CSCI3150 - FILE II

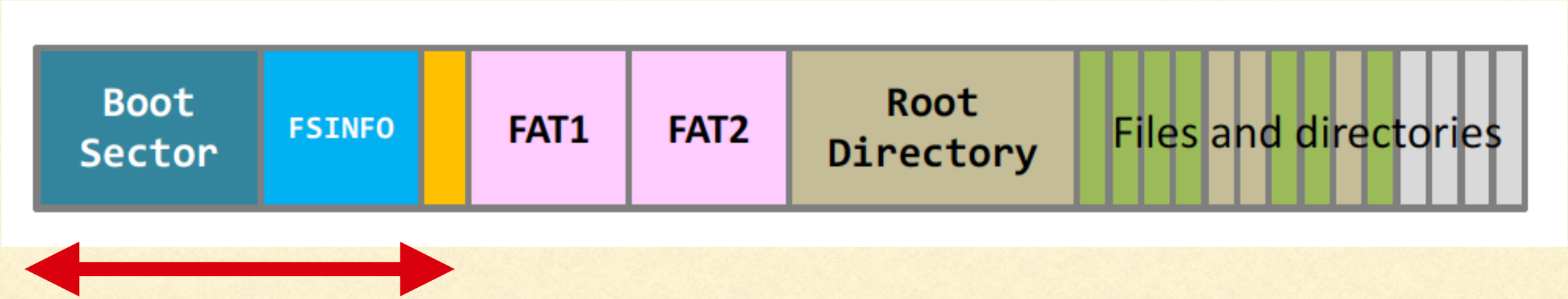
Calvin Kam (hckam@cse)

AGENDA

- FAT32 Overview
- Accessing it using C
  - C Headers
  - 8.3 File Format
  - Traversing Clusters
  - Finding next cluster using FAT table.

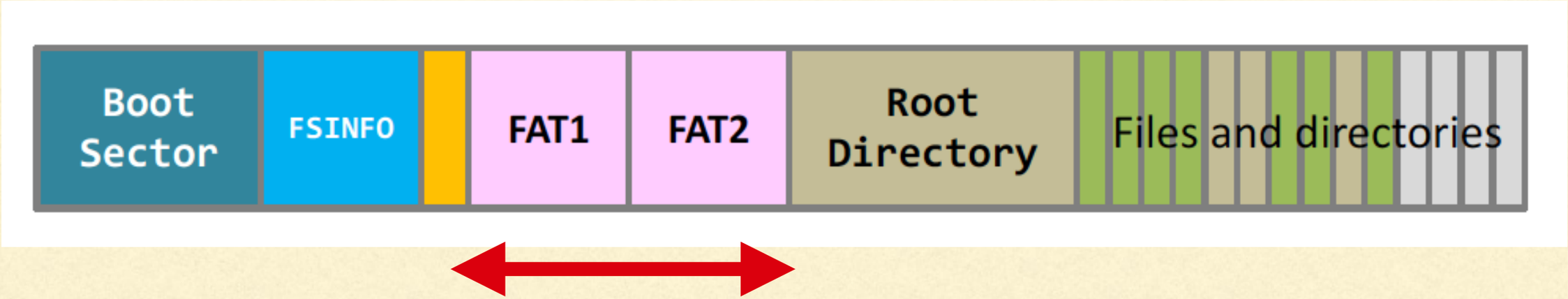
FAT32 OVERVIEW

- Reserved Area (In term of sectors)
  - Boot sector - contains information on FS (eg. sector size)
  - File System Info



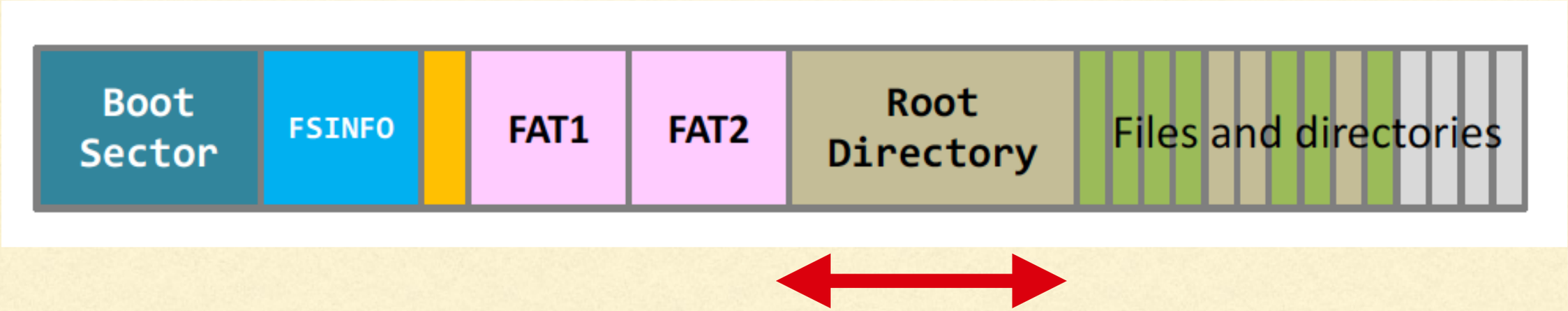
FAT32 OVERVIEW

- File Allocation Table (FAT Table)
  - A linked list pointing to the data cluster of a file.
  - With the current cluster number, it can tell you the next cluster number.



FAT32 OVERVIEW

- Root Directory
  - Entries of files and subdirectories at the very top level of FS.
  - We begin our search here.



ACCESSING FAT32 USING C

- Now we will go into some programming for studying FAT32.
- A handy header: `<linux/msdos_fs.h>`
- Actual location: `/usr/include/linux/msdos_fs.h`
- Contains two important structures:
  - `fat_boot_sector`
  - `msdos_dir_entry`

HEADER: BOOT SECTOR

```
struct fat_boot_sector {
__u8 ignored[3]; /* Boot strap short or near jump */
__u8 system_id[8]; /* Name - can be used to special case
partition manager volumes */
__u8 sector_size[2]; /* bytes per logical sector */
__u8 sec_per_clus; /* sectors/cluster */
__le16 reserved; /* reserved sectors */
__u8 fats; /* number of FATs */
__u8 dir_entries[2]; /* root directory entries */
__u8 sectors[2]; /* number of sectors */
__u8 media; /* media code */
__le16 fat_length; /* sectors/FAT */
__le16 secs_track; /* sectors per track */
__le16 heads; /* number of heads */
__le32 hidden; /* hidden sectors (unused) */
__le32 total_sect; /* number of sectors (if sectors == 0) */

/* The following fields are only used by FAT32 */
__le32 fat32_length; /* sectors/FAT */
__le16 flags; /* bit 8: fat mirroring, low 4: active fat */
__u8 version[2]; /* major, minor filesystem version */
__le32 root_cluster; /* first cluster in root directory */
__le16 info_sector; /* filesystem info sector */
__le16 backup_boot; /* backup boot sector */
__le16 reserved2[6]; /* Unused */
};
```

HOW TO USE IT?

```
FILE *fp = NULL;
struct fat_boot_sector boot_entry;

fp = fopen(device_name, "r+");
if(fp == NULL)
   exit(-1);
uint32_t boot_entry_size = fread(&boot_entry, 1,
sizeof(struct fat_boot_sector), fp);
```

Dst #Data Items

Size Src

HOW TO USE IT?

Access it in structure style

```
disk_info->spc = boot_entry.sec_per_clus;
disk_info->reserved_sectors = boot_entry.reserved;

uint16_t bps = boot_entry.sector_size[0] + ((uint16_t) boot_entry.sector_size[1] << 8);
```

00 02

Each hexadecimal number: 4 bit (2^4)
Shift two digits to left: shift **8 bits**

FILE SYSTEM PARAMETERS

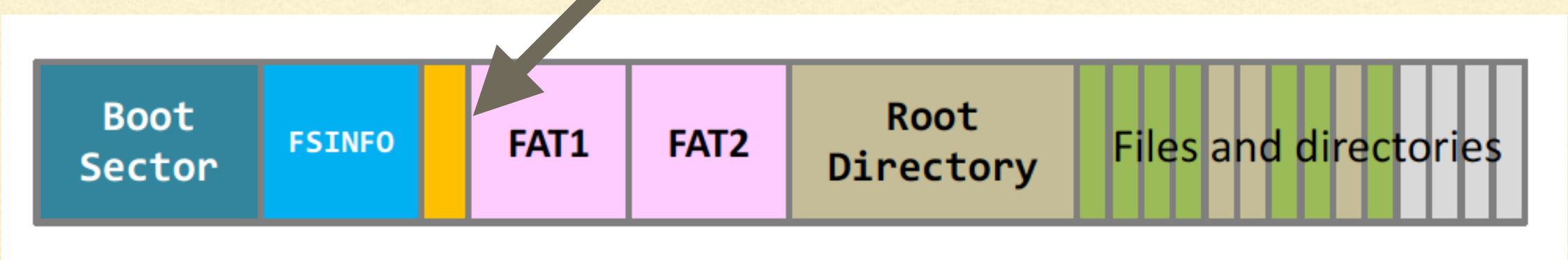
- Items that you can obtain directly

  1. Sectors per cluster (sec_per_clus)
  2. Number of Reserved Area
  3. Number of FATs
  4. Sector Per FATs

FILE SYSTEM PARAMETERS

- Items that you can obtain by manipulation/calculation

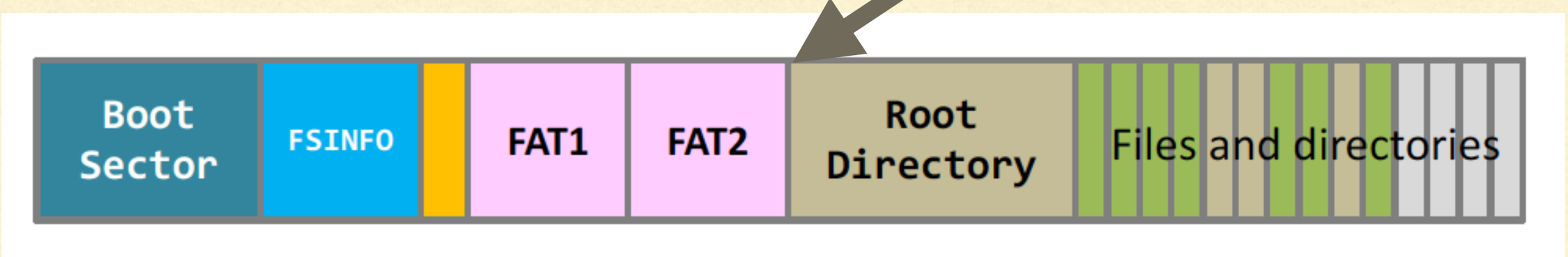
  1. Bytes per sector (By bit shift).
  2. Bytes per cluster (Bytes per sector * sector per cluster)
  3. First FAT starting position: Bytes per sector * Number of Reserved Sector



FILE SYSTEM PARAMETERS

- Items that you can obtain by manipulation/calculation

  1. Bytes per sector (By bit shift).
  2. Bytes per cluster (Bytes per sector * sector per cluster)
  3. First FAT starting position: Bytes per sector * Number of Reserved Sector
  4. Data Area starting position: (Number of Reserved Sector + Number of FATs * Sector Per FAT) * Bytes per Sector



HEADER: DIRECTORY ENTRY

Constant: Guess the value?

```
struct msdos_dir_entry {
__u8 name[MSDOS_NAME];/* name and extension */
__u8 attr; /* attribute bits */
__u8 lcase; /* Case for base and extension */
__u8 ctime_cs; /* Creation time, centiseconds (0-199) */
__le16 ctime; /* Creation time */
__le16 cdate; /* Creation date */
__le16 adate; /* Last access date */
__le16 starthi; /* High 16 bits of cluster in FAT32 */
__le16 time,date,start;/* time, date and first cluster */
__le32 size; /* file size (in bytes) */
};
```

DIRECTORY ENTRY ATTRIBUTE

- There is a 1-byte of data storing the attribute.
- So what are the attributes?

Bit	Mask	Description
0	0x01	Read-Only
1	0x02	Hidden
2	0x04	System Files
3	0x08	Volume Label
4	0x10	Subdirectory
5	0x20	Archive
6	0x40	Device
7	0x80	Reserved
8	0x0F	Long File Name (LFN)

DIRECTORY ENTRY ATTRIBUTE

```
if((dir_entry.attr & 0x10) != 0) {
        /* Sub Directory */
    }
```

Bit	Mask	Description
0	0x01	Read-Only
1	0x02	Hidden
2	0x04	System Files
3	0x08	Volume Label
4	0x10	Subdirectory
5	0x20	Archive
6	0x40	Device
7	0x80	Reserved
8	0x0F	Long File Name (LFN)

8.3 FILE FORMAT

- “Interesting” filename format by Microsoft.
- Consists of a filename, and an extension.
  - filename, at most 8 characters.
  - extension, at most 3 characters, with a period ‘.’
- Stored in an array with a size of 11.
- If the file is deleted, the first character is changed to 0xe5!

8.3 FILE FORMAT EXAMPLES

Filename	DIR Name[]
FOO	FOO␣␣␣␣␣␣␣␣
FOO.BAR	FOO␣␣␣␣␣BAR
FOO.	FOO␣␣␣␣␣␣␣␣
CSCI3150	CSCI3150␣␣␣
CSCI3150.ASG	CSCI3150ASG
.EXT	illegal!

LOCATING CLUSTER

- After we got the position of the root directory, we can traverse each item.
- How to count numbers of items in a cluster?

LOCATING CLUSTER

- After we got the position of the root directory, we can traverse each item.
- How to count numbers of items in a cluster?
- Bytes per cluster / sizeof(dir_entry)

TRAVERSING CLUSTER

If it is allocated

```
for(uint32_t i = 0; i < bpc / sizeof(struct msdos_dir_entry); i++) {
            struct msdos_dir_entry dir_entry;
            size_t ret = fread(&dir_entry, 1, sizeof(struct msdos_dir_entry), fp);
            if(dir_entry.name[0] != 0x00) {
                if(dir_entry.attr != 0x0f) {
                    if(dir_entry.name[0] == 0xe5) {
                        dir_entry.name[0] = '?';
                    }
                    char filename[14] = { 0 };
                    convert_83filename(dir_entry.name, filename);
                    uint32_t dir_first_cluster = get_dir_first_cluster(&dir_entry);
                    if((dir_entry.attr & 0x10) != 0) {
                        strcat(filename, "/");
                    }
                    printf("%d, ", idx++);
                    printf("%s, ", filename);
                    printf("%d, ", dir_entry.size);
                    printf("%d", dir_first_cluster);
                    printf("\n");
                }
                else {
                    printf("%d, LFN entry\n", idx++);
                }
            }
```

Not LFN?

Deleted?

Sub Directory?

NEXT ?

- After getting all the entries in the cluster, we have to move to the next one.
- How? By checking the FAT table :)

NEXT ?

END: 0F FF FF FF

- How? By checking the FAT table :)

Cluster 2: ROOT

Example of FAT32 table start with several cluster chains

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	F0	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	04	00	00	00
+0010	05	00	00	00	06	00	00	00	07	00	00	00	08	00	00	00
+0020	FF	FF	FF	0F	0A	00	00	00	14	00	00	00	0C	00	00	00
+0030	0D	00	00	00	0E	00	00	00	0F	00	00	00	10	00	00	00
+0040	11	00	00	00	FF	FF	FF	0F	00	00	00	00	FF	FF	FF	0F
+0050	15	00	00	00	16	00	00	00	19	00	00	00	F7	FF	FF	0F
+0060	F7	FF	FF	0F	1A	00	00	00	FF	FF	FF	0F	00	00	00	00
+0070	00	00	00	00	F7	FF	FF	0F	00	00	00	00	00	00	00	00

NEXT?

```
uint32_t next_cluster(FILE *fp, off_t fat_offset, uint32_t current_cluster) {
    uint32_t next_cluster_number = 0;
    fseek(fp, fat_offset + current_cluster * sizeof next_cluster_number,
          SEEK_SET);
    size_t ret = fread(&next_cluster_number, 1, sizeof next_cluster_number, fp);

    return next_cluster_number;
}
```

Example of FAT32 table start with several cluster chains

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
+0000	F0	0 FF	FF	0F	FF	FF	1 FF	0F	FF	FF	2 FF	0F	04	00	00	00
+0010	05	00	00	00	06	00	00	00	07	00	00	00	08	00	00	00
+0020	FF	FF	FF	0F	0A	00	00	00	14	00	00	00	0C	00	00	00
+0030	0D	00	00	00	0E	00	00	00	0F	00	00	00	10	00	00	00

SUMMARY

- FAT32 Overview
- Access File System using C
  - Headers
  - Get all parameters of file system
  - navigate to the cluster
  - find next cluster in FAT table.

NEXT WEEK!

- Memory (Last Tutorial!)
- Add Oil for Assignments/Exam!