

CASE STUDY

**ONE YEAR IN ULTRA LOW LATENCY JAVA
PROJECT FOR FINANCIAL INSTITUTION**

ABOUT ME

- Mateusz Przybylski
- Former EPAM Systems employee.
- Java Software Engineer in Sii Gdańsk.
- Trainer at infoShare Academy.

AGENDA

1. Low latency in a few words.
2. Case 1: know your API!
3. Case 2: SBE - working with bits.
4. Case 3: „will it be inlined?”.
5. Case 4: performance > „safety”.
6. Case 5: painful testing.
7. Conclusions.

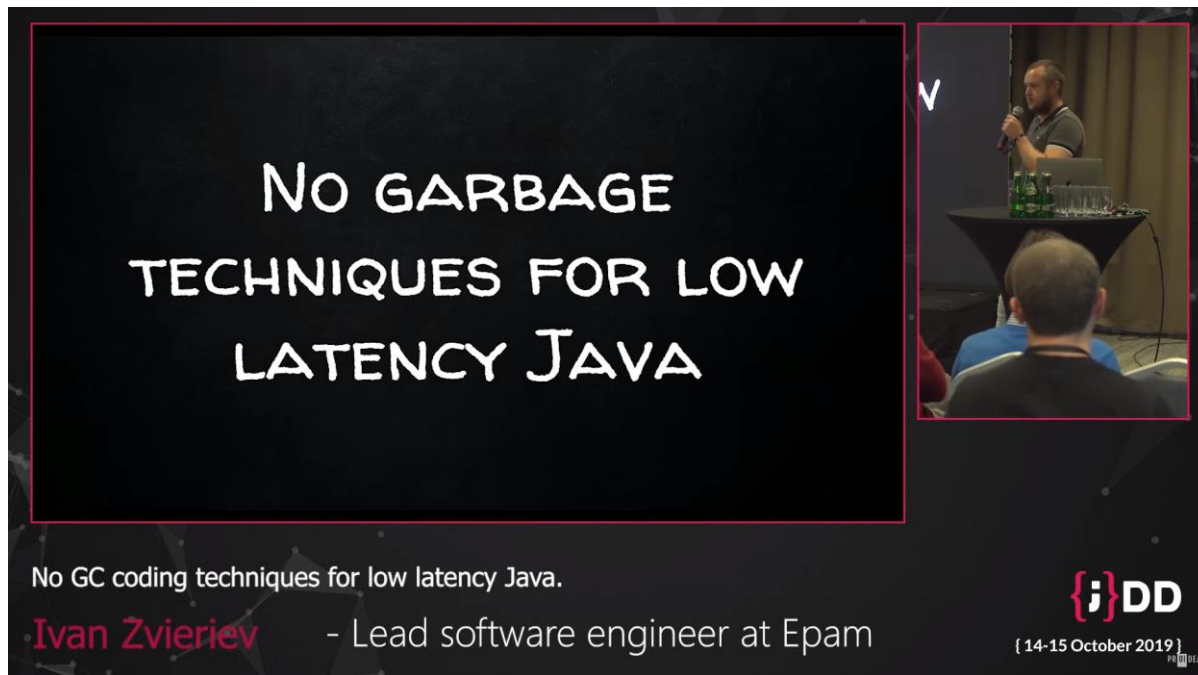
LOW LATENCY IN A FEW WORDS.

LOW LATENCY IN A FEW WORDS.

- What's low latency? – „it depends”.
„The turtle moves fast enough to hunt down a lettuce.”
- Less than 1ms – ultra low latency.
- Physics matters: speed of light, cables' lengths.
- Finance domain: session layer (FIX, FIXP), encoding standard (FIX, SBE).
- Big stress and fault tolerance.
- „When the airplane crashes into the building – then we make money”.
- There are ready-made tools and libraries for low latency. But...
- The fastest wins! Using ready-made tools, we are just as fast as others.

LOW LATENCY IN A FEW WORDS.

- Why Java?
- My former team colleague explained it last year on JDD 2019:



The image is a screenshot of a presentation slide. The main part of the slide has a black background with white text that reads "NO GARBAGE TECHNIQUES FOR LOW LATENCY JAVA". Below this, in smaller white text, it says "No GC coding techniques for low latency Java." and "Ivan Zvieriev - Lead software engineer at Epam". In the bottom right corner, there is a logo for "JDD" (Java Developer Days) with the text "[14-15 October 2019]". On the right side of the slide, there is a small inset video frame showing a man (Ivan Zvieriev) speaking at a podium with a microphone and a laptop. The background of the slide features a faint, abstract geometric pattern.

[YouTube:](#)

[JDD 2019: No GC coding techniques for low latency Java, Ivan Zvieriev](#)

LOW LATENCY IN FINANCE DOMAIN.

„A 1-millisecond advantage in trading applications can be worth **\$100 million** a year to a major brokerage firm.”

„[...] the physical distance between two computers processing a transaction can slow down how fast it happens. [...] To overcome it, many high-frequency algorithmic traders are moving their systems as close to the Wall Street exchanges as possible.”

Source:

<https://www.informationweek.com/wall-streets-quest-to-process-data-at-the-speed-of-light/d/d-id/1054287>

[https://en.wikipedia.org/wiki/Low_latency_\(capital_markets\)](https://en.wikipedia.org/wiki/Low_latency_(capital_markets))

INTERESTING FACTS.



[Slowing Down A Stock Exchange
With 38 Miles Of Cable](#)



<https://hackaday.com/2019/02/26/putting-the-brakes-on-high-frequency-trading-with-physics/>

<https://iextrading.com/>

CASE 1: KNOW YOUR API!

CASE 1: KNOW YOUR API!

- How deep is your knowledge about JDK internals?

```
List<Integer> normalList = new ArrayList<>();
```

```
List<Integer> presizedList = new ArrayList<>( initialCapacity: 1000);
```

```
public boolean add(E e) {  
    modCount++;  
    add(e, elementData, size);  
    return true;  
}
```

```
private void add(E e, Object[] elementData, int s) {  
    if (s == elementData.length)  
        elementData = grow();  
    elementData[s] = e;  
    size = s + 1;  
}
```

```
private Object[] grow(int minCapacity) {  
    int oldCapacity = elementData.length;  
    if (oldCapacity > 0 || elementData != DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {  
        int newCapacity = ArraysSupport.newLength(oldCapacity,  
            minGrowth: minCapacity - oldCapacity, /* minimum growth */  
            prefGrowth: oldCapacity >> 1          /* preferred growth */);  
        return elementData = Arrays.copyOf(elementData, newCapacity);  
    } else {  
        return elementData = new Object[Math.max(DEFAULT_CAPACITY, minCapacity)];  
    }  
}
```

@HotSpotIntrinsicCandidate

```
public static native void arraycopy( @NotNull() @Flow(...) Object src, int srcPos,  
    @NotNull()  
    Object dest, int destPos,  
    int length);
```

```
* 1 th_ {@code src} and {@code dest} arguments refer to the  
* s se array object, then the copying is performed as if the  
* c mpents at positions {@code srcPos} through  
* {@code srcPos+length-1} were first copied to a temporary  
* array with {@code length} components and then the contents of  
* the temporary array were copied into positions  
* {@code destPos} through {@code destPos+length-1} of the  
* destination array.
```

CASE 1: KNOW YOUR API!

- How often do you investigate native method's implementation?

[...]

```
248 void ObjArrayKlass::copy_array(arrayOop s, int src_pos, arrayOop d,  
249                               int dst_pos, int length, TRAPS) {  
250     assert(s->is_objArray(), "must be obj array");  
251  
252     if (!d->is_objArray()) {  
253         ResourceMark rm(THREAD);  
254         stringstream ss;  
255         if (d->is_typeArray()) {  
256             ss.print("arraycopy: type mismatch: can not copy object array[] into %s[]",  
257                     type2name_tab[ArrayKlass::cast(d->klass())->element_type()]);  
258         } else {  
259             ss.print("arraycopy: destination type %s is not an array", d->klass()->external_name());  
260         }  
261         THROW_MSG(vmSymbols::java_lang_ArrayStoreException(), ss.as_string());  
262     }  
263 }
```

[...]

CASE 1: KNOW YOUR API!

- Agrona from <https://real-logic.co.uk/>
 - <https://github.com/real-logic/agrona/tree/master/agrona/src/main/java/org/agrona/collections>

Int2ObjectHashMap.java

IntArrayList.java

IntArrayQueue.java

IntHashSet.java

```
public Int2ObjectHashMap() { this( initialCapacity: 8, loadFactor: 0.55F, shouldAvoidAllocation: true); }

public Int2ObjectHashMap(int initialCapacity, float loadFactor) { this(initialCapacity, loadFactor, shouldAvoidAllocation: true); }

public Int2ObjectHashMap(int initialCapacity, float loadFactor, boolean shouldAvoidAllocation) {
    CollectionUtil.validateLoadFactor(loadFactor);
    this.loadFactor = loadFactor;
    this.shouldAvoidAllocation = shouldAvoidAllocation;
    int capacity = BitUtil.findNextPositivePowerOfTwo(Math.max(8, initialCapacity));
    this.resizeThreshold = (int)((float)capacity * loadFactor);
    this.keys = new int[capacity];
    this.values = new Object[capacity];
}
```

```
public V put(final Integer key, final V value)
{
    return put(key.intValue(), value);
}
```

```
@SuppressWarnings("unchecked")
public V put(final int key, final V value)
{
}
```

CASE 2: SBE – WORKING WITH BITS.

CASE 2: SBE – WORKING WITH BITS.

- Simple Binary Encoding (SBE).

byte[] buffer = [b₁, b₂, b₃, b₄, ...]

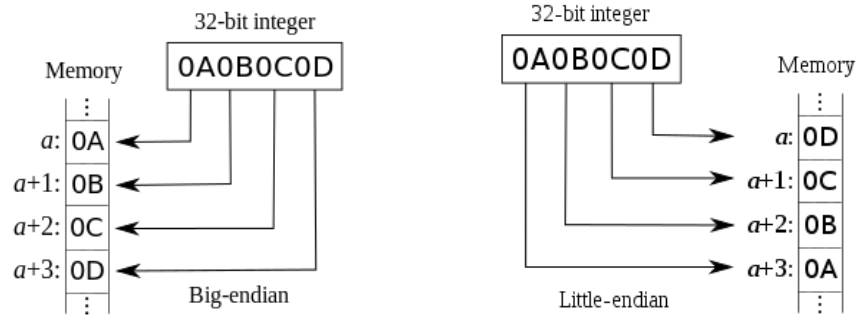
0000 0000 0000 0000 0000 0000 0000 0000

CASE 2: SBE – WORKING WITH BITS.

0010 0001 0001 1101 0001 1010 1110 0011



$[b_1, b_2, b_3, b_4, \dots]$



Source: <https://en.wikipedia.org/wiki/Endianness>

CASE 2: SBE – WORKING WITH BITS.

```
/**
 * A typesafe enumeration for byte orders.
 *
 * @author Mark Reinhold
 * @author JSR-51 Expert Group
 * @since 1.4
 */

public final class ByteOrder {

    private String name;

    private ByteOrder(String name) { this.name = name; }

    public static final ByteOrder BIG_ENDIAN
        = new ByteOrder( name: "BIG_ENDIAN");

    public static final ByteOrder LITTLE_ENDIAN
        = new ByteOrder( name: "LITTLE_ENDIAN");

    private static final ByteOrder NATIVE_ORDER
        = Unsafe.getUnsafe().isBigEndian()
            ? ByteOrder.BIG_ENDIAN : ByteOrder.LITTLE_ENDIAN;
```


CASE 2: SBE – WORKING WITH BITS.

int – 4 bajty – 32 bity

-2 147 483 648 ≤ int ≤ 2 147 483 647

Integer.MIN_VALUE

Integer.MAX_VALUE

```
System.out.println("Binary representation of Integer.MIN_VALUE: " +  
    Integer.toBinaryString(Integer.MIN_VALUE));
```

```
System.out.println("Binary representation of Integer.MAX_VALUE: " +  
    Integer.toBinaryString(Integer.MAX_VALUE));
```

1000 0000 0000 0000 0000 0000 0000 0000

0111 1111 1111 1111 1111 1111 1111 1111

CASE 2: SBE – WORKING WITH BITS.

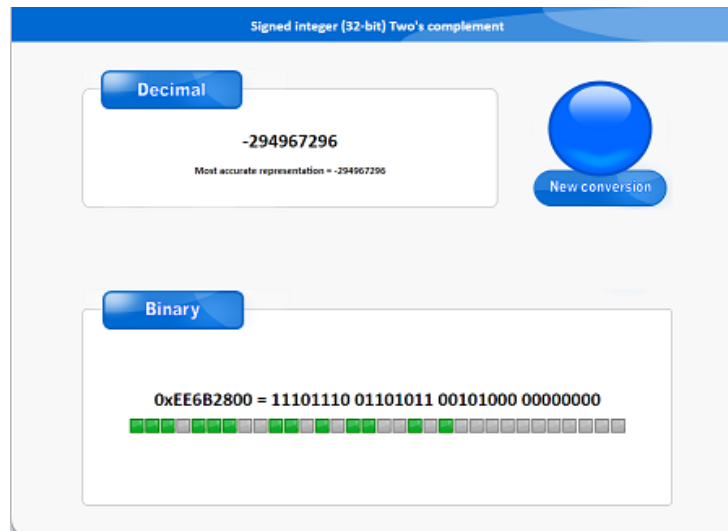
4 000 000 000 -> ile bitów?

```
System.out.println("Binary representation of 4 000 000 000: "  
+ Long.toBinaryString(4_000_000_000L));
```

1110 1110 0110 1011 0010 1000 0000 0000



0000 0000 0000 0000 0000 0000 0000 0000 0000 1110 1110 0110 1011 0010 1000 0000 0000



Source: https://www.binaryconvert.com/result_signed_int.html?hexadecimal=EE6B2800

CASE 2: SBE – WORKING WITH BITS.

```
System.out.println("Binary representation of -294 967 296: " +  
    Integer.toBinaryString(i: -294_967_296));
```

```
System.out.println(Long.toBinaryString(i: 4_000_000_000L).equals(Integer.toBinaryString(i: -294_967_296)));
```



Result of 'Long.toBinaryString(4_000_000_000L).equals(Integer.toBinaryString(-294_967_296))' is always 'true'

f

1111

&

binary AND operator

```
System.out.println("UnsignedInt value of (-294 967 296) as Java long: " + Integer.toUnsignedLong(x: -294967296));  
System.out.println("UnsignedInt value of (-294 967 296) as Java long: " + ((-294967296) & 0xffffffffL));
```

UnsignedInt value of (-294 967 296) as Java long: 4000000000

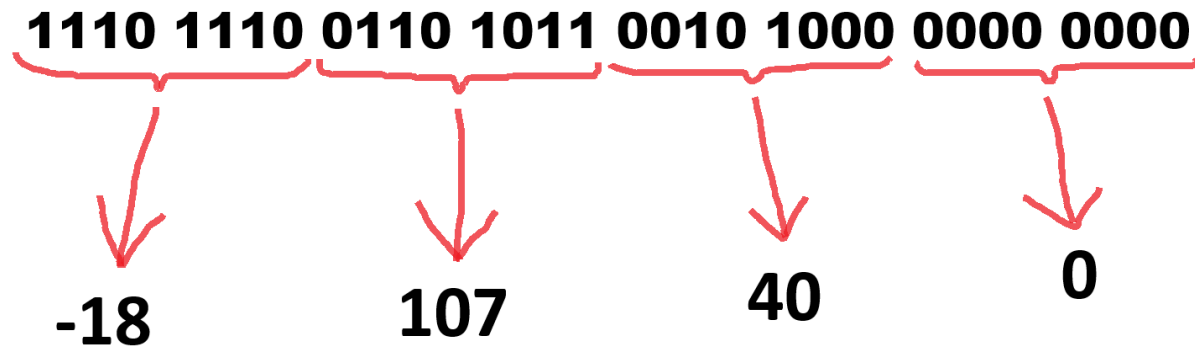
UnsignedInt value of (-294 967 296) as Java long: 4000000000

CASE 2: SBE – WORKING WITH BITS.

$$\mathbf{-128 \leq \text{byte} \leq 127}$$

Byte.MIN_VALUE

Byte.MAX_VALUE



CASE 2: SBE – WORKING WITH BITS.

byte[] buffer = [-18, 107, 40, 0,]

ASCII codes

(character codes)

40 – (

...

65 – A

...

122 – z

CASE 2: SBE – WORKING WITH BITS.

`byte[] buffer = [-18, 107, 40, 0, -63, 2, 76, 100, 13, 0, 88, -113]`

0 1 2 3 4 5 6 7 8 9 10 11

Specification: message_1

Big endian

offset = 0, length = 4 -> price

offset = 4, length = 3 -> buyer ID

offset = 7, length = 1 -> currency symbol

offset = 8, length = 4 -> finance instrument

buffer size = 12

CASE 3: „WILL IT BE INLINED?”.

CASE 3: „WILL IT BE INLINED?”.

//1st case

```
int listSize = list.size();  
for (int i = 0; i < listSize; i++) {  
    mapOne.put(i, listSize );  
}
```

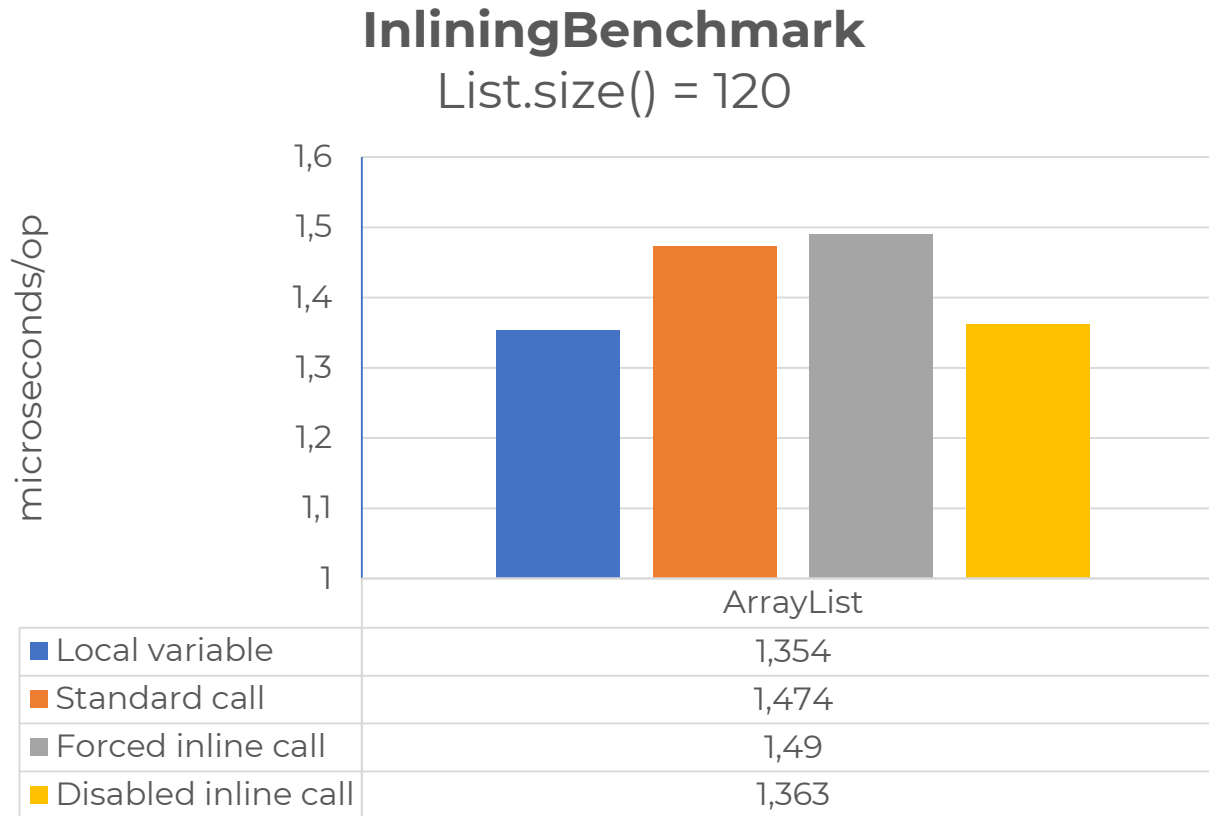
//2nd case

```
for (int i = 0; i < list.size(); i++) {  
    mapTwo.put(i, list.size());  
}
```

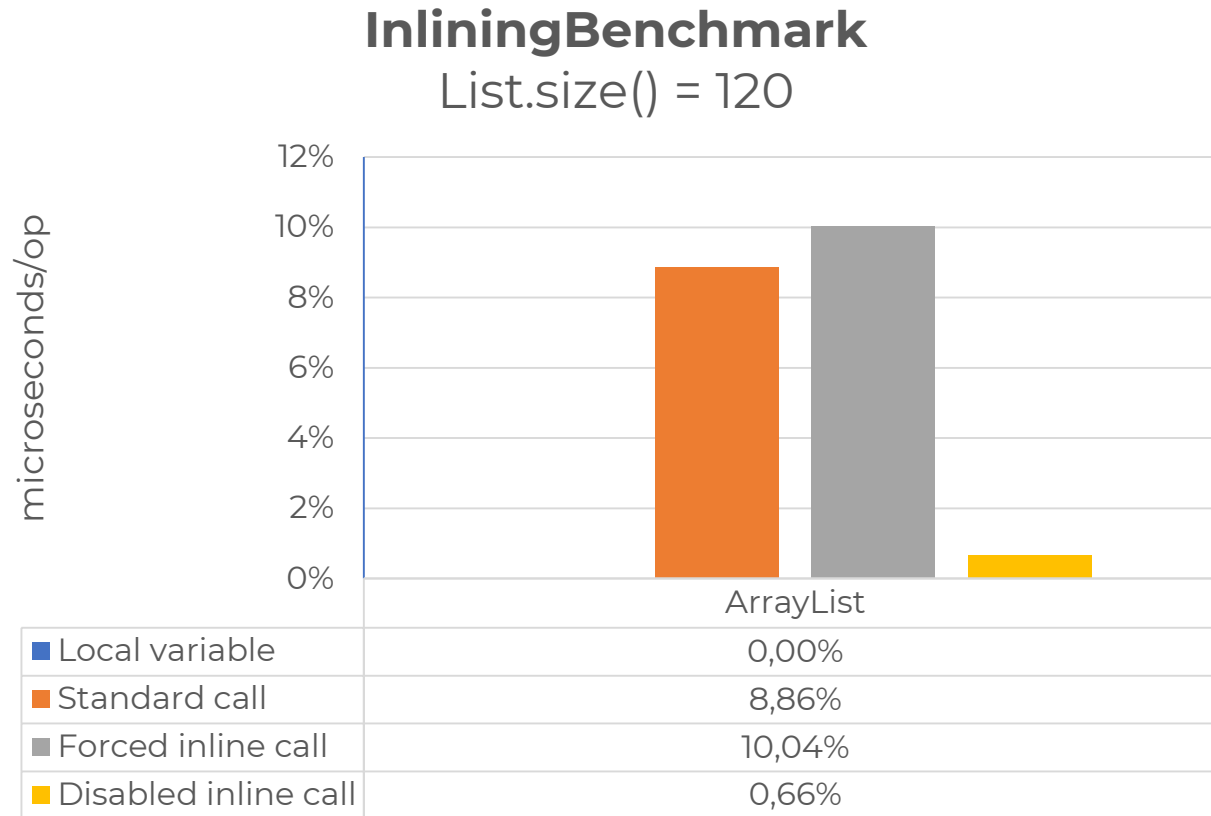

CASE 3: „WILL IT BE INLINED?”.

- What's faster? Will it be inlined? Which solution is better?
- Small Pull Request – around 30 lines of code.
- 5 days of debating in Code review.
- Email corespondence:
 - couple of people CC,
 - sending Java code via email – „benchmark in main”,
 - stacktrace from compilation,
 - JMH benchmarks.

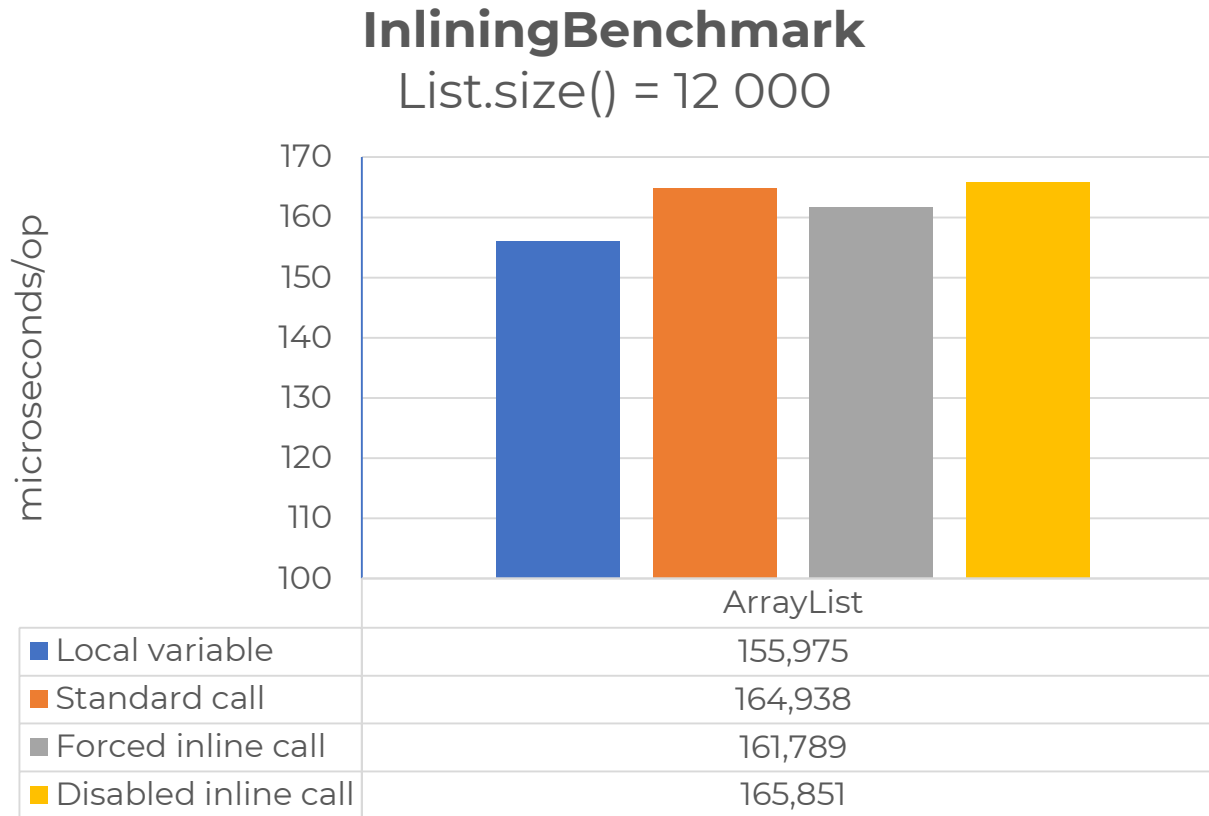
CASE 3: „WILL IT BE INLINED?”.



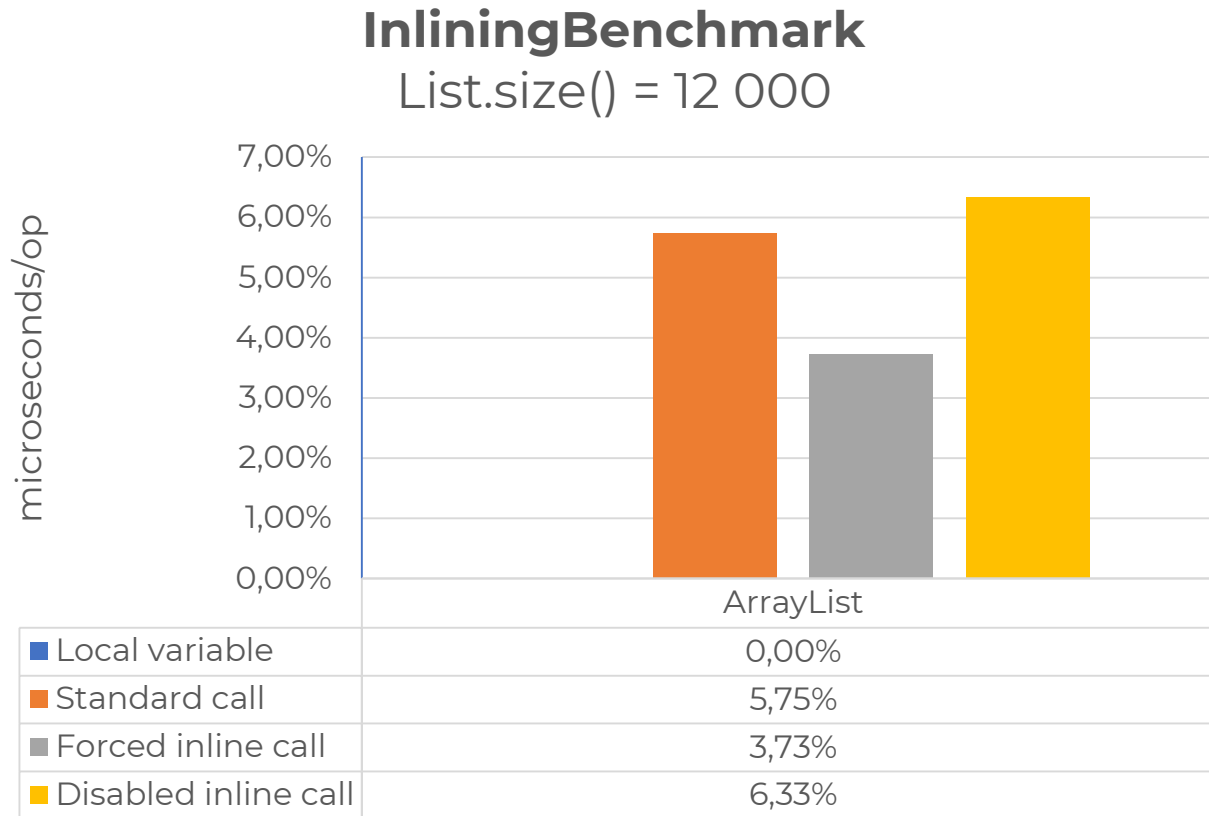
CASE 3: „WILL IT BE INLINED?“.



CASE 3: „WILL IT BE INLINED?“.



CASE 3: „WILL IT BE INLINED?”.



CASE 3: „WILL IT BE INLINED?”.

- Conclusion?
- **THINK REASONABLE!**
- Waste of resources: time, money, health...
- Result?

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

and....

STFU!!!

CASE 4: PERFORMANCE > „SAFETY”.

CASE 4: PERFORMANCE > „SAFETY”.

- Some problems come from they way we code.
 - Performance > „safety”.
 - Run -> POSITIVE
 - Debug -> **NEGATIVE**

CASE 5: PAINFUL TESTING.

CASE 5: PAINFUL TESTING.

- What happens when you:
 - reuse and reset some objects,
 - but maintain state of others,
 - in multithreaded application?
- Tests can become undeterministic...

CONCLUSIONS.

CONCLUSIONS.

- Very demanding work.
- Sophisticated problems.
- A lot to learn.
- Big possibility of growth.
- Easy to get frustrated.
- Easy to get bored.
- You will get some good and bad coding practices.
- Not for everyone!
- „Team-fit” for that kind of project:
 - Tell briefly about the project to the candidate.
 - Ask the most important question: „do you want to do this?”.
 - And then assess necessary knowledge.

THANKS!

QUESTIONS?