Выражаемся регулярно

Владимир Ситников

JPoint 2015

NetCracker®

© 2015 NetCracker Technology Corporation Confidential

О себе

- Владимир Ситников, @VladimirSitnikv
- Инженер по производительности в NetCracker
- 8 лет опыта с Java/SQL
- Мои предложения принимались в j.l.String, j.u.c.CopyOnWriteArrayList

© 2015 NetCracker Technology Corporation Confidential

План

- Небольшое введение
- Разгоняем `/.*/` и `/.*?/`
- Устраняем StackOverflowError
- Regexp vs XPath
- ANTLR, VTD-XML – не на этом докладе

© 2015 NetCracker Technology Corporation Confidential

Regexp

he **comes, 'hi** s u … *nent,*
HTML tags ***leakin*** … ong of
regular exp ressi … ***r tal man***
from the sp *here* … e final
snuffing *of the li* … *ny he*
comes he comes … *CE MY*
FACE °h god ***no M*** … **real**
ZALGO IS TON

© 2015 NetCracker Technology Corporation Confidential

Случаи использования regexp

- Анализ success/fail в логах, xml ответа
  - Поиск “<status>success</status>”
- Засекречивание паролей при логировании
  - Замена “password=(.*)” на “password=****”
- По неосторожности

© 2015 NetCracker Technology Corporation Confidential

Случаи использования regexp

- Анализ success/fail в логах, xml ответа
  - Поиск “<status>success</status>”
- Засекречивание паролей при логировании
  - Замена “password=(.*)” на “password=****”
- По неосторожности (String.replaceAll и т.п.)

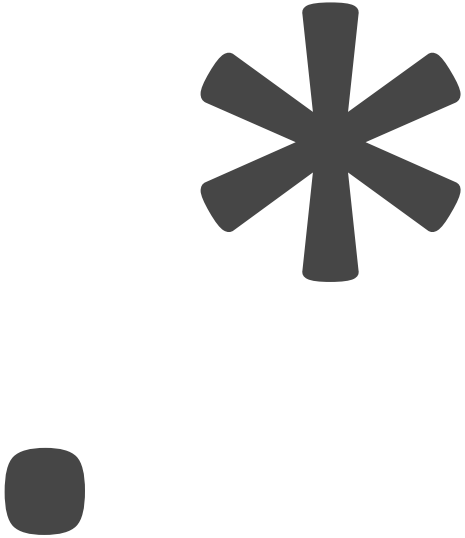
© 2015 NetCracker Technology Corporation Confidential

Две проблемы

- У вас проблема, и вы собрались использовать регулярные выражения для её решения
- Теперь у вас <u>две</u> проблемы

© 2015 NetCracker Technology Corporation Confidential

Ужасы нашего городка



© 2015 NetCracker Technology Corporation Confidential

Что не так с .*?

- Обычно, regexp библиотеки работают «методом перебора»
  - Если хвост «не подошёл», откатим попробуем Regexp ещё раз

© 2015 NetCracker Technology Corporation Confidential

/^.*ab$/.match("ab")

.* – жадная конструкция. Ест как не в себя

|ab
.*ab

a|b
.*ab

ab|
.*ab

ab|
.*ab

a|b
.*ab
откатились

a|b
.*ab
не помогло

|ab
.*ab
откатились

ab|
.*ab
Ура!

Ленивые вычисления спешат на помощь

- Ленивый вариант помогает в конкретном случае

```
/^.*?abc$/.match("abc")
```

- Но помогает далеко не всегда

© 2015 NetCracker Technology Corporation Confidential

/^.*?b$/.match("aaa")

.*? – ленивая конструкция

|aab
.*?ab

a|ab
.*?ab

a|ab
.*?ab

|aab
.*?ab
откатились

a|ab
.*?ab

aa|b
.*?ab

aab|
.*?ab
Ура!

© 2015 NetCracker Technology Corporation Confidential

Ленивые вычисления и реальность

- Если regexp не подходит к строке, то всё равно придётся рассмотреть все случаи

- Был случай, что выражение вида `<.*:item.*>(.*)</.*:item>` работало **30 секунд** на 100 КиБ строке

© 2015 NetCracker Technology Corporation Confidential

Ленивые вычисления и реальность

```
<.*?:item.*?>(.*?)</.*?:item>
```

Уже лучше, regexp engine всё равно будет откатываться при выполнении .*

© 2015 NetCracker Technology Corporation Confidential

Ужасы нашего городка

.*?

© 2015 NetCracker Technology Corporation Confidential

Ленивые вычисления и реальность

```
<[^:>]*:item[^>]*>([^<]*)</...
```

[^>] не выйдет за границы XML тэга, вариантов будет намного меньше, и работать будет гораздо быстрее

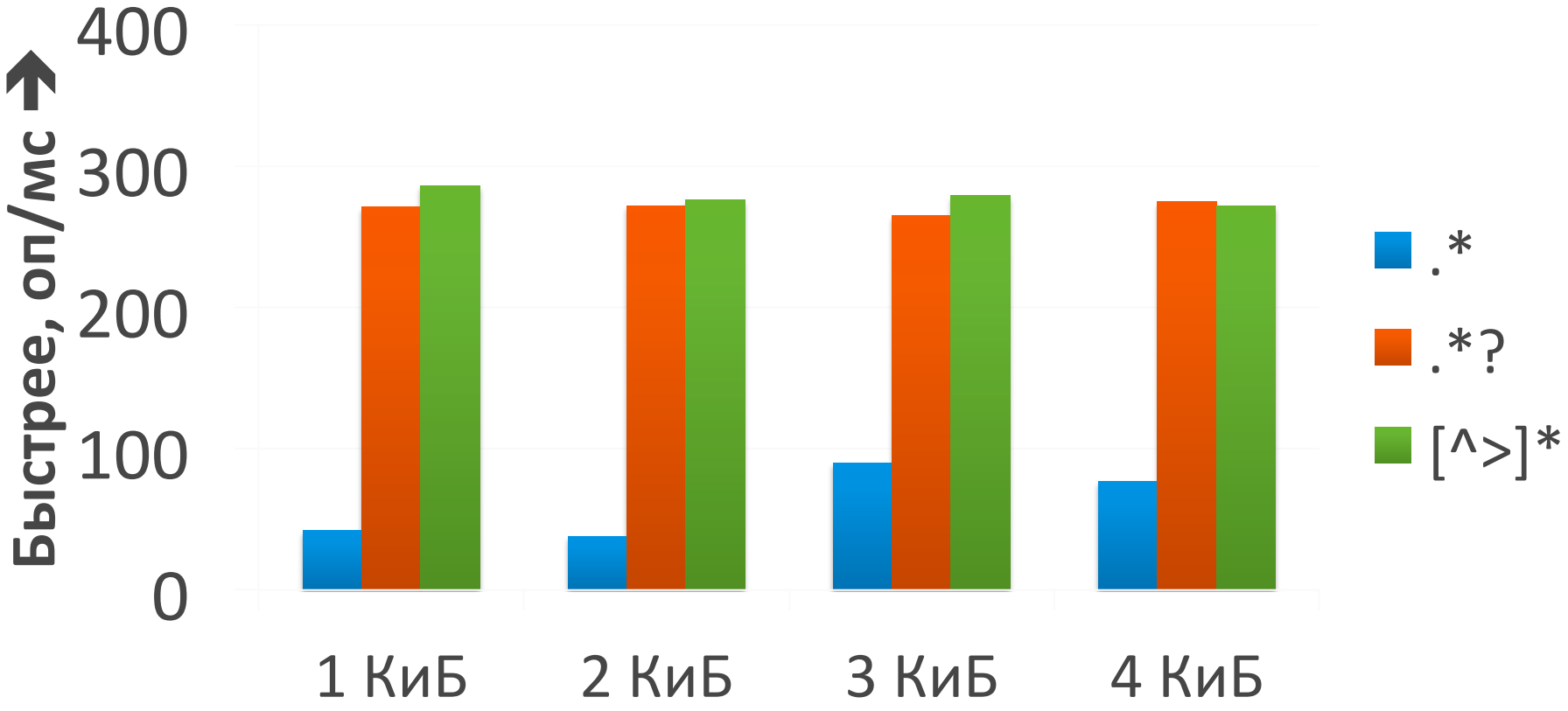
© 2015 NetCracker Technology Corporation Confidential

Сравним 3 варианта

- <ROW>.*<ACCNT>.*</ACCNT>.*</ROW>
- <ROW>.*?<ACCNT>.*?</ACCNT>.*?</ROW>
- <ROW>.*?<ACCNT>[^<]*</ACCNT>.*?</ROW>

© 2015 NetCracker Technology Corporation Confidential

График скорости .*, .*?, [^>]



© 2015 NetCracker Technology Corporation Confidential

График скорости .*, .*?, [^>]

© 2015 NetCracker Technology Corporation Confidential

JPoint или Joker?

```
Pattern.compile("(Joker|JPoint)+")
```

- На строке "JokerJPoint…700раз…Joker" падает с ошибкой StackOverflowError
- Как так?

© 2015 NetCracker Technology Corporation Confidential

JPoint или Joker?

```
java.lang.StackOverflowError
.regex.Pattern$BranchConn.match(Pattern.java:4568
.regex.Pattern$Slice.match(Pattern.java:3972)
.regex.Pattern$Branch.match(Pattern.java:4604)
.regex.Pattern$GroupHead.match(Pattern.java:4658)
.regex.Pattern$Loop.match(Pattern.java:4785)
.regex.Pattern$GroupTail.match(Pattern.java:4717)
.regex.Pattern$BranchConn.match(Pattern.java:4568
```


© 2015 NetCracker Technology Corporation Confidential

JPoint или Joker?

```
Pattern.compile("(Joker|JPoint)+?")
```

- Добавим "?". Помогло?
- Падать перестало, но теперь посещаем только **первую** конференцию из всех

© 2015 NetCracker Technology Corporation Confidential

Однажды в NetBeans

NetBeans Bug 154894 - StackOverflowError at java.util.regex.Pattern$SliceI.match

support/JavadocAndSourceRootDetection.java

```
String whitespace = "…(?:[^*]|\\*[^/])*…"
```


© 2015 NetCracker Technology Corporation Confidential

JPoint или Joker?

```
Pattern.compile("(Joker|JPoint)+")
```

- Regexp engine оставляет шанс отката для проверки альтернативной ветки
- В OpenJDK Pattern использует ~~ООП~~ java stack, при большой вложенности и получается StackOverflow

© 2015 NetCracker Technology Corporation Confidential

Можно ли избавиться от откатов?

Без откатов крайне сложно (невозможно?) сделать

© 2015 NetCracker Technology Corporation Confidential

Можно ли избавиться от откатов?

Без откатов крайне сложно (невозможно?) сделать back-references:

(.*)\1

aa, abab, abcabc, …

© 2015 NetCracker Technology Corporation Confidential

Как это выглядит в документации

https://docs.oracle.com/javase/tutorial/essential/regex/quant.html

Greedy	Reluctant	Posessive	Значение
X?	X??	X?+	X, один раз или ни единого
X*	X*?	X*+	X, ноль или более раз
X+	X+?	X++	X, один или более раз
X{n}	X{n}?	X{n}+	X, ровно n раз
X{n,}	X{n,}?	X{n,}+	X, как минимум n раз
X{n,m}	X{n,m}?	X{n,m}+	X, от n до m раз

© 2015 NetCracker Technology Corporation Confidential

Чего-чего?

X??

© 2015 NetCracker Technology Corporation Confidential

А на это мало кто обращает внимание

Greedy	Reluctant	Posessive	Значение
X?	X??	X?+	X, один раз или ни единого
X*	X*?	X*+	X, ноль или более раз
X+	X+?	X++	X, один или более раз

© 2015 NetCracker Technology Corporation Confidential

C++ спешит на помощь

```
Pattern.compile("(Joker|JPoint)++")
```

- ++ и *+ отключает откаты, и это как раз то, что нужно

© 2015 NetCracker Technology Corporation Confidential

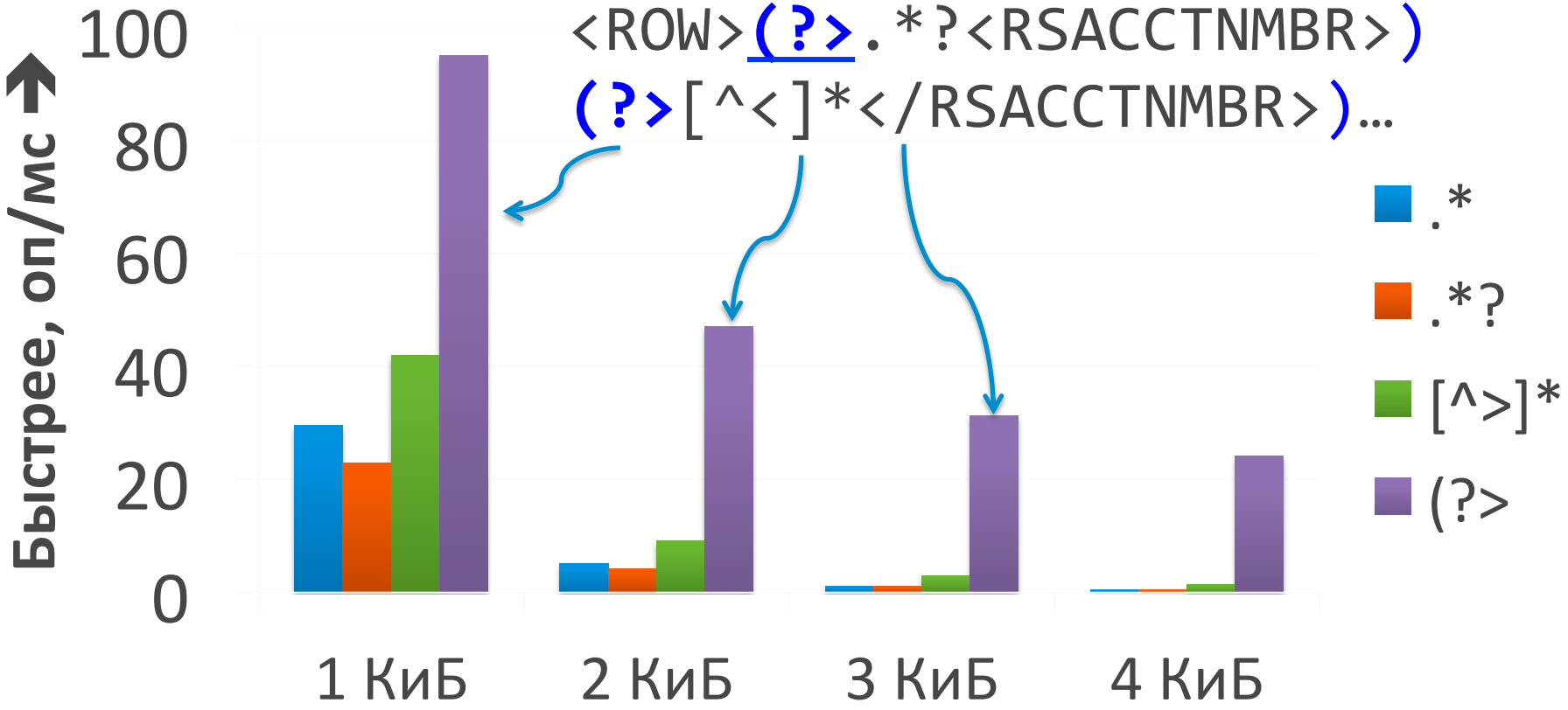
«Мне повезёт»

```
Pattern.compile("(?>Joker|JPoint)+")
```

?> – режим, при котором все откаты внутри скобок «забываются». Первый совпавший вариант принимается как окончательный

© 2015 NetCracker Technology Corporation Confidential

График скорости .*, .*, [^>], шаблон не найден



© 2015 NetCracker Technology Corporation Confidential

/(?>ab|a)b/.match("a")

(?>ab|a) не будет проверять второй вариант

|ab
(?>ab|a)b

a|b
(?>ab|a)b

ab|
(?>ab|a)b

ab|
(?>ab|a)b

ab|
(?>ab|a)b

Шаблон не найден

А давайте ускорим String?

- String.format("...") использует java.util.regex.Pattern для разбора формата

```
formatSpecifier = "%(\\d+\\$)?([-#+ 0,(\\<]*)?(\\d+)?(\\.\\d+)?([tT])?([a-zA-Z%])";
```


© 2015 NetCracker Technology Corporation Confidential

Откуда взять тестовые данные?

Возьмём шаблон из самого JMH:

```
(min, avg, max) = (%s, %s, %s), stdev = %s%n
  CI (99.9%%): [%s, %s] (assumes normal
distribution)
```


© 2015 NetCracker Technology Corporation Confidential

А давайте ускорим String?

```
formatSpecifier = "%(\\d+\\$)?([-#+ 0,(\\<]*)?(\\d+)?(\\.\\d+)?([tT])?([a-zA-Z%])";

fastSpecifier = "%(\\d++\\$)?+([-#+ 0,(\\<]+)?+(\\d++)?+(\\.\\d++)?+([tT])?+([a-zA-Z%])";
```


© 2015 NetCracker Technology Corporation Confidential

Бесплатный сыр

Benchmark	Cnt	Score		Error	Units
jdk18u45	60	1,067	±	0,025	us/op
optimized	60	**0,917**	±	0,021	us/op

NetCracker® © 2015 NetCracker Technology Corporation Confidential

NetBeans, NetBeans

```
(?>[^*]|\*[^/])
```

Работает?

© 2015 NetCracker Technology Corporation Confidential

NetBeans, NetBeans

```
(?>[^*]|\*[^/])
```

StackOverflow не бросит, а на /***/ не сработает

© 2015 NetCracker Technology Corporation Confidential

NetBeans, NetBeans

```
(?>[^*]|\*(?!/))
```

Вот это уже сработает!

(?!/) – negative lookahead

© 2015 NetCracker Technology Corporation Confidential

Признаки плохого регулярного выражения

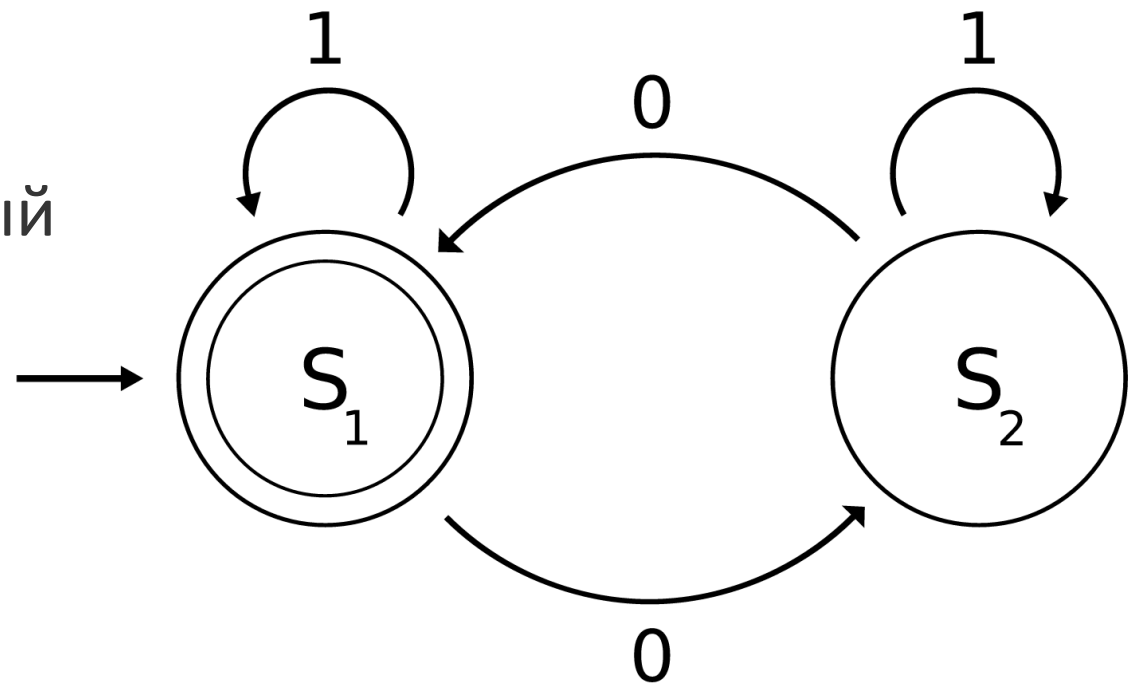
- Неспецифичные символы
  - .*
- Внутри группировок
  - Повторения: (a+)+, ([a-zA-Z]+)*=
  - Альтернативы с пересечениями: (a|aa)+, (a|a?)+

© 2015 NetCracker Technology Corporation Confidential

А можно ли совсем без откатов?

Если отказаться от back-references, то есть быстрые алгоритмы

Детерминированный конечный автомат для /(1|01*0)*/



© 2015 NetCracker Technology Corporation Confidential

Современные regexp библиотеки в java

Название	Год выпуска	Алгоритм
java.util.regex.Pattern	2015	backtrack
com.basistech.tclre	2015	DFA
org.jruby.joni (Nashorn)	2014	backtrack
Apache Oro	2000	backtrack
JRegex	2013	backtrack

© 2015 NetCracker Technology Corporation Confidential

Современные regexp библиотеки в смежных языках

Название	Версия	Алгоритм
Oracle	12c	backtrack
PostgreSQL	9.3	DFA
MySQL	5.6	DFA
JavaScript/Chrome	43	backtrack
JavaScript/FireFox	48	backtrack
Python/Ruby/Perl/PHP		backtrack

© 2015 NetCracker Technology Corporation Confidential

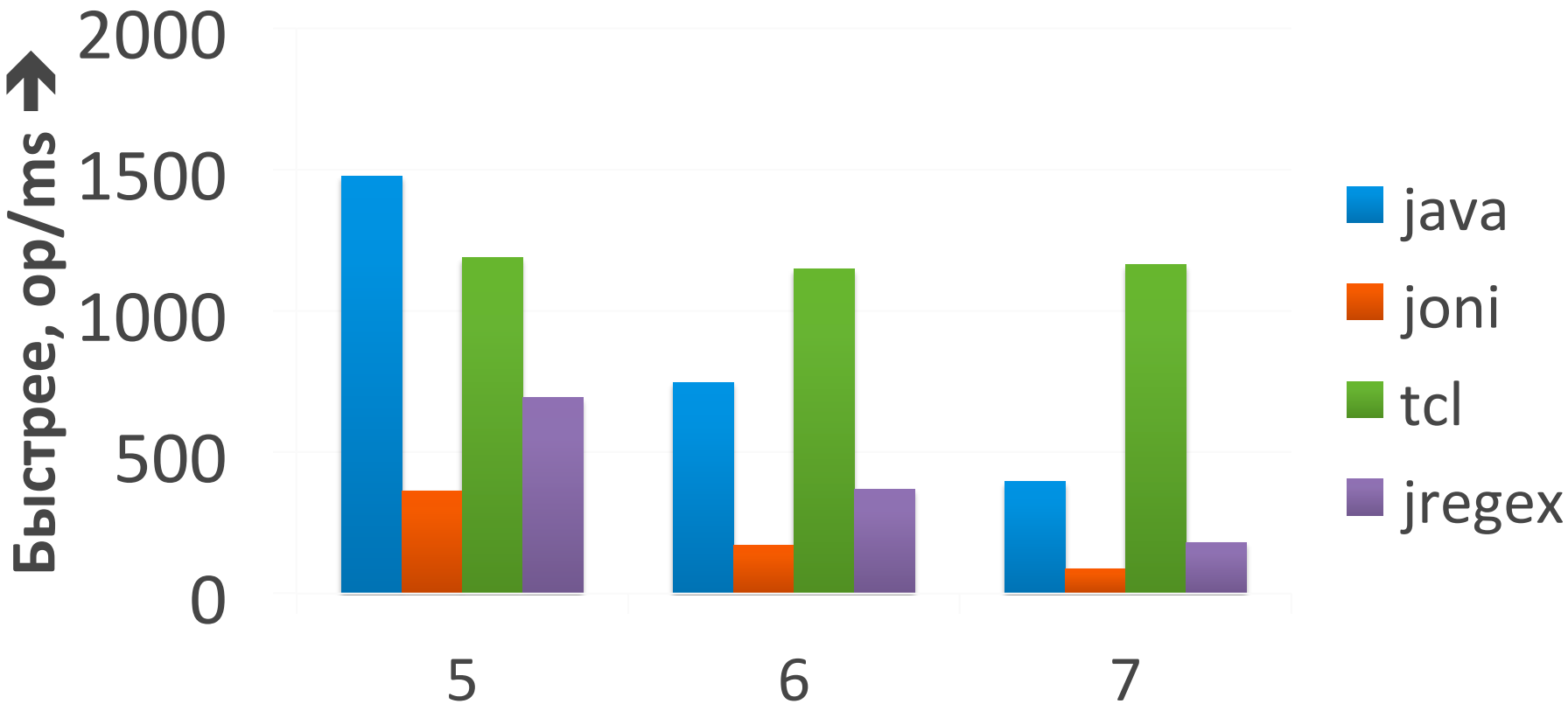
Как проверить backtrack?

```
/(x+x+)+y/.test("xxxxx…20 раз…xxx")
```

Если regexp с откатами, то он будет очень долго подбирать x+

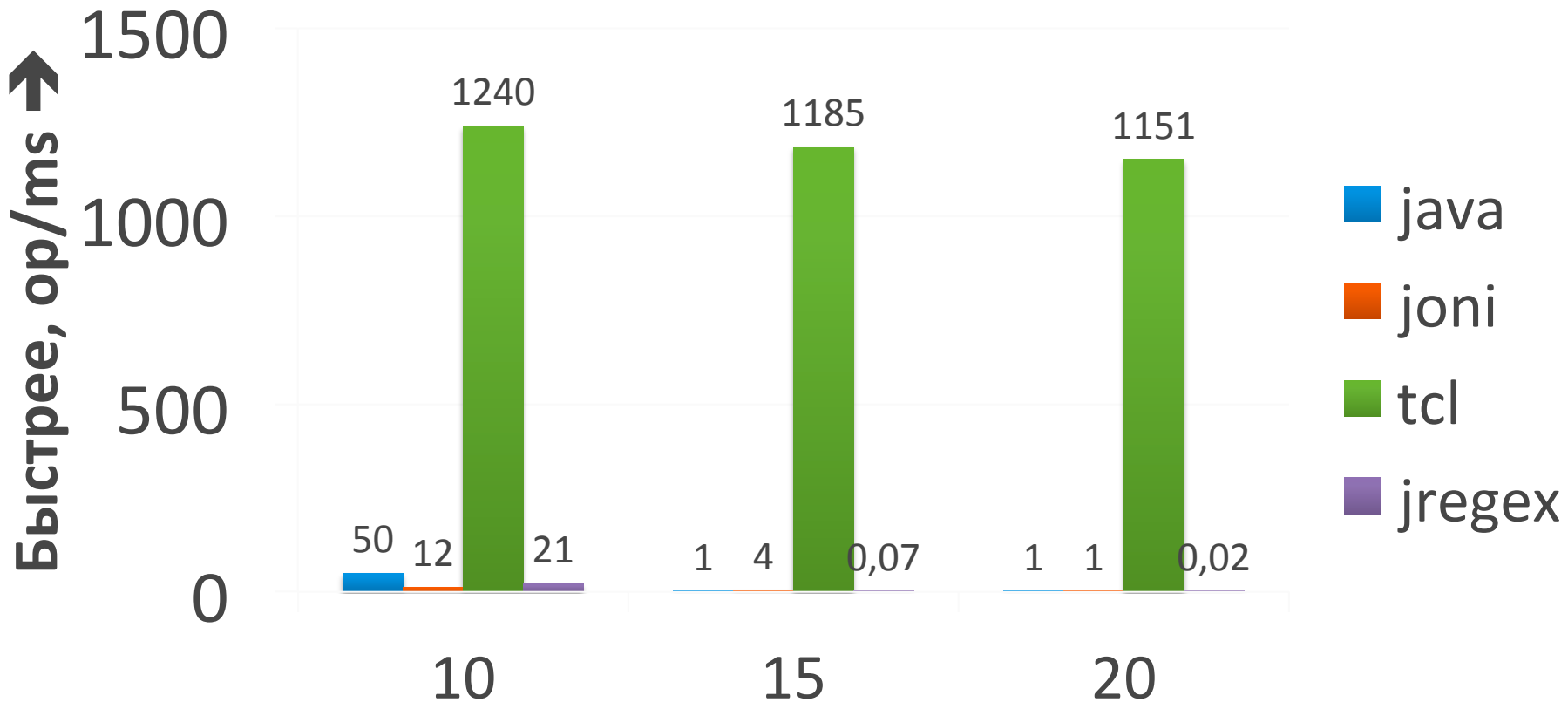
© 2015 NetCracker Technology Corporation Confidential

/(x+x+)+y/.test("xxxx…xxx")



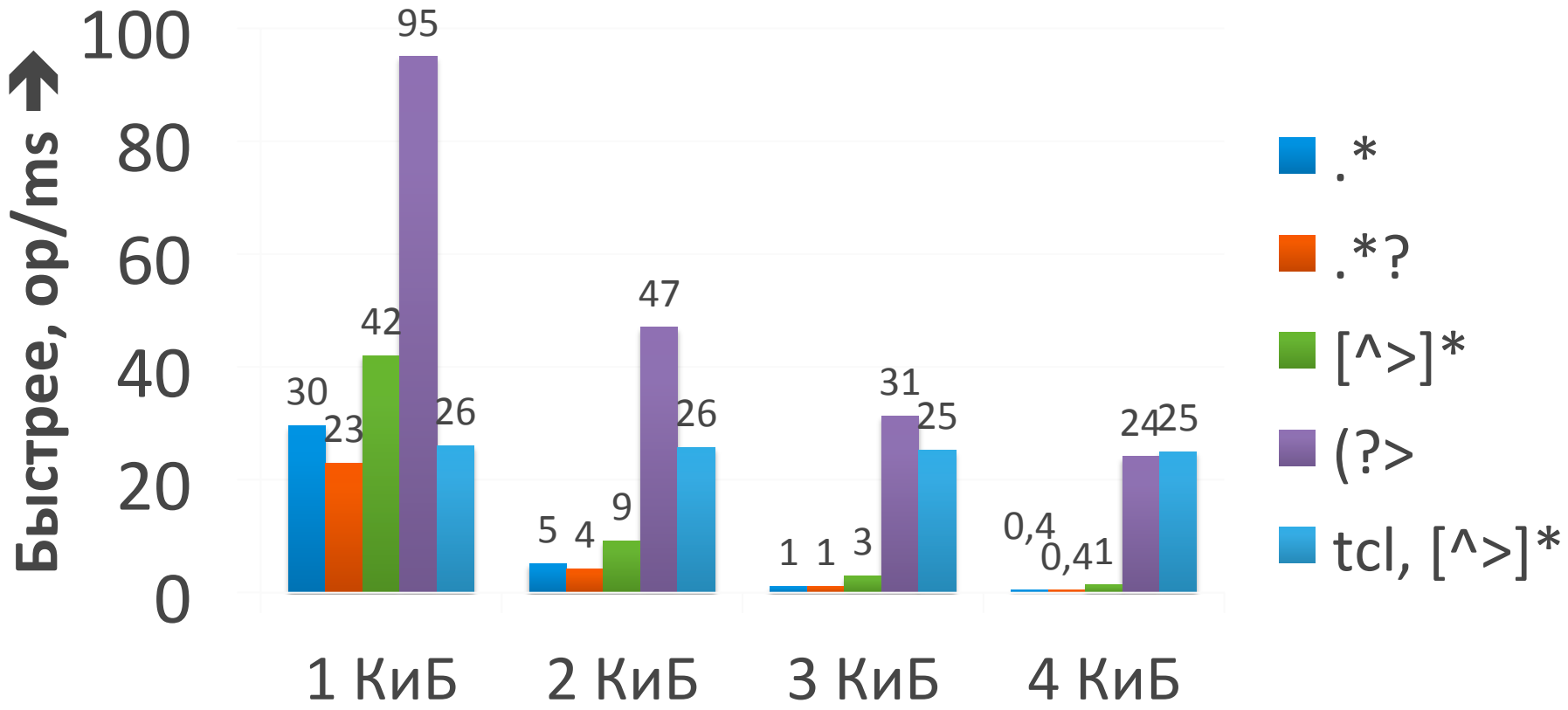
© 2015 NetCracker Technology Corporation Confidential

/(x+x+)+y/.test("xxxx…xxx")



© 2015 NetCracker Technology Corporation Confidential

<ROW>(?>.*?<RSACCTNMBR>)(?>[^<]*</RSACCTNMBR>)



© 2015 NetCracker Technology Corporation Confidential

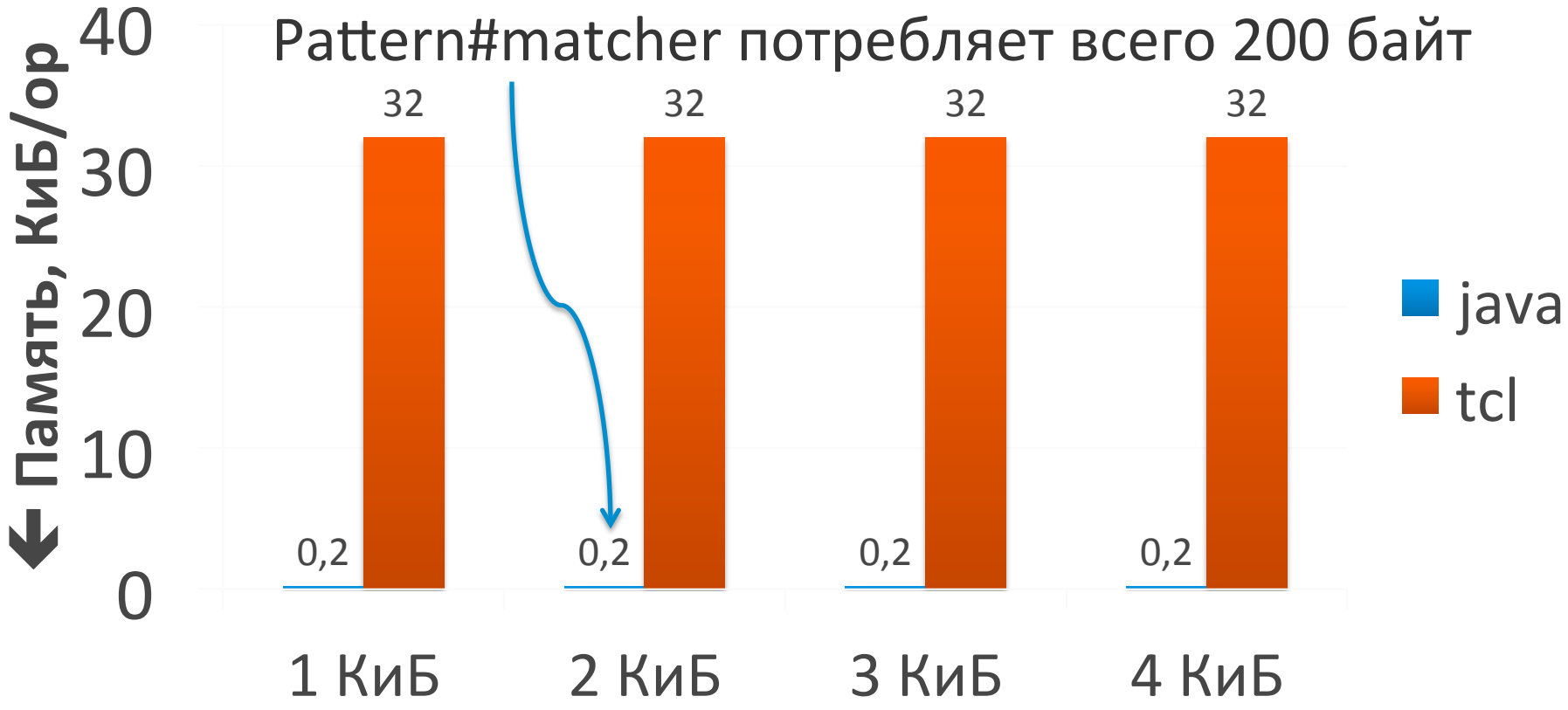
 © 2015 NetCracker Technology Corporation Confidential

CPU vs memory

- DFA подход потребляет больше памяти
- Патчим jmh, и там появляется allocation profiler

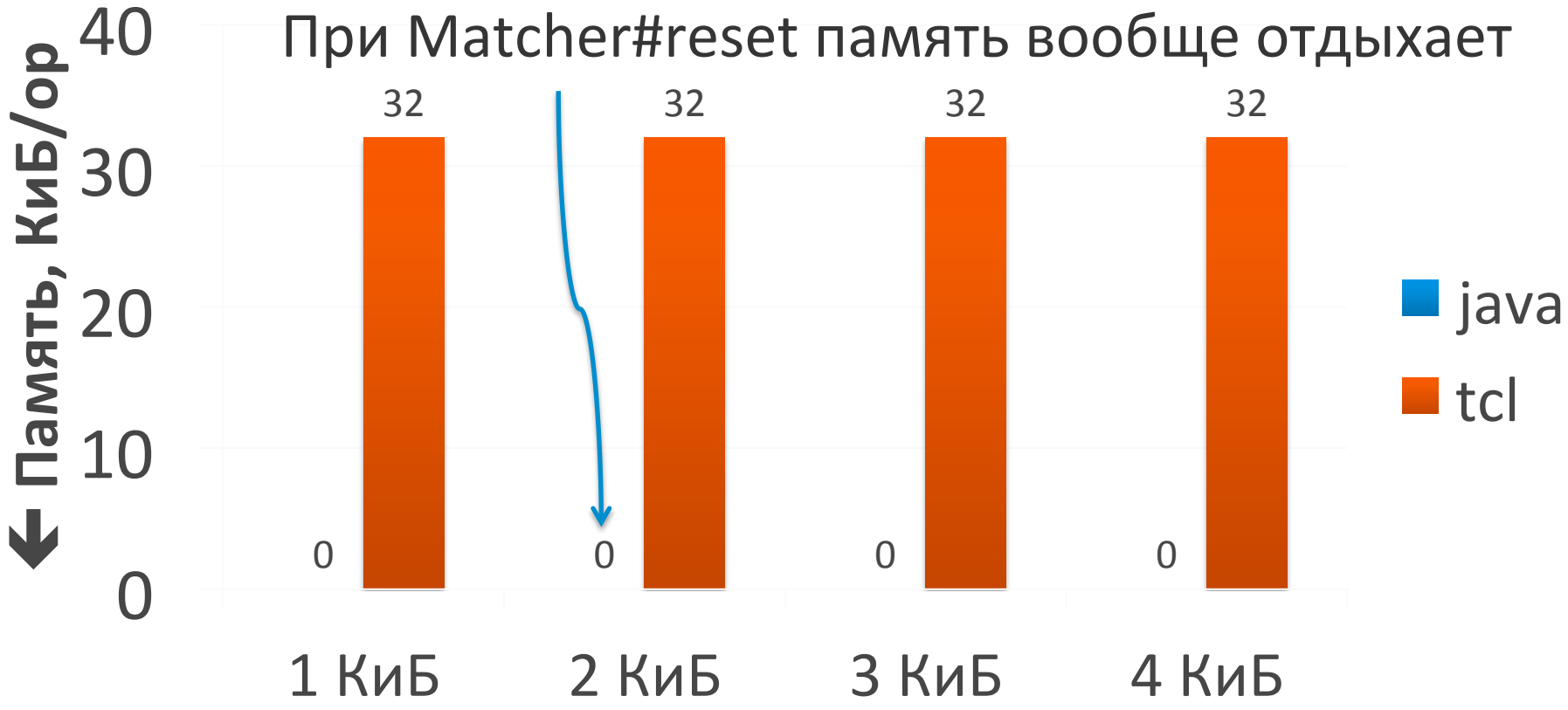
© 2015 NetCracker Technology Corporation Confidential

CPU vs memory: <row>…



© 2015 NetCracker Technology Corporation Confidential

CPU vs memory: <row>…



© 2015 NetCracker Technology Corporation Confidential

j.l.r.Pattern

- Поддерживает Unicode (java 1.7+)
- Управляемый откат
- Малое потребление памяти

© 2015 NetCracker Technology Corporation Confidential

org.jruby.Joni

- Поддерживает Unicode
- Управляемый откат
- Может работать с byte[] в UTF-8 кодировке
- Поддерживает Thread#interrupt()
- Малое потребление памяти

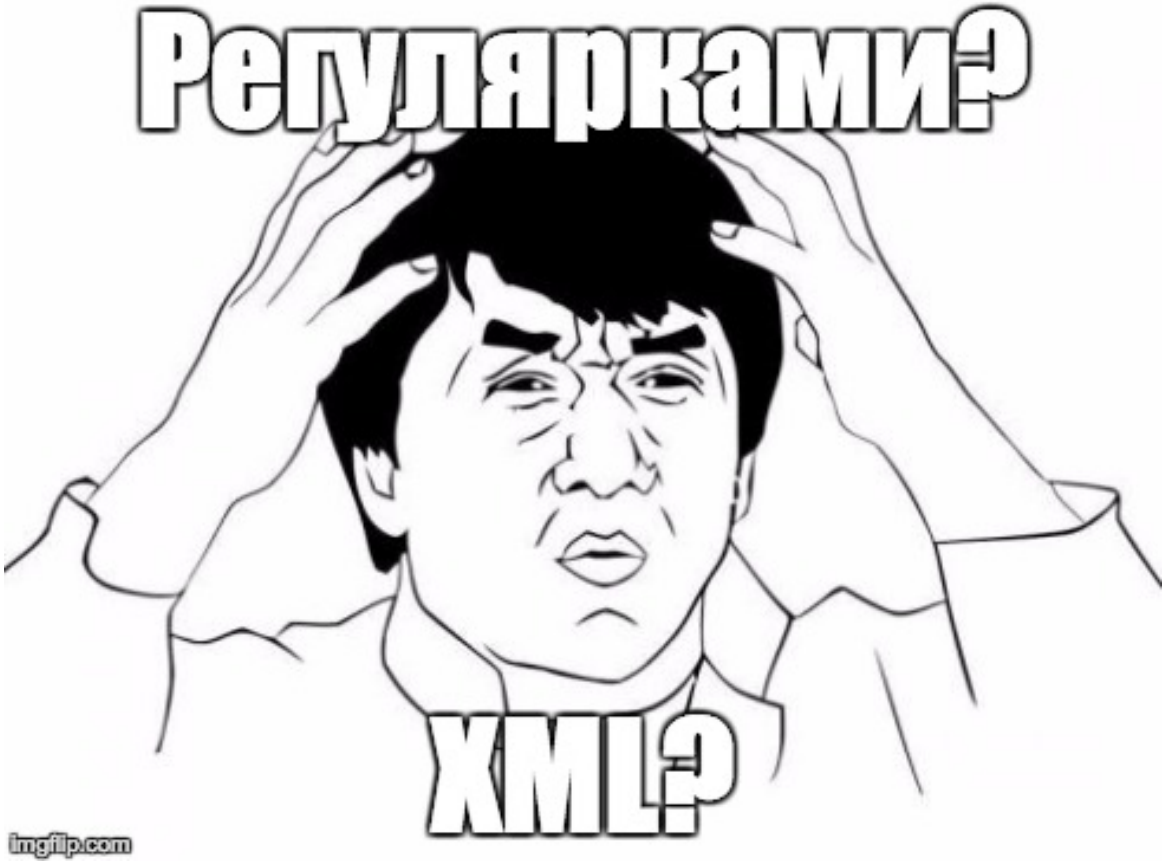
© 2015 NetCracker Technology Corporation Confidential

Tcl-re

- DFA алгоритм, стабильное время работы
- Не самая быстрая на простых шаблонах
- Большое потребление памяти

© 2015 NetCracker Technology Corporation Confidential

Они всё ещё разбирают XML regexp’ами



© 2015 NetCracker Technology Corporation Confidential

Сравним с XPath

- Парсинг XML (20, 40, 60КиБ)
- Поиск тэга по точному пути
- Замена пароля звёздочками (поиск по имени атрибута)

© 2015 NetCracker Technology Corporation Confidential

XPath

- Парсинг XML
- Поиск тэга по точному пути
  - **ICOMS/MAC00029/@ACNT**
- Замена пароля звёздочками (поиск по имени атрибута)
  - **//self::node()[@__hPINNMBR]**

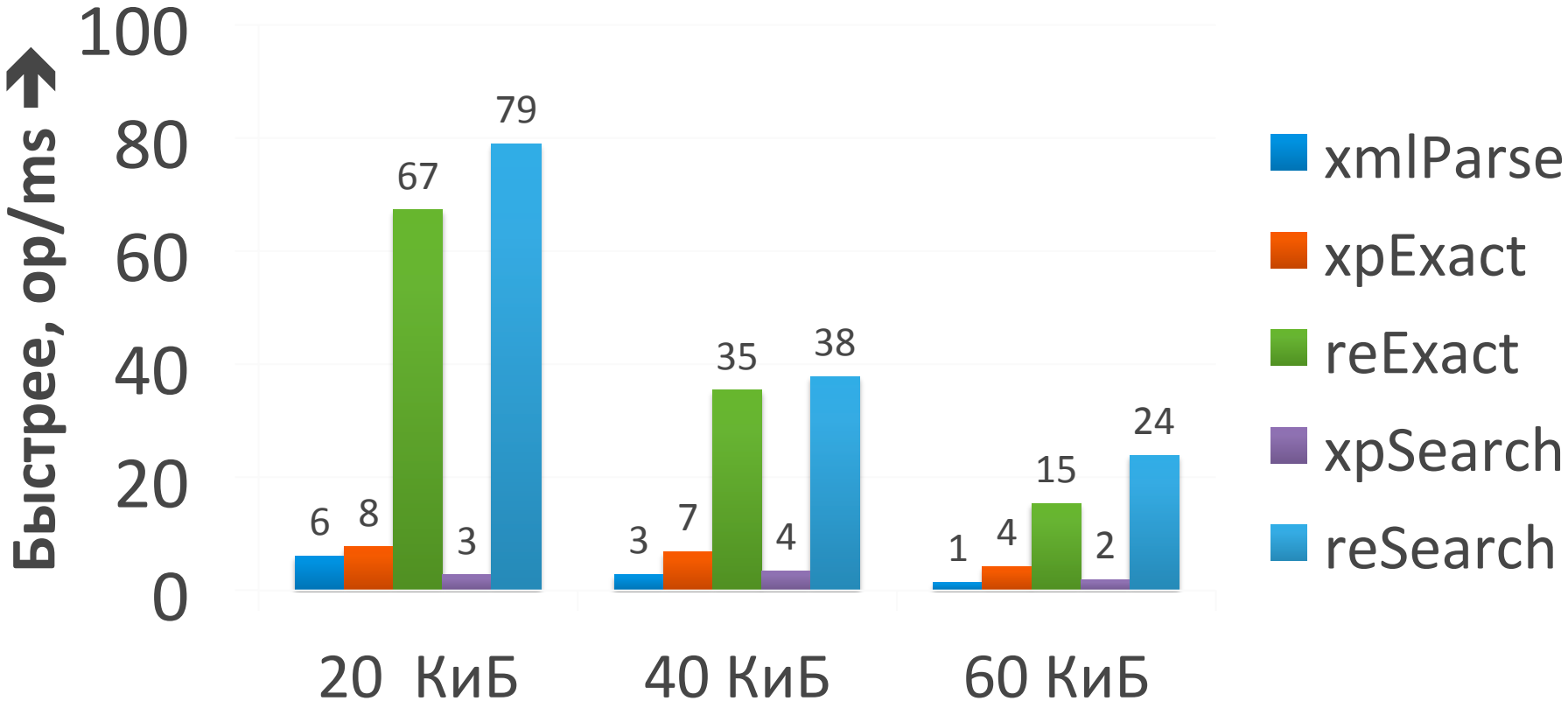
© 2015 NetCracker Technology Corporation Confidential

Regexp

- Парсинг XML
- Поиск тэга по точному пути
  - **<MAC00029(?:.(?! ACNT))*+ ACNTN="([^"]*+)"**
- Замена пароля звёздочками (поиск по имени атрибута)
  - **__hPINNMBR="[^"]*+"**

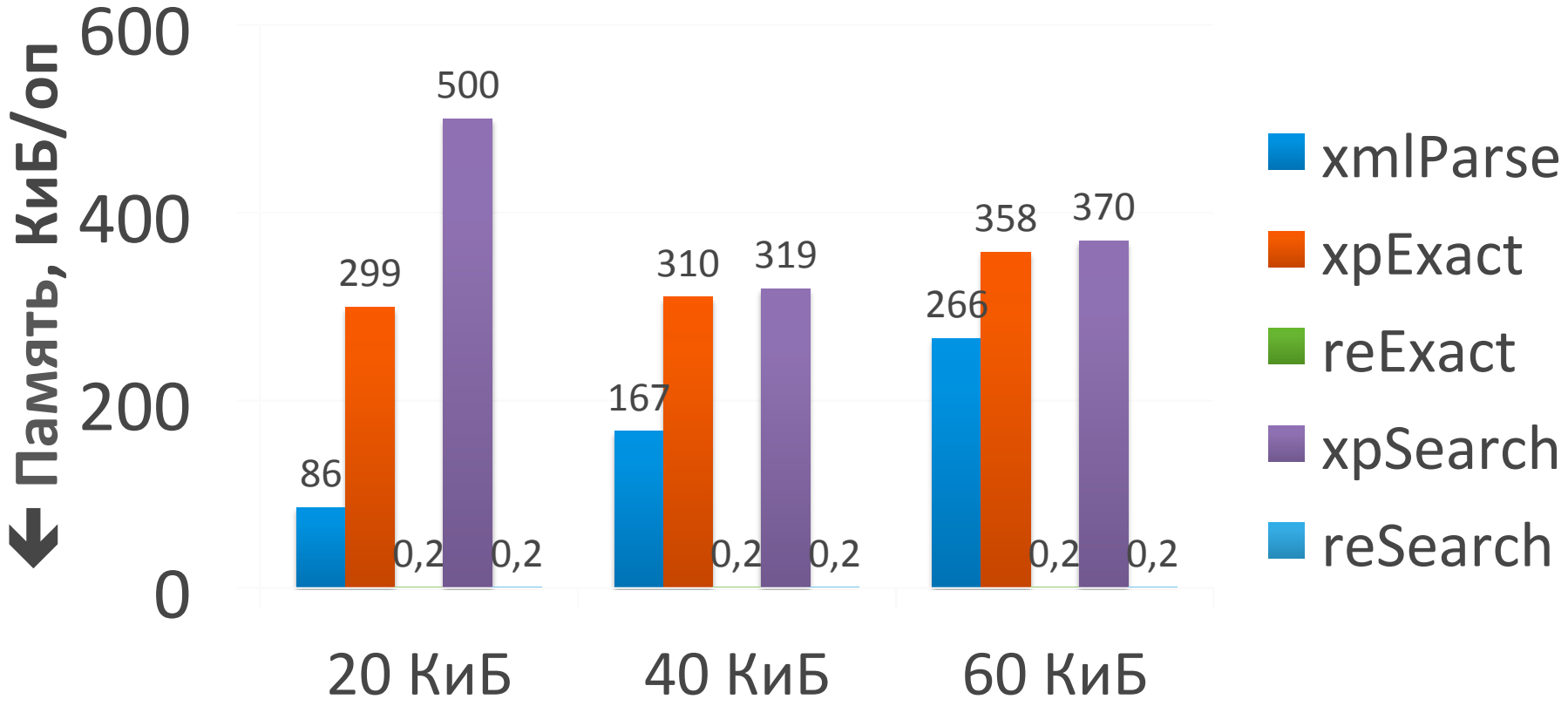
© 2015 NetCracker Technology Corporation Confidential

CPU: XML vs Pattern



© 2015 NetCracker Technology Corporation Confidential

Heap: XML vs Pattern



© 2015 NetCracker Technology Corporation Confidential

Полмегабайта на xpath

- Где моя память, чувак?!

```
.jvmArgs("-XX:+UnlockCommercialFeatures",
 "-XX:+FlightRecorder",
 "-XX:StartFlightRecording=duration=60s,"
 + "settings=profile,"
 + "filename=xpathExact_60k.jfr")
```


© 2015 NetCracker Technology Corporation Confidential

JFR показывает кто виноват

Java FlightRecorder показывает, что виноват XPathContext

Allocation Pressure

Class	Average Objec	Total TLAB Size	Pressure
com.sun.org.apache.xpath.internal.objects.XObject[]	32,02 kB	3,12 GB	42,61%
int[]	8,12 kB	2,50 GB	34,09%
java.lang.Object[]	12,22 kB	1,32 GB	17,98%
com.sun.org.apache.xml.internal.dtm.ref.ExtendedType	32 bytes	160,31 MB	2,14%

Stack Trace

Stack Trace	Total TLAB Size	Pressure	TLABs
com.sun.org.apache.xpath.internal.VariableStack.reset()	1,73 GB	55,26%	872
com.sun.org.apache.xpath.internal.VariableStack.<init>()	1,73 GB	55,26%	872
com.sun.org.apache.xpath.internal.jaxp.JAXPVariable…	1,13 GB	36,26%	567
com.sun.org.apache.xpath.internal.XPathContext.<ini…	607,87 MB	19,00%	305
com.sun.org.apache.xpath.internal.VariableStack.<init>()	1,40 GB	44,74%	657

© 2015 NetCracker Technology Corporation Confidential

One way ticket

JDK-6344064 Potential XPath performance issue in "new XPathContext()".

Fresh XPathContext is created for each call to evaluate(). However, the way the API is set up now, it is difficult not to do this. I'm assuming that the use case is **having a W3C DOM and evaluating multiple XPath expressions on it**.

 © 2015 NetCracker Technology Corporation Confidential

Заключение

- (?>…), ++, *+, ?+ отключают backtracking, что сильно ускоряет regexp
- В простых случаях java.util работает быстро
- Точный XPath работает быстрее Pattern, но аллоцирует уйму памяти

© 2015 NetCracker Technology Corporation Confidential

Вопросы?

NetCracker®

Владимир Ситников,

JPoint 2015

© 2015 NetCracker Technology Corporation Confidential