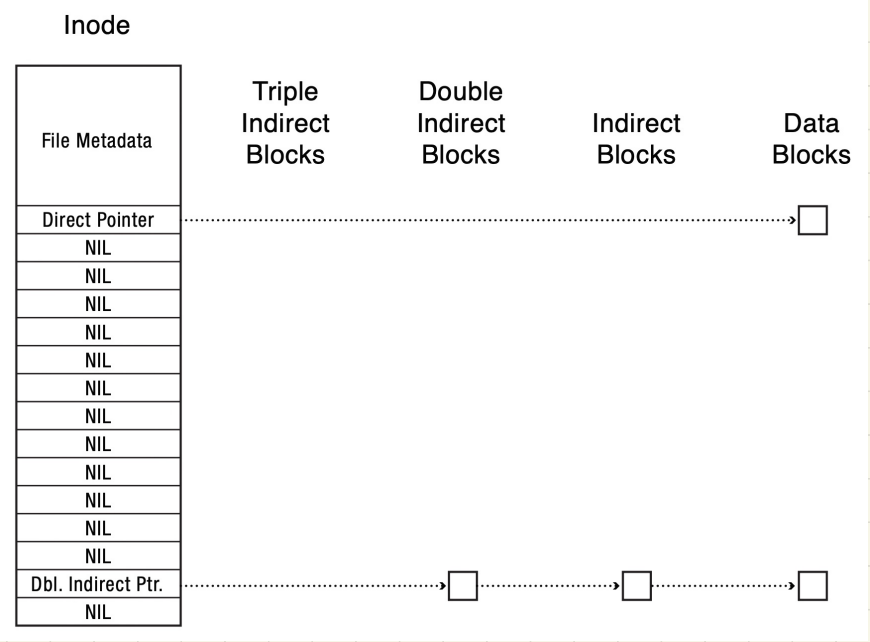
11/20/23 FFS & Crash Consistency

FFS:
· designed for disk, block size: 4KB(8 sectors)
· data layout = direct, indirect, double indirect, triple indirect

lseek to large offset,
then do a write
=> file with gap

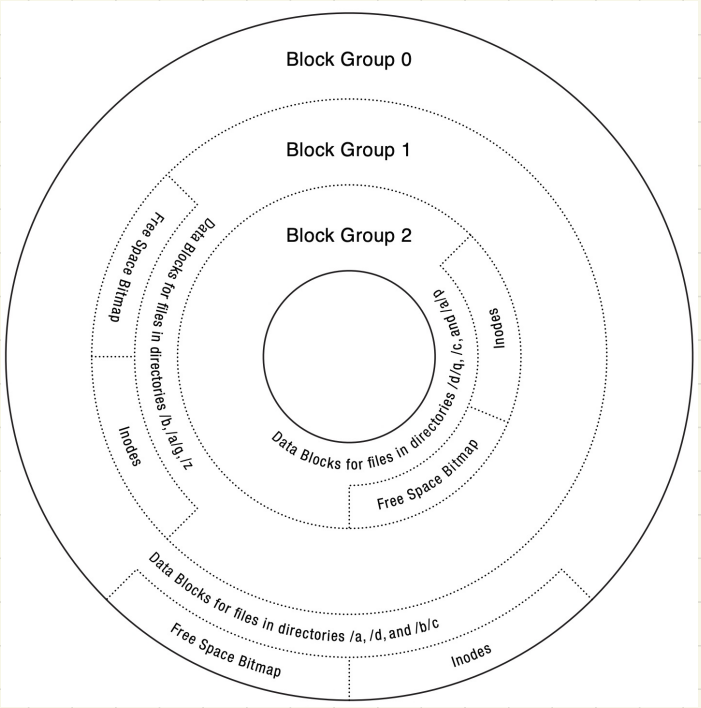


Locality Heuristic

-> sequential access, access tracks close together
-> group tracks into block groups
-> place related things into the same block group
    -> exception for large files, why?
-> place unrelated things into different groups

Related things
    file metadata & data
    file within the same directory

Unrelated things
    different directories



Each block group has its own data bitmap, inode bitmap, and inode table

Crash Consistency

-> block bitmap, inode table, data blocks
-> append that causes a new data block to be allocated
  -> block bitmap updated to reflect the newly allocated data block
  -> inode table (no change in usage status)
  -> inode for file updated with new size, modified time, change in data layout
  -> data block filled with appended data

3 disk writes, disk controller can reorder concurrent requests

-> while we wait for the disk writes to happen, the computer may crash at any point
-> what if only the block bitmap is written to disk?
    block leaks! not pointed to by any inode but is seen as used
-> what if only the inode is written to the disk?
    file may see other file's data, block bitmap may allocate it to someone else
-> what if only the data block is written to disk?
    all good! no inconsistency in the file system metadata

How to deal with inconsistency?

-> resolve inconsistency: fsck
-> avoid inconsistency
  -> we want every operation's disk updates to be atomic
  -> but disk only promises atomicity at a sector level
  -> build abstraction for atomic updates to a group of sectors
    => transaction!

Transaction

tx_begin
update 1
update 2
update 3
tx_commit

defines atomic updates

Journaling / Write ahead Logging

no change to fs on disk yet

-> reserve space for log
-> for every operation, write transaction to log first
-> persist the log
-> apply updates to their actual location

* When is the log / filesys persisted?
  -> upon fsync or sync system call (Posix)

in xk, we don't have sync / fsync
changes should be persisted after
every filesys op.

Cost of Journaling

-> perform updates twice, can be expensive
-> can we reduce what we journal?
    -> data journaling
        log changes to metadata and data, data might be large
    -> metadata journaling
        only log metadata
        what happens to the data?
        write data changes to their actual location (crash can result in partial data update)
        journal metadata once all data blocks are persisted

ext4 data mode

ext4 ordered mode (default)