Tutorial 9

CS2106: Introduction to Operating Systems

Question 1

TLB + Paging + Virtual Paging

Question 1

Setup

- Page Size = Frame Size = 4KB
- TLB = 2 entries (0..1)
- Page Table = 8 entries (i.e. 8 pages, 0..7)
- Physical Frame = 8 frames (0..7)
- Swap Page (Virtual page in hard disk) = 16 pages (0..15)

Below is a snapshot of process P at time T

Page No.	Frame No.
3	7
0	4

TLB

Page No.	Frame No./ Swap Page No.	Memory Resident?	Valid?
0	4	1	1
1	1	1	1
2	1	0	1
3	7	1	1
4	2	1	1
5	15	0	1
6	---	0	0
7	---	0	0

Page Table

P

Setup

- Page Size = Frame Size = 4KB
- TLB = 2 entries (0..1)
- Page Table = 8 entries (i.e. 8 pages, 0..7)
- Physical Frame = 8 frames (0..7)
- Swap Page (Virtual page in hard disk) = 16 pages (0..15)

Virtual Memory Space
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

	Page Table	
0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	Physical memory
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	Secondary Storage
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(a)

Below is the skeleton of the access algorithm for this setup. Give the missing algorithm for the OS **TLB fault** and **page fault** handlers.

Algorithm for accessing virtual address VA:

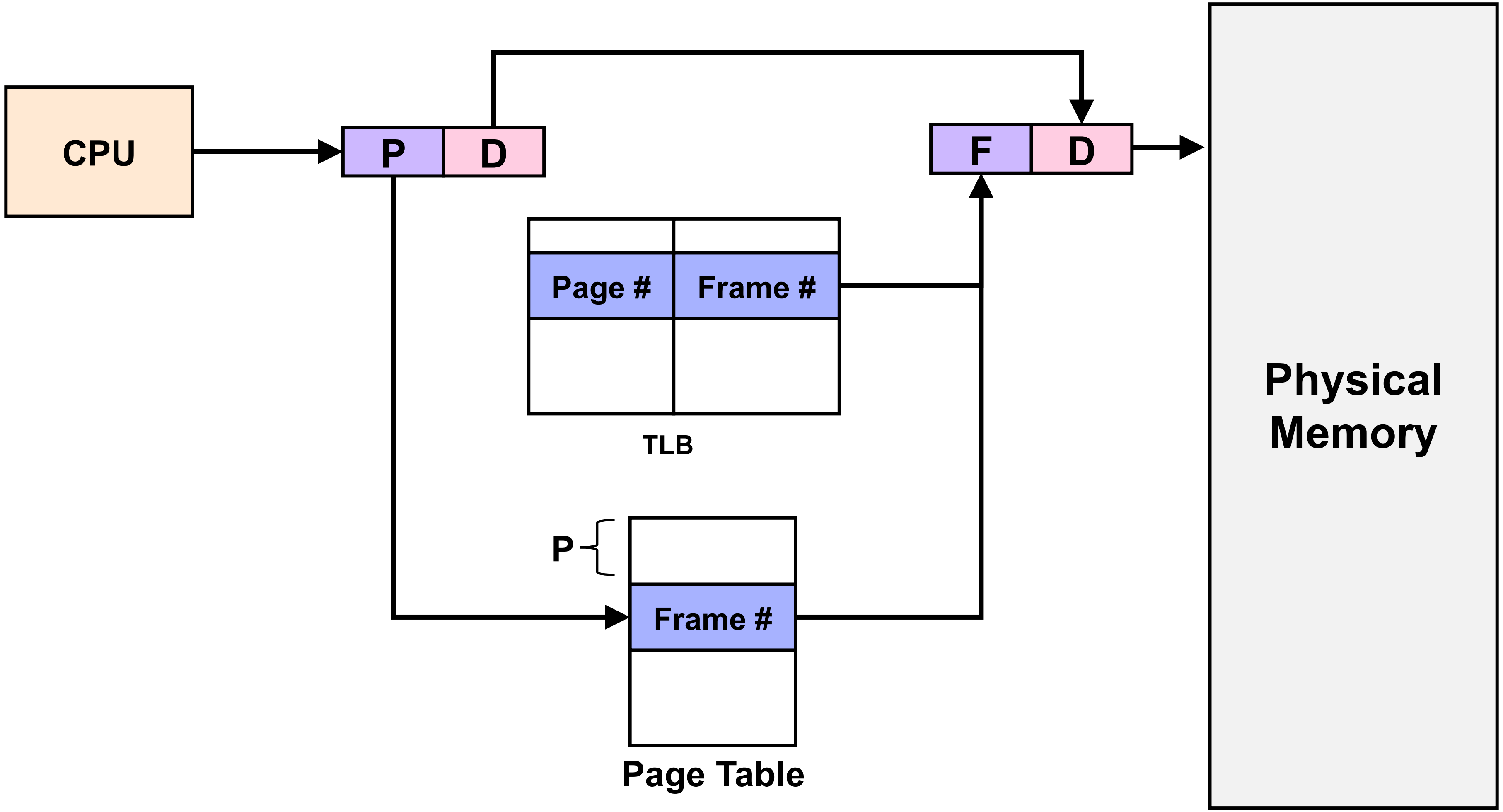
1. **[HW] VA is decomposed into <Page#, Offset>**
2. **[HW] Search TLB for <Page#>:**
    a. **If TLB miss: Trap to OS { TLB Fault }**
3. **[HW] Is <Page#> memory resident?**
    a. **Non-memory resident: Trap to OS { Page Fault }**
4. **[HW] Use <Frame#><Offset> to access physical memory.**

Question 1(a): TLB Fault

Algorithm for TLB Fault Handler, given <Page#>

1. [OS] Access full page table of the process (e.g. in the PCB of the process), <Page#> is the index
2. [OS] Check whether valid bit is set, if not segmentation fault. ←
3. [OS] Load the relevant PTE into TLB.
4. [OS] Return from trap.

Question 1(a): TLB Fault

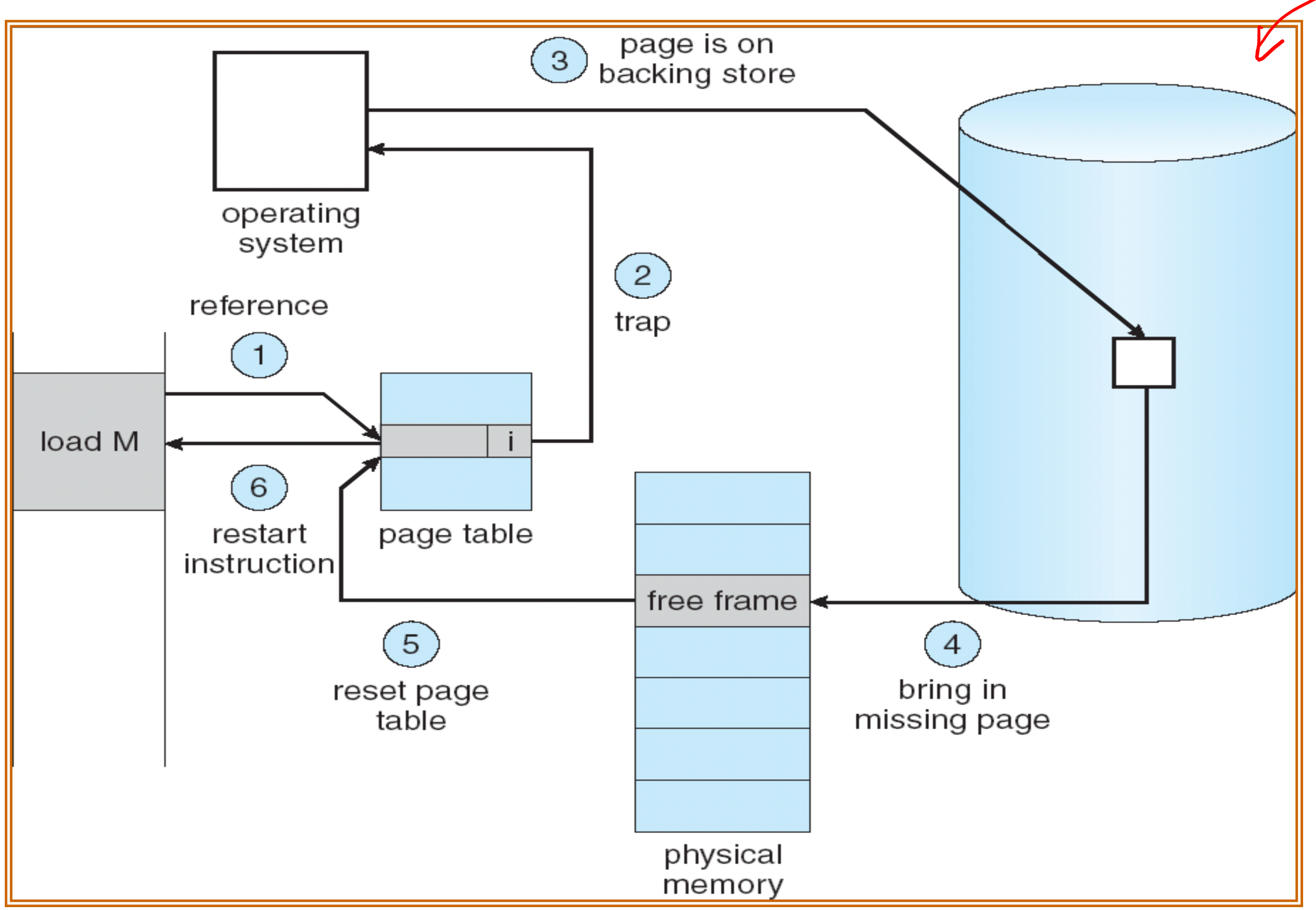


Question 1(a): Page Fault

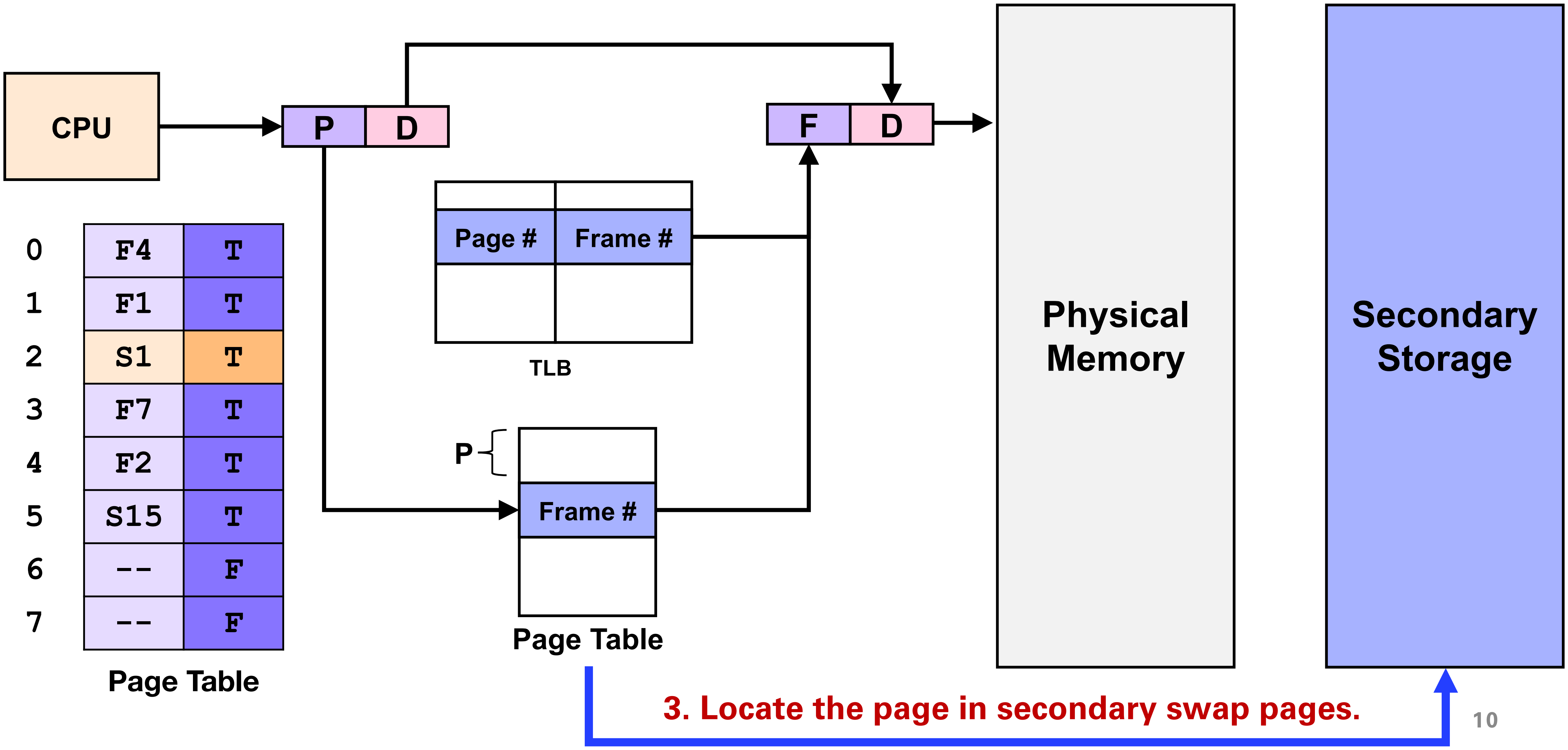
Algorithm for Page Fault Handler, given <Page#>

1. [OS] Global/Local replacement, Replacement algorithm applicable here
2. [OS] Write out the page to be replaced if needed.
3. [OS] Locate the page in secondary swap pages. ←
4. [OS] Load the page into a physical frame. ←
5. [OS] Update page table entries.
6. [OS] Update TLB
7. [OS] Return from Trap

Question 1(a): Page Fault



Question 1(a): Page Fault

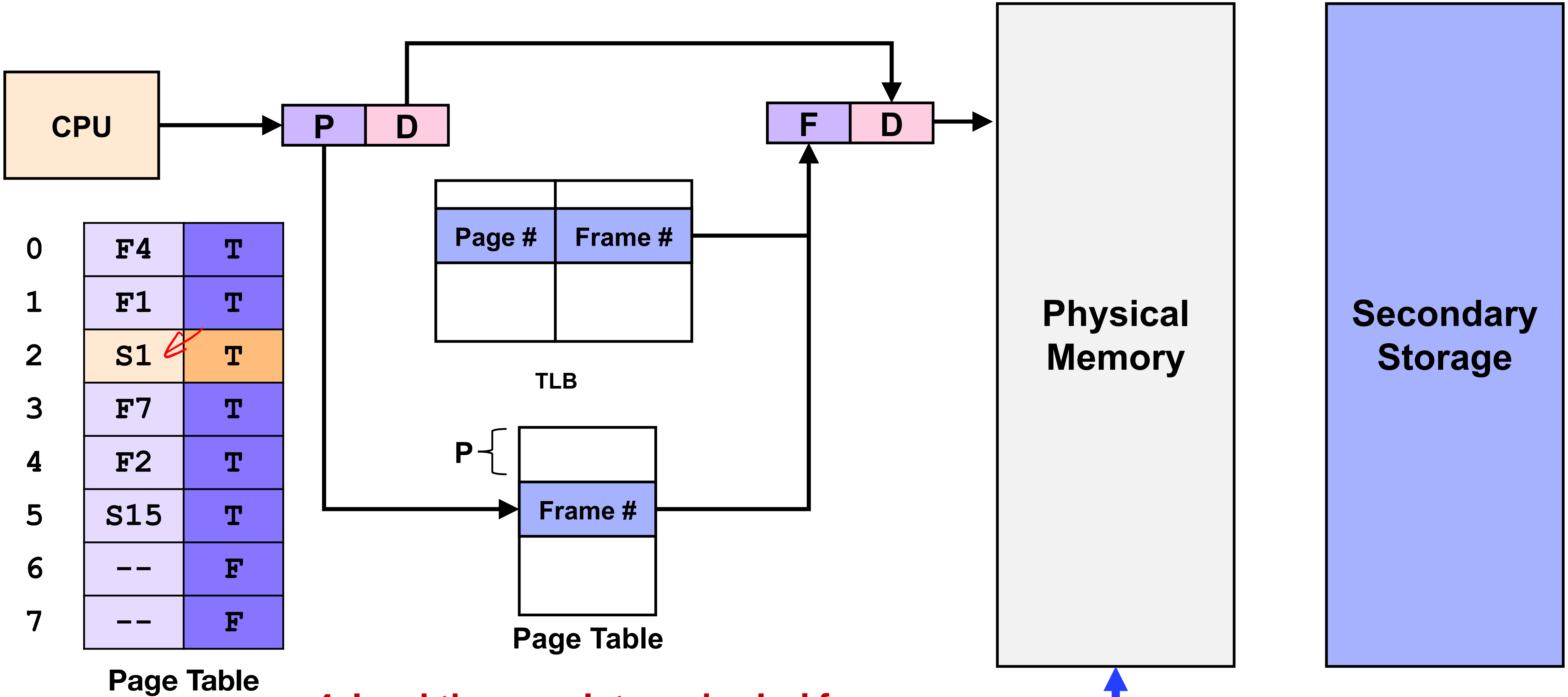


0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

3. Locate the page in secondary swap pages.

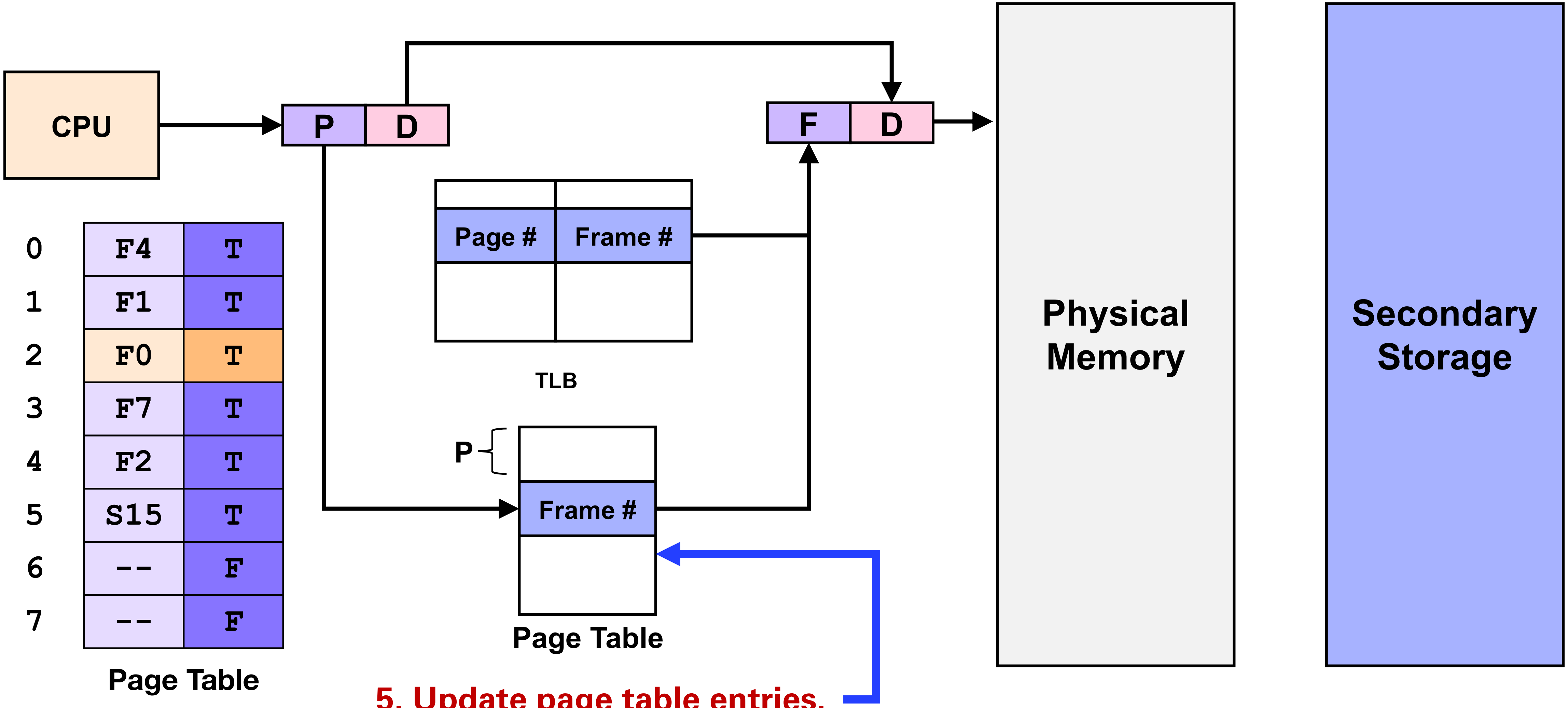
Question 1(a): Page Fault



0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

4. Load the page into a physical frame.

Question 1(a): Page Fault

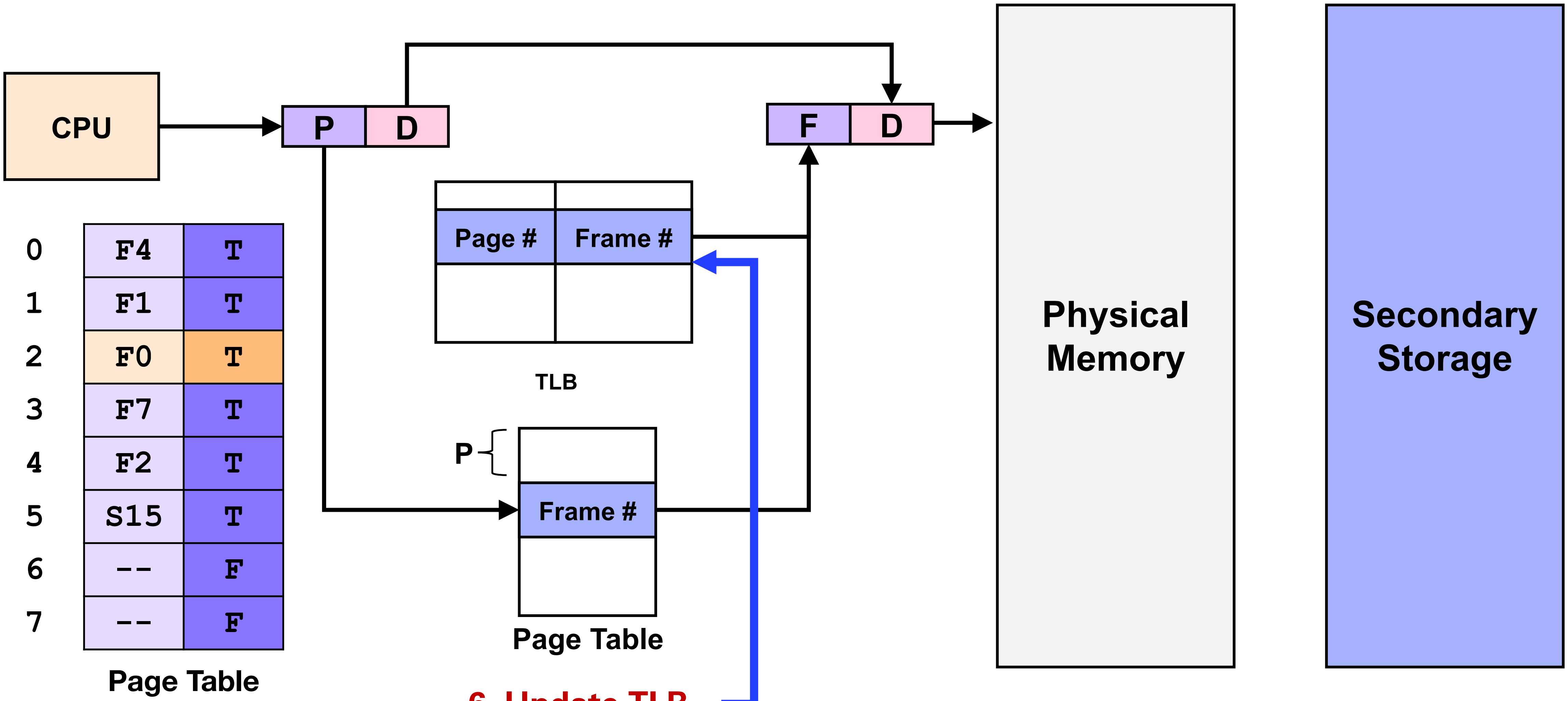


0	F4	T
1	F1	T
2	F0	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

5. Update page table entries.

Question 1(a): Page Fault



0	F4	T
1	F1	T
2	F0	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

6. Update TLB.

Question 1(b)

- Using (a), walkthrough the following access in sequence. For simplicity, only the page number is given.
- If TLB/ Page Replacement is needed, just pick the entry with the smallest page number

i. Access Page 3

ii. Access Page 1

iii. Access Page 5

Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(i) Access Page 3

Is PTE for Page 3 in TLB? **YES**

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(i) Access Page 3

TLB Hit

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

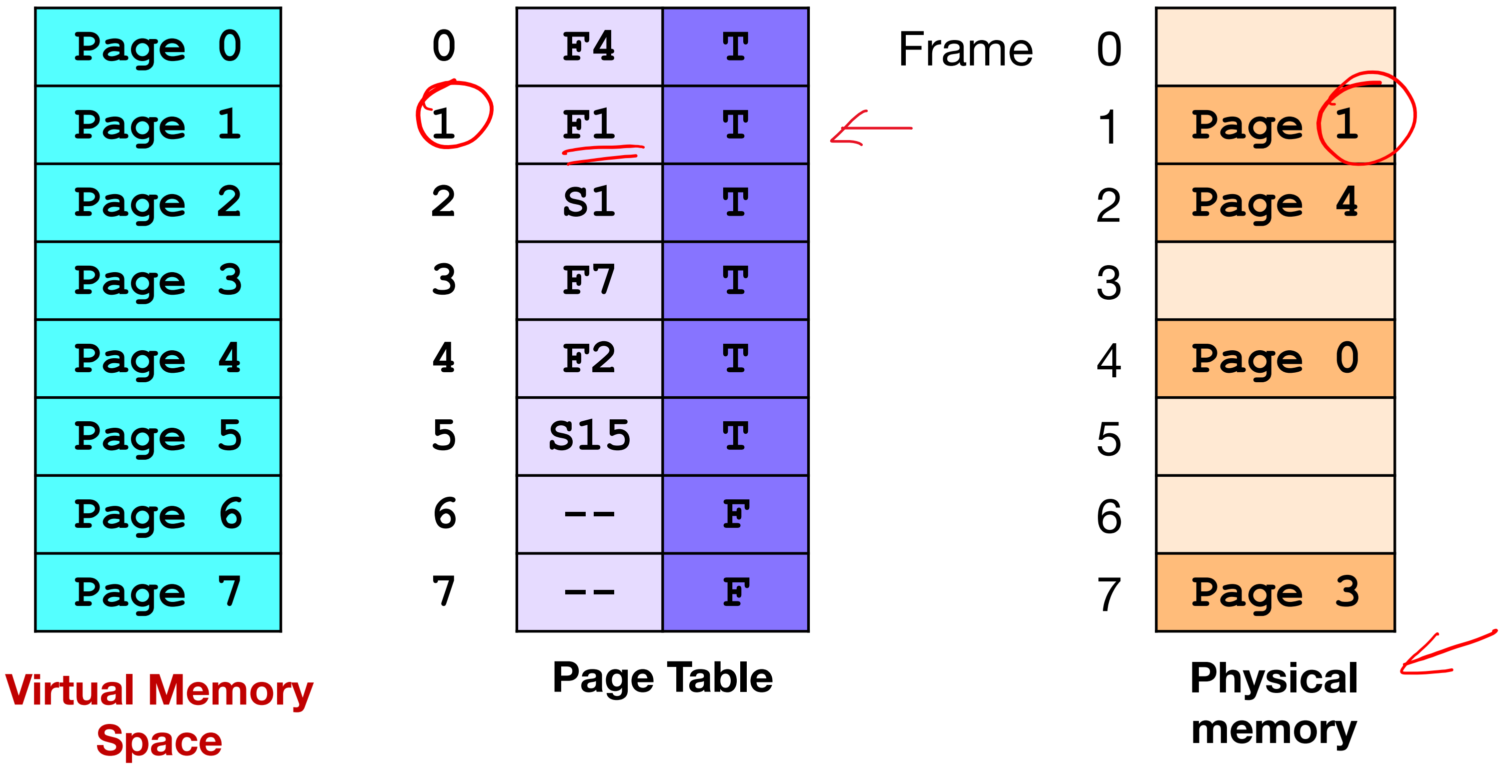
Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(ii) Access Page 1

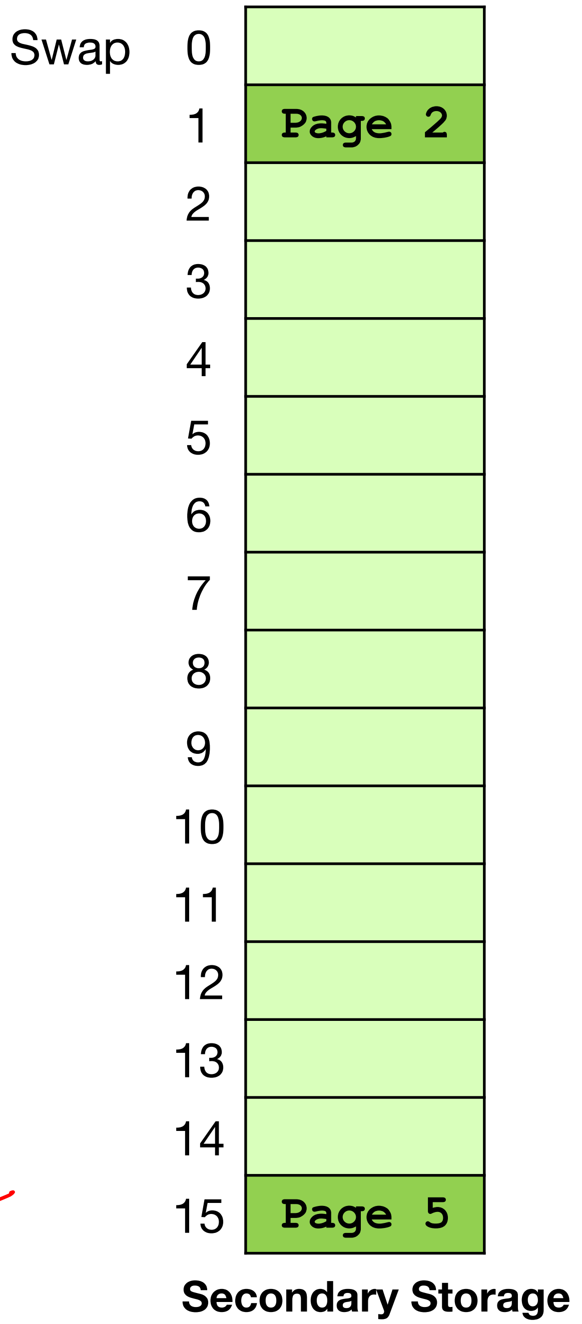
Is PTE for Page 1 in TLB? **NO**



Virtual Memory Space
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Page Table		
0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Frame	Physical memory
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3



Swap	Secondary Storage
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(ii) Access Page 1

Is Page 1 Memory Resident? YES

Virtual Memory Space
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Page Table

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Physical memory

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Secondary Storage

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Question 1(b)

Page No.	Frame No.
3	7
1	1

TLB

(ii) Access Page 1

Load the relevant PTE into TLB (Pick entry with smallest page number for replacement)

Virtual Memory Space
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(b)

Page No.	Frame No.
3	7
1	1

TLB

(ii) Access Page 1

TLB-Miss, Memory Resident, i.e. TLB handler executed

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(b)

(ii) Access Page 1

Page No.	Frame No.
3	7
1	1

TLB

Page No.	Frame No./ Swap Page No.	Memory Resident?	Valid?
0	4	1	1
1	1	1	1
2	1	0	1
3	7	1	1
4	2	1	1
5	15	0	1
6	---	0	0
7	---	0	0

Page Table

Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(iii) Access Page 5

Is PTE for Page 5 in TLB? **NO**

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(iii) Access Page 5

Is Page 5 Memory Resident? **NO**

Virtual Memory Space
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(b)

Page No.	Frame No.
3	7
0	4

TLB

(iii) Access Page 5

Page Replacement is needed, pick the entry with the smallest page number.

Virtual Memory Space
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	F4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	S15	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 0
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	Page 5

Secondary Storage

Question 1(b)

Page No.	Frame No.
3	7
5	4

TLB

(iii) Access Page 5

- **Write out Page 5 from Secondary Storage.**
- **Update page table entries and TLB**
- **Write in Page 0 into Secondary Storage. (Any Swap No.)**

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	S4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	F4	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 5
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	Page 0
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Secondary Storage

Question 1(b)

Page No.	Frame No.
3	7
5	4

TLB

(iii) Access Page 5

TLB Miss, Page Fault, i.e. both TLB and Page Fault handler executed.

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Virtual Memory Space

0	S4	T
1	F1	T
2	S1	T
3	F7	T
4	F2	T
5	F4	T
6	--	F
7	--	F

Page Table

Frame	
0	
1	Page 1
2	Page 4
3	
4	Page 5
5	
6	
7	Page 3

Physical memory

Swap	
0	
1	Page 2
2	
3	
4	Page 0
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Secondary Storage

Question 1(b)

(iii) Access Page 5

Page No.	Frame No.
3	7
5	**4**

TLB

Page No.	Frame No./ Swap Page No.	Memory Resident?	Valid?
0	**Any Swap No.**	**0**	**1**
1	1	1	1
2	1	0	1
3	7	1	1
4	2	1	1
5	**4**	**1**	**1**
6	---	0	0
7	---	0	0

Page Table

Question 1(b)

- Using (a), walkthrough the following access in sequence. For simplicity, only the page number is given.
- If TLB/ Page Replacement is needed, just pick the entry with the smallest page number

i. Access Page 3 **TLB Hit**

ii. Access Page 1 **TLB-Miss, Memory Resident, i.e. TLB handler executed**

iii. Access Page 5 **TLB Miss, Page Fault, i.e. both TLB and Page Fault handler executed.**

Question 1(c)

Hardware Specification:

- TLB access time = 1ns
- Memory access time = 30ns
- Hard disk access time (per page) = 5ms

Give the best and worst memory access scenarios and the respective memory access speed. For simplicity, you can give approximate answers.

Best Scenario: TLB Hit

1ns (Access TLB) + 30ns (Access Memory) = 31ns

Worst Scenario: TLB Miss, Page Fault

1ns (Access TLB) + 30ns (Access Page Table) + 5ms (Swap In Page) + 5ms (Swap Out Page) = 10,000,031ns

Note:

- The above ignores the overhead of re-accessing TLB and/or page table
- 1ms = 10^6 ns

Question 1(d)

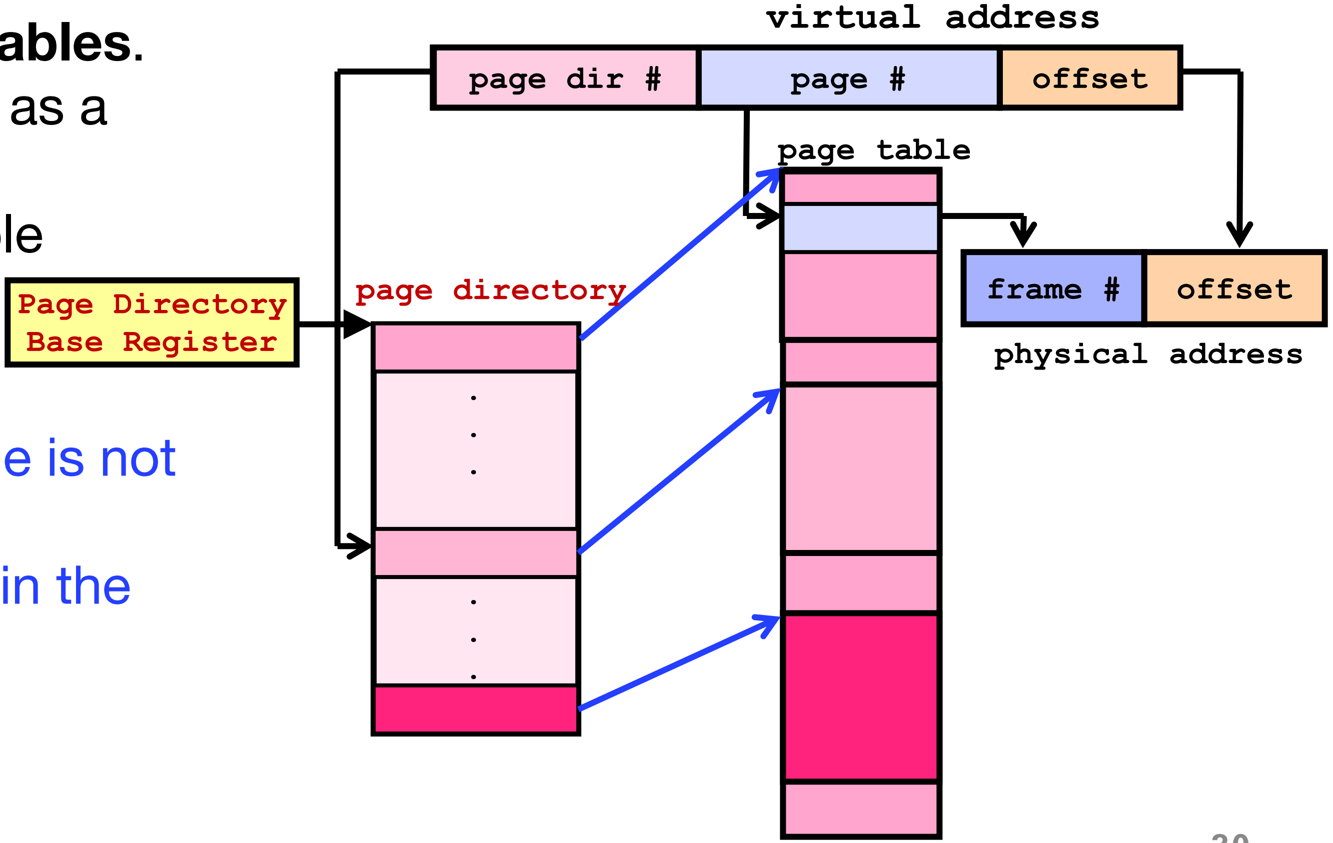
Hardware Specification:

- TLB access time = 1ns
- Memory access time = 30ns
- Hard disk access time (per page) = 5ms

If 2-Level paging is used, is there a worse scenario compared to (c)?

- There are now **two levels of page tables**.
- The page table at the first level acts as a page directory.
- At the second level, there are multiple smaller page tables.

- What happens if a smaller page table is not a memory resident?
- In addition, what happens if a page in the smaller page table is not a memory resident?



Question 1(d)

Hardware Specification:

- TLB access time = 1ns
- Memory access time = 30ns
- Hard disk access time (per page) = 5ms

If 2-Level paging is used, is there a worse scenario compared to (c)?

- When a smaller page table in the PTE of the page directory (first level page table) is not a memory resident (not in physical memory), it results in a page fault.
- In addition, if a particular page to be accessed in the smaller page table is also not a memory resident, it results in a second page fault.
- The possibility of servicing two page faults under the 2-level paging scheme would result in more page swaps and longer access times.

Question 2

Page Table Structure

Question 2

- The Linux machines in the SoC cluster used in the labs have 64-bit processors, but the virtual addresses are 48-bit long (the highest 16 bits are not used).
- The physical frame size is 4KiB.
- The system uses a hierarchical direct page table structure, with each table/directory entry occupying 64 bits (8 bytes) regardless of the level in the hierarchy.

Question 2

Assume Process P runs the following simple program on one of the lab machines:

```
1   #include <stdio.h>
2   #include <stdlib.h>

3   #define ONE_GIG 1 << 30
4   #define NUM_PAGES 1 << 18

5   void main() {
6       char* array = malloc(ONE_GIG);
7       int i;
8       for (i = 0; i < NUM_PAGES; i++)
9           array[i << 12] = 'a';
10      printf("0x%1X\n", array);
11      // point of interest
12      free(array)
```

At line ~~9~~ 10, the program prints out the value of variable **`array`** in hexadecimal:

`0x7F9B55226010`

Consider the point in time when the process execution reaches the point of interest (line 10 in the listing).

Question 2(a)

System Specification
Virtual Addresses: 48-bit long
Frame Size = Page Size = 4 KiB
Page Table / Page Directory Entry: 8 Bytes = 64 Bits

If we use a single-level page table, how much memory space is needed just to store the page table for process P?

Memory Space of a process: maximum **2^{48}** bytes

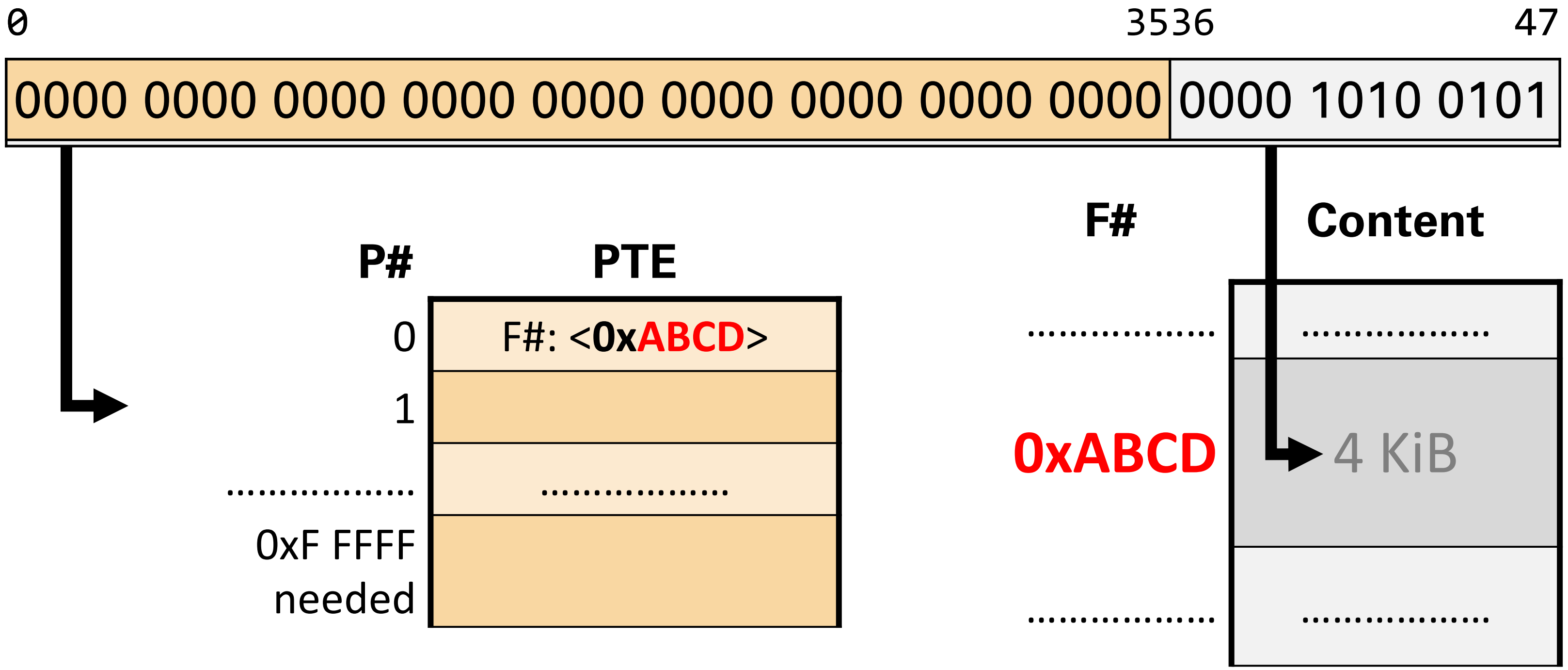
- Split into 4 KiB per page: $2^{48} / 2^{12}$ = **2^{36}** pages
- Entire Page Table = 2^{36} PTEs x 8 bytes per PTE = **2^{39}** bytes (**512GiB**!!)

Question 2(a)

System Specification
Virtual Addresses: 48-bit long
Frame Size = Page Size = 4 KiB
Page Table / Page Directory Entry: 8 Bytes = 64 Bits

Memory Space of a process: maximum **2^{48}** bytes

- Split into 4 KiB per page: 2^{48} / 2^{12} = **2^{36}** pages
- Entire Page Table = 2^{36} PTEs x 8 bytes per PTE = **2^{39}** bytes (**512GiB**!!)



VA = 48 bits | Page/Frame Size = 4KiB| PTE Size = 8 bytes

Question 2(b)

System Specification

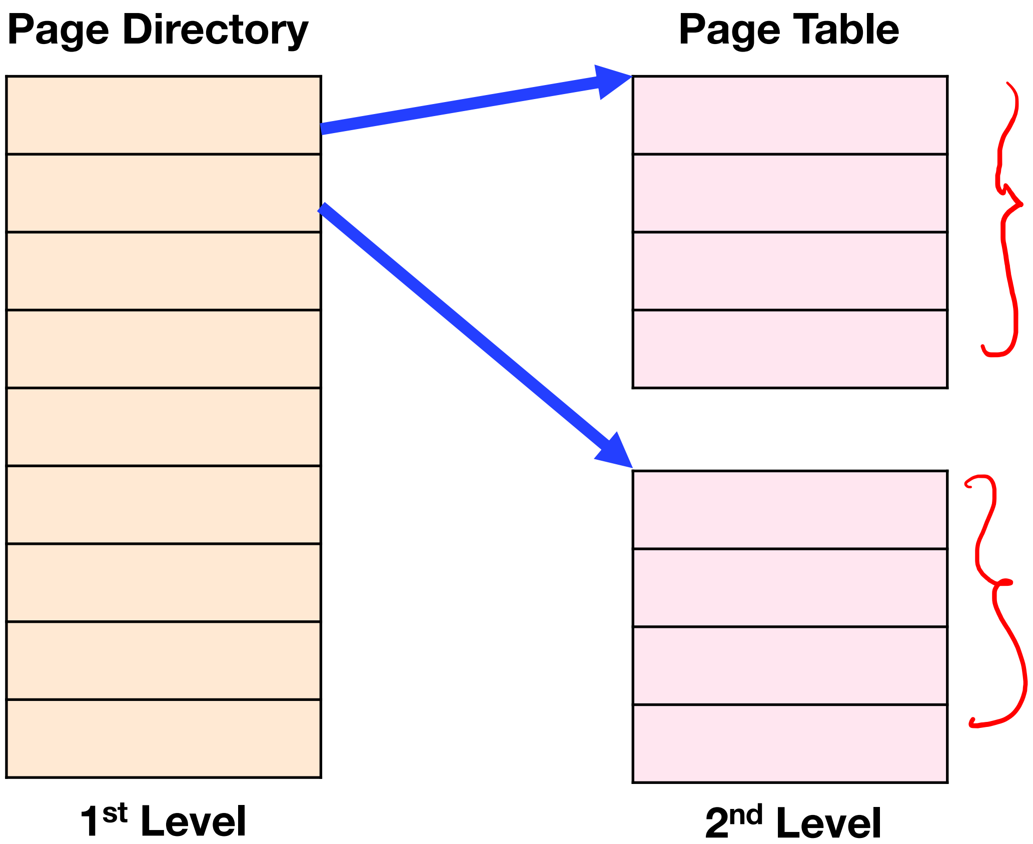
Virtual Addresses: 48-bit long

Frame Size = Page Size = 4 KiB

Page Table / Page Directory Entry: 8 Bytes = 64 Bits

How many potential levels are there in the page-table structure for process P?

Let's start simple with 2-level paging first



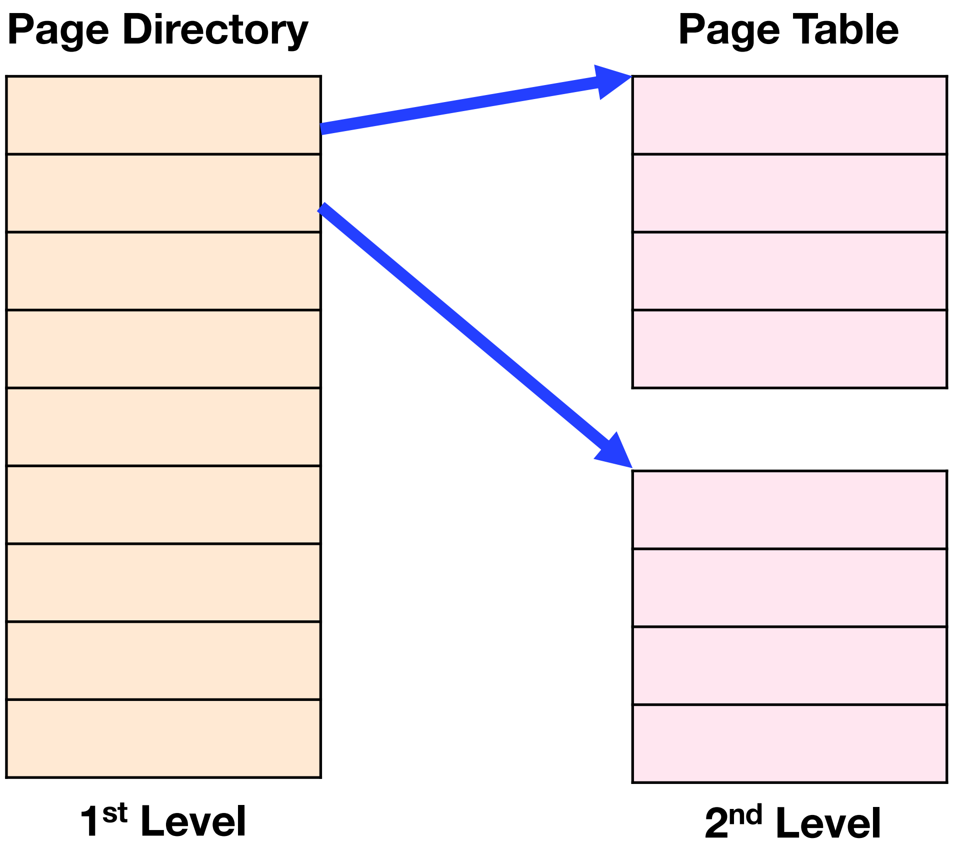
- In multi-level paging, we split the total number of logical pages into multiple regions.
- Each region's information is maintained by a page table.
- Hence there are multiple page tables.
- The total number of page table entries in all page tables should add up to the total number of logical pages.

Question 2(b)

System Specification
Virtual Addresses: 48-bit long
Frame Size = Page Size = 4 KiB
Page Table / Page Directory Entry: 8 Bytes = 64 Bits

How many potential levels are there in the page-table structure for process P?

Let's start simple with 2-level paging first



- We also need to store each page table in physical memory.
- To easily maintain multiple page tables, we can allocate each page table to a frame in physical memory.
- This means that the size of each page table is the same as the size of a logical page.

Number of page table entries in each page table:
4 KiB / 8 Bytes = 2^{12} / 2^{3} Bytes = 512 entries (2^{9})

Question 2(b)

System Specification
Virtual Addresses: 48-bit long
Frame Size = Page Size = 4 KiB
Page Table / Page Directory Entry: 8 Bytes = 64 Bits

Number of page table entries in each page table:
4 KiB / 8 Bytes = 2^{12} / 2^{3} Bytes = 512 entries (2^{9})

Single Level Page Table 2^{36} logical pages

Page No.	Offset
0 … 35	36 … 47
0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 1010 0101

2 Level Page Table

Page Directory No.	Page No.	Offset
0 … 26	27 … 35	36 … 47
0000 0000 0000 0000 0000 0000 000	0 0000 0000	0000 1010 0101

Since each smaller page table only has 2^{9} PTEs, there are 2^{36} / 2^{9} = 2^{27} page directory entries

Still quite big!!
= 2^{27} PDEs x 8 bytes each
= $\mathbf{2^{30}}$ bytes
= **1 GiB**

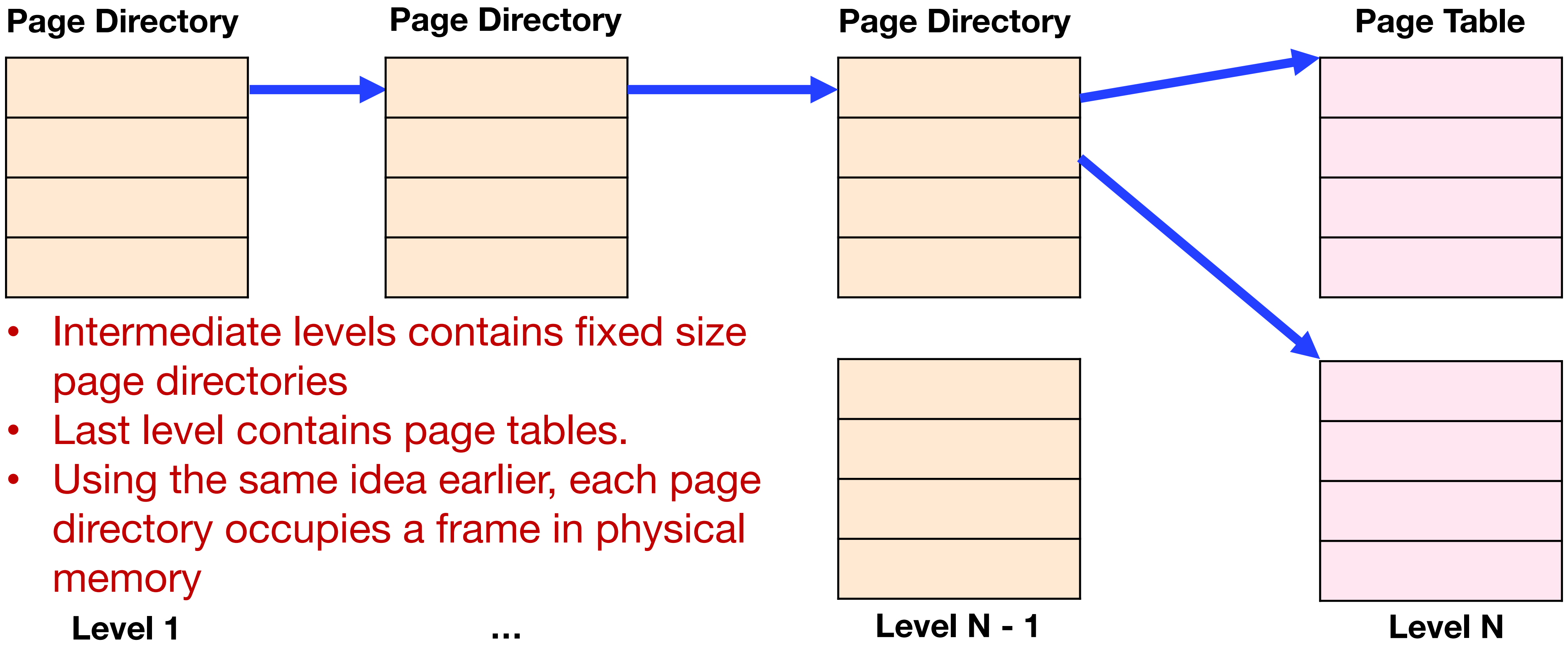
Question 2(b)

System Specification
Virtual Addresses: 48-bit long
Frame Size = Page Size = 4 KiB
Page Table / Page Directory Entry: 8 Bytes = 64 Bits

Number of PTEs / PDEs in each page table / directory:
4 KiB / 8 Bytes = 2^{12} / 2^{3} Bytes = 512 entries (2^{9})

Imagine having more than 2 levels of paging

We need 9 bits of virtual address for each level



- Intermediate levels contains fixed size page directories
- Last level contains page tables.
- Using the same idea earlier, each page directory occupies a frame in physical memory

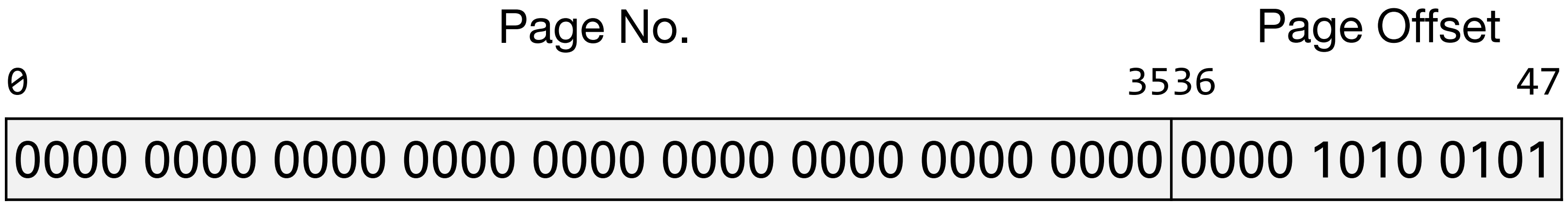
Question 2(b)

System Specification
Virtual Addresses: 48-bit long
Frame Size = Page Size = 4 KiB
Page Table / Page Directory Entry: 8 Bytes = 64 Bits

How many potential levels are there in the page-table structure for process P?

- Now that we know that we need 9 bits for each level

Single Level Page Table 2^{36} logical pages

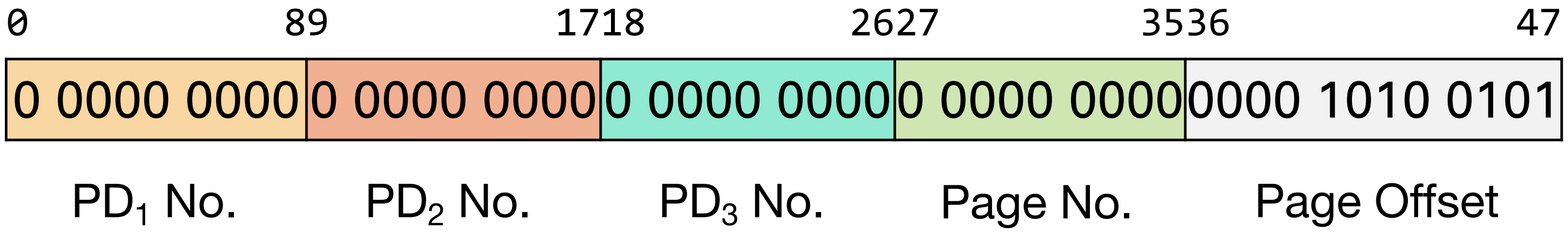


Page No. | Page Offset

0 | 3536 | 47

0000 0000 0000 0000 0000 0000 0000 0000 0000 | 0000 1010 0101

- We can create up to a 4 level page table (36 bits / 9 bits = 4)



0 | 89 | 1718 | 2627 | 3536 | 47

0 0000 0000 | 0 0000 0000 | 0 0000 0000 | 0 0000 0000 | 0000 1010 0101

PD_1 No. | PD_2 No. | PD_3 No. | Page No. | Page Offset

Question 2(c) to 2(f)

System Specification

Virtual Addresses: 48-bit long

Frame Size = Page Size = 4 KiB

Page Table / Page Directory Entry: 8 Bytes = 64 Bits

Value of **`array`** in hexadecimal: **`0x7F9B55226010`**

0 89 1718 2627 3536 47

PD_1 No.	PD_2 No.	PD_3 No.	Page No.	Page Offset
0 1111 1111	0 0110 1101	0 1010 1001	0 0010 0110	0000 0001 0000

```c
#include <stdio.h>
#include <stdlib.h>

#define ONE_GIG 1 << 30
#define NUM_PAGES 1 << 18

void main() {
    char* array = malloc(ONE_GIG);
    int i;
    for (i = 0; i < NUM_PAGES; i++)
        array[i << 12] = 'a';
    printf("0x%1X\n", array);
    // point of interest
    free(array)
```

It would be easier to approach 2(c) to 2(f) in reverse order.

Value of **`array`** in hexadecimal: **`0x7F9B55226010`**

Question 2(f)

`0` `89` `1718` `2627` `3536` `47`

PD$_1$ No.	PD$_2$ No.	PD$_3$ No.	Page No.	Page Offset
0 1111 1111	0 0110 1101	0 1010 1001	0 0010 0110	0000 0001 0000

↑

How many physical frames are holding information related to process P's dynamically allocated data in the last level of the page-table structure?

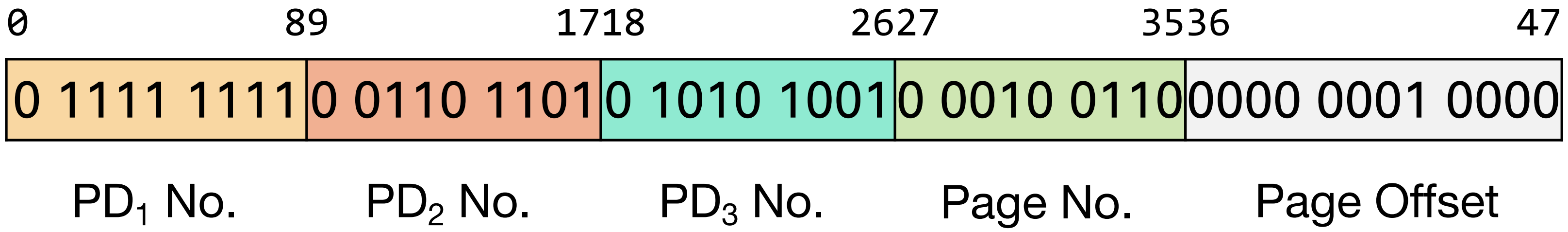
- If we split the dynamically allocated 1GiB of data into 4 KiB pages, there would be $2^{30} / 2^{12} = 2^{18}$ pages.
- However, the array was not allocated at the start of the page boundary.
  - The 12 least significant bits in the virtual address are not 0.
  - The first byte of the array is at offset 16
- Hence, the dynamically allocated data would require $2^{18} + 1$ pages.
- At the last level of the page table structure, we would need
  - $\boxed{(2^{18} / 2^9) + 1}$ = $2^9 + 1 = 513$ frames to hold $2^{18} + 1$ PTEs.

$$\downarrow$$

$$\boxed{\frac{2^{18}+1}{2^9}}$$

Value of **`array`** in hexadecimal: **`0x7F9B55226010`**

Question 2(e)

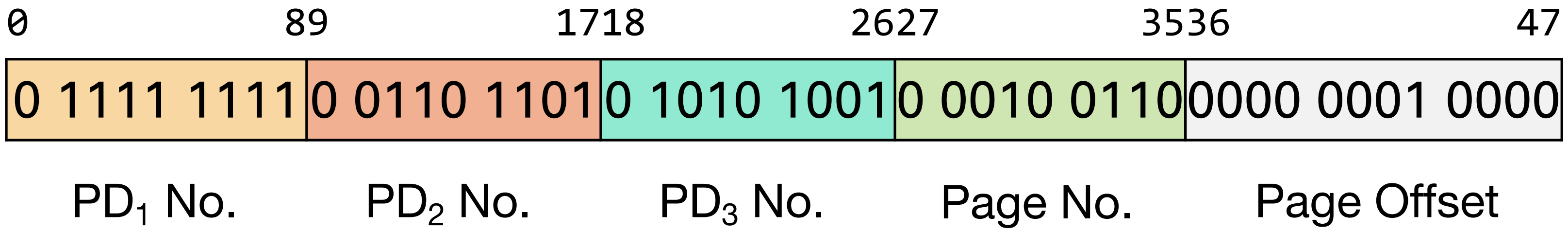


How many physical frames are holding information related to process P’s dynamically allocated data in the penultimate level of the page-table structure (i.e., the level before the last one).

- 2 physical frames to store to store 513 entries = $2^9 + 1$ entries at the penultimate level
- $\lceil \frac{2^9+1}{2^9} \rceil = 2$

Value of **`array`** in hexadecimal: **`0x7F9B55226010`**

Question 2(d)



0 89 1718 2627 3536 47

0 1111 1111 | 0 0110 1101 | 0 1010 1001 | 0 0010 0110 | 0000 0001 0000

PD_1 No. | PD_2 No. | PD_3 No. | Page No. | Page Offset

How many physical frames are holding information related to process P’s dynamically allocated data in the second level of the hierarchical page-table structure (next after the root)?

- 1 physical frame to store 2 entries.

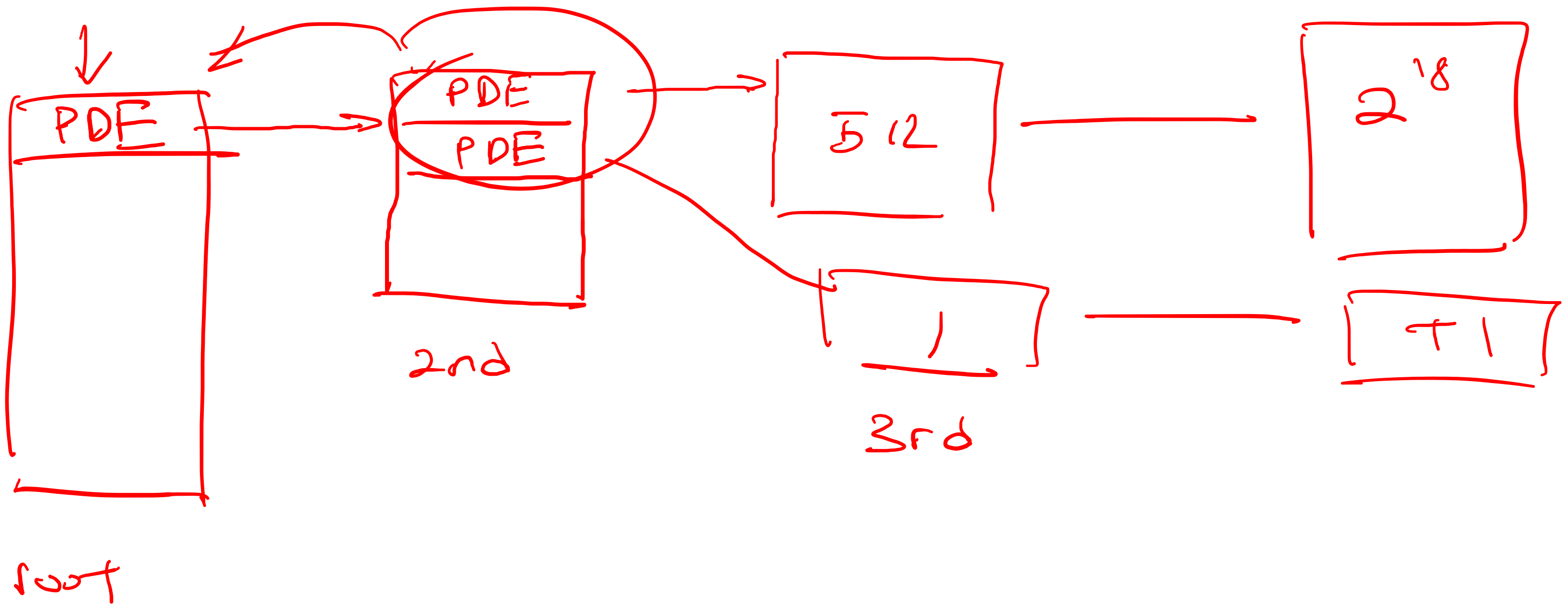
Value of `array` in hexadecimal: `0x7F9B55226010`

Question 2(c)

0	89	1718	2627	3536 47
0 1111 1111	0 0110 1101	0 1010 1001	0 0010 0110	0000 0001 0000
PD$_1$ No.	PD$_2$ No.	PD$_3$ No.	Page No.	Page Offset

How many entries in the root page directory are holding information related to process P's dynamically allocated data?

- 1 entry is needed for the root level page directory.



Question 3

Second Chance Page Replacement

Question 3: Second Chance

- Below is a snapshot of the memory frames in RAM on a system with virtual memory.
- The memory pages in the frame is shown as <Process><Page#>
  - e.g. B17 means Page 17 of Process B.

RAM

Frame #	Content
0	B17
1	A31
2	A04
3	A17

OS

Victim

Ref Bit
~~1~~ 0
1
0
1

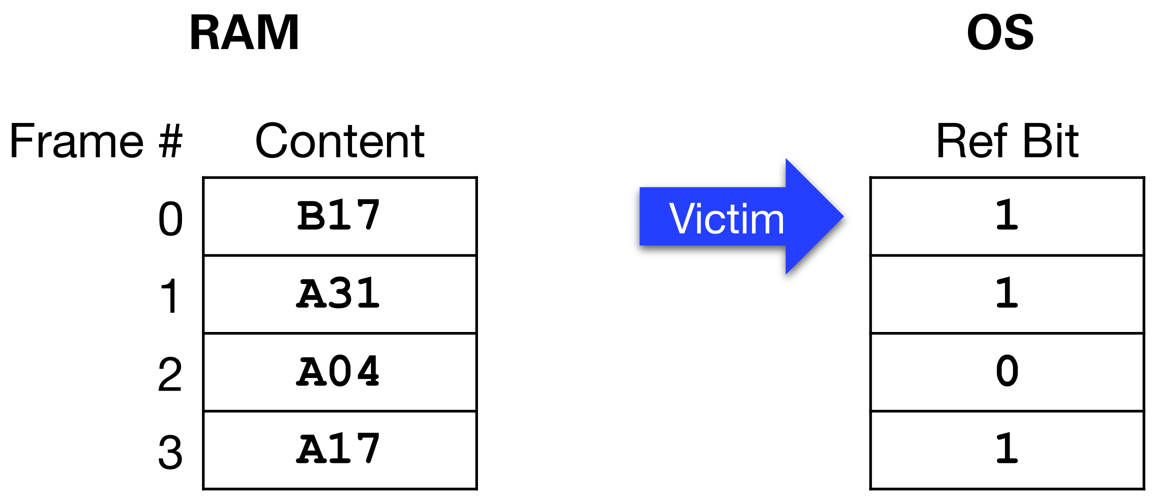
The OS maintains additional information shown on the right to perform **second chance** page replacement algorithm on the memory frames.

Question 3(a)

Suppose **Process A accesses Page 8** (i.e. A08) now, answer the following:

i. Which **memory frame** will be replaced?

ii. Give the steps OS takes to update the affected page table entries.



RAM

Frame #	Content
0	B17
1	A31
2	A04
3	A17

OS

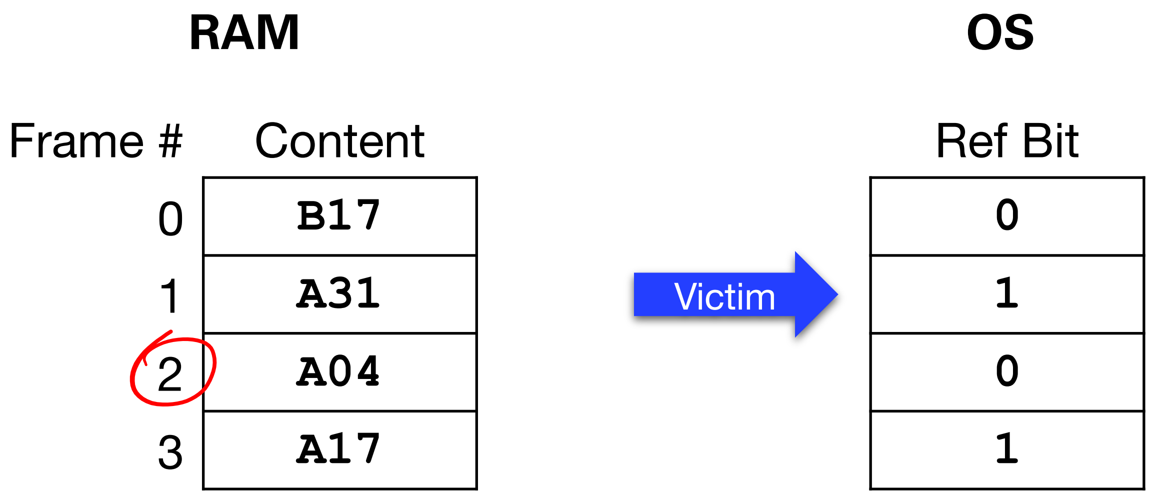
Ref Bit
1
1
0
1

Question 3(a)

Suppose **Process A accesses Page 8** (i.e. A08) now, answer the following:

i. Which **memory frame** will be replaced?

ii. Give the steps OS takes to update the affected page table entries.



RAM

Frame #	Content
0	B17
1	A31
2	A04
3	A17

OS

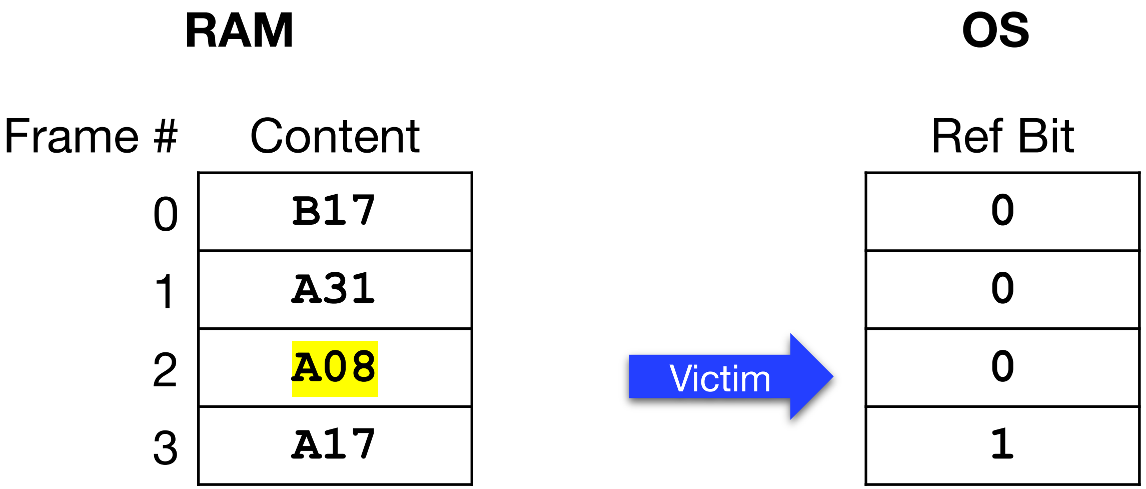
Ref Bit
0
1
0
1

Question 3(a)

Suppose **Process A accesses Page 8** (i.e. A08) now, answer the following:

i. Which **memory frame** will be replaced? **Frame 2** ←

ii. Give the steps OS takes to update the affected page table entries.



Frame #	Content	Ref Bit
0	B17	0
1	A31	0
2	A08	0
3	A17	1

Question 3(a)

Suppose **Process A accesses Page 8** (i.e. A08) now, answer the following:

i. Which **memory frame** will be replaced? **Frame 2**

ii. Give the steps OS takes to update the affected page table entries.

- The OS discovers that Process A has an existing page in frame 2 **(Page 4)**
- The OS will then mark A's page table entry for page 4 as invalid, and page 8 will be marked as valid, with the mapped frame number being 2.
- Without an inverted page table, this procedure will be very slow.

BEFORE

Frame #	Content
0	B17
1	A31
2	A04
3	A17

AFTER

Frame #	Content
0	B17
1	A31
2	A08
3	A17

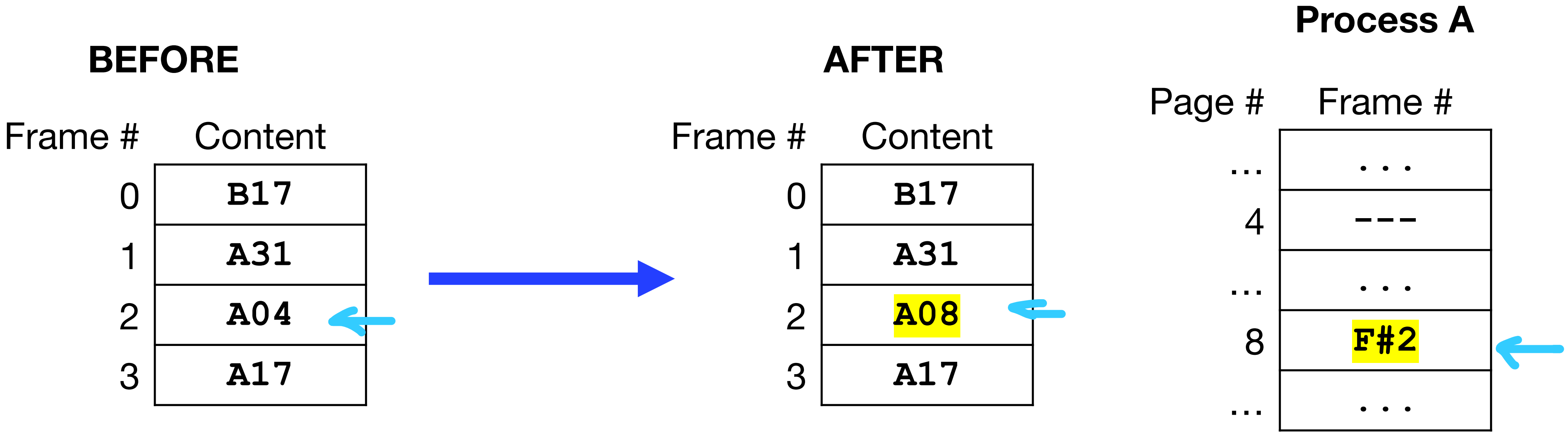
Page #	Frame #
…	. . .
4	---
…	. . .
8	F#2
…	. . .

Question 3(a)

Suppose **Process A accesses Page 8** (i.e. A08) now, answer the following:

i. Which **memory frame** will be replaced? **Frame 2**

ii. Give the steps OS takes to update the affected page table entries.



BEFORE

Frame #	Content
0	B17
1	A31
2	A04
3	A17

AFTER

Frame #	Content
0	B17
1	A31
2	A08
3	A17

Process A

Page #	Frame #
…	. . .
4	---
…	. . .
8	F#2
…	. . .

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

i. Which memory frame will be replaced?

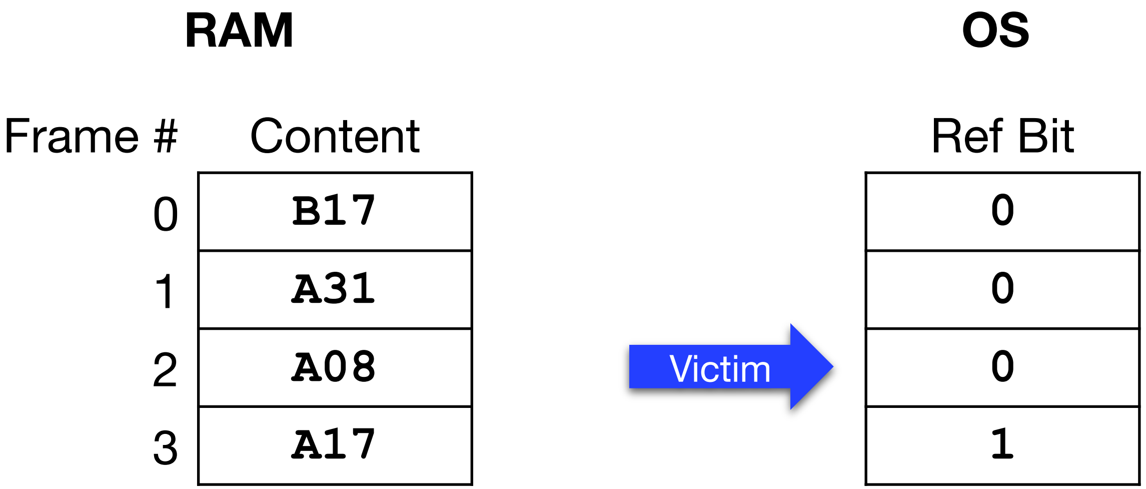
ii. If OS keeps an **inverted page table**, give the content of the inverted page table **after** the replacements.

iii. Give the steps OS takes to update the affected page table entries with the help of inverted page table.

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

i. Which memory frame will be replaced?



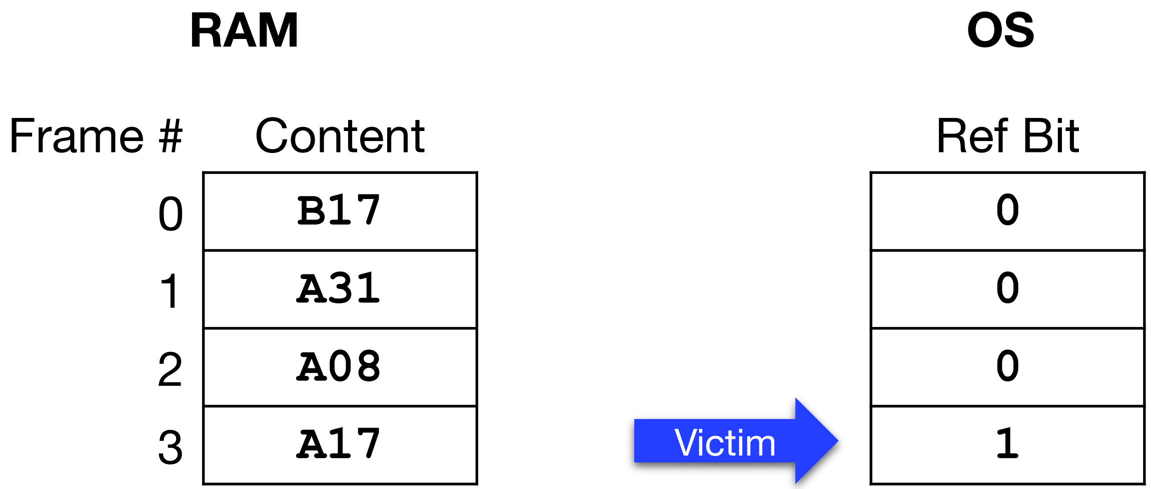
RAM | **OS**

Frame #	Content		Ref Bit
0	B17		0
1	A31		0
2	A08	Victim	0
3	A17		1

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

i. Which memory frame will be replaced?



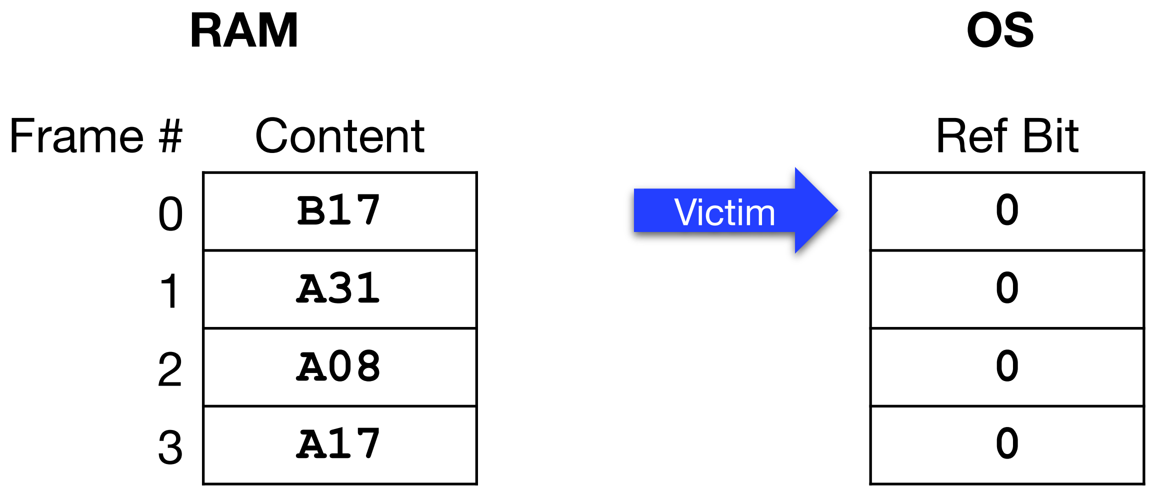
RAM **OS**

Frame #	Content		Ref Bit
0	B17		0
1	A31		0
2	A08		0
3	A17	Victim →	1

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

i. Which memory frame will be replaced? **Frame 0**



RAM

Frame #	Content
0	B17
1	A31
2	A08
3	A17

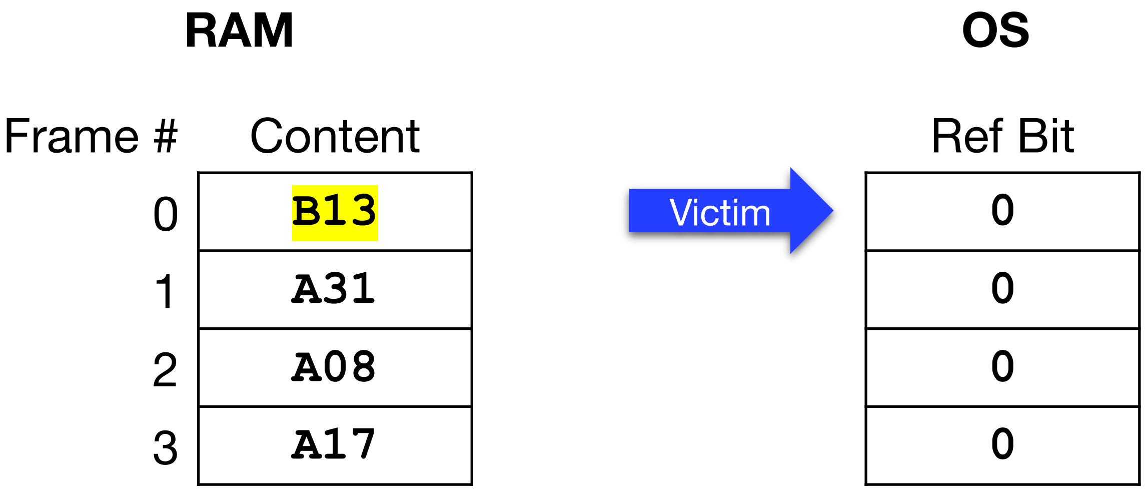
OS

Ref Bit
0
0
0
0

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

i. Which memory frame will be replaced? **Frame 0**



RAM **OS**

Frame #	Content		Ref Bit
0	B13	Victim	0
1	A31		0
2	A08		0
3	A17		0

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

ii. If OS keeps an inverted page table, give the content of the inverted page table after the replacements.

0	B	17
1	A	31
2	A	08
3	A	17

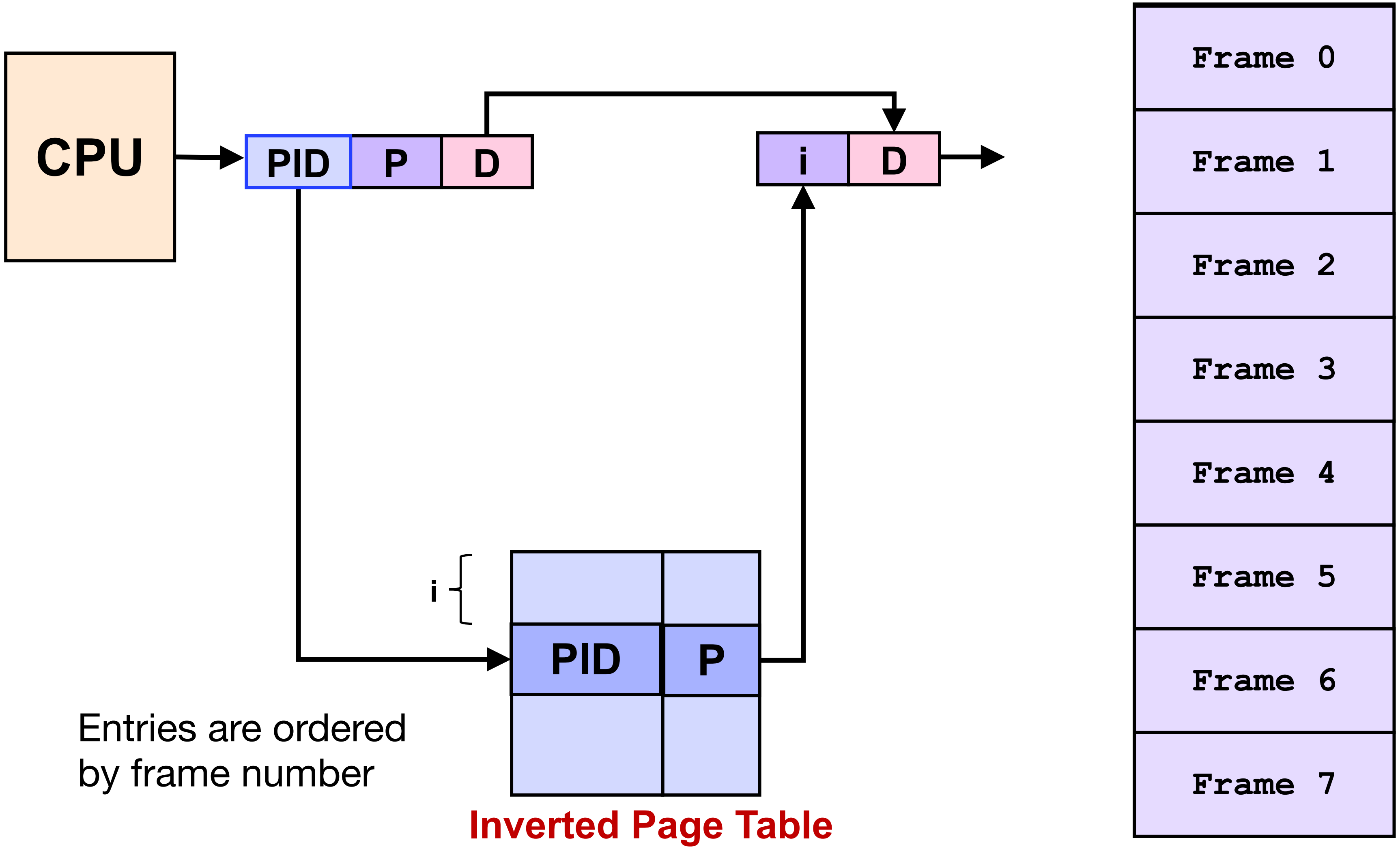
Before

Frame #	PID	Page #
0	B	13
1	A	31
2	A	08
3	A	17

After

inverted page table

Inverted Page Table



Entries are ordered
by frame number

Inverted Page Table

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

iii. Give the steps OS takes to update the affected page table entries with the help of inverted page table.

- The inverted page table immediately shows that Process B has page 17 in frame 0.
- B's page table entry for 17 will be marked as invalid, and page 13 will be marked as valid.

Question 3(b)

Continuing from (a), if **Process B accesses Page 13** (i.e. B13) now, answer the following:

iii. Give the steps OS takes to update the affected page table entries with the help of inverted page table.

BEFORE

Frame #	PID	Page #
0	B	17
1	A	31
2	A	08
3	A	17

AFTER

Frame #	PID	Page #
0	B	13
1	A	31
2	A	08
3	A	17

Process B

Page #	Frame #
…	...
13	F#0
…	...
17	---
…	...

END OF TUTORIAL