SWE3004 Operating Systems, fall 2023

Project 6. File extension

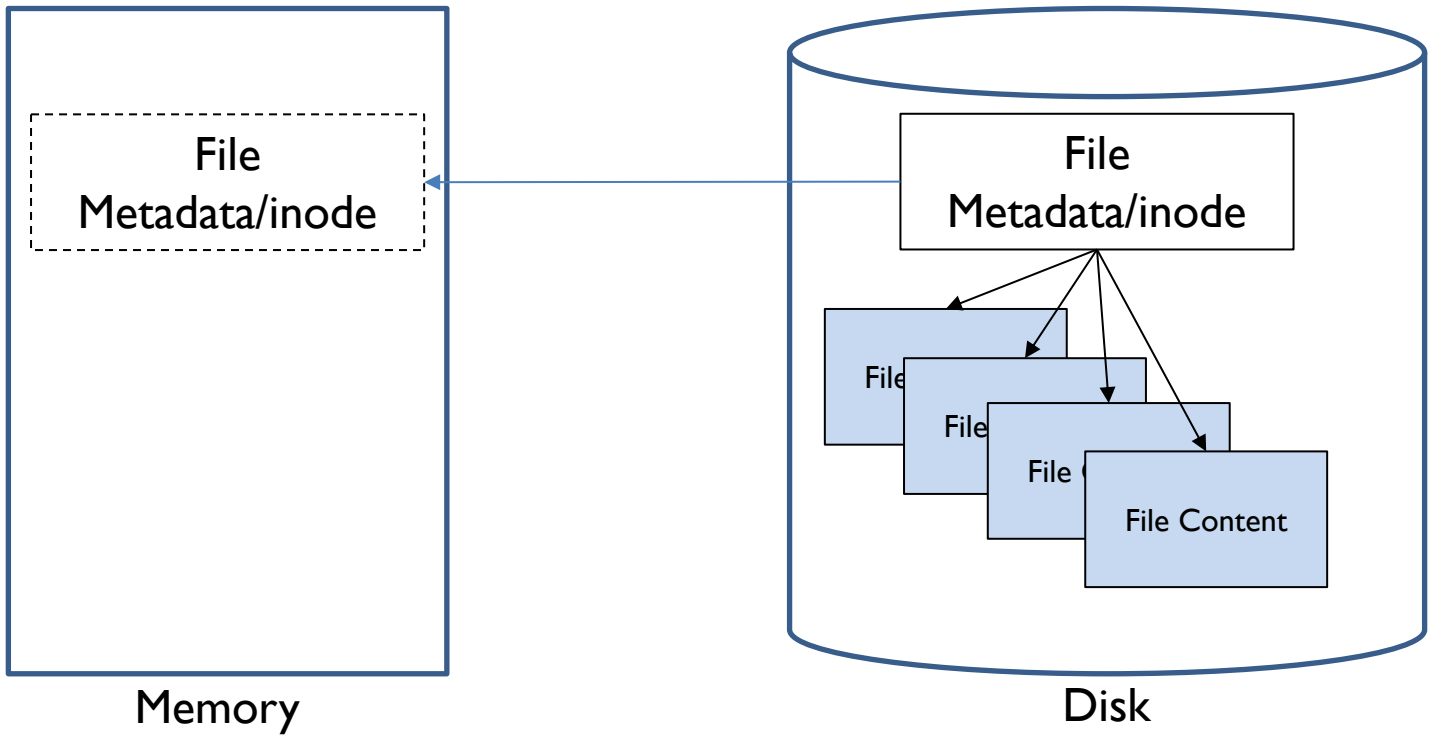
TA)
Jongseok Kim
Shinhyun Park
Hyeonmyeong Lee
Gwanjong Park

Project plan

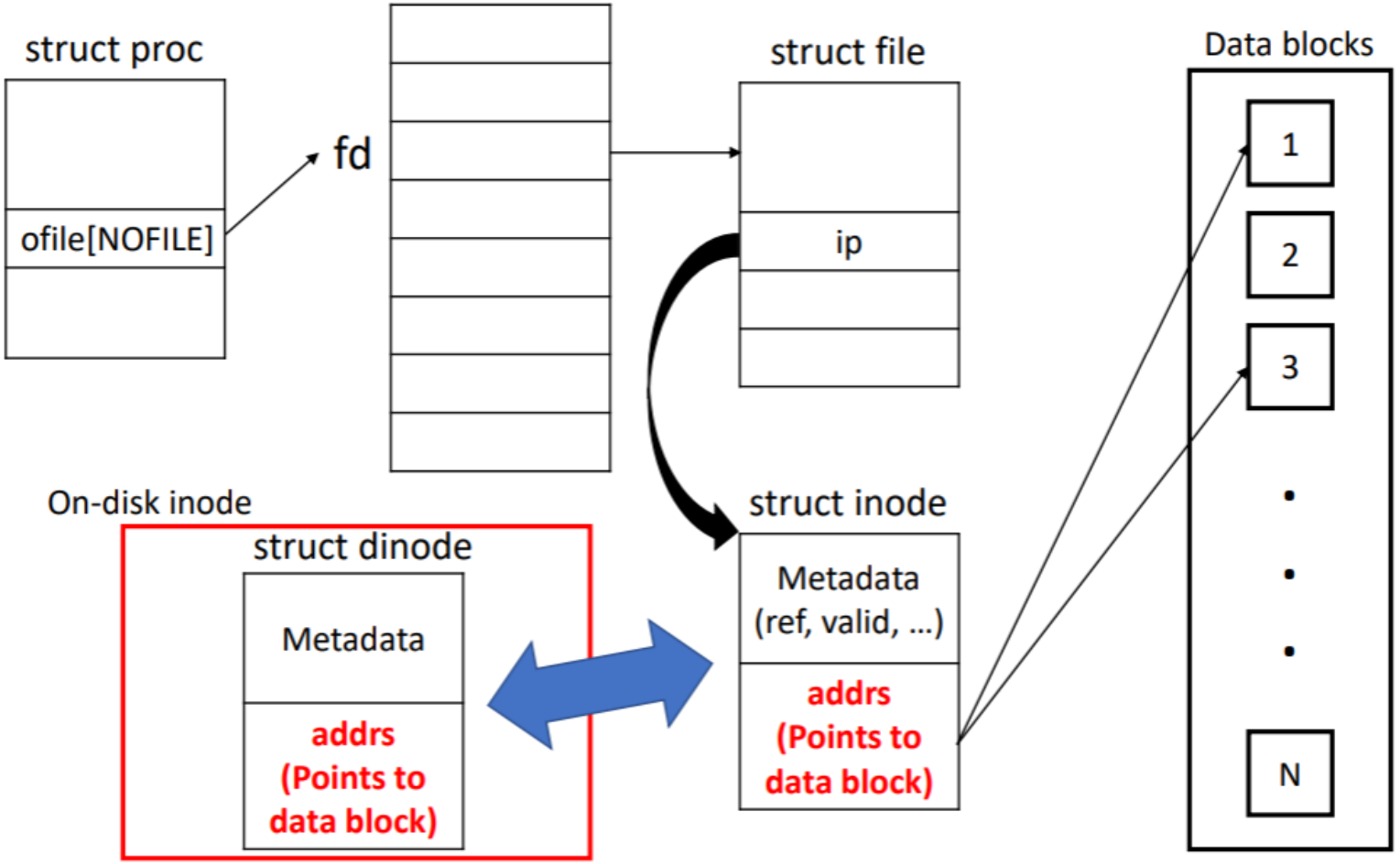
- Total 6 projects
  1) ~~Booting xv6 operating system~~
  2) ~~System call~~
  3) ~~CPU scheduling~~
  4) ~~Virtual memory~~
  5) ~~Page replacement~~
  6) File systems (0pt, optional)

How open file works

- For each file, we have
  - File contents (data)
  - File attributes (metadata or inode)

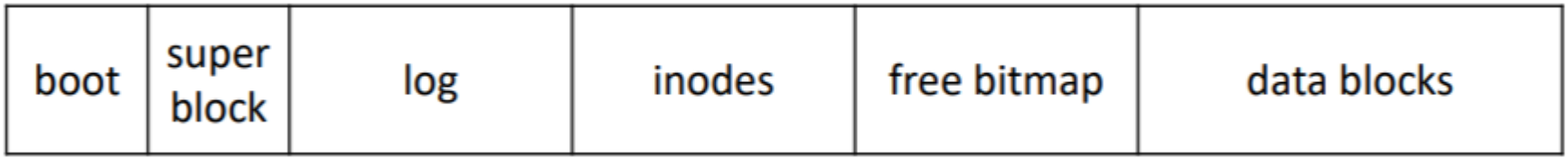


xv6 filesystem



xv6 filesystem: superblock

- Disk layout of xv6 filesystem



- Contents of struct superblock are decided at "mkfs"

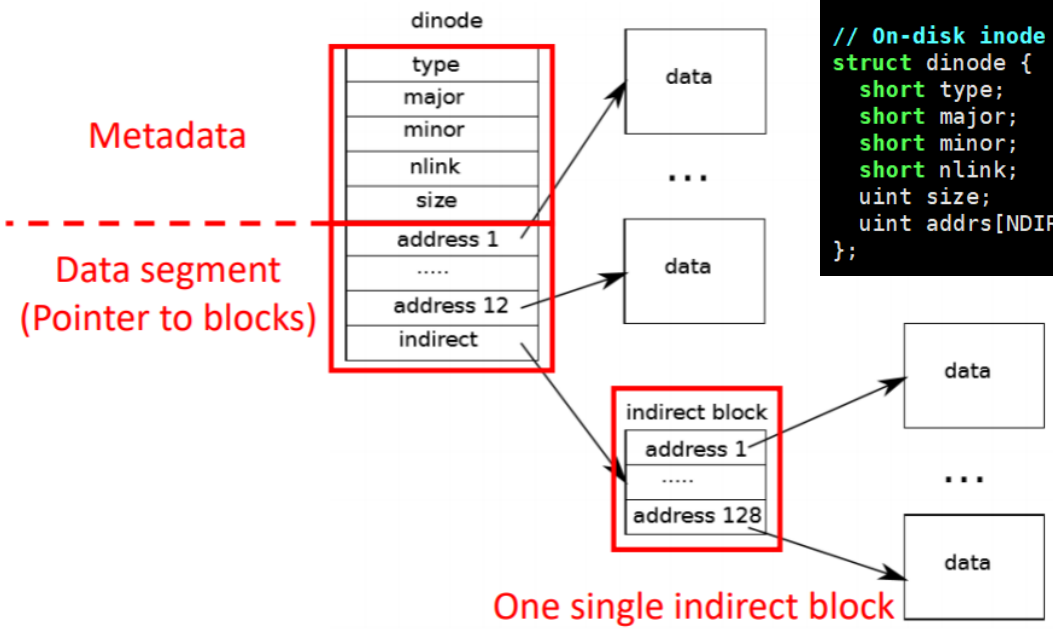
```
struct superblock {
  uint size;         // Size of file system image (blocks)
  uint nblocks;      // Number of data blocks
  uint ninodes;      // Number of inodes.
  uint nlog;         // Number of log blocks
  uint logstart;     // Block number of first log block
  uint inodestart;   // Block number of first inode block
  uint bmapstart;    // Block number of first free map block
};
```

<fs.h>

```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
```

xv6 filesystem: inodes

- On-disk inode is defined by *struct dinode* in fs.h




```
#define NDIRECT 12
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT)

// On-disk inode structure
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint addrs[NDIRECT+1];   // Data block addresses
};
```

<Representation of file on disk of xv6>

xv6 filesystem: bmap() function

- The function bmap() manages the representation
  - bmap() is called both when reading and writing a file
  - When writing, bmap() allocates new blocks as needed to hold file content, as well as allocating an indirect block if needed to hold block addresses

```
static uint
bmap(struct inode *ip, uint bn)
{
  uint addr, *a;
  struct buf *bp;

  if(bn < NDIRECT){
    if((addr = ip->addrs[bn]) == 0)
      ip->addrs[bn] = addr = balloc(ip->dev);
    return addr;
  }
  bn -= NDIRECT;

  if(bn < NINDIRECT){
    // Load indirect block, allocating if necessary.
    if((addr = ip->addrs[NDIRECT]) == 0)
      ip->addrs[NDIRECT] = addr = balloc(ip->dev);
    bp = bread(ip->dev, addr);
    a = (uint*)bp->data;
    if((addr = a[bn]) == 0){
      a[bn] = addr = balloc(ip->dev);
      log_write(bp);
    }
    brelse(bp);
    return addr;
  }

  panic("bmap: out of range");
}
```

NDIRECT blocks are listed in inode itself

On-demand allocation for not allocated blocks

Project 6

- In this project, you have to implement following two features in xv6 file system
  - A. File extension
  - B. Block group
- You should implement both features on one initial xv6 source code

A. File extension in xv6

- Increase the maximum size of an xv6 file
  - Currently xv6 files are limited to 140 sectors (70 kilobytes)
    - 12 "direct" blocks and one "singly-indirect" block
  - Modify the system to support "doubly-indirect" block
    - 11 "direct" blocks, one "singly-indirect" block and one "doubly-indirect block)
  - The result will be that a file will be able to consist of up to 16523 sectors (about 8.5 megabytes)

- Codes to be modified
  - Some header files (fs.h, file.h... as you need)
  - bmap() function (in fs.c)
  - mkfs parameters (in mkfs.c / param.h)

```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
```

```
sb: size 21113 nblocks 21049 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
```

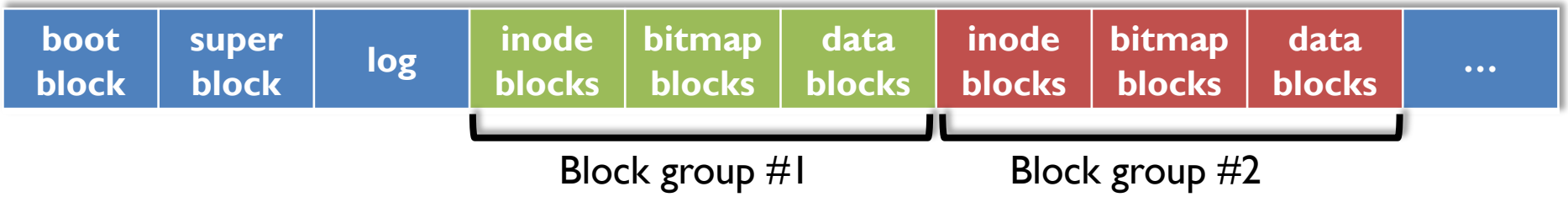
- Use template codes given in Project 0,1,2 (with 1 CPU)
- Test code to check maximum file size will be given

```
wrote 140 sectors → wrote 16523 sectors
```

B. Block Group in xv6

- Block-group is literally a group of blocks that consists of inode blocks, bitmap blocks, data blocks
- With the introduction block group, data and inode blocks are evenly distributed across the physical disk
- Without block group, file system will be look like:

- But with block group, file system will be look like:



B. Block Group in xv6 (cnt'd)

- You should determine a block group size (BGSIZE) in accordance with FSSIZE
  - By default, BGSIZE equals to FSSIZE/32
  - But minimum BGSIZE should be at least 4096 (blocks)
  - If FSSIZE is not divisible by 32, remaining blocks should be added to log area
  - # of inode blocks should be BGSIZE/4
  - If BGSIZE is not divisible by 4, remaining blocks should be added to data blocks
  - You should determine the # of bitmap blocks so that all data blocks can be represented within bitmap blocks
  - Remaining blocks in a block group are all data blocks