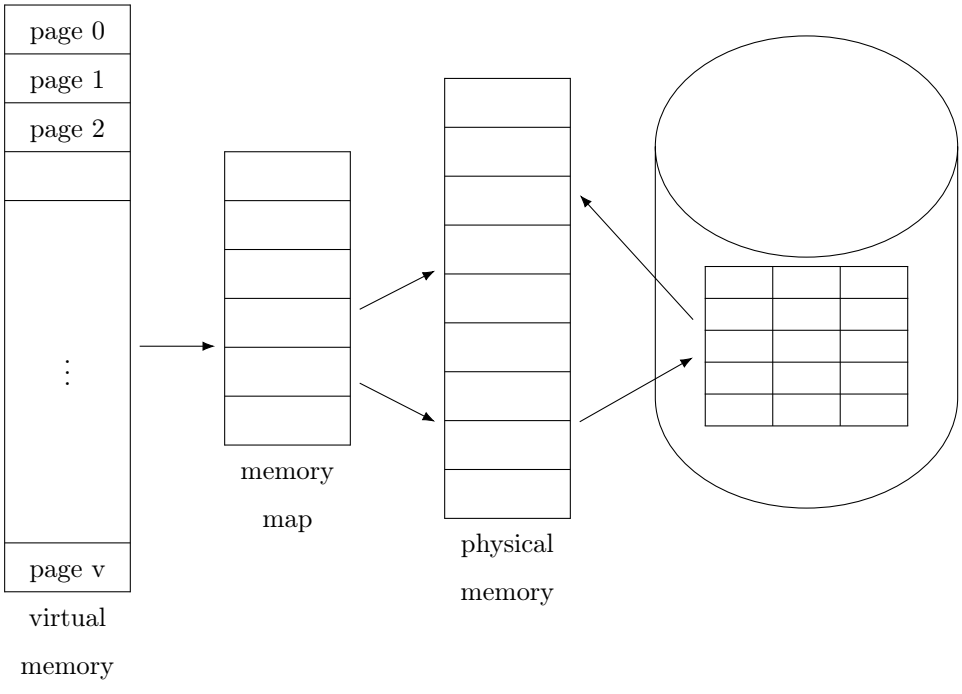
What methods of protecting memory from corruption by bad processes do we have?

1. Hire better programmers
2. Enable subscript/NULL checking
   - requires a language that stores array sizes
   - conditional branches and checking make this very slow
3. Base & Bound Registers
   - simple: only requires 2 registers
   - good for batch environments, bad for dynamic
   - requires contiguous memory per-process and thus can cause external & internal fragmentation
   - not conducive to sharing; requires repetition of code with copying for each process
4. Segmented Memory
   - a hardware table mapping indices to a (array, index)/(segment#, offset) pair
   - does not require adjustment of locations
   - still has external fragmentation

All of these have downsides—a better and the most common one is virtual memory. This is for a few reasons:

- frees programmers from worrying about physical locations and fragmentation
- prevents processes from illegal memory access
- lets processes and threads share memory
- lets VM be larger than actual memory
  while this is technically true, it can cause rapid switching of pages (*thrashing*)

How does it work?



- Each thread has a register %cr3 that points to the page table/memory map.
- The page table maps virtual addresses to physical addresses.

Note that Intel keeps the specific content format of the page table secret, but this abstraction is enough to work with.

Pages are supported by the hardware — x86-64 has a CTRL register that stores page sizes. How do we choose the size of virtual pages?

- large pages → small table (saves time & space)
- small pages → large table (less fragmentation)

Are loops allowed? (virtual → physical → virtual)

- This would allow us to look at the table, so to prevent memory cheating, Linux doesn't even let us look at the table.

How do we edit the page table then?

```
int mmap(void *vir_addr, size_t len, int prot, int flags, int fd, off_t offset);
// returns the virtual address of the new page (not necessarily the one you give)
// flags
//  fixed — do not give me an address other than what I asked for
//  private — other processes are not allowed to see our memory
```

What if the length is not a multiple of the page size?

- this is not supported; use int getpagesize()
- this limits pages to INT_MAX, so we use long sysconf() to use larger pages

Are all protections reasonable?

rw-: ordinary data

r--: read only data such as variables marked const

r-x: code (read is given because they are open source)

rwx: dynamically generated close

- this allows buffer overflow attacks
- the stack is often still rwx because of legacy

---: useful for catching addressing errors

- can be used for guard pages on either side of the array
- can be used to guard a recursive frame from modifying the base pointer in stack
- IS used on 0 to protect from NULL accesses

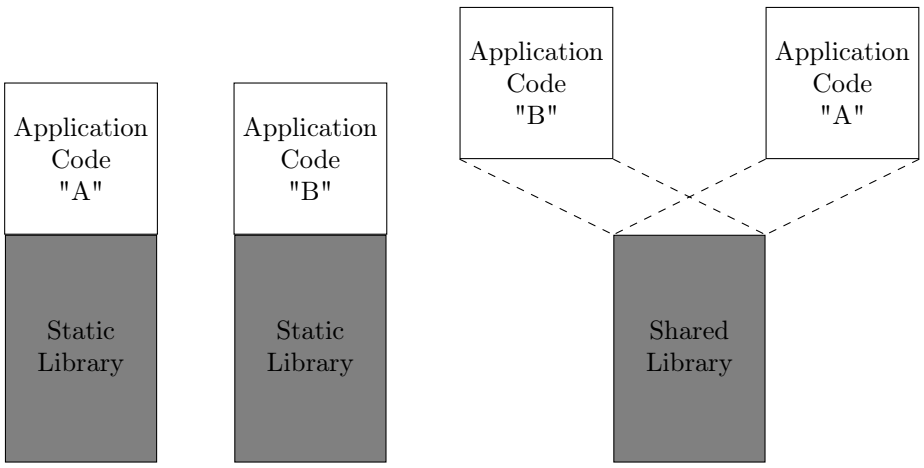
Note that 'x' is expensive, so low end chips with no security risk often combine 'x' with 'r'.

mmap trivia:

- The most common file to map is /dev/zero. This functions as an input only stream of zeroes
  - read — succeed and fill page with zeroes
  - write — succeed without doing anything

- Two processes cannot map to the same address at the same
- If a file has been mapped, it will not be deleted while still mapped
- A mapped page is not intrinsically tied to any file
- It allows us to use dynamic linking of libraries:

```
$ gcc main.c -o executable -lm;
# -lm links to math library
```



Processes share only an image of the code they share, so the virtual addresses do not match, so how do we map virtual addresses to physical addresses?

- We use the *Global Offset Table* := rwx self-modifying code that gives the offset for machine language system calls.