How (NOT) To Benchmark

0. "I don't need to benchmark, my algorithm is O(n)."

1. Use unrepresentative benchmarks.
   (Blackburn et al. 2006)

2. Ignore basic statistics.
   (Georges et al. 2007)

3. Measure the wrong thing.
   (Mytkowicz et al. 2009)

4. Make faulty assumptions.
   (Barrett et al. 2017)

0. Evaluating performance.

defn. A function $f(n)$ is $O(g(n))$ if there exists constants c, n_0 such that for all $n \geq n_0$ then $f(n) \leq c \cdot g(n)$.

- From Week 5 of the second CS course. A full 15 weeks of this in "Baby algorithms."

- Sufficient? No way!
  Necessary? Debatable...

- Architecture independence is seen as an advantage, but it can also lead to practical irrelevance.

- Constant factors matter.
  Architecture matters.

e.g. Persistent vectors.

2-3 finger Trees.

O(1)	O(log n)	O(n)
append	insert	concat
prepend	nth	

Bitmapped Vector Trie.

O(1)	O(log n)	O(n)
	append	concat
	nth	insert
		prepend

- In practice Bitmapped vector trie is faster. Why?
  - $O(\log_{32} n)$ effectively constant
  - Locality of Reference

e.g. GTA Online

- Recently (Feb. 28, 2021) someone looked into why GTA Online is so slow.

- ≥ 81% of players had a loading time of ≥ 3 minutes (for a 7 year old game on today's hardware)

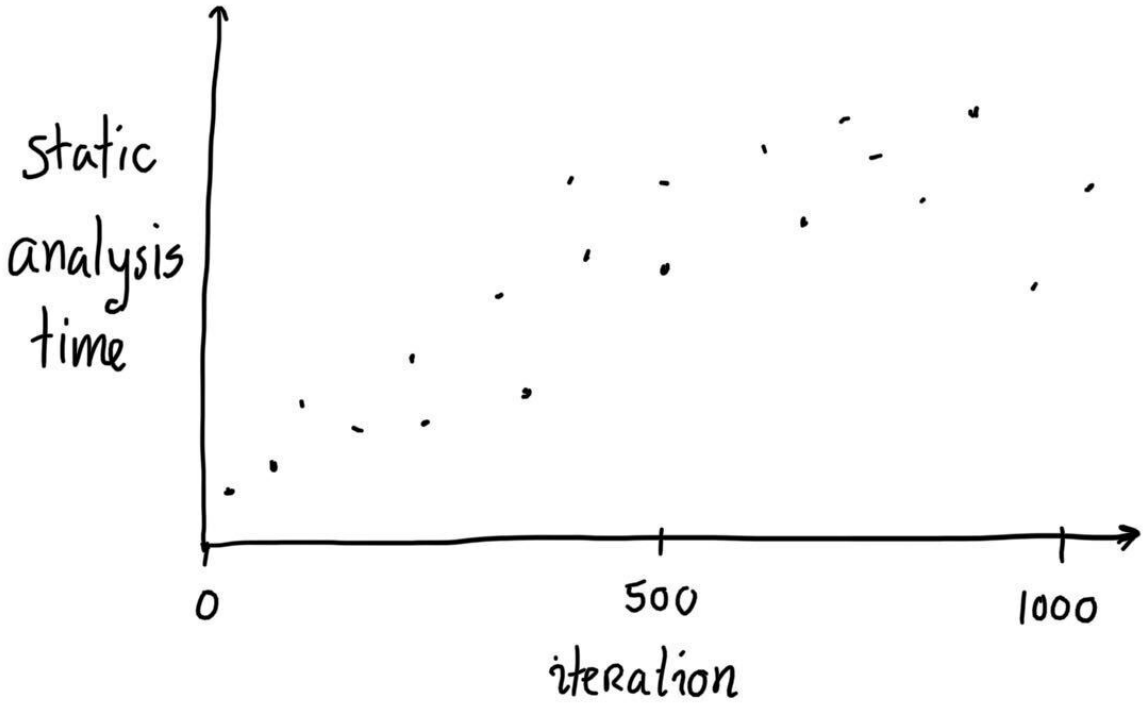
- PROBLEM: Using sscanf to read tokens when parsing JSON (it calls strlen).
  Result: 6min load ⇝ 2 min.

- Any developer who knows how to use a profiler could have found this in an afternoon.

- They didn't. I'm not surprised.

e.g. Long-running experiments.

- I'm not surprised because I've made this mistake too.
- Had experiment which took >1 month to run



- Memory leak. Analysis retained information (and worked with it) on all subsequent iterations
- ~4 line fix ⇒ finishes in 2 weeks
- "anchoring effects"

- Language and library implementers also need to care about performance

Step 1: Invent new _____.
(data structure, algorithm, compiler opt., runtime opt, etc.)

Step 2: Create or acquire some benchmark programs.

Step 3: Run them with and without improvement and measure _____.
(execution time, memory use, etc.)

Step 4: new < old? Declare victory! Write a PLDI paper.

- What could possibly go wrong?

1. Bad benchmarks.

- In 2003, the DaCapo research group was unsatisfied with existing Java benchmarks

- Standard suites: SPECjvm98, SPECjbb2000
  - Don't adequately make use of Java's features
  - Low memory footprint and overprovisioned heap (doesn't stress GC)

- New goals:
  - Diverse and relevant workloads.
  - Easy to measure and use for research.

- Developed a new suite meeting goals

1. Benchmark selection

- Eleven reasonably large real-world actively maintained Java projects (e.g. antlr, eclipse)
- Evaluation on many qualitative and quantitative metrics
    - CK metrics for OO code complexity (methods per class, inheritance hierarchy depth)
    - Heap metrics (maximum objects allocated, nursery survival rate)
    - Architecture metrics (mix of different instructions)
- Use principle component analysis (PCA) to assess diversity.

1. Benchmark packaging.

- Benchmarks should be easy to use and come with infrastructure
- Fixed, manageable workloads (feasible to run many iterations and setups)
- Harness to automatically launch, time, and test correctness
- Configuration options for input size, iterations, warmup, hardware tuning
- Remove need for multiple hosts and third-party components (e.g. databases)
- This is important but <u>thankless</u> work.
- "The [NSF] panel suggested we solve our own problems ... but didn't provide additional funds."

e.g. GTP LNM.

untyped lnm-plot.rkt.

```
(plot-pict
  (function
    (λ (N)
      ... )))
```

typed lnm-plot.rkt.

```
(plot-pict
  (function
    (λ ([N-Raw : Real])
      (: N Nonnegative-Real)
      (define N
        (if (≥ N 0) N ...))
      ... )))
```

this is never run!

typo (N-Raw)

- Got lucky, didn't impact performance.

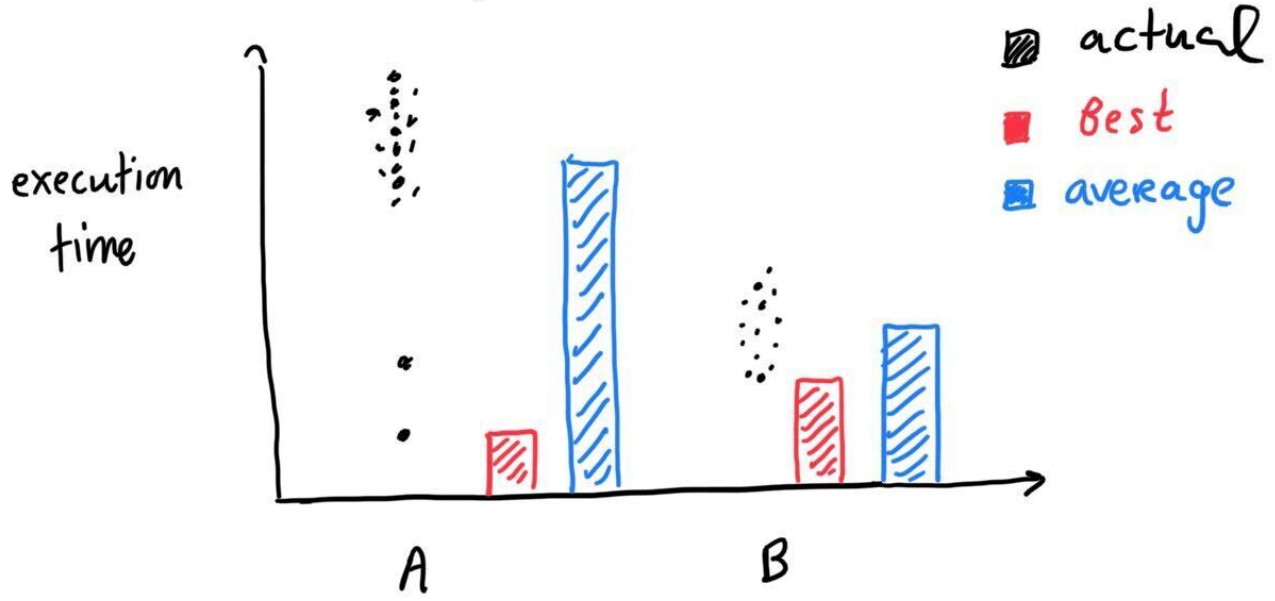
1. JavaScript too.

- Familiar story from Richards et al. in 2011, five years after DaCapo
- SunSpider suite is small and V8's benchmarks are too. They're unrepresentative of real apps.
- Doesn't make use of dynamism found in wild. Totally different memory usage patterns. (Familiar?)
- Developed new benchmarks and show Firefox's 13x improvement on SunSpider was really 3x.
- Summary: Poor benchmarks can yield wasted effort chasing fruitless optimizations
- Where did new benchmarks come from?

1. Generating Benchmarks.

- A realistic corpus is difficult for JavaScript
  - Ecosystem rapidly evolving
  - Applications are interactive
- Wrote a tool for creating a benchmark from any application via "record-and-replay"
  - Deterministic replay
  - Browser agnostic
  - Accurately mimic behavior and performance characteristics of the original
- Instrument JavaScript and output trace. Reconstruct program and memoize non-deterministic events.

- Georges et al. (2007) argue for statistically rigorous data analysis
- What do you report? Best? Second Best? Average? Worst? (SPEC recommends)



- Lots of sources of non-determinism (JIT, thread scheduling, GC, OS, ...). Need to account for random variations
- Solution: Use statistics. 95% CI of mean. T-test. ANOVA. etc... Many standard techniques.

2. How Big is the problem?

- Q: Is runtime execution variability really that Big?
- A: Startup: 8%, "Steady-State": 20% difference between max and min
- Q: Do nonRigorous data analyses lead to incorrect conclusions?
- A: Startup: 16%, "Steady-State": 20% prevalent data analysis methodology comes to "misleading" conclusion
- Summary: Yes it's a problem, a statistical Rigor is the solution. Only one issue...
- STATISTICS IS HARD!

2. Easy you say? Ha.

From Georges et al.

"If they do overlap then we cannot conclude that the differences seen in the mean values are not due to random fluctuations in the measurements."

- 95% CI can overlap by as much as 30% and still have a statistically significant difference
- See "A brief note on overlapping confidence intervals." Austin et al. 2002

2. Critiquing the critique.

- Those numbers I just presented could be totally wrong (it's unclear from their description.)

> "To consult a statistician after an experiment is finished is often merely to ask him to conduct a post mortem examination."
>
> — R.A. Fisher

- Shouldn't this be someone's job?
- Takeaway:

  If you are performing a nontrivial empirical evaluation, ask someone with statistical experience for help.

3. Measuring the wrong thing.

- A basic experiment might compare the execution of a system without optimization (S) and with it (S').
- Each execution runs in some experimental context: OS, processor, compiler flags, ...
- Q: How sensitive are measurements to experimental setup?
- A: Often, very sensitive.
- Consequently, a seemingly innocuous change in experimental setup can yield a different result
- These are confounding variables and can cause omitted-variable bias. Systematic error, not random. Statistics can't help you.

3. Subtle confounders.

- Mytkowicz (2009) demonstrates two subtle confounders
  1. Unix environment size
     - e.g. username, working dir, PATH, etc.
  2. Linking order
     - e.g. ld *.o (alphabetic)
       v.s. ld c.o b.o a.b ...
- Q: How effective is O3 optimization varying these two factors?
- A: Env size: 0.91 – 1.07
  Link order: 0.92 – 1.10
- Consequently, conclusion about whether O3 optimizes can flip depending on either of these.

3. Solutions

- Obvious solution: Don't change anything between S and S'
- Can be easier said than done if you're unaware of possible confounders.
- Many possible confounders even beyond experimental setup
- Benchmark diversity can help cancel some confounding effects
- SPEC CPU 2006 suite was NOT diverse enough to mitigate effects due to experimental setup
- Result is you may end up measuring the <u>wrong thing</u>.

e.g. GTP Zordoz.

untyped zo-find.rkt.

```
(define-values (title children)
  (parse-zo z))
...
```

typed zo-find.rkt.

```
(define-values (title children)
  (parse-zo z))
(define zstr
  (format "%a" z))
...
```

unused with no equivalent in untyped

↖ from untyped code

- Were NOT lucky. Worst case overhead goes from 14x to <4x slowdown.
- One line change can cause an order of magnitude difference

e.g. Branch ordering.

slow.rkt

```
(match
  [(R (B) (= v) (B)) ...]
  [(B (B) (= v) (B)) ...]
  ...
  [(N a x b)
   (cond
     [(< x v) ...]
     [(= x v) ...]
     [(> x v) ...])])
```

base cases

hot branches

better.rkt

```
(match
  [(N a x b)
   (cond
     [(< x v) ...]
     [(= x v)  — Base cases —  ]
     [(> x v) ...])])
```

- a 2x slowdown disappears!

3. Is that good enough?

- Perfectly controlled confounders.
- Is that enough? Nope!
- Internal validity: Does my experiment establish a causal link?
- External validity: Does my conclusion generalize to a broad population?
- Controlling confounders improves internal validity ONLY.
- Additionally, randomize experimental setup. Can improve both.
- Also easier said than done. Requires more time and resources. Produces more data to analyze.

4. Making false assumptions.

"Later iterations usually have less compilation... and may reach a steady state."

— DaCapo team in CACM

"Since most of the JIT compilation is performed during startup, steady state performance suffers from less variability."

— Georges et al. (2007)

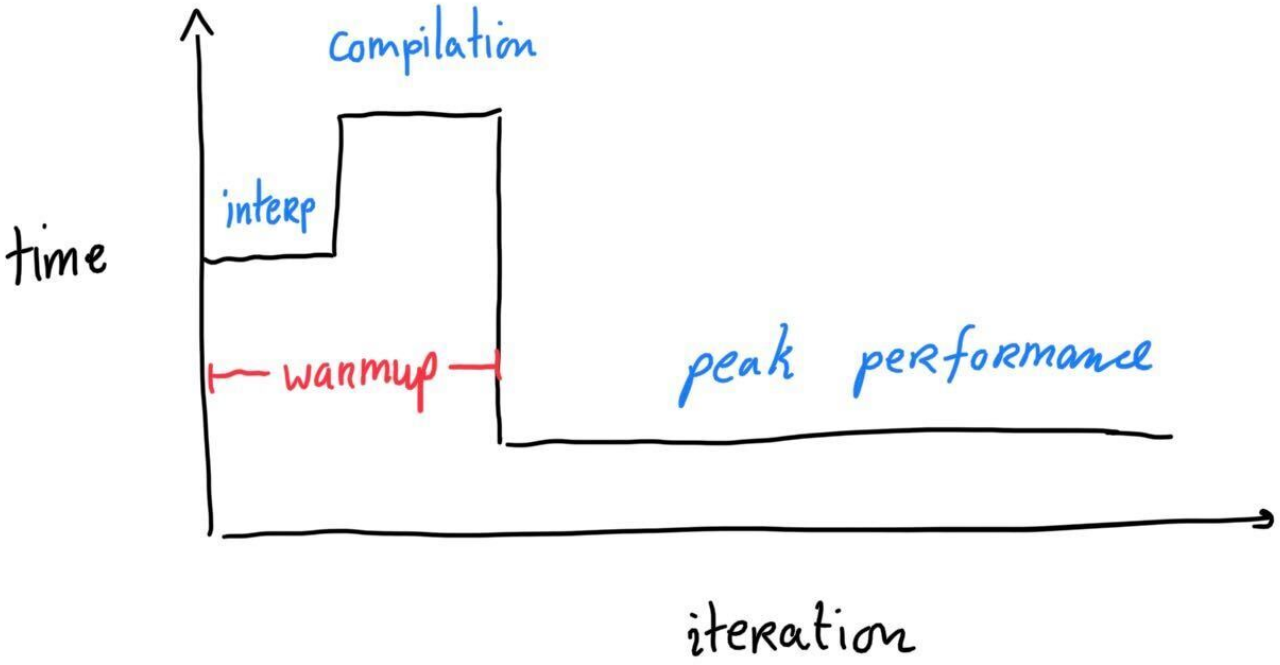
- Ad hoc approach: Drop the first 2 or 3 benchmark iterations.

- More rigorous: Fix desired iterations to keep k. Once coefficient of variation of last k iterations is below a threshold α, then take those.

↑ still somewhat ad hoc

4. How long is warmup?

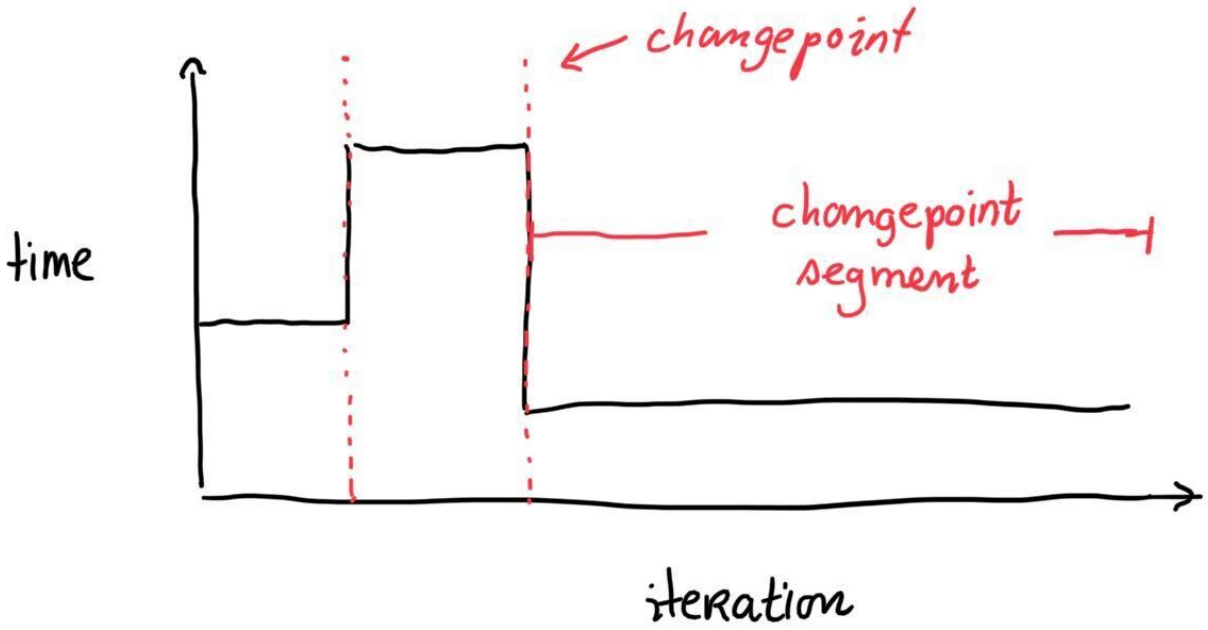
- Expected performance given knowledge of JIT compilation:



- Q: How long is the warmup period?

- Methodology: Take small benchmarks (more predictable), remove as many confounders as possible, several VMs, and measure performance across 2000 iterations

4. Data analysis.



Automatic performance classifier

- Segments equivalent: FLAT
- Final fastest: WARMUP

} good

- Final not fastest: SLOWDOWN
- Never settles: NO STEADY STATE

} bad

Good corresponds to classical expectations of JIT performance.

Bad corresponds to abnormal performance characteristics.

4. Results

- For each VM - Benchmark pair only 40% consistently gave expected performance characteristics when repeating experiment

  4.5% consistent slowdown
  4.5% consistent no steady state
  51% mix containing either slowdown or no steady state

- Takeaways:
  - Your methodolgy cannot assume peak performance or even a steady state will occur
  - Even in carefully controlled experiments, performance can be unpredictable

- Why is performance evaluation so difficult?
- Many answers to this. A few:
  - Satisfies 2/3 criteria for catastrophic failure potential (Perrow 1984): complexity, tight coupling.
  - Mistakes are silent, no typechecker for performance
  - Little academic incentive for going above and beyond in an evaluation
  - Too much folklore, not enough fact: "GC is bloated and slow" (Zorn 1992)
  - Proper evaluation skills are not taught.

5. Broader context.

- This isn't a problem that only affects systems or PL
- Same thing in other fields with a similar timeline
- Replication crisis: Starting with "Why Most Published Research Findings are Wrong" (2005)
  - Standard methodological practices have lead to incorrect results becoming commonplace
  - p hacking, publication bias, underpowered experiments
- Especially as PL adopts more empirical methods, this will be more prevalent

5. A note of optimism.

- Not too long ago papers often didn't come with associated data or software
- Or maybe the link to it died. Or it wasn't portable. Or it just didn't work.
- Now, peer-reviewed artifacts available at permanent storage links is normal
- Other disciplines are beginning to make progress too, e.g. preregistration of clinical trial to prevent p-hacking
- As we've seen, more work to be done though

5. Resources

Tools

Krun and ReBench for eliminating confounders due to operating system and hardware parameters.

Articles

- R3 – Repeatability, Reproducibility, Rigor (Vitek, and Kalibera 2011)
- SIGPLAN Empirical Evaluation Checklist

Books

- Experimental Methods for Science and Engineering Students (Kirkup)
- Experimental Design and Analysis (Seltman, free online!)
- Statistics Done Wrong (Reinhart)

Conclusion.

DON'T Use small contrived benchmark programs

DO Use large real-world programs solving a diverse set of problems

DON'T Just report best or worst performance

DO Use statistical techniques as appropriate

DON'T Introduce experimental confounders

DO Eliminate as many as possible

DON'T Assume classical warmup always happens

DO Visualize long-running benchmarks to double-check your assumptions

Most importantly DO evaluate performance. We DON'T need more folklore!