
The other day

Reverse Sort

```ruby
array = (1..1_000).to_a

array.sort do |item, other|
  other <=> item
end
```

CRuby vs JRuby

```
$ ruby -v
ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-linux]
$ time ruby scripts/sort.rb
real  0m0.151s
user  0m0.052s
sys   0m0.016s
$ asdf local ruby jruby-9.1.8.0
$ ruby -v
jruby 9.1.8.0 (2.3.1) 2017-03-06 90fc7ab OpenJDK 64-Bit
Server VM 25.121-b13 on 1.8.0_121-8u121-b13-
0ubuntu1.16.04.2-b13 +jit [linux-x86_64]
$ time ruby scripts/sort.rb
real  0m3.468s
user  0m8.384s
sys   0m0.232s
```


Success!

The End?

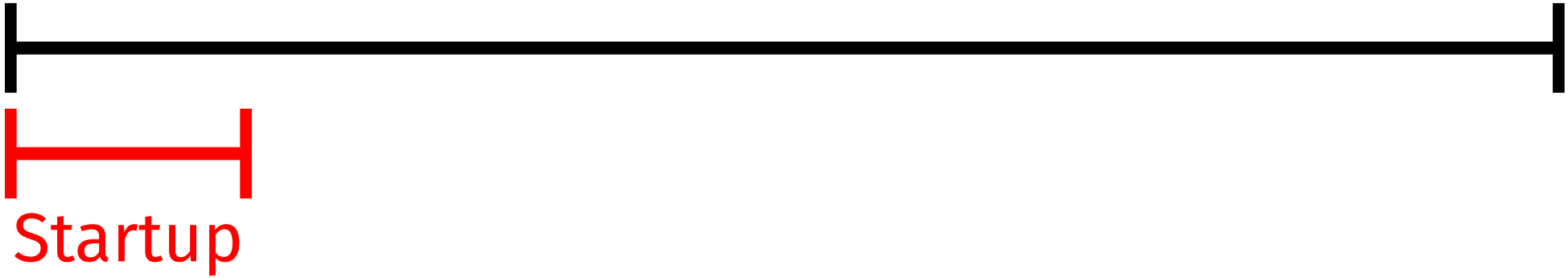

I HAVE NO
IDEA WHAT
I'M DOING

Numerous Failures!

- Way too few samples
- Realistic data/multiple inputs?
- No warmup
- Non production environment
- Does creating the array matter?
- Is reverse sorting really the bottle neck?
- Setup information
- Running on battery
- Lots of applications running

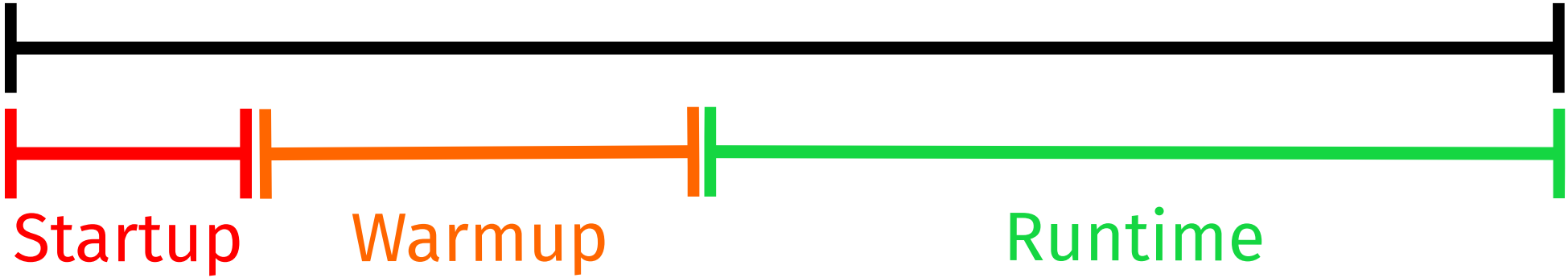
What’s important for you?

What’s important for you?



What's important for you?

What's important for you?



README.md

MJIT

What's the branch about

- The branch `rtl_mjit_branch` is used for development of **RTL (register transfer language)** VM insns and MRI JIT (**MJIT** in brief) of the RTL insns
- The last branch merge point with the trunk is always the head of the branch `rtl_mjit_branch_base`
  - The branch `rtl_mjit_branch` will be merged with the trunk from time to time and correspondingly the head of the branch `rtl_mjit_branch_base` will be the last merge point with the trunk

RTL insns

- The major goal of RTL insns introduction is an implementation of **IR for Ruby code analysis and optimizations**
  - The current **stack based insns** are an inconvenient IR for such goal
- Secondary goal is faster interpretation of VM insns
  - Stack based insns create additional memory traffic. Let us consider Ruby code `a = b + c`. Stack insns vs RTL insns for the code:

```
getlocal_OP__WC__0 <b index>
getlocal_OP__WC__0 <c index>
opt_plus
setlocal_OP__WC__0 <a index>
```

```
vs
```

```
plus <a index>, <b index>, <c index>
```

- Stack based insns are **shorter** but usually **require more insns** than RTL ones for the same Ruby code
  - We save time on memory traffic and insn dispatching

	v2	base	rtl	mjit	mjit-cl	omr	jruby9k	jruby9k-d	graal-22
while	1.0	1.11	1.82	387.29	9.28	1.06	2.3	2.89	2.35
nest-while	1.0	1.11	1.71	4.97	3.97	1.05	1.38	2.58	1.66
nest-ntimes	1.0	1.02	1.13	2.19	2.43	1.01	0.94	0.97	2.19
ivread	1.0	1.13	1.31	13.67	9.48	1.13	2.42	2.99	2.33
ivwrite	1.0	1.18	1.78	15.01	7.59	1.13	2.52	2.93	1.97
aread	1.0	1.03	1.44	19.69	7.03	0.98	1.79	3.53	2.17
awrite	1.0	1.09	1.42	13.09	7.45	0.96	2.18	3.74	2.55
aref	1.0	1.13	1.67	25.73	10.17	1.09	1.87	3.69	3.71
aset	1.0	1.51	2.68	23.45	17.82	1.47	3.61	4.49	6.33
const	1.0	1.09	1.53	27.53	10.15	1.05	2.98	3.89	3.01
const2	1.0	1.12	1.31	26.13	10.06	1.09	3.05	3.81	2.41
call	1.0	1.14	1.54	5.53	4.75	0.9	2.18	4.99	2.86
fib	1.0	1.21	1.43	4.16	3.81	1.1	2.17	5.03	2.26
fannk	1.0	1.05	1.1	1.1	1.1	0.99	1.71	2.32	1.02
sieve	1.0	1.3	1.72	3.34	3.36	1.27	1.49	2.42	2.02
mandelbrot	1.0	0.94	1.11	2.08	2.11	1.08	0.96	1.56	2.45
meteor	1.0	1.24	1.27	1.71	1.71	1.16	0.9	0.92	0.54
nbody	1.0	1.05	1.14	2.73	3.07	1.26	0.97	2.31	2.14
norm	1.0	1.13	1.09	2.52	2.49	1.15	0.91	1.45	1.62
trees	1.0	1.14	1.23	2.3	2.21	1.2	1.41	1.53	0.78
pent	1.0	1.13	1.24	1.71	1.7	1.13	0.6	0.8	0.33
red-black	1.0	1.01	0.94	1.3	1.14	0.88	0.98	2.52	1.03
bench	1.0	1.16	1.18	1.54	1.57	1.15	1.28	2.75	1.81
GeoMean.	1.0	1.12	1.39	6.18	4.02	1.09	1.59	2.48	1.83

Awesome?

	v2	base	rtl	mjit	mjit-cl	omr	jruby9k	jruby9k-d	graal-22
while	1.0	1.11	1.82	387.29	9.28	1.06	2.3	2.89	2.35
nest-while	1.0	1.11	1.71	4.97	3.97	1.05	1.38	2.58	1.66
nest-ntimes	1.0	1.02	1.13	2.19	2.43	1.01	0.94	0.97	2.19
ivread	1.0	1.13	1.31	13.67	9.48	1.13	2.42	2.99	2.33
ivwrite	1.0	1.18	1.78	15.01	7.59	1.13	2.52	2.93	1.97
aread	1.0	1.03	1.44	19.69	7.03	0.98	1.79	3.53	2.17
awrite	1.0	1.09	1.42	13.09	7.45	0.96	2.18	3.74	2.55
aref	1.0	1.13	1.67	25.73	10.17	1.09	1.87	3.69	3.71
aset	1.0	1.51	2.68	23.45	17.82	1.47	3.61	4.49	6.33
const	1.0	1.09	1.53	27.53	10.15	1.05	2.98	3.89	3.01
const2	1.0	1.12	1.31	26.13	10.06	1.09	3.05	3.81	2.41
call	1.0	1.14	1.54	5.53	4.75	0.9	2.18	4.99	2.86
fib	1.0	1.21	1.43	4.16	3.81	1.1	2.17	5.03	2.26
fannk	1.0	1.05	1.1	1.1	1.1	0.99	1.71	2.32	1.02
sieve	1.0	1.3	1.72	3.34	3.36	1.27	1.49	2.42	2.02
mandelbrot	1.0	0.94	1.11	2.08	2.11	1.08	0.96	1.56	2.45
meteor	1.0	1.24	1.27	1.71	1.71	1.16	0.9	0.92	0.54
nbody	1.0	1.05	1.14	2.73	3.07	1.26	0.97	2.31	2.14
norm	1.0	1.13	1.09	2.52	2.49	1.15	0.91	1.45	1.62
trees	1.0	1.14	1.23	2.3	2.21	1.2	1.41	1.53	0.78
pent	1.0	1.13	1.24	1.71	1.7	1.13	0.6	0.8	0.33
red-black	1.0	1.01	0.94	1.3	1.14	0.88	0.98	2.52	1.03
bench	1.0	1.16	1.18	1.54	1.57	1.15	1.28	2.75	1.81
GeoMean.	1.0	1.12	1.39	6.18	4.02	1.09	1.59	2.48	1.83

Disclaimer

	v2	base	rtl	mjit	mjit-cl	omr	jruby9k	jruby9k-d	graal-22
while	1.0	1.11	1.82	387.29	9.28	1.06	2.3	2.89	2.35
nest-while	1.0	1.11	1.71	4.97	3.97	1.05	1.38	2.58	1.66
nest-ntimes	1.0	1.02	1.13	2.19	2.43	1.01	0.94	0.97	2.19
ivread	1.0	1.13	1.31	13.67	9.48	1.13	2.42	2.99	2.33
ivwrite	1.0	1.18	1.78	15.01	7.59	1.13	2.52	2.93	1.97
aread	1.0	1.03	1.44	19.69	7.03	0.98	1.79	3.53	2.17
awrite	1.0	1.09	1.42	13.09	7.45	0.96	2.18	3.74	2.55
aref	1.0	1.13	1.67	25.73	10.17	1.09	1.87	3.69	3.71
aset	1.0	1.51	2.68	23.45	17.82	1.47	3.61	4.49	6.33
const	1.0	1.09	1.53	27.53	10.15	1.05	2.98	3.89	3.01
const2	1.0	1.12	1.31	26.13	10.06	1.09	3.05	3.81	2.41
call	1.0	1.14	1.54	5.53	4.75	0.9	2.18	4.99	2.86
fib	1.0	1.21	1.43	4.16	3.81	1.1	2.17	5.03	2.26
fannk	1.0	1.05	1.1	1.1	1.1	0.99	1.71	2.32	1.02
sieve	1.0	1.3	1.72	3.34	3.36	1.27	1.49	2.42	2.02
mandelbrot	1.0	0.94	1.11	2.08	2.11	1.08	0.96	1.56	2.45
meteor	1.0	1.24	1.27	1.71	1.71	1.16	0.9	0.92	0.54
nbody	1.0	1.05	1.14	2.73	3.07	1.26	0.97	2.31	2.14
norm	1.0	1.13	1.09	2.52	2.49	1.15	0.91	1.45	1.62
trees	1.0	1.14	1.23	2.3	2.21	1.2	1.41	1.53	0.78
pent	1.0	1.13	1.24	1.71	1.7	1.13	0.6	0.8	0.33
red-black	1.0	1.01	0.94	1.3	1.14	0.88	0.98	2.52	1.03
bench	1.0	1.16	1.18	1.54	1.57	1.15	1.28	2.75	1.81
GeoMean.	1.0	1.12	1.39	6.18	4.02	1.09	1.59	2.48	1.83

Focus

time ruby pent.rb

Truffle: 33.58 seconds

vs

MJIT: 9.34 seconds

MJIT 3.5+ times faster!

Truffle: 33.58 seconds

vs

MJIT: 9.34 seconds

Startup

time ruby empty.rb

Truffle: 3.68 seconds

vs

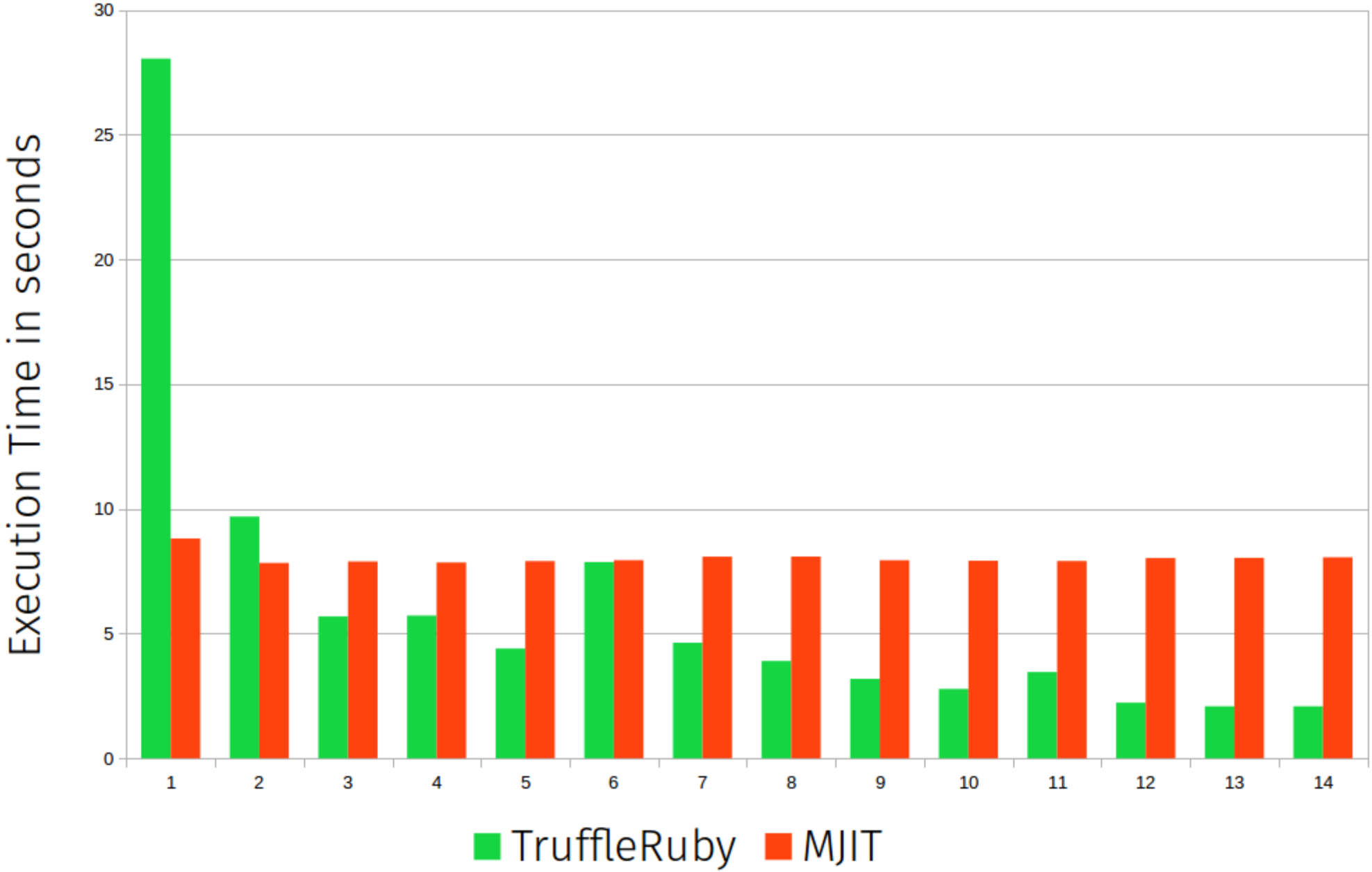
MJIT: 0.00 seconds

Warmup

Proper Benchmark

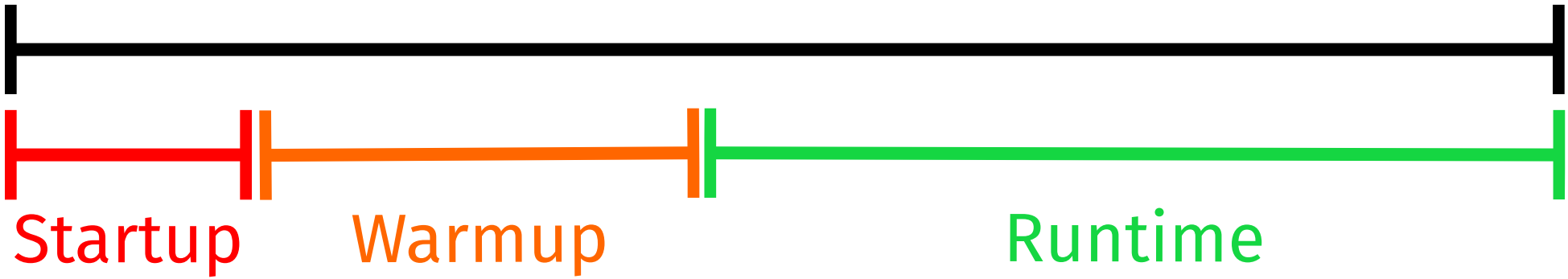
```ruby
Benchmark.avg do |benchmark|
  benchmark.config time: 60, warmup: 95

  benchmark.report "pent" do
    mkpieces
    mkboard
    $p[4] = [$p[4][0]]
    $pnum = (0...$p.length).to_a
    setpiece([],0)
  end
end
```

28s → 2s

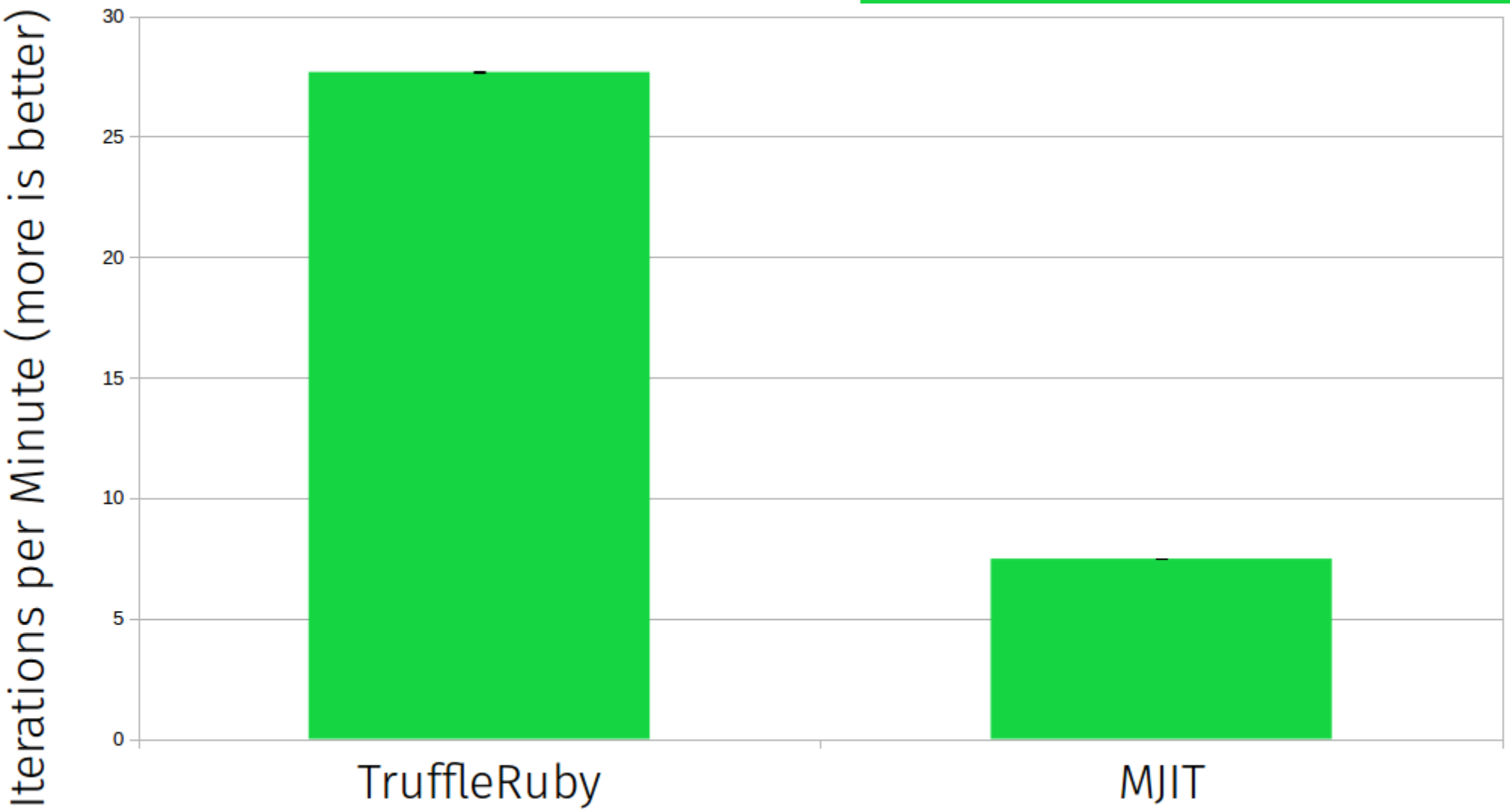

Execution Time in seconds
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
TruffleRuby
MJIT


Warmup

Runtime



Truffle 3.7+ times faster!



We just went from Truffle is 3.5 times slower all the way to Truffle is 3.7 times faster

RUN YOUR OWN BENCHMARKS

Stop Guessing and Start Measuring

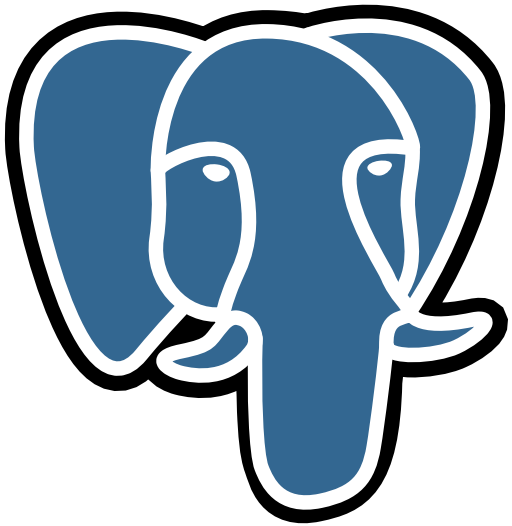
Benchmarking in Practice

Tobias Pfeiffer
@PragTob
pragtob.info

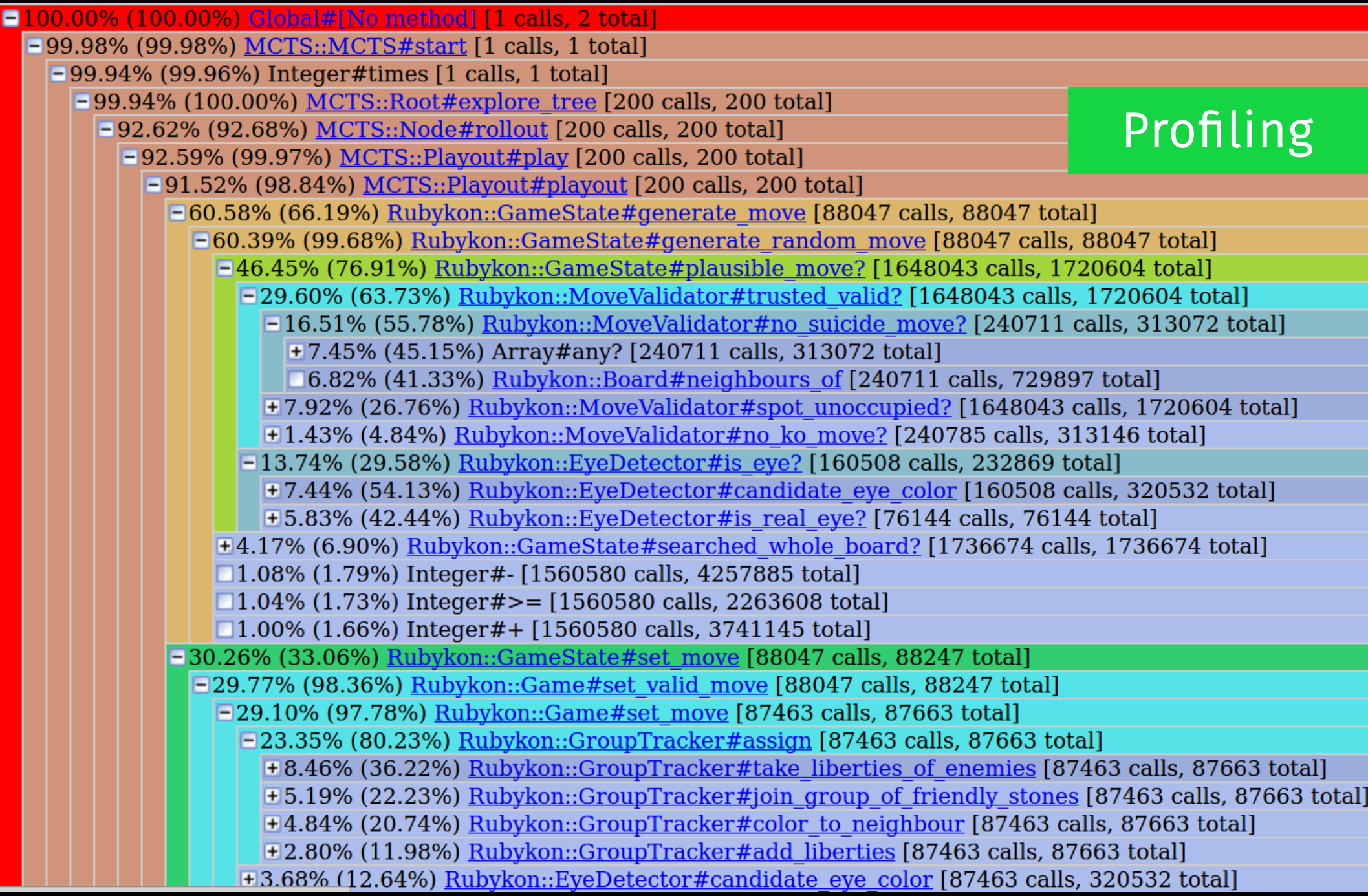
LIEFERY

Concept vs Tool Usage

A Poly-journey



Benchmarking vs. Profiling


Profiling
100.00% (100.00%) Global#[No method] [1 calls, 2 total]
99.98% (99.98%) MCTS::MCTS#start [1 calls, 1 total]
99.94% (99.96%) Integer#times [1 calls, 1 total]
99.94% (100.00%) MCTS::Root#explore_tree [200 calls, 200 total]
92.62% (92.68%) MCTS::Node#rollout [200 calls, 200 total]
92.59% (99.97%) MCTS::Playout#play [200 calls, 200 total]
91.52% (98.84%) MCTS::Playout#playout [200 calls, 200 total]
60.58% (66.19%) Rubykon::GameState#generate_move [88047 calls, 88047 total]
60.39% (99.68%) Rubykon::GameState#generate_random_move [88047 calls, 88047 total]
46.45% (76.91%) Rubykon::GameState#plausible_move? [1648043 calls, 1720604 total]
29.60% (63.73%) Rubykon::MoveValidator#trusted_valid? [1648043 calls, 1720604 total]
16.51% (55.78%) Rubykon::MoveValidator#no_suicide_move? [240711 calls, 313072 total]
7.45% (45.15%) Array#any? [240711 calls, 313072 total]
6.82% (41.33%) Rubykon::Board#neighbours_of [240711 calls, 729897 total]
7.92% (26.76%) Rubykon::MoveValidator#spot_unoccupied? [1648043 calls, 1720604 total]
1.43% (4.84%) Rubykon::MoveValidator#no_ko_move? [240785 calls, 313146 total]
13.74% (29.58%) Rubykon::EyeDetector#is_eye? [160508 calls, 232869 total]
7.44% (54.13%) Rubykon::EyeDetector#candidate_eye_color [160508 calls, 320532 total]
5.83% (42.44%) Rubykon::EyeDetector#is_real_eye? [76144 calls, 76144 total]
4.17% (6.90%) Rubykon::GameState#searched_whole_board? [1736674 calls, 1736674 total]
1.08% (1.79%) Integer#- [1560580 calls, 4257885 total]
1.04% (1.73%) Integer#>= [1560580 calls, 2263608 total]
1.00% (1.66%) Integer#+ [1560580 calls, 3741145 total]
30.26% (33.06%) Rubykon::GameState#set_move [88047 calls, 88247 total]
29.77% (98.36%) Rubykon::Game#set_valid_move [88047 calls, 88247 total]
29.10% (97.78%) Rubykon::Game#set_move [87463 calls, 87663 total]
23.35% (80.23%) Rubykon::GroupTracker#assign [87463 calls, 87663 total]
8.46% (36.22%) Rubykon::GroupTracker#take_liberties_of_enemies [87463 calls, 87663 total]
5.19% (22.23%) Rubykon::GroupTracker#join_group_of_friendly_stones [87463 calls, 87663 total]
4.84% (20.74%) Rubykon::GroupTracker#color_to_neighbour [87463 calls, 87663 total]
2.80% (11.98%) Rubykon::GroupTracker#add_liberties [87463 calls, 87663 total]
3.68% (12.64%) Rubykon::EyeDetector#candidate_eye_color [87463 calls, 320532 total]

Application Performance Monitoring

System Specification

- (Ruby versions mentioned when used)
- OpenJDK 8
- GraalVM 25
- MJIT master as of July 6th (last commit mid of June)
- Elixir 1.3.4
- Erlang 19.1
- i5-7200U – 2 x 2.5GHz (Up to 3.10GHz)
- 8GB RAM
- Linux Mint 18.1 - 64 bit (Ubuntu 16.04 base)
- Linux Kernel 4.8.0
- Firefox 54.0

What to benchmark?

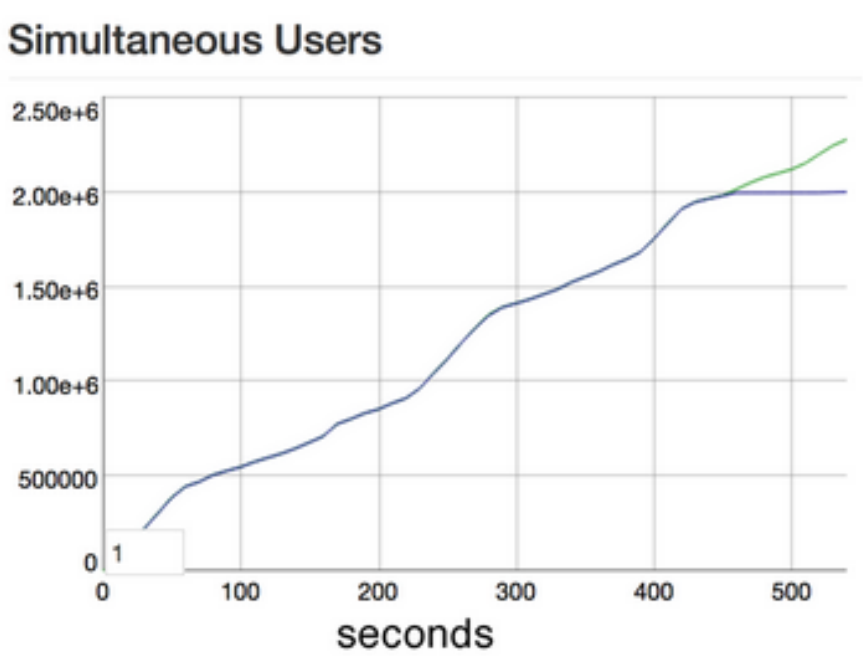
What to measure?

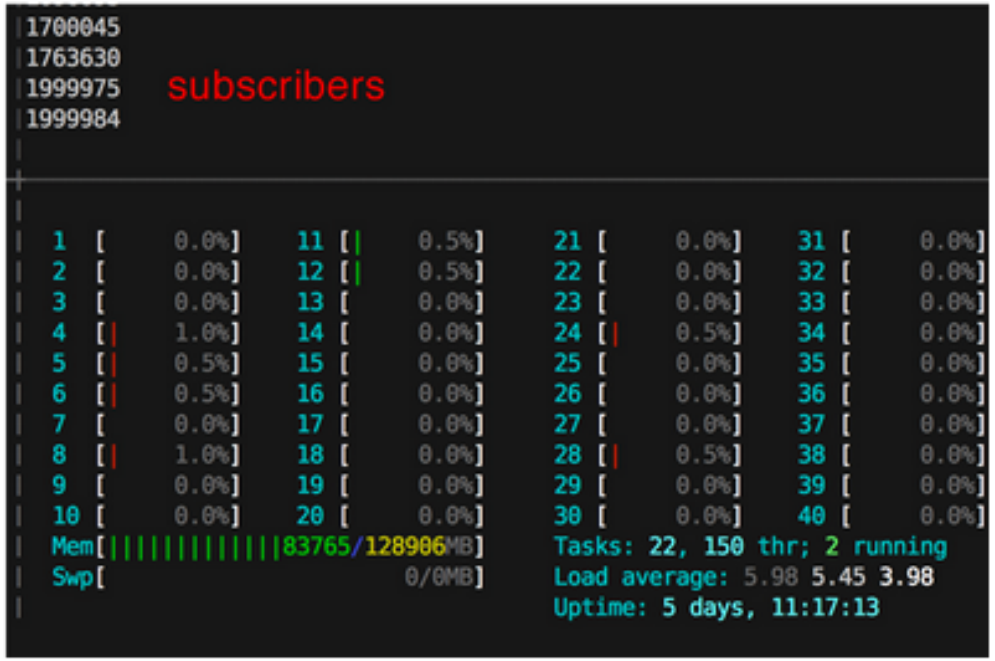
- Runtime?
- Memory?
- Throughput?
- Custom?

Amount of socket connection

The Road to 2 Million Websocket Connections in Phoenix

By Gary Rennie · about a year ago · v1.0.0





If you have been paying attention on Twitter recently, you have likely seen some increasing numbers regarding the number of simultaneous connections the Phoenix web framework can handle. This post documents some of the techniques used to perform the benchmarks.

What to measure?

- **Runtime!**
- Memory?
- Throughput?
- Custom?

Did we make it faster?

“Isn’t that the root of all evil?”

“More likely, not reading the sources is the source of all evil”

Me, just now

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil."

Donald Knuth, 1974

(Computing Surveys, Vol 6, No 4, December 1974)

The very next sentence

"Yet we should not pass up our opportunities in that critical 3%"

Donald Knuth, 1974

(Computing Surveys, Vol 6, No 4, December 1974)

Prior Paragraph

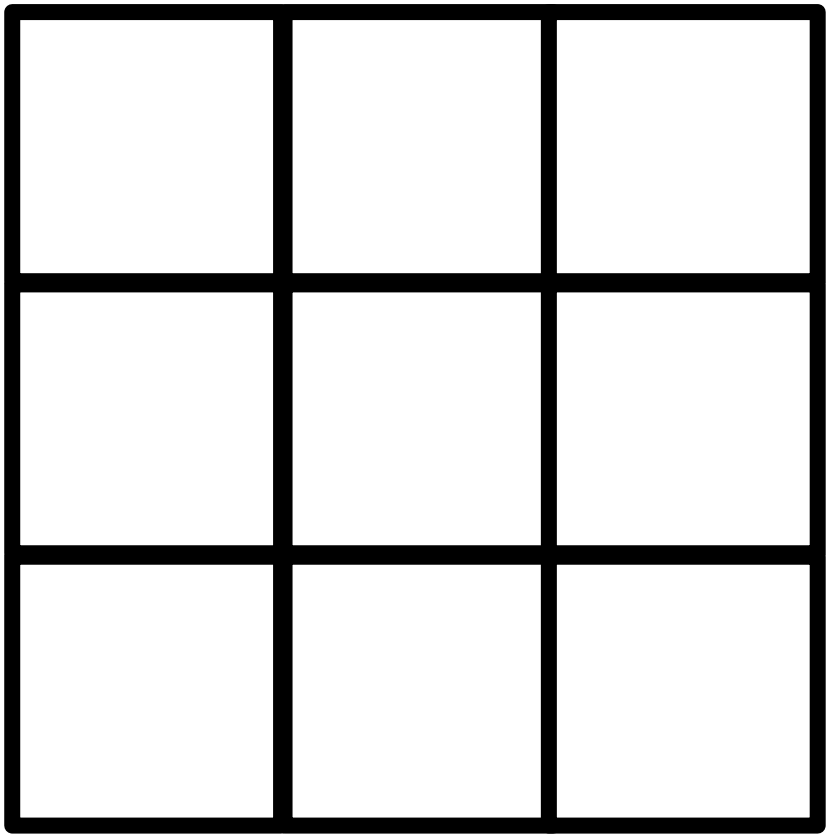
"(...) a 12 % improvement, easily obtained, is never considered marginal."

Donald Knuth, 1974

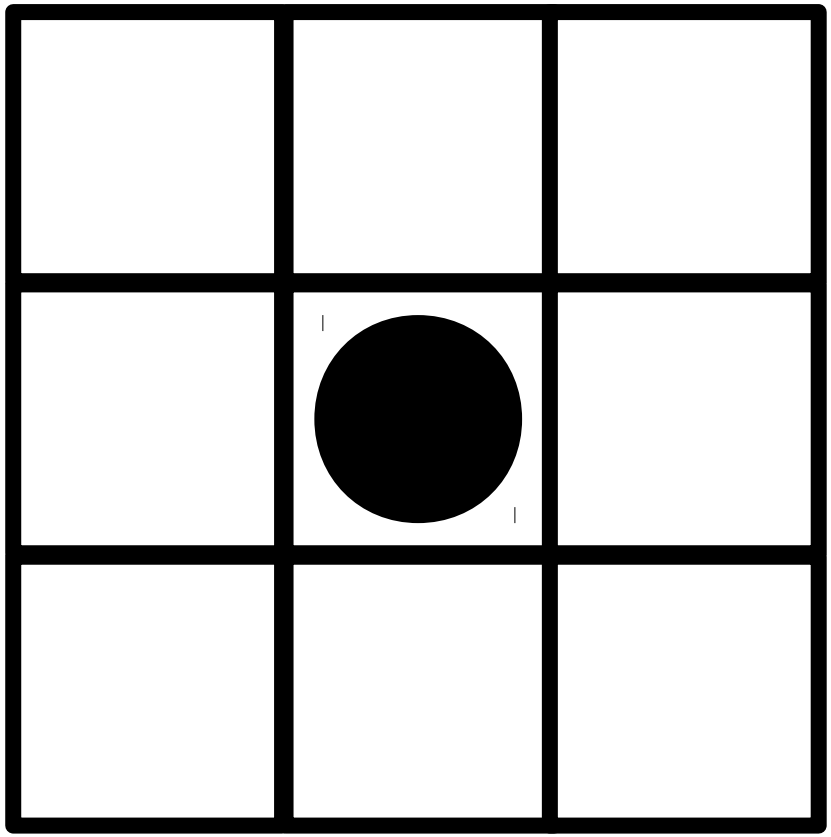
(Computing Surveys, Vol 6, No 4, December 1974)


JS

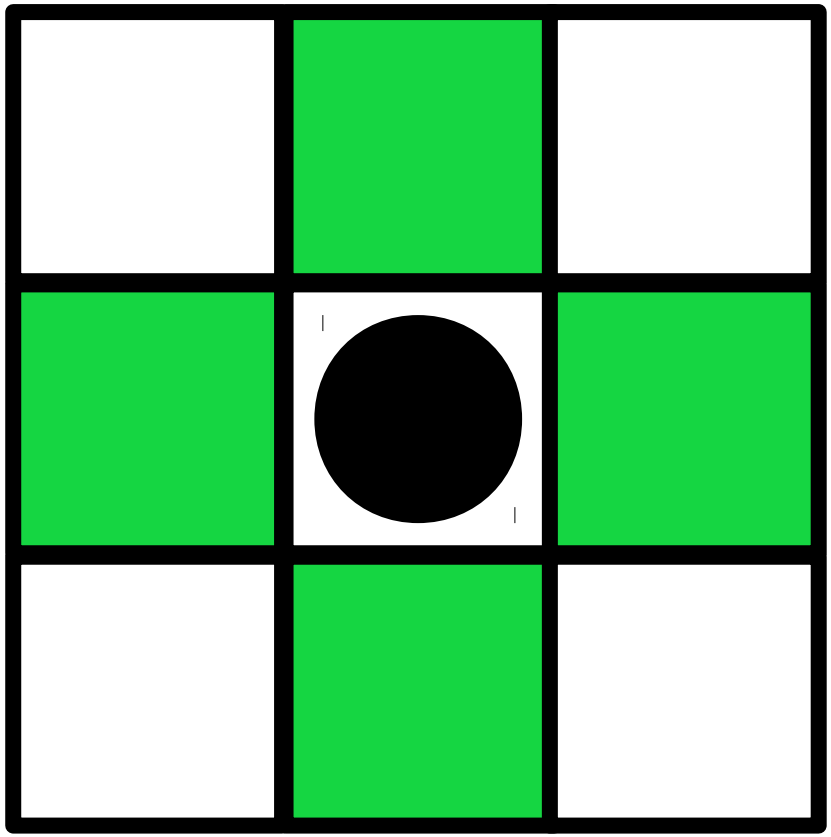

Go-Bots



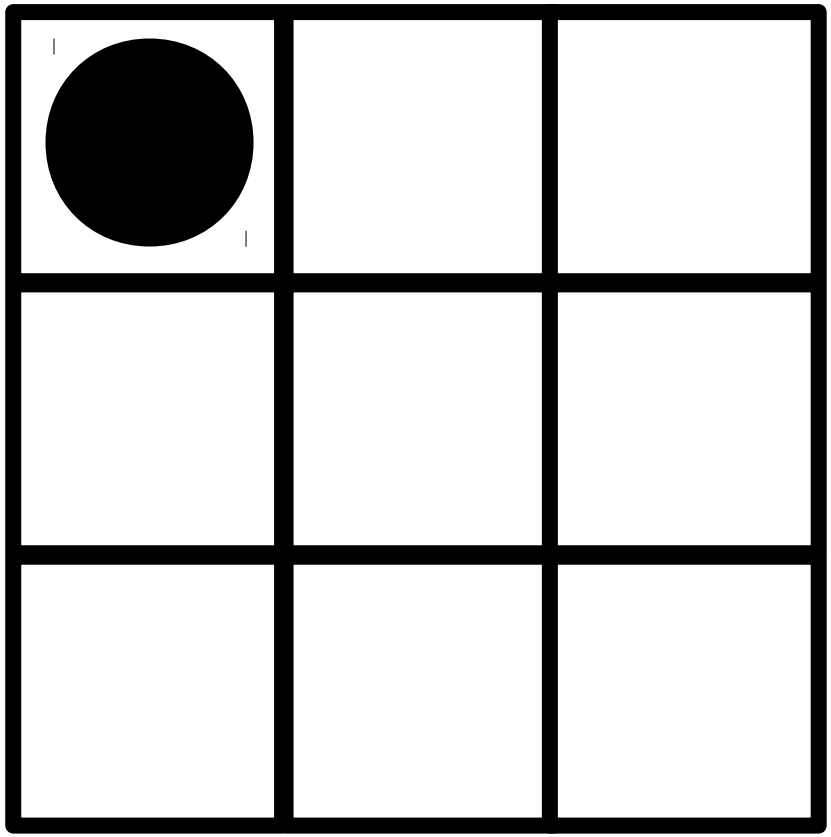
Neighbours?



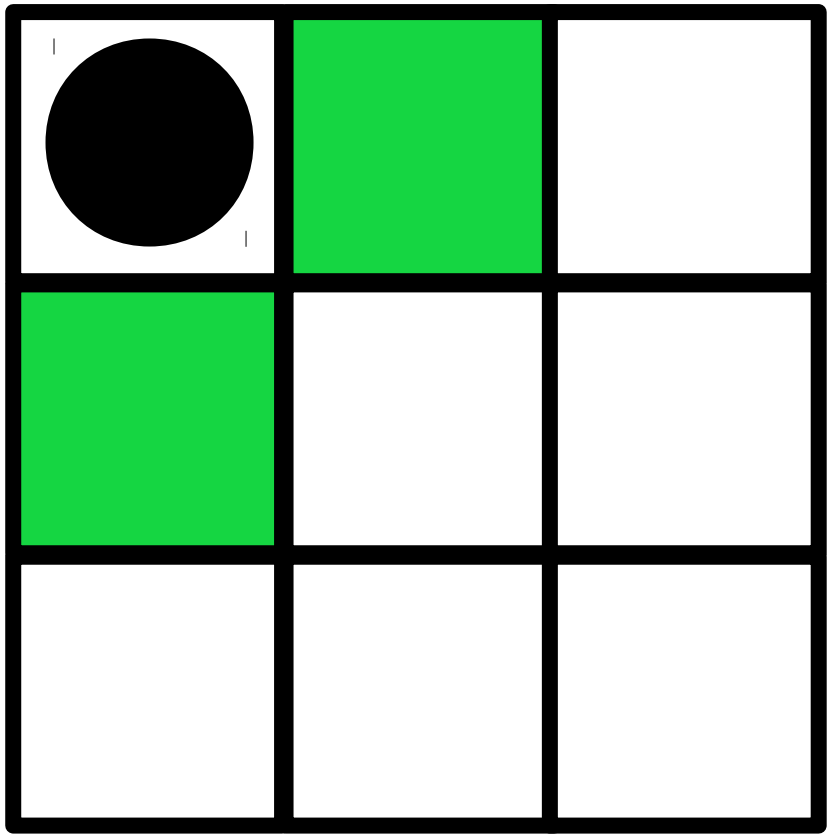
Neighbours?



Neighbours?



Neighbours?



Checks...

```
getColor = function(x, y, board) {
    if (isOutOfBounds(x, y, board)) {
        return NEUTRAL;
    } else {
        return board[y][x];
    }
};


isOutOfBounds = function(x, y, board) {
  var board_size = board.length;
  if ((x < 0) || (y < 0) ||
      (x >= board_size) || (y >= board_size)) {
    return true;
  } else {
    return false;
  }
};
```

Checks...

```
getColor = function(x, y, board) {
    if (isOutOfBounds(x, y, board)) {
        return NEUTRAL;
    } else {
        return board[y][x];
    }
};


isOutOfBounds = function(x, y, board) {
  var board_size = board.length;
  if ((x < 0) || (y < 0) ||
      (x >= board_size) || (y >= board_size)) {
    return true;
  } else {
    return false;
  }
};
```

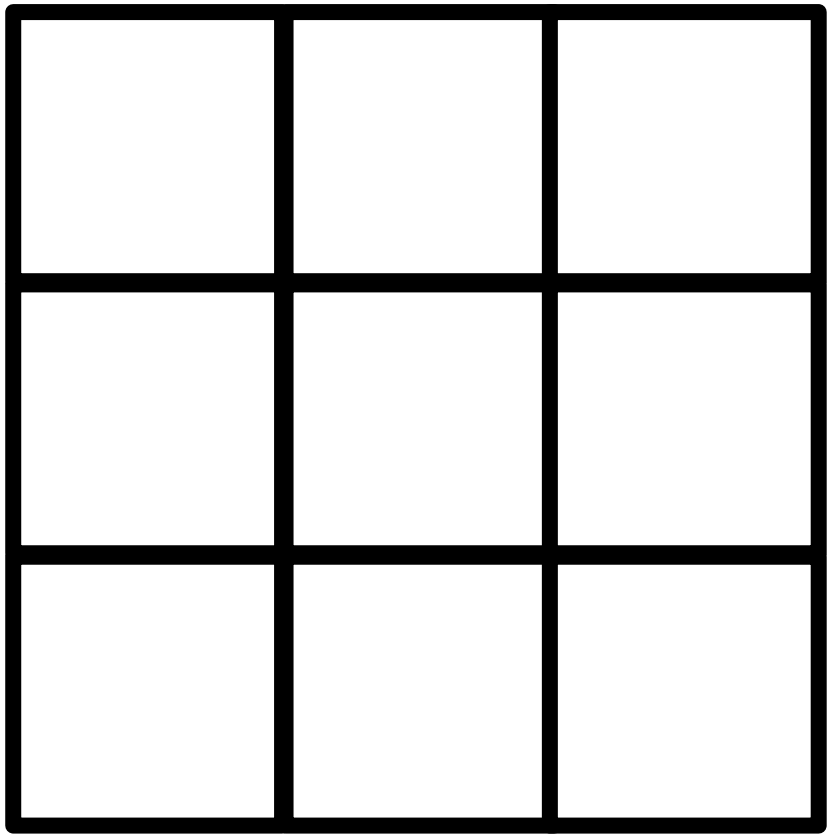
Checks...

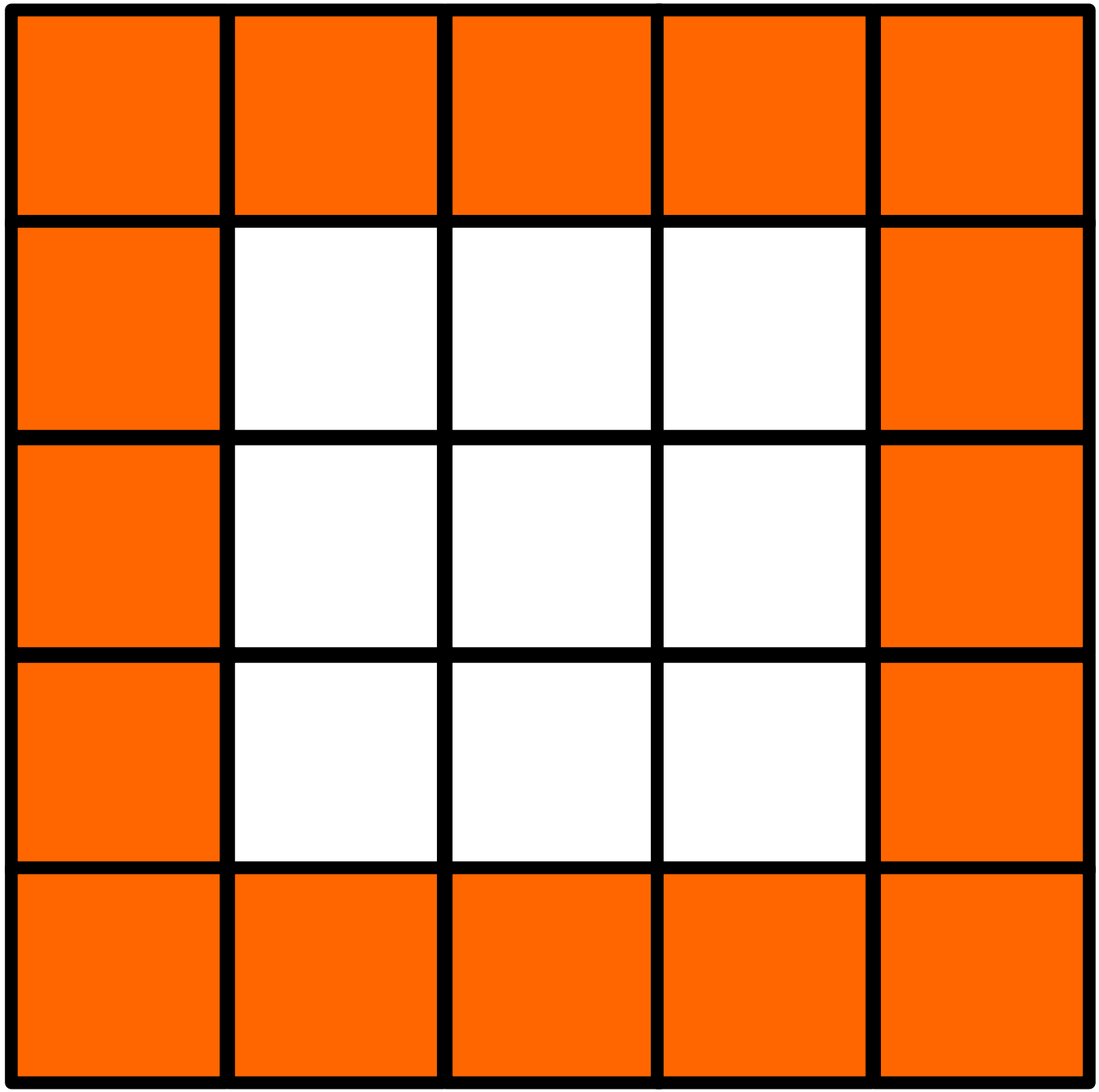
```
getColor = function(x, y, board) {
    if (isOutOfBounds(x, y, board)) {
        return NEUTRAL;
    } else {
        return board[y][x];
    }
};


isOutOfBounds = function(x, y, board) {
  var board_size = board.length;
  if ((x < 0) || (y < 0) ||
      (x >= board_size) || (y >= board_size)) {
    return true;
  } else {
    return false;
  }
};
```

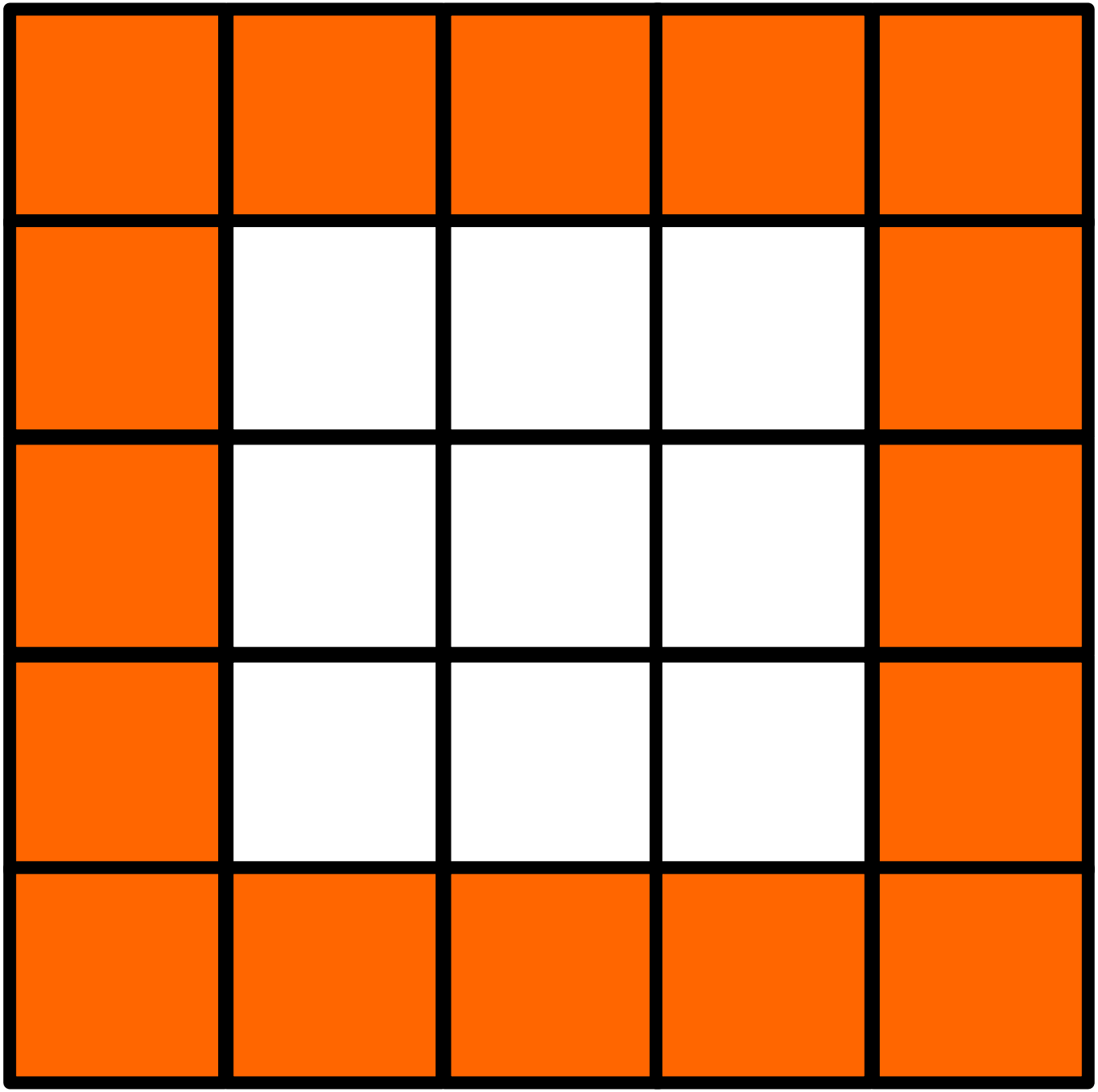
Checks...

```
getColor = function(x, y, board) {
    if (isOutOfBounds(x, y, board)) {
        return NEUTRAL;
    } else {
        return board[y][x];
    }
};


isOutOfBounds = function(x, y, board) {
  var board_size = board.length;
  if ((x < 0) || (y < 0) ||
      (x >= board_size) || (y >= board_size)) {
    return true;
  } else {
    return false;
  }
};
```

What if...

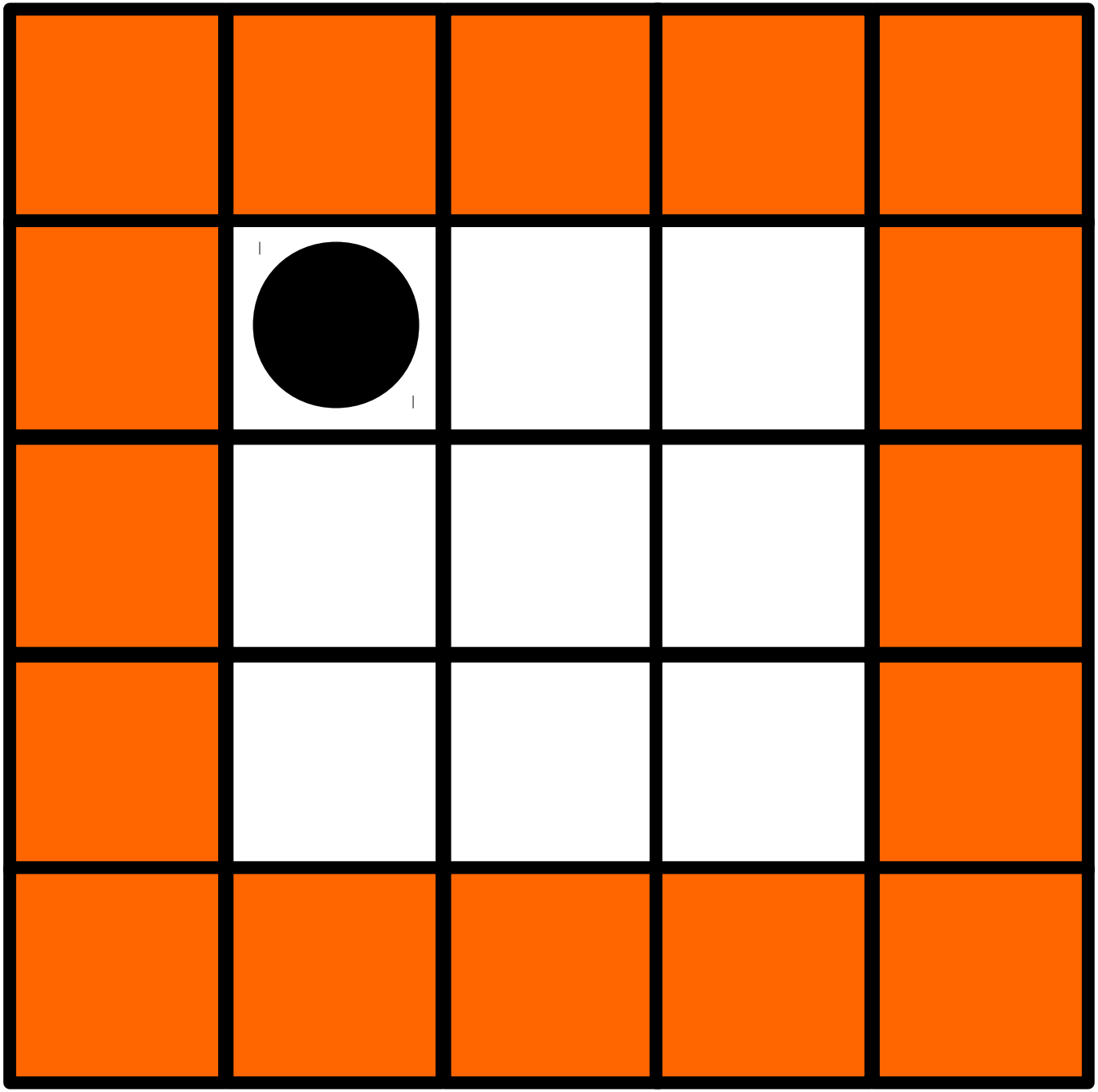



Layer

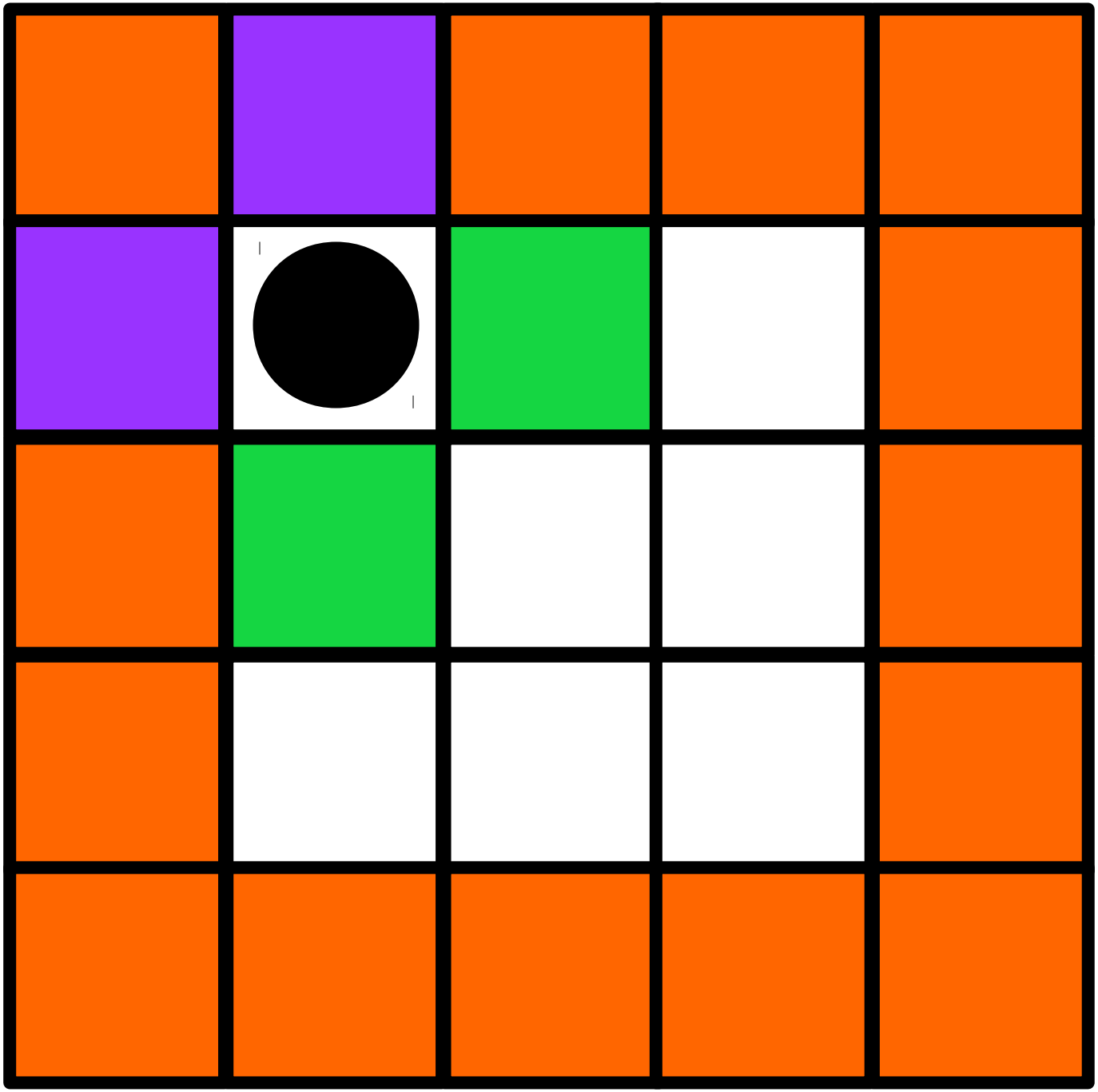


Neutral

Checks



Checks



BEST CODE EVER!

```
getColor = function(x, y, board) {
  return board[y + 1][x + 1];
};
```

Benchmark.js

```
var board = initBoard(3);
var suite = new Benchmark.Suite;
suite.add('getColor', function() {
  getColor(1, 2, board);
}).on('cycle', function(event) {
  // log result
}).on('complete', function() {
  // done
}).run({
  async: true
});
```

Benchmark function

```
var board = initBoard(3);
var suite = new Benchmark.Suite;
suite.add('getColor', function() {
  getColor(1, 2, board);
}).on('cycle', function(event) {
  // log result
}).on('complete', function() {
  // done
}).run({
  async: true
});
```

Run it!

Old

```
getColor x 278,959,094 ops/sec ±0.39%
(98 runs sampled)
```

New

```
getColor x 1,816,254,291 ops/sec ±0.17%
(94 runs sampled)
```

Run it!

Old

```
getColor x ~279 Million ops/sec ±0.39%
(98 runs sampled)
```

New

```
getColor x ~1 816 Million ops/sec ±0.17%
(94 runs sampled)
```

Over 6 times faster!!!

Old

```
getColor x ~279 Million ops/sec ±0.39%
(98 runs sampled)
```

New

```
getColor x ~1 816 Million ops/sec ±0.17%
(94 runs sampled)
```


Success!

Does it even matter?

A bigger benchmark

```
var suite = new Benchmark.Suite;
suite.add('playing a random 9x9 game', function() {
  playoutForBoard(initBoard(9));
}).add('playing a random 13x13 game', function() {
  playoutForBoard(initBoard(13));
}).add('playing a random 19x19 game', function() {
  playoutForBoard(initBoard(19));
}).on('cycle', function(event) {
  // log result here
}).on('complete', function() {
  // done
}).run({
  async: true
});
```

A bigger benchmark

```
var suite = new Benchmark.Suite;
suite.add('playing a random 9x9 game', function() {
  playoutForBoard(initBoard(9));
}).add('playing a random 13x13 game', function() {
  playoutForBoard(initBoard(13));
}).add('playing a random 19x19 game', function() {
  playoutForBoard(initBoard(19));
}).on('cycle', function(event) {
  // log result here
}).on('complete', function() {
  // done
}).run({
  async: true
});
```

Run it!

Old

```
9x9 game x 252 ops/sec ±2.44%
13x13 game x 80.50 ops/sec ±4.14%
19x19 game x 25.19 ops/sec ±6.33%
```

New

```
9x9 game x 199 ops/sec ±18.09%
13x13 game x 67.11 ops/sec ±3.24%
19x19 game x 20.97 ops/sec ±7.56%
```

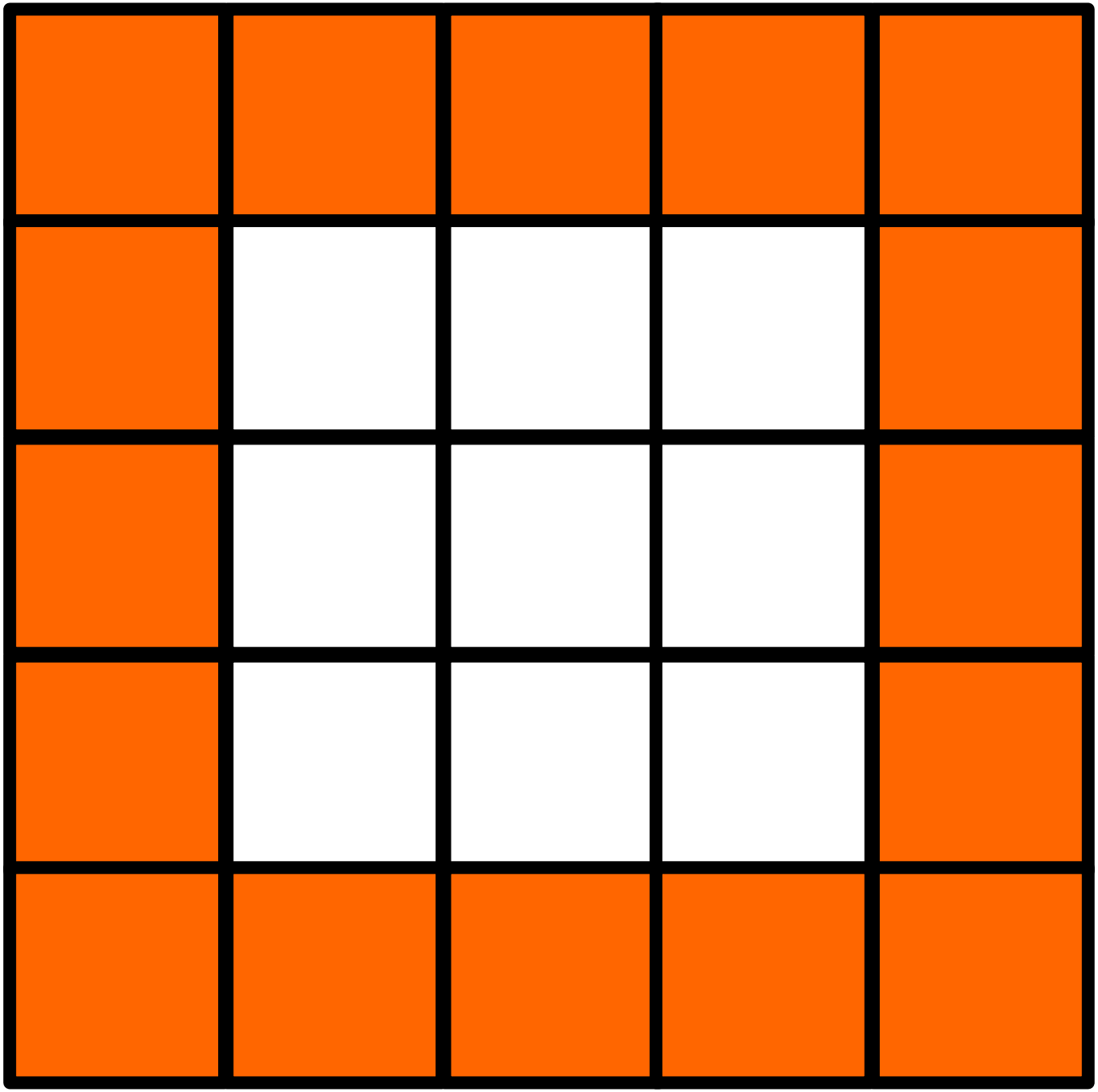
~20% slower?!?!

Old

```
9x9 game x 252 ops/sec ±2.44%
13x13 game x 80.50 ops/sec ±4.14%
19x19 game x 25.19 ops/sec ±6.33%
```

New

```
9x9 game x 199 ops/sec ±18.09%
13x13 game x 67.11 ops/sec ±3.24%
19x19 game x 20.97 ops/sec ±7.56%
```



Initializing & Copying Boards

Old

```
9x9 game x 252 ops/sec ±2.44%
13x13 game x 80.50 ops/sec ±4.14%
19x19 game x 25.19 ops/sec ±6.33%
```

New

```
9x9 game x 199 ops/sec ±18.09%
13x13 game x 67.11 ops/sec ±3.24%
19x19 game x 20.97 ops/sec ±7.56%
```

What are you measuring?
Does it matter?

Tobias Pfeiffer @PragTob · Jul 5

Also I believe in the grand scheme of things perf won't matter unless it's called a ton. Every call to the DB is so much more expensive

1 1

Piotr Szotkowski
@chastell

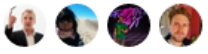
Following

Replying to @PragTob @devoncestes

ARE YOU LOOKING AT MY POLYCONF SLIDES TOBI

11:17 PM - 5 Jul 2017

4 Likes



2 4

Tweet your reply

Piotr Szotkowski @chastell · Jul 5

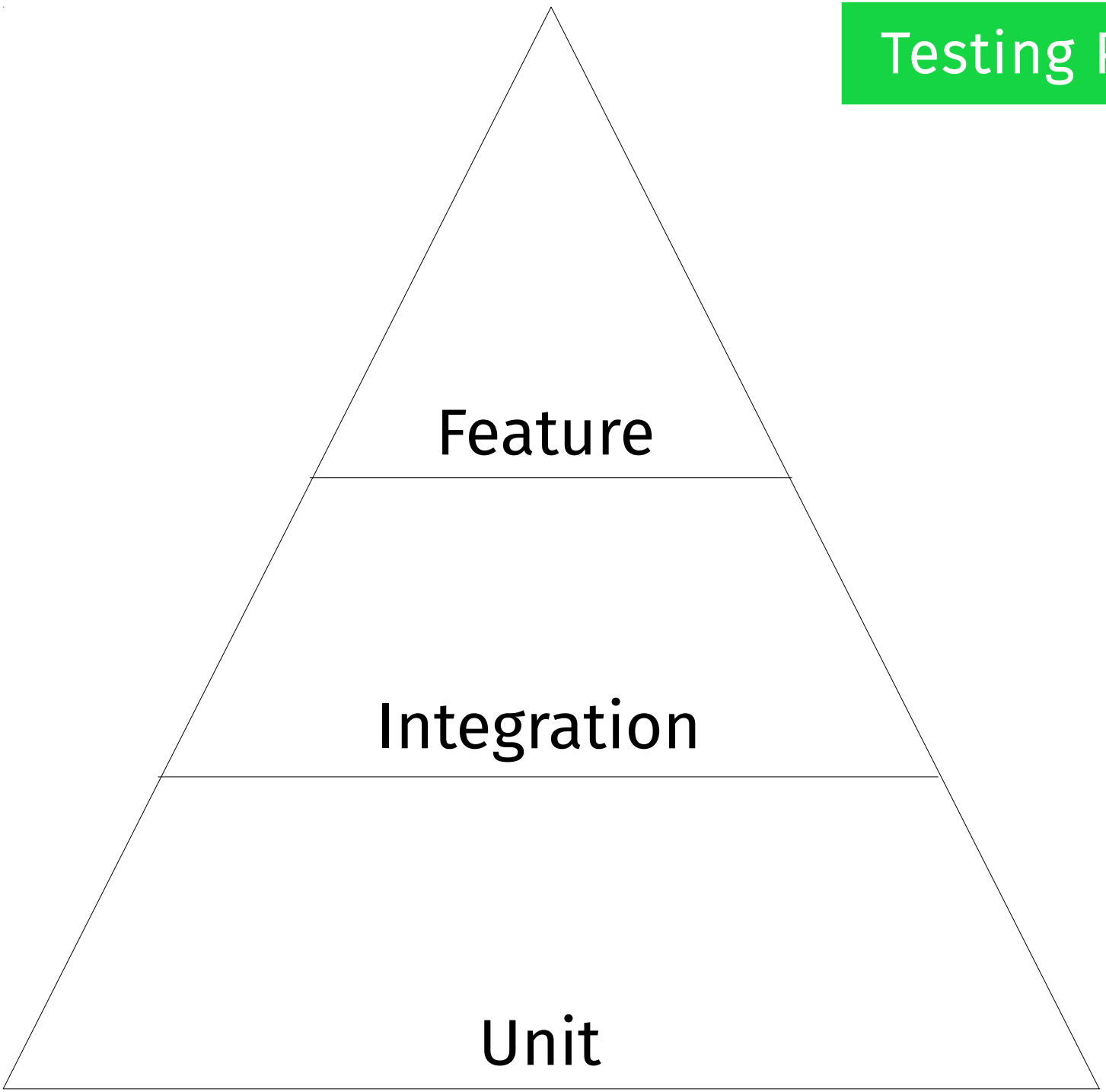
Replying to @chastell @PragTob @devoncestes

literally slide number eight:

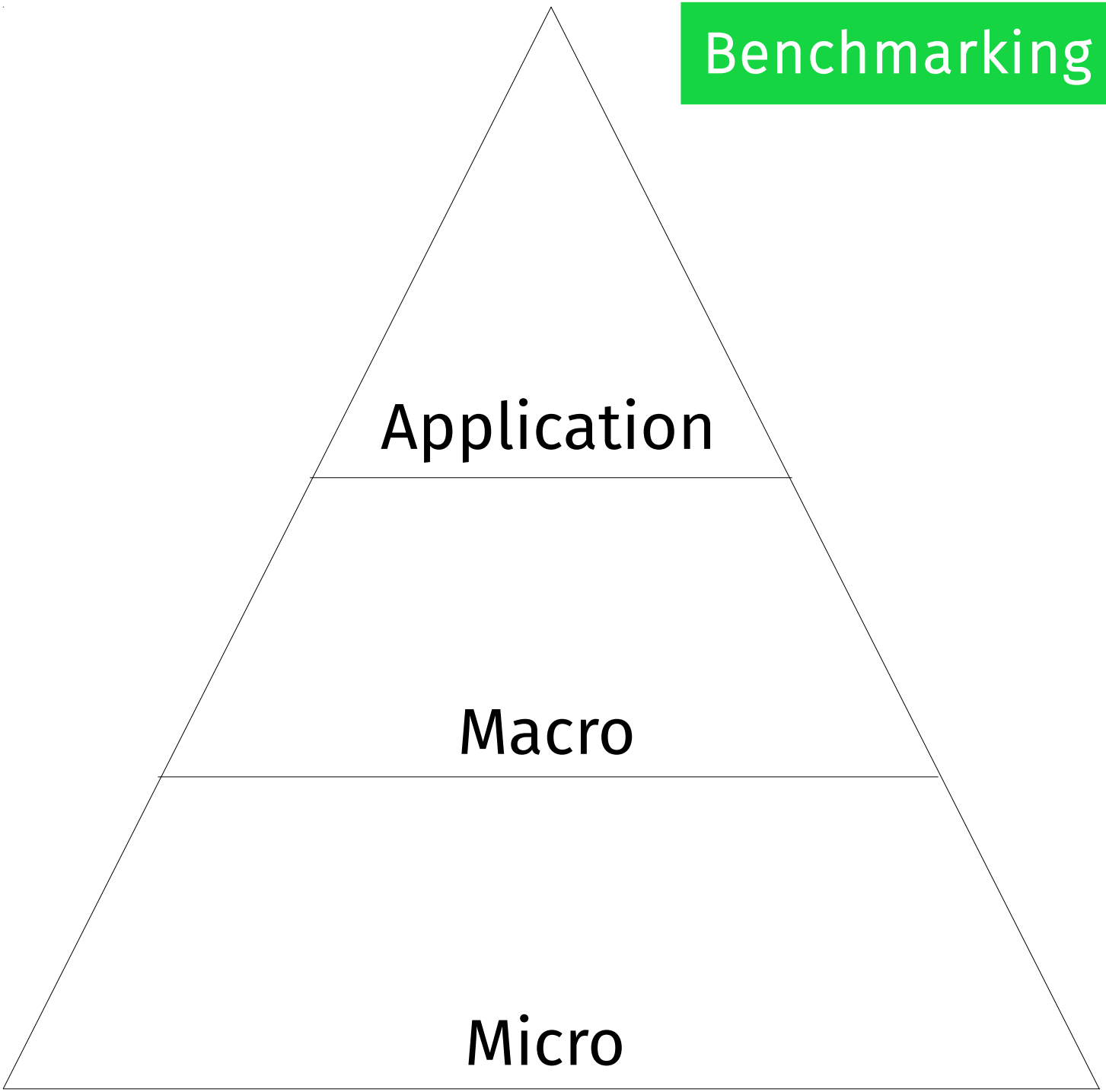
```
Warming up --------------------------------------
      exception miss   441.174k i/100ms
       exception hit   114.559k i/100ms
              SELECT   522.000  i/100ms
              INSERT    21.000  i/100ms
Calculating -------------------------------------
      exception miss     14.669M (± 2.7%) i/s -     73.676M in   5.027647s
       exception hit      1.435M (± 1.6%) i/s -      7.217M in   5.031332s
              SELECT      4.855k (± 3.8%) i/s -     24.534k in   5.060903s
              INSERT    224.948  (± 9.8%) i/s -      1.134k in   5.084542s

Comparison:
      exception miss: 14668954.7 i/s
       exception hit:  1434854.2 i/s - 10.22x  slower
              SELECT:     4855.2 i/s - 3021.30x  slower
              INSERT:      224.9 i/s - 65210.36x  slower
```

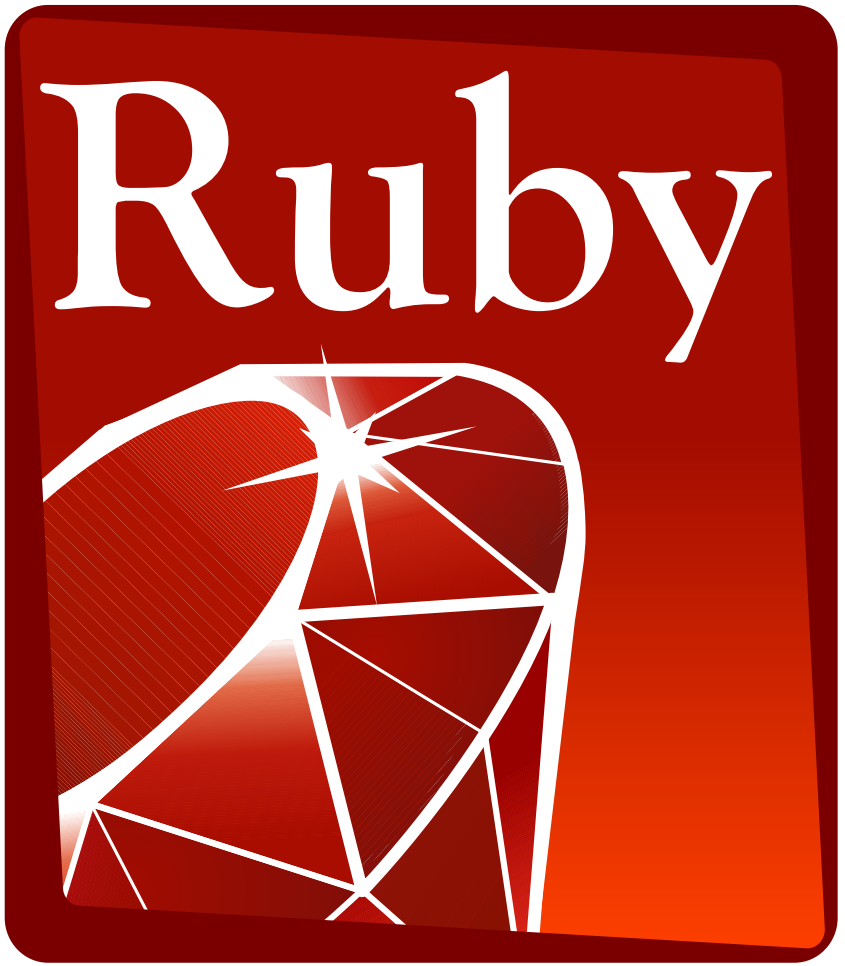
Types of benchmarks

Testing Pyramid



Benchmarking Pyramid





PROGRAMMING

Language

micro: scoring

```ruby
Benchmark.ips do |benchmark|
  game_19 = playout_for(19).game_state.game
  scorer  = Rubykon::GameScorer.new

  benchmark.report '19x19 scoring' do
    scorer.score game_19
  end
end
```

macro: playout

```
Benchmark.ips do |benchmark|
  benchmark.report '19x19 playout' do
    game       = Rubykon::Game.new(19)
    game_state = Rubykon::GameState.new(game)
    mcts       = MCTS::Playout.new(game_state)
    mcts.playout
  end
end
```

Application: tree search

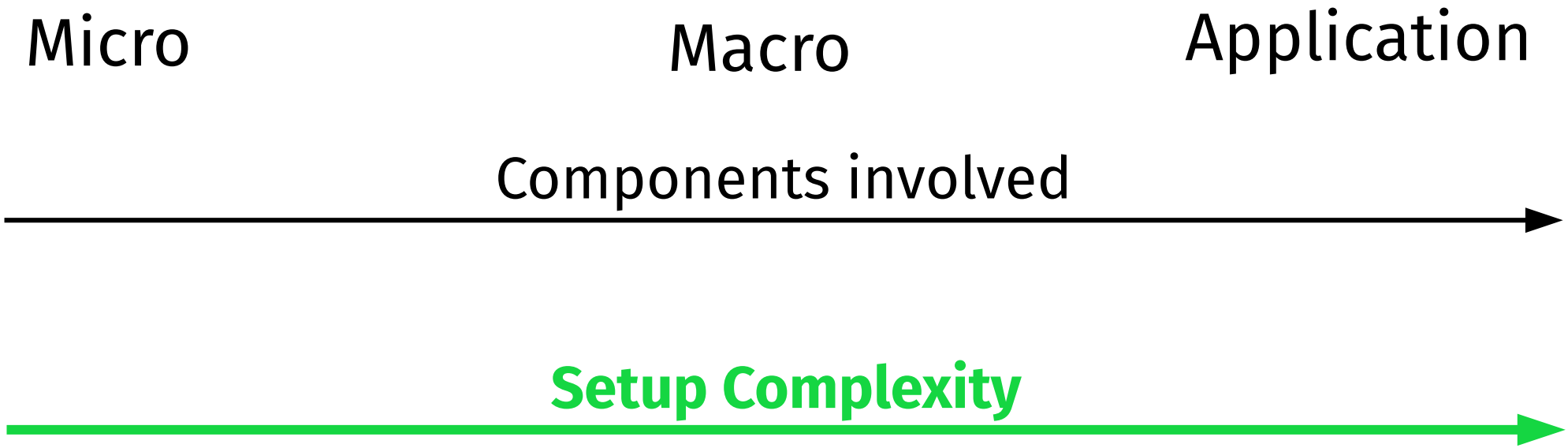
```ruby
Benchmark.avg do |benchmark|
  game_19       = Rubykon::Game.new(19)
  game_state_19 = Rubykon::GameState.new game_19
  mcts          = MCTS::MCTS.new

  benchmark.config warmup: 180, time: 180

  benchmark.report "19x19 1_000 iterations" do
    mcts.start game_state_19, 1_000
  end
end
```

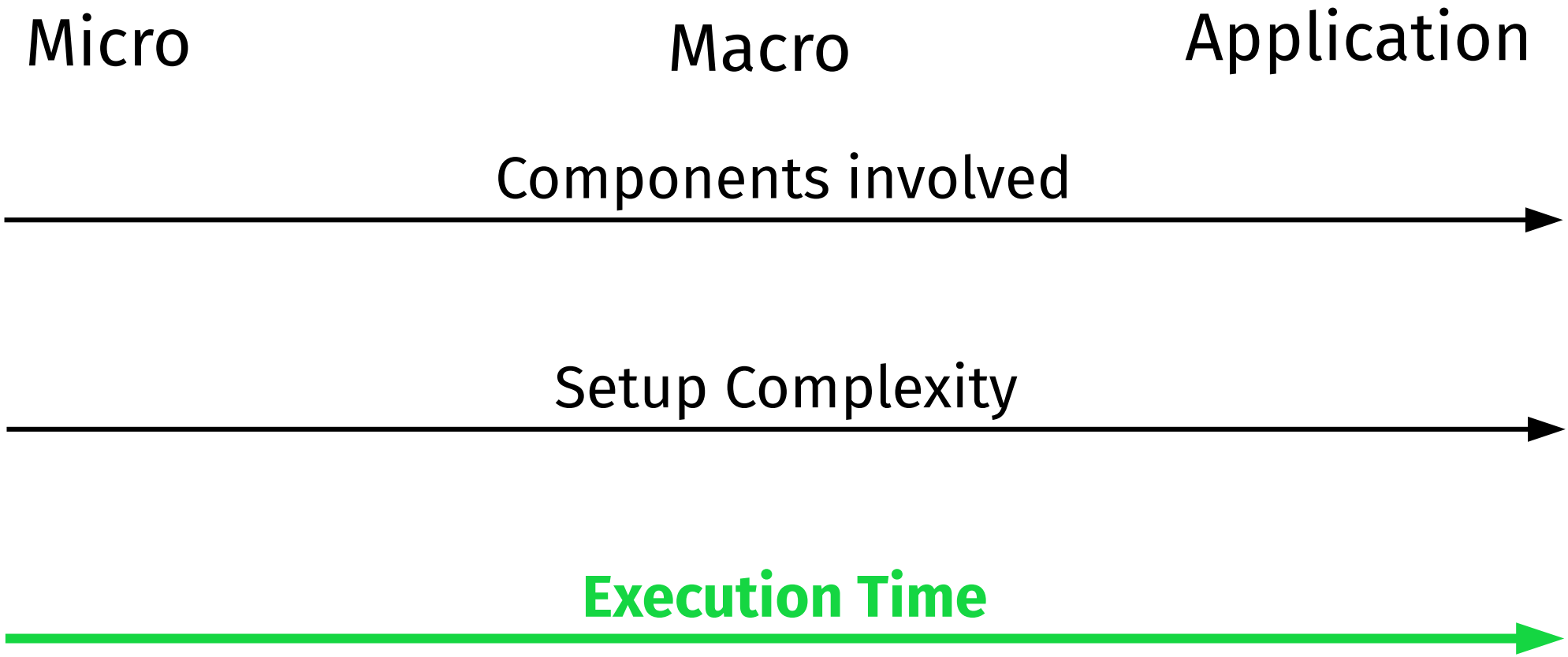
Micro

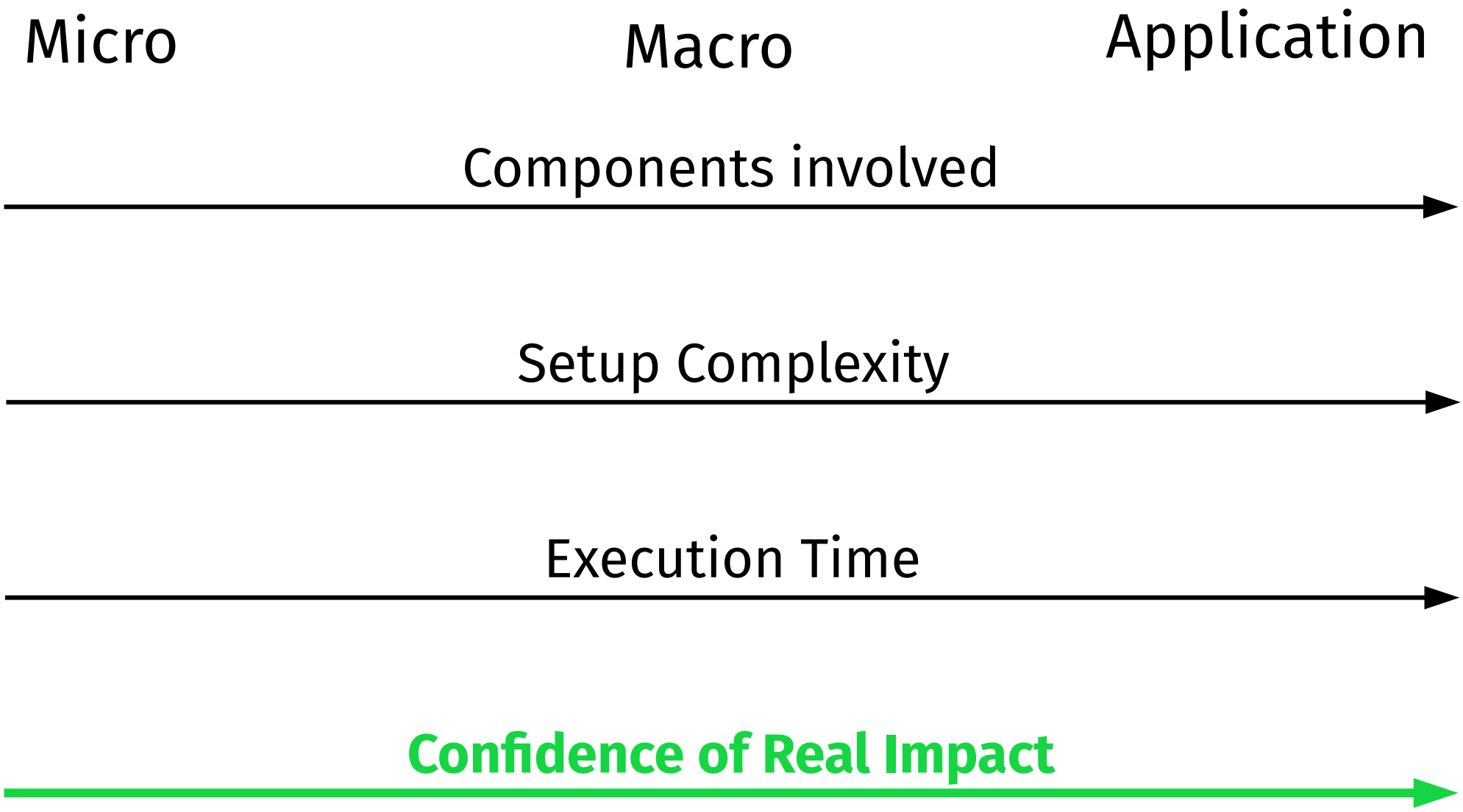
Macro

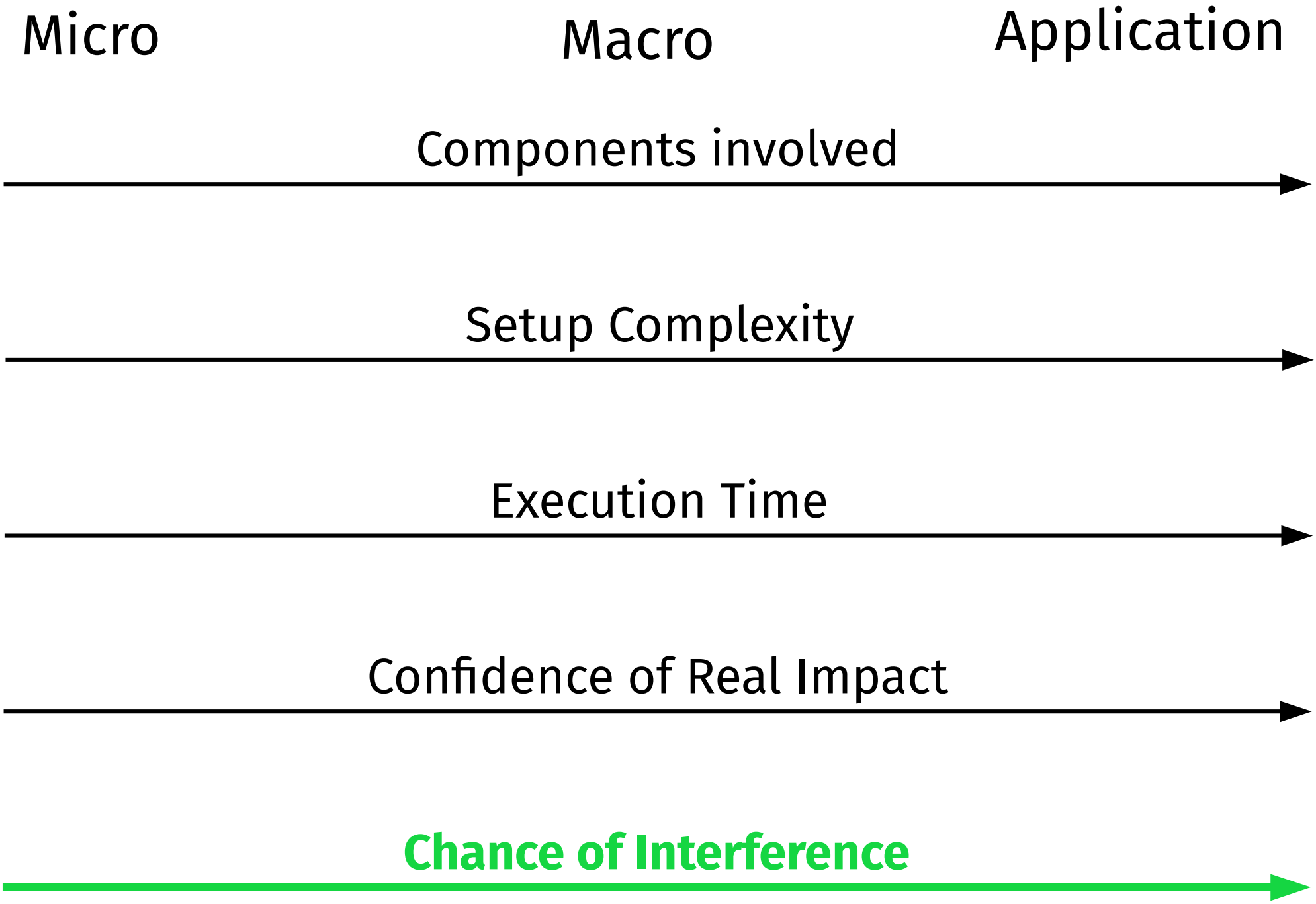
Application

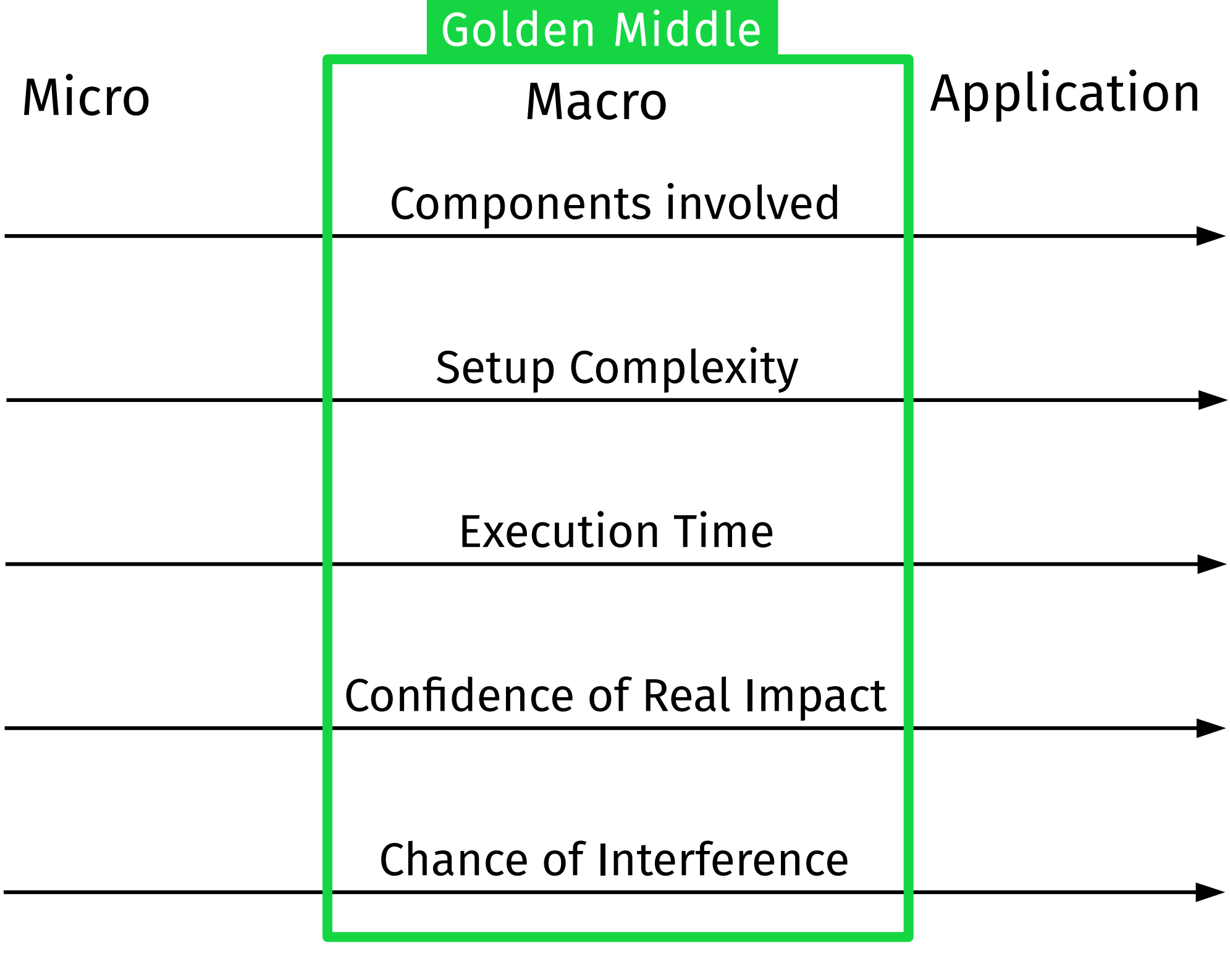

Micro
Macro
Application
Components involved


Micro
Macro
Application
Components involved
Setup Complexity


Micro
Macro
Application
Components involved
Setup Complexity
Execution Time


Micro
Macro
Application
Components involved
Setup Complexity
Execution Time
Confidence of Real Impact

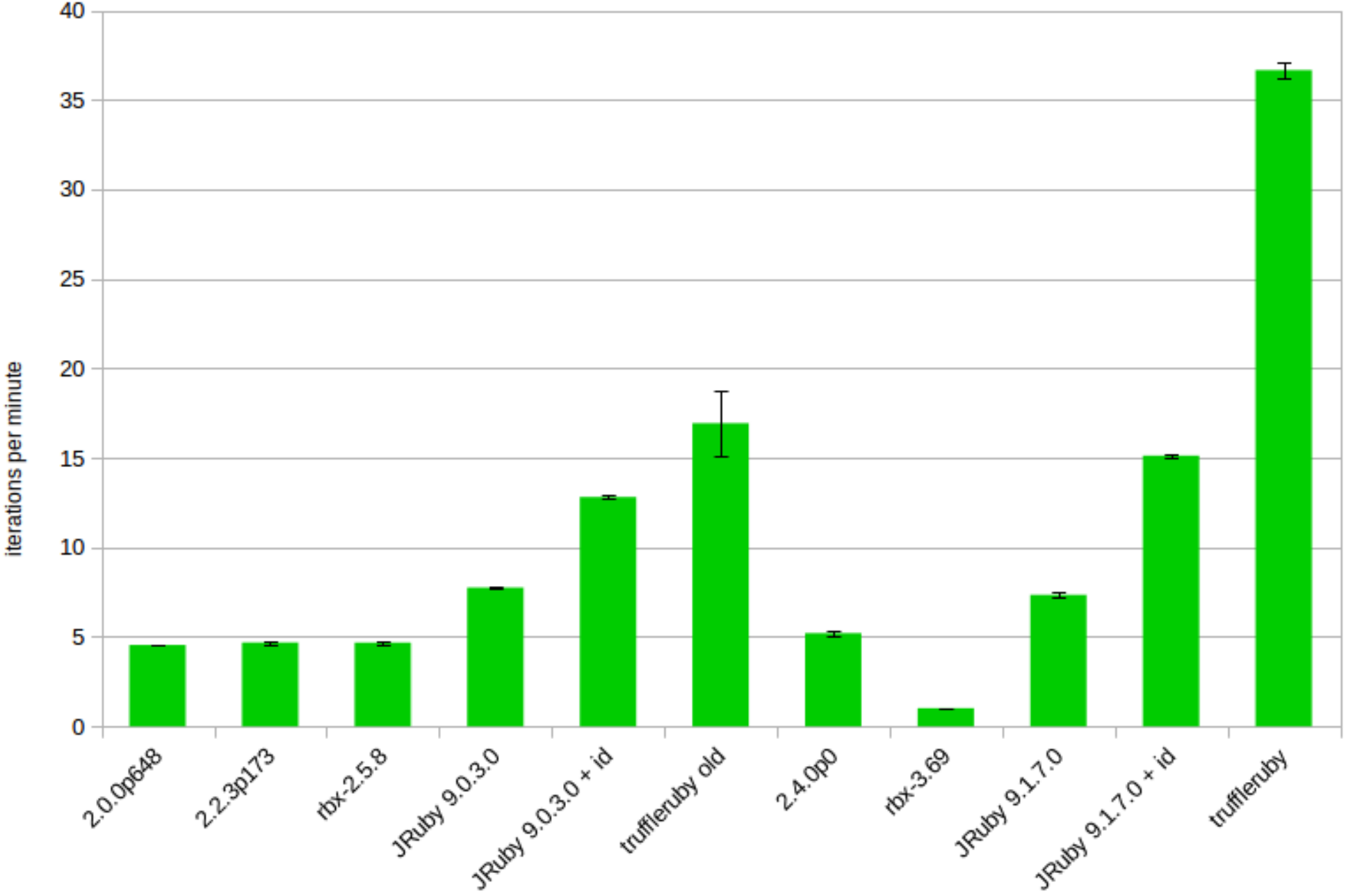

Micro
Macro
Application
Components involved
Setup Complexity
Execution Time
Confidence of Real Impact
Chance of Interference


Golden Middle
Micro
Macro
Application
Components involved
Setup Complexity
Execution Time
Confidence of Real Impact
Chance of Interference

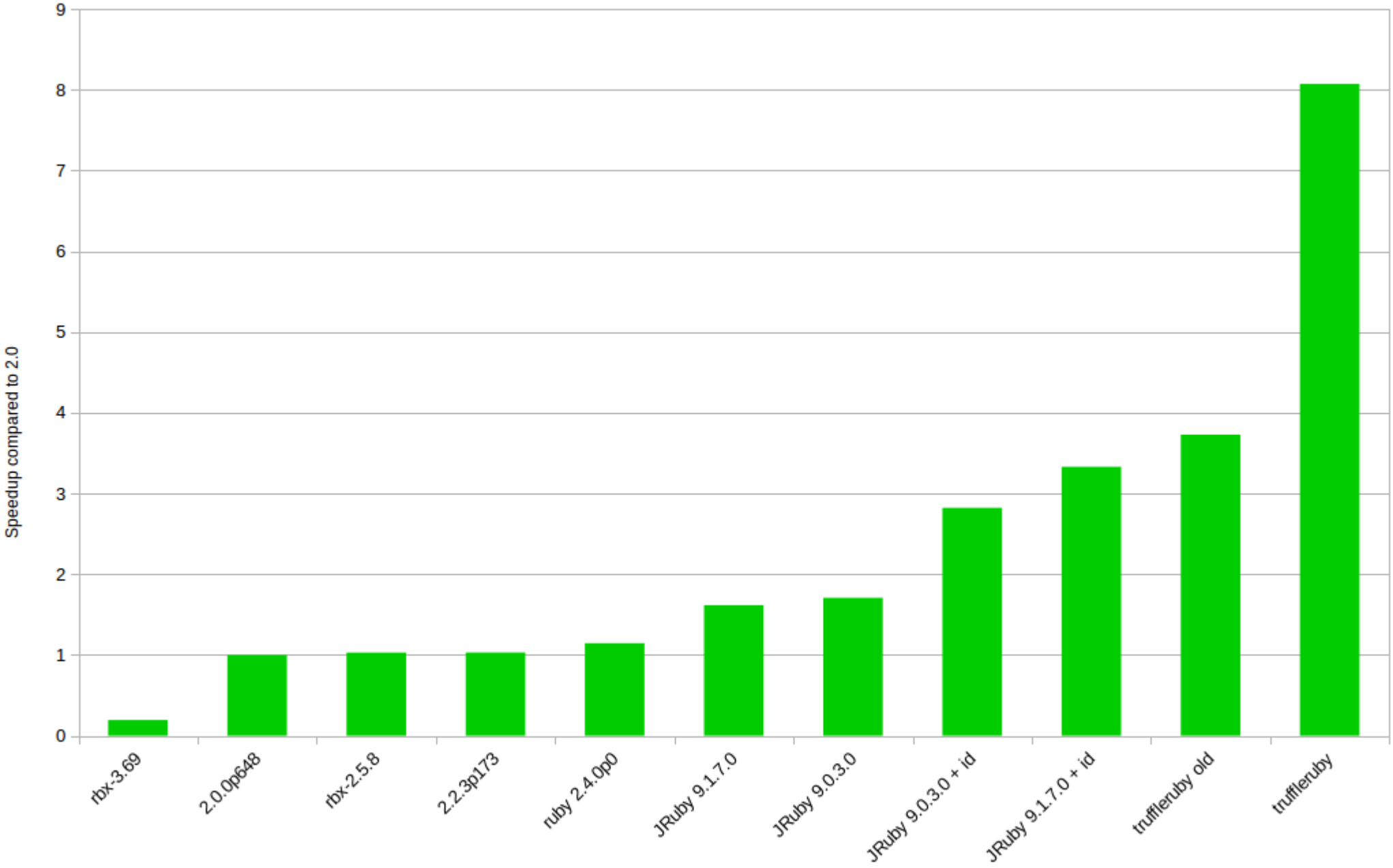
Application: tree search

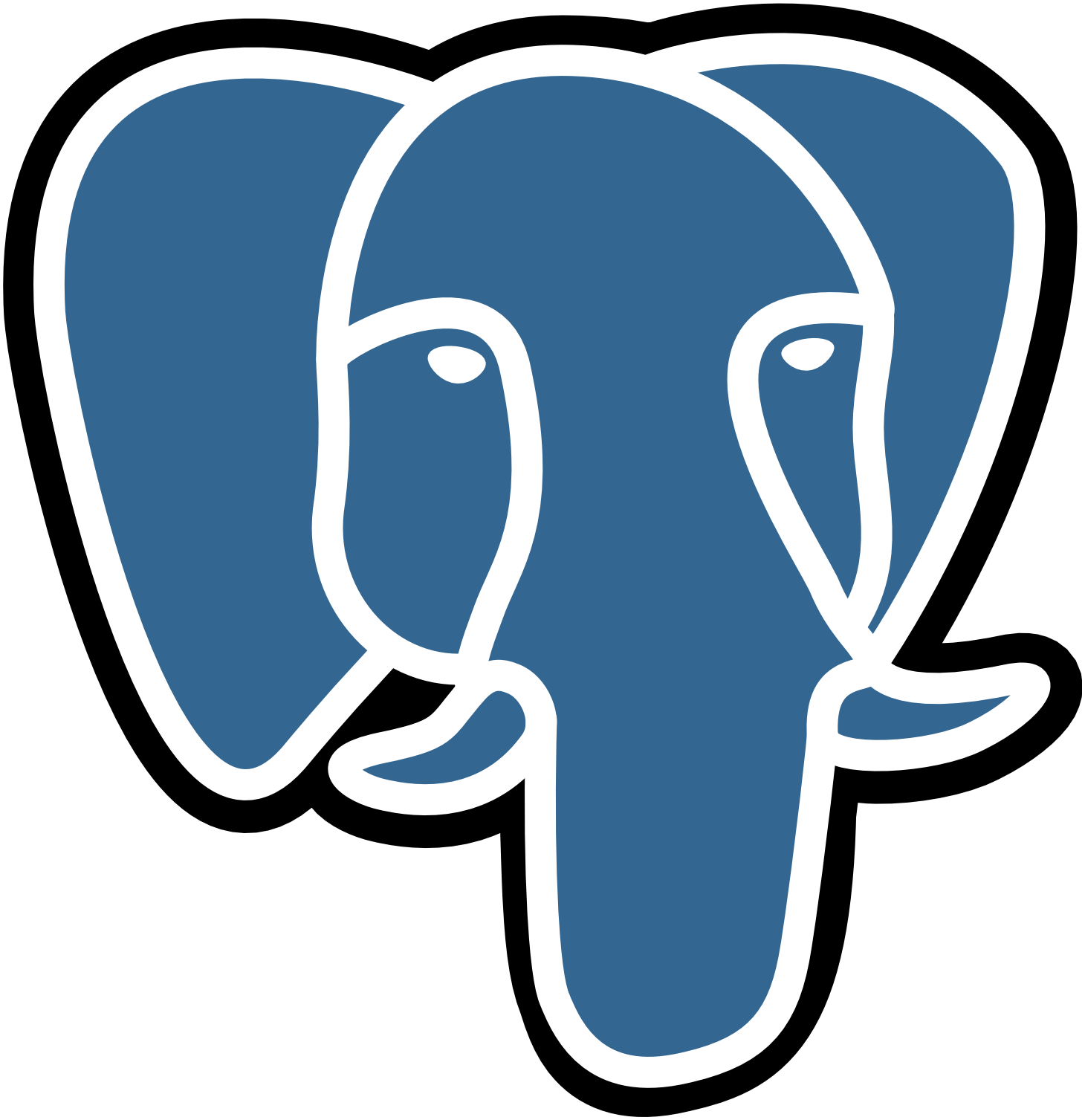
```ruby
Benchmark.avg do |benchmark|
  game_19       = Rubykon::Game.new(19)
  game_state_19 = Rubykon::GameState.new game_19
  mcts          = MCTS::MCTS.new

  benchmark.config warmup: 180, time: 180

  benchmark.report "19x19 1_000 iterations" do
    mcts.start game_state_19, 1_000
  end
end
```

Great ruby rumble

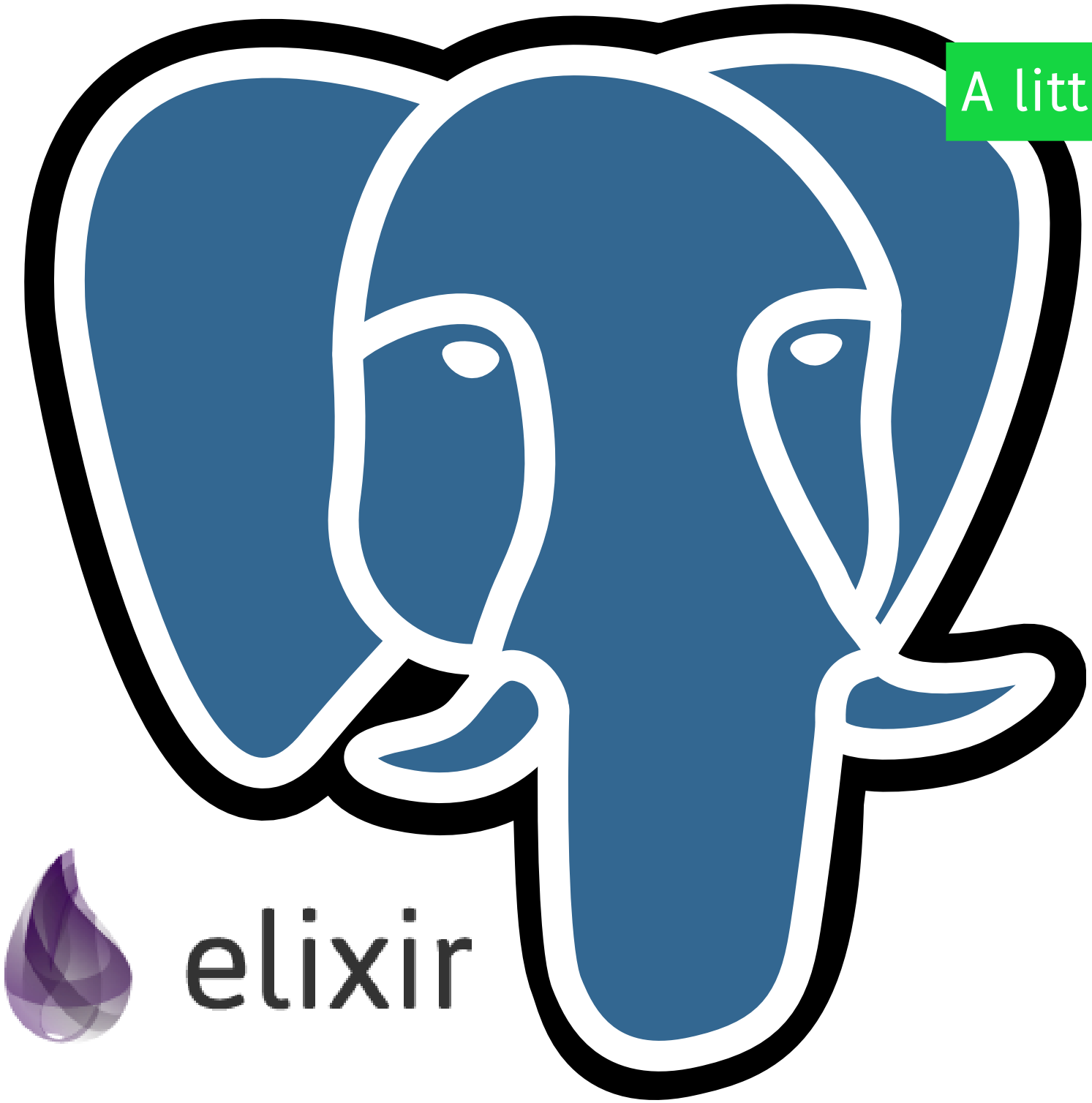


Speedup relative to 2.0





A little help...



elixir

Story Time

Rossella i Albert Caller zaprowadzili Julię i Ricka przed bogato intarsjowaną komodę, na której ustawiono rozmaite cenne przedmioty: dwie statuetki z Chin, z epoki Ming, sztylet toledański i szkatułkę na biżuterię ze Smyrny.

– Nie widzę żadnej pozytywki – odezwała się Julia, rozglądając się wokół z zaciekawieniem.

– Oczywiście, bo jest odrobinę... oryginalna – mruknął Albert, sięgając po krzesło. Zsunął z nóg mokasyny i wszedł na krzesło. Zdjął ze ściany mały obrazek wiszący nad komodą.

– Rick! – wykrzyknęła Julia, rozpoznając namalowany dom i ogród. – Czy to nie jest Willa Argo?

– Co mówisz?

– To jest dom, w którym mieszkamy! – wyjaśniła dziewczynka. – To jest park, urwisko... a tu jest furtka.

– Doprawdy? – spytała Rossella. – Pokaż im ramę, Albercie.

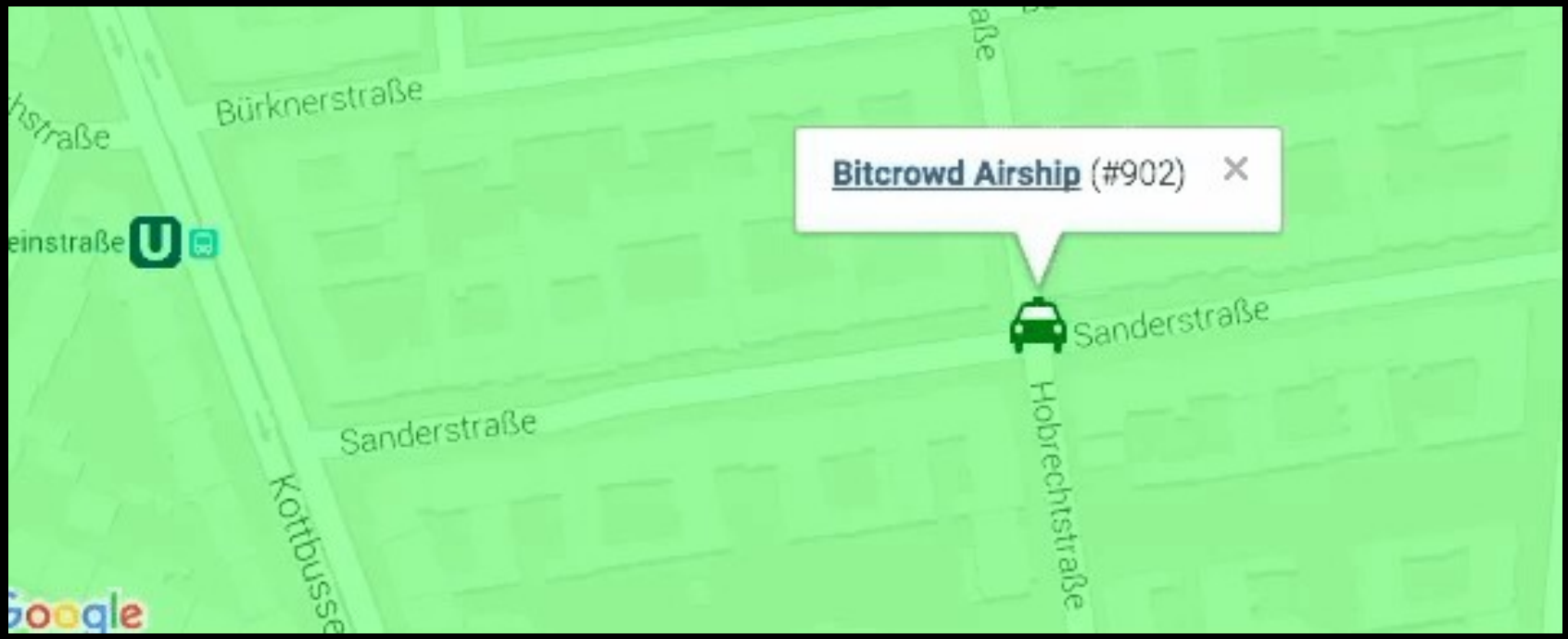
Mężczyzna odwrócił obraz, pokazując dzieciom korbkę wmontowaną w złote ramy. Na metalowym cylinderku była wyryta sowa, znak Petera. Za pomocą koła zębatego korbka łączyła się z metalowym cylinderkiem, najeżonym maleńkimi, metalowymi kołeczkami.

– Zaraz wam puszczę... – szepnął Albert i pokręcił korbką. Kołeczki zaczęły trącać lekko w maleńkie, metalowe pręciki, wydające dźwięki, które układały się w uroczą melodyjkę. Słuchając tych dźwięków, Rick poczuł się nagle tak, jakby przeniesiono go w czasy dzieciństwa. To była ta sama melodia,

...rog usłyszał wiele lat temu, kiedy wes...
...ciera Dedalusa, wtedy gdy dostał sw... z zegarek. Nie miał
...pliwości: ta melodia zapadła mu głęboko w pamięć.
...Wszystko w porządku, chłopcze? – spytała go Rossella.
...otrząsnął się ze swoich wspomnień. Dopiero teraz
...ł, że pozytywka przestała już grać, a Julia i Callero-
...dają mu się.
...?...
...rozpaloną, oczy mu błyszczały jak zawsze wte-
...o swoim ojcu.
...stko w porządku? – Julia wzięła go za rękę.
...wierdził rudzielec, wskazując na pozytywkę.
...Rozpoznałem tę melodię.
...ę, że to cud – stwierdził Albert Caller, usta-
...wiają...braz na podłodze. – Nie wiedziałbym, od czego za-
...cząć poszukiwania. Nie jest łatwo znaleźć człowieka w tak
wielkim mieście jak Wenecja, i mając z tak znikome infor-
macje.
– Próbowaliście już rozmawiać z rzemieślnikami z ulicy
zegarmistrzów? – spytała Rossella.
– Tak, ale wygląda na to, że nikt z nich nic nie wie.
– Jesteście pewni, że ten Peter nie używa przez
czy czegoś w tym rodzaju? – zasugerował Albert.
– Istotnie, mógłby używać innego nazwiska, żeby
kojniejszym...


Boom


Boom
Elixir.DBConnection.ConnectionError


Bürknerstraße
Bitcrowd Airship (#902)
Sanderstraße
Sanderstraße
Hobrechtstraße
Kottbusser
Google

A real case

```
Benchee.run %{
  "Using LatestCourierLocation" => fn(courier_id) ->
    LatestCourierLocation
    |> CourierLocation.with_courier_ids(courier_id)
    |> Repo.one
  end,
  "with_courier_ids + order" => fn(courier_id) ->
    CourierLocation.with_courier_ids(courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end,
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}
```

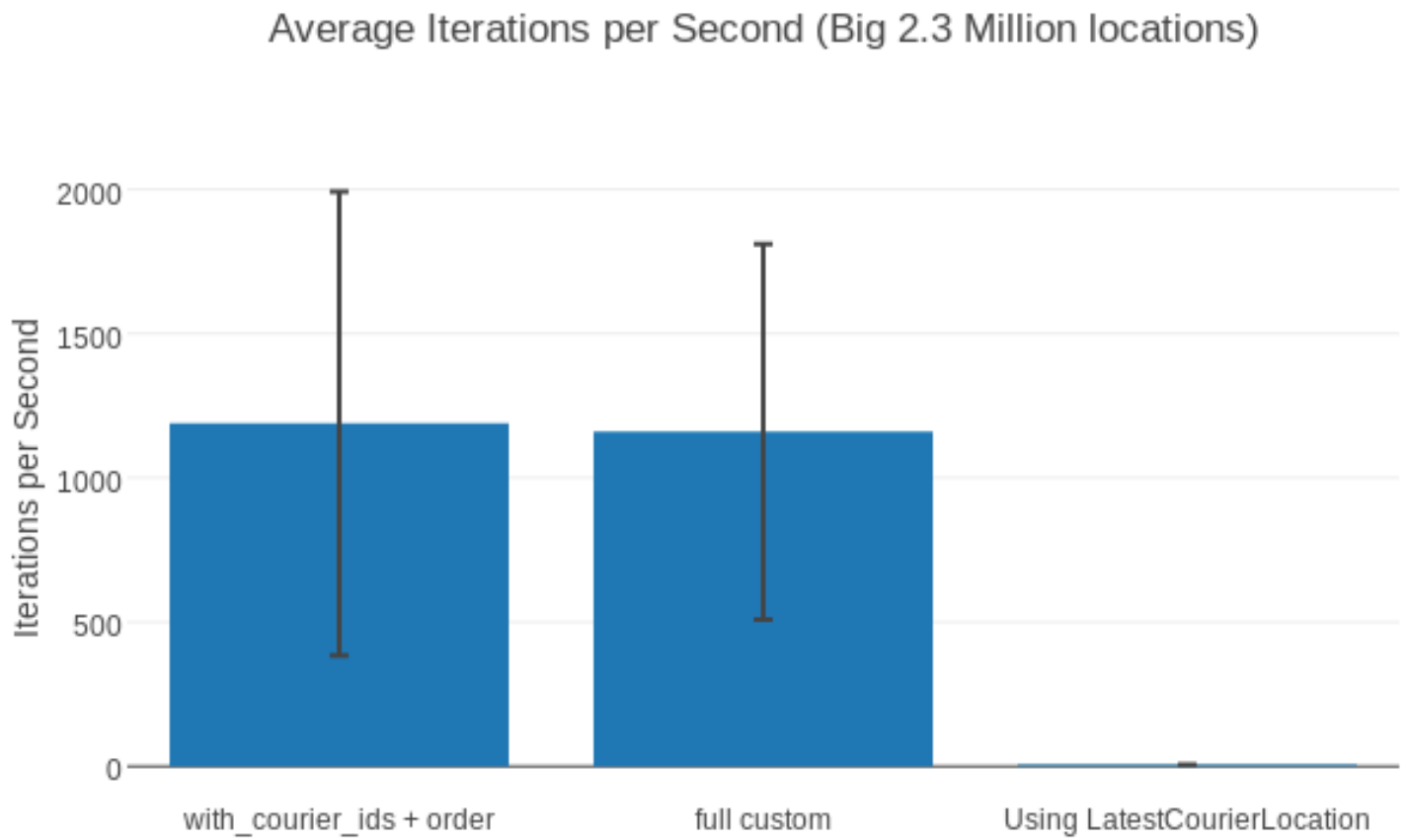
Database View

```
Benchee.run %{
  "Using LatestCourierLocation" => fn(courier_id) ->
    LatestCourierLocation
    |> CourierLocation.with_courier_ids(courier_id)
    |> Repo.one
  end,
  "with_courier_ids + order" => fn(courier_id) ->
    CourierLocation.with_courier_ids(courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end,
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}
```

Only difference

```
Benchee.run %{
  "Using LatestCourierLocation" => fn(courier_id) ->
    LatestCourierLocation
    |> CourierLocation.with_courier_ids(courier_id)
    |> Repo.one
  end,
  "with courier_ids + order" => fn(courier_id) ->
    CourierLocation.with_courier_ids(courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end,
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}
```

Another job well done?




```
Name                                  ips        average  deviation         median
with_courier_ids + order           1.19 K      841.44 µs    ±67.64%      675.00 µs
full custom                        1.16 K      862.36 µs    ±56.06%      737.00 µs
Using LatestCourierLocation     0.00603 K   165897.47 µs     ±2.33%   165570.00 µs

Comparison:
with_courier_ids + order           1.19 K
full custom                        1.16 K - 1.02x slower
Using LatestCourierLocation     0.00603 K - 197.16x slower
```


Boom


Boom
Elixir.DBConnection.ConnectionError


Boom
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError


Boom
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError
Elixir.DBConnection.ConnectionError

```
inputs = %{
  "Big 2.3 Million locations" => 3799,
  "No locations"              => 8901,
  "~200k locations"           => 4238,
  "~20k locations"            => 4201
}

Benchee.run %{
  ...
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}, inputs: inputs, time: 25, warmup: 5
```

Inputs to the rescue!

```
inputs = %{
  "Big 2.3 Million locations" => 3799,
  "No locations"              => 8901,
  "~200k locations"           => 4238,
  "~20k locations"            => 4201
}

Benchee.run %{
  ...
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}, inputs: inputs, time: 25, warmup: 5
```

Inputs to the rescue!

```elixir
inputs = %{
  "Big 2.3 Million locations" => 3799,
  "No locations"              => 8901,
  "~200k locations"           => 4238,
  "~20k locations"            => 4201
}

Benchee.run %{
  ...
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}, inputs: inputs, time: 25, warmup: 5
```

Inputs to the rescue!

```
inputs = %{
  "Big 2.3 Million locations" => 3799,
  "No locations"              => 8901,
  "~200k locations"           => 4238,
  "~20k locations"            => 4201
}

Benchee.run %{
  ...
  "full custom" => fn(courier_id) ->
    CourierLocation
    |> Ecto.Query.where(courier_id: ^courier_id)
    |> Ecto.Query.order_by(desc: :time)
    |> Ecto.Query.limit(1)
    |> Repo.one
  end
}, inputs: inputs, time: 25, warmup: 5
```

Inputs to the rescue!

Old

```
##### With input Big 2.3 Million locations #####
Comparison:
with_courier_ids + order           1.19 K
full custom                        1.16 K - 1.02x slower
Using LatestCourierLocation     0.00603 K - 197.16x slower

##### With input ~200k locations #####
Comparison:
Using LatestCourierLocation         3.66
full custom                        0.133 - 27.57x slower
with_courier_ids + order           0.132 - 27.63x slower

##### With input ~20k locations #####
Comparison:
Using LatestCourierLocation         38.12
full custom                        0.122 - 312.44x slower
with_courier_ids + order           0.122 - 313.33x slower

##### With input No locations #####
Comparison:
Using LatestCourierLocation       2967.48
full custom                         0.114 - 25970.57x slower
with_courier_ids + order            0.114 - 26046.06x slower
```

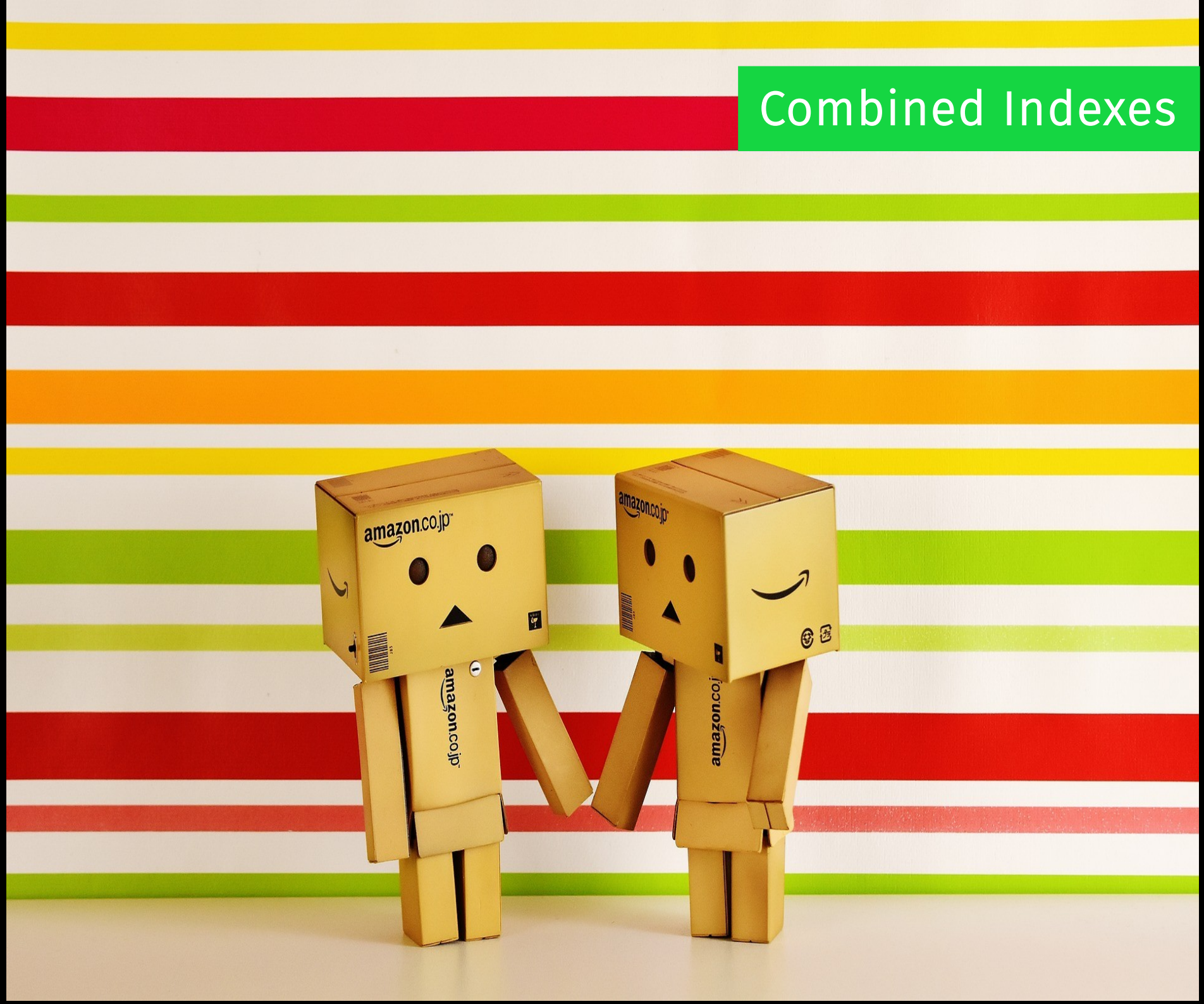
Old

```
##### With input Big 2.3 Million locations #####
Comparison:
with_courier_ids + order                1.19 K
full custom                             1.16 K - 1.02x slower
Using LatestCourierLocation    0.00603 K - 197.16x slower

##### With input ~200k locations #####
Comparison:
Using LatestCourierLocation              3.66
full custom                             0.133 - 27.57x slower
with_courier_ids + order                0.132 - 27.63x slower

##### With input ~20k locations #####
Comparison:
Using LatestCourierLocation             38.12
full custom                             0.122 - 312.44x slower
with_courier_ids + order                0.122 - 313.33x slower

##### With input No locations #####
Comparison:
Using LatestCourierLocation           2967.48
full custom                             0.114 - 25970.57x slower
with_courier_ids + order                0.114 - 26046.06x slower
```

Old

```
##### With input Big 2.3 Million locations #####
Comparison:
with_courier_ids + order              1.19 K
full custom                           1.16 K - 1.02x slower
Using LatestCourierLocation         0.00603 K - 197.16x slower

##### With input ~200k locations #####
Comparison:
Using LatestCourierLocation             3.66
full custom                           0.133 - 27.57x slower
with_courier_ids + order              0.132 - 27.63x slower

##### With input ~20k locations #####
Comparison:
Using LatestCourierLocation             38.12
full custom                           0.122 - 312.44x slower
with_courier_ids + order              0.122 - 313.33x slower

##### With input No locations #####
Comparison:
Using LatestCourierLocation           2967.48
full custom                            0.114 - 25970.57x slower
with_courier_ids + order               0.114 - 26046.06x slower
```

Combined Indexes



Combined Index

```
##### With input Big 2.3 Million locations #####
Comparison:
full custom                          3921.12
with_courier_ids + order               23.05 - 170.09x slower
Using LatestCourierLocation             5.98 - 655.74x slower

##### With input ~200k locations #####
Comparison:
full custom                          4272.84
with_courier_ids + order               14.20 - 300.91x slower
Using LatestCourierLocation             3.80 - 1125.59x slower

##### With input ~20k locations #####
Comparison:
full custom                          3792.97
with_courier_ids + order               78.93 - 48.06x slower
Using LatestCourierLocation            35.62 - 106.47x slower

##### With input No locations #####
Comparison:
full custom                           5.14 K
with_courier_ids + order              3.87 K - 1.33x slower
Using LatestCourierLocation           3.29 K - 1.56x slower
```

Combined Index

```
##### With input Big 2.3 Million locations #####
Comparison:
full custom                          3921.12
with_courier_ids + order               23.05 - 170.09x slower
Using LatestCourierLocation             5.98 - 655.74x slower

##### With input ~200k locations #####
Comparison:
full custom                          4272.84
with_courier_ids + order               14.20 - 300.91x slower
Using LatestCourierLocation             3.80 - 1125.59x slower

##### With input ~20k locations #####
Comparison:
full custom                          3792.97
with_courier_ids + order               78.93 - 48.06x slower
Using LatestCourierLocation            35.62 - 106.47x slower

##### With input No locations #####
Comparison:
full custom                           5.14 K
with_courier_ids + order            3.87 K - 1.33x slower
Using LatestCourierLocation         3.29 K - 1.56x slower
```

Insertion Time

```
# with an index on courier_id and one on time

Name                           ips        average  deviation         median
Updating a location         366.90        2.73 ms    ±36.35%        2.29 ms

# with a combined index on courier_id and time

Name                           ips        average  deviation         median
Updating a location         283.41        3.53 ms    ±52.18%        2.77 ms
```


Inputs matter!

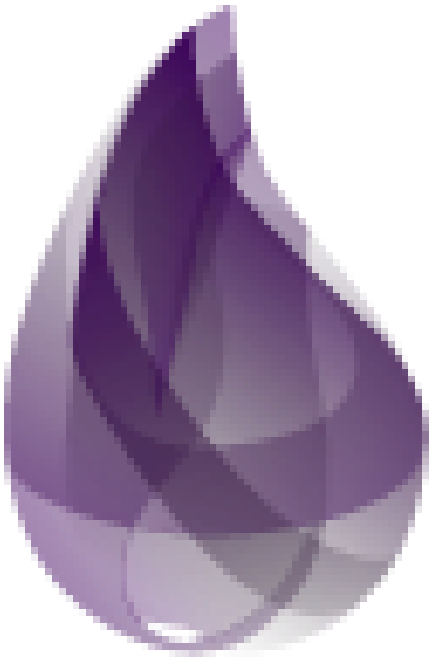

Interference free Environment


Correct & Meaningful Setup


RAILS_ENV=performance

Logging & Friends

```
[info] GET /
[debug] Processing by Rumbl.PageController.index/2
  Parameters: %{}
  Pipelines: [:browser]
[info] Sent 200 in 46ms
[info] GET /sessions/new
[debug] Processing by Rumbl.SessionController.new/2
  Parameters: %{}
  Pipelines: [:browser]
[info] Sent 200 in 5ms
[info] GET /users/new
[debug] Processing by Rumbl.UserController.new/2
  Parameters: %{}
  Pipelines: [:browser]
[info] Sent 200 in 7ms
[info] POST /users
[debug] Processing by Rumbl.UserController.create/2
  Parameters: %{"_csrf_token" =>
"NUEUdRMNAiBfIHEeNwZkfA05PgAOJgAAf0ACXJqCjl7YojW+trdjdg==", "_utf8" => "✓", "user" =>
%{"name" => "asdasd", "password" => "[FILTERED]", "username" => "Homer"}}
  Pipelines: [:browser]
[debug] QUERY OK db=0.1ms
begin []
[debug] QUERY OK db=0.9ms
INSERT INTO "users" ("name","password_hash","username","inserted_at","updated_at") VALUES
($1,$2,$3,$4,$5) RETURNING "id" ["asdasd",
"$2b$12$.qY/kpo0Dec7vMK1ClJoC.Lw77c3oGllX7uieZILMlFh2hFpJ3F.C", "Homer", {{2016, 12,
2}, {14, 10, 28, 0}}, {{2016, 12, 2}, {14, 10, 28, 0}}]
```



elixir

Tail Call Optimization

TCO

```
defmodule MyMap do
  def map_tco(list, function) do
    Enum.reverse do_map_tco([], list, function)
  end

  defp do_map_tco(acc, [], _function) do
    acc
  end
  defp do_map_tco(acc, [head | tail], func) do
    do_map_tco([func.(head) | acc], tail, func)
  end

  def map_body([], _func), do: []
  def map_body([head | tail], func) do
    [func.(head) | map_body(tail, func)]
  end
end
```

TCO

```elixir
defmodule MyMap do
  def map_tco(list, function) do
    Enum.reverse do_map_tco([], list, function)
  end

  defp do_map_tco(acc, [], _function) do
    acc
  end
  defp do_map_tco(acc, [head | tail], func) do
    do_map_tco([func.(head) | acc], tail, func)
  end

  def map_body([], _func), do: []
  def map_body([head | tail], func) do
    [func.(head) | map_body(tail, func)]
  end
end
```

TCO

```
defmodule MyMap do
  def map_tco(list, function) do
    Enum.reverse do_map_tco([], list, function)
  end

  defp do_map_tco(acc, [], _function) do
    acc
  end
  defp do_map_tco(acc, [head | tail], func) do
    do_map_tco([func.(head) | acc], tail, func)
  end

  def map_body([], _func), do: []
  def map_body([head | tail], func) do
    [func.(head) | map_body(tail, func)]
  end
end
```

TCO

```
map_fun = fn(i) -> i + 1 end
inputs = %{
  "Small (10 Thousand)"   => Enum.to_list(1..10_000),
  "Middle (100 Thousand)" => Enum.to_list(1..100_000),
  "Big (1 Million)"       => Enum.to_list(1..1_000_000),
  "Bigger (5 Million)"    => Enum.to_list(1..5_000_000)
}

Benchee.run %{
  "tail-recursive" =>
    fn(list) -> MyMap.map_tco(list, map_fun) end,
  "stdlib map" =>
    fn(list) -> Enum.map(list, map_fun) end,
  "body-recursive" =>
    fn(list) -> MyMap.map_body(list, map_fun) end
}, inputs: inputs
```

TCO

```elixir
map_fun = fn(i) -> i + 1 end
inputs = %{
  "Small (10 Thousand)"   => Enum.to_list(1..10_000),
  "Middle (100 Thousand)" => Enum.to_list(1..100_000),
  "Big (1 Million)"       => Enum.to_list(1..1_000_000),
  "Bigger (5 Million)"    => Enum.to_list(1..5_000_000)
}

Benchee.run %{
  "tail-recursive" =>
    fn(list) -> MyMap.map_tco(list, map_fun) end,
  "stdlib map" =>
    fn(list) -> Enum.map(list, map_fun) end,
  "body-recursive" =>
    fn(list) -> MyMap.map_body(list, map_fun) end
}, inputs: inputs
```

TCO

```
map_fun = fn(i) -> i + 1 end
inputs = %{
  "Small (10 Thousand)"   => Enum.to_list(1..10_000),
  "Middle (100 Thousand)" => Enum.to_list(1..100_000),
  "Big (1 Million)"       => Enum.to_list(1..1_000_000),
  "Bigger (5 Million)"    => Enum.to_list(1..5_000_000)
}

Benchee.run %{
  "tail-recursive" =>
    fn(list) -> MyMap.map_tco(list, map_fun) end,
  "stdlib map" =>
    fn(list) -> Enum.map(list, map_fun) end,
  "body-recursive" =>
    fn(list) -> MyMap.map_body(list, map_fun) end
}, inputs: inputs
```

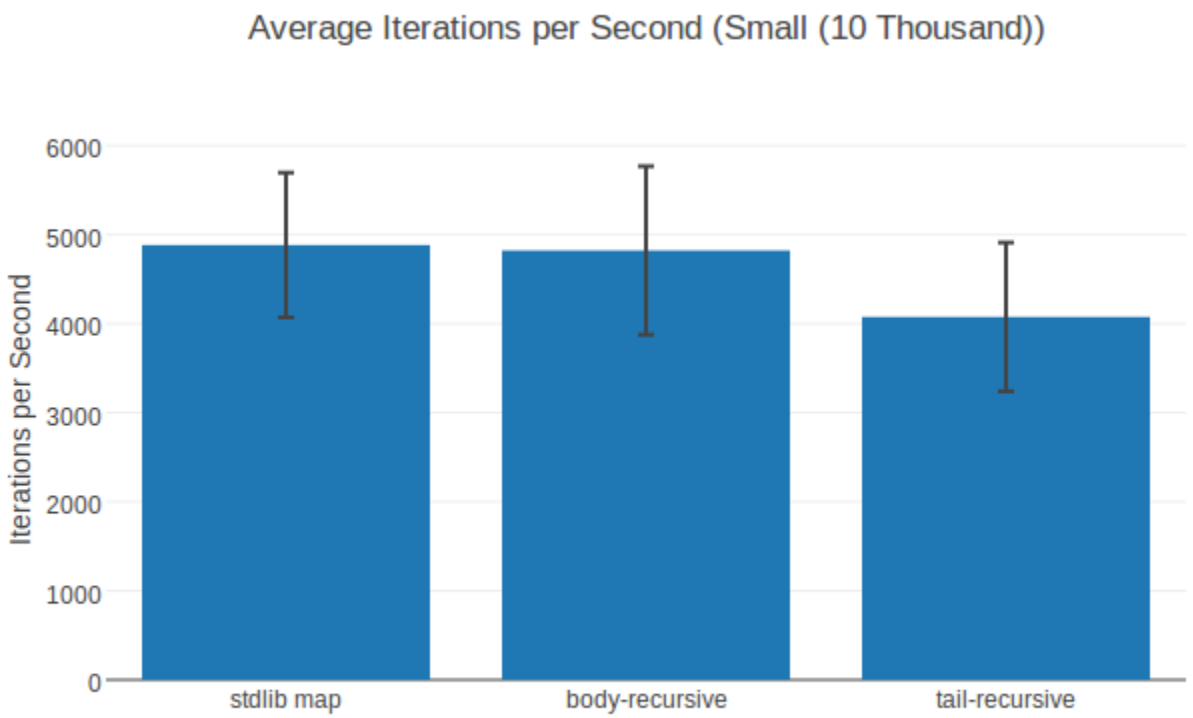
TCO

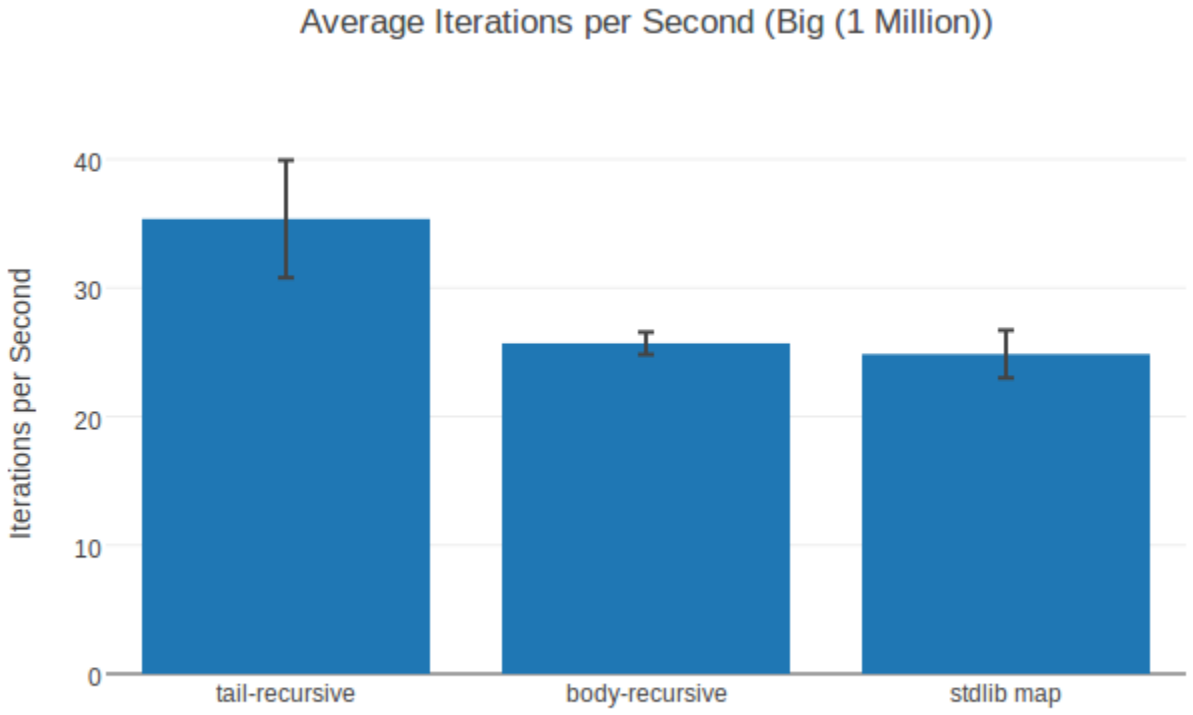
```
##### With input Small (10 Thousand) #####
Comparison:
body-recursive         5.12 K
stdlib map             5.07 K - 1.01x slower
tail-recursive         4.38 K - 1.17x slower

##### With input Middle (100 Thousand) #####
Comparison:
body-recursive         491.16
stdlib map             488.45 - 1.01x slower
tail-recursive         399.08 - 1.23x slower

##### With input Big (1 Million) #####
Comparison:
tail-recursive          35.36
body-recursive          25.69 - 1.38x slower
stdlib map              24.85 - 1.42x slower

##### With input Bigger (5 Million) #####
Comparison:
tail-recursive           6.93
body-recursive           4.92 - 1.41x slower
stdlib map               4.87 - 1.42x slower
```

TCO

```
##### With input Small (10 Thousand) #####
Comparison:
body-recursive          5.12 K
stdlib map              5.07 K - 1.01x slower
tail-recursive          4.38 K - 1.17x slower

##### With input Middle (100 Thousand) #####
Comparison:
body-recursive          491.16
stdlib map              488.45 - 1.01x slower
tail-recursive          399.08 - 1.23x slower

##### With input Big (1 Million) #####
Comparison:
tail-recursive           35.36
body-recursive           25.69 - 1.38x slower
stdlib map               24.85 - 1.42x slower

##### With input Bigger (5 Million) #####
Comparison:
tail-recursive            6.93
body-recursive            4.92 - 1.41x slower
stdlib map                4.87 - 1.42x slower
```

TCO


Average Iterations per Second (Small (10 Thousand))
Iterations per Second
6000
5000
4000
3000
2000
1000
0
stdlib map
body-recursive
tail-recursive


Average Iterations per Second (Big (1 Million))
Iterations per Second
40
30
20
10
0
tail-recursive
body-recursive
stdlib map

What even is a benchmark?

A transformation of inputs

```
config
|> Benchee.init
|> Benchee.system
|> Benchee.benchmark("job", fn -> magic end)
|> Benchee.measure
|> Benchee.statistics
|> Benchee.Formatters.Console.output
|> Benchee.Formatters.HTML.output
```

Enjoy Benchmarking! ❤

Tobias Pfeiffer
@PragTob
pragtob.info
github.com/evanphx/benchmark-ips
github.com/PragTob/benchee
benchmarkjs.com/

LIEFERY