Through the keyhole of #hashCode into JVM

https://github.com/vladimirdolzhenko/hashCodeLegend

Vladimir Dolzhenko, IHS Markit

@dolzhenko
vladimir.dolzhenko @ ihsmarkit.com

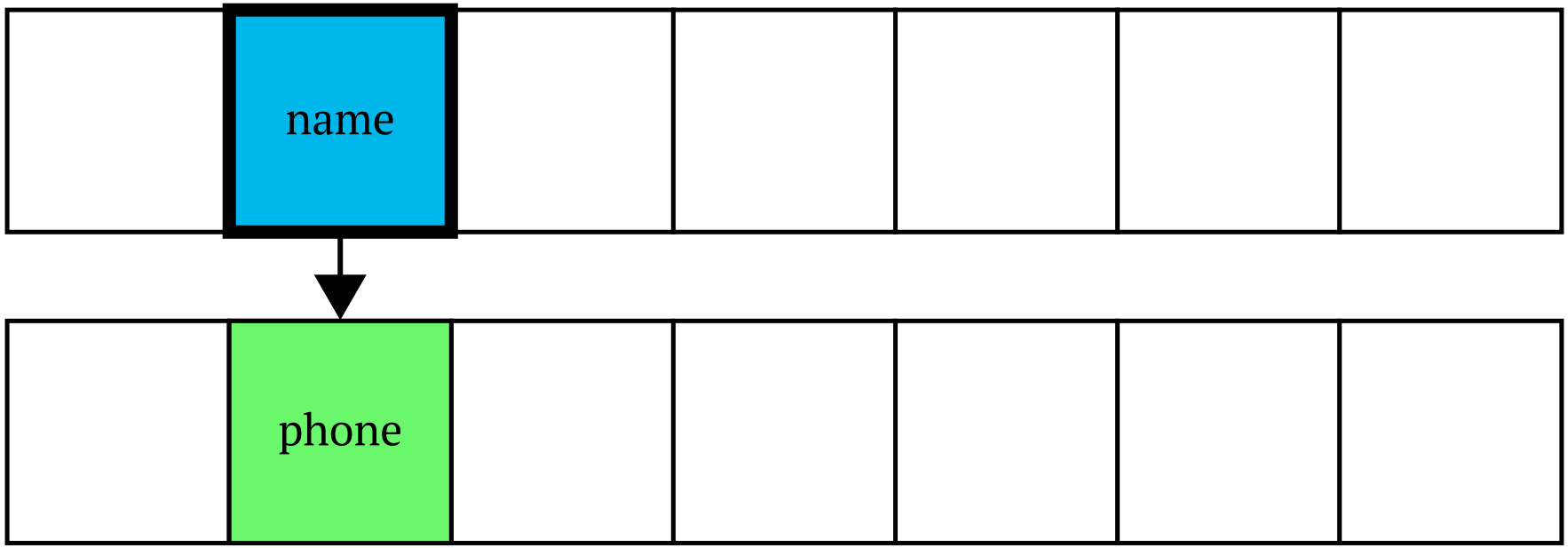
2017-04-10

no warranty

ToC

- A few theory
- hashCode calculation rules
  - hashCode-based DoS attack
- HashCode as address of object - legend or myth
  - JVM internals: Unsafe, GC and allocations
  - Some battles on hashCode
- Make locks cheap again

Associative array / Dictionary / Map




```
int hash =  hash( key );
int index = Math.abs( hash )
        % keys.length;
```

javadoc:

java.lang.Object

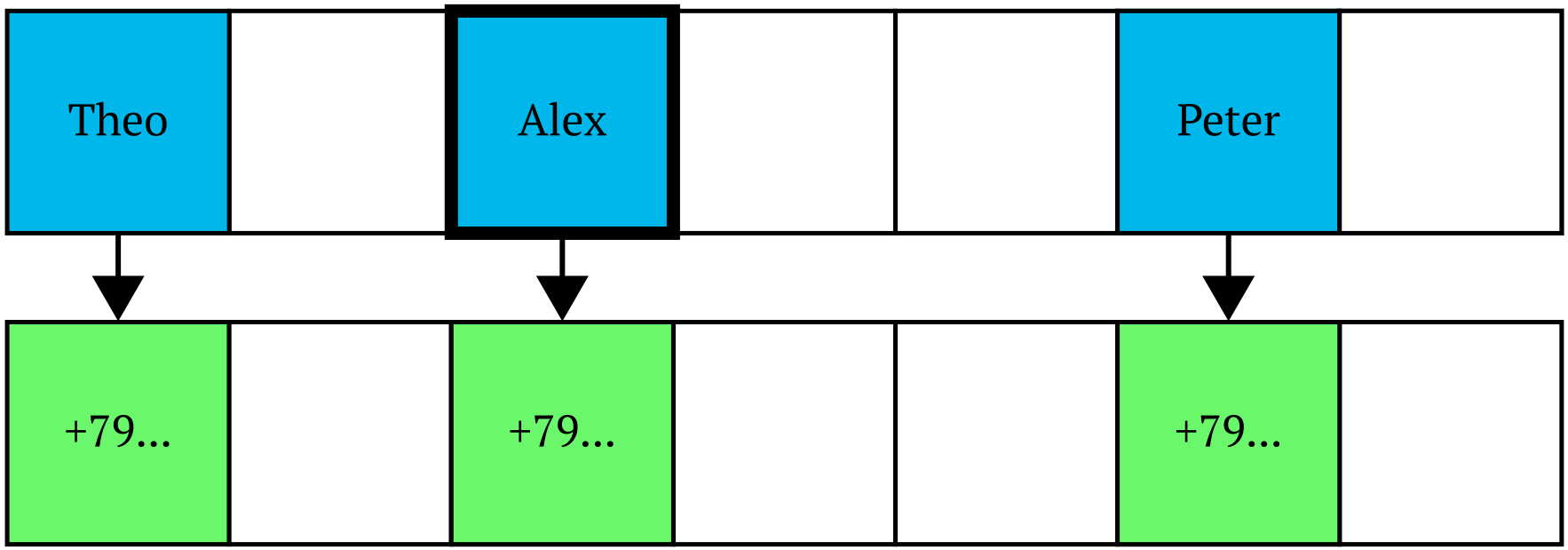
```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by **HashMap**.

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Write down (Alex, +79…)

hashCode(Alex) = 23 index = 23 % array.length = 2



Search / insert complexity

```
String key = "Alex";

int index = Math.abs( key.hashCode() )
      % keys.length;

if ( key.equals( keys[index] ) )
    return values[index];

return null;
```

complexity → O(1)

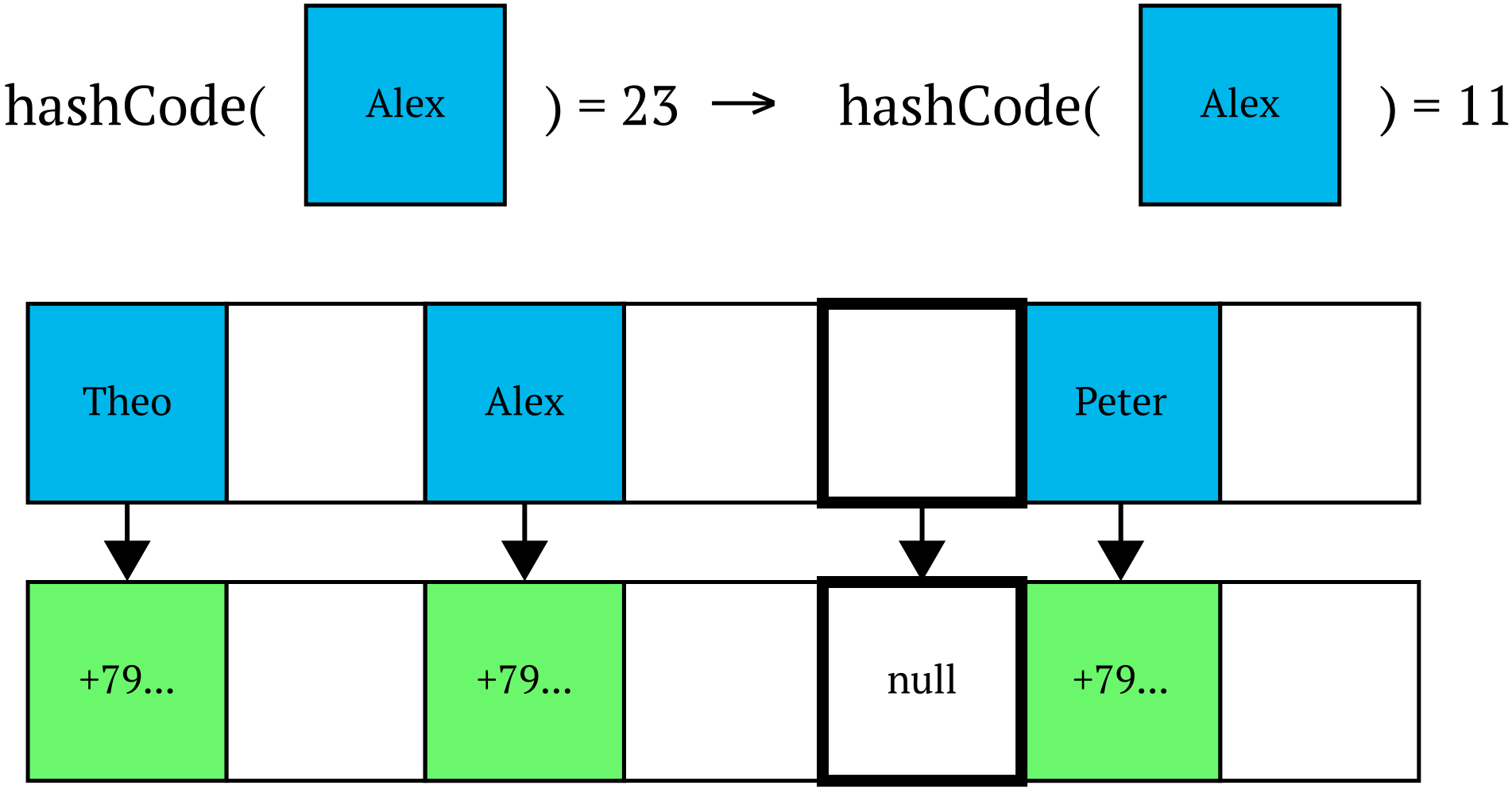
Contract of hashCode

- consitency & persistence

Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method **must *consistently*** return the **same integer**, provided no information used in equals comparisons on the object is modified.

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

hashCode contract violation



Collisions

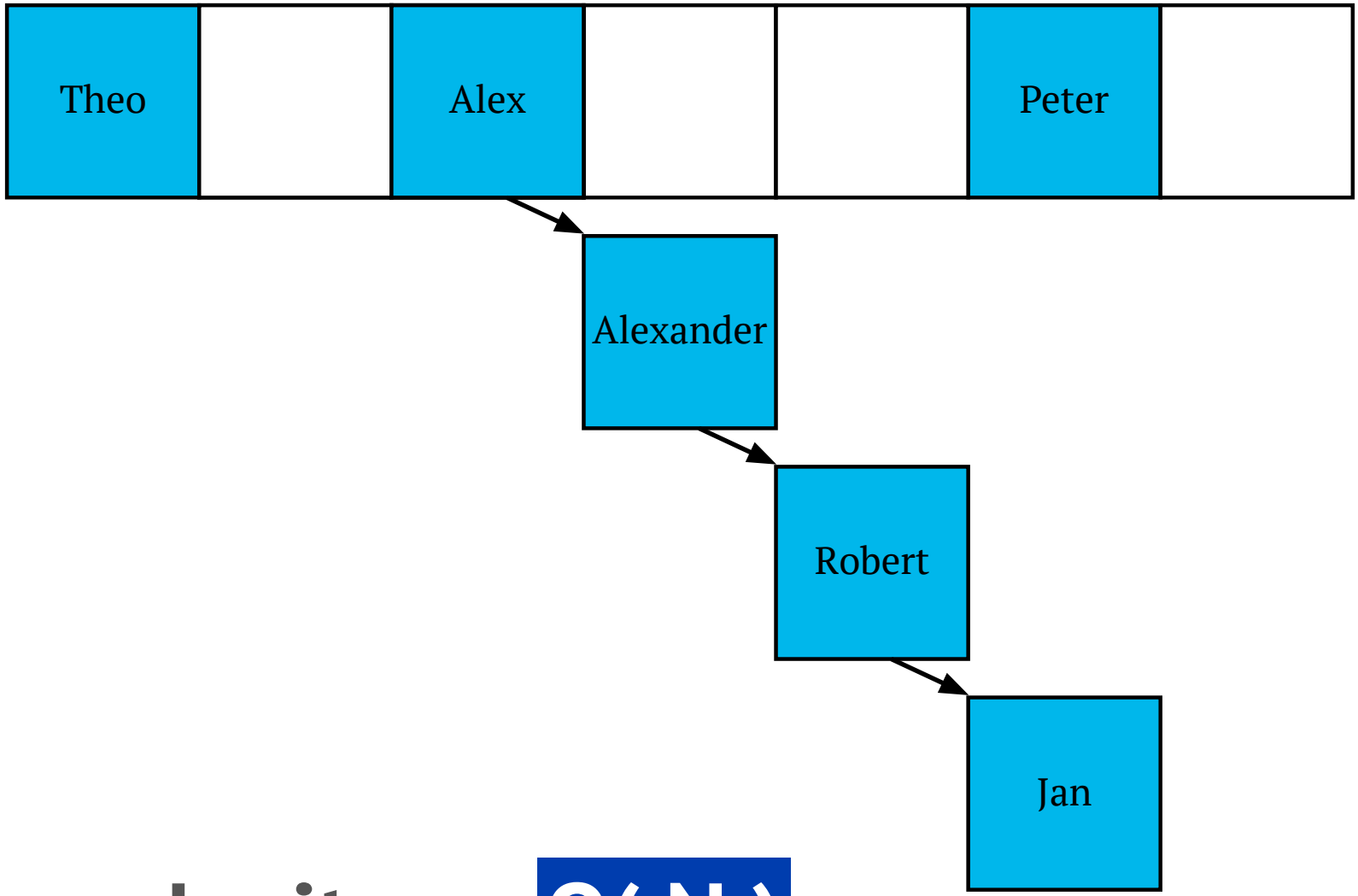
Theo		Alex			Peter	

hashCode(Alexander) = 23

hashCode(Robert) = 23

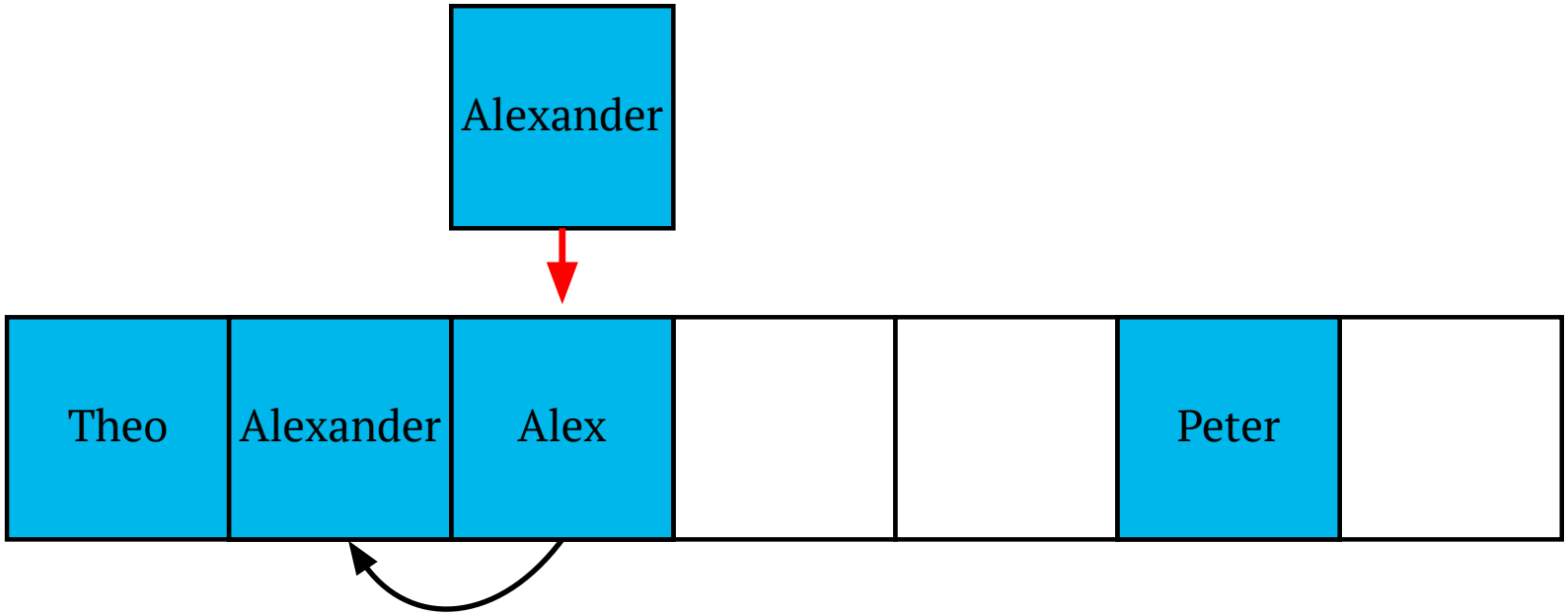
hashCode(Jan) = 23

Chaining

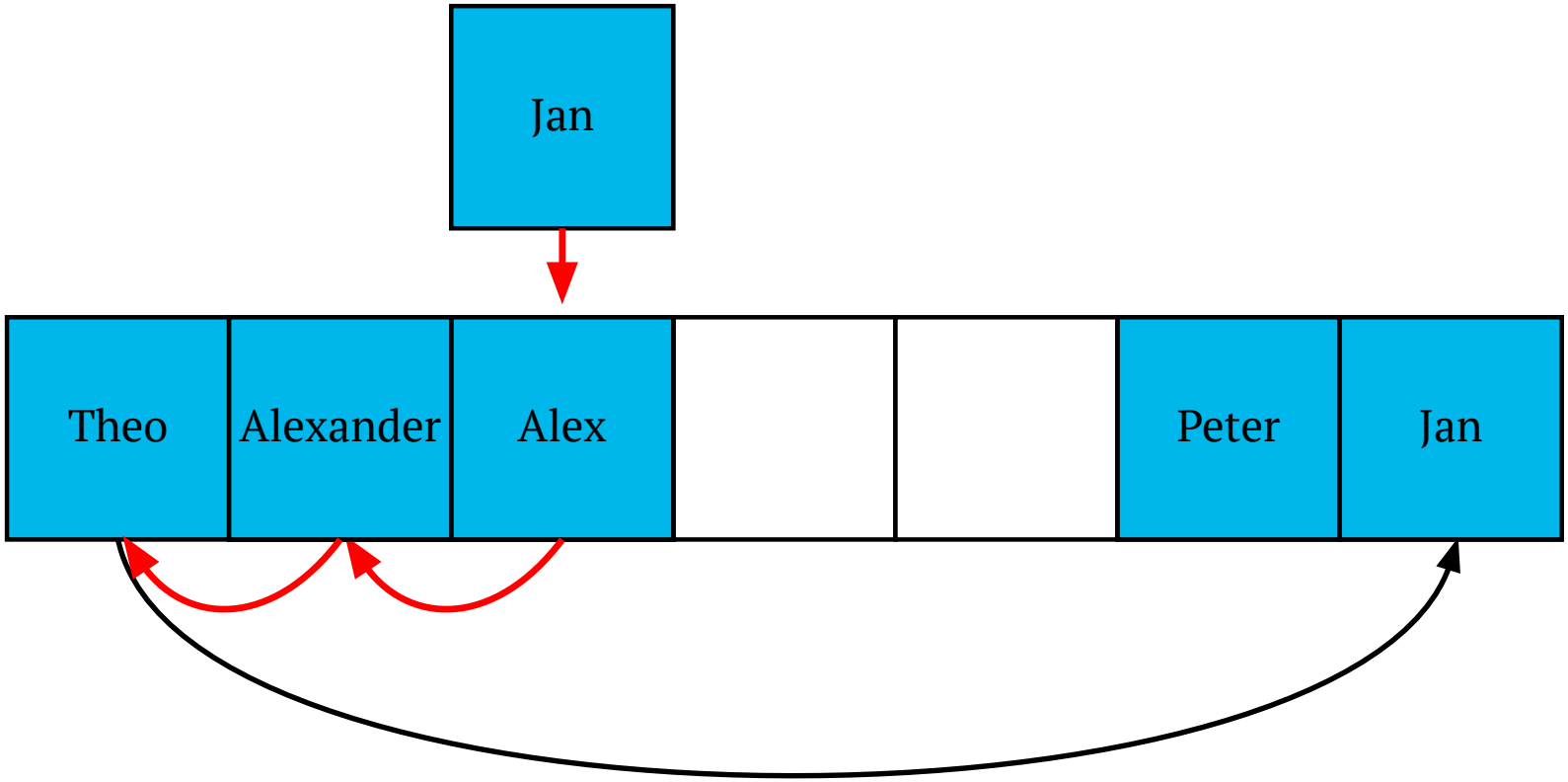


complexity → O(N)

Open Addressing



Open Addressing



complexity → O(N)

hashCode - function of object content

```
String s1 = new String("java");
String s2 = new String("java");

assert s1.hashCode() == s2.hashCode(); // true

Integer i1 = new Integer(42);
Integer i2 = new Integer(42);

assert i1.hashCode() == i2.hashCode(); // true
```

Polynomial hashCode

$$hashCode = \sum_{k=0}^{n} 31^{n-k} \cdot property_k$$

$$equals \Rightarrow$$
$$this.property_k == that.property_k,$$
$$\forall k \in [0, n]$$

String.hashCode()

```
public final class String {
    private final char value[];
    private int hash;

    public int hashCode() {
        int h = hash;
        if (h == 0 && value.length > 0) {
            char val[] = value;

            for (int i = 0; i < value.length; i++)
                h = 31 * h + val[i];

            hash = h;
        }
        return h;
    }
}
```

java v.1.1.1 - String.hashCode()

```java
public int hashCode() {
    int h = 0;
    int off = offset;
    char val[] = value;
    int len = count;

    if (len < 16) {
        for (int i = len ; i > 0; i--) {
            h = (h * 37) + val[off++];
        }
    } else {
        // only sample some characters
        int skip = len / 8;
        for (int i = len ; i > 0; i -= skip, off += skip) {
            h = (h * 39) + val[off];
        }
    }
    return h;
}
```

31 : Detective story

- **1997-04-17 (!!!)**: java bug #4045622
  - All of the words and phrasesin Merriam-Webster's 2nd Int'l Dictionary (311_141 strings, avg length: 10 chars).
  - All of the strings in `/bin/*`, `/usr/bin/*`, `/usr/lib/*`, `/usr/ucb/*` & `/usr/openwin/bin/*` (66_304 strings, avg length: 21 chars).
  - A list of URLs gathered by a web-crawler that ran for several hours last night (28_372 strings, avg length 49 characters).

polynomial of **31**, **33**, **37** → min. avg collision length

String.hashCode is the part of public API

```
public final class String {
  /**
   * Returns a hash code for this string. The hash code
   * for a String object is computed as
   *
   * s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
   *
   * using int arithmetic, where s[i]
   * is the ith character of the string,
   * n is the length of the string,
   * and ^ indicates exponentiation.
   * (The hash value of the empty string is zero.)
   *
   * @return  a hash code value for this object.
   */
    public int hashCode() {          ...       }
}
```

it might be found...

```
assert "Aa".hashCode()
   == "BB".hashCode();
```

$$31 \cdot c_0 + c_1 =$$
$$= 31 \cdot (c_0 - 1) + (c_1 + 31)$$

`username` $\Longrightarrow$ **499_331**

variants of collision

Java Microbenchmark Harness

http://openjdk.java.net/projects/code-tools/jmh/

JMH :: "username" collisions :: benchmark

```java
@State(Scope.Benchmark)
public class MapPerfTest {

  Map map;
  String[] keys;

  @Setup
  public void setup() { ..... }

  @Benchmark @Threads( 1 )
  public Map fillMap() {
    for (String key : keys)
      map.put(key, key);

    return map;
  }

}
```

JMH :: "username" collisions :: benchmark

```
@BenchmarkMode(Mode.AverageTime)
@Warmup(iterations = 5, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Benchmark)
public class MapPerfTest {

  Map map;
  String[] keys;

  @Setup
  public void setup() { ... }

  @Benchmark @Threads( 1 )
  public Map fillMap() {
    // ...
  }
}
```

JMH :: "username" collisions :: benchmark

```
@BenchmarkMode(Mode.AverageTime)
@Warmup(iterations = 5, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Benchmark)
public class MapPerfTest {

  Map map;
  String[] keys;

  @Setup
  public void setup() { ... }

  @Benchmark @Threads( 1 )
  public Map fillMap() {
    // ...
  }
}
```

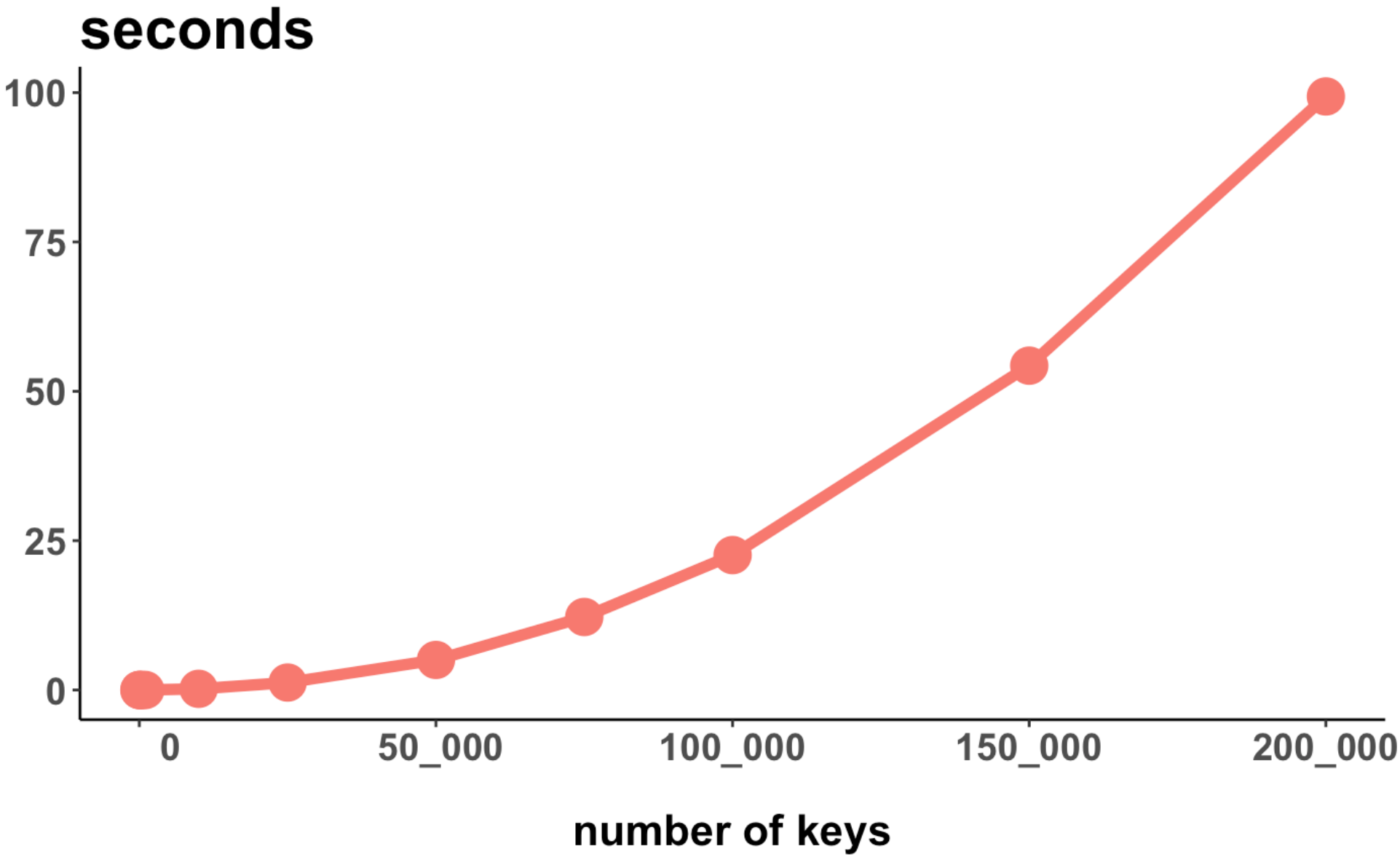
JMH :: "username" collisions :: benchmark

```
@BenchmarkMode(Mode.AverageTime)
@Warmup(iterations = 5, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Benchmark)
public class MapPerfTest {

  Map map;
  String[] keys;

  @Setup
  public void setup() { ... }

  @Benchmark @Threads( 1 )
  public Map fillMap() {
    // ...
  }
}
```

JMH :: "username" collisions :: benchmark

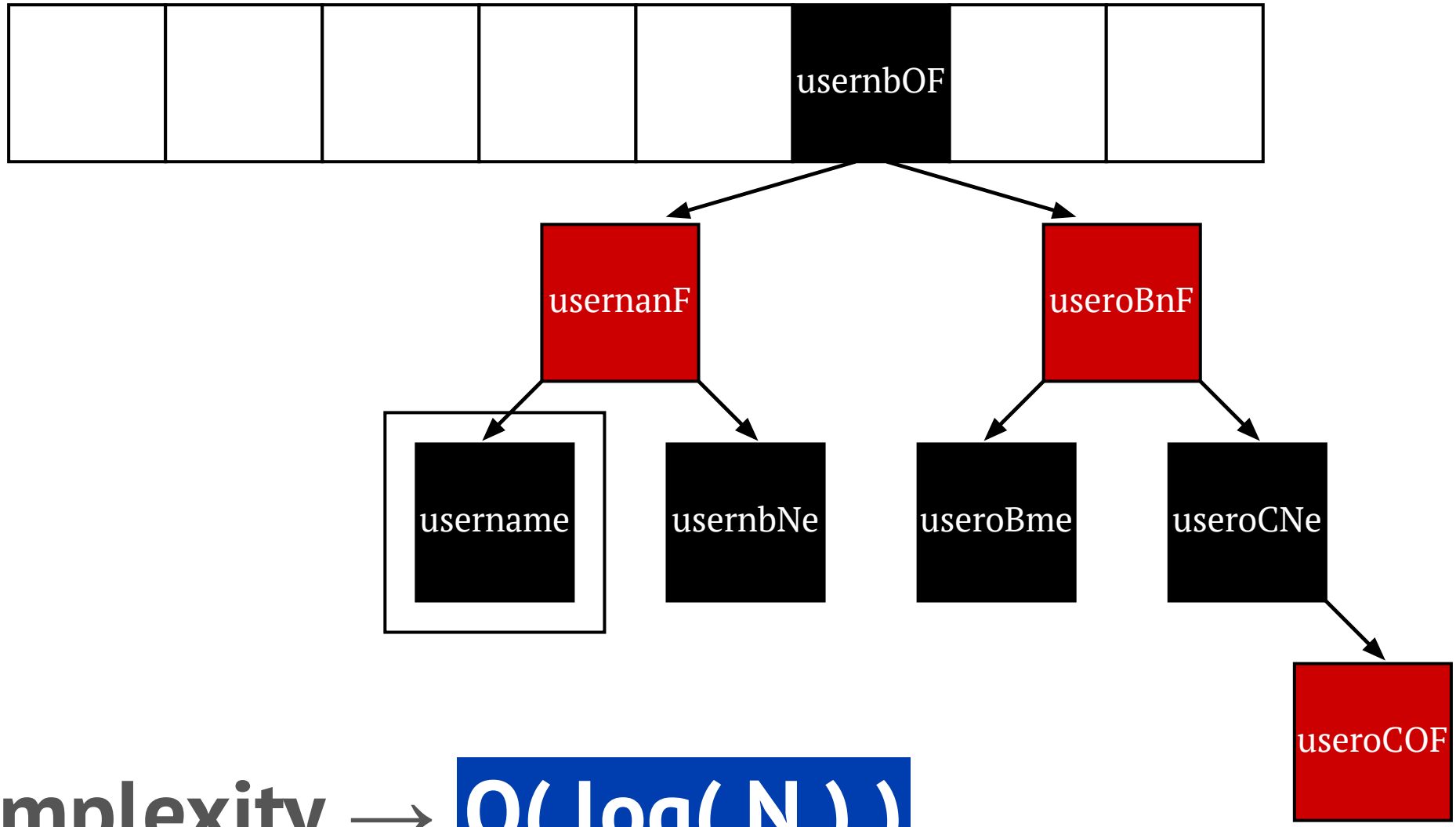
```
@State(Scope.Benchmark)
public class MapPerfTest {

  @Param( {"1", "1000", "10000", "100000", "200000"})
  int size;
  Map map;
  String[] keys;

  @Setup
  public void setup() {
    map = new HashMap<String, String>( size );
    keys = loadUsernameCollisionsFromFile( size );
  }

  @Benchmark @Threads( 1 )
  public Map fillMap() { ... }

}
```

"username" collisions



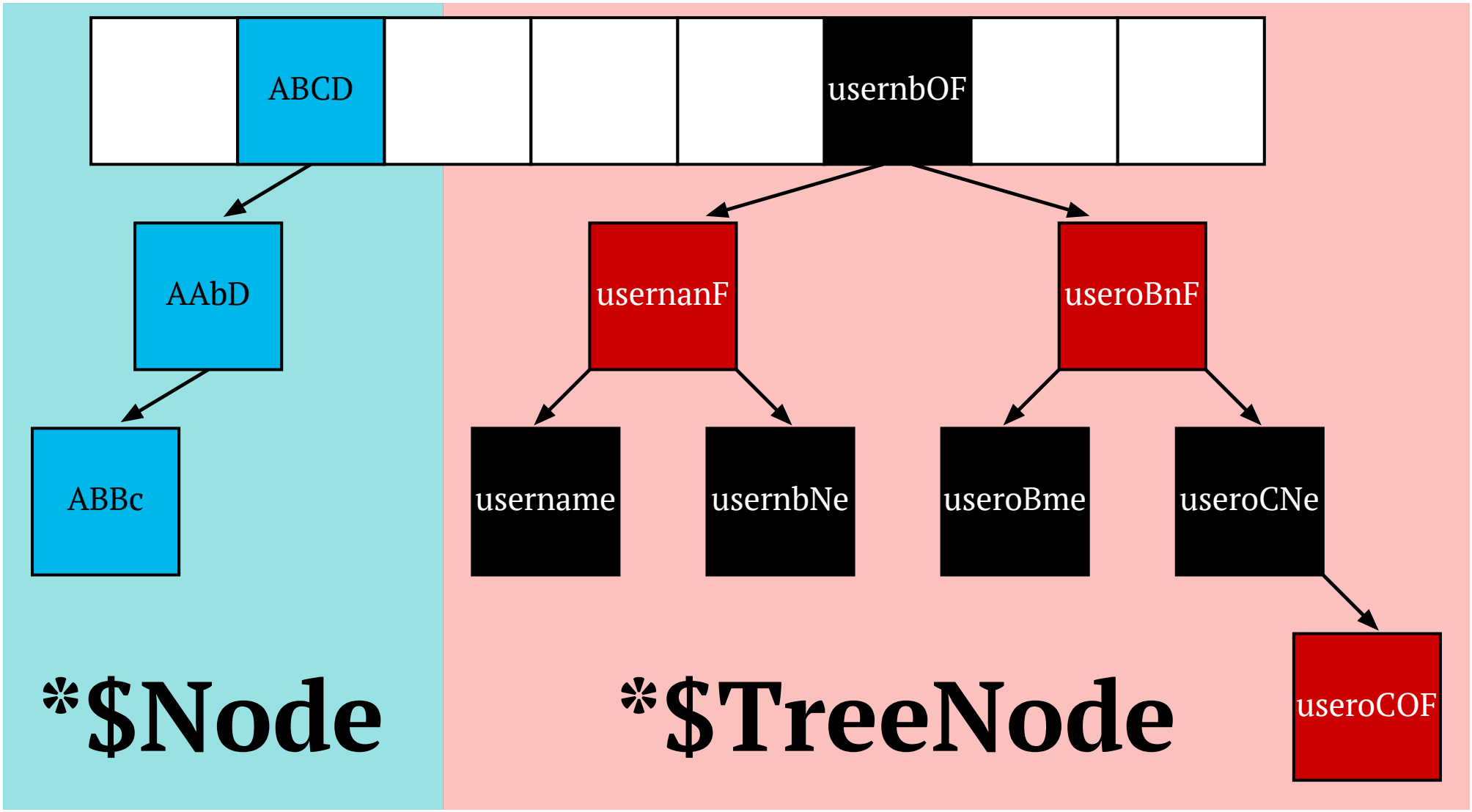
extra function is required

Extra function: compareTo

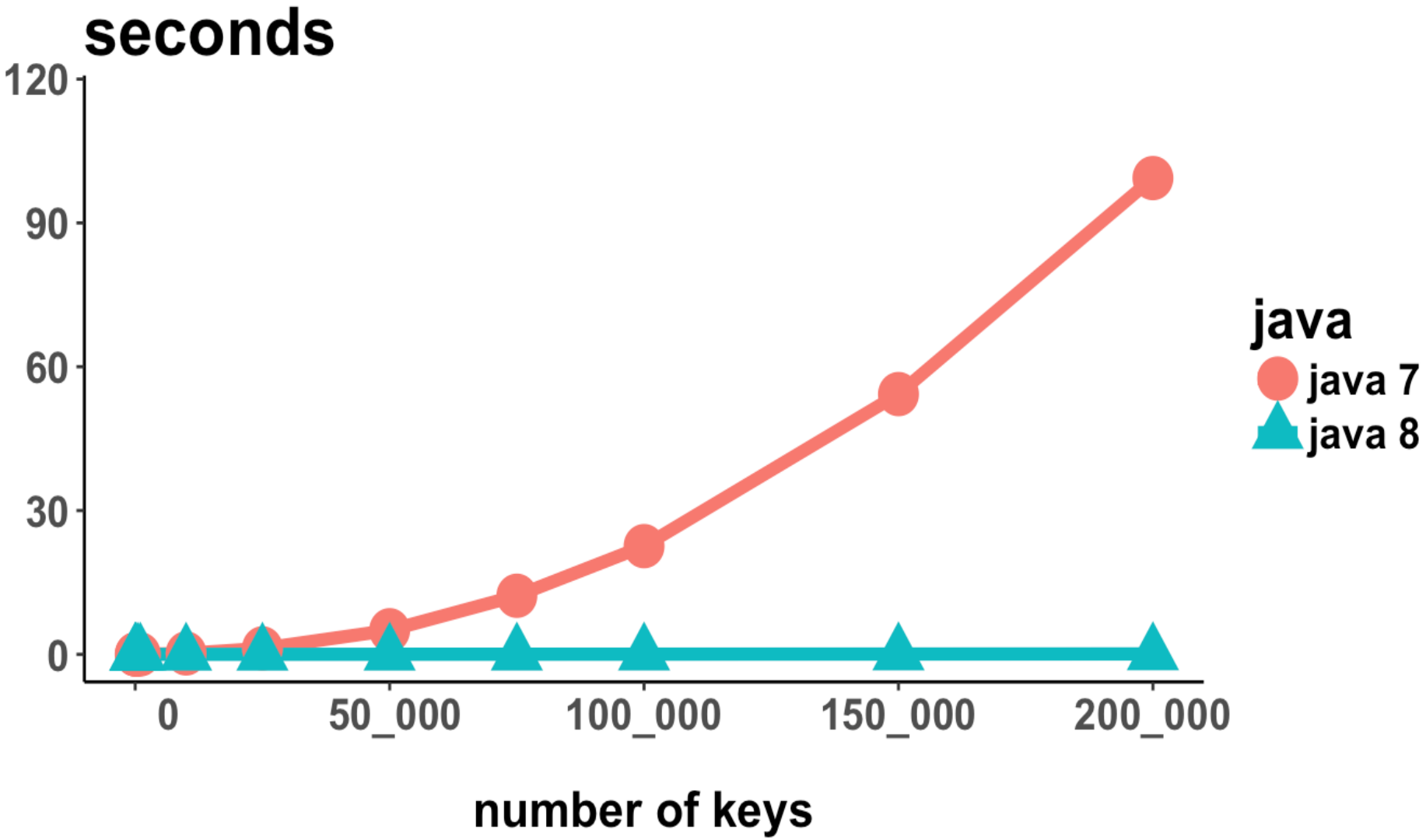


complexity → O(log(N))

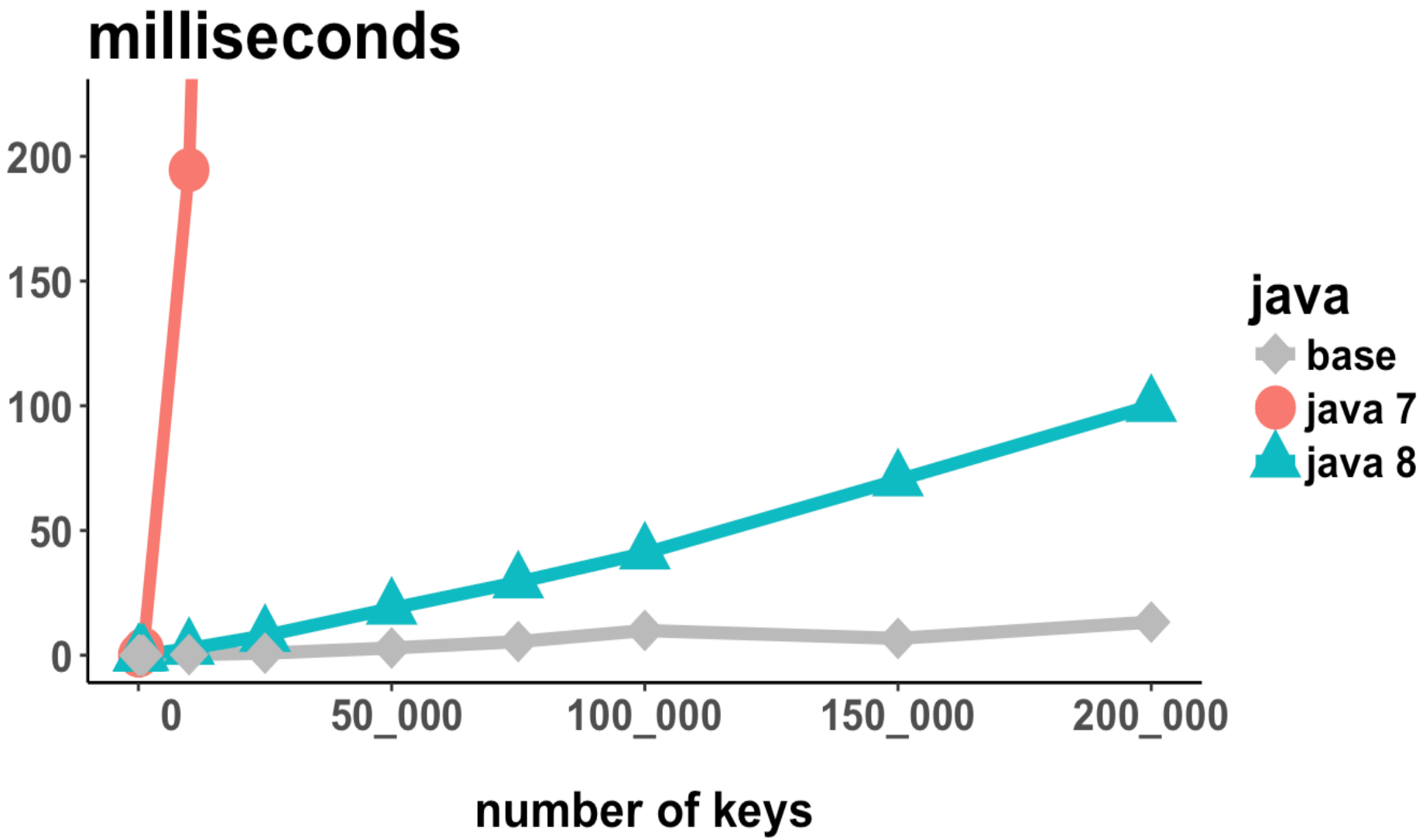
Chaining & Red-Black-Tree



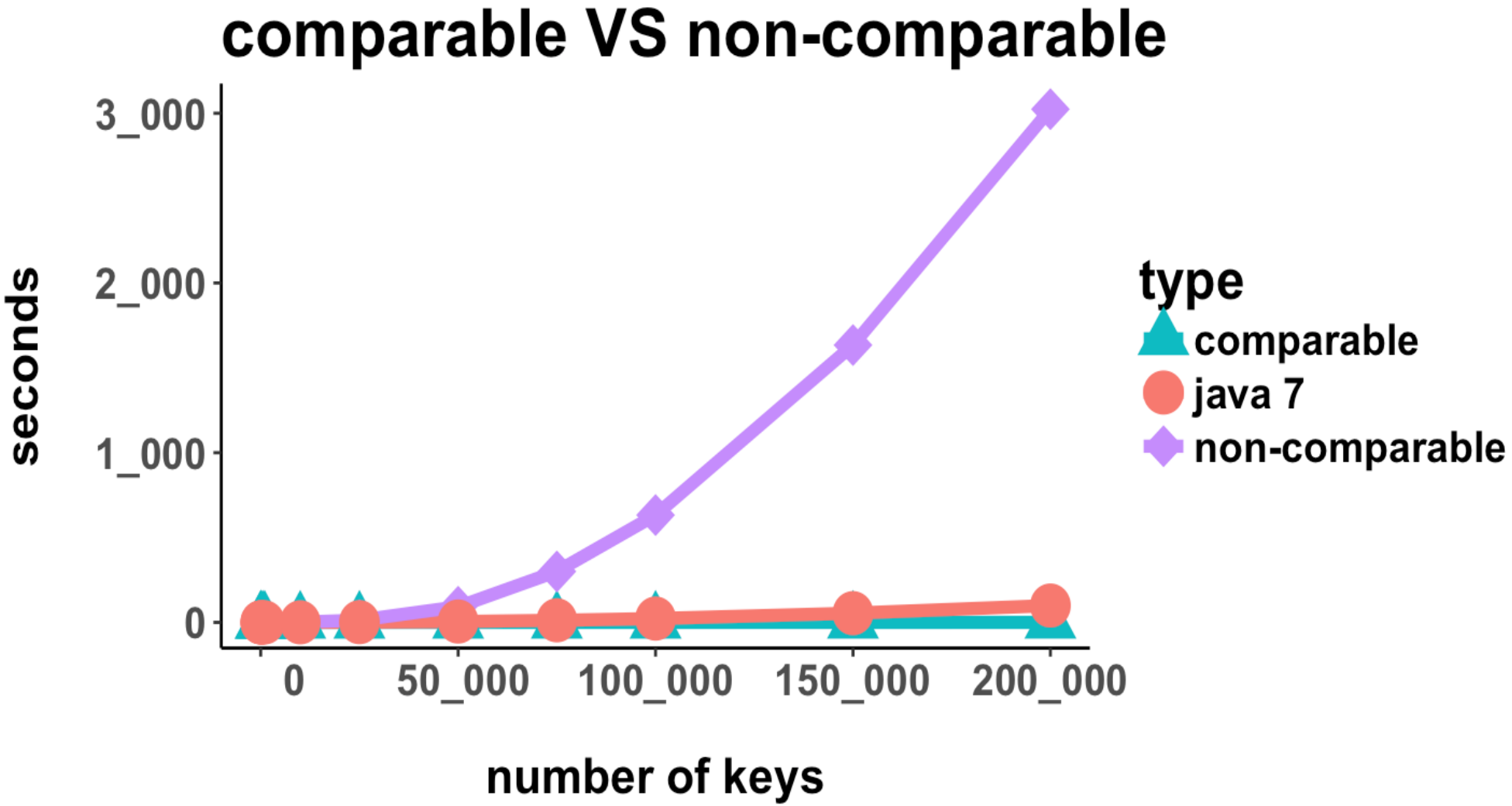
"username" collisions



"username" collisions :: 500x zoom-in



"username" collisions :: Comparable



Object.hashCode
is a leaked abstraction

```
public int hashCode(){
  return &this;
}
```

Urban Legend

Urban Legend: the source

java.lang.Object

```
public int hashCode()
```

This is typically implemented by converting the **internal address of the object** into an integer.

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Address of an object ?

```java
package java.lang;

public class Object {

    // ............

    public native int hashCode();
}
```

native method

sun.misc.Unsafe

```
public native long allocateMemory(long bytes);

public native void freeMemory(long address);

public native void setMemory(Object o,
    long offset, long bytes, byte value);

public native long getLong(Object o, long offset);

public native void putLong(Object o,
    long offset, long x);

// etc ...
```

Unsafe :: address of an object

```
private static Unsafe fetchUnsafe() {
  final Field field =
      Unsafe.class.getDeclaredField( "theUnsafe" );
  field.setAccessible( true );
  return (Unsafe) field.get( null );
}

public long getAddress(final Object object) {
  checkVMBooleanOption("UseCompressedOops", false);

  Unsafe unsafe = fetchUnsafe();

  return unsafe.getLong( new Object[]{ object },
      unsafe.arrayBaseOffset( Object[].class ));
}
```

Unsafe :: address of an object

```
private static Unsafe fetchUnsafe() {
  final Field field =
      Unsafe.class.getDeclaredField( "theUnsafe" );
  field.setAccessible( true );
  return (Unsafe) field.get( null );
}

public long getAddress(final Object object) {
  checkVMBooleanOption("UseCompressedOops", false);

  Unsafe unsafe = fetchUnsafe();

  return unsafe.getLong( new Object[]{ object },
      unsafe.arrayBaseOffset( Object[].class ));
}
```

Unsafe :: address of an object

```
private static Unsafe fetchUnsafe() {
  final Field field =
      Unsafe.class.getDeclaredField( "theUnsafe" );
  field.setAccessible( true );
  return (Unsafe) field.get( null );
}

public long getAddress(final Object object) {
  checkVMBooleanOption("UseCompressedOops", false);

  Unsafe unsafe = fetchUnsafe();

  return unsafe.getLong( new Object[]{ object },
      unsafe.arrayBaseOffset( Object[].class ));
}
```

$\Longrightarrow$ **-XX:-UseCompressedOops**

Java Object Layout

```
Object theObject = new Object();

VirtualMachine vm = VM.current();

long address = vm.addressOf( theObject );

long size = vm.sizeOf( theObject );
```

http://openjdk.java.net/projects/code-tools/jol/

JOL : sizeOf(new Pair())

```java
class Pair<F, S> {
  F first;
  S second;
}
```

```java
class Pair2<F, S> {
  F first;
  S second;
  int hashCode;
}
```

-XX:+UseCompressedOops & heap < 32Gb

-XX:+UseCompressedOops

24 b

```
class Pair<F, S> {
  F first;
  S second;
}
```

24 b

```
class Pair2<F, S> {
  F first;
  S second;
  int hashCode;
}
```

-XX:+UseCompressedOops & heap < 32Gb

-XX:+UseCompressedOops

24 b **24 b**

```
class Pair<F, S> {
  F first;
  S second;
}
```

```
class Pair2<F, S> {
  F first;
  S second;
  int hashCode;
}
```

Pair object internals:

OFFSET	SIZE	TYPE	DESCRIPTION
0	12		(object header)
12	4	Object	Pair.first
16	4	Object	Pair.second
20	**4**		**(loss due to the next object alignment)**

-XX:-UseCompressedOops OR heap ⩾ 32Gb

-XX:-UseCompressedOops

32 b

```
class Pair<F, S> {
  F first;
  S second;
}
```

40 b

```
class Pair2<F, S> {
  F first;
  S second;
  int hashCode;
}
```

-XX:-UseCompressedOops OR heap ⩾ 32Gb

-XX:-UseCompressedOops

32 b

```
class Pair<F, S> {
  F first;
  S second;
}
```

40 b

```
class Pair2<F, S> {
  F first;
  S second;
  int hashCode;
}
```

Pair2 object internals:

OFFSET	SIZE	TYPE	DESCRIPTION
0	16		(object header)
16	4	int	Pair2.hashCode
20	**4**		(**alignment/padding gap**)
24	8	Object	Pair2.first
32	8	Object	Pair2.second

-XX:-UseCompressedOops OR heap ⩾ 32Gb

-XX:-UseCompressedOops

32 b

```
class Pair<F, S> {
  F first;
  S second;
}
```

40 b

```
class Pair2<F, S> {
  F first;
  S second;
  int hashCode;
}
```

Pair2 object internals:

```
 OFFSET  SIZE   TYPE DESCRIPTION
      0    16        (object header)
     16     4    int Pair2.hashCode
     20     4      (alignment/padding gap)
     24     8 Object Pair2.first
     32     8 Object Pair2.second
```

Address and hashCode

class	`java.lang.Object`
address	`0x 07 6B A3 6D B8`
hashCode	`0x 6B A3 6D B8`
size	`16`

-XX: [redacted]

Follow the address of the object

```
final Object theObject = new Object();
final long initialAddress = getAddress( theObject );

List gcKeeper = new ArrayList();
gcKeeper.add( theObject );

long currentAddress = initialAddress;
while (initialAddress == currentAddress) {
    Object o = new Object();

    gcKeeper.add( o );

    currentAddress = getAddress( theObject );
}
```

```
-Xms256m -Xmx256m -XX:+UseSerialGC
```

Follow the address of the object

```
final Object theObject = new Object();
final long initialAddress = getAddress( theObject );

List gcKeeper = new ArrayList();
gcKeeper.add( theObject );

long currentAddress = initialAddress;
while (initialAddress == currentAddress) {
    Object o = new Object();

    gcKeeper.add( o );

    currentAddress = getAddress( theObject );
}
```

```
-Xms256m -Xmx256m -XX:+UseSerialGC
```

Follow the address of the object

```
final Object theObject = new Object();
final long initialAddress = getAddress( theObject );

List gcKeeper = new ArrayList();
gcKeeper.add( theObject );

long currentAddress = initialAddress;
while ( initialAddress == currentAddress ) {
    Object o = new Object();

    gcKeeper.add( o );

    currentAddress = getAddress( theObject );
}
```

```
-Xms256m -Xmx256m -XX:+UseSerialGC
```

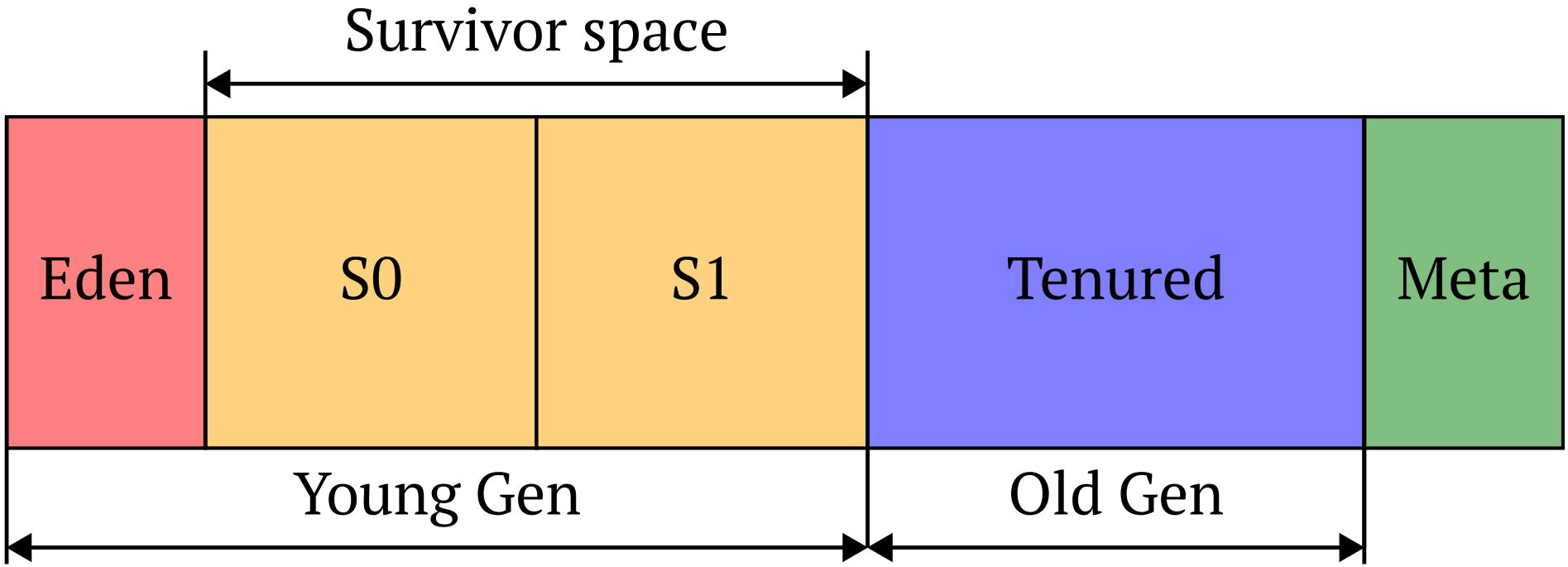
Follow the address of the object

```
final Object theObject = new Object();
final long initialAddress = getAddress( theObject );

List gcKeeper = new ArrayList();
gcKeeper.add( theObject );

long currentAddress = initialAddress;
while (initialAddress == currentAddress) {
    Object o = new Object();

    gcKeeper.add(o);

    currentAddress = getAddress( theObject );
}
```

```
-Xms256m -Xmx256m -XX:+UseSerialGC
```

demo

Weak generational hypothesis



Generational GCs: Serial, Parallel, CMS and *even* G1

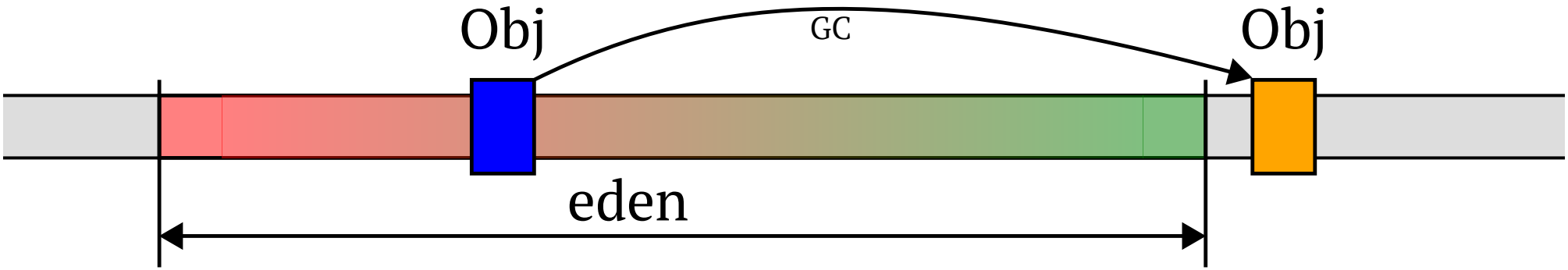
Follow the address of the object together with GC

```
final Object theObject = new Object();
final long initialAddress = getAddress( theObject );

List gcKeeper = new ArrayList();
gcKeeper.add( theObject );

long currentAddress = initialAddress;
while (initialAddress == currentAddress) {
    Object o = new Object();

    gcKeeper.add(o);

    currentAddress = getAddress( theObject );
}
```

-XX:+PrintGCDetails -Xmx256m -XX:+UseSerialGC

GC is more than just a garbage collection

```
eden space 65536K, 2% used
[0x000000012a700000,0x000000012a8db400,0x000000012e700000)
Object reallocated: from 0x12A8109F0 -> 0x12E7200F8
```



0x12A700000	0x12A8109F0	0x12E700000	0x12E7200F8
Start of Eden	original address	End of Eden	new address

Follow the hashcode

```
final Object theObject = new Object();
final long initialHashCode = theObject.hashCode();

List gcKeeper = new ArrayList();
gcKeeper.add(theObject);

long currentHashCode = initialHashCode;
while (initialHashCode == currentHashCode) {
    Object o = new Object();

    gcKeeper.add(o);

    currentHashCode = theObject.hashCode();
}
```

```
-Xms256m -Xmx256m -XX:+UseSerialGC
```

demo

Hidden property

```java
package java.lang;

public class Object {

    // other methods

    public native int hashCode();

}
```

Object memory dump

class	`java.lang.Object`
address	`0x 07 6B A3 6D B8`
hashCode	`0x 6B A3 6D B8`
size	`16`
dump	`01 B8 6D A3 6B 00 00 00`
	`E5 01 00 F8 00 00 00 00`

Intel X86: Little Endian

How many could fit in heap ?

-Xmx256m

```
long freeMemory =
  Runtime.getRuntime().freeMemory();
```

= 239_942_568

```
freeMemory / vm.sizeOf( new Object() )
```

= 14_996_410

Collisions boundary

```
final int maxCollisions = 1;
final int[] collisions = new int[maxCollisions];

final List gcKeeper = new ArrayList();
final HashIntSet uniqueHashCodes = HashIntSets.newMutableSet();

for (int collisionNo = 0; collisionNo < maxCollisions; ) {
    final Object obj = new Object();
    gcKeeper.add(obj);

    int hashCode = obj.hashCode();

    if ( ! uniqueHashCodes.add(hashCode) )
        collisions[collisionNo++] = hashCode;
}
```

-Xms256m -Xmx256m -XX:+UseSerialGC -XX:[redacted]

demo

Young generation

```
eden space 65536K, 2% used
[0x0000000012a700000, 0x0000000012a8db400, 0x00000001
Object reallocated: 0x12A8109F0 -> 0x12E7200F8
```

```
65536K / vm.sizeOf( new Object() )
```

= **4_194_304**

hashCode space

32bit → 22bit

= **4_194_304**
unique values

Collisions boundary - results

```
after 1,331,460 allocation:
hash code collision at 0x3055A90
hash code collision at 0x3055A90
hash code collision at 0x3055A90
hash code collision at 0x3055A90
hash code collision at 0x3055A90
hash code collision at 0x3055A90
hash code collision at 0x3055A90
```

hashCode → address →
→ memory allocation

Memory allocation

```
public interface Allocator {
    long malloc(long size);
}

class SimpleAllocator implements Allocator {
    private long memoryPointer;

    @Override
    public long malloc(long size) {
        long old = memoryPointer;
        memoryPointer += size;
        return old;
    }
}
```

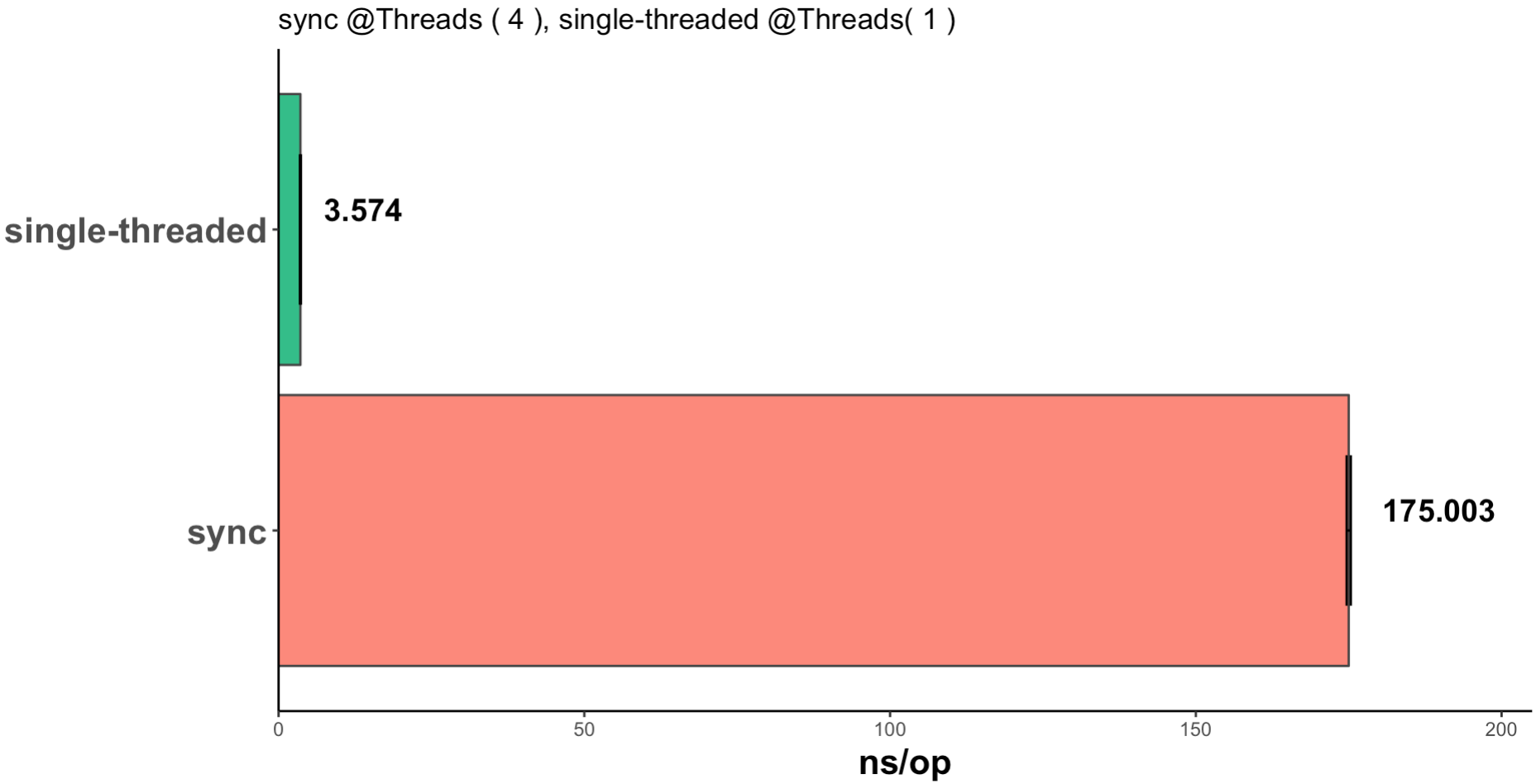
Let's fix malloc: SyncAllocator

```java
class SyncAllocator implements Allocator {
    private long memoryPointer;

    @Override
    public synchronized long malloc(long size) {
        long old = memoryPointer;
        memoryPointer += size;
        return old;
    }
}
```

SyncAllocator Performance Benchmark

```
@Benchmark @Threads( 4 )
public long syncAllocator() {
    return syncAllocator.malloc( 16 );
}
```

SyncAllocator Performance Benchmark



Can make it better ?

Compare-and-Set

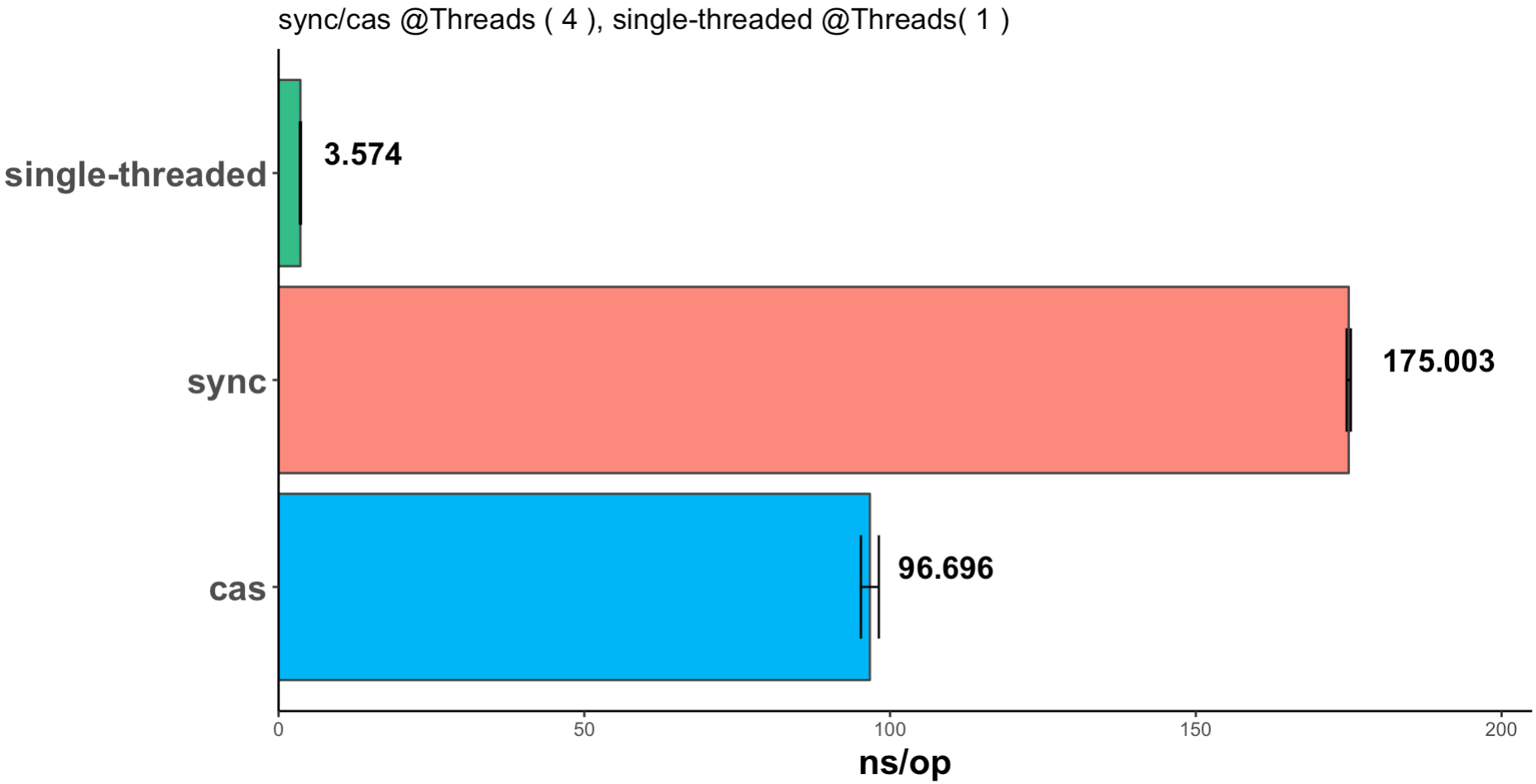
CAS Allocator

```
class CASAllocator implements Allocator {
    private final AtomicLong memoryPointer =
        new AtomicLong();

    @Override
    public long malloc(long size) {
        return memoryPointer.getAndAdd( size );
    }
}
```

Allocators Performance Benchmark

```
@Benchmark @Threads( 4 )
public long casAllocator() {
    return casAllocator.malloc( 16 );
}
```

Allocators Performance Benchmark

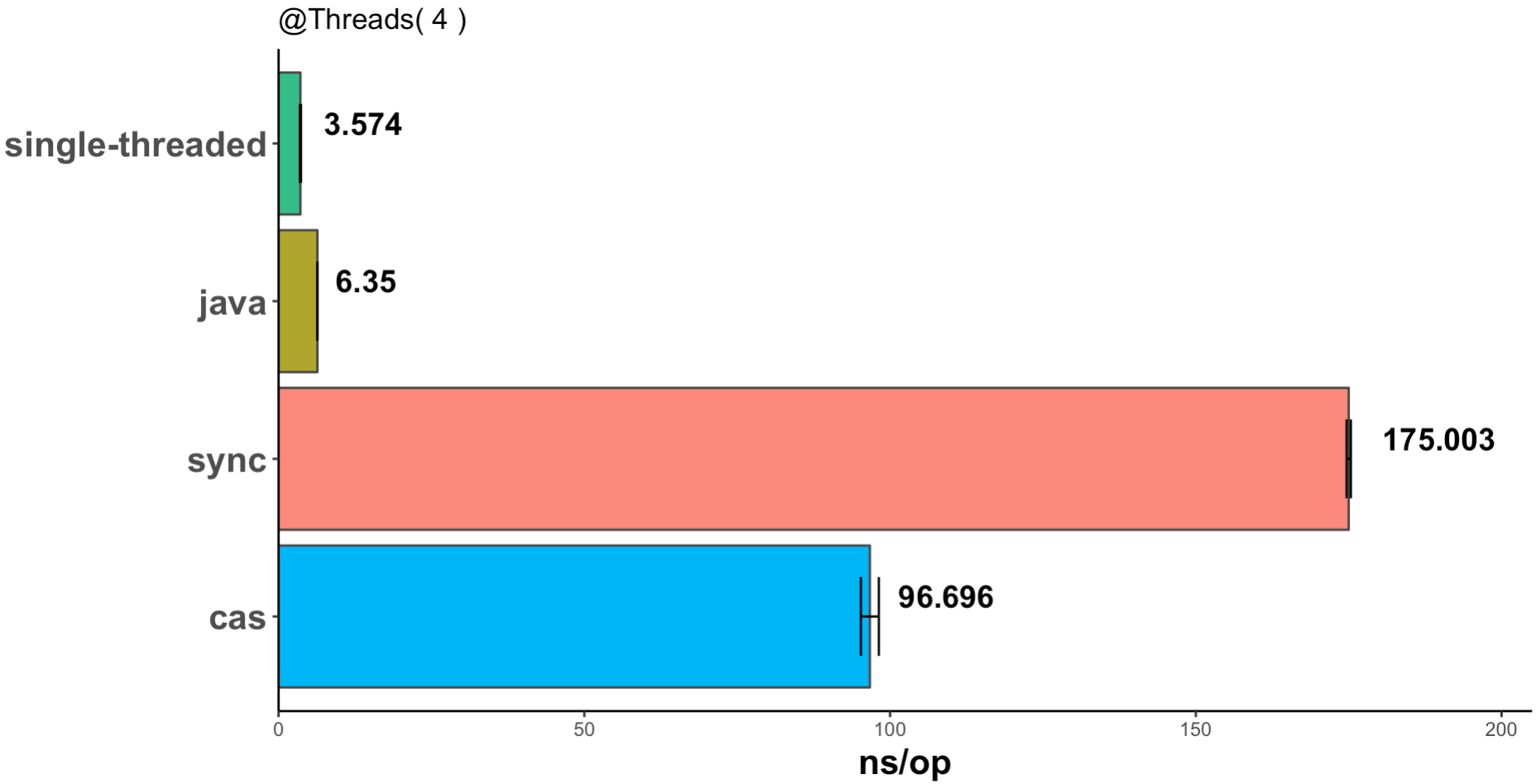


faster ← **slower**

Java Allocation

```
@Benchmark @Threads( 4 )
public Object javaAllocation() {
    return new Object();
}
```

Java Allocation

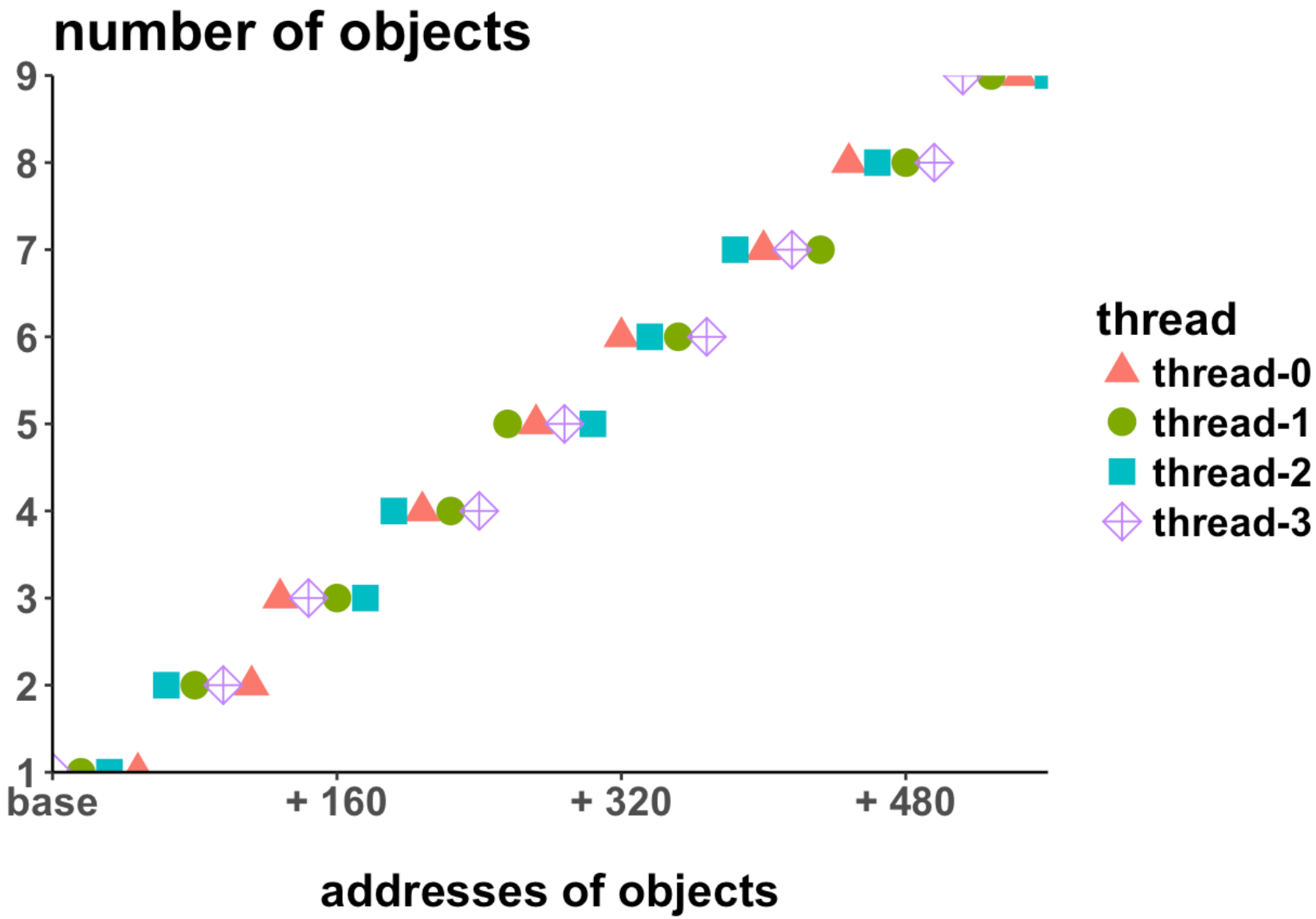


faster ← **slower**

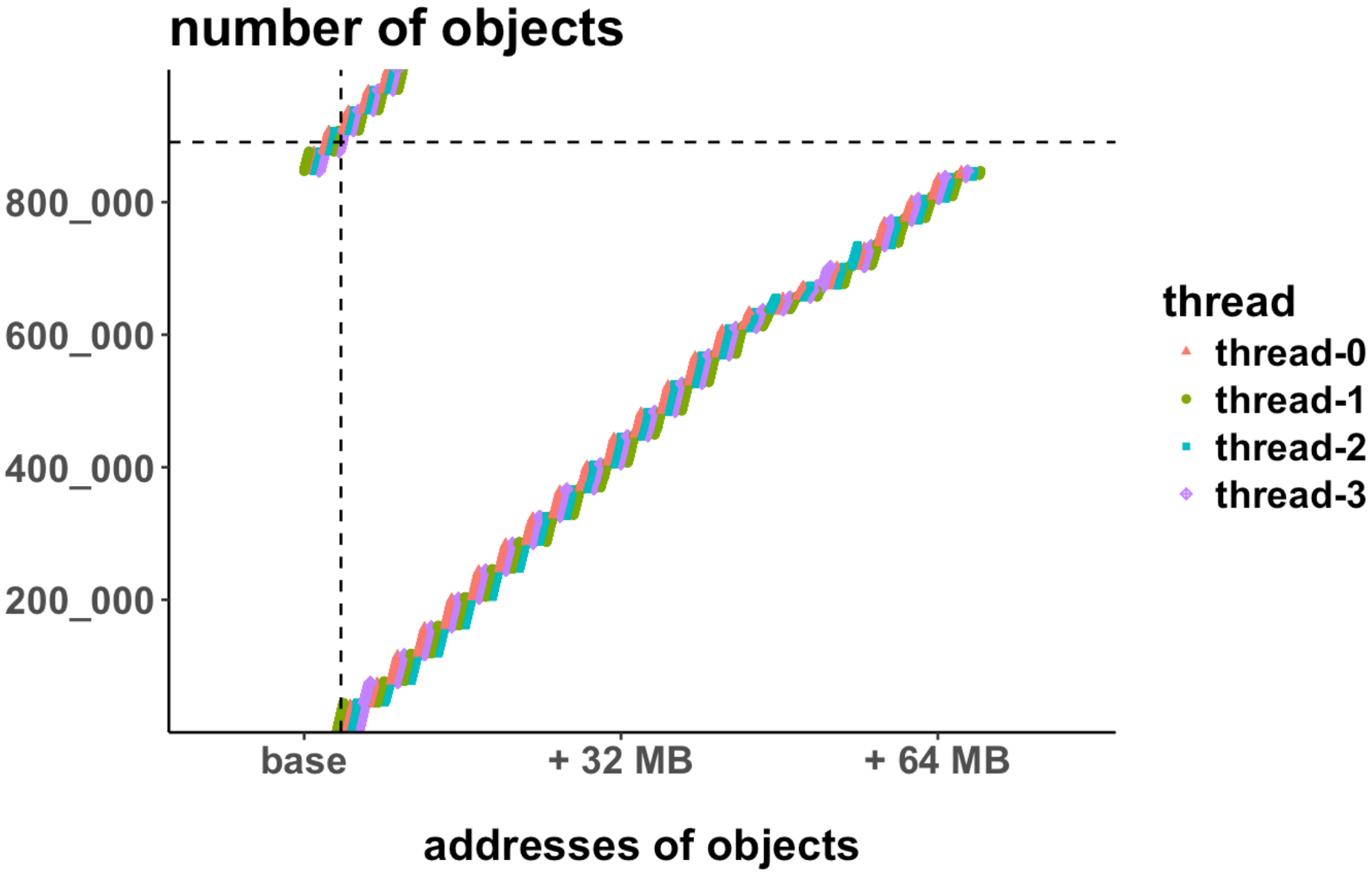
Can make it even better ?

hashCode distribution over threads

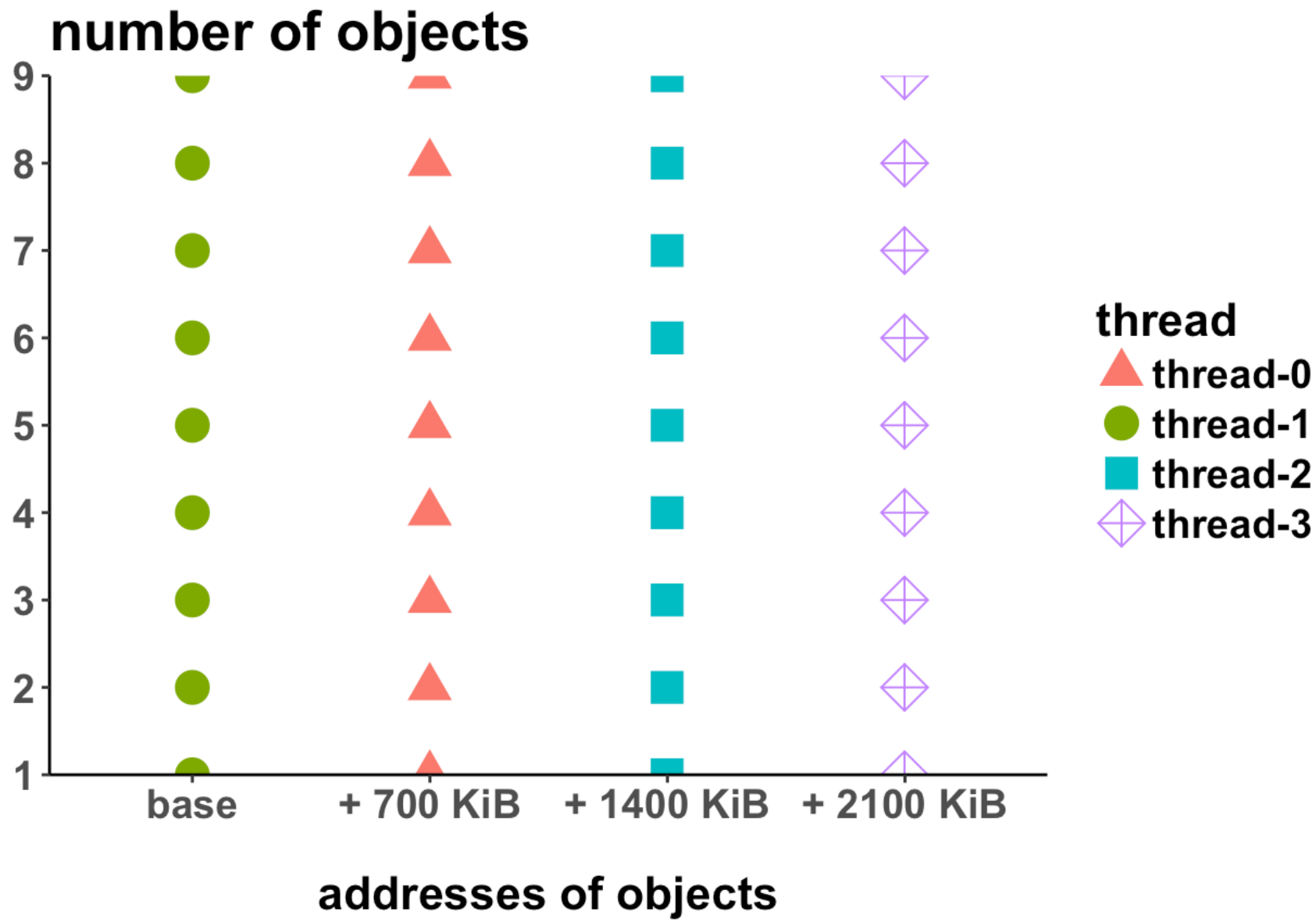
hashCode distribution : Expectations



hashCode distribution over 4 threads



The good news



Let's grab HUGE pieces of memory !

```
public class TLABLikeAllocator implements Allocator {
  private static final long SIZE = 1024L * 1024L;

  private final AtomicLong memoryPointer = new AtomicLong();
  private final ThreadLocal<AddressHolder> threadLocal =
       ThreadLocal.withInitial(() -> new AddressHolder());

  public long malloc( long size ) {
    AddressHolder addressHolder = threadLocal.get();
    while( true ) {
      if (addressHolder.value + size <= addressHolder.maxValue) {
          long old = addressHolder.value;
          addressHolder.value += size;
          return old;
      }

      long value = memoryPointer.getAndAdd( SIZE );
      addressHolder.value = value;
      addressHolder.maxValue = value + SIZE;
    }
  }
}
```

Let's grab HUGE pieces of memory !

```
public class TLABLikeAllocator implements Allocator {
  private static final long SIZE = 1024L * 1024L;

  private final AtomicLong memoryPointer = new AtomicLong();
  private final ThreadLocal<AddressHolder> threadLocal =
       ThreadLocal.withInitial(() -> new AddressHolder());

  public long malloc( long size ) {
    AddressHolder addressHolder = threadLocal.get();
    while( true ) {
      if (addressHolder.value + size <= addressHolder.maxValue) {
          long old = addressHolder.value;
          addressHolder.value += size;
          return old;
      }

      long value = memoryPointer.getAndAdd( SIZE );
      addressHolder.value = value;
      addressHolder.maxValue = value + SIZE;
    }
  }
}
```

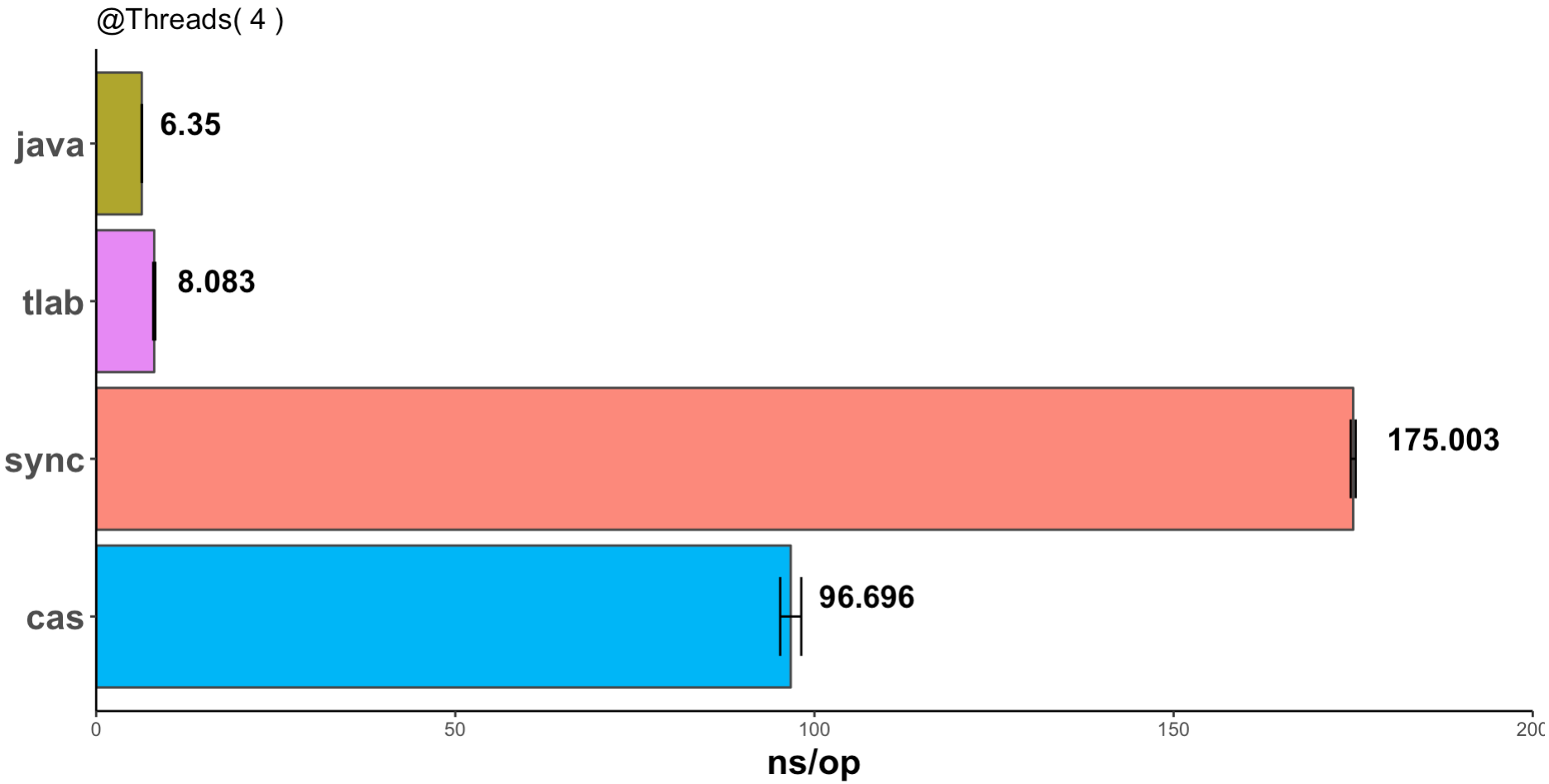
Let's grab HUGE pieces of memory !

```java
public class TLABLikeAllocator implements Allocator {
  private static final long SIZE = 1024L * 1024L;

  private final AtomicLong memoryPointer = new AtomicLong();
  private final ThreadLocal<AddressHolder> threadLocal =
      ThreadLocal.withInitial(() -> new AddressHolder());

  public long malloc( long size ) {
    AddressHolder addressHolder = threadLocal.get();
    while( true ) {
      if (addressHolder.value + size <= addressHolder.maxValue) {
        long old = addressHolder.value;
        addressHolder.value += size;
        return old;
      }

      long value = memoryPointer.getAndAdd( SIZE );
      addressHolder.value = value;
      addressHolder.maxValue = value + SIZE;
    }
  }
}
```

Allocators Performance Benchmark

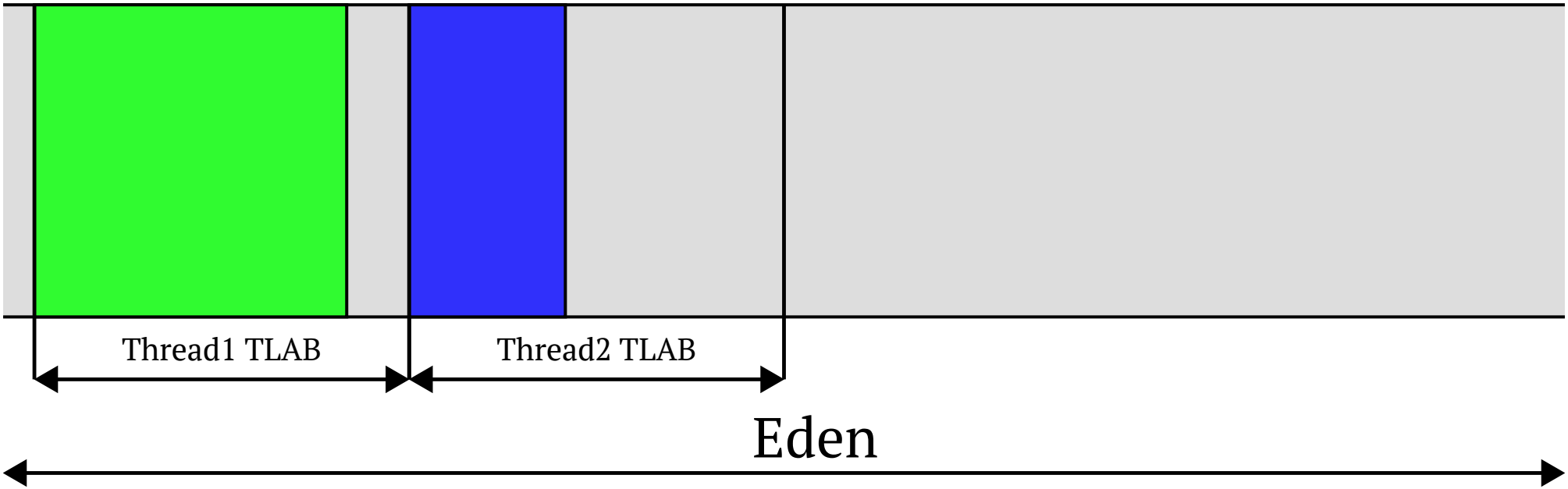
```
@Benchmark @Threads( 4 )
public long tlabAllocator() {
    return tlabAllocator.malloc( 16 );
}
```

Allocators Performance Benchmark

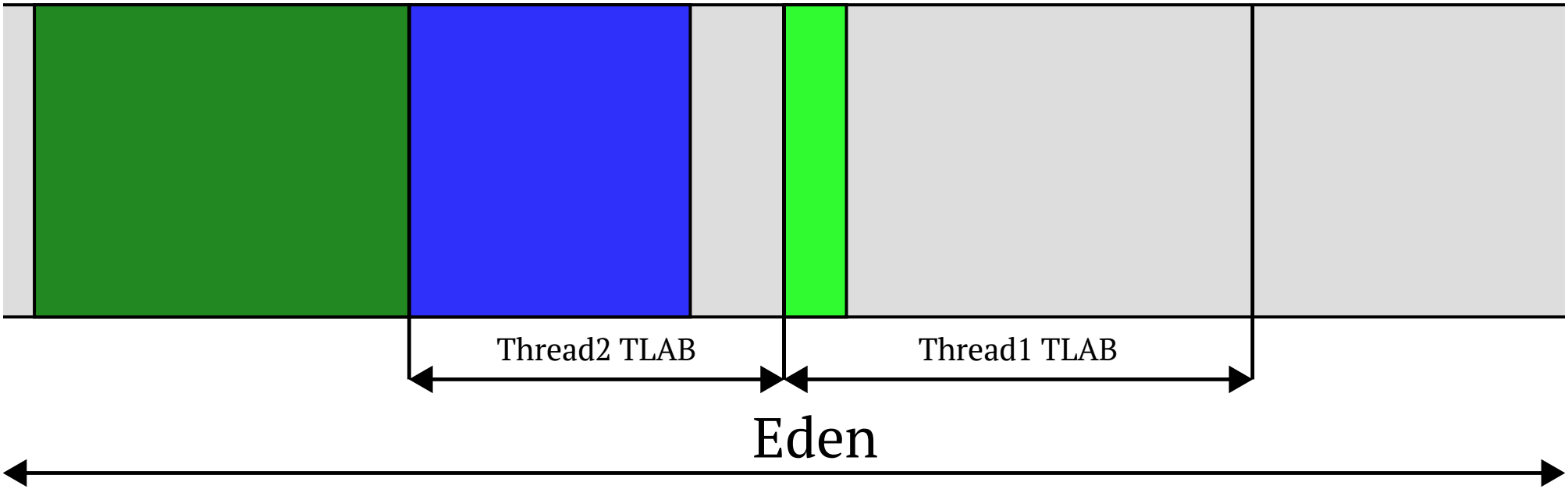


faster ← **slower**

Thread Local Allocation Buffer



Thread Local Allocation Buffer

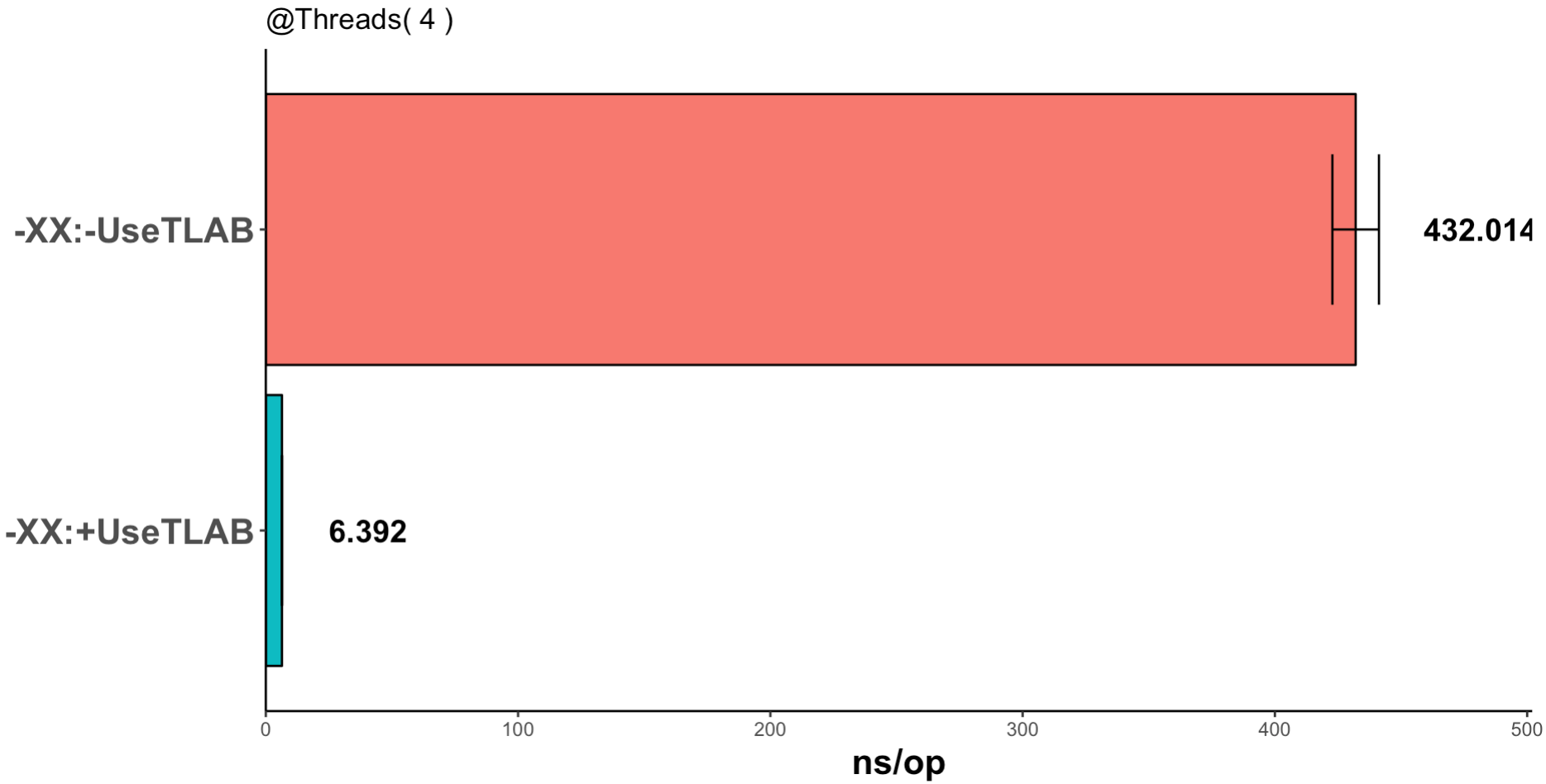


Cost of TLAB

```
@Benchmark @Threads( 4 )
public Object allocate() {
    return new Object();
}
```

-XX:+UseTLAB vs -XX:-UseTLAB

Cost of TLAB



faster ← **slower**

32bit → 20bit

What if it is a Random ?

Birthday problem

$$p_{uniq}(n) = \left(1 - \frac{1}{d}\right) \cdot \left(1 - \frac{2}{d}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{d}\right) =$$

$$= \frac{d}{d^n \cdot (d-n)!}$$

$$n \approx \sqrt{2d \cdot ln(\frac{1}{1 - p_{uniq}})}$$

using $d = 2^{32},\ p = 0.5 \Rightarrow n \approx 77162$

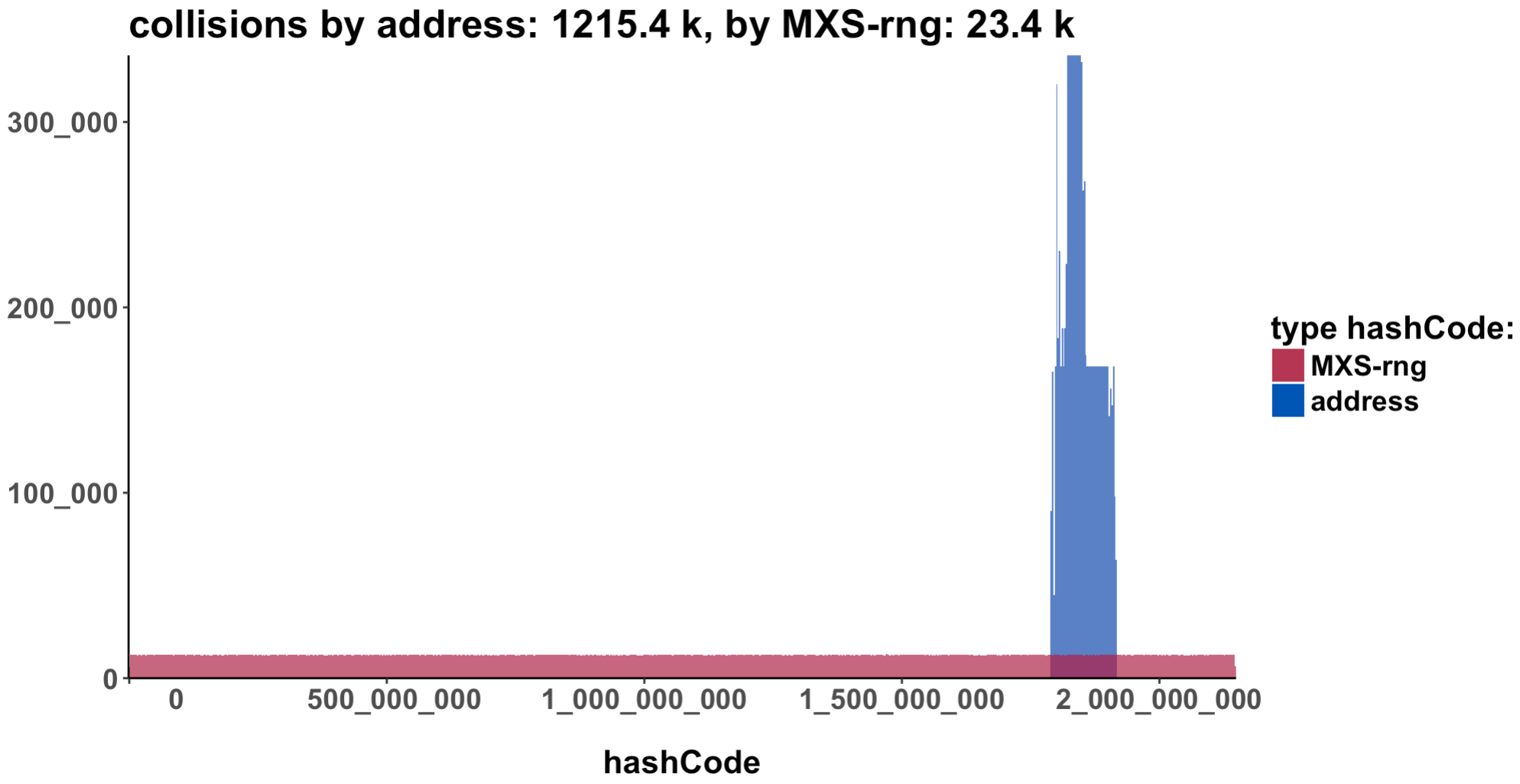
-XX:hashCode

-XX:hashCode=k	Type
0	Park-Miller RNG
1	fn(address of object, global state)
2	const 1
3	incremental counter
4	address of object
5	Marsaglia xor-shift RNG ***by default*** *in java 8*

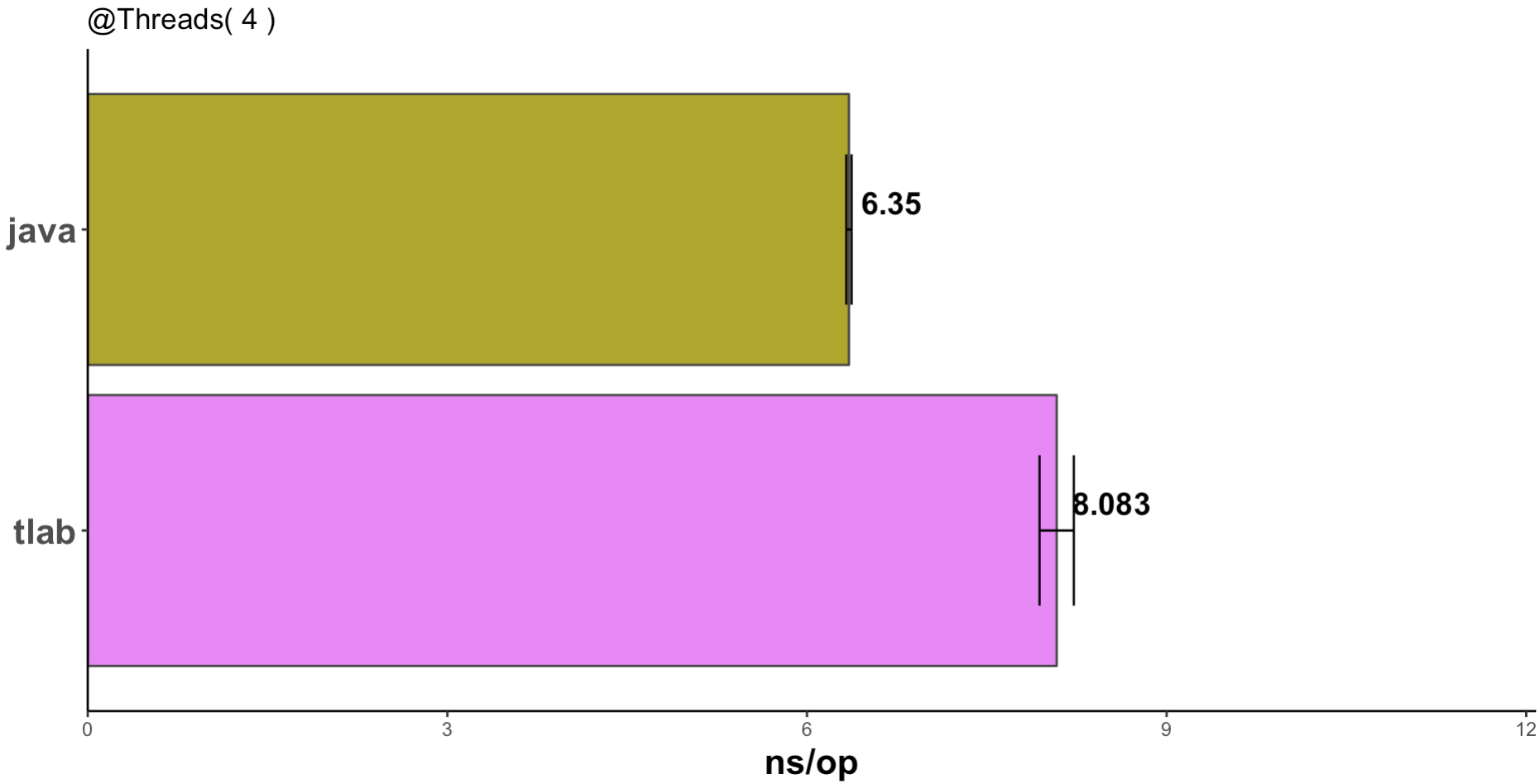
Memory dump of object -XX:hashCode=5

class	`java.lang.Object`
address	`0x 07 6B A3 6D B8`
size	`16`
hashCode	`0x 2A AE 91 90`
dump	`01 90 91 AE 2A 00 00 00`
	`E5 01 00 F8 00 00 00 00`

Distribution of 10 millions objects in 10 threads



has hashCode been calculated ?

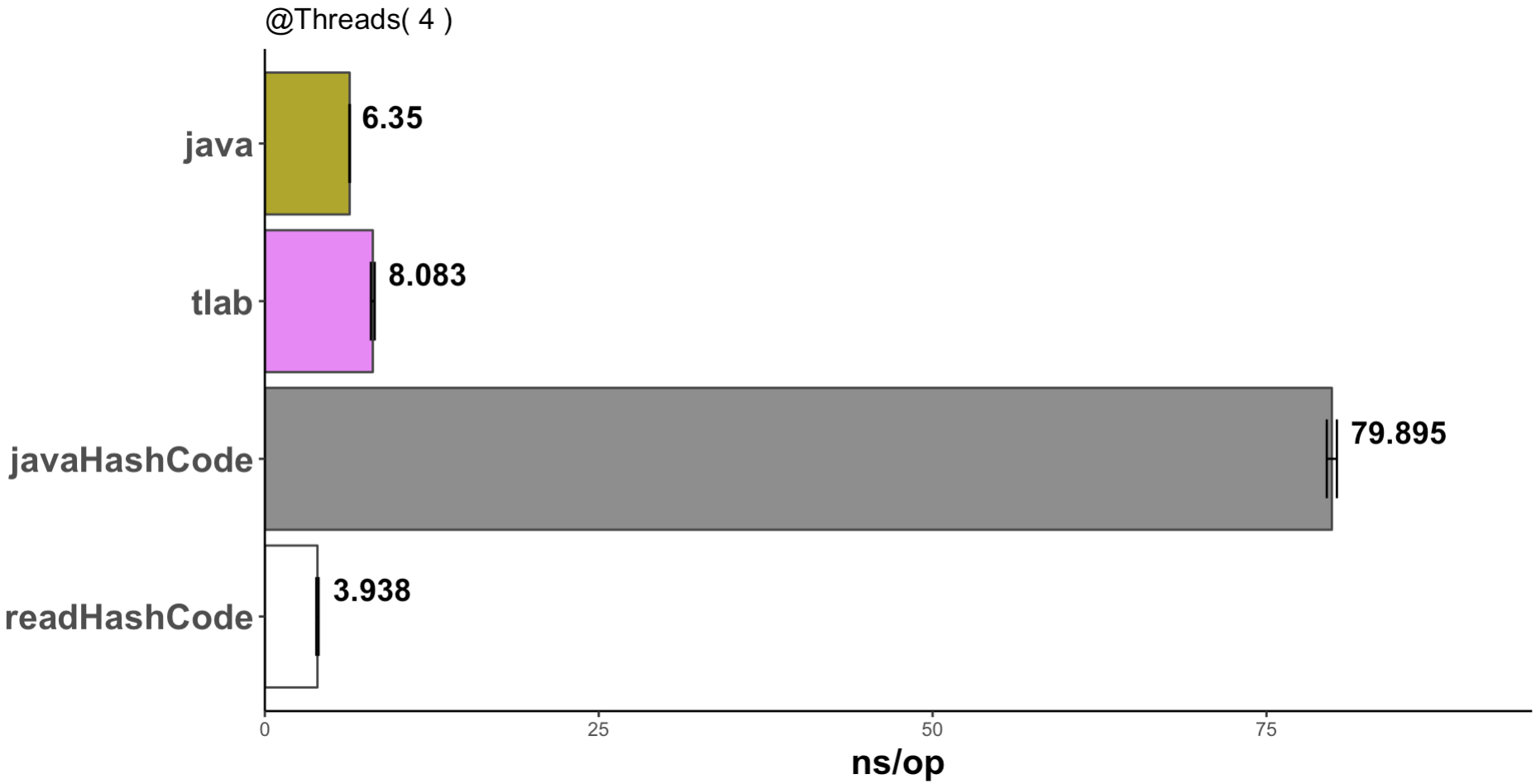


faster ← **slower**

Object.hashCode() benchmark

```
Object theObject = new Object();

@Benchmark @Threads( 4 )
public Object javaHashCode() {
    Object object = new Object();
    object.hashCode();
    return object;
}

@Benchmark @Threads( 4 )
public int readHashCode() {
    return theObject.hashCode();
}
```

Object.hashCode() benchmark



Memory dump of object right after creation

```
Object object = new Object();

dump( object );
```

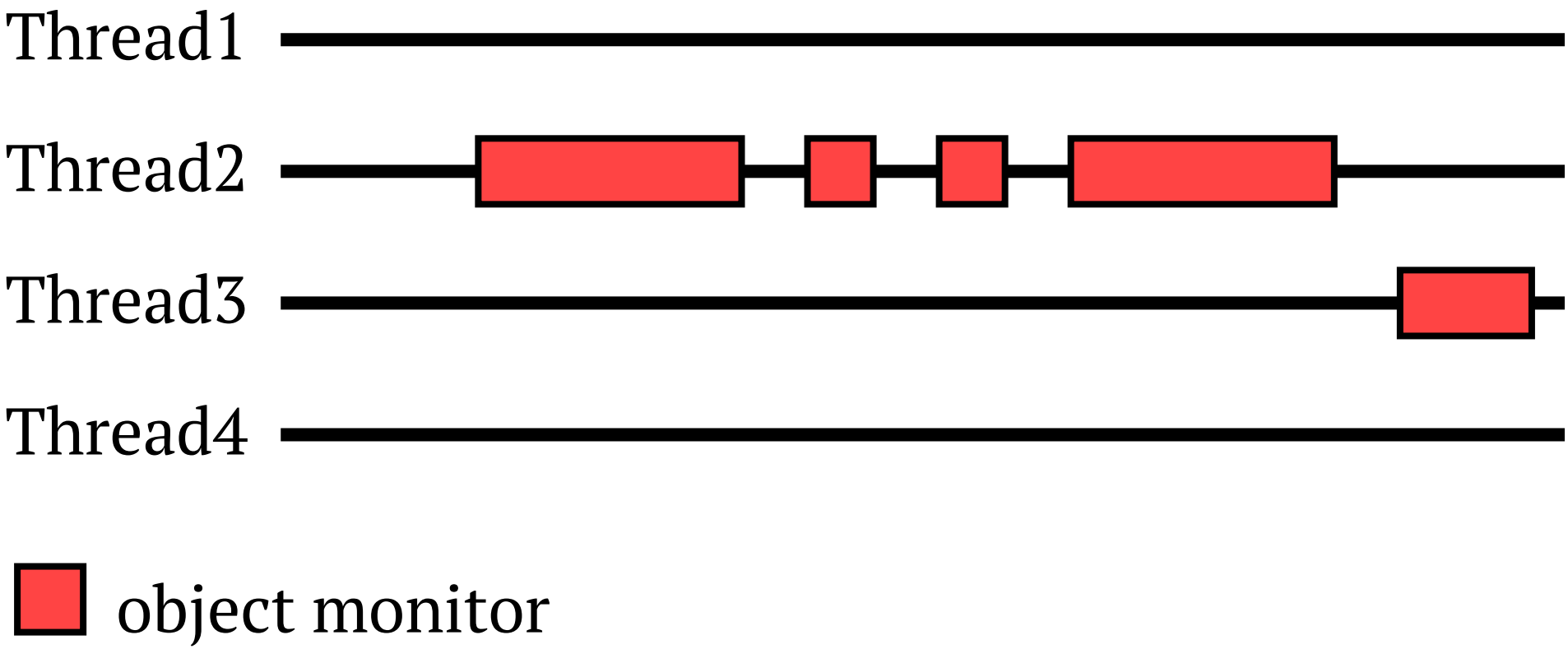
dump

05	00	00	00	00	00	00	00
E5	01	00	F8	00	00	00	00

```java
public class Object {
  //// other methods
  public final native void notify();

  public final native void wait(long timeout)
      throws InterruptedException;
}
```

monitor

Biased Locking



Biased Locking demo:

```
synchronized (object) {
    object.notifyAll();
}
dump( object );
```

tid	00	00	00	00	02	13	80	00
dump	05	80	13	02	00	00	00	00
	E5	01	00	F8	00	00	00	00

StringBufferPerfTest

```
StringBuffer buf = new StringBuffer();

@Benchmark @Thread(1)
public String bufferToString() {
  return buf.toString();
}
```

`-XX:+UseBiasedLocking` VS `-XX:-UseBiasedLocking`

```
-XX:BiasedLockingStartupDelay=0
```

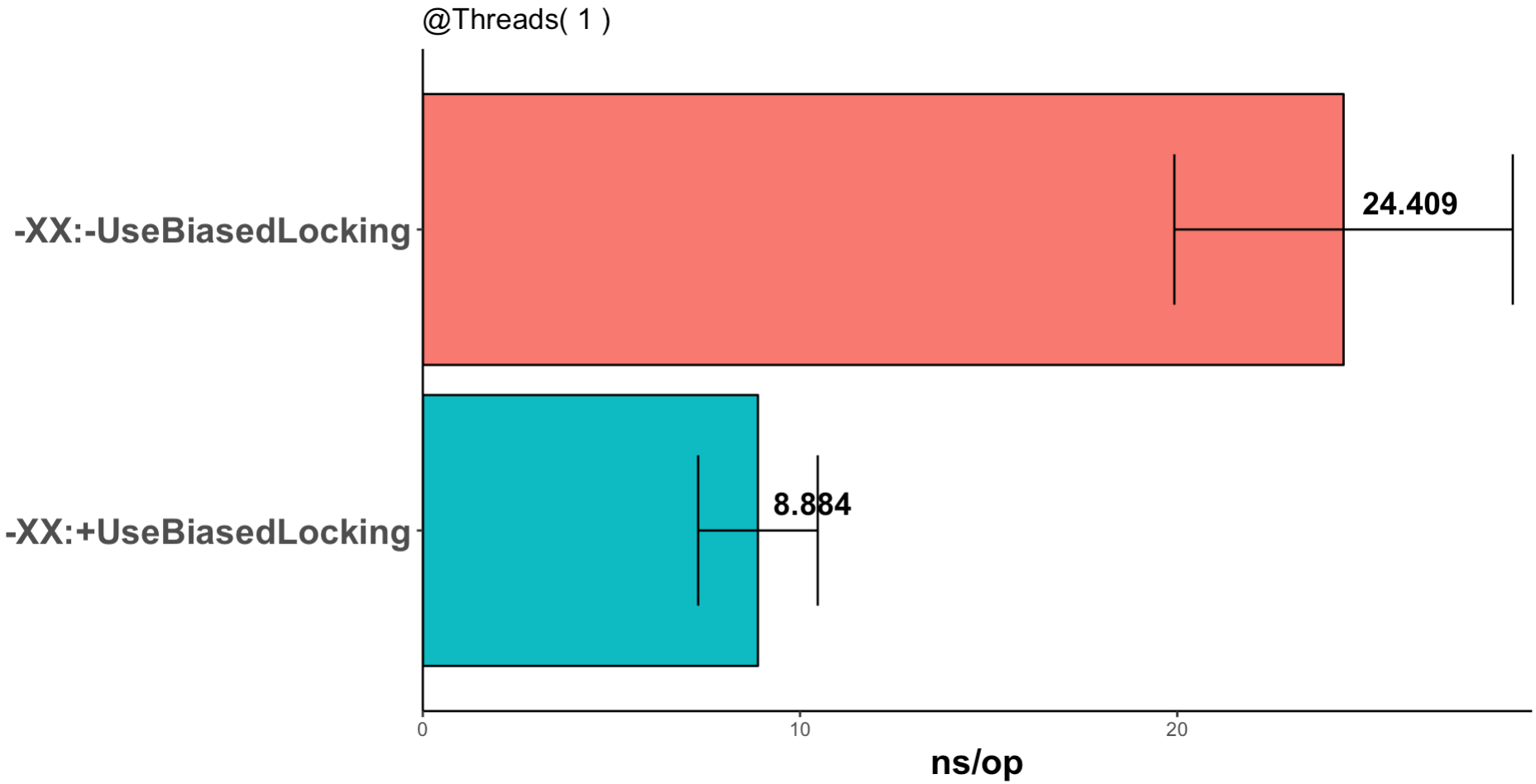
StringBufferPerfTest

```
StringBuffer buf = new StringBuffer();

@Benchmark @Thread(1)
public String bufferToString() {
  return buf.toString();
}
```

`-XX:+UseBiasedLocking` VS `-XX:-UseBiasedLocking`

`-XX:BiasedLockingStartupDelay=0`

StringBufferPerfTest results



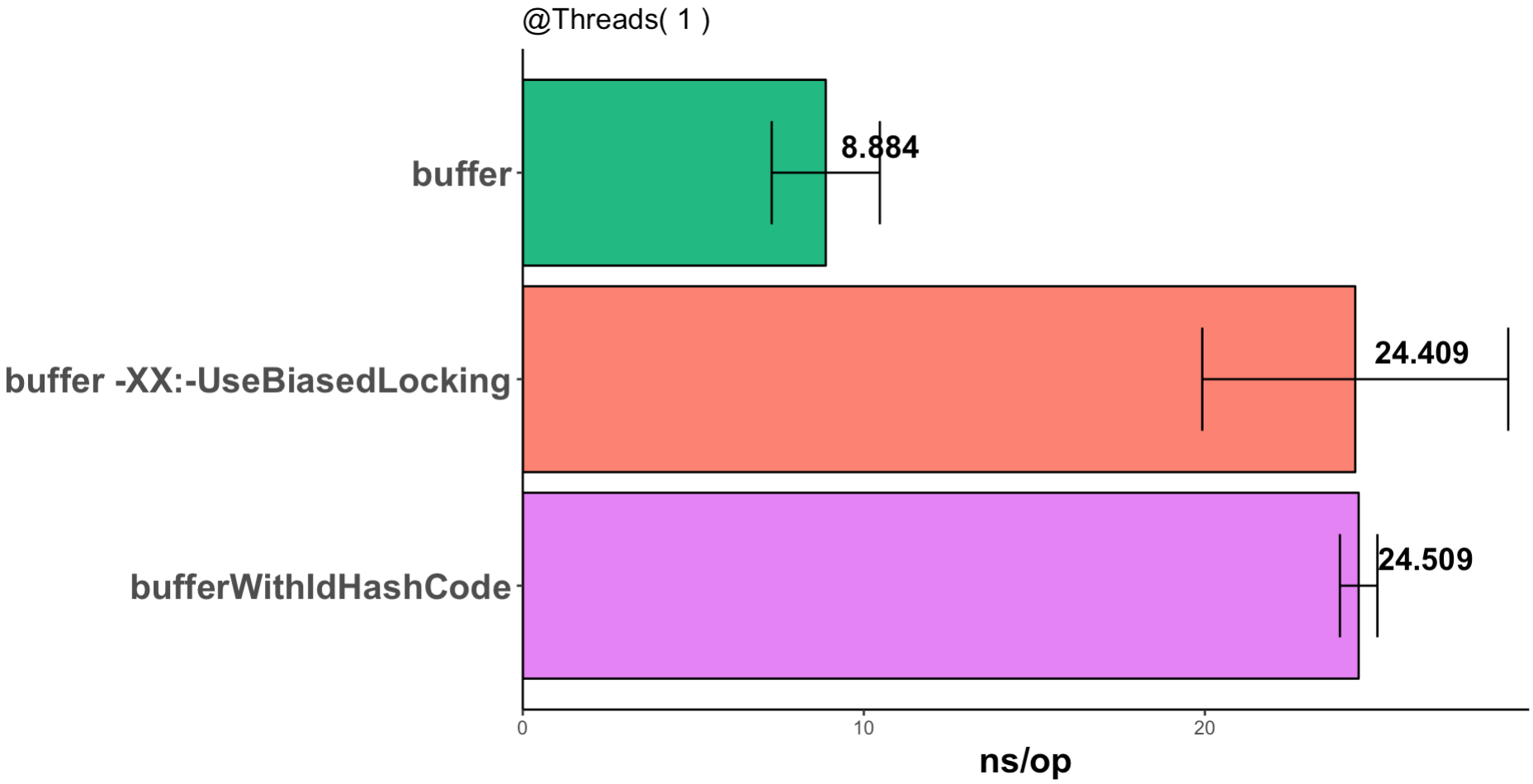
identityHashCode

```
StringBuffer buf = new StringBuffer();
StringBuffer bufferWithIdHashCode =
    new StringBuffer();

@Setup
public void setup(Blackhole bh) {
    bh.consume( System.identityHashCode(
        bufferWithIdHashCode ) );
}

@Benchmark  @Thread(1)
public String bufferWithIdHashCode() {
  return bufferWithIdHashCode.toString();
}

@Benchmark @Thread(1)
public String buffer() {
  return buf.toString();
}
```

identityHashCode

```
StringBuffer buf = new StringBuffer();
StringBuffer bufferWithIdHashCode =
   new StringBuffer();

@Setup
public void setup(Blackhole bh) {
    bh.consume( System.identityHashCode(
        bufferWithIdHashCode ) );
}

@Benchmark  @Thread(1)
public String bufferWithIdHashCode() {
  return bufferWithIdHashCode.toString();
}

@Benchmark  @Thread(1)
public String buffer() {
  return buf.toString();
}
```

identityHashCode results



Revoke Biased Locking:

```
int idHashCode
  = System.identityHashCode( object );

dump( object );
```

idHashCode	02	53	0C	12				
dump	01	12	0C	53	02	00	00	00
	E5	01	00	F8	00	00	00	00

Conclusion

- **Hash data-structures** are the fastest ones
- **Override `hashCode` and `equals`**
  - and even more **implement `compareTo`**
  - investigate your hash-functions
- hashCode is not **address of object**
  - **GC**, **TLAB**, don't saturate **`identityHashCode`**
- **Meten is weten** (Dutch) «Measurement is knowledge»

slides and examples: github.com/vladimirdolzhenko/hashCodeLegend

email: vladimir.dolzhenko@ihsmarkit.com

email: vladimir.dolzhenko@gmail.com

twitter: @dolzhenko

Contacts

thanks: @VladimirSitnikv @dj_begemot @AndreiPangin @gvsmirnov @i_sopov