TORSTEN HOEFLER

Scientific Benchmarking of Parallel Computing Systems

Twelve ways to tell the masses when reporting performance results

BenchCouncil Rising Star Award Lecture
Keynote at 2020 BenchCouncil International Symposium on Benchmarking, Measuring and Optimizing (Bench'20)

OPINION

PNAS, Feb. 2015

Opinion: Reprodu... wr... ...ng

Jeffrey T...
[a]Associate ... ...ology
Johns Hop...

“In the good old days physicists repeated each other’s experiments, just to be sure. Today they stick to FORTRAN, so that they can share each other’s programs, bugs included.” – Edsger Dijkstra (1930-2002), Dutch computer scientist, Turing Award 1972

Reproduci... results—ar... experime... result—ar... of success... findings are the primary means by which scientific evidence accumulates for or against a hypothesis. Yet, of late, there has been a crisis of confidence among researchers worried about the rate at which studies are either

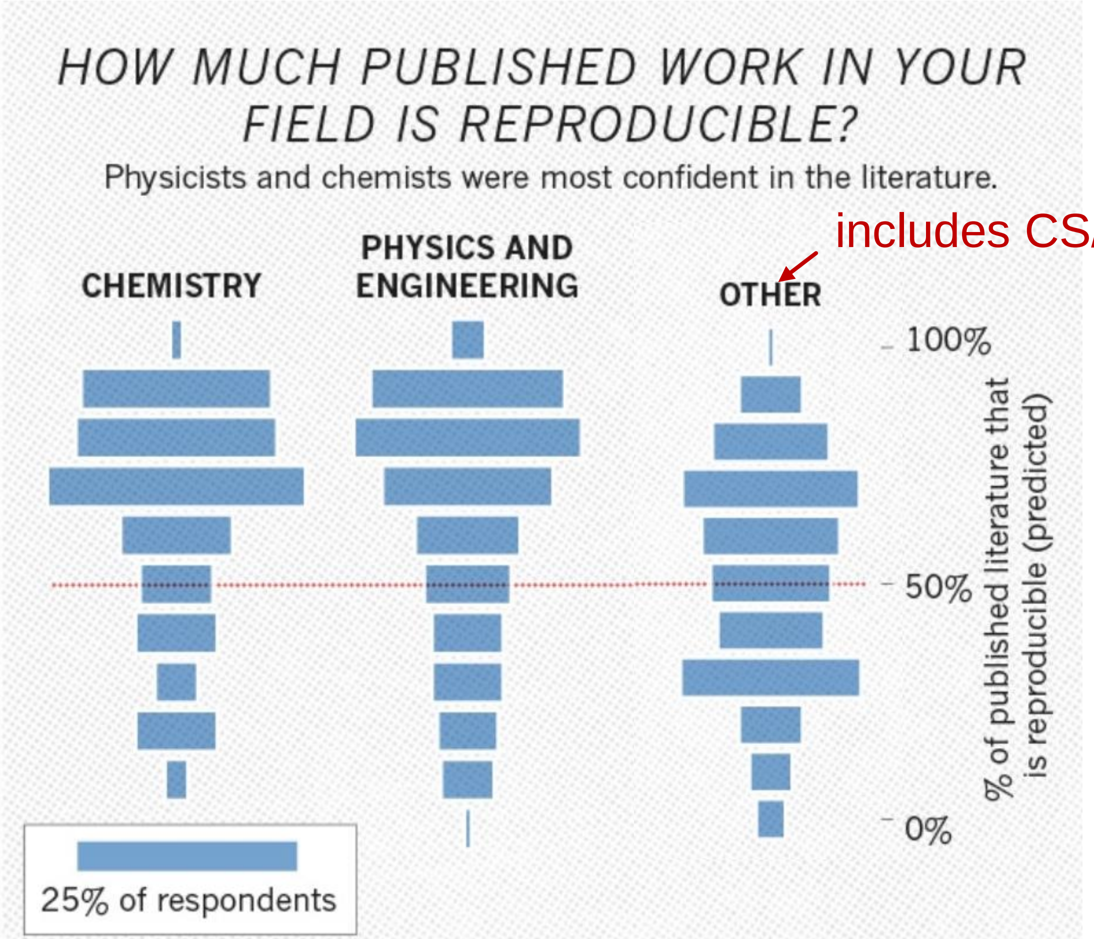
been some very public failings of reproducibility across a range of disciplines from cancer genomics (3) to economics (4), and the data for many publications have not been made publicly available, raising doubts about the quality of data analyses. Popular press articles have raised questions about the reproducibility of all scientific research (5), and the US Congress has convened hearings focused on the transparency of scientific research (6). The result is that much of the

Unfortunately, the mere reproducibility of computational results is insufficient to address the replication crisis because even a reproducible analysis can suffer from many problems—confounding from omitted variables, poor study design, missing data—that threaten the validity and useful interpretation of the results. Although improving the reproducibility of research may increase the rate at which flawed analyses are uncovered, as recent high-profile examples have demonstrated (4), it does not change the fact that

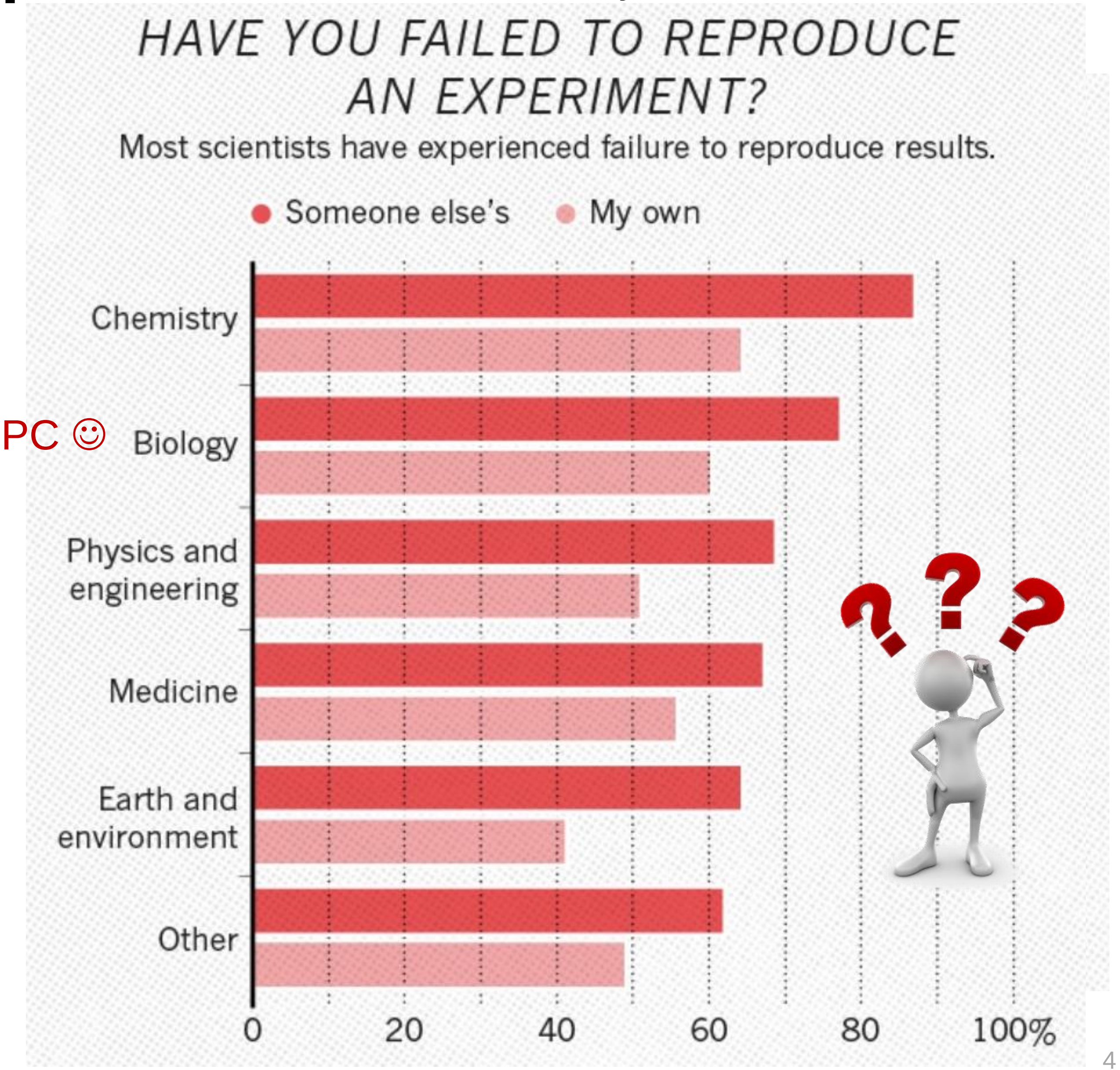
Reproducibility and replicability?

Nature, May 2016

- **Reproducibility – get the exact results**
- **Replicability – repeat the effect/insight**



includes CS/HPC ☺



Functional reproducibility is relatively simple – release the code!

Single-threaded, if you don't care much about performance

Gets a bit more complex when you share parallel codes (IEEE 754 is not associative)

IPDPS'14

Designing Bit-Reproducible Portable High-Performance Applications*

Andrea Arteaga
ETH Zurich, Switzerland
andrea.arteaga@env.ethz.ch

Oliver Fuhrer
Federal Office for Meteorology and Climatology
MeteoSwiss, Zurich, Switzerland
oliver.fuhrer@meteoswiss.ch

Torsten Hoefler
ETH Zurich, Switzerland
htor@ethz.ch

Abstract—**Bit-reproducibility has many advantages in the context of high-performance computing. Besides simplifying and making more accurate the process of debugging and testing the code, it can allow the deployment of applications on heterogeneous systems, maintaining the consistency of the computations. In this work we analyze the basic operations performed by scientific applications and identify the possible sources of non-reproducibility. In particular, we consider the tasks of evaluating transcendental functions and performing reductions using non-associative operators. We present a set of techniques to achieve reproducibility and we propose im-**

runs is often of key importance in order to locate and isolate bugs. Especially, when refactoring an application in a way that the results should not change, reproducibility can significantly ease testing. However, debugging is only a secondary use-case for us. Many applications being run on large, parallel high performance computing facilities simulate the behavior of complex and highly non-linear systems. Prominent examples can be found in molecular dynamics or weather and climate simulation. For example, for weather

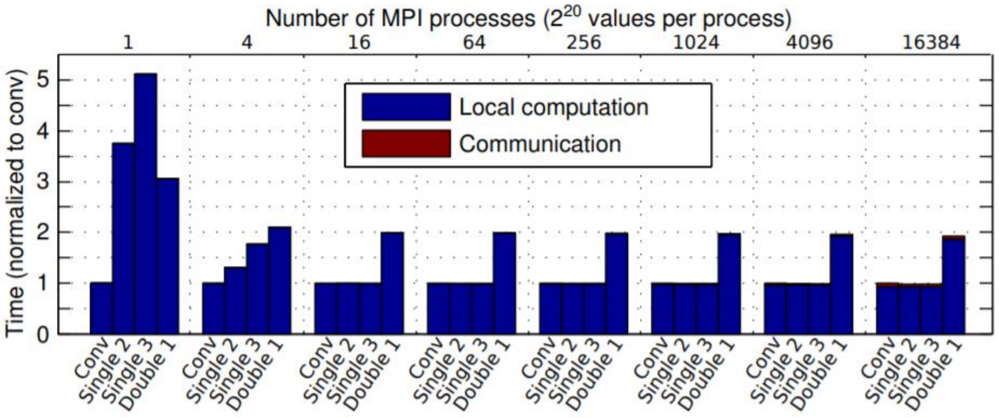


Figure 8. Performance comparison of conventional reduction performed with MKL (*Conv*), single-sweep reduction with two levels (*Single2*), with three levels (*Single3*) and double-sweep reduction with 1 level (*Double 1*) on varying number of processes, each owning 2^{20} double-precision values,

But what if performance is your science result?

- **Original findings:**
  - If carefully tuned, NBC speed up a 3D solver
    Full code published
  - 800^3 domain – 4 GB (distributed) array

Reproducing performance results is hard! Is it even possible?

- **9 years later: attempt to reproduce ☺!**
  - *System A: 28 quad-core nodes, Xeon E5520*
  - *System B: 4 nodes, dual Opteron 6274*

"Neither the experiment in A nor the one in B could reproduce the results presented in the original paper, where the usage of the NBC library resulted in a performance gain for practically all node counts, reaching a superlinear speedup for 96 cores (explained as being due to cache effects in the inner part of the matrix vector product)."

My own replication result

Characterizing the Influence of System Noise on Large-Scale Applications by Simulation

Torsten Hoefler
University of Illinois at Urbana-Champaign
Urbana IL 61801, USA
htor@illinois.edu

Timo Schneider and Andrew Lumsdaine
Indiana University
Bloomington IN 47405, USA
{timoschn,lums}@cs.indiana.edu

structure of the noise. Simulations with different network speeds show that a 10x faster network does not improve application scalability. We quantify noise and conclude that our tools can be utilized to tune the noise signatures of a specific system.

I. Motivation and Background

The performance impact of operating system and architectural overheads (*system noise*) at massive scale is increasingly of concern. Even small local delays on compute nodes, which can be caused by interrupts, operating system daemons, or even cache or page misses, can affect global application performance significantly [1]. Such local delays often cause less than 1% overhead per process but severe performance losses can occur if noise is propagated (*amplified*) through communication or global synchronization. Previous analyses generally assume that the performance impact of system noise grows at scale and Tsafrir et al. [2] even suggest that the

is supported directly by the BG/L hardware, allreduce used a pattern similar to the dissemination pattern. We use LogGP parameters from BlueGene/P running CNL because we do not have access to a BlueGene/L. Thus, we expect the impact to be slightly lower, but asymptotically similar. Like Beckman et al., we used unsynchronized noise with a fixed frequency of 1,000, 100, and 10 Hz causing detours of 16, 50, 100, and

[4]http://www.unixer.de/LogGOPSim (2010)

"[...] a collective communication call may, or may not, have the effect of synchronizing all calling processes. This statement excludes, of course, the barrier function." This invalidates all simple models in use today. The synchronization properties of an application depend on the collective algorithm, point-to-point messaging, and the system's network parameters.

We chose a simulation approach similar to Sottile et al.'s [8] and improve it by using noise traces from existing systems combined with detailed simulation and extrapolation of collec-

Replicating performance results is possible but rare! Make it the default?

results from Ferreira, Bridges, Brightwell as well as Beckman et al. both two years earlier on different machines

HPC Performance reproducibility – is it worth trying?

- ~~**Reproducibility – get the exact results**~~
- ~~**Replicability – repeat the effect/insight**~~

Small Quiz

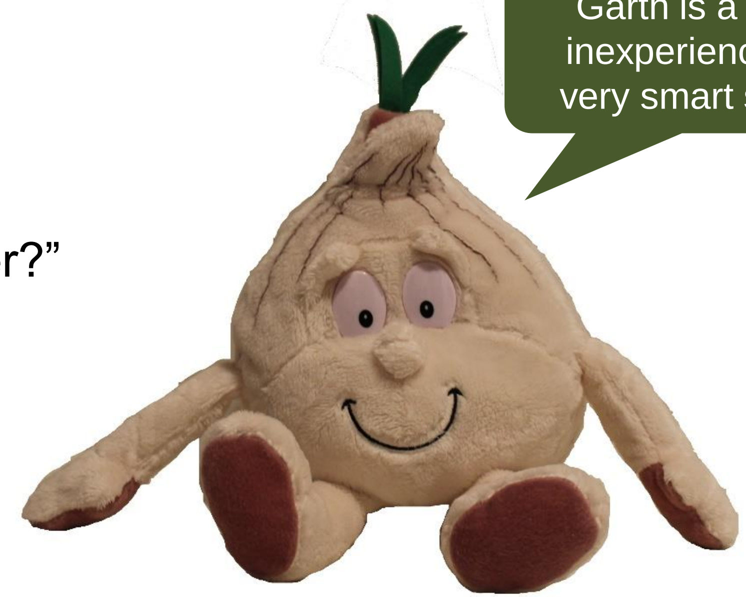
Do you believe one can reproduce any Gordon Bell finalist from before 2015?

Interpretability: *We call an experiment interpretable if it provides enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results.*

How does Garth measure and report performance?

- **We are all interested in High Performance Computing**
  - We (want to) treat it as a science – reproducing experiments is a major pillar of the scientific method

- **When measuring performance, important questions are**
  - “How many iterations do I have to run per measurement?”
  - “How many measurements should I run?”
  - “Once I have all data, how do I summarize it into a single number?”
  - “How do I compare the performance of different systems?”
  - “How do I measure time in a parallel system?”
  - …



- **How are they answered in the field today?**
  - Young scientists ask their advisors … who typically answer based on some intuition
  - We (the community) need to establish scientific principles for benchmarking

  But do we not already have them – let’s see …

State of the Practice in HPC

- **Stratified random sample of three top-conferences over four years**
  - HPDC, PPoPP, SC (years: 2011, 2012, 2013, 2014)
  - 10 random papers from each (10-50% of population)
  - 120 total papers, 20% (25) did not report performance (were excluded)

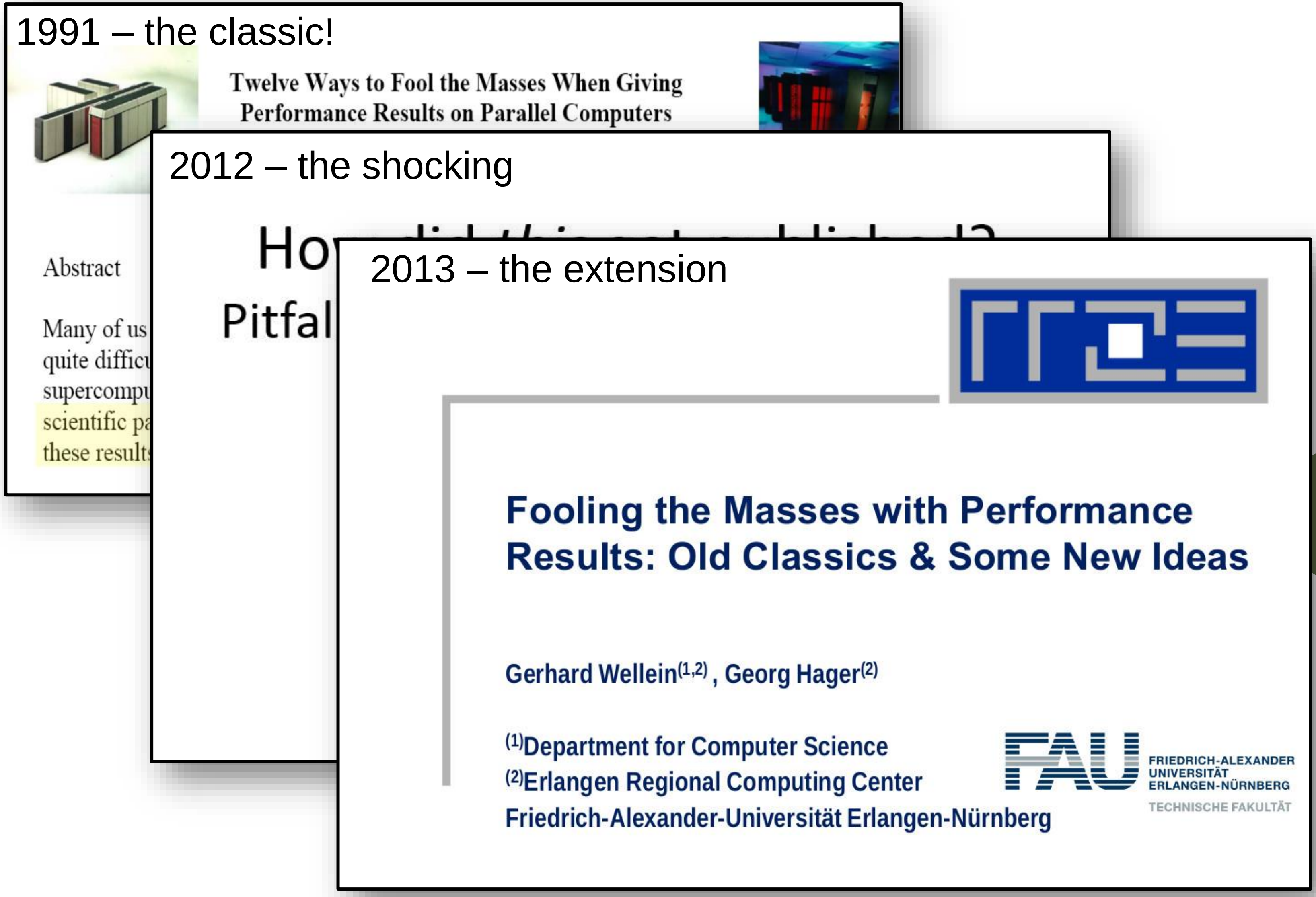
- **Main results:**
  1. Most papers report details about the hardware but fail to describe the software environment.
     Important details for reproducibility missing
  2. The average paper's results are hard to interpret and easy to question
     Measurements and data not well explained
  3. No statistically significant evidence for improvement over the years ☹

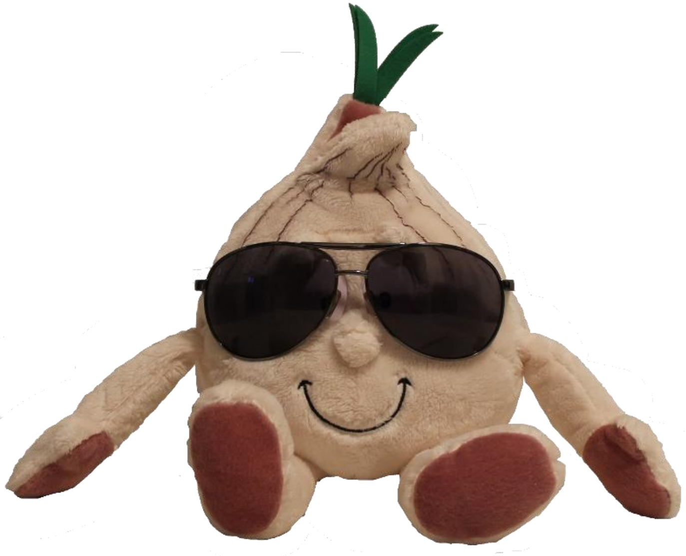
- **Our main thesis:**

Performance results are often nearly impossible to reproduce! Thus, we need to provide enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results.

This is especially important for HPC conferences and activities such as the Gordon Bell award!

Well, we all know this - but do we really know how to fix it?





Yes, this is a garlic press!

Eddie is Garth's advisor

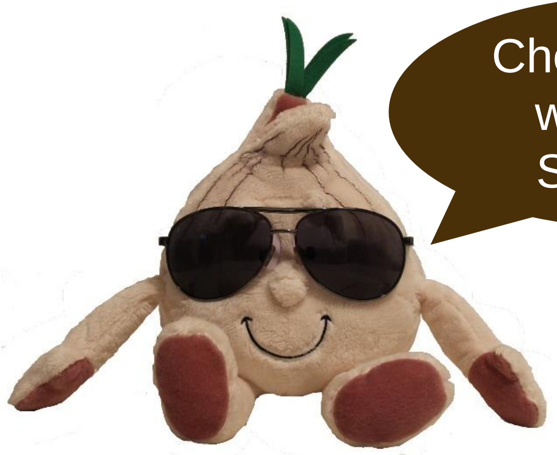
…

Our constructive approach: provide a set of (12) rules

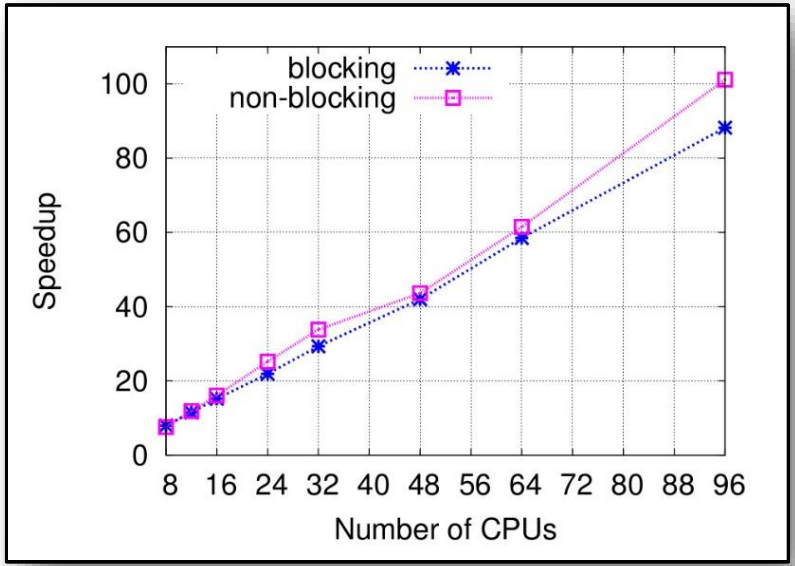
- **Attempt to emphasize interpretability of performance experiments**
- **The set is not complete**
  - And probably never will be
  - Intended to serve as a solid start
  - Call to the community to extend it

- **I will illustrate the 12 rules now**
  - Using real-world examples
    All anonymized!
  - Garth and Eddie will represent the naive/good scientist

The most common issue: speedup plots



Check out my wonderful Speedup!



I can't tell if this is useful at all!

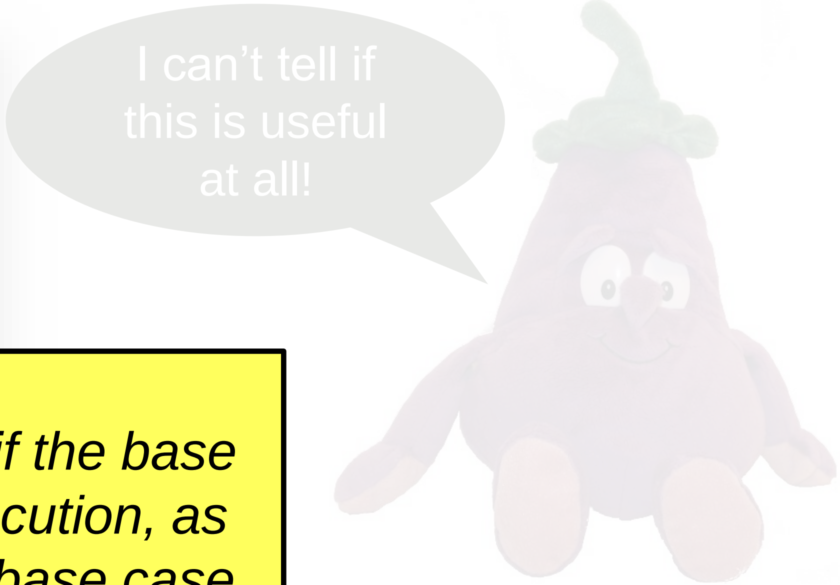
- **Most common and oldest-known issue**
  - First seen 1988 – also included in Bailey's 12 ways
  - 39 papers reported speedups

    15 (38%) did not specify the base-performance ☹
  - Recently rediscovered in the "big data" universe

    A. Rowstron et al.: Nobody ever got fired for using Hadoop on a cluster, HotCDP 2012

    F. McSherry et al.: Scalability! but at what cost?, HotOS 2015

The most common issue: speedup plots



Rule 1: *When publishing parallel speedup, report if the base case is a single parallel process or best serial execution, as well as the absolute execution performance of the base case.*

- **Most common and oldest-known issue**
  - First seen 1988 – also included in Bailey's 12 ways
  - 39 papers reported speedups

    15 (38%) did not specify the base-performance ☹
  - Recently rediscovered in the "big data" universe

    A. Rowstron et al.: Nobody ever got fired for using Hadoop on a cluster, HotCDP 2012

    F. McSherry et al.: Scalability! but at what cost?, HotOS 2015

- **A simple generalization of this rule implies that one should never report ratios without absolute values.**

Garth's new compiler optimization

ck out my compiler!

Performance in Gflop/s

How d… perform… and B…

3.5
3

Rule 2: *Specify the reason for only reporting subsets of standard benchmarks or applications or not using all system resources.*

NAS CG NAS LU NAS EP

ICC LLVM GarthCC

Well, GarthCC segfaulted for FT and was 20% slower for BT.

- **This implies: Show results even if your code/approach stops scaling!**

The mean parts of means – or how to summarize data

Rule 3: *Use the arithmetic mean only for summarizing costs. Use the harmonic mean for summarizing rates.*

Rule 4: *Avoid summarizing ratios; summarize the costs or rates that the ratios base on instead. Only if these are not available use the geometric mean for summarizing ratios.*

- **51 papers use means to summarize data, only four (!) specify which mean was used**
  - **A single paper correctly specifies the use of the harmonic mean**
  - **Two use geometric means, without reason**
  - **Similar issues in other communities (PLDI, CGO, LCTES) – see N. Amaral's report**
- **harmonic mean ≤ geometric mean ≤ arithmetic mean**

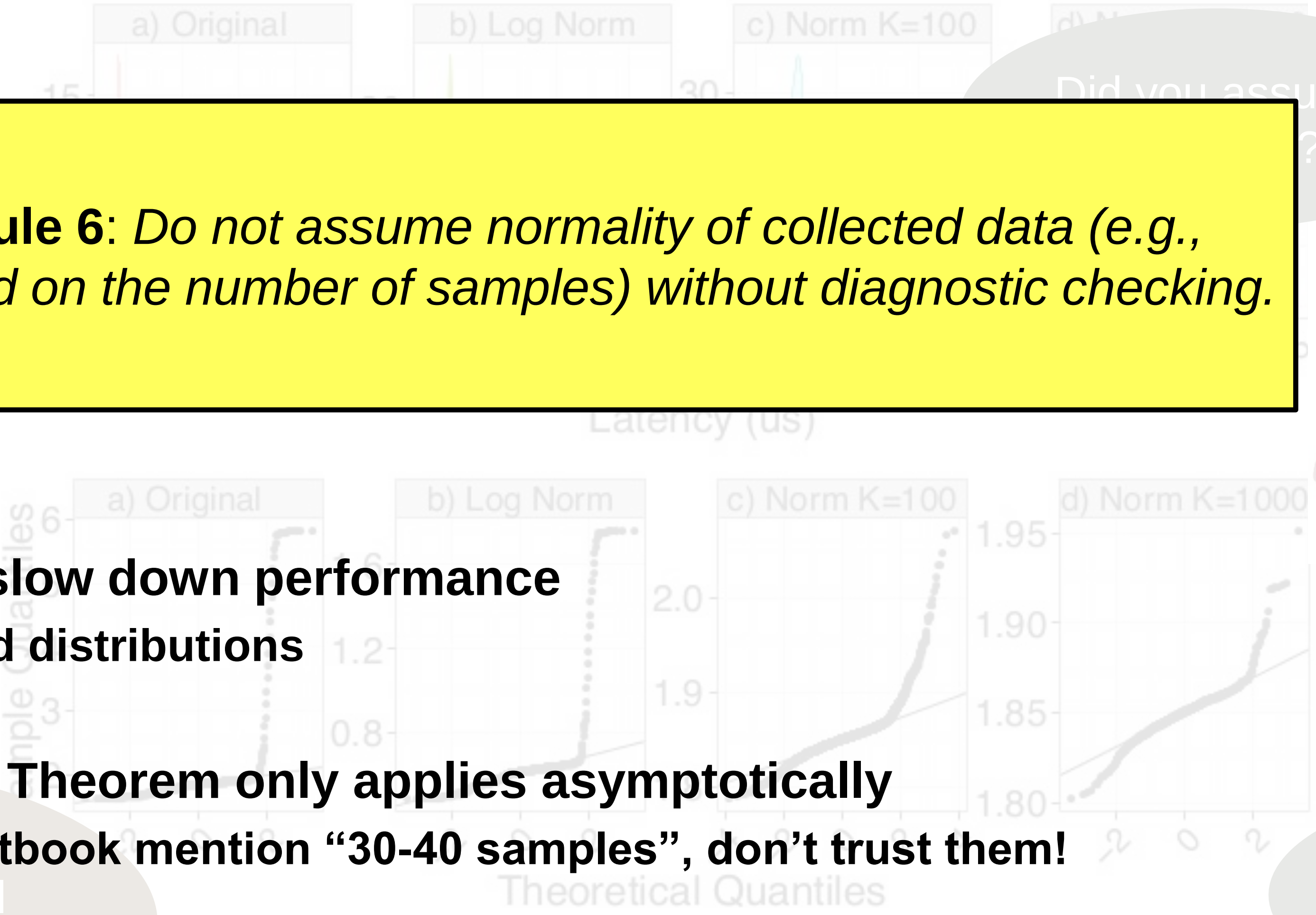
The simplest networking question: ping pong latency!

Rule 5: *Report if the measurement values are deterministic. For nondeterministic data, report confidence intervals of the measurement.*

- **Most papers report nondeterministic measurement results**
  - Only 15 mention some measure of variance
  - Only two (!) report confidence intervals
- **CIs allow us to compute the number of required measurements!**
- **Can be very simple, e.g., single sentence in evaluation:**

  “We collected measurements until the 99% confidence interval was within 5% of our reported means.”

Thou shalt not trust your average textbook!

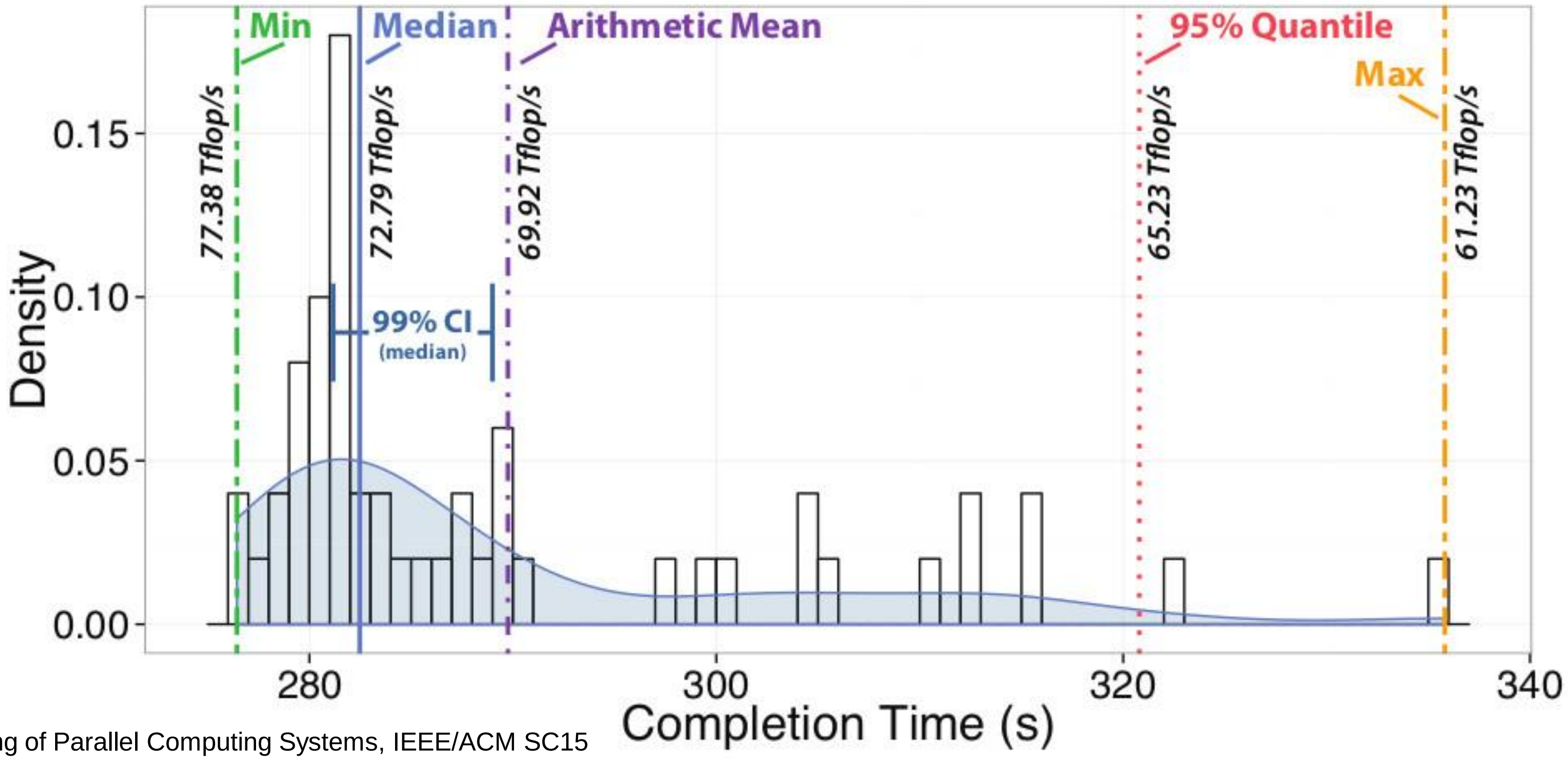


Rule 6: *Do not assume normality of collected data (e.g., based on the number of samples) without diagnostic checking.*

- **Most events will slow down performance**
  - **Heavy right-tailed distributions**
- **The Central Limit Theorem only applies asymptotically**
  - **Some papers/textbook mention “30-40 samples”, don’t trust them!**
- **Two papers used CIs around the mean without testing for normality**

Dealing with non-normal data – nonparametric statistics

- **Rank-based measures (no assumption about distribution)**
  - Essentially always better than assuming normality
- **Example: median (50th percentile) vs. mean for HPL**
  - Rather stable statistic for expectation
  - Other percentiles (usually 25th and 75th) are also useful



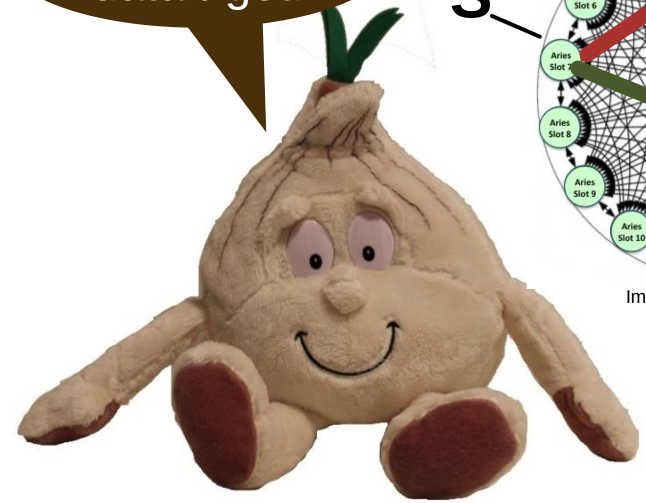
TH, Belli: Scientific Benchmarking of Parallel Computing Systems, IEEE/ACM SC15

Comparing nondeterministic measurements

Rule 7: Compare nondeterministic data in a statistically sound way, e.g., using non-overlapping confidence intervals or ANOVA.

Thou shalt not trust your system!

Look what data I got!



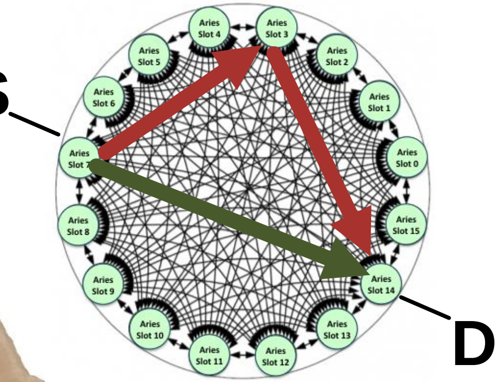
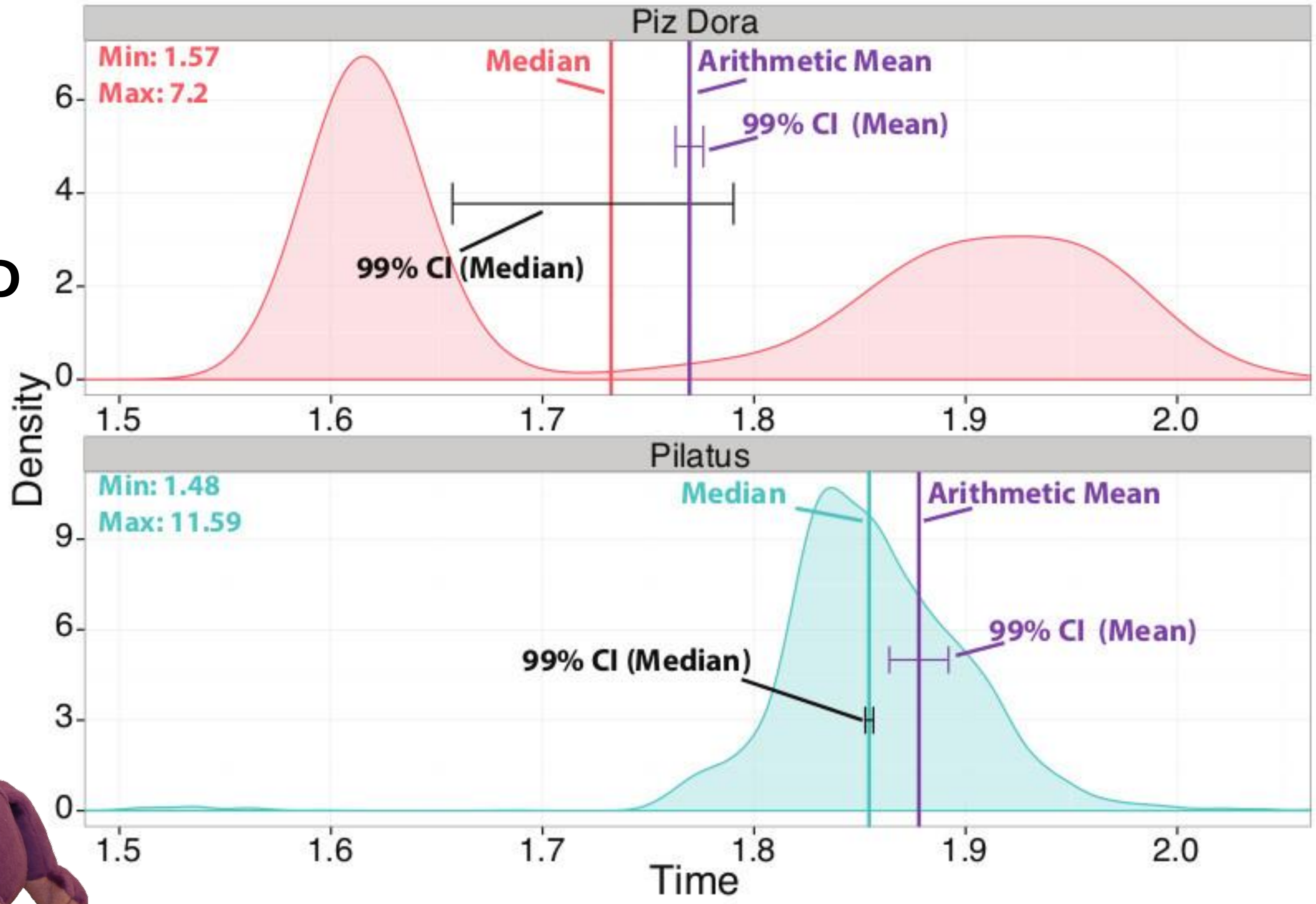
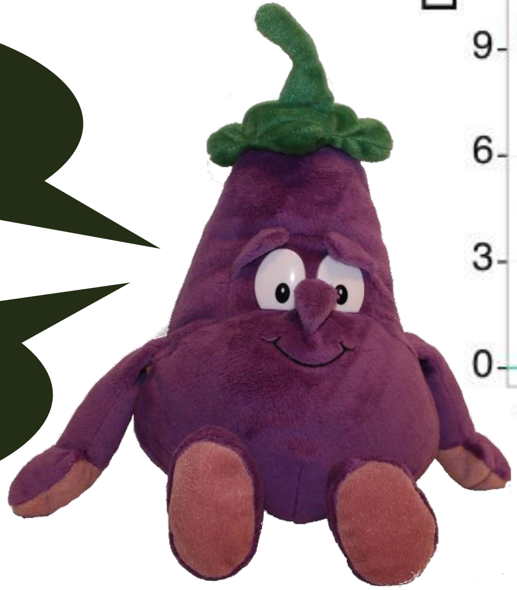


Image credit: nersc.gov

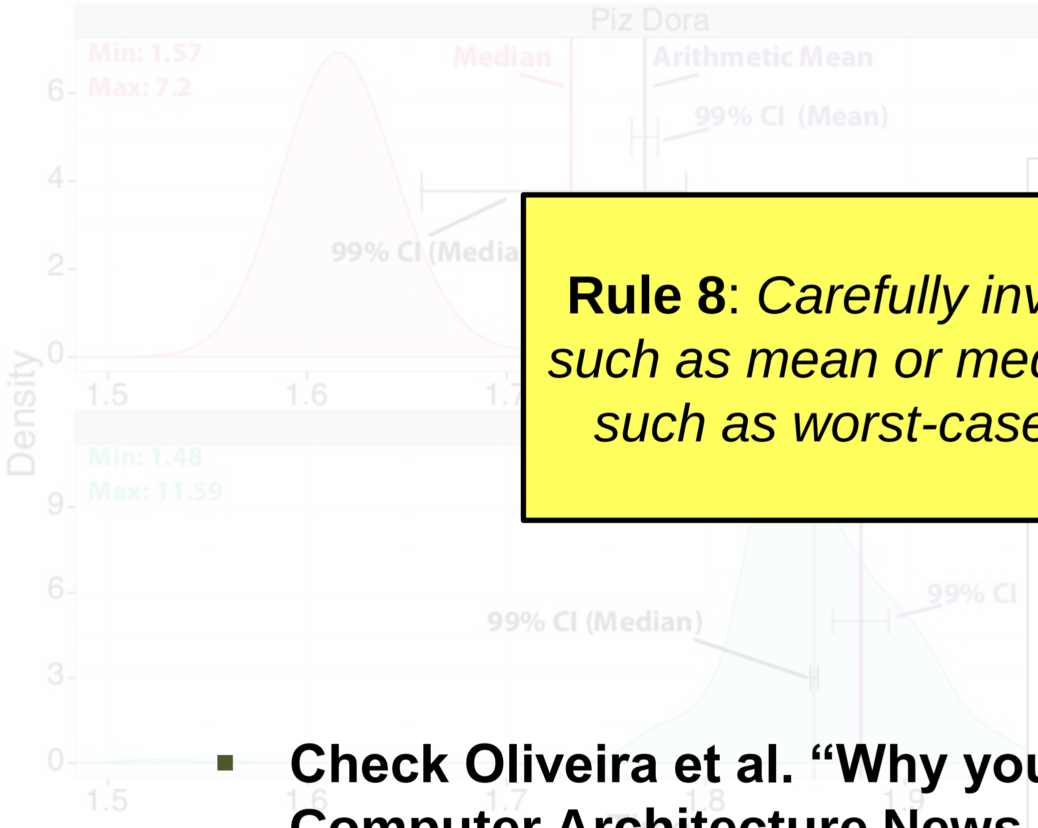


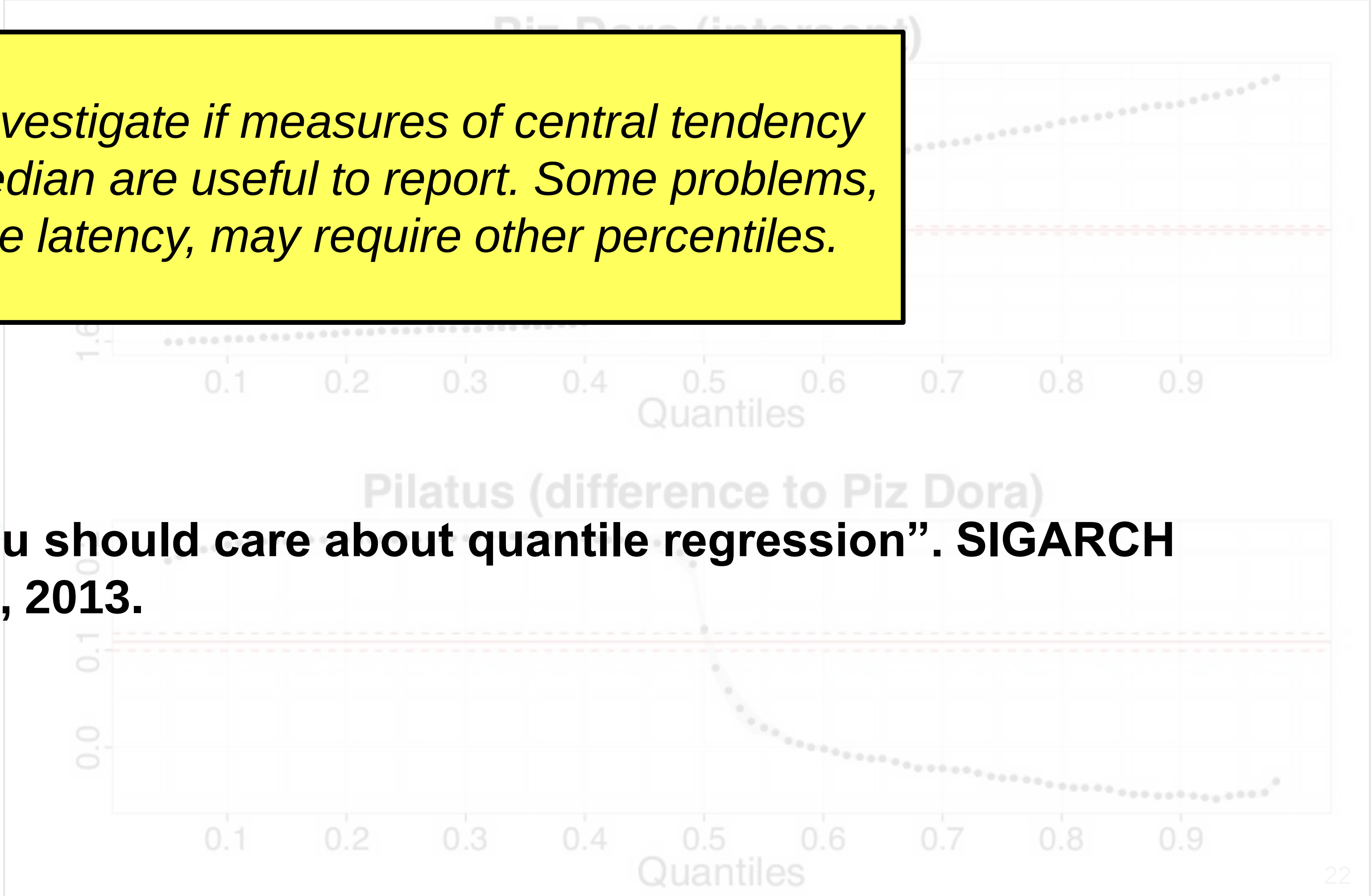
Clearly, the mean/median are not sufficient!

Try quantile regression!



Quantile Regression





Rule 8: *Carefully investigate if measures of central tendency such as mean or median are useful to report. Some problems, such as worst-case latency, may require other percentiles.*

- **Check Oliveira et al. “Why you should care about quantile regression”. SIGARCH Computer Architecture News, 2013.**

How many measurements are needed?

- **Measurements can be expensive!**
  - Yet necessary to reach certain confidence

- **How to determine the minimal number of measurements?**
  - Measure until the confidence interval has a certain acceptable width
  - For example, measure until the 95% CI is within 5% of the mean/median
  - Can be computed analytically assuming normal data
  - Compute iteratively for nonparametric statistics

- **Often heard: “we cannot afford more than a single measurement”**
  - E.g., Gordon Bell runs
  - Well, then one cannot say anything about the variance
    Even 3-4 measurement can provide very tight CI (assuming normality)
    Can also exploit repetitive nature of many applications

Experimental design

I don't believe you, try other numbers of processes!

MPI_Reduce behaves much

Rule 9: *Document all varying factors and their levels as well as the complete experimental setup (e.g., software, hardware, techniques) to facilitate reproducibility and provide interpretability.*

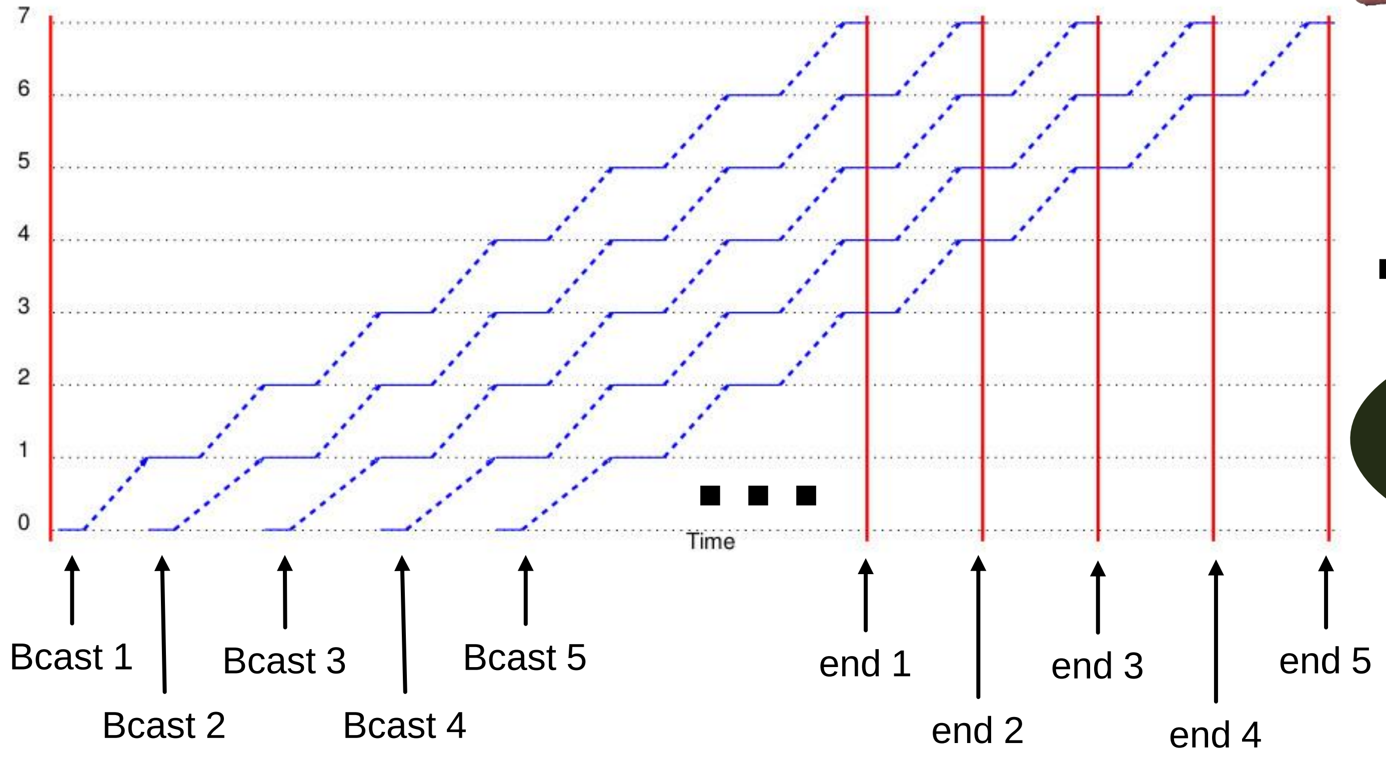
- **We recommend factorial design**
- **Consider parameters such as node allocation, process-to-node mapping, network or node contention**
  - If they cannot be controlled easily, use randomization and model them as random variable
- **This is hard in practice and not easy to capture in rules**

Time in parallel systems

My simple broadcast takes only one latency!

But I measured it so it must be true!

```
t = -MPI_Wtime();
for(i=0; i<1000; i++) {
  MPI_Bcast(…);
}
t += MPI_Wtime();
t /= 1000;
```



That's nonsense!

…

Measure each operation separately!

Summarizing times in parallel systems!

Rule 10: *For parallel time measurements, report all measurement, (optional) synchronization, and summarization techniques.*

- **Measure events separately**
  - **Use high-precision timers**
  - **Synchronize processes**

- **Summarize across processes:**
  - **Min/max (unstable), average, median – depends on use-case**

Rule 11: *If possible, show upper performance bounds to facilitate interpretability of the measured results.*

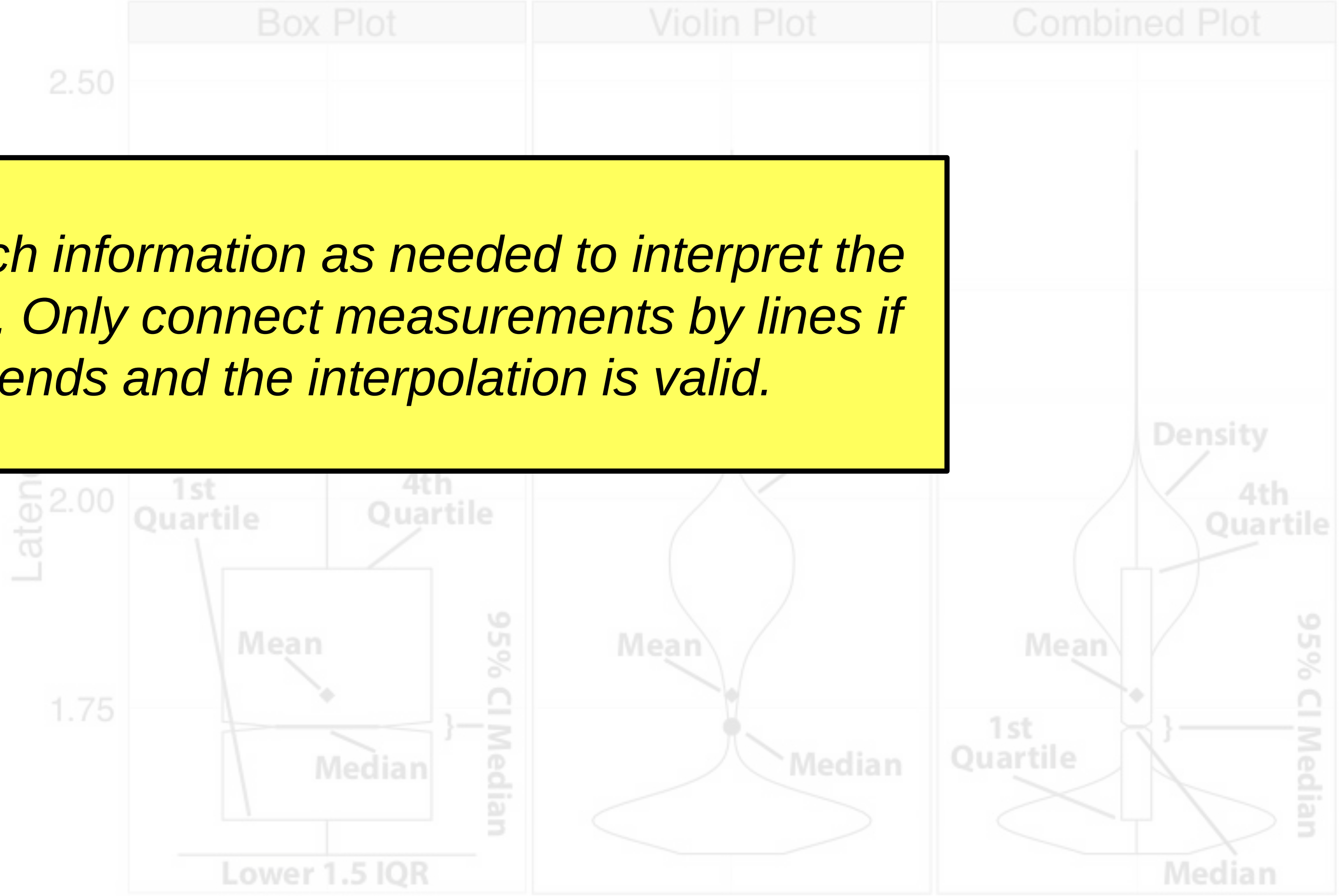
- **Model computer system as k-dimensional space**
  - Each dimension represents a capability
    Floating point, Integer, memory bandwidth, cache bandwidth, etc.
  - Features are typical rates
  - Determine maximum rate for each dimension
    E.g., from documentation or benchmarks
- **Can be used to proof optimality of implementation**
  - If the requirements of the bottleneck dimension are minimal

Plot as much information as possible!

My most common request was "show me the data"

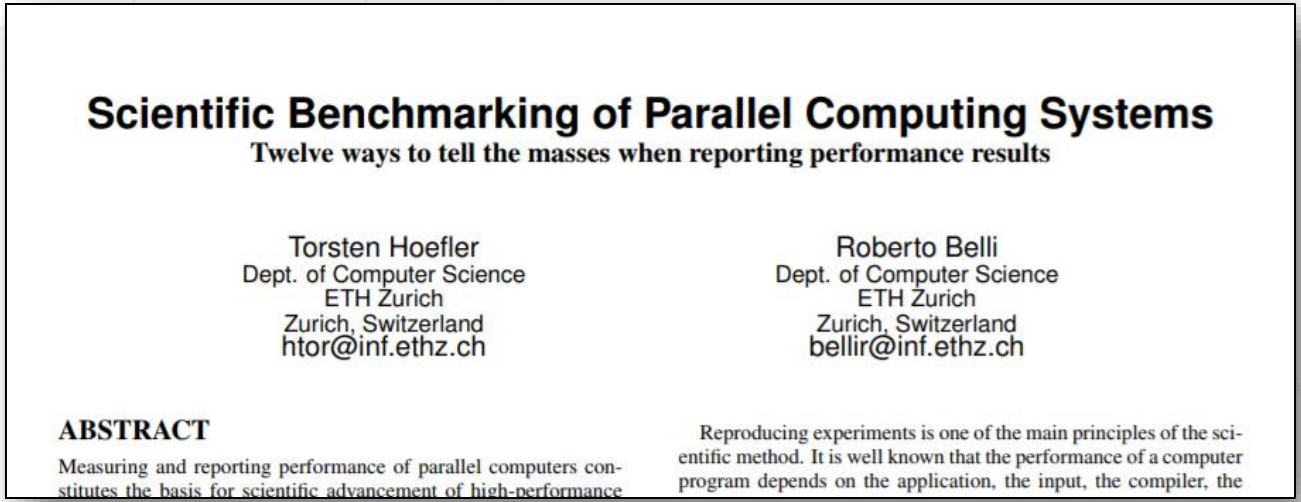
Rule 12: *Plot as much information as needed to interpret the experimental results. Only connect measurements by lines if they indicate trends and the interpolation is valid.*

This is how I should have presented the Dora results.



Wrapping up the 12 rules …

- **Attempt to emphasize interpretability of performance experiments**
  - Teach some basic statistics
- **The set of 12 rules is not complete**
  - And probably never will be
  - Intended to serve as a solid start
  - Call to the community to extend it



Scientific Benchmarking of Parallel Computing Systems
Twelve ways to tell the masses when reporting performance results

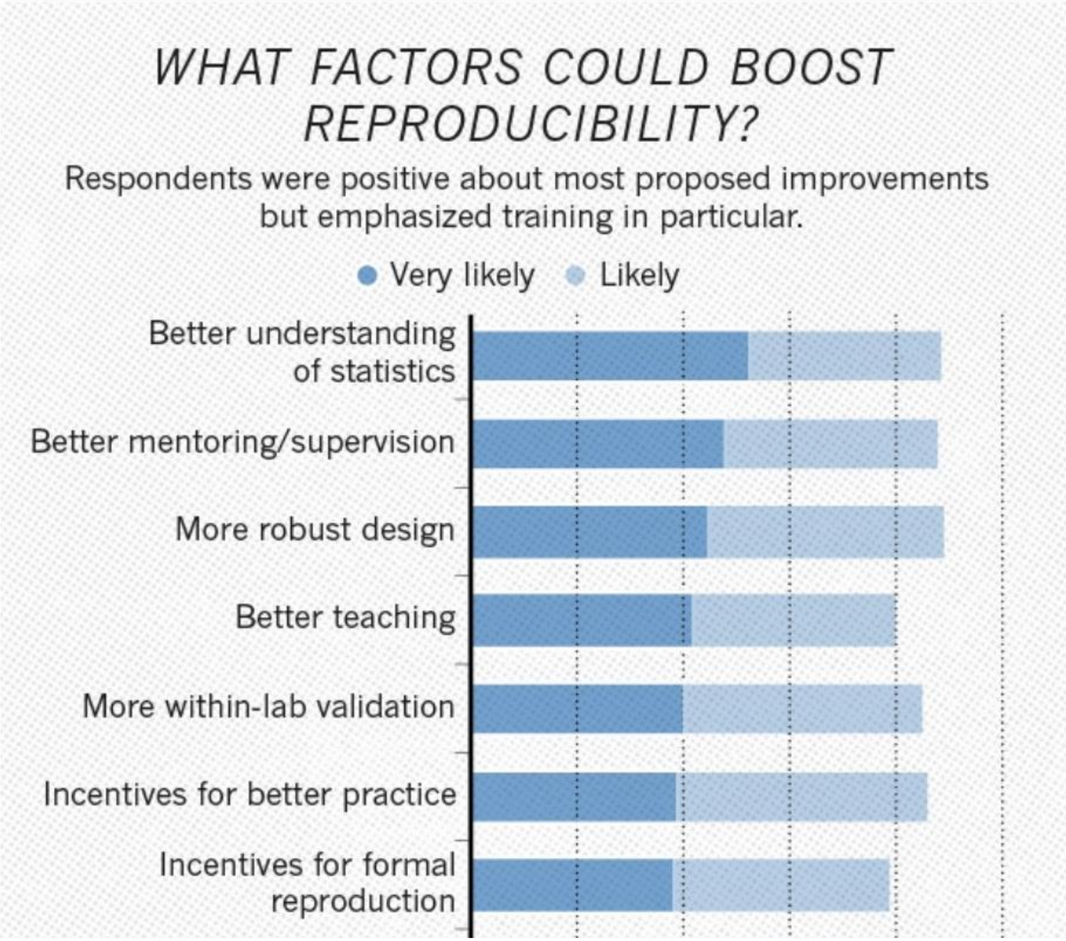
Torsten Hoefler
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

Roberto Belli
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
bellir@inf.ethz.ch

ABSTRACT
Measuring and reporting performance of parallel computers constitutes the basis for scientific advancement of high-performance

Reproducing experiments is one of the main principles of the scientific method. It is well known that the performance of a computer program depends on the application, the input, the compiler, the

Nature, 2016



TH, Belli: Scientific Benchmarking of Parallel Computing Systems, IEEE/ACM SC15

Conclusions and call for action

- **Performance may not be reproducible**
  - At least not for many (important) results
- **Interpretability fosters scientific progress**
  - Enables to build on results
  - Sounds statistics is the biggest gap today
- **We need to foster interpretability**
  - Do it ourselves (this is not easy)
  - Teach young students
  - Maybe even enforce in TPCs
- **See the 12 rules as a start**
  - Need to be extended (or concretized)
  - Much is implemented in LibSciBench [1]

No vegetables were harmed for creating these slides!

Acknowledgments

- **ETH's mathematics department (home of R)**
  - Hans Rudolf Künsch, Martin Maechler, and Robert Gantner
- **Comments on early drafts**
  - David H. Bailey, William T. Kramer, Matthias Hauswirth, Timothy Roscoe, Gustavo Alonso, Georg Hager, Jesper Träff, and Sascha Hunold
- **Help with HPL run**
  - Gilles Fourestier (CSCS) and Massimiliano Fatica (NVIDIA)

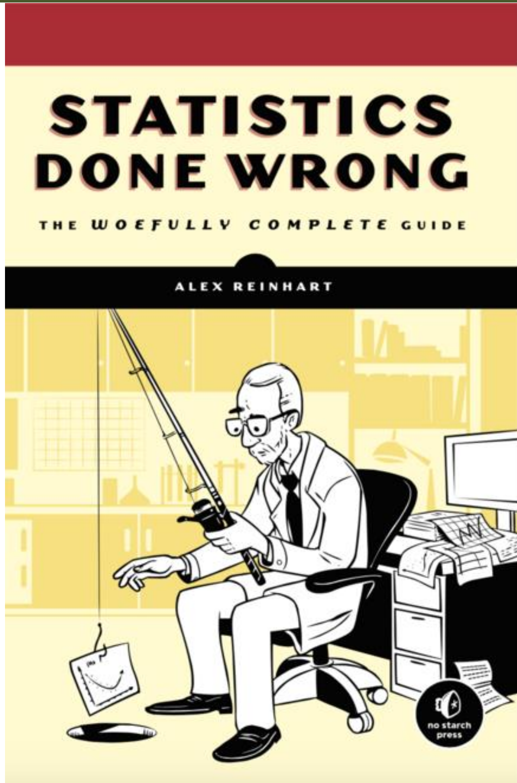
[1]: http://spcl.inf.ethz.ch/Research/Performance/LibLSB/.

How to not benchmark machine/deep learning workloads

"Twelve ways to fool the masses when reporting performance of deep learning workloads"
(my humorous guide to floptimize deep learning, blog post, see URL below)

https://htor.inf.ethz.ch/blog/index.php/2018/11/08/twelve-ways-to-fool-the-masses-when-reporting-performance-of-deep-learning-workloads/

“Statistical performance” vs. “hardware performance”



- **Tradeoffs between those two**
  - Very unusual for HPC people – we always operated in double precision
    Mostly out of fear of rounding issues
- **Deep learning shows how little accuracy one can get away with**
  - Well, examples are drawn randomly from some distribution we don’t know …
  - Usually, noise is quite high …
  - So the computation doesn’t need to be higher precision than that noise
    Pretty obvious! In fact, it’s similar in scientific computing but in tighter bounds and not as well known
- **But we HPC folks like flop/s! Or maybe now just ops or even aiops? Whatever, fast compute!**
  - A humorous guide to **floptimization**
  - Twelve rules to help present your (not so great?) results in a much better light

1) Ignore accuracy when scaling up!

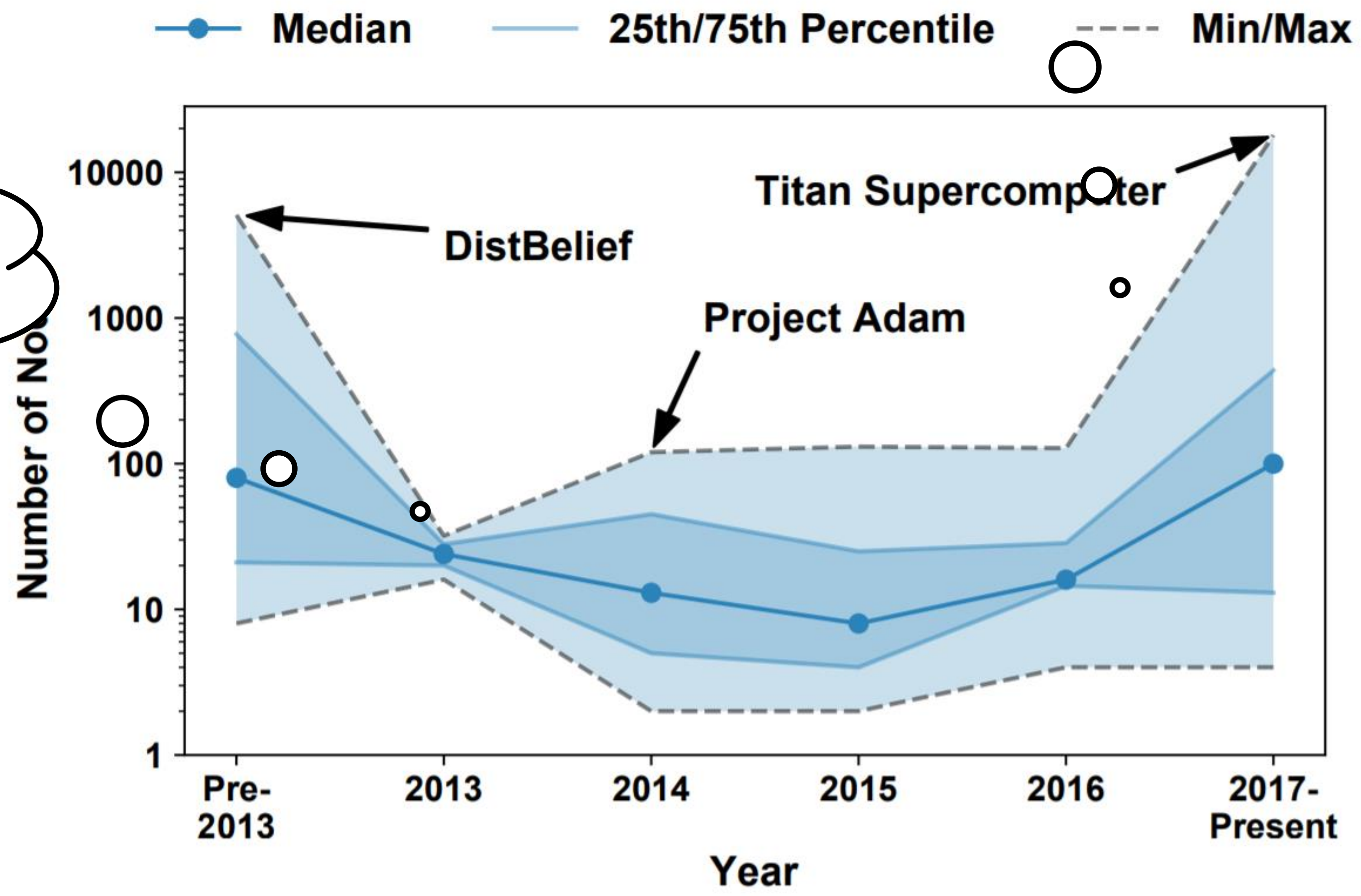
HPC picking up!

- **Too obvious for this audience**
  - Was very popular in 2015!

- **Surprisingly many (still) do this**

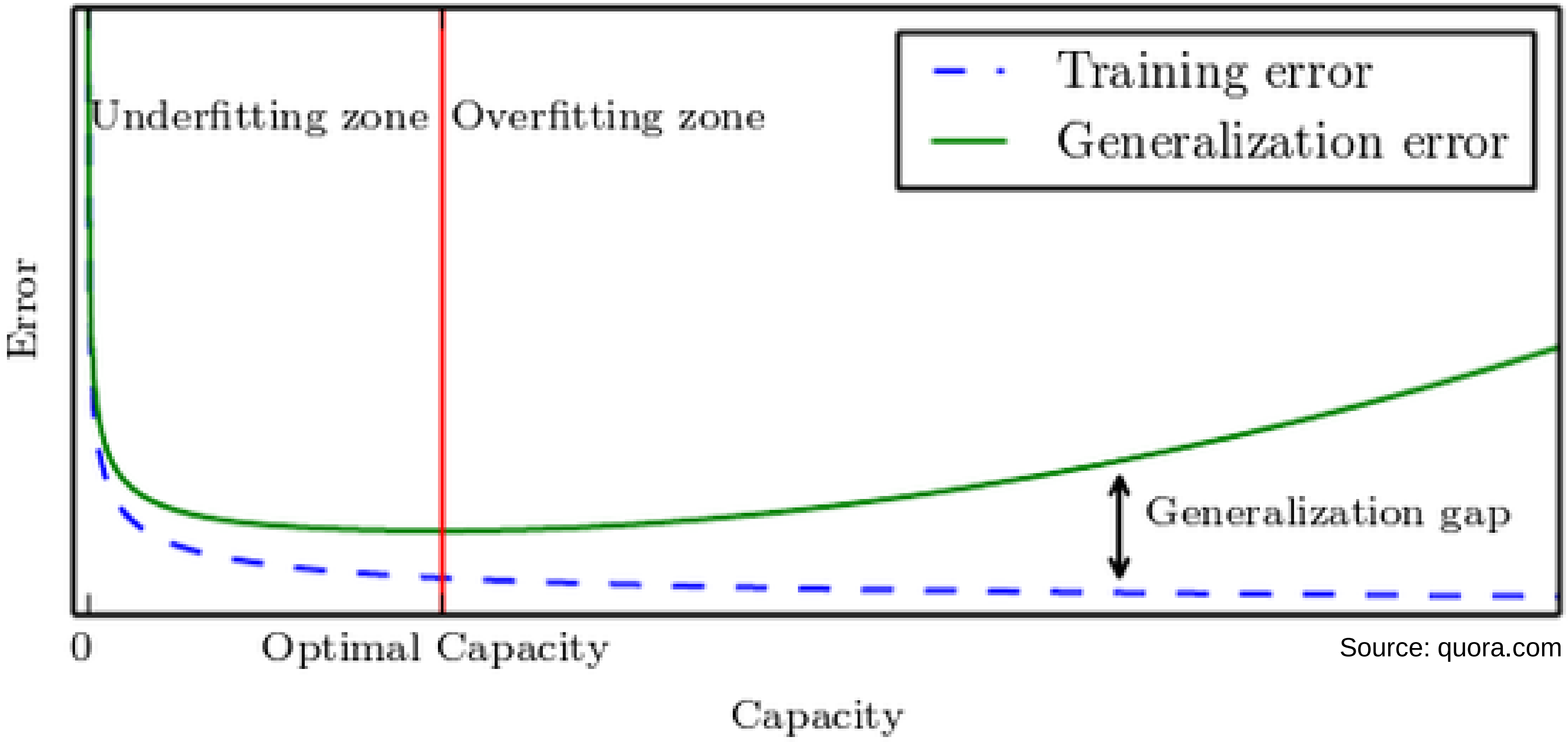
Learning community's self-correction (Y. LeCun)

Scalability without a good baseline? (D. Bailey)



2) Do not report test accuracy!

- **Training accuracy is sufficient isn't it?**

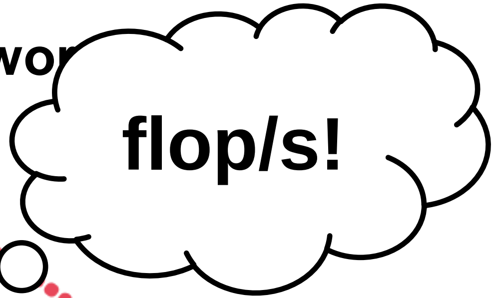


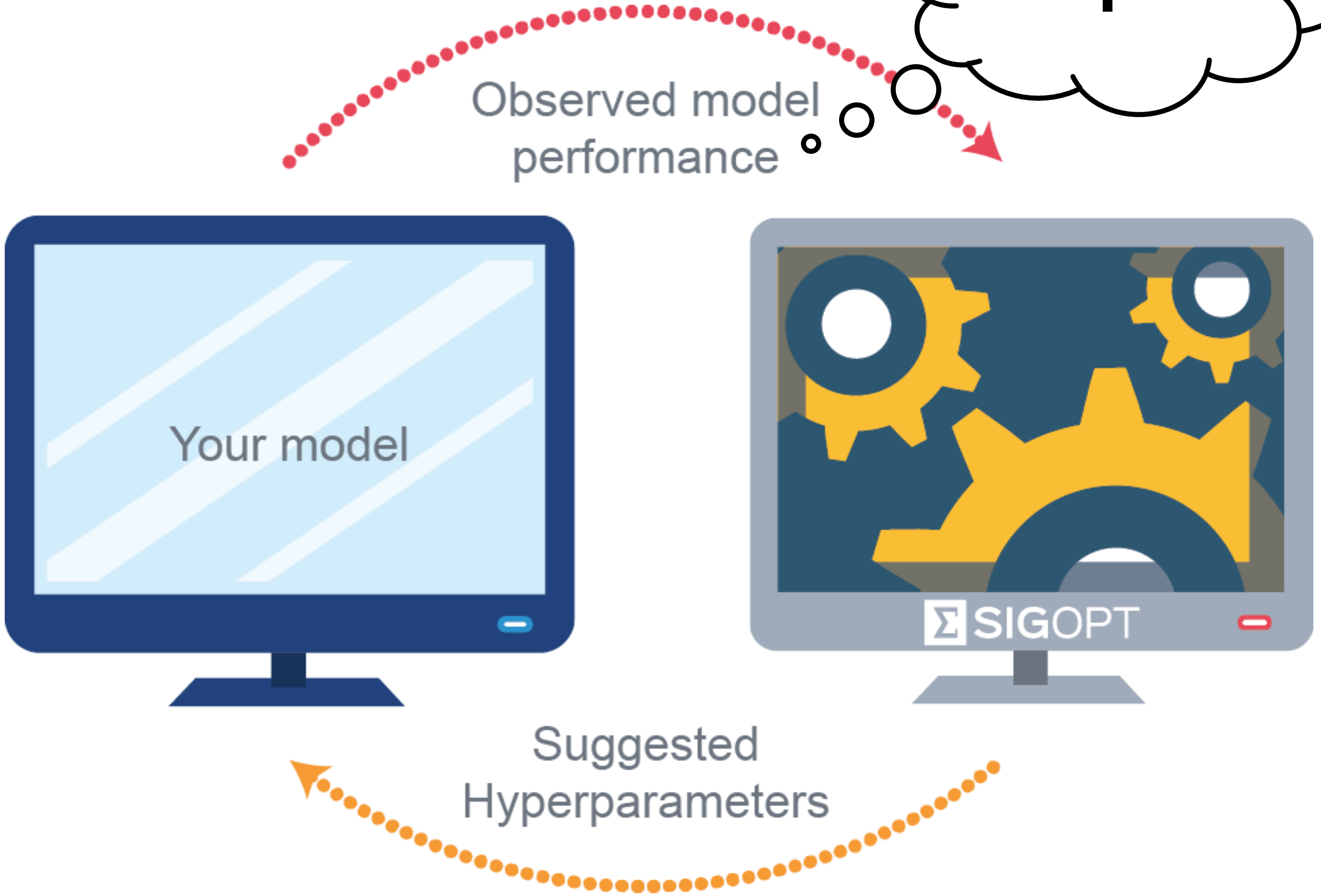
Source: quora.com

3) Do not report all training runs needed to tune hyperparameters!

- **Report the best run – SGD is a bit fragile, so don't wo**

  At the end, the minutes for the final run matter most!

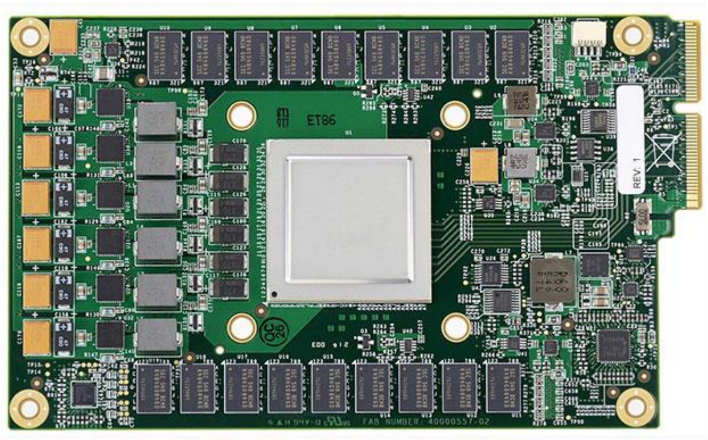




4) Compare outdated hardware with special-purpose hardware!

- **Tesla K20 in 2018!?**

  Even though the older machines would win the beauty contest!



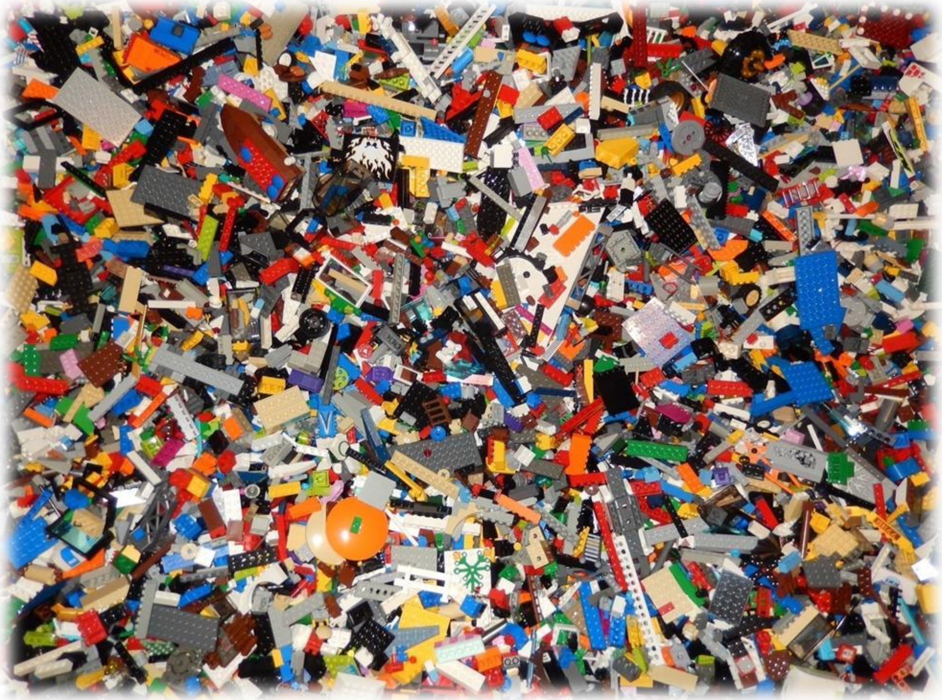
vs.

5) Show only kernels/subsets when scaling!

- **Run layers or communication kernels in isolation**
  - Avoids issues with accuracy completely ☺

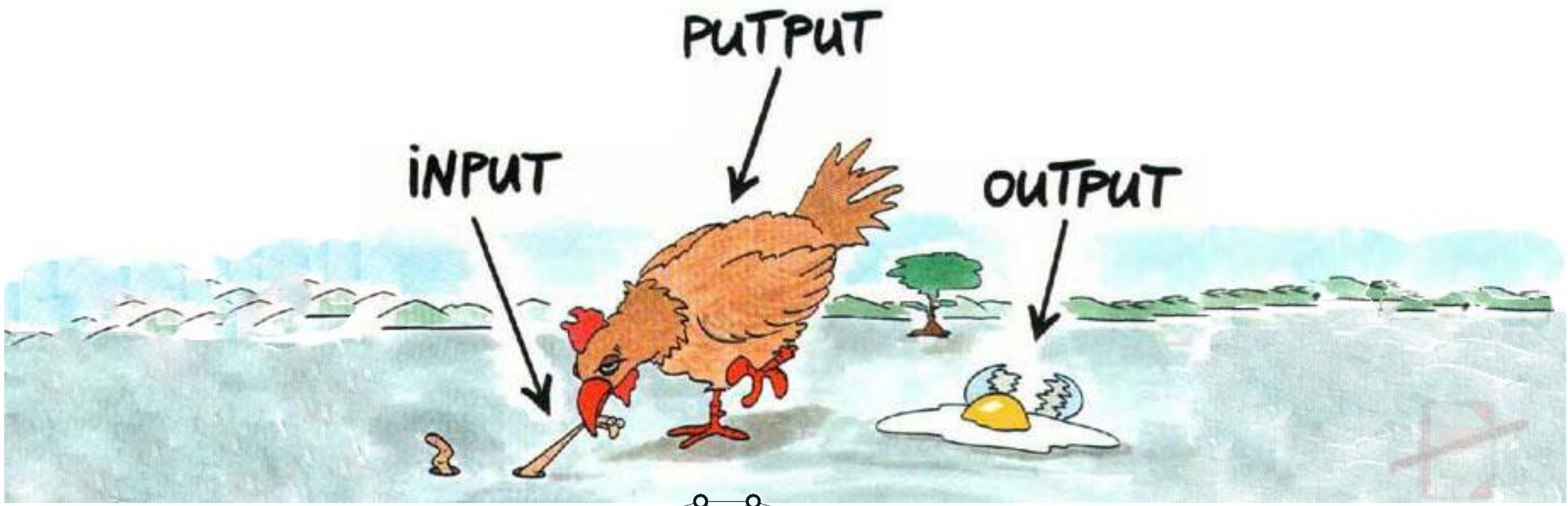
    Doesn't that look a bit like GoogLeNet?

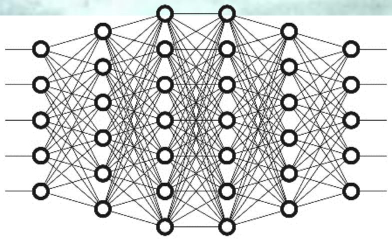
vs.



6) Do not consider I/O!

- **Reading the data? Nah, make sure it's staged in memory when the benchmark starts!**





7) Report highest ops numbers (whatever that means)!

- **Yes, we're talking ops today, 64-bit flops was so yesterday!**
  - If we don't achieve a target fast enough, let's redefine it!

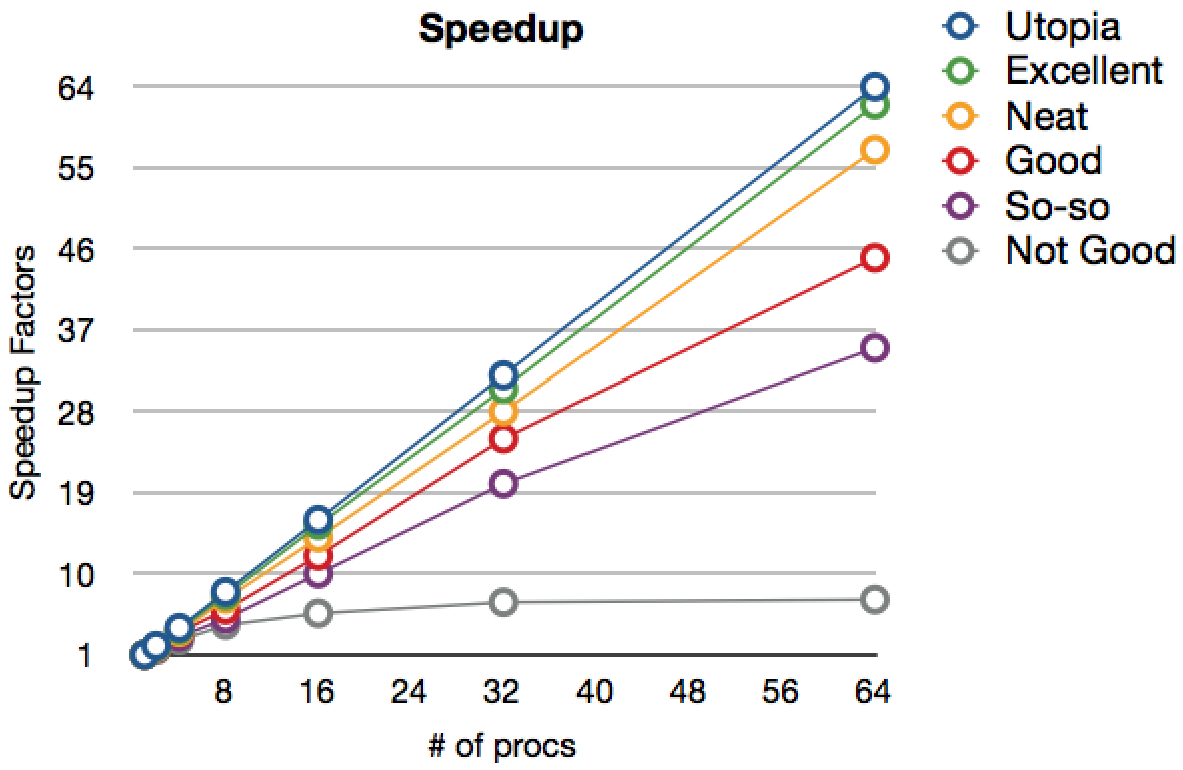
    And never talk about how many more of those ops one needs to find a solution, it's all about the rate, op/s!
- **Actually, my laptop achieves an "exaop":**
  - each of the 3e9 transistors switching a binary digit each at 2.4e9 Hz

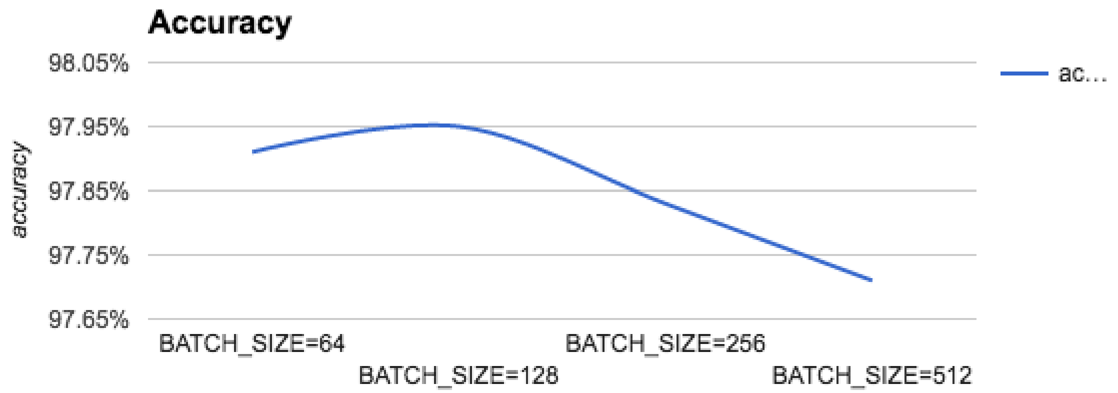
VS.

8) Show performance when enabling option set A and show accuracy when enabling option set B!

- **Pretty cool idea isn't it? Hyperparameters sometimes conflict**

  So always tune the to show the best result, whatever the result shall be!





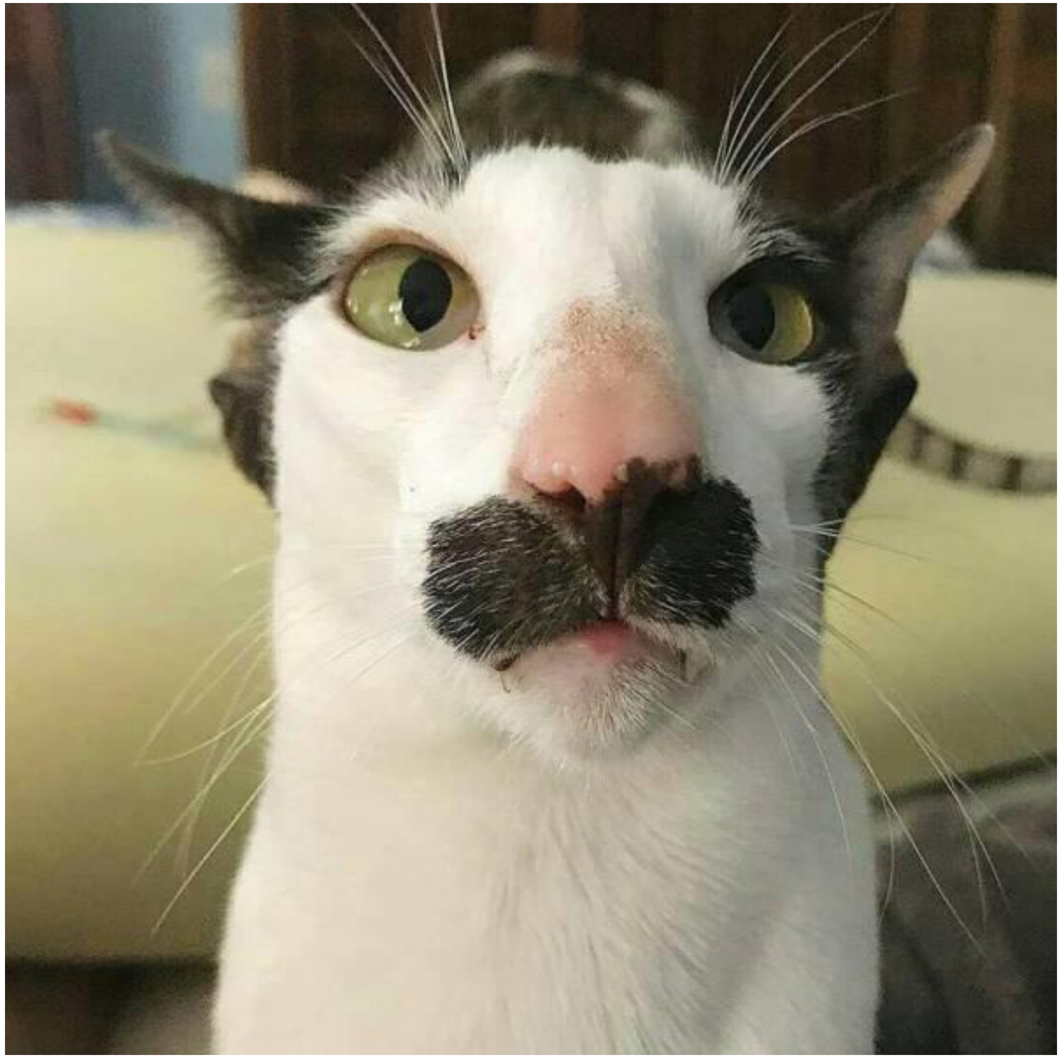
9) Train on (unreasonably) large inputs!

- **The pinnacle of floptimization! Very hard to catch!**

  But Dr. Catlock Holmes below can catch it.

Low-resolution cat (244x244 – 1 Gflop/example)

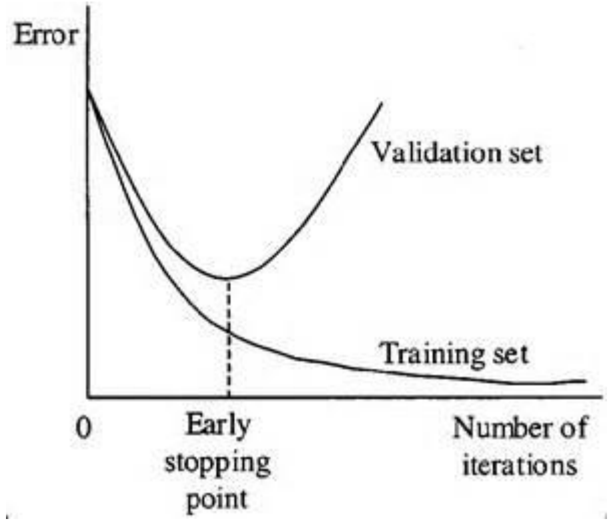
VS.



High-resolution cat (8kx8k – 1 Tflop/example)

10) Run training just for the right time!

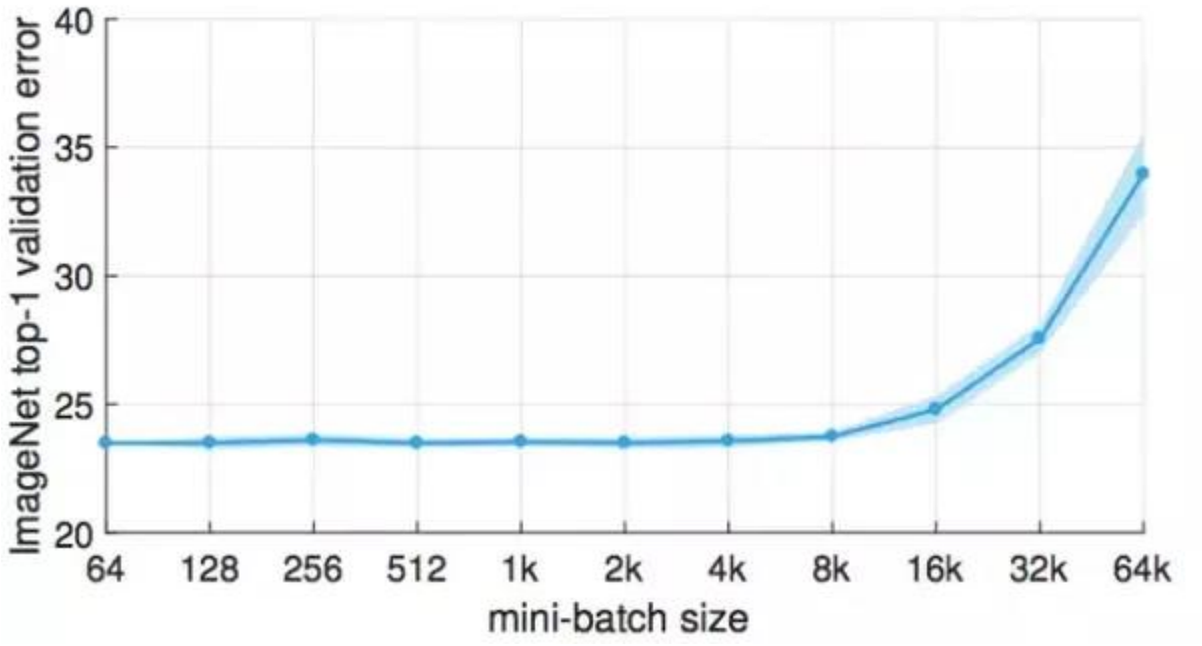
- **Train for fixed wall-time when scaling processors**
  - so when you use twice as many processors you get twice as many flop/s!
    But who cares about application speedup?



11) Minibatch sizing for fun and profit – weak vs. strong scaling.

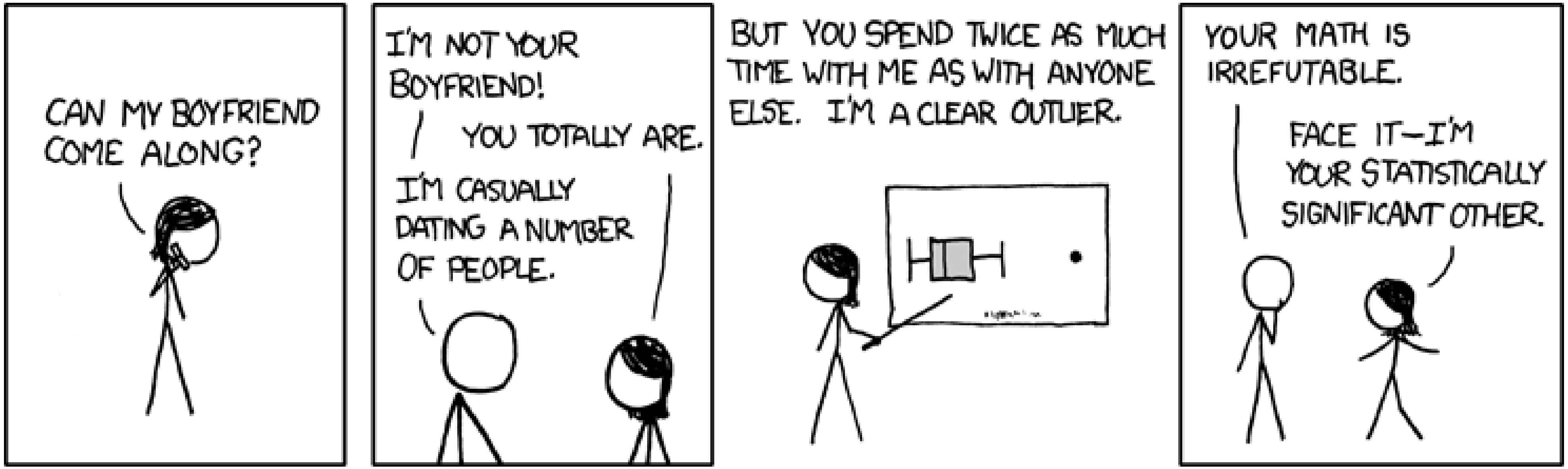
- **All DL is strong scaling – limited model and limited data**
- **So just redefine the terms relative to minibatches:**
  - Weak scaling keeps MB size per process constant – overall grows (less iterations per epoch, duh!)
  - Strong scaling keeps overall MB size constant (better but harder)

- **Microbatching is not a problem!**



12) Select carefully how to compare to the state of the art!

- **Compare either time to solution or accuracy if both together don't look strong!**
  There used to be conventions but let's redefine them.



Conclusions and call for action

- **Performance may not be reproducible**
  - At least not for many (important) results
- **Interpretability fosters scientific progress**
  - Enables to build on results
  - Sounds statistics is the biggest gap today
- **We need to foster interpretability**
  - Do it ourselves (this is not easy)
  - Teach young students
  - Maybe even enforce in TPCs
- **See the 12 rules as a start**
  - Need to be extended (or concretized)
  - Much is implemented in LibSciBench [1]

No vegetables were harmed for creating these slides!

Acknowledgments

- **ETH's mathematics department (home of R)**
  - Hans Rudolf Künsch, Martin Maechler, and Robert Gantner
- **Comments on early drafts**
  - David H. Bailey, William T. Kramer, Matthias Hauswirth, Timothy Roscoe, Gustavo Alonso, Georg Hager, Jesper Träff, and Sascha Hunold
- **Help with HPL run**
  - Gilles Fourestier (CSCS) and Massimiliano Fatica (NVIDIA)

[1]: http://spcl.inf.ethz.ch/Research/Performance/LibLSB/.