



Algo**l**ab

Eher speziell

Kleiner Einblick in die Vector- und Memory-API

Michael Wiedeking

JUG Saxony Day, 25. September 2026

Motivation

- „Native“ Funktionen
- Speicher
- SIMD (Single Instruction – Multiple Data)

Motivation

- Java Native Interface (JNI)
- Unsafe
- ByteBuffer
- —

Foreign Function & Memory API (FFM)

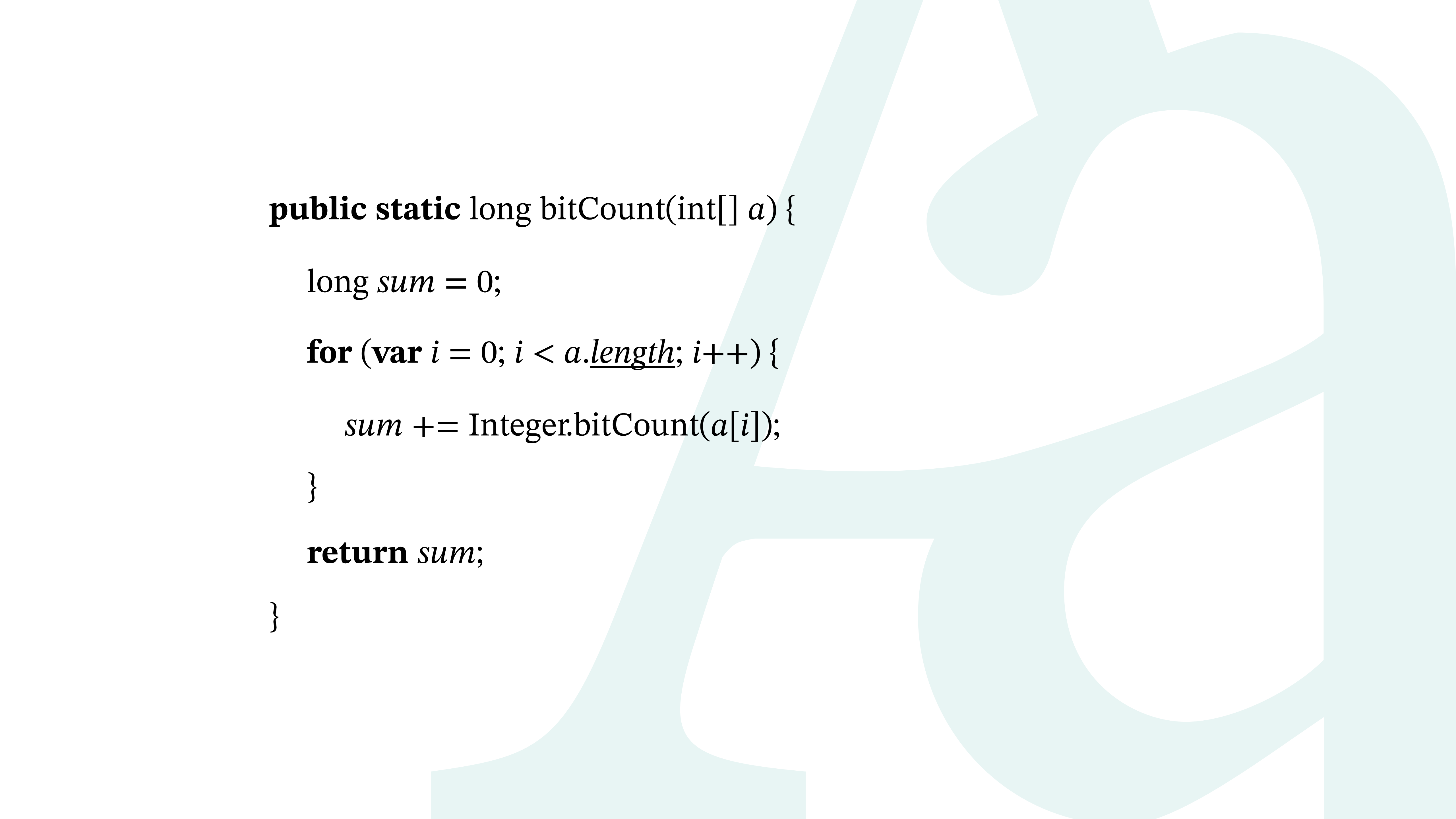
- Off-Heap Memory
- Foreign Functions

Vector API

- Vector Shape (64 Bit, 128 Bit, 256 Bit, 512 Bit)
- Vector Species (byte, short, int, long, float, double)
- Vector, VectorMask und VectorShuffle sind *value-based*

Vectors





```
public static long bitCount(int[] a) {  
    long sum = 0;  
    for (var i = 0; i < a.length; i++) {  
        sum += Integer.bitCount(a[i]);  
    }  
    return sum;  
}
```

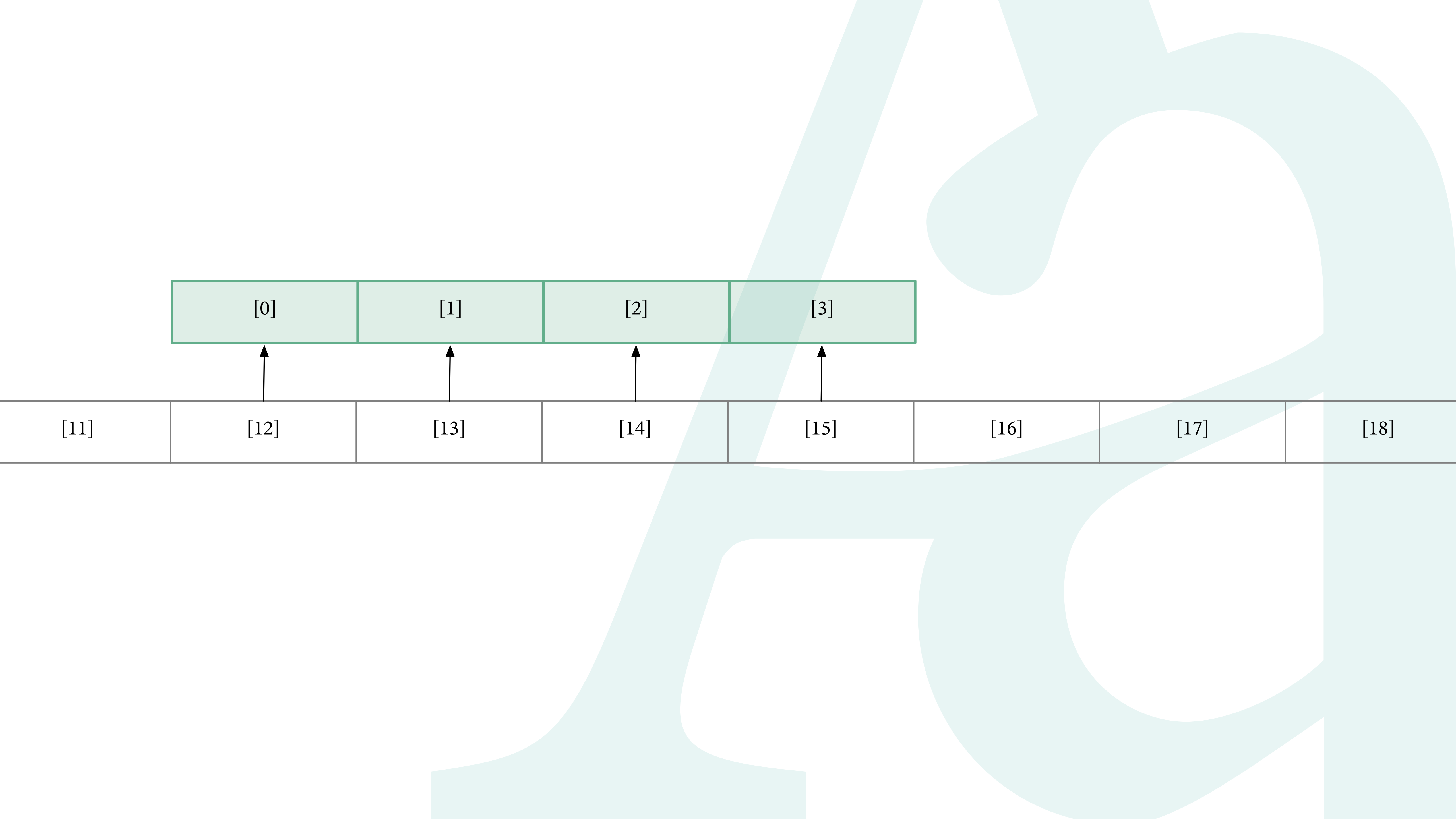
```
public static long bitCount(int[] a) {  
  
    long sum = 0;  
  
    for (var i = 0; i < a.length; i++) {  
  
        sum += Integer.bitCount(a[i]);  
  
    }  
  
    return sum;  
}
```

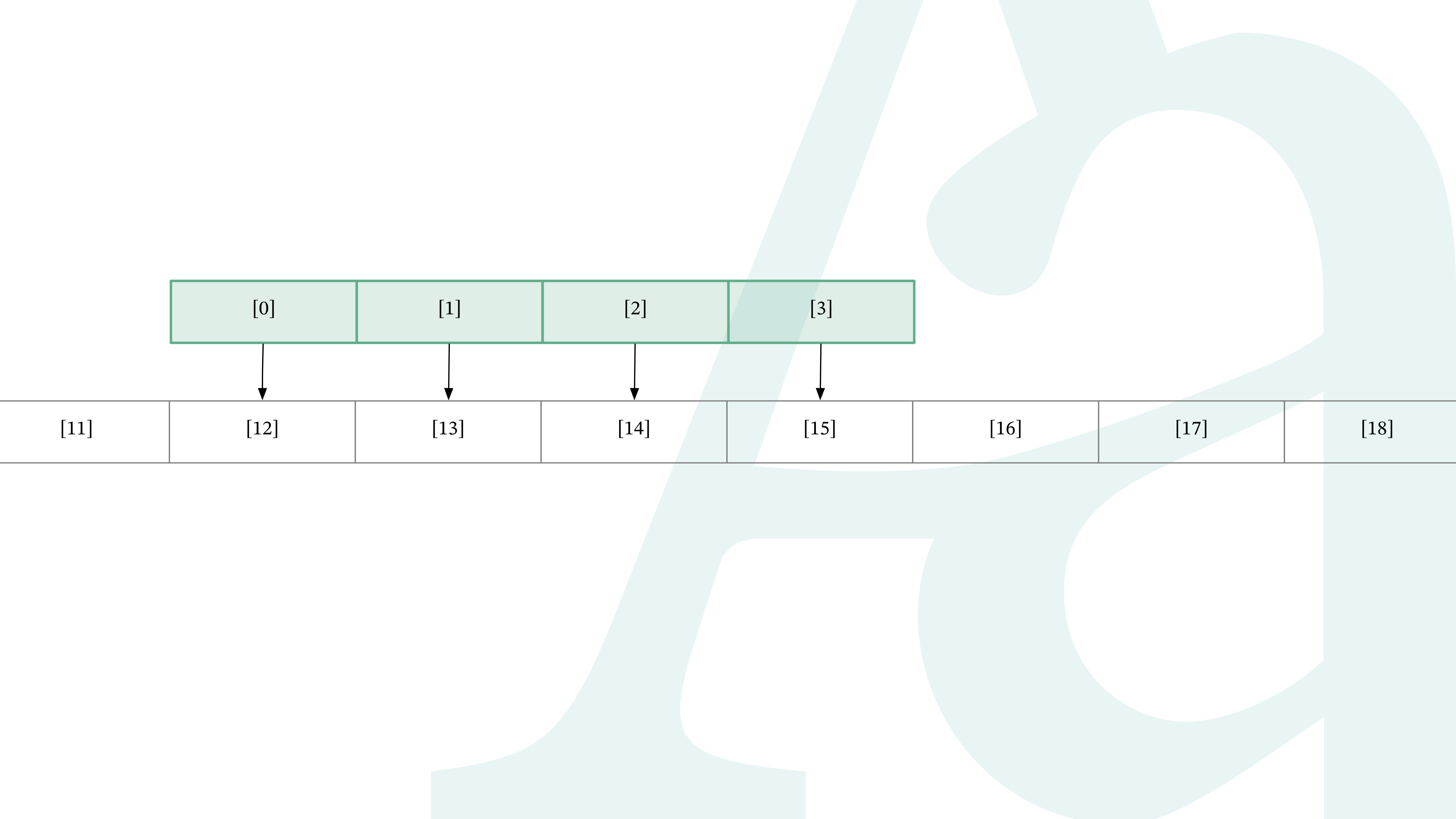
	64 Bit	128 Bit	256 Bit	512 Bit
byte	8	16	32	64
short	4	8	16	32
int	2	4	8	16
long	1	2	4	8
float	2	4	8	16
double	1	2	4	8

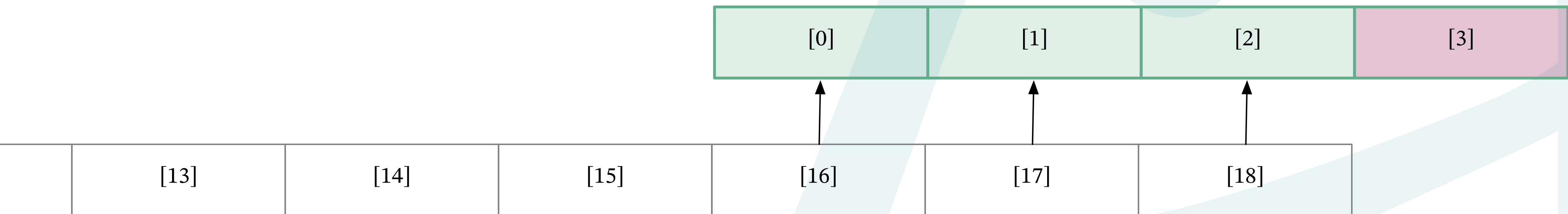
Anzahl Lanes pro Vector



[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
------	------	------	------	------	------	------	------



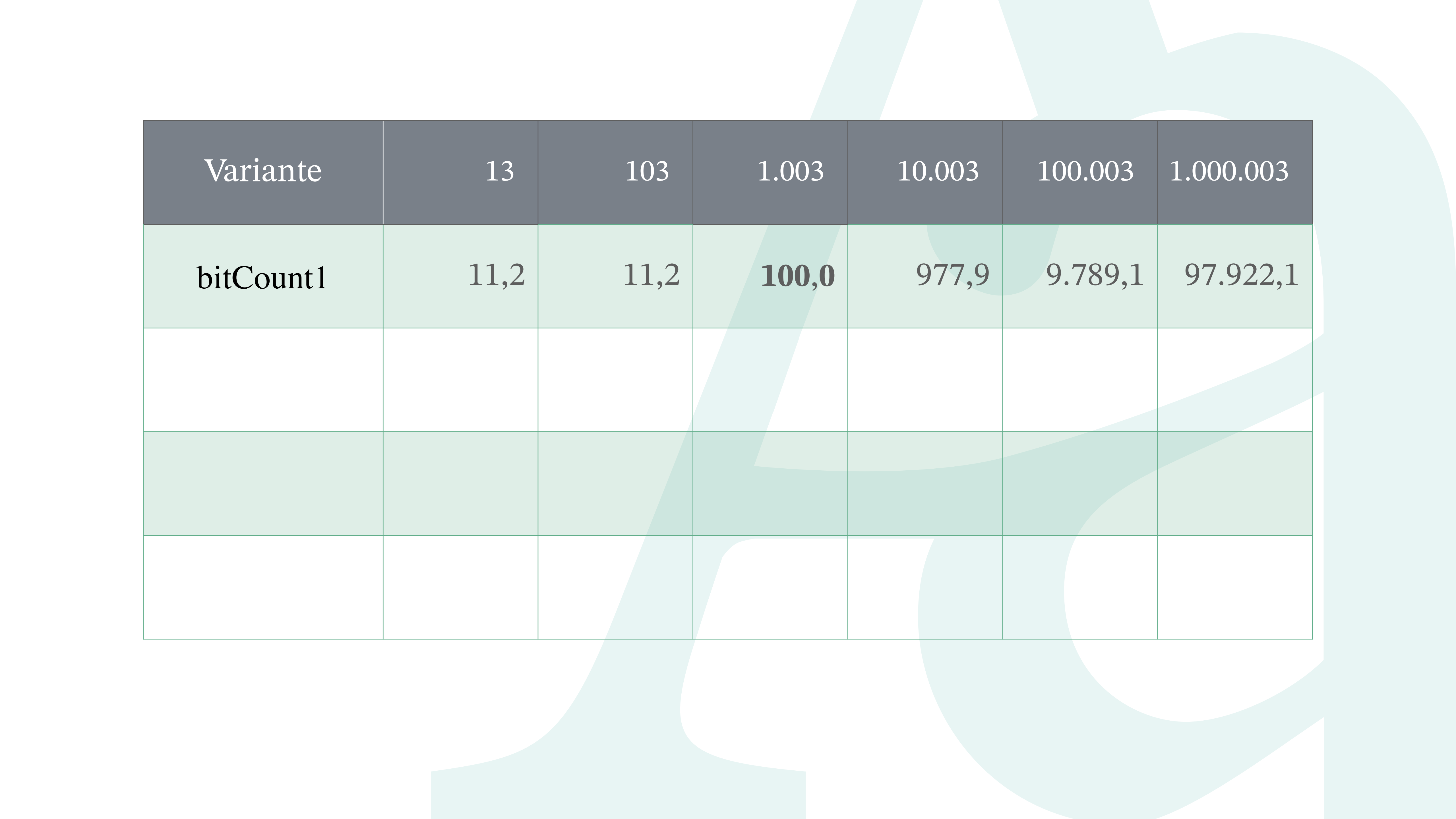






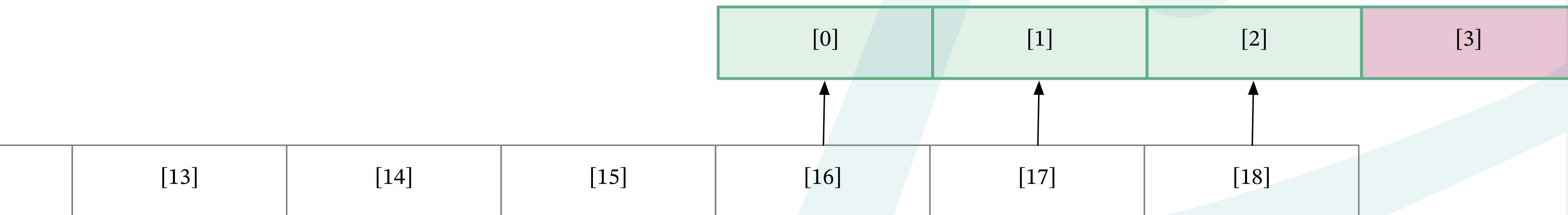
```
static final VectorSpecies<Integer> SPECIES = IntVector.SPECIES_PREFERRED;
```

```
public static long bitCount2(int[] a) {  
    long sum = 0;  
    int i = 0;  
    int upperBound = SPECIES.loopBound(a.length);  
    for (; i < upperBound; i += SPECIES.length()) {  
        var va = IntVector.fromArray(SPECIES, a, i);  
        va = va.lanewise(VectorOperators.BIT_COUNT);  
        sum += va.reduceLanesToLong(VectorOperators.ADD);  
    }  
    for (; i < a.length; i++) {  
        sum += Integer.bitCount(a[i]);  
    }  
    return sum;  
}
```

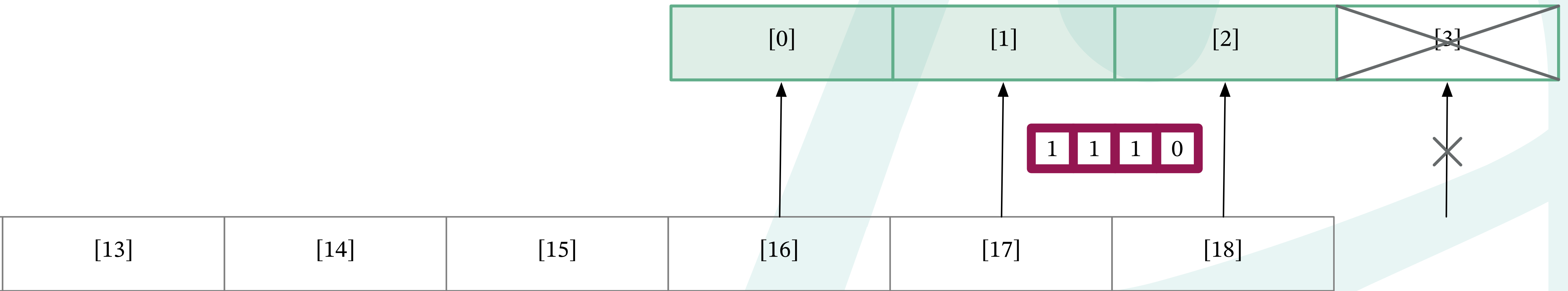


Variante	13	103	1.003	10.003	100.003	1.000.003
bitCount1	11,2	11,2	100,0	977,9	9.789,1	97.922,1

Variante	13	103	1.003	10.003	100.003	1.000.003
bitCount1	11,2	11,2	100,0	977,9	9.789,1	97.922,1
bitCount2	10,9	10,9	66,7	577,9	5.844,5	59.533,3



VectorMask



```
public static long bitCount3(int[] a) {  
    long sum = 0;  
    for (int i = 0; i < a.length; i += SPECIES.length()) {  
        var mask = SPECIES.indexInRange(i, a.length);  
        var va = IntVector.fromArray(SPECIES, a, i, mask);  
        va = va.lanewise(VectorOperators.BIT_COUNT);  
        sum += va.reduceLanesToLong(VectorOperators.ADD, mask);  
    }  
    return sum;  
}
```

Variante	13	103	1.003	10.003	100.003	1.000.003
bitCount1	11,2	11,2	100,0	977,9	9.789,1	97.922,1
bitCount2	10,9	10,9	66,7	577,9	5.844,5	59.533,3
bitCount3	22,4	22,1	244,5	2.822,4	15.588,8	155.966,7

```
public static long bitCount4(int[] a) {  
    long sum = 0;  
    int i = 0;  
    int upperBound = SPECIES.loopBound(a.length);  
    for (; i < upperBound; i += SPECIES.length()) {  
        var va = IntVector.fromArray(SPECIES, a, i);  
        va = va.lanewise(VectorOperators.BIT_COUNT);  
        sum += va.reduceLanesToLong(VectorOperators.ADD);  
    }  
  
    var mask = SPECIES.indexInRange(i, a.length);  
    var va = IntVector.fromArray(SPECIES, a, i, mask);  
    va = va.lanewise(VectorOperators.BIT_COUNT);  
    sum += va.reduceLanesToLong(VectorOperators.ADD, mask);  
  
    return sum;  
}
```

Variante	13	103	1.003	10.003	100.003	1.000.003
bitCount1	11,2	11,2	100,0	977,9	9.789,1	97.922,1
bitCount2	10,9	10,9	66,7	577,9	5.844,5	59.533,3
bitCount3	22,4	22,1	244,5	2.822,4	15.588,8	155.966,7
bitCount4	11,2	11,2	66,7	566,7	5.811,2	57.666,7

```
static void scalarComputation(float[] a, float[] b, float[] c) {  
    assert a.length == b.length  
    assert b.length == c.length  
    for (int i = 0; i < a.length; i++) {  
        c[i] = (a[i] * a[i] + b[i] * b[i]) * (-1.0f);  
    }  
}
```

```
static final VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;

static void vectorComputation(float[] a, float[] b, float[] c) {

    int i = 0;
    int upperBound = SPECIES.loopBound(a.length);
    for (; i < upperBound; i += SPECIES.length()) {

        var va = FloatVector.fromArray(SPECIES, a, i);
        var vb = FloatVector.fromArray(SPECIES, b, i);
        var vc = va.mul(va).add(vb.mul(vb)).neg();
        vc.intoArray(c, i);

    }

    ...

}
```

```
static final VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;
```

```
static void vectorComputation(float[] a, float[] b, float[] c) {
```

```
    int i = 0;
```

```
    int upperBound = SPECIES.loopBound(a.length);
```

```
    for (; i < upperBound; i += SPECIES.length()) {
```

```
        var va = FloatVector.fromArray(SPECIES, a, i);
```

```
        var vb = FloatVector.fromArray(SPECIES, b, i);
```

```
        var vc = va.mul(va).add(vb.mul(vb)).neg();
```

```
        vc.intoArray(c, i);
```

```
    }
```

```
    ...
```

```
}
```

```
    for (; i < a.length; i++) {
```

```
         $c[i] = (a[i] * a[i] + b[i] * b[i]) * (-1.0f);$ 
```

```
    }
```

Und was alles kann man machen?

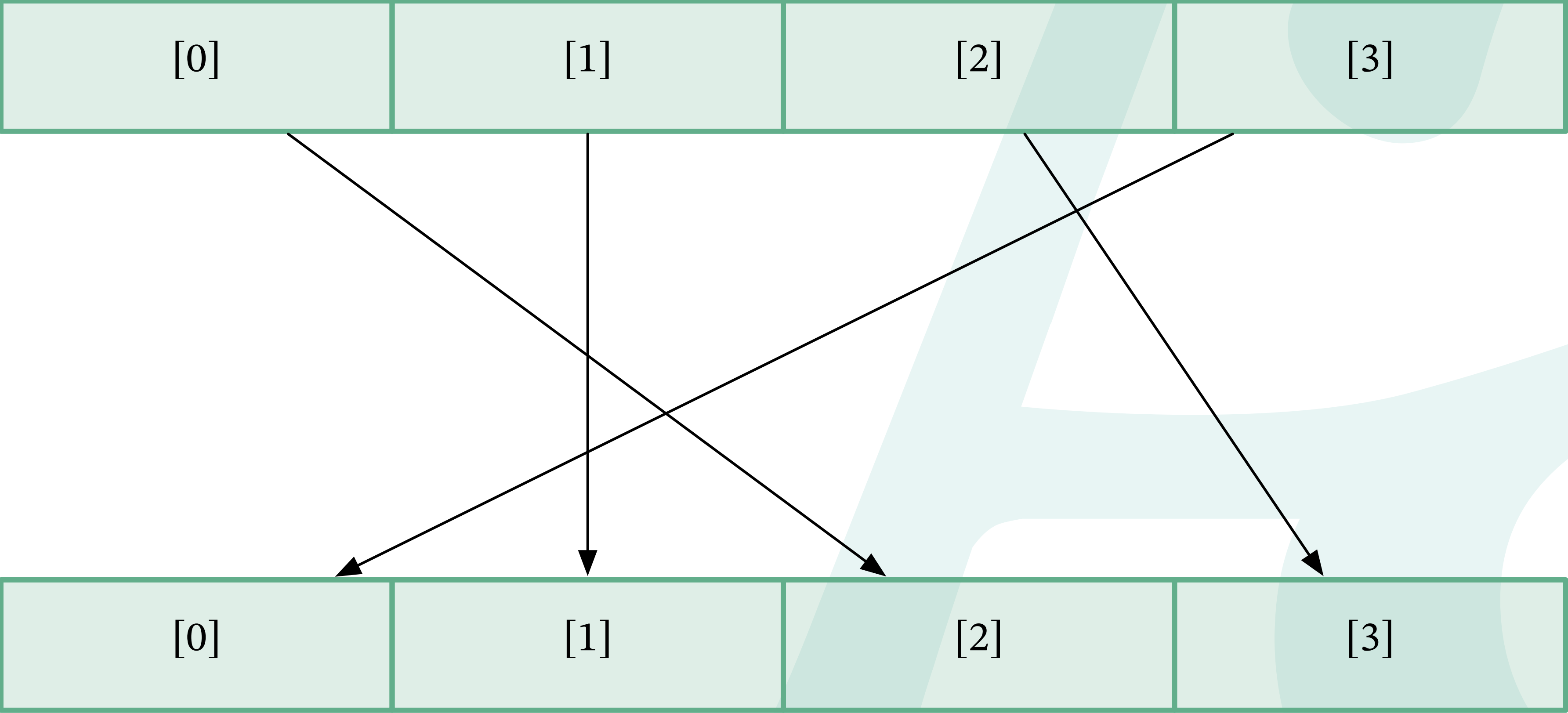
- VectorOperators.Unary
- VectorOperators.Binary
- VectorOperators.Associative
- VectorOperators.Comparison
- VectorOperators.Conversion<E, F>
- VectorOperators.Ternary
- VectorOperators.Test

• ABS	• D2I	• I2S	• NE	• SSUB
• ACOS	• D2L	• IS_DEFAULT	• NEG	• SUADD
• ADD	• D2S	• IS_FINITE	• NOT	• SUB
• AND	• DIV	• IS_INFINITE	• OR	• SUSUB
• AND_NOT	• EQ	• IS_NAN	• POW	• TAN
• ASHR	• EXP	• IS_NEGATIVE	• REINTERPRET_D2L	• TANH
• ASIN	• EXPAND_BITS	• L2B	• REINTERPRET_F2I	• TRAILING_ZEROS...
• ATAN	• EXPM1	• L2D	• REINTERPRET_I2F	• UGE
• ATAN2	• F2B	• L2F	• REINTERPRET_L2D	• UGT
• B2D	• F2D	• L2I	• REVERSE	• ULE
• B2F	• F2I	• L2S	• REVERSE_BYTES	• ULT
• B2I	• F2L	• LE	• ROL	• UMAX
• B2L	• F2S	• LEADING_ZEROS...	• ROR	• UMIN
• B2S	• FIRST_NONZERO	• LOG	• S2B	• XOR
• BIT_COUNT	• FMA	• LOG10	• S2D	• ZERO_EXTEND_B2I
• BITWISE_BLEND	• GE	• LOG1P	• S2F	• ZERO_EXTEND_B2L
• CBRT	• GT	• LSHL	• S2I	• ZERO_EXTEND_B2S
• COMPRESS_BITS	• HYPOT	• LSHR	• S2L	• ZERO_EXTEND_I2L
• COS	• I2B	• LT	• SADD	• ZERO_EXTEND_S2I
• COSH	• I2D	• MAX	• SIN	• ZERO_EXTEND_S2L
• D2B	• I2F	• MIN	• SINH	• ZOMO
• D2F	• I2L	• MUL	• SQRT	

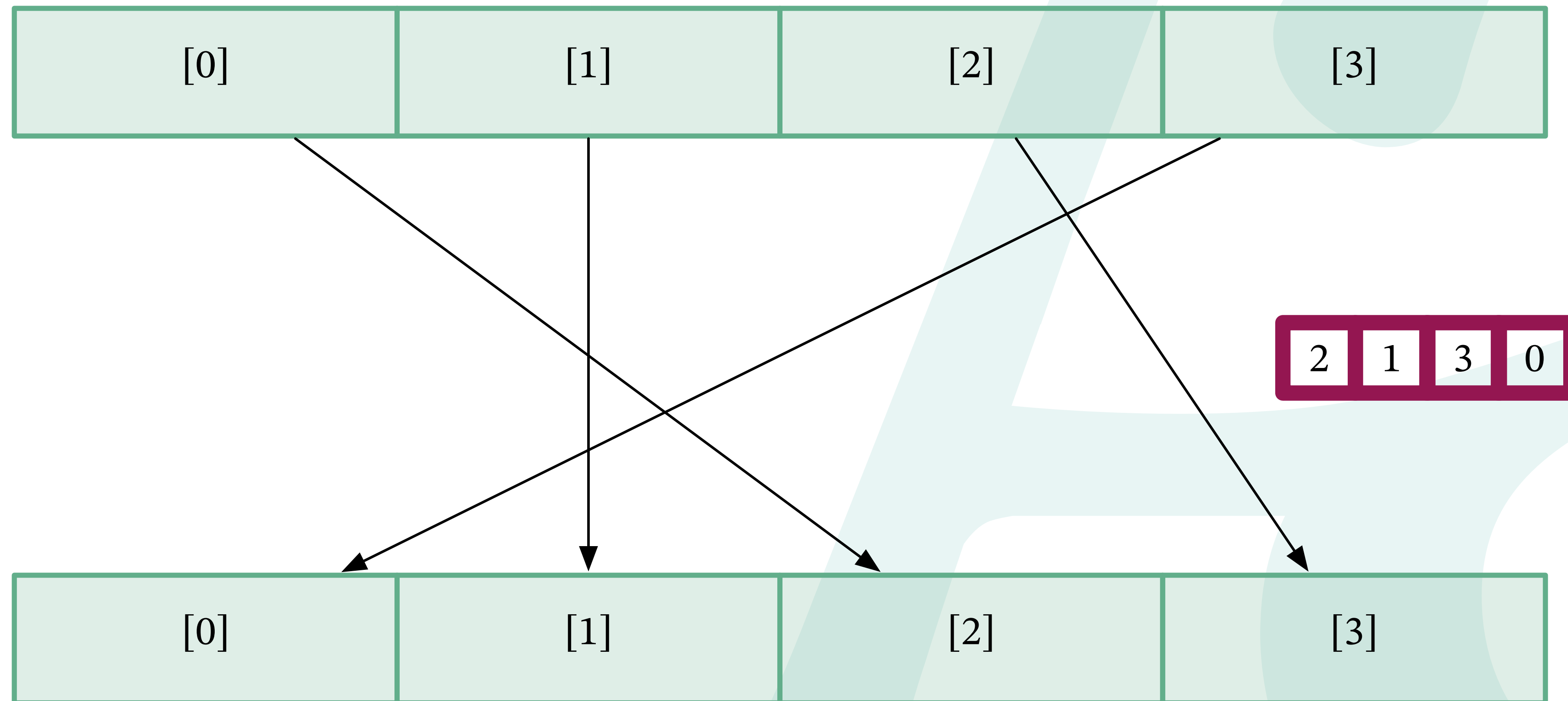
VectorMath

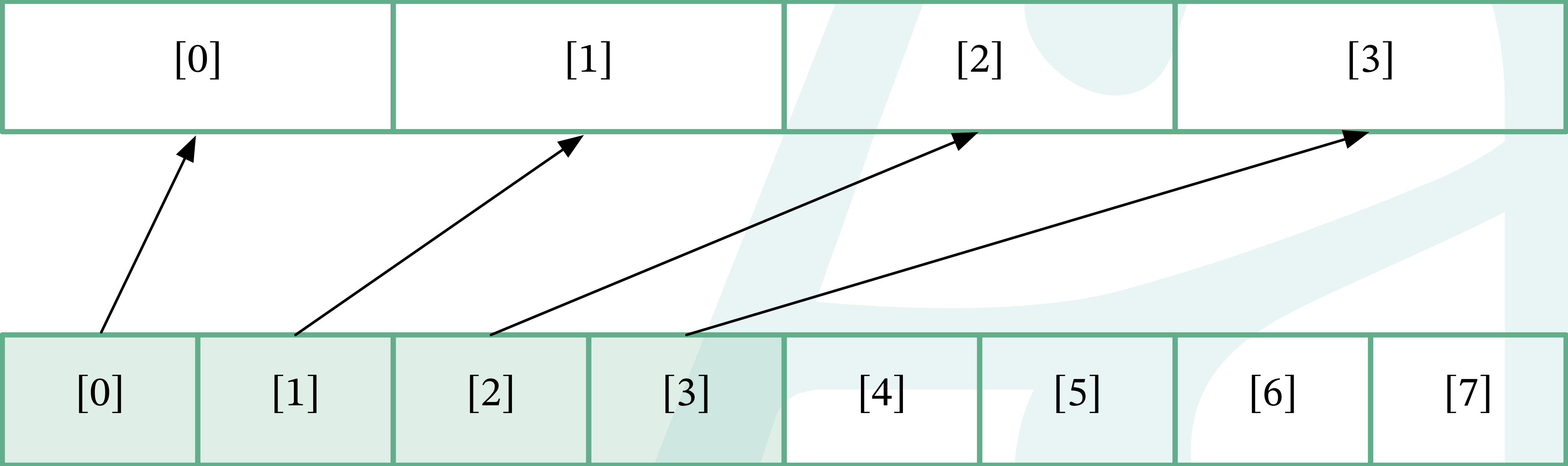
Saturierte Arithmetik

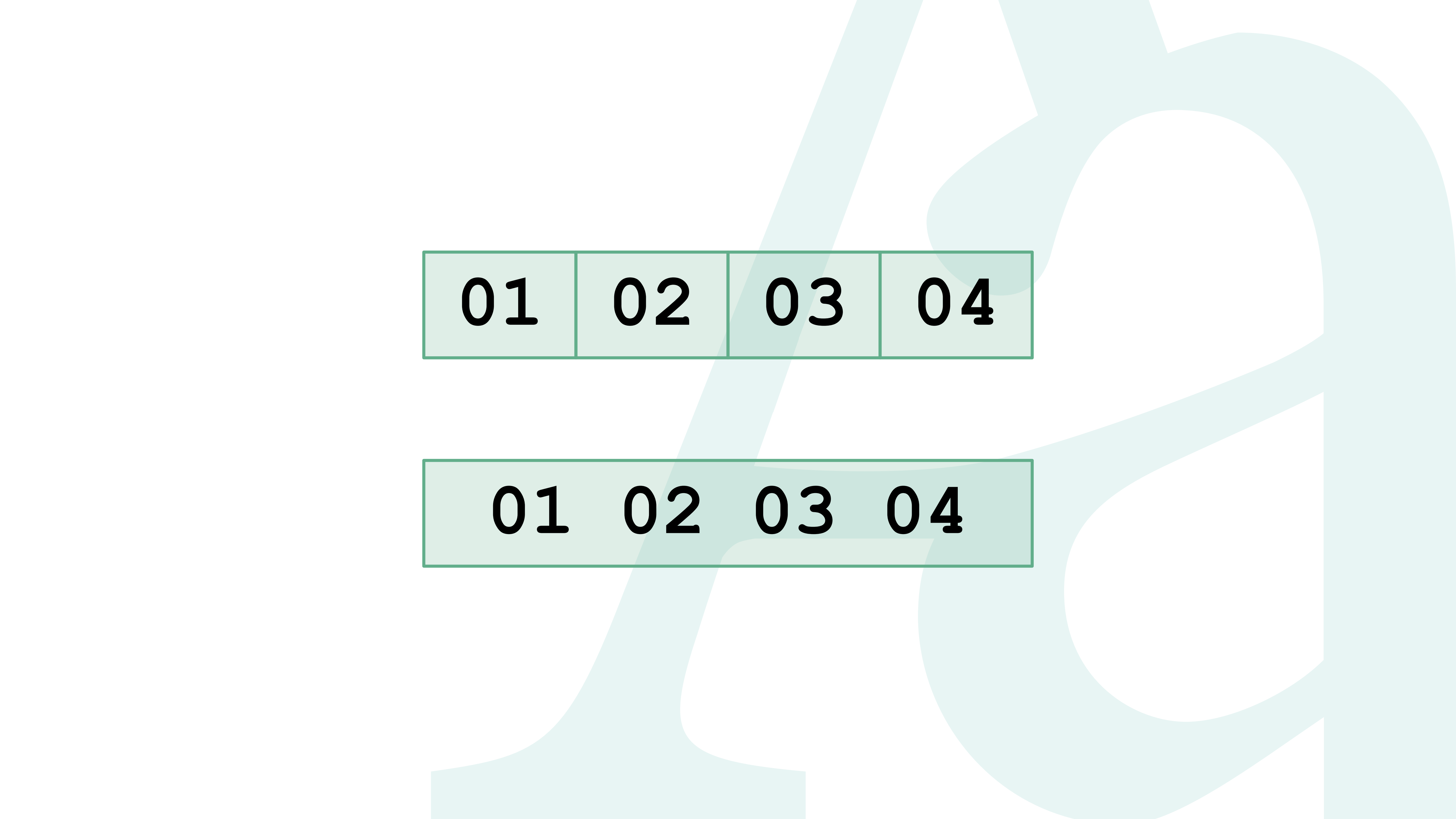
- **static** byte addSaturating(byte a , byte b)
- **static** int addSaturating(int a , int b)
- **static** long addSaturating(long a , long b)
- **static** short addSaturating(short a , short b)
- **static** byte addSaturatingUnsigned(byte a , byte b)
- **static** int addSaturatingUnsigned(int a , int b)
- **static** long addSaturatingUnsigned(long a , long b)
- **static** short addSaturatingUnsigned(short a , short b)
- **static** byte maxUnsigned(byte a , byte b)
- **static** int maxUnsigned(int a , int b)
- **static** long maxUnsigned(long a , long b)
- **static** short maxUnsigned(short a , short b)
- **static** byte minUnsigned(byte a , byte b)
- **static** int minUnsigned(int a , int b)
- **static** long minUnsigned(long a , long b)
- **static** short minUnsigned(short a , short b)
- **static** byte subSaturating(byte a , byte b)
- **static** int subSaturating(int a , int b)
- **static** long subSaturating(long a , long b)
- **static** short subSaturating(short a , short b)
- **static** byte subSaturatingUnsigned(byte a , byte b)
- **static** int subSaturatingUnsigned(int a , int b)
- **static** long subSaturatingUnsigned(long a , long b)
- **static** short subSaturatingUnsigned(short a , short b)



VectorShuffle

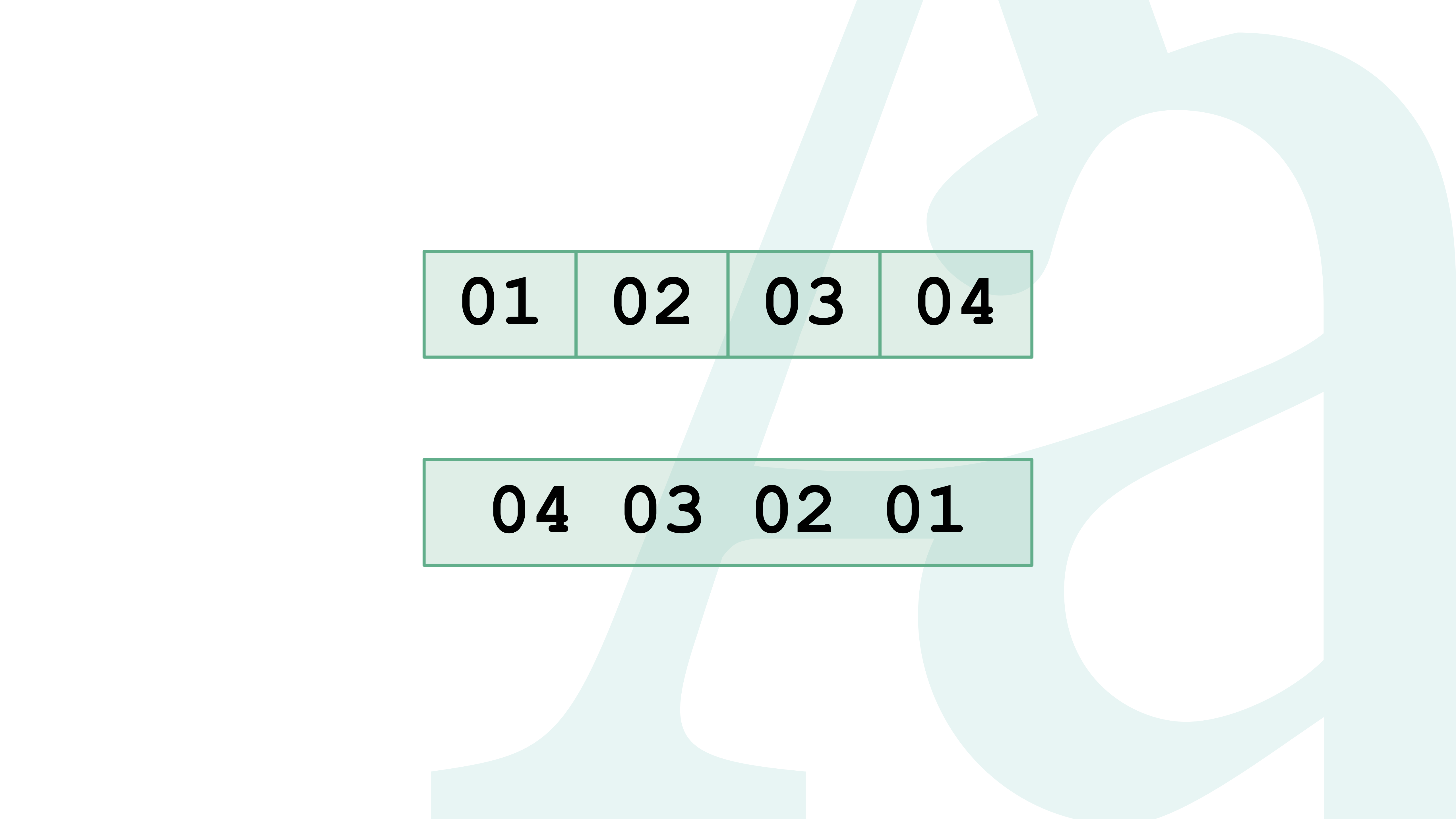






01	02	03	04
----	----	----	----

01	02	03	04
----	----	----	----



01	02	03	04
----	----	----	----

04	03	02	01
----	----	----	----

Float16 [S EEEEE MMMMMMMMMM]

```
var a = Float16[n];
```

```
var b = Float16[n];
```

```
var c = Float16[n];
```

```
for (int i = 0; i < n; i++) {
```

```
    result[i] = Float16.add(a[i], b[i]);
```

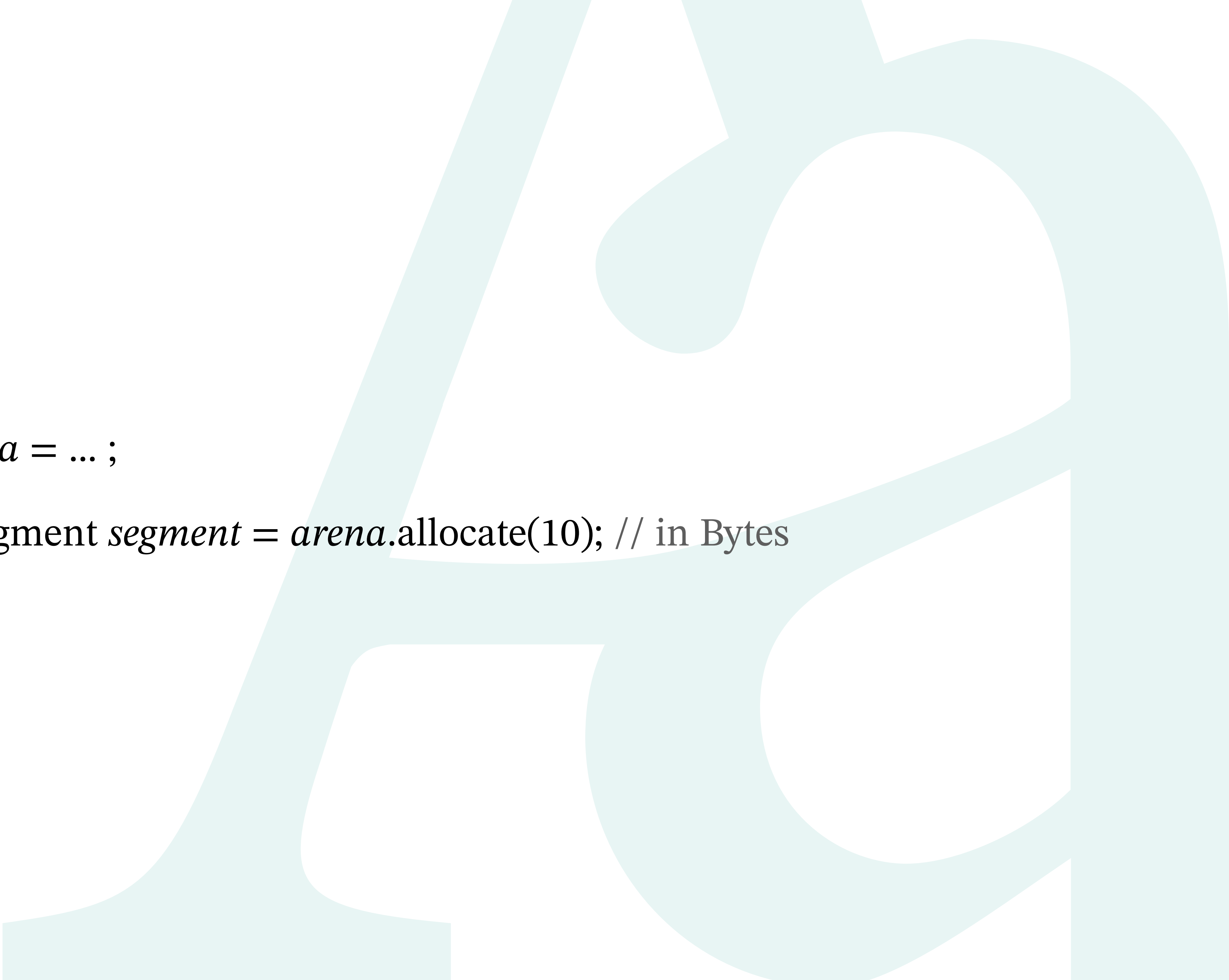
```
}
```



Speicher

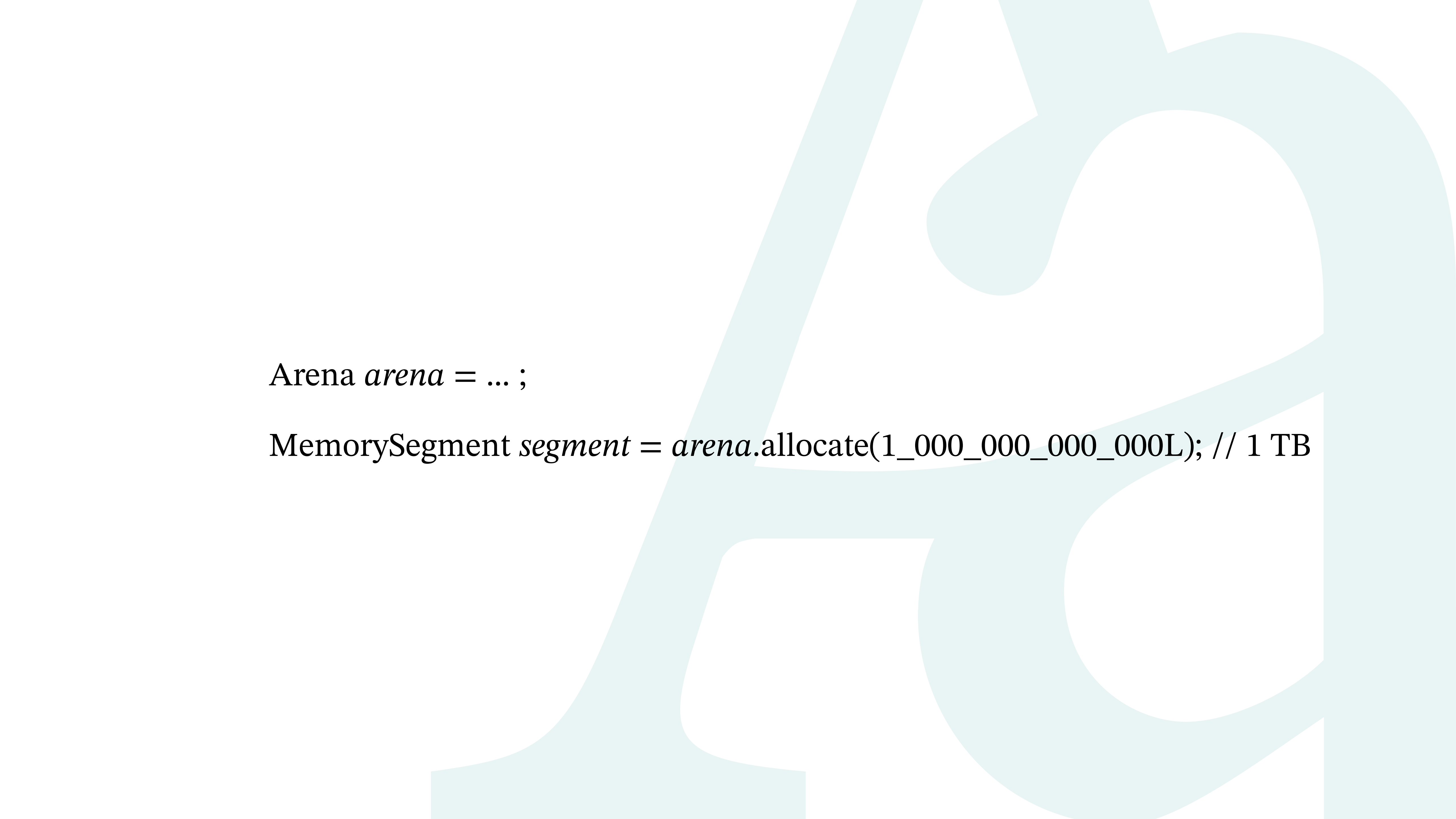
Was ist eigentlich das Problem?

- Die Anzahl der zu dereferenzierenden Bytes
- Die Einschränkungen bei der Adresse, über die die Dereferenzierung erfolgt
- Die Byte-Ordnung des betroffenen Bereichs
- Der Java-Typ, der bei der Dereferenzierung geliefert werden soll (int vs. float)



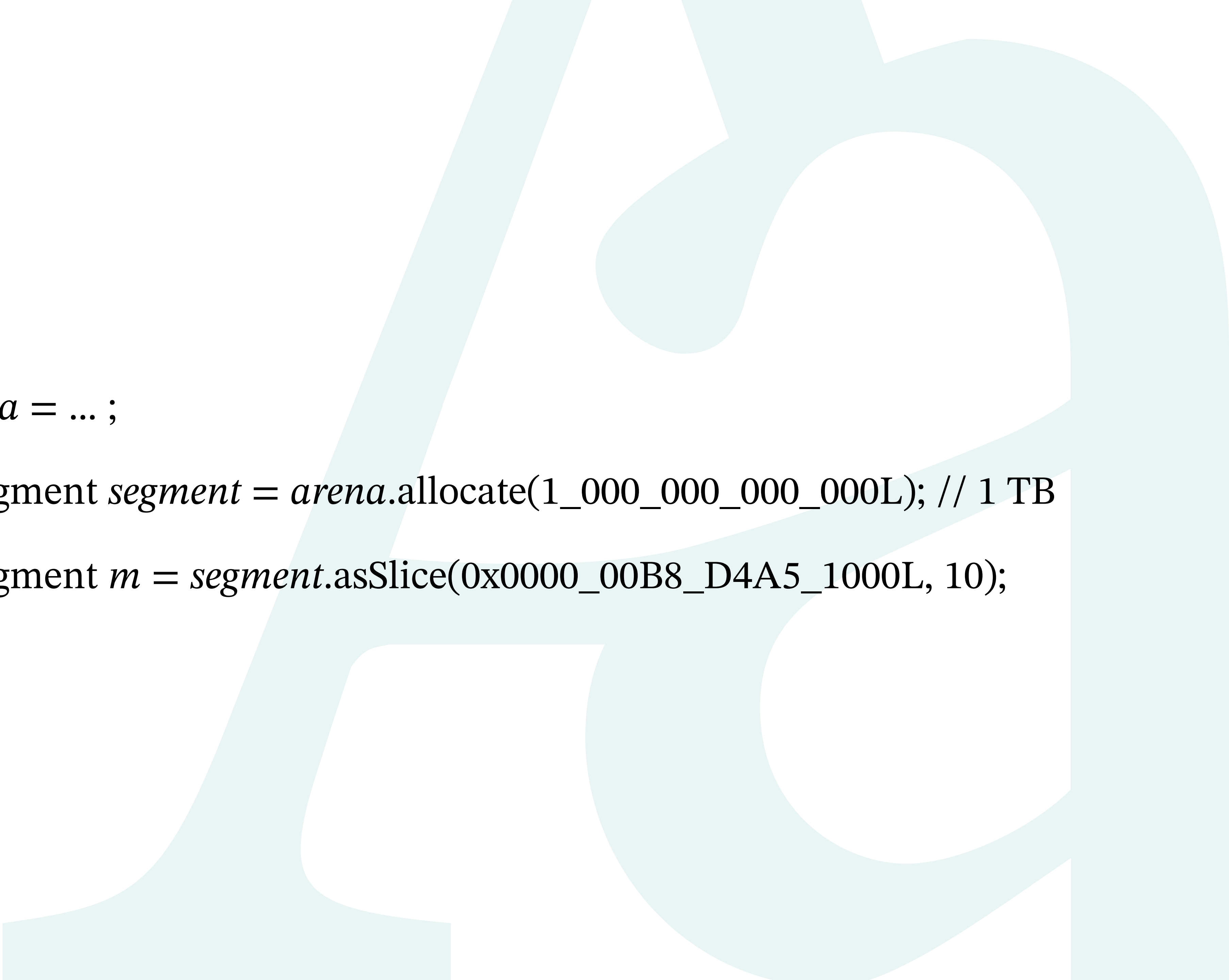
Arena *arena* = ... ;

MemorySegment *segment* = *arena*.allocate(10); // in Bytes



Arena *arena* = ... ;

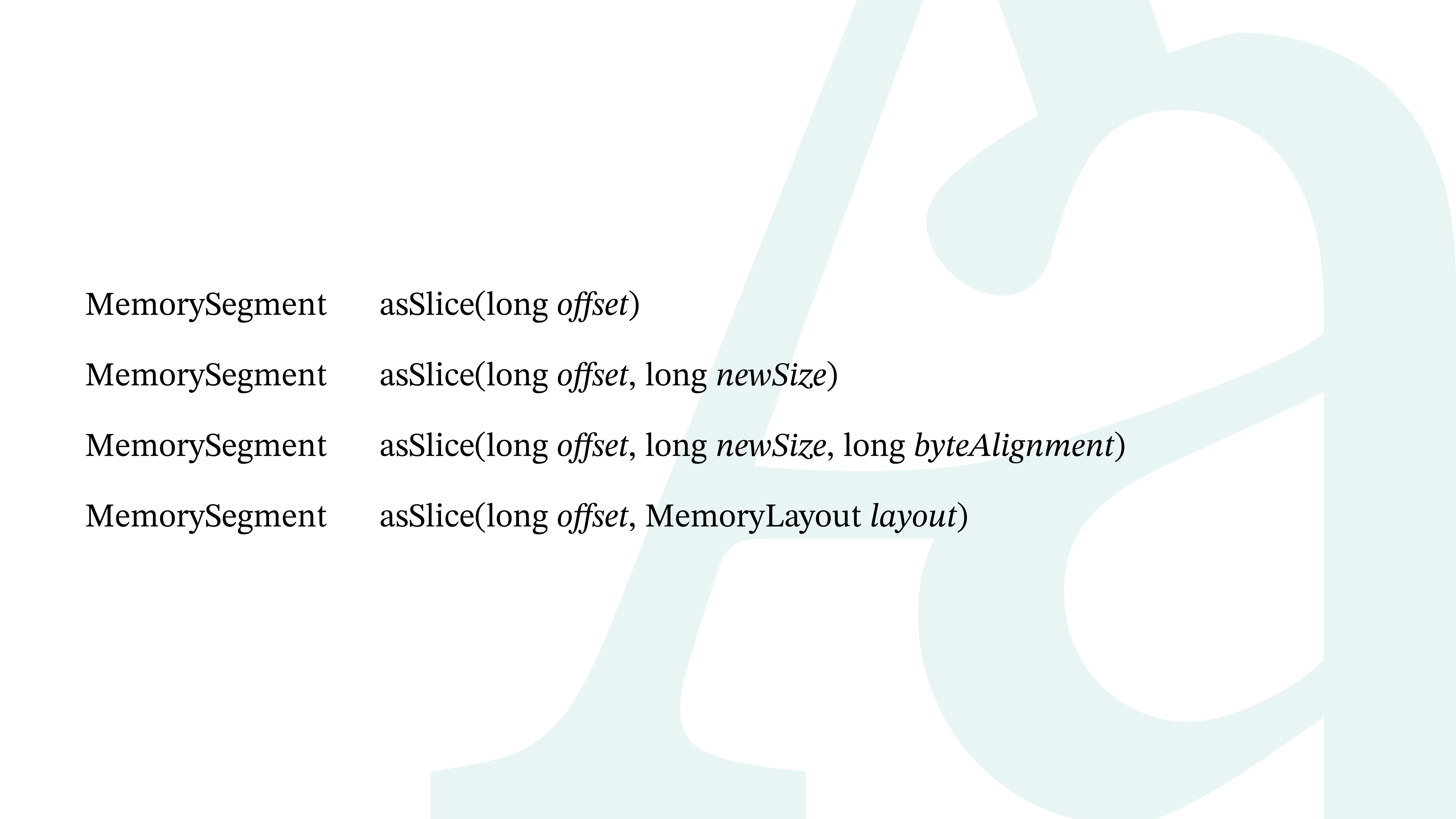
MemorySegment *segment* = *arena*.allocate(1_000_000_000_000L); // 1 TB



Arena *arena* = ... ;

MemorySegment *segment* = *arena*.allocate(1_000_000_000_000L); // 1 TB

MemorySegment *m* = *segment*.asSlice(0x0000_00B8_D4A5_1000L, 10);



MemorySegment asSlice(long *offset*)

MemorySegment asSlice(long *offset*, long *newSize*)

MemorySegment asSlice(long *offset*, long *newSize*, long *byteAlignment*)

MemorySegment asSlice(long *offset*, MemoryLayout *layout*)

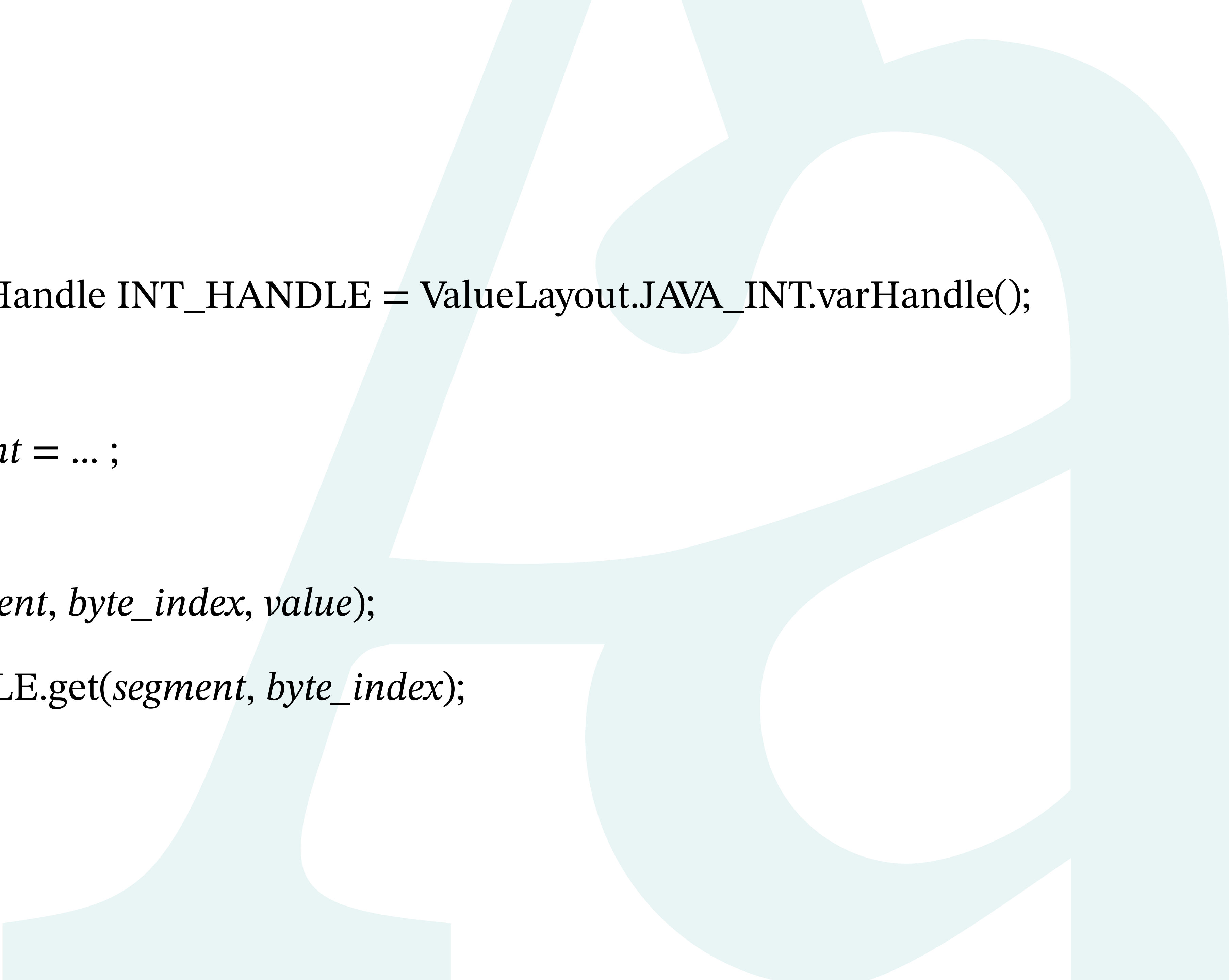


Speicherzugriff

00 3BFF0F	00 3BFF0E	00 3BFF0D	00 3BFF0C
00 3BFF0B	00 3BFF0A	07 3BFF09	D0 3BFF08
00 3BFF07	00 3BFF06	03 3BFF05	E8 3BFF04
00 3BFF03	00 3BFF02	00 3BFF01	00 3BFF00



```
public static final VarHandle INT_HANDLE = ValueLayout.JAVA_INT.varHandle();
```



```
public static final VarHandle INT_HANDLE = ValueLayout.JAVA_INT.varHandle();
```

```
MemorySegment segment = ... ;
```

```
INT_HANDLE.set(segment, byte_index, value);
```

```
int value = INT_HANDLE.get(segment, byte_index);
```

public static final VarHandle INT_HANDLE = ValueLayout.JAVA_INT.varHandle();

MemorySegment *segment* = ... ;

INT_HANDLE.set(*segment*, 4, 1000);

assert INT_HANDLE.get(*segment*, 4) = 1000;

00 3BFF0F	00 3BFF0E	00 3BFF0D	00 3BFF0C
00 3BFF0B	00 3BFF0A	07 3BFF09	D0 3BFF08
00 3BFF07	00 3BFF06	03 3BFF05	E8 3BFF04
00 3BFF03	00 3BFF02	00 3BFF01	00 3BFF00

```
public static final VarHandle INT_HANDLE = ValueLayout.JAVA_INT.varHandle();  
public static final VarHandle FLOAT_HANDLE = ValueLayout.JAVA_FLOAT.varHandle();
```

```
MemorySegment segment = ... ;
```

```
INT_HANDLE.set(segment, 4, 1000);
```

```
assert FLOAT_HANDLE.get(segment, 4) = 1.401E-42;
```

00 3BFF0F	00 3BFF0E	00 3BFF0D	00 3BFF0C
00 3BFF0B	00 3BFF0A	07 3BFF09	D0 3BFF08
00 3BFF07	00 3BFF06	03 3BFF05	E8 3BFF04
00 3BFF03	00 3BFF02	00 3BFF01	00 3BFF00

00 3BFF0F	00 3BFF0E	00 3BFF0D	00 3BFF0C
00 3BFF0B	00 3BFF0A	07 3BFF09	D0 3BFF08
00 3BFF07	00 3BFF06	03 3BFF05	E8 3BFF04
00 3BFF03	00 3BFF02	00 3BFF01	00 3BFF00

```
INT_HANDLE.set(segment, 4, 1000);
```

```
assert INT_HANDLE.get(segment, 5) throws IllegalArgumentException;
```

```
java.lang.IllegalArgumentException: Target offset 5 is incompatible with alignment  
constraint 4 (of i4) for segment MemorySegment ...
```

```
at jdk.internal.foreign.AbstractMemorySegmentImpl.checkEnclosingLayout()
```

```
at java.lang.invoke.VarHandleSegmentAsInts.checkSegment()
```

```
at java.lang.invoke.VarHandleSegmentAsInts.get()
```

Typ	Layout	Größe	Alignment
boolean	JAVA_BOOLEAN	1	1
byte	JAVA_BYTE	1	1
short	JAVA_SHORT	2	2
int	JAVA_INT	4	4
long	JAVA_LONG	8	8
float	JAVA_FLOAT	4	4
double	JAVA_DOUBLE	8	8
char	JAVA_CHAR	2	2
—	ADDRESS	4/8	4/8

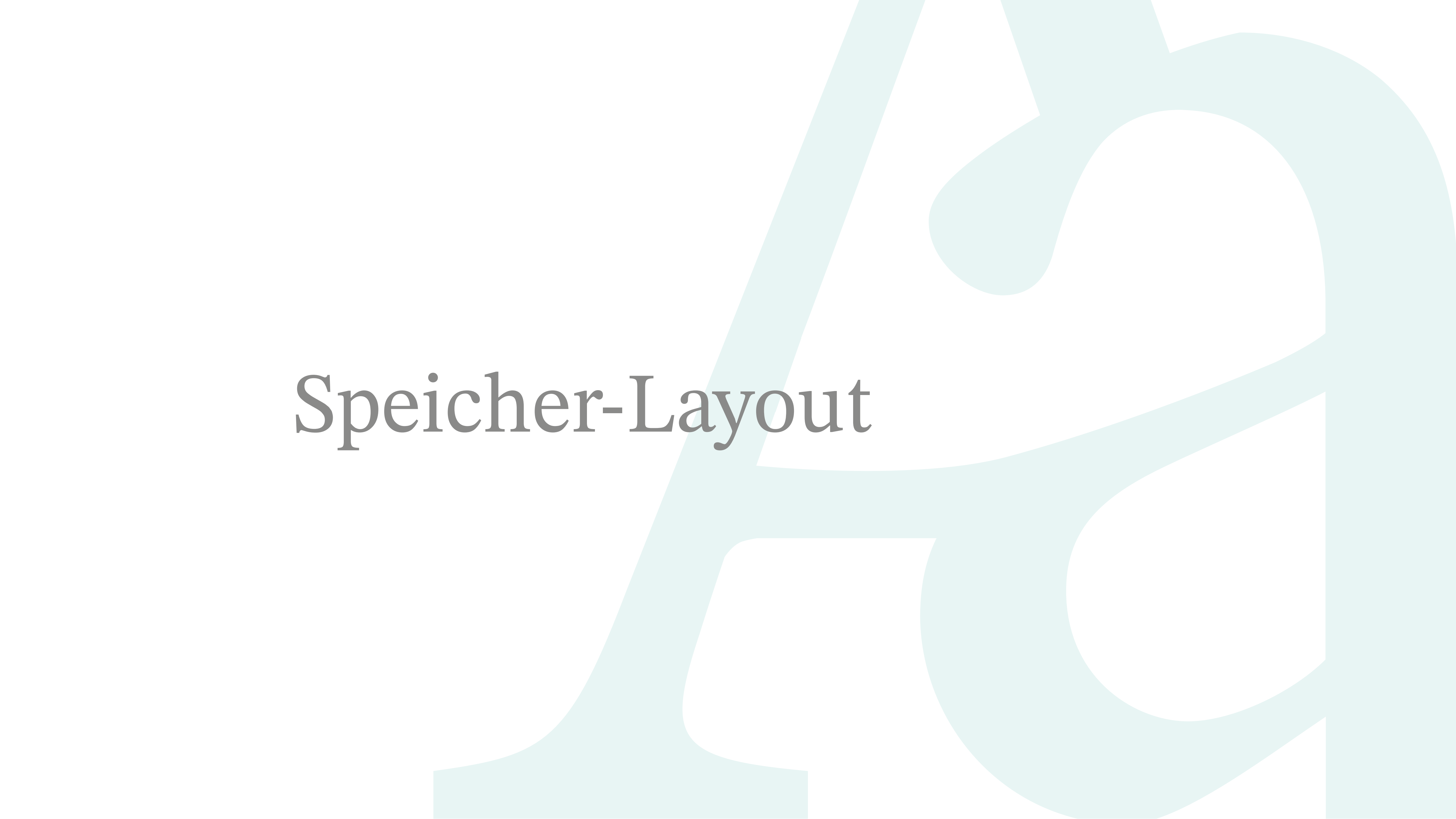
Typ	Layout	Größe	Alignment
boolean	JAVA_BOOLEAN_UNALIGNED	1	1
byte	JAVA_BYTE_UNALIGNED	1	1
short	JAVA_SHORT_UNALIGNED	2	1
int	JAVA_INT_UNALIGNED	4	1
long	JAVA_LONG_UNALIGNED	8	1
float	JAVA_FLOAT_UNALIGNED	4	1
double	JAVA_DOUBLE_UNALIGNED	8	1
char	JAVA_CHAR_UNALIGNED	2	1
—	ADDRESS_UNALIGNED	4/8	1

```
public static final VarHandle INT_HANDLE = (  
    ValueLayout.JAVA_INT.varHandle()  
);  
  
public static final VarHandle UNALIGNED_INT_HANDLE = (  
    ValueLayout.JAVA_INT_UNALIGNED.varHandle()  
    // ValueLayout.JAVA_INT.withByteAlignment(1).varHandle()  
);  
  
INT_HANDLE.set(segment, 4, 1000);  
INT_HANDLE.set(segment, 8, 2000);  
assert UNALIGNED_INT_HANDLE.get(segment, 5) = -805_306_365;
```

00 3BFF0F	00 3BFF0E	00 3BFF0D	00 3BFF0C
00 3BFF0B	00 3BFF0A	07 3BFF09	D0 3BFF08
00 3BFF07	00 3BFF06	03 3BFF05	E8 3BFF04
00 3BFF03	00 3BFF02	00 3BFF01	00 3BFF00

00 3BFF0F	00 3BFF0E	00 3BFF0D	00 3BFF0C
00 3BFF0B	00 3BFF0A	00 3BFF09	00 3BFF08
01 3BFF07	23 3BFF06	45 3BFF05	67 3BFF04
00 3BFF03	00 3BFF02	00 3BFF01	00 3BFF00

```
public static final VarHandle BIG_INT_HANDLE = (  
    ValueLayout.JAVA_INT.withOrder(ByteOrder.BIG_ENDIAN).varHandle()  
);  
  
public static final VarHandle LITTLE_INT_HANDLE = (  
    ValueLayout.JAVA_INT.withOrder(ByteOrder.LITTLE_ENDIAN).varHandle()  
);  
  
BIG_INT_HANDLE.set(segment, 4, 0x67_45_23_01);  
  
assert LITTLE_INT_HANDLE.get(segment, 4) = 0x01_23_45_67;
```



Speicher-Layout



```
typedef struct {
```

```
    int x;
```

```
    int y;
```

```
} Point;
```



```
int size = ... ;
```

```
Point* points = (Point*) malloc(sizeof(Point) * size);
```

```
for (int i = 0 ; i < size ; i++) {
```

```
    ... points[i].x ... points[i].y ...
```

```
}
```



MemoryLayout

PaddingLayout

ValueLayout

SequenceLayout

GroupLayout

StructLayout

UnionLayout

MemoryLayout

ValueLayout

ValueLayout.OfBoolean

ValueLayout.OfByte

ValueLayout.OfChar

ValueLayout.OfShort

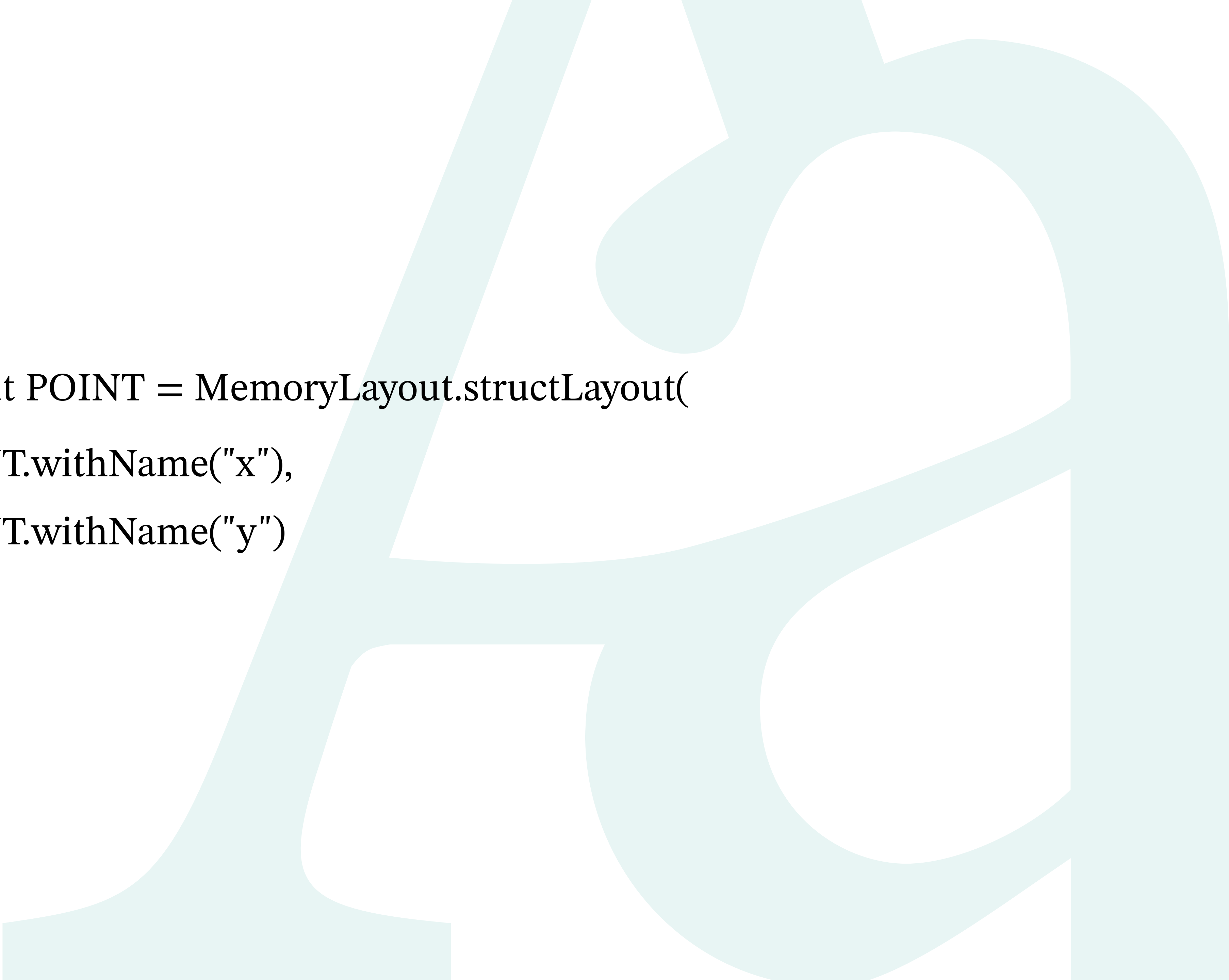
ValueLayout.OfInt

ValueLayout.OfFloat

ValueLayout.OfLong

ValueLayout.OfDouble

AddressLayout



```
static final StructLayout POINT = MemoryLayout.structLayout(  
    ValueLayout.JAVA_INT.withName("x"),  
    ValueLayout.JAVA_INT.withName("y")  
);
```

```
static final VarHandle POINT_ARR_X = (  
    POINT.arrayElementVarHandle(PathElement.groupElement("x"))  
);
```

```
static final VarHandle POINT_ARR_Y = (  
    POINT.arrayElementVarHandle(PathElement.groupElement("y"))  
);
```

```
int size = ... ;
```

```
MemorySegment points = ... ;
```

```
for (int i = 0; i < size; i++) {  
  
    var x = (int) POINT_ARR_X.get(points, 0L, (long) i);  
  
    var y = (int) POINT_ARR_Y.get(points, 0L, (long) i);  
  
    var p = Point(x, y);  
  
    plotPoint(p)  
}
```

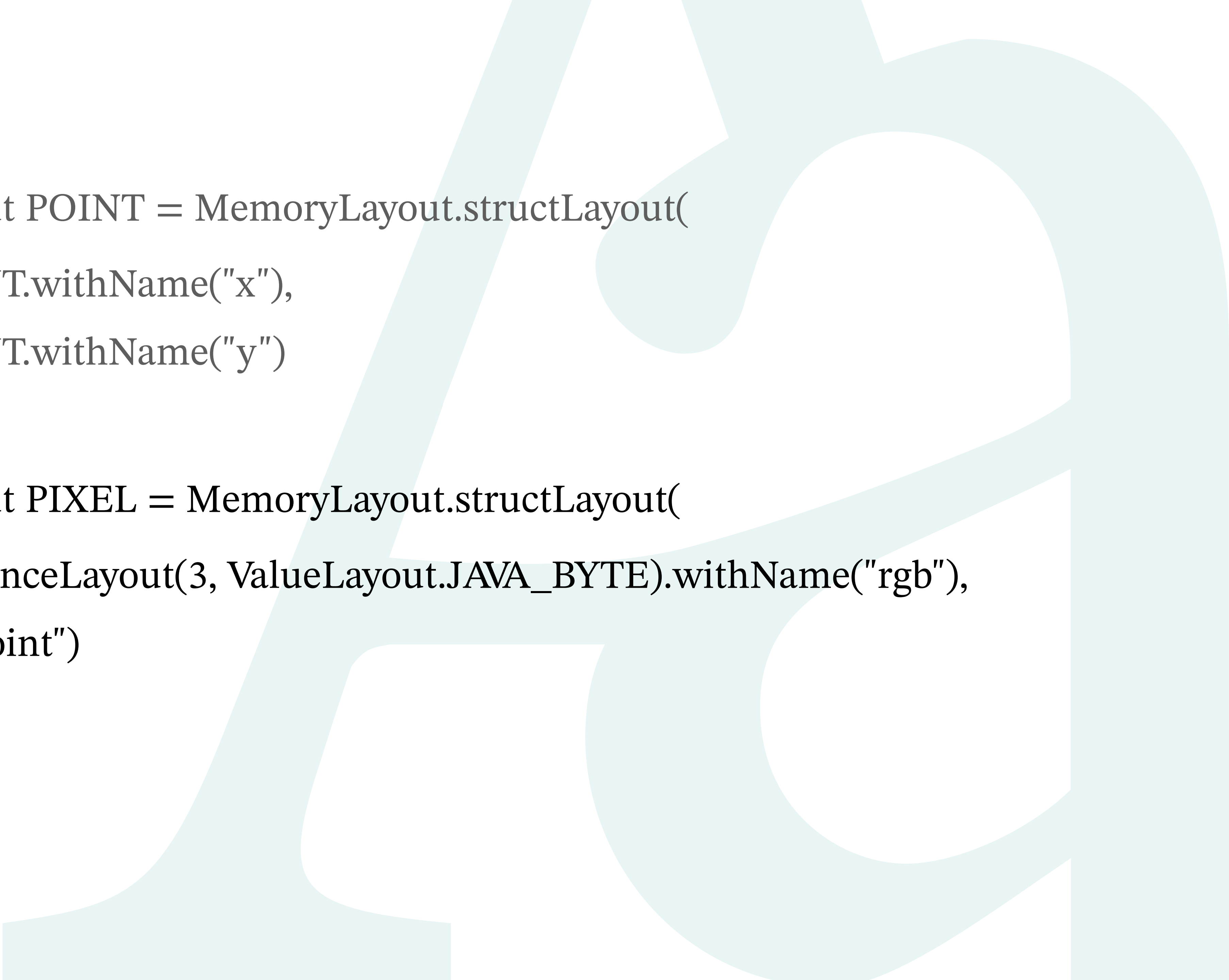


```
typedef struct {
```

```
    char rgb[3];
```

```
    Point point;
```

```
} Pixel;
```



```
static final StructLayout POINT = MemoryLayout.structLayout(  
    ValueLayout.JAVA_INT.withName("x"),  
    ValueLayout.JAVA_INT.withName("y")  
);
```

```
static final StructLayout PIXEL = MemoryLayout.structLayout(  
    MemoryLayout.sequenceLayout(3, ValueLayout.JAVA_BYTE).withName("rgb"),  
    POINT.withName("point")  
);
```

```
static final StructLayout POINT = MemoryLayout.structLayout(  
    ValueLayout.JAVA_INT.withName("x"),  
    ValueLayout.JAVA_INT.withName("y")  
);
```

```
static final StructLayout PIXEL = MemoryLayout.structLayout(  
    MemoryLayout.sequenceLayout(3, ValueLayout.JAVA_BYTE).withName("rgb"),  
    MemoryLayout.paddingLayout(1),  
    POINT.withName("point")  
);
```

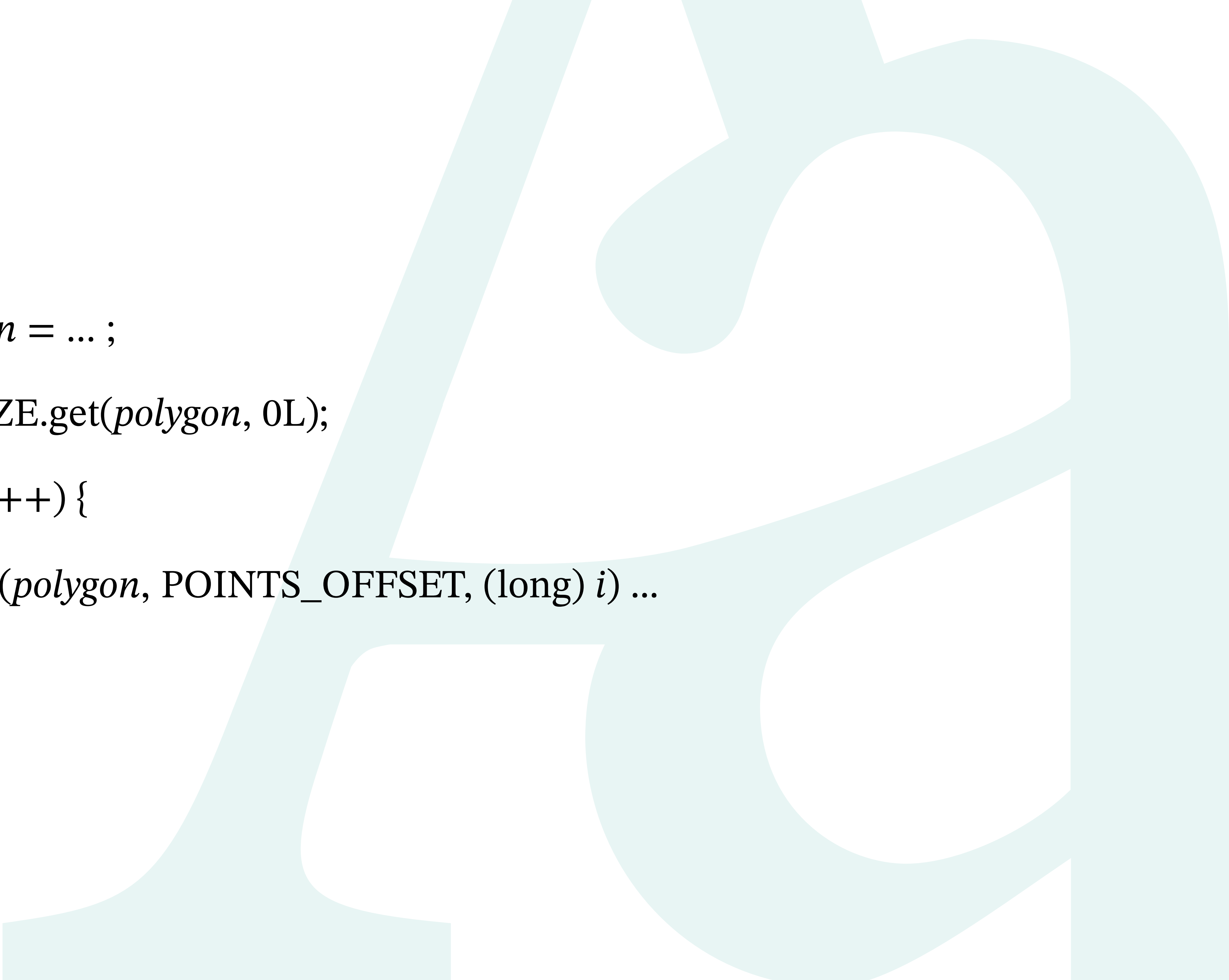


```
typedef struct {  
    int size;  
    Point points[];  
} Polygon;
```

```
private static final StructLayout POLYGON = MemoryLayout.structLayout(  
    ValueLayout.JAVA_INT.withName("size"),  
    MemoryLayout.sequenceLayout(0, POINT).withName("points")  
);
```

```
private static final VarHandle POLYGON_SIZE = POLYGON.varHandle(  
    0,  
    PathElement.groupElement("size")  
);
```

```
private static final long POINTS_OFFSET = (  
    POLYGON.byteOffset(PathElement.groupElement("points"));  
)
```



```
MemorySegment polygon = ... ;  
  
int size = POLYGON_SIZE.get(polygon, 0L);  
  
for (int i = 0 ; i < size ; i++) {  
    ... POINT_ARR_X.get(polygon, POINTS_OFFSET, (long) i) ...  
}
```



Speicher-Allokation



```
MemorySegment segment = Arena.global().allocate(100); // 100 Byte.
```

```
// segment wird nie freigegeben.
```



```
{  
    MemorySegment segment = Arena.ofAuto().allocate(100); // 100 Byte.  
    ...  
}  
  
// segment wird (irgendwann) vom Garbage Collector freigegeben.
```

```
MemorySegment segment = null;  
try (Arena arena = Arena.ofConfined()) {  
    segment = arena.allocate(100);  
    ...  
}  
  
// segment wird mit dem Schließen der arena freigegeben.  
segment.get(ValueLayout.JAVA_BYTE, 0); // throws IllegalStateException
```

```
MemorySegment segment = null;  
try (Arena arena = Arena.ofShared()) {  
    segment = arena.allocate(100);  
    var t = Thread.ofPlatform().start(() -> {... segment ...});  
    ... segment ... ;  
    t.join();  
}  
  
// segment wird mit dem Schließen der arena freigegeben.  
segment.get(ValueLayout.JAVA_BYTE, 0); // throws IllegalStateException
```

Arena

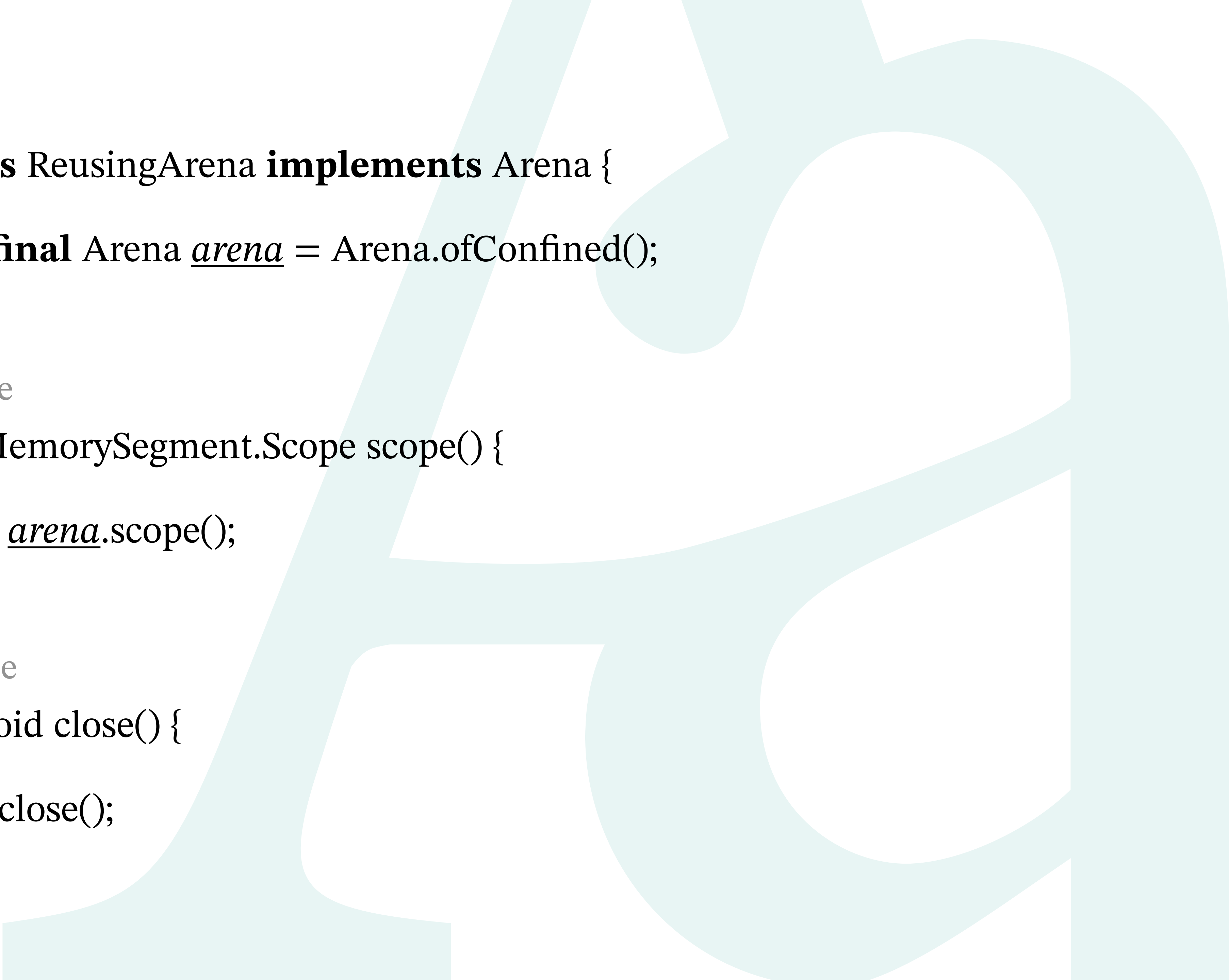
Art	Endliche Lebenszeit	Explizite Freigabe	Mehrere Threads
global	nein	nein	ja
automatic	ja	nein	ja
confined	ja	ja	nein
shared	ja	ja	ja

A large, light blue, stylized letter 'R' serves as the background for the entire image. It is positioned on the right side, with its vertical stroke extending from the top to the bottom of the frame. The letter has a thick, rounded, and somewhat irregular appearance, giving it a hand-drawn or artistic feel.

Eigene Arena

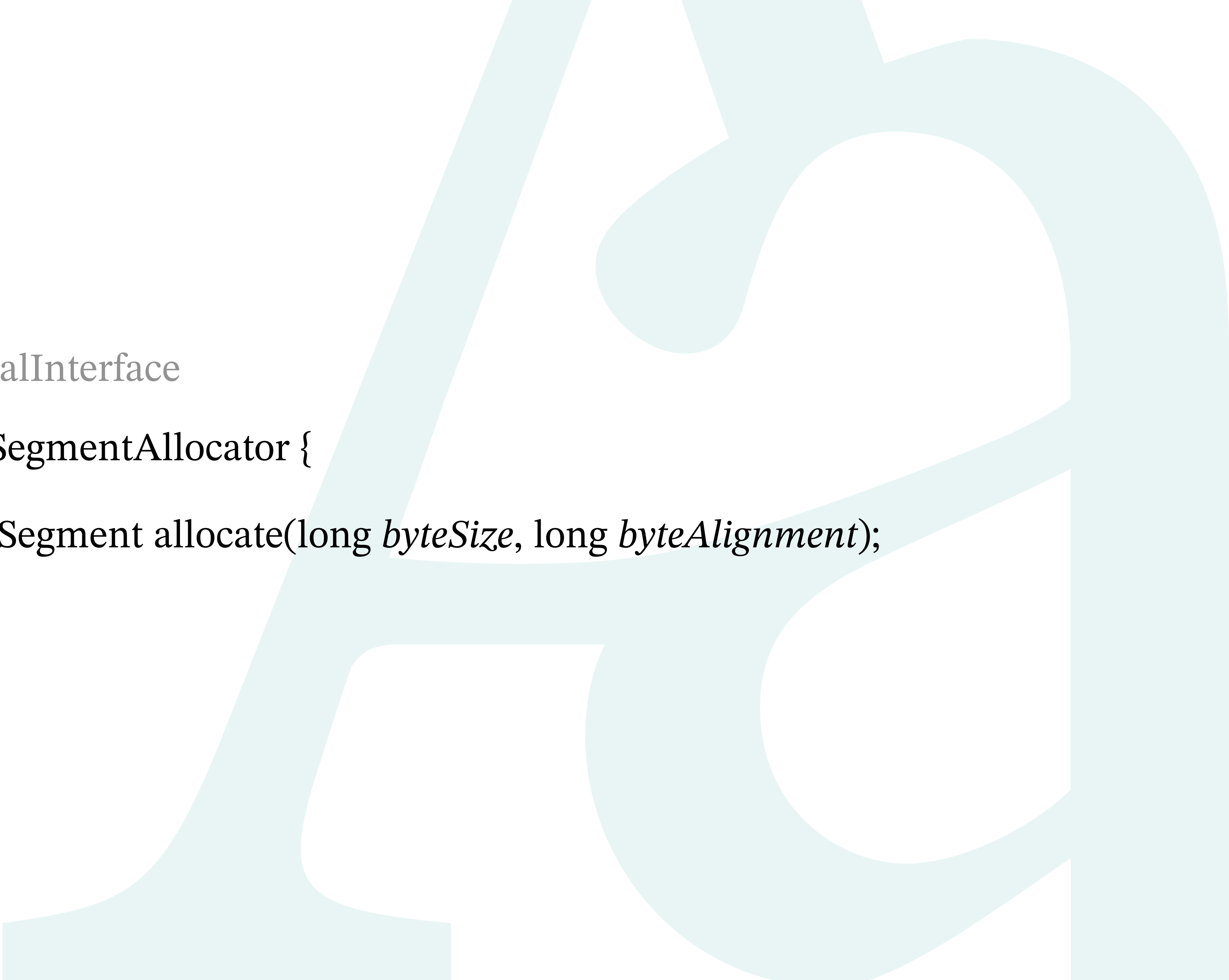
```
public class SlicingArena implements Arena {  
    private final Arena arena = Arena.ofConfined();  
    ...  
    @Override  
    public MemorySegment.Scope scope() {  
        return arena.scope();  
    }  
    @Override  
    public void close() {  
        arena.close();  
    }  
}
```

```
private final SegmentAllocator slicingAllocator;  
public SlicingArena(long size) {  
    slicingAllocator = SegmentAllocator.slicingAllocator(arena.allocate(size));  
}  
  
@Override  
public MemorySegment allocate(long byteSize, long byteAlignment) {  
    return slicingAllocator.allocate(byteSize, byteAlignment);  
}
```



```
public class ReusingArena implements Arena {  
    private final Arena arena = Arena.ofConfined();  
    ...  
    @Override  
    public MemorySegment.Scope scope() {  
        return arena.scope();  
    }  
    @Override  
    public void close() {  
        arena.close();  
    }  
}
```

```
private final MemorySegment reusableSegment;  
private final int maxAlignment;  
public ReusingArena(long size, int maxAlignment) {  
    this.maxAlignment = maxAlignment;  
    this.reusableSegment = arena.allocate(size, maxAlignment);  
}  
  
@Override  
public MemorySegment allocate(long byteSize, long byteAlignment) {  
    assert byteAlignment <= maxAlignment;  
    return reusableSegment.asSlice(byteSize, byteAlignment);  
}
```



```
@FunctionalInterface
```

```
interface SegmentAllocator {
```

```
    MemorySegment allocate(long byteSize, long byteAlignment);
```

```
}
```

default MemorySegment allocate(long *byteSize*)

default MemorySegment allocate(MemoryLayout *layout*)

default MemorySegment allocate(MemoryLayout *elementLayout*, long *count*)

default MemorySegment allocateFrom(AddressLayout *layout*, MemorySegment *value*)

default MemorySegment allocateFrom(ValueLayout.OfByte *layout*, byte *value*)

default MemorySegment allocateFrom(ValueLayout.OfByte *elementLayout*, byte... *elements*)

default MemorySegment allocateFrom(ValueLayout.OfChar *layout*, char *value*)

default MemorySegment allocateFrom(ValueLayout.OfChar *elementLayout*, char... *elements*)

default MemorySegment allocateFrom(ValueLayout.OfDouble *layout*, double *value*)

default MemorySegment allocateFrom(ValueLayout.OfDouble *elementLayout*, double... *elements*)

default MemorySegment allocateFrom(ValueLayout.OfFloat *layout*, float *value*)

default MemorySegment allocateFrom(ValueLayout.OfFloat *elementLayout*, float... *elements*)

default MemorySegment allocateFrom(ValueLayout.OfInt *layout*, int *value*)

default MemorySegment allocateFrom(ValueLayout.OfInt *elementLayout*, int... *elements*)

default MemorySegment allocateFrom(ValueLayout.OfLong *layout*, long *value*)

default MemorySegment allocateFrom(ValueLayout.OfLong *elementLayout*, long... *elements*)

default MemorySegment allocateFrom(ValueLayout.OfShort *layout*, short *value*)

default MemorySegment allocateFrom(ValueLayout.OfShort *elementLayout*, short... *elements*)

default MemorySegment allocateFrom(String *str*)

default MemorySegment allocateFrom(String *str*, Charset *charset*)

default MemorySegment allocate(long *byteSize*)

default MemorySegment allocate(MemoryLayout *layout*)

default MemorySegment allocate(MemoryLayout *elementLayout*, long *count*)

default MemorySegment allocateFrom(AddressLayout *layout*, MemorySegment *value*)

default MemorySegment allocateFrom(ValueLayout.OfInt *layout*, int *value*)

default MemorySegment allocateFrom(ValueLayout.OfInt *elementLayout*, int... *elements*)

default MemorySegment allocateFrom(String *str*)

default MemorySegment allocateFrom(String *str*, Charset *charset*)



Beste Integration

static MemorySegment ofAddress(long *address*)

static MemorySegment ofArray(byte[] *byteArray*)

static MemorySegment ofArray(char[] *charArray*)

static MemorySegment ofArray(double[] *doubleArray*)

static MemorySegment ofArray(float[] *floatArray*)

static MemorySegment ofArray(int[] *intArray*)

static MemorySegment ofArray(long[] *longArray*)

static MemorySegment ofArray(short[] *shortArray*)

static MemorySegment ofBuffer(Buffer *buffer*)

Splitter<MemorySegment> spliterator(MemoryLayout *elementLayout*)

byte[] toArray(ValueLayout.OfByte *elementLayout*)

char[] toArray(ValueLayout.OfChar *elementLayout*)

double[] toArray(ValueLayout.OfDouble *elementLayout*)

float[] toArray(ValueLayout.OfFloat *elementLayout*)

int[] toArray(ValueLayout.OfInt *elementLayout*)

long[] toArray(ValueLayout.OfLong *elementLayout*)

short[] toArray(ValueLayout.OfShort *elementLayout*)

ByteBuffer asByteBuffer()

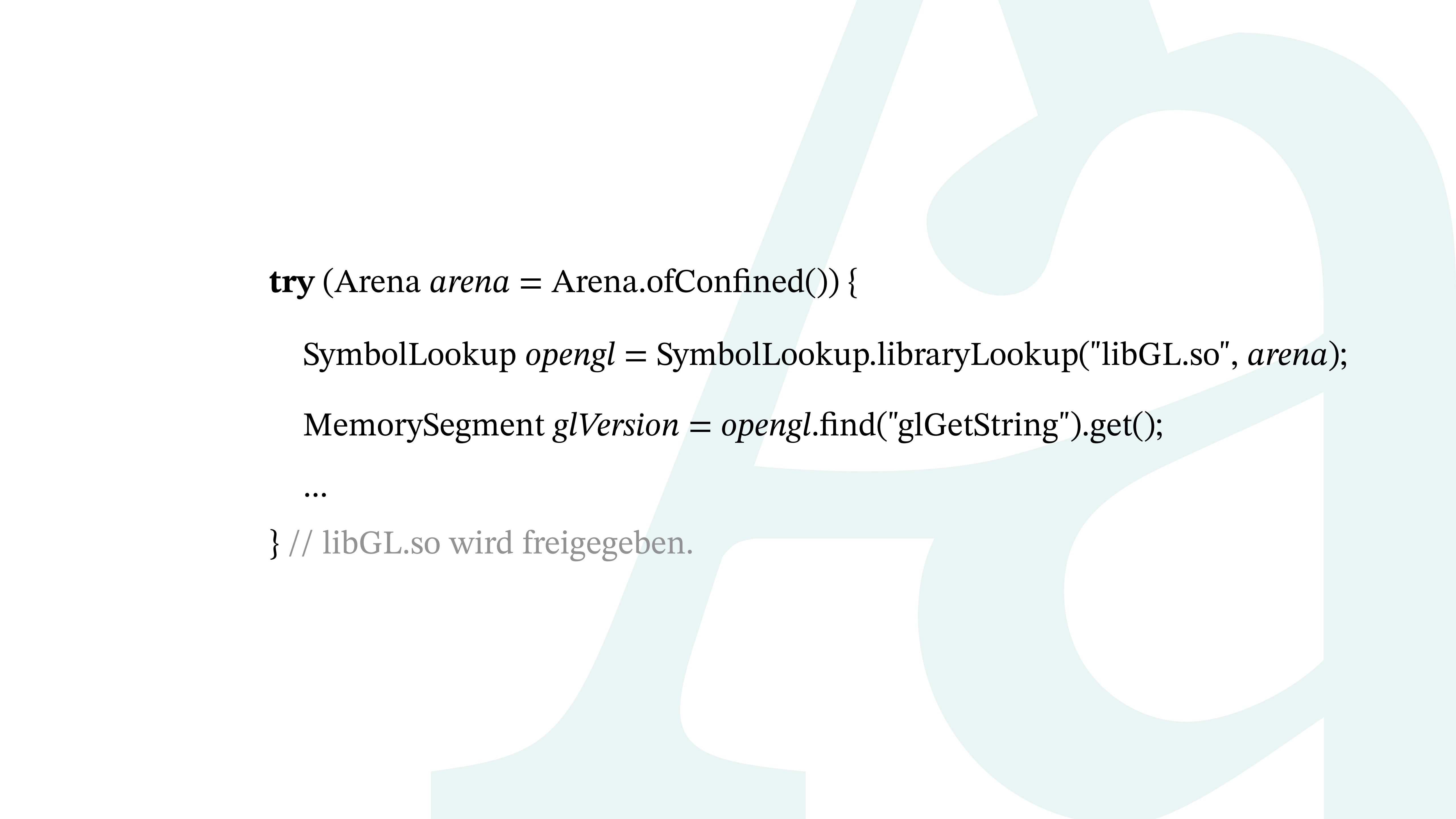
```
try (  
    var fc = FileChannel.open(Path.of("points"), Set.of(CREATE, READ, WRITE));  
    Arena arena = Arena.ofConfined()  
){  
    MemorySegment mapped = fc.map(  
        READ_WRITE,  
        0L,  
        POINT.byteSize(),  
        arena  
    );  
    ...  
}
```



Foreign Functions

- Bibliothek
- Funktion
- Signatur

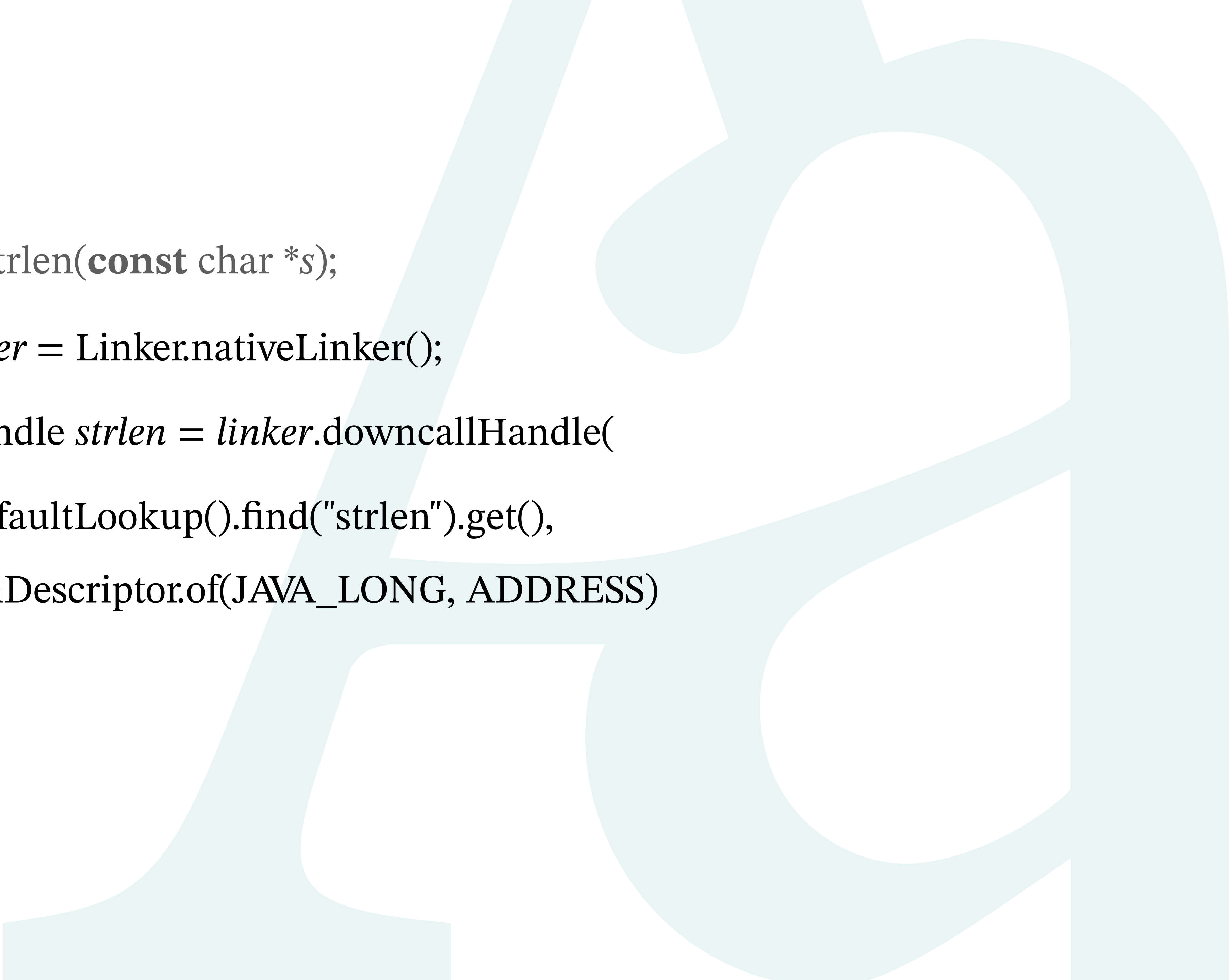
- `SymbolLookup::libraryLookup(String, Arena)`
sucht nach allen Symbolen in der genannten Bibliothek und assoziiert diese mit der gegebenen Arena
- `SymbolLookup::loaderLookup()`
sucht nach allen Symbolen in denjenigen Bibliotheken, die vom aktuellen Class-Loader geladen wurden (mit Hilfe von `System::loadLibrary` bzw. `System::load`)
- `Linker::defaultLookup()`
sucht nach allen Symbolen, die in den typischen Plattformbibliotheken zu finden sind



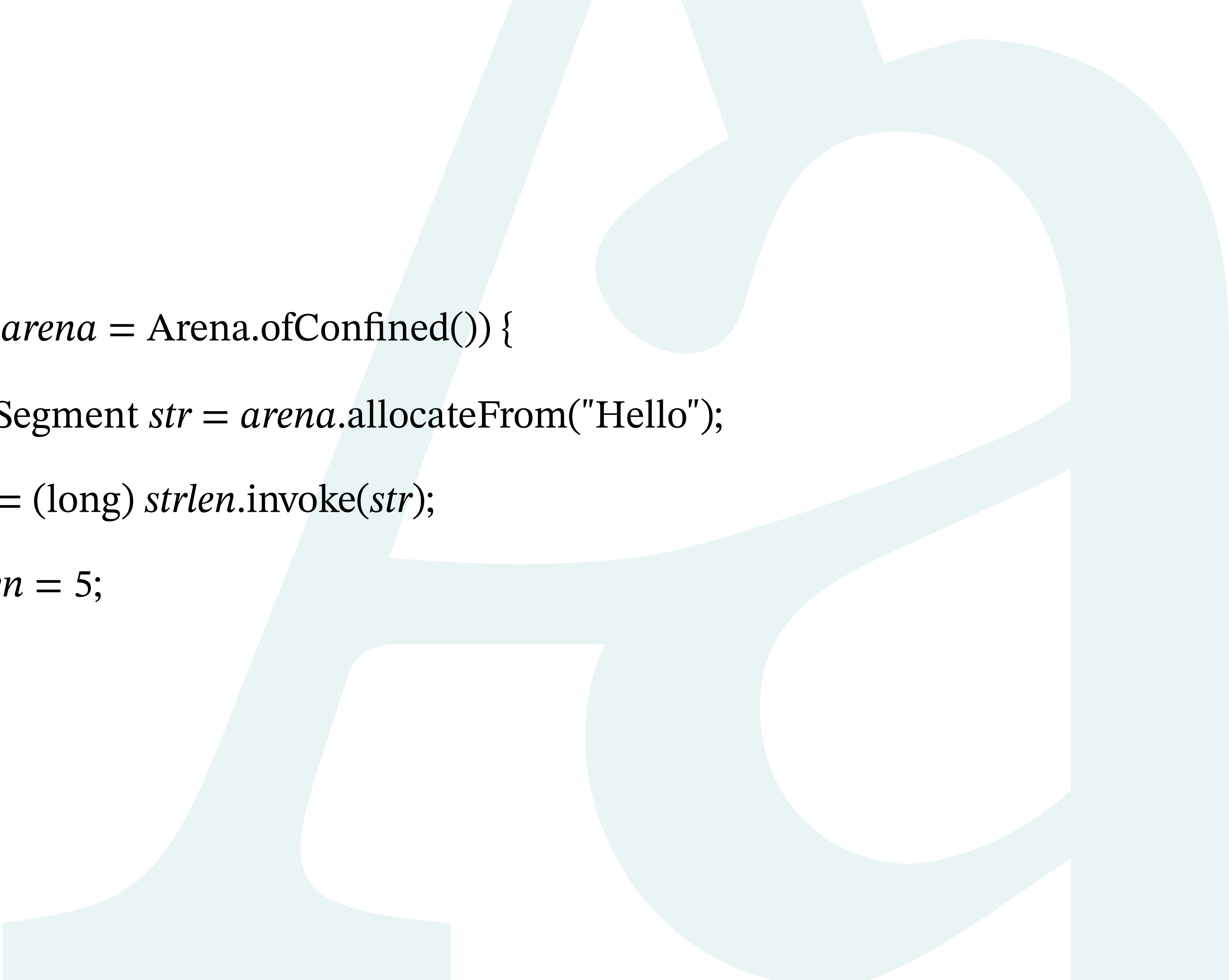
```
try (Arena arena = Arena.ofConfined()) {  
    SymbolLookup opengl = SymbolLookup.libraryLookup("libGL.so", arena);  
    MemorySegment glVersion = opengl.find("glGetString").get();  
    ...  
} // libGL.so wird freigegeben.
```



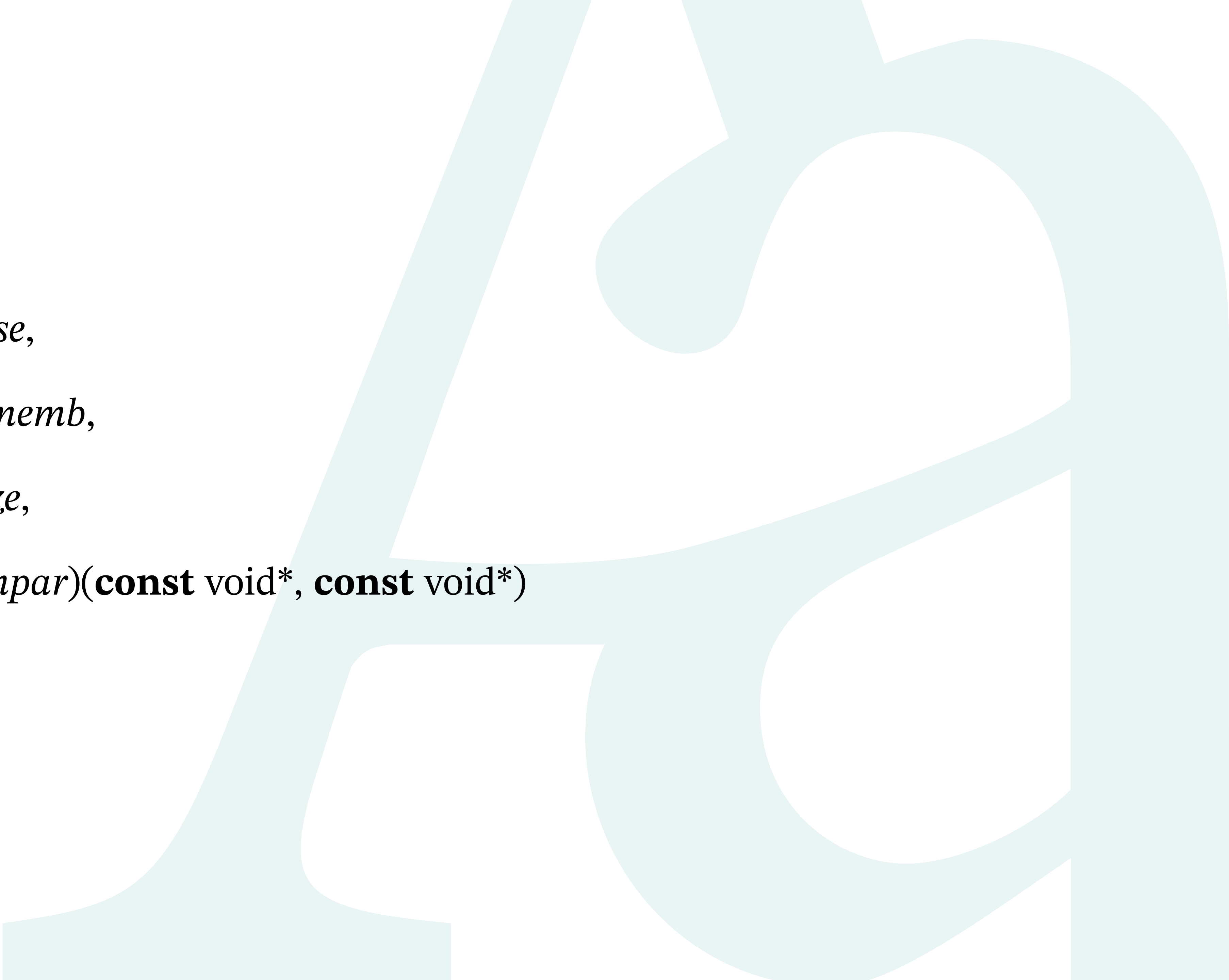
```
size_t  strlen(const char *s);
```



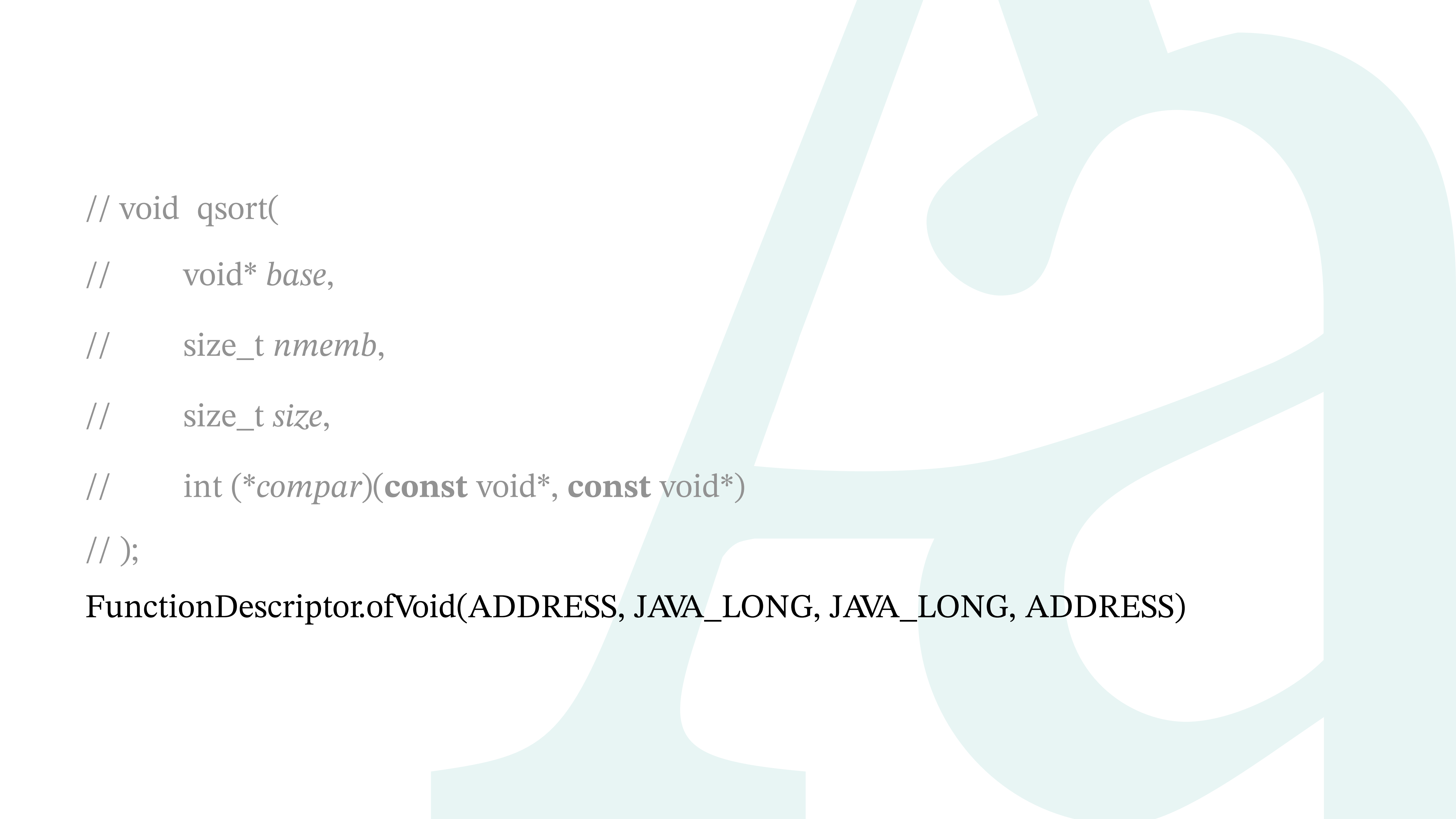
```
// size_t  strlen(const char *s);  
  
Linker linker = Linker.nativeLinker();  
  
MethodHandle strlen = linker.downcallHandle(  
    linker.defaultLookup().find("strlen").get(),  
    FunctionDescriptor.of(JAVA_LONG, ADDRESS)  
);
```



```
try (Arena arena = Arena.ofConfined()) {  
    MemorySegment str = arena.allocateFrom("Hello");  
    long len = (long) strlen.invoke(str);  
    assert len = 5;  
}
```

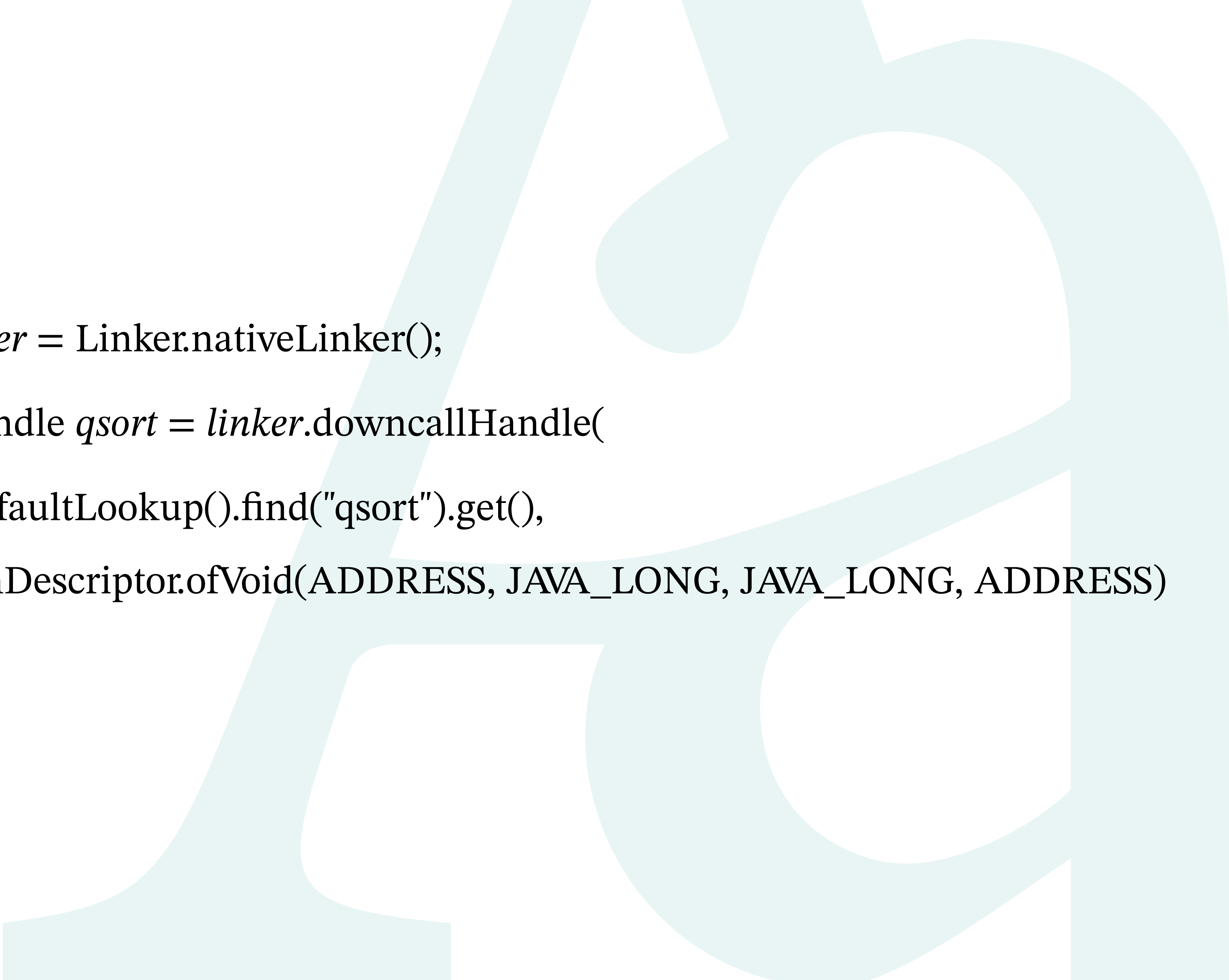


```
void qsort(  
    void* base,  
    size_t nmemb,  
    size_t size,  
    int (*compar)(const void*, const void*)  
);
```

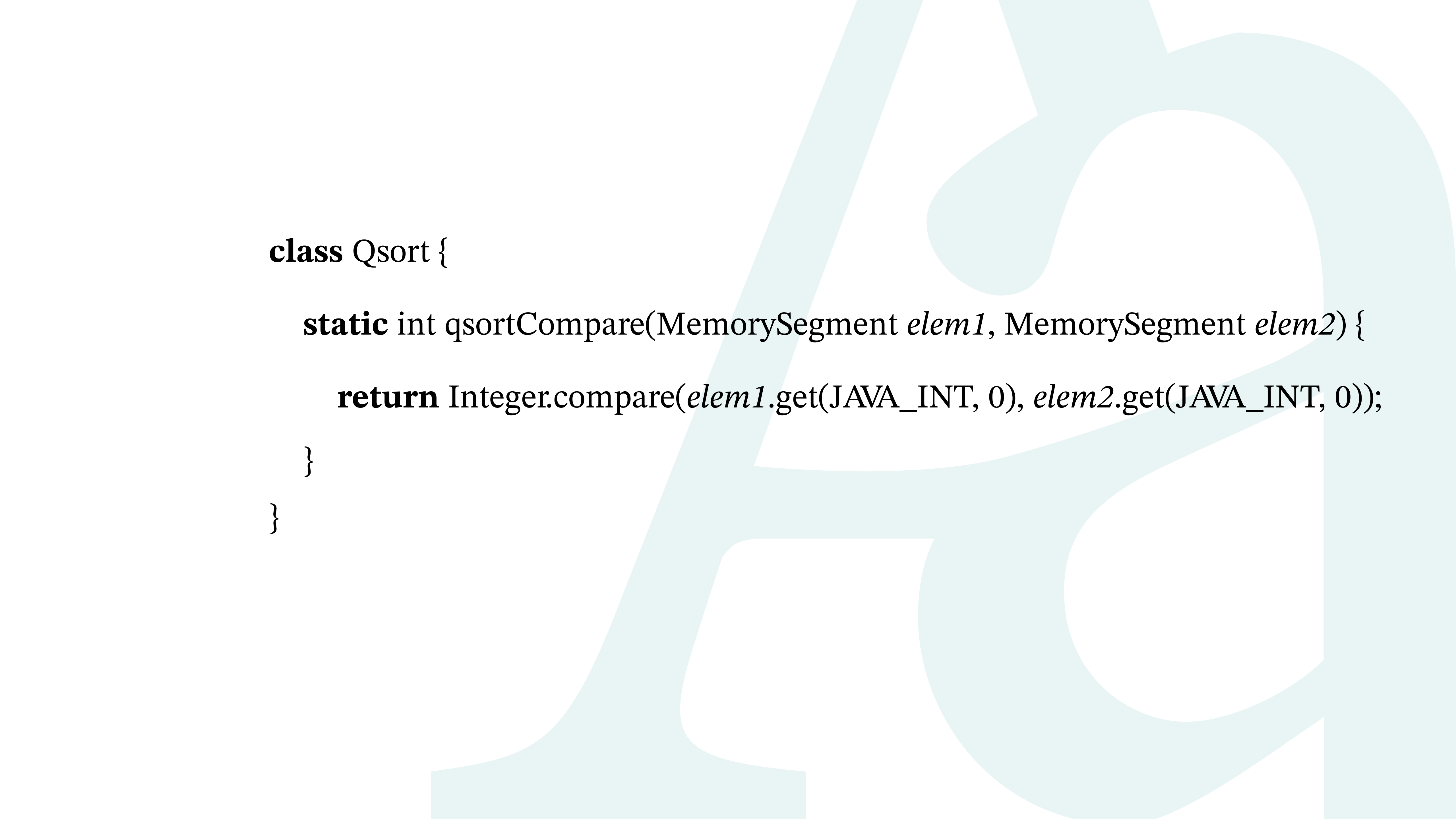


```
// void qsort(  
//     void* base,  
//     size_t nmemb,  
//     size_t size,  
//     int (*compar)(const void*, const void*)  
// );
```

```
FunctionDescriptor.ofVoid(ADDRESS, JAVA_LONG, JAVA_LONG, ADDRESS)
```



```
Linker linker = Linker.nativeLinker();  
MethodHandle qsort = linker.downcallHandle(  
    linker.defaultLookup().find("qsort").get(),  
    FunctionDescriptor.ofVoid(ADDRESS, JAVA_LONG, JAVA_LONG, ADDRESS)  
);
```



```
class Qsort {  
    static int qsortCompare(MemorySegment elem1, MemorySegment elem2) {  
        return Integer.compare(elem1.get(JAVA_INT, 0), elem2.get(JAVA_INT, 0));  
    }  
}
```

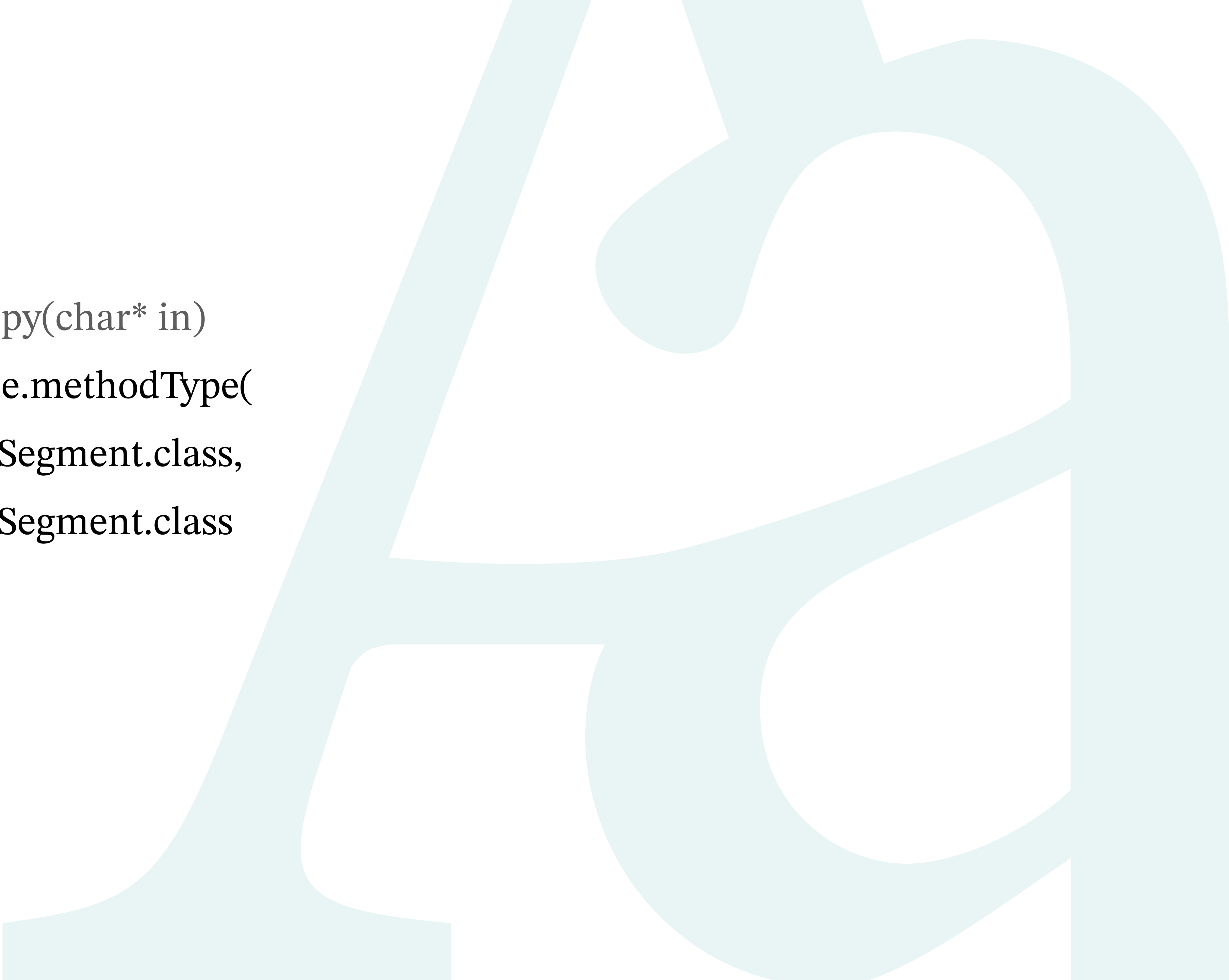
```
// static int qsortCompare(MemorySegment elem1, MemorySegment elem2) {  
  
MethodHandle comparHandle = MethodHandles.lookup().findStatic(  
    Qsort.class,  
    "qsortCompare",  
    MethodType.methodType(  
        int.class,  
        MemorySegment.class,  
        MemorySegment.class  
    )  
);
```

```
MemorySegment comparFunc = linker.upcallStub(  
    comparHandle,  
    // Wegen Integer.compare(elem1.get(JAVA_INT, 0), elem2.get(JAVA_INT, 0))  
    FunctionDescriptor.of(  
        JAVA_INT,  
        ADDRESS.withTargetLayout(JAVA_INT),  
        ADDRESS.withTargetLayout(JAVA_INT)  
    ),  
    Arena.ofAuto()  
);
```

```
int[] sorted = null;  
try (Arena arena = Arena.ofConfined()) {  
    MemorySegment array = arena.allocateFrom(  
        ValueLayout.JAVA_INT,  
        0, 9, 3, 4, 6, 5, 1, 8, 2, 7  
    );  
    qsort.invoke(array, 10L, ValueLayout.JAVA_INT.byteSize(), comparFunc);  
    sorted = array.toArray(JAVA_INT);  
}  
assert sorted = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
```



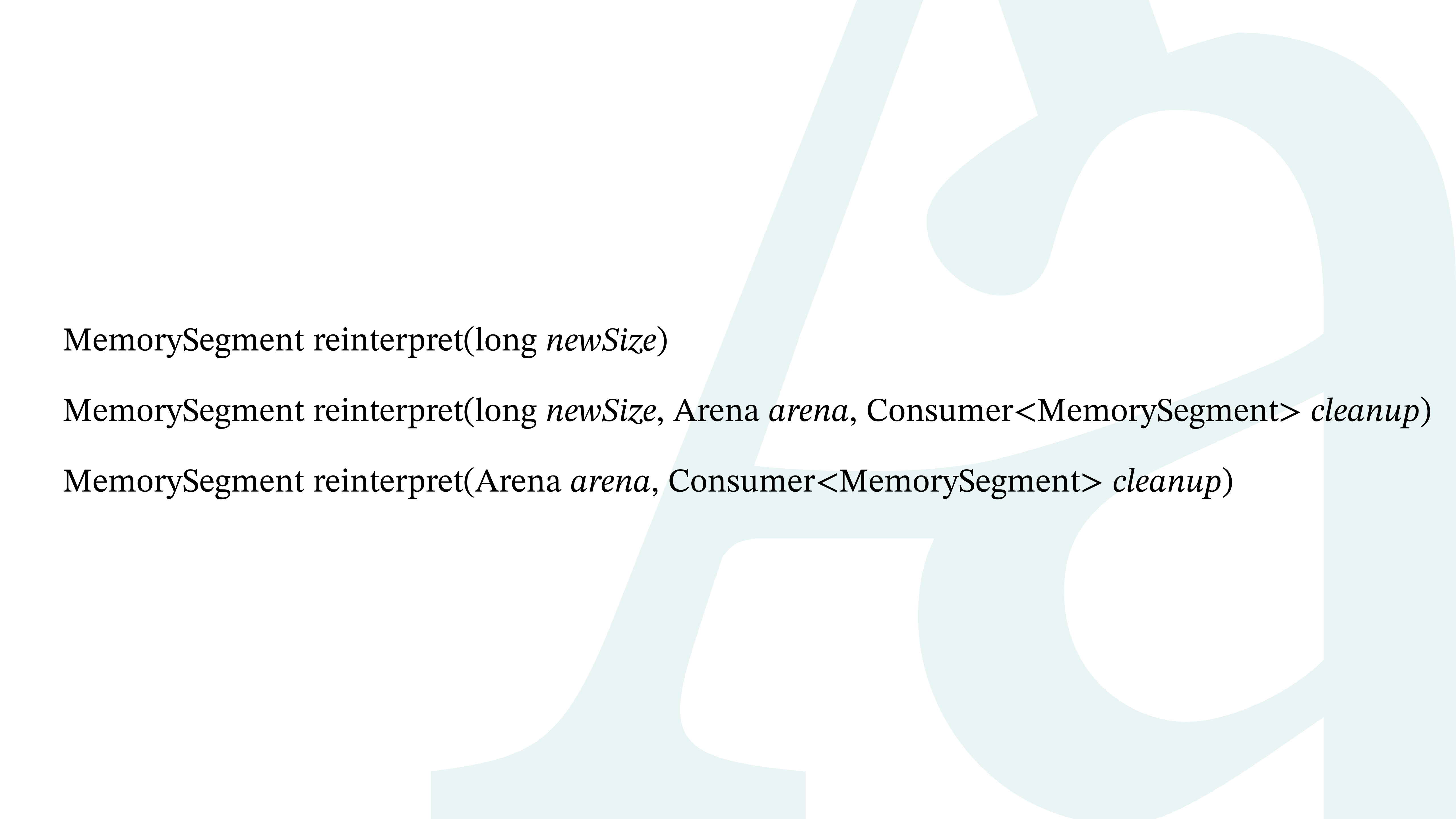
char* copy(char* in)



```
// char* copy(char* in)
MethodType.methodType(
    MemorySegment.class,
    MemorySegment.class
)
```



```
MemorySegment reinterpret(  
    long newSize,  
    Arena arena,  
    Consumer<MemorySegment> cleanup  
)
```



MemorySegment reinterpret(long *newSize*)

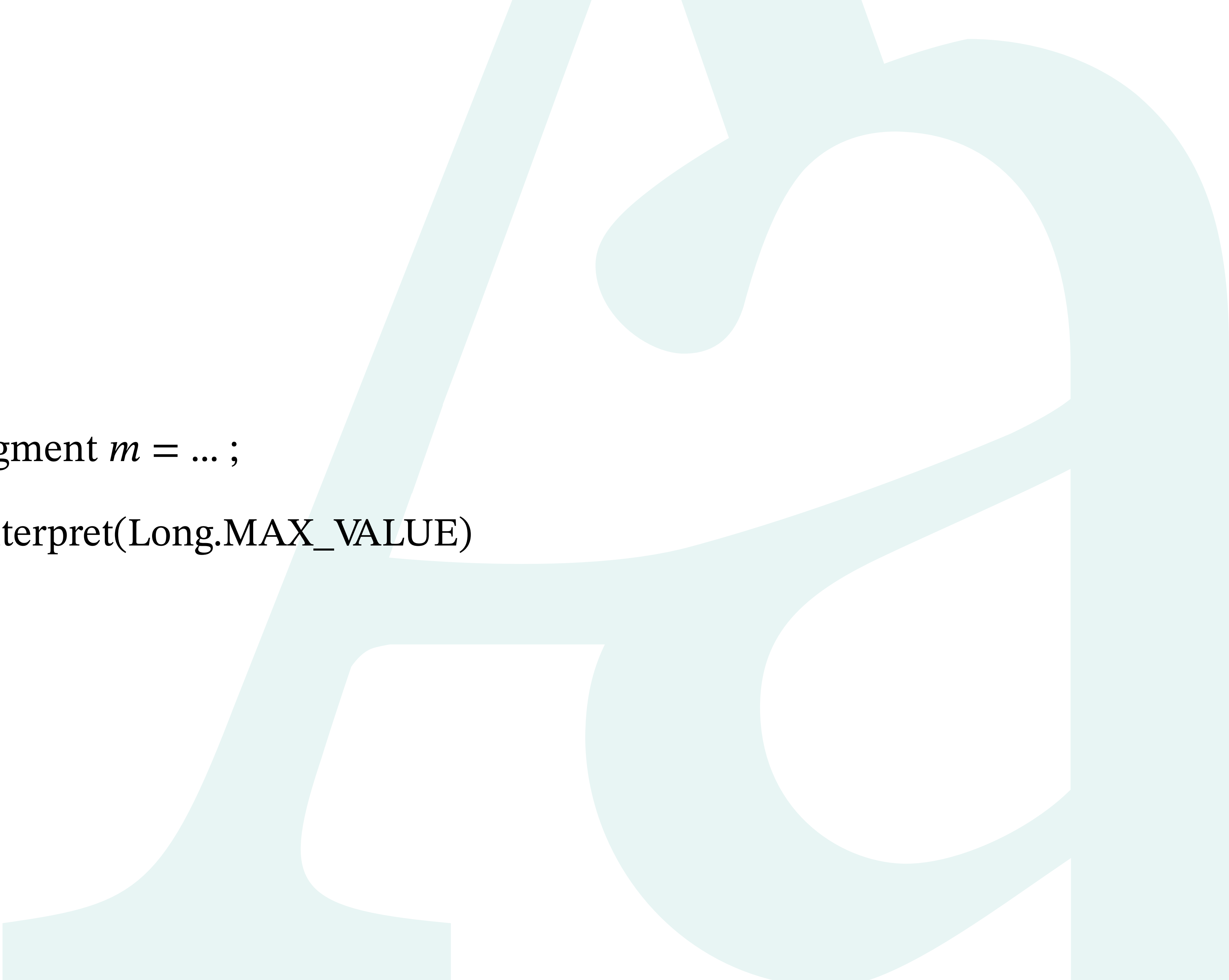
MemorySegment reinterpret(long *newSize*, Arena *arena*, Consumer<MemorySegment> *cleanup*)

MemorySegment reinterpret(Arena *arena*, Consumer<MemorySegment> *cleanup*)



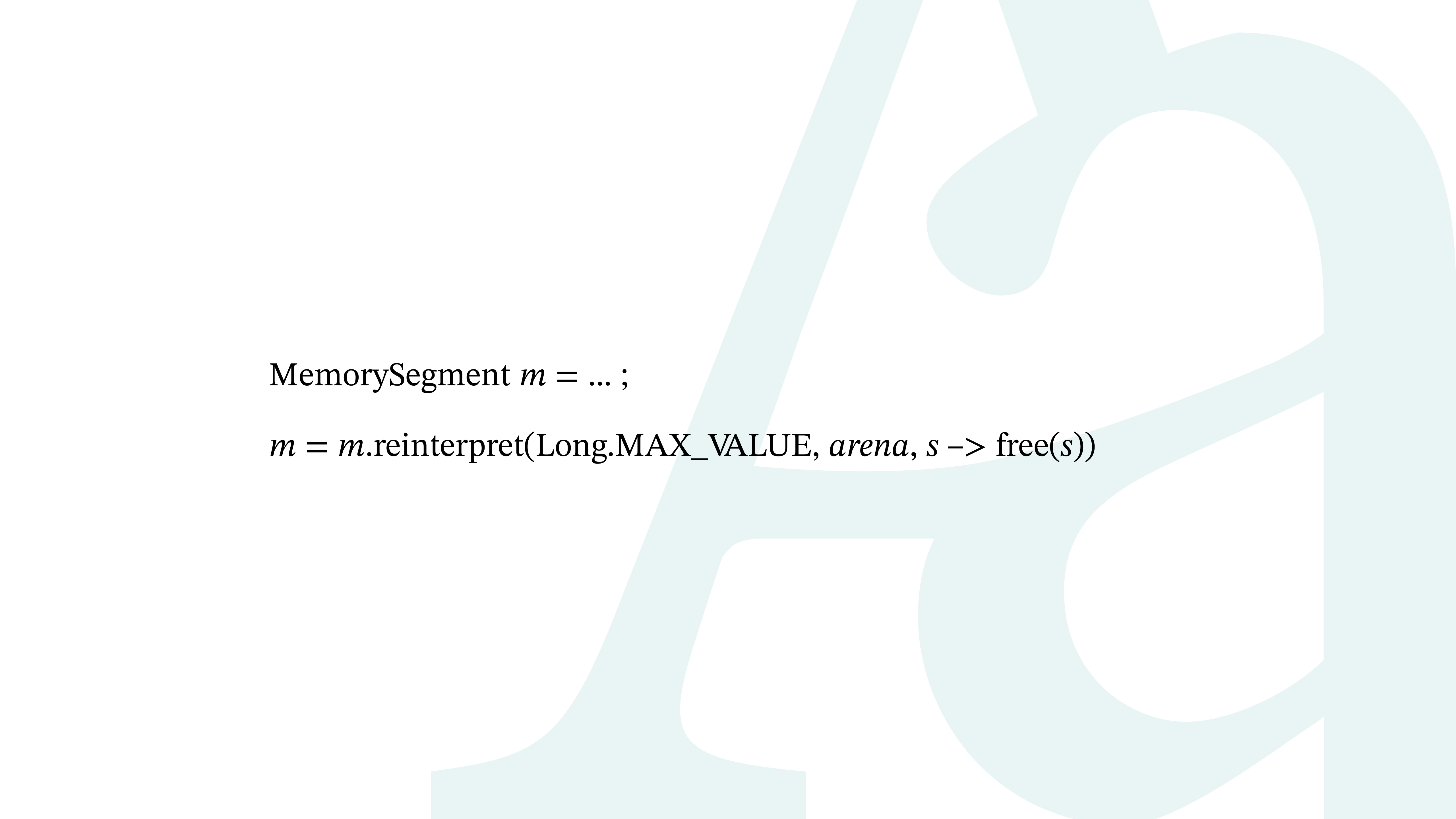
MemorySegment $m = \dots$;

$m = m.\text{reinterpret}(n * \text{RESULT.byteSize}());$



MemorySegment $m = \dots$;

$m = m.\text{reinterpret}(\text{Long.MAX_VALUE})$



MemorySegment $m = \dots$;

$m = m.\text{reinterpret}(\text{Long.MAX_VALUE}, \textit{arena}, s \rightarrow \text{free}(s))$

Hilfe



Jextract

Project Jextract Early-Access Builds

<https://jdk.java.net/jextract/>

Jextract

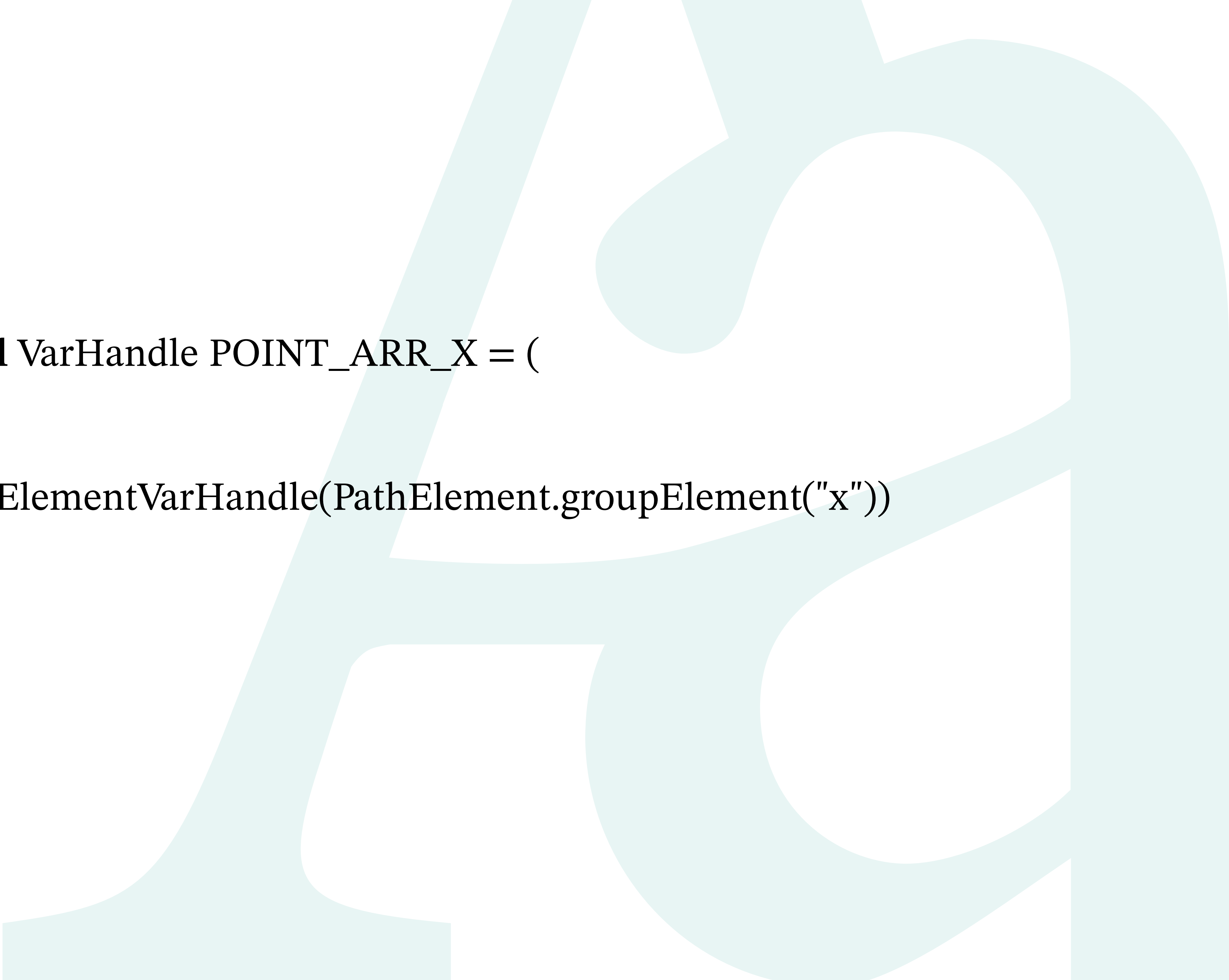
<https://github.com/openjdk/jextract/>

Jextract Guide

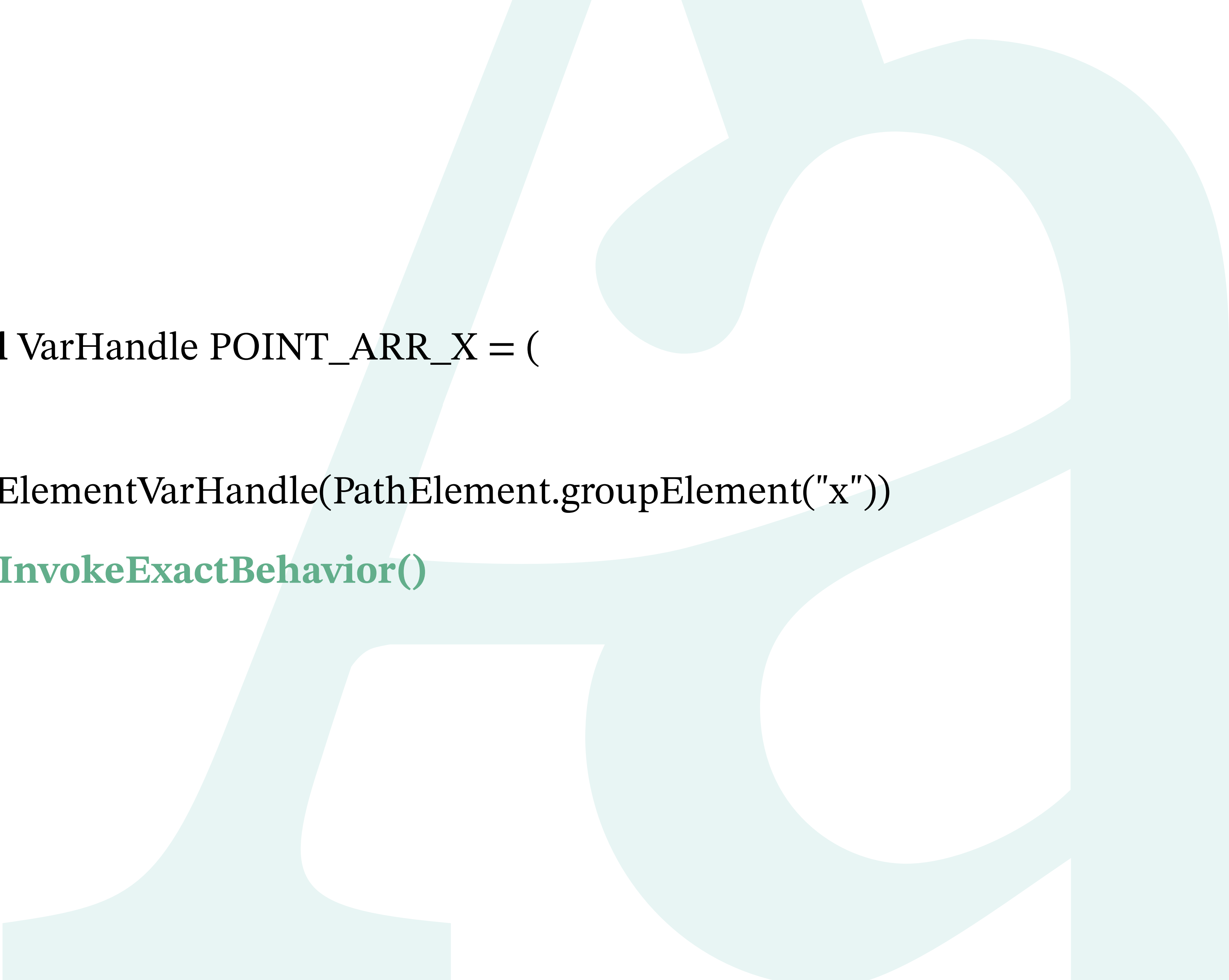
<https://github.com/openjdk/jextract/blob/master/doc/GUIDE.md>



Verpackungsmüll



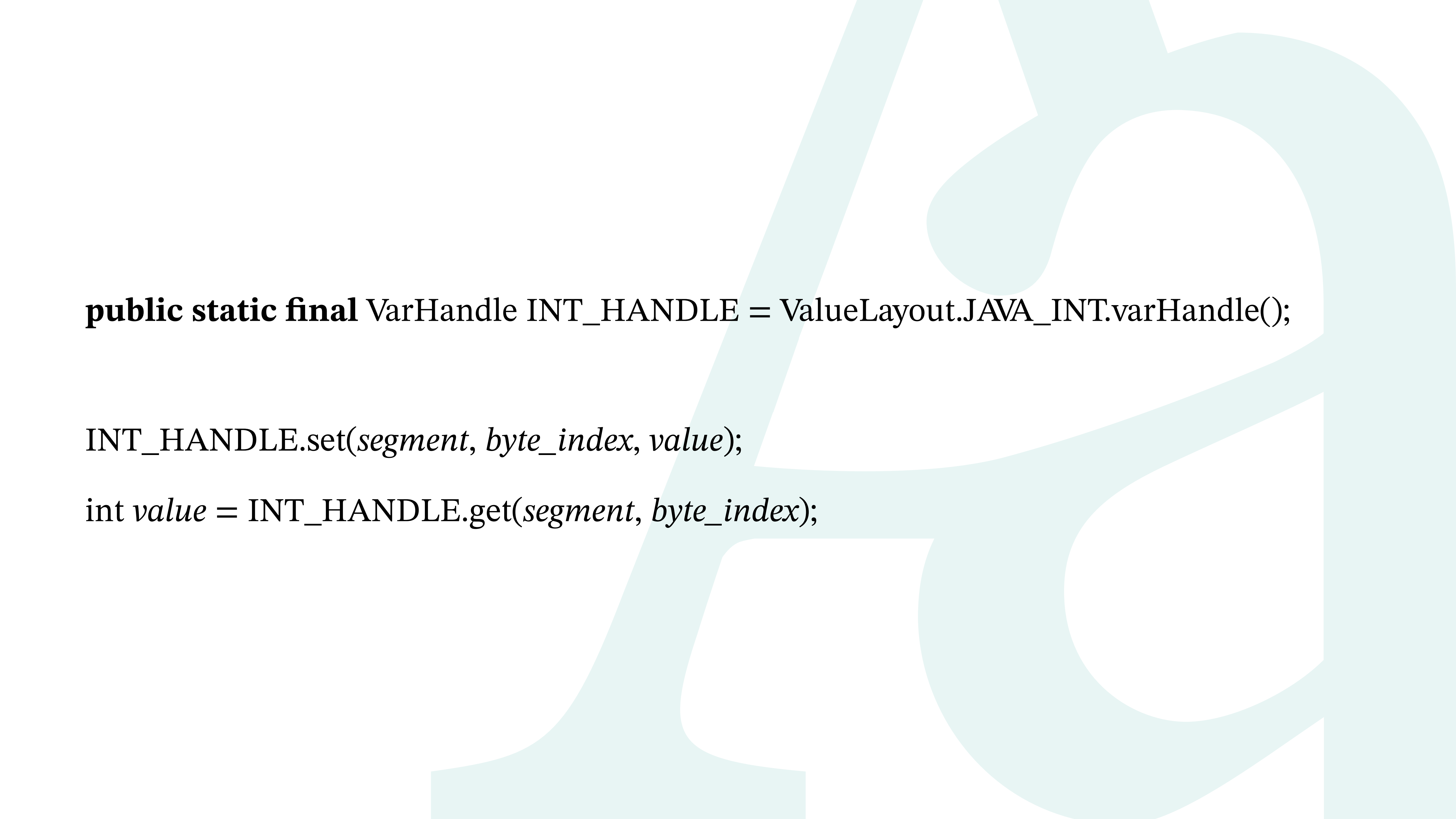
```
static final VarHandle POINT_ARR_X = (  
    POINT  
        .arrayElementVarHandle(PathElement.groupElement("x"))  
);
```



```
static final VarHandle POINT_ARR_X = (  
    POINT  
        .arrayElementVarHandle(PathElement.groupElement("x"))  
        .withInvokeExactBehavior()  
);
```



Spitzfindigkeiten



```
public static final VarHandle INT_HANDLE = ValueLayout.JAVA_INT.varHandle();
```

```
INT_HANDLE.set(segment, byte_index, value);
```

```
int value = INT_HANDLE.get(segment, byte_index);
```

getVolatile

setVolatile

getOpaque

setOpaque

getAcquire

setRelease

compareAndSet

getAndSet

getAndSetAcquire

getAndSetRelease

acquireFence

fullFence

releaseFence



Sicherheit



WARNING: A restricted method in `java.lang.foreign.Linker` has been called

WARNING: `Linker::downcallHandle` has been called by `com.foo.Server` in an unnamed module

WARNING: Use `--enable-native-access=ALL-UNNAMED` to avoid a warning for callers in this module

WARNING: Restricted methods will be blocked in a future release unless native access is enabled

- **Kommandozeile:** `--enable-native-access=MODULE`
 - Unterdrückt die Warnung für die angegebenen Module
 - Führt für alle anderen Module zu einer `IllegalCallerException`
- **JAR-Manifest:** `Enable-Native-Access: ALL-UNNAMED`

Unterdrückt die Warnung für die enthaltenen Klassen
- Mag zukünftig zwingend erforderlich sein, um eingeschränkte (restricted) bzw. unsichere (unsafe) Aufrufe machen zu können



Noch mehr Hilfe

Jextract

Foreign Function and Memory API

<https://docs.oracle.com/en/java/javase/27/core/foreign-function-and-memory-api.html>

1B Rows with the Memory API - JEP Cafe #25

<https://www.youtube.com/watch?v=1VORyDhQzf8>



Vielen Dank!