Does Java need inline(value) types?

What project Valhalla can bring to Java from a performance perspective.

Sergey Kuksenko

Java Platform Group
Oracle
October, 2019

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

Who am I?

- Java/JVM Performance Engineer at Oracle, @since 2010
- Java/JVM Performance Engineer, @since 2005
- Java/JVM Engineer, @since 1996

What is Valhalla?

Valhalla Goals

- Provide denser memory layout (inline/value types)
- Specialized generics (including primitive, value types)
- Smooth library migration
- JVM cleanup (e.g. Nestmates a.k.a. JEP-181)

Object Identity is the root of all evil

Identity (object-oriented programming)

From Wikipedia, the free encyclopedia

An **identity** in object-oriented programming, object-oriented design and object-oriented analysis describes the property of objects that distinguishes them from other objects. This is closely related to the philosophical concept of identity.

In philosophy, **identity**, from Latin: *identitas* ("sameness"), is the relation each thing bears only to itself. The notion of identity gives rise to many philosophical problems, including the identity of indiscernibles, and questions about change and personal identity over time.

Identity gives

- Indirection
- Allocation in heap
- Nullability
- Mutability
- Reference equality (==)
- Locking
- Puzzlers, e.g.

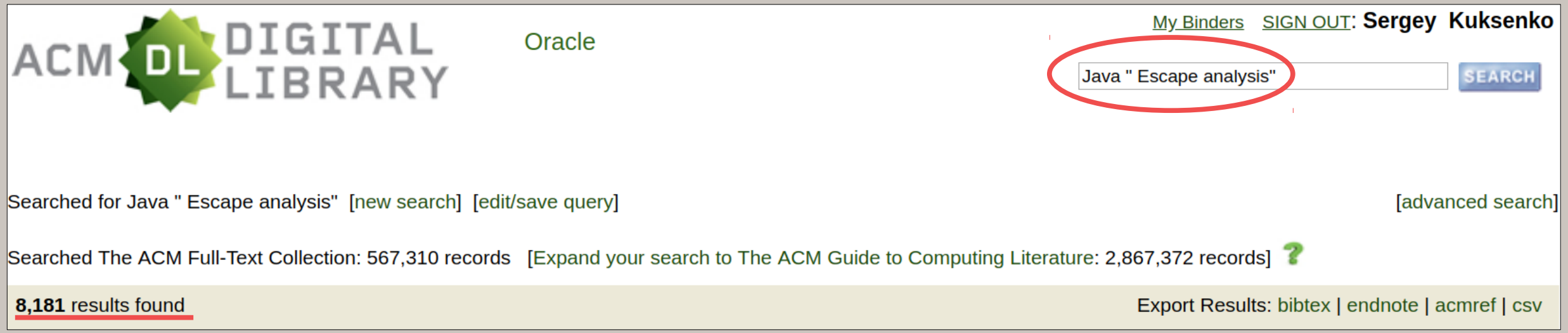
```
Integer.valueOf(42) == Integer.valueOf(42)
```

but

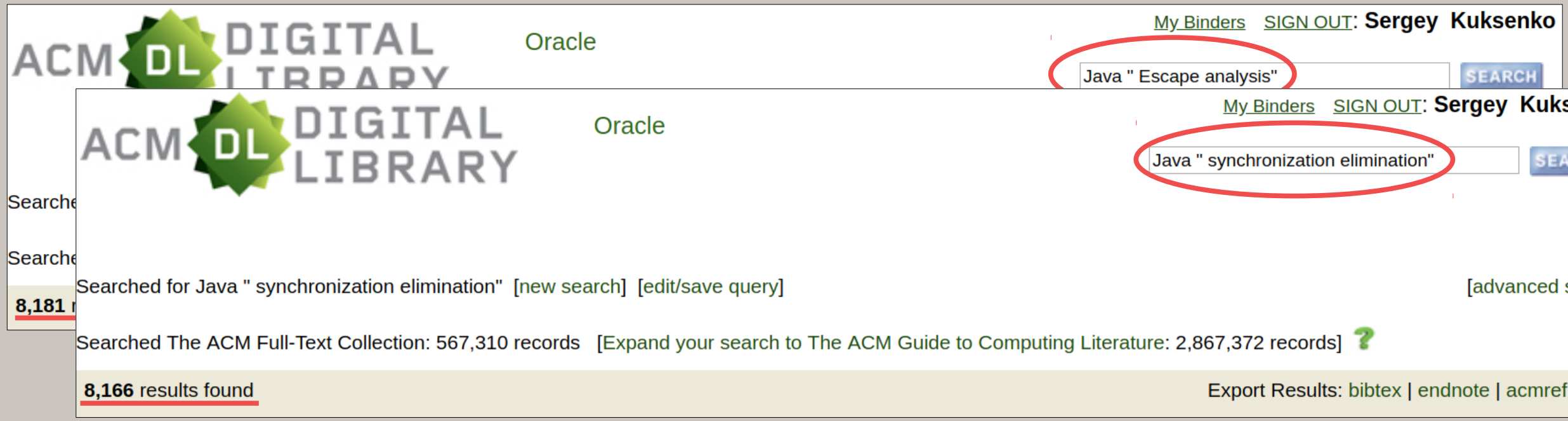
```
Integer.valueOf(420) != Integer.valueOf(420)
```


Why JVM can’t eliminate it?

JVM can!



JVM can!



JVM can, but …

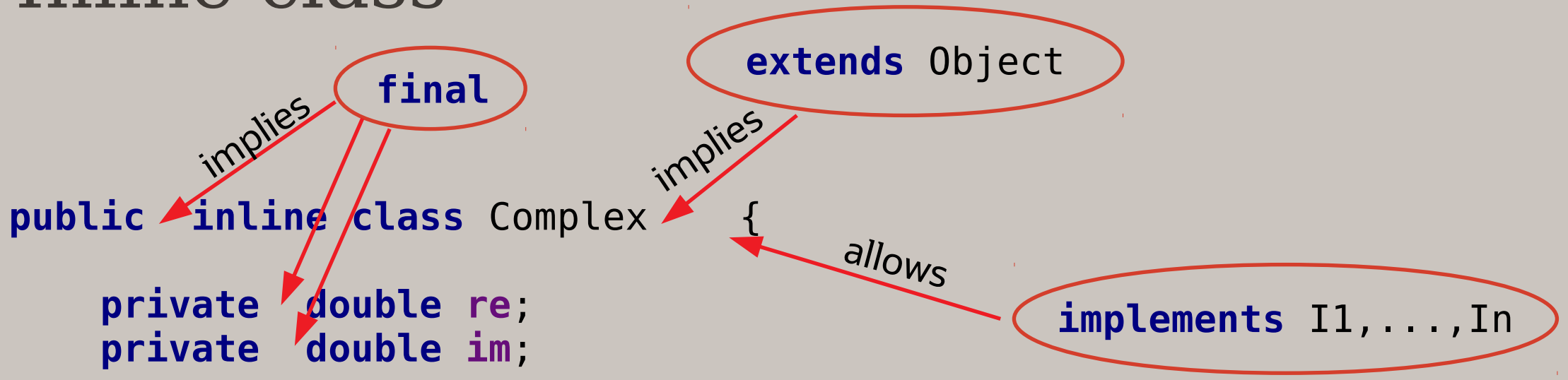
Inline class

- is a class
- no identity
- immutable
- not nullable
- no synchronization

Inline class

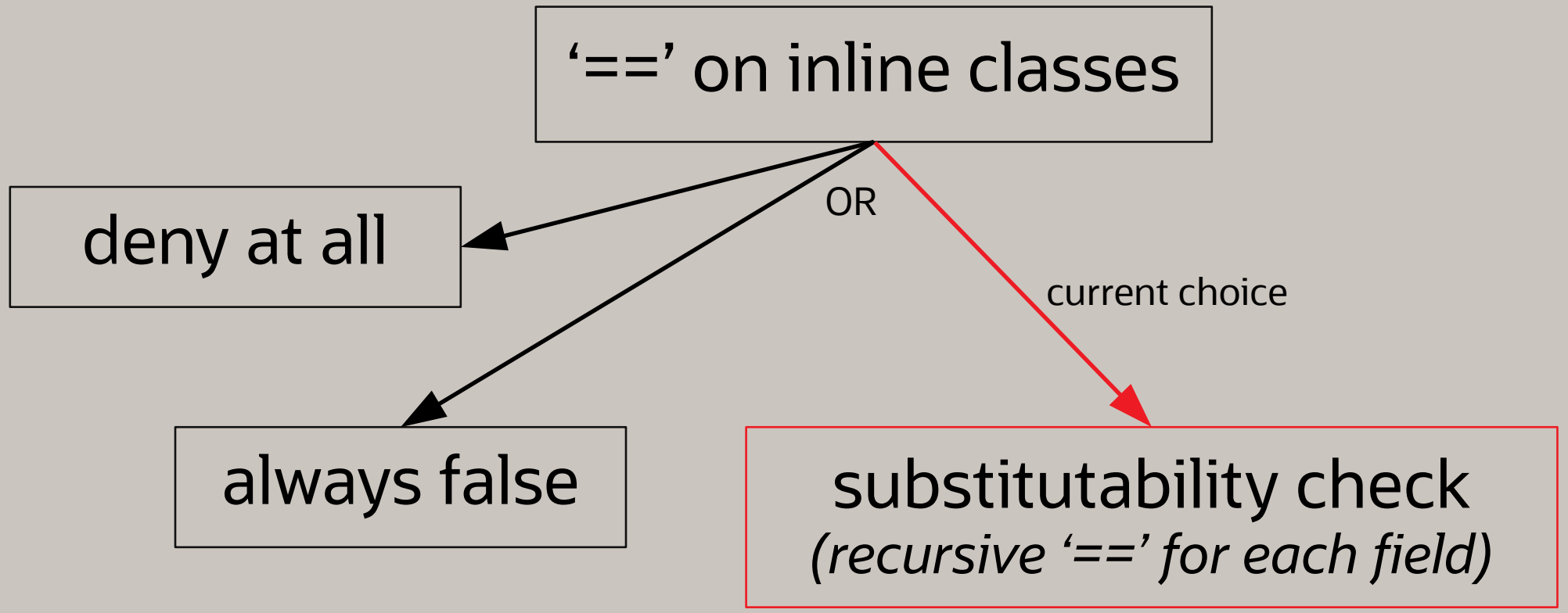
```
public  inline class Complex    {

    private  double re;
    private  double im;

    public Complex(double re, double im) {
        this.re =  re;
        this.im =  im;
    }
    public double re() { return re; }
    public double im() { return im; }
    ...
```


Inline class




```
public inline class Complex {

    private  double re;
    private  double im;

    public Complex(double re, double im) {
        this.re =  re;
        this.im =  im;
    }
    public double re() { return re; }
    public double im() { return im; }
    ...
```

Identity of indiscernibles



Inline class means inlineable

- JVM decides if:
  - allocate on heap

    OR
  - put on stack (locals, parameters, result)
  - inline into container class
  - inline into array (flattened array)

Inline class

- Inline types are subtypes of Object (interface)
- Inline arrays are covariant with Object[]

  (arrays of interface)

Boxing vs boxing

- V? - nullable twin of ‘V’
- means all values of V + ‘null’

compare to:

- Integer - nullable twin of ‘int’
- But Integer has full identity

~~Float like a butterfly, Sting like a bee~~

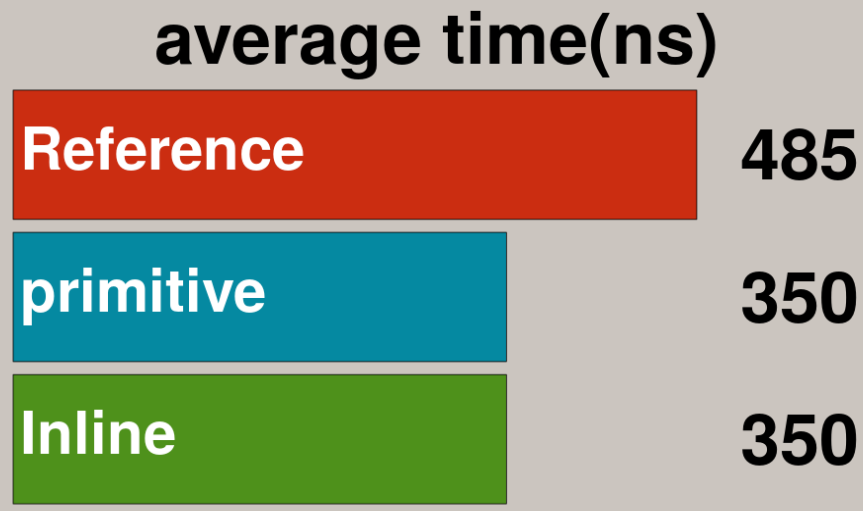
Code like a class, Work like an int

Local variable

```
int count(Complex c) {
    Complex z = c;
    for (int i = 1; i < MAX_ITERATION; i++) {
        if (z.modulus() >= 4.0) return i;
        z = z.square().add(c);
    }
    return MAX_ITERATION;
}
```

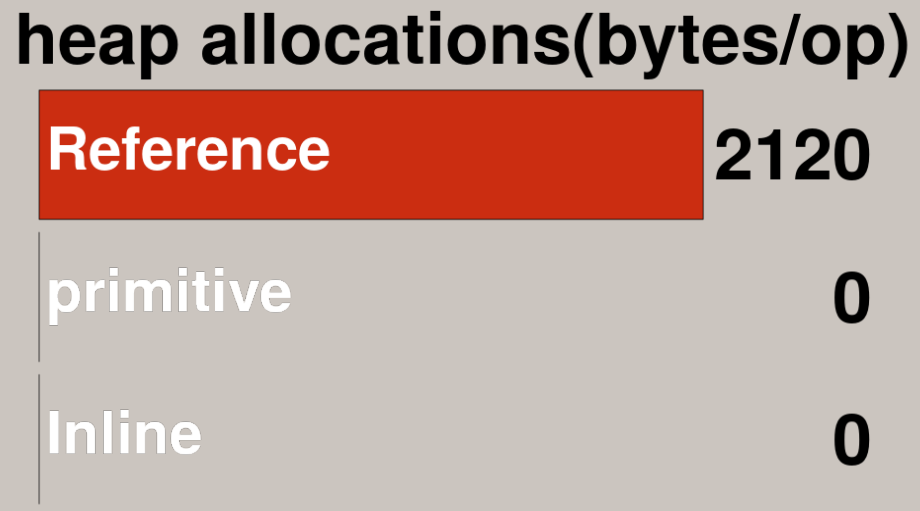

Local variable

```
int count(Complex c) {
    Complex z = c;
    for (int i = 1; i < MAX_ITERATION; i++) {
        if (z.modulus() >= 4.0) return i;
        z = z.square().add(c);
    }
    return MAX_ITERATION;
}
```



Local variable

```
int count(Complex c) {
    Complex z = c;
    for (int i = 1; i < MAX_ITERATION; i++) {
        if (z.modulus() >= 4.0) return i;
        z = z.square().add(c);
    }
    return MAX_ITERATION;
}
```

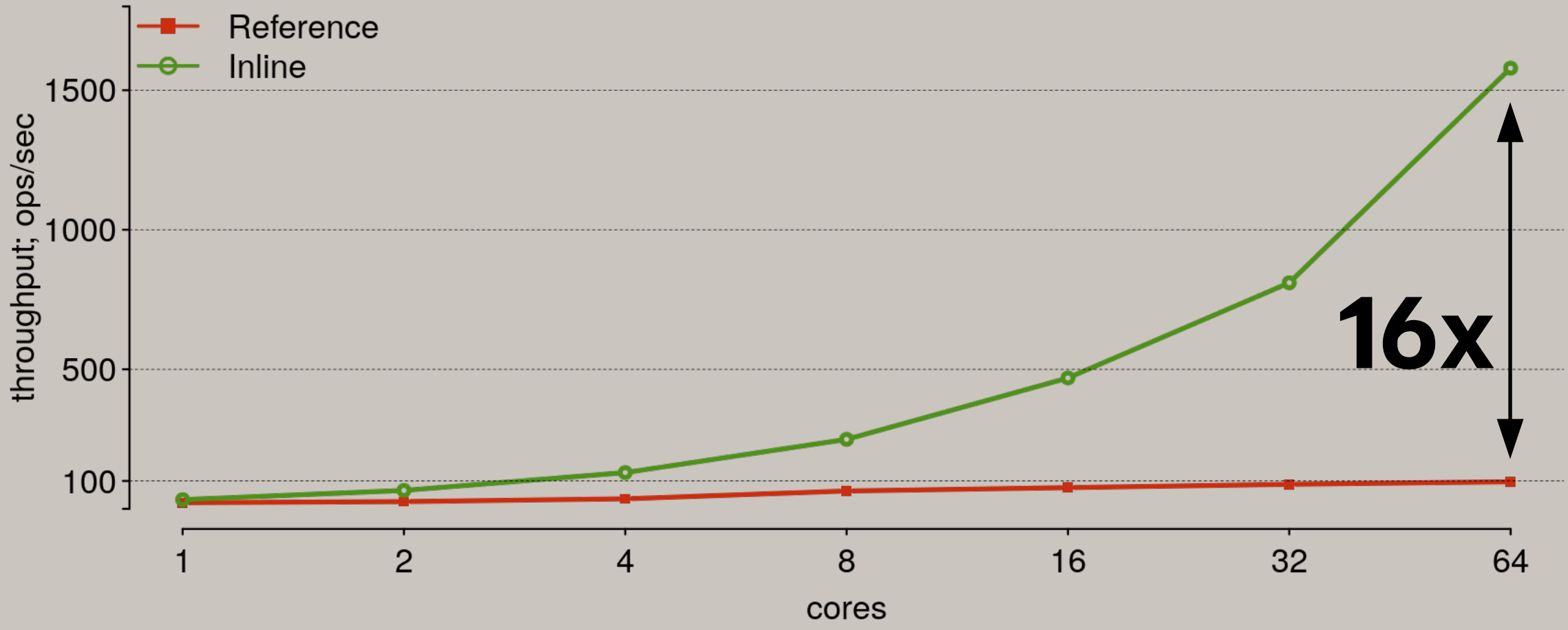


Reference vs Inline (Mandelbrot, 500x500)

Inline class:

- 4x less data loads
- 42x less L1 cache misses
- 5x less L3 cache misses
- 5x less dTLB misses

Scalability (Mandelbrot, 500x500)

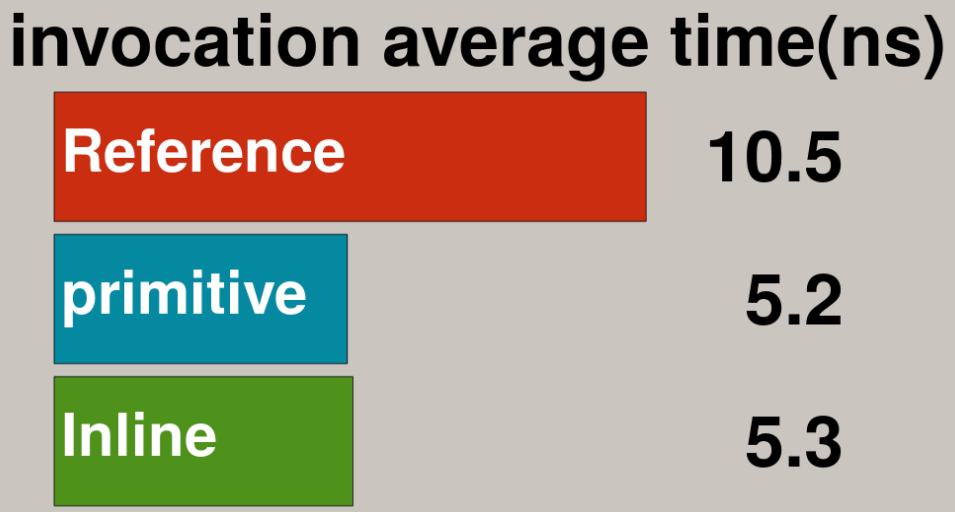


Method parameters/result

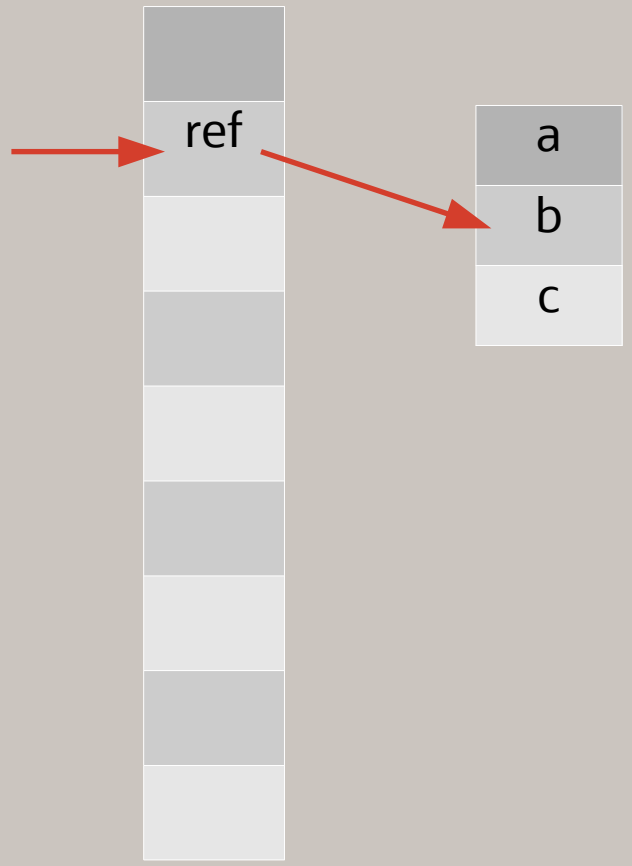
```
static Value ackermann(Value x, Value y) {
  return x.isZero() ? y.inc() :
            (y.isZero() ? ackermann(x.dec(), new Value(1)) :
                          ackermann(x.dec(), ackermann(x, y.dec())));
}
```


Method parameters/result

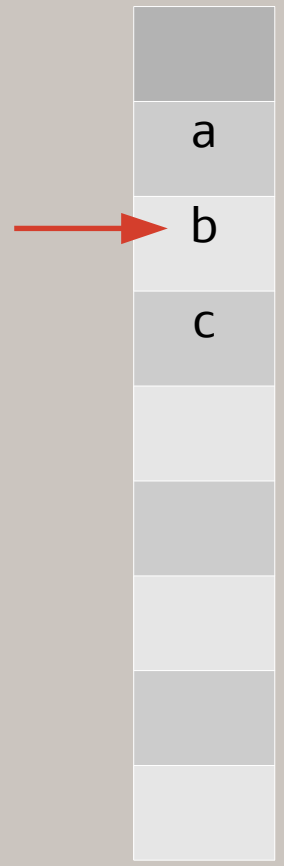
```
static Value ackermann(Value x, Value y) {
  return x.isZero() ? y.inc() :
           (y.isZero() ? ackermann(x.dec(), new Value(1)) :
                         ackermann(x.dec(), ackermann(x, y.dec())));
}
```



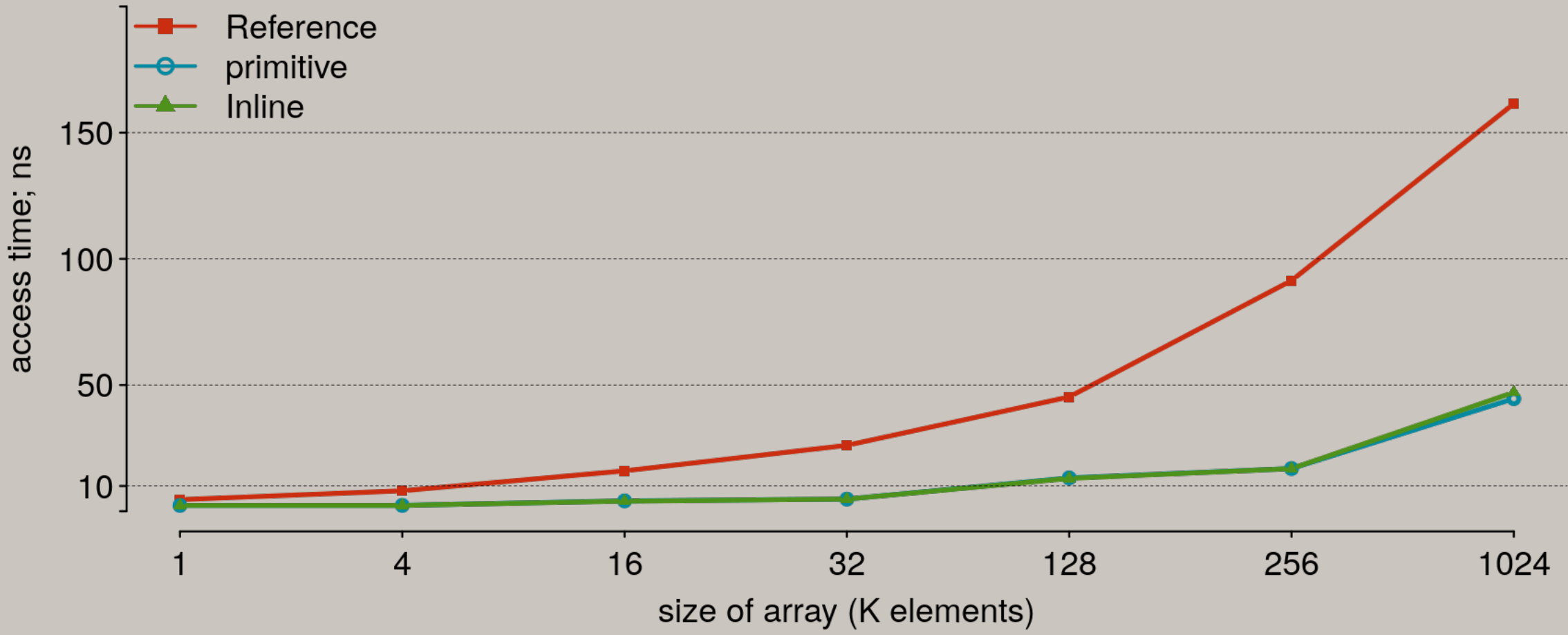
Array access



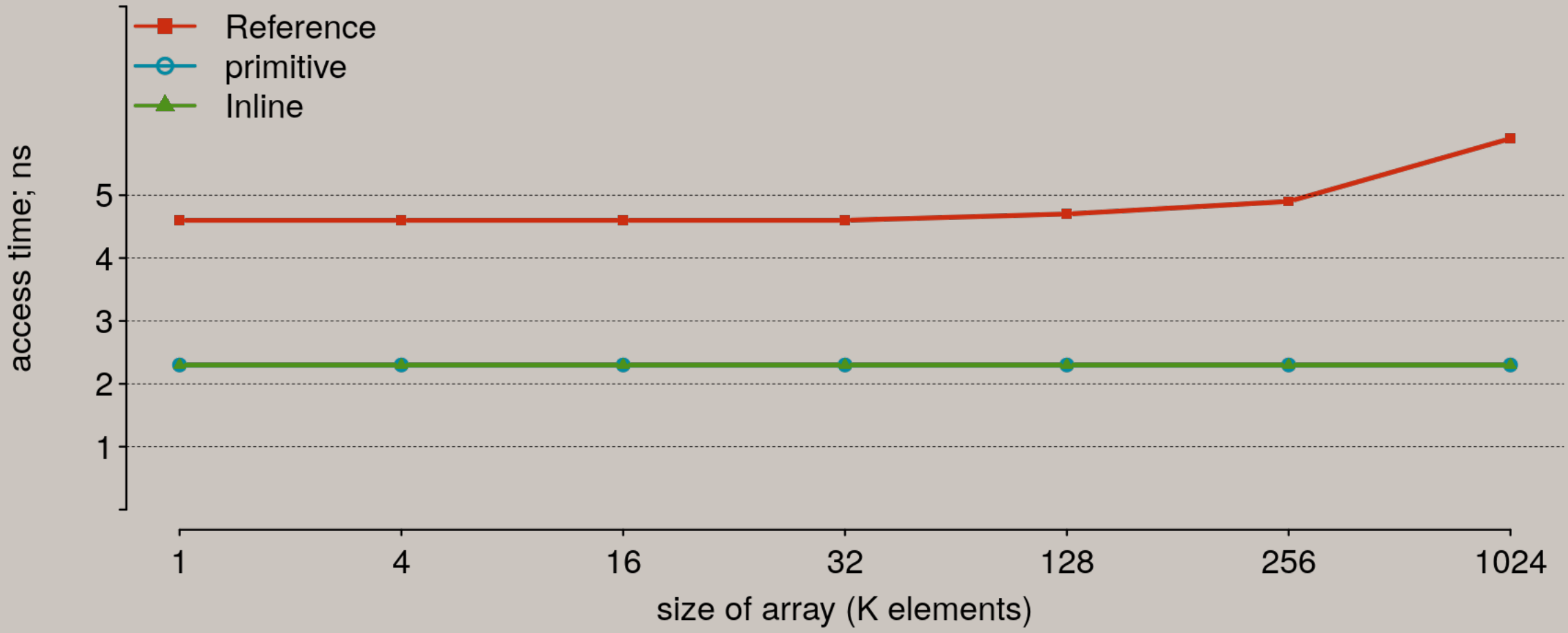
VS



Array Random Access



Array Sequential Access



Collateral Damage
in legacy world

The following section is intended to outline Valhalla current status and development. It is intended for information purposes only, and may not be incorporated into any contract (or slowdown blaming). Any adverted performance regression maybe a subject to removal.

Inline in heap (“boxing”)

- Object o = my_inline_value;
- Interface i = my_inline_value;
- Value? nullable_value = my_inline_value;
- JVM decided

Reference comparison

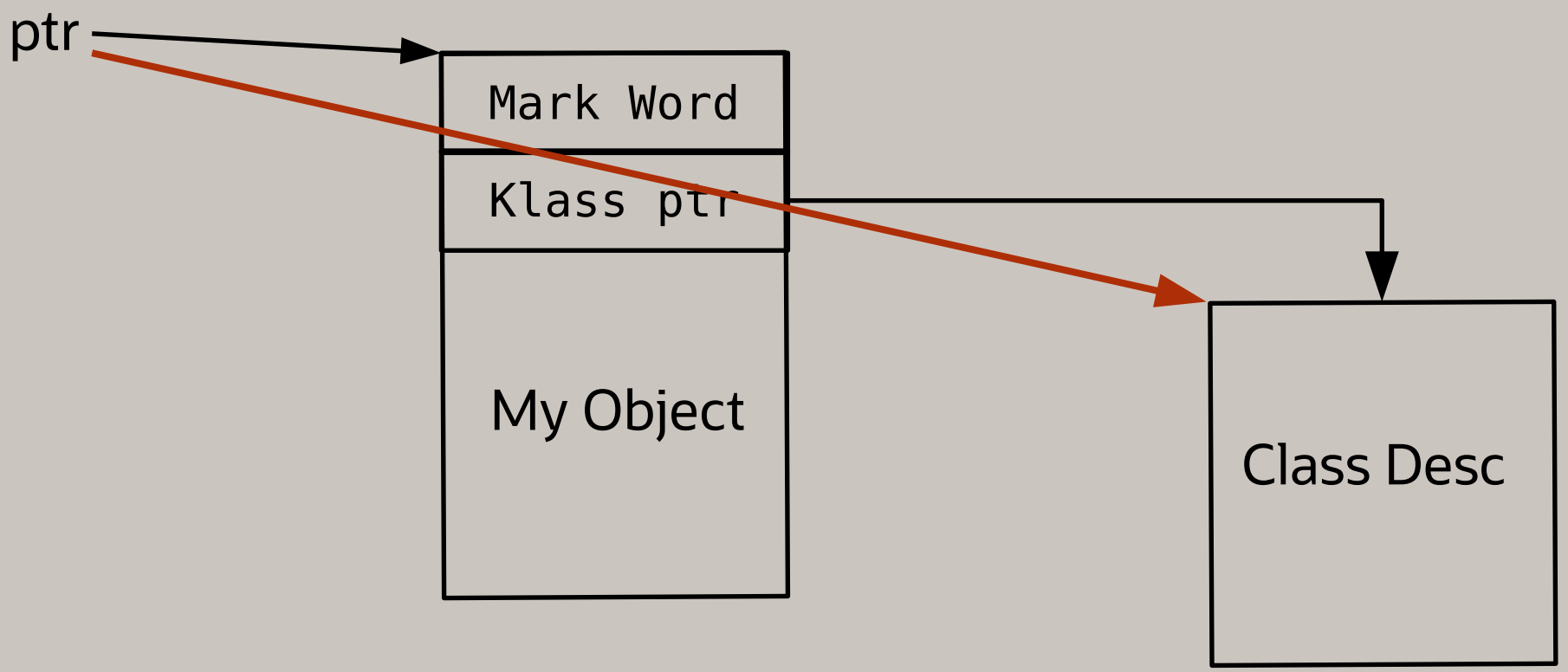
Glorious pre Valhalla past

```
just compare it
```

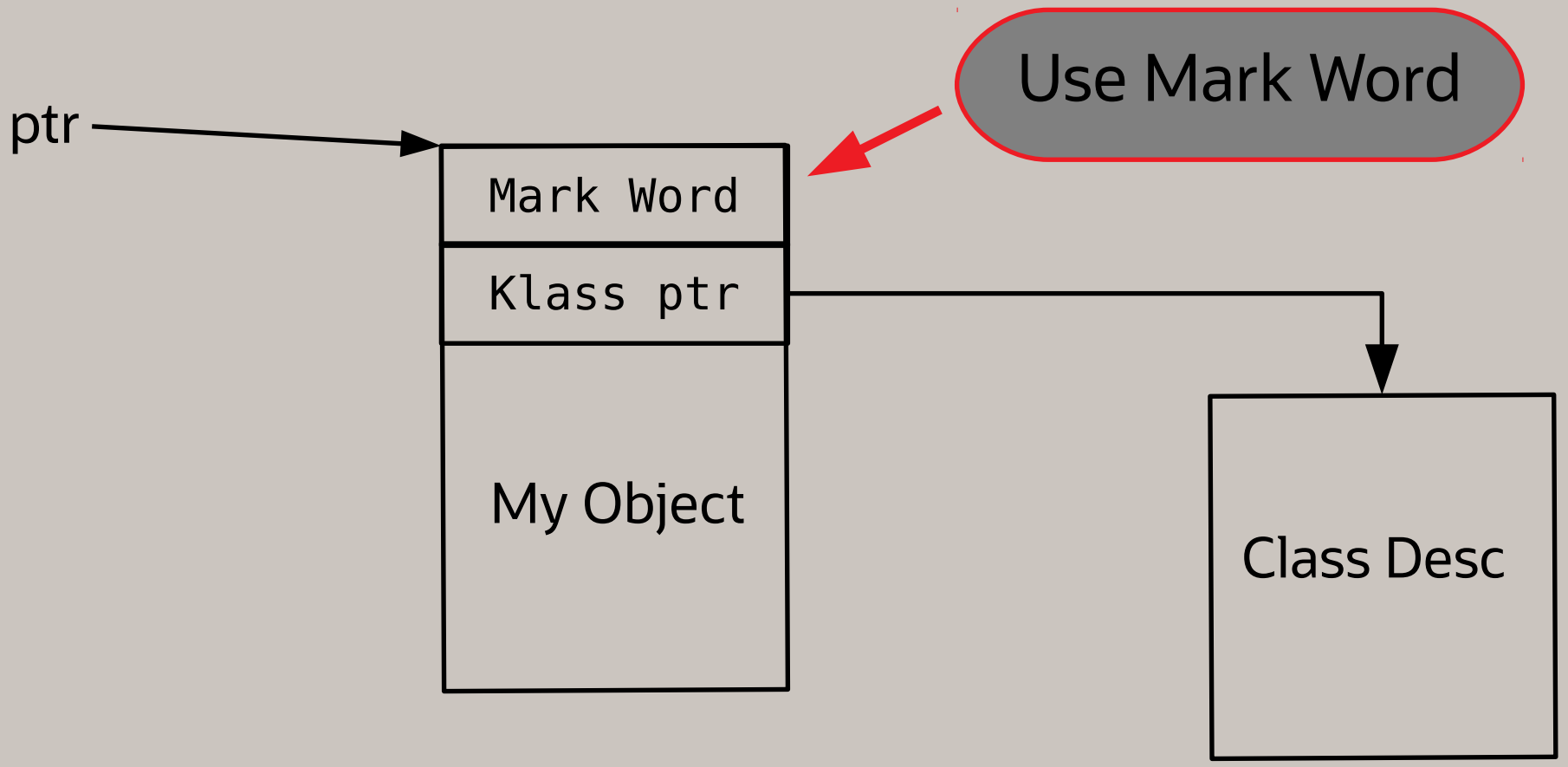
Brighter post Valhalla future

```
if <both refs are inline>
  if <classes are same>
  then
    check substitutability
  else
    false
else
  just compare it
```


If object is inline class



If object is inline class



Mark Word

```
//  64 bits:
//  --------
//  unused:25 hash:31 -->| unused:1   age:4    biased_lock:1 lock:2 (normal object)
//  JavaThread*:54 epoch:2 unused:1   age:4    biased_lock:1 lock:2 (biased object)
//  "1"        :54 epoch:2 unused:1   age:4    biased_lock:1 lock:2 (biased always locked object)
//  PromotedObject*:61 --------------------->| promo_bits:3 ----->| (CMS promoted object)
//  size:64 ----------------------------------------------------->| (CMS free block)
//
//  unused:25 hash:31 -->| cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && normal object)
//  JavaThread*:54 epoch:2 cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && biased object)
//  narrowOop:32 unused:24 cms_free:1 unused:4 promo_bits:3 ----->| (COOPs && CMS promoted object)
//  unused:21 size:35 -->| cms_free:1 unused:7 ------------------->| (COOPs && CMS free block)
```

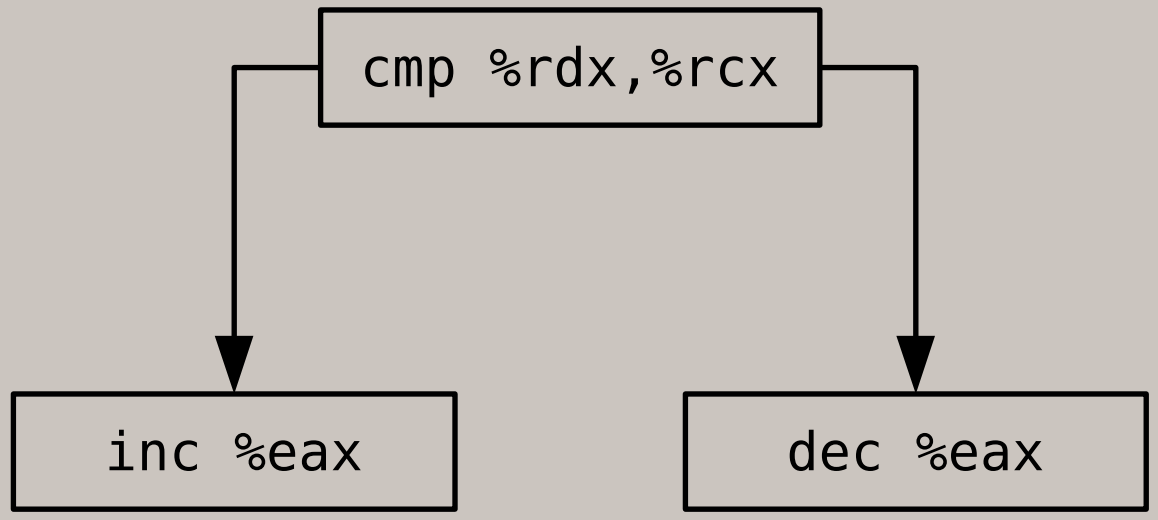

Mark Word

```
//  64 bits:
//  --------
//  unused:25 hash:31 -->| unused:1   age:4    biased_lock:1 lock:2 (normal object)
//  JavaThread*:54 epoch:2 unused:1   age:4    biased_lock:1 lock:2 (biased object)
//  "1"        :54 epoch:2 unused:1   age:4    biased_lock:1 lock:2 (biased always locked object)
//  PromotedObject*:61 --------------------->| promo_bits:3 ----->| (CMS promoted object)
//  size:64 ----------------------------------------------------->| (CMS free block)
//
//  unused:25 hash:31 -->| cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && normal object)
//  JavaThread*:54 epoch:2 cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && biased object)
//  narrowOop:32 unused:24 cms_free:1 unused:4 promo_bits:3 ----->| (COOPs && CMS promoted object)
//  unused:21 size:35 -->| cms_free:1 unused:7 ------------------->| (COOPs && CMS free block)


//    [ <unused> | larval |1| epoch | age | 1 | 01]       permanently locked
```

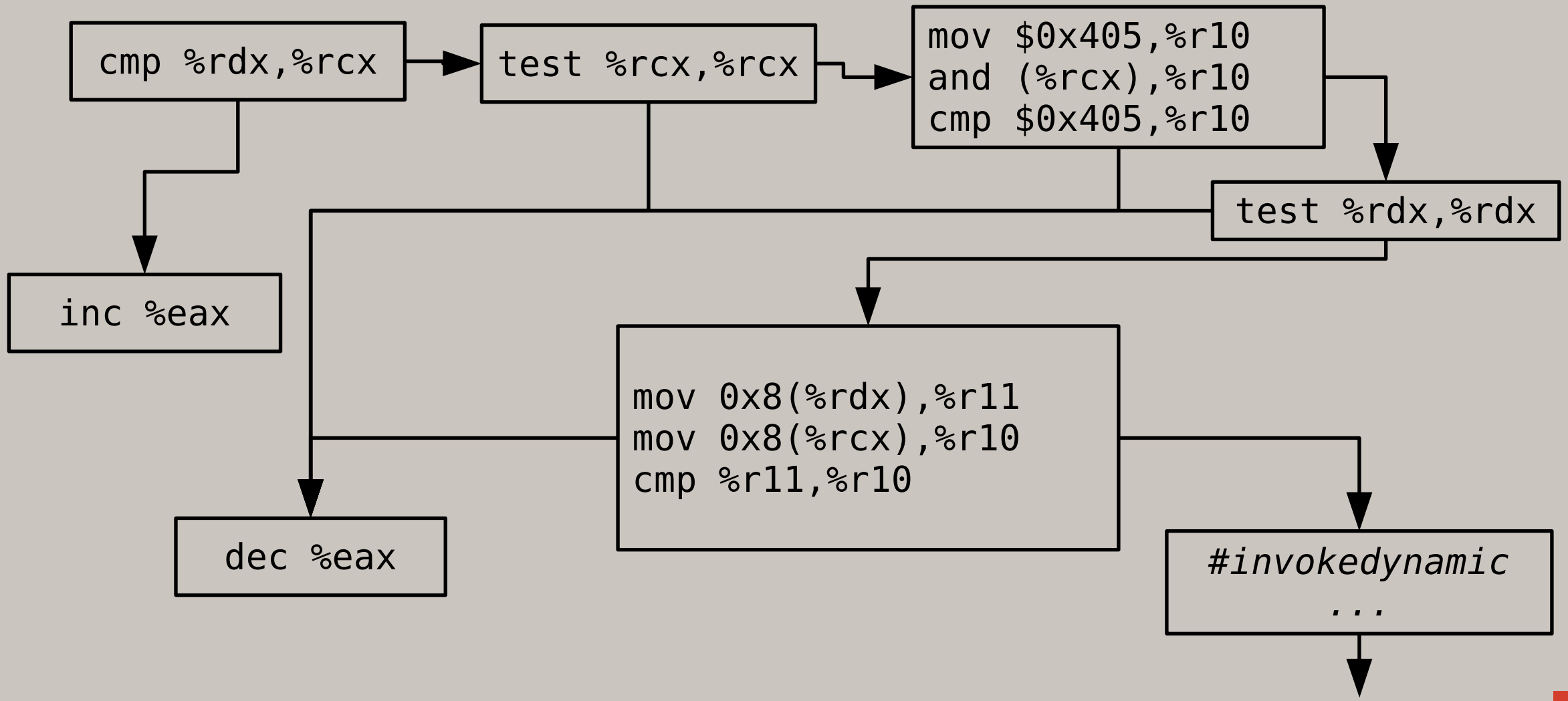

Reference comparison a.k.a. ‘acmp’

```
...
if (o1 == o2) {
    ... = x + 1;
} else {
    ... = x - 1;
}
...
```

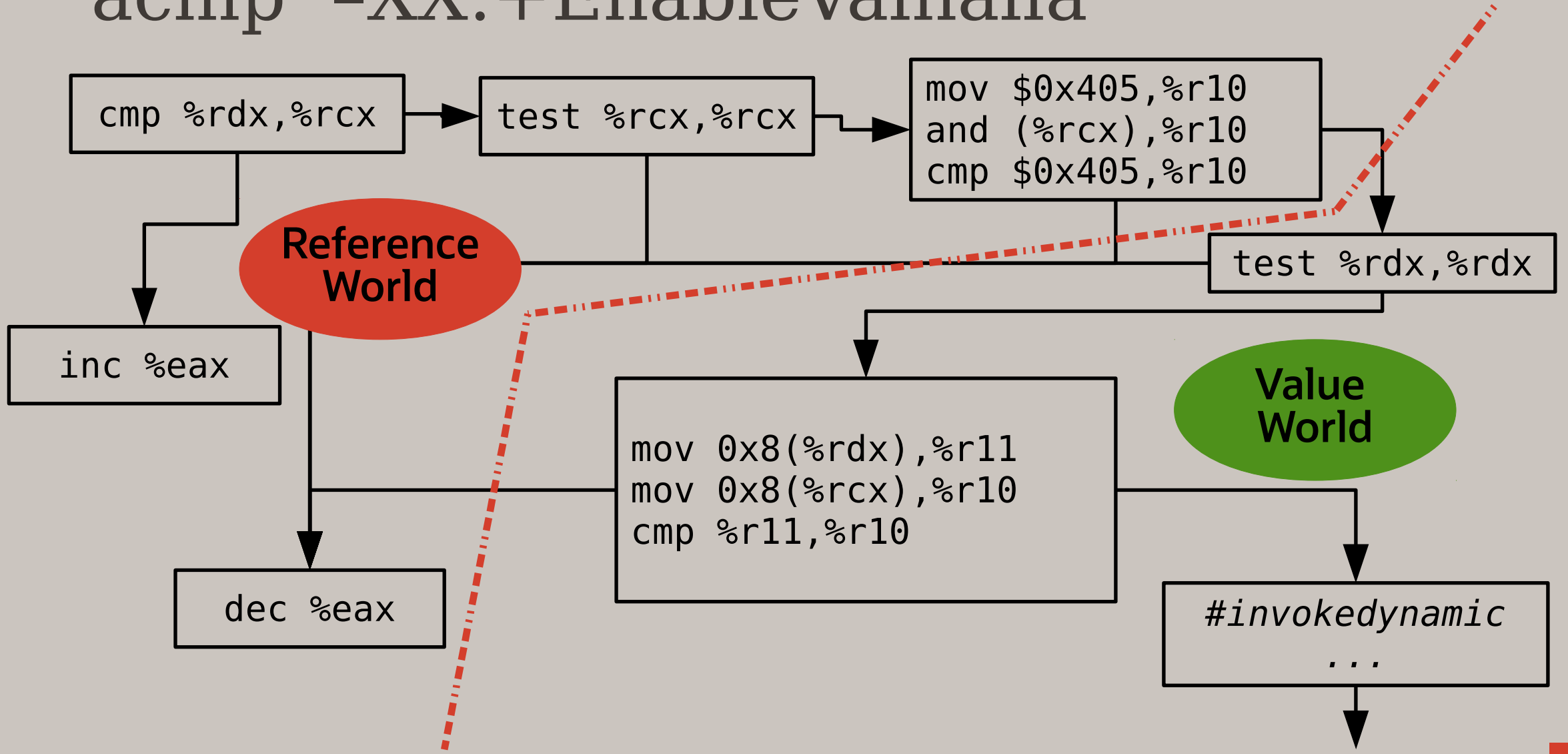

acmp -XX:-EnableValhalla



acmp –XX:+EnableValhalla



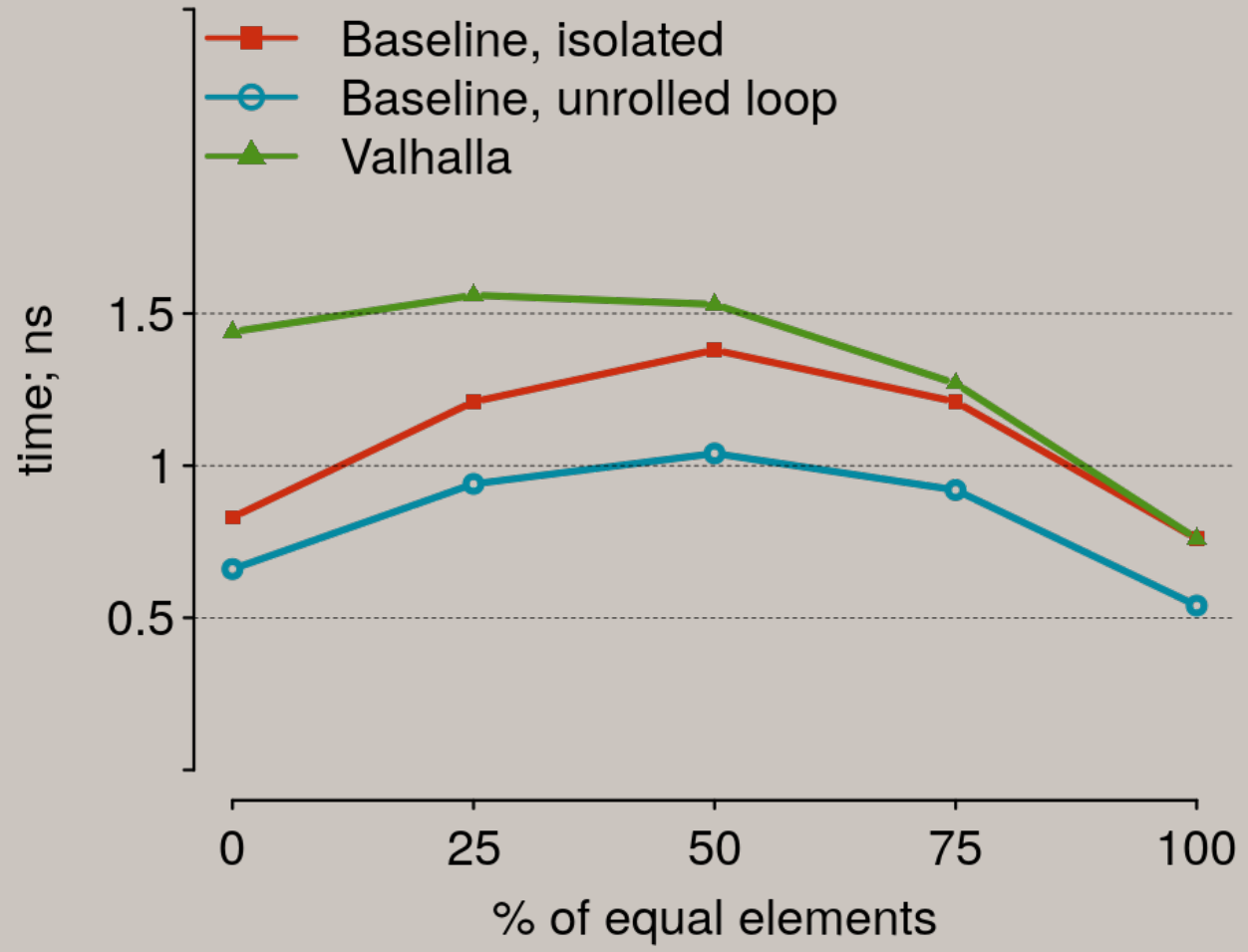
acmp -XX:+EnableValhalla



acmp

- Complex code
- Additional loads
- invokedynamic prevents loop unrolling

acmp performance



synchronized(obj)

Glorious pre Valhalla past

```
do all synch stuff
```

Brighter post Valhalla future

```
if <ref is inline class>
then
  throw exception
else
  do all synch stuff
```


synchronized(obj)

Glorious pre Valhalla past

```
do all synch stuff
```

Brighter post Valhalla future

```
if <ref is inline class>
then
  throw exception
else
  do all synch stuff
```

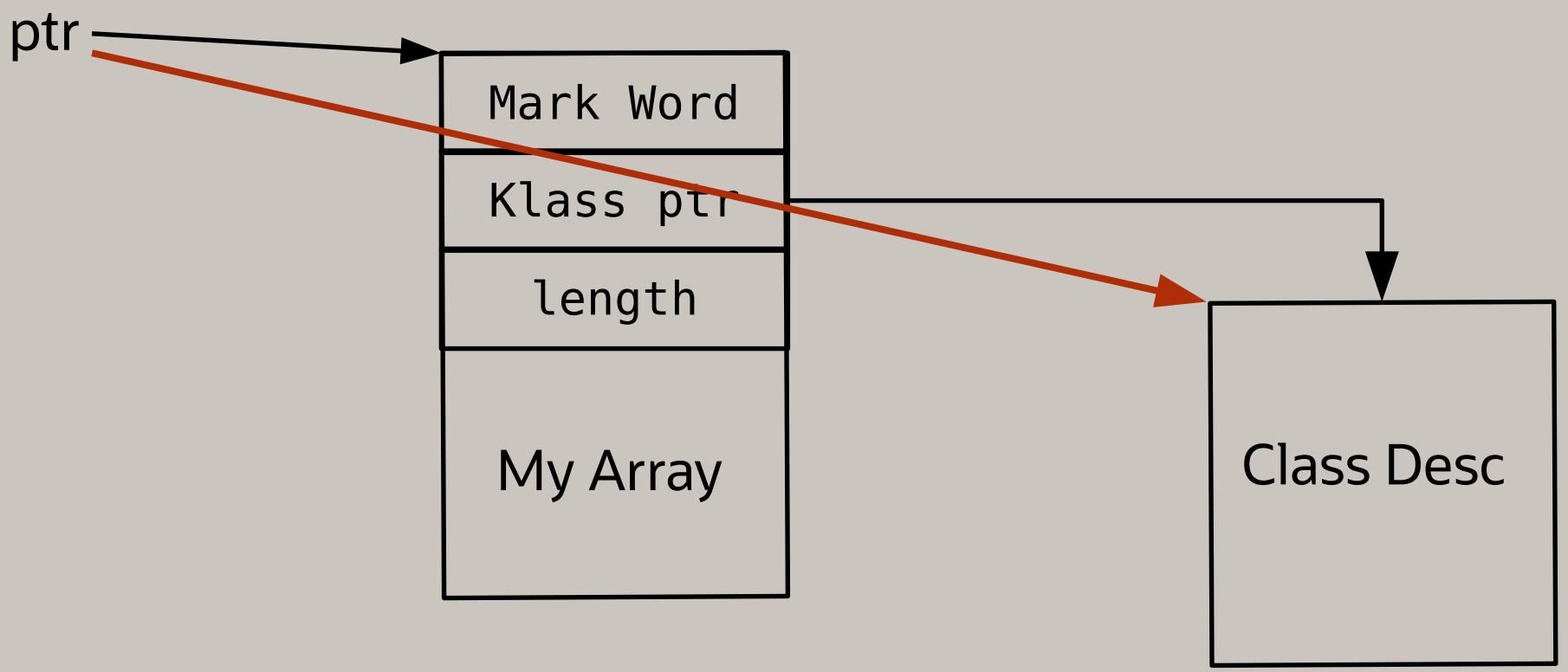
< 1% difference

Arrays (Object[])

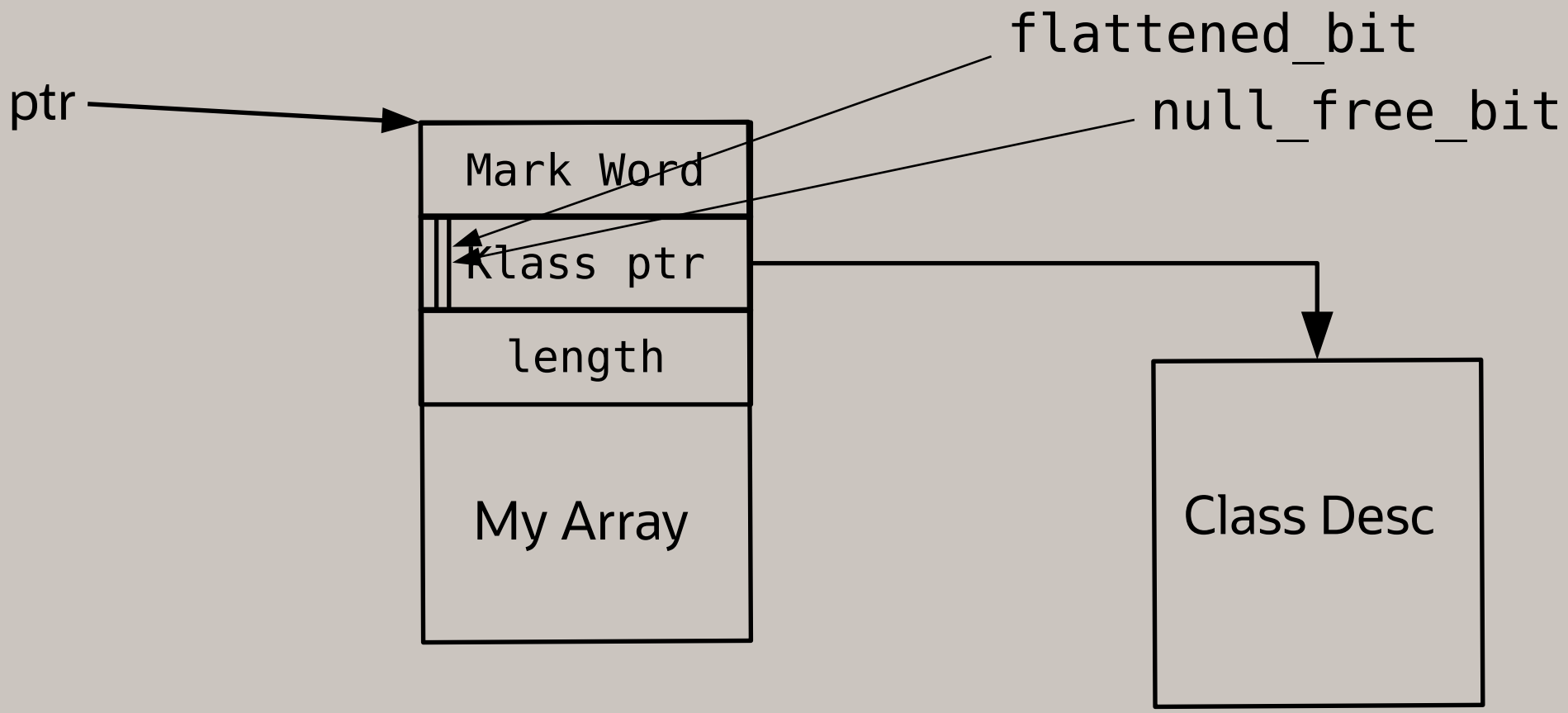
Object[] may be:

- Good old reference array
- Array of inline classes in heap
    - references, but not nullable
- Flattened array of inline classes

Arrays (Object[])



Arrays (Object[])



Arrays (Object[])

- Any access to Klass ptr required clearing:
  - `and $0x1fffffff,%reg`
- HotSpot is good enough at eliminating it
  - knowing that it isn't an array

Load from Object[]

Glorious pre Valhalla past

```
element size is the same:

just load it
```

Brighter post Valhalla future

```
if <array is flattened>
  find element size
  load it
  do boxing if needed
else
  just load it
```


Store to Object[]

Glorious pre Valhalla past

```
do ArrayStoreCheck

store if ok
```

Brighter post Valhalla future

```
do ArrayStoreCheck
if <array is flattened>
  find element size
  do unboxing if needed
  store
else
  store
```


Object[] access

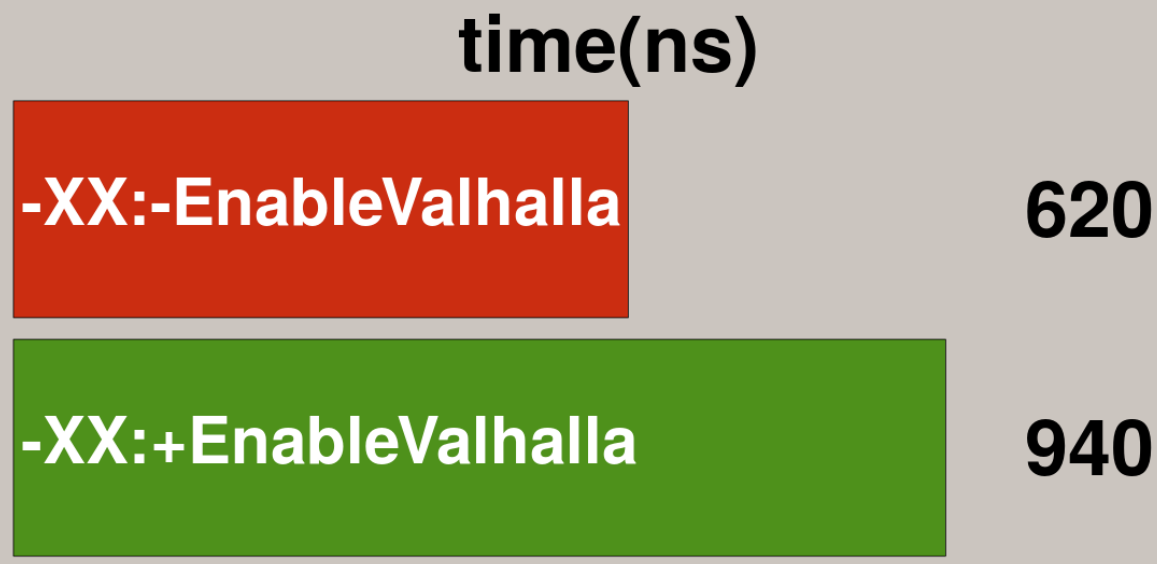
- Targeting benchmarks: –2% . . – 10%
- Solution: aggressive loop hoisting and loop duplication (in progress)

Inline vs inline

```
Integer[] i1 = new Integer[1000];
Integer[] i2 = new Integer[1000];

@Setup
public void setup() {
    for (int i = 0; i < 1000; i++)
        i1[i] = i2[i] = i;
    i2[999] = 394857623;
}

@Benchmark
public boolean arrayEquals() {
    return Arrays.equals(i1, i2);
}
```


Inline vs inline

```
Integer[] i1 = new Integer[1000];
Integer[] i2 = new Integer[1000];

@Setup
public void setup() {
    for (int i = 0; i < 1000; i++)
        i1[i] = i2[i] = i;
    i2[999] = 394857623;
}

@Benchmark
public boolean arrayEquals() {
    return Arrays.equals(i1, i2);
}
```



Methods inline tree

-XX:-EnableValhalla

```
   ...
- @ java.util.Arrays::equals [..., bytes=57, insts=352]
                (inlined: inline (hot))
  - @ java.util.Objects::equals [..., bytes=23, insts=128]
                (inlined: inline (hot))
```

-XX:+EnableValhalla

```
   ...
- @ java.util.Arrays::equals [..., bytes=57, insts=1760]
                (inline failed: already compiled into a big method)
```


Methods inline tree

-XX:-EnableValhalla

```
   ...
- @ java.util.Arrays::equals [..., bytes=57, insts=352]
                (inlined: inline (hot))
  - @ java.util.Objects::equals [..., bytes=23, insts=128]
                (in                          ot))
```

Size of generated code

-XX:+EnableValhall

```
   ...
- @ java.util.Arrays::equals [..., bytes=57, insts=1760]
                (inline failed: already compiled into a big method)
```

Current status

- Checked ~30 big benchmarks:
  - No regressions more than 2%
- Checked ~1600 microbenchmarks:
  - 1200 – ±0% at the first run
  - 300 – fixed
  - 100 – less 5% (in progress)
  - 1 – 14% regression (in progress)

Brighter post Valhalla future

Arithmetic types

- Complex matrix multiplication (100x100)

ref Complex	12.6 ms
inline Complex	2.7 ms
inline Complex + cache friendly algorithm	2.1 ms

java.util.Optional

```
public class HashMap<K,V> ... {
    ...
    /**
     * Returns the value to which the specified key is mapped,
     * or null if this map contains no mapping for the key.
     ...
     */
    public V get(Object key)
```


java.util.Optional

```
public class HashMap<K,V> ... {
    ...
    /**
     * Returns the value to which the specified key is mapped,
     * or null if this map contains no mapping for the key.
     ...
     */
    public V get(Object key)
```

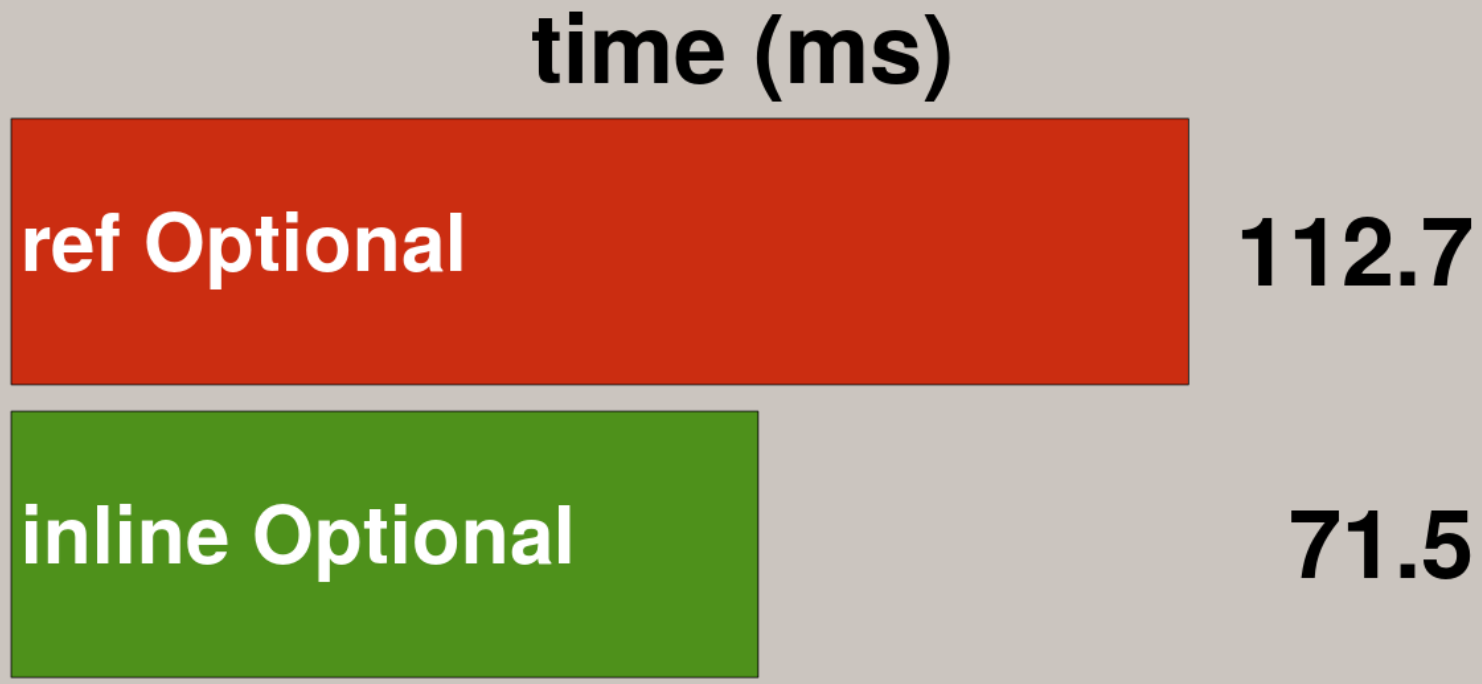
java.util.Optional

```
public class HashMap<K,V> ... {
    ...
    /**
     * Returns an Optional describing the value to which the specified
     * key is mapped, or an empty Optional if this map contains no
     * mapping for the key.
     ...
     */
    public Optional<V> get(Object key)
```

What if?

java.util.Optional

- 1000000 gets from 1000000 map



Example: map with complex key

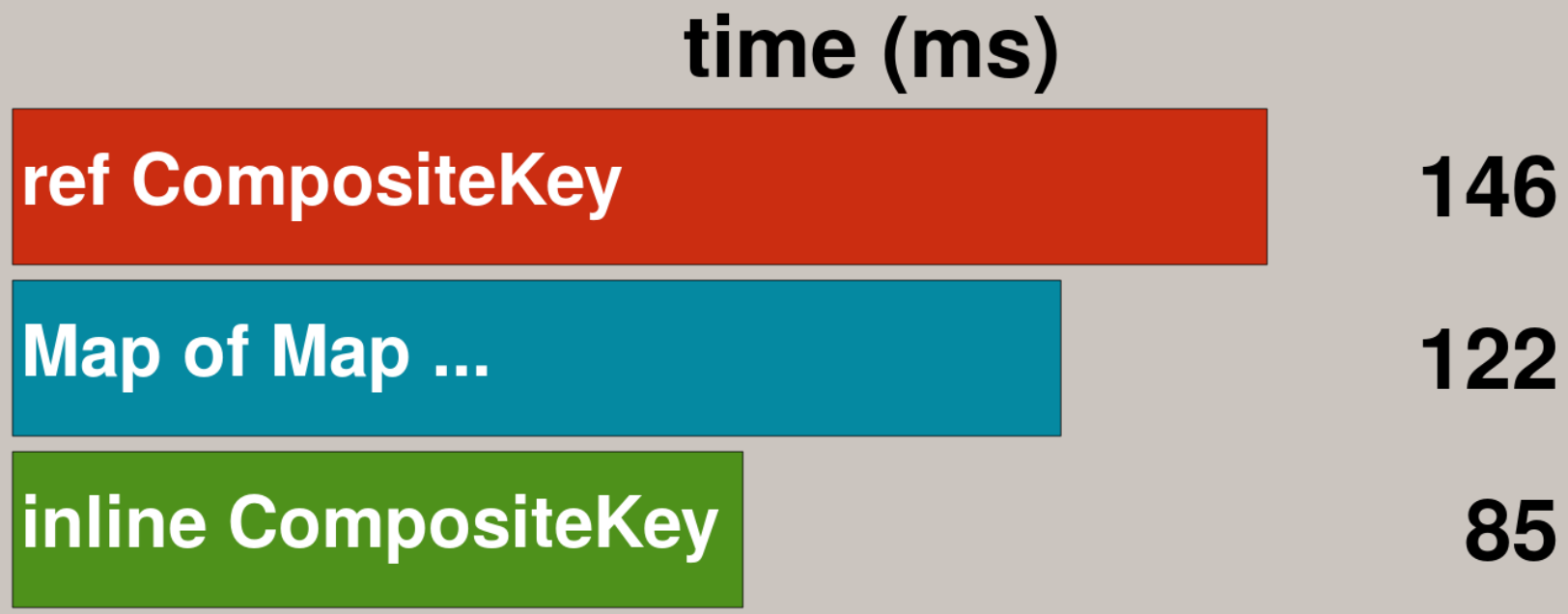
- Map from <K_1, …, K_N> → <V>

  Two ways to implement:

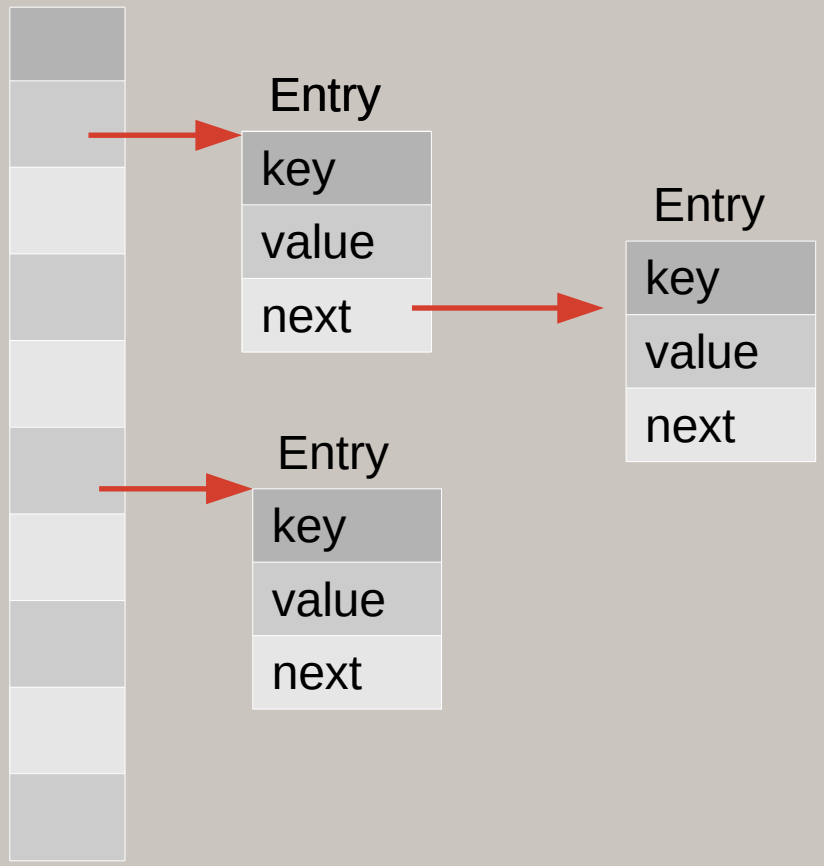
1. Map<CompositeKey<K_1, …, K_N>, V>
2. Map<K_1, Map …, Map<K_N, V>…>

Example: Map from <Integer, Integer> → <Integer>

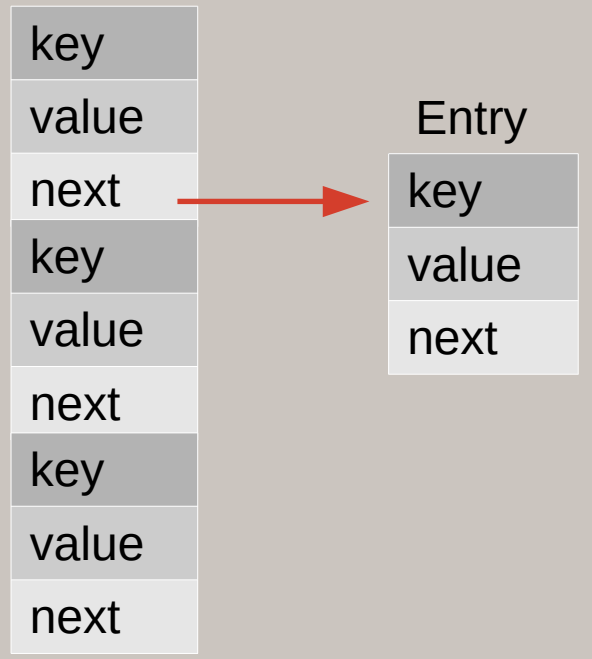
- 1000000 gets from 1000000 map



HashMap inside



VS



HashMap experiments

- Classic HashMap.get(): ~75 ns
- Experimental HashMap.get(): ~60 ns

? Faster only for large maps

? ‘put’ is slower

… to be continued

Iteration

```
HashMap<Integer, Integer> map; // map.size() == 1000000

@Benchmark
public int sumValuesInMap() {
    int s = 0;
    for (Integer i : map.values()) {

        s += i;
    }
    return s;
}
```


Iteration

```
HashMap<Integer, Integer> map; // map.size() == 1000000

@Benchmark
public int sumValuesInMap() {
    int s = 0;
    for (Integer i : map.values()) {

        s += i;
    }
    return s;
}
```

Lucky case	33 ms
Unlucky case	55 ms

Iteration

```
HashMap<Integer, Integer> map; // map.size() == 1000000

@Benchmark
public int sumValuesInMap() {
    int s = 0;
    for (Iterator<Integer> iterator = map.values().iterator();
                                      iterator.hasNext(); ) {
        s += iterator.next();
    }
    return s;
}
```

scalarized	Lucky case	33 ms
on heap	Unlucky case	55 ms

Inside HashMap

```
abstract class HashIterator {
    ...
Node<K,V> next;          // next entry to return
Node<K,V> current;       // current entry

final Node<K,V> nextNode() {
    ...
    if ((next = (current = e).next) == null && (t = table) != null) {
        do {}
        while (index < t.length && (next = t[index++]) == null);
    }
    return e;
}
...
```


Inside HashMap

```
abstract class HashIterator {
    ...
Node<K,V> next;          // next e
Node<K,V> current;       // curren

final Node<K,V> nextNode() {
    ...
    if ((next = (current = e).next) == null && (t = table) != null) {
        do {}
        while (index < t.length && (next = t[index++]) == null);
    }
    return e;
}
...
```

Write to reference field
GC write barriers

inline Cursor

```
public interface Cursor<V> {
    boolean hasElement();
    V get();
    Cursor<V> next();
}
@Benchmark
public int sumValuesInMap() {
    int s = 0;
    for (Cursor<Integer> cursor = map.values().cursor();
                                  cursor.hasElement();
                                  cursor = cursor.next()) {
        s += cursor.get();
    }
    return s;
}
```

Control on heap allocation

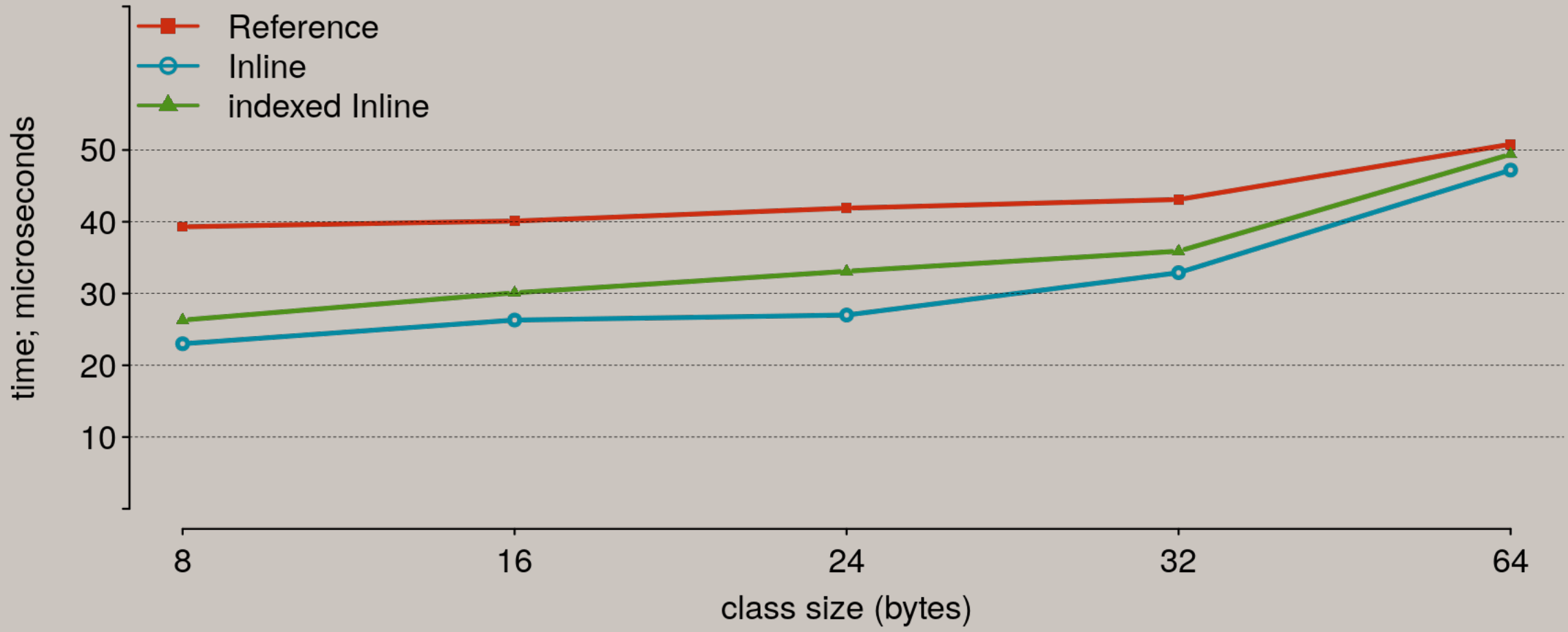
```
public interface Cursor<V
    boolean hasElement()
    V get();
    Cursor<V> next();
}
@Benchmark
public int sumValuesInMa
    int s = 0;
    for (Cursor<Integer> cursor = map.values().cursor();
                                  cursor.hasElement();
                                  cursor = cursor.next()) {

        s += cursor.get();
    }
    return s;
}
```

Scalarized Iterator	33 ms
On heap Iterator	55 ms
inline Cursor	32 ms

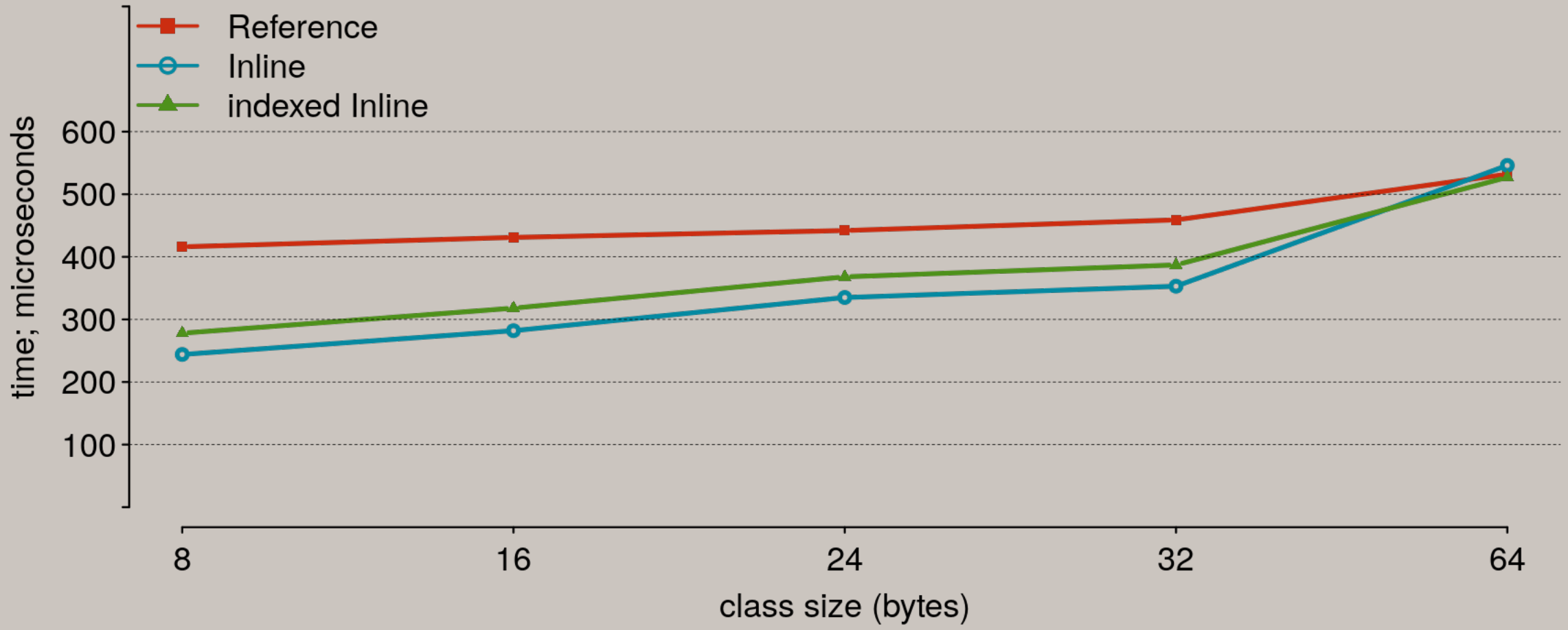
Move/Copy data

- Reference – easy, only reference is moved
- Inline – all data should be moved
  - Sort:
    - Reference – default TimSort from JDK
    - Inline – reimplented TimSort
    - indexed Inline – sort ‘int’ indices first, then copy

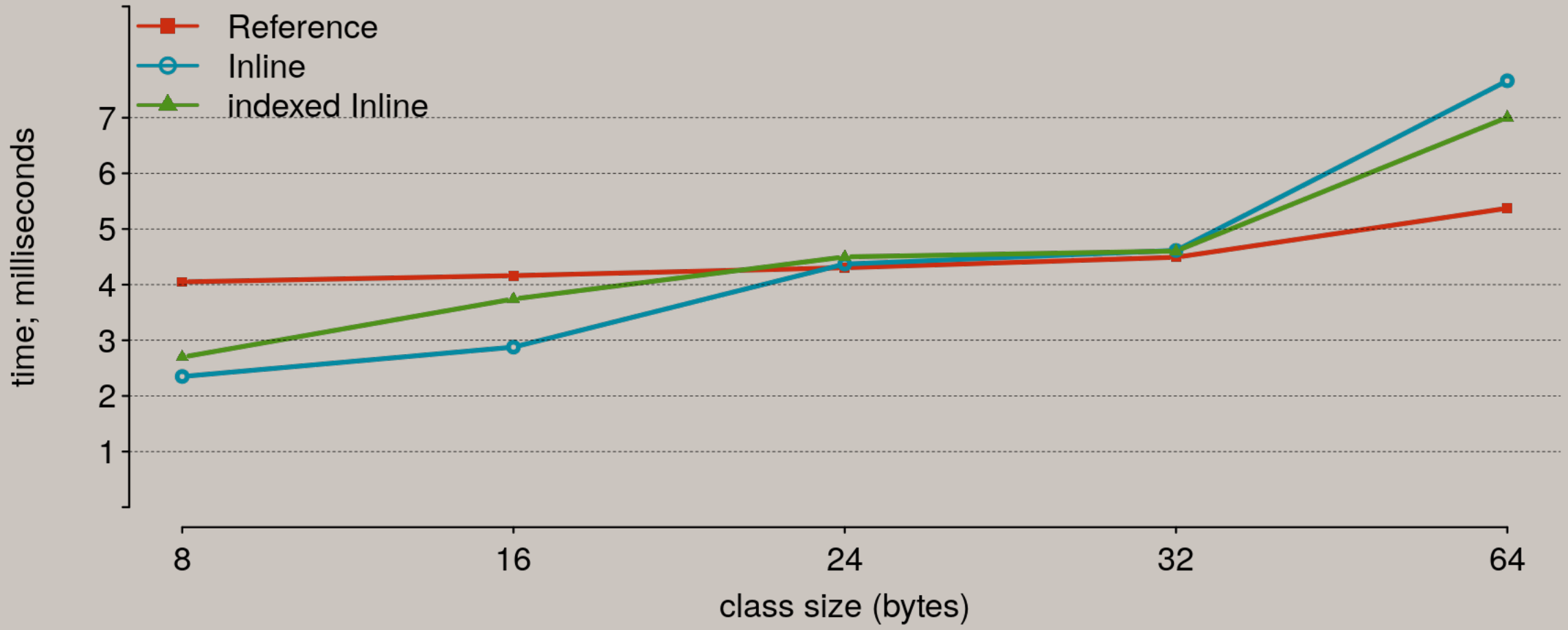
size==400, fit into L1 cache



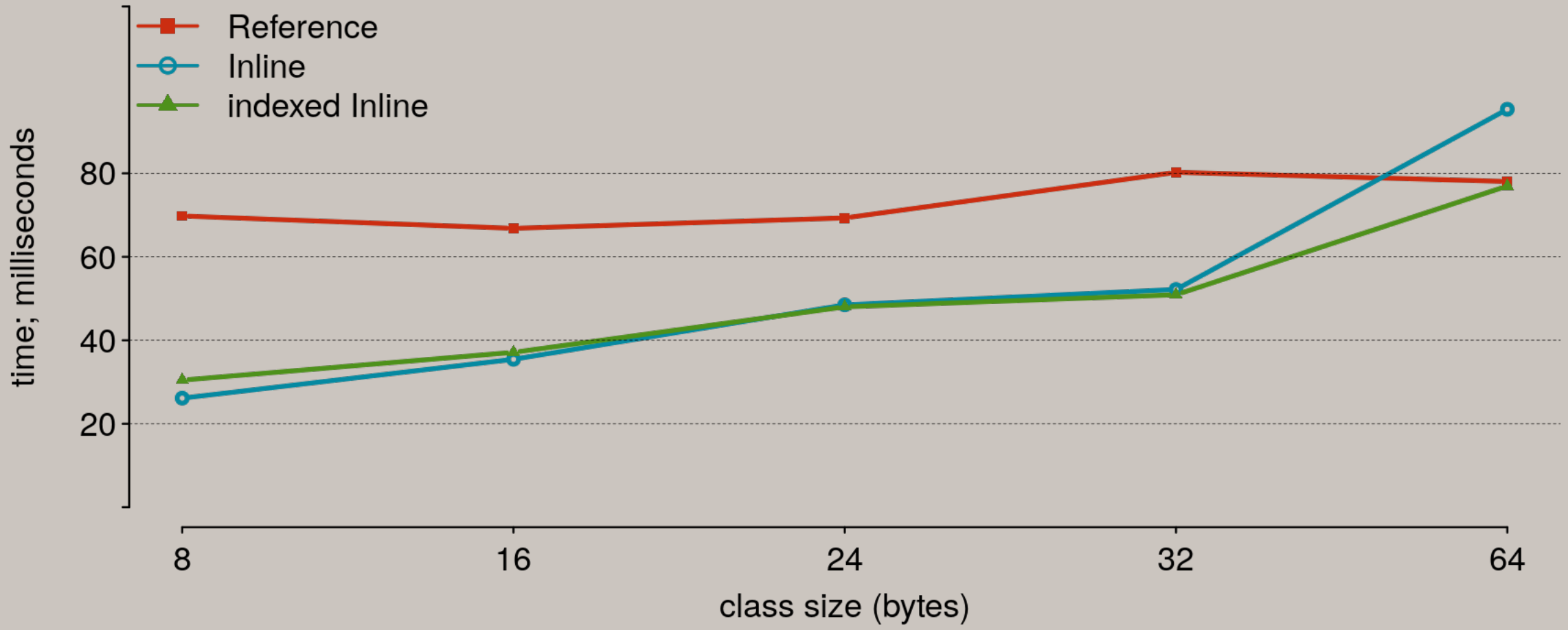
size==4000, fit into L2 cache



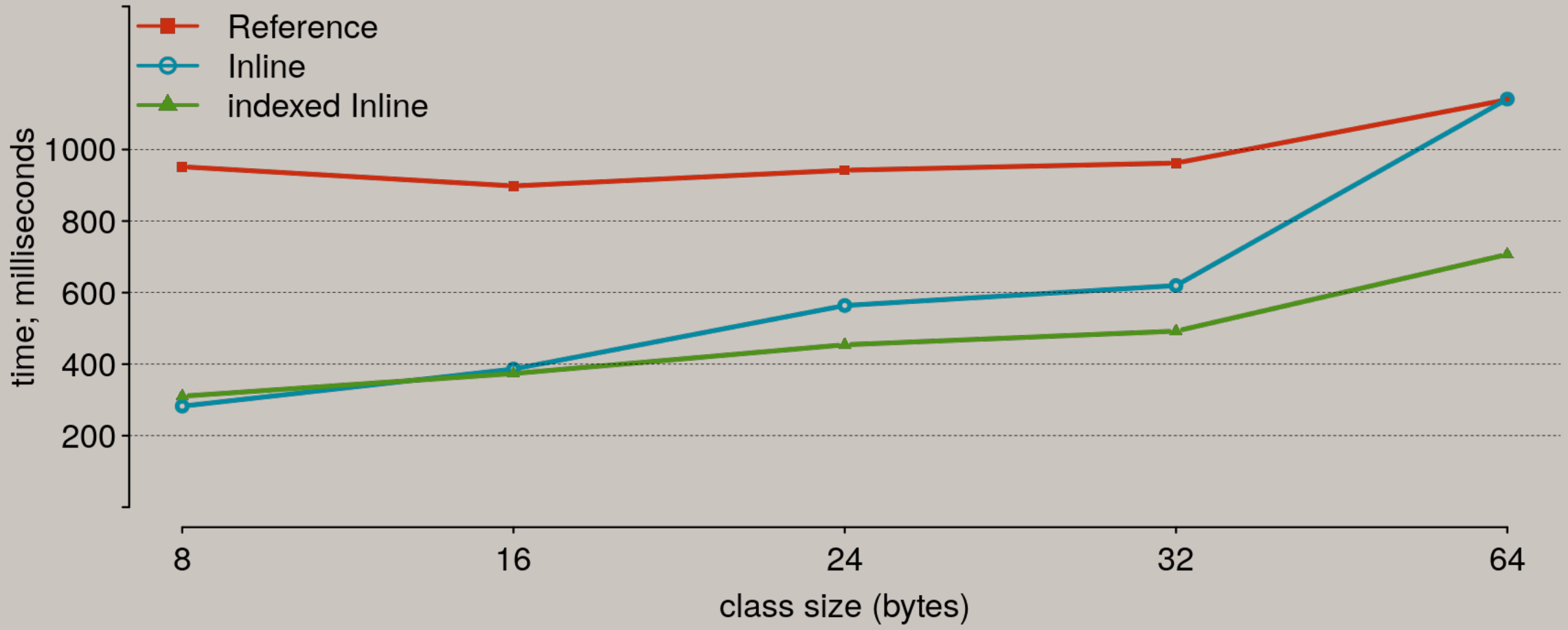
size==40000, fit into L3 cache



size==400000, slightly > L3 cache



size==4000000, much more L3 cache

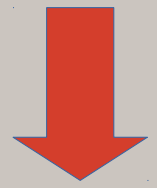


Move/Copy data

- Dense location is better than moving less

Inline classes

- Dense, HW-friendly memory layout
- More control on heap allocations:
  - less GC pressure
  - less GC barriers



Better performance!

Links

- Wiki:

  https://wiki.openjdk.java.net/display/valhalla/Main

- Mailing lists:

  http://mail.openjdk.java.net/mailman/listinfo/valhalla-dev

  http://mail.openjdk.java.net/mailman/listinfo/valhalla-spec-observers

- Repository:

  http://hg.openjdk.java.net/valhalla

Thank You

Sergey Kuksenko

Java Platform Group
Oracle
October, 2019