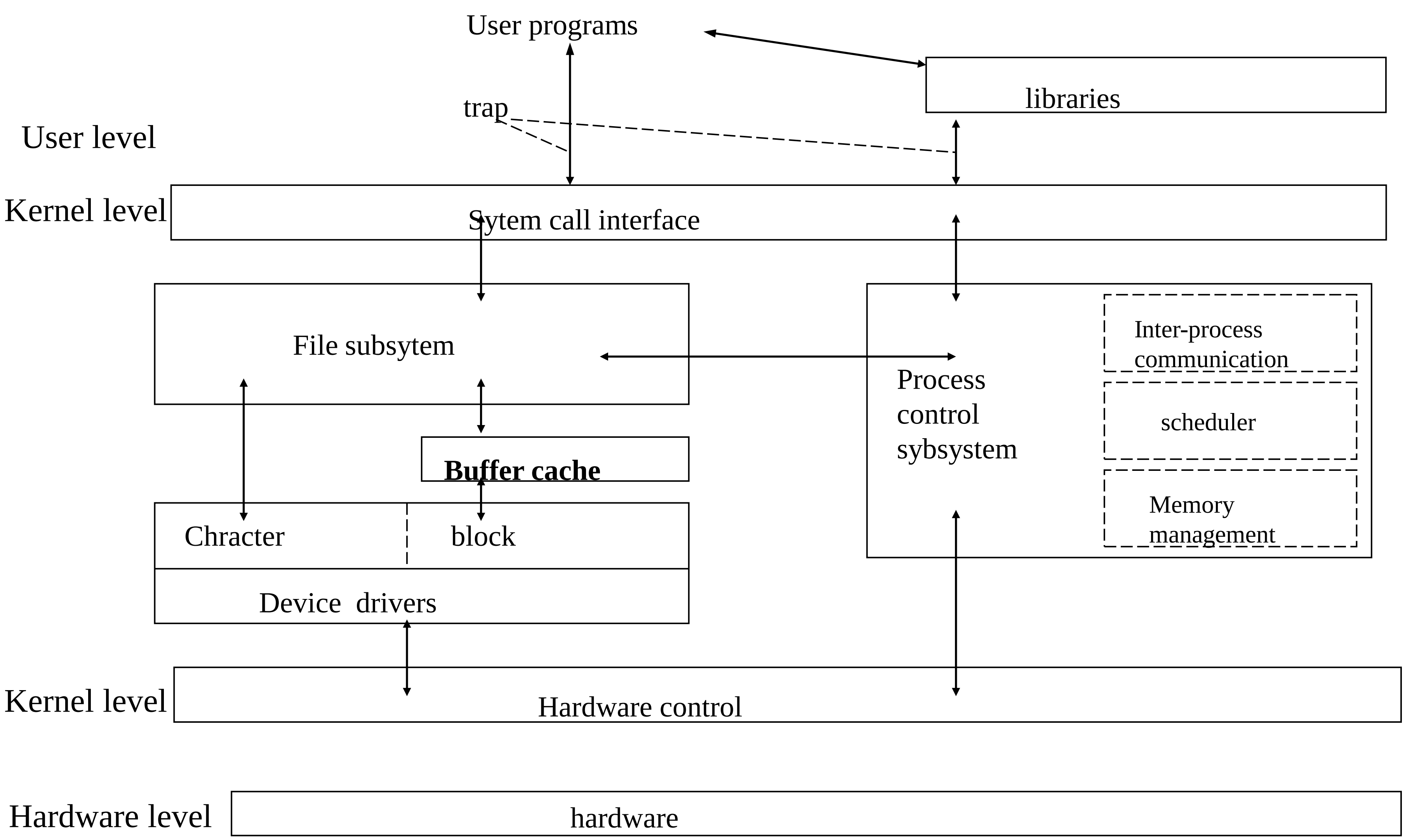
UNIX Kernel Architecture



System Calls

- System call represent the border betwee user program and the kernel
- System call are similar to ordinary functions
- Libraries map these function call to primitives needed to enter OS
- Assembly programs can directly invoke system call without a system call library
- System calls can be categorized into :
  - File subsystem related calls
  - Process control system related calls

- File subsystem related calls (Manage files)
  - Allocating file space
  - Free space management
  - File access control
  - open, close read, write, stat, chown, chmode etc

Buffering

- File subsystem accesses file data using buffering mechanism
- The mechanism regulates the flow between buffer and secondary storage
- The mechanism interacts with block i/o device driver to initiate the data transfer
- block I/O devices are random access storage devices
- Device drivers are kernel module that controls the device operation

Buffer Cache

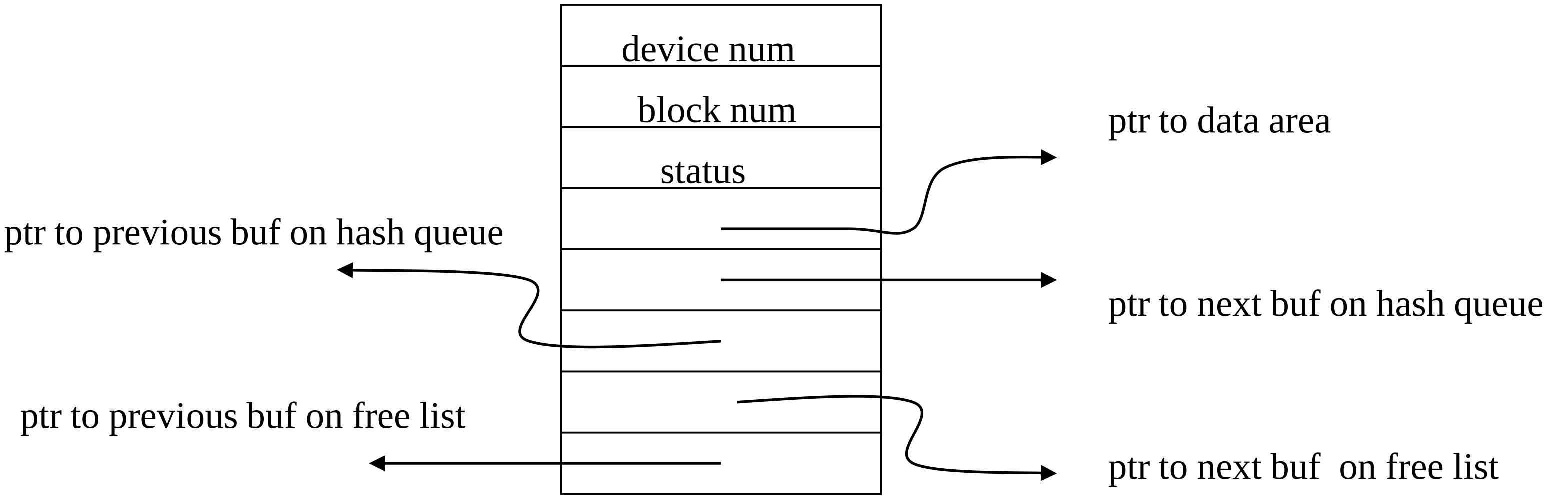
- When a process wants to access data from a file, the kernel brings the data into main memory, alters it and then request to save in the file system
- Buffering improves the response time , throughput
- Buffering minimizes the frequency of disk access by keeping a pool of internal data buffer called buffer cache.

Buffer Cache

- Buffer cache contains the data from recently used disk blocks
- When reading data from disk, the kernel attempts to read from buffer cache.
- If data is already in the buffer cache, the kernel does not read from disk
- If data is not in the buffer cache, the kernel reads the data from disk and cache it

Buffer Headers

- A buffer consists of two parts
  - a memory array
  - buffer header
- disk block : buffer = 1 : 1



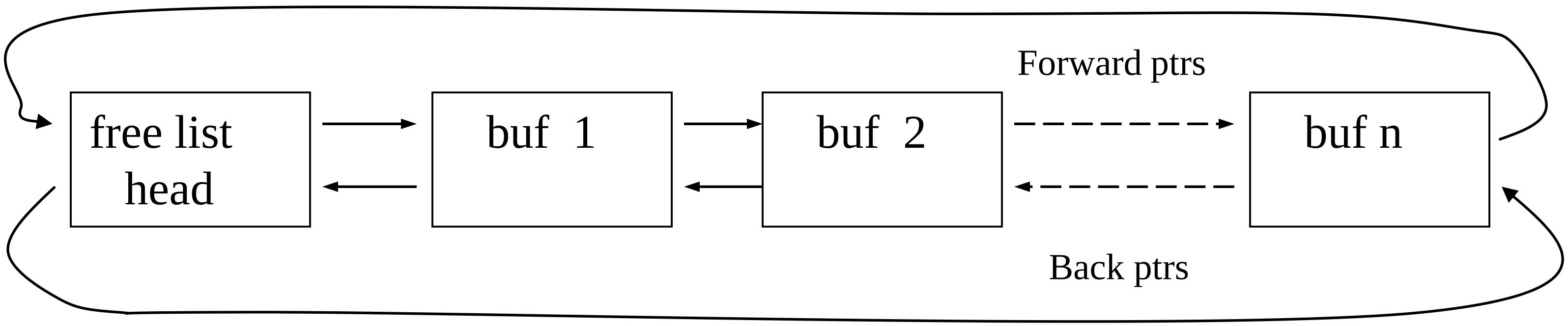
Buffer Header

Buffer Headers

- device num
  - logical file system number
- block num
  - block number of the data on disk
- status
  - The buffer is currently locked/busy.
  - The buffer contains valid data.
  - delayed-write
  - The kernel is currently reading or writing the contents of buffer to disk.
  - A process is currently waiting for the buffer to become free.
- kernel identifies the buffer content by examining device num and block num.

Structures of the buffer pool

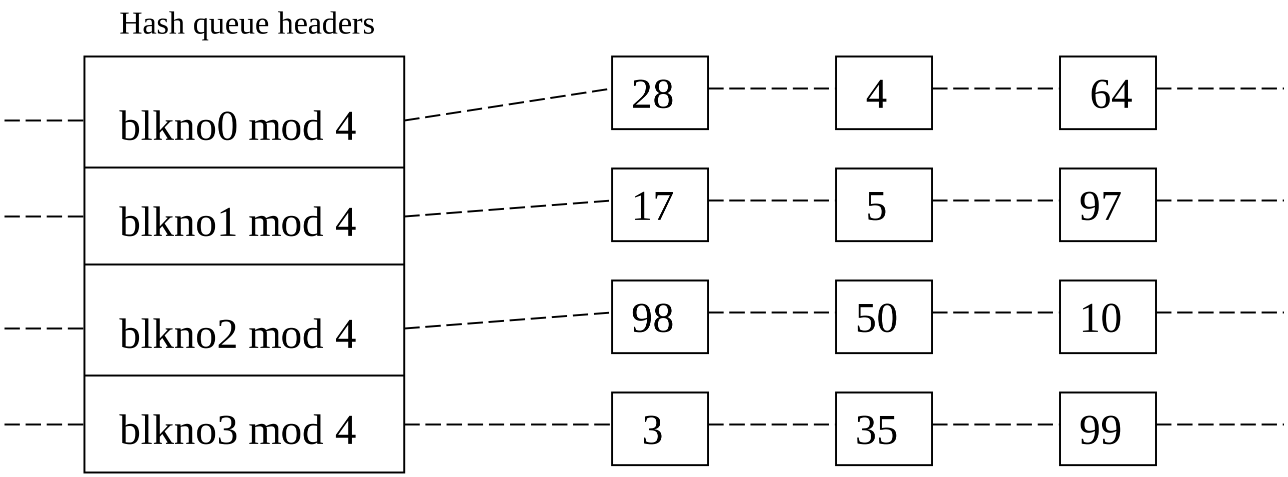
- Buffer pool arranged according to LRU
- The kernel maintains a free list of buffer
  - doubly linked list
  - take a buffer from the head of the free list.
  - When returning a buffer, attaches the buffer to the tail.



Free list of Buffers

Structure of the buffer pool

- When the kernel accesses a disk block
  - Kernel organizes the buffer into separate queue
  - hashed as a function of the device and block num
  - Every disk block exists on one and only on hash queue and only once on the queue



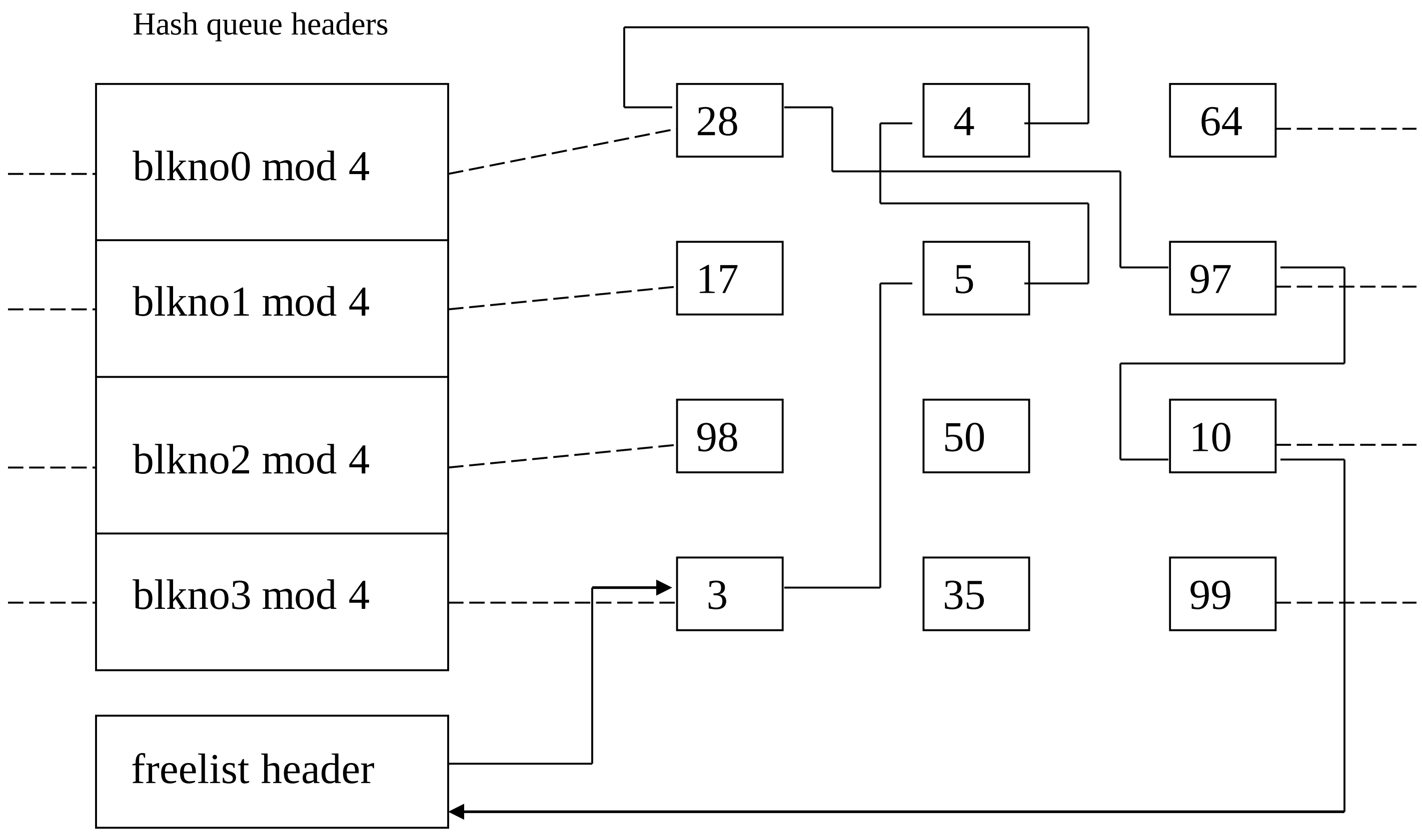
Buffers on the Hash Queues

Scenarios for retrieval of a buffer

- Determine the logical device num and block num
- The algorithms for reading and writing disk blocks use the algorithm *getblk*
  - The kernel finds the block on its hash queue
    - The buffer is free.
    - The buffer is currently busy.
  - The kernel cannot find the block on the hash queue
    - The kernel allocates a buffer from the free list.
    - In attempting to allocate a buffer from the free list, finds a buffer on the free list that has been marked “delayed write”.
    - The free list of buffers is empty.

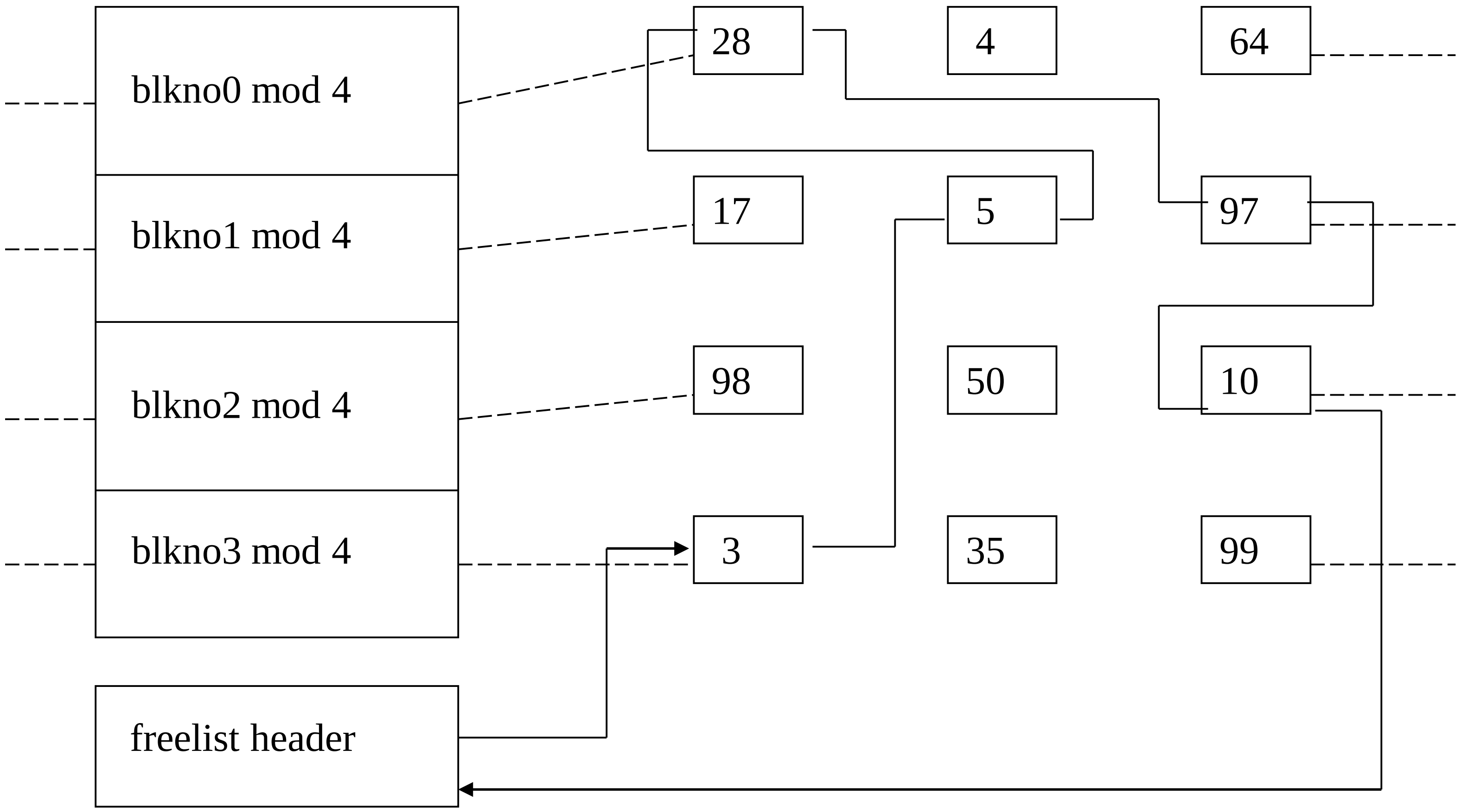
Retrieval of a Buffer: 1^{st} Scenario (a)

- The kernel finds the block on the hash queue and its buffer is free



Search for block 4

Retrieval of a Buffer: 1st Scenario (b)



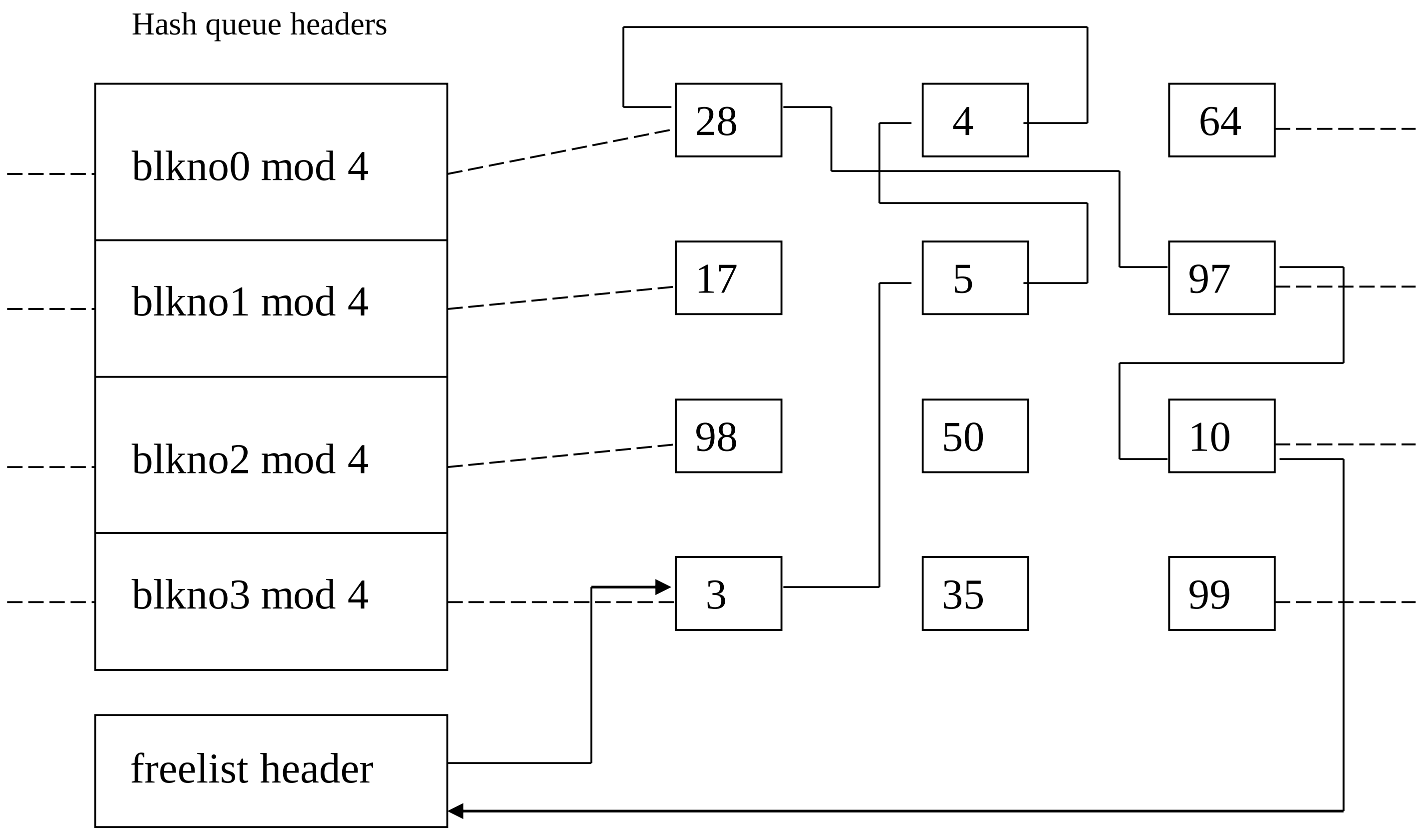
Remove block 4 from free list

Buffer allocation

- getblk algorithm is used for buffer allocation
- Once buffer is allocated kernel can
  - Read data from disk into buffer
  - Manipulate or write data to buffer or disk
  - Mark buffer busy. (no other process can access it and change its content)
- When kernel finishes using buffer , it releases the buffer (brelse) and places it at end of freelist

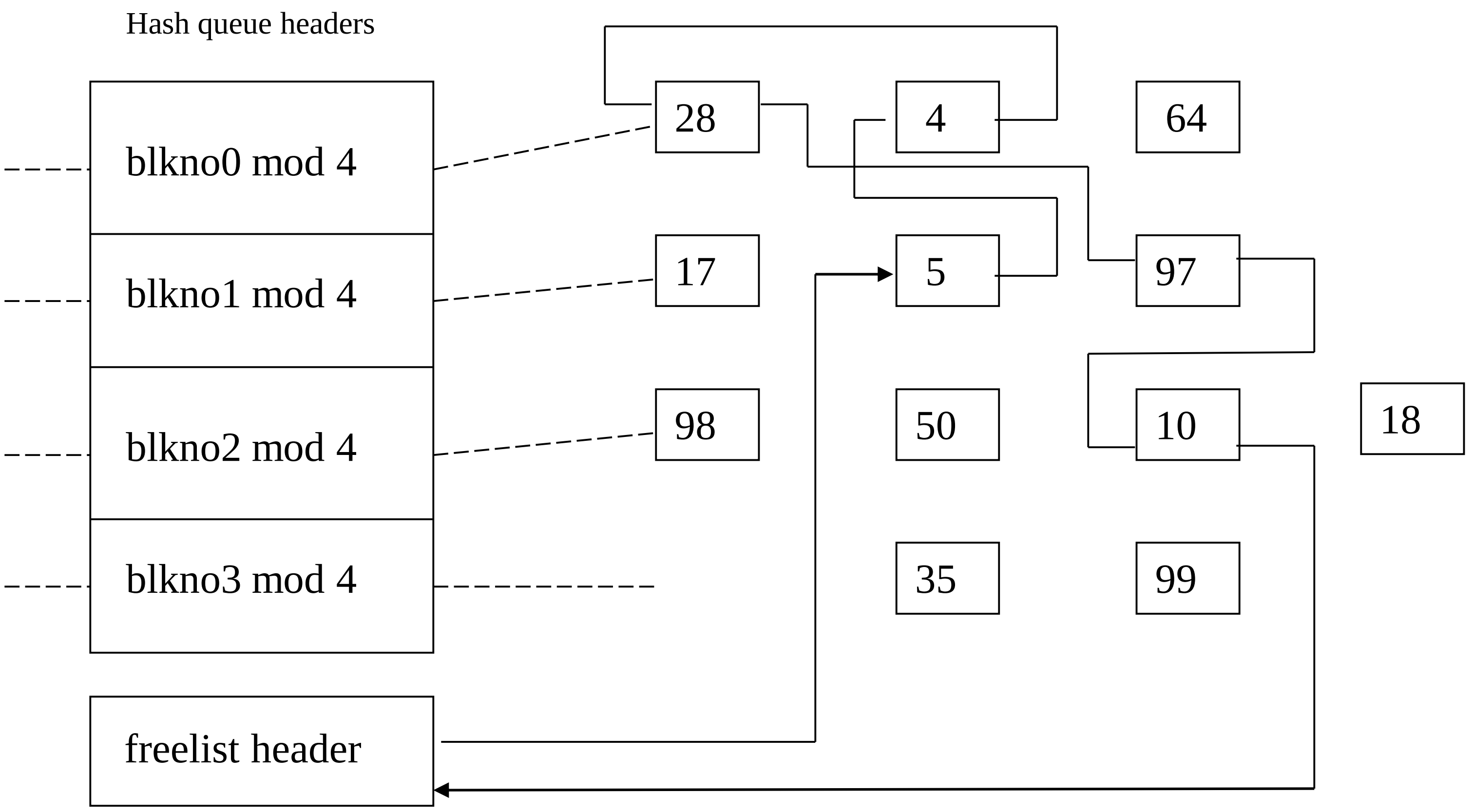
Retrieval of a Buffer: 2nd Scenario (a)

- The kernel cannot find the block on the hash queue, so it allocates a buffer from free list



Search for block 18: Not in cache

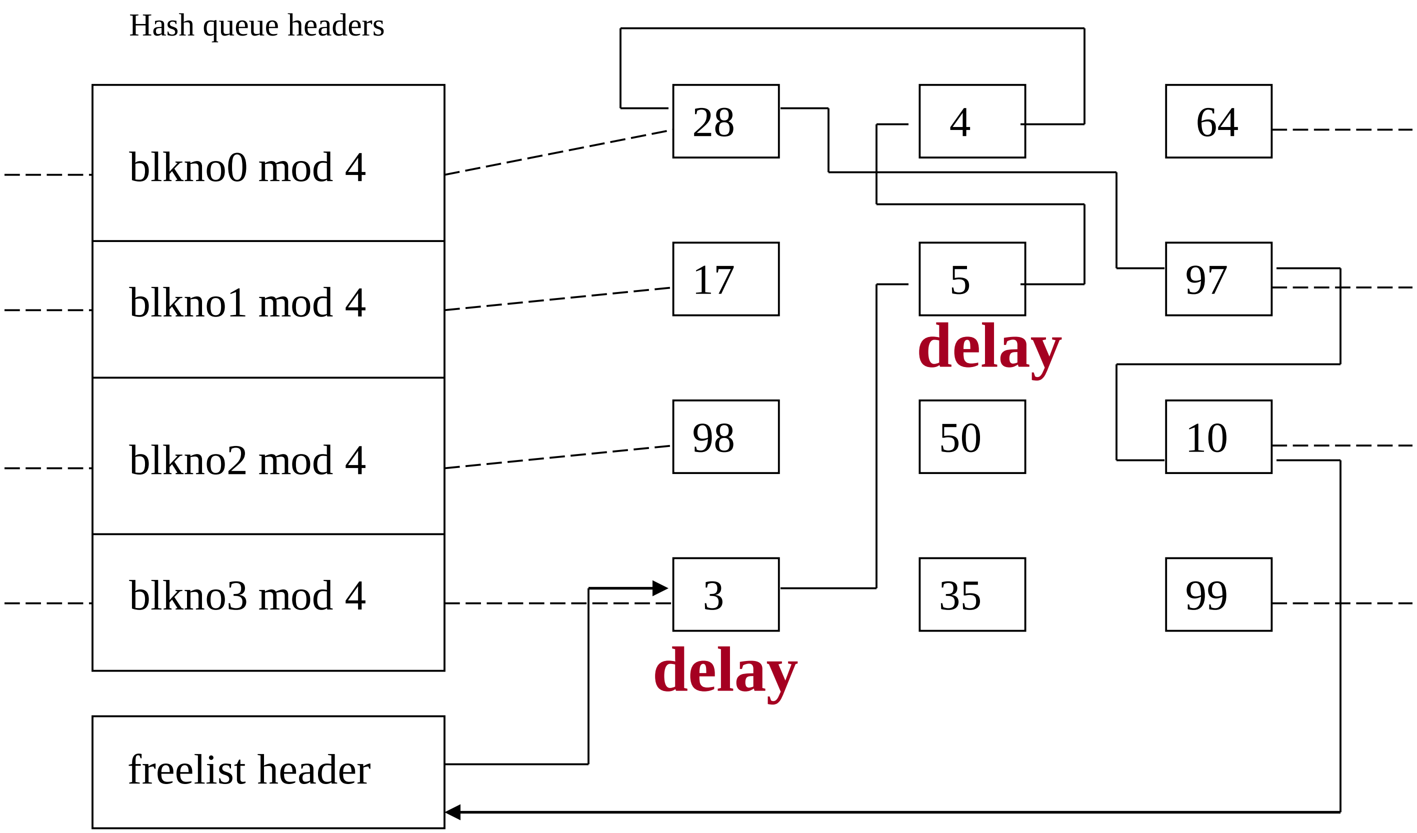
Retrieval of a Buffer: 2nd Scenario (b)



Remove 1st block from free list: Assign to 18

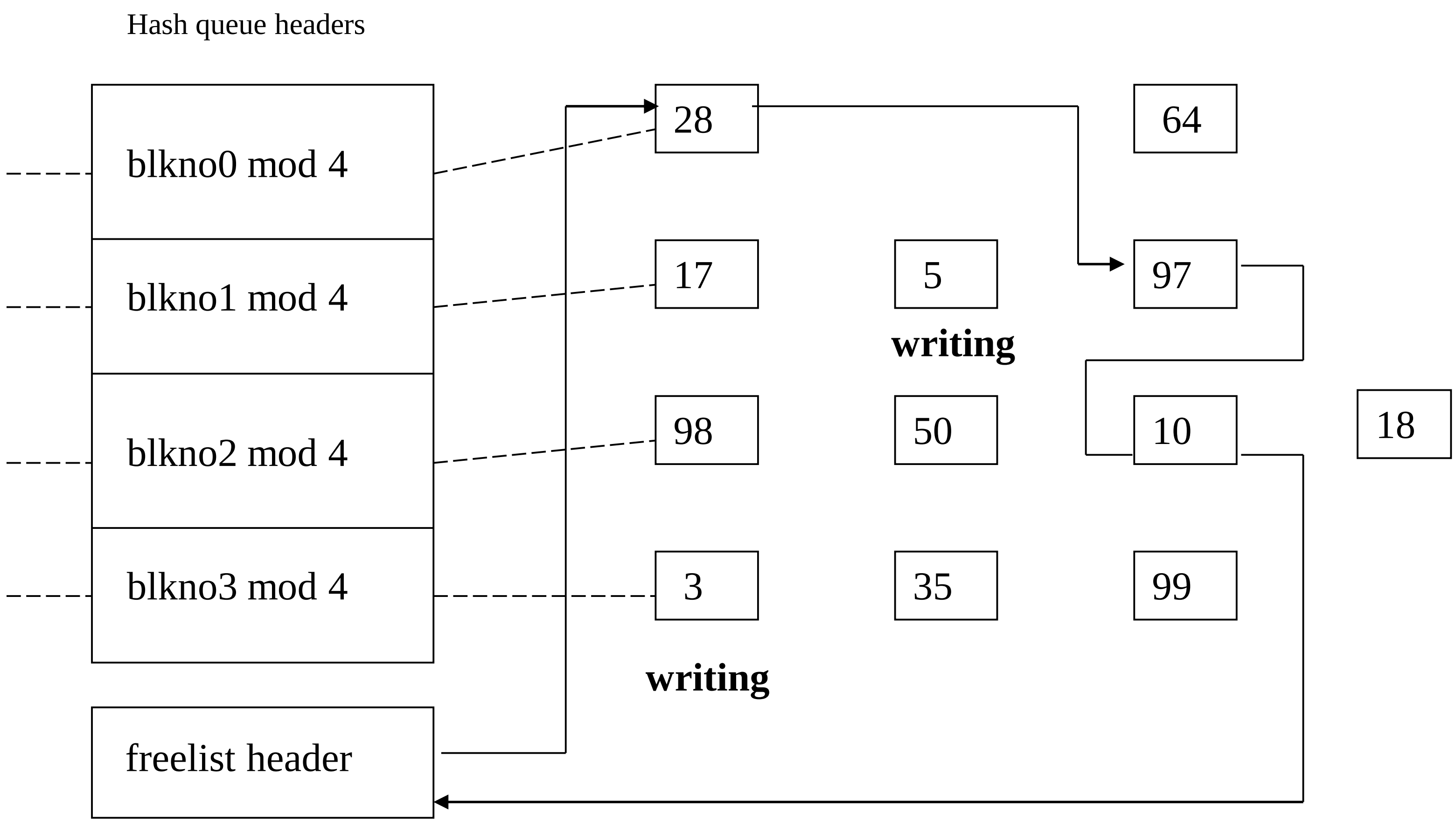
Retrieval of a Buffer: 3rd Scenario (a)

- The kernel cannot find the block on the hash queue, and finds delayed write buffers on hash queue



Search for block 18, Delayed write blocks on free list

Retrieval of a Buffer: 3rd Scenario (b)

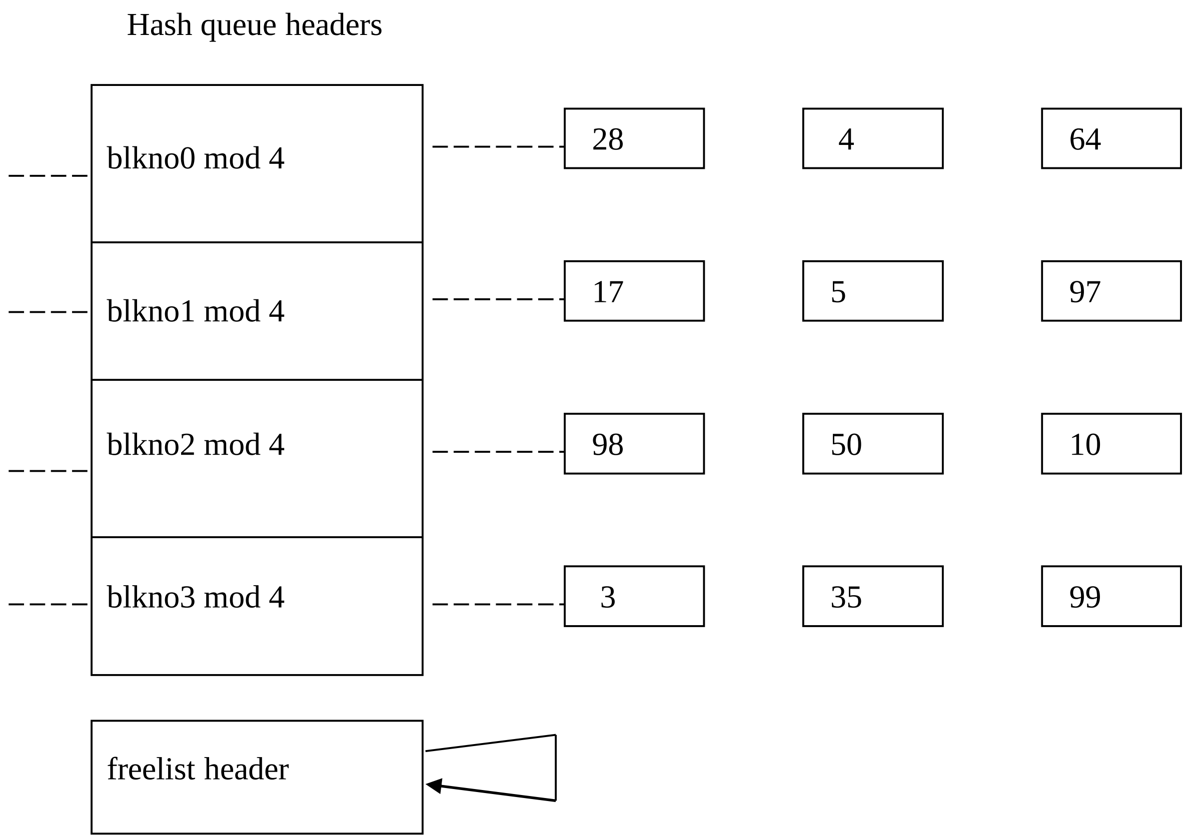


(b) Writing Blocks 3, 5, Reassign 4 to 18

Complete writing and put buffers at the head of free list

Retrieval of a Buffer: 4th Scenario

- The kernel cannot find the buffer on the hash queue, and the free list is empty

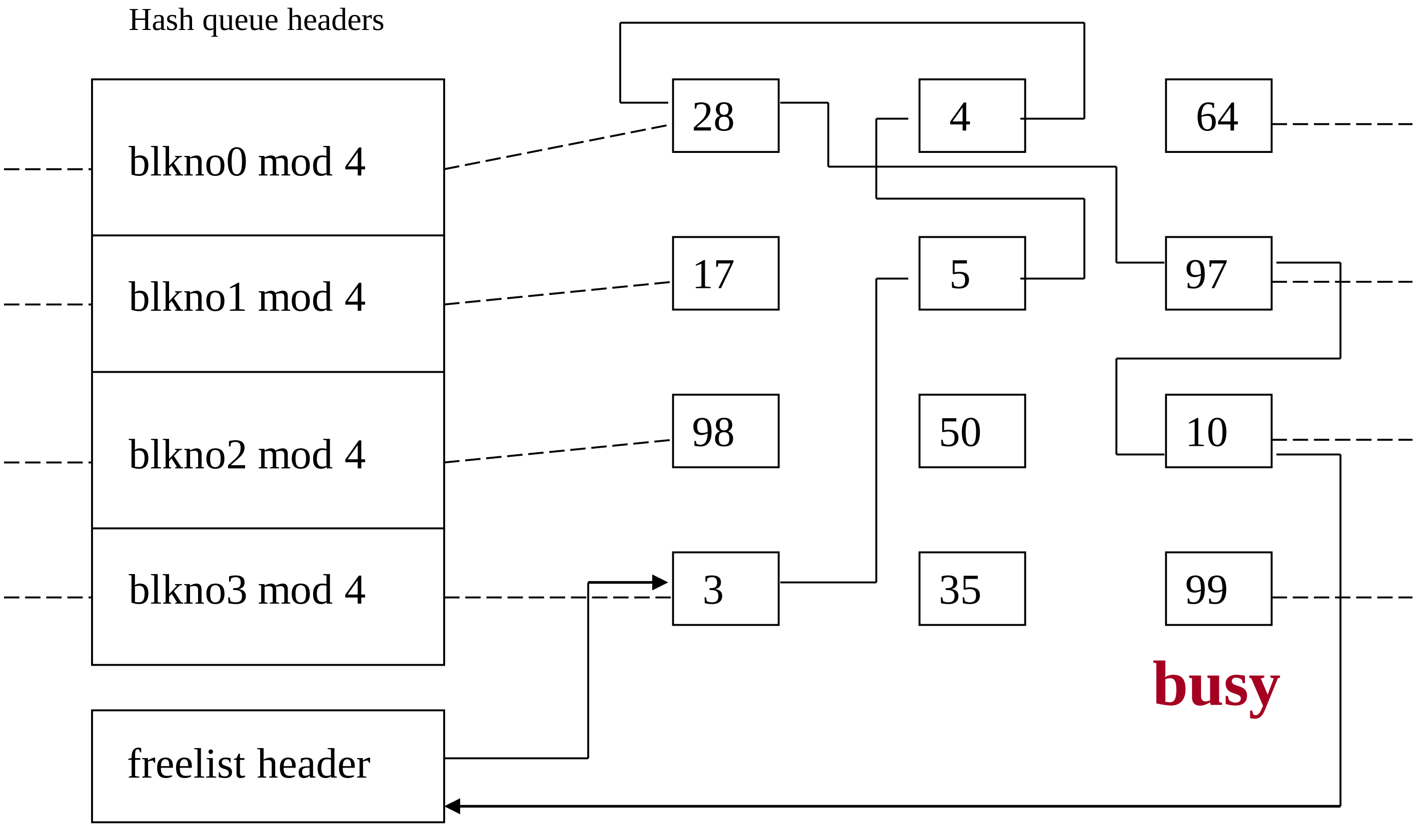


Search for block 18, free list empty

- Process goes to sleep till some other process executes brelse
- When the process is scheduled it must search the hash queue again because some other process might have taken the block already

Retrieval of a Buffer: 5th Scenario

- Kernel finds the buffer on hash queue, but it is currently busy



Search for block 99, block busy

Process A acquires a buffer, marks it busy
Process B needs the same block but find it ‘busy’. So it marks the block as ‘in demand’.

Process A will eventually release the buffer and will find that it was in demand. Process A will issue a wakeup to all processes waiting for the buffer.

B must now check that the buffer is indeed free and no other process has locked it or brought in some other block into it.