versnel je applicaties met java se 7 “dolphin”

jeroen borgers

xebia

jborgers at xebia.com

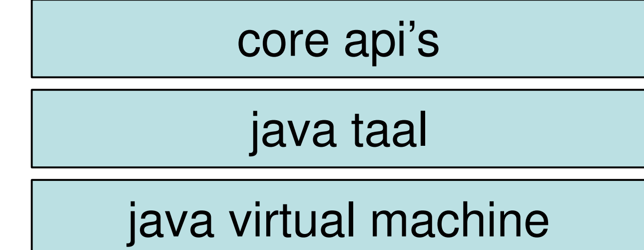
versnel je applicaties met java se 7
“dolphin”
jeroen borgers
xebia
jborgers at xebia.com

doel

- overzicht geven van performance vooruitgang in java se 7
  – wat kun je er nu mee?
  – wat kun je er straks mee?
  – hoe werkt het?

agenda

- **introductie**
- jvm
- java taal
- core api's
- conclusies



dolphin in de maak

dolphin == ruwe snelheid

wat heb je er nu en straks aan?

- jvm is sneller
  - benchmarks tonen 45% sneller dan jvm 6
- maakt nu je apps sneller
  - rich client & browser apps
  - app servers als glassfish, tomcat
- momenteel *early access*: eigen risico
  - build 75; m5: feature complete
- rc1 in februari
- final in april

snelheid kun je benutten

agenda

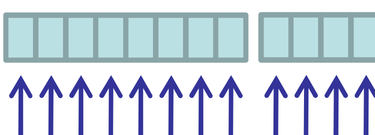
- introductie
- **jvm**
  - 64 bit compressie
  - invokedynamic
  - g1 garbage collector
  - escape analysis
  - arrays
- java taal
- core api’s
- conclusies

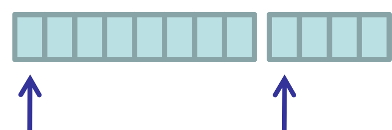
64-bit probleem: pointer overhead

- 64 bit jvm voordeel: grotere heaps
  - 2^32 = 4 GB
  - 2^64 = 16 777 216 TB = 16 EB
- overhead van 64 bit t.o.v. 32 bit jvm:
  - 50% meer geheugengebruik
  - 10-20% slechtere performance
- elke native pointer neemt 8 i.p.v. 4 bytes
- slechtere benutting cpu caches en bus
- hoe oplossen?

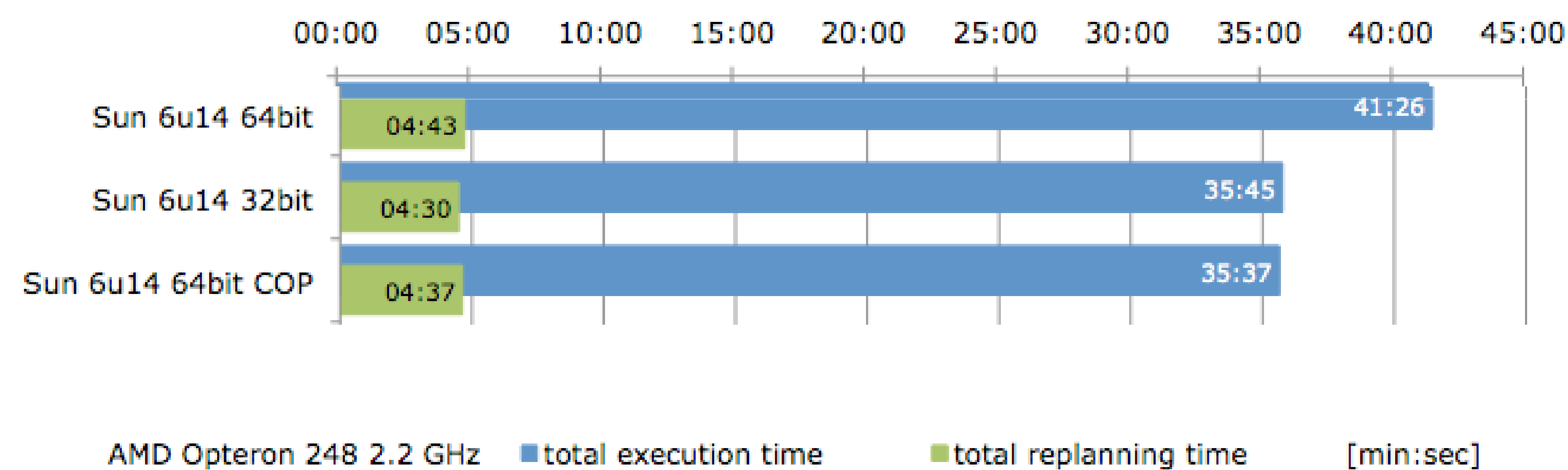
oplossing: pointer compressie

- **`-XX:+UseCompressedOops`**
- jvm interne pointers zijn 32 bit
  - inclusief java references
- decodering naar native:
  - 64-bit basis + 8 * 32-bit waarde
- 4 GB objecten, heaps tot ~32 GB
- ook al beschikbaar in jdk6_u14 (hotspot 14)





matsim benchmark performance



matsim benchmark heap

memory usage [MB]

500
450
400
350
300
250
200
150
100
50
0

Sun 6u14 64bit — Sun 6u14 64bit Compressed Object Pointers — Sun 6u14 32bit

memory measurements over time

.nl.jug

dynamische talen passen slecht op jvm

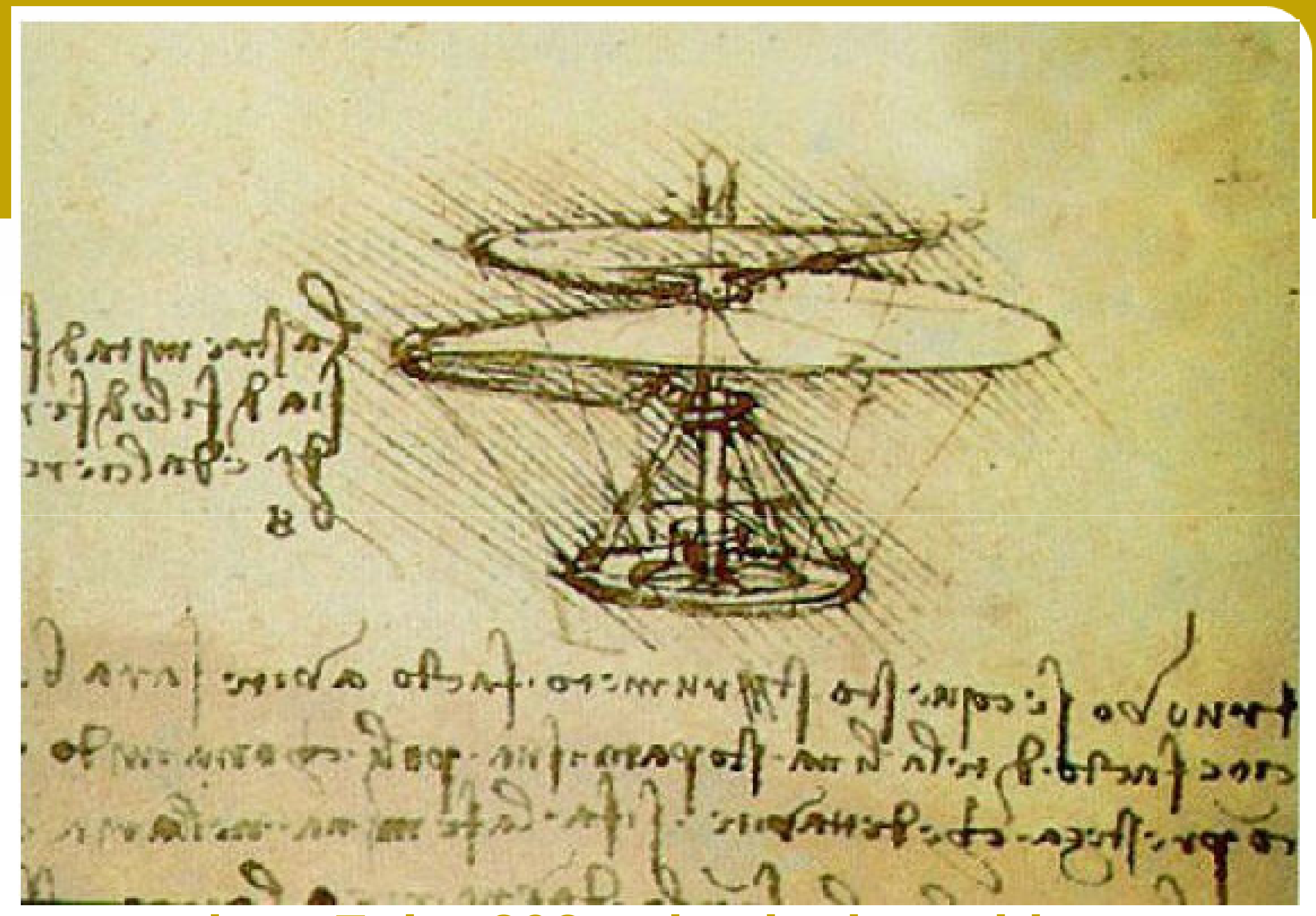
- java 6 introduceerde: jsr 223 - scripting voor het java platform
  - jruby, jython, rhino, groovy, php, clojure
- dynamische methode aanroepen moeilijk te vertalen in jvm bytecode

```
function max (x, y) {
    if x.lessThan(y) then y else x
}
```

- na genereren kunstmatig type:

```
MyObject function max (MyObject x, MyObject y) {
    if x.lessThan(y) then y else x
}
```

- alternatieven:
  - java.lang.reflect.Method, of aanroep interpreteren
- traag



java 7: jsr 292 – da vinci machine

betere ondersteuning voor niet-java talen

oplossing: multi-taal vm

- ondersteuning van dynamisch getypeerde talen
  - nieuwe java byte code: invokedynamic
    - naast invokevirtual, invokeinterface, invokestatic, invokespecial
  - linking met method handles

```
invokedynamic MH.invoke(int) --> MH --> Integer {
                                          int lessThan(int) {..}
                                        }
```

- dl compilers genereren invokedynamic

invokedynamic

- nieuw: target type hoeft niet in byte code
- taal runtime bepaalt target initiëel (*up-call*) en kan dit later nog wijzigen
  - jvm kan bijna direct de target aanroepen:
    - hotspot optimalisaties → snel
- java.dyn.InvokeDynamic, java.dyn.MethodHandle, java.dyn.CallSite, …
  - biedt ook snelle dynamische aanroep mogelijkheden in de java taal

ontsnappen aan de java taal is mogelijk met dolphin

g1 garbage collector

- garbage first: eerst gebieden met veel rommel
- voor multi-processors en grote heaps
- opvolger cms voor korte gc pause tijden
- soft real-time: geen harde garanties op pause tijden

```
-XX:MaxGCPauseMillis=50
-XX:GCPauseIntervalMillis=1000
```

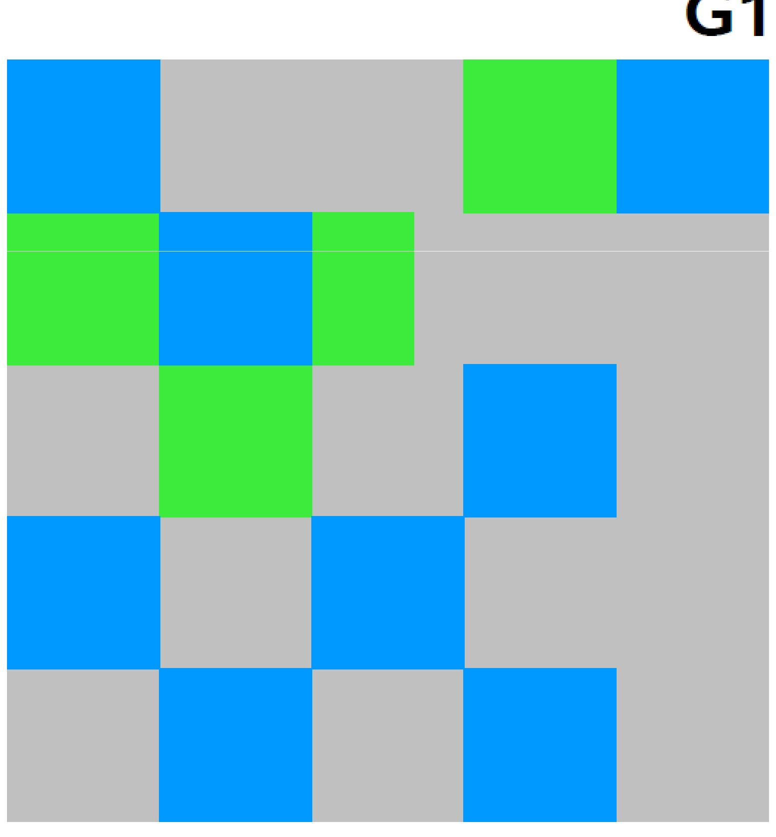
- ook in jdk6u14+:

```
-XX:+UnlockExperimentalVMOptions
-XX:+UseG1GC
```

g1 generaties in veel regios

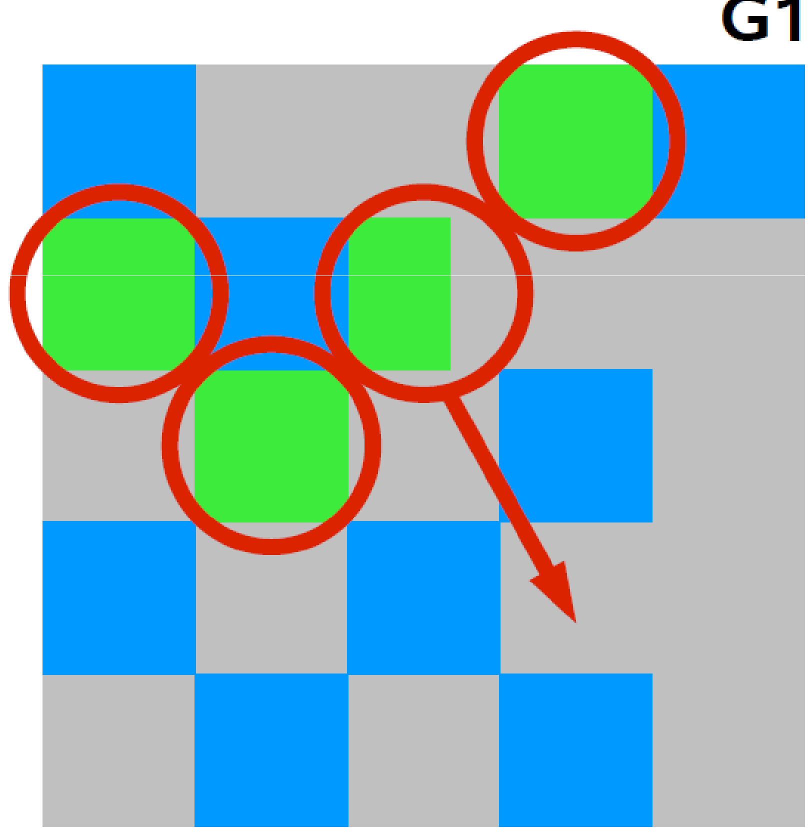
- heap verdeeld in regios
  - 1 mb
- young en old

Non-Allocated Space
Young Generation
Old Generation



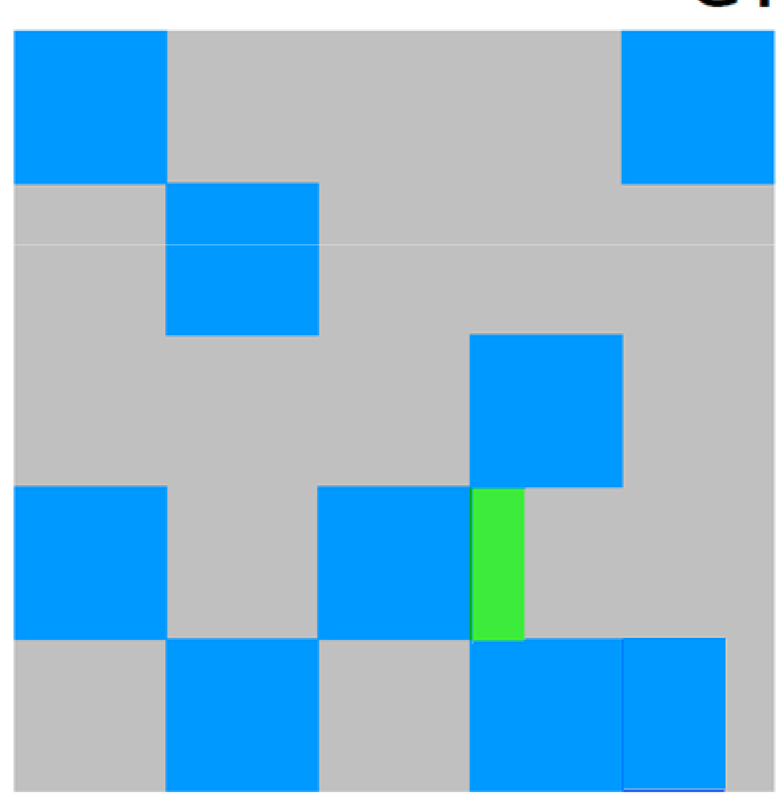
g1 young collect

- live objects gekopiëerd naar
  - survivor regio
  - old regio



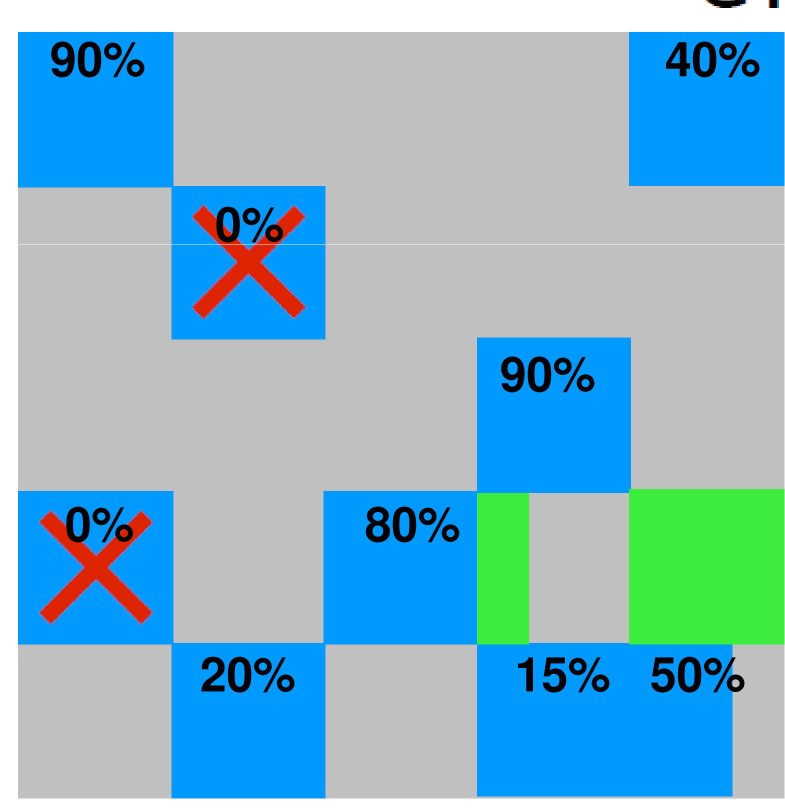
g1 young collect

- lege regio's vrijgegeven
- collect klaar



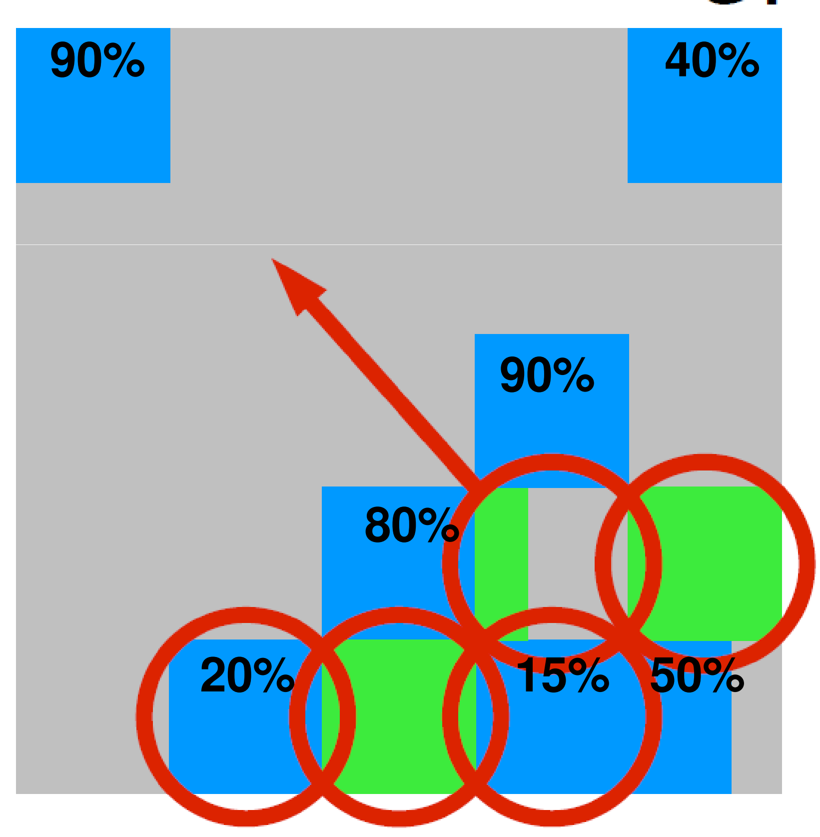
g1 old collect

- parallel/concurrent marking
- live% per regio bekend
- lege regio's vrijgegeven



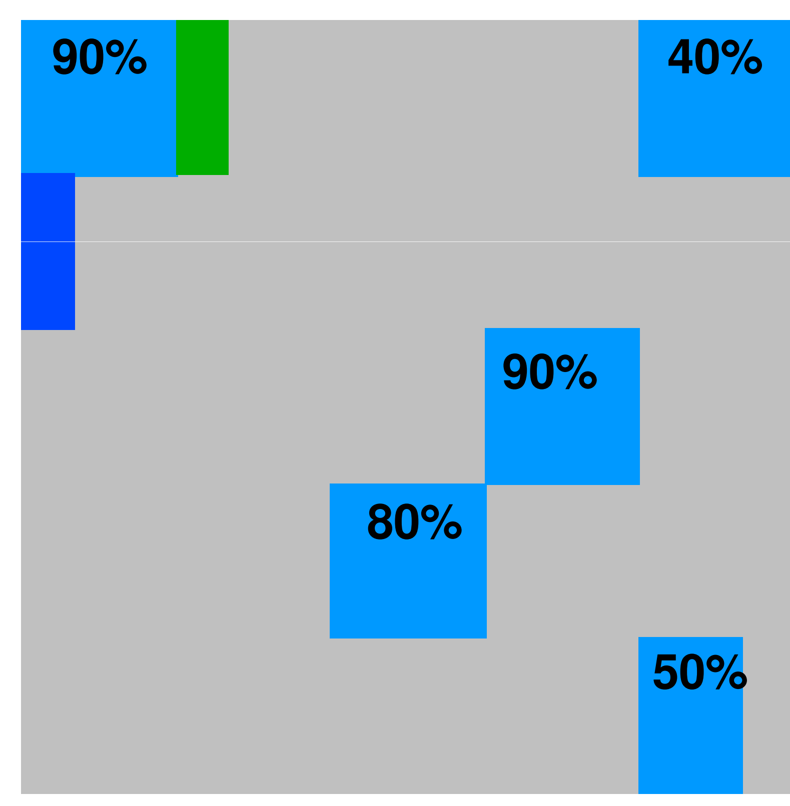
g1 old collect

- collect old regios
  - met young mee
  - met lage live%
  - houd rekening met maximum pauzetijden
- 1½ gc



g1 old collect

- er blijft garbage over
  - in regio's met hoge live%
  - komt later wel
- geen sweep dus geen fragmentatie
  - geen lange gc pauzes



demo g1

```
java -server -XX:+UnlockExperimentalVMOptions -XX:+UseG1GC -verbose:gc
   -Xmx512m Allocator

java -server -XX:+UnlockExperimentalVMOptions -XX:+UseG1GC -
   XX:MaxGCPauseMillis=30 -XX:GCPauseIntervalMillis=100 -verbose:gc -
   Xmx512m Allocator

   public static void main(String[] args) throws InterruptedException {
         for (int i = 0; i < 1000; i++) {
                  Thread.sleep(2);
                  for (int j = 0; j < 5000; j++) {
                           String s = String.valueOf(i);
                           if (i % 2 == 0) {
                                    list.add(s);
                           }
                  }
         }
   }
```

-XX:+UseG1GC

```
[GC pause (young) 7167K->3480K(56M), 0.0141190 secs]
[GC pause (young) 6045K->5077K(56M), 0.0219795 secs]
[GC pause (young) 9172K->8021K(61M), 0.0098968 secs]
[GC pause (young) 16M->12M(65M), 0.0251422 secs]
[GC pause (young) 46M->31M(68M), 0.1021073 secs]
[GC pause (young) 64M->50M(85M), 0.0980120 secs]
[GC pause (young) 85M->70M(125M), 0.1033965 secs]
[GC pause (young) 109M->94M(157M), 0.1041006 secs]
[GC pause (young) (initial-mark)[GC concurrent-mark-start]
 123M->109M(182M), 0.0991442 secs]
[GC pause (young) 139M->124M(202M), 0.1276100 secs]
[GC concurrent-mark-end, 0.6288674 sec]
[GC remark, 0.0002924 secs]
[GC concurrent-count-start]
[GC pause (young) 165M->151M(218M), 0.1139013 secs]
[GC concurrent-count-end, 0.2232887]
```

-XX:+UseG1GC -XX:MaxGCPauseMillis=30 -XX:GCPauseIntervalMillis=100

```
[GC pause (young) 1024K->569K(56M), 0.0055508 secs]
[GC pause (young) 1593K->1072K(56M), 0.0035290 secs]
[GC pause (young) 2096K->1689K(56M), 0.0047902 secs]
[GC pause (young) 2713K->2336K(56M), 0.0059758 secs]
[GC pause (young) 8996K->5725K(61M), 0.0252587 secs]
[GC pause (young) 11M->9237K(65M), 0.0168690 secs]
[GC pause (young) 17M->13M(68M), 0.0236535 secs]
[GC pause (young) 22M->18M(70M), 0.0204467 secs]
[GC pause (young) 25M->22M(72M), 0.0208691 secs]
[GC pause (young) 32M->28M(73M), 0.0251938 secs]
[GC pause (young) 36M->32M(74M), 0.0294801 secs]
[GC pause (young) 44M->40M(75M), 0.0465790 secs]
[GC pause (young) 48M->44M(76M), 0.0421299 secs]
[GC pause (young) 52M->48M(77M), 0.0338746 secs]
[GC pause (young) (initial-mark)[GC concurrent-mark-start]
 56M->52M(118M), 0.0325657 secs]
[GC pause (young) 65M->62M(151M), 0.0306702 secs]
[GC concurrent-mark-end, 0.4122935 sec]
[GC remark, 0.0002662 secs]
[GC concurrent-count-start]
[GC pause (young) 69M->65M(178M), 0.0345765 secs]
[GC concurrent-count-end, 0.1151693]
[GC cleanup 69M->63M(199M), 0.0011035 secs]
[GC concurrent-cleanup-start]
[GC concurrent-cleanup-end, 0.0007728]
[GC pause (young) 66M->63M(199M), 0.0241691 secs]
[GC pause (partial) 69M->66M(216M), 0.0214214 secs]
[GC pause (young) 74M->70M(230M), 0.0232751 secs]
[GC pause (young) 77M->73M(241M), 0.0269685 secs]
[GC pause (young) 80M->77M(250M), 0.0260188 secs]
[GC pause (young) 93M->89M(257M), 0.0285366 secs]
[GC pause (young) 97M->93M(262M), 0.0291883 secs]
[GC pause (young) 101M->97M(266M), 0.0281616 secs]
[GC pause (young) 105M->102M(270M), 0.0291185 secs]
[GC pause (young) 110M->105M(273M), 0.0292420 secs]
[GC pause (young) 113M->110M(275M), 0.0289593 secs]
[GC pause (young) 118M->114M(277M), 0.0286401 secs]
[GC pause (young) 122M->118M(278M), 0.0292285 secs]
[GC pause (young) 125M->121M(279M), 0.0270812 secs]
[GC pause (young) 142M->138M(280M), 0.0282064 secs]
[GC pause (young) 147M->143M(281M), 0.0329289 secs]
```

escape analysis

```
public String concat(String s1, String s2, String s3) {
       StringBuffer sb = new StringBuffer();
       sb.append(s1);
       sb.append(s2);
       sb.append(s3);
       return sb.toString();
}
```

- optimalisaties:
  - lock elision
  - stack/register allocatie
- jdk 6u4 alleen zeer beperkte lock elision
  - mijn infoq/java magazine artikel: “do java 6 threading optimizations actually work?”

escape analysis – lock elision
het werkt, eindelijk!

tijd [ms]

	StringBuffer	StringBuilder

Chart axis: 0, 500, 1000, 1500, 2000, 2500, 3000

Categories: jdk 6_12, jdk 6_14, jdk7_ea_b75

Legend: StringBuffer, StringBuilder

infoq micro benchmark

en escape analysis – allocation?

8000
7000
6000
5000
4000
3000
2000
1000
0

tijd [ms]

jdk 6_12
jdk 6_14
jdk7_ea_b75

EA nonactief
EA actief

vector micro benchmark

escape analysis - allocation

```
class Vector {
    double x, y;
    public void add(Vector other) {
        x += other.x; y += other.y;
    }
    public double length() {
        return Math.sqrt(x * x + y * y);
    }
}
public static double escapeAnalysisRules(int count) {
    Vector result = new Vector(0, 0);
    for (int i = 0; i < count; i++) {
        result.add(new Vector(count, count));
    }
    return result.length();
}
```

arrays

- elke array element gebruik: run-time binnen-grenzen check
- als tijdens jit compilatie blijkt dat de check nooit faalt → check verwijderen

```
if ((i<0) || (i>=a.length) ) {
    throw new AIOOBExc();
}
```

```
int[] a = new int[10];
for (int i = 0; i < a.length; i++) { a[i] = i }
```

- checks kunnen buiten loops gehaald worden
- al in -server (since 1.3.1) nu ook in -client

agenda

- introductie
- jvm
- **java taal**
  - automatisch resource management
- core api's
- conclusies

puzzler: what is wrong?

```
try {
   URLConnection conn = url.openConnection();
   is = conn.getInputStream();
   while (is.read(buf) > 0) { processBuf(buf); }
   is.close();
} catch (IOException e) {
   try {
        respCode = ((HttpURLConnection)conn).getResponseCode();
        es = ((HttpURLConnection)conn).getErrorStream();
        while (es.read(buf) > 0) { processBuf(buf); }
        es.close();
   } catch(IOException ex) {// deal with the exception }
}
```

automatic resource management

- sun voorbeeld, java puzzler, 2/3 jdk fout! – Josh Bloch
- Closable resources vrijgeven niet meer handmatig

```
BufferedReader b = new BufferedReader(new FileReader(path));
try {
   return b.readLine();
} finally {
   b.close();
}
```

automatic resource management

- sun voorbeeld, java puzzler, 2/3 jdk fout! – Josh Bloch
- Closable resources vrijgeven niet meer handmatig

```
BufferedReader b = new BufferedReader(new FileReader(path));
try {
   return b.readLine();
} finally {
   b.close();
}
```

```
try (BufferedReader b = new BufferedReader(new FileReader(path))) {
   return b.readLine();
}
```

agenda

- introductie
- jvm
- java taal
- **core api’s**
  - strings
  - fork-join framework
  - asynchrone i/o
- conclusies

strings

- encoding/decoding naar bytes was traag
  - `String(byte[] bytes, String charsetName)`
  - `byte[] getBytes(String charsetName)`
- onnodige object creatie + defensief kopieëren
  - -36% objects gealloceerd
- versnellingsfactor: 2-4

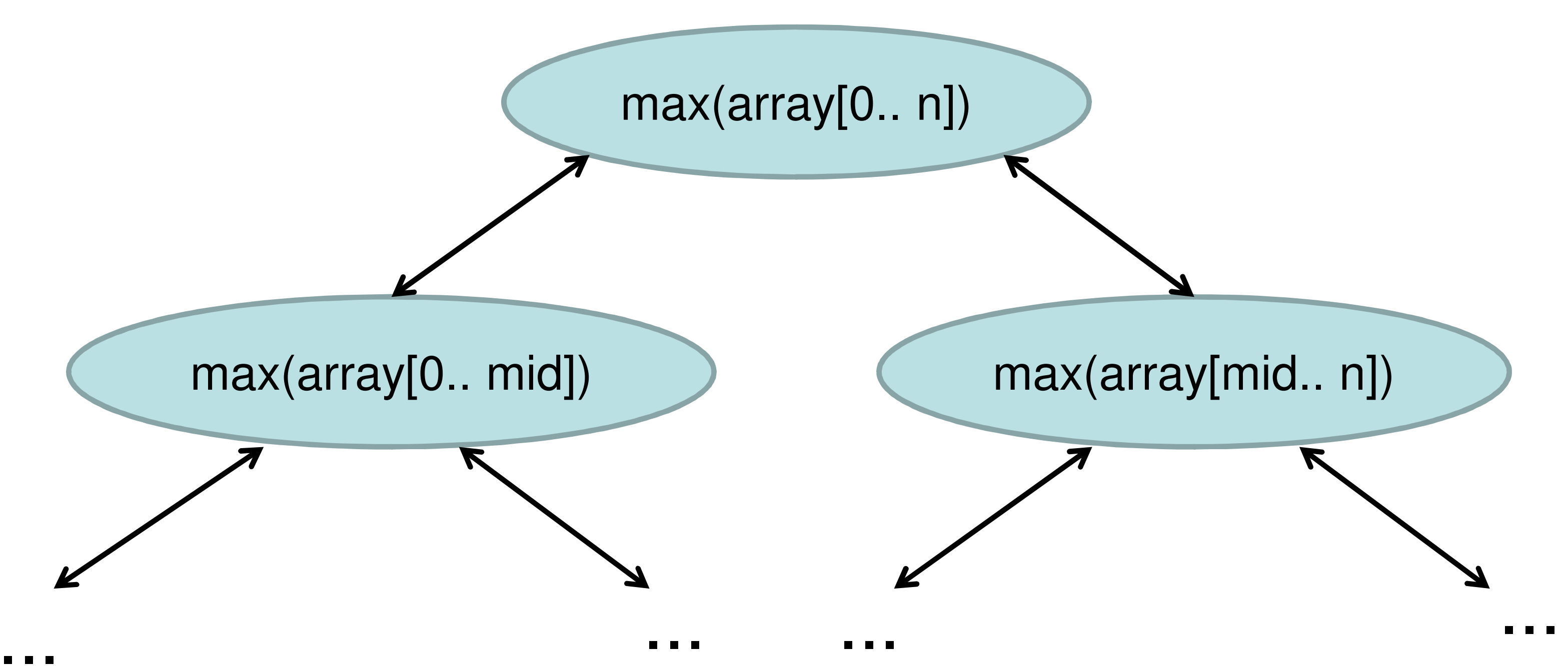
concurrency library: fork-join

- hardware trends sturen programmeer idioms
- asynchroon
  - Java 1, 1995: `synchronized, wait/notify, Thread`
- concurrency, parallelisme: multi-core processors
  - Java 5, 2004: java.util.concurrent
- high parallelisme: many-core processoren(100+)
  - om aan Moore's law te voldoen
  - hoe benutten we al deze cores?
  - java 7, 2009: java.util.concurrent.forkjoin

java.util.concurrent.forkjoin

- user request-response is te grofmazig
  - per request parallellisatie nodig
- voor cpu intensief werk zoals `Arrays.sort()`
- gebruikt `Deques` en *work* *`stealing`* voor minimale contention
- efficiënte uitvoer op veel hardware

devide and conquer algoritmes



fork-join voorbeeld

```
class MaxSolver extends RecursiveAction {
    private final int[] array, int lo, int hi;
    int result;
    protected void compute() {
            if (hi – lo < THRESHOLD)
                        //result = solveSequentially: max of array[lo..hi]
            else {
                        int mid = (lo + hi) / 2;
                        MaxSolver left = new MaxSolver(array, lo, mid);
                        MaxSolver right = new MaxSolver(array, mid, hi);
                        invokeAll(left, right);
                        result = Math.max(left.result, right.result);
            }
    }
}
ForkJoinExecutor pool = new ForkJoinPool(Runtime.availableProcessors());
MaxSolver solver = new MaxSolver(array, 0, array.length);
pool.invoke(solver);
return MaxSolver.result;
```

nio.2

- < 1.4 alleen thread-geörienteerd, blocking i/o
- nio jsr 51 in jdk 1.4 bracht non-blocking i/o
  - schaalbaarder
  - *selectors*, *selectable channels*, en *selection keys*.
- bijv. voor web & app servers
  - grizzly framework gebruikt in glassfish, jetty
- nio.2 jsr 203 brengt:
  - verbeterd filesysteem interface, socket channel api
  - asynchrone i/o naar sockets en files
    - polling met java.util.concurrent.Future
    - callbacks met java.nio.channels.CompletionHandler

asynchrone i/o code

```
AsynchronousSocketChannel input = ...
ByteBuffer buffer = ByteBuffer.allocate (MESSAGE_SIZE);

input.read(buffer, null, new CompletionHandler<Integer, Void>) {
   public void completed (Integer result, Void attachment) {..}
   public void failed(Throwable t, Void attachment) {..}
});

doSomethingElse();
...
```

asynchrone i/o

- maakt zomogelijk gebruik maken van asynchrone os functies
  - performance afhankelijk van os
  - windows, solaris goed
  - kan zonder cpu gebeuren
- gebruikt fixed of cached thread pools voor CompletionHandler
- proactor i.p.v. reactor design pattern

conclusies

- je kunt nu gebruik maken van de snellere jvm
- meeste jvm verbeteringen ook in java 6u14+
  - via jvm opties, vaak niet default
    - -XX:+DoEscapeAnalysis
- automatische Closable resource management
- aantal nuttige api’s voor many-core tijd
  - fork-join
  - asynchrone i/o

benut de mogelijkheden, have fun!

- meer weten?
  - volg speeding up java applications training
  - lees blog.xebia.com/category/performance
- vragen?