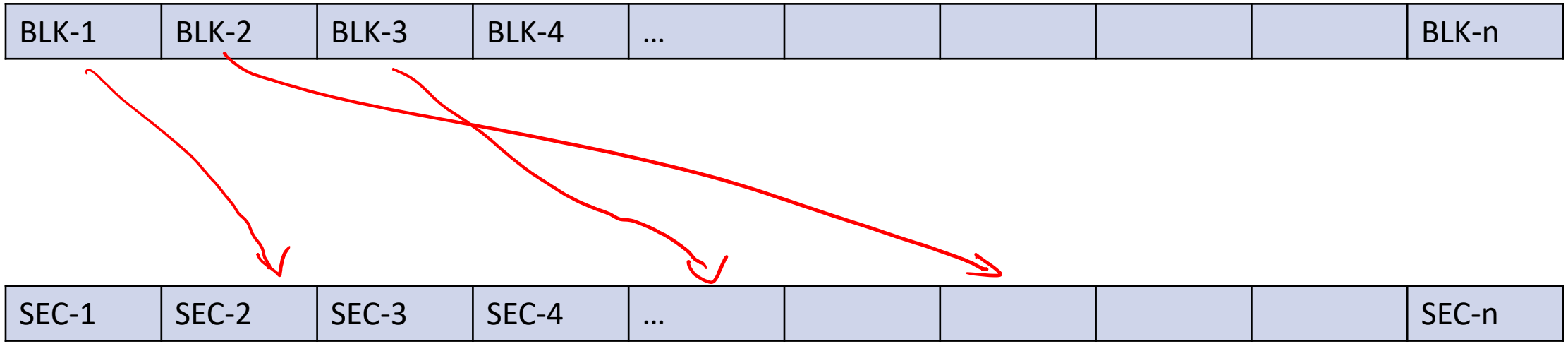
Files



File offset to block

read (offset)

$$\text{blockno} = \frac{\text{offset}}{\text{BLOCK\_SIZE}}$$

Files

- For every file xv6 maintains a table that contains mapping from blocks to sectors
- These mappings are stored into inode
- For every file there is a corresponding inode
- Where inodes are stored?

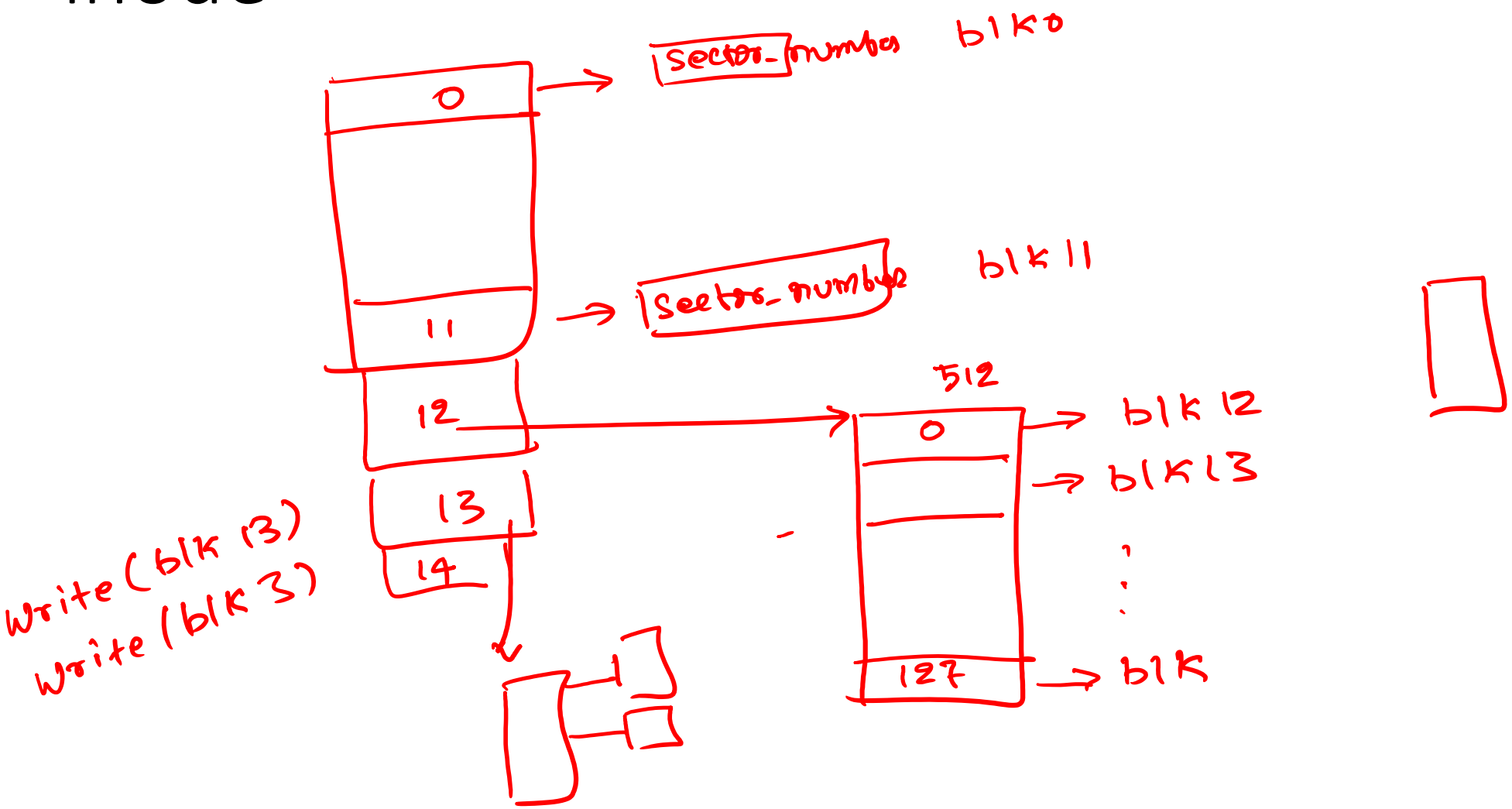
inode

```
struct dinode {
    short type;          // directory or file or device
    short major;         // device specific
    short minor;         // device specific
    short nlink;         // number of links to this file
    uint size;           // size of the file
    unit addrs[12+1];    // block to address table
};
```

inode

- 12 direct addresses
- 1 indirect address
  - an indirect address points to a disk sector that contains 128 addresses

inode


0
11
12
13
14
sector_number
blk0
sector_number
blk11
512
0
127
blk12
blk13
blk
write(blk 13)
write(blk 3)

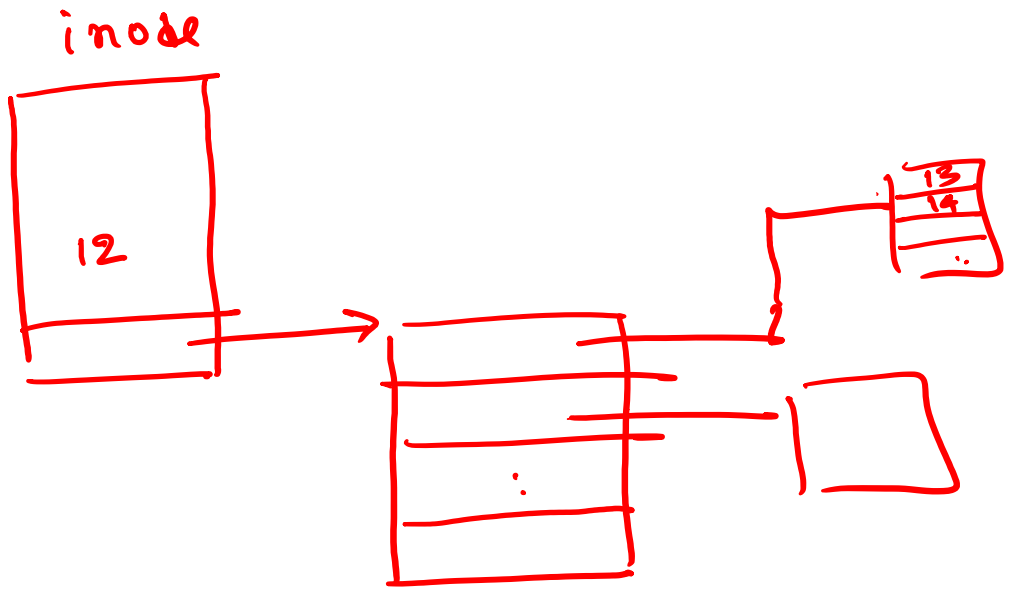
Maximum file size in xv6

$12 \times 512 = 6$ KB

$128 \times 512 = 64$ KB

How to support large files

2 - dimen



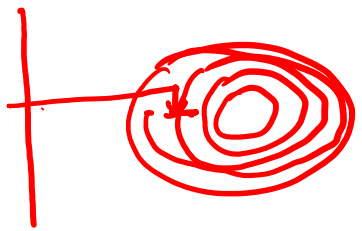
How do we reduce the metadata?

a disk access

seek time (time to move the arm)

+

rotational latency (



Large block size

- sequential access is very fast

Large blocks

- Pros
  - good for sequential access
- Cons
  - fragmentation
- A file system must decide a block size that preserves the spatial locality of common workloads
  - NTFS: 4 KB
  - FAT32: 512 bytes – 4 KB
  - EXT3: 1 KB – 4 KB
  - xv6: 512 bytes

Directory

```
struct dirent {
    ushort inum;
    char name[14];
};
```

Each directory entry is 16 bytes long.

Maximum number of entries in a directory

$$\frac{\text{maximum file size}}{16}$$

Path to inode

- /x/y/z

- How to get inode corresponding to “z”

root_inode

inode = find (root_inode, x)

inode = find (inode, y)

inode = find (inode, z)

inode

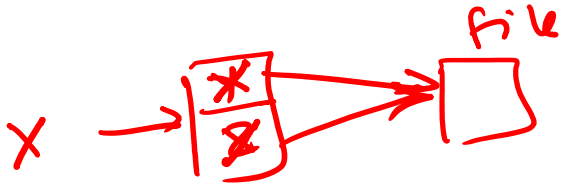
```
struct dinode {
    short type;          // directory or file or device
    short major;         // device specific
    short minor;         // device specific
    short nlink;         // number of links to this file
    uint size;           // size of the file
    unit addrs[12+1];    // block to address table
};
```

link system call

fd = open (“x/y”, O_CREATE);

link (“x/y”, “x/z”); // “y” and “z” points to same file

unlink (“x/y”); // remove “y” from directory “x”



Can we delete the file “y” after unlink?

link system call

```
fd = open (“x/y”, O_CREATE);
link (“x/y”, “x/z”);
unlink (“x/y”);
unlink (“x/z”)
```

Can we delete the file “y” after both unlinks?

struct inode

```
struct inode {        /* in memory copy of the inode */
  uint dev;           // Device specific
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock;  // protects everything below here
  int valid;              // inode has been read from disk?
  struct dinode d;
};
```

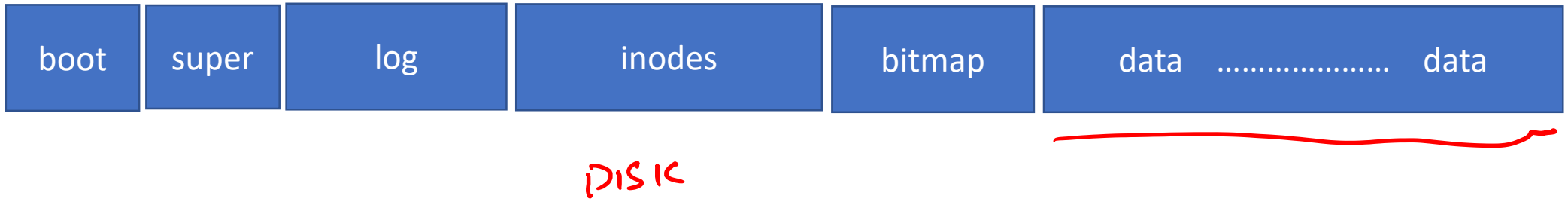
inode

- in memory copy of inode is shared among all the open file descriptors
- the reference count holds the number of open file descriptors

deallocation of inode

- deallocation of inodes are deferred until all the references and links are gone

xv6 File system



Super block

```
struct superblock {
    uint size;          /* total number of blocks */
    uint nblocks;       /* number of data nodes */
    uint ninodes;       /* number of inodes */
    uint nlog;          /* number of log blocks */
    uint logstart;      /* first log block */
    uint inodestart;    /* first inode block */
    uint bmapstart;     /* first free map block */
}
```

inodes

- Limit on maximum number of inodes

bitmap

1 1 1 1 1 1 1 1

$\frac{128}{8}$

512 MB

1 MB bits

512 MB

$\frac{512 \times 2^{20}}{512 \times 8} = 2^{17}$

$\frac{2^{20}}{8}$ byte

$\frac{2^{17}}{2^{29}} \times 100$ 2^{17} bytes

bitmap

- Free bitmap contains information about which blocks are free
  - one bit per block
- balloc allocates a new disk block
  - iterates through the free bitmap to obtain a free block
  - marks the block as allocated in the bitmap
- bfree frees a block
  - find the right bitmap bit corresponding to the sector and clears it

Creating a file

b

a1	100
a2	201
a3	301
a	50

48

64

b/a

b/a

b/c

echo > a

- allocate an inode (update type)
- update nlink, size, direct and indirect blocks fields in the inode
- add a new entry to the parent directory
- update the size of the parent inode (directory)

Creating a file

- create, ialloc, iupdate, dirlink routines in xv6

Synchronization

echo > a

- allocate an inode (update type)
- update nlink, size, direct and indirect blocks fields in the inode
- add a new entry to the parent directory
- update the size of the parent inode (directory)

Synchronization

- xv6 use fine grained locking
  - Processes can write to different files concurrently
- Only one process is allowed to read/write to a file at a given time
- lock the inode before accessing a file/directory

struct inode

```
struct inode {        /* in memory copy of the inode */
  uint dev;           // Device specific
  uint inum;          // Inode number
  int ref;            // Reference count
  struct sleeplock lock;  // protects everything below here
  int valid;          // inode has been read from disk?
  struct dinode d;
};
```

Synchronization

```
read (fd, buf, size);

i = inode (fd)

acquire (&i->lock);
- - -
release (&i->lock);
```