

# File Systems (Part 2)

# Outline

- **Further speed improvements**
  - clustering
  - extents
  - log-structured file systems
- **Static vs. dynamic inodes**

## Further Improvements?

- **S5FS: 0.16% of capacity**
- **FFS without block interleaving: 3.8% of capacity**
- **FFS with block interleaving: 50% of capacity**
- **What next?**

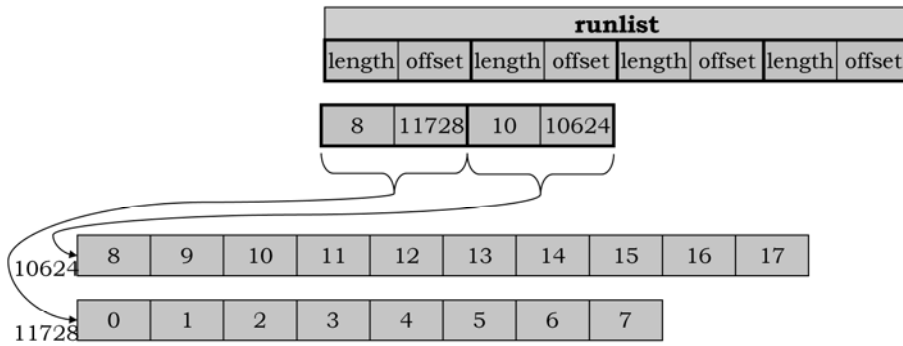
## Larger Transfer Units

- **Allocate in whole tracks or cylinders**
  - too much wasted space
- **Allocate in blocks, but group them together**
  - transfer many at once

## Block Clustering

- **Allocate space in blocks, eight at a time**
- **Linux's Ext2 (an FFS clone):**
  - allocate eight blocks at a time
  - extra space is available to other files if there is a shortage of space
- **FFS on Solaris (~1990)**
  - delay disk-space allocation until:
    - 8 blocks are ready to be written
    - or the file is closed

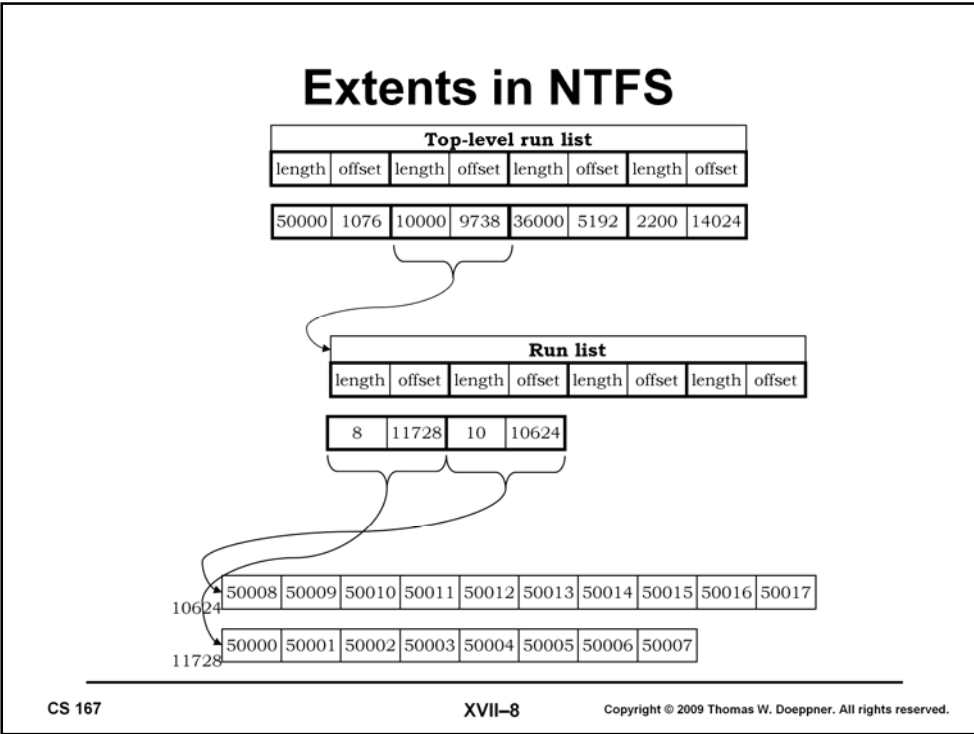
# Extents



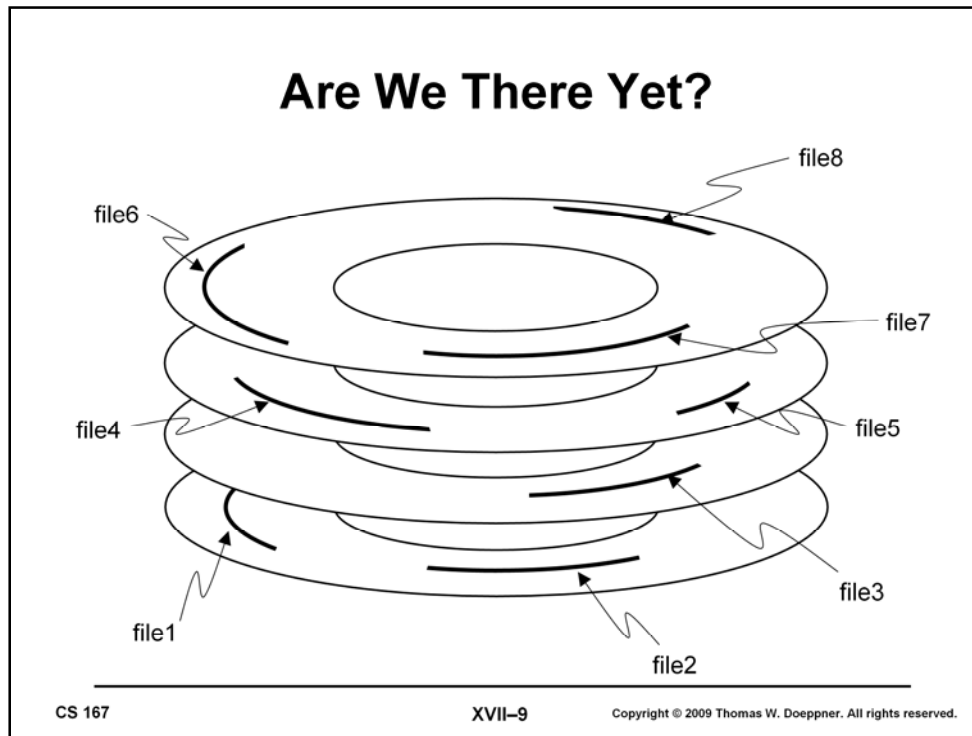
## Problems with Extents

- **Could result in highly fragmented disk space**
  - lots of small areas of free space
  - solution: use a *defragmenter*
- **Random access**
  - linear search through a long list of extents
  - solution: multiple levels

# Extents in NTFS



A two-level run list in NTFS. To find block 50011, we search the top-level run list for a pointer to the file record containing the run list that refers to this block. In this case it's pointed to by the second entry, which contains a pointer to the run list that refers to extents containing file blocks starting with 50,000 and continuing for 10,000 blocks. We show the first two references to extents in that run list. The block we're after is in the second one.

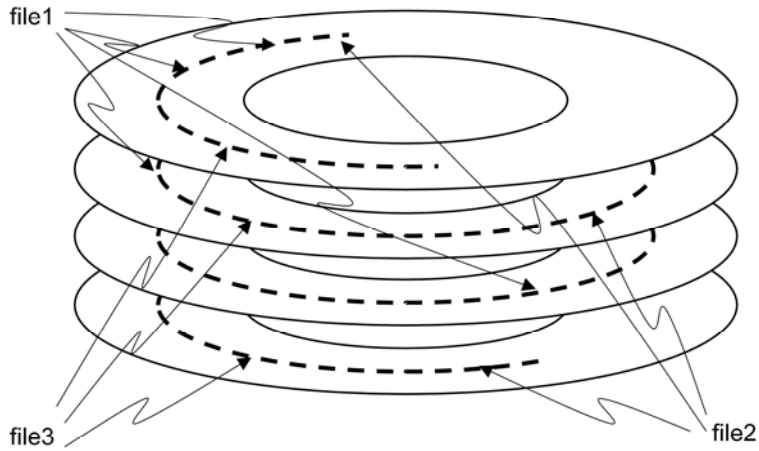


The slide shows a number of files, each stored in a single extent. This allows fast, perhaps optimal read access for each file, once the disk is positioned for the beginning of that file. Of course, files requiring multiple extents aren't read quite so quickly. However, we seem to be optimizing for (sequential) read access. Is this the right strategy?

## A Different Approach

- We have lots of primary memory
  - enough to cache all commonly used files
- Read time from disk doesn't matter
- Time for writes does matter

# Log-Structured File Systems

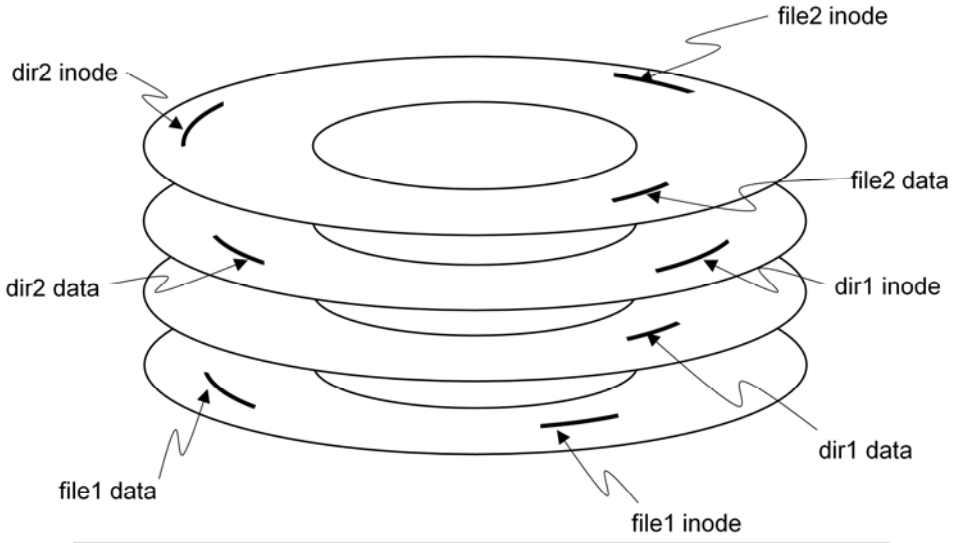


Portions of files are placed in a log (shown here as occupying most of one cylinder) in the order in which they were written.

## Example

- We create two single-block files
  - *dir1/file1*
  - *dir2/file2*
- FFS
  - allocate and initialize inode for *file1* and write it to disk
  - update *dir1* to refer to it (and update *dir1* inode)
  - write data to *file1*
    - allocate disk block
    - fill it with data and write to disk
    - update inode
  - six writes, plus six more for the other file
    - seek and rotational delays

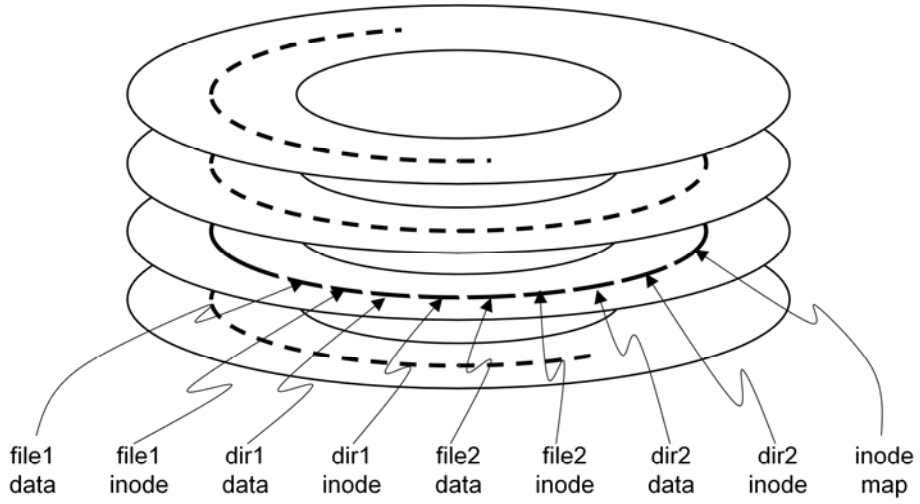
# FFS Picture



## Example (Continued)

- **Sprite (a log-structured file system)**
  - one single, long write does everything

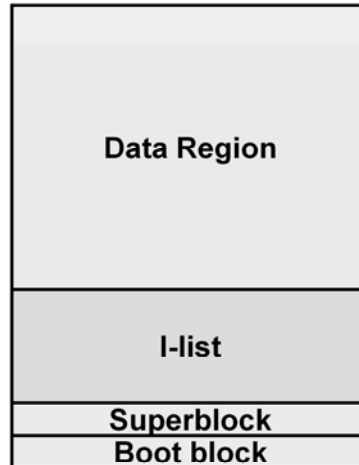
# Sprite Picture



## Some Details

- **Inode map cached in primary memory**
  - indexed by inode number
  - points to inode on disk
  - written out to disk in pieces as updated
  - checkpoint file contains locations of pieces
    - written to disk occasionally
    - two copies: current and previous
- **Commonly/Recently used inodes and other disk blocks cached in primary memory**

## S5FS Layouts



Note that the size of the I-List is fixed.



## NTFS Master File Table

<b>MFT</b>
<b>MFT Mirror</b>
<b>Log</b>
<b>Volume Info</b>
<b>Attribute Definitions</b>
<b>Root Directory</b>
<b>Free-Space Bitmap</b>
<b>Boot File</b>
<b>Bad-Cluster File</b>
<b>Quota Info</b>
<b>Expansion entries</b>
<b>User File 0</b>
<b>User File 1</b>

NTFS's Master File Table (MFT). Each entry is one kilobyte long and refers to a file. Since the MFT is a file, it has an entry in itself (the first entry). There's a copy of the MFT called the MFT mirror. There are a number of other entries for system files and metadata files, and then the entries for ordinary files.

Each entry of the MFT plays the role of an inode. But, since the MFT is a file that, like any other file, may grow and shrink dynamically, these inode equivalents are not allocated statically but are allocated dynamically.