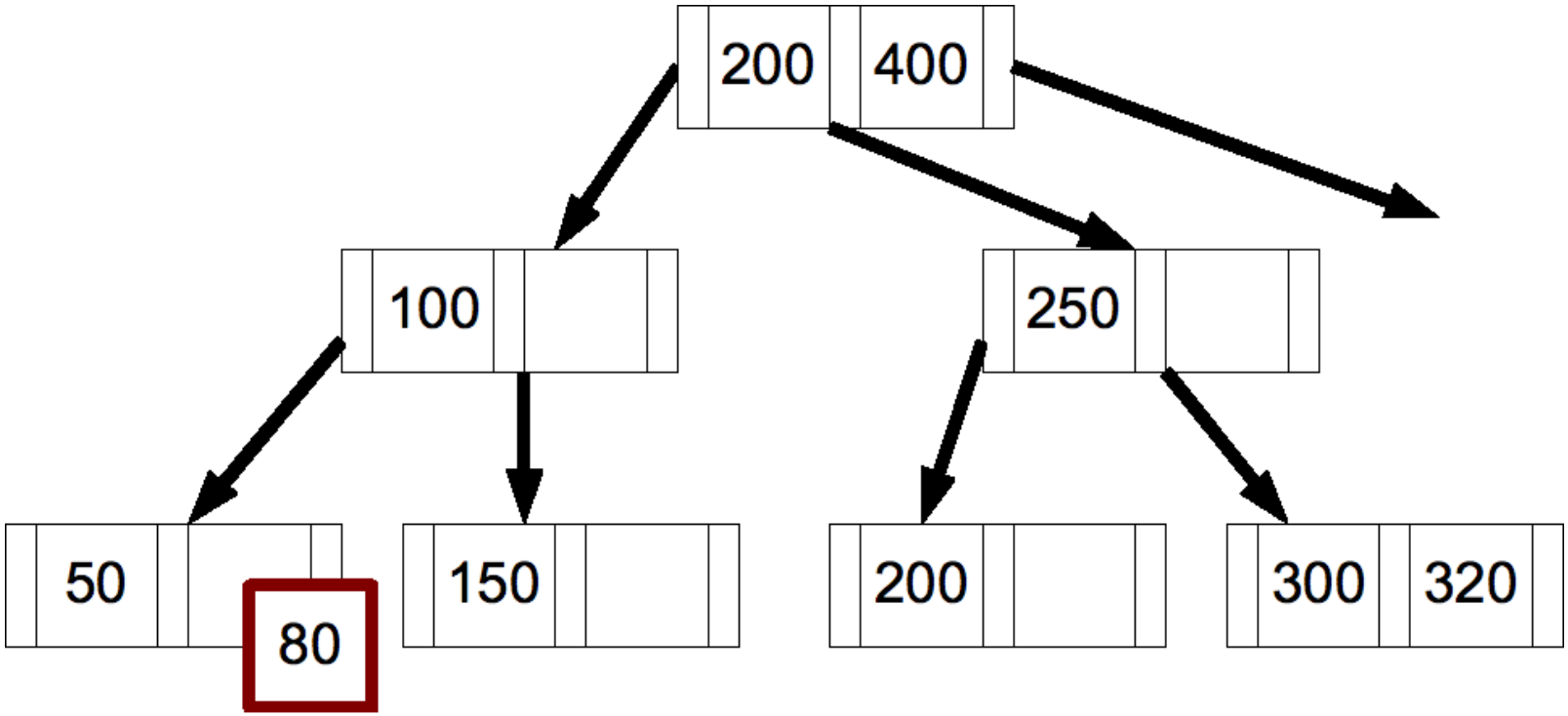
procedure insert-in-parent(node N, value K', node N')
  if N is the root **then begin**
    Create a new node $R = (N, K', N')$
    Make R the new root; **return**
  end
  Let $P = parent(N)$
  if P has less than n pointers **then**
    Insert K', N' into P, after N
  else begin
    Copy P to T
    Insert K', N' into T, after N
    Erase all from P; Create node P'
    Copy $T.P_1 \ldots T.P_{\lceil n/2 \rceil}$ to P
    Let $K'' = T.K_{\lceil n/2 \rceil}$
    Copy $T.P_{\lceil n/2 \rceil + 1} \ldots T.P_{n+1}$ to P'
    insert-in-parent(P, K'', P')
  end

Insertion Example



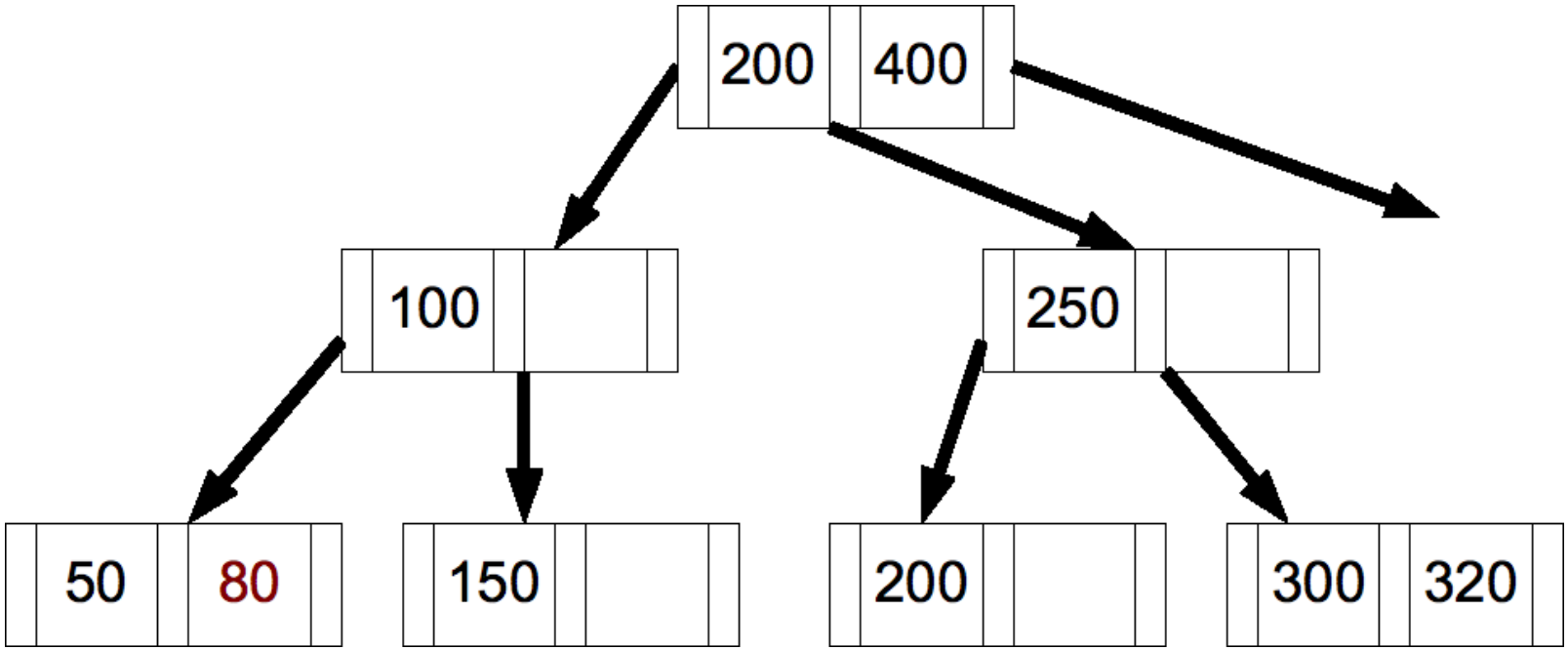
Find the leaf node L that should contain K

Insertion Example



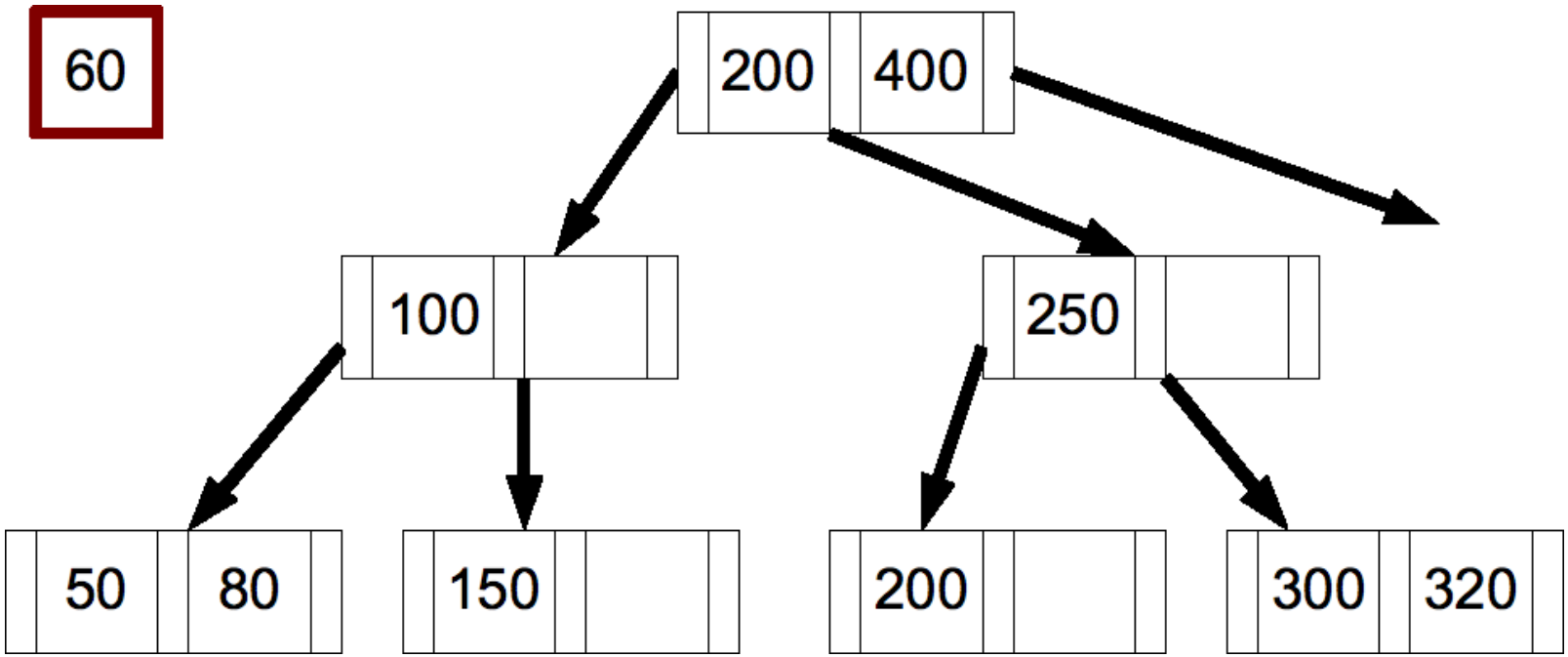
Find the leaf node L that should contain K

Insertion Example



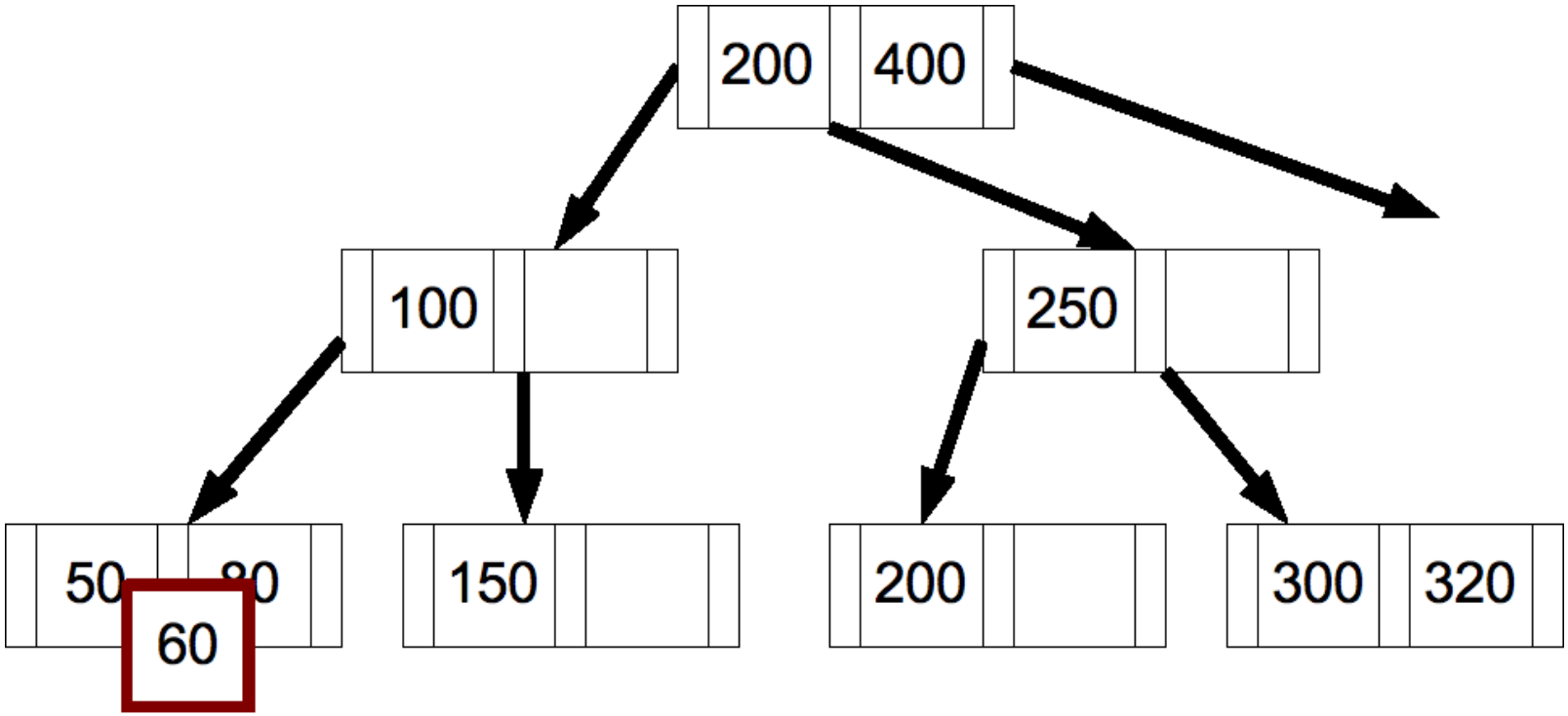
if L has less than $n - 1$ values **then**
insert-in-leaf(L, K, P)

Insertion Example



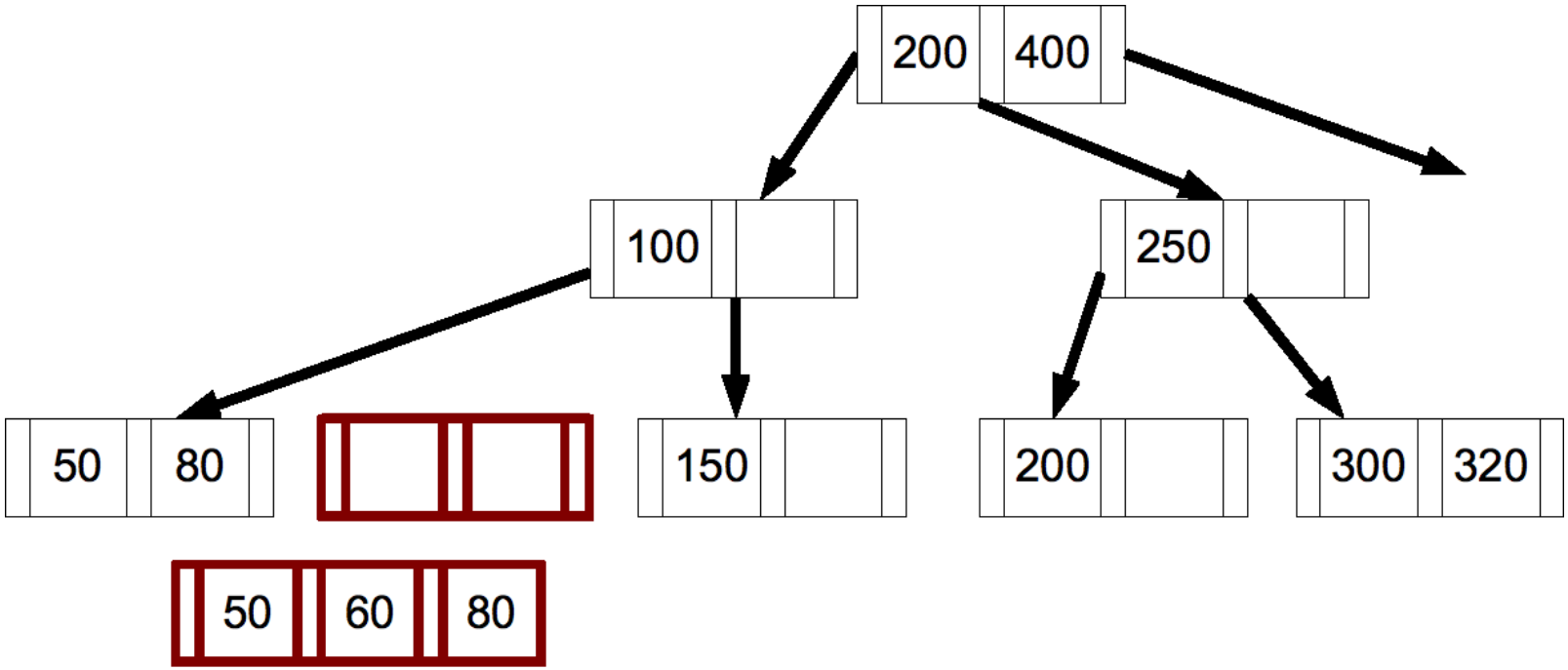
Find the leaf node L that should contain K

Insertion Example



Find the leaf node L that should contain K

Insertion Example

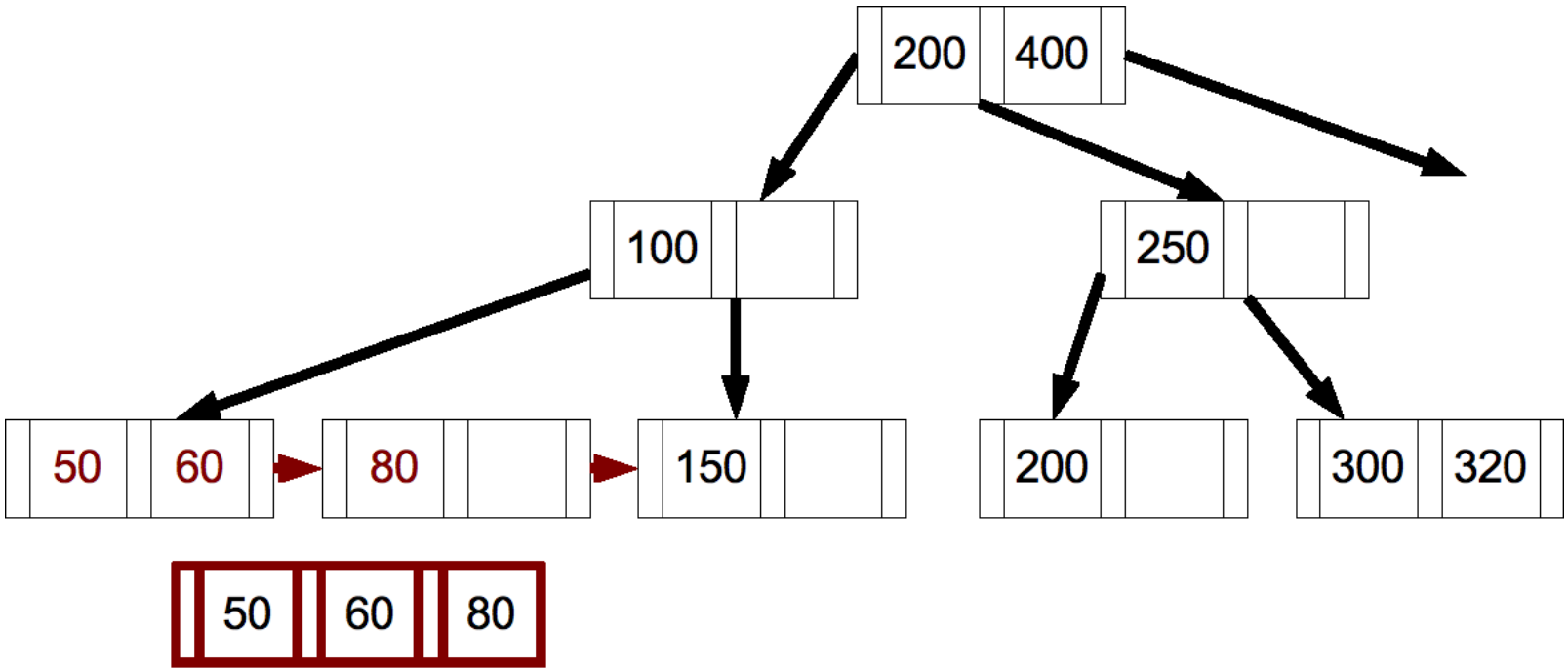


Create node L'

Copy $L.P_1 \ldots L.K_{n-1}$ to T (an auxiliary node)

insert-in-leaf(T, K, P)

Insertion Example



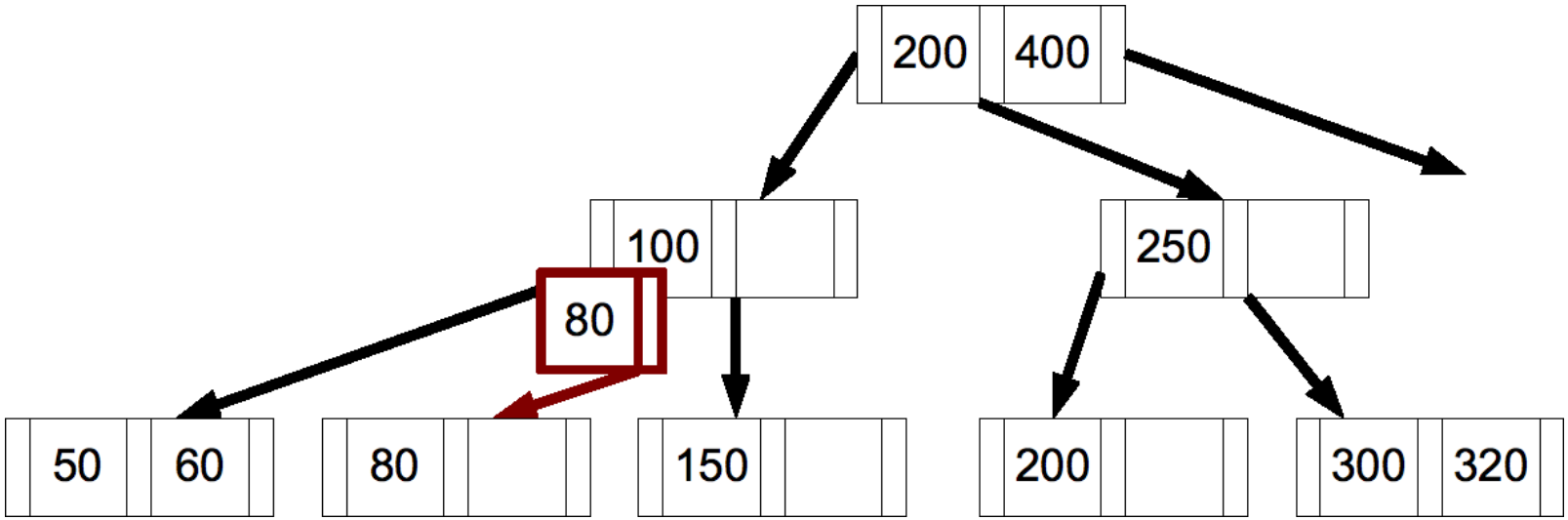
Set $L'.P_n = L.P_n$; Set $L.P_n = L'$

Erase $L.P_1 \ldots L.K_{n-1}$ from L

Copy $T.P_1 \ldots T.K_{\lceil n/2 \rceil}$ to $L.P_1 \ldots$

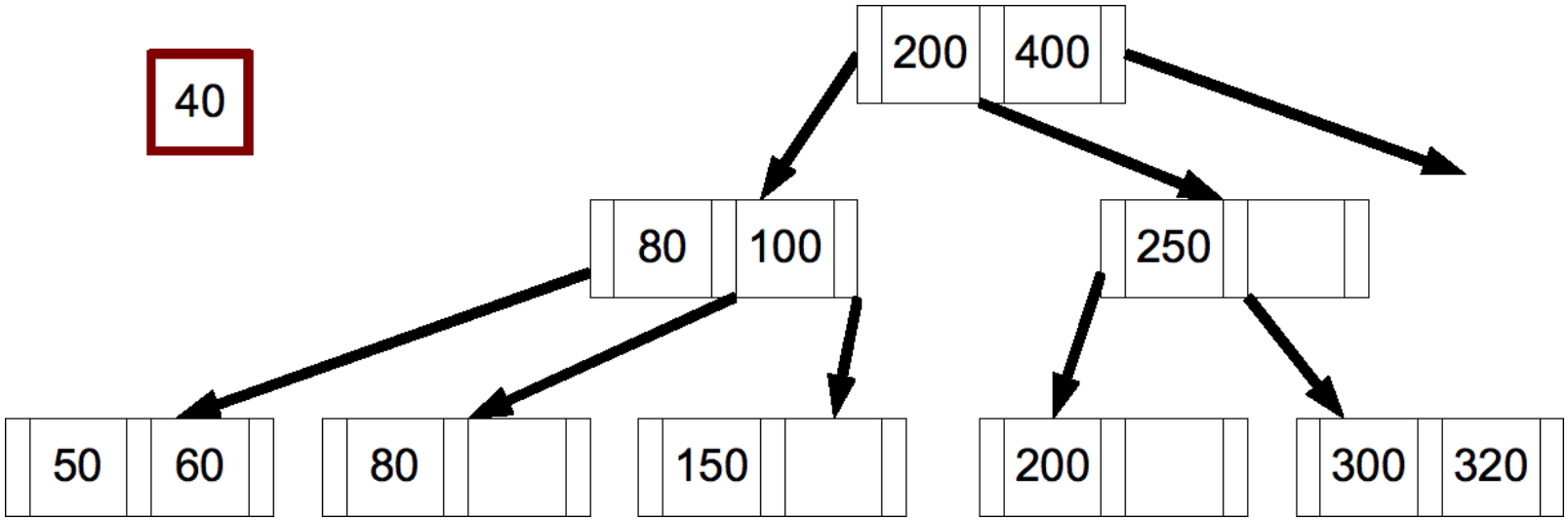
Copy $T.P_{\lceil n/2 \rceil + 1} \ldots T.K_n$ to $L'.P_1 \ldots$

Insertion Example



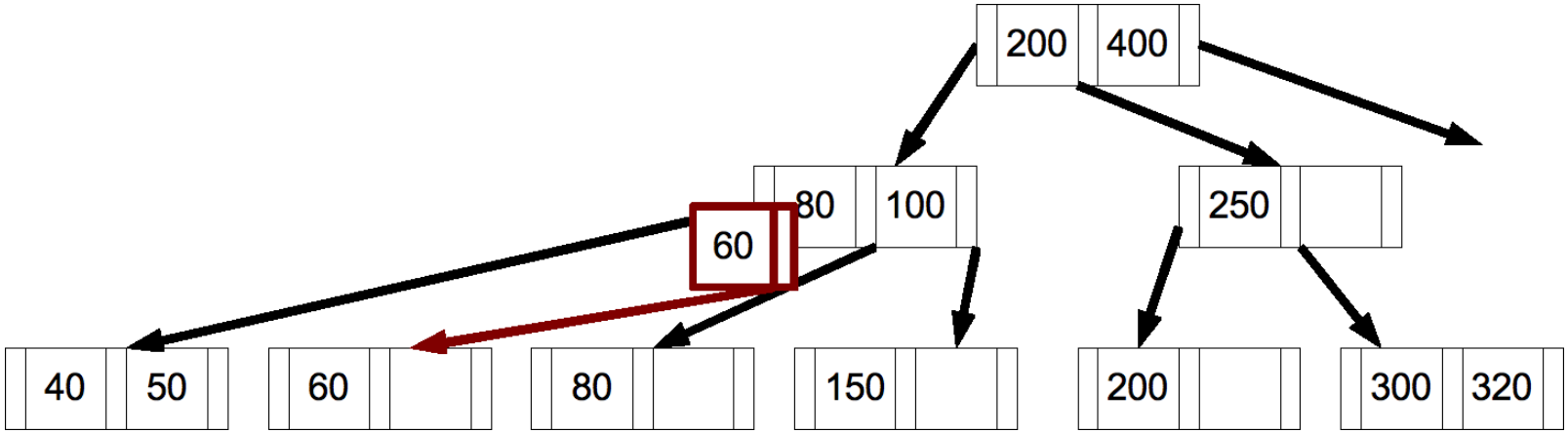
Let K' be the smallest key in L'
insert-in-parent(L, K', L')

Insertion Example



Find the leaf node L that should contain K

Insertion Example

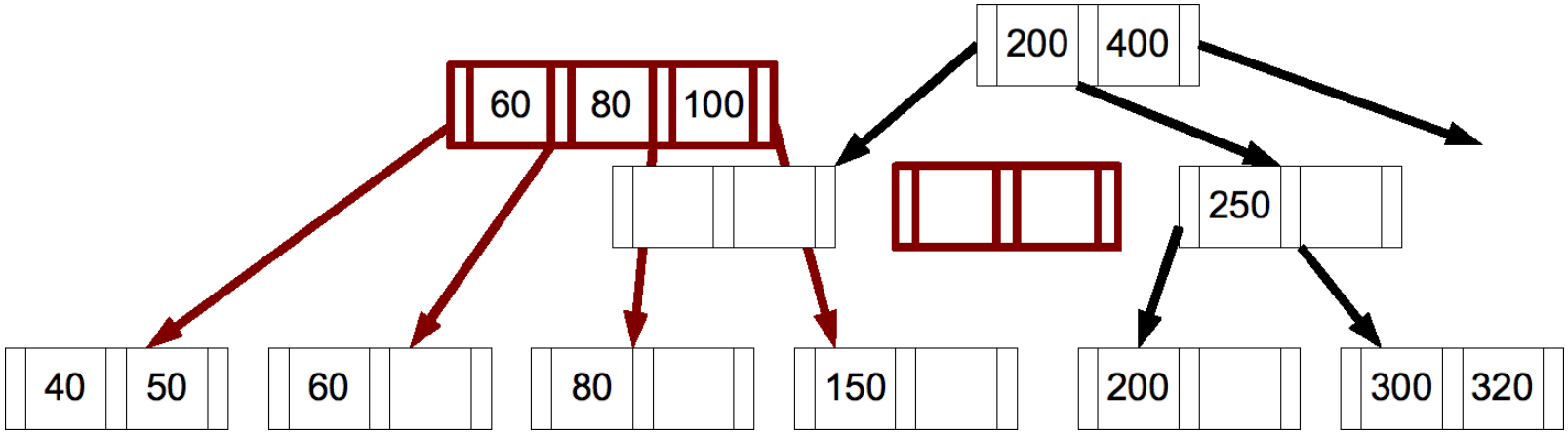


Create node L'

...

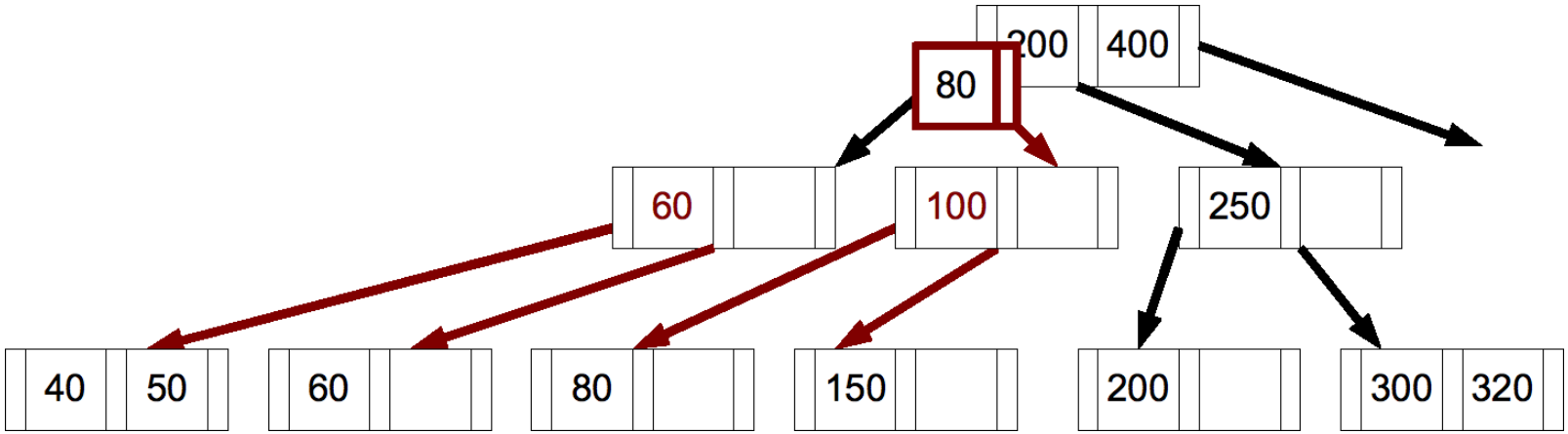
insert-in-parent(L, K', L')

Insertion Example



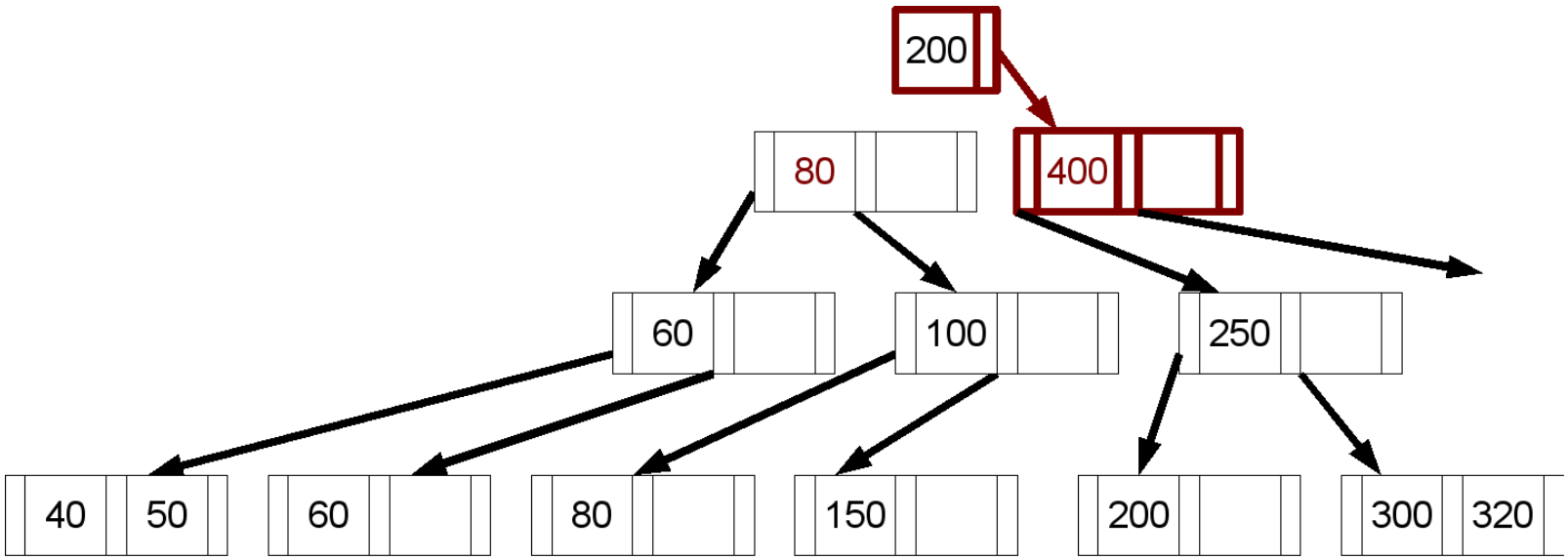
Let $P = parent(L)$
Copy P to T
Insert K', L' into T, after L
Erase all from P; Create node P'

Insertion Example



Copy $T.P_1 \ldots T.P_{\lceil n/2 \rceil}$ to P
Let $K'' = T.K_{\lceil n/2 \rceil}$
Copy $T.P_{\lceil n/2 \rceil + 1} \ldots T.P_{n+1}$ to P'
insert-in-parent(P, K'', P')

Insertion Example

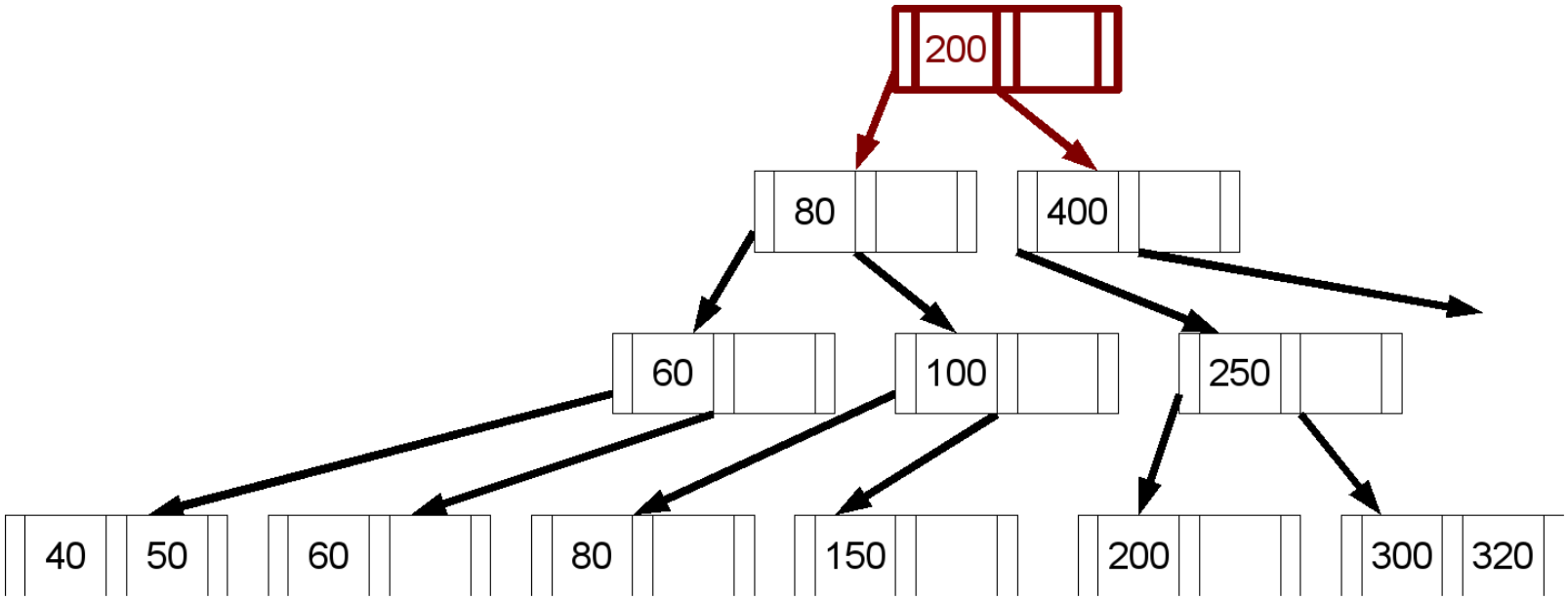


Let $P = parent(L)$

...

insert-in-parent(P, K'', P')

Insertion Example



if N is the root **then begin**

Create a new node $R = (N, K', N')$

Make R the new root; **return**

Outline

Procedure *Delete*

procedure delete(value V, pointer P)

Find the leaf node L that contains V, P

delete-entry(L, V, P)

Procedure *delete-entry*

procedure delete-entry(node L, value V, pointer P)
  delete V, P from L
  if L is root and has only 1 child **then begin**
    delete L
    make the child the new root
    return
  end
  if L has too few pointers/values **then**
  begin
    Let L' be the previous or next child of *parent*(L)
    Let V' be the value between L and L' in *parent*(L)
    if entries in L and L' fit in one node **then**
      coalesce-nodes(L, V', L')
    else
      redistribute-nodes(L, V', L')
  end

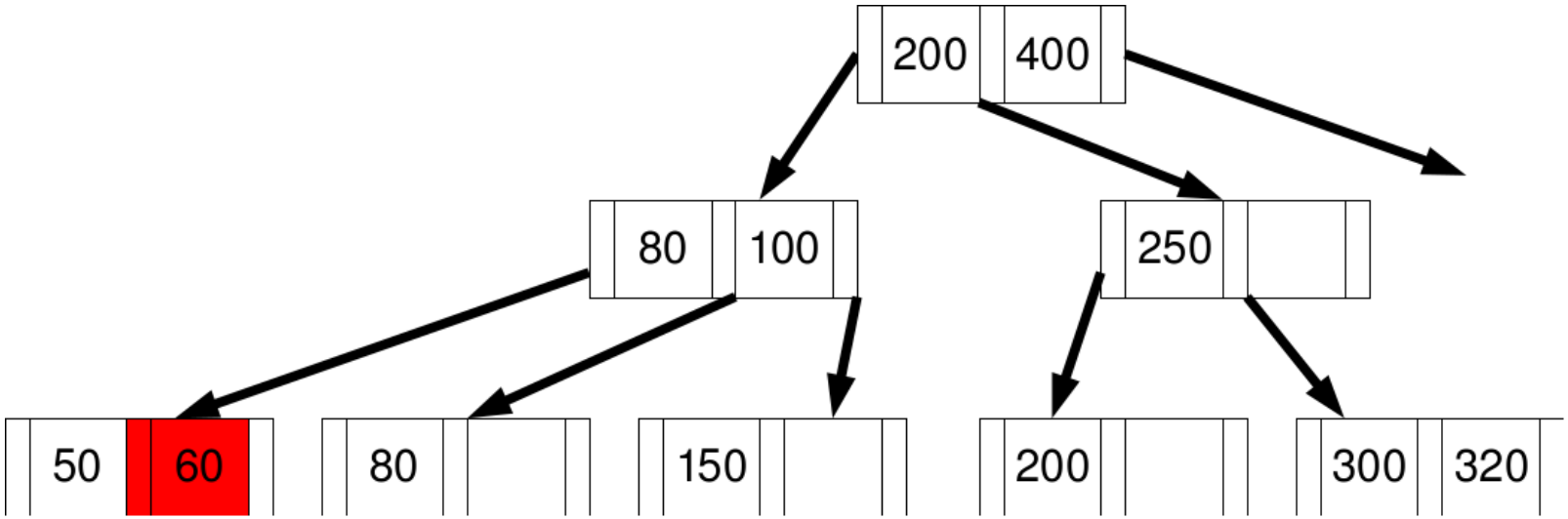
Procedure *coalesce-nodes*

procedure coalesce-nodes(node L, value V', node L')
  if L is a leaf **then**
  begin
    Append all (K_i, P_i) pairs in L to L'
    Set $L'.P_n$ to $L.P_n$, if needed
  end
  else
    Append V' and all pointers/values in L to L'
  delete-entry($parent(L), V', L$)
  delete node L

Procedure *redistribute-nodes*

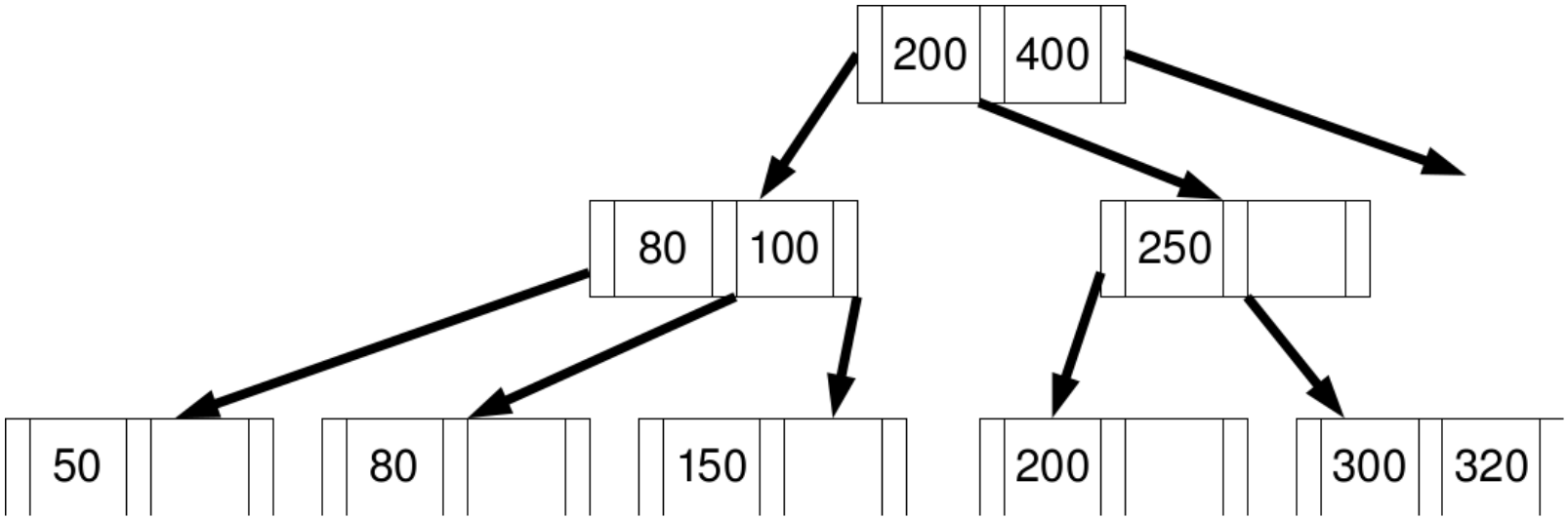
procedure redistribute-nodes(node L, value V', node L')
  if L' is a predecessor of L **then begin**
    if L is a non-leaf node **then begin**
      Let $L'.P_m$ be the last pointer in L'
      Remove $L'.K_{m-1}, L'.P_m$ from L'
      Insert $L'.P_m, V'$ as the first pointer/value in L
      Replace V' in *parent*(L) by $L'K_{m-1}$
    end
    else begin
      Let $L'.P_m, L'.K_m$ be the last pointer/value pair in L'
      Remove $L'.P_m, L'.K_m$ from L'
      Insert $L'.P_m, L'.K_m$ as the first pointer/value in L
      Replace V' in *parent*(L) by $L'K_m$
    end
  end
  else . . . perform the symetric case . . .

Deletion Example



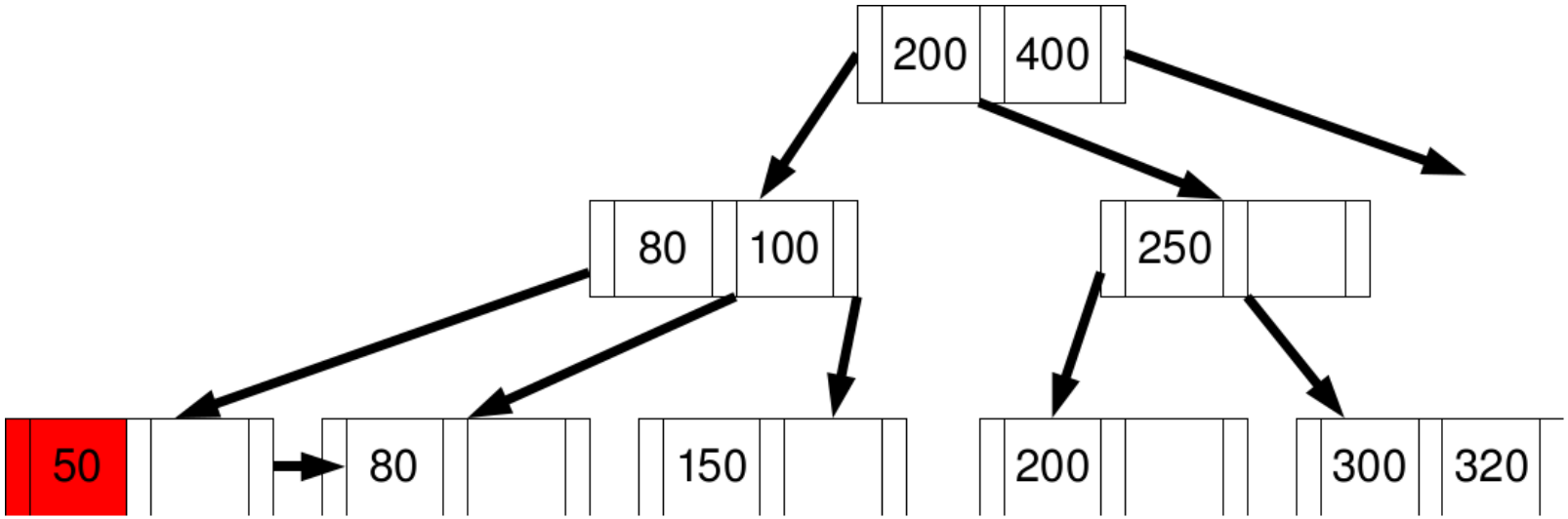
Find the leaf node L that contains V, P

Deletion Example



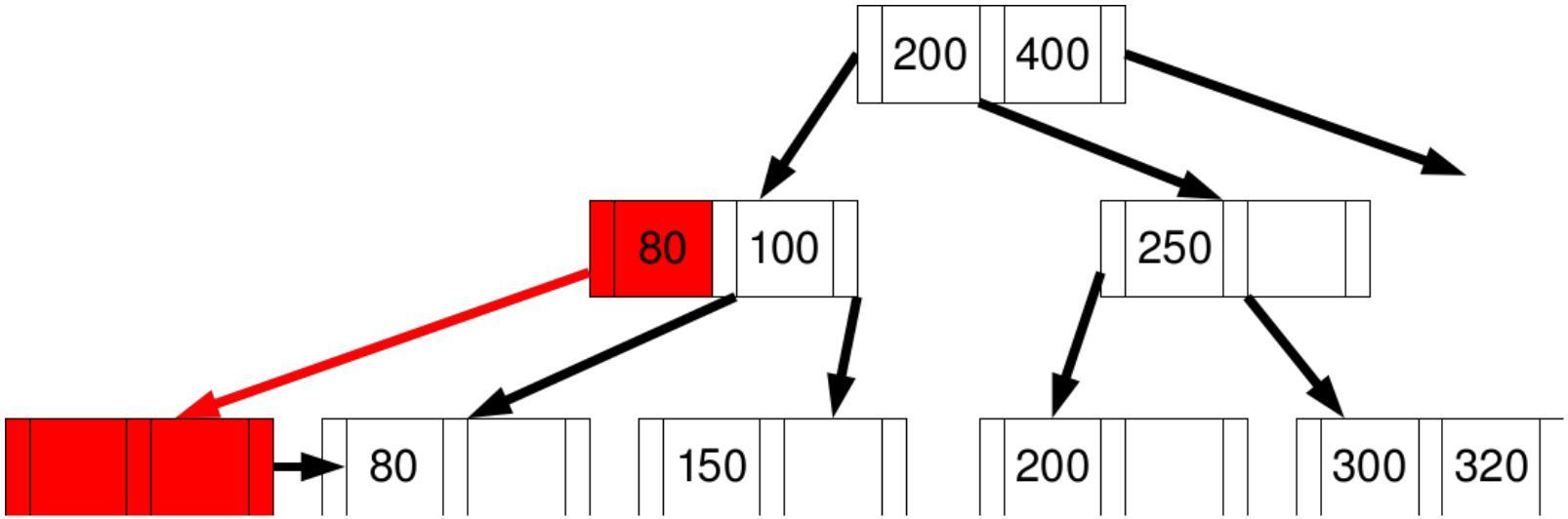
delete V, P from L

Deletion Example



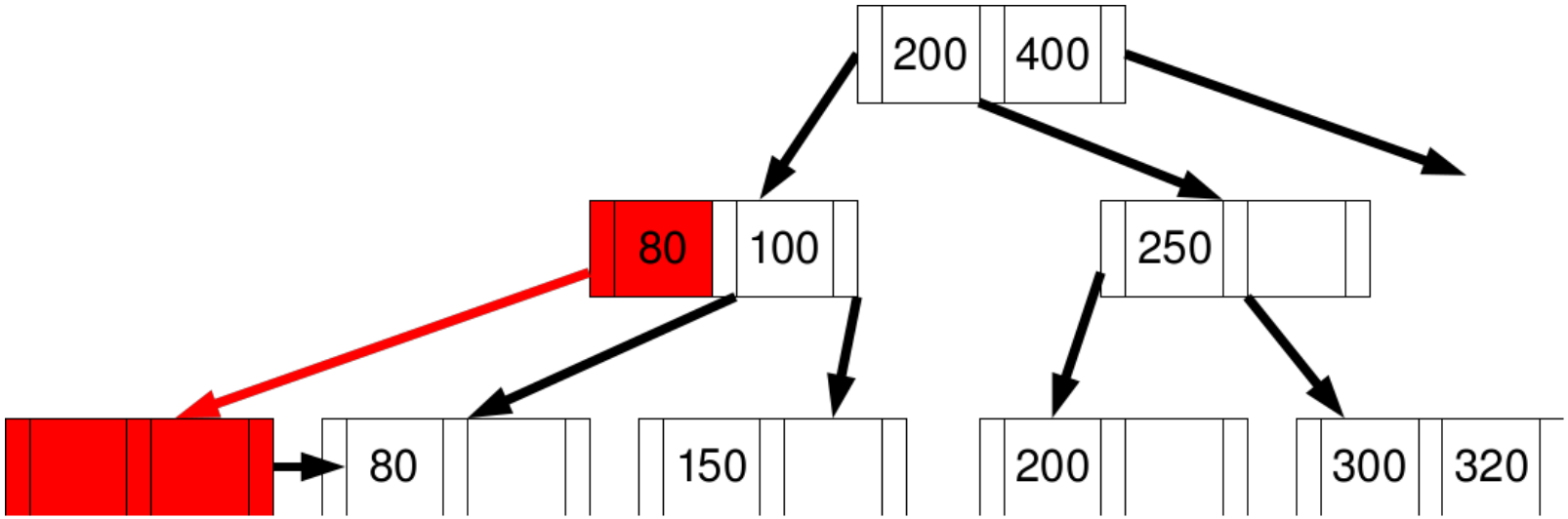
Find the leaf node L that contains V, P

Deletion Example



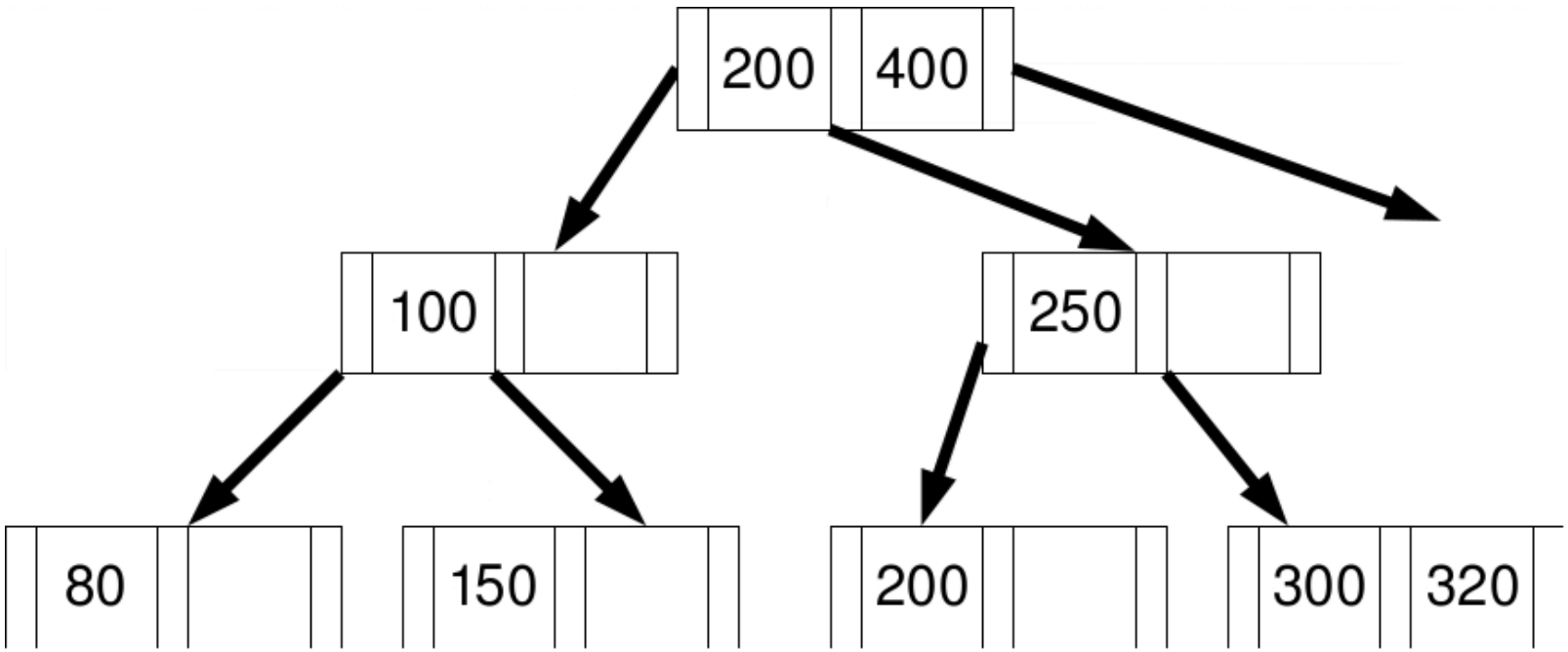
delete V, P from L
if L has too few pointers/values **then**
begin
    Let L' be the previous or next child of $parent(L)$
    Let V' be the value between L and L' in $parent(L)$
    if entries in L and L' fit in one node **then**
        coalesce-nodes(L, V', L')

Deletion Example



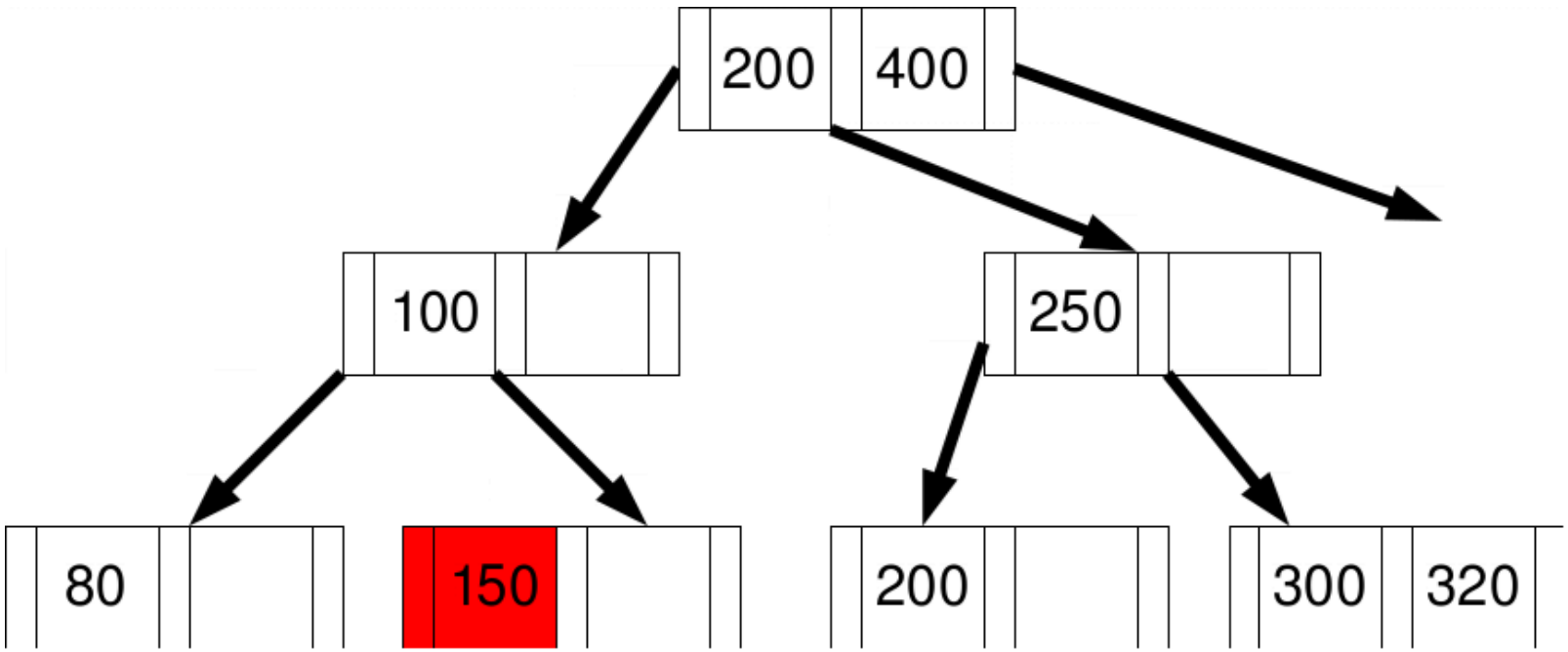
if L is a leaf **then**
begin
Append all (K_i, P_i) pairs in L to L'
delete-entry($parent(L), V', L$)

Deletion Example



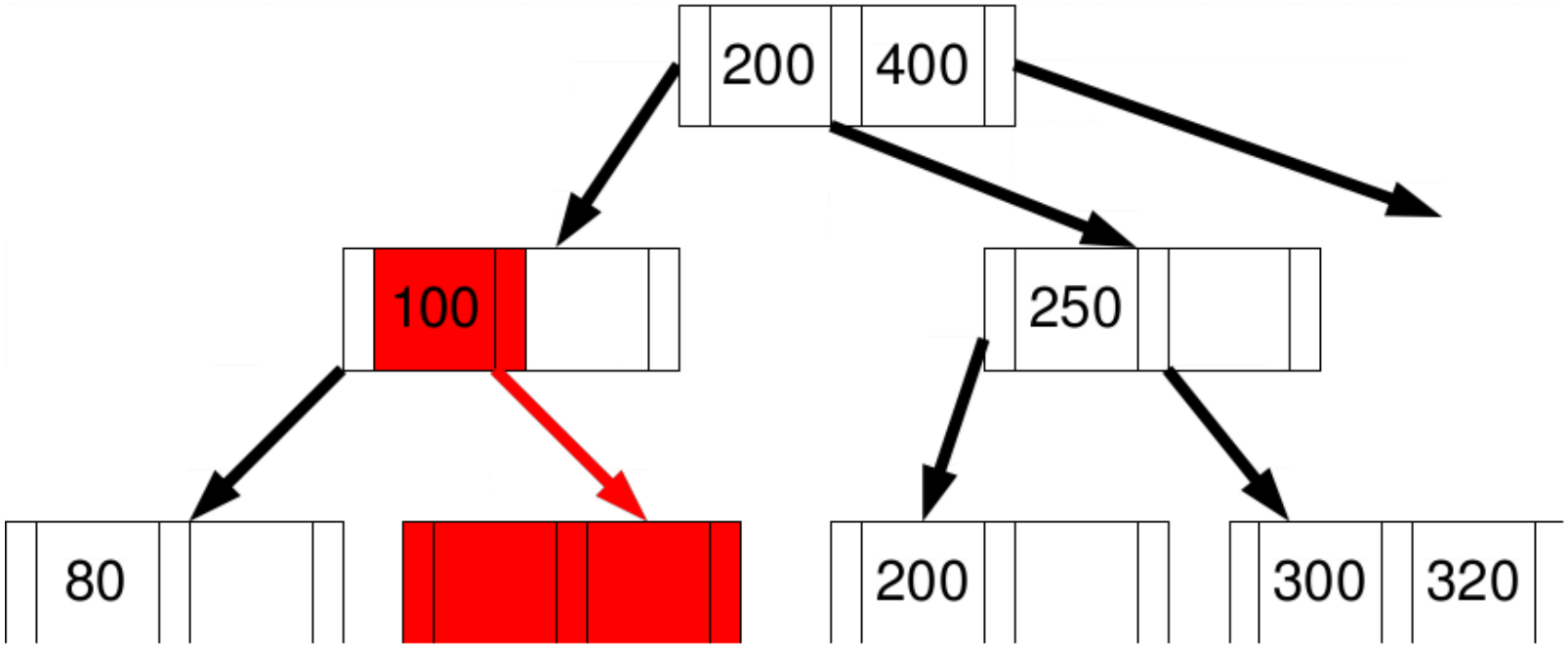
delete node L

Deletion Example



Find the leaf node L that contains V, P

Deletion Example



. . .

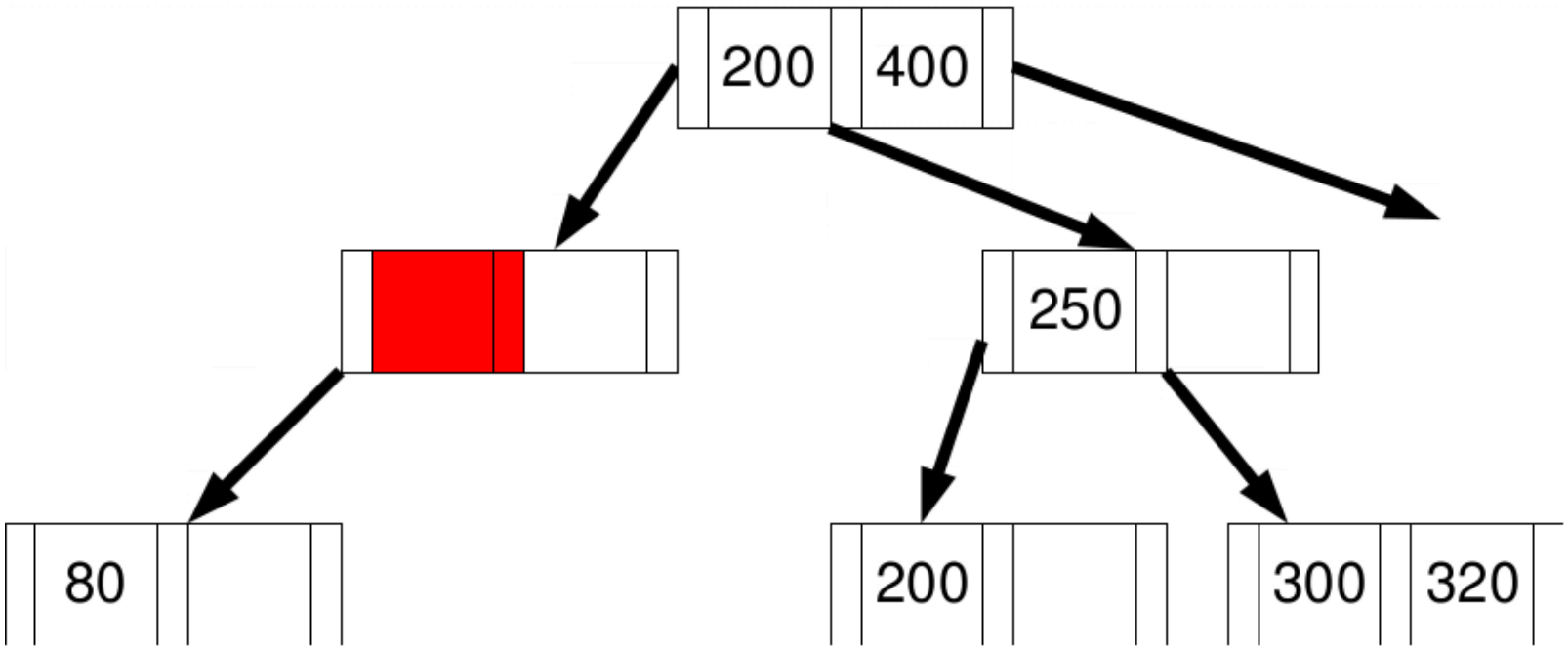
coalesce-nodes(L, V', L')

. . .

delete-entry($parent(L)$, V', L)

delete node L

Deletion Example

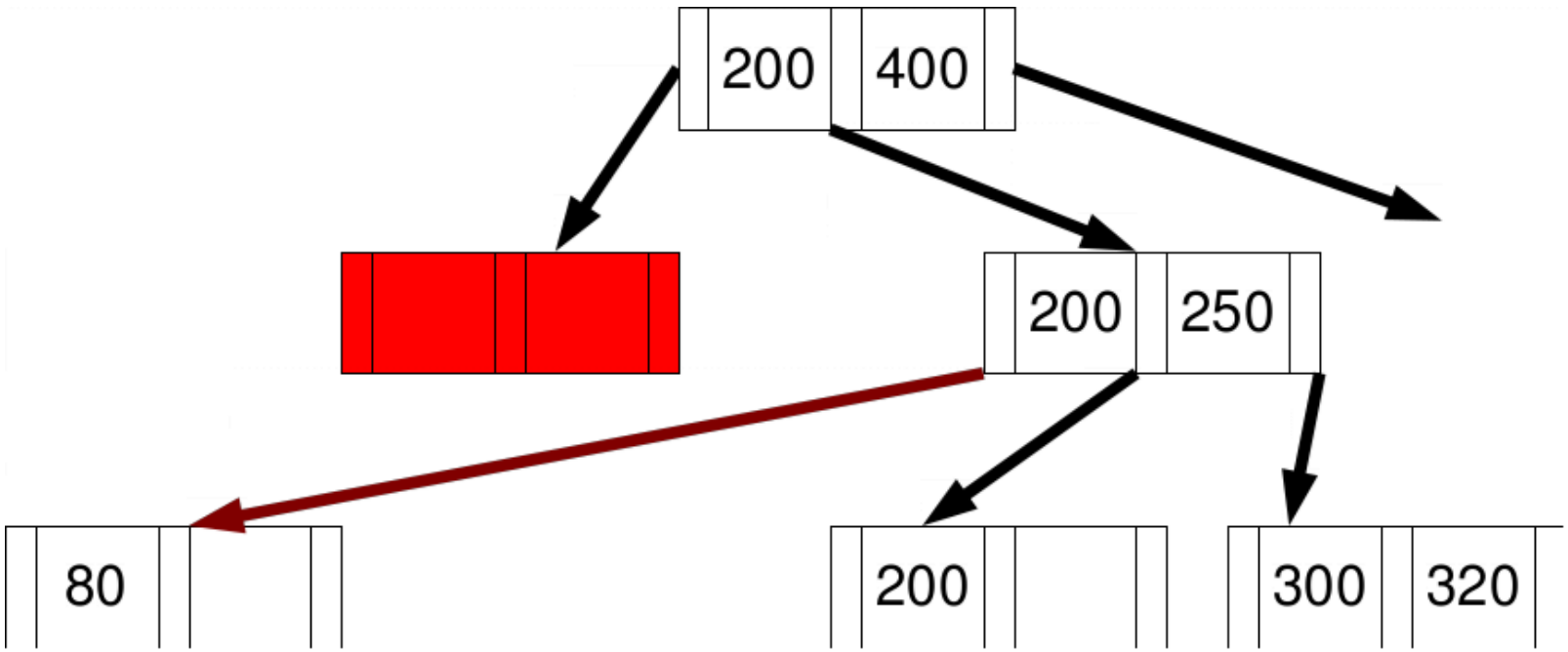


delete V, P from L

...

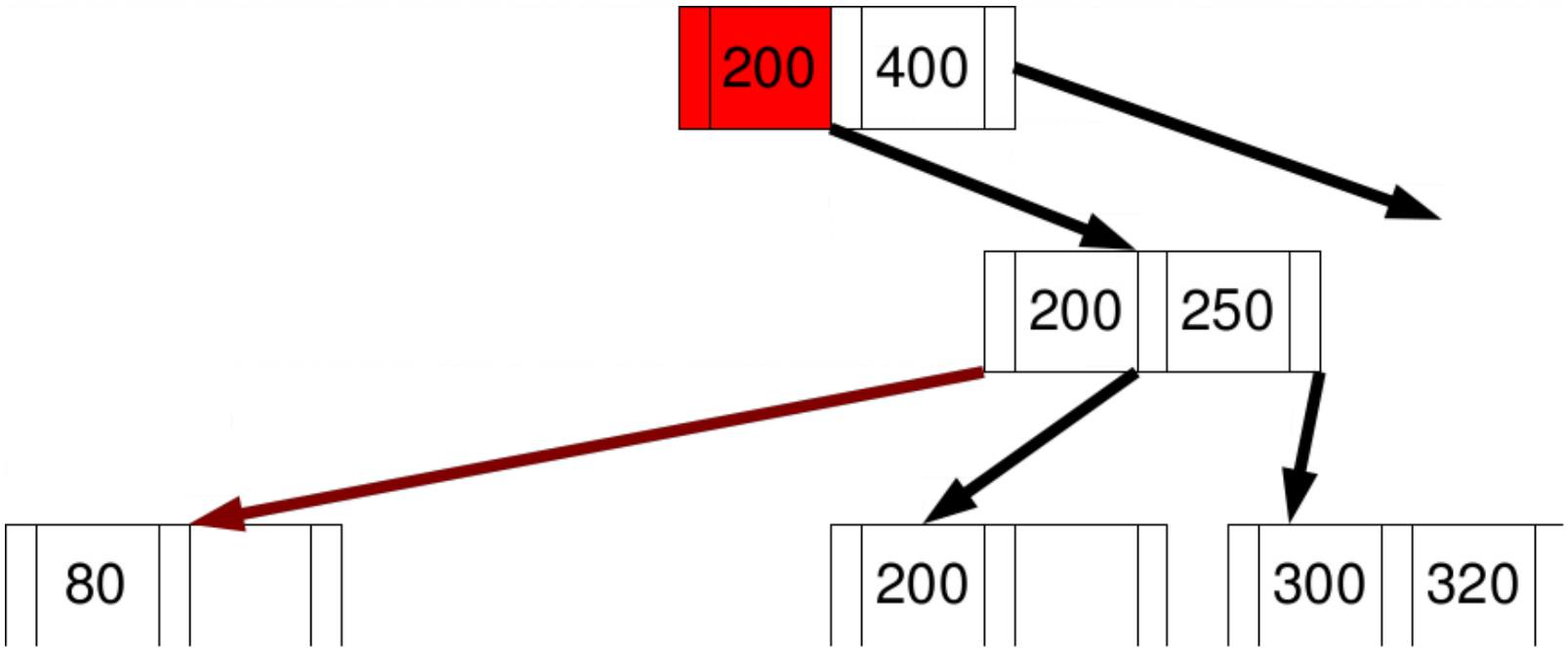
coalesce-nodes(L, V', L')

Deletion Example



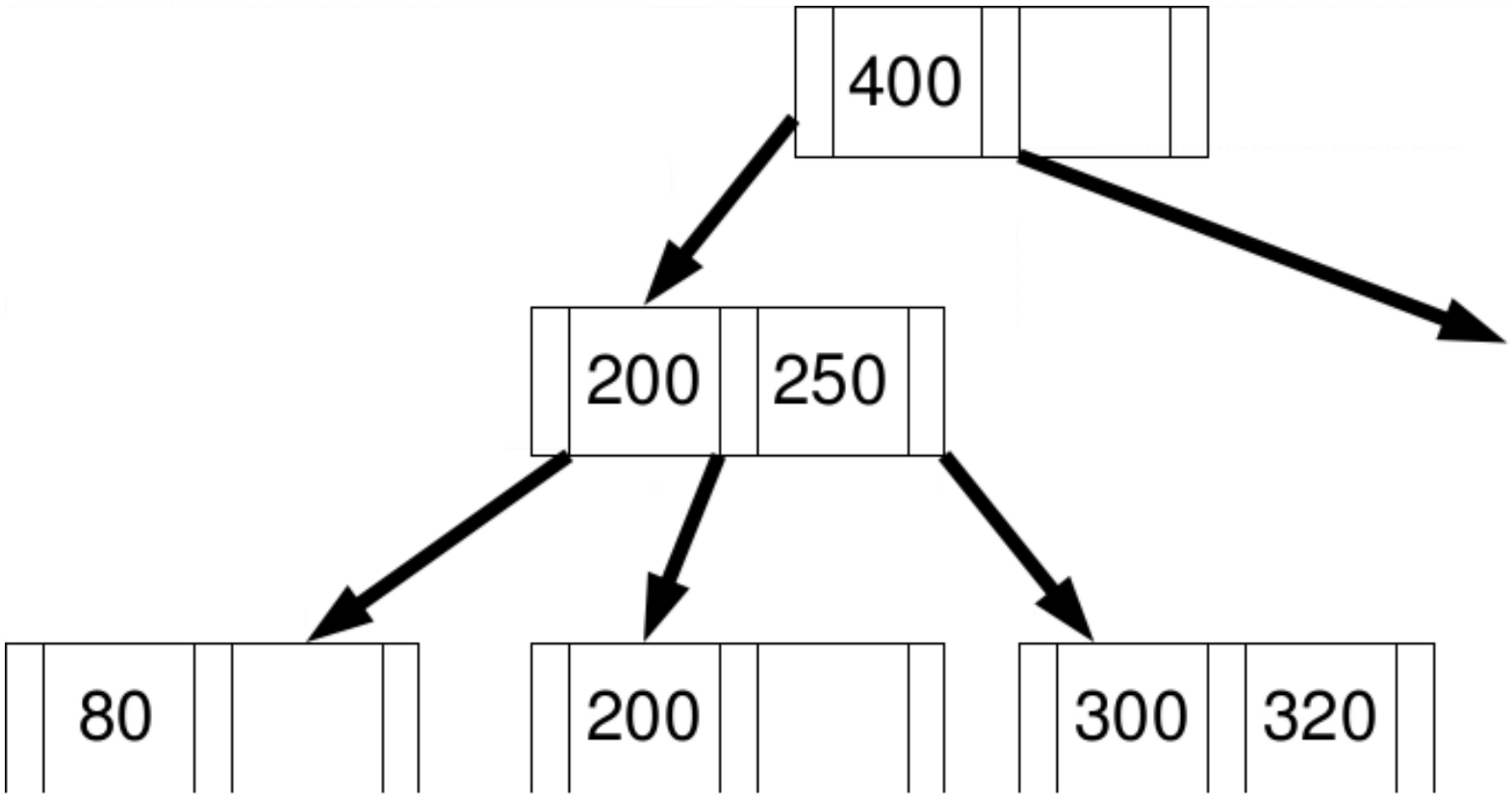
Append V' and all pointers/values in L to L'

Deletion Example



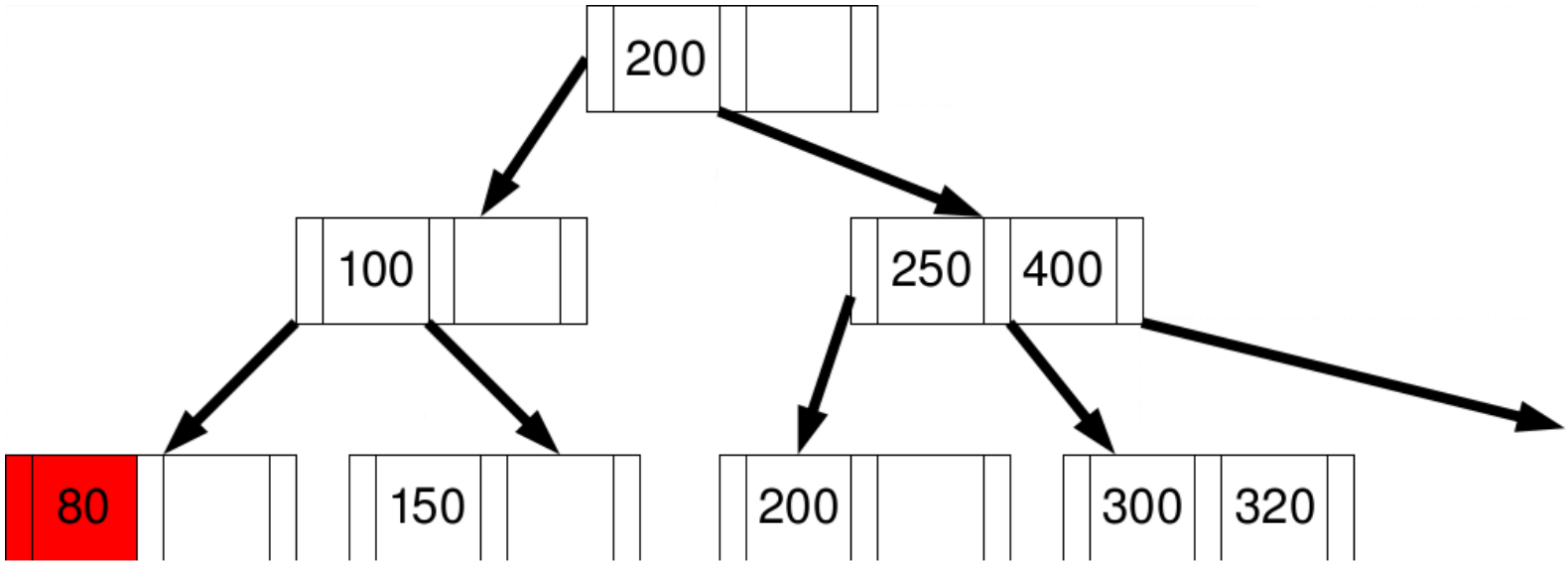
delete-entry($parent(L), V', L$)
delete node L

Deletion Example



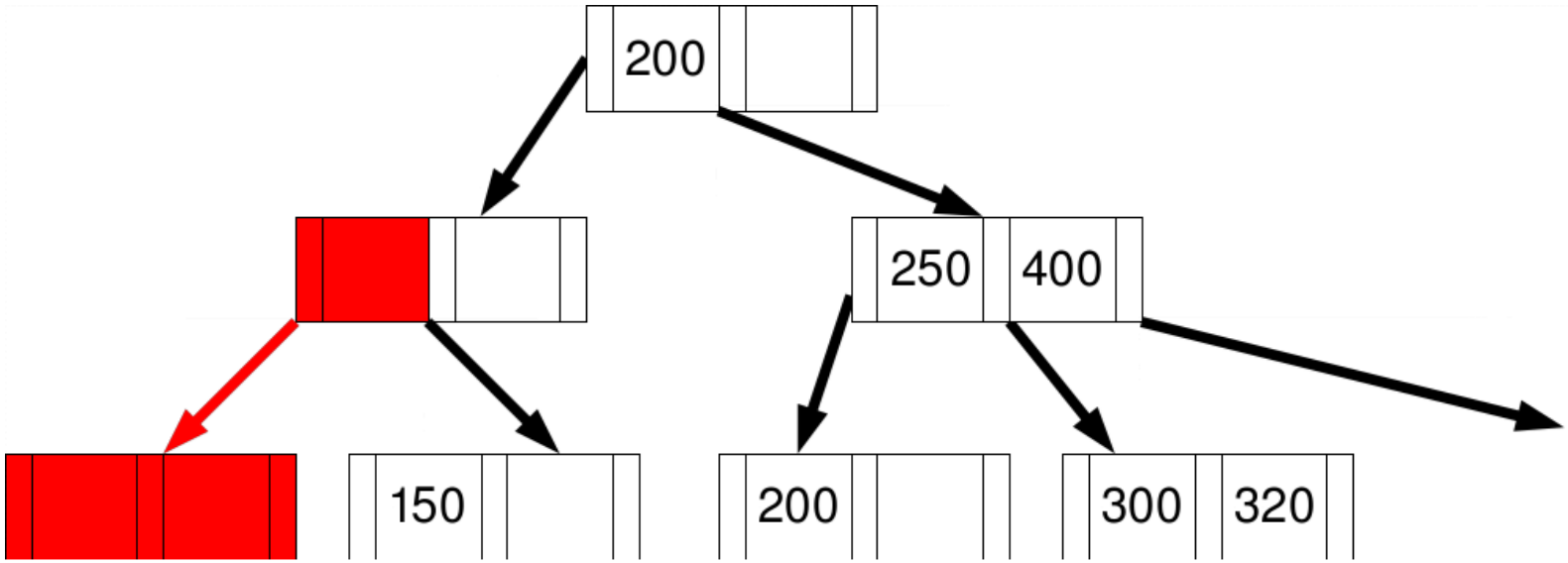
end

Deletion Example



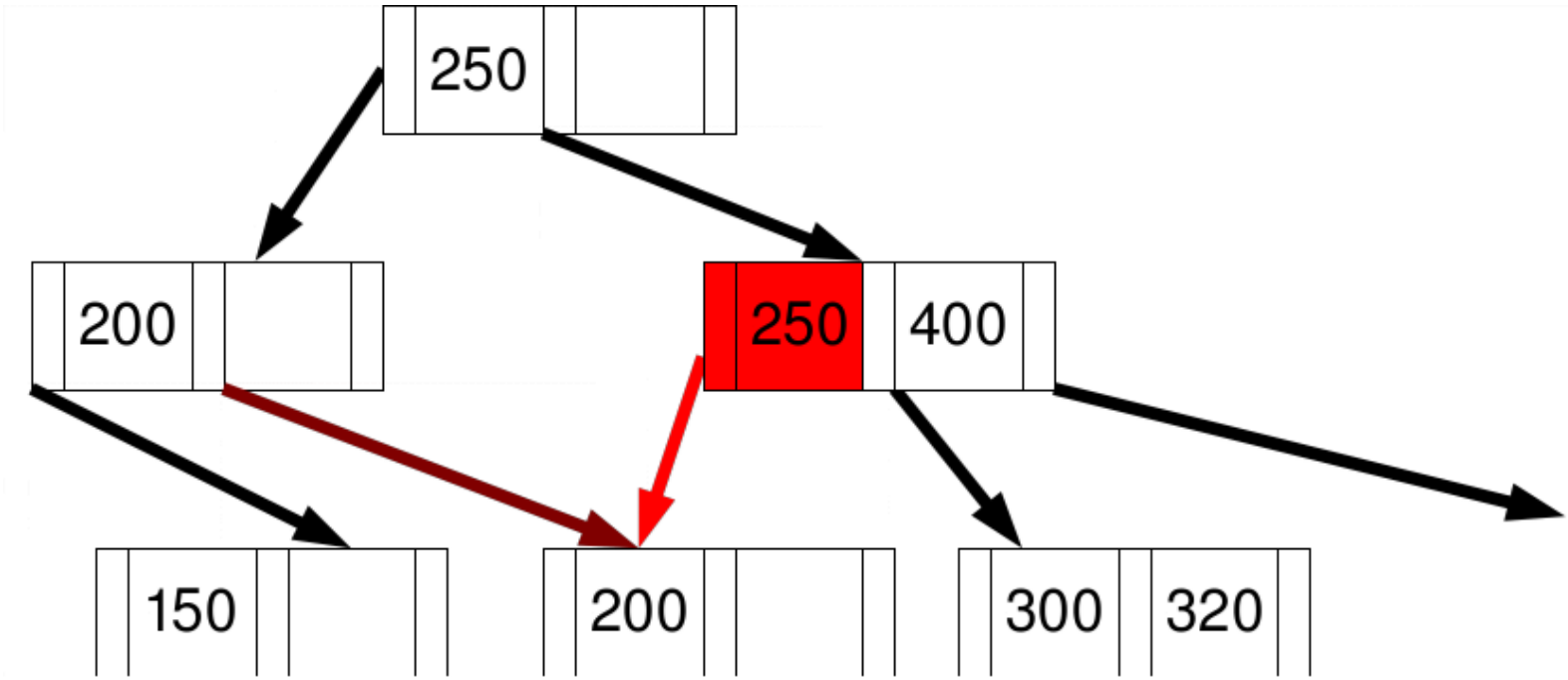
Find the leaf node L that contains V, P

Deletion Example



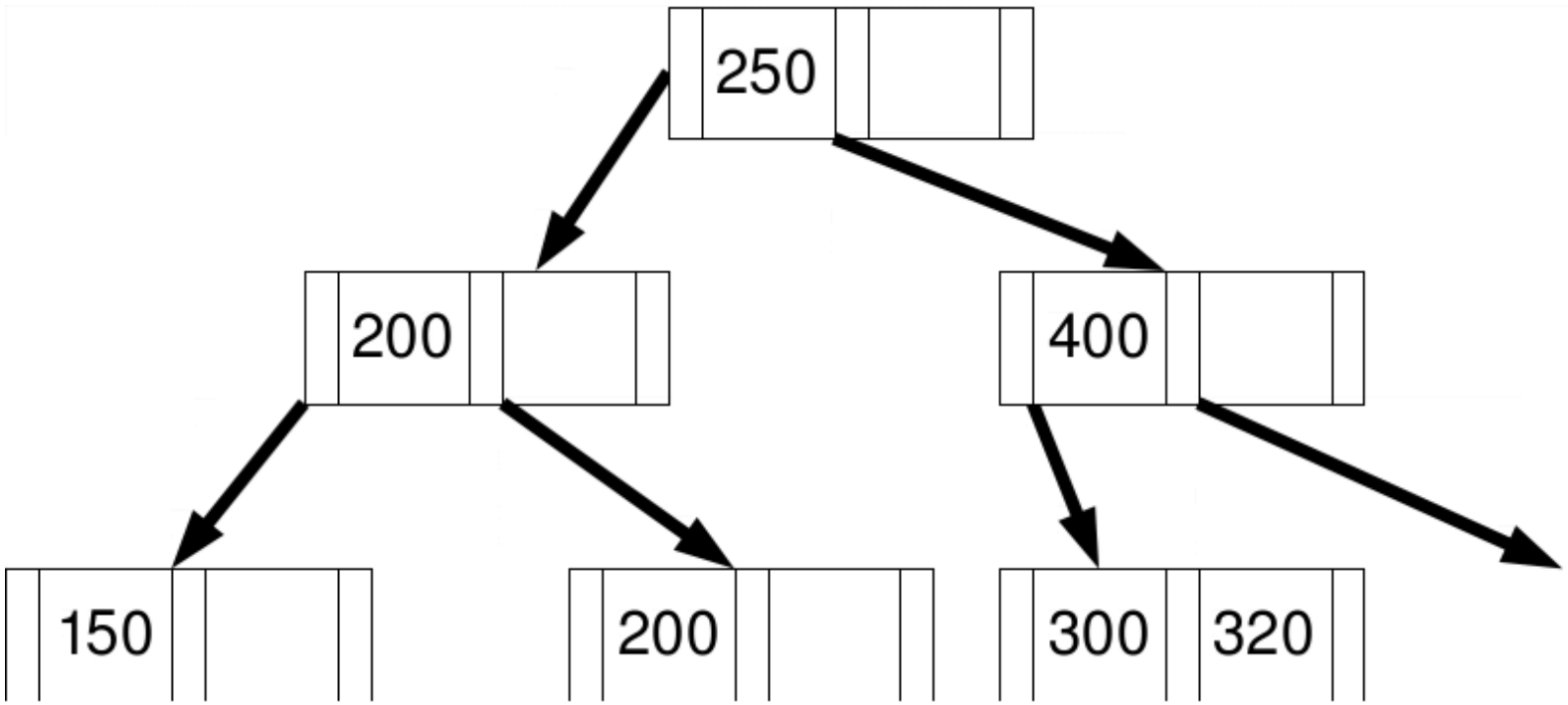
redistribute-nodes(L, V', L')

Deletion Example



Let $L'.P_m$ be the first pointer in L'
Insert $V', L'.P_m$ as the last pointer/value in L
Replace V' in $parent(L)$ by $L'K_{m+1}$

Deletion Example



Remove $L'.K_{m+1}, L'.P_m$ from L'

The End