# Refactoring to Eclipse Collections
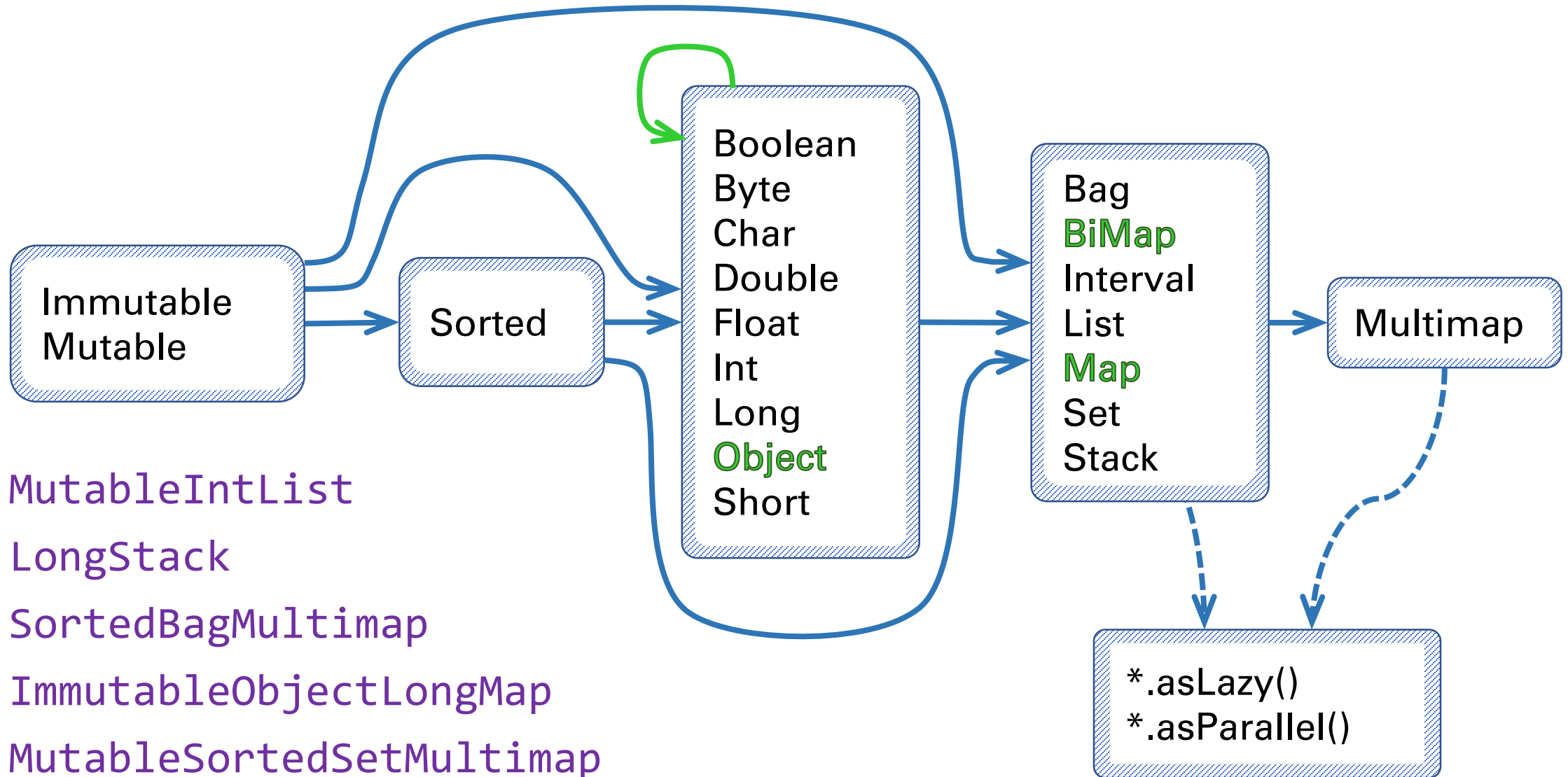
Making Your Java Streams
Leaner, Meaner, and Cleaner

# Introduction

- ## What is Eclipse Collections?
  - Feature rich, memory efficient Java Collections framework
- ## History
  - Eclipse Collections started off as an internal collections framework named Caramel at Goldman Sachs in 2004
  - In 2012, it was open sourced to GitHub as a project called GS Collections
  - GS Collections was migrated to the Eclipse Foundation, re-branded as Eclipse Collections in 2015
- ## Learn Eclipse Collections with Kata
- ## Eclipse Collections is open for contributions!

# Any Types You Need



Immutable
Mutable

Sorted

Boolean
Byte
Char
Double
Float
Int
Long
Object
Short

Bag
BiMap
Interval
List
Map
Set
Stack

Multimap

*.asLazy()
*.asParallel()

MutableIntList

LongStack

SortedBagMultimap

ImmutableObjectLongMap

MutableSortedSetMultimap

# Instantiate Them Using Factories

| Primitive Type | Container Type | Mutability | Multimap | Initialized | Lazy |
|---|---|---|---|---|---|
| Boolean | Bags | .mutable | .bag | .empty()\|.of()\|.with() | .asLazy() |
| Byte | BiMaps | .immutable | .list | .of(one)\|.with(one) | |
| Char | Lists | .fixedSize | .set | ... | |
| Double | Maps | | .sortedSet | .of(one,…,ten) | |
| Float | Multimaps | | | .with(one,…,ten) | |
| Int | Sets | | | .of(... elements) | |
| Long | SortedBags | | | .with(... elements) | |
| Object | SortedMaps | | | .ofAll(Iterable) | |
| Short | SortedSets | | | .withAll(Iterable) | |
| | Stacks | | | | |

ImmutableLongStack

LongStacks.*immutable*.with(1, 2, 3).asLazy()

LazyLongIterable

4

# Methods by Category - Highlights

## transform

collect**[With]**
collect**[Boolean,Byte,Char,Double,Float,Int,Long,Short]**
collectIf
collectKeysAndValues
collectValues
collectWithIndex
collectWithOccurrences
flatCollect

## group

groupBy
groupByEach
groupByUniqueKey
sumBy**[Double,Float,Int,Long]**
sumOf**[Double,Float,Int,Long]**
aggregateBy
aggregateInPlaceBy

## convert

toArray
toBag
toImmutable
toList
toMap
toReversed
toSet
toSortedBag**[By]**
toSortedList**[By]**
toSortedMap
toSortedSet
toSortedSet**[By]**
toStack
toString

## filter

select**[With]**
selectByOccurrences
selectInstancesOf
reject**[With]**
partition**[With]**
partitionWhile

## wrap

asLazy
asParallel
asReversed
asSynchronized
asUnmodifiable

## find

detect**[With]**
detect**[With]**IfNone
detect**[With]**Optional
max**[By]**
min**[By]**

## test

allSatisfy**[With]**
anySatisfy**[With]**
noneSatisfy**[With]**
notEmpty
isEmpty

# Methods – Lots More…

# Why Refactor to EC?

Lazy or Eager

TestUtils

Testability/Debuggability

No "Bun" methods

Drop-in replacement for JDK types

Concise Code —— Benefits —— It's Easy!

Intuitive humane APIs

Rich[er] APIs

Memory Savings

Comprehensive Support
for Primitive Types

Better Performance [citation needed]

Efficient Maps and Sets
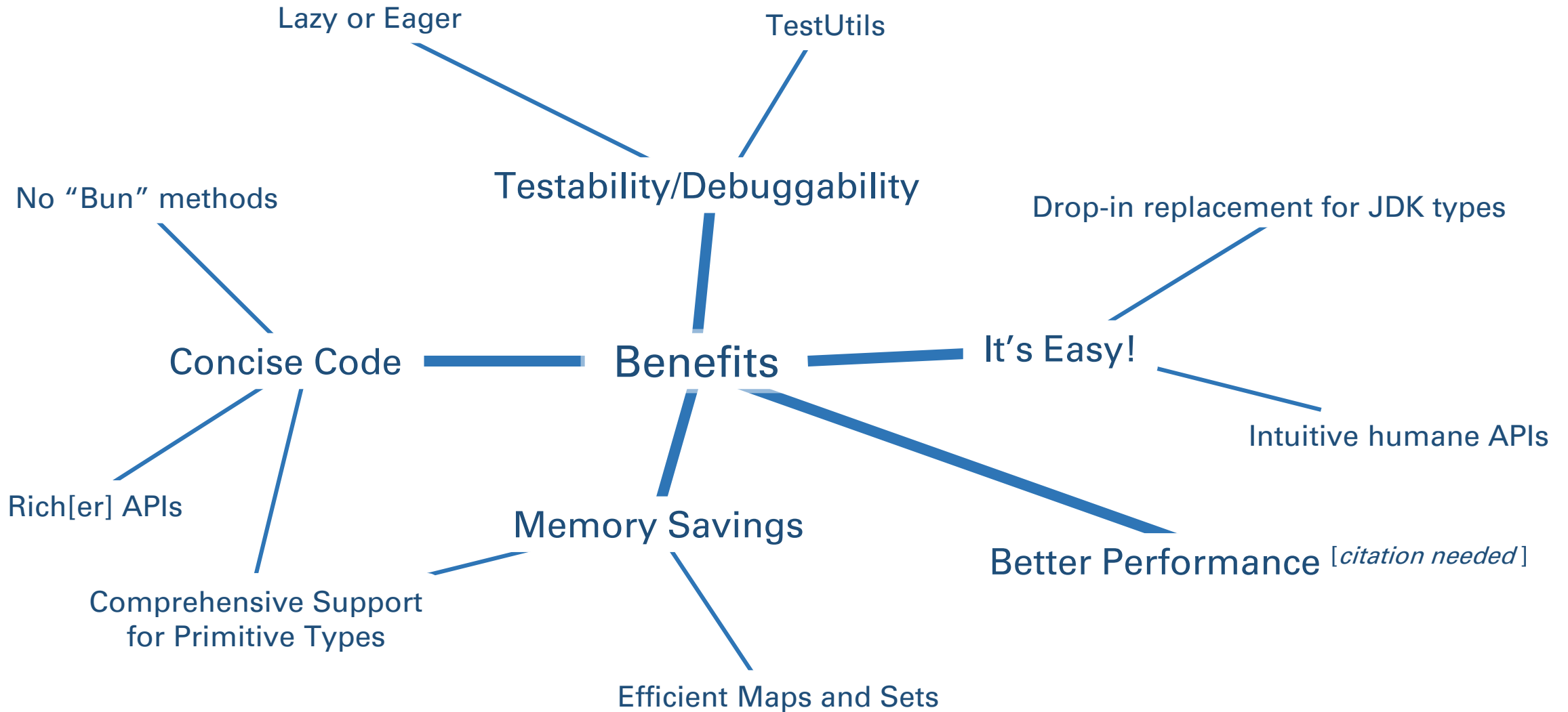
# Let's Do It!

# JMH Benchmark Results



Ops/s

EC ■ JDK
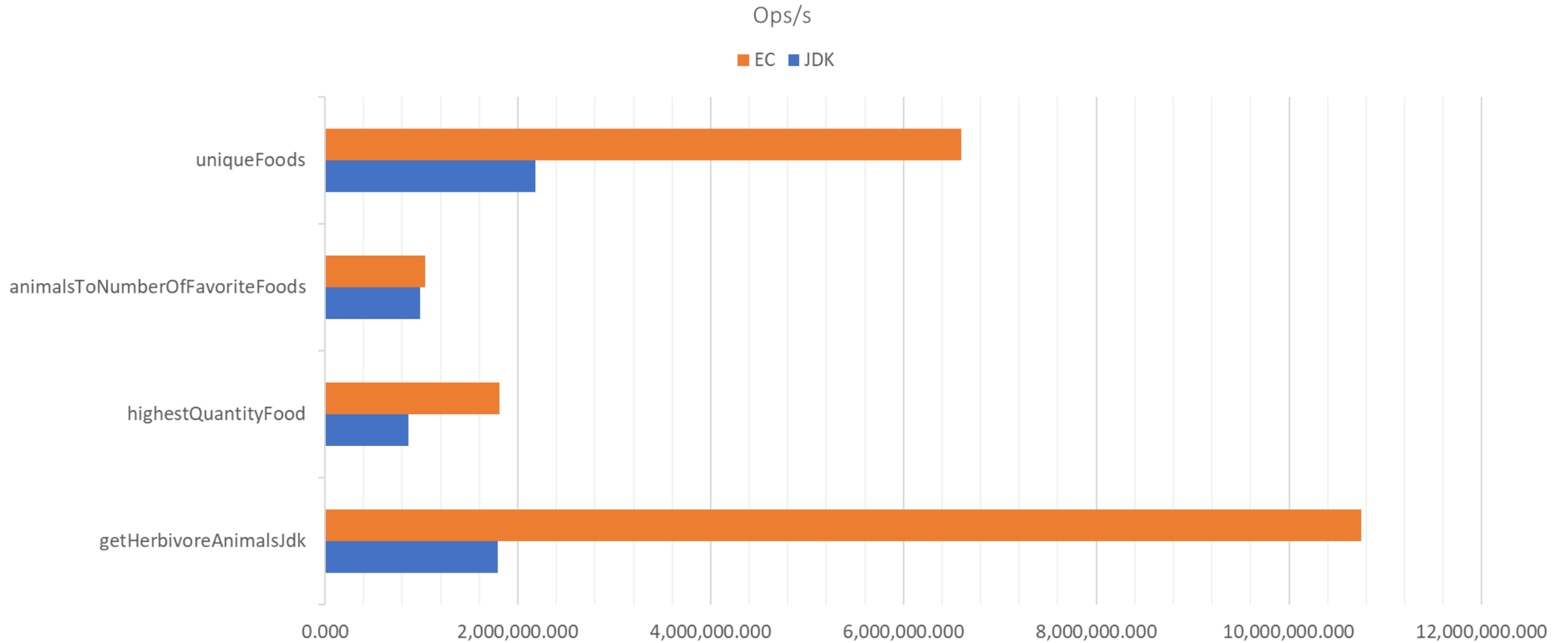
# Memory Usage (Overhead in KB, Count)



List of ints vs. List<Integer>
— EC IntArrayList — JDK ArrayList

Map<String, String>
— EC UnifiedMap — JDK HashMap

Bag<Strings> vs. Map<String, Integer>
— EC HashBag — JDK HashMap

Sets<String>
— EC UnifiedSet — JDK HashSet