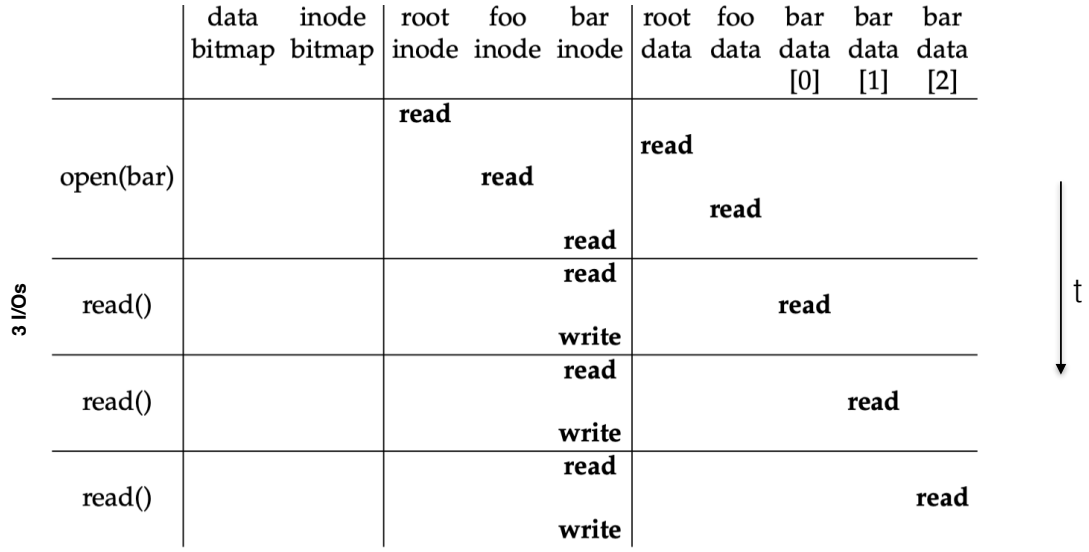
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories
- 40 - File System Implementation
- 41 - Locality and the Fast File System
- 42 - Crash Consistency
- 43 - Log-structured File Systems
- 44 - Flash-based SSD
- 45 - Data Integrity and Protection

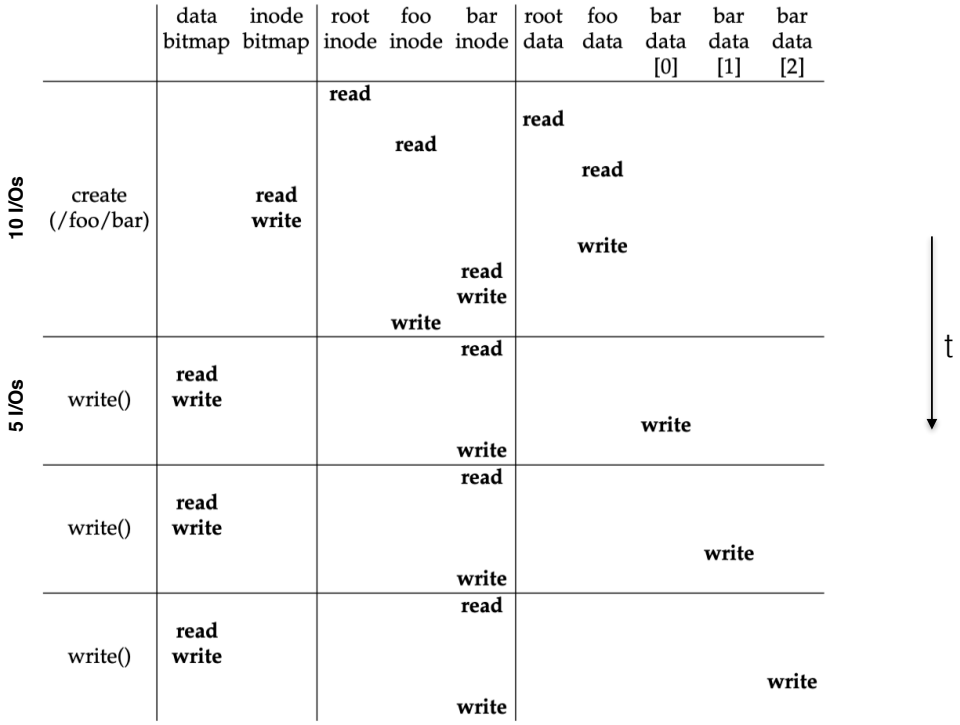
Reading a file from disk

- Issue an open("/foo/bar", O_RDONLY),
- Traverse the pathname
  - begin at the root of the file system (/)
    - root inode number often 2 (specified in superblock)
    - read in block containing inode 2.
    - use "/" pointer blocks to get "/" directory contents
  - recurse on "/foo"
  - check permissions, memory for metadata, file descriptor
- when `read()` issued
  - consult inode, find and read in first block
  - update open file table, file offset
- When file closed
  - dealloc file descriptor, logically the file may be deallocated, but not usually done here

Open and read `/foo/bar` timeline



Create `/foo/bar` timeline

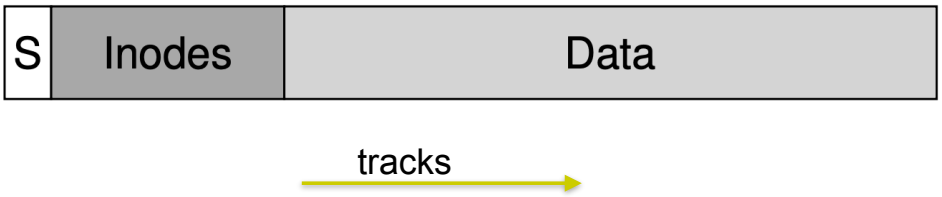


Unified Buffer Cache

- buffer cache
  - recently used pages/disk-blocks held in memory
  - writes *buffered* (delayed)
    - consecutive writes batched
    - *scheduled* more efficiently
  - dynamically partitioned (not fixed size)
- some apps (databases) ignore the cache and file system
  - call `rsync()`
  - use *direct I/O* interfaces to disk
  - write to raw disks

Locality and the Fast File System *FFS*

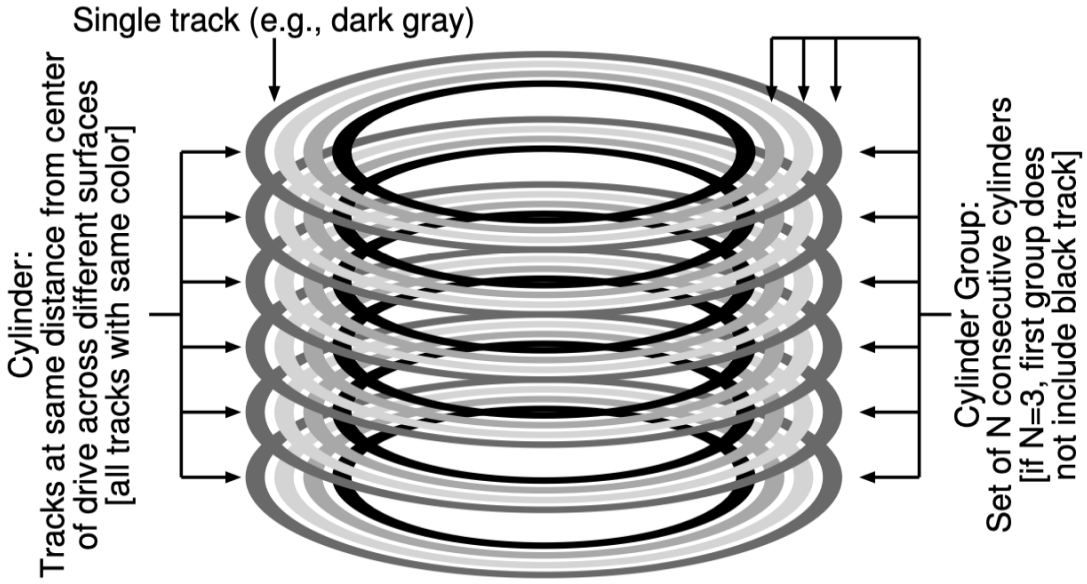
- "old" file system



  - performance starts bad, gets worse
    - fragmentation as files deleted and created
    - block size too small (slow transfers)
    - `inodes` not near data
- FFS fixed many of these problems
  - ….but we are still talking about the 1990s…

Cylinder groups *FFS*

- …. could be useful but disks do not export enough info



Block Groups *FFS*

- FFS uses *block groups*
  - disk maps onto cylinder groups
  - each has superblock, bitmaps, inodes, data

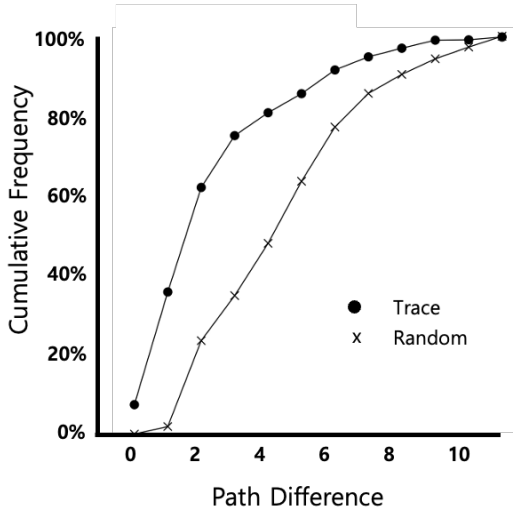
- "Keep related stuff together" - single group
  - most directories
  - file data and related inodes
  - large files have chunks sprayed across multiple groups

How Far are Accesses *FFS*

- How "**far away**" file accesses were from one another in the directory tree.

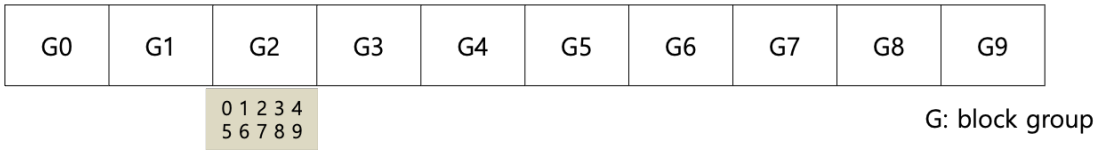
```
            proc/src/foo.c
            proc/src/bar.c
the distance of two file access is 1

            proc/src/foo.c
            proc/obj/foo.o
the distance of two file access is 2
```

- 7% of file accesses to the same file
- Nearly 40% of file accesses in the same directory
- 25% of file accesses were distance two



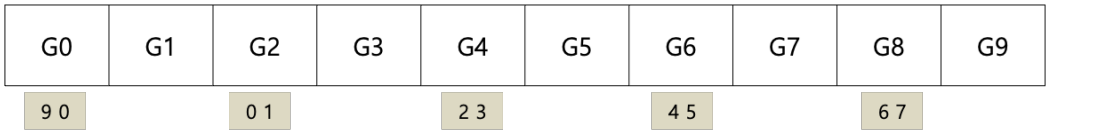
Large File Exception *FFS*

- Usual file placement policy
  - a large file might fill a group entirely
  - lose directory locality



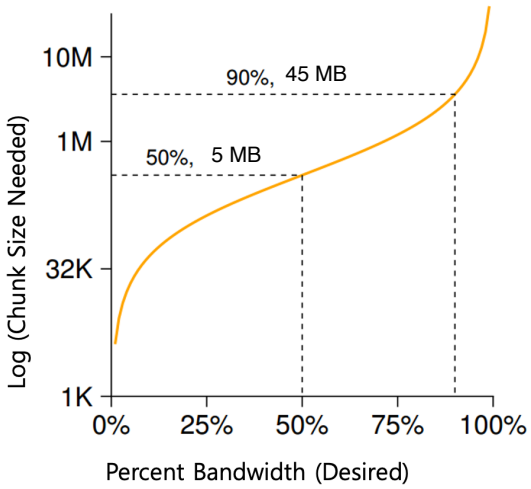
G: block group

- Fix: spread large chunks across multiple groups
  - make chunks large enough to perform well



How Large Should Chunks Be? *FFS*

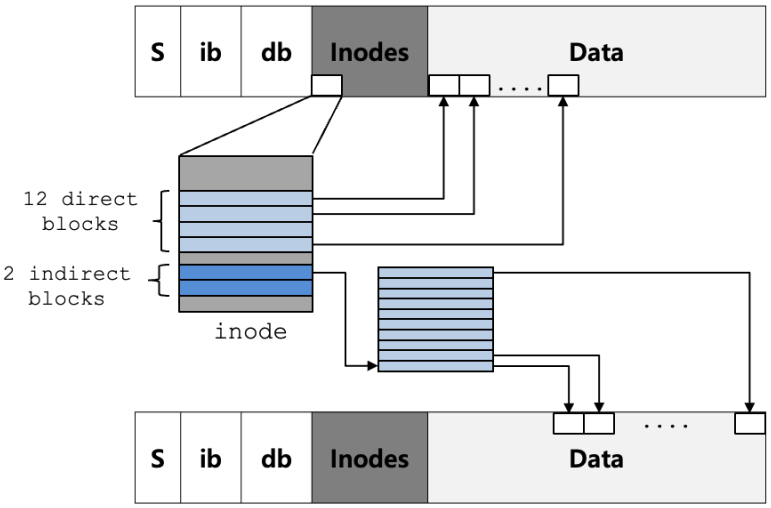
- If want 50% of peak disk performance
  - ⇒ half time seeking, half time transferring data
  - If we assume:
    - disk bandwidth 1GB/sec
    - positioning time 5 msec
    - $\frac{1GB}{\text{sec}} = \frac{1MB}{\text{msec}}$ ⇒ need $5 \times 1\text{MB} = 5\text{MB}$ chunks



- 90% peak performance w/ 45 MB chunks

iNodes *FFS*

- Use *inode* structure
  - direct links and blocks same group
  - indirect blocks, and blocks pointed to, different group
  - each 1024 blocks (4MB) in different group (4K pages, 4-byte ptr)



Errata

- more internal fragmentation
- used subblocks for space efficiency *(small disks in stone ages)*
  - copy to regular blocks when full
  - `libc` buffers, so most large files never subblock'd
- parameterization
  - blocks laid out so that OS has time to request block $i + 1$ after reading block it, *before* $i + 1$ rotates past
- track buffer
- long file names
- symbolic links