Improving Android Performance

#perfmatters

Raimon Ràfols

Agenda

- Disclaimer
- Who am I?
- Our friend the java compiler
- Examples – Do's & don'ts
- Tooling

Disclaimer

This presentation contains bytecode

Who am I?

- Mobile Software Engineering Manager at Imagination Technologies (@imgtec)

- Twitter: @rrafols
- http://blog.rafols.org
- http://imgtec.com/careers

Our friend the java compiler

Android → Java

*.java → [javac] → *.class

*.class → [dx] → dex file

ART

dex file → [dex2oat] → elf file

Javac vs other compilers

Compilers

Produces optimised code for target platform

Javac

Doesn't optimise anything

Javac

Doesn't know on which architecture will the code be executed

For the same reason

Java bytecode is stack based

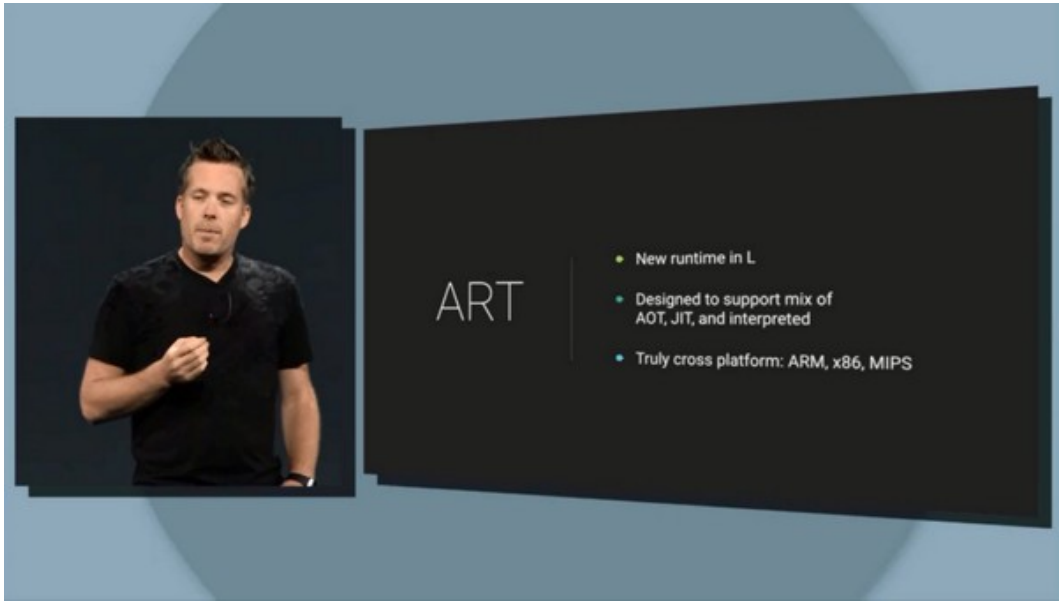
Easy to interpret

No assumptions

But not the most optimal solution (regarding performance)

Dalvik VM and ART* are register based architectures

*ART will support ARM, MIPS and x86



Quick example

Stack based vs Register based

Stack based integer addition Java bytecode

iload_3
iload_2
iadd
istore_2

Register based integer addition Dalvik bytecode

add-int/lit8 v2, v3, #1

Register based integer addition
ART (ARM)

adds r5, r5, #1

Java VM (JVM)

Only the JVM knows on which architecture is running

Java VM (JVM)

All optimisations are left to be done by the JVM

Maybe takes this concept a bit too far...

Imagine this simple C code

```
#include <stdio.h>
int main() {
  int a = 10;
  int b = 1 + 2 + 3 + 4 + 5 + 6 + a;

  printf("%d\n", b);
}
```

GCC compiler

```
#include <stdio.h>
int main() {
  int a = 10;
  int b = 1 + 2 + 3 + 4 + 5 + 6 + a;

  printf("%d\n", b);
}
```

```
…
movl $31, %esi
call _printf
…
```

* Using gcc & -O2 compiler option

javac

```
public static void main(String args[]) {
  int a = 10;
  int b = 1 + 2 + 3 + 4 + 5 + 6 + a;

  System.out.println(b);
}
```

```
0: bipush        10
2: istore_1
3: bipush        21
5: iload_1
6: iadd
7: istore_2
...
```

Let's do a small change

```
#include <stdio.h>
int main() {
  int a = 10;
  int b = 1 + 2 + 3 + 4 + 5 + a + 6;

  printf("%d\n", b);
}
```

GCC compiler

```
#include <stdio.h>
int main() {
  int a = 10;
  int b = 1 + 2 + 3 + 4 + 5 + a + 6;

  printf("%d\n", b);
}
```

```
…
movl $31, %esi
call _printf
…
```

* Using gcc & -O2 compiler option

javac

```
public static void main(String args[]) {
  int a = 10;
  int b = 1 + 2 + 3 + 4 + 5 + a + 6;

  System.out.println(b);
}
```

```
0: bipush        10
2: istore_1
3: bipush        15
5: iload_1
6: iadd
7: bipush        6
9: iadd
10: istore_2

...
```

Let's do another quick change..

```
public static void main(String args[]) {
  int a = 10;
  int b = a + 1 + 2 + 3 + 4 + 5 + 6;

  System.out.println(b);
}
```

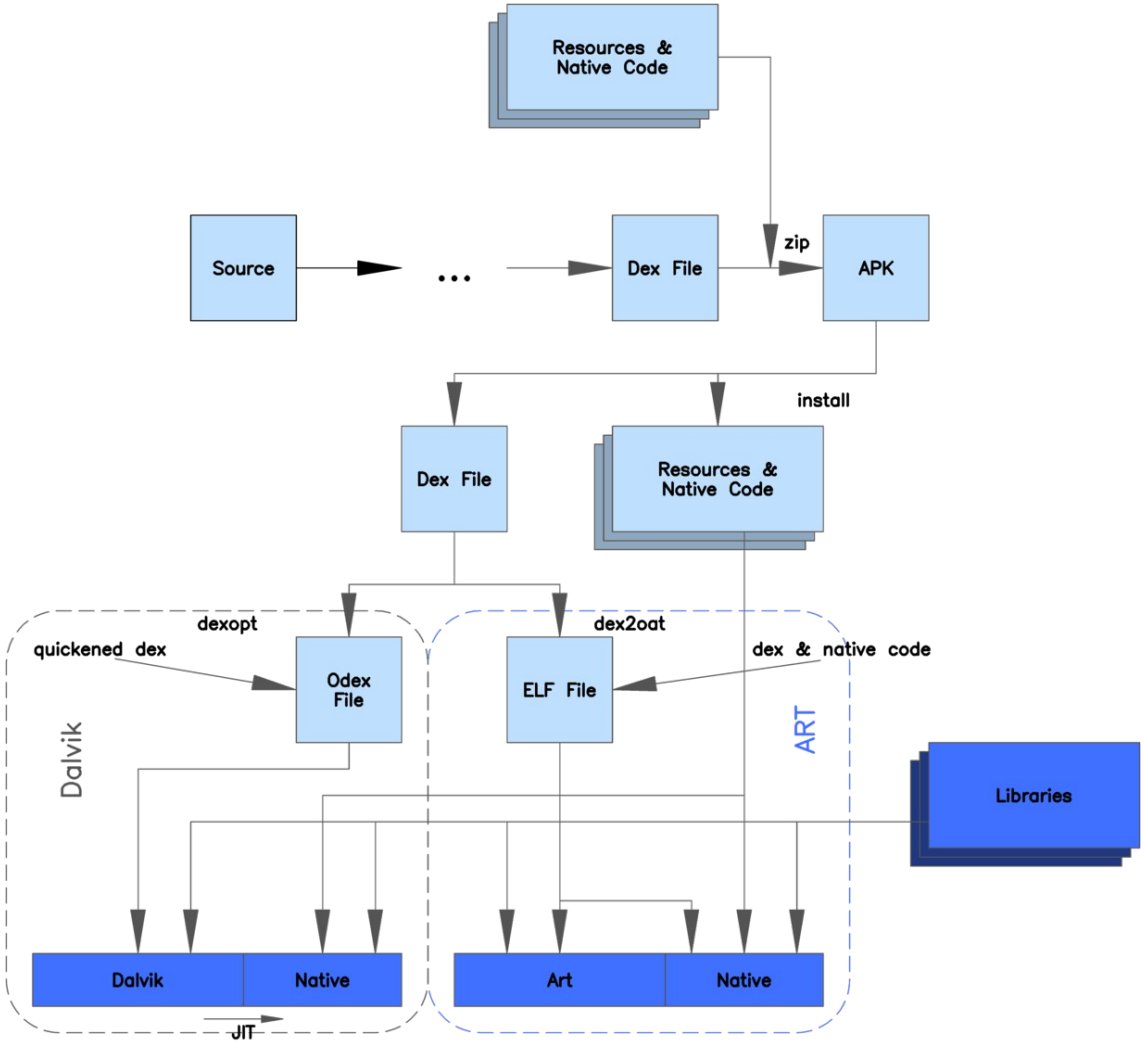
javac

```
public static void main(String args[]) {
  int a = 10;
  int b = a + 1 + 2 + 3 + 4 + 5 + 6;

  System.out.println(b);
}
```

```
0: bipush        10
2: istore_1
3: iload_1
4: iconst_1
5: iadd
6: iconst_2
7: iadd
8: iconst_3
9: iadd
10: iconst_4
11: iadd
12: iconst_5
13: iadd
14: bipush        6
16: iadd
17: istore_2
```

Dalvik VM / ART


Resources &
Native Code
Source
...
Dex File
zip
APK
install
Dex File
Resources &
Native Code
dexopt
quickened dex
Odex
File
dex2oat
ELF File
dex & native code
Dalvik
ART
Libraries
Dalvik
Native
Art
Native
JIT

Generated dex bytecode & native (by ART) are based in the original java bytecode

Examples

Do's & Don'ts

Autoboxing

Transparent to the developer but compiler adds some 'extra' code

Autoboxing

```
long total = 0;
for(int i = 0; i < N; i++) {
  total += i;
}
```

```
4: lconst_0
5: lstore_3
6: iconst_0
7: istore 5
9: iload 5
11: ldc #6;
13: if_icmpge 28
16: lload_3
17: iload 5
19: i2l
20: ladd
21: lstore_3
22: iinc 5,1
25: goto 9
```

Autoboxing

```
Long total = 0;
for(Integer i = 0; i < N; i++) {
  total += i;
}
```

```
9: iconst_0
10: invokestatic #4; //Method java/lang/Integer.valueOf:
    (I)Ljava/lang/Integer;
13: astore 4
15: aload 4
17: invokevirtual #5; //Method java/lang/Integer.intValue:()I
20: ldc #6; //int 10000000
22: if_icmpge 65
25: aload_3
26: invokevirtual #7; //Method java/lang/Long.longValue:()J
29: aload 4
31: invokevirtual #5; //Method java/lang/Integer.intValue:()I
34: i2l
35: ladd
36: invokestatic #3; //Method java/lang/Long.valueOf:
    (J)Ljava/lang/Long;
39: astore_3
40: aload 4
42: astore 5
44: aload 4
46: invokevirtual #5; //Method java/lang/Integer.intValue:()I
49: iconst_1
50: iadd
51: invokestatic #4; //Method java/lang/Integer.valueOf:
    (I)Ljava/lang/Integer;
54: dup
55: astore 4
57: astore 6
59: aload 5
61: pop
62: goto 15
```

Autoboxing

- This is what that code is actually doing:

```
Long total = 0;
for(Integer i = Integer.valueOf(0);
            i.intValue() < N;
            i = Integer.valueOf(i.intValue() + 1)) {

  total = Long.valueOf(total.longValue() + (long)i.intValue())
}
```

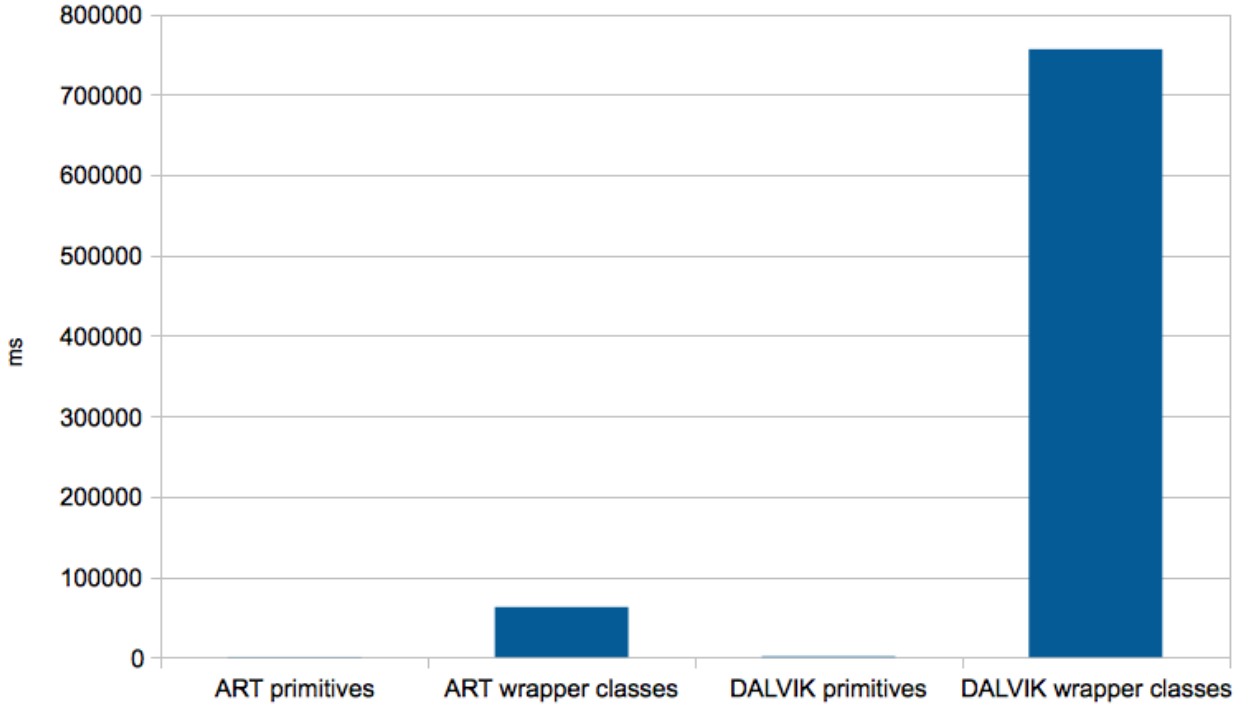
Autoboxing

Let's run that loop 100.000.000

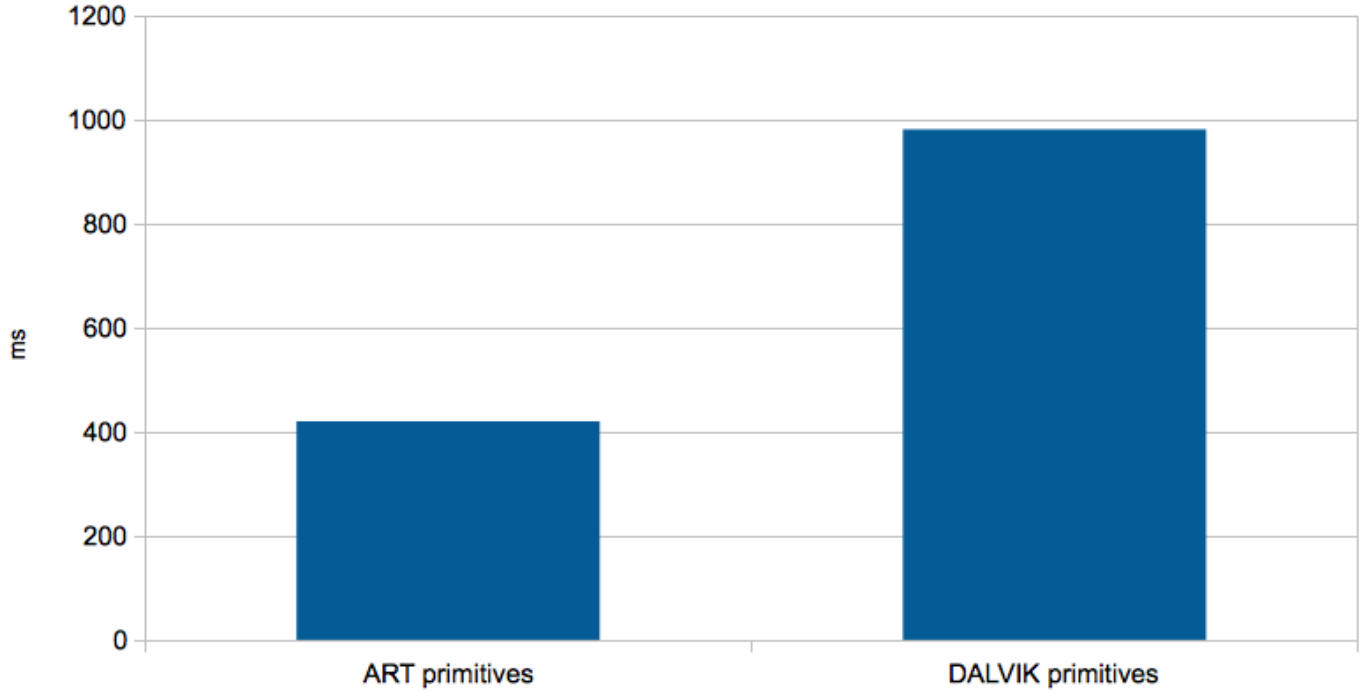
Times on two Nexus 5

Dalvik and ART

Autoboxing


800000
700000
600000
500000
400000
300000
200000
100000
0
ms
ART primitives
ART wrapper classes
DALVIK primitives
DALVIK wrapper classes

Autoboxing - details



Primitives vs Wrapper Classes

ART: 142x times

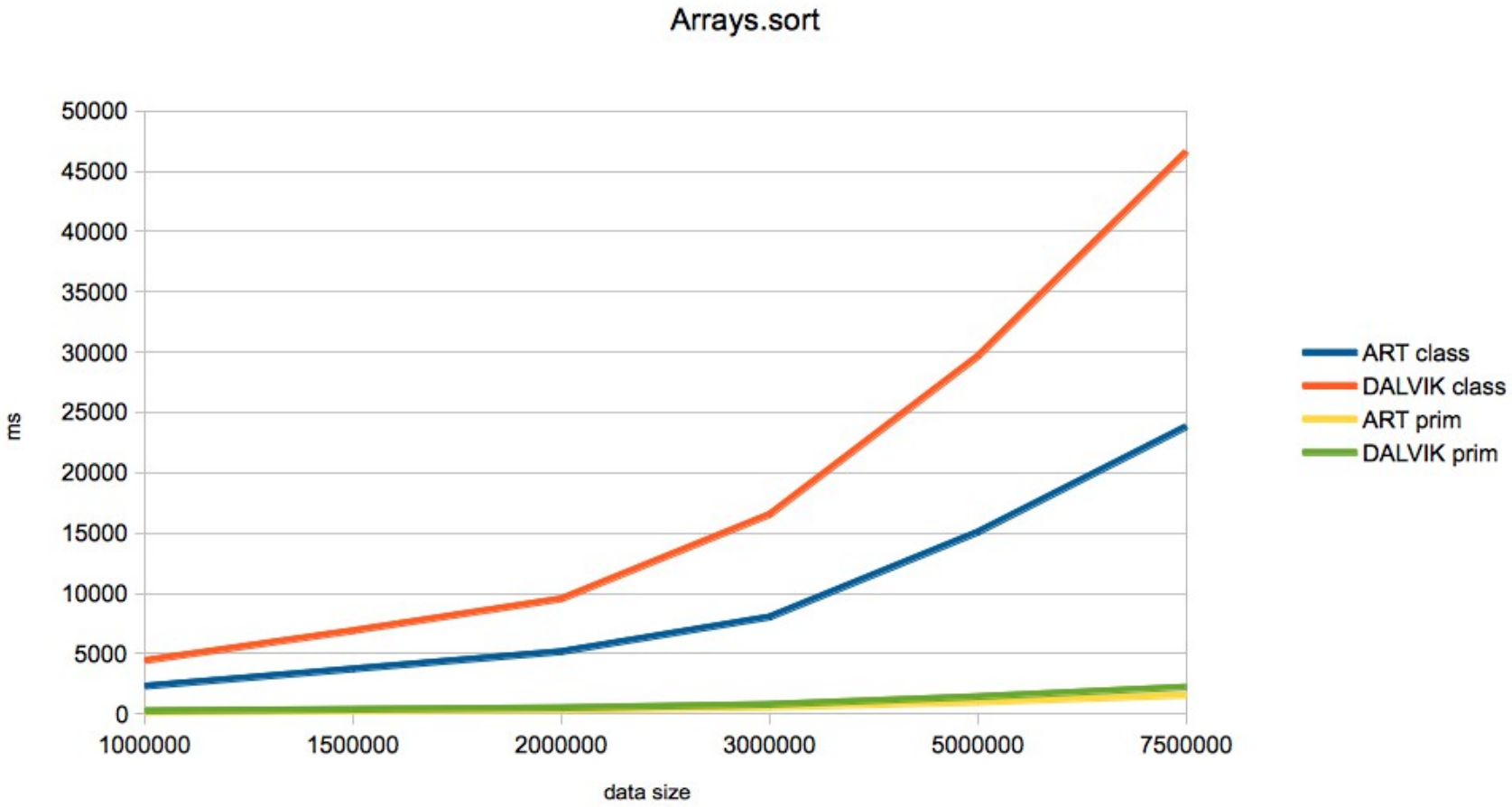
Dalvik: 771x times

ART vs Dalvik: 2,3x // 12.1x

Sorting

The easy way

Let's sort some numbers


Arrays.sort
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
ms
1000000
1500000
2000000
3000000
5000000
7500000
data size
ART class
DALVIK class
ART prim
DALVIK prim

Difference between sorting primitive types & objects

Sorting objects is a stable sort

Default java algorithm: TimSort (derived from MergeSort)

Sorting primitives doesn't require to be stable sort

Default java algorithm:
Dual-Pivot quicksort

Sorting

Use primitive types as much as possible

Loops

What's going on under the hood

Loops - List

```
ArrayList<Integer> list = new …
static long loopStandardList() {
    long result = 0;
    for(int i = 0; i < list.size(); i++) {
        result += list.get(i);
    }
    return result;
}
```

Loops - List (Java bytecode)

```
7: lload_0
8: getstatic     #26      // Field list:Ljava/util/ArrayList;
11: iload_2
12: invokevirtual #54     // Method java/util/ArrayList.get:(I)Ljava/lang/Object;
15: checkcast     #38     // class java/lang/Integer
18: invokevirtual #58     // Method java/lang/Integer.intValue:()I
21: i2l
22: ladd
23: lstore_0
24: iinc          2, 1
27: iload_2
28: getstatic     #26     // Field list:Ljava/util/ArrayList;
31: invokevirtual #61     // Method java/util/ArrayList.size:()I
34: if_icmplt     7
```

Loops - foreach

```
ArrayList<Integer> list = new …
static long loopForeachList() {
    long result = 0;
    for(int v : list) {
        result += v;
    }
    return result;
}
```

Loops - foreach (Java bytecode)

```
12: aload_3
13: invokeinterface #70,  1   // InterfaceMethod java/util/Iterator.next:()
18: checkcast     #38         // class java/lang/Integer
21: invokevirtual #58         // Method java/lang/Integer.intValue:()I
24: istore_2
25: lload_0
26: iload_2
27: i2l
28: ladd
29: lstore_0
30: aload_3
31: invokeinterface #76,  1   // InterfaceMethod java/util/Iterator.hasNext:()Z
36: ifne          12
```

Loops - Array

```
static int[] array = new ...
static long loopStandardArray() {
    long result = 0;
    for(int i = 0; i < array.length; i++) {
        result += array[i];
    }
    return result;
}
```

Loops - Array (Java bytecode)

```
7: lload_0
8: getstatic     #28                 // Field array:[I
11: iload_2
12: iaload
13: i2l
14: ladd
15: lstore_0
16: iinc          2, 1
19: iload_2
20: getstatic     #28                 // Field array:[I
23: arraylength
24: if_icmplt     7
```

Loops - size cached

```
static int[] array = new ...
static long loopStandardArraySizeStored() {
    long result = 0;   int length = array.length;
    for(int i = 0; i < length; i++) {
        result += array[i];
    }
    return result;
}
```

Loops - size stored (Java bytecode)

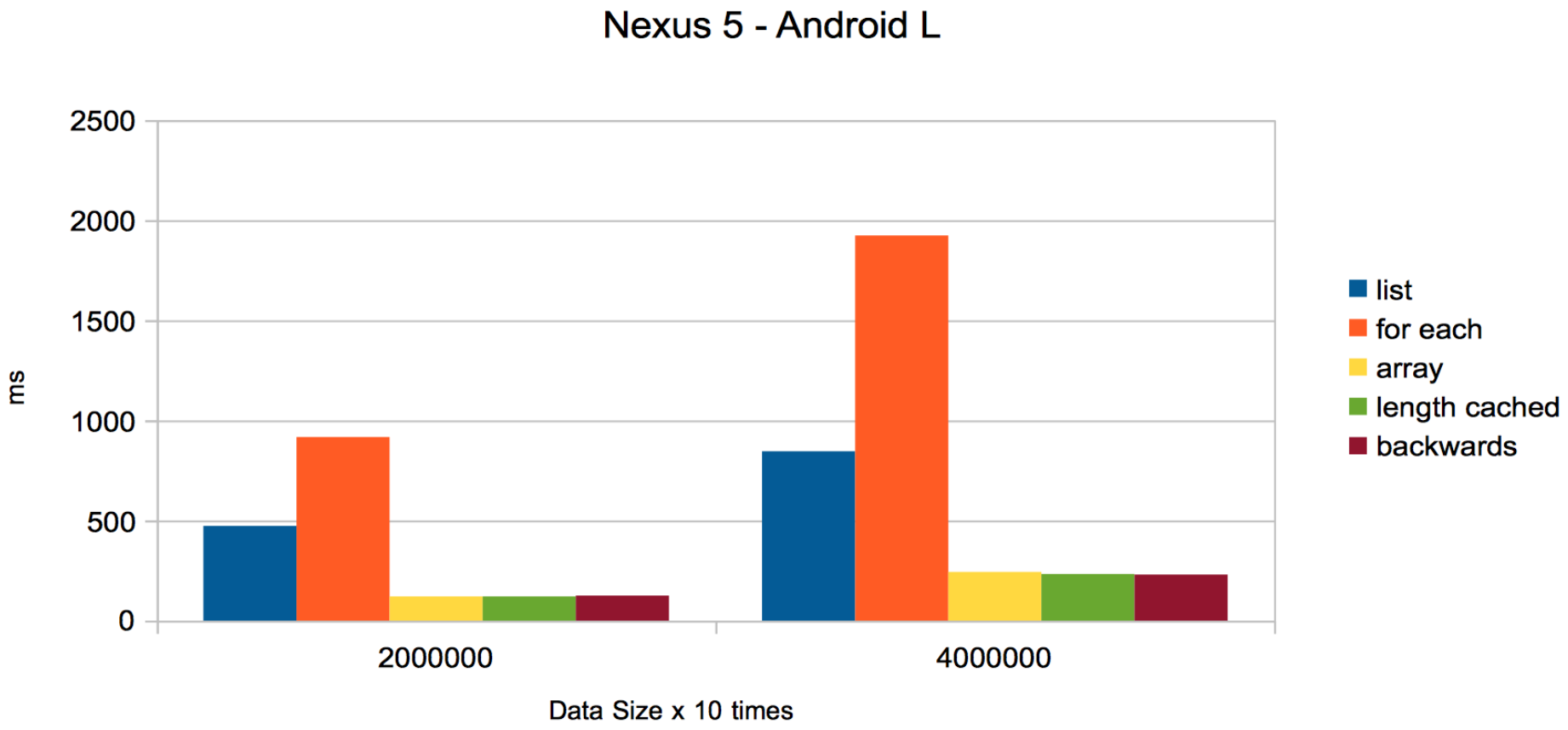
```
12: lload_0
13: getstatic     #28                 // Field array:[I
16: iload_3
17: iaload
18: i2l
19: ladd
20: lstore_0
21: iinc          3, 1
24: iload_3
25: iload_2
26: if_icmplt     12
```

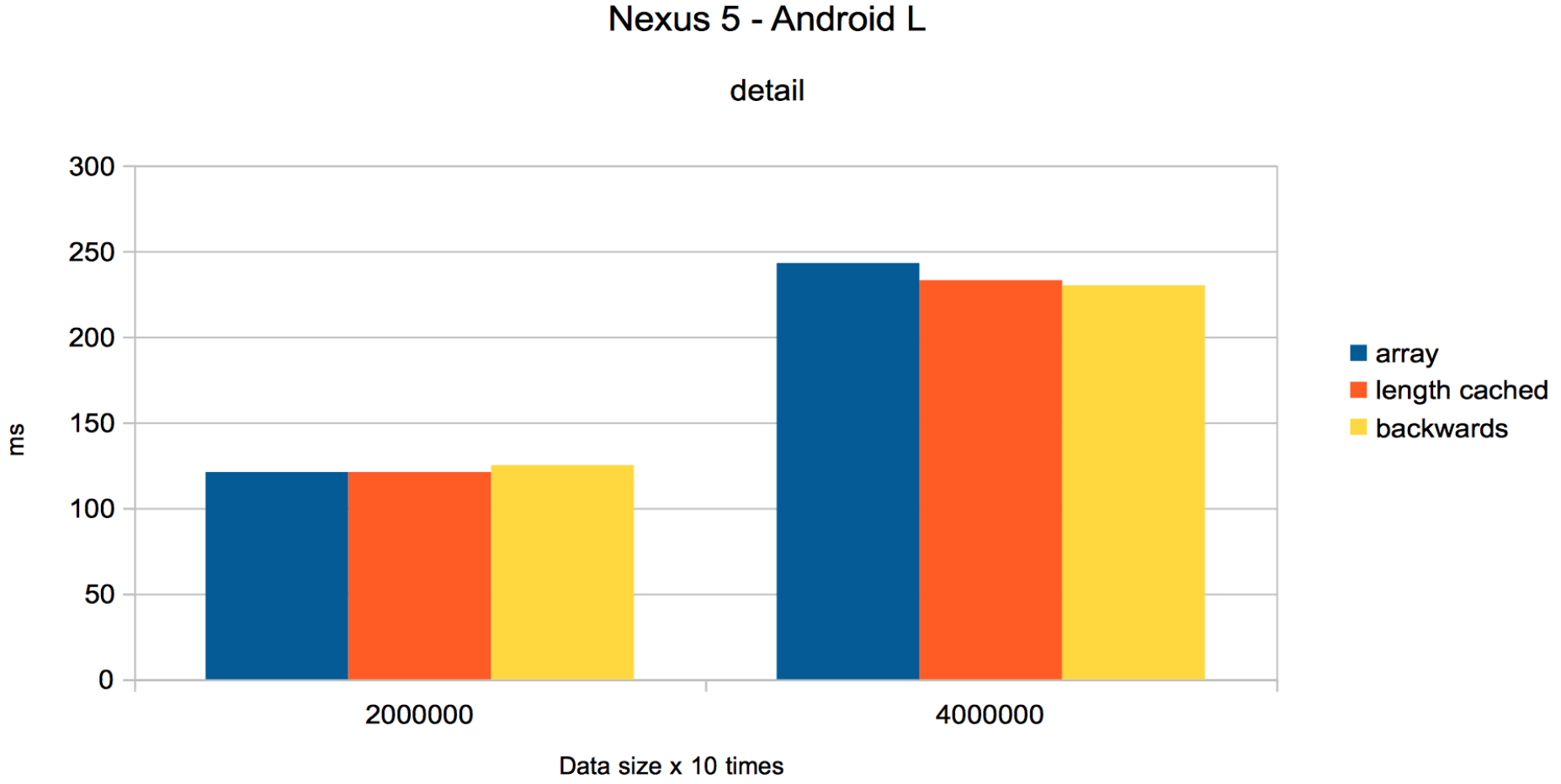
Loops - backwards

```
static int[] array = new ...
static long loopStandardArrayBackwards() {
  long result = 0;
  for(int i = array.length - 1; i >= 0; i--) {
        result += array[i];
  }
  return result;
}
```

Loops - backwards (Java bytecode)

```
12: lload_0
13: getstatic     #28                 // Field array:[I
16: iload_2
17: iaload
18: i2l
19: ladd
20: lstore_0
21: iinc          2, -1
24: iload_2
25: ifge          12
```


Nexus 5 - Android L
2500
2000
1500
1000
500
0
ms
2000000
4000000
Data Size x 10 times
list
for each
array
length cached
backwards


Nexus 5 - Android L
detail
300
250
200
150
100
50
0
ms
2000000
4000000
Data size x 10 times
array
length cached
backwards

Loops

Avoid foreach constructions as much as possible

Loops

Use only backwards loop
if makes your life easier
(be careful with cache)

Calling a method

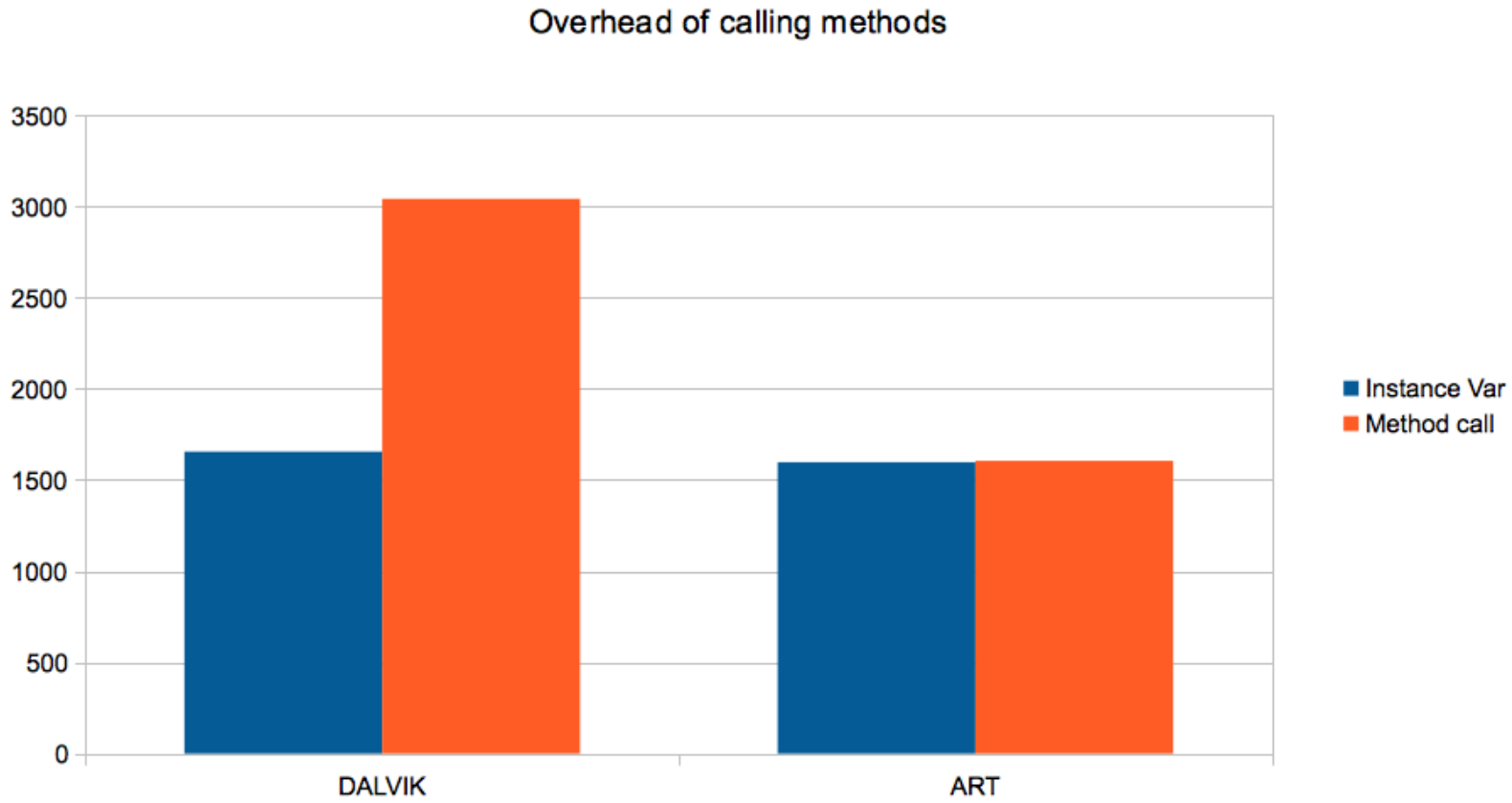
Is there an overhead?

Calling a method overhead

```
for(int i = 0; i < N; i++) {
    setVal(getVal() + 1);
}
```

VS

```
for(int i = 0; i < N; i++) {
    val = val + 1;
}
```


Overhead of calling methods
3500
3000
2500
2000
1500
1000
500
0
DALVIK
ART
Instance Var
Method call

String concatenation

The evil + sign

String concatenation

```
String str = "";
for(int i = 0;  i  < ITERATIONS; i++) {
  str += ANY_OTHER_STRING;
}
```

String concatenation

```
8: new           #26           // class java/lang/StringBuilder
11: dup
12: aload_1
13: invokestatic  #28          // Method java/lang/String.valueOf:
  (Ljava/lang/Object;)Ljava/lang/String;
16: invokespecial #34          // Method java/lang/StringBuilder."<init>":(Ljava/lang/String;)V
19: ldc           #11          // String ANY_OTHER_STRING
21: invokevirtual #37          // Method java/lang/StringBuilder.append:(Ljava/lang/String;)
24: invokevirtual #41          // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
27: astore_1
28: iinc          2, 1
31: iload_2
32: bipush        ITERATIONS
34: if_icmplt     8
```

String concatenation

```
String str = "";
for(int i = 0;  i  < ITERATIONS; i++) {
    StringBuilder sb = new StringBuilder(String.valueOf(str));
    sb.append(ANY_OTHER_STRING);
    str = sb.toString();
}
```

String concatenation

alternatives

String.concat()

- Concat cost is O(N) + O(M) - (N,M) length of each String
- Concat returns a new String Object.

```
String str = "";
for(int i = 0;  i  < ITERATIONS; i++) {
  str = str.concat(ANY_OTHER_STRING);
}
```

StringBuilder

- StringBuffer.append cost is O(M) amortized time (M length of appended String)
- Avoids creation of new objects.

```
StringBuilder sb = new StringBuilder()
for(int i = 0;  i  < ITERATIONS; i++) {
  sb.append(ANY_OTHER_STRING);
}
str = sb.toString();
```

String concatenation

Use StringBuilder (properly) as much as possible. StringBuffer is the thread safe implementation.

Tooling

Tooling - Disassembler

Java

- javap -c <classfile>

Android:

- Dexdump -d <dexfile>
- Smali - https://code.google.com/p/smali/

Tooling – Disassembler - ART

adb pull /data/dalvik-cache/arm/data@app@<package>-1@base apk@classes.dex

gobjdump -D <file>

Tooling – Disassembler - ART

adb shell oatdump --oat-file=/data/dalvik-cache/arm/data@app@<package>-1@base.apk@classes.dex

Tooling - Obfuscation

Obfuscation not only make your code harder to hack, but also optimizes your bytecode!

Performance measurements

Avoid doing multiple tests in one run

JIT might be evil!

Do not trust the compiler!

@rrafols

http://blog.rafols.org