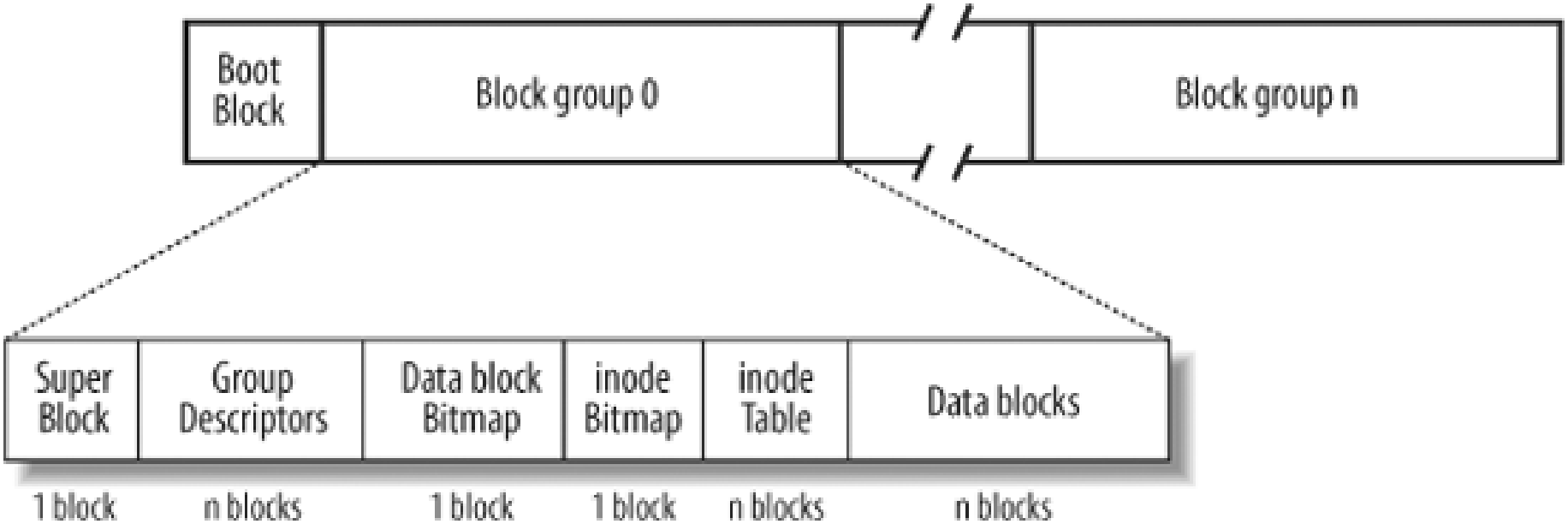
The Ext2 and Ext3 Filesystems

- The first versions of Linux were based on the MINIX filesystem
- The ***Extended Filesystem (Ext FS)*** was introduced as Linux matured, but offered unsatisfactory performance
- The ***Second Extended Filesystem (Ext2)*** was introduced in 1994 (Ext3 in 2001, Ext4 in 2008)
  - It is quite efficient and robust and is, together with its offspring Ext3, the most widely used Linux filesystem

General Characteristics of Ext2

- The following features contribute to the efficiency of Ext2:
  - When creating an Ext2 filesystem, the system administrator may choose the optimal block size (from 1,024 to 4,096 bytes)
  - When creating an Ext2 filesystem, the system administrator may choose how many inodes to allow for a partition of a given size
  - The filesystem partitions disk blocks into groups (often called cylinder groups)
    - Groups includes data blocks and inodes stored in adjacent tracks
  - The filesystem ***preallocates*** disk data blocks to regular files before they are actually used (for contiguous block allocation)
  - Fast symbolic links are supported
    - If the symbolic link represents a short pathname (at most 60 characters), it can be stored in the inode

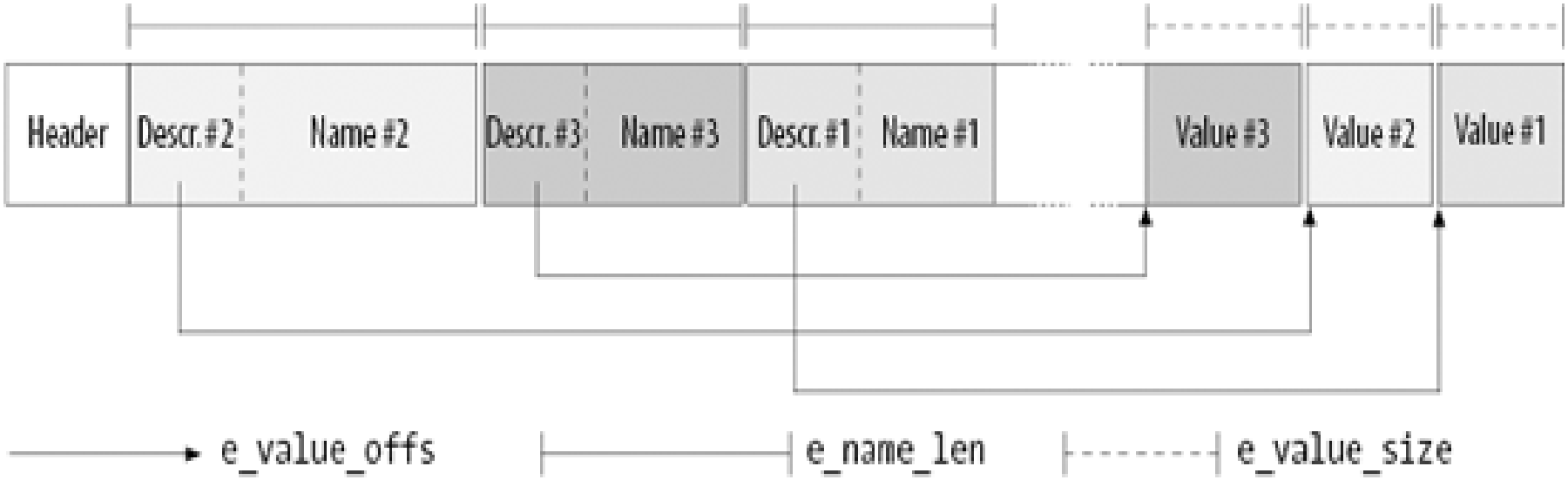
Layouts of an Ext2 partition and of an Ext2 block group



Extended Attributes of an Inode

- Inodes are 128 bytes long with virtually every byte used for some file attribute
  - To accommodate extended attributes (such as ACLs) Linux includes a single inode field called **i_file_acl** that points to a data block with extended attributes

Layout of a block containing extended attributes



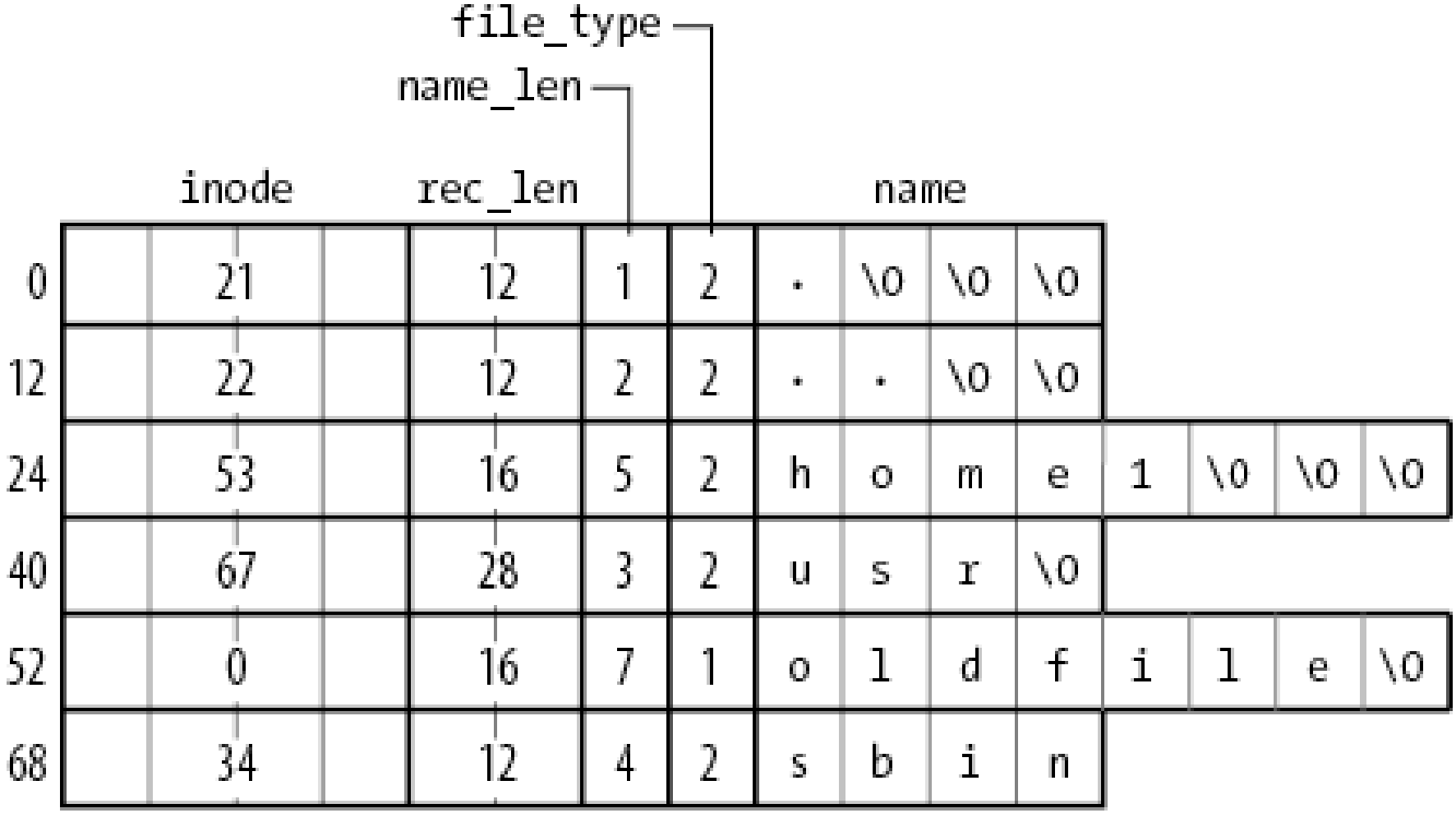
Ext2 file types

File_type	Description
1	Regular file
2	Directory
3	Character device
4	Block device
5	Named pipe
6	Socket
7	Symbolic link

The fields of an Ext2 directory entry

Type	Field	Description
_ _le32	inode	Inode number
_ _le16	rec_len	Directory entry length
_ _u8	name_len	Filename length
_ _u8	file_type	File type
char [EXT2_NAME_LEN]	name	Filename

An example of the Ext2 directory



VFS images of Ext2 data structures

Type	Disk data structure	Memory data structure	Caching mode
Superblock	ext2_super_block	ext2_sb_info	Always cached
Group descriptor	ext2_group_desc	ext2_group_desc	Always cached
Block bitmap	Bit array in block	Bit array in buffer	Dynamic
inode bitmap	Bit array in block	Bit array in buffer	Dynamic
inode	ext2_inode	ext2_inode_info	Dynamic
Data block	Array of bytes	VFS buffer	Dynamic
Free inode	ext2_inode	None	Never
Free block	Array of bytes	None	Never

Ext2 Filesystem Format

- Ext2 filesystems are created by the ***mke2fs*** utility program; it assumes the following default options:
  - Block size: 1,024 bytes (default value for a small filesystem)
  - Fragment size: block size (block fragmentation is not implemented)
  - Number of allocated inodes: 1 inode for each 8,192 bytes
  - Percentage of reserved blocks: 5 percent

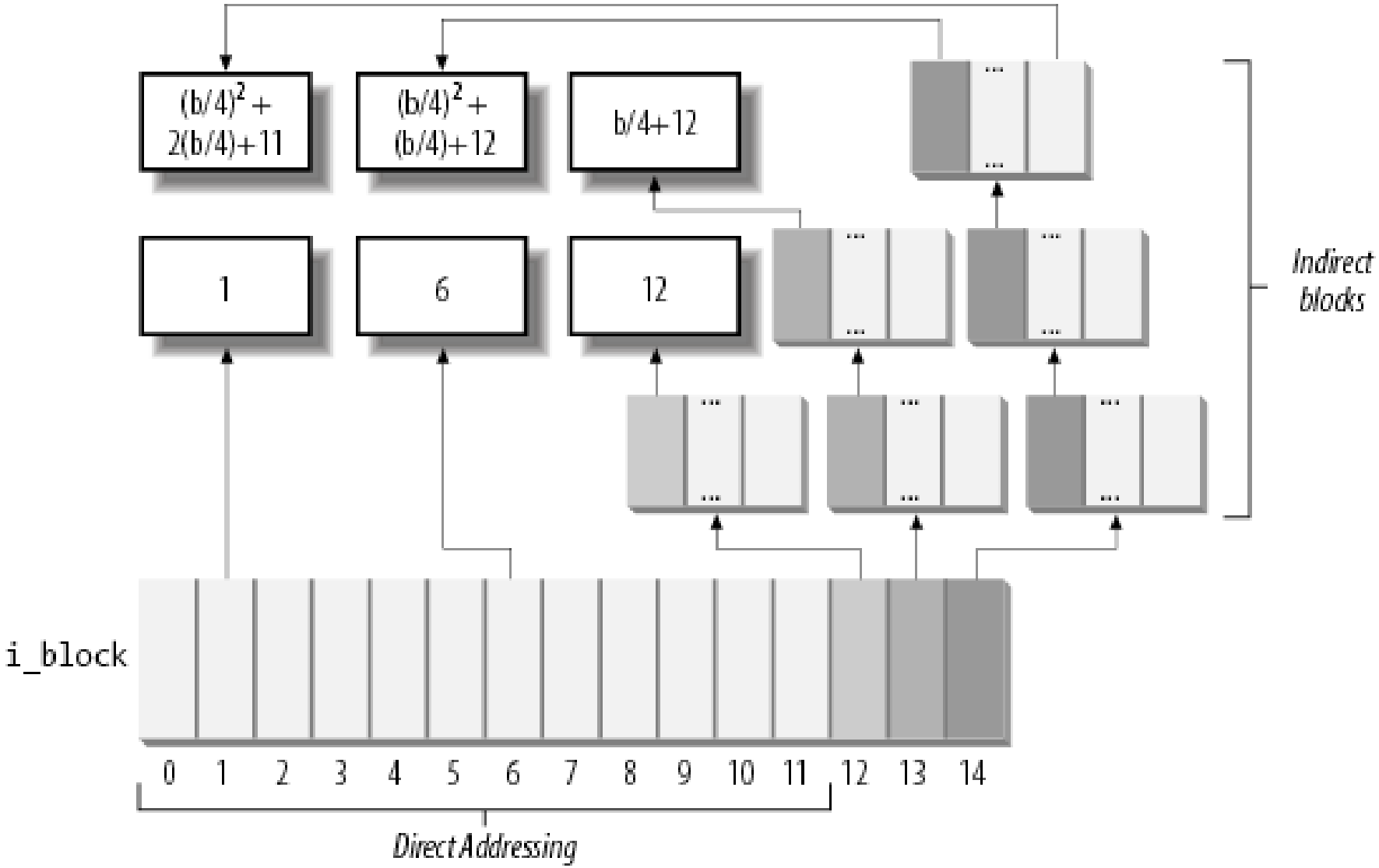
Managing Ext2 Disk Space

- The ***ext2_new_inode()*** function creates an Ext2 disk inode, returning the address of the corresponding inode object (or NULL, in case of failure)
  - In the case of a directory, it will try to allocate from a group which is more empty than average
  - In the case of other file types it will try and allocate from the group that the parent directory is in
- The ***ext2_free_inode()*** function deletes a disk inode, which is identified by an inode object
  - The kernel should invoke the function after a series of cleanup operations involving internal data structures and the data in the file itself

Data Blocks Addressing

- The first 12 components yield the logical block numbers corresponding to the first 12 blocks of the object
- The component at index 12 contains the logical block number of a block, called a first-level ***indirect block***, that is filled with an array of logical data block numbers
- The component at index 13 contains the logical block number of a second-level indirect block containing a second-order array of logical block numbers, each of which points to a block filled with logical data block numbers
- The component at index 14 uses triple indirection

Data structures used to address the file's data blocks



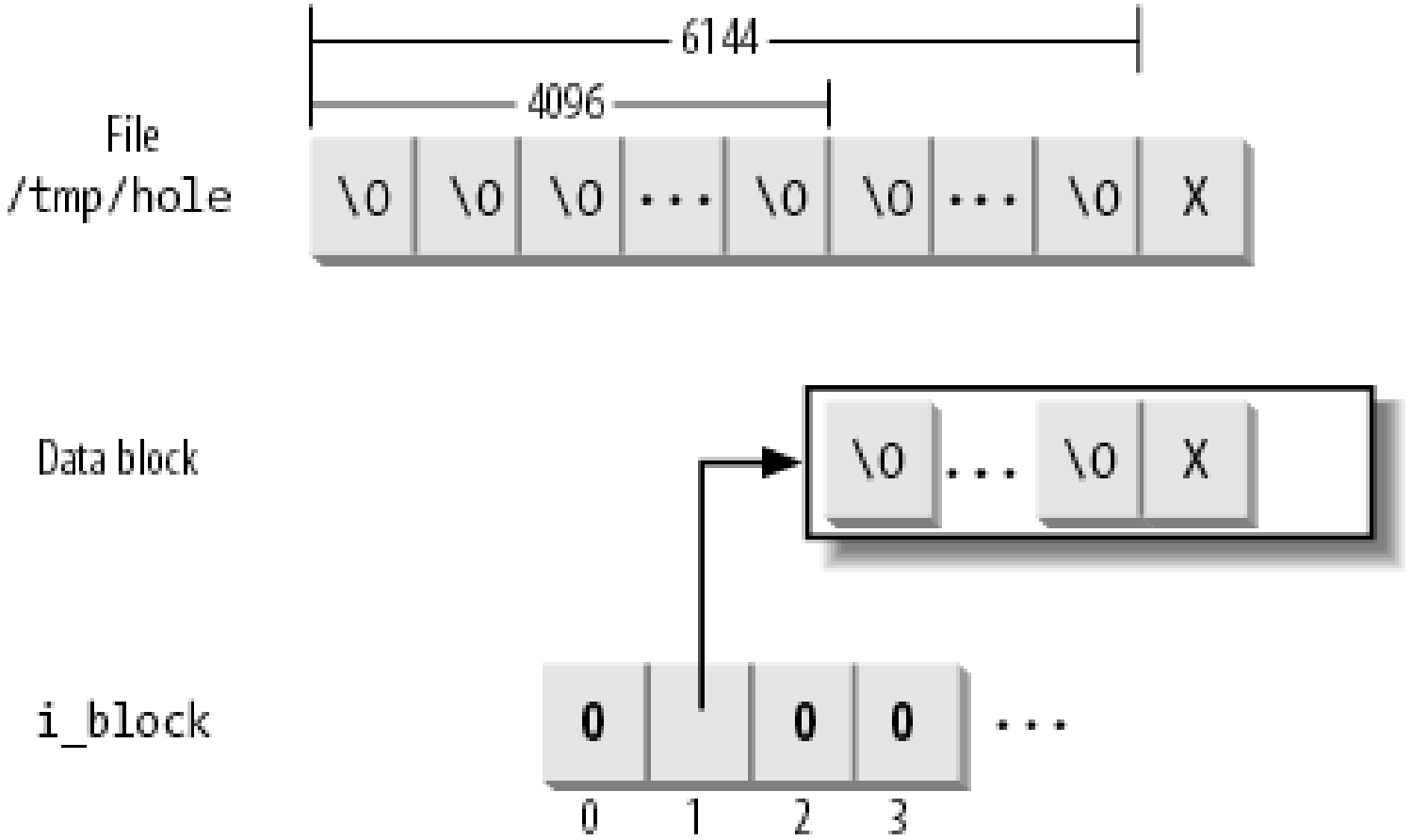
File-size upper limits for data block addressing

Block size	Direct	1-Indirect	2-Indirect	3-Indirect
1,024	12 KB	268 KB	64.26 MB	16.06 GB
2,048	24 KB	1.02 MB	513.02 MB	256.5 GB
4,096	48 KB	4.04 MB	4 GB	~ 4 TB

A file with an initial hole

Sparse files using NULL pointers

Assume a new file has been created and
an ***lseek(6144)*** was done before a ***write()*** of a single byte ***'X'***



Allocating a Data Block

- When the kernel has to locate a block holding (or to hold) data for an Ext2 regular file, it invokes the ***ext2_get_block()*** function
- The ***ext2_get_block( )*** function handles the data structures already described, and when necessary, invokes the ***ext2_alloc_block()*** function to actually search for a free block
  - If necessary, the function also allocates the blocks used for indirect addressing
  - To reduce file fragmentation, the Ext2 filesystem tries to get a new block for a file near the last block allocated for the file

The Ext3 Filesystem

- The enhanced filesystem that has evolved from Ext2, is named ***Ext3***
- The new filesystem has been designed with two simple concepts in mind:
  - To be a journaling filesystem for fast failure recovery
  - To be, as much as possible, compatible with the old Ext2 filesystem

The Ext3 Journaling Filesystem

- The idea behind Ext3 journaling is to perform each high-level change to the filesystem in two steps
  - First, a copy of the blocks to be written is stored in the journal
  - Then, when the I/O data transfer to the journal is completed (in short, data is ***committed to the journal***), the blocks are written in the filesystem
  - When the I/O data transfer to the filesystem terminates (data is ***committed to the filesystem***), the copies of the blocks in the journal are discarded

Recovery From System Failure

- While recovering after a system failure, the ***e2fsck*** program determines the following cases:
  - ***The system failure occurred before a commit to the journal***
    - Either the copies of the blocks relative to the high-level change are missing from the journal or they are incomplete; in both cases, ***e2fsck*** ignores them
  - ***The system failure occurred after a commit to the journal***
    - The copies of the blocks are valid, and ***e2fsck*** writes them into the filesystem
  - In the first case, the high-level change to the filesystem is lost, but the filesystem state is still consistent
  - In the second case, *e2fsck* applies the whole high-level change, thus fixing every inconsistency due to unfinished I/O data transfers into the filesystem

Journaling Options

- The Ext3 filesystem can be configured to log the operations affecting both **metadata** and **data blocks**
- The system administrator decides what must be logged:
  - ***Journal***
    - All data and metadata changes are logged into the journal
  - ***Ordered***
    - Only changes to filesystem metadata are logged into the journal
    - The Ext3 filesystem groups metadata and related data blocks so that data blocks are written to disk ***before*** the metadata
    - This way, the chance to have data corruption inside the files is reduced; for instance, each write access that enlarges a file is guaranteed to be fully protected by the journal
    - This is the default Ext3 journaling mode
  - ***Writeback***
    - Only changes to filesystem metadata are logged
    - This is the method of other journaling filesystems and is the fastest