



Lecture 8

Paging

Operating Systems



Virtual Memory

- The combined size of
 - Program
 - Data
 - Stack
- May exceed the amount of Physical memory available (RAM)
- VM can work in multiprogramming
 - Bits and pieces of many programs in RAM at once

Virtual Memory

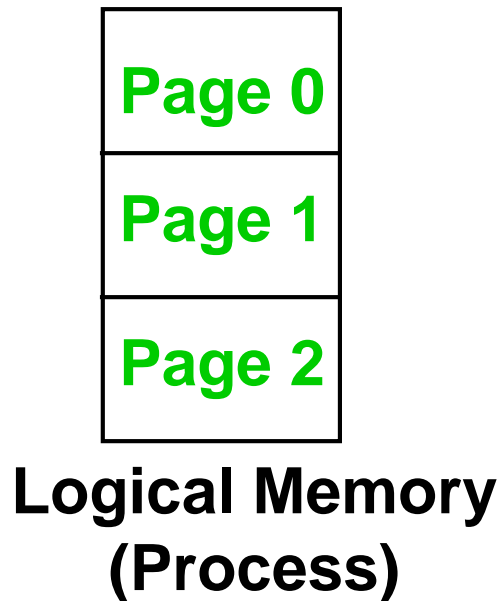
- A program generates *Virtual addresses* of *Virtual Address Space*
- Which are then mapped to *Physical addresses* of *Physical Address Space*
- In VM *Virtual Address Space* can be larger than *Physical Address Space*

Paging

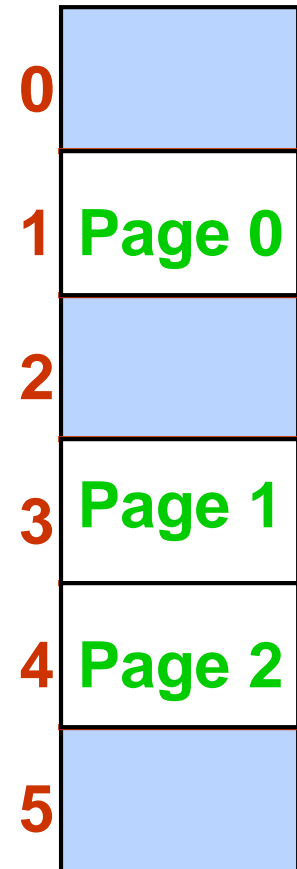
- The Address space can be non-contiguous in Physical Memory
- *Virtual Address Space* is divided into units called *Pages or Virtual Pages*
- *Physical Address Space* is divided into units called *Frame or Page Frames*
- *Pages* and *Frames* are always of the same size

Paging

- Permits the address space to be non-contiguous
- Physical memory is divided into same sized blocks called **Frames**
- Logical memory is divided into same sized blocks called **Pages**



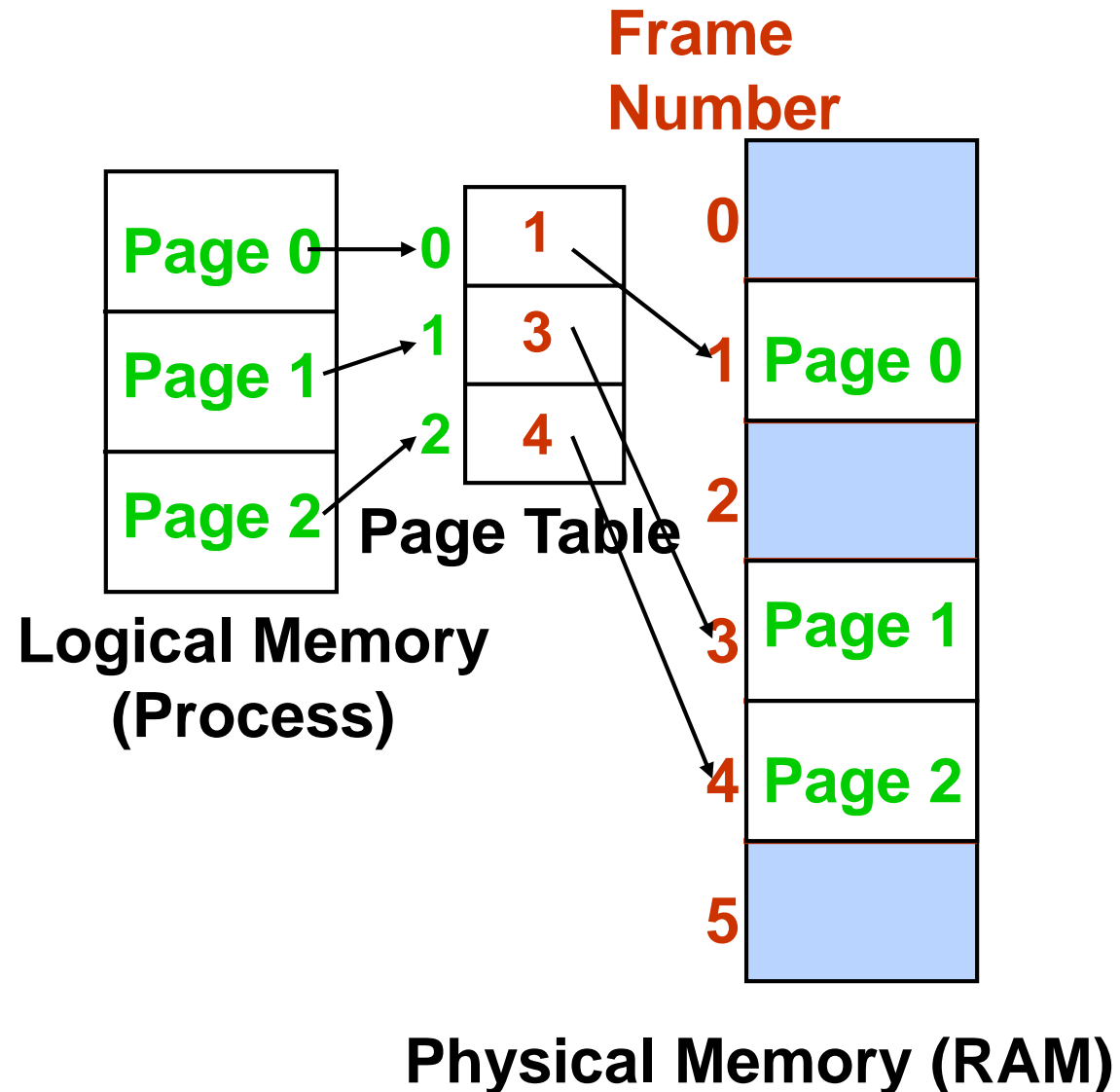
Frame
Number

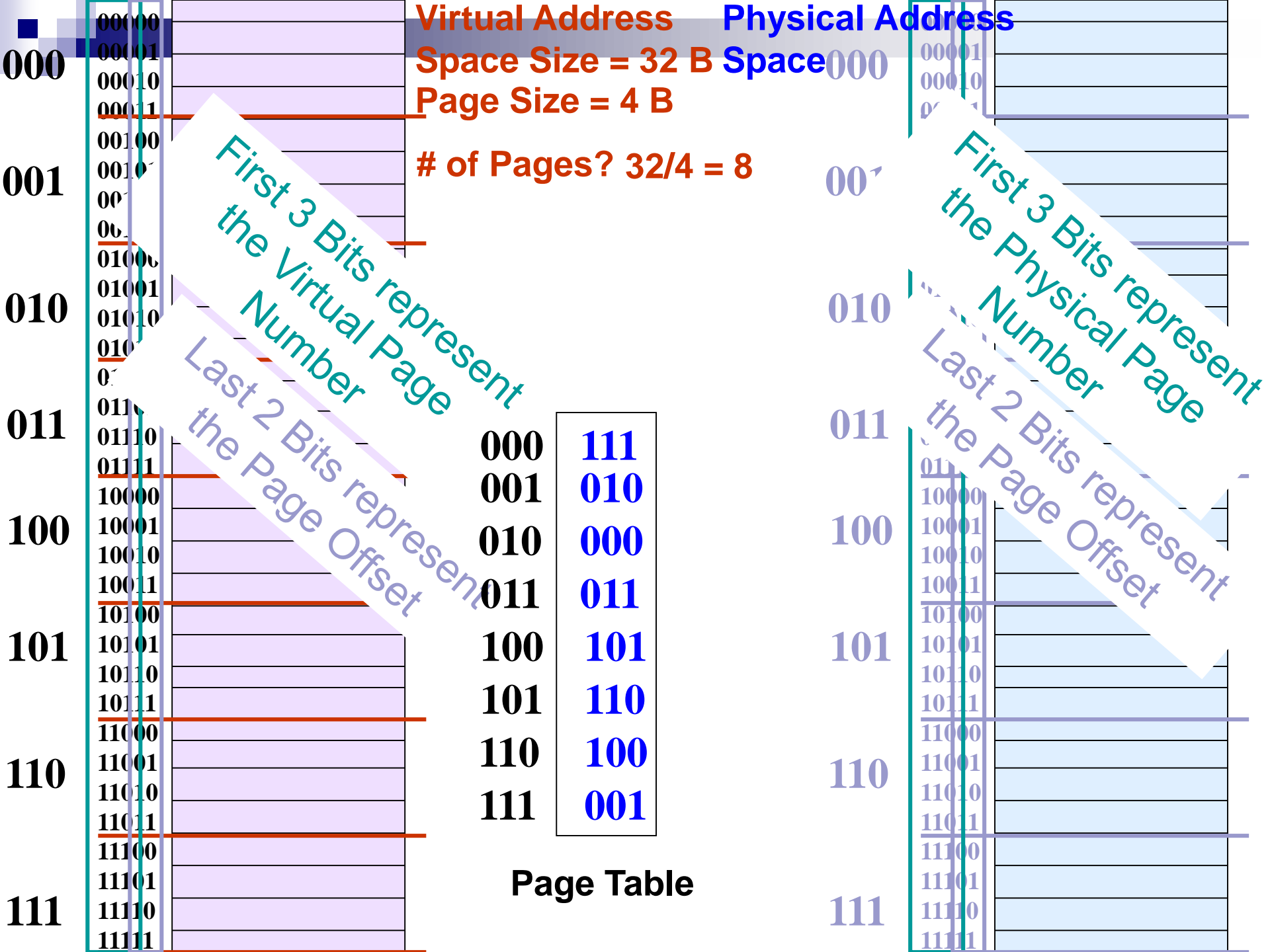


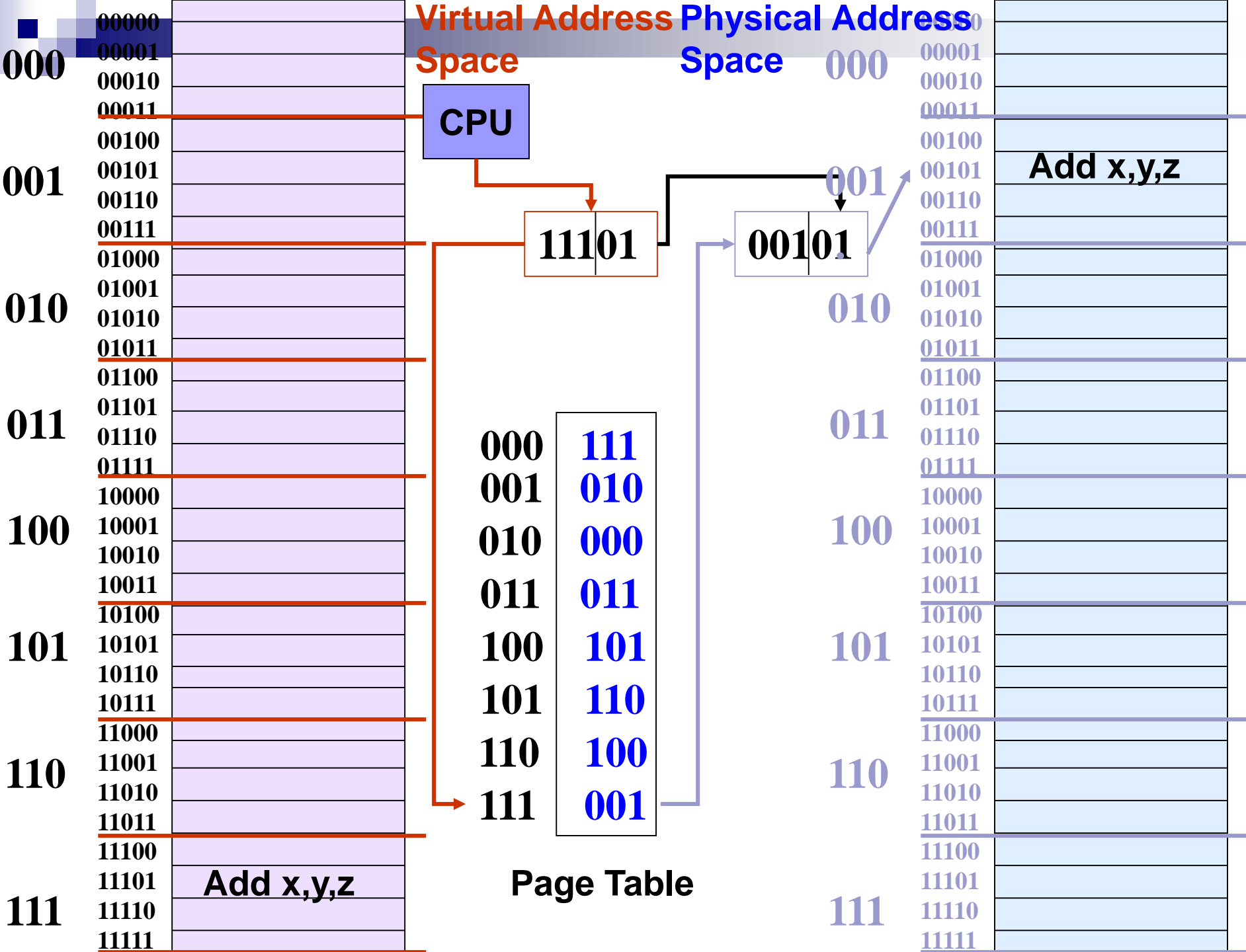
Physical Memory (RAM)

Paging

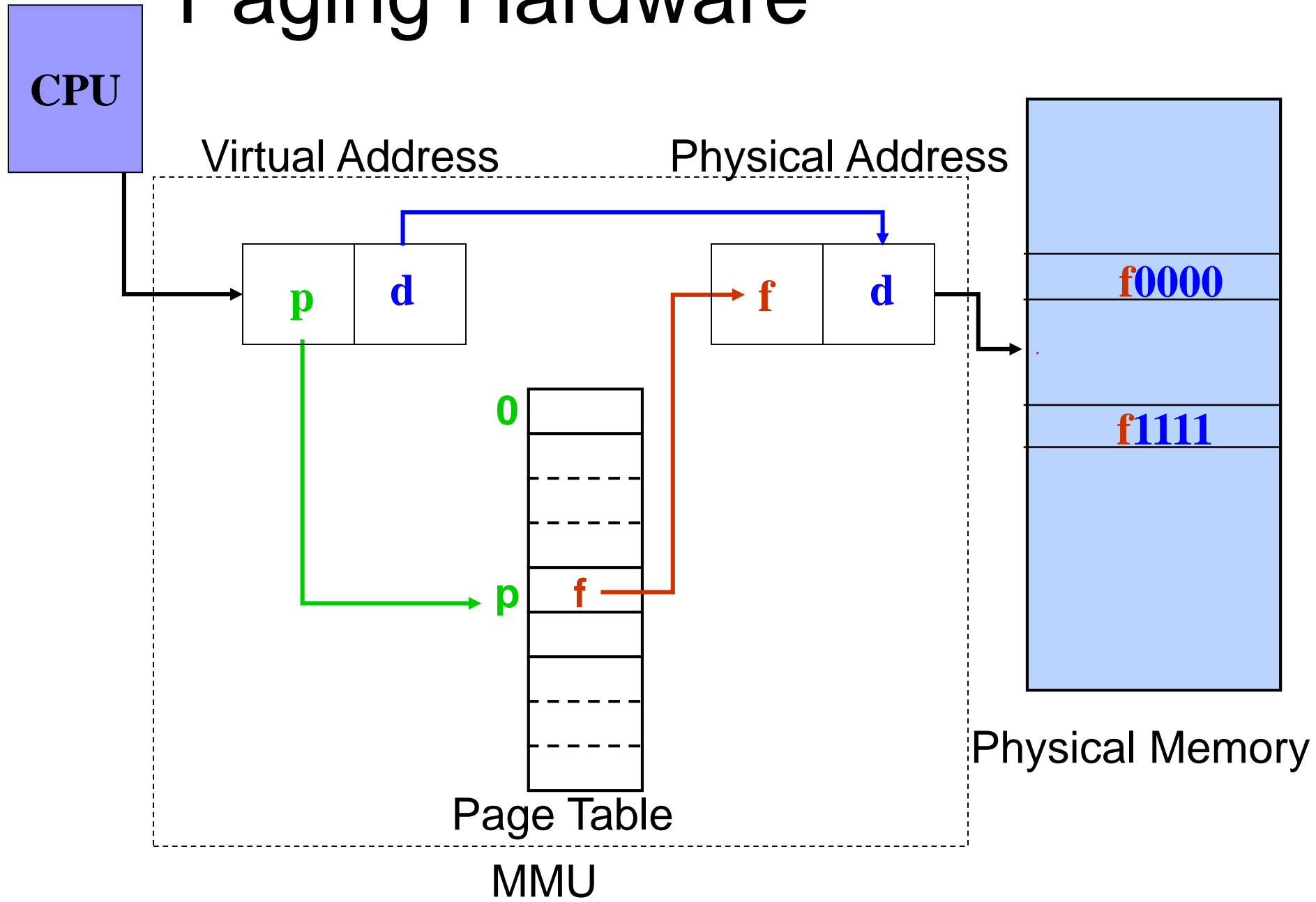
- We need to keep track of which Page is loaded in which Frame







Paging Hardware



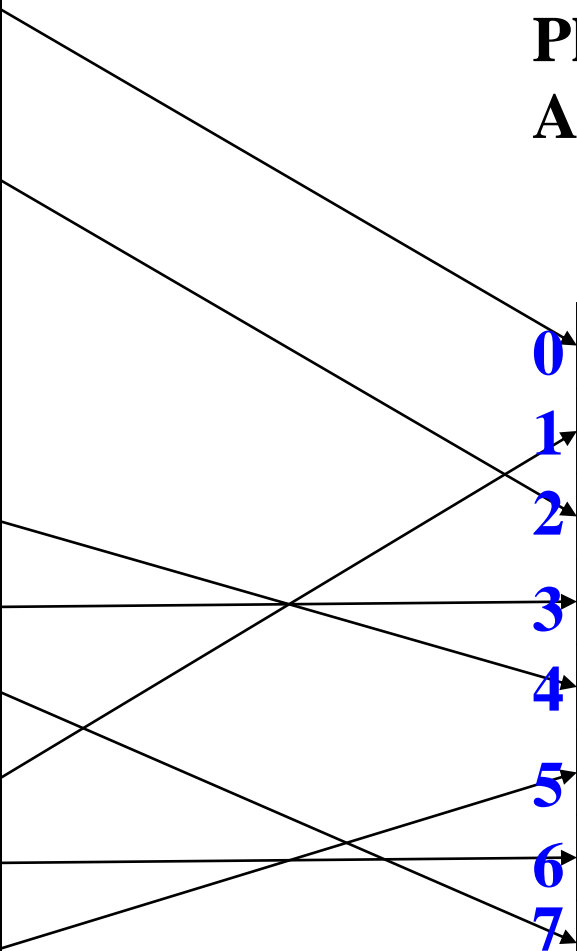
**Virtual
Address
Space**

0k – 4k	X
4k – 8k	X
8k – 12k	X
12k-16k	X
14k-20k	0
20k-24k	X
24k-28k	2
28k-32k	X
32k-36k	X
36k-40k	X
40k-44k	4
44k-48k	3
48k-52k	7
52k-56k	1
56k-60k	6
60k-64k	5

Virtual Address Space can be
larger than *Physical Address
Space*

**Physical
Address Space**

0		0k – 4k
1		4k – 8k
2		8k – 12k
3		12k-16k
4		14k-20k
5		20k-24k
6		24k-28k
7		28k-32k



Paging

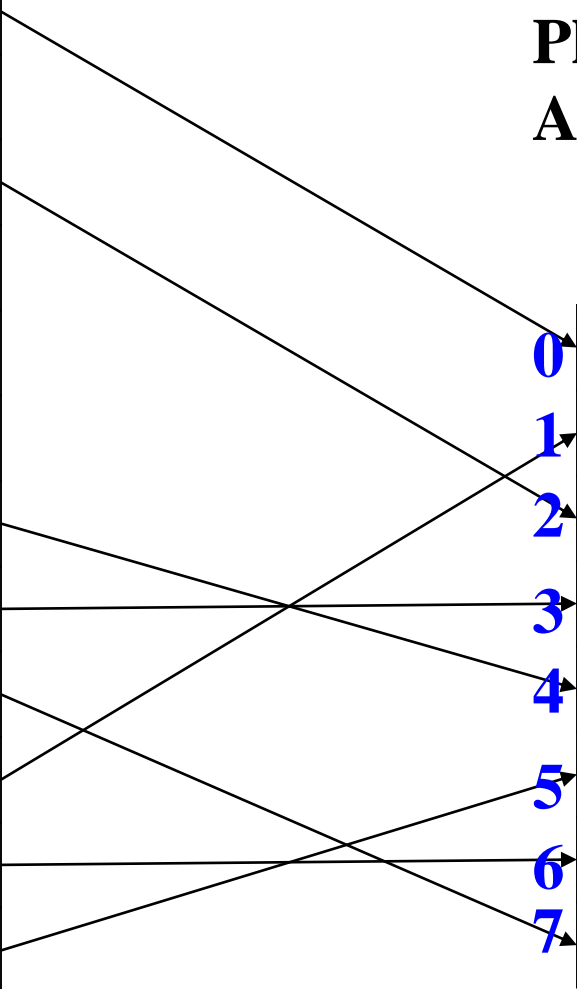
- Only 8 pages are mapped
- What if a Virtual Address between 8k – 12k is generated?
- A Present/Absent bit is kept for each Page
- Present/Absent bit = 1
 - Page is Present in Physically Memory
- Present/Absent bit = 0
 - Page is **not** Present in Physically Memory

Paging

Virtual Address Space	0k – 4k	X	0
	4k – 8k	X	0
	8k – 12k	X	0
	12k-16k	X	0
	14k-20k	0	1
	20k-24k	X	0
	24k-28k	2	1
	28k-32k	X	0
	32k-36k	X	0
	36k-40k	X	0
	40k-44k	4	1
	44k-48k	3	1
	48k-52k	7	1
	52k-56k	1	1
	56k-60k	6	1
	60k-64k	5	1

Physical
Address Space

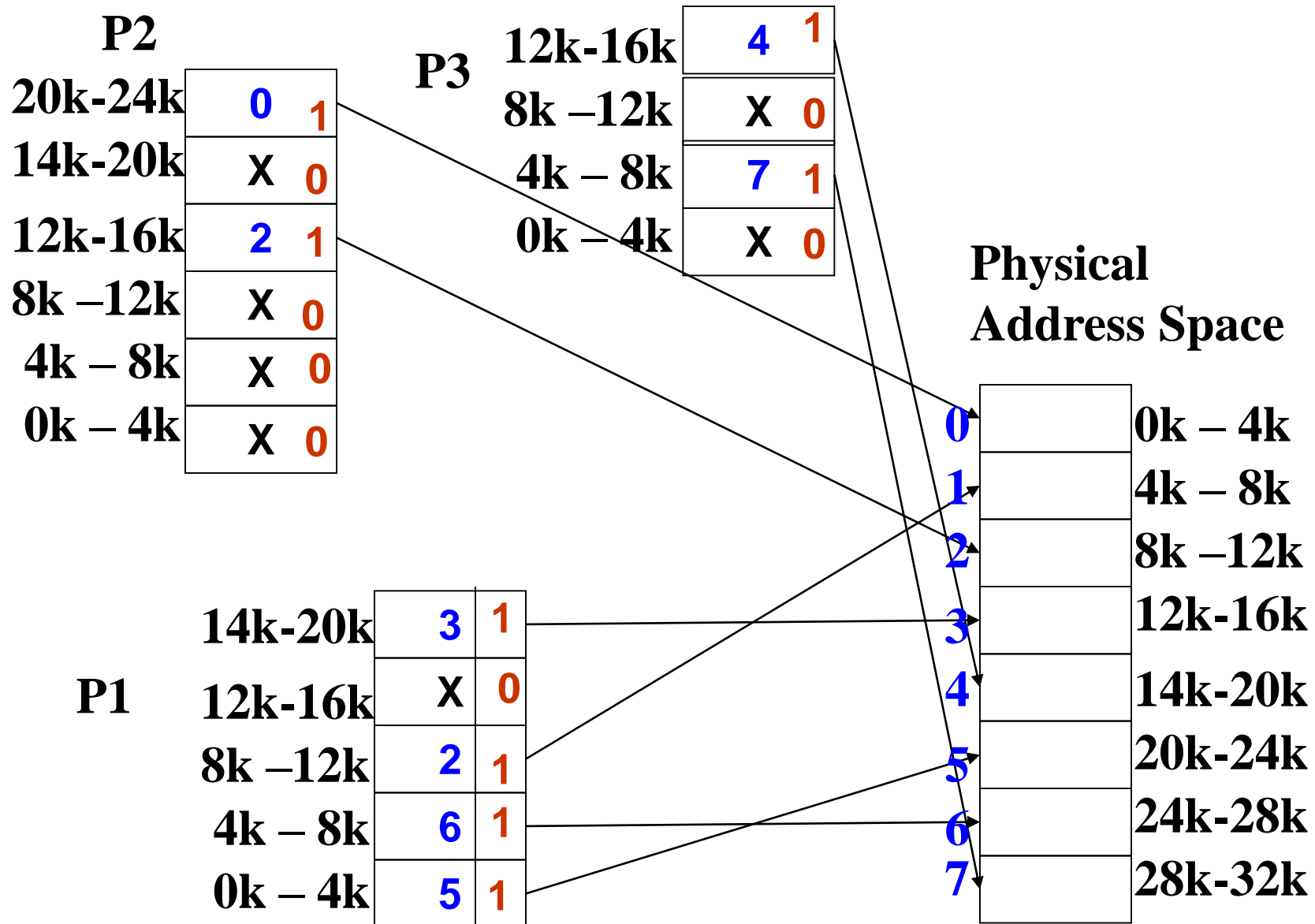
0	0k – 4k
1	4k – 8k
2	8k – 12k
3	12k-16k
4	14k-20k
5	20k-24k
6	24k-28k
7	28k-32k



Paging

- If the Program generates the Virtual Address of a page that is unmapped
 - Present/Absent bit = 0
- MMU generates an Interrupt called *Page Fault*
- This invokes the Operating System
 - OS swaps out one of the Page from RAM
 - Swaps in the required page
 - Updates the Page Table

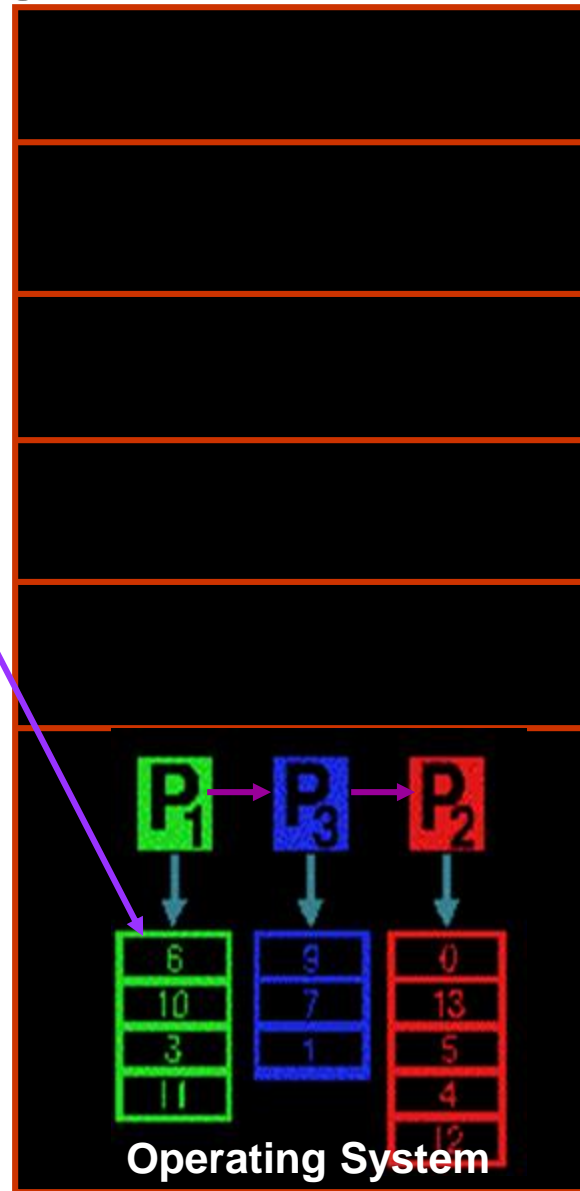
Multiprogramming with Paging



Where is the Page Table Stored?

**Page Table Base Register
(PTBR)**

*Points to the Page
Table of Running
Process*





Page Table

- The only hardware required is a register
- Points to the starting address of the Page table
- Whenever, there is a context switch
- The register points to the starting address of the new process's page table



Page Table

- Two major issues are:
 - The Page table entries can be extremely large
 - The mapping must be fast

PT in Memory (Addresses)

- CPU generates address for
 - Fetching/Storing Variables
 - Fetching Instructions
 - Jumping to labels (in case of loops)
 - etc
- Each address generated by CPU now requires two memory references
 - 1. To access the Page Table
 - 2. To access the actual address of the instruction/variable/label



RAM Vs. Registers

- Remember!!!
- If referring to RAM is like walking
- Then referring a Register is like flying an aeroplane



Solution

- Keep a number of fast hardware registers
- When a process starts
- OS loads the registers with the Process's entire Page table
- Process's Page table is available in RAM
- During execution no memory references are required for Page tables
- But is expensive, if page size is large

Translation Look-aside Buffer (TLB)

- A better solution is TLB:
 - A special, small, fast-lookup Hardware cache
- The TLB contains a few Page-Table entries
- Whenever, a virtual address is generated:
- The page number is presented to the TLB
- If the entry is found
 - Its used to access memory
- Else (TLB miss)
 - The actual Page table stored in RAM is used
 - We may would like to add this Page Table entry into TLB

Hit Ratio

- Percentage of times a particular page is found in the TLB
- 20 ns to Search TLB
- 100 ns to Access Memory
- → 120 ns to Access Memory when the Entry is in the TLB
- → 220 ns to Access Memory when the Entry is not in the TLB
- If Hit-Ratio = 80%
 - Effective access time = $0.80 * 120 + 0.20 * 220 = 140$ ns
- If Hit-Ratio = 98%
 - Effective access time = $0.98 * 120 + 0.02 * 220 = 122$ ns

Translation Look-aside Buffer (TLB)

- All entries in TLB may correspond to a single process
 - The currently running process
 - TLB must be erased/flushed on every context switch
- Or it may have entries from multiple processes
 - We need to distinguish between the Page Table entries from multiple processes
 - So, each entry in TLB will also have a Process Identifier

Logical/Physical Address

- There are 2^k pages
 - k bits
- How many bits are required to save Page number?
- If the length of a page is 2^n
- How many bits are required to save Page Offset?
 - n bits
- Total Bits for storing address?
 - $k + n$
- Maximum Possible Addresses?
 - 2^{k+n}

Page Table Size

- If we have

- 32-bit addresses

- Page size = 4 KB = 2^{12} B

- Possible Pages??

- $2^{32}/2^{12}$

- = 2^{20}

- = 10^6

- = 1 Million

- → We need 1 Million Page Table entries in RAM all the Time!!!



Page Table Size

- Increasing the Page size will decrease the number of pages
- ...but...
- More Internal Fragmentation

Example: suppose page size is 4 bytes.

virtual memory

a
b
c
d
e
f
g
h
i
j
k
l

4
3
1

page table

physical memory

0	
4	i
	j
	k
	l
8	
12	e
	f
	g
	h
16	a
	b
	c
	d

Where is virtual address 6? 9?



A system implements a paged virtual address space for each process using a one-level page table. The maximum size of an address space is 16 megabytes. The page table for the running process includes the following entries:

<u>page</u>	<u>frame number</u>
0	4
1	8
2	16
3	17
4	9

The page size is 1024 bytes and the maximum physical memory size of the machine is 2 megabytes. Assuming two bits for protection and reference etc. How many bits are required for each page table entry? What is the maximum number of entries in a page table? How many bits are there in a virtual address? To which physical address will the virtual address 1524 translate to? Which virtual address will translate to physical address 10020?

What to fetch?

- Page selection: when to bring pages into memory
 - Like all caches: we need to know the future.
- Easy load-time: demand paging
 - Load initial page(s). Run. Load others on fault.
 - Most systems do some sort of variant of this
- Pre-paging. Get page & its neighbors (why?)

ld init pages  ld page  ld page  ld page  ...

What to eject & when?

- Random: pick any page.
 - Pro: good for avoiding worst case
 - con: good for avoiding best case
- FIFO: throw out oldest page
 - fair: all pages get equal residency
 - inefficient: ignores usage.
- MIN (optimal):
 - throw out page that will be not used for longest time.
 - Impractical, but good yardstick
- Least recently used (LRU).
 - throw out page that hasn't been used in the longest time.
 - Past = future? LRU = MIN.

Reference string: A B C A B D A D B C B

	FIFO	MIN	LRU																																																																																													
	<table><tr><td></td><td></td><td></td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>D</td><td>B</td><td>C</td></tr><tr><td>D</td><td>A</td><td>C</td></tr><tr><td>D</td><td>A</td><td>C</td></tr><tr><td>B</td><td>D</td><td>A</td><td>B</td></tr><tr><td>C</td><td>C</td><td>A</td><td>B</td></tr><tr><td>B</td><td>C</td><td>A</td><td>B</td></tr></table>				A	B	C	A	B	C	A	B	C	D	B	C	D	A	C	D	A	C	B	D	A	B	C	C	A	B	B	C	A	B	<table><tr><td></td><td></td><td></td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>				A	B	C	A	B	C	A	B	C																			<table><tr><td></td><td></td><td></td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>A</td><td>B</td><td>C</td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>				A	B	C	A	B	C	A	B	C																		
A	B	C																																																																																														
A	B	C																																																																																														
A	B	C																																																																																														
D	B	C																																																																																														
D	A	C																																																																																														
D	A	C																																																																																														
B	D	A	B																																																																																													
C	C	A	B																																																																																													
B	C	A	B																																																																																													
A	B	C																																																																																														
A	B	C																																																																																														
A	B	C																																																																																														
A	B	C																																																																																														
A	B	C																																																																																														
A	B	C																																																																																														
A B C																																																																																																
A																																																																																																
B																																																																																																
D																																																																																																
A																																																																																																
D																																																																																																
B																																																																																																
C																																																																																																
B																																																																																																

Faults:
FIFO 7
MIN 5
LRU 5