

Introduction to Operating Systems and SQL for Data Science

Lecture 5 - Files

Previous lecture

- Memory hierarchy – Disk, RAM
- Memory management
- Swapping
- Virtual memory
- Page vs. Frame
- Garbage collection

RAM allocation

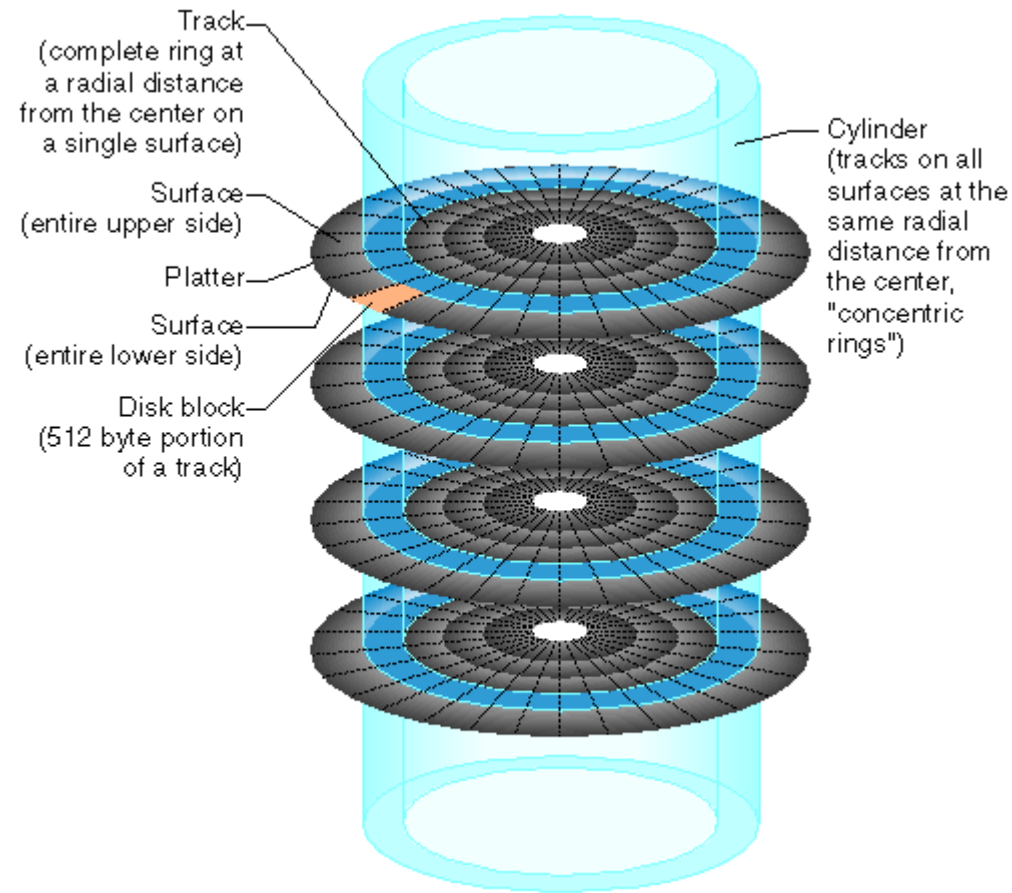


Blocks & Files

Block – an atomic data unit for reading and writing to a disk.

File – a group of blocks that constitutes an information unit.

Physical Disk Structure



File attribute

All the file attributes are stored in the metadata.

(Metadata is "data that provides information about other data")

- Size
- Extensions
- Access permissions
- Passwords
- Time – time of creation/modifying/reading










File types

- Text – represented in an ascii code

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	
29	1d	035	GS	61	3d	075	=	93	5d	135]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137		127	7f	177	DEL

File types

- Text – represented in an ascii code

Emoticon, Smiley Description		Unicode Hex	HTML Dec Code
	Grinning Face	U+1F600	😀
	Grinning Face With Big Eyes	U+1F603	😃
	Grinning Face With Smiling Eyes	U+1F604	😄
	Beaming Face With Smiling Eyes	U+1F601	😁
	Grinning Squinting Face	U+1F606	😆
	Grinning Face With Sweat	U+1F605	😅
	Rolling On The Floor Laughing	U+1F923	🤣
	Face With Tears Of Joy	U+1F602	😂
	Slightly Smiling Face	U+1F642	🙂

File types

- Text – represented in an ascii code
- Binary – not text

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

File types

- Executable files – is a file that is used to perform various functions or operations on a computer. (compiled on the specific machine – cannot be read)



File types

- Text – represented in an ascii code
- Binary – not text (contains images, sounds, compressed versions of other files, etc. – in short, any type of file content whatsoever)
- Executable files – is a file that is used to perform various functions or operations on a computer. (compiled on the specific machine – cannot be read)
- Folders (contains files)

File extensions

Extension	Meaning
.bak	Backup file
.c	C source program
.gif	Compuserve Graphical Interchange Format image
.hlp	Help file
.html	World Wide Web HyperText Markup Language document
.jpg	Still picture encoded with the JPEG standard
.mp3	Music encoded in MPEG layer 3 audio format
.mpg	Movie encoded with the MPEG standard
.o	Object file (compiler output, not yet linked)
.pdf	Portable Document Format file
.ps	PostScript file
.tex	Input for the TEX formatting program
.txt	General text file
.zip	Compressed archive

Figure 4-1. Some typical file extensions.

File's structure

File's structure

Files can be structured in several of ways.
The most three common ones in Figure 4-2.

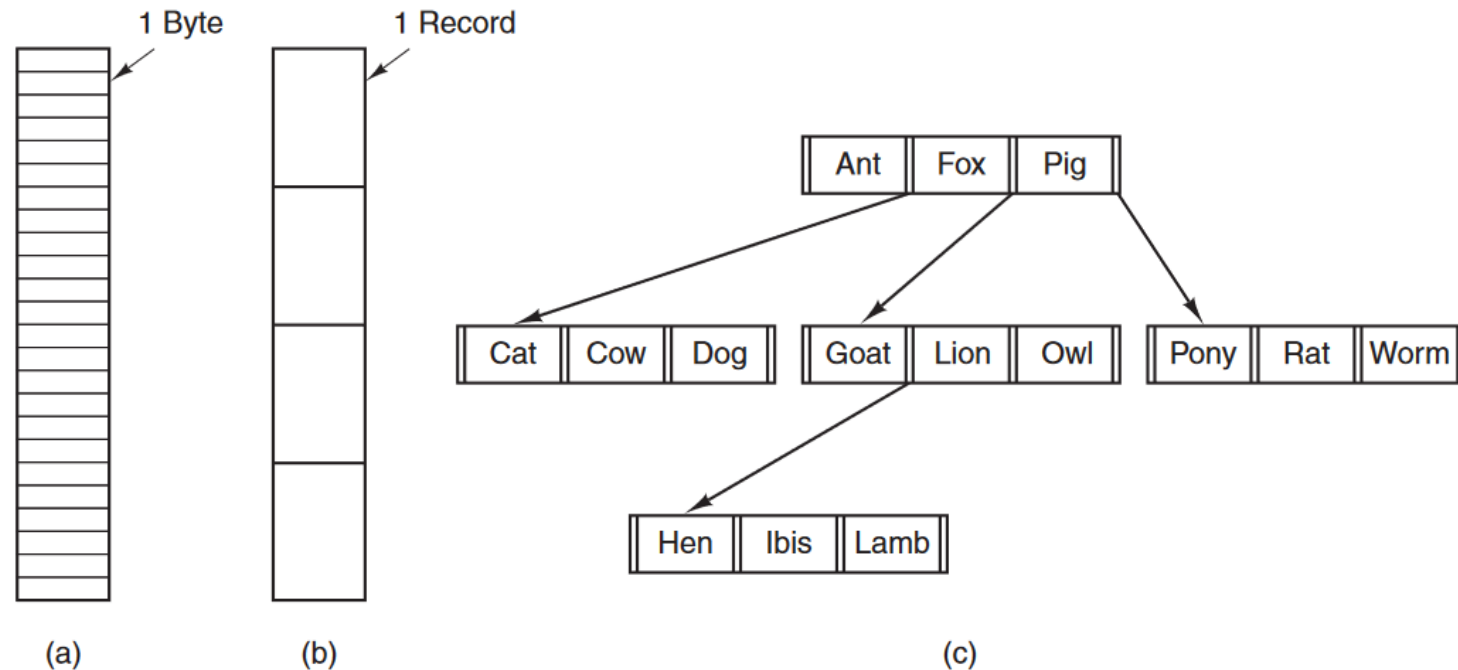


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

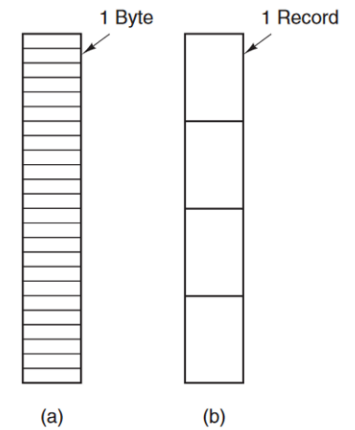
Byte/Record sequence

Only continuous blocks could be files.

For example, $b_1, b_2, b_3, b_4 \Rightarrow$ file.

But b_1, b_7, b_8 can't be a file.

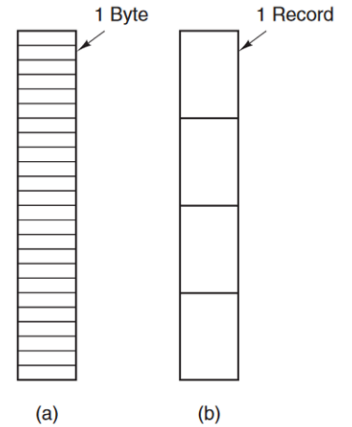
A minimal file is in the size of a block.



Byte/Record sequence

Disadvantages – not support in dynamic changing.

Advantages – good for read-only files, which we don't want to change. Like a movie file or an app installer.



Byte/Record sequence

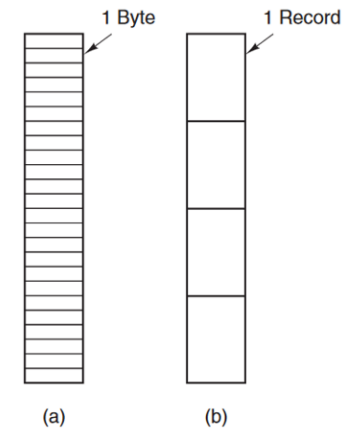
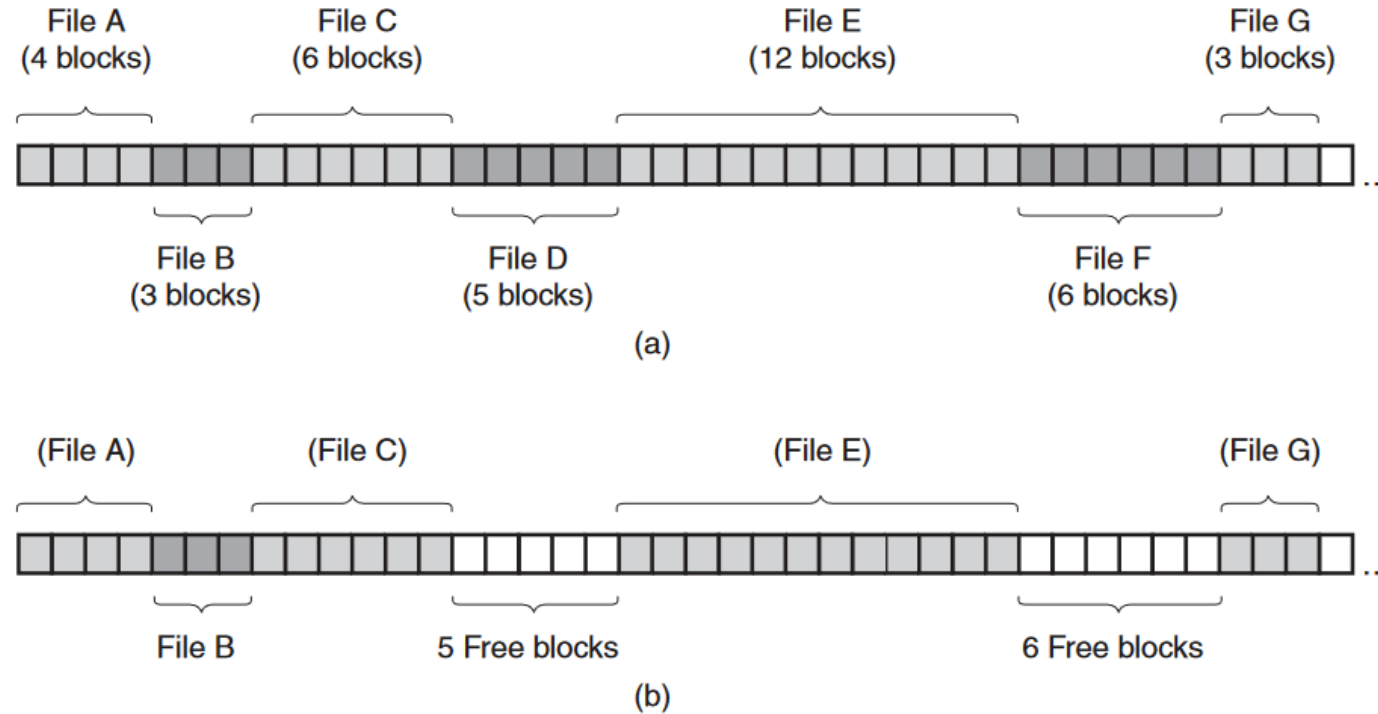
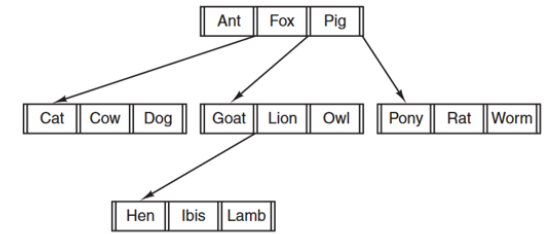
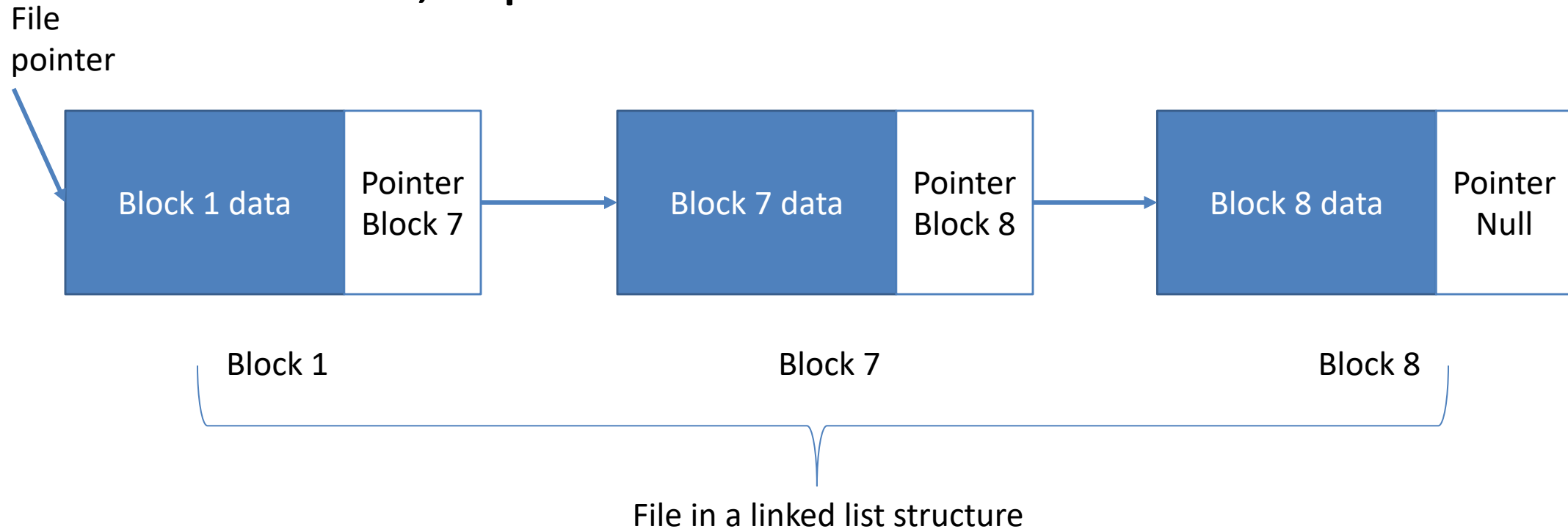


Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Linked list / Tree



A linked list that contain blocks and at the end of the block, a pointer for the next block.



Linked list / Tree

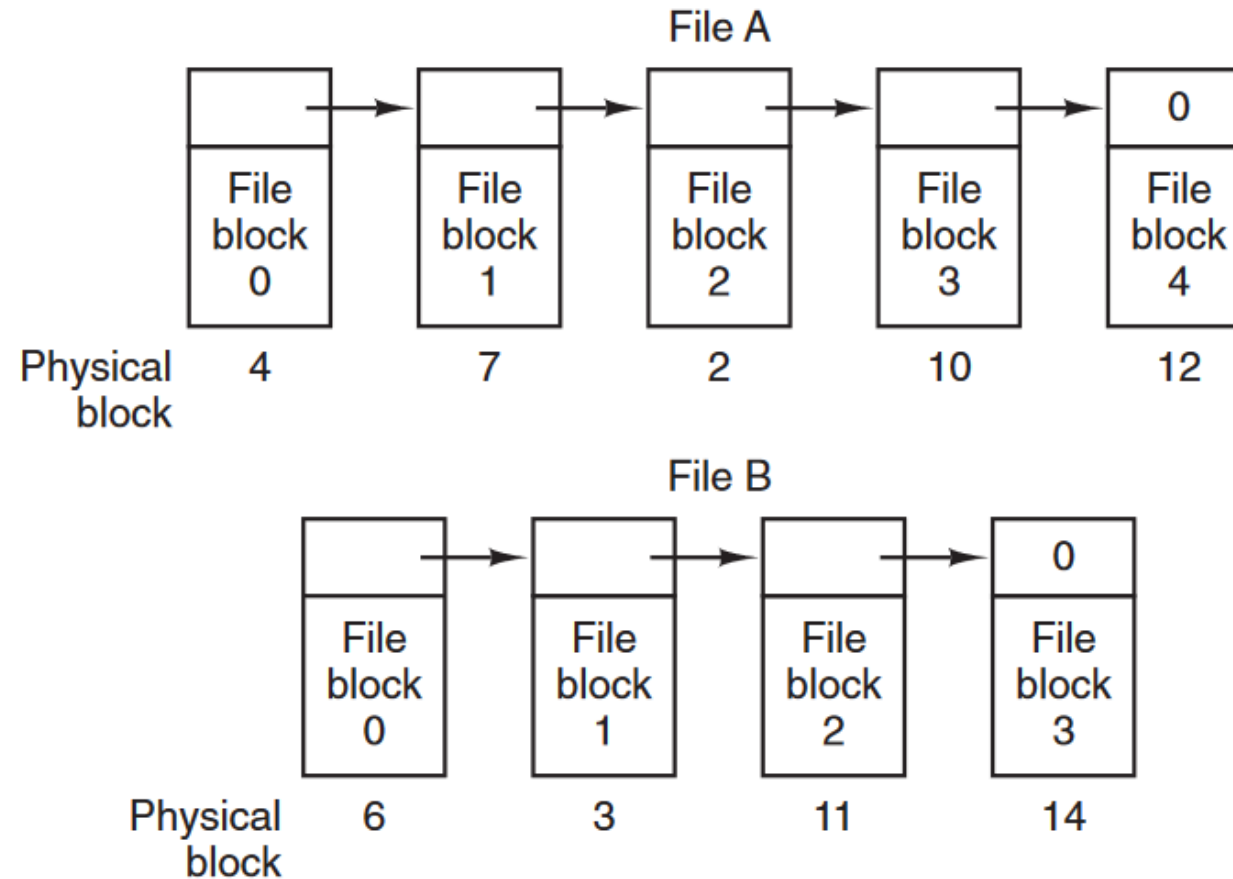
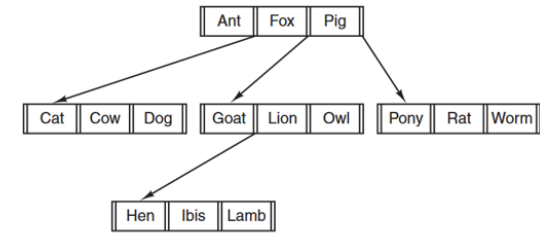
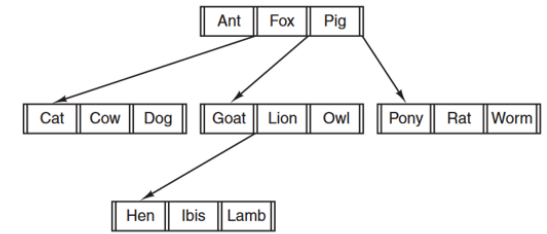


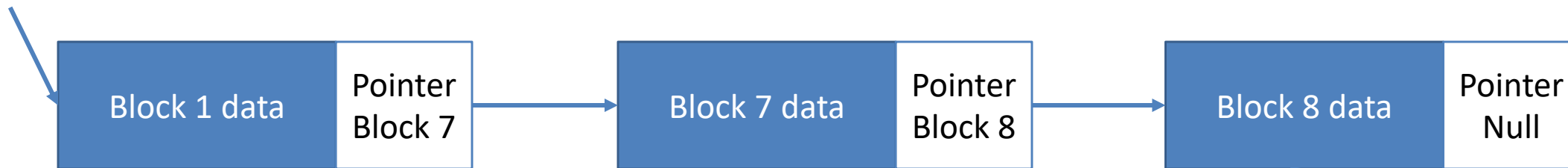
Figure 4-11. Storing a file as a linked list of disk blocks.

Linked list - Problem



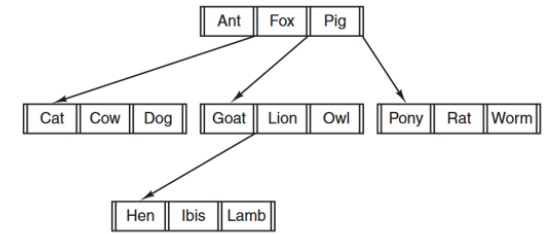
If we want to access the middle of a file, we need to read all prior block to reach a middle block.

For example, if we want to read Block 8, we firstly need to read Block 1 and 7.



We can't access directly to block 8

Linked list - Solution



When would we like to read a middle block? For example, see the middle of a movie. Or jump to the middle of a text file.

Solution – we will separate the structure(blocks order) of the list from the data.

Different solutions implementations

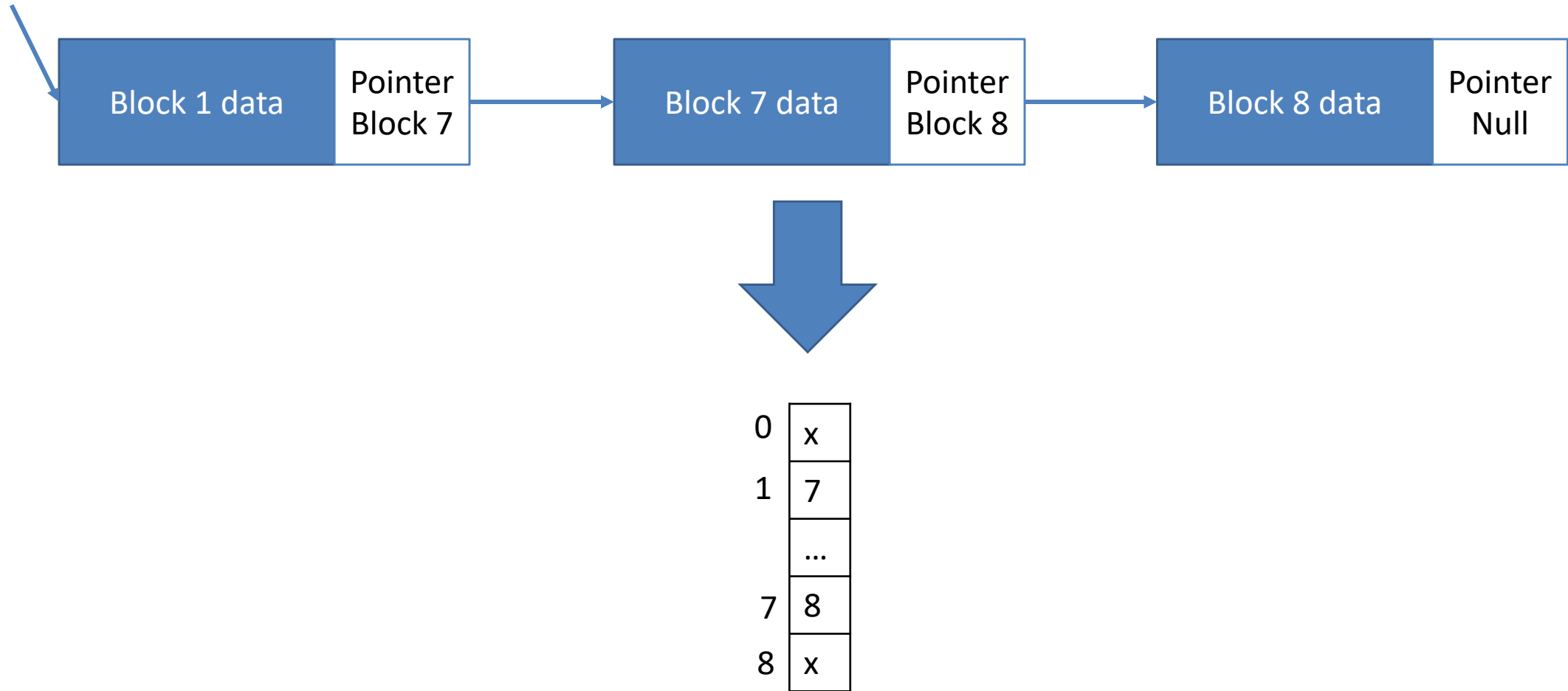
- Windows implementation (FAT) – we store the structure of the file (blocks order) in an auxiliary table.
- Linux implementation (inode) – We save the list of the structure of each file at the “start of the file”. We will define special blocks that save the list structure and other blocks that saves the information.

File implementation

File Allocation Table (FAT)

- We will save one table with size of number of blocks.
- While each cell in the table indicate the pointer of the block.
- We can trace the relevant blocks through the table instead of reading the entire blocks (and then read the pointer)

File Allocation Table (FAT)



File Allocation Table (FAT)

0	x
1	7
	...
7	8
8	x

Given the FAT we can recover the last block without reading all the prior blocks.

When can see that for the specific file the blocks order are: 1->7->8

File Allocation Table (FAT)

0	x
1	7
	...
7	8
8	x

We can save the FAT in the memory and while we delete a file, we delete its reference in the FAT.

This is much efficient way to delete blocks as all the table is on memory

File Allocation Table (FAT)

Book example:

File A: 4->7->2->10->12

File B: 6->3->11->14

Physical block		
0		
1		
2	10	
3	11	
4	7	← File A starts here
5		
6	3	← File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	
13		
14	-1	
15		← Unused block

Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

File Allocation Table (FAT) - example

Assume we have a disk of size 250GB, and a block size 2KB, the Bus size is 16bits.

What is the size of the FAT?

Hint – the number of the FAT rows is equal to the number of blocks in the disk

File Allocation Table (FAT) - example

How many blocks we have on the disk?

$$\#blocks = \frac{disk\ size}{block\ size} = \frac{250\ GB}{2\ KB} = \frac{250 \times 2^{30}}{2 \times 2^{10}}$$

$$\approx \frac{2^8 \times 2^{30}}{2^1 \times 2^{10}} = \frac{2^{38}}{2^{11}} = 2^{27}$$

Hence, the number of bits we have to represent an index is 27.

We add another bit to represent EOF(-1/x), total of 28.

File Allocation Table (FAT) - example

Size of word is 16 bits (the size of the Bus), we can insert 2 words into each FAT cell.

(32>28>16, we round up)

The size of the FAT:

$$2 \text{ words} \times 2^{28} = 4B \times 2^{28} = 2^{30}B \approx 1GB$$

(2*16bits/8bits in Byte=4B)

Index node (inode)

- We define a special blocks that saves the order of the file blocks
- In Linux the inode also save the file's attributes (as metadata).
- How many pointers can we have in a single inode?

Index node (inode)

- How many pointers can we have in a single inode?

$$\#indexes = \frac{block\ size - attribute\ size}{index\ size}$$

Let's continue with the previous example, where the attribute size is 512B.

Index node (inode)

- How many pointers can we have in a single inode?

$$\#indexes = \frac{2KB - 512B}{2 \text{ words}} = \frac{1.5KB}{4B} = 3 \times 2^7$$

What is the maximal file size we can save with a single inode?

Index node (inode) – maximal size file

What is the maximal file size we can save with a single inode?

$$\begin{aligned} & \text{number of pointers} \times \text{block size} \\ &= (3 \times 2^7) \times (2 \times 2^{10}) = 3 \times 2^{18} = 0.75 \text{MB} \end{aligned}$$

Index node (inode) – maximal size file

What is the maximal file size we can save with a single inode?

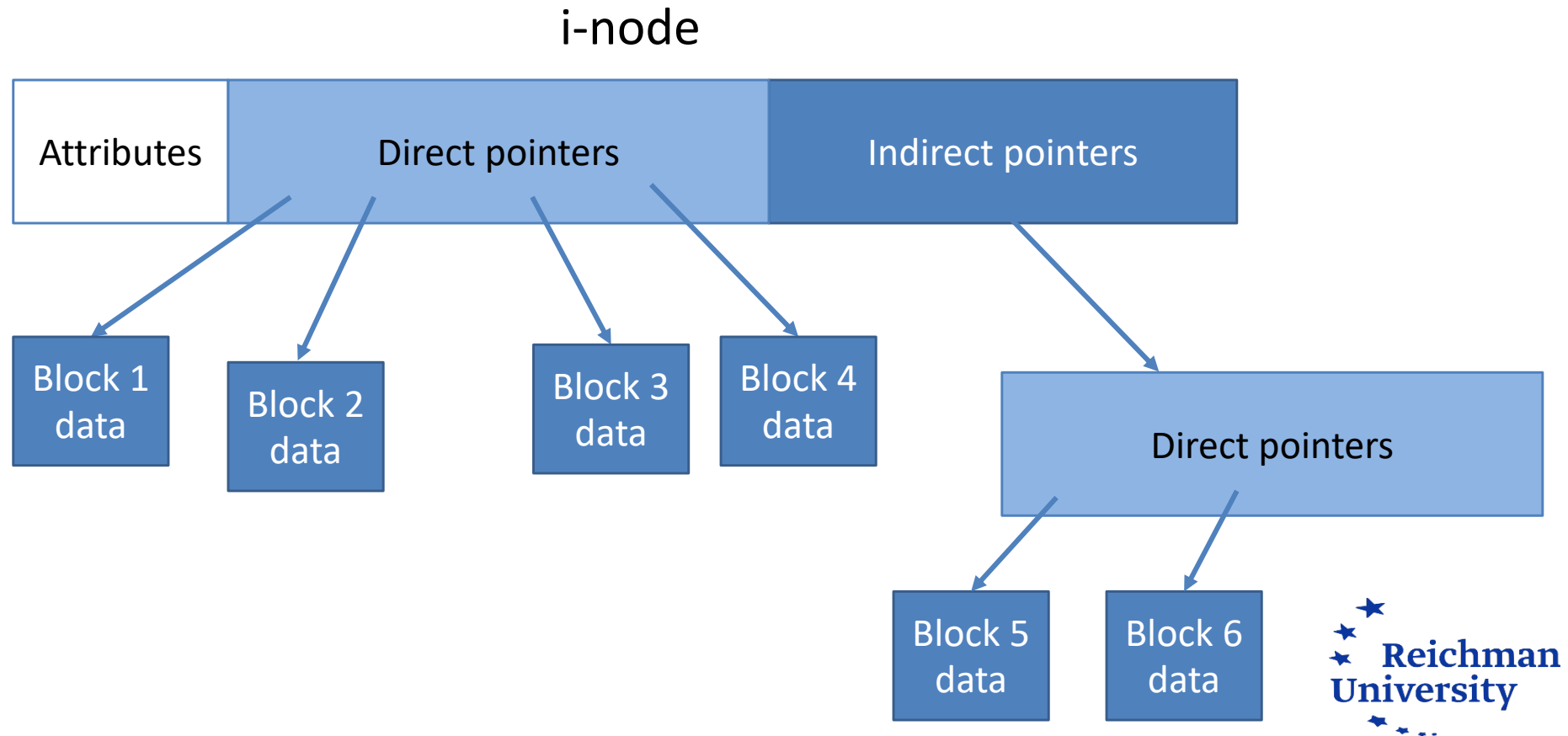
$$\begin{aligned} & \text{number of pointers} \times \text{block size} \\ &= (3 \times 2^7) \times (2 \times 2^{10}) = 3 \times 2^{18} = 0.75 \text{MB} \end{aligned}$$

How can we save larger files?

We need to create more pointers...

Index node (inode) – indirect pointers

From the inode we use the pointers not to a block of data, but rather to another block of pointers.



Index node (inode) – indirect pointers

Attributes – the metadata of the file (size, type)

Direct – pointers to a block of data

Indirect – pointers to a block of direct pointers

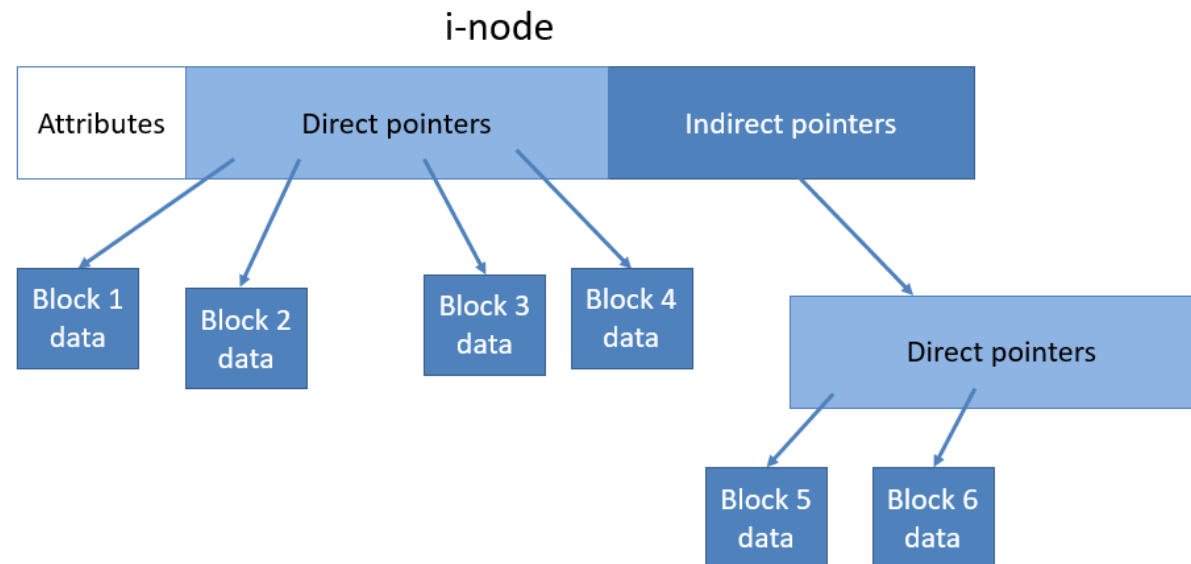
Double indirect – pointers to a block of indirect pointers

Double indirect - example

Let's expand our example.

Assume from the ten last pointers in the i-node they point to a direct block of pointers.

What is the maximal file size



Indirect - example

We have 10 indirect pointers (given from the question)

How much direct pointers we have?

We have 3×2^7 pointers overall:

$$\text{direct} = 3 \times 2^7 - 10(\text{indirect pointers})$$

How much direct pointers we have from the indirect pointers?

$$\text{indirect} = \# \text{indirect}$$

Indirect - example

How much direct pointers we have from the indirect pointers?

$$\#indirect \times \#pointers \text{ in block} = 10 \times \frac{2KB}{4B} = 10 \times 2^9$$

$$\#pointers \text{ in block} = \frac{\text{block size}}{\text{pointer size}}$$

Indirect - example

Maximal file size is: $\#direct\ pointers \times block\ size$

$$\#direct\ pointers = 10 \times 2^9 + 3 \times 2^7 - 10 \approx 11 \times 2^9$$

$$block\ size = 2KB = 2^{11}$$

$$\max\ file\ size = 11 \times 2^9 \times 2^{11} = 11MB$$

Folders

- A table of files belongs to the folder – each row is a file
- In Unix, we need to hold the number of the i-node of a file
- In Windows, we need to hold the first block of the file and the attributes of the file.
- In Unix, it takes more time to show details on the folder since we need to read the attributes data from each i-node while in Windows we have all the data in the FAT.

Memory Vs. Time....

Directories

Folders

- To keep track of files, file systems normally have directories or folders, which are themselves files.
- There are few ways to implement directories:
 - Single-Level Directory System
 - Hierarchical Directory System

Folders

We can use linking to avoid redundant space allocations

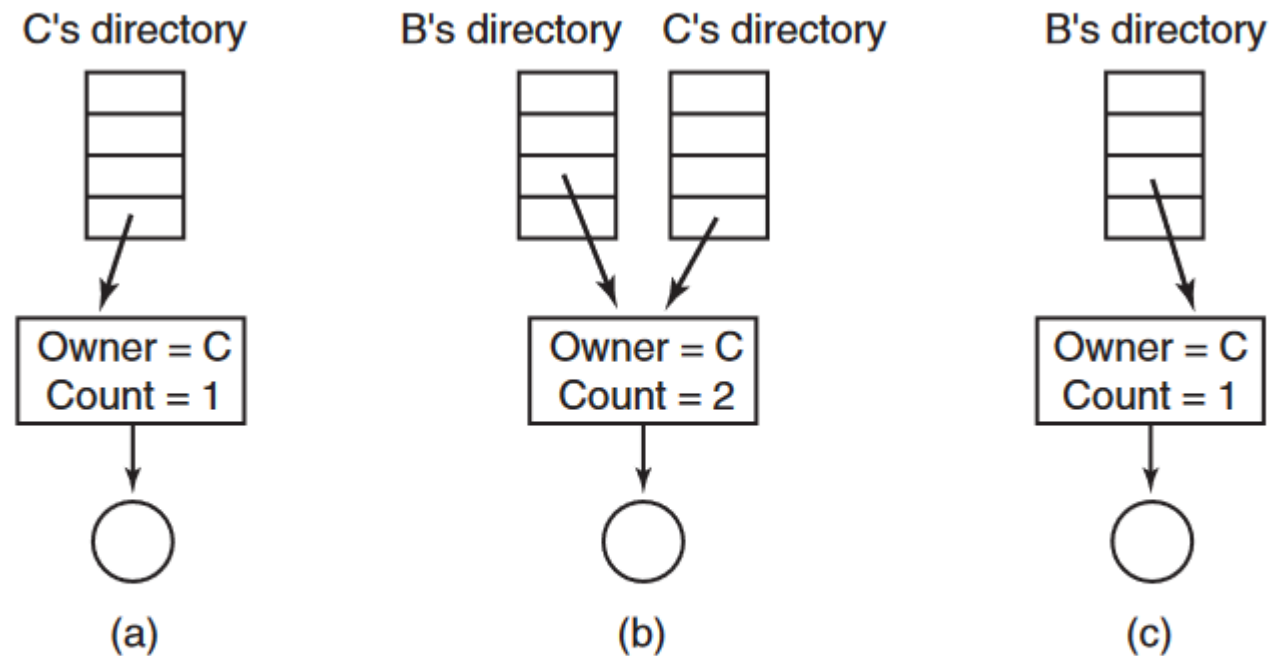


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.