# FAT, I-nodes

## Computer Operating Systems, Summer 2025

**Instructors:** Joel Ramirez Travis McGaha

**TAs:** Ash Sid Maya

# Lecture Outline

- **Inodes**
- Directories
- Block Caching

Poll Everywhere

pollev.com/cis5480

- What was the big downside of using FAT?

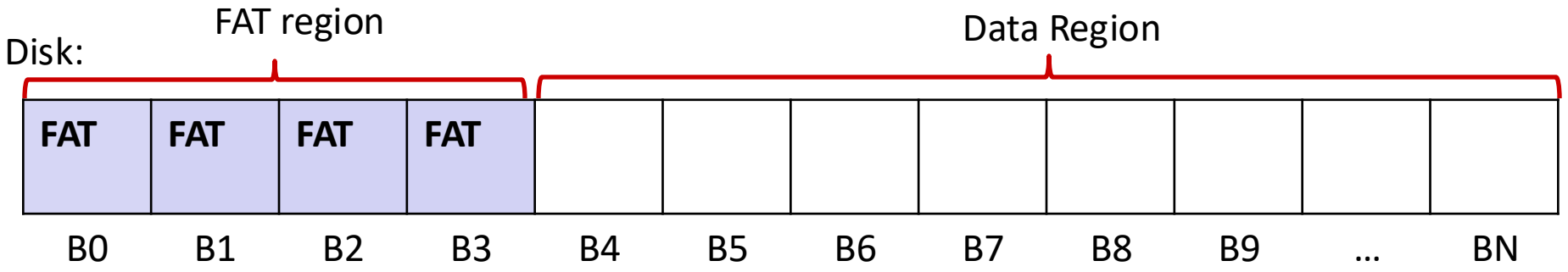
Poll Everywhere

pollev.com/cis5480

- Instead, could we store ***most*** FAT blocks ***on disk*** and only load into memory the FAT blocks that are used for looking up files that are currently open used (aka have entries in the file table, etc)?

# Explanation

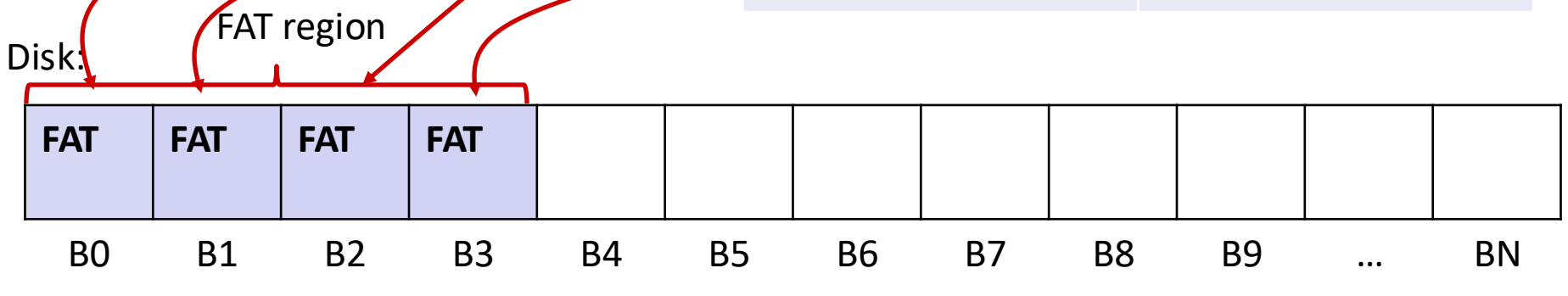
- Blocks of a file could be spread out across disk. We may have to load all FAT blocks to lookup a file anyways
- Small example:
  - Consider block size 256,
  - FAT entry 2 bytes, so 128 entries per FAT block
  - FAT takes up 4 blocks
- **Reminder: FAT region is separate from the data region (blocks it manages)**



# Explanation

- Blocks of a file could be spread out across disk. We may have to load all FAT blocks to lookup a file anyways
- Small example:
  - Consider block size 256,
  - FAT entry 2 bytes, so 128 entries per FAT block
  - FAT takes up 4 blocks

| Block # | Next |
|---|---|
| … | |
| 2 | 128 |
| … | |
| 128 | 256 |
| … | |
| 256 | 500 |
| … | |
| 500 | |



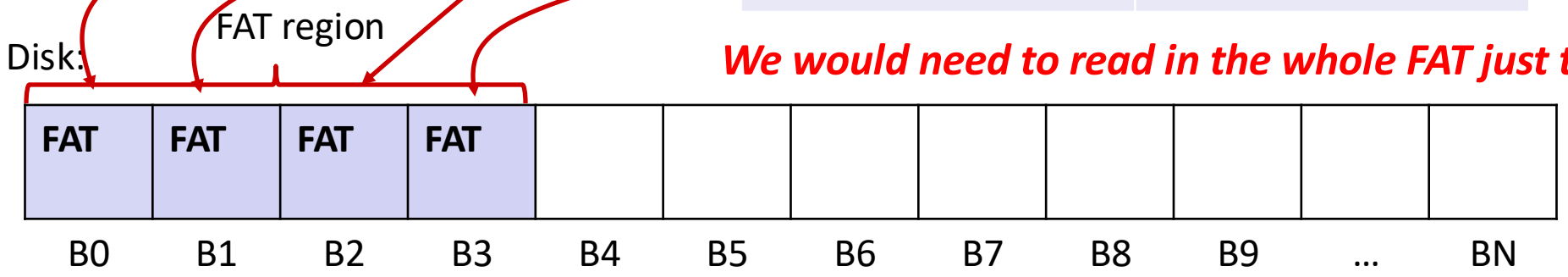

**Consider we have a file that starts at the second block of the Data Region**

# Explanation

- Blocks of a file could be spread out across disk. We may have to load all FAT blocks to lookup a file anyways
- Small example:
  - Consider block size 256,
  - FAT entry 2 bytes, so 128 entries per FAT block
  - FAT takes up 4 blocks

| Block # | Next |
|---|---|
| … | |
| 2 | 128 |
| … | |
| 128 | 256 |
| … | |
| 256 | 500 |
| … | |
| 500 | |

**Consider we have a file that starts at the second block of the Data Region**

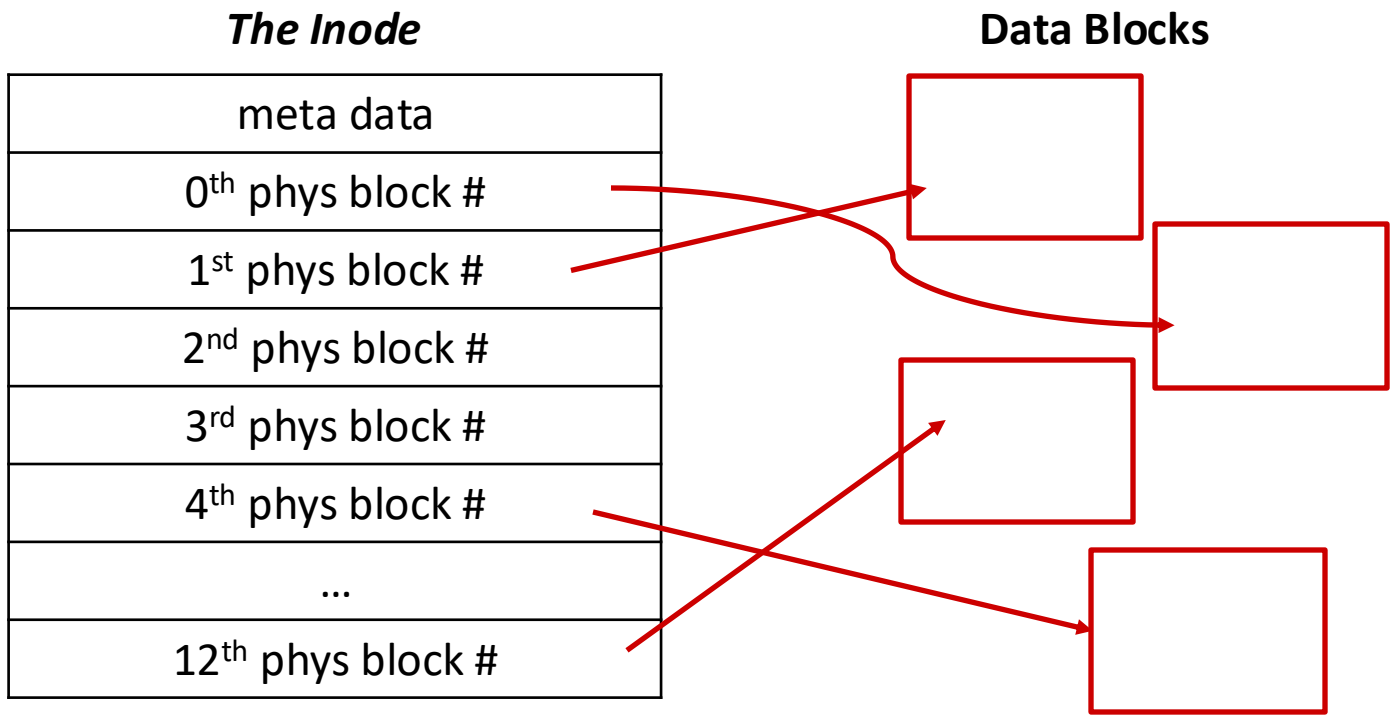



***We would need to read in the whole FAT just to look up this file***

# Inode motivation

- Idea: we usually don't care about ALL blocks in the file system, just the blocks for the currently open files
- Instead of spreading out the block numbers in a table, can we group the block numbers of a file together?

- ***Yes: we call these inodes***:
  - Contains some metadata about the file and 12 physical block numbers corresponding to the first 12 logical blocks of a file




*not all data blocks shown.

# Inode layout

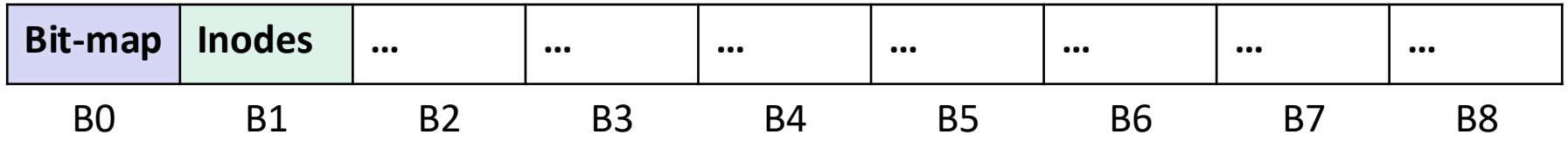
- Inodes contain:
  - some metadata about the file
    - Owner of the file
    - Access permissions
    - **Size of the file**
    - Time of
      - last change of file, last access to file, last change to INODE of file.
  - 12 physical block numbers corresponding to the first 12 logical blocks of a file

```
typedef block_no_t int

struct inode_st {
  attributes_t metadata;
  block_no_t blocks[12];
  // more fields to be shown
  // on later slides
};
```

# Inodes Disk Layout

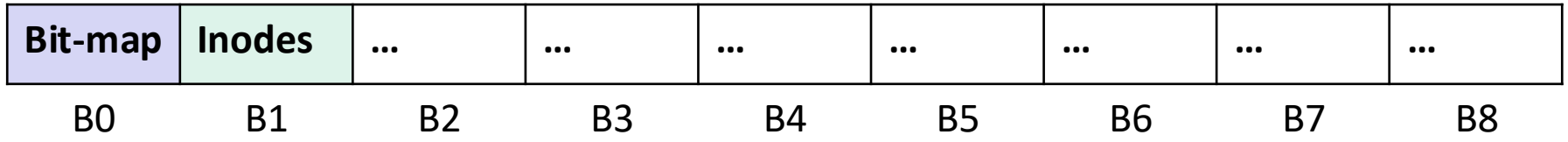
- When we use Inodes instead of FAT, we get something like this instead:



Poll Everywhere

pollev.com/cis5480

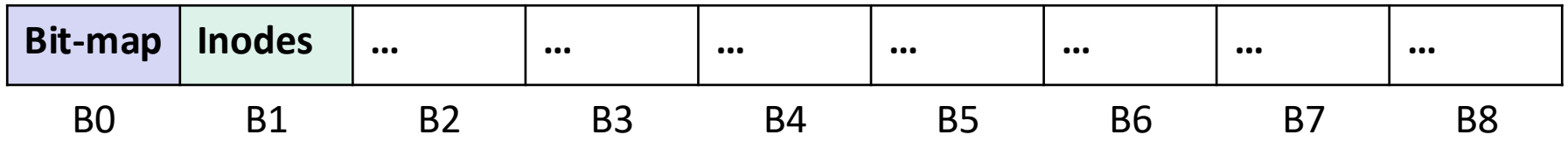
- When we use Inodes instead of FAT, we get something like this instead:



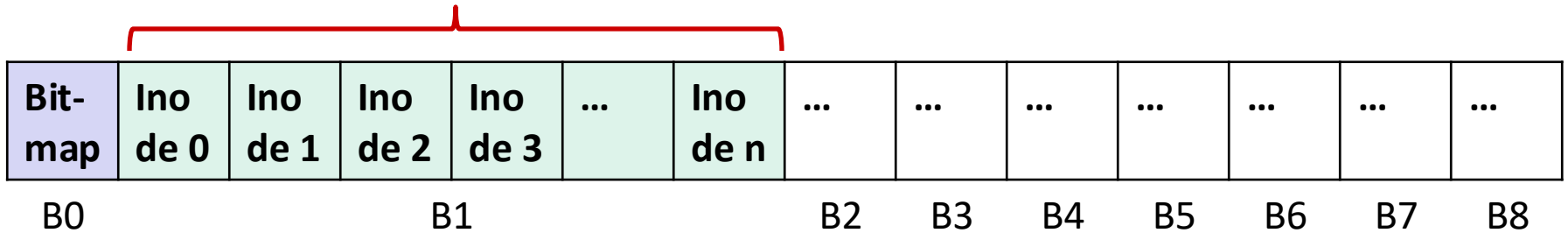

**Wait, why do we need a Bit-Map for this filesystem implementation?**
**How many blocks could we track if a block size is 512 bytes?**

# Inodes Disk Layout

- When we use Inodes instead of FAT, we get something like this instead:

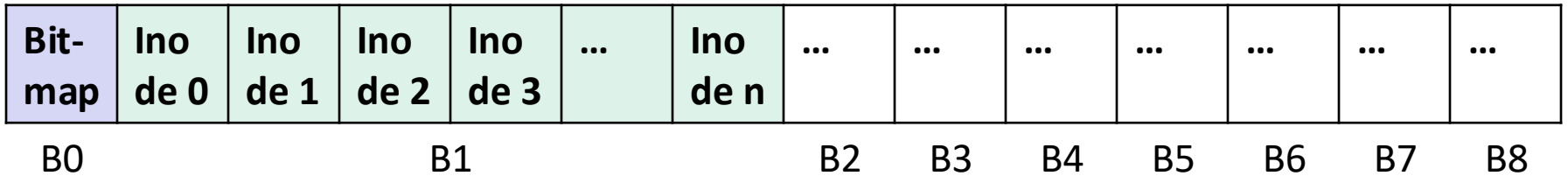



- Inodes are smaller than a block, can fit multiple inodes in a single block
- Each Inode is numbered



# Example File Block Lookup

- Each File will have an Inode number
- Suppose that we wanted to look up a file that is made of 4 blocks.
  - First, we need the Inode number for the file (lets assume it is 2)



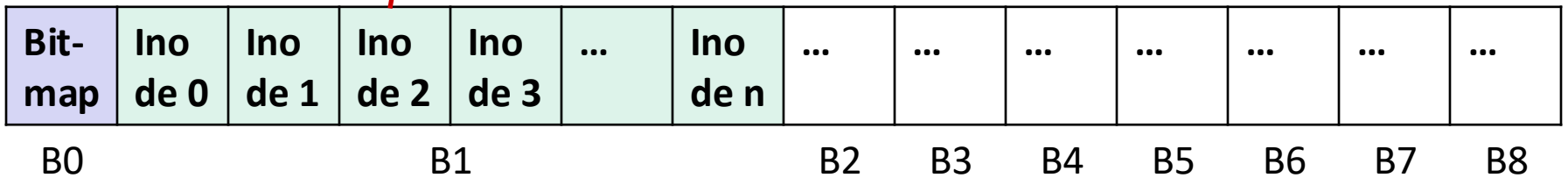
# Example File Block Lookup

- Each File will have an Inode number
- Suppose that we wanted to look up a file that is made of 4 blocks.
  - First, we need the Inode number for the file (lets assume it is 2)
  - We can read the Inode to see which blocks makeup the file

| meta data | … |
|---|---|
| 0th phys block # | 0 |
| 1st phys block # | 5 |
| 2nd phys block # | 3 |
| 3rd phys block # | 2 |
| … | |

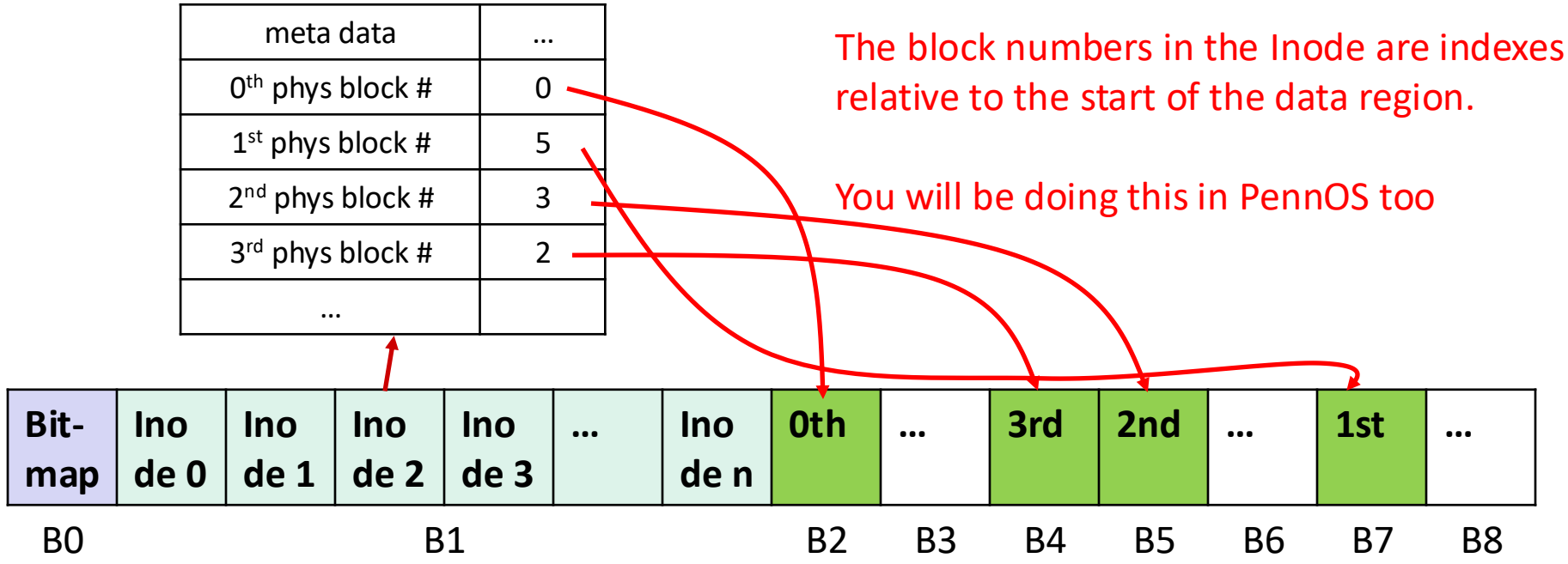
The block numbers in the Inode are indexes relative to the start of the data region.

You will be doing this in PennOS too



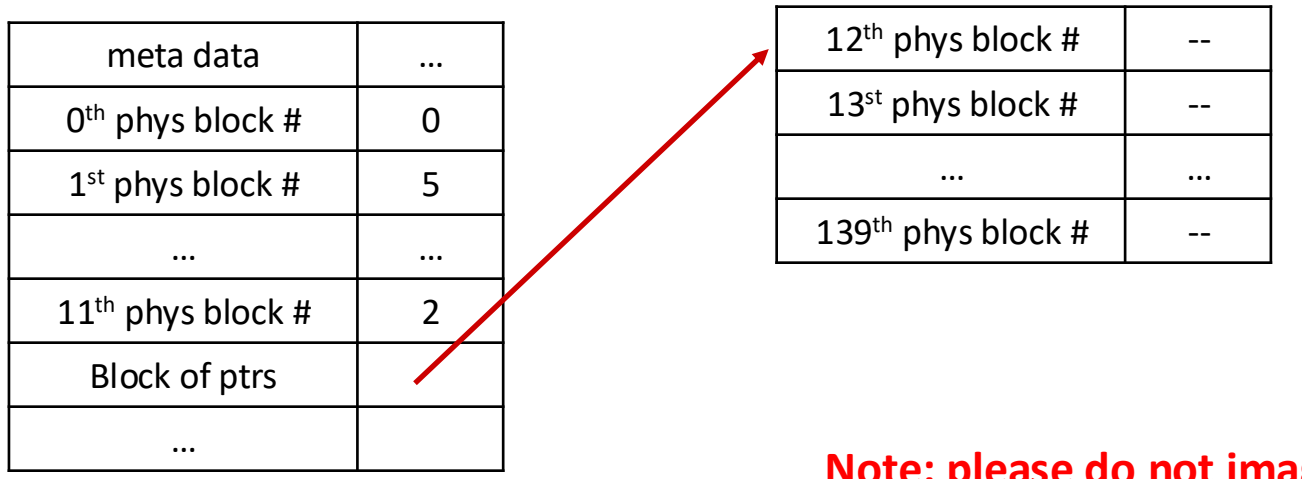
# Example File Block Lookup

- Each File will have an Inode number
- Suppose that we wanted to look up a file that is made of 4 blocks.
  - First, we need the Inode number for the file (lets assume it is 2)
  - We can read the Inode to see which blocks makeup the file



# File Sizes with Inode

- So with Inodes, how many blocks can we have per file?
  - So far: 12 blocks per file (this is not enough, way too small!)
    - About 6,000 bytes.
    - An average MP4 song would at least 3,000,000 bytes.
- We can allocate a ***block*** to hold more block numbers
  - This block can hold 128 block numbers




This is a singly indirect pointer;
it points to a block of pointers (or block numbers)

**Note: please do not imagine these structures like tables.**
**They are not.**
**They are purely arrays of integers.**

# File Sizes with Inode

- So with Inodes, how many blocks can we have per file?
  - So far: 12 blocks per file (this is not enough, way too small!)
    - About 6,000 bytes.
    - An average MP4 song would at least 3,000,000 bytes
- We can allocate a ***block*** to hold more block numbers

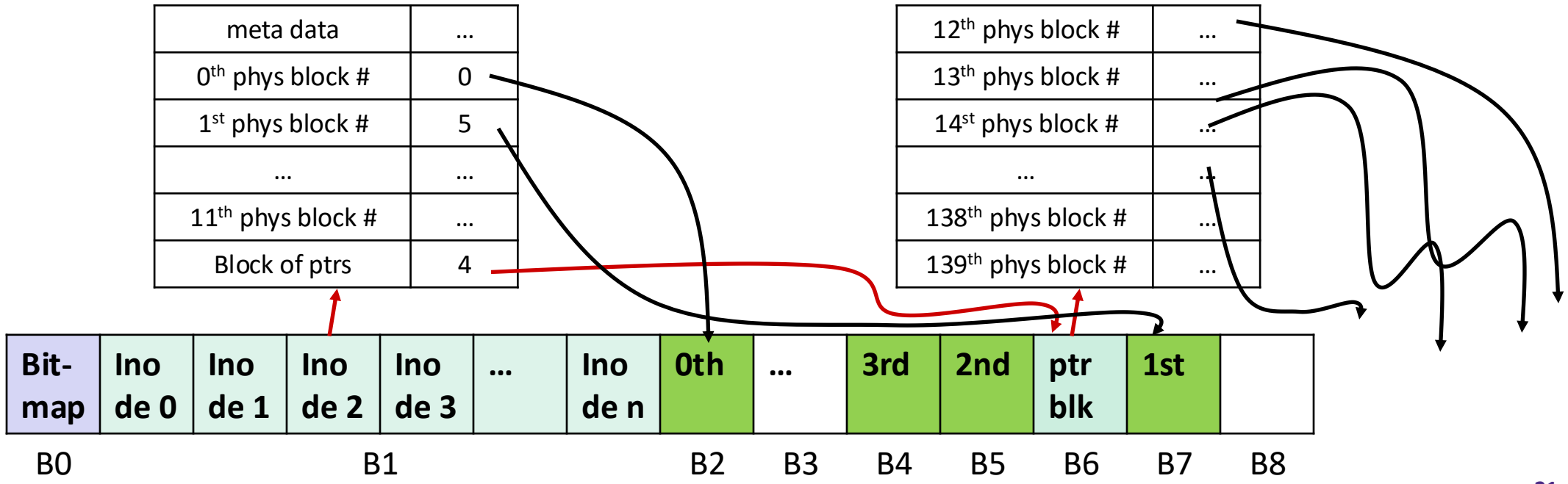
```
struct inode_st {
  attributes_t metadata;
  block_no_t blocks[12];
  block_no_t more_pointers;
  // more fields to be shown
  // on later slides
};
```

# File Sizes with Inode

- So with Inodes, how many blocks can we have per file?
  - So far: 12 blocks per file (this is not enough, way too small!)

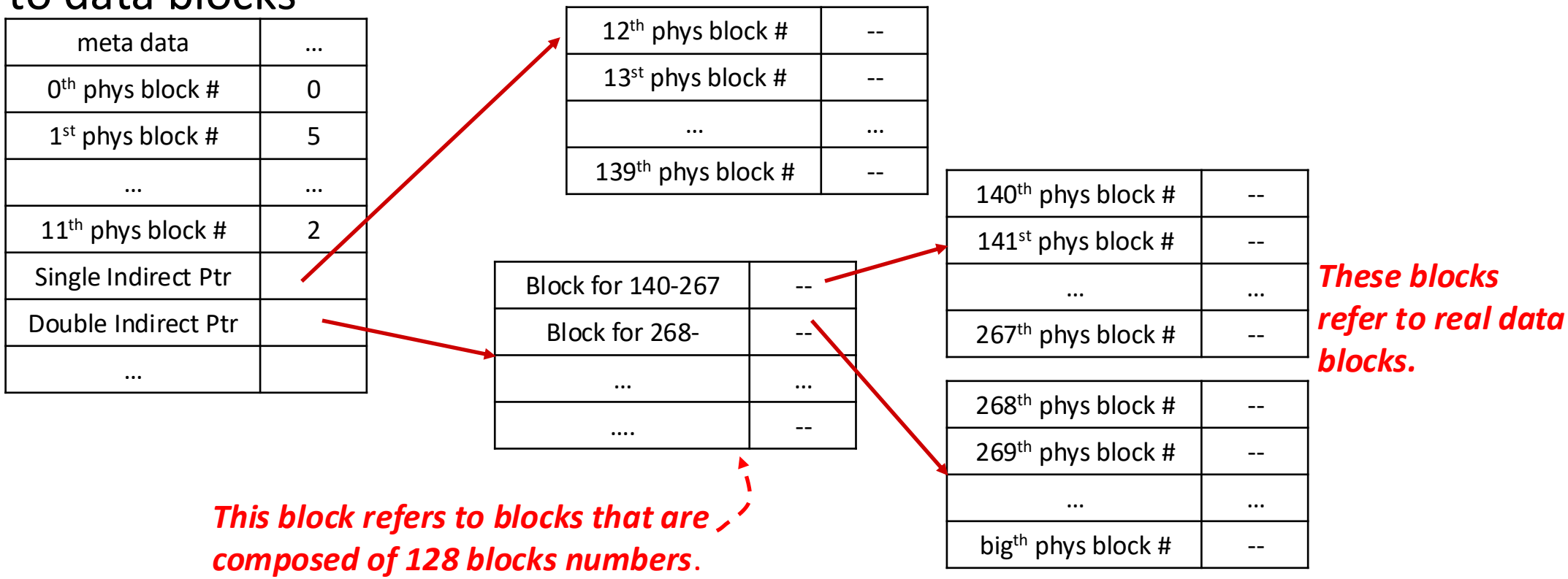
- We can allocate a block to hold more block numbers

If each block is 512 bytes, we can hold 128 block #s in a single block.



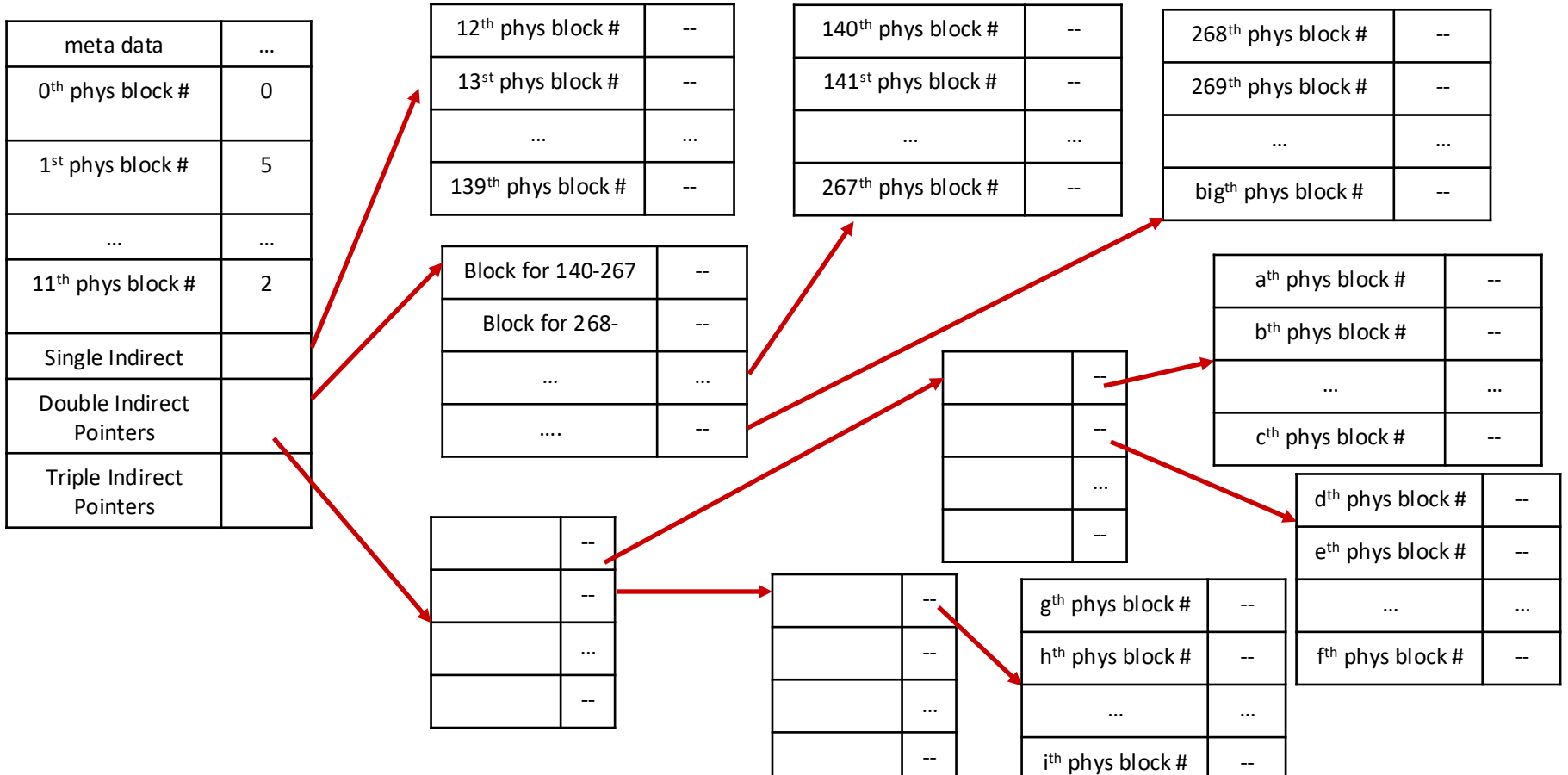
# We need moreeeeee

- What if a file needs more than 140 blocks?
- Add another field to the inode that refers to a block that refers to other blocks that refer to data blocks



# MORE MORE MORE MORE MORE MORE MORE

- What if our file needs more than that?
  - We can add another field to our Inode that refers to a pointer block that refers to pointer blocks that refer to data blocks…



# More?

- No more (at least on Linux ext2)
- If you need more space than this, the operating system will tell you **no**

Poll Everywhere

pollev.com/cis5480

# More?

- No more (at least on Linux ext2)
- If you need more space than this, the operating system will tell you **no**

```
struct inode_st {
  attributes_t metadata;
  block_no_t blocks[12];
  block_no_t *single_ind;
  block_no_t **double_ind;
  block_no_t ***triple_ind;
};
```

What is the *largest file possible if each block is 512 bytes and each block_n_t is 4 bytes?*

Poll Everywhere

pollev.com/cis5480

- How is this better than FAT?

# Lecture Outline

- Inodes
- **Directories**
- Block Caching

# Directory Entries with Inodes

- With FAT we said a directory entry had:
  - The file name
  - The number of the first block of the file

- With Inodes, we instead store the inode number for the file in the directory entry

# Reminder: Directories

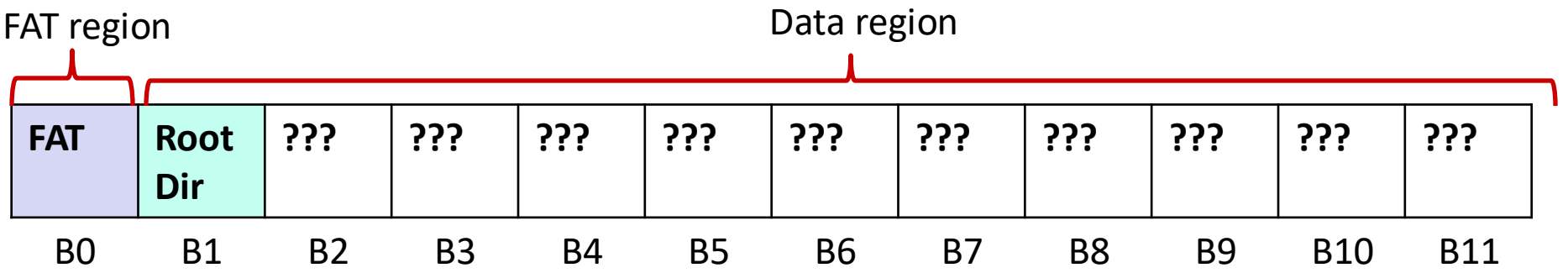
- A directory is essentially like a file
  - We will store its data on disk inside of blocks (like a file)

- The directory content format is known to the file system.
  - Contains a list of directory entries
  - Each directory entry contains the name of the file, some metadata and…
    - If using Inodes, the inode for the file
    - If using FAT, the first block number of the file

  - I know we just said Inodes are better and more modern, but PennOS uses FAT (:/) so my examples will follow that, it is not much different for Inodes though

# Review: Directories

- In FAT our file system looked something like this:
  - 2 regions, and assuming FAT is just 1 block

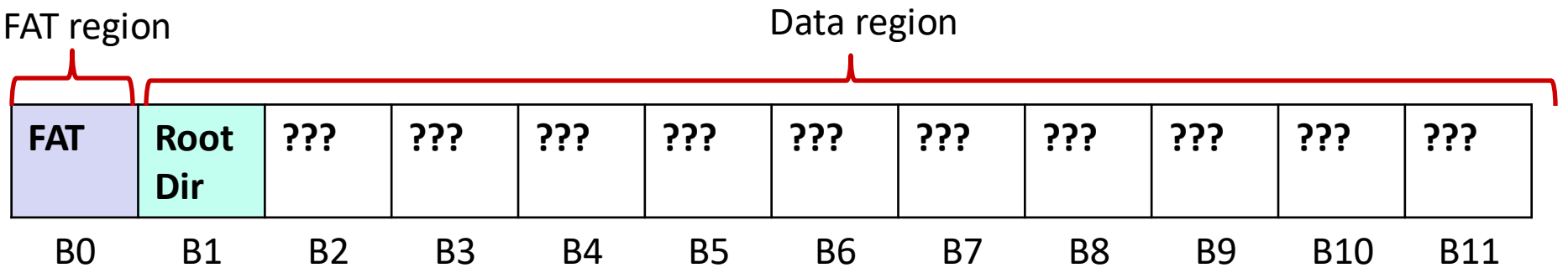



- And the root Directory contains a list of directory entries

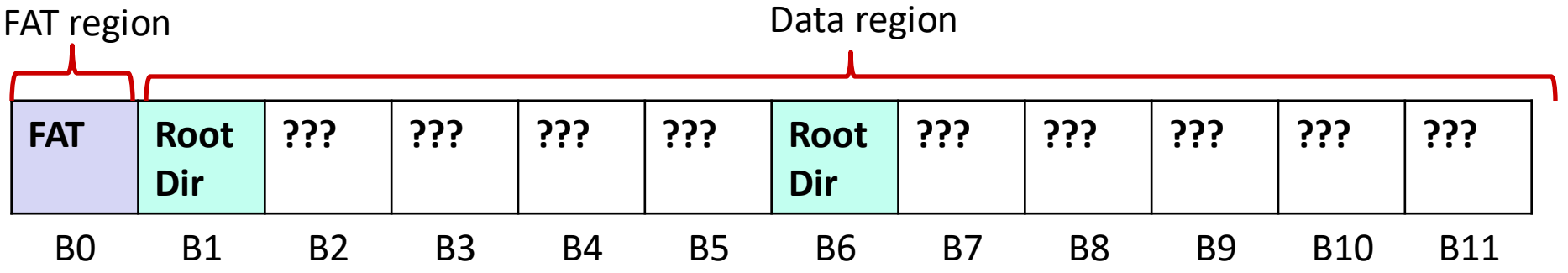
| File Name | Block Number |
|---|---|
| A | 7 |
| B | 4 |
| C | 9 |
| D | 2 |
| E | 10 |

# Growing a Directory

- In FAT our file system looked something like this:
  - 2 regions, and assuming FAT is just 1 block




- What happens if the root directory starts filling up?
  - The root directory is itself a file, it can expand to another block



# Growing a Directory

- We would also need to update the FAT to account for this change.
  - Root directory in PennFAT starts at index 1 into the data region
  - Index 1 into the data region is the first block in the data region 

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | END |
| … | … |
| …. | … |
| … | … |
| 6 | EMPTY |
| 7 | EMPTY |
| … | … |

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 6 |
| … | … |
| …. | … |
| … | … |
| 6 | END |
| 7 | EMPTY |
| … | … |

# Poll Yourself

Discussss

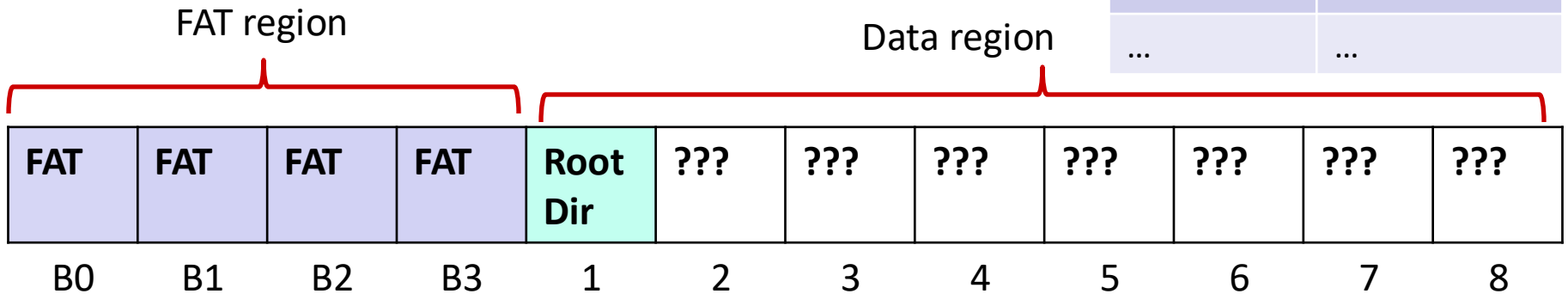
- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

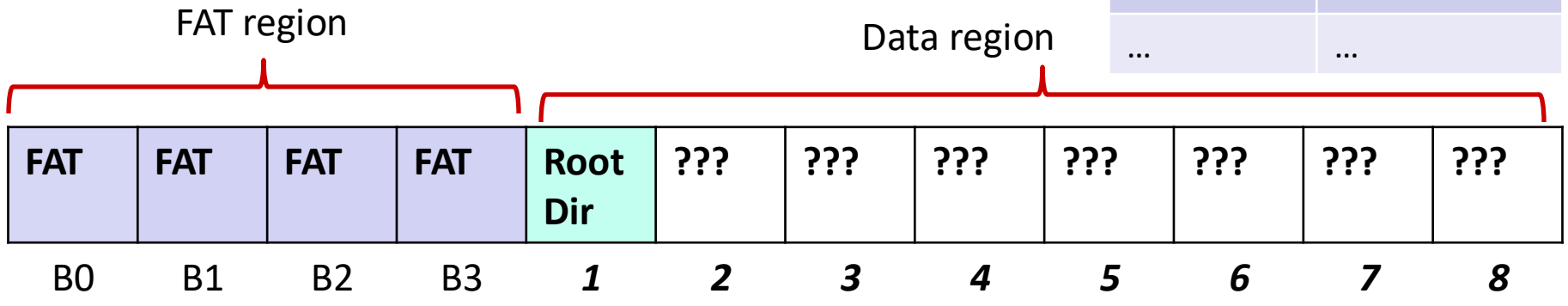
Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |

**Hint: Index into data region starting at index 1**



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

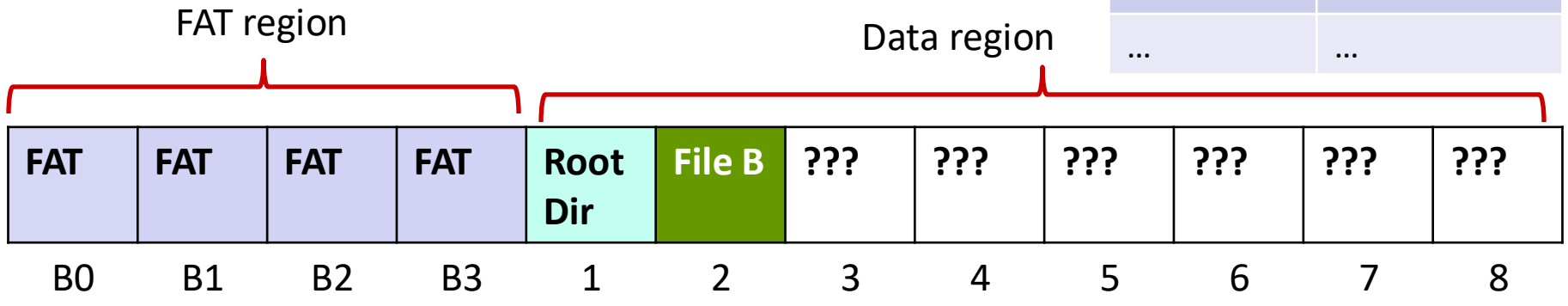
**Hint: Index into data region starting at index 1**

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

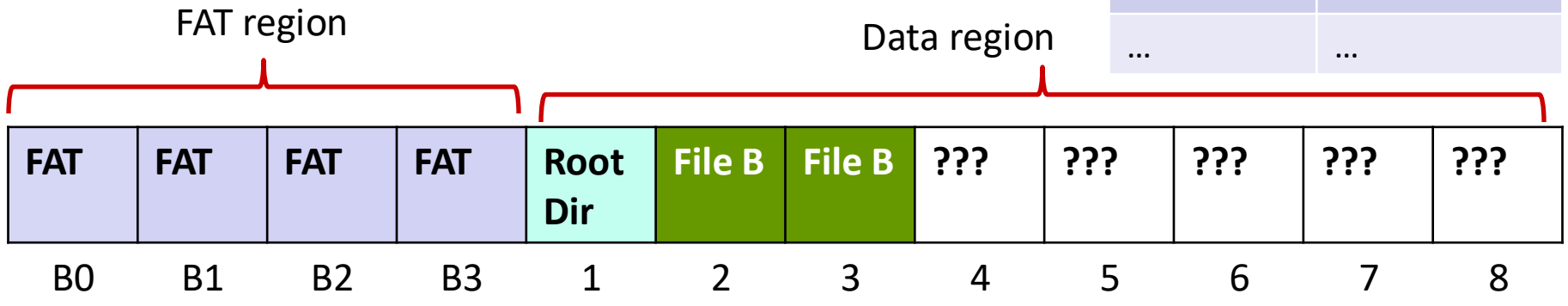
**Hint: Index into data region starting at index 1**

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

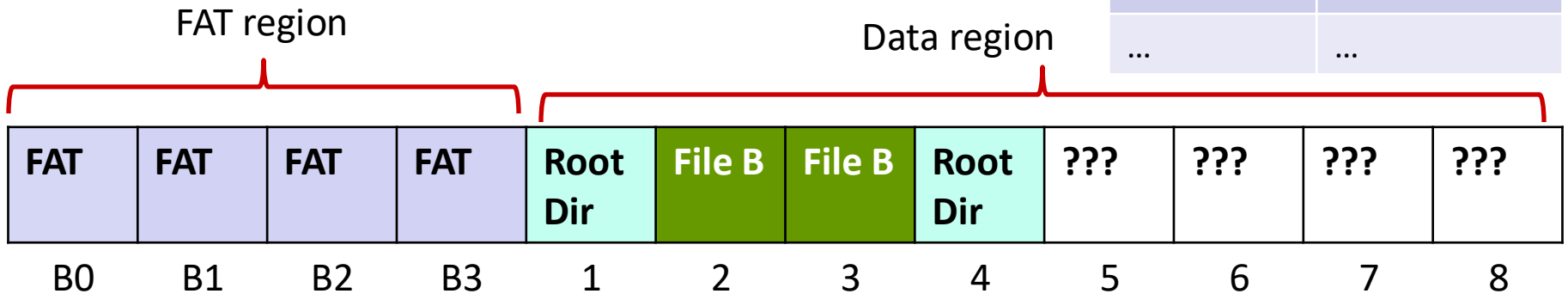
**Hint: Index into data region starting at index 1**

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

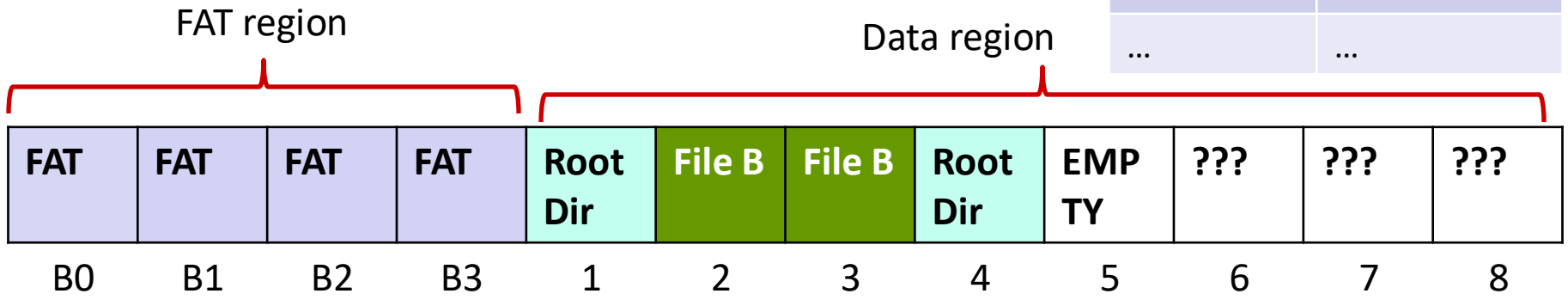
**Hint: Index into data region starting at index 1**

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

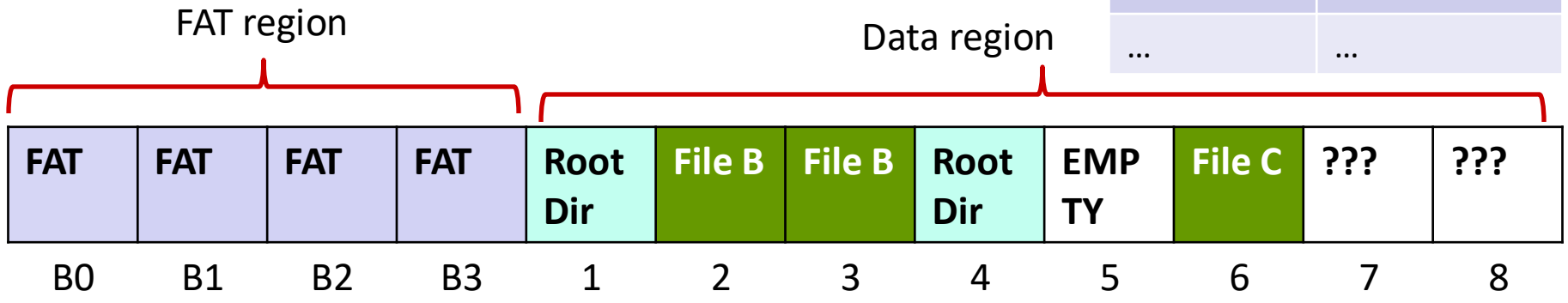
Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |

**Hint: Index into data region starting at index 1**



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

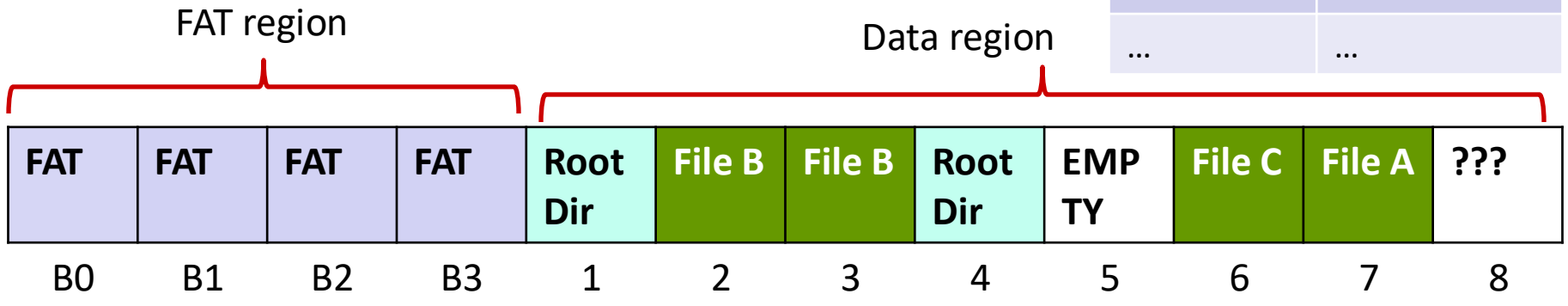
**Hint: Index into data region starting at index 1**

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



# Poll Yourself

Discussss

- Let's say PennFAT is 4 blocks
- What are value of the remaining blocks in the diagram?

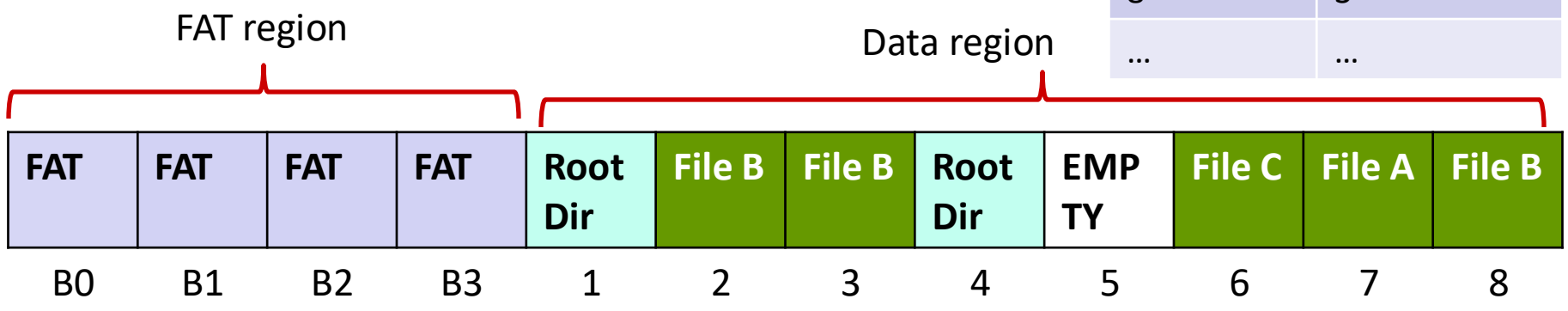
**Hint: Index into data region starting at index 1**

Root DIR

| File Name | Block Number |
|---|---|
| A | 7 |
| B | 2 |
| C | 6 |

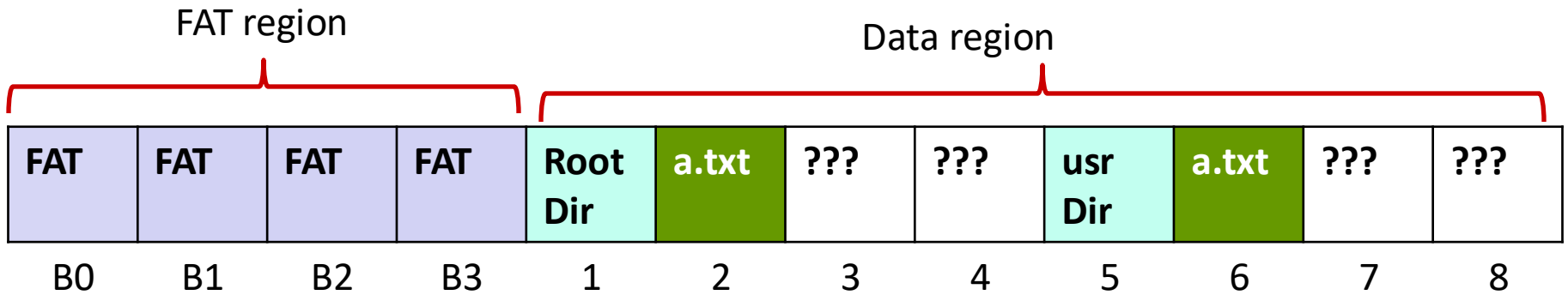
FAT

| Block # (FAT Index) | Next (FAT value) |
|---|---|
| 0 | METADATA |
| 1 | 4 |
| 2 | 8 |
| 3 | END |
| 4 | END |
| 5 | EMPTY |
| 6 | END |
| 7 | END |
| 8 | 3 |
| … | … |



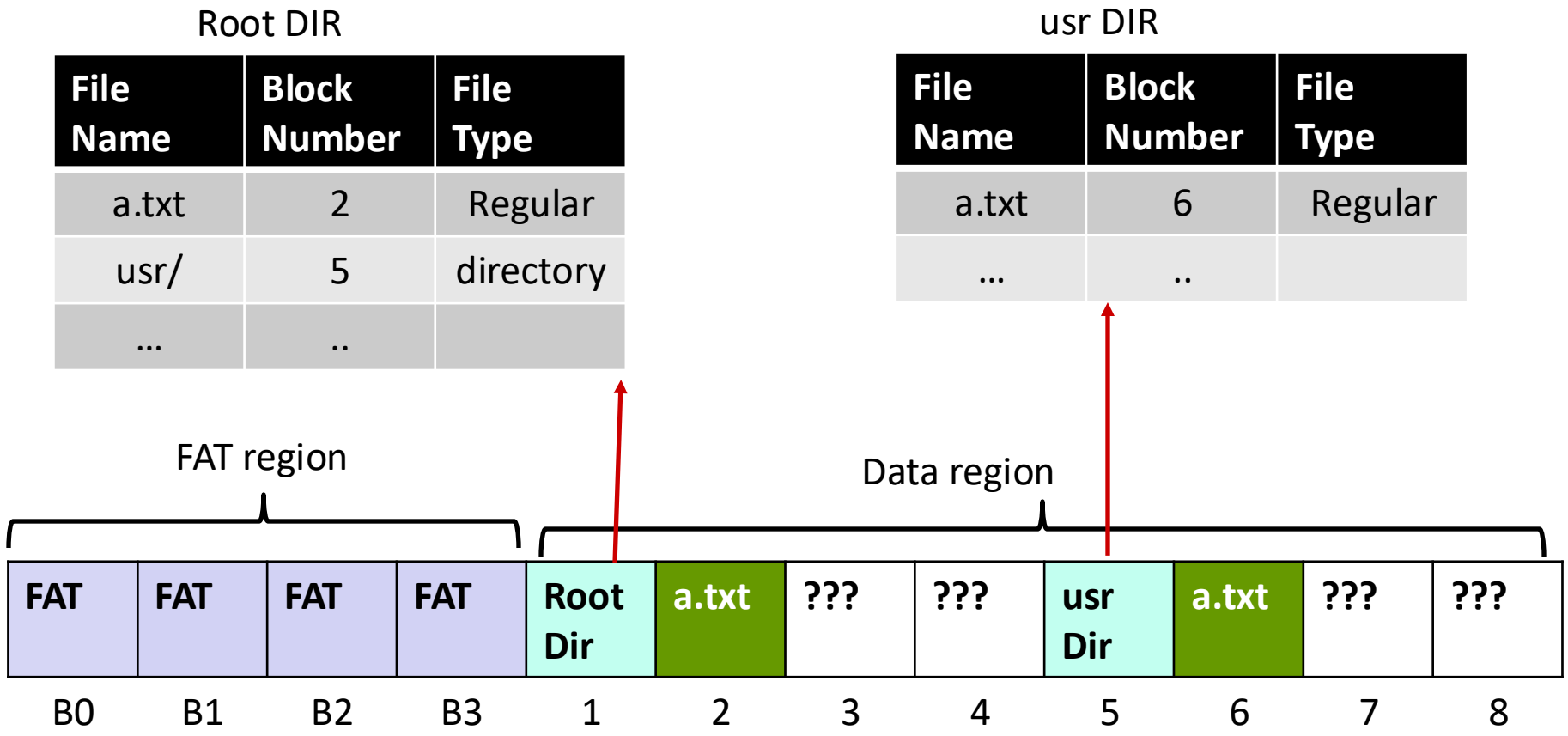
# Sub Directories

- In PennOS, we are only required to deal with 1 directory, but you can implement sub-directories.
  - Sub directories are just other (special) files
- Consider we have the following two directories and files
  - /a.txt
  - /usr/a.txt
  - Above are two separate files!



# Sub Directories

- We would also have some information in a directory entry to specify what kind of file it is



# . and ..

- It would be useful to support . and . .
  - . Refers to the current directory, . . refers to parent directory

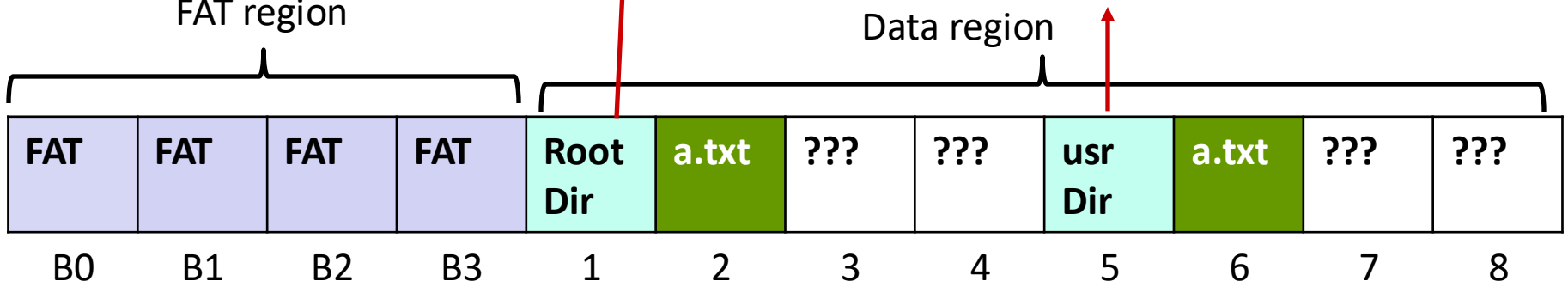
root DIR

| File Name | Block Number | File Type |
|---|---|---|
| . | 1 | directory |
| .. | 1 | directory |
| a.txt | 2 | Regular |
| usr/ | 5 | directory |
| … | .. | |

Has no parent, refers to self

usr DIR

| File Name | Block Number | File Type |
|---|---|---|
| . | 5 | directory |
| .. | 1 | directory |
| a.txt | 6 | Regular |
| … | .. | |



# Lecture Outline

- FAT & PennFAT wrap-up
- Inodes
- Directories
- **Block Caching**

# Block Caching

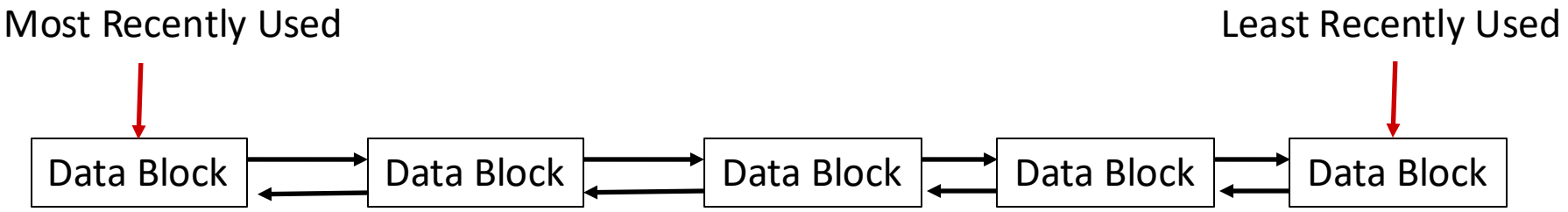
- Disk I/O is really slow (relative to accessing memory)

- What can we do instead to make it faster?
  - Keep data that we want to access in memory ☺
  - We already did this with FAT and Inodes for open files

- We can do the same for data blocks we think we may use again in the future

# Block Caching Data Structure

- We can use a linked list to store blocks in LRU

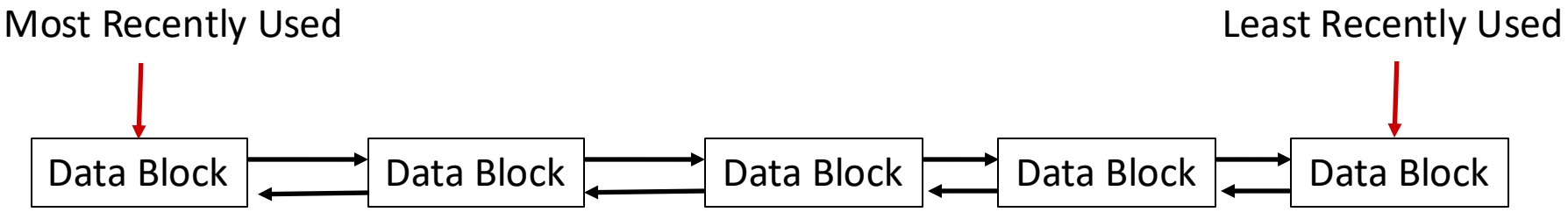



- What is the algorithmic runtime analysis to:
  - lookup a specific block?
  - Removal time of LRU?
  - Time to move a block to the front or back?
    - Consider search time

**Discuss**

# Block Caching Data Structure

- We can use a linked list to store blocks in LRU




- What is the algorithmic runtime analysis to:
  - lookup a specific block? O(n)
  - Removal time of LRU? O(1)
  - Time to move a block to the front or back? O(n)
    - Consider search time

Discuss

Is there a structure we know of that has O(1) lookup time?

# Chaining Hash Cache

- We can use a combination of two data structures:
  - `linked_list<block>`
  - `hash_map<block_num, node*>`