
The C++ Lands
Its amazing creatures and weird beasts
explorer
Alena (http://alenacpp.blogspot.com)
cartographer
Jim (http://jim.pp.ru)


TERRA INCOGNITA
STL
BOOST
λ
external libraries seas
Loki
THE HOLY STANDARD TERRITORY
The C++ Lands
Its amazing creatures and weird beasts
Memory management desert

0. Einführung & Motivation

Ansatz: "C++ für Java-Kenner"

- Konzentration auf semantische Unterschiede 'gleichartiger' Konzepte
- Erörterung der C++ -spezifischen Konzepte (Overloading, Templates)

Anspruch auf Vollständigkeit

Sprache laut Standard **ISO/IEC 14882 von 2011 (2014)**
incl. Standardbibliothek (**STL and more**)

Schwerpunkt auf Diskussion von Konzepten anhand von Beispielen

0. Einführung & Motivation

Warum noch eine OO-Sprache?

- die zudem
  - syntaktisch sehr ähnlich zu Java ist
  - älter ist als Java ...
  - 'gefährlicher' ist als Java ...

Weil C++ eine Sprache ist

- die
  - syntaktisch sehr ähnlich zu Java ist
  - älter ist als Java ...
  - 'gefährlicher' ist als Java ...

-- und potenziell effizienteren Code ermöglicht

http://www.research.att.com/~bs/applications.html

0. Einführung & Motivation

Java

- zahlreiche Sicherheitsvorkehrungen
  kosten Zeit & Raum
- virtual machine
- architekturneutral

C++

- keinerlei Sicherheitsvorkehrungen
  Reserven für Zeit & Raum
- native code
- architekturabhängig

Ein erster Blick: Hello World

Java

Hello.java

```
class Hello {
        public static void main(String s[]) {
                if (s.length < 1) return;
                Hello h = new Hello(", " + s[0]);
                h.speak();
        }
        String what;
        void speak() {
                System.out.println("Hello" + what);
        }
        Hello(String s) {
                this.what = s;
        }
        protected void finalize() {
                System.out.println("bye, bye");
        }
}
```

Ein erster Blick: Hello World

C++

h.cc

Java-Style

```
#include <iostream>
#include <string>
class Hello {
public: static void main(int c, char* v[]) {
                    if (c < 2) return;
                    Hello h = Hello(", "+std::string(v[1]));
                    h.speak();
          }
private: std::string what;
          virtual void speak() {
                    std::cout << "Hello" + what << std::endl;
          }
          Hello(std::string s) {
                    this->what = s;
          }
          ~Hello() {
                    std::cout << "bye, bye" << std::endl;
          }
}; // !!!!!!!!!!
```

Ein erster Blick: Hello World

C++

h.cc

Java-Style

```
// ... continued

int main(int argc, char* argv[])
{
        Hello::main(argc, argv);
}
```

Ein erster Blick: Hello World

C++

h0.cc

C - Style

```
#include <cstdio>

int main(int argc, char* argv[])
{
        if (argc > 1)
           std::printf("Hello, %s\n", argv[1]);
}
```

Ein erster Blick: Hello World

- in C++ kann man offenbar Java-like (OO) programmieren, muss es aber nicht:
  - C++ ist eine sog. **multi-paradigm**-Sprache
- Abweichungen in syntaktischen Feinheiten
- semantische Unterschiede

```
Java:   Hello h = new Hello(....);      // reference !
        h.speak();
C++:    Hello h = Hello(....);          // value !
        h.speak();
```

Ein erster Blick: Hello World

In C++ muss jeder verwendete Bezeichner zuvor (oder in der gleichen Klasse) deklariert werden!
(auch Bezeichner aus der Standard-Bibliothek)

```
#include <string> .... std::string
```

statt → /usr/include/g++/string

```
String (--->java.lang.String)
```

Ein erster Blick: Hello World

- In C++ gibt es globale Funktions- (Variablen- und Typ-) Deklarationen
- Es gibt geschachtelte Gültigkeitsbereiche (Klassen und namespaces) aber ohne implizite Abbildung auf eine hierarchische Verzeichnisstruktur
- Ein Compilerlauf behandelt GENAU EINE Quelldatei pro Aufruf! (.... make !)
- Dies entspricht dem klassischen Paradigma der Übersetzung von C-Programmen: ermöglich Migration, Portierbarkeit, Unix-Konformität

Ein erster Blick: Hello World

- In C++ gibt es Zeiger, Felder sind de facto Zeiger - keine Objekte, this ist ein Zeiger !
- Konvention des Programmaufrufs ist etwas anders
- Virtualität ist explizit zu spezifizieren
- Es gibt sog. Destruktoren
- Syntax von Zugriffsrechten ist etwas anders

Der zweite Blick: Effizienz

Problem 1: integer - Arithmetik

$$3^{10^n}$$

(modulo int-overflow)

Problem 2: double - Arithmetik

$$e \sim (1+ 1/n)^n$$

[n = 10e8, 10e9, 10e10]

Der zweite Blick: Effizienz

```
class i {
        public static void main(String []s)
        {
                int i=1;

                for(int n=1; n<=100000000;++n)
                        i*=3;
                System.out.println( i );
        }
}
```

Java

long
bei 10^{10}

10000000000L
bei 10^{10}

Der zweite Blick: Effizienz

```
#include <iostream>

class i {
public:
static void main()
        {
                int i=1;

                for(int n=1; n<=100000000;++n)
                        i*=3;
                std::cout << i << std::endl;
        }
};

int main() {
        i::main();
        return 0;
}
```

long
bei 10^{10}

10000000000L
bei 10^{10}

C++

Der zweite Blick: Effizienz

```
class d {

        public static void main(String []s)
        {
                double e=1;

                for(int n=1; n<=100000000;++n)
                        e*=1.00000001;
                System.out.println( e );
        }
}
```

Java

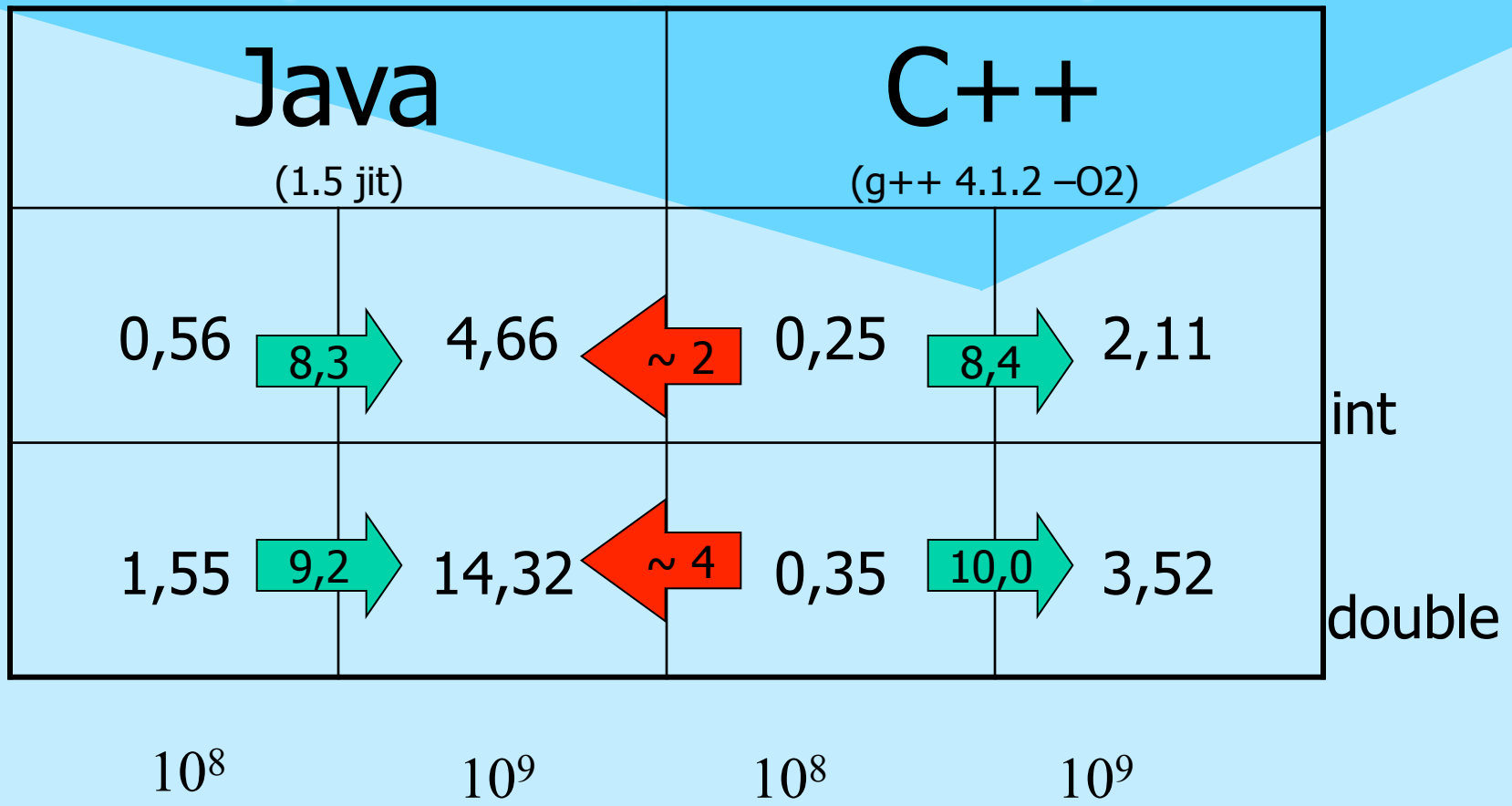
Der zweite Blick: Effizienz

```
#include <iostream>

class d {
public:
static void main()
        {
                double e=1;

                for(int n=1; n<=100000000;++n)
                        e*=1.00000001;
                std::cout << e << std::endl;
        }
};

int main() {
        d::main();
        return 0; // not needed but good style
}
```

C++

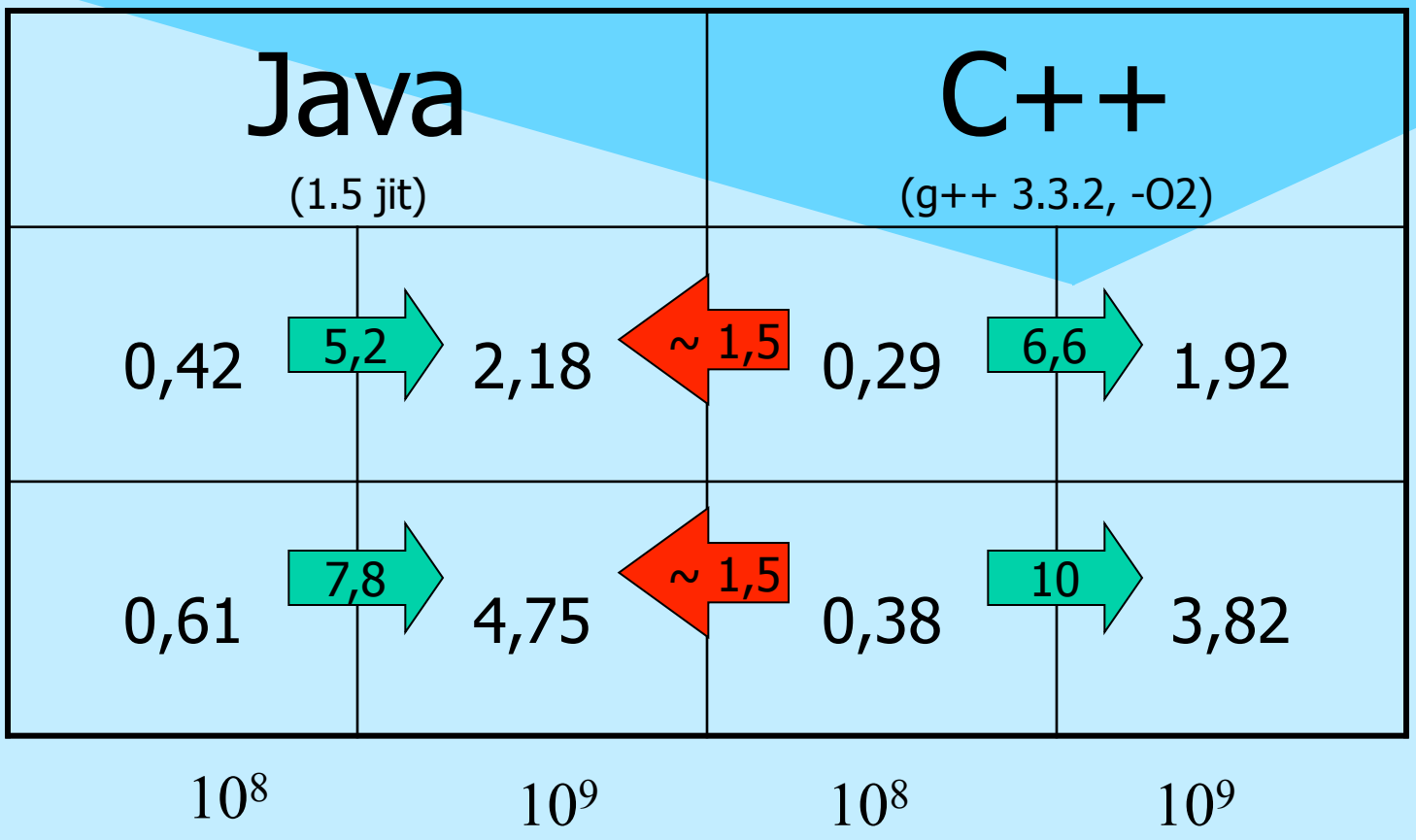
Der zweite Blick: Effizienz

Laufzeiten (Debian Linux, Pentium4 2 GHz)



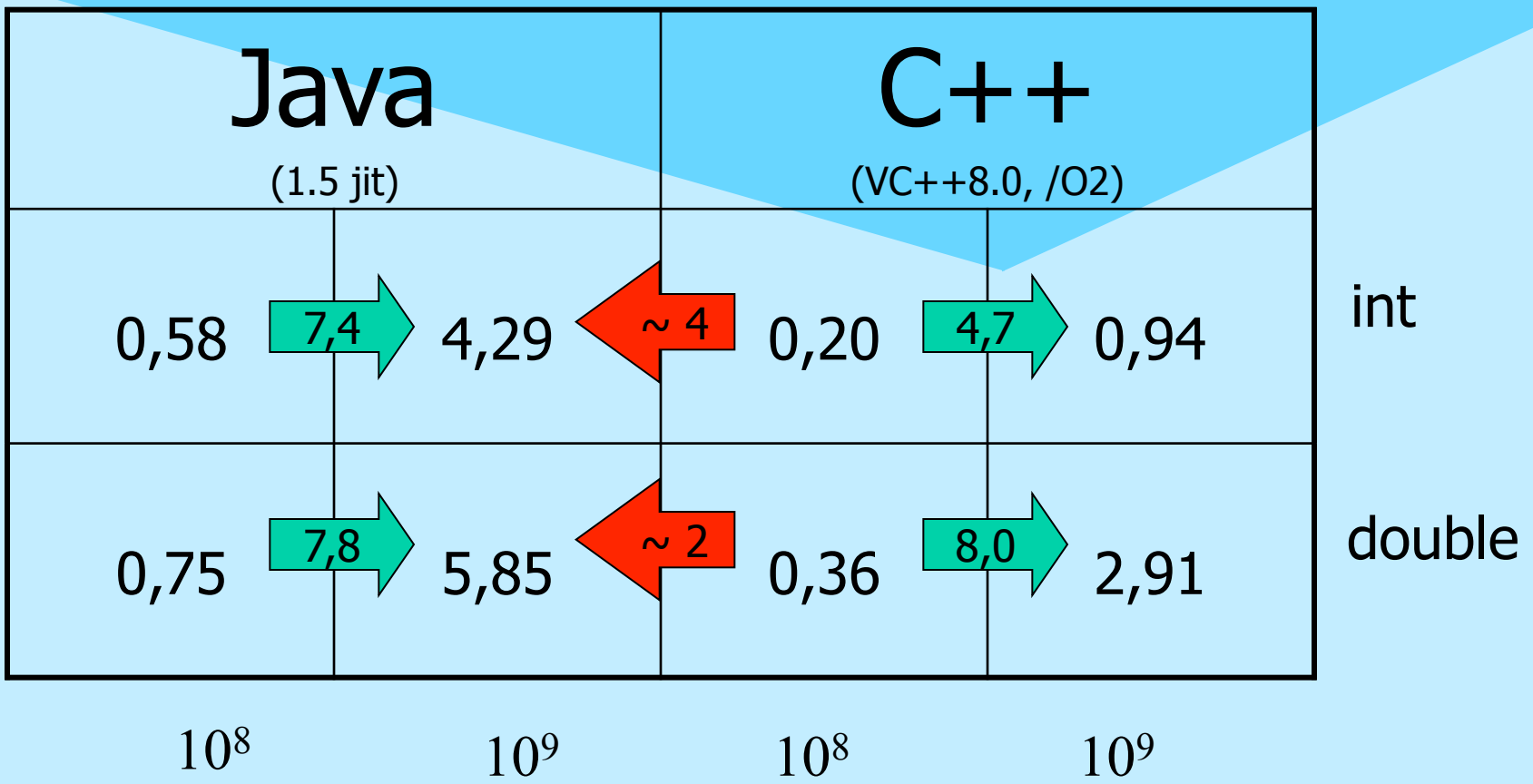
Der zweite Blick: Effizienz

Laufzeiten (Solaris 5.8, UltraSparc [8x]1050 MHz)



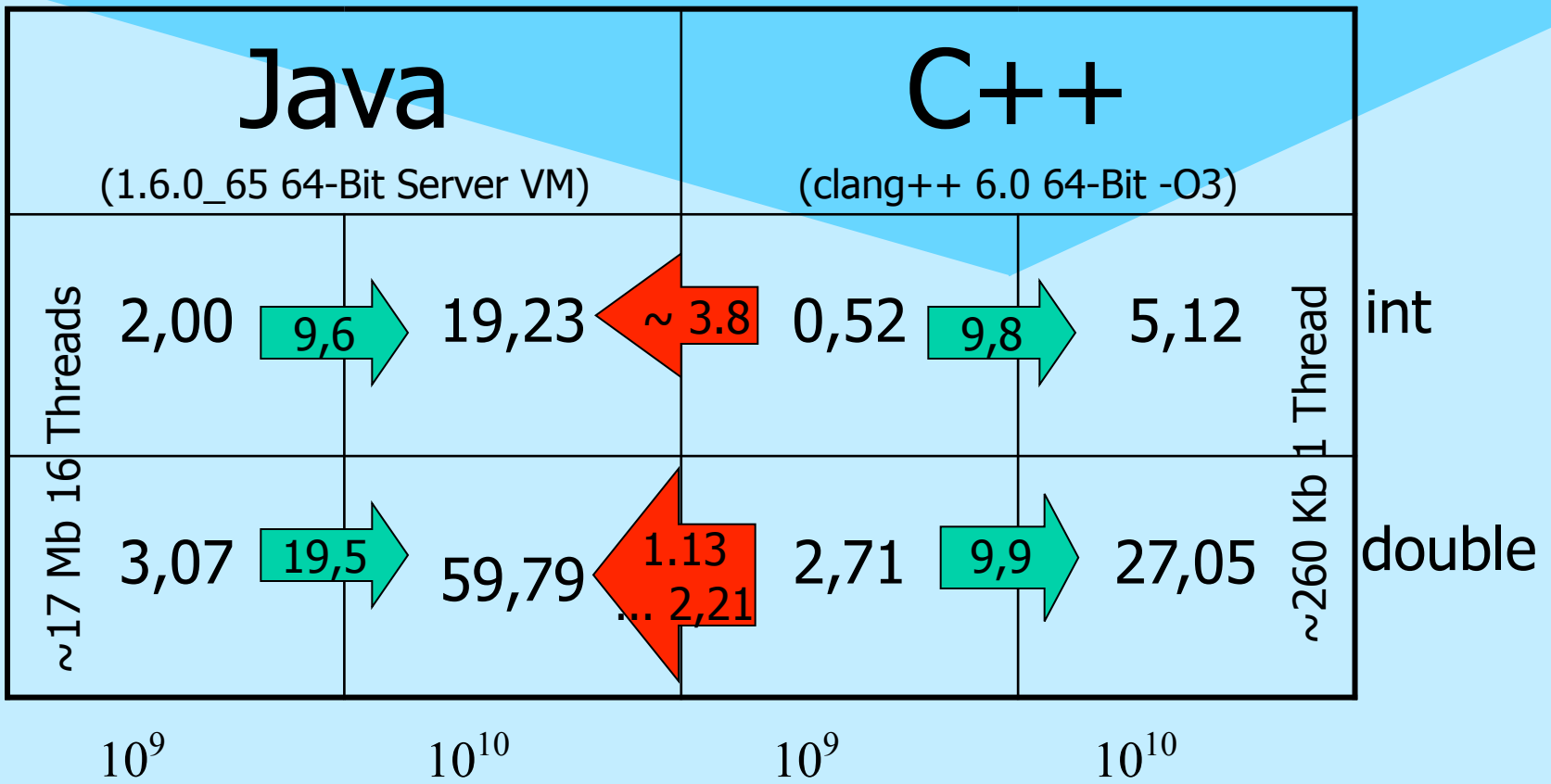
Der zweite Blick: Effizienz

Laufzeiten (Win XP, Pentium M 1,8 GHz)



Der zweite Blick: Effizienz (up to date :-)

Laufzeiten (Mac OS X 10.9.5, 1.86 GHz Intel Core 2 Duo)



Der zweite Blick: Effizienz

Problem 3: sorting really many, many, many strings

erzeuge N zufällig (4..23) lange Zeichenketten mit zufälligem Inhalt aus Kleinbuchstaben ('a' .. 'z') uns sortiere diese lexikographisch

$$[\ N = 1*10^7, 2*10^7, 4*10^7, 8*10^7\ ]$$

N=10

```
eenpphkhthbkwyrwamnug
eylnbxwtkrtkdmfwnawa
eyxmwsylkxsv
fwbghxmshwrlyuj
gizyvmfohigeabgksfnb
hppfxhdikqtpnlqiyjyanha
jhrsovkv
mffxjffqbua
qrphyrbjqdjqavctgx
vjlixapejrb
```

Der zweite Blick: Effizienz (strings)

Laufzeiten (SunOS 5.11, Sparc64 [32x] 2530 MHz)

n * 10	Java (1.7.0_55 64-Bit Server VM)	C++ (g++ 4.5.2 -O3)
1	494s	63s
2	OutOfMemory after 4241s	134s
4	no way :-(	281s
8	:-(((((	601s

~ 7.8