HAMT

Hash Array Mapped Trie

Ideal Hash Trees (Bagwell, 2001)

hash

key3	0x3b	obj3
key1	0x3b	obj1
key0	0x0e	obj0
key2	0x1a	obj2
…	…	…

hash+

key3	0x3b	obj3
key1	0x3b	obj1
key0	0x0e	obj0
key2	0x1a	obj2
key4	0x22	obj4
…	…	…

(hash-set hash key4 obj4) ⟶ **hash+**

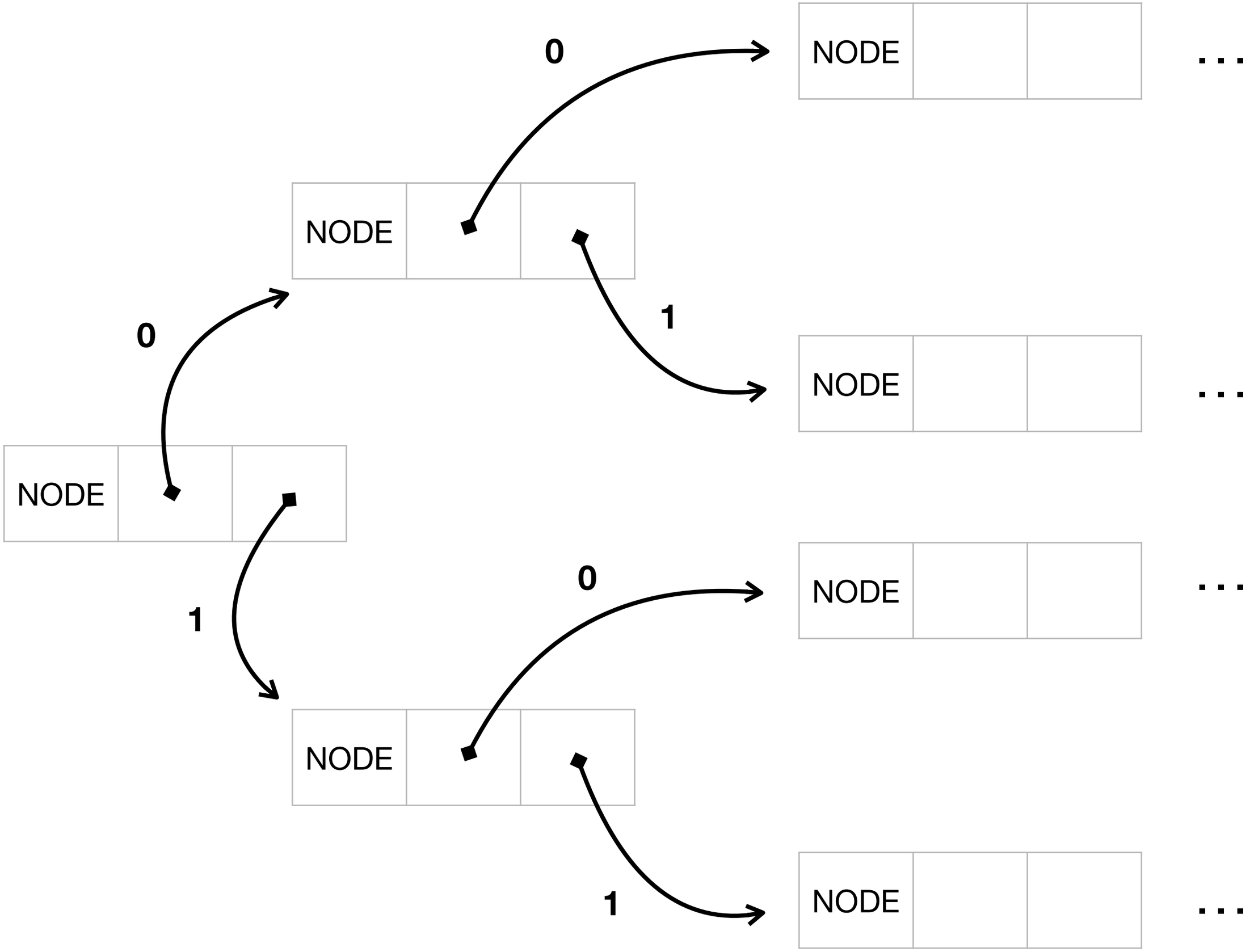
Persistent hashes

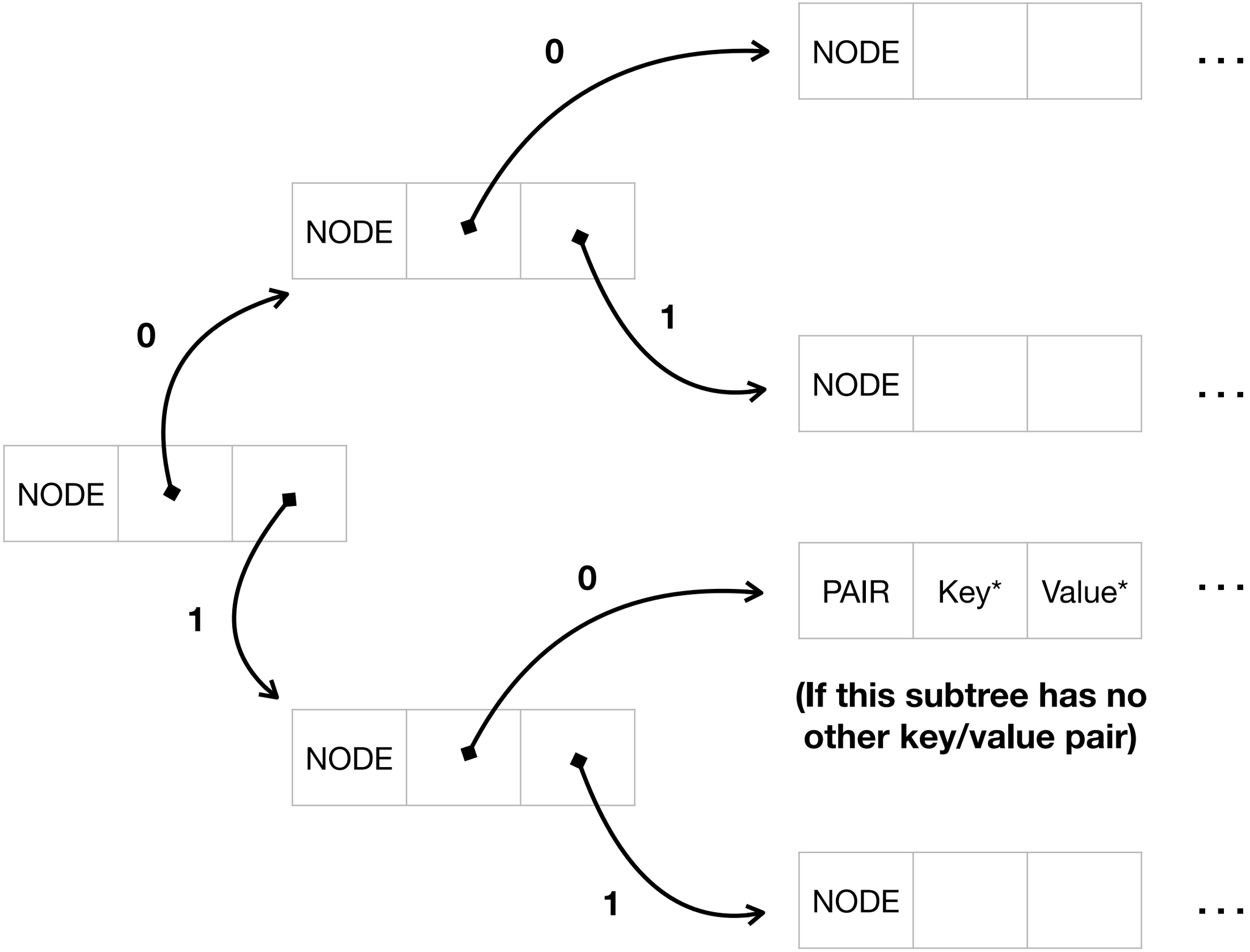
- Instead, we want a immutable linked structure that can be partially updated while portions of the old hash persist.

- Hash Array Mapped Trie (HAMT); From “Ideal hash trees” (Bagwell, 2001)

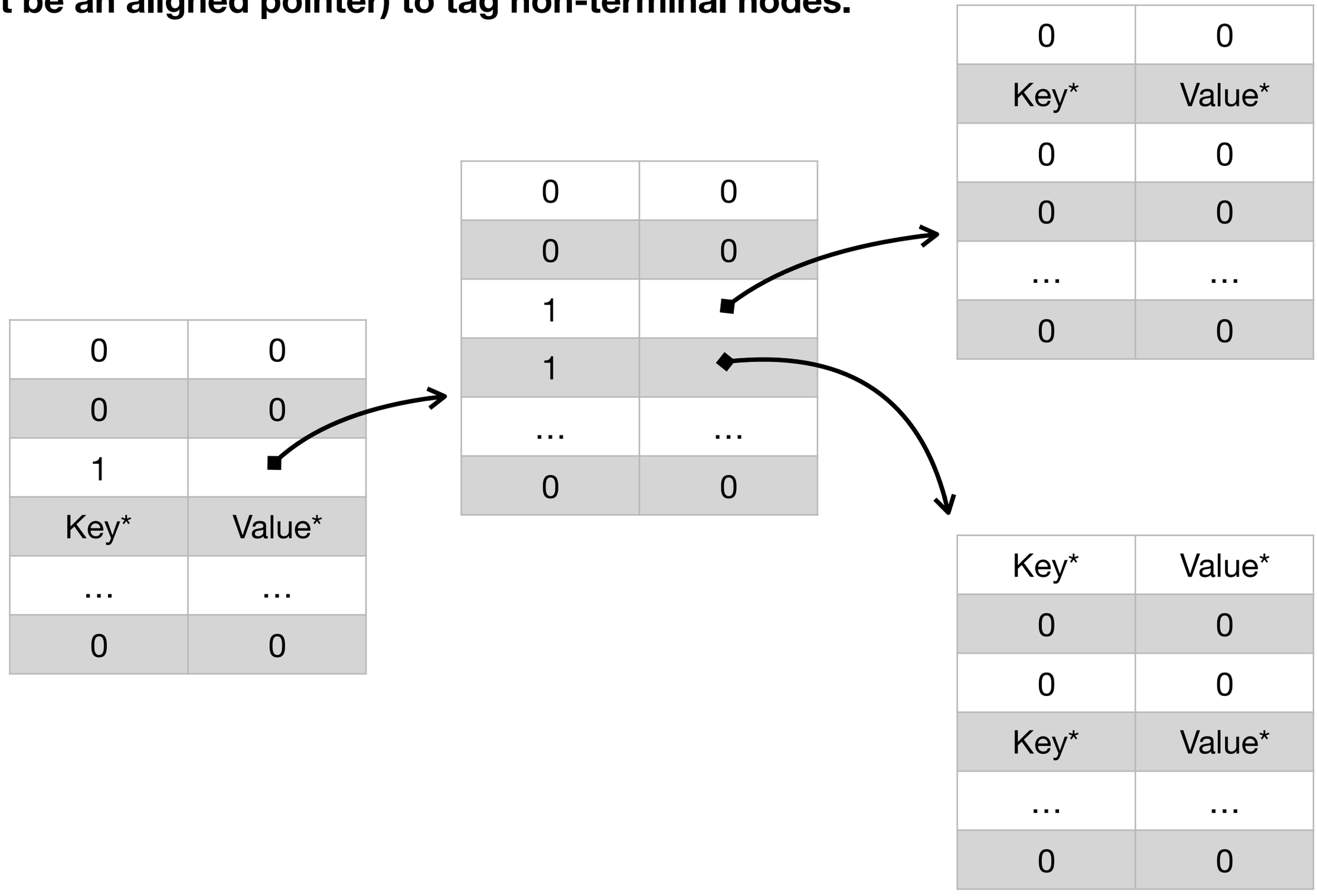
- Traditional balanced trees (AVL, RB, B-trees, etc) use mutability to dynamically re-balance when necessary.

- HAMT first hashes its keys to obtain good dispersion characteristics, as a Hash Table would, but then uses a linked trie (prefix tree) structure.

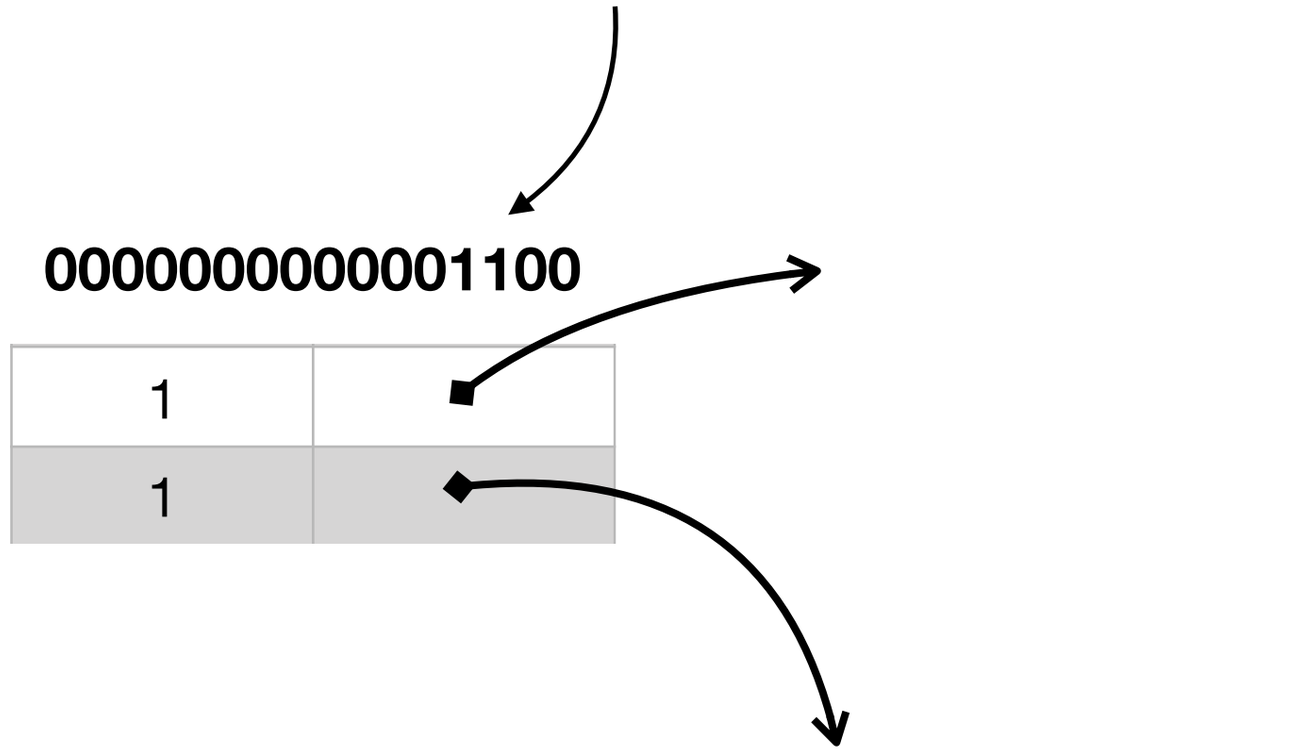

NODE
0
1
NODE
0
1
NODE
0
1
NODE
NODE
NODE
NODE
NODE
NODE
. . .
. . .
. . .


NODE
0
NODE
1
NODE
. . .
NODE
0
1
NODE
. . .
PAIR
Key*
Value*
. . .
(If this subtree has no other key/value pair)
NODE
0
1
NODE
. . .

Increase the branching factor. Use the value 1 (can’t be an aligned pointer) to tag non-terminal nodes.



These 1s indicate the 2nd and 3rd indices were non-null and either contain a Key/Value or a pointer to an inner node.



Then, we can compress these nodes using a bitmap.

```
__builtin_popcount(u32)
__builtin_popcountll(u64)
```

```
u32 popcount(u32 x)
{
    x = x - ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x + (x >> 4)) & 0x0F0F0F0F;
    x = x + (x >> 8);
    x = x + (x >> 16);
    return x & 0x0000003F;
}
```

Technique is called SIMD within a register (SWAR).

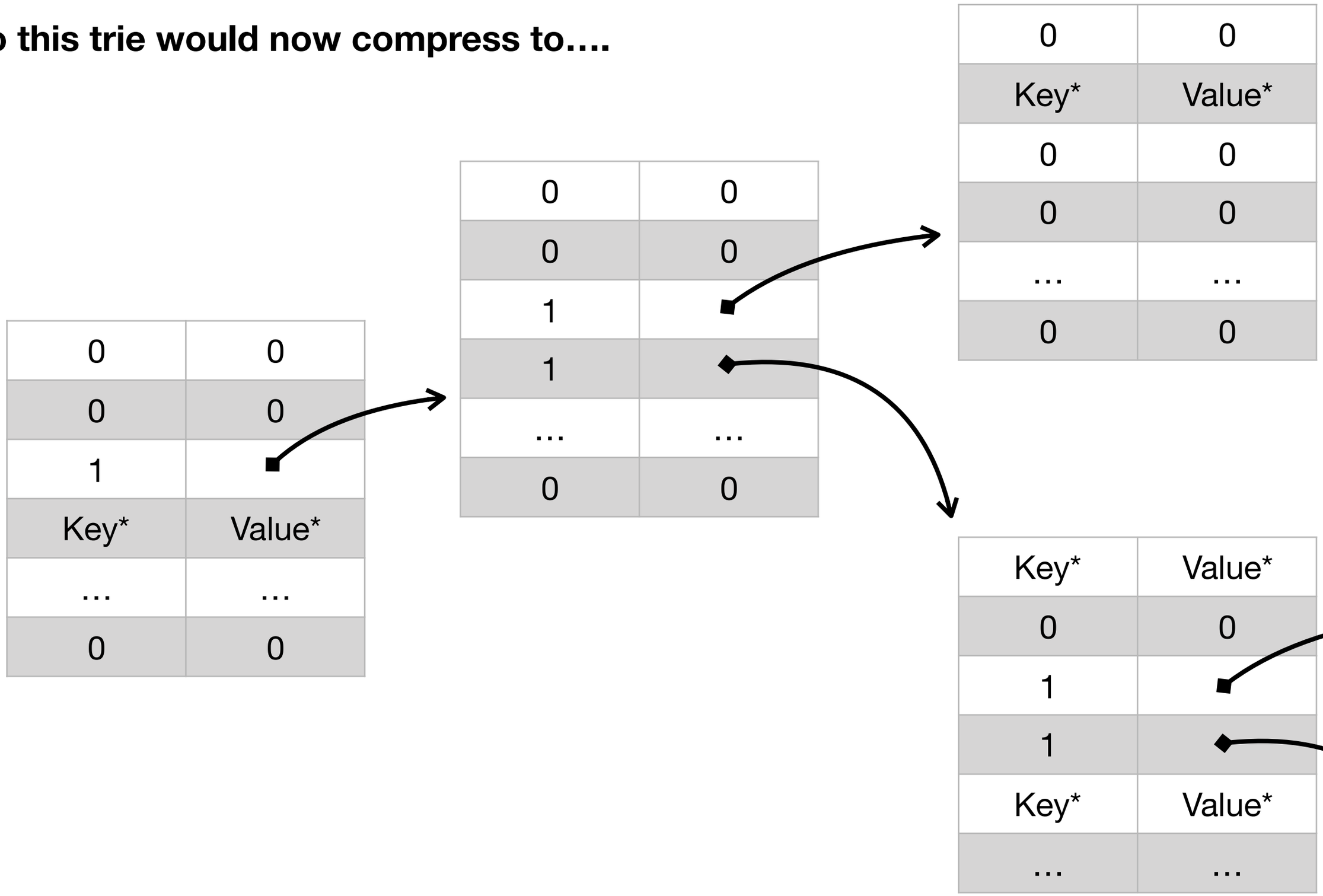
Faster population counts using AVX2 Instructions (Mula, et al. 2017)

length of the bitmap

```
__builtin_popcountll(bm << (max - hindex))
```

**Returns the number of 1s preceding the hash-index we want,
which is the same as the compressed index.**

So this trie would now compress to….



This trie, with 31 or 63bit bitmaps in the parent node.