Trees

M-ary Trees - M-ary Search Trees - B-Trees

D.E ZEGOUR

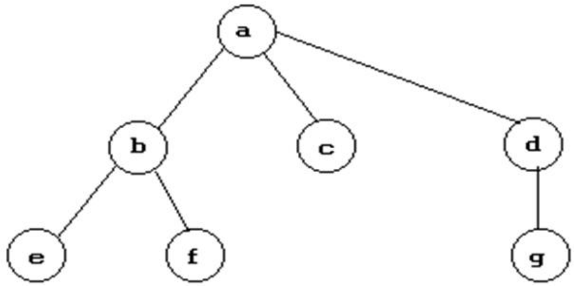
Ecole Supérieure d'Informatique

ESI

M-ary Trees

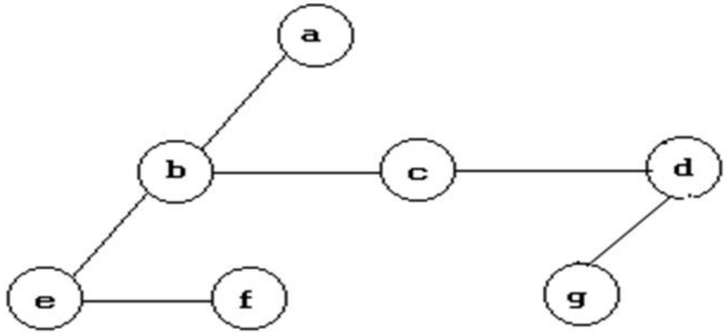
Definition

Generalisation of the binary tree



Node structure:

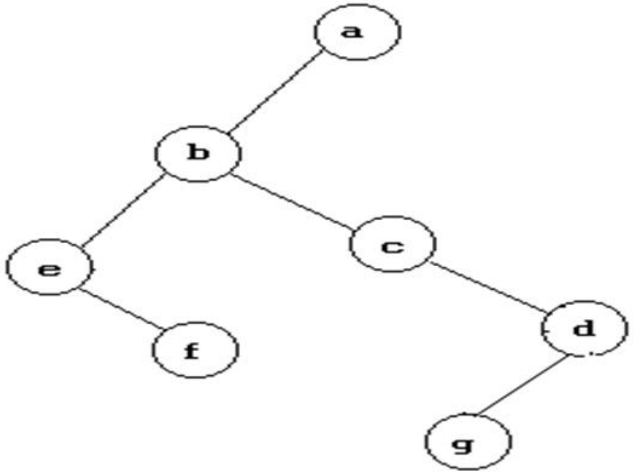
$$(k_1, s_1, s_2 \,.......,s_n)$$



Link the brothers

No order relation.

Can be transformed to a binary tree



Rotation 45%

M-ary Trees

Node Structure :
(k_1, s_1, s_2 ....,s_n)

Abstract machine

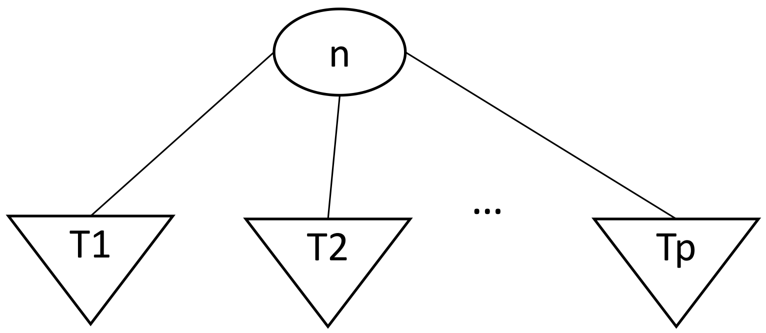
ALLOCATE_NODE (Val) : Create a node with value Val and return the address of the node. The other fields are to Nil.
FREE_NODE (P) : Free the node of address P.

CHILD (P, I) : Accessing the I-th child of the node referenced by P.
PARENT (P) : Accessing the field Parent of the node referenced by P.
NODE_VALUE_MST (P) : Accessing the value of the node referenced by P.
DEGREE(P) : Provide the number of values stocked inside the node referenced by P.

ASS_CHILD (P, I, Q) : Assign the address Q to the I-th child of the node referenced by P.
ASS_PARENT (P, Q) : Assign the address Q to the field Parent of the node referenced by P.
ASS_NODE_VAL_MST(P, Val) : Assign the value Val to value field of the node referenced by P.
ASS_DEGREE(P, n) : Set the Degree field of the node referenced by P to the value n.

M-ary Trees

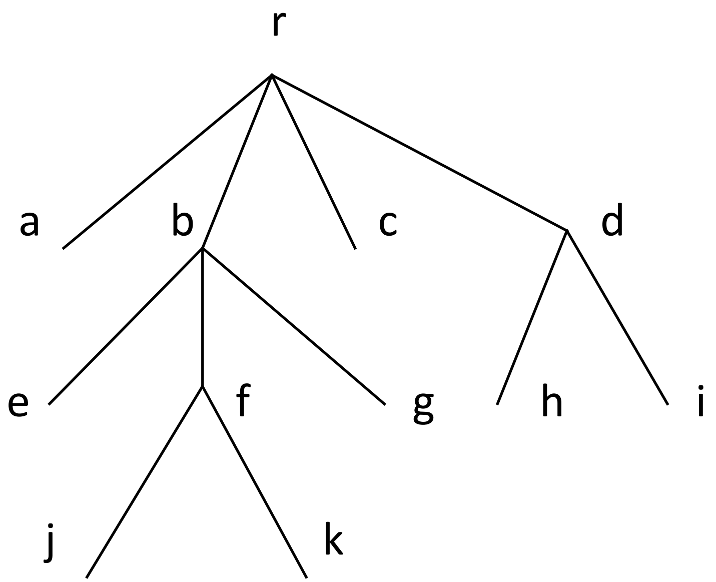
Traversal



Preorder : n T1 T2 .. . Tp
(r a b e f j k g c d h i)

Inorder : T1 n T2 ....Tp
(a r e b j f k g c h d i)

Postorder : T1 T2 ... Tp n
(a e j k f g b c h i d r)



M-ary SearchTrees

Definition

Generalisation of the binary search tree

Node structure :

$$(s_1, k_1, s_2, ...k_{n-1}, s_n)$$

$k_1 < k_2 .... < k_{n-1}$

(Elements in s_1) $< k_1$

(Elements in s_j) $> k_{j-1}$ and $< k_j$

($j=2,3, ...n-1$)

(Elements in s_n) $> k_{n-1}$

TOP-DOWN M-ary search tree: Any non-fulled node must be a leaf

B-arbre : Balanced M-ary search tree

M-ary SearchTrees

Abstract machine

Node structure :

$$(s_1, k_1, s_2, \ldots\ldots k_{n-1}, s_n)$$

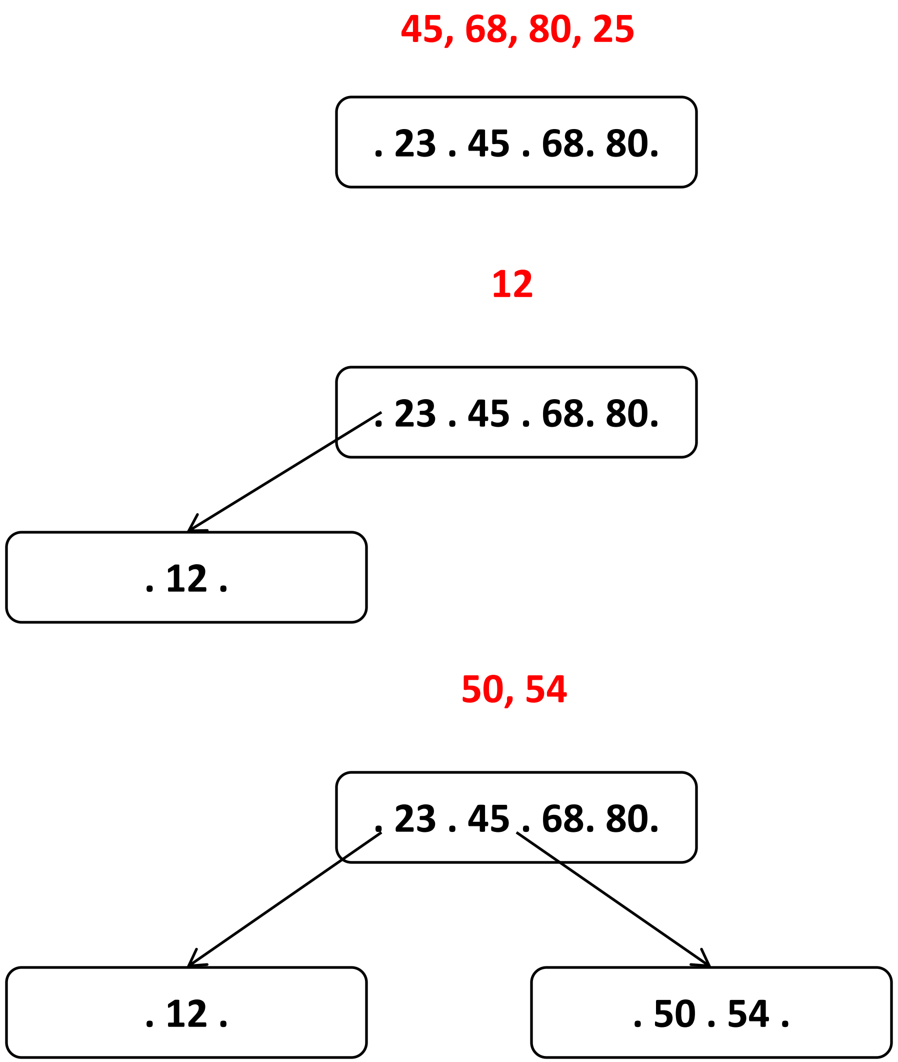
ALLOCATE_NODE (Val) : Create a node with value Val and return the address of the node. The other fields are to Nil.
FREE_NODE (P) : Free the node of address P.

CHILD (P, I) : Accessing the I-th child of the node referenced by P.
PARENT (P) : Accessing the field Parent of the node referenced by P.
NODE_VALUE_MST (P, I) : Accessing the I-th value of the node referenced by P.
DEGREE(P) : Provides the number of values stocked inside the node referenced by P.

ASS_CHILD (P, I, Q) : Assign the address Q to the I-th child of the node referenced by P.
ASS_PARENT (P, Q) : Assign the address Q to the field Parent of the node referenced by P.
ASS_NODE_VAL_MST(P, I, Val) : Assign the value Val to the I-th value of the node referenced by P.
ASS_DEGREE(P, n) : Set the Degree field of the node referenced by P to the value n.

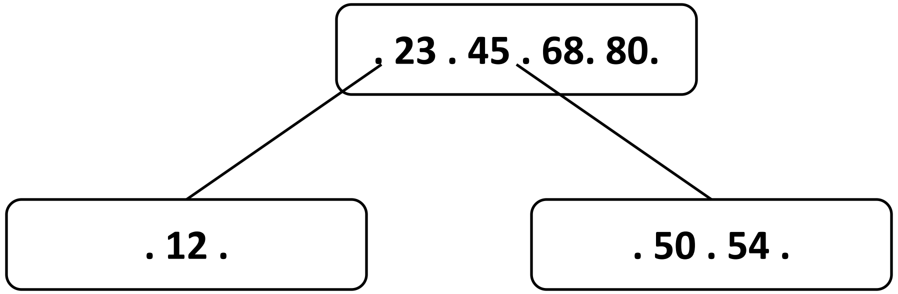
M-ary SearchTrees

TOP-DOWN M-ary Search Tree
Order=5
Insertion mechanism

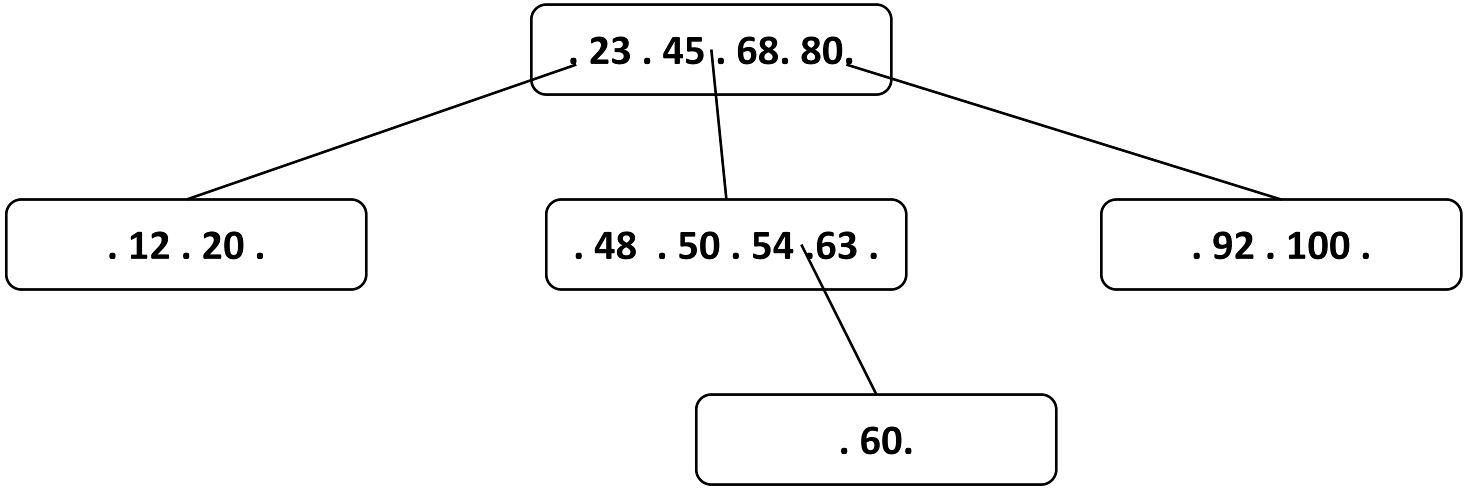


M-ary SearchTrees

TOP-DOWN M-ary Search Tree
Order=5
Insertion mechanism



20, 92, 48, 63, 60, 100



M-ary SearchTrees

TOP-DOWN M-ary Search Tree
Deletion Mechanism

Search for data d to delete
1. If data d exists on a leaf N:
   - Delete it
   - If d was the only data, free node N
   - Stop

2. If data d exists on an internal node N:
   2.a) If d has a non-empty left (or right) subtree:
      - Replace d with its successor (or predecessor) p from node N'
      - Set d:=p and N:=N', Go to 1

2.b) If data d does not have a non-empty left (or right) subtree:

- Take a data d' from node N (on the right or left) that has a left or right subtree (it must exist)

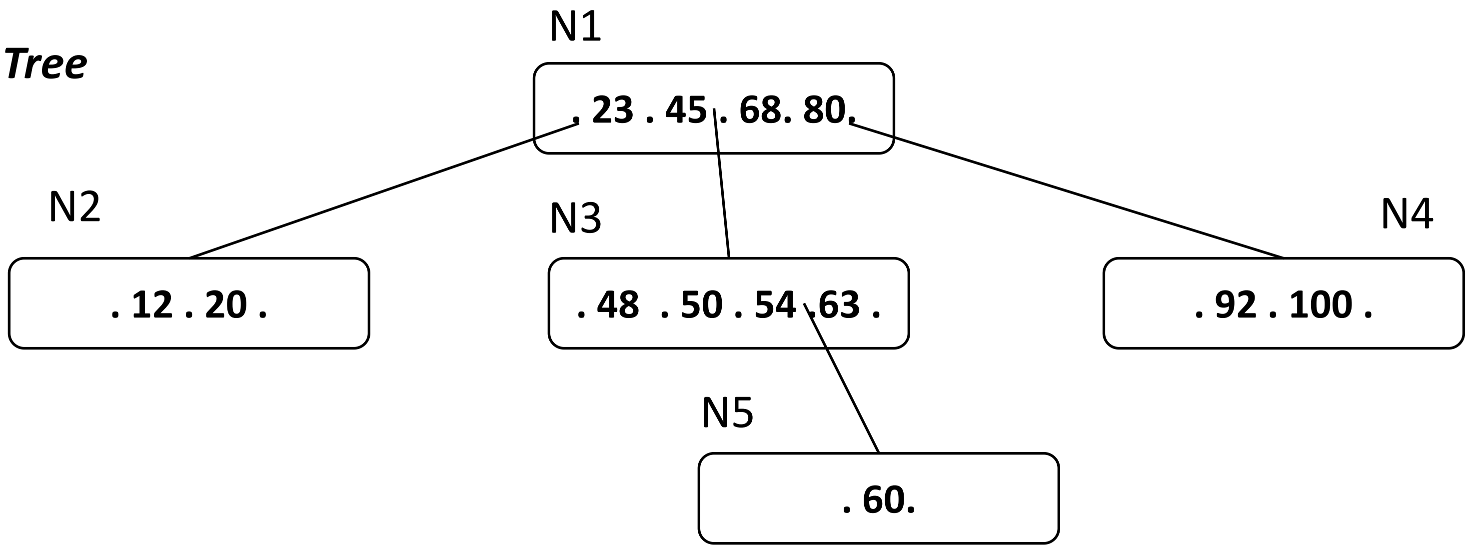
- Find the successor (predecessor) p' of d' from node N'

- Delete d from node N and add p' to node N by shifting

- Set d:=p' and N:=N', Go to 1

M-ary SearchTrees

TOP-DOWN M-ary Search Tree
Order=5
Deletion mechanism

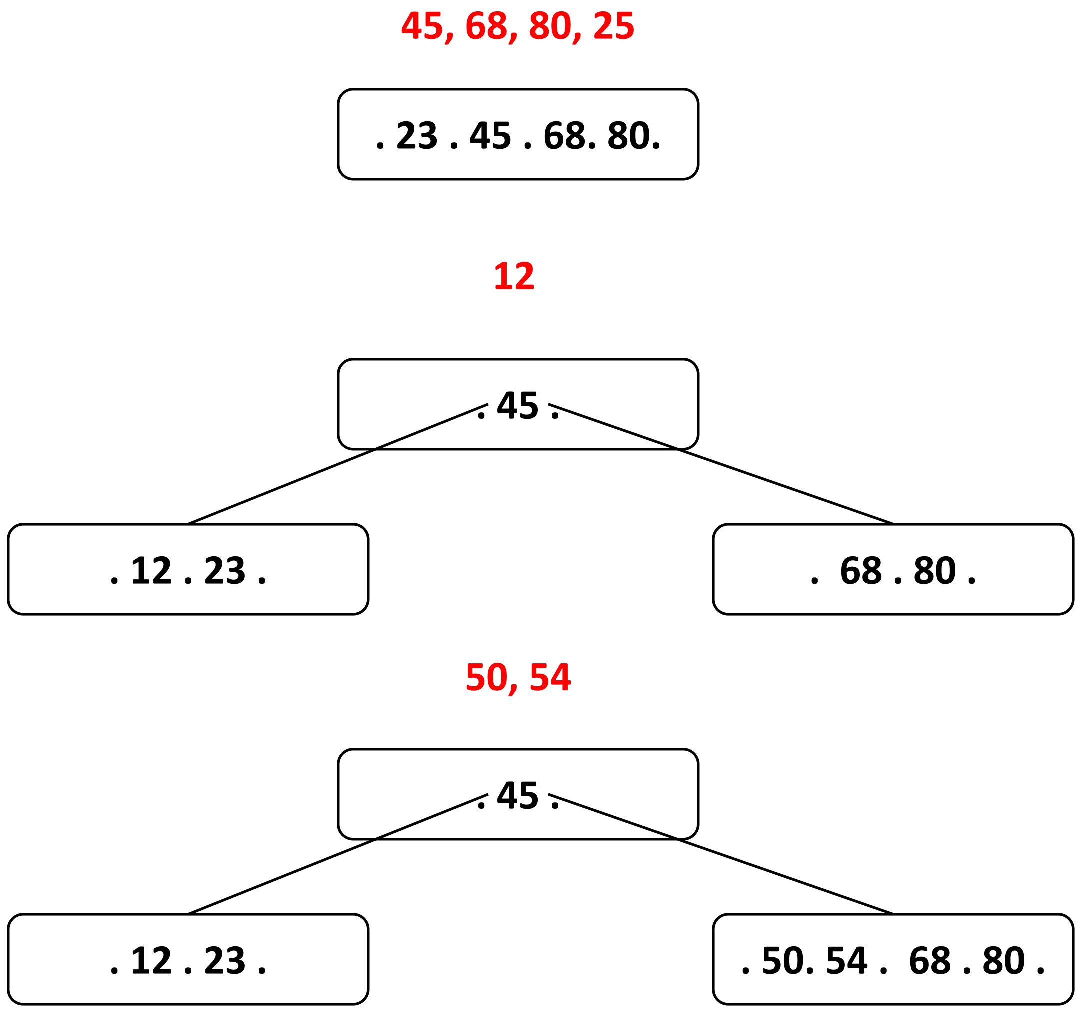


Deleting 92 :
→ N4 holds 100
Deleting 80 :
→ Replace 80 by 92 in N1, N4 holds 100
Deleting 60 :
→ Free N5
Deleting 45 :
→ Replace 45 by 48 in N1, move 60 to N3, Free N5

B-Trees

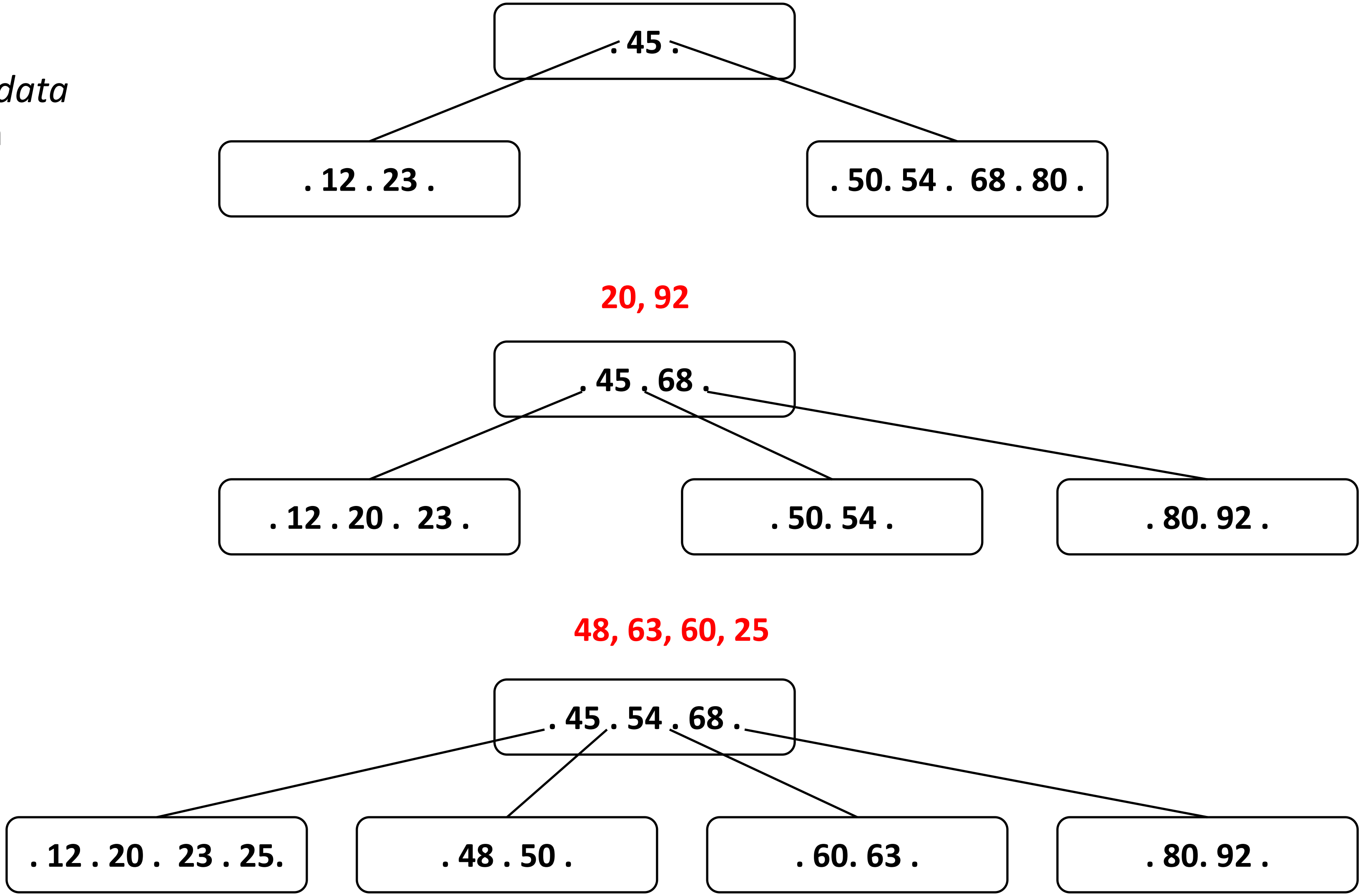
B-Tree(order=5)
Min=2 data; Max=4 data
*Insertion me*chanism

Definition (B-tree of order n):
- The root has at least two children.
- Each node (non-root) has at least n/2 children.
- All leaves are at the same level.



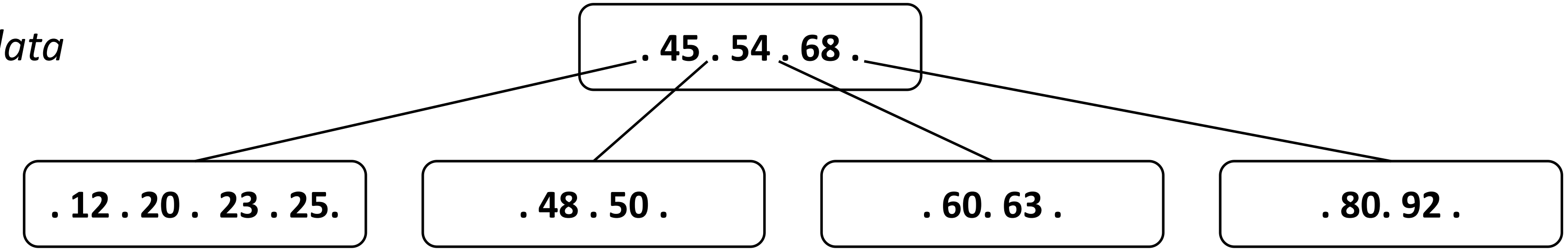
B-Trees

B-Tree(order=5)
Min=2 data; Max=4 data
Insertion mechanism

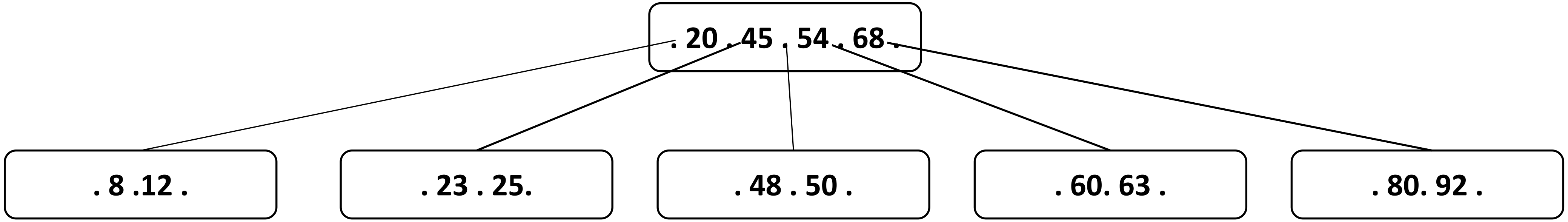


B-Trees

B-Tree(order=5)
Min=2 data; Max=4 data
Insertion mechanism

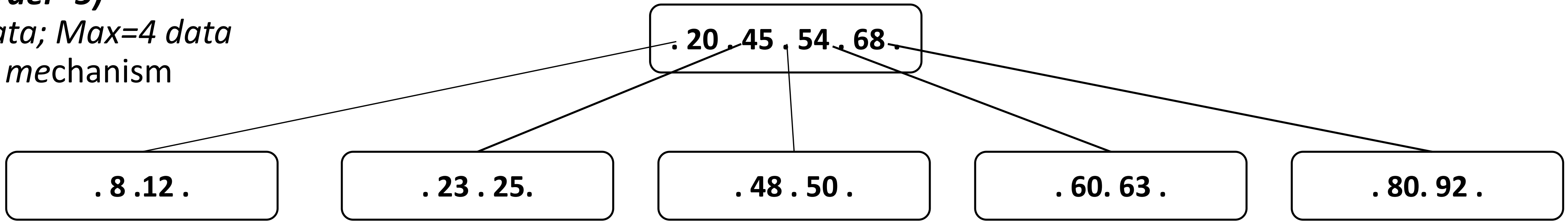


8

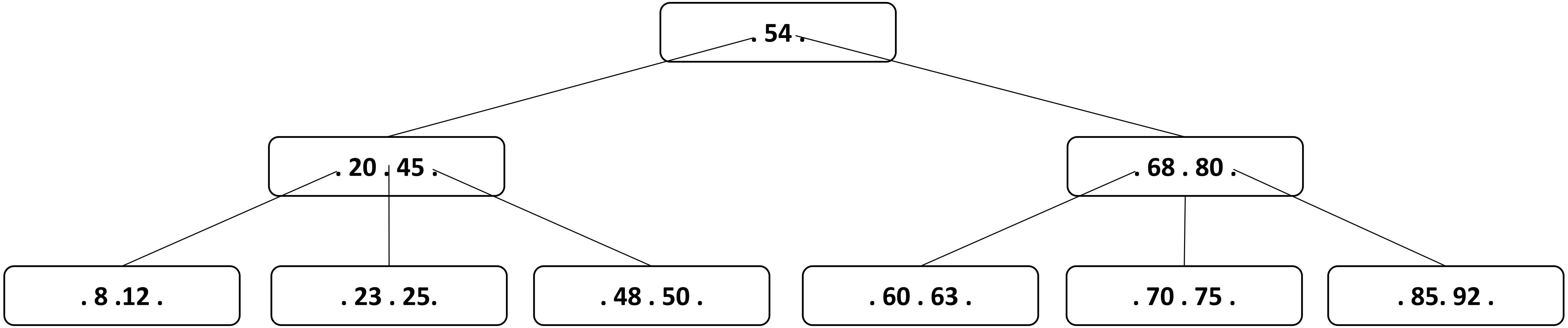


B-Trees

B-Tree(order=5)
Min=2 data; Max=4 data
Insertion mechanism



70, 75, 85



B-Trees

Deletion

Same principle as the physical deletion in an ARM.

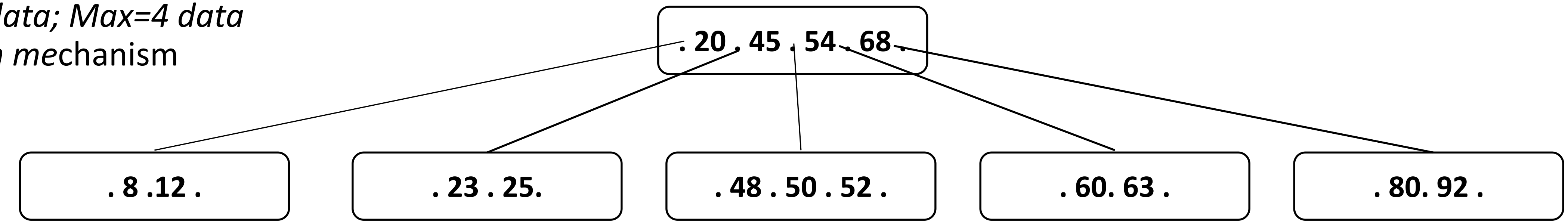
Furthermore, if the leaf node that contained the successor has fewer than (n div 2) data, various actions will be taken.

B-Trees

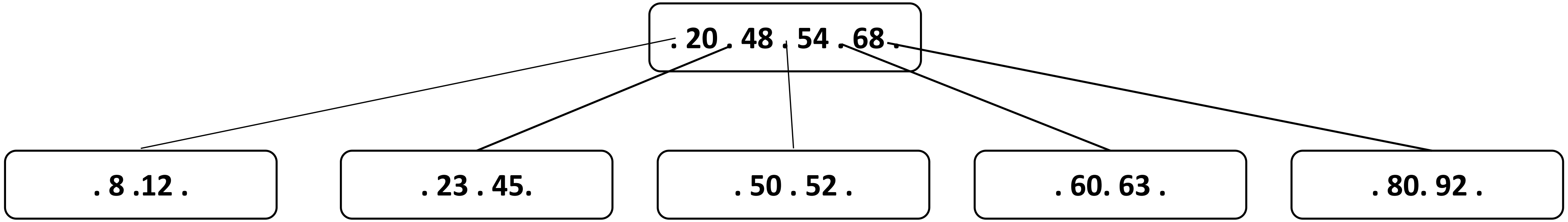
Examine the brothers on the right and on the left.

If one of the brothers (left or right) contains more than n div 2 data , <u>Borrow</u>

B-Tree(order=5)
Min=2 data; Max=4 data
*Deletion me*chanism



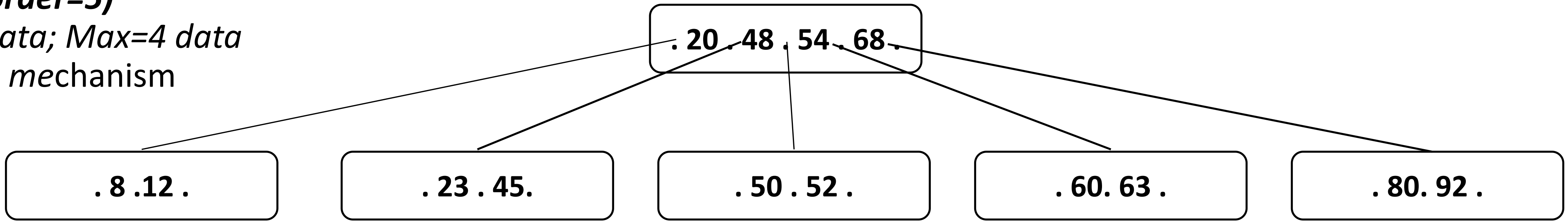
25



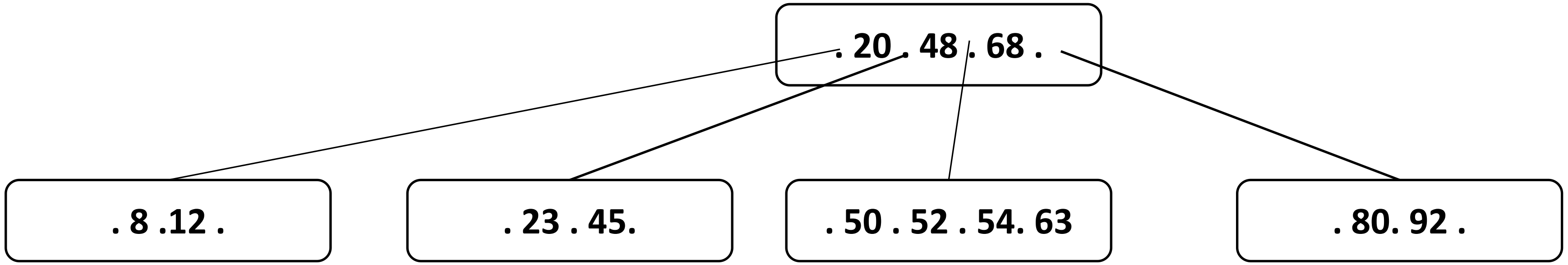
If both brothers contain exactly n div 2 data , <u>Fusion</u>

B-Trees

B-Tree(order=5)
Min=2 data; Max=4 data
*Deletion me*chanism



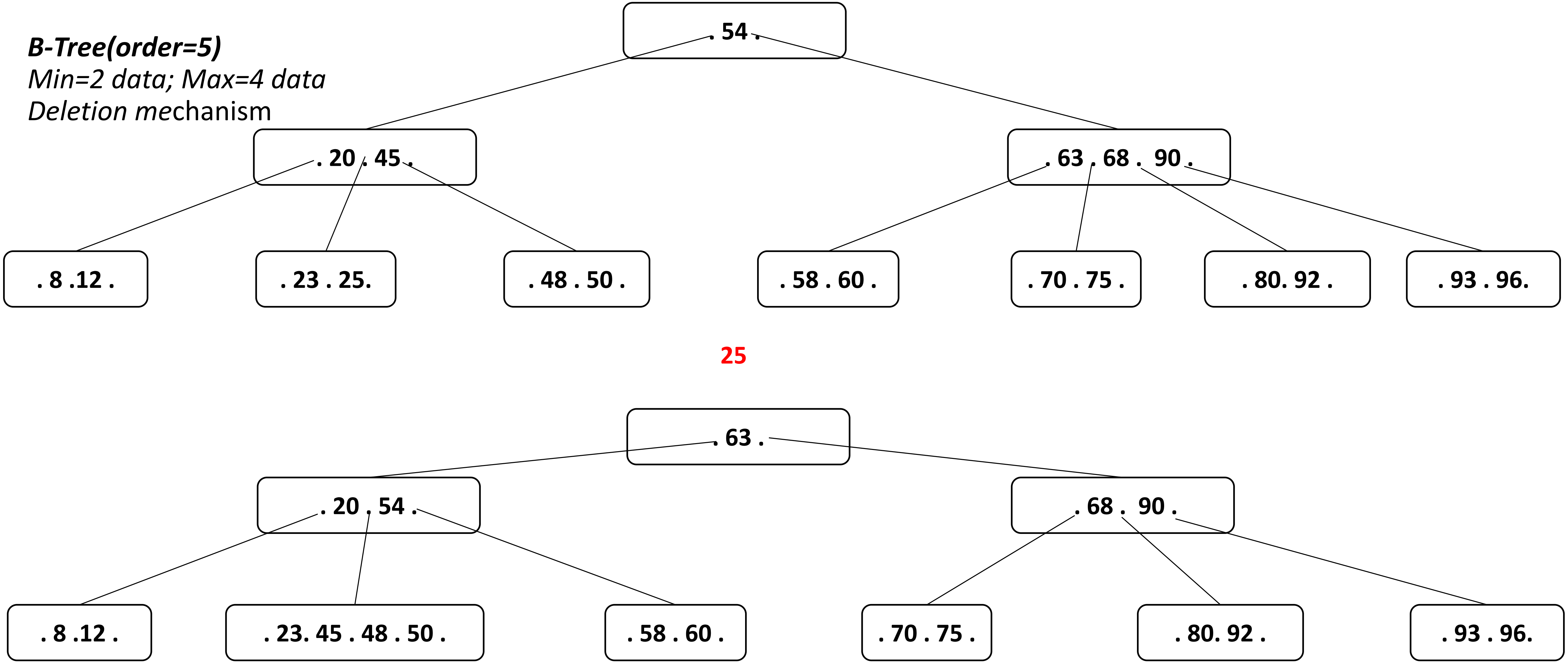
60



B-Trees

Case where the father contains only n div 2 data, and therefore he has no data to give. --> He can borrow from his father and brother.

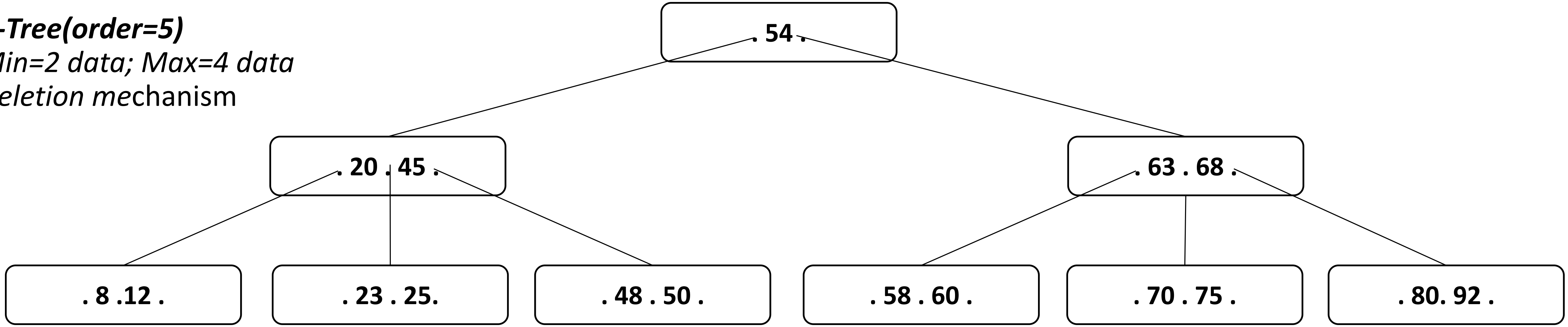
B-Tree(order=5)
Min=2 data; Max=4 data
*Deletion me*chanism



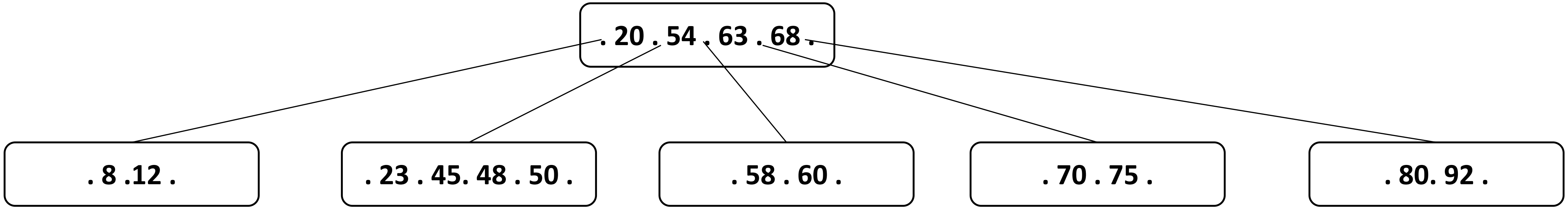
B-Trees

The parent's brothers have no data to provide; the parent and his brother can also be concatenated, and a data is taken from the grandparent.

B-Tree(order=5)
Min=2 data; Max=4 data
*Deletion me*chanism



23



B-Trees

Use in RAM : 2-3 trees (Definition / Properties)

It is a balanced m-ary search tree (B-tree) of order 3.

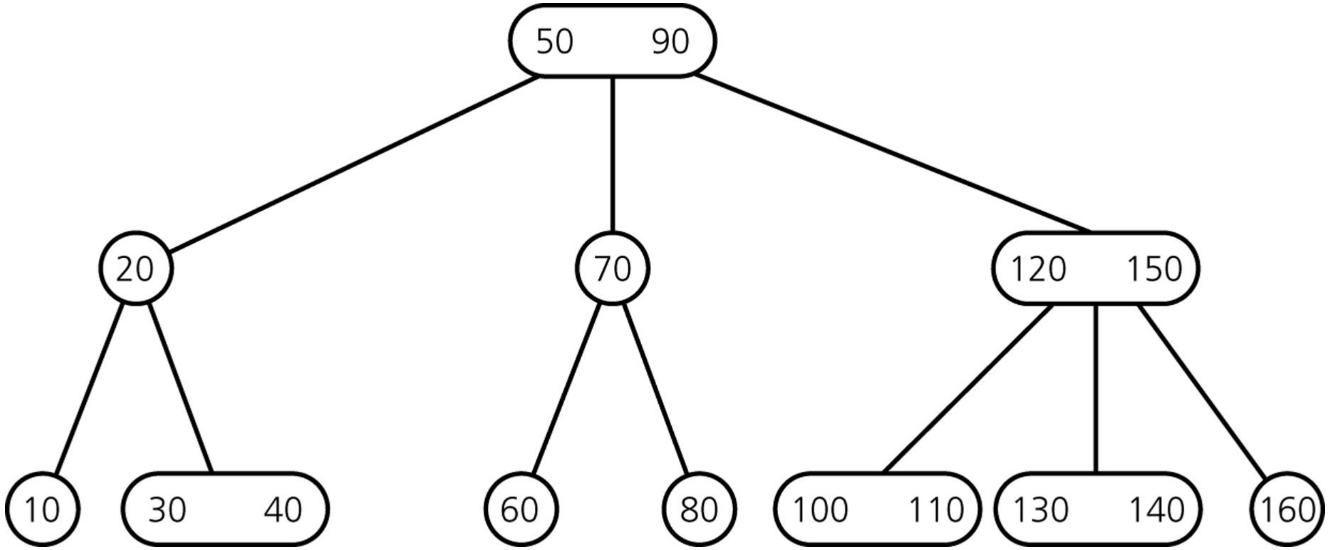
Balance guaranteed by construction.

The number of elements in a 2-3 tree of height h is between $2^h - 1$ and $3^h - 1$.

Therefore, the height of a 2-3 tree with n elements is between $INT(\log_3 (N+1))$ and $INT(\log_2 (N+1))$

B-Trees

Example of a 2-3 tree



B-Trees

Use in RAM : 2-4 trees (Definition / Properties)

It is a balanced m-ary search tree (B-tree) of order 4.

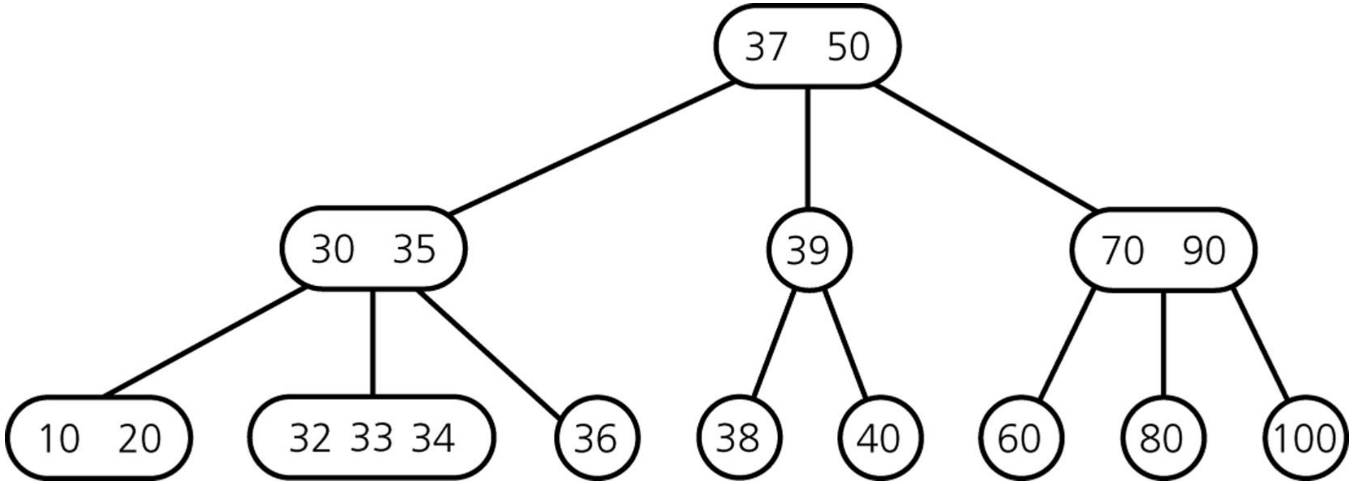
Balance guaranteed by construction.

The number of elements in a 2-4 tree of height h is between $2^h - 1$ and $4^h - 1$.

Therefore, the height of a 2-4 tree with n elements is between INT($\log_4$ (N+1)) and INT($\log_2$ (N+1))

B-Trees

Example of a 2-4 tree



M-ary SearchTrees

```
LET
      A : MST ( 4 ) ;
BEGIN
END
```

C Dynamic Implementation

```
 #include <stdio.h>
  #include <stdlib.h>

  /** -Implementation- **\: M-ARY
SEARCH TREE OF INTEGERS**/

  /** M-ary search trees **/
  typedef int Typeelem_M4i   ;
  typedef struct Node_M4i * Pointer_M4i ;

  struct Node_M4i
    {
      int Degree ;
      Pointer_M4i Child[4] ;
      Typeelem_M4i Infor[4] ;
      Pointer_M4i Parent ;
    } ;

  Typeelem_M4i
Node_value_mst_M4i(Pointer_M4i P, int I)
    { return P->Infor[I-1] ;  }
```

```
  Pointer_M4i Child_M4i( Pointer_M4i P,
int I)
    { return P->Child[I-1] ; }
  Pointer_M4i Parent_M4i( Pointer_M4i P)
    { return P->Parent ; }
  void Ass_node_val_mst_M4i
( Pointer_M4i P, int I, Typeelem_M4i Val)
    {   P->Infor[I-1] = Val ;    }

  void Ass_child_M4i( Pointer_M4i P, int I,
Pointer_M4i Q)
    { P->Child[I-1] =  Q ; }

  void Ass_parent_M4i( Pointer_M4i P,
Pointer_M4i Q)
    { P->Parent =  Q ; }
  int Degree_M4i ( Pointer_M4i P )
    { return P->Degree ; }
  void Ass_degree_M4i ( Pointer_M4i P, int
N)
    { P->Degree = N ; }
```

```
  void Allocate_node_M4i(  Pointer_M4i *P )
    {
      int I ;
      *P = (struct Node_M4i *)
malloc( sizeof( struct Node_M4i))   ;
      for (I=0; I< 4; ++I) (*P)->Child[I] = NULL;
      (*P)->Degree = 0 ;
    }

  void Free_node_M4i(Pointer_M4i P)
    { free  ( P );}

  /** Variables of main program **/
  Pointer_M4i M=NULL;
  int main(int argc, char *argv[])
    {
      system("PAUSE");
      return 0;
    }
```

M-ary SearchTrees

Static Implementation

- Multiple M-ary search trees can be placed in the same array.
- The array consists of quadruples: (Info, Children, Degree, Occupied).
- Info is an array of (Order-1) values.&
- Children is an array of (Order) indices.
- The Degree field contains the current number of values in the node.
- The Occupied field is necessary for the CreateNode and FreeNode operations.

An initialization phase is required before using this array.
Therefore, the array is global.

An M-ary search tree is defined by the index of its first element.

M-ary SearchTrees

C Static Implementation

```
#define Max 100
#define Order 8
#define True 1
#define False 0
#define Nil -1
typedef int Bool;
typedef int Anytype;
struct TypeMst
{
  int Child[Order];
  int Info[Order-1];
  int Parent;
  short Degree;
  Bool Occupied;
};

struct TypeMst Mst[Max];
```

```
void Init()
{
  int I;
  for (I=0; I<Max; I++)
    Mst[I].Occupied = False;
}

void Allocate_node ( int *P )
{
  Bool Found;
  *P = 0;
  Found = False;
  while ( *P < Max && !Found )
  if ( Mst[*P].Occupied )
  *P++ ;
  else
  Found = True;
  if ( !Found ) *P = -1;
}
```

M-ary SearchTrees

C Static Implementation

```
void Free_node ( int P )
{   Mst[P].Occupied = False ;}

Anytype Node_value_mst ( int P, int I )
{    return( Mst[P].Info[I-1] );}

int Child ( int P, int I )
{    return ( Mst[P].Child[I-1] ) ;}

int Parent ( int P )
{    return ( Mst[P].Parent ) ; }

void Ass_parent(int P, int I)
{    Mst[P].Parent = I; }

void Ass_node_val_mst ( int P, int I,
Anytype Val)
{   Mst[P].Info[I-1] = Val; }
```

```
void Ass_Child ( int P, int I, int J)
{
  Mst[P].Child[I-1] = J;
}

short Degree ( int P )
{
  return ( Mst[P].Degree );
}

void Ass_degree ( int P, short I )
{
  Mst[P].Degree = I ;
}

int main(int argc, char *argv[])
{
  system("PAUSE");
  return 0;
}
```