Paging

How to make physical memory more flexible?

By Matthew Tancreti
For COM S 352
Iowa State University

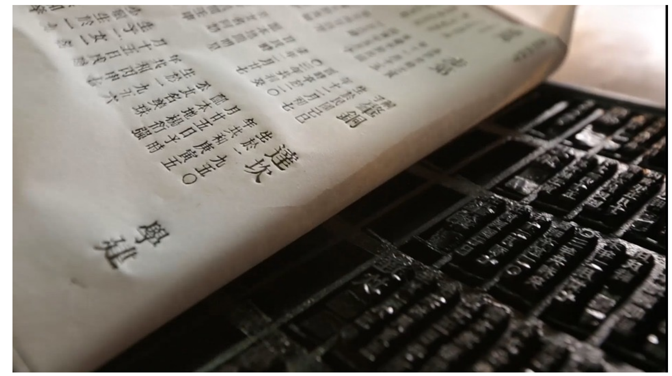
Paging

Segmentation made code, stack and heap independently relocatable

Segments can become arbitrarily large contiguous regions of memory

Resulted in **external fragmentation**

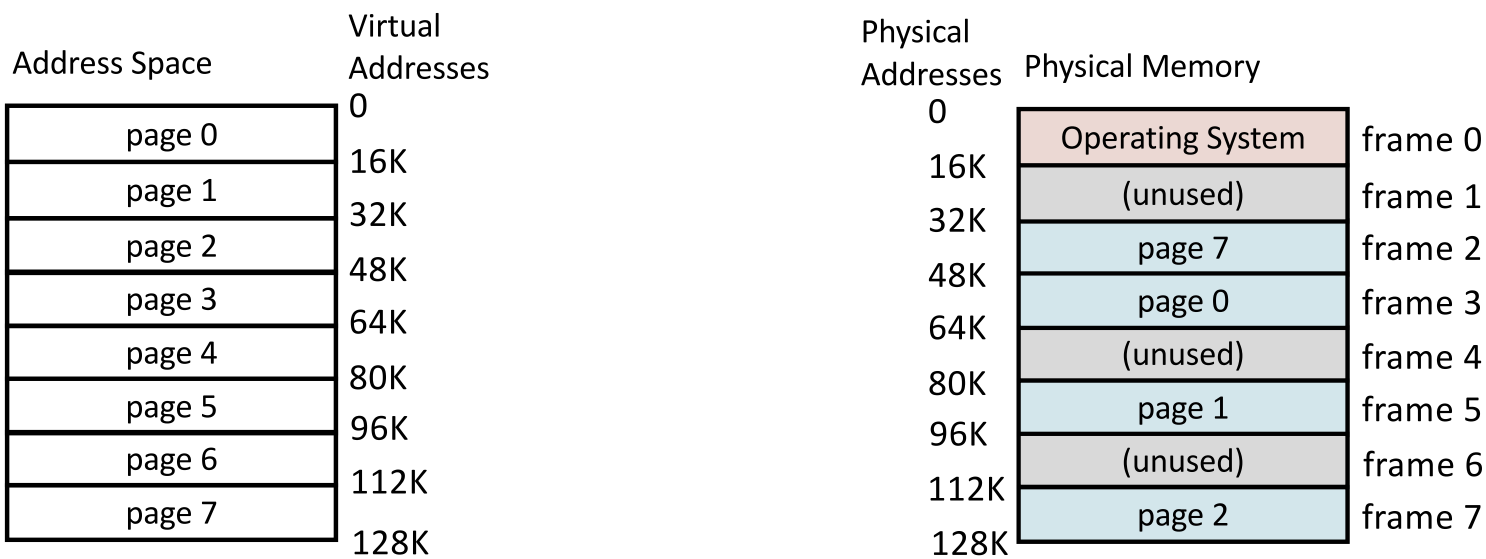
What if we divide the address space into equal size pages?



Movable type printing invented by Bi Sheng 1031-1095 [source]

Paging

Address space divided into equal sized pages that can be stored in frames



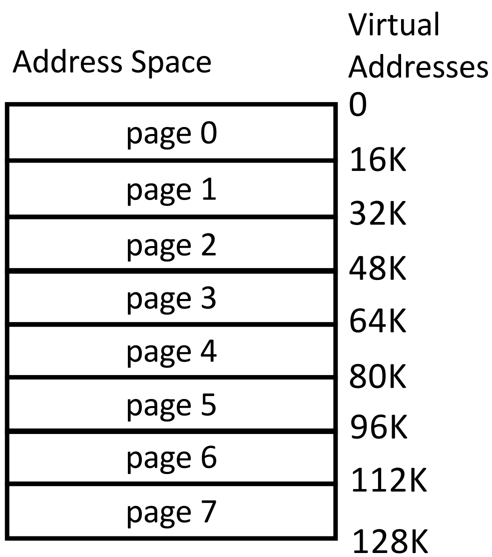
Page Table

One page table for each process

Virtual Page Number (VPN) is the index of the table

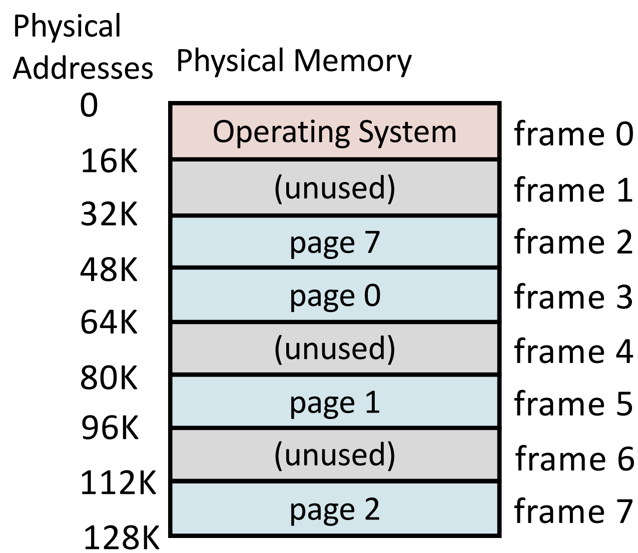
Physical Frame Number (PFN) points to the frame in physical memory

Valid bit indicates if table entry is valid (not all of address space needs to be mapped)



Page Table

VPN	PFN	valid
0	3	1
1	5	1
2	7	1
3	-	0
4	-	0
5	-	0
6	-	0
7	2	1



Virtual Address Bits

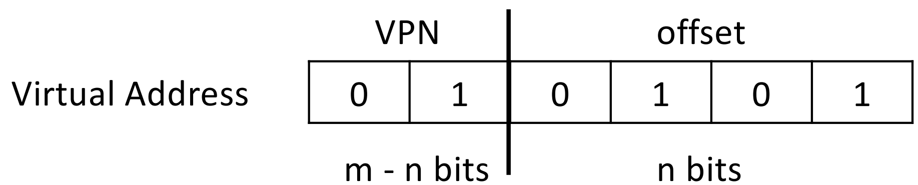
Virtual address divided into VPN and offset

	VPN		offset			
Virtual Address	0	1	0	1	0	1

```
VPN = VirtualAddress >> NUM_OFFSET_BITS

Offset = VirtualAddress & OFFSET_MASK
```

Virtual Address VPN Bit Size



If total address space is 2^m bytes and page size is 2^n bytes then

VPN is m-n bits

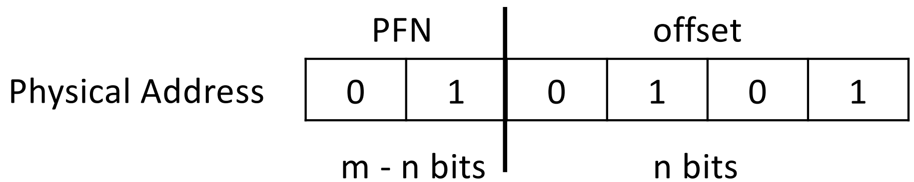
offset is n bits

Question: If address space is 1GB and page size is 4KB how many bits is the VPN?

Answer: 1GB = 2^{30}, 4KB = 2^{12}, therefore VPN is 30 - 12 = 18 bits

Check: 4KB * 2^{18} = 1GB

Physical Address Bits



If physical memory is 2^m bytes and frame (also page) size is 2^n bytes then

PFN is m-n bits

offset is n bits

Therefore, calculate physical address as:

```
PhysAddr = PFN * frame_size + offset
```

or in binary arithmetic:

```
PhysAddr = (PFN << NUM_OFFSET_BITS) | offset
```

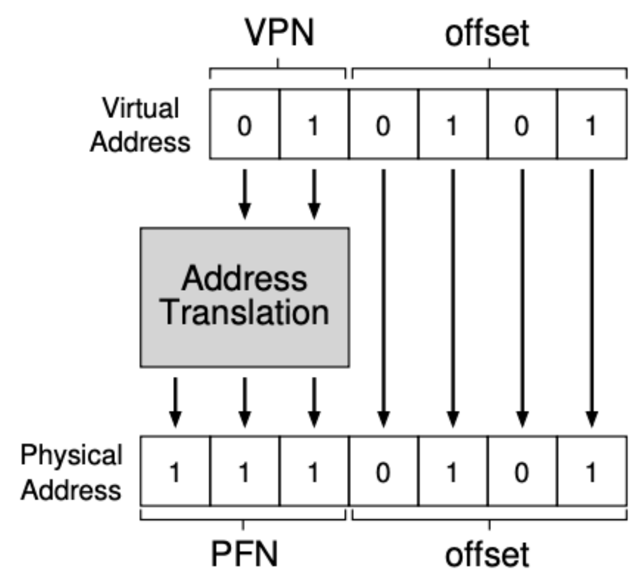
Keep in mind, total physical memory and address space size may not be the same

Question: Assume the page size is 4KB, what is the address for frame 10 and offset 128?
Answer: 4KB * 10 + 128 = 41,088

Address Translation

Steps performed **in hardware** (MMU):

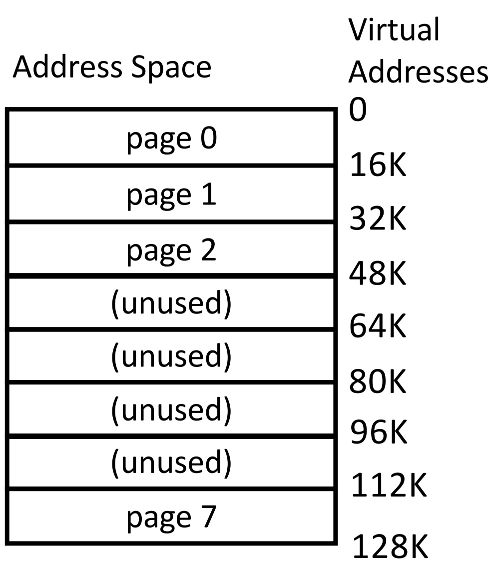
1. Compute page number
   `VPN = VirtualAddress >> NUM_OFFSET_BITS`
2. Look up page in page table to find frame number
3. Confirm that access to page is allowed (e.g., valid bit set)
4. Compute physical address
   PhysAddr = (PFN << NUM_OFFSET_BITS) | offset



Unused Pages

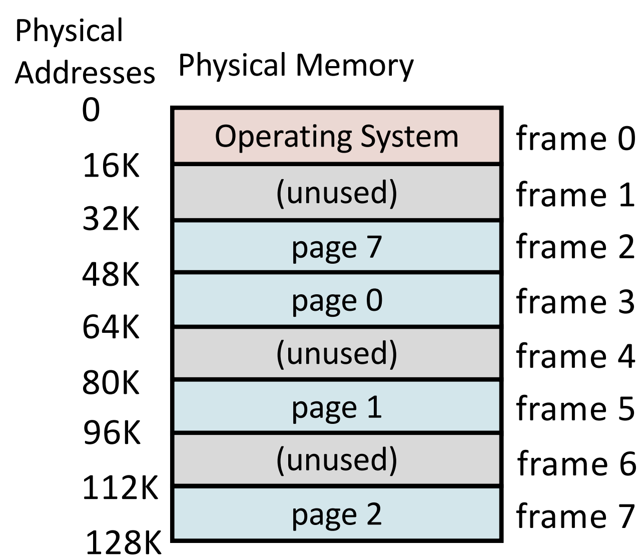
Unused pages don’t need to be mapped to physical memory

Process is unaware of pages, when it requests more memory allocated for heap or stack the OS maps pages as needed



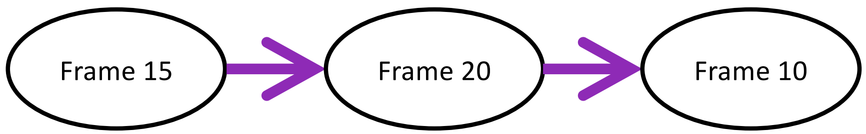
Page Table

VPN	PFN	valid
0	2	1
1	5	1
2	7	1
3	-	0
4	-	0
5	-	0
6	-	0
7	2	1



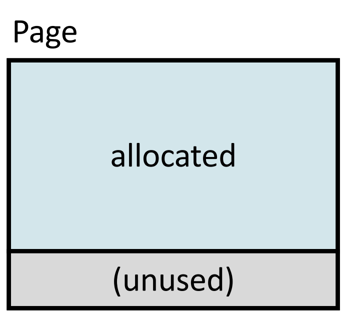
Free Memory Management

OS needs to know which frames are not in use, simple method is linked list



No external fragmentation - no unusable gaps between frames

Pages can have **internal fragmentation** - unused portion of page



Page Size

Tradeoff

- Small page size means bigger page table (more page table entries)
- Bigger page size means more internal fragmentation

Typical page size is 8KB in Linux

Concern

Each process has its own page table which the OS stores in frames in physical memory

Linux page size is 8KB, on pyrite we saw the address space is 140TB

~17 million page table entries for every process!

Linux reduces size with multi-level (3-level tree structure) page tables

Page table lookup is slow, every memory access requires additional memory access(es)! How to speed up memory?