Discussion Week 7

TA: Kyle Dewey

Overview

- Midterm debriefing
- Virtual memory
- Virtual Filesystems / Disk I/O
- Project #3

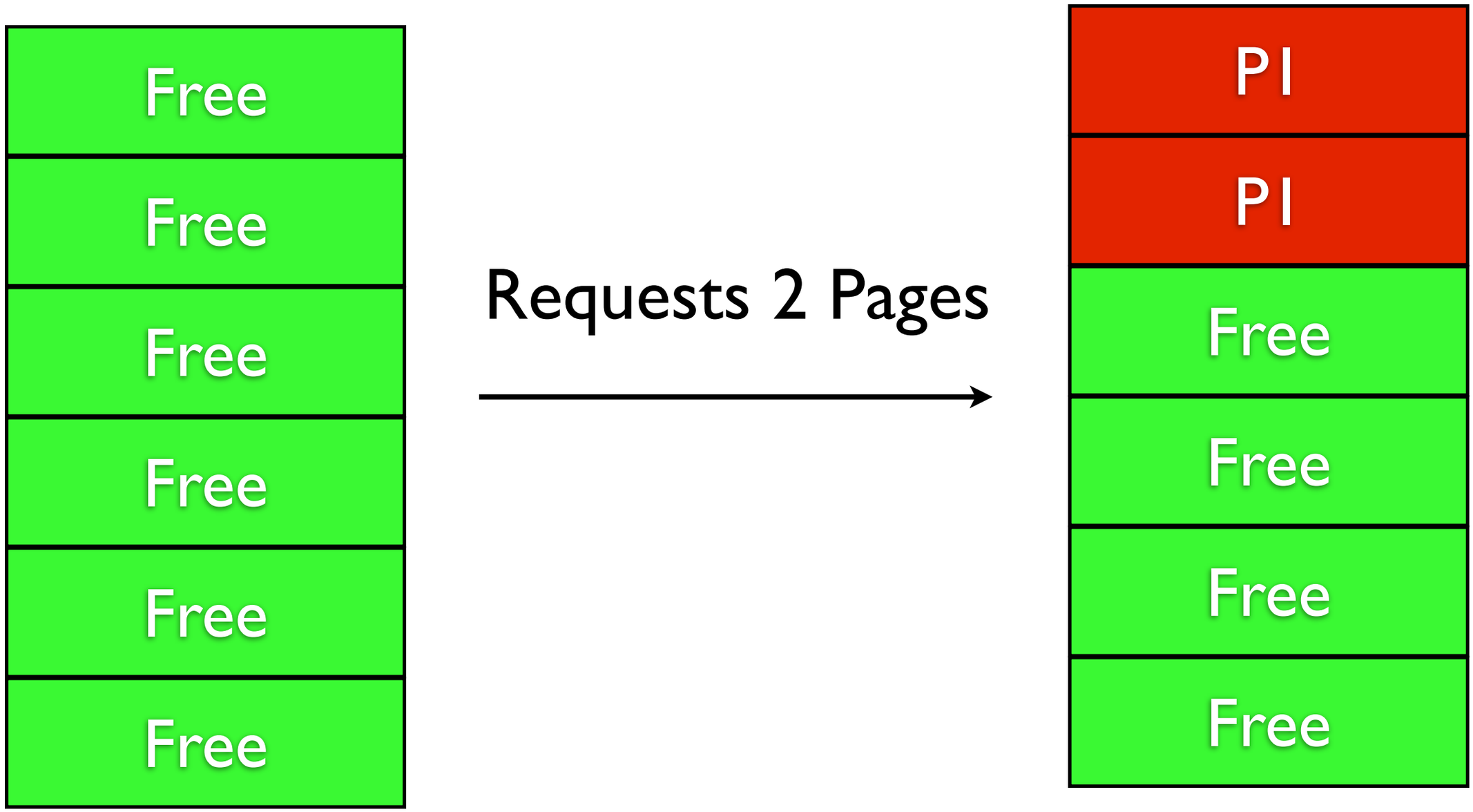
How was the midterm?

Recap

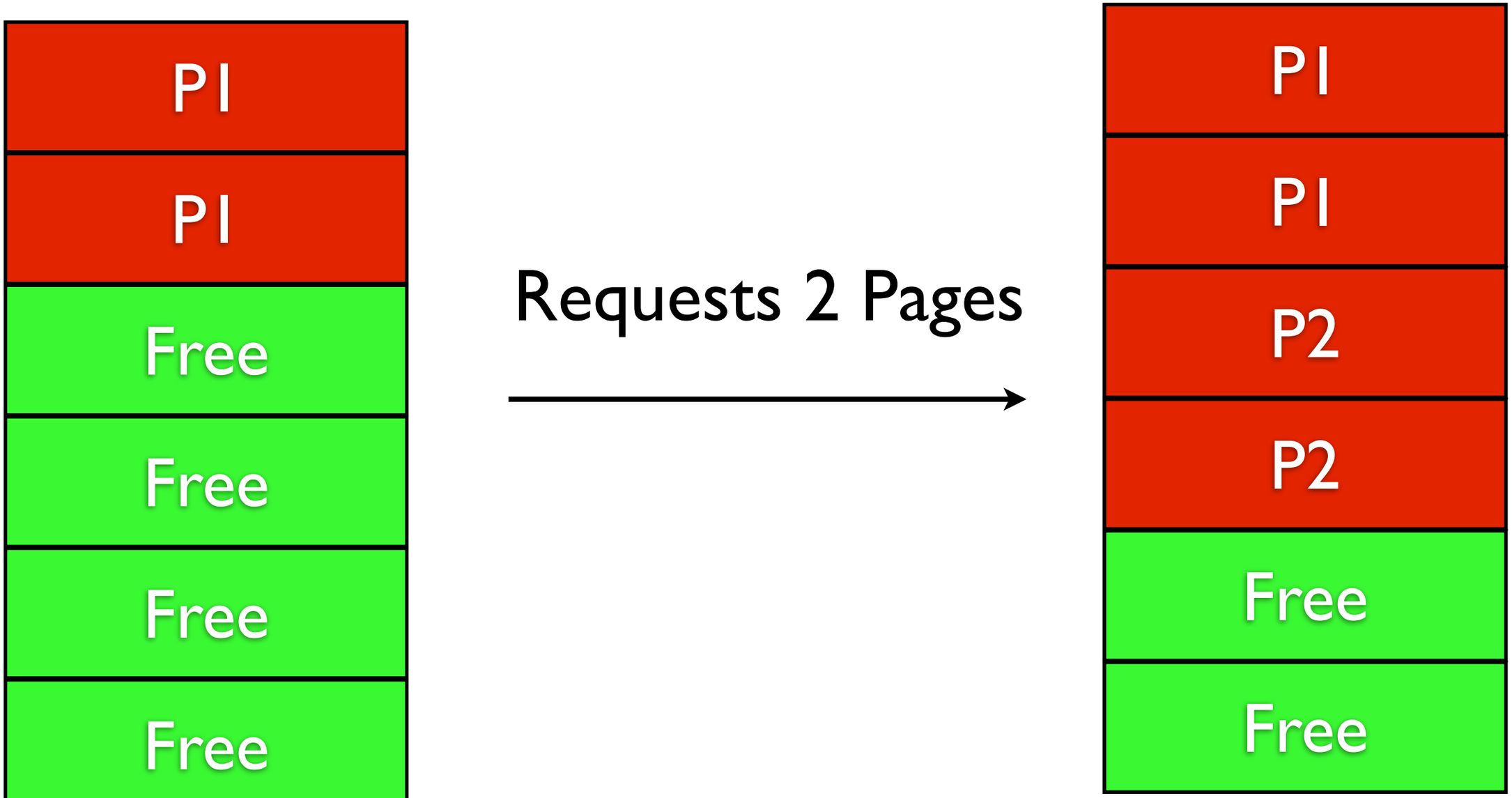
- Implemented a page table in Project #2
- Provides a mechanism for virtualizing memory

Without Virtual Memory

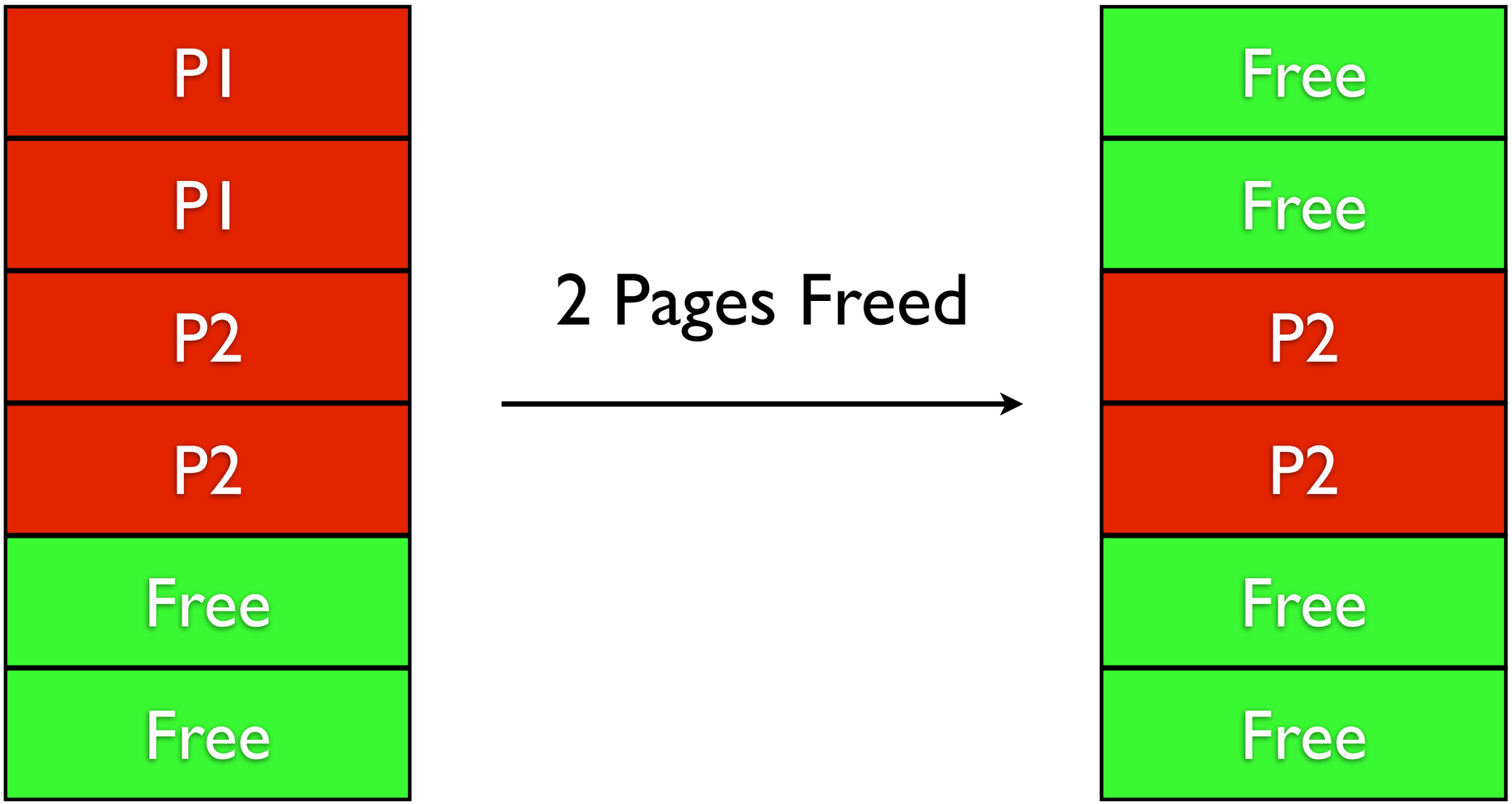
P1 Enters


Free
Free
Free
Free
Free
Free
Free
Requests 2 Pages
P1
P1
Free
Free
Free
Free

P2 Enters


P1
P1
Free
Free
Free
Free
Requests 2 Pages
P1
P1
P2
P2
Free
Free

P1 Exits



P3 Enters

Requests 3 Pages

???

Problem

- Not enough contiguous free memory to allocate request
- Plenty of free memory in total
- This is external fragmentation

VM Advantage

- Avoids external fragmentation
- Far more flexible

Paging

- Way to exploit virtual memory
- Idea: use memory as a cache for the whole disk
- Virtual memory makes this caching transparent to processes

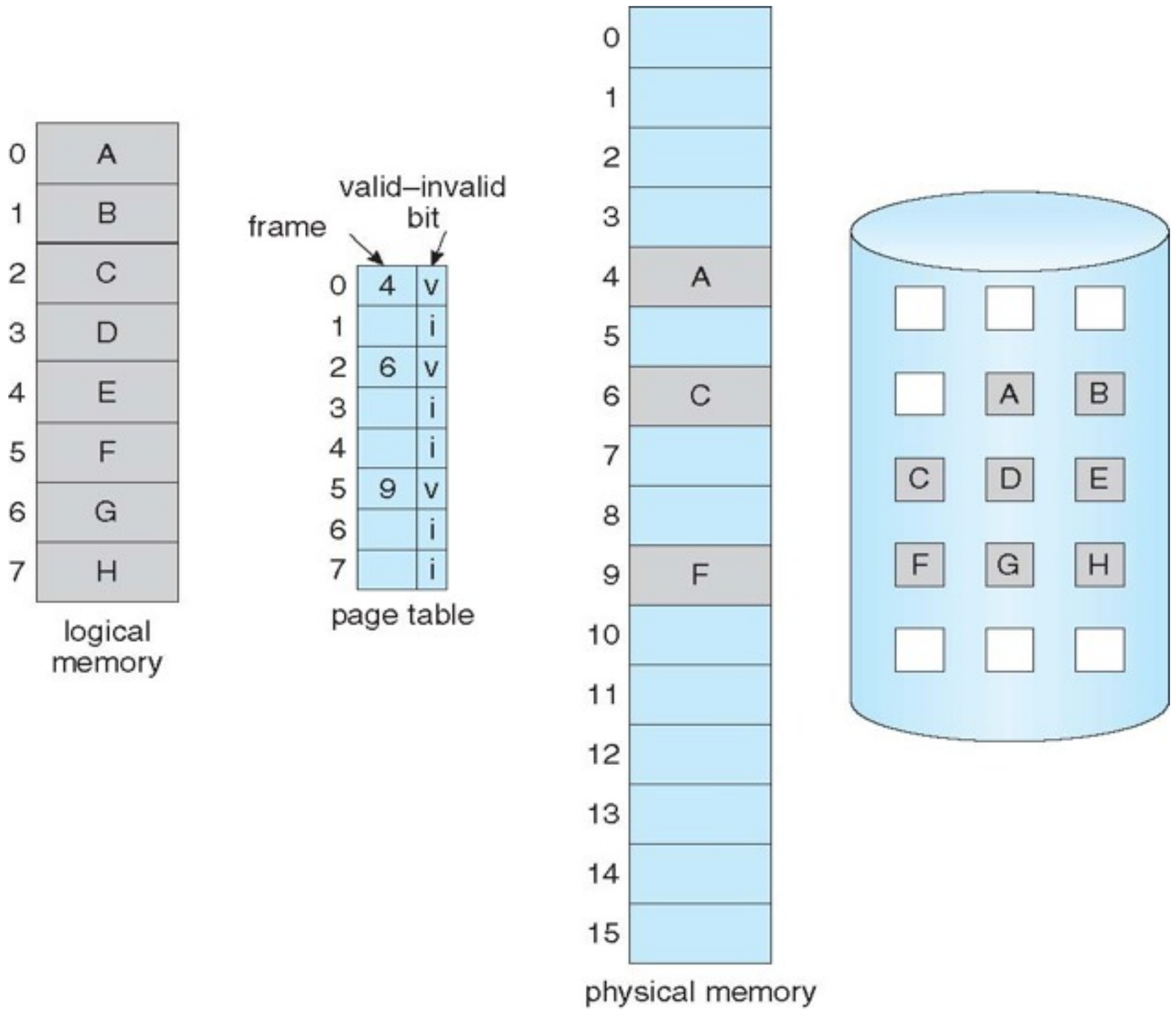
Paging Details

- In an ideal world, each page has the following bits:
  - Valid?
  - Dirty?
  - Referenced?
- NACHOS is an ideal world

Valid Bit

- Project #2: does this process have access to the given page?
- Modern OS / Project #3: is this page in memory and/or does this process have access to it?

Valid Bit Example



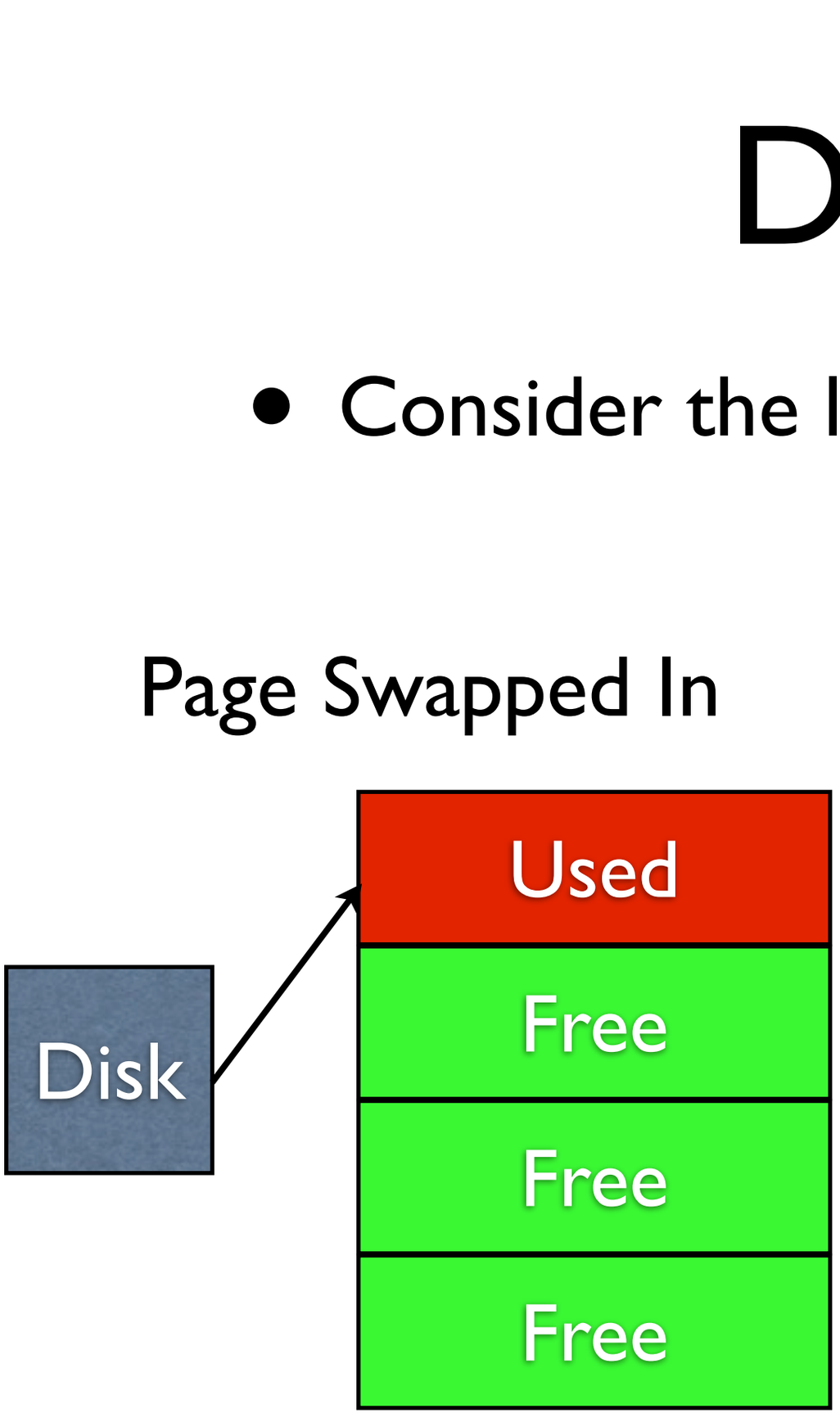
What of Permissions?

- Reference to page with invalid bit set traps to OS anyway
- More accurately, it's a "Trap to OS on Use" bit
- Still need to check if caller can access page

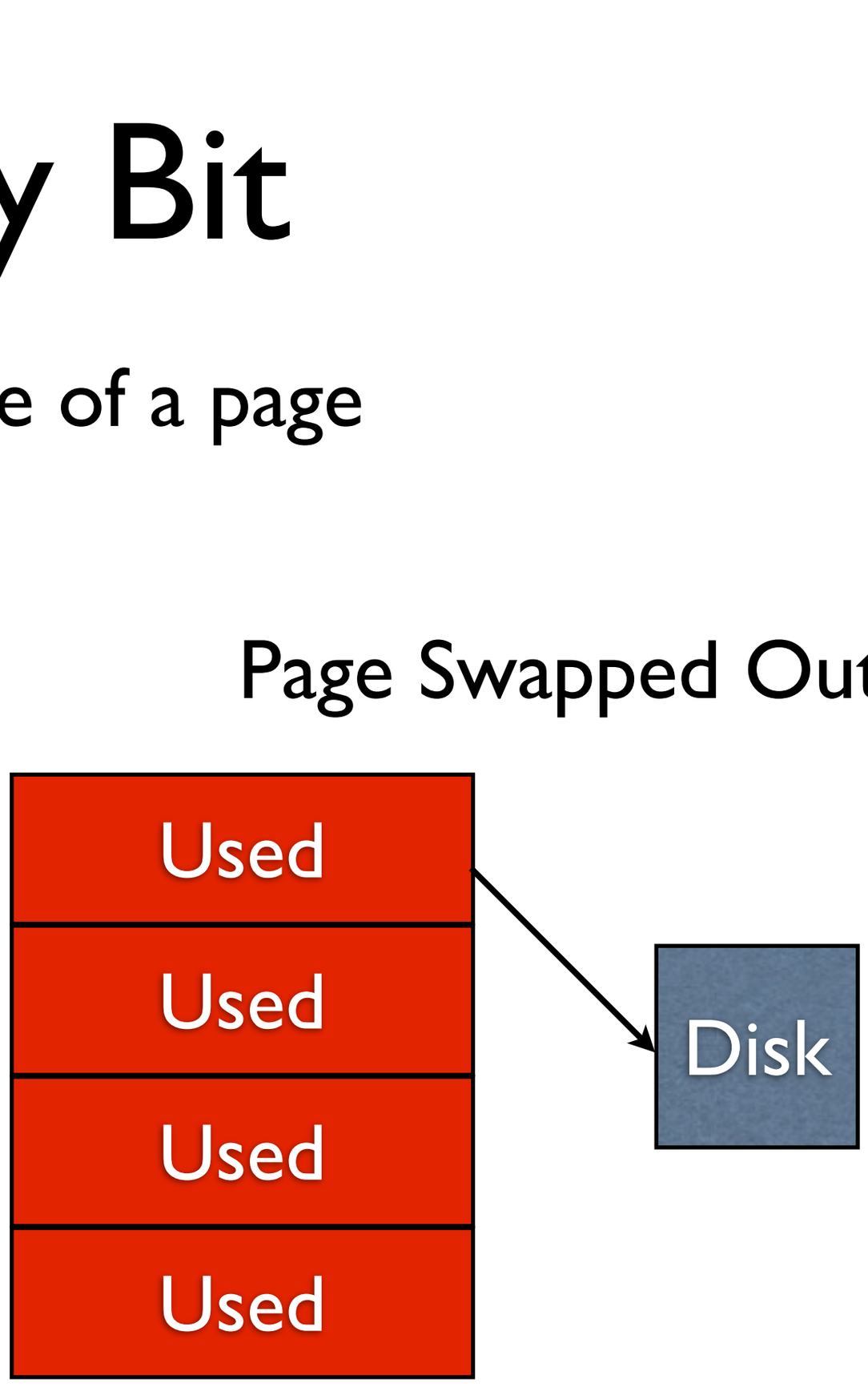
Dirty Bit

- Consider the lifetime of a page

Page Swapped In



Page Swapped Out



Swapping Out

- What if a page is read-only?
- What if a page was never modified since being swapped in?
- In these cases, we end up paging out information that is the same as what is already on disk
  - Complete waste!

Dirty Bit

- If a page is modified, the dirty bit is set (by hardware)
- Only write out if dirty bit is set
- Potentially cuts I/O on paging in half

Referenced Bit

- If a page is used, set the referenced bit (by hardware)
- Allow software to reset the bit
- Makes certain algorithms easier to implement

Question: Can kernel pages be paged out?

Project #3 Task 1

- Implement paging with either FIFO with second chance or LRU
- Step-by-step implementation instructions included

- Stages in Demand Paging

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted

Virtual Filesystems

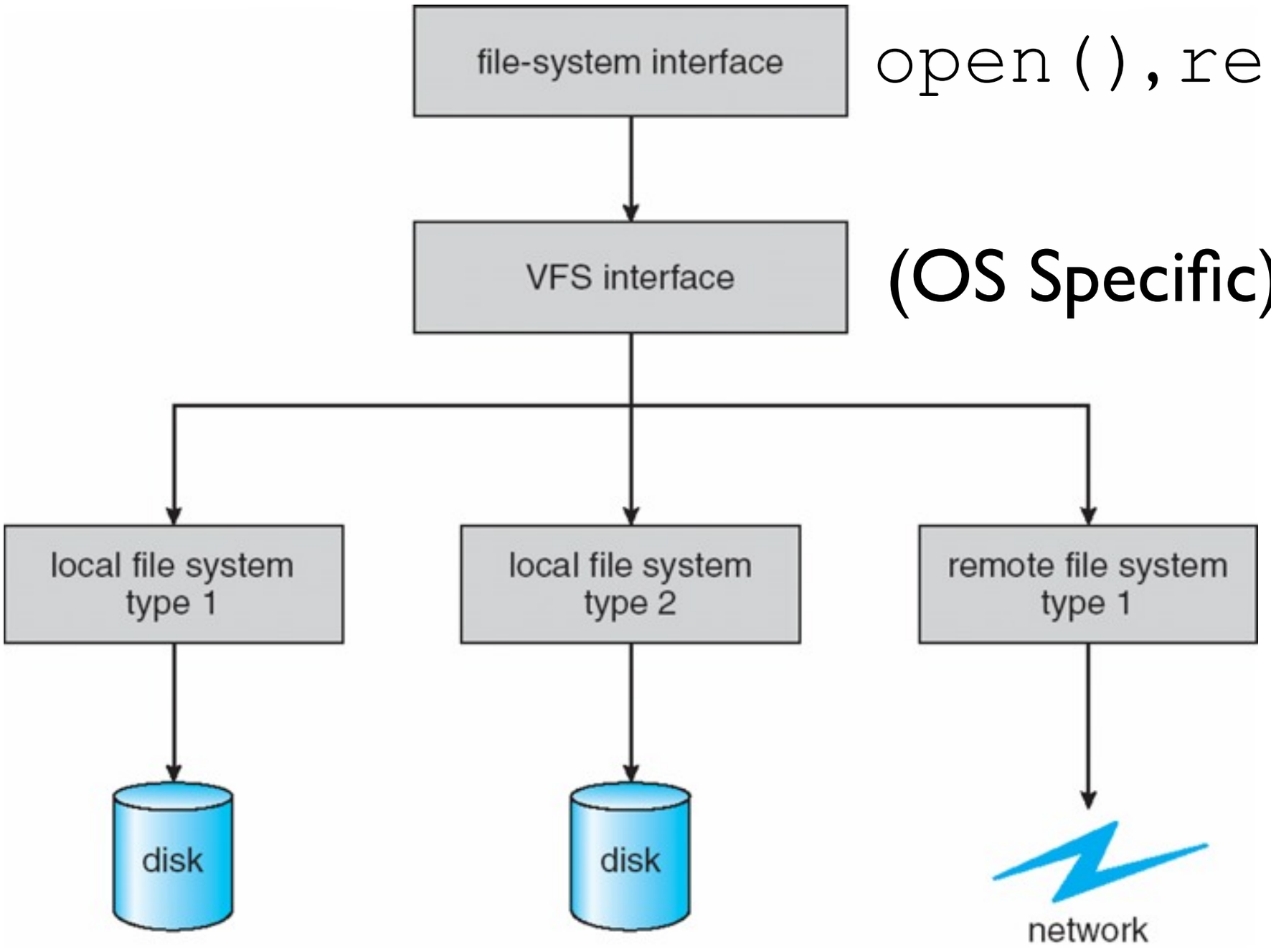
Behold!

```
[kyledewey@csil ~]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda6              26G  8.9G   16G  37% /
tmpfs                 4.0G  1.4M  4.0G   1% /dev/shm
/dev/sda1             504M   68M  411M  15% /boot
/dev/sda7              26G  2.7G   22G  12% /local
/dev/sda3             4.0G  137M  3.7G   4% /tmp
/dev/sda2             7.9G 1011M  6.5G  14% /var
odin:/local/home/faculty
                      268G  171G   84G  68% /cs/faculty
letters:/spool/mail   268G   62G  194G  25% /cs/mail
frigga:/local/home/student
                      268G  252G  3.3G  99% /cs/student
frigga:/local/home/class
                      268G  252G  3.3G  99% /cs/class
hal1.engr.ucsb.edu:/fs.real/hal1a/home
                      4.5T 1006G  3.3T  24% /fs/home1
offside:/local/home   547G  451G   69G  87% /cs/arch
[kyledewey@csil ~]$ ls /cs
arch  class  faculty  mail  student
[kyledewey@csil ~]$
```

Virtual Filesystem

- Puts underlying filesystems into a single, consistent view
- Object-oriented design at its best

Design Hierarchy



`open()`, `read()`...

(OS Specific)

(FS specific)

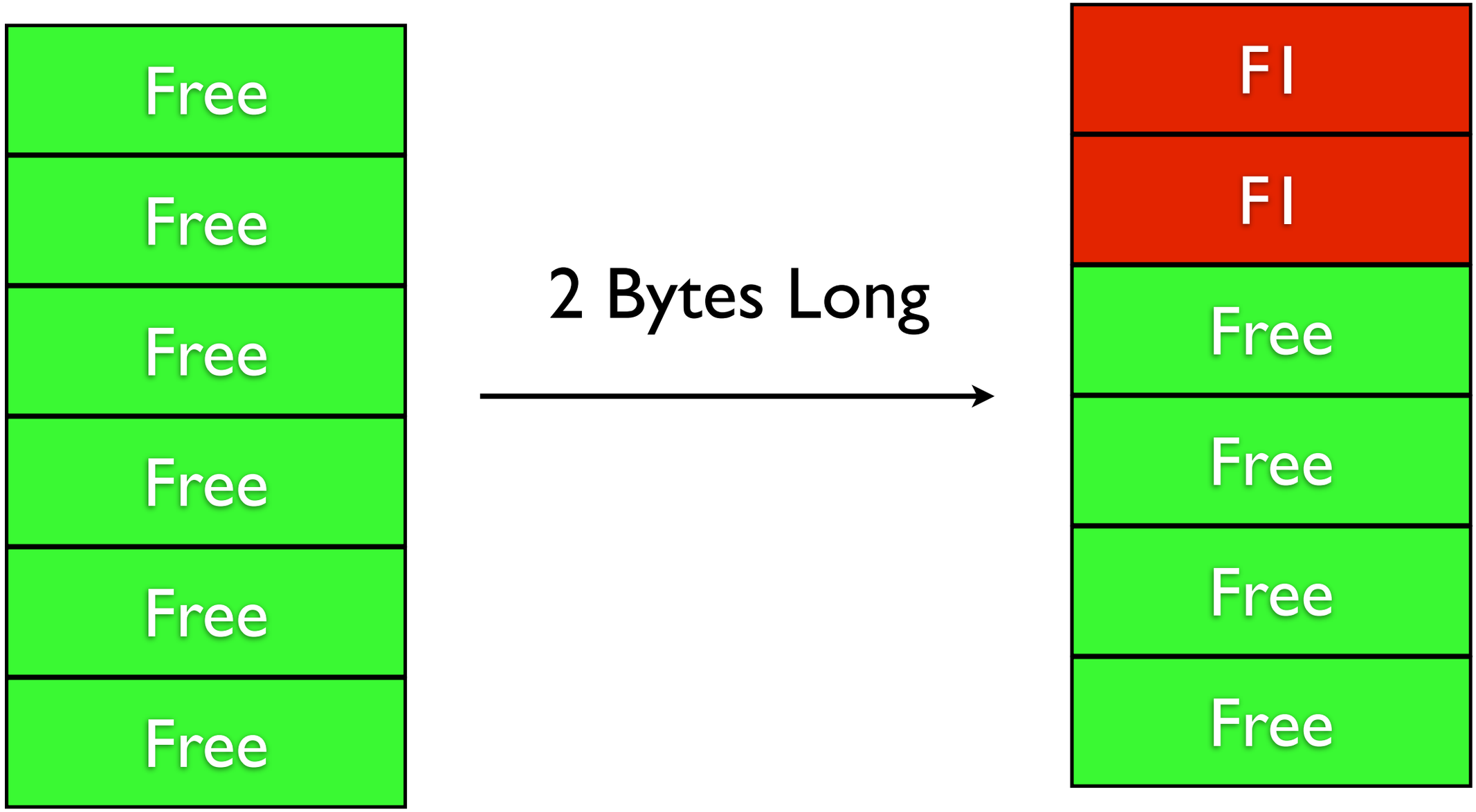
Advantage

- Easy to extend to multiple filesystems
- LOTS of code sharing possible (caching, file management, ...)

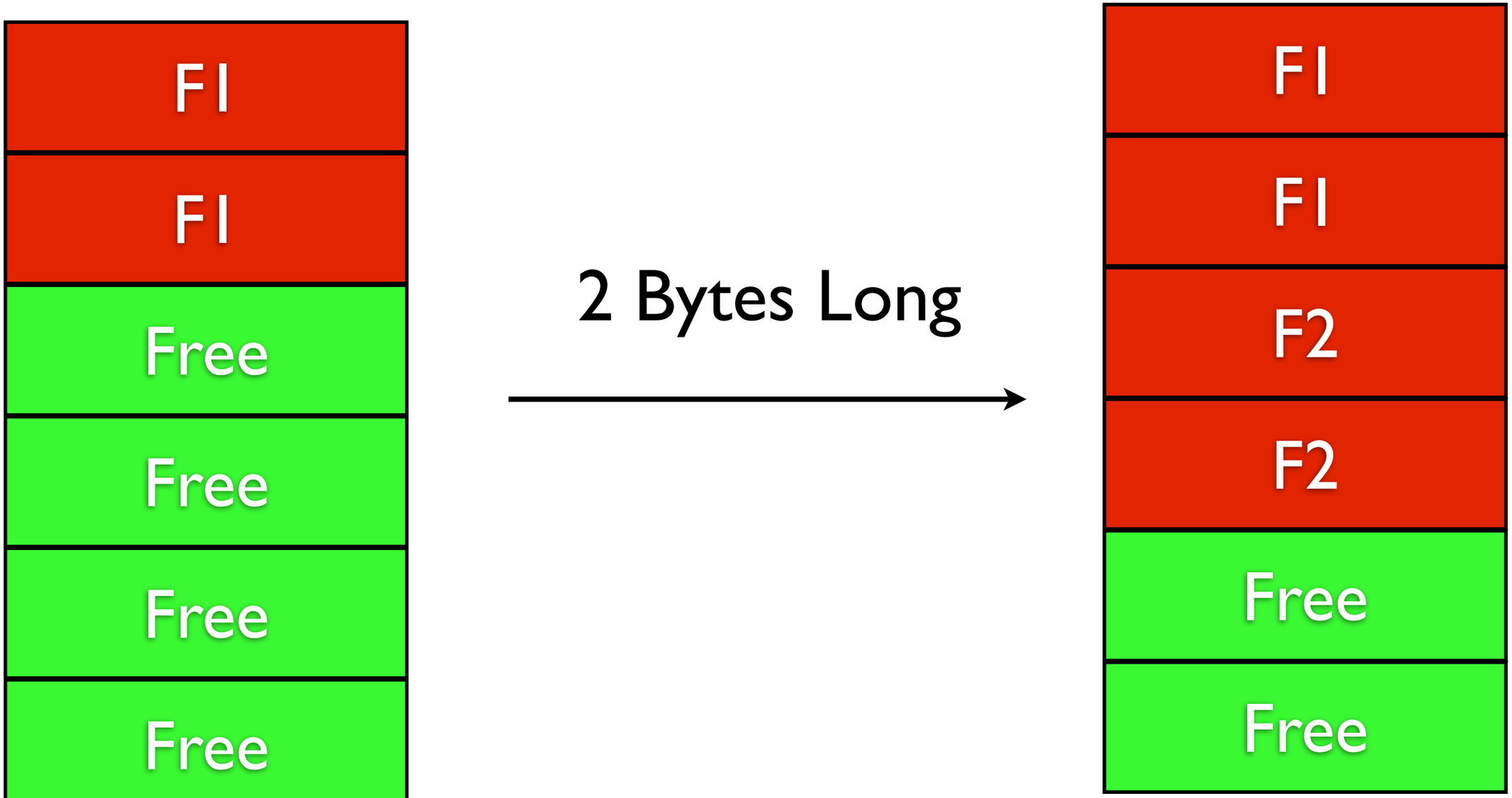
Storing Files

- Similar problems as with physical memory
- Differences:
  - Access times are **MUCH** slower
  - Much better performance if accessing contiguous blocks

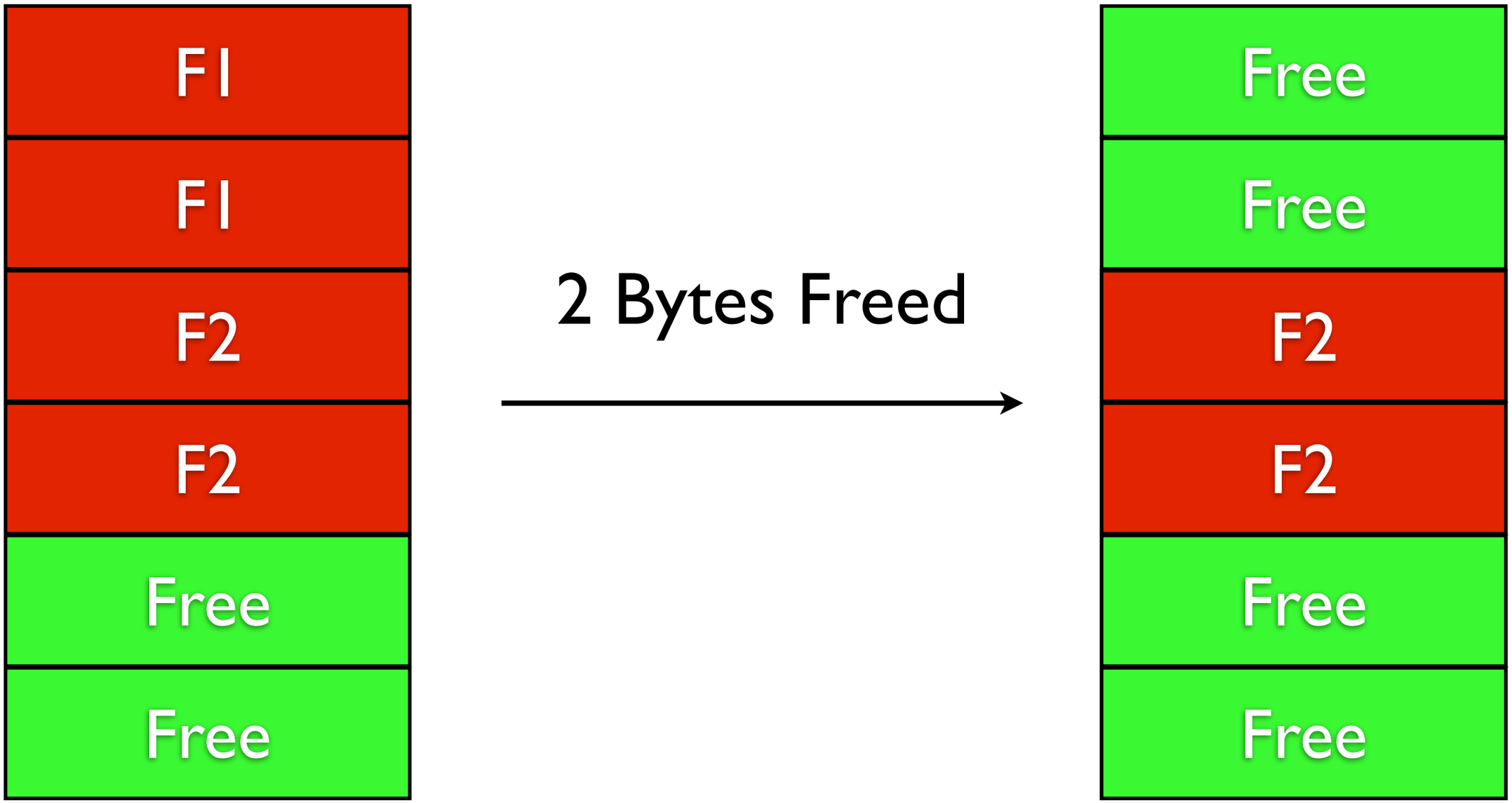
F1 Written


Free
Free
Free
Free
Free
2 Bytes Long
F1
F1
Free
Free
Free
Free

F2 Written



F1 is Deleted



P3 Written

3 Bytes Written

Solution

- Indirection in much the same way as virtual memory
- This is what the File Allocation Table (FAT) filesystem is named for

FAT

- Separates disks into blocks
- Intuitively, blocks are to disk as pages are to memory
- The number afterward defines the number of bits used for an entry

FAT

Intuitively

Block	Next Block
0	∞
1	3
2	9
3	6
4	-1
5	-1
6	∞
7	-1
8	∞
9	8

Actually Stored

Next Block
∞
3
9
6
-1
-1
∞
-1
∞
8

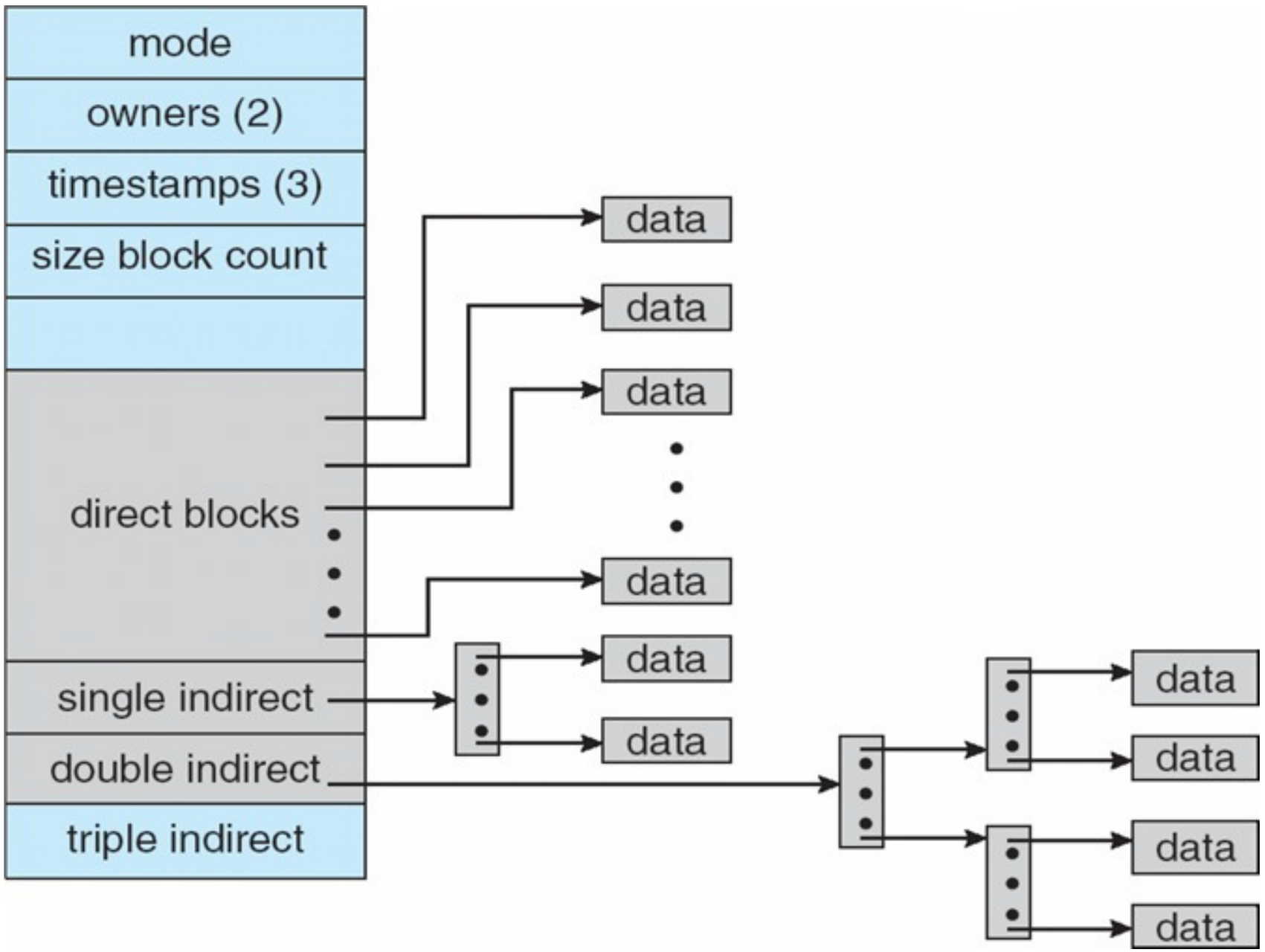
Issues

- Fragile with respect to faults
- Worst case: entire FAT must be read for a single file
  - `seek()` is actually O(n), where n is the number of blocks used in a file

Tradeoff

- Represent some blocks directly, others indirectly
- Make the whole file metadata fit into one block
- UNIX inodes usually do this

UNIX inode


mode
owners (2)
timestamps (3)
size block count
direct blocks
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data

Advantages

- For small files, only direct blocks are needed
  - `seek()` will be O(1)
- Still can represent large files
  - `seek()` will be either O(1) or O(n), depending how far into the file we are seeking

Relevance to NACHOS

- NACHOS has file size limitation of 4 Kb
- Need to extend to 100 Kb
- Will involve adding an indirect level on top of existing direct level

Project #3

Implementation Notes