DATA STORAGE TECHNOLOGIES & NETWORKS

(CS C446, CS F446 & IS C446)

LECTURE 27– STORAGE

Buffer

- A buffer consists of 2 parts
  - Memory array that contain data from disk
  - Buffer header that identifies the buffer
- Kernel identifies the buffer content by examining identifier fields in buffer header.
- Buffer is an in-memory copy of a disk block
- A disk block can never map into more than one buffer at a time.

Buffer header

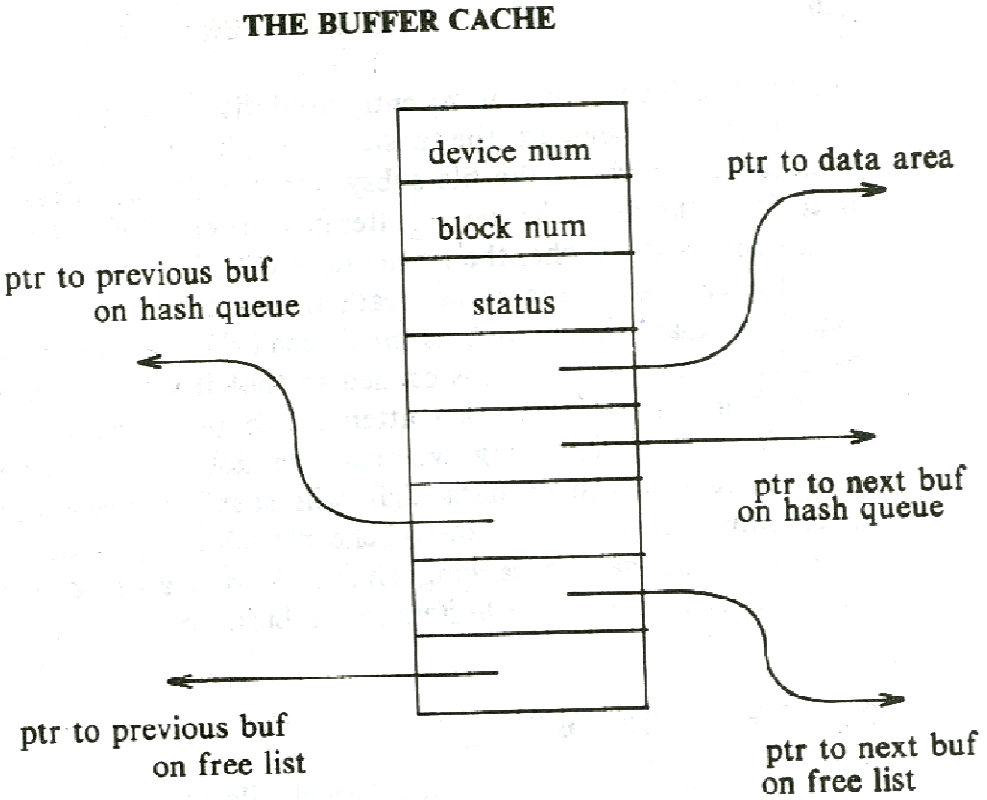


Figure 3.1. Buffer Header

Buffer header

- Device # and block # fields specifies block # of the data on disk and uniquely identifies the buffer.
  - Device # is not a physical device unit number
- Pointers in buffer header points to a data array for the buffer, whose size must be at least as big as the size of disk block
- Status of the buffer contain following information
  - The buffer is currently locked?
  - The buffer contains valid data?
  - Delayed write?
  - Kernel is currently reading or writing the contents of the buffer to disk?
  - A process is currently waiting for the buffer to become free.

Structure of the buffer pool

- Follows LRU algorithm
- Kernel maintains a free list of buffers in LRU order
- Free list is a doubly linked circular list with a dummy buffer header (marks its beginning and end).
- Every block is put on to free list when the system is booted
- Normally removal (free block) is from the head (can be from the middle as well) and addition is to the tail (occasionally to head → for error cases)

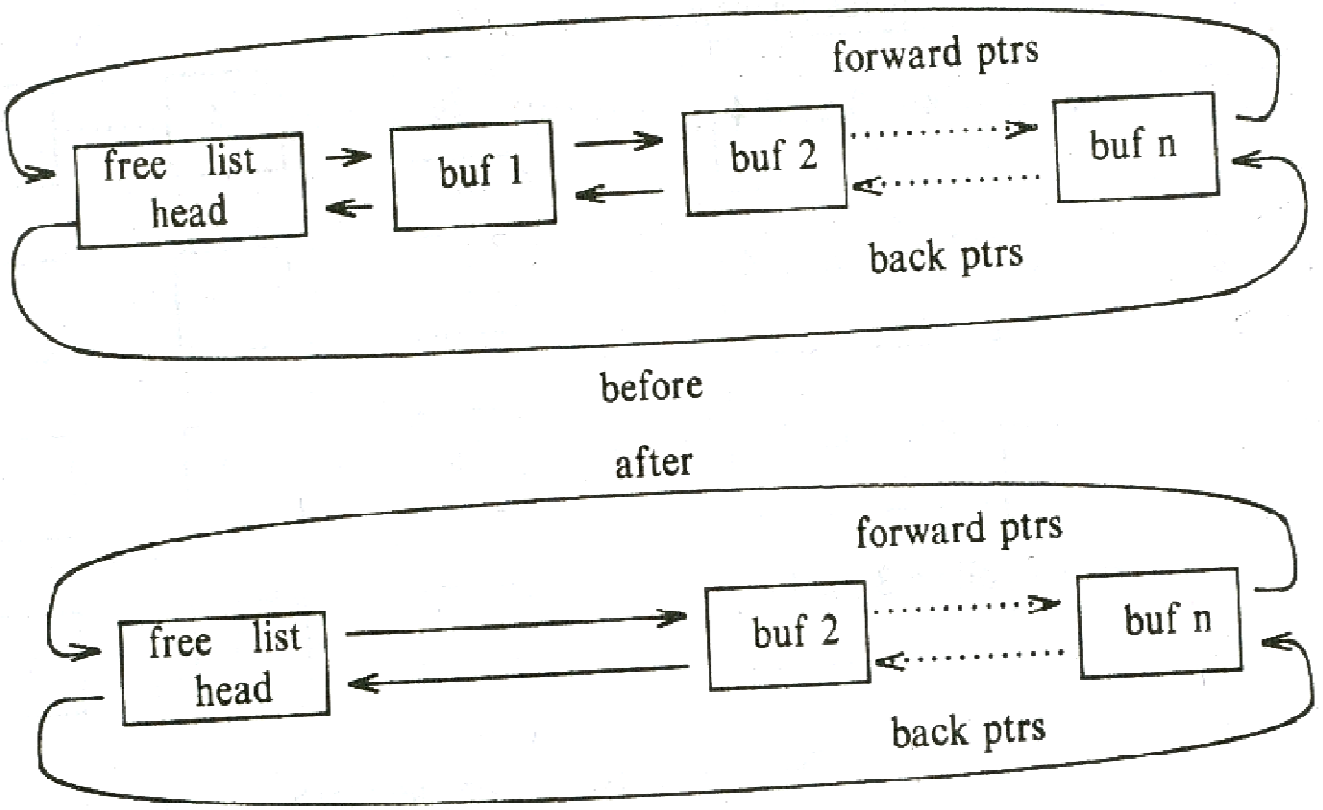


Figure 3.2. Free List of Buffers

- When kernel searches for a disk block
  - Searches for a buffer with the device – block # combination
    - Organizes buffers into separate queues, hashed as a function of device and block number.
  - Kernel links the buffers on a hash queue into a circular, doubly linked list, similar to the structure of the free list.
  - System administrator configure the number of hash queues when generating the OS.

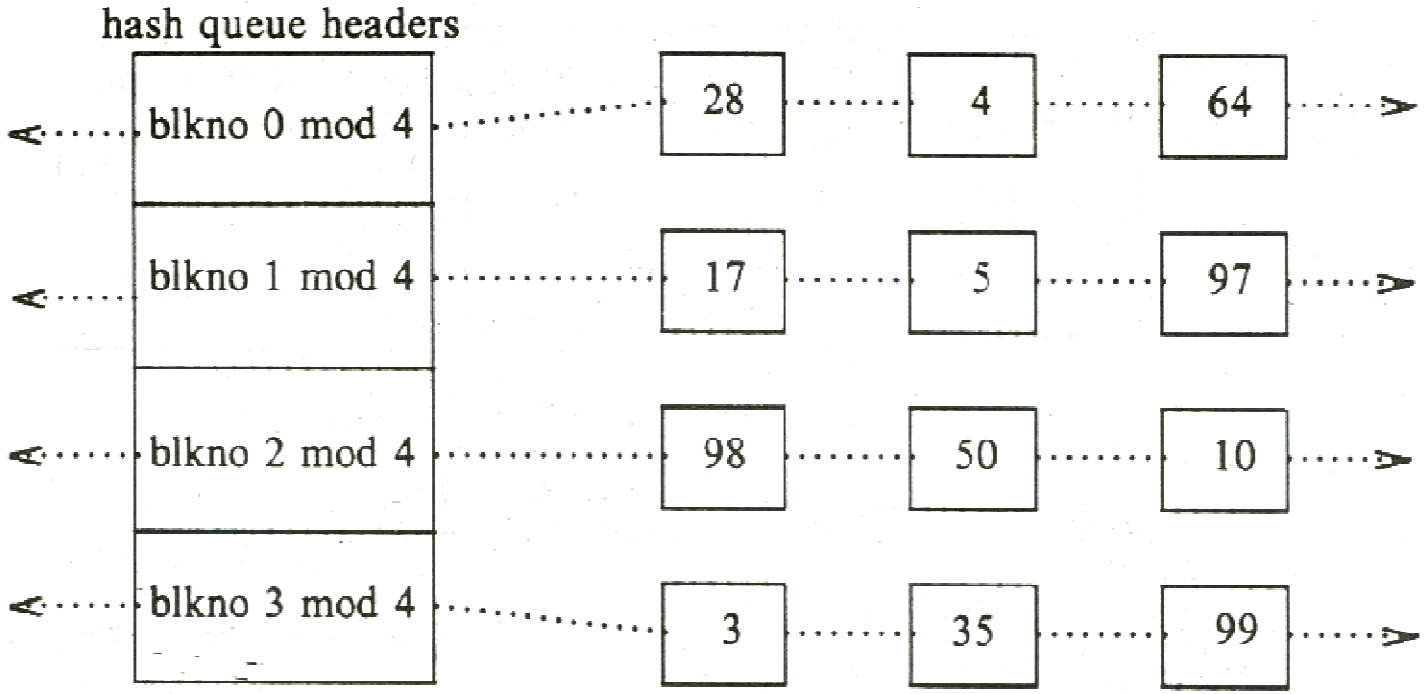


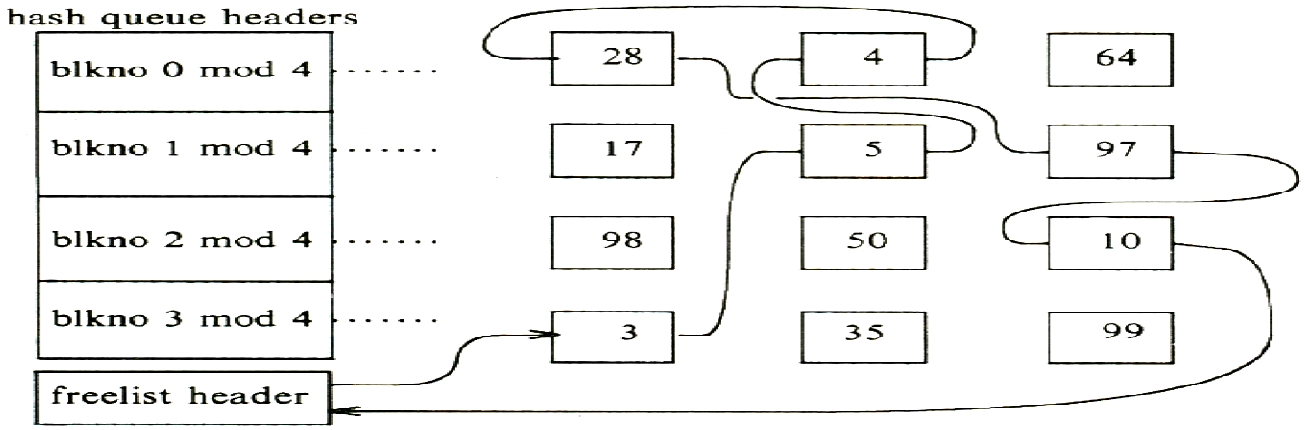
Figure 3.3. Buffers on the Hash Queues

Scenarios for retrieval of a buffer

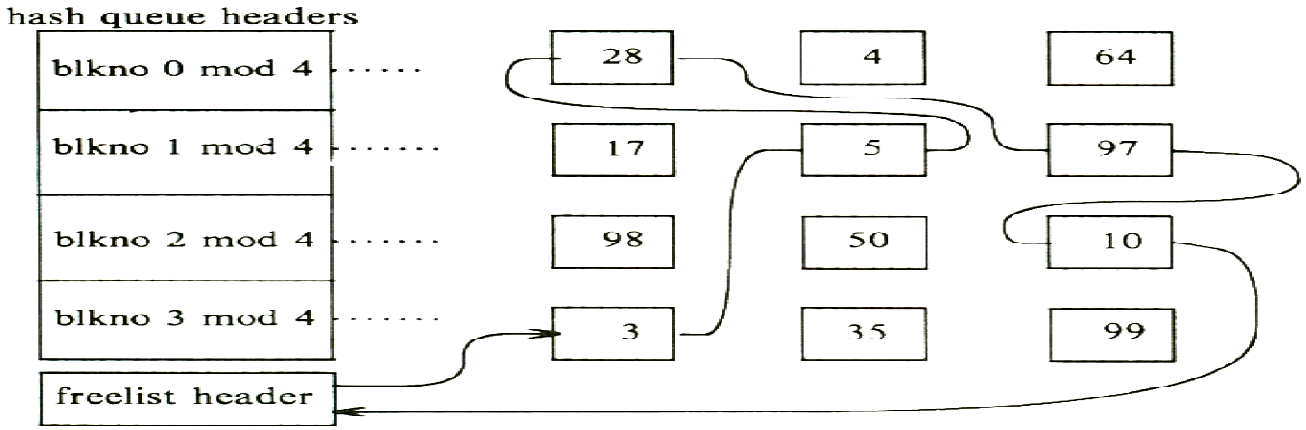
- The high level algorithms determine the logical device # and block # they wish to access
  - Example. If a process want to read data from a file
    - Kernel determines
      - Which file system
      - Which block in the file system
      - Whether the block is in the buffer pool
      - If not assign a free pool
- The algorithm for reading and writing disk blocks use the algorithm **getblk** to allocate buffer from the pool.

Scenarios in getblk to allocate a buffer for a disk block

- The kernel finds the block on its hash queue, and its buffer is free.
- The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list
- The kernel cannot find the block on the hash queue and in attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked “delayed write”. The kernel must write the delayed write buffer to disk and allocate another buffer.
- The kernel cannot find the block on the hash queue, and the free list of buffer is empty.
- The kernel finds the block on the hash queue, but its buffer is currently busy.



(a) Search for Block 4 on First Hash Queue



(b) Remove Block 4 from Free List

Figure 3.5. Scenario 1 in Finding a Buffer: Buffer on Hash Queue

```
algorithm getblk
input:   file system number
         block number
output: locked buffer that can now be used for block
{
        while (buffer not found)
        {
                if (block in hash queue)
                {
                        if (buffer busy)        /* scenario 5 */
                        {
                                sleep (event buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        mark buffer busy;       /* scenario 1 */
                        remove buffer from free list;
                        return buffer;
                }
                else        /* block not on hash queue */
                {
                        if (there are no buffers on free list)        /* scenario 4 */
                        {
                                sleep (event any buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        remove buffer from free list;
                        if (buffer marked for delayed write) {      /* scenario 3 */
                                asynchronous write buffer to disk;
                                continue;       /* back to while loop */
                        }
                        /* scenario 2 -- found a free buffer */
                        remove buffer from old hash queue;
                        put buffer onto new hash queue;
                        return buffer;
                }
        }
}
```

Scenario 1

Why an algorithm for release buffer

- After allocation
  - Kernel may read data from the disk to the buffer and manipulate it or write data to the buffer and possibly to disk.
  - Kernel leaves the buffer marked busy and no other process can access and read it.
    - Preventing the integrity of data in the buffer.
- So when kernel finishes using buffer, it releases the buffer according to **brelse** algorithm.

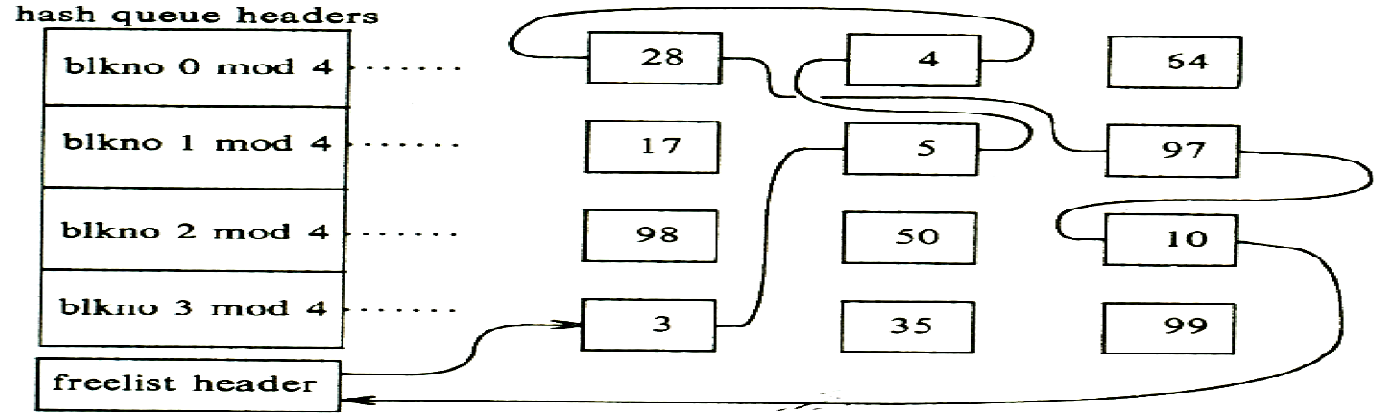
Algorithm for releasing a buffer

```
Algorithm brelse
Input: locked buffer
Output: none
{
        wakeup all proc: event, waiting for any buffer to become free;
        wakeup all proc: event, waiting for this buffer to become free;
        raise processor execution level to block interrupts;
        if(buffer contents valid and buffer not old)
                enqueue buffer at end of the free list;
        else
                enqueue buffer at beginning of the free list;
        lower processor execution level to allow interrupts;
        unlock (buffer);
}
```

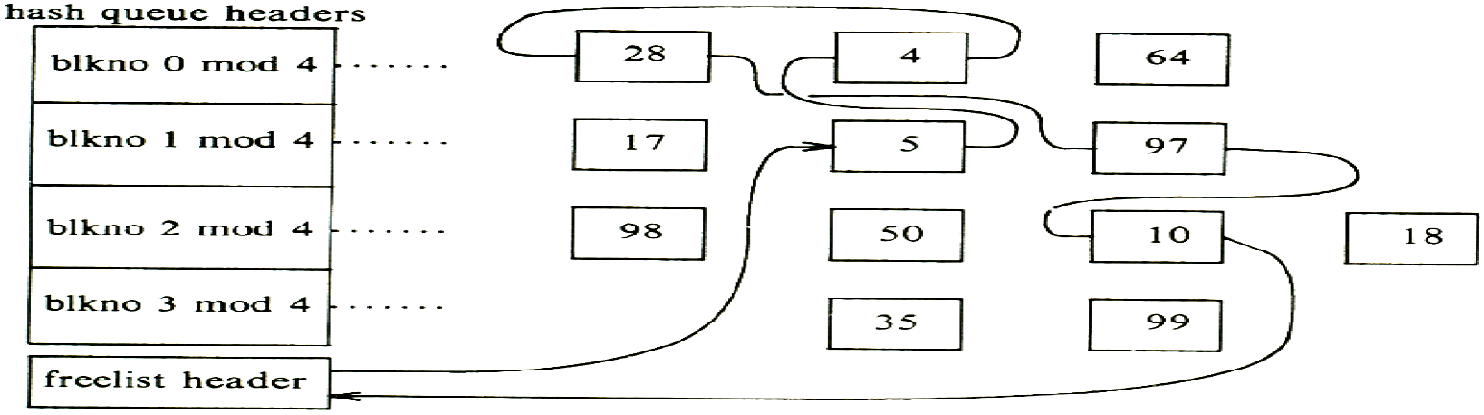
- brelse
  - Wakeup processes that had fallen asleep
    - Because the buffer was busy
    - Because no buffer remained in the free list
  - Kernel places the buffer at the end of the free list (unless an I/O error occurred or specifically marked as old)
  - Kernel raises the processor execution level to prevent disk interrupts, while manipulating free list
    - Preventing corruption of the buffer pointers that could rest in nested call to brelse
  - Kernel raises the processor execution level of getblk
    - To avoid invoking brelse by interrupt handler, while process was executing getblk.

Scenarios in getblk to allocate a buffer for a disk block

- The kernel finds the block on its hash queue, and its buffer is free.
- The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list
- The kernel cannot find the block on the hash queue and in attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked “delayed write”. The kernel must write the delayed write buffer to disk and allocate another buffer.
- The kernel cannot find the block on the hash queue, and the free list of buffer is empty.
- The kernel finds the block on the hash queue, but its buffer is currently busy.



(a) Search for Block 18 - Not in Cache



(b) Remove First Block from Free List, Assign to 18

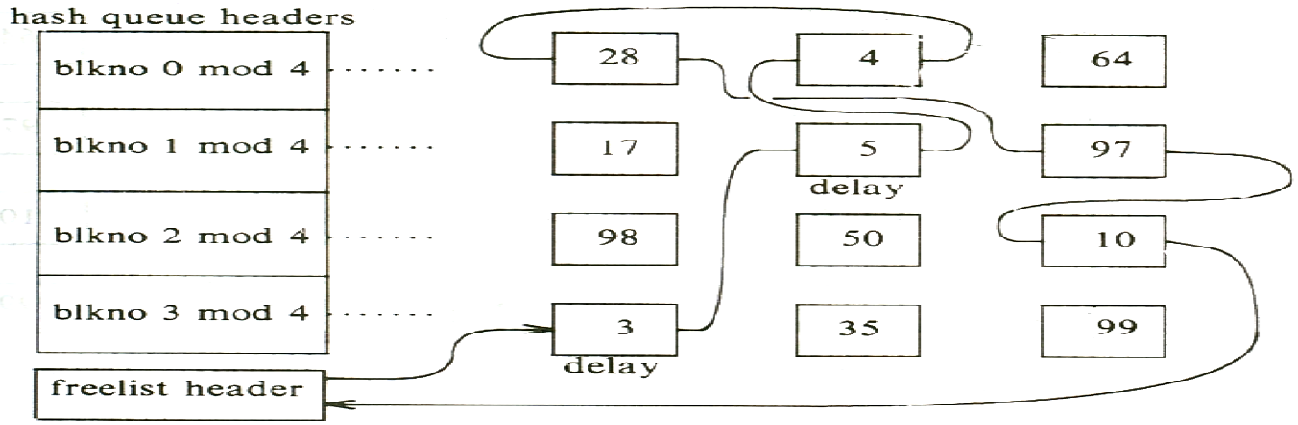
Figure 3.7. Second Scenario for Buffer Allocation

```
algorithm getblk
input:  file system number
        block number
output: locked buffer that can now be used for block
{
        while (buffer not found)
        {
                if (block in hash queue)
                {
                        if (buffer busy)        /* scenario 5 */
                        {
                                sleep (event buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        mark buffer busy;       /* scenario 1 */
                        remove buffer from free list;
                        return buffer;
                }
                else        /* block not on hash queue */
                {
                        if (there are no buffers on free list)        /* scenario 4 */
                        {
                                sleep (event any buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        remove buffer from free list;
                        if (buffer marked for delayed write) {      /* scenario 3 */
                                asynchronous write buffer to disk;
                                continue;       /* back to while loop */
                        }
                        /* scenario 2 -- found a free buffer */
                        remove buffer from old hash queue;
                        put buffer onto new hash queue;
                        return buffer;
                }
        }
}
```

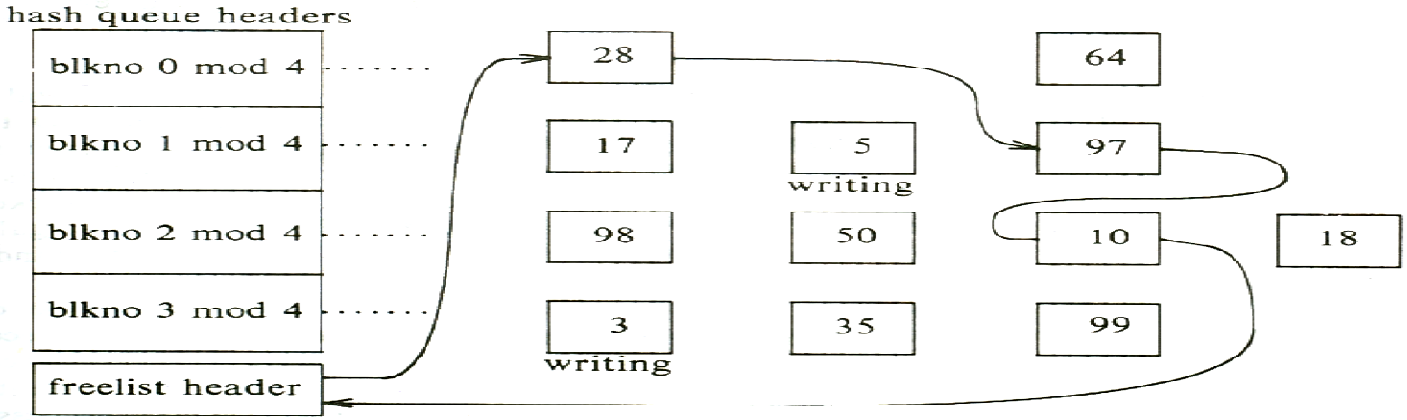
Scenario 2

Scenarios in getblk to allocate a buffer for a disk block

- The kernel finds the block on its hash queue, and its buffer is free.
- The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list
- The kernel cannot find the block on the hash queue and in attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked "delayed write". The kernel must write the delayed write buffer to disk and allocate another buffer.
- The kernel cannot find the block on the hash queue, and the free list of buffer is empty.
- The kernel finds the block on the hash queue, but its buffer is currently busy.



(a) Search for Block 18, Delayed Write Blocks on Free List



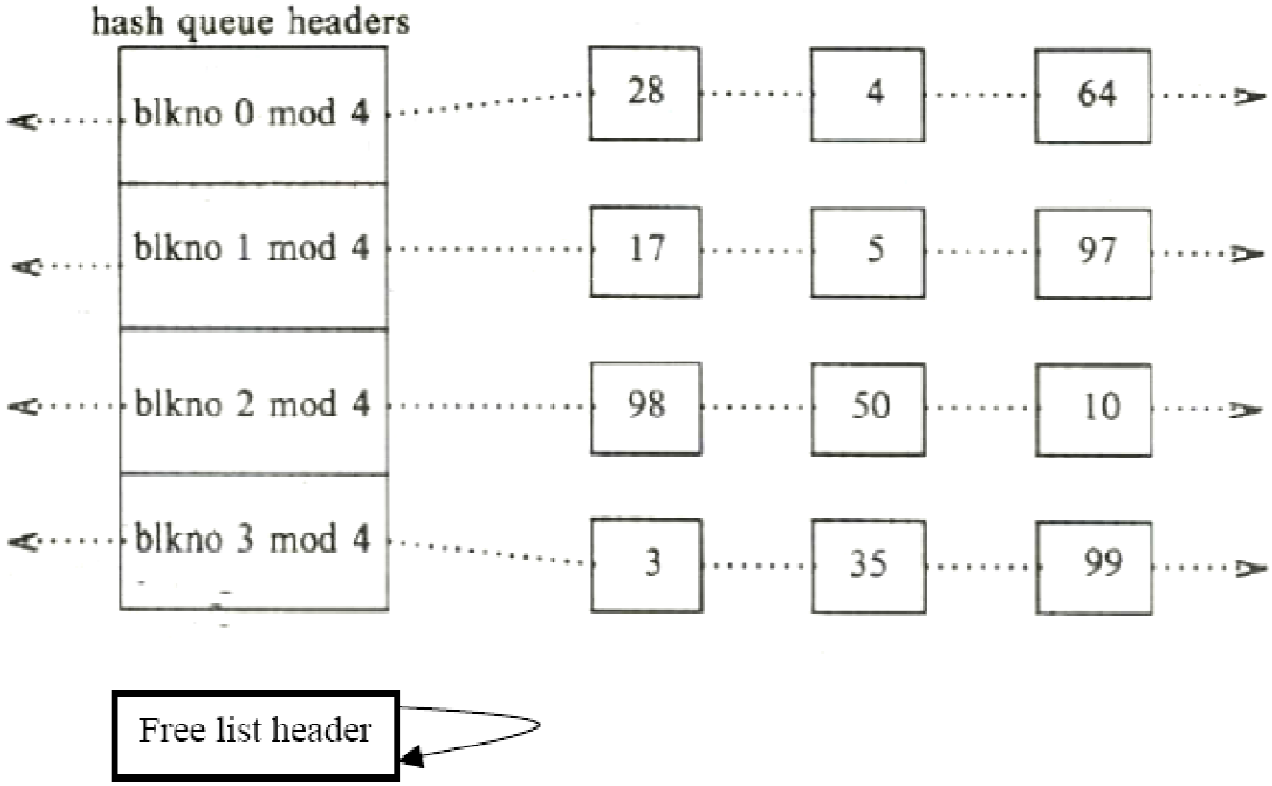
(b) Writing Blocks 3, 5, Reassign 4 to 18

Figure 3.8. Third Scenario for Buffer Allocation

```
algorithm getblk
input:  file system number
        block number
output: locked buffer that can now be used for block
{
        while (buffer not found)
        {
                if (block in hash queue)
                {
                        if (buffer busy)        /* scenario 5 */
                        {
                                sleep (event buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        mark buffer busy;       /* scenario 1 */
                        remove buffer from free list;
                        return buffer;
                }
                else        /* block not on hash queue */
                {
                        if (there are no buffers on free list)        /* scenario 4 */
                        {
                                sleep (event any buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        remove buffer from free list;
                        if (buffer marked for delayed write) {      /* scenario 3 */
                                asynchronous write buffer to disk;
                                continue;       /* back to while loop */
                        }
                        /* scenario 2 -- found a free buffer */
                        remove buffer from old hash queue;
                        put buffer onto new hash queue;
                        return buffer;
                }
        }
}
```

Scenarios in getblk to allocate a buffer for a disk block

- The kernel finds the block on its hash queue, and its buffer is free.
- The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list
- The kernel cannot find the block on the hash queue and in attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked “delayed write”. The kernel must write the delayed write buffer to disk and allocate another buffer.
- The kernel cannot find the block on the hash queue, and the free list of buffer is empty.
- The kernel finds the block on the hash queue, but its buffer is currently busy.



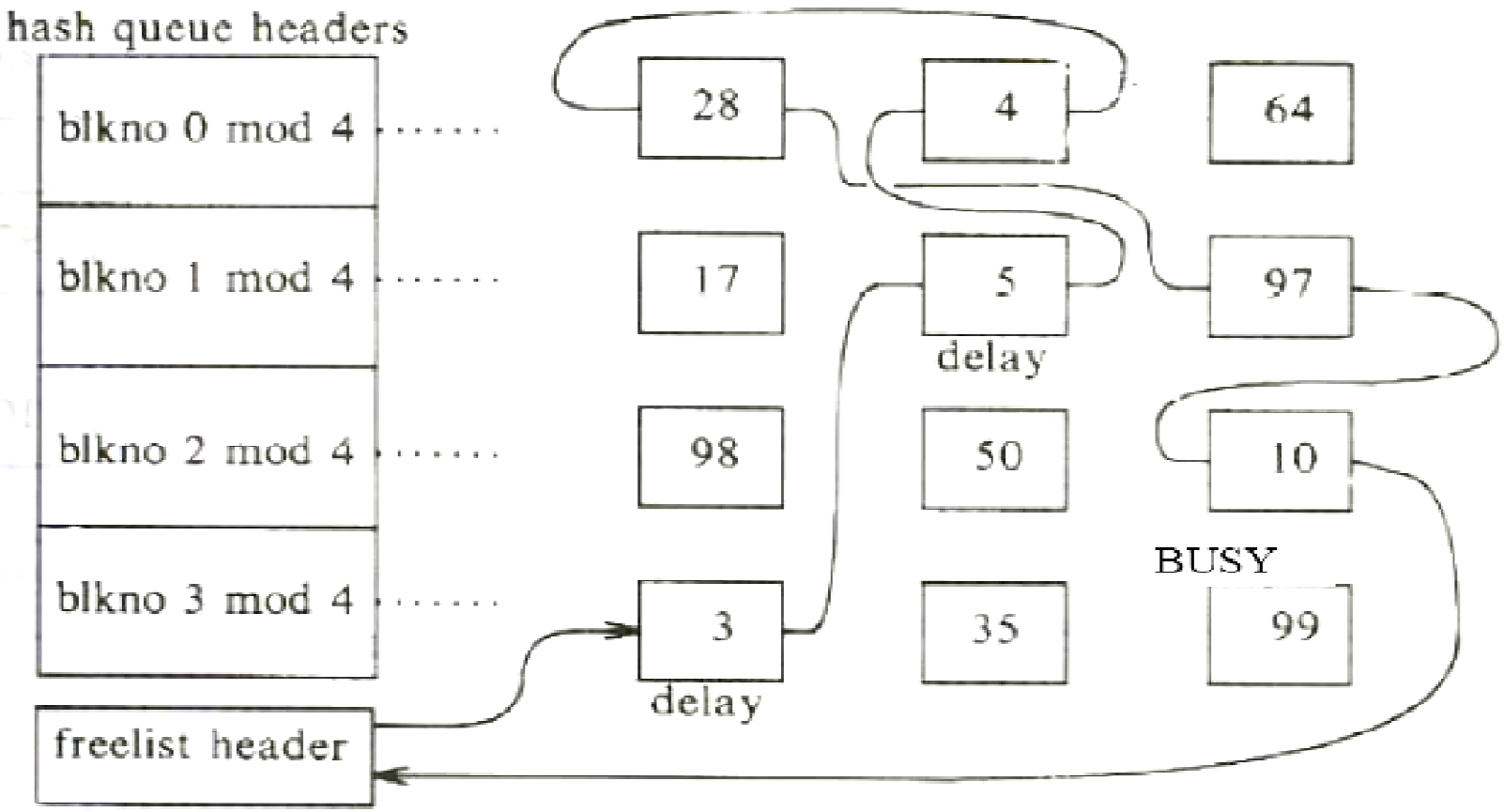
Search for Block 18, Empty Free list

Fourth scenario for Allocating Buffer

```
algorithm getblk
input:  file system number
        block number
output: locked buffer that can now be used for block
{
        while (buffer not found)
        {
                if (block in hash queue)
                {
                        if (buffer busy)        /* scenario 5 */
                        {
                                sleep (event buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        mark buffer busy;       /* scenario 1 */
                        remove buffer from free list;
                        return buffer;
                }
                else        /* block not on hash queue */
                {
                        if (there are no buffers on free list)        /* scenario 4 */
                        {
                                sleep (event any buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        remove buffer from free list;
                        if (buffer marked for delayed write) {      /* scenario 3 */
                                asynchronous write buffer to disk;
                                continue;       /* back to while loop */
                        }
                        /* scenario 2 -- found a free buffer */
                        remove buffer from old hash queue;
                        put buffer onto new hash queue;
                        return buffer;
                }
        }
}
```

Scenarios in getblk to allocate a buffer for a disk block

- The kernel finds the block on its hash queue, and its buffer is free.
- The kernel cannot find the block on the hash queue, so it allocates a buffer from the free list
- The kernel cannot find the block on the hash queue and in attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked “delayed write”. The kernel must write the delayed write buffer to disk and allocate another buffer.
- The kernel cannot find the block on the hash queue, and the free list of buffer is empty.
- The kernel finds the block on the hash queue, but its buffer is currently busy.



Search for Block 99, Block Busy

Fifth scenario for Allocating Buffer

```
algorithm getblk
input:  file system number
        block number
output: locked buffer that can now be used for block
{
        while (buffer not found)
        {
                if (block in hash queue)
                {
                        if (buffer busy)        /* scenario 5 */
                        {
                                sleep (event buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        mark buffer busy;       /* scenario 1 */
                        remove buffer from free list;
                        return buffer;
                }
                else        /* block not on hash queue */
                {
                        if (there are no buffers on free list)        /* scenario 4 */
                        {
                                sleep (event any buffer becomes free);
                                continue;       /* back to while loop */
                        }
                        remove buffer from free list;
                        if (buffer marked for delayed write) {      /* scenario 3 */
                                asynchronous write buffer to disk;
                                continue;       /* back to while loop */
                        }
                        /* scenario 2 -- found a free buffer */
                        remove buffer from old hash queue;
                        put buffer onto new hash queue;
                        return buffer;
                }
        }
}
```

Reading and Writing Disk Blocks

```
Algorithm bread
Input:    file system block number
Output: buffer containing data
{
      get buffer for block (algorithm getblk);
      if (buffer data valid)
            return buffer;
      initiate disk read;
      sleep (event disk read complete);
      return (buffer);
}
```

Algorithm for Reading a Disk Block