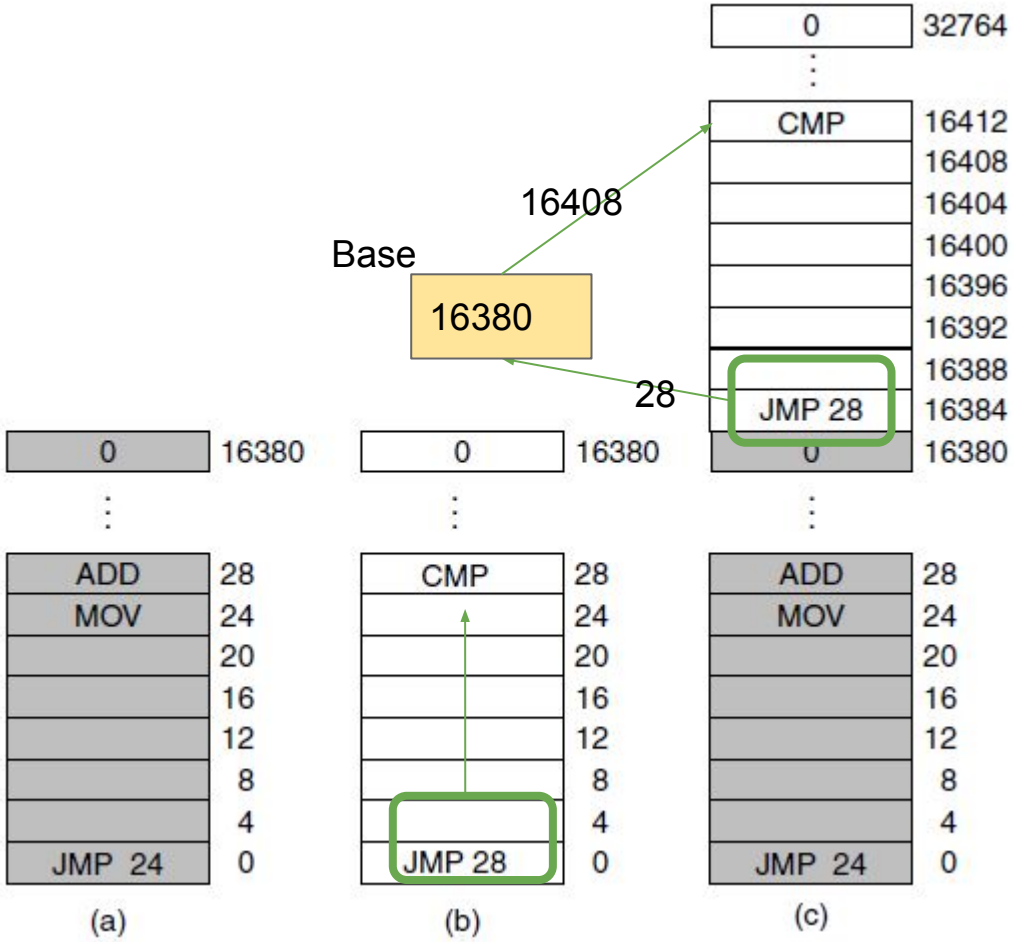
Base and Limit

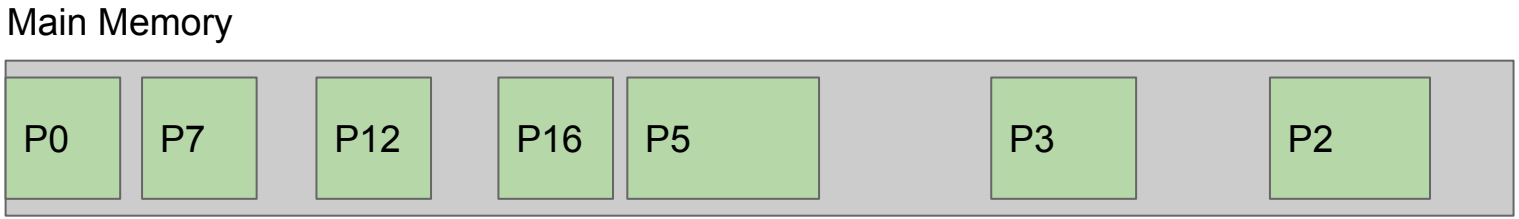
- This supports moving the process later - because there is no editing
- Translation is in **hardware** - the OS is not active!



External Fragmentation

- Overtime, as programs start and terminate, holes in memory develop
- The longer this goes on, the smaller the “fragments” become, and the more prevalent they are!

Eventually, only very small new process can be allocated!



Maximizing Swap Speed

- Disk is read in blocks of regular size
  - ie. 1024 bytes
- It’s more efficient if “processes” are allocated in the same increments
- Having processes stored in increments mean locating them are faster as well.

Swapping

- Even with compaction, eventually we won’t be able to fit all processes in memory
- The OS can move processes that are not running onto Disk (temporarily).
- They can be **swapped** back onto Disk when they are ready to be run

Allocating in Increments

- Processes are likely to be allocated *more* memory than the actually need:
  - Assume block size is 100 bytes
  - A process requiring 95 bytes will get 100
    - 5% waste of space
  - A process requiring 102 bytes will get 200 bytes!
    - 98/200 = 49% waste of space!

This waste is referred to as **Internal Fragmentation**

Virtual Memory

Module 10

Problems with our model

1) External Fragmentation is a **big** problem
2) We can't fit all programs in memory…
    a) We can't even fit 1 program's address space!

- The root cause for #1 is **contiguous allocation**

Our solution to this will pave the way to solving problem #2 as well!

Paging

- An address space can be divided into equally sized “pages” (except for the last page…)
- Each address that the program generates is considered a ***virtual address***
  - Sometimes also called “logical address”
- Much like with base/limit registers, we convert the virtual address to a physical address

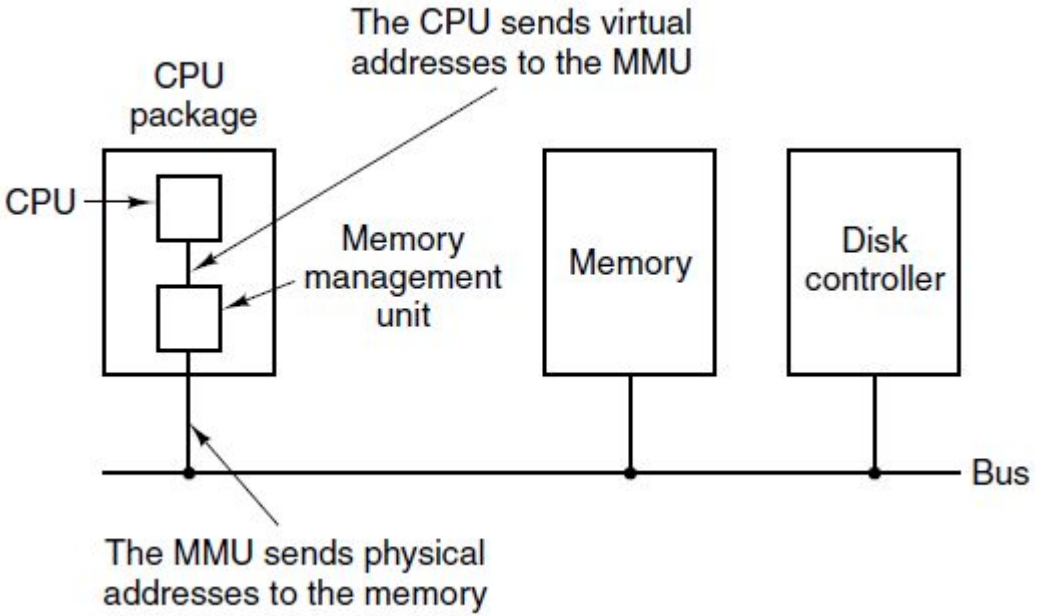
Frames

- The virtual address space is divided into pages
- The physical address space is divided into frames.
- We'll load pages into frames…
  - Each page can reside in ANY frame
  - Consecutive pages do not reside in consecutive frames
- Page size = Frame size
  - # of Pages is not necessarily = # Frames
    - In fact, we likely have WAY more pages.

Pages and Frames

- To perform the translation, we'll employ a ***new*** part of the hardware - called the Memory Management Unit (MMU)
- The MMU will inspect a virtual address and determine its ***page*** number.
- It will convert the page number to a **frame number**.
  - To resolve a particular address:
  - The offset within the page is used as the offset within the frame

The MMU



MMU is a blend of software + hardware support.

External Fragmentation?

Why does paging solve this?

Inside the MMU

- The simplest MMU is a **page table**.
- Each process has a page table (managed by OS)
- There is an entry for each page, with a frame number associated
- To resolve a virtual address, simply compute page number and index into page table.

Process 1 Logical Addresses			
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Process 2 Logical Addresses			
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

Page Table		
Page #		Frame #
0	0	4
0	1	2
1	0	6
1	1	0

Page Table		
Page #		Frame #
0	0	3
0	1	1
1	0	X
1	1	X

Physical Address Space	
F	Data
0	7
	21
	41
	17
1	78
	65
	18
	100
2	87
	24
	40
	89
3	50
	39
	31
	97
4	35
	19
	24
	25
5	13
	23
	16
	20
6	1
	13
	67
	89
	34

Process 1 Logical Addresses			
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Process 2 Logical Addresses			
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

Page Table		
Page #		Frame #
0	0	4
0	1	2
1	0	6
1	1	0

Page Table		
Page #		Frame #
0	0	3
0	1	1
1	0	X
1	1	X

Physical Address Space	
F	Data
0	7
	21
	41
	17
1	78
	65
	18
	100
2	87
	24
	40
	89
3	50
	39
	31
	97
4	35
	19
	24
	25
5	13
	23
	16
	20
6	1
	13
	67
	89
	34

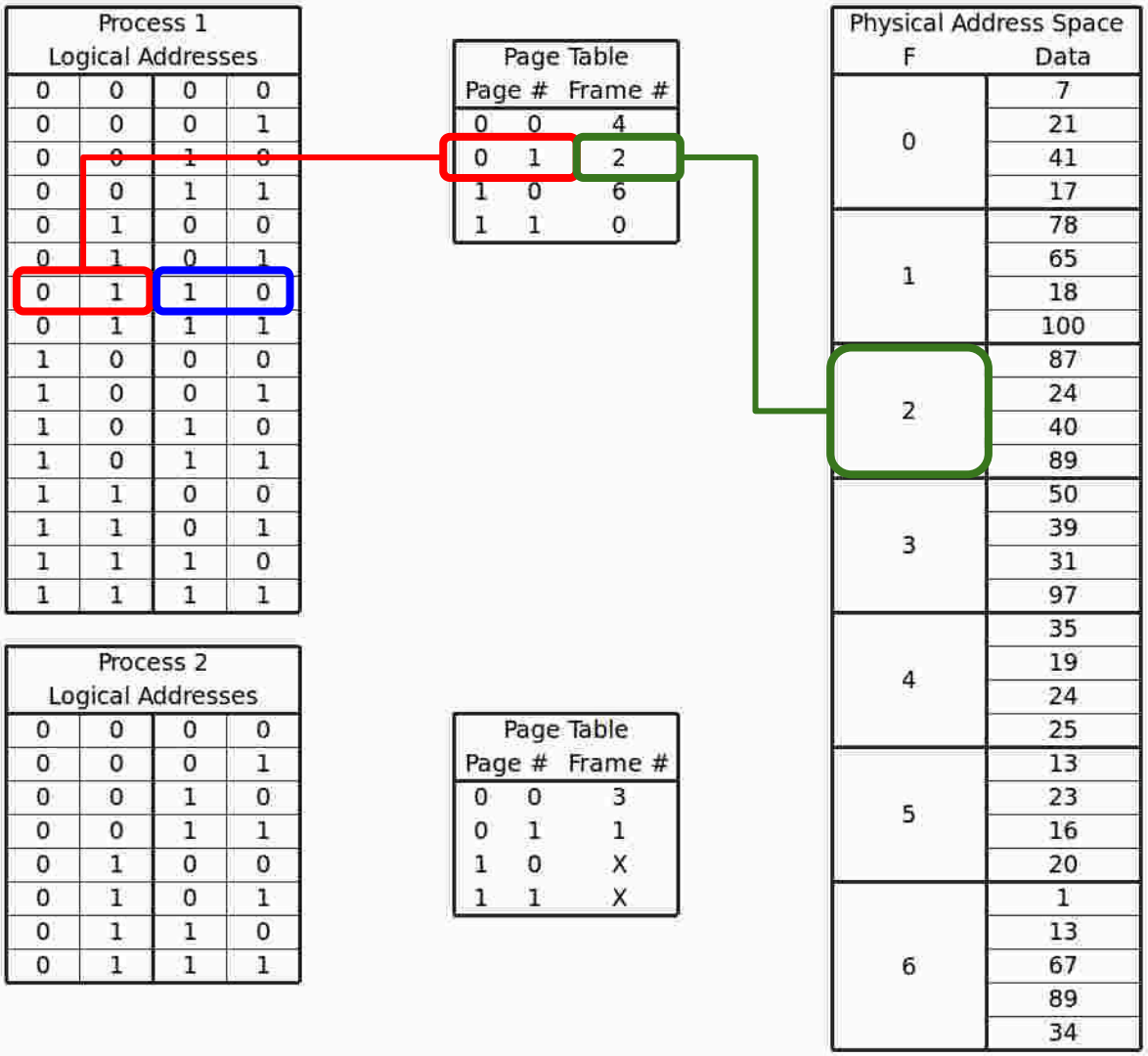
Process 1 Logical Addresses			
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

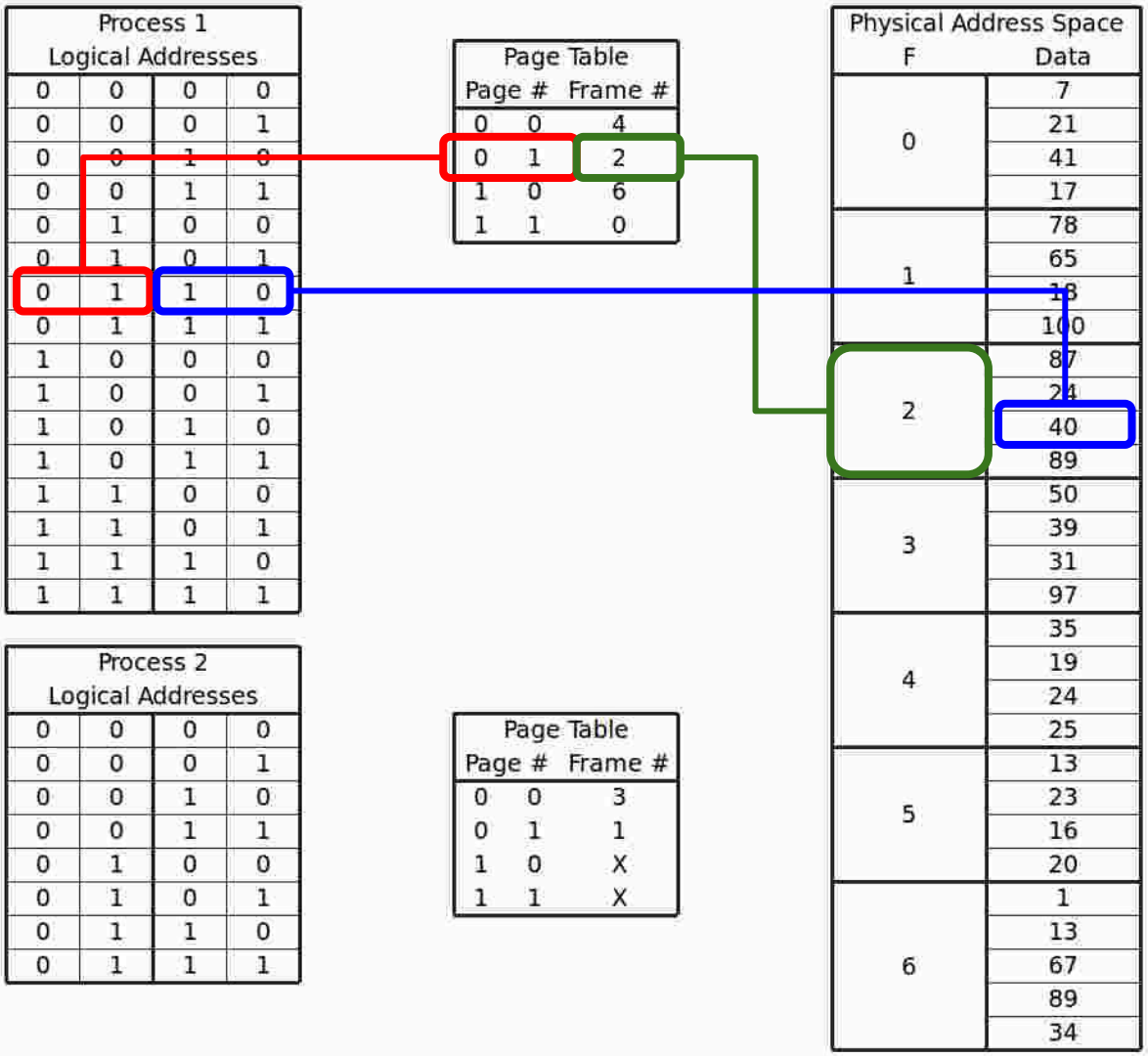
Page Table	
Page #	Frame #
0 0	4
0 1	2
1 0	6
1 1	0

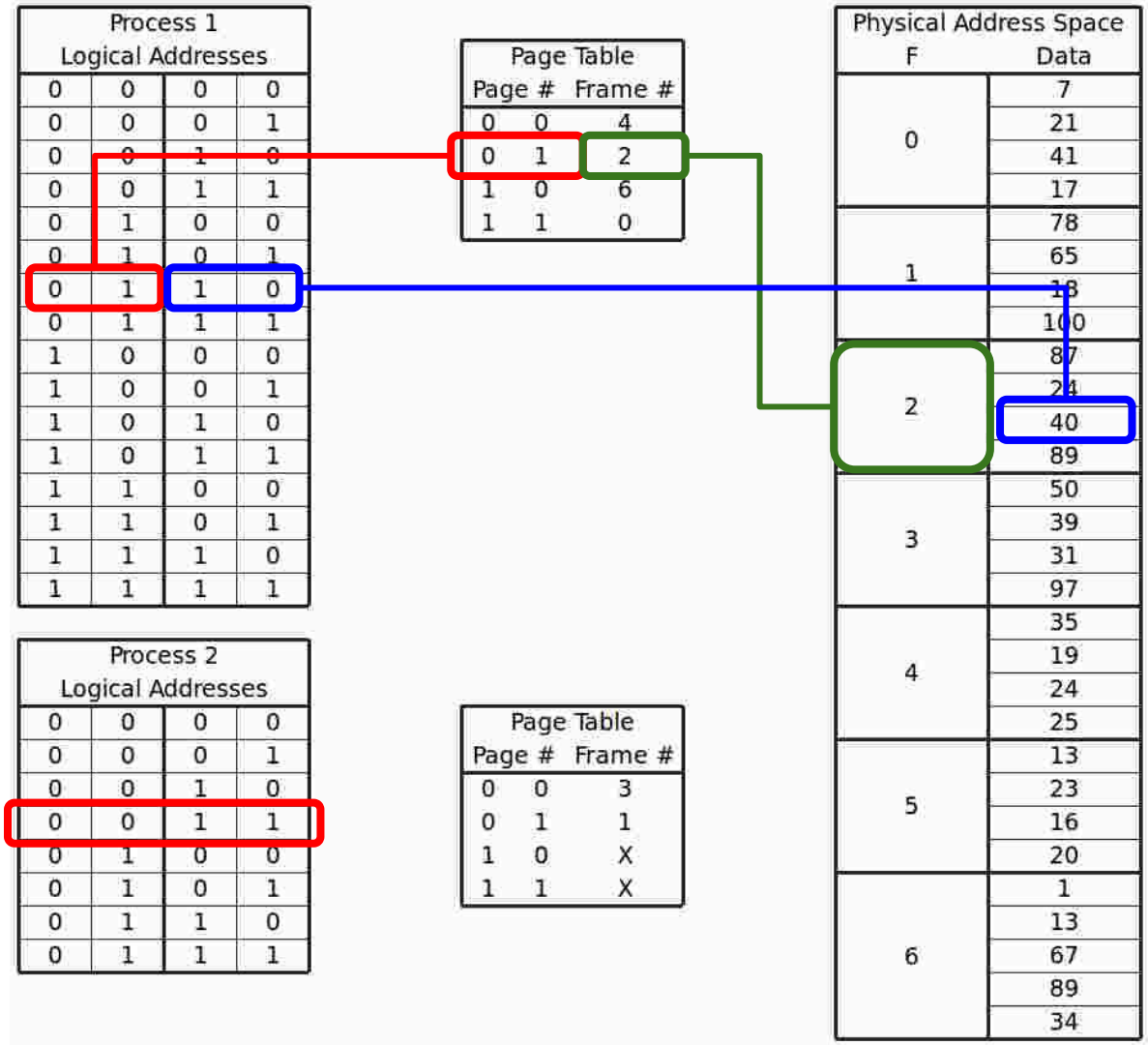
Process 2 Logical Addresses			
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

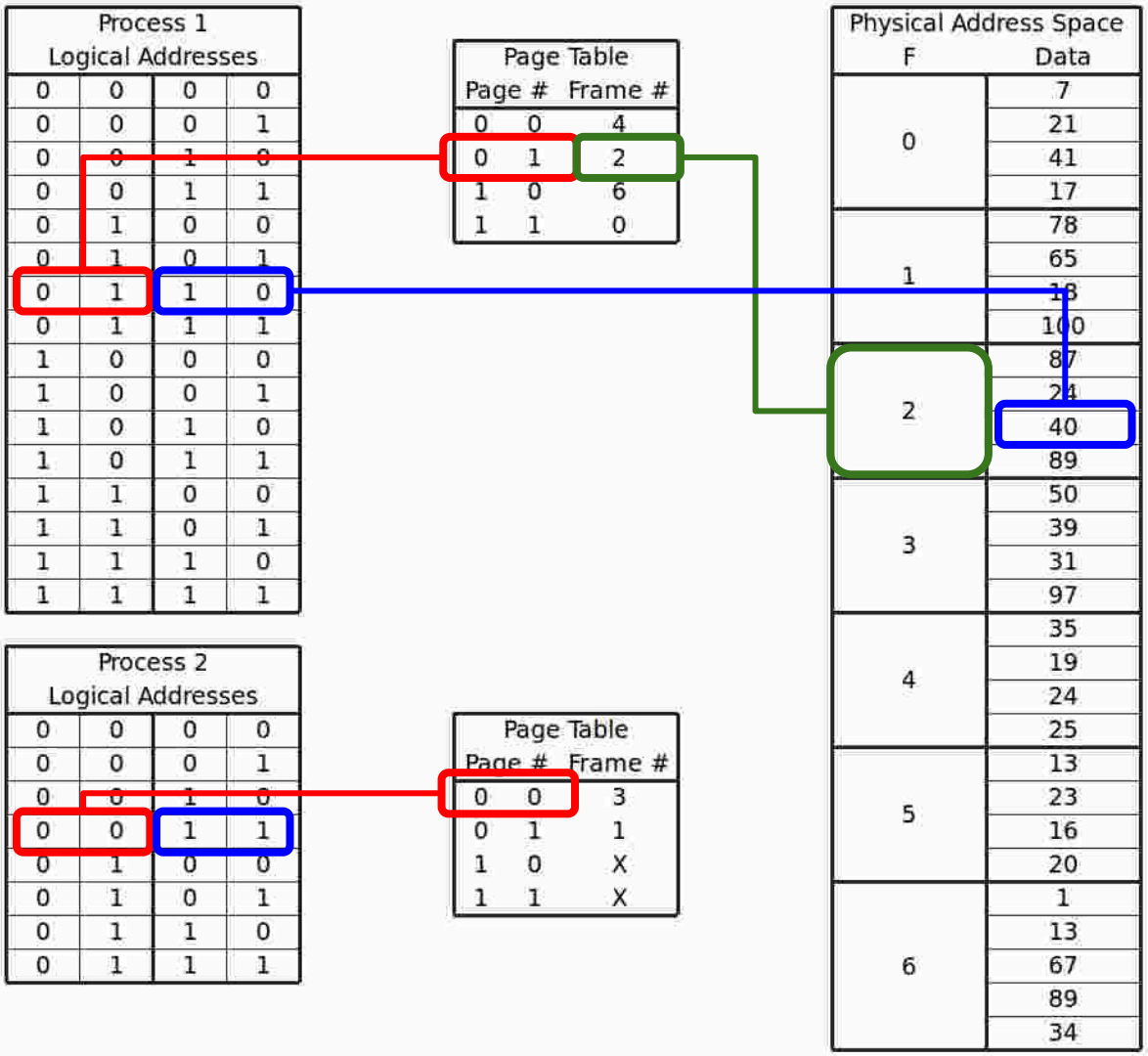
Page Table	
Page #	Frame #
0 0	3
0 1	1
1 0	X
1 1	X

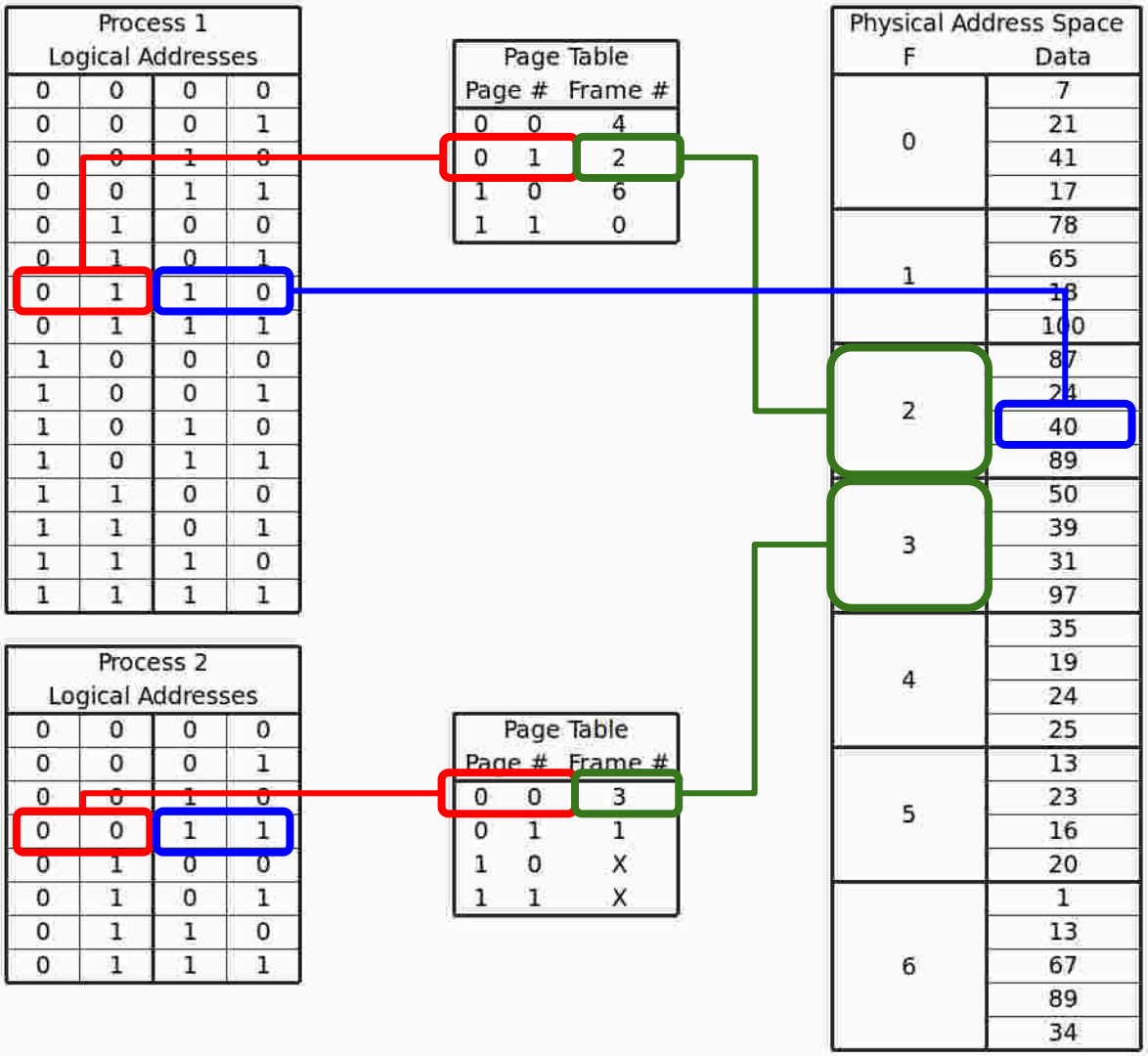
Physical Address Space	
F	Data
0	7
	21
	41
	17
1	78
	65
	18
	100
2	87
	24
	40
	89
3	50
	39
	31
	97
4	35
	19
	24
	25
5	13
	23
	16
	20
6	1
	13
	67
	89
	34

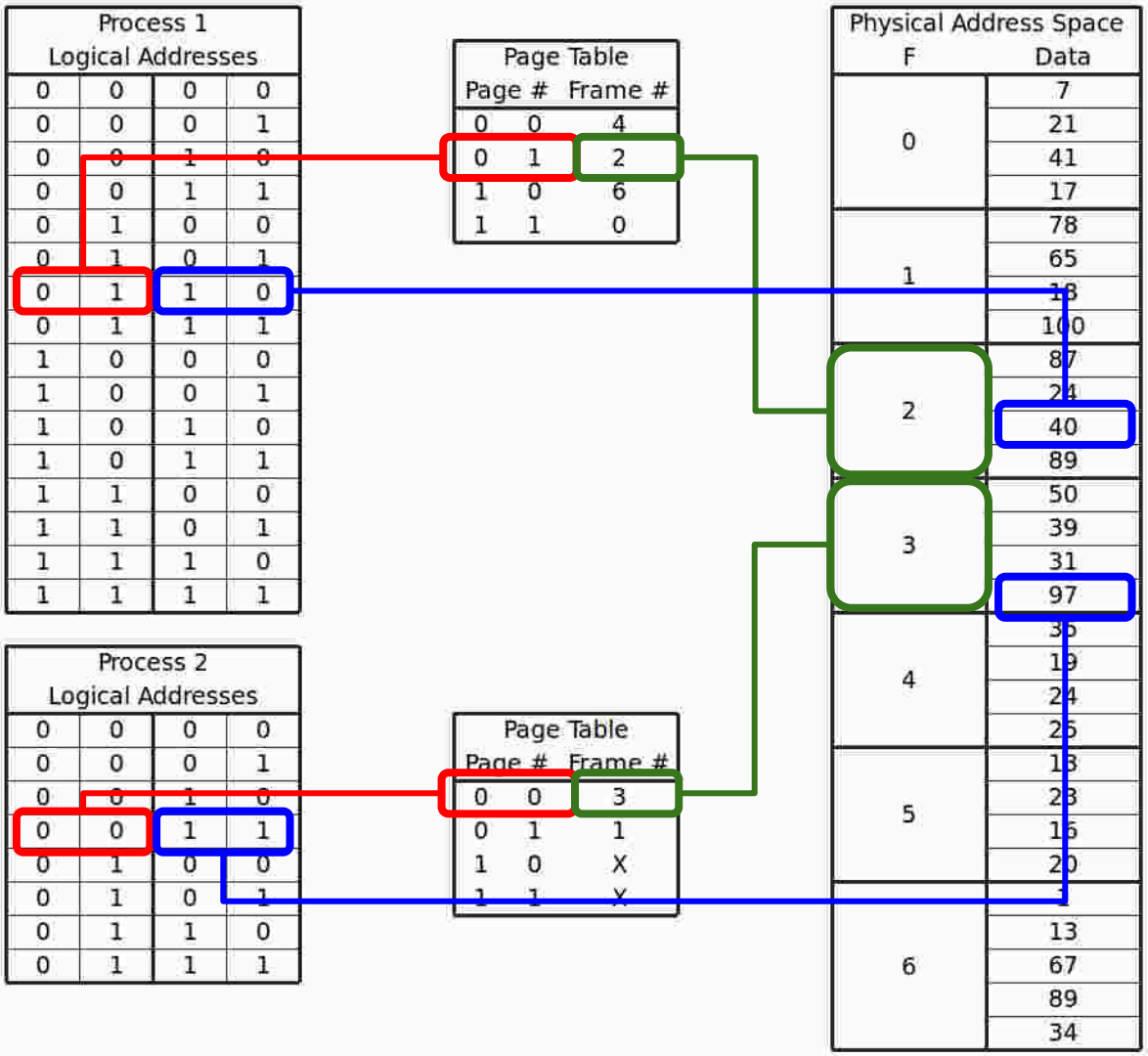

Process 1
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
Page Table
Page # Frame #
0 0 4
0 1 2
1 0 6
1 1 0
Physical Address Space
F Data
0 7
21
41
17
1 78
65
18
100
2 87
24
40
89
3 50
39
31
97
4 35
19
24
25
5 13
23
16
20
6 1
13
67
89
34
Process 2
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
Page Table
Page # Frame #
0 0 3
0 1 1
1 0 X
1 1 X


Process 1
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
Page Table
Page # Frame #
0 0 4
0 1 2
1 0 6
1 1 0
Physical Address Space
F Data
0 7
21
41
17
1 78
65
18
100
2 87
24
40
89
3 50
39
31
97
4 35
19
24
25
5 13
23
16
20
6 1
13
67
89
34
Process 2
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
Page Table
Page # Frame #
0 0 3
0 1 1
1 0 X
1 1 X


Process 1
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
Page Table
Page # Frame #
0 0 4
0 1 2
1 0 6
1 1 0
Physical Address Space
F Data
0 7
21
41
17
1 78
65
18
100
2 87
24
40
89
3 50
39
31
97
4 35
19
24
25
5 13
23
16
20
6 1
13
67
89
34
Process 2
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
Page Table
Page # Frame #
0 0 3
0 1 1
1 0 X
1 1 X


Process 1
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
Page Table
Page # Frame #
0 0 4
0 1 2
1 0 6
1 1 0
Physical Address Space
F Data
0 7
21
41
17
1 78
65
18
100
2 87
24
40
89
3 50
39
31
97
4 35
19
24
25
5 13
23
16
20
6 1
13
67
89
34
Process 2
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
Page Table
Page # Frame #
0 0 3
0 1 1
1 0 X
1 1 X


Process 1
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
Page Table
Page # Frame #
0 0 4
0 1 2
1 0 6
1 1 0
Physical Address Space
F Data
0 7
21
41
17
1 78
65
18
100
2 87
24
40
89
3 50
39
31
97
4 35
19
24
25
5 13
23
16
20
6 1
13
67
89
34
Process 2
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
Page Table
Page # Frame #
0 0 3
0 1 1
1 0 X
1 1 X


Process 1
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
Page Table
Page # Frame #
0 0 4
0 1 2
1 0 6
1 1 0
Physical Address Space
F Data
0
7
21
41
17
1
78
65
18
100
2
87
24
40
89
3
50
39
31
97
4
36
19
24
25
5
13
23
15
20
6
1
13
67
89
34
Process 2
Logical Addresses
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
Page Table
Page # Frame #
0 0 3
0 1 1
1 0 X
1 1 X

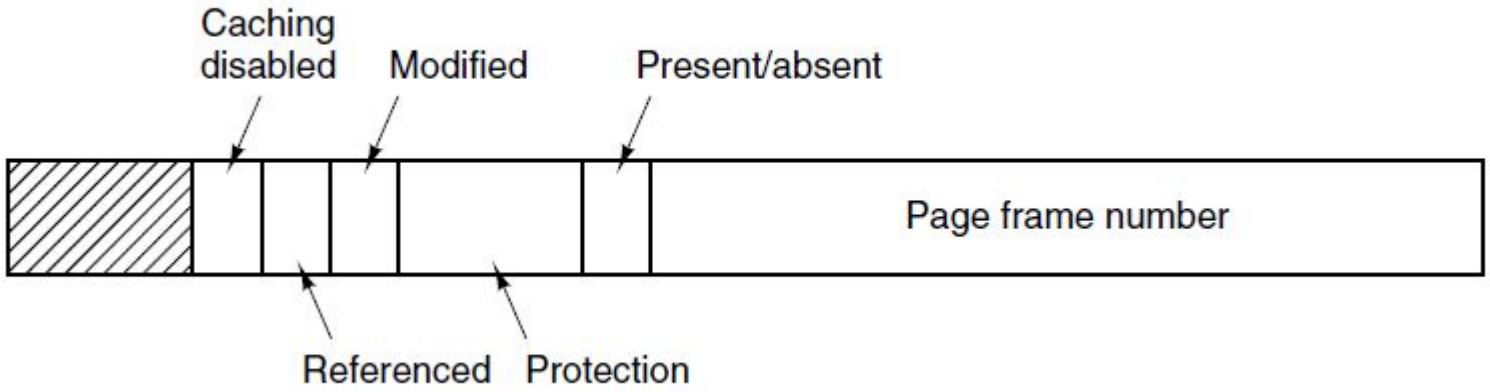
Problems with our model

1) External Fragmentation is a **big** problem
2) We can't fit all programs in memory…
    a) We can't even fit 1 program's address space!

What about Problem #2

- We actually don’t need to put **every** page into a frame… in fact we can’t!
- On-Demand Paging:
  - The full process resides on disk (swap space)
  - Brought into memory on access
    - We’ll deal with the question of how many frames each process gets later...

A page table entry



Page Faults

1. Translate Virtual Address to determine page number, consult page table
2. If page is invalid, trap to OS
3. OS locates page on backing store / swap
4. Find free frame in memory and copy page from backing store
5. Update page table
6. Restart program that initiated the request

External/Internal Fragmentation

- External Fragmentation is gone because any process can be broken into pages
- Any page can be allocated to any Frame
- Internal Fragmentation is proportional to page/frame size (just like it was before)

MMU Problems...

There are some major problems with this view of the MMU:

- Problem 1: Where's the page table?
  - If it's dedicated registers, remember there could be **millions of entries**… for each process
  - Swapping process page tables into registers would be too slow
  - Millions of registers is silly expensive.

Where’s the page table?

OK… so let’s put the page table in memory instead…

Now each memory reference requires two memory accesses!

Solution: A new Cache

A Translation Lookaside Buffer (TLB)

- Caches commonly used page->frame translations
- Dedicated hardware (fast)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Now we might have a TLB hit or a TLB miss
On miss, we consult page table in memory

TLB Miss Rate

In order to *really* improve things, the TLB miss rate must be exceptionally low (< 0.1%)

How is this possible?

MMU Problem 2: Page Table Size

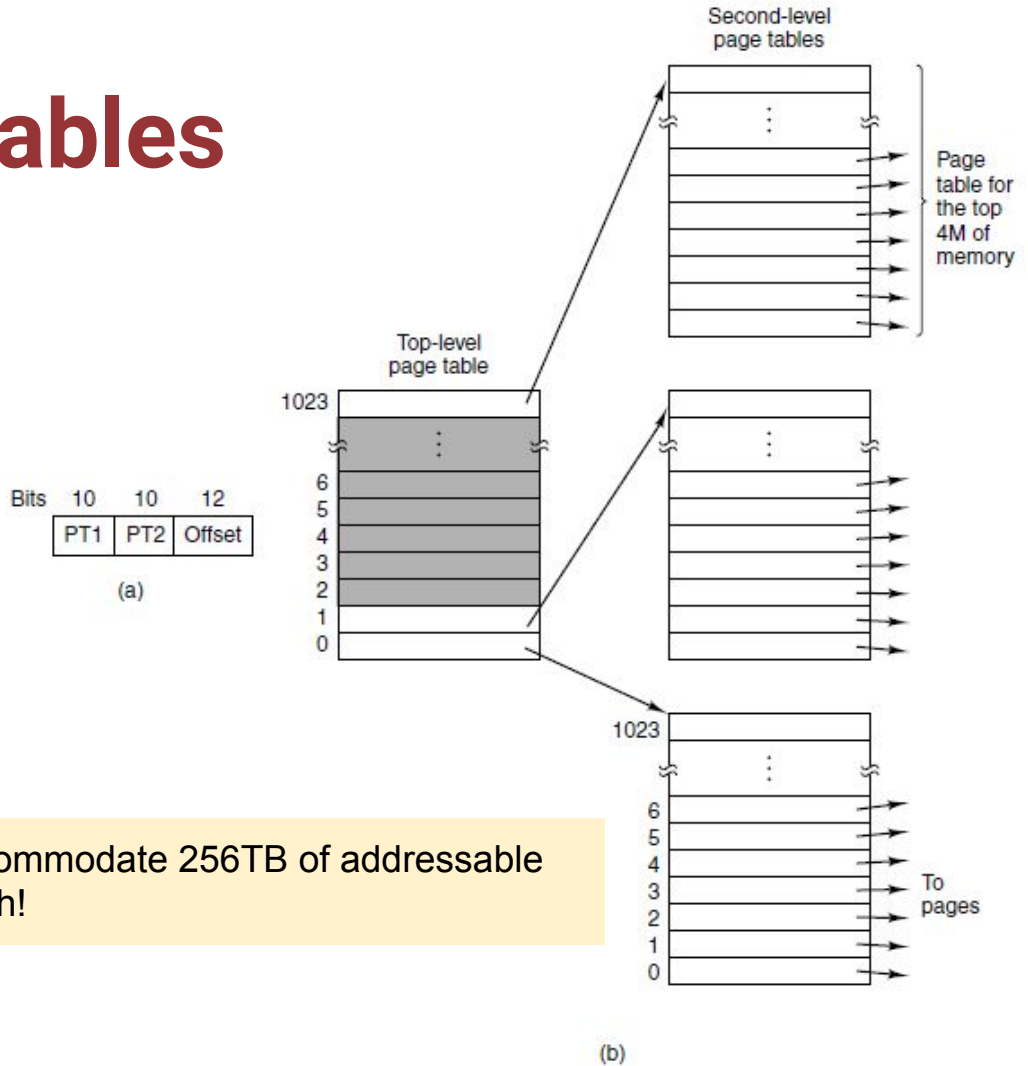
Page tables need 1 entry per page

- A 32-bit address space, using 4KB pages requires 1 million pages -> 1MB
- A 64-bit address space is ridiculously larger.
- Compounding this problem: We need a page table for every process!

We can’t store all these in contiguous memory!

Multi-level page tables

By introducing a hierarchy, we can avoid holding the entire page table in memory!



Modern processes employ this (4 levels) to accommodate 256TB of addressable space. It comes with a performance cost though!

Still… this is really a lot of space

- Each process has a unique virtual address space
- The vast majority of virtual addresses are unused
- # of Physical Frames is **fixed** and limited though…

Let's have ONE table, indexed by FRAME.

Each entry will tell is which process, and which page within that process, is stored in the given frame.

To resolve a virtual address, **search** the table!

Search vs. Index

- For standard page tables, the page number is an **index** into the table
  - Virtual Address > Page # is trivial
  - Index lookup is trivial
- For inverted page table, we must actually search for the right entry!
  - This can be SLOW - we'd need a really nice TLB!
  - We can also use hashing to speed up search.

Up next...

We have one last pending issue:

- For each process, only a few pages are going to be in physical memory
  - Which ones?
  - How many?
  - How do we decide?