# Средства распараллеливания в Java 1.7 (jsr166y...)

Михаил Пономаренко,
Tech Lead компании Sigma Ukraine

SIGMA

# немного истории

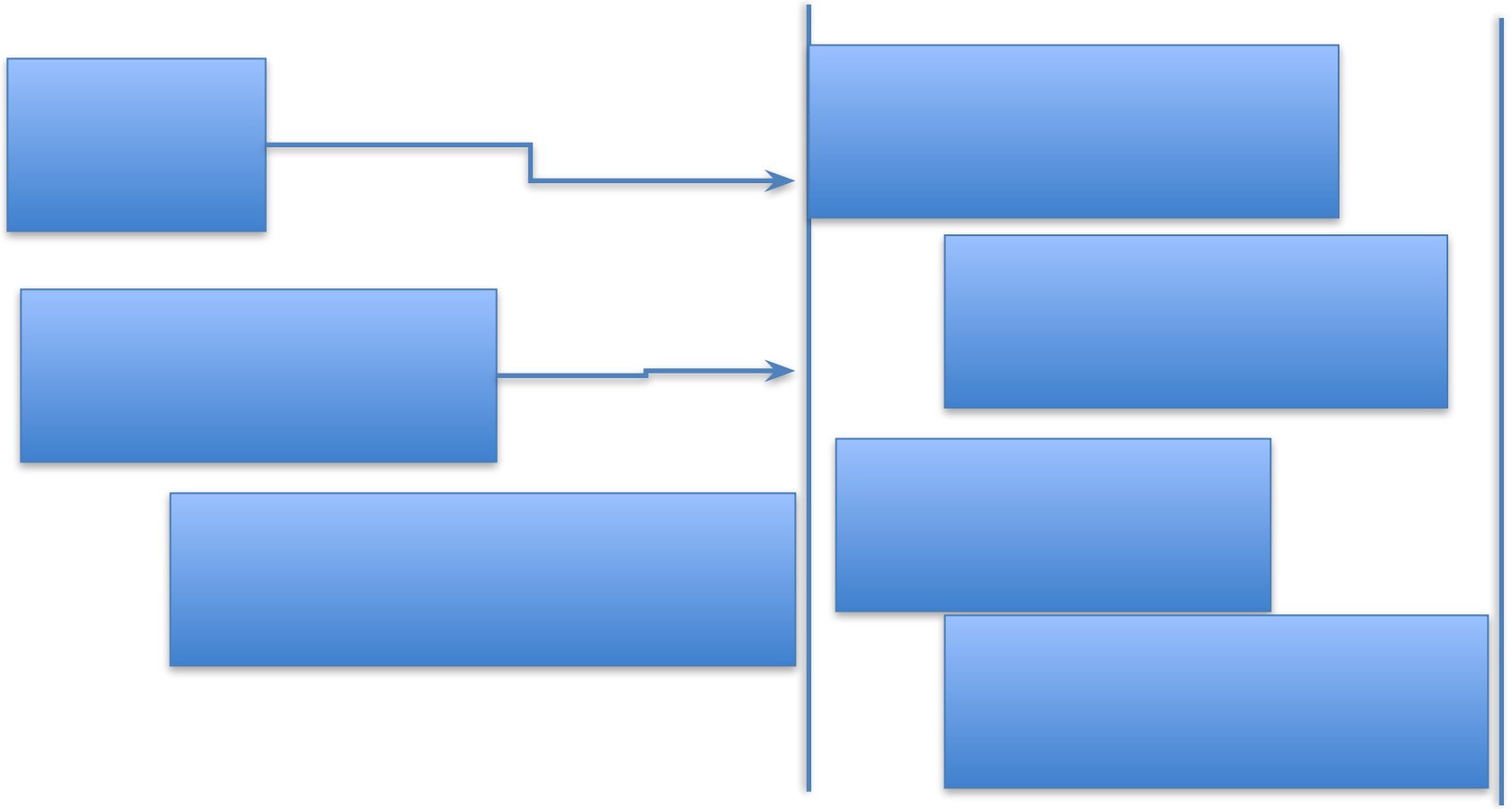
- до java 1.5 были
  - wait
  - notify
  - synchonized
- в 1.5 - jsr166
  - java.util.concurrent
  - Future, ThreadExecutor
  - ConcurrentMap - асинхронные итераторы

SIGMA

# немного истории

- в 1.6 - jsr166x
  - BlockingDeque
  - ConcurrentNavigableMap, NavigableMap
- в 1.7 - jsr166y
  - ForkJoinPool
  - Phaser
  - *ParallelArray (jsr166y extra)*

# java.util.concurrent.Phaser



SIGMA

# java.util.concurrent.Phaser

```
void startTasks(List<Runnable> tasks, final int iterations) {
  final Phaser phaser = new Phaser() {
    protected boolean onAdvance(int phase, int registeredParties) {
      return phase >= iterations || registeredParties == 0;    }
};

  phaser.register();
  for (final Runnable task : tasks) {
    phaser.register();
    new Thread() {    public void run() {
        do {
          task.run();
          phaser.arriveAndAwaitAdvance();
        } while (!phaser.isTerminated());
      }  }.start();
  }
  phaser.arriveAndDeregister(); // deregister self, don't wait
 }
```

SIGMA

# Fork Join – рекурсивная декомпозиция

- если задача маленькая - посчитать
- если большая разбить и посчитать рекурсивно

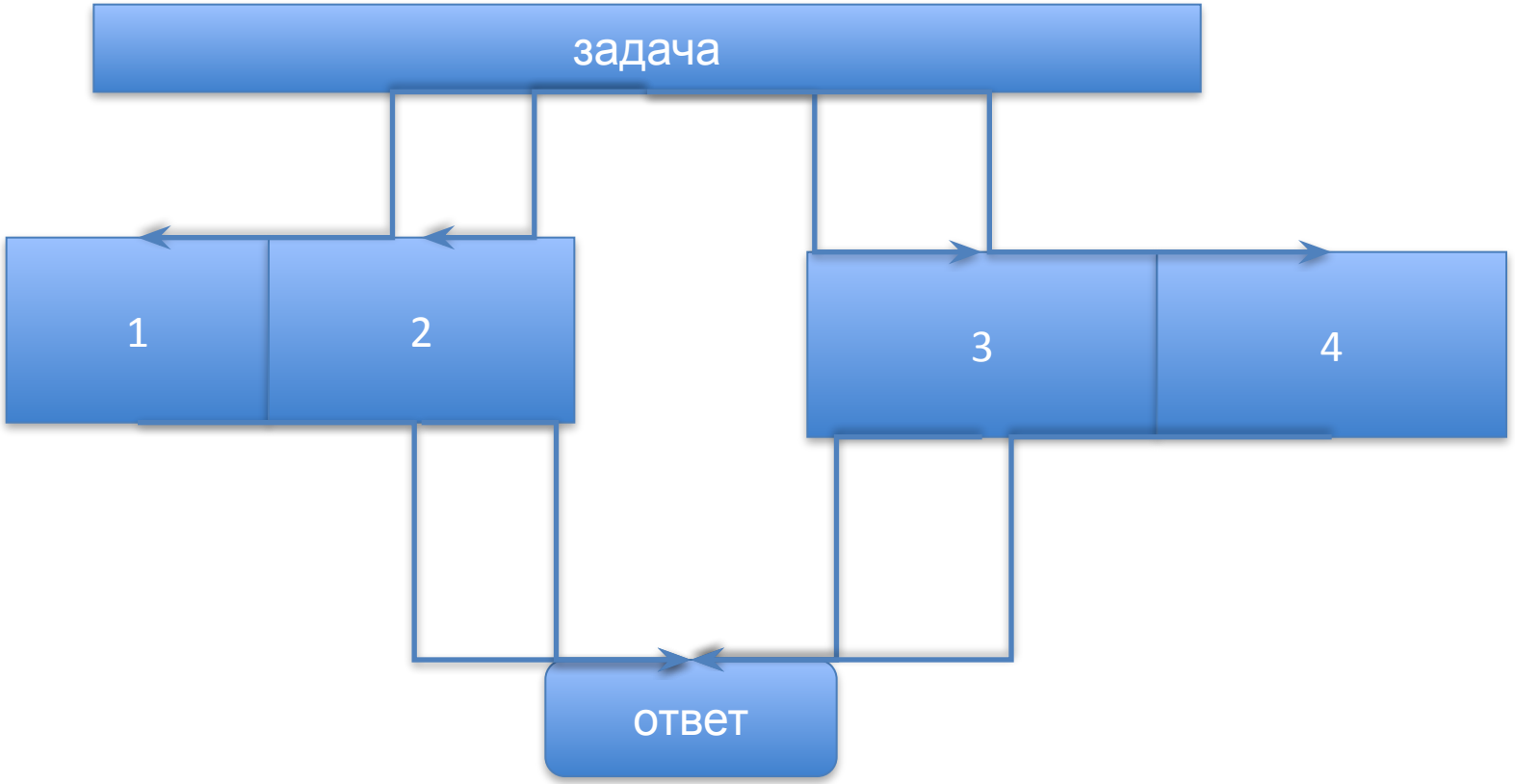
SIGMA

# Fork Join – рекурсивная декомпозиция

- JDK7 дает возможность дробить мелко
- Минимум взаимодействия

SIGMA

# ForkJoin




SIGMA

# Детали реализации

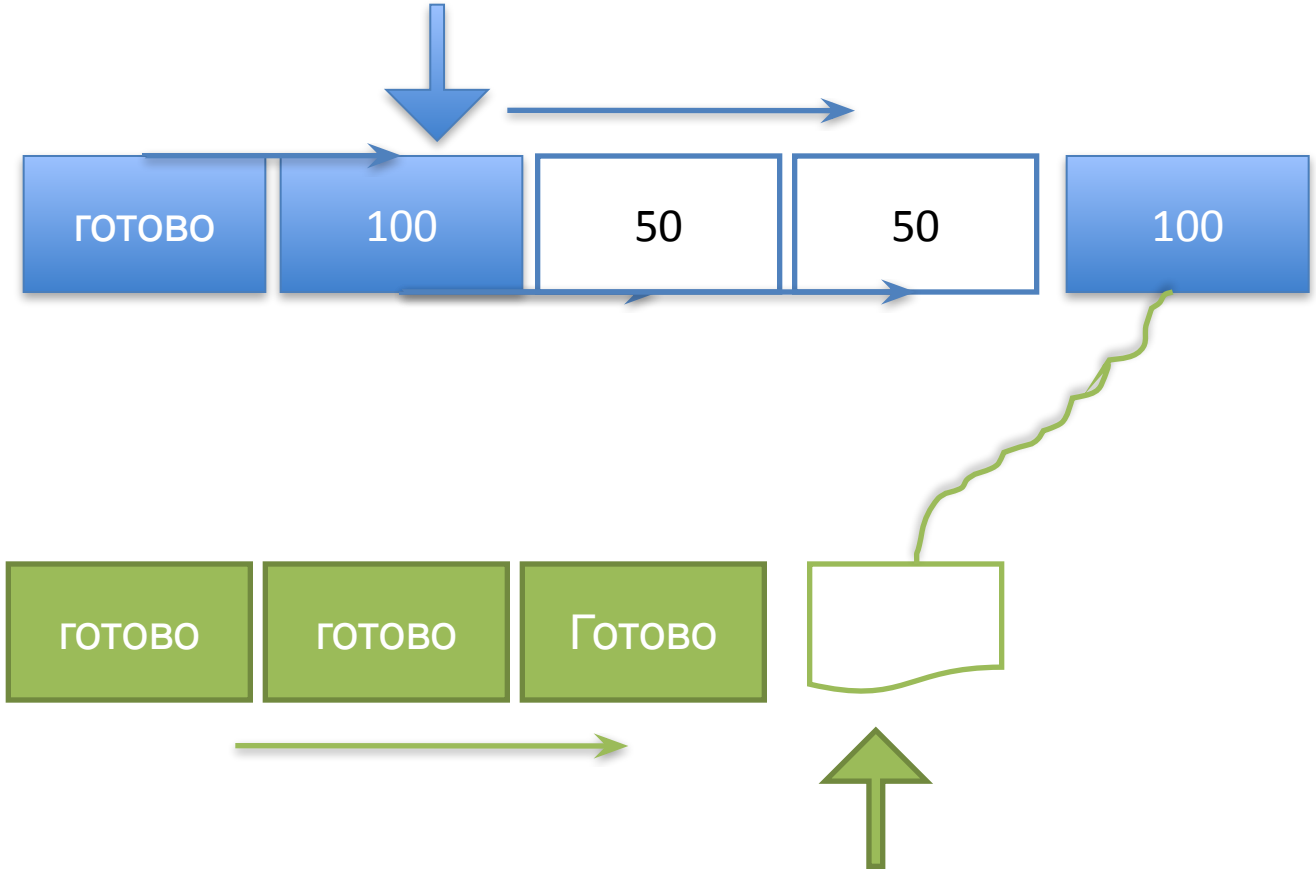
- дополнительной сихронизации не требуется
- старые разбиения "больше“

Поэтому:

1. У каждого потока свой дек задачь
2. поток выполнения берет задачу из дека
3. задача добавляет подзадачи в дэк или производит вычисления
4. если задачи кончилисть - "украсть" задачу у другого потока

SIGMA

# Детали реализации




SIGMA

# ForkJoinTask<V>

- protected abstract  boolean exec()
- ForkJoinTask<V>  fork()
  - Не ждет
- public final V join()
  - То же но без исключений
- public static void invokeAll(ForkJoinTask<?>... tasks))
- Invoke = fork(); join();

SIGMA

# RecursiveAction

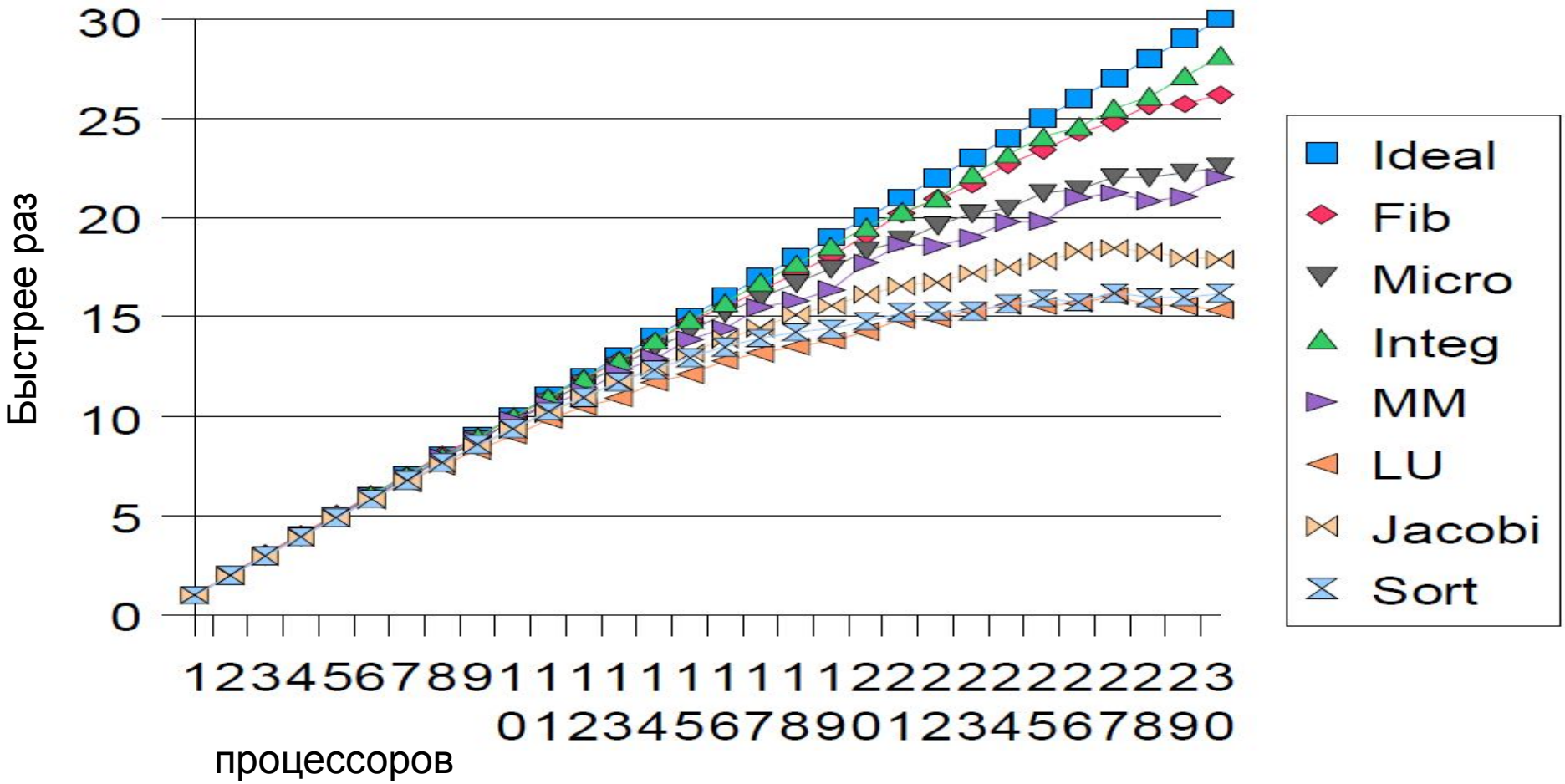
- extends ForkJoinTask<Void>
- **protected void compute()**
  - **Посчитать**
  - **Поделить**
  - **Вызвать invokeAll**
  - **Сделать join**
- Нужно что то сделать, но нет возвращаемого значения
- RecursiveTask – есть возаращаемое значение – его вернет join

SIGMA

# Примеры Doug Lea

| Пример | Описание |
|---|---|
| Fib | Фибоначчи - подсчет числа фибоначчи для 47. Меньше 13 считается влоб |
| Integrate | Интегрирование - суммирование численного интеграла от -47 до 48 для $(2i\text{-}1)x^{(2i\text{-}1)}$ |
| Micro | Поиск лучшего хода в настольной игре. Прогноз на 4 хода вперед |
| Sort | Сортировка слиянием 100 миллионов чисел |
| MM | Умножение двух матриц 2048x2048. double |
| LU | lu-разложеные невыраженной матрицы 4096x4096. double |
| Jacobi | Итерационная релаксация сетки с ограничениями. Усреднение по 100 соседям. матрица 4096x4096 |

# Результаты Doug Lea




SIGMA

# Пример

- 2^28 произвольных чисел double  (2 гб)

```
ForkJoinPool pool = new ForkJoinPool(p);
SinCosHuge task = new
SinCosHuge(randomData);
pool.execute(task);
Double rz = task.join();
```

SIGMA

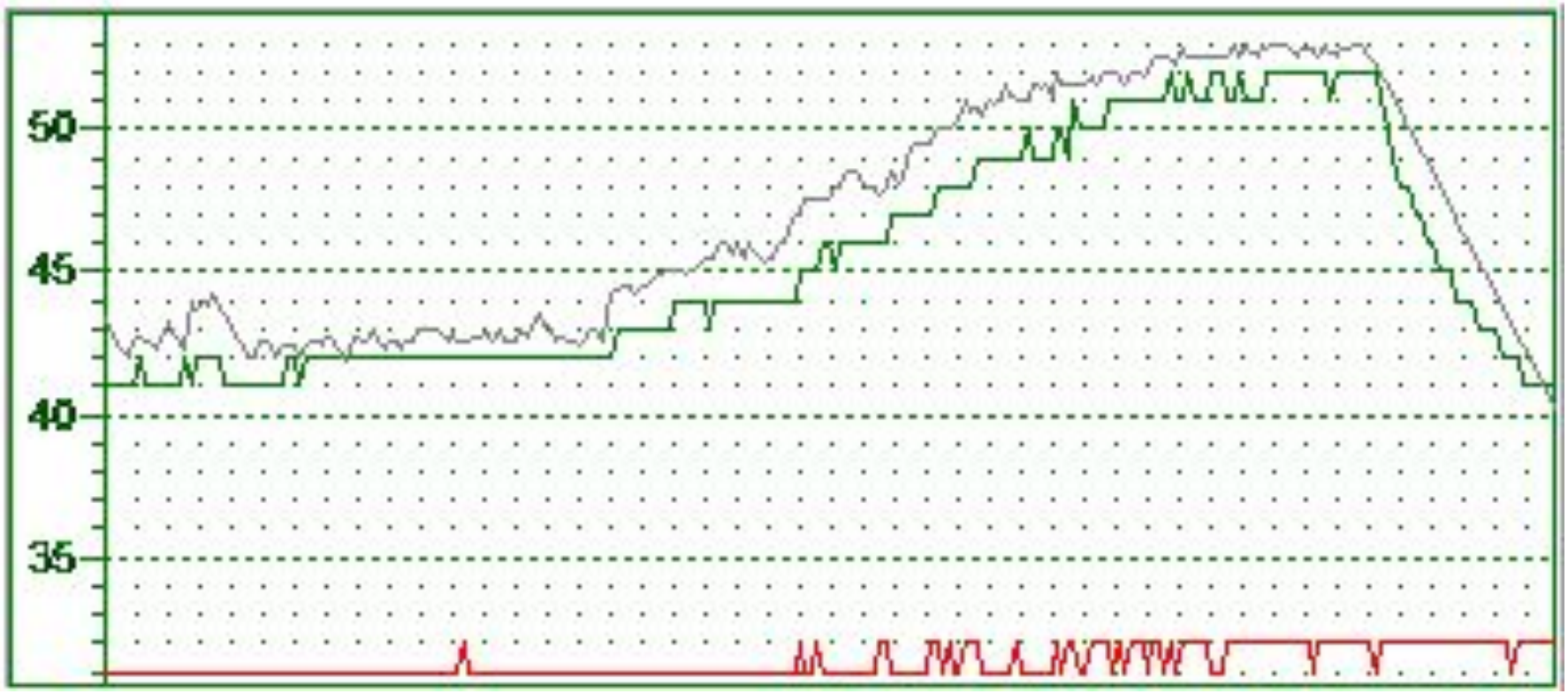
# Пример

```
import java.util.concurrent.RecursiveTask;
public class SinCosHuge extends RecursiveTask<Double> {
…
protected Double compute() {
        if (to - from < threshold) {
            double rz = 0;
            for (int i = from; i < to; i++) {
                rz += Math.sin(data[i]) + Math.atan(data[i]);
            }
            return rz;
        } else {
            int i = (from + to) / 2;
            SinCosHuge right = new SinCosHuge(data, from, i);
            SinCosHuge left = new SinCosHuge(data, i, to);
            invokeAll(left,right);
            return right.join() + left.join();
        }
    }
…
```

SIGMA

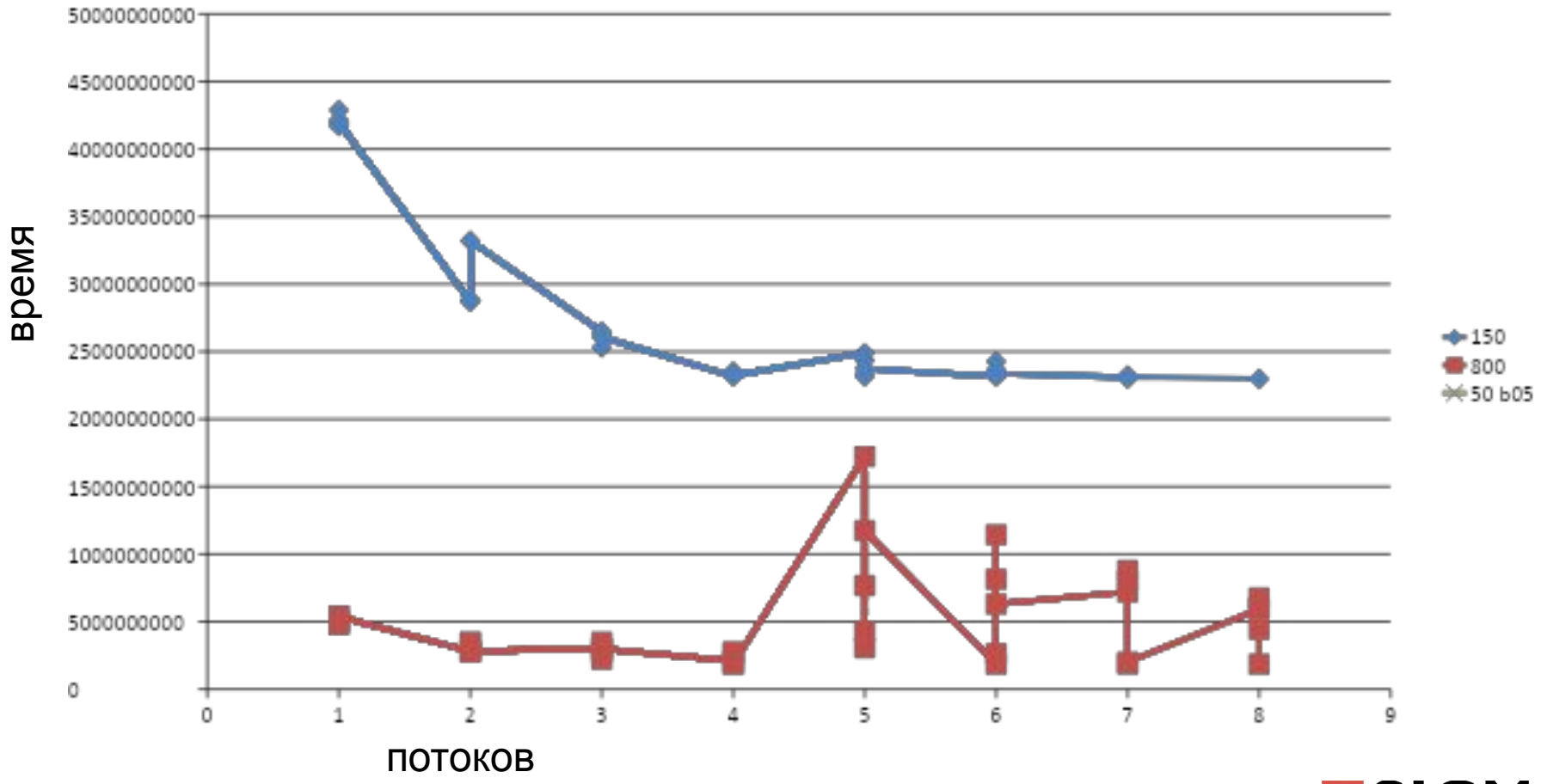
# Мои измерения

Загрузка процессора 100%
температура

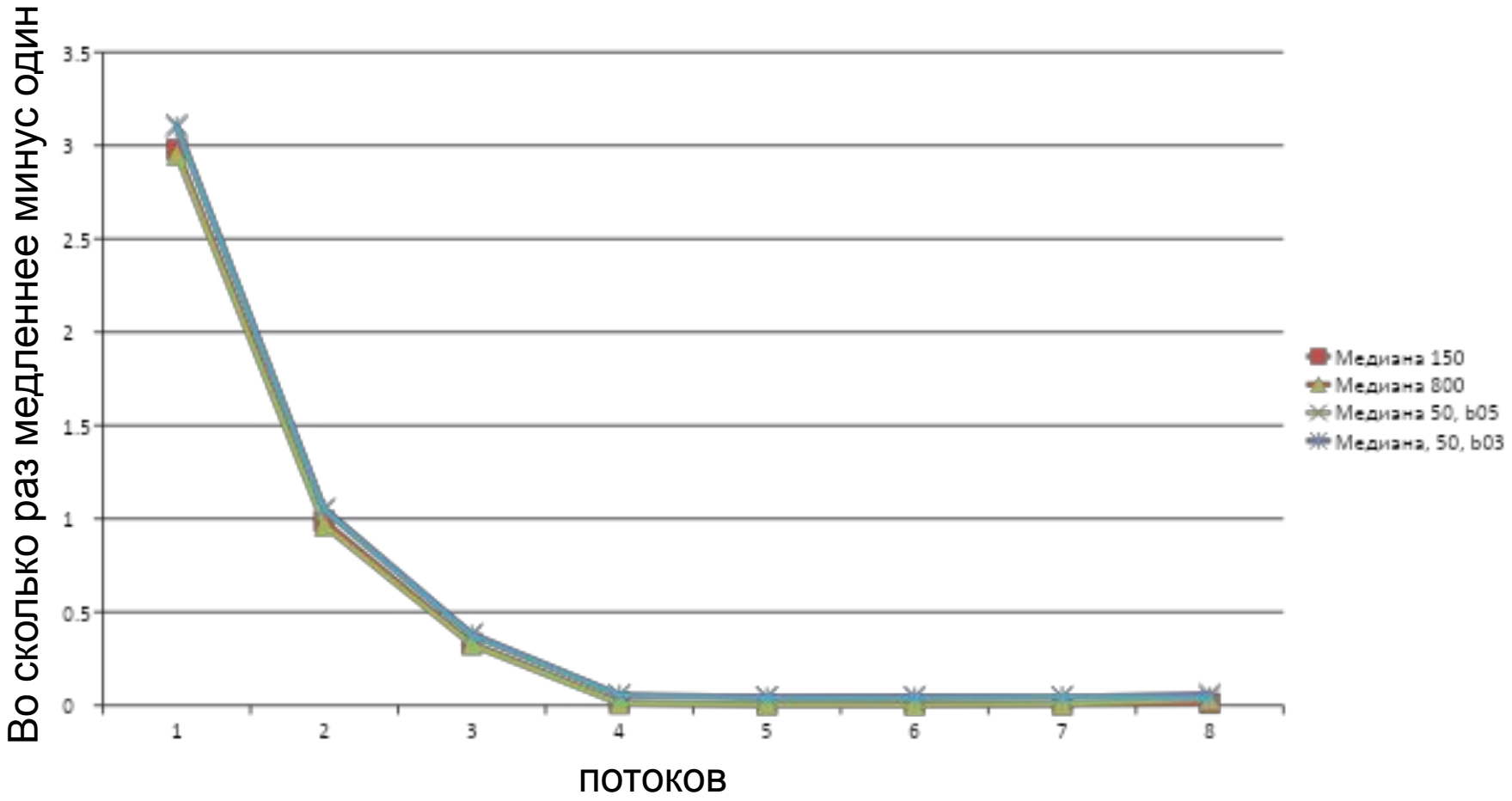



SIGMA

# Мои измерения, время выполнения.

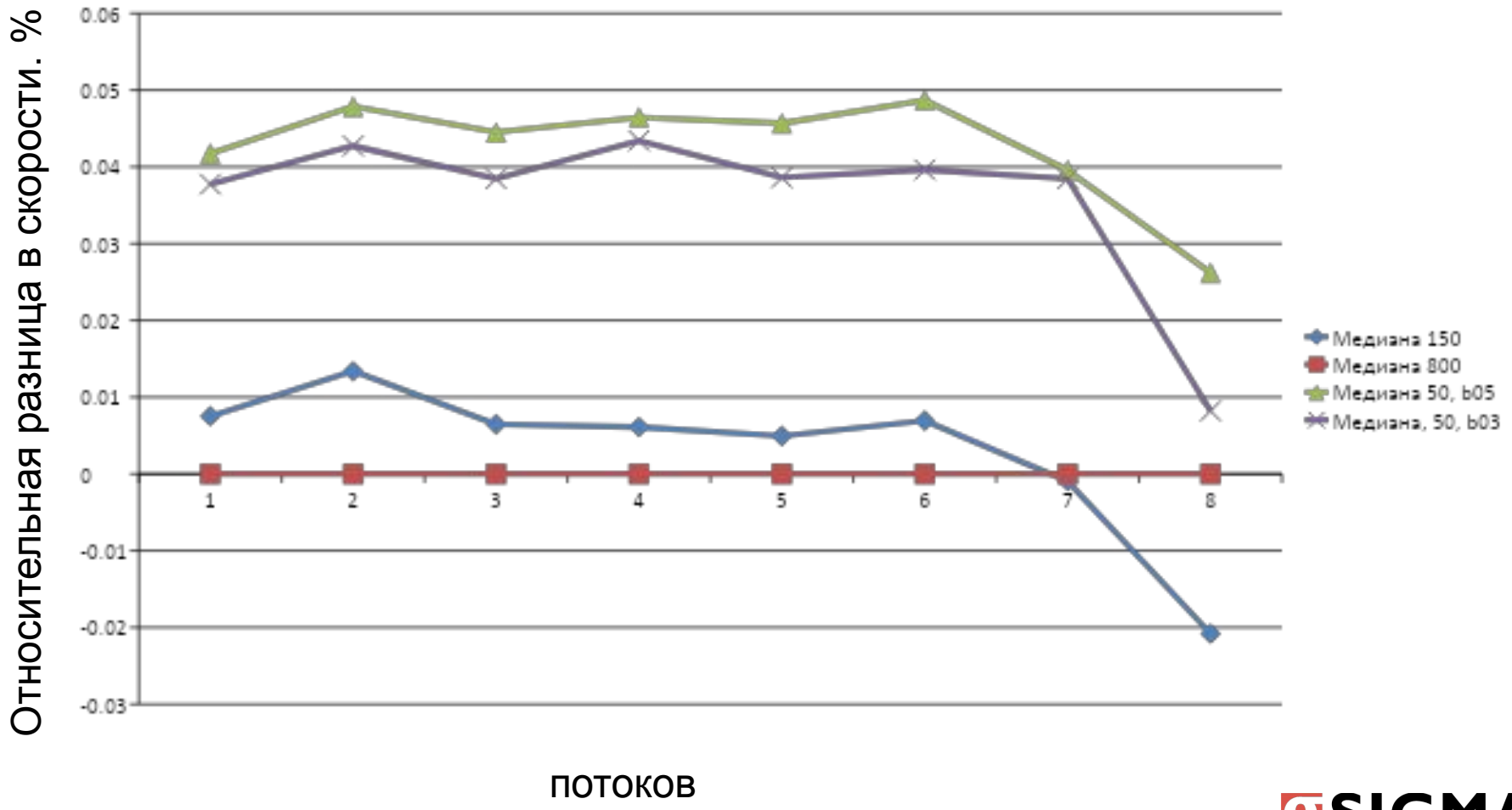



SIGMA


Во сколько раз медленнее минус один
3.5
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7
8
потоков
Медиана 150
Медиана 800
Медиана 50, b05
Медиана, 50, b03

SIGMA

# Мои измерения – относительно самого быстрого




SIGMA

# А если по старому?

- ThreadExecutorPool
- Result
  - add(double)
  - waitDone
  - fork

```
ThreadPoolExecutor tpe = new ThreadPoolExecutor(p, p, 10,SECONDS, workQueue);
Result rz = new Result();
tpe.submit(new SinCosHugePool(randomData, tpe, rz));
rz.waitDone();
```

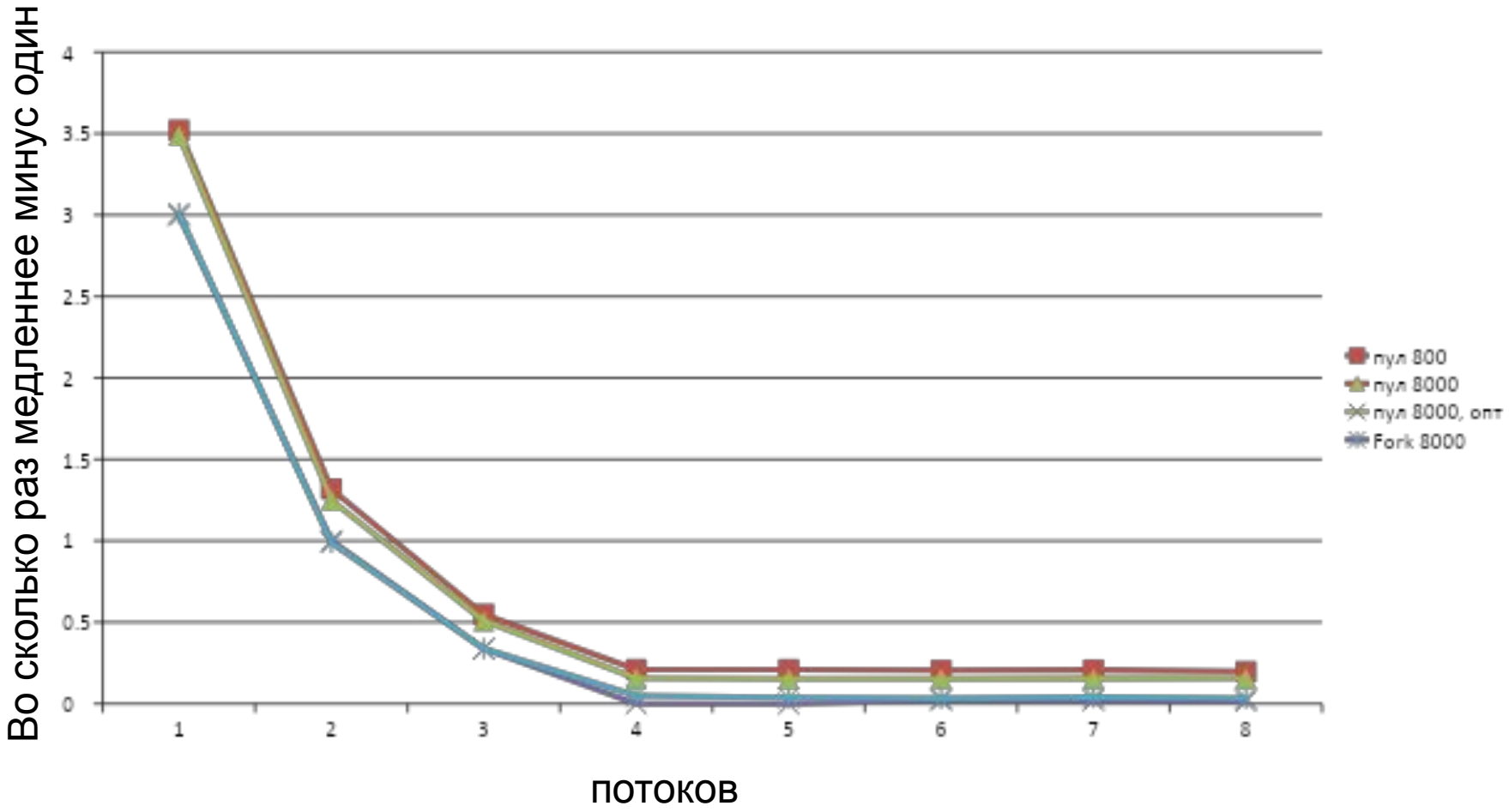
SIGMA

# Runnable.run

```
if (to - from < treshold) {
        double rz = 0;
        for (int i = from; i < to; i++) {
                rz += Math.sin(data[i]) + Math.atan(data[i]);
        }
        result.add(rz);
} else {
        int i = (from + to) / 2;
        SinCosHugePool right = new SinCosHugePool(data, from, i, executor,result);
        SinCosHugePool left = new SinCosHugePool(data, i, to, executor,result);
        result.fork();
        executor.execute(left);
        executor.execute(right);
}
```

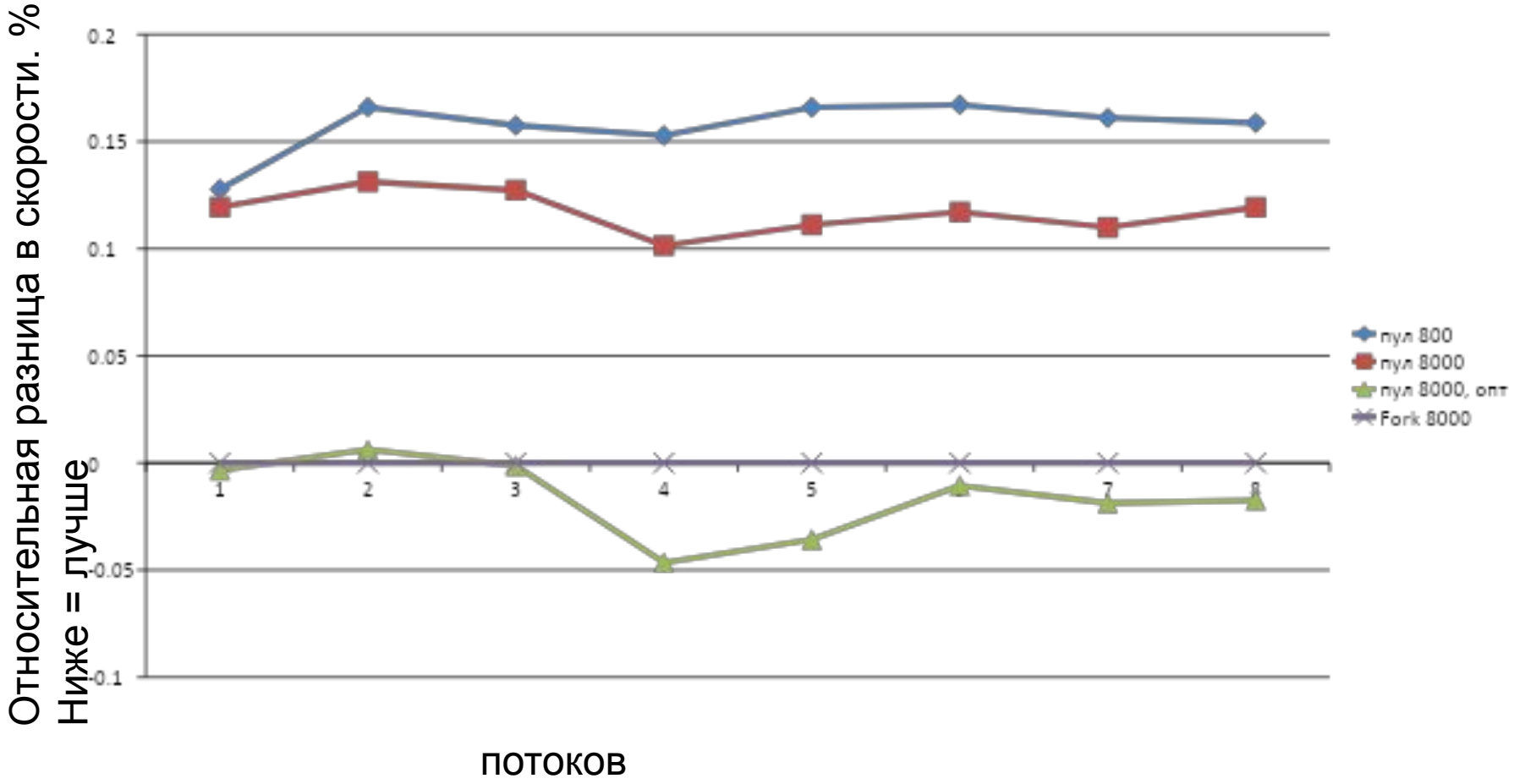
SIGMA

# А если по старому?




SIGMA

# А если по старому




SIGMA

# ParallelArray

- Судя по всему НЕ попадет в JDK 1.7, но исходники доступны
- MapReduce в пределах одной машины

SIGMA

# Пример IBM

```
ParallelArray<Student> students = new ParallelArray<Student>(fjPool, data);
double bestGpa = students.withFilter(isSenior).withMapping(selectGpa).max();

public class Student {
    String name;    int graduationYear;   double gpa;
}

static final Ops.Predicate<Student> isSenior = new Ops.Predicate<Student>() {
    public boolean op(Student s) { return s.graduationYear == Student.THIS_YEAR;   }
};

static final Ops.ObjectToDouble<Student> selectGpa = new Ops.
ObjectToDouble<Student>() {
    public double op(Student student) {  return student.gpa;   }
};
```

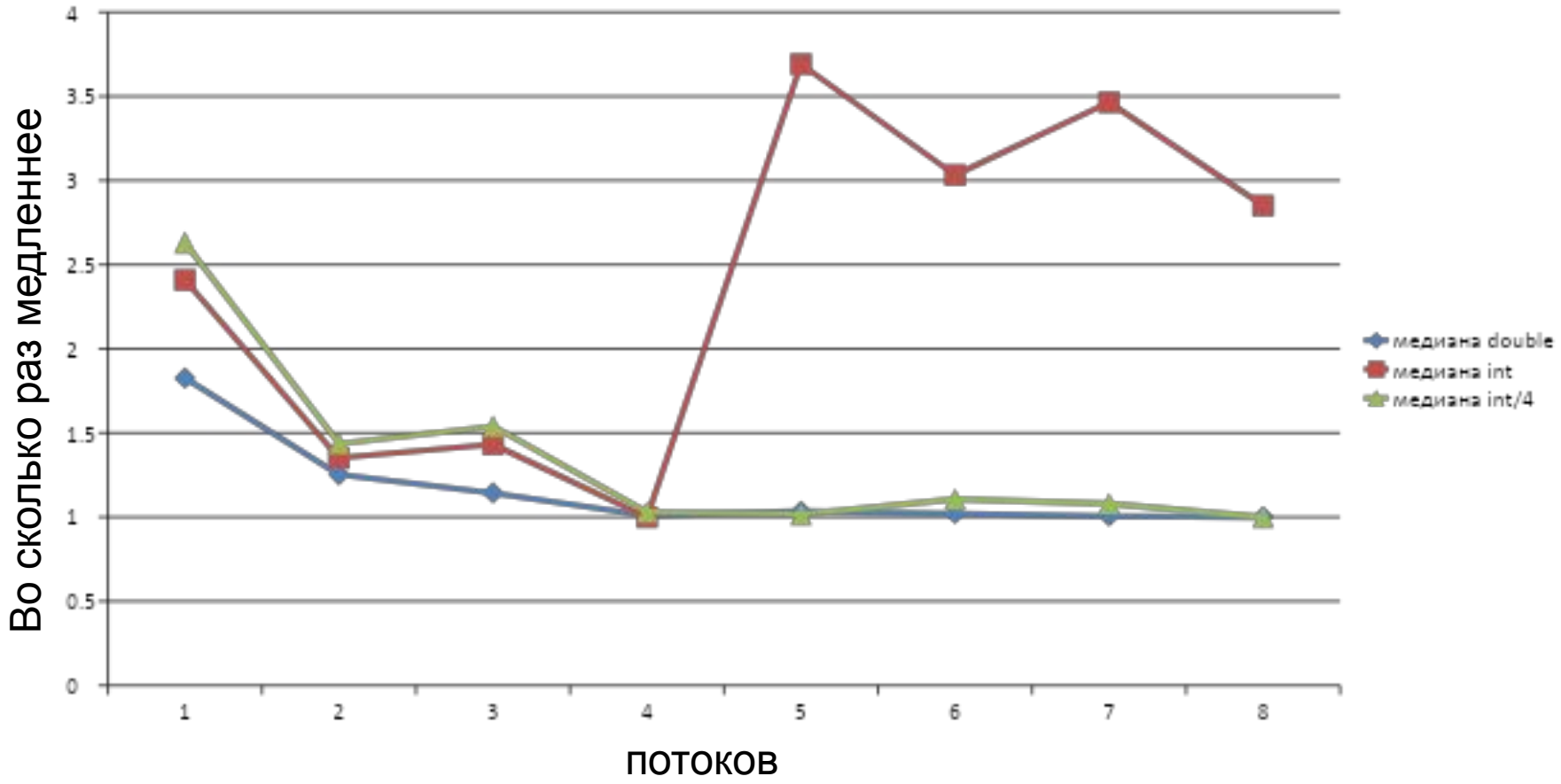
SIGMA

# Мой пример

- new long[16384 * 16384/8]  - 1 гб рабочей памяти,
- double[16384 * 16384/2] – 6гб рабочей

```
import jsr166y.ForkJoinPool;
import extra166y.ParallelLongArray;
…..
long[] randomData = new long[16384 * 16384/8];
ForkJoinPool pool = new ForkJoinPool(p);
ParallelLongArray arr = ParallelLongArray.createUsingHandoff(randomData, pool);
int uniqueCount = arr.allUniqueElements().size();
ParallelLongArray.SummaryStatistics summary = arr.summary();
```

# Результаты




SIGMA

# Откуда начать

- Concurrency JSR-166 Interest Site: http://g.oswego.edu/dl/concurrency-interest/

SIGMA