CS111, Lecture 4

Filesystem Design, Continued

Optional reading:

Operating Systems: Principles and Practice (2nd Edition): Sections 13.1-13.2

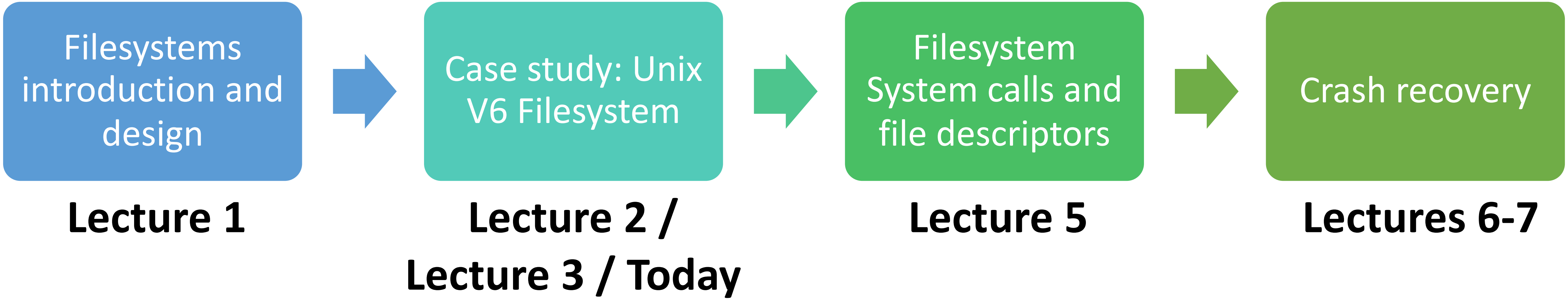
😷 masks strongly recommended

Announcements

- Sections start in person this week! Check the course website for your section assignment.
  - You can change your enrollment to any sections that have space available
  - If you are e.g. sick, you can attend another section as a guest that week, but please email the section TA to confirm there is space
  - If you have exceptional circumstances that prevent you from attending any section during a given week, please email the instructor
- Still working on room WiFi for lecture
- Assign0 due last night (Tues. night), late deadline Thurs.
- assign1 released! **YEAH Hours** (”Your Early Assignment Help”) announced soon, happening later this week via Zoom.

Topic 1: Filesystems **-** How can we design filesystems to manage files on disk, and what are the tradeoffs inherent in designing them? How can we interact with the filesystem in our programs?

CS111 Topic 1: Filesystems



Filesystems introduction and design

Lecture 1

Case study: Unix V6 Filesystem

Lecture 2 / Lecture 3 / Today

Filesystem System calls and file descriptors

Lecture 5

Crash recovery

Lectures 6-7

assign1: implement portions of the Unix v6 filesystem!

Learning Goals

- Explore the design of the Unix V6 filesystem
- Understand the design of the Unix v6 filesystem in how it represents directories
- Practice with the full process of going from file path to file data

Plan For Today

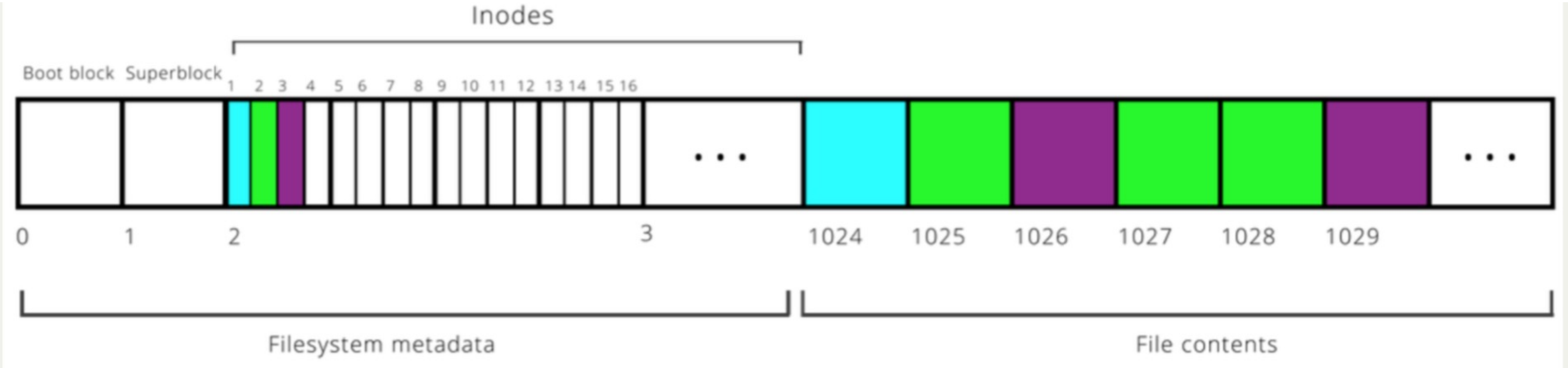
- **Recap**: the Unix V6 Filesystem so far
- **Practice:** doubly-indirect addressing
- Directories and filename lookup
- **Practice:** filename lookup
- Summary

Plan For Today

- **Recap: the Unix V6 Filesystem so far**
- **Practice:** doubly-indirect addressing
- Directories and filename lookup
- **Practice:** filename lookup
- Summary

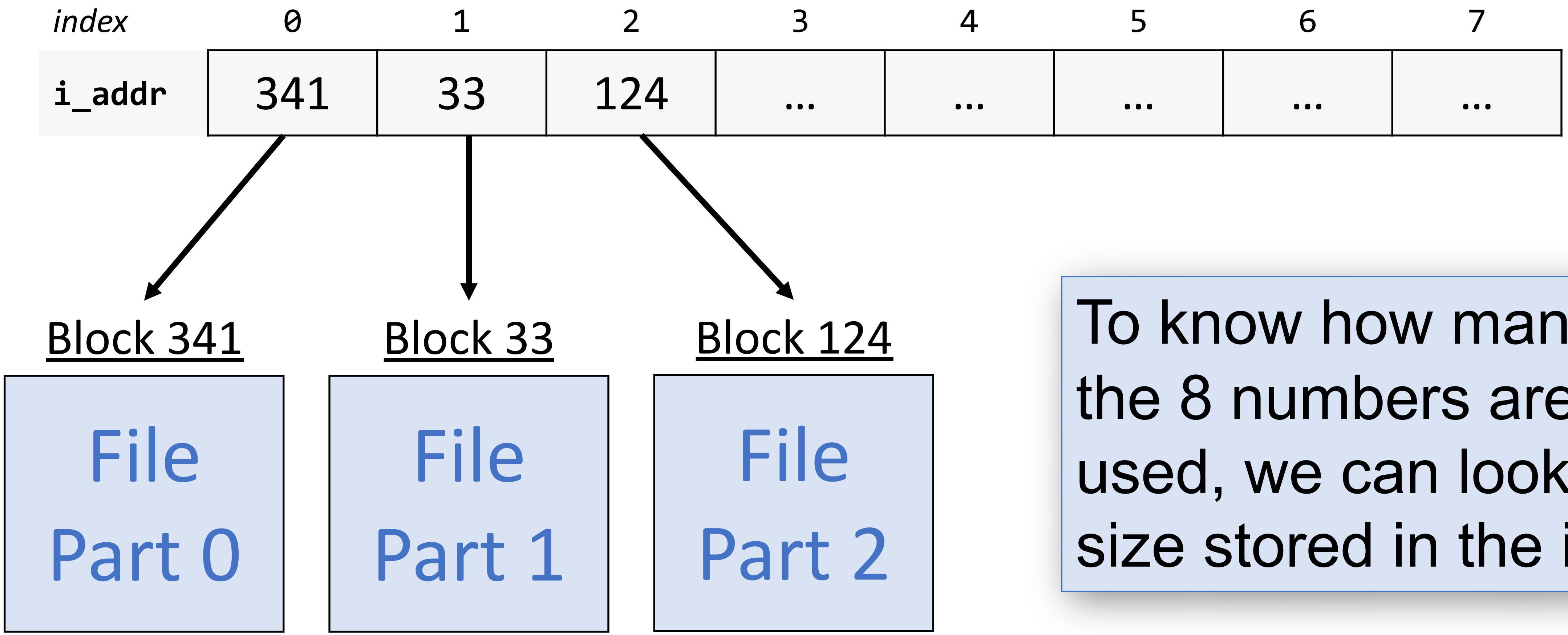
Unix V6 Filesystem

- Inodes are stored starting at sector 2 on disk, and are numbered starting at 1
- There is 1 inode for each file
- An inode has space for up to 8 block numbers
- Those block numbers are used differently depending on whether the file is “small” or “large”
- **`if ((inode.i_mode & ILARG) != 0) { // file is “large”`**



Small File Scheme

If the file is small, **i_addr** stores numbers of blocks that contain payload data.

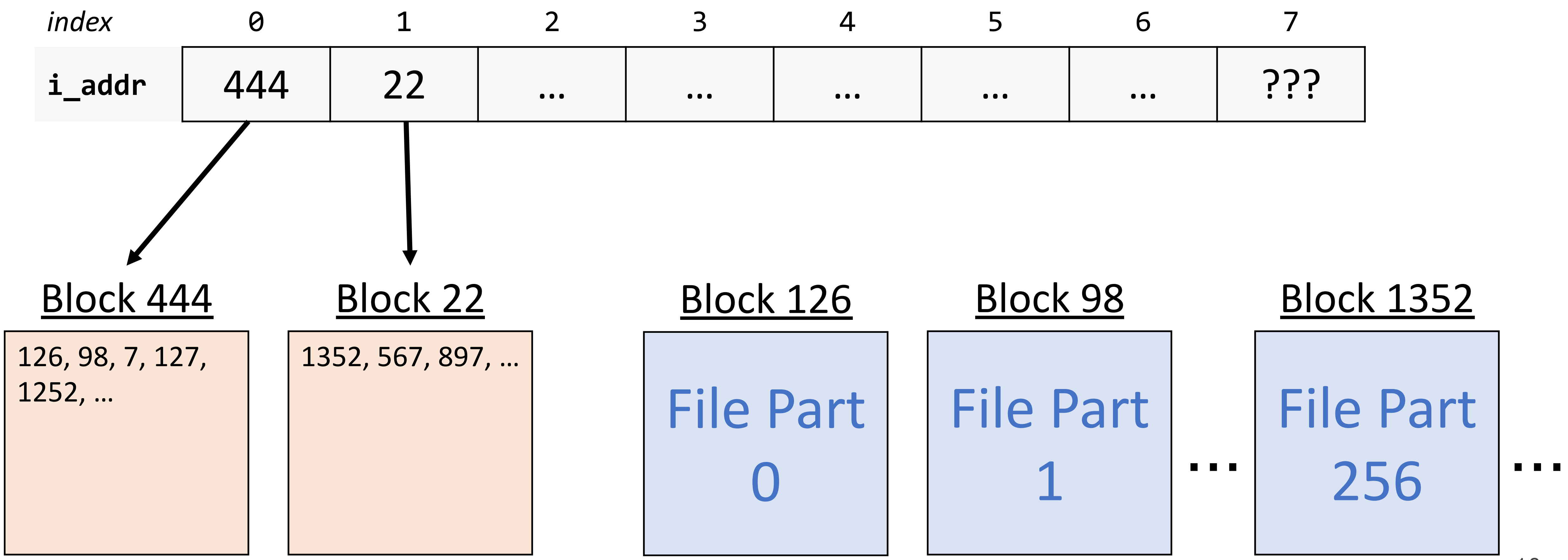


index	0	1	2	3	4	5	6	7
i_addr	341	33	124	…	…	…	…	…

To know how many of the 8 numbers are used, we can look at the size stored in the inode.

Large File Scheme

If the file is large, **i_addr** stores 7 numbers of blocks that contain block numbers, and those block numbers are of blocks that contain payload data.



index	0	1	2	3	4	5	6	7
i_addr	444	22	…	…	…	…	…	???

Even Larger Files

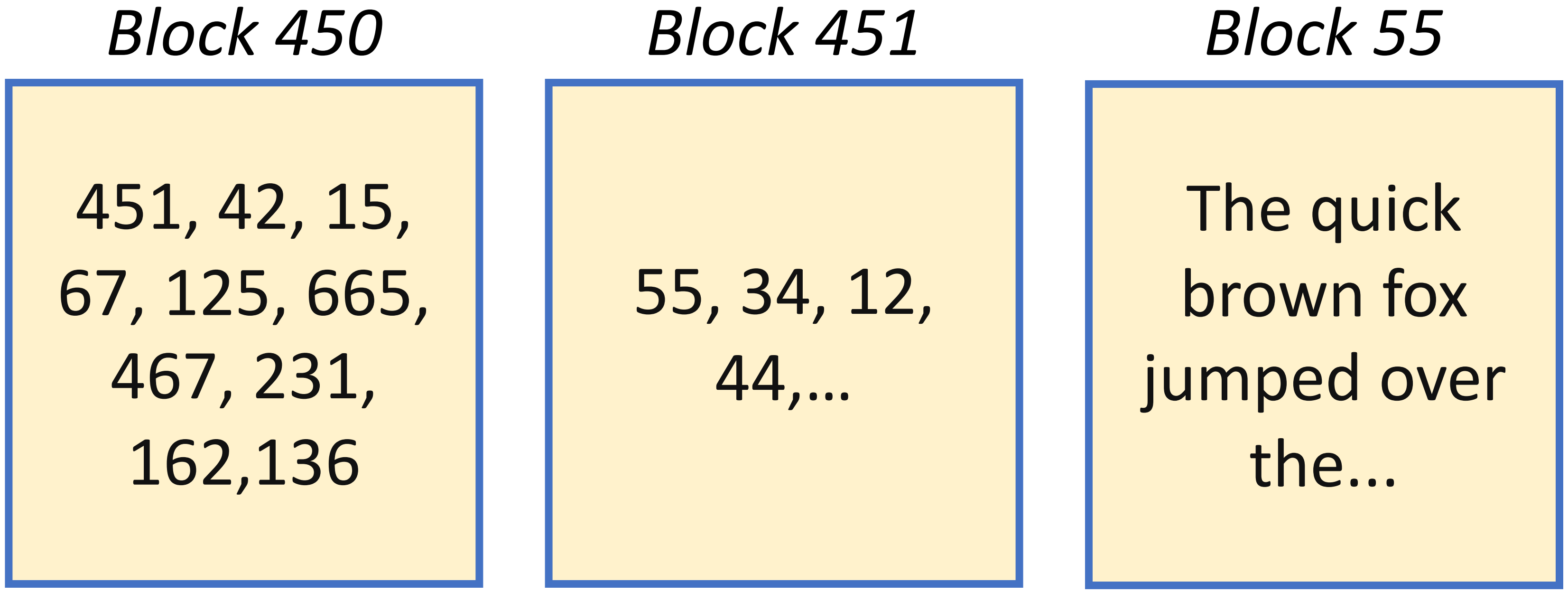
Problem: even with singly-indirect addressing, the largest a file can be is 8 * 256 * 512 = 1,048,576 bytes (~1MB). That still isn't realistic!

Solution: let's use *doubly-indirect addressing*; store a block number for a block that contains *singly-indirect block numbers*.

Even Larger Files

Solution: let's use *doubly-indirect addressing*; store a block number for a block that contains *singly-indirect block numbers*.

Allows even larger files, but data takes even more steps to access. How do we employ this idea?



Block 450

451, 42, 15, 67, 125, 665, 467, 231, 162,136

Block 451

55, 34, 12, 44,…

Block 55

The quick brown fox jumped over the...

Indirect Addressing

The Unix V6 filesystem uses *indirect addressing* (blocks that store payload block numbers) just for large files.

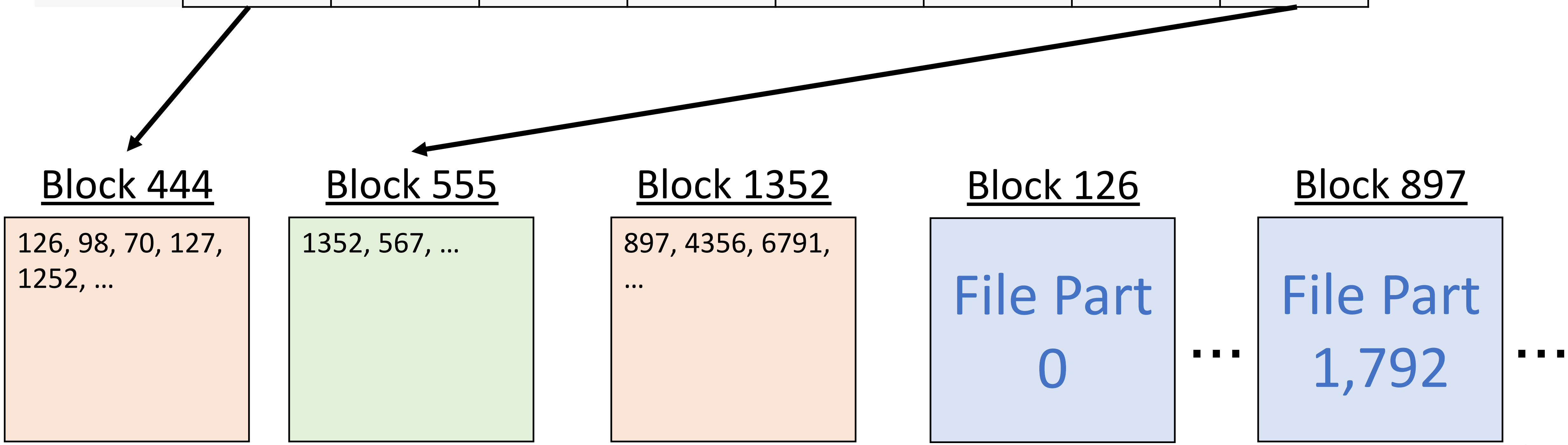
- If small, each block number in the inode stores payload data
- If large, **first 7 block numbers** are singly-indirect
- NEW: If large (and if needed), **8th block number** is doubly-indirect (it refers to a block that stores singly-indirect block numbers)
- Files only use the block numbers they need (depending on their size)

In other words; a file can be represented using at most 256 + 7 = 263 singly-indirect blocks. The first seven are stored in the inode. The remaining 256 are stored in a block whose block number is stored in the inode.

Large File Scheme

If the file is large, **i_addr** stores 7 numbers of blocks that contain block numbers, and those block numbers are of blocks that contain payload data.

index	0	1	2	3	4	5	6	7
i_addr	444	22	34	792	168	976	2467	555



Block 444

126, 98, 70, 127, 1252, …

Block 555

1352, 567, …

Block 1352

897, 4356, 6791, …

Block 126

File Part 0

…

Block 897

File Part 1,792

…

Indirect Addressing

An inode for a large file stores 7 singly-indirect block numbers and 1 doubly-indirect block number. What is the largest file size this supports? Each block number is 2 bytes big.

(7+256) singly-indirect block numbers total x

256 block numbers per singly-indirect block x

512 bytes per block

= ~34MB

Indirect Addressing

An inode for a large file stores 7 singly-indirect block numbers and 1 doubly-indirect block number. What is the largest file size this supports? Each block number is 2 bytes big.

OR:

*(7 * 256 * 512) + (256 * 256 * 512) ~ 34MB*

(singly indirect) + (doubly indirect)

Better! still not sufficient for today's standards, but perhaps in 1975. Moreover, since block numbers are 2 bytes, we can number at most 2^16 - 1 = 65,535 blocks, meaning the entire filesystem can be at most 65,535 * 512 ~ 32MB.

Plan For Today

- **Recap**: the Unix V6 Filesystem so far
- **Practice: doubly-indirect addressing**
- Directories and filename lookup
- **Practice:** filename lookup
- Summary

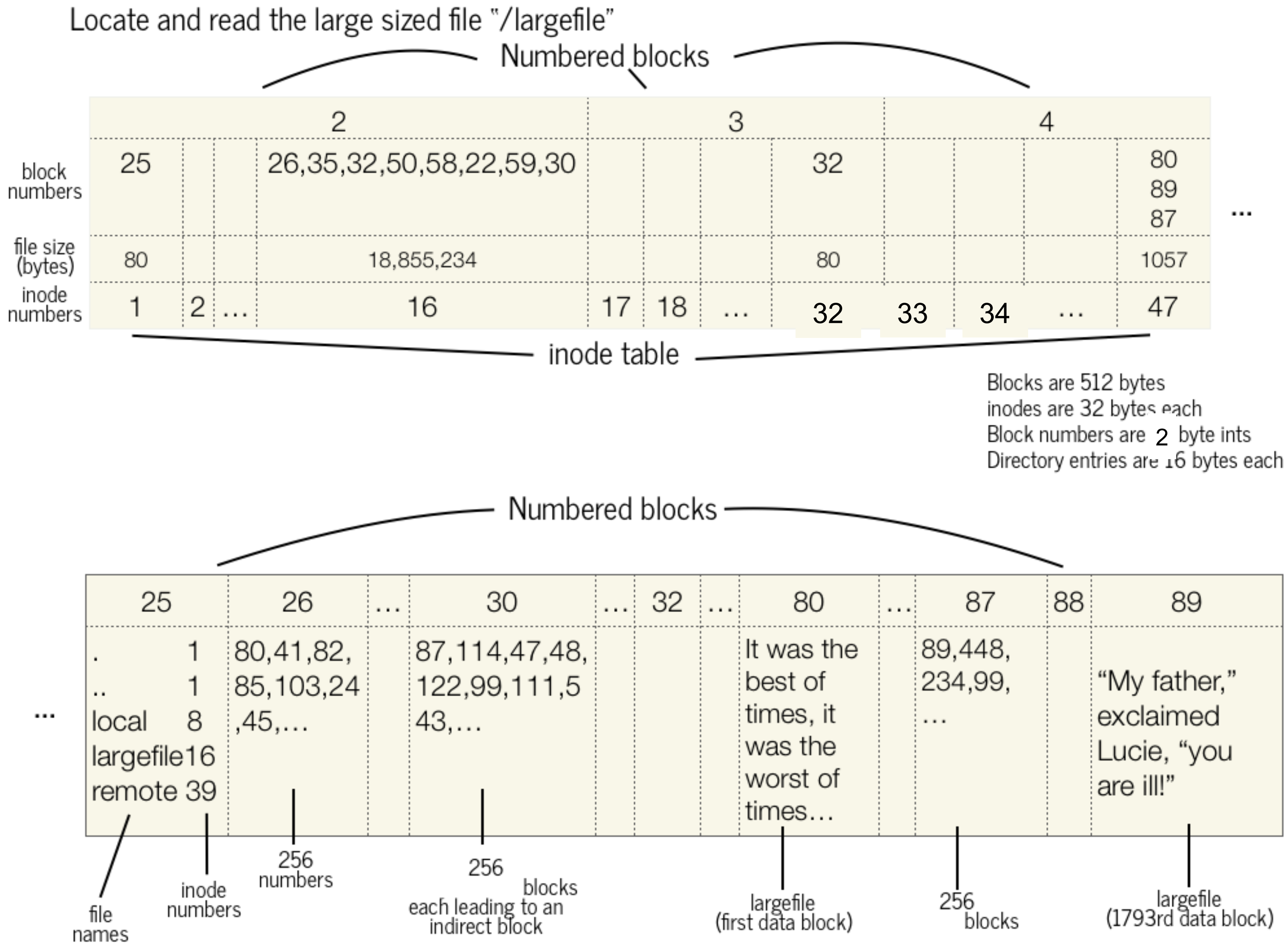
Doubly-Indirect Addressing

What is the smallest file size (in bytes) that would require using the doubly-indirect block to store its data?

Files up to (7 * 256 * 512) bytes are representable using just the 7 singly-indirect blocks. Files of (7 * 256 * 512) + 1 or more bytes would need the doubly-indirect block as well.

Doubly-Indirect Addressing

Assume we have a large file with inumber 16. How do we find the block containing the start of its payload data? How about the remainder of its payload data?



Locate and read the large sized file "/largefile"

Numbered blocks

	2				3				4				
block numbers	25			26,35,32,50,58,22,59,30				32				80 89 87	…
file size (bytes)	80			18,855,234				80				1057	
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47	

inode table

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

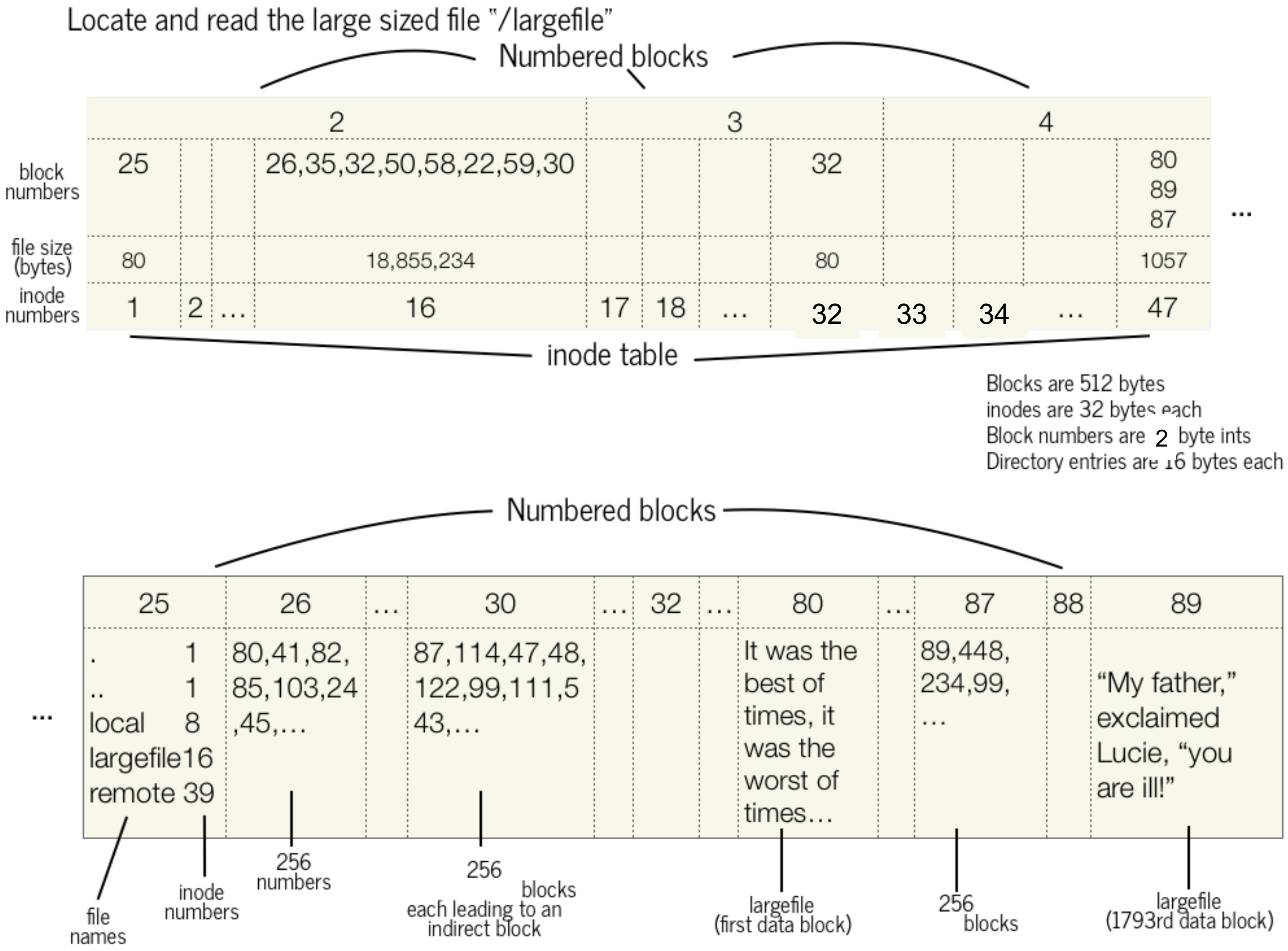
Numbered blocks

	25	26	…	30	…	32	…	80	…	87	88	89
…	. 1 .. 1 local 8 largefile 16 remote 39	80,41,82, 85,103,24 ,45,…		87,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		89,448, 234,99, …		"My father," exclaimed Lucie, "you are ill!"

file names
inode numbers
256 numbers
256 blocks each leading to an indirect block
largefile (first data block)
256 blocks
largefile (1793rd data block)

Doubly-Indirect Addressing

1. Go to block 26 and start reading block numbers. For the first number, 80, go to block 80 and read the beginning of the file (the first 512 bytes). Then go to block 41 for the next 512 bytes, etc.



Locate and read the large sized file "/largefile"

	2				3				4				
block numbers	25			26,35,32,50,58,22,59,30				32				80 89 87	…
file size (bytes)	80			18,855,234				80				1057	
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47	

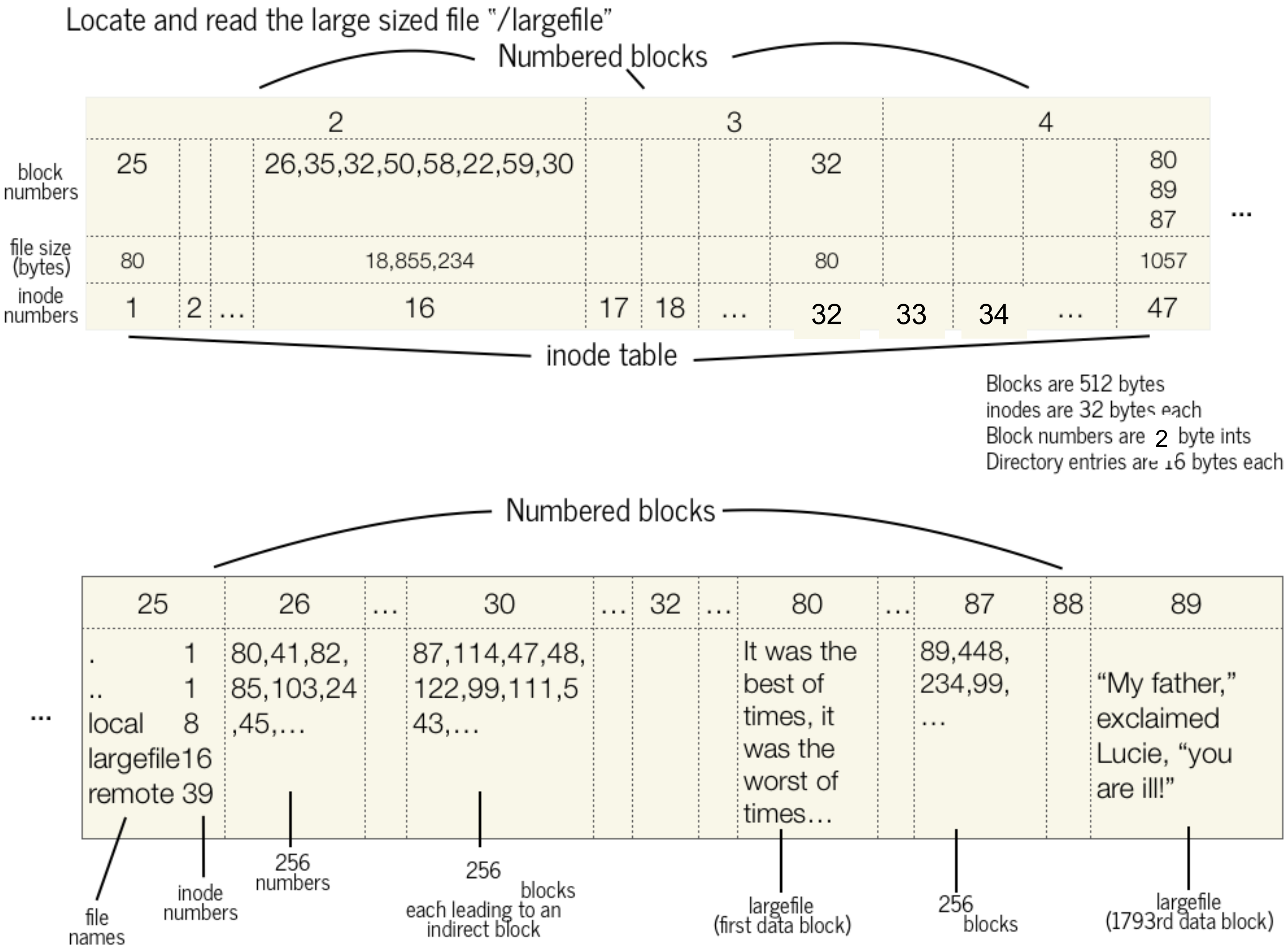
Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

	25	26	…	30	…	32	…	80	…	87	88	89
…	. 1 .. 1 local 8 largefile16 remote 39	80,41,82, 85,103,24 ,45,…		87,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		89,448, 234,99, …		"My father," exclaimed Lucie, "you are ill!"

file names
inode numbers
256 numbers
256 blocks each leading to an indirect block
largefile (first data block)
256 blocks
largefile (1793rd data block)

Doubly-Indirect Addressing

2. After 256 blocks, go to block 35, repeat the process. Do this a total of 7 times, for blocks 26, 35, 32, 50, 58, 22, and 59, reading 1792 blocks.



Locate and read the large sized file "/largefile"

	2				3				4			
block numbers	25			26,35,32,50,58,22,59,30				32				80 89 87
file size (bytes)	80			18,855,234				80				1057
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47

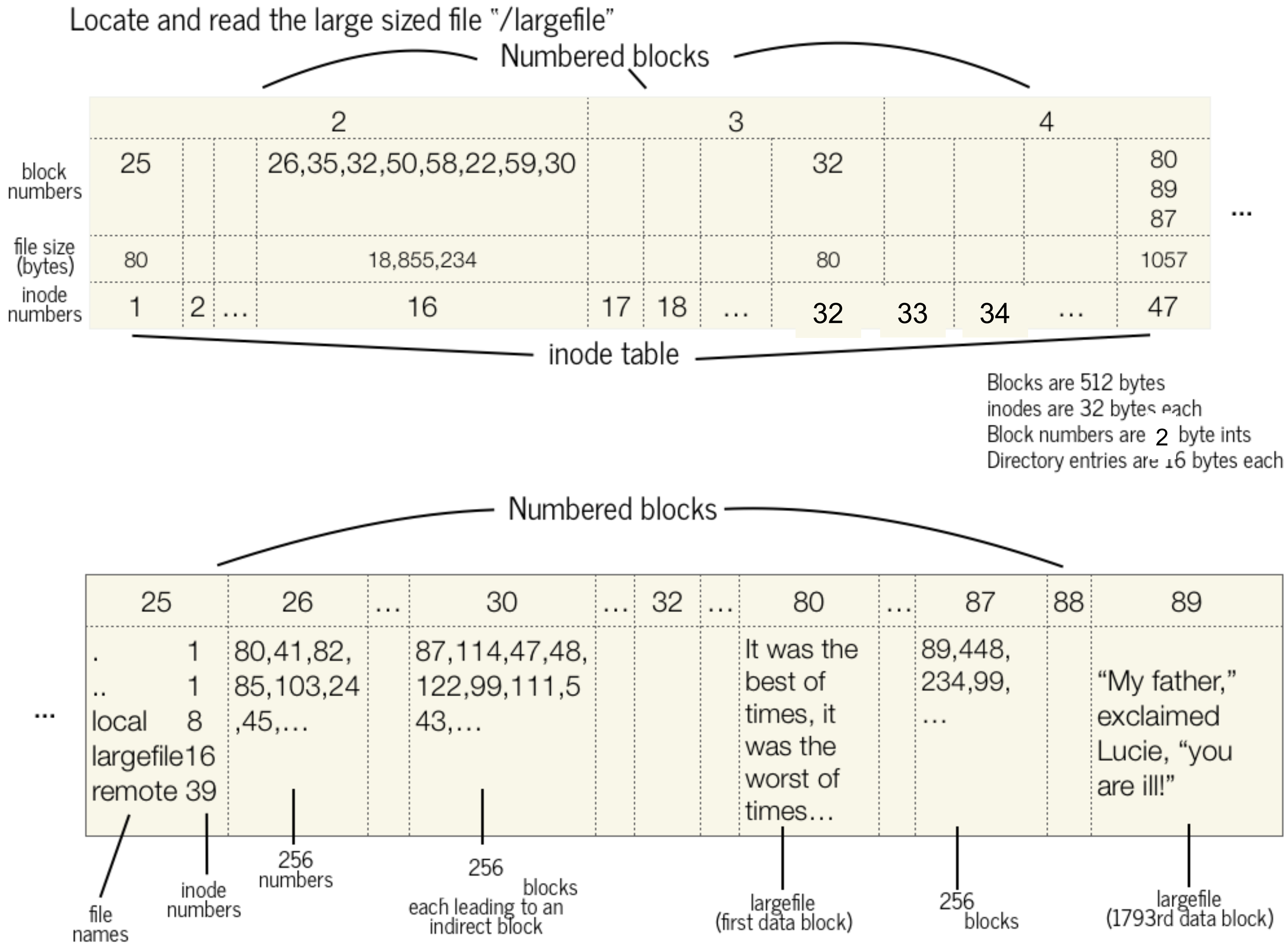
Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

25	26	…	30	…	32	…	80	…	87	88	89
. 1 .. 1 local 8 largefile16 remote 39	80,41,82, 85,103,24 ,45,…		87,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		89,448, 234,99, …		"My father," exclaimed Lucie, "you are ill!"

file names
inode numbers
256 numbers
256 blocks each leading to an indirect block
largefile (first data block)
256 blocks
largefile (1793rd data block)

Doubly-Indirect Addressing

3. Go to block 30, which is a doubly-indirect block. From there, go to block 87, which is an indirect block. From there, go to block 89, which is the 1793rd (256*7 + 1) block.



Plan For Today

- **Recap**: the Unix V6 Filesystem so far
- **Practice:** doubly-indirect addressing
- **Directories and filename lookup**
- **Practice:** filename lookup
- Summary

Now we understand how files are *stored*. But how do we *find* them?

The Directory Hierarchy

Filesystems usually support directories ("folders")

- A directory can contain files and more directories
- A directory is a file container. It needs to store what files/folders are contained within it. It also has associated metadata.
- On Unix/Linux, all files live within the root directory, "/"
- We can specify the location of a file via the path to it from the root directory:

```
/classes/cs111/index.html
```

Common filesystem task: given a filepath, get the file's contents.

Key Idea: let's model a directory *as a file*. We have already designed support for storing payloads and metadata. Why not use it?

Directories as Files

A directory is a file container. It needs to store what files/folders are contained within it. It also has associated metadata.

- Have an inode for each directory
- A directory's "payload data" is a list of info about the files it contains
- A directory's "metadata" is information about it such as its owner
- Inodes can store a field telling us whether something is a directory or file

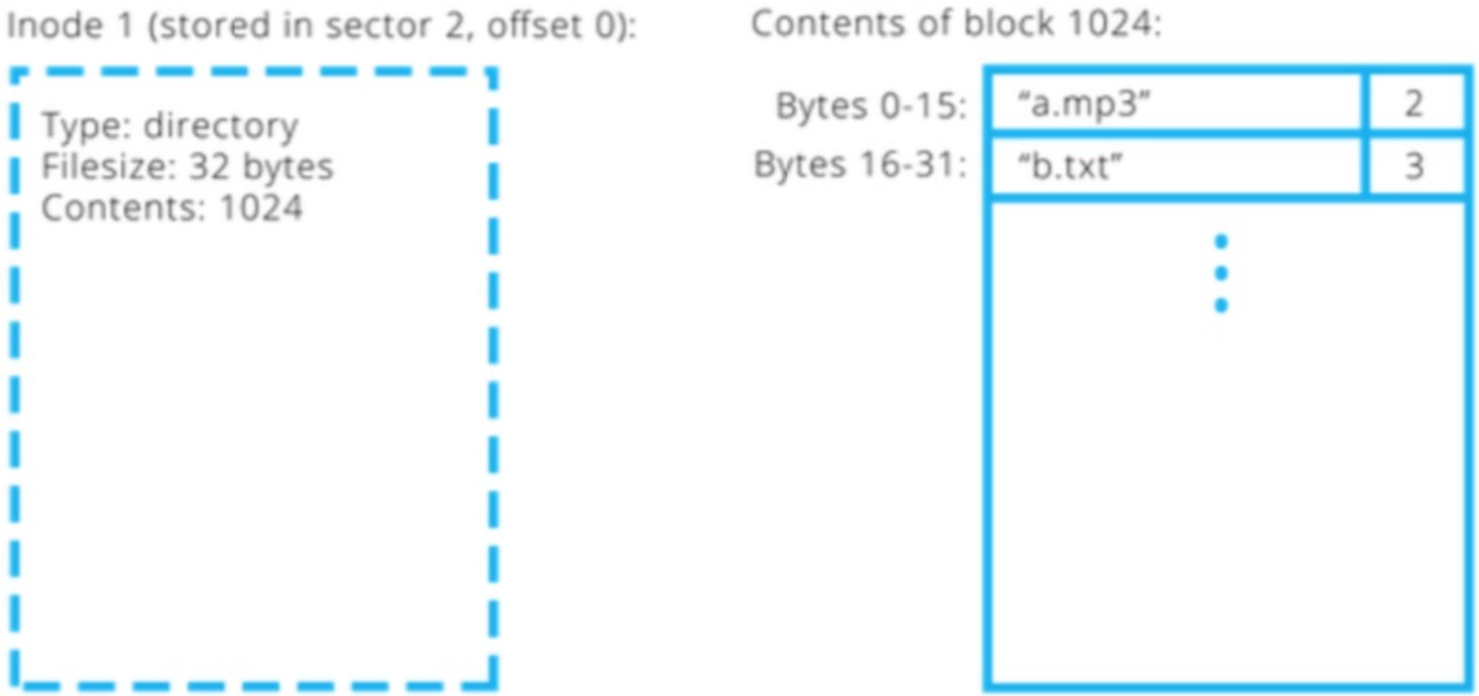
We can layer support for directories right on top of our implementation for files!

Representing Directories

Design decision: the Unix V6 filesystem makes directory payloads contain a 16 byte entry for each file/folder that is in that directory, in no particular order.

- The first 14 bytes are the name (not necessarily null-terminated!)
- The last two bytes are the inumber

```
struct direntv6 {
  uint16_t d_inumber;
  char     d_name[14];
};
```



Inode 1 (stored in sector 2, offset 0):

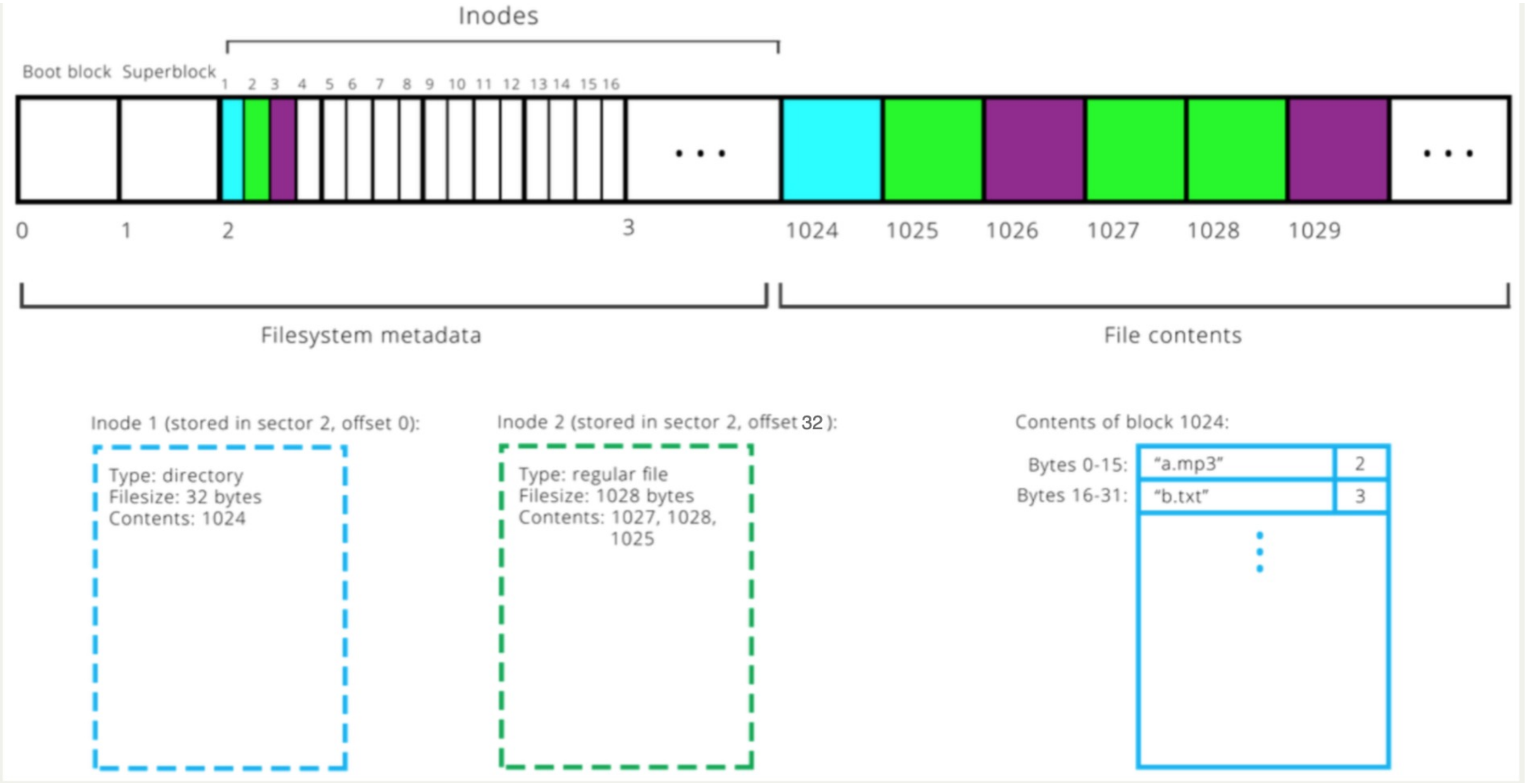
Type: directory
Filesize: 32 bytes
Contents: 1024

Contents of block 1024:

Bytes 0-15:	“a.mp3”	2
Bytes 16-31:	“b.txt”	3

Representing Directories

Given the inode for a directory, **how could we find the inumber for a file it contains called "b.txt"?**



Inode 1 (stored in sector 2, offset 0):

Type: directory
Filesize: 32 bytes
Contents: 1024

Inode 2 (stored in sector 2, offset 32):

Type: regular file
Filesize: 1028 bytes
Contents: 1027, 1028, 1025

Contents of block 1024:

Bytes 0-15:	"a.mp3"	2
Bytes 16-31:	"b.txt"	3
	⋮	

What about translating from a filepath to an inumber? How does that work?

The Lookup Process

```
/classes/cs111/index.html
```

Start at the root directory

The Lookup Process

```
/classes/cs111/index.html
```

In the root directory, find the entry named "classes".

The Lookup Process

```
/classes/cs111/index.html
```

In the "classes" directory, find the entry named "cs111".

The Lookup Process

```
/classes/cs111/index.html
```

In the "cs111" directory, find the entry named "index.html". Then read its contents.

The Lookup Process

The root directory ("/") is set to have inumber 1. That way we always know where to go to start traversing. (0 is reserved to mean "NULL" or "no inode").

The Lookup Process

```
/classes/cs111/index.html
```

Go to inode with inumber 1 (root directory).

The Lookup Process

```
/classes/cs111/index.html
```

In its payload data, look for the entry "classes" and get its inumber. Go to that inode.

The Lookup Process

```
/classes/cs111/index.html
```

In its payload data, look for the entry "cs111" and get its inumber. Go to that inode.

The Lookup Process

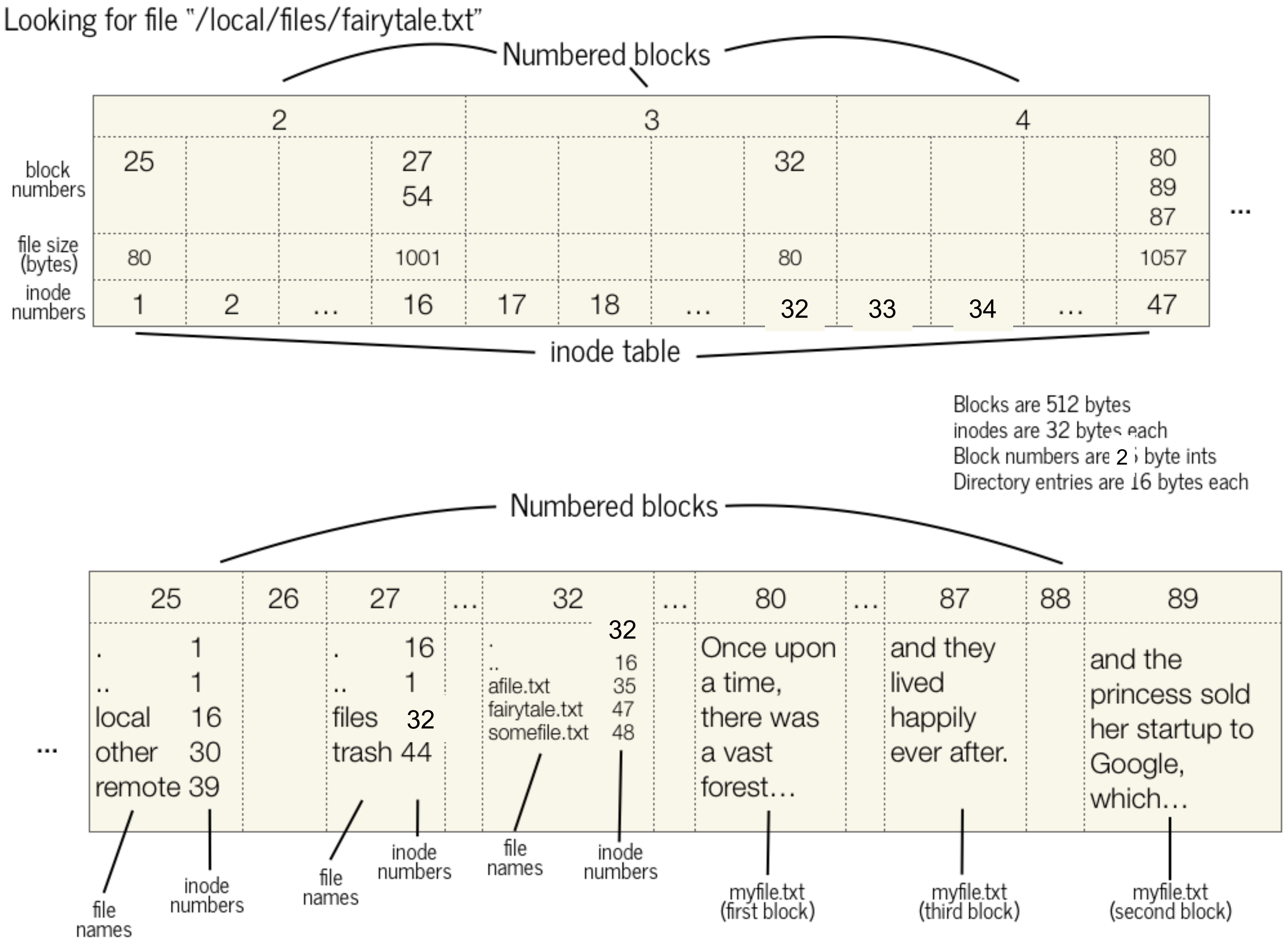
```
/classes/cs111/index.html
```

In its payload data, look for the entry "index.html" and get its inumber. Go to that inode and read in its payload data.

Plan For Today

- Recap: the Unix V6 Filesystem so far
- Practice: doubly-indirect addressing
- Directories and filename lookup
- **Practice: filename lookup**
- Summary

Ex.: Finding "/local/files/fairytale.txt" (small file)



Looking for file "/local/files/fairytale.txt"

	2				3				4			
block numbers	25			27 54				32				80 89 87
file size (bytes)	80			1001				80				1057
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47

…

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

25	26	27	…	32	…	80	…	87	88	89
. 1 .. 1 local 16 other 30 remote 39		. 16 .. 1 files 32 trash 44		. 32 .. 16 afile.txt 35 fairytale.txt 47 somefile.txt 48		Once upon a time, there was a vast forest…		and they lived happily ever after.		and the princess sold her startup to Google, which…

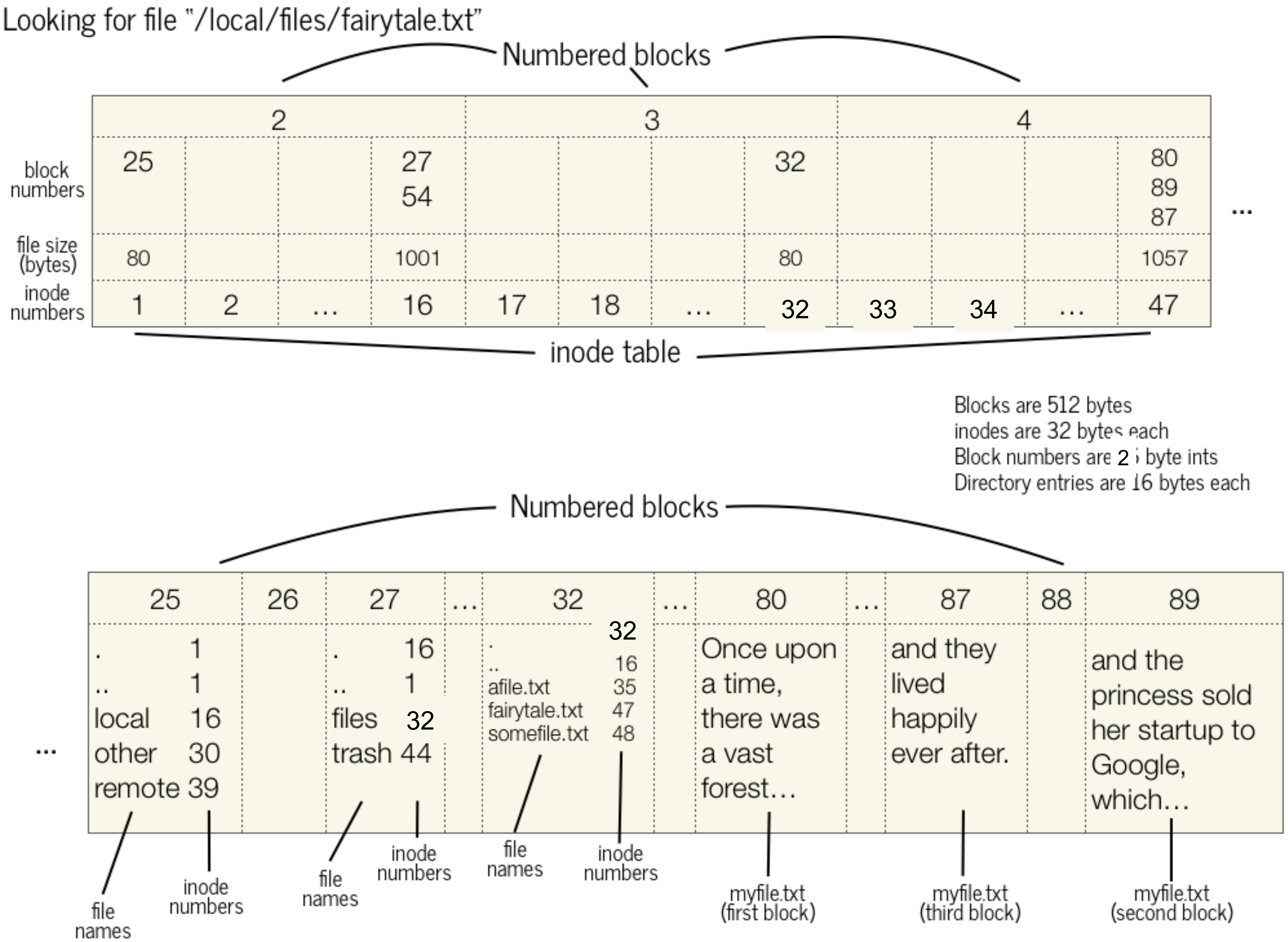
…

1. go to inode 1. It's small. We need to look in block 25 for the list of its entries.

2. Look in block 25 for "local" -> inode 16.

3. Go to inode 16. It's small. We need to look in blocks 27/54 for the list of its entries.

Ex.: Finding "/local/files/fairytale.txt" (small file)

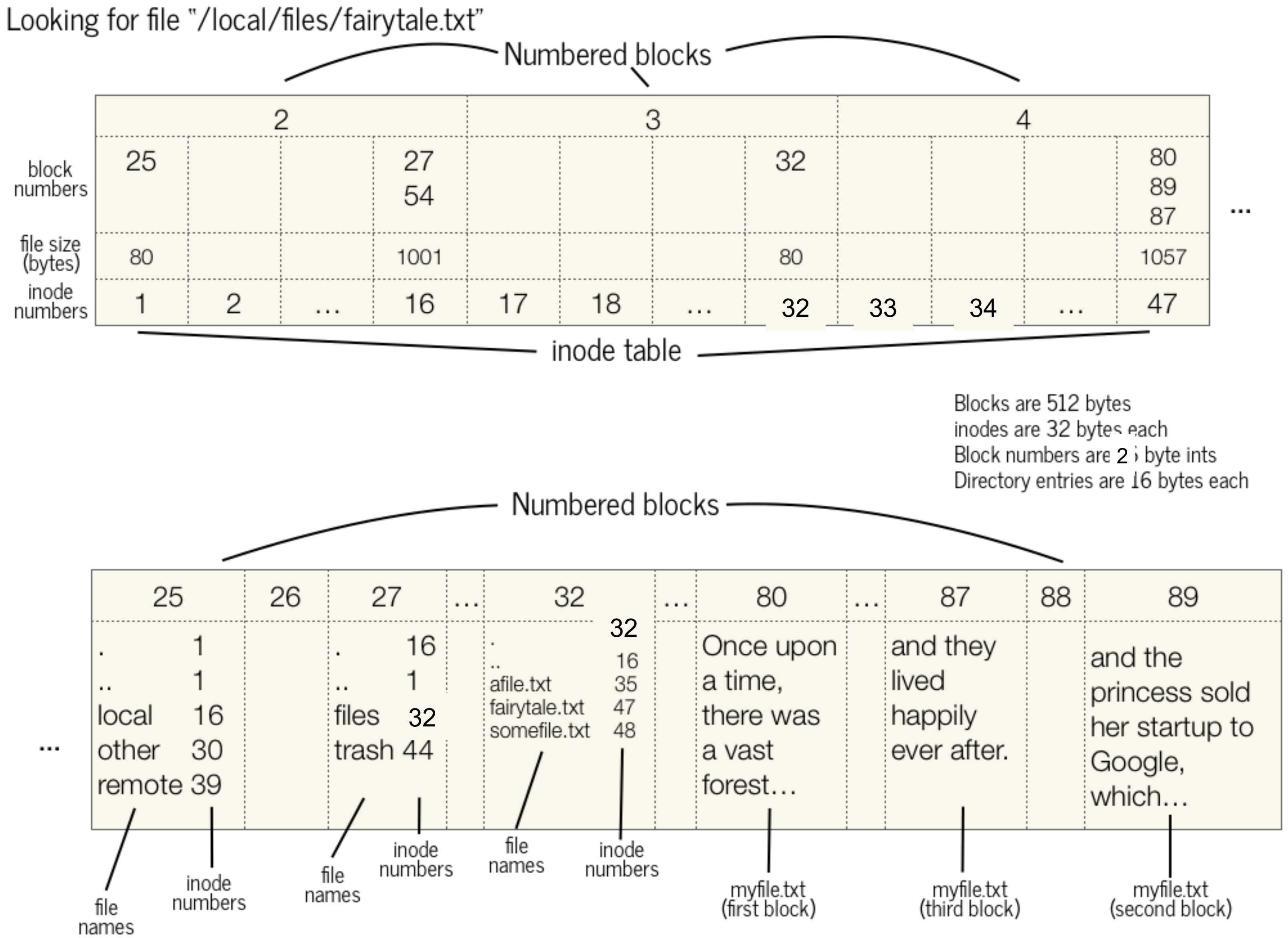


4. Look in block 27 for "files" (and then 54 if necessary) -> inode 32.

5. Go to inode 32. It's small. We need to look in block 32 for the list of its entries.

6. Look in block 32 for "fairytale.txt" -> inode 47.

Ex.: Finding "/local/files/fairytale.txt" (small file)



Looking for file "/local/files/fairytale.txt"

	2				3				4				
block numbers	25			27 54				32				80 89 87	...
file size (bytes)	80			1001				80				1057	
inode numbers	1	2	...	16	17	18	...	32	33	34	...	47	

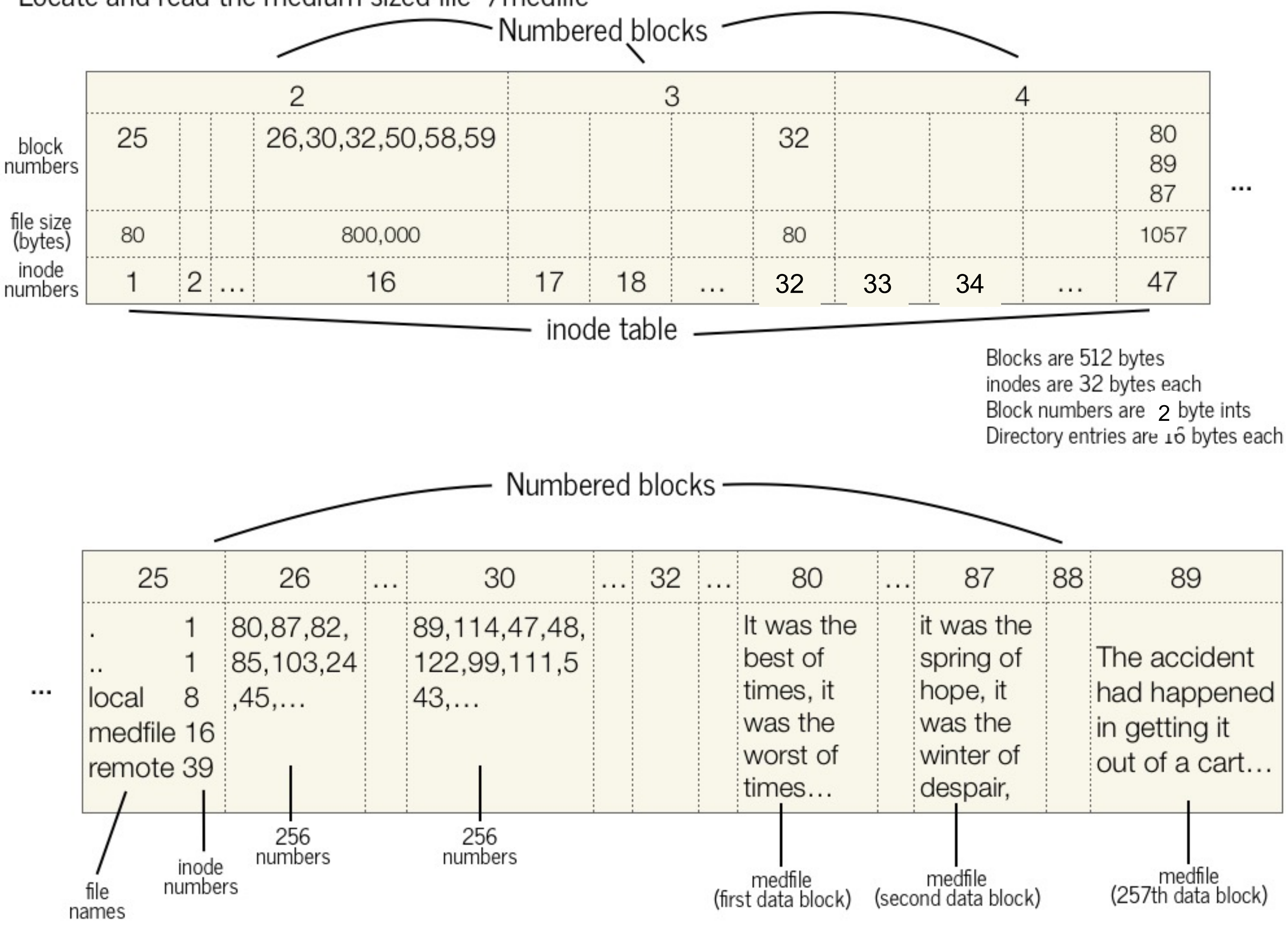
Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

	25	26	27	...	32	...	80	...	87	88	89
...	. 1 .. 1 local 16 other 30 remote 39		. 16 .. 1 files 32 trash 44		. 32 .. 16 afile.txt 35 fairytale.txt 47 somefile.txt 48		Once upon a time, there was a vast forest…		and they lived happily ever after.		and the princess sold her startup to Google, which…

7. go to inode 47. It's small. We need to look in blocks 80,89,87 in order for its 1,057 bytes of payload data.

Ex.: Finding "/medfile" (large file)

Locate and read the medium sized file "/medfile"



	2				3				4			
block numbers	25			26,30,32,50,58,59				32				80 89 87
file size (bytes)	80			800,000				80				1057
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

Numbered blocks
…
file names
inode numbers
256 numbers
256 numbers
medfile (first data block)
medfile (second data block)
medfile (257th data block)

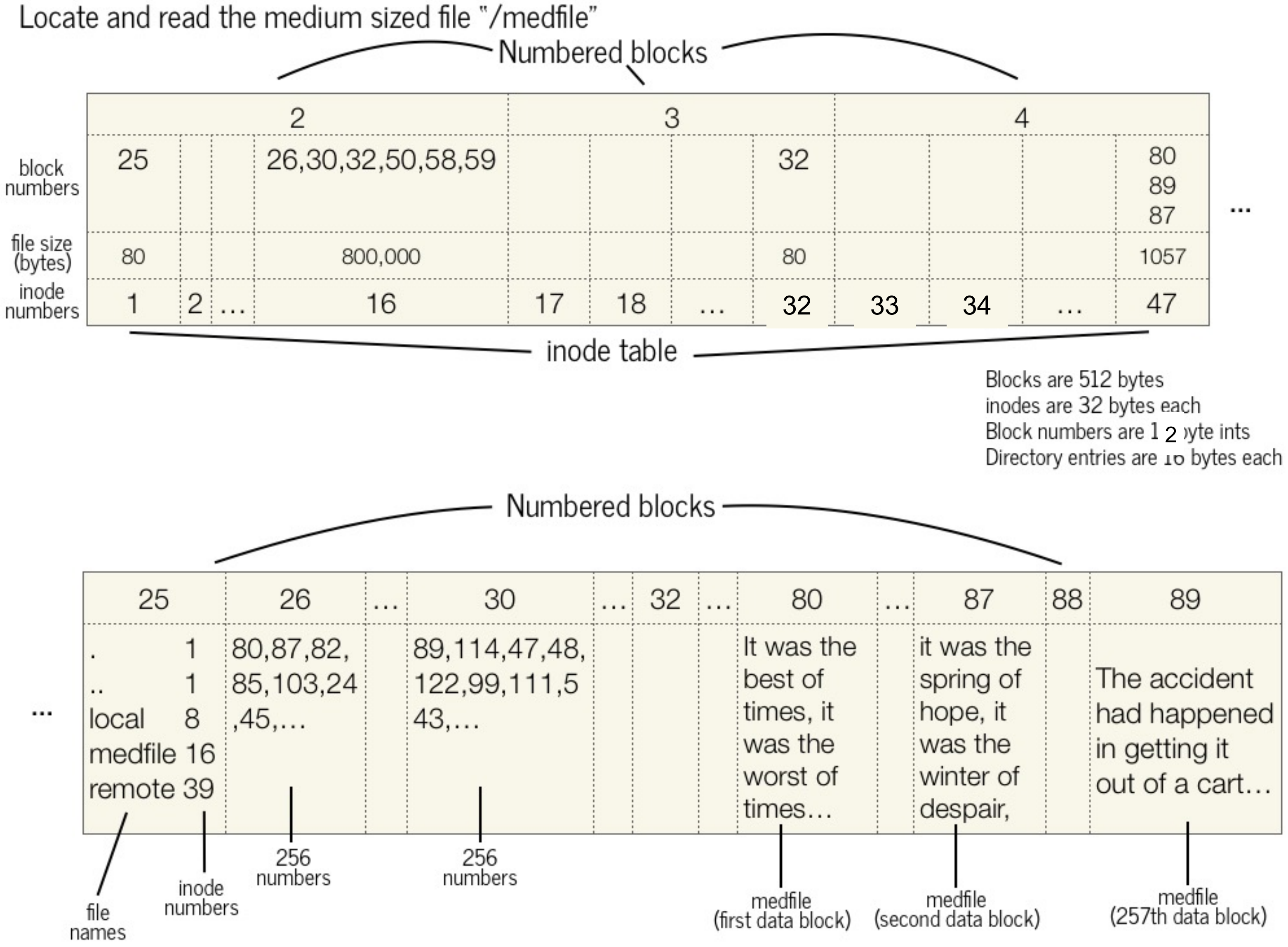
25	26	…	30	…	32	…	80	…	87	88	89
. 1 .. 1 local 8 medfile 16 remote 39	80,87,82, 85,103,24 ,45,…		89,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		it was the spring of hope, it was the winter of despair,		The accident had happened in getting it out of a cart…

1. go to inode 1. It's small. We need to look in block 25 for the list of its entries.

2. Look in block 25 for "medfile" -> inode 16.

3. Go to inode 16. It's large. We need to look in block 26 for the first 256 payload block numbers.

Ex.: Finding "/medfile" (large file)



Locate and read the medium sized file "/medfile"

	2				3				4			
block numbers	25			26,30,32,50,58,59				32				80 89 87
file size (bytes)	80			800,000				80				1057
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 1 2 byte ints
Directory entries are 16 bytes each

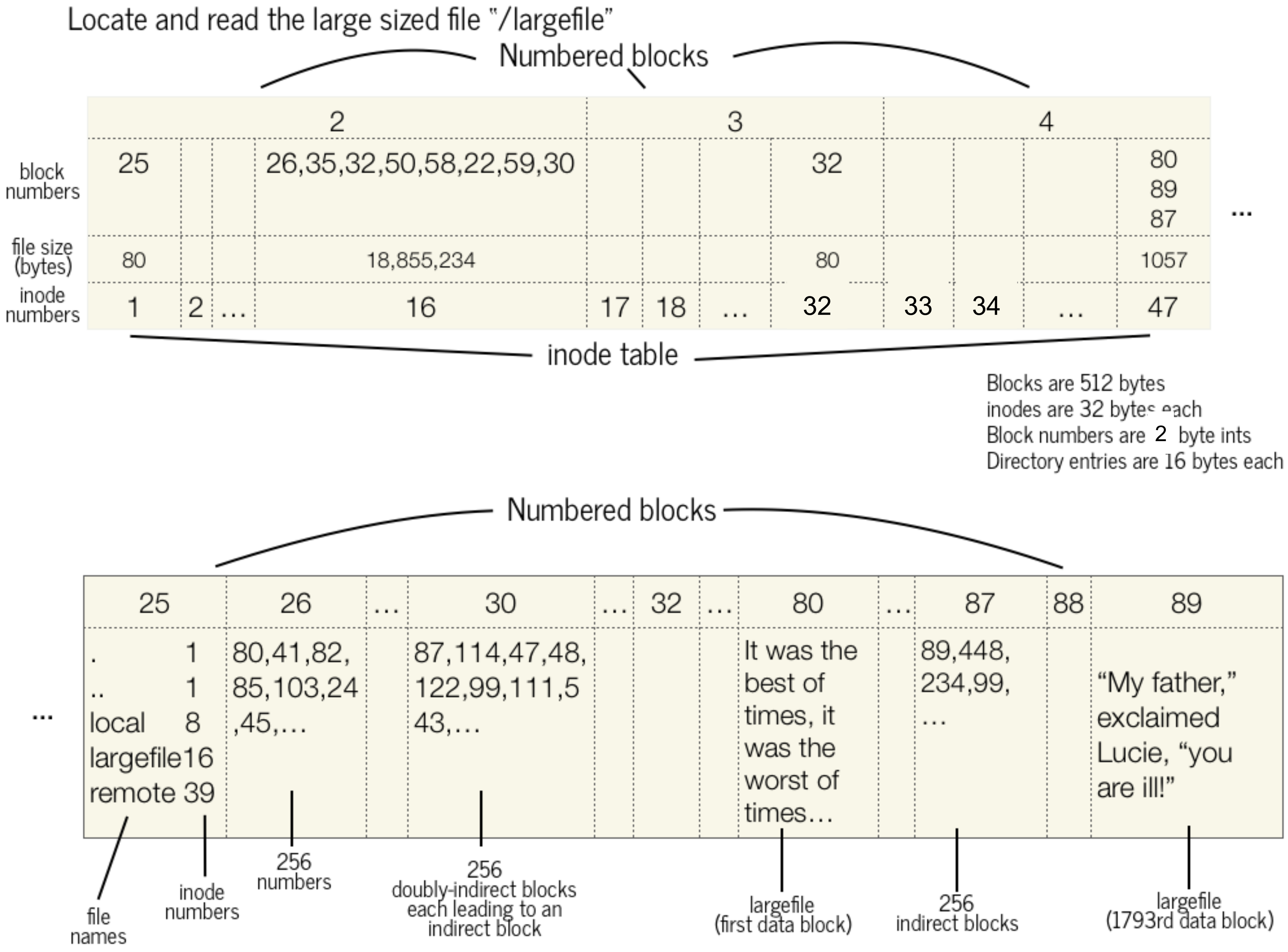
25	26	…	30	…	32	…	80	…	87	88	89
. 1 .. 1 local 8 medfile 16 remote 39	80,87,82, 85,103,24 ,45,…		89,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		it was the spring of hope, it was the winter of despair,		The accident had happened in getting it out of a cart…

file names
inode numbers
256 numbers
256 numbers
medfile (first data block)
medfile (second data block)
medfile (257th data block)

4. Read through numbers in block 26. First, go to block 80 for the first 512 payload bytes. Then, go to block 87 for the second 512 payload bytes.

5. After doing this 256 times, go to block 30 and repeat. Then continue with all remaining singly-indirect blocks in the inode.

Ex.: Finding “/largefile” (large file)



Locate and read the large sized file "/largefile"

Numbered blocks

	2				3				4				
block numbers	25			26,35,32,50,58,22,59,30				32				80 89 87	…
file size (bytes)	80			18,855,234				80				1057	
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47	

inode table

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

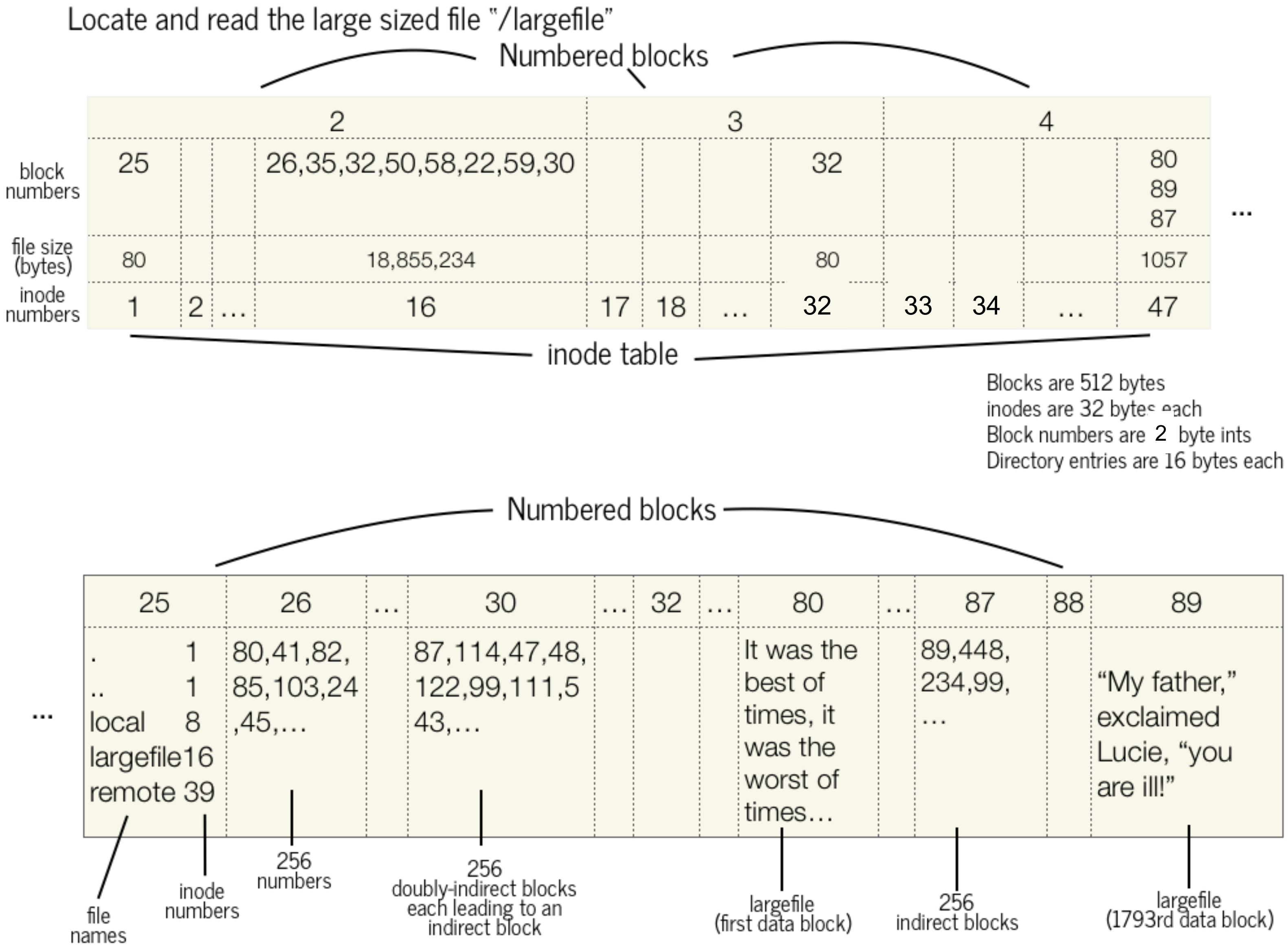
Numbered blocks

	25	26	…	30	…	32	…	80	…	87	88	89
…	. 1 .. 1 local 8 largefile16 remote 39	80,41,82,85,103,24,45,…		87,114,47,48,122,99,111,543,…				It was the best of times, it was the worst of times…		89,448,234,99,…		“My father,” exclaimed Lucie, “you are ill!”

file names
inode numbers
256 numbers
256 doubly-indirect blocks each leading to an indirect block
largefile (first data block)
256 indirect blocks
largefile (1793rd data block)

Question: What is the number of the block that stores the first 512 bytes of **largefile**?

Ex.: Finding "/largefile" (large file)



Locate and read the large sized file "/largefile"

	2				3				4				
block numbers	25			26,35,32,50,58,22,59,30				32				80 89 87	…
file size (bytes)	80			18,855,234				80				1057	
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47	

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

	25	26	…	30	…	32	…	80	…	87	88	89
…	. 1 .. 1 local 8 largefile 16 remote 39	80,41,82, 85,103,24 ,45,…		87,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		89,448, 234,99, …		"My father," exclaimed Lucie, "you are ill!"

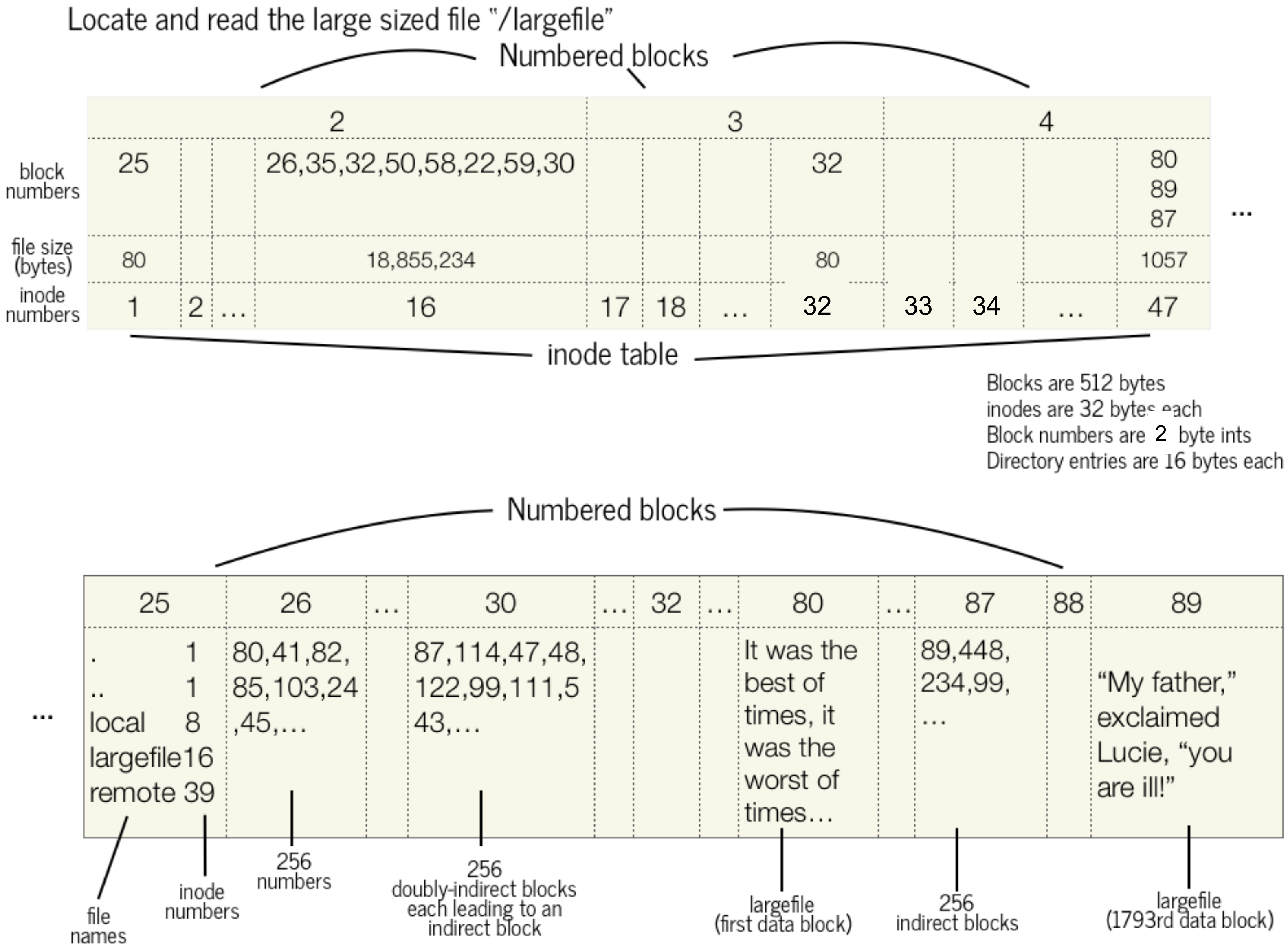
file names
inode numbers
256 numbers
256 doubly-indirect blocks each leading to an indirect block
largefile (first data block)
256 indirect blocks
largefile (1793rd data block)

1. go to inode 1. It's small. We need to look in block 25 for the list of its entries.

2. Look in block 25 for "largefile" -> inode 16.

3. Go to inode 16. It's large. For the first seven block numbers, go to those blocks and read their 256 block numbers to get payload blocks.

Ex.: Finding “/largefile” (large file)



Locate and read the large sized file "/largefile"

Numbered blocks

	2				3				4				
block numbers	25			26,35,32,50,58,22,59,30				32				80 89 87	…
file size (bytes)	80			18,855,234				80				1057	
inode numbers	1	2	…	16	17	18	…	32	33	34	…	47	

inode table

Blocks are 512 bytes
inodes are 32 bytes each
Block numbers are 2 byte ints
Directory entries are 16 bytes each

Numbered blocks

	25	26	…	30	…	32	…	80	…	87	88	89
…	. 1 .. 1 local 8 largefile16 remote 39	80,41,82, 85,103,24 ,45,…		87,114,47,48, 122,99,111,5 43,…				It was the best of times, it was the worst of times…		89,448, 234,99, …		“My father,” exclaimed Lucie, “you are ill!”

file names
inode numbers
256 numbers
256 doubly-indirect blocks each leading to an indirect block
largefile (first data block)
256 indirect blocks
largefile (1793rd data block)

4. For the eighth block, go to block 30. For each block number, go to that block and read in its block numbers to get payload blocks.

First payload block number = 80.

Plan For Today

- **Recap**: the Unix V6 Filesystem so far
- **Practice:** doubly-indirect addressing
- Directories and filename lookup
- **Practice:** filename lookup
- **Summary**

Assignment 1

- Assignment 1 due Thurs. 1/26
- Implement core functions to read from a Unix v6 filesystem disk!
  - **inode_iget** -> fetch a specific inode
  - **inode_indexlookup** -> fetch a specific payload block number
  - **file_getblock** -> fetch a specified payload block
  - **directory_findname** -> fetch directory entry with the given name
  - **pathname_lookup** -> fetch inumber for the file with the given path
- **"YEAH" Hours** (Your Early Assignment Help Hours) to be announced soon!

Recap

- **Recap**: the Unix V6 Filesystem so far
- **Practice:** doubly-indirect addressing
- Directories and filename lookup
- **Practice:** filename lookup
- Summary

Lecture 4 takeaway: The Unix V6 Filesystem represents directories as files, with payloads containing directory entries. Lookup begins at the root directory. Filesystem design is challenging, with many possibilities and tradeoffs!

Next time: how do we interact with the filesystem in our programs?