Benchmarking, Statistics & PHPBench

A framework for creating glorious benchmarks

by Daniel Leech

@dantleech @phpbench

About Me

- Daniel Leech
- PHP Developer
  - Symfony CMF and associated projects
    - PHPCR-ODM, Jackalope, PHPCR
  - Sulu CMF
  - PHPBench
- Cycle tourer

Symfony CMF doctrine SULU PHPCR

What is Benchmarking?

STANDARDIZED

COMPARATIVE

PERFORMANCE

TESTING

SCPT

- **Testing**: Running an operation and obtaining a result.
- **Performance**: We are concerned about the performance of the subject.
- **Comparative**: Ultimately, we are always concerned about the performance of one thing as compared to another thing.
- **Standardized**: The tests should be standardized so that they can be applied meaningfully to all subjects.

Differences to Profiling

	Benchmarking	**Profiling**
Measures	External wall time	Internal call times
Observer effect?	Minimal	Major
Scope (typical)	Scenario	Request
Applies to (typical)	Libraries	Applications

Benchmarking Script

http://stackoverflow.com/questions/13483219/what-is-faster-in-array-or-isset

```php
<?php

$a = array();
$start = microtime( true );
for ($i = 0; $i < 10000; ++$i) {
    isset($a['key']);
}
$total_time = microtime( true ) - $start;
echo "Total time: ", number_format($total_time, 6), PHP_EOL;
$start = microtime( true );
for ($i = 0; $i < 10000; ++$i) {
    in_array('key', $a);
}
$total_time = microtime( true ) - $start;
echo "Total time: ", number_format($total_time, 6),     EOL;
```

found in the wild

Issues

- Boilerplate code.
- Relies on single samples.
- Results are not persisted.
- Not scalable.

Benchmarking Framework Should

- Execute code units.
- Perform tests multiple times to verify results.
- Provide visual feedback.
- Perform statistical analysis.
- Be able to serialize and store results.

Concepts

Revolutions

Number of times we consecutively execute our benchmark subject.

```php
<?php

$revolutions = 10000;
for ($i =0; $i < $revolutions; $i++) {
    // do something
}
```

Iterations

Each iteration records the time taken to execute the revolutions.

```
<?php

$iterations = 100;
$revolutions = 10000;
$results = [];

for ($iteration = 0; $iteration < $iterations; $iteration++) {
    $startTime = microtime(true);
    for ($rev = 0; $rev < $revolultions; $rev++) {
        // do something
    }
    $time = microtime(true) - $startTime;
    $times[] = $time;
}
```

illustrative example

Subjects

The actual code we are benchmarking.

Subjects could be:

- PHP internal functions.
- User functions (think unit tests).
- User services (think functional tests).
- External services.
- Anything you can wrap in a function.

Subjects

```php
<?php

isset($a['b']);
```

Algorithms

```php
<?php

$pdoConnection->query(“SELECT * FROM foo”);
```

External services

```php
<?php

$container->get('some.service');
```

User libraries

```php
<?php

$guzzle->get('https://github.com/phpbench/phpbench');
```

HTTP

Benchmarking with PHPBench

PHPBench is a command line based benchmarking tool similar to PHPUniut.

```
$ phpbench
phpbench version 0.10.0-dev

Usage:
  command [options] [arguments]

...

Available commands:
  help    Displays help for a command
  list    Lists commands
  report  Generate a report from an XML file
  run     Run benchmarks
```

Benchmark Class

```php
<?php

class HashBench
{
    public function benchHash()
    {
        md5('hello world');
    }
}
```

HashBench.php

actual working example!

- Class name suffixed with “Bench”.
- Subject prefixed with “bench”.
- Does not know about PHPBench.

Benchmark Class

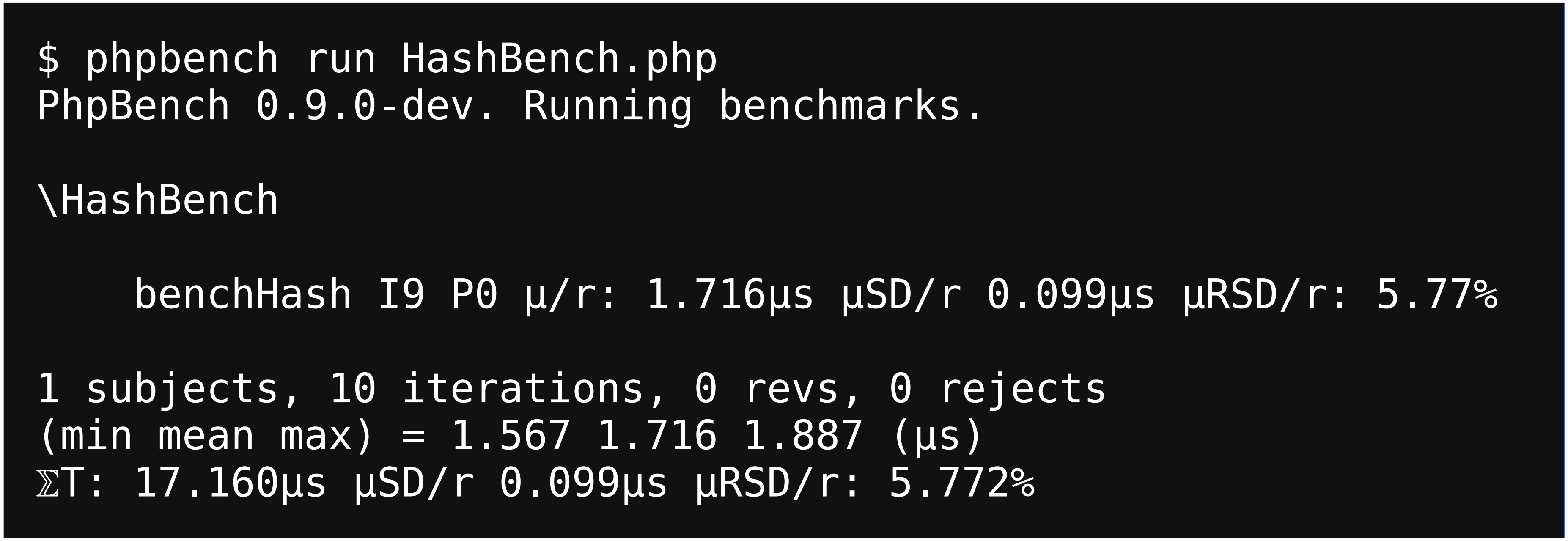
```php
<?php

class HashBench
{
    /**
     * @Revs(10000)
     * @Iterations(100)
     */
    public function benchHash()
    {
        md5('hello world');
    }
}
```

HashBench.php

- Annotations determine how the subject is executed.
- Class and/or method level.
- Aware of class inheritance.

Running Benchmarks



```
$ phpbench run HashBench.php
PhpBench 0.9.0-dev. Running benchmarks.

\HashBench

    benchHash I9 P0 μ/r: 1.716μs μSD/r 0.099μs μRSD/r: 5.77%

1 subjects, 10 iterations, 0 revs, 0 rejects
(min mean max) = 1.567 1.716 1.887 (μs)
∑T: 17.160μs μSD/r 0.099μs μRSD/r: 5.772%
```

I: Iteration #

P: Parameter set #

/r: Times are divided by number of revolutions.

μ: Mean (average).

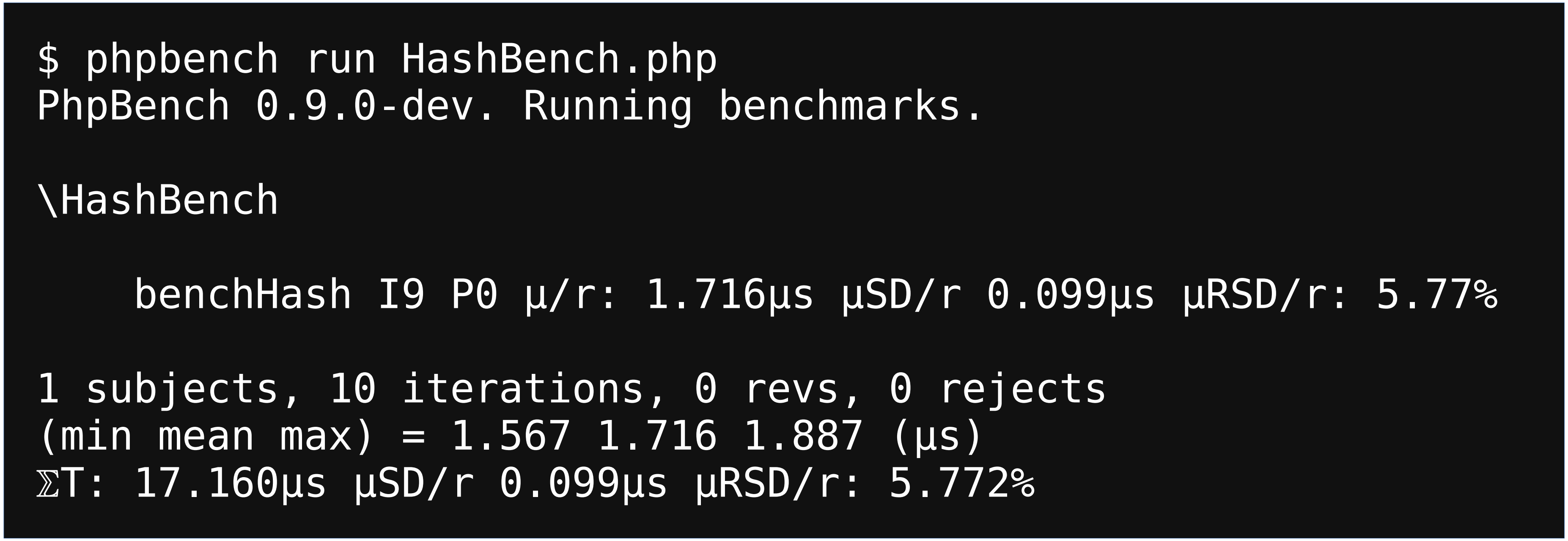
μs: Microseconds.

∑T: Total time.

SD: Standard deviation.

RSD: Relative standard deviation.

Running Benchmarks



```
$ phpbench run HashBench.php
PhpBench 0.9.0-dev. Running benchmarks.

\HashBench

    benchHash I9 P0 μ/r: 1.716μs μSD/r 0.099μs μRSD/r: 5.77%

1 subjects, 10 iterations, 0 revs, 0 rejects
(min mean max) = 1.567 1.716 1.887 (μs)
∑T: 17.160μs μSD/r 0.099μs μRSD/r: 5.772%
```

- Progress loggers provide realtime feedback.
- Specified with --progress
- Reports can also be generated.

Stability and Accuracy

Revolutions

How do revolutions affect time?

```php
<?php

$revolutions = 10000;
$start = microtime(true);
for ($i =0; $i < $revolutions; $i++) {
    md5('hello world');
}
$elapsed = microtime(true) - $start;
```

Revolutions

Reported time is total time divided by number of revolutions.

```php
<?php

$time = $iterationTime / $revolutions;
```

Answers the question:

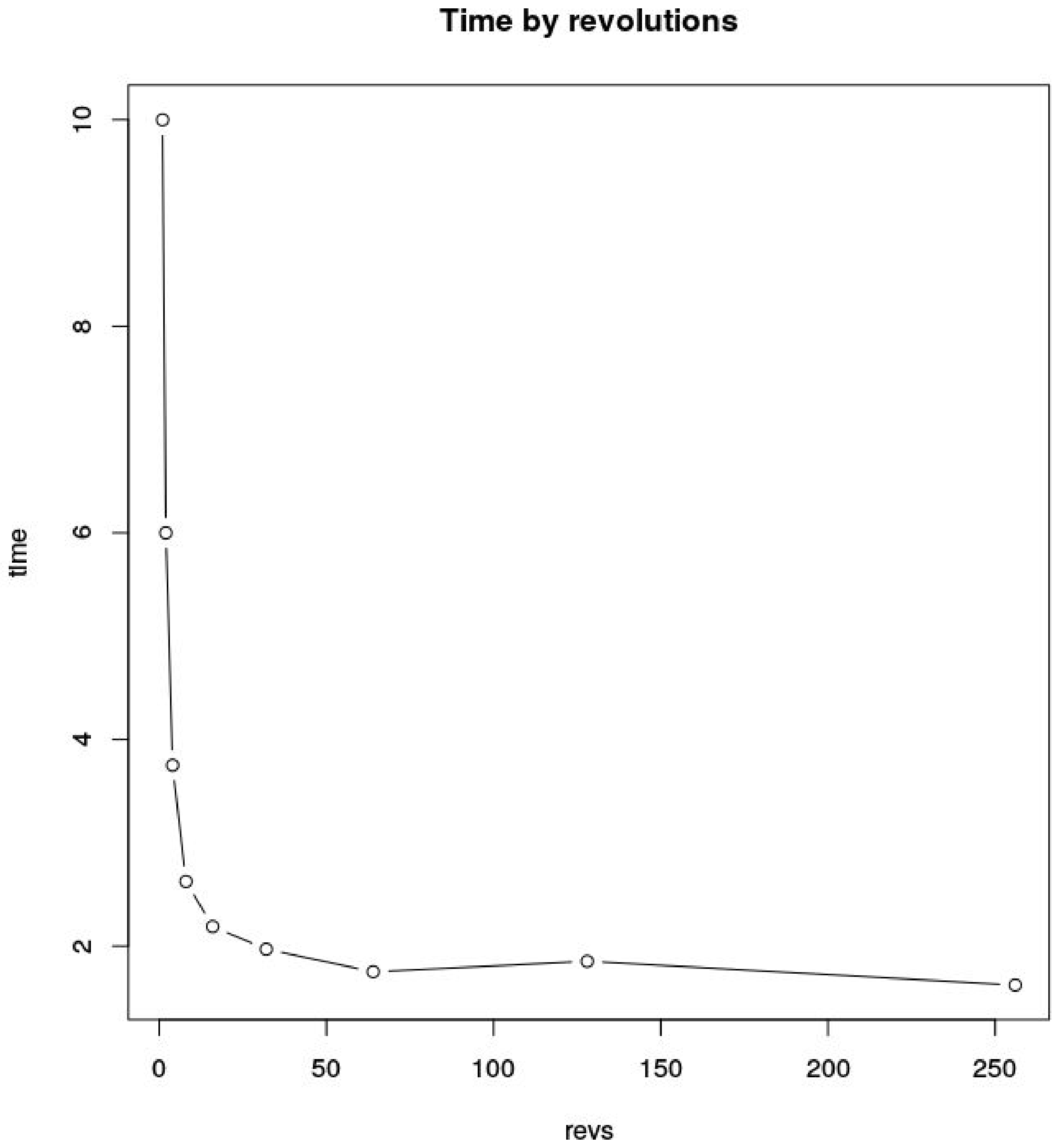
What is the average execution time of the method?

Micro Revolutionary Effect

Micro (microseconds) benchmarks are heavily affected by the number of revolutions.

Revolutions	Time (μs/r)	
1	10	100%
2	6	60%
4	3.75	37.5%
8	2.625	26.2%
16	2.188	21.8%
32	1.969	19.6%
64	1.75	17.5%
10,000	1.72	17.2%

- Includes microtime and loop calls.
- Begins to stabilise at ~32 revolutions.
- Regression to ~17% of first value.

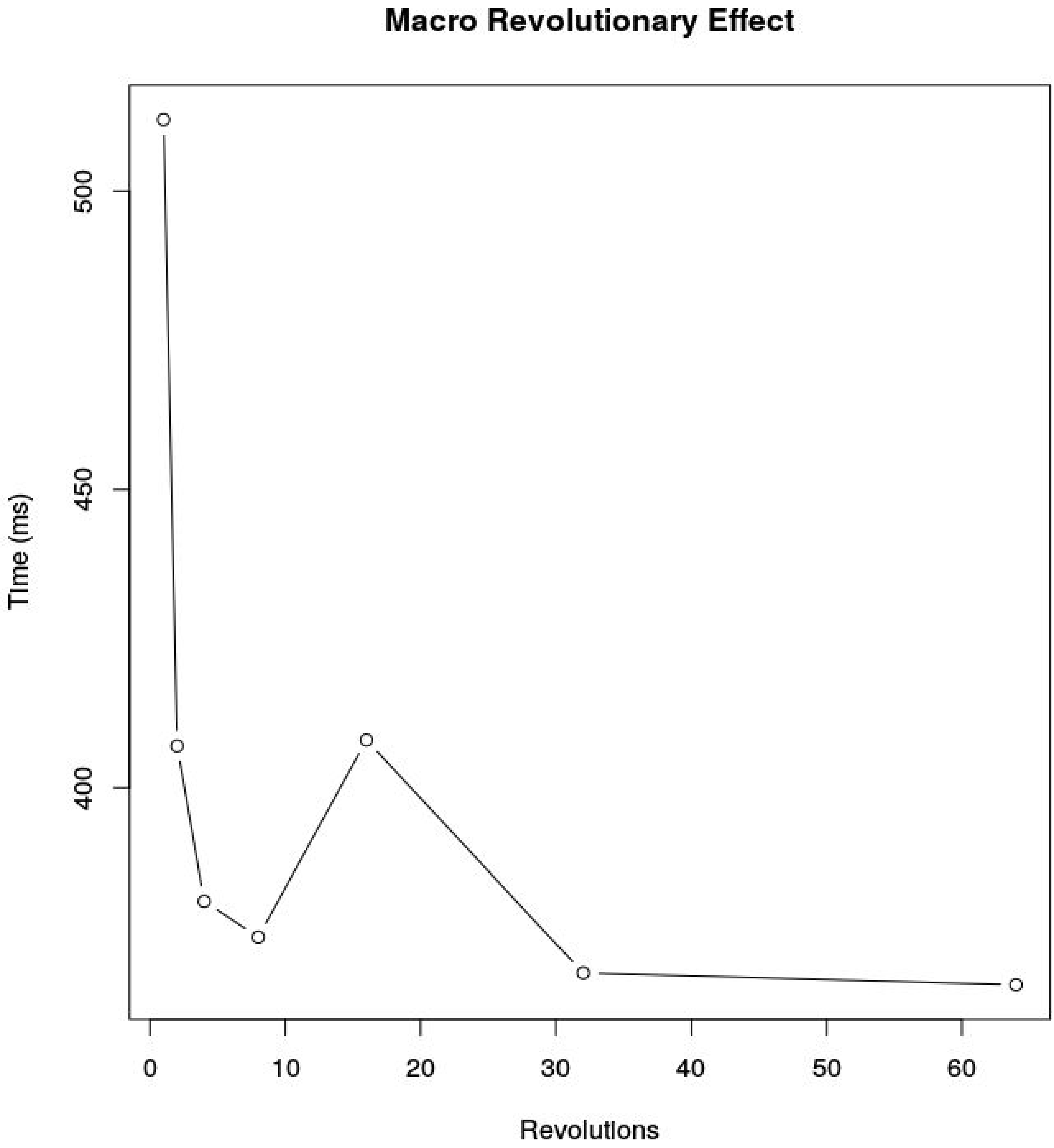

Time by revolutions
time
revs
10
8
6
4
2
0
50
100
150
200
250

Macro Revolutionary Effect

Macro (milliseconds, seconds) benchmarks less affected.

Revolutions	**Time (ms/r)**	
1	512	100%
2	407	79%
4	381	74%
8	375	73%
16	408	79%
32	369	72%
64	367	71%

- Warmup factor (autoload cache, etc)
- Earlier stabilisation.
- Regression to 71% of first value.


Macro Revolutionary Effect
Time (ms)
500
450
400
0
10
20
30
40
50
60
Revolutions

Iterations

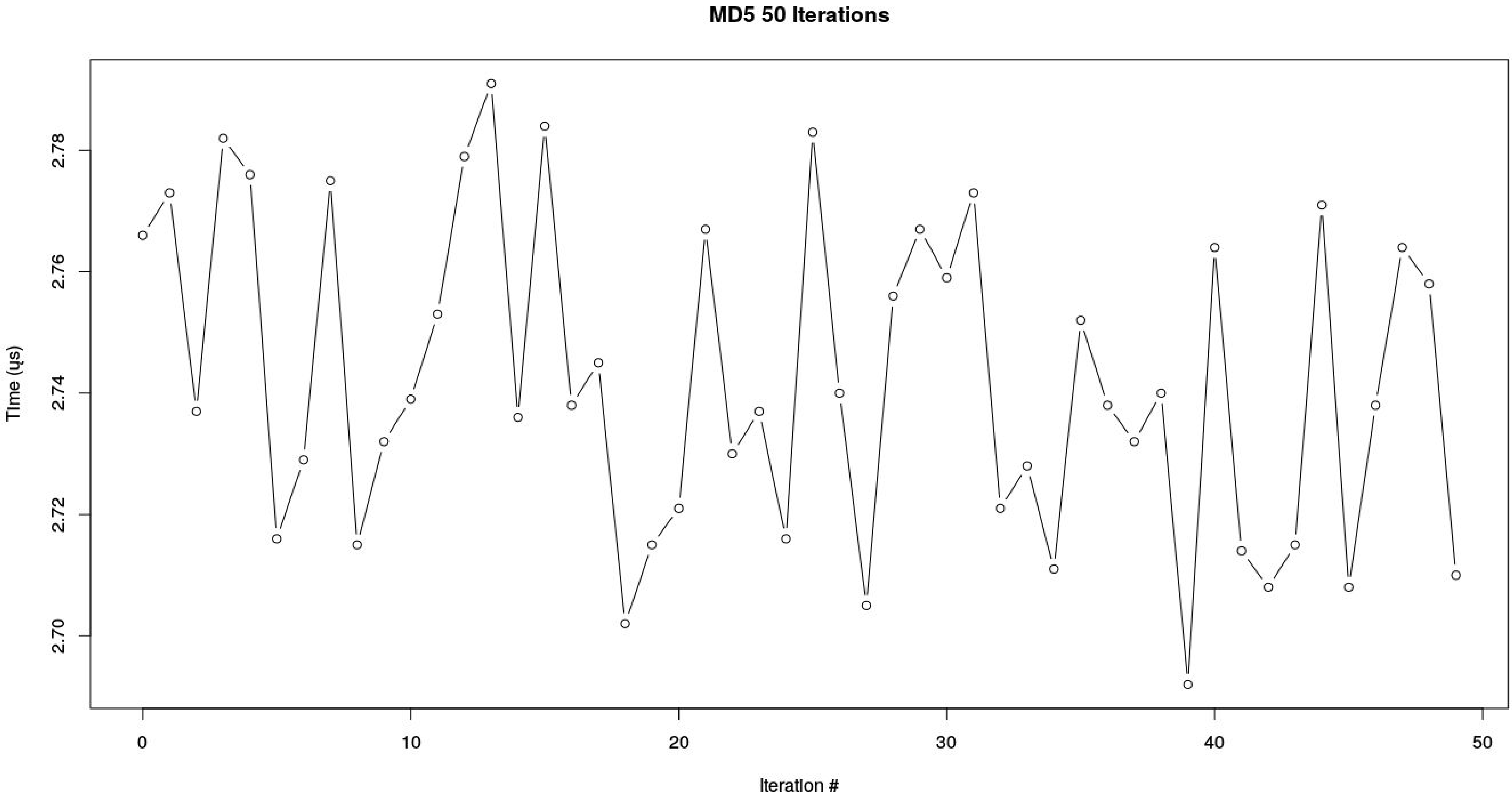
- Each iteration provides a time measurement.
- Iterations produce a range of different time measurements.

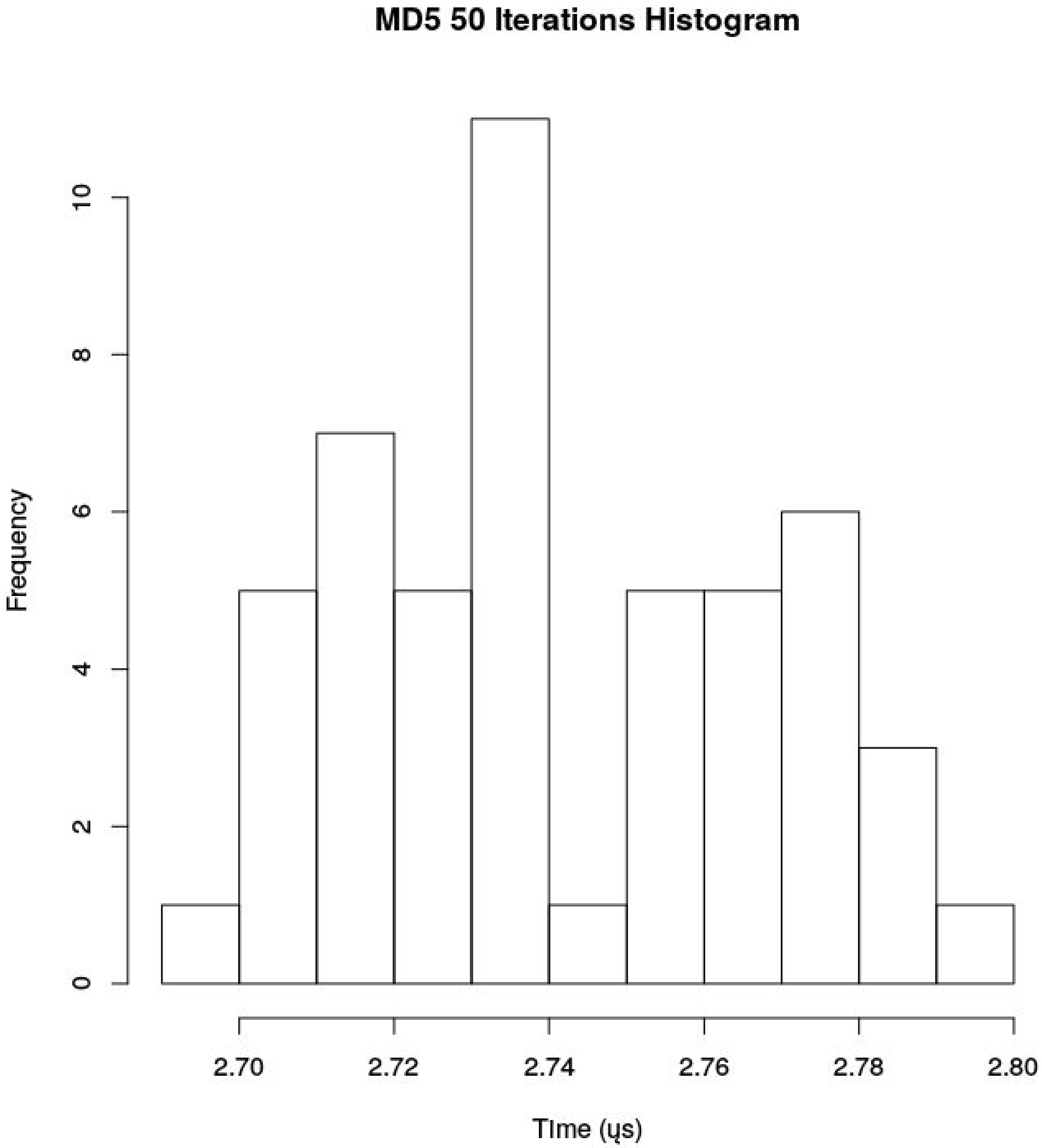
```php
<?php

class HashBench
{
    /**
     * @Revs(10000)
     * @Iterations(100)
     */
    public function benchHash()
    {
        md5('hello world');
    }
}
```

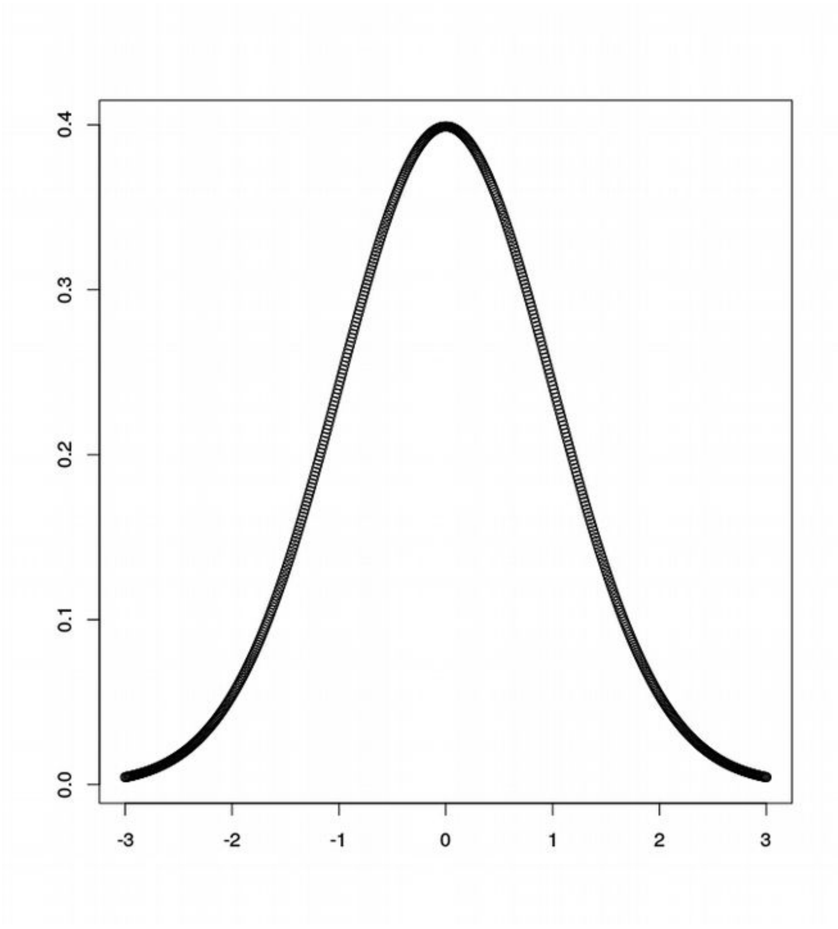
Iterations

Iteration	μ Time (μs)
1	2.766
2	2.773
3	2.737
4	2.782
5	2.776
6	2.716
7	2.729
..	..

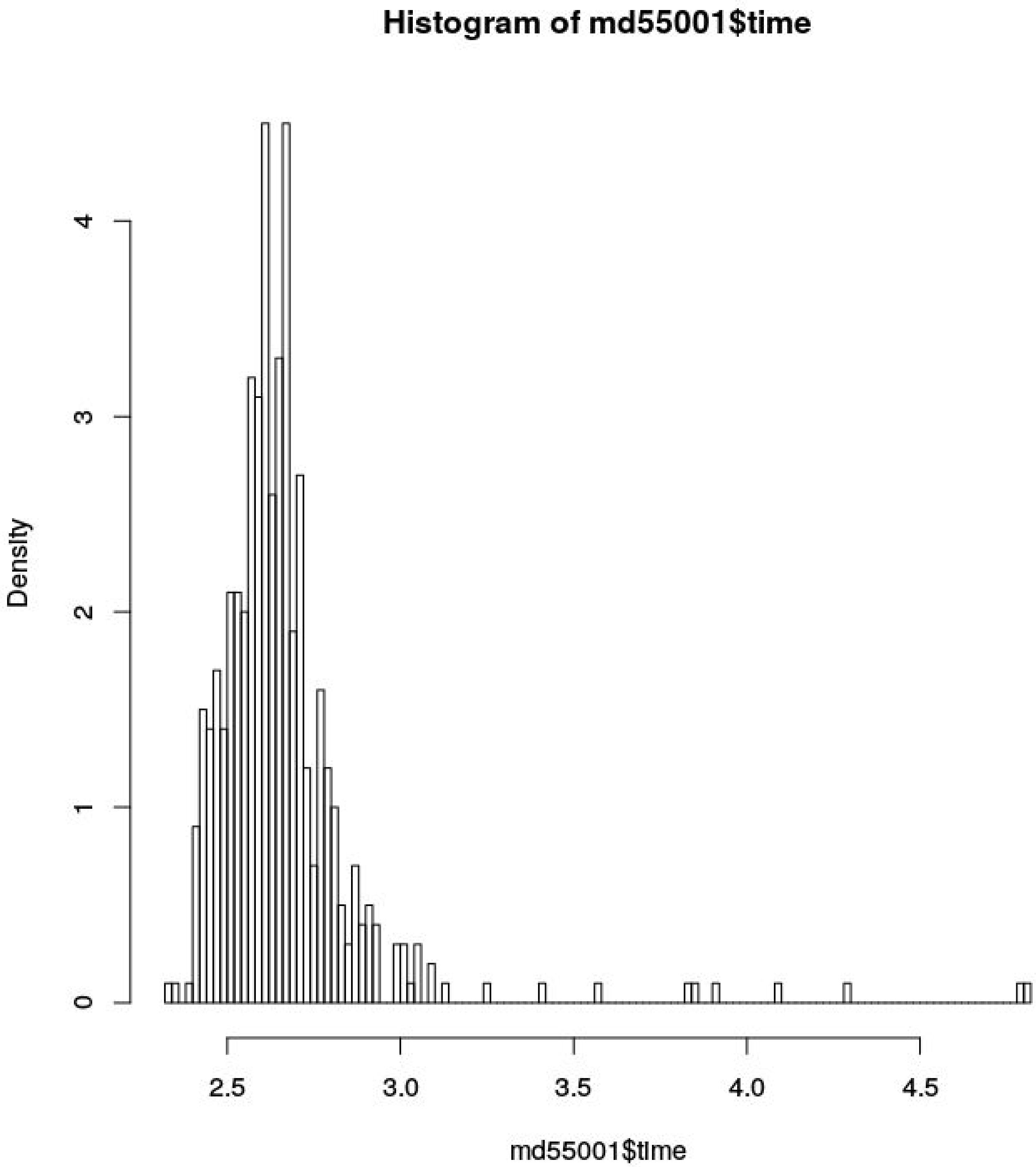

MD5 50 Iterations
Time (μs)
2.70
2.72
2.74
2.76
2.78
Iteration #
0
10
20
30
40
50

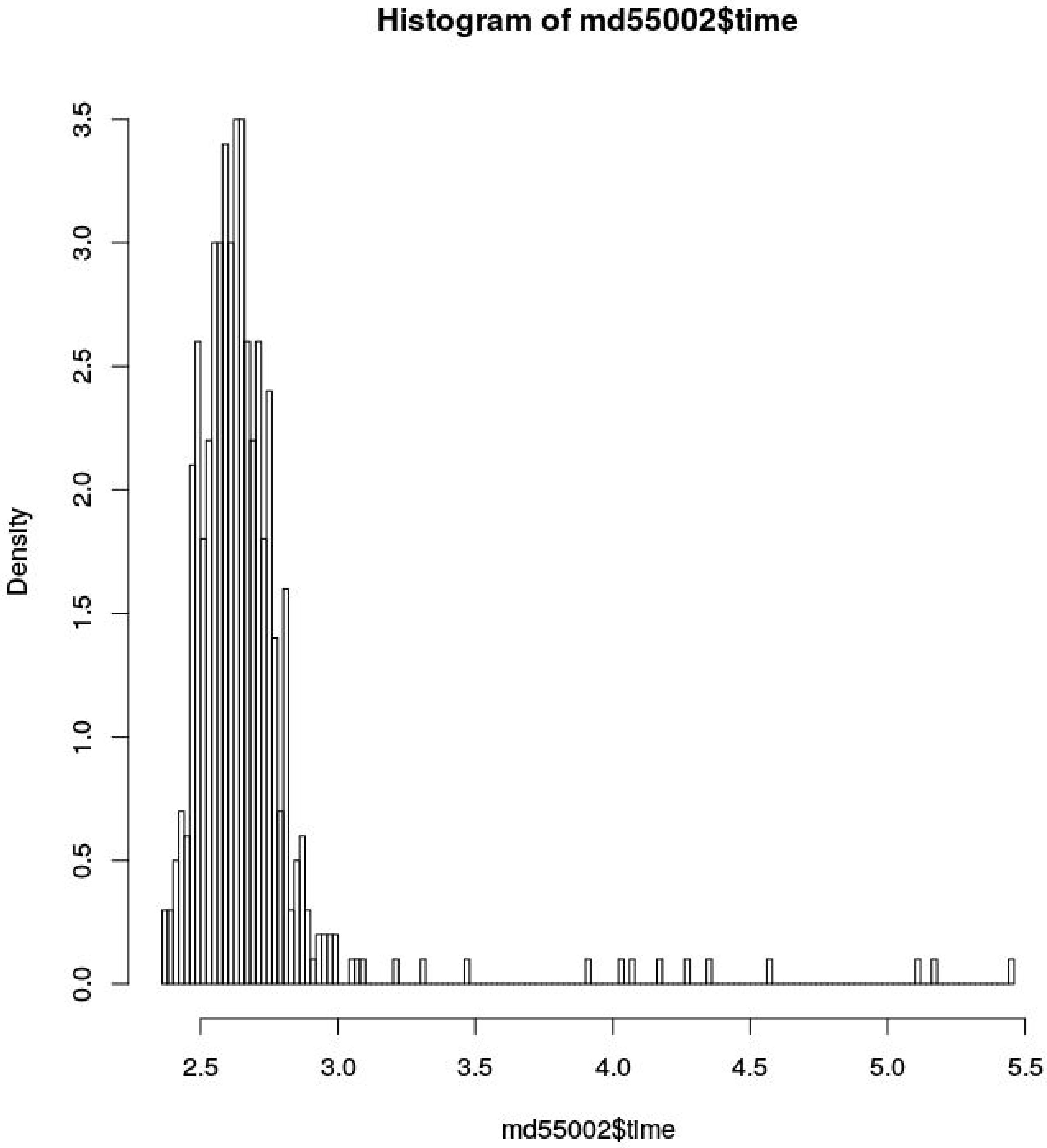

MD5 50 Iterations Histogram
Frequency
0
2
4
6
8
10
2.70
2.72
2.74
2.76
2.78
2.80
Tıme (ųs)

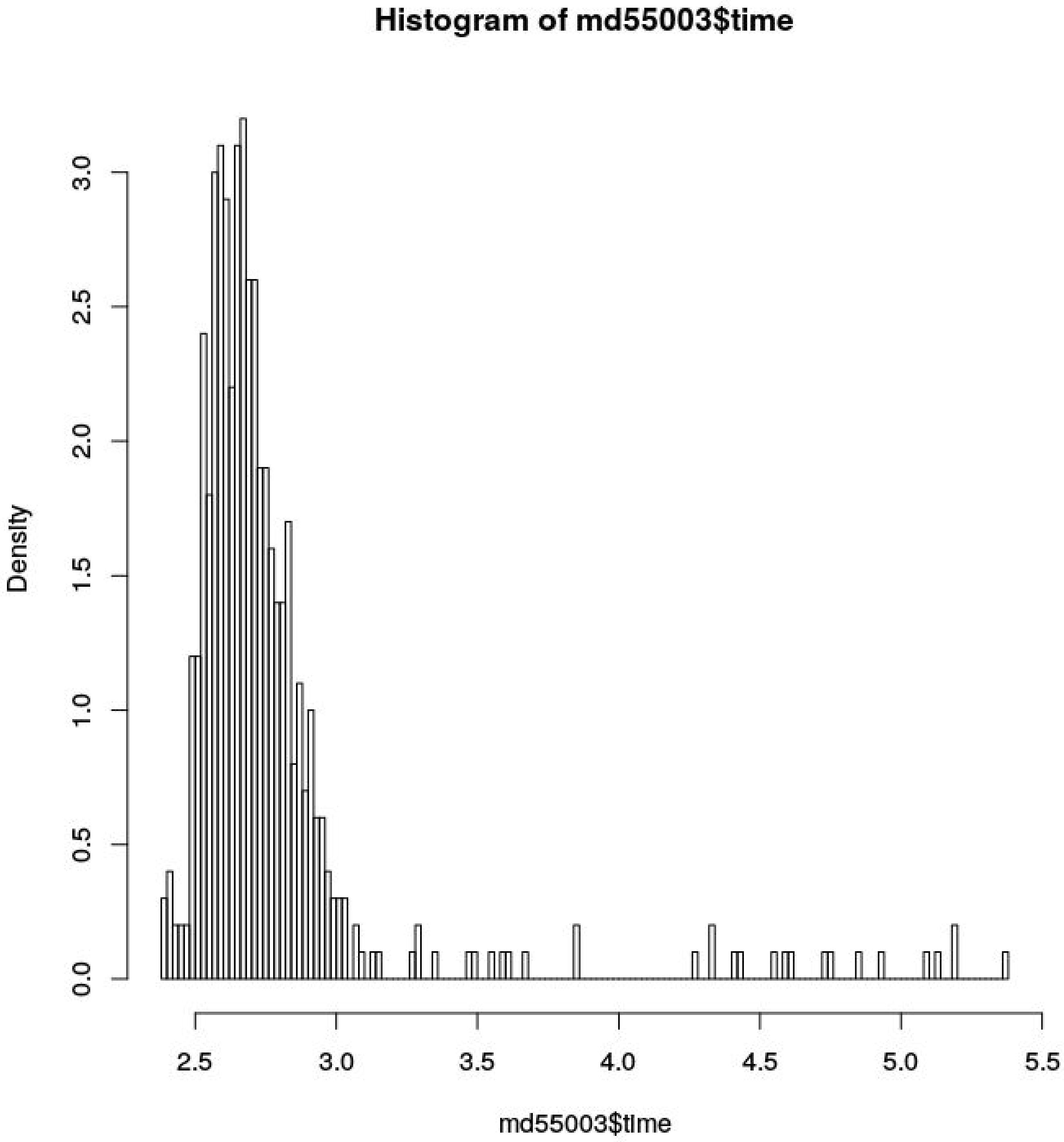
Normal Distribution

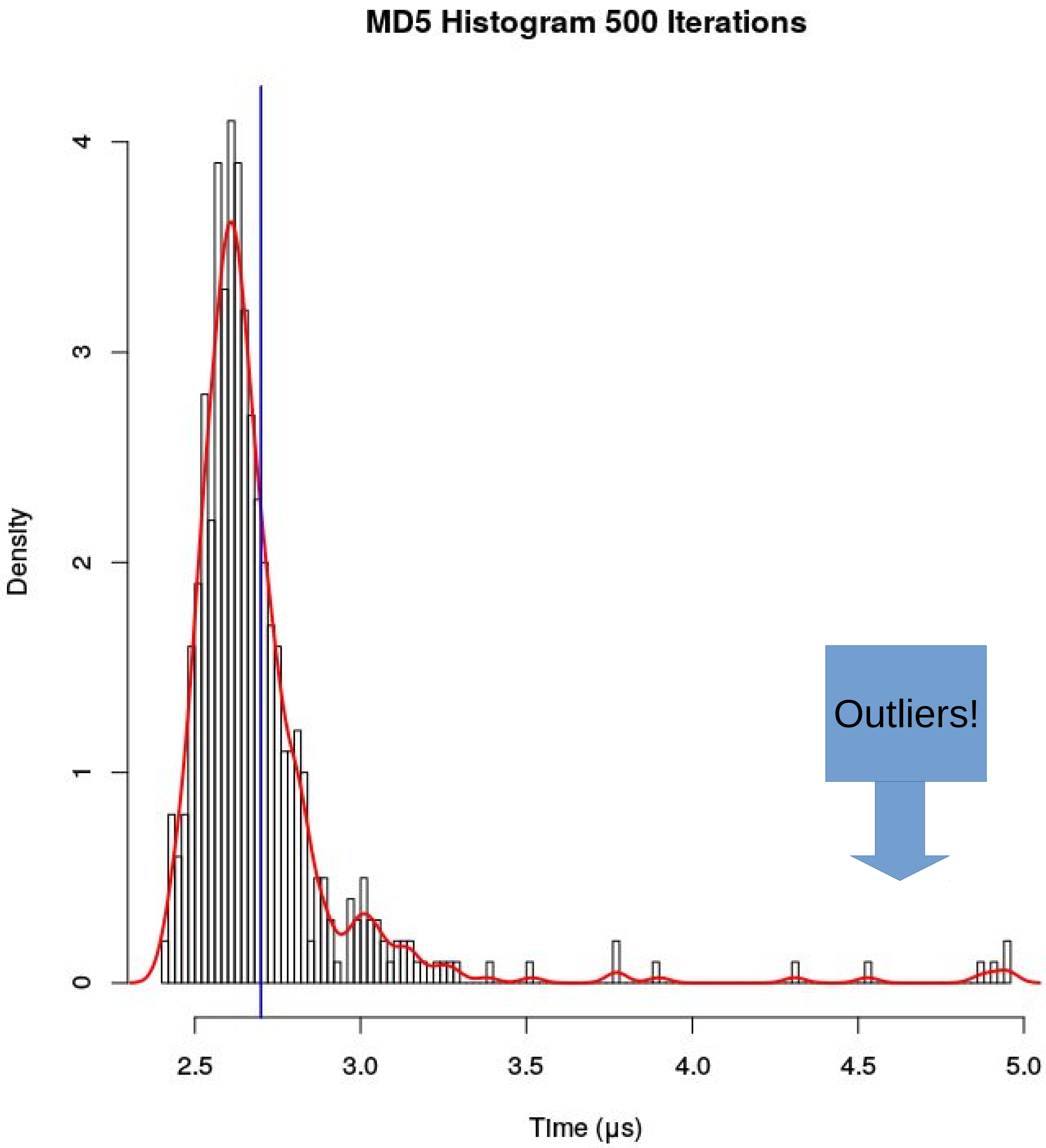


- Applies to any set of random variables with an Expected Value.
- Benchmark times should be normally distributed.
- Peak is the most probable time.


Histogram of md55001$time
Density
0
1
2
3
4
2.5
3.0
3.5
4.0
4.5
md55001$time


Histogram of md55002$time
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
2.5
3.0
3.5
4.0
4.5
5.0
5.5
md55002$time


Histogram of md55003$time
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
md55003$time


MD5 Histogram 500 Iterations

Outliers!

The Mean and the Mode

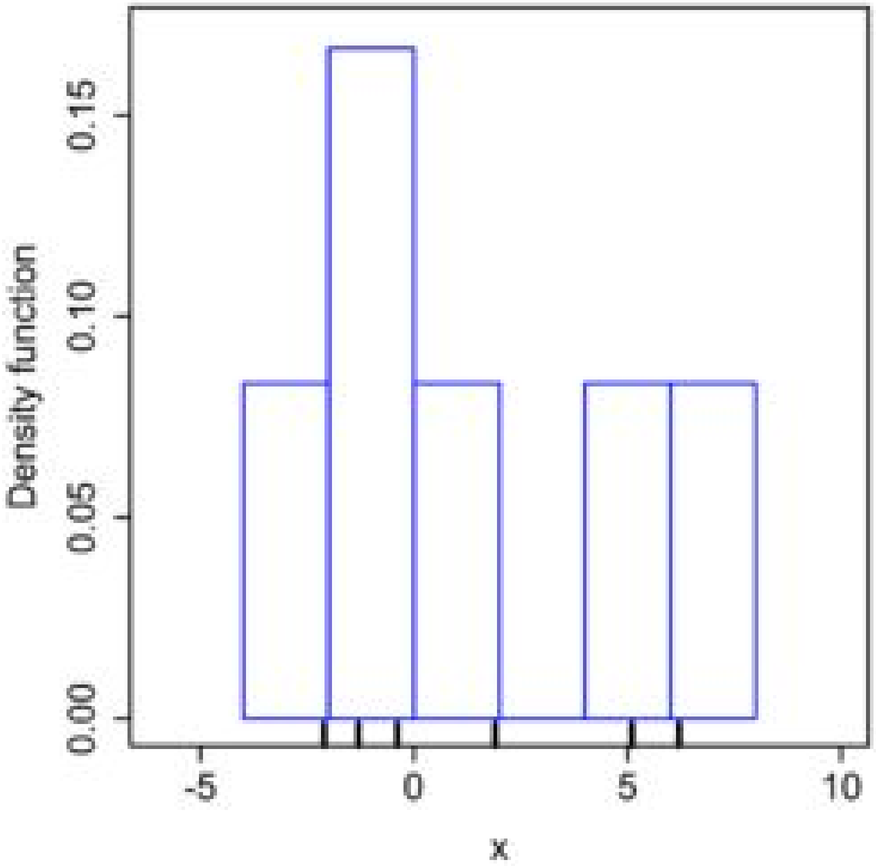
Mean: Average value of the set:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

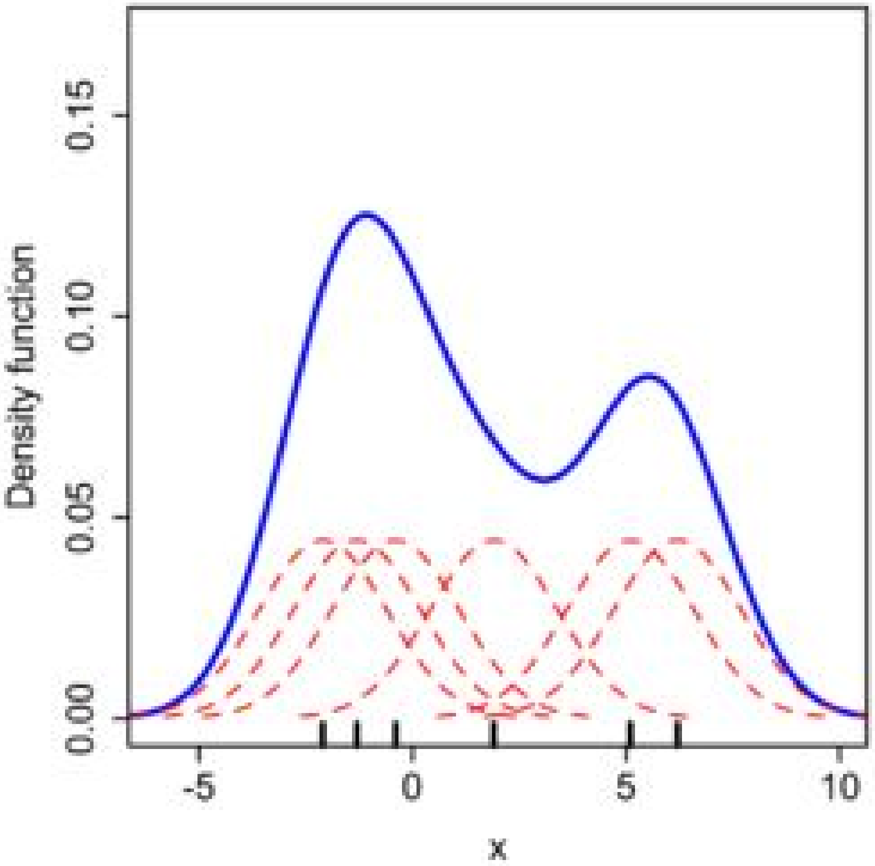
Mode: Most common value in the set...

Determining the mode

histogram



kernel density estimate



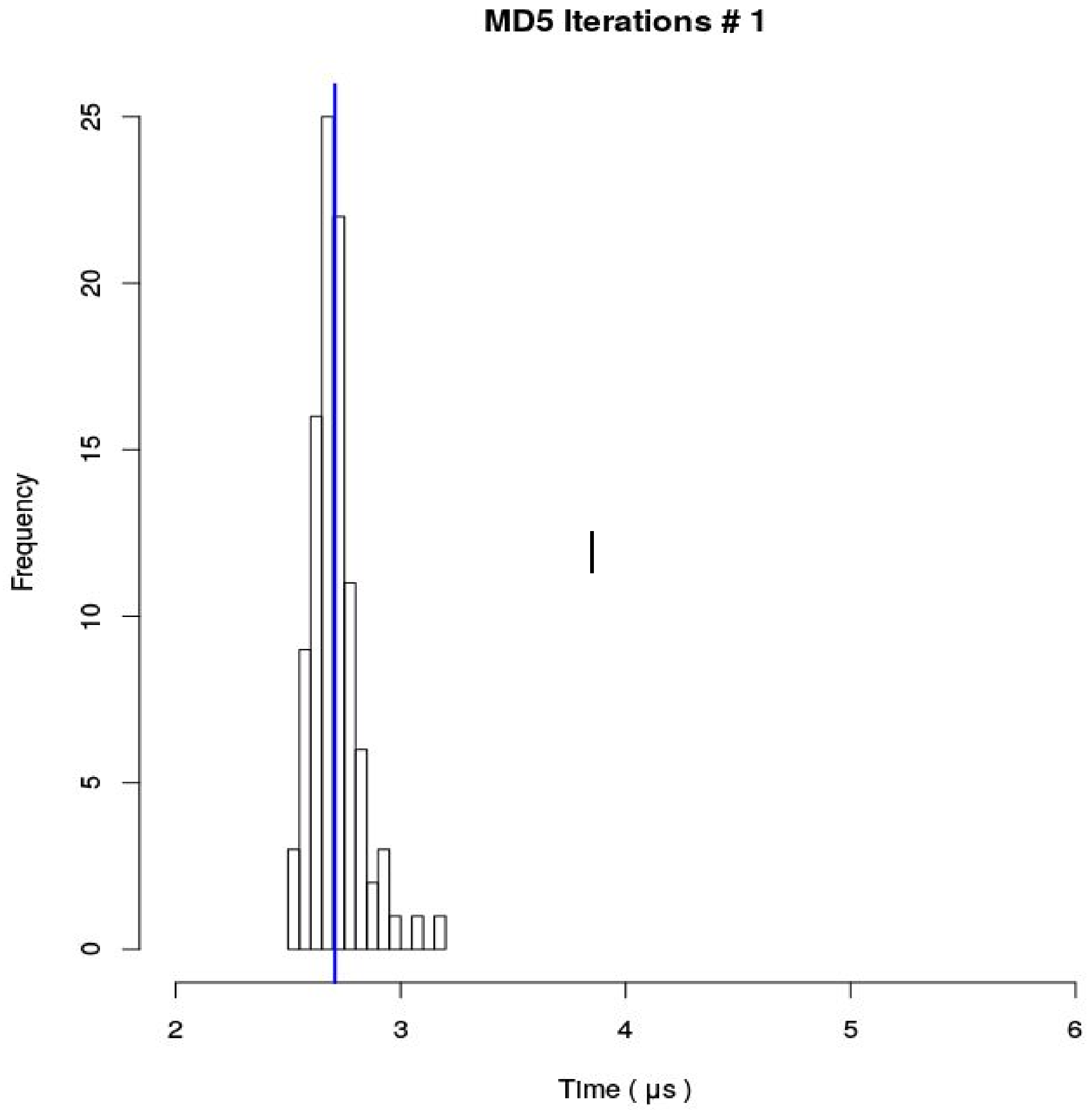
https://en.wikipedia.org/wiki/Kernel_density_estimate

Measuring Stability

Standard Deviation (SD)

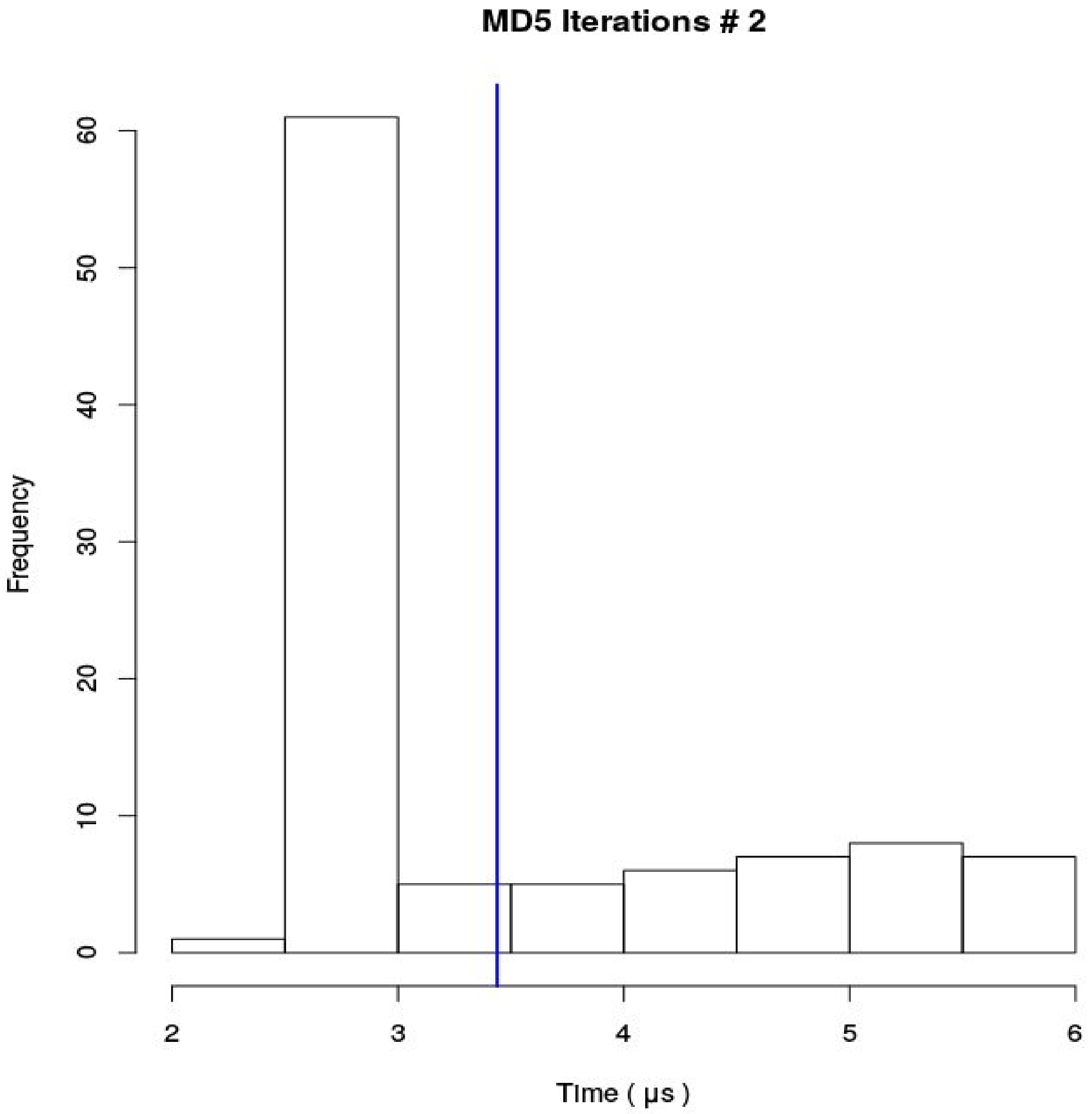
- Unit represented as “sigma” **σ**
- ***Almost*** the average distance between values in the set.
- An SD of zero indicates that all the values are identical.
- Relative standard deviation (RSD) is the SD divided by the mean. It is a percentage.
- RSD provides a comparable number.

Standard Deviation



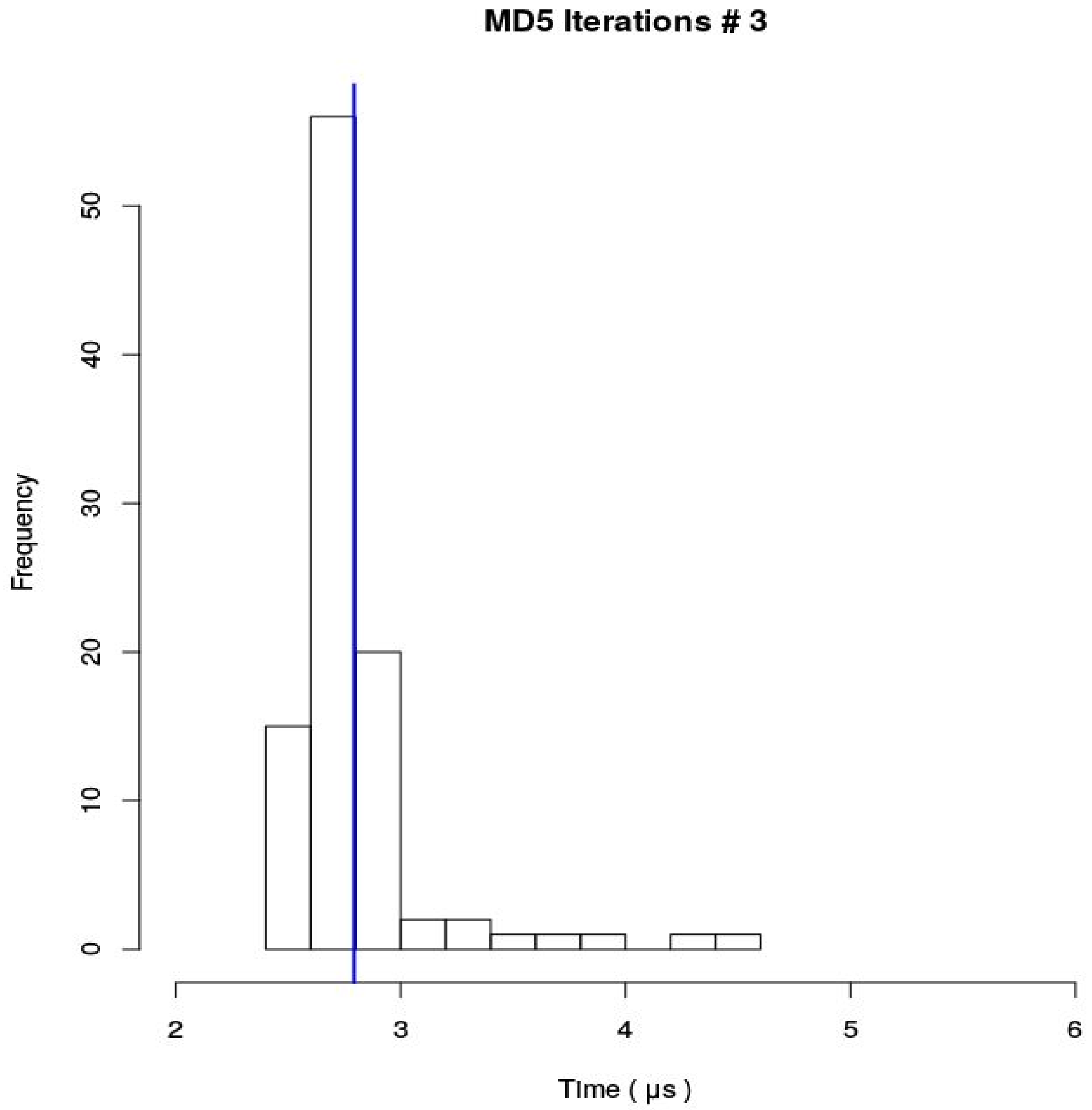
RSD = 3%
MEAN = 2.078
MIN = 2.5
MAX = 3.2

Standard Deviation



RSD = 30%
MEAN = 3.440
MIN = 2
MAX = 6

Standard Deviation



RSD = 11%
MEAN = 2.793
MIN = 2.5
MAX = 4.5

Improving Stability

Shutdown Unecessary Stuff

- No music processes.
- No video processes.
- No Grand Theft Aauto.

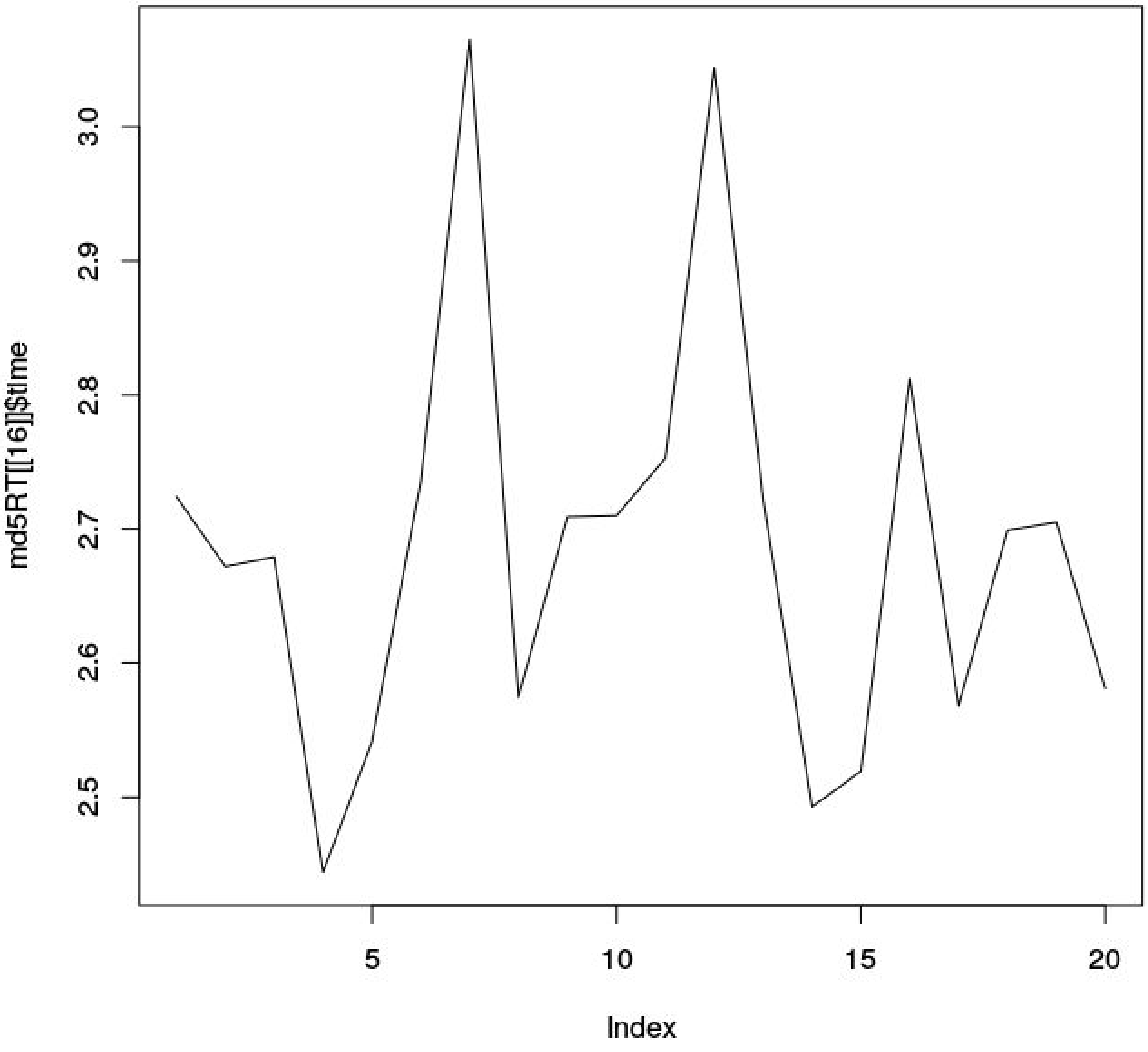
Retry Threshold

Keep running the iteration set until all times fall into a given margin of error (MOE).

- Feature of PHPBench
- Retries iterations when they fall outside of the MOE.

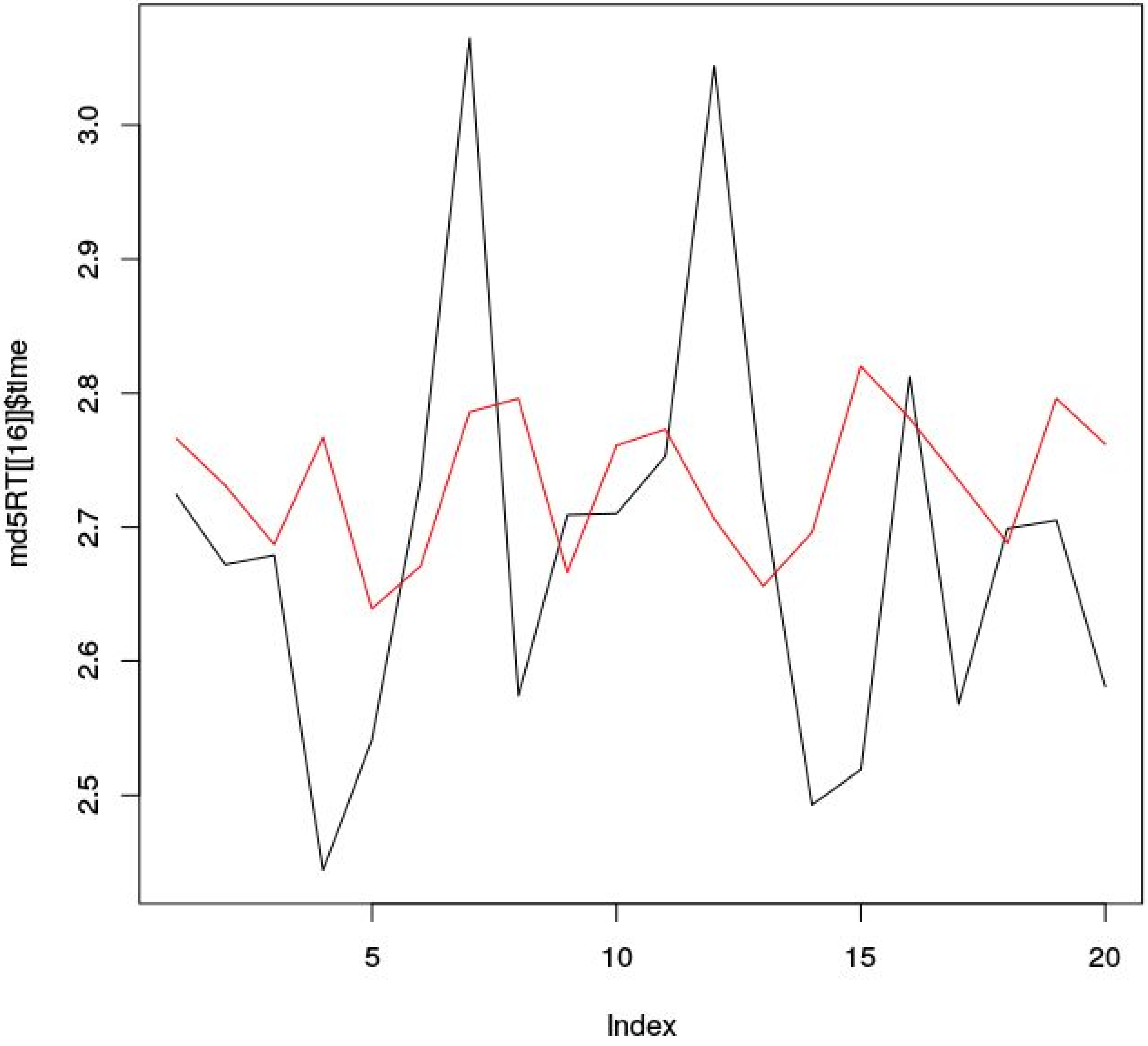
```
phpbench run examples/HashBench.php --retry-threshold=2
```

16% retry threshold



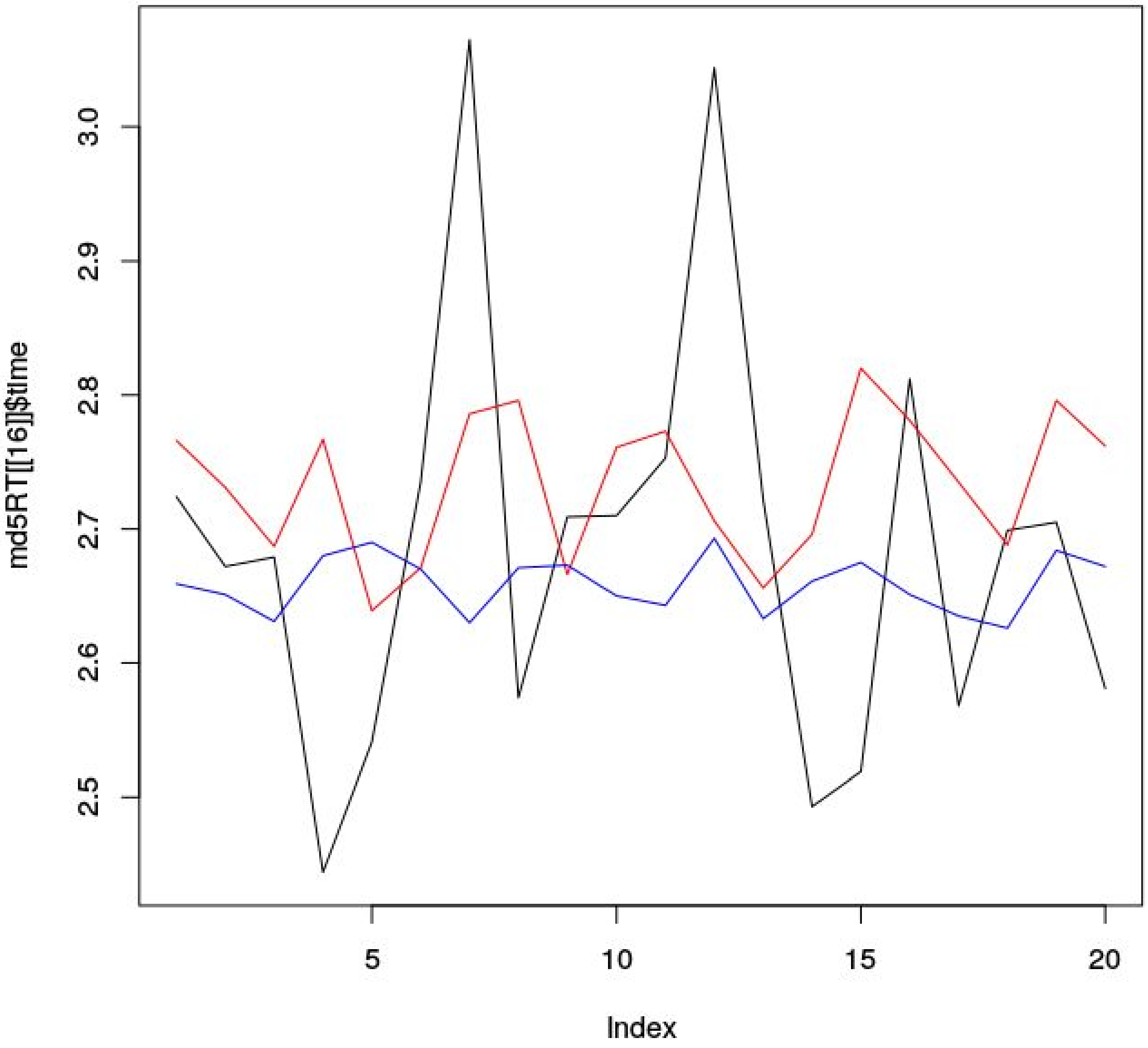
RSD: 5.7%

4% retry threshold



RSD: 1.9%

2% retry threshold



RSD: < 1%

Take Away

- Use a large number of revolutions for micro benchmarks (at least 1000).
- For best results use a large number of Iterations (100-500).
- Use less for casual use.
- Enforce a 2-5% margin-of-error consensus to reduce standard deviation.

Demonstration!

The End

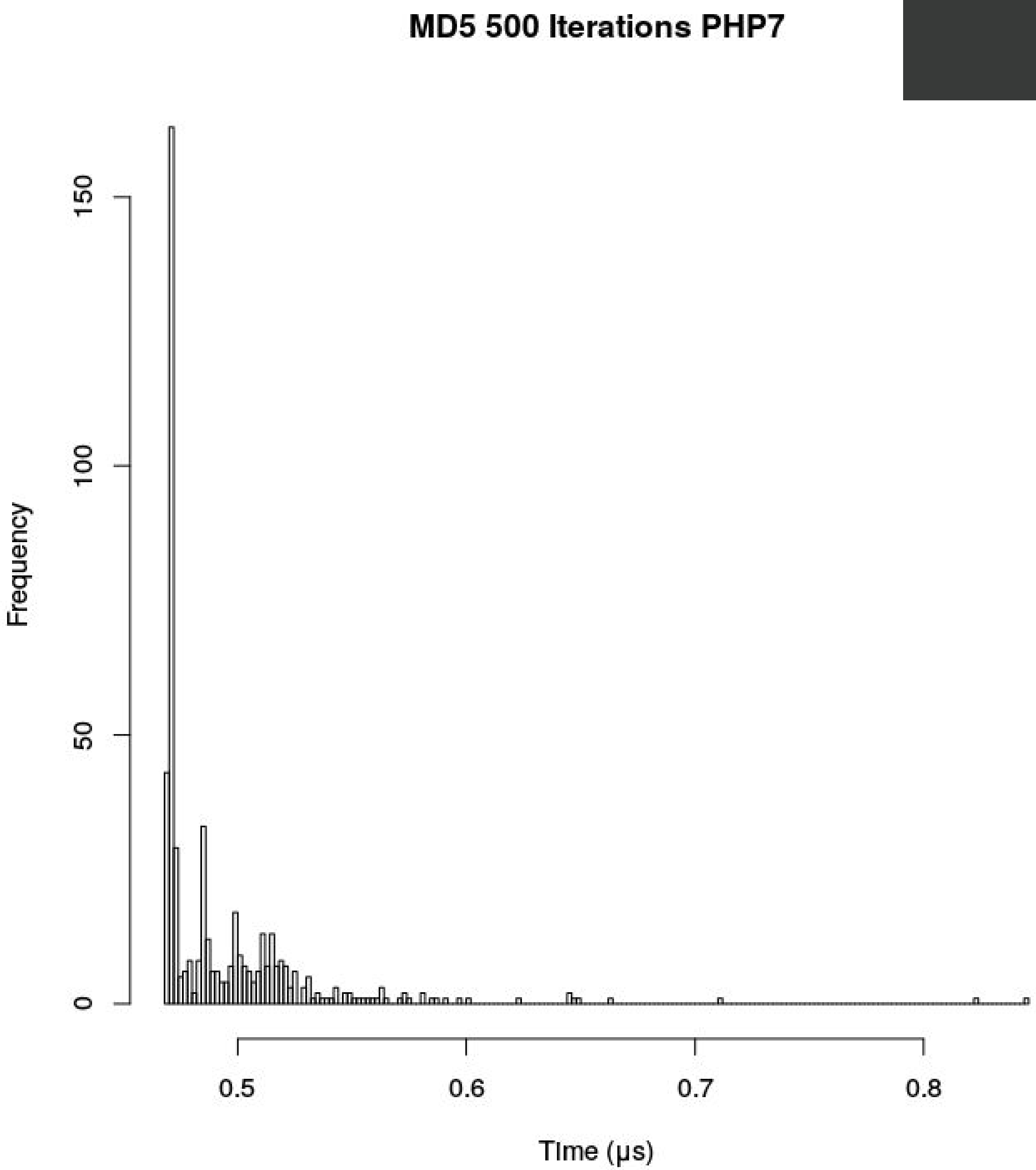
Github: https://github.com/phpbench/phpbench

Readthedocs: https://phpbench.readthedocs.org/

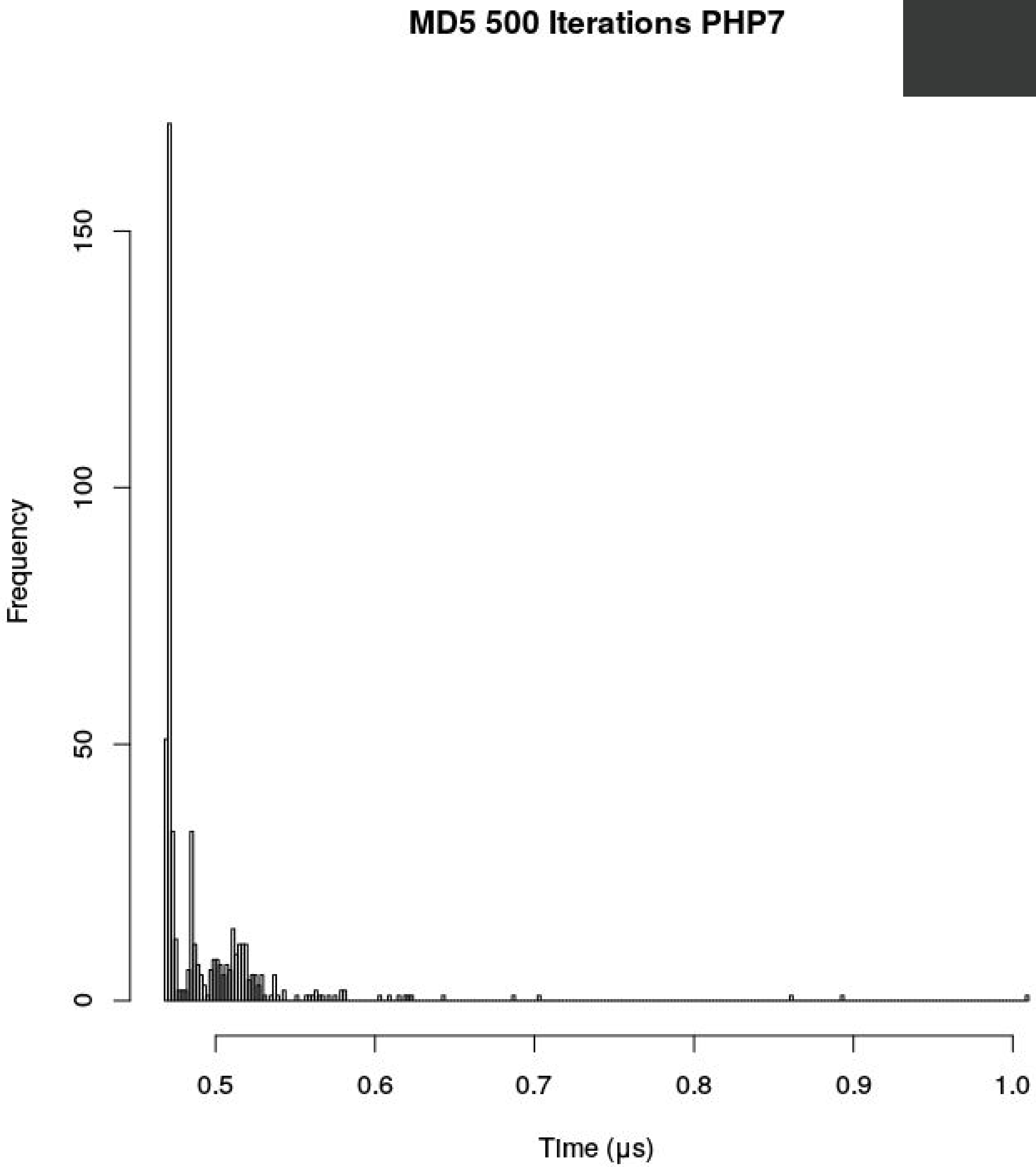
Twitter: @phpbench @dantleech

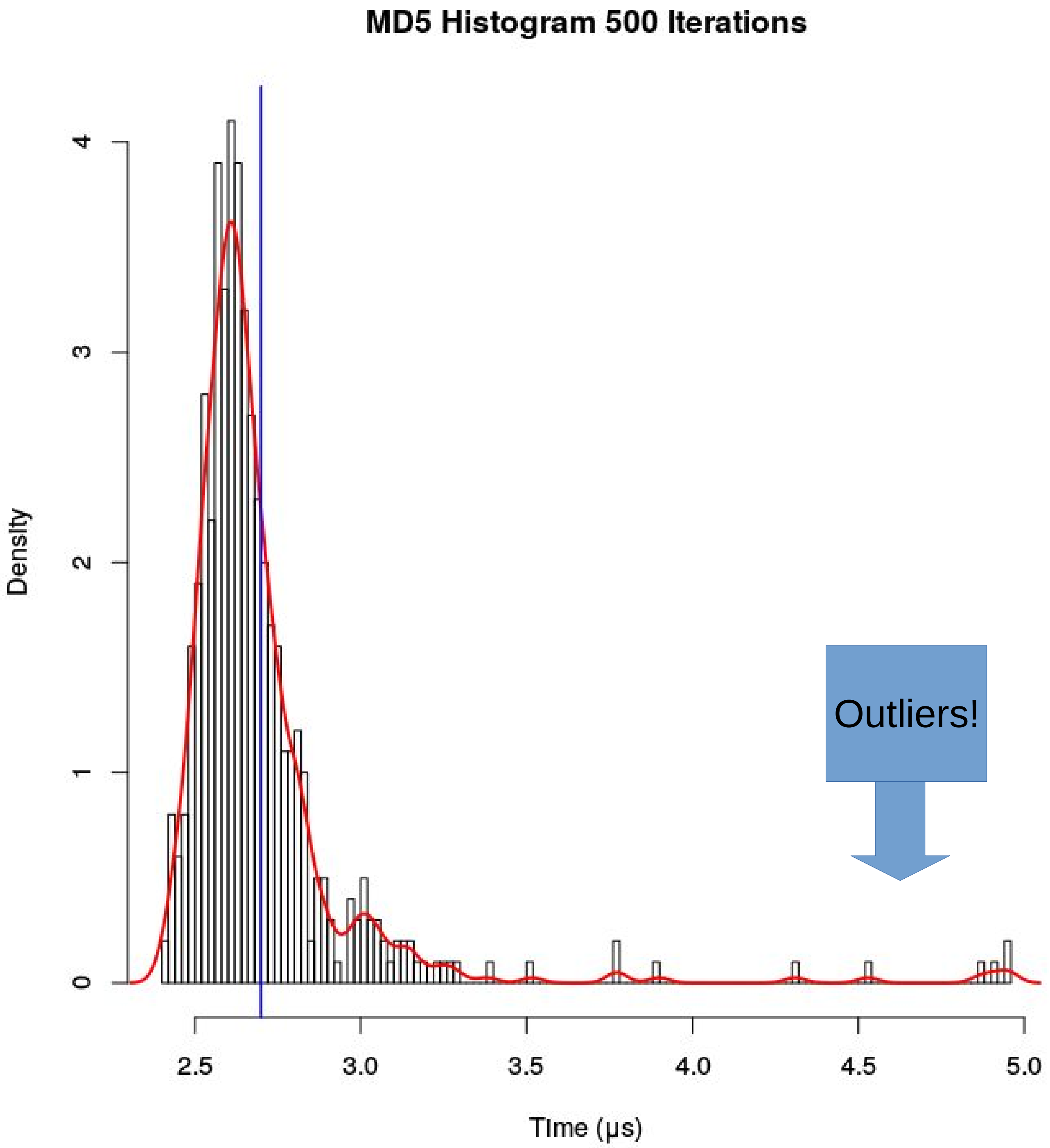
Things we should take with us

- **Wall time**: Total (external) time taken.
- **CPU time**: CPU time taken.
- **Microsecond**: 1,000,000th of a second
- **Mean**: Average of a set of numbers.
- **Standard Deviation (SD)**: Average deviation from the mean between samples (kind of), often represented as sigma (σ).
- **Relative SD**: As above but expressed as a percentage (dimensionless quantity).


MD5 500 Iterations PHP7
Frequency
150
100
50
0
0.5
0.6
0.7
0.8
Time (µs)

php7


MD5 500 Iterations PHP7
Frequency
0
50
100
150
0.5
0.6
0.7
0.8
0.9
1.0
Time (μs)


MD5 Histogram 500 Iterations
Density
0
1
2
3
4
Time (μs)
2.5
3.0
3.5
4.0
4.5
5.0

Outliers!