文件系统

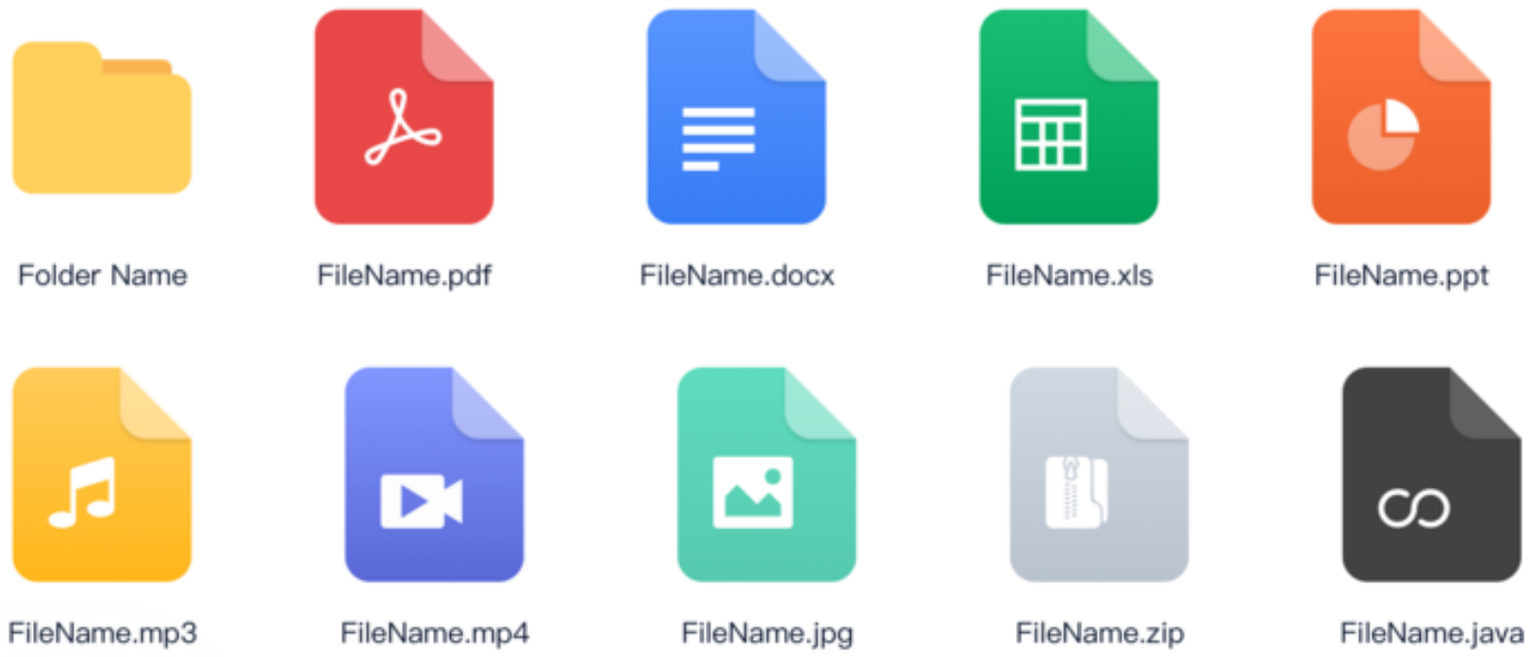
Section 5: Part I

2025

文件

支持数据的持久化存储是现代操作系统的重要功能之一

- 应用程序数据 (可执行文件、动态链接库 ...)
- 用户数据 (文档、图片、视频 ...)
- 系统数据 (配置文件 ...)



文件

文件 (File) 是操作系统为存储信息而创建的一个抽象概念

- 一个文件就是一个按名存取的字符序列 (an array of bytes)
  - 命名 (naming) 是文件抽象概念的一个重要特征
    - 用一个具有意义的名称来引用系统中的特定数据
      (a collection of bytes that you have a name for it)
  - 提供了一种将信息存储在磁盘上并在随后进行访问的机制

文件

文件由两部分组成

- 文件数据 (File Data): 字符序列
  - 用户可以对其进行创建、读取、写入和删除等操作
  - 具体内容由应用程序负责解释 (for flexibility)
- 文件属性 (File Metadata): 用于支撑文件功能的其他信息
  - *Size*: 文件大小 (字符序列的长度)
  - *Owner*: 文件的所有者
  - *Protection:* 不同用户对文件的访问权限
  - *Time*: 文件的创建时间、最近访问时间、最近修改时间
  - …

文件

围绕文件这一概念提供一组特定的文件操作 (接口)

- `open()`:打开一个文件
- `read()`:顺序读取一个打开文件中的若干字节
- `write()`:顺序向一个打开文件中写入若干字节
- `lseek()`:移动当前的偏移量
- `fsync()`:将当前对文件的修改立即写回磁盘
- `close()`:关闭一个已打开的文件
- `stat()`:获取文件的属性信息
- ...

文件

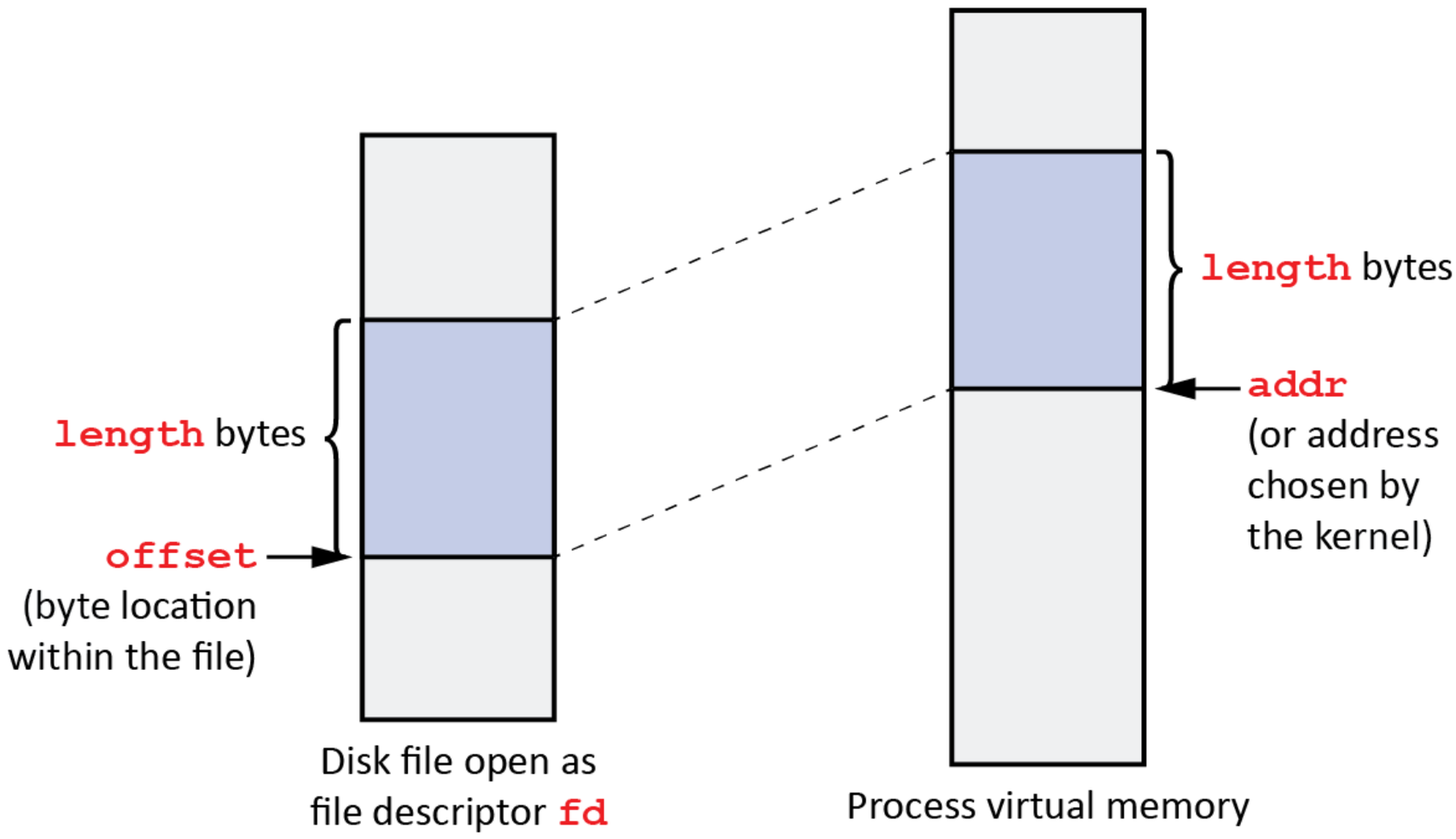
Unix 的 Everything is a File (an array of bytes / an object / a file descriptor)

- 使用同一组 APIs (with a file descriptor) 来操作不同的资源
  - `-`：普通文件、`d`：目录文件、`b`：块设备文件、`c`：字符设备文件、`p`：命名管道文件、`l`：符号链接文件 …
- Shell 中的 `stdin(fd = 0)`、`stdout(fd = 1)`、`stderr(fd = 2)`
- `/proc`：内核内部数据结构的接口
  e.g., `/proc/{PID}/maps`、`/proc/{PID}/mem`
- `/dev`：设备驱动的接口
  e.g., `/dev/random`、`/dev/null`

文件

内存映射文件 (Memory-Mapped File): 将文件字节序列映射到进程虚拟地址空间，随后进程可以使用内存访问指令来直接访问文件内容

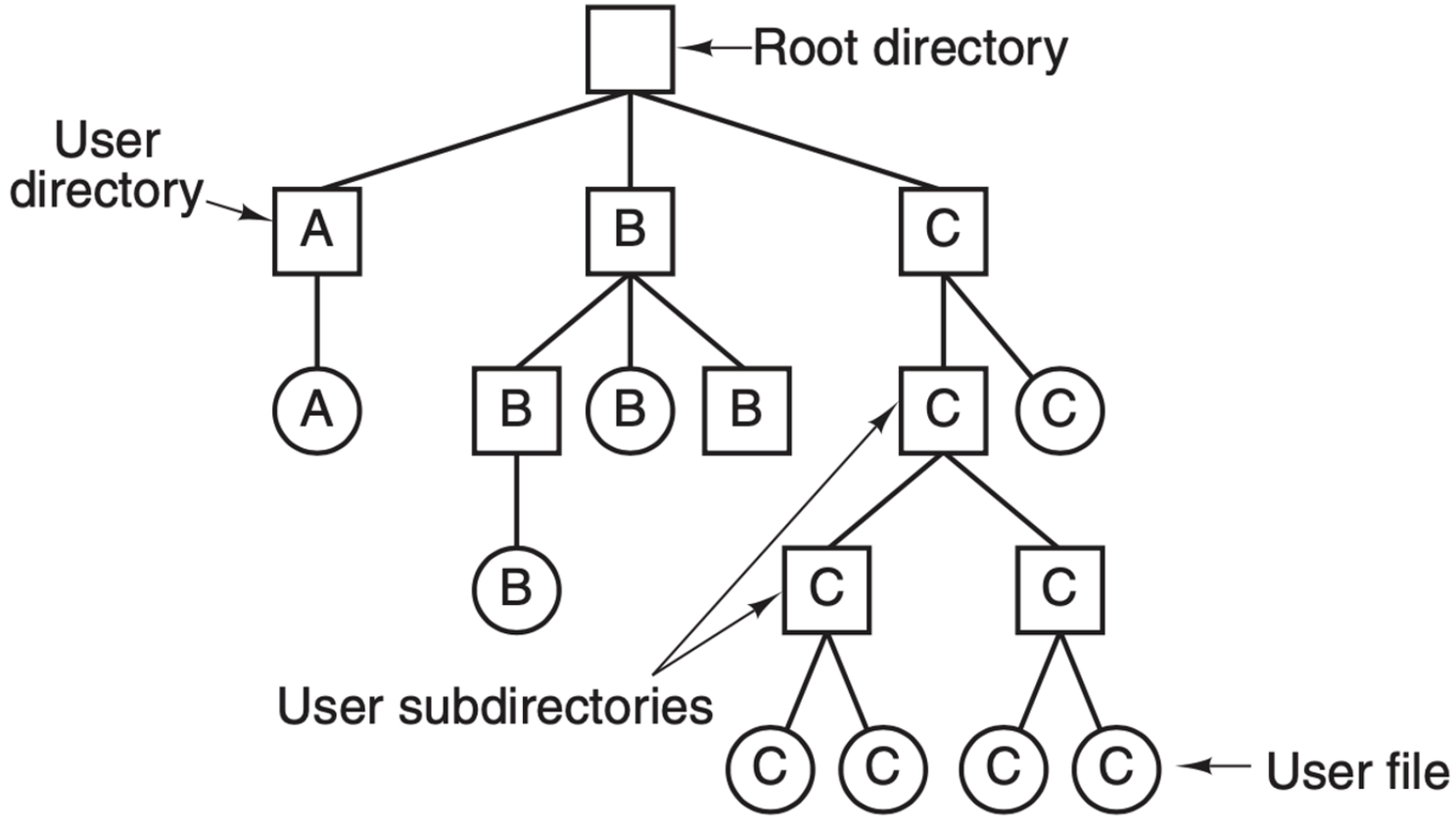
```
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset)
```



目录

在文件系统中通常使用目录 (directory) 来管理文件

- 从用户视角，可以将逻辑相关的数据放在同一个目录
- 通过在目录中创建子目录，形成一种目录树 (directory tree) 层次结构
  - 绝对路径和相对路径
  - Unix 中的 . 和 ..



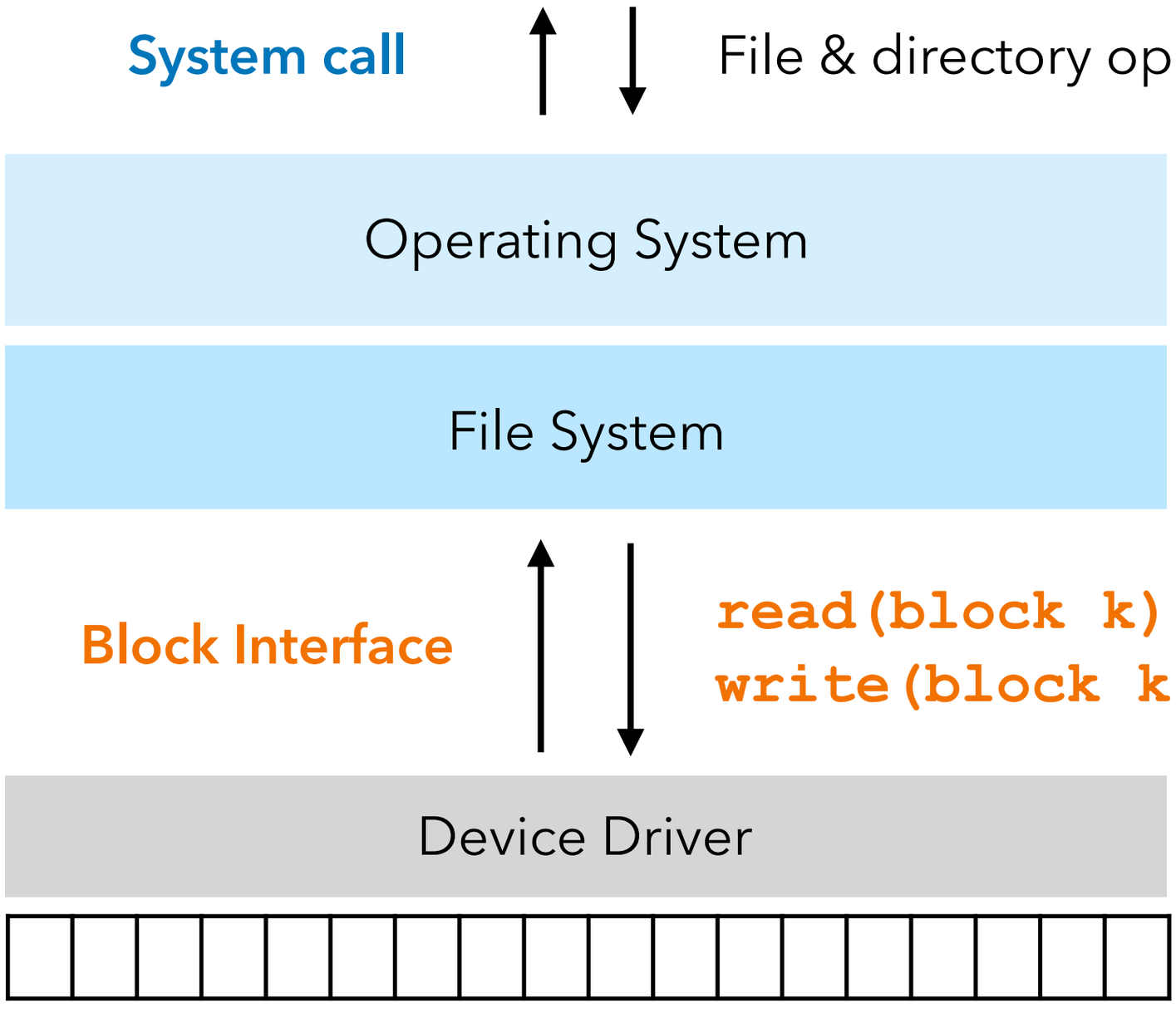
目录

围绕目录也有一组特定的目录操作

- `mkdir()`: 创建一个目录
- `rmdir()`: 删除一个目录
- `opendir()`:打开一个目录
- `readdir()`:返回已打开目录中的下一个目录项
- `link()`:将一个已有文件链接到一个目录项(创建文件)
- `unlink()`:删除目录中的一个目录项(删除文件)
- ...

文件系统

实现文件接口并负责管理文件数据和属性的系统即为文件系统



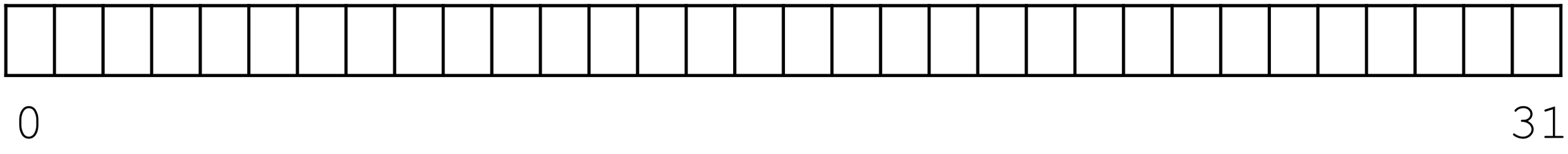
构建对数据持久化存储的分层抽象

以特定格式在存储设备上维护每个文件的数据和属性信息

- 高效利用磁盘空间
- 尽可能可靠

文件系统的实现

File System Layout



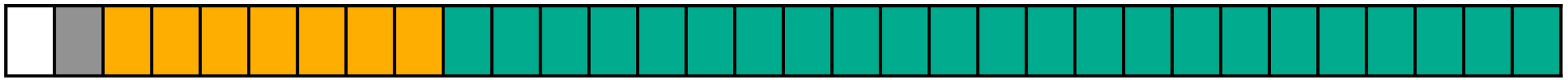
将整个磁盘划分为若干固定大小的磁盘块 (Disk Blocks)

文件系统的实现

File System Layout

Data Blocks

分配一定数量的 blocks 用于存储文件数据 (File Data)



0

31

File Attributes and Allocation Structure

分配一定数量的 blocks 用于存储文件属性 (File Metadata)、以及空闲空间管理相关的信息

Super Block

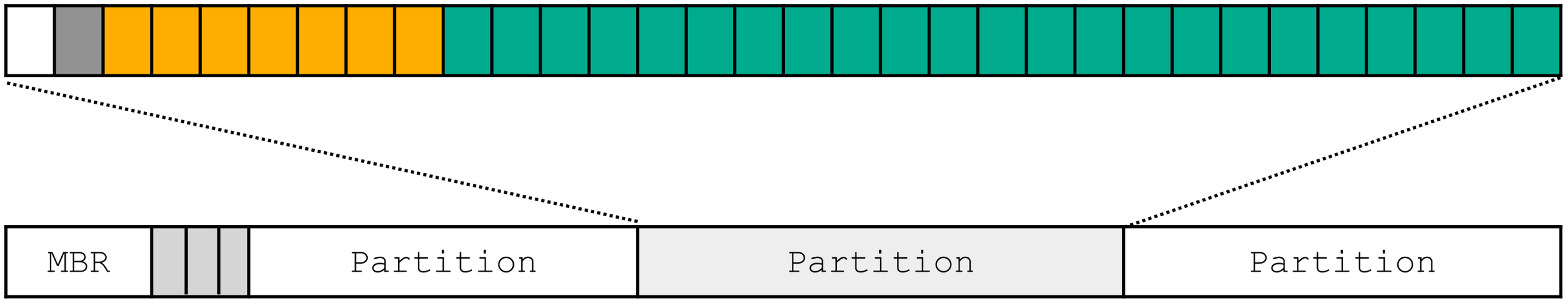
存储文件系统的 metadata 信息

Boot Block

存储 Boot OS 相关的信息 (empty if no OS)

文件系统的实现

File System Layout



Disk Partitions

不同 partition 可格式化为不同文件系统

Partition Table

存储每个 partition 的属性信息 (one is marked as active)

Master Boot Record (MBR)

存储 Boot Computer 相关的信息

文件系统的实现

File Control Block

文件系统需要设计和维护特定的数据结构来实现对文件的存储和管理

- 文件控制块 (File Control Block)
  - 记录文件属性 (File Metadata) 信息
  - 记录文件数据 (File Data) 在磁盘中的存储位置
    - 该结构同样需要在磁盘中持久化存储 (分配某些磁盘块)
- 在 Unix 文件系统中，该结构被称为 inode (index node)

文件系统的实现

File Control Block

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

A simplified **ext2** inode structure

文件的实现方式

记录 File Data 在磁盘中的存储位置：即文件的第 *N* 个 bytes (第 *M* 个 logical block) 存储在 File System Layout 中的哪个 Data Block

- 连续方式 (Contiguous Allocation)
- 链表方式 (Linked List Allocation)
  - FAT 文件系统的 File Allocation Table
- 索引方式 (Index Allocation)
  - Unix 文件系统的 inode Structure

文件的实现方式

Contiguous Allocation

使用磁盘上的若干连续 Data Blocks 来存储 File Data

✅ 仅需记录 number of first block + total number of blocks

✅ 有较好的顺序读写性能、且能快速定位随机 Data Block

Data Blocks


```
start = 0
length = 4
```

文件的实现方式

Contiguous Allocation

使用磁盘上的若干连续 Data Blocks 来存储 File Data

✅ 仅需记录 number of first block + total number of blocks

✅ 有较好的顺序读写性能、且能快速定位随机 Data Block

❎ 但显然不够灵活

- 在创建文件时需确定文件大小
- 还会存在严重的外部碎片问题
- 在特定场景下仍是很好的选择 (e.g., CD-ROM)

文件的实现方式

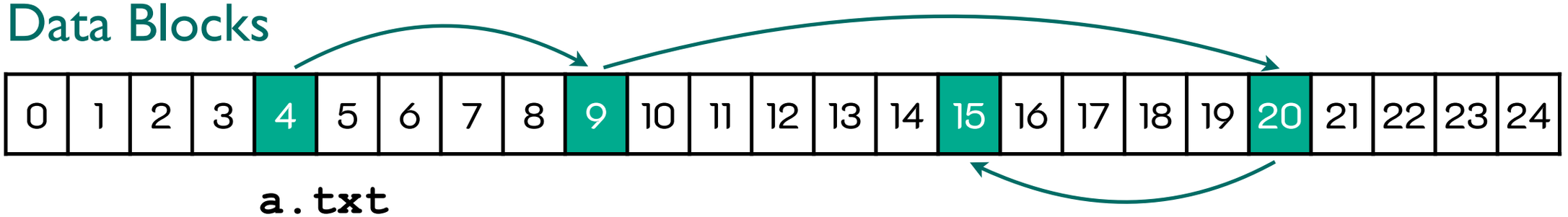
Linked List Allocation

使用 Data Blocks 的一个链表来存储 File Data

✅ 仅需记录 number of first block

✅ 可以把所有磁盘 Data Blocks 都利用起来 (没有外部碎片)

Data Blocks



a.txt

start = 4

文件的实现方式

Linked List Allocation

使用 Data Blocks 的一个链表来存储 File Data

✅ 仅需记录 number of first block

✅ 可以把所有磁盘 Data Blocks 都利用起来 (没有外部碎片)

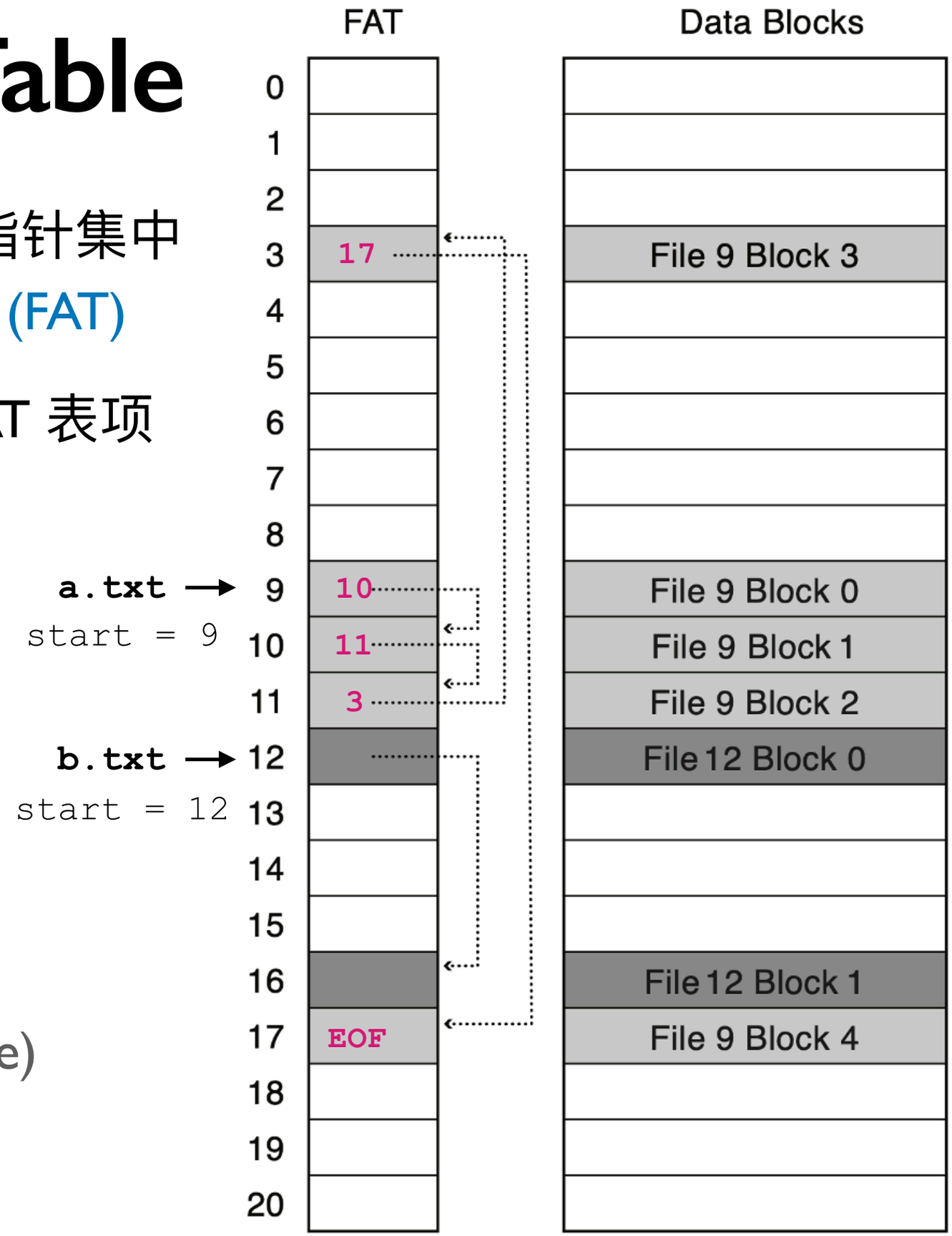
❎ 比较适合小文件，但是

- 定位一个大文件的随机 Data Block 需要遍历链表
- 每个 Data Block 中需要额外存储一个 next 指针 (Data Block 的可用大小不再是 2^k)
- 指针损坏导致文件后续数据丢失

File Allocation Table

把 Data Blocks 中的所有 next 指针集中存放到一个 File Allocation Table (FAT)

- 每个 Data Block 对应一个 FAT 表项
- 每个 FAT 表项存储
  - 文件下一个 Data Block 编号 (pointer to the next FAT entry of the file)
- 文件终止标记 (end of file) `0xFFF`
- 或空闲空间标记 (free space) `0x000`



File Allocation Table

- 有效改进了简单的 Linked List 实现
  - 遍历链表 ➡️ 只需访问 FAT (缓存在内存) 而不需依次读 Data Blocks
  - 可用空间 ➡️ Data Block 中不需要预留额外空间存储 next 指针
  - 指针损坏 ➡️ 在磁盘中存储多份 FAT
- 但是，随着磁盘存储空间的变大，在内存中缓存整个 FAT 会导致较高的存储开销
  - 对于 1TB (2^{40}) 大小的磁盘和 4KB (2^{12}) 大小的 Disk Block
  - 总共需要 2^{28} 个 FAT 表项
  - 如果每个表项占用 4 bytes (FAT 32)，则整个 FAT 大小为 1GB

文件的实现方式

Indexed Allocation

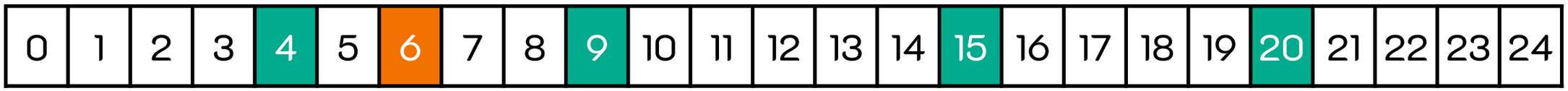
使用一个记录 Data Blocks 编号的数组来存储 File Data 的索引信息

✅ 仅需记录存储该数组结构的 Data Block 的编号

✅ 快速定位任意 Data Block (在内存中缓存索引结构)

✅ 需要缓存的大小只和当前已打开文件数相关 (而不是整个磁盘大小)

Data Blocks



a.txt

index = 6

```
4
9
20
15
END
```

第 *i* 项对应文件第 *i* 个 Data Block 的编号

文件的实现方式

Indexed Allocation

使用一个记录 Data Blocks 编号的数组来存储 File Data 的索引信息

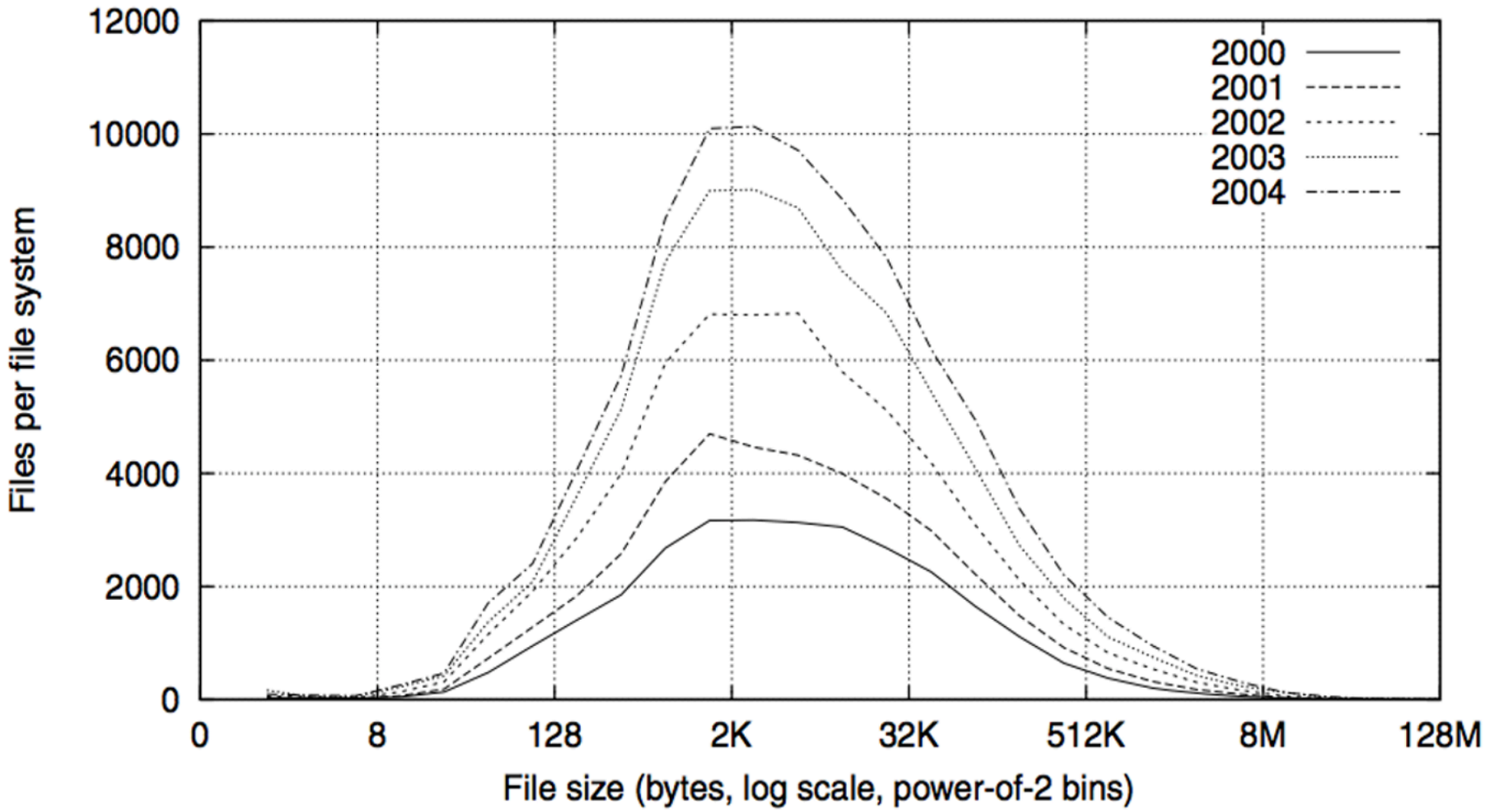
❇️ 但是，如何决定所需数组结构的长度?

- 文件系统中的文件大小不定
  - 既有小文件 (few bytes ~ KB)
  - 又有大文件 (several GB / TB)
- 通常希望 File Metadata 是固定长度 (便于管理)

文件的实现方式

Characteristics of Real-World File Systems

- Most files are small: ~2K is the most common size



Agrawal N, Bolosky W J, Douceur J R, et al. A five-year study of file-system metadata. ACM Transactions on Storage (TOS), 2007, 3(3): 9-es.

文件的实现方式

Characteristics of Real-World File Systems

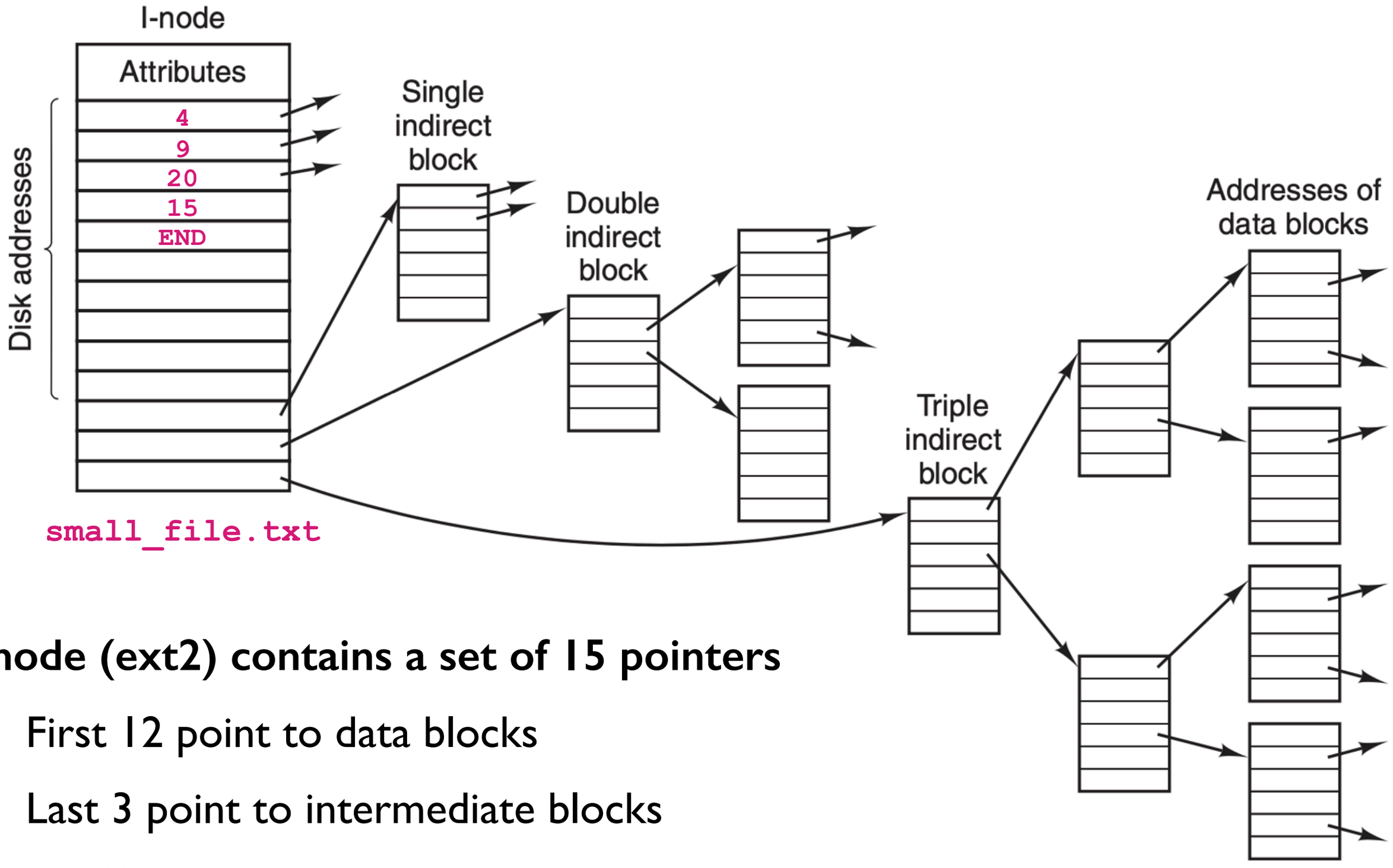
- Most files are small: ~2K is the most common size
- Average file size is growing: ~200K is the average
- Most bytes are in large files: a few big files use most of space
- File systems contains lots of files: almost 100K on average
- File systems are roughly half full: even as disks grow, file systems remain about 50% full
- Directories are typically small: many have few entries; most have 20 or fewer

Agrawal N, Bolosky W J, Douceur J R, et al. A five-year study of file-system metadata. ACM Transactions on Storage (TOS), 2007, 3(3): 9-es.

inode Structure

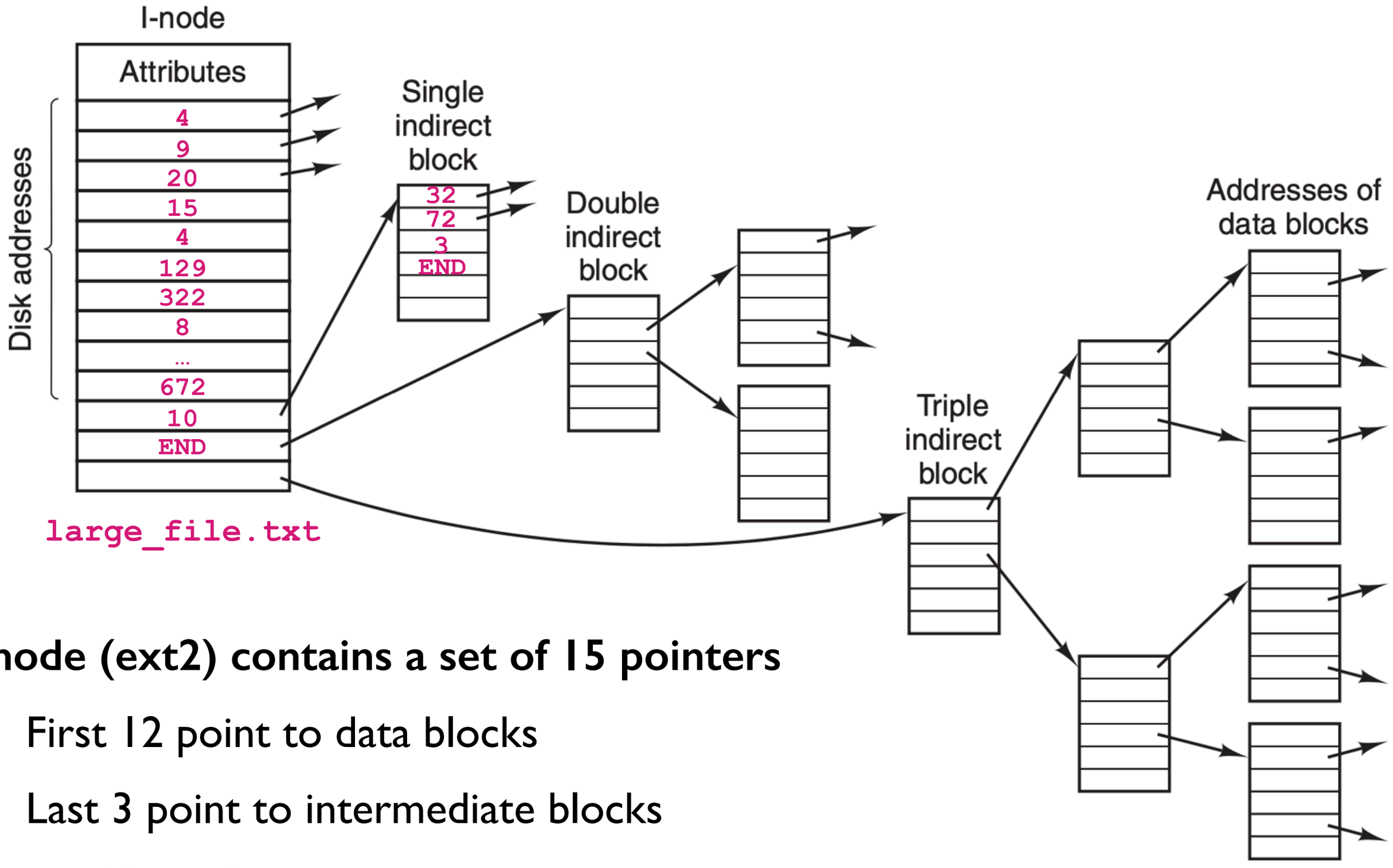
对小文件 (fast path) 和大文件 (slow path) 都提供良好支持

- 在 inode 结构中维护一组固定数目的 pointers
  - Direct pointers: 直接指向包含文件数据的 Data Block
  - Indirect pointers: 指向一个包含 direct pointers 的 Data Block
- 支持不同文件的快速随机访问
  - 小文件直接使用 direct pointers ➡️ An array structure
  - 大文件额外使用 indirect pointers ➡️ A tree structure



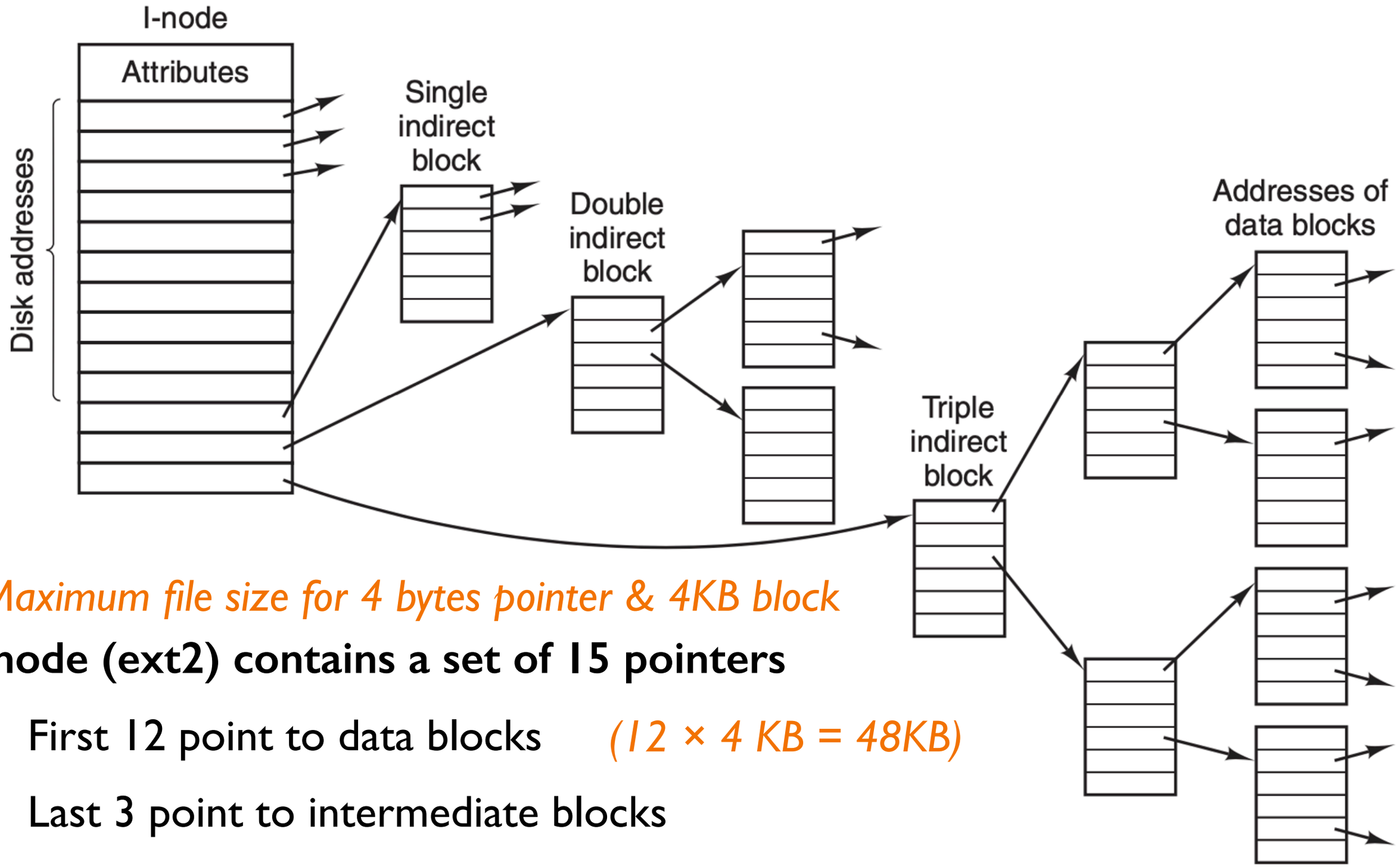
inode (ext2) contains a set of 15 pointers

- First 12 point to data blocks
- Last 3 point to intermediate blocks
  - #13: single indirect pointer
  - #14: double indirect pointer
  - #15: triple indirect pointer



inode (ext2) contains a set of 15 pointers

- First 12 point to data blocks
- Last 3 point to intermediate blocks
  - #13: single indirect pointer
  - #14: double indirect pointer
  - #15: triple indirect pointer



Maximum file size for 4 bytes pointer & 4KB block

inode (ext2) contains a set of 15 pointers

- First 12 point to data blocks *(12 × 4 KB = 48KB)*
- Last 3 point to intermediate blocks
  - #13: single indirect pointer *(2^{10} × 4 KB = 4MB)*
  - #14: double indirect pointer *(2^{10} × 2^{10} × 4 KB = 4GB)*
  - #15: triple indirect pointer *(2^{10} × 2^{10} × 2^{10} × 4 KB = 4TB)*

文件系统的实现

File and Directory

- 在已知 File Metadata 的情况下 (low-level name)，我们可以利用相关结构信息找到 File Data 存储在磁盘上的哪些 Data Block
  - FAT 文件系统 ➡️ number of first block + FAT
  - Unix 文件系统 ➡️ inode (direct & indirect pointers)
- 如何根据文件名 (human-friendly name) 来找到 File Metadata？

文件系统的实现

File and Directory

在文件系统中，目录 (Directory) 是一种用于存储文件名和文件底层结构间映射关系的特殊文件

- 目录文件也是一个文件
  - 同样具有 File Data 和 File Metadata
- 但目录文件的 File Data 在文件系统中具有特殊含义
  - 记录该目录下的每个文件对应的属性信息
  - 该信息属于文件系统内部维护的数据，通常不允许用户直接使用文件操作进行访问

FAT 文件系统

Directory Implementation

文件的属性 (File Metadata) 信息直接存储在目录文件中

- 目录文件的内容就是一个目录项 (directory entry) 的列表，其中每个目录项对应一个文件

```
dir
|-- a.txt
|-- list/
|-- main.c
```

File Name	Attributes	First Data Block
a.txt	…	9
list	…	102
main.c	…	82

File data of the `dir` file

FAT 文件系统

Directory Implementation

文件的属性 (File Metadata) 信息直接存储在目录文件中

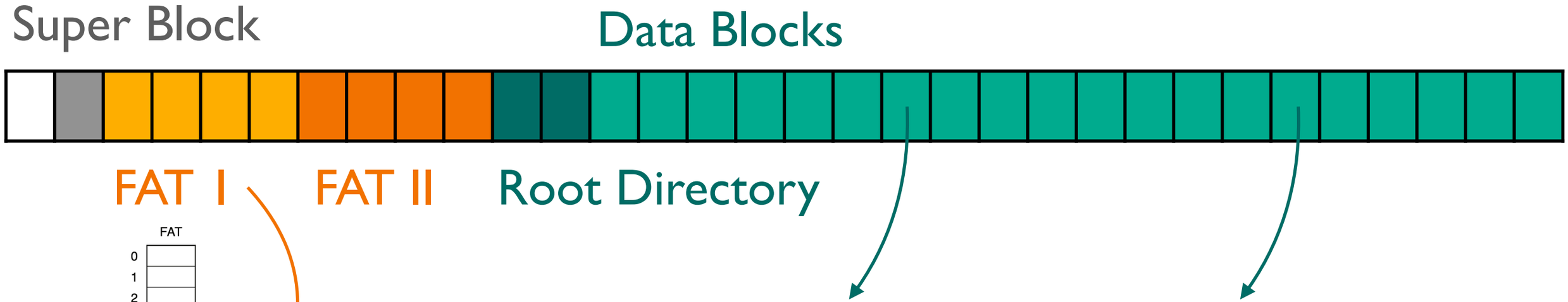
- 对于 FAT 32 文件系统，每个目录项大小为 32 bytes

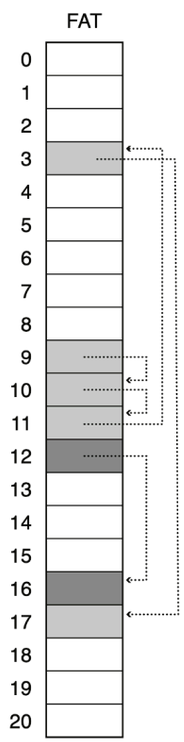
Descriptive name of field	Offset (byte)	Size (bytes)	Description
DIR_Name	0	11	“Short” file name limited to 11 characters (*8.3 format*).
DIR_Attr	11	1	Legal file attribute types are as defined below: ATTR_READ_ONLY 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 ATTR_DIRECTORY 0x10 ATTR_ARCHIVE 0x20 ATTR_LONG_NAME is defined as follows: (ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID) The upper two bits of the attribute byte are reserved and must always be set to 0 when a file is created. These bits are not interpreted.
DIR_NTRes	12	1	Reserved. Must be set to 0.

FAT 文件系统

File System Layout

FAT 文件系统的磁盘布局：结合其文件 (by File Allocation Table) 和目录 (Save File Metadata in Directory) 的实现方式





文件的存储和空闲空间管理信息

File Name	Attributes	First Data Block
a.txt	…	9
list	…	102
main.c	…	82

对于目录文件

Lorem ipsum dolor sit amet, singulis liberavisse complectitur his te. Eam ei graeci denique detracto, ea eius autem qui. Magna labore graecis vel et, mea ut propriae detraxit. Appetere moderatius his at, no quod erant vocent eam. Modus possit nec ex, ad quo invidunt oporteat.

对于普通文件

Unix 文件系统

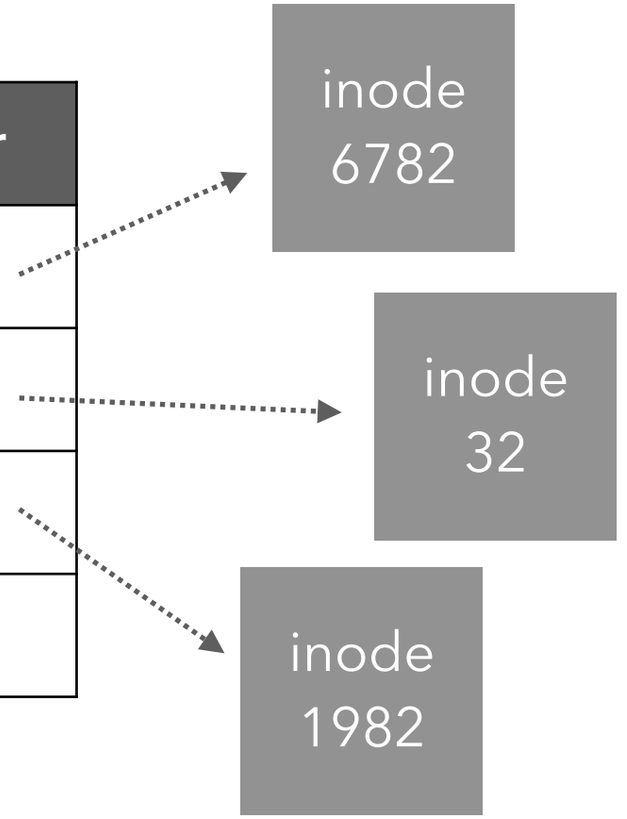
Directory Implementation

文件的属性信息存储在一个专门的结构中 (inode)

- 每个目录项记录 ⟨file name, inode number⟩ 的对应关系
- 为根目录分配一个特殊的 inode

```
dir
|-- a.txt
|-- list/
|-- main.c
```

File Name	inode Number
a.txt	6782
list	32
main.c	1982

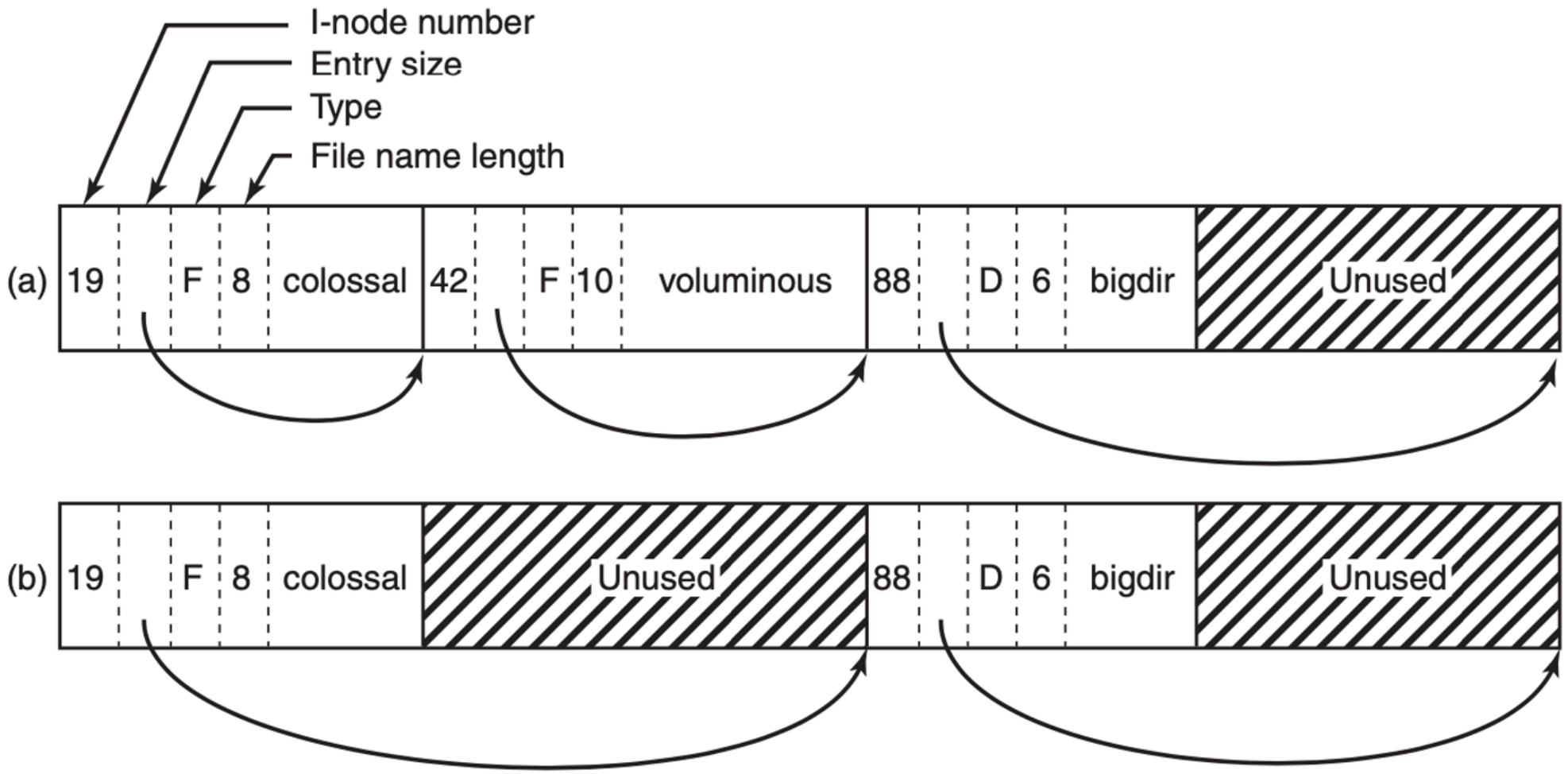


File data of the `dir` file

Unix 文件系统

Directory Implementation

可在目录项中额外记录文件名长度来支持不定长文件名 (或者利用多个目录项来记录一个长文件名文件)

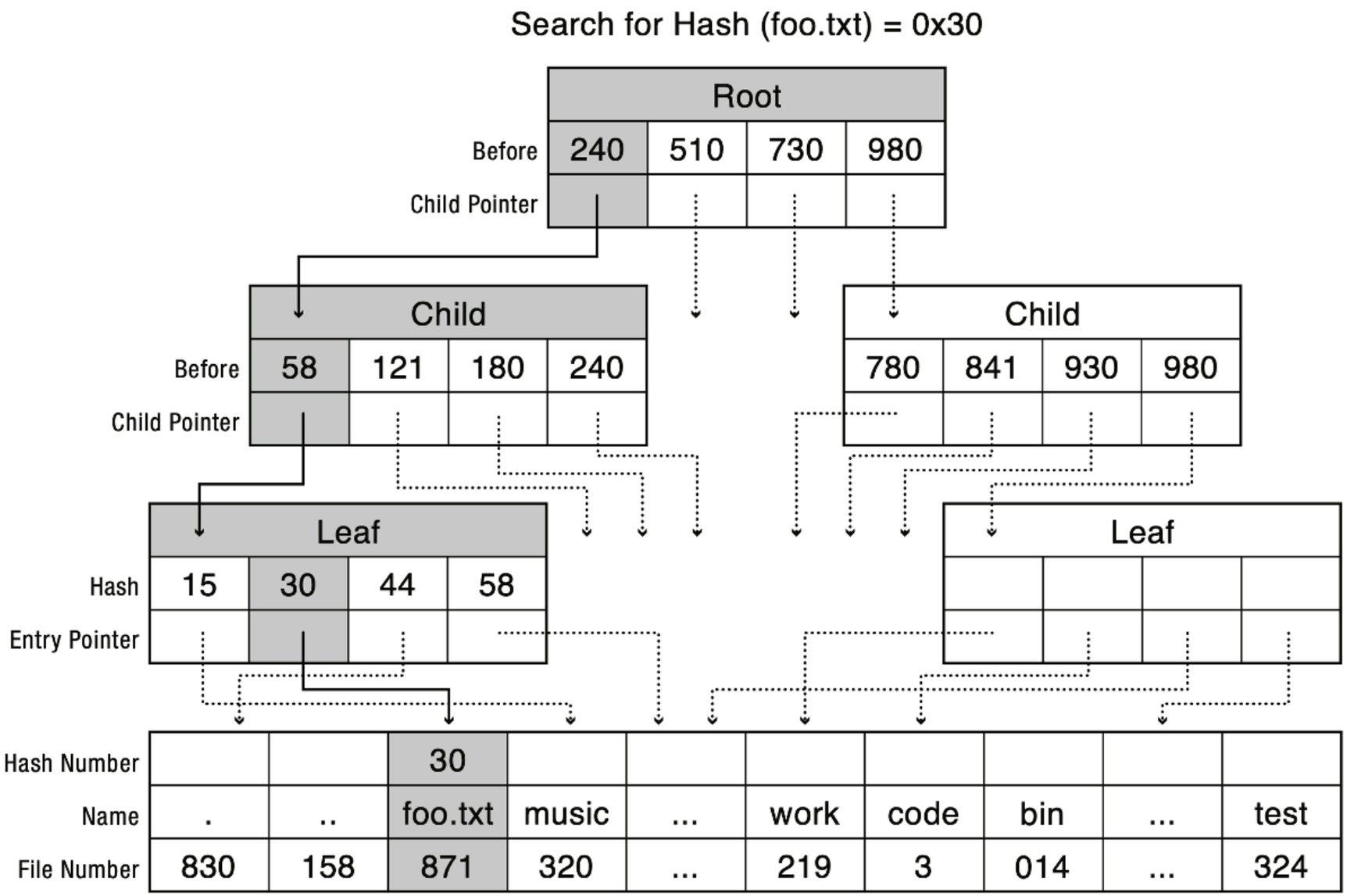


the ext2 example

Unix 文件系统

Directory Implementation

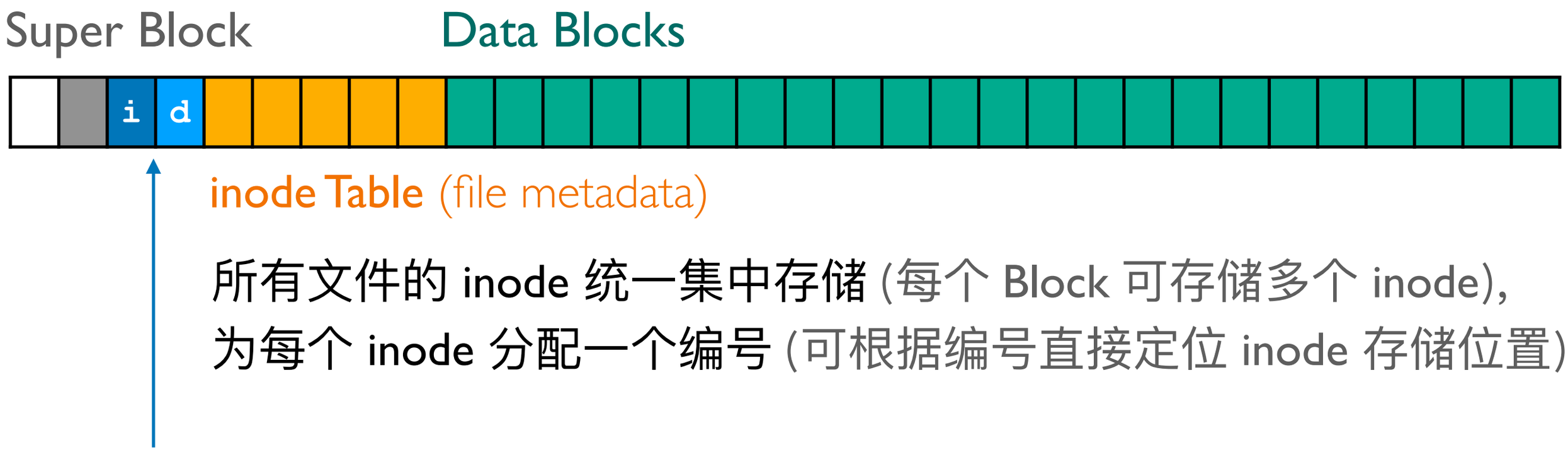
为了加速大目录文件的目录项查找速度，可进一步建立 ⟨file name, inode number⟩ 的某种索引结构 (例如使用 B+ tree)



Unix 文件系统

File System Layout

Unix 文件系统的磁盘布局：结合其文件 (by Direct/Indirect Pointers in inode structure) 和目录 (Save File Metadata in inode) 的实现方式



Unix 文件系统

File System Layout

在 Unix 文件系统中查找 `/usr/ast/mbox` 的过程 (Resolve File Name)