CACHE ME IF YOU CAN

Speed Up Your JVM With Project Valhalla

Theresa Mammarella
𝕏 @t_mammarella

Theresa Mammarella

- Software Engineer @ IBM
- Eclipse OpenJ9 JVM
- Conference speaker
- Toronto JUG Co-Organizer

@t_mammarella

tmammarella

theresa-m

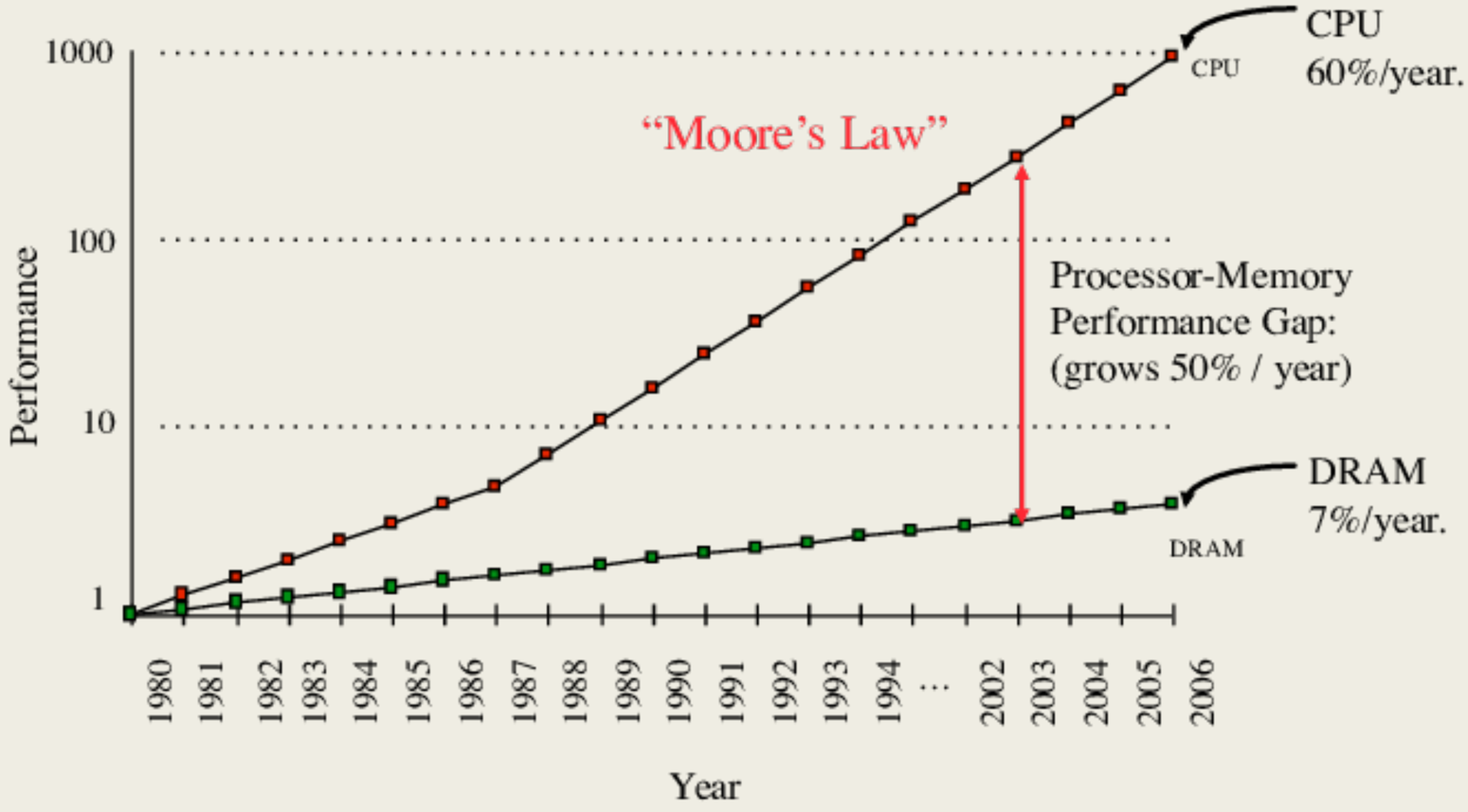
Outline

What problem does Project Valhalla solve?

Value Objects and Classes

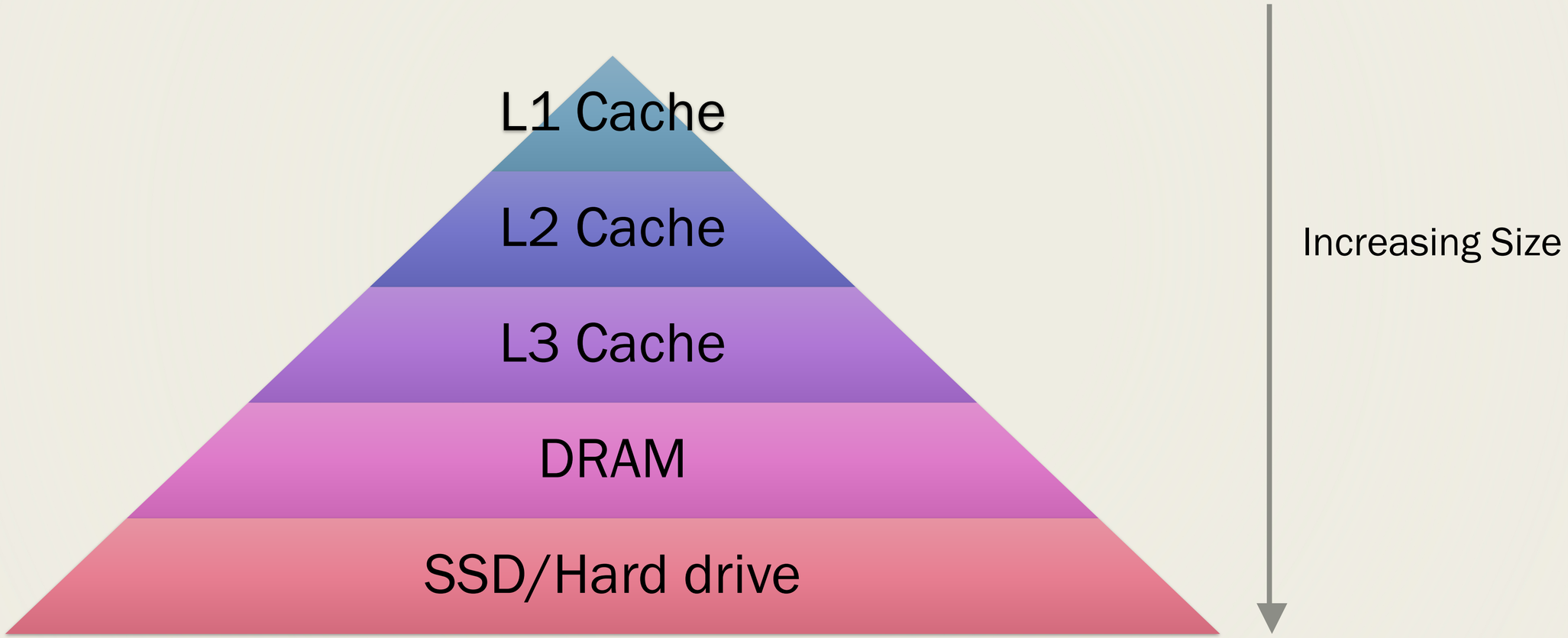
What's next for Project Valhalla?

Performance Gap



en.algorithmica.org/hpc/external-memory/

Memory Hierarchy



Cache Latency

Memory (smallest to largest)	Latency	Bandwidth
L1 cache	1 nanosecond	1 TB/second
L2 cache	4 nanoseconds	1 TB/second
L3 cache	10x slower than L2	> 400 GB/second
DRAM	2x slower than L3	100 GB/second
Hard drive I/O	100x - 1000x slower than DRAM	25 GB/second

Based on intel.com/content/www/us/en/developer/articles/technical/memory-performance-in-a-nutshell.html

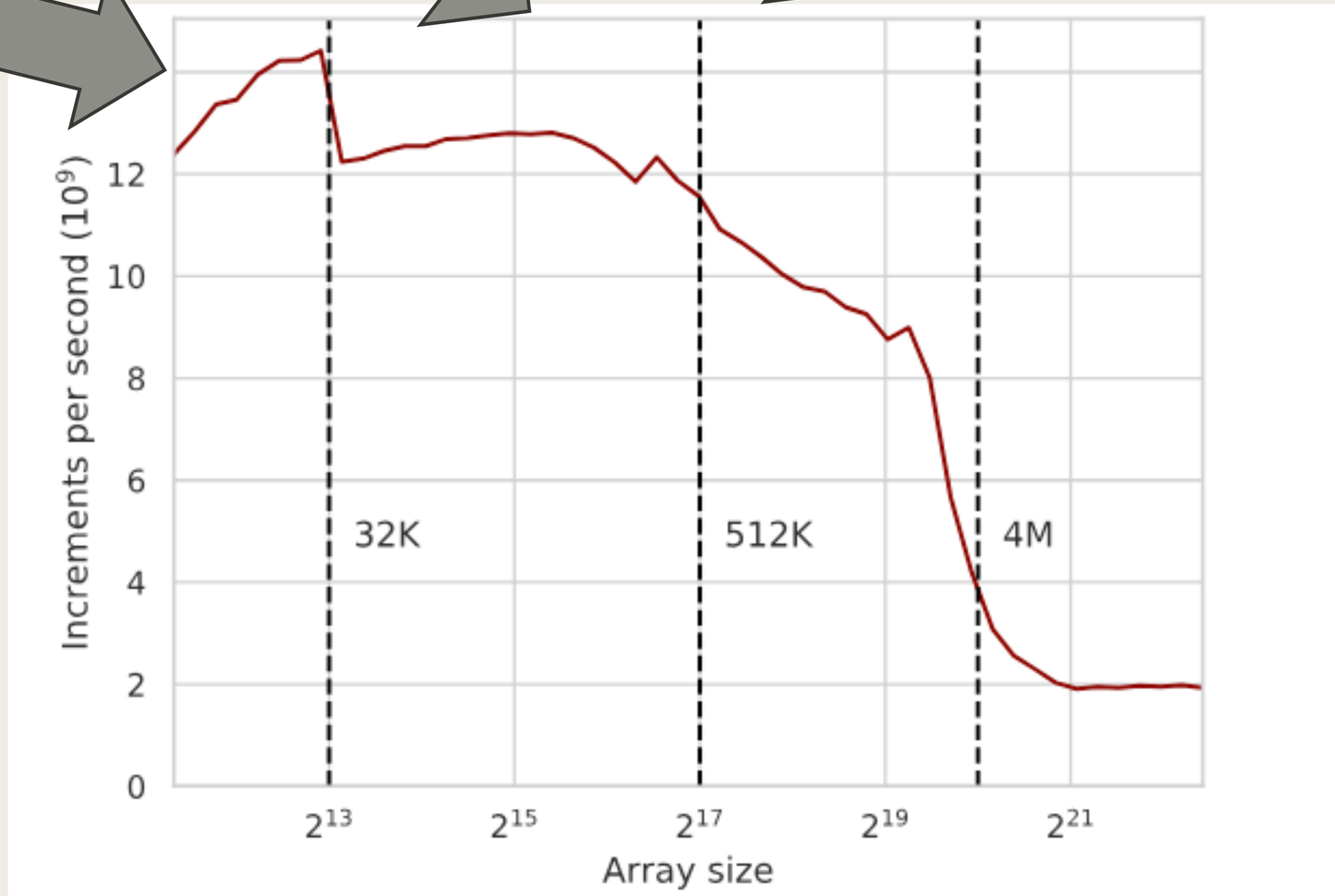
Performance Sabotage

Limited by L1

Limited by L2

Limited by CPU

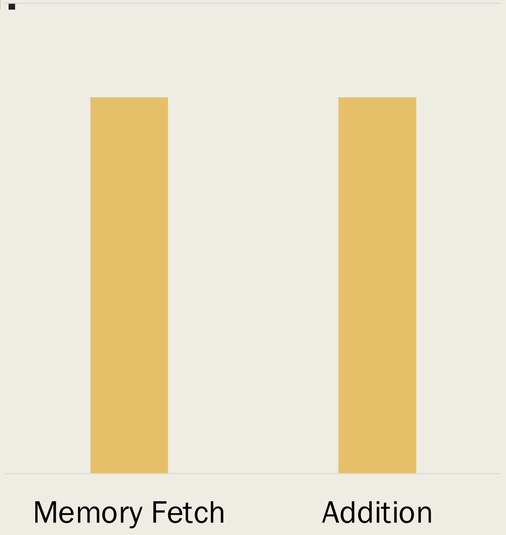
```
int a[N];

for (int t = 0; t < K; t++)
    for (int i = 0; i < N; i++)
        a[i]++;
```



en.algorithmica.org/hpc/cpu-cache/bandwidth/

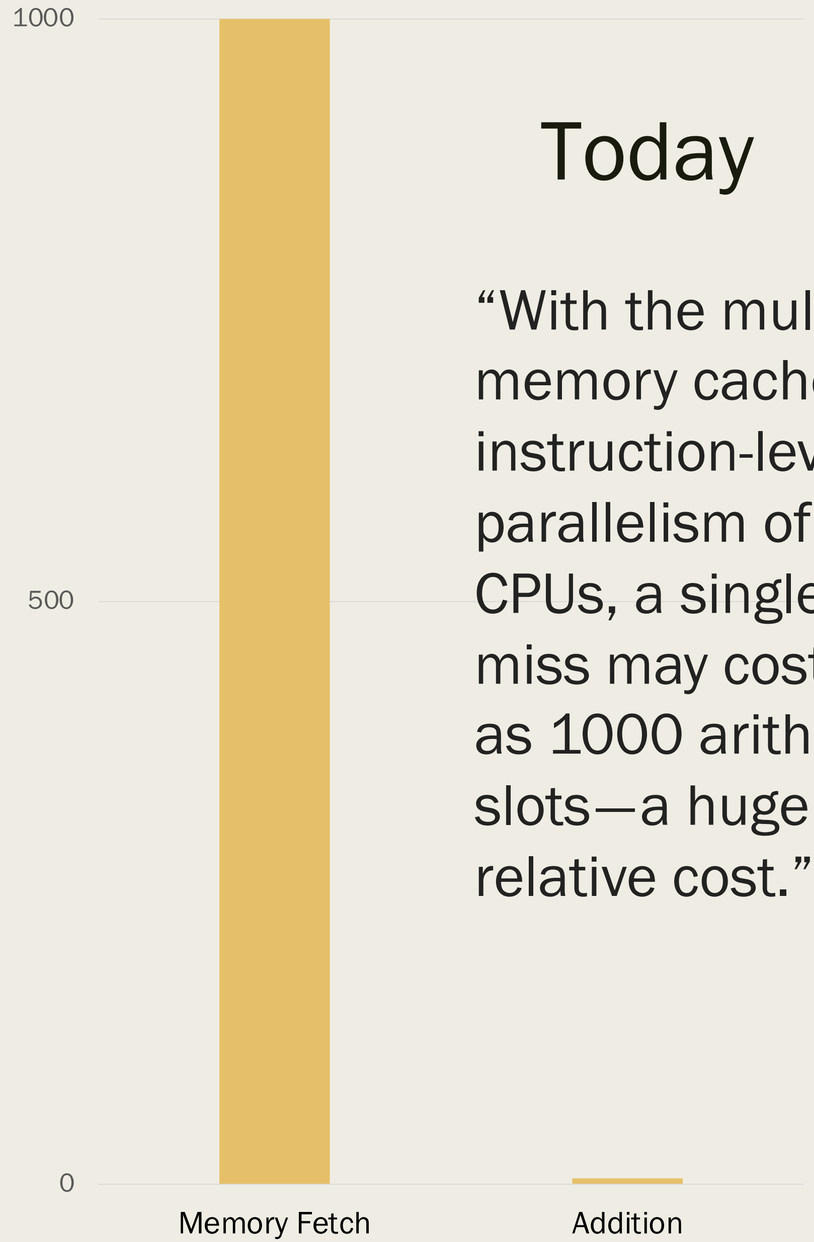
Early Days of Java

“When the Java Virtual Machine was being designed in the early 1990s, the cost of a memory fetch was comparable in magnitude to computational operations such as addition. “



Today

“With the multi-level memory caches and instruction-level parallelism of today’s CPUs, a single cache miss may cost as much as 1000 arithmetic issue slots—a huge increase in relative cost.”



openjdk.org/projects/valhalla/design-notes/state-of-valhalla/01-background

Object Representation in the JDK

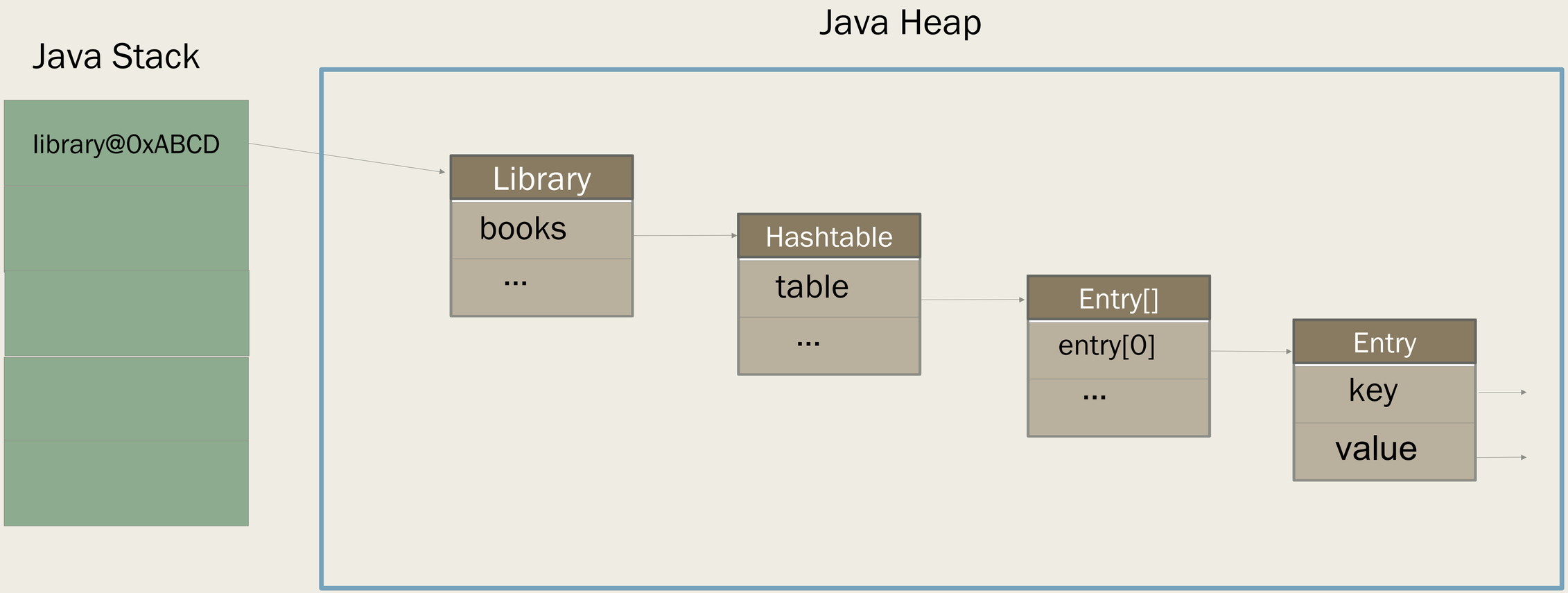
JVM

- Object references and relationships
- Stack model (program counting)
- No knowledge of memory allocation

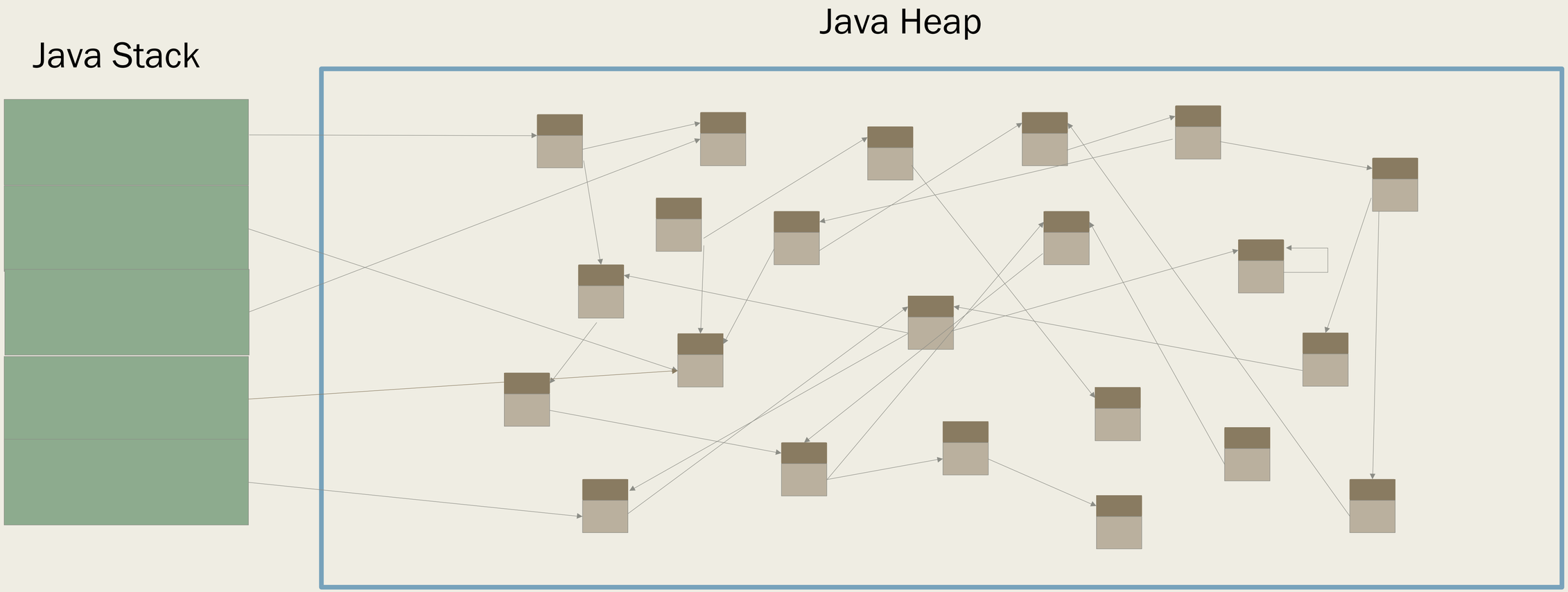
Garbage Collector

- Handles heap memory allocation
- (Mostly) no knowledge of object relationships

Object References in the JVM



LOTS of Object References

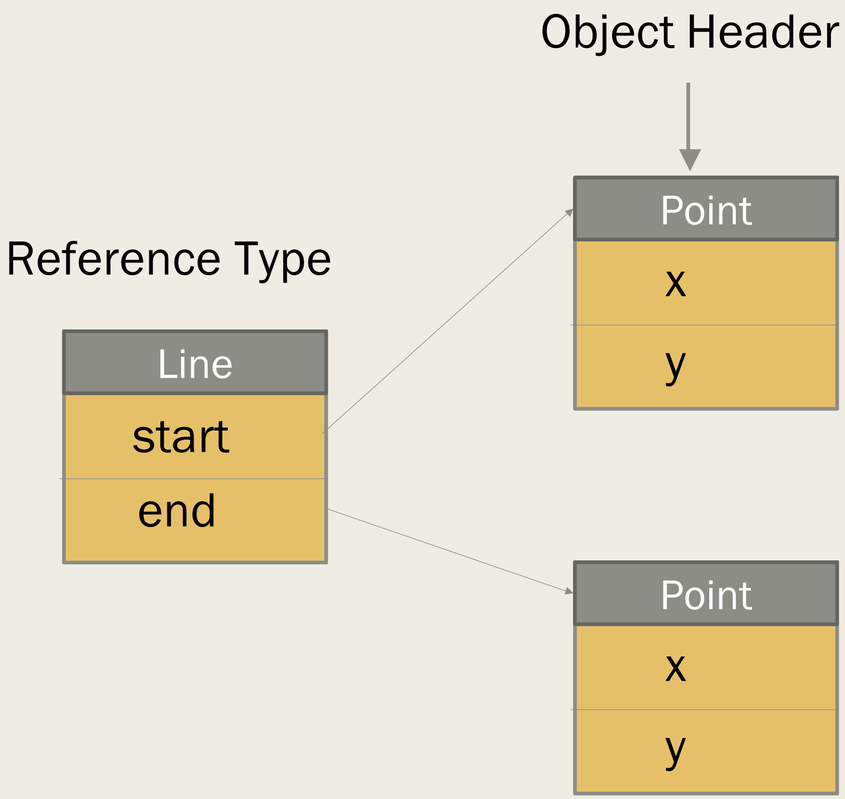


Identity

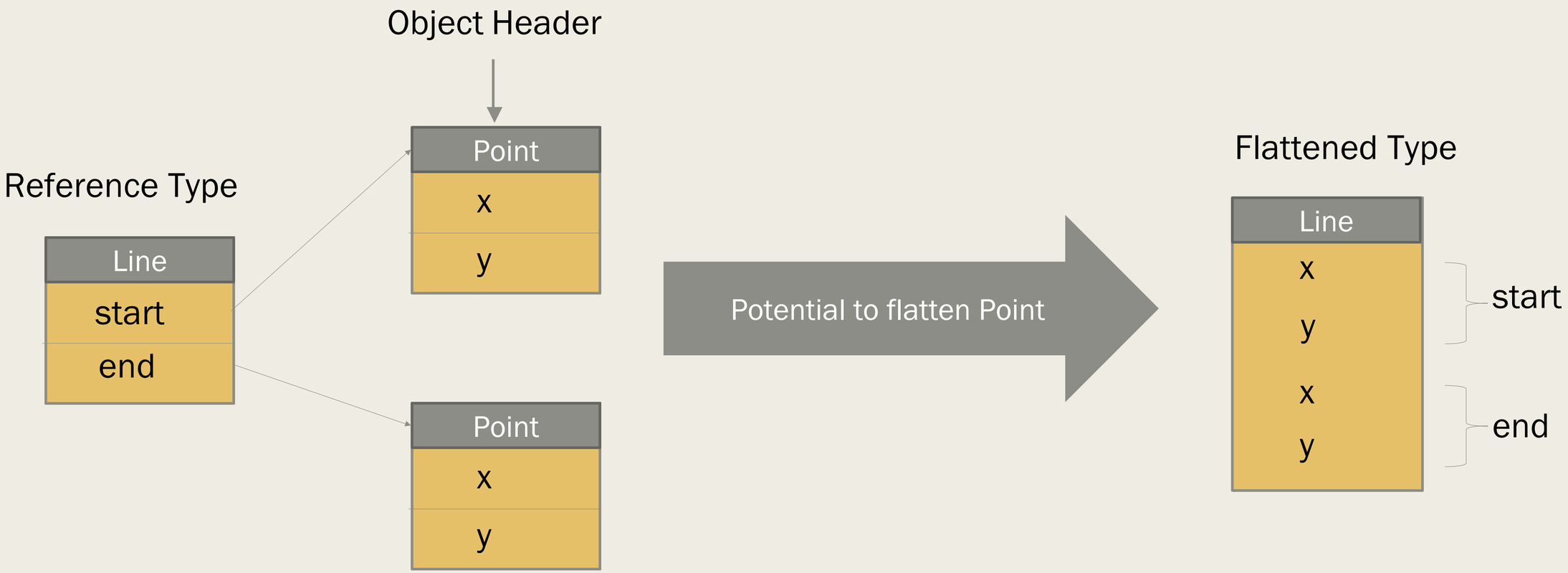
- Java objects are unique
- Think of == vs Object.equals()
- Allows for field mutability and synchronization
- In a Valhalla world these are known as “Identity Class” or “Identity Object”

DO WE ALWAYS NEED IDENTITY?

Eliminating References



Eliminating References



Project Valhalla

- Augment the Java object model with value objects
- Combine abstractions of OOP with the performance of primitives
  - “Codes like a class, works like an int”
- Heal the rift between primitives and objects

openjdk.org/projects/valhalla

JEP 401: Value Classes and Objects

01

Add a programming model option in which objects are distinguished solely by their field values.

02

Migrate popular classes that represent simple values in the JDK to this programming model.

03

Maximize the freedom of the JVM to encode simple values in ways that improve memory footprint, locality, and garbage collection efficiency.

openjdk.org/jeps/401

Value Class

- A class without identity
- Nullable
- Atomic access by default

Class and instance fields are are implicitly final

Synchronization on value objects is not allowed

== compares fields and will have the same behavior as Object.equals when not overridden

openjdk.org/jeps/401

Value Class

- A class without identity
- Nullable
- Atomic by default

```
value class Line2D {
    Point2D st;
    Point2D en;
    ...
}

value class Point2D {
    int x;
    int y;
    ...
}
```

openjdk.org/jeps/401

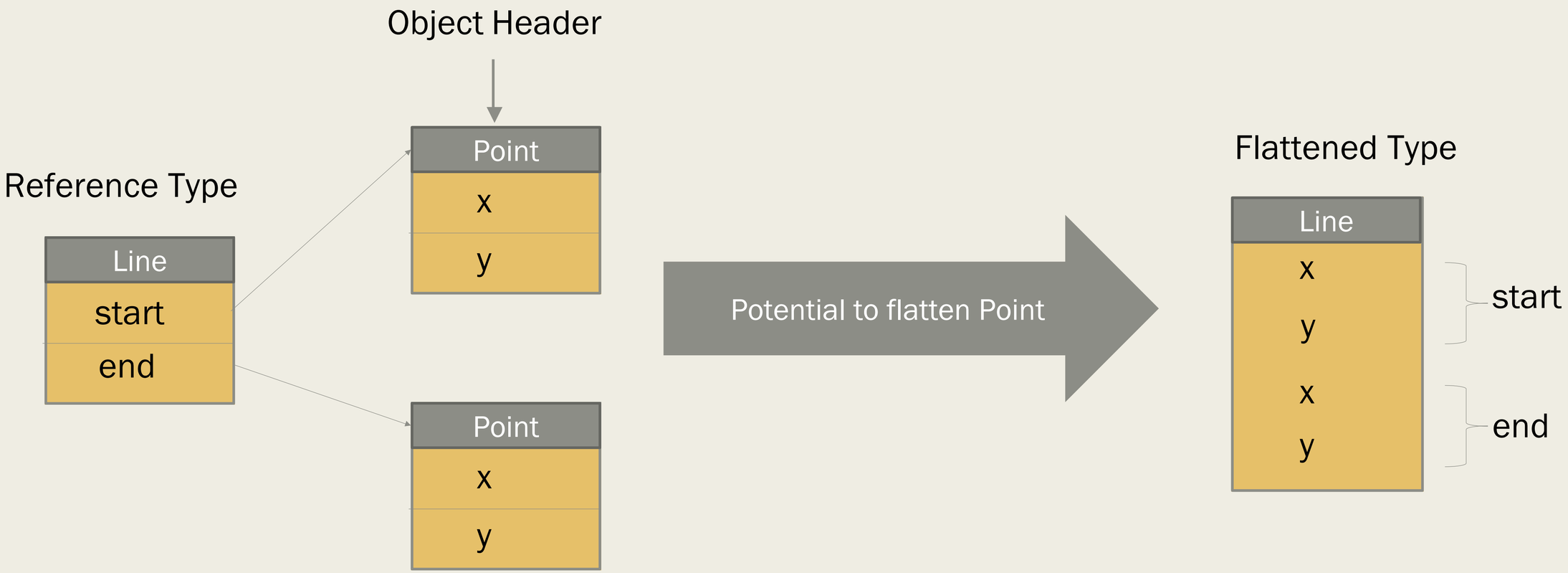
Class Preloading

- The JVM takes a more energetic approach to class loading for value classes
- Speculatively load classes to have complete layout information by the time layout decisions are made
- Timing is JVM dependent

```
class MyClass {
    VT vt;
    VT2 vt2;
}

value class VT {}
value class VT2 {}
```

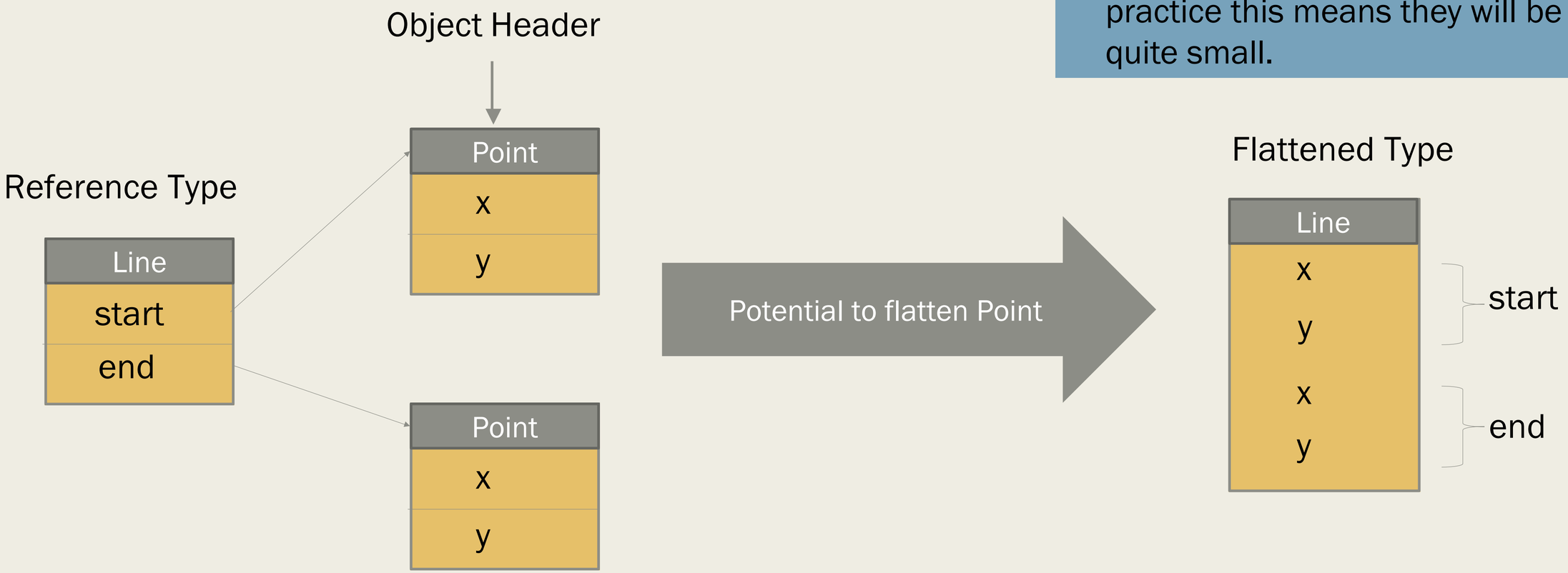
```
Classes to be preloaded:
  #18;                                          // value class VT
  #20;                                          // value class VT2
```

Flattening Value Objects

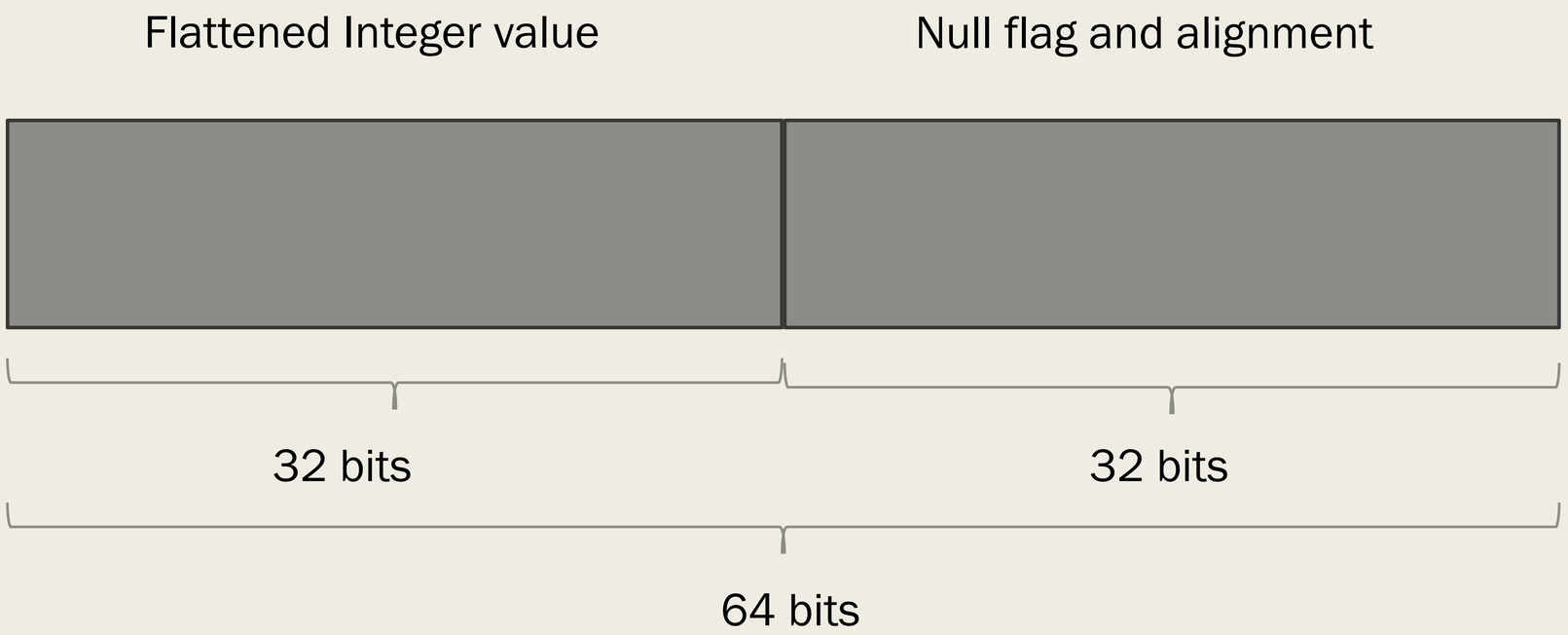


Flattening Value Objects

- Flattening is not guaranteed, especially for types > 32 bits
- Objects must be accessed atomically to be flattened… in practice this means they will be quite small.



Nullability is Expensive



Null-Restricted Value Class

- Value classes with fields that cannot be null
- Opt-in to automatic creation of default field values
- Allows for flattening of larger value classes

```
value class Line2D {
    Point2D! st;
    Point2D! en;
    ...
}

value class Point2D {
    int x;
    int y;

    public implicit Point2D();
}
```

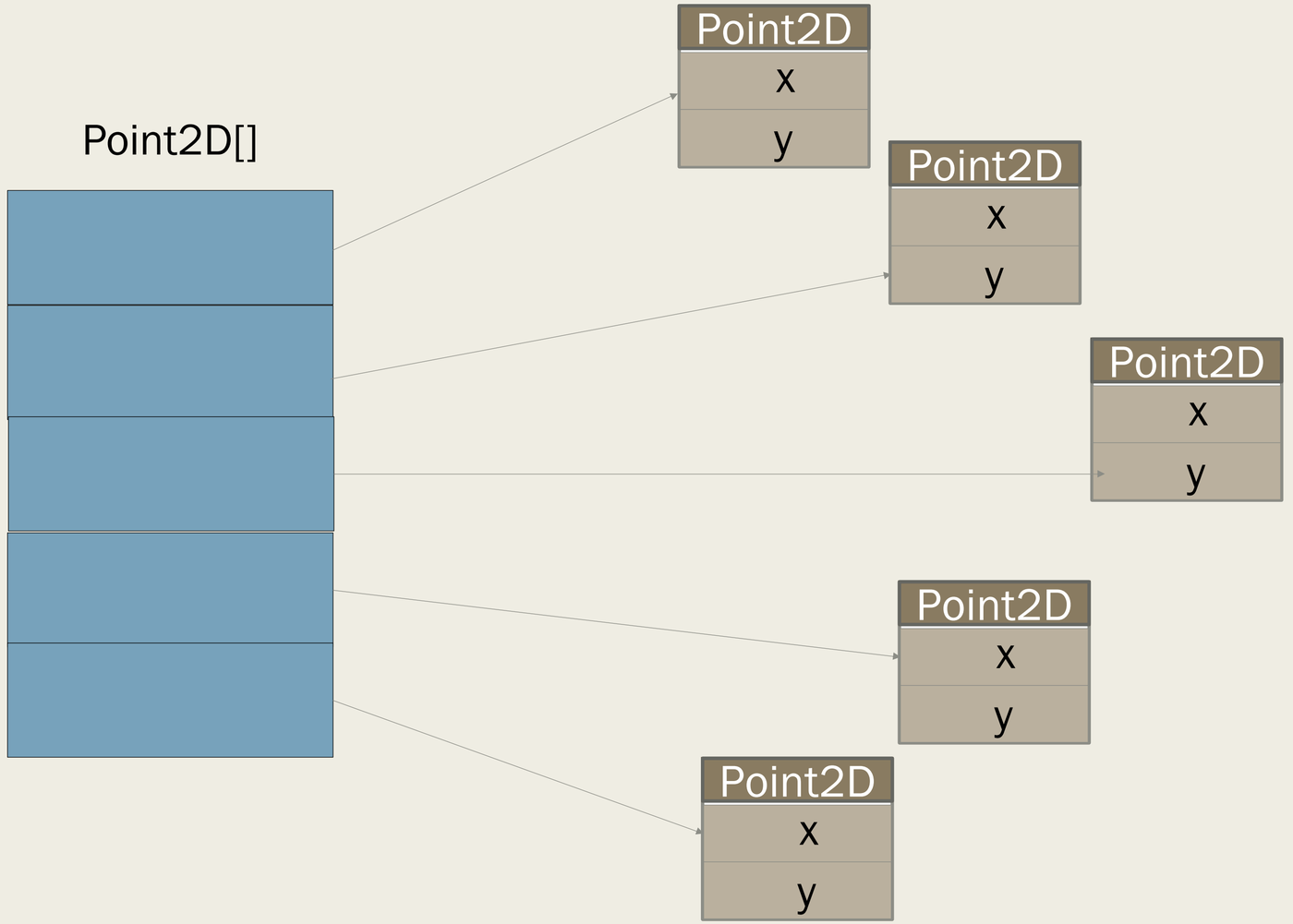
openjdk.org/jeps/8316779

Even Bigger Flattened Classes

- By default flattened value types need to be small enough to read and write atomically
- Non-volatile primitives (long and double) do not have these restrictions in Java
- Developers can opt-in to to remove this rule for programs that can tolerate non-atomic object access

```
value class Line2D {
    Point2D! st;
    Point2D! en;
    ...
}

value class Point2D
  implements LooselyConsistentValue
{
    int x;
    int y;

    public implicit Point2D();
}
```

Flattening Arrays



Arrays with Null-Restriction

```
Point2D![] array = new Point2D![8];

Point2D[]! array = new Point2D[8]!;
```

```
value class Point2D {
    int x;
    int y;

    public implicit Point2D();
}
```

IMPACT OF VALUE CLASSES

Value Classes and Records

- Records are available in Java 17+
- Generates data carrier methods that can cause errors if forgotten
- Record classes are final with final instance fields

```
value record Line2D(Point2D st, Point2D en) {
}
```

```
Compiled from "Line2D.java"
final value class Line2D extends java.lang.Record {
  private final Point2D st;
  private final Point2D en;
  public final java.lang.String toString();
  public final int hashCode();
  public final boolean equals(java.lang.Object);
  public Point2D st();
  public Point2D en();
  static Line2D Line2D(Point2D, Point2D);
}
```

Migrating Value Classes

- It may make sense to migrate data carrier classes to value classes
- In JEP 401 some core libraries will be considered to be value classes:
  - *java.lang.Byte*
  - *java.lang.Short*
  - *java.lang.Integer*
  - *java.lang.Long*
  - *java.lang.Float*
  - *java.lang.Double*
  - *java.lang.Boolean*
  - *java.lang.Character*
  - *java.util.Optional*
  - *java.lang.Number*
  - *java.lang.Record*

Java 8: Value-Based Classes

- Class declares only final instance fields
- equals/hashcode/toString determined from values, not identity
- Methods treat objects with equal values as substitutable
- The class performs no synchronization using an instance’s monitor’

Escape Analysis: More Opportunities for JIT

```
class Main {
    public static void main(String[] args) {
        example();
    }
    public static void example() {
        Foo foo = new Foo(); //alloc
        Bar bar = new Bar(); //alloc
        bar.setFoo(foo);
    }
}

class Foo {}

class Bar {
    private Foo foo;
    public void setFoo(Foo foo) {
        this.foo = foo;
    }
}
```

Example from: en.wikipedia.org/wiki/Escape_analysis

Get Involved

> We welcome input from interested Java developers. Keep in mind that most theoretical ideas have been well explored over the last few years! The greatest help can be provided by those who try out concrete prototypes and can share their experiences with real-world code bases.

OpenJDK early access builds: https://jdk.java.net/valhalla/

Build OpenJ9 With Value Types Enabled

git clone https://github.com/ibmruntimes/openj9-openjdk-jdk.valuetypes.git

cd openj9-openjdk-jdk.valuetypes

bash ./get_source.sh -openj9-repo=<url> -openj9-branch=<name>

bash ./configure --with-boot-jdk=<jdkpath> --enable-inline-types

make images

OpenJ9 Build Instructions: github.com/eclipse-openj9/openj9/tree/master/doc/build-instructions

Eventually it will be available as part of IBM Semeru Runtimes: ibm.com/semeru-runtimes

WHAT'S NEXT FOR PROJECT VALHALLA?

1. *Value Objects*, introducing class instances that lack identity
2. *Flattened Heap Layouts for Value Objects*, supporting null-free flattened storage
3. *Enhanced Primitive Boxing*, allowing primitives to be treated more like objects
4. *Null-Restricted and Nullable Types*, providing language support for managing nulls
5. *Parametric JVM*, preserving and optimizing generic class and method parameterizations at runtime

Thank you ConFoo!

@t_mammarella

tmammarella

theresa-m

ConFoo 2024 Slides