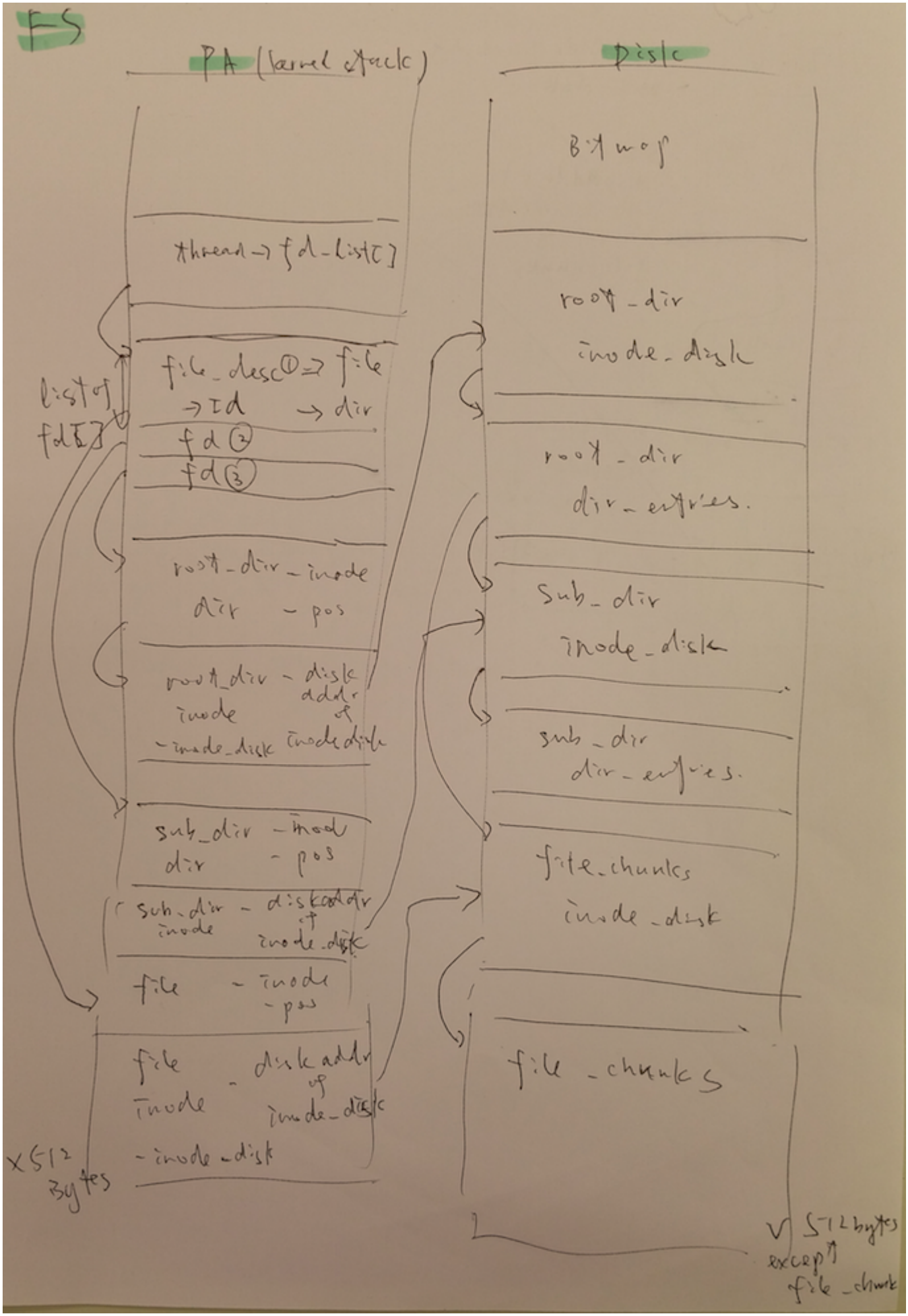
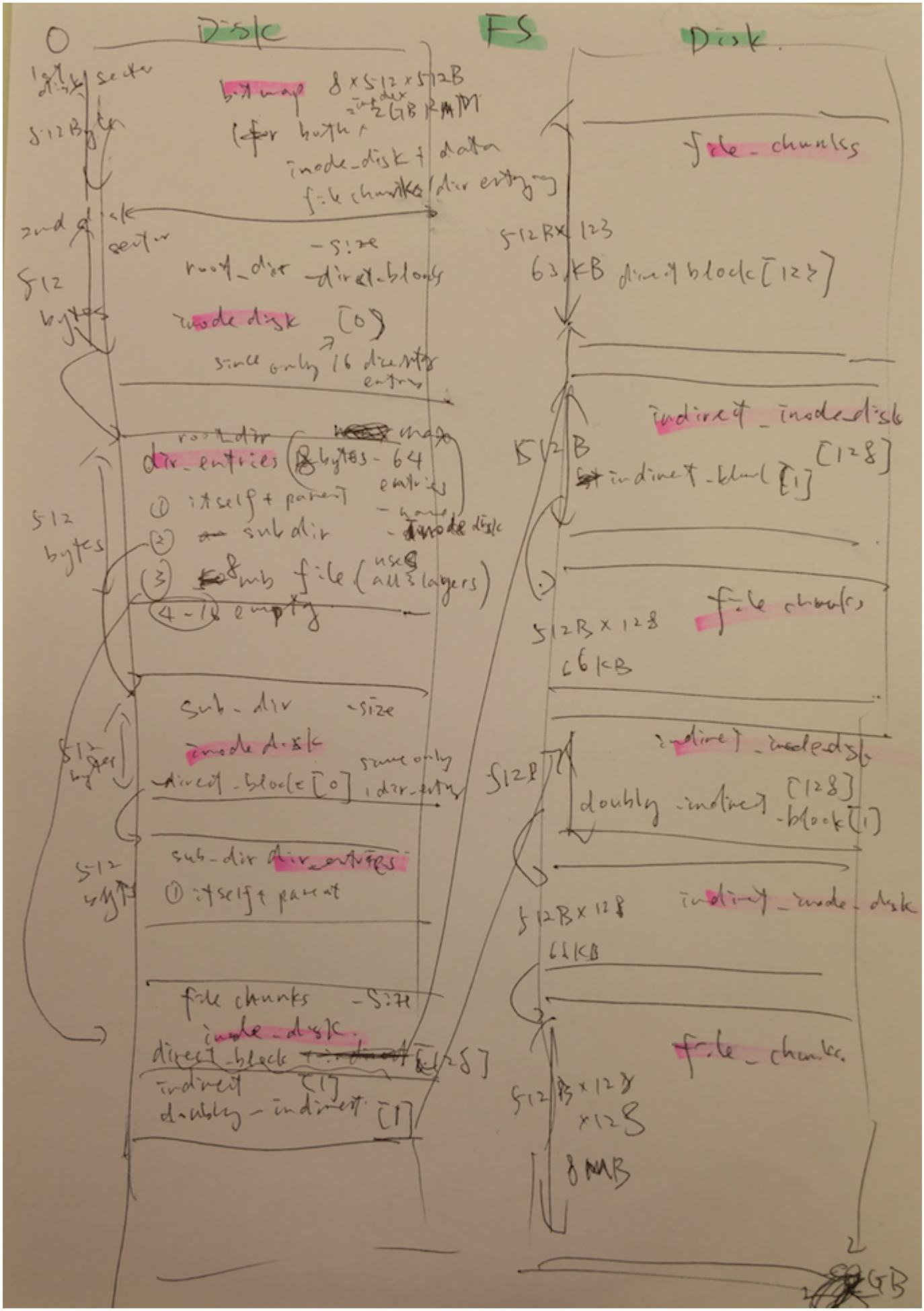
intuition/visualization

```c
#define DIRECT_BLOCKS_COUNT 123
#define INDIRECT_BLOCKS_PER_SECTOR 128

/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
// 1 inode_disk == 1 block sector size i.e. 512 bytes long
// inode_disk == direct blocks + indirect blocks (disk addrs)
// file inode_disk == direct blocks + indirect blocks (disk addrs) of 512 bytes file chunk
// dir inode_disk == direct blocks + indirect blocks (disk addrs) of 64 dir entries (in 512 bytes)
// dir entry == 8 bytes storing disk addr of either subdir / file inode_disk
struct inode_disk
  {
    // Data sectors -> 500 bytes
    // must be pointing at 1 block sector, could be:
    // 1. dir inode disk 2. file inode disk 3. indirect inode disk
    // 3. block size of dir entires 4. block size file chunk
    block_sector_t direct_blocks[DIRECT_BLOCKS_COUNT]; // 4*123 == 492 bytes
    block_sector_t indirect_block; // 4 bytes
    block_sector_t doubly_indirect_block; // 4 bytes

    // remaining part -> 12 bytes
    bool is_dir;
    off_t length;                       /* File size in bytes. */
    unsigned magic;                     /* Magic number. */
  }; // total == 512 bytes

/* In-memory inode. */
struct inode
  {
    struct list_elem elem;              /* Element in inode list. */ // for loop able
    int open_cnt;                       /* Number of openers. */
    bool removed;                       /* True if deleted, false otherwise. */
    int deny_write_cnt;                 /* 0: writes ok, >0: deny writes. */
    block_sector_t sector;              // inode_disk's disk addr / sector no.
    struct inode_disk data;             // 1 block size 512 bytes of indexes to disk addrs
  };
```

```c
// file in RAM == directly pointing at 123 * 512 bytes of file at disk
// dir in RAM == directly pointing at 7872 dir entries at disk
// inode == dir inode_disk (supposedly in disk) in RAM
// dir inode_disk == direct blocks + indirect blocks (disk addrs) of block of 64 dir entries
// direct_block == points at 64 dir entries on 1 block sector (512 bytes)
// direct block alone == could support 123 * 64 = 7872 file / subdir's inode_disk
struct dir // memory
  {
    struct inode *inode;                /* Backing store. */
    off_t pos;                          /* Current position. */
  };

// dir version of file chunk, BUT divided in 64 small pieces n storing subdir / file inode_disk
// dir entry == 1 entry in a directory e.g. file or subdir
struct dir_entry // disk
  {
    block_sector_t inode_disk_diskaddr; // pointing at either subdir / file inode_disk
    char name[NAME_MAX + 1];            // sub_dir name or file name
    bool in_use;                        /* In use or free? */
  };
```



DEMO **inode on RAM and disk**



DEMO **inode on disk**

- **file** -> **file_inode** (RAM) -> **file_inode_disk** (both RAM n disk) -> **file_chunks** (disk)

- **dir** -> **dir_inode** (RAM) -> **dir_inode_disk** (both RAM n disk) -> **dir_entries** (disk) -> **file_inode_disk**/**dir_inode_disk** (both RAM n disk)

HOW DATA ON DISK CHANGES BY KERNEL FUNCTIONS

```
Initial Disk State

init_root_dir()

inode_create(sector,entry_cnt*sizeof(dir_entry))  -> POPULATE, WRITE root_dir inode_disk INTO DISK
1. w() root_dir inode_disk into inodes block 0
2. r() w() bitmap for root_dir's dir_entries
3. w() inodes block 0 to point to data_block 0
4. w() 0s at data block 0 (root_dir dir_entries)

struct dir *dir = dir_open(inode_open(sector)) -> WRITE PARENT DIR + SUB DIR dir_entries
1. for_loop() open_list check inode_disk cache
2. if NO cache, r() inode_disk from inode_disk block 1
3. populate() dir w inode_disk

struct dir_entry de.inode_sector = sector;  -> POPULATE "." ".." dir_entry
inode_write_at(dir->inode, &de, sizeof de, 0) -> WRITE dir_entry TO DISK

## dir_entries at data block 0, w 2 dir_entry inside
## root_dir at data block 0 w 2 dir_entries : {curr_dir, inode block 0},{parent_dir, 0}
inode bitmap  10000000
inodes        [d a:0 r:2] [] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0)] [] [] [] [] [] [] []
```

DEMO `init_root_dir (ROOT_DIR_SECTOR, 16)`

- this is showing layout of hard disk
- bitmap
  - easy way of showing whether 1 block / disk / sector (512 bytes) is free
  - bitmap itself is 1 block / disk / sector (512 bytes) long
  - inode, data bitmap are combined into one single bitmap in pintos
- []
  - 1 block / disk / sector (512 bytes) long on disk
- inodes
  - file / dir inode_disk
  - d: dir inode disk, f : file inode disk, a: disk addr, r : number of files inside
  - 1 block / disk / sector (512 bytes) long
- data
  - 1 file chunk / 64 dir_entries
  - 1 block / disk / sector (512 bytes) long

```
creat("/y");

sys_create("/y") file

struct dir *parent_dir = dir_open_path(dir_path)     -> SEARCH FOR PARENT DIR INODE
1. read() root_dir inode_disk
2. traverse() dir_entries data block for matching dir name
3. if found dir_entry{name, inode}, populate() parent_dir w inode

free_map_allocate(1, &inode_sector)          -> BITMAP ALLOCATE file's inode_disk
1. r() w() bitmap for file's inode_disk

inode_create(inode_sector, initial_size, is_dir) -> POPULATE, WRITE file inode_disk INTO DISK
1. w() file inode_disk into inodes block 1
2. r() w() bitmap for file's file_chunks data block
3. w() inodes block 1 to point to data_block 1
4. w() 0s at data block 1 (reserve file_chunks data)

dir_add(dir, dir_name, inode_sector, is_dir)) -> WRITE PARENT DIR dir_entries
1. w() file inode_disk addr at parent_dir dir_entries at data block 0


BEFORE:
inode bitmap  10000000
inodes        [d a:0 r:2] [] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0)] [] [] [] [] [] [] []

AFTER:
inode bitmap  11000000        -> free_map_allocate(size, &disk_addr_allocated) to store inode_disk
inodes        [d a:0 r:3] [f a:1 r:1] [] [] [] [] [] []     -> PintOS does not update dir size +
inode_create()
data bitmap   11000000        -> inode_reserve()
data          [(.,0) (..,0) (y,1)] [000000000] [] [] [] [] [] []    -> dir_add() + inode_reserve()
```

DEMO create file

- inodes == inode disk
  - 1st dir inode disk
    - 0 data disk addr
    - dir entries w 3 files / sub dirs
  - 1st file inode disk
    - 1 data disk addr
    - this file only takes up 1 file chunk
- data == dir entries / file chunks
  - 1st dir entry
    - 1 parent dir at 0 inode disk addr
    - 1 cur dir at 0 inode disk addr
    - 1 file “y” at 1 inode disk addr (you need to read from here to get to 1st file inode disk)
  - 1st file chunk
    - 1 file chunk filed with 0s

```
mkdir("/f");      -> filesys_create() sub_dir "f"

struct dir *parent_dir = dir_open_path(dir_path)     -> SEARCH FOR PARENT DIR INODE
1. read() root_dir inode_disk
2. traverse() dir_entries data block for matching dir name
3. if found dir_entry{name, inode}, populate() parent_dir w inode

free_map_allocate(1, &inode_sector)           -> BITMAP ALLOCATE dir's inode_disk
1. r() w() bitmap for sub_dir's inode_disk

inode_create(inode_sector, initial_size, is_dir) -> POPULATE, WRITE sub_dir inode_disk INTO DISK
1. w() sub_dir inode_disk into inodes block 3
2. r() w() bitmap for sub_dir's dir_entries
3. w() inodes block 3 to point to data_block 2
4. w() 0s at data block 2 (sub_dir f's dir_entries)

dir_add(dir, dir_name, inode_sector, is_dir)) -> WRITE PARENT DIR + SUB DIR dir_entries
1. w() sub_dir inode_disk addr at parent_dir dir_entries at data block 0
2. w() "." ".." inode_disk addr at sub_dir dir_entry at data block 2

BEFORE:
inode bitmap  11100000
inodes        [d a:0 r:5] [f a:1 r:1] [f a:-1 r:1] [] [] [] [] []
data bitmap   11000000
data          [(.,0) (..,0) (y,1) (z,2)] [u] [] [] [] [] [] []

AFTER:
inode bitmap  11110000        -> free_map_allocate(size, &disk_addr_allocated) to store inode_disk
inodes        [d a:0 r:6] [f a:1 r:1] [f a:-1 r:1] [d a:2 r:2] []
data bitmap   11100000        -> inode_reserve()
data          [(.,0) (..,0) (y,1) (z,2) (f,3)] [u] [(.,3) (..,0) 000000] [] ...
```

DEMO create dir

- inodes == inode disk
  - 1st dir inode disk
    - 0 data disk addr
    - dir entries w 6 files / sub dirs
  - 1st file inode disk
    - 1 data disk addr
    - this file only takes up 1 file chunk
  - 2nd file inode disk
    - -1 data disk addr (file was deleted ??)
    - this file only takes up 1 file chunk
  - 2nd dir inode disk
    - 2 data disk addr
    - dir entries w 2 files / sub dirs
- data == dir entries / file chunks
  - 1st dir entry
    - 1 cur dir at 0 inode disk addr
    - 1 par dir at 0 inode disk addr
    - 1 file “y” at 1 inode disk addr (you need to read from here to get to 1st file inode disk)
    - 1 file “z” at 2 inode disk addr (you need to read from here to get to 2nd file inode disk)
    - 1 dir “f” at 3 inode disk addr (you need to read from here to get to 2nd dir inode disk)
  - 1st file chunk
    - 1 file chunk with “u” in it
  - 2nd dir entry
    - 1 cur dir at 3 inode disk addr
    - 1 parent dir at 0 inode disk addr

```
fd=open("/y", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);

int fd = sys_open("/y")          -> open both file n parent_dir file is in

struct dir *dir = dir_open_path(dir_path)     -> SEARCH FOR DIR INODE
1. read() root_dir inode_disk
2. traverse() dir_entries data block for matching dir name
3. if found dir_entry{name, inode}, populate() parent_dir w inode

IF open(dir)
struct inode *inode = dir_get_inode(dir)
1. return inode

IF open(file)
dir_lookup(dir, file_name, &inode)
1. traverse() dir_entries data block for matching file name
2. if found dir_entry{name, inode}, return inode


int bytes_written = sys_write(fd, "data to write into file", size)

file_write(file_d->file, buffer, size)
1. int fd -> struct file_desc -> struct file -> inode_disk
2. r() w() bitmap for free data block
3. traverse() inode_disk + write() 0s to file_chunks data block (depends on size)
4. traverse() inode_disk + write() "data" to file_chunks data block (depends on size)


BEFORE:
inode bitmap  11000000
inodes        [d a:0 r:3] [f a:1 r:1] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0) (y,1)] [000000000] [] [] [] [] [] []

AFTER:
inode bitmap  11000000
inodes        [d a:0 r:3] [f a:1 r:1] [] [] [] [] [] []
data bitmap   11000000
data          [(.,0) (..,0) (y,1)] [u] [] [] [] [] [] []
```

DEMO open file + write file

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
open(bar)			read			read				
				read						
							read			
					read					
read()					read					
								read		
					write					
read()					read					
									read	
					write					
read()					read					
										read
					write					

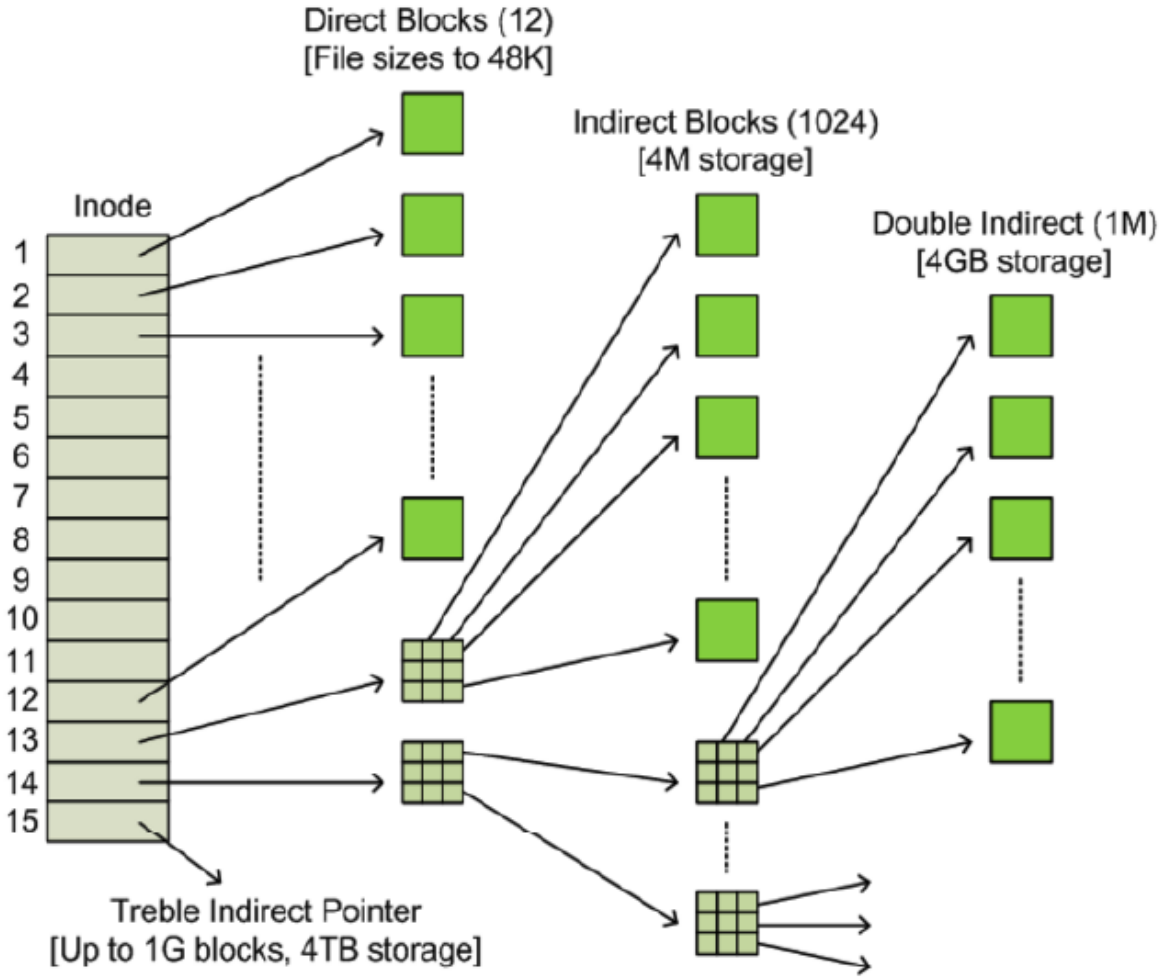
Figure 40.3: File Read Timeline (Time Increasing Downward)

DEMO **open() "/foo/bar"**

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
create (/foo/bar)			read			read				
				read						
							read			
		read								
		write								
							write			
					read					
					write					
				write						
write()					read					
	read									
	write							write		
					write					
write()					read					
	read									
	write								write	
					write					
write()					read					
	read									
	write									write
					write					

Figure 40.4: File Creation Timeline (Time Increasing Downward)

DEMO **create() "/foo/bar"**



DEMO inode (USC project 4 slide)

- **USC example (DIFFERENT FROM PINTOS)**
  - 1 inode_disk = index() 1 file's data
  - inode_disk 1st layer - 12 direct_blocks, 1024 indirect_blocks, 1MB double_indirect_blocks, 1GB treble_indirect_blocks
    - Sector/Block == 4KB (green)
    - Direct == 12 * 4KB == 48KB
    - Indirect == 1024 * 4KB == 4MB
    - Double Indirect == 1MB * 4KB == 4GB
- **Pintos example**
  - 1 inode_disk = index() 1 file's data
    - inode_disk == 512 Bytes == 123*4 + 4 + 4 + 12 bytes
    - indirect_inode_disk == 512 bytes == 128 * 4 bytes
    - file_chunks / dir_entries block == 512 bytes
  - inode_disk 1st layer - 123 direct_blocks, 1*128 indirect_blocks, 1*128*128(16384) double_indirect_blocks
    - Sector/Block == 512 Bytes (green)
    - Direct == 123 * 512B == 63KB
    - Indirect 128 * 512B == 66KB
    - Double Indirect = 16384 * 512B = 8.3MB