podlodka

Java Crew

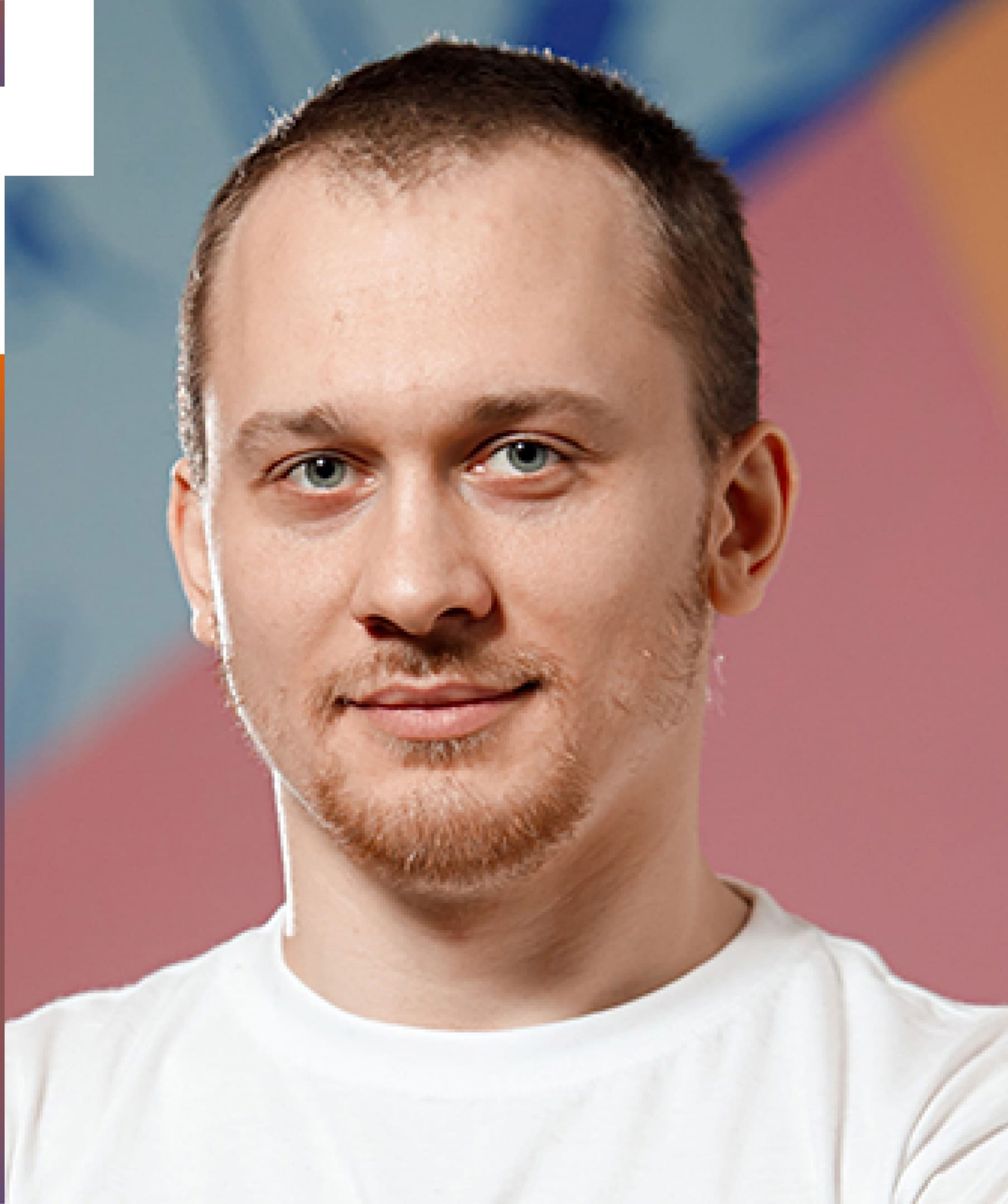
26 – 30 мая

Java и производительность

Григорий Кошелев

Контур

# Воркшоп
# JMH: вводный курс по микробенчмаркам

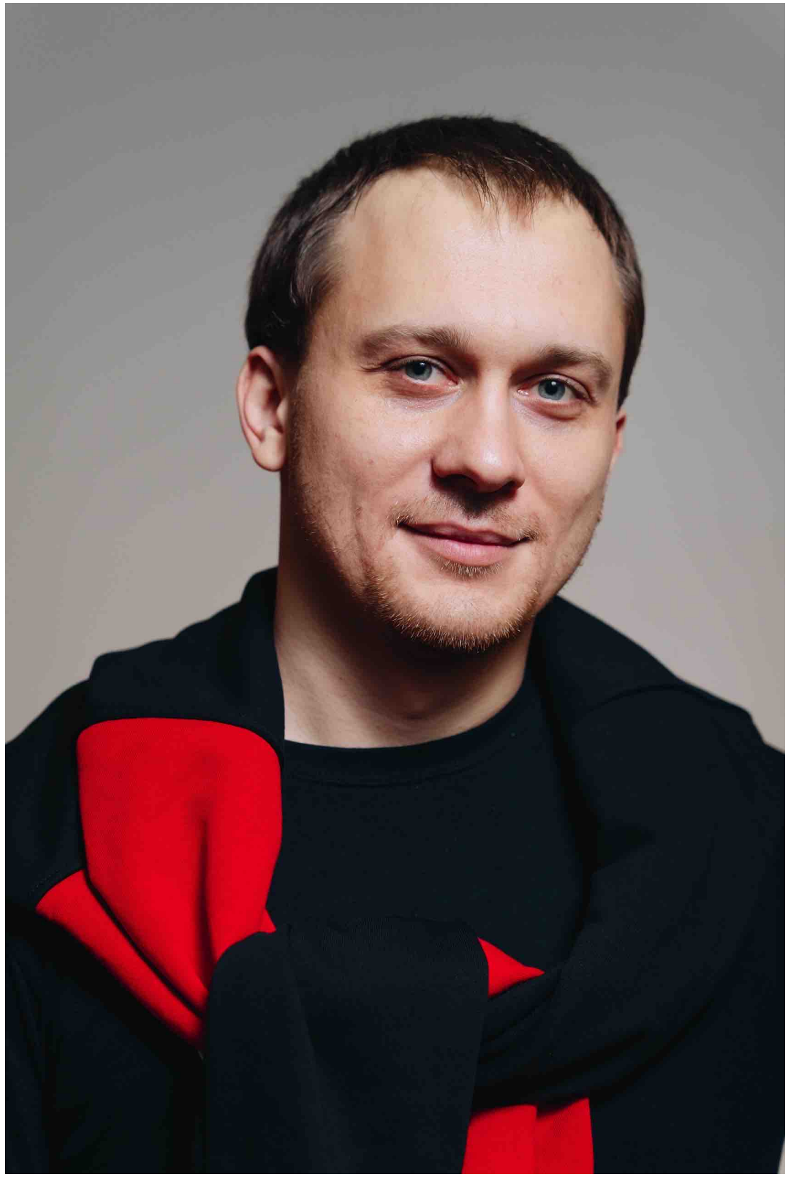


# JMH
## вводный курс по микробенчмаркам

Григорий Кошелев

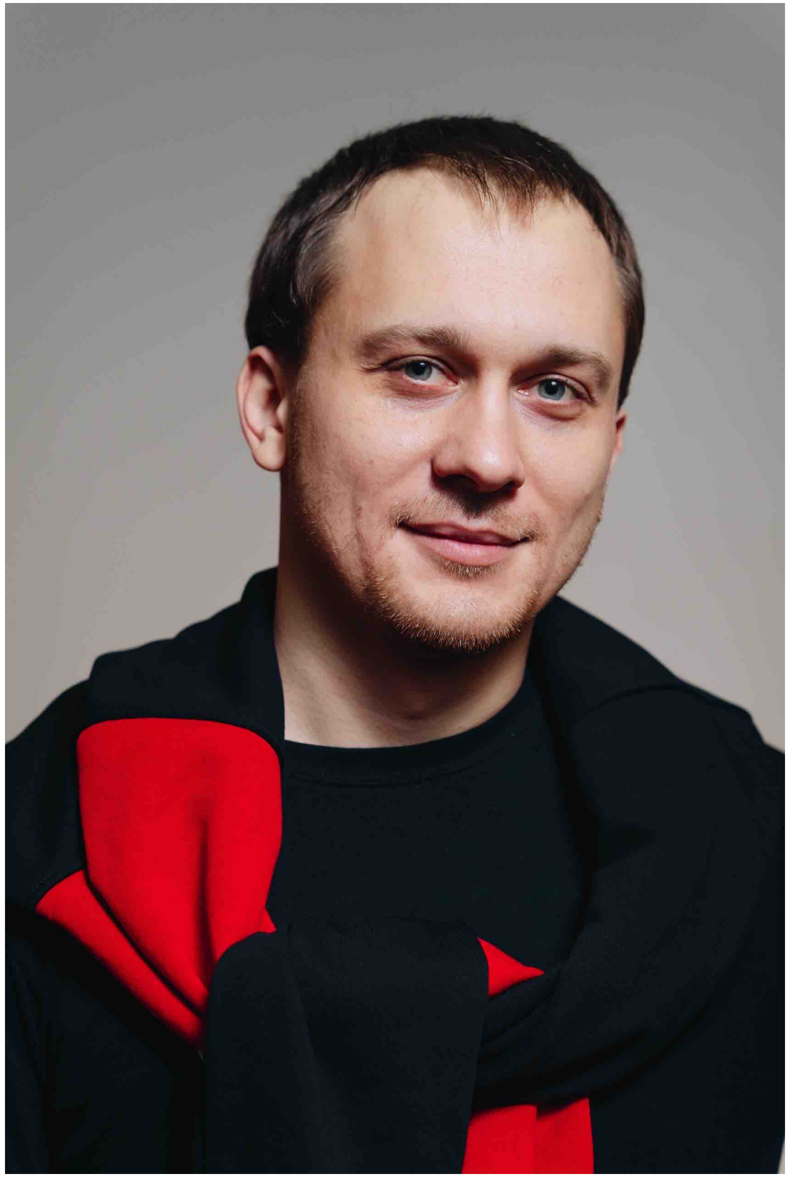
Podlodka Java Crew 2025

# Григорий Кошелев



# Григорий Кошелев

— Веду Telegram-канал про технологии (~10 постов с тегом #performance)

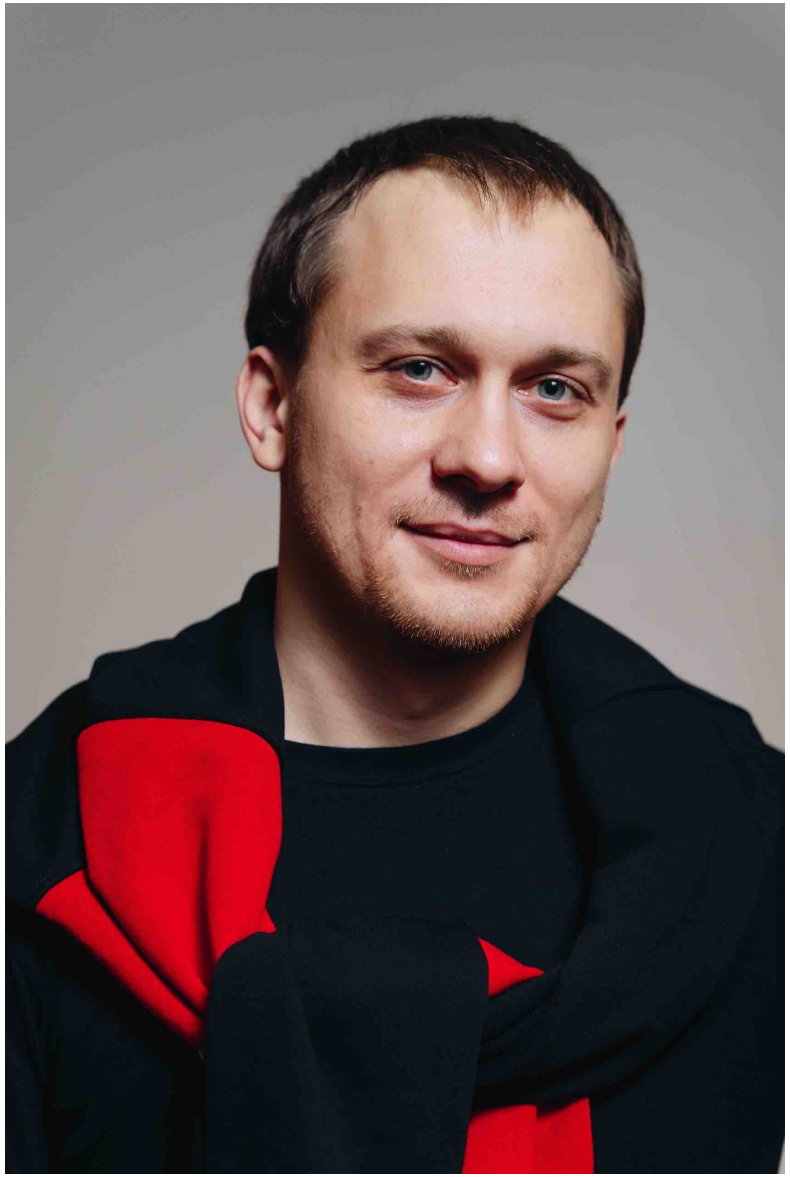


https://t.me/chnl_GregoryKoshelev

# Григорий Кошелев

— Веду Telegram-канал про технологии (~10 постов с тегом #performance)

— Часто рассказываю про Apache Kafka (10+ докладов и выступлений)



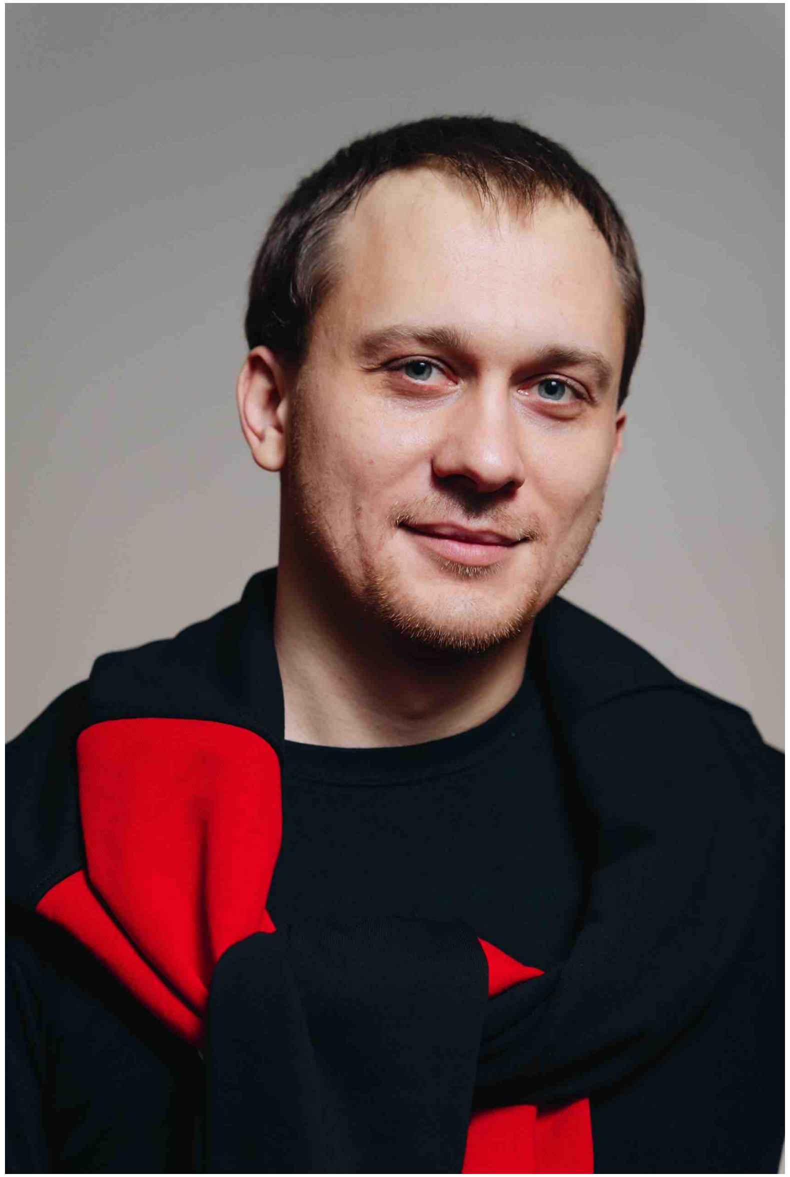
https://t.me/chnl_GregoryKoshelev

# Григорий Кошелев

— Веду Telegram-канал про технологии (~10 постов с тегом #performance)

— Часто рассказываю про Apache Kafka (10+ докладов и выступлений)

— Иногда рассказываю о performance



https://t.me/chnl_GregoryKoshelev

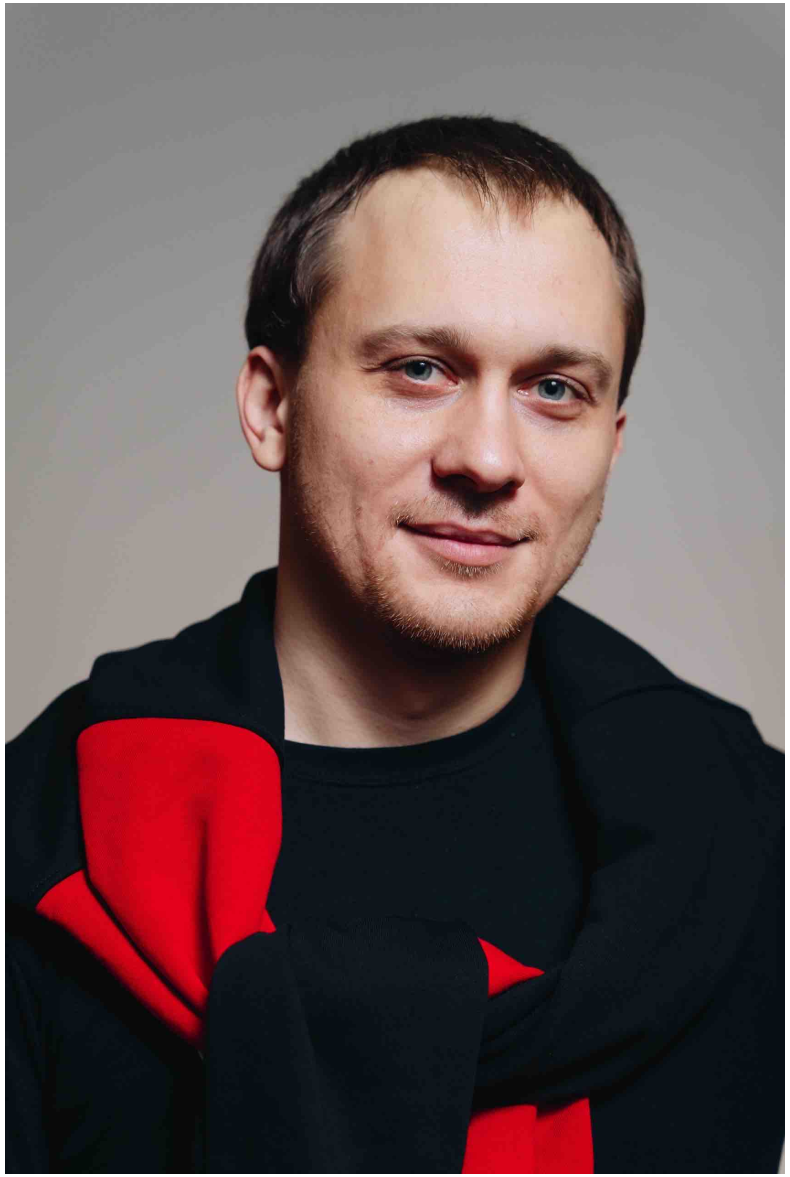
# Григорий Кошелев

— Веду Telegram-канал про технологии (~10 постов с тегом #performance)

— Часто рассказываю про Apache Kafka (10+ докладов и выступлений)

— Иногда рассказываю о performance

* Как (не) надо проводить нагрузочное тестирование (Heisenbug 2023)



https://www.youtube.com/watch?v=MreHXvi7AIQ

# Григорий Кошелев

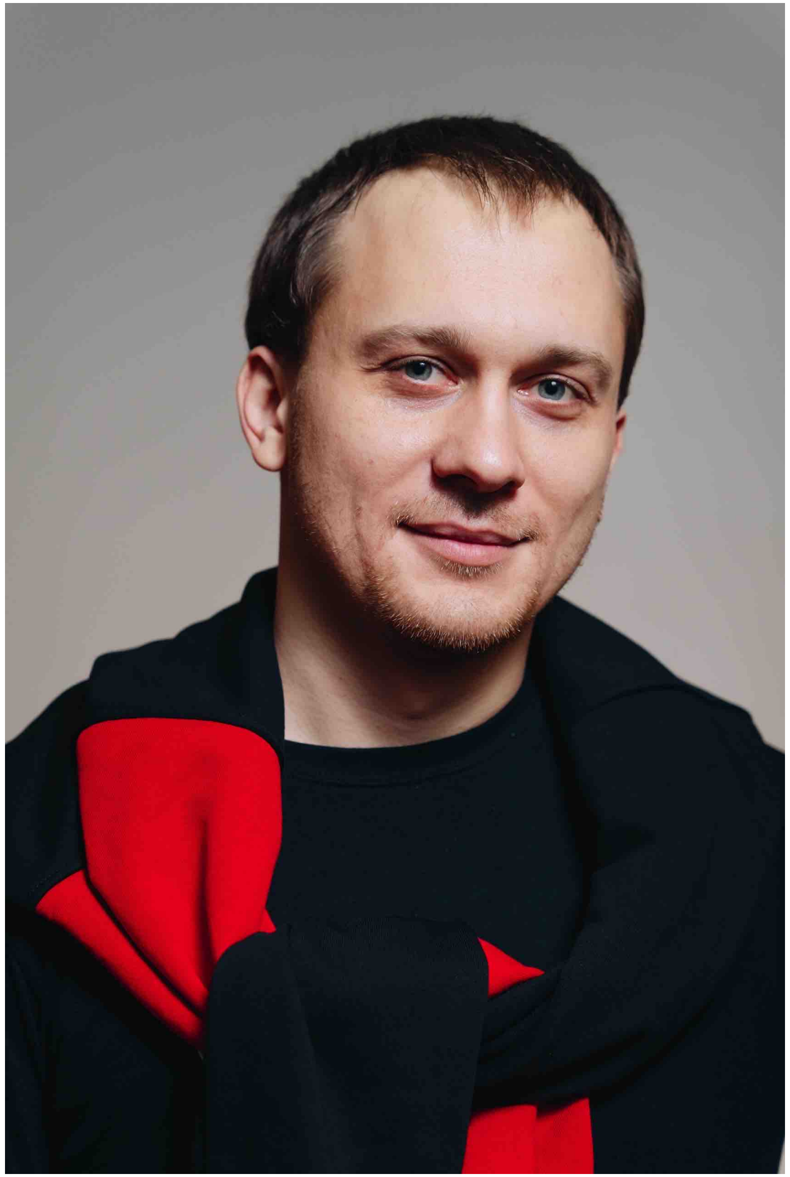
— Веду Telegram-канал про технологии (~10 постов с тегом #performance)

— Часто рассказываю про Apache Kafka (10+ докладов и выступлений)

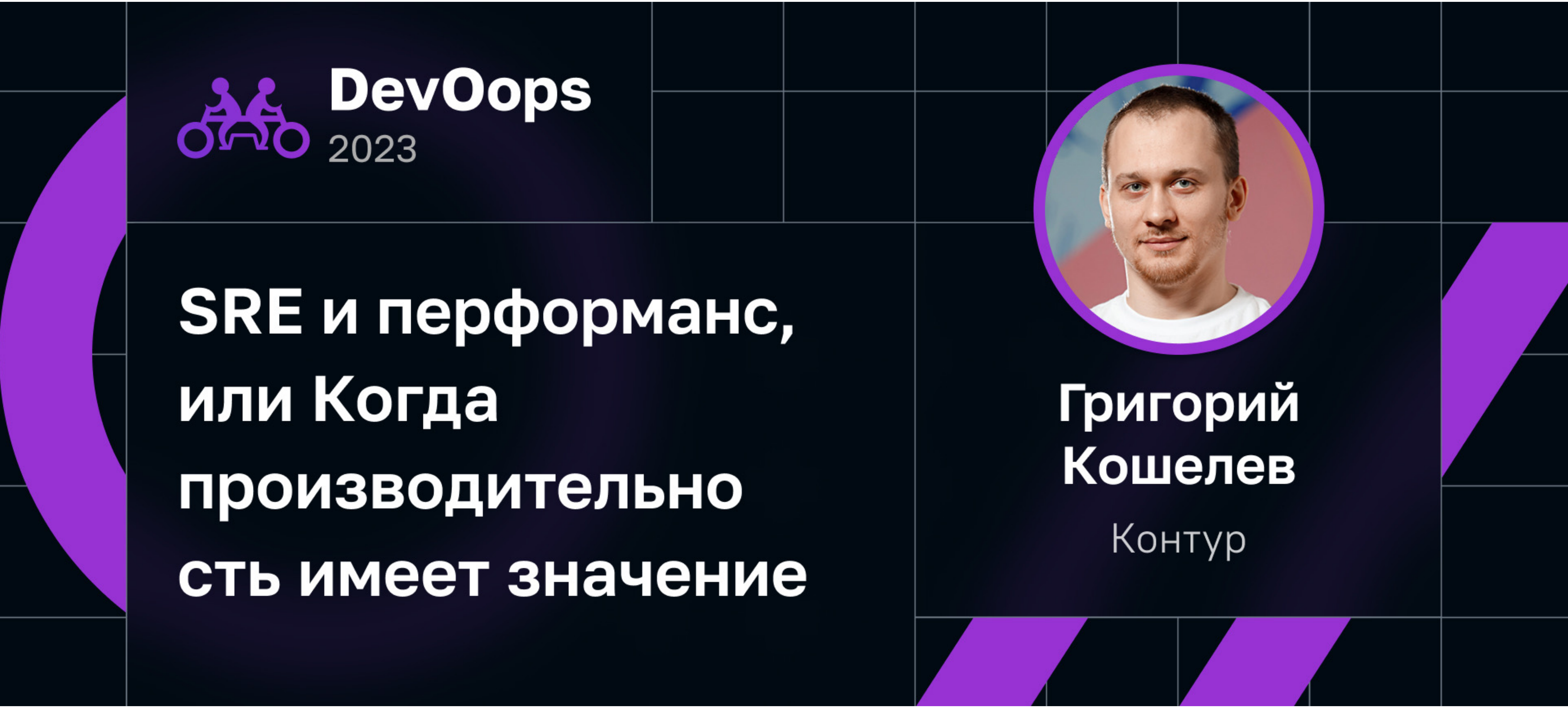
— Иногда рассказываю о performance

* Как (не) надо проводить нагрузочное тестирование (Heisenbug 2023)

* SRE и перформанс, или Когда производительность имеет значение (DevOops 2023)



https://www.youtube.com/watch?v=ukXD3t_ab0I



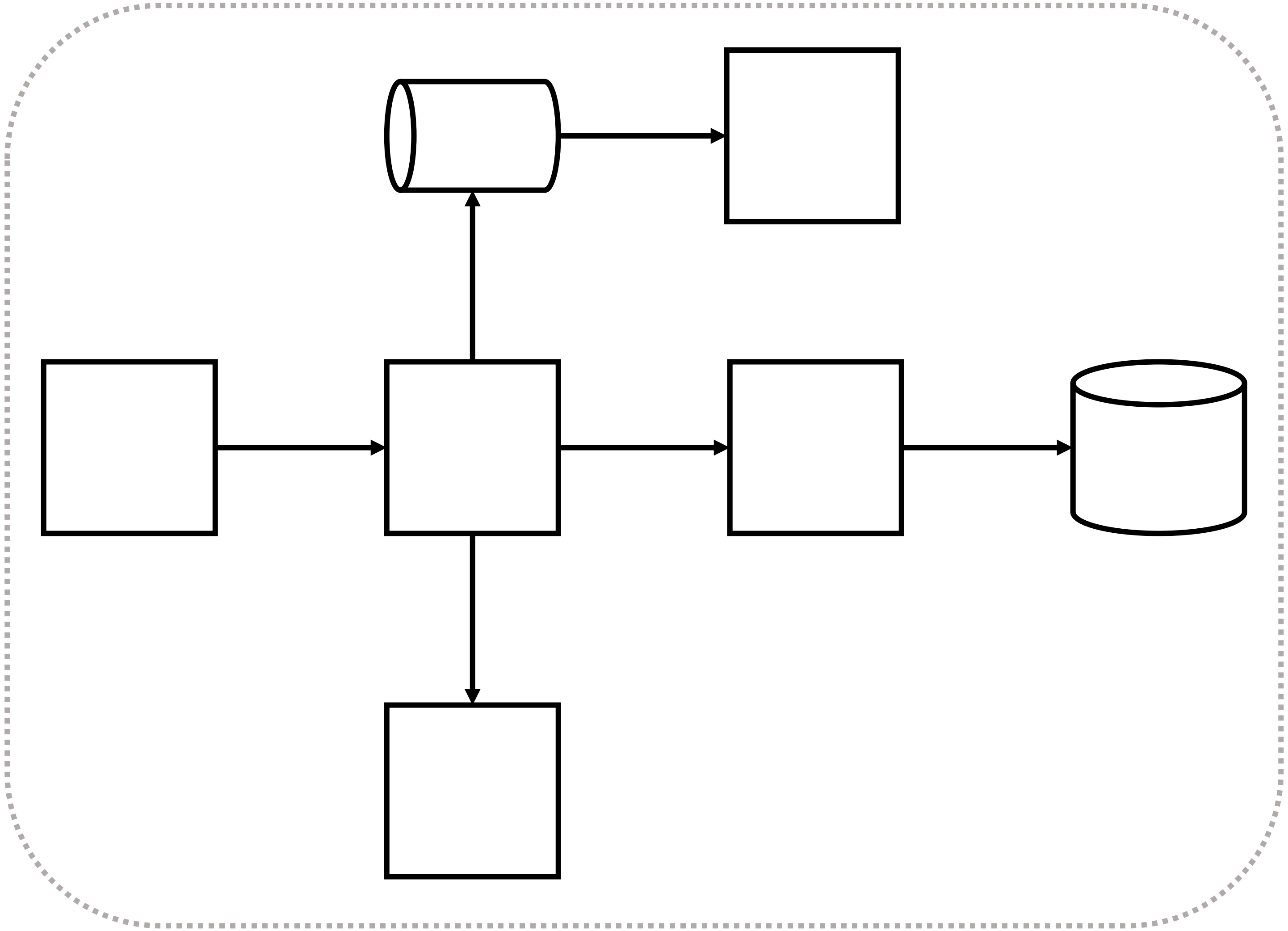
DevOops
2023

SRE и перформанс, или Когда производительность имеет значение

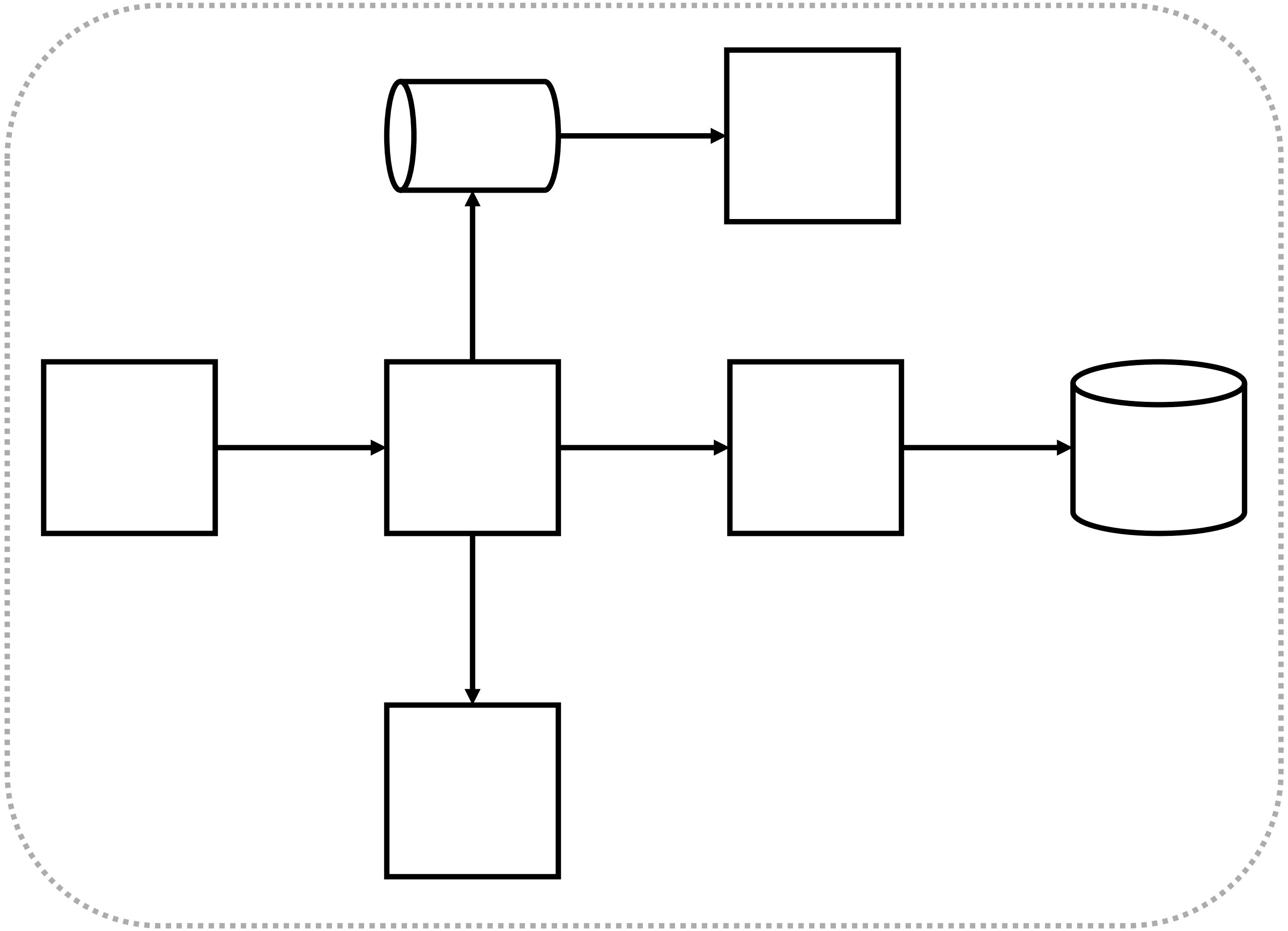
Григорий Кошелев

Контур

https://www.youtube.com/watch?v=ukXD3t_abOI



https://www.youtube.com/watch?v=ukXD3t_abOI



jattach

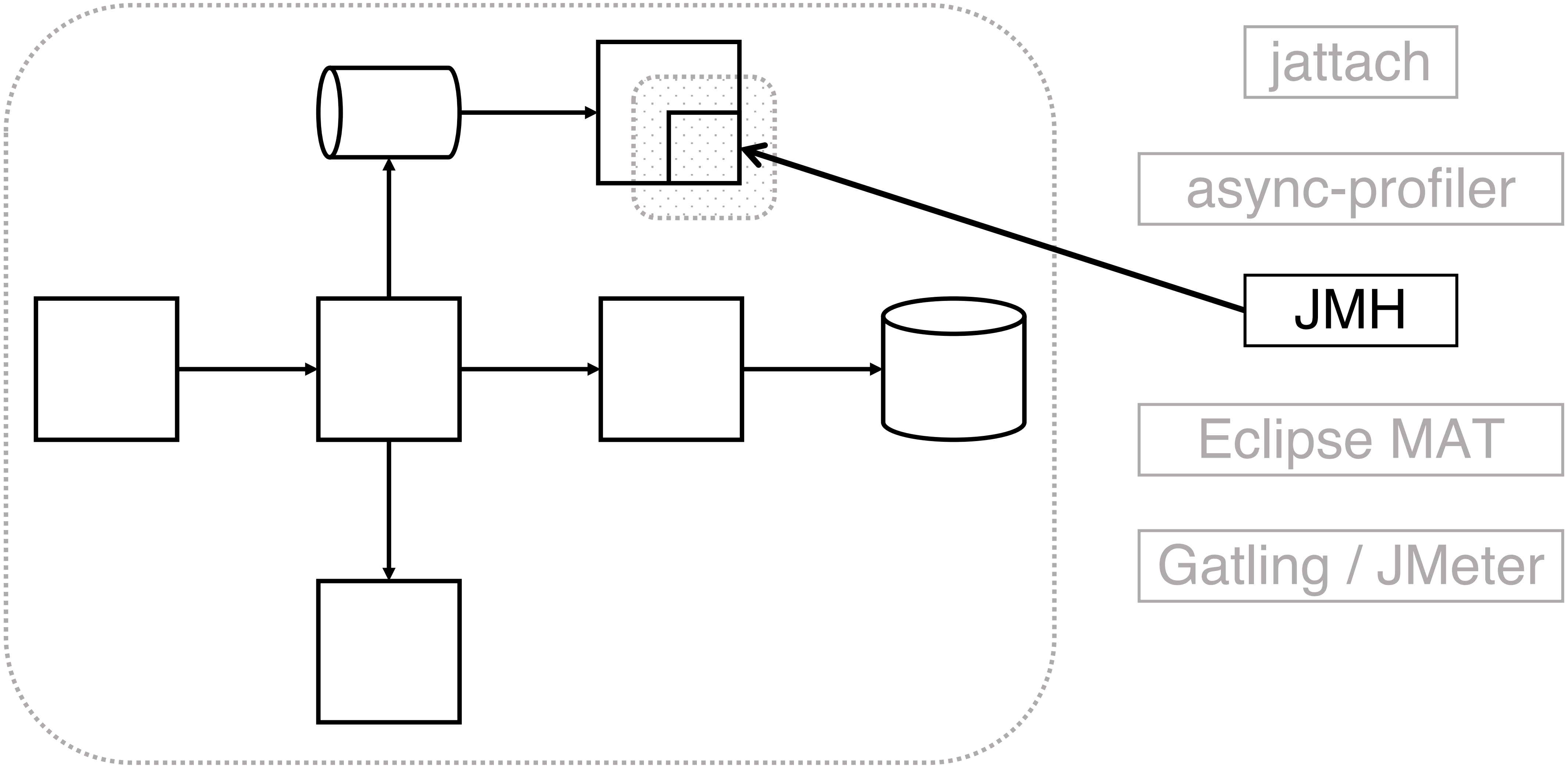
async-profiler

JMH

Eclipse MAT

Gatling / JMeter

https://www.youtube.com/watch?v=ukXD3t_abOI



jattach

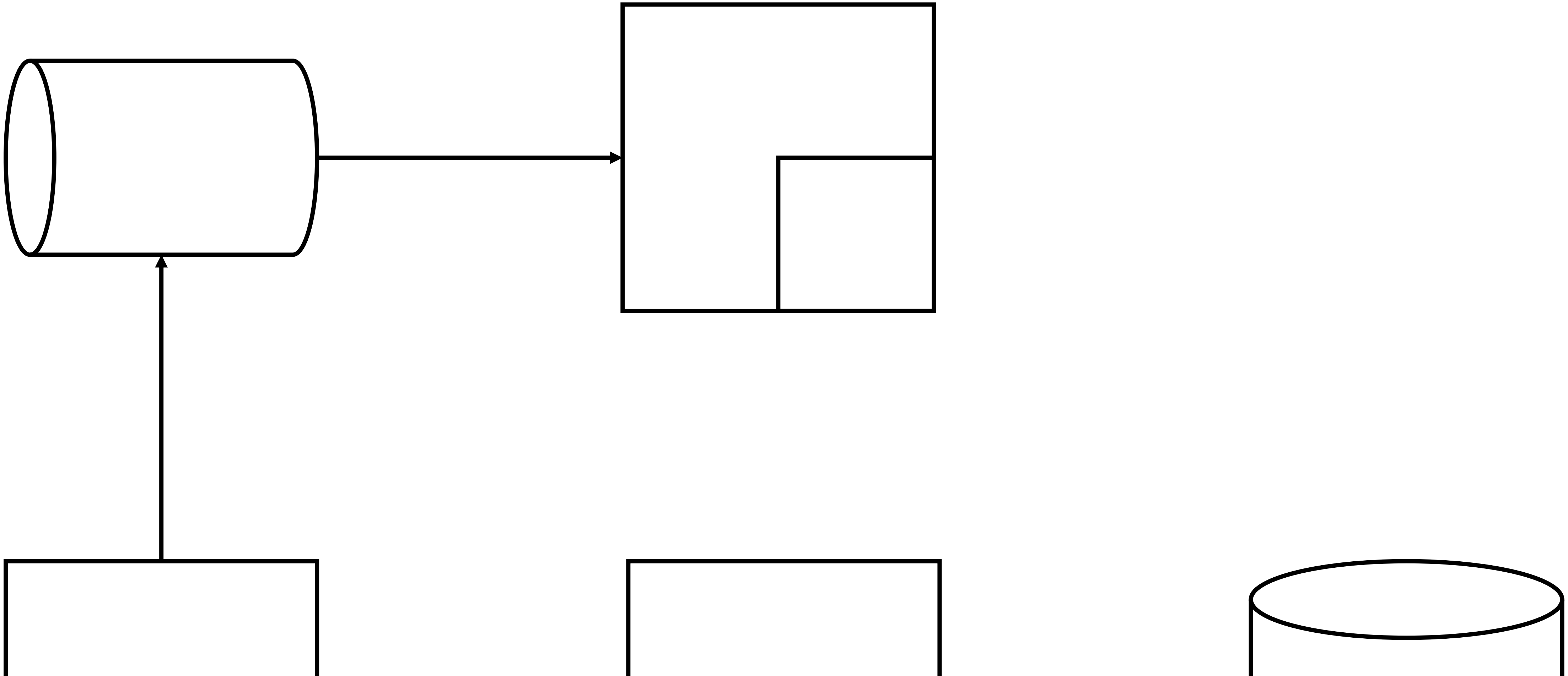
async-profiler

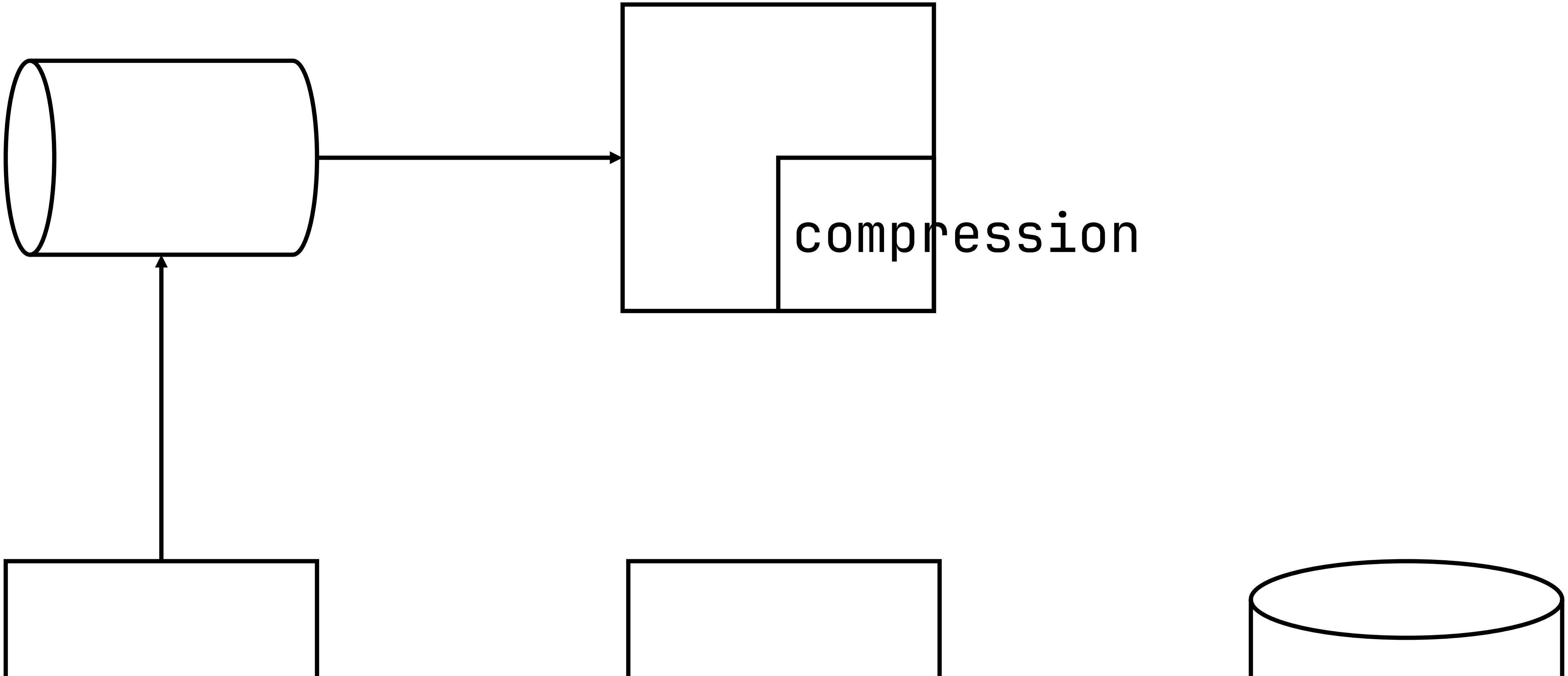
JMH

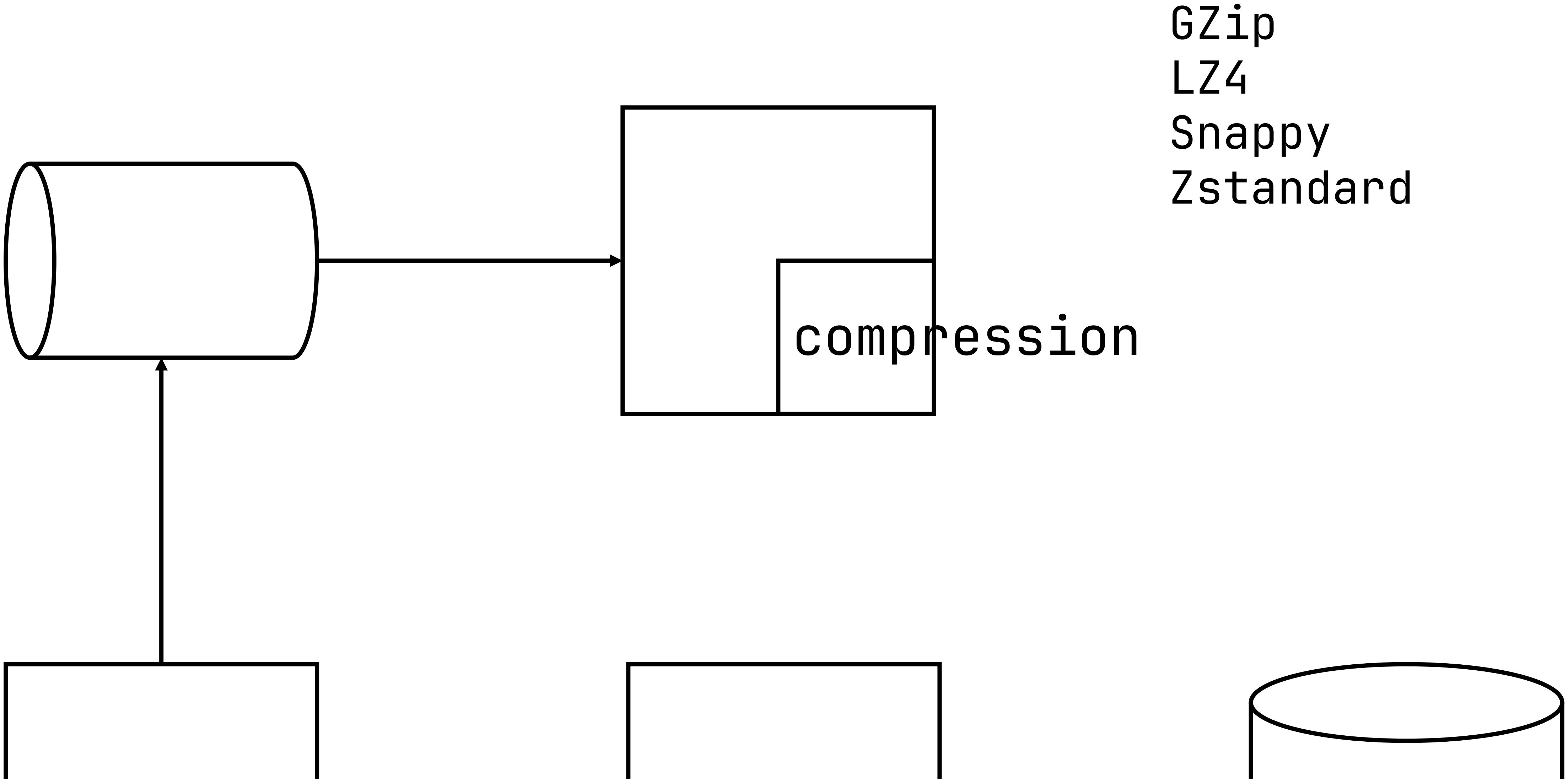
Eclipse MAT

Gatling / JMeter

https://www.youtube.com/watch?v=ukXD3t_abOI




compression



GZip
LZ4
Snappy
Zstandard

compression

# Когда микробенчмарки нужны?

— Сравнение скорости библиотек

# Когда микробенчмарки нужны?

— Сравнение скорости библиотек

— Оценка эффективности алгоритма

# Когда микробенчмарки нужны?

— Сравнение скорости библиотек

— Оценка эффективности алгоритма

— Пруфы для замены структуры данных

# Когда микробенчмарки нужны?

— Сравнение скорости библиотек

— Оценка эффективности алгоритма

— Пруфы для замены структуры данных

— Многопоточные тесты

# Когда микробенчмарки нужны?

— Сравнение скорости библиотек

— Оценка эффективности алгоритма

— Пруфы для замены структуры данных

— Многопоточные тесты

— Изучение работы JIT-компилятора

# Когда микробенчмарки нужны?

— Сравнение скорости библиотек

— Оценка эффективности алгоритма

— Пруфы для замены структуры данных

— Многопоточные тесты

— Изучение работы JIT-компилятора

— Собеседование с Ситниковым

«Premature optimization
is the root of all evil»

***Donald Knuth***

# «Premature optimization is the root of all evil»

«We *should* forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%»

*Structured Programming With Go To Statements, 1974*
***Donald Knuth***

http://pplab.snu.ac.kr/courses/adv_pl05/papers/p261-knuth.pdf

# «Premature optimization is the root of all evil»

«We *should* forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%»

*Structured Programming With*
*Go To Statements, 1974*
***Donald Knuth***

http://pplab.snu.ac.kr/courses/adv_pl05/papers/p261-knuth.pdf

# «Premature optimization is the root of all evil»

«We *should* forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%»

*Structured Programming With Go To Statements, 1974*
***Donald Knuth***

http://pplab.snu.ac.kr/courses/adv_pl05/papers/p261-knuth.pdf

# «Premature optimization is the root of all evil»

«We *should* forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%»

*Structured Programming With Go To Statements, 1974*
***Donald Knuth***

http://pplab.snu.ac.kr/courses/adv_pl05/papers/p261-knuth.pdf

# Как писать микробенчмарки?

# Как писать микробенчмарки?

«The scary thing about microbenchmarks is that they always produce a number, even if that number is meaningless.
They measure *something*; we're just not sure what»

*Java theory and practice:*
*Anatomy of a flawed microbenchmark, 2005*
***Brian Goetz***

http://www.ibm.com/developerworks/java/library/j-jtp02225/index.html

# Как писать микробенчмарки?

«The scary thing about microbenchmarks is that they always produce a number, even if that number is meaningless.
They measure *something*; we’re just not sure what»

*Java theory and practice:*
*Anatomy of a flawed microbenchmark, 2005*
***Brian Goetz***

http://www.ibm.com/developerworks/java/library/j-jtp02225/index.html

# Как писать микробенчмарки?

«The scary thing about microbenchmarks is that they always produce a number, even if that number is meaningless.
They measure *something*; we're just not sure what»

*Java theory and practice:*
*Anatomy of a flawed microbenchmark, 2005*
***Brian Goetz***

http://www.ibm.com/developerworks/java/library/j-jtp02225/index.html

# Как писать микробенчмарки?

«The scary thing about microbenchmarks is that they always produce a number, even if that number is meaningless. They measure *something*; we're just not sure what»

*Java theory and practice:*
*Anatomy of a flawed microbenchmark, 2005*
***Brian Goetz***

http://www.ibm.com/developerworks/java/library/j-jtp02225/index.html

# Как писать микробенчмарки?

«The scary thing about microbenchmarks is that they always produce a number, even if that number is meaningless. They measure *something*; we're just not sure what»

*Java theory and practice:*
*Anatomy of a flawed microbenchmark, 2005*
***Brian Goetz***

http://www.ibm.com/developerworks/java/library/j-jtp02225/index.html

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

https://wiki.openjdk.org/display/HotSpot/MicroBenchmarks

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

* Project Panama

https://wiki.openjdk.org/display/HotSpot/MicroBenchmarks

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

* Project Panama

https://wiki.openjdk.org/display/HotSpot/MicroBenchmarks

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

* Project Panama

* Project Valhalla

https://wiki.openjdk.org/display/HotSpot/MicroBenchmarks

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

https://wiki.openjdk.org/display/HotSpot/MicroBenchmarks

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

http://www.ibm.com/developerworks/java/library/j-jtp02225/index.html

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)

https://stackoverflow.com/questions/29170673/does-ahead-of-time-compilation-need-warmup-in-jmh

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

- Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)
- Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)
- Правило #2: убедись, что JVM не занята чем-то другим во время измерений (`--XX:+PrintCompilation -verbose:gc`)

# Как писать микробенчмарки?

## So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)

Правило #2: убедись, что JVM не занята чем-то другим во время измерений (`--XX:+PrintCompilation -verbose:gc`)

*Правило #3 устарело (`-client` vs –`server`)*

# Как писать микробенчмарки?

## So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)

Правило #2: убедись, что JVM не занята чем-то другим во время измерений (`--XX:+PrintCompilation -verbose:gc`)

*Правило #3 устарело (`-client` vs –`server`)*

Правило #4: помни, что вызов кода (печать измерений) может загружать классы

# Как писать микробенчмарки?

## So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)

Правило #2: убедись, что JVM не занята чем-то другим во время измерений (`--XX:+PrintCompilation -verbose:gc`)

*Правило #3 устарело (`-client` vs –`server`)*

Правило #4: помни, что вызов кода (печать измерений) может загружать классы

Правило #5: помни про деоптимизацию и перекомпиляцию кода

https://wiki.openjdk.org/display/HotSpot/PerformanceTechniques

# Как писать микробенчмарки?

## So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)

Правило #2: убедись, что JVM не занята чем-то другим во время измерений (`--XX:+PrintCompilation -verbose:gc`)

*Правило #3 устарело (`-client` vs –`server`)*

Правило #4: помни, что вызов кода (печать измерений) может загружать классы

Правило #5: помни про деоптимизацию и перекомпиляцию кода

Правило #6: следи за тем, во что компилируется код (`-XX:+PrintAssembly`)

https://wiki.openjdk.org/display/HotSpot/PrintAssembly

# Как писать микробенчмарки?

So You Want to Write a Micro-Benchmark (John Rose*, 2008)

Правило #0: изучи мат.чать (*Anatomy of a flawed microbenchmark, 2005, Brian Goetz*)

Правило #1: делай прогрев перед измерениями (JIT-компиляция и ***не только***)

Правило #2: убедись, что JVM не занята чем-то другим во время измерений (`--XX:+PrintCompilation -verbose:gc`)

*Правило #3 устарело (`-client` vs `–server`)*

Правило #4: помни, что вызов кода (печать измерений) может загружать классы

Правило #5: помни про деоптимизацию и перекомпиляцию кода

Правило #6: следи за тем, во что компилируется код (`-XX:+PrintAssembly`)

Правило #7: уменьши шум в замерах. Запускай бенчмарк на изолированной машине несколько раз, отбрасывай выбросы

# **J**ava **M**icrobenchmark **H**arness

# Почему использовать JMH?

# Почему использовать JMH?

«Use **JMH** to write useful benchmarks that **produce accurate results**»

*Avoiding Benchmarking Pitfalls on the JVM, 2014*
*Julien Ponge*

[https://www.oracle.com/technical-resources/articles/java/architect-benchmarking.html](https://www.oracle.com/technical-resources/articles/java/architect-benchmarking.html)

# Почему использовать JMH?

Caliper (Google)

https://github.com/google/caliper

# Почему использовать JMH?

Caliper (Google)

«For JVM benchmarks, use **JMH**, which
generally provides **more accurate results** than Caliper»

https://github.com/google/caliper

# Почему использовать JMH?

Caliper (Google)

«For JVM benchmarks, use **JMH**, which generally provides **more accurate results** than Caliper»

https://groups.google.com/g/mechanical-sympathy/c/m4opvy4xq3U/m/7lY8x8SvHgwJ

# Почему использовать JMH?

Алексей Шипилёв

# Почему использовать JMH?

Алексей Шипилёв

— (The Art of) (Java) Benchmarking II

*JPoint 2013*

https://www.youtube.com/watch?v=Mw0Vimj39cI

# Почему использовать JMH?

Алексей Шипилёв

— (The Art of) (Java) Benchmarking II

*JPoint 2013*

— Java Benchmarking:
как два таймстампа прочитать!

*Joker 2014*

https://www.youtube.com/watch?v=8pMfUopQ9Es

# Почему использовать JMH?

Алексей Шипилёв

— (The Art of) (Java) Benchmarking II

*JPoint 2013*

— Java Benchmarking:
как два таймстампа прочитать!

*Joker 2014*

— Nanotrusting nanotime

*[https://shipilev.net](https://shipilev.net), 2014*

https://shipilev.net/blog/2014/nanotrusting-nanotime/

# Почему использовать JMH?

— Воспроизводимость и достоверность результатов

# Почему использовать JMH?

— Воспроизводимость и достоверность результатов

— Инструменты для нивелирования JVM-эффектов

# Почему использовать JMH?

— Воспроизводимость и достоверность результатов

— Инструменты для нивелирования JVM-эффектов

— Широкие возможности для тюнинга микробенчмарков

# Почему использовать JMH?

— Воспроизводимость и достоверность результатов

— Инструменты для нивелирования JVM-эффектов

— Широкие возможности для тюнинга микробенчмарков

— Быстрый старт

# Крэш-курс по написанию микробенчмарков с использованием JMH

# Быстрый старт

# Быстрый старт

— Maven Archetype

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-java-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0
```

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-java-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0
```

*** java**

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-kotlin-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0
```

* java
* **kotlin**

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-groovy-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0
```

```
* java                  * groovy
* kotlin
```

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-scala-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0
```

```
* java                  * groovy
* kotlin                * scala
```

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-java-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0
```

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-java-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0

$ cd benchmarks/
$ mvn clean verify
```

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

```
$ mvn archetype:generate \
  -DinteractiveMode=false \
  -DarchetypeGroupId=org.openjdk.jmh \
  -DarchetypeArtifactId=jmh-java-benchmark-archetype \
  -DarchetypeVersion=1.37 \
  -DgroupId=ru.jpoint2025 \
  -DartifactId=benchmarks \
  -Dversion=1.0.0

$ cd benchmarks/
$ mvn clean verify

$ java -jar target/benchmarks.jar
```

https://github.com/openjdk/jmh#preferred-usage-command-line

# Быстрый старт

— Maven Archetype

— Gradle плагин

```
// gradle.build
      plugins {
        id "me.champeau.jmh" version "0.7.2"
      }
```

https://github.com/melix/jmh-gradle-plugin

# Быстрый старт

— Maven Archetype

— Gradle плагин

— Scala SBT плагин

```
// project/plugins.sbt
addSbtPlugin("pl.project13.scala" % "sbt-jmh" % "0.4.7")

// build.sbt
enablePlugins(JmhPlugin)
```

https://github.com/ktoso/sbt-jmh

# @Benchmark

```
package ru.jpoint2025;

import org.openjdk.jmh.annotations.Benchmark;

public class MyBenchmark {
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

# @Benchmark

```java
package ru.jpoint2025;

import org.openjdk.jmh.annotations.Benchmark;

public class MyBenchmark {
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# java -jar benchmarks.jar

```
$ java -jar target/benchmarks.jar
```

# java -jar benchmarks.jar

```
$ java -jar target/benchmarks.jar

$ java -jar target/benchmarks.jar MyBenchmark
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

https://shipilev.net/jvm/anatomy-quarks/27-compiler-blackholes/

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# JMH version: 1.37
# VM version: JDK 17.0.7, OpenJDK 64-Bit Server VM, 17.0.7+7-LTS
# VM invoker: /Library/Java/JVMs/jdk-17.jdk/Contents/Home/bin/java
# VM options: <none>
# Blackhole mode: compiler
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: ru.jpoint2025.MyBenchmark.testMethod
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 3478508218,760 ops/s
# Warmup Iteration   2: 3529259925,190 ops/s
# Warmup Iteration   3: 3465653930,005 ops/s
# Warmup Iteration   4: 3526605964,572 ops/s
# Warmup Iteration   5: 3533178891,560 ops/s
Iteration   1: 3546236109,396 ops/s
Iteration   2: 3545891025,826 ops/s
Iteration   3: 3542776397,619 ops/s
Iteration   4: 3542510489,779 ops/s
Iteration   5: 3559980495,192 ops/s
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# java -jar benchmarks.jar

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                 Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod   thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt          Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
      stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt          Score          Error  Units
MyBenchmark.testMethod  thrpt   25 3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt             Score             Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
      stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# java -jar benchmarks.jar

Вывод в CSV

```
$ java -jar target/benchmarks.jar -rf csv
```

# java -jar benchmarks.jar

Вывод в CSV

```
$ java -jar target/benchmarks.jar -rf csv
```

Вывод в JSON

```
$ java -jar target/benchmarks.jar -rf json
$ java -jar target/benchmarks.jar -h
```

```
...

-l                List the benchmarks that match a
                  filter, and exit.

-lp               List the benchmarks that match a filter, along with
                  parameters, and exit.

-lrf              List machine-readable result formats, and exit.

-lprof            List profilers, and exit.

-h                Display help, and exit.
```

# @Fork

```
# Run progress: 0,00% complete, ETA 00:08:20
# Fork: 1 of 5
...
# Run progress: 20,00% complete, ETA 00:07:01
# Fork: 2 of 5
...
# Run progress: 40,00% complete, ETA 00:05:16
# Fork: 3 of 5
...
# Run progress: 60,00% complete, ETA 00:03:30
# Fork: 4 of 5
...
# Run progress: 80,00% complete, ETA 00:01:45
# Fork: 5 of 5
...
```

# @Fork

```
import org.openjdk.jmh.annotations.Benchmark;


public class MyBenchmark_1 {


    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Fork

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Fork;

@Fork(value = 3)
public class MyBenchmark_1 {


    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Fork

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Fork;

@Fork(value = 3)
public class MyBenchmark_1 {


    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

# @Fork

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Fork;

@Fork(value = 3)
public class MyBenchmark_1 {


    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

# @Fork

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Fork;

@Fork(value = 3)
public class MyBenchmark_1 {

    @Fork(value = 1)
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Fork

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Fork;

@Fork(value = 3)
public class MyBenchmark_1 {

    @Fork(value = 1)
    @Benchmark
    public void testMethod() {
        // здесь должен быть котд
    }
}
```

Победит ближайший
к @Benchmark

# @Fork

Зачем запускать бенчмарки в отдельных JVM?

# @Fork

Зачем запускать бенчмарки в отдельных JVM?
— Avoiding Benchmarking Pitfalls on the JVM (Julien Ponge)

https://www.oracle.com/technical-resources/articles/java/architect-benchmarking.html

# @Fork

Зачем запускать бенчмарки в отдельных JVM?

— Avoiding Benchmarking Pitfalls on the JVM (Julien Ponge)

— JMHSample_12_Forking

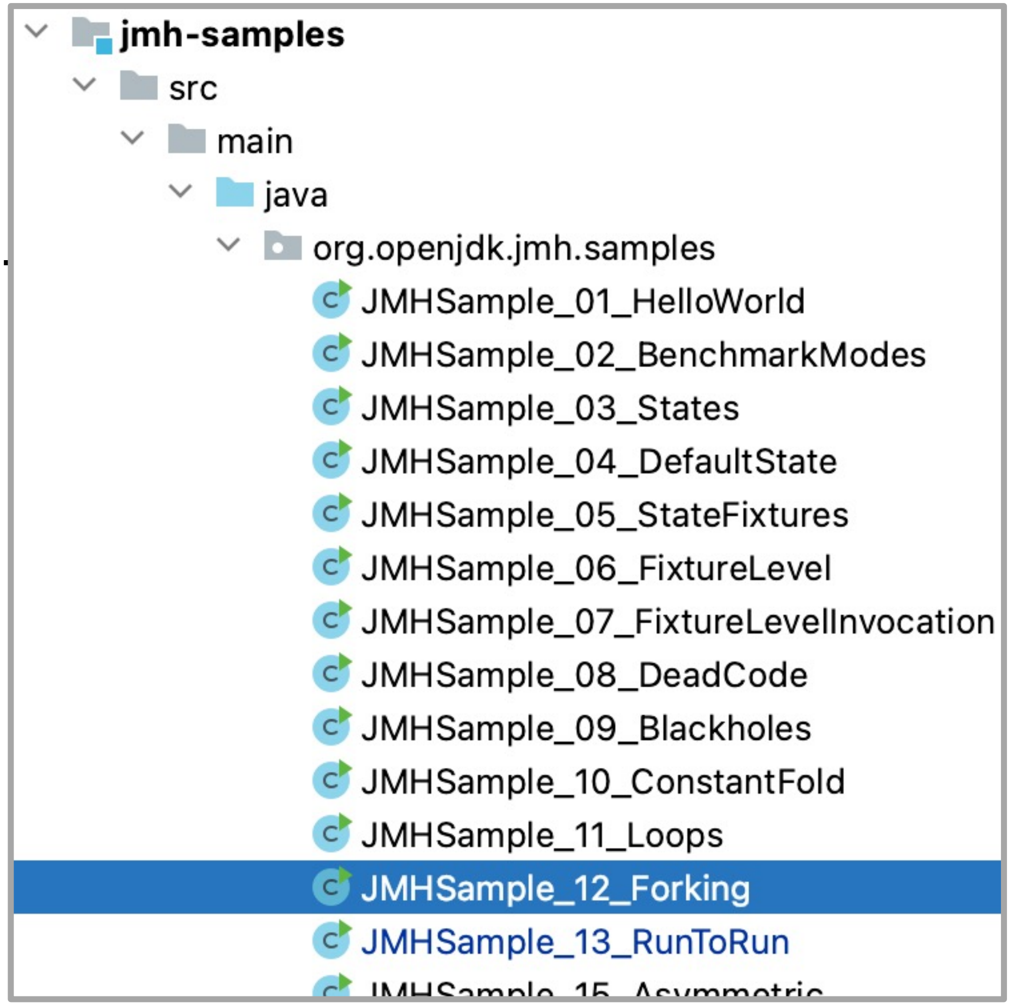
https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

Зачем запускать бенчмарки

— Avoiding Benchmarking Pit

— JMHSample_12_Forking



- jmh-samples
  - src
    - main
      - java
        - org.openjdk.jmh.samples
          - JMHSample_01_HelloWorld
          - JMHSample_02_BenchmarkModes
          - JMHSample_03_States
          - JMHSample_04_DefaultState
          - JMHSample_05_StateFixtures
          - JMHSample_06_FixtureLevel
          - JMHSample_07_FixtureLevelInvocation
          - JMHSample_08_DeadCode
          - JMHSample_09_Blackholes
          - JMHSample_10_ConstantFold
          - JMHSample_11_Loops
          - JMHSample_12_Forking
          - JMHSample_13_RunToRun
          - JMHSample_15_Asymmetric

https://github.com/openjdk/jmh/tree/master/jmh-samples

# @Fork

Зачем запускать бенчмарки в отдельных JVM?

— Avoiding Benchmarking Pitfalls on the JVM (Julien Ponge)

— JMHSample_12_Forking

```
public class Counter1
       implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```
public class Counter2
       implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

```
Benchmark                 Mode  Cnt   Score   Error  Units
```

```java
public class Counter1
        implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```java
public class Counter2
        implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

```
Benchmark              Mode  Cnt   Score   Error  Units
measure_1_c1           avgt    5   1,148 ± 0,003  ns/op
```

```java
public class Counter1
      implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```java
public class Counter2
      implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

```
Benchmark            Mode  Cnt   Score   Error  Units
measure_1_c1         avgt    5   1,148 ± 0,003  ns/op
measure_2_c2         avgt    5  13,603 ± 3,110  ns/op
```

```
public class Counter1
       implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```
public class Counter2
       implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

```
Benchmark                 Mode  Cnt    Score    Error  Units
measure_1_c1              avgt    5    1,148 ±  0,003  ns/op
measure_2_c2              avgt    5   13,603 ±  3,110  ns/op
```

```java
public class Counter1
        implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```java
public class Counter2
        implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

```
Benchmark              Mode  Cnt   Score    Error  Units
measure_1_c1           avgt    5   1,148 ± 0,003  ns/op
measure_2_c2           avgt    5  13,603 ± 3,110  ns/op
measure_3_c1_again     avgt    5   7,086 ± 0,223  ns/op
```

```java
public class Counter1
        implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```java
public class Counter2
        implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# @Fork

```
Benchmark              Mode  Cnt   Score   Error  Units
measure_1_c1           avgt    5   1,148 ± 0,003  ns/op
measure_2_c2           avgt    5  13,603 ± 3,110  ns/op
measure_3_c1_again     avgt    5   7,086 ± 0,223  ns/op
```

```
public class Counter1
       implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

```
public class Counter2
       implements Counter {
    private int x;

    @Override
    public int inc() {
        return x++;
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_12_Forking.java

# ⚠️ Неправильный бенчмарк

Проблема

Влияние бенчмарков друг на друга
в пределах одной JVM

# ⚠️ Неправильный бенчмарк

Проблема

Влияние бенчмарков друг на друга
в пределах одной JVM

Решение

```
@Fork(value = n),
где n>0
```

# @Fork

Зачем несколько форков (>1)?

# @Fork

```
   Benchmark  Mode  Cnt    Score       Error  Units
    baseline  avgt    5  580,483 ±    1,073  ms/op
```

# @Fork

```
   Benchmark   Mode   Cnt      Score       Error   Units
    baseline   avgt     5  580,483 ±     1,073   ms/op
```

# @Fork

```
   Benchmark   Mode   Cnt      Score        Error   Units
    baseline   avgt     5   580,483 ±     1,073   ms/op
    fork_3     avgt    15   365,287 ±   355,951   ms/op
    fork_5     avgt    25   669,307 ±   110,744   ms/op
```

# @Fork

```
  Benchmark   Mode   Cnt     Score        Error   Units
   baseline   avgt     5   580,483 ±    1,073   ms/op
   fork_3     avgt    15   365,287 ±  355,951   ms/op
   fork_5     avgt    25   669,307 ±  110,744   ms/op
```

# @Fork

```
// ...
@State(Scope.Benchmark)
public class NonSteadyBenchmark {
    private long sleepTime = (long) (Math.random() * 1000);

    @Benchmark @Fork(1)
    public void baseline() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }

    @Benchmark @Fork(3)
    public void fork_3() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }
}
```

# @Fork

```
// ...
@State(Scope.Benchmark)
public class NonSteadyBenchmark {
    private long sleepTime = (long) (Math.random() * 1000);

    @Benchmark @Fork(1)
    public void baseline() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }

    @Benchmark @Fork(3)
    public void fork_3() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }
}
```

# @Fork

```java
// ...
@State(Scope.Benchmark)
public class NonSteadyBenchmark {
    private long sleepTime = (long) (Math.random() * 1000);

    @Benchmark @Fork(1)
    public void baseline() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }

    @Benchmark @Fork(3)
    public void fork_3() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }
}
```

# @Fork

```
// ...
@State(Scope.Benchmark)
public class NonSteadyBenchmark {
    private long sleepTime = (long) (Math.random() * 1000);

    @Benchmark @Fork(1)
    public void baseline() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }

    @Benchmark @Fork(3)
    public void fork_3() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(sleepTime);
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# ⚠️ Неправильный бенчмарк

Проблема

Отсутствует воспроизводимость
результатов бенчмарка

# ⚠️ Неправильный бенчмарк

Проблема

Отсутствует воспроизводимость
результатов бенчмарка

Диагностика

```
@Fork(value = n),
где n≥3
```

# @Warmup + @Measurement

# @Warmup + @Measurement

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Warmup;

import java.util.concurrent.TimeUnit;

@Warmup(iterations = 3, time = 5, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
public class MyBenchmark_2 {
    @Benchmark
    public void testMethod() {
        // здесь должен быть код
    }
}
```

# @Warmup + @Measurement

```java
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Warmup;

import java.util.concurrent.TimeUnit;

@Warmup(iterations = 3, time = 5, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
public class MyBenchmark_2 {
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

# @Warmup + @Measurement

```java
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Warmup;

import java.util.concurrent.TimeUnit;

@Warmup(iterations = 3, time = 5, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
public class MyBenchmark_2 {
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко∓д
    }
}
```

# @Warmup + @Measurement

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Warmup;

import java.util.concurrent.TimeUnit;

@Warmup(iterations = 3, time = 5, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
public class MyBenchmark_2 {
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Warmup + @Measurement

```java
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Warmup;

import java.util.concurrent.TimeUnit;

@Warmup(iterations = 3, time = 5, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
public class MyBenchmark_2 {
    @Benchmark
    public void testMethod() {
        // здесь должен быть код
    }
}
```

# @Warmup + @Measurement

```
import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Warmup;

import java.util.concurrent.TimeUnit;

@Warmup(iterations = 3, time = 5, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = TimeUnit.SECONDS)
public class MyBenchmark_2 {
    @Benchmark
    public void testMethod() {
        // здесь должен быть ко~~т~~д
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# ⚠️ Неправильный бенчмарк

Проблема

Отсутствует фаза прогрева
(не учитывается переходный процесс)

# ⚠️ Неправильный бенчмарк

Проблема

Отсутствует фаза прогрева
(не учитывается переходный процесс)

Решение

```
@Warmup(value = n),
где n≥3
```

# @Measurement

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
public class HeavyBenchmark_1 {
    @Benchmark
    @Measurement(iterations = 3, time = 1)
    public void measure_03() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @Measurement

```java
@Fork(1)
@Warmup(iterations = 3, time = 10)
public class HeavyBenchmark_1 {
    @Benchmark
    @Measurement(iterations = 3, time = 1)
    public void measure_03() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @Measurement

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
public class HeavyBenchmark_1 {
    @Benchmark
    @Measurement(iterations = 3, time = 1)
    public void measure_03() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @Measurement

```java
@Fork(1)
@Warmup(iterations = 3, time = 10)
public class HeavyBenchmark_1 {
    @Benchmark
    @Measurement(iterations = 3, time = 1)
    public void measure_03() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Measurement

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
public class HeavyBenchmark_1 {
    @Benchmark
    @Measurement(iterations = 3, time = 1)
    public void measure_03() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @Measurement

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
public class HeavyBenchmark_1 {
    @Benchmark
    @Measurement(iterations = 3, time = 1)
    public void measure_03() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Measurement

```
@Measurement(iterations = X, time = 1)
Benchmark    Mode    X  Score   Error  Units
measure_03  thrpt    3  3,937 ± 0,169  ops/s
measure_05  thrpt    5  3,939 ± 0,043  ops/s
measure_20  thrpt   20  3,940 ± 0,011  ops/s
```

# @Measurement

```
@Measurement(iterations = X, time = 1)
Benchmark    Mode    X  Score   Error  Units
measure_03  thrpt    3  3,937 ± 0,169  ops/s
measure_05  thrpt    5  3,939 ± 0,043  ops/s
measure_20  thrpt   20  3,940 ± 0,011  ops/s
```

# @Measurement

```
@Measurement(iterations = X, time = 1)
Benchmark     Mode      X  Score    Error  Units
measure_03  thrpt      3  3,937 ± 0,169  ops/s
measure_05  thrpt      5  3,939 ± 0,043  ops/s
measure_20  thrpt     20  3,940 ± 0,011  ops/s
```

```
@Measurement(iterations = X, time = 10)
Benchmark     Mode      X  Score    Error  Units
measure_03  thrpt      3  3,941 ± 0,021  ops/s
measure_05  thrpt      5  3,938 ± 0,012  ops/s
measure_20  thrpt     20  3,939 ± 0,006  ops/s
```

# ⚠️ Неправильный бенчмарк

Проблема

Низкая точность измерения
из-за малого количества измерений

# ⚠ Неправильный бенчмарк

Проблема

Низкая точность измерения
из-за малого количества измерений

Решение

`@Fork(value = f)` *и*
`@Measurement(value = n, time = t)`,
*где* n×f≥15, t ≫ время метода

```
Benchmark       Mode  Cnt   Score    Error  Units
measure_03     thrpt    3   3,937 ± 0,169  ops/s
```

```java
public void heavyMethod() throws InterruptedException {
    TimeUnit.MILLISECONDS.sleep(250);
}
```

```
Benchmark       Mode  Cnt   Score    Error  Units
measure_03     thrpt    3   3,937 ± 0,169  ops/s
```

```java
public void heavyMethod() throws InterruptedException {
    TimeUnit.MILLISECONDS.sleep(250);
}
```

# @BenchmarkMode

```java
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.Throughput)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @BenchmarkMode

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.Throughput)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @BenchmarkMode

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.AverageTime)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @BenchmarkMode

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.SampleTime)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @BenchmarkMode

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.SingleShotTime)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    public void heavyMethod() throws InterruptedException {
        TimeUnit.MILLISECONDS.sleep(250);
    }
}
```

# @BenchmarkMode

```
Benchmark                Mode  Cnt  Score     Error  Units
measure                 thrpt   10  3,936 ±  0,008  ops/s
measure                  avgt   10  0,254 ±  0,001   s/op
measure                sample  391  0,257 ±  0,001   s/op
measure:p0.00          sample       0,250            s/op
measure:p0.50          sample       0,258            s/op
measure:p0.90          sample       0,260            s/op
measure:p0.95          sample       0,260            s/op
measure:p0.99          sample       0,260            s/op
measure:p0.999         sample       0,261            s/op
measure:p0.9999        sample       0,261            s/op
measure:p1.00          sample       0,261            s/op
measure                    ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark          Mode  Cnt  Score    Error  Units
measure           thrpt   10  3,936 ±  0,008  ops/s
measure            avgt   10  0,254 ±  0,001   s/op
measure          sample  391  0,257 ±  0,001   s/op
measure:p0.00    sample       0,250            s/op
measure:p0.50    sample       0,258            s/op
measure:p0.90    sample       0,260            s/op
measure:p0.95    sample       0,260            s/op
measure:p0.99    sample       0,260            s/op
measure:p0.999   sample       0,261            s/op
measure:p0.9999  sample       0,261            s/op
measure:p1.00    sample       0,261            s/op
measure              ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark                Mode  Cnt  Score    Error  Units
measure                 thrpt   10  3,936 ±  0,008  ops/s
measure                  avgt   10  0,254 ±  0,001   s/op
measure                sample  391  0,257 ±  0,001   s/op
measure:p0.00          sample       0,250            s/op
measure:p0.50          sample       0,258            s/op
measure:p0.90          sample       0,260            s/op
measure:p0.95          sample       0,260            s/op
measure:p0.99          sample       0,260            s/op
measure:p0.999         sample       0,261            s/op
measure:p0.9999        sample       0,261            s/op
measure:p1.00          sample       0,261            s/op
measure                    ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark          Mode  Cnt  Score    Error  Units
measure           thrpt   10  3,936 ±  0,008  ops/s
measure            avgt   10  0,254 ±  0,001   s/op
measure          sample  391  0,257 ±  0,001   s/op
measure:p0.00    sample       0,250            s/op
measure:p0.50    sample       0,258            s/op
measure:p0.90    sample       0,260            s/op
measure:p0.95    sample       0,260            s/op
measure:p0.99    sample       0,260            s/op
measure:p0.999   sample       0,261            s/op
measure:p0.9999  sample       0,261            s/op
measure:p1.00    sample       0,261            s/op
measure              ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark          Mode  Cnt  Score    Error  Units
measure           thrpt   10  3,936 ±  0,008  ops/s
measure            avgt   10  0,254 ±  0,001   s/op
measure          sample  391  0,257 ±  0,001   s/op
measure:p0.00    sample       0,250            s/op
measure:p0.50    sample       0,258            s/op
measure:p0.90    sample       0,260            s/op
measure:p0.95    sample       0,260            s/op
measure:p0.99    sample       0,260            s/op
measure:p0.999   sample       0,261            s/op
measure:p0.9999  sample       0,261            s/op
measure:p1.00    sample       0,261            s/op
measure              ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark          Mode  Cnt  Score    Error  Units
measure           thrpt   10  3,936 ±  0,008  ops/s
measure            avgt   10  0,254 ±  0,001   s/op
measure          sample  391  0,257 ±  0,001   s/op
measure:p0.00    sample       0,250            s/op
measure:p0.50    sample       0,258            s/op
measure:p0.90    sample       0,260            s/op
measure:p0.95    sample       0,260            s/op
measure:p0.99    sample       0,260            s/op
measure:p0.999   sample       0,261            s/op
measure:p0.9999  sample       0,261            s/op
measure:p1.00    sample       0,261            s/op
measure              ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark          Mode  Cnt  Score    Error  Units
measure           thrpt   10  3,936 ±  0,008  ops/s
measure            avgt   10  0,254 ±  0,001   s/op
measure          sample  391  0,257 ±  0,001   s/op
measure:p0.00    sample       0,250            s/op
measure:p0.50    sample       0,258            s/op
measure:p0.90    sample       0,260            s/op
measure:p0.95    sample       0,260            s/op
measure:p0.99    sample       0,260            s/op
measure:p0.999   sample       0,261            s/op
measure:p0.9999  sample       0,261            s/op
measure:p1.00    sample       0,261            s/op
measure              ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark           Mode  Cnt  Score    Error  Units
measure            thrpt   10  3,936 ±  0,008  ops/s
measure             avgt   10  0,254 ±  0,001   s/op
measure           sample  391  0,257 ±  0,001   s/op
measure:p0.00     sample       0,250            s/op
measure:p0.50     sample       0,258            s/op
measure:p0.90     sample       0,260            s/op
measure:p0.95     sample       0,260            s/op
measure:p0.99     sample       0,260            s/op
measure:p0.999    sample       0,261            s/op
measure:p0.9999   sample       0,261            s/op
measure:p1.00     sample       0,261            s/op
measure               ss   10  0,257 ±  0,006   s/op
```

# @BenchmarkMode

```
Benchmark          Mode  Cnt  Score    Error  Units
measure           thrpt   10  3,936 ±  0,008  ops/s
measure            avgt   10  0,254 ±  0,001   s/op
measure          sample  391  0,257 ±  0,001   s/op
measure:p0.00    sample       0,250            s/op
measure:p0.50    sample       0,258            s/op
measure:p0.90    sample       0,260            s/op
measure:p0.95    sample       0,260            s/op
measure:p0.99    sample       0,260            s/op
measure:p0.999   sample       0,261            s/op
measure:p0.9999  sample       0,261            s/op
measure:p1.00    sample       0,261            s/op
measure              ss   10  0,257 ±  0,006   s/op
```

```
Benchmark     Mode  Cnt  Score    Error  Units
measure       avgt   10  0,254 ± 0,001   s/op
```

```
Benchmark      Mode  Cnt  Score     Error   Units
measure        avgt   10  0,254 ± 0,001     s/op
```

# @OutputTimeUnit

```java
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    // ...
}
```

# @OutputTimeUnit

```
@Fork(1)
@Warmup(iterations = 3, time = 10)
@Measurement(iterations = 10, time = 10)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class HeavyBenchmark_3 {
    @Benchmark
    public void measure() throws InterruptedException {
        heavyMethod();

    }

    // ...
}
```

# @OutputTimeUnit

```
Benchmark  Mode  Cnt    Score   Error  Units
measure    avgt   10  253,783 ± 0,550  ms/op
```

# @OutputTimeUnit

```
Benchmark  Mode  Cnt    Score   Error  Units
measure    avgt   10  253,783 ± 0,550  ms/op
```

# @State

# @State

```java
private double compute(double d) {
    for (int c = 0; c < 10; c++) {
        d = d * d / Math.PI;
    }
    return d;
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
@Benchmark
public double baseline() {
    return Math.PI;
}

@Benchmark
public double measure() {
    return compute(Math.PI);
}
```

# @State

```
Benchmark    Mode  Cnt  Score     Error  Units
baseline     avgt    5  0,278 ±  0,001  ns/op
measure      avgt    5  0,278 ±  0,001  ns/op
```

```java
@Benchmark
public double baseline() {
    return Math.PI;
}

@Benchmark
public double measure() {
    return compute(Math.PI);
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @State

```
     Benchmark    Mode  Cnt  Score       Error  Units
     baseline     avgt    5  0,278 ±  0,001  ns/op
     measure      avgt    5  0,278 ±  0,001  ns/op
```

```java
@Benchmark
public double baseline() {
    return Math.PI;
}

@Benchmark
public double measure() {
    return compute(Math.PI);
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @State

```java
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```java
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

[https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java](https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java)

# @State

```java
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```java
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
public class JMHSample_10_ConstantFold {
    private double x = Math.PI;
    private final double wrongX = Math.PI;

    @Benchmark public double baseline() { return Math.PI; }

    @Benchmark public double wrong() { return compute(wrongX); }

    @Benchmark public double measureRight() { return compute(x); }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
Benchmark    Mode  Cnt  Score    Error  Units
baseline     avgt   15  0,310 ± 0,091  ns/op
right        avgt   15  4,717 ± 0,530  ns/op
wrong        avgt   15  0,288 ± 0,030  ns/op
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
Benchmark    Mode  Cnt  Score   Error   Units
baseline     avgt   15  0,310 ± 0,091  ns/op
right        avgt   15  4,717 ± 0,530  ns/op
wrong        avgt   15  0,288 ± 0,030  ns/op
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
Benchmark    Mode  Cnt  Score     Error  Units
baseline     avgt   15  0,310 ± 0,091  ns/op
right        avgt   15  4,717 ± 0,530  ns/op
wrong        avgt   15  0,288 ± 0,030  ns/op
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# @State

```
Benchmark    Mode  Cnt  Score     Error  Units
baseline     avgt   15  0,310 ± 0,091  ns/op
right        avgt   15  4,717 ± 0,530  ns/op
wrong        avgt   15  0,288 ± 0,030  ns/op
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_10_ConstantFold.java

# ⚠️ Неправильный бенчмарк

Проблема

Некорректное измерение
из-за Constant Folding

# ⚠️ Неправильный бенчмарк

Проблема

Некорректное измерение
из-за Constant Folding

Решение

```
@State
+
НЕ final
```

# @Setup

# @Setup

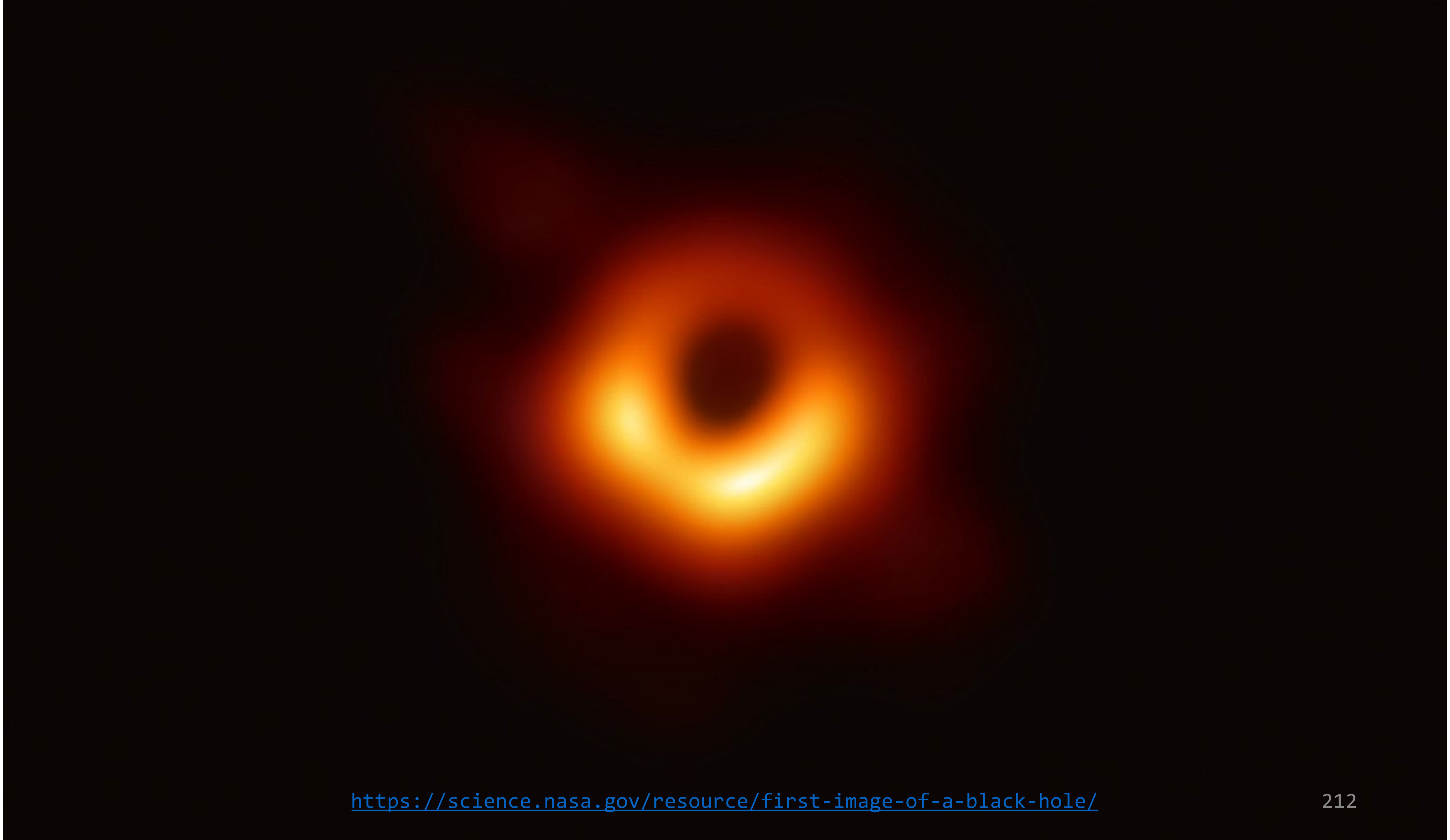
```java
// ...
@State(Scope.Thread)
public static class Maps {
    private Map<Integer, Integer> map;

    private int begin, end;

    @Setup
    public void setup() {
        map = new HashMap<>();
        begin = 1; end = 256;
        for (int i = begin; i < end; i++) {
            map.put(i, i);
        }
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# @Setup

```
// ...
@State(Scope.Thread)
public static class Maps {
    private Map<Integer, Integer> map;

    private int begin, end;

    @Setup
    public void setup() {
        map = new HashMap<>();
        begin = 1; end = 256;
        for (int i = begin; i < end; i++) {
            map.put(i, i);
        }
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java



https://science.nasa.gov/resource/first-image-of-a-black-hole/

# Blackhole

```java
private double compute(double d) {
    for (int c = 0; c < 10; c++) {
        d = d * d / Math.PI;
    }
    return d;
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```java
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
Benchmark     Mode  Cnt  Score   Error  Units
baseline      avgt   25  4,333 ± 0,041  ns/op
measureRight  avgt   25  8,662 ± 0,101  ns/op
measureWrong  avgt   25  4,616 ± 0,353  ns/op
```

# Blackhole

```
Benchmark      Mode  Cnt  Score   Error  Units
baseline       avgt   25  4,333 ± 0,041  ns/op
measureRight   avgt   25  8,662 ± 0,101  ns/op
measureWrong   avgt   25  4,616 ± 0,353  ns/op
```

# Blackhole

```
Benchmark     Mode  Cnt  Score   Error  Units
baseline      avgt   25  4,333 ± 0,041  ns/op
measureRight  avgt   25  8,662 ± 0,101  ns/op
measureWrong  avgt   25  4,616 ± 0,353  ns/op
```

# Blackhole

```
Benchmark     Mode  Cnt  Score   Error  Units
baseline      avgt   25  4,333 ± 0,041  ns/op
measureRight  avgt   25  8,662 ± 0,101  ns/op
measureWrong  avgt   25  4,616 ± 0,353  ns/op
```

# Blackhole

```java
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# Blackhole

```
// ...
public class JMHSample_09_Blackholes {
    @Benchmark
    public double baseline() { return compute(x1); }

    @Benchmark
    public double measureWrong() {
        compute(x1); return compute(x2);
    }

    @Benchmark
    public void measureRight(Blackhole bh) {
        bh.consume(compute(x1)); bh.consume(compute(x2));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_09_Blackholes.java

# ⚠️ Неправильный бенчмарк

Проблема

Некорректное измерение
из-за Dead Code Elimination

# ⚠ Неправильный бенчмарк

Проблема

Некорректное измерение
из-за Dead Code Elimination

Решение

*Бенчмарк-метод возвращает значение* `x`
*или*
*используется* `Blackhole.consume(x)`

# @Param

# @Param

```
// ...
@State(Scope.Benchmark)
public class ParametrizedBenchmark {

    @Param({"1", "2", "3.14159"})
    private double a;

    @Param({"0", "1", "2", "2.71828"})
    private double b;

    @Benchmark
    public double power() {
        return Math.pow(a, b);
    }
}
```

# @Param

```
// ...
@State(Scope.Benchmark)
public class ParametrizedBenchmark {

    @Param({"1", "2", "3.14159"})
    private double a;

    @Param({"0", "1", "2", "2.71828"})
    private double b;

    @Benchmark
    public double power() {
        return Math.pow(a, b);
    }
}
```

# @Param

```java
// ...
@State(Scope.Benchmark)
public class ParametrizedBenchmark {

    @Param({"1", "2", "3.14159"})
    private double a;

    @Param({"0", "1", "2", "2.71828"})
    private double b;

    @Benchmark
    public double power() {
        return Math.pow(a, b);
    }
}
```

# @Param

```java
// ...
@State(Scope.Benchmark)
public class ParametrizedBenchmark {

    @Param({"1", "2", "3.14159"})
    private double a;

    @Param({"0", "1", "2", "2.71828"})
    private double b;

    @Benchmark
    public double power() {
        return Math.pow(a, b);
    }
}
```

# @Param

```java
// ...
@State(Scope.Benchmark)
public class ParametrizedBenchmark {

    @Param({"1", "2", "3.14159"})
    private double a;

    @Param({"0", "1", "2", "2.71828"})
    private double b;

    @Benchmark
    public double power() {
        return Math.pow(a, b);
    }
}
```

# @Param

```
Benchmark      (a)      (b)  Mode  Cnt   Score   Error  Units
    power        1        0  avgt    5   7,703 ± 0,007  ns/op
    power        1        1  avgt    5  10,335 ± 0,026  ns/op
    power        1        2  avgt    5   1,904 ± 0,003  ns/op
    power        1  2.71828  avgt    5  11,000 ± 0,028  ns/op
    power        2        0  avgt    5   6,375 ± 0,007  ns/op
    power        2        1  avgt    5   9,559 ± 0,007  ns/op
    power        2        2  avgt    5   2,443 ± 0,458  ns/op
    power        2  2.71828  avgt    5   9,557 ± 0,017  ns/op
    power  3.14159        0  avgt    5   9,338 ± 0,391  ns/op
    power  3.14159        1  avgt    5  10,414 ± 0,075  ns/op
    power  3.14159        2  avgt    5   4,457 ± 0,002  ns/op
    power  3.14159  2.71828  avgt    5   9,567 ± 0,054  ns/op
```

# @Param

```
Benchmark         (a)       (b)  Mode  Cnt   Score    Error  Units
    power           1         0  avgt    5   7,703 ± 0,007  ns/op
    power           1         1  avgt    5  10,335 ± 0,026  ns/op
    power           1         2  avgt    5   1,904 ± 0,003  ns/op
    power           1   2.71828  avgt    5  11,000 ± 0,028  ns/op
    power           2         0  avgt    5   6,375 ± 0,007  ns/op
    power           2         1  avgt    5   9,559 ± 0,007  ns/op
    power           2         2  avgt    5   2,443 ± 0,458  ns/op
    power           2   2.71828  avgt    5   9,557 ± 0,017  ns/op
    power     3.14159         0  avgt    5   9,338 ± 0,391  ns/op
    power     3.14159         1  avgt    5  10,414 ± 0,075  ns/op
    power     3.14159         2  avgt    5   4,457 ± 0,002  ns/op
    power     3.14159   2.71828  avgt    5   9,567 ± 0,054  ns/op
```

# @Param

```
Benchmark        (a)      (b)  Mode  Cnt   Score    Error  Units
    power          1        0  avgt    5   7,703 ± 0,007  ns/op
    power          1        1  avgt    5  10,335 ± 0,026  ns/op
    power          1        2  avgt    5   1,904 ± 0,003  ns/op
    power          1  2.71828  avgt    5  11,000 ± 0,028  ns/op
    power          2        0  avgt    5   6,375 ± 0,007  ns/op
    power          2        1  avgt    5   9,559 ± 0,007  ns/op
    power          2        2  avgt    5   2,443 ± 0,458  ns/op
    power          2  2.71828  avgt    5   9,557 ± 0,017  ns/op
    power    3.14159        0  avgt    5   9,338 ± 0,391  ns/op
    power    3.14159        1  avgt    5  10,414 ± 0,075  ns/op
    power    3.14159        2  avgt    5   4,457 ± 0,002  ns/op
    power    3.14159  2.71828  avgt    5   9,567 ± 0,054  ns/op
```

https://habr.com/ru/companies/skbkontur/articles/351812/

САМЫЙ БЫСТРЬЙ POW
НА ДИКОМ ЗАПАДЕ
Math.pow(
x,2
x,y

# @Threads

# @Threads

```
@State(value = Scope.Thread)
public class ThreadsBenchmark {
    private AtomicInteger counter = new AtomicInteger();

    @Threads(1)
    @Benchmark
    public int measure_1() {
        return counter.incrementAndGet();
    }

    @Threads(2)
    @Benchmark
    public int measure_2() {
        return counter.incrementAndGet();
    }
}
```

# @Threads

```
@State(value = Scope.Thread)
public class ThreadsBenchmark {
    private AtomicInteger counter = new AtomicInteger();

    @Threads(1)
    @Benchmark
    public int measure_1() {
        return counter.incrementAndGet();
    }

    @Threads(2)
    @Benchmark
    public int measure_2() {
        return counter.incrementAndGet();
    }
}
```

# @Threads

```
@State(value = Scope.Thread)
public class ThreadsBenchmark {
    private AtomicInteger counter = new AtomicInteger();

    @Threads(1)
    @Benchmark
    public int measure_1() {
        return counter.incrementAndGet();
    }

    @Threads(2)
    @Benchmark
    public int measure_2() {
        return counter.incrementAndGet();
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Threads

```
@State(value = Scope.Benchmark)
public class ThreadsBenchmark {
    private AtomicInteger counter = new AtomicInteger();

    @Threads(1)
    @Benchmark
    public int measure_1() {
        return counter.incrementAndGet();
    }

    @Threads(2)
    @Benchmark
    public int measure_2() {
        return counter.incrementAndGet();
    }
}
```

⚠ https://github.com/gnkoshelev/jmh-crash-course

# @Threads

```
Benchmark                         Mode  Cnt   Score    Error  Units
baseline  (return 42)             avgt   15   1,553 ± 0,028  ns/op
1 thread  (non-shared)            avgt   15   2,092 ± 0,026  ns/op
2 threads (non-shared)            avgt   15   2,184 ± 0,042  ns/op
3 threads (non-shared)            avgt   15   2,225 ± 0,031  ns/op
4 threads (non-shared)            avgt   15   2,418 ± 0,025  ns/op
1 thread  (shared)                avgt   15   2,095 ± 0,047  ns/op
2 threads (shared)                avgt   15  17,144 ± 1,342  ns/op
3 threads (shared)                avgt   15  38,726 ± 0,950  ns/op
4 threads (shared)                avgt   15  49,483 ± 3,843  ns/op
```

# @Threads

```
Benchmark                         Mode   Cnt     Score    Error   Units
baseline   (return 42)            avgt    15     1,553 ± 0,028   ns/op
1 thread   (non-shared)           avgt    15     2,092 ± 0,026   ns/op
2 threads  (non-shared)           avgt    15     2,184 ± 0,042   ns/op
3 threads  (non-shared)           avgt    15     2,225 ± 0,031   ns/op
4 threads  (non-shared)           avgt    15     2,418 ± 0,025   ns/op
1 thread   (shared)               avgt    15     2,095 ± 0,047   ns/op
2 threads  (shared)               avgt    15    17,144 ± 1,342   ns/op
3 threads  (shared)               avgt    15    38,726 ± 0,950   ns/op
4 threads  (shared)               avgt    15    49,483 ± 3,843   ns/op
```

# @Threads

```
Benchmark                          Mode  Cnt   Score    Error  Units
baseline  (return 42)              avgt   15   1,553 ± 0,028  ns/op
1 thread  (non-shared)             avgt   15   2,092 ± 0,026  ns/op
2 threads (non-shared)             avgt   15   2,184 ± 0,042  ns/op
3 threads (non-shared)             avgt   15   2,225 ± 0,031  ns/op
4 threads (non-shared)             avgt   15   2,418 ± 0,025  ns/op
1 thread  (shared)                 avgt   15   2,095 ± 0,047  ns/op
2 threads (shared)                 avgt   15  17,144 ± 1,342  ns/op
3 threads (shared)                 avgt   15  38,726 ± 0,950  ns/op
4 threads (shared)                 avgt   15  49,483 ± 3,843  ns/op
```

# @Threads

```
Benchmark                     Mode  Cnt   Score    Error  Units
baseline  (return 42)         avgt   15   1,553 ± 0,028  ns/op
1 thread  (non-shared)        avgt   15   2,092 ± 0,026  ns/op
2 threads (non-shared)        avgt   15   2,184 ± 0,042  ns/op
3 threads (non-shared)        avgt   15   2,225 ± 0,031  ns/op
4 threads (non-shared)        avgt   15   2,418 ± 0,025  ns/op
1 thread  (shared)            avgt   15   2,095 ± 0,047  ns/op
2 threads (shared)            avgt   15  17,144 ± 1,342  ns/op
3 threads (shared)            avgt   15  38,726 ± 0,950  ns/op
4 threads (shared)            avgt   15  49,483 ± 3,843  ns/op
```

# @Threads

```
Benchmark                      Mode  Cnt   Score    Error  Units
baseline  (return 42)          avgt   15   1,553 ± 0,028  ns/op
1 thread  (non-shared)         avgt   15   2,092 ± 0,026  ns/op
2 threads (non-shared)         avgt   15   2,184 ± 0,042  ns/op
3 threads (non-shared)         avgt   15   2,225 ± 0,031  ns/op
4 threads (non-shared)         avgt   15   2,418 ± 0,025  ns/op
1 thread  (shared)             avgt   15   2,095 ± 0,047  ns/op
2 threads (shared)             avgt   15  17,144 ± 1,342  ns/op
3 threads (shared)             avgt   15  38,726 ± 0,950  ns/op
4 threads (shared)             avgt   15  49,483 ± 3,843  ns/op
```

# @Threads

```
Benchmark                      Mode  Cnt   Score    Error  Units
baseline  (return 42)          avgt   15   1,553 ± 0,028  ns/op
1 thread  (non-shared)         avgt   15   2,092 ± 0,026  ns/op
2 threads (non-shared)         avgt   15   2,184 ± 0,042  ns/op
3 threads (non-shared)         avgt   15   2,225 ± 0,031  ns/op
4 threads (non-shared)         avgt   15   2,418 ± 0,025  ns/op
1 thread  (shared)             avgt   15   2,095 ± 0,047  ns/op
2 threads (shared)             avgt   15  17,144 ± 1,342  ns/op
3 threads (shared)             avgt   15  38,726 ± 0,950  ns/op
4 threads (shared)             avgt   15  49,483 ± 3,843  ns/op
```

# @Threads

```
Benchmark                    Mode  Cnt   Score   Error  Units
baseline  (return 42)        avgt   15   1,553 ± 0,028  ns/op
1 thread  (non-shared)       avgt   15   2,092 ± 0,026  ns/op
2 threads (non-shared)       avgt   15   2,184 ± 0,042  ns/op
3 threads (non-shared)       avgt   15   2,225 ± 0,031  ns/op
4 threads (non-shared)       avgt   15   2,418 ± 0,025  ns/op
1 thread  (shared)           avgt   15   2,095 ± 0,047  ns/op
2 threads (shared)           avgt   15  17,144 ± 1,342  ns/op
3 threads (shared)           avgt   15  38,726 ± 0,950  ns/op
4 threads (shared)           avgt   15  49,483 ± 3,843  ns/op
```

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score           Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

```
Result "ru.jpoint2025.MyBenchmark.testMethod":
  3555831120,759 ±(99.9%) 31620380,966 ops/s [Average]
  (min, avg, max) = (3454572269,124, 3555831120,759, 3613755629,883),
       stdev = 42212303,923
  CI (99.9%): [3524210739,794, 3587451501,725] (assumes normal distribution)

REMEMBER: The numbers below are just data...

# Run complete. Total time: 00:08:46

Benchmark                Mode  Cnt           Score          Error  Units
MyBenchmark.testMethod  thrpt   25  3555831120,759 ± 31620380,966  ops/s
```

# Профилирование

# Профилирование

```
$ java -jar target/benchmarks.jar -lprof

$ java -jar target/benchmarks.jar -prof <profiler>
```

# Профилирование

```
$ java -jar target/benchmarks.jar -lprof

$ java -jar target/benchmarks.jar -prof <profiler>
```

# Профилирование

```
cl: Classloader profiling via standard MBeans
comp: JIT compiler profiling via standard MBeans
gc: GC profiling via standard MBeans
jfr: Java Flight Recorder profiler
mempool: Memory pool/footprint profiling via standard MBeans
pauses: Pauses profiler
safepoints: Safepoints profiler
stack: Simple and naive Java stack profiler

async: async-profiler profiler provider
perf / perfc2c / perfasm / perfnorm: Linux perf
xperfasm: Windows perf + PrintAssembly profiler
dtraceasm: DTrace profile provider + PrintAssembly profiler
```

# Профилирование

```
cl: Classloader profiling via standard MBeans
comp: JIT compiler profiling via standard MBeans
gc: GC profiling via standard MBeans
jfr: Java Flight Recorder profiler
mempool: Memory pool/footprint profiling via standard MBeans
pauses: Pauses profiler
safepoints: Safepoints profiler
stack: Simple and naive Java stack profiler

async: async-profiler profiler provider
perf / perfc2c / perfasm / perfnorm: Linux perf
xperfasm: Windows perf + PrintAssembly profiler
dtraceasm: DTrace profile provider + PrintAssembly profiler
```

# Профилирование

```
// ...
@State(Scope.Thread)
public static class Maps {
    private Map<Integer, Integer> map;

    private int begin, end;

    @Setup
    public void setup() {
        map = new HashMap<>();
        begin = 1; end = 256;
        for (int i = begin; i < end; i++) {
            map.put(i, i);
        }
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование

```
// ...
@State(Scope.Thread)
public static class Maps {
    private Map<Integer, Integer> map;

    private int begin, end;

    @Setup
    public void setup() {
        map = new HashMap<>();
        begin = 1; end = 256;
        for (int i = begin; i < end; i++) {
            map.put(i, i);
        }
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование

```java
@Benchmark
public void test(Blackhole bh) {
    for (int i = begin; i < end; i++) {
        bh.consume(map.get(i));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование

```
@Benchmark
public void test(Blackhole bh) {
    for (int i = begin; i < end; i++) {
        bh.consume(map.get(i));
    }
}
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование

```java
@Benchmark
public void test(Blackhole bh) {
    for (int i = begin; i < end; i++) {
        bh.consume(map.get(i));
    }
}
```

```
$ java -jar target/benchmarks.jar JMHSample_35.Maps -prof gc
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование

```
Benchmark                             Cnt     Score     Error   Units
test:                                   5  1553.201 ±   6.199   ns/op
test:gc.alloc.rate                      5  1257.046 ±   5.675  MB/sec
test:gc.alloc.rate.norm                 5  2048.001 ±   0.001    B/op
test:gc.churn.PS_Eden_Space             5  1259.148 ± 315.277  MB/sec
test:gc.churn.PS_Eden_Space.norm        5  2051.519 ± 520.324    B/op
test:gc.churn.PS_Survivor_Space         5     0.175 ±   0.386  MB/sec
test:gc.churn.PS_Survivor_Space.norm    5     0.285 ±   0.629    B/op
test:gc.count                           5    29.000            counts
test:gc.time                            5    16.000                ms
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование

```
Benchmark                           Cnt     Score     Error  Units
test:                                 5  1553.201 ±   6.199  ns/op
test:gc.alloc.rate                    5  1257.046 ±   5.675 MB/sec
test:gc.alloc.rate.norm               5  2048.001 ±   0.001   B/op
test:gc.churn.PS_Eden_Space           5  1259.148 ± 315.277 MB/sec
test:gc.churn.PS_Eden_Space.norm      5  2051.519 ± 520.324   B/op
test:gc.churn.PS_Survivor_Space       5     0.175 ±   0.386 MB/sec
test:gc.churn.PS_Survivor_Space.norm  5     0.285 ±   0.629   B/op
test:gc.count                         5    29.000           counts
test:gc.time                          5    16.000               ms
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование — Linux perf

```
cl: Classloader profiling via standard MBeans
comp: JIT compiler profiling via standard MBeans
gc: GC profiling via standard MBeans
jfr: Java Flight Recorder profiler
mempool: Memory pool/footprint profiling via standard MBeans
pauses: Pauses profiler
safepoints: Safepoints profiler
stack: Simple and naive Java stack profiler

async: async-profiler profiler provider
perf / perfc2c / perfasm / perfnorm: Linux perf
xperfasm: Windows perf + PrintAssembly profiler
dtraceasm: DTrace profile provider + PrintAssembly profiler
```

# Профилирование — Linux perf

```
$ java -jar target/benchmarks.jar JMHSample_37 -prof perfnorm
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_37_Profilers.java

# Профилирование — Linux perf

```
Benchmark                          Mode  Cnt   Score    Error  Units
colFirst                           avgt   25   5.306 ±  0.020  ns/op
colFirst:CPI                       avgt    5   0.621 ±  0.011   #/op
colFirst:L1-dcache-load-misses     avgt    5   2.177 ±  0.044   #/op
colFirst:L1-dcache-loads           avgt    5  14.804 ±  0.261   #/op
colFirst:LLC-loads                 avgt    5   2.165 ±  0.091   #/op
colFirst:cycles                    avgt    5  22.272 ±  0.372   #/op
colFirst:instructions              avgt    5  35.888 ±  1.215   #/op
```

```
$ java -jar target/benchmarks.jar JMHSample_37 -prof perfnorm
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_37_Profilers.java

# Профилирование — Linux perf

```
Benchmark                           Mode  Cnt   Score    Error  Units
colFirst                            avgt   25   5.306 ±  0.020  ns/op
colFirst:CPI                        avgt    5   0.621 ±  0.011   #/op
colFirst:L1-dcache-load-misses      avgt    5   2.177 ±  0.044   #/op
colFirst:L1-dcache-loads            avgt    5  14.804 ±  0.261   #/op
colFirst:LLC-loads                  avgt    5   2.165 ±  0.091   #/op
colFirst:cycles                     avgt    5  22.272 ±  0.372   #/op
colFirst:instructions               avgt    5  35.888 ±  1.215   #/op
```

```
$ java -jar target/benchmarks.jar JMHSample_37 -prof perfnorm
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_37_Profilers.java

# Профилирование — async-profiler

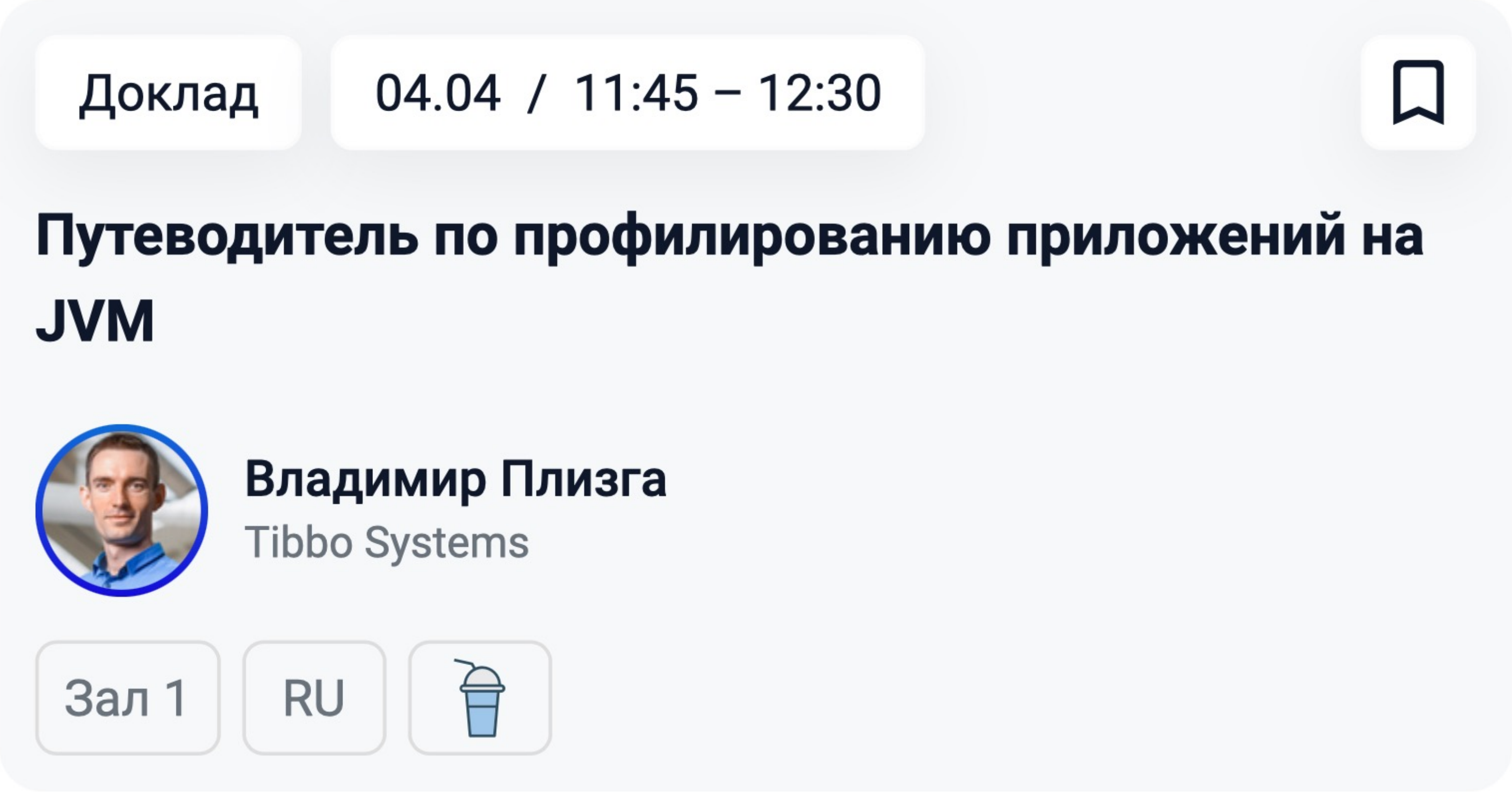
```
cl: Classloader profiling via standard MBeans
comp: JIT compiler profiling via standard MBeans
gc: GC profiling via standard MBeans
jfr: Java Flight Recorder profiler
mempool: Memory pool/footprint profiling via standard MBeans
pauses: Pauses profiler
safepoints: Safepoints profiler
stack: Simple and naive Java stack profiler

async: async-profiler profiler provider
perf / perfc2c / perfasm / perfnorm: Linux perf
xperfasm: Windows perf + PrintAssembly profiler
dtraceasm: DTrace profile provider + PrintAssembly profiler
```

# Профилирование — async-profiler

```
$ java -jar target/benchmarks.jar JMHSample_35_Profilers \
       -prof async:libPath=/path/to/libasyncProfiler.dylib
```

https://github.com/openjdk/jmh/blob/master/jmh-samples/src/main/java/org/openjdk/jmh/samples/JMHSample_35_Profilers.java

# Профилирование — async-profiler



Доклад

04.04 / 11:45 – 12:30

**Путеводитель по профилированию приложений на JVM**

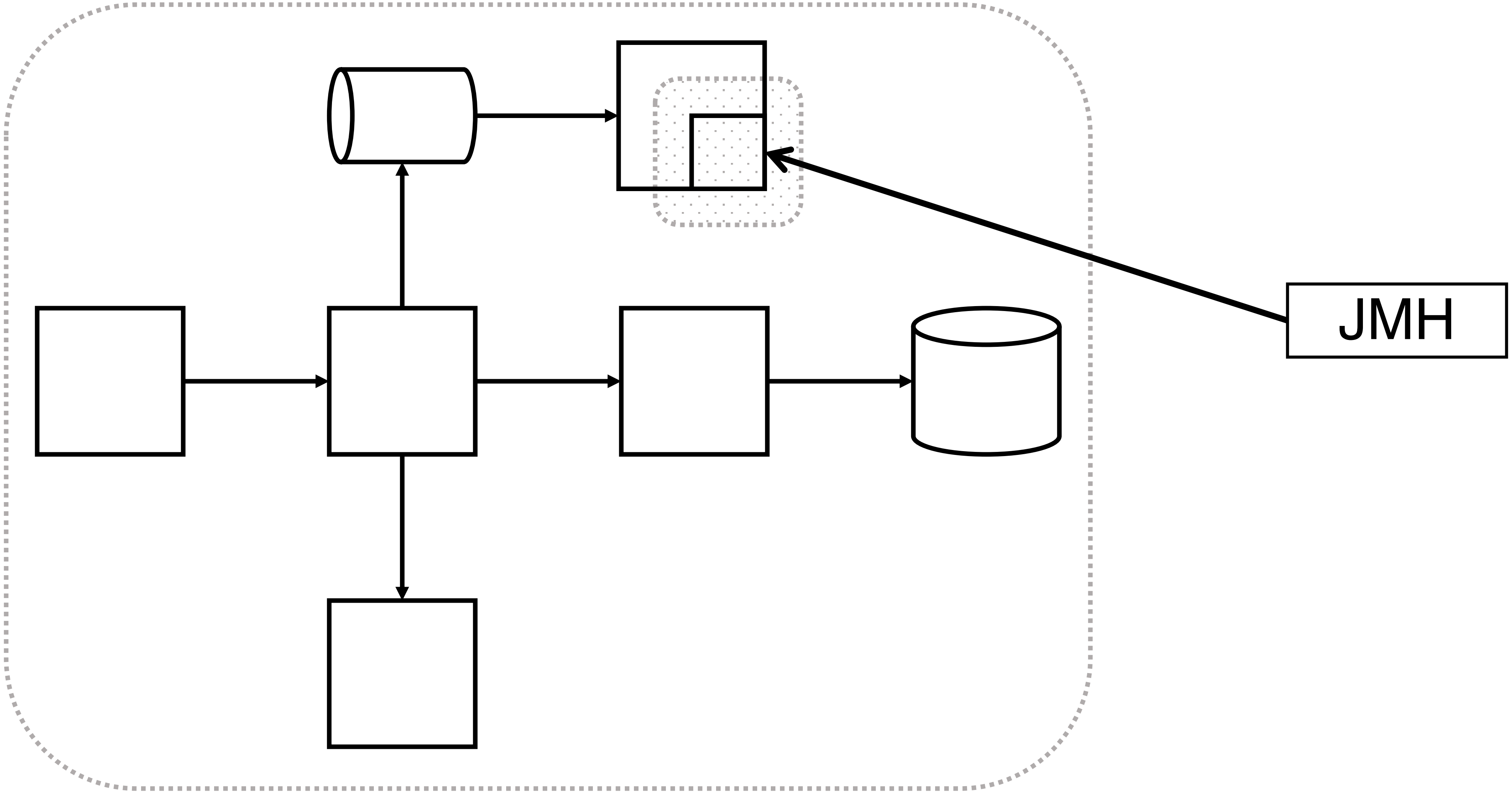
**Владимир Плизга**

Tibbo Systems

Зал 1

RU

https://jpoint.ru/talks/8f6a60900d8e488f98b81333137a093e/


JMH

# Гигиенический минимум «6 аннотаций»

```java
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Гигиенический минимум «6 аннотаций»

```
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Гигиенический минимум «6 аннотаций»

```
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Гигиенический минимум «6 аннотаций»

```
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Гигиенический минимум «6 аннотаций»

```
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Гигиенический минимум «6 аннотаций»

```
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Гигиенический минимум «6 аннотаций»

```
@Fork(value = 3)
@Warmup(iterations = 3, time = 5, timeUnit = SECONDS)
@Measurement(iterations = 5, time = 5, timeUnit = SECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
public class UuidBenchmark
```

# Путь Java-инженера

# Путь Java-инженера

1. Бенчмарки не нужны!

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

2. Бенчмарки — это легко!

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

2. Бенчмарки — это легко!

-- крутим настройки --

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

2. Бенчмарки — это легко!

-- крутим настройки --

-- профилируем --

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

2. Бенчмарки — это легко!

-- крутим настройки --

-- профилируем --

3. Бенчмарки — это сложно!

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

2. Бенчмарки — это легко!

-- крутим настройки --

-- профилируем --

3. Бенчмарки — это сложно!

-- познаём дзен JMH --

# Путь Java-инженера

1. Бенчмарки не нужны!

-- пробуем JMH --

2. Бенчмарки — это легко!

-- крутим настройки --

-- профилируем --

3. Бенчмарки — это сложно!

-- познаём дзен JMH --

4. Поздравляю, вы — Шипилёв!

# Выводы

# Выводы

1. **+1 инструмент** в вашу копилку

# Выводы

1. **+1 инструмент** в вашу копилку
2. Гигиенический минимум «**6 аннотаций**»

```
@Fork  @Warmup      @BenchmarkMode
@State @Measurement @OutputTimeUnit
```

# Выводы

1. **+1 инструмент** в вашу копилку
2. Гигиенический минимум «**6 аннотаций**»
3. jmh-crash-course

⚠ https://github.com/gnkoshelev/jmh-crash-course

# Выводы

1. **+1 инструмент** в вашу копилку
2. Гигиенический минимум «**6 аннотаций**»
3. jmh-crash-course + JMHSamples

https://github.com/openjdk/jmh/blob/master/jmh-samples

# Выводы

1. **+1 инструмент** в вашу копилку
2. Гигиенический минимум «**6 аннотаций**»
3. jmh-crash-course + JMHSamples
4. Следуй за ~~Бел~~ Шипилёвым

# Q/A

Другие доклады и материалы:

https://tg.me/chnl_GregoryKoshelev