Inode:

Technically, in Unix, the term inode refers to an unnamed file in the file system, but the precise meaning can be one of three, depending on context.

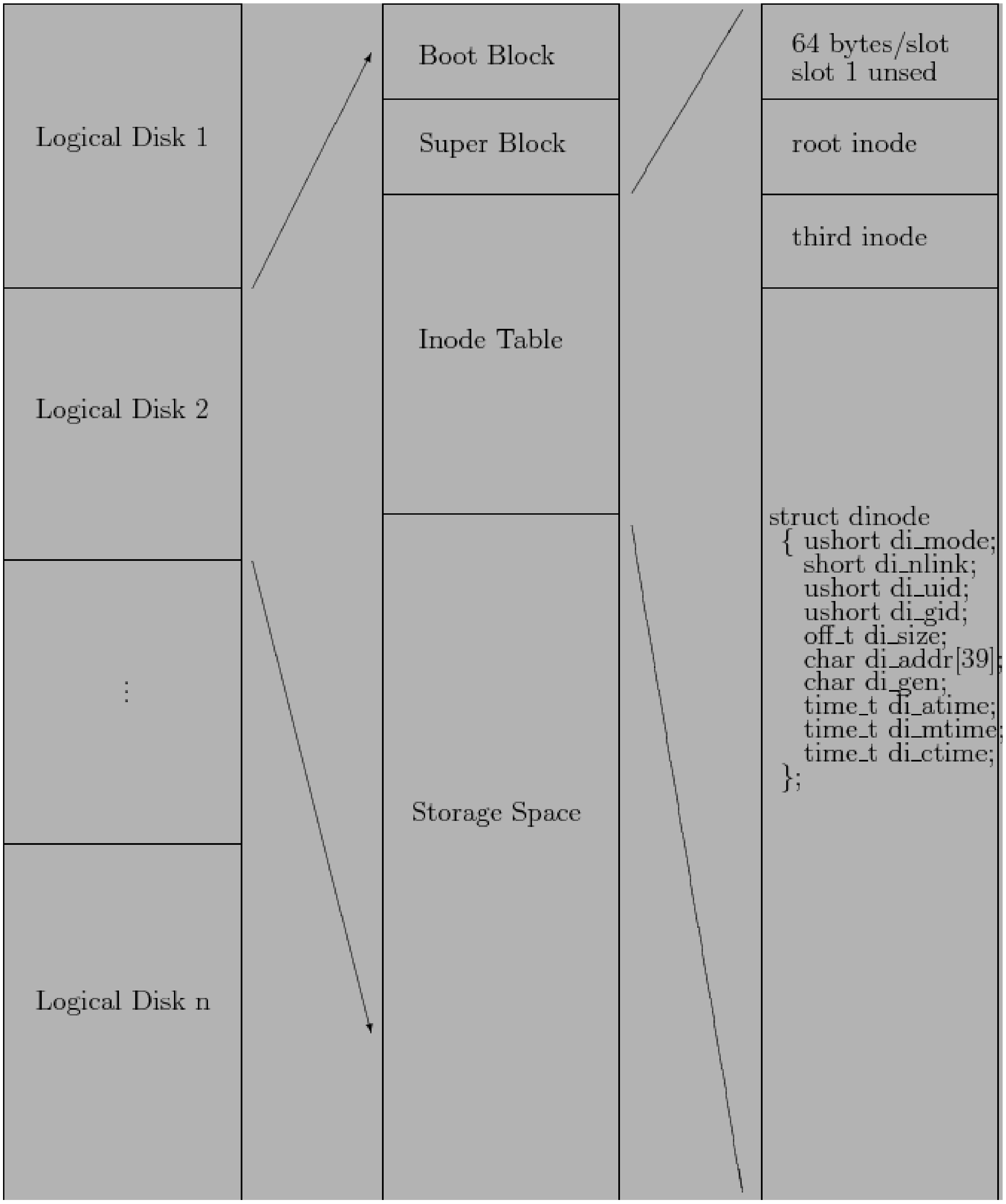
- First, there is the on-disk data structure, which contains metadata about the inode, like its size and the list of blocks storing its data.
- Second, there is the in-kernel data structure, which contains a copy of the on-disk structure but adds extra metadata needed within the kernel.
- Third, there is the concept of an inode as the whole unnamed file, including not just the header but also its content, the sequence of bytes in the data blocks.

Filesystems internally refer to files and directories via inodes. Inodes are unique identifiers of the entities stored in a filesystem. Whenever an application has to operate on a file/directory (read/modify), the filesystem maps that file/directory to the right inode and start referring to that inode whenever an operation has to be performed on the file/directory.

Each time we create a file in a directory, the system simply allocates a free I-number from the file system and uses an empty slot in the correspondent directory to record this I-number and the name of the file we created. When we issue a command to delete a file, the system simply replaces the file's I-number by 0, and the slot is still occupied by its file name, although we are no longer able to access that file anymore. Therefore, if we frequently create and delete files within a directory, the (file) size of the directory becomes bigger and bigger and the I-numbers are no longer consecutive, this will slow down the file searching operation (and hence, the system performance, too) considerably. This is one of the main reasons why the system performance is gradually deteriorating. We can prevent this from happening by grouping the files which are changing constantly in a fixed file system, say /tmp, and periodically repairing the file system. The following diagram depicts the directory contents of some ext2 directory.

```
/* include/linux/dirent.h */
struct dirent {
        long            d_ino;
        __kernel_off_t  d_off;
        unsigned short  d_reclen;
        char            d_name[256]; /* We must not include limits.h! */
};
```

Harddisk layout:



```
struct dinode
 { ushort di_mode;
   short di_nlink;
   ushort di_uid;
   ushort di_gid;
   off_t di_size;
   char di_addr[39];
   char di_gen;
   time_t di_atime;
   time_t di_mtime;
   time_t di_ctime;
 };
```

Inode structure:

Inode metadata is stored in an inode structure, and all the inode structures for the file system are packed into a separate section of disk called the inode blocks. Every inode structure is the same size, so it is easy, given a number n, to find the nth inode structure on the disk. In fact, this number n, called the inode number or i-number, is how inodes are identified in the implementation.
Each time we create a file in a directory, the system simply allocates a free I-number from the file system and uses an empty slot in the correspondent directory to record this I-number and the name of the file we created. When we issue a command to delete a file, the system simply replaces the file's I-number by 0, and the slot is still occupied by its file name, although we are no longer able to access that file anymore.
The administrative information of the file, such as owner, permissions, size, times, etc., is stored in the inode structure of the file. All of the file system's inodes are collected together to form an inode table. Each file system occupies a logical disk. Starting from the second block of a logical disk, the kernel stores the inode table of the file system in a consecutive disk blocks. Each inode, an entry in the inode table, is a data structure which the system uses to store the following information about a file:
Type of file (ordinary, directory or special file).

1. Access permissions for the file owner, the owner's group members and others (i.e. the general public).
2. Number of links.
3. File owner's user and group Ids.
4. File size in bytes.
5. The disk addresses of the data blocks where the contents of the file are actually stored.
6. Time of last access (read or executed), time of last modification (i.e. written) and time which the inode itself was last changed.

Fields contained in inode structure:

```
struct dinode
{ ushort  di_mode;     /* mode and type of file   */
  short   di_nlink;    /* number of links to file */
  ushort  di_uid;      /* owner's user id         */
  ushort  di_gid;      /* owner's group id        */
  off_t   di_size;     /* number of bytes in file */
  char    di_addr[39]; /* disk block addresses    */
  char    di_gen;      /* file generation number  */
  time_t  di_atime;    /* time last accessed      */
  time_t  di_mtime;    /* time last modified      */
  time_t  di_ctime;    /* time created            */
};
```

<u>Structure for inode table:</u>

```
struct inode_table{
        pthread_mutext_t  lock;
        size_t            hashsize;  /* bucket size of inode hash */
        char              name;    /*name of the inode table*/
        inode_t           *root;   /*root directory inode*/
        xlator_t          xl;    /*maintains th inode table*/
        uint32_t          lru_limit;  /*maximum LRU cache size*/
        struct list_head  *inode_hash;   /*buckets for inode hash table*/
        struct list_head  *name_hash;
        struct list_head  active;  /*list of inode currently active*/
        uint32_t          active_size; /*count of inodes in active list*/
        struct list_head  lru;    /*list of inode currently used*/
        uint32_t          lru_size;  /*count of inodes in lru list*/
        struct list_head  purge;   /*list of inodes to be purged soon*/
        uint32_t          purge_size;  /*count of inodes in purge list*/
        struct_mem_pool   inode_pool; /memory pool for inodes
        struct_mem_pool   *dentry_pool;
        struct_mem_pool   *fd_mem_pool;
        int               ctxcount; /*number of slots in inode->ctx*/
};
```

Life cycle of inode table:

*inode_table_new (size_t lru_limit, xlator_t *xl) :*

This is a function which allocates a new inode table. Usually the top xlators in the graph such as protocol/server (for bricks), fuse and nfs (for fuse and nfs mounts) and libgfapi do inode managements. Hence they are the ones which will allocate a new inode table by calling the above function. Each xlator graph in glusterfs maintains an inode table. So in fuse clients, whenever there is a graph change due to add brick/remove brick or addition/removal of some other xlators, a new graph is created which creates a new inode table. Thus an allocated inode table is destroyed only when the filesystem daemon is killed or unmounted.
A new inode is created whenever a new file/directory/symlink is created OR a successful lookup of an existing entry is done. The xlators which does inode

management (as of now protocol/server, fuse, nfs, gfapi) will perform inode_link operation upon successful lookup or successful creation of a new entry.

inode_link (inode_t inode, inode_t parent, const char name, struct iattbuf):

inode_link actually adds the inode to the inode table (to be precise it adds the inode to the hash table maintained by the inode table. The hash value is calculated based on the gfid). Copies the gfid to the inode (the gfid is present in the iatt structure). Creates a dentry with the new name.
A inode is removed from the inode table and eventually destroyed when unlink or rmdir operation is performed on a file/directory, or the the lru limit of the inode table has been exceeded.

The on-disk data structures will have the general layout:

boot	superblock	data map	inode blocks	data blocks

Boot Block

On many architectures, the first block of a bootable disk has to contain the bootloader executable.

Superblock

The superblock has the master information about the filesystem. The block also contains the inode map after the filesystem meta-data. The superblock keeps file system-level information, things like how many inodes are on disk, where they're located, and the like. the superblock record has the following format:

```
struct cse451_super_block {
        __u16 s_nNumInodes;             // inode map is tail of superblock
        __u16 s_nDataMapStart;          // block # of first data map block
        __u32 s_nDataMapBlocks;         // data map size, in blocks
        __u32 s_nInodeStart;            // block # of first inode block
        __u32 s_nNumInodeBlocks;        // number of blocks of inodes
        __u32 s_nDataBlocksStart;       // block # of first data block
        __u32 s_nDataBlocks;            // number of blocks of data

        __u32 s_nBusyInodes;            // number of inodes in use
        __u16 s_magic;                  // magic number
        char  s_imap[0];                // name for inode map
};
```

Following the superblock record (but on the same disk block) is the inode map. The inode map tracks which inodes are currently in use.

Data Map

The data map tracks which data blocks are currently in use. Like the inode map, it is a bit array.

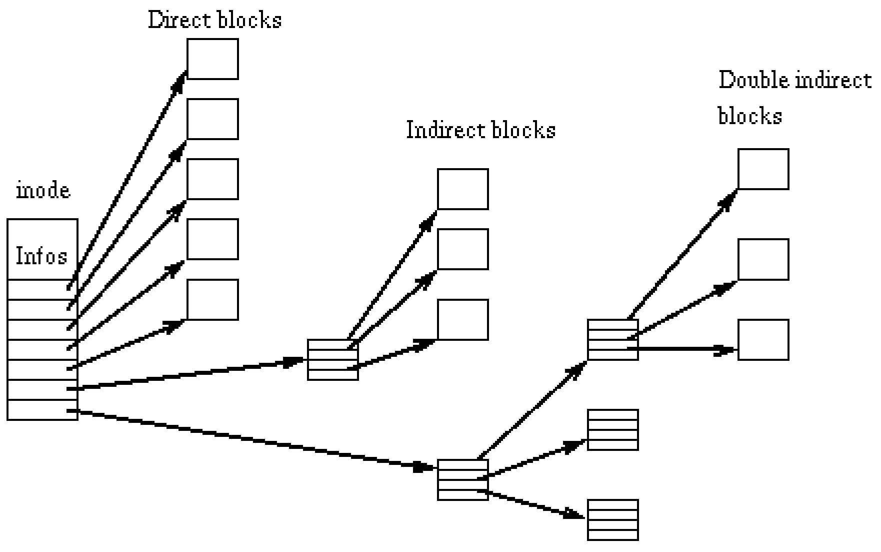
```
struct cse451_inode {

        __u16 i_mode;
        __u16 i_nlinks;
        __u16 i_uid;
        __u16 i_gid;
        __u32 i_filesize;
        __u32 i_datablocks[CSE451_NUMDATAPTRS];
};
```

The on-disk data structures used to locate files. The major components are the superblock, an array of inodes, an array of data blocks, a definition of what a directory file looks like internally, and two maps, one for the inodes and one for the data blocks.

Each inode keeps file meta-data and an index that can be used to locate the disk blocks containing the file contents. Each directory entry contains a fixed-length name field and an inode number. The two maps (inode and data block) are simply bit strings. There is one bit for each inode (data block). If bit n is 0 it means inode (data block) n is free; otherwise, it's in use.

The **inode pointer structure** is a structure adopted by the inode of a file in the Unix File System (UFS) to list the addresses of a file's data blocks.


Direct blocks
Indirect blocks
Double indirect blocks
inode
Infos