Lesson Summary

- paging is great
  - No external fragmentation
  - Easy to share pages among processes
- mechanism
  - Each process as a page table - Each page table entry maps a logical page to a physical frame - Each page table entry has a valid bit - Address translation is based on the page table - The OS manages the list of free frames, and gives out frames to processes

Conclusion from Main Memory

- Assumption: Each process is in a contiguous address space
  - Good - Address virtualization is simple (base register)
  - Bad - No "best" memory allocation strategies
  - Worse - Fragmentation can be very large
  - Even Worse - There can be process starvation in spite of sufficient available RAM due to fragmentation
- Conclusion: base assumption is flawed

Computational Complexity

P Problem	NP Problem	NP Hard Problem
A decision problem where the "yes" or "no" answer can be decided in polynomial time	An optimization problem where a solution can be verified in polynomial time e.g., traveling salesman	problem which is at least as hard as the hardest NP problems; suspected that there are no polynomial-time algorithms that can solve NP-hard problems, but this has never been proven. EX: Bin Packing

Conclusion for OS Design

Variable-size chunks are a bad idea

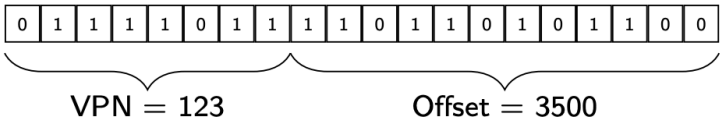
Paging

- Pages
  same size chunks
  bin packing for this special case is not NP Hard
- Paging
  a policy of process splitting address space into fixed size pages

- Frames
  physical memory is split in fixed size frames and each frame can hold a page
  like that we have non contiguous memory allocation
- Virtual Page Number (VPN)
  a page is virtual (logical
- Physical Frame Number
  a frame is physical (PFN)
- still have internal fragmentation but never external fragmentation

(Virtual) Page Number

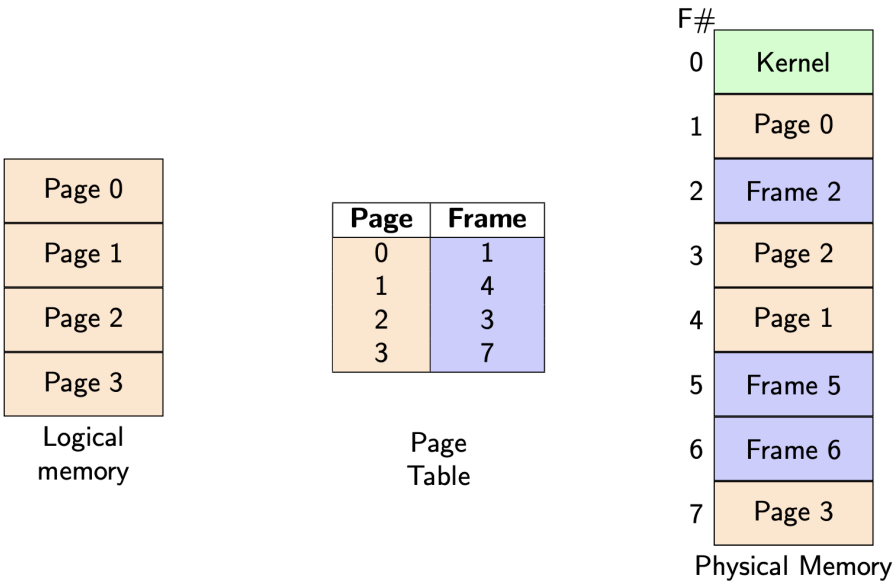
- virtual address issued by the CPU are split into two parts
  - p: virtual logical page number
  - d: offset within page
  - think of it as "read the value at offset d in page p"



Page to Frame Translation

- VPN has to be translated to corresponding PFN
- more sophisticated address translation scheme than what we saw in the previous module for contiguous memory allocation
- recall: instead of "read the value at address x", a program program does "read the value at offset d in page p"
- Therefore we need to keep track for each process of the mapping between each of its pages and the physical frame that page is in
- each process has a page table

Page Table Example



Page Size

- page size is defined by architecture
  x86-64: 4 KiB, 2 MiB, and 1 GiB
  ARM: 4 KiB, 64 KiB, and 1 MiB
- The page size in bytes is always a power of 2
- The **pagesize** command gives you the page size on UNIX-like systems
- For instance, on my laptop: 4096
- You can easily reconfigure your OS to use a different page size
- But that page size has to be supported by the hardware

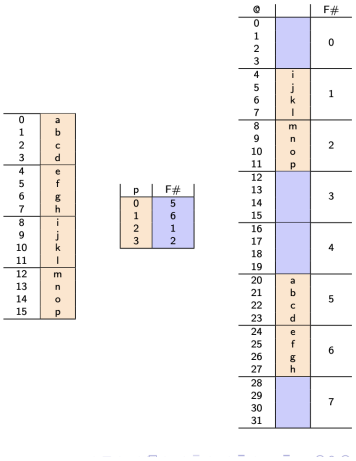
Address Decomposition

- Say the size of the logical address space is 2^m bytes
- Say a page is 2^n bytes (n < m) then
  - The n low-order bits of a logical address are the offset into the page (offset ranges between 0 and $2^n - 1$, each one corresponding to a byte in the page)
  - remaining m-n high order bits are logical page number

Example of Page Size

Example (continued)

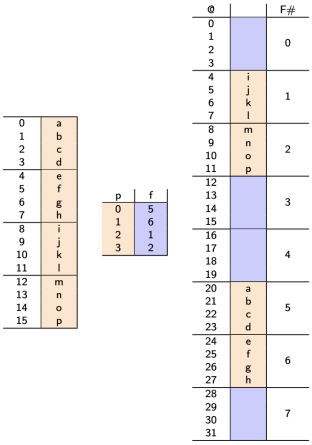
- $2^5 = 32$ bytes of physical RAM
- 5-bit physical addresses
- 4-byte physical frames
- 8 frames in RAM
- 4-byte pages
- Let's say we have a process with a 16-byte address space
- How many pages it this address space?
  $\frac{16(\text{bytes})}{4(\text{bytes/page})} = 4$ pages
- Say the address space contains values a, b, . . ., p
- Say the OS has placed Page 0 into Frame 5, Page 1 into Frame 6, Page 2 into Frame 1, and Page 3 into Frame 2.
- Therefore, the OS will have created a page table with 4 entries for that process
- How many bits in a virtual address for that process?
  4 bits (because we have 2^4 bytes)
  2-bit page index
  2-bit offset in the page



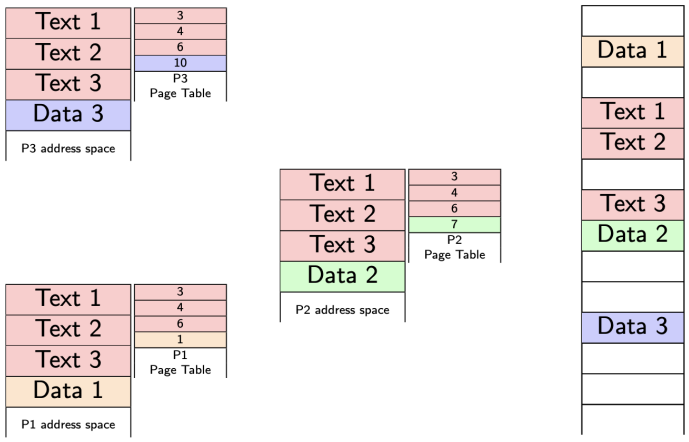
Example (the end)

- What is the logical address of g?
  logical @ = (page nbr)*(page size) + offset
  Page=1; Offset=2
  (often written 1:2)
  $= 1 \times 4 + 2 = 6$

- What is the physical address of g?
  physical @ = (frame nbr)*(frame size) + offset
  Page 1 is in Frame 6,
  same offset: 2,
  therefore: $6 \times 4 + 2 = 26$



Sharing Memory Pages Across Processes? EASY!



Just insert page table entries that point to the same physical frames!

Valid Bit

- a page table contains entries for all possible pages (up to the maximum allowed number of pages for a process, as defined by the OS)
- in reality it looks like

Page Table

P0	14
P1	13
P2	18
P3	20
P4	not used (yet)
P5	not used (yet)
P6	not used (yet)
P7	not used (yet)

- each page entry is augmented by a **valid bit**
- Set to valid if the process is allowed to access the page
- otherwise set to invalid

Page Table

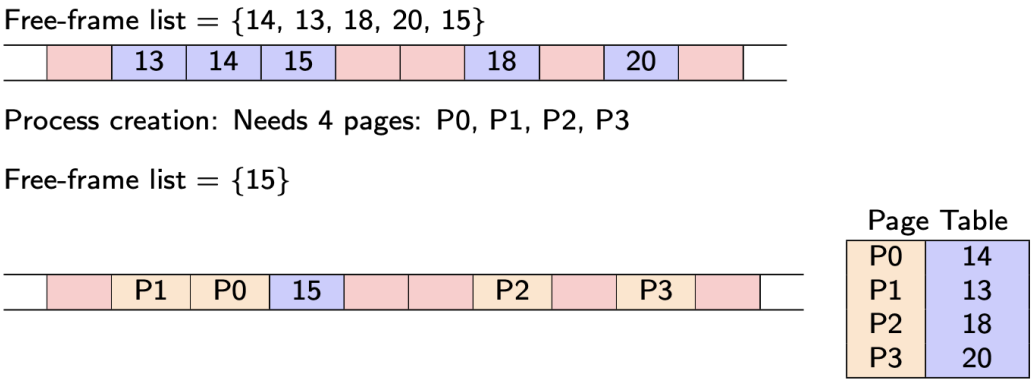
P0	14	✓
P1	13	✓
P2	18	✓
P3	20	✓
P4	xx	-
P5	xx	-
P6	xx	-
P7	xx	-

What about fragmentation?

- No external fragmentation!!
- Only internal fragmentation
  - Worse Case
    A process address space is n pages plus 1 byte
  - Average Case
    Uniform distribution of address space sizes: 50%
- Using smaller pages reduces internal fragmentation
- large pages have advantages
  - smaller page tables
  - less frequent page table look up
  - Loading one large page from disk takes less time than loading many small ones

Frames Management

- OS needs to keep track of frames
  - Which frames are used (and by which processes?)
  - Which frames are free?
- OS thus has a data structure: the **free frame list**
- Much simpler than a list of holes with different sizes
- When a process needs a frame, then the OS takes a frame from the free frame list and allocate them to a process



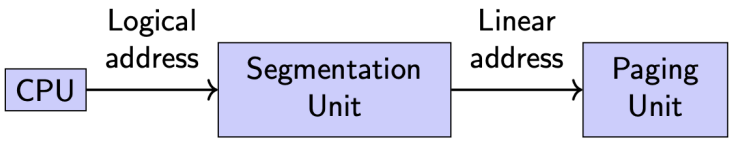
Segmentation and Paging: e.g., IA 32/64

- The Intel architecture provides both segmentation and paging
- A logical/virtual address is transformed into a linear address via segmentation

$$\text{logical address} = (\text{segment selector, segment offset})$$

- linear address is transformed into a physical address via paging

linear address = (page number level-1, p-2, p-3, p-4, offset)



In-Class Exercises

Exercise 1

A computer has 4 GiB of RAM with a page size of 8KiB. Processes have 1 GiB address spaces.

- How many bits are used for physical addresses?
  Physical RAM: 4GiB = 2^{32} bytes
  $\Rightarrow$ 32-bit physical addresses
- How many bits are used for logical addresses?
  Logical Address space: 1GiB = 2^{30} bytes
  $\Rightarrow$ 30-bit logical addresses
- How many bits are used for logical page numbers?
  Page size = 2^{13} bytes
  Number of pages in logical address space: $2^{30}/2^{13} = 2^{17}$
  To address 2^{17} things, we need 17 bits
  $\Rightarrow$ 17-bit logical page numbers
  (and 13-bit offsets)

Exercise 2

Logical addresses are 44-bit, and a process can have up to 2^{27} pages.

- What is the page size?

The address space can have up to 2^{44} bytes
There are up to 2^{27} pages
Therefore, a page is $2^{44}/2^{27} = 2^{17}$ bytes

Exercise 3

On my computer the page size is 16 KiB, and my process' address space is 4GiB.

- How many bits are used for the page number in a logical address?

The address space contains 2^{32} bytes
A page is 2^{14} bytes
Therefore, my address space has $2^{32}/2^{14} = 2^{18}$ pages
Therefore, we need **18 bits** for the page number if a logical address
(and we have 14 bits in the offset

Exercise 4

A computer has 32-bit physical addresses. The logical page number of a logical address is 14-bit. A process can have up to a 2GiB address space. Let's consider a process with currently a 1GiB address space (i.e., it can get up to another 1GiB during execution).

- What is the page size?
  How many bytes in 2GiB (the max address space): 2^{31}
  Therefore: 31-bit logical addresses
  Therefore: 31 - 14 = 17-bit offsets
  Therefore: 2^{17} bytes in a page
  Therefore: 128KiB pages
- How many entries are there in the process' page table?
  The process has a 1GiB = 2^{30}-byte address space
  Number of pages in the address space: $2^{30}/2^{17} = 2^{13}$
  Therefore: there are 2^{13} entries in the page table
  (one entry per page)

Exercise 5

Logical addresses are 40-bit, and a process can use at most 1/4 of the physical RAM.

- How bit is the RAM?

With 40-bit logical addresses, an address space is at most 2^{40} bytes
So the RAM is 4 times as big: 2^{42} bytes
which is 4TiB

- My process has 2^{22} pages, how many bits are used for the "offset" part of logical addresses?

Since we have 2^{22} pages, 22 bits are used for the page number
Therefore 40 - 22 = 18 bits are used for the offset

Exercise 6

- Consider a system with 4-byte pages. A process has the following entries in its page table:

logical	physical
0	4
1	5
2	30

- What is the physical address of the byte with logical address 2?
- The byte with logical address 2 is the 3rd byte in page 0 (because that page contains the bytes at addresses 0, 1, 2, and 3)
- Page 0, according to the page table is in physical frame 4
- The first byte of physical frame 0 is at physical address $4 \times 0 = 0$ (the first byte in physical RAM)
- The first byte of physical frame 1 is at physical address $4 \times 1 = 4$ (the fifth byte in physical RAM)
- ...
- The first byte of physical frame 4 is at physical address $4 \times 4 = 16$
- The 3rd byte of physical frame is thus at address $16 + 2$
- Therefore, the byte at logical address 2 is at physical address **18**

- What is the physical address of the byte with logical address 9?
- The byte with logical address 9 is in page 9 / 4 = 2 (integer division)
- Its offset in that page is 9 % 4 = 1
- Page 2 is in frame 30
- Therefore, the byte at logical address 2 is at physical address $30 \times 4 + 1 = 121$