操作系统

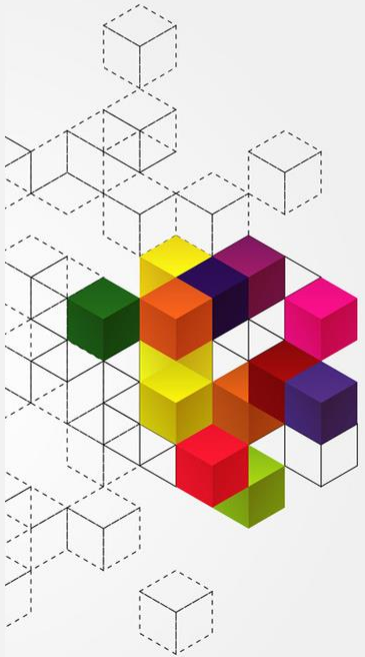
Operating system

胡燕

大连理工大学

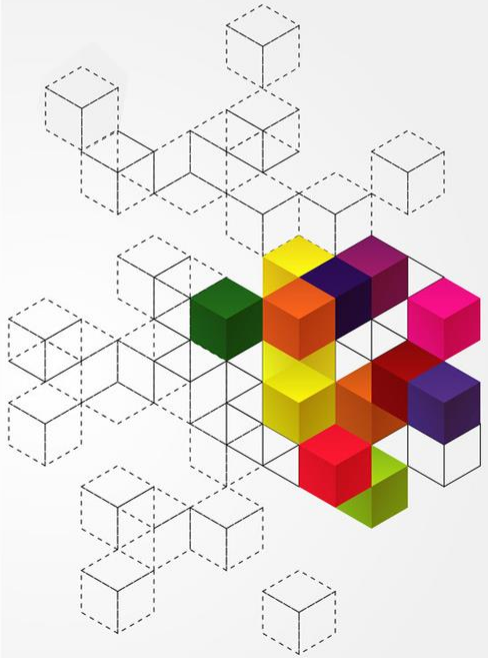
文件系统接口：回顾　　文件编程API的使用

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    FILE *fp，*fp2;
    float num=11.22334;
    fp=fopen(  “/tmp/recent/test.txt","wb");
    if (NULL==fp)    {
        printf ("cannt open\n");
        exit (1);
    }
    fwrite(&num,sizeof(float),1,fp);//此时文本里面的是地址
    fclose(fp);//这里注意 每次对文本操作一次都要进行关闭
    fp2=fopen(" /tmp/recent/test.txt","rb");
    float num2;
    if (NULL==fp)    {
        printf ("cannt open\n");
        exit (1);
    }
    fread(&num2,sizeof(float),1,fp2);
    printf ("%lf\n",num2);
    fclose(fp2);
    system("pause");
    return 0;
}
```



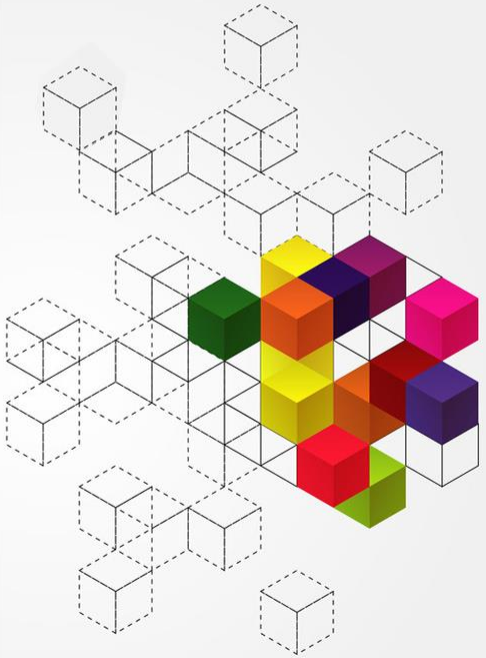
文件系统接口：回顾　　文件编程API的使用

```
/* fseek example */
#include <stdio.h>
int main () {
    FILE * pFile;
    pFile = fopen ( "example.txt" , "wb" );
    fputs ( "This is an apple." , pFile );
    fseek ( pFile , 9 , SEEK_SET );
    fputs ( " sam" , pFile );
    fclose ( pFile );
    return 0;
}
```



文件系统接口：回顾　　UNIX下文件系统调用的使用

```c
#include <fcntl.h>
#include <stdio.h>

int main(void){
        int fd,byteNum,result;
        char wbuf[10] = "123456789";
        char rbuf[10];
        if((fd = open("./a.txt", O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR|S_IXUSR))<0){
                perror("open");
                return -1;
        }

        if((byteNum = write(fd, wbuf, 10))<0){
                perror("write");
                return -1;
        }

        if((result = lseek(fd, 4, SEEK_SET))<0){
                perror("lseek");
                return -1;
        }

        if((byteNum = read(fd, rbuf, 10)) < 0){
                perror("read");
                return -1;
        }

        printf("read content:%s\n",rbuf);

        close(fd);
        return 0;
}
```

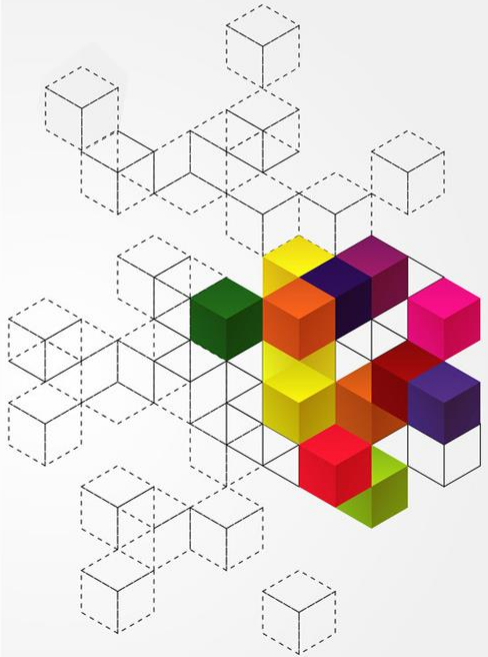


文件系统接口：回顾　　文件的逻辑结构

文件的两种结构：对应两种不同的视角

（1）用户视角：文件的逻辑结构

用户“思维”中的文件结构，或称逻辑文件。
侧重点在为用户提供一种逻辑清晰、使用简便的逻辑文件形式。用户按照这种形式去存储、检索和加工对应文件中的信息



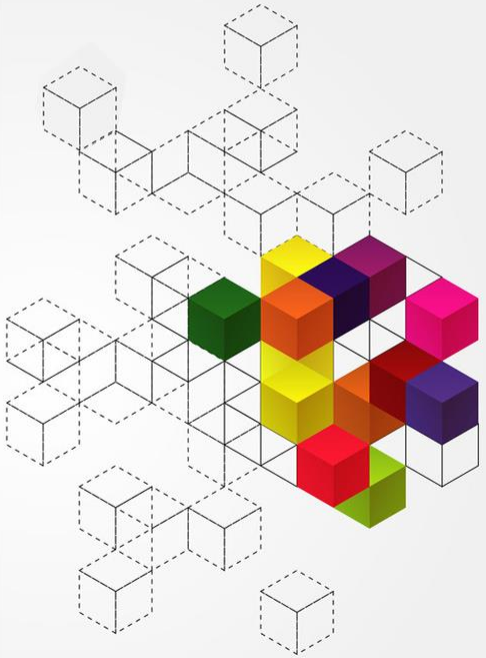
文件系统接口：回顾　　文件的逻辑结构

文件的两种结构：对应两种不同的视角

（1）用户视角：文件的逻辑结构

（2） 实现视角：文件的物理结构

研究驻留在设备“介质”中的实际文件，或称物理文件。它的侧重点是选择一些工作性能良好、设备利用率高的物理文件形式。系统按照这种形式在外存中存储信息，并控制信息在外存和内存之间的传输。



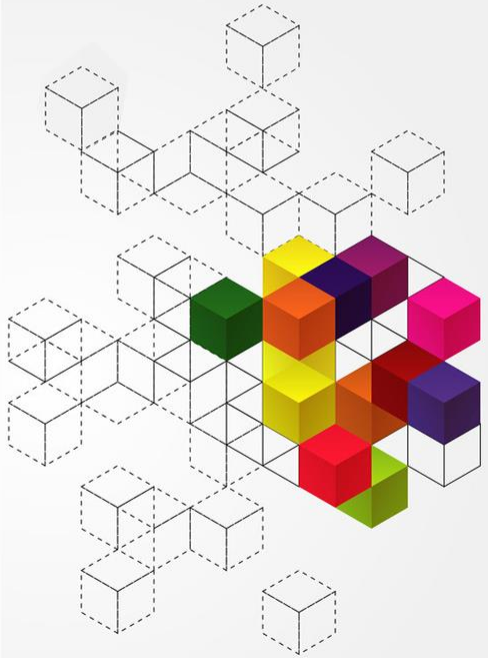
文件系统接口：回顾　　文件的访问模式

文件的集中访问模式

（1）顺序访问

（2）随机访问

（3）索引访问

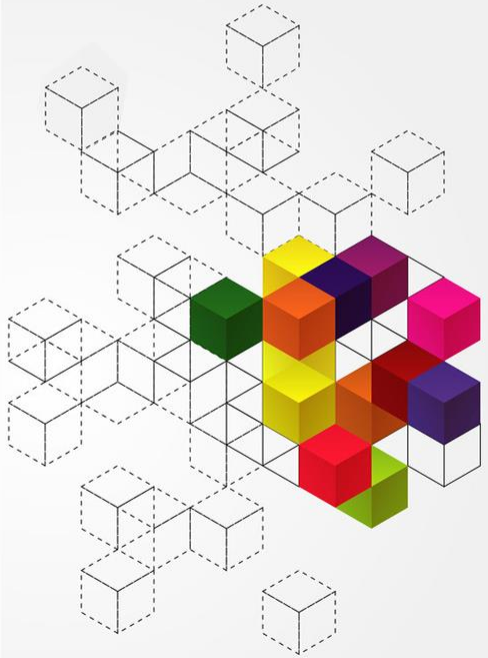


内容纲要

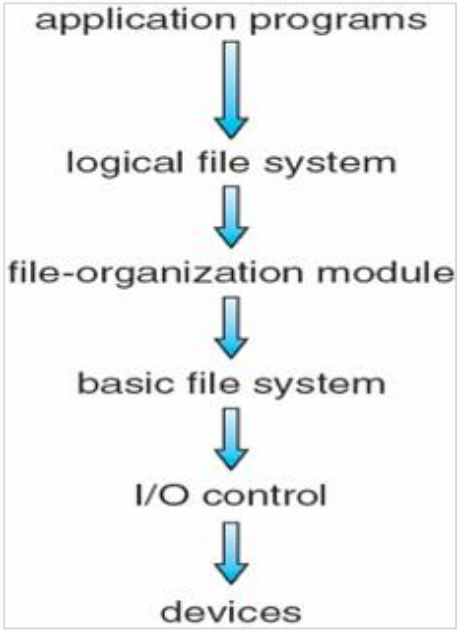
11.1 文件系统结构

一、 文件系统分层设计

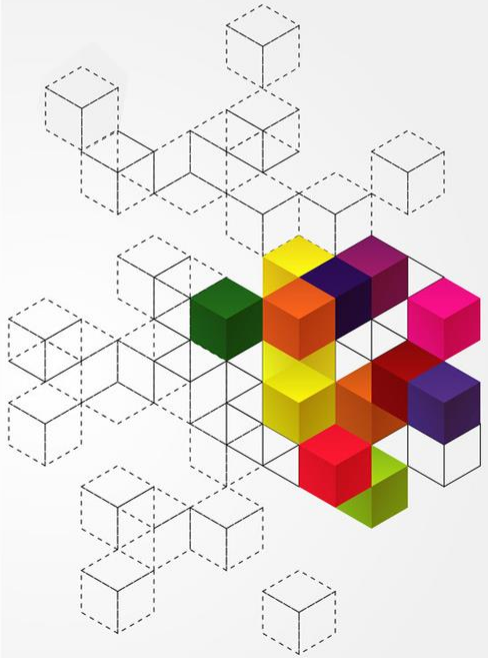
二、 虚拟文件系统(VFS)



一、文件系统分层设计



- 对文件系统进行设计与实现时，分层设计思想是主流
- 典型层次：
  - ① 应用程序接口层
  - ② 逻辑文件系统层
  - ③ 文件组织模块
  - ④ 基础文件系统
  - ⑤ I/O控制



一、文件系统分层设计

Application Programs

文件系统的应用程序层，为应用程序提供文件操作接口（open/close、read/write等操作）

READ(2) Linux Programmer's Manual READ(2)

NAME top

read - read from a file descriptor

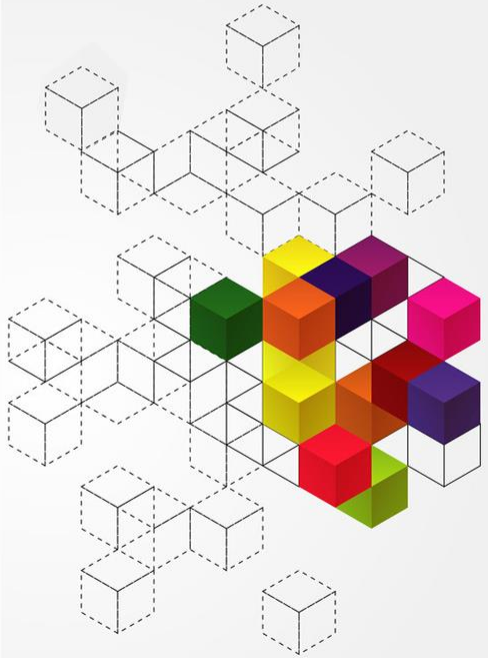
SYNOPSIS top

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

例：

fh = open(filename, access)

read(fh, buf, size)



一、文件系统分层设计

Application Programs

↓

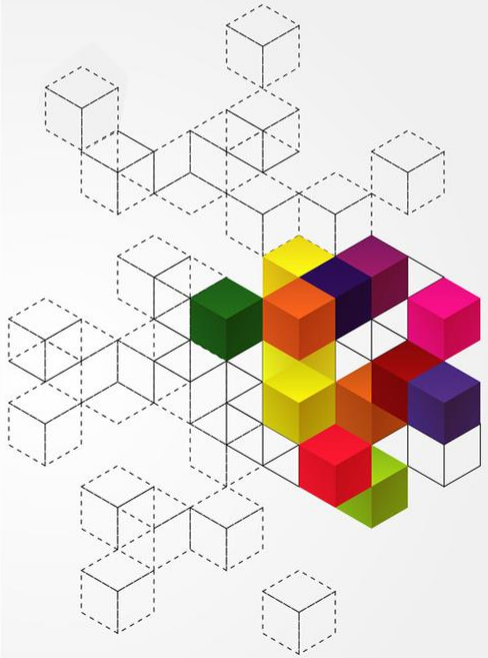
Logical File System

逻辑文件系统层

管理目录与文件相关元数据，维持文件系统结构

主要职能：

- 管理目录结构
- 维护文件名到文件控制块的映射
- 对文件实施访问控制
- 处理文件关联属性（UID, GID, 创建时间等）



一、文件系统分层设计

Application Programs

↓

Logical File System

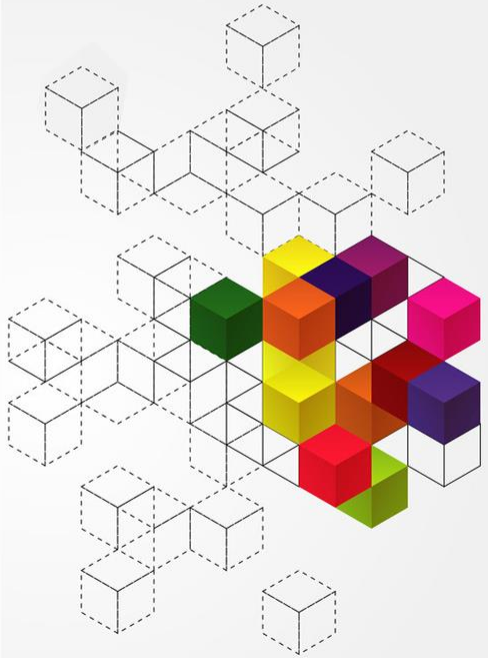
↓

File Organization Module

文件组织模块

维护逻辑块到物理块的映射，文件的物理结构，空闲空间管理

- Knows about files and their logical blocks (1..N)
- Files are organized in blocks of 32 bytes to 4K bytes
- Translates logical blocks to physical
- Knows location of file, file allocation type
- Free space management



一、文件系统分层设计

Application Programs

↓

Logical File System

↓

File Organization Module

↓

Basic File System

块接口示例：
read(123), write(123，blkdata)

基础文件系统

提供以块为单位的文件系统操作接口

- Issues commands to the device driver
- Identify each physical block by a disk address (e.g. disk 2, cylinder 34, track 2, sector 11)

一、文件系统分层设计

Application Programs

↓

Logical File System

↓

File Organization Module

↓

Basic File System

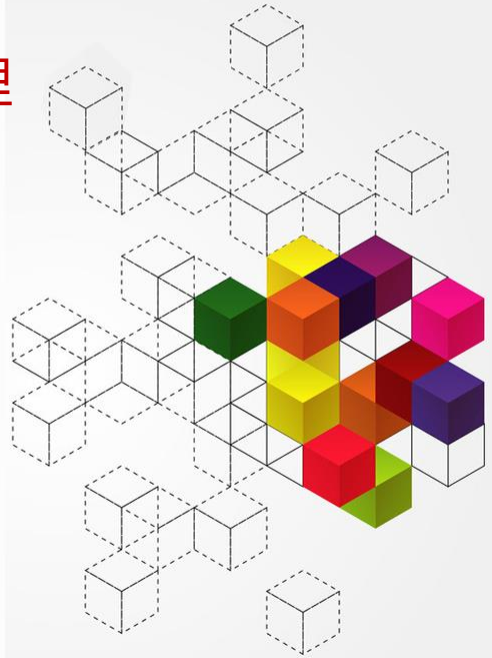
↓

I/O Control

I/O控制软件层

提供设备驱动和设备中断处理例程

- transfer information between memory and disk
- A device driver translates commands such as “get me block 111” into hardware specific ISA used by hardware controller, This is accomplished by writing specific bits into IO registers



一、文件系统分层设计

read(fh, buf, size)

↓

manages metadata (fp, access)

↓

logical ←→ physical mapping

↓

read d1, c73, t5, s10

↓

store read register 145, 5

↓

seagate disk

app program (API)

↓

logical file system

↓

file-organization module

↓

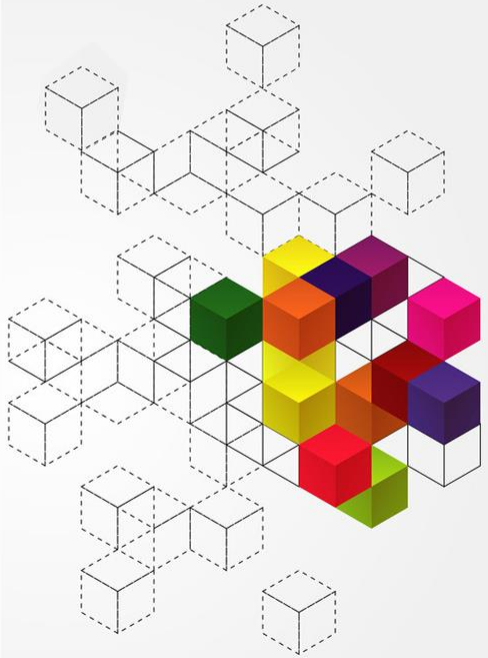
basic file system

↓

I/O control

↓

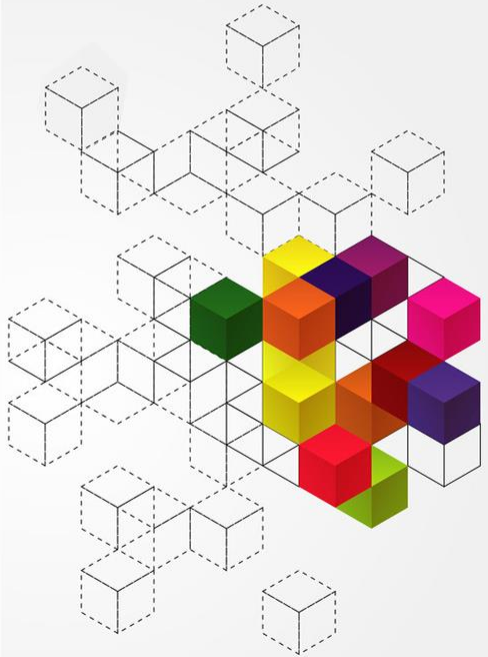
devices



一、文件系统分层设计

论分层的必要性

在系统从0到1的阶段，为了让系统快速上线，通常开发团队倾向于不考虑分层。



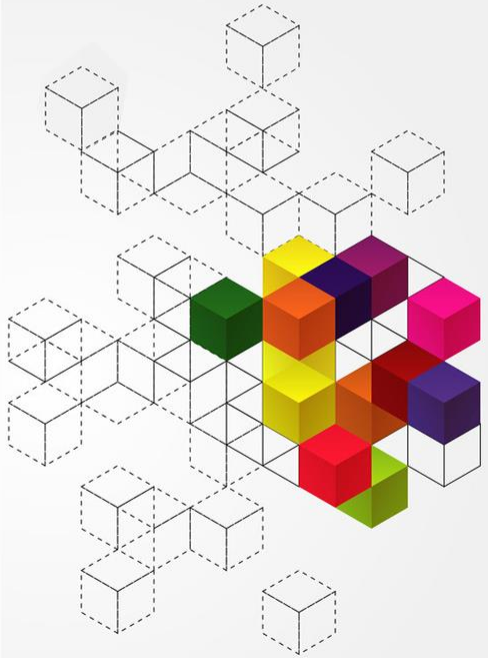
参考资料: 架构分层：我们为什么一定要这么做，https://zhuanlan.zhihu.com/p/196733107

一、文件系统分层设计

论分层的必要性

在系统从0到1的阶段，为了让系统快速上线，通常开发团队倾向于不考虑分层。

随着业务越来越复杂，大量代码纠缠在一起，会出现逻辑不清晰、各模块相互依赖、代码扩展性差、改动一处就牵一发而动全身等问题。

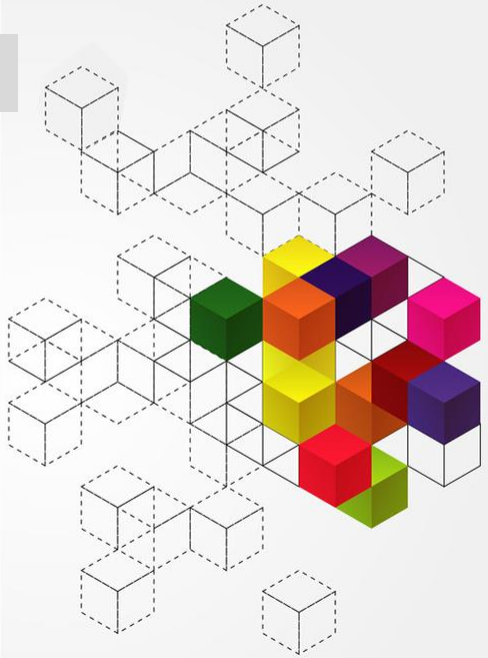


一、文件系统分层设计

论分层的必要性

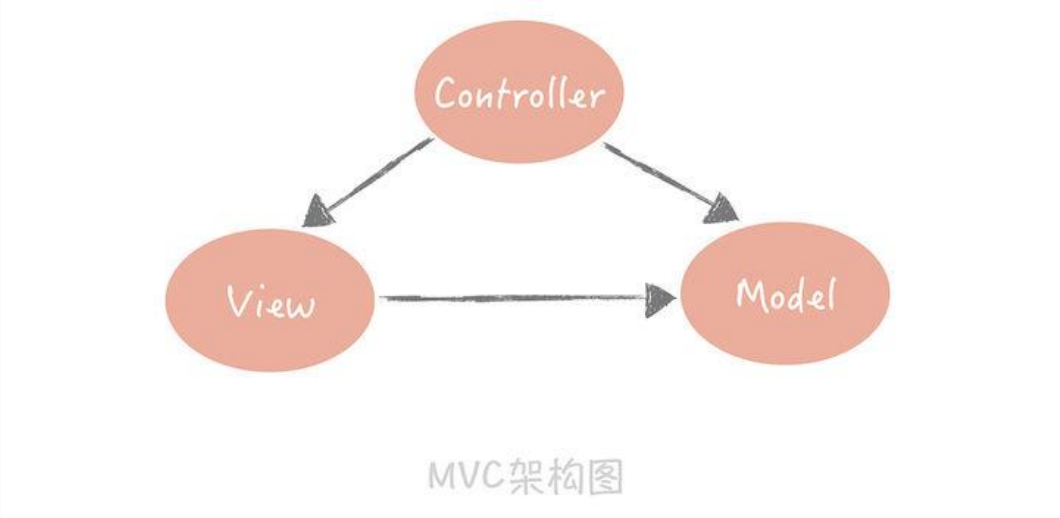
分层的好处

- 简化设计，让不同的人专注做某一层次的事
- 有利于代码复用
- 便于系统进行横向扩展

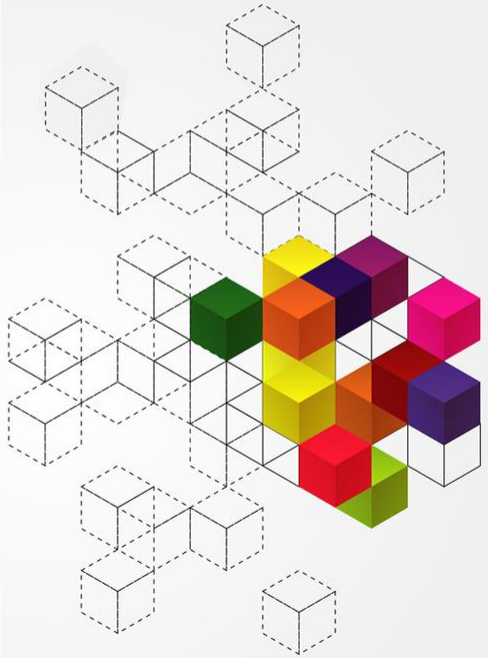


一、文件系统分层设计

分层架构实例

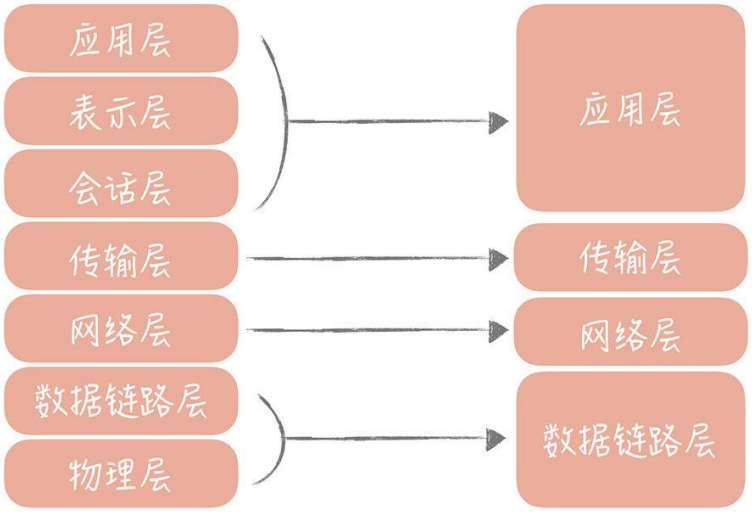


MVC架构图

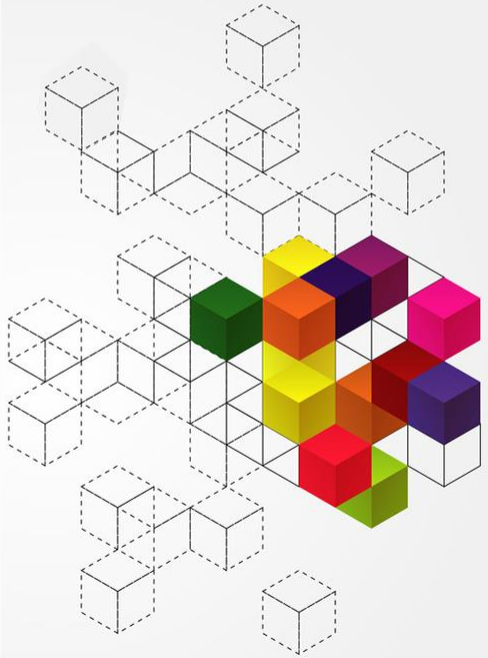


一、文件系统分层设计

分层架构实例

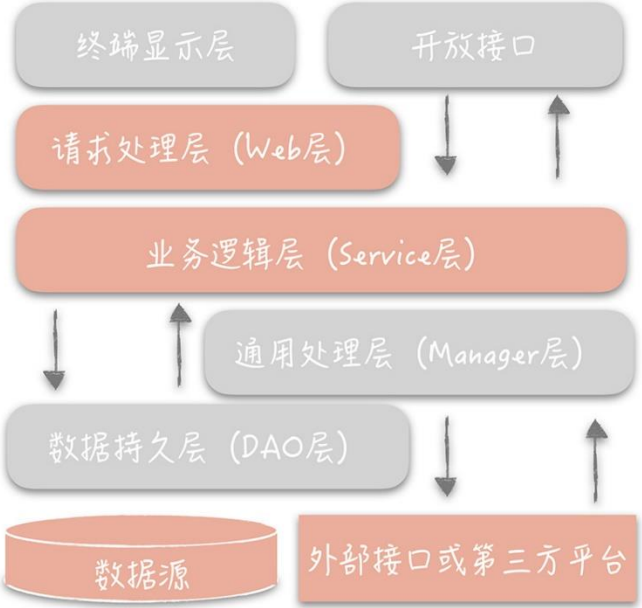


网络分层模型

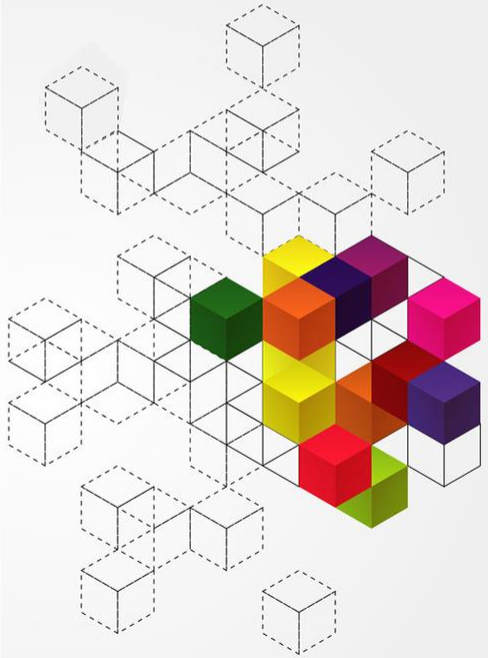


一、文件系统分层设计

分层架构实例

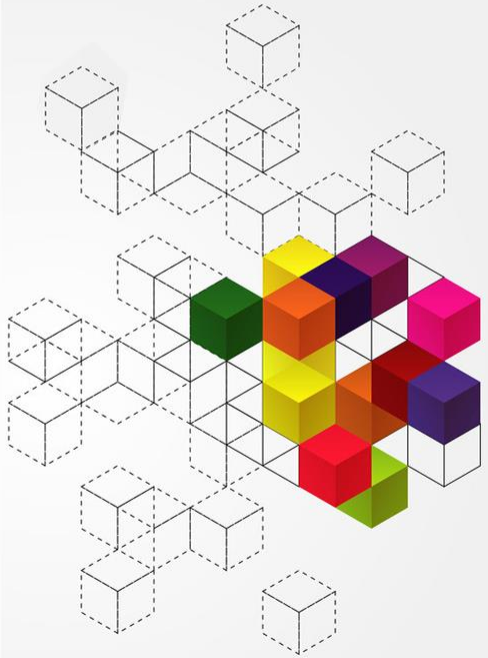


阿里系统分层的规约



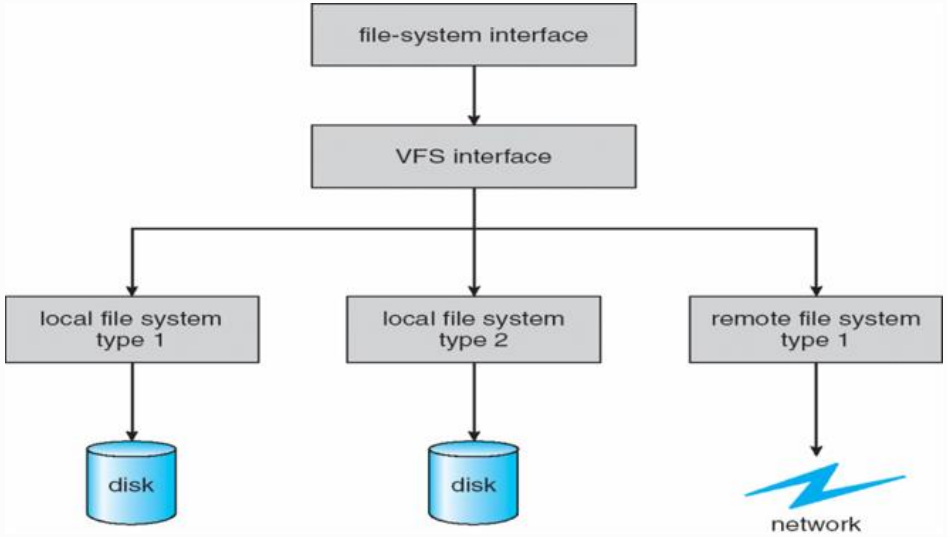
一、文件系统分层设计

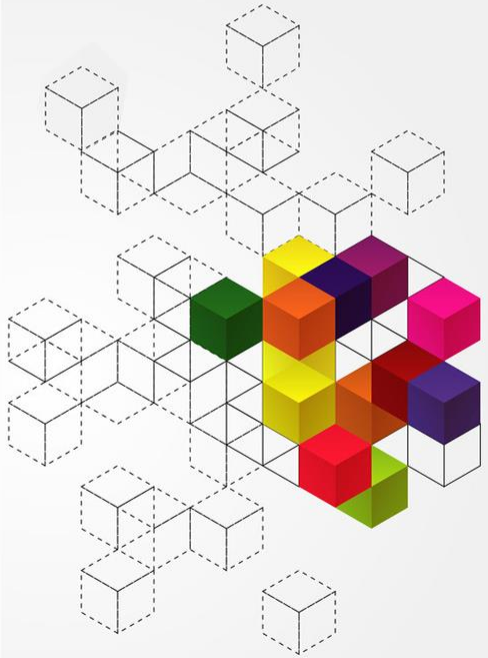
慕课堂讨论1：分层设计是否存在不足?



二、虚拟文件系统

- 虚拟文件系统是物理文件系统之上的一个软件抽象层，其基本目的是为应用访问下层不同类型的物理文件系统提供统一的操作接口





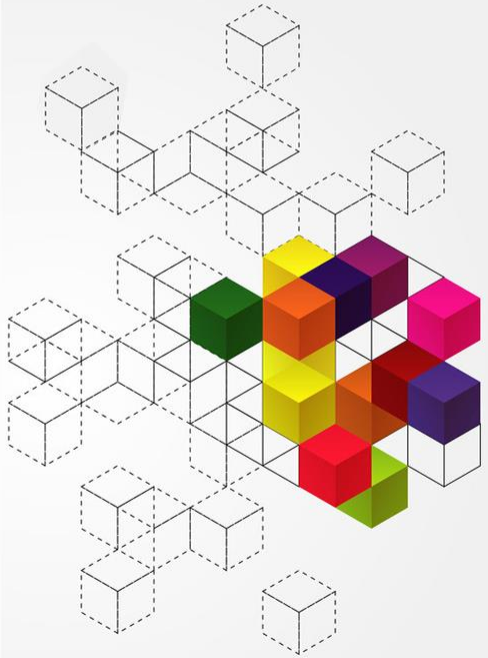
VFS概念，是Sun公司在定义网络文件系统（NFS）时所创造

二、虚拟文件系统

VFS通用文件模型

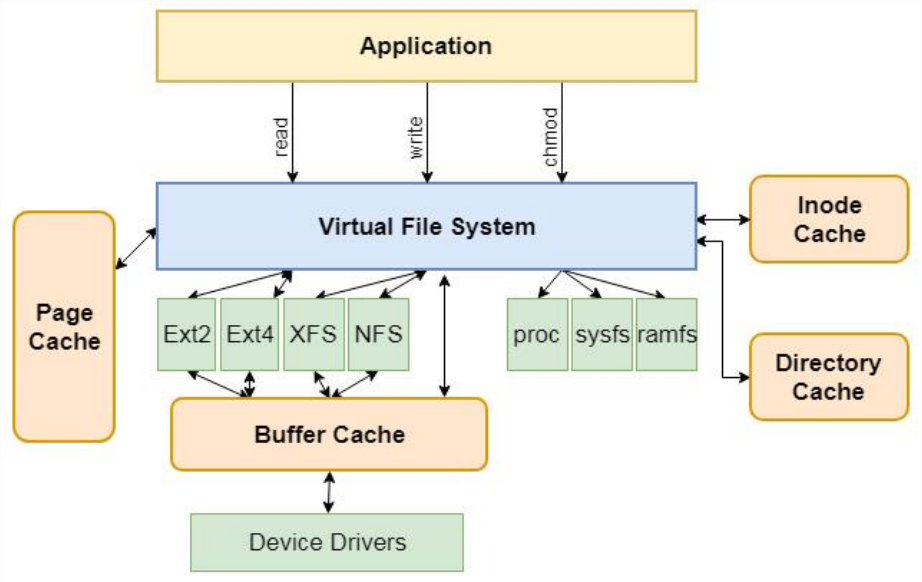
1. 提供一个最小的通用模型，使得这个模型支持的功能是所有文件系统的最小交集
2. 提供一个尽量大的通用模型，使得这个模型包含所有文件系统功能的合集。

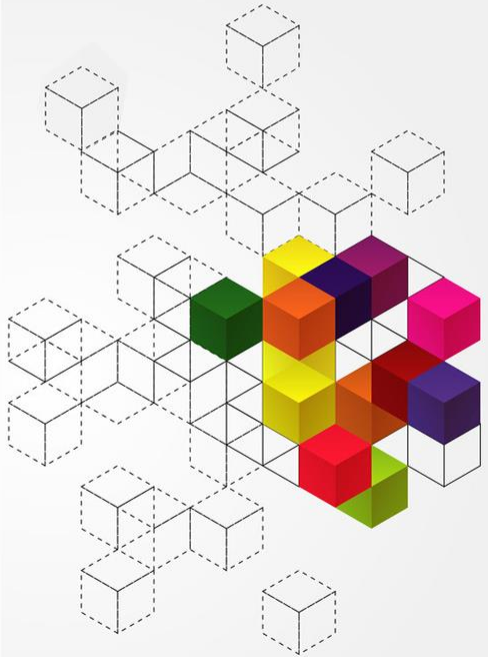
Linux采用第二种策略来实现VFS：
VFS封装了底层文件系统的所有功能和抽象，负责把应用层的请求转发给特定的文件系统。



二、虚拟文件系统

- Linux VFS结构

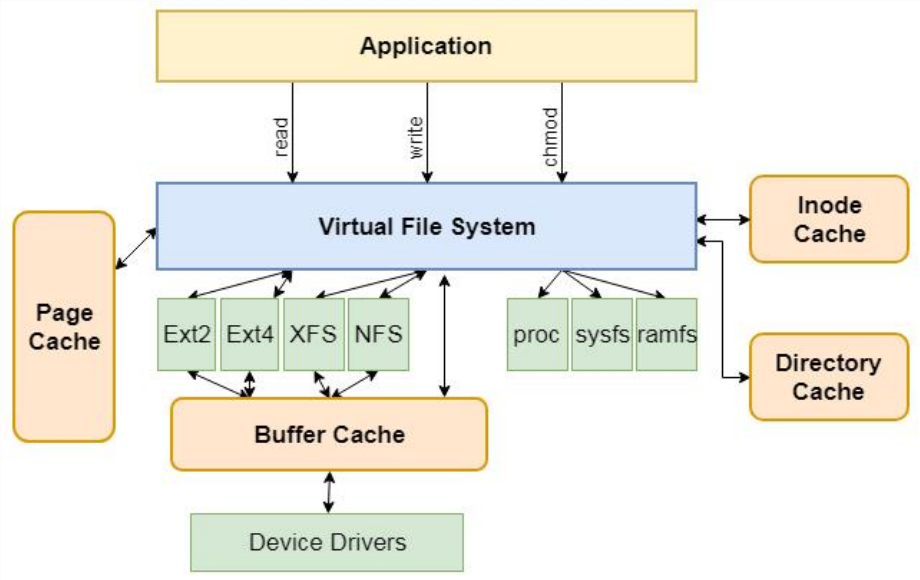


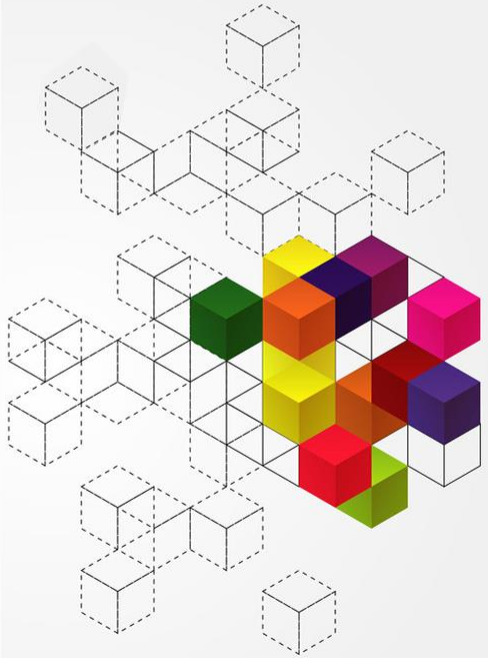


提供一个标准的接口，以便其他操作系统的文件系统可以方便的移植到Linux上

二、虚拟文件系统

- Linux VFS结构

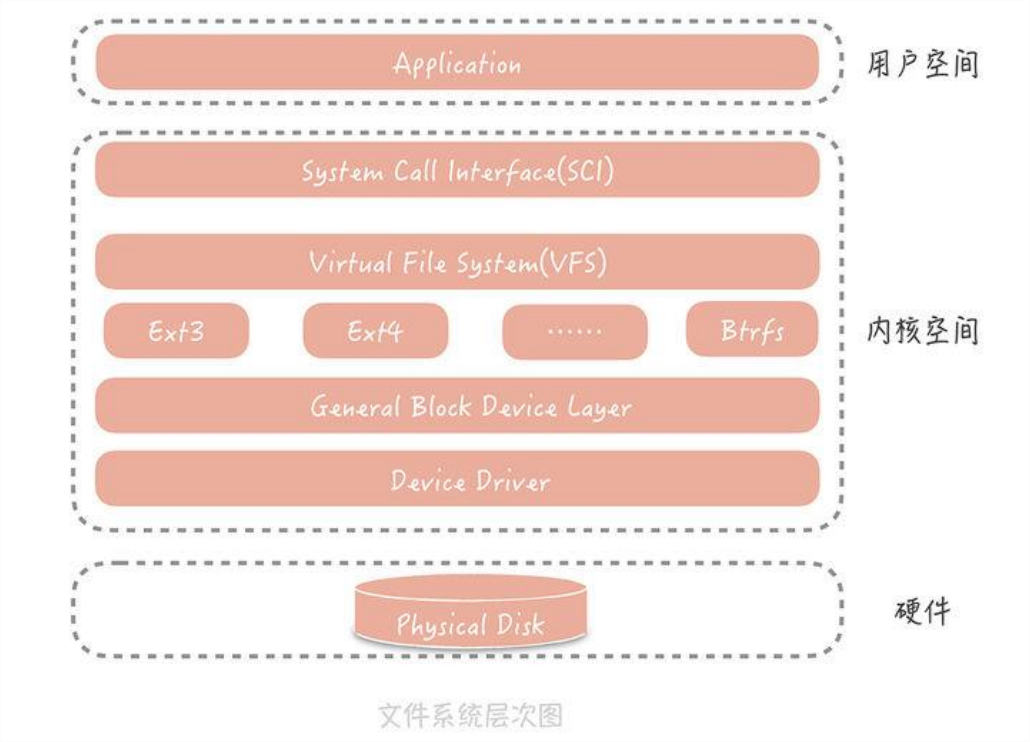




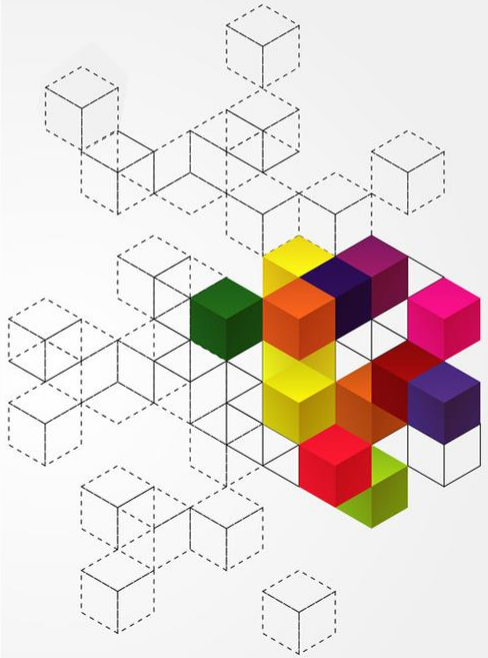
通过一系列高效的管理机制，比如inode cache, dentry cache 以及文件系统的预读等技术,获得高性能

二、虚拟文件系统

- Linux系统下的文件系统层次图

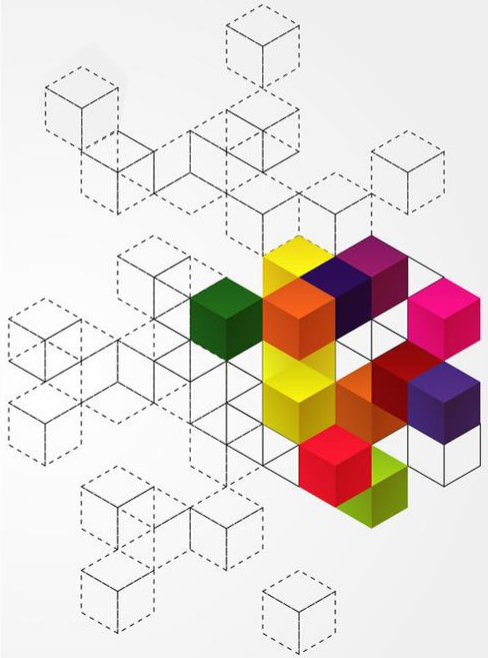


文件系统层次图



本讲小结

- 文件系统分层设计
- 虚拟文件系统



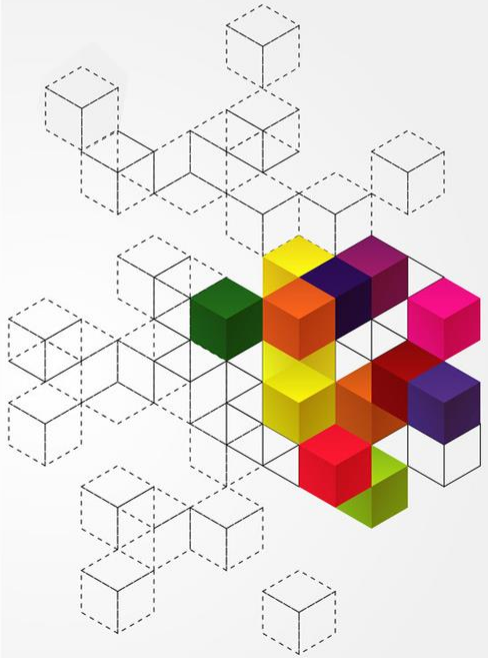
内容纲要

11.2 文件系统实现

一、 文件系统实现考虑

二、 文件系统的磁盘数据结构

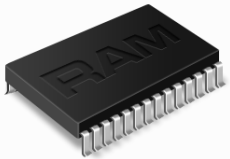
三、 文件系统的内存数据结构



一、文件系统实现考虑

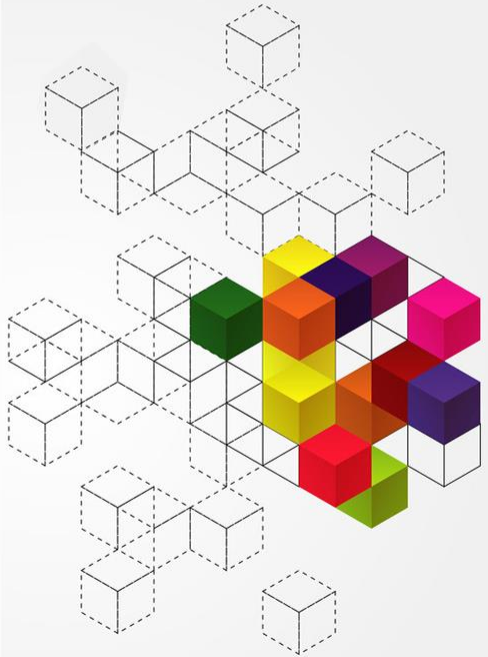
磁盘数据结构

启动控制块(Boot Control Block)
超级块（Superblock)
文件控制块（File Control Block)



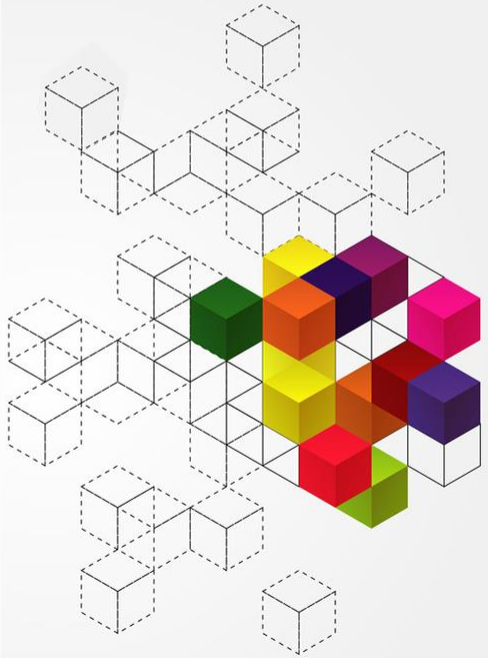
内存数据结构

文件加载表
目录缓存
系统打开文件表
进程打开文件表



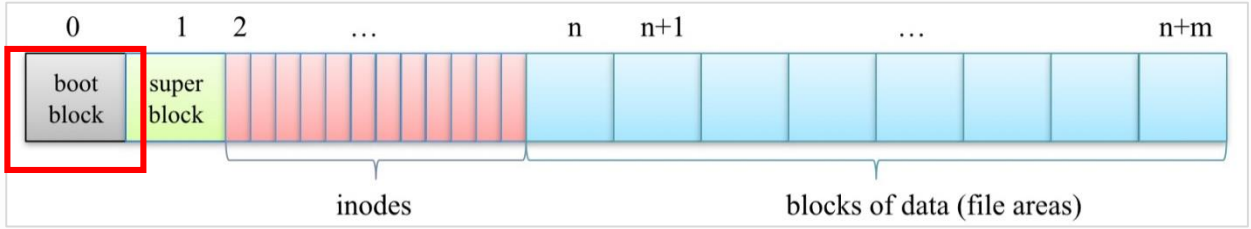
一、文件系统分层设计

慕课堂讨论2：文件系统中为何需要磁盘数据结构，又为何需要内存数据结构?



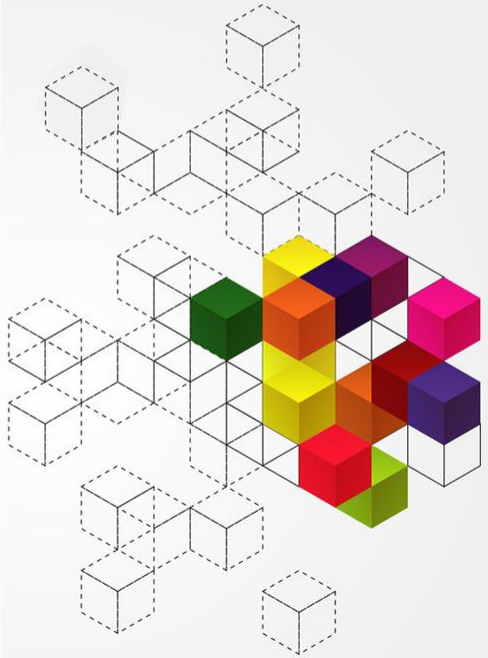
二、文件系统磁盘数据结构

磁盘数据结构逻辑示意图：



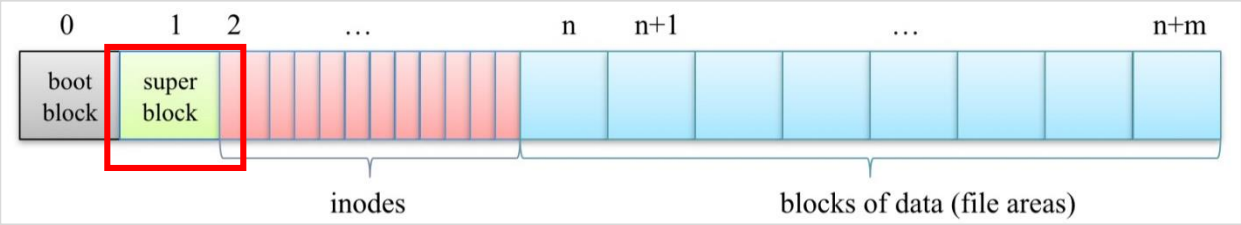
启动控制块(Boot Control Block)

包含加载内核的初始代码



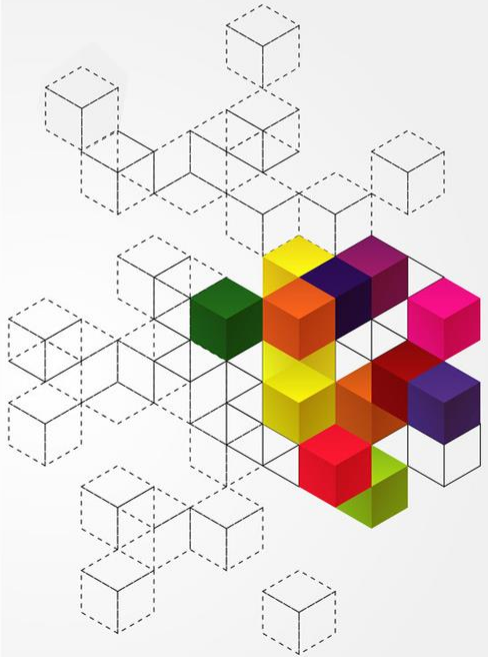
二、文件系统磁盘数据结构

磁盘数据结构逻辑示意图：



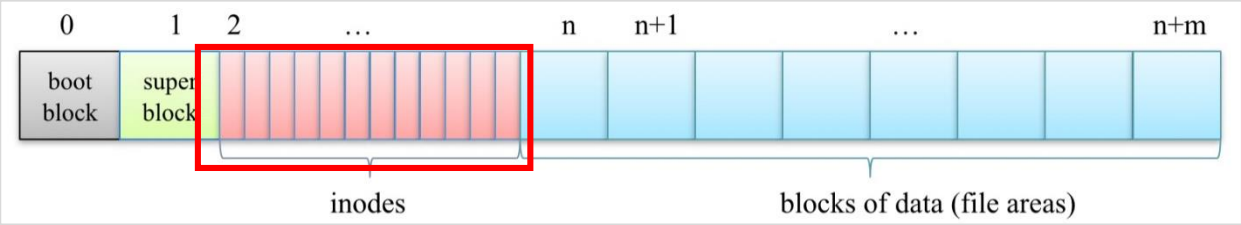
超级块(Superblock)

包含分区信息



二、文件系统磁盘数据结构

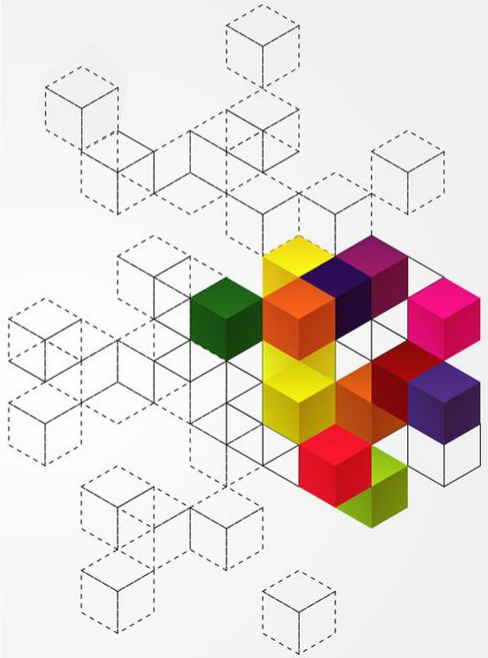
磁盘数据结构逻辑示意图：



文件控制块(File Control Block, FCB)

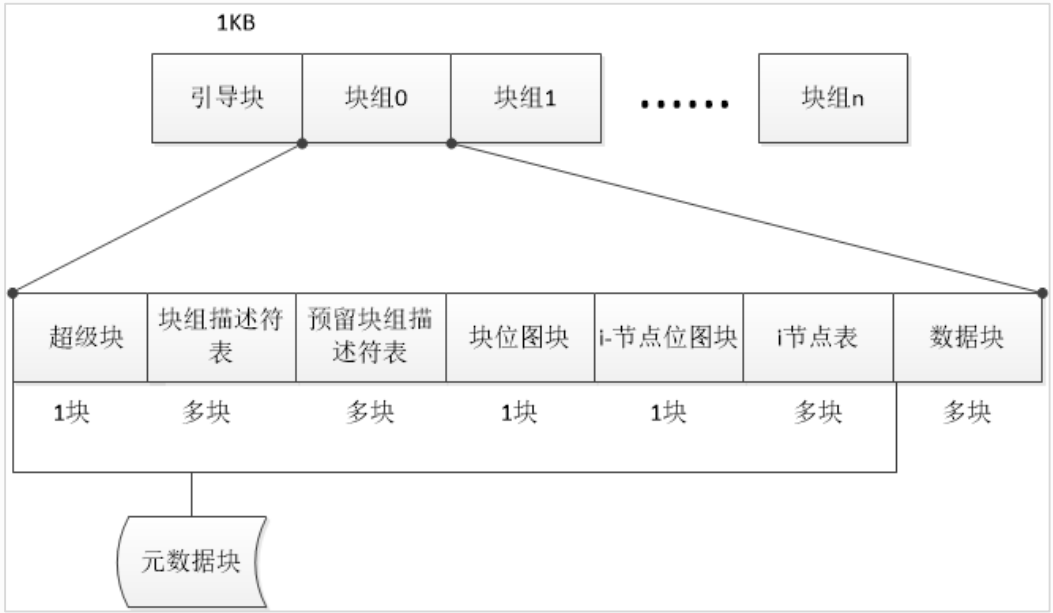
1. Basic Info	文件名\文件类型
	文件物理位置（盘号，外存起始地址 start_addr、size）
	文件的逻辑结构
	文件的物理结构
2.Control Info	文件owner
	访问权限：只读、读写、执行等
3.Manipulating Info	时间：创建和修改时间点
	使用该文件的当前进程数量，互斥锁情况

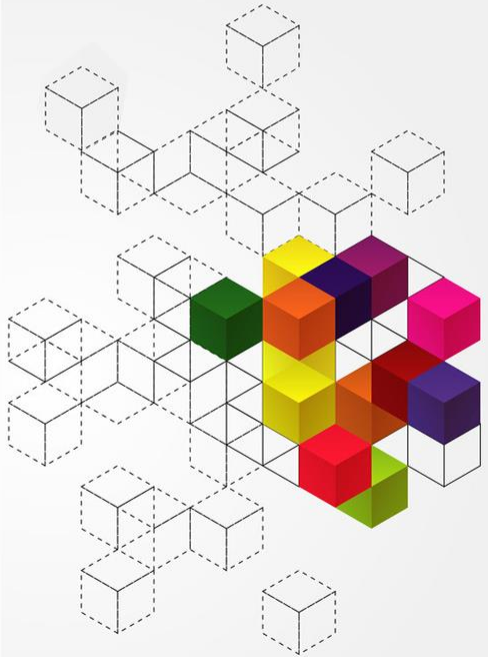
FCB是操作系统中用来唯一标识一个文件的数据结构



二、文件系统磁盘数据结构

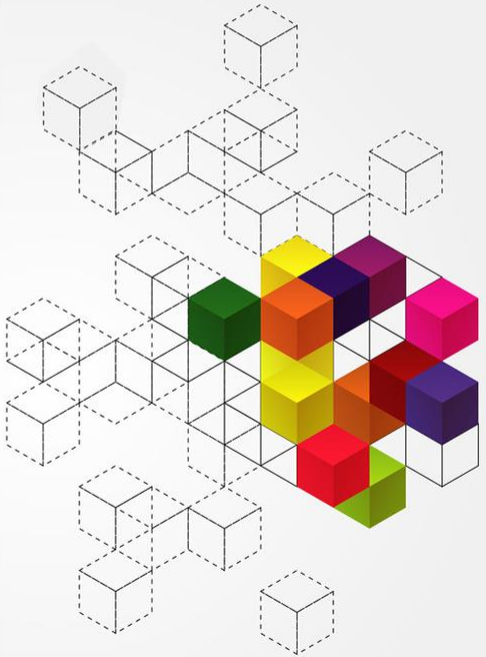
- **磁盘数据结构示例:Linux ext4**





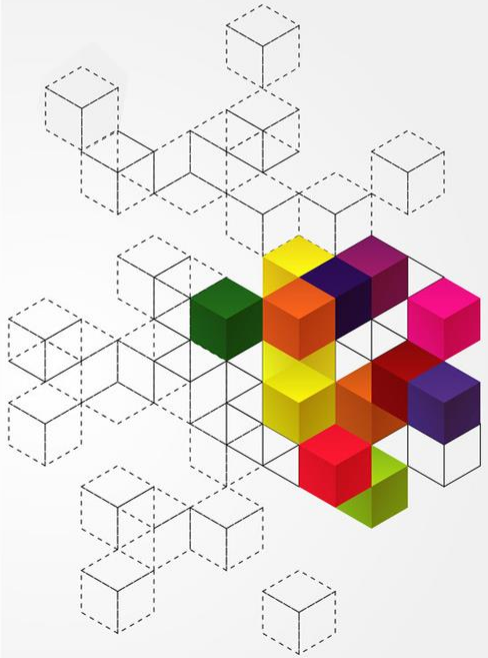
三、文件系统内存数据结构

- 文件系统内存数据结构（用于缓存关键和常用的文件系统数据）
  - 文件系统加载表（In-Memory Mount Table）
  - 目录缓存（In-Memory Directory Structure Cache）
  - 全系统打开文件表（System-wide open file table）
  - 进程打开文件表 (Per-process open file table)



三、文件系统内存数据结构

- **创建文件(File Creation)**
  - Create FCB for the new file
  - Update directory contents (in memory)
  - Write new directory contents to disk



三、文件系统内存数据结构

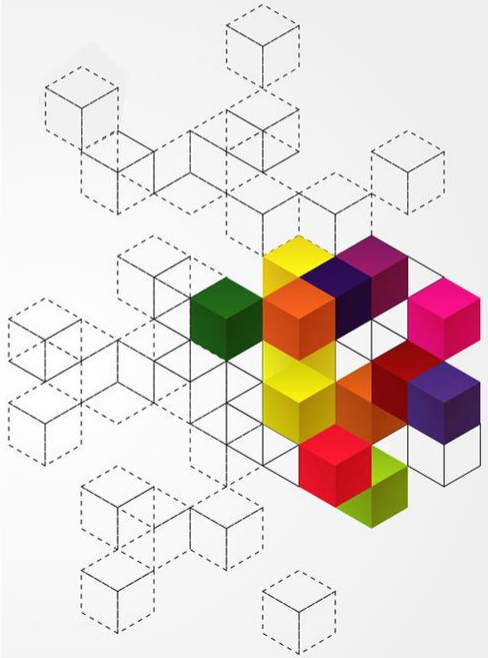
- **打开文件操作(File Open)**

  - Application passes the name through open system call

  - sys_open searches the system-wide open file table to see if the file is already in use by another process

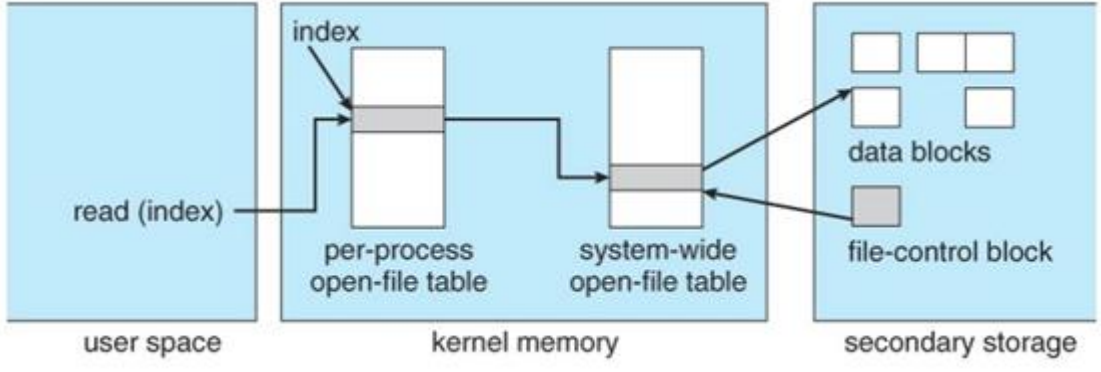
    - If yes, then increment usage count and add pointer in per-process open file table
    - If no, search directory structure for the name (either in the cache or disk) and add to system-wide open table and per-process open table

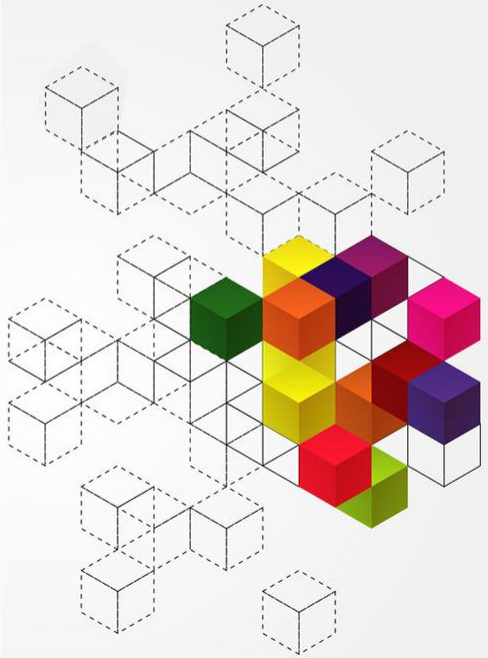
  - The pointer (or index) in the per-process open file table is returned to application, which becomes the file descriptor.



三、文件系统内存数据结构

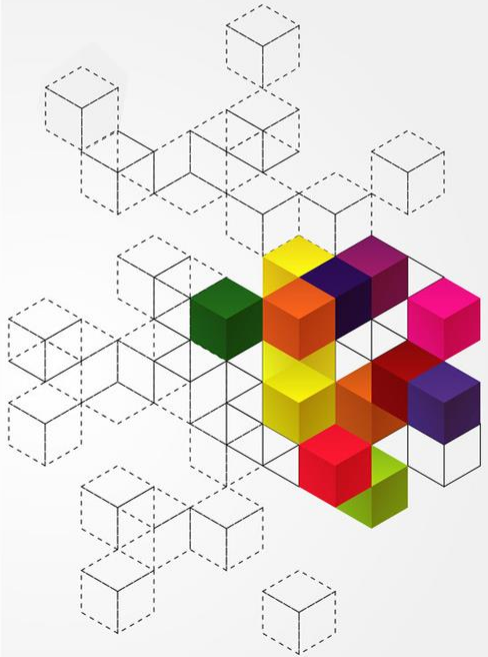
- 读文件操作(File Read)





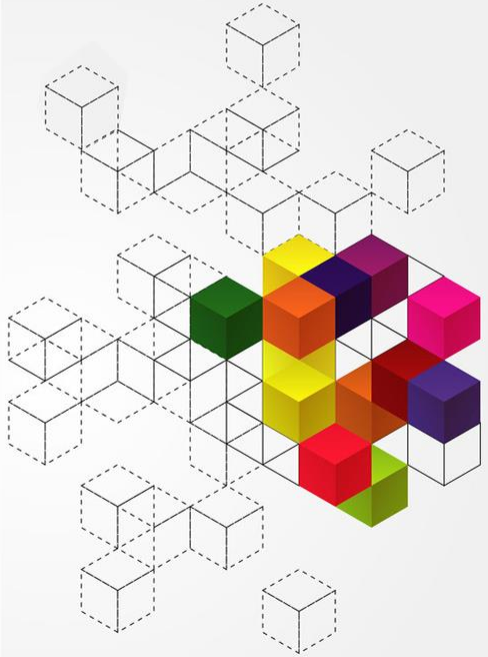
三、文件系统内存数据结构

- **关闭文件操作(File Close)**
  - Per-process open table entry is removed
  - System-wide open table reference count decremented by 1
    - If this value becomes 0 then updates copied back to disk (if needed)
    - Remove system-wide open table entry



本讲小结

- 文件系统实现考虑
- 文件系统磁盘数据结构
- 文件系统内存数据结构



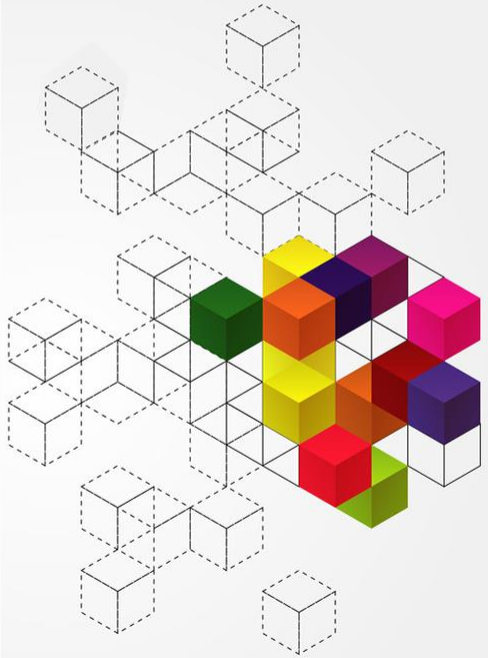
内容纲要

11.3 目录实现

一、 目录与文件按名存取

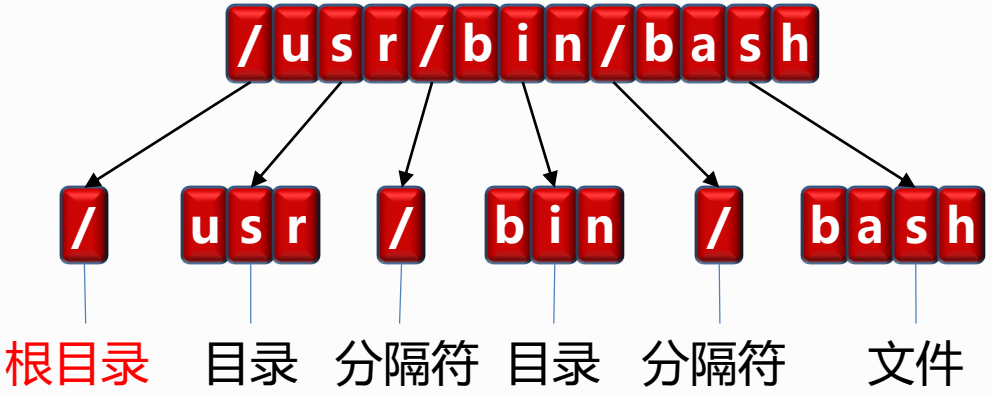
二、 目录数据结构

三、 文件操作示例

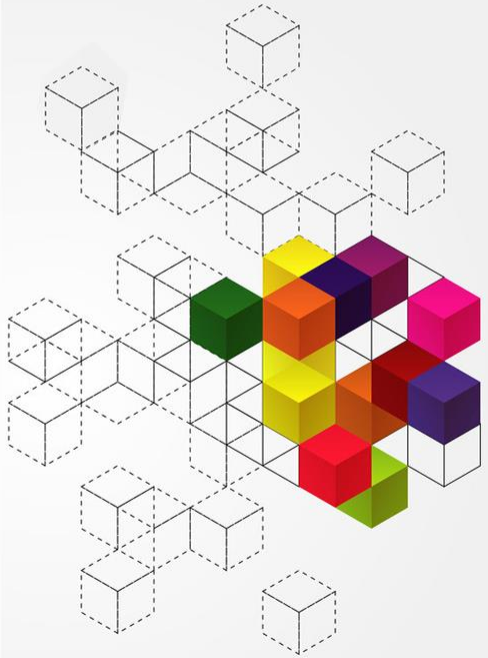


一、目录与文件按名存取

- **文件路径名组成**

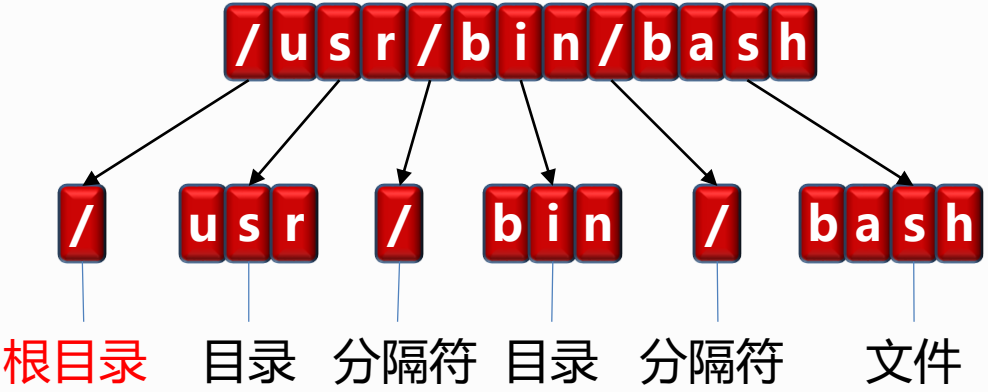


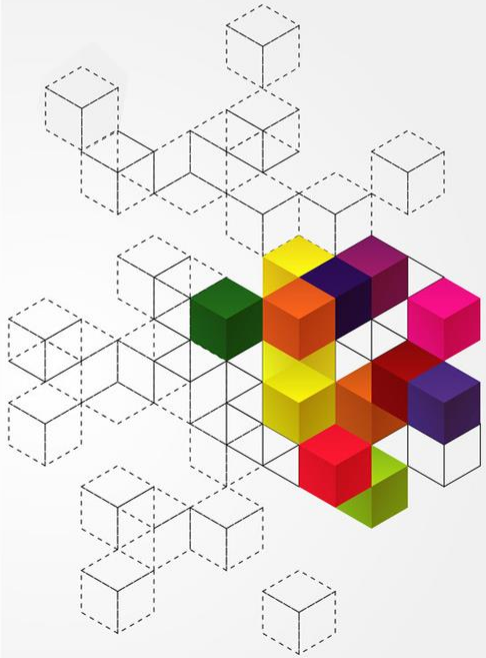
在访问文件/usr/bin/bash时，首先需要将路径以/为间隔符，进行逐级解析。第一级根目录中，以目录名usr在根目录中查找，得到其目录项，再在usr目录中查找名为bin的目录项，最后在bin目录中查找bash文件



一、目录与文件按名存取

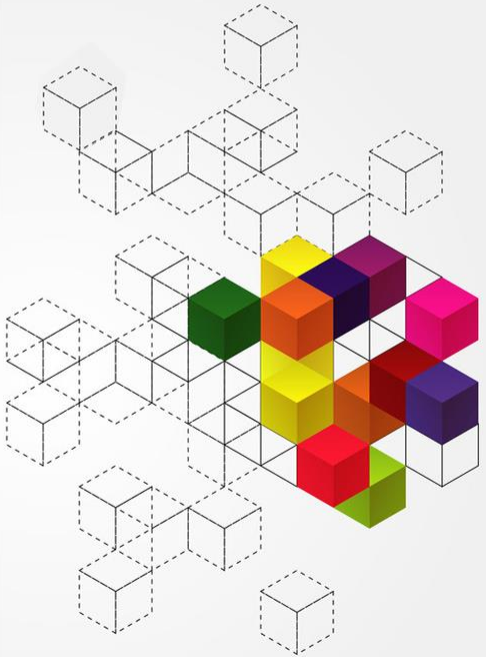
- **文件路径名解析**
  - 目的：根据文件路径，找到对应的文件控制块
  - 路径名解析在文件open等操作中经常被使用
  - 解析过程中，会频繁使用目录查找操作，对目录查找效率提出较高要求





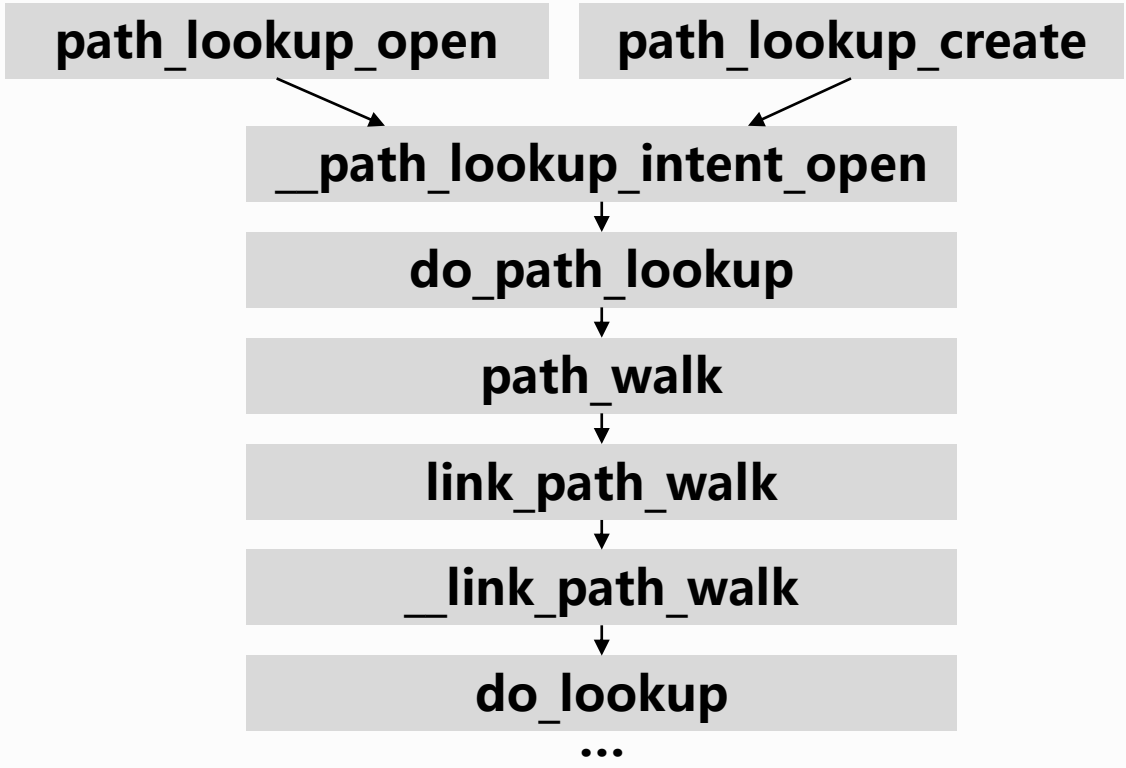
一、目录与文件按名存取

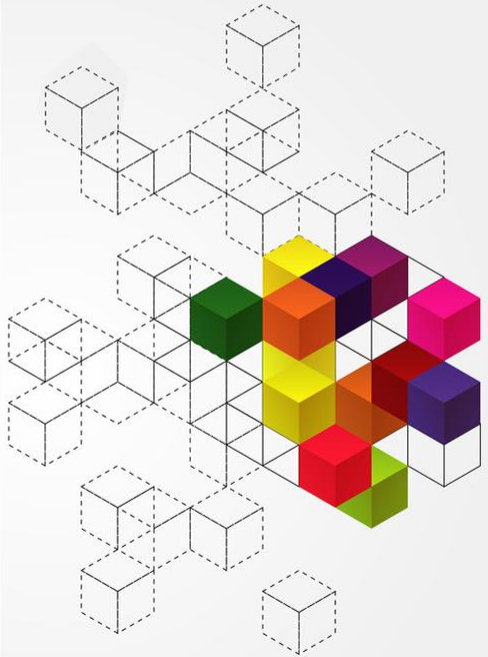
- **文件路径名解析**
  - 解析的起点：
    - 绝对路径：current->fs->root
    - 相对路径：current->fs->pwd



一、目录与文件按名存取

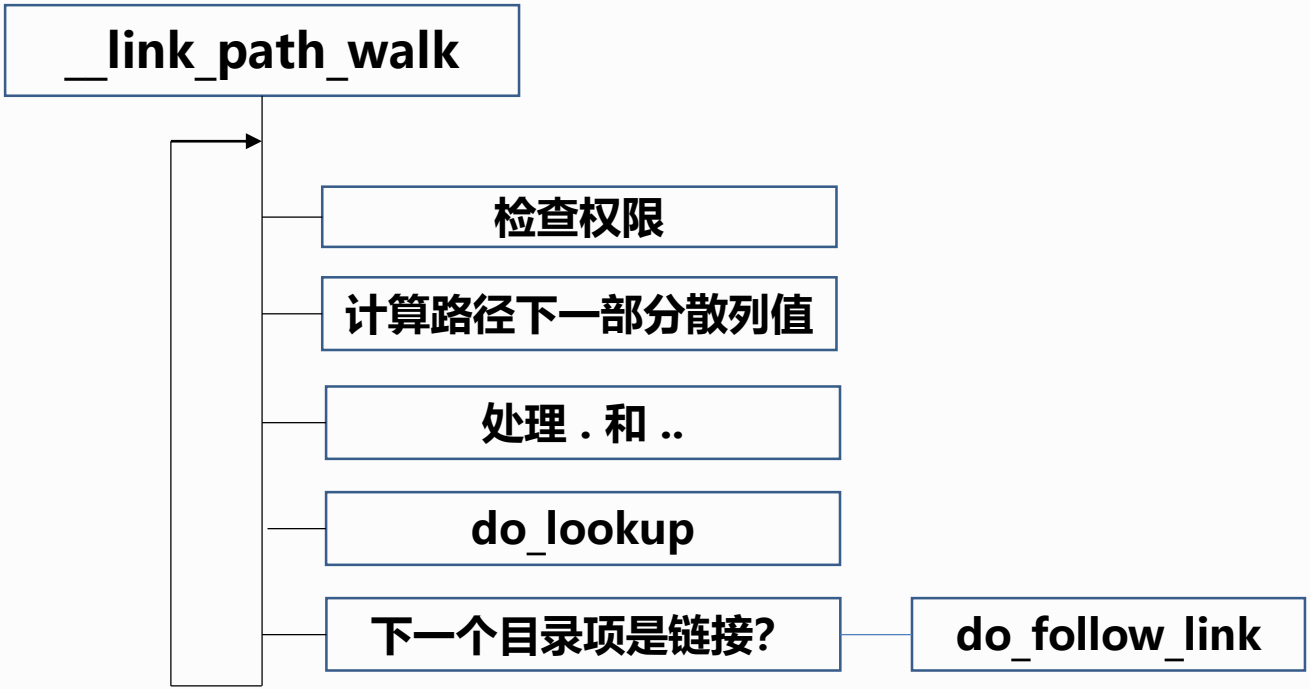
- **文件路径名解析**
  - Linux中解析代码的函数调用序列

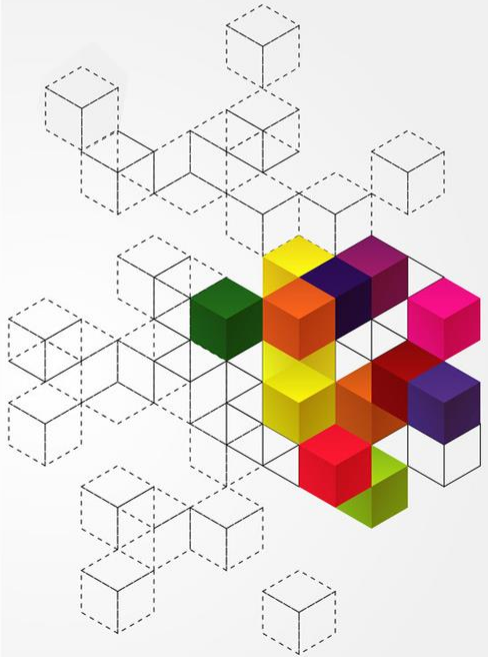




一、目录与文件按名存取

- Linux文件遍历实现：__link_path_walk()

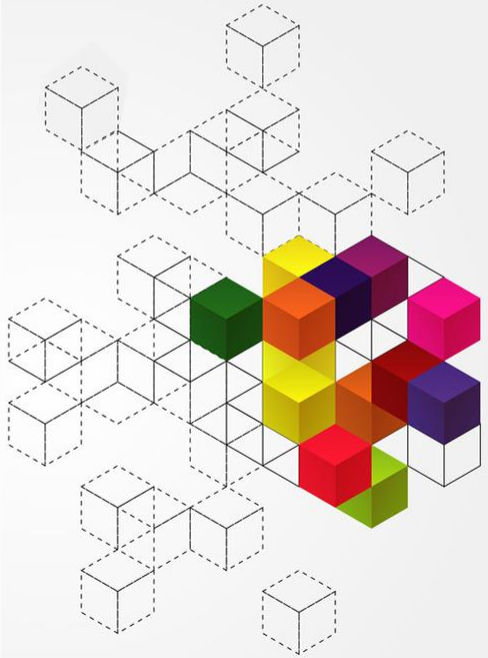




一、目录与文件按名存取

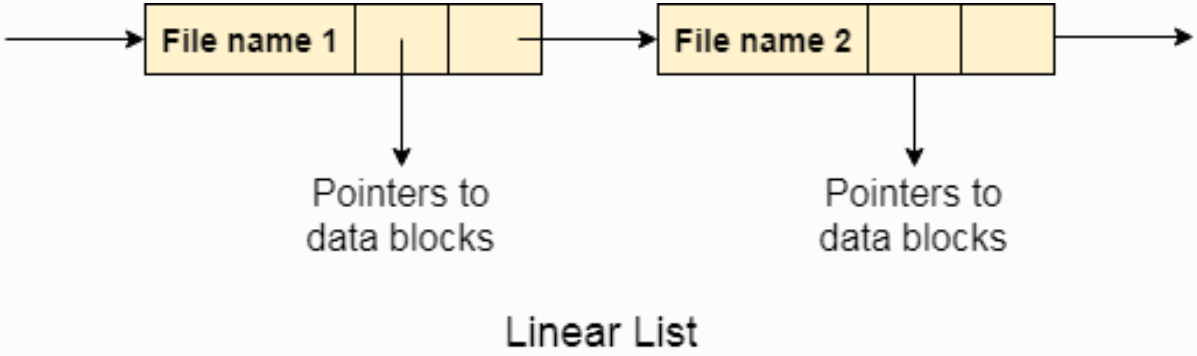
- **Linux路径查找函数示例：path_lookup**

```
1  int fastcall path_lookup(const char *name, unsigned int flags,
2                           struct nameidata *nd)
3  {
4          return do_path_lookup(AT_FDCWD, name, flags, nd);
5  }
```

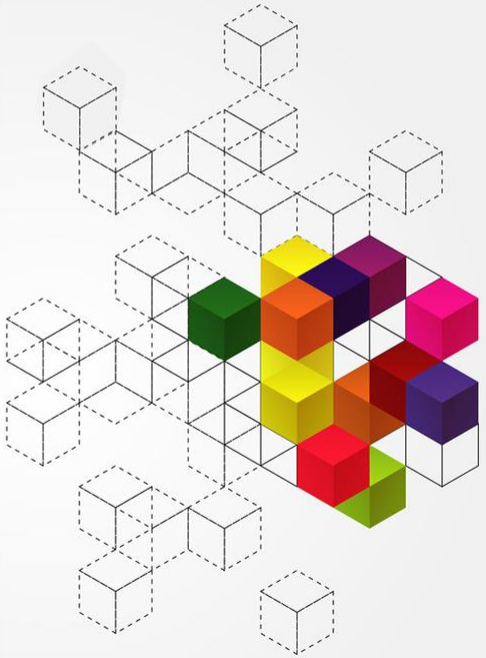


二、目录数据结构

- **目录结构1：线性表**

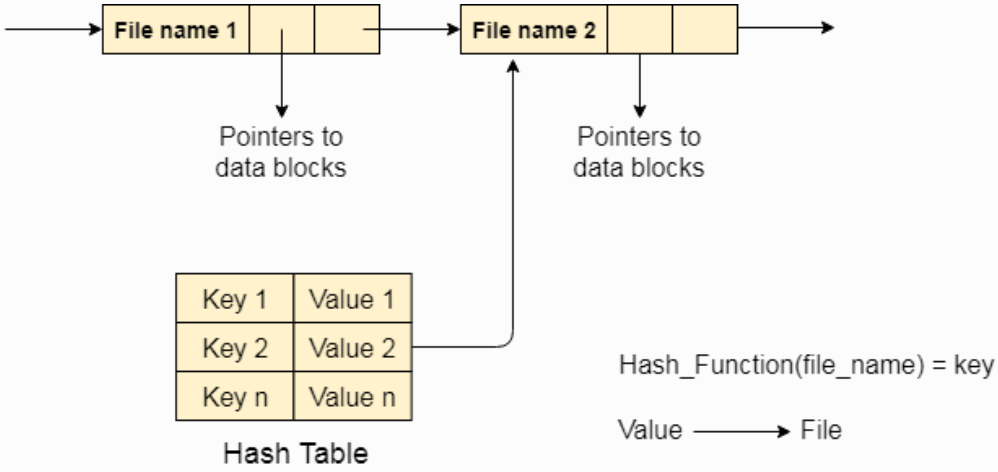


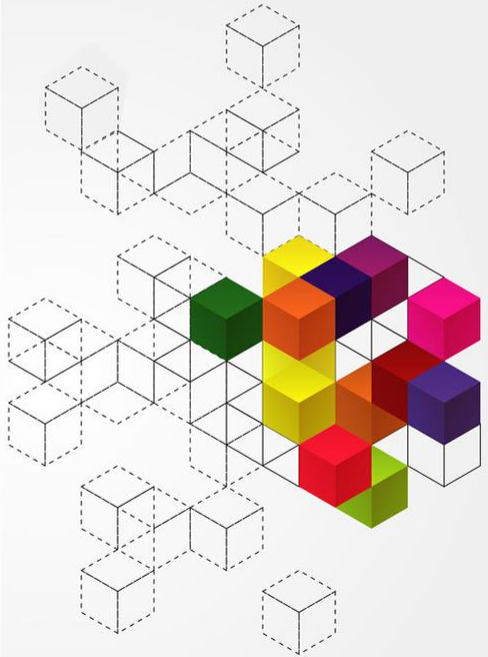
Linear List



二、目录数据结构

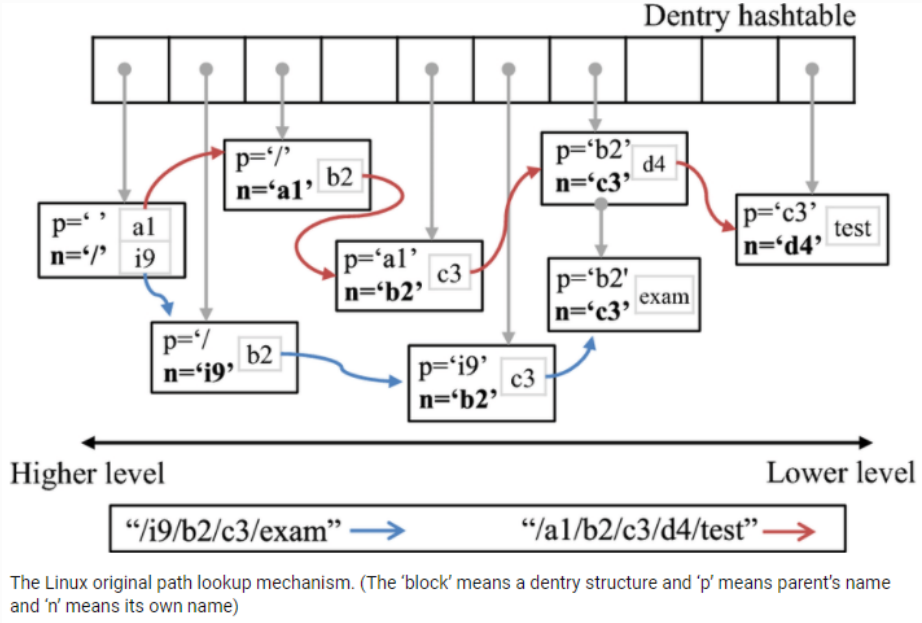
- **目录结构2：哈希表**



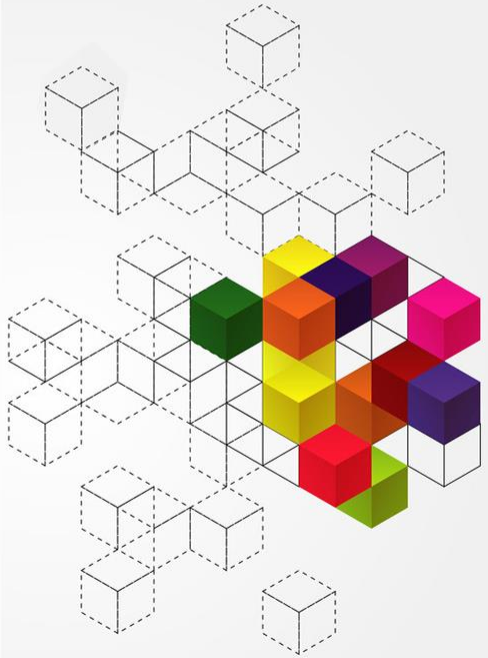


二、目录数据结构

- **Linux中目录查找过程示例**

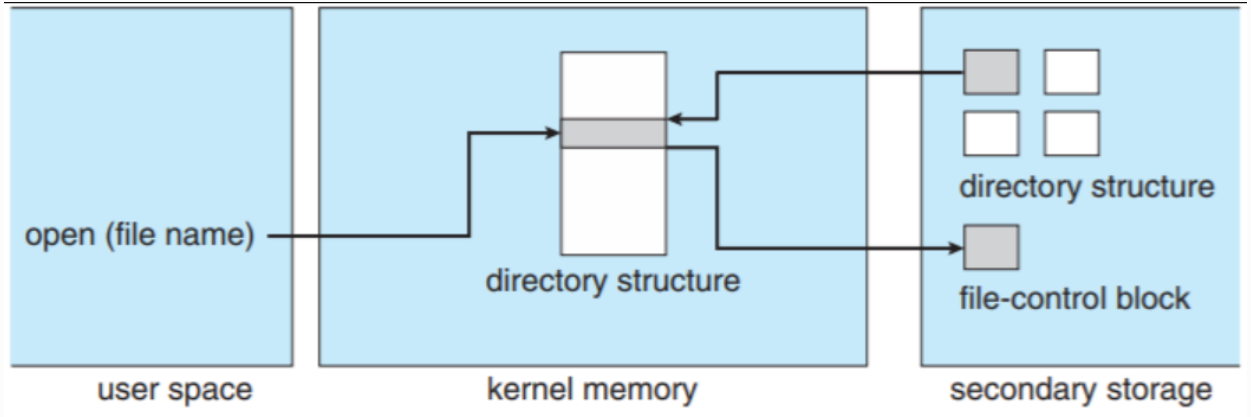


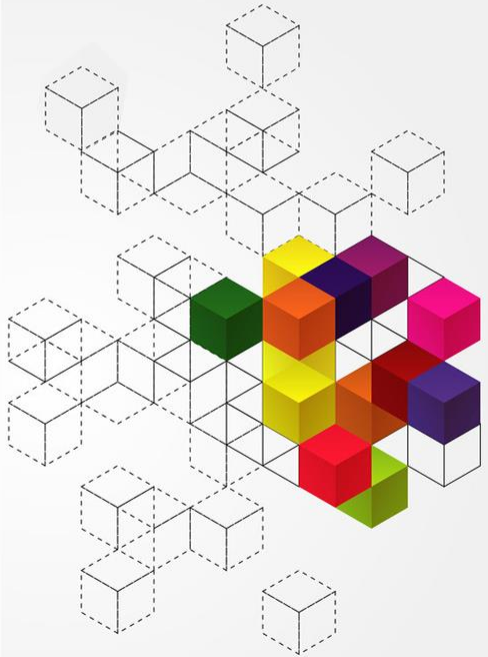
The Linux original path lookup mechanism. (The ‘block’ means a dentry structure and ‘p’ means parent’s name and ‘n’ means its own name)



三、文件操作实现示例

- 示例：文件open操作实现





本讲小结

- 目录与文件按名存取
- 目录数据结构
- 文件操作实现示例