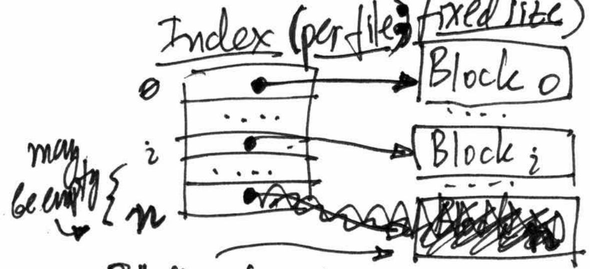
COMSW4118-22-1 11/28/2017

HW6 = no demos!

- **Storage / Disk** → Primary abstraction is Files, supported by a File System – key issues are performance & reliability.

▶ Allocation is also an important issue:

- Linked approach – both performance & reliability are poor
- FAT – replication gives good reliability, caching gives good performance
- Indexed is what Linux uses; AKA index node or inode



- Each file has an index table w/ each index entry pointing to a block entry of the file. (size of index is fixed)
- Index is crucial for reliability, but it is small (can be cached) and per file (can't lose all files)
- Indexes are stored on disk @ special location.

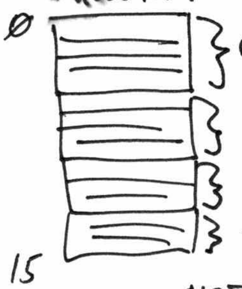
→ Block = fixed size unit of storage on the disk (often = frame) ← typically blocks don't contain multiple files.

→ Real file systems are not strictly trees, since links b/n files are cycles.

→ symlink = removing the link leaves the file; hard link = removing link removes file.

→ inode size is fixed (typically); block size is fixed too! typically, an inode has around 15 entries; block size that is large will result in wasted space (most files will be much smaller & can't share blocks)

→ So a hierarchy is used with mixed direct and indirect index entries: Index: (ex. 15 entries) and say a disk address is 4 bytes and a block is = disk page = 4K = 1,000 addresses



- direct (11 entries) → can store 11 blocks ≈ 44 KB
- single-indirect (2 entries) → can store 2 × 1000 = 2000 blocks ≈ 8 MB
- double-indirect (1 entry) → can store 1000^2 blocks ≈ 4GB
- triple-indirect (1 entry) → can store 1000^3 blocks ≈ 4TB

NOTE: The inode is NOT a disk block, it's a separate (much smaller structure) and inodes are pre-allocated (in certain disk blocks). The indirect blocks are special blocks that are interpreted as arrays of (indirect or direct) addresses.

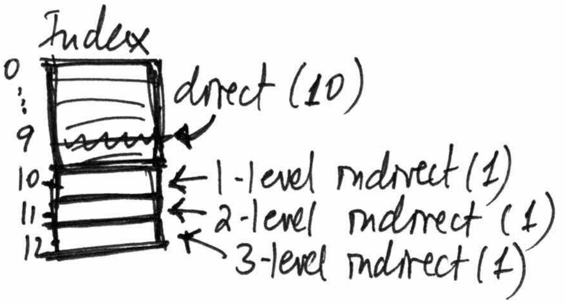
Ex1: UNIX System V (S5FS): inode = 64 bytes, Block = 512 bytes ← everything gets setup when you configure a disk. Inode list is fixed, lots are allocated though, since they are small.

Boot	superblock	inode List	data blocks....
↑ Boot block	↑ metadata for whole filesystem	↑ list of inodes	↑ both data & indirect blocks

COMSW4118-22-2

11/28/2017

Ex.1 (cont.)



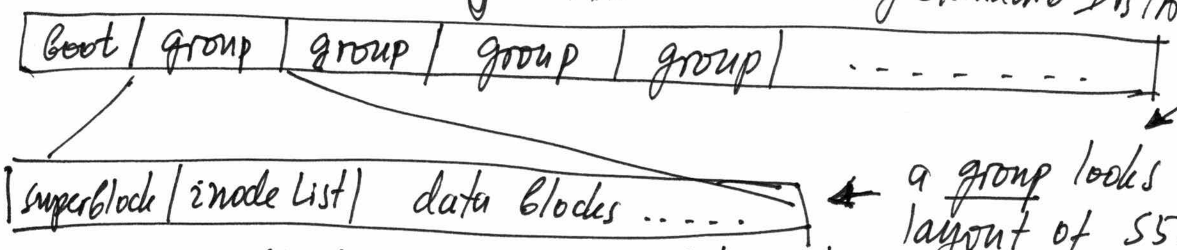
} 13 entries on each i-node.
But inode is 64 bytes?
Addresses were 3 bytes, so 39 bytes
+ metadata (size, access time, owner, permissions)

But, this implementation of the index scheme is not used widely anymore (the index scheme is still used but w/ different implementation).

Performance - inodes & data blocks are separated, need to go back and forth

Reliability - FS metadata and inodes are contiguous and single copy, if problem then the whole FS or most of the files will be lost.

Ex.2: FFS (Fast File system) in BSD (Berkeley Standard Distribution)

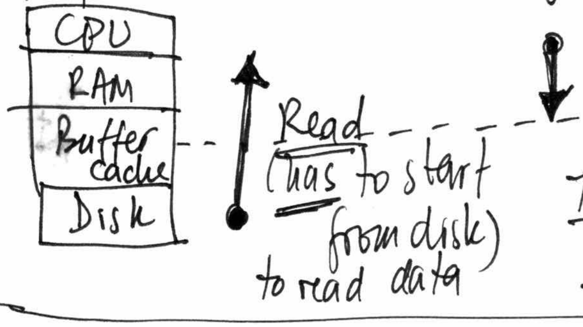


If you run out of space (inode or data) in a group, you start using other groups.

a group looks like the original layout of S5FS, But:

1. Superblock is replicated = better reliability
2. Locality is increased, since inodes & datablocks don't span the whole disk

⇒ both performance and reliability are improved (about 5 (write) to 8 (read) times better!)



Read (has to start from disk) to read data

Write (does not have to necessarily read disk) this can be flushed later (sync)

That's why disk improvements have more of an impact to the read!

FFS

- This became the Linux ext2. → ext3 → ext4

Ex.3: Journaling (logging) added (w/o journaling, file system checks done by fsck on ext2 take a long time.)

- In addition to storing the data normally, you add a journal / log
- The log / journal is linearly ordered by time = never go back, just keep adding changes
- You can recover the entire file system from the log and no need to look at the entire file system, just the log.