Patricia

Patricia

- Practical Algorithm To Retrieve Information Coded In Alphanumeric.
- Compressed binary trie.
- All nodes are of the same data type (binary tries differentiate branch and element nodes).
  - Pointers to only one kind of node.
  - Simpler storage management.

Patricia (cont'd)

- Uses a header node.
- Remaining nodes define a trie structure that is the left subtree of the header node.
- Trie structure is the same as that for the compressed binary trie of previous lecture.

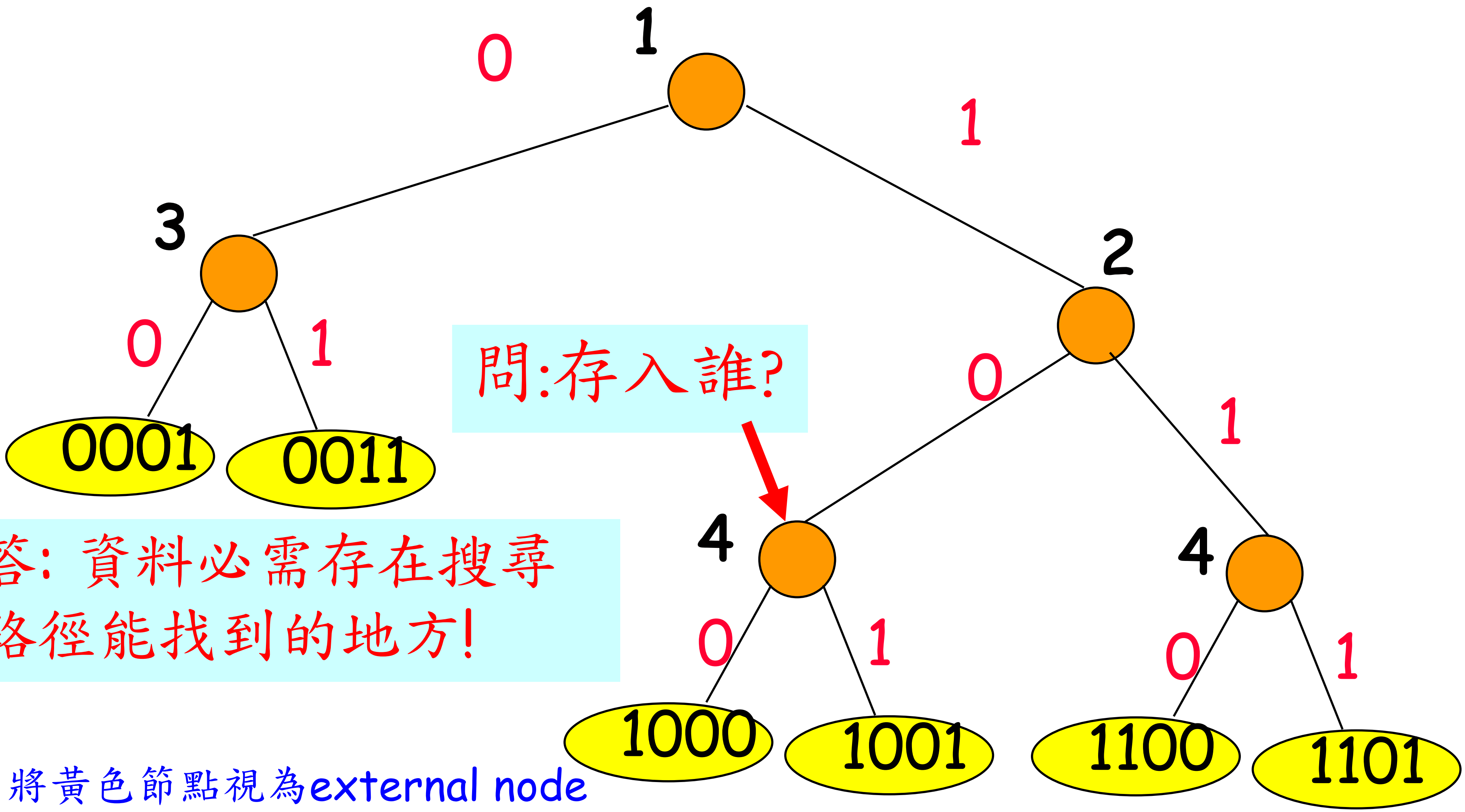
Node Structure

bit#	LC	Pair	RC

- bit# = bit used for branching // key
- LC = left child pointer
- Pair = dictionary pair // value
- RC = right child pointer

Compressed Binary Trie to Patricia

Move each element into an "ancestor" or "header" node.

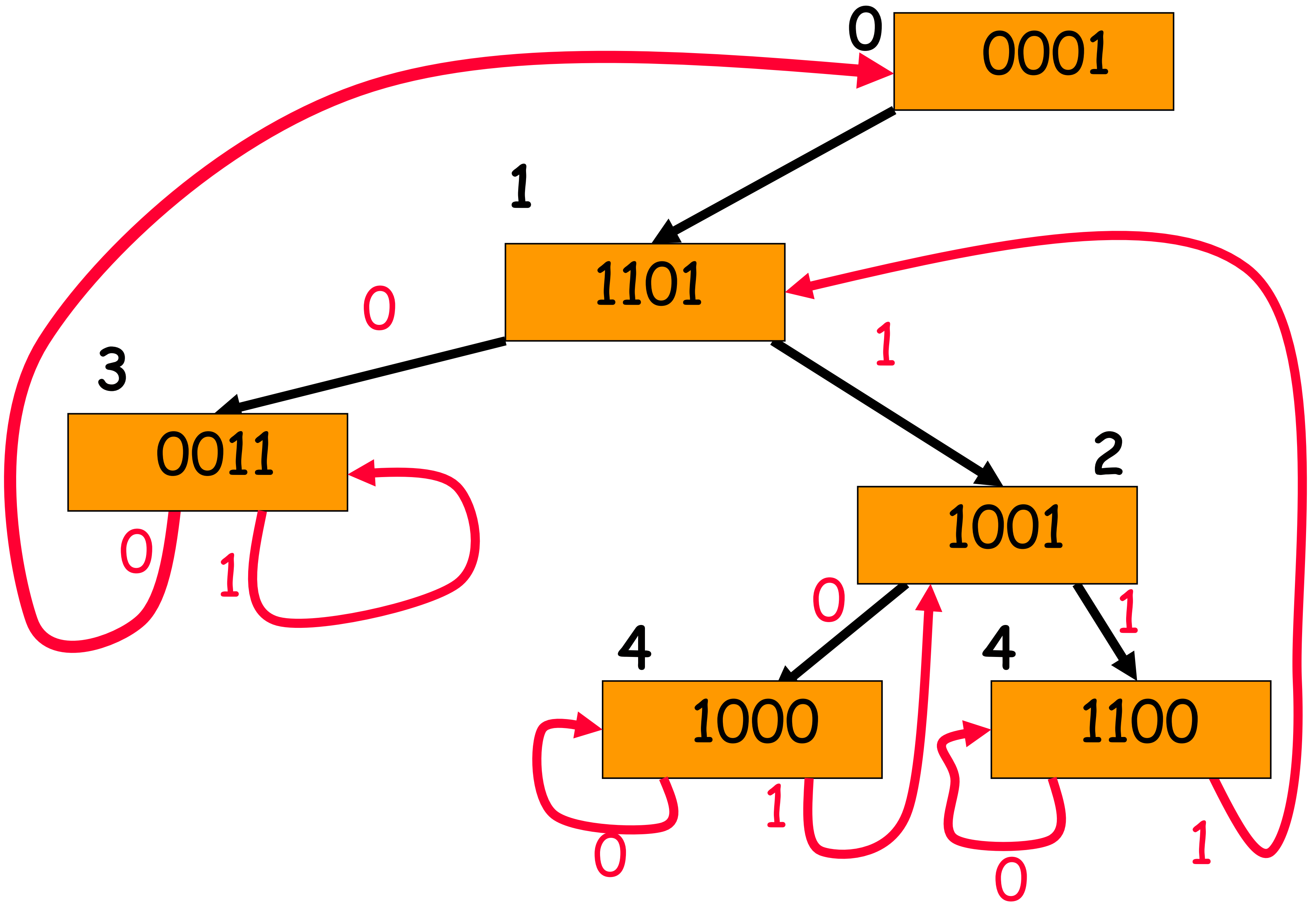


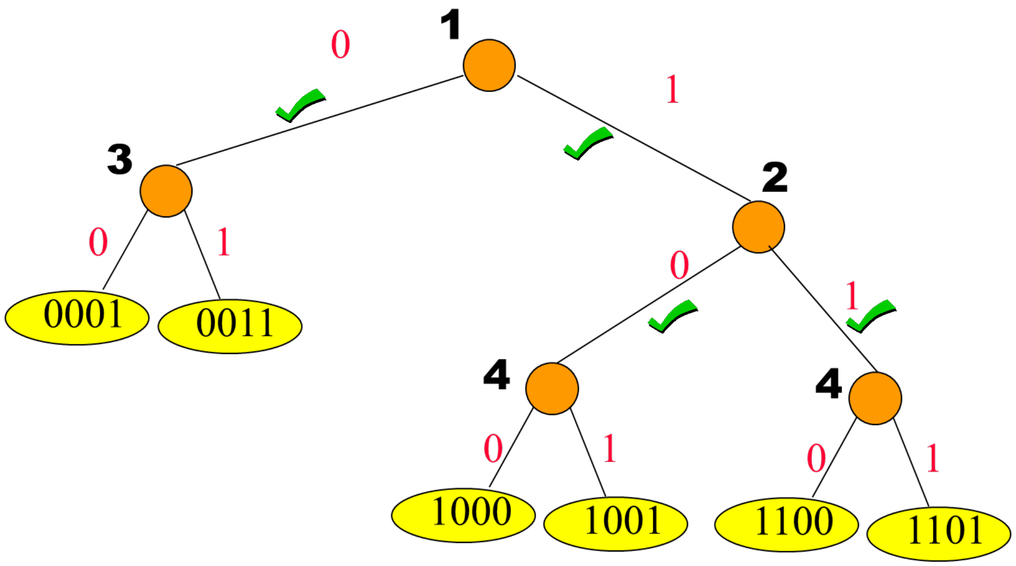
問:存入誰?

答: 資料必需存在搜尋路徑能找到的地方!

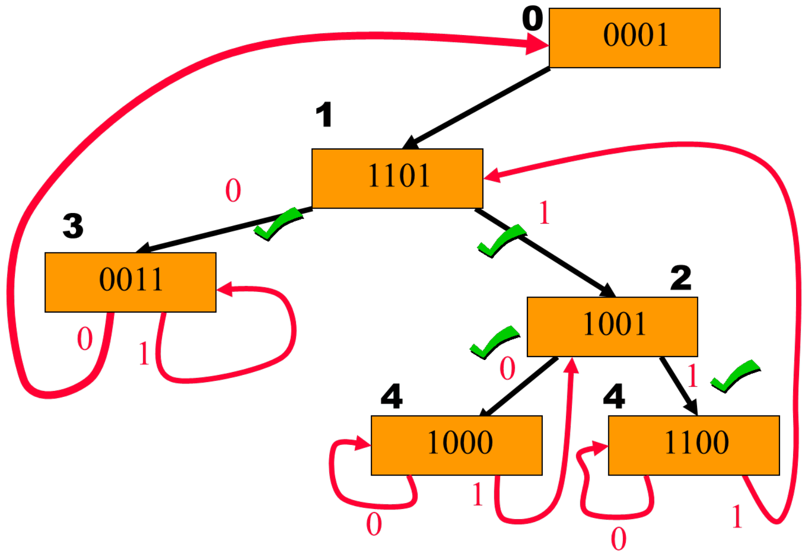
// 將黃色節點視為external node
// #(黃色)=#(橘色)+1

Compressed Binary Trie To Patricia





Compressed trie



Patricia

- 請檢視一下黑邊 ✓ (in patricia)
- 請檢視一下紅邊 (in patricia)

Pointers in Patricia

- 所有external nodes (data items) 的個數為branch nodes的個數 “加一”，因此需要一個額外的header來儲存某一個leaf的data item
- Black pointer: 原來在compressed tries裡的邊 (**搜尋用的**)，除了root出來那個黑邊
- Red pointer: 儲存在external node裡的data將被安排在其 “至” root的路徑上當中的 “某一” 節點儲存 (say, p)，red pointer指標則指向p
  - Red pointer為搜尋過程中的最後一哩路 (last mile)
- 一資料會安排在哪個branch node需視資料插入Patricia的 “次序” 而定 (即使同一組資料，其插入順序不同，則所得的Patricia也可能會不同)

Search(key)

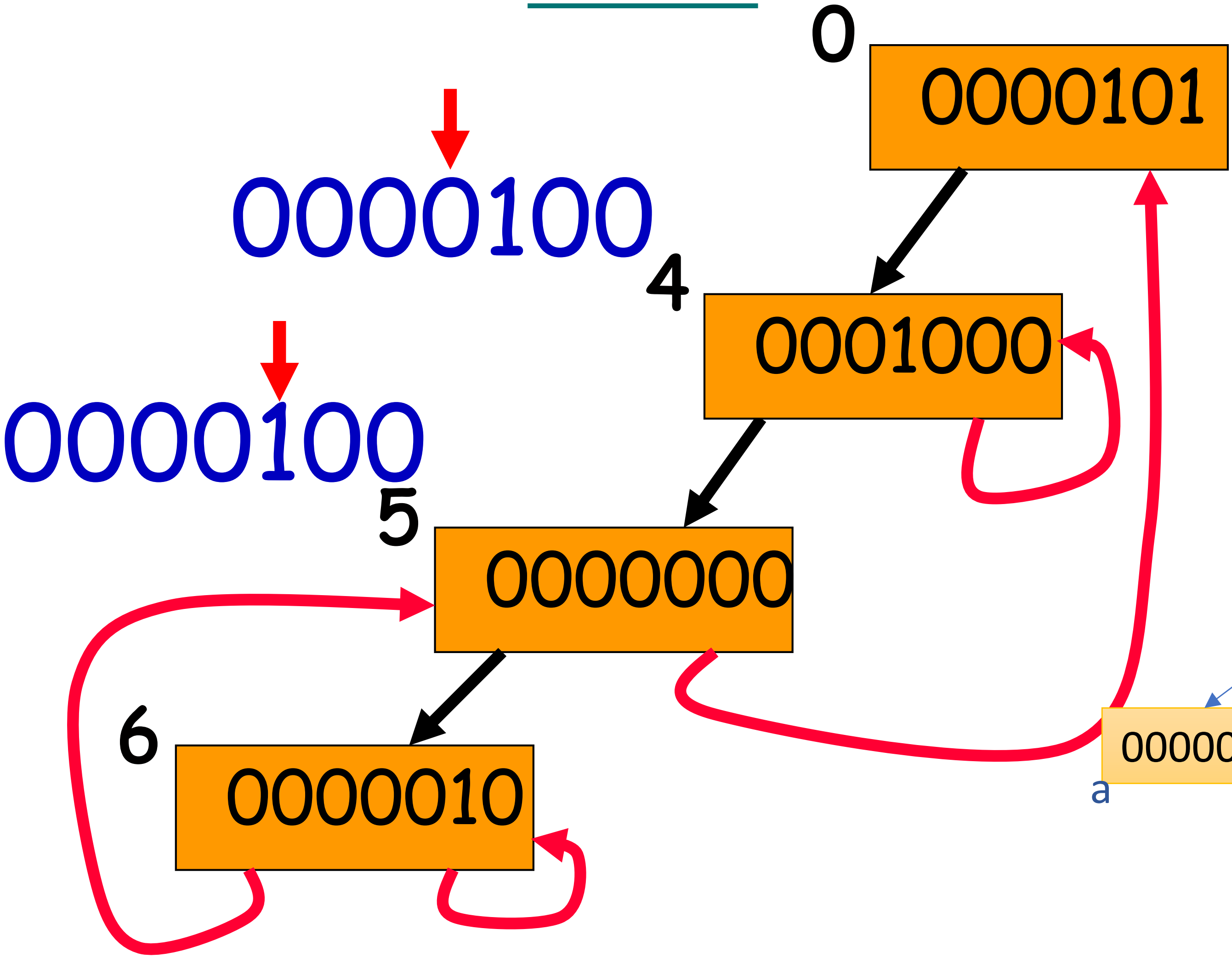
- 只看key的第bit#，直到走到了red pointer所指向的branch node (亦即抵達了對應compressed trie的leaf data node)

- 往 “遞增bit#” 的方向搜尋 (可能數次的black pointers)，最終使用了一次red pointer
  - 最後那一個red pointer到達的節點其bit#將小於等於倒數第二個路徑節點的 (因該red pointer指向了往root方向路徑上的branch node) // 判斷是否走到了red pointer
  - Red pointer是方便我們講解用，其實不需要定義邊的顏色

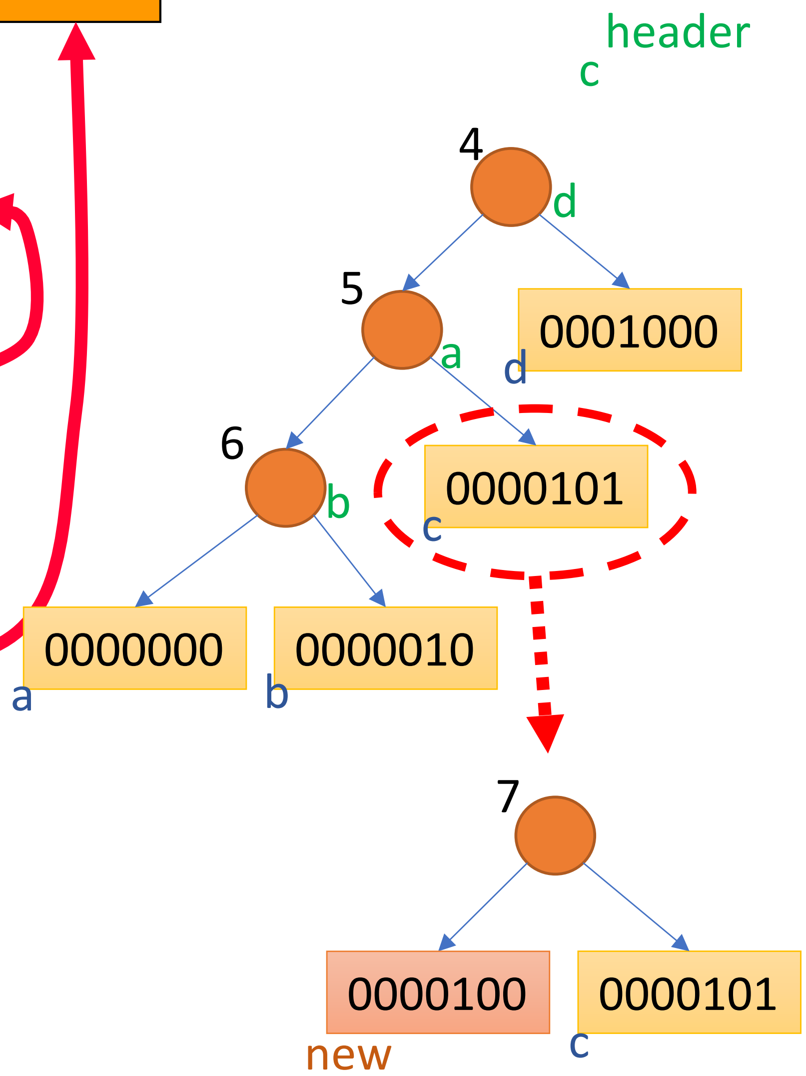
- Root沒有選擇，必定往其left child (bit# = 0)搜尋

How to Insert 0000100?

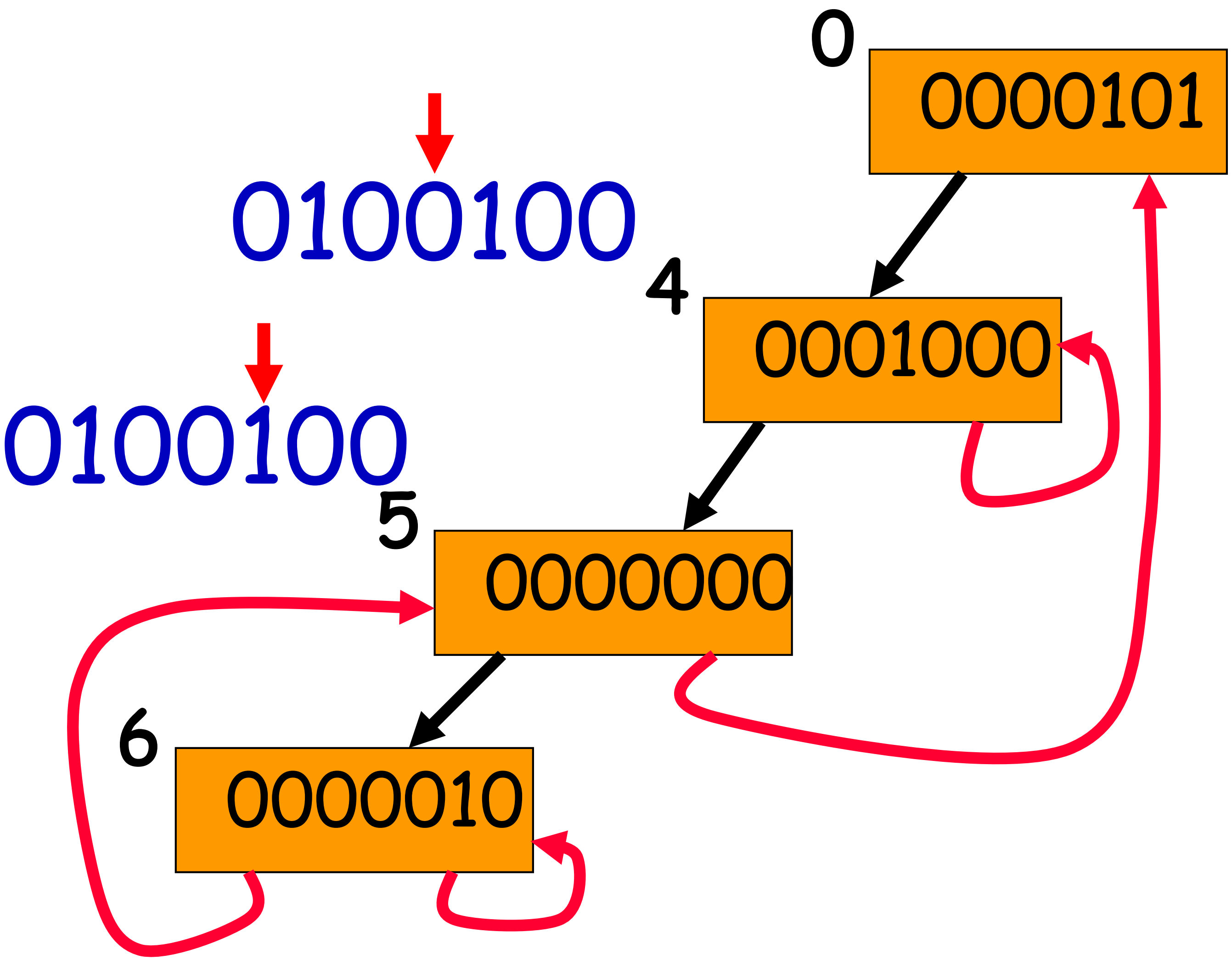
Patricia

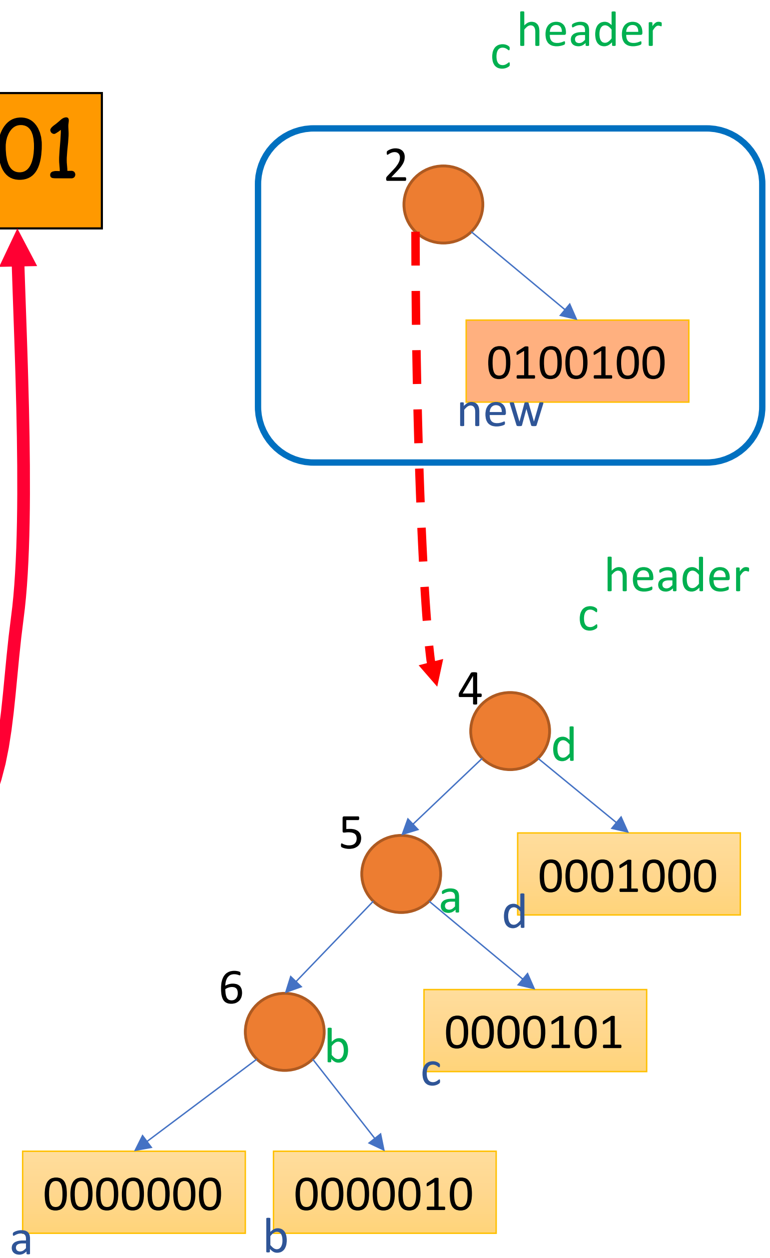


Compressed tries



How to Insert 0100100?





Insert的概念

(請轉換成Compressed Tries來理解這些步驟)

給定要插入的key x

Step 1: 從root開始在Patricia裡搜尋並停止在branch node, y

(這裡y所儲存的key為對應在compressed trie的leaf node中的key)

(本步驟乃在探索x加入後如何影響Patricia的結構)

(只走黑邊，最後停在紅邊指向的節點上)

Step 2: 算出x和y的common prefix

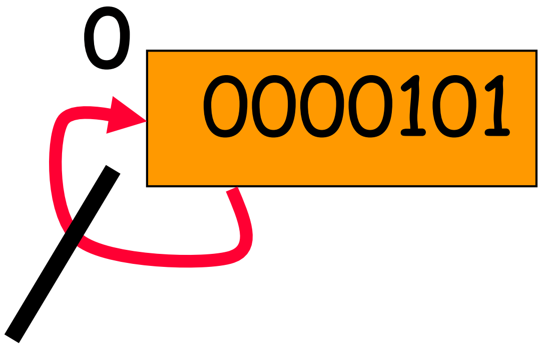
Step 3: 以上述common prefix從root開始再搜尋Patricia一次使確定何處插入一新的的branch node

(意圖在compressed trie裡建立一個branch node來呼應common prefix+1 bit的差異)

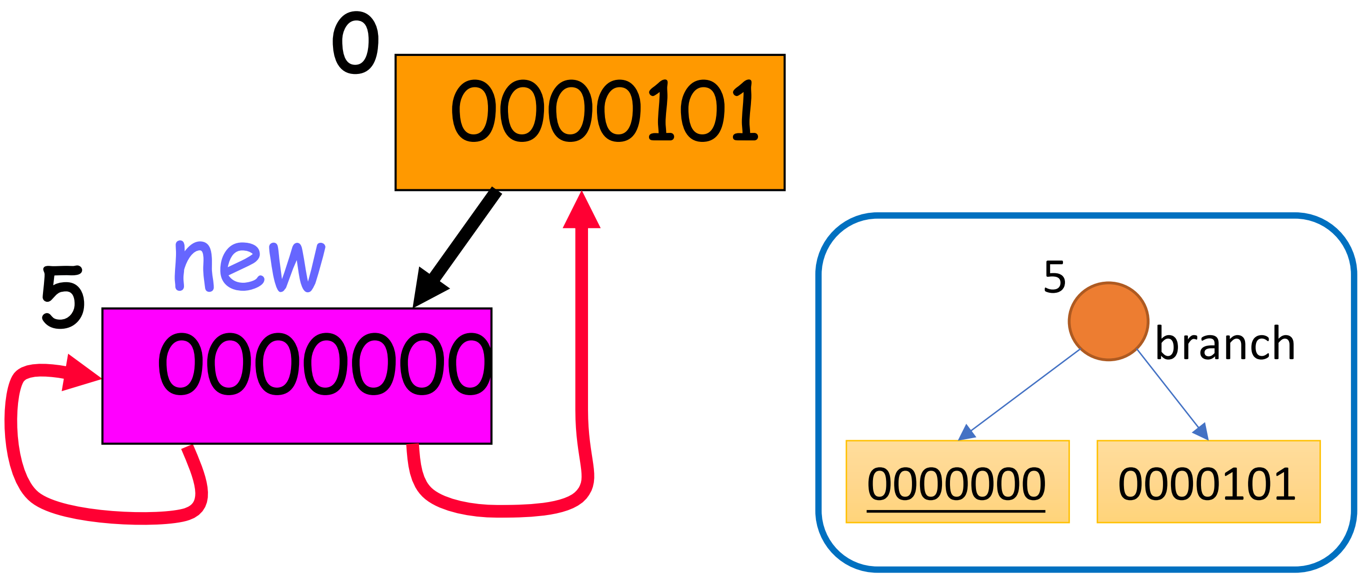
Step 4: 新增branch node 、新增及調整必要的指標、加入x至該branch node、填入common prefix length+1的長度到bit#

Insert

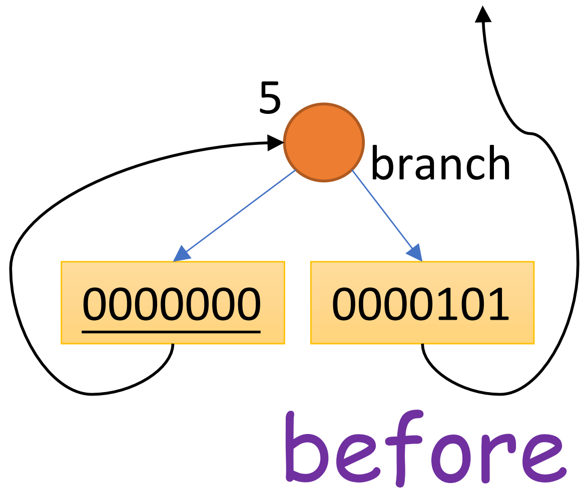
Insert
0000101



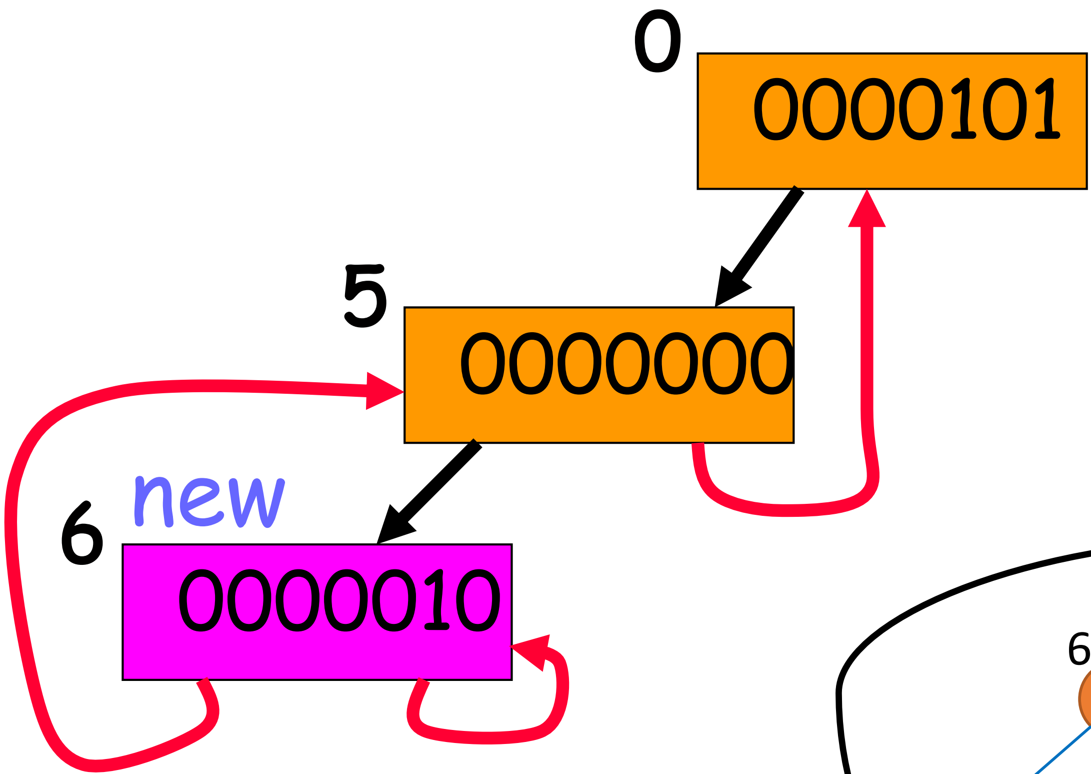
Insert
0000000

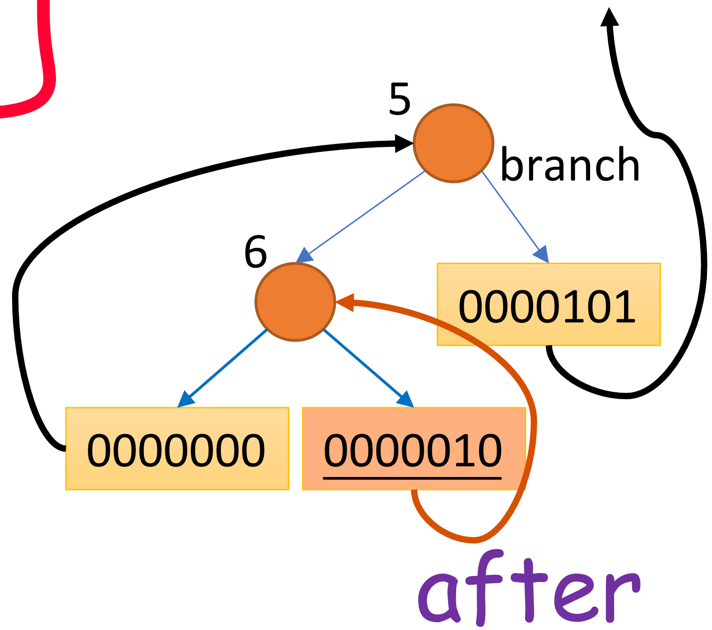


Insert

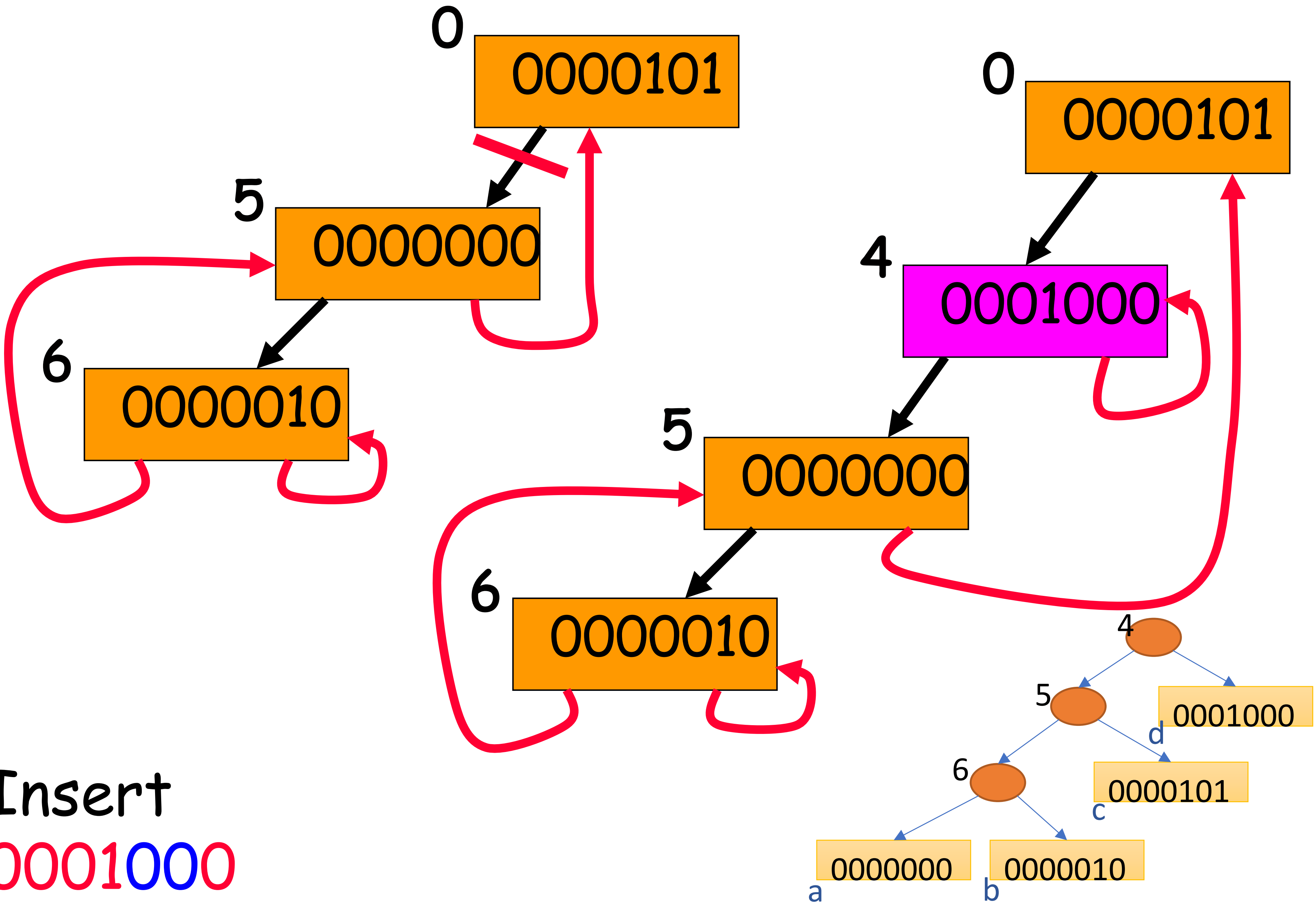


Insert
0000010





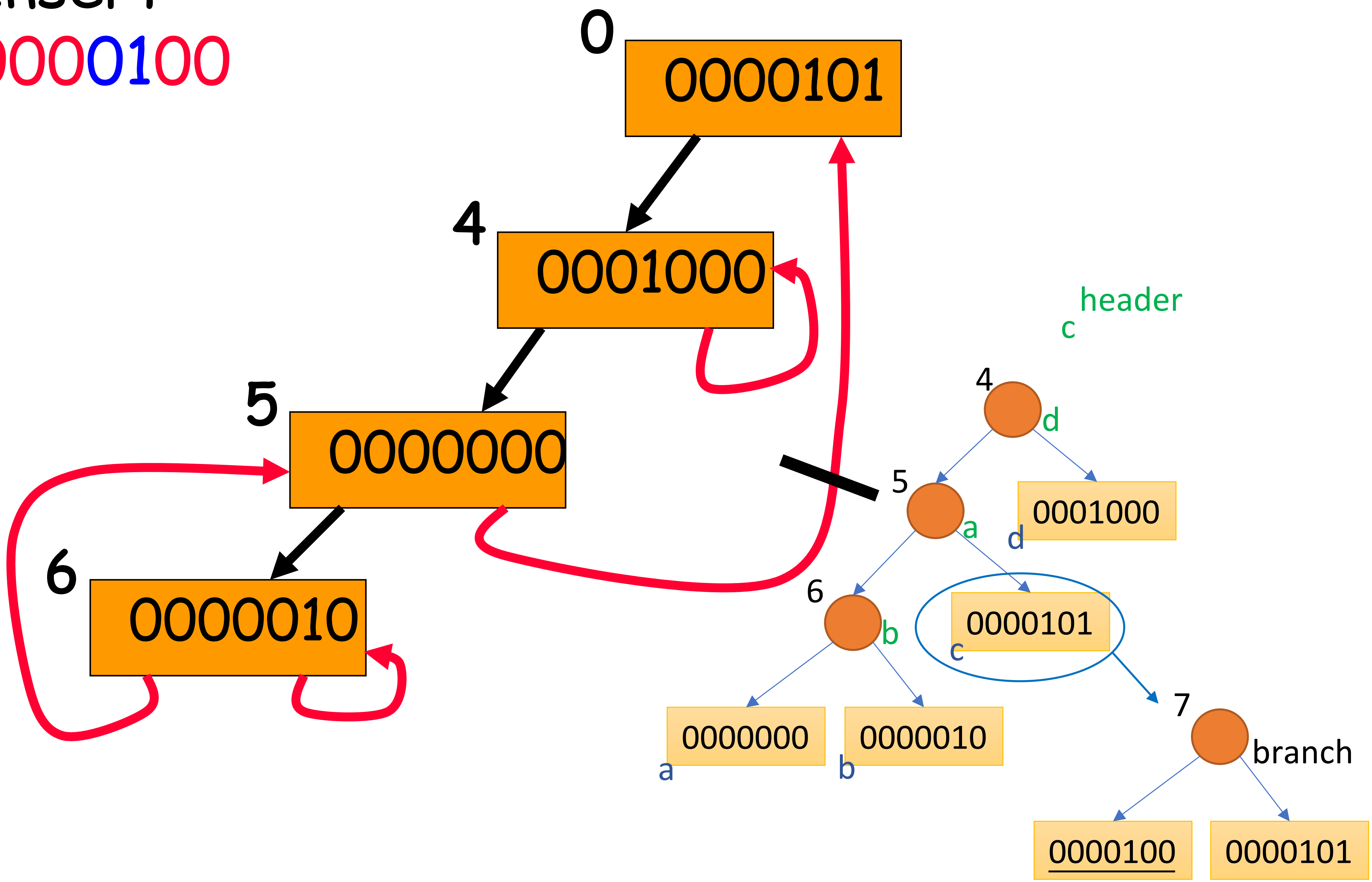
Insert



Insert
0001000

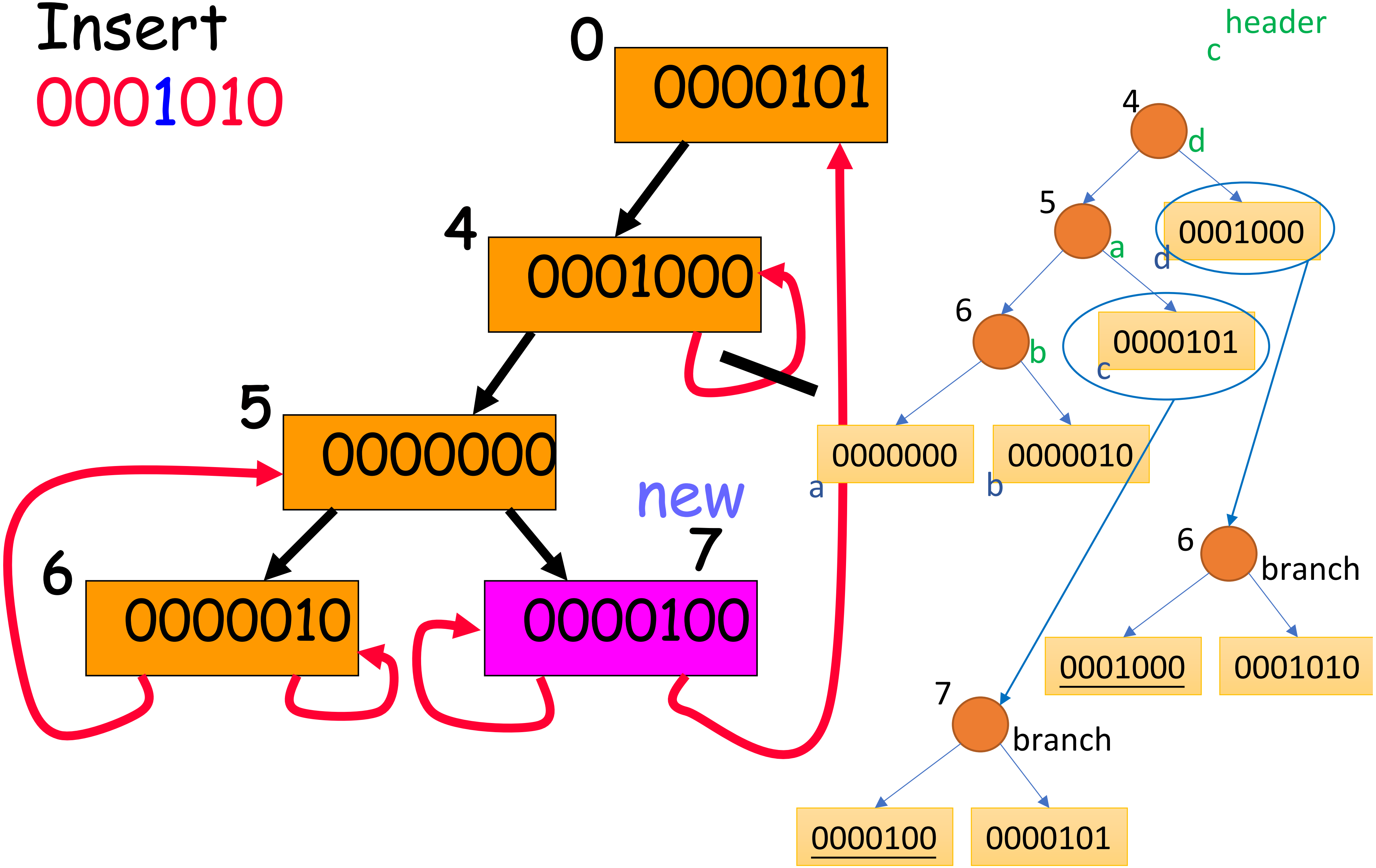
Insert

Insert
0000100

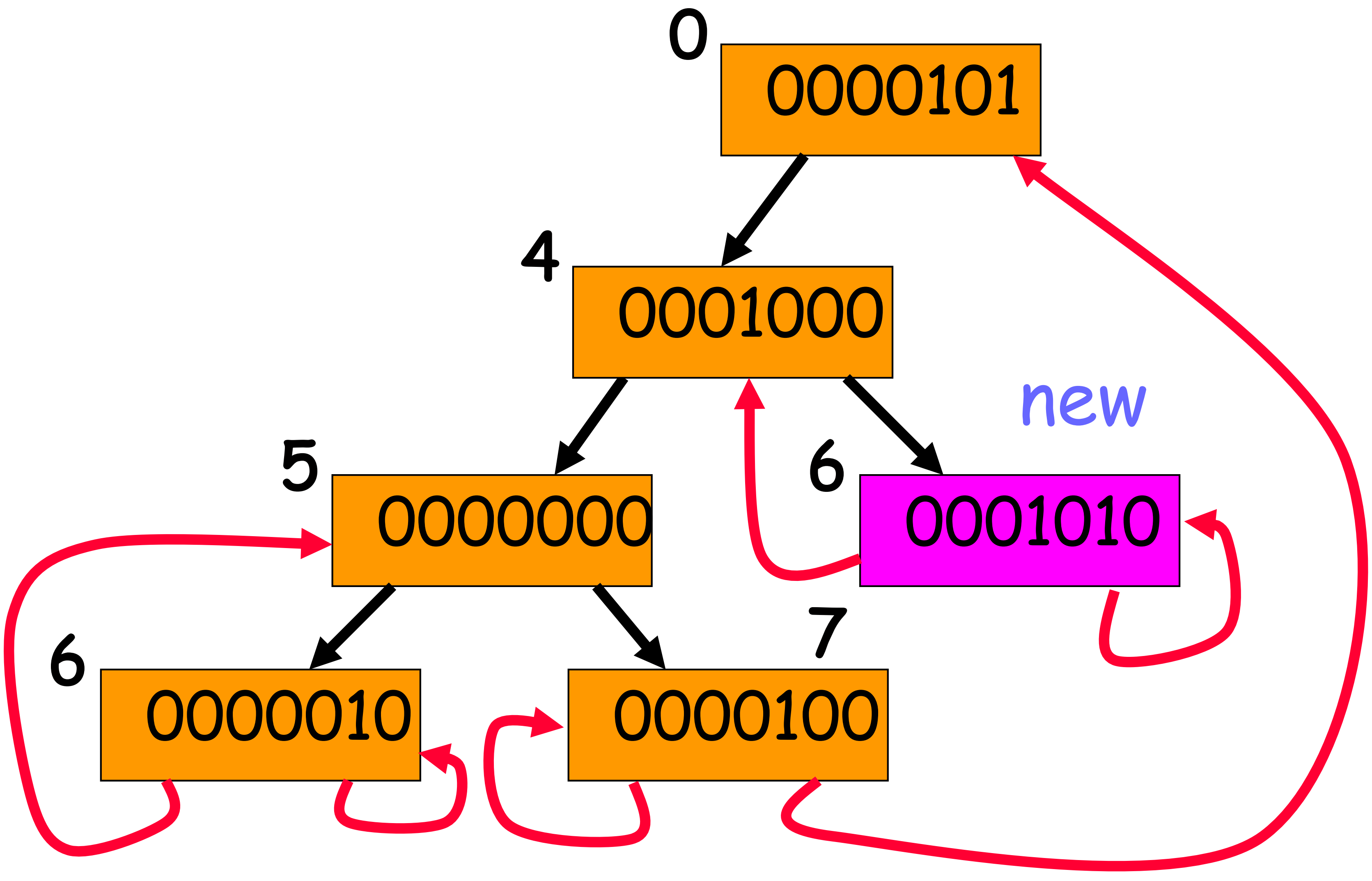


Insert

Insert
0001010



Insert

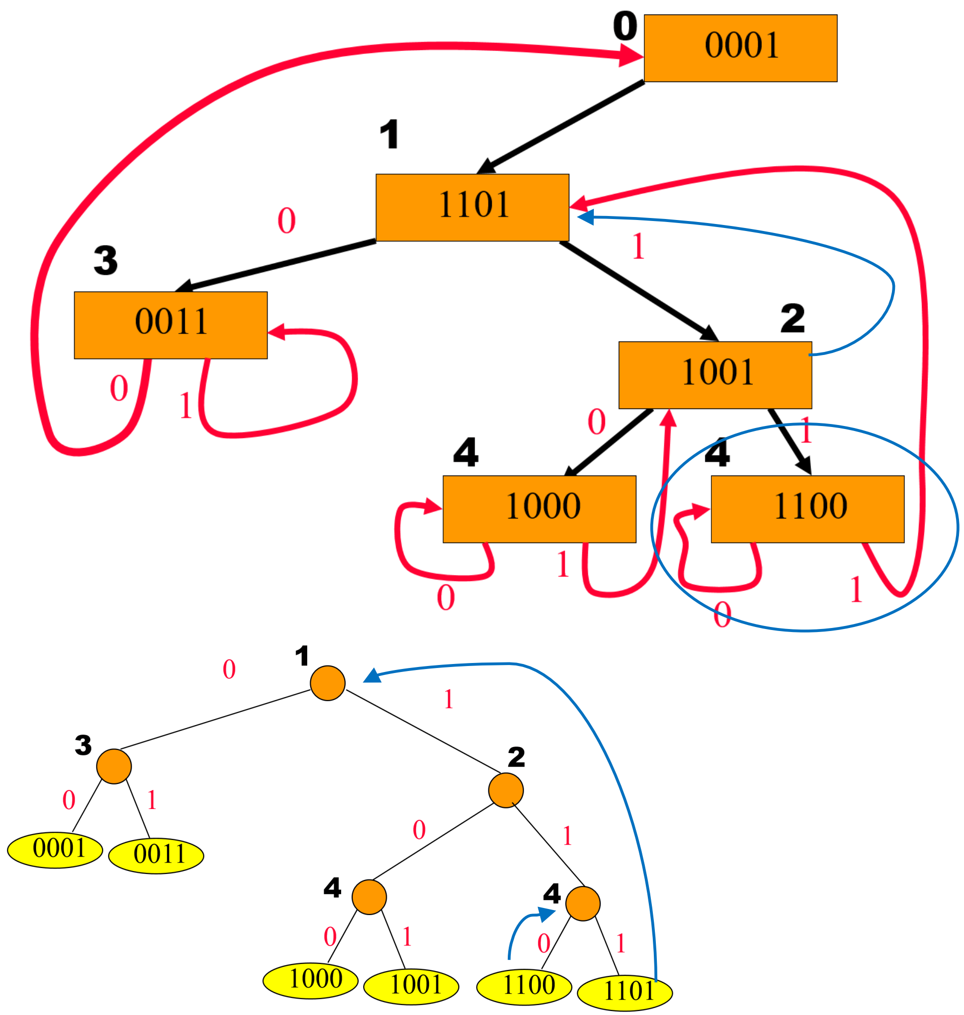

0
0000101
4
0001000
5
0000000
new
6
0001010
6
0000010
7
0000100

Delete

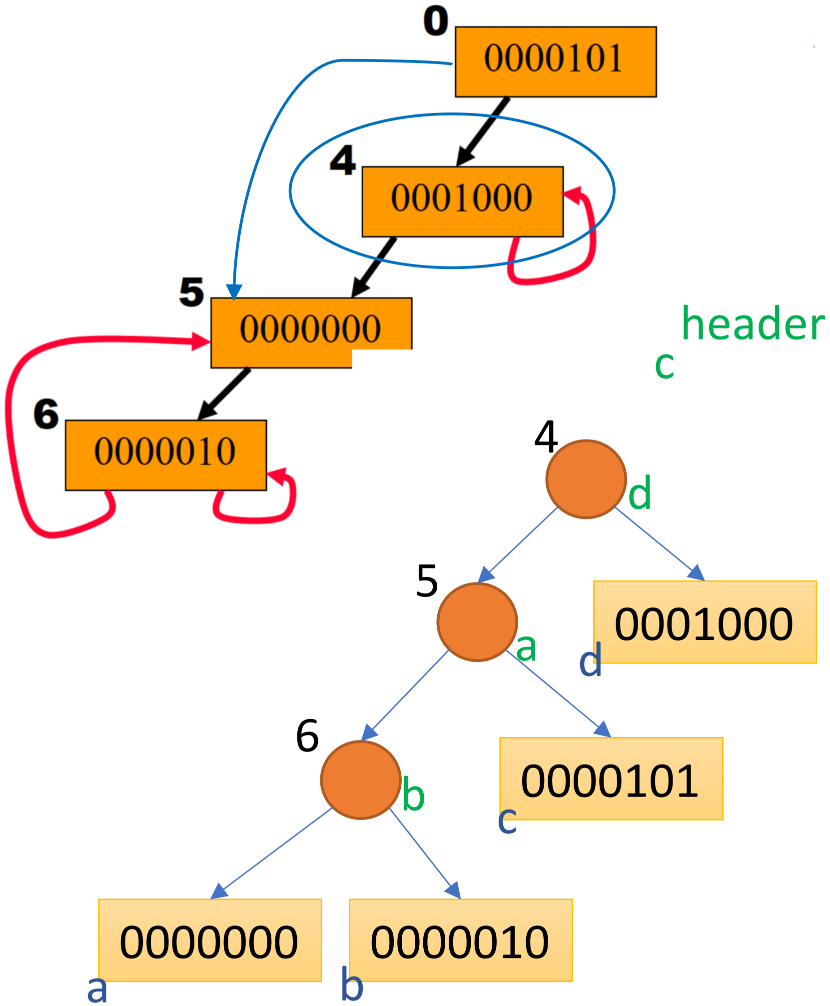
- Let p be the node that contains the dictionary pair that is to be deleted.
- Case 1: p has one self pointer.
  - p搶到了parent的位置
- Case 2: p has no self pointer.
  - p沒搶到parent的位置 (而被安排在某個ancestor的位置)

Examples

Delete 1100

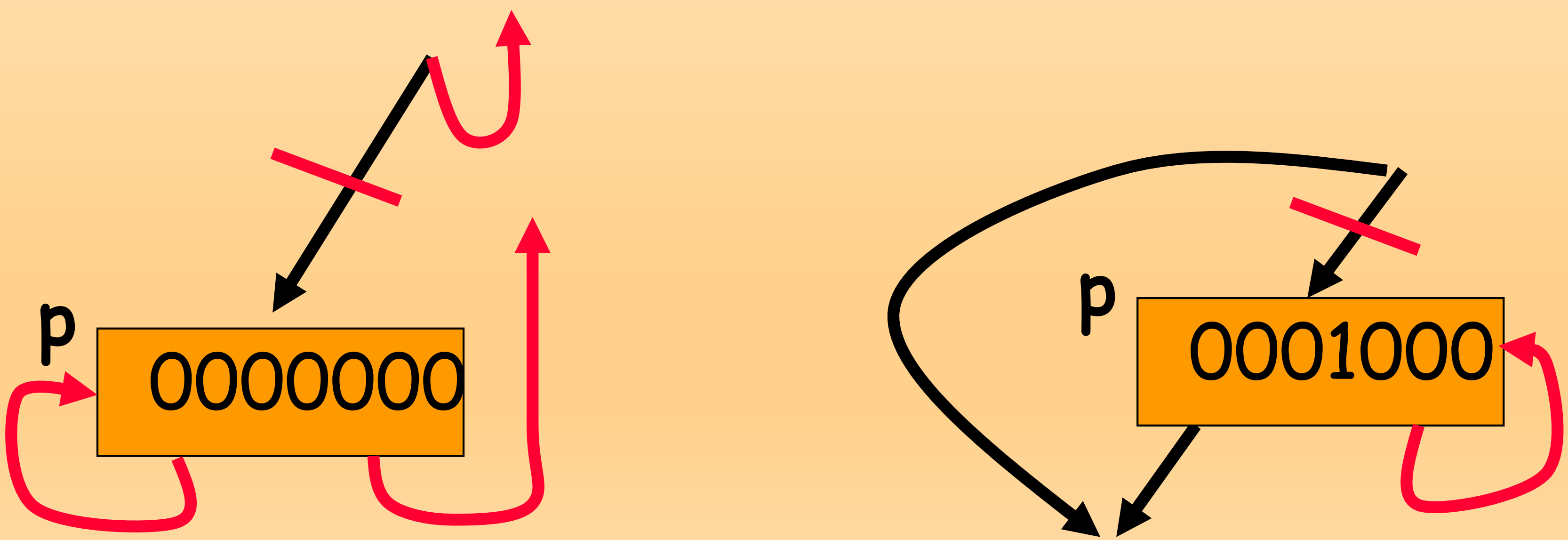


Delete 0001000



p Has One Self Pointer (p is to deleted)

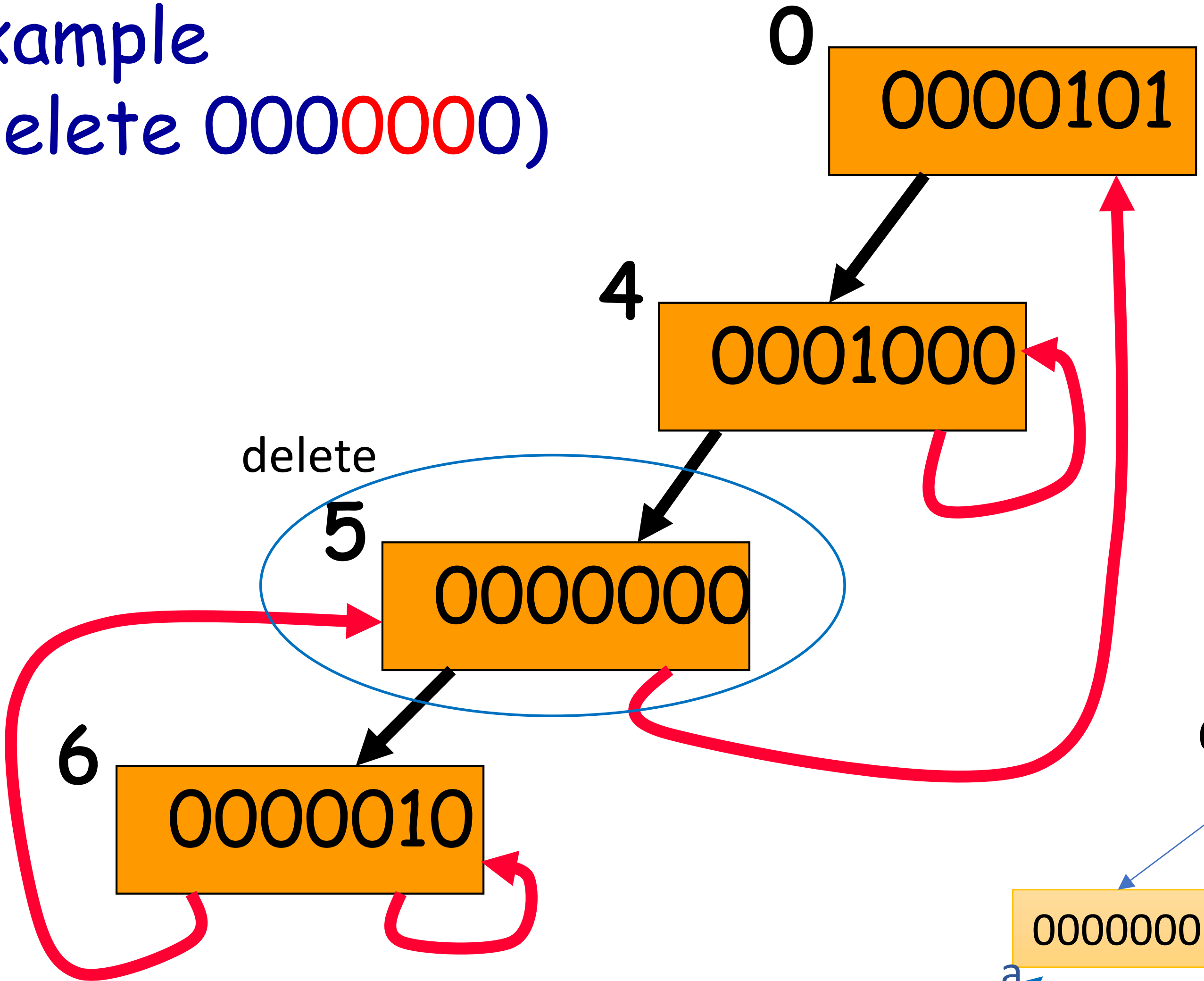
- p = header => trie is now empty.
  - Set trie pointer to null.
- p != header => remove node p and update pointer to p.

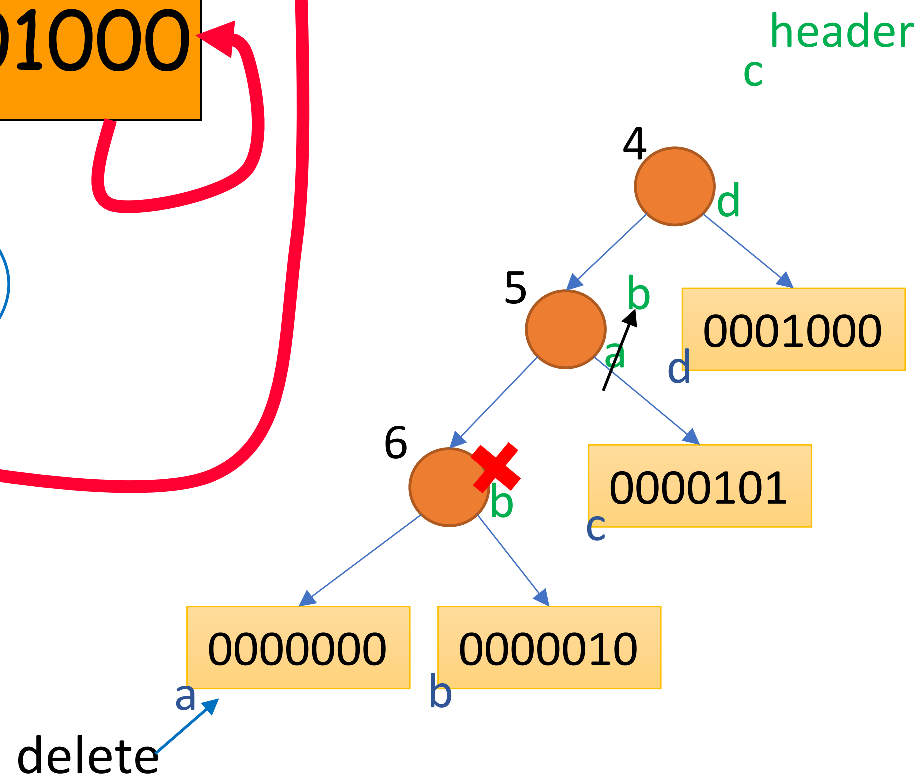


Delete

- Let p be the node that contains the dictionary pair that is to be deleted.
- Case 1: p has one self pointer.
  - p搶到了parent的位置
- Case 2: p has no self pointer.
  - p沒搶到parent的位置 (而被安排在ancestor的位置)

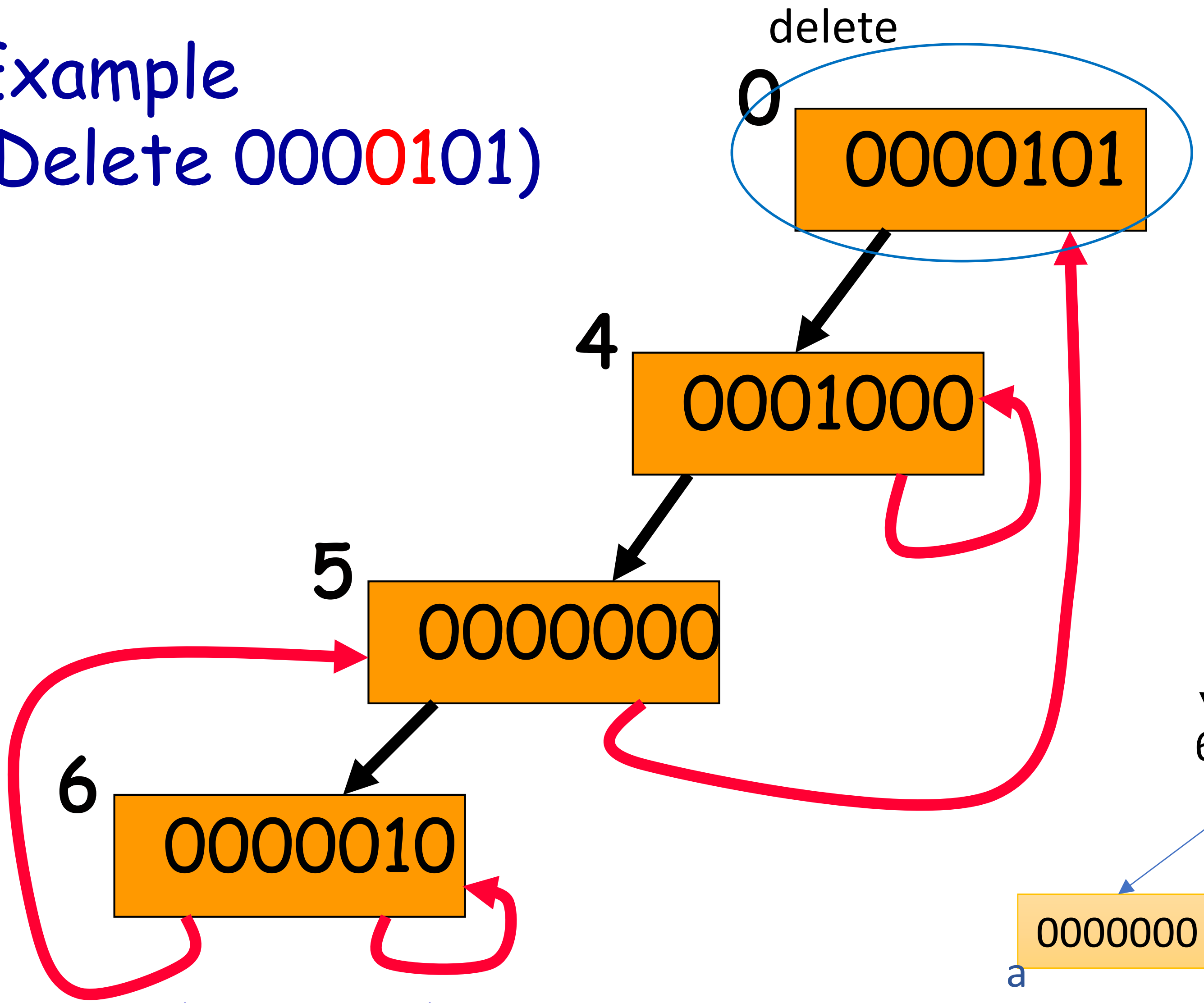
Example
(Delete 0000000)

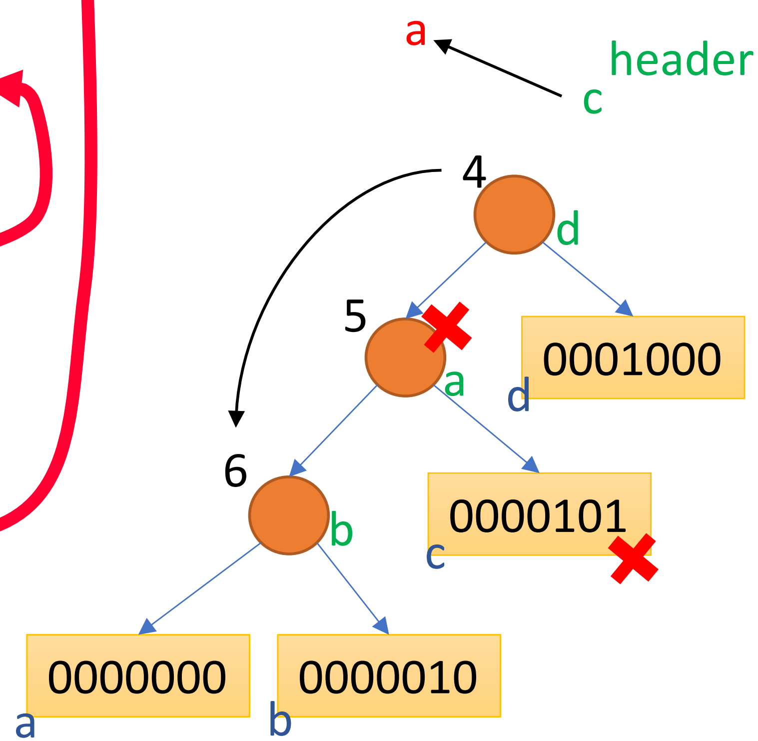




Step 1: 找到a (i.e., 0000000)
Step 2: 找到與a相像的資料b，且當時b贏了a而佔住了一branch node (in terms of common prefix+1)
PS: Steps 1 & 2在探索tries的結構!!!
Step 3: 刪除一branch node (i.e., 綠色的b in corresponding compressed tries), representing a與b的maximum common prefix+1
Step 4: b取代a (a所在的branch node改為儲存b的資料)

Example
(Delete 0000101)

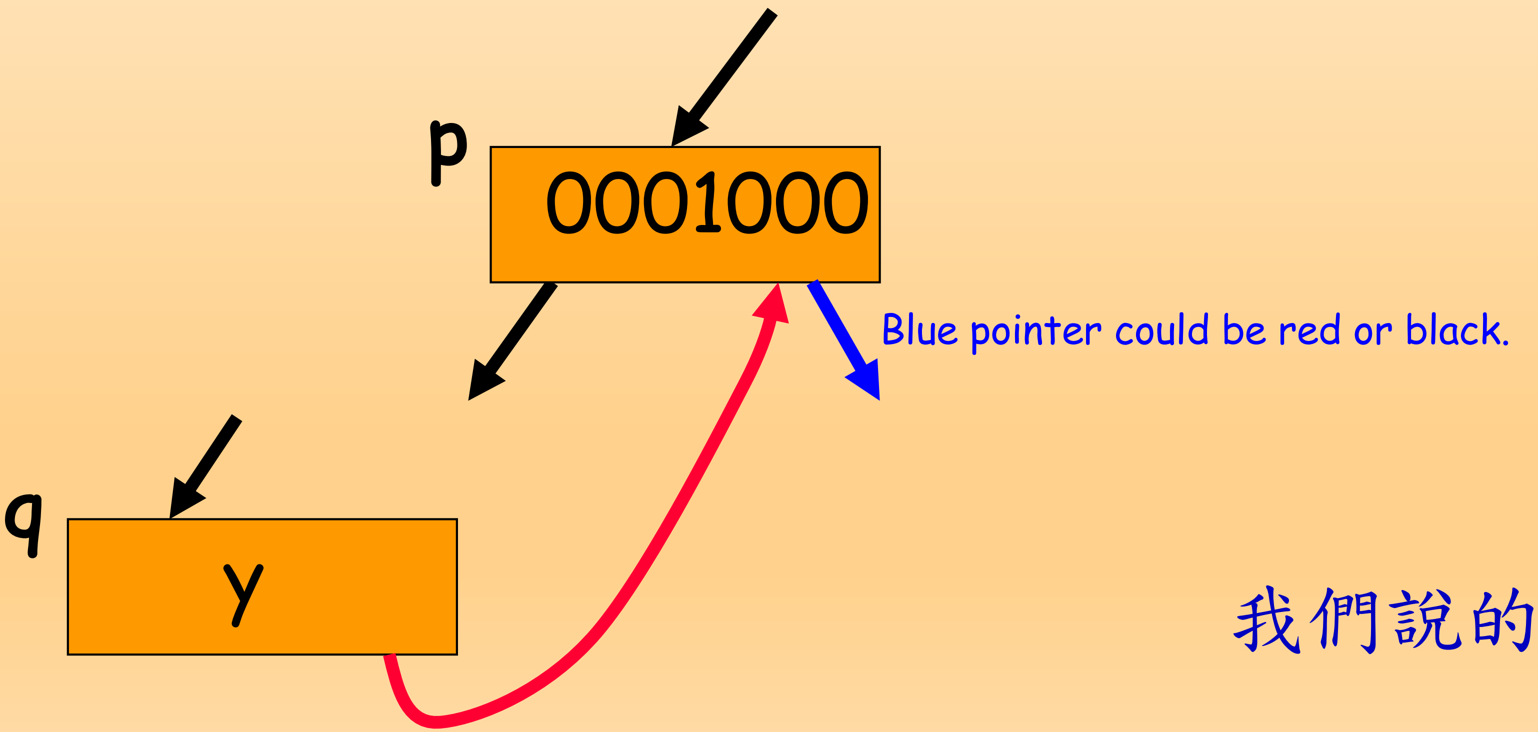




Step 1: 找到c (i.e., 0000101)
Step 2: 找到與c相像的資料a (or b)，且當時a (not b) 贏了c而佔住了一branch node (in terms of common prefix+1)
Step 3: a取代c (c所在的branch node改為儲存a的資料)
Step 4: 刪除一branch node (i.e., 綠色的a in corresponding compressed tries)
Step 5: d指向b

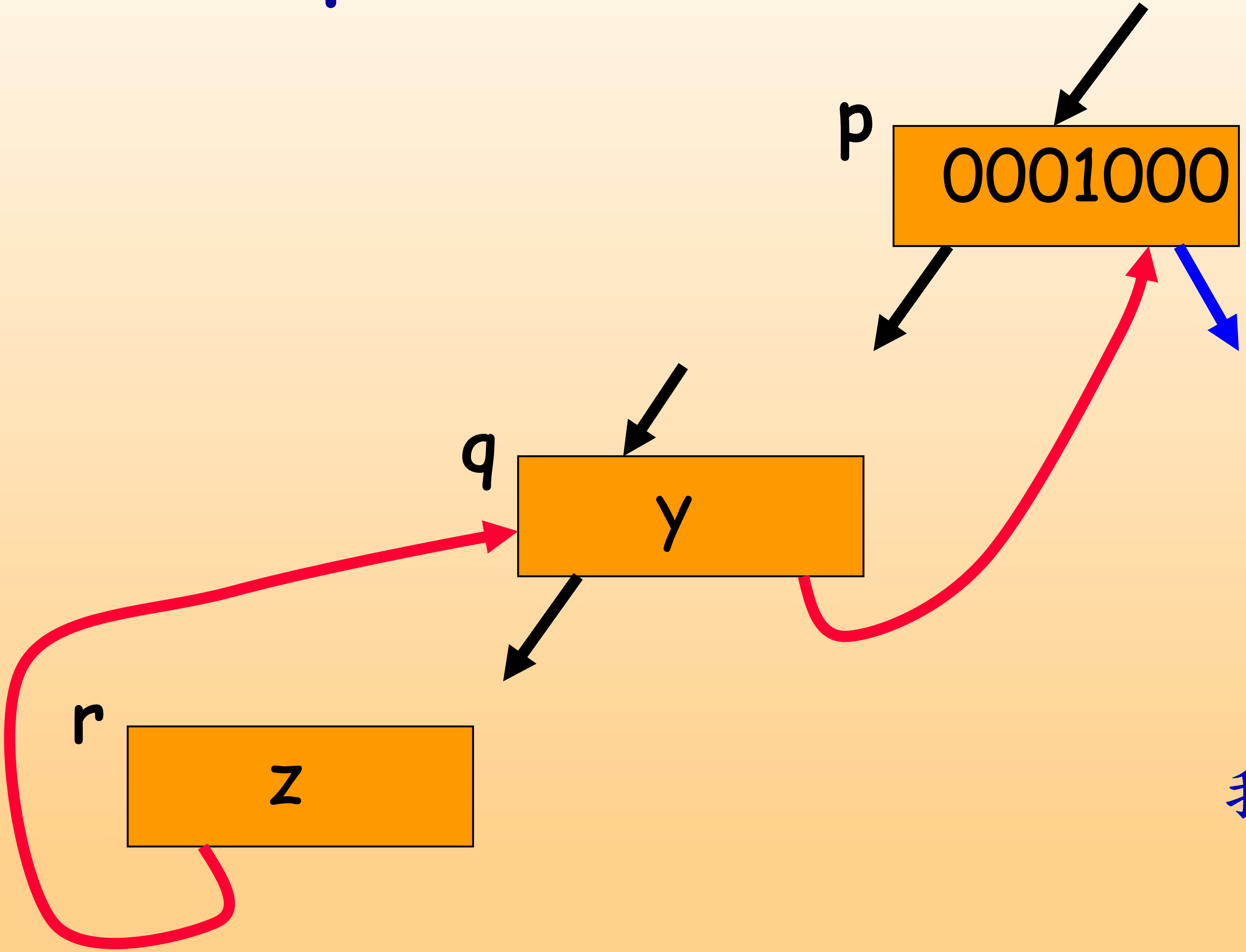
p Has No Self Pointer (Delete 0001000)

- Let q be the node that has a back (red) pointer to p.
- Node q was determined during the search for the pair with the deleted key k.



我們說的Step 1

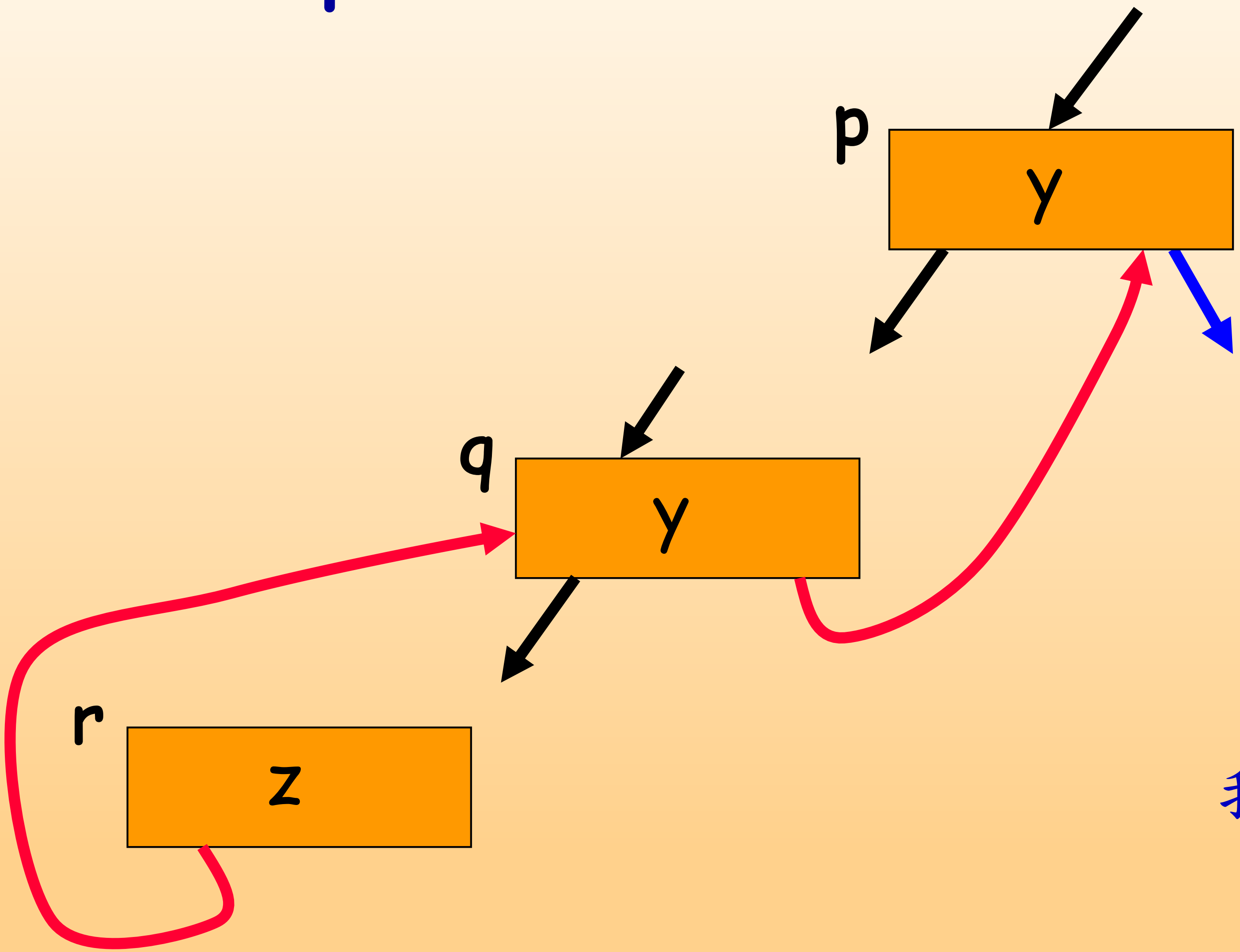
p Has No Self Pointer



我們說的Step 2

- Use the key y in node q to find the unique node r that has a back pointer to node q.

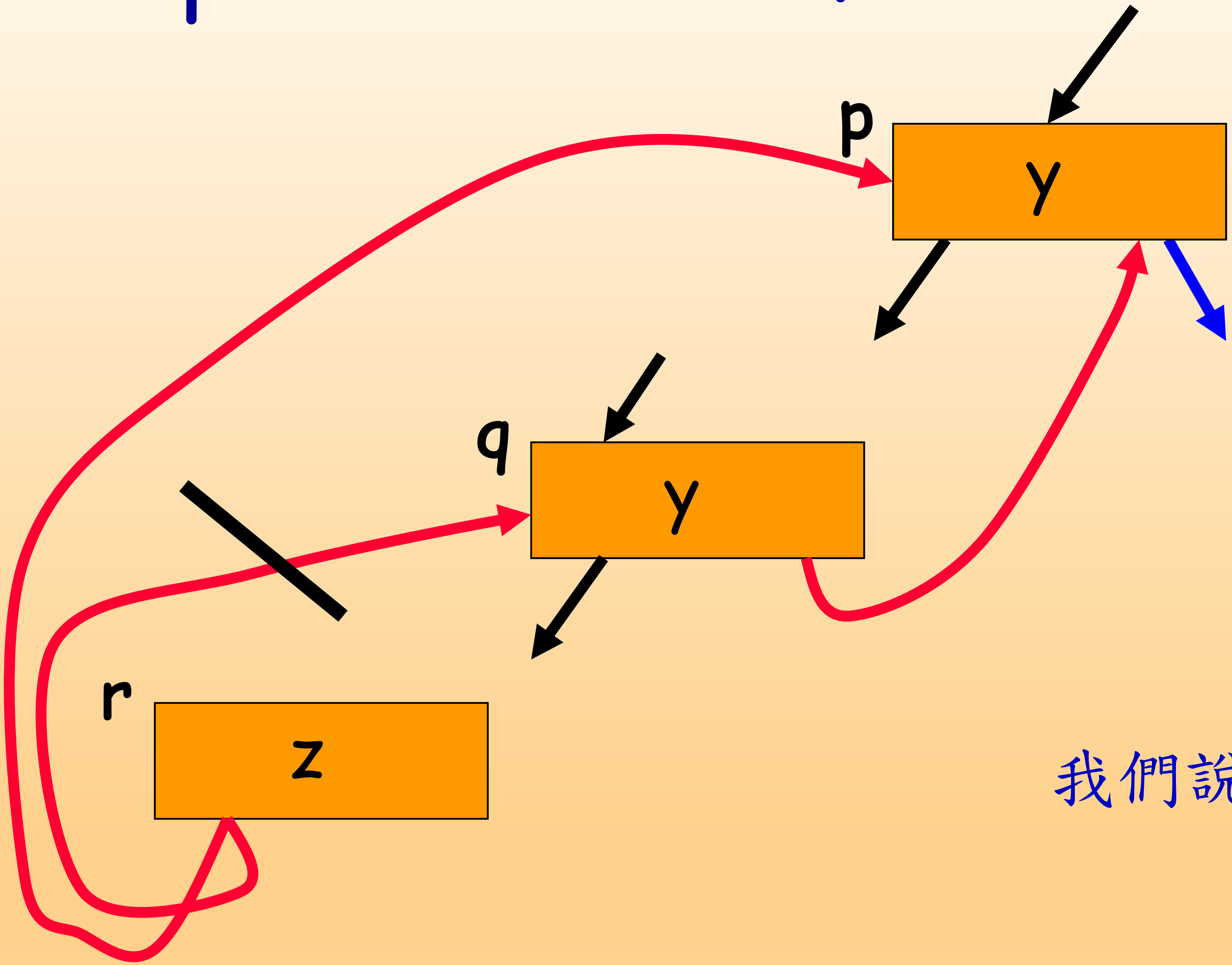
p Has No Self Pointer



我們說的Step 3

- Copy the pair whose key is y to node p.

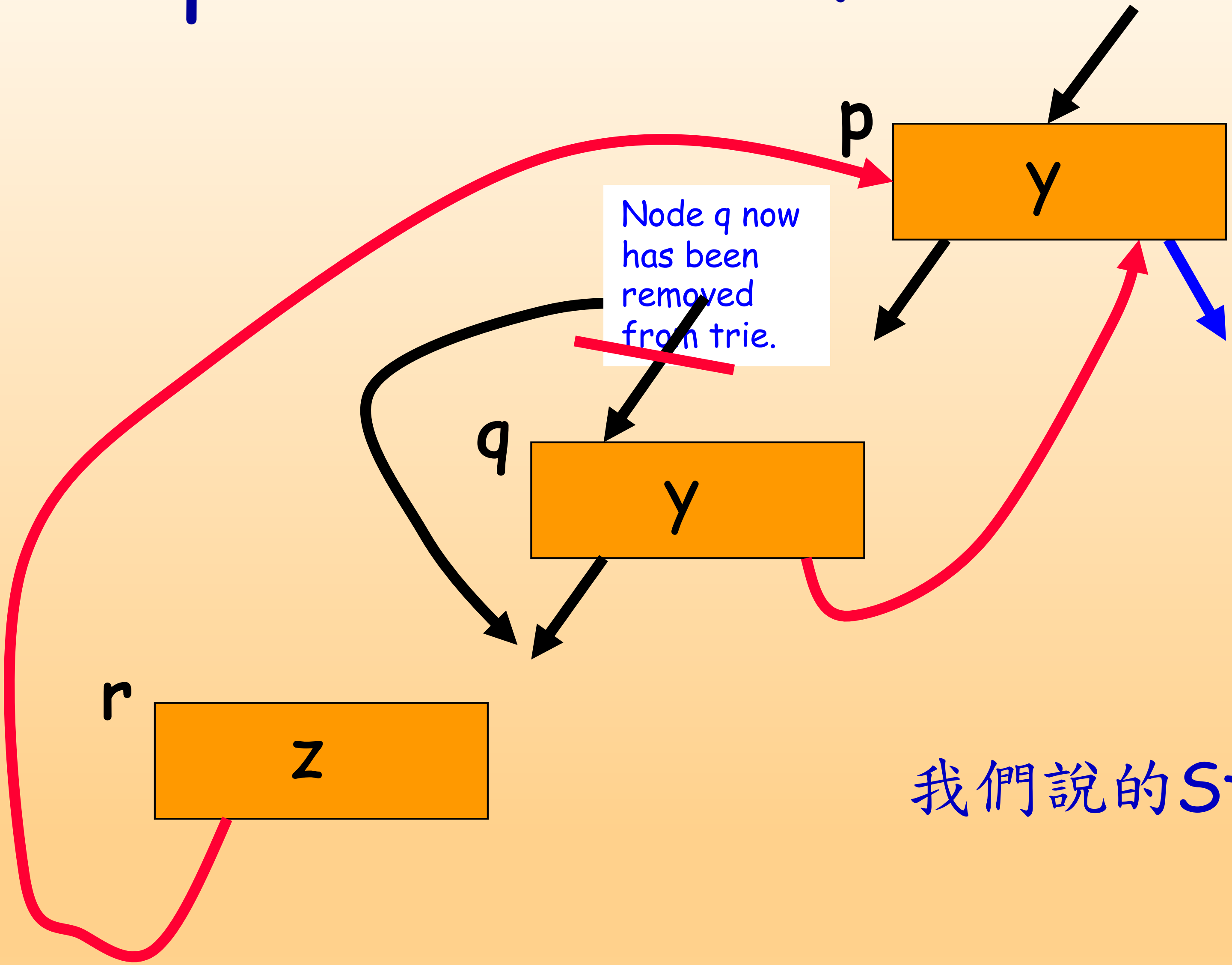
p Has No Self Pointer



我們說的Step 3

- Change back pointer to q in node r to point to node p.

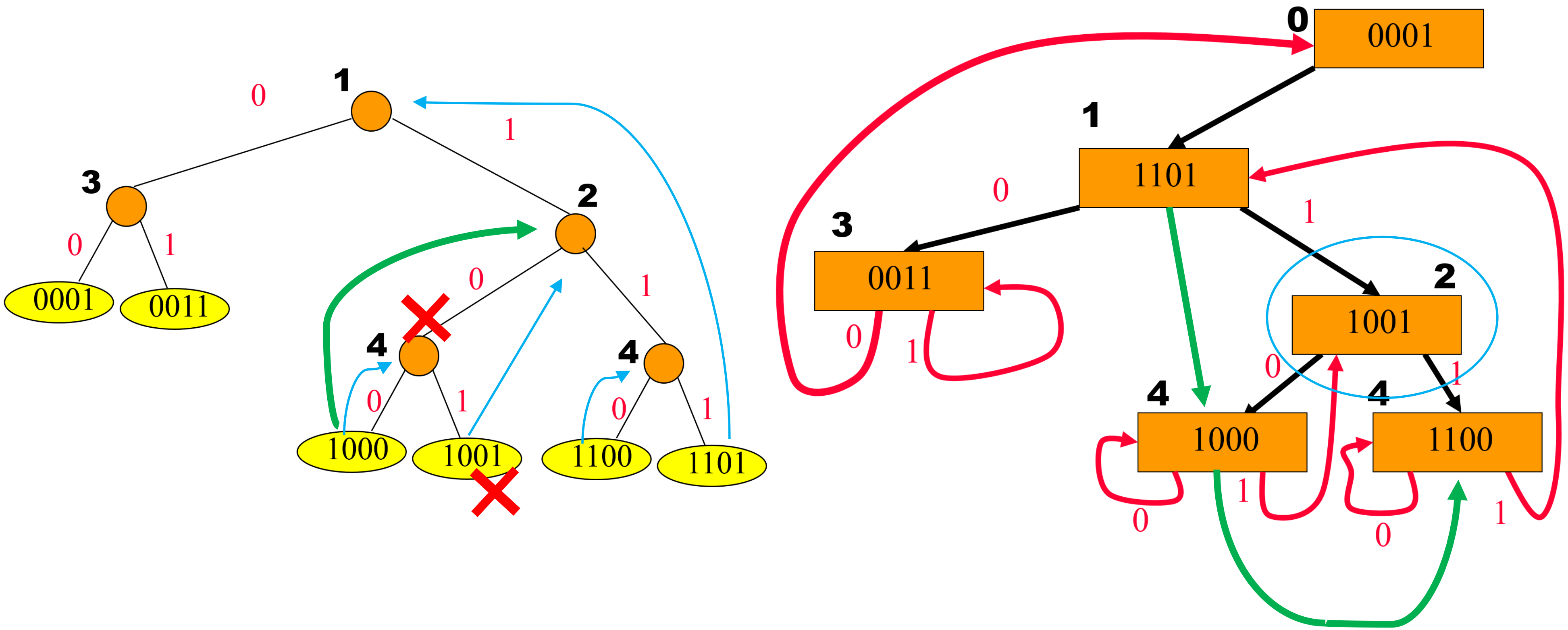
p Has No Self Pointer



我們說的Steps 4 & 5

- Change forward pointer to q from parent(q) to child of q.

Another Example (Delete 1001)



Example (Delete 1101)