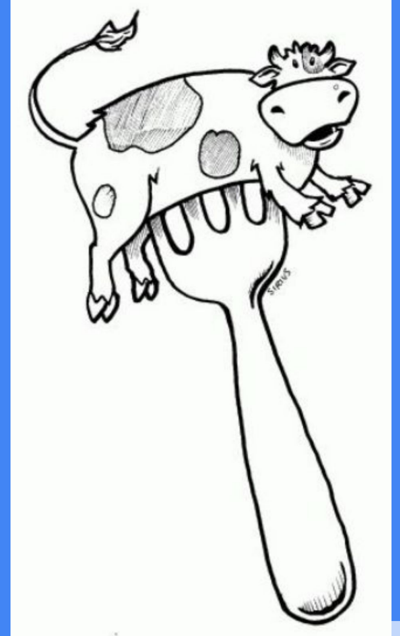
Lab COW

CSE451 Section 4



fork() right now

Currently, fork() walks through the entirety of user memory and copies every single page to the child process pagetable.

- Why is this bad?
  - (Think of typical usecases for fork())
- How can we improve this?

```
int
uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
{
  pte_t *pte;
  uint64 pa, i;
  uint flags;
  char *mem;

  for(i = 0; i < sz; i += PGSIZE){
    if((pte = walk(old, i, 0)) == 0)
      panic("uvmcopy: pte should exist");
    if((*pte & PTE_V) == 0)
      panic("uvmcopy: page not present");
    pa = PTE2PA(*pte);
    flags = PTE_FLAGS(*pte);
    if((mem = kalloc()) == 0)
      goto err;
    memmove(mem, (char*)pa, PGSIZE);
    if(mappages(new, i, PGSIZE, (uint64)mem, flags) != 0){
      kfree(mem);
      goto err;
    }
  }
  return 0;

 err:
  uvmunmap(new, 0, i, 1);
  return -1;
}
```

(Not actually fork, but what is called when you call fork())

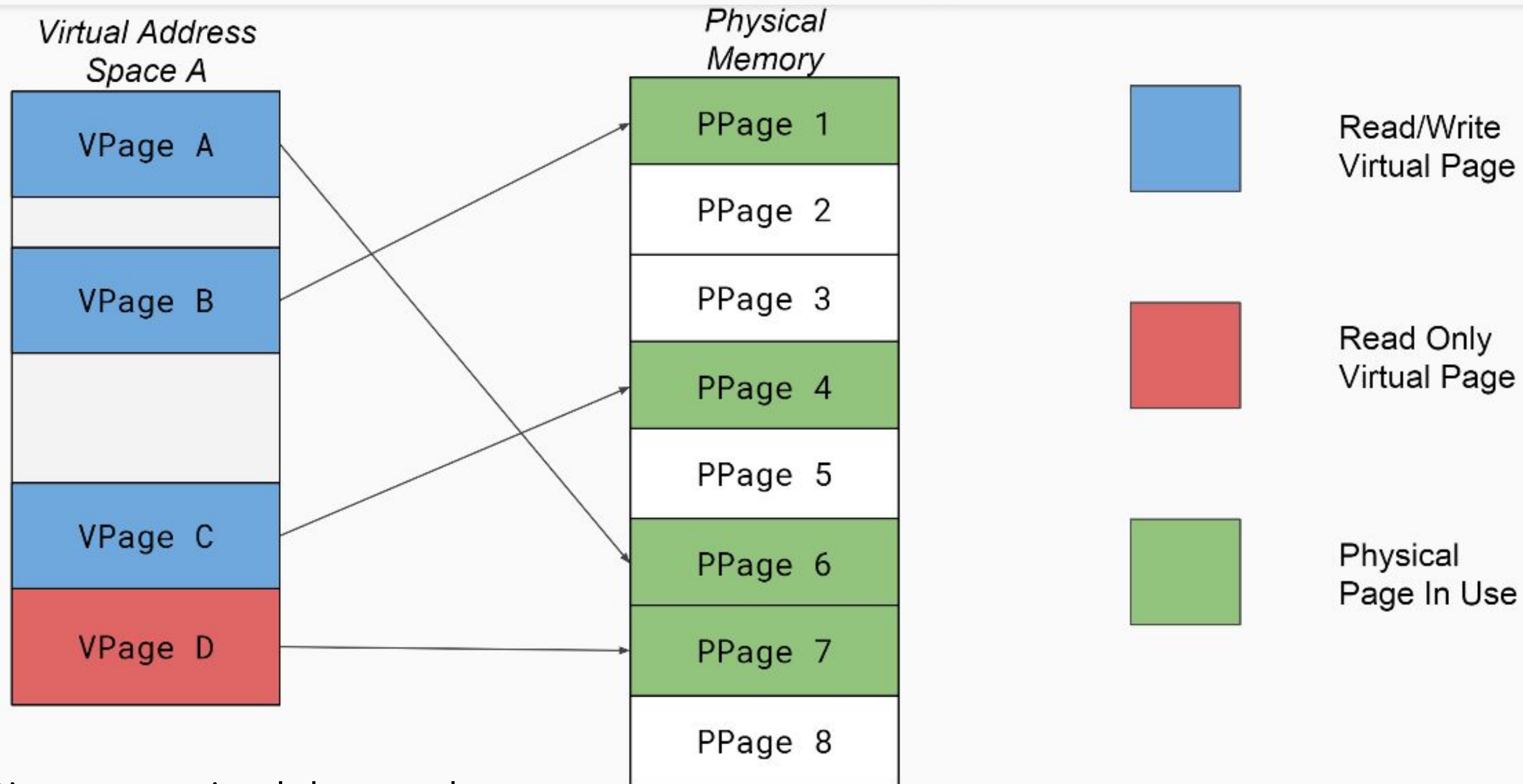
Copy-On-Write (COW) Fork

- Similar idea to lazy heap allocation

- In fork, don't copy immediately; map to same physical page & mark as readonly
    - How to distinguish between normal, readonly pages and COW pages?
    - Should *every* page be copy on write?

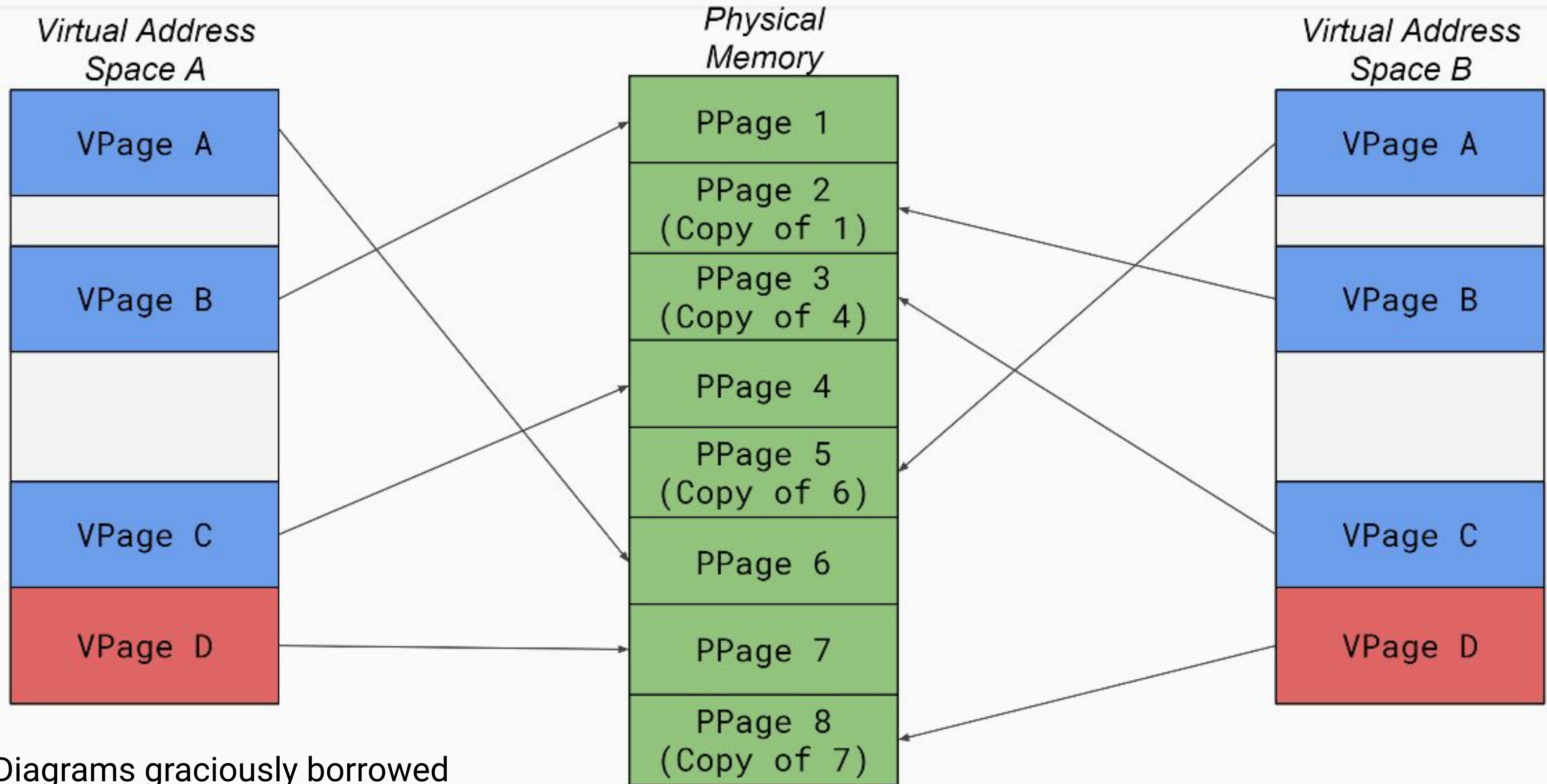
- When you page fault, copy the page then, and allow faulting process to write to new page

Before old fork:



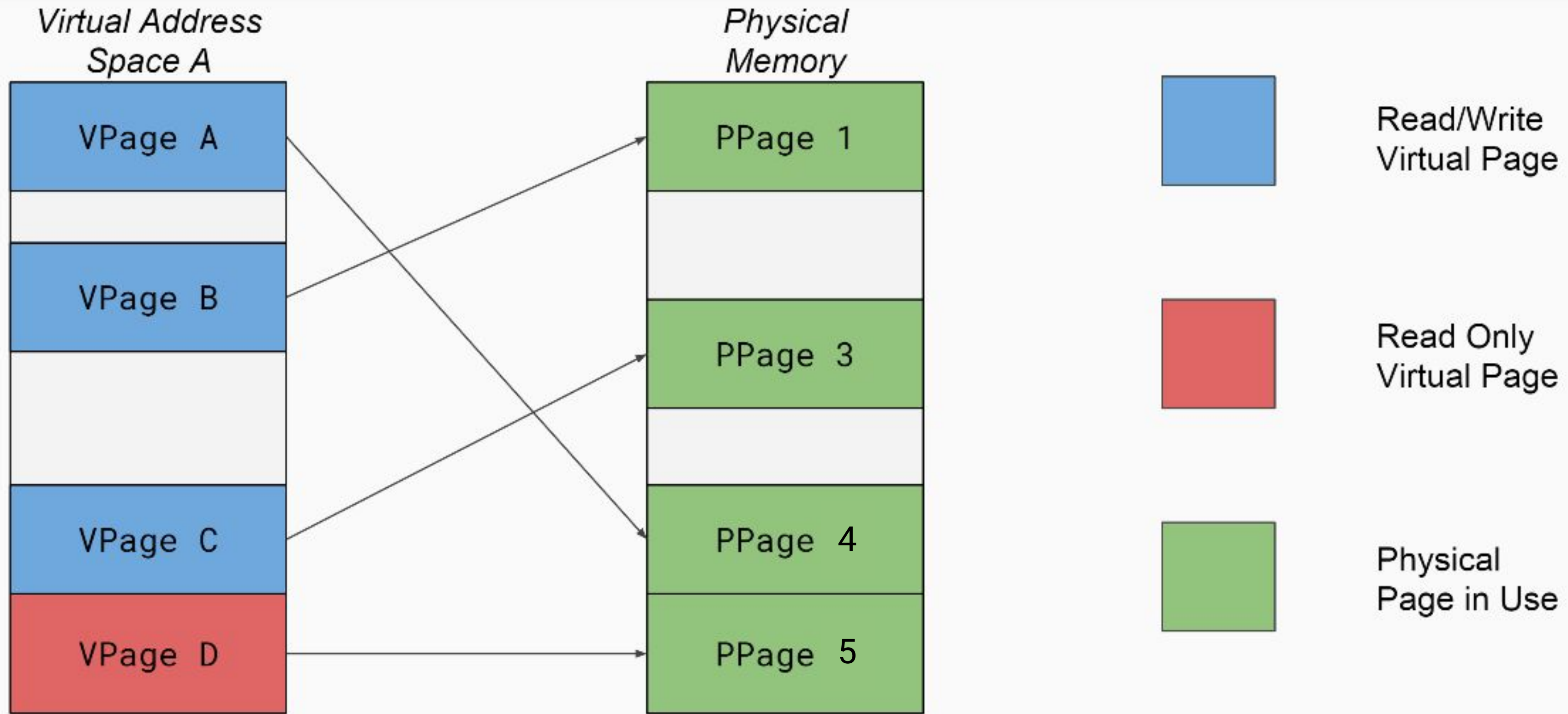
(Diagrams graciously borrowed from CSE451 18au)

After old fork:



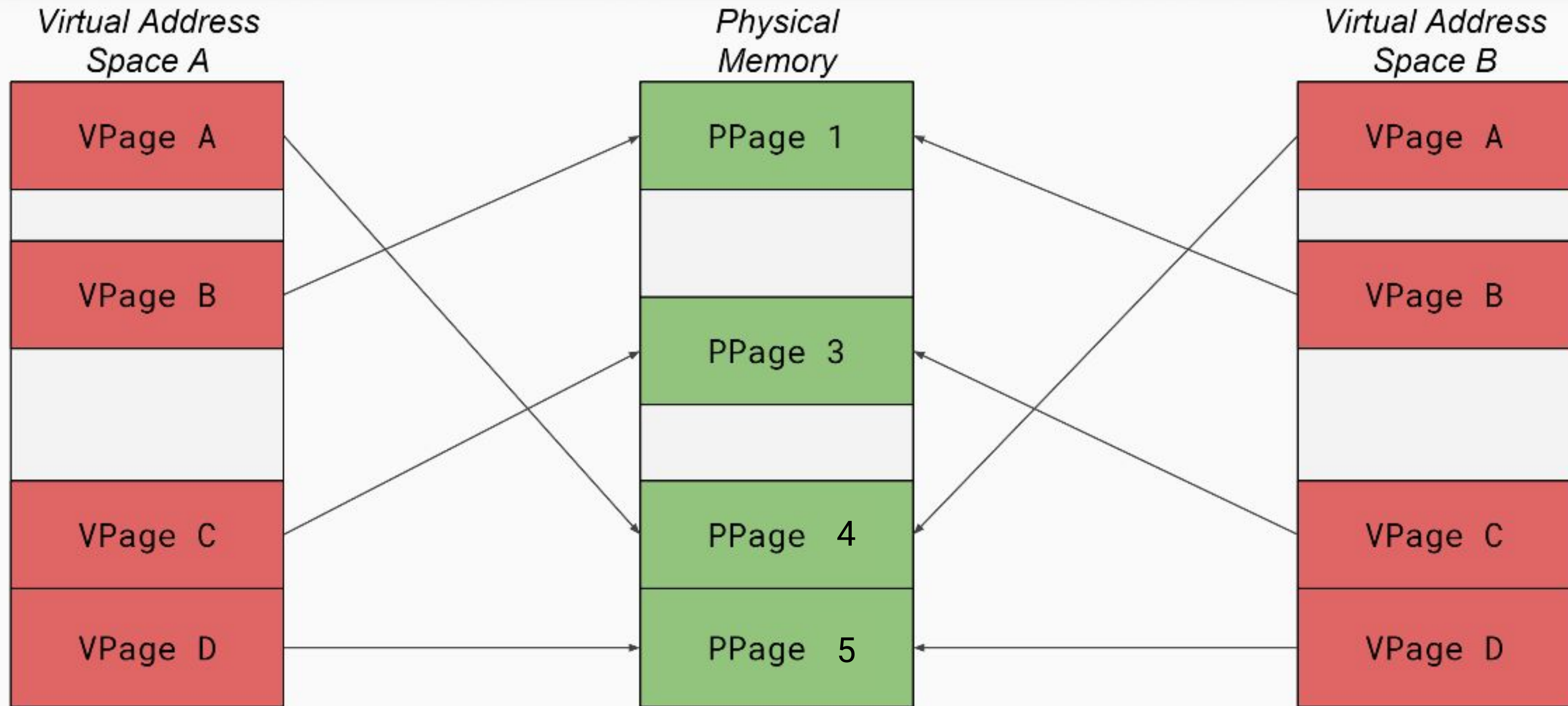
(Diagrams graciously borrowed from CSE451 18au)

Before COW-fork:



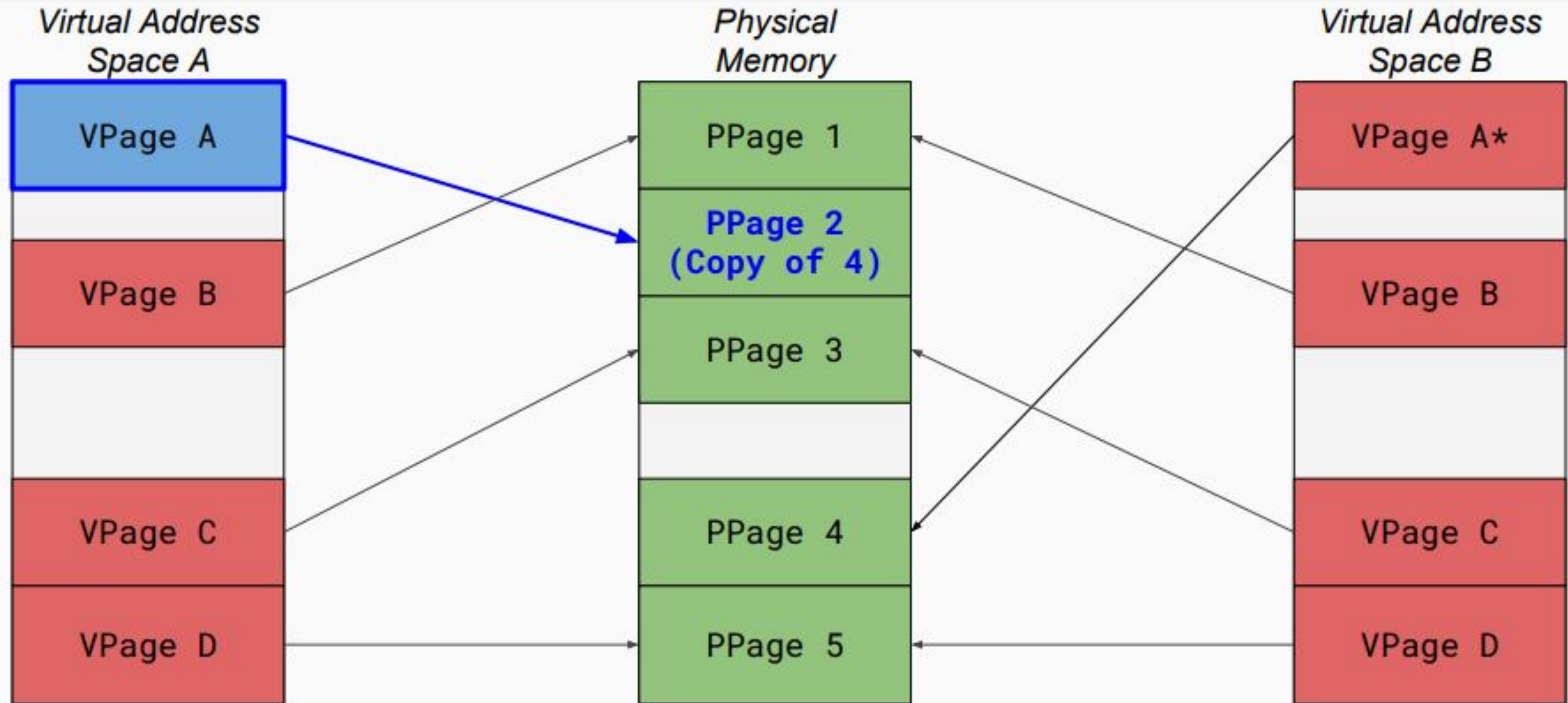
(Diagrams graciously borrowed from CSE451 18au)

After COW-fork:



(Diagrams graciously borrowed from CSE451 18au)

After Process A tries to write to VPage A/PPage 4 (with COW-fork):



(Diagrams graciously borrowed from CSE451 18au)

How do we keep track of all of this?

- Need some way of knowing whether or not a PTE is copy on write (why?)
  - Can use RSW bits to store this
  - Feel free to add your own macro for accessing this bit

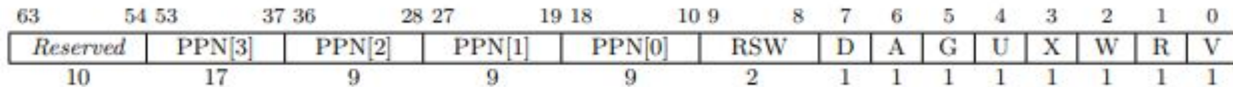


Figure 4.21: Sv48 page table entry.

How do we keep track of all of this?

- Need to know reference count of given physical page
  - So if a process calls kfree, page isn't freed if other processes are using it
  - Also know when to convert page from COW to non-COW
- Lab writeup suggests the following:
  - It's OK to to keep these counts in a fixed-size array of integers. You'll have to work out a scheme for how to index the array and how to choose its size. For example, you could index the array with the page's physical address divided by 4096, and give the array a number of elements equal to highest physical address of any page placed on the free list by `kinit()` in `kalloc.c`

Things to look out for

- Concurrency issues
  - If you choose to have an array of reference counts, make sure you edit it atomically [there is already a lock in kalloc.c]
- copyout()
  - Like in lab lazy, the kernel will need to handle COW allocation here, as it will attempt to write to user memory
- Be careful about what pages you make COW
  - Some pages are *meant* to be readonly

Good luck!

Questions?