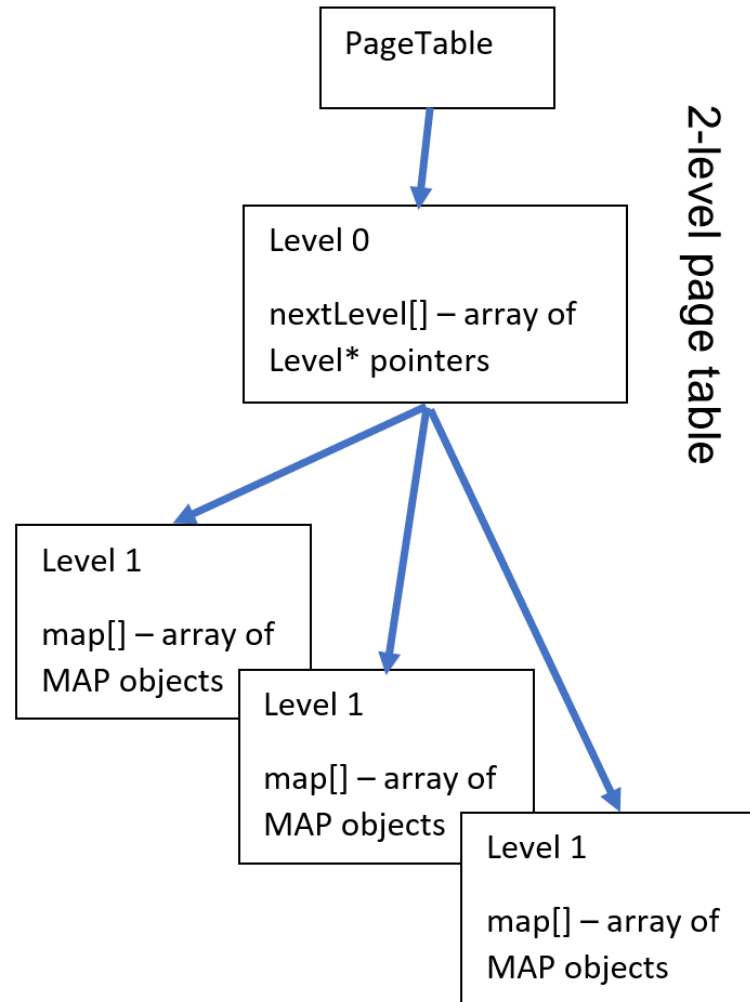


# A sample data structure for N-level page tables

# Sample Data Structure

- PageTable – Contains information about the tree
  - Level – A structure describing a specific level of the page table.
    - nextLevel[ ] – Array of Level\* pointers to the next level. (non-leaf or interior level)
    - map[ ] – Array of Map objects, each mapping a logical/virtual page to a physical frame. (leaf level)
- Map object can have a frame number and valid flag



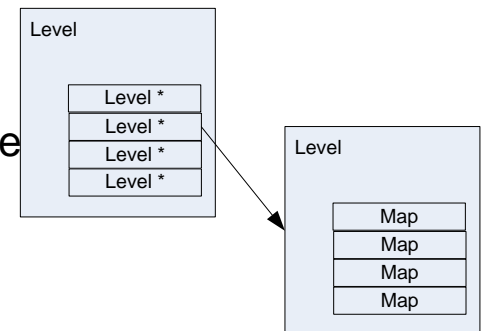
# PageTable

- Contains information about the tree:
  - LevelCount: Number of levels
  - Bitmask [i]: bit mask for level i
  - BitShift [i]: # of bits to shift level i page bits
  - EntryCount [i]: # of possible pages for level i

# Levels of the page table

- Each level of the page table is represented by a pair of structures:
  - Conceptually, Level contains an array of pointers to the next level (Level \*) or Map entries
    - C/C++ does not permit variable size structures.
    - We circumnavigate this by using a pointer to a runtime allocated structure.
    - See the course FAQ for allocating arrays at runtime.
  - Interior levels use Level and nextLevel [ ] (NextLevelPtr, Level \*[] or Level \*\*), each element is a Level \* pointer pointing to a next Level object
  - Leaf levels use Level and map[ ] (MapPtr, Map[] or Map\*), each element is a Map object
- Useful information to have in Level
  - Current depth
  - Pointer to the PageTable structure/object to access information

Conceptual organization



# Initialization

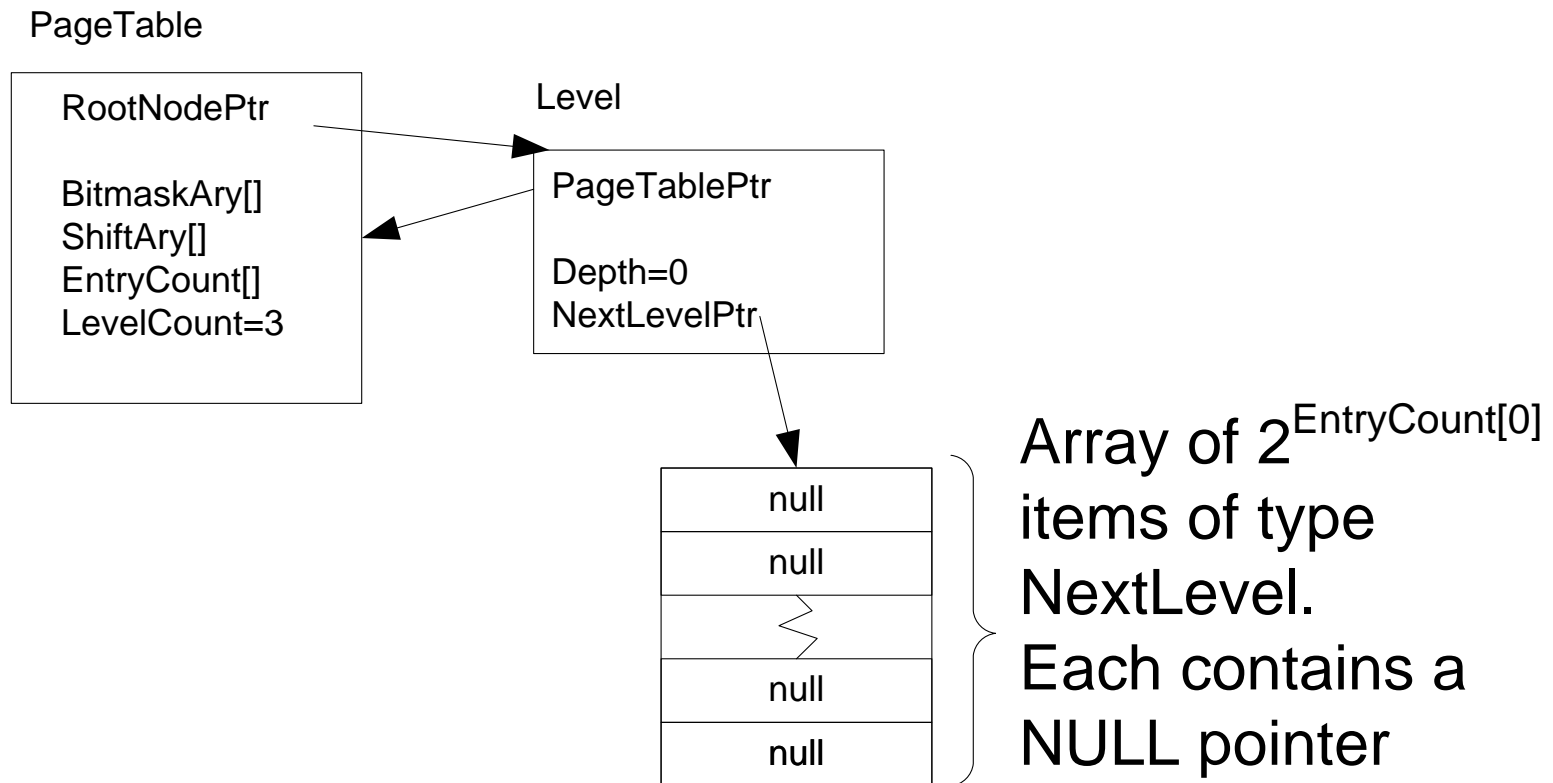
- Suppose we wanted to create a 3 level page table with 8 bits per level on a 32 bit address space.
  - We would allocate a PageTable structure and populate it with the following values:
    - LevelCount = 3
    - BitmaskAry [] = {0xFF000000, 0x00FF0000, 0x0000FF00}
    - ShiftAry [] = {24, 16, 8}
    - EntryCount [] = {2<sup>8</sup>, 2<sup>8</sup>, 2<sup>8</sup>}
- BitmaskAry, ShiftAry, and EntryCount should all be computed dynamically based on your PageTable specification.

# Initialize Data Structure

- In addition, we would allocate the level 0 information:
  - Allocate a Level structure
    - Set its depth to 0
    - Have it point back to the PageTable
    - Allocate an array of 256 ( $2^8$ ) pointers to Level structures.
      - Initialize all to NULL (number of level 1 entries)
      - If this had been a 1 level page table we would have allocated Map structures instead of pointers to Levels

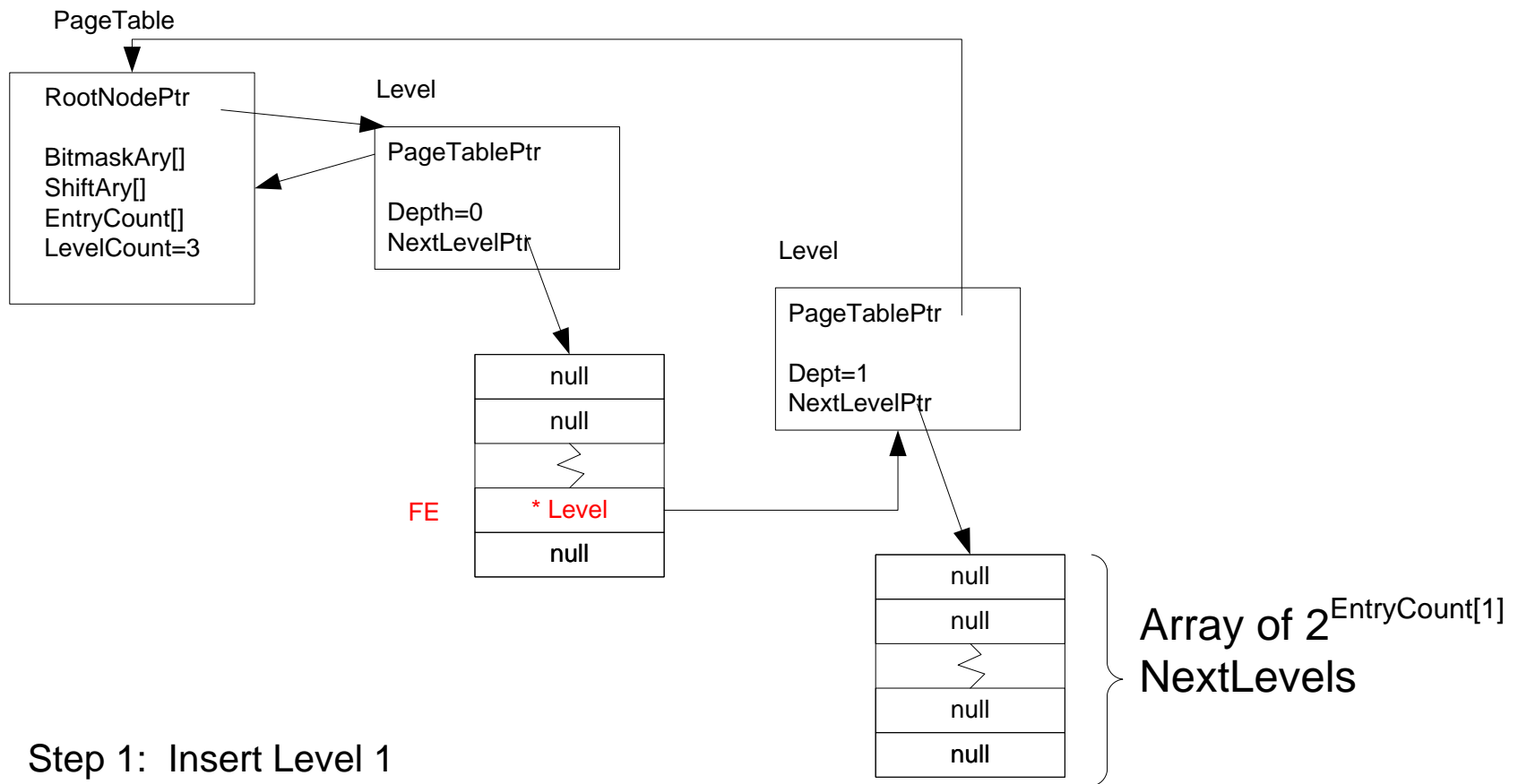
# 3 level example

- Empty table



# Page Insertion

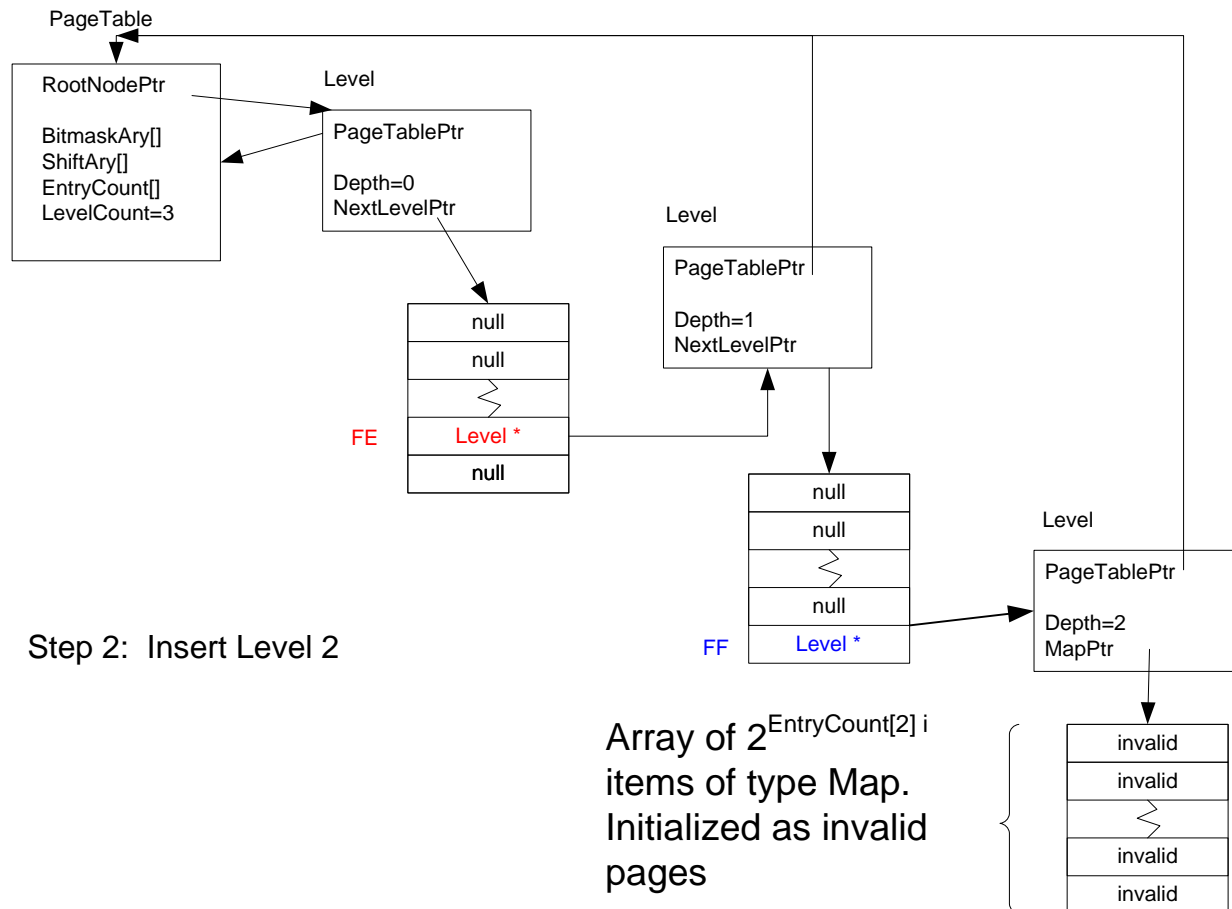
Assume 32 bit word, 8 bit pages for each level  
Insert address 0x**FE****FF****FE**C2 mapping to frame 3





# Page Insertion

Assume 32 bit word, 8 bit pages for each level  
 Insert address 0x**F**E**F**F**F**E**C**2 mapping to frame 3

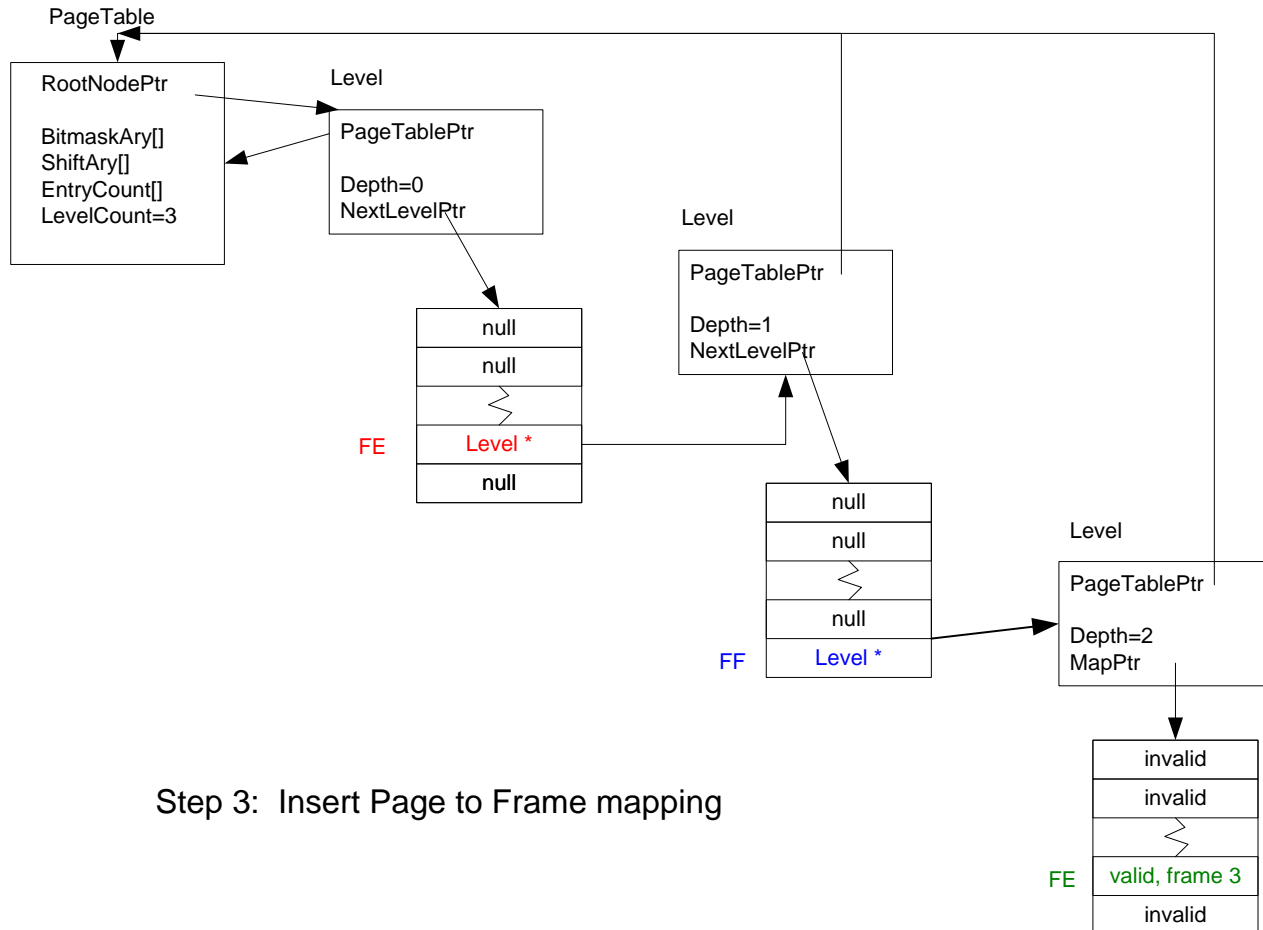


# Inserting leaf nodes

- Next, we insert the level 2 node which is a leaf in a 3 level page table.
- This time, we allocate Maps instead of pointers to next Level.
- Initialize the pages (maps) to invalid.
- Set the level 2 pages (maps) to valid and store the frame.

# Page Insertion

Assume 32 bit word, 8 bit pages for each level  
Insert address 0x**FE****FF****FE****C2** mapping to frame 3

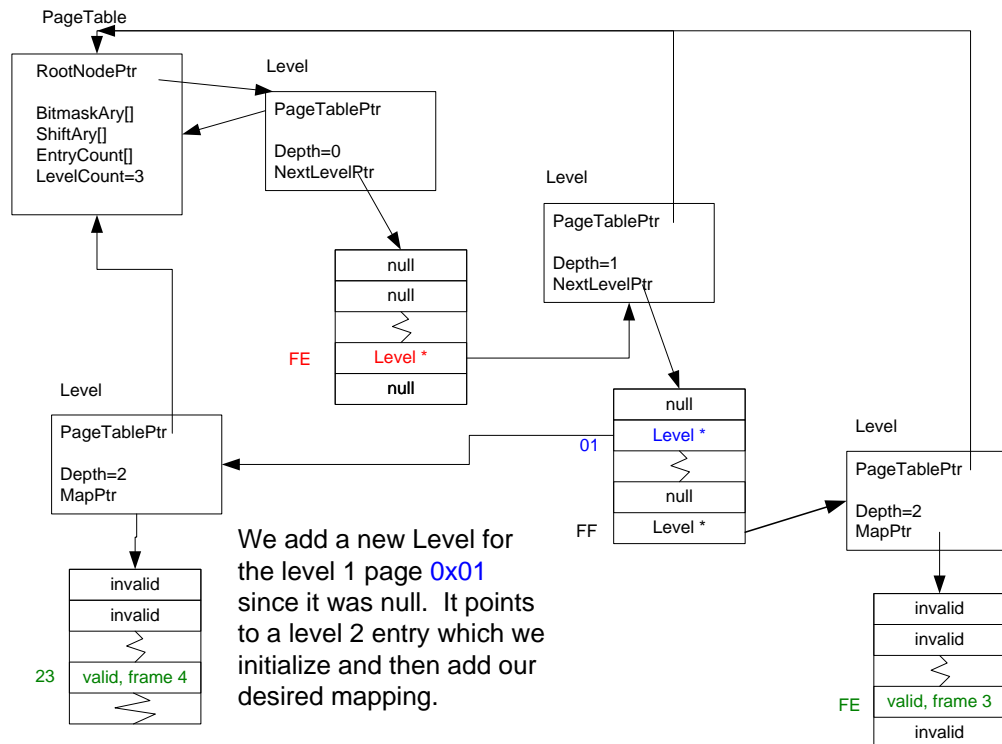


# Another example

- Next, add a mapping between the page associated with address 0xFE0123C2 and frame 4.
- Pay attention to the fact that the level 0 page, 0xFE, already exists and note how the new entries are added.

# Adding a second page

Assume 32 bit word, 8 bit pages for each level  
Insert address 0xFE0123C2 mapping to frame 4



# Page Insertion Pseudo-Code

## Page Table

```
pageInsert(pageTablePtr, address, frame) {  
    // C users, you would have to rename the 2nd pageInsert  
    // function since C cannot distinguish two functions with  
    // the same name and different signatures.  
    pageInsert(pageTablePtr->rootNodePtr, address, frame)  
}
```

## Level

```
pageInsert(levelPtr, address, frame) {  
    Find index into current page level  
    if leaf node(levelPtr) {  
        Set appropriate page index to valid and store Frame  
    } else {  
        Create a new Level and set level to current depth + 1  
        Create an array of Level * entries based upon the number of entries in the  
        new level and initialize to null/invalid as appropriate  
        pageInsert(pointer to new Level, address, frame)  
    }  
}
```