CSE 451: Operating Systems

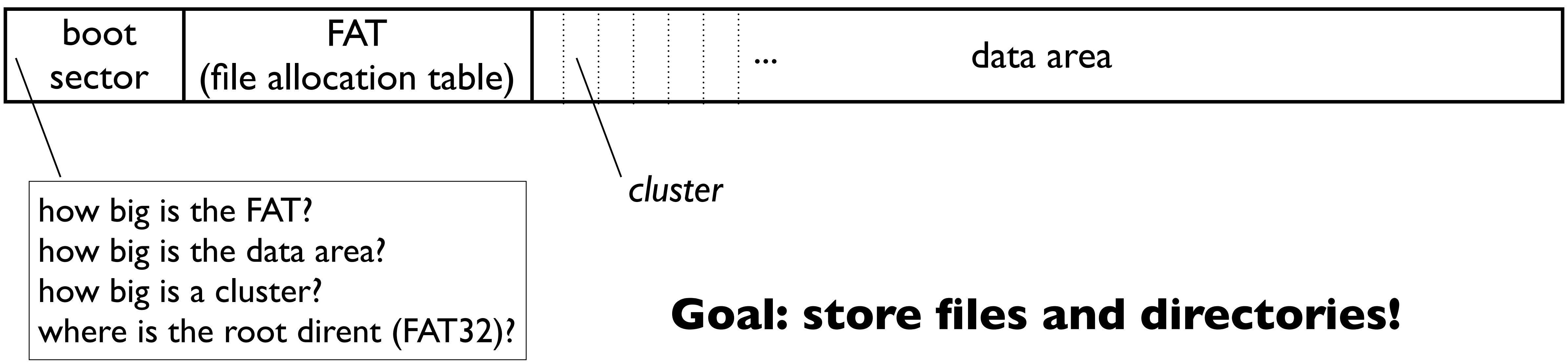
Lab Section: Week 8

Today

- Project 4
- File system issues
  - disk utilization
  - consistency
  - performance

(I believe there is no quiz tomorrow!)

The FAT File System



Goal: store files and directories!

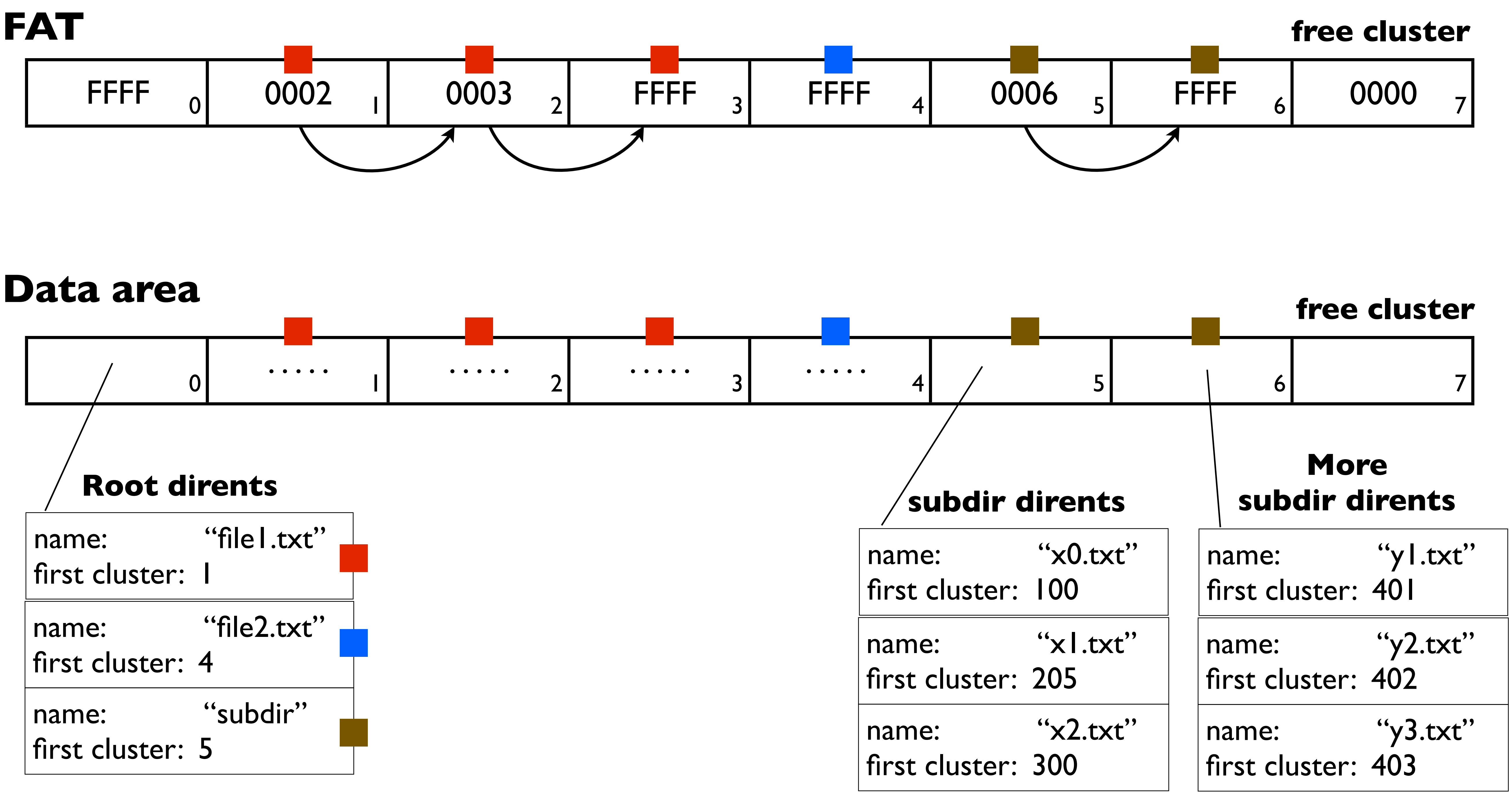
Each cluster has one of two purposes:

- stores data for a file
- stores lists of files in a directory (dirents)

FAT

- linked lists of clusters (for big files and directories)
- which clusters are free?

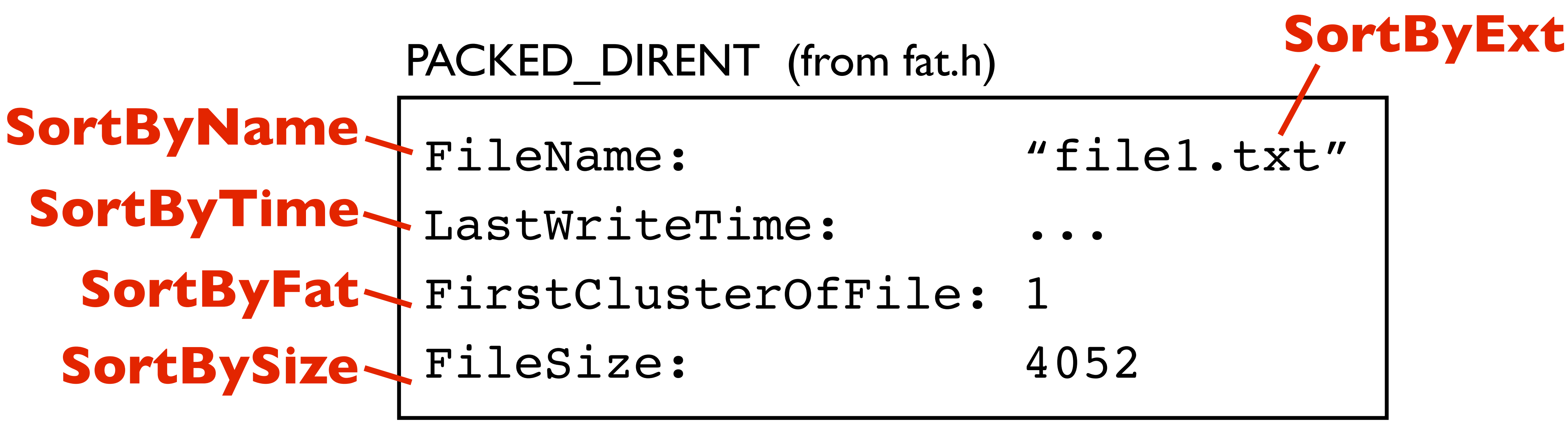
The FAT File System



Project 4

Goal: keep dirents sorted in each directory

- based on *volume label* (more on this in a minute)



(actual volume labels are ALL CAPS)

(PACKED_DIRENT aka DIRENT)

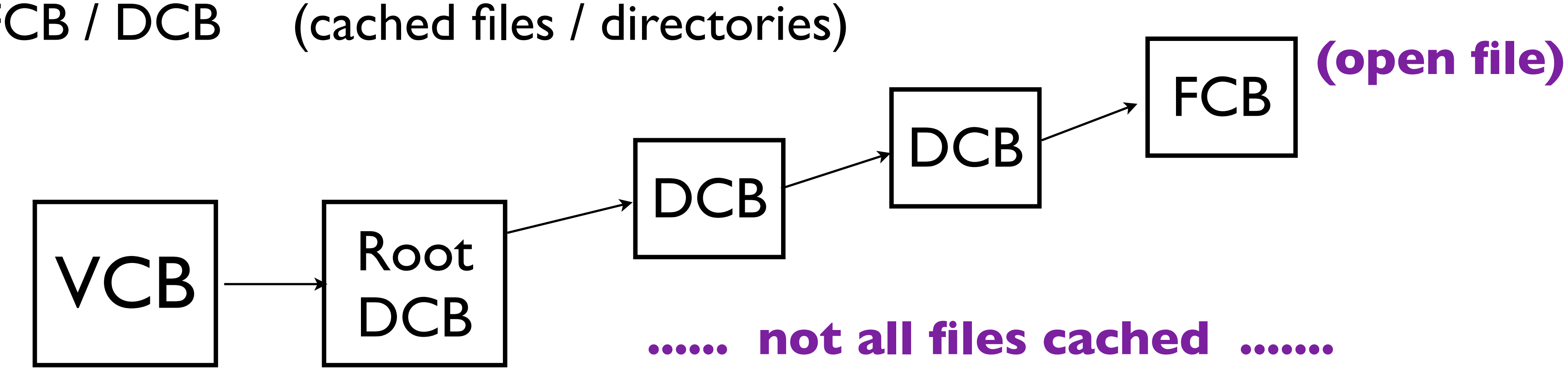
Project 4

Kernel data structures: on-disk FAT (see fat.h)

PACKED_BOOT_SECTOR	(you don't need to use this)
BIOS_PARAMETER_BLOCK	(you don't need to use this: part of boot sector)
PACKED_DIRENT	(aka DIRENT)

Kernel data structures: in-memory FAT (see fatstruc.h)

VCB	(info about a mounted volume)
FCB / DCB	(cached files / directories)



Project 4

- You need to resort the dirents when:
  - creating a new file (SortByName, SortByExt, SortByFat)
  - closing a file (SortByTime, SortBySize)
- So, where do I start????
  - look at FatDefragDirectory() in dirsup.c
  - this compresses a dirent list by removing deleted dirents

    .... very similar to what you need to do

    .... maybe modify this? maybe call it from more places?*
- How do I get the volume label?
  - use VCB→Vpb→VolumeLabel (see FatMountVolume and FatLocateVolumeLabel)
- Extra credit
  - dealing with long file names

*disclaimer: I have not done the project

Project 4

- How do I test my kernel?
  - run the test scripts in Project4/TestScripts
  - look at the output
    (the `dir` command prints dirents in the order they are on disk)

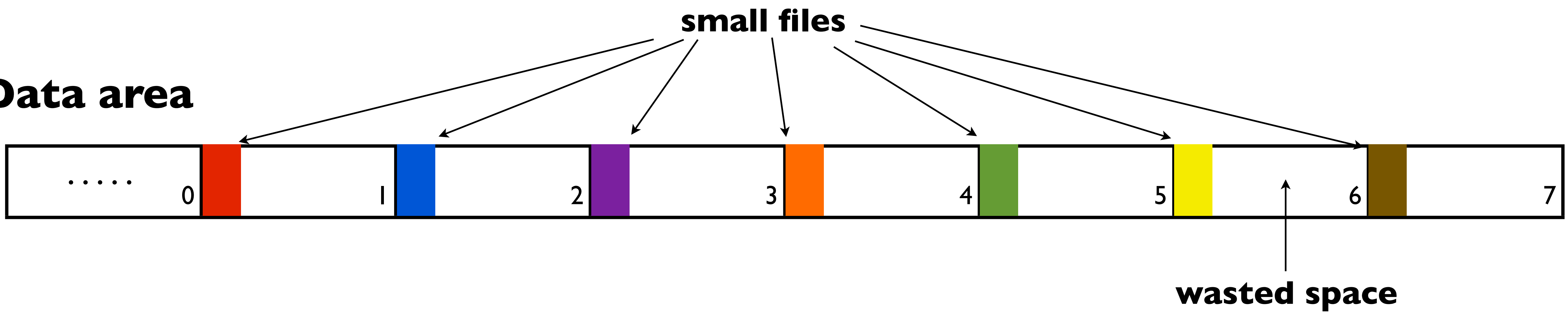
Today

- ~~Project 4~~
- File system issues
  - disk utilization
  - consistency
  - performance

(I believe there is no quiz tomorrow!)

Disk Utilization

Data area



FAT16

Drive Size	Cluster Size
32 MB - 64 MB	1 KB
64 MB - 128 MB	2 KB
128 MB - 256 MB	4 KB
256 MB - 512 MB	8 KB
512 MB - 1 GB	16 KB
1 GB - 2 GB	32 KB

how big can we get?

Windows cuts off at 4GB

16-bit FAT pointers:

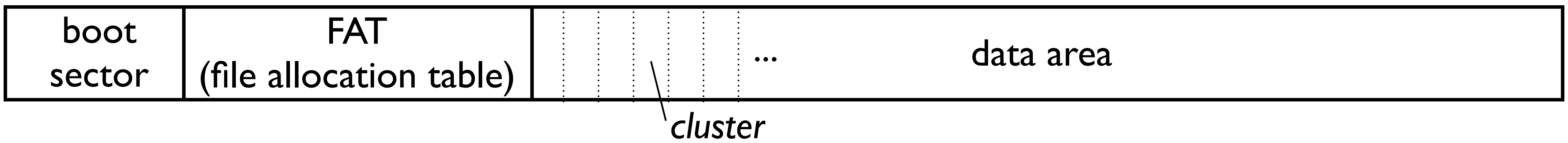
$$\text{driveSize} = 2^{16} \cdot \text{clusterSize}$$

Say your disk was full of 1KB files ...

- pathological, but not completely crazy
  (a typical system has many small directories and text files)

96.9% of HD is wasted!

Disk Utilization



- Big clusterSize is bad
  - wasted space
- Maximum disk size supported ≈ numClusters · clusterSize
  - for FAT16, numClusters = 2^{16}
  - for FAT32, numClusters = 2^{32}

FAT16 (16-bit fat ptrs)

Drive Size	Cluster Size
32 MB - 64 MB	1 KB
64 MB - 128 MB	2 KB
128 MB - 256 MB	4 KB
256 MB - 512 MB	8 KB
512 MB - 1 GB	16 KB
1 GB - 2 GB	32 KB

max drive size: 4GB

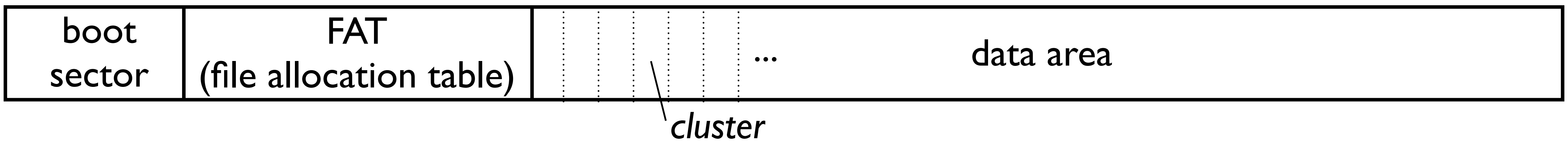
FAT32 (32-bit fat ptrs)

Drive Size	Cluster Size
32 MB - 64 MB	512 bytes
64 MB - 128 MB	1 KB
128 MB - 256 MB	2 KB
256 MB - 8 GB	4 KB
8 GB - 16 GB	8 KB
16 GB - 32 GB	16 KB

max drive size: 2TB*

***depends on OS**

Disk Utilization

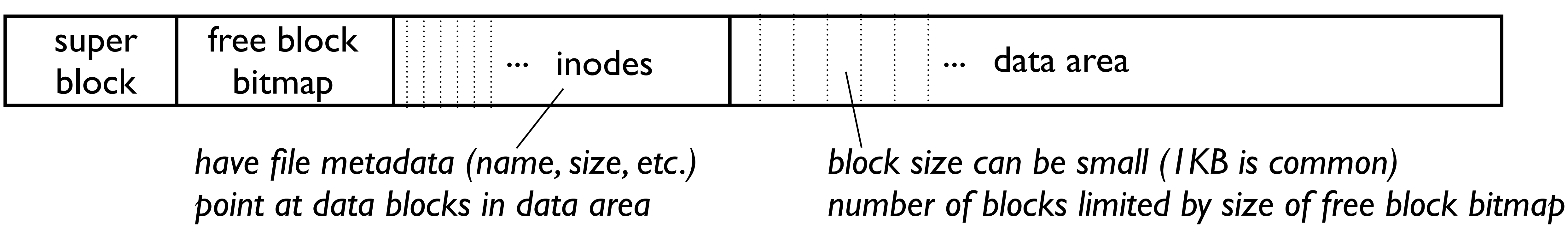


- Is FAT64 a good idea?
  - *no* ... would require storing 2^{64} 64-bit entries on disk (many exobytes!)
  - could just limit the number of files, but 64-bits per entry feels like a lot ...
- New idea: eliminate FAT!
  - use <u>block bitmap</u> (one bit per entry: if bit=1, the block is free)
  - store file data pointers in <u>inodes</u>

Disk Utilization

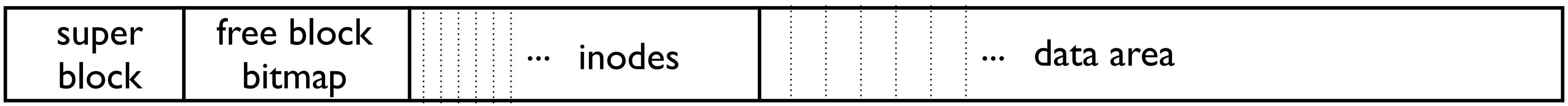
- Is FAT64 a good idea?
  - *no* ... would require storing 2^{64} 64-bit entries on disk (many exobytes!)
  - could just limit the number of files, but 64-bits per entry feels like a lot ...
- New idea: eliminate FAT!
  - use <u>block bitmap</u> (one bit per entry: if bit=1, the block is free)
  - store file data pointers in <u>inodes</u>

Unix FS (many other file systems roughly similar)



Inodes

Unix FS (many other file systems roughly similar)



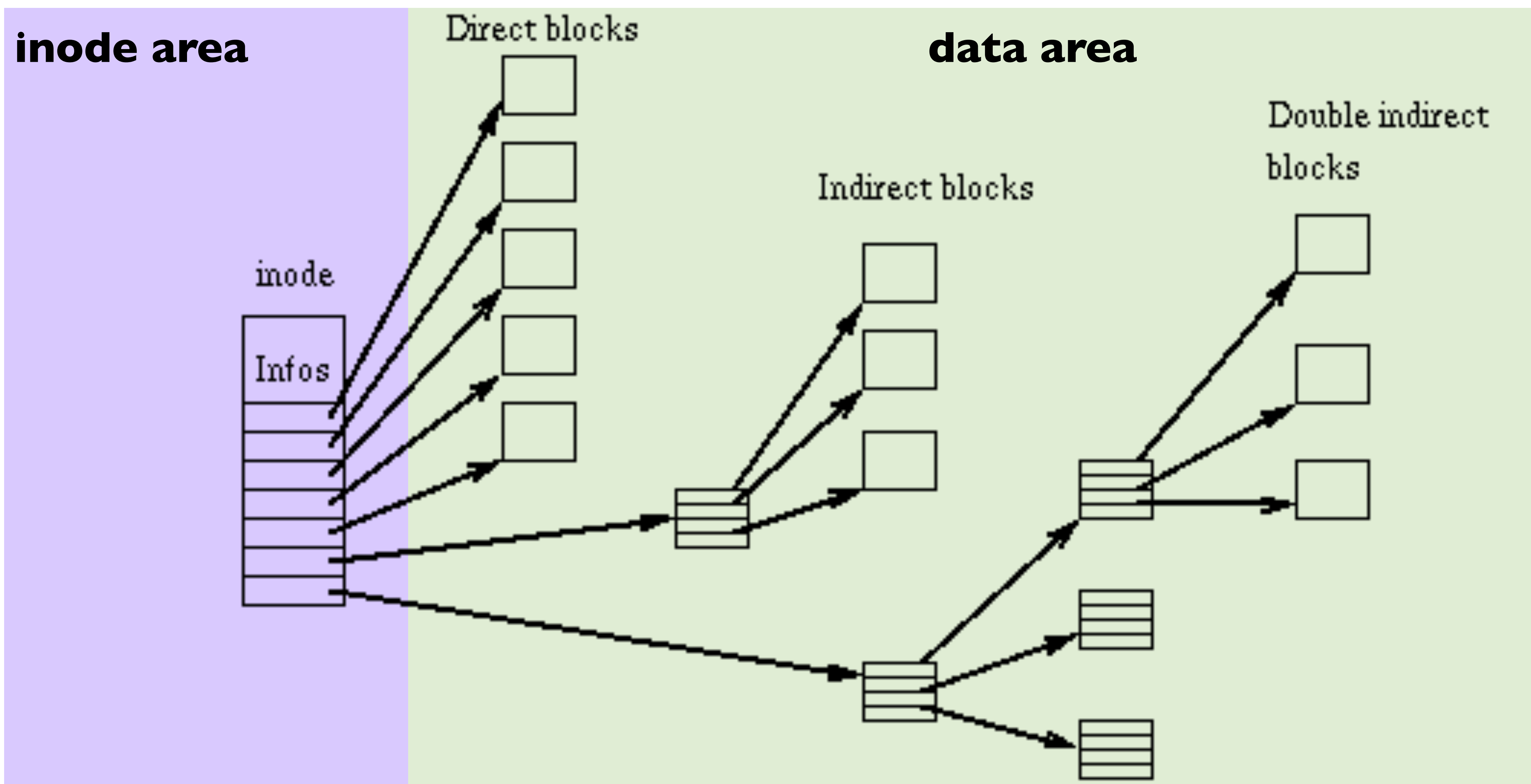


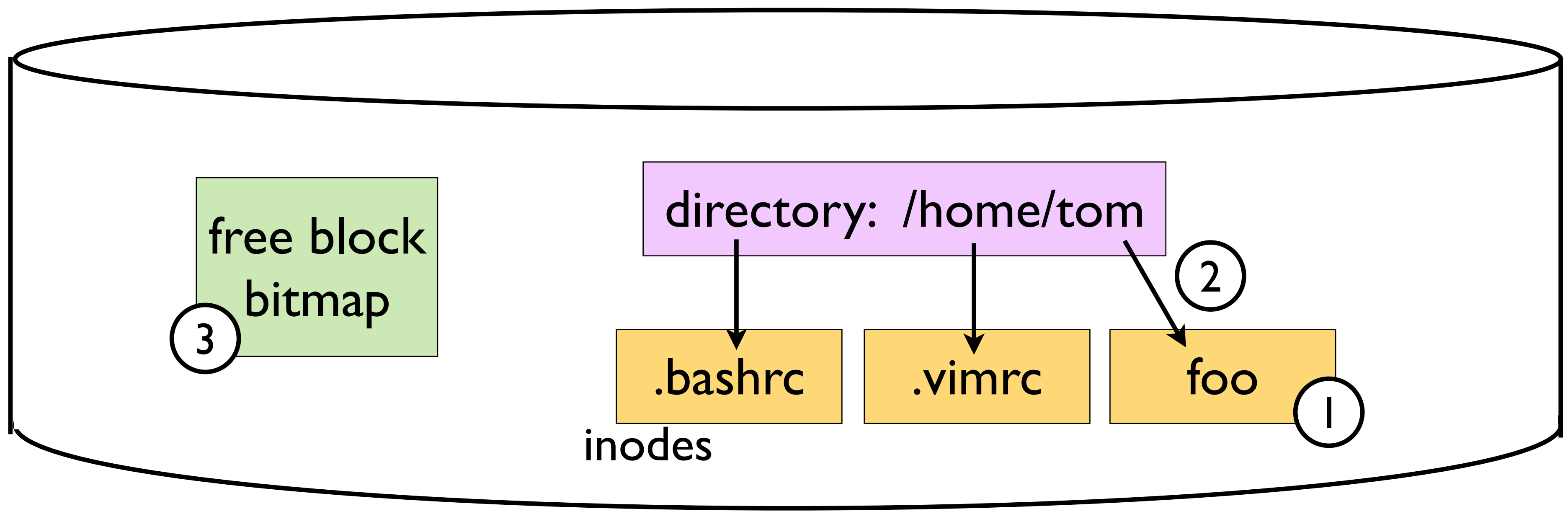
image courtesy of Wikipedia

Today

- ~~Project 4~~
- File system issues
    - ~~disk utilization~~
    - consistency
    - performance

Consistency

How do we create a new file?



In some order:

(1) - write inode data to a free block

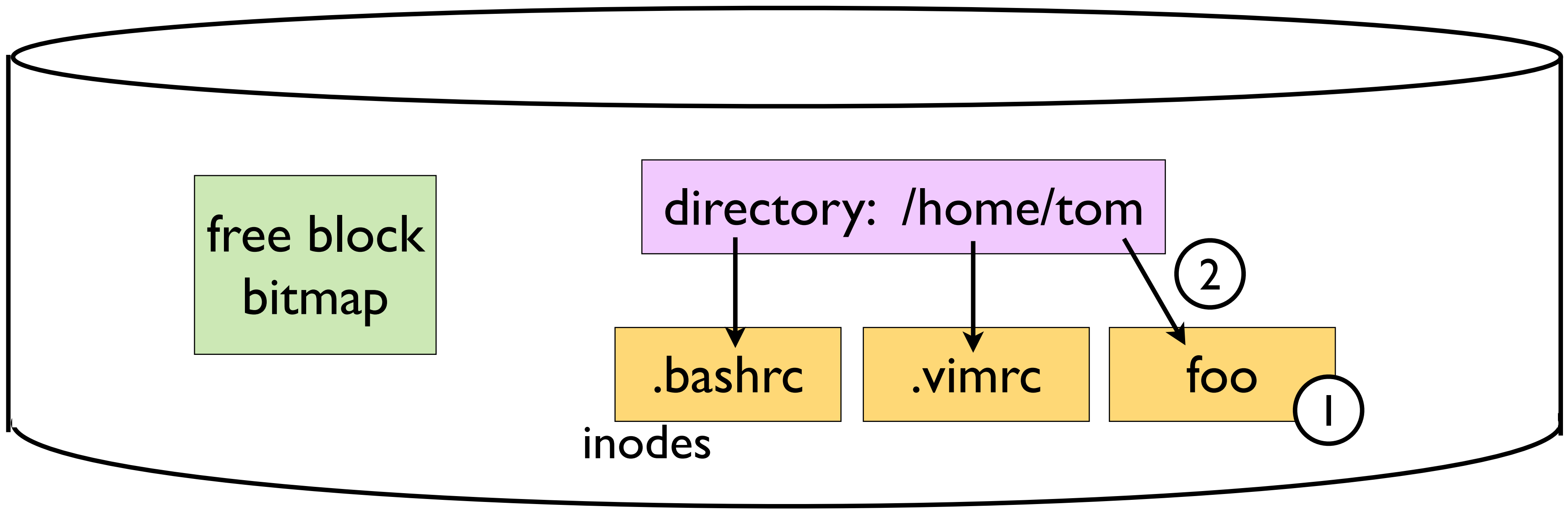
(2) - link directory to new inode

(3) - update bitmap

Hmm ... what order do we do them in?

Consistency

How do we create a new file?



In some order:

(1) - write inode data to a free block

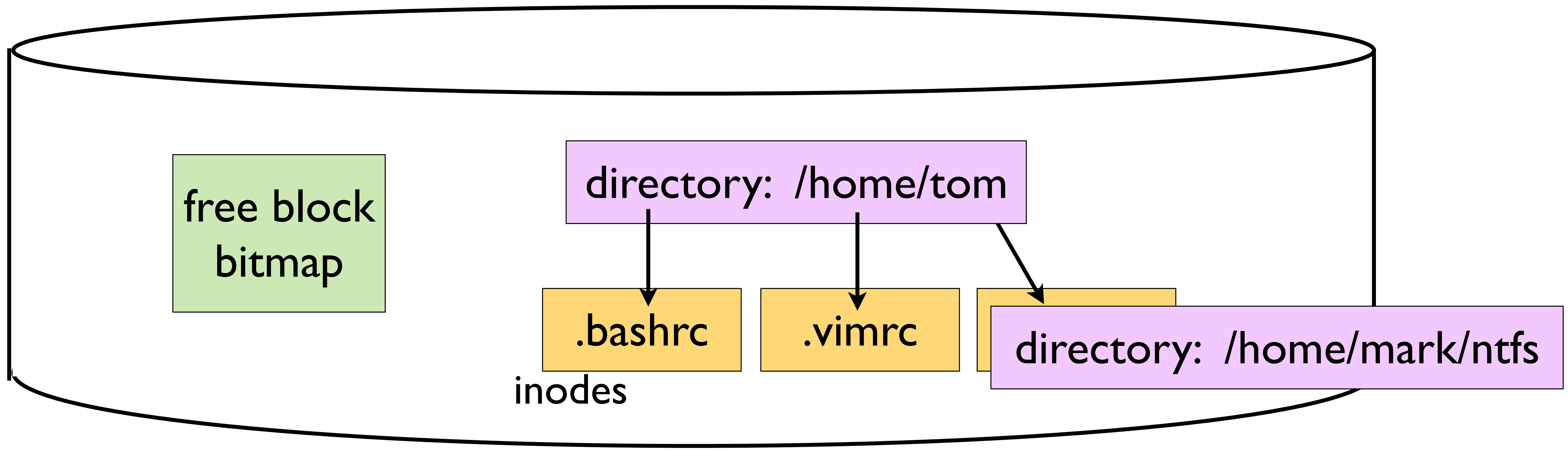
(2) - link directory to new inode

- update bitmap

Hmm ... what order do we do them in?

Consistency

How do we create a new file?



In some order:

- write inode data to a free block
- link directory to new inode
- ~~update bitmap~~

oops, we didn't get to this!

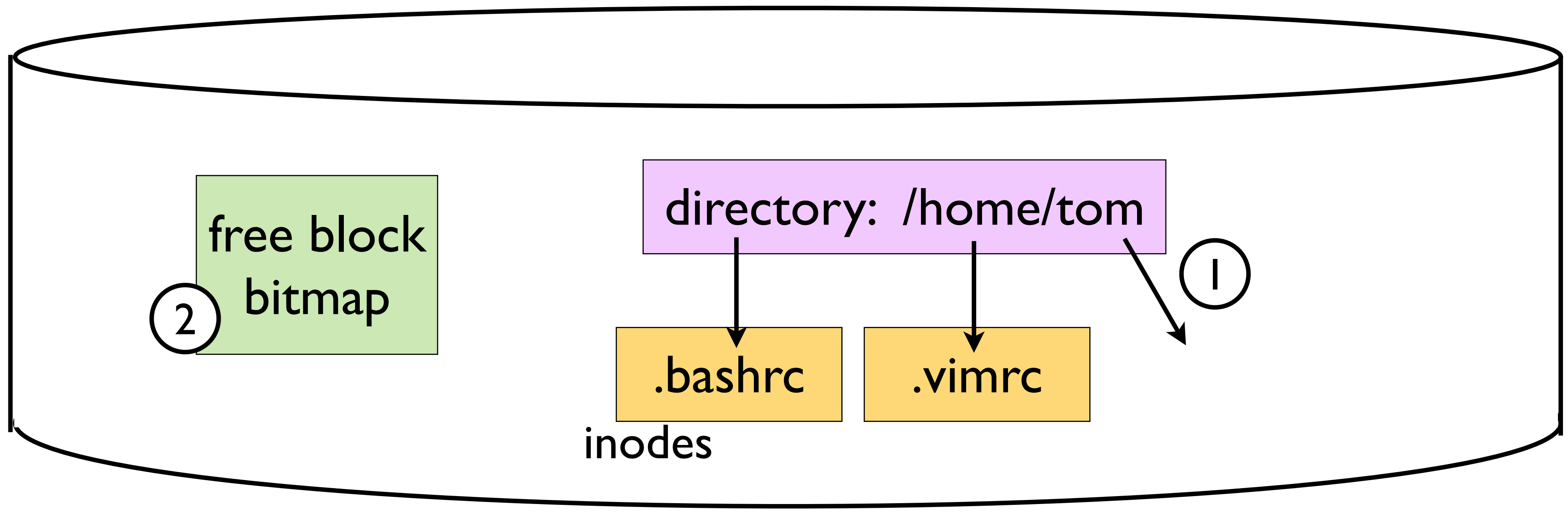
That block can be reused!!!

- what if it gets allocated as a directory?
- really bad:

  .... by writing to /home/tom/foo, I can change the metadata for another user's directory!

Consistency

How do we create a new file?



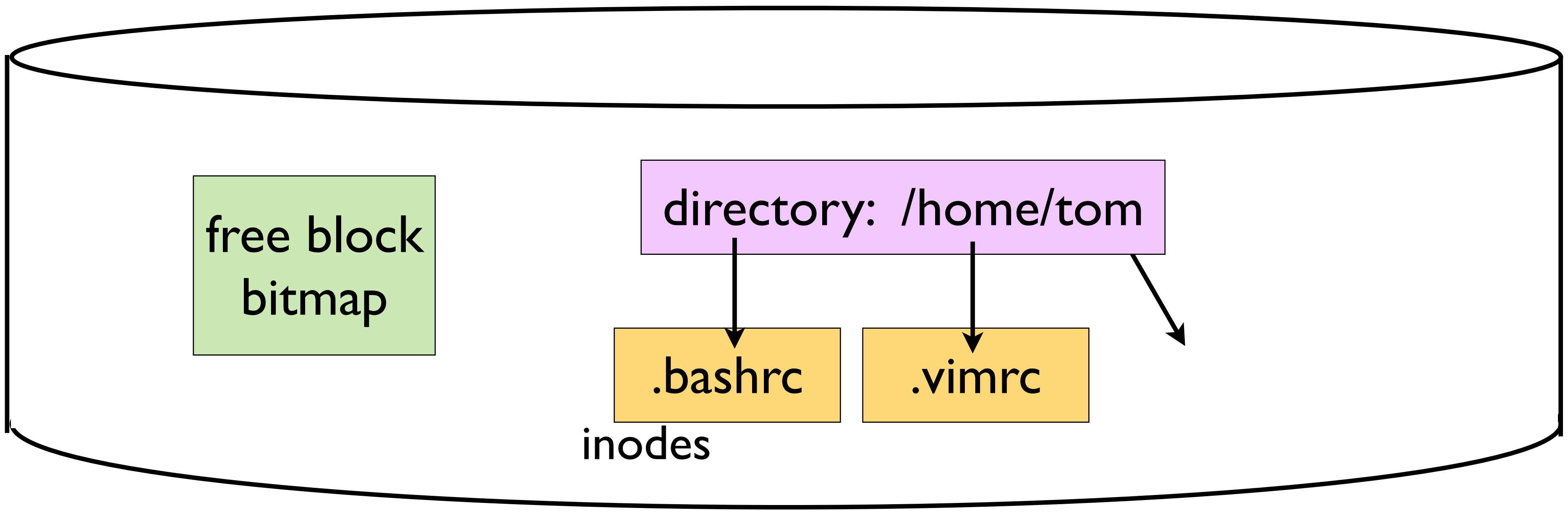
In some order:

- write inode data to a free block
- (1) - link directory to new inode
- (2) - update bitmap

Hmm ... what order do we do them in?

Consistency

How do we create a new file?



In some order:

- ~~write inode data to a free block~~
- link directory to new inode
- update bitmap

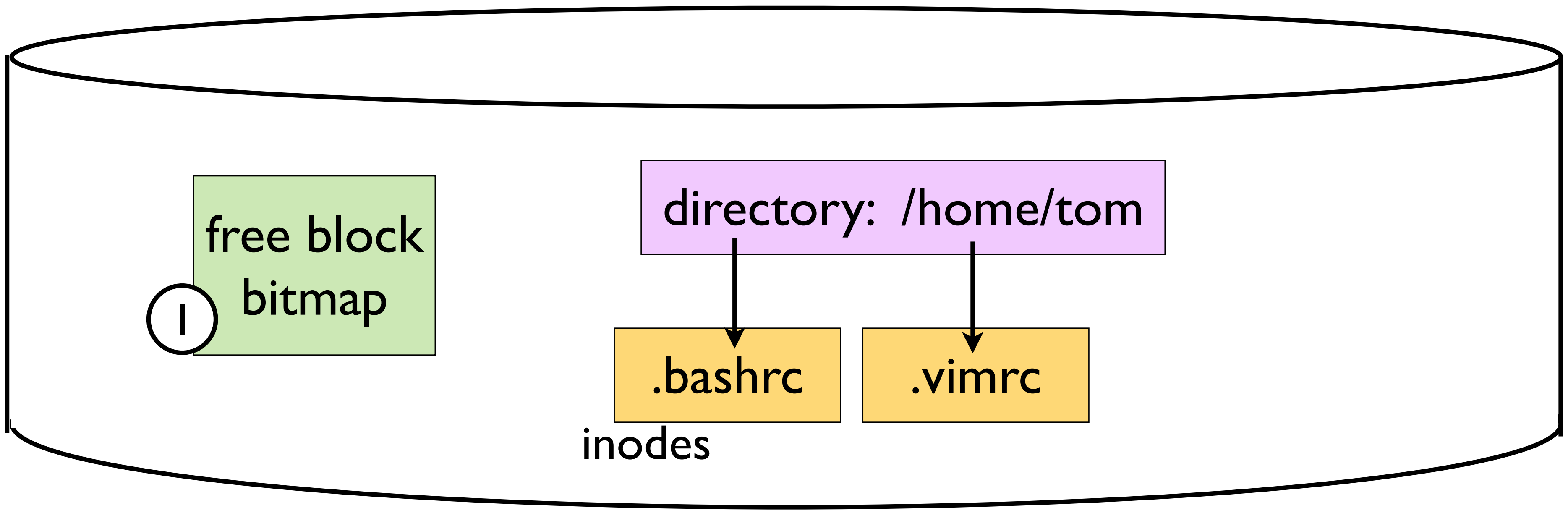
oops, we didn't write the inode!

The inode has garbage!!!

- dangling pointer

Consistency

How do we create a new file?



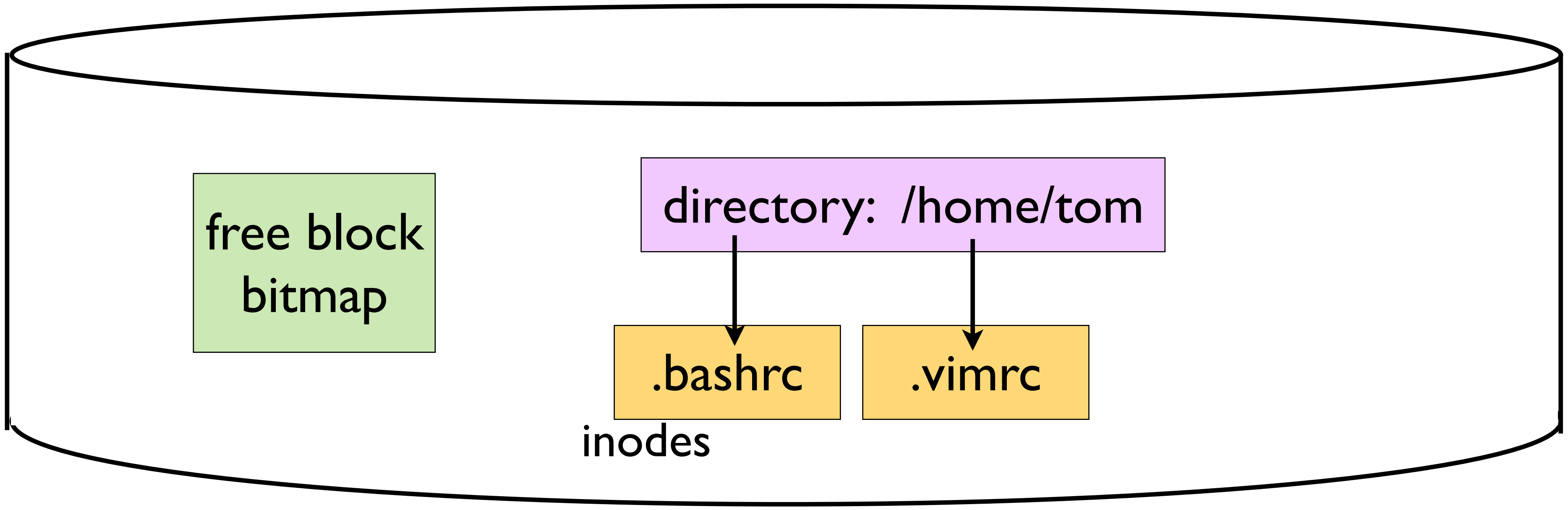
In some order:

- write inode data to a free block
- link directory to new inode
- (1) update bitmap

Hmm ... what order do we do them in?

Consistency

How do we create a new file?



In some order:

- ~~write inode data to a free block~~
- ~~link directory to new inode~~
- update bitmap

oops!

Block is marked used, but not linked to!

- this actually isn't that bad
- we can garbage collect the unused block
  (fsck: this takes time ...)

Consistency

Moral of the story

- file system consistency is *hard*

We wanted to do three things *atomically*:

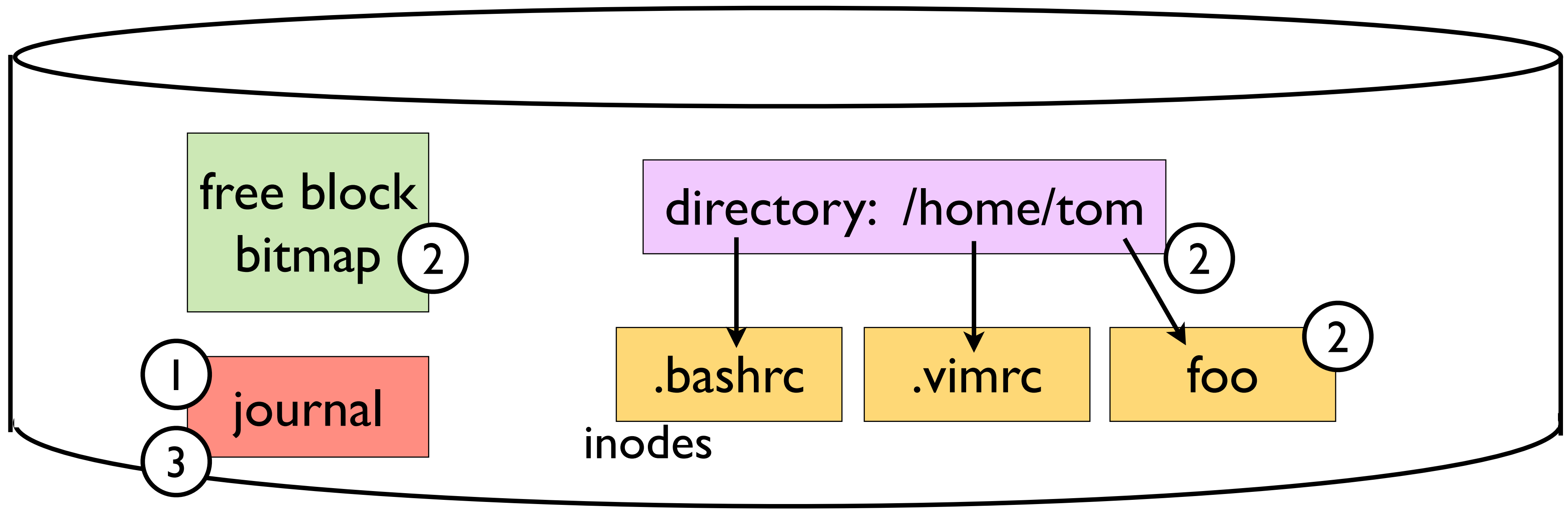
- write inode data to a free block
- link directory to file's inode
- update bitmap

How?

- transactions!

Journaling file systems

Add an *undo log*



To create a new file:

(1) add an *undo* entry to the journal

(2) do create file operations in any order (update bitmap, add link, write inode)

(3) add a *commit* entry to the journal

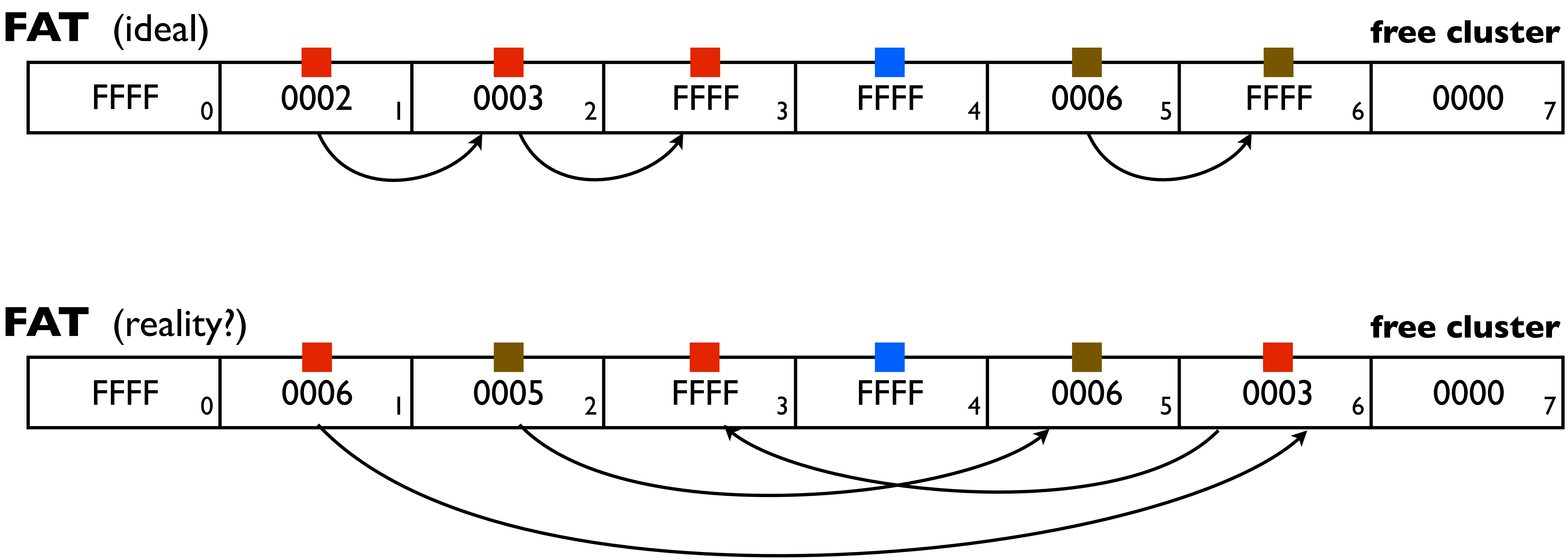
After a crash:

- undo everything after the last *commit* entry

Today

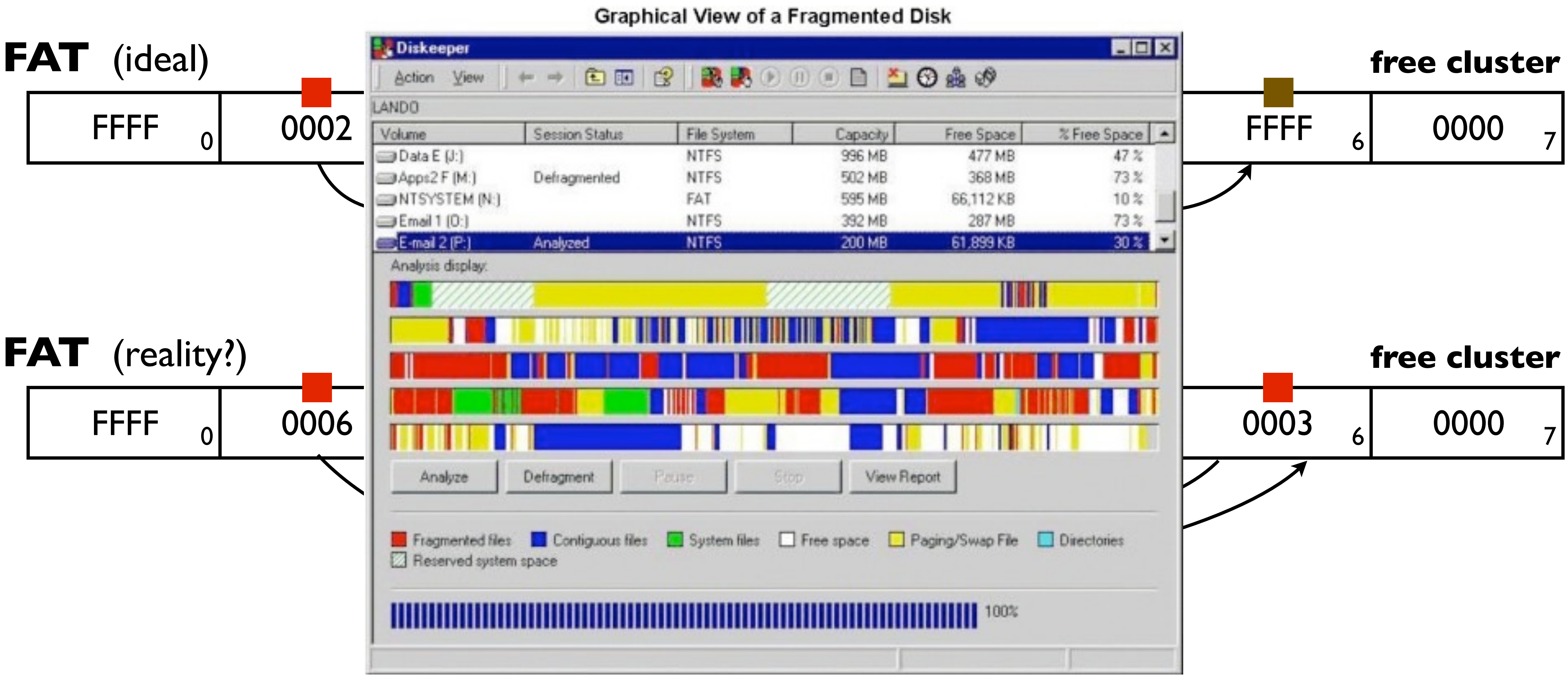
- ~~Project 4~~
- File system issues
  - ~~disk utilization~~
  - ~~consistency~~
  - performance

FAT32 performance



reading and writing:
many seeks

FAT32 performance



Fragmentation is bad!

Observed trends in HDs

Disk bandwith increases w/ new HDs

- writing to the disk in large contiguous chunks is cheap, and getting cheaper
- but seeking is still slow

Think about FAT:

... to update a file, you have to update the data blocks and the FAT

... requires seeking (bad!)

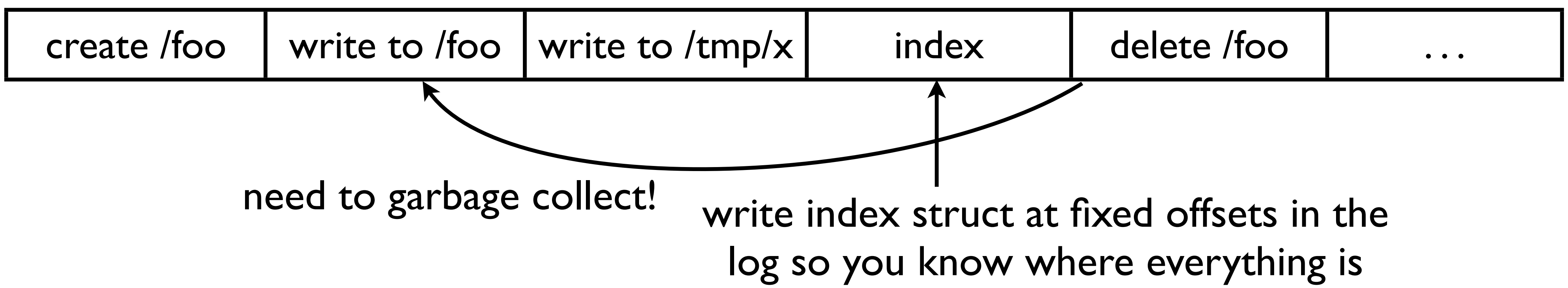
Memory capacity is increasing

- we can build large caches
- most reads can hit the cache?
- can coalesce small writes?

Log-structured File Systems

(a crazy idea from 1988)

Make the disk one big log
(a very high level overview)



Advantages:

- writes are super fast (can coalesce, and do one big write to end-of-log)

Disadvantages:

- complicated (when do you garbage collect?)
- what if my read set is too big to fit in the cache?
  (LFS can actually have worse fragmentation than other file systems!)