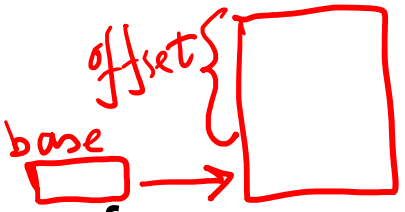
High Performance Computer Architecture (CS60003)

Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Spring 2022

Virtual Memory

Virtual vs Physical Memory



- Consider a 32-bit architecture with size of programmable registers 32 bits.

$2^{30} \times 2^2 = 4$ GB

1 kB = 2^{10} B.
1 MB = 2^{20} B
1 GB = 2^{30} B.

- Memory references which are generated as the sum of a register and a small constant (the offset) are also 32 bits.
- The addressing space is thus 2^{32} bytes, or 4 GB.

Tera
Peta
ExaBytes
ZettaBytes

Todays servers, laptops, machines have such memory capacities.

However, applications will have data sets larger than 4GB. Hence, we have 64-bit ISAs, ie. registers with 64 bits. Clearly, this range of addresses is much too large for main memory capabilities!

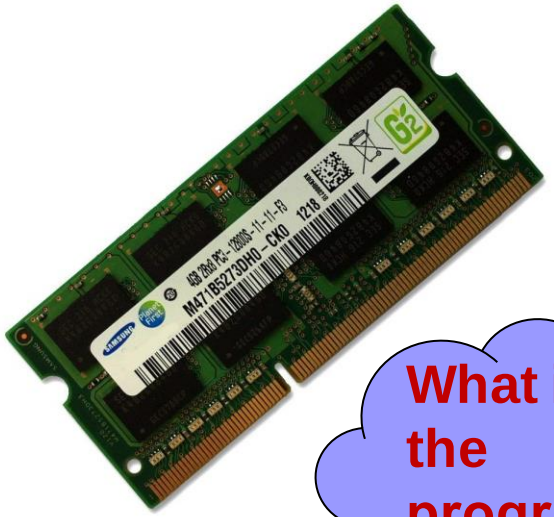
Gap between addressing space and main memory

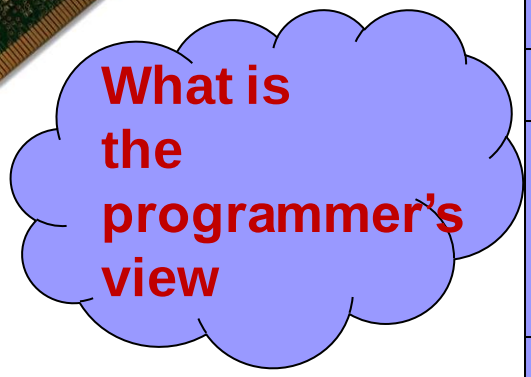
- This was also present in older machines.
- A single program ran on the whole machine.
- Even, there was not enough main memory for a given program and its data.
- Programs and data were statically partitioned into overlays so that parts of the program or data that were not used simultaneously, share the same memory location.
- I/O, which was a much slower process than the computations, became a bottleneck.
- This lead to multiprogramming, where more than one program is resident in main memory at the same time.
- When the I/O is needed by the executing program, it relinquishes the CPU to another program.

Questions about the memory management

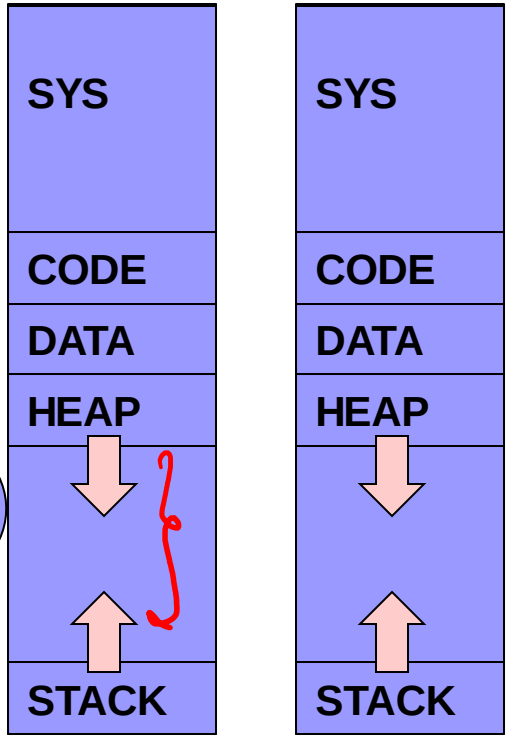
- How and where are the program loaded in main memory?
- How does one program ask for more main memory, if required?
- How is one program protected from another? Ie. What prevents one program to access the data of another program?

Programmer's view of memory



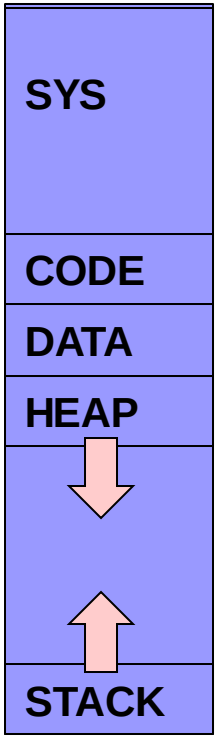


App 1



App 2

App 3.



- User does not bother about the memory remaining between the heap and the stack.
- Typically, there are multiple programs running. Each of them sees the same address space, as if it has the entire memory.

Virtual Memory

- It helps to bridge this gap between programmer's view and the actual hardware memory.
- Programs are compiled and linked as if they could address the whole addressing space.
- Addresses generated by the CPU are virtual addresses that is translated into real or physical addresses when referencing the memory.
- In the early 1960s computer scientists at the University of Manchester introduced the term virtual memory and implemented using a paging system.

Quiz 1

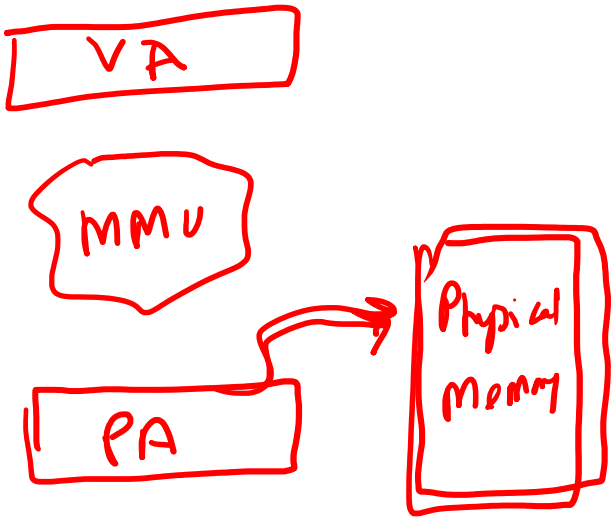
- A computer has 16 active applications each with a 32-bit address space (4 GB). What does the system actually have?
  - ☐ 2 2 GB memory modules
  - ☐ 4 4 GB memory modules
  - ☐ 8 8 GB memory modules
  - ☐ 1 16 GB memory modules

Quiz 1

- A computer has 16 active applications each with a 32-bit address space (4 GB). What does the system actually have?
  - 2 2 GB memory modules
  - 4 4 GB memory modules
  - 8 8 GB memory modules
  - 1 16 GB memory modules
- Any of the above is possible.
- Virtual address decouples, what the application accesses and what the memory system has.

Processor's View of Memory

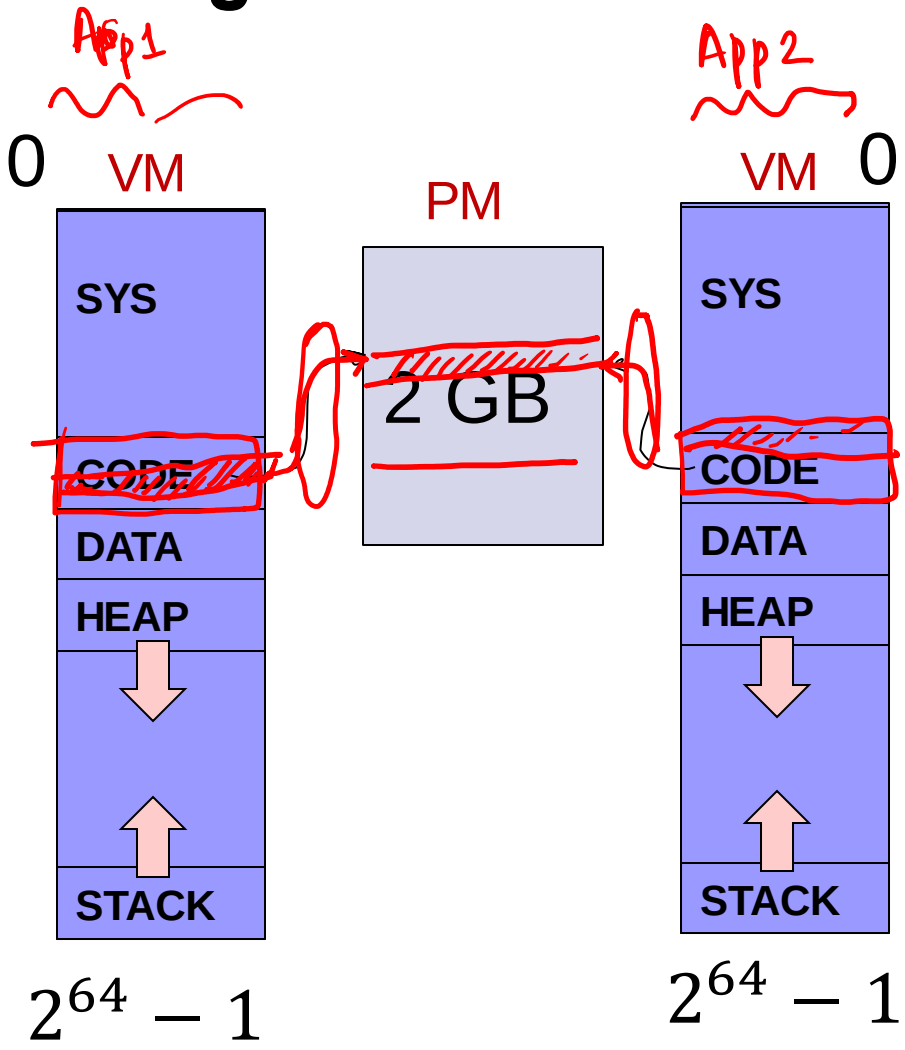
- Physical memory is the actual memory
- Sometimes less than say 4GB
- Almost never: 4GB/Process
- Never: 16 Exabytes/Process



The physical memory space is usually less than what we have in the virtual address space.

There is a 1:1 mapping between bytes/words in physical memory, and the physical address.

Programmer's View of Memory



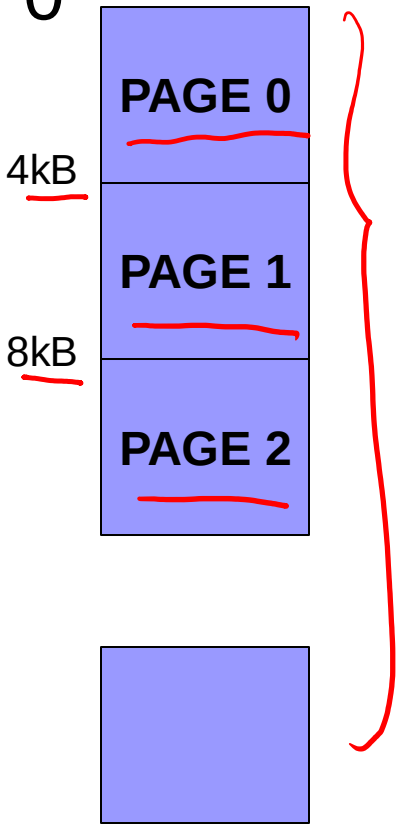
User has a perception of a large memory, the huge space between the heap and the memory.

How do we figure out where the virtual address is getting to? From 2 different codes, they may map to the same location in the PM

- If two programs are sharing data, the codes are mapped to same location
- In case of code sharing
- Their Virtual addresses need not be the same.

Virtual to Physical Address Mapping

- We cannot develop a look up as the VA space is huge

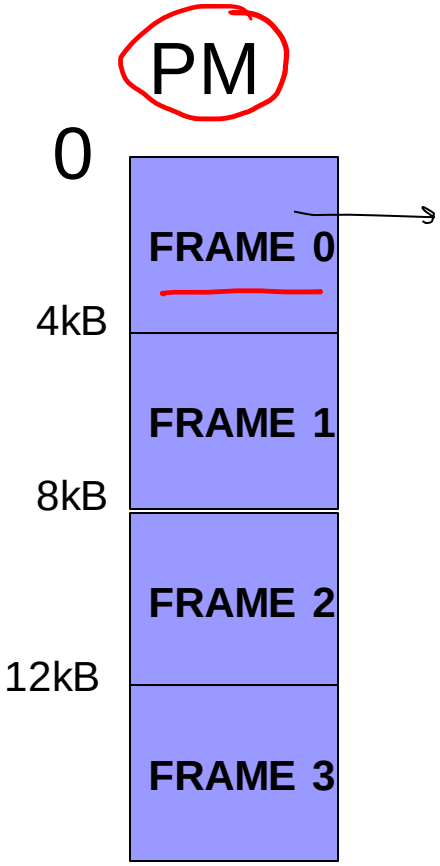


Programmer's view of memory is divided into equal size chunks called pages.

Each page size is aligned to page size.

Each page is of say 4kB and starts at boundaries of 0-4-8 kB, etc.

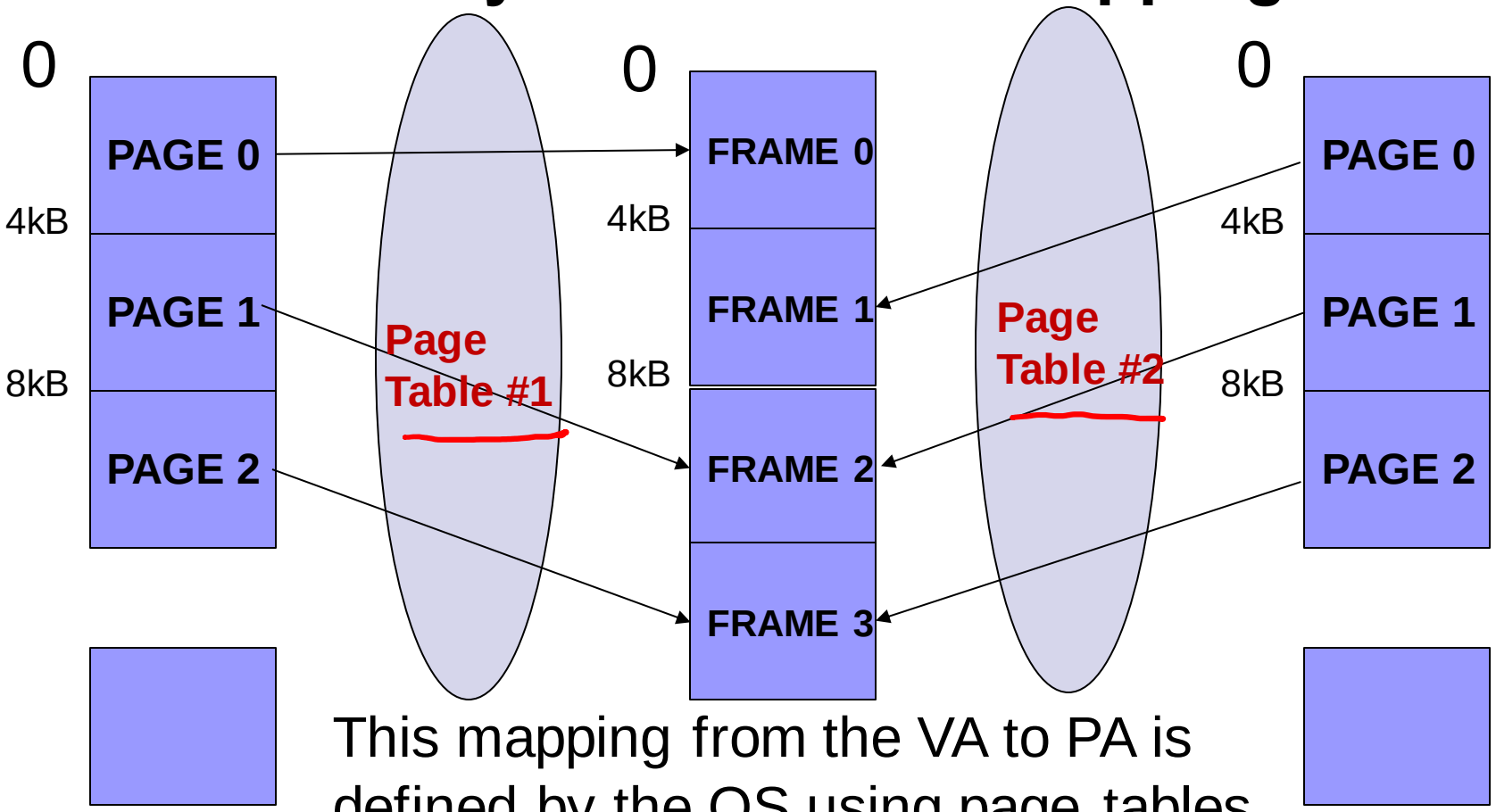
Virtual to Physical Address Mapping



Slots/frames that can hold pages.

So, the PM acts like a cache for the VM. It has certain number of locations where we can store the pages.

Virtual to Physical Address Mapping



This mapping from the VA to PA is defined by the OS using page tables

It tells for each page in a process where it is actually mapped in the main memory.

Quiz 2

- Physical Memory is 2GB
- Virtual Memory is 4GB
- Page Size is 4kB
- How many page frames?
- How many entries are there in each page table?

$\frac{2GB}{4kB} = \frac{2 \times 2^{30}}{2^2 \times 2^{10}} = 2^{19}.$

$\frac{4GB}{4kB} = 2^{20}.$

Quiz 2

- Physical Memory is 2GB
- Virtual Memory is 4GB
- Page Size is 4kB
- How many page frames?
- How many entries are there in each page table?

Frame size is same as page size.

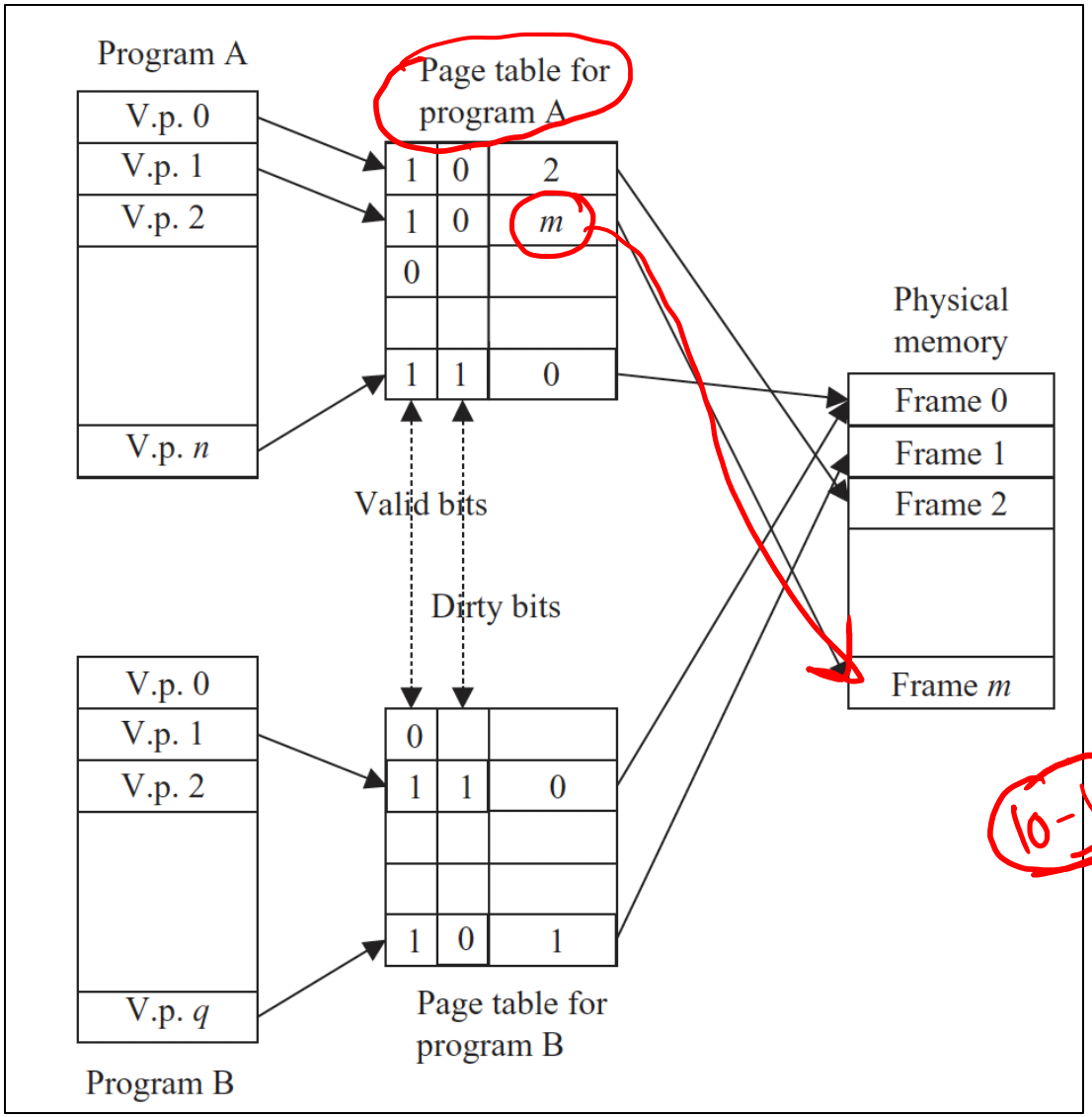
$$\frac{2\text{GB}}{4\text{kB}} = \frac{2 \times 2^{30}}{2^2 \times 2^{10}} = 2^{19}$$

We need one entry per page.
So, we need to compute number of pages.
Thus number of entries per page table,

$$\frac{4GB}{4kB} = \frac{4 \times 2^{30}}{4 \times 2^{10}} = 2^{20}$$

This number of entries are needed for the page table of every process!

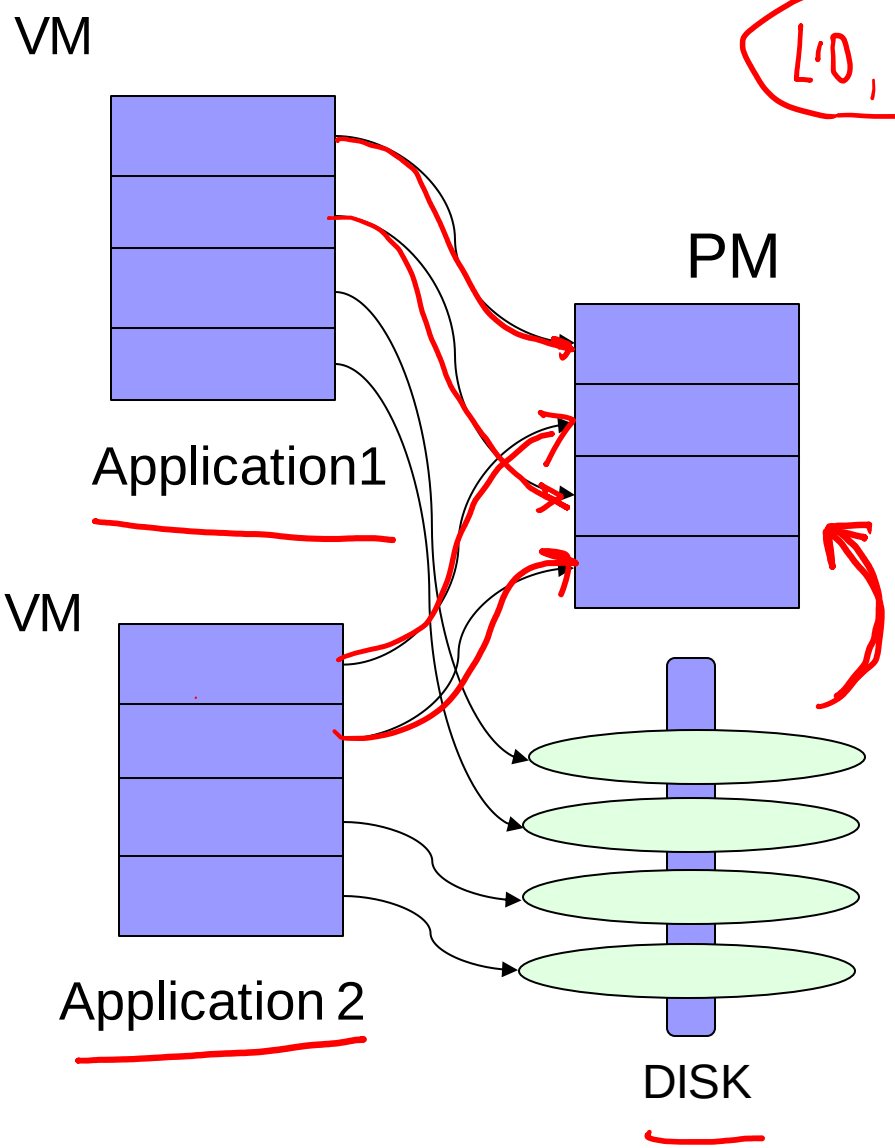
Overview of Paging System



Key Points:

- Virtual address space can be larger than physical address space.
- Not all of the programs and its data can be in the main memory at a time.
- Physical memory can be shared between programs.
- Auxiliary bits:
  - Valid: whether the mapping is current or not.
  - Dirty: whether the page has been modified since bringing into main memory.

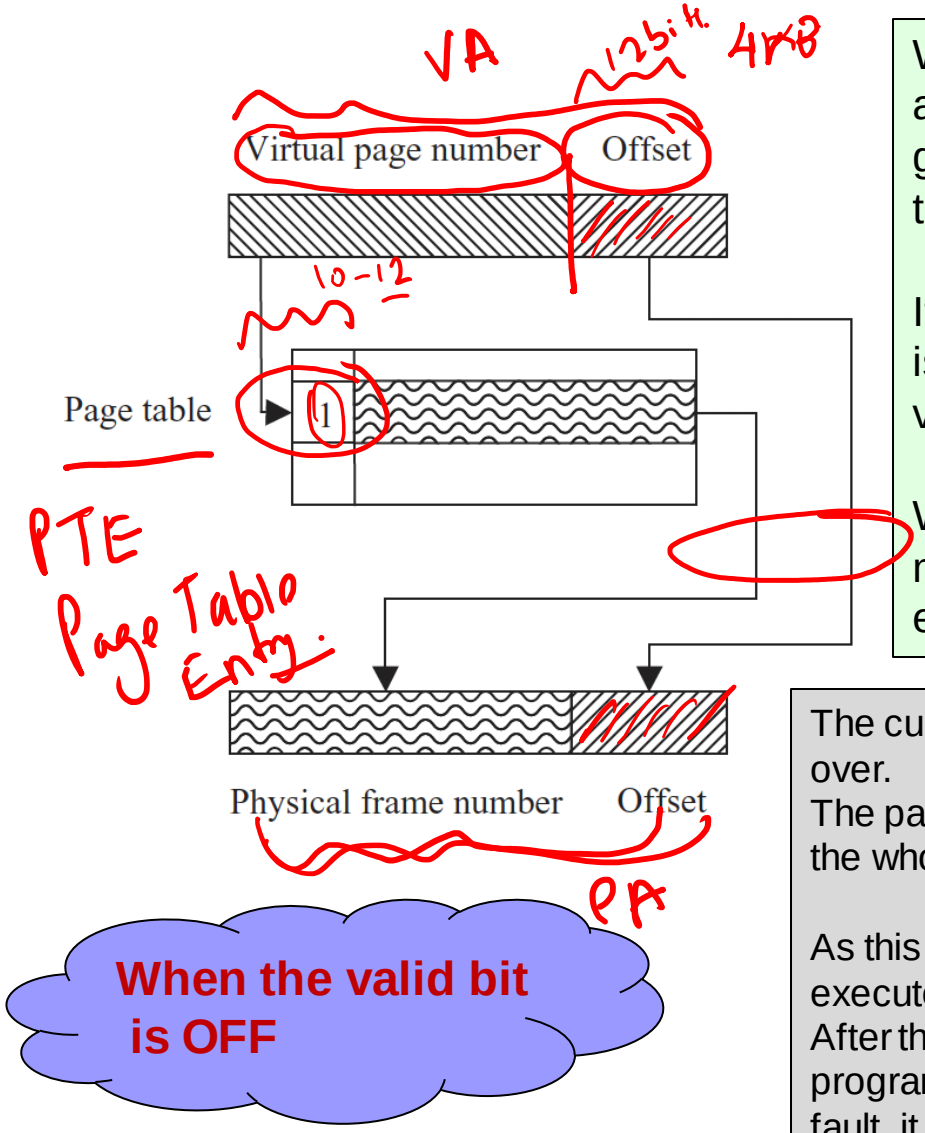
Where did the extra memory go?



These pages cannot be accessed by the processor, as it can only access main memory through loads and stores.

So, if the processor wants to access these pages, the processor needs to bring them from the disk, before they can be accessed.

Translation Architecture and the OS



When a program executes and needs to access memory, the virtual address that is generated is translated through the page table.

If the valid bit of the PTE (page table entry) is ON, the translation will result in the virtual address.

When OFF, the virtual page is not in the memory, we have a page fault, which is an exception.

When the valid bit is OFF

The current program is suspended, and the OS will take over.
The page fault handler will initiate the I/O read to bring the whole virtual page from disk.

As this often takes several milli-seconds, time enough to execute billion instructions, a context switch will occur. After the OS saves the process state of the faulting program and initiates the I/O process to handle the page fault, it will give control of the CPU to another program by restoring the process state of the later.

Quiz 3

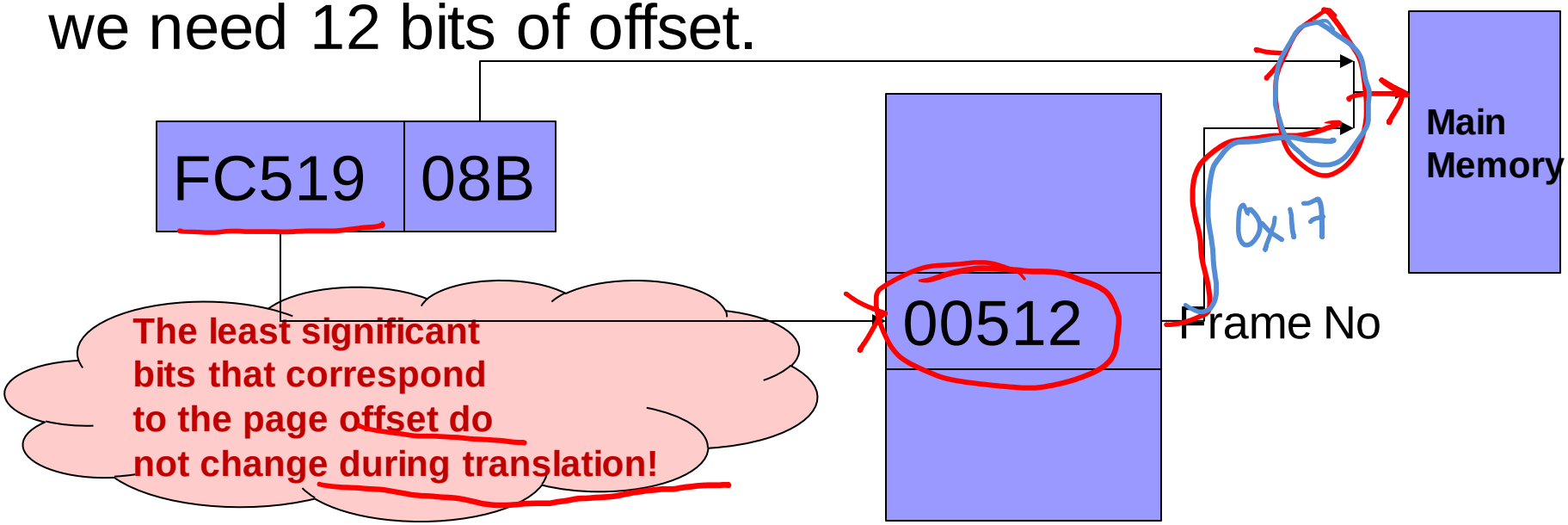
- What is the PA for the VA=FC51908B?
  - Assume a 4kB page size.

Quiz 3

- What is the PA for the VA=FC51908B?
  - Assume a 4kB page size.

VIRTUAL PAGE#	PAGE OFFSET

If we have a 4kB page size, ie, $4 \times 2^{10} = 2^{12} bytes$, we need 12 bits of offset.



Quiz 4

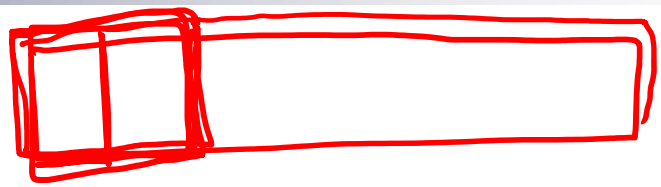
Consider,
16 bit virtual address space
20 bit physical address space

00	0x1F
01	0x3F
10	0x23
11	0x17

Write down the PA for the following VAs:
0xF0F0 => ?

0x001F => ?

Quiz 4



Consider,
16 bit virtual address space
20 bit physical address space

00	0x1F
01	0x3F
10	0x23
11	0x17

Page Table

Write down the PA for the following VAs:

0xF0F0 => ?

1111 0000 1111 0000

=> 0001 0111 11 0000 1111 0000

0101 1111 0000 1111 0000=>5F0F0 (PA)

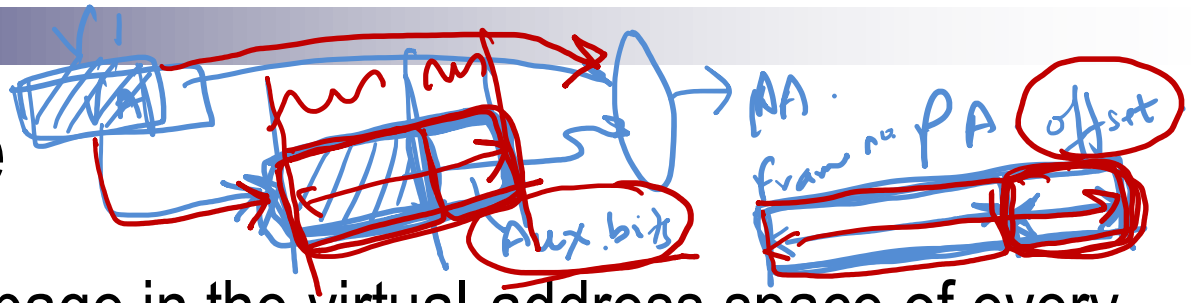
0x001F => ?

0000 0000 0001 1111

=>0001 1111 00 0000 0001 1111

0111 1100 0000 0001 1111=>7C01F

Flat Page Table



- There is 1 entry per page in the virtual address space of every process.
- The entry contains the frame no. plus few auxiliary bits, which tells if the page is accessible.
- The entry size contains the frame no. which is most of the bits in the physical address. In place of the page offset, it has the extra bits.
  - So, roughly if we have a 32/64 bit physical address, the entry size in the page table is also 32/64 bits.
- So,

$$Size\ of\ a\ Page\ Table = \frac{Virtual\ Memory}{Page\ Size} \times Size\ of\ entry$$

Example

- Virtual address size for 32 bit addressing=4 GB, Page Size=4kB, PTE=4 bytes.
  - Page table Size=$\frac{4\ GB}{4\ kB} \times 4Bytes = 4MB$ per process.
  - A process might actually be using less than 4MB of virtual memory. Most of the pages are hence unused in such a layout.
- For 64 bit virtual address space, virtual memory=2^{64}.
  - Then page table size=$\frac{2^{64}}{2^2 \times 2^{10}} \times 4B = \frac{2^{64}}{2^2 \times 2^{10} \times 2^{20}} \times 4MB = 2^{32} \times 4MB$
  - This is most likely more than the physical memory
  - The page table cannot fit in memory because of the large virtual address space.

Quiz 5

64bits

- Assume a page table has 8Bytes/entry ie. assume a 64-bit physical address space.
- The page size is 4kB
- There are 2 processes in the system
- The physical memory is 2GB
- Assume a 32 bit virtual address space.
- Process 1 uses only 1MB of memory
- Process 2 uses only 1 GB of memory
- Compute size of the flat page table.

$\frac{2^{32}}{4KB} \times 8$

$= 2^{23}$

$= 8 \times 2^{20}$

$= 8\,MB$

16MB

Quiz 5

- Assume a page table has 8Bytes/entry ie. assume a 64-bit physical address space.
- The page size is 4kB
- There are 2 processes in the system
- The physical memory is 2GB
- Assume a 32 bit virtual address space.
- Process 1 uses only 1MB of memory
- Process 2 uses only 1 GB of memory
- Compute size of the flat page table.

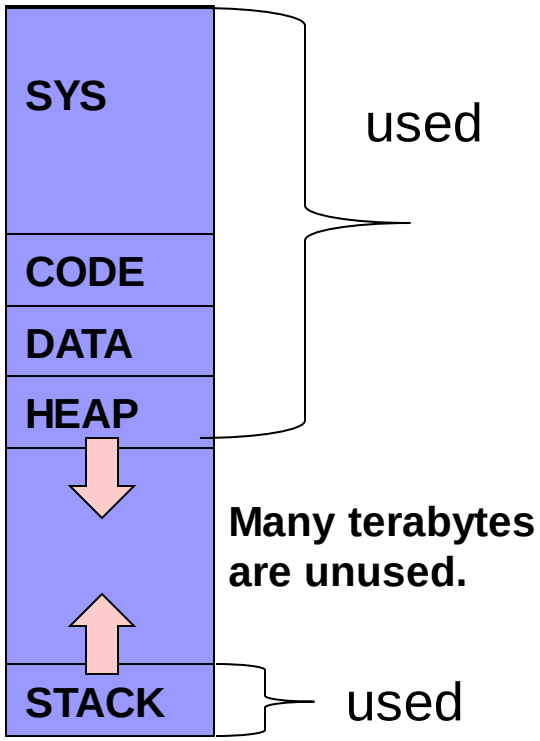
The process details do not matter.

Size=$2 \times \frac{2^{32}B}{4kB} \times 8B = 16 \times 2^{20}B = 16MB$

Multi-Level Page Table

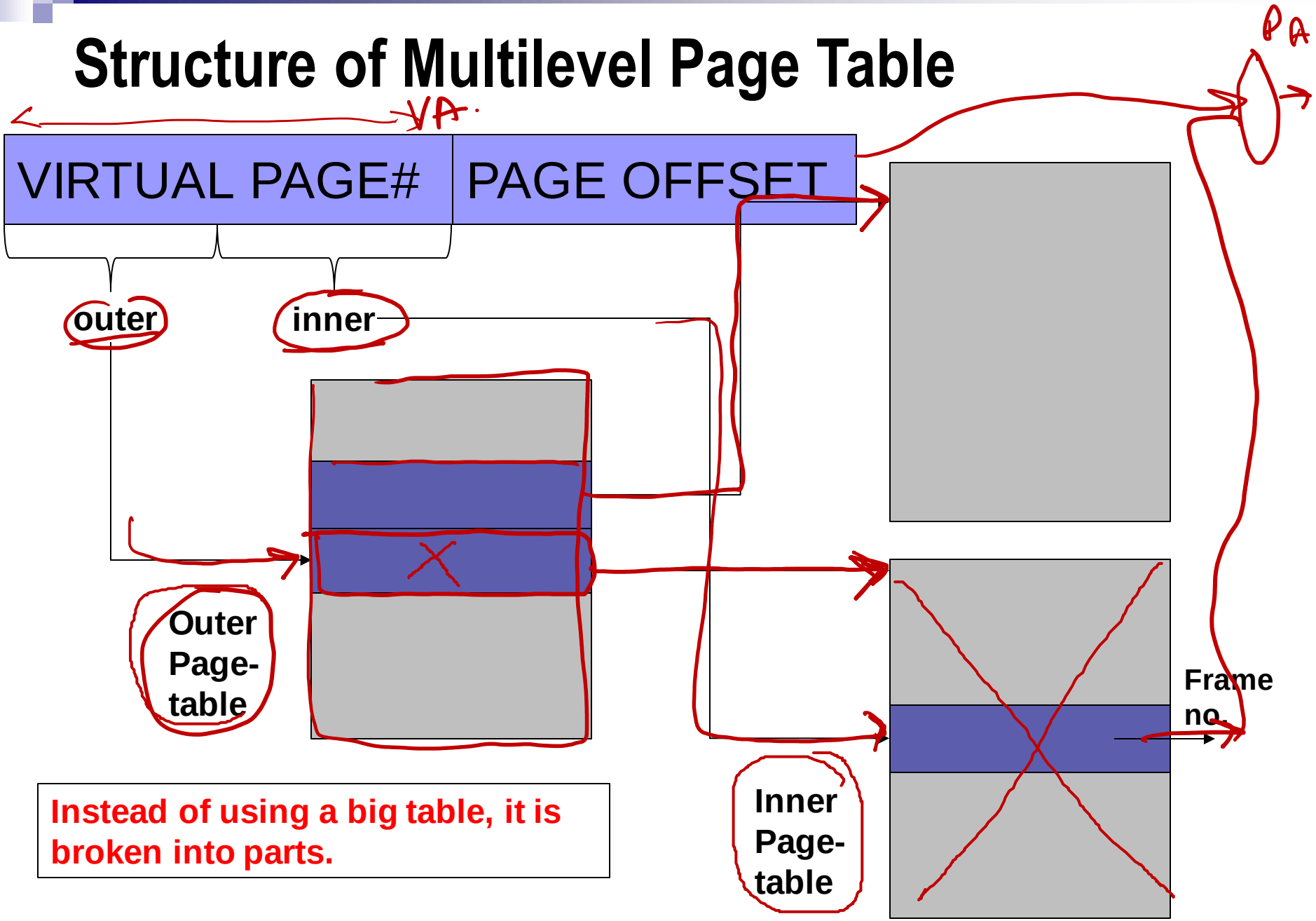
- A flat page table is wasteful!
  - Size proportional to the address space which can be addressed for an application.
  - Does not depend on the actual address space being accessed by an application.
  - So, for 64 bit VA, the size is too big for a flat page table.

Idea of Multiple Page Table



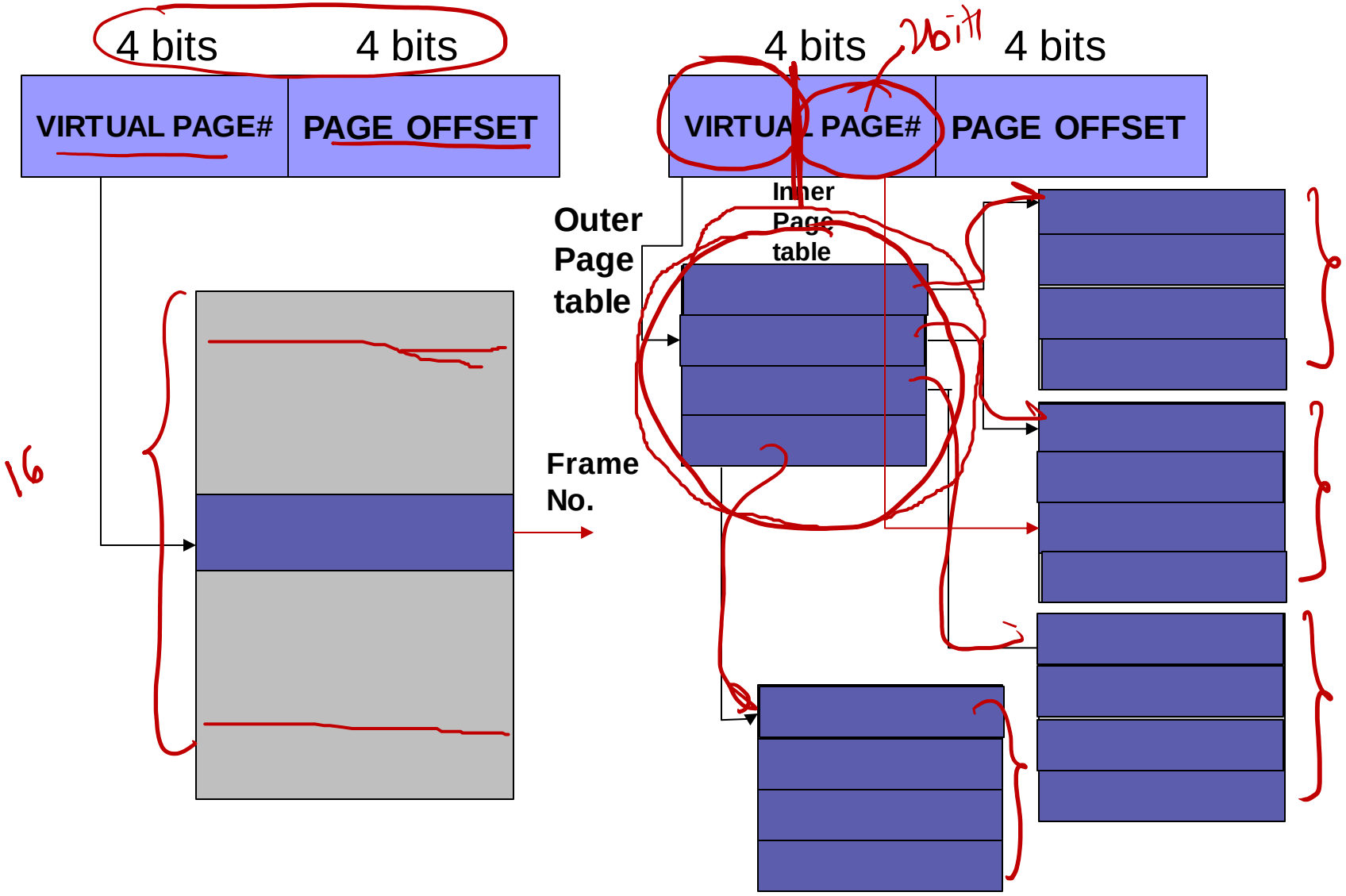
- Uses idea of flat table.
- But avoids tables for the unused portions.
- It uses the VA to index the table as usual.
- But the table entries are maintained only for used locations.

Structure of Multilevel Page Table



Instead of using a big table, it is broken into parts.

Example of a 2 Level Page Table

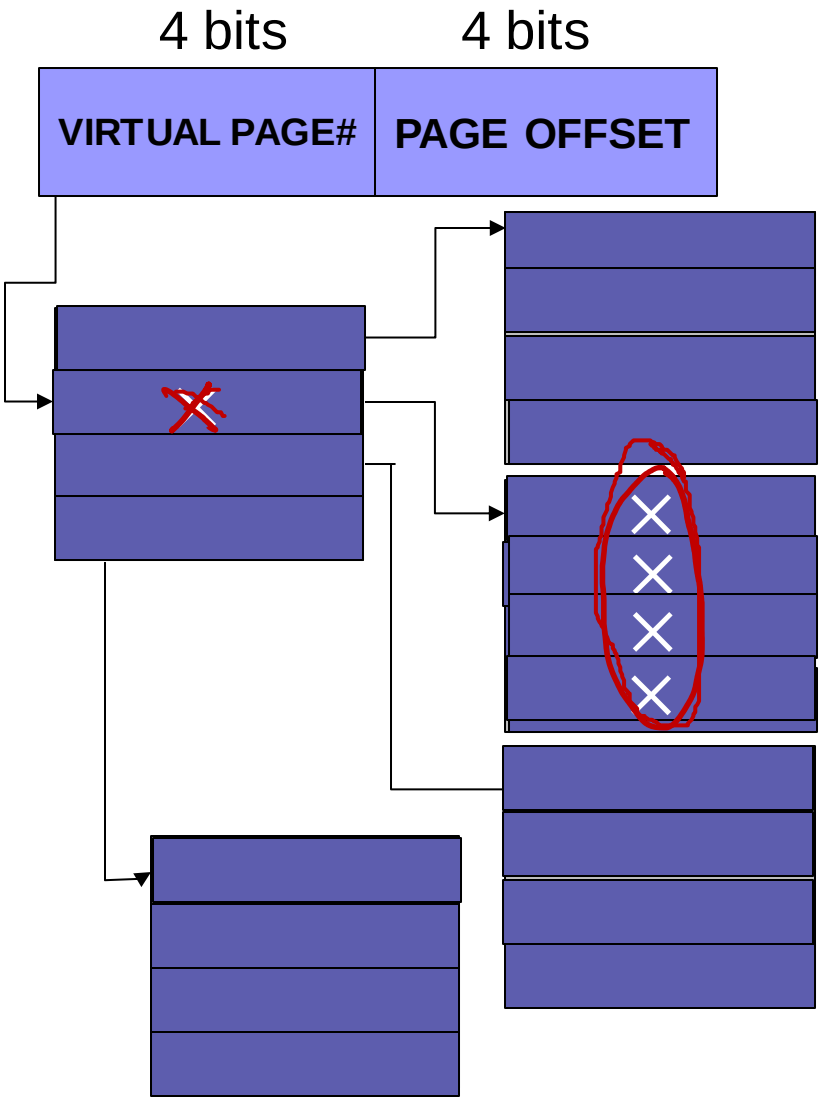


Example of a 2 Level Page Table

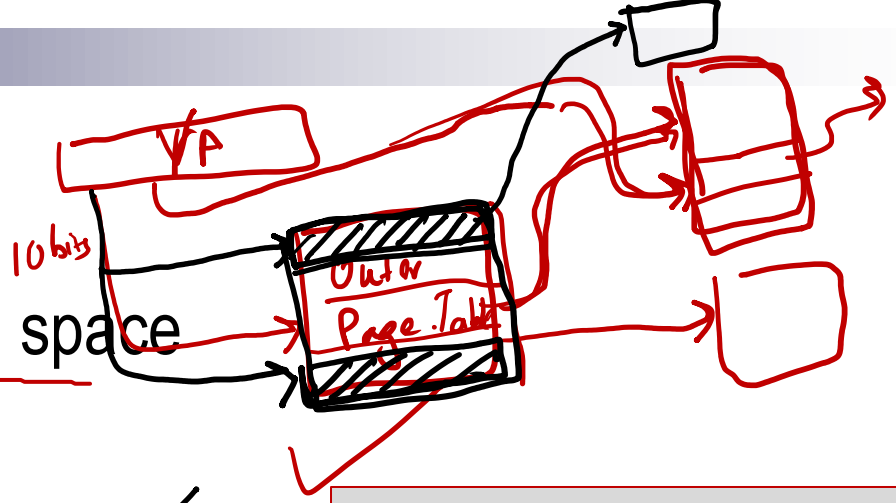
In most cases, we have unused part of address space, such that the corresponding inner table has no entry.

We mark in the outer page table that the entry is not required. Then we eliminate the inner page table.

So, we have one reasonably large outer table and small number of inner tables.

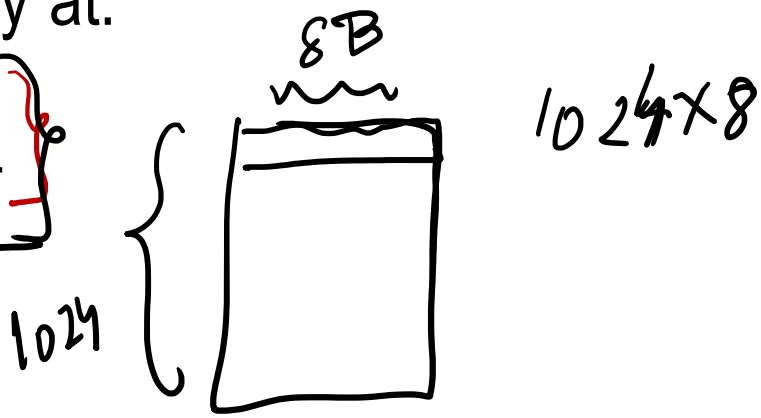


Quiz 6

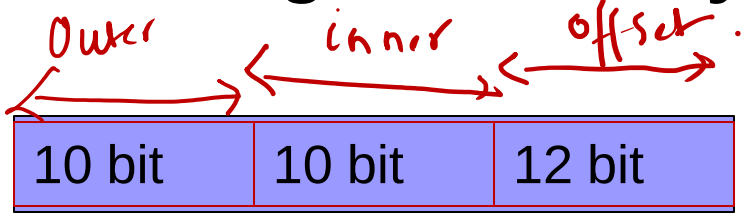


- Consider a 32-bit address space
- 4 kB page size
- 1024 entries outer page table ✓
- 1024 entries inner page table ✓
- PTE 8 bytes
- Program uses virtual memory at:
  - 0 to 0x00010000
  - 0xFFFF0000 to 0xFFFFFFFF

Flat Page table size= $\frac{2^{32}}{2^{12}} \times 8 = 8MB$



2 level Page Table Layout



0x00010000

0x0000|0000|0000|0001|0000|0000|0000|0000

First 10 bits are all zeros.

0

0xFFFFFFFF

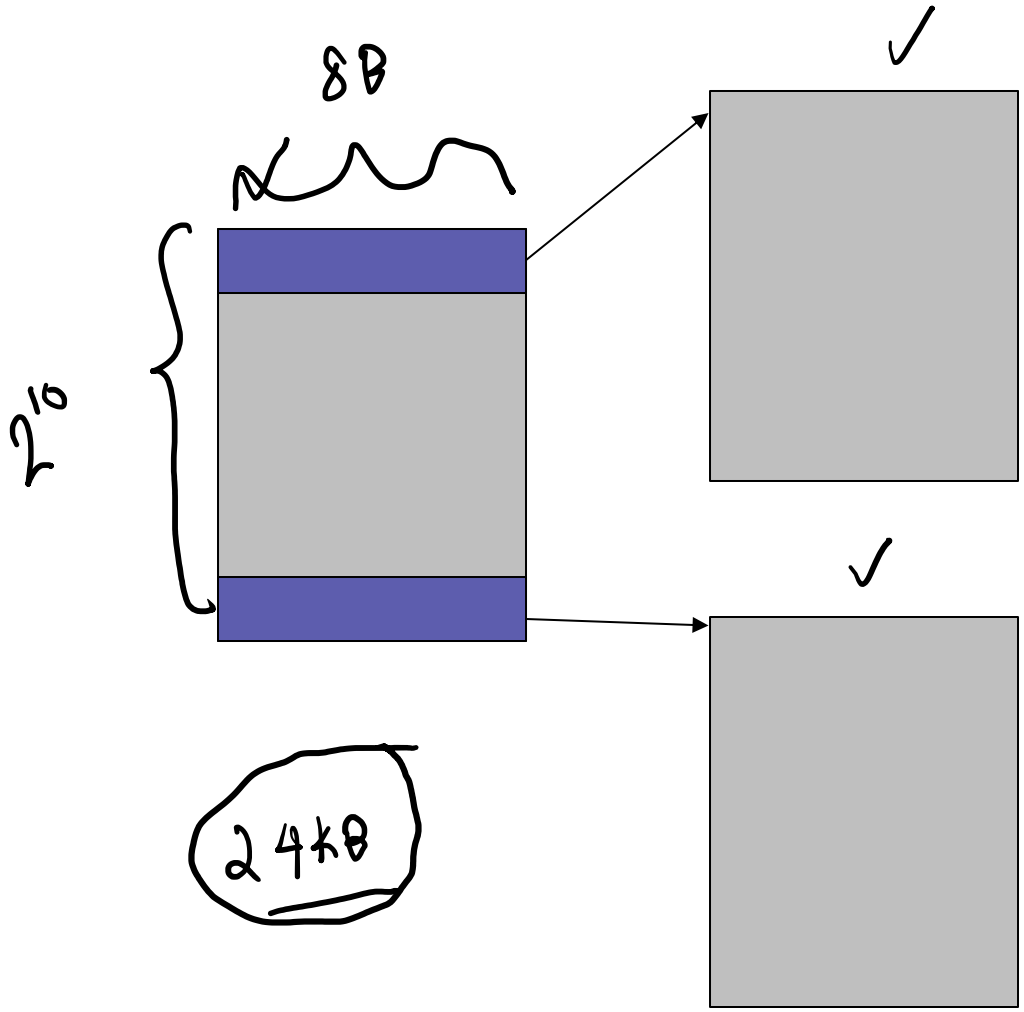
First 10 bits are all ones.

0x1111|1111|1111|1111|1111|1111|1111|1111

0xFFFF0000

0x1111|1111|1111|1111|0000|0000|0000|0000

2 level Page Table Size

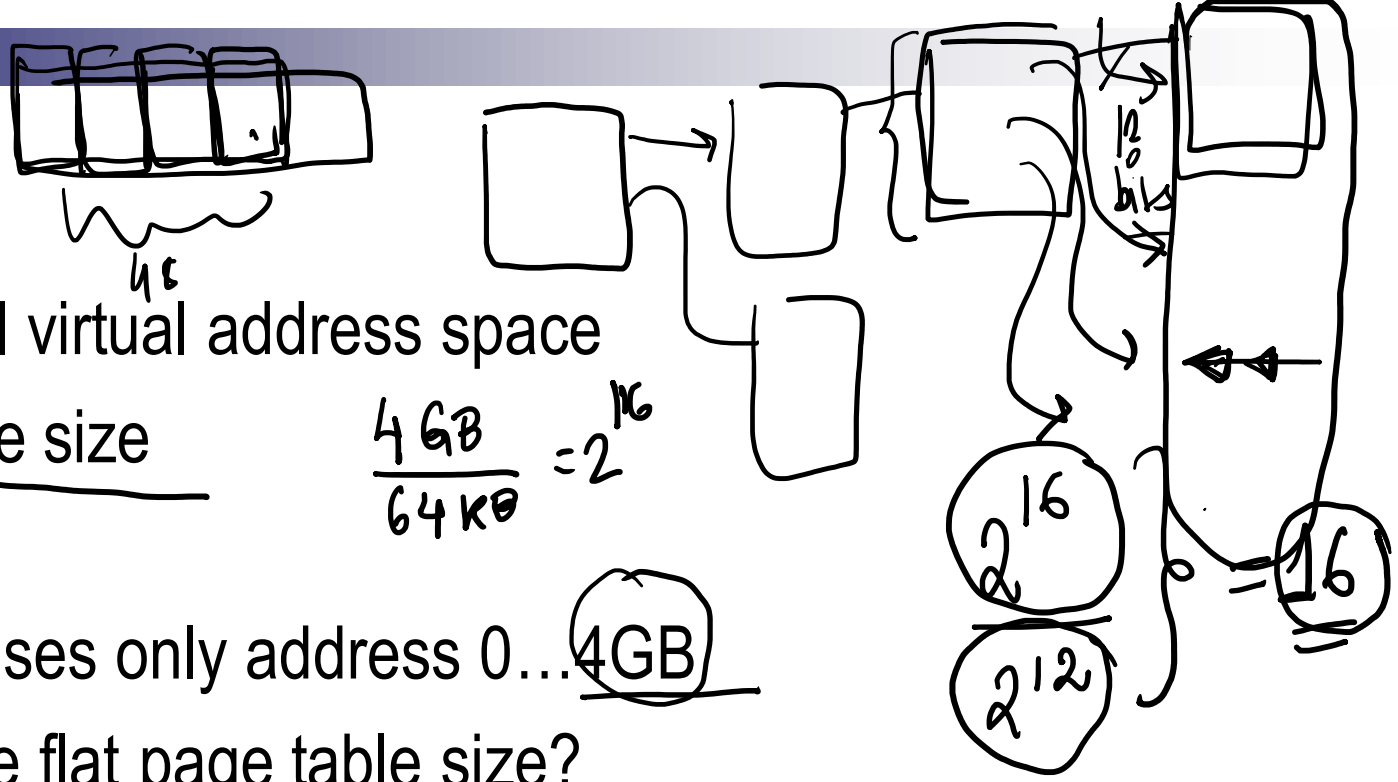


Size of each table = 1024x8=8kB. Thus, overall size=24kB

Quiz 7

- 64 bit level virtual address space
- 64 kB page size
- 8 B PTE
- Program uses only address 0…4GB
- What is the flat page table size?
- What is the size of a 4 level page table. Note Page number (#) in the VA is split equally among the levels.

Quiz 7



- 64 bit level virtual address space
- 64 kB page size
- 8 B PTE
- Program uses only address 0…4GB
- What is the flat page table size?
- What is the size of a 4 level page table. Note Page number (#) in the VA is split equally among the levels.

Flat page table size=$\frac{2^{64}}{2^{16}} \times 8 = 2^{51} Bytes.$ ✓

Page#=2^{48}. Thus, 48 bits are for page number. Thus for each level we have 48/4=12 bits.

Quiz 7

- 64 bit level virtual address space
- 64 kB page size
- 8 B PTE
- Program uses only address 0…4GB
- What is the flat page table size?
- What is the size of a 4 level page table. Note Page number (#) in the VA is split equally among the levels.

For a 4GB space, we need $2^2 \times 2^{30} = 2^{32}$.

Thus, no of pages needed=$\frac{2^{32}}{2^6 \times 2^{10}} = 2^{16}$

Quiz 7

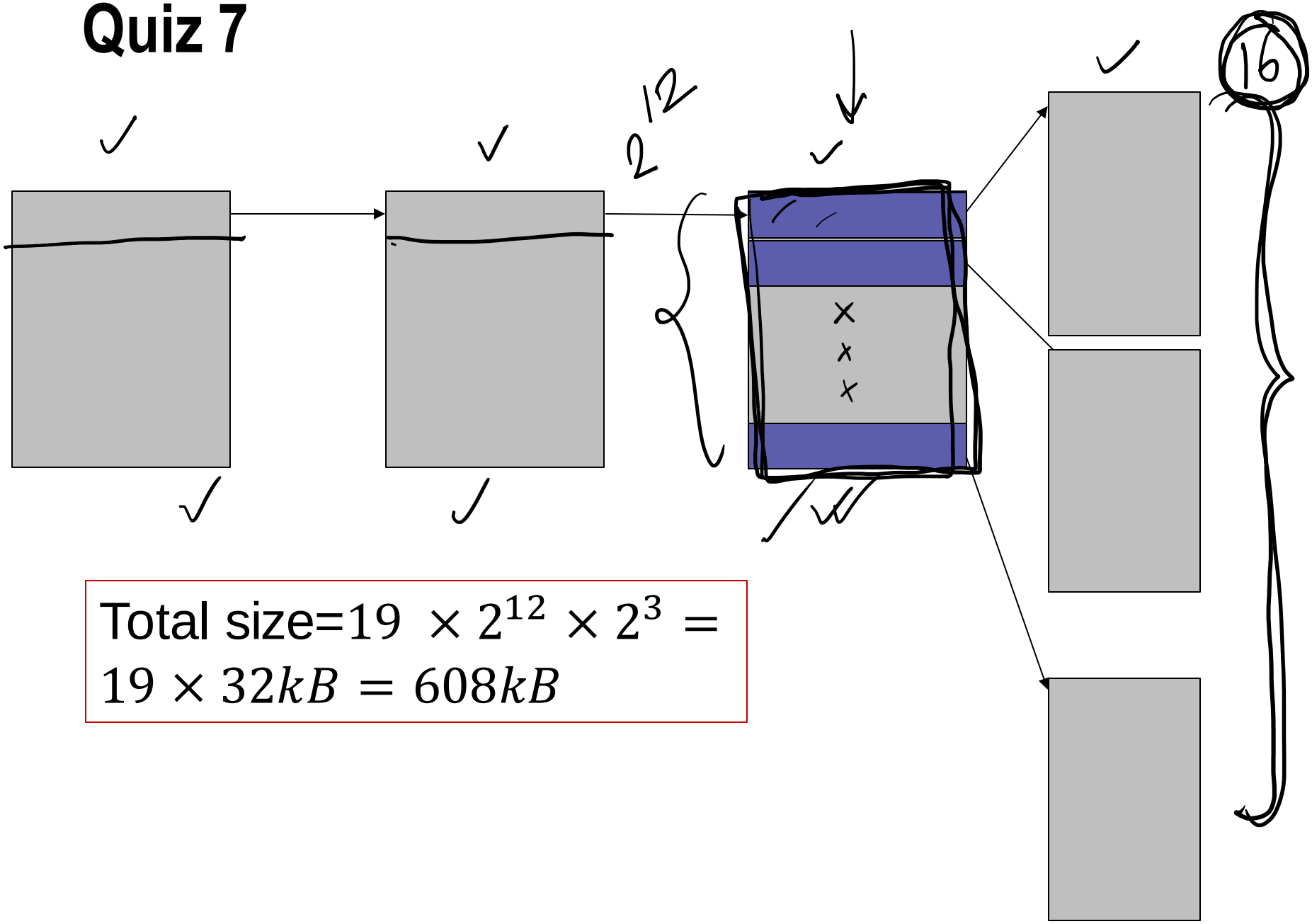
- 64 bit level virtual address space
- 64 kB page size
- 8 B PTE
- Program uses only address 0…4GB
- What is the flat page table size?
- What is the size of a 4 level page table. Note Page number (#) in the VA is split equally among the levels.

$\frac{4\,GB}{64\,KB}$

Each inner/outer page table has 2^{12} pages.

Thus, we need $\frac{2^{16}}{2^{12}} = 16$ inner tables.

Quiz 7



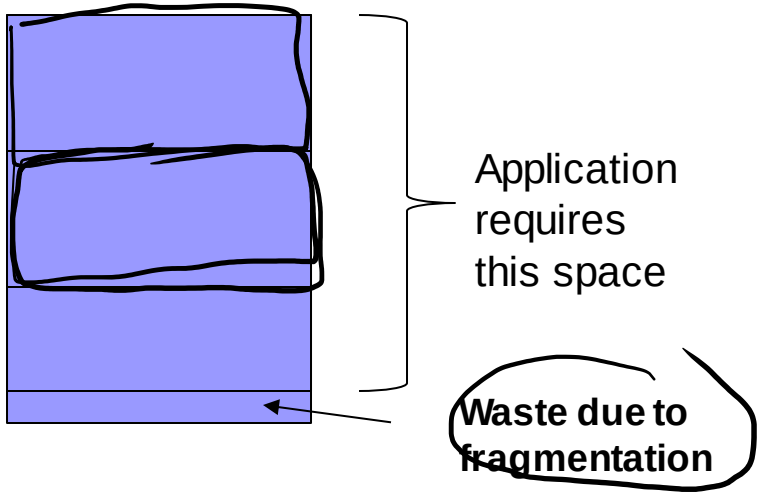
Total size=$19 \times 2^{12} \times 2^{3} = 19 \times 32kB = 608kB$

Choosing Page Size

Smaller Page	Larger Page
Larger Page Table	Smaller Page Table
	Internal Fragmentation

In legacy processors, like x86, we use 4kB page sizes. Because we did not want to waste space.
In modern processors, the choice is more inclined towards few MB, as memory has become cheaper so we can live with some wastage due to fragmentation, and thus have smaller page tables.

Internal fragmentation occurs because we have to store translations in multiples of pages.



Memory Access Time due to Translation

- Consider, LD R1, 4(R2). Here the VA needs to be translated.
- Page tables require at least two memory accesses per instruction:
  - One to fetch the page table entry (PTE)
  - Secondly, to fetch the corresponding data from the cache/memory
- The steps are:
  - Compute the VA (fast, add the base register with offset)
  - Compute the Page Number (fast, using the VA bits)
  - Compute the PA of the needed PTE by adding the page number to the beginning address of the page table
  - Read the PTE (this could be slow)
  - Compute the PA by combining the physical frame no from PTE with the page offset from the VA
  - Access cache/memory to get the data (this could be slow)

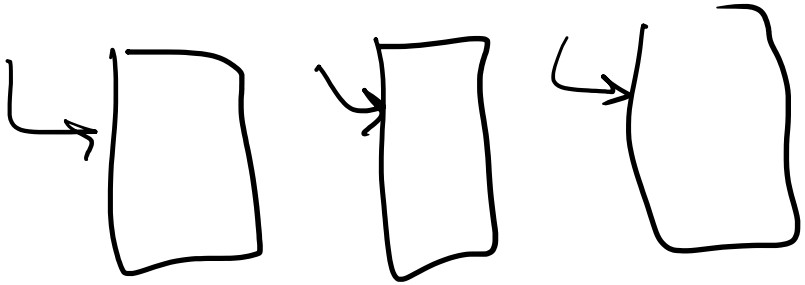
Quiz 8

- 1 cycle to compute VA
- 1 cycle to access cache
- 10 cycles to access memory
- 90% hit rate for data
- PTEs are not cached
- 3 level page table

How many clock cycles are needed for LW R1, 4(R2)?

Quiz 8

- 1 cycle to compute VA
- 1 cycle to access cache
- 20 cycles to access memory
- 90% hit rate for data
- PTEs are not cached
- 3 level page table



How many clock cycles are needed for LW R1, 4(R2)?

1+3x20+(1+0.1x20)=1+60+3=64 (60 cycles for Page Table access)

Quiz 8

- 1 cycle to compute VA
- 1 cycle to access cache
- 10 cycles to access memory
- 90% hit rate for data
- PTEs are cached
- 3 level page table

How many clock cycles are needed for LW R1, 4(R2)?

Quiz 8

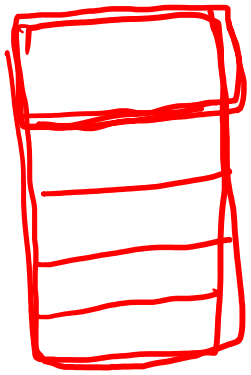
- 1 cycle to compute VA
- 1 cycle to access cache
- 10 cycles to access memory
- 90% hit rate for data
- PTEs are cached
- 3 level page table

How many clock cycles are needed for LW R1, 4(R2)?

1+3(1+0.1x20)+(1+0.1x20)=1+9+3=13 (9 cycles for Page Table access)

Translation Look-Aside Buffer (TLB)

- TLB is a cache for translation
- How is TLB better than using the cache itself?
  - TLB storing only translations can be very small and hence fast
- Consider, a program that needs say 20 kB of data to be accessed.
  - With cache we would need to store translations for 20 kB of data.
  - On the other hand, assuming a page size of 4kB, we need just 5 entries in the TLB.



TLB smaller faster

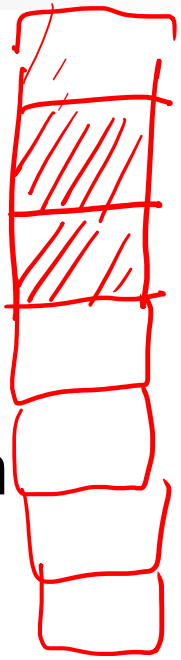
TLB Miss

- Perform Translation using Page Table
- Put the correct translation in the TLB for future
- Software TLB Miss Handling:
  - OS accesses the page tables. It can use fancy data-structures like B-trees, hash tables
  - The OS puts the correct translation into the TLB
- Hardware TLB Miss Handling:
  - Processor reads the page tables
  - Thus, the page tables are organized as flat page tables, or multi-level so that they are easily accessible by the hardware.

Alpha Pal
Privileged.
Access Library.
Intel.
walks the hierarchical structure to obtain the PTE.

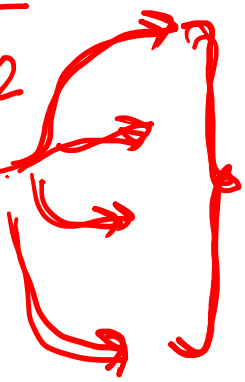
Quiz 9

$\frac{32\,kB}{4\,kB} = 8$



- 32 KB cache, 64 B block size
- 4 kB page size ✓
- How many TLB entries if we want similar miss rates in cache and TLB:
  - ☐ 8 to 512
  - ☐ 32 to 64
  - ☐ <64
  - ☐ >32

$\#\,Blocks = \frac{32\,kB}{64B} = \frac{32 \times 2^{10}}{64} = 512$



64 to 512 ✓

Quiz 9

- 32 KB cache, 64 B block size
- 4 kB page size
- How many TLB entries if we want similar miss rates in cache and TLB:
  - 8 to 512
  - 32 to 64
  - <64
  - >32

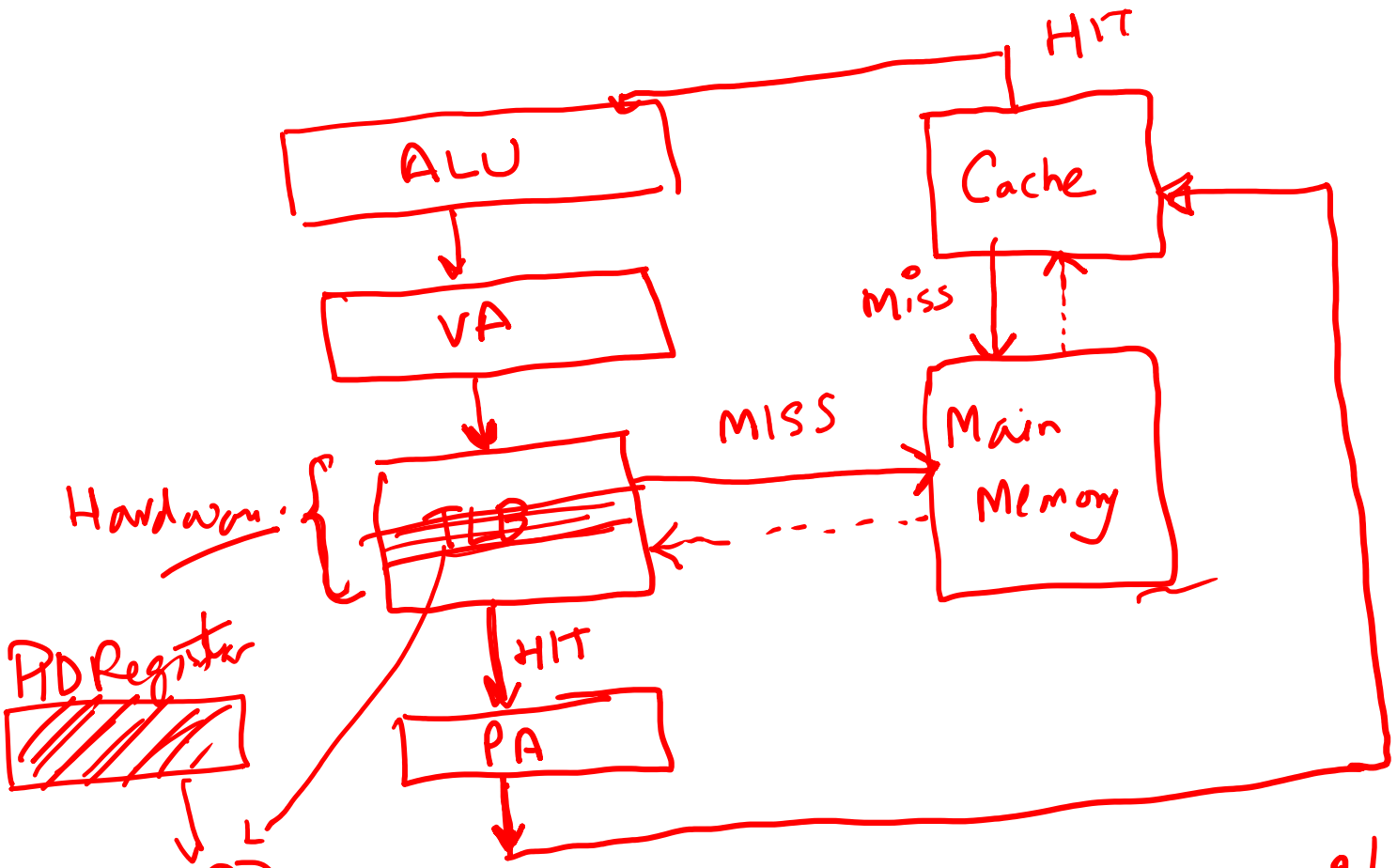
TLB needs to cover same amount of memory as the cache size for similar miss rates. Thus, it will have $\frac{32kB}{4kB} = 8$ entries.

However, 32kB with 64 B block size has $\frac{32 \times 2^{10}}{64} = 512$ blocks. This could be theoretically spread in 512 pages. The TLB in that case should have 512 entries.

TLB Organization

Intel Core Duo · Page Size 4KB ·
I-TLB } 64 entry (FA)
D-TLB } 1
512 entries

- Associativity? Small => Fast.
  - It tends to be usually fully or highly associative.
  - And not direct mapped, as then hit rate gets affected.
- Size? We want hit rates like the cache, and thus cover more memory than the cache. Usually say between 64 to 512 entries.
  - Making it large would make the TLB slow!
  - If we want more entries we have more levels.
  - A small and fast L1 TLB (one cycle access)
  - A slower but larger L2 TLB (several cycles access, but faster than physical translation and going to memory)

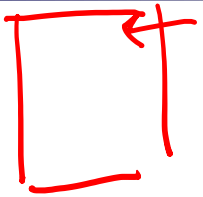


Memory Reference Process

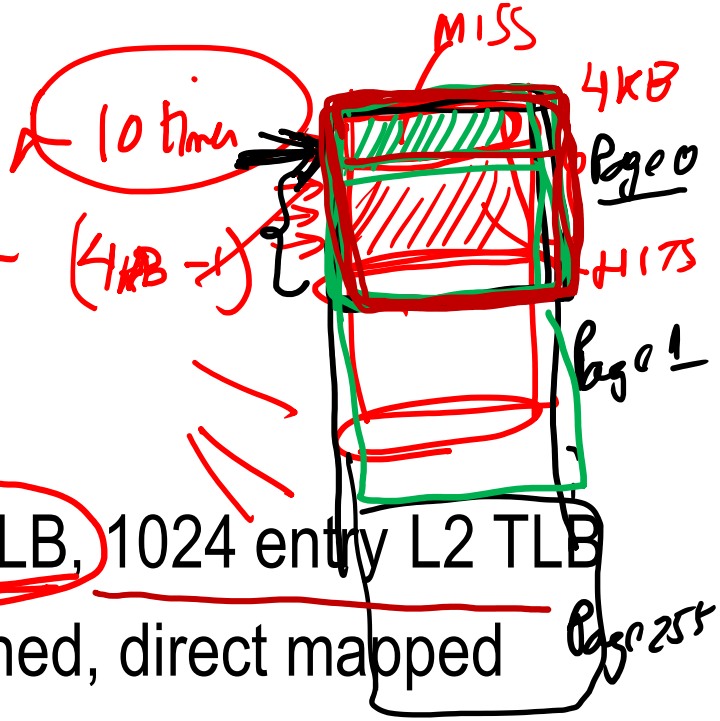
- On a context switch in case of a page fault we need to perform bookkeeping ~~to~~ performed by appending PID to the tag.

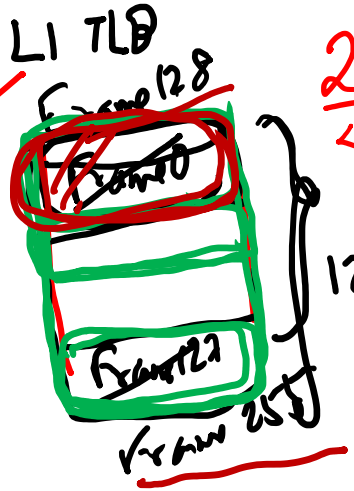
- LRU replacement policy.

Quiz 10



- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses





Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses

Virtual space for the array: 1 MB=$2^{20}B$. Thus it needs $\frac{2^{20}}{2^{12}} = 256$ pages.

When we have the first access we will have TLB L1 miss.
A L2 TLB miss will also occur, and a translation will be performed by the PT.
The translation will be updated in both L1 and L2 TLBs.

Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses

Next we access the second byte.
We should get a TLB L1 hit!
This is because it is in the same page as the first one.

Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses

Next we access the second byte.
We should get a TLB L1 hit!
This is because it is in the same page as the first one.

Thus, in the first iteration there will be 256 L1 TLB misses and (4kB-1)x256=4095x256 hits in L1 TLB.

Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses

Thus, in the first iteration there will be 256 L1 TLB misses and (4kB-1)x256=4095x256 hits in L1 TLB.

In the next iteration, L1 TLB contains entries for 2nd half of the array.

This is because it has only 128 entries and the array needs 256 PTEs.
The L2 TLB contains all entries.

Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses

Thus, after 1st iteration, and for all subsequent iterations, L1 TLB will continue to have 256 hits and 4095x256 misses.

But L2 TLB?

Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: ? Hits ? Misses
- L2 TLB: ? Hits ? Misses

Thus, after 1st iteration, and for all subsequent iterations, L1 TLB will continue to have 256 hits and 4095x256 misses.

L2 TLB is accessed only when there is a miss in L1 TLB, ie for the 10x256 cases.
In the 1st iteration, L2 TLB also had 256 misses, but for the remaining 9 iterations it will have all hits.

Quiz 10

- 1 MB array, read one byte at a time
- Repeat this for 10 times
- There is no other memory accessed
- Consider, 4 kB page, 128 entry L1 TLB, 1024 entry L2 TLB
- TLBs initially empty, array page-aligned, direct mapped TLBs
- L1 TLB: **10x4095x256** Hits **10x256** Misses
- L2 TLB: **9x256** Hits **256** Misses

Thank You!