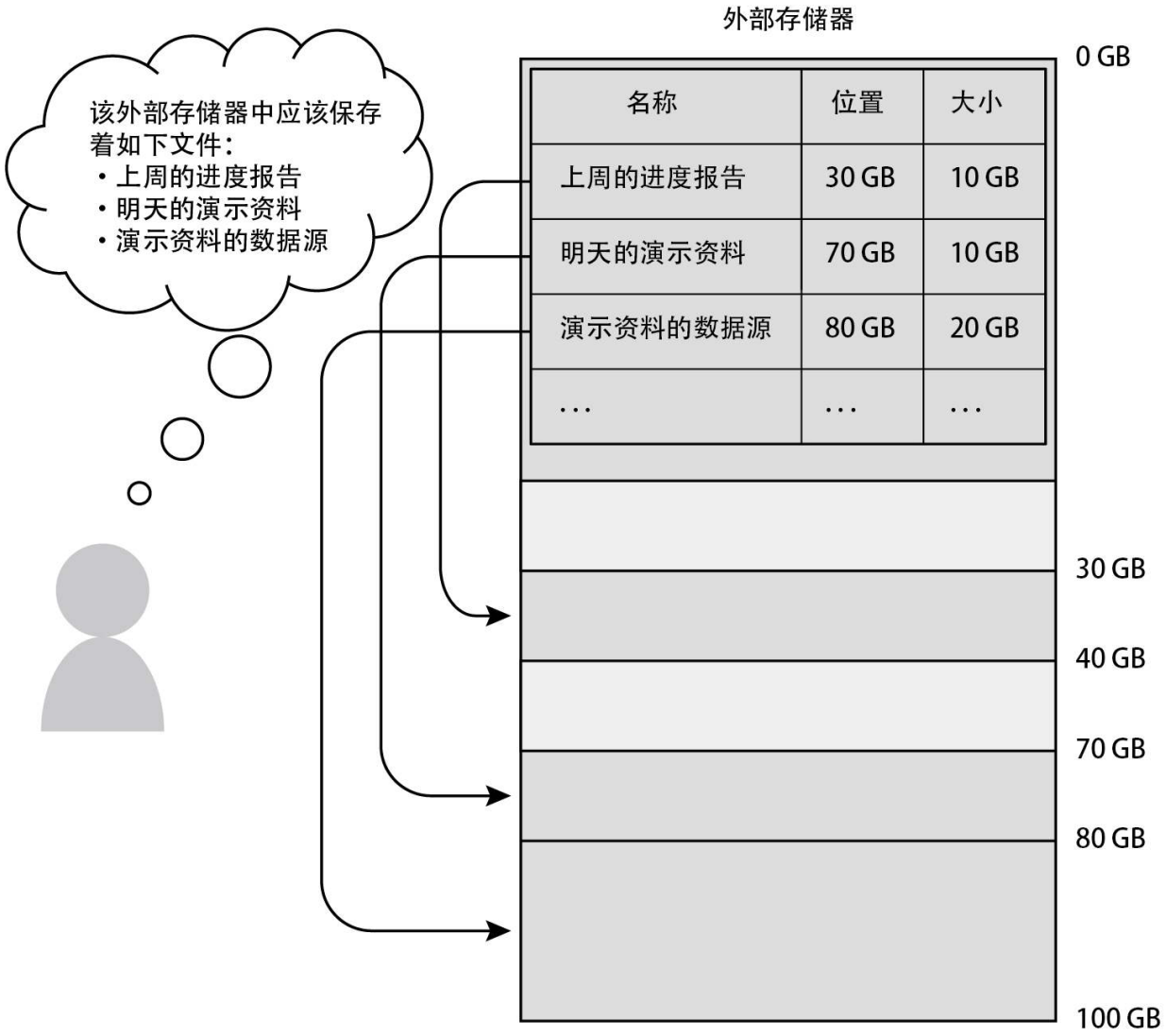

图 7-3　一个简单的文件系统

用户（准确来说是用户使用的进程）只需要通过读取文件的系统调用，指定文件名、文件的偏移量以及文件大小，负责操作文件系统的处理即可找到相关数据并转交给用户，如图 7-4 所示。

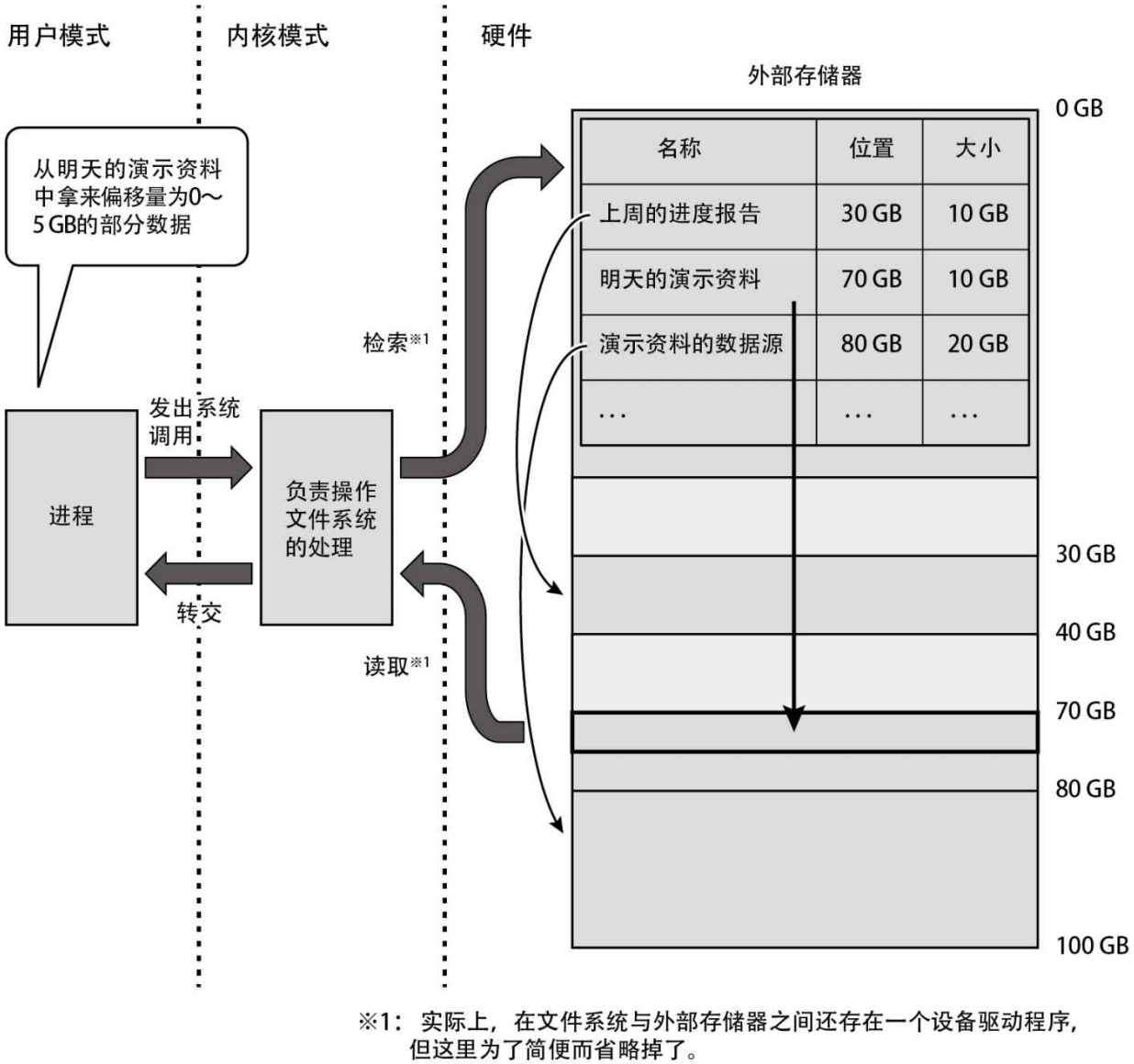

图 7-4　只需提供文件名、文件的偏移量以及文件大小，即可读取符合条件的数据

大家感觉如何？通过这个不限制文件大小的非常简单的文件系统，希望大家可以了解到文件系统的数据结构的基础知识。

7.1 Linux 的文件系统

为了分门别类地整理文件，Linux 的文件系统提供了一种可以收纳其他文件的特殊文件，这种文件称为**目录**。我们不仅能把文件放到目录中，甚至还能把别的目录放到目录中。不同目录下的多个文件可以取相同的名称。目录的存在使得 Linux 的文件系统呈现树状结构，如图 7-5 所示。

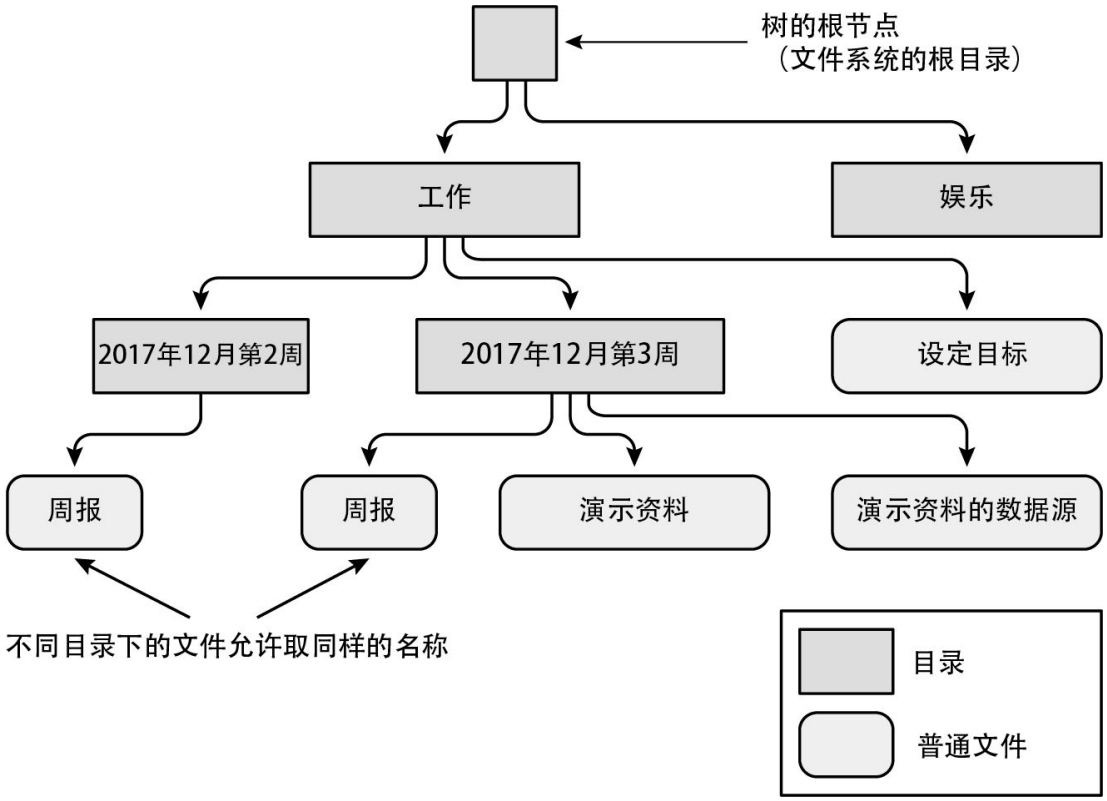

图 7-5 树状结构的文件系统

Linux 能使用的文件系统不止一种，ext4、XFS、Btrfs 等不同的文件系统可以共存于 Linux 上。这些文件系统在外部存储器中的数据结构以及用于处理数据的程序各不相同。各种文件系统所支持的文件大小、文件系统本身的大小以及各种文件操作（创建、删除与读写文件等）的速度也不相同。

但是，不管使用哪种文件系统，面向用户的访问接口都统一为下面这些系统调用。

- 创建与删除文件：**create()**、**unlink()**
- 打开与关闭文件：**open()**、**close()**
- 从已打开的文件中读取数据：**read()**
- 往已打开的文件中写入数据：**write()**
- 将已打开的文件移动到指定位置：**lseek()**
- 除了以上这些操作以外的依赖于文件系统的特殊处理：**ioctl()**

在请求这些系统调用时，将按照下列流程读取文件中的数据。

① 执行内核中的全部文件系统通用的处理，并判断作为操作对象的文件保存在哪个文件系统上。

② 调用文件系统专有的处理，并执行与请求的系统调用对应的处理。

③ 在读写数据时，调用设备驱动程序执行操作。

④ 由设备驱动程序执行数据的读写操作。

以上流程如图 7-6 所示。

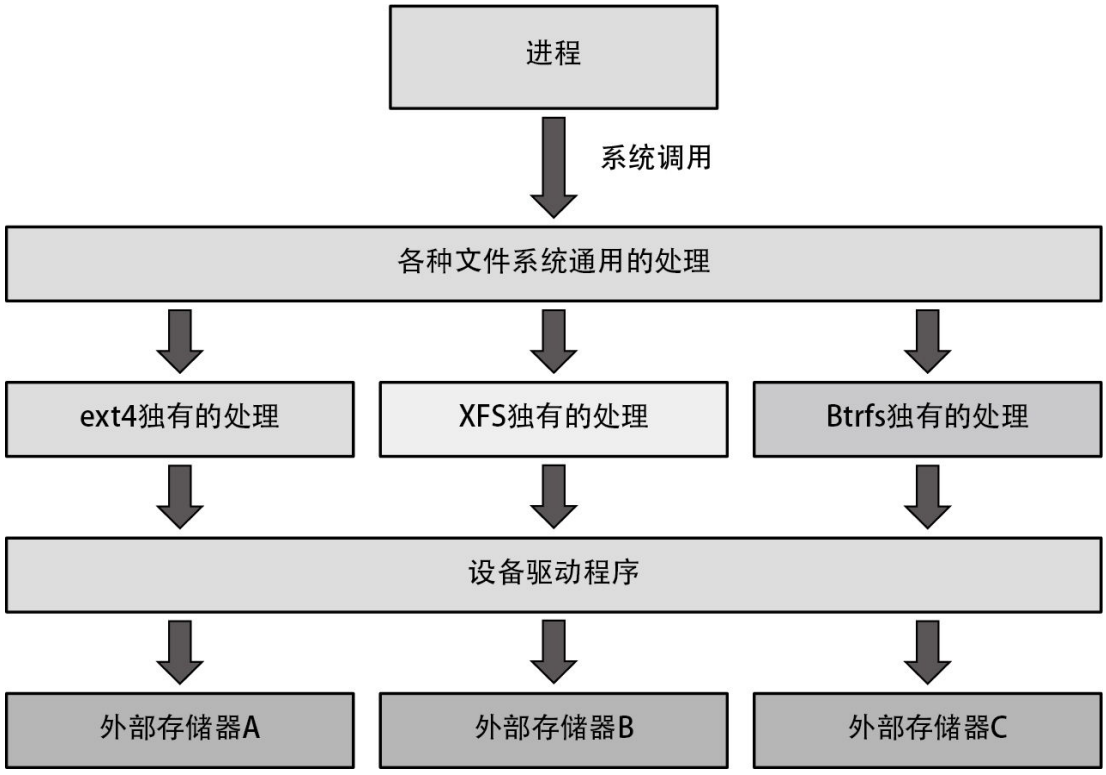

图 7-6　无关文件系统的种类，可以通过统一的接口进行访问

与前几章中的例子一样，不管是使用哪种编程语言编写的程序，在操作文件时，程序底层请求的都是上面列出的这些系统调用。

7.2 数据与元数据

文件系统上存在两种数据类型，分别是**数据**与**元数据**。

- **数据：用户创建的文档、图片、视频和程序等数据内容**
- **元数据：文件的名称、文件在外部存储器中的位置和文件大小等辅助信息**

另外，元数据分为以下几种。

- **种类：用于判断文件是保存数据的普通文件，还是目录或其他类型的文件的信息** [2]
- **时间信息：包括文件的创建时间、最后一次访问的时间，以及最后一次修改的时间**
- **权限信息：表明该文件允许哪些用户访问**

> [2]关于文件有哪些种类，我们将在后文详细说明。

顺便一提，通过 df 命令得到的文件系统所用的存储空间[3]，不但包括大家在文件系统上创建的所有文件所占用的空间，还包括所有元数据所占用的空间，这一点需要特别注意。

> [3]在 Btrfs 中，df 命令并不会返回正确的结果，需要使用 btrfs filesystem df 命令。

```
$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
...
devsdc1        95990964     61104  91030668   1% mnt
$ sudo su
# cd mnt
# for ((i=0;i<100000;i++)) ; do mkdir $i ; done
                                ←创建目录（目录的数据类型为元数据）
# df
Filesystem     1K-blocks      Used Available Use% Mounted on
...
devsdc1        95990964    463180  90628592   1% /mnt  #
```

可以看到，整体来说已用空间非常小，在创建大量小文件的系统上，与文件的总占用量相比，原来那部分存储空间使用量意料之外的小。这种情况通常是因为元数据占用的存储空间变大了。

7.3 容量限制

当系统被同时用于多种用途时，假如针对某一用途不限制文件系统的容量，就有可能导致其他用途所需的存储容量不足。特别是在 root 权限下运行的进程，当系统管理相关的处理所需的容量不足时，系统将无法正常运行，如图 7-7 所示。

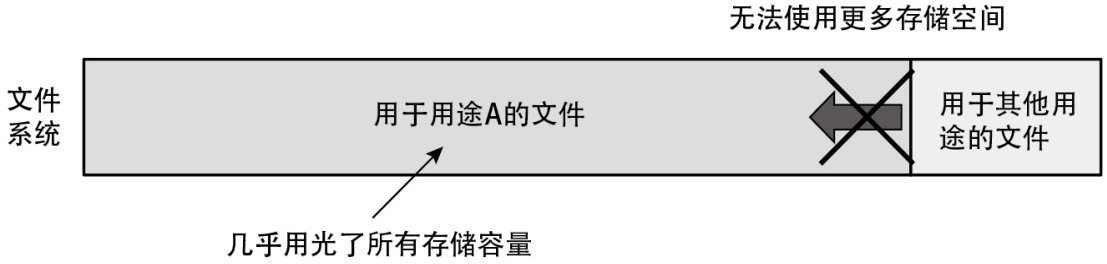

图 7-7 当文件系统容量不足时，系统无法正常运行

为了避免这样的情况出现，可以通过**磁盘配额**（quota）功能来限制各种用途的文件系统的容量，如图 7-8 所示。

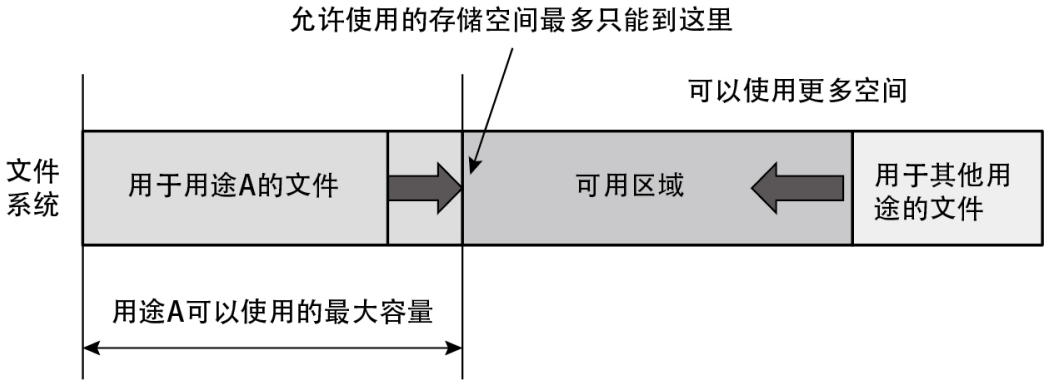

图 7-8 磁盘配额

磁盘配额有以下几种类型。

- **用户配额：限制作为文件所有者的用户的可用容量。例如防止某个用户用光 /home 目录的存储空间。ext4 与 XFS 上可以设置用户配额**

- 目录配额：限制特定目录的可用容量。例如限制项目成员共用的项目目录的可用容量。ext4 与 XFS 上可以设置目录配额
- 子卷配额：限制文件系统内名为子卷的单元的可用容量。大致上与目录配额的使用方式相同。Btrfs 上可以设置子卷配额

7.4　文件系统不一致

只要使用系统，那么文件系统的内容就可能不一致。比如，在对外部存储器读写文件系统的数据时被强制切断电源的情况，就是一个典型的例子。

接下来，我们看一下文件系统不一致具体是什么样的状态。下面以移动一个目录到别的目录下的情景为例来说明（图 7-9）。

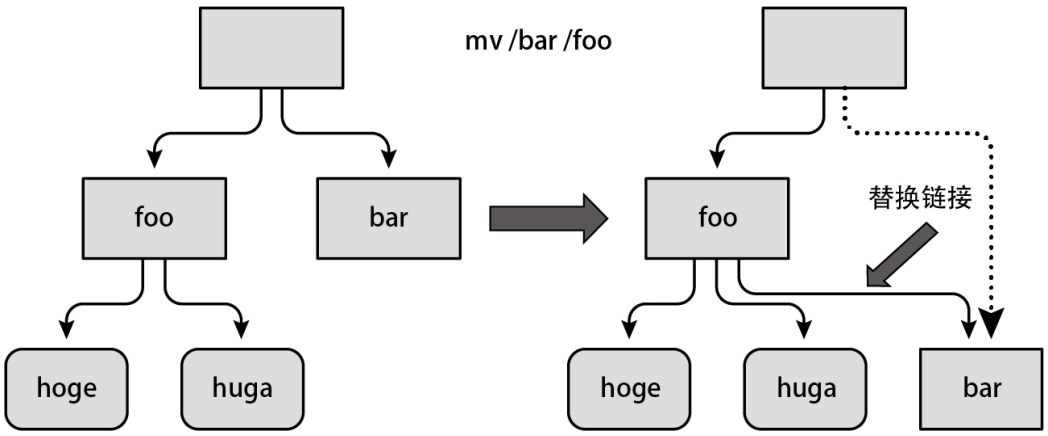

图 7-9　移动目录

这一处理的具体流程如图 7-10 所示。

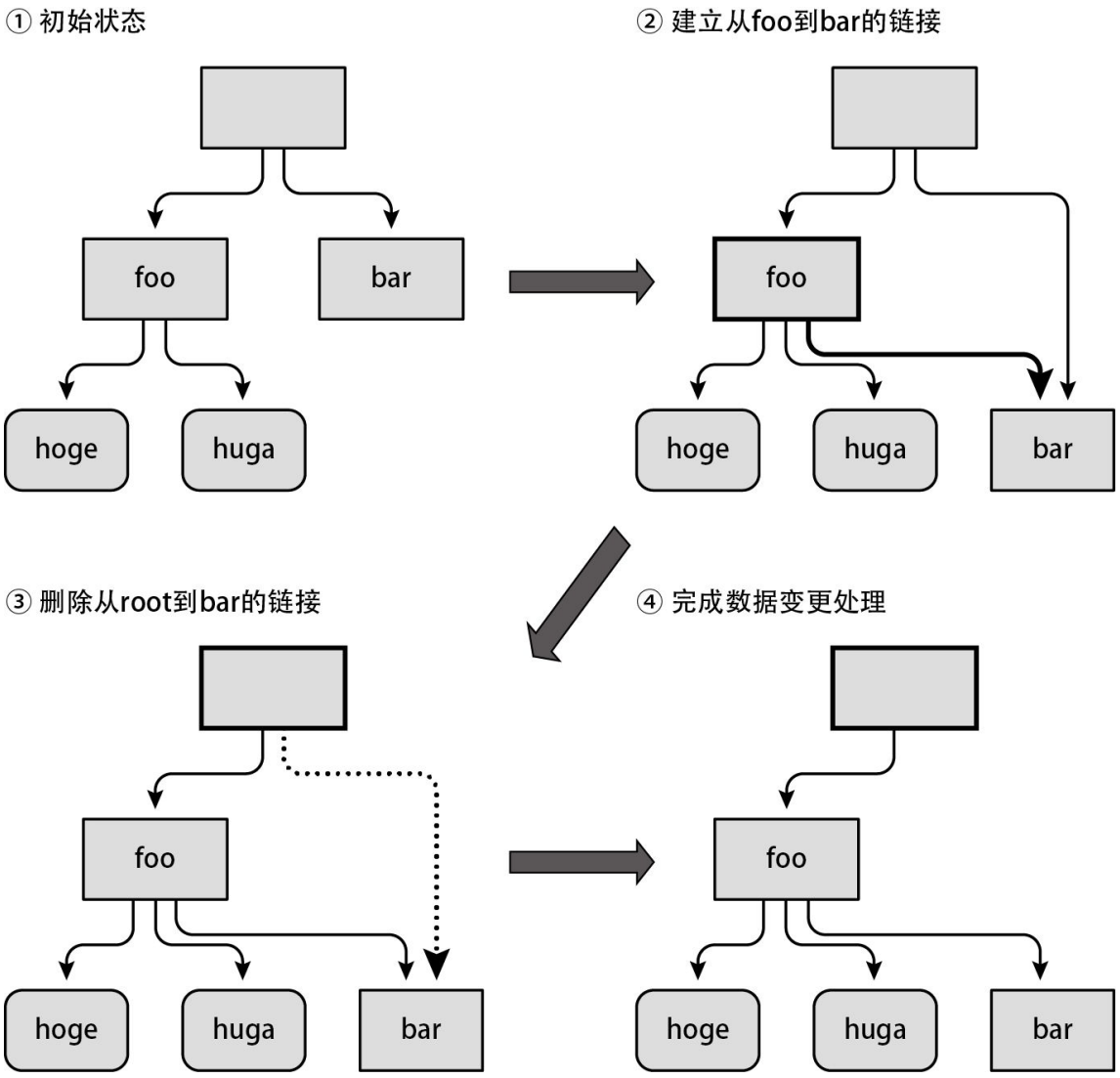

图 7-10　移动目录的处理流程

这一连串的操作是不可分割的，称为**原子操作**。这里，由于对外部存储器的读写操作一次只能执行一步，所以如果在第一次写入操作（foo 文件的数据更新）完成后、第二次写入操作（root 的数据更新）开始前中断处理，文件系统就会变成不一致的状态，如图 7-11 所示。

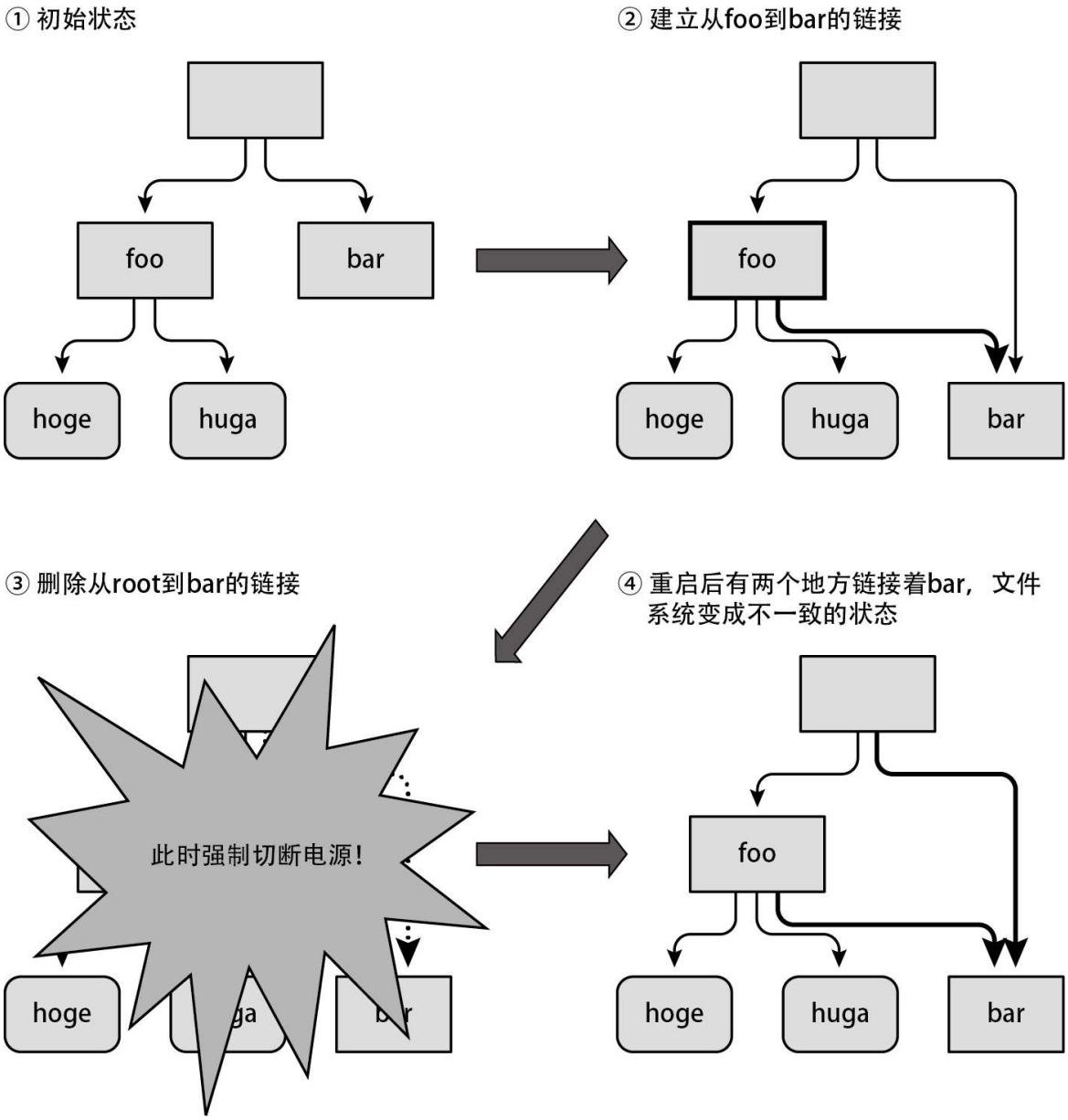

图 7-11　文件系统不一致

只要发生过这种不一致的状况，迟早会被文件系统检查出来。如果在挂载时检测到不一致，就会导致文件系统无法被挂载；如果在访问过程中检测到不一致，则可能会以只读模式重新挂载该文件系统，在最坏的情况下甚至可能导致系统出错。

目前，防止文件系统不一致的技术有很多，常用的是**日志**（journaling）与**写时复制**。ext4 与 XFS 利用的是日志，而 Btrfs 利用的是写时复制。

接下来将逐一说明这两种技术。

7.5 日志

日志功能在文件系统中提供了一个名为**日志区域**的特殊区域。日志区域是用户无法识别的元数据。文件系统的更新按照以下步骤进行。

① 把更新所需的原子操作的概要暂时写入日志区域，这里的“概要”就称为日志。

②基于日志区域中的内容，进行文件系统的更新。

上述步骤如图 7-12 所示。

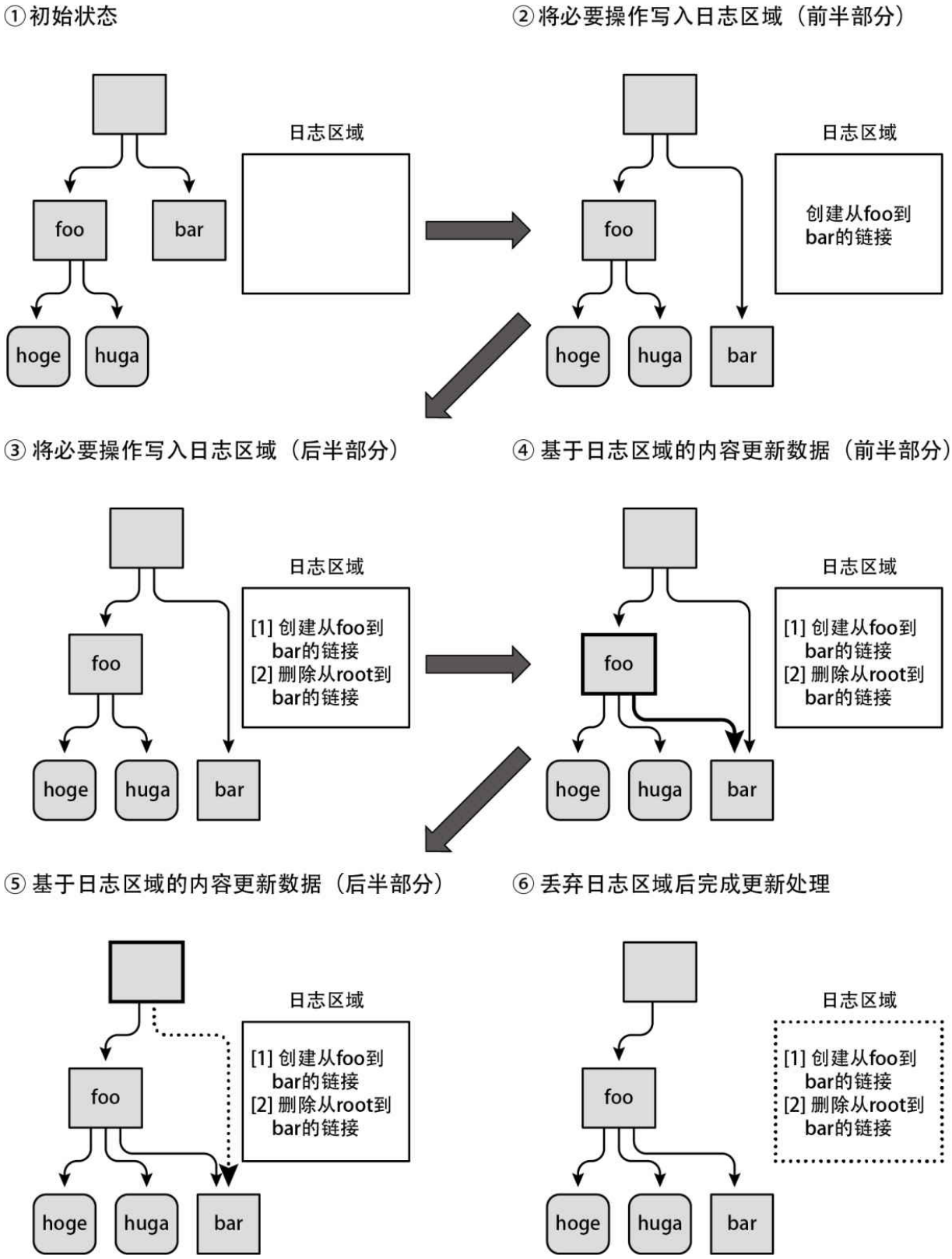

图 7-12　日志方式的文件系统更新处理