PRINCIPLES OF DATA ORGANISATION

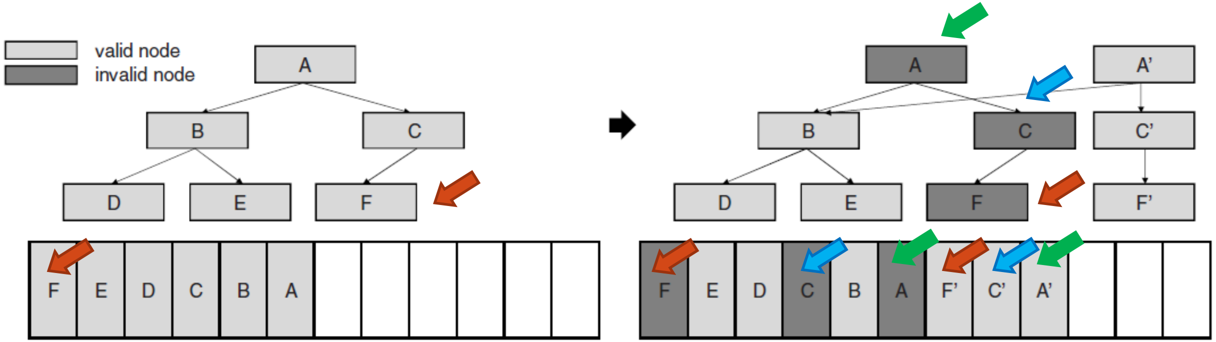
μ-Tree

MOTIVATION

- Raw flash, no in-place update, no FTL
  - Flash translation layer
- B-Trees are good – we want to use them

B-TREE ISSUE ~ WANDERING TREE

- B-tree efficiently makes use of block-oriented storage by keeping related records together on the same page.
  - Inner nodes of a B+-tree store only keys and pointers
- When a record update happens, the leaf node needs to be modified
- Since in-place update is not supported by the SSDs, the whole page needs to be moved into a new location
- Page move requires modification of the pointer in the parent node → iterative process ending only in the root

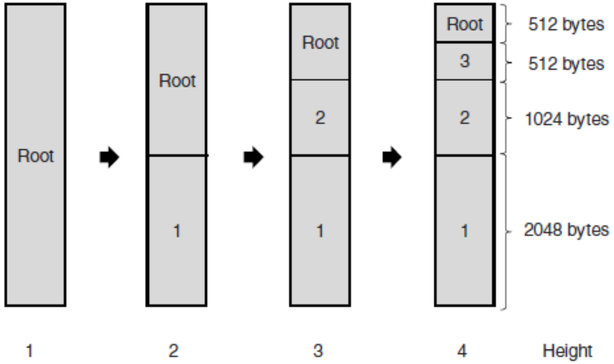


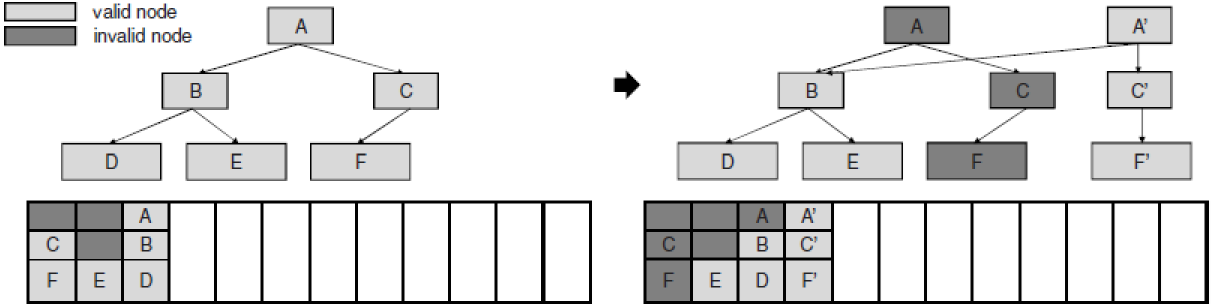
μ-TREE

- Kang et. al. 2007
- In μ-Tree all the nodes along the path from the root to the leaf are put together into a single flash memory page
- μ-Tree outperforms B+-tree by up to 28% and by up to 90% with a 8KB in-memory cache.
- Unlike B-tree, μ-Tree's nodes are of variable size
- No significant difference between B+-Tree and μ-Tree except that the size of a node in a **µ**-Tree is determined by its level and the height of the tree

μ-TREE

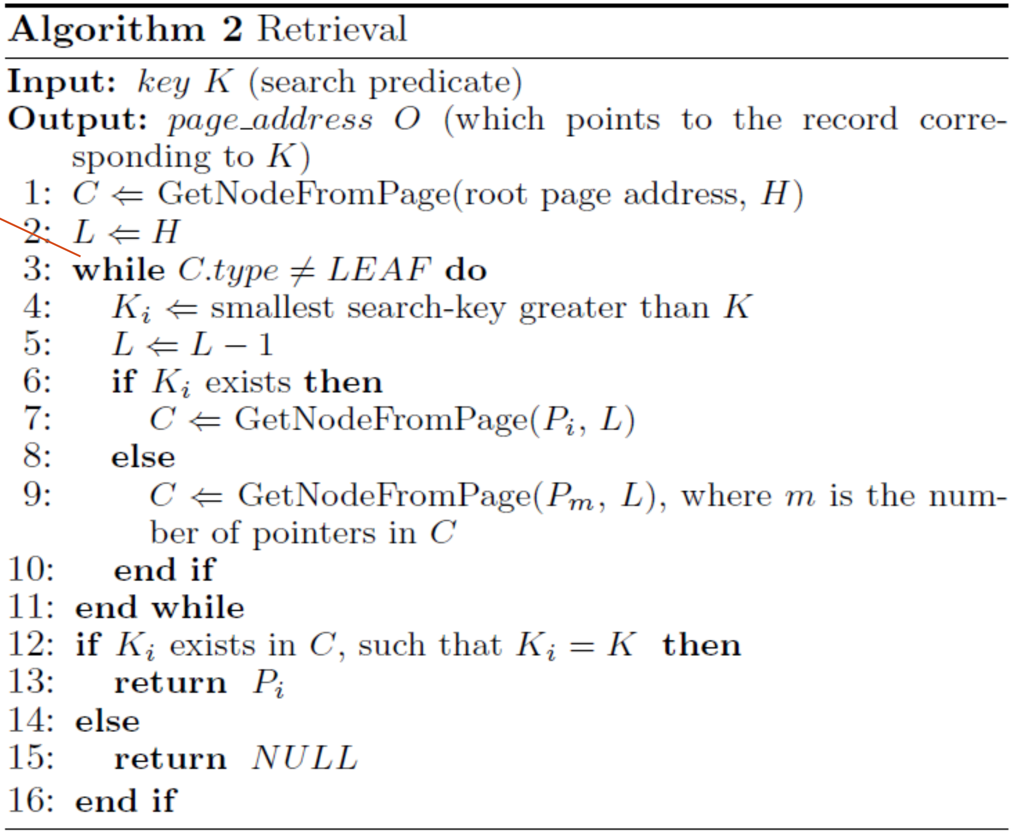
- Leaf node always occupies half of the page.
- As the level is increased, the node size is reduced by half
- Only the root node has the same size as its children nodes





ALGORITHM

Leaves reside in level 1



Algorithm 2 Retrieval

Input: *key* K (search predicate)

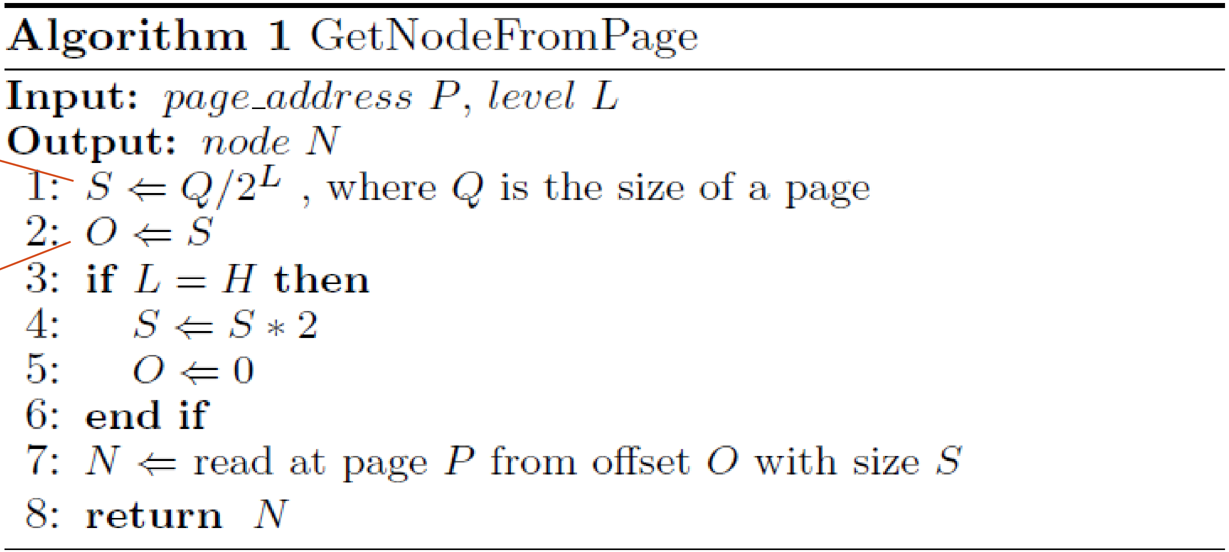
Output: *page_address* O (which points to the record corresponding to K)

```
$C \Leftarrow$ GetNodeFromPage(root page address, $H$)
$L \Leftarrow H$
while $C.type \neq LEAF$ do
    $K_i \Leftarrow$ smallest search-key greater than $K$
    $L \Leftarrow L - 1$
    if $K_i$ exists then
        $C \Leftarrow$ GetNodeFromPage($P_i$, $L$)
    else
        $C \Leftarrow$ GetNodeFromPage($P_m$, $L$), where $m$ is the number of pointers in $C$
    end if
end while
if $K_i$ exists in $C$, such that $K_i = K$ then
    return $P_i$
else
    return $NULL$
end if
```

ALGORITHM

Size of the block/node to read

Offset of the block/node to read



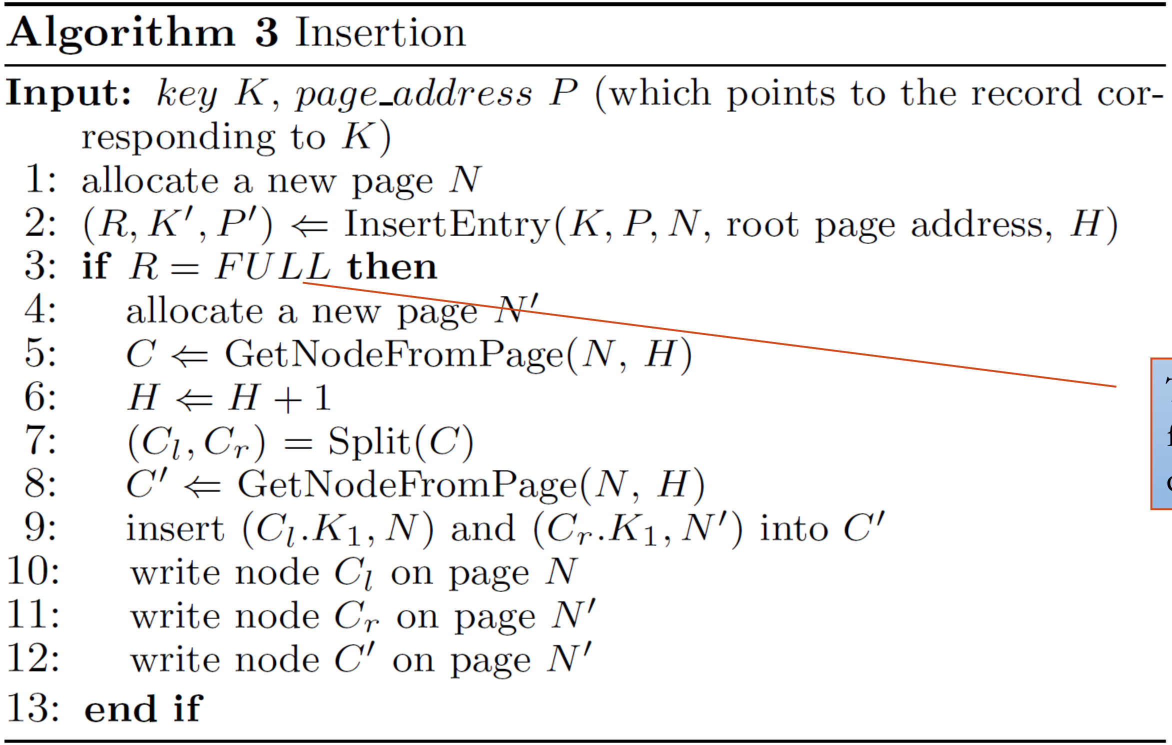
Algorithm 1 GetNodeFromPage

Input: *page_address* P, *level* L

Output: *node* N

```
$S \Leftarrow Q/2^L$ , where $Q$ is the size of a page
$O \Leftarrow S$
if $L = H$ then
    $S \Leftarrow S * 2$
    $O \Leftarrow 0$
end if
$N \Leftarrow$ read at page $P$ from offset $O$ with size $S$
return $N$
```

ALGORITHM



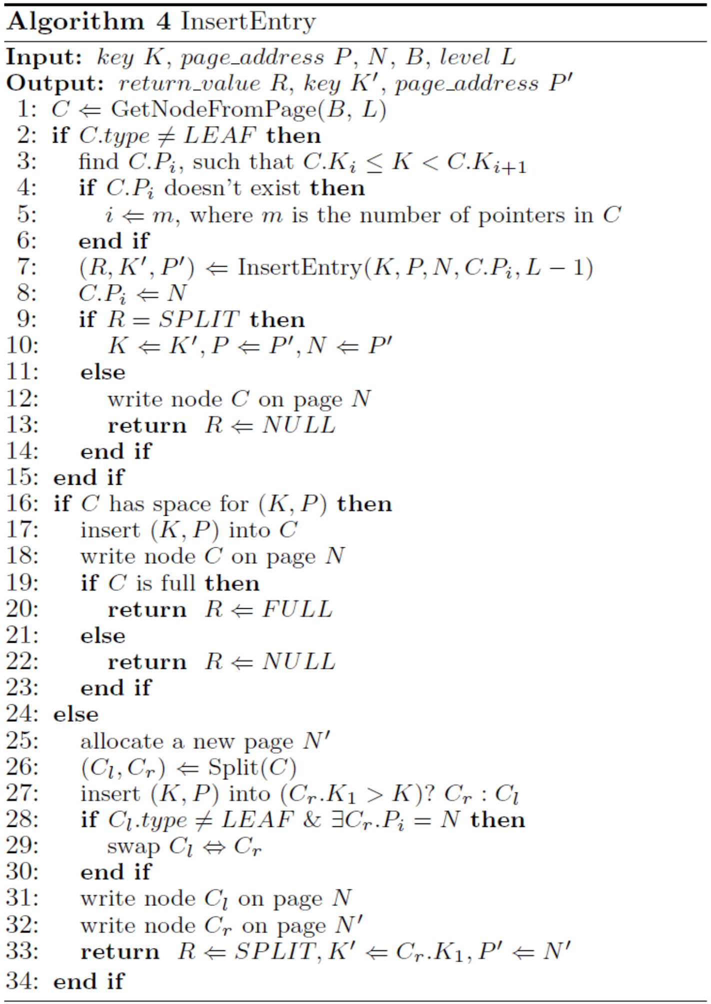
Algorithm 3 Insertion

Input: *key* K, *page_address* P (which points to the record corresponding to K)

```
allocate a new page $N$
$(R, K', P') \Leftarrow$ InsertEntry($K, P, N$, root page address, $H$)
if $R = FULL$ then
    allocate a new page $N'$
    $C \Leftarrow$ GetNodeFromPage($N$, $H$)
    $H \Leftarrow H + 1$
    $(C_l, C_r) =$ Split($C$)
    $C' \Leftarrow$ GetNodeFromPage($N$, $H$)
    insert $(C_l.K_1, N)$ and $(C_r.K_1, N')$ into $C'$
    write node $C_l$ on page $N$
    write node $C_r$ on page $N'$
    write node $C'$ on page $N'$
end if
```

The root node becomes full as a result of the current insertion

ALGORITHM



Algorithm 4 InsertEntry

```
Input: key $K$, page_address $P$, $N$, $B$, level $L$
Output: return_value $R$, key $K'$, page_address $P'$
$C \Leftarrow$ GetNodeFromPage($B$, $L$)
if $C.type \neq LEAF$ then
    find $C.P_i$, such that $C.K_i \leq K < C.K_{i+1}$
    if $C.P_i$ doesn't exist then
        $i \Leftarrow m$, where $m$ is the number of pointers in $C$
    end if
    $(R, K', P') \Leftarrow$ InsertEntry$(K, P, N, C.P_i, L - 1)$
    $C.P_i \Leftarrow N$
    if $R = SPLIT$ then
        $K \Leftarrow K', P \Leftarrow P', N \Leftarrow P'$
    else
        write node $C$ on page $N$
        return $R \Leftarrow NULL$
    end if
end if
if $C$ has space for $(K, P)$ then
    insert $(K, P)$ into $C$
    write node $C$ on page $N$
    if $C$ is full then
        return $R \Leftarrow FULL$
    else
        return $R \Leftarrow NULL$
    end if
else
    allocate a new page $N'$
    $(C_l, C_r) \Leftarrow$ Split$(C)$
    insert $(K, P)$ into $(C_r.K_1 > K)$? $C_r : C_l$
    if $C_l.type \neq LEAF$ & $\exists C_r.P_i = N$ then
        swap $C_l \Leftrightarrow C_r$
    end if
    write node $C_l$ on page $N$
    write node $C_r$ on page $N'$
    return $R \Leftarrow SPLIT, K' \Leftarrow C_r.K_1, P' \Leftarrow N'$
end if
```