Deutsche Bank
Group Technology & Operations

GC: FRIEND || FOE

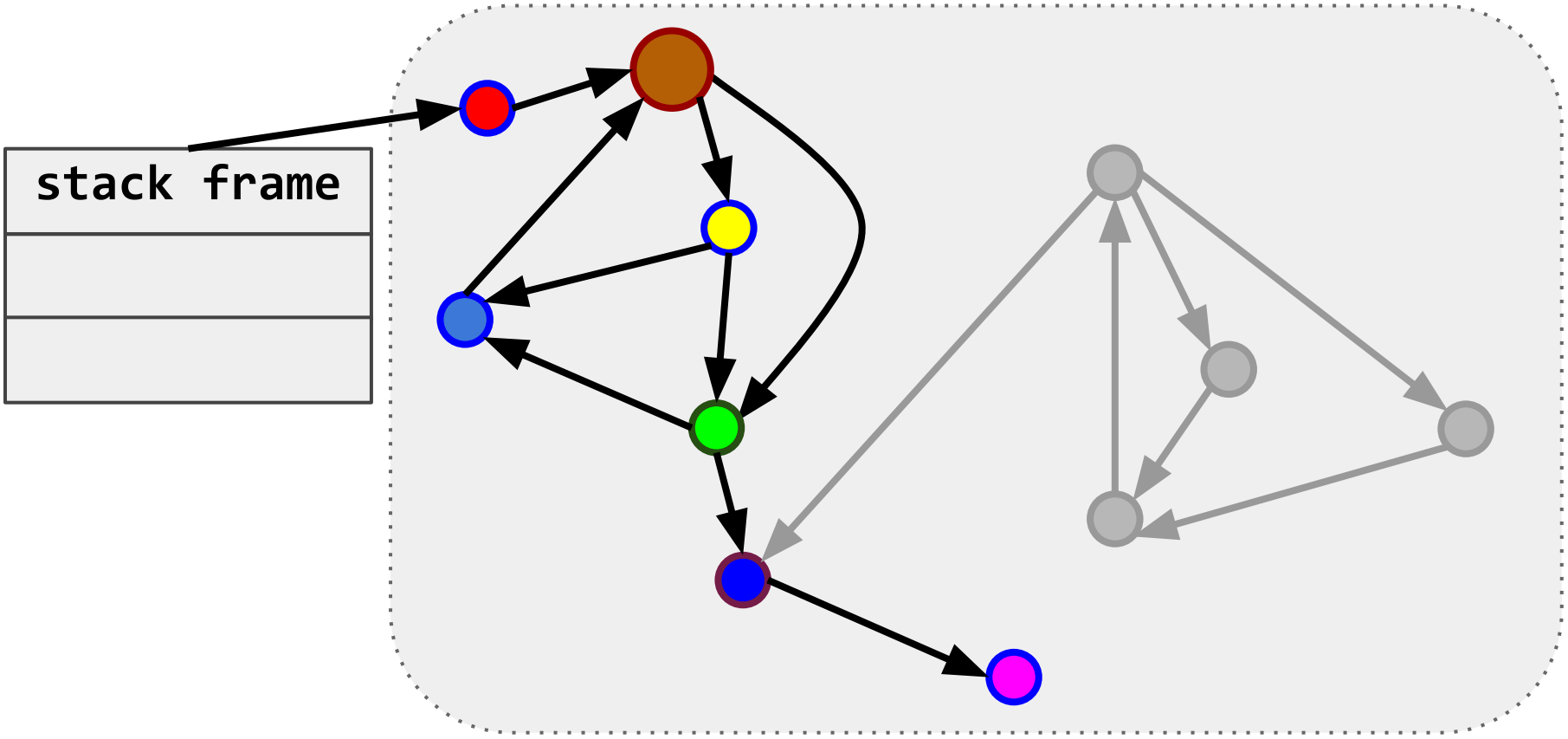
Vladimir Dolzhenko,
application developer at Deutsche Bank

Passion to Perform

Управление памятью

- Выделение
  - статическое
  - динамическое
    - *alloc
    - new
- Использование
- Освобождение
  - ручное
    - free, delete
  - garbage collector
    - lisp, haskell, java, python, erlang etc

Основной принцип работы GC: достижимость объекта



заморозка дерева и умерщвление объектов

Мониторинг stop-the-world

```
java

-verbose:gc -XX:+PrintGC
-XX:+PrintGCDetails
-XX:+PrintGCApplicationStoppedTime

c.d.f.MyApp
```

постоянный мониторинг приложения

Меньше ощущать GC

- Настройка параметров GC
  - выбор JVM
  - типы Garbage Collector-ов
  - начальные и максимальные размеры кучи
  - настройка поколений
  - и т.д
- Garbage-Less
- Выпилить GC из JVM
  - open jdk

мусорить надо меньше !

Зачем Garbage Less ?

- повышение производительности
  - явное
    - меньше latency
    - б***о***льшая пропускная способность
  - неявное
    - меньше и реже stop-the-world паузы
    - переработка дизайна, API, кода

как один из приёмов оптимизации приложения

Garbage-Less рекомендации

- выбор *правильных* структур данных
- garbage-less API
- преаллокация

бритва Оккама:
не плодить новые сущности без надобности

Правильные структуры данных

- структуры данных с необходимой спецификой
- избегать auto-boxing
  - цепочек inboxing-outboxing-inboxing
- избегать объекты-обёртки
- коллекции для примитивных типов
- в многих случаях LinkedList неуместен
- mutable и immutable объекты
- ручное управление памятью
  - непрерывный блок памяти для ~*blob*-объектов

Garbage-Less API

- пример **не** garbage-less friendly API:

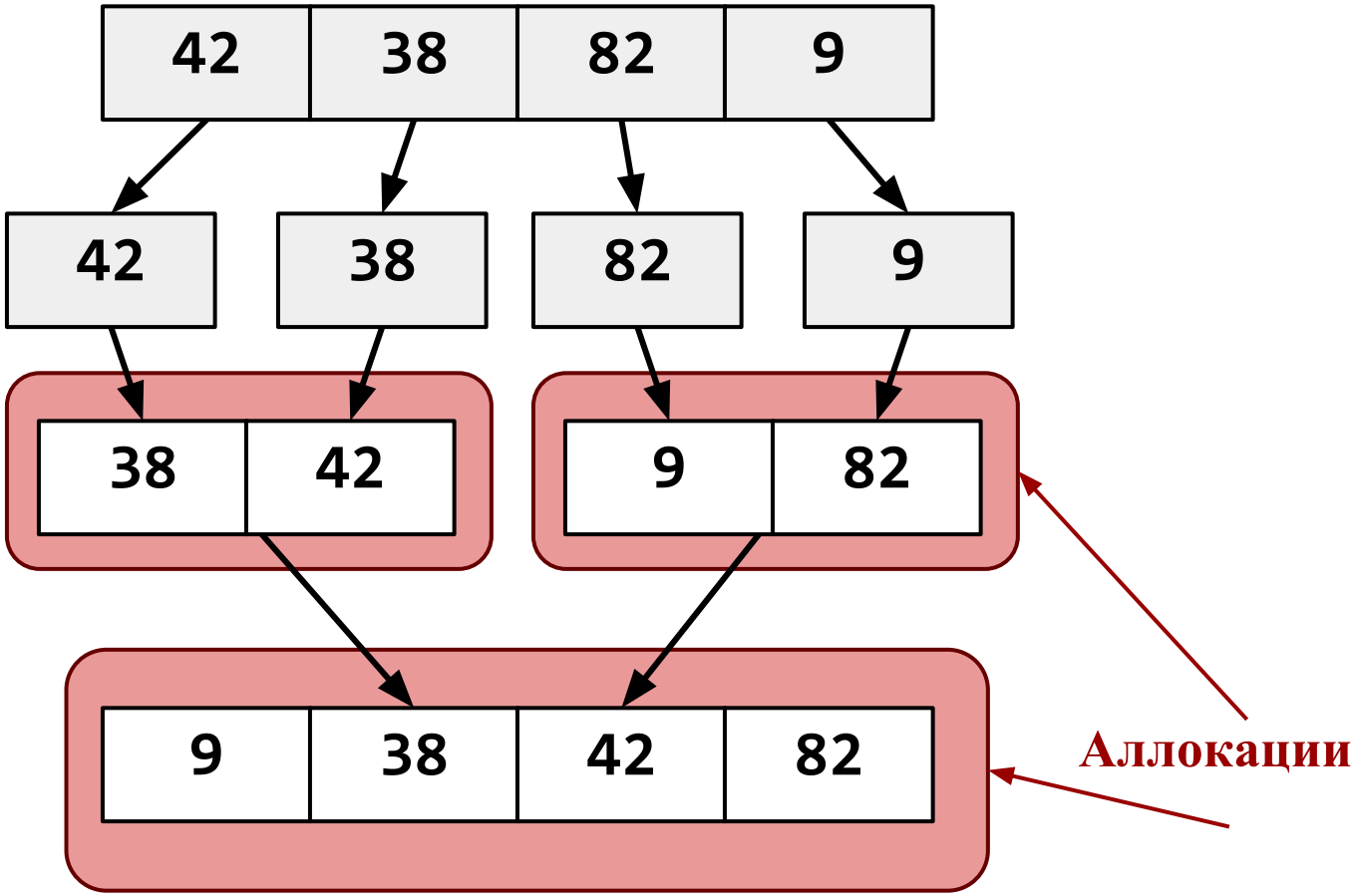
```
void smthMethod( Object ... args );

Iterator<T> iterator();
```

Не впадайте в солипсизм !

Preallocation

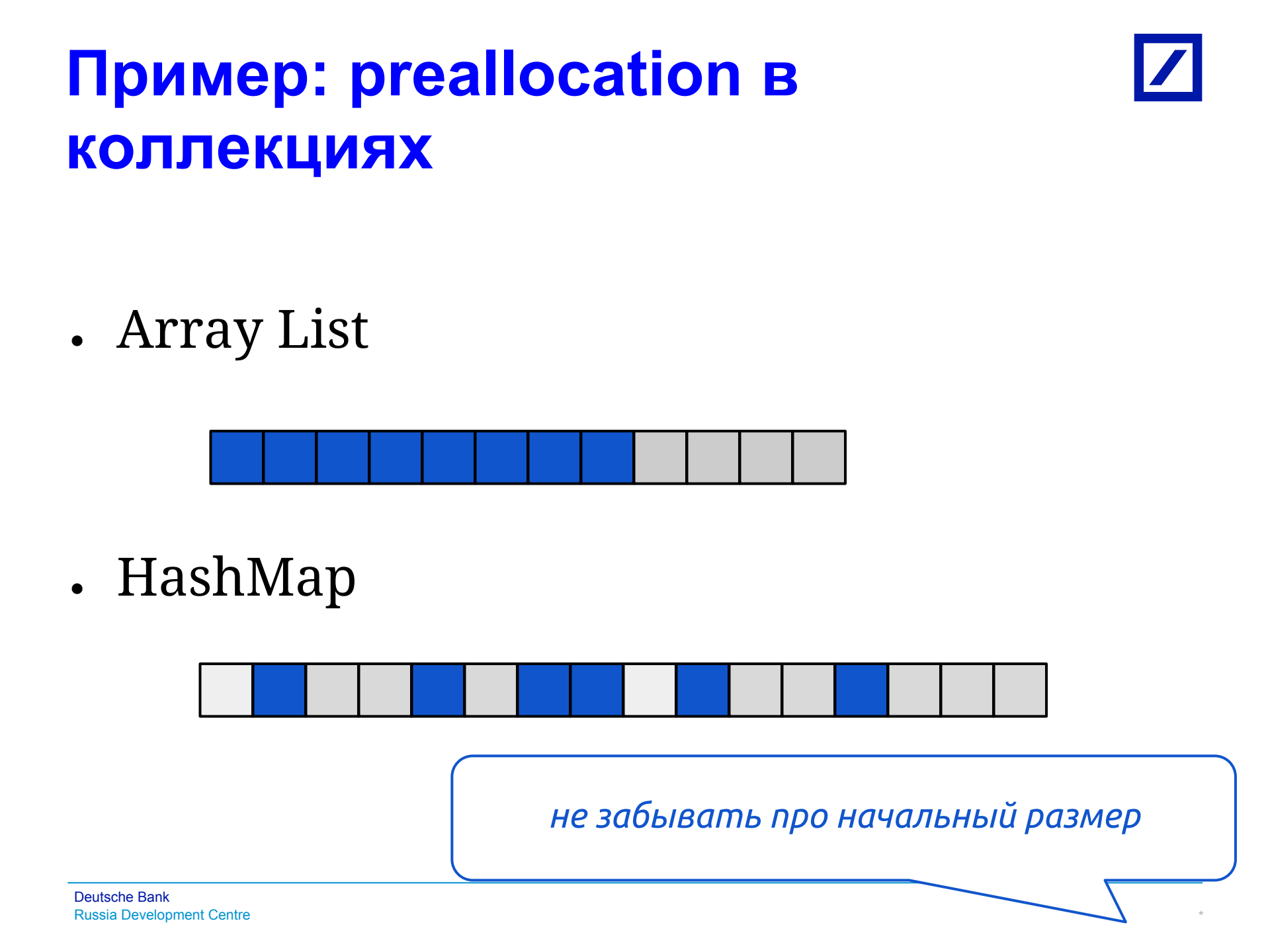
- Буферы
  - Массивы
  - Коллекции
- Потоки
  - threads, i/o streams
- Переиспользование объектов

Merge Sort



*нужен только **один** вспомогательный буфер*

Пример: preallocation в коллекциях



- Array List

- HashMap

не забывать про начальный размер

http://bitbucket.org/vladimir.dolzhenko/gflogger

Конкуренты до...

- log4j
  - плюсы : универсальность и т.п
  - минусы :
    - new StringBuilder, new String
- logback
  - плюсы : универсальность и т.п
  - минусы: varargs, autoboxing

В поисках мусора

- Метод *пристального взгляда*
- Получение heap dump
- jvisualvm
- GarbageCollectorMXBean
- Инструментирование байт-кода с помощью java agent

всё держать под контролем

Allocation callback

http://code.google.com/p/java-allocation-instrumenter

```
AllocationRecorder.addSampler(new Sampler() {

    public void sampleAllocation( String desc,
        Object newObj, long size) {

   System.out.println( " allocated " +
      newObj + " of type " + desc +
      " whose size is " + size );

    }

});
```

Так вот не хитро можно следить за каждым созданным объектом.

Начнём ...

Считать количество созданных объектов используя инструментарий:

log4j и logback:

```
log.info("value");
```

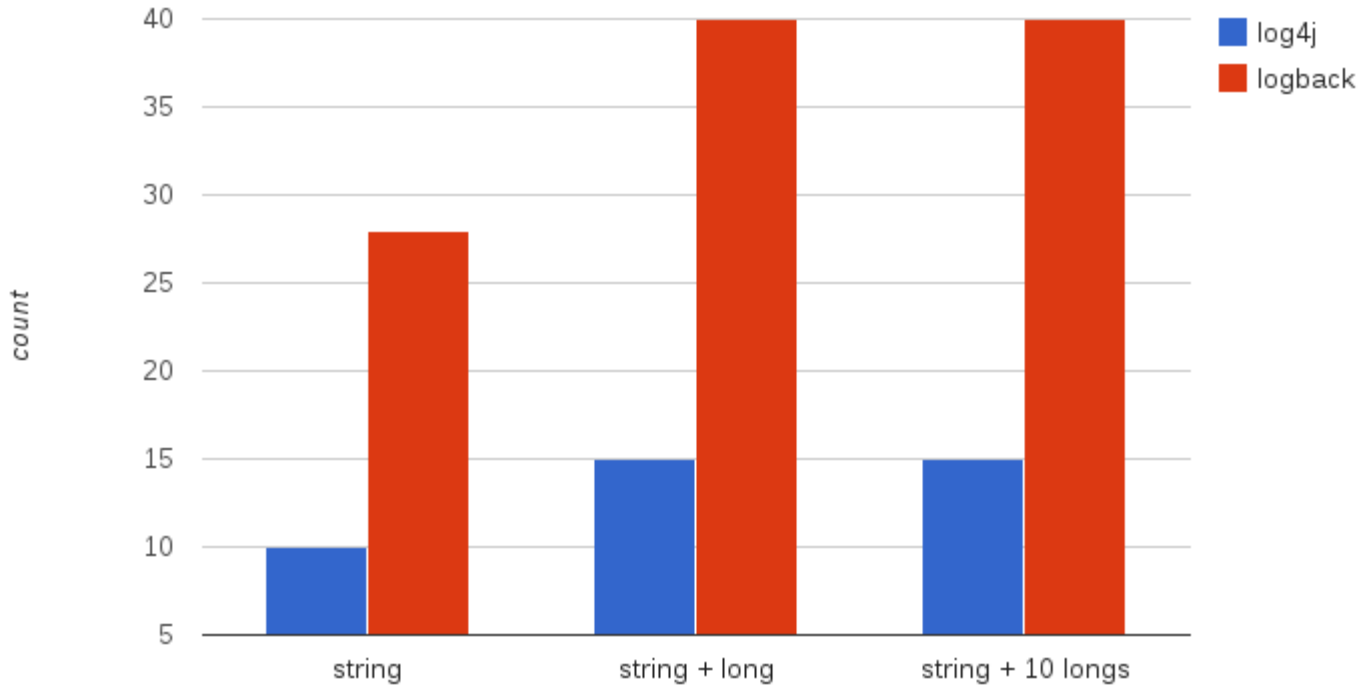
log4j:

```
log.info("value:" + v);
log.info("value:" + v + " " + v + " " +
   ... + v);
```

logback:

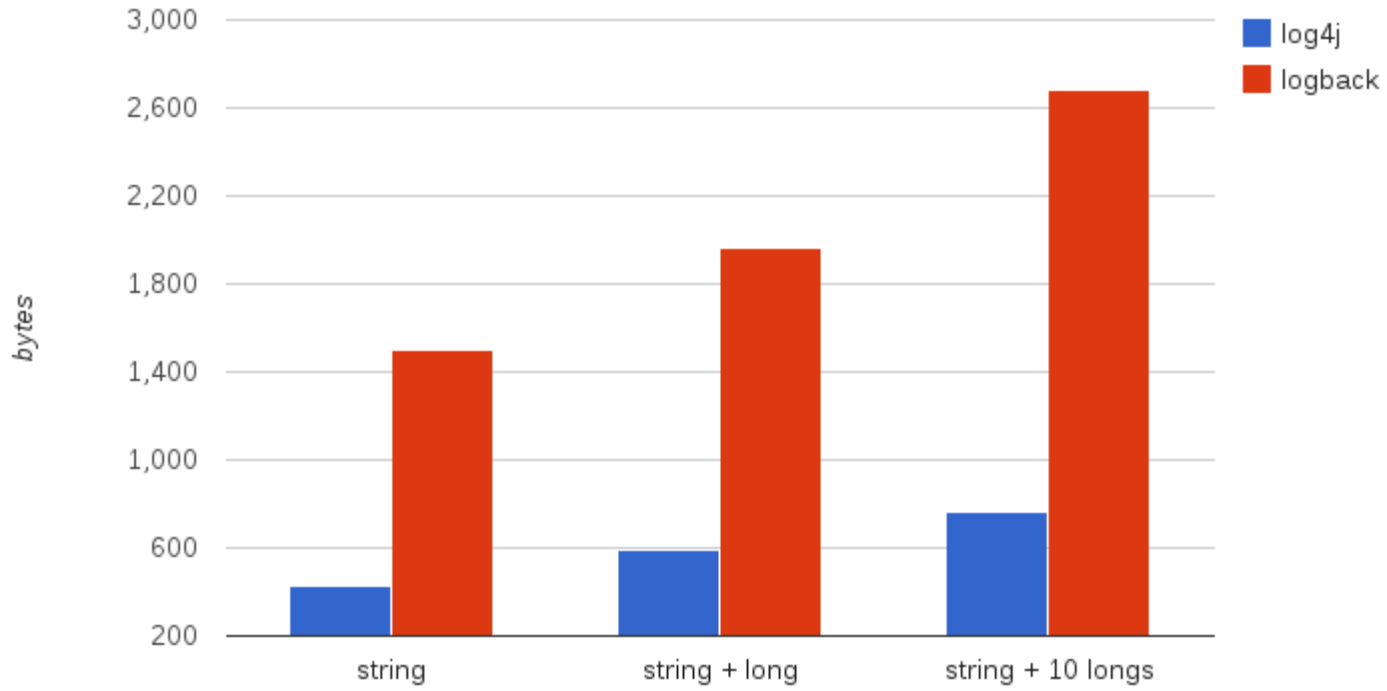
```
log.info("value:{}", v);
log.info("value:{} {} ... {}", v, ..., v);
```

Количество созданных объектов (на одно сообщение)



получено с помощью allocation instrumentation

Потребление памяти (на одно сообщение)



получено с помощью allocation instrumentation

gflogger

- логгер для критических к latency блоков кода
  - т.е **не** универсальный логгер на каждый день
  - не для холодных блоков и начальной фазы
- меньше неявно замедляет приложение посредством
  - нагрузки на gc, ведущим к stop-the-world паузам
  - выделения памяти
- выше пропусканая способность в сравнении с log4j, logback

логирование - вспомогательная функциональность!

Принципы

- преаллокация и переиспользуемость
- single writer
- lock free
- garbage-free форматирование: *boolean*, *integer*, *long*, *double* и *pluggable*
- схожесть конфигурирования с log4j

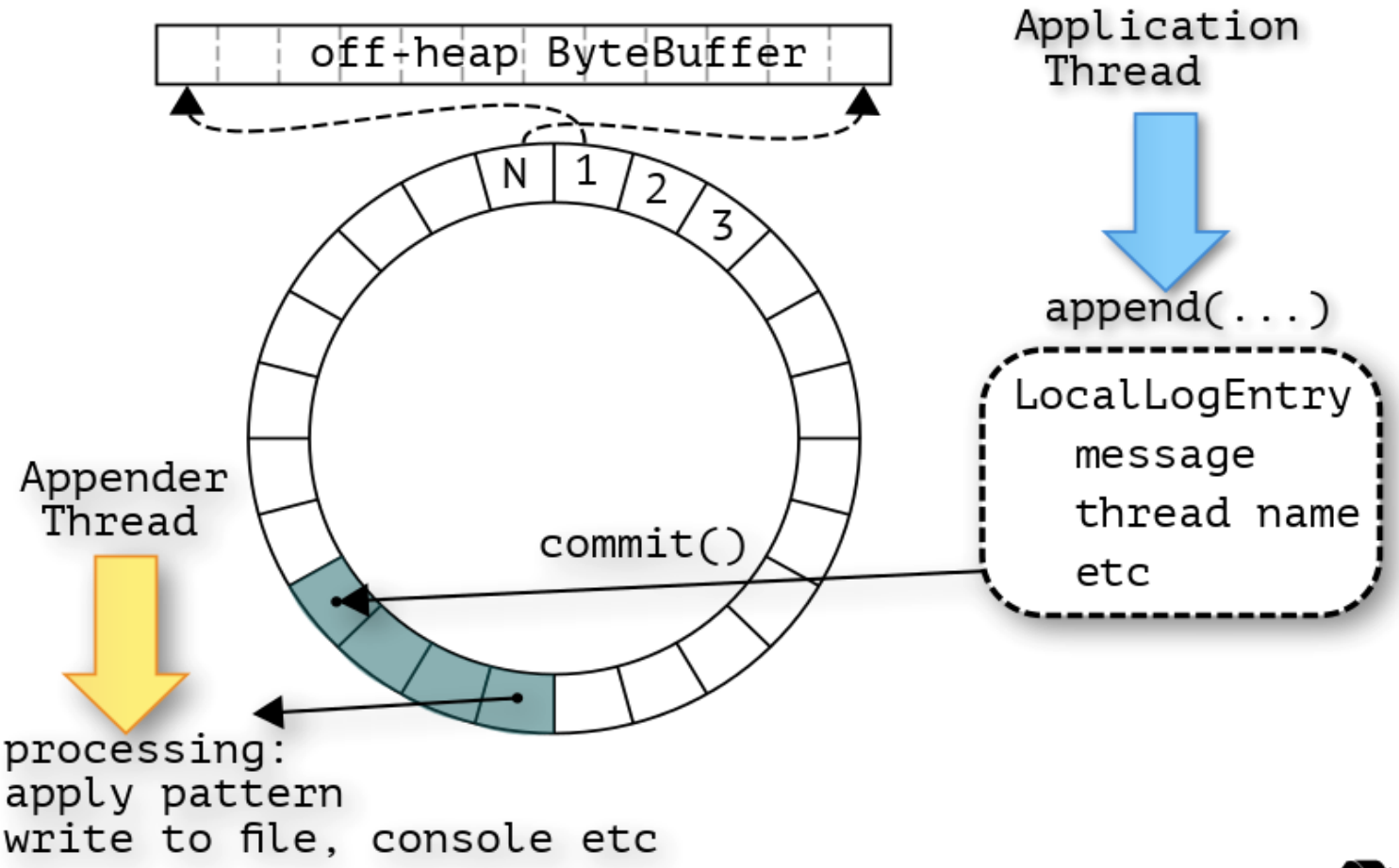
У нас НЕТ:

- следов мусора в runtime
  - *кроме* начальной фазы
- неограниченой длины сообщений
- нельзя динамически менять appenderы
- нельзя следить изменения:
  - имени потока выполнения
  - часовой зоны и локали
- палитры appenderов, н-р: jms, email и т.п.

Пример

```
private static final GFLog log =
 GFLogFactory.getLog(MyClass.class);


// ... somewhere in the method body:

log.info("%s value: %s").
   with( name ).with( value );
```

gflogger design



gflogger

B E N C H M A R K

- Similar way loggers configuration: async mode, buffered i/o
- Log message pattern
  `%d{HH:mm:ss,SSS zzz} %p %m [%c{2}] [%t]%n`
  example:
  `18:58:54,820 GMT INFO test 8069 [pkg.MyClass] [thread-1]`
- Buffer size: 1024 msgs
- Appenders: one file appender
- Metrics
  - throughput
  - total stop the world pauses
- Versions:
  - gflogger 0.1.0-alpha5, disruptor 2.10.3
  - log4j 1.2.15, logback 0.9.29
- Intel Xeon X5687 @ 3.6GHz : 4 ph / 16 virt cores, 8 threads/core
- SLES, java 6u34 64bit

а в это время в соседней аудитории...

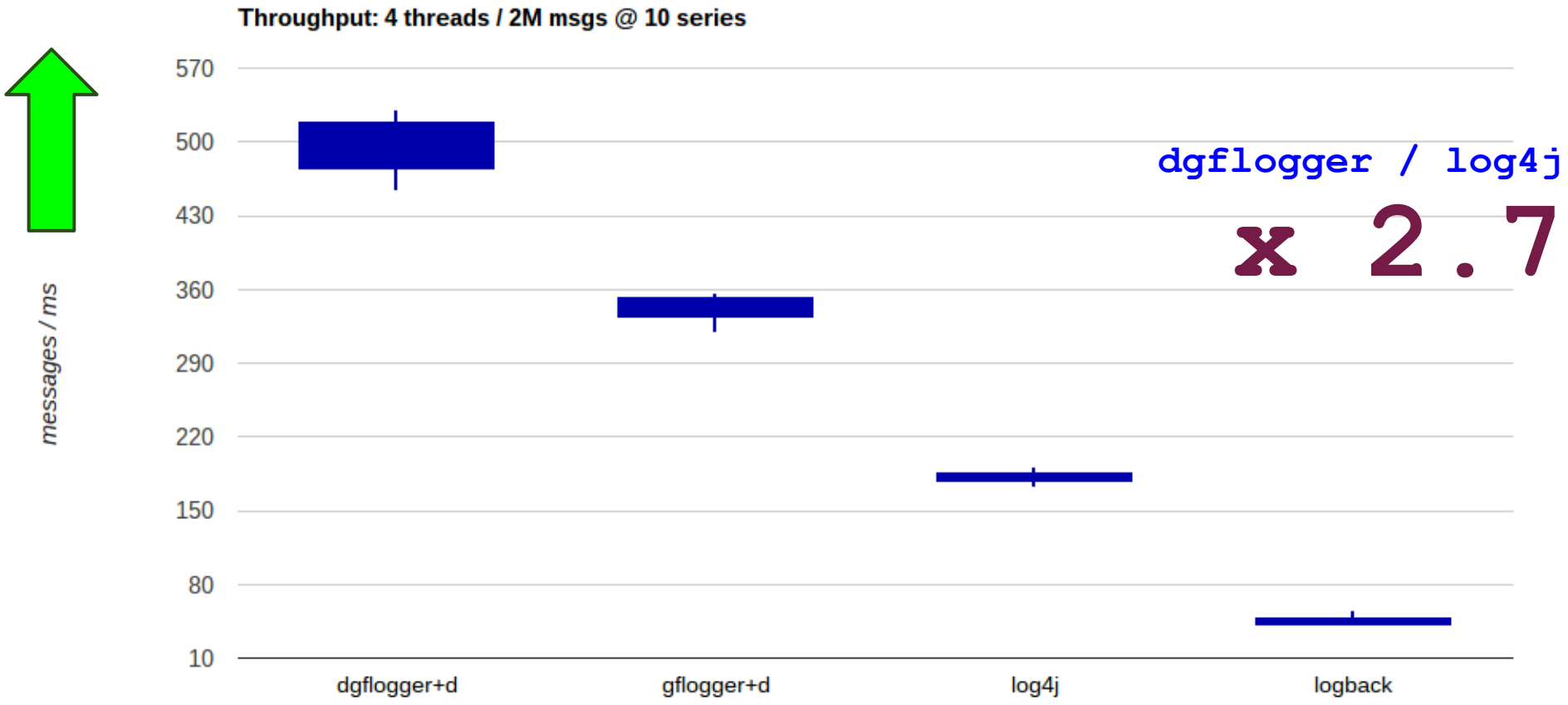
B E N C H M A R K

- high contention:
  - 5*N bogus операций + 5*N сообщений на разогрев
  - **N** { сообщений `"value:" + longValue`}
  - 5*N bogus операций

- low contention:
  - 5*N f(threadNo, i), где f() ~= math.log
  - разогрев 5*N { сообщений + f(threadNo, i) }
  - **N** { сообщений `"value:" + longValue` *и* f(threadNo, i) **}**
  - 5*N f(threadNo, i)

logback vs log4j vs gflogger

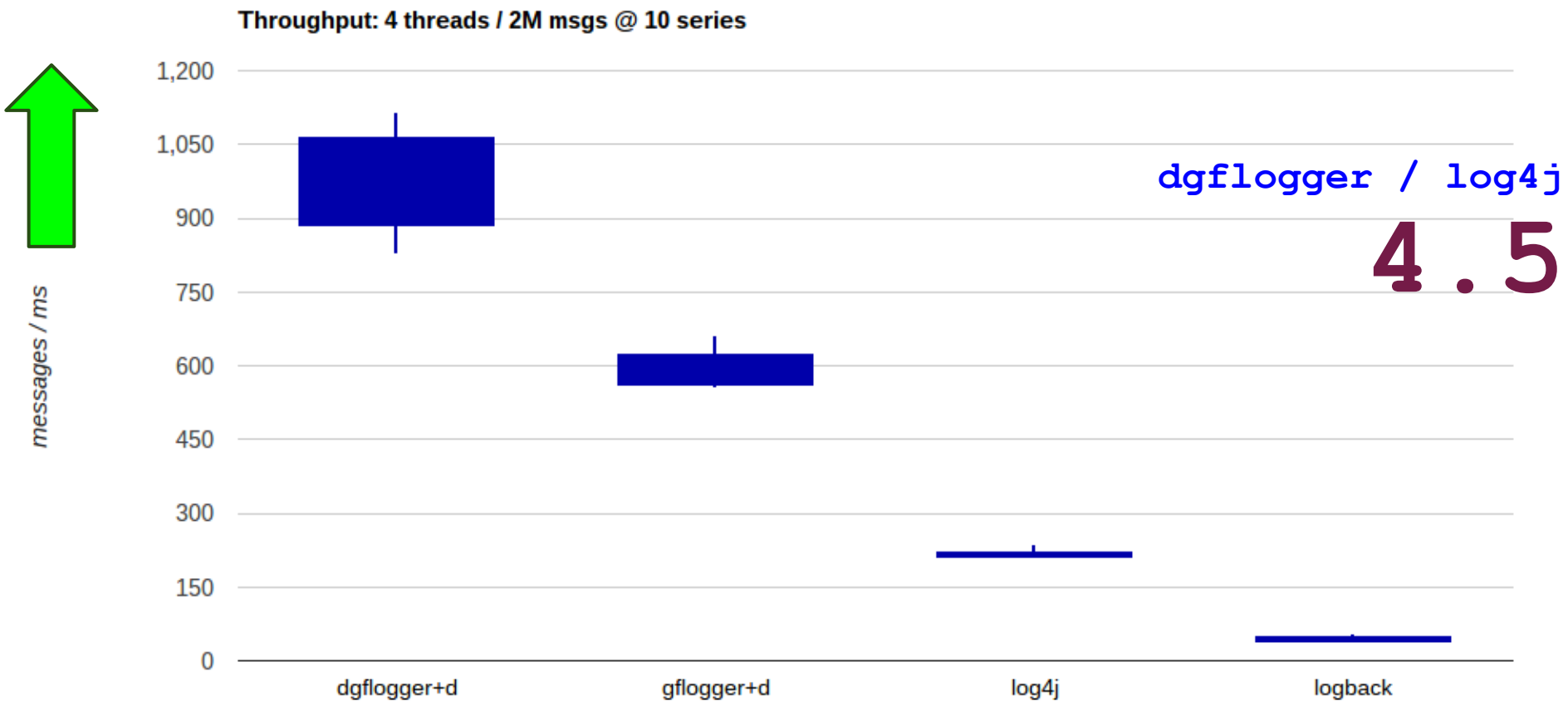
high contention



в среднем: 2.0 / 2.8 / 5.6 / 21 µs / сообщение

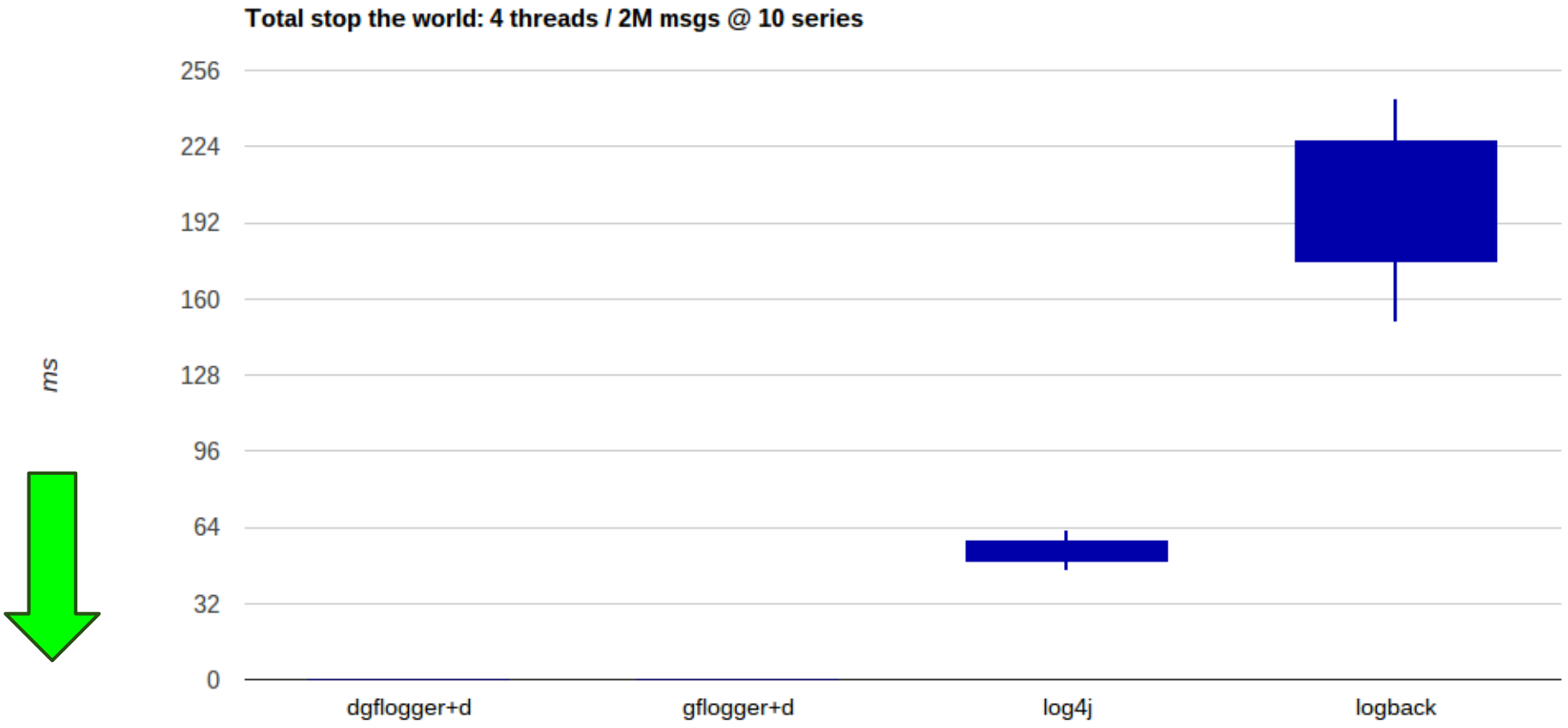
logback vs log4j vs gflogger

low contention



в среднем: 1.0 , 1.7 , 4.6 , 21 µs / сообщение

Суммарое время stop-the-world паузы



в среднем: 0% , 0% , 0.5% , 0.5%

Конкуренты сейчас

- Menta log
  - api напоминает sl4j
    - varargs и autoboxing
  - *garbage free* (внутри), кольцевой буфер

- jShadow
  - event-driven logging
  - SHM
  - lock free, garbage free
  - fast ~25 ns / msg
  - Магия Unsafe

Что дальше ?

- mapped file
  - прототип: ~60 нс/сообщение
- бинарное логирование, бинарные grep'n'sed
- редизайн:
  - безразмерное сообщение
  - без вспомогательной нитки

ИЩЕМ ТАЛАНТЫ

http://**www.dbdevcenter.ru**/

Контакты

эл.почта:

vladimir . dolzhenko @ gmail . com

блог:

dolzhenko . blogspot . com

Вопросы / Ответы