Virtual Memory

Instructor: Linyi Li
Slides adapted from Dr. B. Fraser

Topics

1) How can each process have its own memory space?
2) How can the OS allocate memory to processes?
3) What if we run out of memory?

Context:

What is the problem we are trying to solve?

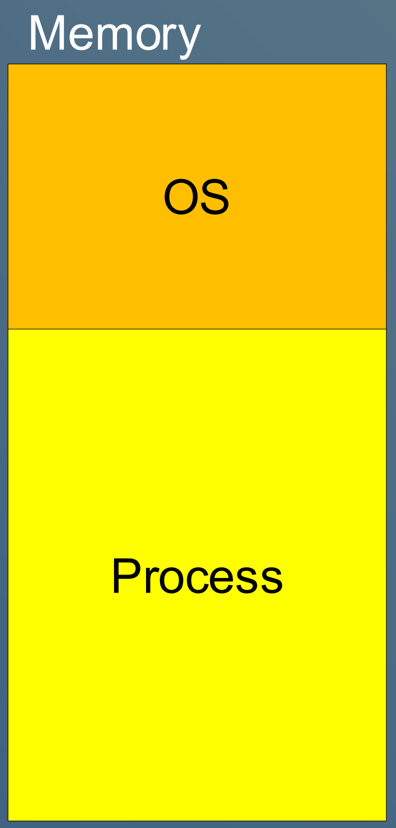
Details

- Virtual memory is one of the most important OS concepts.
  - It is also a good example that shows
..

- Can find more info in OSTEP book
  (more depth than we require)
  - Chapter 13 The Abstraction: Address Spaces
https://pages.cs.wisc.edu/~remzi/OSTEP/vm-intro.pdf
  - Chapter 15 Mechanism: Address Translation
https://pages.cs.wisc.edu/~remzi/OSTEP/vm-mechanism.pdf
  - Chapter 18 Paging: Introduction
https://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf
  - Chapter 16 Segmentation
https://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf

Memory Layout in the Early Days

- The entire memory was divided into two: OS and program.
  - Could run only a single program.
  - However, there were many users who wanted to run their own programs!
  - Users could not share the machine, lead to low utilization.



Early Memory Sharing Attempt

- Memory was divided into
  ..
- Could run multiple processes:
  ..
- Problems
  - Each process could only use a fixed (small) size memory region.
  - ..

a “bad” pointer in one process could access another process's memory.

Understanding Memory

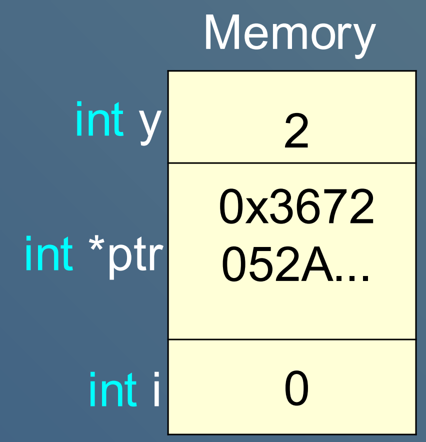
Address-Based Memory Operations

- Variables are a convenience for programmers:
  - ..
  - Most instruction reads from or write to memory.

- Random Access Memory (RAM)
  - ..
  - Unlike a hard-drive (disk) which spins like a record / CD / DVD:
    Disks cannot access all data equally fast, but are bigger!

Code

```c
int i = 0;
int *ptr = &i;
int y = i + 2;
```



Locality

- At any given moment, a program is likely to be accessing ..
- Code:
  - Mostly accessed sequentially
  - Loops and 'if' (branches) jump around only a little usually.
- Data: Access small parts of data at once
  - Variables are often accessed repeatedly (a loop), or same data structure accessed over and over.
- ..
  - recently used data is likely to be reused (i.e., loops)
- ..
  - the next data you need is likely nearby previous data you used. (e.g., an array / struct)

Understanding Memory Solutions

- Fundamental Properties of Memory Use
  - Programs really work on memory.
  - Programs access the same data over and over again (temporal locality)
  - Programs access data nearby to previously accessed data (spacial locality)
- Can we use these to design how to share memory?

ABCD: Locality

- Assume a program has just accessed memory locations 6 and 12.
  - Spacial locality suggests we might soon access?
    - (a) 0, 3, 9
    - (b) 6, 12
    - (c) 5, 7, 11, 13
    - (d) 4, 8, 10, 14
  - Temporal locality suggests we might soon access?
    - (a) 0, 3, 9
    - (b) 6, 12
    - (c) 5, 7, 11, 13
    - (d) 4, 8, 10, 14

14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

Solution:

Virtual Memory

Memory Abstraction

"All problems in computer science can be solved by another level of indirection, except for the problem of too many levels of indirection."
-- David Wheeler

- Virtual memory is a mechanism to enable:

(i)..

(ii)..

- Simply put,..

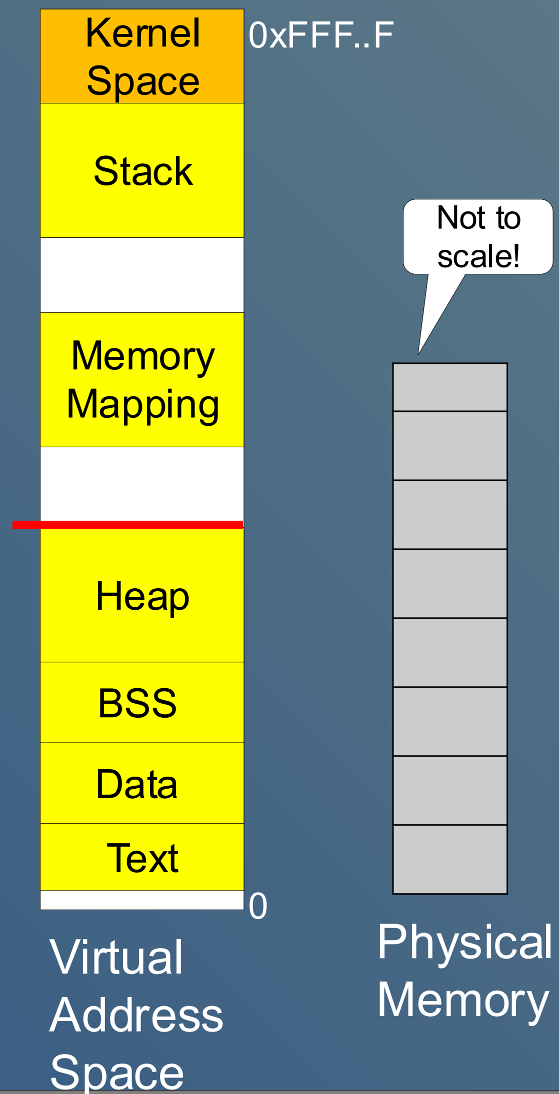
- Virtual memory consists of:

– virtual address space and address translation.

– Virtual memory is a good example that demonstrates the power of abstractions.

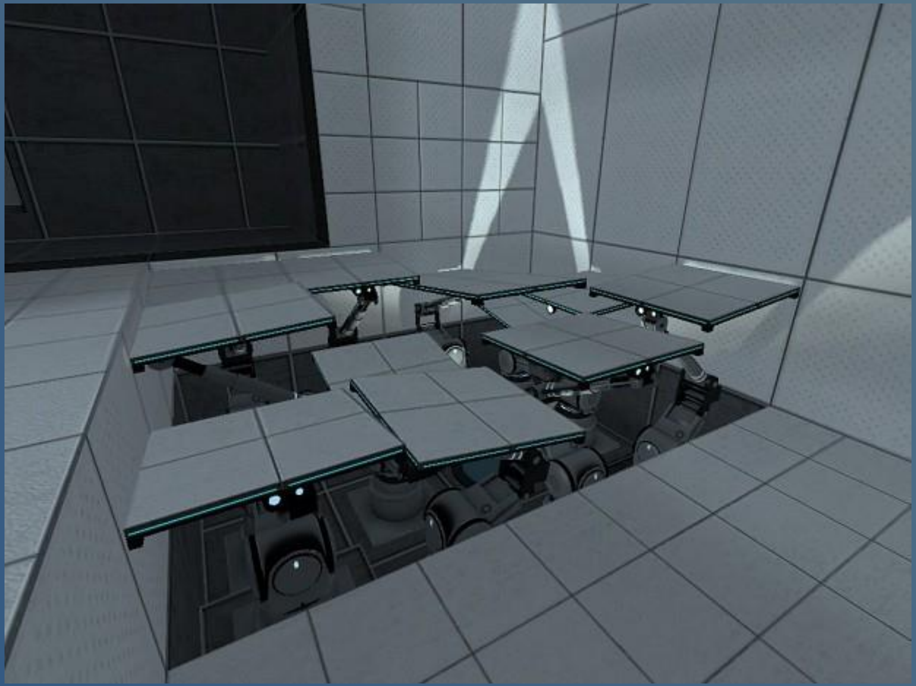
Virtual Address Space

- All memory discussed so far ..
  - Virtual memory size is ..
  - Virtual memory is a memory abstraction (imaginary space) that the program & programmer operates in.
  - The OS + hardware build us this imaginary space.
- Virtual vs Physical
  - User-level processes works with virtual addresses.
  - Kernel-level components can deal with both virtual and physical addresses.



Room Analogy

- Imagine a process as a room
  - It's virtual memory space is the surface of the walls.
  - There are no ***real*** walls; they are an illusion:
  - ..



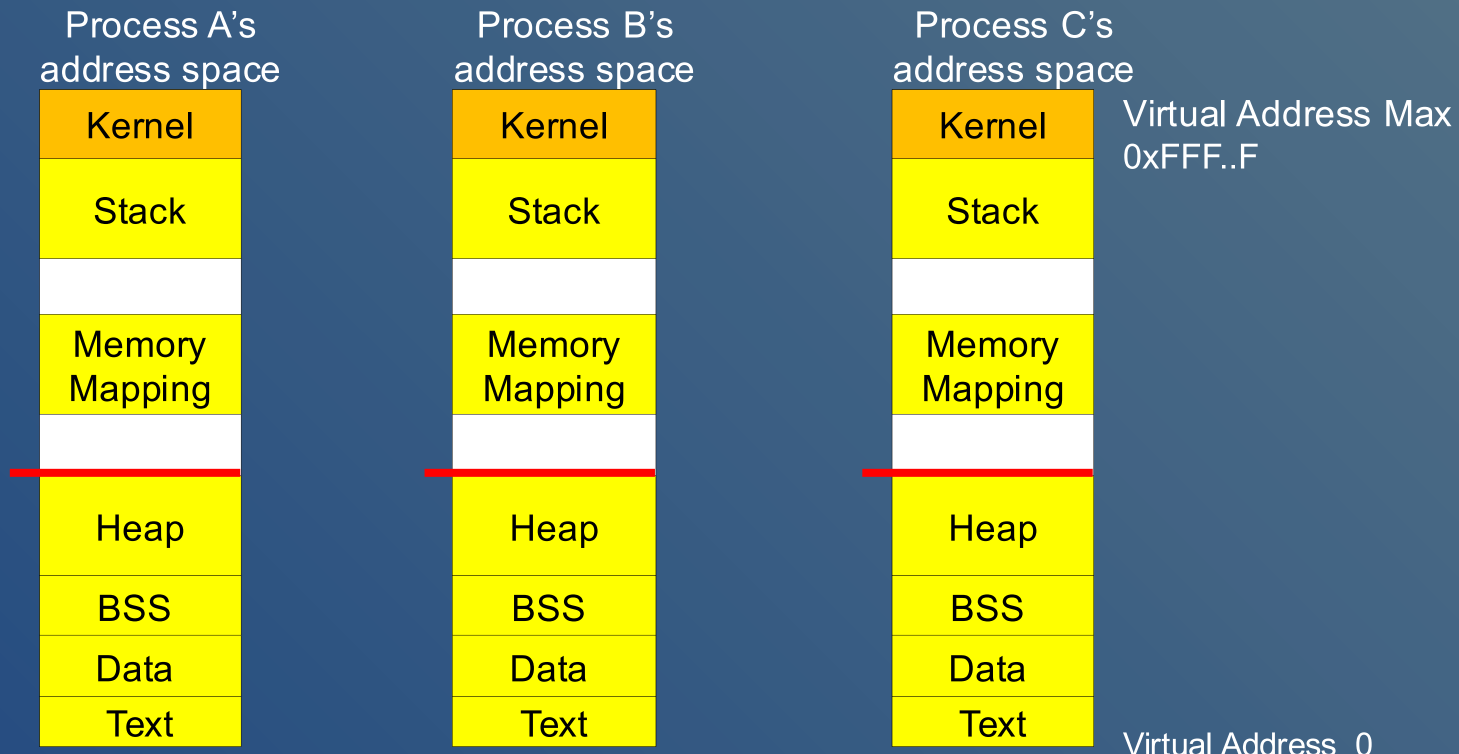
Room Analogy (cont)

- Imagine a process as a room
  - Virtual memory space is the walls:
  Pointers can point to the wall, can read/write on wall.
  - Walls have 2^{64} locations; much bigger than physical memory
- OS + hardware only put "physical" memory panels behind a few areas of the wall.
  - Operations on areas with physical panels work;
  - Operations outside of those areas fail: page-faults
  - E.g., reading from address 0x100 is virtual memory address.
    - Program doesn't know (or care) which physical "panel" of memory we are writing to.

- A physical "panel" is called either a page frame or segment

Process Virtual Memory

- Each process..
  - 0 to 2^{64}-1 (or 2^{32}-1).
- Each address in a virtual address space is a virtual address (physical address points to a physical memory location)



Benefits of Virtual Memory

- A process only sees its own address space,
  - i.e., ..
- Temporal & spacial locality mean

  ..
  - Don't have 16EB per process of physical memory!
  - OS can swap to a file on disk areas that have not recently been used
    - Makes physical memory available for other processes.

This file is called the..

Room Analogy

- Out of Memory
  - We can run out of physical memory panels for our room
  - So take an "older" panel, save it to disk, and then re-use it in a new place.
- If Needed Again
  - If needed later, we take another physical memory panel and reload the swapped out data from disk
  - Map virtual memory to the correct physical memory location.
  - Program never knows the difference!
- Works across multiple processes
  - OS manages mapping virtual address to physical memory panels
  - Panels are shared across all processes

Address Translation

Address Translation

- Process knows virtual addresses; hardware needs physical address
  - Must translate between them!
- Virtual Memory is..
  - Each page is

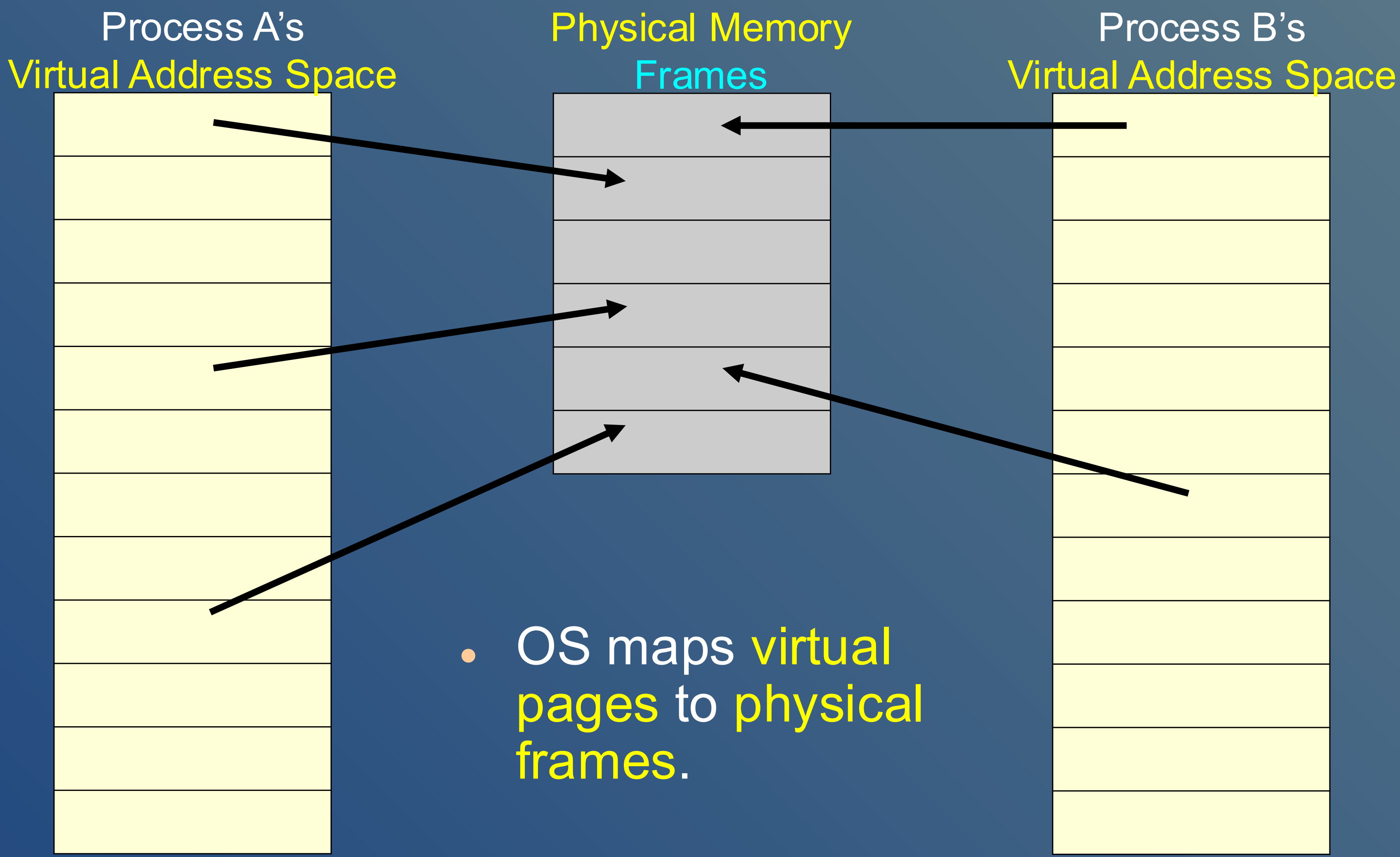
  ..
  - Kernel controls the mapping
  - Kernel configures hardware to translate virtual addresses into physical addresses

Address Translation

- Consider a memory operation like:
```
int *ptr;
*ptr = 10;
```
- Steps in translation:

1. Figure out which virtual memory page *ptr is on.
2. Figure out which physical frame it maps to
3. Redirects the access to the correct physical memory frame and address within it.

Address Translation



- OS maps virtual pages to physical frames.

Address Translation

- Approaches to Mapping "Panels' to Memory
  - How do we divide our virtual address space into smaller regions ("panels" in our analogy)?

Paging

Pages

- ..
  - 4 KB is a popular size but modern OSs have bigger pages (e.g., 4 MB) as well.
- Example
  - If we have 16KB virtual address space and page size 4K
  - How many pages? ..
  - Here are 2 process, each with its own virtual address space. Page numbers are in binary:

Process A's Address Space	Process B's Address Space
page 11	page 11
page 10	page 10
page 01	page 01
page 00	page 00

Page Frames

- Physical memory divided into
  ..
  - Each is the same size as pages.
- Example:
  if we have 8KB of memory with 4KB page size = 2 frames
  (#'s in binary)

Physical Memory
Frames

page frame 01
page frame 00

Address Translation

- A virtual address is divided into two parts:
  - ..
- Example:
  4 pages, each of 16 bytes.
  - 4 pages need..
  - 16 bytes need..
  - 6-bit virtual address space divided into 2-bit page numbers and 4-bit offsets
    - Address 100101 is divided into page number 10 and offset 0101.
    - Address 000010 is divided into page number 00 and offset 0010.

ABCD: Address Translation

- Consider a computer where
  - each page is 32 bytes
  - have 8 pages

What does the memory address 10011010b mean?

(a) Page 10011b, Offset 010b
(b) Page 100b, Offset 11010b
(c) Page 010b, Offset 10011b
(d) Page 11010b, Offset 100b

Page Table

- When a process accesses a (virtual) address, ..

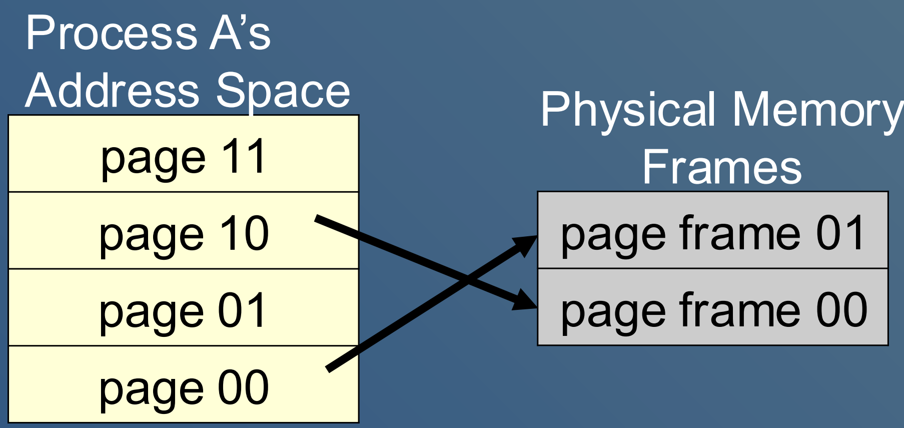
–The offset does not change.

- Kernel maintains a page table per process.

–Maps page number (virtual) to
a page frame number (physical).

Page Table

Page Number	Page Frame Number
00	01
10	00

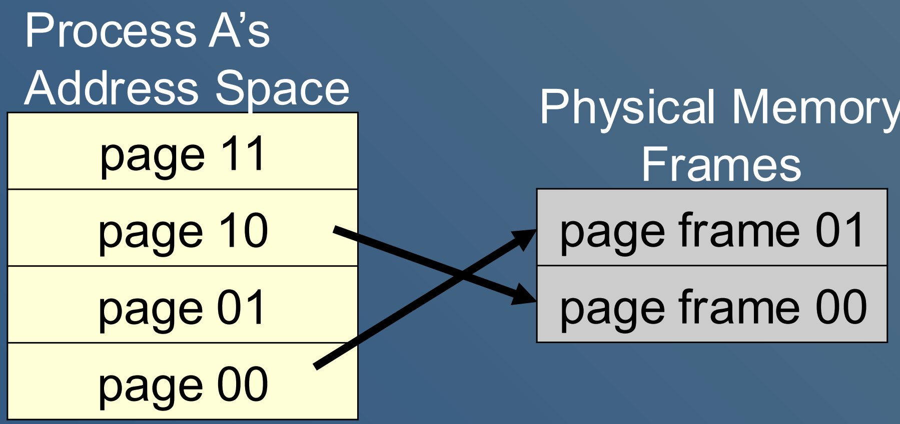


Address Translation Example

- Example:
  Convert virtual address 101011b to physical address
  - Assume 16 byte pages.
    So, offset is..
  - Address is 6 bits therefore
    ..
  - Page: 10b ..
    Offset: 1011b (unchanged)
  - So physical memory 001011b

Page Table

Page Number	Page Frame Number
00	01
10	00



Number of Pages

- There are vastly more (virtual) pages than (physical) page frames.
  - ..
  - OS only maps a page to a frame when needed (more later).
- Hardware supports converting pointers from virtual to physical addresses
  - OS configures the page table
  - HW looks mappings at runtime

Page Table Size

- Page Table Size
  - If page numbers use **n** bits,
    the maximum possible number of pages is $\mathbf{2^n}$.
  - If offsets use **m** bits,
    the maximum possible page size is $\mathbf{2^m}$.
- For example,
  - Page size 4 KB on a 32-bit architecture.
  - m =..
  - n =..
  - Therefore can have 2^{20} pages.
    This is 1M pages!

ABCD: Address Translation

- Given the page table below, what is the physical address for (virtual) address 0010 1011 1101 1100b?

(a) <000001, 1111011100>
(b) <001010, 1111011100>
(c) <111010, 1111011100>
(d) <000101, 1111011100>

Page Table

Page Number	Page Frame Number
000001	001010
111010	000011
101001	000111
001010	000101

Segmentation

Segmentation

- Segmentation is another solution
  ..
- Segmentation is similar to paging:
  - Memory is divided it sections
  - Each section is located in physical memory
  - Virtual memory addresses are translated to physical memory addresses.
- Segmentation is different because:
  - ..
  - E.g., text segment, data segment, stack segment, heap segment, ...

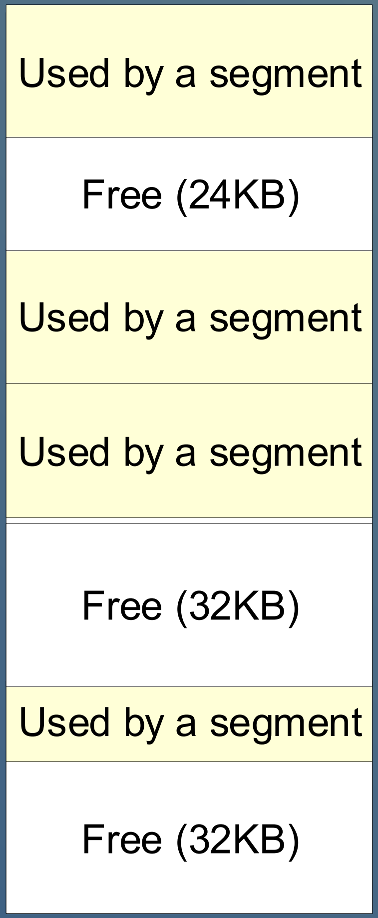
Segmentation Address translation

- Segmentation Address translation
  - Must still translate virtual memory address to physical memory address (beyond scope of this course)
- Modern OSs typically use paging rather than segmentation.

Segmentation and External Fragmentation

- External Fragmentation (recall)
  When free space is broken up into many different places.
  - Over time, with segmentation, free space gets broken up into different places.
  - ..
  - Since segments are of different sizes, no one free block might be big enough, even if we have enough total free memory.
- Example:
  - Unable to allocate 40KB segment

Physical Memory



Paging and External Fragmentation

- ..

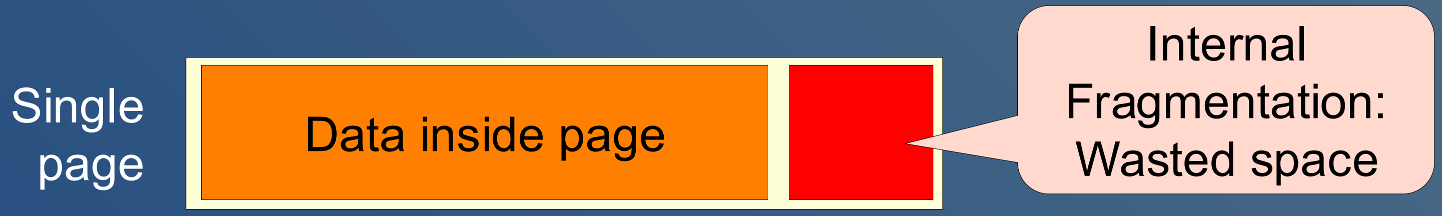
  – We only have one page size,
  so when you need a page..

- Internal Fragmentation

  – Since pages are handed out at a fixed size,
  there is very likely to be wasted space at the end of a page.

  – It happens with paging.

  – Combat it by keeping page size small.



Running out of Memory

Out of Memory

- Out of memory
  - Limited physical memory but virtual memory space is vast!
  - Can't bring in all virtual pages to physical memory.

  What do we do?

- Demand paging & swapping
  - Demand paging:

  ..

  - Swapping:

  if we don't have a free page frame, kick out a page already in memory to disk and bring in the new page.

  - Swap space:

  disk space dedicated to store swapped-out pages.

- How do we decide which memory page to swap out? We need a page replacement algorithm

Demand Paging

- Why does demand paging work?
  - Insight: ..
  - This is based on locality of access.
- Recall:
  - Temporal locality: if a program accesses a memory location, it is likely that it's going to access it again in the near future.
  - Spatial locality: if a program accesses a memory location, it is likely that it's going to access other memory locations nearby.
- ..
  - when a memory location is accessed, it brings in the region that the location belongs to, not just the specific memory location.
- Demand paging & swapping leverage temporal locality:
  - ..

Page Replacement Algorithms

- Page Fault
  - when a memory location is accessed but
  ..
  - we need to bring in the page into a memory frame.
- The Question
  - When the memory is full (i.e., all page frames are used) and we need to load a new page,
  ..

Optimal Page Replacement Algorithm

- Optimal page replacement algorithm

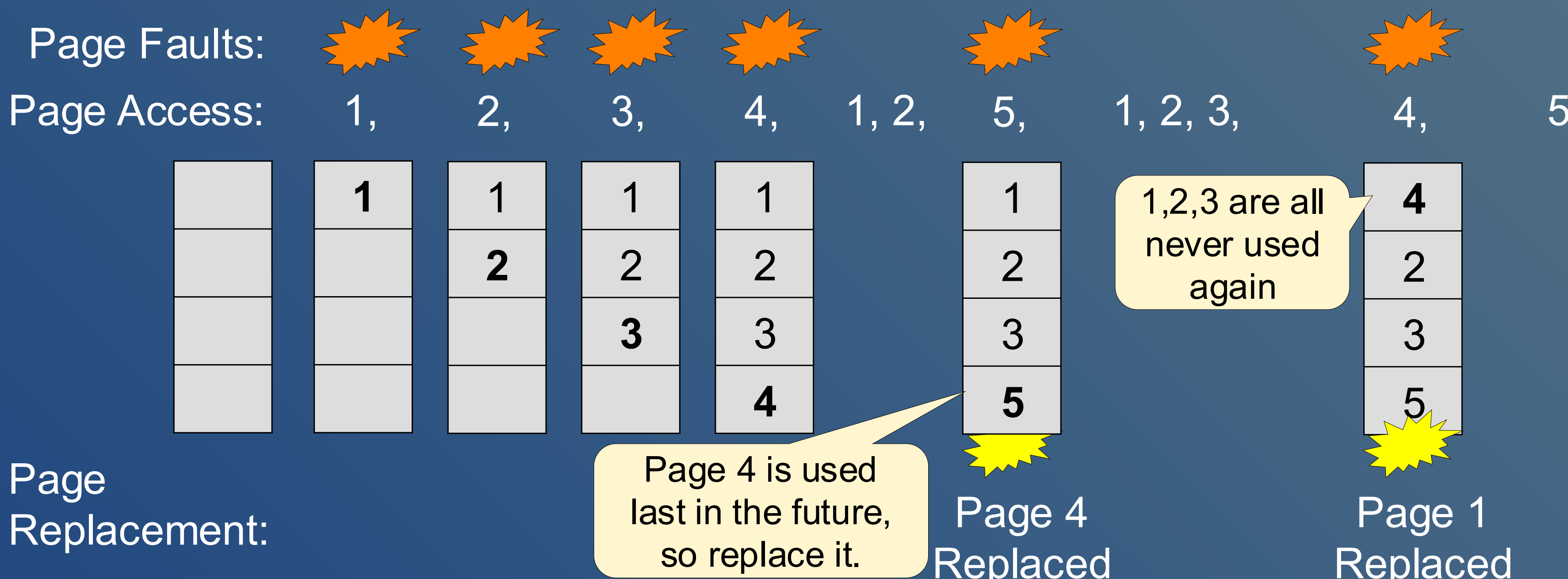
..

– This assumes that we know the future (which is impossible). Thus, this is only a theoretical exercise.

– Page replacement algorithms try to approximate this as much as possible.

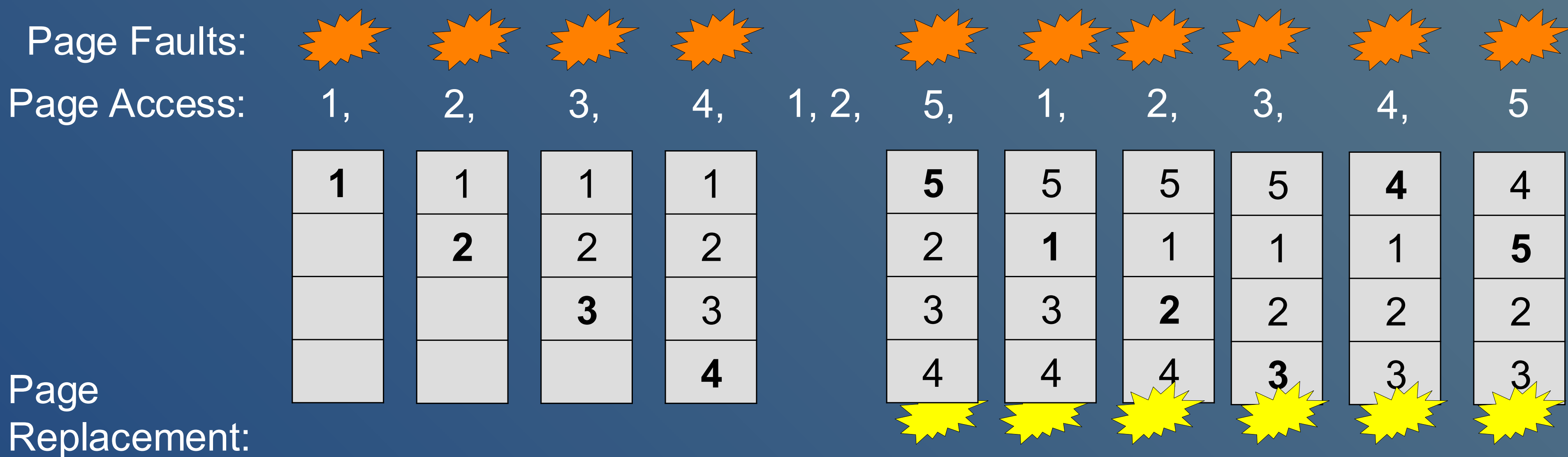
Optimal Page Replacement Example

- Example
  - Memory has 4 page frames.
  - Memory page access order (by page number): 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



FIFO (First In, First Out)

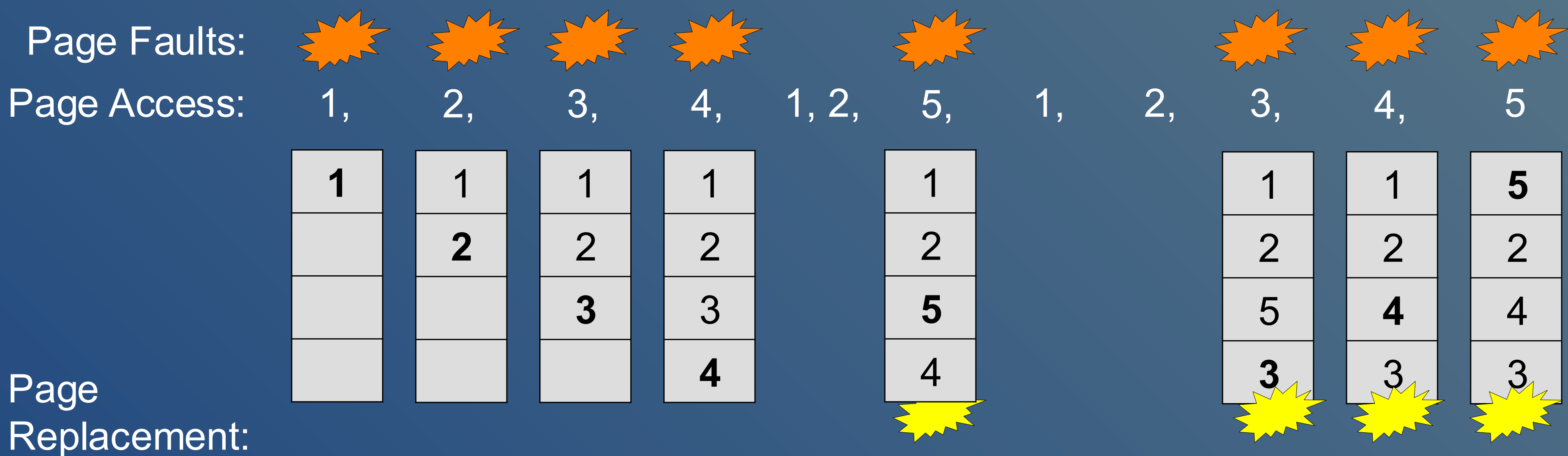
- FIFO
  - Keeps track of when a page was brought in to memory.
  - ..



- 10 page faults!
- This is simple but does not consider useful properties like locality.

LRU (Least Recently Used)

- ..
  - It tries to approximate the optimal algorithm.
  - It tries to infer the future based on past.



- 8 page faults
- Tracking access time is not simple to implement. Approximate it?

ABCD: LRU Paging

- Consider the following computer:
  - 4 page frames
  - Uses LRU page replacement algorithm

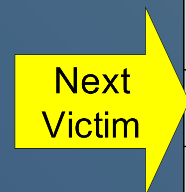
  How many page faults are there for the following sequence of page accesses?

```
–1,   2,   3,   4,   5,   2,   4,   5,   1,   5
*    *    *    *    *(1)                *(3)
```

(a) 2 page faults
(b) 5 page faults
(c) 6 page faults
(d) 10 page faults

Second-Chance

- Second Chance is an approximation of LRU
  - Each page has a reference bit (ref_bit), initially = 0
  - ..
  - We maintain a moving pointer to the next (candidate) “victim”.
- When choosing a page to replace,
  check ref_bit of victim:
  - ..
  - Else
    - Clear ref_bit to 0.
    - Leave page in memory
    - ..
    - Move pointer to next page (wrapping around)
    - Repeat till a victim is found.



Ref. Bit	Pages
0	...
1	...
1	...
0	...

Second Chance Example

- Example
  - Assume we have triggered a page fault.
  - No empty pages, so must replace.
  - Let's find the victim page to replace.

Initial State

Next Victim

Ref. Bit	Pages
0	...
1	...
1	...
0	...

Clear flag; Move on

Ref. Bit	Pages
0	...
0	...
1	...
0	...

Clear flag; Move on

Ref. Bit	Pages
0	...
0	...
0	...
0	...

Found Victim Page!

Ref. Bit	Pages
0	...
0	...
0	...
1	changed

ABCD: Second Chance

- Using second chance page replacement algorithm, which page will be the next victim?

Ref. Bit	Pages
1	Page 110
1	Page 111
0	Page 101
1	Page 001

Next Victim (→ Page 001)

(a) Page 110b
(b) Page 111b
(c) Page 101b
(d) Page 001b

Thrashing

Thrashing

- If a process access a large amount of memory,
  OS could keep needing to bring new pages into memory
  - Example:
    An process that jumps through a huge amount of memory, reading one value every 4K (once per page).
- Thrashing:
  - a process is too busy swapping in and out pages and not really executing its program on the CPU.

Summary

- Virtual Memory
  - Process works only in the virtual memory space.
  - OS can flexibly share memory between processes.
  - Gives process memory isolation.
- Address Translation
  - Converting (virtual) address to physical address.
- Paging
  - Virtual memory broken up into identical size pages.
  - Physical memory broken up into page frames ("frames").
- Segmentation
  - Like paging, but different size regions (segments).
- Page replacement algorithms:
  - Optimal, FIFO, LRU, Second Chance