Q1

Indicate whether the following statements are true or false. Justify your answers.

a) Assuming the protection bits of a file are “r - x r - x - - x”, only the owner of this file is able to read the file.

b) Open file table is used to temporarily cache data blocks of a file to improve efficiency.

c) Linked file allocation method may result in external fragmentation.

Q1

Indicate whether the following statements are true or false. Justify your answers.

a) Assuming the protection bits of a file are "r - x r - x - - x", only the owner of this file is able to read the file.
False: Both owner of the file and users in the same group of the owner can read the file.

b) Open file table is used to temporarily cache data blocks of a file to improve efficiency.
False: Open file table is used to temporarily cache file control block of a file to improve efficiency

c) Linked file allocation method may result in external fragmentation.
False: Any free physical block can be used in the linked allocation method. So, there is no external fragmentation.

Q2

(a) Consider a file system where a file can be deleted and its disk space reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?

Q2 (a)

- Let F1 be the old file and F2 be the new file, respectively.
- Two problems for the “phantom”:
  - Data content: accessing F2 wrongly.
  - Access protection: accessing F2 without checking its access protection (for hard-link)

How to Solve the Problem?

- Insure that all links to a deleted file are deleted also.
  - Delete all the links when the file is deleted.
  - Delete the file only after all links have been deleted.

Q2

(b) Some systems provide file sharing by maintaining a single copy of a file; other systems maintain several copies, one for each of the users sharing the file. Discuss the relative merits of each approach.

Q2 (b): Single Copy vs. Multiple Copies

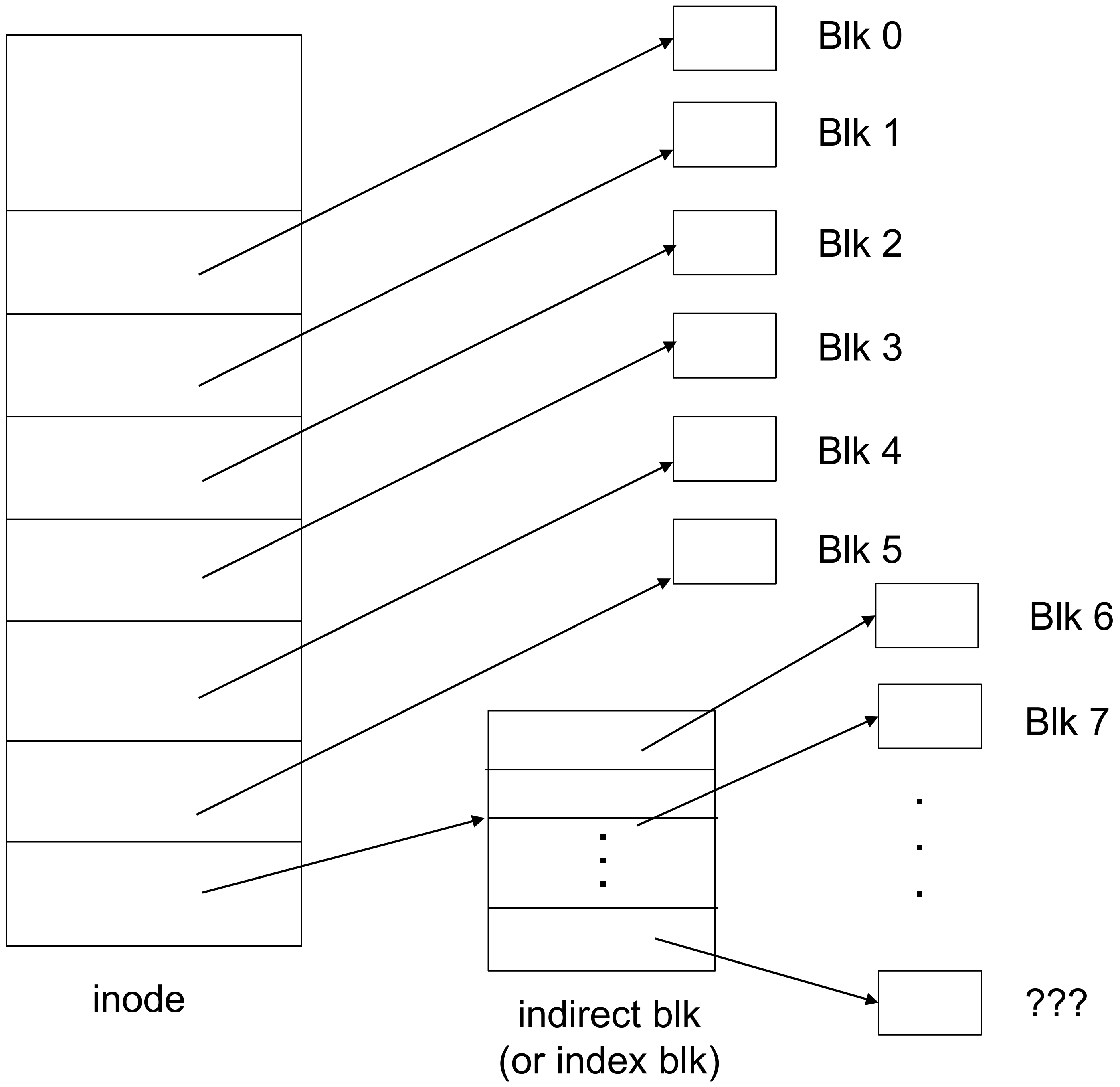
- With single copy:
  - Concurrent updates may result in race condition.
  - Mutual Exclusion must be enforced on file access.
- With multiple copies,
  - Storage waste.
  - The various copies may be inconsistent.
  - Performance is better (mutual exclusion is not necessary).

Q3

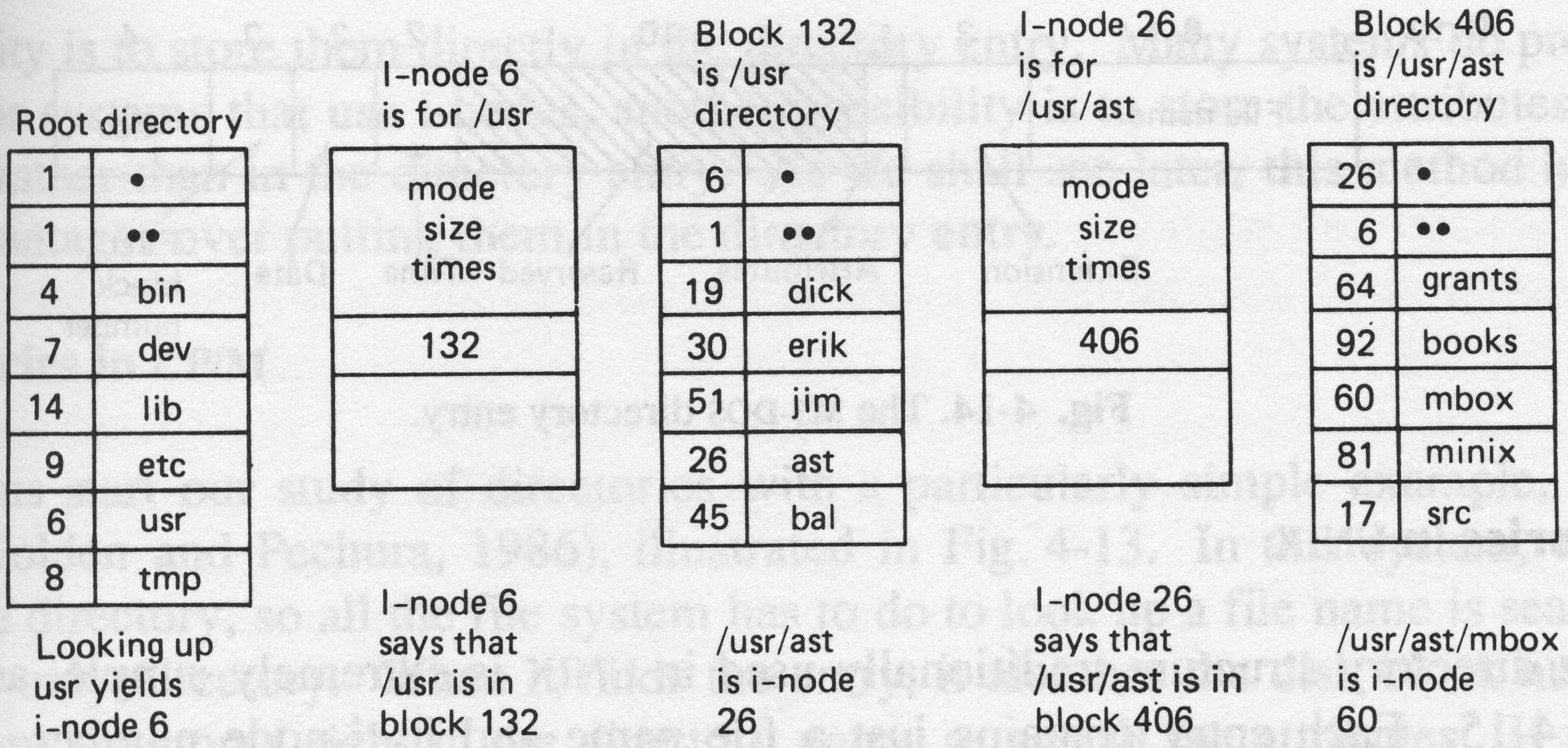
Assume that a file system uses Unix-like inodes with six direct pointers and one single-indirect pointer. The file system is block-oriented with both logical and physical block sizes of 1000 bytes. A user executes a program that contains the following code:

fd = open("/usr/ast/mbox"); // open a file
seek(fd, 5900); // move the file pointer
read(fd, buf, 200); // read 200 bytes of data from the file

(a) How many disk read operations are required for the first system call (i.e., open) in the above program fragment? Assume that initially, the root directory is in memory, inodes are in disk, and that a directory can fit into a single block.



Blk 0
Blk 1
Blk 2
Blk 3
Blk 4
Blk 5
Blk 6
Blk 7
???
inode
indirect blk
(or index blk)



Root directory

1	•
1	••
4	bin
7	dev
14	lib
9	etc
6	usr
8	tmp

Looking up usr yields i-node 6

I-node 6 is for /usr

mode size times
132

I-node 6 says that /usr is in block 132

Block 132 is /usr directory

6	•
1	••
19	dick
30	erik
51	jim
26	ast
45	bal

/usr/ast is i-node 26

I-node 26 is for /usr/ast

mode size times
406

I-node 26 says that /usr/ast is in block 406

Block 406 is /usr/ast directory

26	•
6	••
64	grants
92	books
60	mbox
81	minix
17	src

/usr/ast/mbox is i-node 60

Q3 (a)

- Five.
- Open**("/usr/ast/mbox")**:

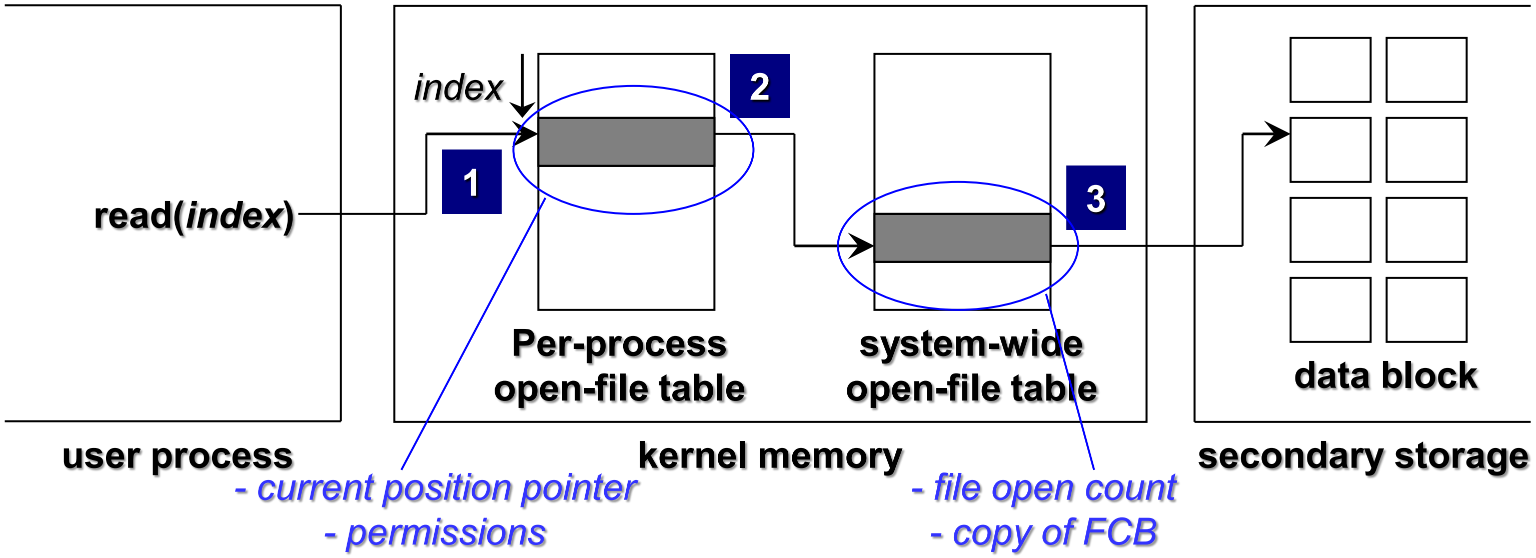
    load inode of "usr"

    load data block of "usr" (i.e., directory "usr")

    load inode for "ast"

    load data block of "ast" (i.e., directory "ast")

    load inode for "mbox"



- *Current position pointer & amount of data to read ⇒ logical file blocks to be accessed*
- *Information in FCB & file allocation method ⇒ where these blocks are on hard-disk*

Q3 (b)

How many disk read operations are required for the last two system calls (i.e., seek and read) in the above program fragment?

```
seek(fd, 5900);
```

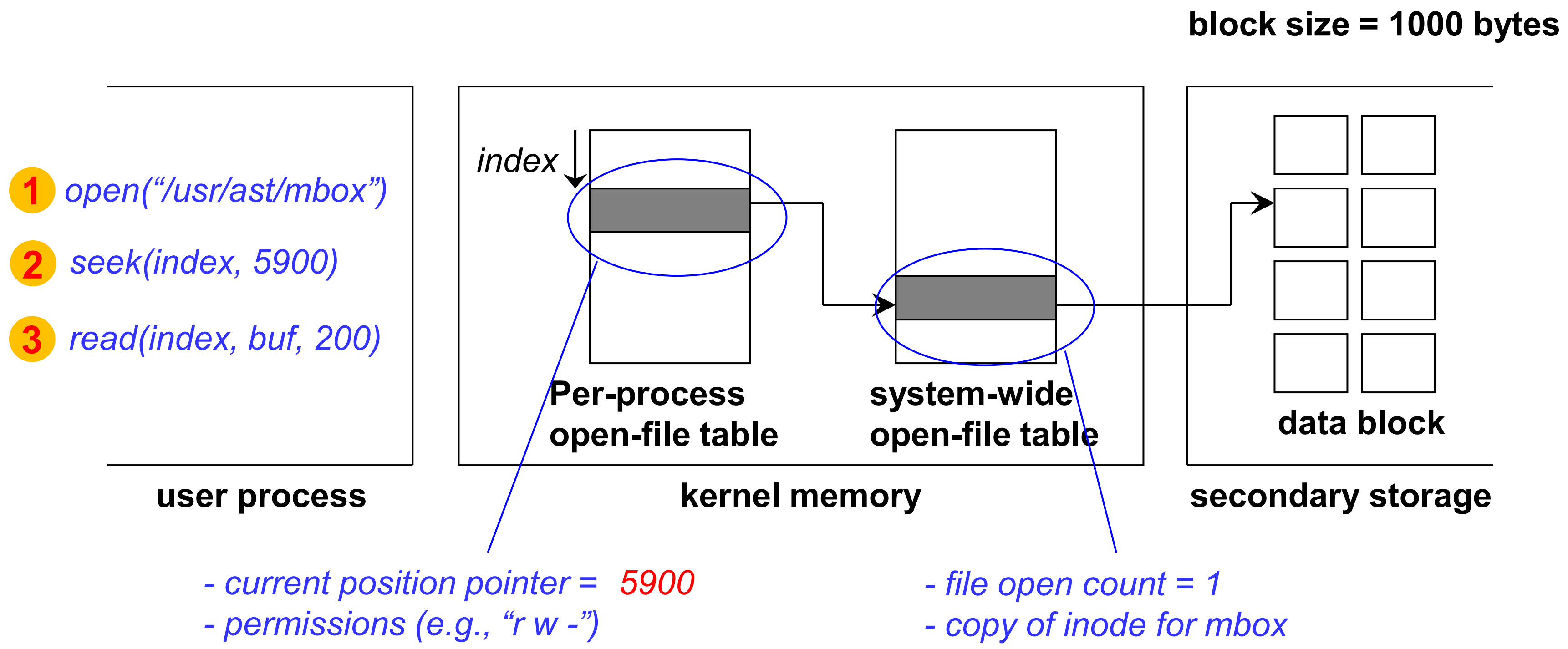
The seek requires **no** disk read operation.

```
read(fd, buf, 200);
```

The read partially loads blocks 5 and 6 of the file.
Three disk block read operations:

- read block 5
- read indirect block
- read block 6

Q3 (a)&(b)



Local data block on hard-disk:

- *current position pointer & amount of data to read ⇒ need to access logical blocks 5 and 6*
- *information in inode ⇒ where logical blocks 5 & 6 are on hard-disk (i.e., physical block numbers)*

Q3 (c)

What is the maximum file size the file system can support? Assume that each entry in the indirect block takes 2 bytes.

$$\underbrace{6 \times 1000}_{\text{Direct pointers}} + \underbrace{(1000 / 2)}_{\text{\#indirect pointers}} \times 1000 = 506{,}000 \text{ bytes}$$

Q4

Some file systems use two block sizes for disk storage allocation, for example, 4-Kbyte and 512-byte blocks. Thus, a 6 Kbytes file can be allocated with a single 4-Kbyte block and four 512-byte blocks. Discuss the advantage of this scheme compared to the file systems that use one block size for disk storage allocation.

Answer: The approach is to take advantages of both small and large block size.
→What if the use of small block size?
Reduce fragmentation.
→What if the use of large block size?
Improve throughput.