A Bit of Algebra

Massive Amounts of In-memory Key/Value Storage + In-Memory Search + Java == NoSQL Killer?

A bit of Algebra

Massive Amounts of In-memory Key/Value Storage +
In-Memory Search +
Java
== NoSQL Killer?

Kunal Bhasin, Deputy CTO, Terracotta

What is NoSQL?

- NoSQL = “Not only SQL”
- Structured Data not stored in traditional RDBMS
- E.g. Key-Value stores, Graph Databases, Document Databases
- It really is “Not only RDB” = NoRDB
- Key-Value stores
  - BigTable (disk)
  - Cassandra
  - Dynamo
  - BigTable

Why NoSQL?

- "One Size Fits All" is .. umm .. a little restrictive
- Use the right tool for the job
- Or the right strategy depending on business data
  - Not all data is equal – creation and consumption
    - Data Volume
    - Data access patterns
    - Consistency
    - Latency, Throughput
    - Scalability
    - Availability
- Not meant to be "anti"-RDBMS

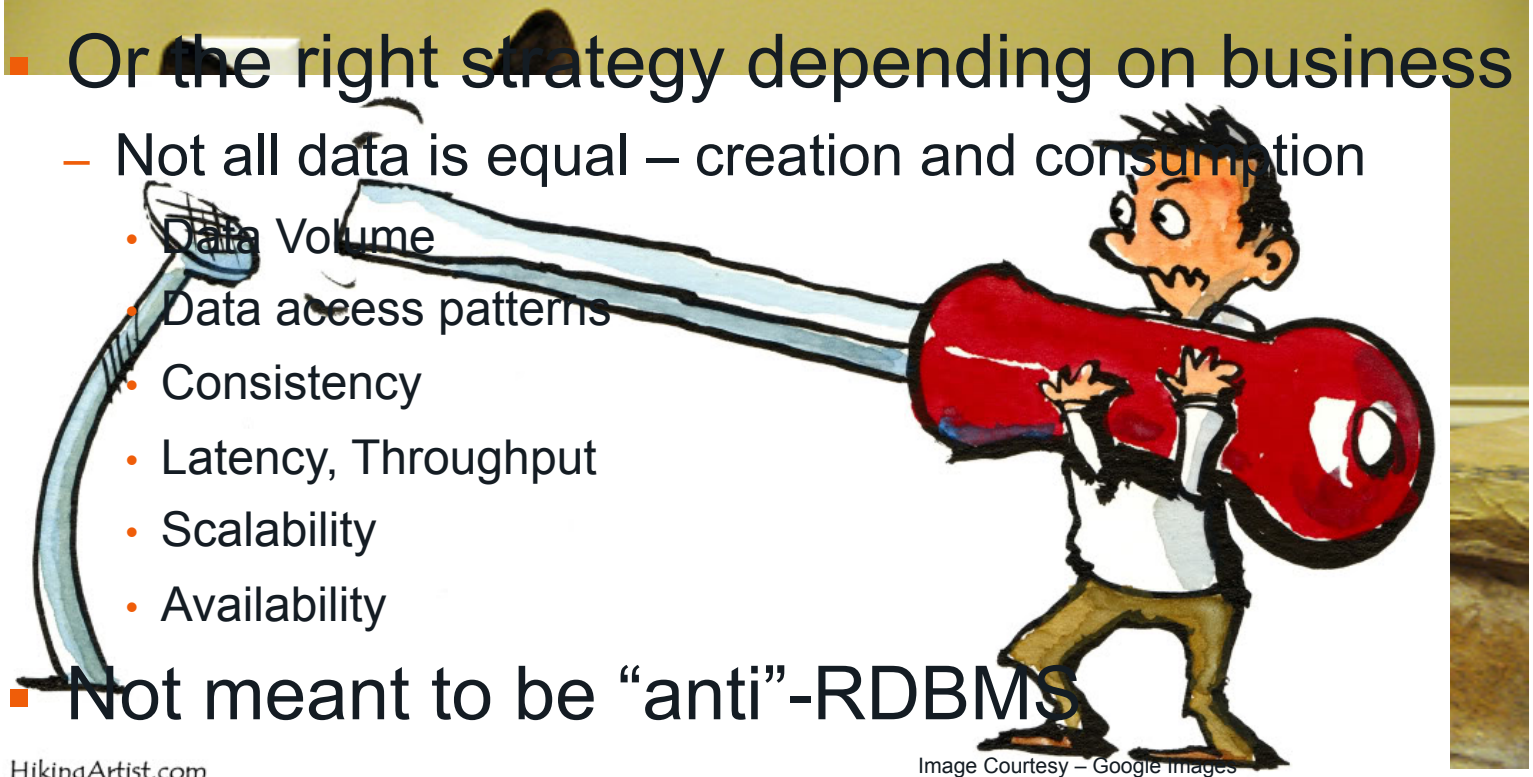


Image Courtesy – Google Images

Image URL - http://farm3.static.flickr.com/2523/4193330368_b22b644ddd.jpg

http://farm4.static.flickr.com/3620/3402670280_5e8be9f09c.jpg

What are we looking for?

- Lots of data
  - > 1 TB to PBs
- Performance
  - Low latency, high throughput access
- Scalability and Availability
- Flexibility in CAP tradeoffs
  - Consistency – eventual, strong, ACID
  - Availability – > 99.99 up time, Durability to failures
  - Automatic recovery on failures, real time alerts
- Flexibility in data consumption
  - Analytics, Compute

Algebra

Lots of data +

Performance +

Scalability and Availability +

Flexible CAP tradeoffs +

Flexible data consumption

= NoSQL or NoRDB

What is Ehcache?

- Simple API honed by 100,000's of production deployments

```
Cache cache = manager.getCache("sampleCache1");
Element element = new Element("key1", "value1");
cache.put(element);
```

- Default cache for popular frameworks
  - Hibernate, MyBatis, Open JPA
  - Spring (Annotations), Google Annotations
  - Grails
  - JRuby
  - Liferay
  - Cold Fusion

Simple Get/Put API

Sample Code:

```
public Object testCache(String key) throws Exception {
        CacheManager cacheManager = new CacheManager( “<path to my ehcache.xml>”);
        Cache myCache = cacheManager.getCache("MyCache");
        Object value;

        Element element = myCache.get(key);
        if (element == null) {
                value = "go get it from somewhere like DB or service, etc";
                myCache.put(new Element(key, value));
        } else {
                value = (Object) element.getValue();
        }

        return value;
}
```

Simple and flexible configuration

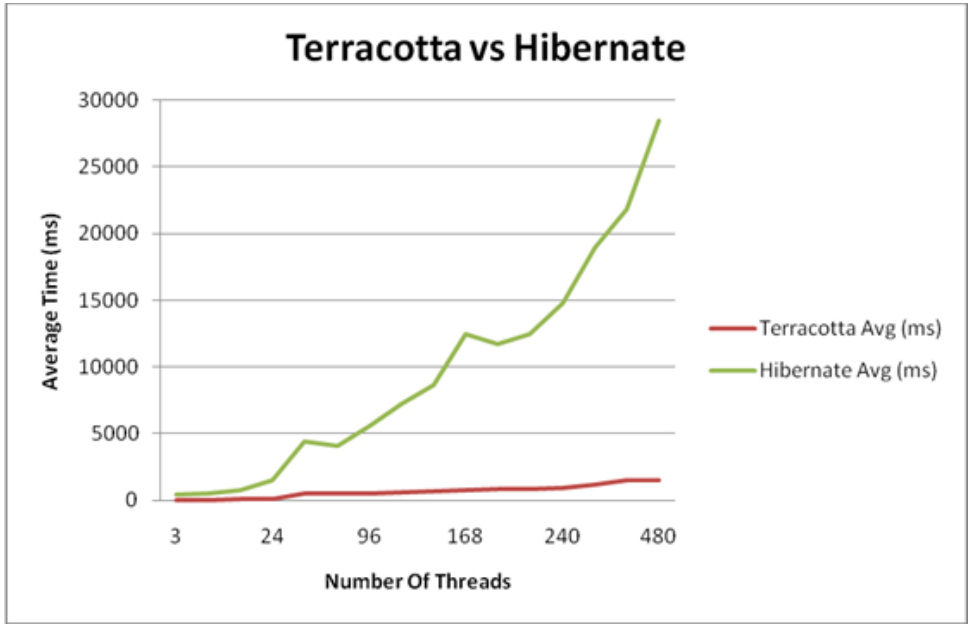
```
<ehcache>
   <defaultCache
          maxElementsInMemory="10000"
          eternal="false"
          timeToLiveSeconds="120”
        memoryStoreEvictionPolicy="LFU”/>

   <cache name=”WheelsCache"
         maxElementsInMemory="10000"
         timeToIdleSeconds="300”
       memoryStoreEvictionPolicy="LFU”/>

   <cache name=”CarCache"
         maxElementsInMemory="10000"
         timeToIdleSeconds="300”
      memoryStoreEvictionPolicy="LFU"/>
</ehcache>
```

Efficient implementation

- Highly concurrent and scalable
- Complements multi-threaded app servers
- Max utilization of hardware, scales to multi-core CPUs



K	V
Key 1	Value 1
Key 2	Value 2
Key 3	Value 3

Some more features

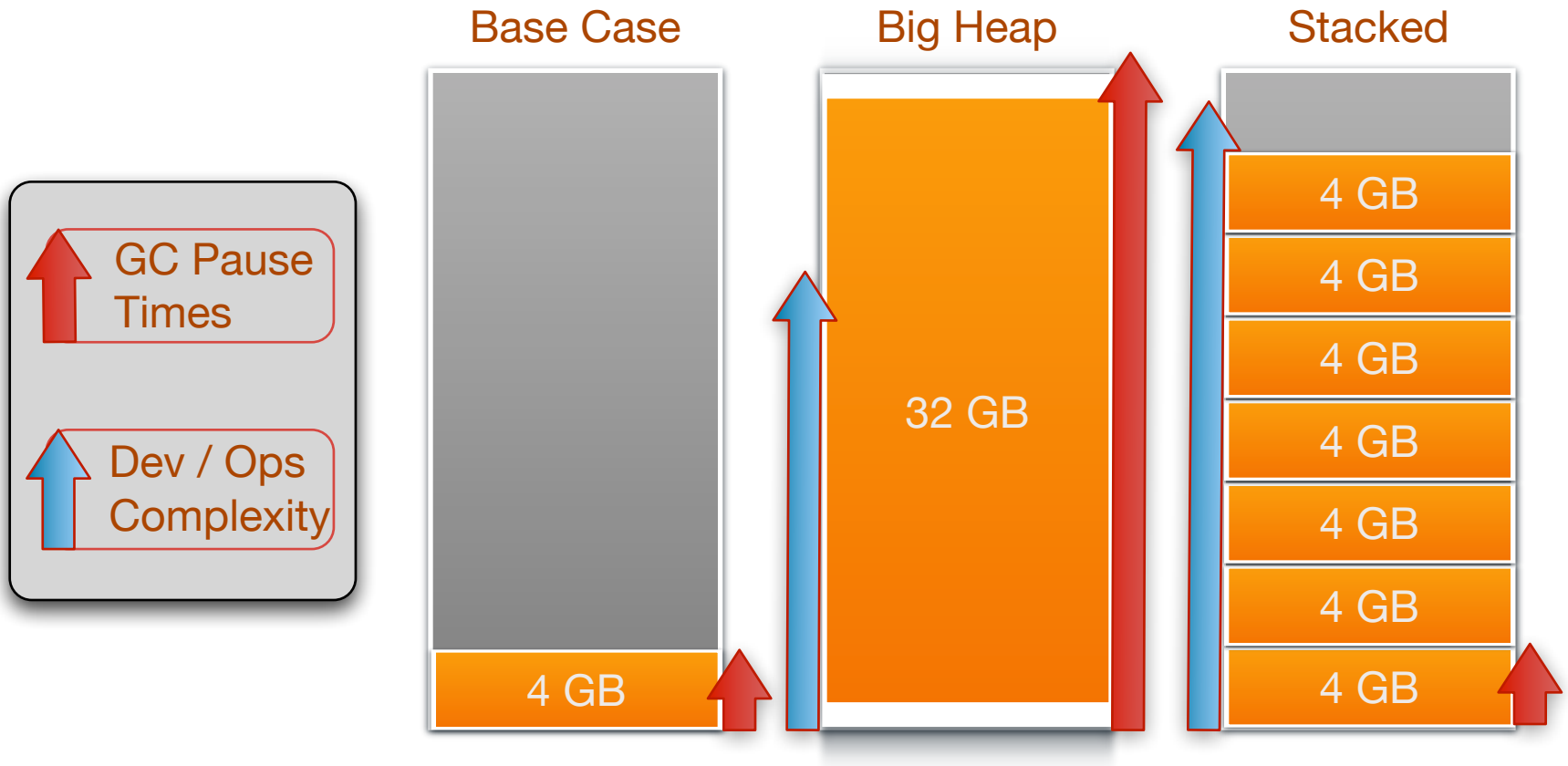
- Pluggable eviction policy
- Async write-behind
- JTA support
- Third-party monitoring integration
- Large caches, GC free
- Bulk loader API's
- Management console
- WAN replication

Lots of Data + Performance

Ehcache BigMemory

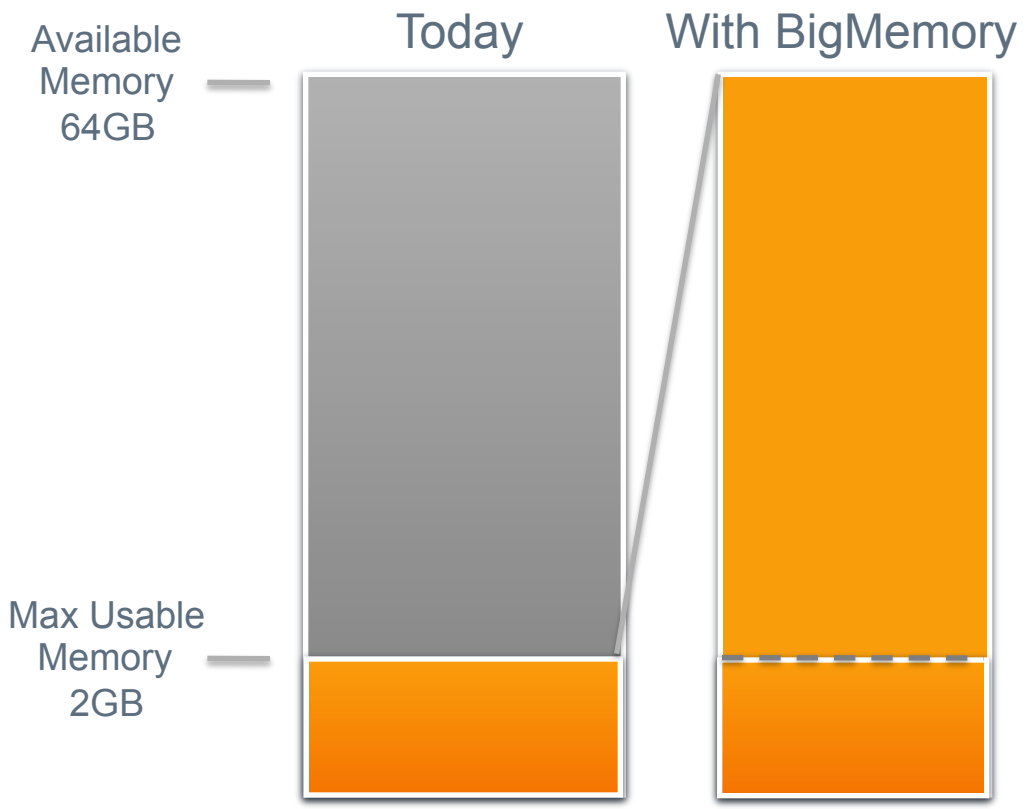
Why BigMemory?

Java has not kept up with Hardware (because of GC)



BigMemory: Scale Up GC Free

- Dramatically increased usable memory per JVM
- >64GB/JVM
- 10x JVM density
- Predictable latency
- Easier SLAs
- No GC pauses
- No tuning
- Pure Java



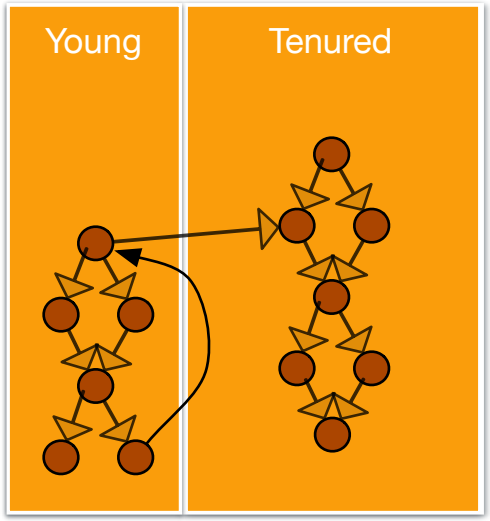
BigMemory: Scale Up GC Free

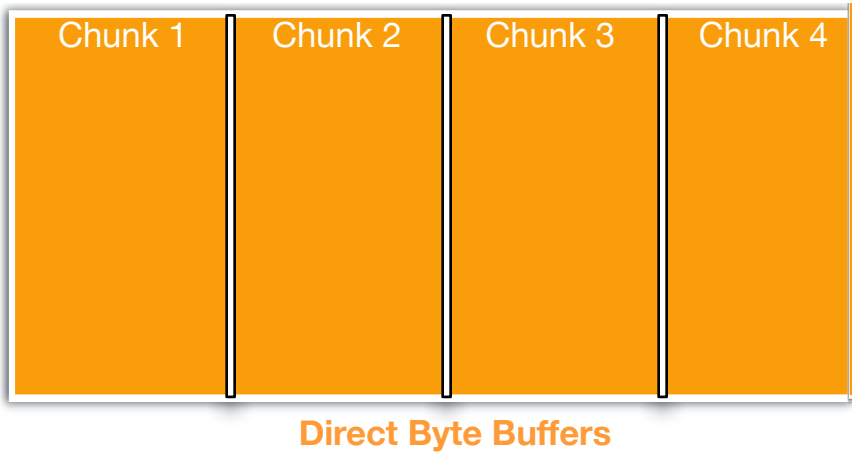
GC

- Complex, dynamic reference based object store
- Costly (walk the entire object graph) to find “unused/unreachable” objects and reclaim memory

BigMemory

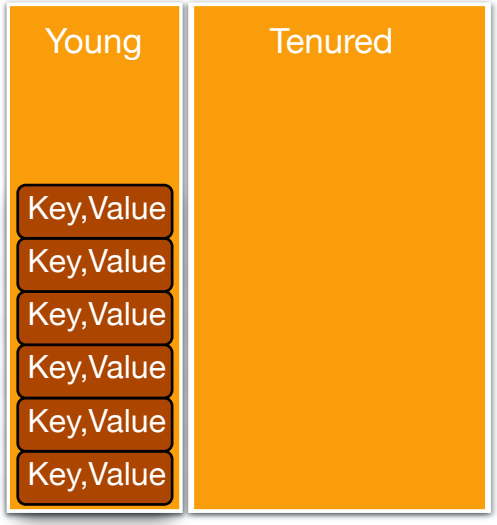
- Transparent to Ehcache users,
- Simple <Key,Value> store with no cross-references,
- Uses RAM directly
- Clean interfaces (get, put, remove) for CRUD operations



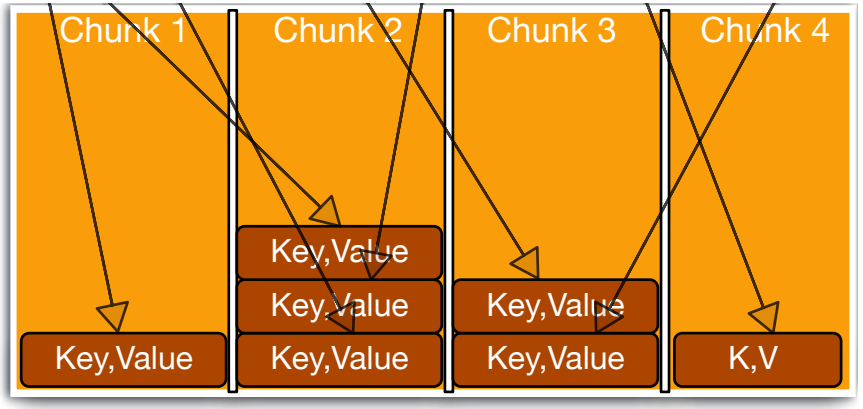


BigMemory: Scale Up GC Free

GC
New objects created in Young Generation of heap



BigMemory
New objects are stored on RAM, away from java heap

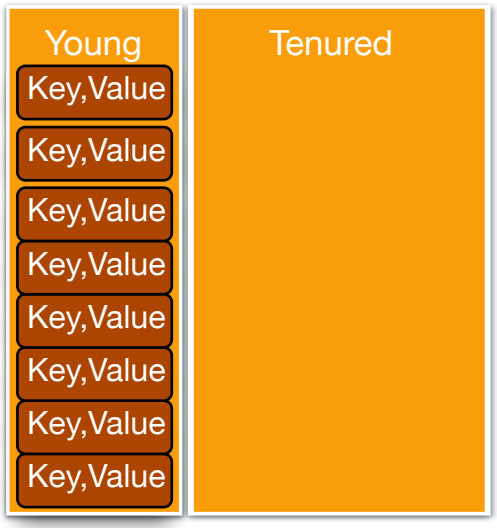


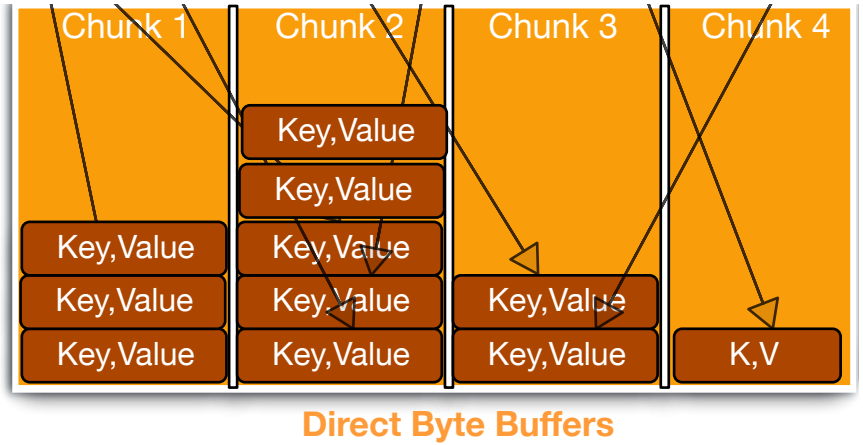
Direct Byte Buffers

BigMemory: Scale Up GC Free

GC
Young generation full causes
Young GC, costly but not as bad

BigMemory
Hot objects are kept in BigMemory
based on access pattern

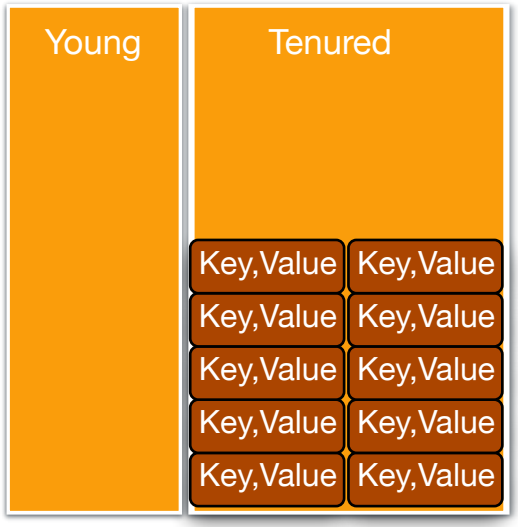


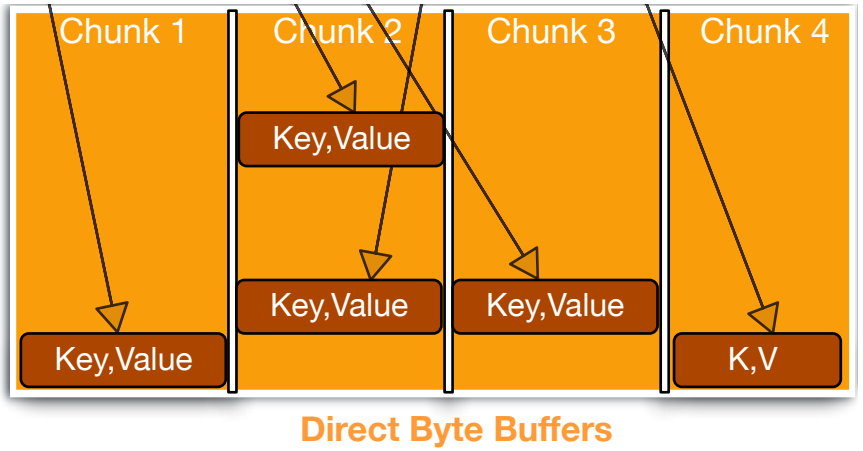


BigMemory: Scale Up GC Free

GC
Parallel Collector: Medium to long lived objects end up in Tenured Space

BigMemory
Objects removed on remove(key), TimeToLive, TimeToIdle, frequency of access; no need to walk the graph



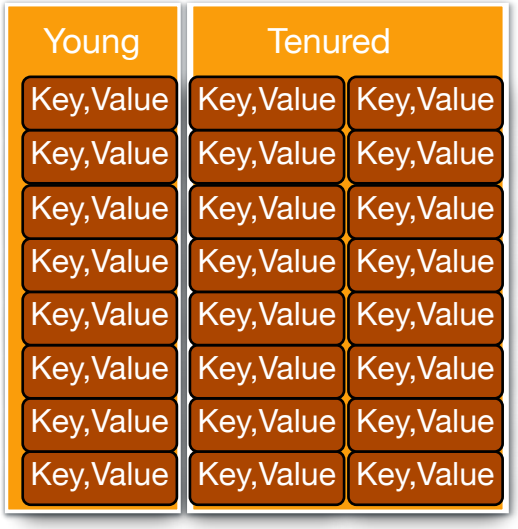


Direct Byte Buffers

BigMemory: Scale Up GC Free

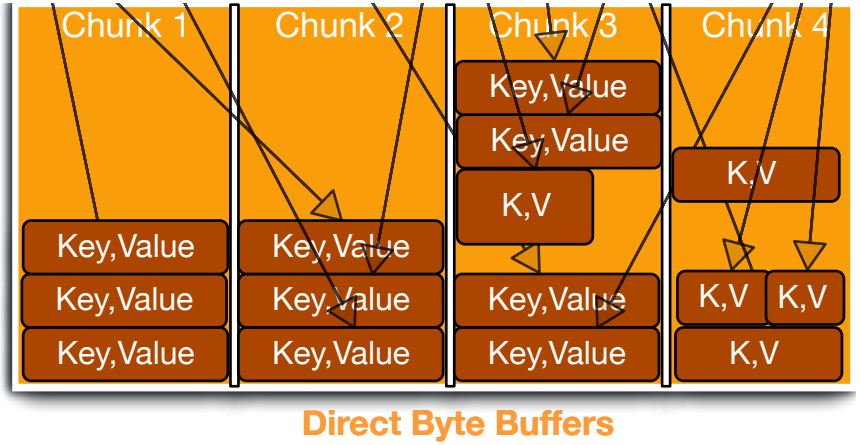
GC

Parallel Collector: Long (stop the world) pauses proportional to size of the heap and amount of “collectable” objects



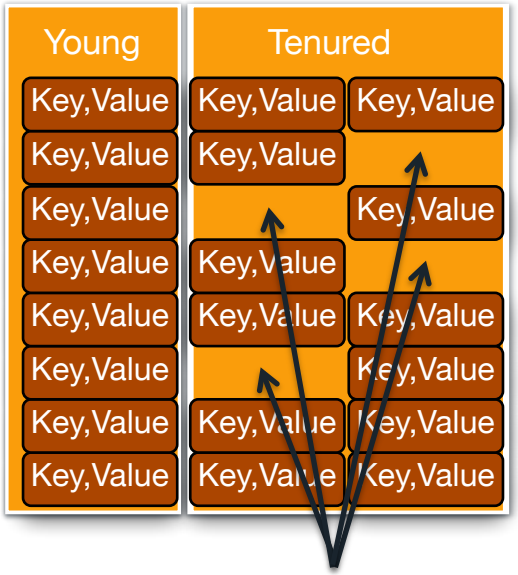
BigMemory

Highly concurrent and Intelligent algorithms to seek “best fit” free memory chunks: No pauses



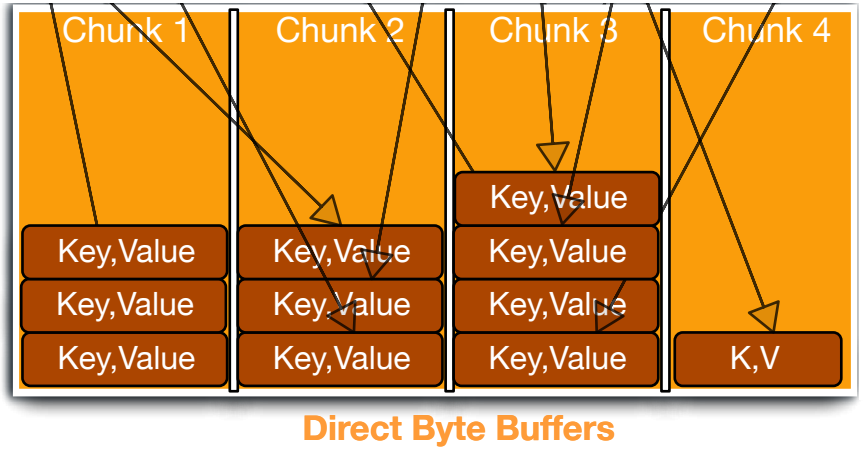
BigMemory: Scale Up GC Free

GC
CMS Fragmentation: Not enough contiguous space to copy from young to tenured, long pauses (stop the world) to run compaction cycles



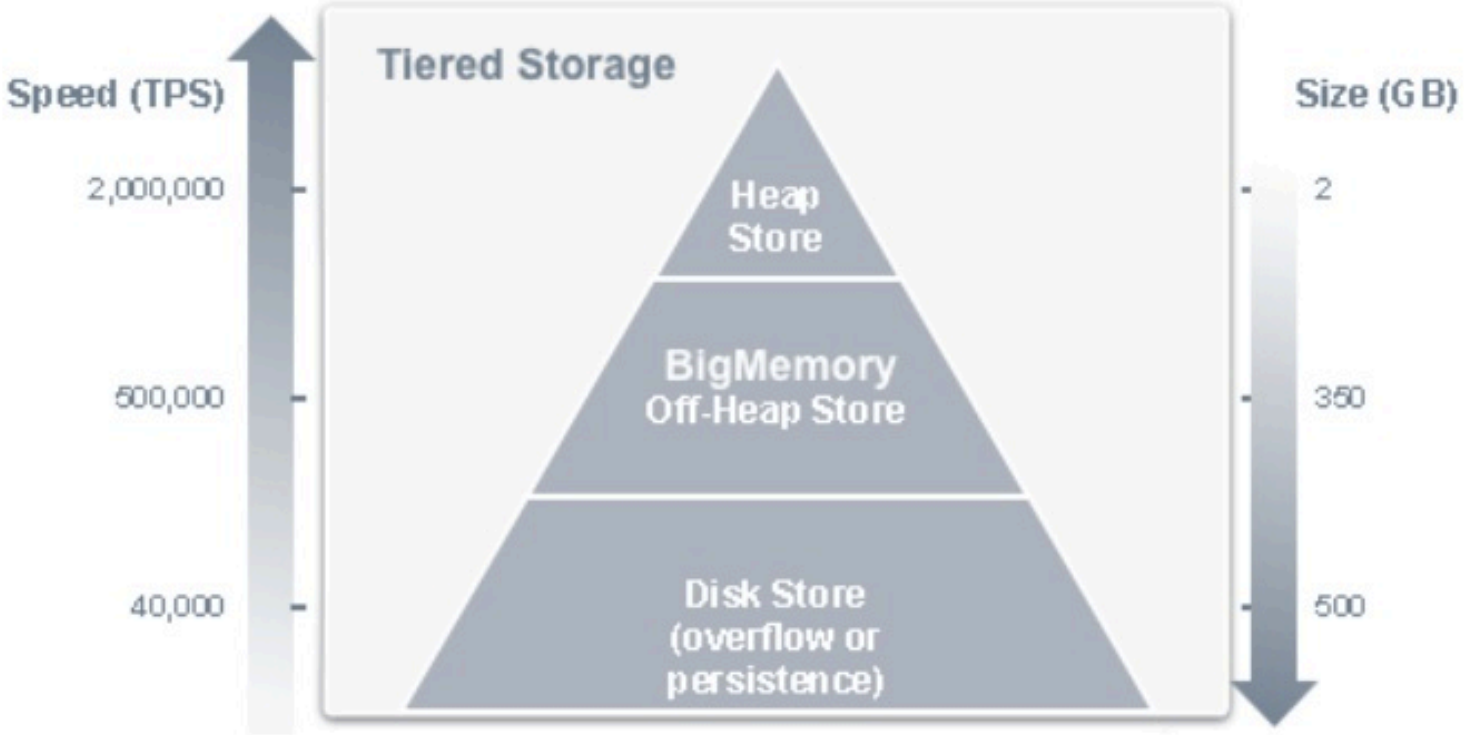
Not enough contiguous space = Fragmentation = Full GC

BigMemory
Striped Compaction = No Fragmentation + Good Performance

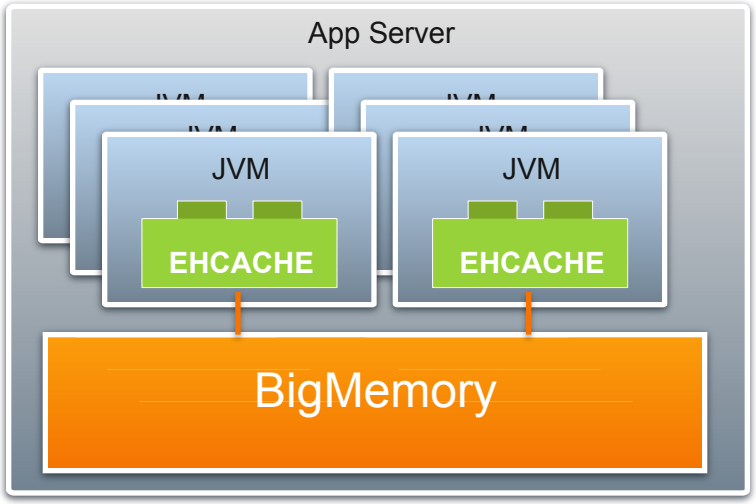


Direct Byte Buffers

BigMemory - Tiered Storage



Ehcache with BigMemory



- Up to 350 GB tested
- < 1 second GC pauses
- Standalone or Distributed
- > 1 TB with Terracotta Server Array

Sample ehcache.xml for standalone Flexibility – add BigMem Selectively

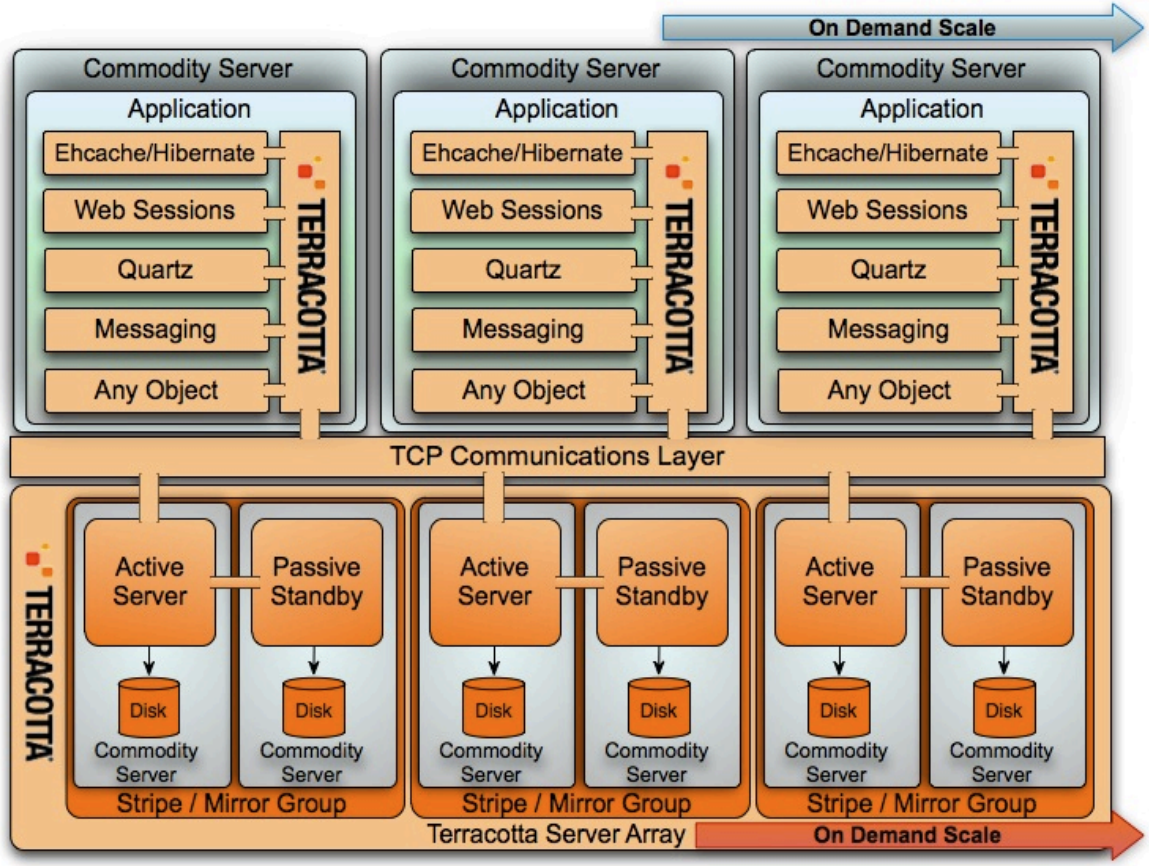
```
<ehcache>
    <defaultCache
          maxElementsInMemory="10000"
          eternal="false"
          timeToLiveSeconds="120”
          memoryStoreEvictionPolicy="LFU”/>
    <cache name=”WheelsCache"
          maxElementsInMemory="10000"
          timeToIdleSeconds="300”
          memoryStoreEvictionPolicy="LFU”
          overflowToOffHeap="true”
          maxMemoryOffHeap=”30G"/>
    <cache name=”CarCache"
          maxElementsInMemory="10000"
          timeToIdleSeconds="300”
          memoryStoreEvictionPolicy="LFU"/>
</ehcache>
```

Scalability & Availability

Terracotta Server Array

What is Terracotta?

- Enterprise class data management
  - Clustering, Distributed Caching
  - Highly Available (99.999)
  - Linear Scale Out
  - BigMemory - More scalability with less Hardware
  - ACID, Persistent to Disk (& SSD)
  - Ease of Operations
  - Flexibility with CAP tradeoffs

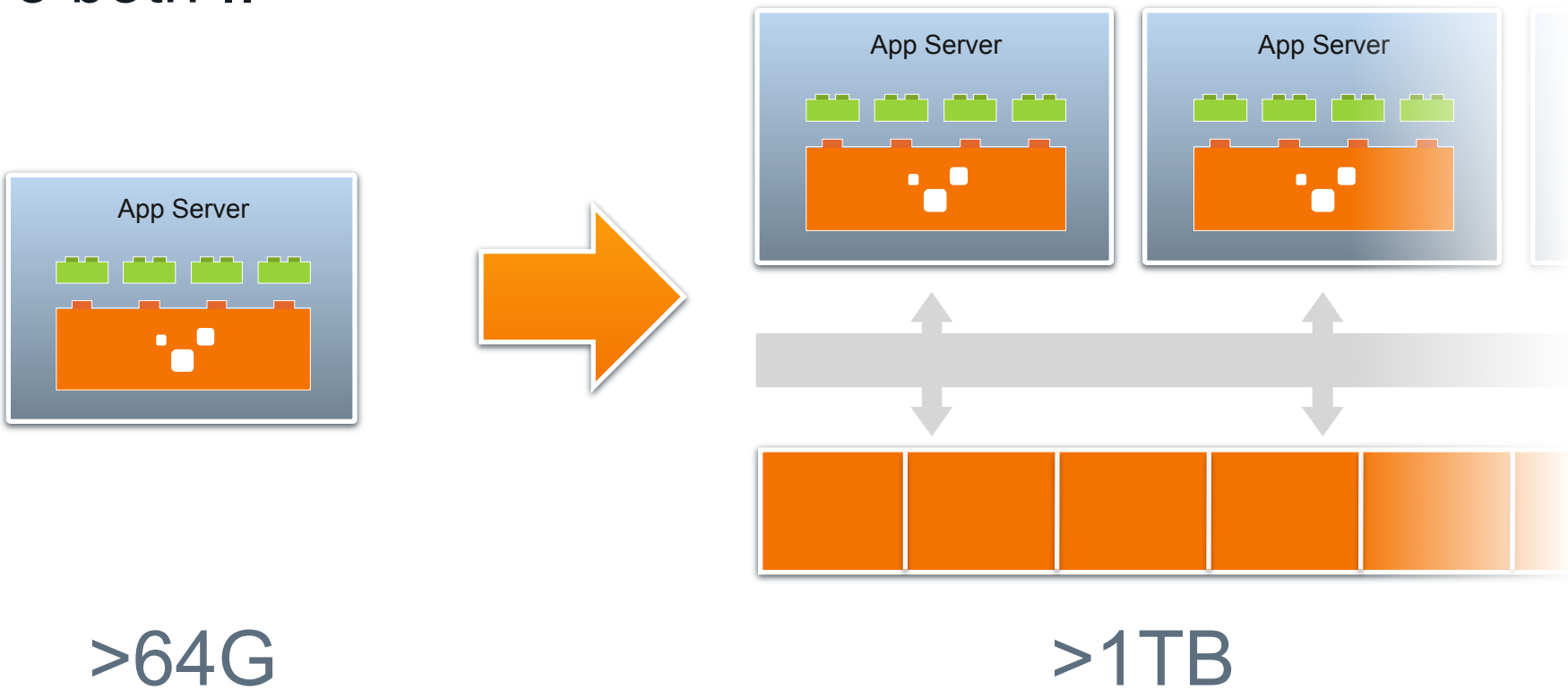


Snap In

```
<ehcache>
   <terracottaConfigurl="someserver:9510"/>
   <defaultCache
        maxElementsInMemory="10000”
        eternal="false”
        timeToLiveSeconds="120”/>
   <cache name="com.company.domain.Pets"
        maxElementsInMemory="10000"
        timeToLiveSeconds="3000”>
     <terracotta clustered="true” />
   </cache>
</ehcache>
```

Scale up or Scale out?

- Do both ..



BigData?

Do you need PBs in BigMemory?

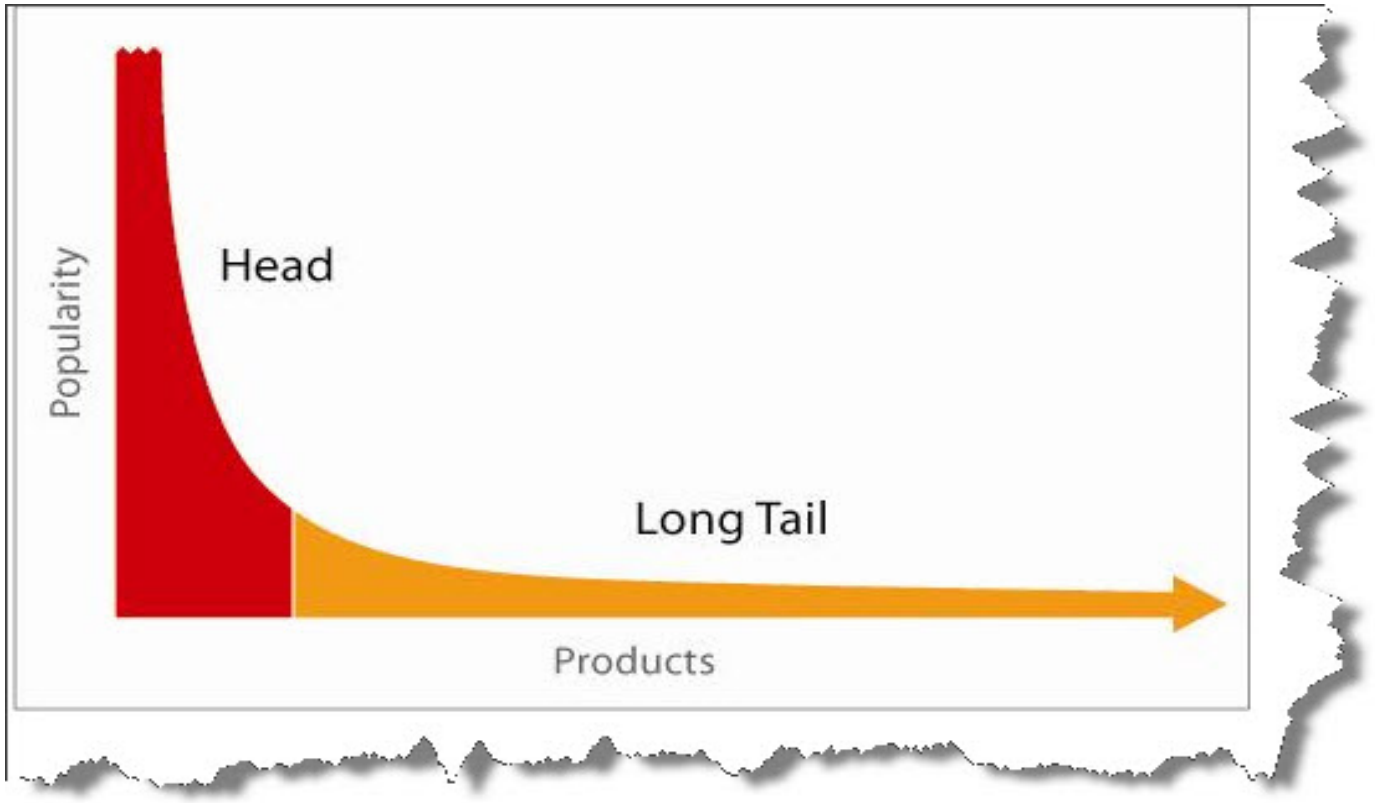
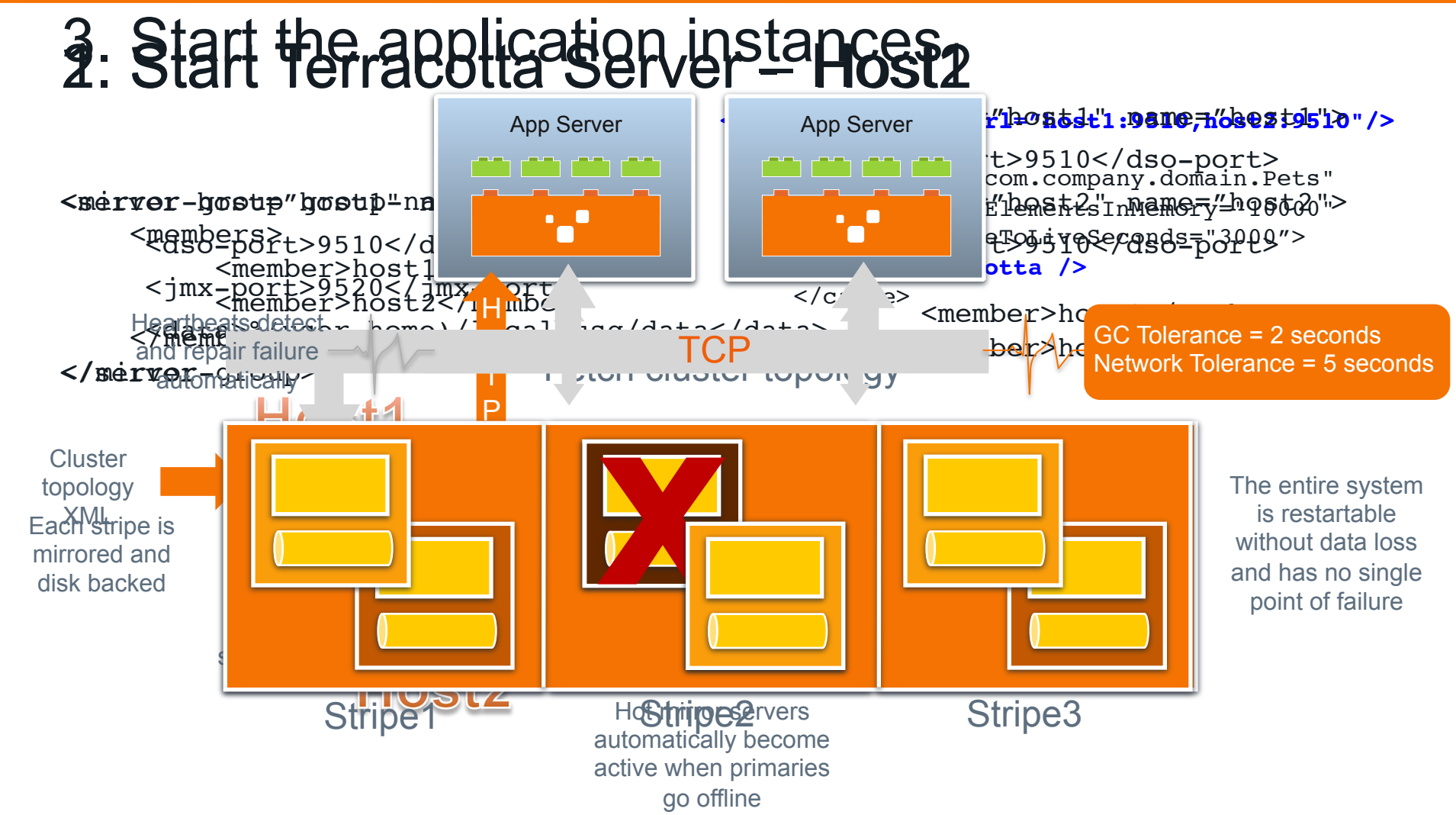


Image Courtesy – Google Images

Image URL - http://blog.hubspot.com/Portals/249/images//ltail-long-tail.jpg

High Availability



CAP Tradeoffs

- Consistency-Availability-Partition Tolerance theorem
  - Conjecture coined by Eric Brewer of UC Berkeley - 2000
  - Proven by Nancy Lynch and Seth Gilbert of MIT - 2002

It is impossible for a distributed system to simultaneously provide all three of the following guarantees:

Consistency (All nodes see the same data at the same time)

Availability (Node failures do not prevent others from continuing to operate)

Partition Tolerance (The system continues to operate despite arbitrary message loss or network partitions)

CAP Tradeoffs

- Consistency-Availability-Partition Tolerance theorem
  - Conjecture coined by Eric Brewer of UC Berkeley - 2000
  - Proven by Nancy Lynch and Seth Gilbert of MIT - 2002

It is impossible for a distributed system to simultaneously provide all three of the following guarantees:

Consistency (All nodes see the same data at the same time)

Availability (Node failures do not prevent others from continuing to operate)

Partition Tolerance (The system continues to operate despite arbitrary message loss or network partitions)

PACELC

If Partition, then tradeoff between

Availability &

Consistency

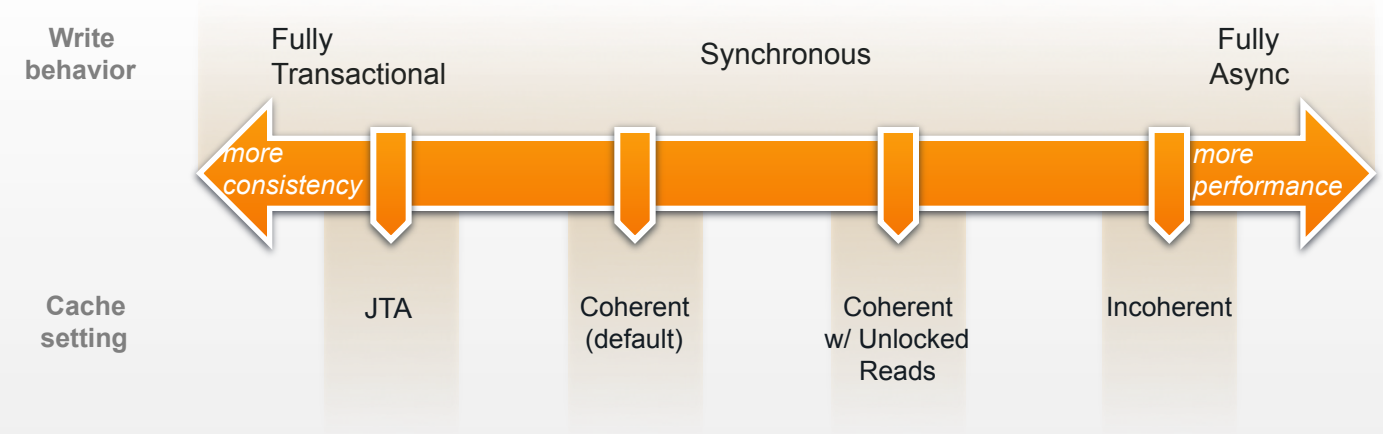
Else, tradeoff between

Latency &

Consistency

- Other considerations
  - Durability
  - Levels of consistency – eventual, weak, strong (ACID)

Consistency-Latency Spectrum




```
<cache name="UserPreferencesCache"
       maxElementsInMemory="10000"
       timeToIdleSeconds="300"
            memoryStoreEvictionPolicy="LFU">
    <terracotta clustered="true"
  consistency="eventual"/>
 </cache>
```

```
<cache name="ShoppingCartCache"
       maxElementsInMemory="10000"
       timeToIdleSeconds="300"
            memoryStoreEvictionPolicy="LFU">
    <terracotta clustered="true"
  consistency="strong"/>
 </cache>
```

Flexibility

```
<ehcache>
<terracottaConfigurl="someserver:9510"/>
  <cache name=”LocalCache”
        timeToIdleSeconds="300”
        memoryStoreEvictionPolicy="LFU”/>

  <cache name=”UserCache”
        timeToIdleSeconds="300”
        memoryStoreEvictionPolicy="LFU”
        overflowToOffHeap="true”
        maxMemoryOffHeap=”30G”/ >

   <cache name=”ShoppingCartCache”
        timeToIdleSeconds="300”
        memoryStoreEvictionPolicy="LFU”>
        <terracotta clustered="true" consistency=”strong"/>
   </cache>
</ehcache>
```

Flexibility in data consumption

Search for Analytics,
Quartz Where for Compute

Ehcache Search

- Full featured Search API
- Any attribute in the Value Graph can be indexed
- Supports large indices on BigMemory
- Time Complexity
  - log(n/number of stripes)
- Intuitive Fluent API
  - E.g. Search for 32 year old males and return the cache keys.

```
Results results = cache.createQuery().includeKeys()
                .addCriteria
                (age.eq(32))
                .and
                (gender.eq("male"))
                .execute();
```

Ehcache Search

- Make a cache searchable

```
<cache name="cache2” >
  <searchable metadata="true"/>
</cache>
```

- What is searchable?
  - Element keys, values and metadata, such as creation time
    - Attribute types: Boolean, Byte, Character, Double, Float, Integer, Long, Short, String, Date, Enum
    - Metadata: creationTime, expirationTime, lastAccessTime, lastUpdateTime, version
- Specify attributes to index

```
<cache name="cache2" maxElementsInMemory="10000” >
  <searchable>
     <searchAttribute name="age" class="net.sf.ehcache.search.TestAttributeExtractor"/>
     <searchAttribute name="gender" expression="value.getGender()"/>
   </searchable>
</cache>
```

Quartz

- Enterprise job scheduler
- Drive Process Workflow
- Schedule System Maintenance
- Schedule Reminder Services
- Master-Worker, Map-Reduce
- Simple configuration to cluster with Terracotta Array
- Automatic load balancing and failover of jobs in a cluster

Quartz

- Scheduler, Jobs and Triggers

```
JobDetail job = new JobDetail("job1", "redTriggers", HelloJob.class);

SimpleTrigger trigger = new SimpleTrigger("trigger1", "blueGroup", new
   Date());

scheduler.scheduleJob(job, trigger);
```

- Powerful, flexible triggers (like cron)

0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month
0 11 11 11 11 ?	Fire every November 11th at 11:11am
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day

Quartz Where

- Locality of Execution
  - Node Groups

```
org.quartz.locality.nodeGroup.fastNodes = fastNode
org.quartz.locality.nodeGroup.slowNodes = slowNode
org.quartz.locality.nodeGroup.allNodes = fastNode,slowNode
```

  - Trigger Groups

```
org.quartz.locality.nodeGroup.fastNodes.triggerGroups = fastTriggers
org.quartz.locality.nodeGroup.slowNodes.triggerGroups = slowTriggers
```

  - JobDetail Groups

```
org.quartz.locality.nodeGroup.fastNodes.jobDetailsGroups = fastJobs
org.quartz.locality.nodeGroup.slowNodes.jobDetailsGroups = slowJobs
```

Quartz Where

- Execute compute intensive jobs on fast nodes

```
LocalityJobDetail jobDetail =
    localJob(
        newJob(ImportantJob.class)
            .withIdentity(”computeIntensiveJob")
            .build())
        .where(
            node()
                .is(partOfNodeGroup("fastNodes")))
        .build();
```

- Execute memory intensive jobs with a memory constraint
  - E.g. At least 512 MB

```
scheduler.scheduleJob(
    localTrigger(
        newTrigger()
        .forJob(”memoryIntensiveJob"))
        .where(node()
        .has(atLeastAvailable(512, MemoryConstraint.Unit.MB)
        .build());
```

Quartz Where

- Execute CPU intensive jobs with a CPU constraint
  - E.g. At least 16 CPU cores

```
.forJob("memoryIntensiveJob"))
        .where(node()
        .has(coresAtLeast(16)
        .build());
```

  - E.g. At most 0.5 CPU load

```
.forJob("memoryIntensiveJob"))
        .where(node()
        .has(loadAtMost(0.5)
        .build());
```

- Execute a job on Linux OS

```
.forJob("memoryIntensiveJob"))
        .where(node()
        .is(OSConstraint.LINUX)
        .build());
```

Algebra

Ehcache BigMemory (Lots of Data, Perf) +

Terracotta (Scalability, Availability) +

Ehcache Search (Analytics) +

Quartz Where (Compute)

= Is it NoSQL or NoRDB?

I wouldn't want to call it that, but it addresses a lot of the similar concerns.

Kunal Bhasin

www.terracotta.org
kunal@terracotta.org
@kunaalb

Kunal Bhasin, Deputy CTO, Terracotta