Outline

- Overview
- Testing and performance results
- Detailed description of block subsystem

Overview of Filesystem

Overview

- We implemented an inode file system very similar to UNIX system V presented in Bach
  - Block management layer abstracts away block servers and volatile storage
  - Inode and system call interface are unchanged except for error handling
- Buffer cache and asynchronous write and free operations provide performance boosts

Block Functionality

- Allocate persistent or volatile blocks
- Load, release, and free allocated blocks
- Represents volatile blocks as persistent using a thread to refresh 10 seconds before timeout
- Writes and frees are performed asynchronously by another thread for performance increase
- Cache of most recently used blocks prevents a read operation on many get requests

Inode Functionality

- Allocates, loads, releases, and frees inodes
- Provides abstract read/write operations that handle direct and indirect data blocks
- Provides location and path traversal capabilities
- Stores multiple inodes per block

Inode Structure

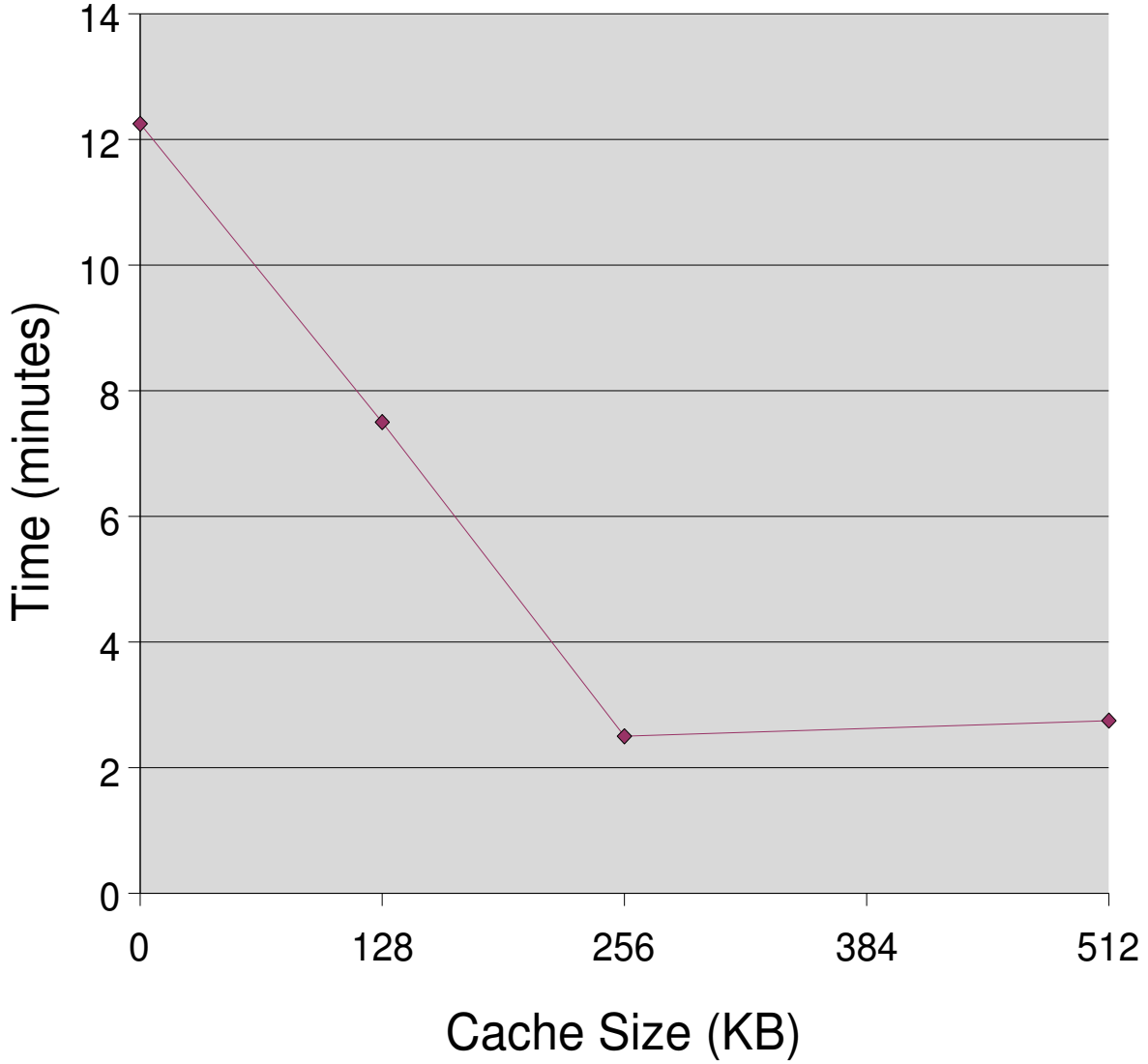
- Common inode fields: owner, group, permissions, access times, link counts, data addresses
- Max file size ~4GB: 9 direct, 1 single indirect and 1 double indirect data pointers
- Inodes identified by a unique triplet: server address, block id, and offset into block

Performance Results

Initial Test Scripts

- Ran all test cases successfully except 10 (which timed out during final testing)
- Notably good performance on 6 due to cache performance improvements on path lookup

Cache Benchmarking



Benchmark Code

```
for i = 1 to 8
        mkdir “$i”
        for j = 1 to 8
                mkdir “$/$j”
        end
end
find .
rm -rf *
```

Final Demo Results

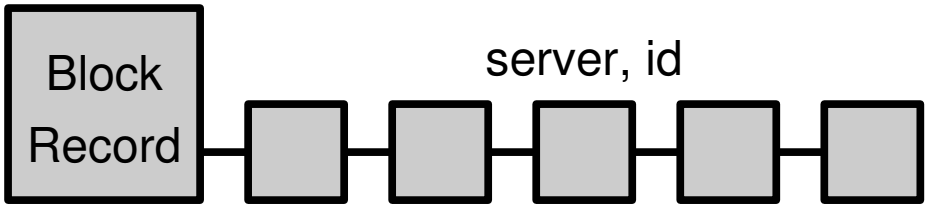
- Successful test cases: 1, 2, 3, 4, 5, 8, 9, 10
- Failed test cases: 6, 7 (path parsing problem)
- Test 9 ran to completion twice due to performance improvement of cache

Detailed Description of Block Subsystem

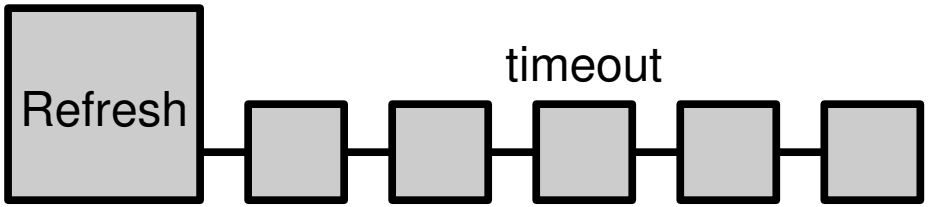
caching, asynchronous

operations, and refreshing

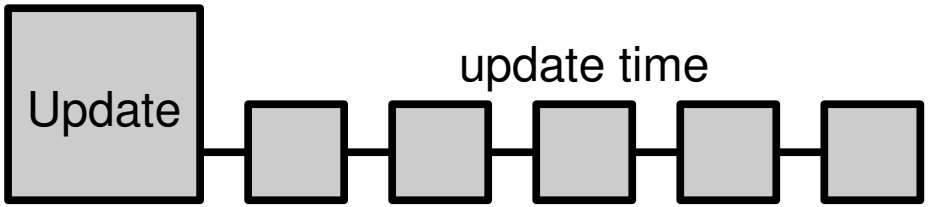
Block Subsystem: Data Structures



- Stores an entry for every allocated block on the servers
- Sorted by the server, id tuple

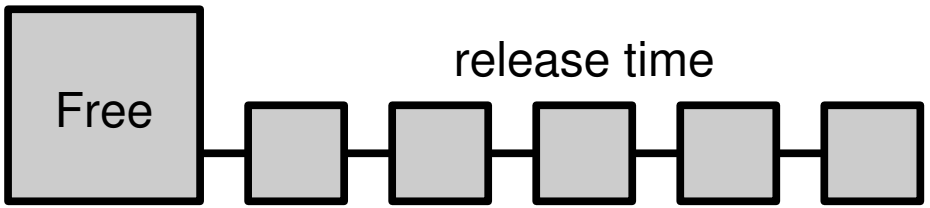


- Stores an entry for every allocated block on the volatile servers
- Sorted by the timeout on the block



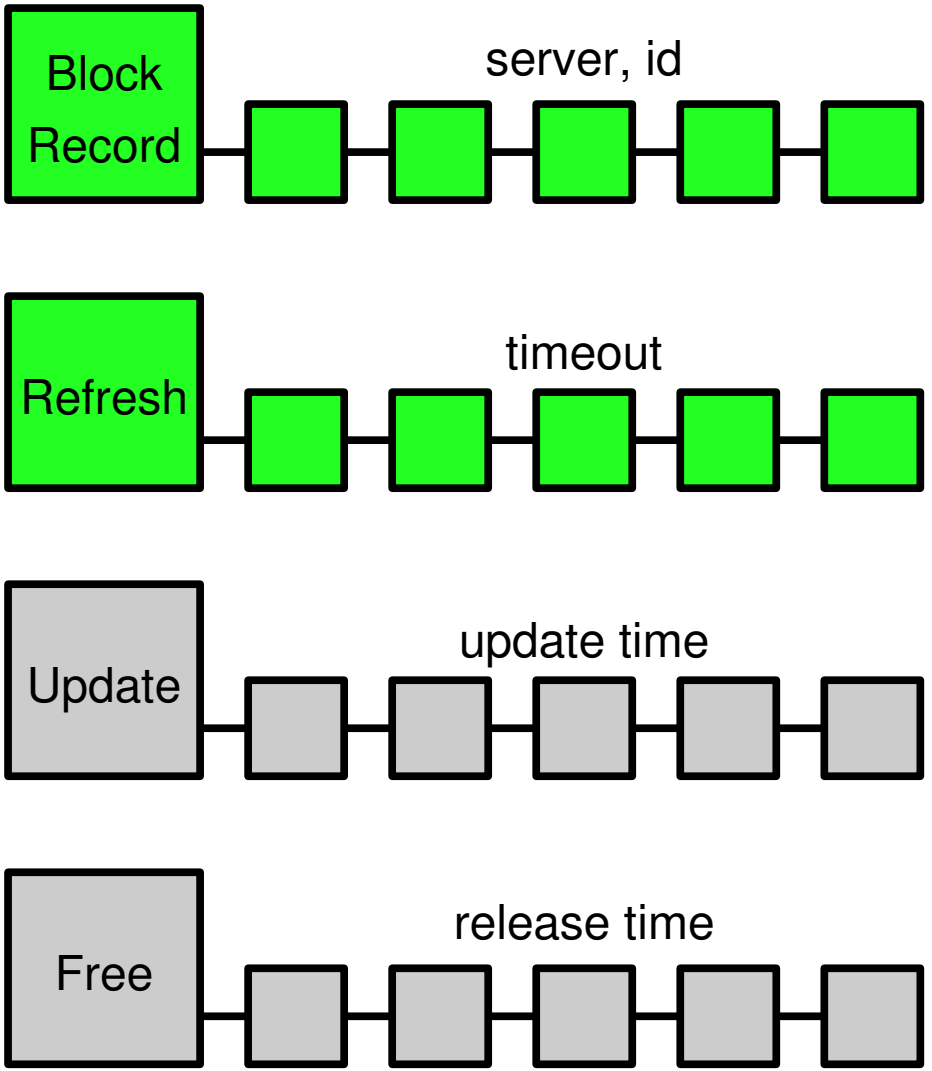
- Keeps an LRU list of the blocks that need asynchronous writes or frees

---



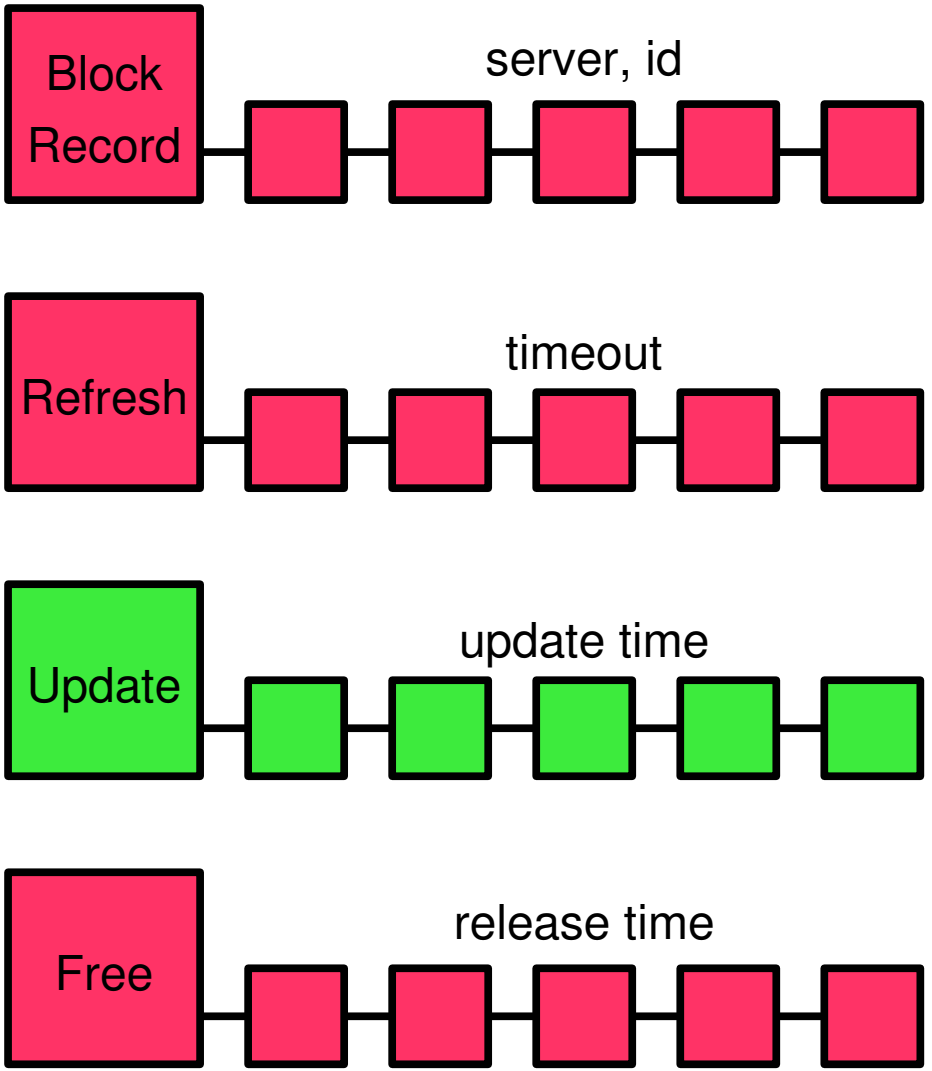
- Keeps an LRU list of the blocks that are not required to be in core
- Size limited by the total number of blocks in core

Block Subsystem: balloc()



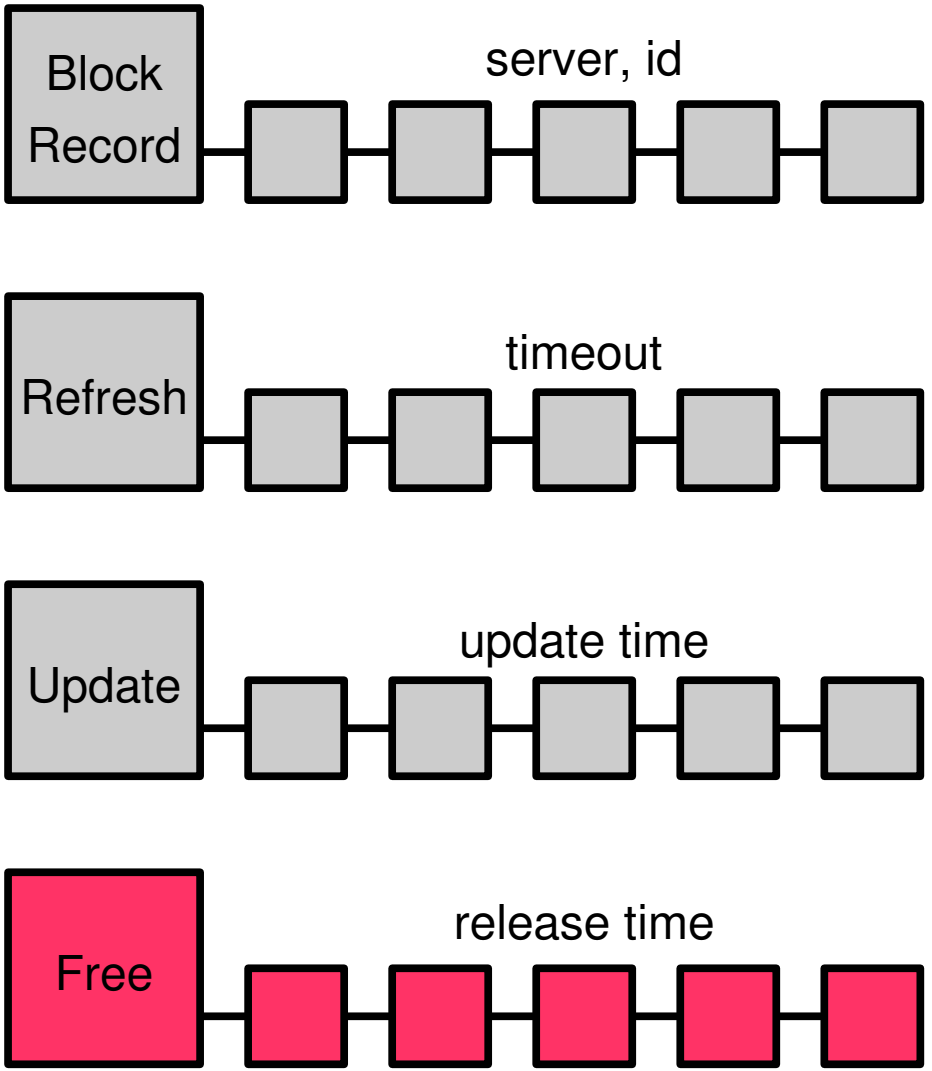
- Adds records for all blocks
- Adds volatile blocks to a refresh list

Block Subsystem: bfree()



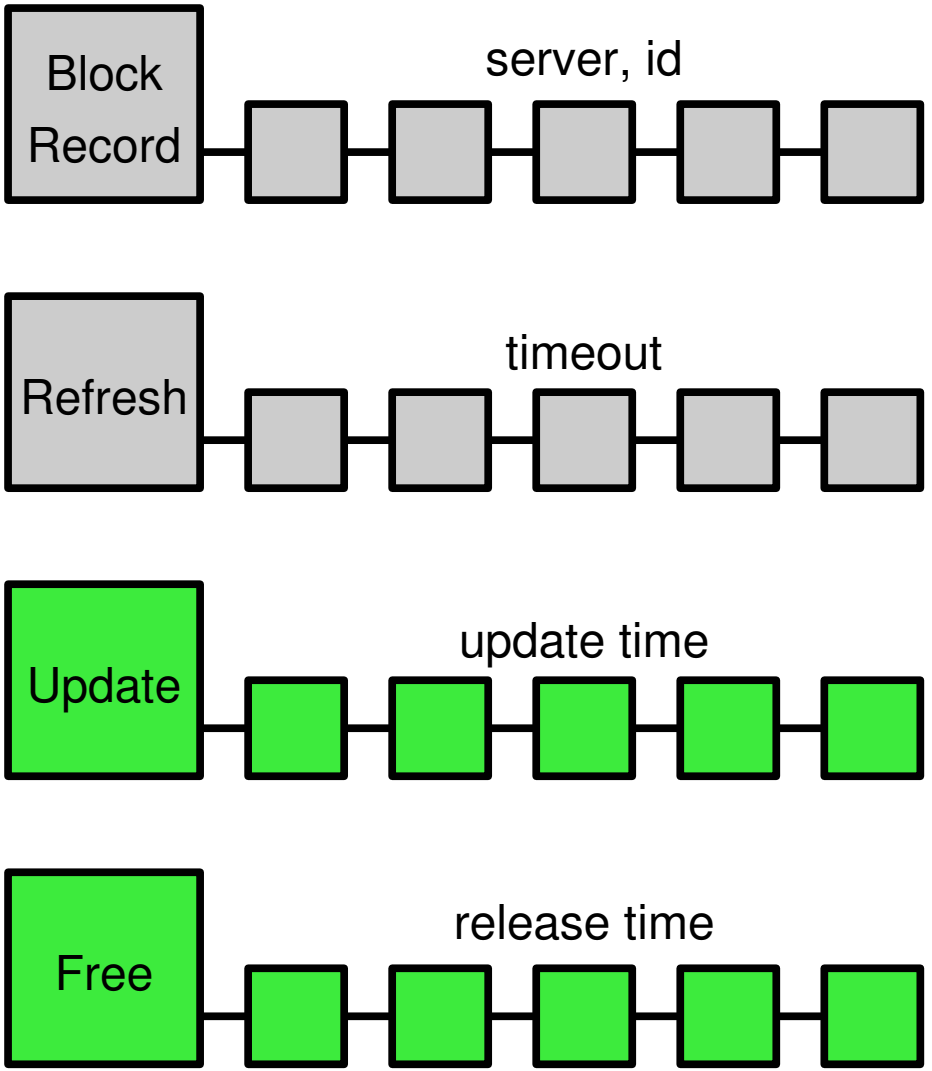
- Removes the record of a block as well as the refresh and free list if necessary
- Adds items to the update list to be freed asynchronously

Block Subsystem: bget()



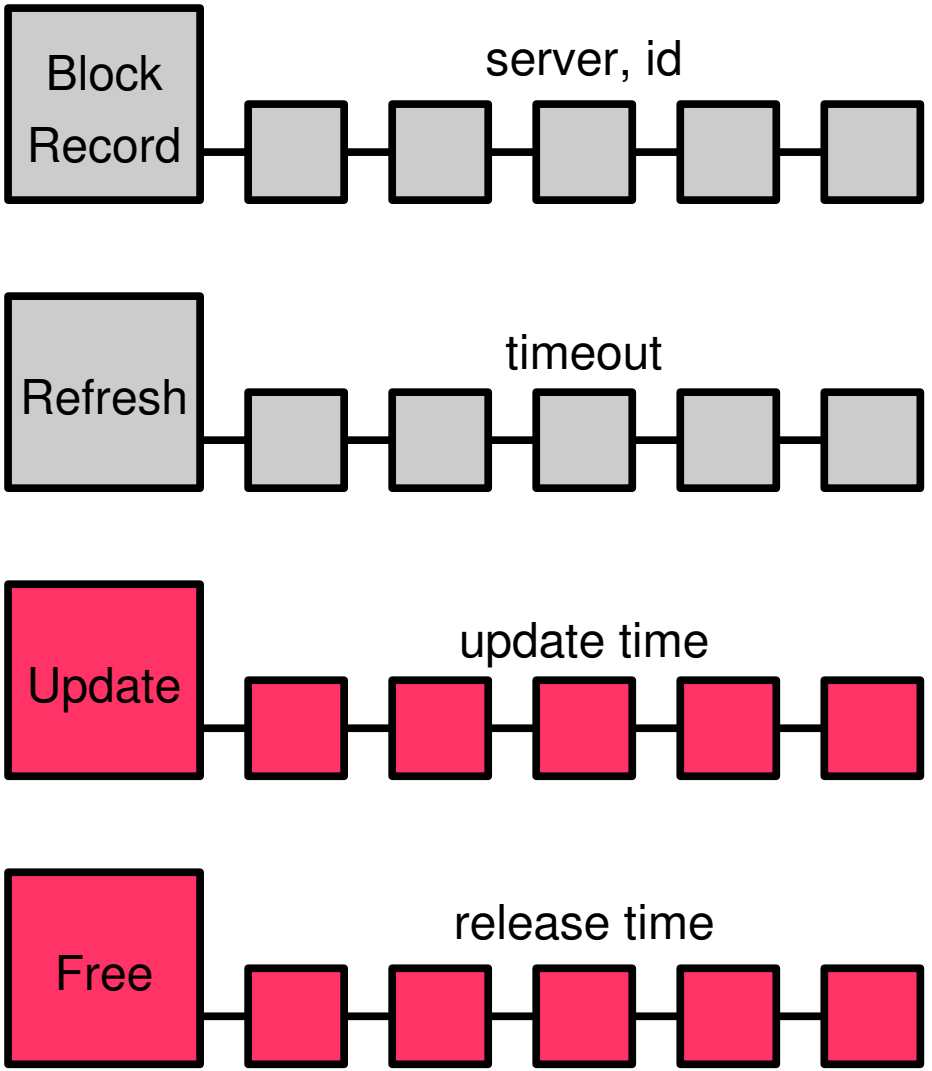
- Removes entries from the free list if there is no space in the cache

Block Subsystem: brelease()



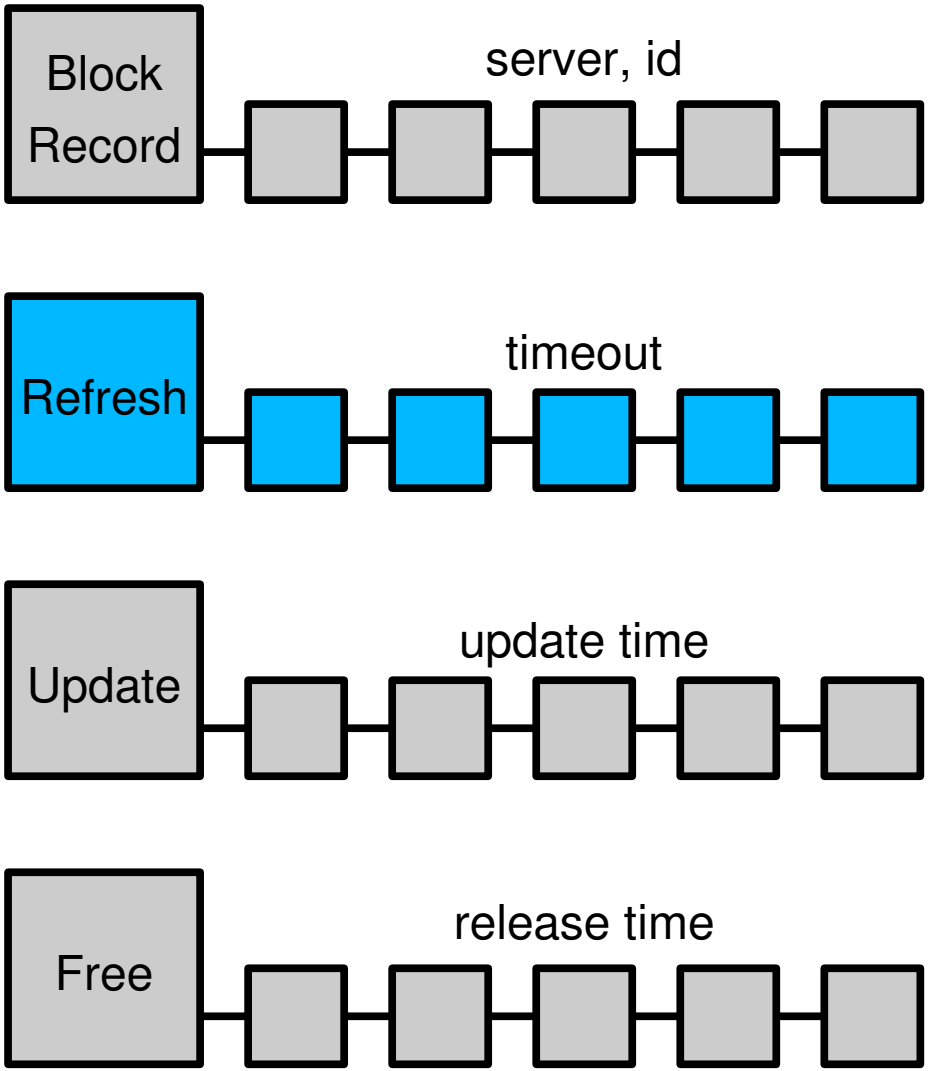
- Adds entries to the update list if they are dirty
- Adds or updates items in the free list to reflect new release time

Block Subsystem: update thread



- Sleeps until items are added to Update List
- Removes entries from the update list after processing
- Removes entries from the free list if cache size exceeded

Block Subsystem: refresh thread



- Sleeps until items are added to Refresh List, or items are about to expire
- Updates entries in the refresh list after processing