BELLSOFT

Дорогая, попробуем Arm?

Александр Белокрылов, Алексей Войтылов

WWW.BELL-SW.COM

2018

Кто здесь?

Александр Белокрылов

@gigabel

BELLSOFT

Liberica JDK – supported Java binaries
http://bell-sw.com

Бывшие работодатели:

ORACLE

Sun microsystems

Есть еще кто?

Алексей Войтылов

@AVoitylov

BELLSOFT

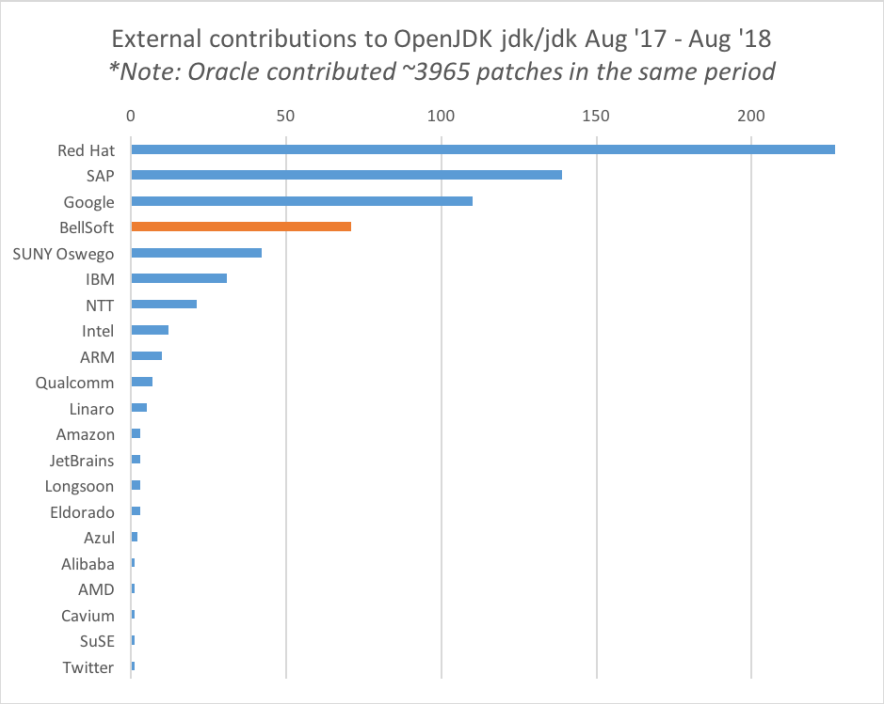
Liberica JDK – supported Java binaries
http://bell-sw.com

Бывшие работодатели:

ORACLE

Sun microsystems

BellSoft


External contributions to OpenJDK jdk/jdk Aug '17 - Aug '18
*Note: Oracle contributed ~3965 patches in the same period
0
50
100
150
200
Red Hat
SAP
Google
BellSoft
SUNY Oswego
IBM
NTT
Intel
ARM
Qualcomm
Linaro
Amazon
JetBrains
Longsoon
Eldorado
Azul
Alibaba
AMD
Cavium
SuSE
Twitter

Действующие лица

- **Дорогая** – руководитель IT департамента крупного предприятия по добыче и переработки газа, система управления которого работает на Java.
- **Дорогой** - его заместитель, по совместительству ответственный за внедрение новых технологий.

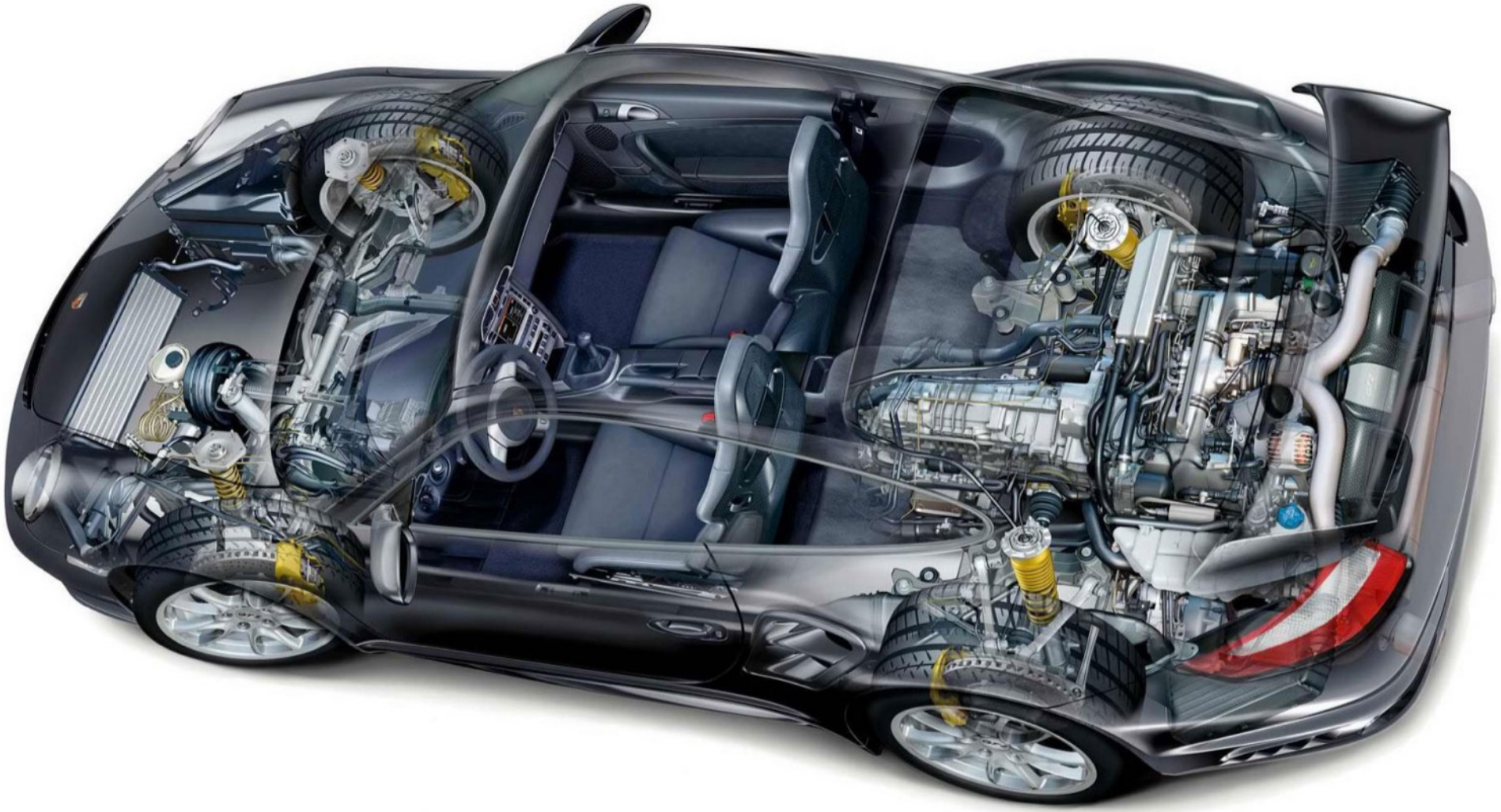
Дорогая, попробуем ARM?

Что мы знаем об Arm?

- Arm = Advanced RISC Machines
- основана в 1990 году
- Англия, Кембридж
- ARM = RISC архитектура

- 30 миллиардов процессоров 2013
- Планирует продать 100 миллиардов процессоров к 2020


11:59
Mon, February 24
search
11:59
Search
14:59 PM
Now playing
"AMERICAN MOUTH"
Try The Rock Orchestra
1:22
3:05
14:59 PM
1 new message



IoT Gateways

Liberica JDK

SuperMicro

Dell

Eurotech

Advantech

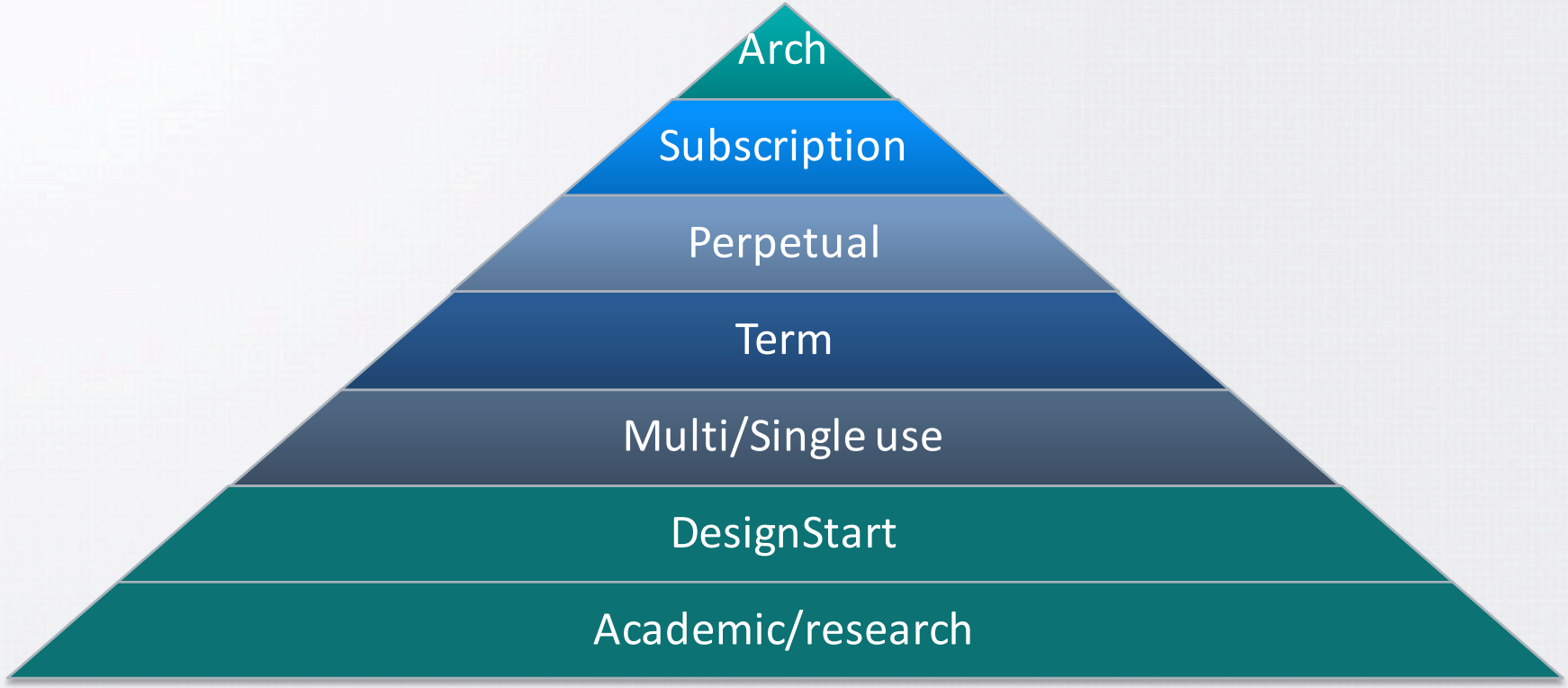
Серверы

Модель распространения Arm

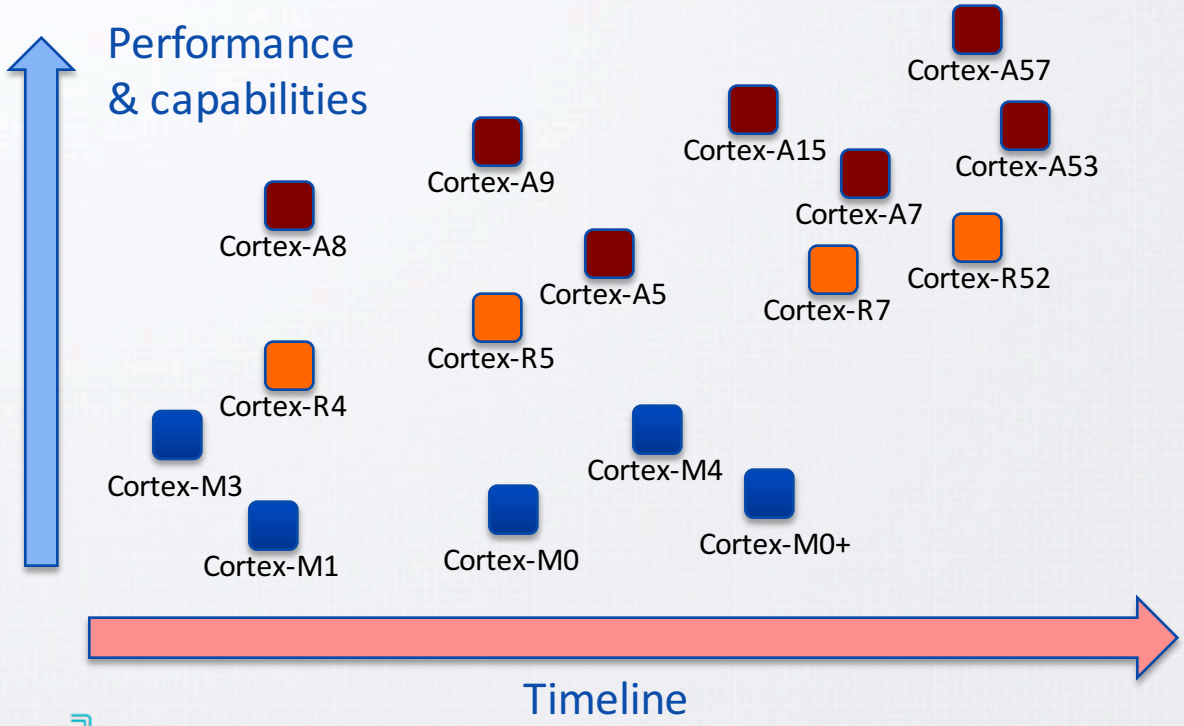
ARM

Производитель процессоров

Лицензирование Arm



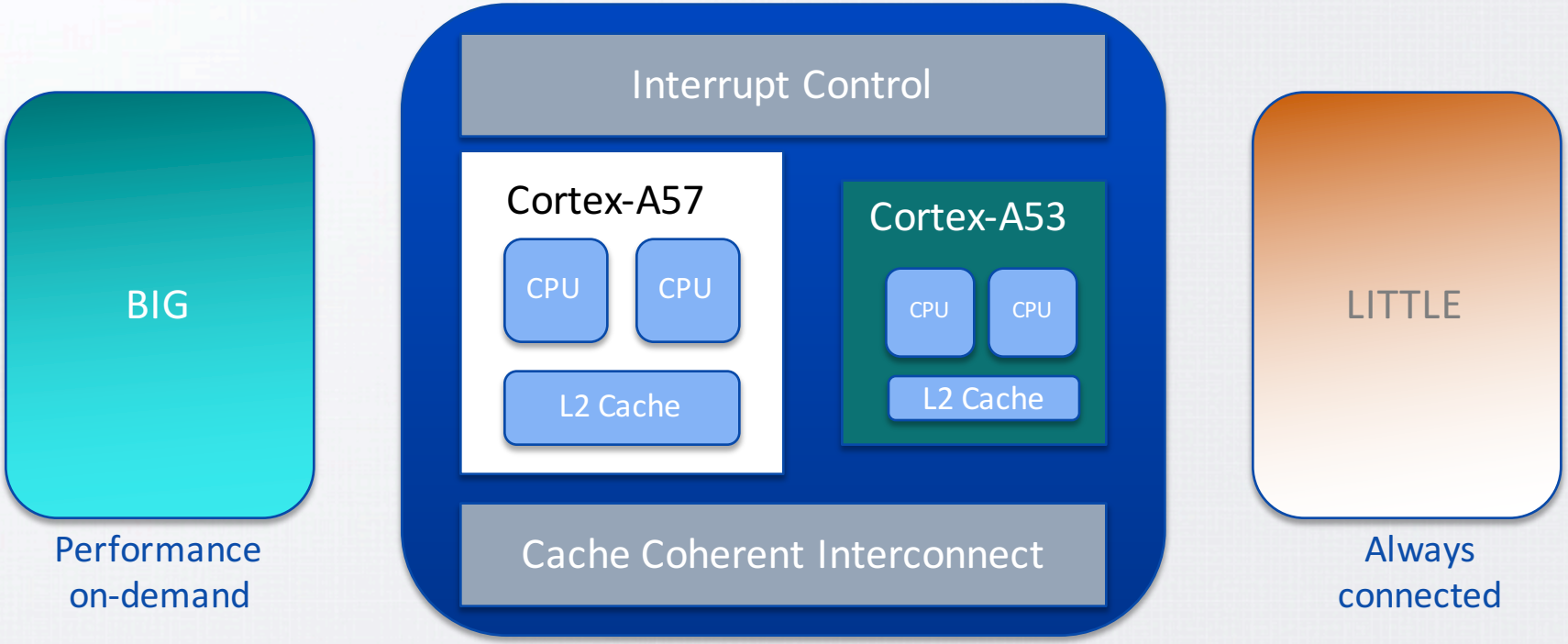
Arm: архитектура, профайл и имплементация



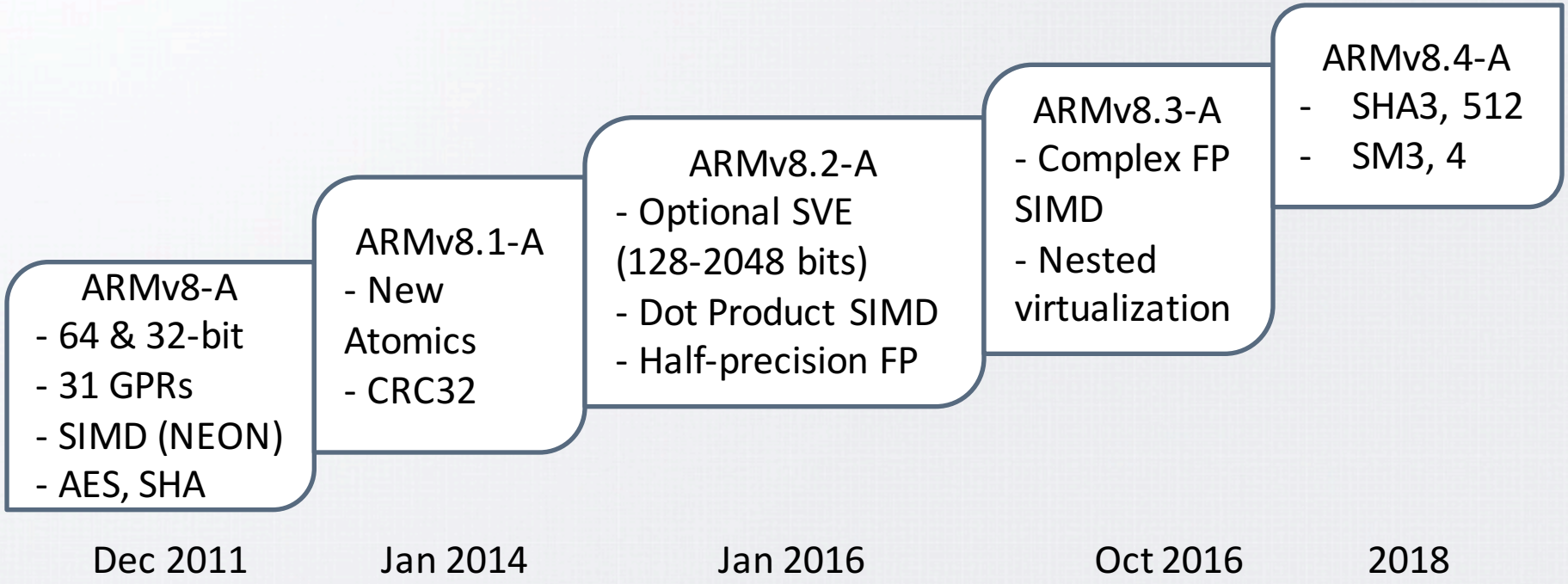
- ARM v7
- Architecture profiles
  - v7-M (Embedded)
  - V7-R (Real-Time)
  - V7-A (Application)

- ARM v8
- Architecture profiles
  - v8-M (Embedded)
  - V8-R (Real-Time)
  - V8-A (Application)

Arm: big.LITTLE

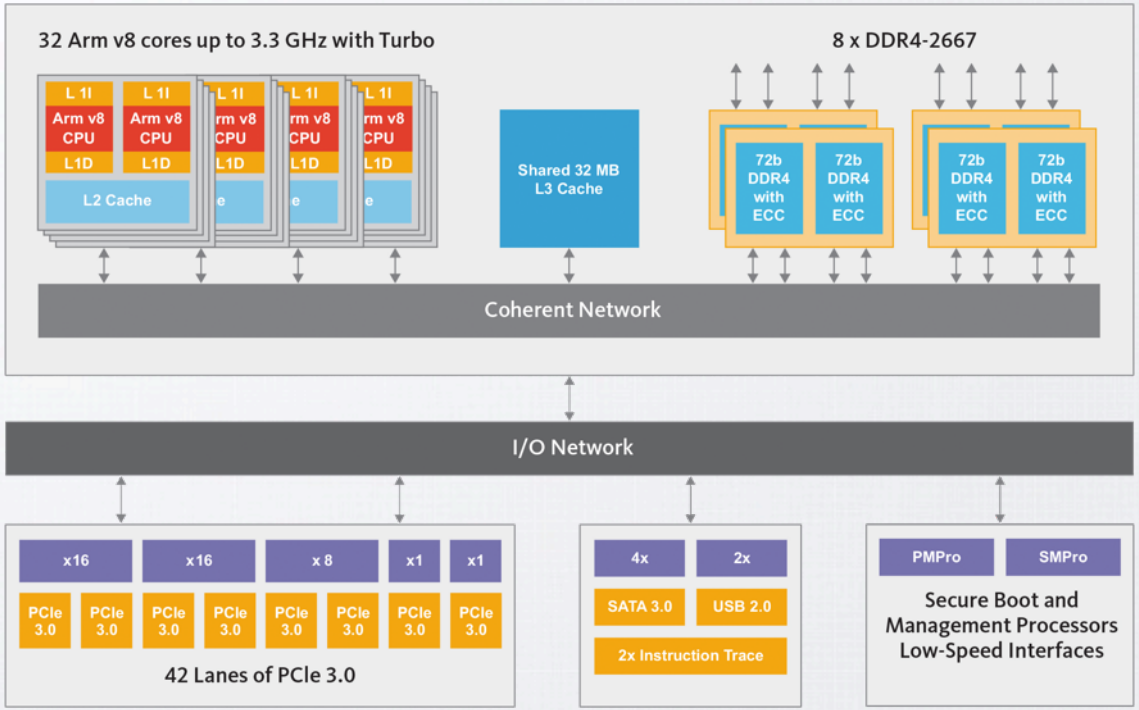


ARMv8-A Specification



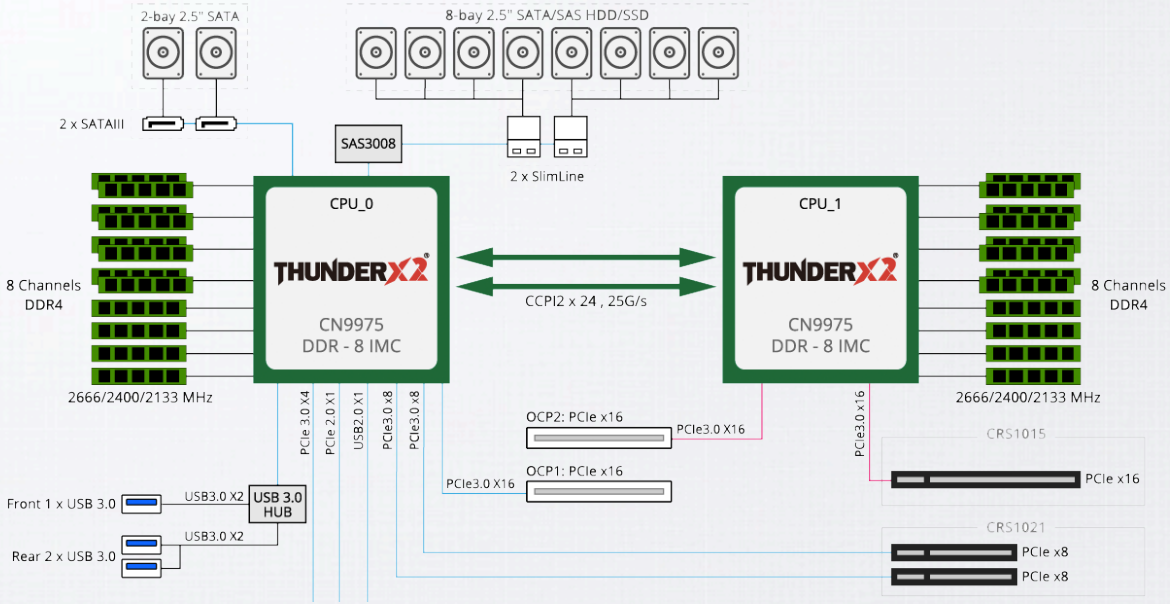
Архитектурные лицензенты

Ampere Computing (ex APM)



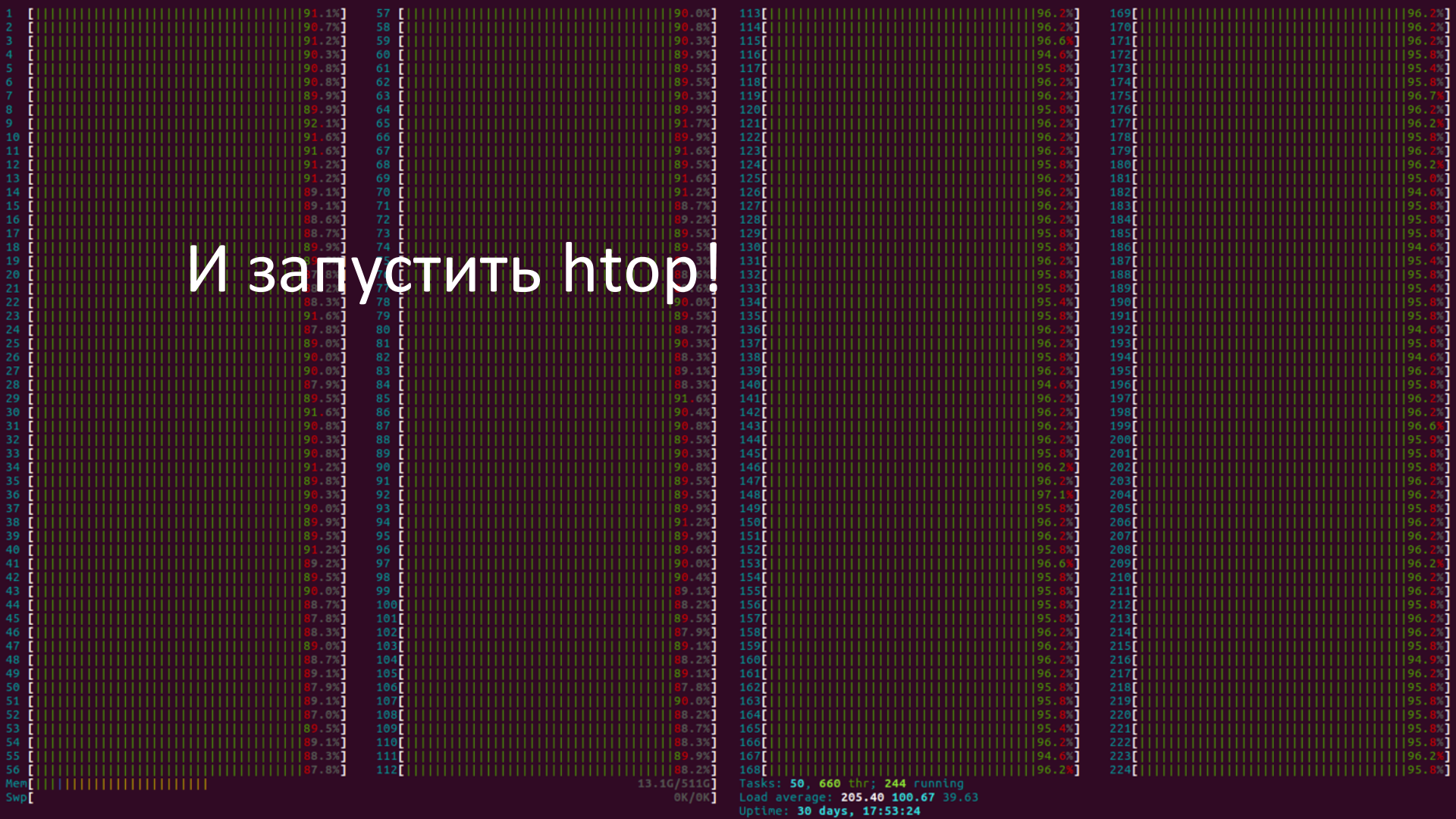
Up to 32 cores
Up to 32 threads
8 DDR Channels
32 Mb L3

Cavium/Marvell ThunderX2



32 cores/128 threads
32 Mb L3
8 DDR Channels/socket
Multi-socket
Up to 4 TB RAM

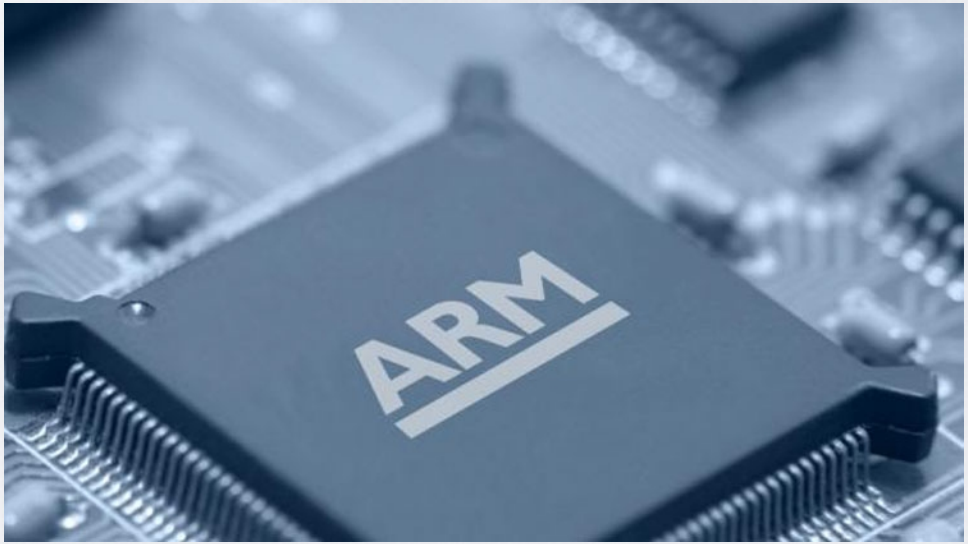

Можно
и потрогать!
MARVE
LIBERICA
Process.


И запустить htop!
Mem[
Swp[
13.1G/511G]
0K/0K]
Tasks: 50, 660 thr; 244 running
Load average: 205.40 100.67 39.63
Uptime: 30 days, 17:53:24

Arm Software ecosystem

OpenJDK ARM ports

- ARM
  - 32 bit / “64 bit”
    - ARM v6
    - ARM v7
    - ARM v8
- AARCH64
  - 64 bit only



Intrinsics

Intrinsic:

“function (subroutine) available for use in a given programming language which implementation is handled specially by the compiler.”

Intrinsics

- GCC/LLVM
  - Специализированные инструкции не выражаемые посредством конструкций языка
  - Обертки над вызовами libc
    - Что происходит при компиляции программы на C с вызовом strlen()?
- HotSpot
  - Манипулирование с C2 IR
    - Обычно вызов специализированной ASM инструкции для данной архитектуры
  - Stub – ассемблерные или нативные вставки
    - Подставляются вместо кода java
    - Не работает с JVMTI/JDWP
    - Универсальны (C1/C2/Interpreter)
    - Код под них аллоцируется один раз
    - Стоимость вызова не 0

Пример в студию!

Что сделает C2 из математического кода на Java?

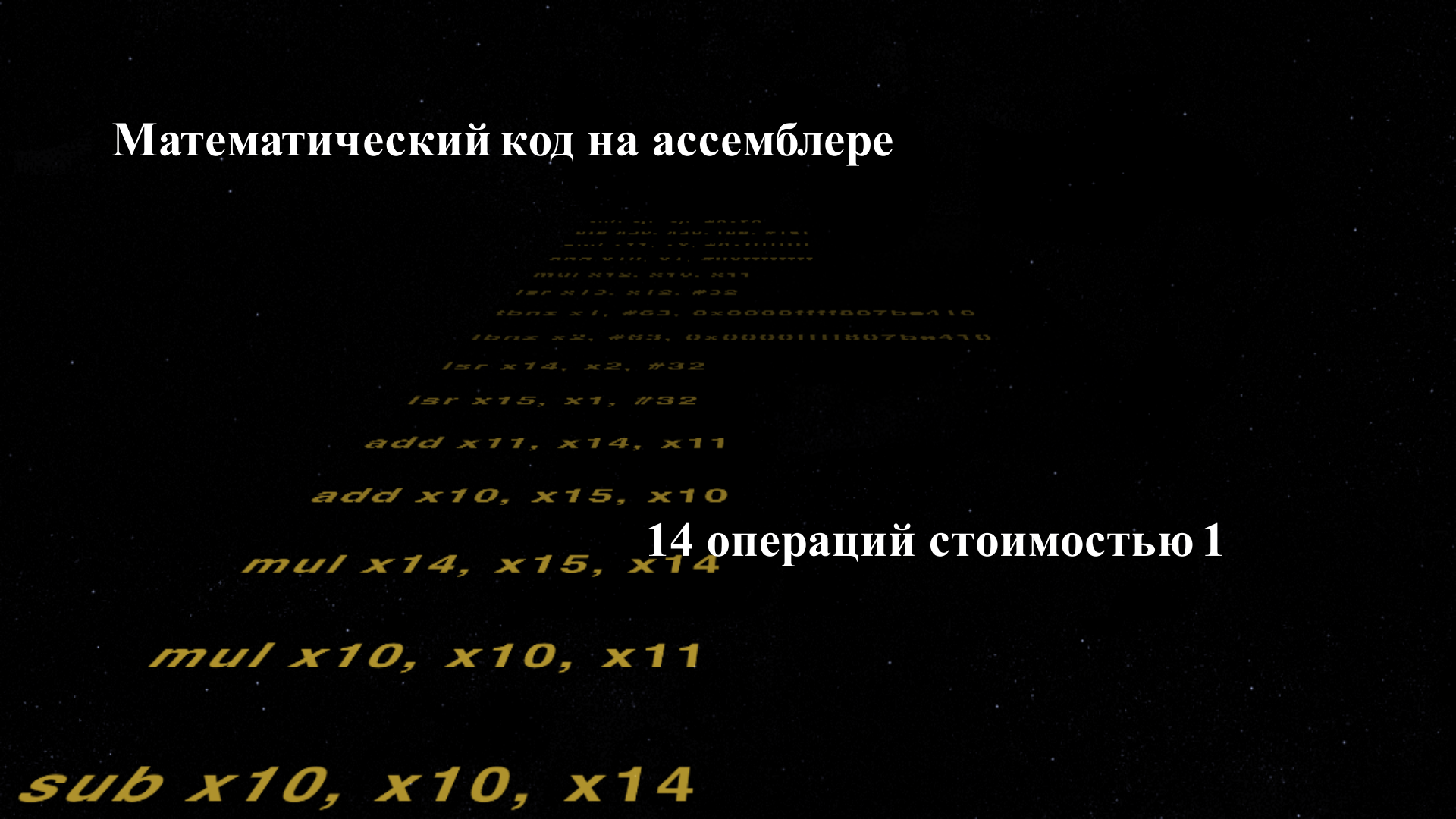
java.lang.Math:

```
/**
 * Returns as a {@code long} the most significant 64 bits of the
 * 128-bit product of two 64-bit factors.
 * @since 9
 */
public static long multiplyHigh(long x, long y) {
   if (x < 0 || y < 0) {
        long x1 = x >> 32;
        long x2 = x & 0xFFFFFFFFL;
        long y1 = y >> 32;
        long y2 = y & 0xFFFFFFFFL;
        long z2 = x2 * y2;
        long t = x1 * y2 + (z2 >>> 32);
        long z1 = t & 0xFFFFFFFFL;
        long z0 = t >> 32;
        z1 += x2 * y1;
        return x1 * y1 + z0 + (z1 >> 32);
    } else { …
```

Что сделает C2 из математического кода на Java?

```
java.lang.Math:
/**
 * Returns as a {@code long} the most significant 64 bits of the 128-
bit
 * product of two 64-bit factors.
 * @since 9
 */
public static long multiplyHigh(long x, long y) {
    // Use technique from section 8-2 of Henry S. Warren, Jr.,
    // Hacker's Delight (2nd ed.) (Addison Wesley, 2013), 173-174.
    ...
    // Use Karatsuba technique with two base 2^32 digits.
    ...
    return ...;
}
```

Математический код на ассемблере



14 операций стоимостью 1

Может, можно быстрее?

- Переписать на C + JNI call
  - Будет медленнее
- Научить HotSpot оптимизировать IR этого кода*
  - Даже если получится, будут регрессии
- Научить HotSpot распознавать этот метод и подставлять вместо него оптимальный код

```
SMULH Xd, Xn, Xm
```

“Signed multiply high”

C2 Intrinsic How-to

1) Добавляем инструкцию `SMULH` в `${arch}/assembler_${arch}.hpp`
2) Описываем ноду с инструкцией и ее стоимостью в `${arch}.ad`
3) Помечаем в `share/classfile/vmSymbols.hpp` метод как intrinsic
4) Подстановка в IR с инлайнингом

```
bool LibraryCallKit::inline_math_multiplyHigh() {
    set_result(_gvn.transform(new MulHiLNode(arg (0), arg (2))));
    return true;
}
```

5) Аннотируем `j.l.Math.multiplyHigh()` `@HotSpotIntrinsicCandidate`
6) Измеряем производительность

Benchmarking (latency)

```
public class MultiplyHighJMHBench {

    @Benchmark
    @OperationsPerInvocation(10001)
    public long bench() {
        long op1 = 1L;
        long op2 =10L;
        for (int i = 0; i < 10000; i++) {
            op1 = Math.multiplyHigh(op1, op2++);
        }
        return Math.multiplyHigh(op1, op2);
    }
}
```

2.5x better

SMULH cost: 4

Benchmarking (throughput)

```
public class MultiplyHighJMHBench {

    @Benchmark
    @OperationsPerInvocation(10000)
    public long bench() {
        long op = System.currentTimeMillis();
        long accum = 0;
        for (int i = 0; i < 10000; i++) {
            accum += Math.multiplyHigh(op + i, op + i);
        }
        return accum;
    }
}
```

3.5x better

Заливаем в JDK 11!

Польза для народного хозяйства

- Что делает JVM при исполнении усредненной Enterprise программы?
  - Создает, копирует объекты, строки, массивы, освобождает память
  - Ищет или сравнивает объекты, строки, массивы
  - Проверяет что получена или отправлена верная информация

String s = new String(“Can this work faster?”);

- Compact Strings @since JDK 9
  - подавляющее большинство строк не требуют UTF-16 для хранения
  - Внутреннее представление строк:
    - char[] -> byte[], coder
    - Либо ISO-8859-1/Latin-1

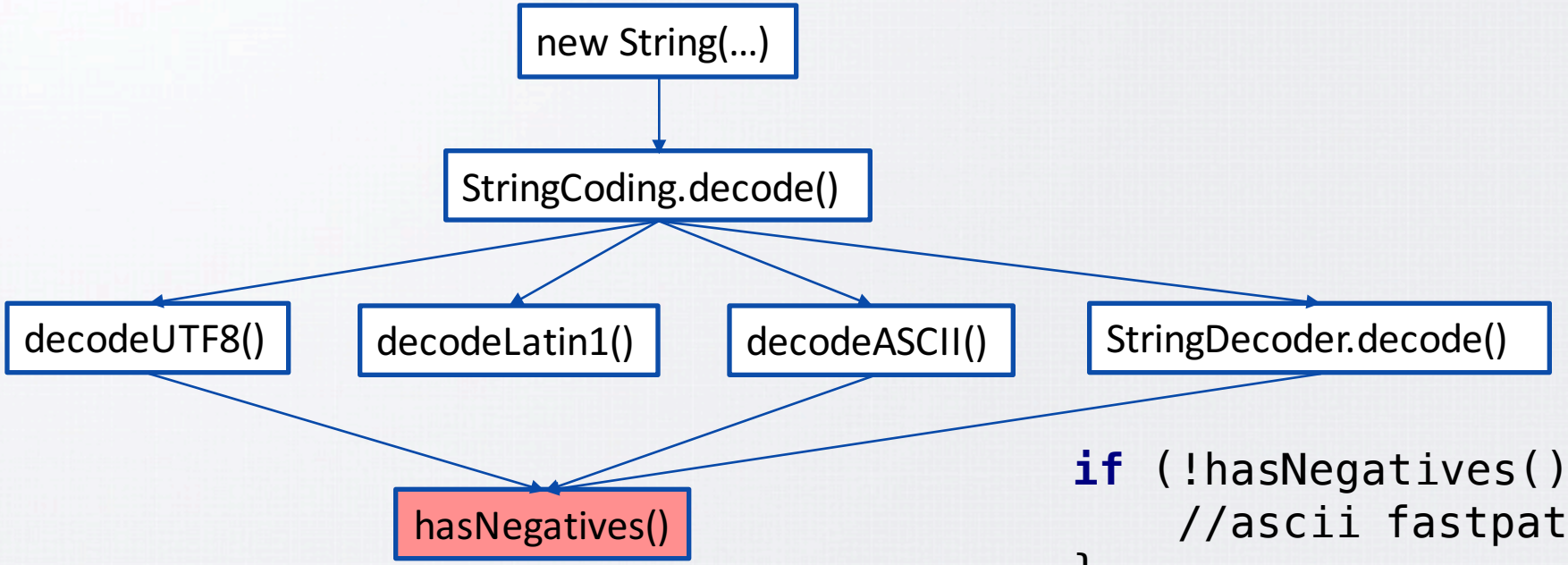
    - Либо UTF-16 если требуется

1001 Heap Dump

- Лог-нормальное распределение
- $< 0.3\%$ всех строк не Latin-1
- 18% строк < 8 символов
- 66% строк < 32 символов
- 95% строк < 128 символов

Изменения не должны сделать хуже этому датасету

String s = new String("Can this work faster?");




```
if (!hasNegatives()){
    //ascii fastpath
}
```

StringCoding.hasNegatives()


```
@HotSpotIntrinsicCandidate
public static boolean hasNegatives(byte[] ba, int off, int len) {
    for (int i = off; i < off + len; i++) {
        if (ba[i] < 0) {
            return true;
        }
    }
    return false;
}
```

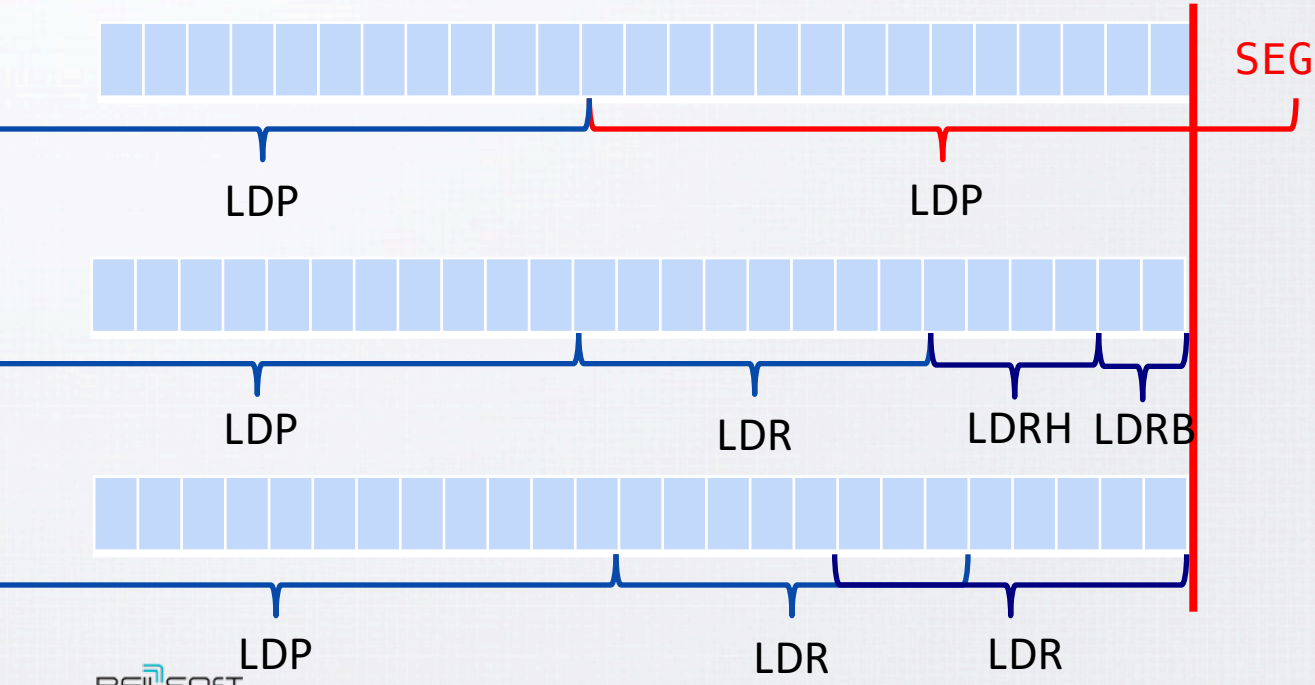
Немного ассемблера ARM – чтение из памяти

	Регистр	Ширина (байты)	Задержка (циклы)
LDRB	GPR	8	4
LDRH	GPR	16	4
LDR	GRP	32	4
LDP	FP/SIMD	128	5

нельзя просто так взять

и начать читать следующую страницу

Учимся заново читать

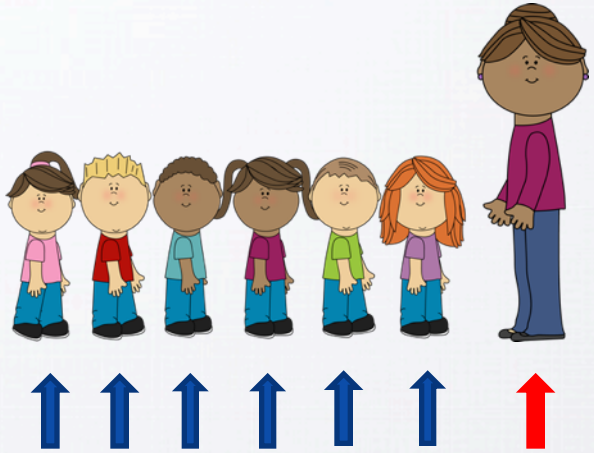


И сравнивать 8 байт за раз с 0

```
for(int i = off; i < off + len; i++) {
        if (ba[i] < 0) {
            return true;
        }
}
```

```
const uint64_t UPPER_BIT_MASK=0x8080808080808080;
...
__ tst(rscratch2, UPPER_BIT_MASK);
```

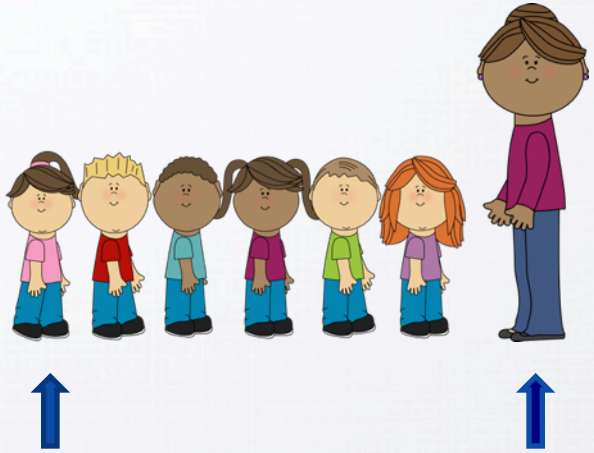
Выровнять чтение из памяти



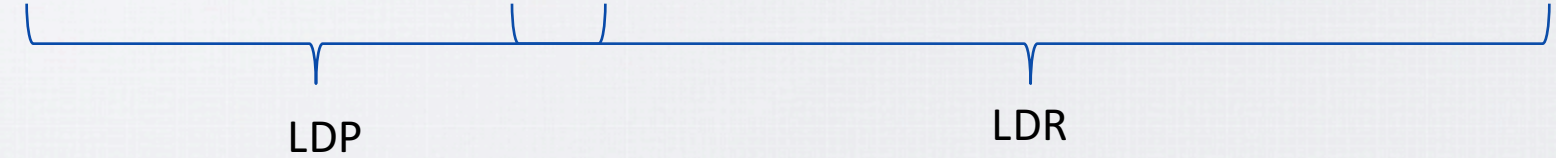
x86:
- в большинстве случаев на современных процессорах нет штрафа на unaligned memory access

ARM это спецификация:
- у одних производителей CPU нет штрафа
- у других есть (20%, 50%, 100%)

Выровнять чтение из памяти



```
// pre-loop
   __ ldp();
   …
   __ tst(…, UPPER_BIT_MASK);
// main loop
   __ ldr(); //aligned
   …
   __ tst(…, UPPER_BIT_MASK);
```



Итак, наш коварный план

- Читать как можно больше байт за раз, не выходя за пределы страницы
  - Если близко край страницы
    - Читать меньше байт
    - Сдвигать чтение влево
- Сравнивать как можно больше байт за раз
- Выровнять чтение из памяти

- Реальность
  - Код получается слишком большой – 200 инструкций
    - Это мешает инлайнингу: C2 инлайнит до 1500 инструкций

Код слишком большой – что делать?

взять все, да и поделить!

Код слишком большой – что делать?

- Псевдокод ARM ASM на Java, который меньше исходного (27 инструкций)
  - не оптимальный, unaligned, но короткий

```
if (len > 32)
    return stubHasNegatives(ba, 0, len);
for (int i = 0; i < 32; i++) {
    if (ba[i] < 0) {                  // ldr, tst
        return true;
    }
}
return stubHasNegatives(ba, 32, len); // ldp, tst
```

- Весь остальной код – в stub

Что такое stub?

- Тип ассемблерных вставок в HotSpot
- Ближайшая аналогия – функция
  - Его можно вызывать из macroAssembler
  - Код подгружается в момент старта JVM один раз
  - Не инлайнится
- Возможны несколько точек входа
- Стоимость вызова stub не 0

А что в stub?

```
// align memory access
__ bind(LARGE_LOOP); // 64 byte at a time
4x   __ ldp(); //ary1, ary1+16, ary1+32, ary1+48
     __ add(ary1, ary1, large_loop_size);
     __ sub(len, len, large_loop_size);
7x   __ orr(…);
     __ tst(tmp2, UPPER_BIT_MASK);
     __ br(Assembler::NE, RET_TRUE);
     __ cmp(len, large_loop_size);
     __ br(Assembler::GE, LARGE_LOOP);
```

ОК, мы помогли C2. Процессор тоже не всегда молодец. Поможем ему?

Software Prefetching

Подскажем процессору, откуда мы будем читать из памяти в следующий раз:

```
__ prfm(Address(ary1, SoftwarePrefetchHintDistance));
   // do local register or operations on data in cache
__ ldp();
```

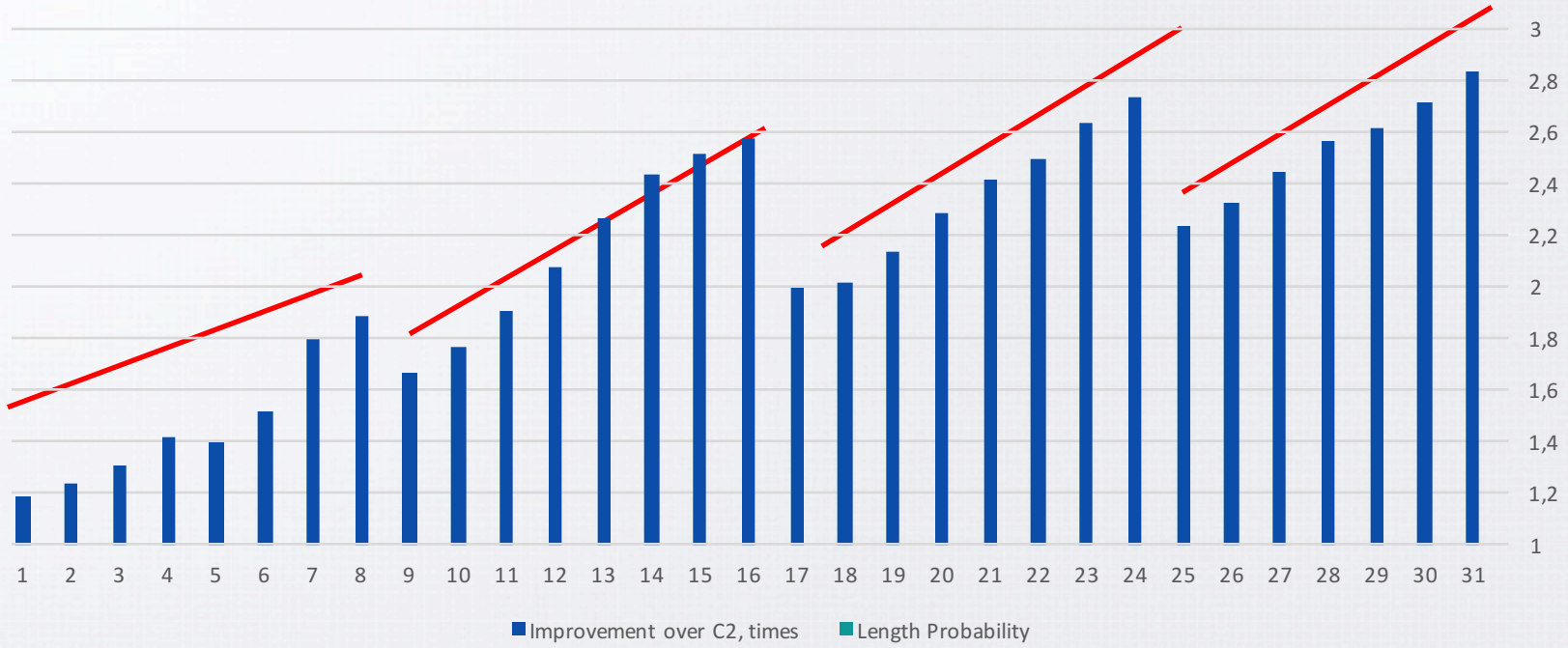
- Можно очень много выиграть в производительности если
  - Есть операции, которые процессор может выполнять пока идет загрузка (обычно цикл)
  - Правильно определить `SoftwarePrefetchHintDistance:`
    `> d_cache`

Бенчмарк для new String() – много символов

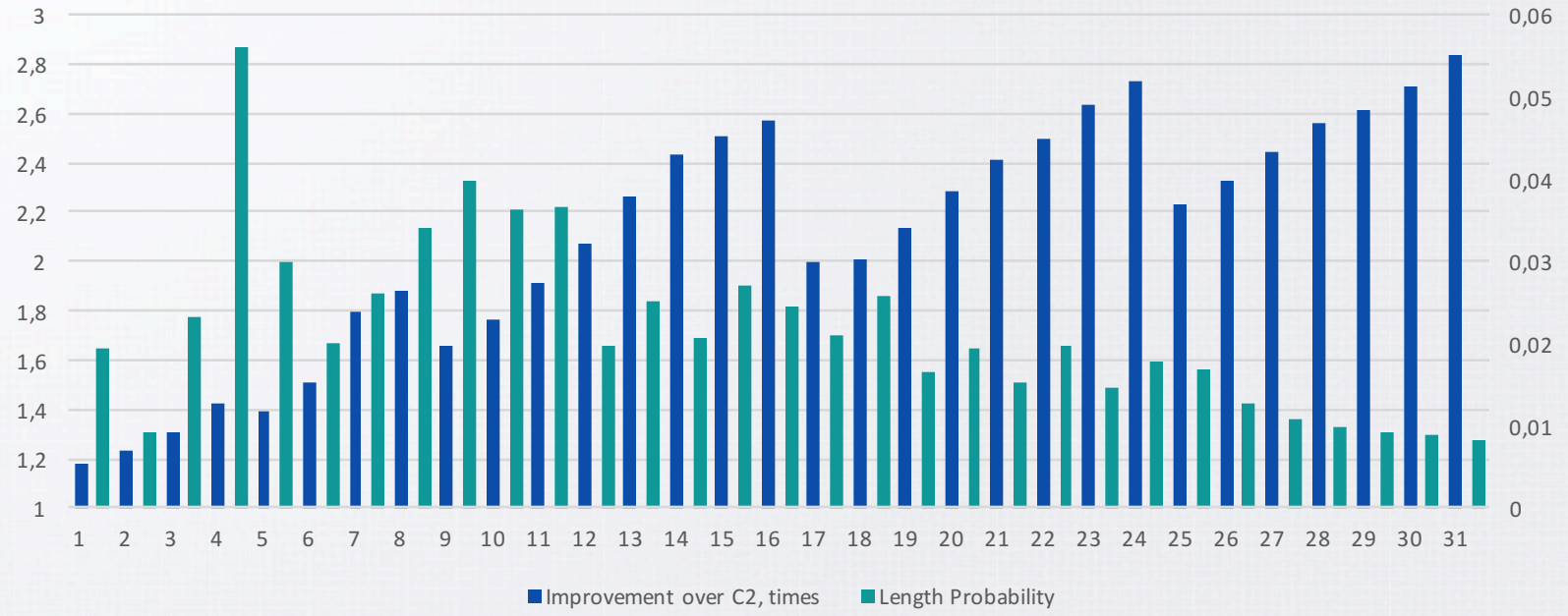
Хорошие новости:
- ускорение в 5-6 раз
- вроде не видно регрессий

Действительно ли нет регрессий?

Бенчмарк для new String() – результат



Бенчмарк для new String() – результат



Let's have a JEP, darling!

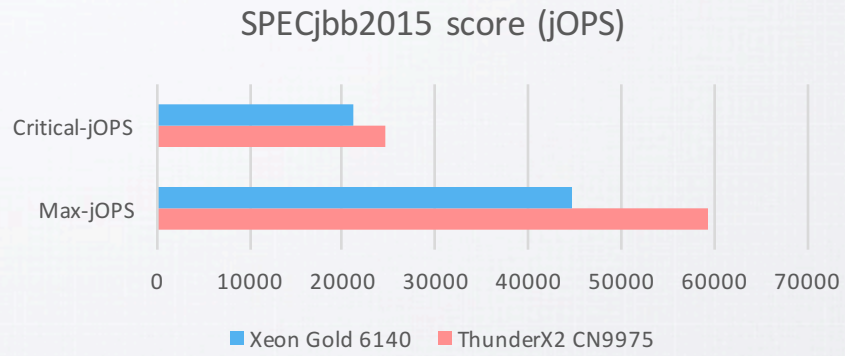
- JEP 315: Improve Aarch64 Intrinsics – Integrated in JDK 11
  - java.lang.String.new String()
  - java.lang.String.compareTo()
  - java.lang.StringUTF16.compress()
  - java.lang.StringLatin1.inflate()
  - java.lang.String.indexOf()
  - java.util.zip.CRC32.update()
  - java.utils.Arrays.equals()
  - java.lang.Math.log()
  - java.lang.Math.sin()
  - java.lang.Math.cos()

Улучшение производительности

- В микробенчмарках улучшение производительности - до 78x

* Среднеквадратичное улучшение производительности по разным размерам, длинам, кодировкам

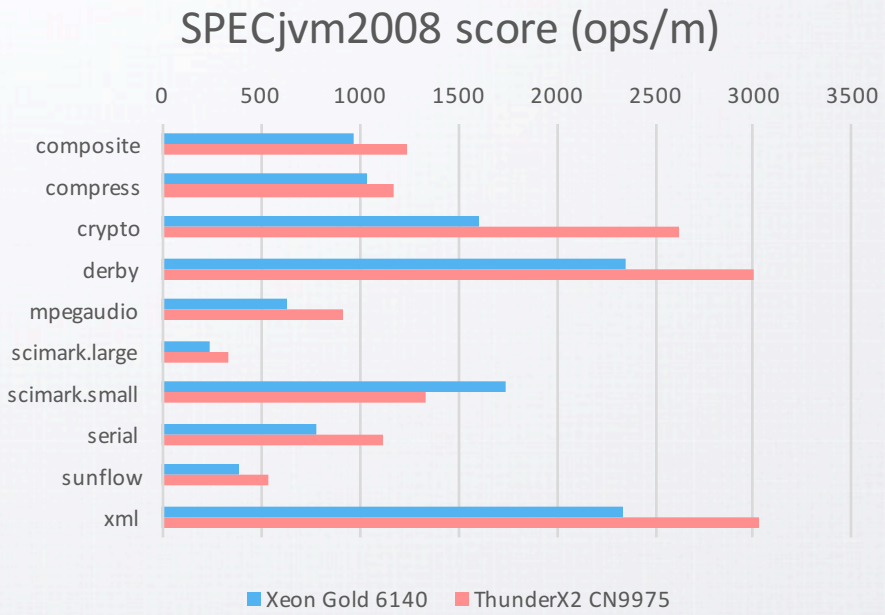
JVM Benchmark #1 results



- OpenJDK 11
- Geomean over 20 runs
- JEP 315 in JDK 11
- Cavium Thunder X2 outperforms Xeon 6140
  - by 33% in Max-jOPS score
  - by 16% in Critical-jOPS score

ARMv8: -Xmx24G -Xms24G -Xmn16G -XX:+AlwaysPreTouch -XX:+UseParallelGC -XX:+UseTransparentHugePages -XX:-UseBiasedLocking
X86: -Xmx24G -Xms24G -Xmn16G -XX:+AlwaysPreTouch -XX:+UseParallelGC -XX:+UseTransparentHugePages -XX:+UseBiasedLocking

JVM Benchmark #2 results



- OpenJDK 11
- Default JVM settings
- Geomean over 20 runs
- Thunder X2 outperforms Xeon 6140
  - by 62% in Crypto
  - by 42% in MpegAudio
  - By 29% in XML
  - by 12% in Compress
- Xeon 6140 outperforms Thunder X2
  - By 29% in scimark.small

Как потрогать ARM?

Bare Metal

VPS

DEMO

Выводы

- Производители ARM серверов сделали конфетку
- Cloud провайдеры предоставляют доступ к ARM серверам прямо сейчас
- Ubuntu, Red Hat, Oracle – все имеют поддерживаемые порты ОС на ARMv8
- Весь software ecosystem работает
- OpenJDK 11 не просто работает – он оптимизирован для архитектуры ARMv8

Скачай и используй Liberica JDK для ARMv8

OpenJDK. JCStress on architectures without TSO

AArgh... Memory models! Stress. jcstress

org.openjdk.jcstress.tests.fences.UnfencedPublicationTest

State	*Intel*			*Aarch64*		Interesting
	Skylake	Ivy Bridge	Westmere	Thunder X	Thunder X2	
0, 0	4 - 70 M	8 - 60 M	8 - 60 M	1 – 39 M	1 – 55 M	
1, 0	0	0	0	1	0 – 4 K	✔
1, 1	0.3 – 100 M	0.5 – 128 M	0.5 – 128 M	0.3-0.9 M	0.4 – 3 M	

JCStress test explained

UnfencedPublicationTest

```
@Outcome(id = "0, 0", ... "Data not yet published")
@Outcome(id = "0, 1", ... "Data not yet published")
@Outcome(id = "1, 0", ... "Reads the default value for field $x after publication.")
@Outcome(id = "1, 1", ... "Must read the written value for $x after publication.")
```

```
Data data;

static class Data {
   int x; // default 0
}
```

```
public void actor1() {
  Data d = new Data();
  d.x = 1;
  data = d;
}
```

```
public void actor2(II_Result  r) {
  int sy, sx;
  Data d = data;
  if (d == null) {
    sy = 0;
    sx = 0;
  } else {
    sy = 1;
    sx  = d.x;
  }
  r.r1 = sy; // published flag
  r.r2 = sx; // data $x
}
```