Tutorial 8

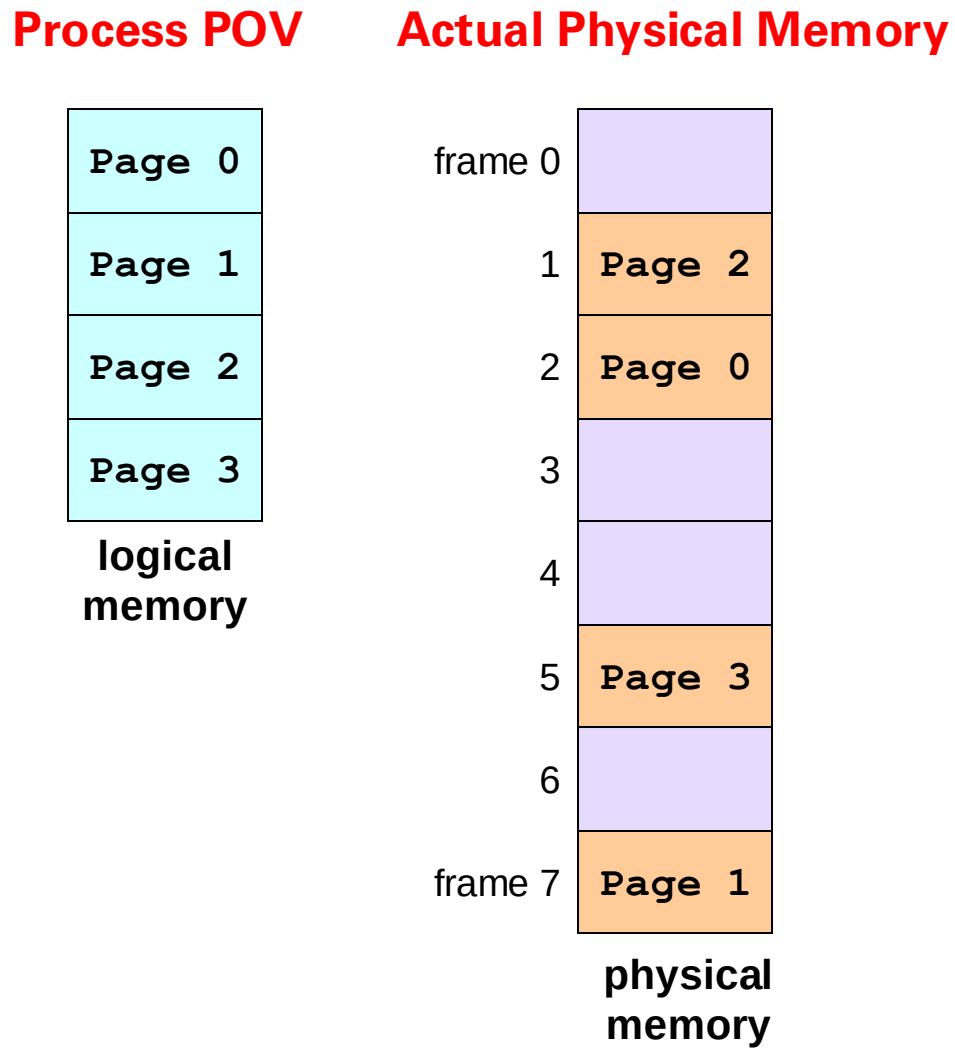
CS2106: Introduction to Operating Systems

Disjoint Memory Management

- Last Week: Contiguous Memory Management
  - Memory is allocated to a process in a single, continuous block.
    - Fixed Partitioning
    - Dynamic Partitioning
    - Partition Info
- This Week: Disjoint Memory Management
  - Now processes **do not** occupy a contiguous memory region.
    - Paging Scheme
    - Segmentation Scheme
    - Segmentation with Paging

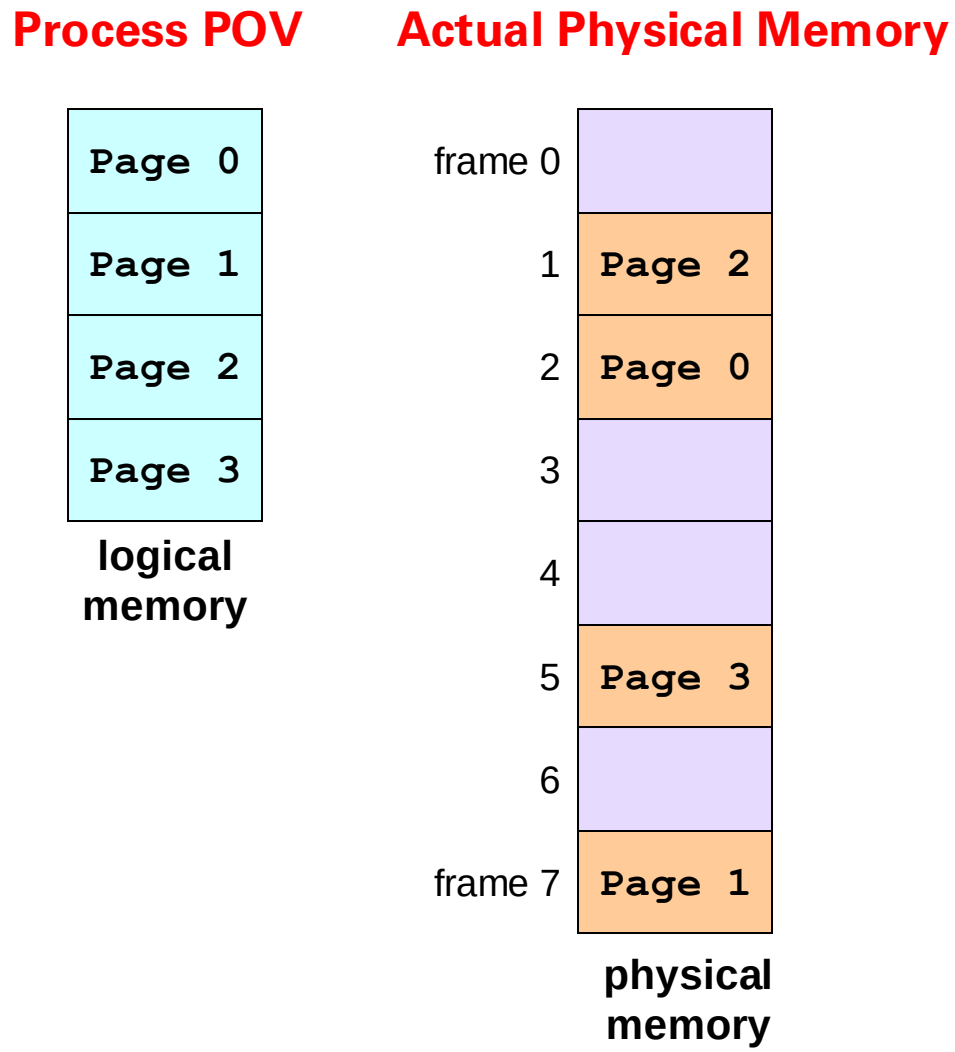
Paging Scheme

- Logical Addresses are how a process view its memory space.
- The physical memory is split into regions of fixed size.
  - Known as the **physical frame**.
- The logical memory of a process is split into regions of the **same size**.
  - Known as the **logical page**.



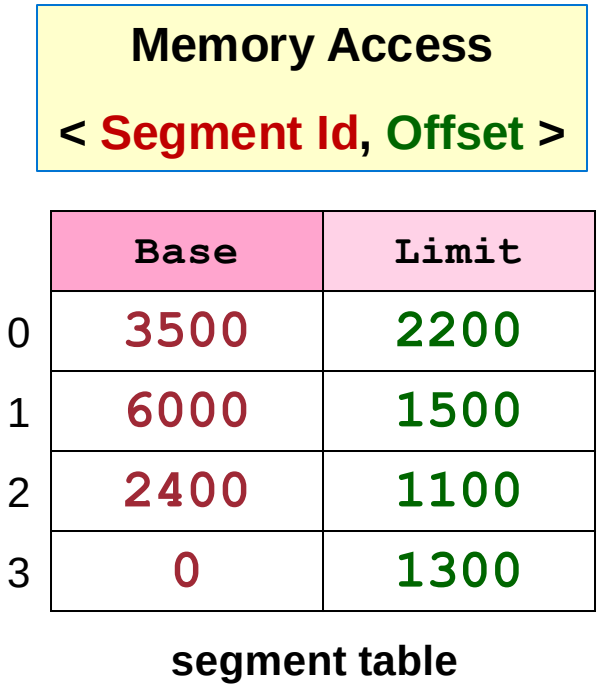
Paging Scheme

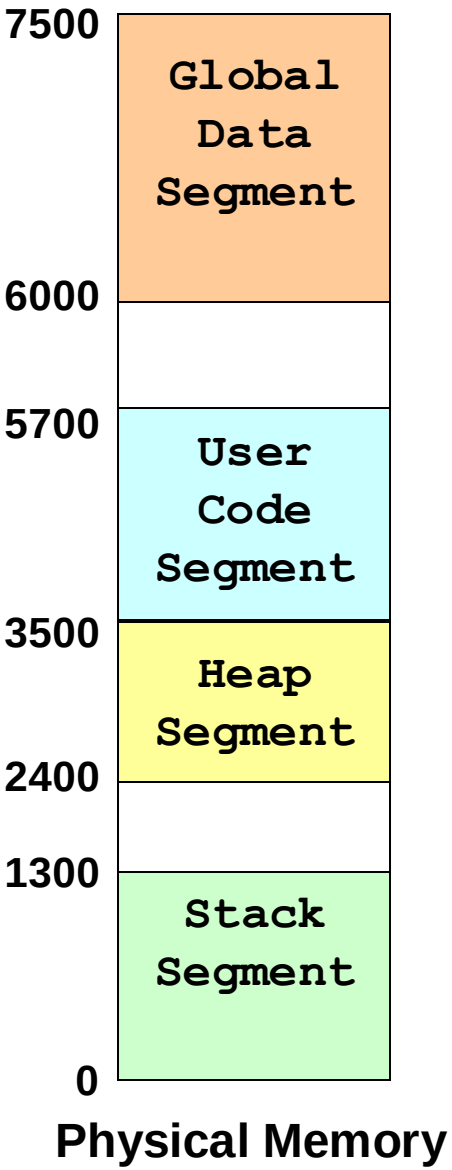
- To a process, its logical memory space appears contiguous
  - Logical Memory Address appears continuous to a process
- But actually, its occupied physical memory can be disjoined.
  - Physical Memory Location may not be always at continuous memory



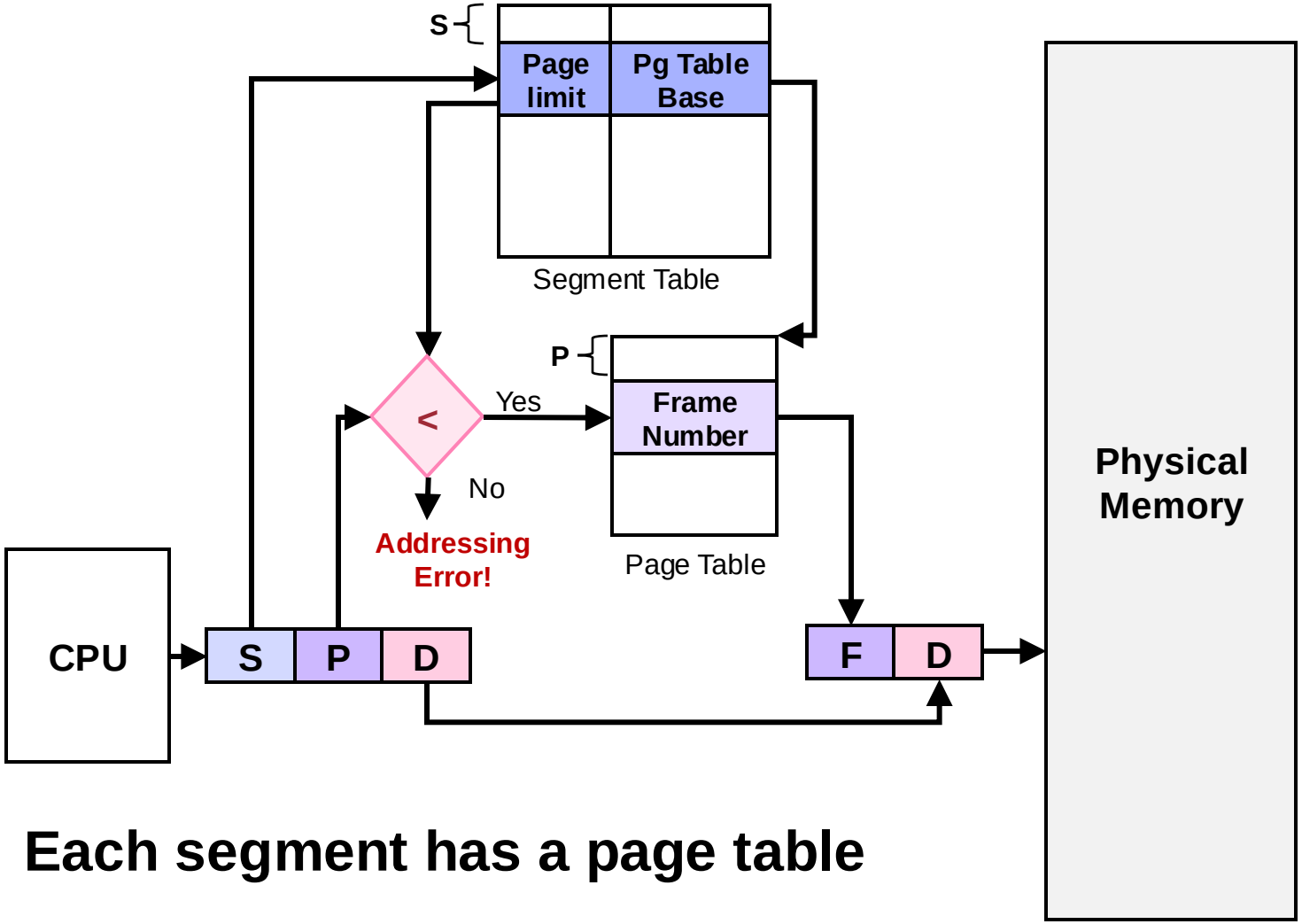
Segmentation Scheme

- Separate the regions in the memory space of a process into multiple segments.
- Each segment is mapped to contiguous physical memory region.

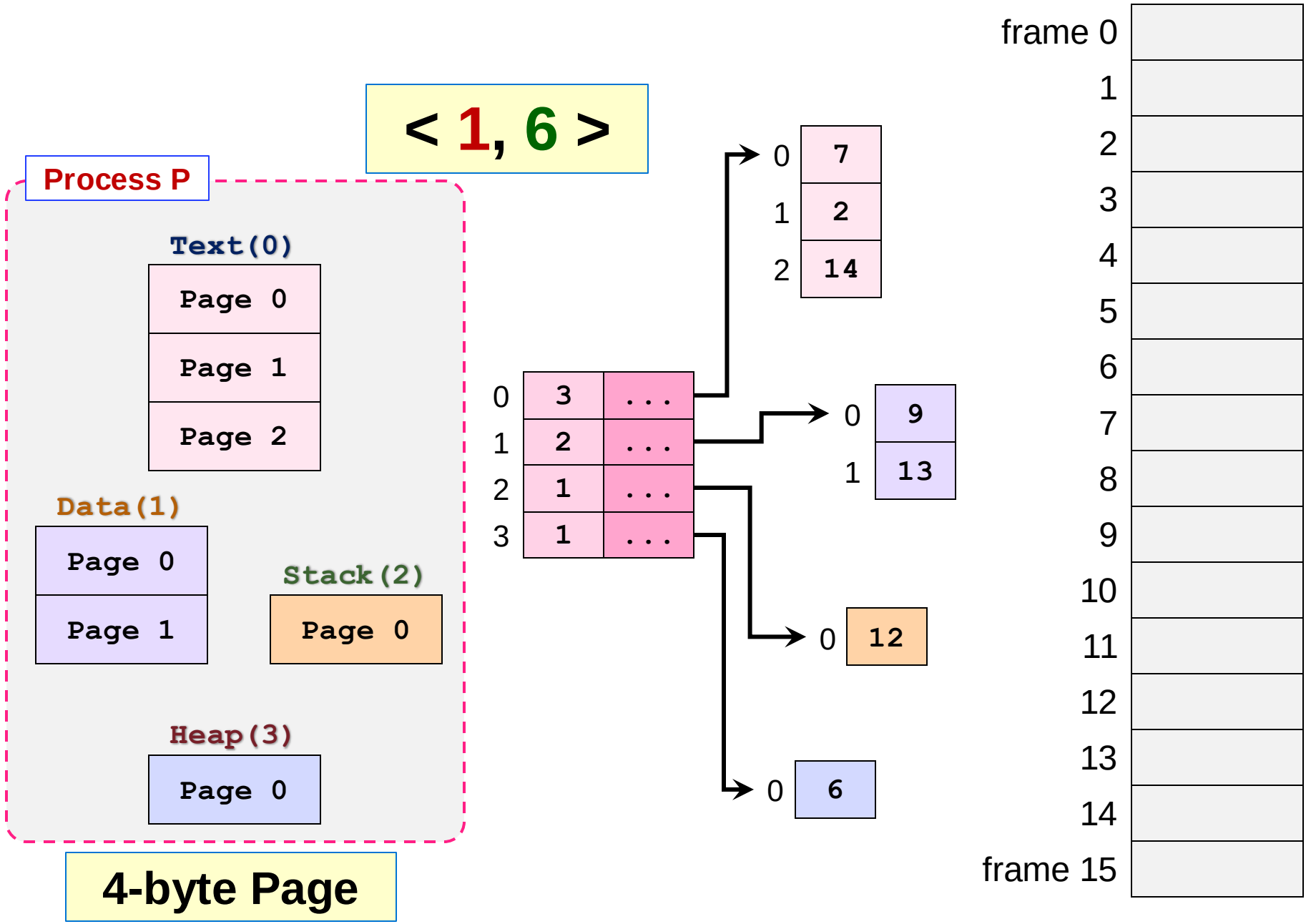




Segmentation with Paging



Each segment has a page table


< 1, 6 >
Process P
Text(0)
Page 0
Page 1
Page 2
Data(1)
Page 0
Page 1
Stack(2)
Page 0
Heap(3)
Page 0
0 3 . . .
1 2 . . .
2 1 . . .
3 1 . . .
0 7
1 2
2 14
0 9
1 13
0 12
0 6
frame 0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
frame 15
4-byte Page

Question 1

Paging/Segmentation/Hybrid Schemes

Question 1

- Let us use a tiny example to understand the various disjoint memory schemes.
- For simplicity, we assume there are only two types of memory usage in a program:
  - **`text`** (instruction) and
  - **`data`** (global variables).

Question 1

The following questions assume that program **P** has:

- 6 instructions, each fitting in a processor word
  - (instruction words #1 to #6)
- 5 data words
  - (data words #1 to #5)

Q1: Paging

- Given the following parameters
  - Page Size = Frame Size = 4 words
  - Largest logical memory size = 16 words
  - Number of physical memory frames = 16
- Assuming that P's data region is placed right after the instruction region in the logical memory space, fill in the following page table.
- Use frames 5, 2, 10, 9 for pages 0, 1, 2, 3 respectively (note: you may not need all frames).
- Indicate the value of the valid bit for all page table entries

Q1: Paging

Processor Action	Page Number
Fetch 1st Instruction	0
Load the 2nd Data Word	1
Load the 3rd Data Word	2
Load the 6th Data word (This is intentionally outside of the range)	None

Page #	Frame #	Valid
0	5	T
1	2	T
2	10	T
3	--	F

Frame 9 is not in the page table because the program P does not use Page 3

Logical Memory Space

Page 0	0 1 2 3	1st Instruction 2nd Instruction 3rd Instruction 4th Instruction
Page 1	4 5 6 7	5th Instruction 6th Instruction 1st Data Word 2nd Data Word
Page 2	8 9 10 11	3rd Data Word 4th Data Word 5th Data Word <empty>
Page 3	12 13 14 15	<empty> <empty> <empty> <empty>

Q1: Paging

- Note 1: Observe that the consecutive logical addresses may not be consecutive in physical memory (e.g. 2nd and 3rd Data word).
- Note 2: Incorrect memory access is not catchable if it is still within valid page boundary.

Find out the logical address and the corresponding physical address for the following actions taken by the processor.

Page #	Frame #	Valid
0	5	T
1	2	T
2	10	T
3	--	F

Processor Action	Page
Fetch 1st Instruction	0
Load the 2nd Data Word	1
Load the 3rd Data Word	2
Load the 6th Data word	None

Processor Action	Logical Address	Physical Address
Fetch 1st Instruction	0	5 × 4 + 0 = 20
Load the 2nd Data Word	7	2 × 4 + 3 = 13
Load the 3rd Data Word	8	10 × 4 + 0 = 40
Load the 6th Data word (This is intentionally outside of the range)	11	10 × 4 + 3 = 43

Q1: Segmentation

- Assuming **`text`** and **`data`** are stored in segments 0 and 1 respectively, fill in the following segment table.
- Use addresses 50 and 23 as the starting addresses for the two segments, respectively.

Segment #	Base Address	Limit
0	50	6
1	23	5

Q1: Segmentation

Similar to (a), find out the logical address and physical address for the following processor actions:

Segment #	Base Address	Limit
0	50	6
1	23	5

Processor Action	Logical Address	Physical Address
Fetch 1st Instruction	<0, 0>	50 + 0 = 50
Load the 2nd Data Word	<1, 1>	23 + 1 = 24
Load the 3rd Data Word	<1, 2>	23 + 2 = 25
Load the 6th Data word (This is intentionally outside of the range)	<1, 5>	Triggers memory addressing error

Q1: Segmentation with Paging

- Assuming the following parameters:
  - Page Size = Frame Size = 4 words
  - Number of physical memory frames = 16
  - Maximum size of each segment = 4 pages
- Furthermore, assume the
  - pages from the code segment are allocated to frames 7, 4, 1 and 2
  - pages from the data segment allocated to frames 9, 3, 14 and 6
  - (note that you may not need all of them).

Q1: Segmentation with Paging

- Draw the segment and page tables for this setup, then fill in the processor action table.
- For the logical addresses, use the notation of
  - <segment id, page number, offset>.

Q1: Segmentation with Paging

Logical Memory Space

Page 0	0 1 2 3	1st Instruction 2nd Instruction 3rd Instruction 4th Instruction
Page 1	4 5 6 7	5th Instruction 6th Instruction <empty> <empty>
Page 0	8 9 10 11	1st Data Word 2nd Data Word 3rd Data Word 4th Data Word
Page 1	12 13 14 15	5th Data Word <empty> <empty> <empty>

Segment #	Page Limit	Page Table Base
0	2	
1	2	

Page #	Frame #	Valid
0	7	T
1	4	T
2	--	F
3	--	F

Page #	Frame #	Valid
0	9	T
1	3	T
2	--	F
3	--	F

Q1: Segmentation with Paging

- Draw the segment and page tables for this setup, then fill in the processor action table.
- For the logical addresses, use the notation of
  - <segment id, page number, offset>.

Processor Action	Logical Address	Physical Address
Fetch 1st Instruction	<0, 0, 0>	7 × 4 + 0 = 28
Load the 2nd Data Word	<1, 0, 1>	9 × 4 + 1 = 37
Load the 3rd Data Word	<1, 0, 2>	9 × 4 + 2 = 38
Load the 6th Data word (This is intentionally outside of the range)	<1, 1, 1>	3 × 4 + 1 = 13

Question 2

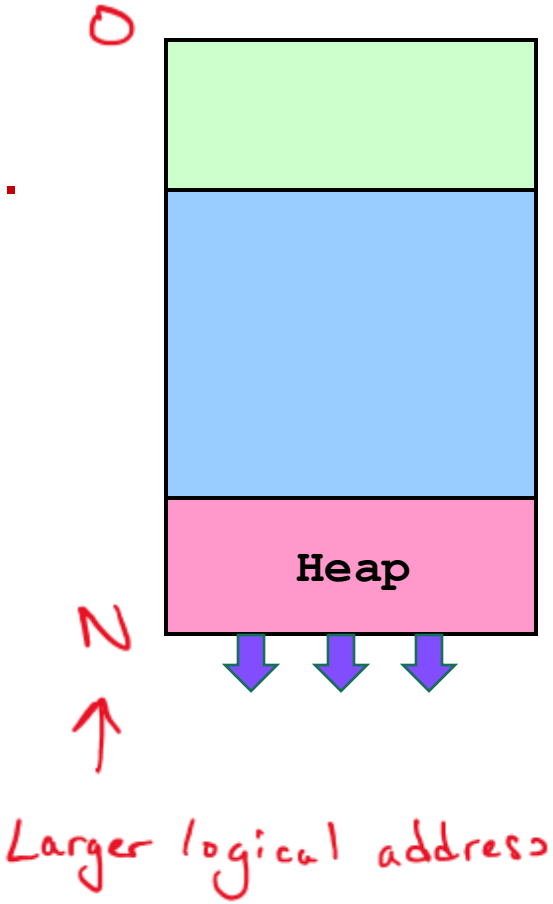
Dynamic Allocation

Question 2: Dynamic Allocation

- It is possible for a program to dynamically allocate (i.e., enlarge the memory usage) during runtime.
- For example, the system call **malloc()** in C or **new** in Java/C++ can enlarge the **heap region** of process memory.
- Discuss the OS mechanisms needed to support dynamic allocation in the following schemes:
  a) Contiguous memory allocation
     - Fixed and dynamic size partitioning
  b) Pure Paging
  c) Pure Segmentation

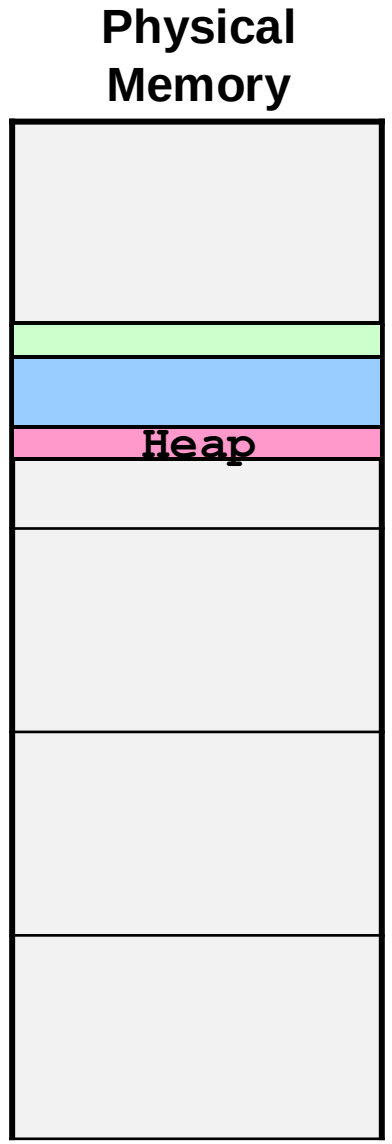
Q2: Contiguous Memory Allocation

- For simplicity, we have the heap region to be allocated at the end of the logical memory space.
- Then, we can enlarge the heap region by enlarging the partition allocated to the process.



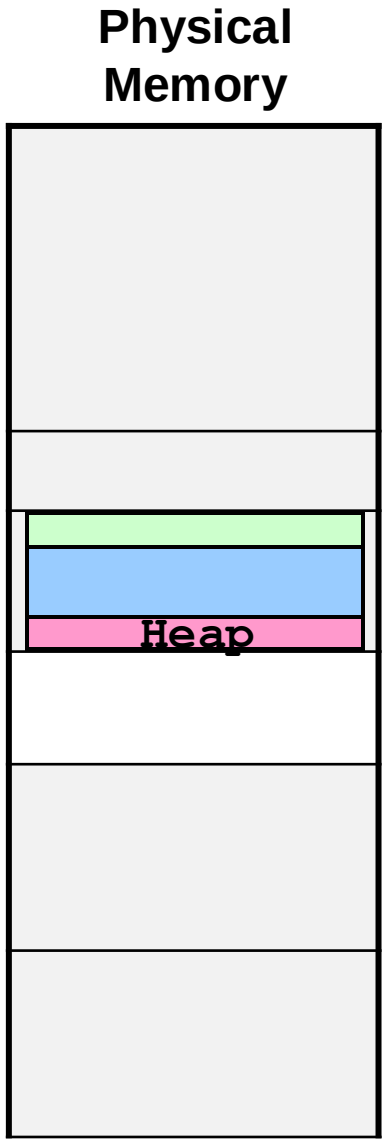
Q2: Contiguous Memory Allocation

- Fixed Partitioning
  - No extra work is needed, as heap region can simply use up the free space (the internal fragmentation) between the partition size and the actual memory size



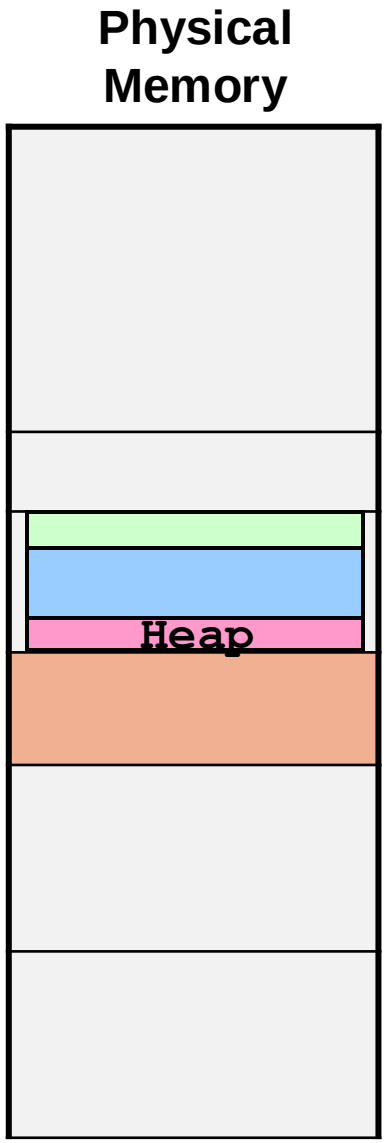
Q2: Contiguous Memory Allocation

- Dynamic partitioning:
  - If the adjacent partition is free:
    - Simply modify the partition information
    - i.e. change the length of current partition and shorten the free partition.



Q2: Contiguous Memory Allocation

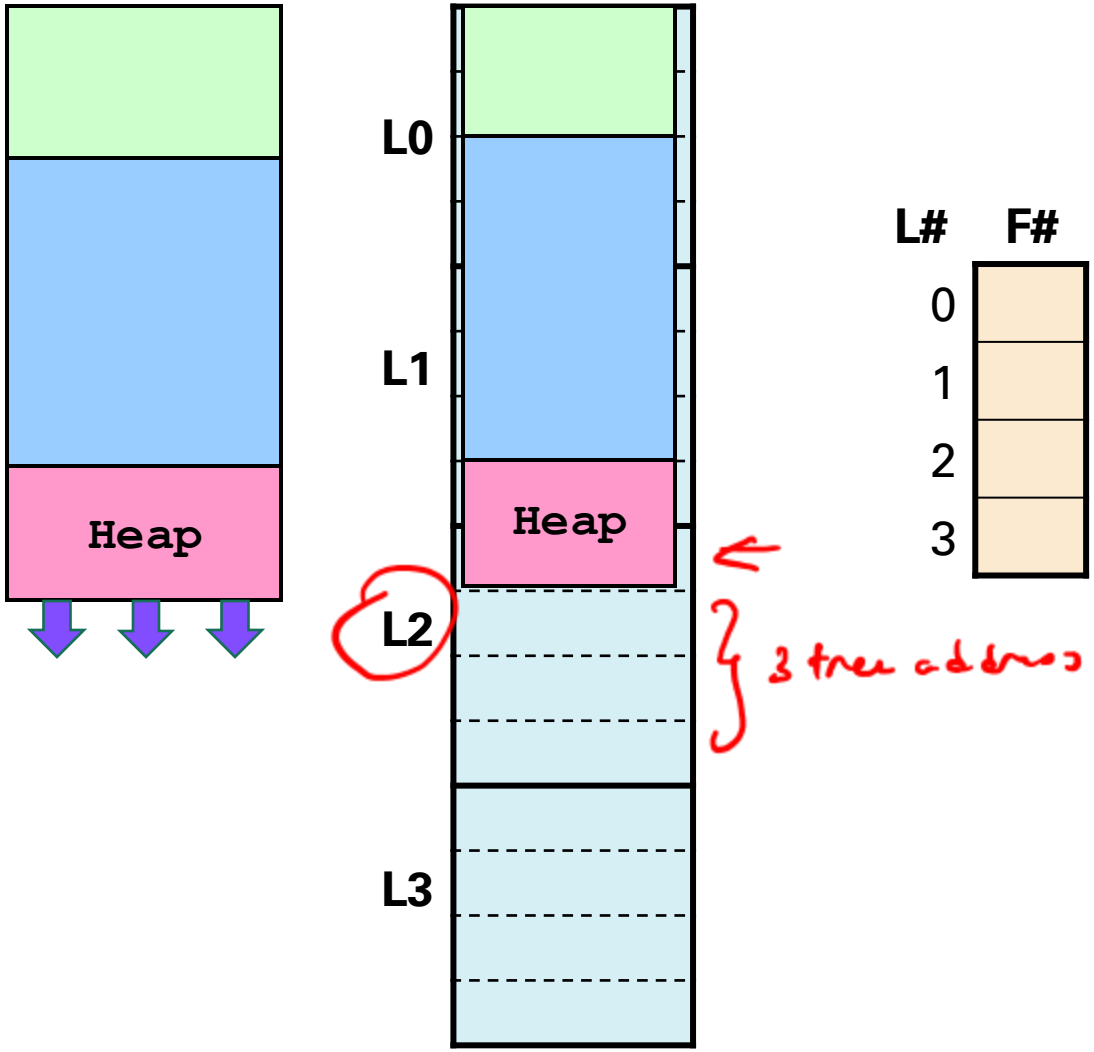
- Dynamic partitioning:
  - If adjacent partitions are occupied:
    - The current partition cannot be enlarged.
    - Relocation is required.
    - OS need to look for a large enough free partition to fit the enlarged partition.
    - Once located, the current partition is moved over.



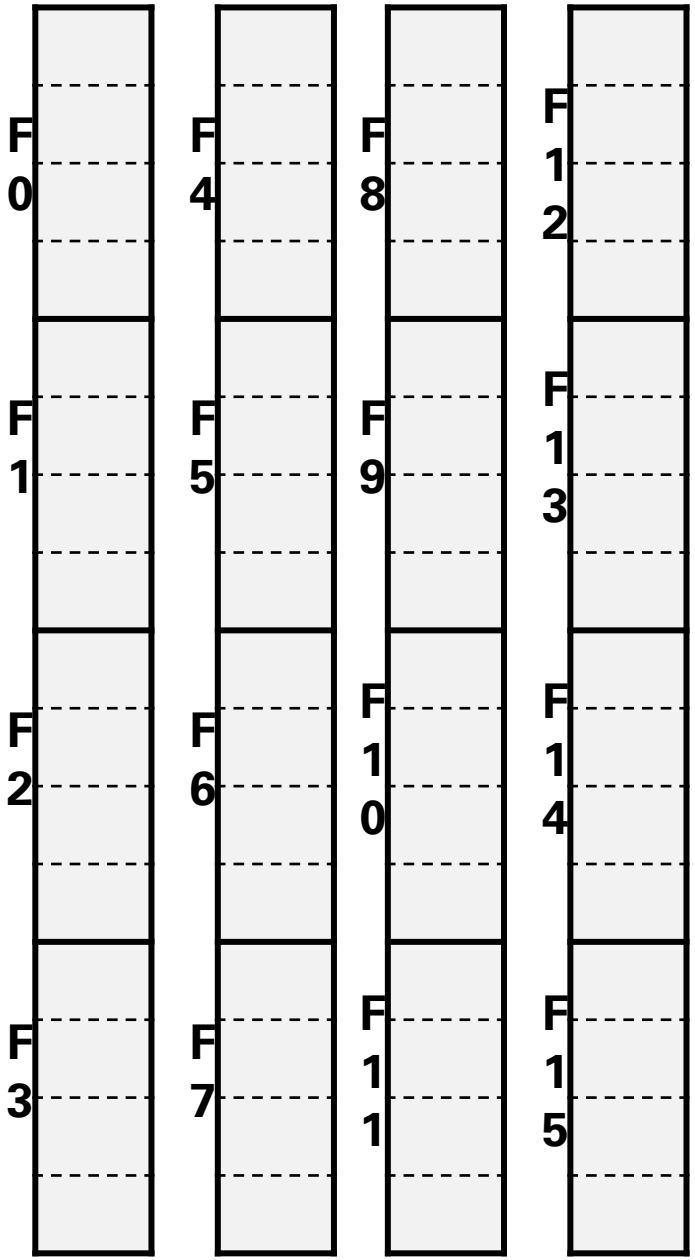
Q2: Pure Paging

- Due to internal fragmentation of the paging scheme, it is possible that the allocation can use the remaining free space in the page.
- Suppose the allocation overshoot the page boundary, then OS needs to look for a free physical frame f.
- Afterward, update the page table by changing the first invalid page table entry from invalid to valid and fill in frame number f

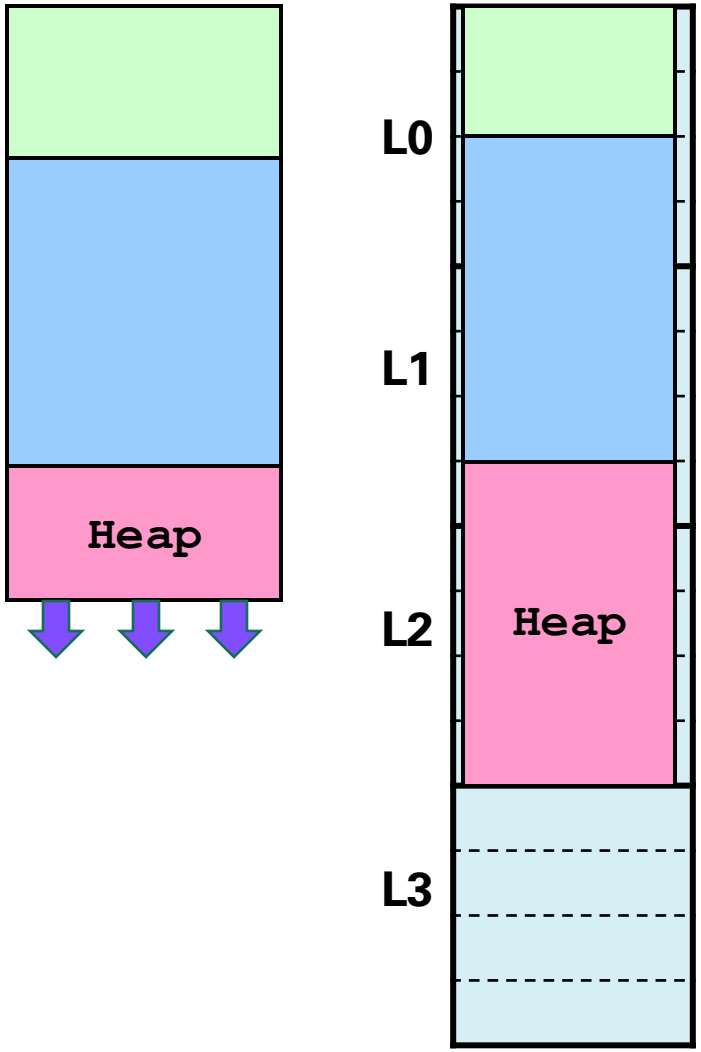
Q2: Pure Paging

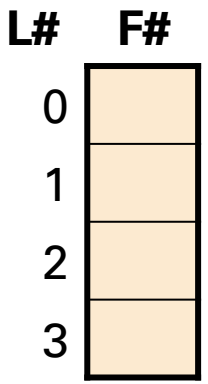


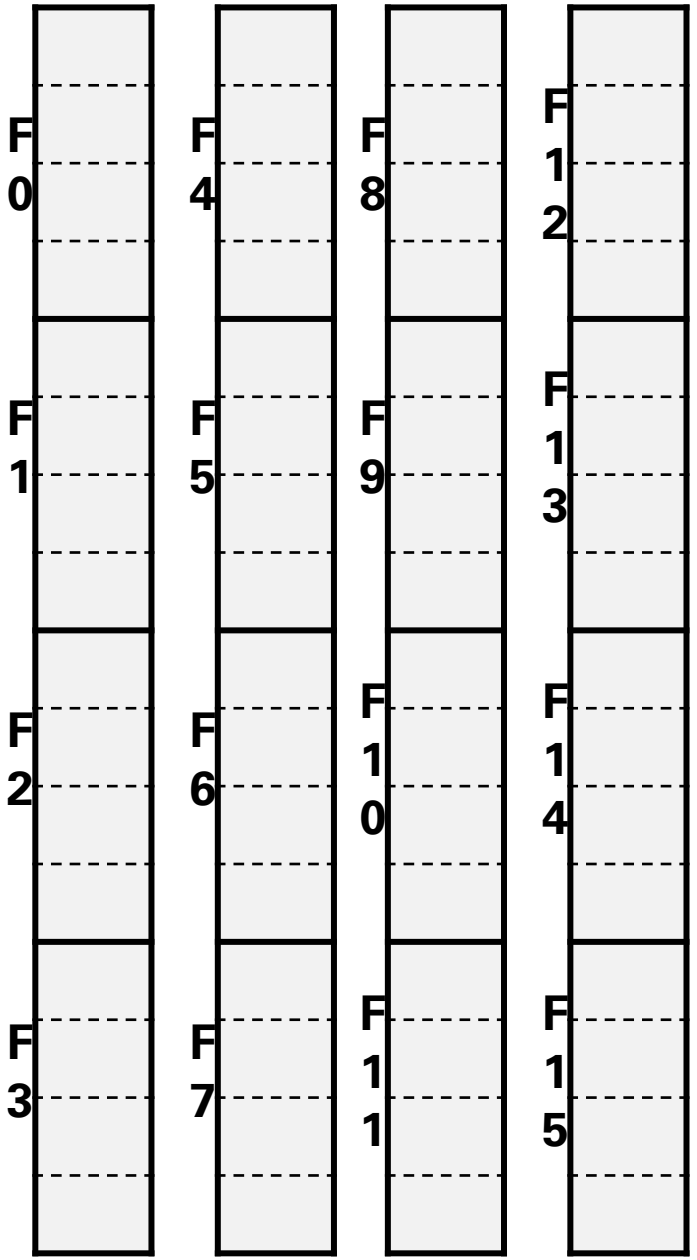
L#	F#
0	
1	
2	
3	



Q2: Pure Paging



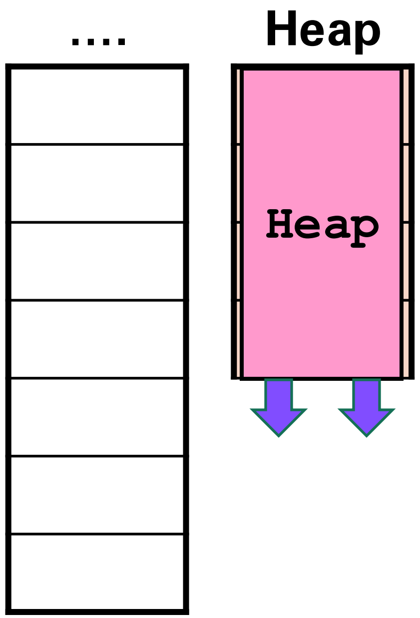




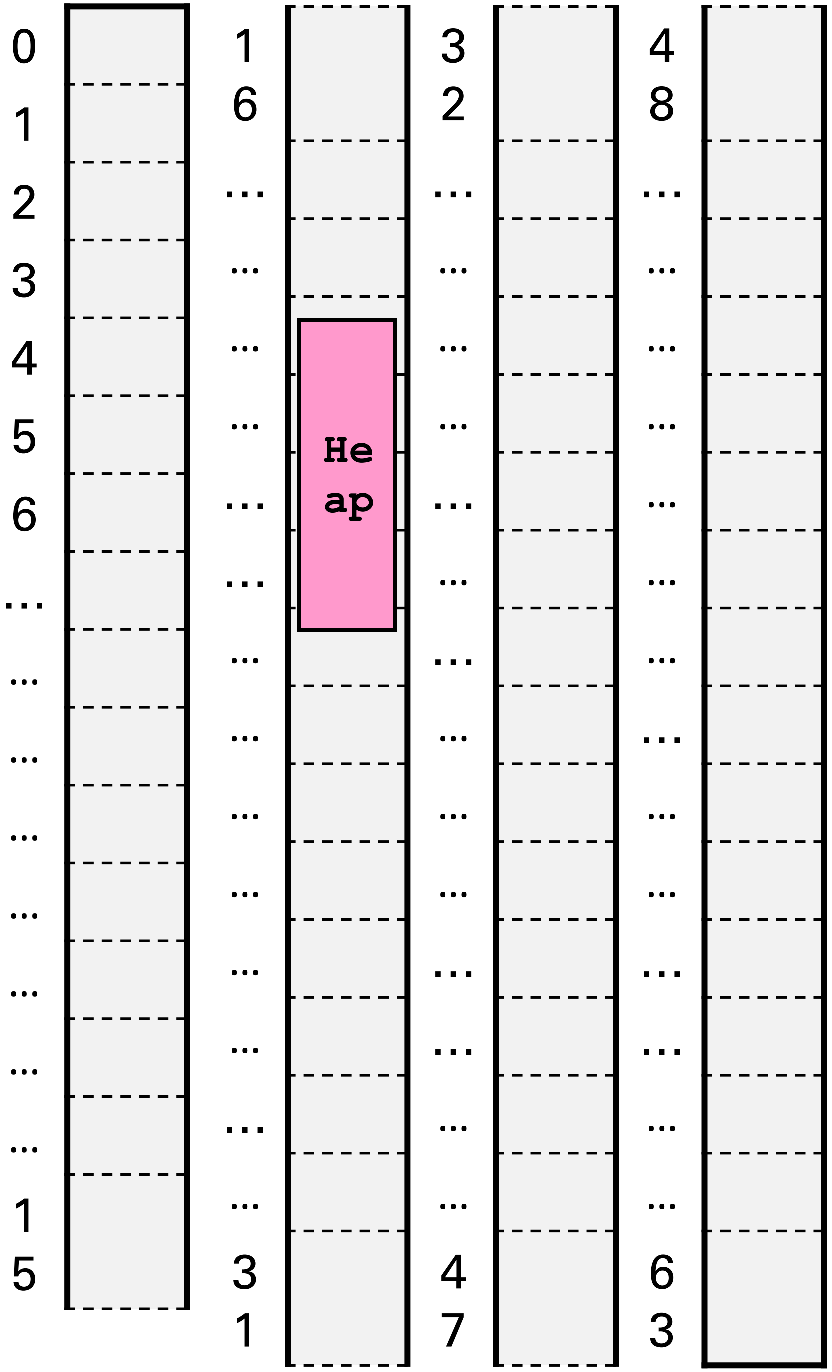
Q2: Pure Segmentation

- Idea is similar to the dynamic partitioning.
- If there is free memory at the end of the heap segment, then OS can simply update the limit of the segment and reduce the size of the affected free partition.
- If there is no free memory, then relocation is required. After relocation, both base and limit of the heap segment needs to be updated.

Q2: Pure Segmentation



Seg#	Base	Limit
0		
1		

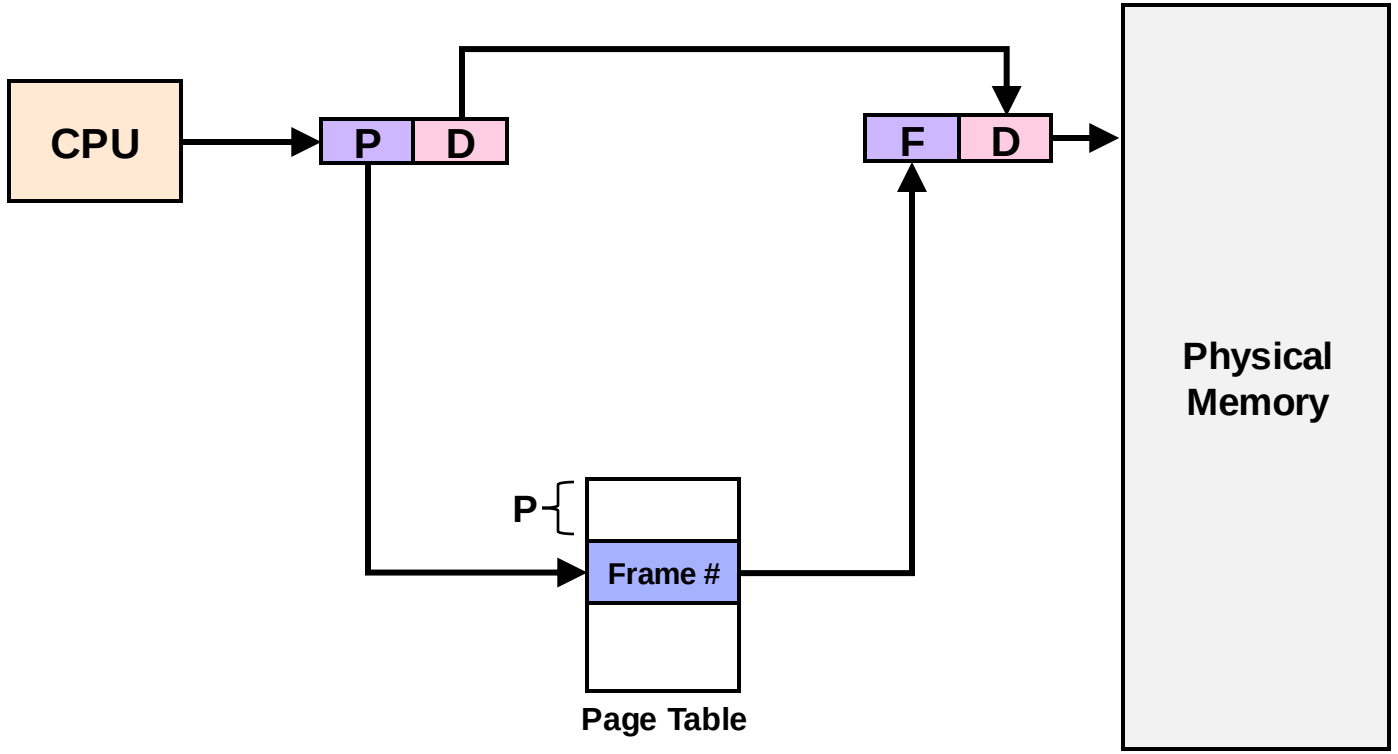


Question 3

Paging and TLB

Question 3: Paging and TLB

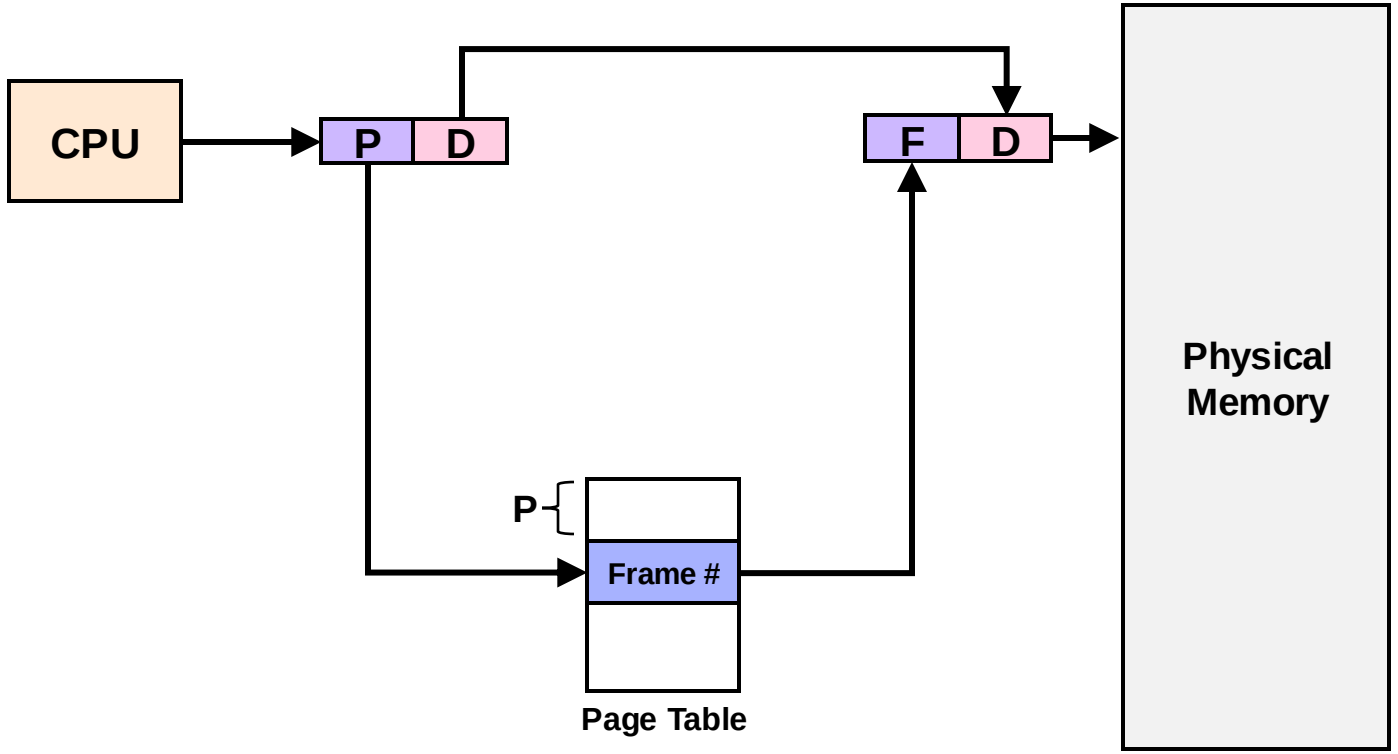
Suppose the system uses the paging scheme with the page tables entirely stored in physical memory (DRAM). The page size is 4KB, and the logical addresses are 32-bit long.



Question 3(a)

If accessing DRAM takes 50ns (nanoseconds), what is the latency of accessing a global variable of type `char`?

How many times do we need to access physical memory?



Question 3(a)

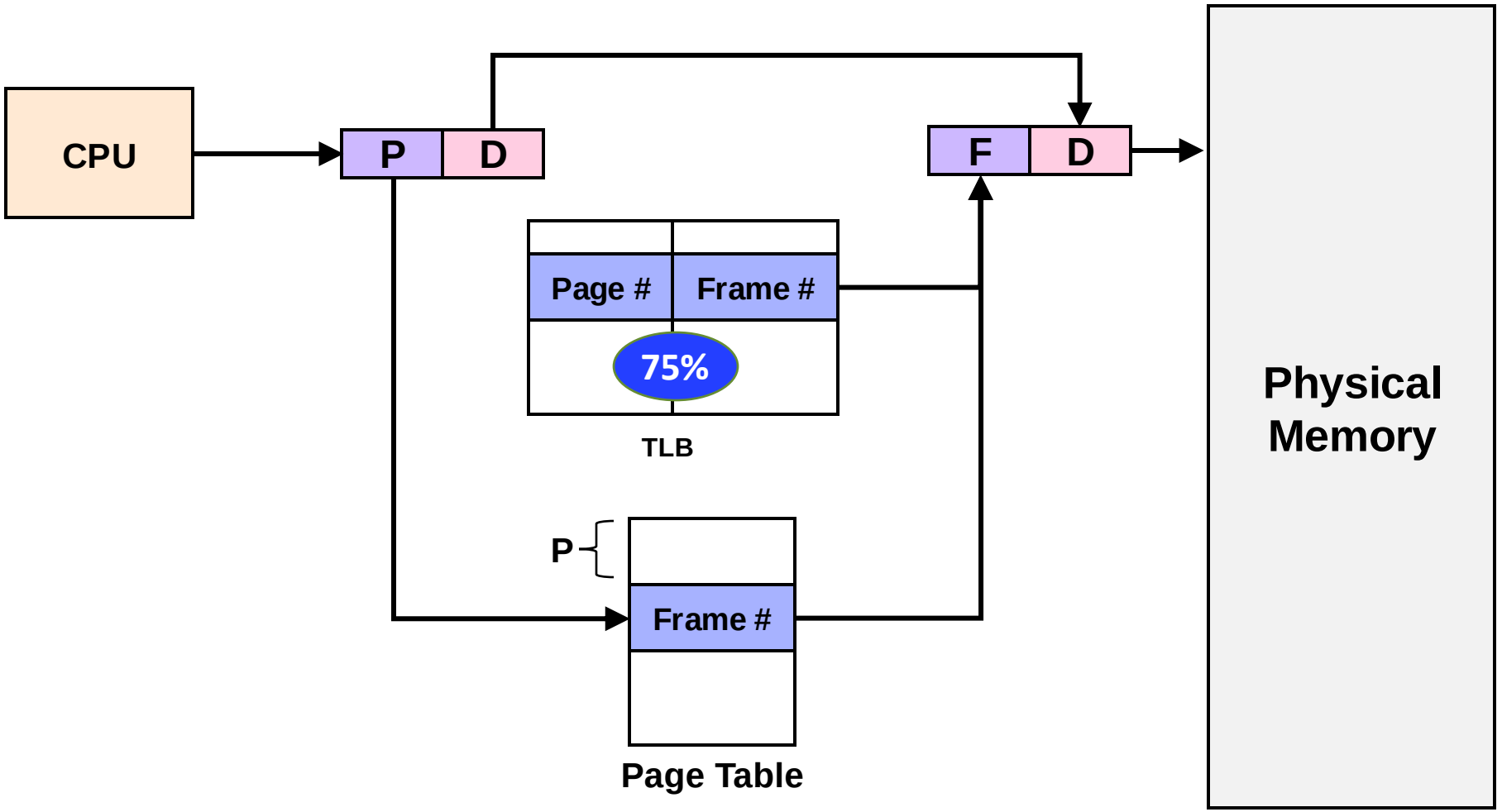
If accessing DRAM takes 50ns (nanoseconds), what is the latency of accessing a global variable of type `char`?

- To access the global variable of type char,
  - The process needs to access it's page table in memory to get the page to frame mapping.
  - Then it can access the actual item in physical memory
- 50ns (access page table) + 50ns (access actual item) = 100ns

Question 3(b)

Assuming the system uses a TLB and 75% of all page-table references hit in the TLB. What is the average memory access time? You can assume that looking up a page table entry in TLB takes negligible time.

Question 3(b)


CPU
P
D
F
D
Page #
Frame #
75%
TLB
Physical Memory
P
Frame #
Page Table

Question 3(b)

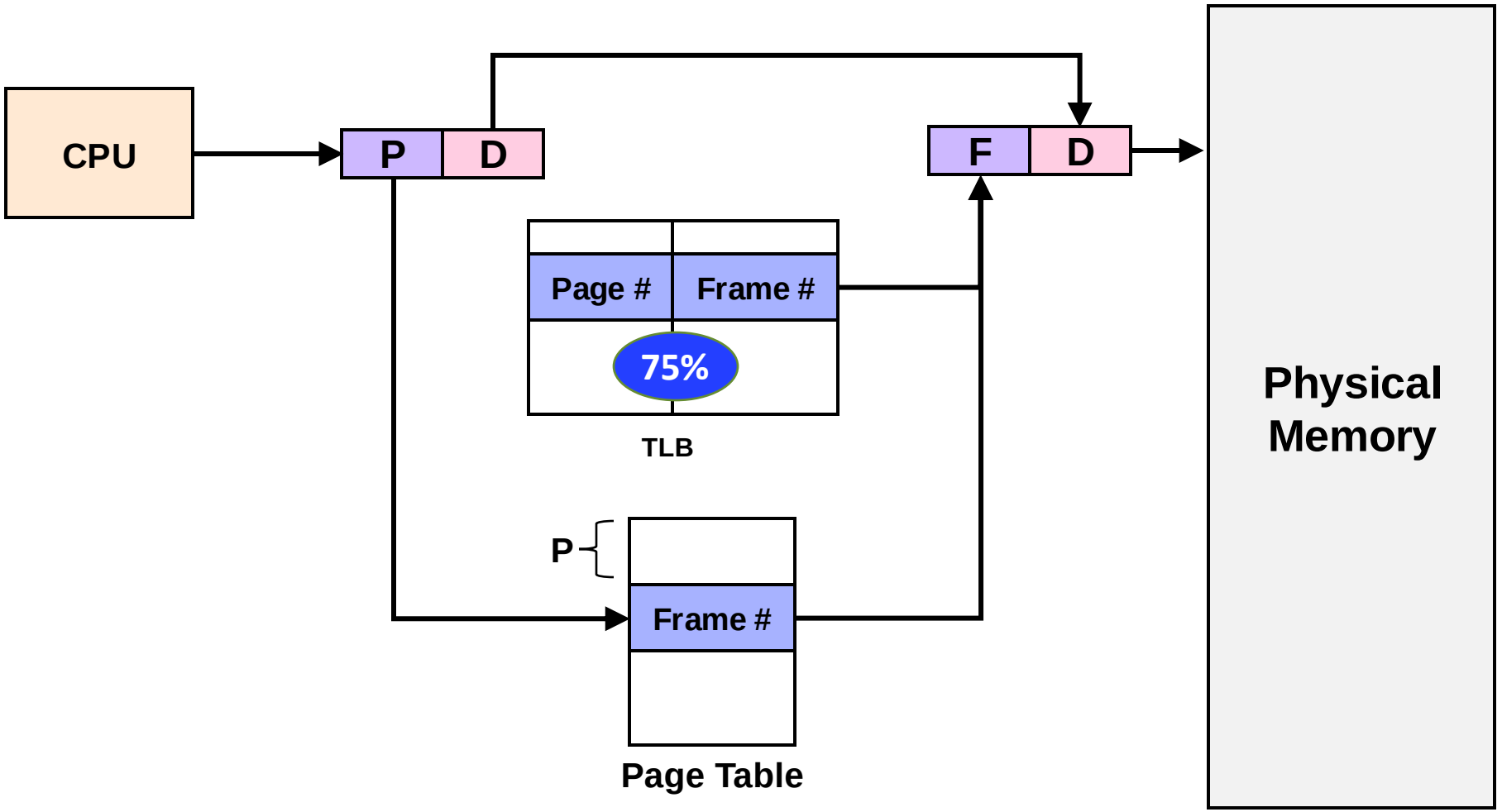
Assuming the system uses a TLB and 75% of all page-table references hit in the TLB. What is the average memory access time? You can assume that looking up a page table entry in TLB takes negligible time.

- TLB Hit: 50ns (Access TLB, Physical Memory)
- TLB Miss: 100ns (Access TLB, Page Table, Physical Memory)
- 0.75 * 50ns + (1-0.75)*100ns = 62.5ns

Question 3(c)

How many entries does a TLB need to have to achieve a hit ratio of 75%? Assume the program generates logical memory addresses uniformly at random.

Question 3(c)


CPU
P
D
F
D
Page #
Frame #
75%
TLB
P
Frame #
Page Table
Physical Memory

Question 3(c)

How many entries does a TLB need to have to achieve a hit ratio of 75%? Assume the program generates logical memory addresses uniformly at random.

- Page Size = 4KB = 2^{12} Bytes
- Total number of page table entries = $2^{(32-12)} = 2^{20}$ = 1M entries
- If we assume the memory access are uniformly distributed between all the possible pages, then we need at least 1M*0.75 = 786,432 entries in the TLB.

1048576

Question 3(c)

- Do you think a TLB in an actual machine is this large?
  - It seems impossible to have a TLB this large to guarantee a high hit-rate.
- If not, then how is it possible to achieve a high TLB-hit rate?
  - Despite TLB being quite small (32 to 1024 entries) in real-world machines, TLB hit ratio is still very high (commonly 99% hit rate) in actual usage.
  - This is because memory access is **not uniformly distributed.**
  - This can be understood with the aid of locality principle:
    - Temporal Locality: Memory address which is used is likely to be used again
    - Spatial Locality: Memory addresses close to a used address is likely to be used
    - it is more likely to have repeated memory accesses to same (temporal locality) or different (spatial locality) parts of the same memory page in a time interval rather than uniformly spread accesses.

Question 3(c)

- For example, after accessing the 1st instruction, it its likely the process access the 2nd instruction in the same logical page.
- Another way to think of this is that we are running a program on our computer.
- It is likely that the page to frame mappings of the frequently used instructions are loaded on the TLB.

Page 0	0 1 2 3	1st Instruction 2nd Instruction 3rd Instruction 4th Instruction
Page 1	4 5 6 7	5th Instruction 6th Instruction <empty> <empty>
Page 3	8 9 10 11	1st Data Word 2nd Data Word 3rd Data Word 4th Data Word
Page 4	12 13 14 15	5th Data Word <empty> <empty> <empty>

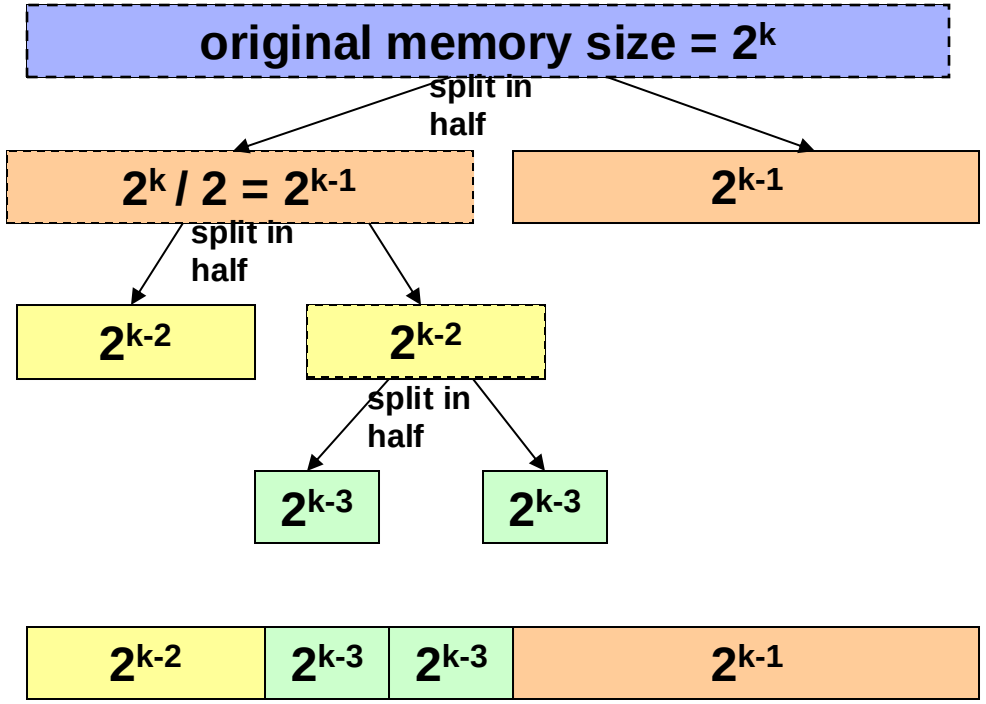
Question 4

Fragmentation in Buddy Allocator

Question 4

i. Calculate the average amount of memory capacity lost to internal fragmentation in a system that uses the Buddy allocator.

ii. Can the Buddy allocator suffer from external fragmentation?



Question 4

Calculate the average amount of memory capacity lost to internal fragmentation in a system that uses the Buddy allocator.

- For an allocation request of N bytes:
  - min fragmentation = 0% (exact fit, N=2^k),
  - max fragmentation = ~50% (N=$2^{k-1}+1$),
  - on average 25%.
- Note that internal fragmentation > 50% is not possible.
  - Would have been allocated a smaller free block of (N=2^{k-1})

Question 4

Can the Buddy allocator suffer from external fragmentation?

External fragmentation can still happen!

128KB A (100/128)	128KB B (98/128)	256KB C (240/256)	256KB D (225/256)	128KB E (114/128)	128KB Free

Deallocate B

128KB A (100/128)	128KB FREE	256KB C (240/256)	256KB D (225/256)	128KB E (114/128)	128KB Free

Suppose we want to allocate Process F, that is 200KB, we are unable to do so.

END OF TUTORIAL