13. File Systems

Take the ORNL Cray Cluster Star

- Storage: 1 EB (10^18 bytes → 1 million TB)
- Speed: 10 TB/s output
- Size: 40 cabinets with 19 inch racks
- Cost: $50 million
- 2 systems: Lustre for distributed & ZFS for local
- Utilizes flash & disk

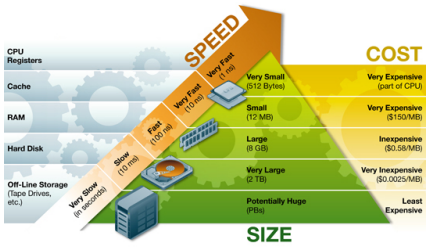
	Flash	Disk
Cost	$110/TB	$14/TB
Reliability		✓
Durability		✓
Speed	✓	

How do we judge performance?

a. throughput ~ total requests or bytes per second (read is advertised, since it's fast)
b. latency ~ delay between request and response
c. utilization ~ fraction of capacity doing useful work

But (a) and (b) are competing values!

What strategies do we have?



We exploit **locality of reference** by caching data we expect to need into higher memory:

- **spacial locality** ~ accessing address i means accessing address i ± 1 is likely
- **temporal locality** ~ accessing address i means accessing i again is likely

The file system takes advantage of this locality in 3 main ways:

1. Prefetching ~ the OS caches memory it expects to need ahead of time
2. Batching ~ the OS reads all around a piece of data on read (includes prefetching)
3. Dallying ~ the OS caches writes with the hopes that it can perform adjacent writes

But what if we need the data to be written NOW?

- sync()
  - flush all buffers to permanent storage

    - performed system wide — often a waste of effort
- fsync(fd)
    - flush all buffers for a given file descriptor & wait for completion
- fdatasync(fd)
    - flush all data buffers for a given file descriptor & wait for completion
    - fast, since metadata is the slow part (it is stored in memory)

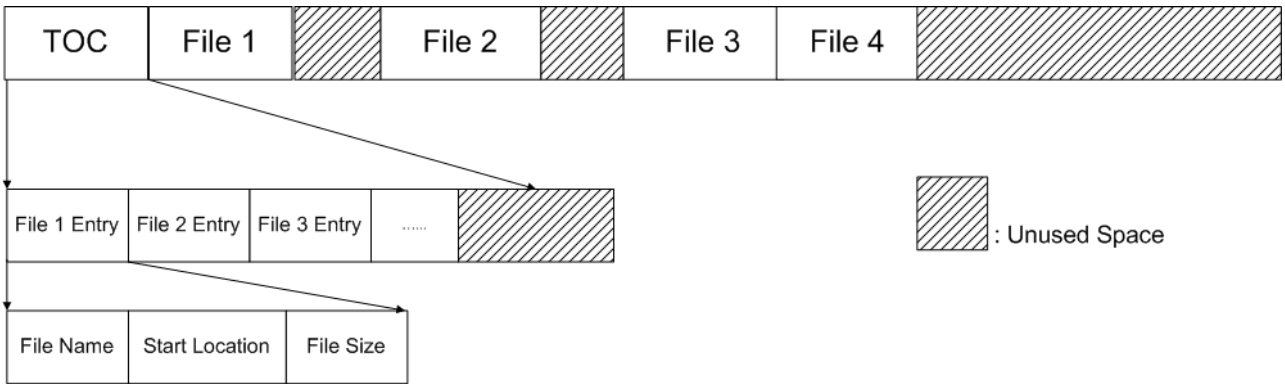
The IBM General Parallel File System uses a few other tricks for optimization:

- Striping
- Distributed Metadata
    - copies of file metadata exist to alleviate IO bottleneck
- Distributed Locking
    - Systems allow users to lock sections of file, and not the whole file
- Efficient Directory Indexing
    - As a result, GPFS uses a fancier data structure to represent files.
- File System Stays Live During Maintenance

File Systems

- a data structure for primary and secondary storage with support for searching

VERY SIMPLE FILE SYSTEM (RT-11)

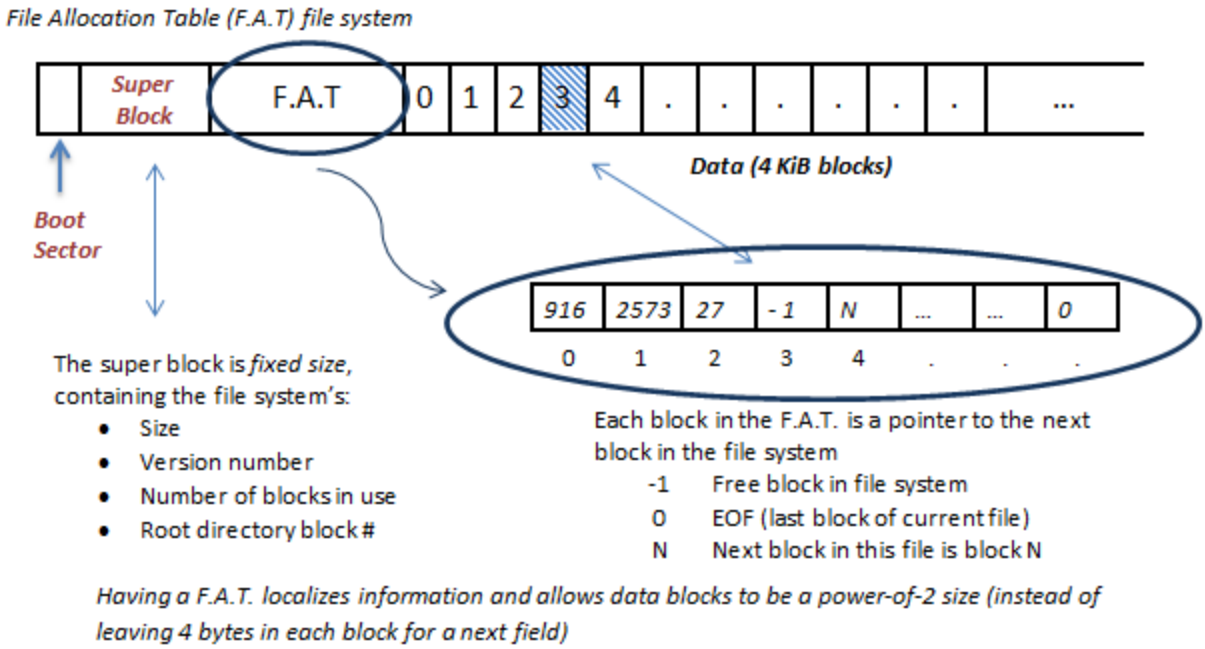


- RULES:
    i. 2mb of memory broken into 512 byte sectors
    ii. files start on sector boundaries
    iii. files are continuous (like in RT-11)
    iv. sectors will only be partially used
    v. first 10 sectors form a table of contents

    vi. file size is statically decided on creation
    vii. first byte indicates if the directory is full
    viii. files can only span a single region of free space
- PROS:
    - predictable
    - easy truncation
    - good sequential access
- CONS
    - file number limit (2^6)
    - difficult to grow files
    - only one directory — no user ones
    - no file permissions
    - fragmentation
        i. internal ~ wasted data within allocated space
        ii. external ~ scattered and therefore unusable free space

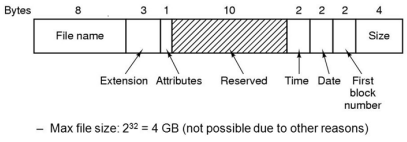
FAT File System

The VSFS was the primary file system until Bill Gates came along in the 70s and created...



- The goal: turn files into linked lists of blocks to avoid external fragmentation

- RULES:
    i. Boot block contains setup code
    ii. A super block holds system data including root
    iii. Directories form a tree rooted at the root directory
    iv. Blocks contain discrete size blocks of file data
    v. FAT contains address of the block which comes next in the file
    vi. directories contain data blocks just like files, but the form is different
        - name + attr + date/time + ptr to file + size
        - name = 0xE5 — the file is deleted
        - delete: prev directory size += size



File System Directories

- PROS:
    - no external fragmentation
    - no preallocation
    - large file count limit
- CONS:
    - internal fragmentation is not fixed
        - can be addressed by periodic **defragmentation** of files
            - this is costly in time and risky in terms of corruption
    - moving a file to another directory can lose both copies!
        - this is forbidden
    - no random access
        - We would need to walk through the whole list to find a byte offset

Random access was important to Unix, so Linux made this easy