ORACLE

Memory API: Patterns, Uses Cases, and Performance

From the Panama Foreign Functions and Memory API

José Paumard
Java Developer Advocate
Java Platform Group

Rémi Forax
Maître de conferences
Université Gustave Eiffel

https://twitter.com/Nope!

https://github.com/forax

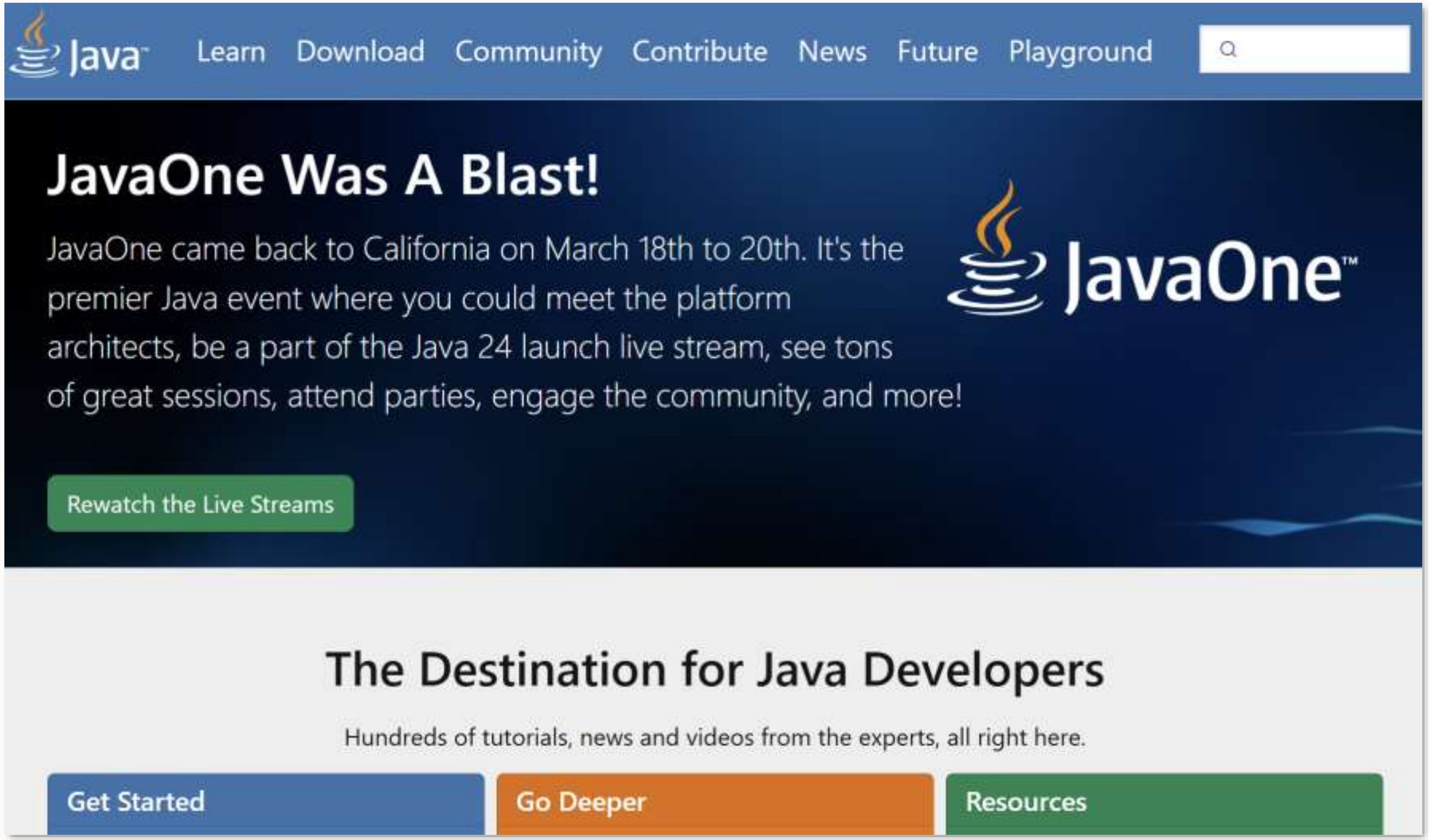
https://speakerdeck.com/forax

OpenJDK, ASM, Tatoo, Pro, etc…
One of the Father of invokedynamic (Java 7)
Lambda (Java 8), Module (Java 9)
Constant dynamic (Java 11)
Record, text blocks, sealed types (Java 14 / 15)
Valhalla (Java 25+)

@josepaumard.bsky.social

https://dev.java

https://dev.java/



JavaOne Was A Blast!

JavaOne came back to California on March 18th to 20th. It's the premier Java event where you could meet the platform architects, be a part of the Java 24 launch live stream, see tons of great sessions, attend parties, engage the community, and more!

Rewatch the Live Streams

The Destination for Java Developers

Hundreds of tutorials, news and videos from the experts, all right here.

Tune in!

Java's Plans for 2025 - Java Newscast #83

Java 24 Release Notes Review for Developers -...

Inside Java Newscast

o Java 21

Java 21 new feature: Virtual

Java 21 JVM & GC Improvements #RoadTo21

Java 21 Tool Enhancements: Better Across the Board...

Road To 21 series

Inside Java Podcast

Better Java Streams with Gatherers - JEP Cafe #23

JEP Café

Cracking the Java coding interview

Inside Java

News and views from members of the Java team at Oracle

Shows: Podcast | Newscast | JEP Café | Sip Of Java   dev.java | About | Jobs

Sort by: Date | Author | Tag

Java 20 Unboxing - Inside Java Newscast #44

Billy Korando on March 23, 2023

Java First. Java Always. | Level Up Keynote

Georges Saab on March 22, 2023

JDK 20 Security Enhancements

Sean Mullan on March 22, 2023

The Arrival of Java 20!

Sharat Chander on March 21, 2023

Inside.java

What's New in Java 23 in 2 Minutes... More or Less - Si...

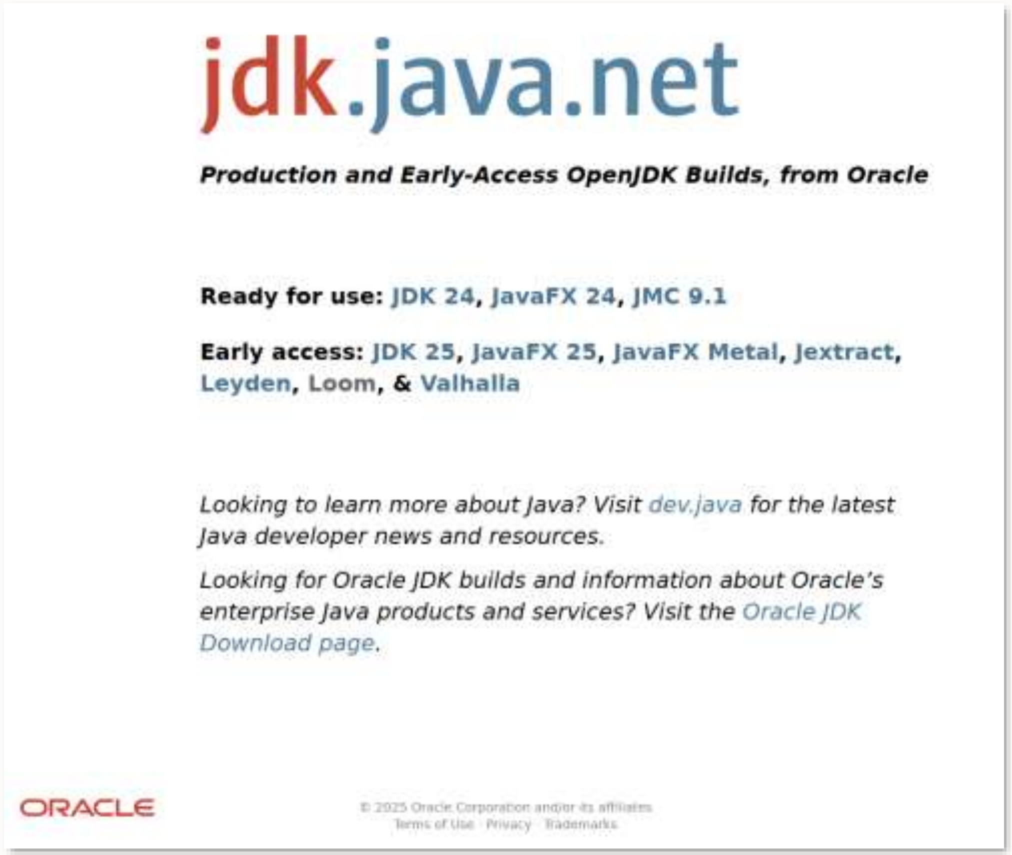
Sip of Java

OpenJDK is the place where it all happens

What is this? The place to collaborate on an open-source implementation of the Java Platform, Standard Edition, and related projects.

https://openjdk.org/

OpenJDK is the place where it all happens



jdk.java.net

Production and Early-Access OpenJDK Builds, from Oracle

Ready for use: JDK 24, JavaFX 24, JMC 9.1

Early access: JDK 25, JavaFX 25, JavaFX Metal, Jextract, Leyden, Loom, & Valhalla

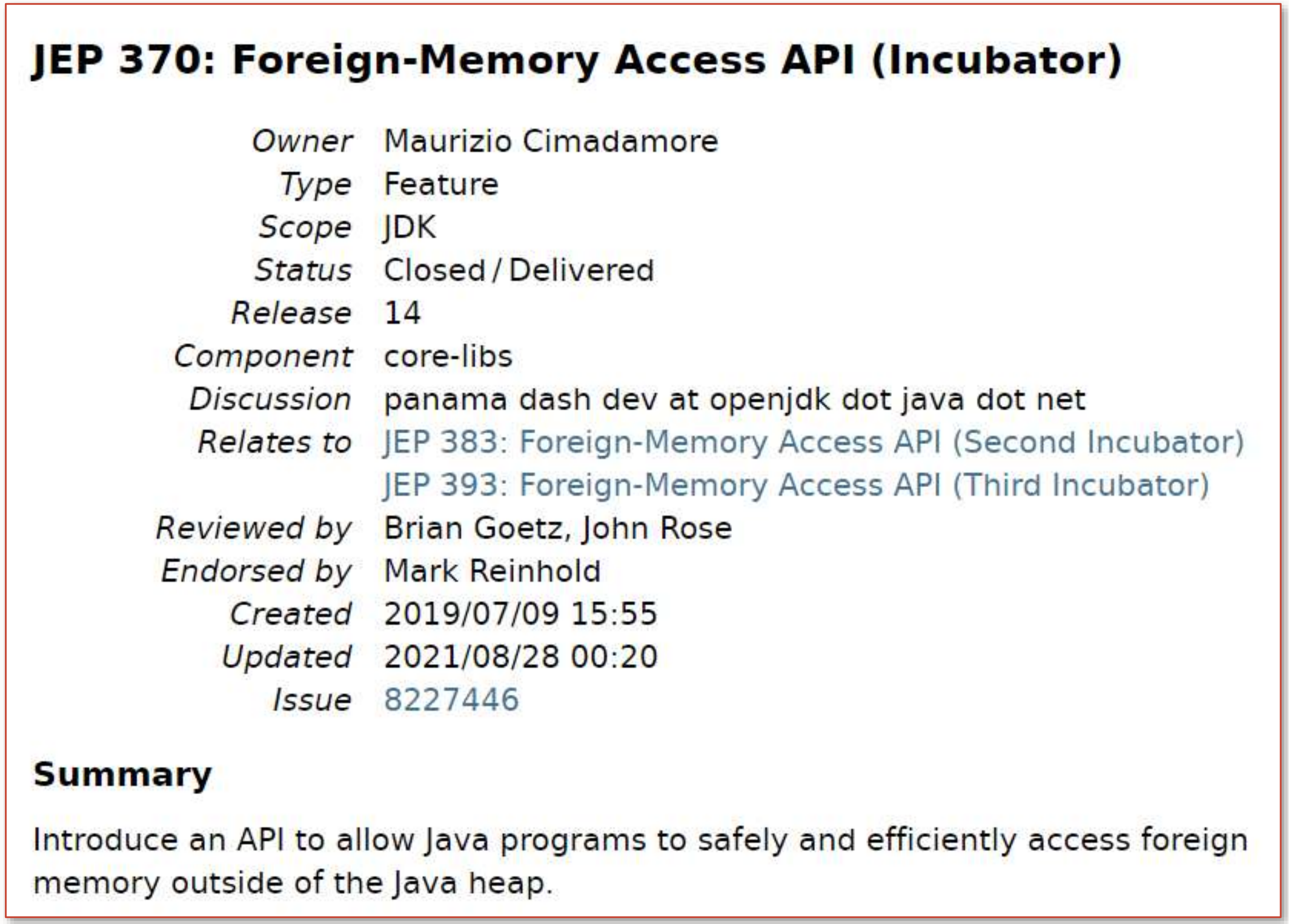
Looking to learn more about Java? Visit dev.java for the latest Java developer news and resources.

Looking for Oracle JDK builds and information about Oracle's enterprise Java products and services? Visit the Oracle JDK Download page.

https://jdk.java.net/

Panama

First Preview in the JDK 14



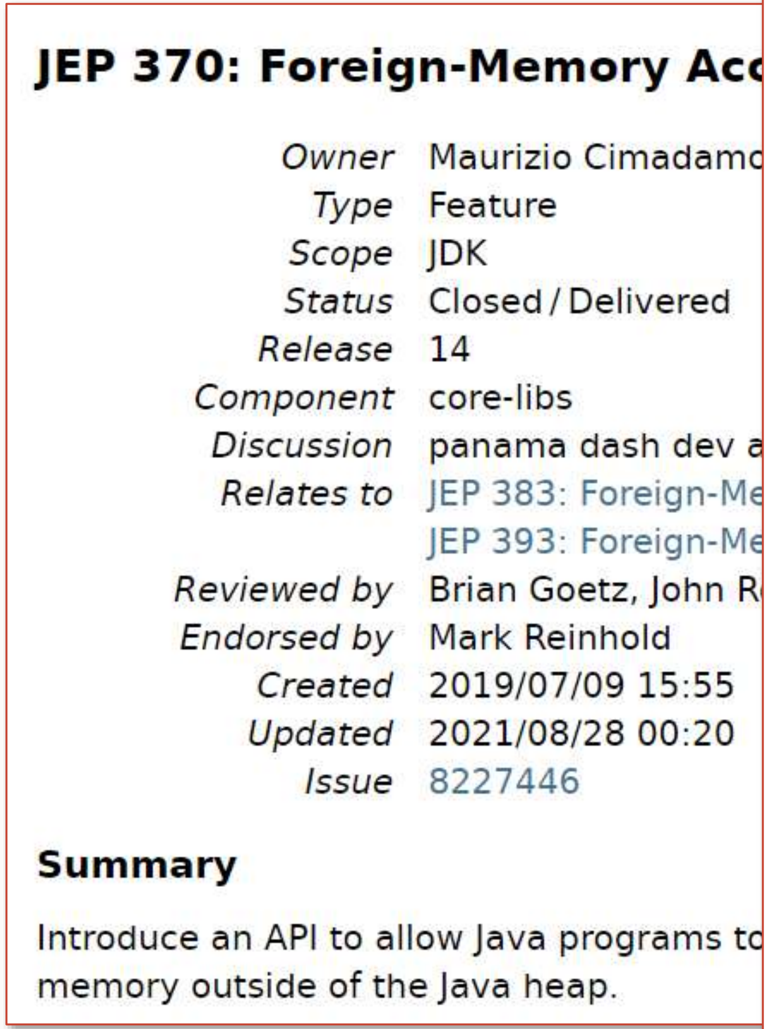
JEP 370: Foreign-Memory Access API (Incubator)

Owner	Maurizio Cimadamore
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	14
Component	core-libs
Discussion	panama dash dev at openjdk dot java dot net
Relates to	JEP 383: Foreign-Memory Access API (Second Incubator)
	JEP 393: Foreign-Memory Access API (Third Incubator)
Reviewed by	Brian Goetz, John Rose
Endorsed by	Mark Reinhold
Created	2019/07/09 15:55
Updated	2021/08/28 00:20
Issue	8227446

Summary

Introduce an API to allow Java programs to safely and efficiently access foreign memory outside of the Java heap.

Final Feature in the JDK 22

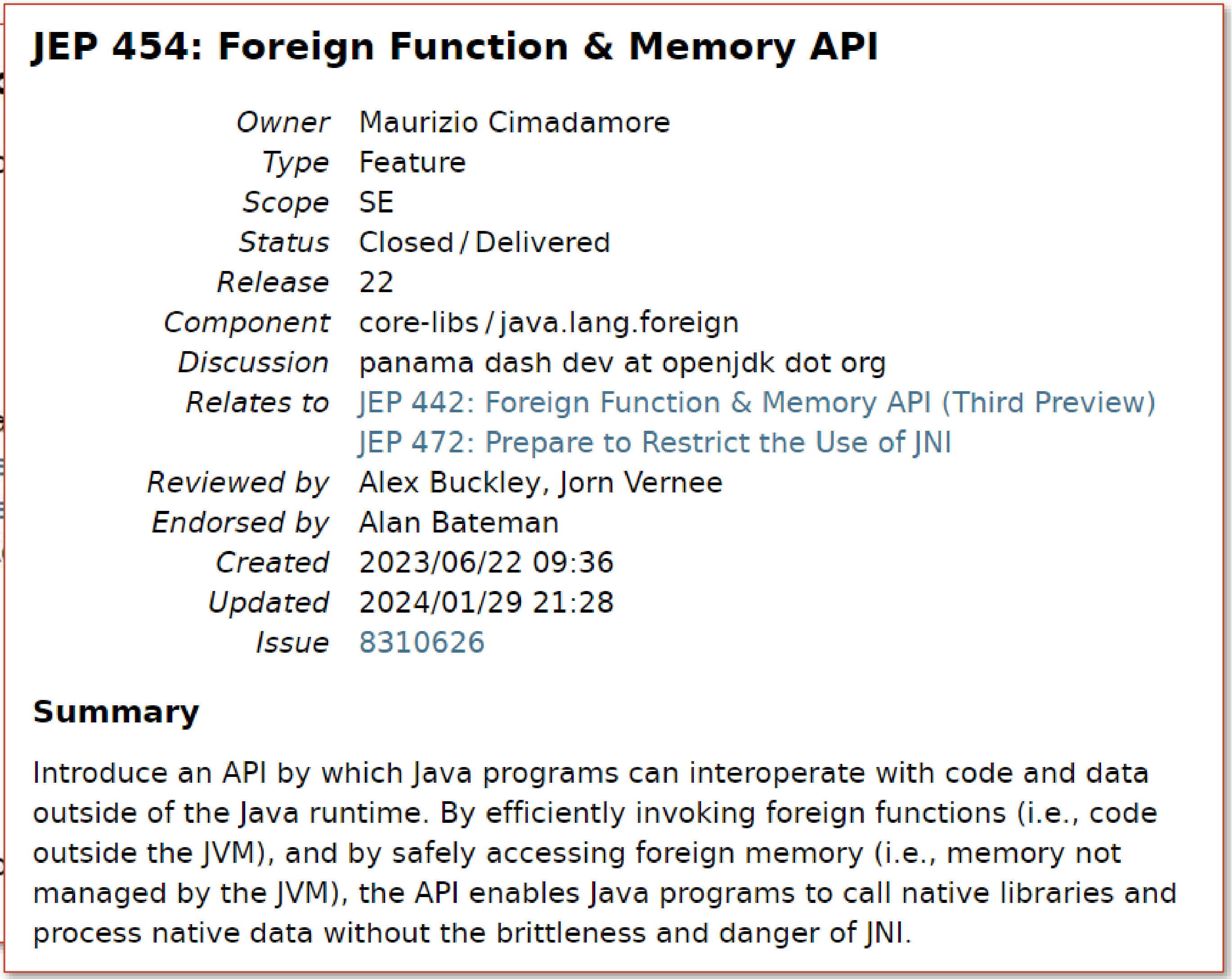


JEP 370: Foreign-Memory Acc

Owner	Maurizio Cimadamo
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	14
Component	core-libs
Discussion	panama dash dev a
Relates to	JEP 383: Foreign-Me JEP 393: Foreign-Me
Reviewed by	Brian Goetz, John R
Endorsed by	Mark Reinhold
Created	2019/07/09 15:55
Updated	2021/08/28 00:20
Issue	8227446

Summary

Introduce an API to allow Java programs to
memory outside of the Java heap.



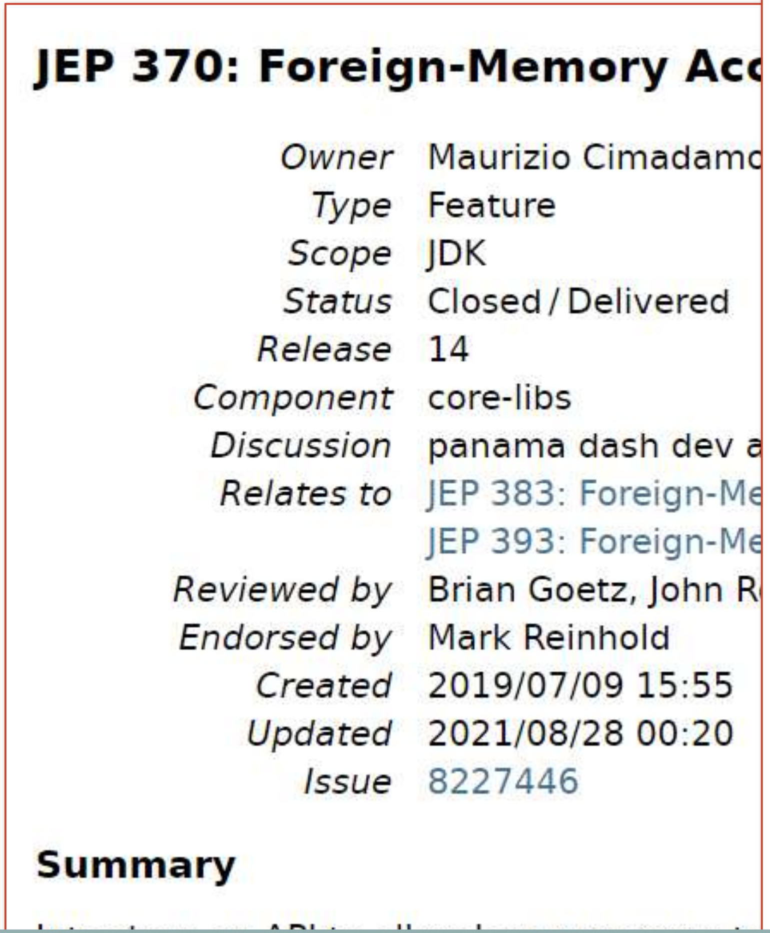
JEP 454: Foreign Function & Memory API

Owner	Maurizio Cimadamore
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	22
Component	core-libs / java.lang.foreign
Discussion	panama dash dev at openjdk dot org
Relates to	JEP 442: Foreign Function & Memory API (Third Preview) JEP 472: Prepare to Restrict the Use of JNI
Reviewed by	Alex Buckley, Jorn Vernee
Endorsed by	Alan Bateman
Created	2023/06/22 09:36
Updated	2024/01/29 21:28
Issue	8310626

Summary

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI.

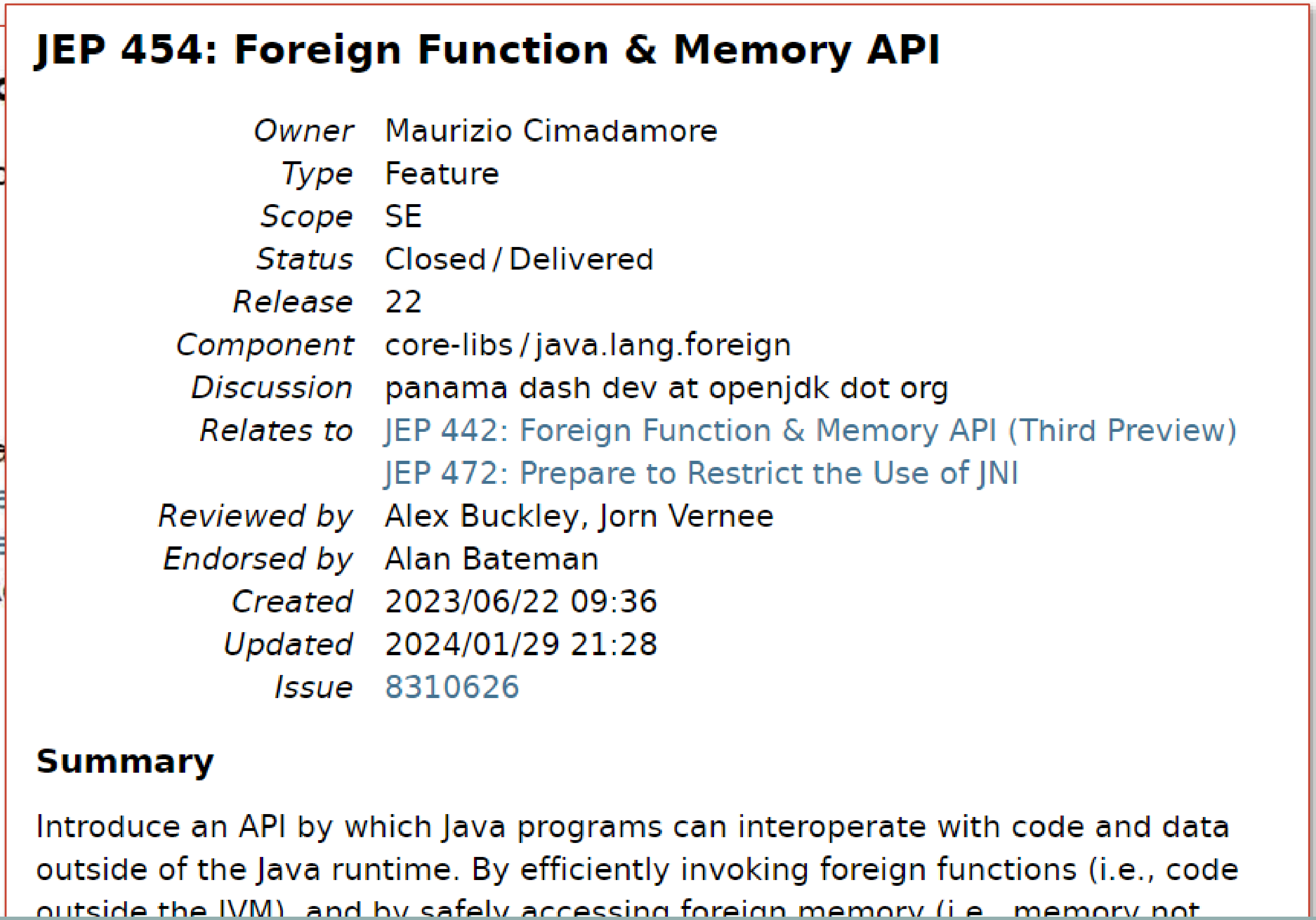
Final Feature in the JDK 22



JEP 370: Foreign-Memory Acc

Owner	Maurizio Cimadamo
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	14
Component	core-libs
Discussion	panama dash dev a
Relates to	JEP 383: Foreign-Me
	JEP 393: Foreign-Me
Reviewed by	Brian Goetz, John R
Endorsed by	Mark Reinhold
Created	2019/07/09 15:55
Updated	2021/08/28 00:20
Issue	8227446

Summary



JEP 454: Foreign Function & Memory API

Owner	Maurizio Cimadamore
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	22
Component	core-libs / java.lang.foreign
Discussion	panama dash dev at openjdk dot org
Relates to	JEP 442: Foreign Function & Memory API (Third Preview)
	JEP 472: Prepare to Restrict the Use of JNI
Reviewed by	Alex Buckley, Jorn Vernee
Endorsed by	Alan Bateman
Created	2023/06/22 09:36
Updated	2024/01/29 21:28
Issue	8310626

Summary

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not

https://openjdk.org/projects/panama/

Demo time!

What is Panama About?

Heal the rift between Java and C

Fixing issues in the Java NIO API
Namely, fix and update what you can do with ByteBuffer

ByteBuffer where released in Java 4, in 2002

What’s New in Java 4? (2002)

New assert keyword
Exception chaining
XML Parser
Java NIO! (New Input / Output, JSR 51)

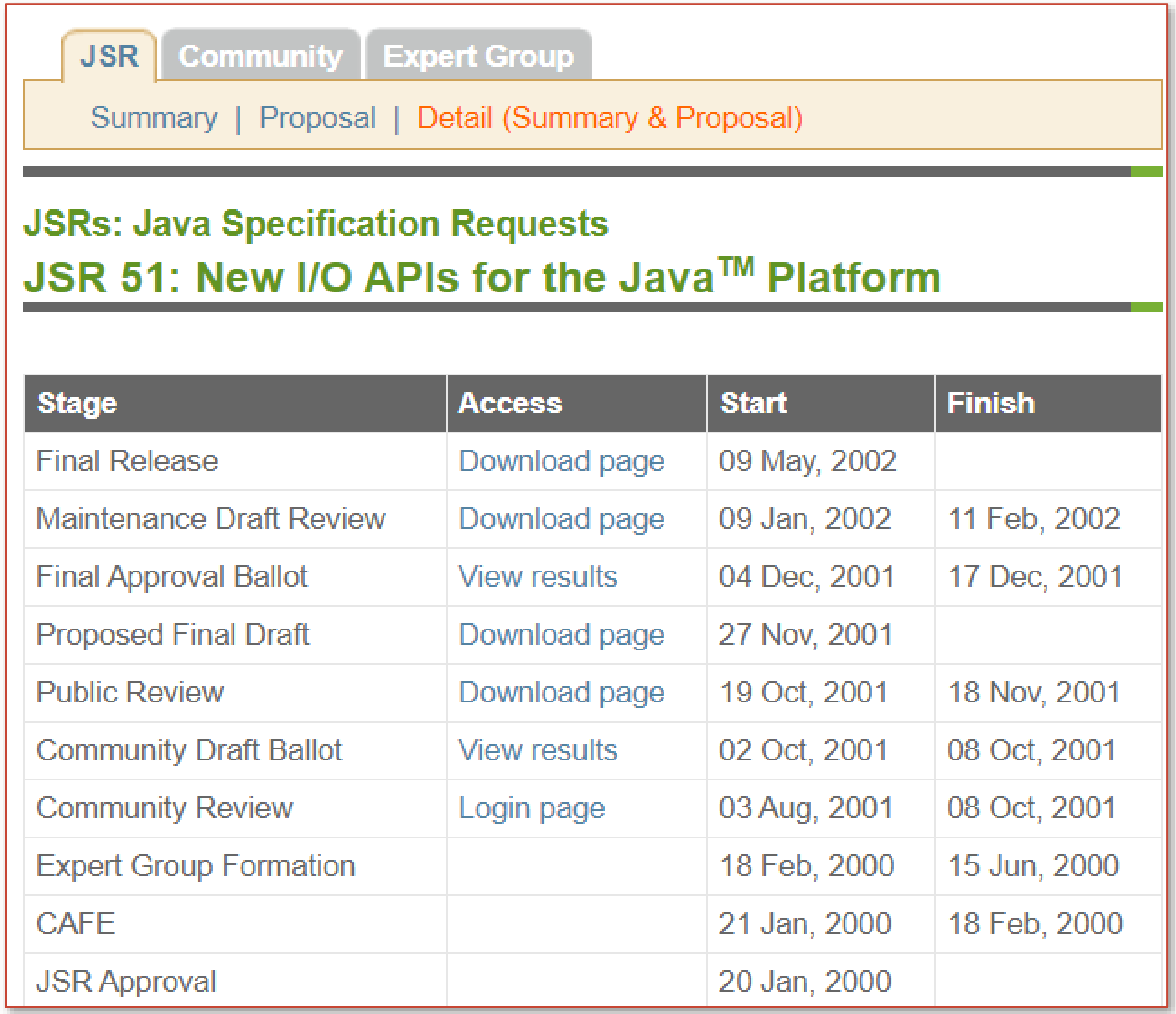
What's New in Java 4?

New assert keyword

Exception chaining

XML Parser

Java NIO



JSR | Community | Expert Group

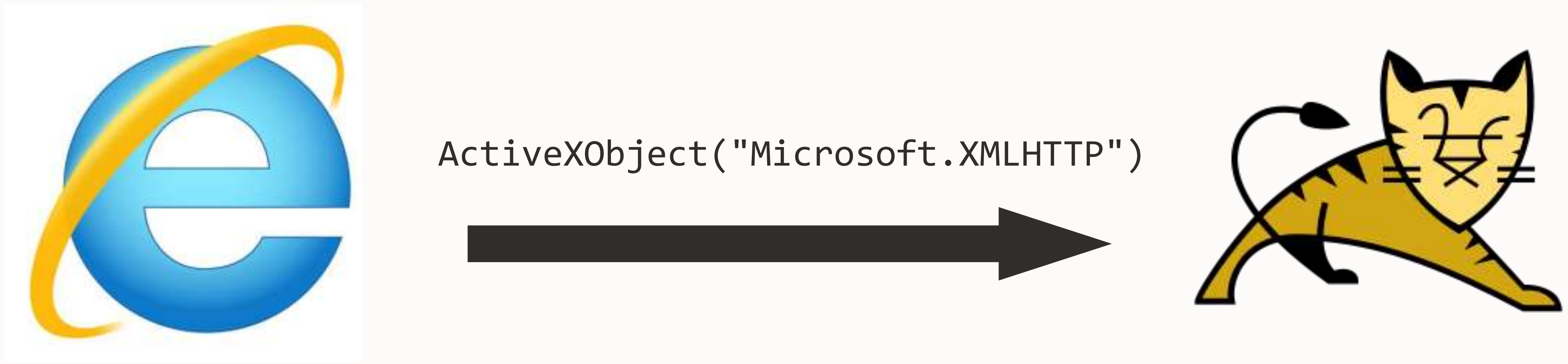
Summary | Proposal | Detail (Summary & Proposal)

JSRs: Java Specification Requests

JSR 51: New I/O APIs for the Java™ Platform

Stage	Access	Start	Finish
Final Release	Download page	09 May, 2002	
Maintenance Draft Review	Download page	09 Jan, 2002	11 Feb, 2002
Final Approval Ballot	View results	04 Dec, 2001	17 Dec, 2001
Proposed Final Draft	Download page	27 Nov, 2001	
Public Review	Download page	19 Oct, 2001	18 Nov, 2001
Community Draft Ballot	View results	02 Oct, 2001	08 Oct, 2001
Community Review	Login page	03 Aug, 2001	08 Oct, 2001
Expert Group Formation		18 Feb, 2000	15 Jun, 2000
CAFE		21 Jan, 2000	18 Feb, 2000
JSR Approval		20 Jan, 2000	

Dynamic Web Site in 2002

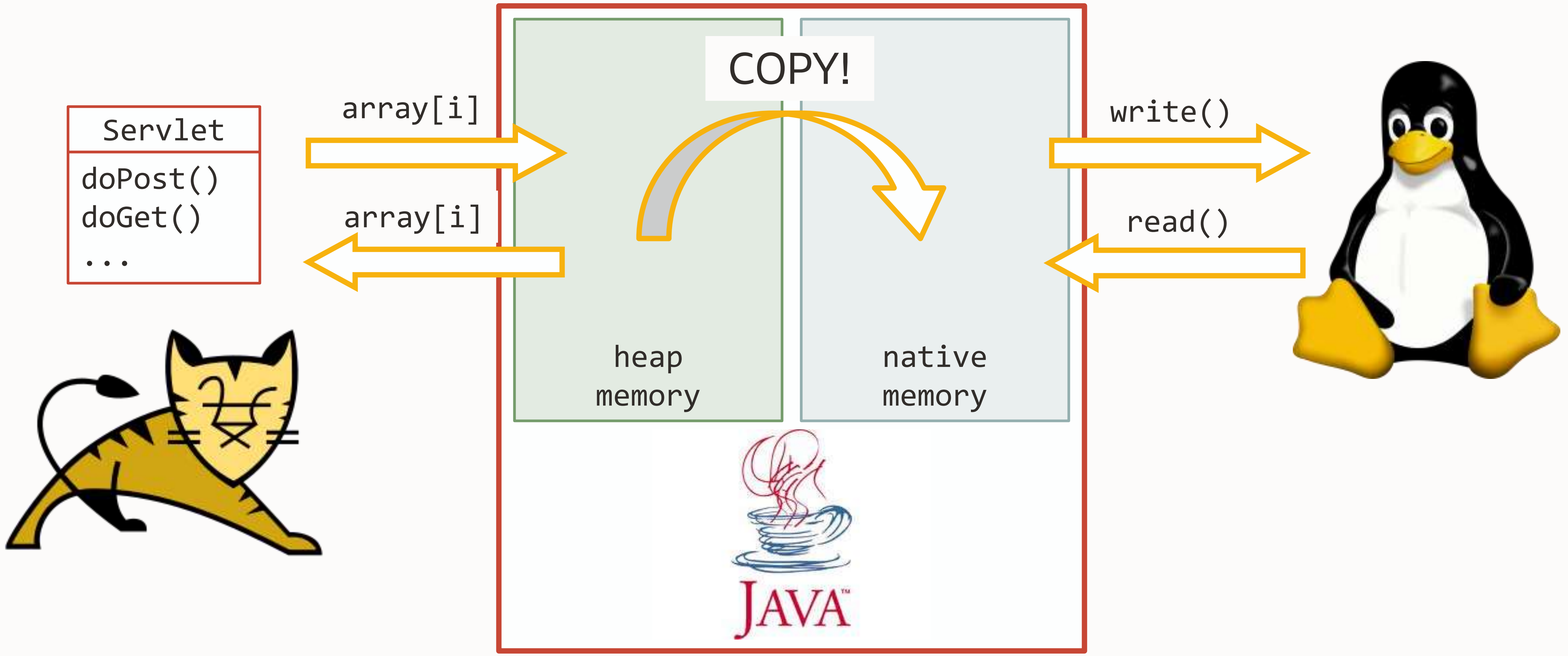


```
ActiveXObject("Microsoft.XMLHTTP")
```

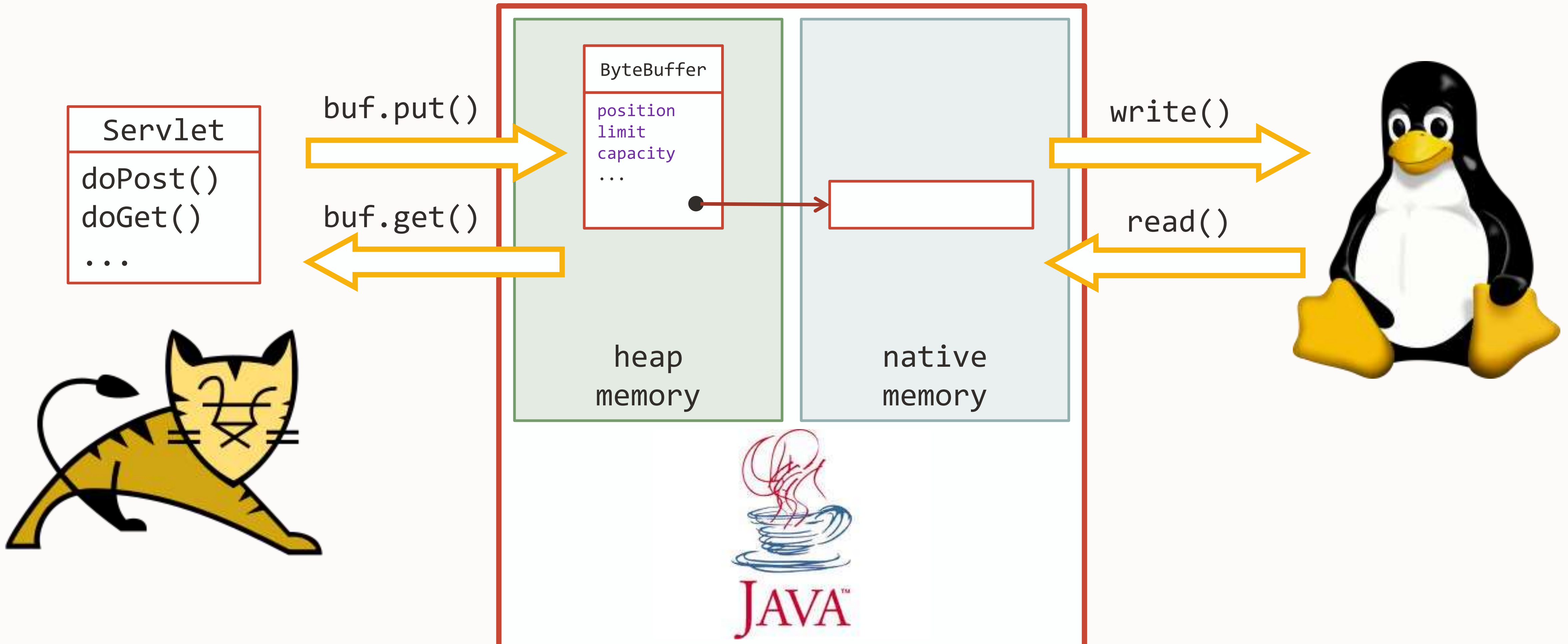
IE 5.5 SP2?
IE 6?

Tomcat 3
Maybe 4?

The World Before Java 4



The World with NIO



Creating a ByteBuffer

Off-heap allocation

```
var buffer = ByteBuffer.allocateDirect(1_024); // int
```

File mapping

```
var buffer = FileChannel.map(
                 READ_WRITE, position, size); // longs
```

Issues with ByteBuffers

Demo time!

Issues with the ByteBuffer API

Too high level for a Memory Access API

    position, capacity, reset are not needed

32 bits indexing only

Allow for unaligned access, but may be very slow

Non-deterministic deallocation!

    closing a mapped file does not close the ByteBuffer

How is Deallocation Working?

The GC

- selects a region containing the ByteBuffer
- then sees that the ByteBuffer is dead
- then a Cleaner code (weakref) is pushed to a Cleaner queue

Later, a cleaner thread dequeues the Cleaner code and calls free on the off-heap memory (or not…)

Welcome to Panama

New API: the MemorySegment API
- lower level than the ByteBuffer API
- ByteBuffer are now built on top of MemorySegment

Goals:
- fix ByteBuffer issues
- better interaction with C code

Introducing MemorySegment

MemorySegment

A MemorySegment:

- is safe (cannot be used once freed)
- gives you control over the allocation / deallocation
- brings close to C performance (and Unsafe)
- offers direct access, indexed 64 bits access, structured access
- opt-in unsafe access (for C interop, may crash later)
- retrofit ByteBuffer on top

What about sun.misc.Unsafe?

It is unsafe!

- Close to C performance
- No use after free protection (security)
- Can peek/poke everywhere (may crash later)
- No null check for on heap array access (may crash)

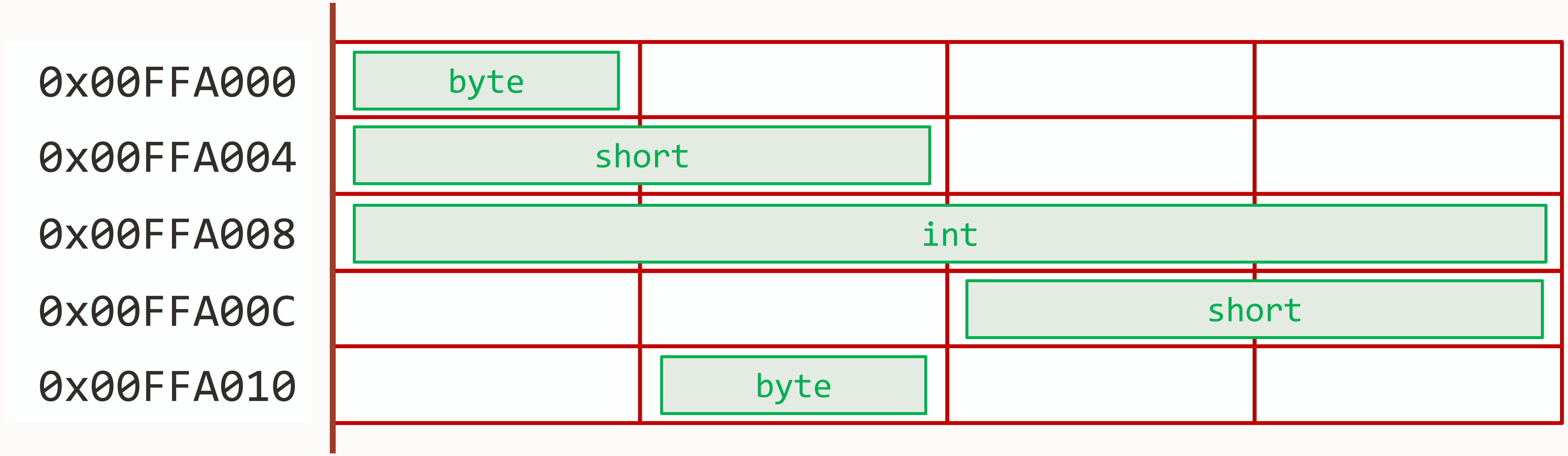
Memory access methods are

- deprecated for removal (JEP 471, del. JDK 23)
- warnings since 2006

Demo time!

Alignement

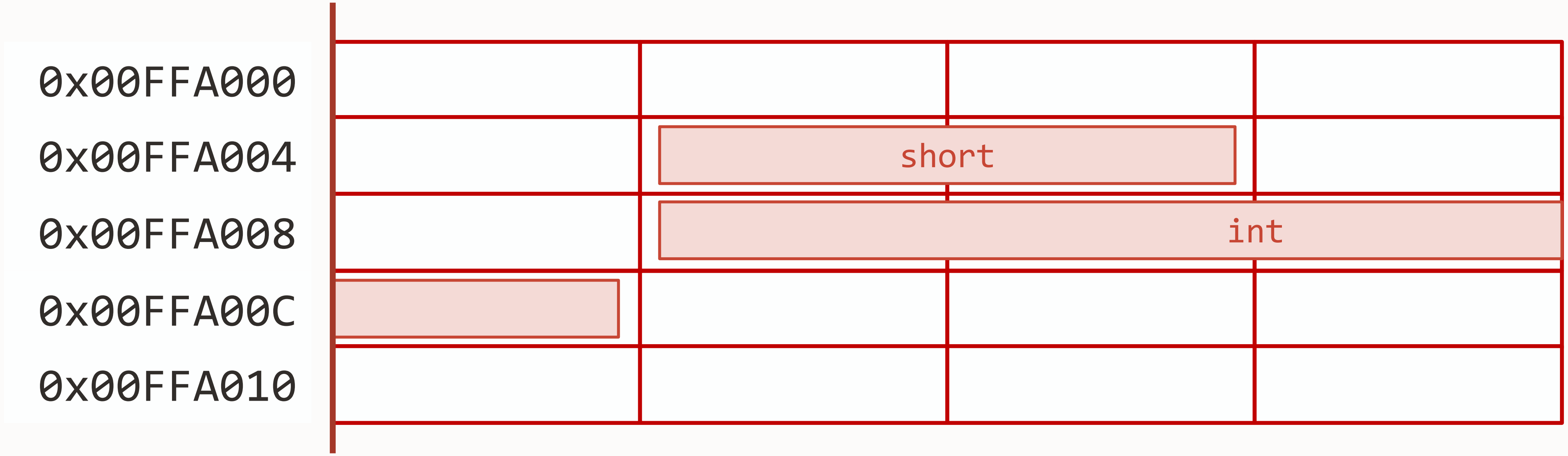
Most CPU require your data to be aligned in memory



These are properly aligned

Alignement

Most CPU require your data to be aligned in memory



These are misaligned

The MemorySegment API

Off-heap allocation, direct access

```
var arena = Arena.global();
var segment = arena.allocation(1_024L); // long, off-heap

segment.set(ValueLayout.JAVA_INT, 4L, 42);
var value = segment.get(ValueLayout.JAVA_INT, 4L);
```

The MemorySegment API

Heap allocation, indexed access

```
var ints = new int[] {1, 2, 3, 4};
var segment = MemorySegment.ofArray(ints); // on-heap

segment.setAtIndex(ValueLayout.JAVA_INT, 2L, 65);
var cell =
   segment.getAtIndex(ValueLayout.JAVA_INT, 2L);
```

The MemorySegment API: File Mapping

Copy from on-heap to off-heap

```
var ints = new int[] {1, 2, 3, 4};
var arraySegment = MemorySegment.ofArray(ints);              // on-heap

var offHeapSegment = Arena.global().allocate(64L);           // off-heap
offHeapSegment.copyFrom(arraySegment);
```

The MemorySegment API: File Mapping

Writing data to a mapped file: you need a ByteBuffer

```
var ints = new int[] {1, 2, 3, 4};
var arraySegment = MemorySegment.ofArray(ints);          // on-heap

var offHeapSegment = Arena.global().allocate(64L);       // off-heap
offHeapSegment.copyFrom(arraySegment);

var byteBuffer = offHeapSegment.asByteBuffer();          // this is a view!
byteBuffer.limit(
   ints.length * (int) ValueLayout.JAVA_INT.byteSize());

try (var file = FileChannel.open(path, CREATE, WRITE)) {
    file.write(byteBuffer);
}
```

Introducing Arena to Allocate / Deallocate

Demo time!

Introducing Arena

An Arena can create off-heap memory segments

It initializes memory segment with zeroes

It is AutoCloseable (more on this in a mn)

It deallocates the memory segments it created on close()

What is this Arena object?

There are four of them:

```
var global   = Arena.global(); // singleton

var confined = Arena.ofConfined();
var shared   = Arena.ofShared();

var auto     = Arena.ofAuto(); // supports legacy
                               // ByteBuffer semantics
```

What is this Arena object?

There are four of them:

	Bounded Lifetime	Closed by the User	Shared among threads
Global	No	No	Yes
Auto			
Confined			
Shared			

What is this Arena object?

There are four of them:

	Bounded Lifetime	Closed by the User	Shared among threads
Global	No	No	Yes
Auto	Yes	No (GC)	Yes
Confined			
Shared			

What is this Arena object?

There are four of them:

	Bounded Lifetime	Closed by the User	Shared among threads
Global	No	No	Yes
Auto	Yes	No (GC)	Yes
Confined	Yes	Yes	No
Shared			

What is this Arena object?

There are four of them:

	Bounded Lifetime	Closed by the User	Shared among threads
Global	No	No	Yes
Auto	Yes	No (GC)	Yes
Confined	Yes	Yes	No
Shared	Yes	Yes	Yes

Benchmarks

`int[]` vs `memorySegment.get(JAVA_INT, ...)`

```
Array                        0.728 ± 0.009   ns/op

OfArray                      1.358 ± 0.003   ns/op

Unsafe                       0.627 ± 0.001   ns/op


Confined                     1.255 ± 0.002   ns/op
Auto                         1.254 ± 0.002   ns/op
Shared                       1.254 ± 0.013   ns/op
Global                       1.258 ± 0.026   ns/op
```

Benchmarks

Looping and summing 512 ints

```
Array                        128.338 ± 0.084  ns/op

OfArray                      131.927 ± 0.761  ns/op

Unsafe                       128.083 ± 0.131  ns/op


Confined                     131.829 ± 0.077  ns/op
Auto                         131.832 ± 0.491  ns/op
Shared                       131.760 ± 0.068  ns/op
Global                       131.727 ± 0.137  ns/op
```

Random Access Performance

Access time is independent of the type of arena

For random direct access
- Overhead is important (2x)
- 3 checks
  1. Is it the right thread?
  2. Has the Arena been Closed?
  3. Is access in bounds?

Loop Performance

Access time is independent of the type of arena

For loop + indexed access

- Fixed cost at the beginning of the loop
- 3 Checks are hoisted out of the loop
  1. Is it the right thread? Is done once
  2. Has the Arena been Closed? Is done once
  3. Is access in bounds? Is elided

After the Break

Memory fragmentation

Application integrity

Jextract

Memory Layout

Structured memory access with offsets and VarHandle

Lazy allocation using Stable Value (prev. 25)

Coffee Break!

Welcome Back!

About Arenas:

- Different arena types with different semantics
- Same access time

What about allocation / deallocation?

Benchmarks

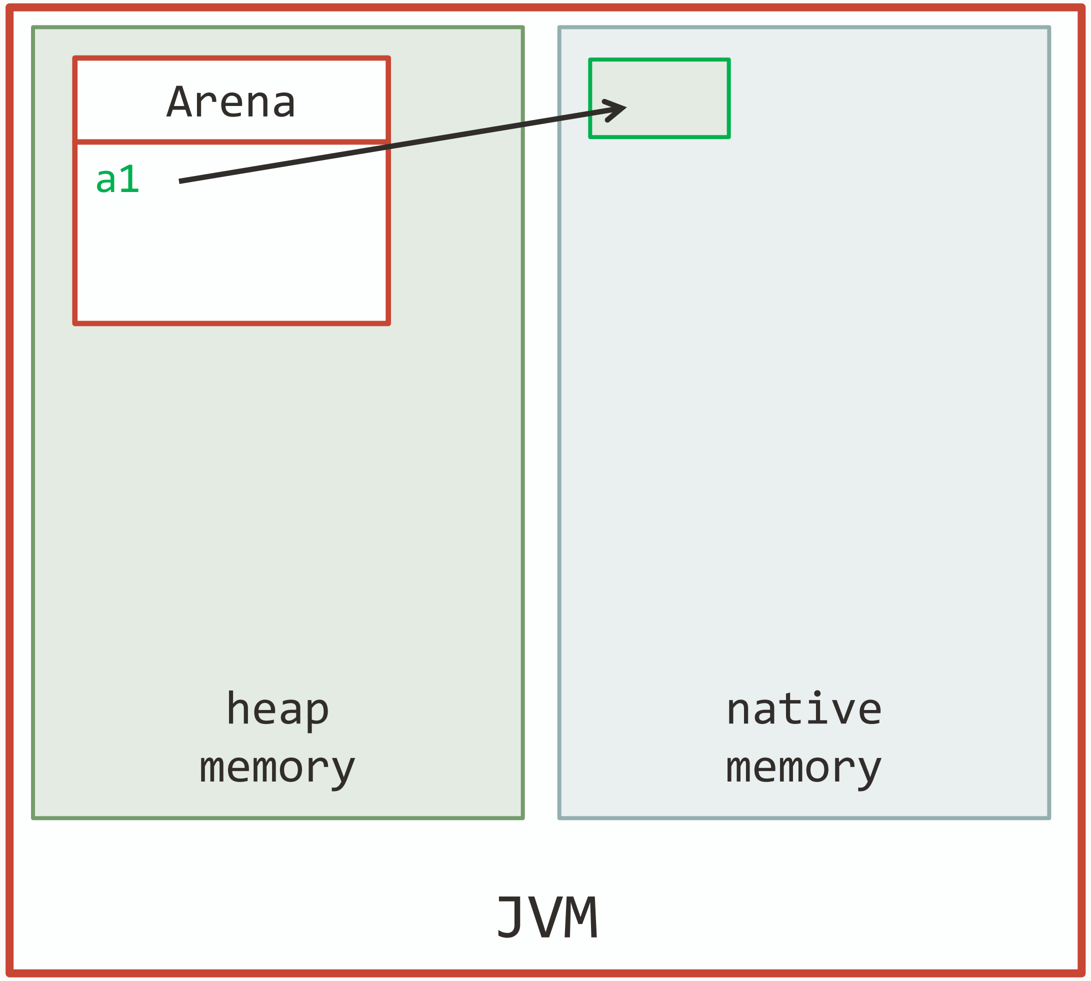
Allocation/Deallocation of an Arena + MemorySegment

```
Array                          2.522 ±   0.015  ns/op

OfArray                        6.494 ±   0.093  ns/op

Unsafe (malloc)               22.834 ±   0.097  ns/op
Unsafe with init              72.338 ±   0.390  ns/op

Confined                      82.287 ±   1.530  ns/op
```

How Do Arenas Allocate Segments?

With two arenas

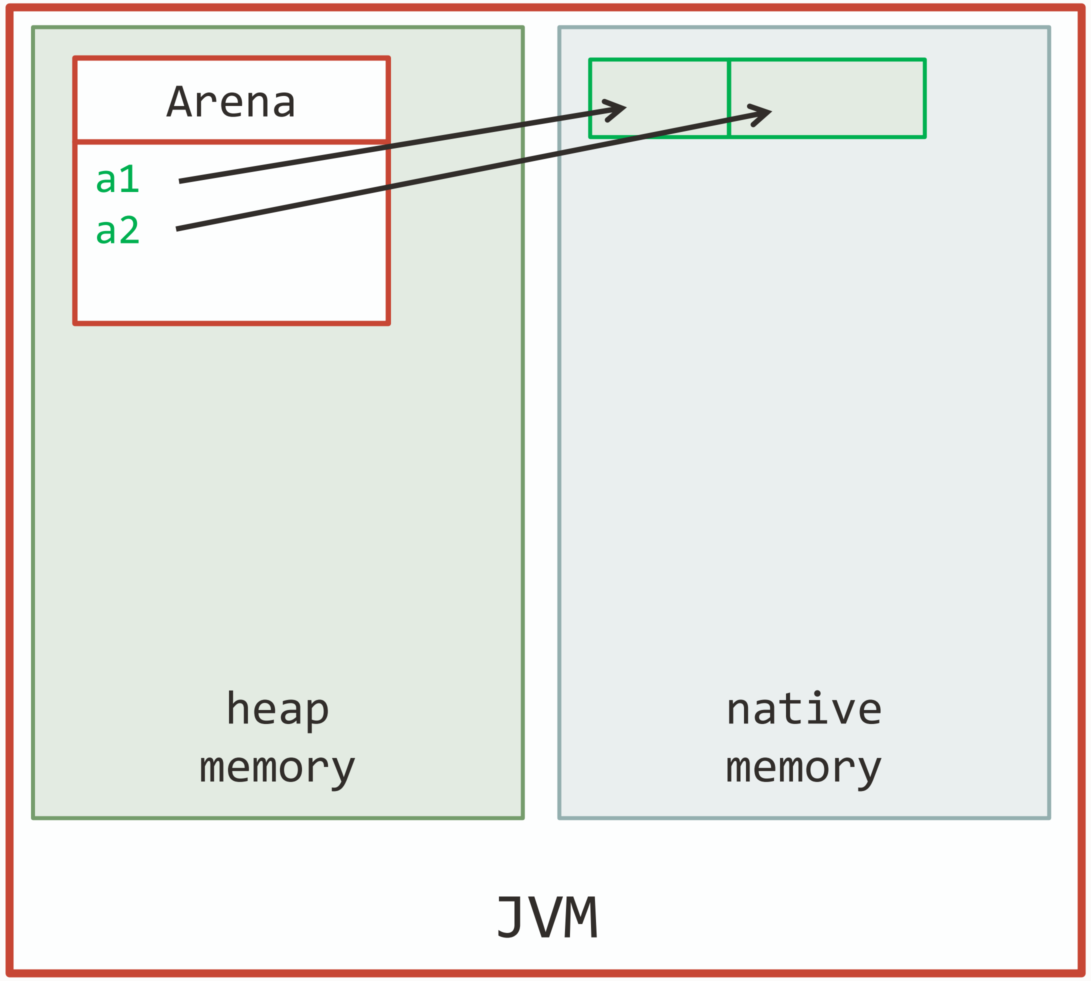
```
var arenaA = Arena.ofConfined();

var a1 = arenaA.allocate(...);
```



How Do Arenas Allocate Segments?

With two arenas

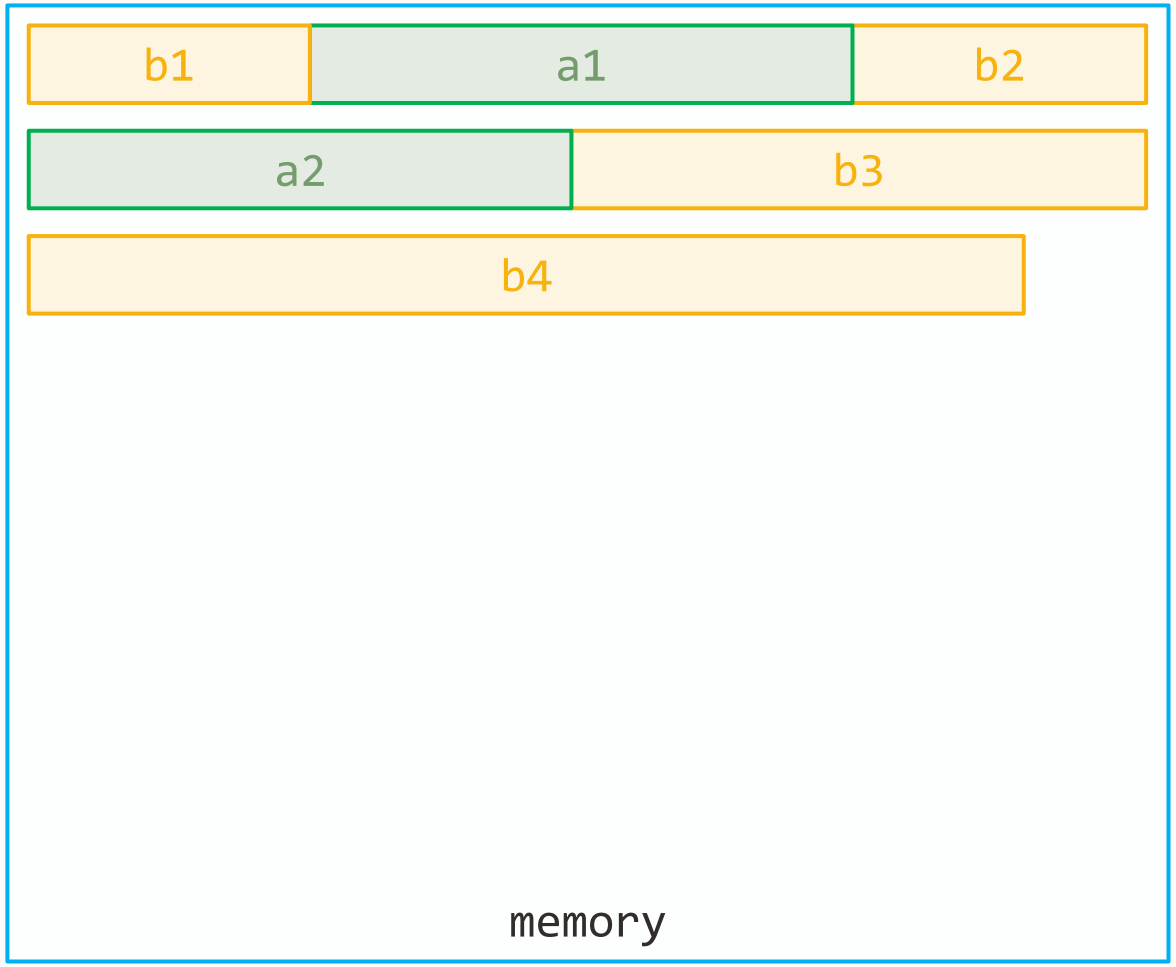
```
var arenaA = Arena.ofConfined();

var a1 = arenaA.allocate(...);
var a2 = arenaA.allocate(...);
```



How Do Arenas Allocate Segments?

With two arenas

```
var arenaA = Arena.ofConfined();
var arenaB = Arena.ofConfined();

var b1 = arenaB.allocate(...);
var a1 = arenaA.allocate(...);

var b2 = arenaB.allocate(...);
var a2 = arenaA.allocate(...);

var b3 = arenaB.allocate(...);
var b4 = arenaB.allocate(...);
```

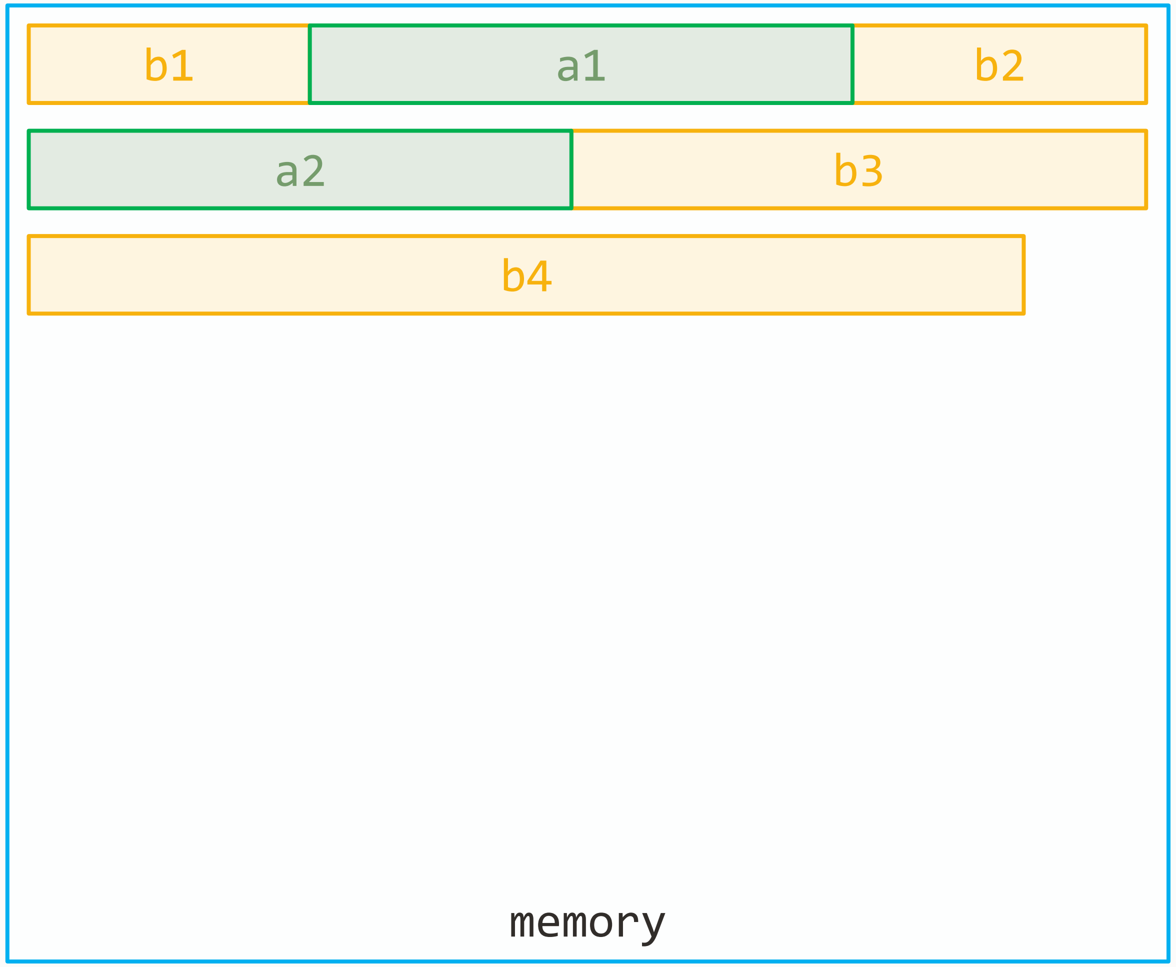


How Do Arenas Allocate Segments?

Then arenaA is closed

```
arenaA.close();
```

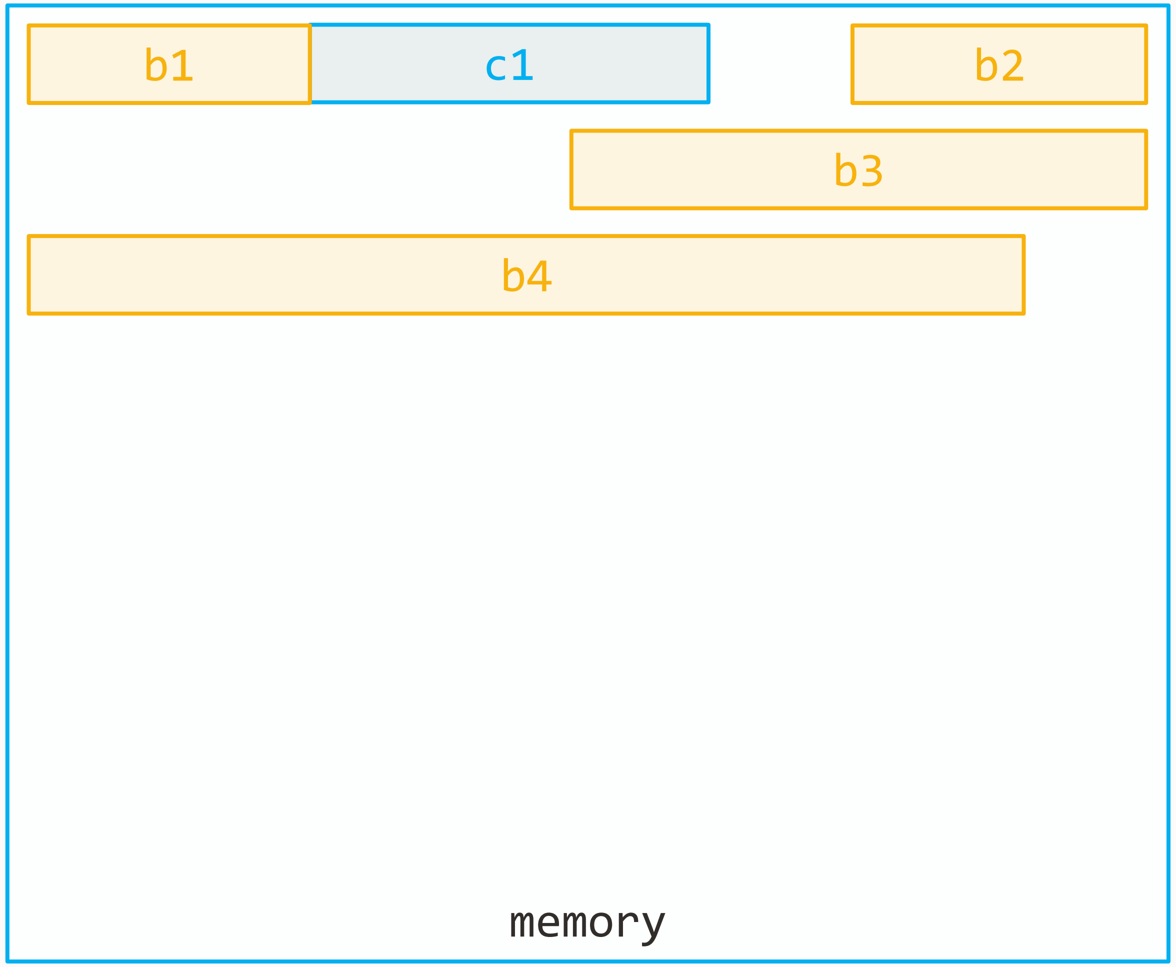
And all its memory segments are deallocated



How Do Arenas Allocate Segments?

Then arenaC is created

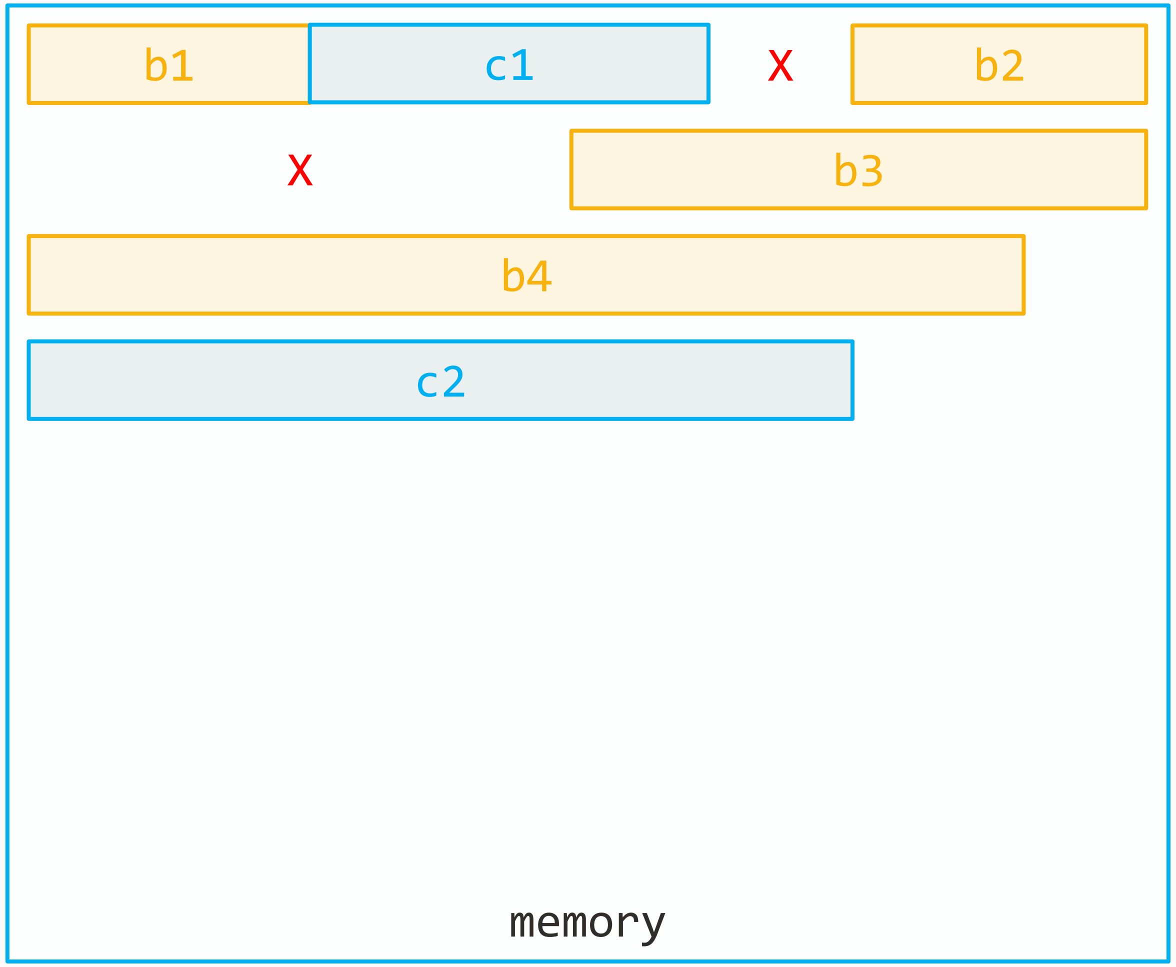
```
var arenaC = Arena.ofConfined();

var c1 = arenaC.allocate(...);
```



How Do Arenas Allocate Segments?

Then arenaC is created

```
var arenaC = Arena.ofConfined();

var c1 = arenaC.allocate(...);
var c2 = arenaC.allocate(...);
```

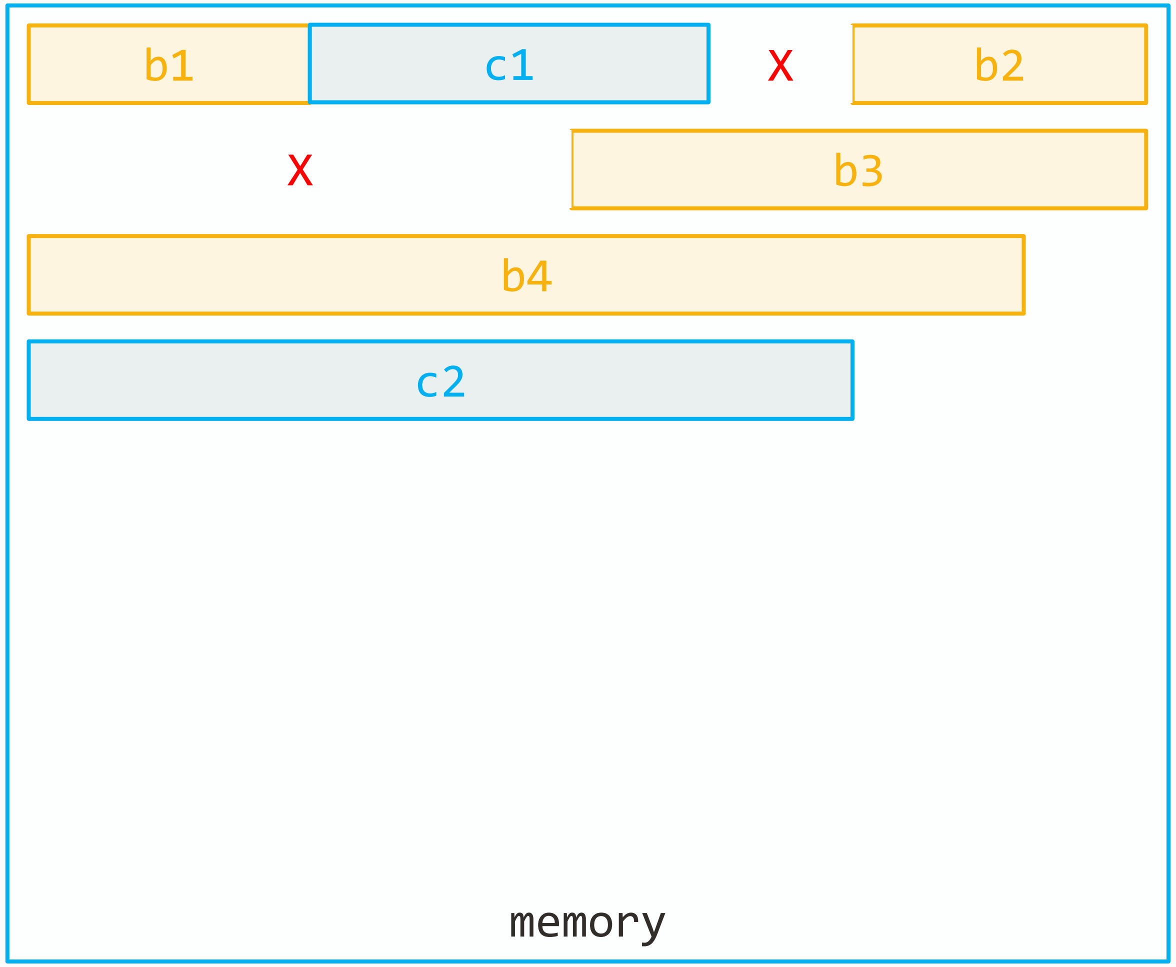


How Do Arenas Allocate Segments?

Then arenaC is created

```
var arenaC = Arena.ofConfined();

var c1 = arenaC.allocate(...);
var c2 = arenaC.allocate(...);
```

And you end up with fragmentation!



Fragmentation

Holes in native memory

It leads to two problems:

- Allocation is slow, you need to find a large enough space for your memory segment
- Not enough contiguous free memory may prevent the creation of a large memory segment

Custom Arenas

Default arenas create fragmentation

But Arena is an interface!

So you can implement your own allocation strategy

- https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/lang/foreign/Arena.html#custom-arenas

Benchmarks

Allocation/Deallocation of an Arena + MemorySegment

```
Array                          2.522 ±   0.015  ns/op

OfArray                        6.494 ±   0.093  ns/op

Unsafe (malloc)               22.834 ±   0.097  ns/op
Unsafe with init              72.338 ±   0.390  ns/op

Confined                      82.287 ±   1.530  ns/op
Auto                         434.694 ± 247.868  ns/op
Shared                      6696.144 ±  37.833  ns/op
```

Deallocation / close()

`ofAuto()`:

`close()` has the same semantics as `ByteBuffer`

> Slooooow!
> In the worst case scenario it calls System.gc()
> (even slooooooower!)

Deallocation / close()

`ofShared()`: you want to avoid having a volatile access in get() (to know if the arena has been closed)

`close()` performs a VM Handcheck with all other threads

- checks method on top of the stack is annotated as performing an access
- checks if the locals contains the closing arena
- ⇒ Linear with the number of platform threads 😭

Arena Uses Cases

Confined: default choice, manual deallocation
Shared = confined + multi thread access

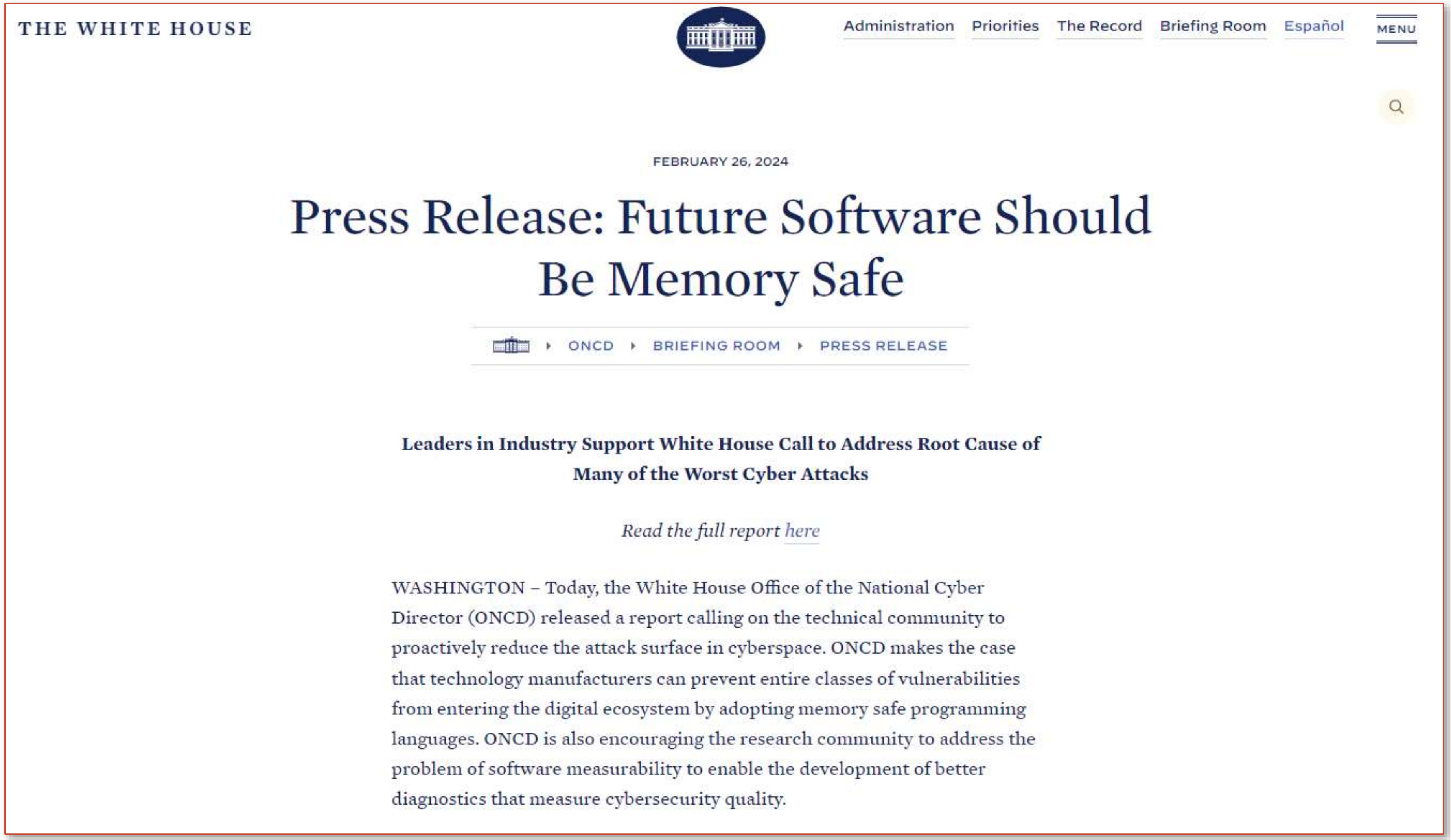
Not the malloc API, you should try to group allocations

Global: permanent memory
Auto: legacy, multi thread access, GC triggered deallocation

Demo time!

Application Integrity

Memory Integrity is a Big Deal!



FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of vulnerabilities from entering the digital ecosystem by adopting memory safe programming languages. ONCD is also encouraging the research community to address the problem of software measurability to enable the development of better diagnostics that measure cybersecurity quality.

Unsafe MemorySegment

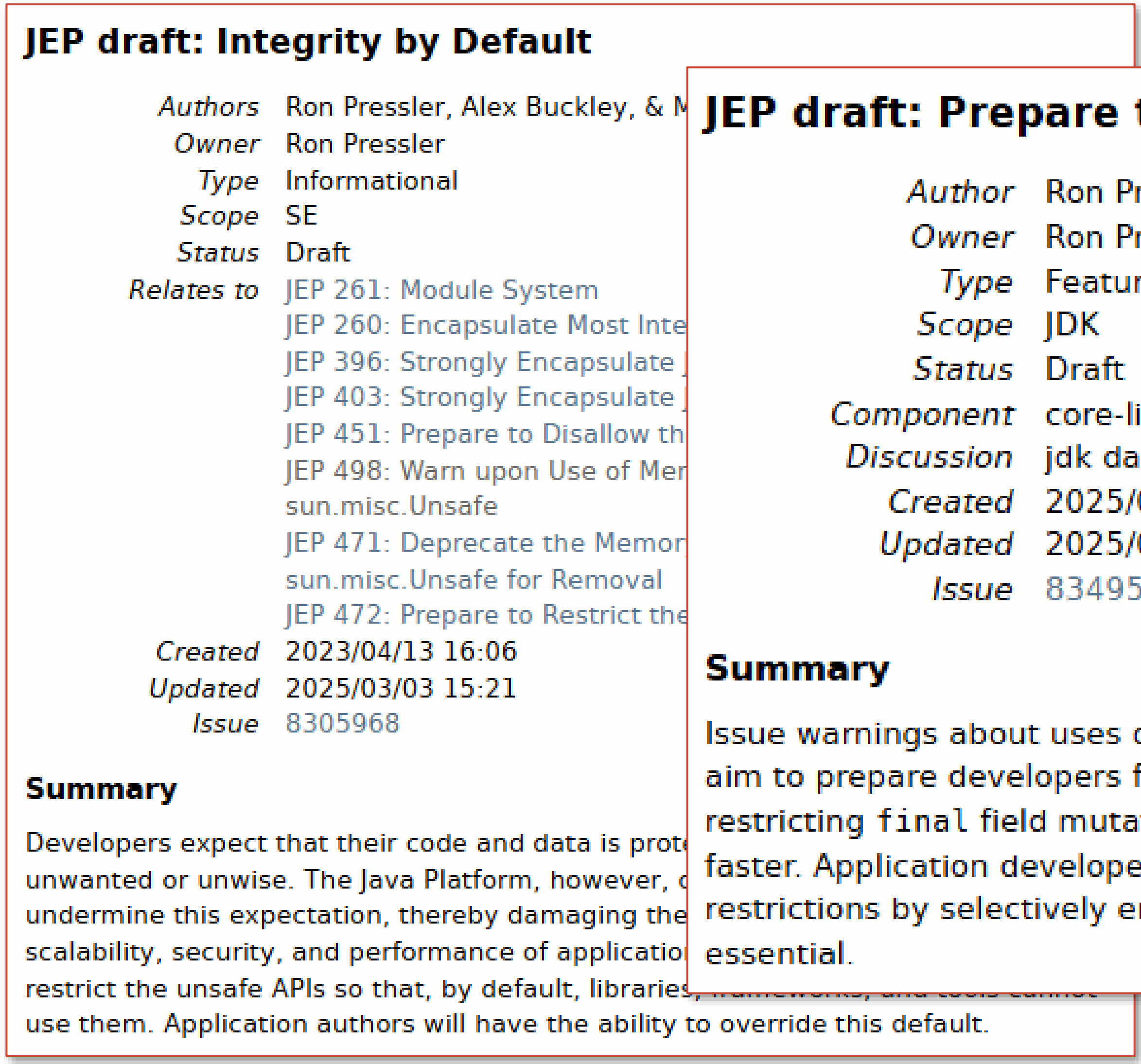
Memory segments are safe by default

- even creating a memory segment from a long is safe (`byteSize` is 0)

`MemorySegment.reinterpret(newSize)`

- opt-in to unsafe, only for native memory
- requires `--enable-native-access` on the command line
  - emits a warning in Java 23, will be an error in the future

Draft JEP: Integrity by Default

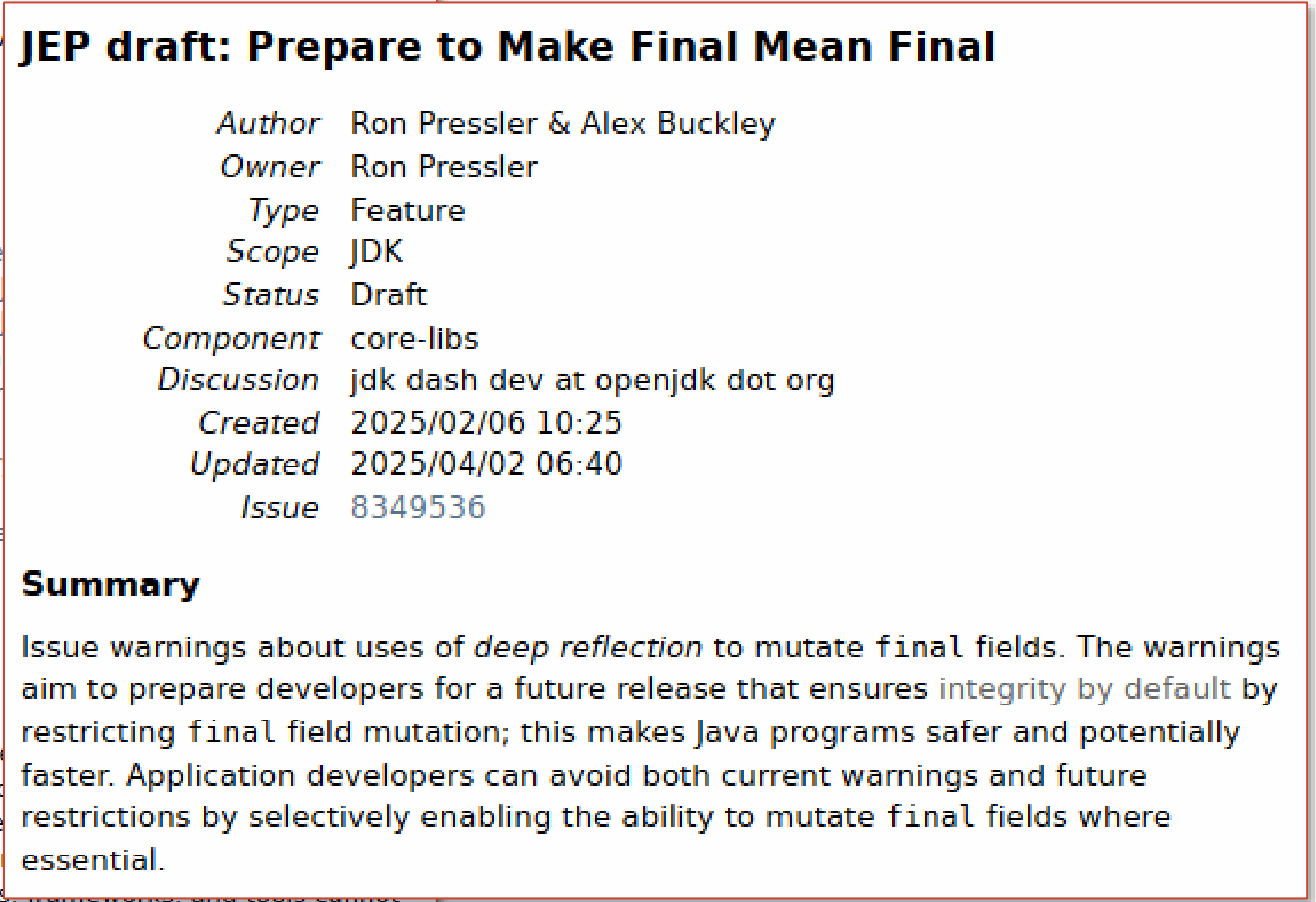


JEP draft: Integrity by Default

Authors	Ron Pressler, Alex Buckley, & M
Owner	Ron Pressler
Type	Informational
Scope	SE
Status	Draft
Relates to	JEP 261: Module System JEP 260: Encapsulate Most Inte JEP 396: Strongly Encapsulate J JEP 403: Strongly Encapsulate J JEP 451: Prepare to Disallow th JEP 498: Warn upon Use of Mer sun.misc.Unsafe JEP 471: Deprecate the Memor sun.misc.Unsafe for Removal JEP 472: Prepare to Restrict the
Created	2023/04/13 16:06
Updated	2025/03/03 15:21
Issue	8305968

Summary

Developers expect that their code and data is prote
unwanted or unwise. The Java Platform, however, c
undermine this expectation, thereby damaging the
scalability, security, and performance of applicatio
restrict the unsafe APIs so that, by default, libraries, frameworks, and tools cannot
use them. Application authors will have the ability to override this default.



JEP draft: Prepare to Make Final Mean Final

Author	Ron Pressler & Alex Buckley
Owner	Ron Pressler
Type	Feature
Scope	JDK
Status	Draft
Component	core-libs
Discussion	jdk dash dev at openjdk dot org
Created	2025/02/06 10:25
Updated	2025/04/02 06:40
Issue	8349536

Summary

Issue warnings about uses of *deep reflection* to mutate `final` fields. The warnings aim to prepare developers for a future release that ensures integrity by default by restricting `final` field mutation; this makes Java programs safer and potentially faster. Application developers can avoid both current warnings and future restrictions by selectively enabling the ability to mutate `final` fields where essential.

Draft JEP: Integrity by Default

JEP 261:	Module System
JEP 260:	Encapsulate Most Internal APIs
JEP 396:	Strongly Encapsulate JDK Internals by Default
JEP 403:	Strongly Encapsulate JDK Internals
JEP 451:	Prepare to Disallow the Dynamic Loading of Agents
JEP 471:	Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal
JEP 472:	Prepare to Restrict the Use of JNI
JEP 498:	Warn upon Use of Memory-Access Methods in sun.misc.Unsafe

Demo time!

Jextract and MemoryLayout

What is Jextract?

Simplify C interoperability

- Jextract takes a `.h` file and creates java classes from it
- It creates one class for the `.h` with the function definitions
- Then one per `struct`

What is Jextract?

Uses LLVM internally

To correctly parse C declarations

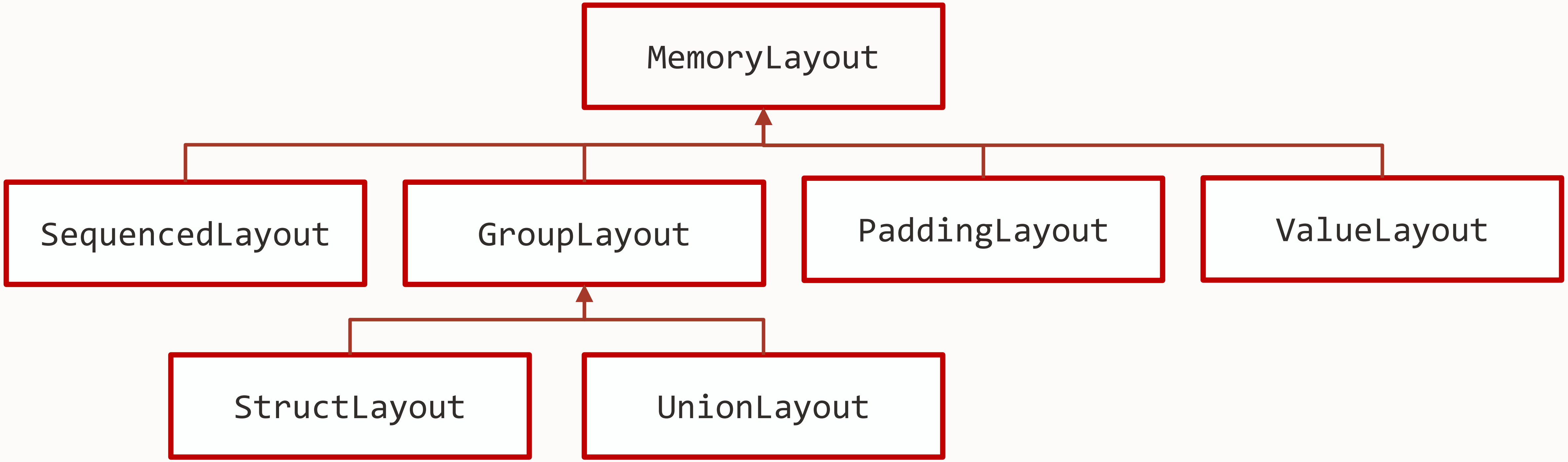
And to extract platform/OS definitions (eg: what is the size of an int)

It is an external tool, that needs to be downloaded separately

https://jdk.java.net/jextract

What is MemoryLayout?

It is an interface that describe a piece of memory



A MemoryLayout can be Named

Defining a struct Point

```
var pointLayout = MemoryLayout.structLayout(
   ValueLayout.JAVA_INT,
   ValueLayout.JAVA_INT
);
```

MemoryLayout Size and Offset

Size of the struct Point, offset of x and y in Point

```
var pointLayout = MemoryLayout.structLayout(
   ValueLayout.JAVA_INT,
   ValueLayout.JAVA_INT
);

long pointLayoutSize = pointLayout.byteSize();
long xOffset = pointLayout.byteOffset(0);   // by index
```

MemoryLayout Size and Offset

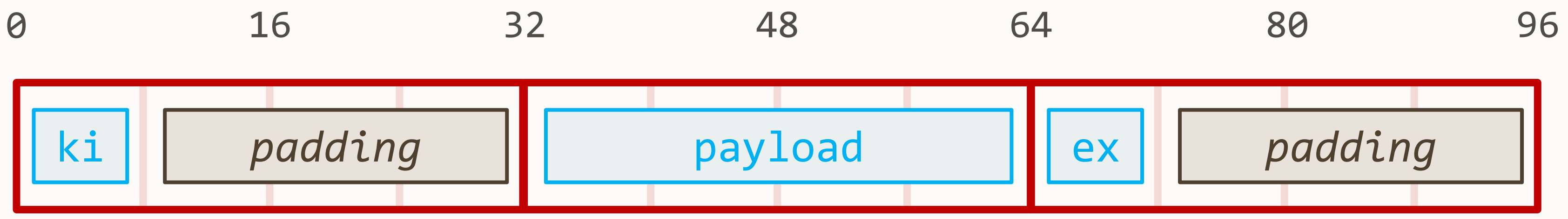
Size of the struct `Point`, offset of x and y in `Point`

```
var pointLayout = MemoryLayout.structLayout(
   ValueLayout.JAVA_INT.withName("x"),
   ValueLayout.JAVA_INT.withName("y")
).withName("point");

long pointLayoutSize = pointLayout.byteSize();
long xOffset = pointLayout.byteOffset(0);   // by index
long yOffset = pointLayout.byteOffset("y"); // by name
```

Alignment and Padding

Memory layouts need padding

```
struct {
  char kind;
  int payload;
  char extra;
}
```



```
static final MemoryLayout LAYOUT =
   MemoryLayout.structLayout(
      ValueLayout.JAVA_BYTE.withName("kind"),
      MemoryLayout.paddingLayout(3),
      ValueLayout.JAVA_INT.withName("payload"),
      ValueLayout.JAVA_BYTE.withName("extra"),
      MemoryLayout.paddingLayout(3)
   );
```

Demo time!

VarHandle

What is a VarHandle?

An object that gives access to fields with different semantics:

- a get / set access (plain, opaque, volatile)
- and concurrent access: compareAndSet, getAndAdd, …

It hides the offset and size computations to access the elements of your memory layout

Demo time!

VarHandle Caveats

1) The compiler doesn’t do any type checking: it trusts you! (and the different IDE are not there to hep you…)

2) It allows conversion at runtime

    It can be convenient, but can lead to autoboxing

    You can use `withInvokeExactBehavior()`

Benchmarks

Compute the sum of 512 point.x + point.y

```
OfArray with offset       214.479 ± 1.712  ns/op
OfArray with VarHandle    141.404 ± 0.081  ns/op

Unsafe with offset        137.518 ± 0.476  ns/op

Arena with offset         212.881 ± 3.436  ns/op
Arena with VarHandle      141.190 ± 0.107  ns/op
```

Using Offsets or VarHandle?

Offset computation by the user is slow

VarHandle offers an access pattern to the JVM, which gives you better performance

Jextract does not create VarHandles, only offsets

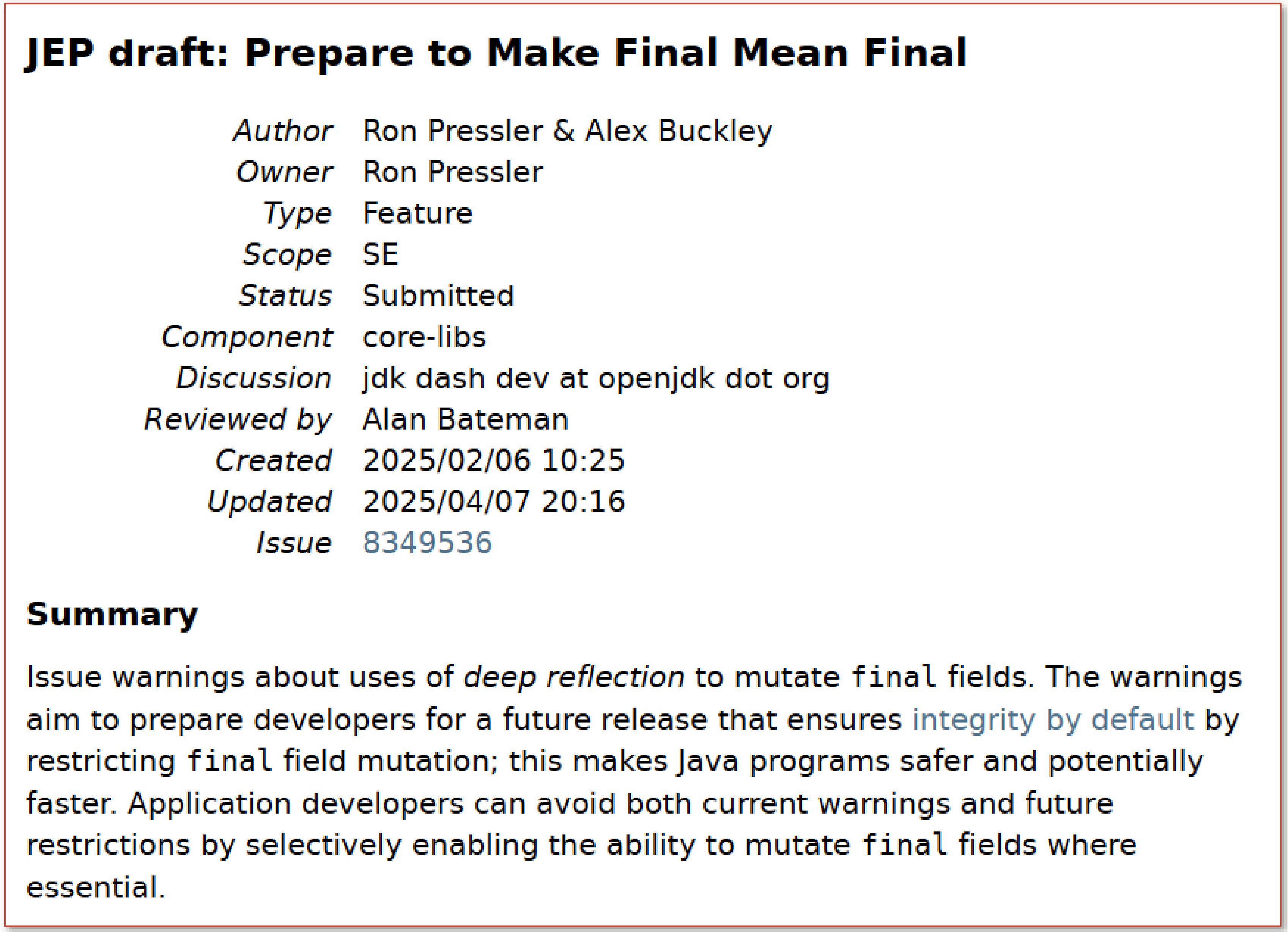
Stable Values

Stable Value

A stable value holds an eventually non-modifiable data

Three guarantees:

1) Initialized when the value is first requested
   Not initialized at application startup nor at class initialization
2) Initialization code is run once
3) Once initialized, treated as (a real) constant by the JVM



JEP draft: Prepare to Make Final Mean Final

Author	Ron Pressler & Alex Buckley
Owner	Ron Pressler
Type	Feature
Scope	SE
Status	Submitted
Component	core-libs
Discussion	jdk dash dev at openjdk dot org
Reviewed by	Alan Bateman
Created	2025/02/06 10:25
Updated	2025/04/07 20:16
Issue	8349536

Summary

Issue warnings about uses of *deep reflection* to mutate `final` fields. The warnings aim to prepare developers for a future release that ensures integrity by default by restricting `final` field mutation; this makes Java programs safer and potentially faster. Application developers can avoid both current warnings and future restrictions by selectively enabling the ability to mutate `final` fields where essential.

Stable Value

Current state of the high-level API (prev. in 25)

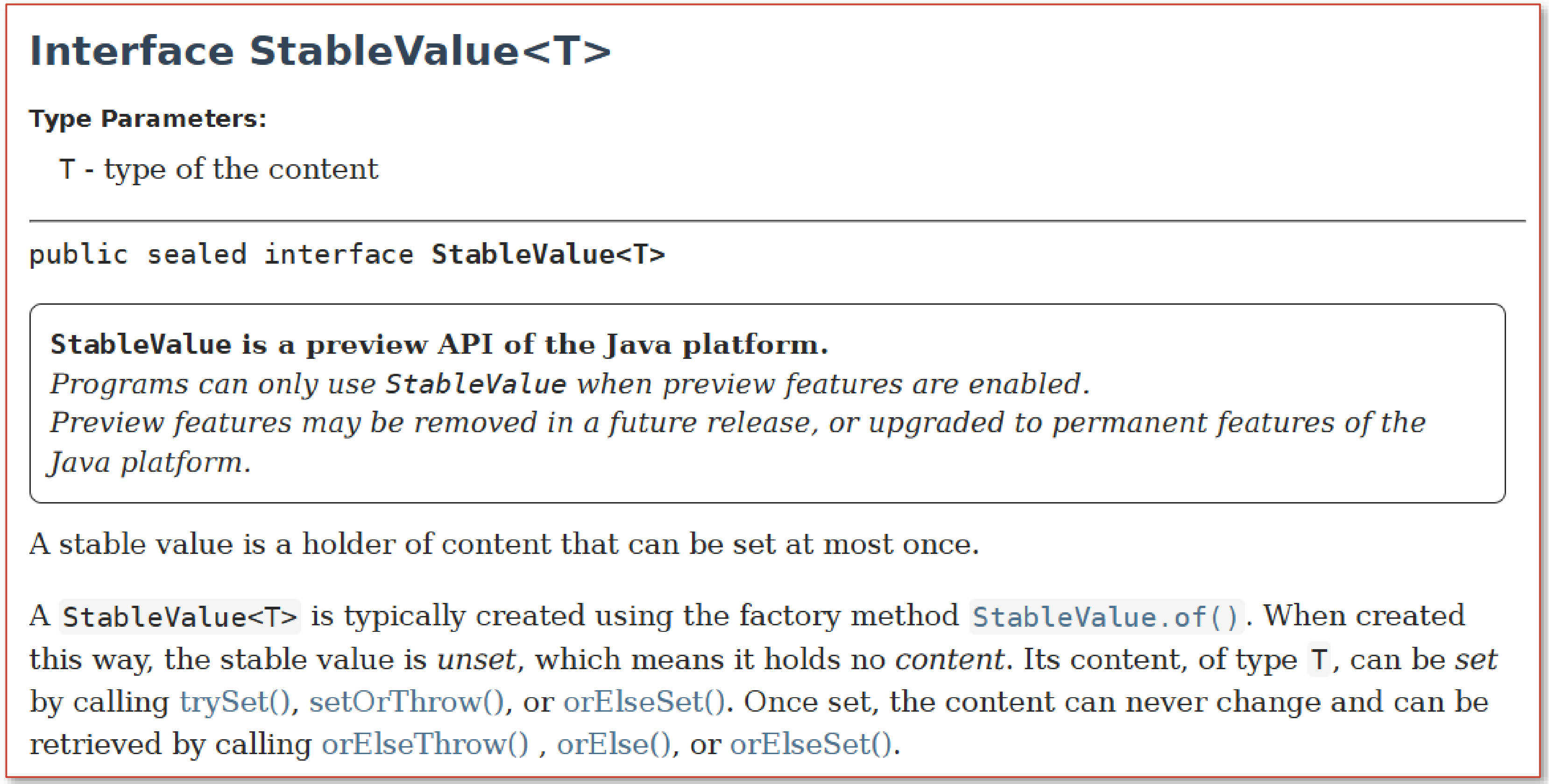
```
class StableValue {
   static <T> Supplier<T>
   supplier(Supplier<? extends T> supplier) { ... }

   static <T> List<T>
   list(int size, IntFunction<? extends T> mapper) { ... }

   static <K, V> Map<K, V>
   map(Set<K> keys, Function<? super K, ? extends V> mapper) { ... }
}
```

Stable Value – Performance

```
Java 24
confinedStableMapLoop     avgt    5  140.513 ± 0.132  ns/op
confinedStableValueLoop   avgt    5  140.973 ± 0.468  ns/op
confinedVarHandleLoop     avgt    5  140.862 ± 0.583  ns/op

Java 25 (EA)
confinedStableMapLoop     avgt    5  22294.893 ± 99.458  ns/op
confinedStableValueLoop   avgt    5    140.634 ±  0.319  ns/op
confinedVarHandleLoop     avgt    5    140.533 ±  0.082  ns/op
```

Stable Value

`https://cr.openjdk.org/~pminborg/stable-values2/api/java.base/java/lang/StableValue.html`



Interface StableValue<T>

Type Parameters:

`T` - type of the content

```
public sealed interface StableValue<T>
```

`StableValue` is a preview API of the Java platform.
Programs can only use `StableValue` when preview features are enabled.
Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.

A stable value is a holder of content that can be set at most once.

A `StableValue<T>` is typically created using the factory method `StableValue.of()`. When created this way, the stable value is *unset*, which means it holds no *content*. Its content, of type `T`, can be *set* by calling trySet(), setOrThrow(), or orElseSet(). Once set, the content can never change and can be retrieved by calling orElseThrow() , orElse(), or orElseSet().

Demo time!

Panama rocks!