Inodes & Dirents in-depth

Operating Systems, Fall 2026

Instructors: Travis McGaha

Head TAs: Shriya Sane

Arnuv Batra
Khoi Dinh
Sydney Fitzgerald
Rania Souissi
Jeffrey Oduman
Abdullah Goher
Monisha Krothapalli
Veer Kakar
Kevin Xue
Shreya Mukunthan
Daniel Price
Wesley Liu
Leo Zhang

Any Questions?

Logistics

- Penn-FAT is out!
  - Partner sign-up is out now! Due end of day October 2nd.
  - It is due October 29th in pairs!
  - It is a HUGE assignment. And make sure to put in a lot of elbow grease.
  - Some questions on the midterm will ask about concepts enforced by this assignment.

- MAKE SURE CANVAS IS CONSISTENT WITH WHO YOUR PARTNER IS.

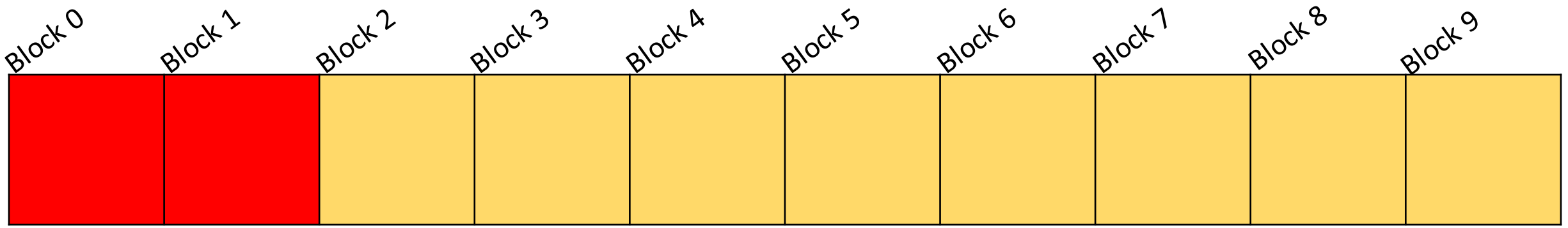
- Penn-fat milestone details posted soon!

Lecture Outline

- Linux Filesystem Implementation
  - Quick Review
  - Reserved Inodes & Root Inode
- File Paths
  - Absolute Paths & Relative Paths
- Resolving Absolute Paths
  - Directory Entries
- Accessing a Struct Dirent

Review: Filesystem and inodes

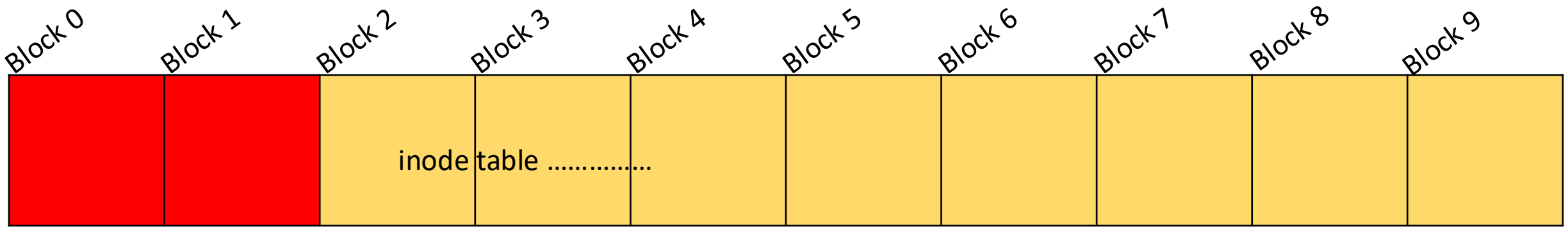
- The filesystem is composed of blocks that are at the smallest size, 512 bytes.
  - Today, filesystems blocks are much larger as we saw previously.
  - On some, blocks reach up to 4096 bytes (4 KB).
- Linux Filesystem ext2, blocks smallest size is 1024 bytes.
- Physical Block Numbers start from 0. They correspond to ***physical blocks*** of memory on the memory device (e.g. hdd, ssd, sd cards, cds, floppy disk)



note: this is an attempt to present information as truthful as possible while trying to make it digestible

Review: Filesystem and inodes

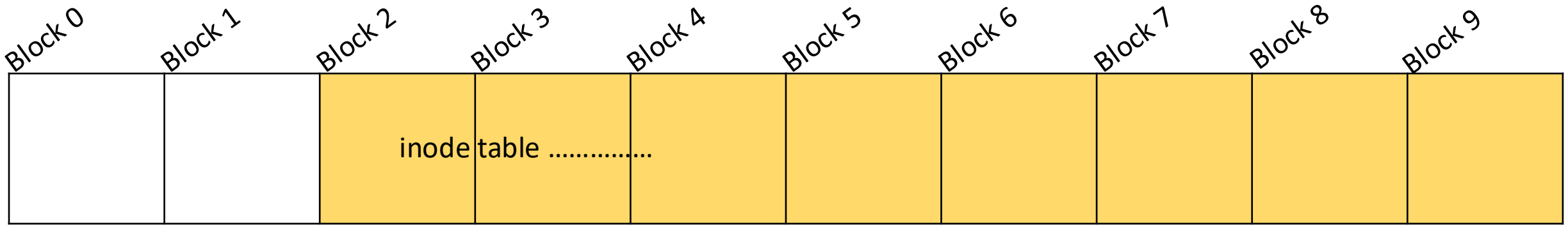
- From our perspective, the inode table starts on Block 2
  - Each block in the inode table is full of inodes, *even if some inodes do not refer to a file*
    - *(i.e. is unallocated)*
  - You can also think of the inode table as: `inode_array[]`
    - Where the `inode_array` is split up across blocks



note: this is an attempt to present information as truthful as possible while trying to make it digestible

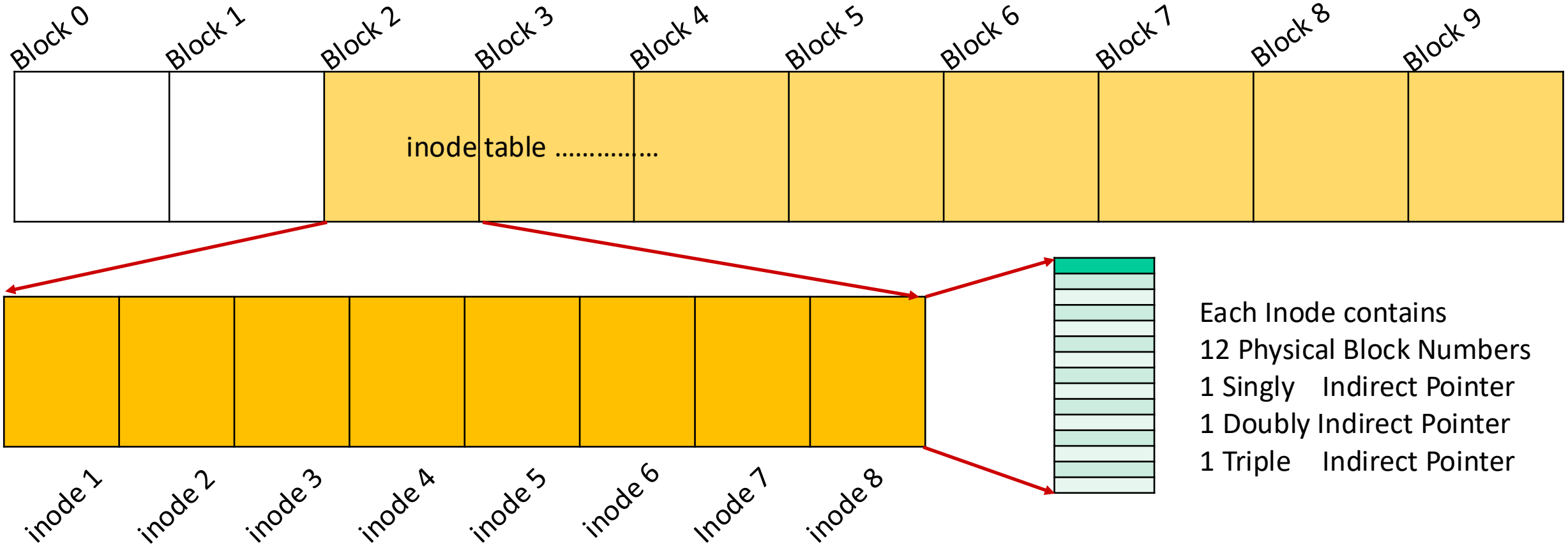
Review: Filesystem and inodes

- From our perspective, the inode table starts on Block 2
  - Each block in the inode table is full of inodes, *even if some inodes do not refer to a file*
    - *(i.e. is unallocated)*
  - You can also think of the inode table as: `inode_array[]`
- Let's assume;
  - Block Size: 1024 Bytes
  - Inode Size: 128 bytes
    - Yes. They're that big in ext2.



note: this is an attempt to present information as truthful as possible while trying to make it digestible

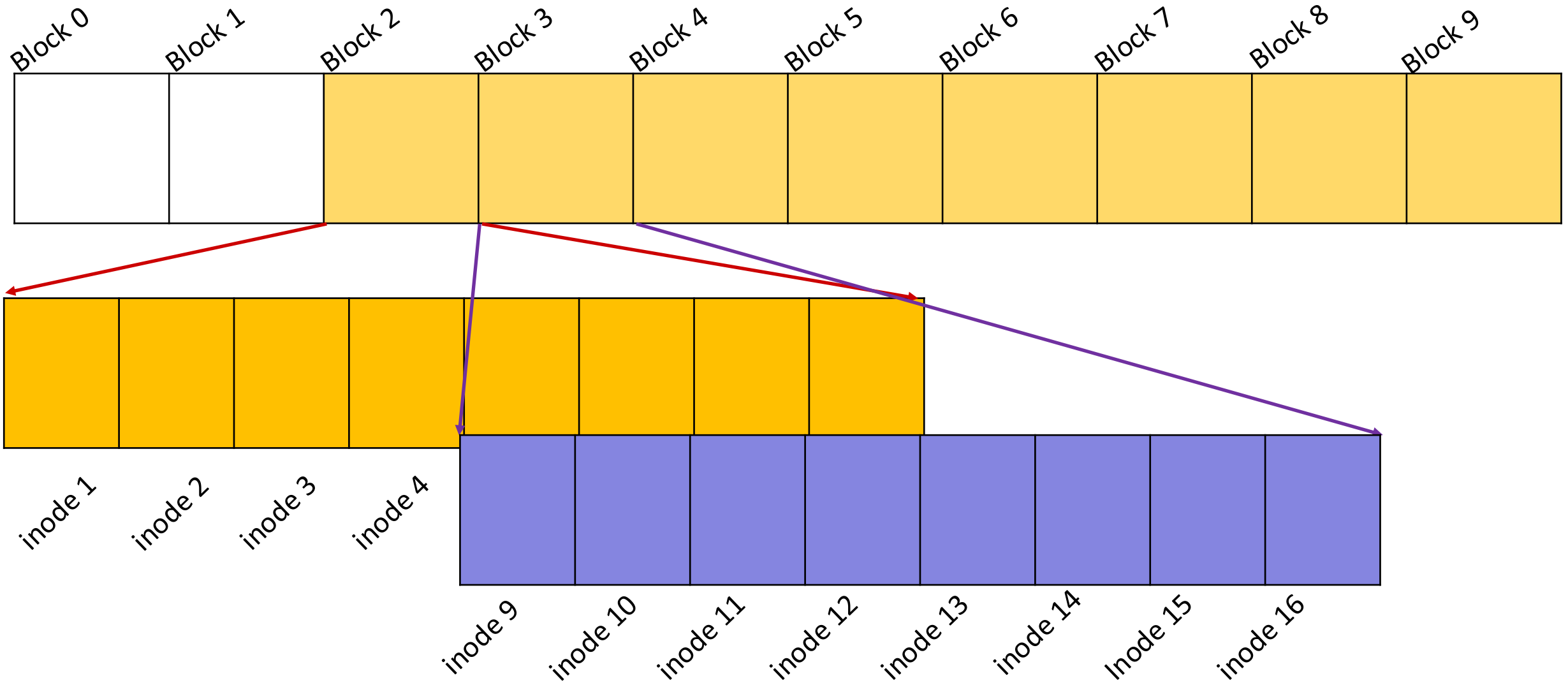
Inodes at a glance: Physical Block 2



Each Inode contains
12 Physical Block Numbers
1 Singly Indirect Pointer
1 Doubly Indirect Pointer
1 Triple Indirect Pointer

*we say pointer, but they're physical block numbers.

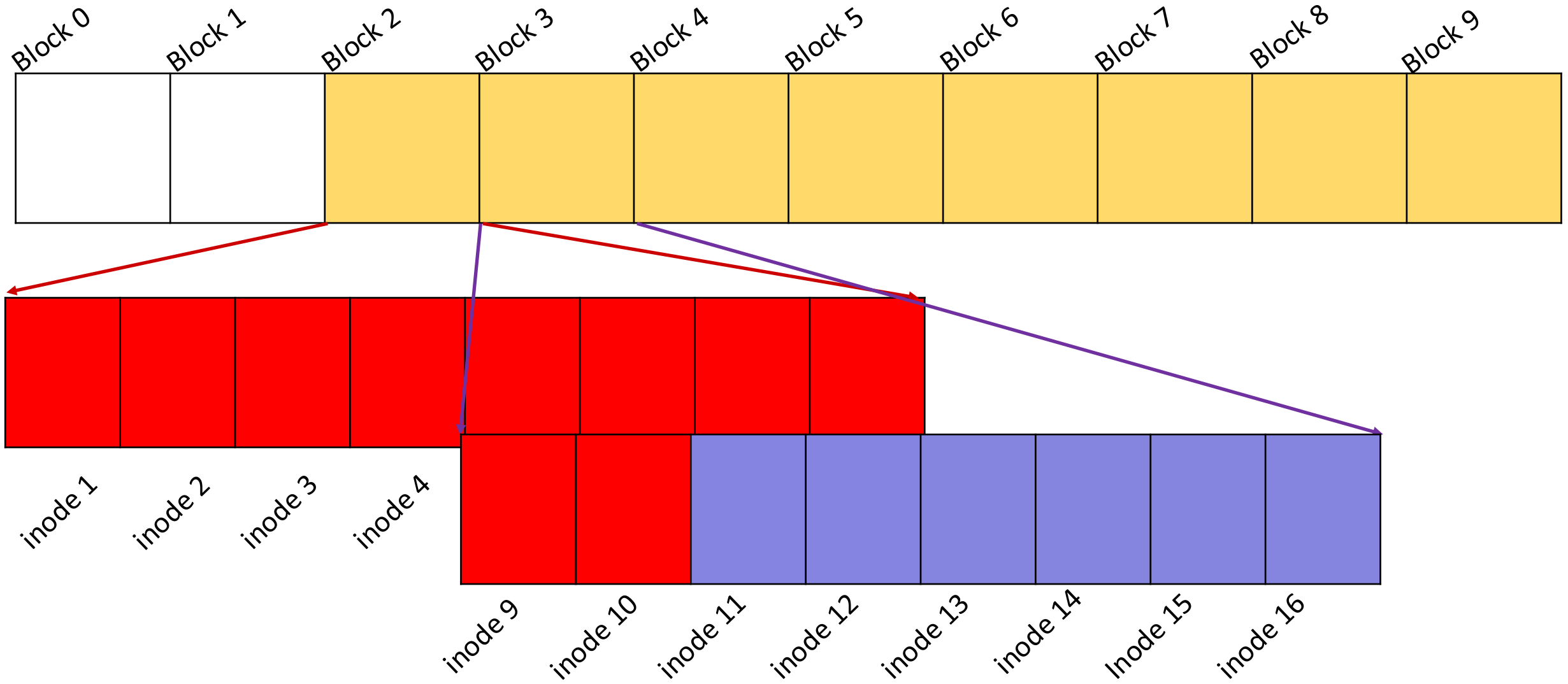
Inodes at a glance: Physical Block 2



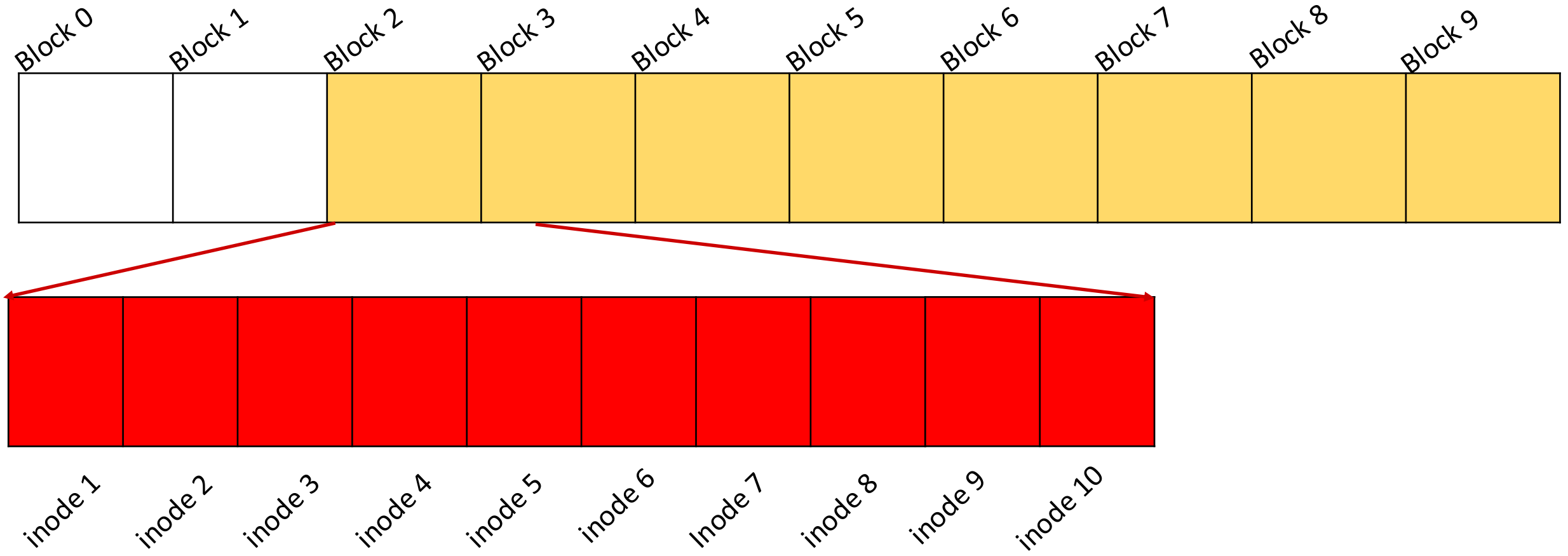
Lecture Outline

- Linux Filesystem Implementation
  - Reserved Inodes
  - Root Inode
- File Paths
  - Absolute Paths & Relative Paths
- Resolving Absolute Paths
  - Directory Entries
- Accessing a Struct Dirent

Inodes at a glance: Reserved Inodes

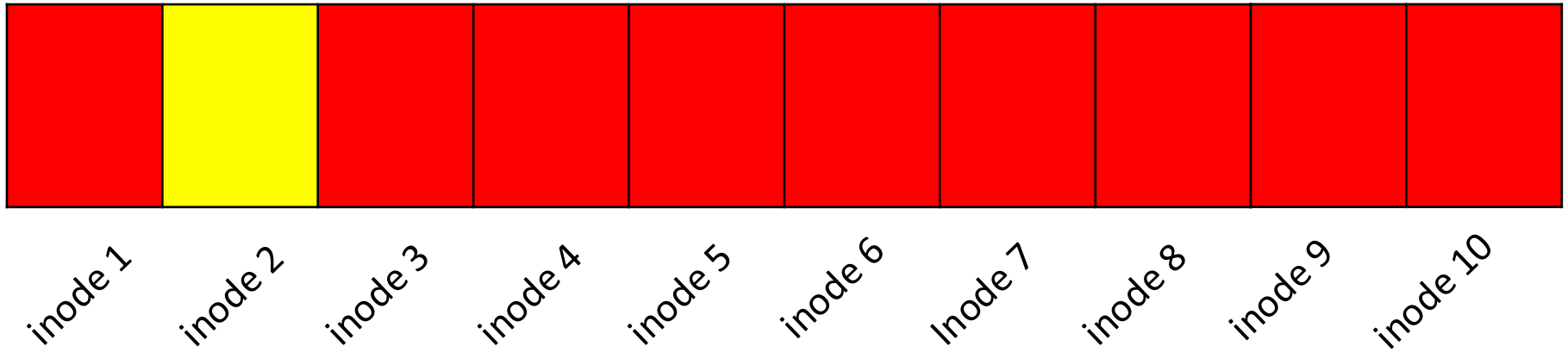


Inodes at a glance: Reserved Inodes



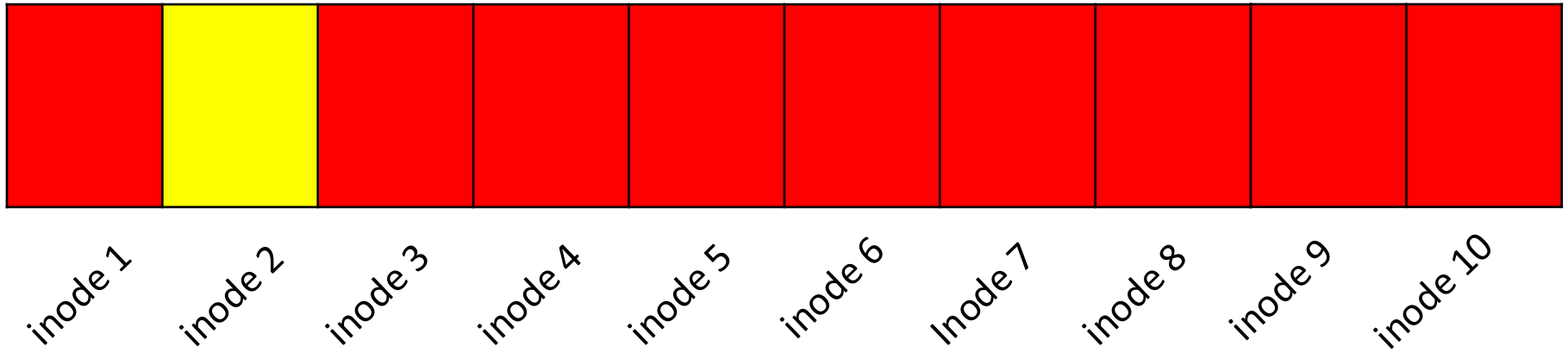
There are up to 10 inodes that are reserved for special purposes, although many are unused and were never implemented for their use case.

The Most Important Special Inode: Inode 2



- Inode 2
  - Is the “root” directory of the filesystem (i.e. '/')
  - Does not contain the physical file information for the root directly
  - The block numbers within the Inode tell us ***where*** to expect the data to be.

The Most Important Special Inode: Inode 2



- Inode 2
  - Is the “root” directory of the filesystem (i.e. '/')
  - `/Users/joelrmrz/Documents/file.txt`

The corresponding inode for this directory is inode 2.

Wait, so where is the rest of “`Users/joelrmrz/Documents/file.txt`” ?

Lecture Outline

- Linux Filesystem Implementation
  - Quick Review
  - Reserved Inodes & Root Inode
- File Paths
  - Absolute Paths & Relative Paths
- Resolving Absolute Paths
  - Directory Entries
- Accessing a Struct Dirent
- Putting It All Together
  - Bitmaps
  - Super Block
  - Boot Block

Resolving Paths; Looking for a file

- Let's say I want to find: **/Users/joelrmrz/Documents/file.txt**
  - How would you usually go about it?
- You'd probably search for the Users directory first,
  - then **joelrmrz**,
  - then **Documents,**
  - **and finally, file.txt.**
- **The way absolute pathnames are resolved is very similar, albiet, with a bit more technical details. ☺**

Absolute Paths

- **/Users/joelrmrz/Documents/file.txt**
  - Is an example of an absolute path, where the entire path, starting from root, /, is given.
- Not an absolute path: file.txt
  - Doesn't tell us ***where in the file system*** we can find a file with that name unless we traverse the whole thing.

```
/root
├── dir_one
│   ├── file.txt
│   └── subdir_one
├── dir_two
│   ├── file.txt
│   └── subdir_two
├── dir_three
│   ├── file.txt
│   └── subdir_three
```

Which `file.txt` are we referring to?
No clue.

Relative Paths

- **./file.txt**
  - Is an example of a relative path, where the ***starting directory*** is relative to the "."
  - Recall that a "." is a self reference to a directory.
  - Does tell us ***where in the file system*** we can find it, namely, in the directory we are 'in'.

```
/root
├── dir_one
│   ├── file.txt
│   └── subdir_one
├── dir_two
│   ├── file.txt
│   └── subdir_two
├── dir_three
│   ├── file.txt
│   └── subdir_three
```

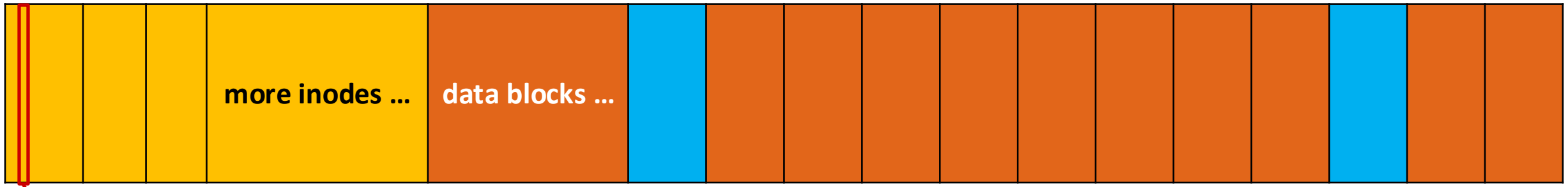
Which `./file.txt` are we referring to?

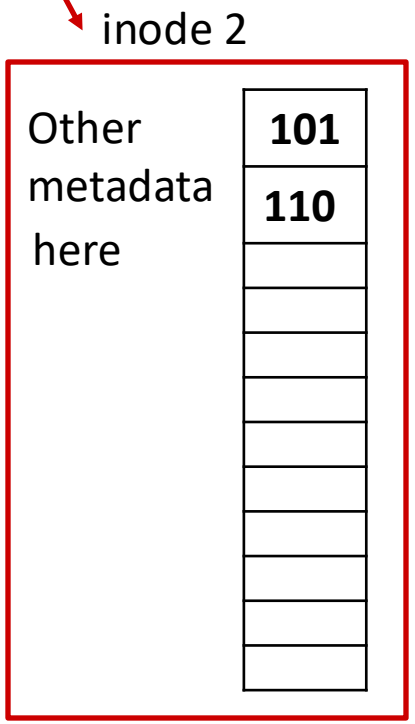
If we are "in" `dir_one`,
then we are referring to `/root/dir_one/file.txt`
And so forth.

Lecture Outline

- Linux Filesystem Implementation
  - Quick Review
  - Reserved Inodes & Root Inode
- File Paths
  - Absolute Paths & Relative Paths
- Resolving Absolute Paths
  - Directory Entries
- Accessing a Struct Dirent
- Putting It All Together
  - Bitmaps
  - Super Block
  - Boot Block

Walkthrough: Resolving an Absolute Pathname



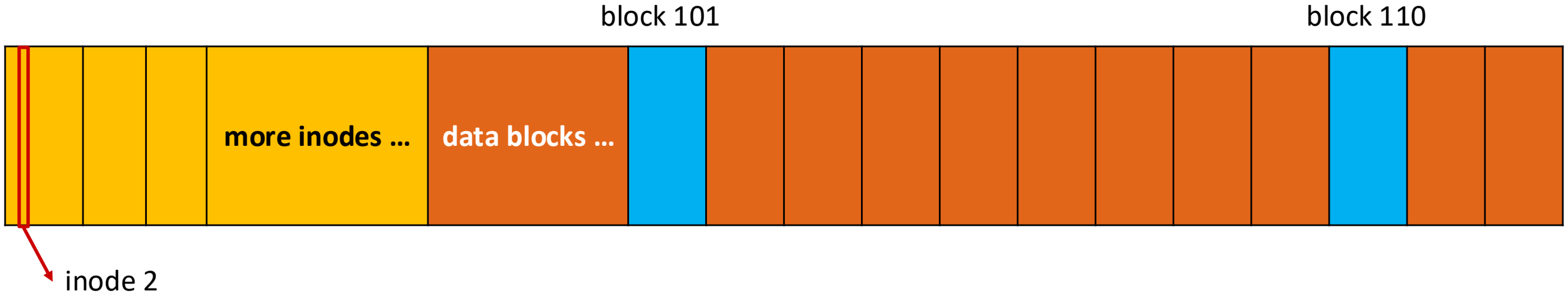


Let's say I want to find: **/Users/joelrmrz/Documents/file.txt**

First: we need to look for the "Users" directory starting from the root directory. But, ***which blocks are the root directory?***

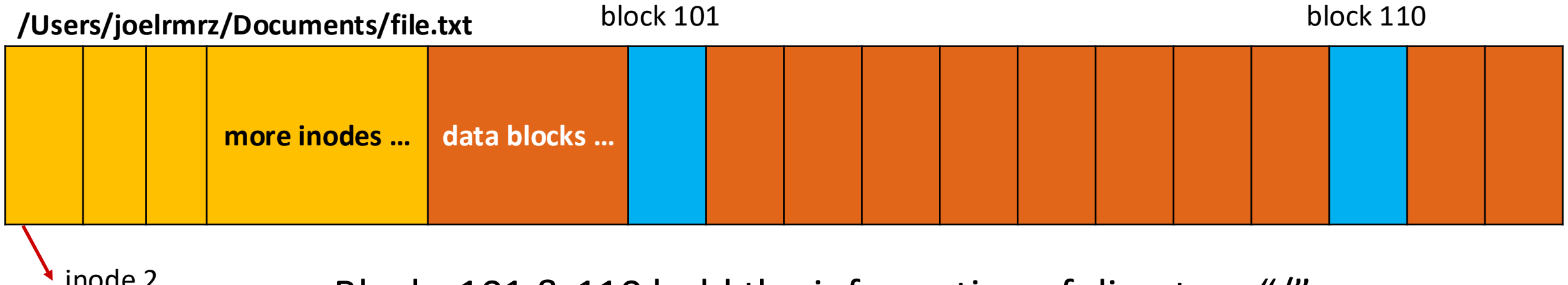
The inode tells us where! Blocks 101 and 110!

Walkthrough: Resolving an Absolute Pathname



Let's go ahead and traverse the information in these blocks to find the directory "Users"

Walkthrough: Resolving an Absolute Pathname

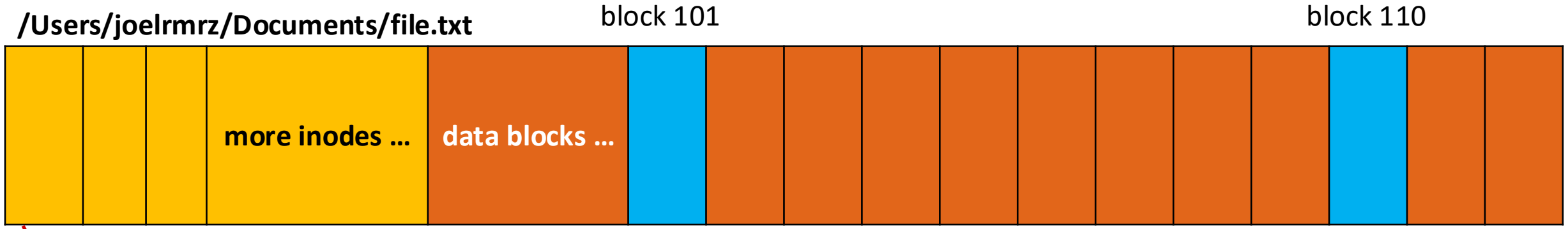


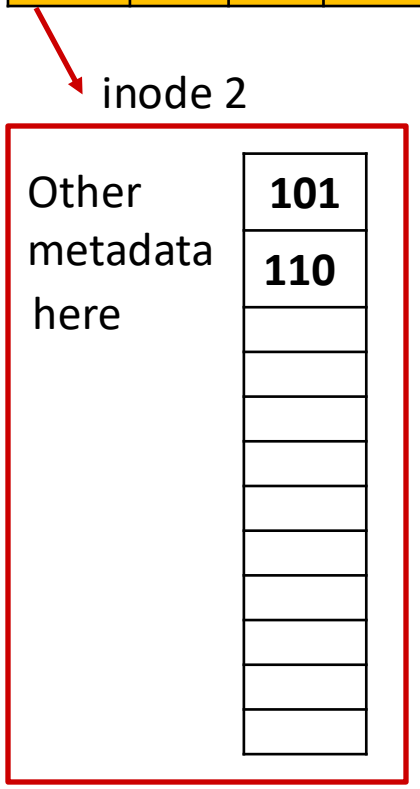
- Blocks 101 & 110 hold the information of directory "/"
  - These are the files/directories we would find **inside* the root directory.
  - We need to look for the ***directory entry*** "`Users`" within the data blocks for the root directory.

*they are not "inside" those directories at all. The directories contain only references to those files.

Walkthrough: Resolving an Absolute Pathname

/Users/joelrmrz/Documents/file.txt





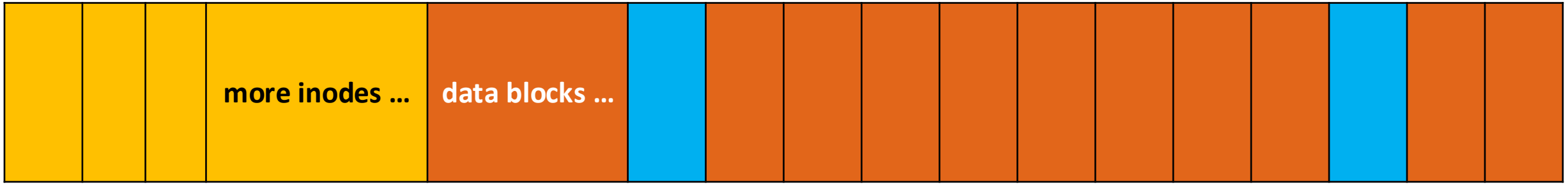
- The directory entries in the root directory.

```
/root
├── bin
├── usr
├── USERS
├── .
├── ..
│
├── tmp
├── dev
```

These are the “entries” within the root directory.
Shorted to the name “dirent” for directory entry.

A Necessary Evil: Directory Entries

/Users/joelrmrz/Documents/file.txt



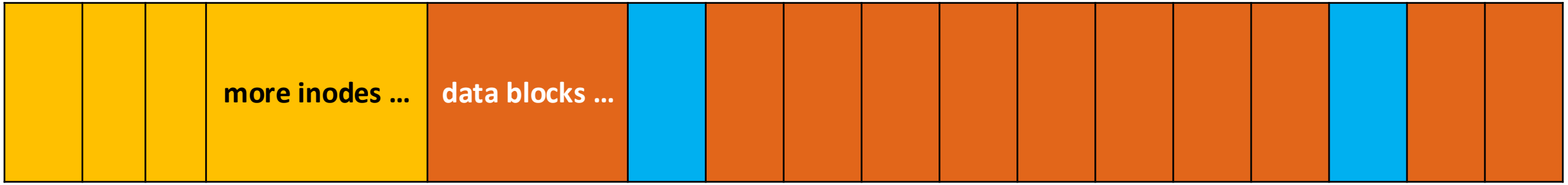
- Directory Entries
  - Are structs that ***represent*** the files/subdirectories within a directory.

```
struct dirent {
    ino_t d_ino;       /* File inode number */
    uint8_t d_type;    /* Indicator of file type ONLY IN ext2, 3, 4 */
    char d_name[];     /* Null terminated name of file */
}
```

- If we come across **d_name == "Users",** we know where to look for "joelrmrz" now
  - namely, in the data blocks for the corresponding inode for this entry, **d_ino**.

A Necessary Evil: Directory Entries

/Users/joelrmrz/Documents/file.txt



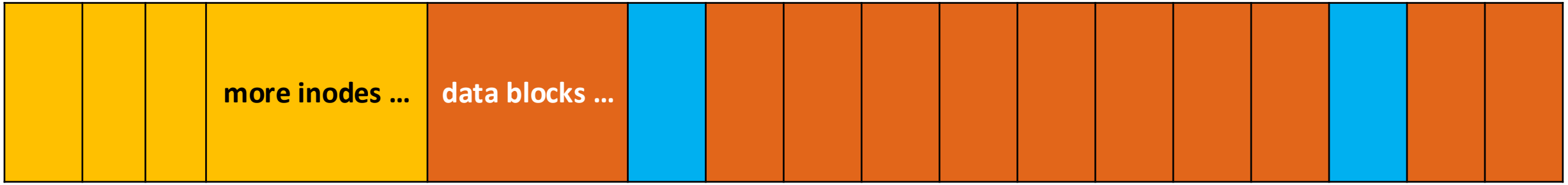
- Directory Entries
  - Are structs that ***represent*** the files/subdirectories within a directory.

```
struct dirent {
    ino_t d_ino;       /* File inode number */
    uint8_t d_type;    /* Indicator of file type ONLY IN ext2, 3, 4 */
    char d_name[];     /* Null terminated name of file */
}
```

- d_type
  - Not defined in ALL file systems but is useful in telling us the ***type*** of file we are referring to.
  - Directory, Regular File, and much more.

A Necessary Evil: Directory Entries

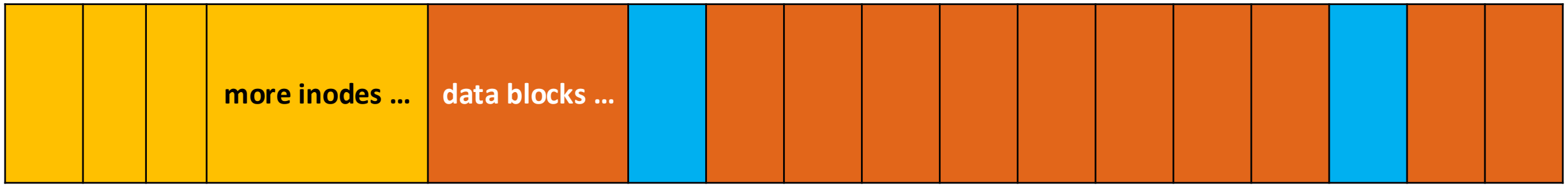
/Users/joelrmrz/Documents/file.txt



- Directory Entries
  - These ***exist within the file system itself;*** the inode tells us the data blocks that contain these.
  - *You know where to start looking,* **inode 2.**
  - **You can not just start reading the values in a directory via "open"; if only life were that simple.**
    - **More on that later….**

Walkthrough: Resolving an Absolute Pathname

/Users/joelrmrz/Documents/file.txt



inode 2

Other metadata here

101
110

Block 101

entry 1	inode: 2	d_name: "."
entry 2	inode: 2	d_name: ".."
entry 3		
….		
entry 10	inode: 107	d_name: ”Users"
entry 11	inode: 243	d_name: ”bin"

}

Yup, these have their own directory entries.

We are in the root directory, so "..” just refers to the root again as it has no parent.

Awesome! Here it is!
But, ***where*** is Inode 107?

Poll Everywhere

pollev.com/tqm

Which physical block contains inode 107?

Assume:

- The inode table starts at Block 2
- A block is 1024 bytes
- Each inode is 128 bytes

First ask yourself:
How would I find Inode 8? What about inode 15?

We'll give you around 10 minutes to figure it out…really try to find out!

You'll have to do similar math to find which physical block FAT entries are in, you can't just index normally. You need to load the block first.

reminder: we start counting inodes from 1 ***(there is no inode 0)***

Poll Everywhere

pollev.com/tqm

Which physical block contains inode 107?

Left blank for any work you'd like to do here:

Poll Everywhere

pollev.com/tqm

What is the index of the inode 107 relative to the block it is in?

e.g. is it the first inode in the block, second, or...

Assume:

- The inode table starts at Block 2
- A block is 1024 bytes
- Each inode is 128 bytes

First ask yourself:

What index is Inode 8 in its block? What about inode 15?

reminder: we start counting inodes from 1 ***(there is no inode 0)***

Poll Everywhere

pollev.com/tqm

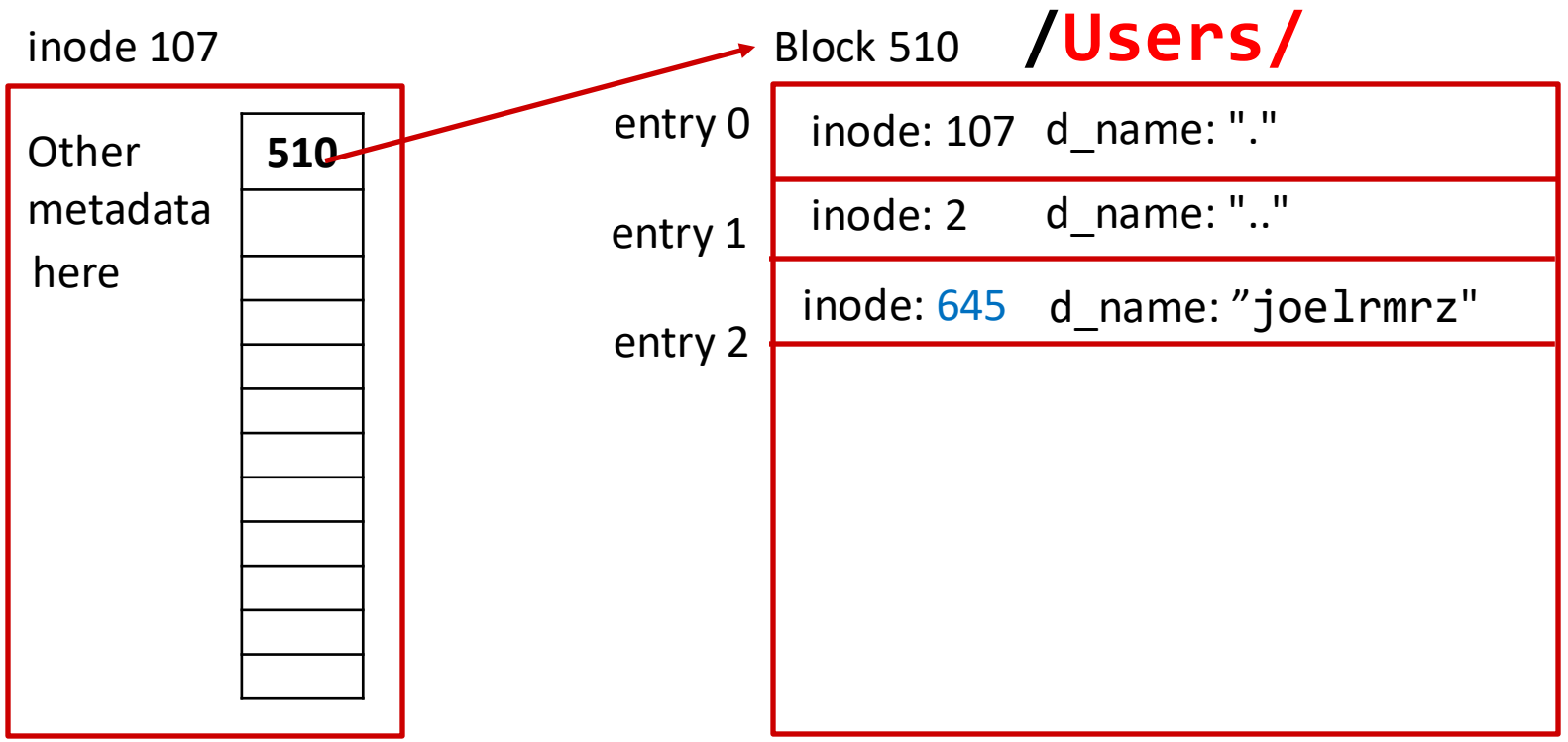
What is the index of the inode 107 relative to the block it is in?

e.g. is it the first inode in the block, second, or...

Left blank for any work you'd like to do here:

Directories in the Filesystem

- Now that we know where inode 107 is, let's look at its directory entries
  - Now, "." is the current directories inode and ".." is the root directory inode!
  - And there is only one user on my computer, so we only have 3 entries in this directory.
  - Now, we go to inode 645 and continue on doing the same thing.



We are currently "inside" the Users/ directory; or rather examining the entries for this directory.

note: there is no natural ordering to the dirents.

Resolving Absolute Paths for *Regular Files*

If directories are huge, this requires navigating singly, doubly, or even triple indirect blocks.

- To open `/dir_one/dir_two/dir_three/file.txt`:
  1. Start from the root directory (inode 2).
  2. Check the data blocks associated with this inode.
  3. If you reached the end of the path and it's not a directory, you've found your file! Otherwise, continue.
  4. Iterate through the directory entries.
  5. If d_name matches the target file or directory, follow its inode (d_inode).
  6. Repeat from step 2 with the new inode.
  7. If d_name is not found at any step, ***the path cannot be resolved.***
     - ***i.e. the file does not exist***

Poll Everywhere

pollev.com/tqm

To open `/dir_one/dir_two/dir_three/file.txt`, what is the minimum number of inodes that must be accessed to fully resolve the path and retrieve the data blocks of file.txt? Assume you have not accessed the root inode.

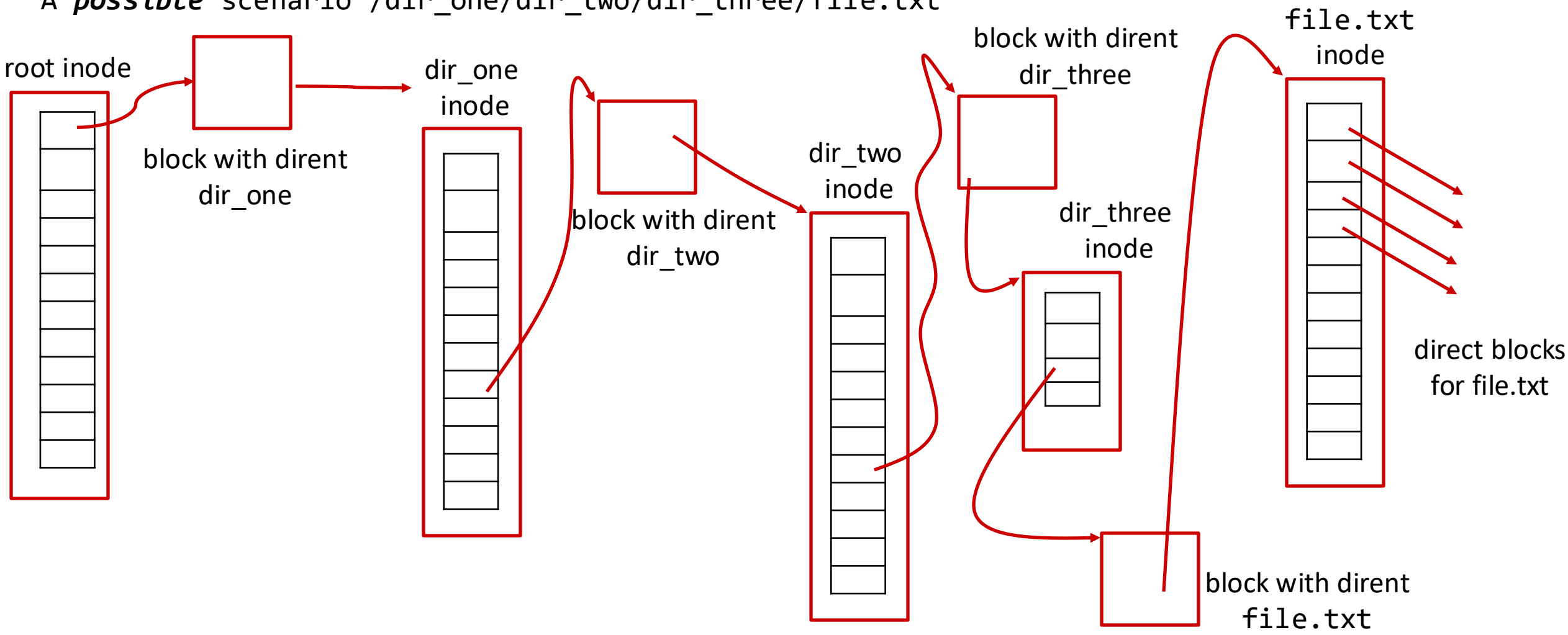
Poll Everywhere

pollev.com/tqm

What is the minimum number of directory entries (struct dirents) that must be traversed to resolve the path `/dir_one/dir_two/dir_three/file.txt` and access the data blocks of file.txt in the best-case scenario? Assume you have not accessed the root inode.

A Possible Scenario for Resolving Paths

A ***possible*** scenario /dir_one/dir_two/dir_three/file.txt



Lecture Outline

- Linux Filesystem Implementation
  - Quick Review
  - Reserved Inodes & Root Inode
- File Paths
  - Absolute Paths & Relative Paths
- Resolving Absolute Paths
  - Directory Entries
- Accessing a Struct Dirent

Accessing a Struct Dirent

```
#include <dirent.h>
```

```
DIR *opendir(const char *pathname);
```

- returns a "directory stream" which is just a maintained iterator over the entries in the directory.

```
struct dirent *readdir(DIR *dir_stream);
```

- returns a struct dirent that is ***allocated by the kernal*** for you (do not free it)
  - Returns the dirent the `dir_stream` is currently pointing to
- The directory entry remains valid until the next call to **`readdir()`** or **`closedir()`** on the same directory stream, `dir_stream`.

```
int closedir(DIR *dir_stream)
```

- Does what you think it does.

Filtering and Sorting Dirents

```
int scandir(const char *restrict pathname,
                struct dirent ***restrict direntry_pointer_array,
                typeof(int (const struct dirent *)) *filter,
                typeof(int (const struct dirent **, const struct dirent **)) *generic_compare);
```

- The **`scandir`**`()` function reads the directory `pathname` (relative or absolute) and builds an array of pointers to directory entries using malloc.
  - It returns the number of entries in the array.
  - A pointer to the array of directory entries is stored in the location referenced by *`direntry_pointer_array`*
- The `filter` argument is a pointer to a user supplied subroutine to select which entries are to be included in the array.
  - Should return a non-zero value if the directory entry is to be included in the array.
  - If `filter` is null, then all the directory entries will be included.
- The `generic_compare` argument is a pointer to a user supplied subroutine which is passed to qsort(3) to sort the completed array. If this pointer is null, the array is not sorted.

This is here purely for completeness. And gives you a way to manually implement the “ls” command in the real world.

Have a good fall break!!!!

- ☺

- Next time:
  - Wrap-up the filesystem: Partitions, Super Block, Boot Block
  - The Scheduler!!!

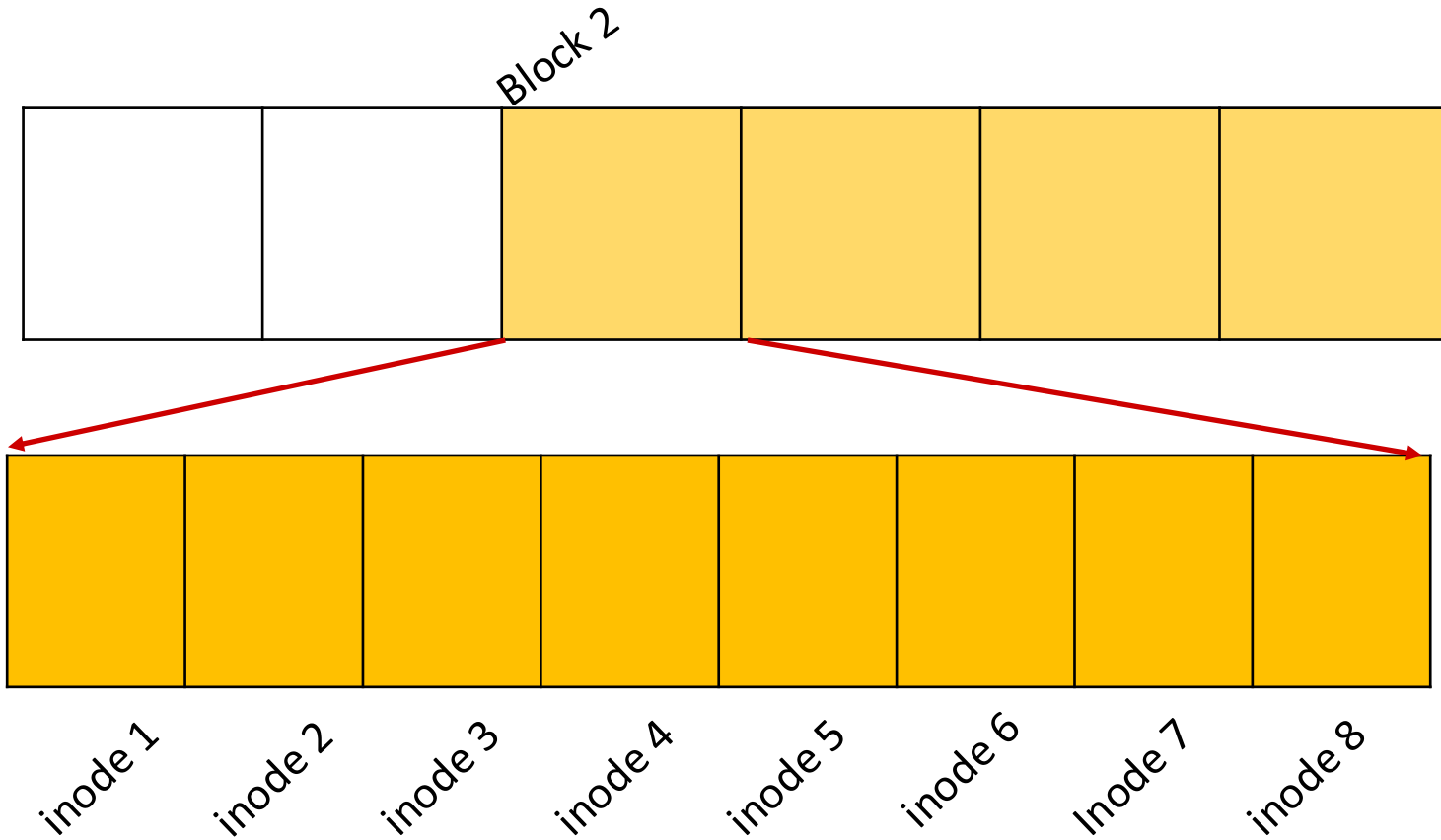
Poll Everywhere

pollev.com/cis5480

Which physical block contains inode 107?

First ask yourself:

How would I find inode 8?



inode 8 is in the first block! How would you know this?

(8 – 1) * sizeof(inode) / sizeof(block)

7 * 128 / 1024 = 0

This tells us, inode 8 is in the 0th block of the inode table (the first block).

Poll Everywhere

pollev.com/cis5480

Which physical block contains inode 107?

First ask yourself:

How would I find inode 8?

```
#define INODE_TABLE_BLOCK_START 2
```

How to find the physical block number that contains the inode?

```
block_number = INODE_TABLE_BLOCK_START + ( (inode_num – 1) * sizeof(inode) )/sizeof(block)

block_number = 2 + (8 – 1) * 128 / 1024

block_number = 2 + (7 * 128)/1024

block_number = 2 + 0

block_number = 2
```

Why is this zero?
Integer Division

Note: Need to do the multiplication first in software, integer division will make the quotient zero.

Poll Everywhere

pollev.com/cis5480

Which physical block contains inode 107?

First ask yourself:

What about inode 15?

```
#define INODE_TABLE_BLOCK_START 2
```

How to find the physical block number that contains the inode?

```
block_number = INODE_TABLE_BLOCK_START + ((inode_num – 1) * sizeof(inode))/sizeof(block)

block_number = 2 + ((15 – 1) * 128)/1024

block_number = 2 + (14 * 128) / 1024

block_number = 2 + 1
```

Why is this one?
Integer Division

`block_number = 3`

```
#define INODES_PER_BLOCK sizeof(BLOCK)/sizeof(inode)
```

You could also do: `INODE_TABLE_BLOCK_START + ((inode_num – 1) / INODES_PER_BLOCK)`

Poll Everywhere

pollev.com/cis5480

Which physical block contains inode 107?

First ask yourself:

What about inode 15?

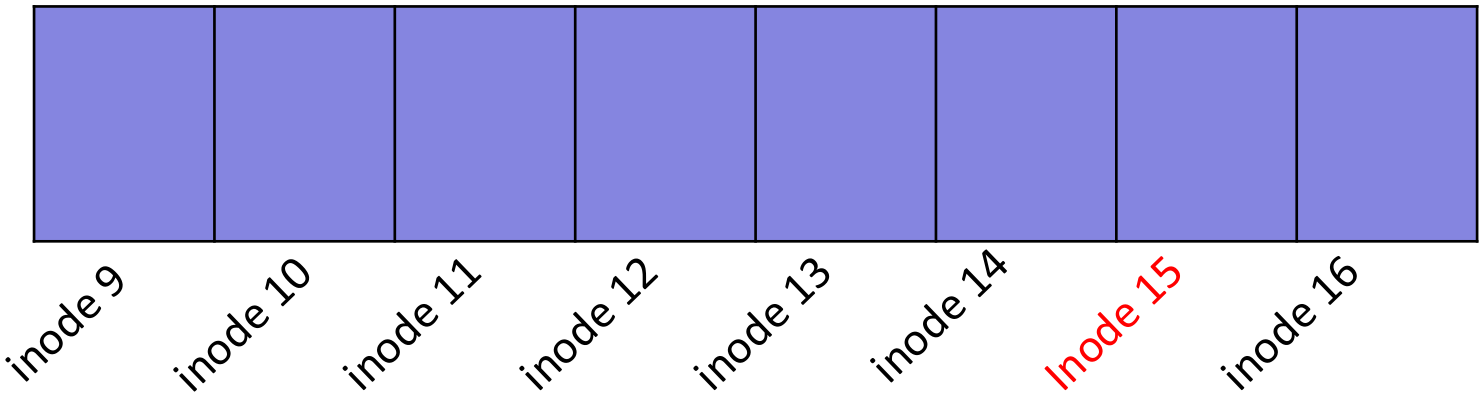
```
#define INODE_TABLE_BLOCK_START 2
```

How to find the physical block number that contains the inode?

```
block_number = INODE_TABLE_BLOCK_START + ((inode_num – 1) * sizeof(inode))/sizeof(block)
```

`block_number = 3`

The Second Block of the Inode Table (Physical Block 3)



Poll Everywhere

pollev.com/cis5480

Which physical block contains inode 107?

```
block_number = 2 + (inode_num – 1) * sizeof(inode) /sizeof(block)

block_number = 2 + ((107 – 1) * 128) / 1024

block_number = 2 + 13

block_number = 15
```

Poll Everywhere

pollev.com/cis5480

What is the index of the inode 107 relative to the block it is in?

First ask yourself:

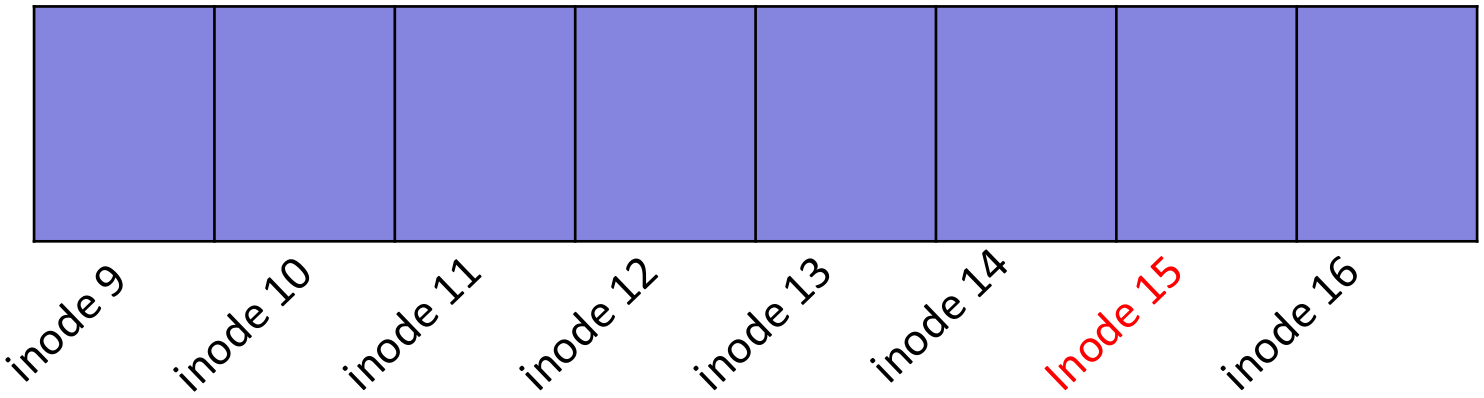
What about inode 15?

How would I be able to find the ***index*** of the inode relative to the physical block itself?

```
index = (inode_num - 1) % INODES_PER_BLOCK
```

```
index = (14) % 8 = 6
```

The Second Block of the Inode Table (Physical Block 3)




```
struct inode curr = inode_table_blocktwo[6];
```

psuedocode btw.

Poll Everywhere

pollev.com/cis5480

So, where is inode 107?

```
block_number = 2 + (inode_num – 1) * sizeof(inode) /sizeof(block)

block_number = 2 + ((107 – 1) * 128) / 1024

block_number = 2 + 13

block_number = 15

index = (inode_num – 1) % INODES_PER_BLOCK

index = (107 – 1) % 8

index = 106 % 8

index = 2
```

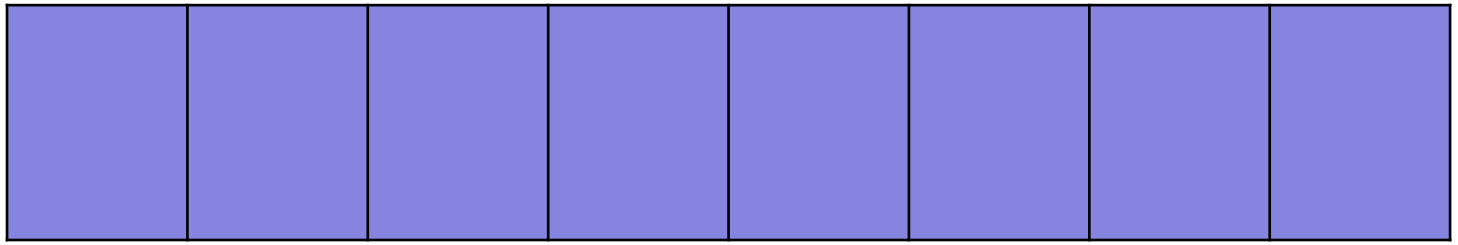
Poll Everywhere

pollev.com/cis5480

So, where is inode 107?

```
block_number = 15   index = 2
```

The InodeTableBlock[13] (Physical Block 15)



inode 105 inode 106 inode 107

Finally, we're done!

Poll Everywhere

pollev.com/cis5480

To open `/dir_one/dir_two/dir_three/file.txt`, what is the minimum number of inodes that must be accessed to fully resolve the path and retrieve the data blocks of file.txt? Assume you have not accessed the root inode.

Let's walk through this:

- The first inode we need is inode 2, the root directory
- Next, we need the inode for dir_one
- Next, we need the inode for dir_two
- Next, we need the inode for dir_three
- Next, we need the inode for file.txt
  - This final inode contains the block numbers with the data which contains the ***actual file.txt.***

So, in total, 5 inodes!

Poll Everywhere

pollev.com/cis5480

What is the minimum number of directory entries (struct dirents) that must be traversed to resolve the path `/dir_one/dir_two/dir_three/file.txt` and access the data blocks of file.txt in the best-case scenario? Assume you have not accessed the root inode.

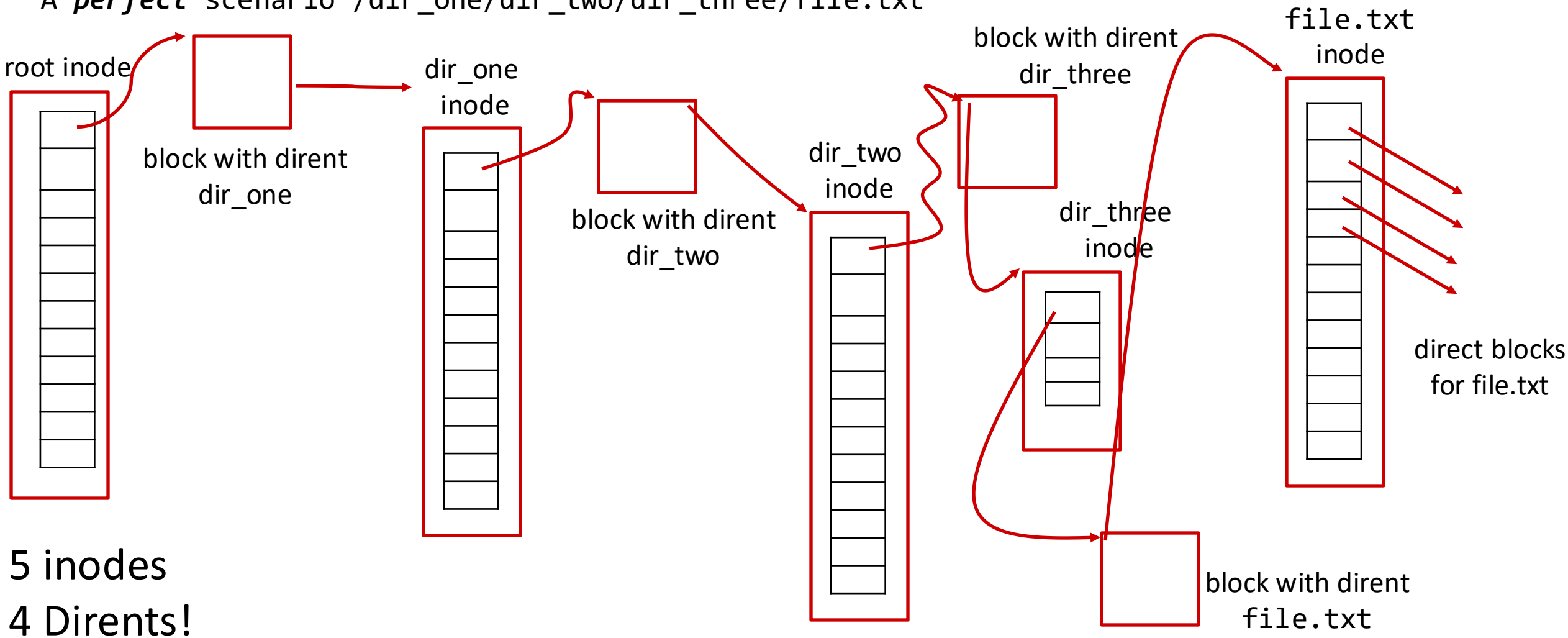
Let's walk through this best case scendario:

- The first `struct dirent` we encounter within the data blocks of the root inode is
  - the directory entry for dir_one
- The second `struct dirent` we encounter in the data blocks of the dir_one inode is
  - the directory entry for dir_two
- The third `struct dirent` we encounter in the data blocks of the dir_two inode is
  - the directory entry for dir_three
- The fourth `struct dirent` we encounter in the data blocks of the dir_three inode is
  - the directory entry for file.txt
- And we are done!

Poll Everywhere

pollev.com/cis5480

A ***perfect*** scenario /dir_one/dir_two/dir_three/file.txt



5 inodes

4 Dirents!