CSC369 - Tutorial 8

Ext2 overview (metadata and files)
Starter code explained
Exercises 7 and 8 (no handout, they're online)

Felipe de Azevedo Piovezan, Fall 2017, University of Toronto

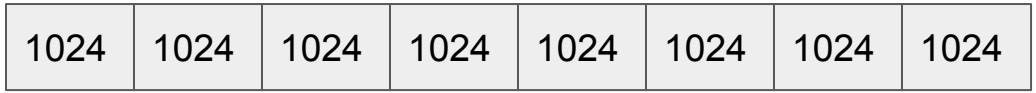
Ext2 overview

- Ext2 splits the HD into **blocks** of 1024 bytes. Think of it as an array of blocks!

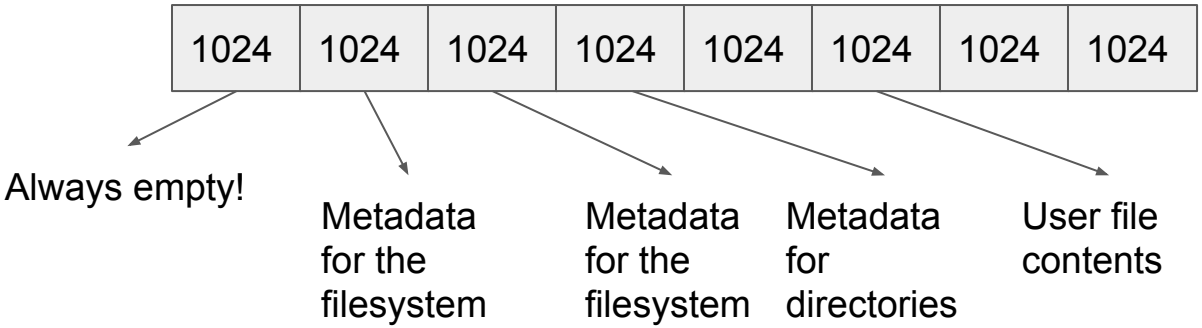
1024	1024	1024	1024	1024	1024	1024	1024

Ext2 overview

- Ext2 splits the HD into **blocks** of 1024 bytes. Think of it as an array of blocks!



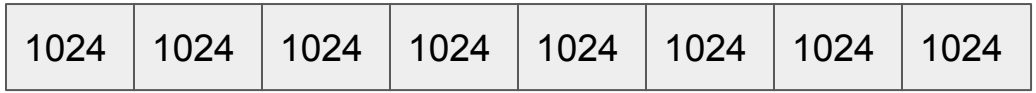
- And assigns different semantics to each of them.



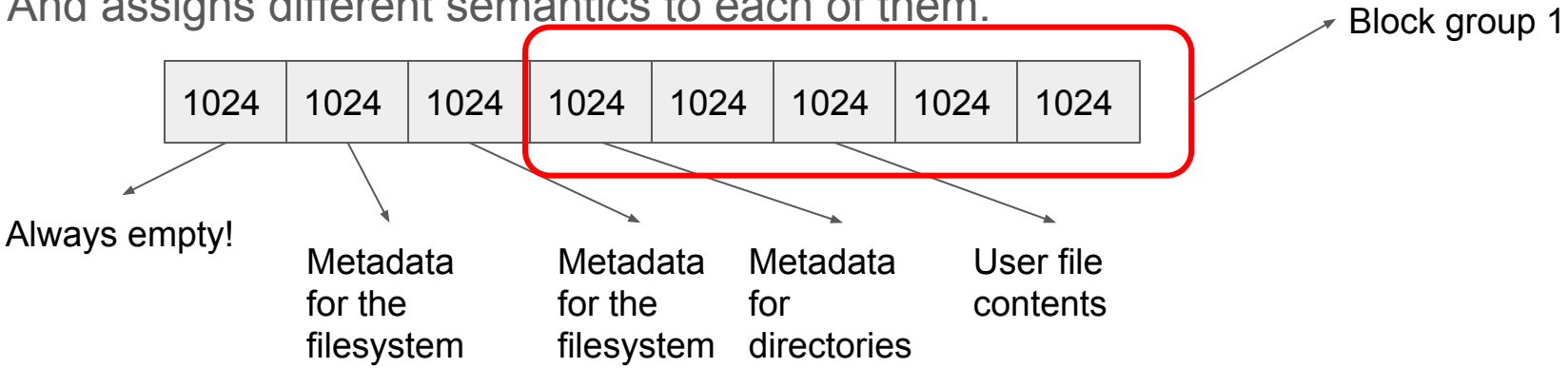
Don’t use this order as reference, this is merely illustrative.

Ext2 overview

- Ext2 splits the HD into **blocks** of 1024 bytes. Think of it as an array of blocks!



- And assigns different semantics to each of them.



- Ext2 groups blocks into block groups
  - For A4, you can assume only one block group.

Ext2 overview

- Each file has an associated **inode.**
  - **This does not contain the file itself!**
- A data structure with information about the file! Which information?

```c
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;        /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;         /* Low 16 bits of Owner Uid */
        unsigned int   i_size;        /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;       /* Access time */
        unsigned int   i_ctime;       /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;       /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;       /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;         /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;      /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;       /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;          /* OS dependent 1 */
        unsigned int   i_block[15];   /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;    /* File ACL */
        unsigned int   i_dir_acl;     /* Directory ACL */
        unsigned int   i_faddr;       /* Fragment address */
        unsigned int   extra[3];
};
```

```c
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignmer
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the ass
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];   /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;    /* File ACL */
        unsigned int   i_dir_acl;     /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

```c
/*
 * Type field for file mode
 */
#define    EXT2_S_IFLNK  0xA000    /* symbolic link */
#define    EXT2_S_IFREG  0x8000    /* regular file */
#define    EXT2_S_IFDIR  0x4000    /* directory */
/* Other types, irrelevant for the assignment */
/* #define EXT2_S_IFSOCK 0xC000 */ /* socket */
/* #define EXT2_S_IFBLK  0x6000 */ /* block device */
/* #define EXT2_S_IFCHR  0x2000 */ /* character device */
/* #define EXT2_S_IFIFO  0x1000 */ /* fifo */
```

```c
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;        /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;         /* Low 16 bits of Owner Uid */
        unsigned int   i_size;        /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;       /* Access time */
        unsigned int   i_ctime;       /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;       /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;       /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;         /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;      /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;       /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;          /* OS dependent 1 */
        unsigned int   i_block[15];   /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;    /* File ACL */
        unsigned int   i_dir_acl;     /* Directory ACL */
        unsigned int   i_faddr;       /* Fragment address */
        unsigned int   extra[3];
};
```

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner Uid */
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];     /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;      /* File ACL */
        unsigned int   i_dir_acl;       /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

What is this? There are important details you **must** know in order to complete this assignment.

```c
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner Uid */
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];     /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;      /* File ACL */
        unsigned int   i_dir_acl;       /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

What is this? There are important details you **must** know in order to complete this assignment.
Check the documentation!!
http://www.nongnu.org/ext2-doc/ext2.html#I-LINKS-COUNT

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner Uid */
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count;   /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];     /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;    /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;      /* File ACL */
        unsigned int   i_dir_acl;       /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

Another pitfall here… check the documentation!

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner Uid */
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];     /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;      /* File ACL */
        unsigned int   i_dir_acl;       /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

The most important part of the inode!
An array of block numbers, telling you in which blocks the **contents** of the file are located!

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner Uid */
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];     /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;      /* File ACL */
        unsigned int   i_dir_acl;       /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

If this is a text file with contents “Hello world”,
and i_block[0] = 10,
then the text is located in the 10th block, byte number 10*1024 = 10240.

```
/*
 * Structure of an inode on the disk
 */
struct ext2_inode {
        unsigned short i_mode;          /* File mode */
        /* Use 0 as the user id for the assignment. */
        unsigned short i_uid;           /* Low 16 bits of Owner Uid */
        unsigned int   i_size;          /* Size in bytes */
        /* You don't need to set access time for the assignment. */
        unsigned int   i_atime;         /* Access time */
        unsigned int   i_ctime;         /* Creation time */
        /* You don't need to set modification time for the assignment. */
        unsigned int   i_mtime;         /* Modification time */
        /* d_time must be set when appropriate */
        unsigned int   i_dtime;         /* Deletion Time */
        /* Use 0 as the group id for the assignment. */
        unsigned short i_gid;           /* Low 16 bits of Group Id */
        unsigned short i_links_count; /* Links count */
        unsigned int   i_blocks;        /* Blocks count IN DISK SECTORS*/
        /* You can ignore flags for the assignment. */
        unsigned int   i_flags;         /* File flags */
        /* You should set it to 0. */
        unsigned int   osd1;            /* OS dependent 1 */
        unsigned int   i_block[15];     /* Pointers to blocks */
        /* You should use generation number 0 for the assignment. */
        unsigned int   i_generation;  /* File version (for NFS) */
        /* The following fields should be 0 for the assignment.  */
        unsigned int   i_file_acl;      /* File ACL */
        unsigned int   i_dir_acl;       /* Directory ACL */
        unsigned int   i_faddr;         /* Fragment address */
        unsigned int   extra[3];
};
```

Read the documentation, there is much more to this than what I've just described.

Ext2 overview

- Each file has an associated **inode.**
  - **This is does not contain the file itself!**
- A data structure with information about the file! Which information?
- For each block group, Ext2 will allocate a constant number of consecutive blocks to serve as an array of inodes: the inode table.
  - (notice each inode has a fixed size)

Ext2 overview

- Each file has an associated **inode.**
  - **This is does not contain the file itself!**
- A data structure with information about the file! Which information?
- For each block group, Ext2 will allocate a constant number of consecutive blocks to serve as an array of inodes: the inode table.
  - (notice each inode has a fixed size)
  - In A4, what does that mean for the number of inodes available?

Ext2 overview

- Each file has an associated **inode.**
  - **This is does not contain the file itself!**
- A data structure with information about the file! Which information?
- For each block group, Ext2 will allocate a constant number of consecutive blocks to serve as an array of inodes: the inode table.
  - (notice each inode has a fixed size)
  - In A4, what does that mean for the number of inodes available?
  - You have a very limited number of inodes, and therefore a maximum number of files you can store!
  - Some inodes are reserved… documentation!

Ext2 overview

- Suppose you have a pointer to your disk:

```
char *disk;   //Suppose, for now, that your disk is in memory.
```

Ext2 overview

- Suppose you have a pointer to your disk:

```
char *disk;   //Suppose, for now, that your disk is in memory.
```

- Suppose your inode table is at block 10,

```
struct ext2_inode *inodes = (struct ext2_inode *)(disk + 1024 * 10);
```

Ext2 overview

- Suppose you have a pointer to your disk:

```
char *disk;   //Suppose, for now, that your disk is in memory.
```

- Suppose your inode table is at block 10,

```
struct ext2_inode *inodes = (struct ext2_inode *)(disk + 1024 * 10);
```

- and you want to print the contents of the 7th inode…

```
printf("Size in bytes = %d", inodes[6].i_size);
```

Ext2 overview

- Suppose Ext2 has 32 inodes in total. When you create a file, you need grab a free inode to store the file’s metadata.

Ext2 overview

- Suppose Ext2 has 32 inodes in total. When you create a file, you need grab a free inode to store the file's metadata.
- How do we find a free inode?
  - Iterate over the inode table? But the inode structure doesn't have a member indicating whether it is associated with a file… It could be filled with garbage values...

Ext2 overview

- Suppose Ext2 has 32 inodes in total. When you create a file, you need grab a free inode to store the file's metadata.
- How do we find a free inode?
  - Iterate over the inode table? But the inode structure doesn't have a member indicating whether it is associated with a file… It could be filled with garbage values...
- Ext2 allocates a block to be the inode bitmap!
  - A vector of bits, where inode_bitmap[i] = 1 iff the i-th inode is in use.

Ext2 overview

- Suppose your inode bitmap is located at block 4.

Ext2 overview

- Suppose your inode bitmap is located at block 4.

```
unsigned char *inode_bits = (unsigned char *)(disk + 1024 * 4);
```

Ext2 overview

- Suppose your inode bitmap is located at block 4.

```
unsigned char *inode_bits = (unsigned char *)(disk + 1024 * 4);
```

- From the documentation:
  - *The first inode of this block group is represented by bit 0 of byte 0, the second by bit 1 of byte 0. The 8th inode is represented by bit 7 (most significant bit) of byte 0 while the 9th inode is represented by bit 0 (least significant bit) of byte 1.*

Ext2 overview

- Suppose your inode bitmap is located at block 4.

```
unsigned char *inode_bits = (unsigned char *)(disk + 1024 * 4);
```

- From the documentation:
  - *The first inode of this block group is represented by bit 0 of byte 0, the second by bit 1 of byte 0. The 8th inode is represented by bit 7 (most significant bit) of byte 0 while the 9th inode is represented by bit 0 (least significant bit) of byte 1.*
- If you have 32 inodes, you are looking at the first 32 bits, i.e., the first 4 bytes of the bitmap:

```
for (byte = 0; byte < 32 / 8; byte++)
        for (bit = 0; bit < 8; bit++)
                in_use = inode_bits[byte] & (1 << bit);
```

Ext2 overview

- Recall that file contents are also stored in blocks.
- So when you add a new file, you also need find free blocks for it.

Ext2 overview

- Recall that file contents are also stored in blocks.
- So when you add a new file, you also need find free blocks for it.
- Just like for inodes, Ext2 will allocate a special block to be the block bitmap!

Ext2 overview

- How do we find:
  - The inode table?
  - The block bitmap?
  - The inode bitmap?

Ext2 overview

- How do we find:
  - The inode table?
  - The block bitmap?
  - The inode bitmap?

Ext2 overview

- How do we find:
  - The inode table?
  - The block bitmap?
  - The inode bitmap?
- Ext2 has a special block called block group descriptor table.
  - An array of block group descriptors

Ext2 overview

- How do we find:
  - The inode table?
  - The block bitmap?
  - The inode bitmap?
- Ext2 has a special block called block group descriptor table.
  - An array of block group descriptors

Ext2 overview

- How do we find:
  - The inode table?
  - The block bitmap?
  - The inode bitmap?
- Ext2 has a special block called block group descriptor table.
  - An array of block group descriptors
  - Recall that there is only one block group in the disks used in A4.

Ext2 overview

```
/*
 * Structure of a blocks group descriptor
 */
struct ext2_group_desc
{
        unsigned int    bg_block_bitmap;        /* Blocks bitmap block */
        unsigned int    bg_inode_bitmap;        /* Inodes bitmap block */
        unsigned int    bg_inode_table;         /* Inodes table block */
        unsigned short bg_free_blocks_count; /* Free blocks count */
        unsigned short bg_free_inodes_count; /* Free inodes count */
        unsigned short bg_used_dirs_count;   /* Directories count */
        /* The pad and reserved fields should be 0 for the assignment. */
        unsigned short bg_pad;
        unsigned int    bg_reserved[3];
};
```

Ext2 overview

- How do we find:
  - The inode table?
  - The block bitmap?
  - The inode bitmap?
- Ext2 has a special block called block group descriptor table.
  - An array of block group descriptors
  - Recall that there is only one block group in the disks used in A4.
- How do we find this block?
  - It is, for this assignment, the third block in the disk.

Ext2 overview

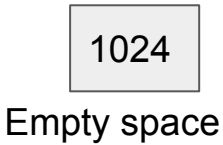
- There is one last important data structure: the superblock.
- It contains information about the whole filesystem:
  - number available inodes/blocks across all groups,
  - constants related to sizes, etc.
  - the code is too big to paste here, but it is the **ext2_super_block** struct in the header.

Ext2 overview

- What does your typical A4 disk image actually look like:

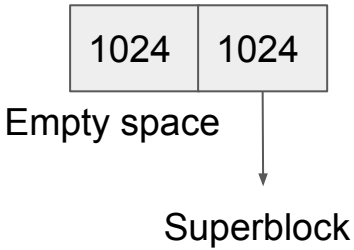
Ext2 overview

- What does your typical A4 disk image actually look like:



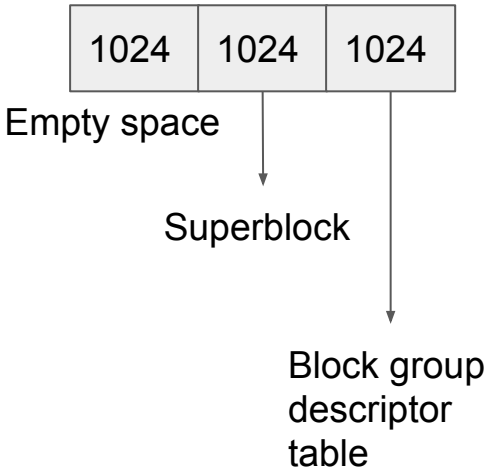
Ext2 overview

- What does your typical A4 disk image actually look like:



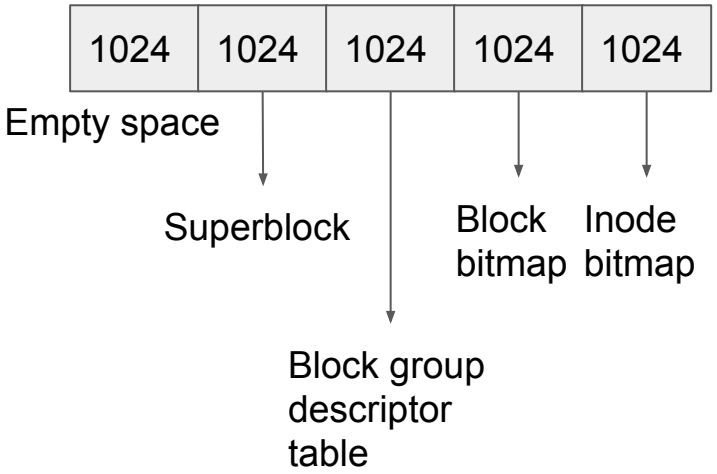
Ext2 overview

- What does your typical A4 disk image actually look like:



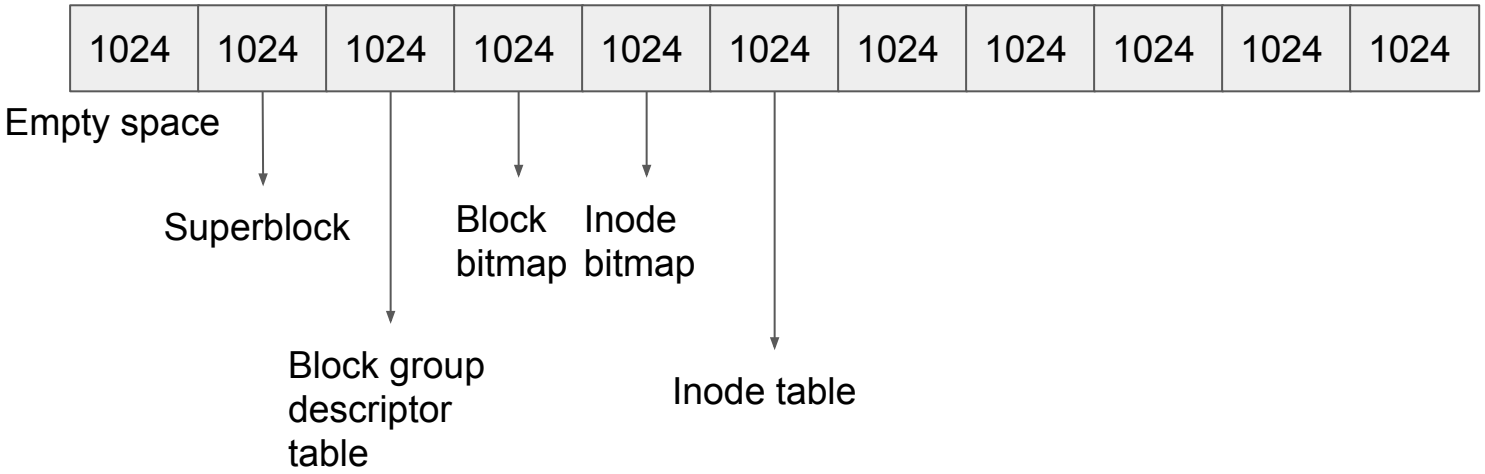
Ext2 overview

- What does your typical A4 disk image actually look like:



Ext2 overview

- What does your typical A4 disk image actually look like:



char *disk;

Starter code

```
13 int main(int argc, char **argv) {
14
15     if(argc != 2) {
16         fprintf(stderr, "Usage: %s <image file name>\n", argv[0]);
17         exit(1);
18     }
19     int fd = open(argv[1], O_RDWR);
20
21     disk = mmap(NULL, 128 * 1024, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
22     if(disk == MAP_FAILED) {
23         perror("mmap");
24         exit(1);
25     }
26
27     struct ext2_super_block *sb = (struct ext2_super_block *)(disk + 1024);
28     printf("Inodes: %d\n", sb->s_inodes_count);
29     printf("Blocks: %d\n", sb->s_blocks_count);
30
31     return 0;
32 }
```

Exercises 7 and 8

- Print some attributes of the block group descriptor.
  - Exercise 7
- Print some attributes of each inode that is being used.
  - Exercise 8
  - Some inodes are reserved and should be ignored. Which ones? Look at the documentation.

Looking at dumps: debugging your code

- It will be useful to examine the images you create/modify using external tools.
- Dump the files emptydisk.img and onefile.img
  - xxd -c 1 emptydisk.img emptydisk.dump
  - xxd -c 1 onefile.img onefile.dump
- This dumps one byte per line. You can change this, but I find it useful to have line numbers matching the byte number you have.

Looking at dumps: debugging your code

- It will be useful to examine the images you create/modify using external tools.
- Dump the files emptydisk.img and onefile.img
  - xxd -c 1 emptydisk.img emptydisk.dump
  - xxd -c 1 onefile.img onefile.dump
- This dumps one byte per line. You can change this, but I find it useful to have line numbers matching the byte number you have.
- Now open your files with:
  - vimdiff emptydisk.dump onefile.dump

Looking at dumps: debugging your code

free blocks count

free inodes count

Mind the endianness!

Vim uses 1 based indexing for lines, so we're one byte off (you can simply delete the first line to fix that…)

Directories

- A directory is represented just like a file:
  - An inode + some data blocks.
- While the data blocks for a file store its contents, a directory data block stores a linked list of directory entries.
- **An entry must ALWAYS have a length that is a multiple of 4.**
  - **The last entry must always occupy the remaining space of the block.**

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- **An entry must ALWAYS have a length that is a multiple of 4.**
  - **The last entry must always occupy the remaining space of the block.**
- This is why we have two “length” fields:
  - To jump to the following record, just do base + rec_len (how do you tell when you have reached the last entry?).
  - To print the name, we use name_len.

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- Suppose some directory stores its contents in block 9.

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- Suppose some directory stores its contents in block 9.
- How do you find the name of the first entry?
  - char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
  - … problems?

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- Suppose some directory stores its contents in block 9.
- How do you find the name of the first entry?
  - char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
  - … problems?
  - Isn’t this “eating” the first 4 bytes (or however many bytes a pointer needs) of the name?

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- Suppose some directory stores its contents in block 9.
- How do you find the name of the first entry?
  - char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
  - Isn’t this “eating” the first 4 bytes (or however many bytes a pointer needs) of the name?
  - No! In C, declaring char name[]; is an indication that the struct will have variable size. That member has “size” 0!

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
- How do we print this? This is **not** NULL terminated.

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
- How do we print this? This is **not** NULL terminated.
  - ext2_dir_entry *entry = (ext2_dir_entry*) disk + 1024* 9;
  - printf("%.*s", entry->name_len, name);

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
- How do we print this? This is **not** NULL terminated.
  - ext2_dir_entry *entry = (ext2_dir_entry*) disk + 1024* 9;
  - printf("%.*s", entry->name_len, name);
- The next entry is located at…?

Directories

```
struct ext2_dir_entry {
        unsigned int   inode;      /* Inode number */
        unsigned short rec_len;    /* Directory entry length */
        unsigned char  name_len;   /* Name length */
        unsigned char  file_type;
        char           name[];     /* File name, up to EXT2_NAME_LEN */
};
```

- char* name = disk + 1024 * 9 + sizeof(ext2_dir_entry)
- How do we print this? This is **not** NULL terminated.
  - ext2_dir_entry *entry = (ext2_dir_entry*) disk + 1024* 9;
  - printf("%.*s", entry->name_len, name);
- The next entry is located at…?
  - ((char*) entry)+ entry->rec_len;

Exercise 9

- Iterate over all inodes.
- If the inode is being used, and if it is related to a directory,
  - Access the directory data blocks and print each directory entry.
- Some pitfalls:
  - How do you detect deleted entries? Do you need to?
  - What if there is a deleted entry as the very first entry of the block?
  - Again… check the documentation for answers (or ask me next week).