Bonus Section: Lab 5 Swap

CSE 451 21wi

*most of this content taken from 19sp when swap was lab 4

Admin

- Lab 5 Due Thursday, 3/18/21 in Finals week
  - Send the course staff an email when you finish it so we know to grade it
  - Reminder, it's strictly extra credit, no penalty for not completing lab 5
- You don't need to turn in a design doc for lab 5
  - But we recommend you do one anyways for your own sake :)

Memory vs Disk

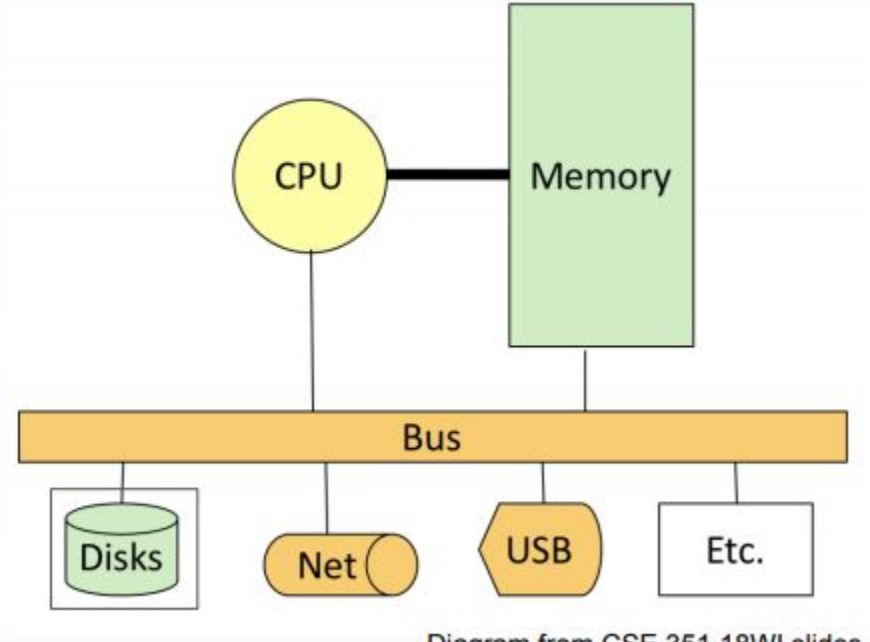
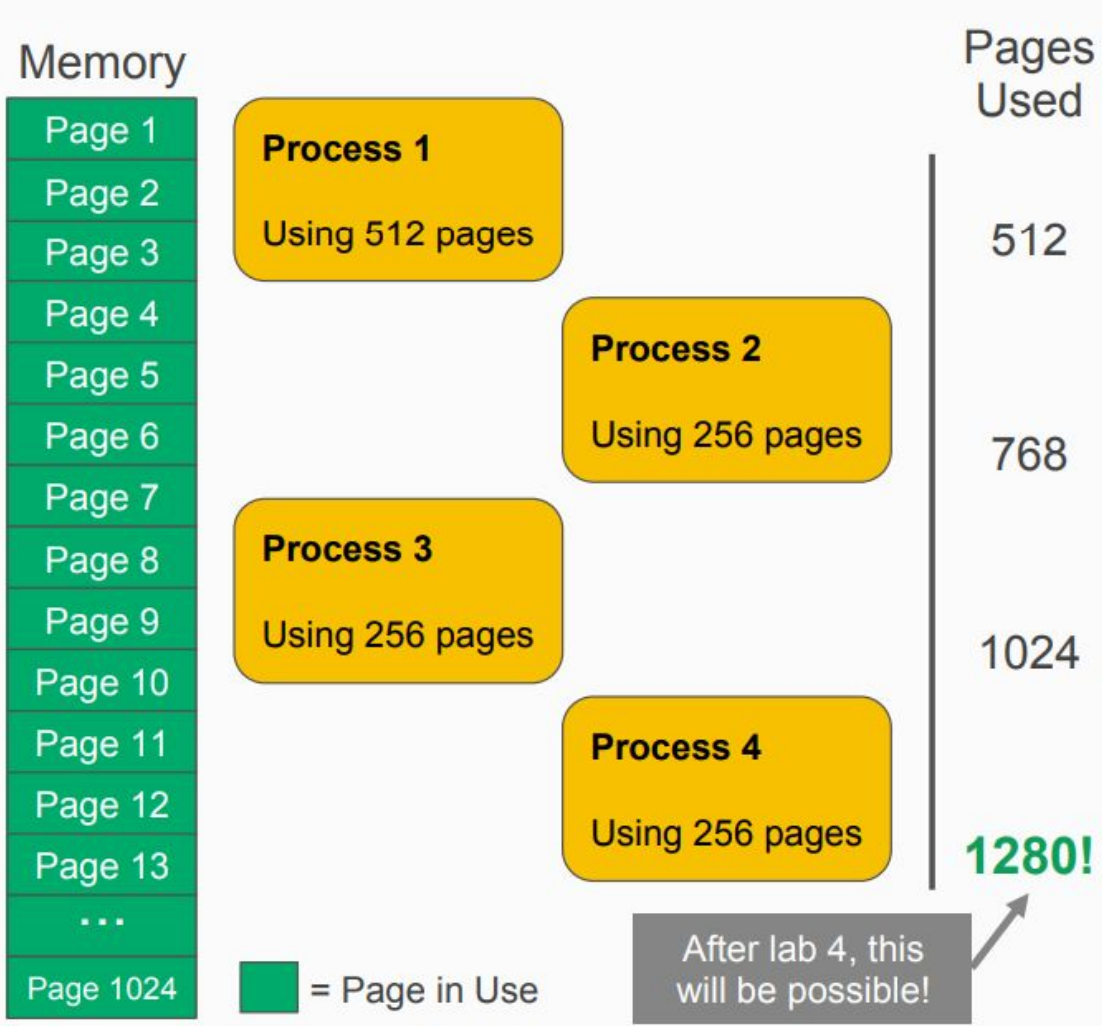


Diagram from CSE 351 18WI slides

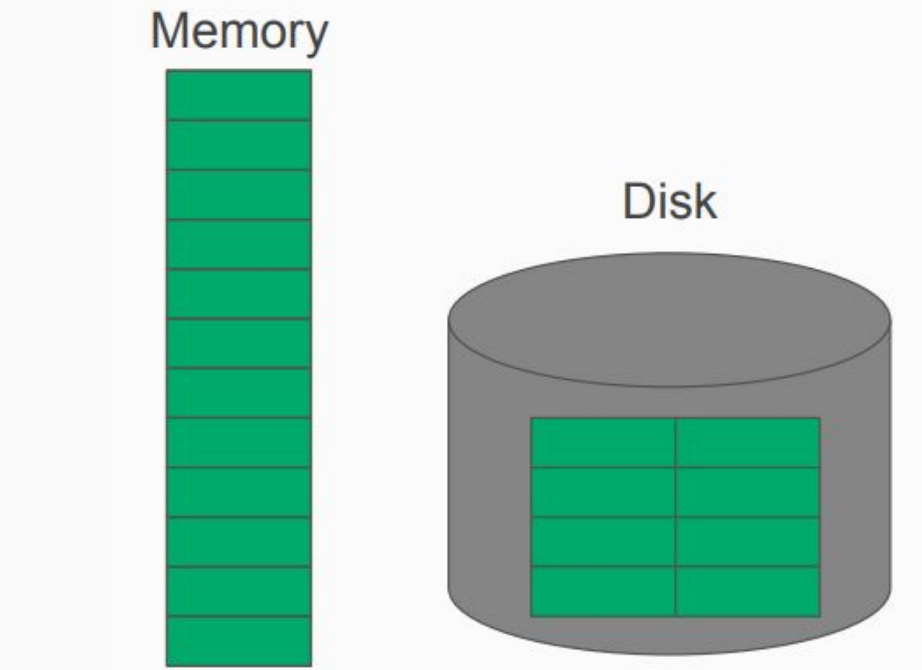
- Memory is in close proximity to CPU
  - Fast!
  - Volatile (loss of power == loss of data in memory)
  - More expensive (in actual cost, not latency)
- Disk is farther away from CPU
  - Much slower than main memory
  - Non-volatile
  - Less expensive

Virtual Memory

- Illusion that each process has all of memory to itself
- Would be nice if this illusion held even when processes together use more space than available memory

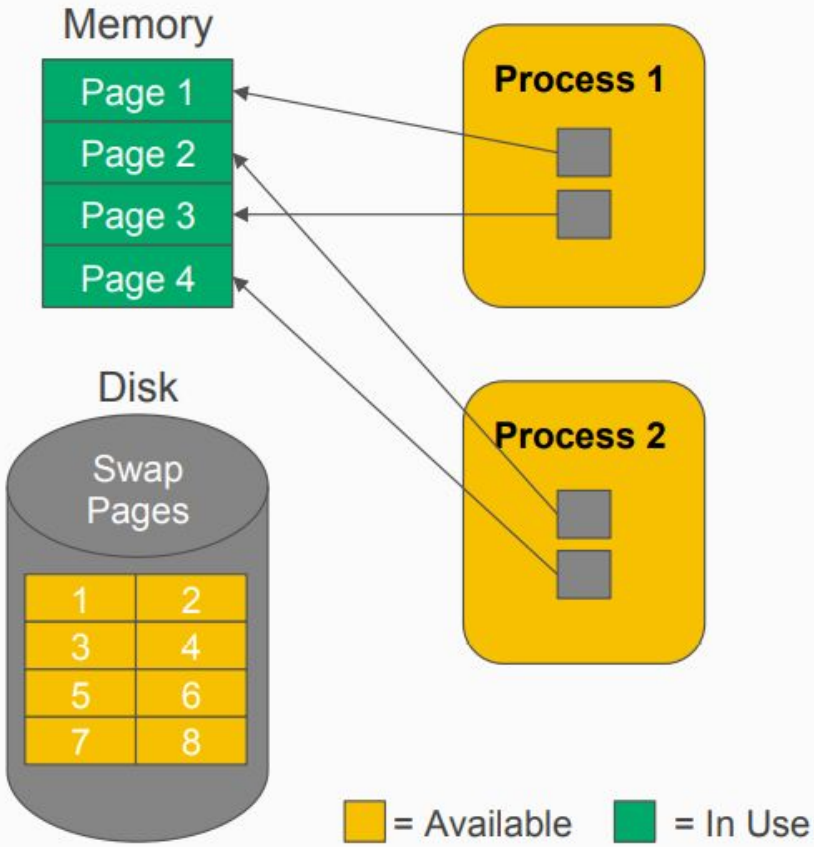


Creating the illusion of more memory



- Since we need to make it seem like there is more than 4MB of memory, we will need somewhere else to store data
- Idea: use the disk to store extra data, and page it in to memory on demand (called “**paging**”)

Paging Example - Assumes OS has only 4 pages memory for simplicity



This mapping could be obtained as a result of the following requests:

Proc 1: Requests a page of memory

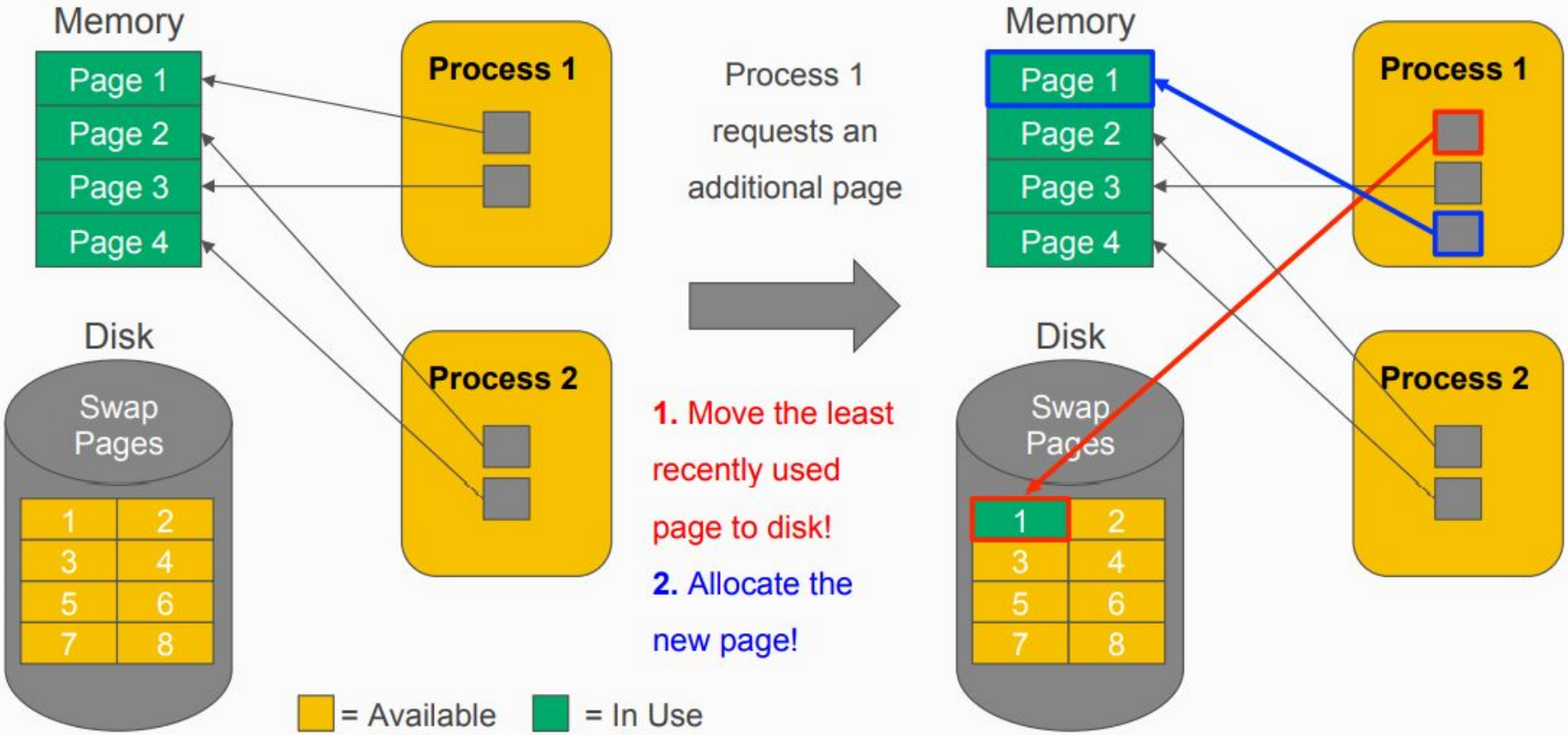
Proc 2: Requests a page of memory

Proc 1: Requests a page of memory

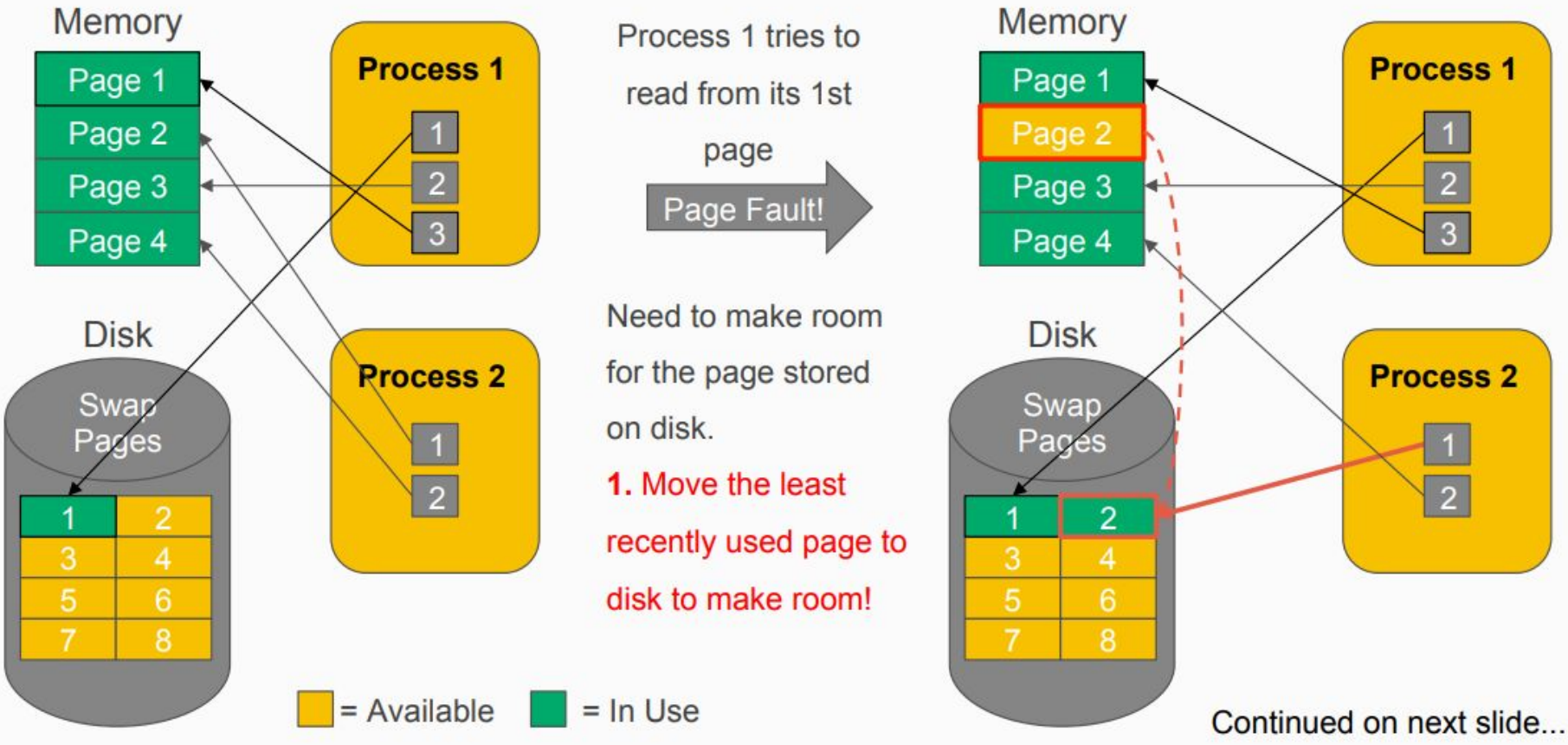
Proc 2: Requests a page of memory

Note: This example is highly simplified

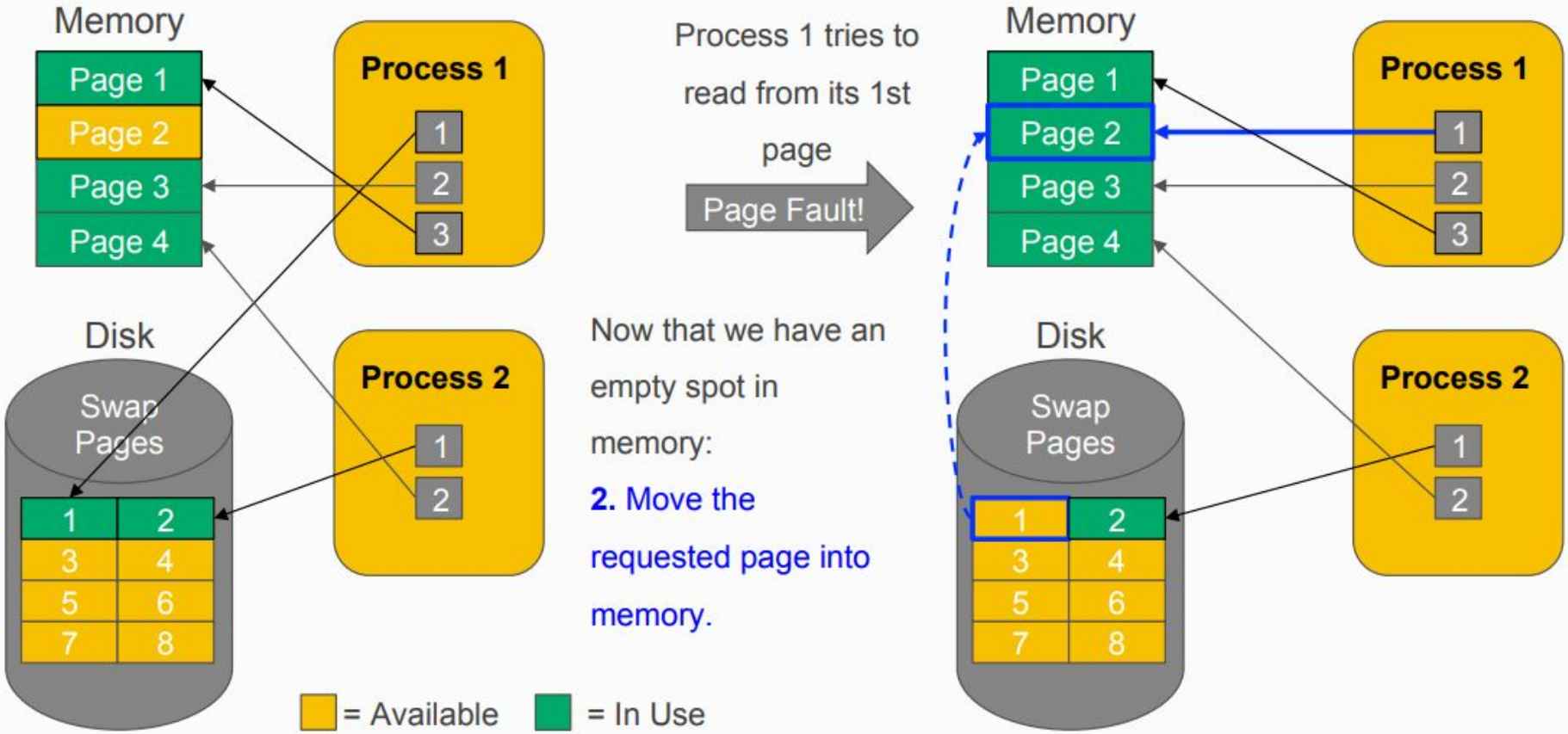
Paging Example - Swap page to disk



Paging Example - Page fault (Page not present), Part 1



Paging Example - Page fault (Page not present), Part 2



Eviction Policy

- Previous example evicted based on least recently used (LRU) policy
  - Faster, though requires a lot of bookkeeping on pages
  - If you choose to do this, props (but no extra points)
  - A ton of info on Linux's page reclamation if you're curious
- Evicting a random page is also fine for lab 5
  - `get_random_user_page()` function in `kernel/kalloc.c`

xk's Memory

xk's hardware is emulated by QEMU. In **kernel/Makefrag** we set up the options we will pass to QEMU

Before (Labs 1-4):

16MB (4096 pages)

```
QEMUOPTS += -m 16M
```

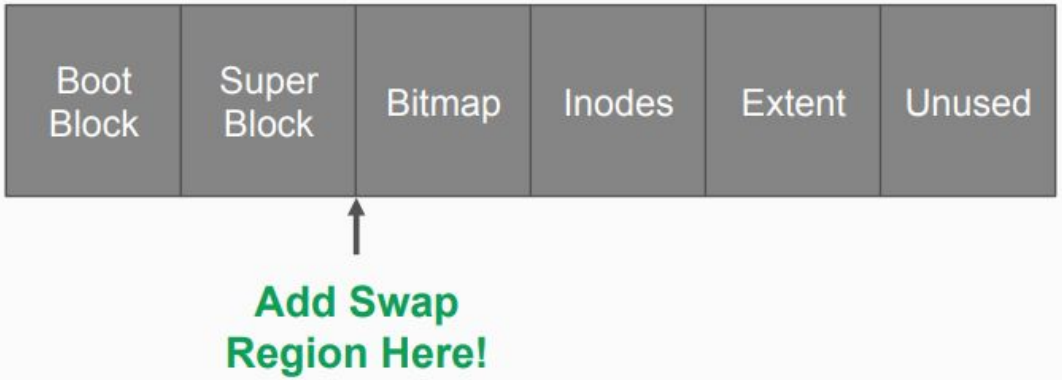
After (Lab 5):

4MB (1024 pages)

```
QEMUOPTS += -m 4M
```

xk's Disk

- Similar to the log region, you will need to add a swap region to use for pages swapped out to disk in mkfs.c



- 512 bytes in a disk block
- 4096 bytes in a page
- Therefore, need 8 disk blocks per swap page

nswapblocks to use given in lab5.md

Representing the Swap

- How should we keep track of a memory page that is in the swap region?
  - Hint: See how **kalloc.c** tracks physical pages for a design example (core_map)
- How do you track in a vspace whether a page is in physical or swap memory?
  - Hint: look at **vpage_info** struct and how that was used in Lab 3 COW fork
- What should happen when a swapped memory page is shared via copy-on-write fork?

Swap In

- When should we load pages from the swap region?
  - Hint: similar to lab 3's "when should we make a physical copy of a COW page?"
- When a page is swapped in, what needs to be updated?
  - Hint: who/what keeps track of whether a virtual page is in the swap?
  - What if the swapped in page is a COW page?

Swap Out

- When should we flush pages to the swap?
- Is there a set of memory pages you don't want to flush to swap?
  - Hint: What happens if the trap code page is in the swap?
  - In particular, **don't evict page 0**
- When a page is swapped out, what needs to be updated?
  - Hint: who/what keeps track of whether a virtual page is present in physical memory?
  - …and what if the page is a COW page?

Some more discussion questions

- What will happen when forking a process with some of its memory stored in the swap region?
  - What about on exit?
- You found a page to evict and know its virtual address, on what conditions should you update a vspace's entry?

Concurrency Notes

- Cannot hold a spin lock while reading/writing to/from disk
- *Cannot* **acquiresleep()** a sleeplock while holding a spinlock
  - Since it may call **sleep()**, which calls **sched()**
  - You *can* **acquire()** a spin lock while holding a sleeplock
- When swapping a page in be careful
  - It may call **vspaceinvalidate()**, which may in turn call **kalloc()**
  - **vspaceinvalidate()** may require up to 3 additional pages per process
  - You might get a **acquire()** panic if you're not careful!

- Lots of potential concurrency bugs in this lab, be careful!

Questions?

Good Luck!