



Refactoring to Java 8

Trisha Gee (@trisha_gee)
Developer & Technical Advocate, JetBrains

Why Java 8?



It's Faster

- Performance Improvements in Common Data Structures
- Fork/Join Speed Improvements
- Changes to Support Concurrency
- ...and more

<http://bit.ly/refJ8>

Easy to Parallelize

Fewer Lines of Code

New Solutions to Problems

Minimizes Errors

Safety Check



Test Coverage

	Unit Test Coverage	Integration Test Coverage	System Test Coverage
org.mongodb	0% (0/0)	0% (0/0)	0% (0/0)
org.mongodb.morphia	80% (8/10)	78% (160/203)	77% (675/870)
org.mongodb.morphia.aggregation	87% (7/8)	73% (72/98)	82% (199/242)
org.mongodb.morphia.annotations	0% (0/0)	0% (0/0)	0% (0/0)
org.mongodb.morphia.converters	90% (28/31)	90% (113/125)	83% (467/556)
org.mongodb.morphia.dao	100% (1/1)	69% (27/39)	73% (44/60)
org.mongodb.morphia.geo	96% (28/29)	82% (95/115)	80% (242/301)
org.mongodb.morphia.logging	50% (1/2)	25% (7/28)	47% (23/48)
org.mongodb.morphia.logging.jdk	40% (2/5)	21% (12/56)	27% (42/153)
org.mongodb.morphia.mapping	100% (15/15)	88% (208/234)	85% (1169/1372)
org.mongodb.morphia.mapping.cache	100% (3/3)	60% (9/15)	62% (40/64)
org.mongodb.morphia.mapping.lazy	100% (3/3)	90% (9/10)	84% (45/53)
org.mongodb.morphia.mapping.lazy.proxy	100% (6/6)	58% (21/36)	61% (64/104)
org.mongodb.morphia.mapping.validation	100% (7/7)	82% (19/23)	82% (86/104)
org.mongodb.morphia.mapping.validation.classrules	90% (9/10)	66% (10/15)	66% (55/83)
org.mongodb.morphia.mapping.validation.fieldrules	90% (9/10)	91% (11/12)	82% (78/95)
org.mongodb.morphia.query	96% (26/27)	83% (205/246)	85% (771/905)
org.mongodb.morphia.query.validation	100% (23/23)	96% (92/95)	97% (197/203)
org.mongodb.morphia.utils	75% (6/8)	80% (46/57)	69% (247/357)

Performance Tests

```
# JMH 1.12 (released 73 days ago)
# VM version: JDK 1.8.0_91, VM 25.91-b15
# VM invoker: C:\Program Files\Java\jre1.8.0_91\bin\java.exe
# VM options: <none>
# Warmup: 10 iterations, 1 s each
# Measurement: 10 iterations, 1 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: com.mechanitis.BasicDAOBenchmark.originalIterationCode
# Parameters: (numberOfItems = 1)

# Run progress: 0.00% complete, ETA 04:40:00
# Fork: 1 of 10
# Warmup Iteration  1: 27846.545 ops/ms
# Warmup Iteration  2: 26967.979 ops/ms
# Warmup Iteration  3: 23699.331 ops/ms
# Warmup Iteration  4: 25039.632 ops/ms
# Warmup Iteration  5: 26682.563 ops/ms
# Warmup Iteration  6: 28453.666 ops/ms
# Warmup Iteration  7: 31094.559 ops/ms
# Warmup Iteration  8: 31502.342 ops/ms
# Warmup Iteration  9: 31321.723 ops/ms
# Warmup Iteration 10: 31956.021 ops/ms
Iteration   1: 31411.909 ops/ms
Iteration   2: 31639.921 ops/ms
Iteration   3: 31319.392 ops/ms
```

Decide on the Goals

Limit the Scope

Morphia

<https://github.com/mongodb/morphia>

Refactoring!



Lambda Expressions

Automatic Refactoring

- Predicate
- Comparator
- Runnable
- etc...

Abstract classes

Advanced Search

Collections & Streams API

Automatic Refactoring

- For loop to collect
- For loop to forEach
- ...with and without Streams

Manual Refactoring

Optional

```

    protected Class toClass(final Type t) {
        if (t == null) {
            return null;
        }
    }

    protected Optional<Class> toClass(final Optional<? extends Type> theType) {
        if (!theType.isPresent()) {
            return Optional.empty();
        }
        return toClass(theType.get());
    }

    protected Optional<Class> toClass(final Type t) {
        } else if (t instanceof Class) {
            return Optional.of((Class) t);
        } else if (t instanceof GenericArrayType) {
            final Type type = ((GenericArrayType) t).getGenericComponentType();

            aClass = (Class) type;
        }
        return Optional.of(Array.newInstance(aClass, 0).getClass());
    } else if (t instanceof ParameterizedType) {
        return Optional.ofNullable((Class) ((ParameterizedType) t).getRawType());
    } else if (t instanceof WildcardType) {
        return Optional.of((Class) ((WildcardType) t).getUpperBounds()[0]);
    }

    throw new RuntimeException("Generic TypeVariable not supported!");
}

// get the subtype T, T[]/List<T>/Map<?,T>; subtype of Long[], List<Long> is Long
subType = (realType.isArray()) ? realType.getComponentType() : ReflectionUtils.getParameterizedType(field, isMap ? 1
    Optional.ofNullable(realType.getComponentType())
    : ReflectionUtils.getParameterizedType(field, isMap ? 1 : 0);

if (isMap) {
}

```

MappedField.java

logging-slf4j/src/main.../morphia/logging/slf4j

* SLF4JLogger.java

morphia

* build.gradle

morphia/src/main/java/org/mongodb/morphia

* Key.java

morphia/src/main/java/...odb/morphia/converters

* Converters.java

* EnumSetConverter.java

* IterableConverter.java

* MapOfValuesConverter.java

morphia/src/main/java/org/mongodb/morphia/mapping

* DefaultCreator.java

* EmbeddedMapper.java

* EphemeralMappedField.java

* MappedClass.java

* MappedField.java

* Mapper.java

* ReferenceMapper.java

morphia/src/main/java/...hia/mapping/validation

* MappingValidator.java

morphia/src/main/java/.../validation/fieldrules

* MapKeyDifferentFromString.java

* MapNotSerializable.java

* ReferenceToUnidentifiable.java

morphia/src/main/java/org/mongodb/morphia/query

* QueryImpl.java

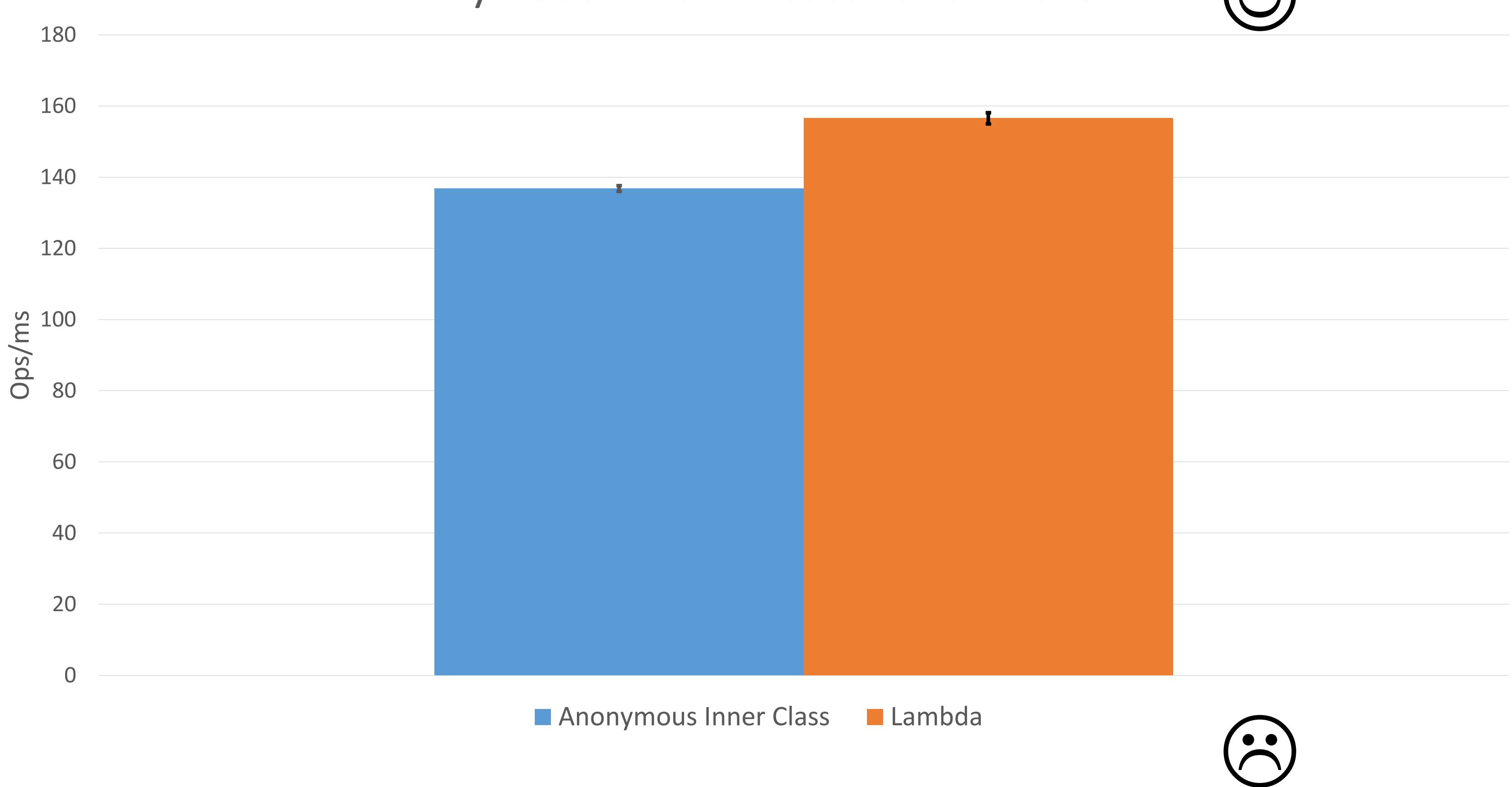
* QueryValidator.java

But what about performance?

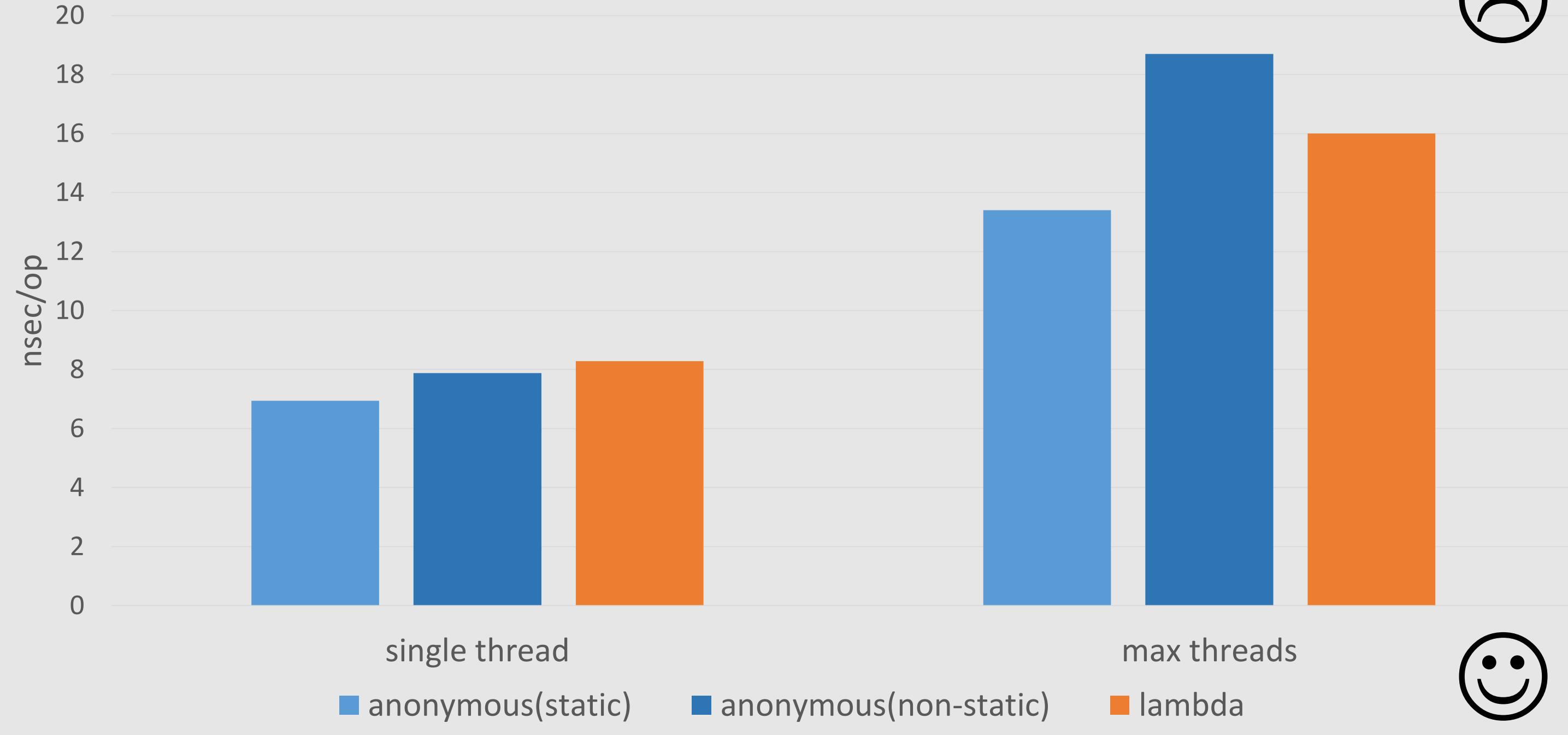


Lambdas

Anonymous Inner Classes vs Lambdas

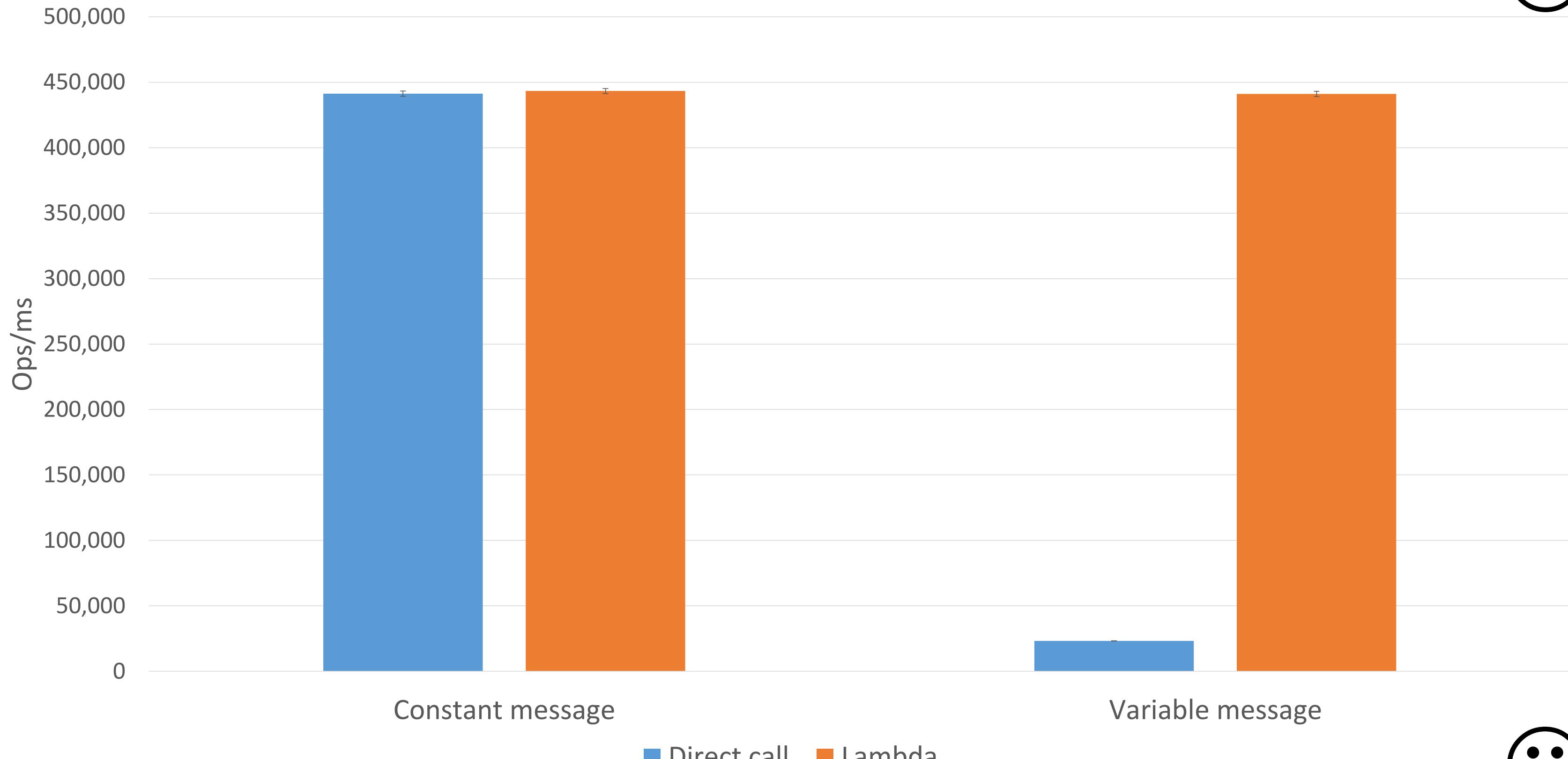


Performance of Capture



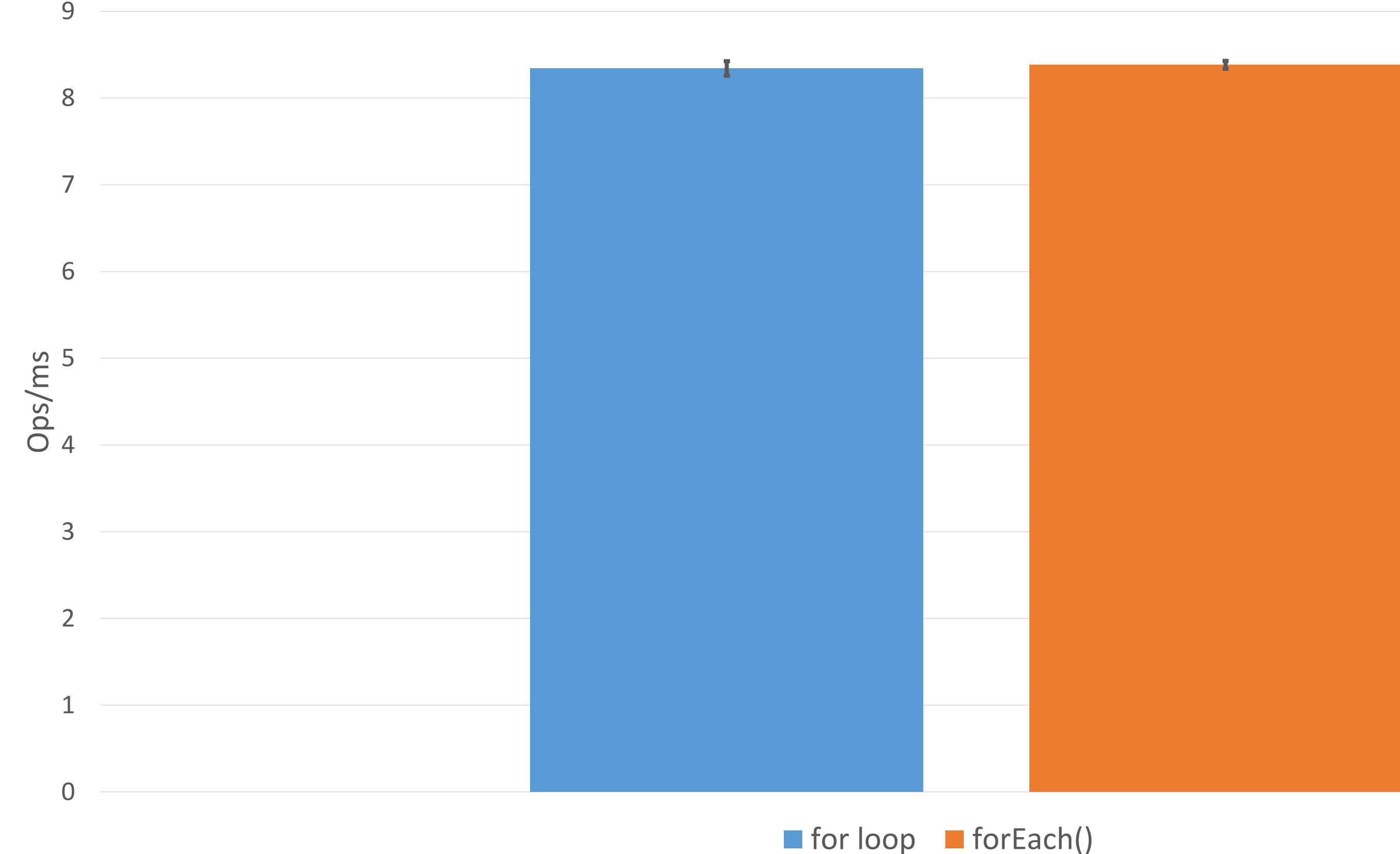
<http://www.oracle.com/technetwork/java/jvmls2013kuksen-2014088.pdf>

Logging Performance



Streams vs Iteration

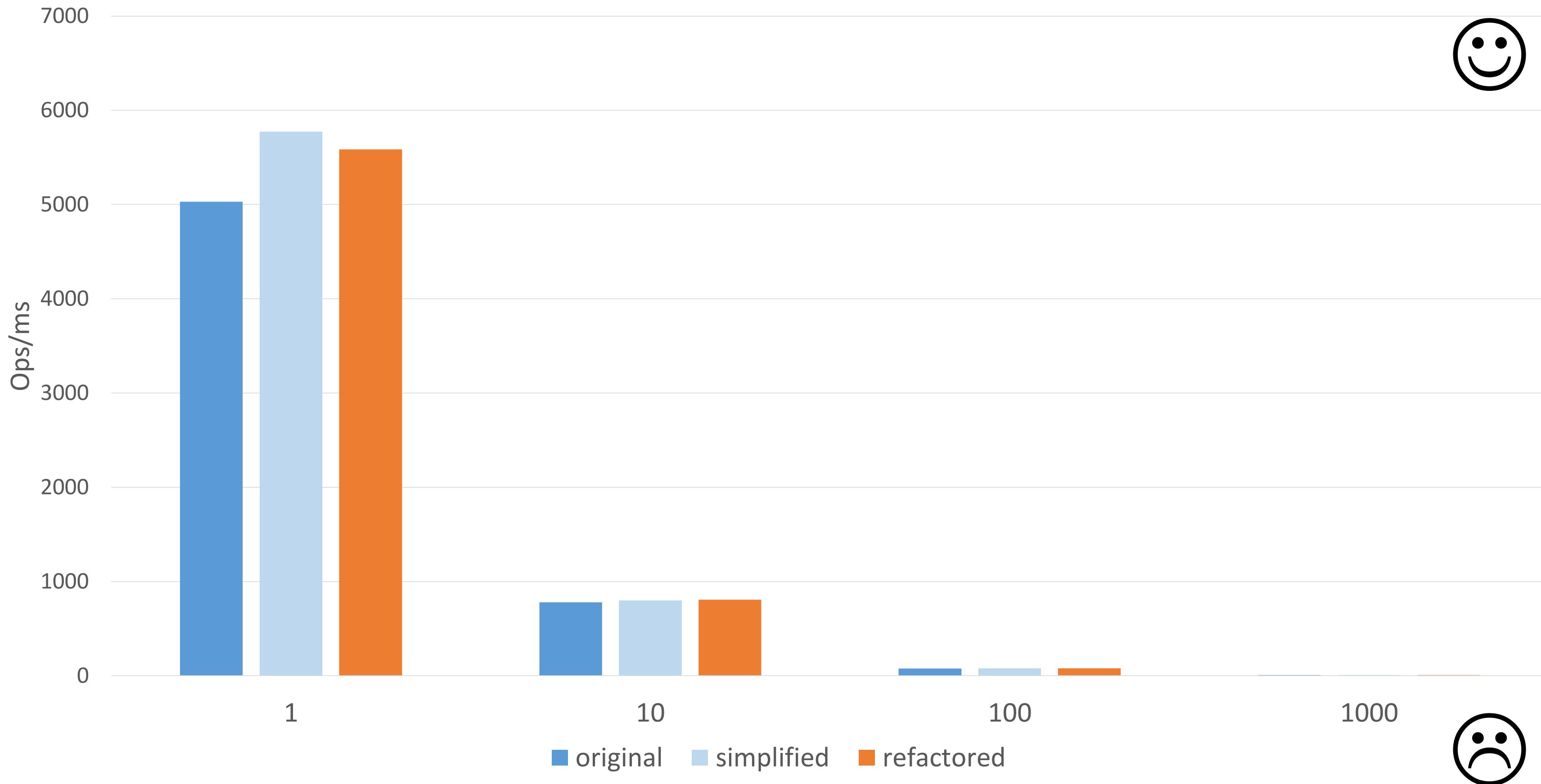
Iterator vs Stream (1000 elements)



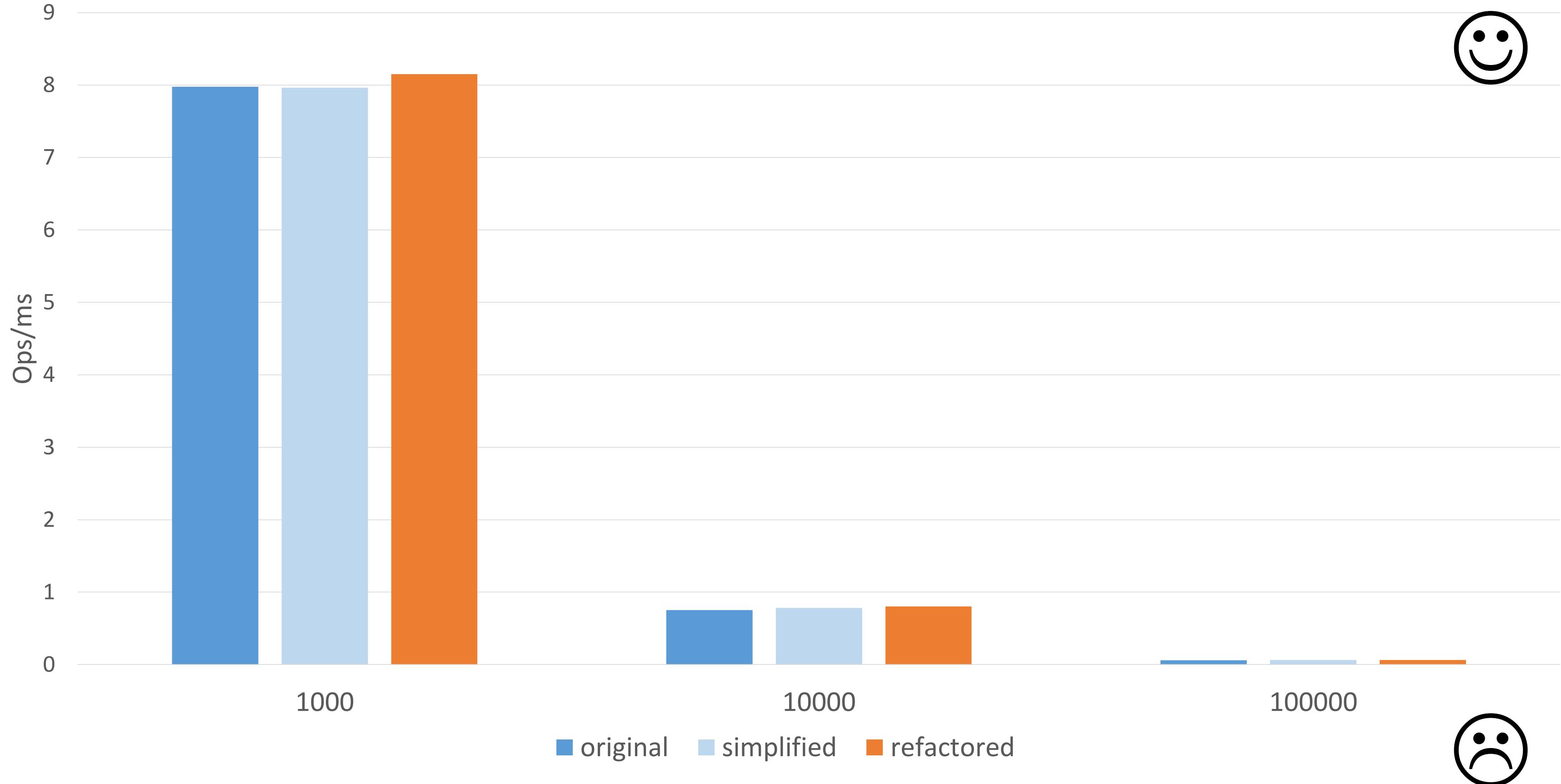
■ for loop ■ forEach()



IterHelper



IterHelper



BasicDAO – map & collect

```
protected List<?> keysToIds(final List<Key<T>> keys) {  
    final List<Object> ids = new ArrayList<Object>(keys.size() * 2);  
    for (final Key<T> key : keys) {  
        ids.add(key.getId());  
    }  
    return ids;  
}
```

278

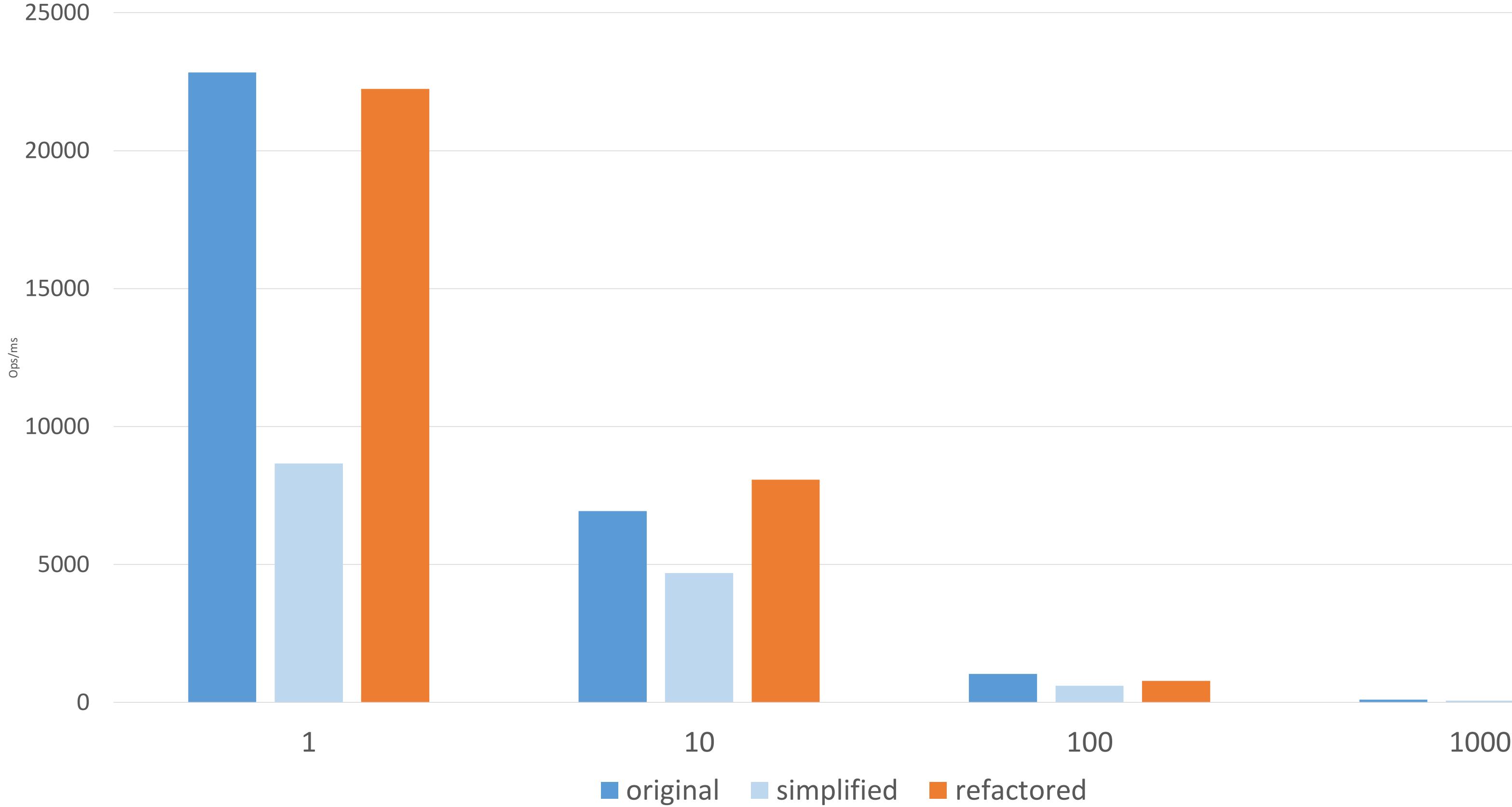
279

280

281

```
protected List<?> keysToIds(final List<Key<T>> keys) {  
    return keys.stream().map(Key::getId).collect(toList());  
}
```

BasicDAO

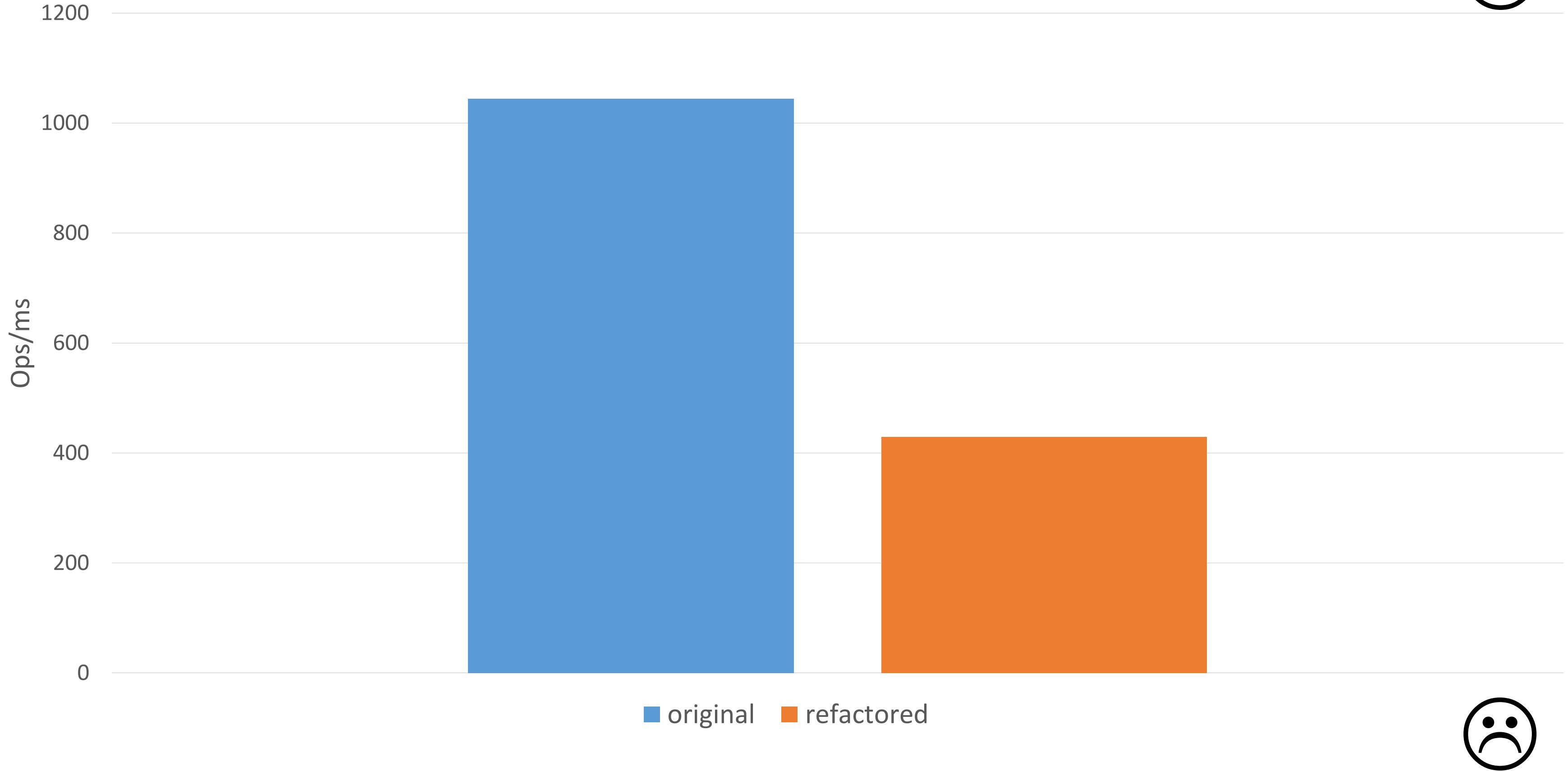


DuplicatedAttributeNames – filter, map & collect

```
@Override
public void check(final Mapper mapper, final MappedClass mc, final Set<ConstraintViolation>
    final Set<String> foundNames = new HashSet<String>();
    for (final MappedField mappedField : mc.getPersistenceFields()) {
        for (final String name : mappedField.getLoadNames()) {
            if (!foundNames.add(name)) {
                ve.add(new ConstraintViolation(Level.FATAL, mc, mappedField, getClass(),
                    "Mapping to MongoDB field name '" + name
                    + "' is duplicated; you cannot map different j
                )
            }
        }
    }
}
```

```
22
23 ↑ @Override
24     public void check(final Mapper mapper, final MappedClass mc, final Set<ConstraintViolation>
25         final Set<String> foundNames = new HashSet<>();
26         for (final MappedField mappedField : mc.getPersistenceFields()) {
27             // TODO: I suspect a re-think of this approach would lead to further code simplification
28             ve.addAll(mappedField.getLoadNames().stream()
29                     .filter(name -> !foundNames.add(name))
30                     .map(name -> new ConstraintViolation(FATAL, mc, mappedField, getClass(),
31                         "Mapping to MongoDB field name '" + name
32                         + "' is duplicated; you cannot map different j
33                     )
34             );
35     }
36 }
```

DuplicatedAttributeNames



EntityScanner– forEach

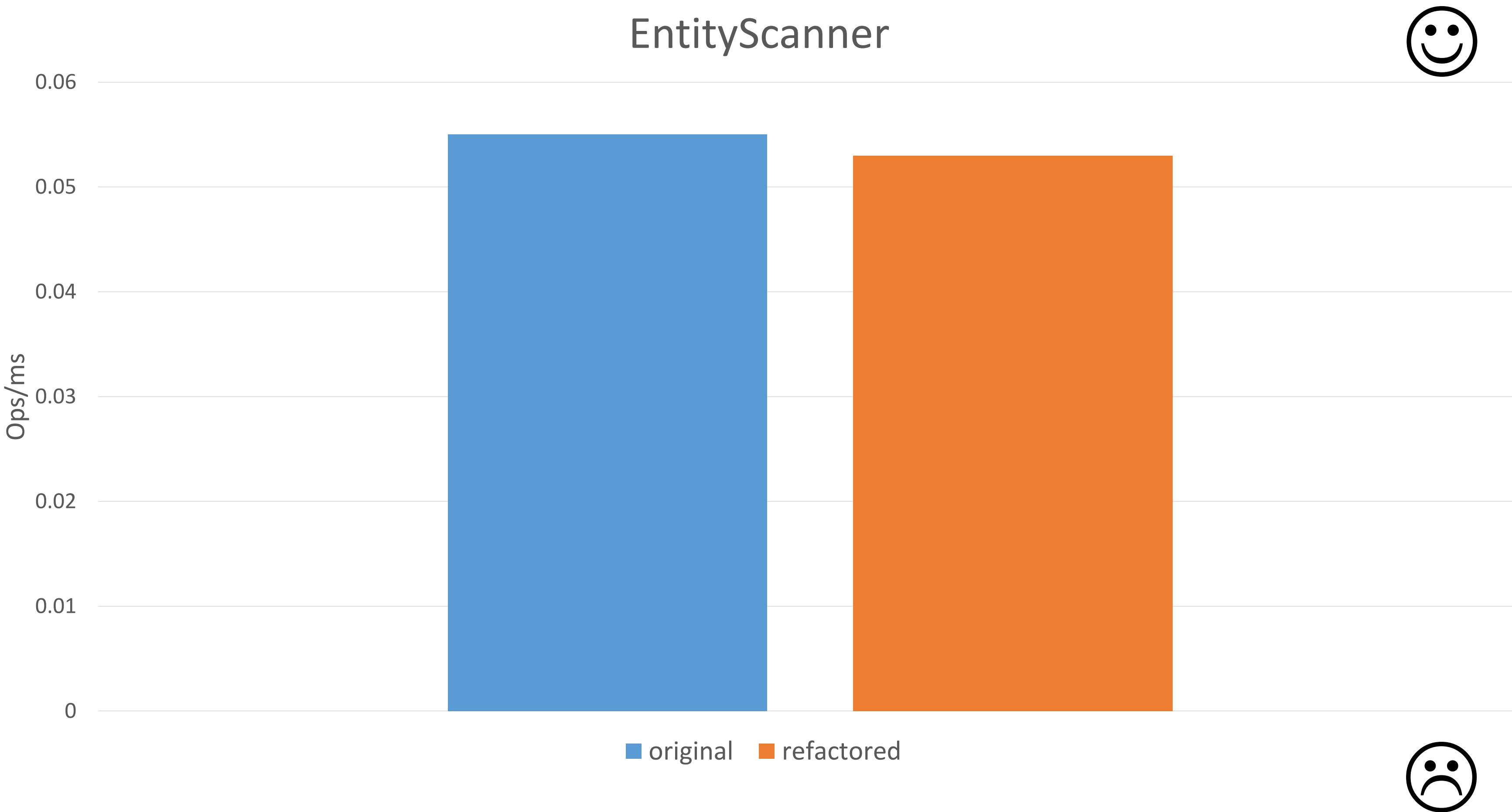
```
final Reflections r = new Reflections(conf);

final Set<Class<?>> entities = r.getTypesAnnotatedWith(Entity.class);
for (final Class<?> c : entities) {
    m.map(c);
}
```

67

```
new Reflections(conf).getTypesAnnotatedWith(Entity.class).forEach(
```

EntityScanner



DatastoreImpl – filter & forEach

```
if (indexes != null) {
    for (final Indexes idx : indexes) {
        if (idx.value().length > 0) {

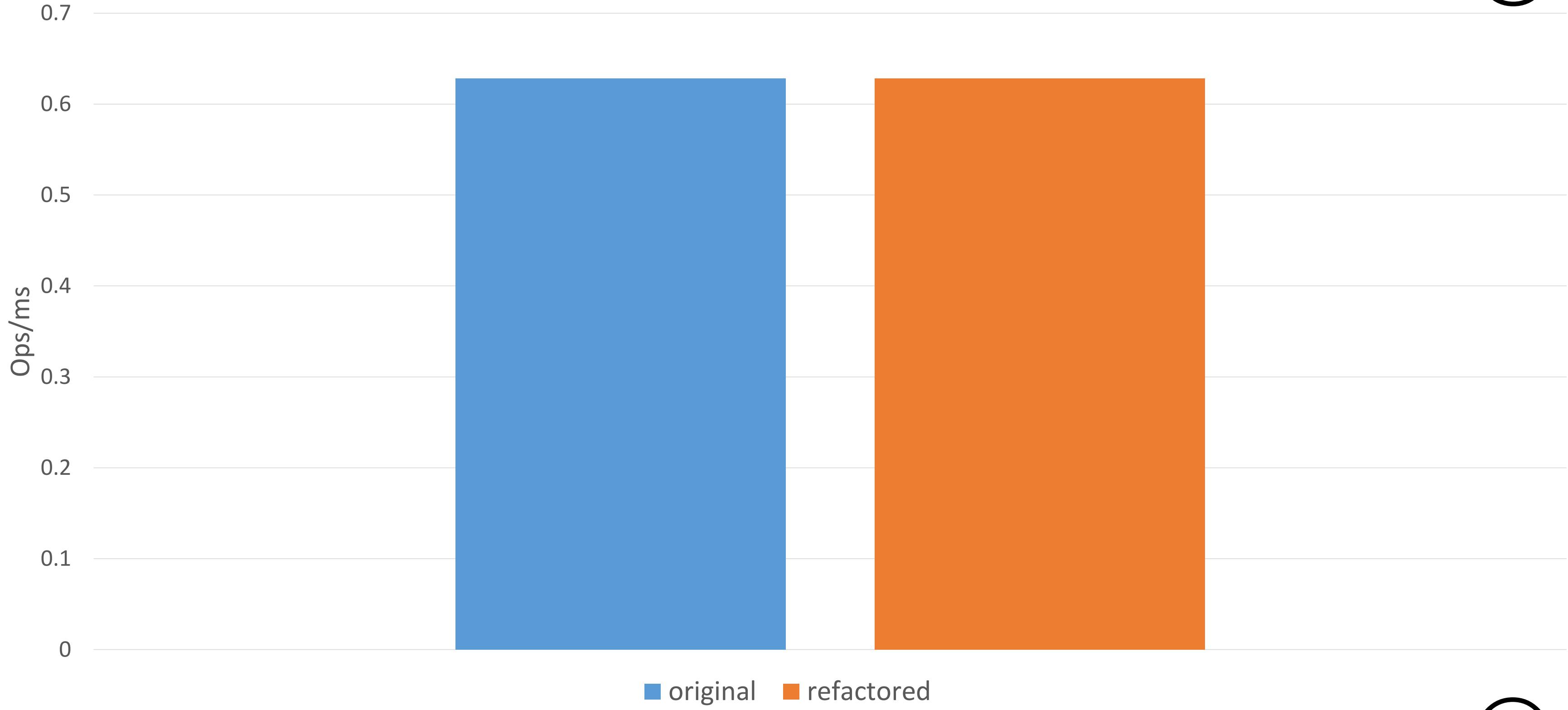
            for (final Index index : idx.value()) {
                if (index.fields().length != 0) {
                    ensureIndex(mc, dbColl, index.fields(), index.options(), background);
                } else {
                    LOG.warning(() -> format("This index on '%s' is using deprecated config", index.name())
                        + "use the fields value on @Index: %s", mc.getClazz().getName());
                    final BasicDBObject fields = parseFieldsString(index.value(), mc.getClazz());
                    !index.disableValidation(), parentMCs, parentMFs);
                    ensureIndex(dbColl, index.name(), fields, index.unique(), index.drop(),
                        index.background() ? index.background() : background, index.sparse());
                }
            }
        }
    }
}

1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481

if (indexes != null) {
    indexes.stream()
        .filter(idx -> idx.value().length > 0)
        .forEach(idx -> {
            for (final Index index : idx.value()) {
                if (index.fields().length != 0) {
                    ensureIndex(mc, dbColl, index.fields(), index.options(), background);
                } else {
                    LOG.warning(() -> format("This index on '%s' is using deprecated config", index.name())
                        + "update to use the fields value on @Index: %s", mc.getClazz().getName());
                    final BasicDBObject fields = parseFieldsString(index.value(), mc.getClazz());
                    !index.disableValidation(), parentMCs, parentMFs);
                    ensureIndex(dbColl, index.name(), fields, index.unique(),
                        index.background() ? index.background() : background, index.sparse());
                }
            }
        });
}

1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
```

DatastoreImpl



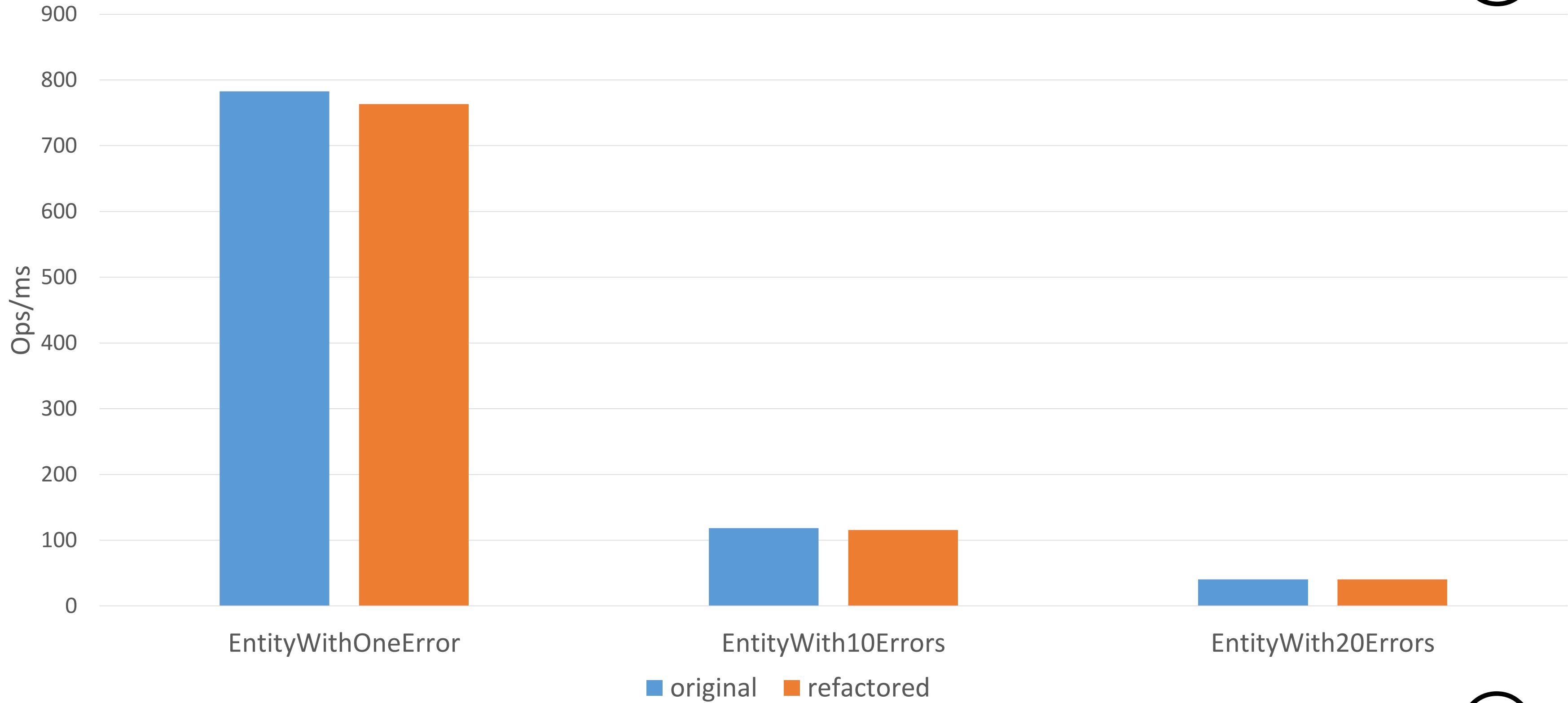
MappingValidator – single stream operation

```
final List<LogLine> l = new ArrayList<LogLine>();
for (final ConstraintViolation v : ve) {
    l.add(new LogLine(v));
}
sort(l);

for (final LogLine line : l) {
    line.log(LOG);
}
```

```
98     ve.stream()
99         .map(LogLine::new)
100        .sorted()
101        .forEach(logLine -> logLine.log(LOG));
102 //CHECKSTYLE:ON
```

MappingValidator

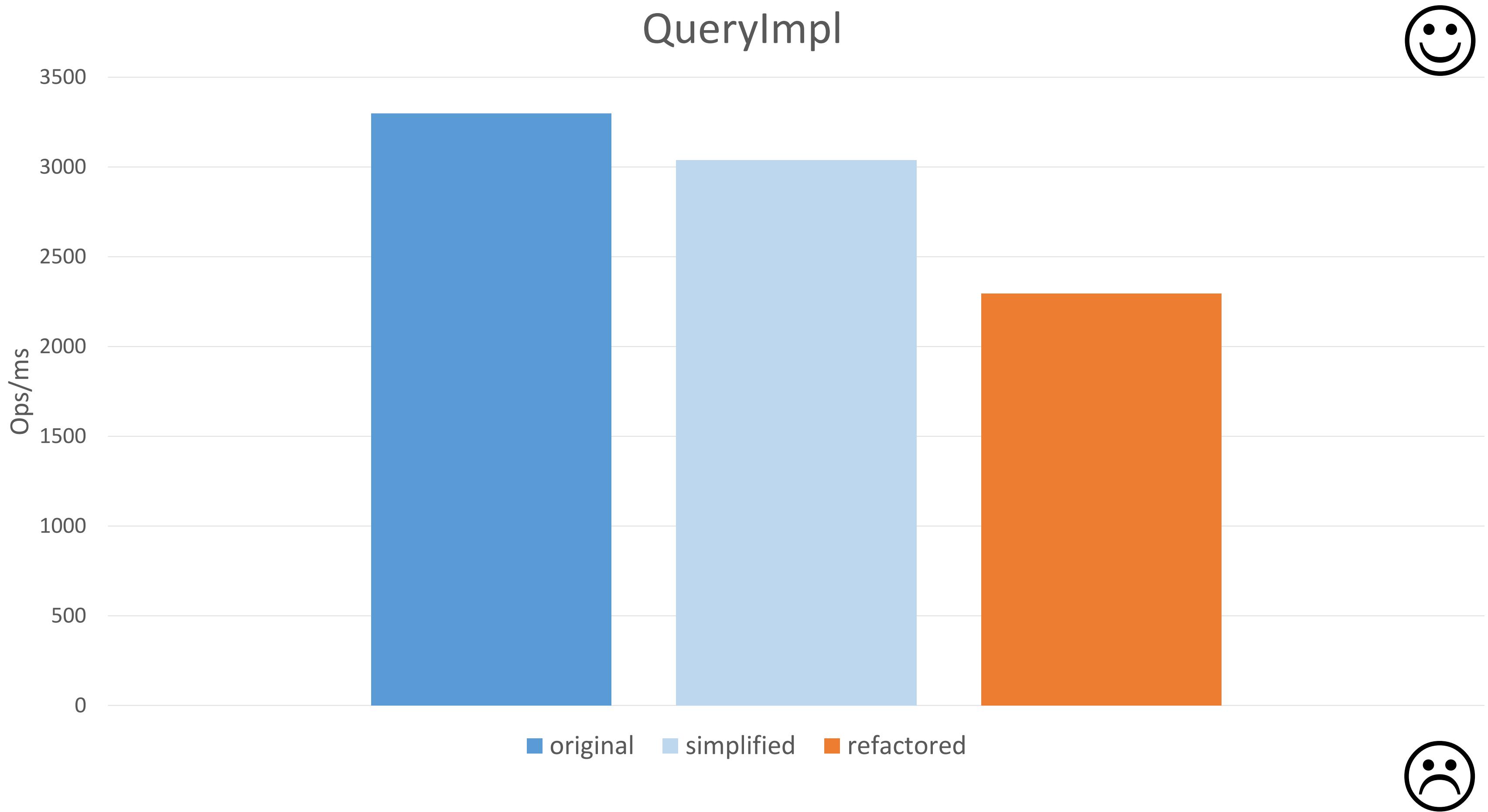


QueryImpl – multiple operations

```
final MappedClass mc = ds.getMapper().getMappedClass(clazz);
final List<String> fields = new ArrayList<String>(mc.getPersistenceFields());
for (final MappedField mf : mc.getPersistenceFields()) {
    fields.add(mf.getNameToStore());
}
retrievedFields(true, fields.toArray(new String[fields.size()]));
```

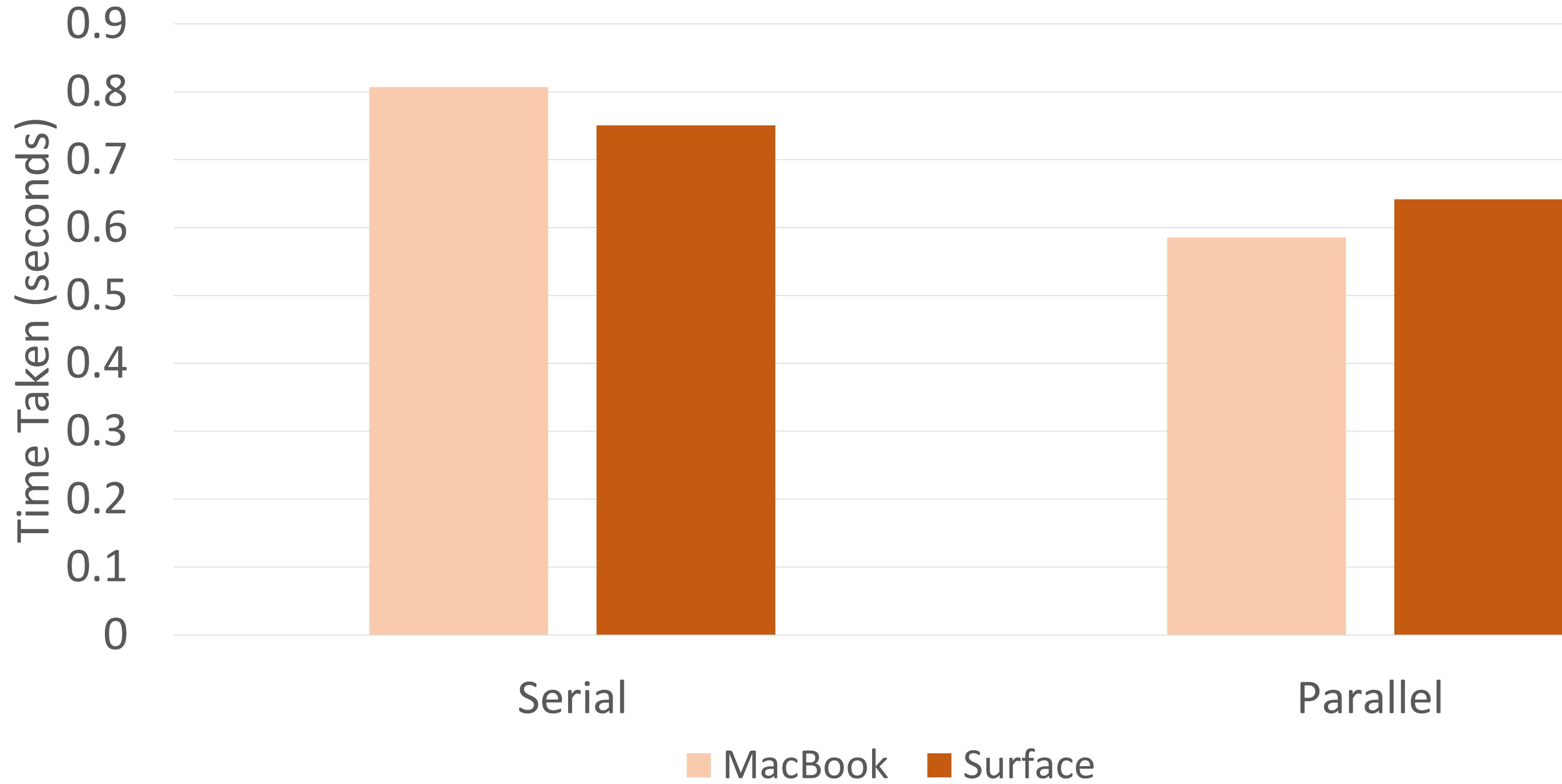
```
495 final String[] fields = ds.getMapper().getMappedClass(clazz).get
496 .stream()
497 .map(MappedField::getNameToStore)
498 .collect(Collectors.toList())
499 .toArray(new String[0]);
500 retrievedFields(true, fields);
```

QueryImpl

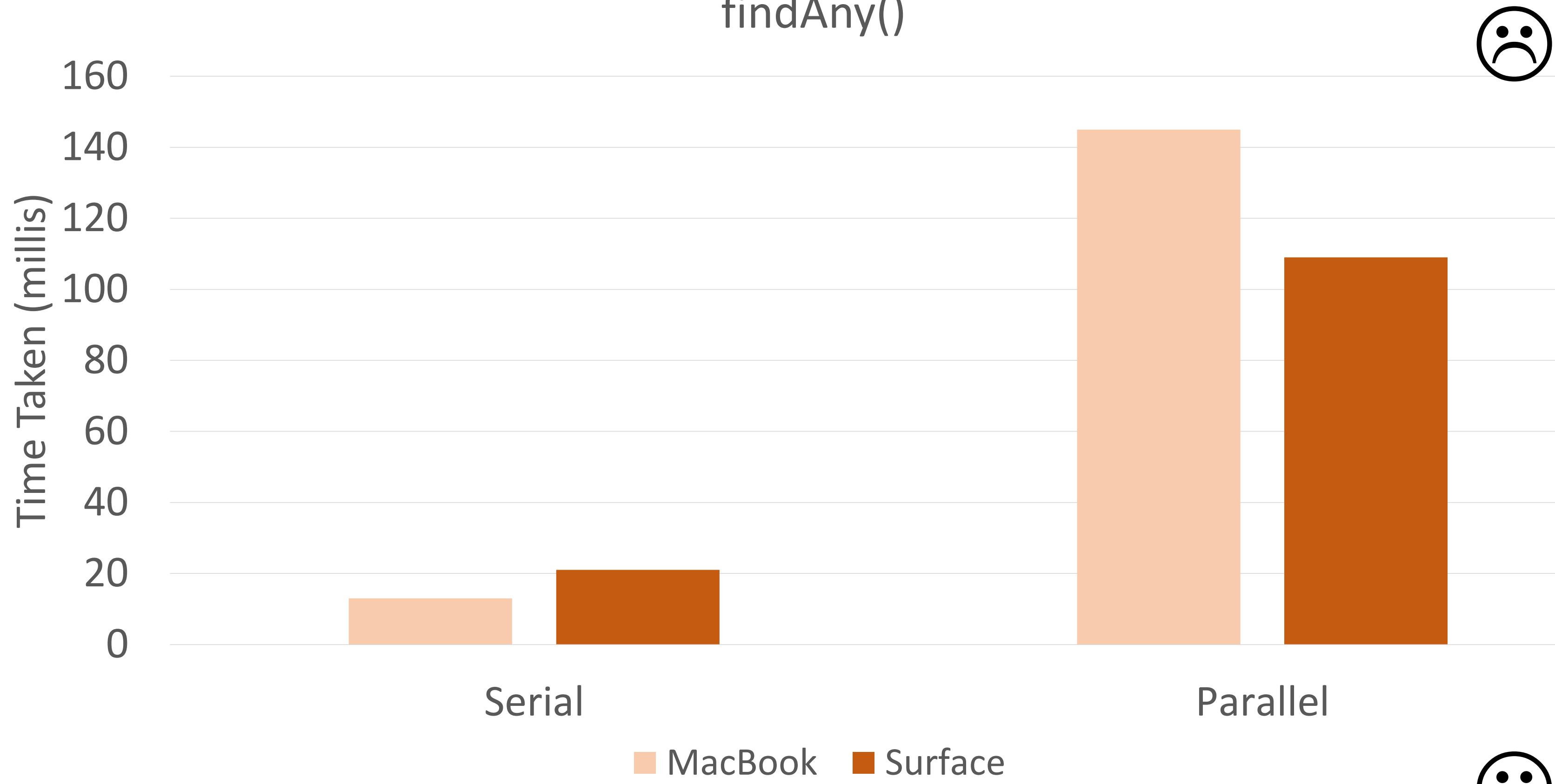


Going parallel

map()



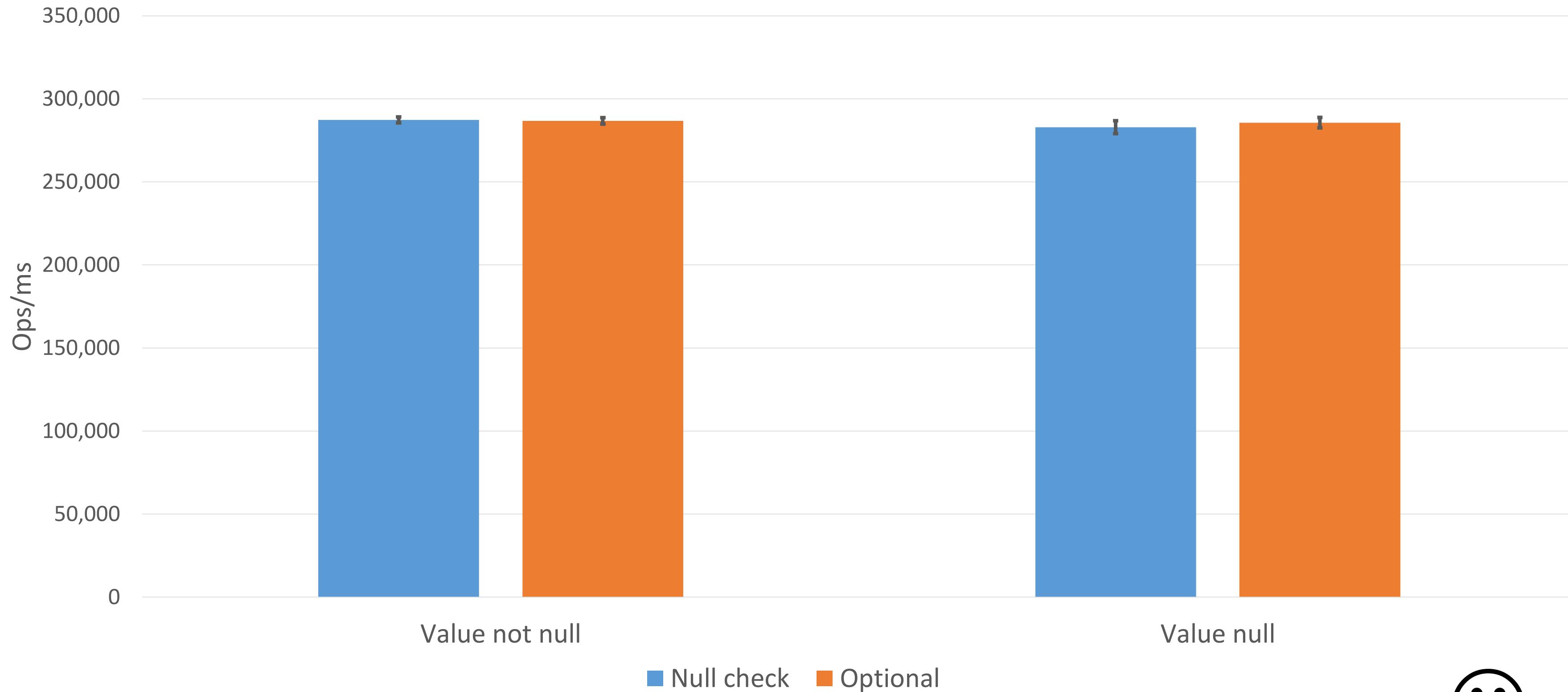
findAny()



Optional

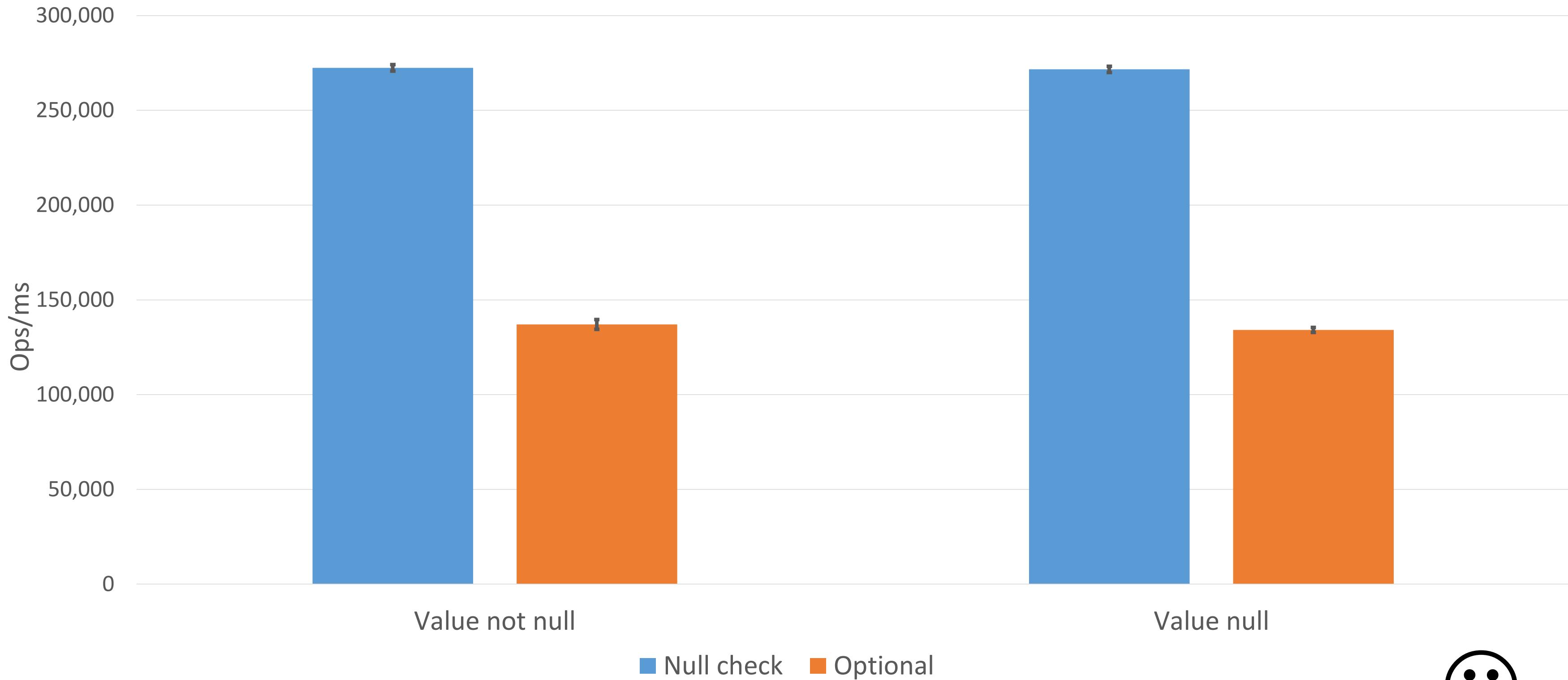


Compare Constant Field Value with Null





Compare Variable Field Value with Null



Summary



Sometimes new idioms decrease clutter

```
protected List<?> keysToIds(final List<Key<T>> keys) {  
    final List<Object> ids = new ArrayList<Object>(keys.size() * 2);  
    for (final Key<T> key : keys) {  
        ids.add(key.getId());  
    }  
    return ids;  
}
```

278

279

280

```
protected List<?> keysToIds(final List<Key<T>> keys) {  
    return keys.stream().map(Key::getId).collect(toList());  
}
```

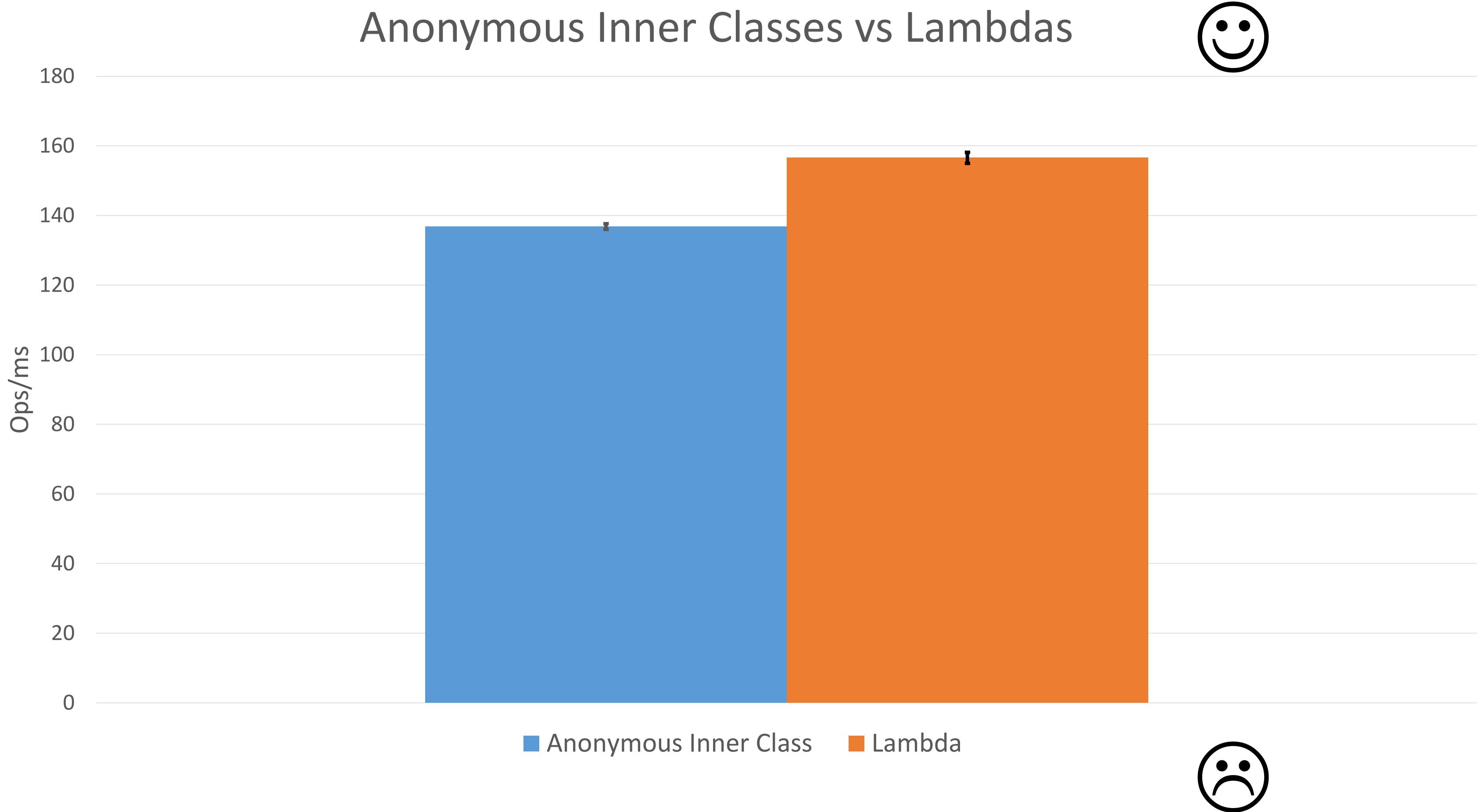
...but sometimes they don't

```
final List<Indexes> indexes = mc.getAnnotations(Indexes.class);
if (indexes != null) {
    for (final Indexes idx : indexes) {
        if (idx.value().length > 0) {
            for (final Index index : idx.value()) {
                if (index.fields().length != 0) {
                    ensureIndex(mc, dbColl, index.fields(), index.opt
                } else {
                    LOG.warning(() -> format("This index on '%s' is u
                        + "use the fields value on @Index: %s", m
                    final BasicDBObject fields = parseFieldsString(in
                        !index.disableValidation(), parentMCs, pa
                    ensureIndex(dbColl, index.name(), fields, index.u
                        index.background() ? index.background() :
                    }
                }
            }
        }
    }
}
```

```
final List<Indexes> indexes = mc.getAnnotations(Indexes.class);
// TODO: Questionable value add
if (indexes != null) {
    indexes.stream()
        .filter(idx -> idx.value().length > 0)
        .forEach(idx -> {
            for (final Index index : idx.value()) {
                if (index.fields().length != 0) {
                    ensureIndex(mc, dbColl, index.fields(), in
                } else {
                    LOG.warning(() -> format("This index on '%
                        + "update to use the fields value
                    final BasicDBObject fields = parseFieldsSt
                        !index.disableValidation(), parent
                    ensureIndex(dbColl, index.name(), fields,
                        index.background() ? index.backgro
                            : background, i
                }
            }
        });
}
```

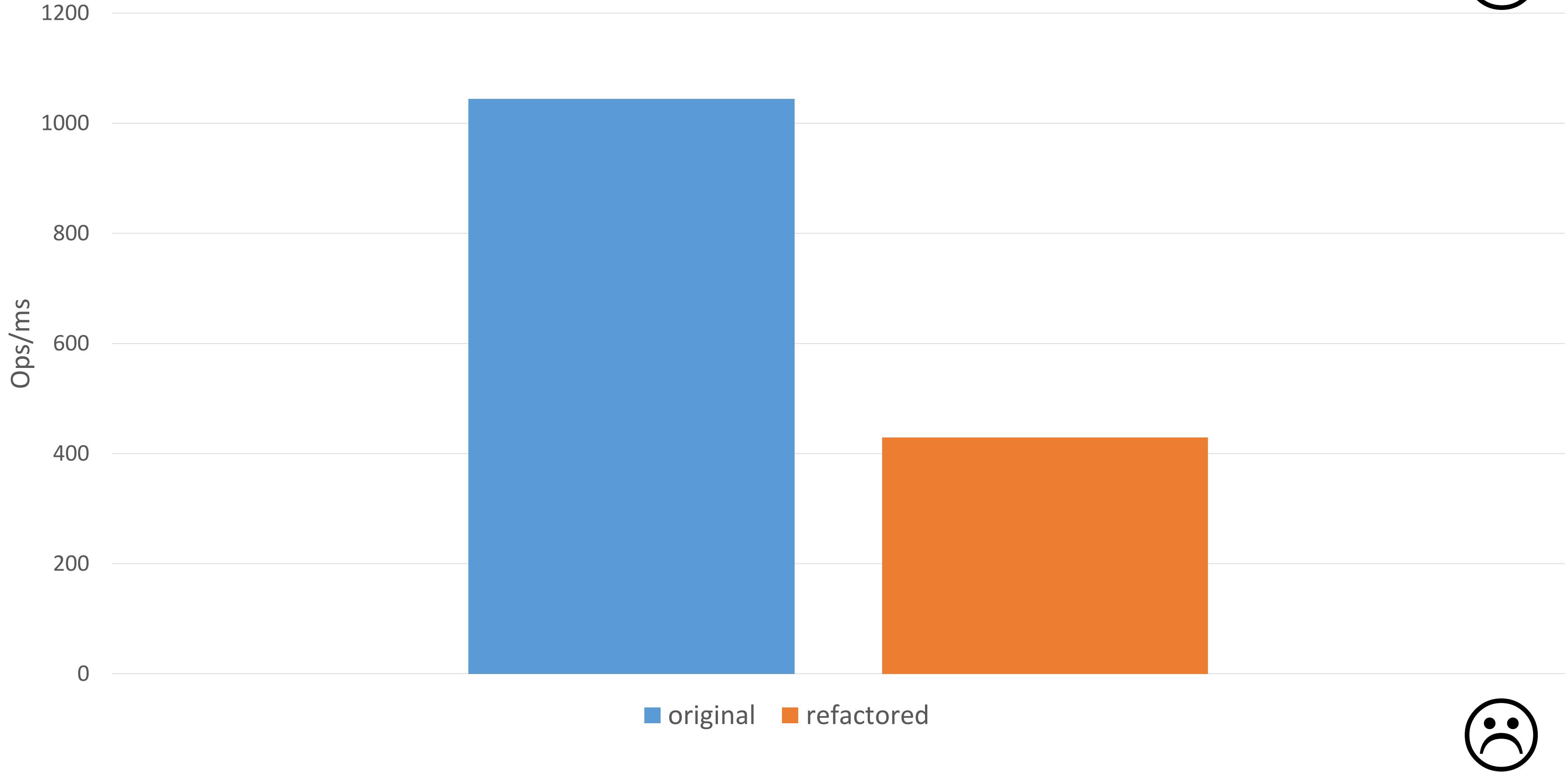
Sometimes the new features improve performance

Anonymous Inner Classes vs Lambdas



...and sometimes they don't

DuplicatedAttributeNames



Sometimes a new feature makes life easier

```
if (LOG.isTraceEnabled()) {  
    LOG.info("Executing " + cmd.toString());  
}
```

595

```
LOG.trace(() -> "Executing " + cmd.toString());
```

...sometimes not so much

```

    protected Class toClass(final Type t) {
        if (t == null) {
            return null;
        }
    }

    protected Optional<Class> toClass(final Optional<? extends Type> theType) {
        if (!theType.isPresent()) {
            return Optional.empty();
        }
        return toClass(theType.get());
    }

    protected Optional<Class> toClass(final Type t) {
        } else if (t instanceof Class) {
            return Optional.of((Class) t);
        } else if (t instanceof GenericArrayType) {
            final Type type = ((GenericArrayType) t).getGenericComponentType();

            aClass = (Class) type;
        }
        return Optional.of(Array.newInstance(aClass, 0).getClass());
    } else if (t instanceof ParameterizedType) {
        return Optional.ofNullable((Class) ((ParameterizedType) t).getRawType());
    } else if (t instanceof WildcardType) {
        return Optional.of((Class) ((WildcardType) t).getUpperBounds()[0]);
    }

    throw new RuntimeException("Generic TypeVariable not supported!");
}

// get the subtype T, T[]/List<T>/Map<?,T>; subtype of Long[], List<Long> is Long
subType = (realType.isArray()) ? realType.getComponentType() : ReflectionUtils.getParameterizedType(field, isMap ? 1
    Optional.ofNullable(realType.getComponentType())
    : ReflectionUtils.getParameterizedType(field, isMap ? 1 : 0);

if (isMap) {
}

```

MappedField.java

logging-slf4j/src/main.../morphia/logging/slf4j

* SLF4JLogger.java

morphia

* build.gradle

morphia/src/main/java/org/mongodb/morphia

* Key.java

morphia/src/main/java/...odb/morphia/converters

* Converters.java

* EnumSetConverter.java

* IterableConverter.java

* MapOfValuesConverter.java

morphia/src/main/java/org/mongodb/morphia/mapping

* DefaultCreator.java

* EmbeddedMapper.java

* EphemeralMappedField.java

* MappedClass.java

* MappedField.java

* Mapper.java

* ReferenceMapper.java

morphia/src/main/java/...hia/mapping/validation

* MappingValidator.java

morphia/src/main/java/.../validation/fieldrules

* MapKeyDifferentFromString.java

* MapNotSerializable.java

* ReferenceToUnidentifiable.java

morphia/src/main/java/org/mongodb/morphia/query

* QueryImpl.java

* QueryValidator.java

**Some refactoring can safely be done
automatically**

```
new EntityScanner(m, new Predicate<String>() {  
  
    @Override  
    public boolean apply(final String input) {  
        return input.startsWith(EntityScannerTest.class.getPackage().getName());  
    }  
});
```

24

```
new EntityScanner(m, input -> input.startsWith(EntityScannerTest.class.getPackage().getName()));
```

...often you need a human brain

```
74 //TODO: I think returning a collection of errors from check() would simplify
75 final Set<ConstraintViolation> ve = new TreeSet<>((o1, o2) -> o1.getLevel().c
76
77 final List<ClassConstraint> rules = getConstraints();
78 classes.forEach(mappedClass -> rules.forEach(v -> v.check(mapper, mappedClas
79
80 if (!ve.isEmpty()) {
81     final ConstraintViolation worst = ve.iterator().next();
82     final Level maxLevel = worst.getLevel();
83     if (maxLevel.ordinal() >= Level.FATAL.ordinal()) {
84         throw new ConstraintViolationException(ve);
85     }
86
87     // sort by class to make it more readable
88     final List<LogLine> l = new ArrayList<LogLine>();
89     for (final ConstraintViolation v : ve) {
90         l.add(new LogLine(v));
91     }
92     sort(l);
93
94     for (final LogLine line : l) {
95         line.log(LOG);
96     }
97 }
```

Conclusion



Should you migrate your code to Java 8?

It Depends

Always remember what your goal is

And compare results to it

Understand what may impact performance

And if in doubt, measure

Code may magically improve

Or you may expose areas for improvement

Your tools can help you

But you need to apply your brain too



<http://bit.ly/refJ8>

@trisha_gee