

COSI 131a: Programming Assignment 3

Review Session

11/29/2016



What is this assignment about?

- Completing the design of a simple file system on top of a disk.
- Tasks:
 - Complete bitwise operations to fix the free space bitmap
 - Implement single-, double- and triple-indirection to support large files.
- Deadline: **December 5, 11:55pm.**

File System

- Basic storage unit on disk?
- In your simulated disk we have NUM_BLOCKS blocks of BLOCK_SIZE bytes per block.
- A file may occupy multiple blocks depending on how large it is.
- Disk is presented as a very large file in your file system.
- Beginning of the disk contains metadata. After metadata we have the actual data of the files.

File System

- Basic storage unit on disk?

BLOCK

- In your simulated disk we have NUM_BLOCKS blocks of BLOCK_SIZE bytes per block.
- A file may occupy multiple blocks depending on how large it is.
- Disk is presented as a very large file in your file system.
- Beginning of the disk contains metadata. After metadata we have the actual data of the files.

Free Space

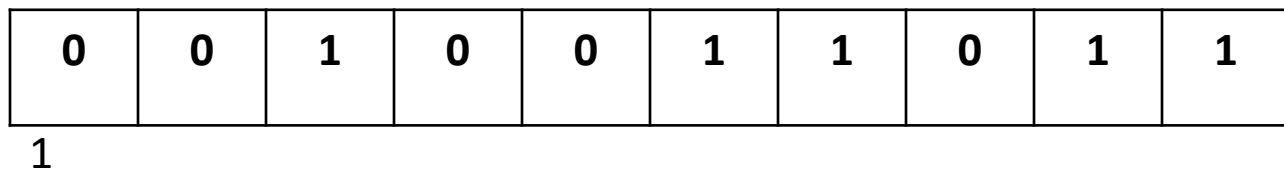
- How to keep track of the free blocks?
 - Use a BITMAP
- Bitmap is a very simple array of bits.
- Each position in the array represents what?

Free Space

- How to keep track of the free blocks?
 - Use a BITMAP
- Bitmap is a very simple array of bits.
- Each position in the array represents what?
 - A block.
 - If position i in array has 1 then block i is currently free.
 - else if block is occupied position i has 0.

Free Space

- How to keep track of the free blocks?
 - Use a BITMAP
- Bitmap is a very simple array of bits.
- Each position in the array represents what?
 - A block.
 - If position i in array has 1 then block i is currently free.
 - else if block is occupied position i has 0.
- If table is full of 0s disk is?
 - Completely full.
- Where do you think bitmap is stored?
 - At the metadata part of the disk.
- How many positions in the free space table?
 - NumBlocks.



Free Space

- How to keep track of the free blocks?
 - Use a BITMAP
- Bitmap is a very simple array of bits.
- Each position in the array represents what?
 - A block.

HOW CAN WE STORE BITS IN AN ARRAY IN JAVA?

- If table
 - C
- Where
 - A
- How many positions in the free space table?
 - NumBlocks.

0	0	1	0	0	1	1	0	1	1
---	---	---	---	---	---	---	---	---	---

1

msize

Fix methods in Bitwise.java

- FreeMap is already implemented as an array of bytes.
- A byte is a primitive type in Java. A byte is made up of 8 individual bits. For example this is a byte: `byte b = (byte)0x00;` this is: "00000000"
- FreeMap depends on methods in Bitwise.java that you have to implement.
- These methods ask you to manipulate individual bits since each bit represents a block.
- Most methods ask to set a bit or clear a bit. Setting a bit means set it to 1 , clearing a bit means clear it to 0.

Example

```
private static final int bitmasks[] = {1, 2, 4, 8, 16, 32, 64, 128};

/**
 * Check to see if bit i is set in byte. Returns true if it is
 * set, false otherwise.
 */
public static boolean isset(int i, byte b) {
    //FIXME
}

/**
 * Check to see if bit i is set in array of bytes. Returns true if
 * it is set, false otherwise.
 */
public static boolean isset(int i, byte bytes[]) {
    //FIXME
}
```

Bitwise Operations in Java

- `&` :bitwise AND operation.
- `|` :bitwise inclusive OR operation.
- `^` :bitwise exclusive OR operation (XOR).
- `~` :inverts a bit pattern.

Logical Bitwise Operations

bit 1	bit 2	OR (<code> </code>)	AND (<code>&</code>)	XOR (<code>^</code>)
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

Example

- Let's try to set the seventh bit of byte b = (byte)0x00 to be 1.
- `\\b = "00000000"`
- b do_something = 01000000

Example

- Let's try to set the seventh bit of byte `b = (byte)0x00` to be 1.
- `\\b = "00000000"`
- `b do_something = 01000000`
- In Bitwise.java we have this: `int[] bitmasks = {1, 2, 4, 8, 16, 32, 64, 128};`
- You should think of a way to use it.
- THINK: What is the binary representation of 64?
- 64 in binary is:
 $(0*2^7)+(1*2^6)+(0*2^5)+(0*2^4)+(0*2^3)...$
which is: 01000000
- `b do_something = 01000000 ????`

Example

- Let's try to set the second bit of byte `b = (byte)0x00` to be 1.
- `\\b = "00000000"`
- `b do something = 01000000`
- In `Bitwise.java` we have this: `int[] bitmasks = {1, 2, 4, 8, 16, 32, 64, 128};`
- You should think of a way to use it.
- THINK: What is the binary representation of 64?
- 64 in binary is:
 $(0 \cdot 2^7) + (1 \cdot 2^6) + (0 \cdot 2^5) + (0 \cdot 2^4) + (0 \cdot 2^3) \dots$
which is: 01000000
- `b do something = 01000000 ????`
- `b | bitmasks[6] = 01000000`
- Because:
$$\begin{array}{r} 00000000 \\ | 01000000 \\ \hline = 01000000 \end{array}$$
- Use this kind of logic to implement the functions in `Bitwise.java`

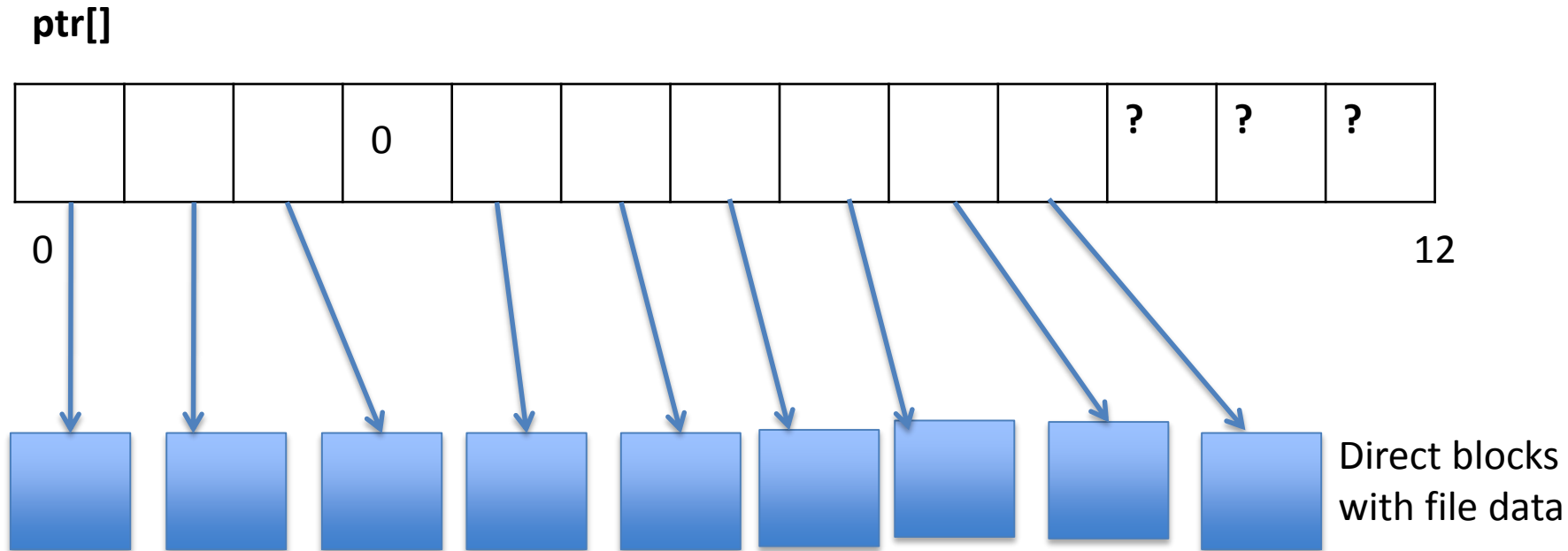
Testing

- When you are done implementing the 7 methods in Bitwise.java test them with TestBitwise.java

inodes

- Each file in the system is described by an index node -> inode.
- Inode has info about the file: owner, type of file, size etc.
- The one we really care about:
`int[] ptr = new int[13];`
- Pointer fields point directly/indirectly to the data blocks the file occupies.
 - A direct block contains file data.
 - An indirect block contains pointers that point to direct blocks or other indirect blocks.
 - They are both disk blocks so they have the same size.

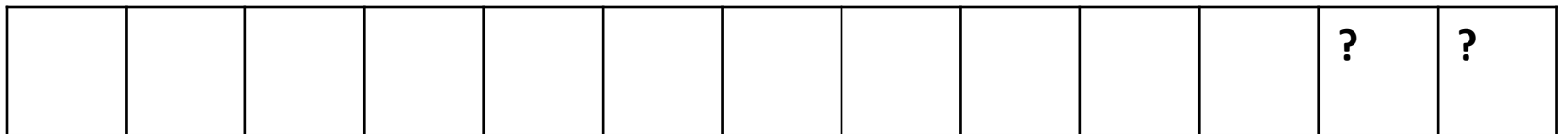
Direct Blocks



- The first 10 pointers in `ptr[]` point to the first 10 **direct** blocks of the file.
- `Ptr[3] = 0` is a null pointer that indicates a **hole** in the file.

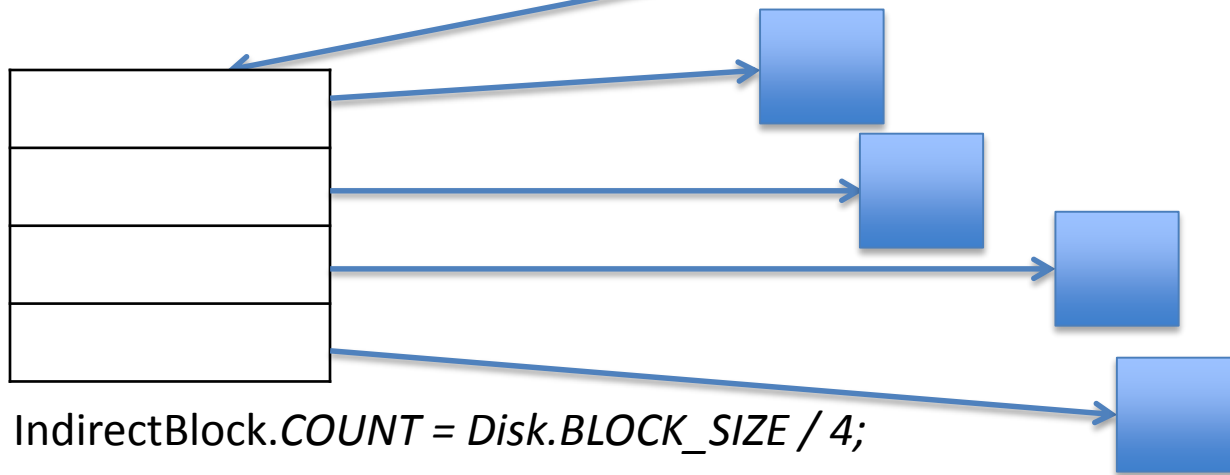
Indirect Blocks

ptr[]



0

12



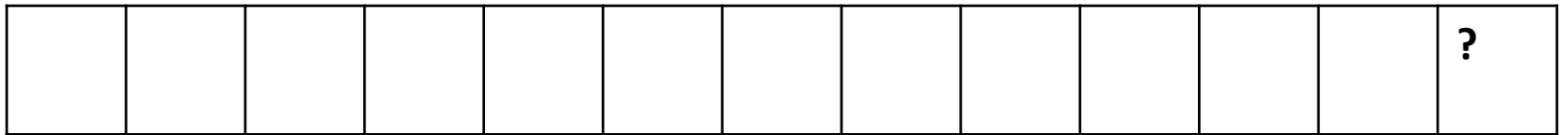
SINGLY INDIRECT POINTER

IndirectBlock.COUNT = Disk.BLOCK_SIZE / 4;

- The 11th pointer at position ptr[10] points to an indirect block.
- It points to a block of pointers that they point directly to the data blocks of the file.

Indirect Blocks

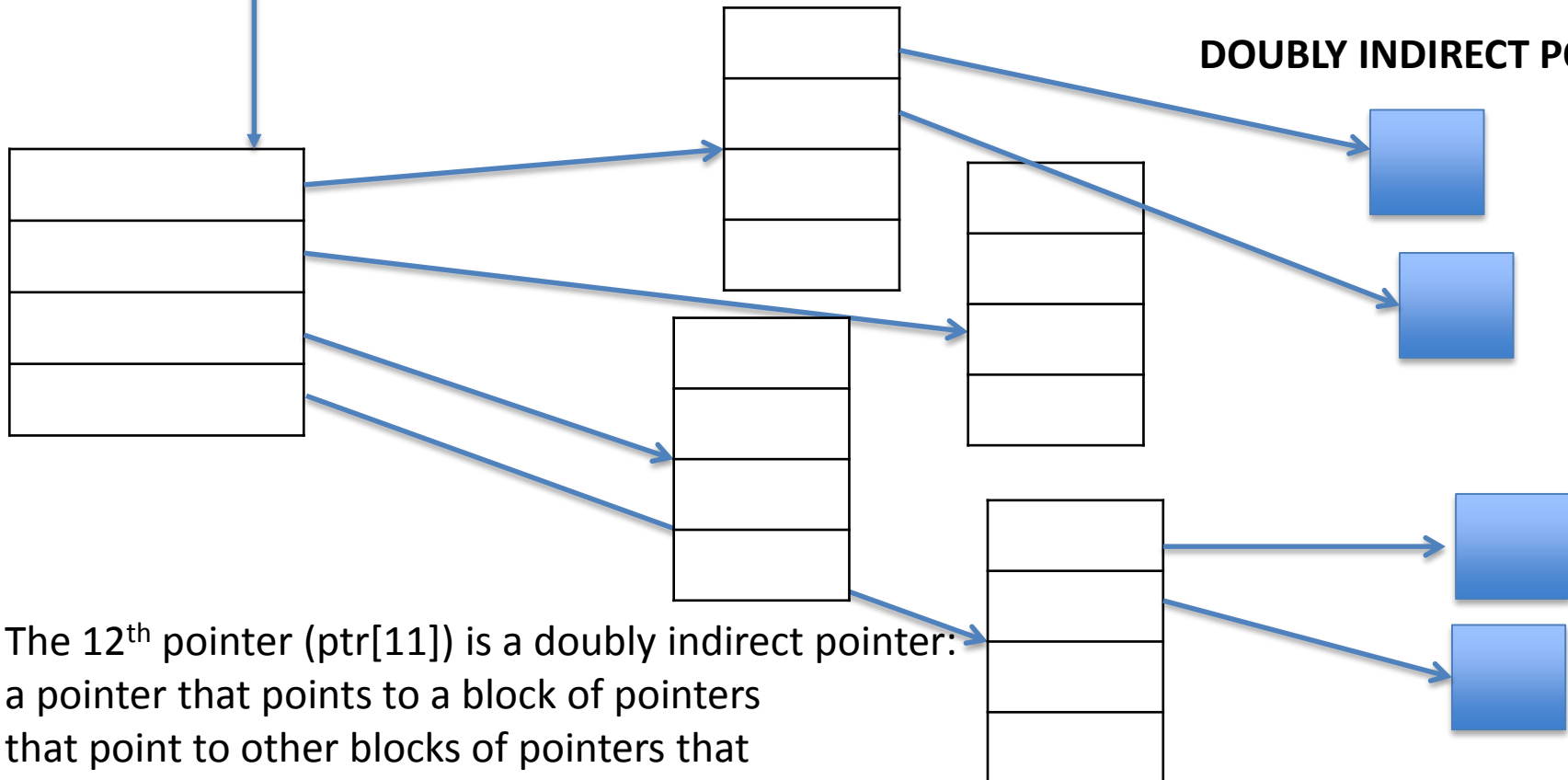
ptr[]



0

12

DOUBLY INDIRECT POINTER



The 12th pointer (ptr[11]) is a doubly indirect pointer:
a pointer that points to a block of pointers
that point to other blocks of pointers that
then point to blocks of the file's data)

Indirect Blocks

- You can imagine what `ptr[12]` holds...
- A triply indirect pointer: a pointer that points to a block of pointers that point to other blocks of pointers that point to other blocks of pointers that then point to data blocks containing the file's data.

Indirect Blocks

- You can imagine what `ptr[12]` holds...
- A triply indirect pointer: a pointer that points

What is the maximum file size we can have in blocks?
Assume `indirect_block.count` is 4.

containing the file's data.

Indirect Blocks

- You can imagine what ptr[12] holds...
- A triply indirect pointer: a pointer that points

What is the maximum file size we can have in blocks?
Assume indirect_block.count is 4.

containing the file's data.

ANSWER:

$$10 + 4 + 4*4 + 4*4*4 =$$

94 BLOCKS!

Your task...

- Implement single-, double-, triple- indirection to support large files.
- Just one function in MyFileSystem.java

```
private DirectBlock getDirectBlock(int fd, MODE mode) { //FIXME!!!
    Inode inode = fileTable.getInode(fd);
    int seekPtr = fileTable.getSeekPointer(fd);
    int blockNum = seekPtr / Disk.BLOCK_SIZE;
    int blockOff = seekPtr % Disk.BLOCK_SIZE;

    if(blockNum > 9) {
        System.err.println("Large files unsupported");
        System.exit(1);
    }
    boolean fresh = inode.ptr[blockNum] == 0;

    // The blockNum is a logical block number referring to a
    // pointer in the inode.

    if(fresh)
        if(mode == MODE.r)
            return DirectBlock.hole;
        else if((inode.ptr[blockNum] = freeMap.find()) == 0)
            return null;
    return new DirectBlock(disk, inode.ptr[blockNum], blockOff, fresh);
}
```

Your task...

- Implement single-, double-, triple- indirection to support large files.
- Just one function in MyFileSystem.java

```
private DirectBlock getDirectBlock(int fd, MODE mode) { //FIXME!!!  
    Inode inode = fileTable.getInode(fd);  
    int seekPtr = fileTable.getSeekPointer(fd);  
    int blockNum = seekPtr / Disk.BLOCK_SIZE;  
    int blockOff = seekPtr % Disk.BLOCK_SIZE;
```

file descriptor of an open file

MODE.r for read
MODE.w for write

```
    }  
    boolean fresh = inode.ptr[blockNum] == 0;  
  
    // The blockNum is a logical block number referring to a  
    // pointer in the inode.  
  
    if(fresh)  
        if(mode == MODE.r)  
            return DirectBlock.hole;  
        else if((inode.ptr[blockNum] = freeMap.find()) == 0)  
            return null;  
    return new DirectBlock(disk, inode.ptr[blockNum], blockOff, fresh);
```

```
}
```


Your task...

- Implement single-, double-, triple- indirection to support large files.
- Just one function in MyFileSystem.java

```
private DirectBlock getDirectBlock(int fd, MODE mode) { //FIXME!!!  
    Inode inode = fileTable.getInode(fd);  
    int seekPtr = fileTable.getSeekPointer(fd);  
    int blockNum = seekPtr / Disk.BLOCK_SIZE;  
    int blockOff = seekPtr % Disk.BLOCK_SIZE;
```

```
    if(blockNum > 9) {  
        System.err.println("Large files unsupported");  
        System.exit(1);  
    }
```

```
    boolean fresh = inode.ptr[blockNum] == 0;
```

```
    // The blockNum is a logical block number referring to a
```

return a DirectBlock object representing the direct block given the current seek position in the open file identified by fd.

```
        return DirectBlock.hole;  
    else if((inode.ptr[blockNum] = freeMap.find()) == 0)  
        return null;  
    return new DirectBlock(disk, inode.ptr[blockNum], blockOff, fresh);
```

```
}
```

Your task...

- Implement single-, double-, triple- indirection to support large files.
- Just one function in MyFileSystem.java

```
private DirectBlock getDirectBlock(int fd, MODE mode) { //FIXME!!!
    Inode inode = fileTable.getInode(fd);
    int seekPtr = fileTable.getSeekPointer(fd);
    int blockNum = seekPtr / Disk.BLOCK_SIZE;
    int blockOff = seekPtr % Disk.BLOCK_SIZE;

    if(blockNum > 9) {
        System.err.println("Large files unsupported");
        System.exit(1);
    }
    boolean fresh = inode.ptr[blockNum] == 0;

    if(fresh)
        if(mode == MODE.r)
            return DirectBlock.hole;
        else if((inode.ptr[blockNum] = freeMap.find()) == 0)
            return null;
    return new DirectBlock(disk, inode.ptr[blockNum], blockOff, fresh);
}
```

points somewhere inside the open file.

logical block number referring to a data block.

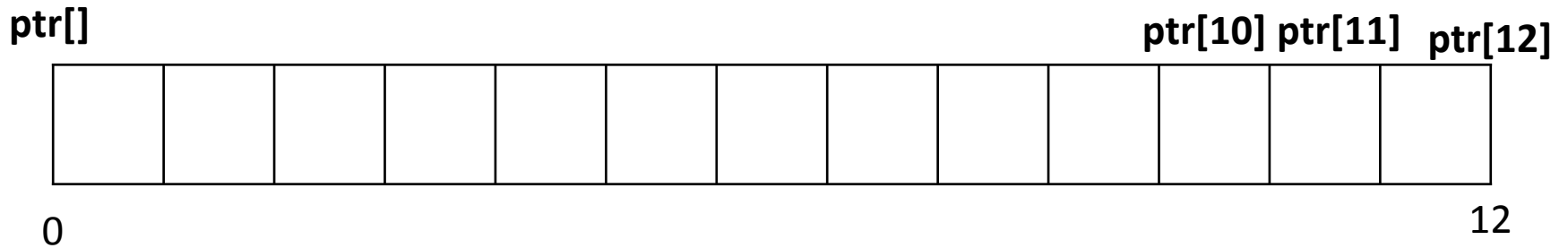
offset within that block that the seek pointer is at.

Example with Indirection

- Let's assume you want to read a direct block. Here is some info:
 - SeekPointer = 192
 - Disk.Block_Size = 16
 - IndirectBlock.COUNT = Disk.Block_size/4 = 4
- BlockNum ?

Example with Indirection

- Let's assume you want to read a direct block. Here is some info:
 - SeekPointer = 192
 - Disk.Block_Size = 16
 - IndirectBlock.COUNT = Disk.Block_size/4 = 4
- BlockNum ?
- = $192/16 = 12$
- BlockOff ?
- = $192\%16 = 0$
- BlockNum is 12 ($12 > 10$) so we know there is indirection involved. What level though?

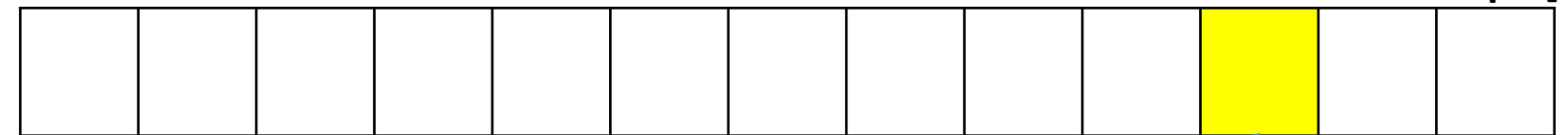


Example with Indirection

- Let's assume you want to read a direct block. Here is some info:
 - SeekPointer = 192
 - Disk.Block_Size = 16
 - IndirectBlock.COUNT = Disk.Block_size/4 = 4
- BlockNum ?
- = $192/16 = 12$
- BlockOff ?
- = $192\%16 = 0$
- BlockNum is 12 ($12 > 10$) so we know there is indirection involved. What level though?
- Single Indirection level.
- To answer why, we should think what is the max BlockNum we can have in the single indirection level?
- 14

ptr[]

ptr[10] ptr[11] ptr[12]



0 12

In read mode:

//check if there is a hole

//if yes return DirectBlock.hole

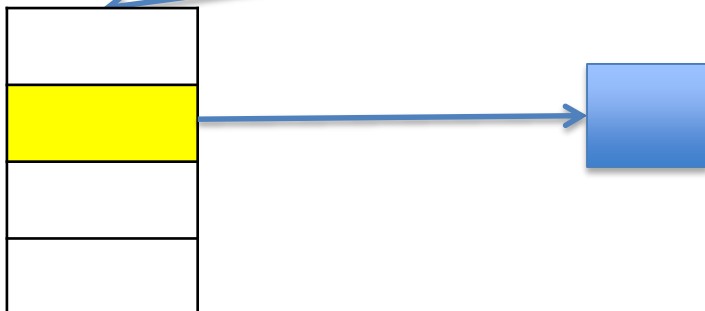
//else return DirectBlock

In write mode:

//check if there is space to write (using freeMap)

//if yes return DirectBlock

//else return null



Testing

- When you are done implementing indirection in `MyFileSystem.java` test it with `TestMyFileSystem.java`.
- Different tests for each indirection level so you can individually test your implementation for all indirection levels.

Questions?