File system: a hotpot lesson

Liwei Guo, Ph.D.

Huawei Technologies Co., Ltd

郭力维

OS技术研究员，华为技术有限公司

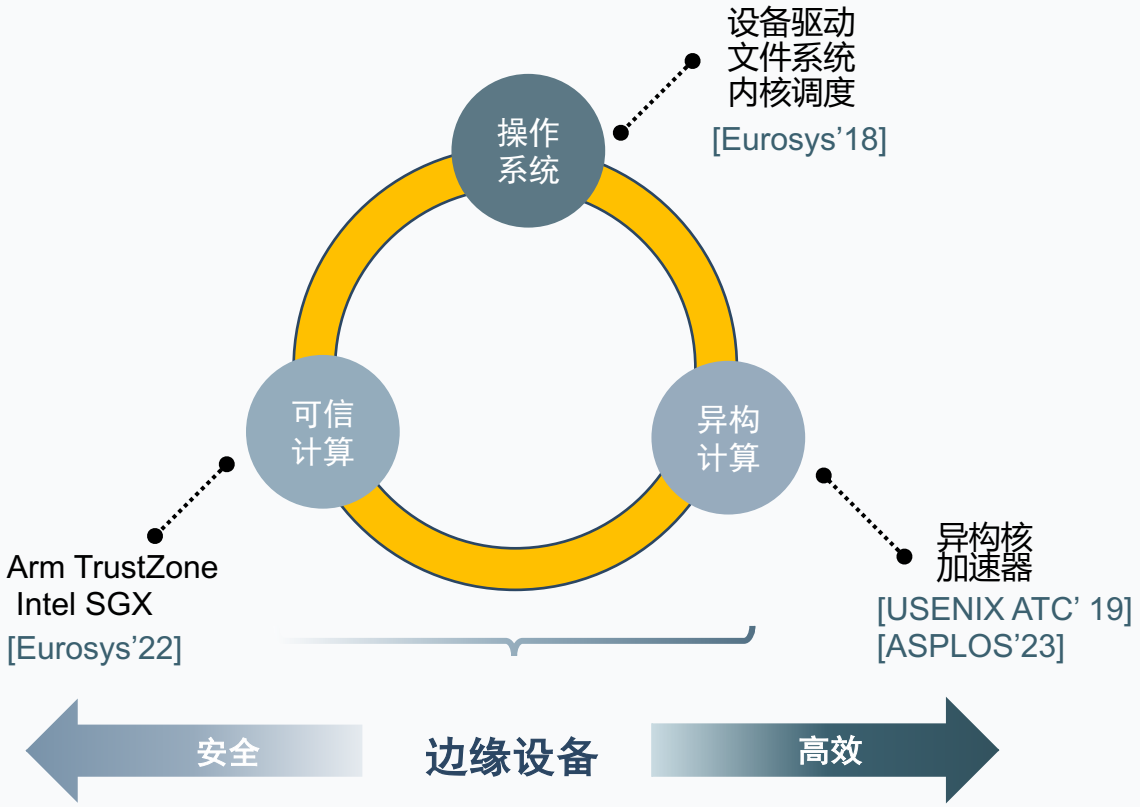
教育经历

本科	软件工程	中山大学	2012 - 2016
硕士	计算机工程	普渡大学（美国）	2016 - 2020
博士	计算机科学	弗吉尼亚大学（美国）	2020 - 2022

工作经历

Meta	博士实习生	2020.05 - 2020.08
HUAWEI	OS技术研究员	2023.03 至今

研究方向：系统软件 (OS x Arch x PL)

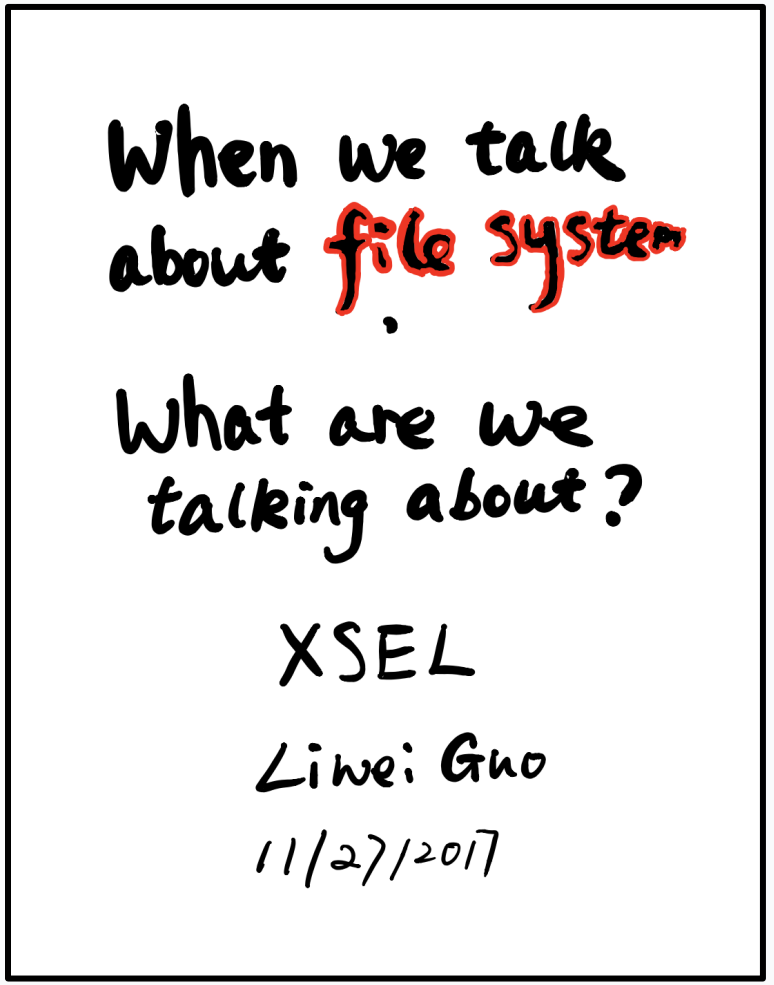


Outline

- Recap: file system fundamental
  - **What** is a file system?
  - **Why** does it matter?
  - **How** does it work?
- Fun stuff about file systems
  - Practical lessons
  - Advanced features
  - Modern file system research

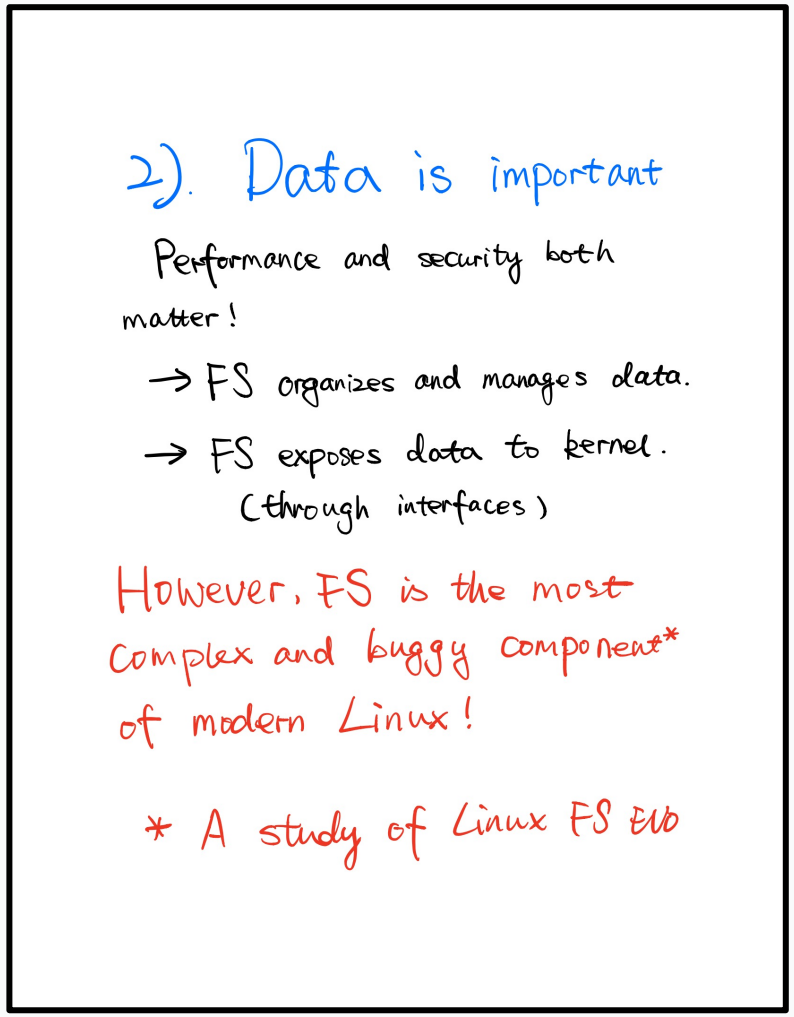
Recap: file system fundamentals

- Original talk @ XSEL @ Purdue
  - Was traveling, not so formal :)
- An introductory lesson
  - How file systems (generally) work
- Useful to cover high-level ideas



Why do we care about file systems?

- Simple, because **data** matters
  - Application binary
  - Photos, videos
  - Sensor data
  - ...
- Data need proper management
  - Performance (e.g. loading time)
  - Security & confidentiality
- Part time job in 电脑城



A brief history

- Earliest file system
  - Multics by Ken Tompson & Dennis Ritchie (60s)

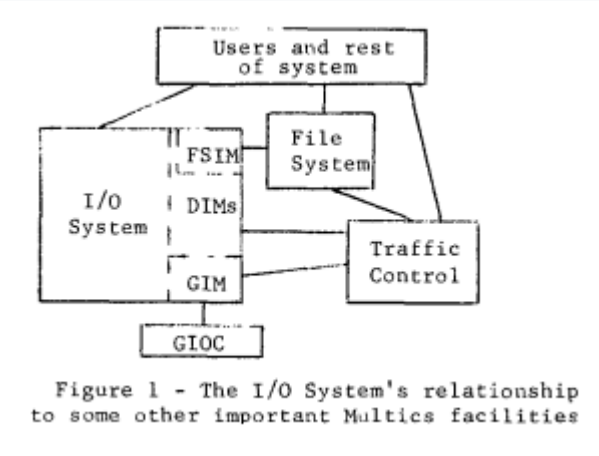
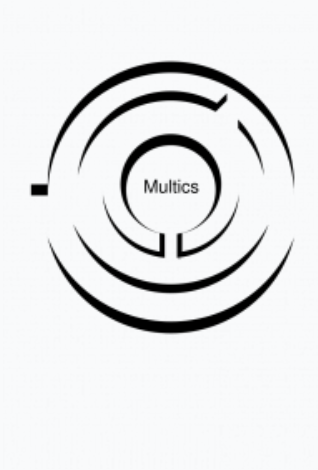


Figure 1 - The I/O System's relationship to some other important Multics facilities



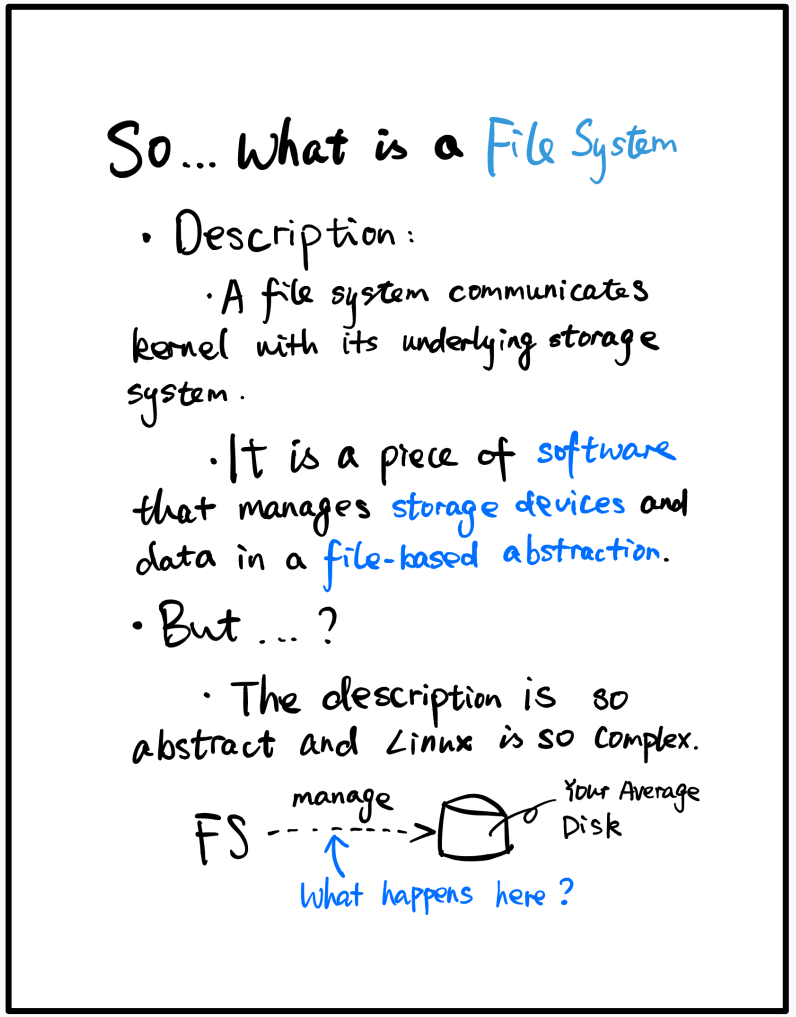
- First introduced to Linux in around 90s
  - MINIX file system
- Quickly become one of the largest kernel subsystem

Some History:

* Linux File System was first introduced around 90s (minix)
* Hundreds → >100K
  then present
* Large # of FS:
  FAT, EXT, NTFS.
  NFS. BTRFS ...

What is a file system anyway?

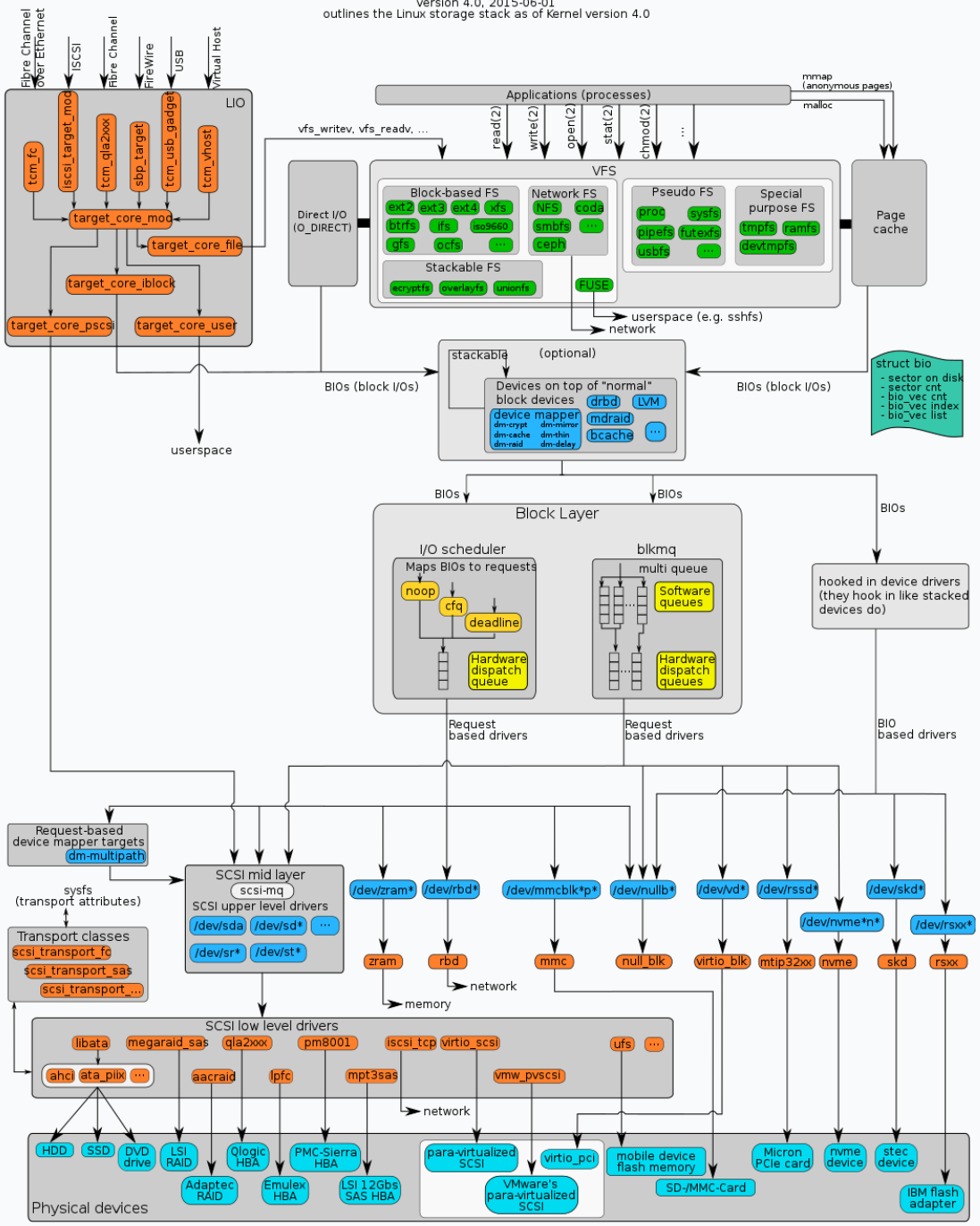
- FS in one sentence
  - A piece of software which manages data on the underlying storage deevices in a file-based abstraction
- Updated after 6 yrs:
  - A piece of software which manages disk address space
- I will give more specific explanation!



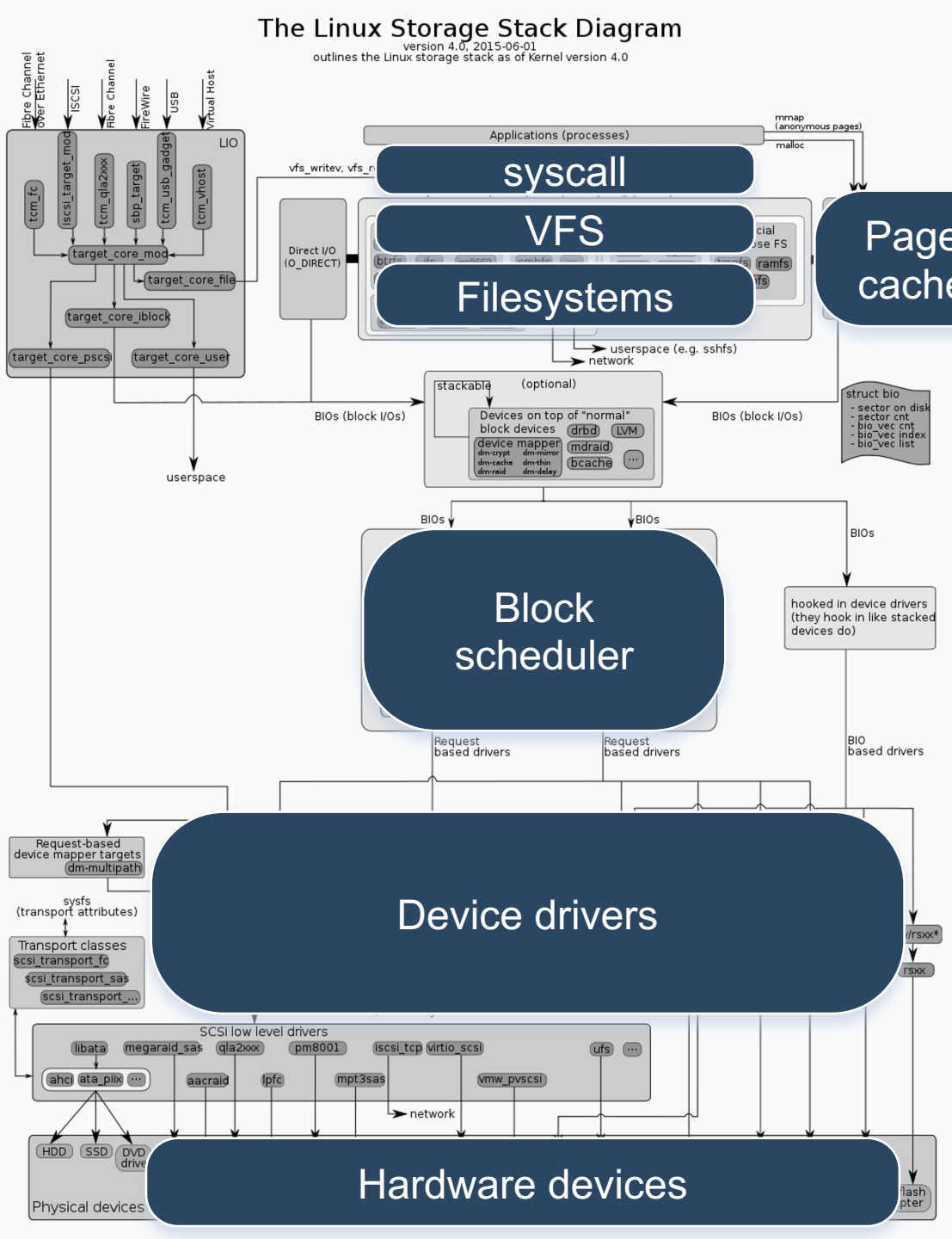
The Linux Storage Stack Diagram

version 4.0, 2015-06-01

outlines the Linux storage stack as of Kernel version 4.0



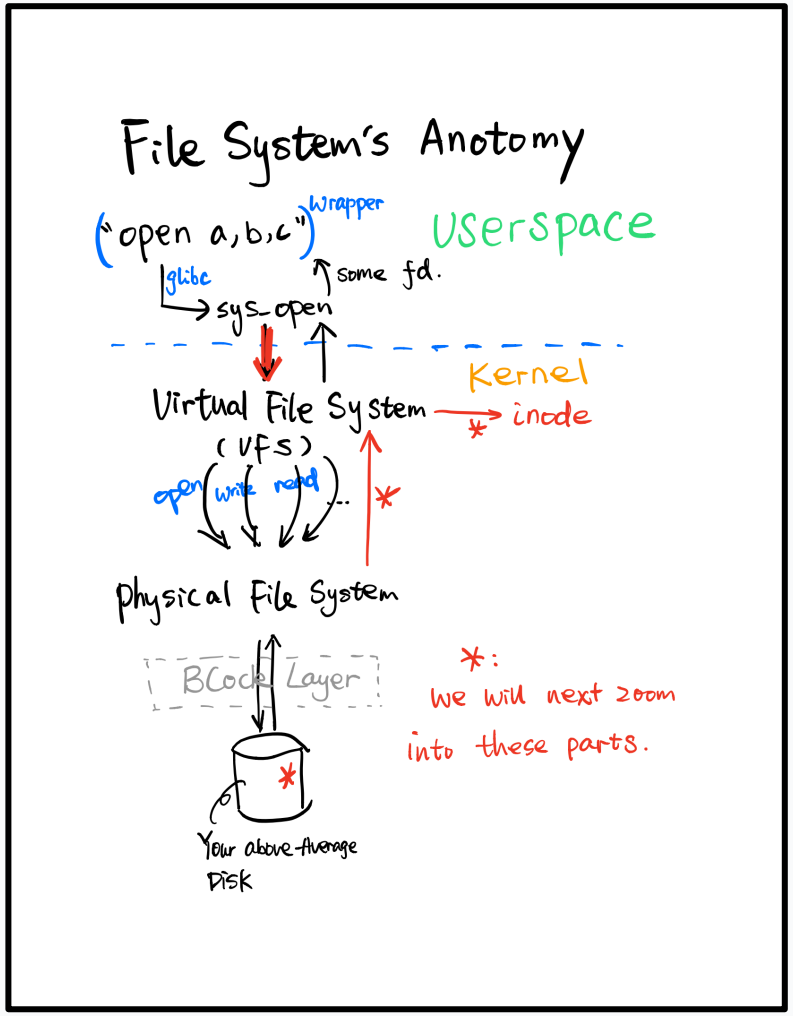
https://en.wikibooks.org/wiki/The_Linux_Kernel/Storage



https://en.wikibooks.org/wiki/The_Linux_Kernel/Storage

FS anatomy

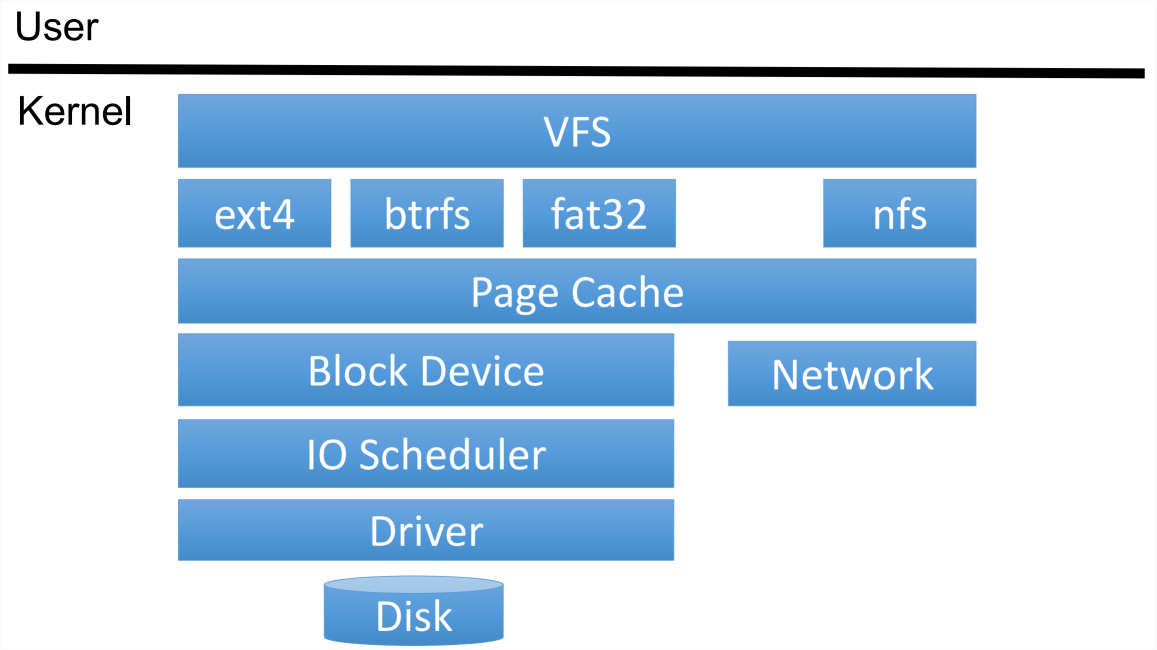
- A concrete example:
  - open(...) => sys_open => svc(arm)/int (x86)
  - Virtual File System (VFS):
    - Caching
    - Maintaining file states (e.g. FD)
    - Dispatching requests to physical fs
  - Phyiscal File System (e.g. EXT4)
    - File <-> Disk blocks
  - Block Layer
    - Receives block requests, returns data



Virtual File System (VFS)

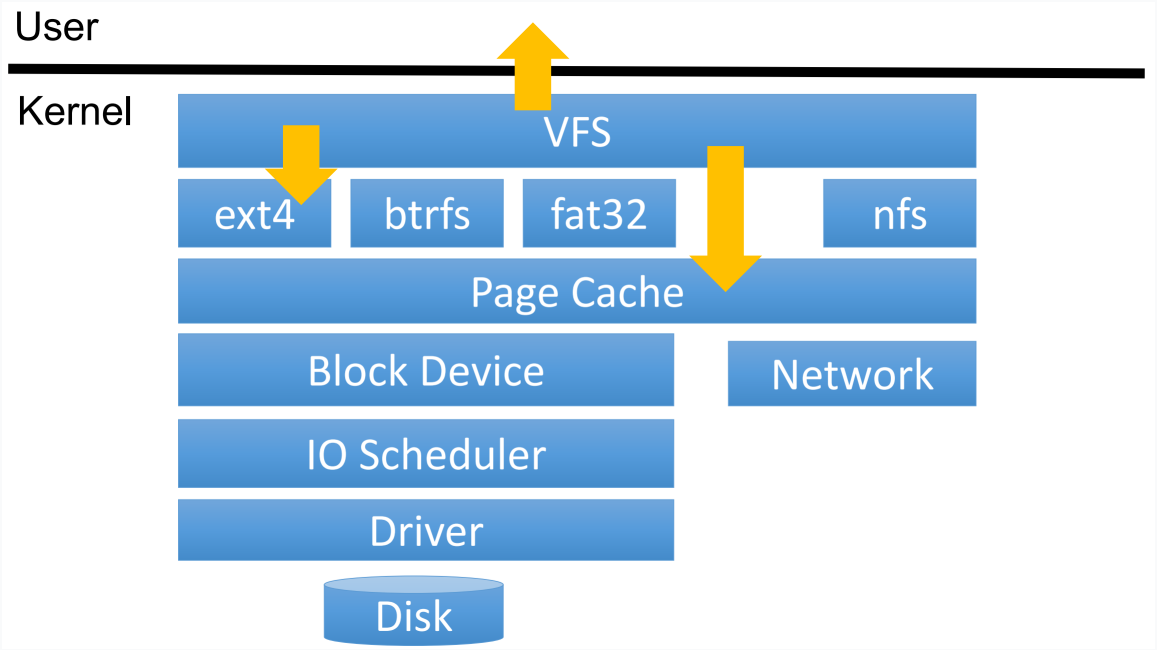
- VFS abstracts the underlying persistent storage and provides the users with common POSIX interfaces
  - open/close
  - read/write
  - lseek
  - sync

- Why need?
  - All problems in CS can be solved with another layer of abstraction!



VFS responsibilities

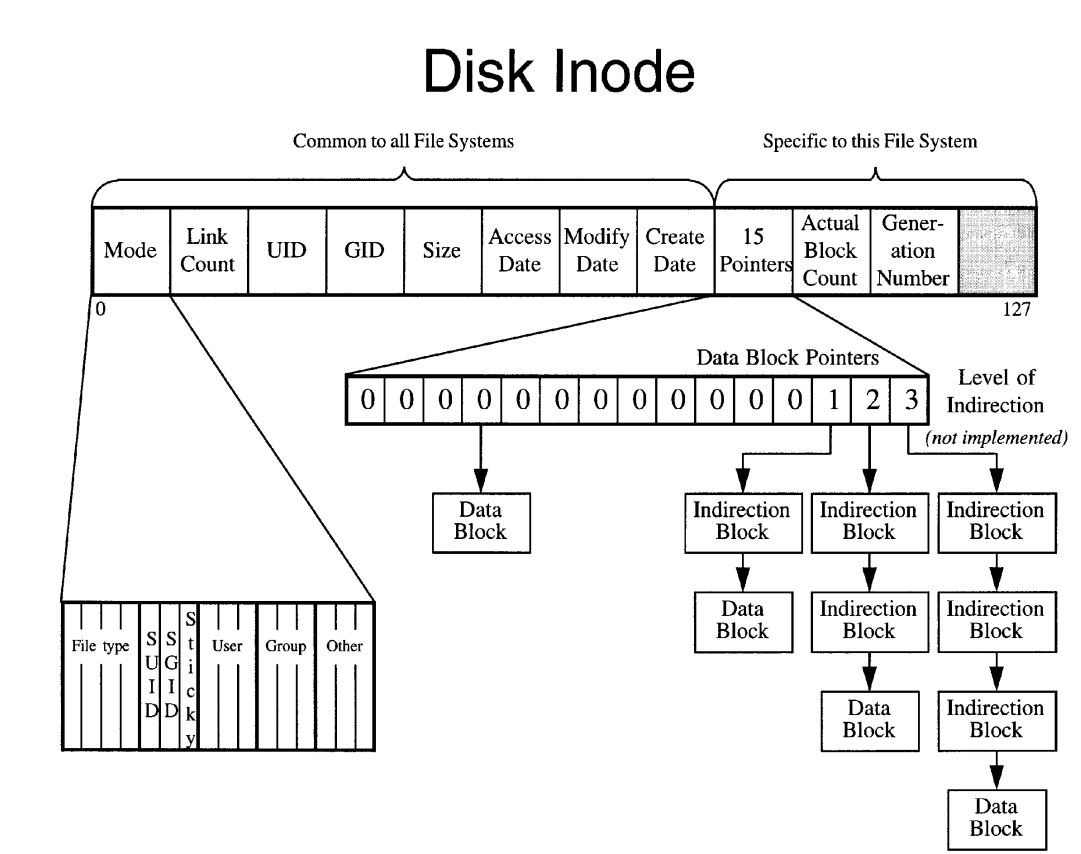
- To user
  - POSIX interfaces
- To phyiscal file system
  - Hooks for name resolution, file data fetching, etc.
- To performance
  - Maintaining page cache
- In reality, a substantial piece of code
  - IIRC, larger than many file system implementation
- Core abstraction
  - Inodes, dentries, super block



Recap: Inodes/Dentry

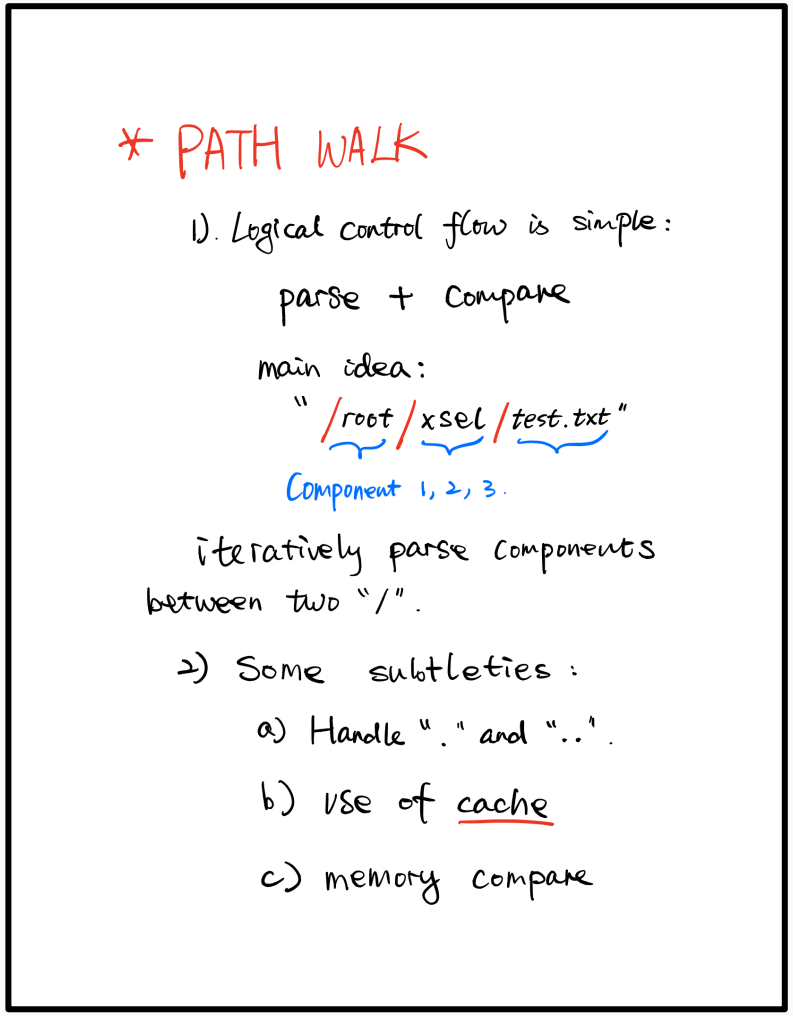
- Internally, files don't have a name but only **inodes**
- Directory entry, i.e. **dentry**, bookkeeps inode to file name mapping
  - A Special type of file
- **Inodes are pointers to files**
- **Inodes are file metadata**
  - File block layout
  - Permission bits
  - Do not include file names!

Inode Number	File Name
34	.
63	..
2	file1
56	file2
133	dir1



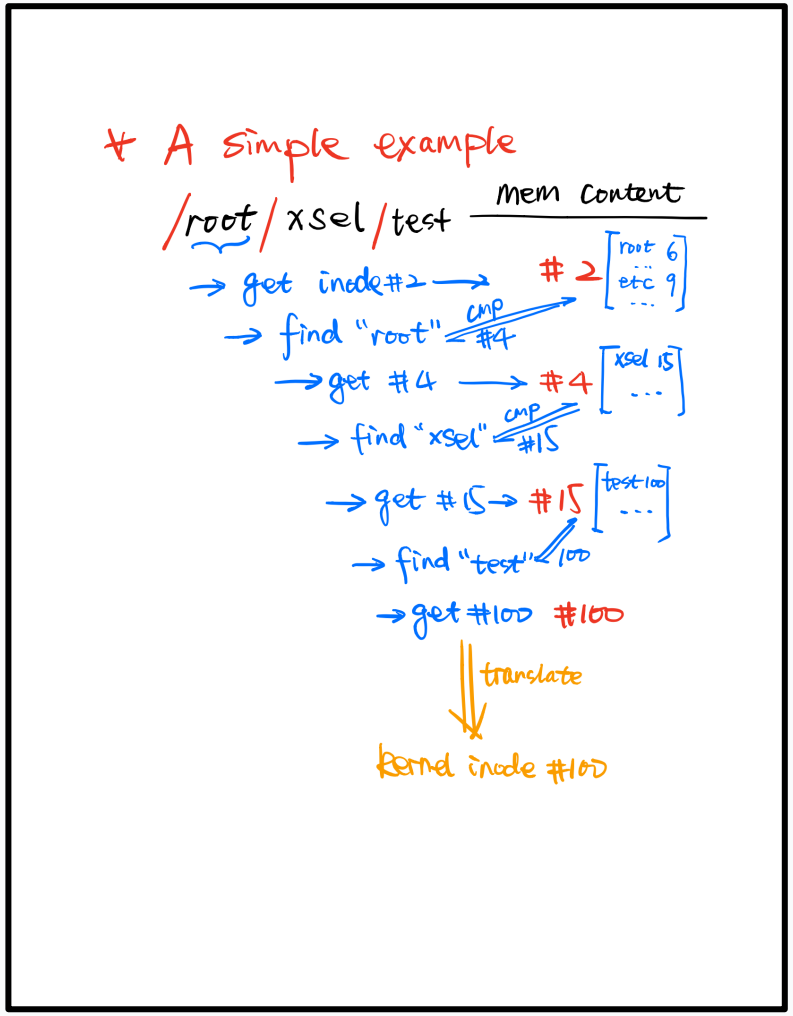
Name resolution via *path walk*

- What happens when you try to open a file:
  - /root/xsel/test.txt
- Key idea:
  - Recursively search file/directory name under current working directory
- Quirks in kernel implementation
  - “.” and “..”
  - Kernel exploits VFS caching for storing



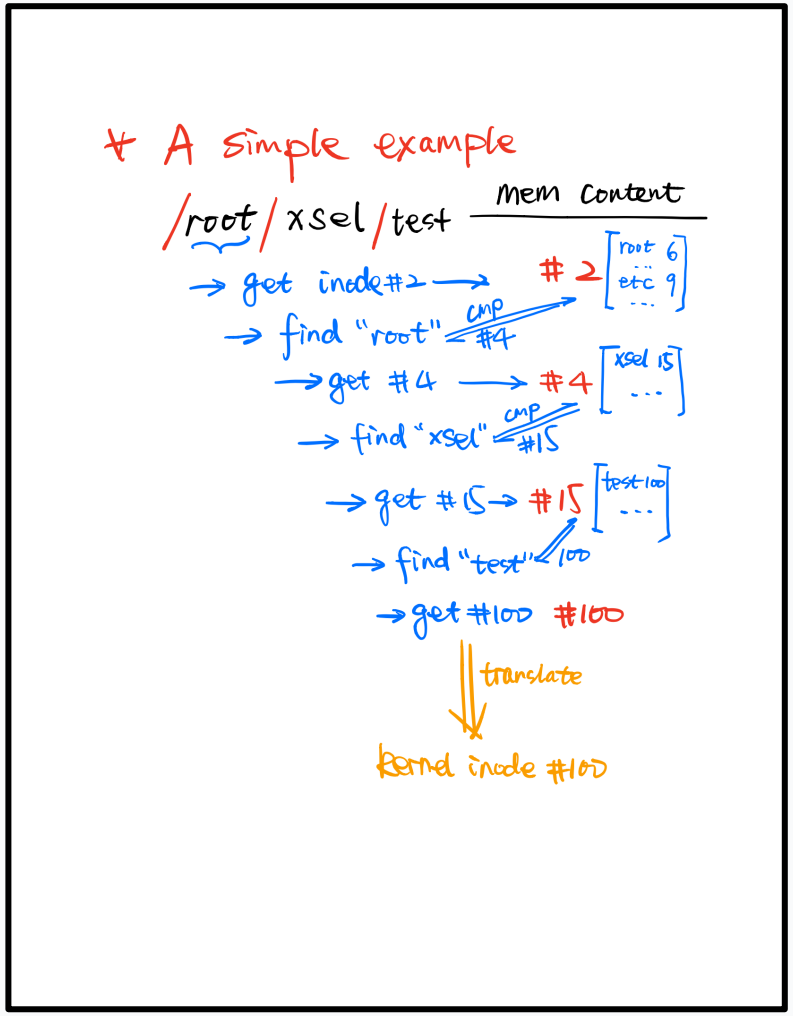
Iterative path walk

- “/” : where is this directory?
- Specifically, what is its inode?



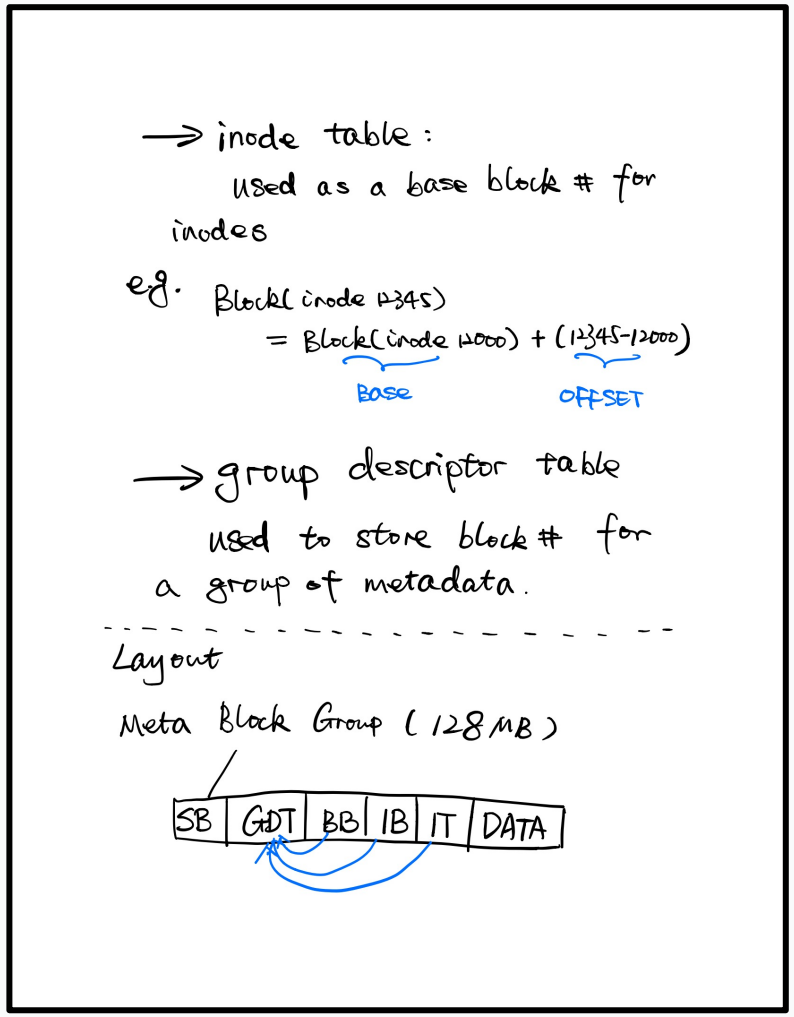
Iterative path walk

- “/” : the root directory w/ inode #2
  - root: inode #4
  - etc: inode #9
- “/root”: fetch #4
  - xsel: inode #15
- “/root/xsel”: fetch #15
  - test: inode #100
- “/root/xsel/test.txt”: fetch #100



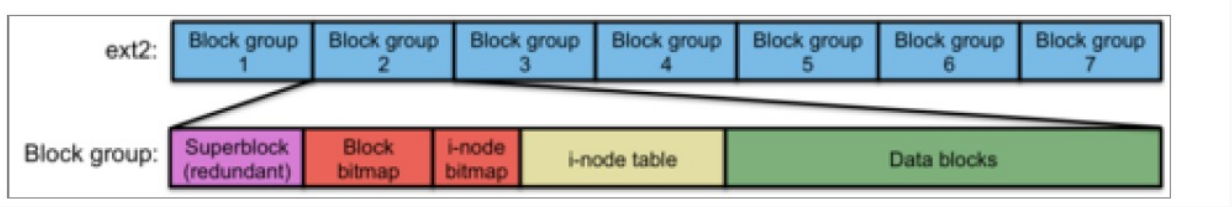
How to locate inodes?

- By querying fs-level *metadata*
  - SB ==> GDT
- Specific to file system design & implemntation
- Key metadata (EXT2/3/4)
  - Super block (SB)
  - Group descriptor table (GDT)
  - Block Bitmap (BB)
  - Inode Bitmap (IB)
  - Inode Table (IT)



How to allocate inodes?

- Block allocation
  - Often have a minimal unit of allocation, e.g. 8KB
  - Delayed allocation for saving disk space
  - Data loaclity in mind
- Still an iterative process
  - SB => GDT => IB => BB



Block Allocation:

* Speculatively 8KB
* Delayed allocation
* Data Locality

⇓

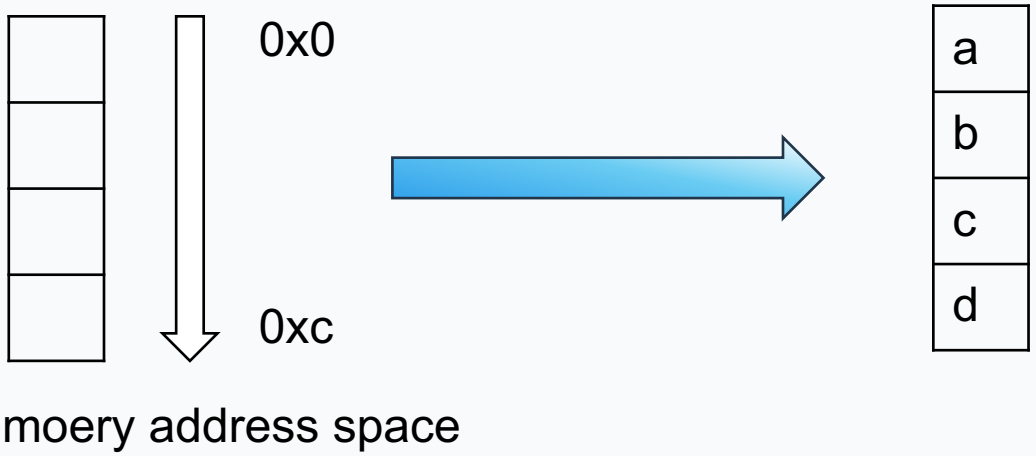
Query SB to find GDT, then look into IB to assign inode, then look into BB to allocate blocks, write inode content back.

File systems manage disk address space

File systems manage disk address space

- Let's look at another analogy: **struct** in C

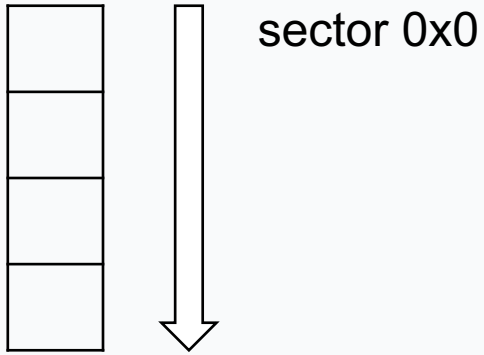
```
typedef struct {
        int a; // 4 bytes
        int b; // 4 bytes
        int c; // ...
        int d;
} foo;
```



The struct *foo* manages address space of a data structure

File systems manage disk address space

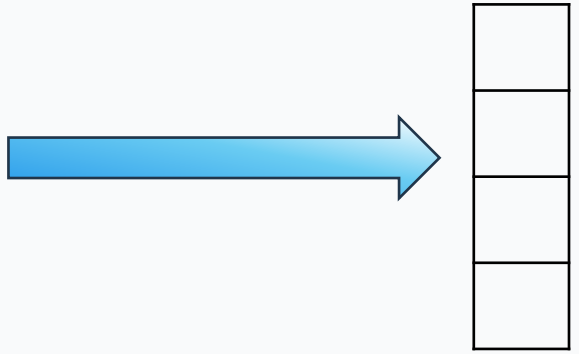
- **File systems work similarly, too!**



sector 0x0

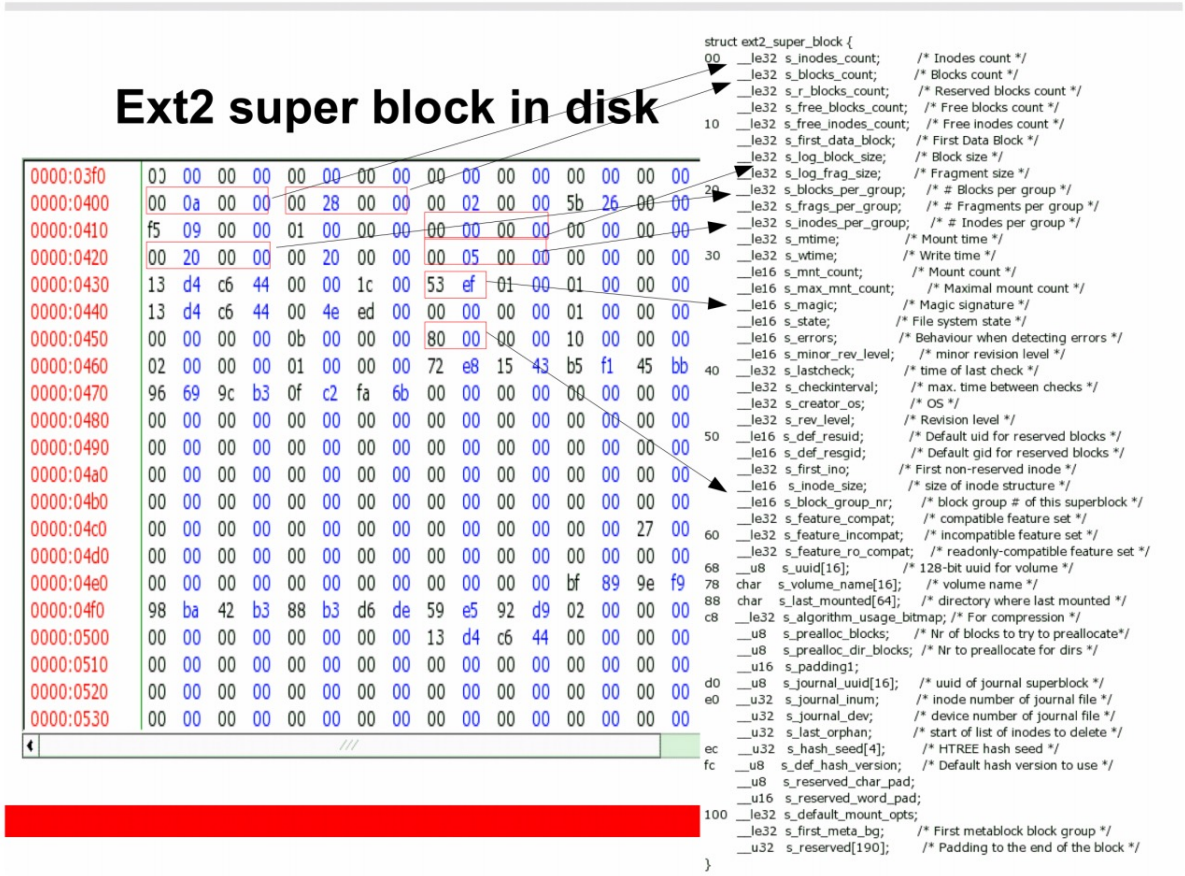
disk address space

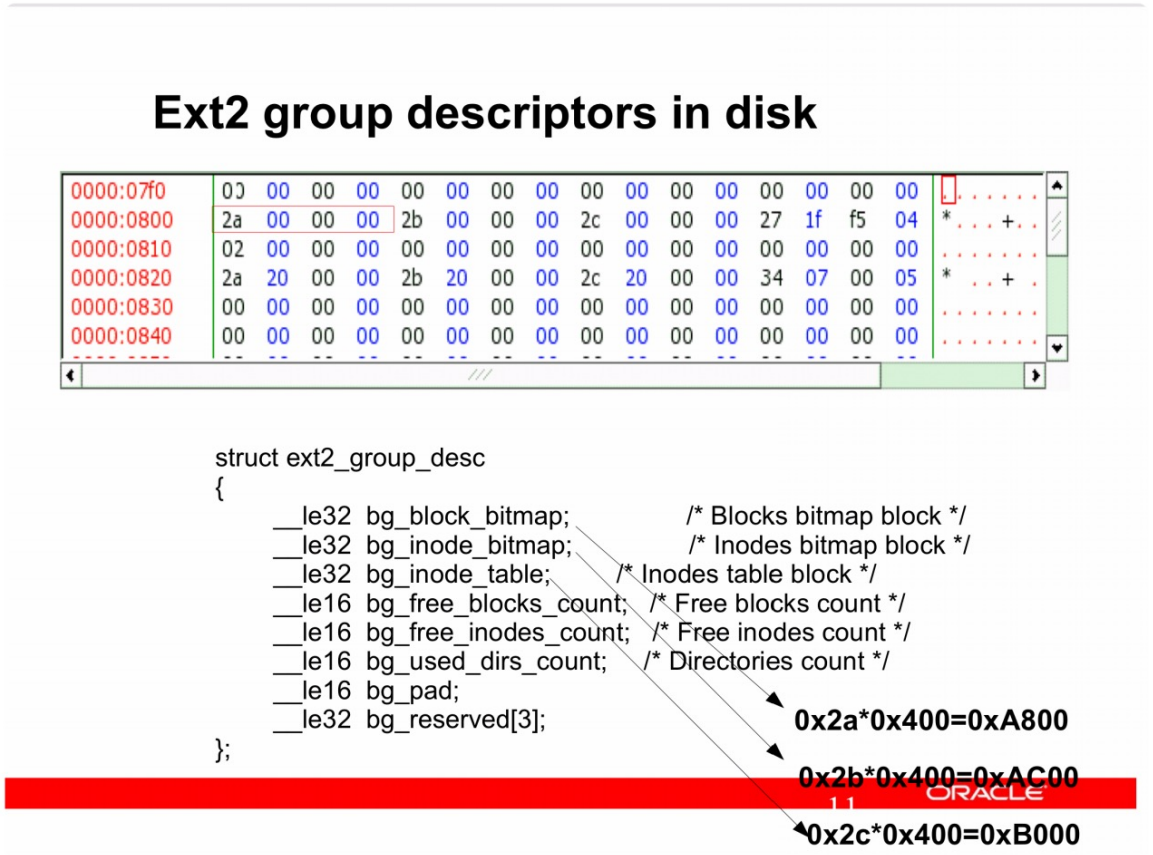
```
typedef {
        struct sb; // N sectors
        struct gdt;
        ...
        ...
} ext4_sb;
```



ext4 disk image

Deep dive into file system images





*le: little endian. Why do we need to specify this?

Try yourself: inspecting ext2 filesystem

What we will do:

- Mount an ext2 filesystem image with a few dir/files.
- Use Linux tools to examine key data structures: inodes/bitmaps...
- Show how such a filesystem image is created

Objectives:

- Reinforcing understanding of ext2
- Familiarizing with Linux file tools

The tools we will use

ls: show directory content

dd: read/write raw content of a file (or a disk partition)

mkfs.ext2: create a new ext2 filesystem

debugfs: ext filesystem debugger

stat: get file status

dumpe2fs: dump ext filesystem info

xxd, hexdump: display binary data

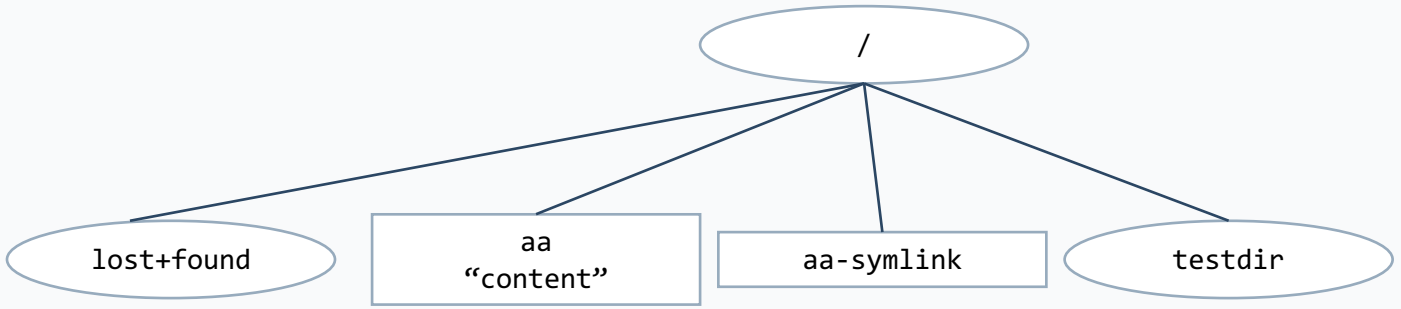
Note: permissions

- Fiddling with a whole filesystem (and disk image) requires root
- Do this on your computer
- Or in QEMU

Get image & mount

```
wget https://engineering.purdue.edu/~ee469/lectures/disk.img.gz
gzip -d disk.img.gz

ls -lh disk.img
-rw-r--r-- 1 xzl xzl 2.0M Apr  6 13:33 disk.img

mkdir /tmp/myfs

# mount it
xzl@precision[tmp]$ sudo mount -o loop disk.img /tmp/myfs

xzl@precision[tmp]$ df -hP /tmp/myfs/
Filesystem      Size  Used Avail Use% Mounted on
/dev/loop2      2.0M   21K  1.9M   2% /tmp/myfs

xzl@precision[tmp]$ mount|grep myfs
/tmp/disk.img on /tmp/myfs type ext2 (rw,relatime)
```

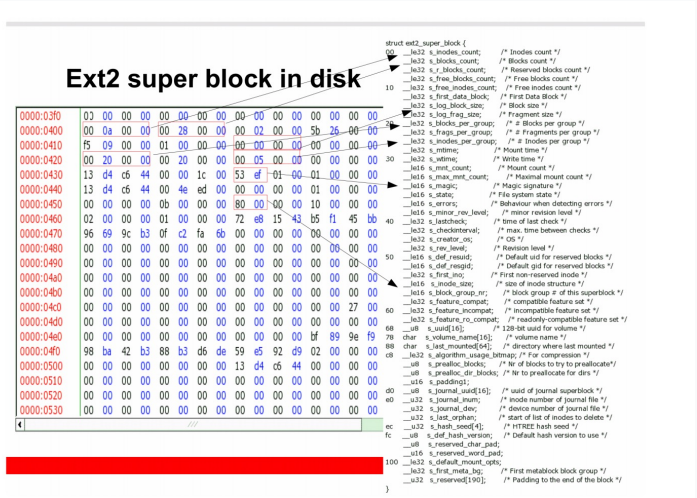
The dir structure




```
xzl@precision[tmp]$ ls -lh /tmp/myfs/
total 14K
-rw-r--r-- 1 root root   10 Apr  6 11:44 aa
lrwxrwxrwx 1 root root    2 Apr  6 12:30 aa-symlink -> aa
drwx------ 2 root root  12K Apr  2 23:53 lost+found
drwxr-xr-x 2 root root 1.0K Apr  6 11:35 testdir
```

Check the superblock

xzl@precision[tmp]$ sudo dumpe2fs /dev/loop2
dumpe2fs 1.42.13 (17-May-2015)
Filesystem volume name: <none>
Last mounted on: /tmp/myfs
Filesystem UUID: 6430eccd-11d0-4ea9-bd26-ce6e946dc02b
Filesystem magic number: **0xEF53**
Filesystem revision #: 1 (dynamic)
Filesystem features: ext_attr resize_inode dir_index filetype sparse_super large_file
Filesystem flags: signed_directory_hash
Default mount options: user_xattr acl
Filesystem state: clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 256
Block count: 2048
Reserved block count: 102
Free blocks: 1990
Free inodes: 244
First block: 1
Block size: 1024
Fragment size: 1024
Reserved GDT blocks: 7
Blocks per group: 8192
Fragments per group: 8192
Inodes per group: 256
Inode blocks per group: 32
Filesystem created: Thu Apr 2 23:53:31 2020
Last mount time: Thu Apr 2 23:56:50 2020
Last write time: Thu Apr 2 23:57:02 2020
Mount count: 2



Maximum mount count: -1
Last checked: Thu Apr 2 23:53:31 2020
Check interval: 0 (<none>)
Lifetime writes: 8 kB
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)
First inode: 11
Inode size: 128
Default directory hash: half_md4
Directory Hash Seed: 15323132-fa4b-4822-ab37-f49b15b57487

Group 0: (Blocks 1-2047)
  Primary superblock at 1, Group descriptors at 2-2
  Reserved GDT blocks at 3-9
  Block bitmap at 10 (+9), Inode bitmap at 11 (+10)
  Inode table at 12-43 (+11)
  1988 free blocks, 242 free inodes, 3 directories
  Free blocks: 59-512, 514-2047
  Free inodes: 15-256

 # we only have 1 block group

an octet = a display “group”

Check the bitmaps

```
# dump inode bitmap. 256 inodes (display in 32 octets). Bitmap at block 11.
xzl@precision[myfs]$ sudo dd if=/dev/loop1 bs=1024 skip=11 count=1 status=none |xxd -b -l 32
00000000: 11111111 00111111 00000000 00000000 00000000 00000000  .?....
00000006: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000000c: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000018: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000001e: 00000000 00000000                                      ..

# dump block bitmap. 2048 blocks (256 display octets). Bitmap at block 10
xzl@precision[myfs]$ sudo dd if=/dev/loop1 bs=1024 skip=10 count=1 status=none |xxd -b -l 256
00000000: 11111111 11111111 11111111 11111111 11111111 11111111  ......
00000006: 11111111 00000011 00000000 00000000 00000000 00000000  ......
0000000c: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000018: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000001e: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000024: 00000000 00000000 00000000 00000000 00000000 00000000  ......
```

inodes

```
# inode 12. testfile “aa”
xzl@precision[myfs]$ sudo debugfs -R "stat <12>" /dev/loop2
Inode: 12   Type: regular    Mode:  0644   Flags: 0x0
Generation: 4138714773    Version: 0x00000001
User:     0   Group:     0   Size: 10
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 2
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x5e8b4e5f -- Mon Apr  6 11:44:31 2020
atime: 0x5e8b4e4d -- Mon Apr  6 11:44:13 2020
mtime: 0x5e8b4e5f -- Mon Apr  6 11:44:31 2020
BLOCKS:
(0):513
TOTAL: 1

# inode 2. root dir
xzl@precision[myfs]$ sudo debugfs -R "stat <2>" /dev/loop2
Inode: 2   Type: directory    Mode:  0755   Flags: 0x0
Generation: 0    Version: 0x00000002
User:     0   Group:     0   Size: 1024
File ACL: 0    Directory ACL: 0
Links: 4   Blockcount: 2
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x5e8b4c40 -- Mon Apr  6 11:35:28 2020
atime: 0x5e8b4c41 -- Mon Apr  6 11:35:29 2020
mtime: 0x5e8b4c40 -- Mon Apr  6 11:35:28 2020
BLOCKS:
(0):44
TOTAL: 1
```

Check root dir content

```
# block content?
xzl@precision[myfs]$ sudo debugfs -R "cat <2>" /dev/loop2 | hexdump -C
debugfs 1.42.13 (17-May-2015)
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  0c 00 02 01 61 61 00 00  0d 00 00 00 c8 03 07 02  |....aa..........|
00000040  74 65 73 74 64 69 72 00  00 00 00 00 00 00 00 00  |testdir.........|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400
```

what is 0x400?

what are the strings on the right?

what is the inodes for directory:

- lost+found

- aa

- testdir

Check root dir content

```
# block content?
xzl@precision[myfs]$ sudo debugfs -R "cat <2>" /dev/loop2 | hexdump -C
debugfs 1.42.13 (17-May-2015)
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  0c 00 02 01 61 61 00 00  0d 00 00 00 c8 03 07 02  |....aa..........|
00000040  74 65 73 74 64 69 72 00  00 00 00 00 00 00 00 00  |testdir.........|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400
```

what is 0x400?
what are the strings on the right?
what is the inodes for directory:
- lost+found
- aa
- testdir

```
struct ext2_dir_entry_2 {
        __u32   inode;
        /* Inode number */
        __u16   rec_len;
        /* Directory entry length */
        __u8    name_len;
        /* Name length */
        __u8    file_type;
        char    name[EXT2_NAME_LEN];
        /* File name */
};
```

Check root dir content

```
# block content?
xzl@precision[myfs]$ sudo debugfs -R "cat <2>" /dev/loop2 | hexdump -C
debugfs 1.42.13 (17-May-2015)
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  0c 00 02 01 61 61 00 00  0d 00 00 00 c8 03 07 02  |....aa..........|
00000040  74 65 73 74 64 69 72 00  00 00 00 00 00 00 00 00  |testdir.........|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400
```

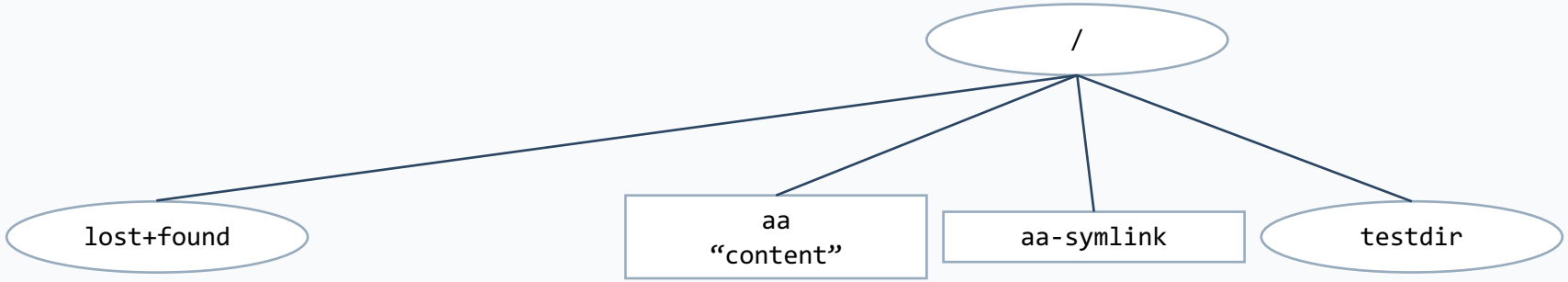
lost+found: 0xb = 11
aa: 0xc = 12
testdir: 0xd = 13

```
struct ext2_dir_entry_2 {
        __u32   inode;
        /* Inode number */
        __u16   rec_len;
        /* Directory entry length */
        __u8    name_len;
        /* Name length */
        __u8    file_type;
        char    name[EXT2_NAME_LEN];
        /* File name */
};
```

Check the dirs

```
xzl@precision[myfs]$ ls -ila
total 54
      2 drwxr-xr-x  4 root root  1024 Apr  6 11:35 .
5636097 drwxrwxrwt 38 root root 36864 Apr  6 11:35 ..
     12 -rw-r--r--  1 root root     0 Apr  2 23:56 aa
     11 drwx------  2 root root 12288 Apr  2 23:53 lost+found
     13 drwxr-xr-x  2 root root  1024 Apr  6 11:35 testdir
```

```
xzl@precision[myfs]$ ls -ila testdir/
total 2
13 drwxr-xr-x 2 root root 1024 Apr  6 11:35 .
 2 drwxr-xr-x 4 root root 1024 Apr  6 11:35 ..
```

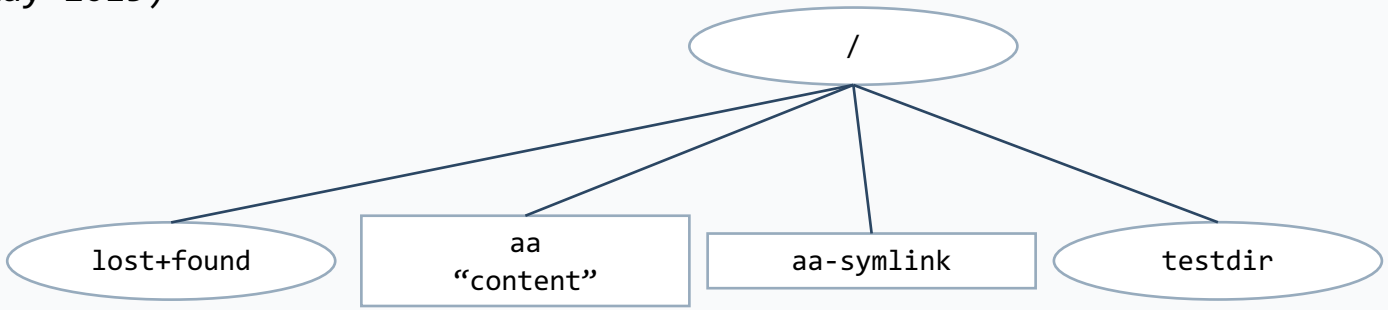


total: the # of blocks

Check root dir content

```
# Showing inodes and type (2=dir, 1=file)
xzl@precision[myfs]$ sudo debugfs -R "ls -l <2>" /dev/loop2
      2   40755 (2)      0      0    1024  6-Apr-2020 11:35 .
      2   40755 (2)      0      0    1024  6-Apr-2020 11:35 ..
     11   40700 (2)      0      0   12288  2-Apr-2020 23:53 lost+found
     12  100644 (1)      0      0       0  2-Apr-2020 23:56 aa
     13   40755 (2)      0      0    1024  6-Apr-2020 11:35 testdir
```

```
# dump the block used by the root dir

# block id?
xzl@precision[myfs]$ sudo debugfs -R "blocks /." /dev/loop2
debugfs 1.42.13 (17-May-2015)
44
```

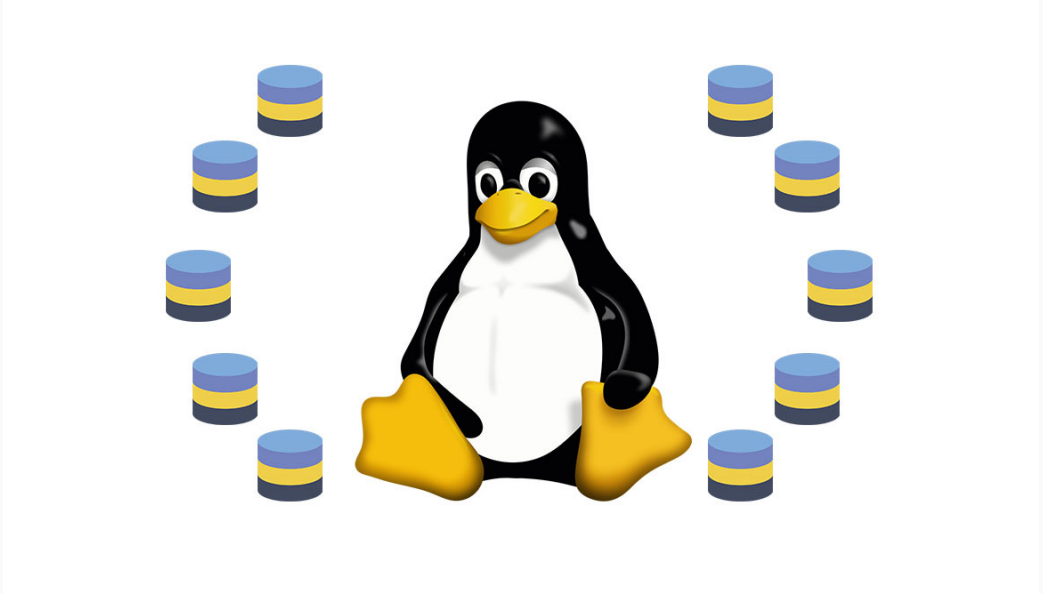


Check data file content

```
xzl@precision[myfs]$ sudo debugfs -R "blocks <12>" /dev/loop2
debugfs 1.42.13 (17-May-2015)
513

# direct inspect disk image
xzl@precision[myfs]$ sudo dd if=/dev/loop2 bs=1024 skip=513 count=1 status=none |hexdump -C
00000000  6d 79 63 6f 6e 74 65 6e  74 0a 00 00 00 00 00 00  |mycontent.......|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400
```

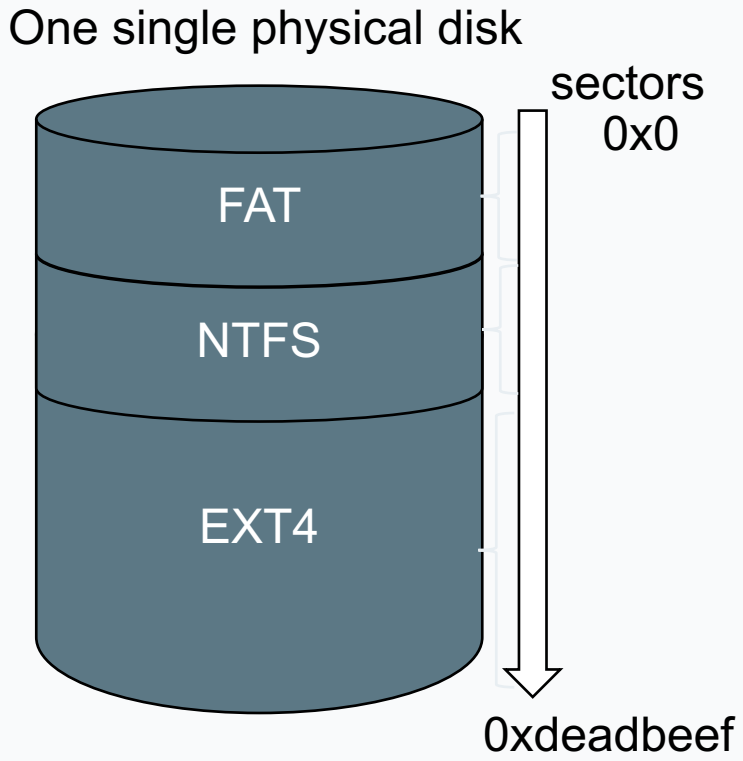
Fun stuff about file systems

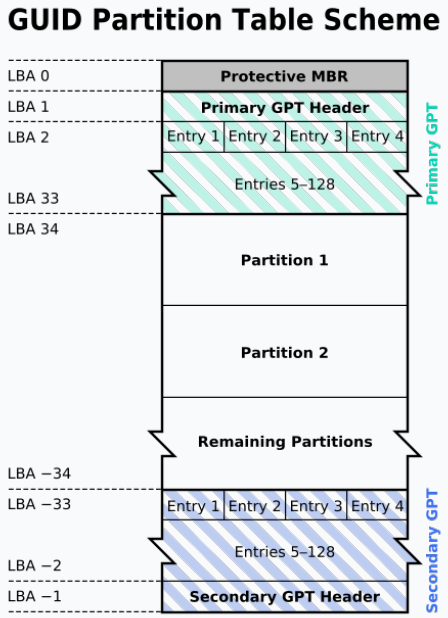
- The *(most)* practical side of the OS course!
  - You will be very popular among your friends and relatives :)
- Partitions 分区
- Loop device 环回设备
- Defragmentation 碎片整理
- Data recovery 数据恢复
- steganography 隐写



Partitions 分区

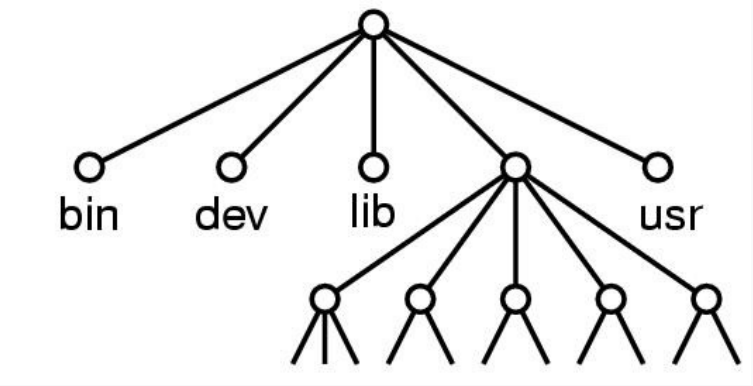
- Partition ==> Disk address space partitioning
- Each partition starts from logical block 0
  - One layer of indirection
  - Virtualizaiton
- Each partition can have itself own file system
- A partition table stores which partition starts from what **phyiscal** block
  - Similar to linear mapping of memory address space
- **Why matter?**
  - **Install new OSes, Dual boot, Boot issues**





Loop device 环回设备

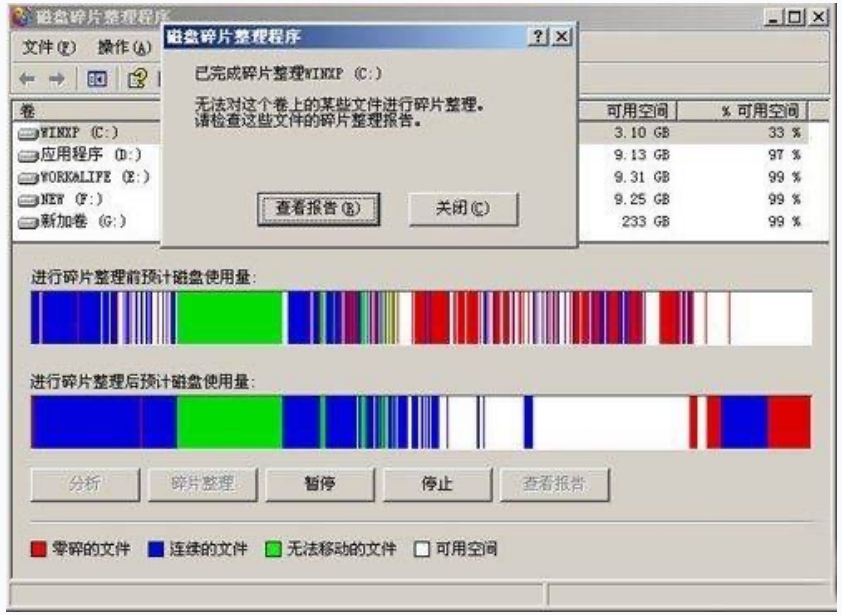
- Have you ever met .iso files, e.g.
  - Windows installer .iso
  - Matlab installer .iso
  - CTex installer .iso
- These files are ***disk images***
- Commodity OSes support treating files as storage devices, i.e. **loop device**
  - **Just an array of disk blocks**
- To maniuplate, mount the file as a loop device
- This is how .dmg on a Mac works!
- This is also how **刷机** works



Mount -o loop mycd.iso /mnt

Defragmentation 磁盘碎片整理

- A very practical tool to improve speed at my time!
  - At the moment, I did not know why it works
- Fragmentation 磁盘碎片
  - Caused by discontiguous disk block allocation
  - Leads to file system aging 文件系统老化
- Why it slows down your computer?
  - Hint: the access pattern of a large file
- Defragment 磁盘碎片整理
  - Relocate disk blocks for **locality**
- **Aging is one of the key reasons of your phone getting slower**
- **If you're using Huawei phone, it does defrag very often**



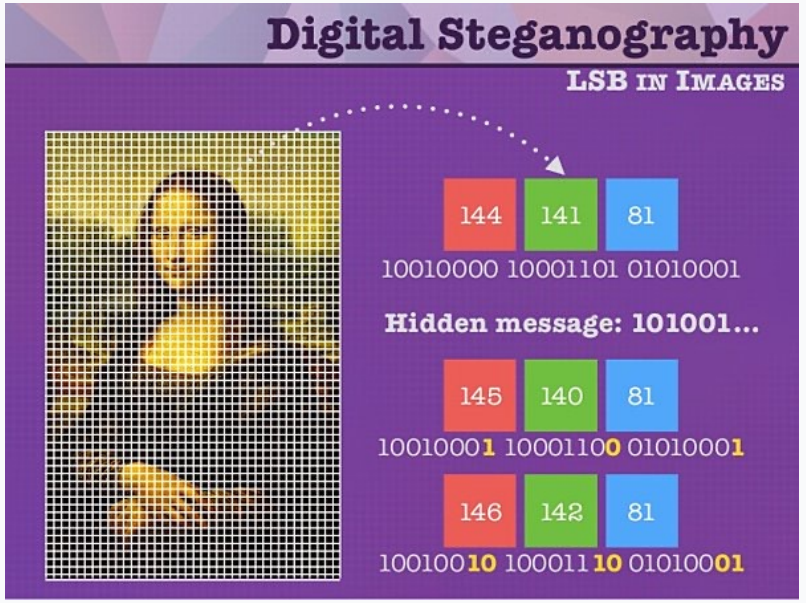
Data recovery 数据恢复

- Happen in many crime movies
  - As in digital forensics
- Possible because your OS/storage is not faithful!
- For performance/longevity reasons, OS/storage may:
  - Delay allocation
  - Delay deletion
- Even worse, deletion may never happen!
  - How?
- **So make sure to erase your data, completely!**

hard disk forensics

Steganography 隐写

- What is steganography?
  - Hide data in hind sight
- A common case is to hide data in pixels
  - Human eyes often cannot distinguish the variation of a single bit of the RGB color
- File system steganography
  - Hide data without file abstraction
- Possible approaches
  - Use hidden partition (erase entry from partition table)
  - Leverage metadata!
- **Donot try to steal information from your employer :(**



Hide data in pixel colors

```
struct _ntfs_inode {
  uint64 mft_no;
  MFT_RECORD *mrec;
  ntfs_volume *vol;
  unsigned long state;
  FILE_ATTR_FLAGS flags;
  uint32 attr_list_size;
  uint8 *attr_list;

  uint32 nr_extents;
  union {
    ntfs_inode **extent_nis;
    ntfs_inode *base_ni;
  };

  uint64 data_size;
  uint64 allocated_size;

  ntfs_time creation_time;
  ntfs_time last_data_change_time;
  ntfs_time last_mft_change_time;
  ntfs_time last_access_time;

  uint32 owner_id;
  uint32 security_id;
  uint64 quota_charged;
  uint64 usn;
};
```

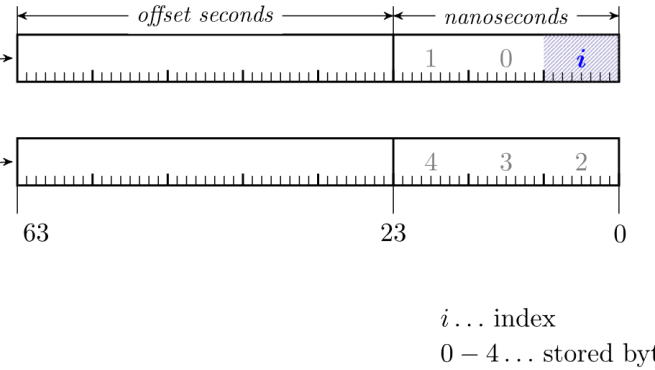
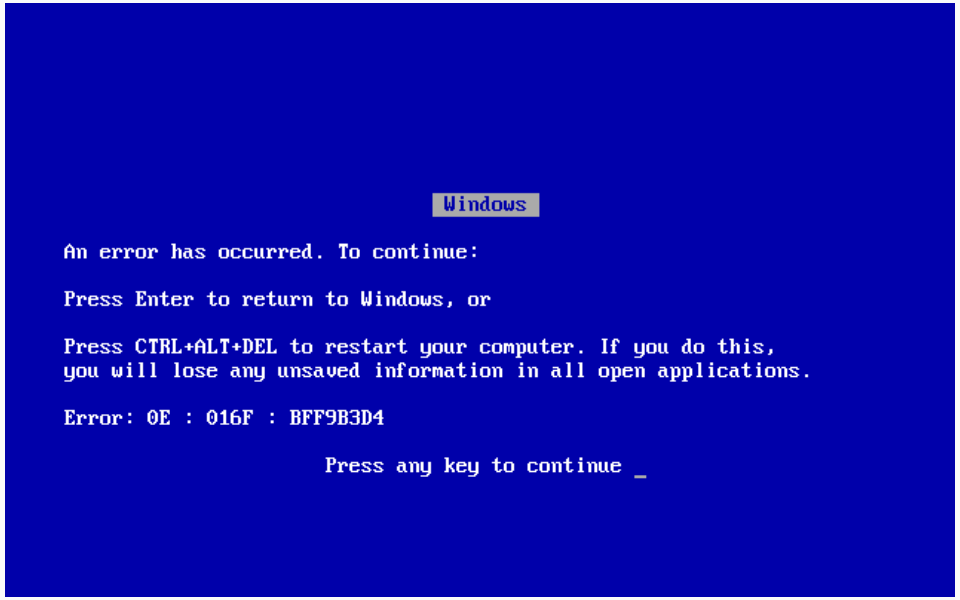


Fig. 2. Overview of storing data in the nanoseconds part of the timestamp fields.

Advanced features: journaling

- Have you met the following?
  - Unsaved WORD files due to program crashes
  - Sudden power failure in the middle of a team fight
- Failures are common in computer systems
- Disastratous for your file systems to fail, e.g. you can no longer access stored data!

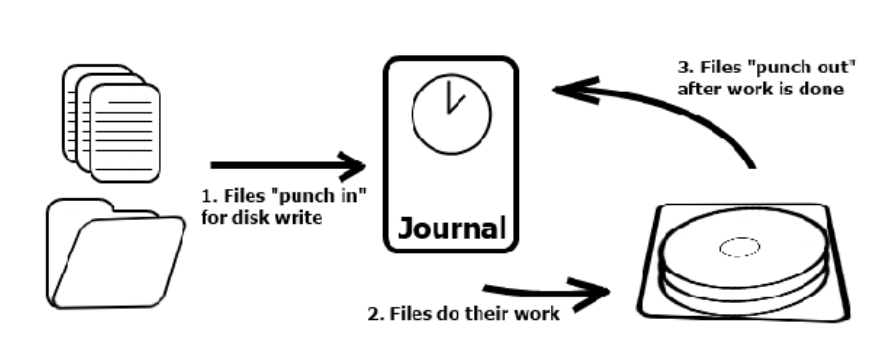


Journaling file systems 日志文件系统

- Became popular ~2002, but date to early 80's
- There are several options that differ in their details
  - Ntfs (Windows), Ext4 (Linux), ReiserFS (Linux), XFS (Irix), JFS (Solaris)
- Basic idea
  - update metadata, or all data, *transactionally*
    - *"all or nothing" -- data journaling (slow) vs metadata journaling (fast)*
    - *Failure atomicity*
  - if a crash occurs, you may lose a bit of work, but the disk will be in a consistent state
    - more precisely, you will be able to quickly get it to a consistent state by using the transaction log/journal – rather than scanning every disk block and checking sanity conditions

Undo/Redo log

- Log: an append-only file containing log records
  - <start t>
    - transaction t has begun
  - <t,x,v>
    - transaction t has updated block x and its new value is v
      - Can log block "diffs" instead of full blocks
      - Can log *operations* instead of data
  - <commit t>
    - transaction t has committed – updates will survive a crash
- Committing involves writing the records – the home data needn't be updated at this time
- Logs are often kept in a separation partition
- Once transactions are committed, logs can be cleaned up!



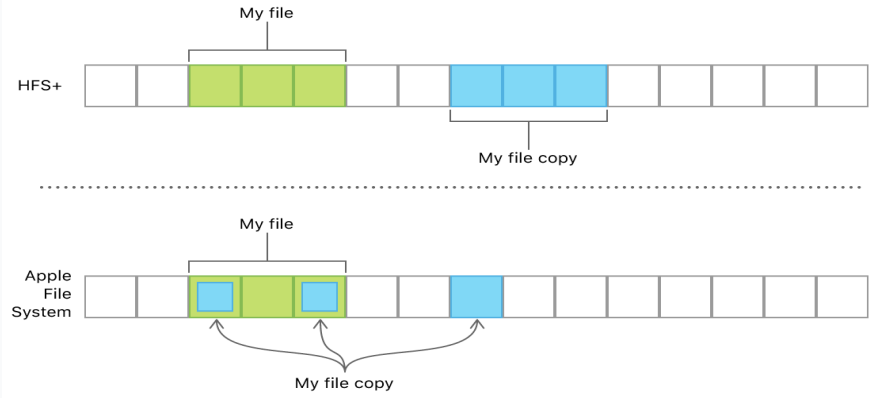
If a crash occurs

- Open the log and parse
  - <start> <commit> => committed transactions
  - <start> no <commit> => uncommitted transactions
- Redo committed transactions
  - Re-execute updates from all committed transactions
  - Aside: note that update (write) is *idempotent*: can be done any positive number of times with the same result.
- Undo uncommitted transactions
  - Undo updates from all uncommitted transactions
  - Write “compensating log records” to avoid work in case we crash during the undo phase

Advanced features: Copy-on-Write (CoW)

- Many data files are replicated, e.g.
  - Just copy the file around but forget to delete
  - As back ups
- It will be a huge waste of time to faithfully replicate all file data as-is
- Basic idea:
  - Allocate new inode for the new file
  - Point all data blocks of the original file to the new file
  - Once the _shared_ data block is modified, write it to a new location
- Apple File System is a CoW file system

CoW is a file system feature



File system research: file system

- Different types of file systems, targeting different **workloads**
- E.g.
  - F2FS (Samsung)
    - Focused on improving sequential access and wear-leveling of flash storage
  - EROFS (Huawei)
    - Focused on improving compression on a read-only file system, e.g. system partition

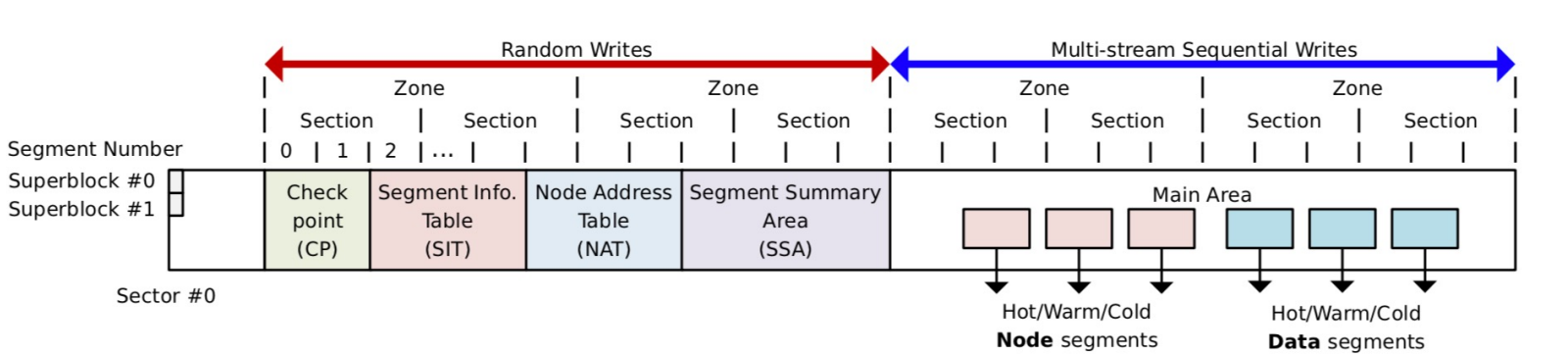
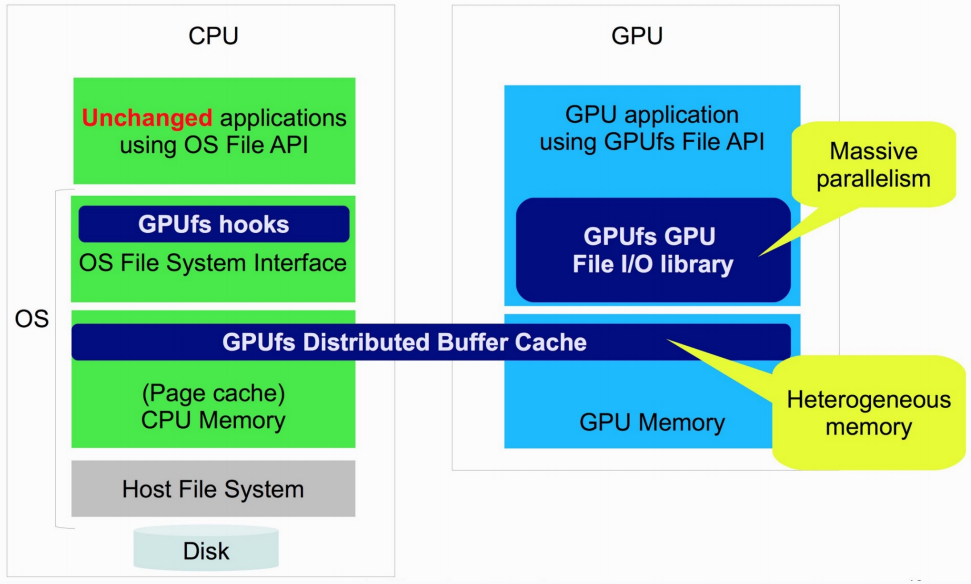
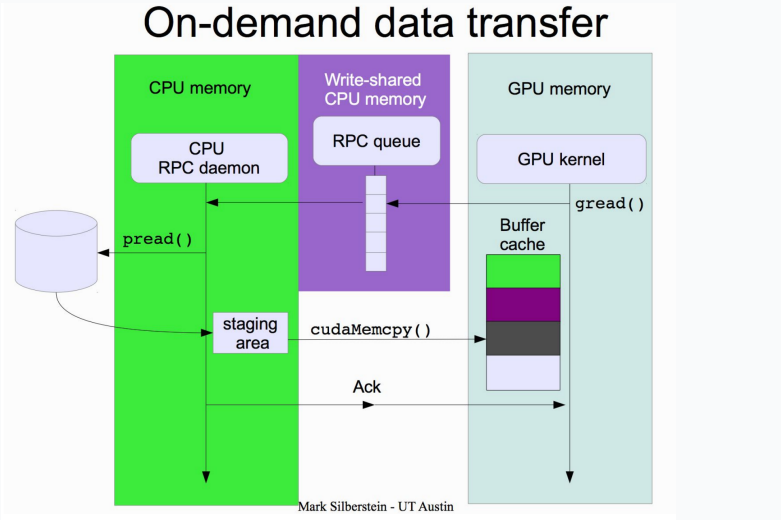


Figure 1: On-disk layout of F2FS.

File system research: GPUFS, ASPLOS'13

- Motivation: enable GPU access to files
  - Must delegate to OS (slow)
- Idea: GPUfs
  - Massive parallelism w/ CUDA
  - High memory bandwidth (e.g. >1000GB/s)
- Results:
  - Tasks: image search & text search
  - ~7X improvements over CPUx8
- Nits:
  - Poor rand access performance
  - Limited to throughputs





Takeaways of today

- File systems: the manager of disk address space
  - On-disk data layout, etc.
- Practical lessons of file systems
  - Partitions, loop devices, defragmentation, data recovery/forensics, steganography
- Advanced file system features:
  - Journaling
  - Copy-on-Write
- File system research
  - Design file systems w.r.t. workloads!

Thank you!