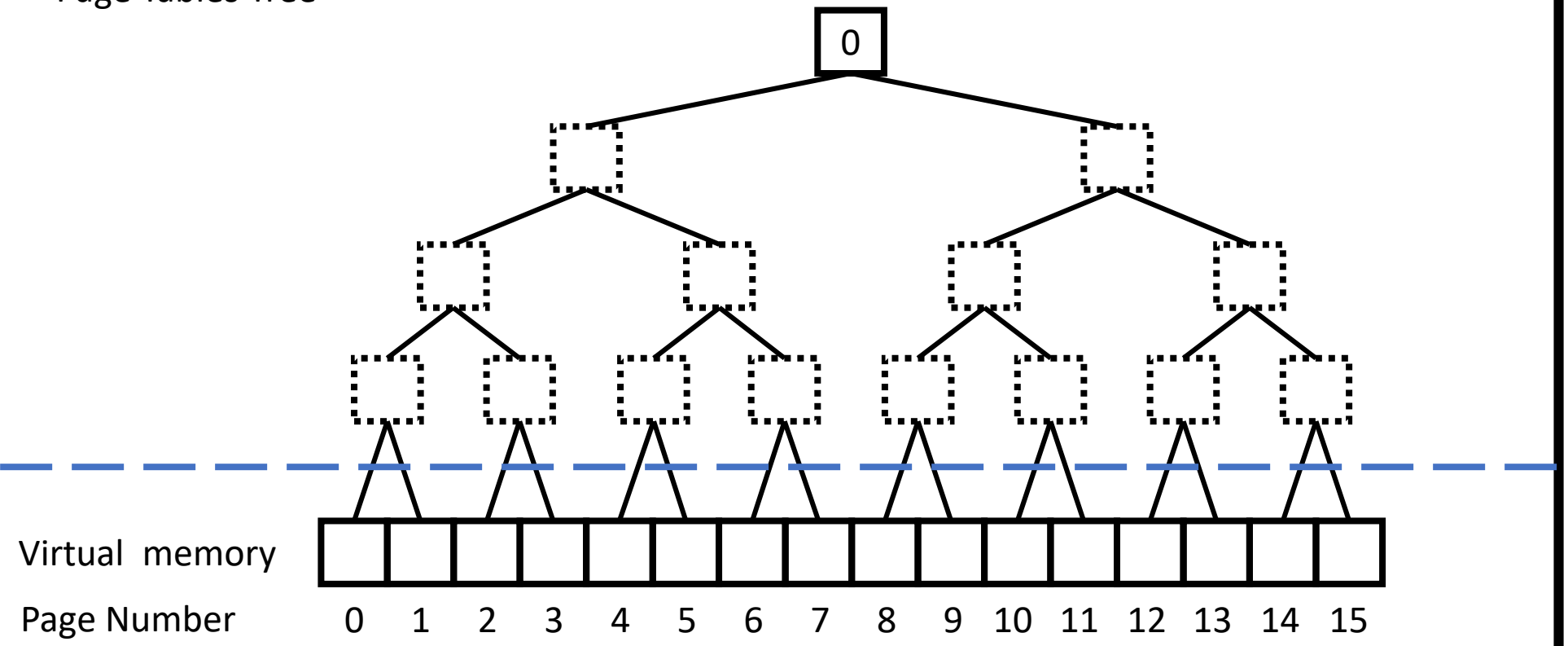
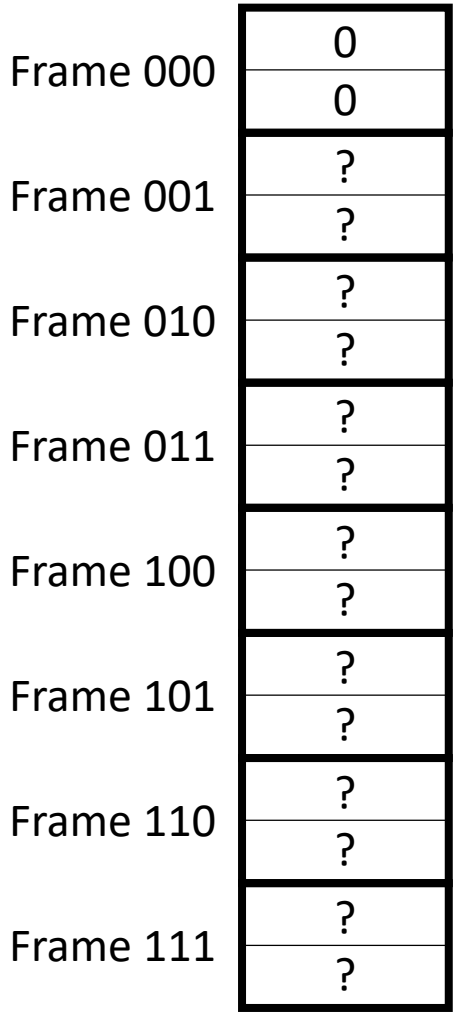
Page Tables Tree



Physical memory

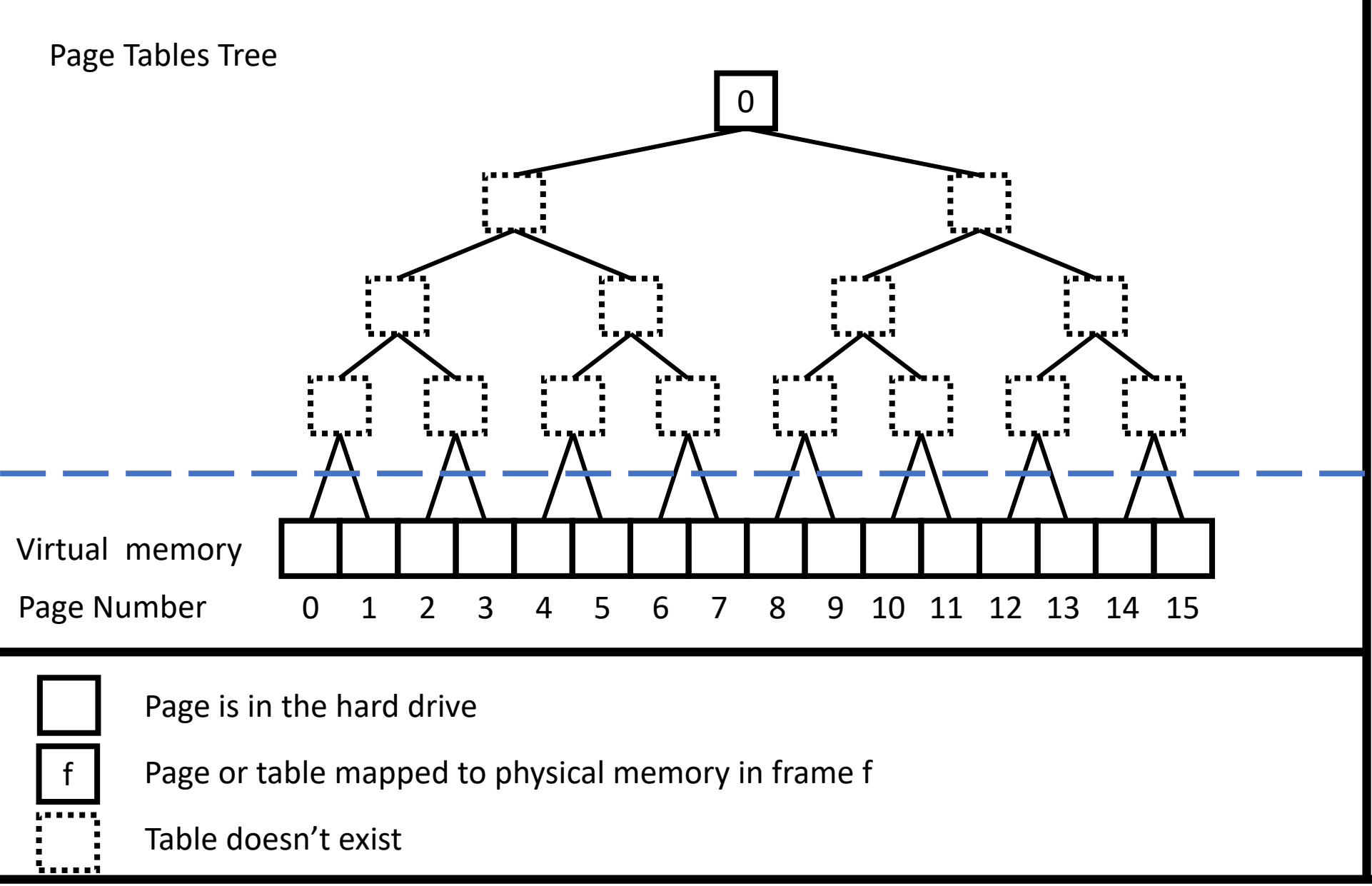


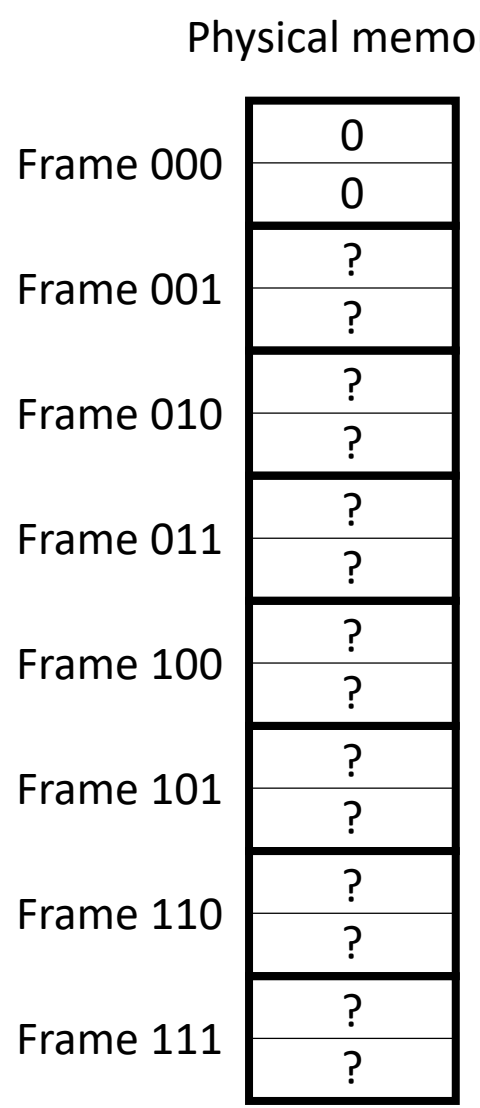
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

Virtual memory size is $32=2^5$, page/frame/table size is $2=2^1$, therefore we need ceil((5-1)/1) = 4 layers in the tree.
A single page table for all pages would require 16 rows, but we only have 16 words in our entire physical memory!
That's why we must use a hierarchical page table.





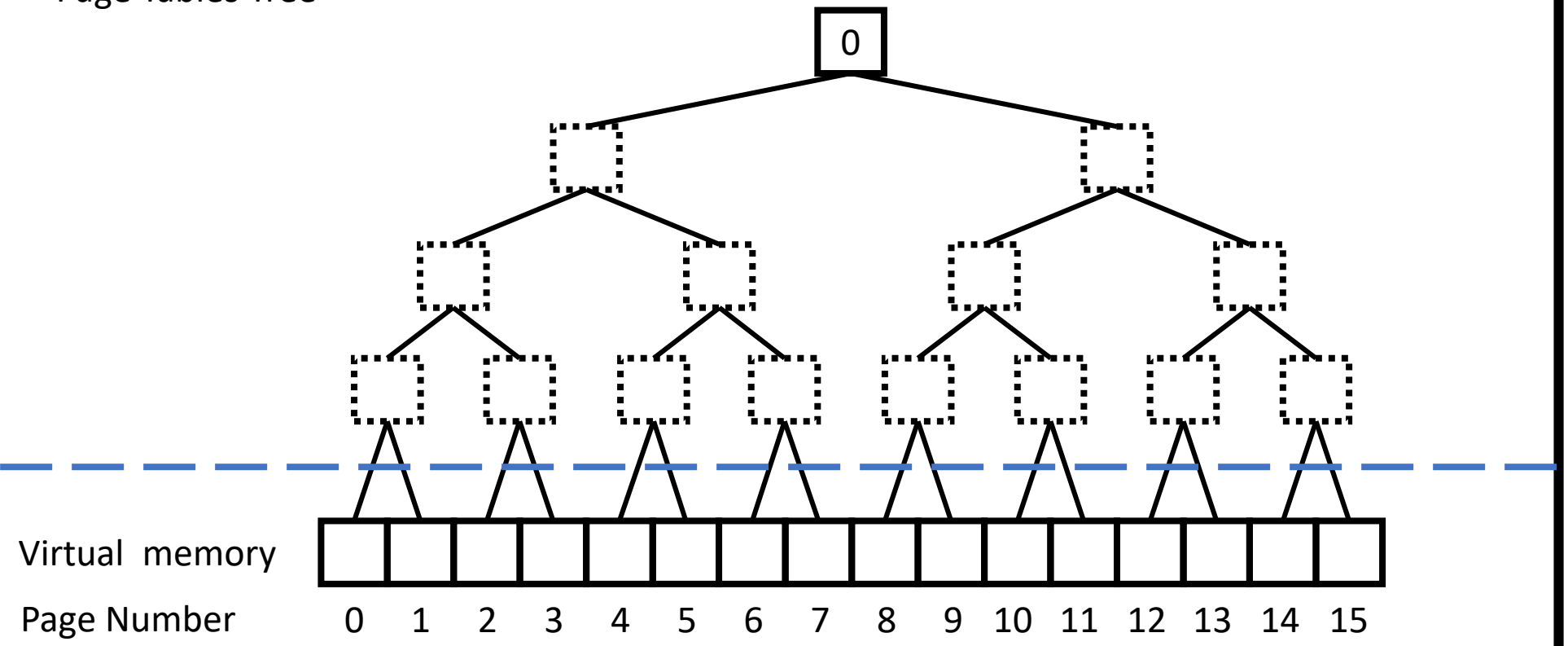
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

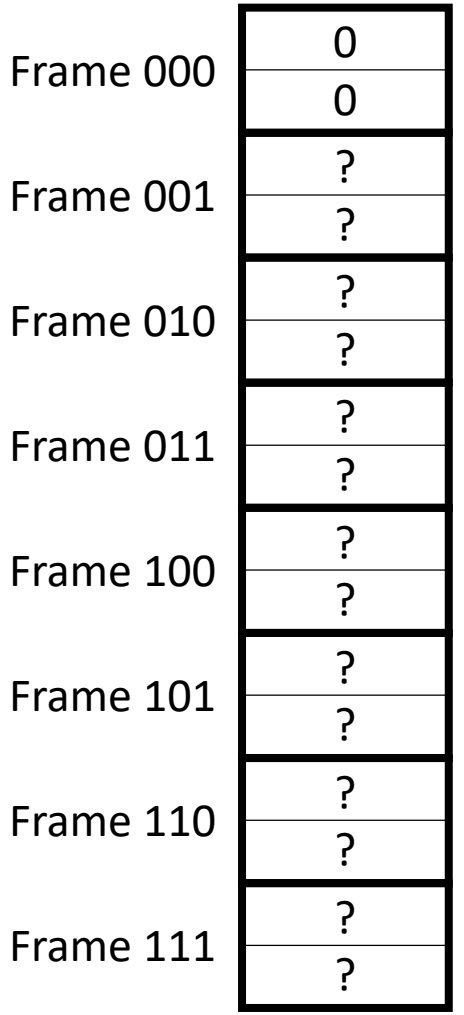
Table doesn't exist

After initialization, the physical memory only contains the root table. It will always be in frame 0.
Both its rows are now 0, as no other table exists yet.

Page Tables Tree



Physical memory

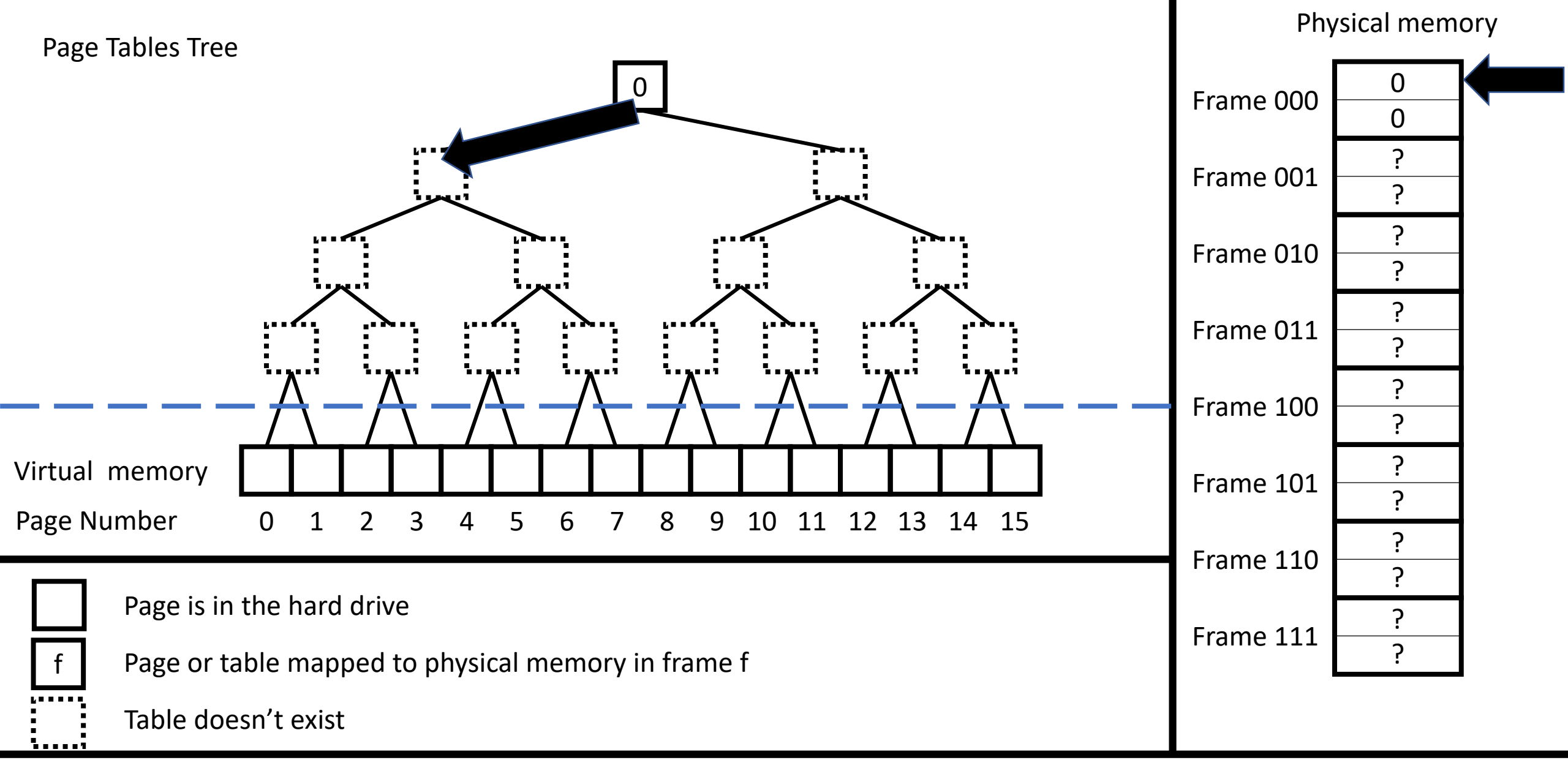


Page is in the hard drive

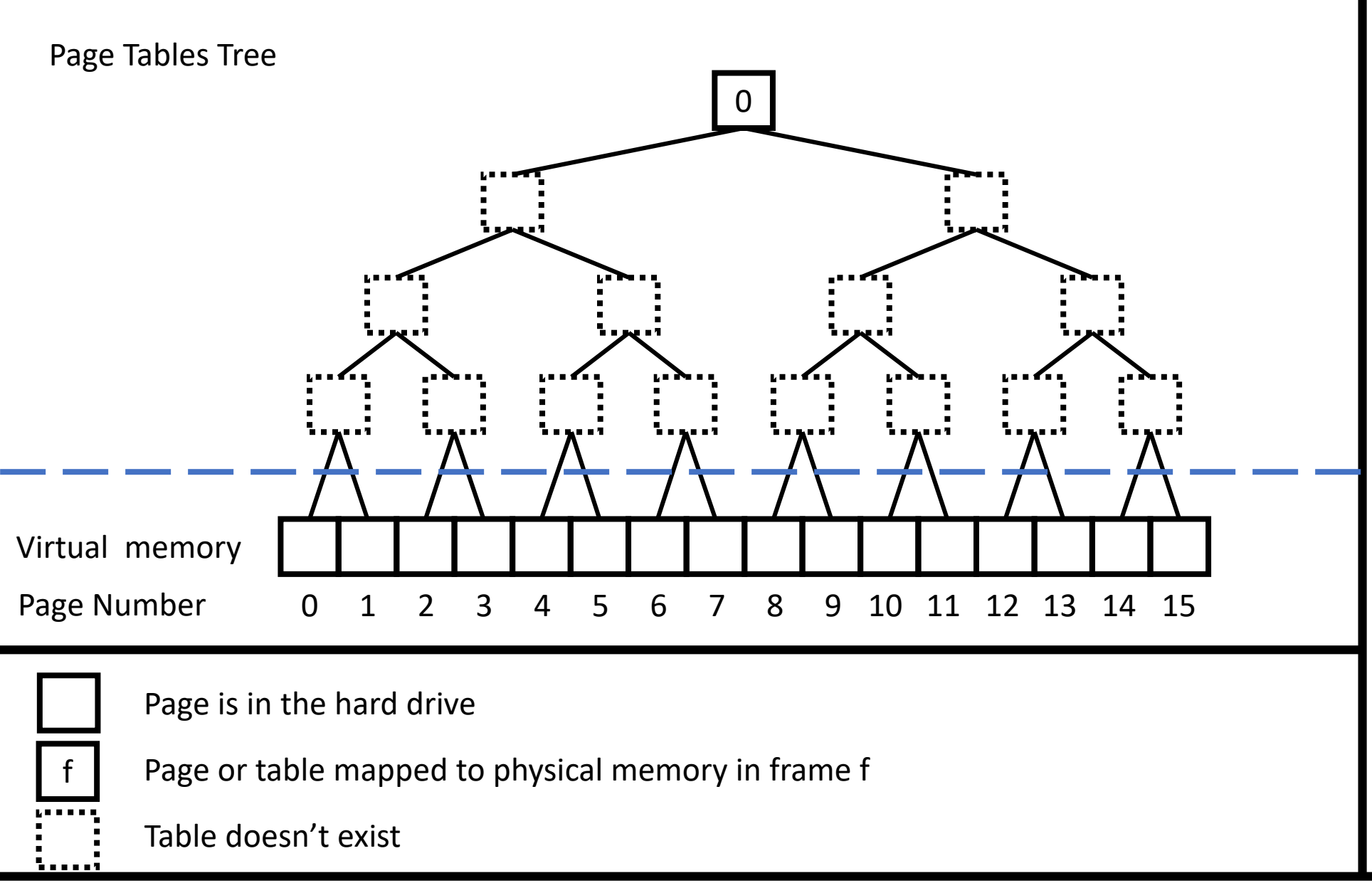
f — Page or table mapped to physical memory in frame f

Table doesn't exist

Let's now run VMwrite(13, 3).
13 is 01101, offset width is 1, so we want to write 3 into word 1 of page 6.

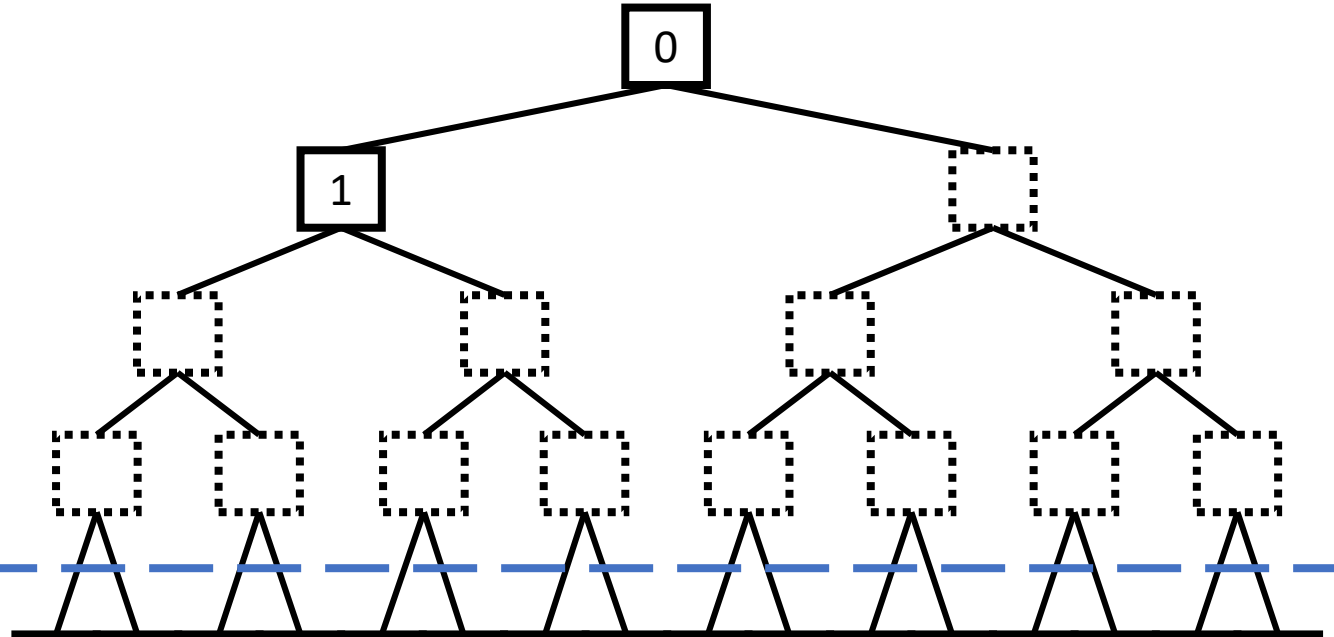


We have to first find page 6, we do it by traversing the tree according to the address 0110.
The root is in frame 0 and the first part of the address is 0, we read the RAM in address 0 + 0 = 0.
Since its content is zero, we know that the table below it is not in the RAM and we must create it.



In order to create a new table, we need a frame. We find one by traversing the entire tree in DFS.
This is done by going to the root table (always in frame 0), iterating over its rows, and recursively entering every entry that isn't 0.
This time the traversal we only saw one frame during the traversal – 0, therefore we know that frame 1 is unused.

Page Tables Tree



Physical memory

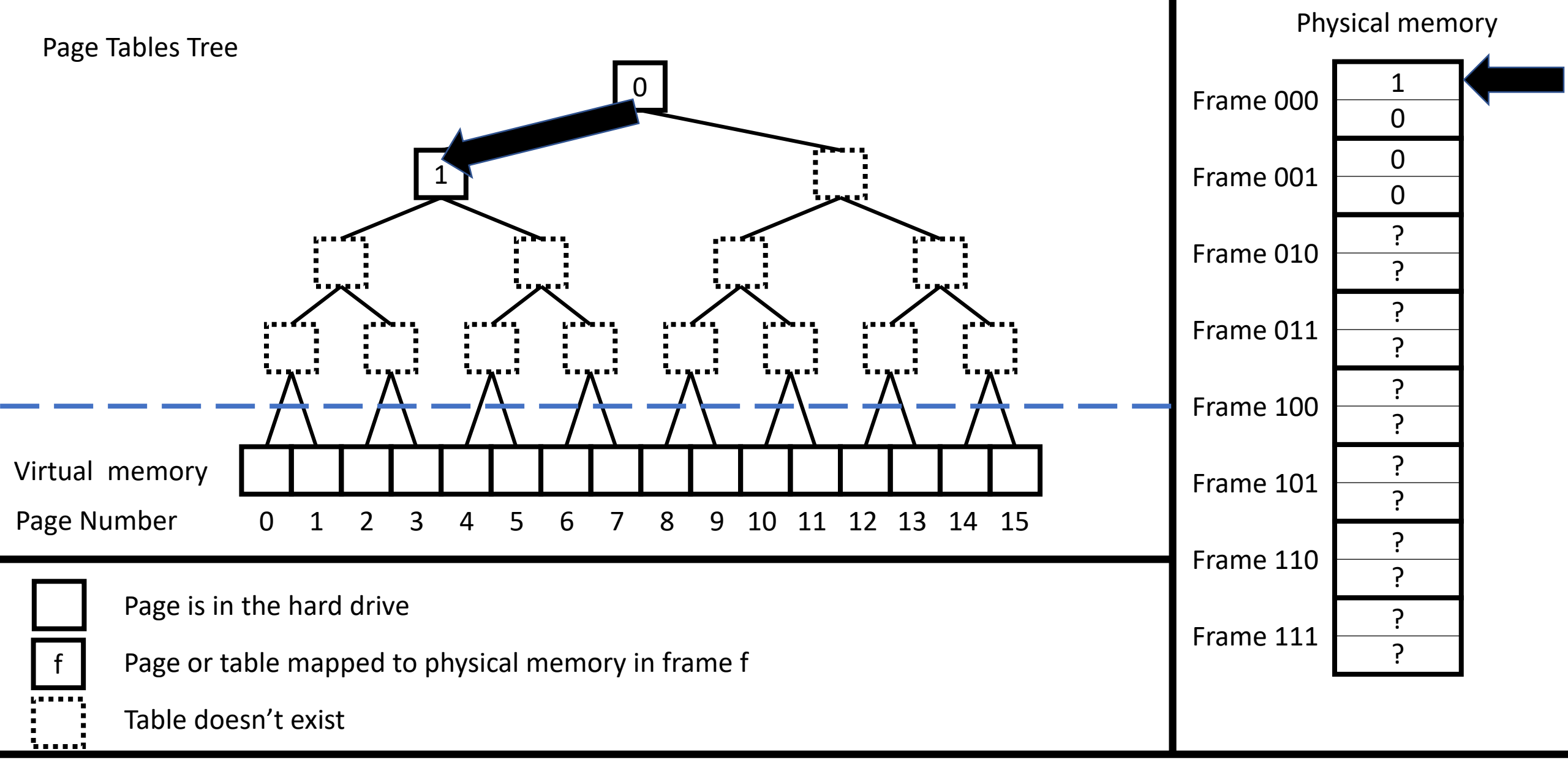
Frame 000	1
	0
Frame 001	0
	0
Frame 010	?
	?
Frame 011	?
	?
Frame 100	?
	?
Frame 101	?
	?
Frame 110	?
	?
Frame 111	?
	?

Page is in the hard drive

f — Page or table mapped to physical memory in frame f

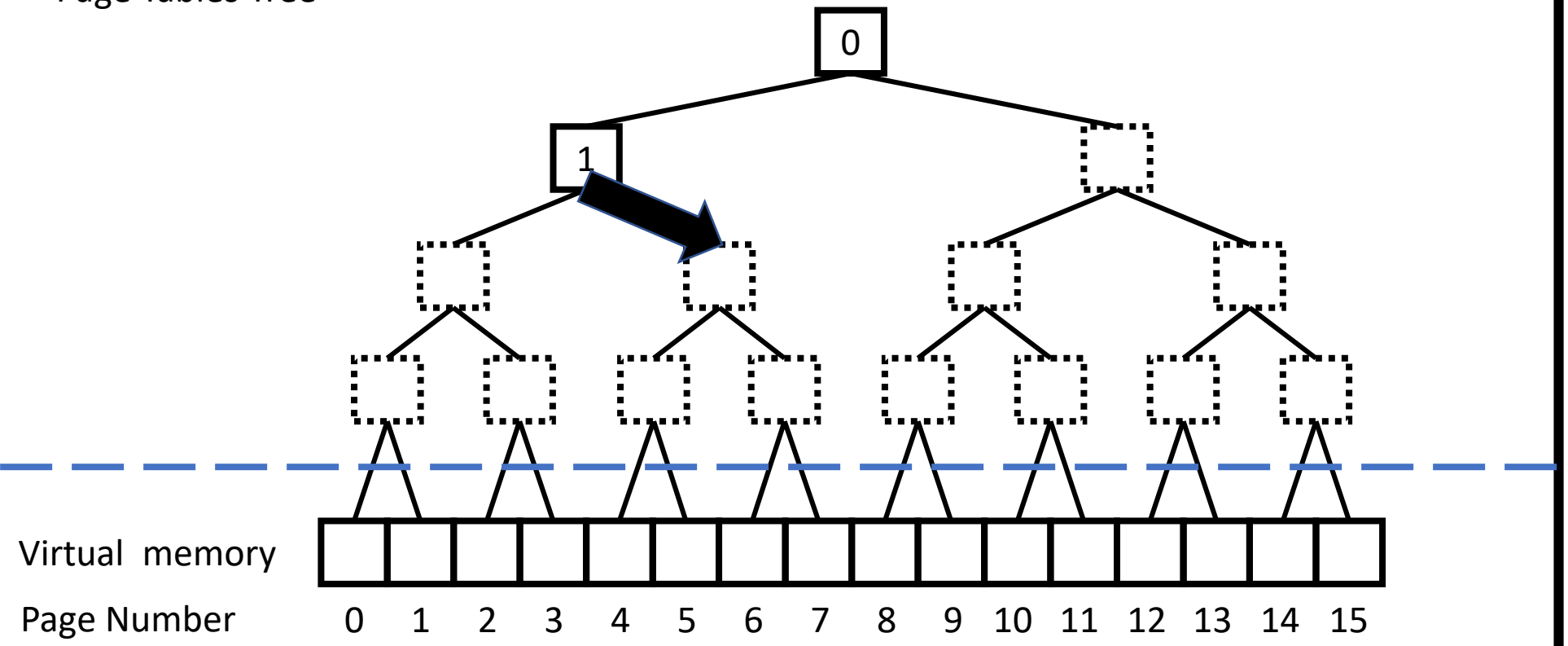
Table doesn't exist

We map our new table to frame 1 and fill it with zeros. We also link it from the root table.

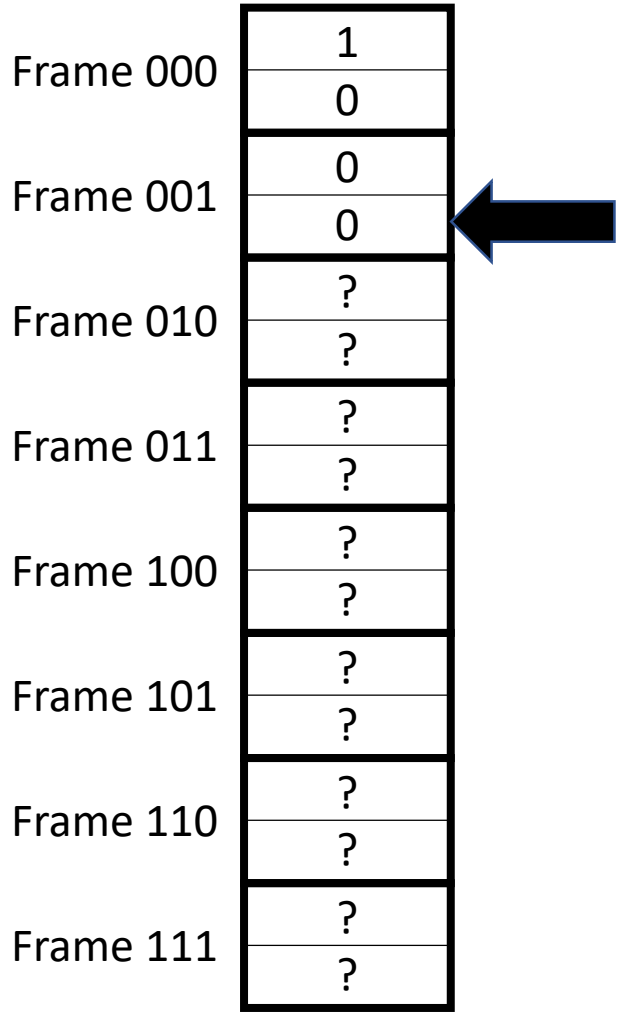


Now that the table exists, we continue down to address 0110.

Page Tables Tree



Physical memory

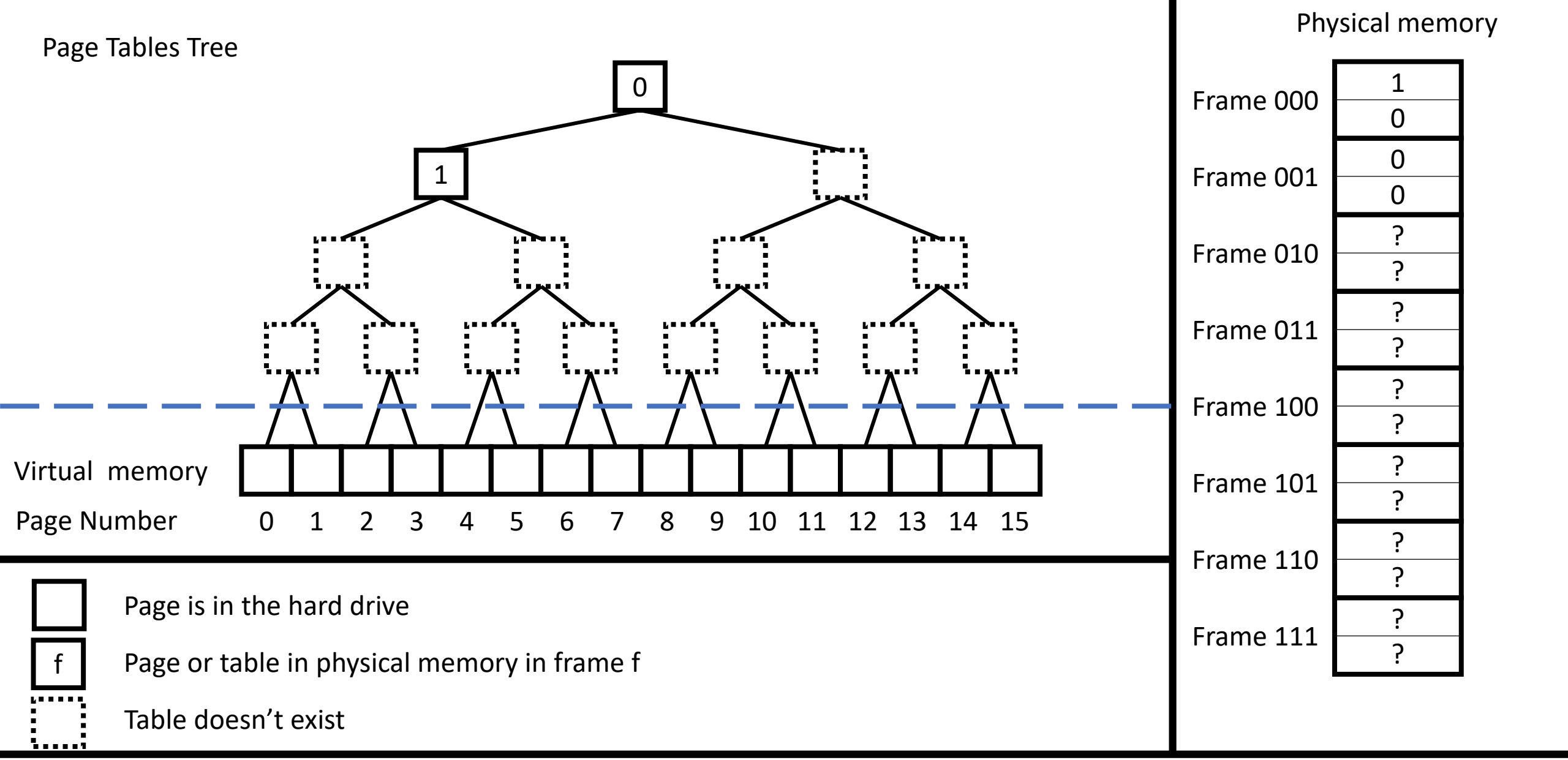


Page is in the hard drive

f — Page or table mapped to physical memory in frame f

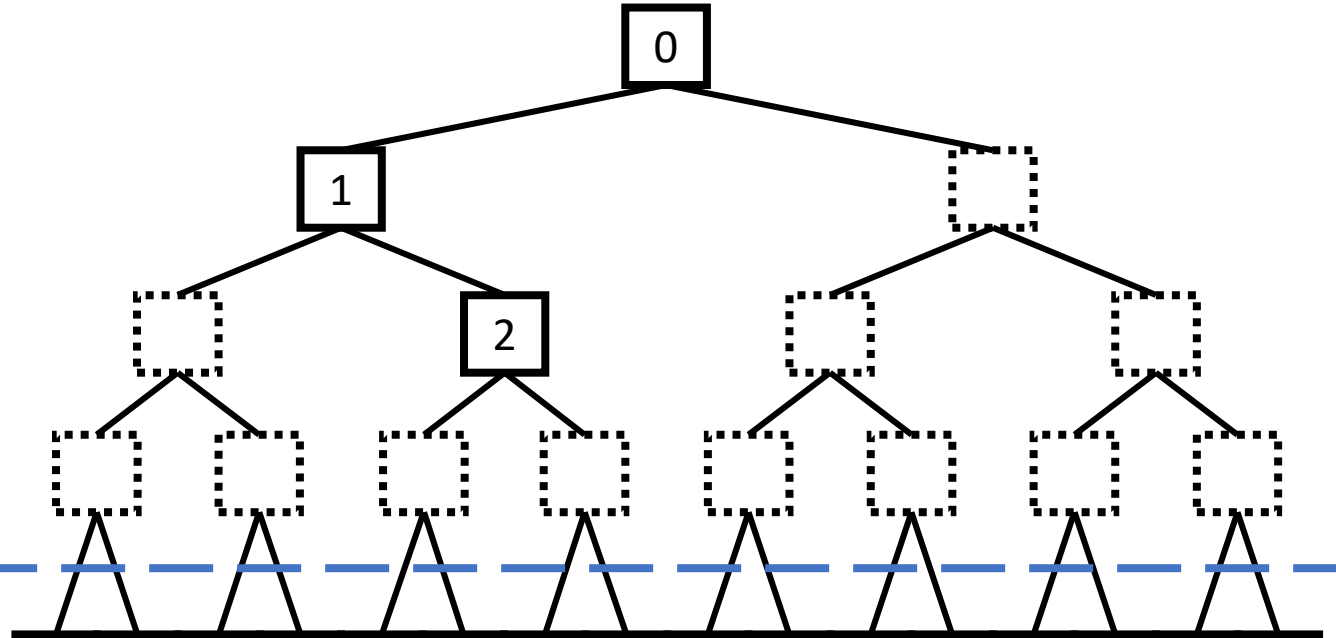
Table doesn't exist

Only to hit the same issue again.



Now when traversing the tree we will enter frame 1 from the entry in frame 0.
Again, we keep track of the maximal frame visited during the traversal and we see that this time it is 1, so frame 2 must be unused.

Page Tables Tree



Physical memory

Frame 000	1 0
Frame 001	0 2
Frame 010	0 0
Frame 011	? ?
Frame 100	? ?
Frame 101	? ?
Frame 110	? ?
Frame 111	? ?

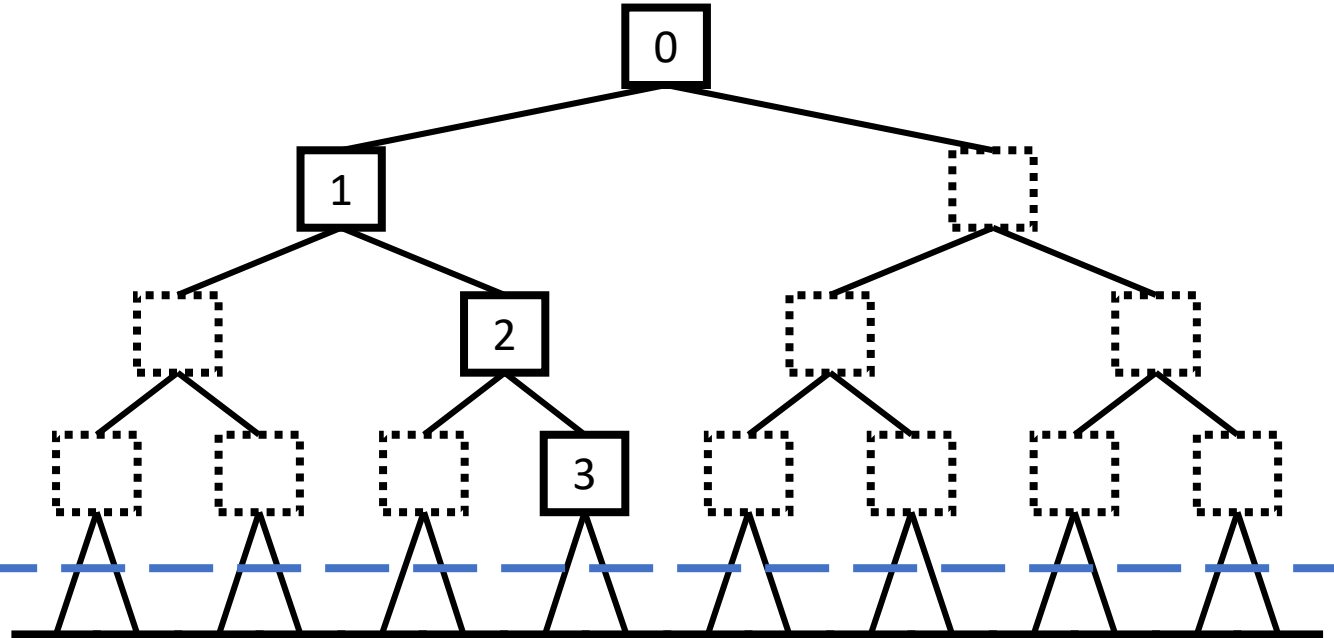
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

We map the table to frame 2 and fill it with zeros. We also link it from its parent table.

Page Tables Tree



Physical memory

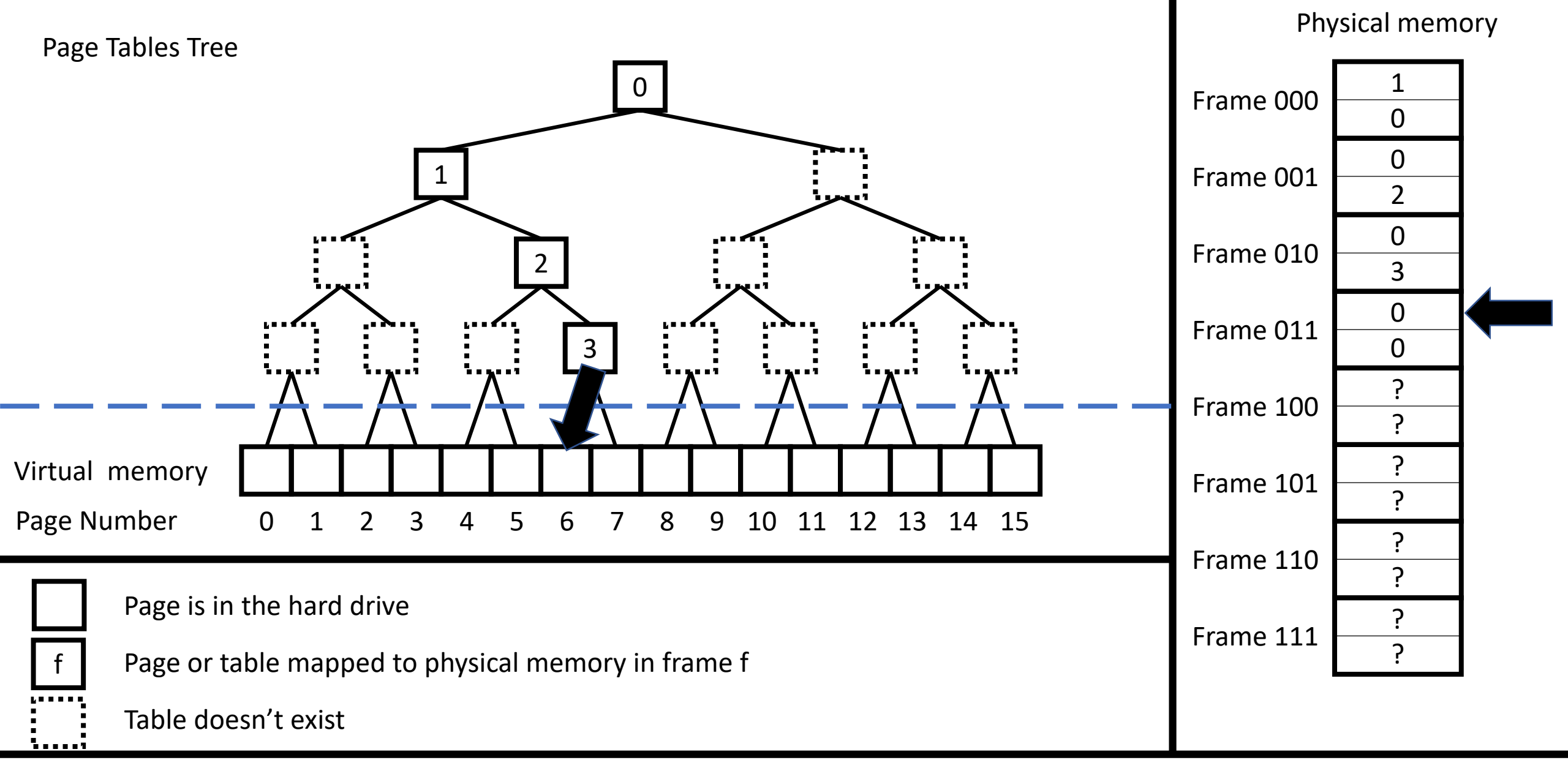
Frame 000	1	0
Frame 001	0	2
Frame 010	0	3
Frame 011	0	0
Frame 100	?	?
Frame 101	?	?
Frame 110	?	?
Frame 111	?	?

Page is in the hard drive

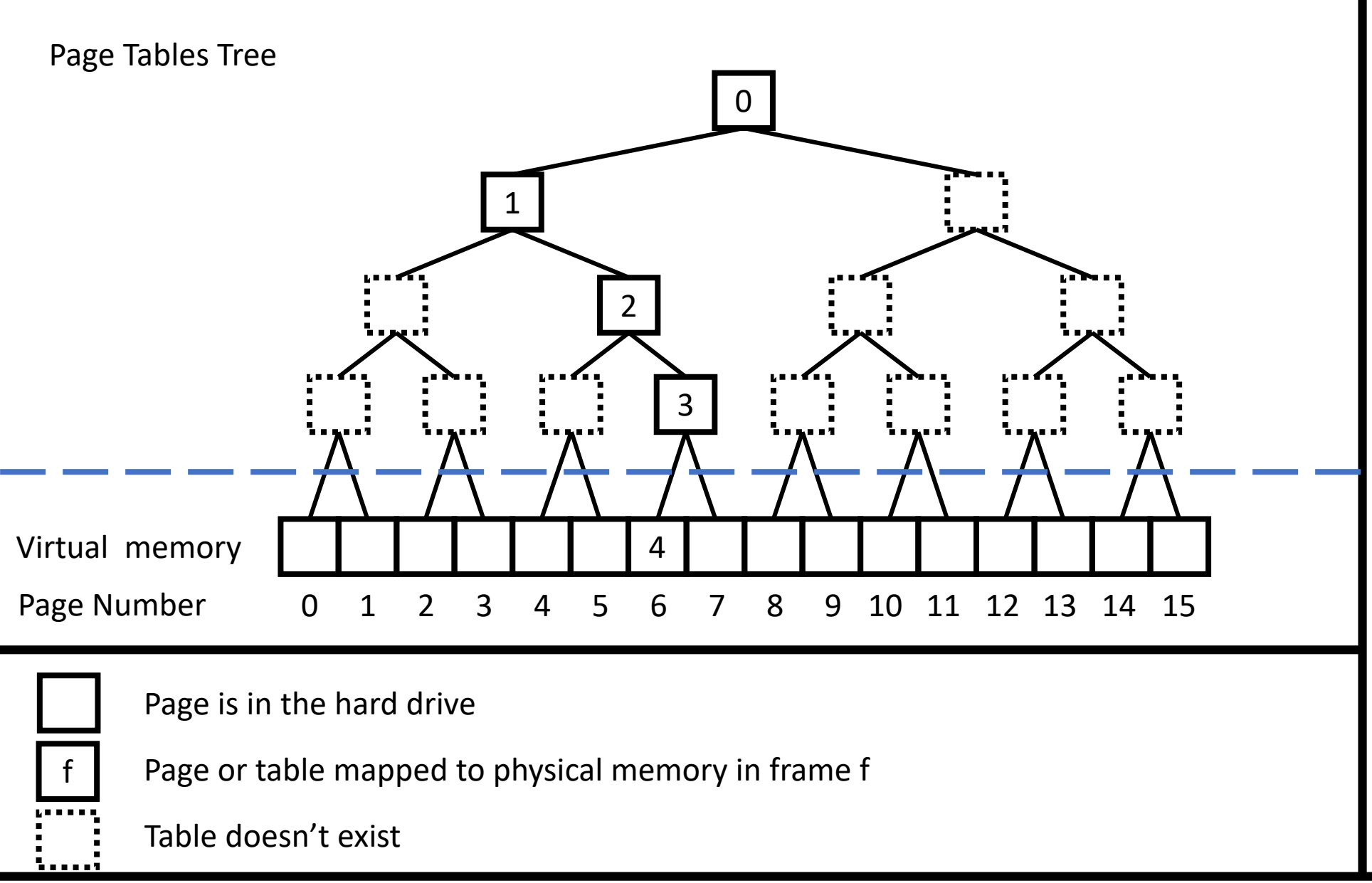
f — Page or table mapped to physical memory in frame f

Table doesn't exist

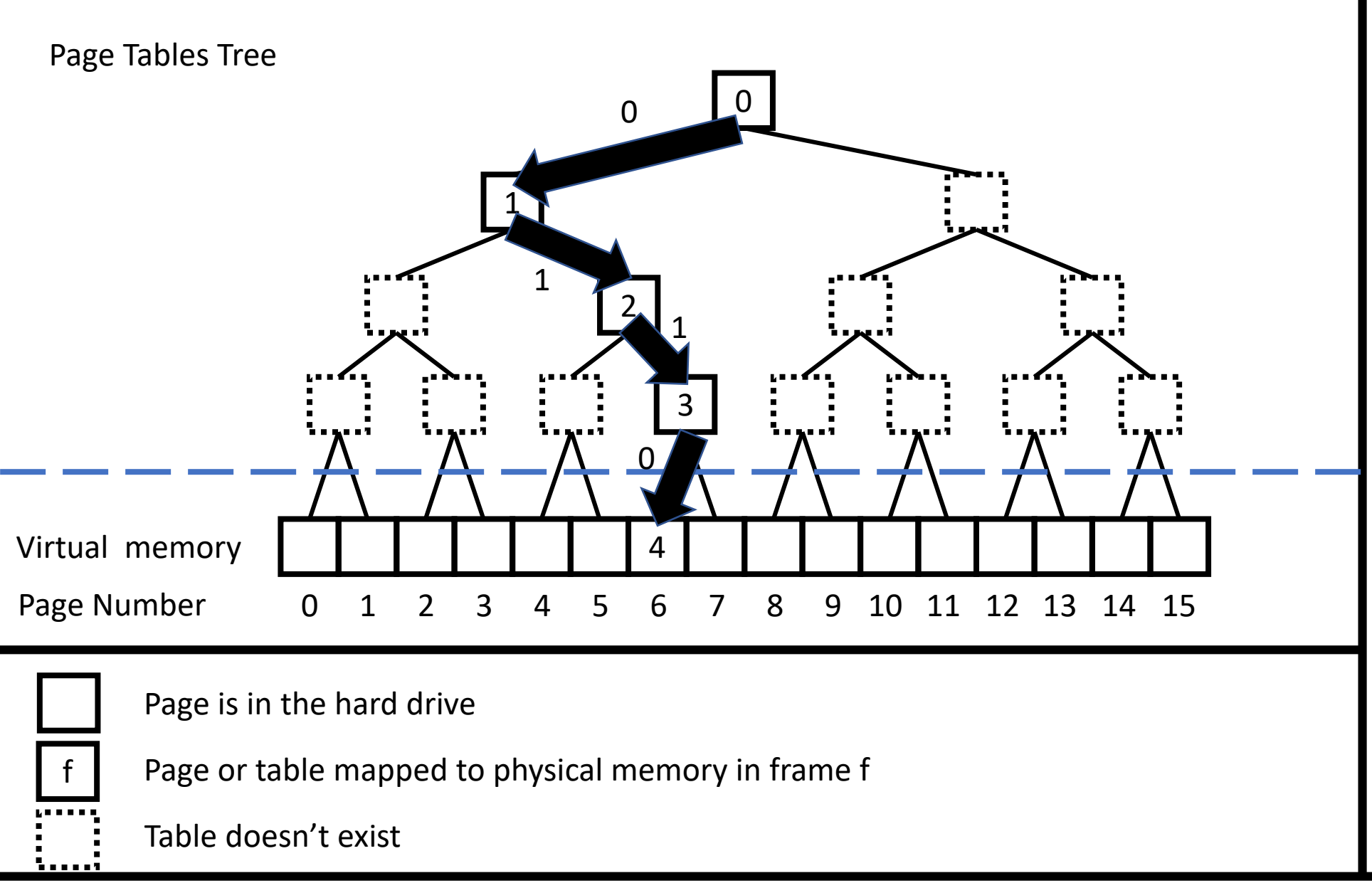
We continue in the same way and create another table for the next layer.

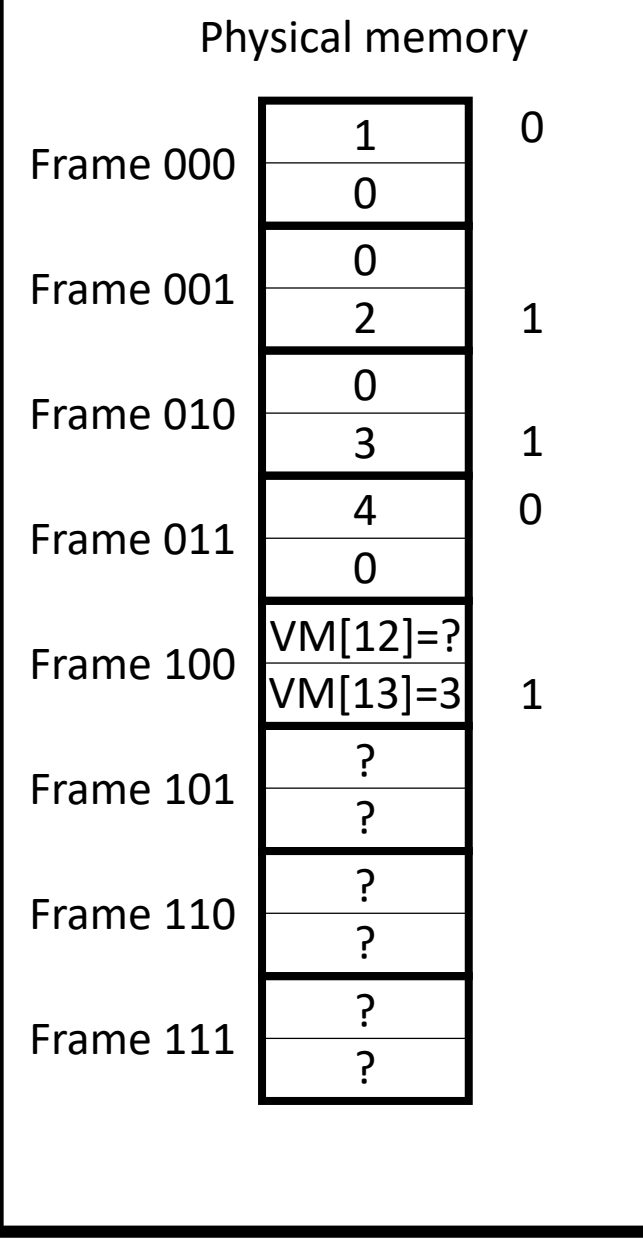


Arriving at the leaf we now want to bring page 6 into the physical memory.



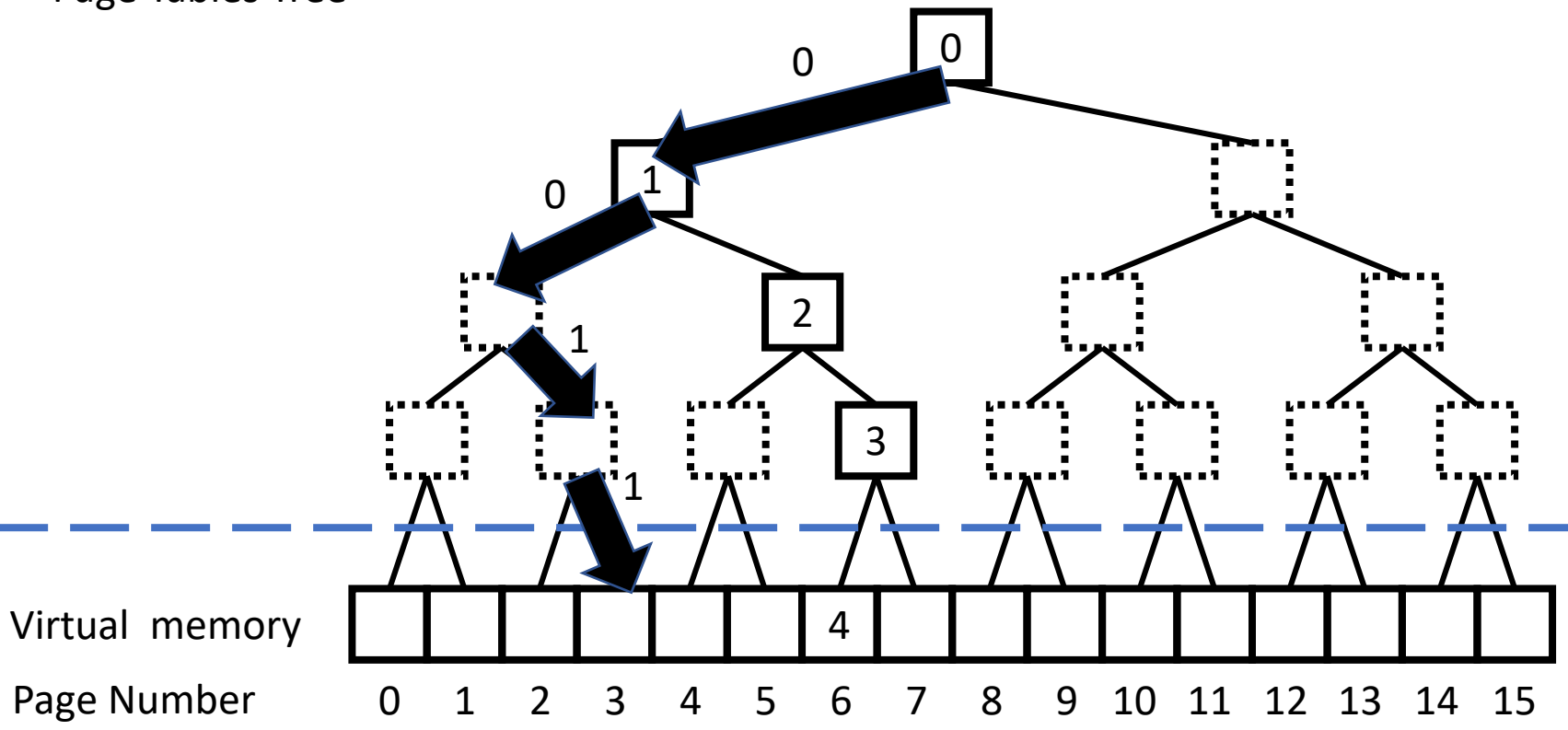
Similar to before, we find that frame 4 is unused and call PMrestore(4, 6).
We now combine our physical address 4, or 100 with the offset 1 to get 1001, or 9, and call PMwrite(9, 3)





Now if we run VMread(13, &value), going down the tree will go smoothly and we can call PMread(9, &value) to get the appropriate value.

Page Tables Tree



Physical memory

Frame 000	1
	0
Frame 001	0
	2
Frame 010	0
	3
Frame 011	4
	0
Frame 100	VM[12]=?
	VM[13]=3
Frame 101	?
	?
Frame 110	?
	?
Frame 111	?
	?

Page is in the hard drive

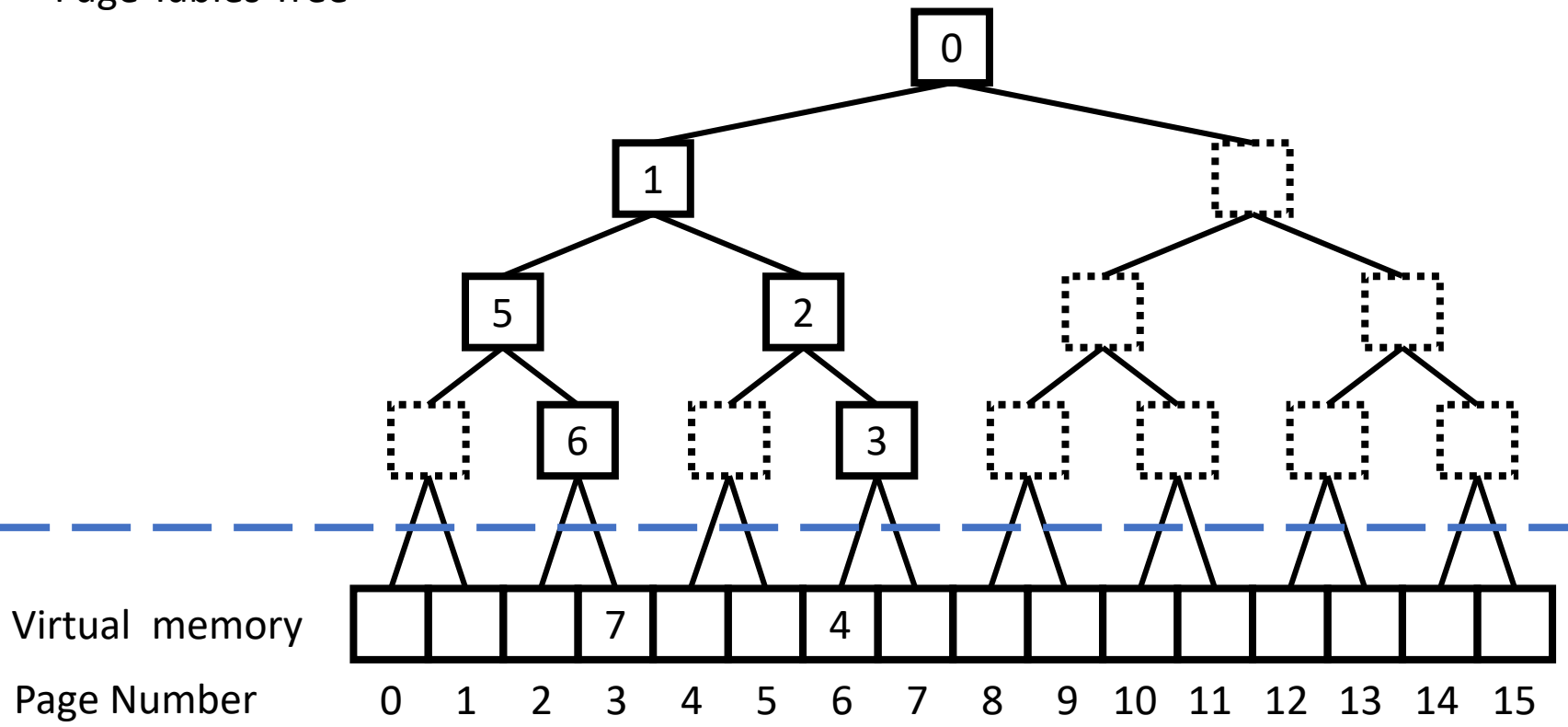
f

Page or table mapped to physical memory in frame f

Table doesn't exist

But if we run VMread(6, &value), we're looking for address 00110, and we have to do the same process we did before to find frames for the missing tables and the missing page.

Page Tables Tree



Physical memory

Frame 000	1
	0
Frame 001	5
	2
Frame 010	0
	3
Frame 011	4
	0
Frame 100	VM[12]=?
	VM[13]=3
Frame 101	0
	6
Frame 110	0
	7
Frame 111	VM[6]=?
	VM[7]=?

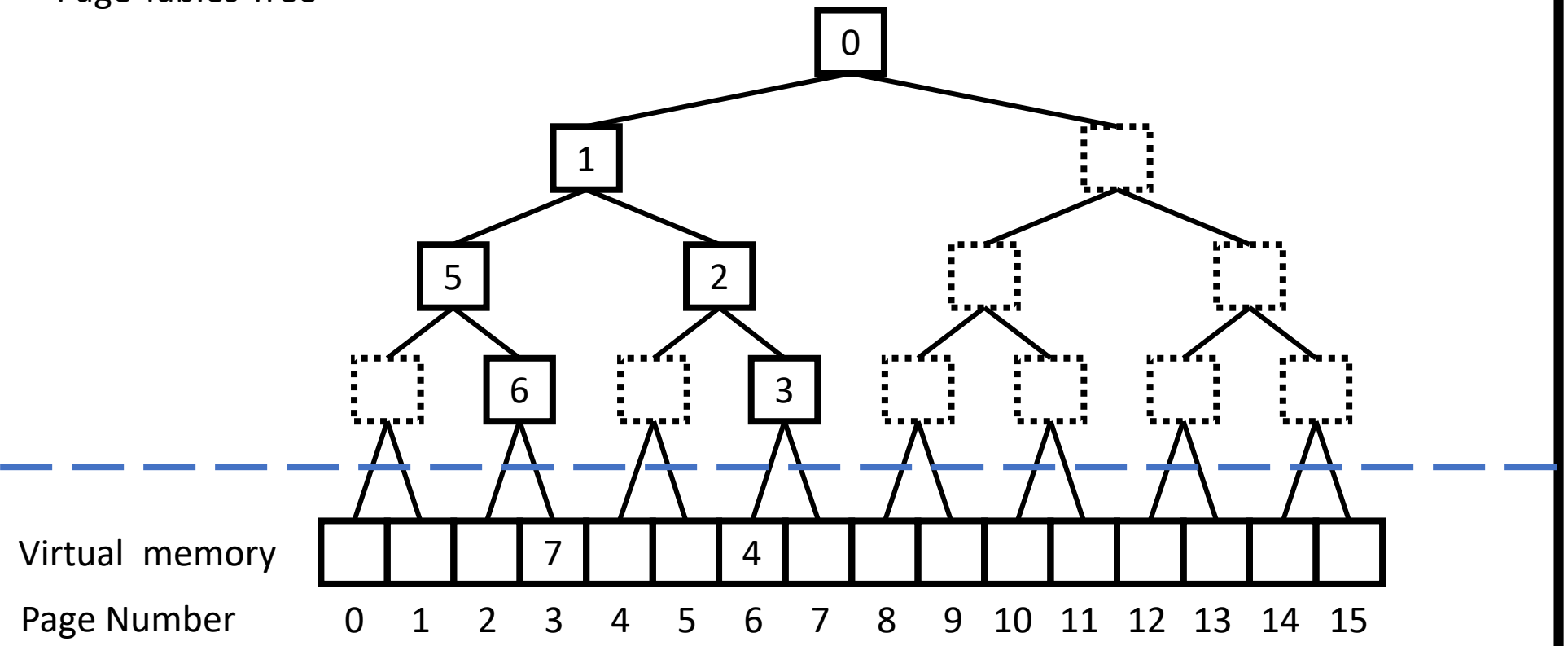
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

We end up with the situation above. Where our entire memory is full.

Page Tables Tree



Physical memory

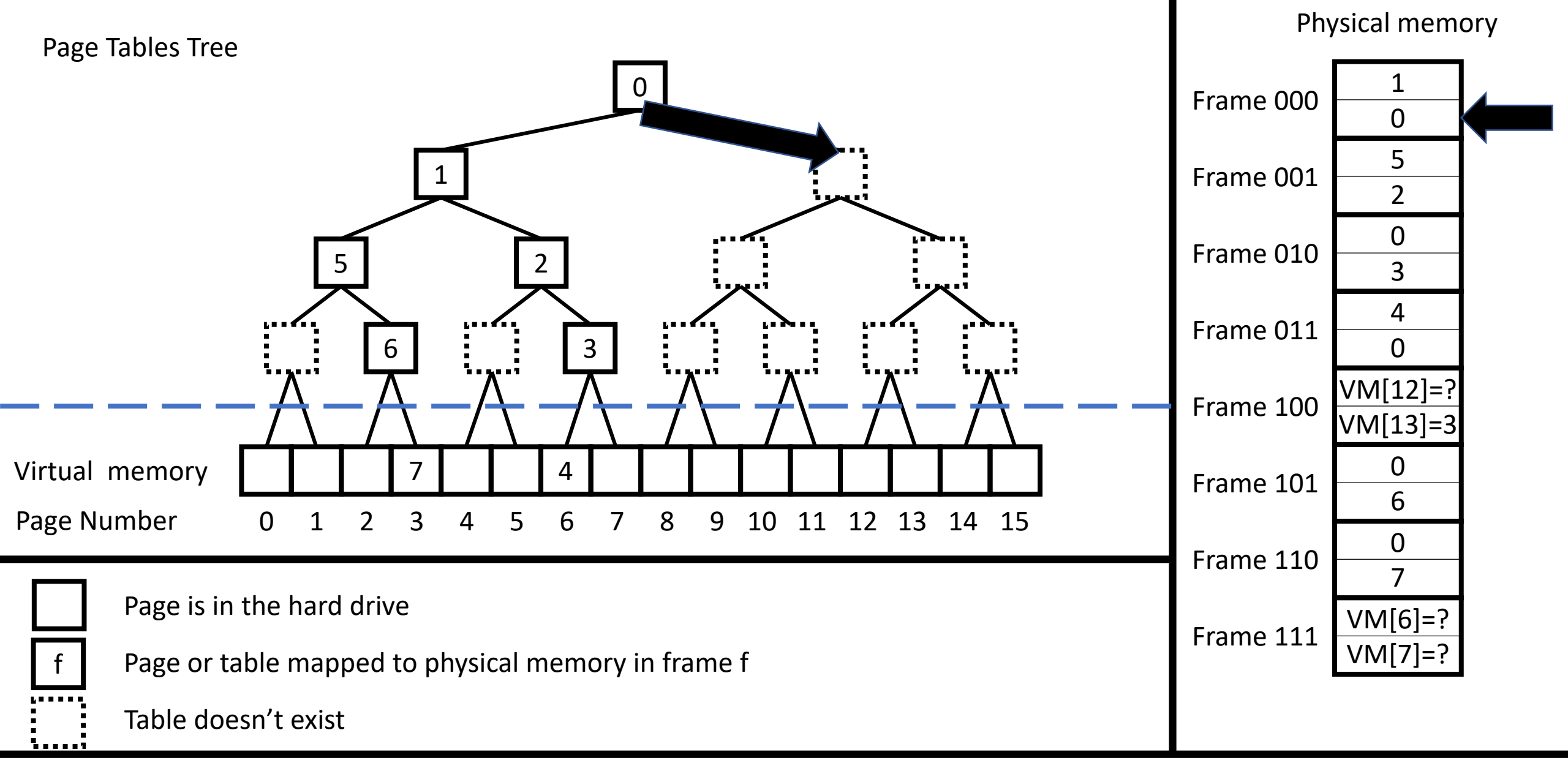
Frame 000	1
	0
Frame 001	5
	2
Frame 010	0
	3
Frame 011	4
	0
Frame 100	VM[12]=?
	VM[13]=3
Frame 101	0
	6
Frame 110	0
	7
Frame 111	VM[6]=?
	VM[7]=?

Page is in the hard drive

f — Page or table mapped to physical memory in frame f

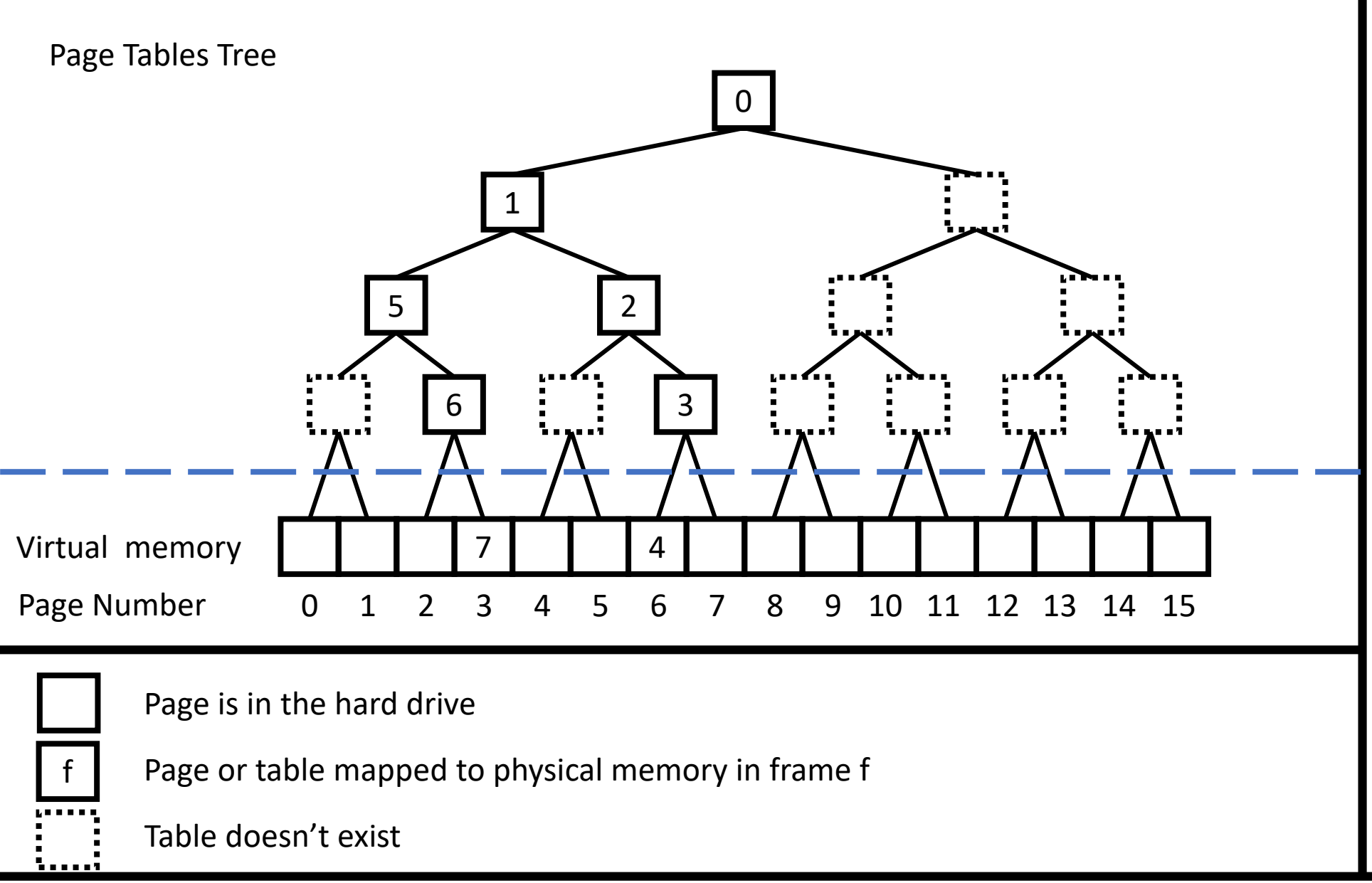
Table doesn't exist

Let's now call VMread(31, &value).

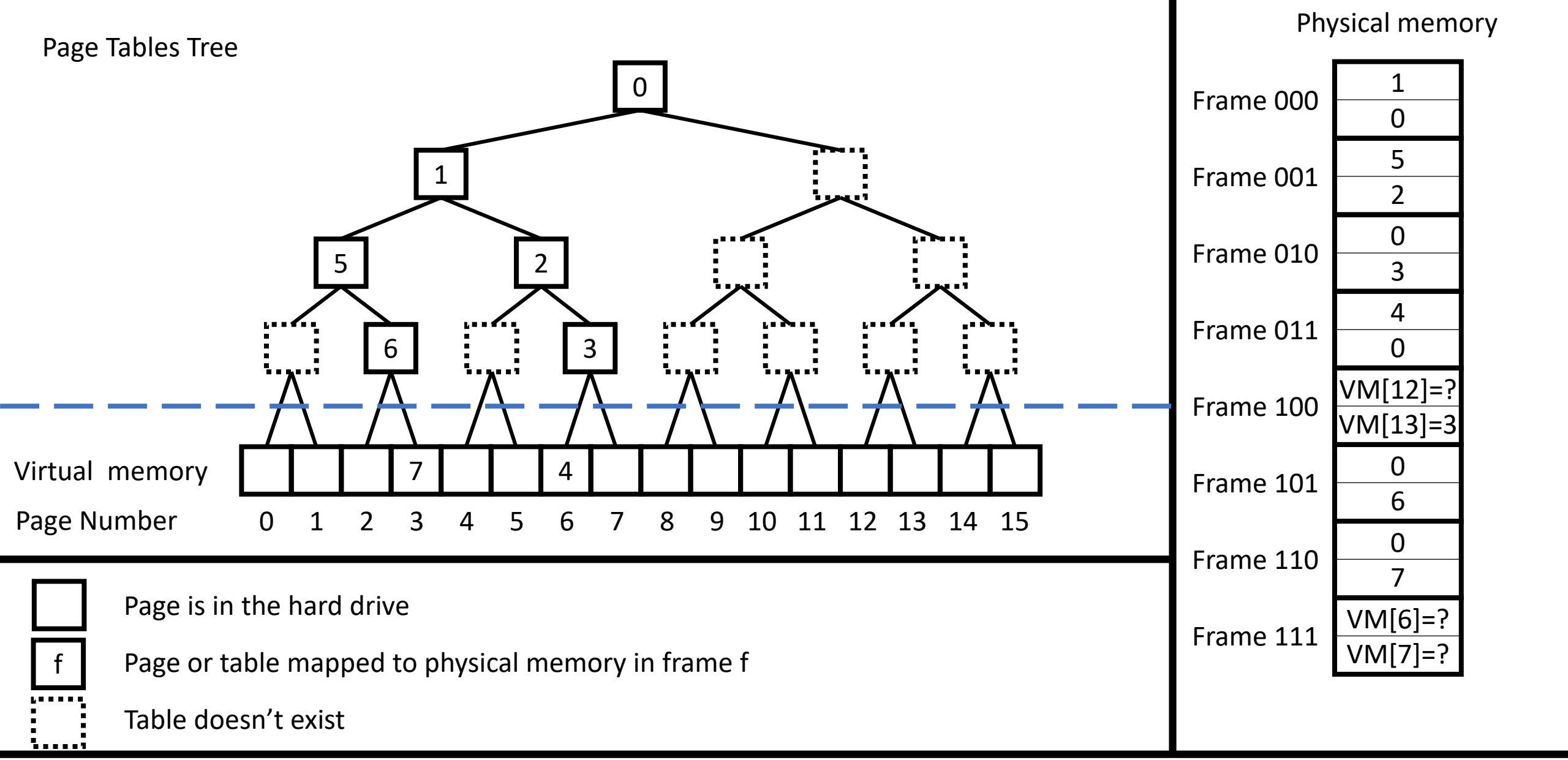


The address is 11111, so the page number is 1111 and the offset is 1

We immediately hit a missing table directly from the root, so we will have to find a frame to create it in.

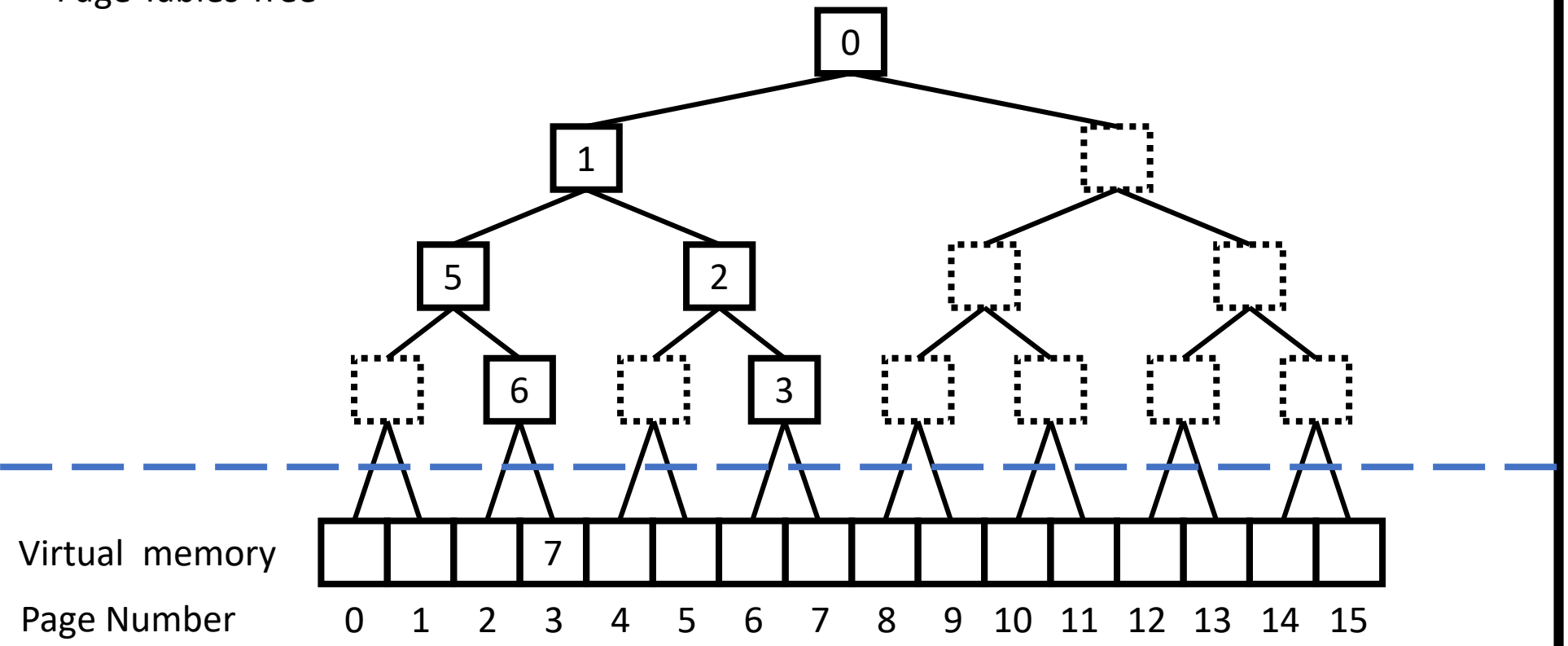


This time, traversing through the tree we see that frame 7 is in use, which means there are no unused frames.
We also haven't seen any tables along the way with only zeros, so we have to evict a page.
Also during the traversal we checked for each page we've seen what its cyclic distance from 15 (the page we're trying to access).



So since min{15-6, 16-(15-6)} = 7 > 4 = min{15-3, 16-(15-3)}, after the traversal we know that the page most distant from 15 currently in memory is page 6 which is in frame 4.

Page Tables Tree



Physical memory

Frame 000	1
	0
Frame 001	5
	2
Frame 010	0
	3
Frame 011	0
	0
Frame 100	?
	3
Frame 101	0
	6
Frame 110	0
	7
Frame 111	VM[6]=?
	VM[7]=?

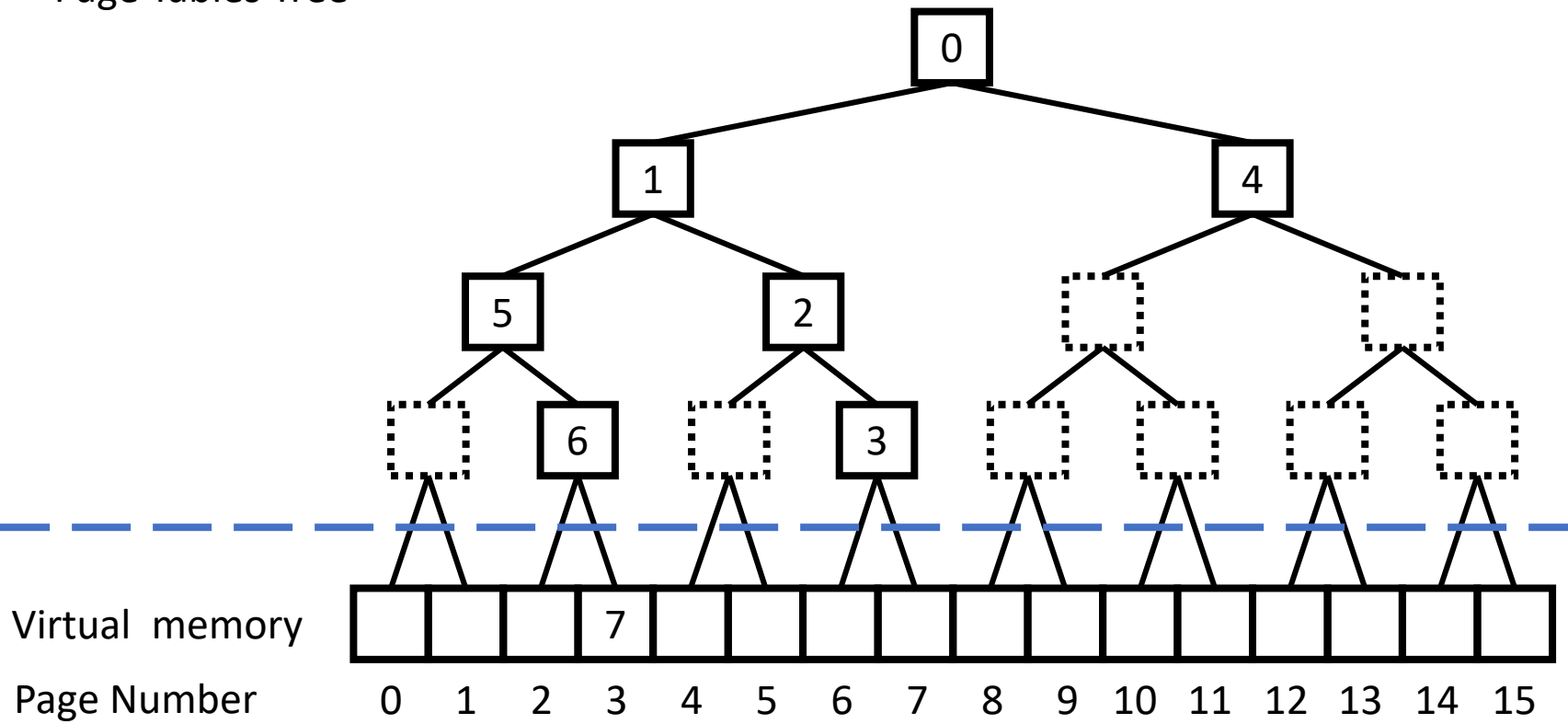
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

We call PMevict(4,6) and remove the link to it from its parent table which is in frame 3.

Page Tables Tree



Physical memory

Frame 000	1 4
Frame 001	5 2
Frame 010	0 3
Frame 011	0 0
Frame 100	0 0
Frame 101	0 6
Frame 110	0 7
Frame 111	VM[6]=? VM[7]=?

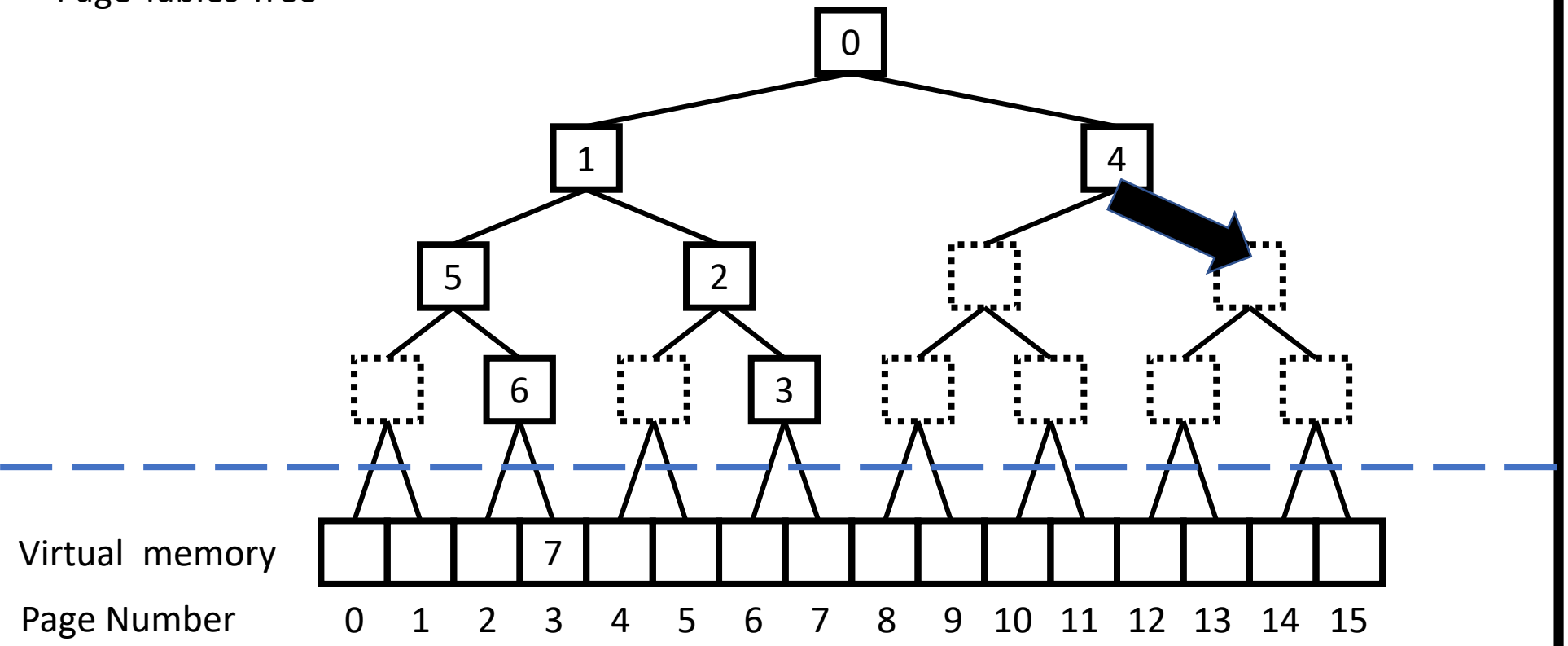
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

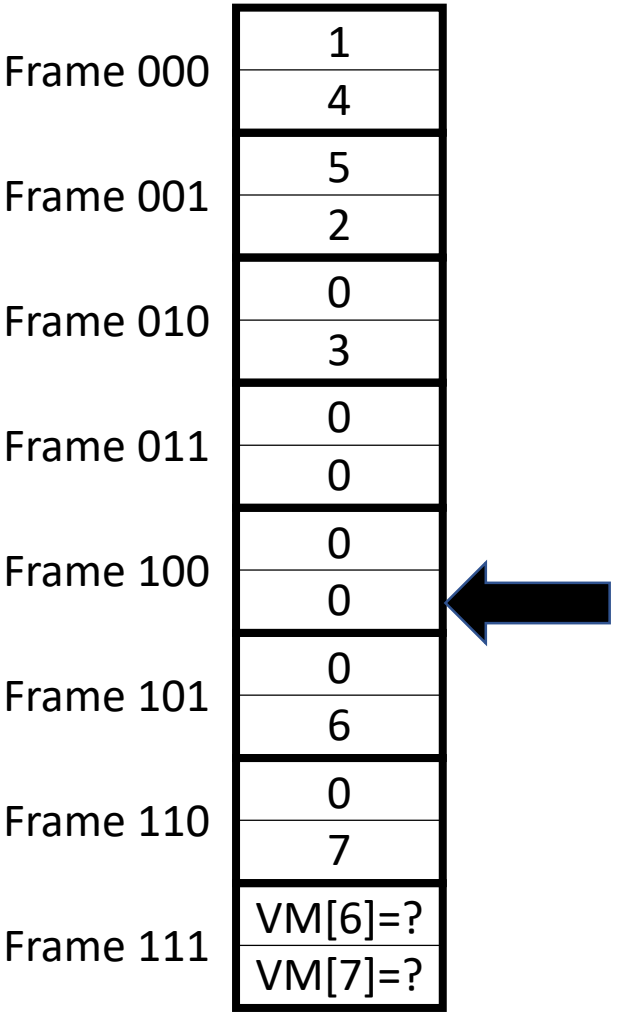
Table doesn't exist

We create our new table in frame 4, fill it with zeros and link it from its parent – the root.

Page Tables Tree



Physical memory



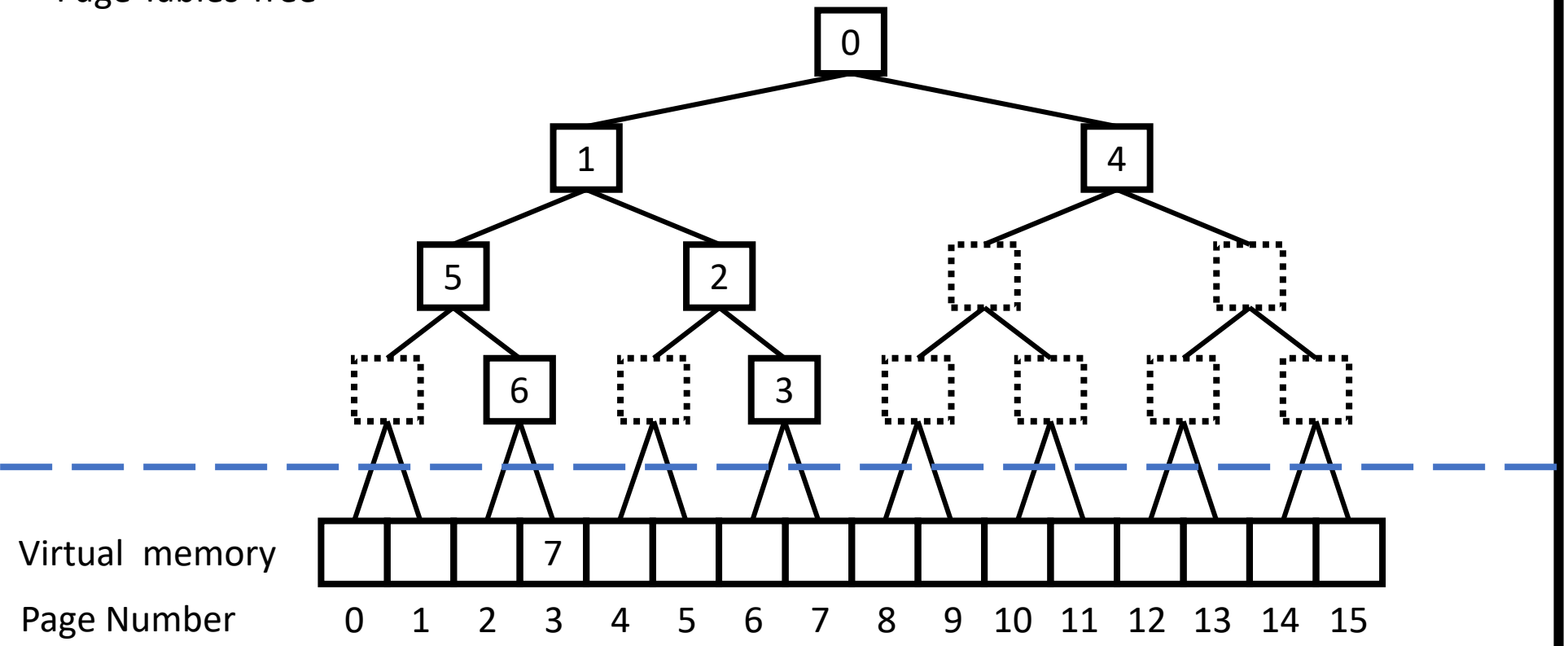
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

In the next level we hit another non-existing table, so again we must find a frame.

Page Tables Tree



Physical memory

Frame 000	1
	4
Frame 001	5
	2
Frame 010	0
	3
Frame 011	0
	0
Frame 100	0
	0
Frame 101	0
	6
Frame 110	0
	7
Frame 111	VM[6]=?
	VM[7]=?

Page is in the hard drive

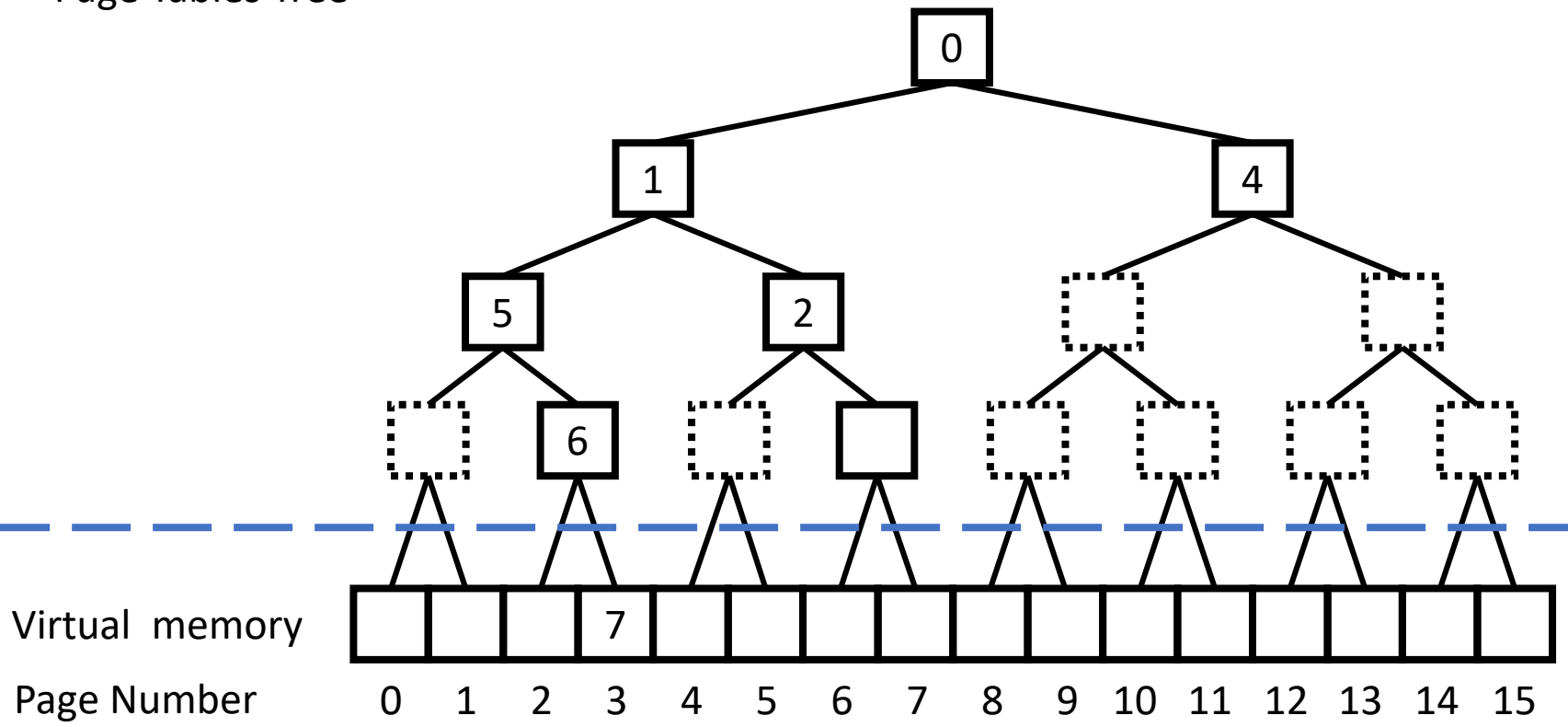
f — Page or table mapped to physical memory in frame f

Table doesn't exist

This time when traversing the tree we see that frame 3 only contains zeros, which means there are no pages below it and it can be safely removed.

Note that while frame 4 also only contains zeros, we don't want to remove it as we need it for page 15 (remember?).

Page Tables Tree



Physical memory

Frame 000	1
	4
Frame 001	5
	2
Frame 010	0
	0
Frame 011	0
	0
Frame 100	0
	0
Frame 101	0
	6
Frame 110	0
	7
Frame 111	VM[6]=?
	VM[7]=?

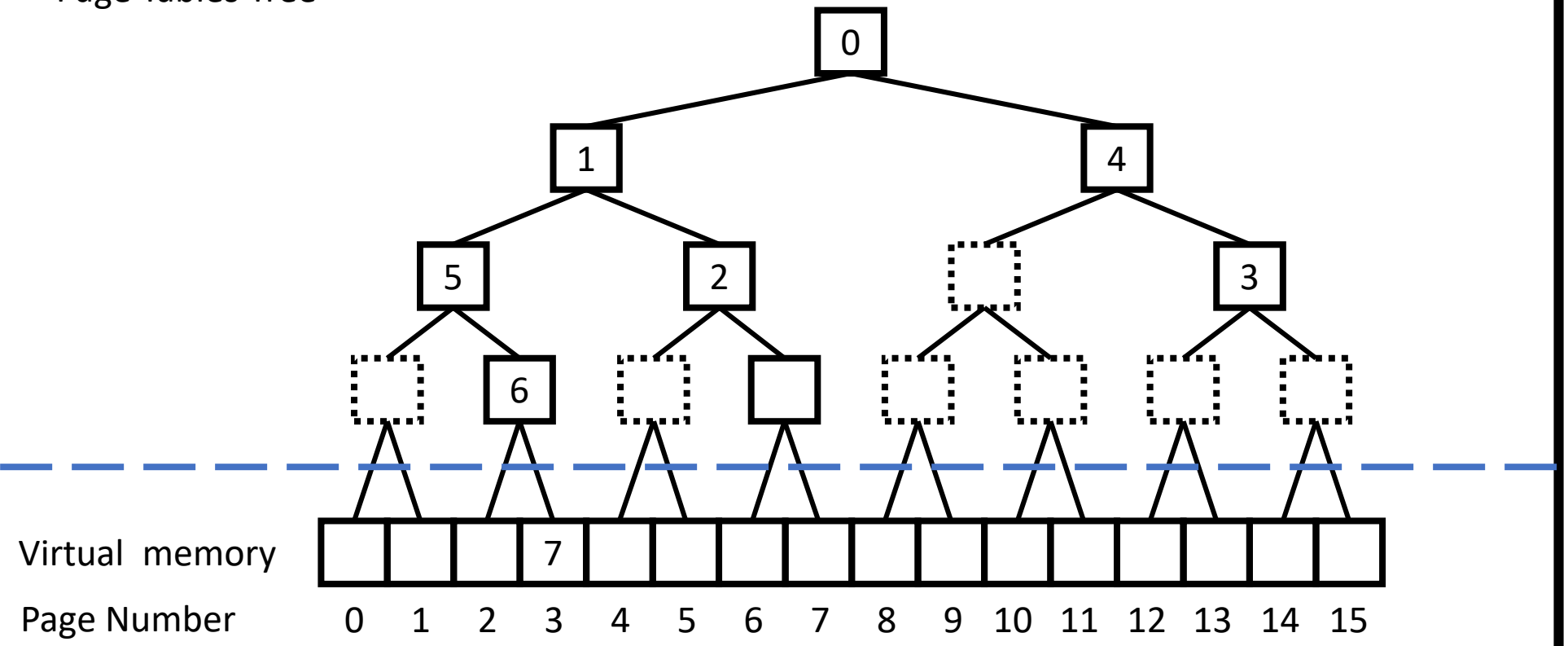
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

So we unlink frame 3 from the table above it which is in frame 2.

Page Tables Tree



Physical memory

Frame 000	1
	4
Frame 001	5
	2
Frame 010	0
	0
Frame 011	0
	0
Frame 100	0
	3
Frame 101	0
	6
Frame 110	0
	7
Frame 111	VM[6]=?
	VM[7]=?

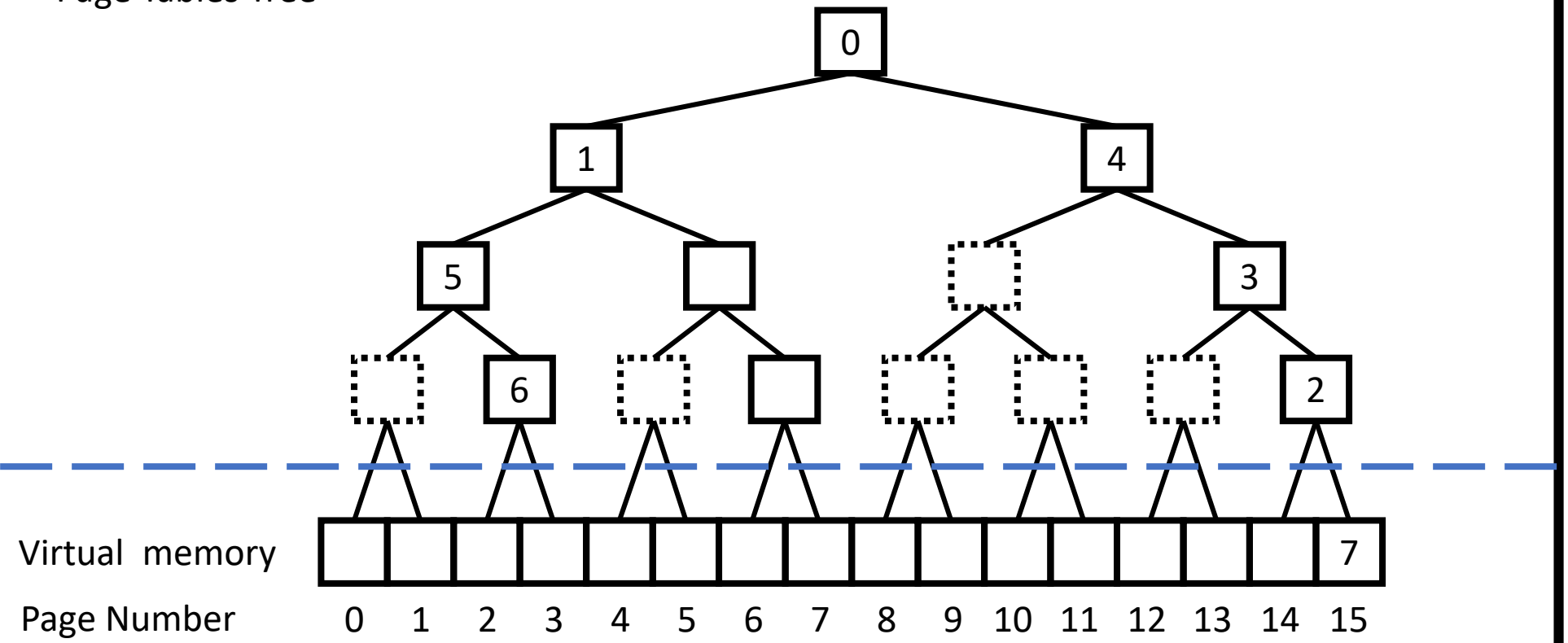
Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

And we create our new table.

Page Tables Tree



Physical memory

Frame 000	1
	4
Frame 001	5
	0
Frame 010	0
	0
Frame 011	0
	2
Frame 100	0
	3
Frame 101	0
	6
Frame 110	0
	0
Frame 111	VM[30]=?
	VM[31]=?

Page is in the hard drive

f — Page or table mapped to physical memory in frame f

Table doesn't exist

We continue this process until finally page 15 is brought in.
Make sure you understand the steps leading to the state above.