Ext File Systems

Extended File Systems

(Carrier: Chapters 14 and 15)

Introduction

Unix-like (e.g., Linux) operating systems are usually able to use a wide variety of different file systems

Examples (not exhaustive)

File System	Description	Default for
UFS1, UFS2		Unix
Ext3, Ext4	Extended FS	Debian, Ubuntu
XFS	iriX FS	RHEL, CentOS, Fedora 22, OpenSUSE
JFS	Journaling File System	IBM's AIX & OS2
Reiser		SUSE (earlier versions)
HFS, HFS+	Hierarchical FS	MacOS 10
Btrfs	B-Tree File System	Fedora 33, OpenSuse
CRFS	Coherent Remote File System	Oracle
NTFS		Windows

A Bit More Info About Different Unix/Linux File Systems

Mac OS10

- *Through v 10.12 runs a version of FreeBSD and the HFS+ file system*
- *Version 10.13 runs APFS (Apple File System)*
  - Optimized for solid-state drive storage and encryption

Many of the Linux distributions use either Ext2 – Ext4

- *Red Hat & Ubuntu default to Ext3 or Ext4*

Novell SUSE's default was Reiser. The defaults are now

- *Btrfs (B-tree File System) for OS*
- *XFS for data partitions (optimized for manipulating large files)*
- *EXT2-EXT4 are supported*

But you don't need to use the default

- *With Ubuntu and most other versions of Linux you can choose Reiser, XFS, Ext2, Ext3, Ext4, XFS or other FSs*

A Bit More Info About Different Unix/Linux File Systems

Today Ext3 & Ext4 are arguably the most often used Linux file systems

- *Btrfs has gained some (B-tree, Oracle)*
- *XFS has gained acceptance as well*

Ext3 is essentially Ext2 plus journaling

Ext4 extends Ext3 volume sizes

- Up to 16 terabytes ($16x10^{12} = 2^{40}$ bytes)

A Bit More About XFS

XFS

High-performance journaling file system

Created in 1993 by Silicon Graphics for their IRIX OS

Efficient for

Processing very large files

Transferring such files

Intrinsically a 64-bit system supporting up to 8 exabytes

Exabyte = 1024^6 = 1024 gigabytes = 2^{60} bytes

For 32-bit OSs, XFS supports up to 16 terabytes

Terabyte = 1024^4 = 1024 petabytes = 2^{40} bytes

Journaling File Systems

Journaling file systems log changes to a journal

Usually, a circular log in a dedicated area

After logging, the FS commits the changes to the file system itself

Helps assure the integrity of the file system through events such as power failures and system crashes

Reduces performance

What else?

Journals and Cyber Forensics

Journals can greatly help cyber forensics

By recording events and timing

- Actions of applications and system
- User activity
- Error messages
- Network activity
- …

What happened and when

- Time lines
- Sequences of events

TSK tools that list contents of file system journals

- jls
- jcat

ExtX

This lecture will discuss Ext2 – Ext4 together per Carrier

Following Carrier's notation, these slides will refer to either of them as **ExtX**

ExtX evolved from UFS

Simplified

Ext4 looks a lot like Ext3

Differences include

Volume size > 1 exabyte (1024^6 bytes)

File size >16 terabytes (16×10^{12} bytes)

Extents

Can reserve contiguous blocks for a file

Minimizes fragmentation but consumes a lot of space that may never be used

ExtX File System Structure

Overall Structure

The ExtX file system consists entirely of ***blocks***

- Blocks *are sets of contiguous sectors on a hard disk*
- Blocks *are of a fixed size for the entire file system*
  - e.g., 1024, 2048 or 4096 bytes
  - Block size is defined when ExtX is established
- Blocks *are numbered starting with the value* **0**

Space on a hard disk is allocated a *block* at a time

Blocks are atomic to ExtX

In this regard, what in FAT or NTFS is analogous to a *block*?

- ***Clusters*** *are atomic to FAT or NTFS*

Overall Structure

Blocks are further organized into ***block groups***

- *Often referred to as simply* **groups**
- **groups** *are also numbered starting with the value* **0**

The entire ExtX file system is made up of ***groups***

- *With one exception*

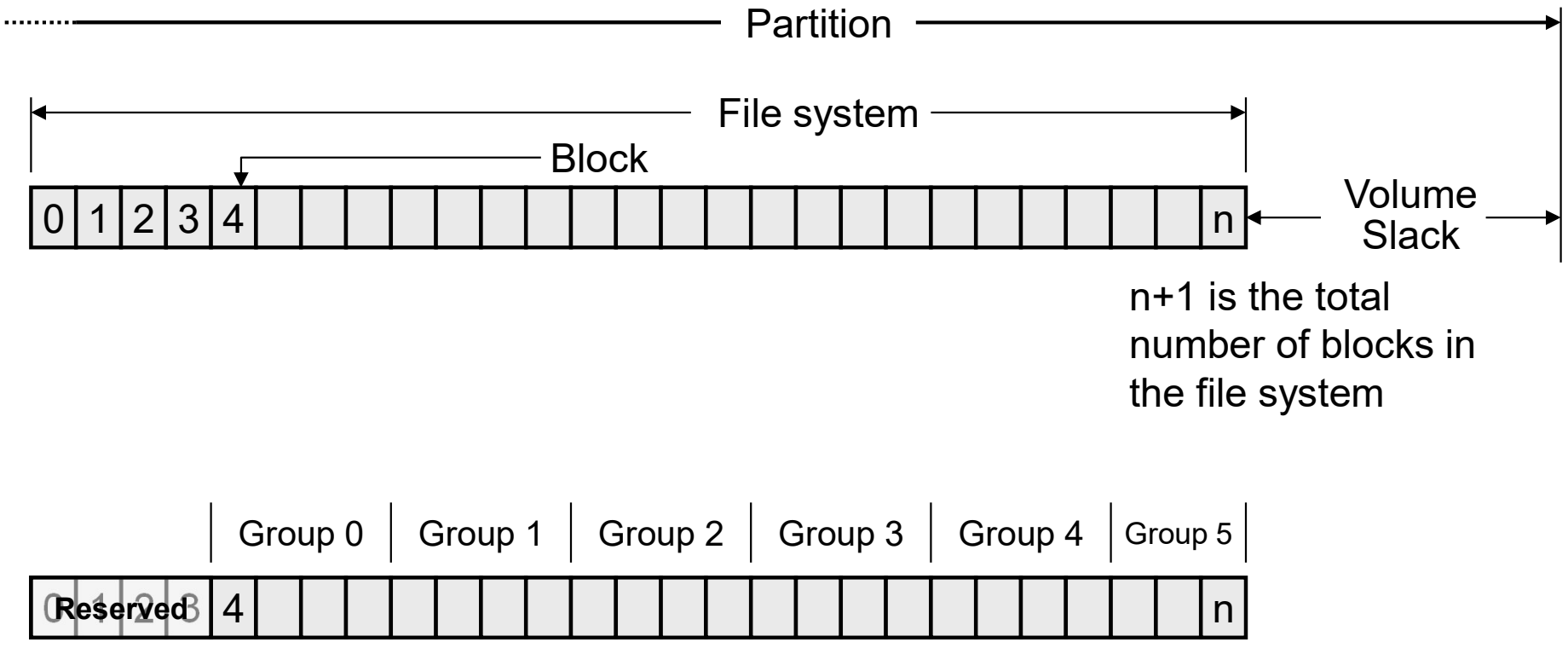
The exception

- *Optionally, a* **reserved area** *of blocks can be defined at the beginning of an ExtX file system*
- *The reserved area is an integer number of blocks*

All ***groups*** contain the same number of blocks…

- *Except for the last* **group**, *which can have fewer blocks*

Overall Structure



Superblock

The ***superblock*** contains much "essential" stuff

The size, location and configuration information for the entire file system

Without the **superblock** *an ExtX file system cannot function*

Superblock Content*
(slide 1 of 3)

Byte Range	Description	Essential
0–3	Number of inodes in file system	Yes
4–7	Number of blocks in file system	Yes
8–11	Number of blocks reserved to prevent file system from filling up	No
12–15	Number of unallocated blocks	No
16–19	Number of unallocated inodes	No
20–23	Block where block group 0 starts	Yes
24–27	Block size (saved as the number of places to shift 1,024 to the left)	Yes
28–31	Fragment size (saved as the number of bits to shift 1,024 to the left)	Yes
32–35	Number of blocks in each block group	Yes
36–39	Number of fragments in each block group	Yes
40–43	Number of inodes in each block group	Yes
44–47	Last mount time	No

Superblock Content *(slide 2 of 4)*
Superblock Bits 24-27 Explained

24-27: Number of places to shift 1024 to the left

If block size = 1024: How many bits is the value 1024 shifted?

$1024 = 2^{10} = 0100\ 0000\ 0000_2$ What is the value stored in bits 24-27?

The value stored in bits 24-27 = $0_{10} = 000$

If block size = 2048: How many bits is the value 1024 shifted?

$2048 = 2^{11} = 1000\ 0000\ 0000_2$ What is the value stored in bits 24-27?

The value stored in bits 24-27 = $1_{10} = 001$

If block size = 4096: How many bits is the value 1024 shifted?

$4096 = 2^{12} = 1\ 0000\ 0000\ 0000_2$ What is the value stored in bits 24-27?

The value stored in bits 24-27 = $2_{10} = 010$

Superblock Content*
(slide 2 of 3)

Byte Range	Description	Essential
48–51	Last written time	No
52–53	Current mount count	No
54–55	Maximum mount count	No
56–57	Signature (0xef53)	No
58–59	File system state (see Table 15.2)	No
60–61	Error handling method (see Table 15.3)	No
62–63	Minor version	No
64–67	Last consistency check time	No
68–71	Interval between forced consistency checks	No
72–75	Creator OS (see Table 15.4)	No
76–79	Major version (see Table 15.5)	Yes
80–81	UID that can use reserved blocks	No
82–83	GID that can use reserved blocks	No
84–87	First non-reserved inode in file system	No
88–89	Size of each inode structure	Yes
90–91	Block group that this superblock is part of (if backup copy)	No

Superblock Content*
(slide 3 of 3)

Byte Range	Description	Essential
92–95	Compatible feature flags (see Table 15.6)	No
96–99	Incompatible feature flags (see Table 15.7)	Yes
100–103	Read only feature flags (see Table 15.8)	No
104–119	File system ID	No
120–135	Volume name	No
136–199	Path where last mounted on	No
200–203	Algorithm usage bitmap	No
204–204	Number of blocks to preallocate for files	No
205–205	Number of blocks to preallocate for directories	No
206–207	Unused	No
208–223	Journal ID	No
224–227	Journal inode	No
228–231	Journal device	No
232–235	Head of orphan inode list	No
236–1023	Unused	No

Primary Superblock Location

The 1st KB (1024 bytes) of the non-reserved part of the file system contains either

The boot code of the loader is contained in the file system,

or

It is unused

The *primary superblock* is stored in the next 1KB

All superblocks *are 1KB in size*

Primary Superblock Location

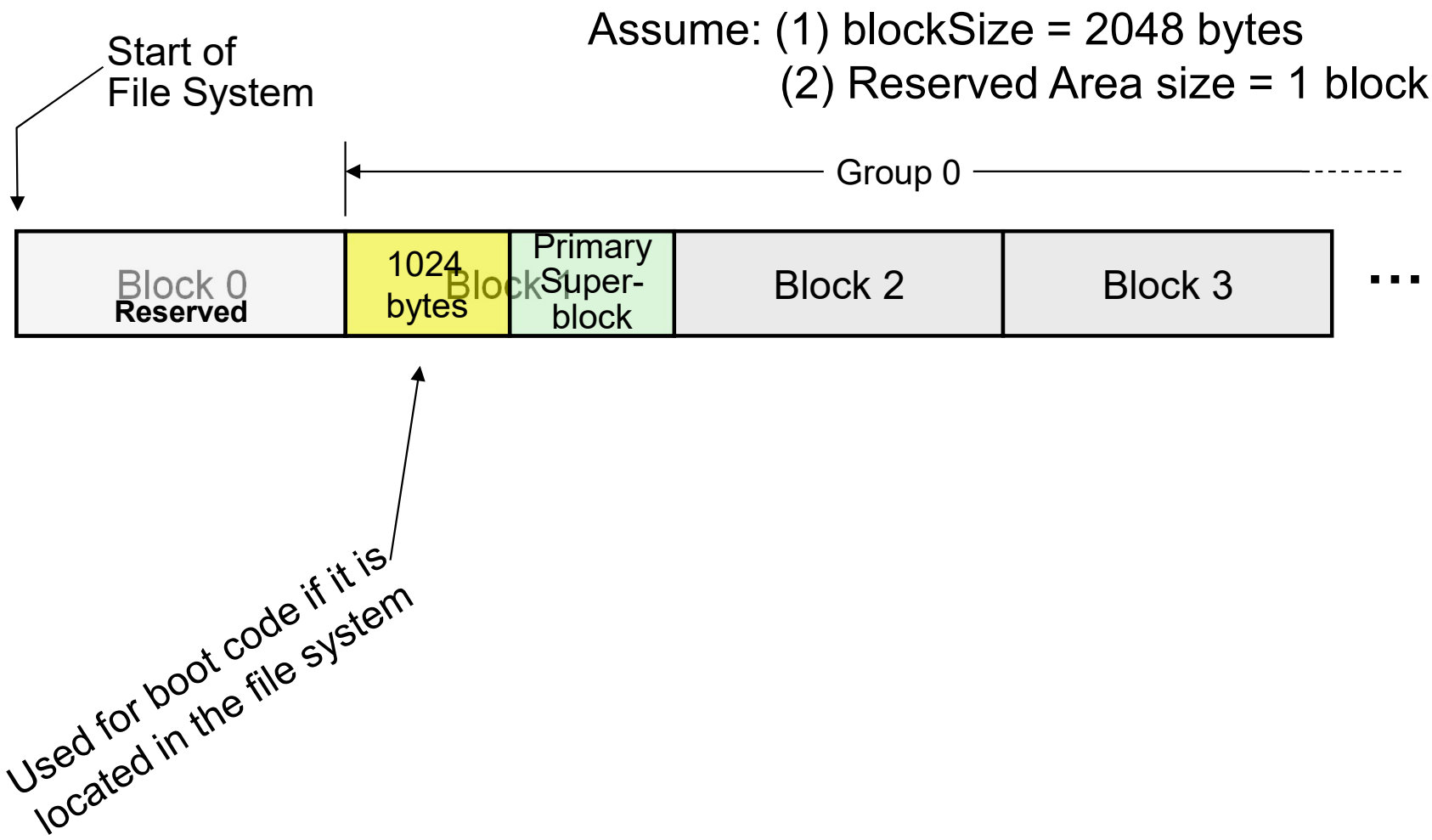
If the block size is defined to be 1 KB

The boot information (if it exists) is contained in block 0, and

The primary superblock is contained in block 1

Larger blocks (i.e., 2kB or 4kB) contain both the boot space and the primary superblock

Primary Superblock Location Example



More On Blocks

Everything is stored in *blocks*

Blocks can contain

- *File system structure information*
  - ***Superblocks***, ***Group Descriptor Tables***
- *File content*
- *Directory content*
- *File and directory metadata*
  - ***Inodes*** and ***Extended Attributes***
  - ***Block bitmaps***, ***Inode bitmaps***
- *File and directory names*

Block Groups / Groups*

Block Groups are groups of *blocks*

Block Groups are often referred to as *Groups*

Each *group* contains *blocks* that contain, in the following order:

1. *A* **Superblock**

   Unless sparse superblock feature is enabled

2. *A* **Group Descriptor Table**

   Unless sparse superblock feature is enabled

3. *A* **Block Bitmap** *for the group*
4. *An* **Inode Bitmap** *for the group*
5. *An* **Inode Table** *for the group*
6. *Blocks with* **file content**

Block Groups / Groups

If sparse superblock is used, only some block groups contain backup copies of the superblock & group descriptor table.

In a sample file system, Carrier found that

Groups 1, 3, 5, 7, and 9 had backup copies

The next copies did not occur until groups 25 and 27

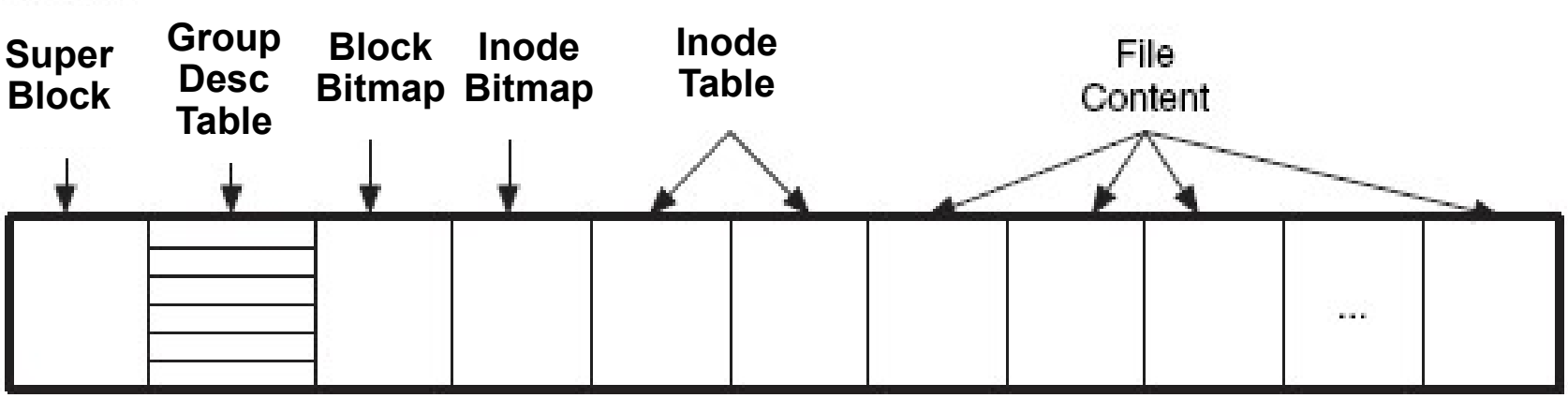
Sparse superblock is enabled by default in Linux ExtX.

Forensic Implications

Should not influence an investigation

But remember that only some **Groups** *have reserved space for superblock backups*

Structure of a Block ***Group***



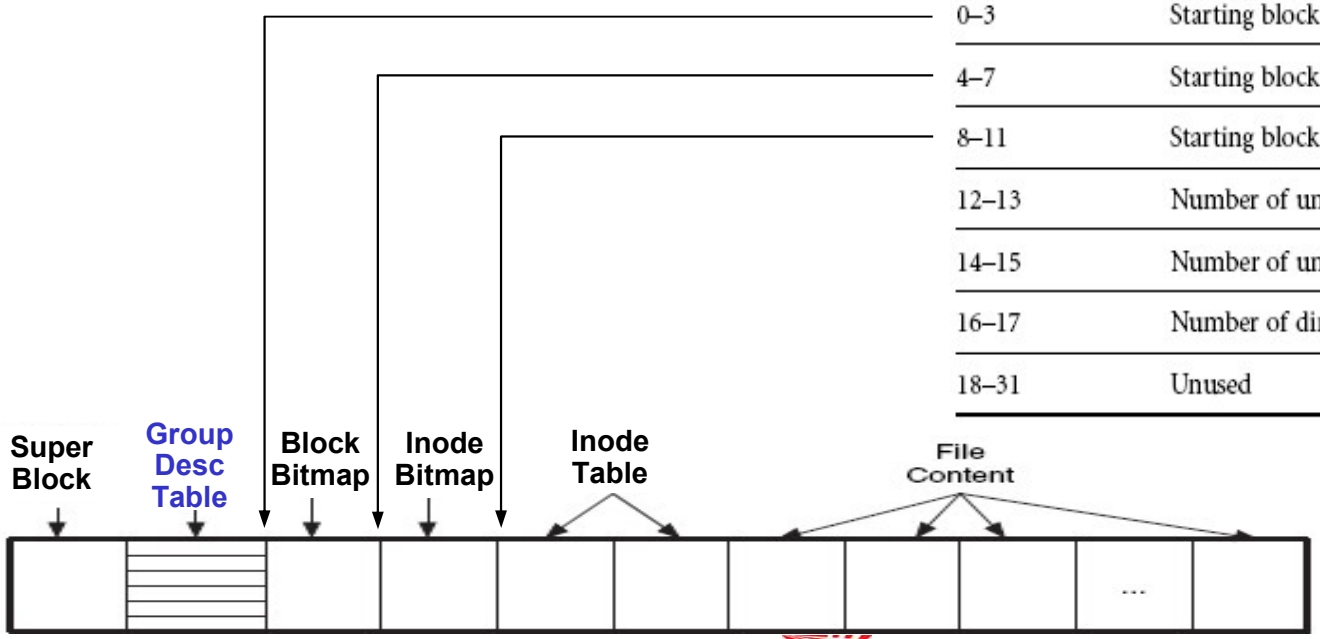
Group Descriptor Table

Contains a data structure table entry for every group in the file system

May not exist in all groups if sparse superblock is enabled

Each 32-byte *Group Descriptor Table* entry contains the following:

Byte Range	Description	Essential
0–3	Starting block address of block bitmap	Yes
4–7	Starting block address of inode bitmap	Yes
8–11	Starting block address of inode table	Yes
12–13	Number of unallocated blocks in group	No
14–15	Number of unallocated inodes in group	No
16–17	Number of directories in group	No
18–31	Unused	No



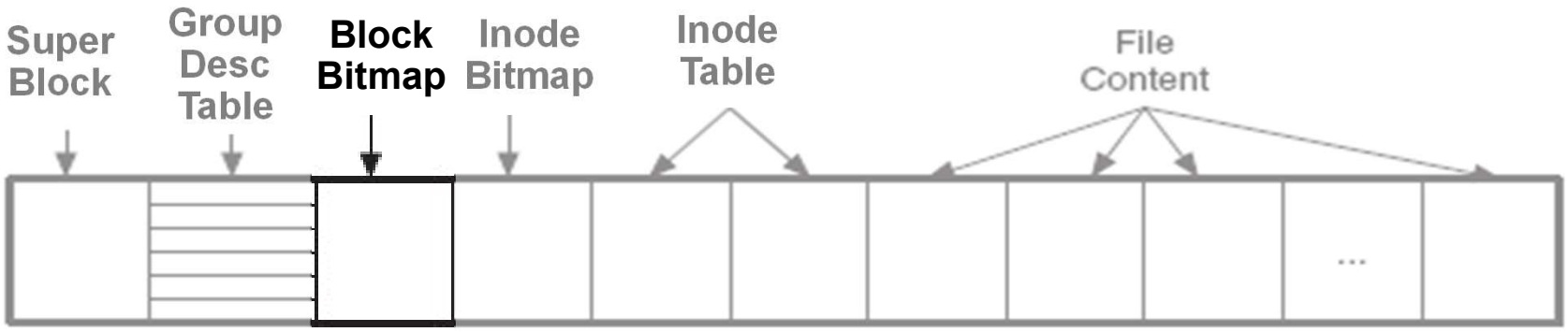
Group Descriptor Table

Again for emphasis

*The Group Descriptor Table contains an entry for **every block group** in the **entire file system***

Not only the block in which the Group Descriptor Table resides

Block Bitmap*



Contains the block allocation status for each block in the group

One bit for each block in the group

What's this analogous to in another file system that you studied?

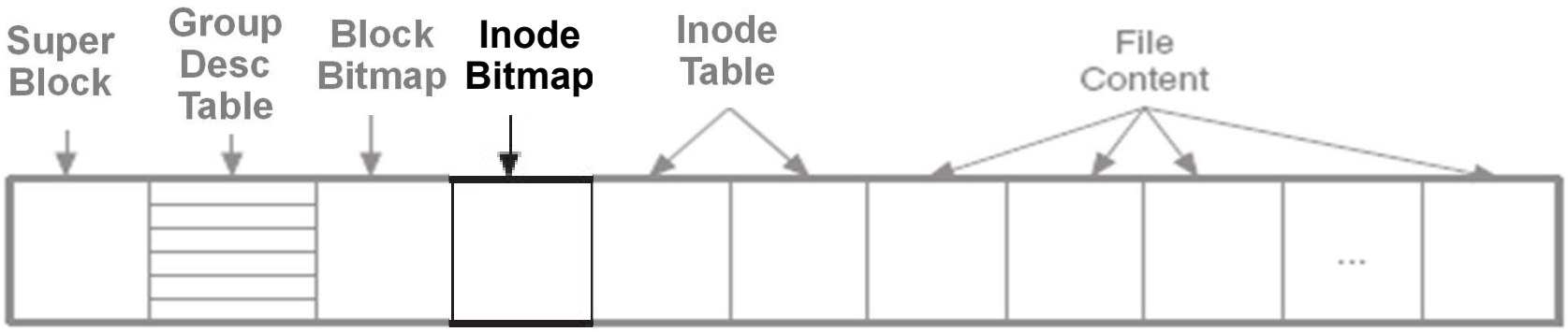
Occupies one or more blocks (usually only one)

Group Descriptor Table entry for the specific *group* gives the Block Bitmap starting block number

Last block **used** for Block Bitmap can be determined as follows:

Last Block Bitmap block = Inode Bitmap starting block number -1

Inode Bitmap*



Contains the inode allocation status for each inode in the group

One bit for each inode in the group

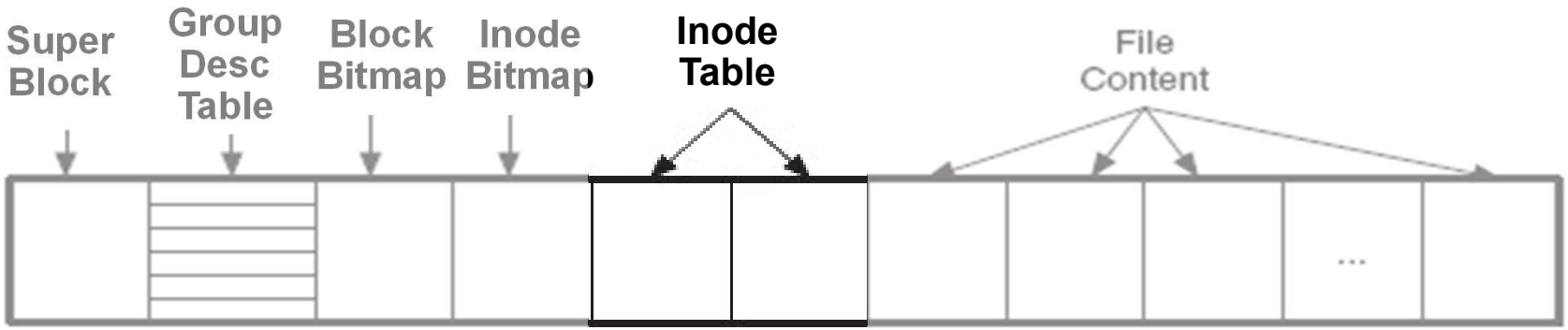
Occupies one or more blocks (usually only one)

Group Descriptor Table entry for the specific *group* gives the Inode Bitmap starting block number

Number of Inodes per Group is defined in the SuperBlock

Size of Inode Bitmap = (Number of Inodes per Group) ÷ 8.

Inode Table*



Data structure containing 128-byte inode entries

Each entry contains metadata for a single file or directory

Table occupies one or more blocks

Group Descriptor Table entry for the specific *group* gives the Inode Table starting block number

Number of Inodes per Group is defined in the SuperBlock

Size of Inode Table = (Number of Inodes per Group) x 128

Contents of Group 0

Primary SuperBlock

Scope: Entire file system

Primary Group Descriptor Table

Scope: Entire file system (all groups)

Block 0 Bitmap

Scope: Group 0

Block 0 Inode Bitmap

Scope: Group 0

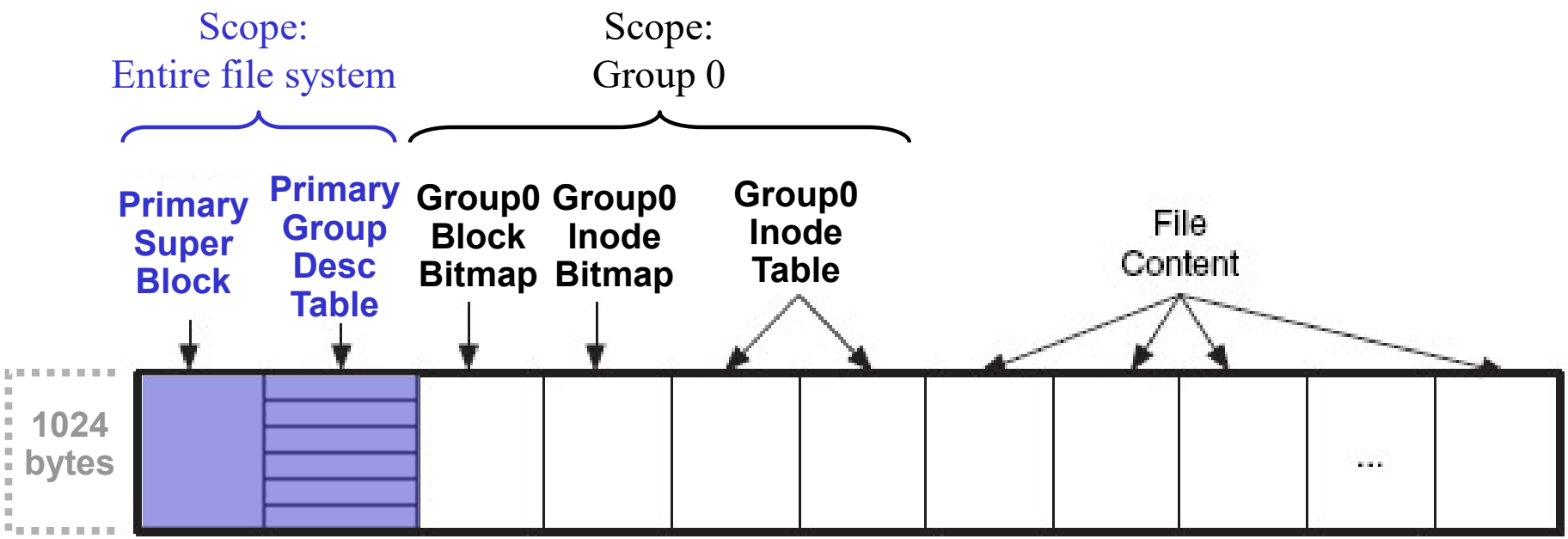
Block 0 Inode Table

Scope: Group 0

Block 0 file contents

Scope: Group 0, but not a meaningful concept

Contents of Group 0



Contents of Groups 1 to n

Backup/Secondary SuperBlock

Scope: Entire file system

Backup/Secondary Group Descriptor Table

Scope: Entire file system (all groups)

Block i Bitmap

Scope: Group i

Block i Inode Bitmap

Scope: Group i

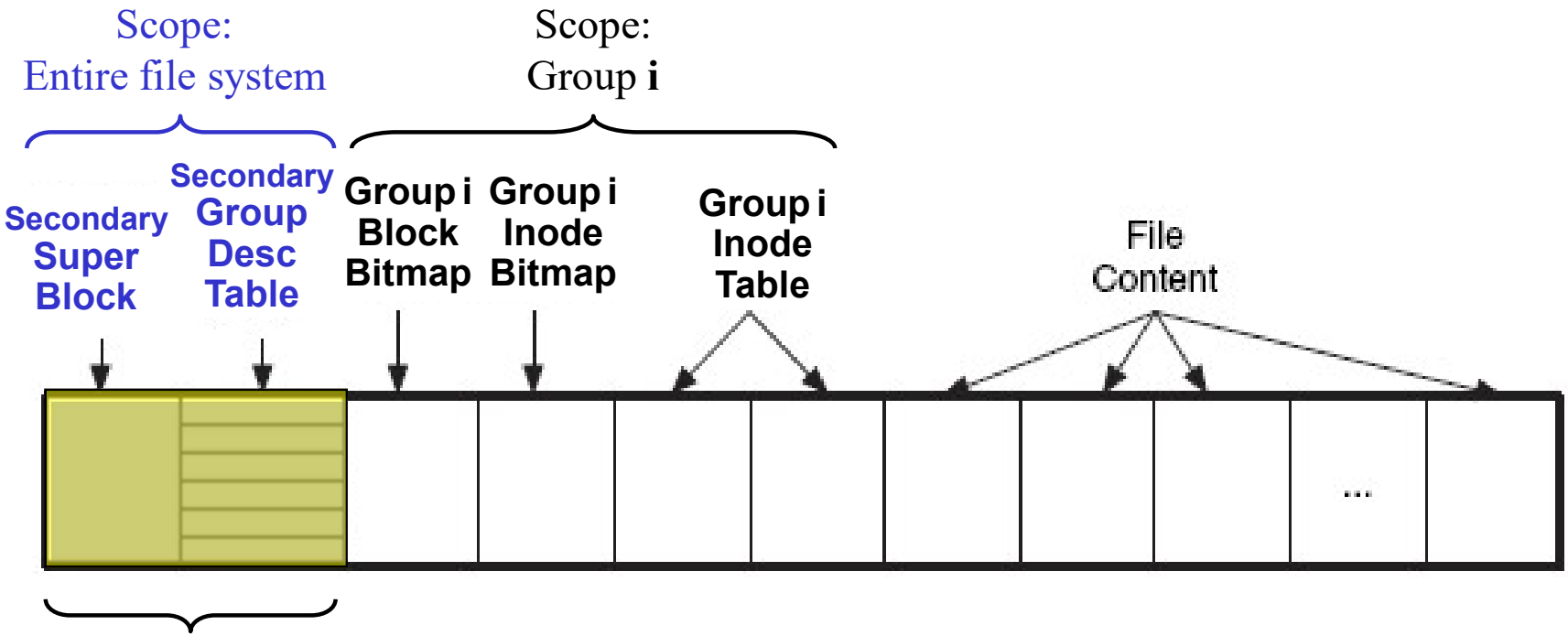
Block i Inode Table

Scope: Group i

Block i file contents

Scope: Group i

Contents of Groups 1 to n



May not exist if ***sparse superblock*** feature is enabled

Inode Table Detail

What's an *inode*?

Index node? Information Node?

Used since the first version of UNIX

Contains metadata about files and directories contained in a ***group***

Contents of Inode Table Entry

Byte Range	Description	Essential
0–1	File mode (type and permissions) (see Tables 15.11, 15.12, and 15.13) *(pp 458-459)*	Yes
2–3	Lower 16 bits of user ID	No
4–7	Lower 32 bits of size in bytes	Yes
8–11	Access Time	No
12–15	Change Time	No
16–19	Modification time	No
20–23	Deletion time	No
24–25	Lower 16 bits of group ID	No
26–27	Link count	No
28–31	Sector count	No
32–35	Flags (see Table 15.14) *(p 460)*	No
36–39	Unused	No
40–87	12 direct block pointers *(4 bytes each)*	Yes
88–91	1 single indirect block pointer	Yes
92–95	1 double indirect block pointer	Yes

Byte Range	Description	Essential
96–99	1 triple indirect block pointer	Yes
100–103	Generation number (NFS)	No
104–107	Extended attribute block (File ACL)	No
108–111	Upper 32 bits of size / Directory ACL	Yes / No
112–115	Block address of fragment	No
116–116	Fragment index in block	No
117–117	Fragment size	No
118–119	Unused	No
120–121	Upper 16 bits of user ID	No
122–123	Upper 16 bits of group ID	No
124–127	Unused	No

Contents of Inode Table Entry
Block Pointers

Byte Range	Description	Essential
0–1	File mode (type and permissions) (see Tables 15.11, 15.12, and 15.13) *(pp 458-459)*	Yes
2–3	Lower 16 bits of user ID	No
4–7	Lower 32 bits of size in bytes	Yes
8–11	Access Time	No
12–15	Change Time	No
16–19	Modification time	No
20–23	Deletion time	No
24–25	Lower 16 bits of group ID	No
26–27	Link count	No
28–31	Sector count	No
32–35	Flags (see Table 15.14) *(p 460)*	No
36–39	Unused	No
40–87	12 direct block pointers *(4 bytes each)*	Yes
88–91	1 single indirect block pointer	Yes
92–95	1 double indirect block pointer	Yes

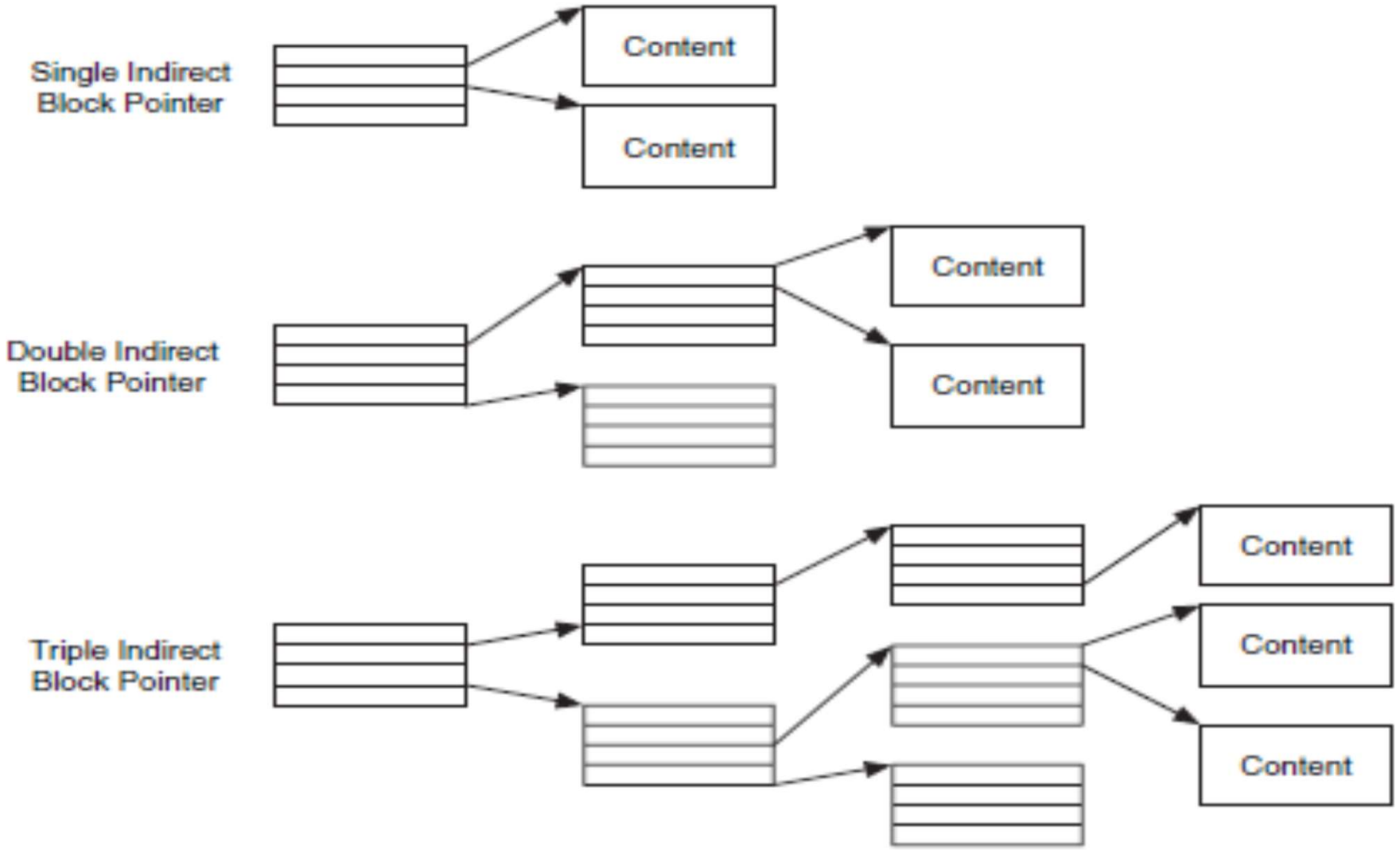
Byte Range	Description	Essential
96–99	1 triple indirect block pointer	Yes
100–103	Generation number (NFS)	No
104–107	Extended attribute block (File ACL)	No
108–111	Upper 32 bits of size / Directory ACL	Yes / No
112–115	Block address of fragment	No
116–116	Fragment index in block	No
117–117	Fragment size	No
118–119	Unused	No
120–121	Upper 16 bits of user ID	No
122–123	Upper 16 bits of group ID	No
124–127	Unused	No

Contents of Inode Table Entry
Block Pointers

Byte Range	Description	Essential
0–1	File mode (type and permissions) (see Tables 15.11, 15.12, and 15.13) *(pp 458-459)*	Yes
2–3	Lower 16 bits of user ID	No
4–7	Lower 32 bits of size in bytes	Yes
8–11	Access Time	No
12–15	Change Time	No
16–19	Modification time	No
20–23	Deletion time	No
24–25	Lower 16 bits of group ID	No
26–27	Link count	No
28–31	Sector count	No
32–35	Flags (see Table 15.14) *(p 460)*	No
36–39	Unused	No
40–87	12 direct block pointers *(4 bytes each)*	Yes
88–91	1 single indirect block pointer	Yes
92–95	1 double indirect block pointer	Yes
96–99	1 triple indirect block pointer	Yes
100–103	Generation number (NFS)	No

- Gains efficiency by the use of both direct & indirect block pointers
- The file *inode* can store pointers to the 1st 12 blocks containing file data
- If more than 12 blocks are needed, the *single indirect block pointer* points to a block containing more pointers
  - *This block can contain many 32-bit pointers depending on block size*
- If still more blocks are needed, *double indirect block pointer* is used
- And then *triple indirect block pointer*

Indirect Block Pointers



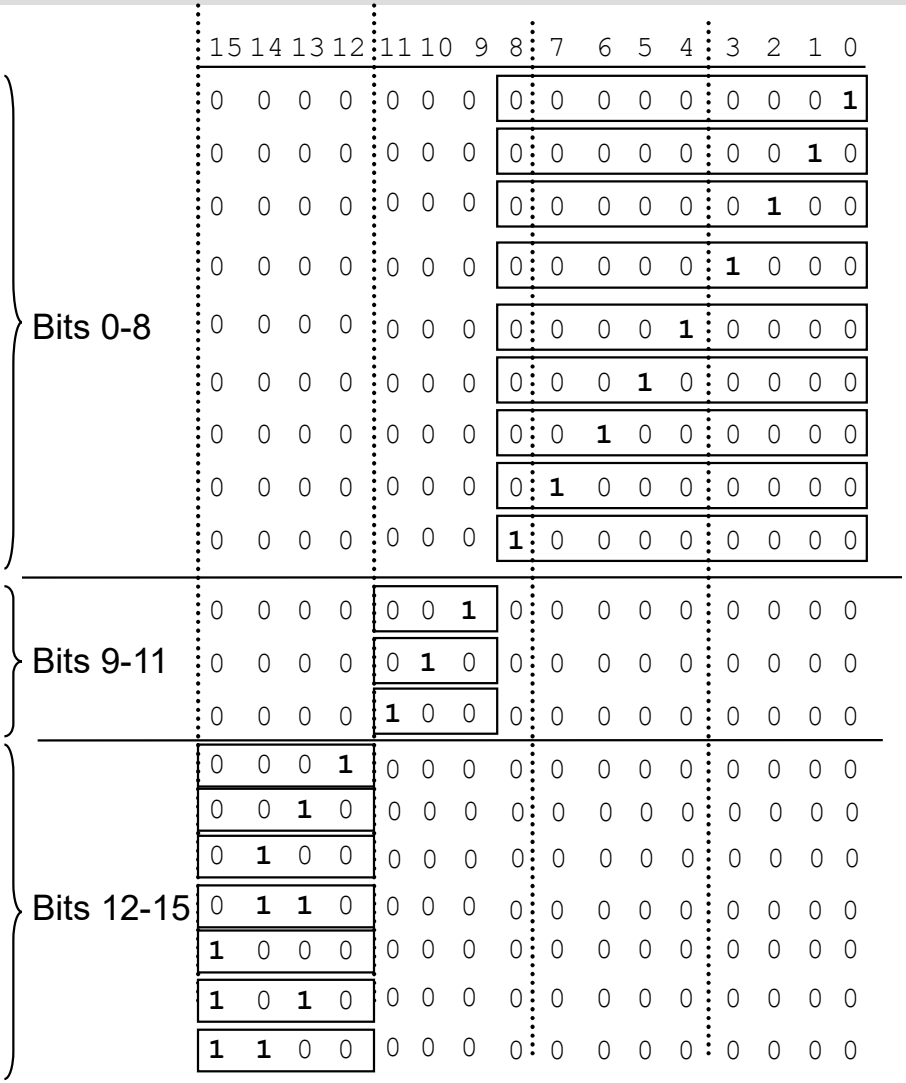
Contents of Inode Table Entry
File Mode

Byte Range	Description	Essential
0–1	File mode (type and permissions) (see Tables 15.11, 15.12, and 15.13) *(pp 458-459)*	Yes
2–3	Lower 16 bits of user ID	No
4–7	Lower 32 bits of size in bytes	Yes
8–11	Access Time	No
12–15	Change Time	No
16–19	Modification time	No
20–23	Deletion time	No
24–25	Lower 16 bits of group ID	No
26–27	Link count	No
28–31	Sector count	No
32–35	Flags (see Table 15.14) *(p 460)*	No
36–39	Unused	No
40–87	12 direct block pointers *(4 bytes each)*	Yes
88–91	1 single indirect block pointer	Yes
92–95	1 double indirect block pointer	Yes
96–99	1 triple indirect block pointer	Yes
100–103	Generation number (NFS)	No
104–107	Extended attribute block (File ACL)	No
108–111	Upper 32 bits of size / Directory ACL	Yes / No
112–115	Block address of fragment	No
116–116	Fragment index in block	No
117–117	Fragment size	No
118–119	Unused	No
120–121	Upper 16 bits of user ID	No
122–123	Upper 16 bits of group ID	No
124–127	Unused	No

File Mode Field in inode Entry

Carrier pp 458 & 459 Explained

Bit Position	Description
0x001	Other—execute permission
0x002	Other—write permission
0x004	Other—read permission
0x008	Group—execute permission
0x010	Group—write permission
0x020	Group—read permission
0x040	User—execute permission
0x080	User—write permission
0x100	User—read permission
0x200	Sticky bit
0x400	Set group ID
0x800	Set user ID
0x1000	FIFO
0x2000	Character device
0x4000	Directory
0x6000	Block device
0x8000	Regular file
0xA000	Symbolic link
0xC000	Unix socket



Segue

In ExtX so far, there are two important and related items that we've not discussed

Any ideas of what they are?

File Names

Where file names are located

File Modes

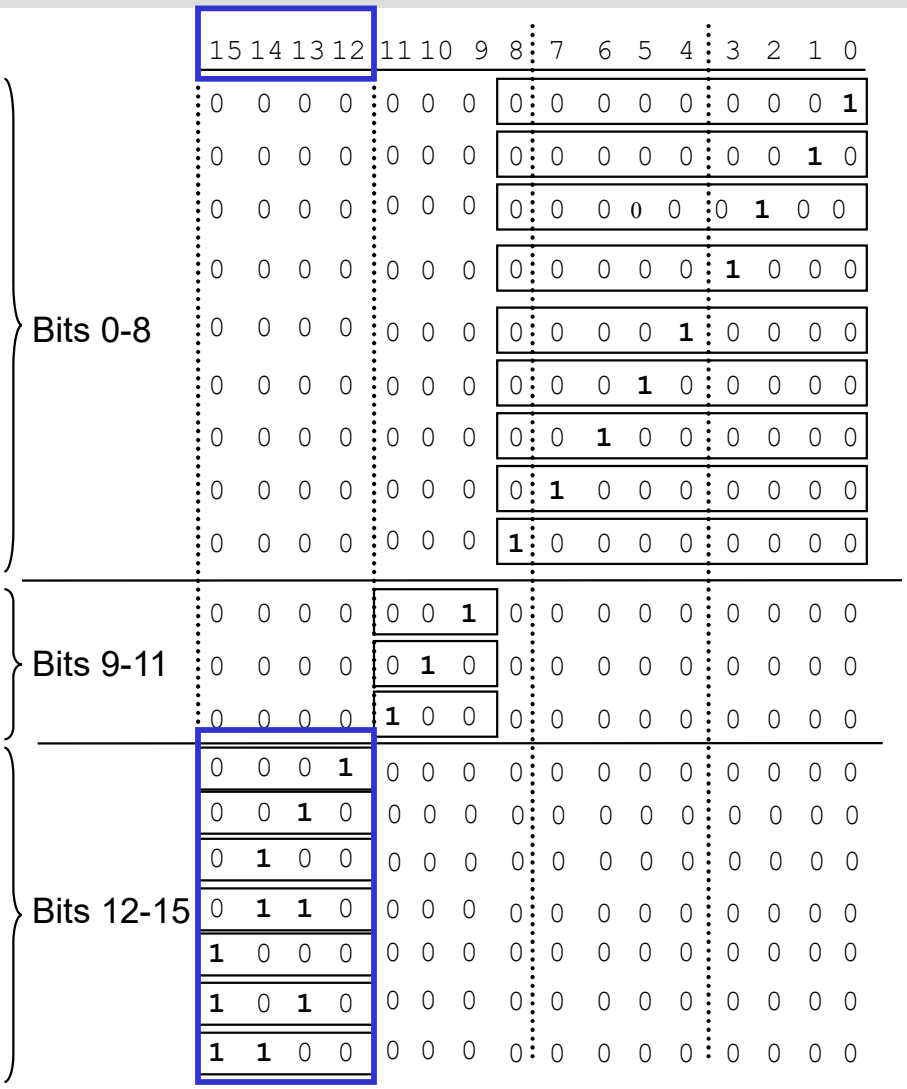
File Names

ExtX has several types of files

The types of files are called "modes"

File Mode Field in inode Entry

Bit Position	Description
0x001	Other—execute permission
0x002	Other—write permission
0x004	Other—read permission
0x008	Group—execute permission
0x010	Group—write permission
0x020	Group—read permission
0x040	User—execute permission
0x080	User—write permission
0x100	User—read permission
0x200	Sticky bit
0x400	Set group ID
0x800	Set user ID
0x1000	FIFO
0x2000	Character device
0x4000	Directory
0x6000	Block device
0x8000	Regular file
0xA000	Symbolic link
0xC000	Unix socket



Directory Mode Files

Directory mode files (or for Window's folk: ***folders***) contain, guess what?

Maybe some info about each file that it contains?

Maybe the name of each file In that directory?

What a revolutionary idea!

Directory Mode Files

If a file is a ***Directory***, it has a known & fixed format

It's a list of directory entries

Each *Directory* **entry** contains:

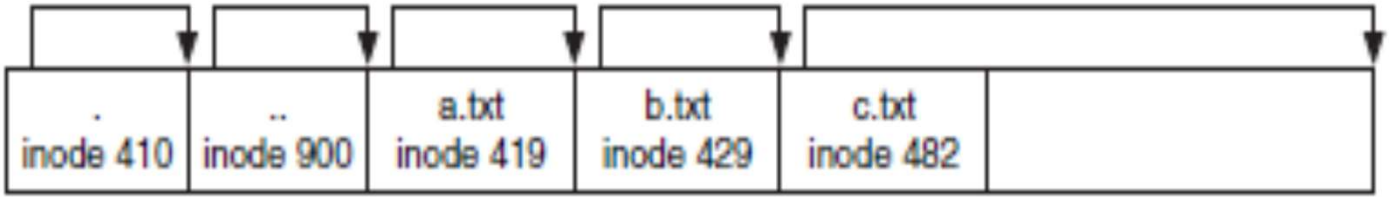
File name (1 to 255 characters)

inode pointer (to the inode for the file)

File name length (in multiples of 4 bytes)

Pointer to the beginning of the next file name in the directory

An Example Directory Mode File

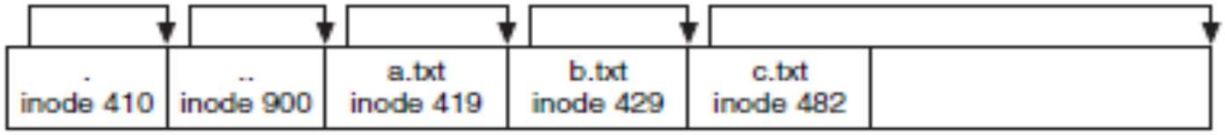


The 1st two entries in each Directory contains itself, "**.**" and its parent Directory, "**..**"

These entries are followed by entries for each file and sub directory

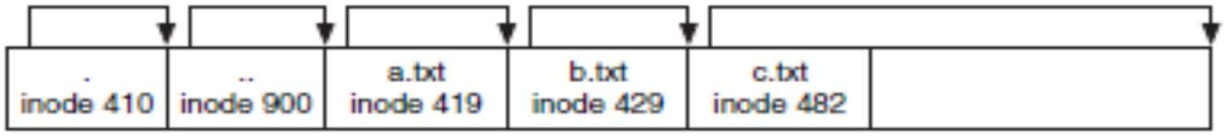
The last entry points to the end of the block allocated for the ***Directory***

Deleting a File

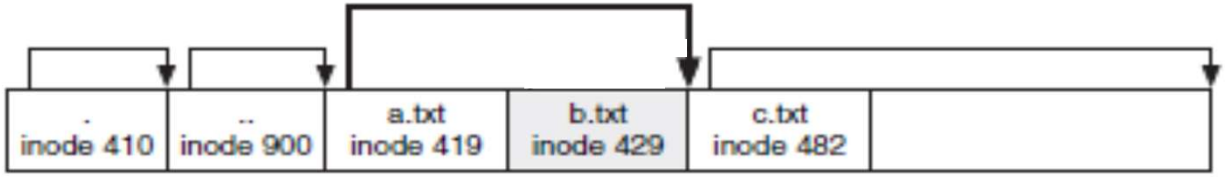


Now let's delete b.txt

Deleting a File



Before ***b.txt*** is deleted.



After ***b.txt*** is deleted.

In this example, file ***b.txt*** is deleted

Nothing is deleted!

Instead, the file before *b.txt* (*a.txt*) just points to the file after *b.txt*, which in this case is *c.txt*

Adding a File

When a file is added to a directory the OS:

- *Determines the space needed for the new entry*
- *Searches for the possibility of creating a space between existing file entries that can accommodate the new entry*
  - If it finds one, it adjusts the entry pointers and inserts the new entry
  - If it doesn't find one, it puts the new entry at the end of the list

Ordering of File Entries

By default, entries are ordered on a first-available basis

Carrier refers to this scheme as "unsorted"

There are other schemes such as hash-tree entries

In this scheme, entries are ordered by the hash of the file name

Hash trees are described by Carrier. There's no time to discuss them here.