# ByteBuffers and Off-Heap Memory

# 64 Bits is BIG

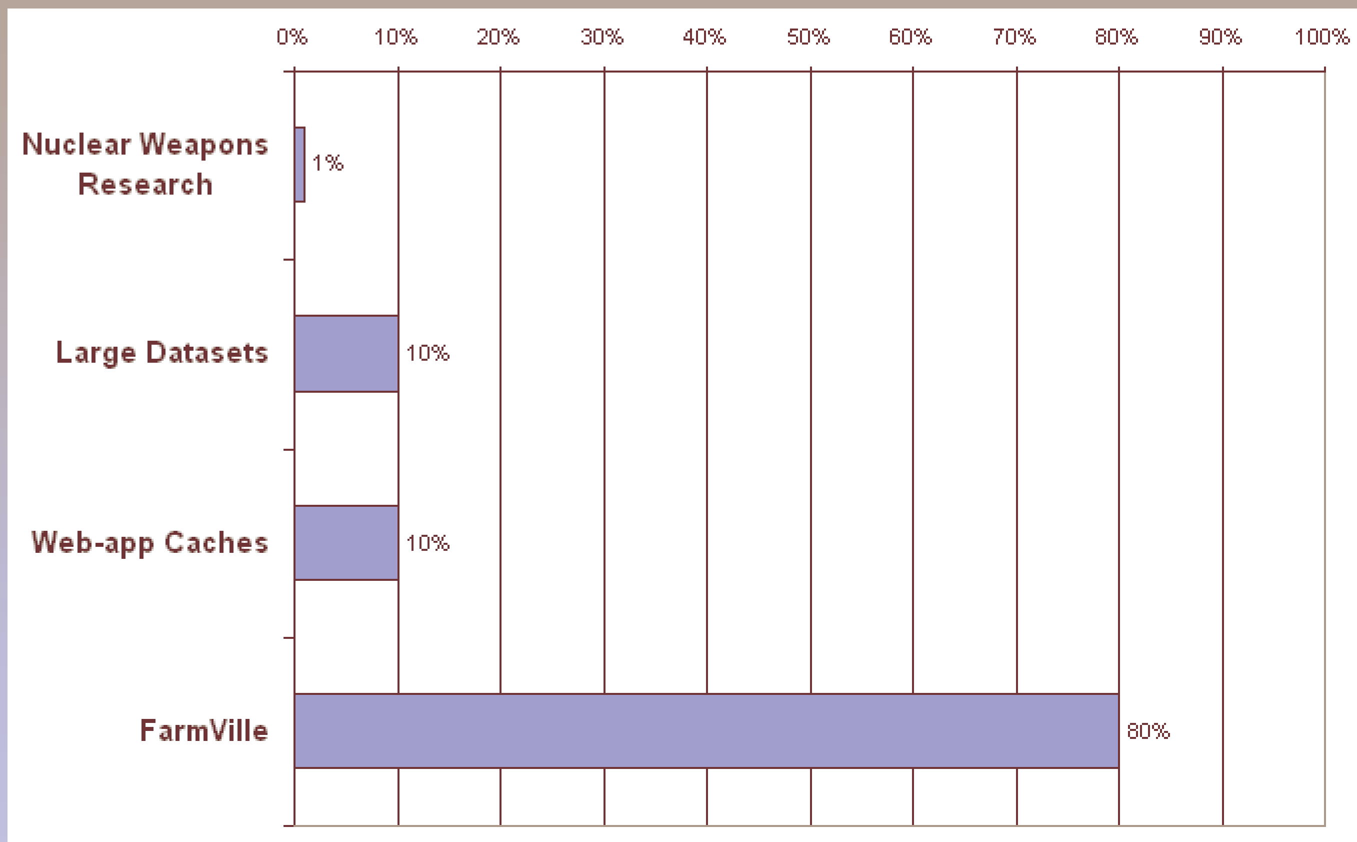
96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **16 exabytes *aka* 16 billion gigabytes**
  - Not big enough to individually address all the atoms in the universe
- **Current processors can’t actually address it all**
  - And operating systems impose their own limits
- **64-bit machines are ubiquitous**
  - 2012 Consumer PC: 8 Gb main memory, < $500
  - 1977: Cray-1: 64 bit data bus, 24-bit address bus, 8 Mb main memory, $8.8MM

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# InfoGraphic: What Are We Doing With All This Memory?

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

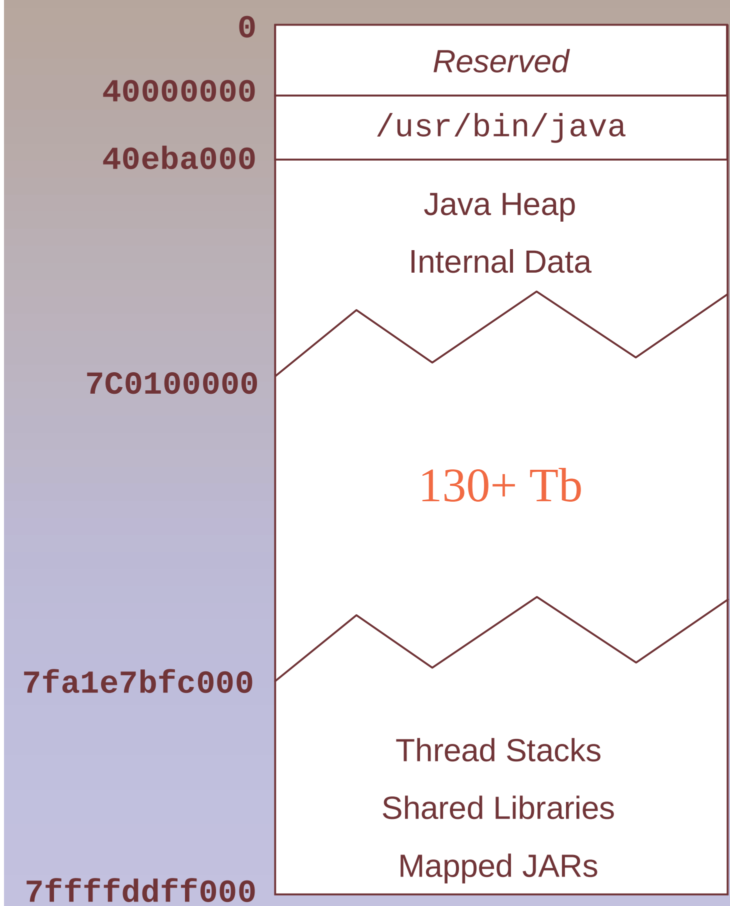



8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# The JVM Memory Map

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

**java -Xms1024m -Xmx4096m com.example.Hello**




```
0000000040000000     36K r-x--  /usr/local/java/jdk-1.6-x64/bin/java
0000000040108000      8K rwx--  /usr/local/java/jdk-1.6-x64/bin/java
0000000040eba000    676K rwx--    [ anon ]
00000006fae00000  21248K rwx--    [ anon ]
00000006fc2c0000  62720K rwx--    [ anon ]
0000000700000000 699072K rwx--    [ anon ]
000000072aab0000 2097152K rwx--    [ anon ]
00000007aaab0000 349504K rwx--    [ anon ]
00000007c0000000 1048576K rwx--    [ anon ]
00007fa1e7bfc000      4K -----    [ anon ]
00007fa1e7bfd000   1024K rwx--    [ anon ]
00007fa1e7cfd000     12K -----    [ anon ]
00007fa1e7d00000   1016K rwx--    [ anon ]
00007fa1e7dfe000     12K -----    [ anon ]
...
00007fa1ed00d000   1652K r-xs-  /usr/local/java/jdk-1.6-x64/jre/lib/rt.jar
...
00007fa1f34aa000   1576K r-x--  /lib/x86_64-linux-gnu/libc-2.13.so
00007fa1f3634000   2044K -----  /lib/x86_64-linux-gnu/libc-2.13.so
00007fa1f3833000     16K r-x--  /lib/x86_64-linux-gnu/libc-2.13.so
00007fa1f3837000      4K rwx--  /lib/x86_64-linux-gnu/libc-2.13.so
...
00007fa1f3e80000      4K r-x--  /lib/x86_64-linux-gnu/ld-2.13.so
00007fa1f3e81000      8K rwx--  /lib/x86_64-linux-gnu/ld-2.13.so
00007ffffdc5b000    132K rwx--    [ stack ]
00007ffffddff000      4K r-x--    [ anon ]
ffffffffff600000      4K r-x--    [ anon ]
```

total 4478020K

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# java.nio.ByteBuffer

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Part of the JDK since 2002 (1.4)**
  - Channel selectors got all the press
- **Three flavors**
  - non-direct (on-heap)
  - direct (off heap)
  - mapped (off heap, contents backed by file)
- **References an unstructured block of memory**
  - No pointers, must use indexes
- **Limited to 2Gb each**
- **Not thread-safe**

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Creating/Using a ByteBuffer



```
byte[] data = new byte[1024];
ByteBuffer buf1 = ByteBuffer.wrap(data);

ByteBuffer buf2 = ByteBuffer.allocate(1024);

ByteBuffer buf3 = ByteBuffer.allocateDirect(1024);

buf1.position(12);
buf1.putInt(0x12345678);
int x = buf1.getInt();

int y = buf1.getInt(12);
buf1.putInt(12, x + 231);
```

- On-heap (buf1, buf2)
- Off-heap (allocateDirect)
- Relative positioning (buf1.putInt(0x12345678))
- Absolute positioning (buf1.getInt(12))

# Practical Use: Off-Heap Web-App Cache

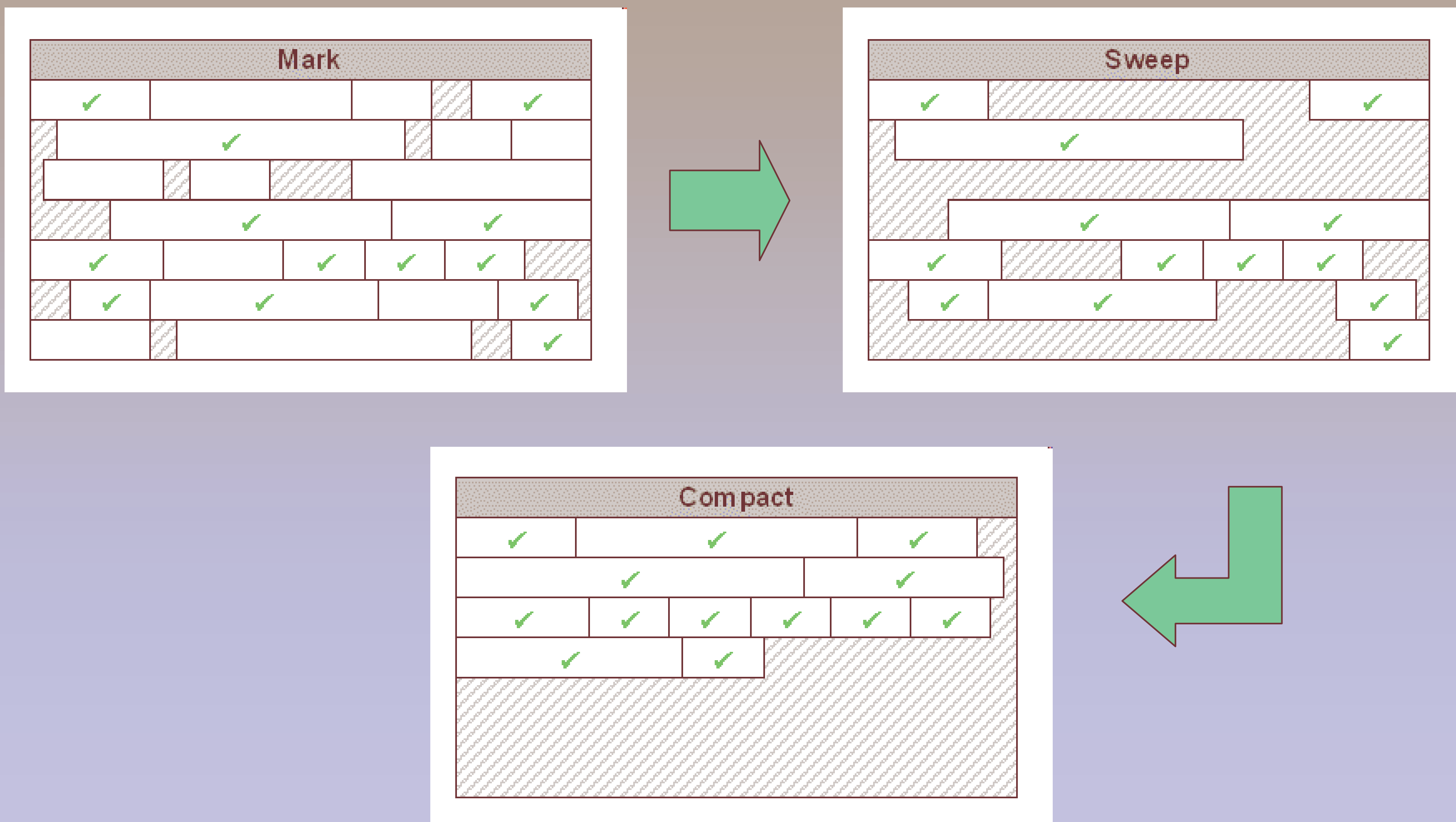
96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Cache stands between response body and database**
  - Entire pages
  - Pieces of pages
  - Data used to construct pages
- **Options**
  - External (Akamai, mod_cache)
  - Internal (servlet filter, map in application scope)
  - Distributed (Memcached, Terracotta, Coherence)
- **What we want**
  - In-process, to minimize cost of access
  - Off-heap, to minimize impact on garbage collector

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# How the Garbage Collector Works

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D




8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# The Problem with Internal Caches

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Big Heap == Slow GC**
  - “Mark” phase increased by number of live objects
  - “Compact” phase increased by holes, size of remaining objects
  - If heap is almost full, may get into cycle of constant GC
- **Paging is BAD**
  - And you can’t control it

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Solution: Off-Heap Cache

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Heap just holds operational data**
  - Can be much smaller
  - More frequent collections, but less gets collected
- **Garbage collector doesn't touch cache**
  - If data gets paged to disk, it stays there until needed
- **Cache can be larger than physical memory**

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Issues in Implementing an Off-Heap Cache

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- Accessing Non-Heap Memory
- Memory Management / Fragmentation
- Managing Cache
- Marshalling/Unmarshalling Cached Data

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Memory Management

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Explicitly assign/release memory**
- **Manage the freelist**
- **Prevent fragmentation**

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Constructing the Cache

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

```
public class OffHeapStringCache
{
    private int _maxLength;
    private LinkedHashMap<String,CharBuffer> _map;
    private LinkedList<CharBuffer> _freeList;


    public OffHeapStringCache(int size, int maxLength)
    {
        _maxLength = maxLength;


        _map = new LinkedHashMap<String,CharBuffer>(size, .75f, true);


        _freeList = new LinkedList<CharBuffer>();
        for (int ii = 0 ; ii < size ; ii++)
        {
            CharBuffer buf = ByteBuffer.allocateDirect(2 * maxLength).asCharBuffer();
            _freeList.add(buf);
        }
    }
```

Max size of contained strings

Ensures LRU behavior

`LinkedList` is fast for head inserts/removes

Direct allocation requires `ByteBuffer`

But we want to access as characters

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Managing Cached Data

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **A cache acts like a Map**
- **But has fixed size**
- **And an expiration (eviction) strategy**
  - Simple strategy: least recently used
  - More complex: least frequently used
  - Even more complex: least cost to recreate

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Store

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

```
public synchronized void put(String key, String value)
{
    if (value.length() > _maxLength)
        throw new IllegalArgumentException("string too long: " + value.length());

    CharBuffer buf = _map.get(key);

    if (buf == null)
        buf = (_freeList.size() > 0) ? _freeList.removeFirst() : null;

    if (buf == null)
    {
        Entry<String,CharBuffer> eldest = _map.entrySet().iterator().next();
        buf = eldest.getValue();
        _map.remove(eldest.getKey());
    }


    buf.clear();
    buf.put(value);
    buf.limit(value.length());


     _map.put(key, buf);
}
```

Don’t want to leak buffers!

Map iterates in LRU order

This just resets the buffer’s position/limit

Must explicitly restrict the “active” portion of the buffer

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Retrieve

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

```
public synchronized String get(String key)
{
    CharBuffer buf = _map.get(key);
    if (buf == null)
        return null;

    buf.position(0);
    return buf.toString();
}
```

Position will be updated by any read/write, so must be reset each time

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Reconstituting Cached Data

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Simply copying data from cache to heap negates benefit of off-heap cache**
  - Large arrays are stored directly in tenured generation
  - But it will be almost immediately eligible for collection
- **Solution: move directly from cache to output**
  - `void write(String key, Writer out)`
  - Alternative: return `InputStream` that wraps buffer
- **Think about scope of synchronization!**

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Writing Directly to Output

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

This will lock the cache for a (relatively) long time!

```
public synchronized boolean write(String key, Writer writer)
throws IOException
{
    CharBuffer buf = _map.get(key);
    if ((buf == null) || (buf.limit() == 0))
        return false;

    buf.position(0);
    for (int ii = 0 ; ii < buf.limit() ; ii++)
        writer.write(buf.get(ii));

    return true;
}
```

Should get a performance boost from a small `char[]`

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Practical Use: Memory-Mapped Files

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **File's contents are mapped into process address space**
  - Once page is loaded, all access is in-process
- **Operating System loads/writes pages as needed**
  - Unlike RandomAccessFile, which requires context switch
- **Most useful when you access relatively small areas of the file repeatedly**
  - Especially if file is large vis-à-vis available RAM

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Example: Memory-Mapped JARs

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

```
…
a0990000 1632K r-xs-  /usr/local/java/jdk-1.6/jre/lib/rt.jar
…
b6d08000    8K r-xs-  /usr/local/nexus-oss-webapp-1.9.2.4/runtime/apps/nexus/lib/aether-spi-1.8.1.jar
b6d0a000   28K r-xs-  /usr/local/nexus-oss-webapp-1.9.2.4/runtime/apps/nexus/lib/log4j-1.2.14.jar
b6d11000    4K r-xs-  /usr/local/nexus-oss-webapp-1.9.2.4/runtime/apps/nexus/lib/plexus-slf4j-logging-1.1.jar
b6d12000    8K r-xs-  /usr/local/nexus-oss-webapp-1.9.2.4/runtime/apps/nexus/lib/plexus-task-scheduler-1.4.2.jar
…
```

- **JAR directory is at end of file**
  - Directory references individual entries by offset
  - Fast to copy entry data into array, pass to classloader
- **Alternative: read file from start to finish**

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Using Memory-Mapped Files

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Best for data that is structured as an array**
  - Or that has fixed-size offsets
- **Create Java class that acts as “view” on ByteBuffer**
  - Getter/Setter methods that use absolute buffer positions
- **Mapping is multi-step process**
  - Create Channel first, then map
- **ByteBuffer is limited to 2Gb, files can be bigger**
  - Option 1: individual mappings for sections of files
  - Option 2: create “megabuffer” that combines buffers, has similar API

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Mapping a File

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

RandomAccessFile allows both read and write

```
RandomAccessFile raf = new RandomAccessFile("/tmp/example.dat", "rw");
try
{
    FileChannel channel = raf.getChannel();
    MappedByteBuffer buf = channel.map(MapMode.READ_WRITE, 8L, 16);
    // do something with buf
}
finally
{
    raf.close();
}
```

Section of file to map

Mapping could be read-only even if RAF is opened read-write

Mapping remains until file is closed

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Code Sample: Object View on ByteBuffer

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

```
public class TempNodeData
{
   private static int OFF_PVID         = 0;     // long
   private static int OFF_LAT          = 8;     // float
   private static int OFF_LON          = 12;    // float
   private static int OFF_MORTON       = 16;    // int

   public static int RECORDSIZE        = 20;

   private ByteBuffer buffer;

   public TempNodeData(ByteBuffer srcBuf, int index)
   {
      srcBuf.position(index * RECORDSIZE);
      buffer = srcBuf.slice();
   }

   public long getPVID()
   {
      return buffer.getLong(OFF_PVID);
   }

   // ...
```

not thread safe!

`slice()` creates new buffer with same backing store

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# Example: Scaling Traffic Applications

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

- **Navteq is integrating GPS/cellphone probe data into traffic model**
  - Map-match, discover route, apply travel time to road segments
- **Problem: size x volume**
  - North America, South America, Europe total 100MM road segments
  - Peak design volume is 3 billion probe points/day
- **Solution: files sorted by location hash**
  - Nearby locations will be physically colocated on disk
  - Can limit probe traffic to small geographic area
  - Operating system loads only those pages that are needed

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6

# For More Information

96 97 1D 59 33 7B 7E 25 03 BC C2 51 AC F6 1D 0F 62 DA 4D 88 09 DD 3B 58 6F D5 84 1F 99 20 3B B1 7B 40 B4 77 CB 8A E3 05 23 2A 72 7D 5E 19 1C ED 7D

**http://www.kdgregory.com/index.php?page=java.byteBuffer**

8C 35 F4 7D F3 F5 E4 8E 50 4D 9A 35 C4 95 DE C1 82 42 0B 84 31 E9 AD FE 07 CF B5 EB AE E2 62 0D 3A C1 80 07 1E B5 77 3D 3C DE 95 1B 51 99 A5 BB C6