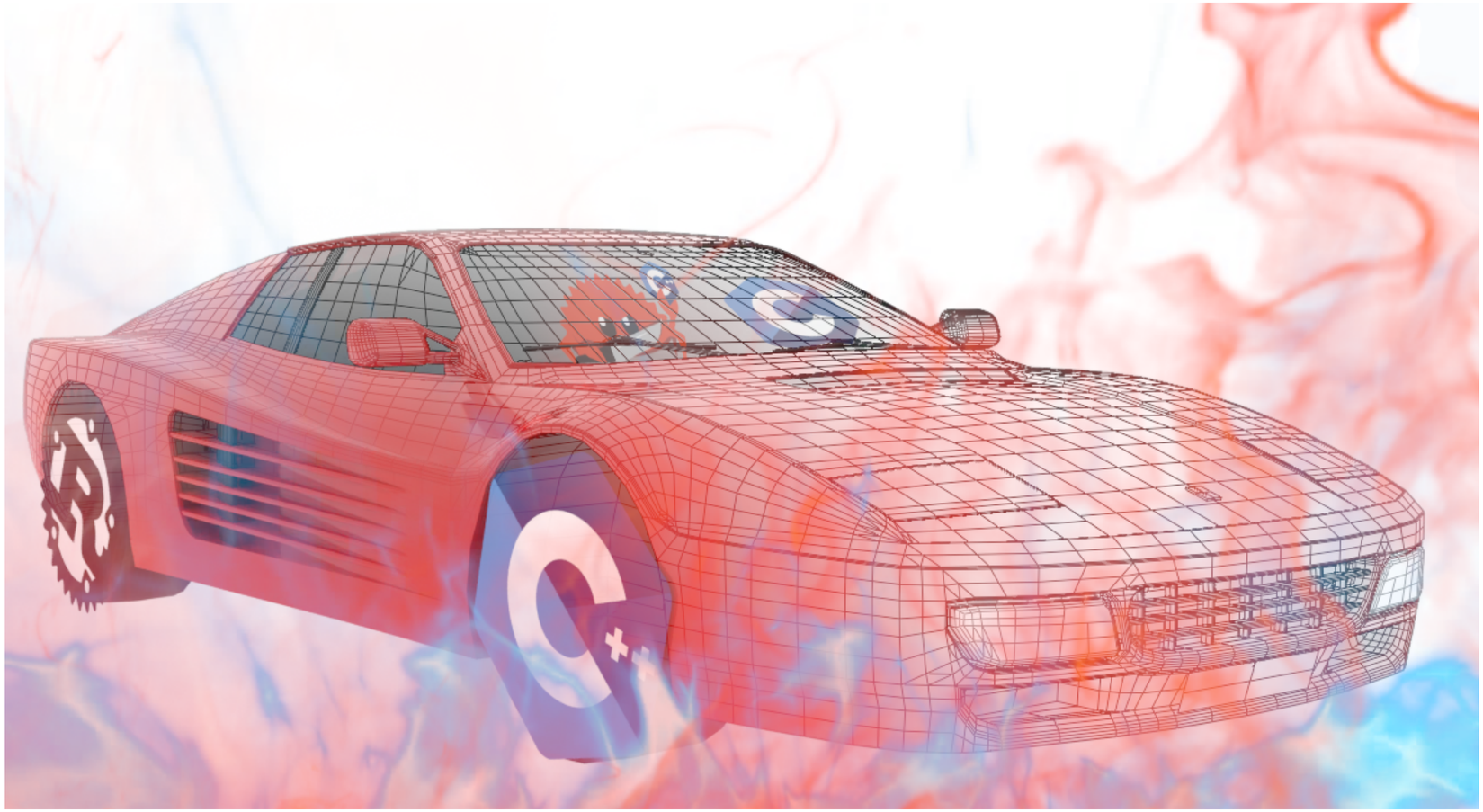
Effizientes Programmieren in C, C++ und Rust

Benchmarking

Colin Bretl, Nikolai Maas, Peter Maucher | 24.10.2024



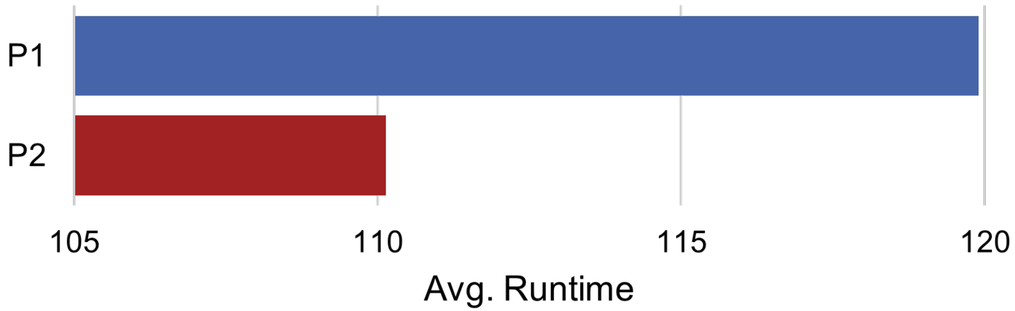
Questions of Today

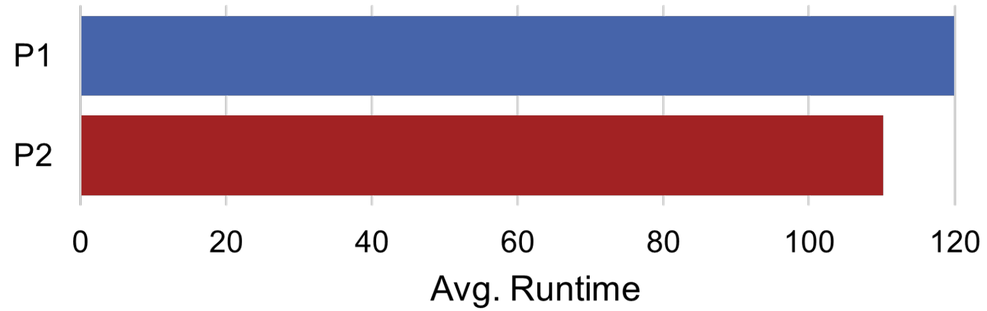
1. What do you need for benchmarking?
2. How noisy is your computer?
3. When do you believe that you are better than others?
4. When to believe that you are consistently good?

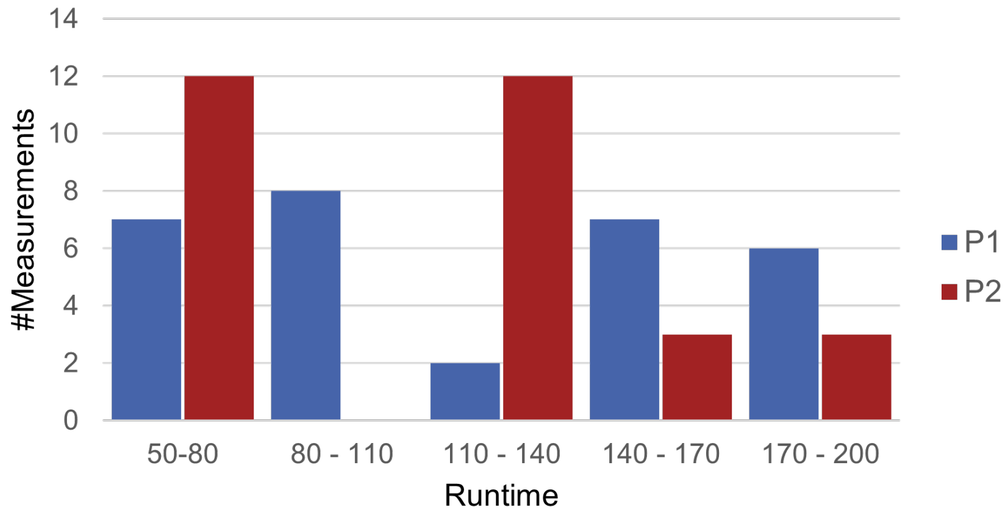
Table of Contents

Motivation

Motivation



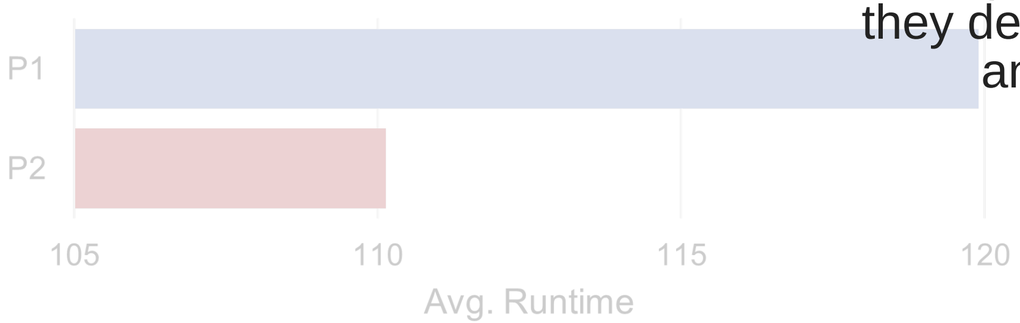


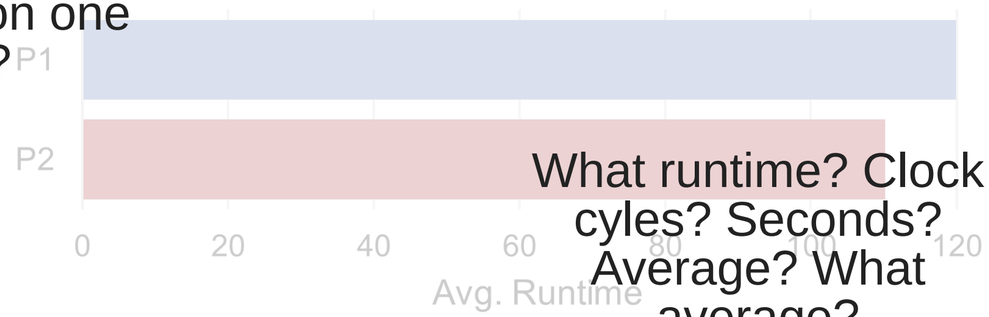


	P1	**P2**
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

Motivation

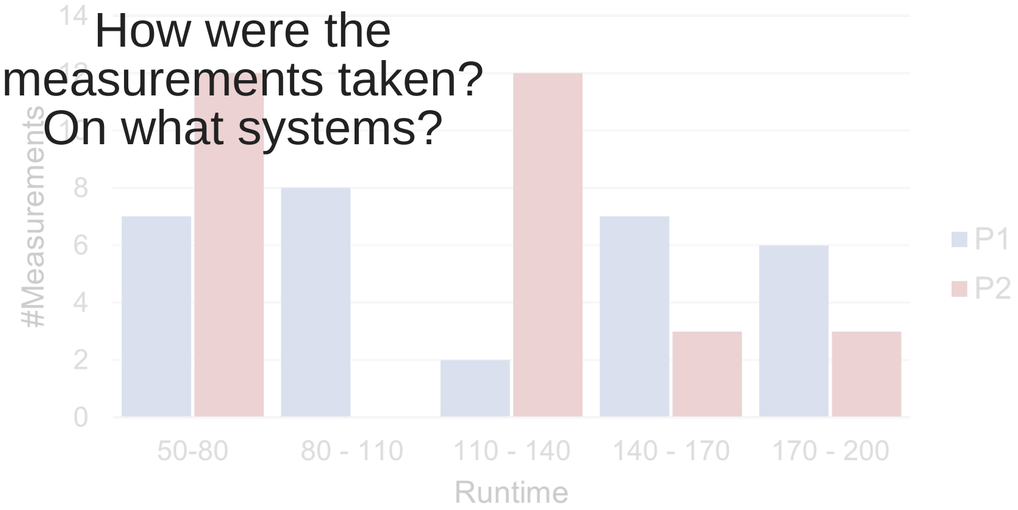
What programs? Do they depend on one another?





What runtime? Clock cyles? Seconds? Average? What average?

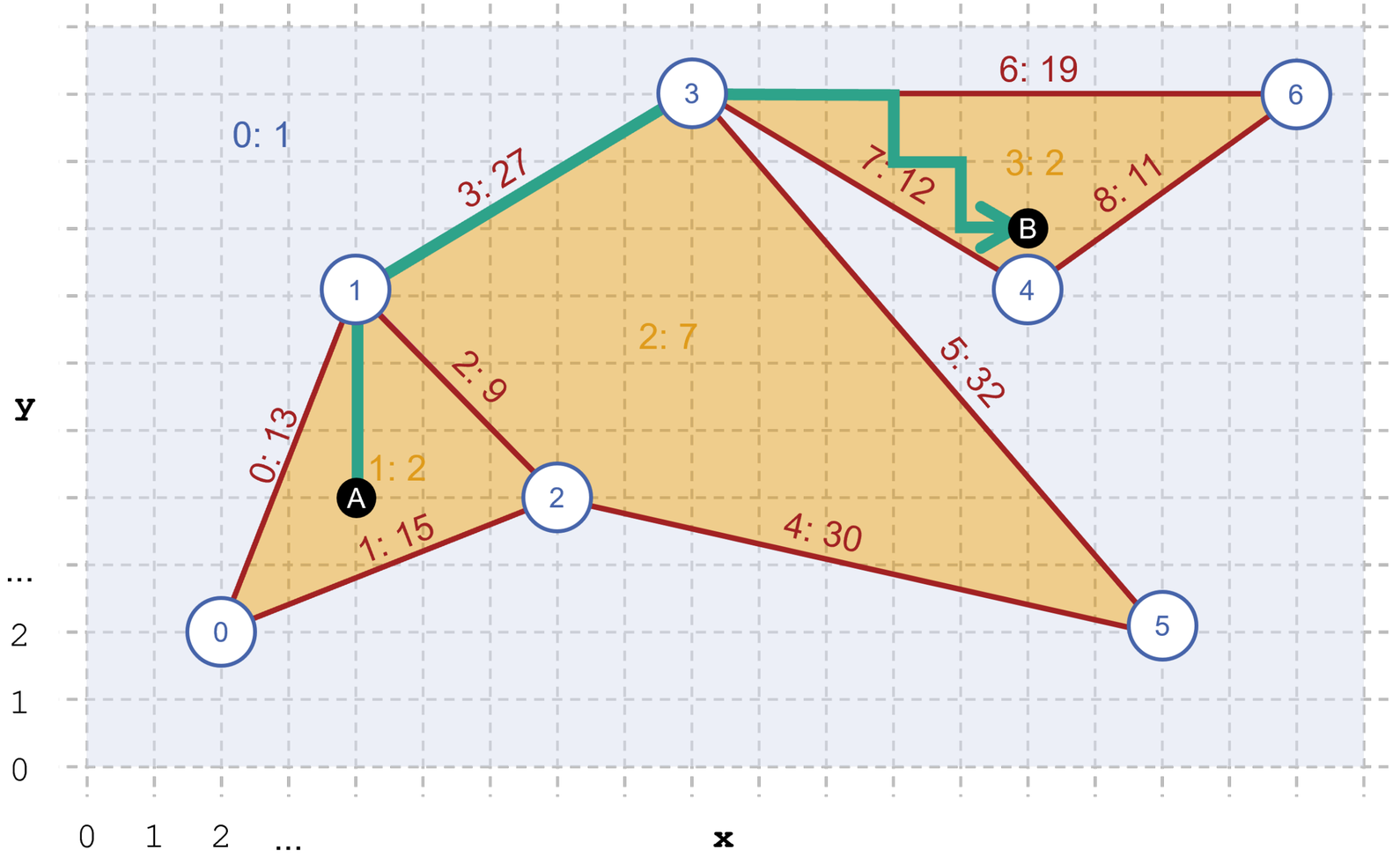
How were the measurements taken? On what systems?



How to interpret all of those?

	P1	P2
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

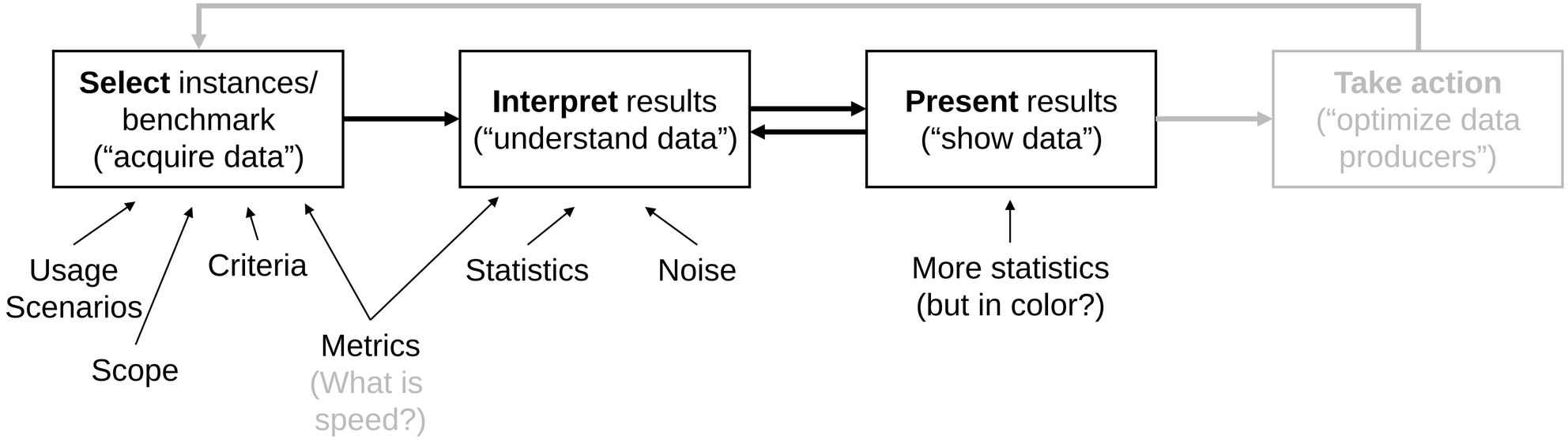
Toy Problem: Optimal Routing



Benchmarking

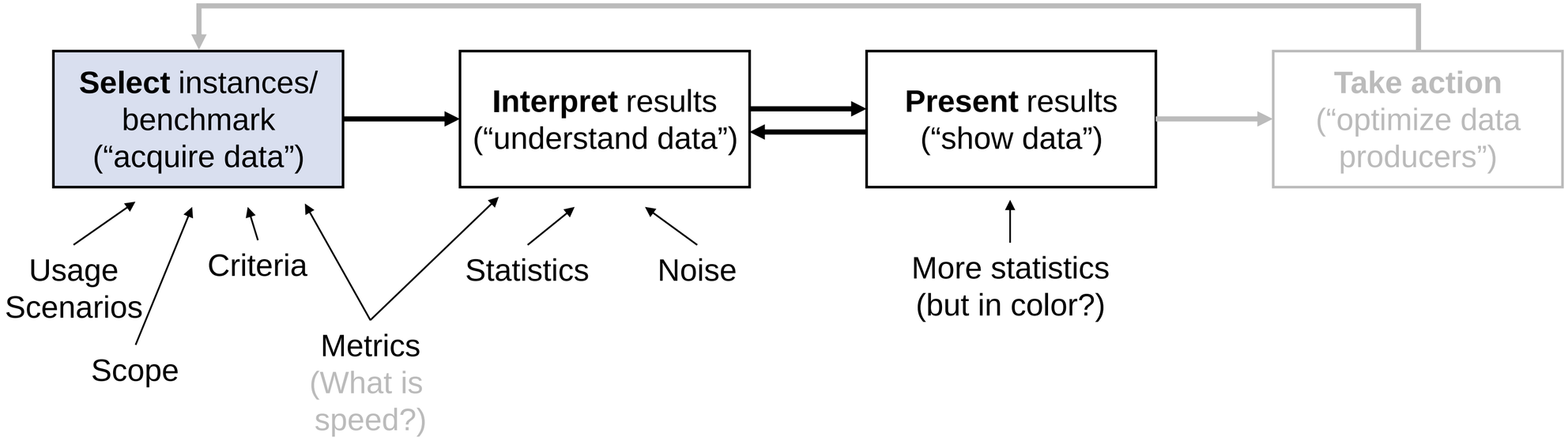
Benchmarking

"A benchmark is a tool coupled with a methodology for the evaluation and comparison of systems or components (system under test, SUT) in defined usage scenarios (workload) with respect to specific characteristics (metrics), e.g., performance."



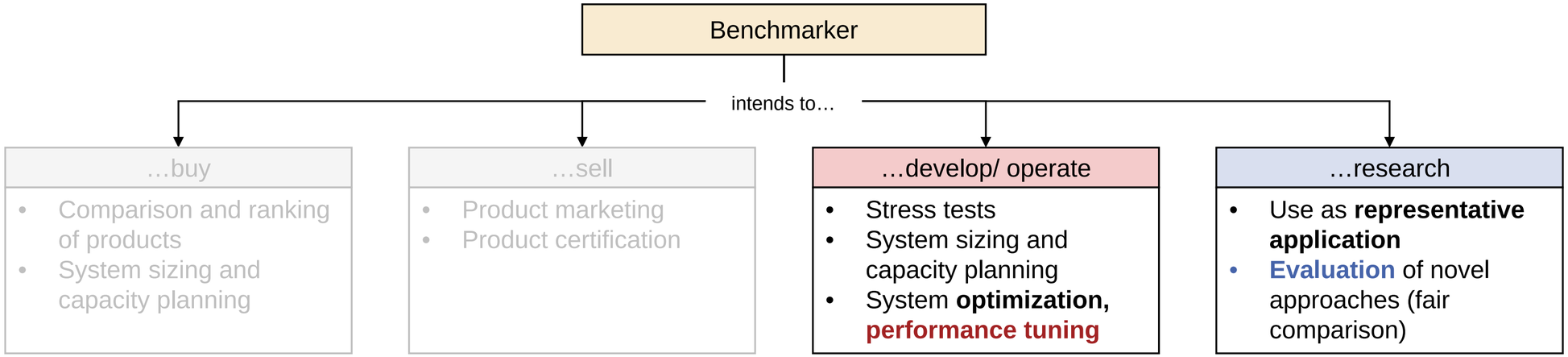
Benchmarking

"A benchmark is a tool coupled with a methodology for the evaluation and comparison of systems or components (system under test, SUT) in defined usage scenarios (workload) with respect to specific characteristics (metrics), e.g., performance."



Benchmarking

Usage Scenarios



OPTIMIZATION BENCHMARKS

- Help to identify bottlenecks
- Steer direction of improvement
  - Does not need to be 100% accurate
  - Should remain consistent
  - Fewer instances and runs
  - Automated (including evaluation)
  - Additional outputs (E.g.,...?)

EVALUATION BENCHMARKS

- Asses and compare different options
- Focus on accuracy and use case
- More instances and runs
- Manual evaluation acceptable or desired

style="position:absolute; top:835px; left:650px; width:10%" />

Benchmarking

Quality Criteria

- Relevance (How closely benchmark behaviour correlates to behaviours of interest)
  - Can heavily depend on usage scenario
- Reproducability (Producing consistent results when the benchmark is run with same configuration)
  - Run-to-run and between testers
  - Ideally: function of hardware and software (perfect consistency)
- Fairness (Allowing different test configurations to compete on their merits without artificial limits)
  - Run rules
  - Prevent unrealistic configurations from taking advantage of simplistic nature of benchmarks
- Verifiability (Providing confidence that a benchmark result is correct)
- Usability (Avoiding roadblocks for users to run the benchmark in their test environments)

Example: Planning Competitions

- Relevance
  - Synthetic planning problems
  - "The goals are to promote planning research [...]"
- Reprocudability
  - All participating planners can be used as Apptainer images ("Docker for HPC")
  - Configurations are automatically extracted
- Fairness
  - CPU, memory and time limit
  - Different tracks with different scores (e.g., ignoring optimality of plans or not)
- Verifiability
  - Verifier that checks correctness
- Usability
  - Compulsory input language for planning problems
  - Results and test problems are published

Benchmarking

Scope

scope

- Application benchmarks
  (Real, representative application programs that do "real world" work)
  - 👍 Most realistic (and therefore relevant)
  - 👎 Often expensive or impossible to get
- Kernel benchmarks
  (Small programs that capture the essential portion of an application ("inner loop with all critical functions"))
  - 👍 Easy/ easier to obtain
  - 👍 Easy/ easier to port
  - 👎 Ignore OS, middleware,...)
  - 👎 May not stress memory realistically
- Microbenchmarks
  (Small programs to test some specific part (e.g., single function(s) or a small piece of code) independent of the rest)
  - 👍 Useful if limiting factor
  - 👍 Useful for libraries
  - 👎 Ignores context of SUT

Advantages/ disadvantages of each?

Benchmarking

Scope

- Synthetic benchmarks
  - Artificial programs that mimic characteristics of a class of applications
  - E.g., mix of important operations (sorting, encryption, floating point operations, ...)
  - Ideally similar to the actual problem
  - Cannot capture impact of interactions between operations

  👍 Some are useful

  👎 All are wrong

  👎 "Easier" to make SUT appear faster on these than on real data

Instance Selection

- Homogeneous
  - Does the SUT perform consistently under similar circumstances?
  - Useful for narrow use cases
- Heterogeneous
  - Useful for broad use cases
- Random vs. Realistic/ Synthetic vs. Real
  - Random instances: provide little structure to utilize (e.g., stickerwall); easy to generate
  - Real world instances: high quality insights, hard to obtain
  - Realistic/ synthetic: simplified model of reality, compromise
- Worst case vs. average case vs. best case
  - How does the SUT perform in pessimistic/ realistic/ optimistic scenarios?
  - How to generate reliably?
- Parametrization, Parameter exploration
  - Problem size (e.g., number of nodes and queries in toy problem)
  - Parameter space (node degree distribution, query distance distribution, edge cases,...)

Benchmarking

Instance Selection

- If it's so hard to get good instances, can't we just reuse them...?
  - Sure! (consistency; especially for evaluation benchmarks)
  - But: beware of overfitting! (especially for optimizing benchmarks)
  - Don't ~~train~~ optimize on ~~test~~ evaluation data!
- Granularity
  - Different benchmarks on same input (e.g., type of queries on same graph; these count as new instances!)
  - Entirely new input (e.g., new graph for new kind of query)

Instance Evaluation

- Within instances
  - Arithmetic mean ↗
- Between instances
  - Geometric mean ↗
- Between classes of instances
  - Can lead to better understanding of algorithm/ optimizations
  - Where does it perform very good?
  - Where does it perform very bad?

What is Speed?

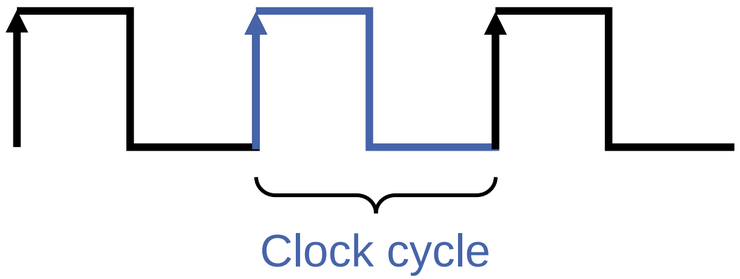
Baby don't bench me, don't bench me no more!

- Execution time
  - Wall-clock time (elapsed time from start of program to end of program)
  - CPU time (w/o I/O / other programs)
  - User-cpu time
  - System-cpu time
  - Waiting time

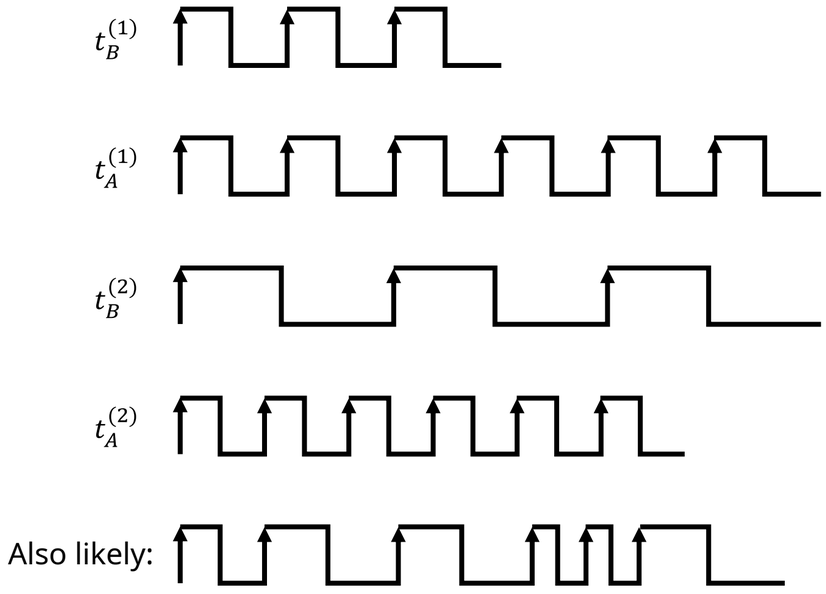
What is Speed?

Baby don't bench me, don't bench me no more!

- Execution time
  - Wall-clock time (elapsed time from start of program to end of program)
  - CPU time (w/o I/O / other programs)
  - User-cpu time
  - System-cpu time
  - Waiting time
- Clock rate (pulses per second) fundamentally affects performance (see example on the right)



- Consider an instruction that...
  - takes $c_A = 6$ clock cylces on machine A
  - takes $c_B = 3$ clock cylces on machine B
- Is B faster than A?



What is Speed?

Baby don't bench me, don't bench me no more!

- MIPS/ FLOPS
  - (Million) instructions per second
  - Aka misleading information to promote sales aka meaningleass information about processor speed
  - Difficult to compare across different architectures
  - Higher instructino throughput does not imply shorter execution time
  - Often used (see roofline model)

What is Performance?

style="width:60%" />

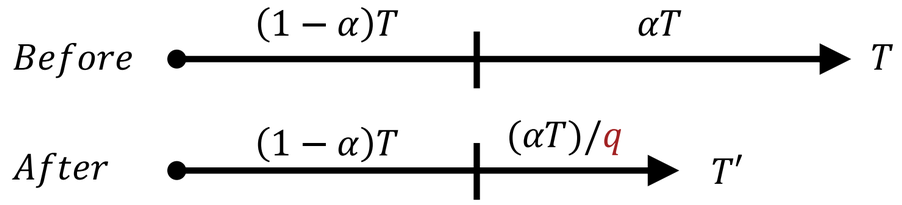
- Performance is the amount of useful work accomplished by a computer system"
  - Let W_i denote amount of work by system/ program/ component i
  - Express "usefulness" by relating it to time T_i needed for the work W_i
  - System speed (execution rate): $R_i = W_i / T_i$
  - Speed-up: $S_{(i,j)} = R_j / R_i$
  - Time for processing unit of work: $M_i = T_i / W_i$
- Classical performance benchmarking strategies
  - Fixed-work, i.e., $W_1 = W_2 = ... = W$
  - Fixed-time, i.e., $T_1 = T_2 = ... = T$
  - Variable

What is Performance?

- Performance is the amount of useful work accomplished by a computer system"
  - Let W_i denote amount of work by system/ program/ component i
  - Express "usefulness" by relating it to time T_i needed for the work W_i
  - System speed (execution rate): $R_i = W_i/T_i$
  - Speed-up: $S_{(i,j)} = R_j/R_i$
  - Time for processing unit of work: $M_i = T_i/W_i$
- Classical performance benchmarking strategies
  - Fixed-work, i.e., $W_1 = W_2 = ... = W$
  - Fixed-time, i.e., $T_1 = T_2 = ... = T$
  - Variable

Fixed-work (assesses runtime)

- Consider optimizing a single component of a program/ system at a time
- Only a fraction α of the whole run time is spent on the optimized component



- $q > 1$: optimization factor
- Speedup limit (Amdahl's Law, "perfect optimization"):

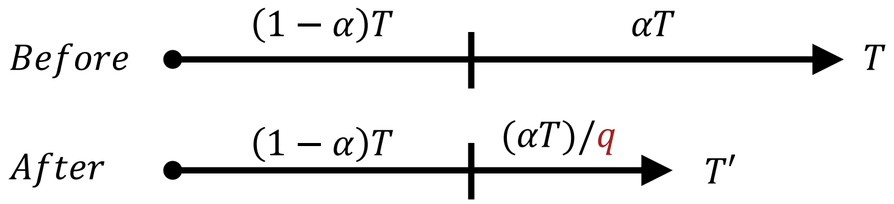
$$\lim_{q\to\infty} \frac{W/T}{W/T'} = \frac{1}{1-\alpha(1-1/q)} = \frac{1}{\alpha} < \infty$$

What is Performance?

- Performance is the amount of useful work accomplished by a computer system"
  - Let W_i denote amount of work by system/ program/ component i
  - Express "usefulness" by relating it to time T_i needed for the work W_i
  - System speed (execution rate): $R_i = W_i/T_i$
  - Speed-up: $S_{(i,j)} = R_j/R_i$
  - Time for processing unit of work: $M_i = T_i/W_i$
- Classical performance benchmarking strategies
  - Fixed-work, i.e., $W_1 = W_2 = ... = W$
  - Fixed-time, i.e., $T_1 = T_2 = ... = T$
  - Variable

Fixed-time (assesses throughput)

- Idea: whatever part is optimized leads to saving some time that can be used for processing other units of work
- Scaled version of Amdahl's law
- Consider same optimization factor q and a hypothetical runtime T_h without optimization



- New speedup limit:

$$\lim_{q\to\infty} S = \frac{T_h}{T} = \alpha q + (1-\alpha) = \infty$$

What is Performance?

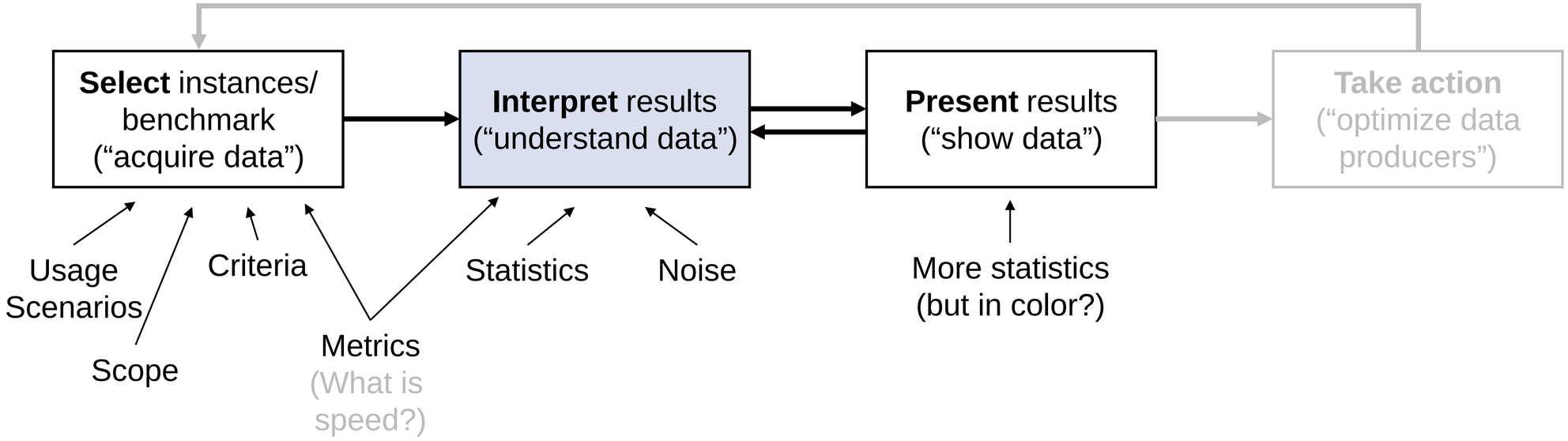
- Performance is the amount of useful work accomplished by a computer system"
  - Let W_i denote amount of work by system/ program/ component i
  - Express "usefulness" by relating it to time T_i needed for the work W_i
  - System speed (execution rate): $R_i = W_i / T_i$
  - Speed-up: $S_{(i,j)} = R_j / R_i$
  - Time for processing unit of work: $M_i = T_i / W_i$
- Classical performance benchmarking strategies
  - Fixed-work, i.e., $W_1 = W_2 = ... = W$
  - Fixed-time, i.e., $T_1 = T_2 = ... = T$
  - Variable

Variable

- Fixed-work is intuitive
- Fixed-time scales automatically
- Choose what is more reasonable/ doable in your context (often: fixed-work)
- Variable: fix neither T_i nor W_i
- More flexible but harder to interpret

Benchmarking

"A benchmark is a tool coupled with a methodology for the evaluation and comparison of systems or components (system under test, SUT) in defined usage scenarios (workload) with respect to specific characteristics (metrics), e.g., performance."



What kind of data do we deal with?

- Count how often something occurs (e.g., cache misses per second)
- Size of some parameter (amount of data written)
- Duration of a time interval (e.g., time for single instruction)
- Lots of samples, aggregate!

Data Scales (Service Slide)

Information density

Scale	Explanation	Mapping	Transformations	Operations	Example
Nominal	Categories/ labels with no meaningful ordering	1:1	1:1 substitutions	=, !=	Processor type (Intel, AMD, ...)
Ordinal	Meaningful ordering but distances among values are meaningless	+ ordering	Monotonic and increasing	=, !=, <, >	Processor generations, grades
Interval	Meaningful ordering and distances among values; 0 does not indicate absence	+ distances	M' = aM + b, a > 0	=, !=, <, >, +, -	CPU temperature (degree Celsius)
Ratio	Meaningful 0, same ratio at two different places on the scale carries same meaning	+ unit, 0	M' = aM, a > 0	=, !=, <, >, +, -, *, /	Execution Time
Absolute	Ratio scale with natural unit	+ natural unit	Identity, M' = M		Probability of a cache miss

What is a good metric?

- **Easy to measure**
  Easier metrics are more likely to be used and determined correctly
- **Repeatable**
  If the metric is measured multiple times using the same procedure, the same value should be measured (small differences are usually acceptable)
- **Reliable**
  A good metric should rank systems consistently with respect to the property under evaluation (e.g., higher means better)
- **Linear**
  A metric is linear, if its value is linearly proportional to the degree to which the SUT exhibits it (e.g., a twice as high value should indicate a twice as good performance)
- **Consistent**
  A good metric should have the same units and definition across different systems or configurations
- **Independent**
  A good metric should not be influenced by hidden agendas or interests (e.g., of processor vendors)

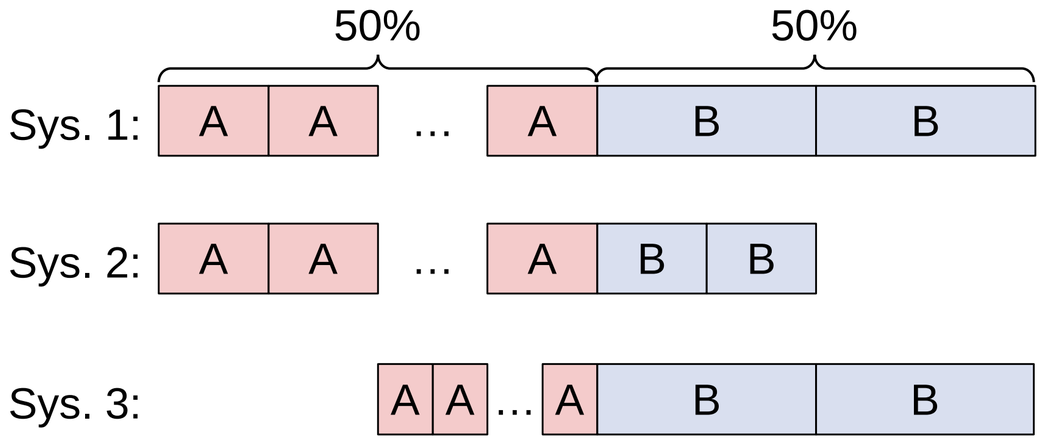
	Clock Rate	Instructions per Second	FLOP
Easy to measure	YES	YES	YES
Repeatable	(YES)	YES	YES
Reliable	NO (faster clockrate ! = better perf.)	NO	NO
Linear	NO	NO	NO
Consistent	YES	NO	NO
Independent	YES	YES	NO

Benchmarking

What is a good metric?

	System 1	System 2	System 3
Sub-Program A (e.g. find facette)	$10ms$	$10ms$	$5ms$
Sub-Program B (e.g. routing)	$1000ms$	$500ms$	$1000ms$
Avg. ($\overline{x}$)	$505ms$	$255ms$	$502.5ms$

- System 2 appears to be fastest
- What if System 1 spends 50% of its total runtime in both A and B?



- System 2 and 3 save same amount of time!
- Arithmetic mean is unfair for this kind of comparison!
- It is fairer to compare speedups here

Benchmarking

What is a good metric?

	System 1	System 2	System 3
Sub-Program A	1	1	2
Sub-Program B	1	2	1
$\bar{x}$	1	1.5	1.5
Rank ($\bar{x}$)	2	1	1
$\bar{x}_g$	$\sqrt{1}$	$\sqrt{2}$	$\sqrt{2}$
Rank ($\bar{x}_g$)	2	1	1

- Shown: Speedups relative to System 1
- Speedups are a relative quantity!

	S 1	S 2	S 3
A	1	1	2
B	0.5	1	0.5
$\bar{x}$	0.75	1	1.25
Rank ($\bar{x}$)	3	2	1
$\bar{x}_g$	$\sqrt{0.5}$	$\sqrt{1}$	$\sqrt{1}$
Rank ($\bar{x}_g$)	2	1	1

	S 1	S 2	S 3
A	0.5	0.5	1
B	1	2	1
$\bar{x}$	0.75	1.25	1
Rank ($\bar{x}$)	3	1	2
$\bar{x}_g$	$\sqrt{0.5}$	$\sqrt{1}$	$\sqrt{1}$
Rank ($\bar{x}_g$)	2	1	1

- "Average speedup" is inconsistent and (can be manipulated by choice of baseline); use geometric mean!
- Geometric mean of $x_1, x_2, \ldots, x_n$:

$$\bar{x}_g = \left(\prod_{i=1}^{n} x_i \right)^{1/n} = \exp\left(\frac{1}{n} \sum_{i=1}^{n} \ln(x_i) \right)$$

Benchmarking

Different kinds of means

- Arithmetic mean: $\bar{x} = 1/n \sum_{i=1}^{n} x_i$
  - If raw values have an (absolute) meaning
  - Sensitive to outliers
- Geometric mean: $\bar{x}_g = (\prod_{i=1}^{n} x_i)^{1/n}$
  - If products of raw values have a meaning
  - E.g., average speedups
- Harmonic mean: $\bar{x}_h = n/(x_1^{-1} + ... + x_n^{-1})$
  - Inverse of arithmetic mean of reciprocals
  - For measured ratios (instructions per second, cycles per instruction, cache miss rate, ...)
- Mode (value that occurs most often)
- Median, quantiles ("value in the middle")

On composite metrics

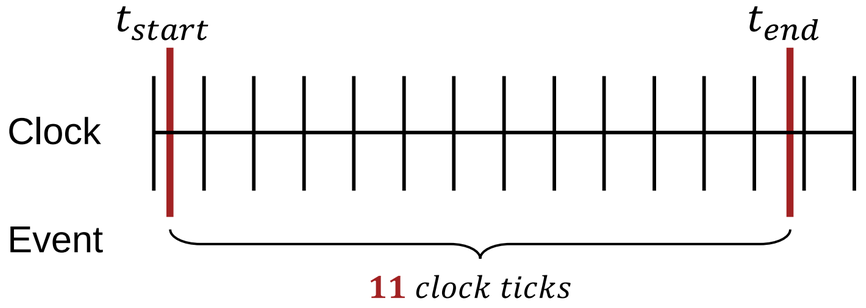
- Composite metrics are designed by aggregating multiple data into single values
  - Information loss
  - Loss of meaning (means of means of means...)
  - Single values should be weighted in a far way (e.g., to the respective fraction of time expected in the target application scenario)

Aggregating/ averaging data should be repeatable. Is it?

Benchmark Environments

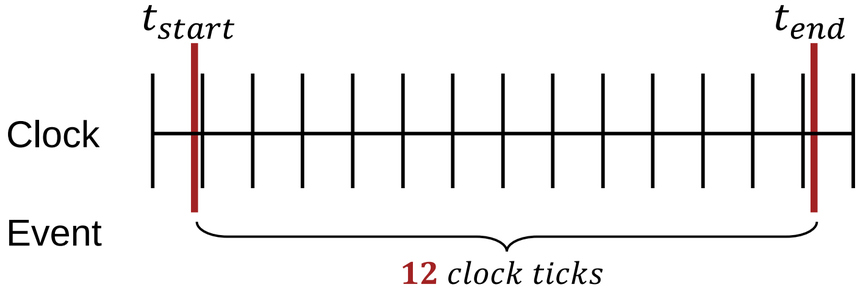
RANDOM NOISE

- Unpredictable
- Non-deterministic
- Limited to measurement tools
- Noise within system



SYSTEMATIC NOISE

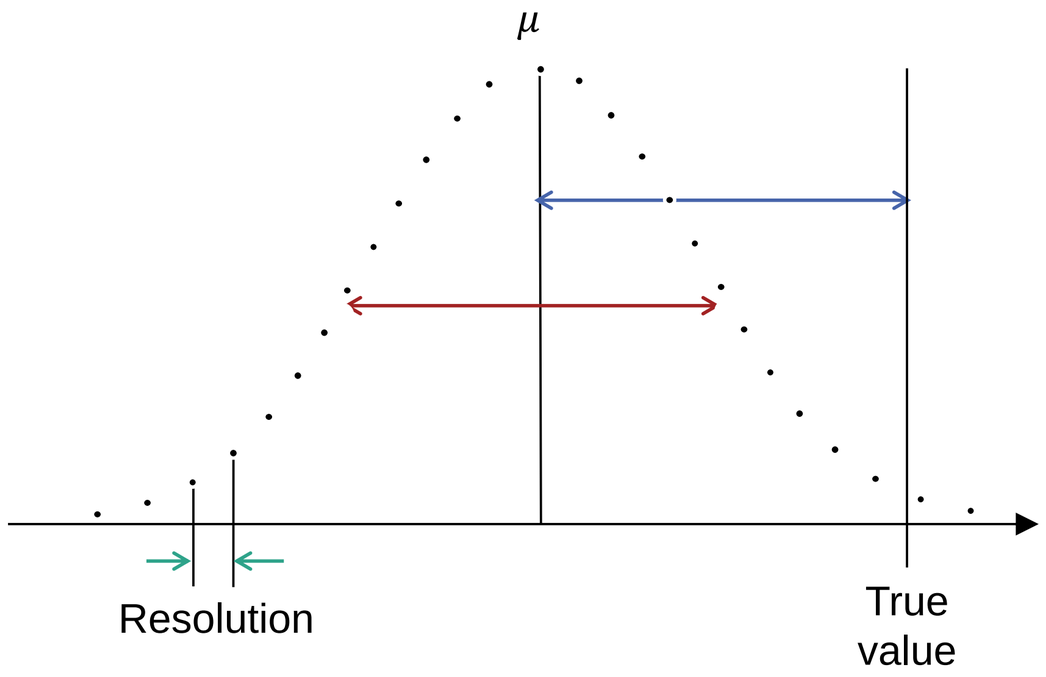
- Oversights or mistakes in the measurement process
- Different machines (architectures, ...) can have a significant impact on performance
- Hard to detect/ model
- https://www.youtube.com/watch?v=tDacjrSCeq4



A Model of Random Error

- Quantities affecting measurement quality
  1. Accuracy (absolute difference between true value and (mean of) actual value)
  2. Precision (characterizes repeatability/ variance; the more sources of random error, the lower)
  3. Resolution (Smallest difference between two possibly measured values)

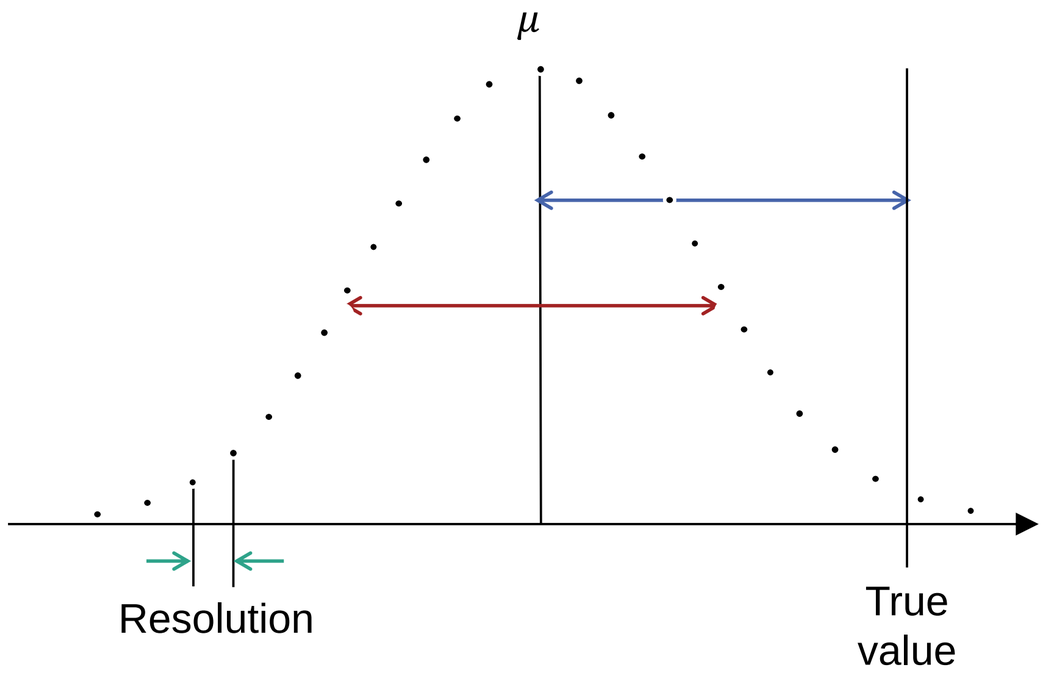
- For many error sources, it is reasonable to assume normally distributed errors
- Warning: this is only valid for random noise, not systematic noise



Benchmarking

How to deal with it?

- Many runs
- Isolated benchmark environment (dedicated server)
- Disable OS resource management (fixed CPU cores for processes,...)
- Include error bars in results (not always feasible; see later)
- ...



Variance

- Means measure central tendencies
- Variances measure disperions about mean
- $\sigma^2 = (\sum_{i=1}^{n}(x_i - \bar{x})^2)/n$
- Normalization: $\sigma = \sqrt{(\sigma^2)}$ (standard deviation)
- Treats all deviations from mean same
  - 👎 No directional information (slower/ faster)
  - 👎 Includes potential outliers

Statistical Tests

On Statistical Tests

- (Un-/Fortunately), this is not a lecture on probability theory and statistics
- Important results: Central Limit Theorem (if we run benchmark(s) often enough and they are not too odd, it is safe to assume that they are normally distributed)
- Statistical tests may help to argue about the significance of improvements on a more scientific/ academic appearing level

WHEN TO USE?

- Imho, more often than usually done*
- In rather evaluative settings
- Small n (e.g., comparing large instances of NP-hard problems)
- As reminder to think about significance, power, effect size, sample size before arguin
- When effects are expecte to be small

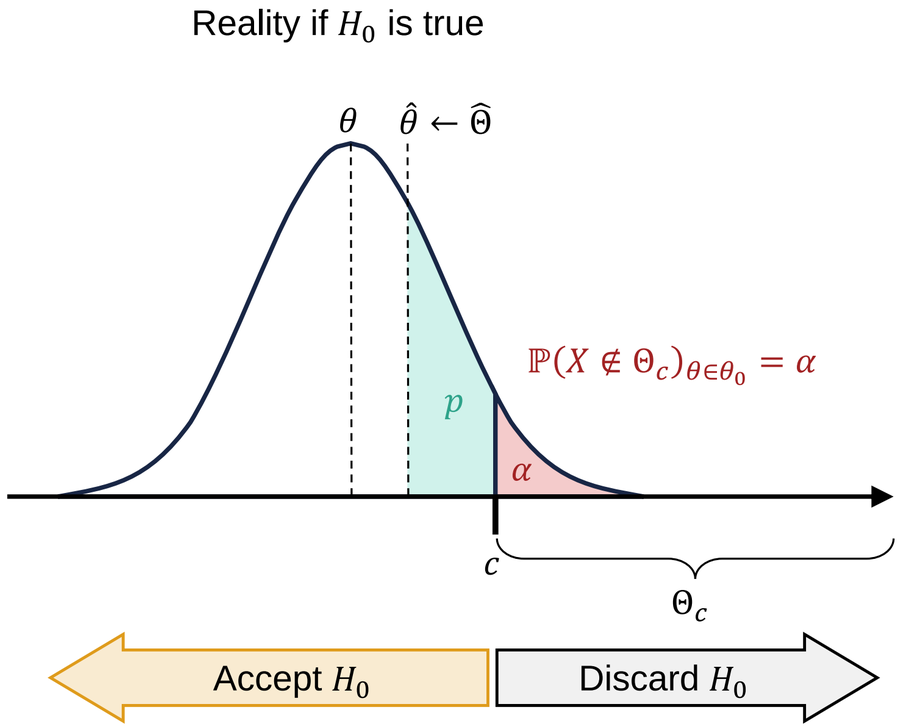
WHEN NOT TO USE?

- *If you don't know, what you're actually doing or testing
- In rater optimizing settings
- Larger n *(CLT goes brrr)*
- Mean and/or vriance suffice to eyeball differences

Statistical Tests

Setup

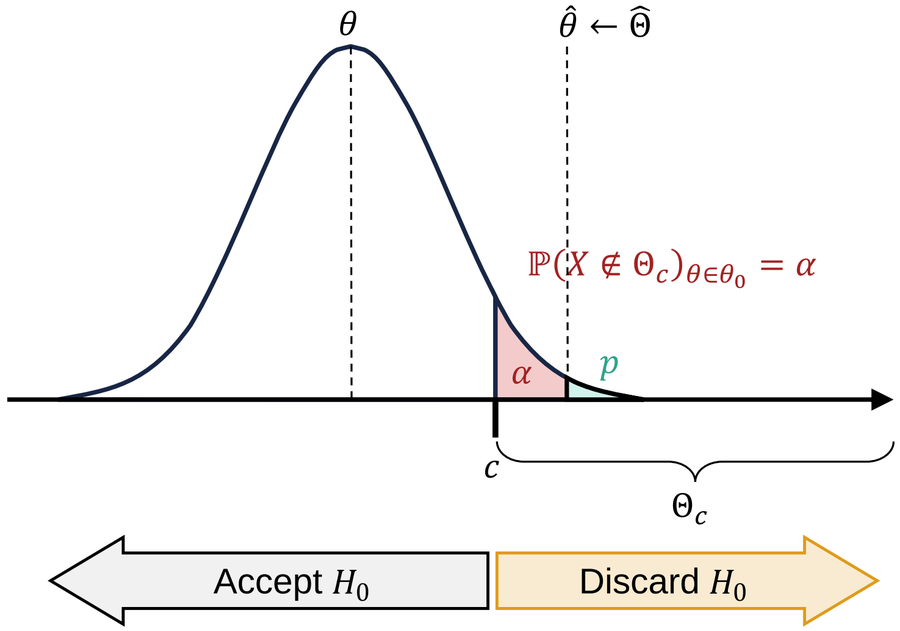
1. Confirm/ discard hypotheses *("We more speed?")*
   - Null hypothesis: $H_0 : \theta \in \Theta_0$
     ("We less speed") 👎
   - Alternative hypothesis: $H_1 : \theta \in \Theta_1$
     ("We more speed") 👍
   - Level of significance: α, e.g. $\alpha = 0.05$
     ("If re run this 100 times, how often are we more speed though we aren't?")
2. Use proper statistic to estimate $\hat{\theta}$ of θ ↑
3. Define a range of critical values Θ_c for which we believe that H_0 is wrong
4. Evaluate statistic (yields p-value) and interpret result: $p > \alpha$?
   (Probability to observe $\hat{\theta}$ or more extreme result under H_0. If we were less speed, this would be very unlikely)



Statistical Tests

Setup

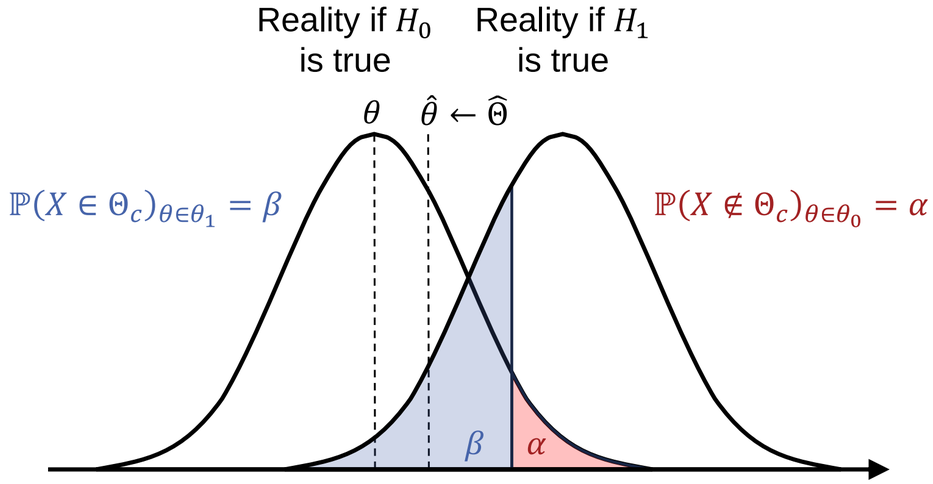
1. Confirm/ discard hypotheses *("We more speed?")*
   - Null hypothesis: $H_0 : \theta \in \Theta_0$
     ("We less speed") 👎
   - Alternative hypothesis: $H_1 : \theta \in \Theta_1$
     ("We more speed") 👍
   - Level of significance: α, e.g. $\alpha = 0.05$
     ("If re run this 100 times, how often are we more speed though we aren't?")
2. Use proper statistic to estimate $\hat{\theta}$ of θ ↑
3. Define a range of critical values Θ_c for which we believe that H_0 is wrong
4. Evaluate statistic (yields p-value) and interpret result: $p > \alpha$?
   (Probability to observe $\hat{\theta}$ or more extreme result under H_0. If we were less speed, this would be very unlikely)



Statistical Tests

Errors

- Error of first kind: Discard H_0 although it is true (quantified by α, e.g., 0.05)
  (We believe to be more speed though we aren't)
- Error of second kind: Accept H_0 although it is false (quantified by β, e.g., 0.2)
  (We believe to be less speed though we aren't)
- $1 - \beta$ is called (statistical) power
  - "Missed chance"
  - Tradeoff: Increasing $\alpha \Rightarrow$ Decreasing $\beta \Rightarrow$ Better power, error of first kind more likely!



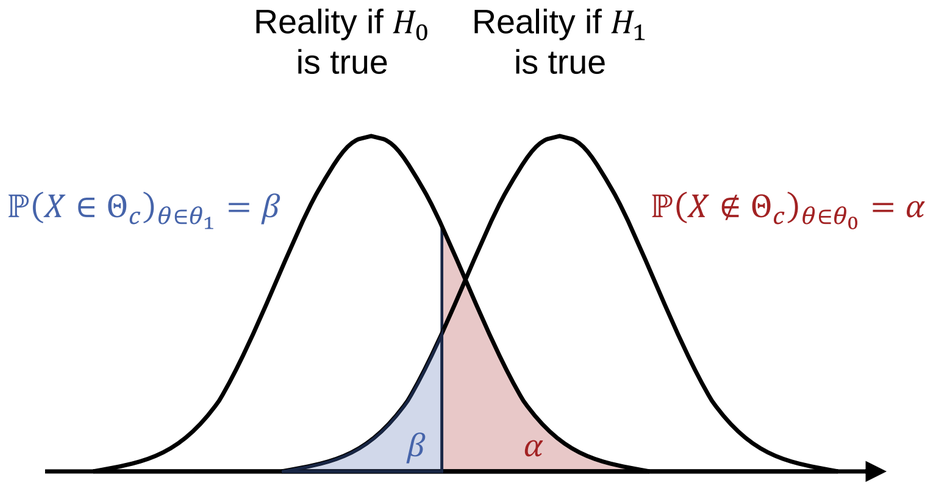
(Unknown) Reality

Decision	H_0 is true	H_0 is false
Reject H_0	α	$1 - \beta$
Accept H_0	$1 - \alpha$	β

Statistical Tests

Errors

- Error of first kind: Discard H_0 although it is true (quantified by α, e.g., 0.05)
  (We believe to be more speed though we aren't)
- Error of second kind: Accept H_0 although it is false (quantified by β, e.g., 0.2)
  (We believe to be less speed though we aren't)
- $1 - \beta$ is called (statistical) power
  - "Missed chance"
  - Tradeoff: Increasing $\alpha \Rightarrow$ Decreasing $\beta \Rightarrow$ Better power, error of first kind more likely!



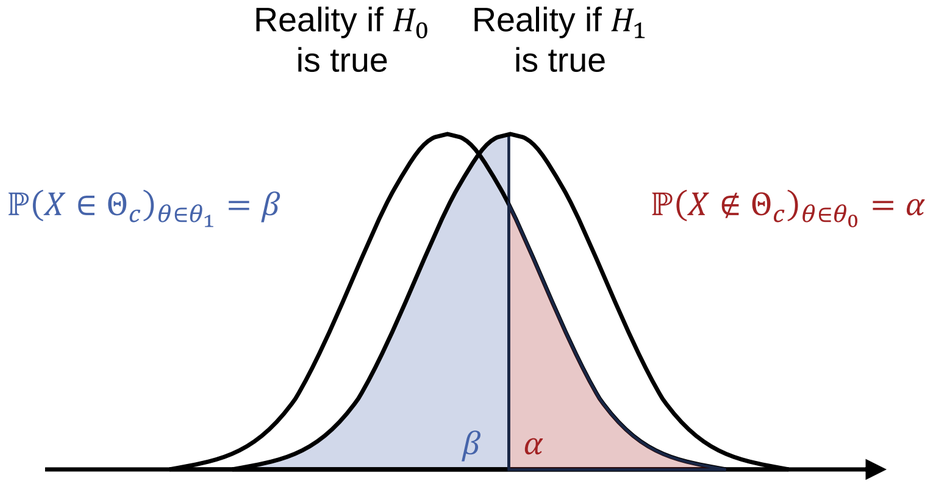
(Unknown) Reality

Decision	H_0 is true	H_0 is false
Reject H_0	α	$1 - \beta$
Accept H_0	$1 - \alpha$	β

Statistical Tests

Errors

- Error of first kind: Discard H_0 although it is true (quantified by α, e.g., 0.05)
  (We believe to be more speed though we aren't)
- Error of second kind: Accept H_0 although it is false (quantified by β, e.g., 0.2)
  (We believe to be less speed though we aren't)
- $1 - \beta$ is called (statistical) power
  - The more similar the distributions under H_0 and H_1 are, the harder it beomces to balance α and β
  - The smaller the variance is, the better the power

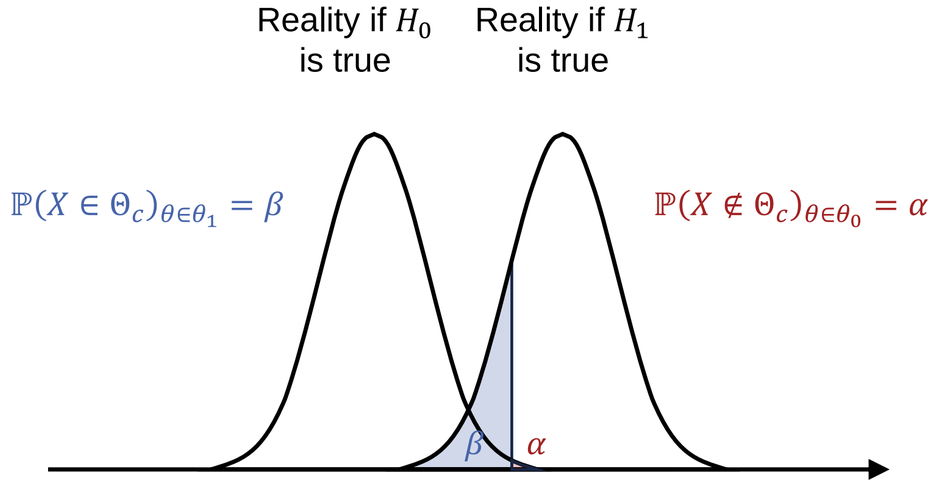


Decision	(Unknown) Reality: H_0 is true	H_0 is false
Reject H_0	α	$1 - \beta$
Accept H_0	$1 - \alpha$	β

Statistical Tests

Errors

- Error of first kind: Discard H_0 although it is true (quantified by α, e.g., 0.05)
  (We believe to be more speed though we aren't)
- Error of second kind: Accept H_0 although it is false (quantified by β, e.g., 0.2)
  (We believe to be less speed though we aren't)
- $1 - \beta$ is called (statistical) power
  - The more similar the distributions under H_0 and H_1 are, the harder it beomces to balance α and β
  - The smaller the variance is, the better the power



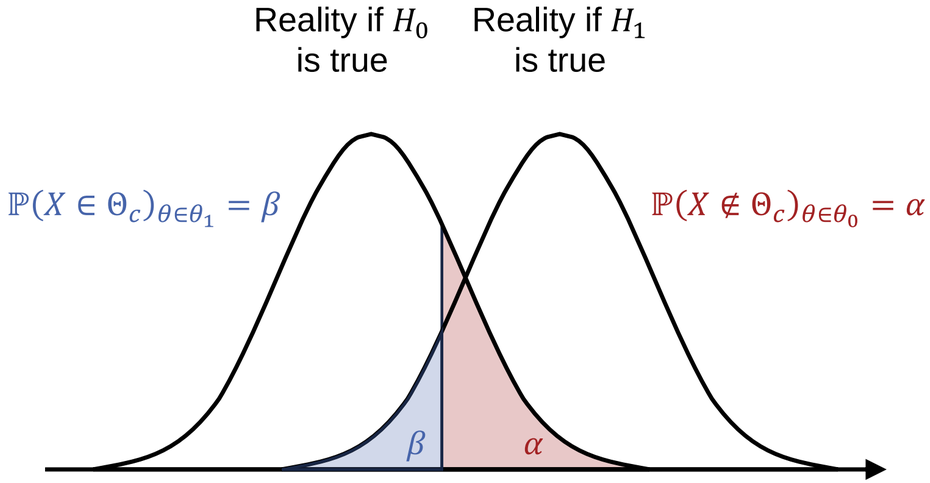
(Unknown) Reality

Decision		H_0 is true	H_0 is false
	Reject H_0	α	$1 - \beta$
	Accept H_0	$1 - \alpha$	β

Statistical Tests

Errors

- Error of first kind: Discard H_0 although it is true (quantified by α, e.g., 0.05)
  (We believe to be more speed though we aren't)
- Error of second kind: Accept H_0 although it is false (quantified by β, e.g., 0.2)
  (We believe to be less speed though we aren't)
- $1 - \beta$ is called (statistical) power
- Power is related to effect size
  - Degree to which a phenomenon is present/ detectable
  - (Normalized) difference between means
  - The smaller the effect size, the smaller the power



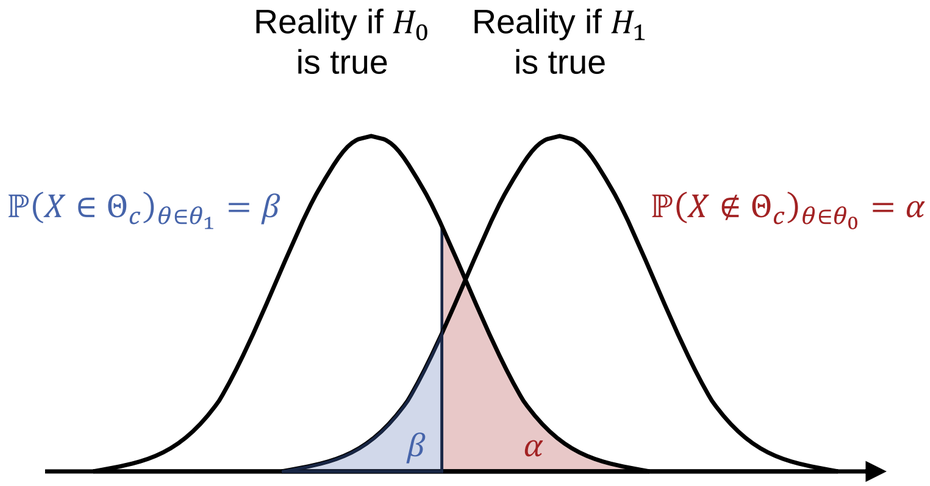
(Unknown) Reality

Decision	H_0 is true	H_0 is false
Reject H_0	α	$1 - \beta$
Accept H_0	$1 - \alpha$	β

Statistical Tests

Power Analysis

- Four fundamental parameters
  - Error of first kind: α
  - Power $1 - \beta$
  - Effect size
  - Sample size n



(Unknown) Reality

Decision	H_0 is true	H_0 is false
Reject H_0	α	$1 - \beta$
Accept H_0	$1 - \alpha$	β

Statistical Tests

Rules of Thumb

- Sample variance can be considered to be a good estimate if $n > 30$ and medium ES
- For small ES, aim for $n > 150$
- If H_0 is rejected, you don't need to consider statistical power
- For normal distribution: differences of means are significant if $(\mu_1 - \mu_2) > 2\sigma$

Values of Thumb

- $\alpha = 0.05$ (Statistics war! ↑)
- $\beta = 0.02$
- Effect size (ES)
  - 0.2 ~ differences in size between 15 year old and 16 year old younglings
  - 0.5 ~ differences in size between 14 year old and 18 year old younglings
  - 0.8 ~ differences in size between 13 year old and 18 year old younglings

Statistical Tests

Unpaired Observations

- E.g., between two independent implementations of the same algorithm
- Assume sample means μ_1, μ_2 of both measurements are normally distributed
- Thus is their difference $\mu_1 - \mu_2$ ($\Rightarrow$ difference of means)

Differences of means are meaningful!

Paired Observations

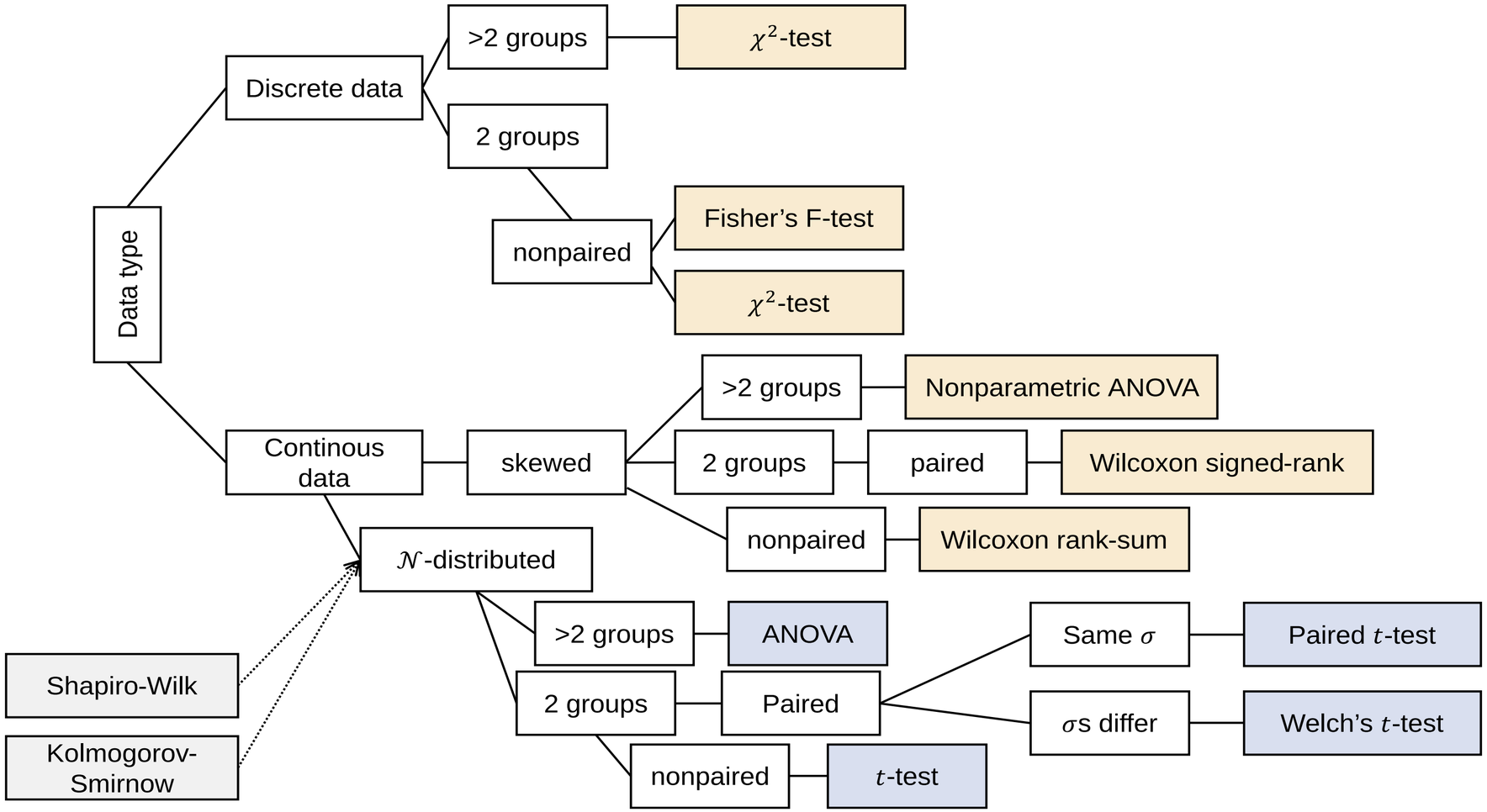
- Before-and-after comparisons, e.g., after a performance improvement
- Can no longer assume independence!
- Group measurements into pairs (b_i, a_i) (b: before, a: after) and consider differences $d_i = b_i - a_i$
- Confidence intervals can now be constructed by considering the mean of differences (under normal assumptions)

Means of differences are meaningful!

Common Statistical Tests (Service Slide)

Test	Use Case
One sample t-test	To tell whether a mean is smaller or larger than a given value
Wilcoxon rank sum test	Alternative for t-test, if N-assumption not justified (ordinal scale)
Paired t-test	Paired-difference test to tell whether means of two populations are the same (assuming $\sigma_1^2 = \sigma_2^2 = \sigma$)
Welch's t-test	Paired t-test if variances differ
Wilcoxon signed-rank test	Similar to paired t-test, if populations means are not of interest or N-assumption not justified (paired-difference test)
Mann-Whitney U test	Same as Wilcoxon rank sum test
Pearson correlation	To assess linear relationships between variables
Spearman correlation	To assess monotonic relationships between variables
Fishers F-test	To test whether random variables are independent (good for small populations)
χ-squared test	Tests that use χ^2-distributed test statistic (for independence, goodness of fit,...)
Shapiro-Wilk test	To test whether a population is N-distributed (better for smaller populations)
Kolmogorov-Smirnow test	To test whether a population is N-distributed

Common Statistical Tests (Service Slide)

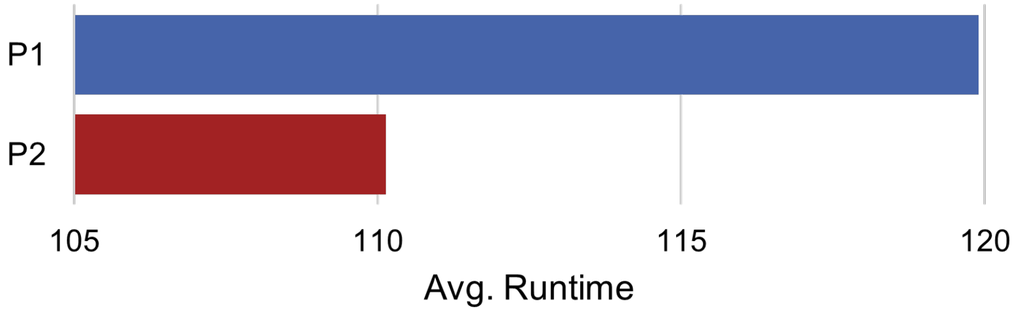


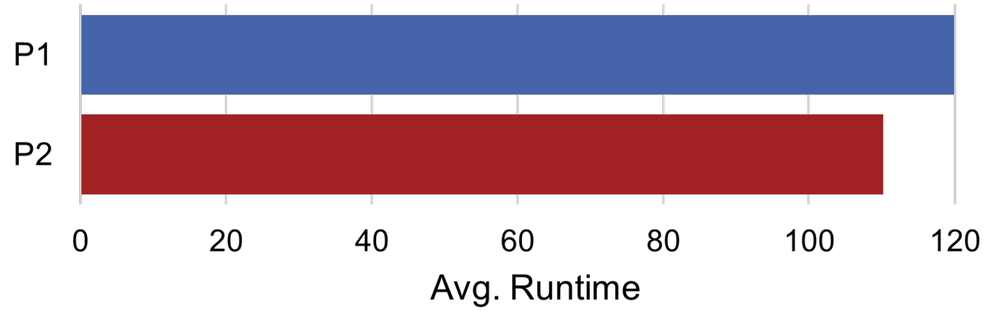
Statistical Tests

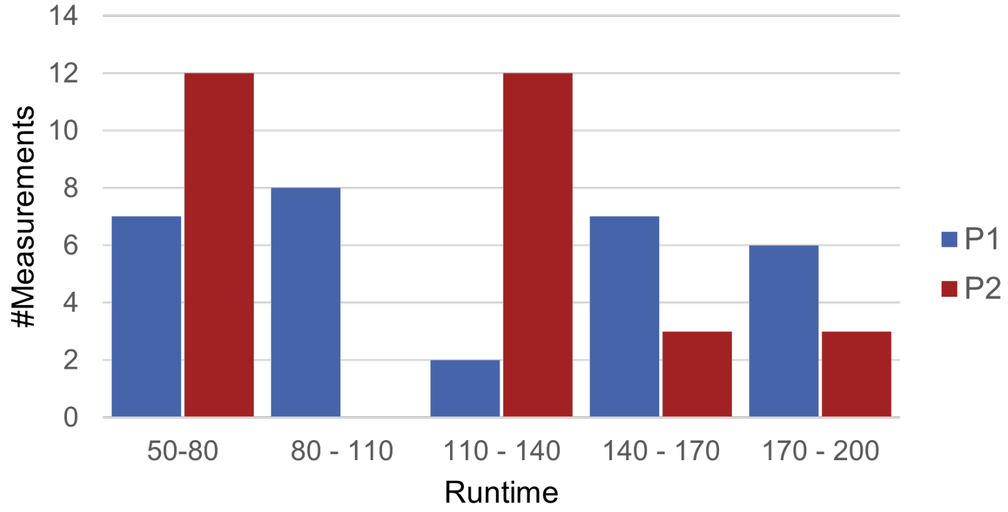
Bonferroni Correction

- Data is like underwear - you should wash it between use...
- Performing multiple tests on the same set of data, the probability for an error of first kind increases
  - Some test will by definition of significance falsely tell you to reject H_0
  - Looks statistically significant but isn't!
  - Leads to false conclusions...
  - ...and happens frustratingly often in peer-reviewed and high-ranked publications
- Solution: for N tests on the same data set, set $\alpha = \frac{\alpha_{target}}{N}$

Statistical Tests

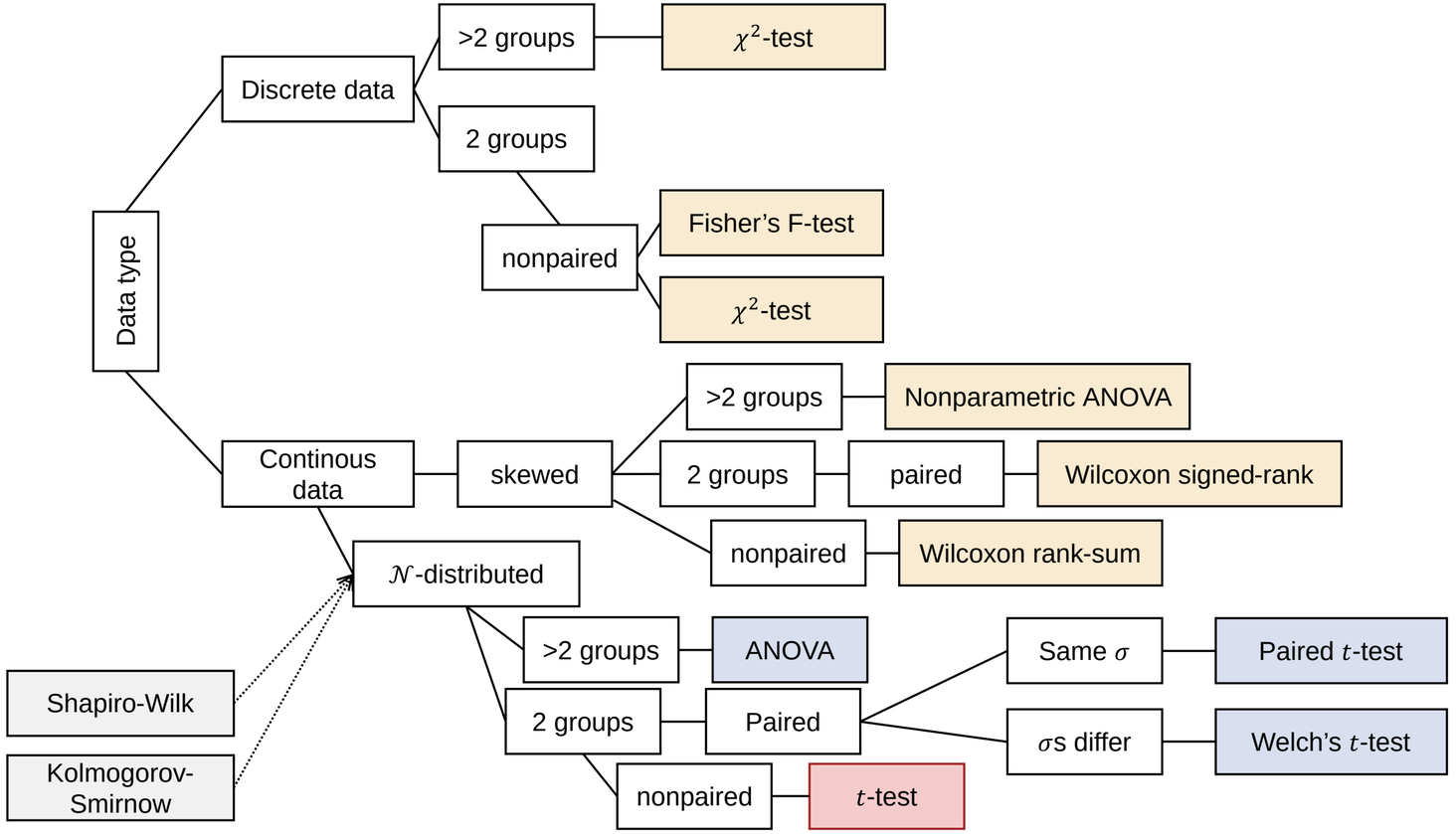






	P1	**P2**
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

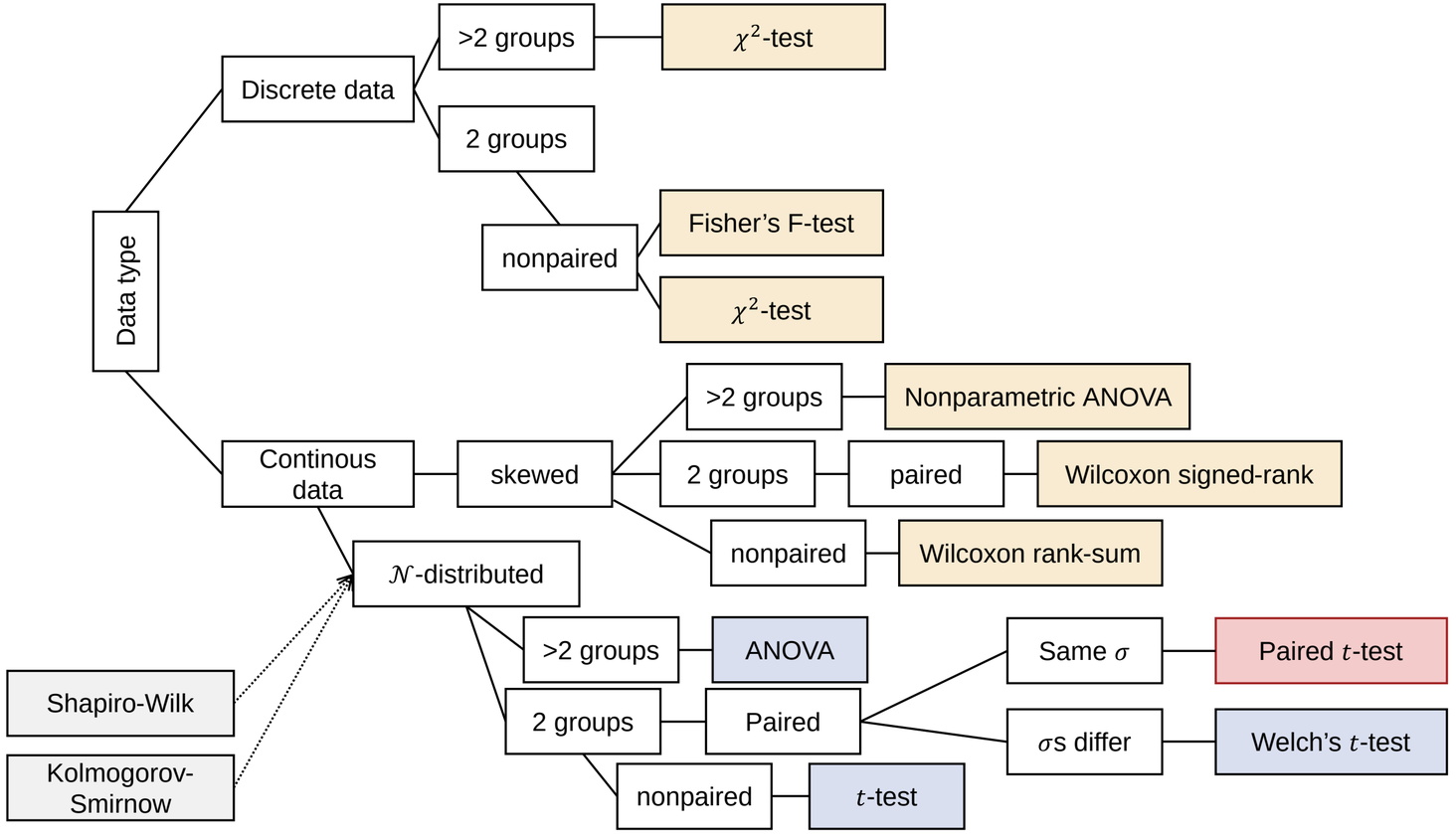
Statistical Tests



	P1	P2
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

p-value: 0.39 > 0.05

Statistical Tests

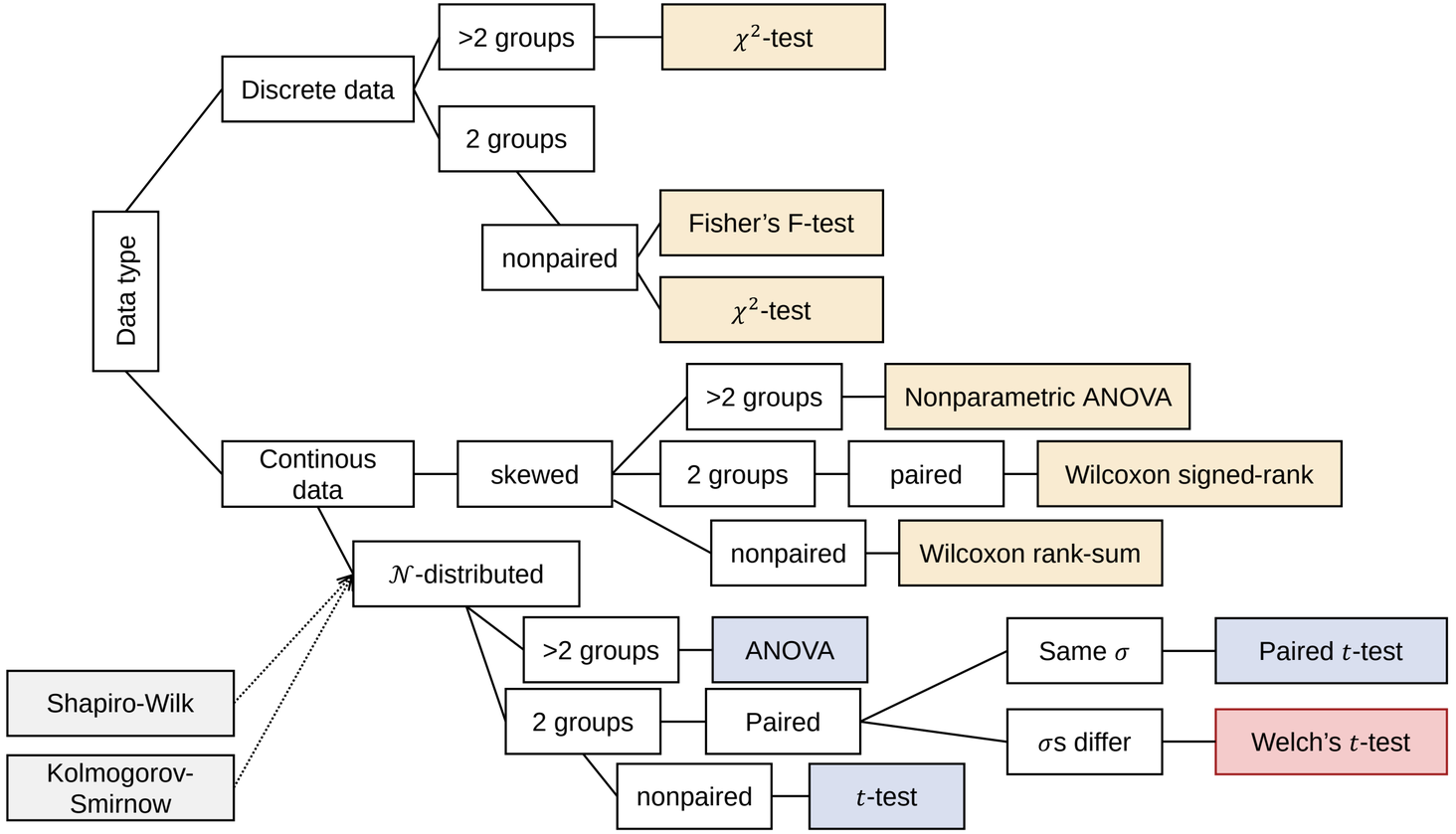


	P1	P2
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

p-value: 0.0042 < 0.05

Requires pairing!

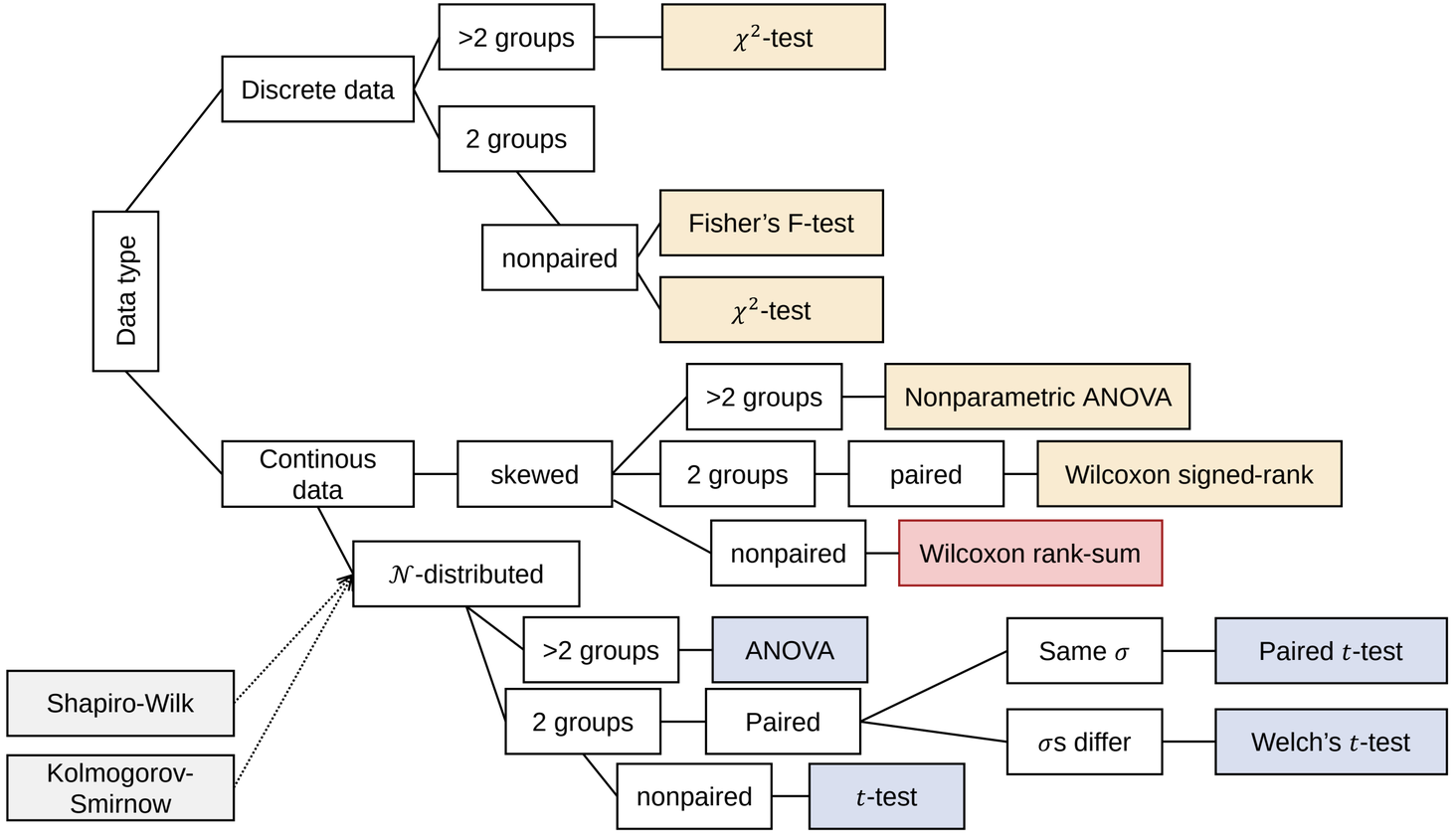
Statistical Tests

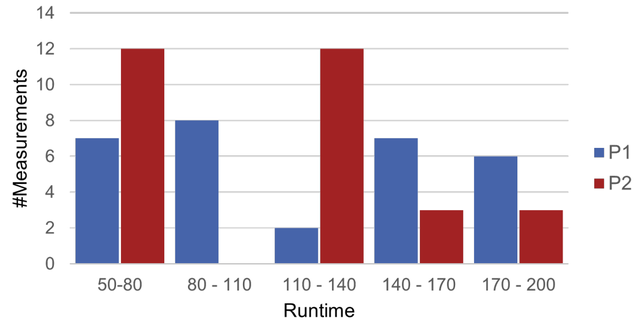


	P1	P2
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

p-value: 0.39 > 0.05

Statistical Tests

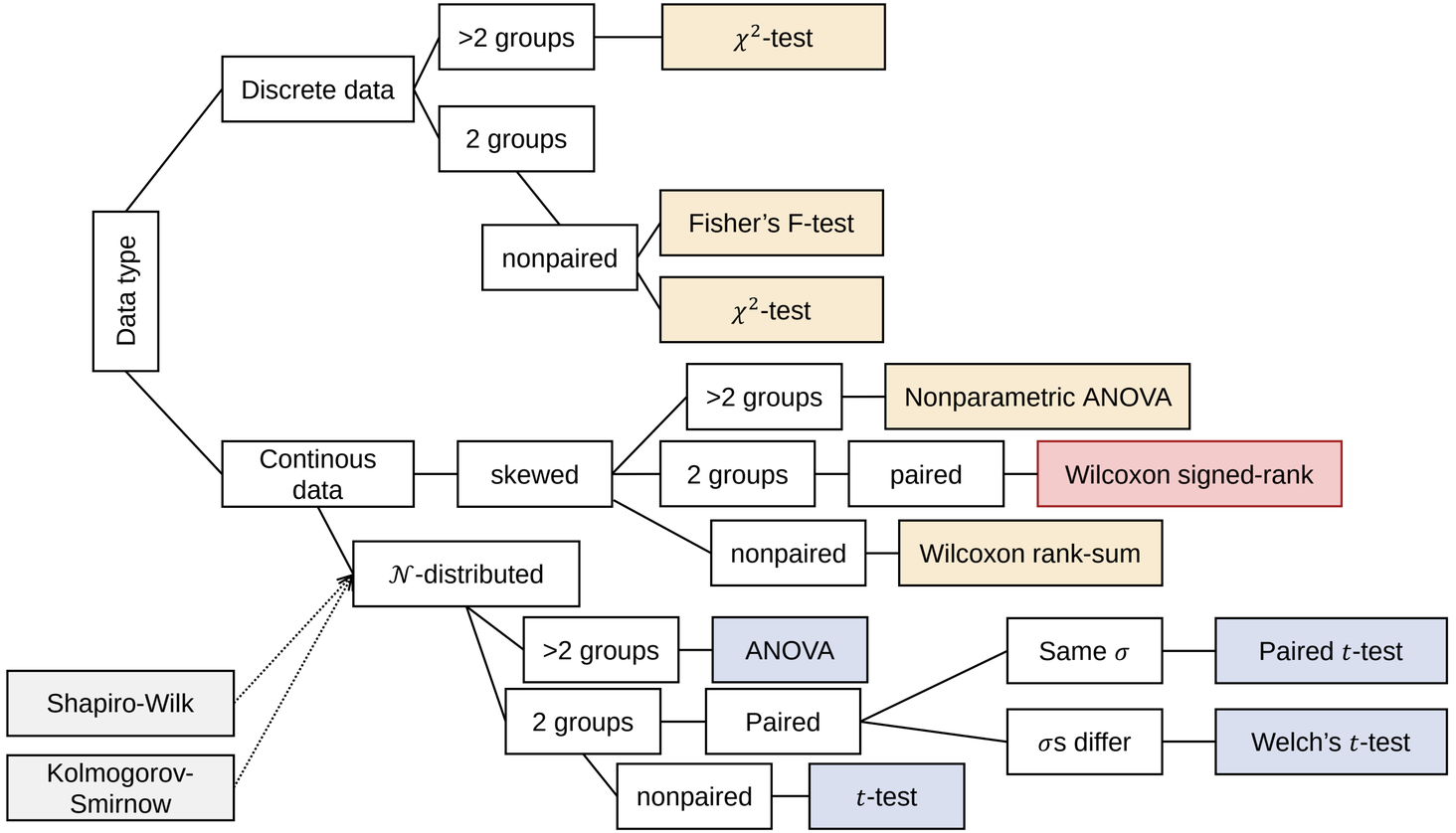


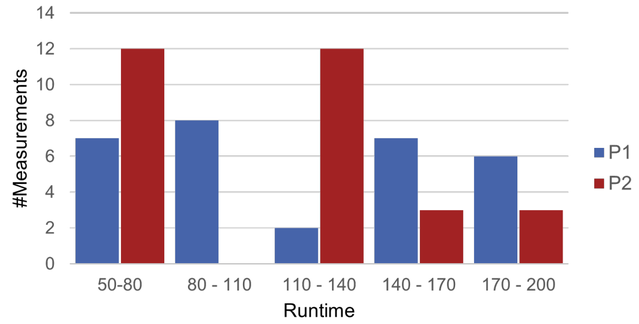


p-value: 0.49 > 0.05

Does the histogram look normal?

Statistical Tests





p-value: 0.02 < 0.05

Finally! Unquestionable significance!

How many tests did we do already...?

p-value: 0.02 > 0.05/4 (Bonferroni)

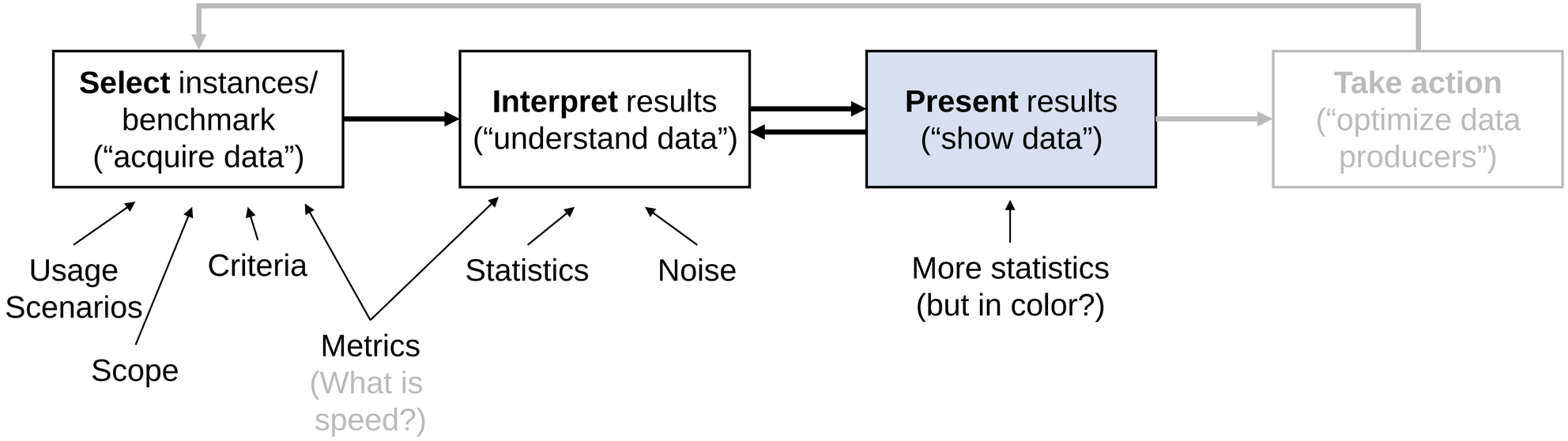
On Statistical Tests

- (Un-/Fortunately), this is not a lecture on probability theory and statistics
- Important results: Central Limit Theorem (if we run benchmark(s) often enough and they are not too odd, it is safe to assume that they are normally distributed)
- Statistical tests may help to argue about the significance of improvements on a more scientific/ academic appearing level
  - May not provide clear answers
  - With large enough n, (practically) neglectable performance differences can appear to be "significant"
  - Conclusion?

Presenting Results

Presenting Results

"A benchmark is a tool coupled with a methodology for the evaluation and comparison of systems or components (system under test, SUT) in defined usage scenarios (workload) with respect to specific characteristics (metrics), e.g., performance."



Presenting Results

Basic Principles

- Form follows function. Function: clearly demonstrate performance results
- Be as objective as possible
- Measurements should be reproducable
- Avoid overwhelming complexity (human brain can process around 4-5 chunks of information simultaneously)
- Make conclusions evident

Presenting Results

Tables ("1D plots")

- 👍 Easy
- 👍 Accurate, objective
- 👍 Compact
- 👍 Good for unrelated instances

- 👎 Overused
- 👎 Boring
- 👎 Hard to process visually

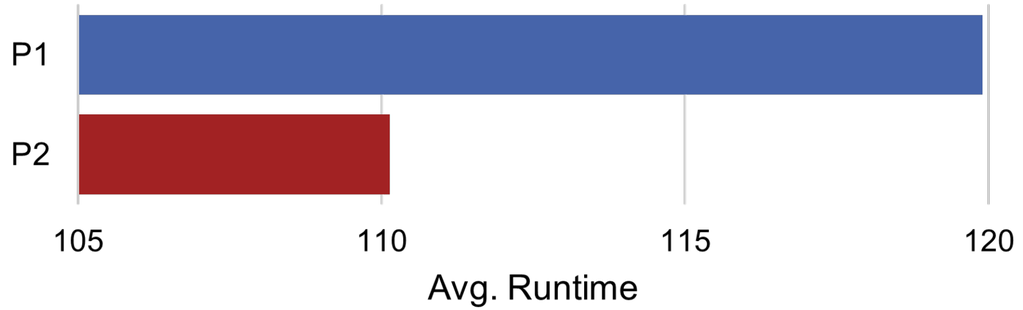
	P1	P2
min	51	55
max	195	198
$\bar{x}$ (average)	119.9	110.1
$m_{1/2}$ (median)	108	122
m^* (mode)	104	113
s	46.3	41.8

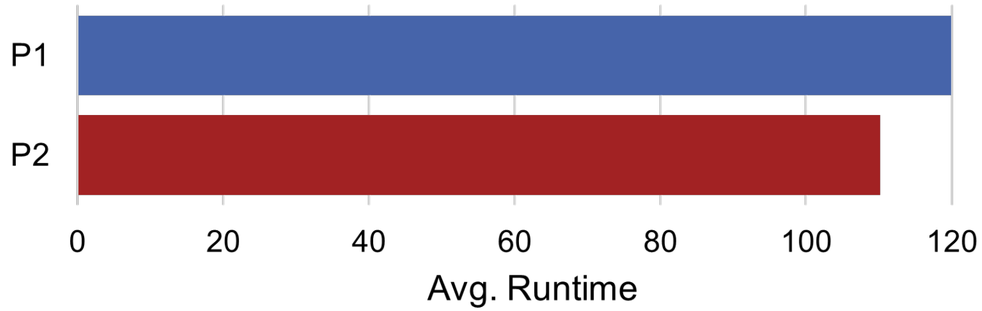
Rule of thumb: outperform graphs for small sets of around 20 numbers or less

Presenting Results

2D Plots

- E.g., problem size vs. runtime
- Choose appropriate scaling (linear, logarithmic)
- Avoid log scale for y axis
- Scale such that horizontal/ vertical/ diagonal lines are meaningful
- Start from 0 if possible, be fair





👍 High information density
👍 Faster (visual) processing

👎 Easily overdone (too many curves)
👎 Can become (unintentionally) subjective

Presenting Results

3D plots

👍 Interactive contexts

👎 Hard to read off absolute values

👎 Information loss (projection)

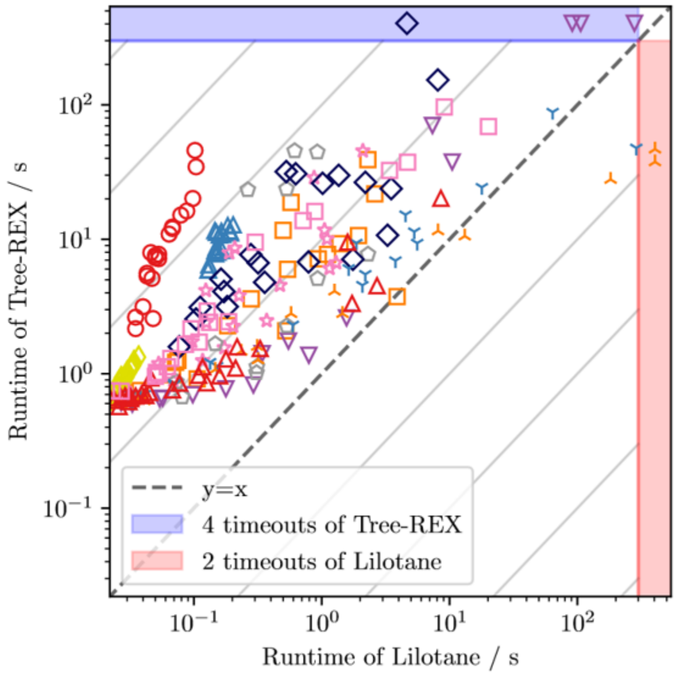
7 Deadly Sins

1. Forget explaining axes
2. Connecting unrelated points by lines
3. Mindless double-log plotting
4. Cryptic abbreviations
5. Small lettering
6. Excessive complexity
7. Pie charts

Presenting Results

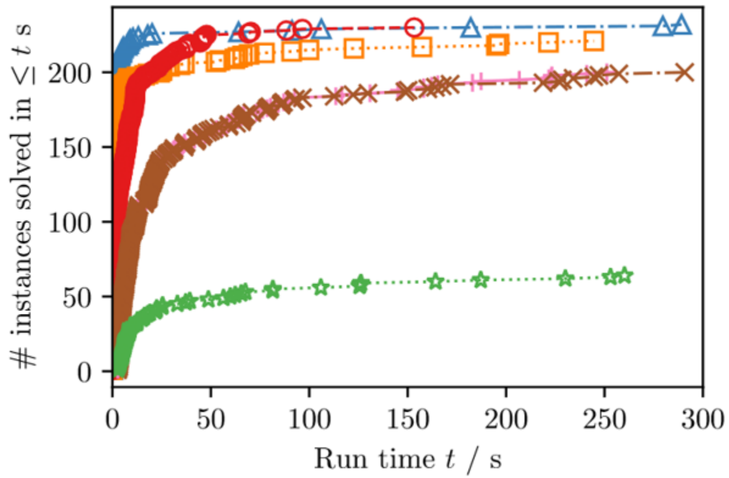
Arranging Instances

- Bar charts
- Scatter plots
- Different plots for different classes of instances



Benchmarking hard to solve instances (NP-hard or worse)

- Cumulative density plots ("Cactus plots")



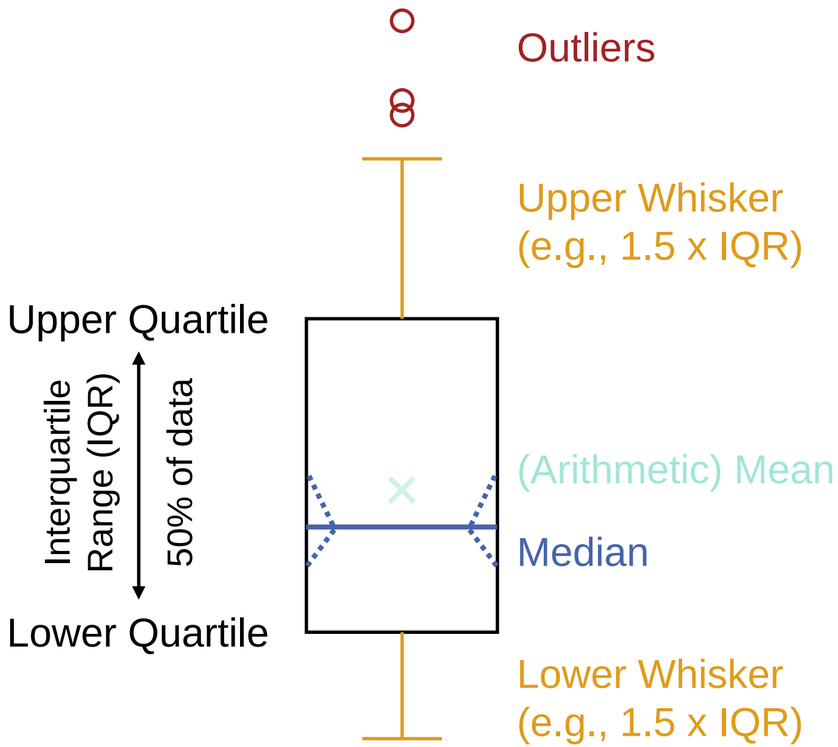
Presenting Results

Presenting Distirbutiong

- Average, median, variance
- Box (Whisker) Plots
  - Lenght of whiskery varies
  - Arithmetic mean is no robust metric
  - Notched box-plots show confidence interval for median

👍 Provide high-level inforation
👍 See where bulk of data is
👍 Spot outliers

👎 Simplicity limits information density (e.g., shape of distribution)



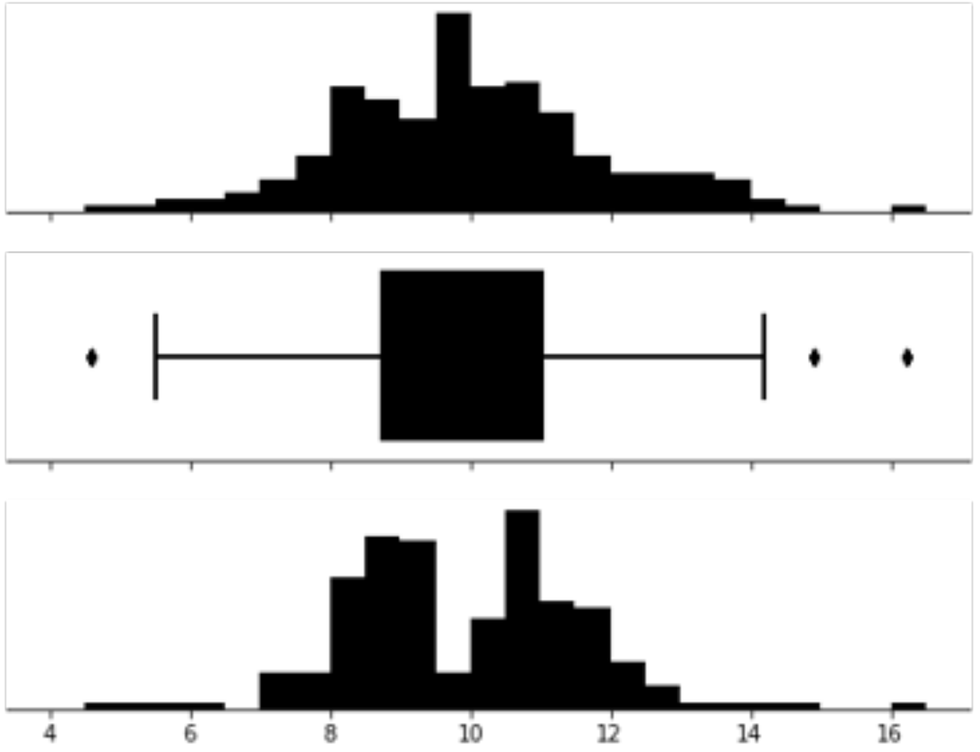
Presenting Results

Presenting Distirbutiong

- Average, median, variance
- Box (Whisker) Plots
  - Lenght of whiskery varies
  - Arithmetic mean is no robust metric
  - Notched box-plots show confidence interval for median

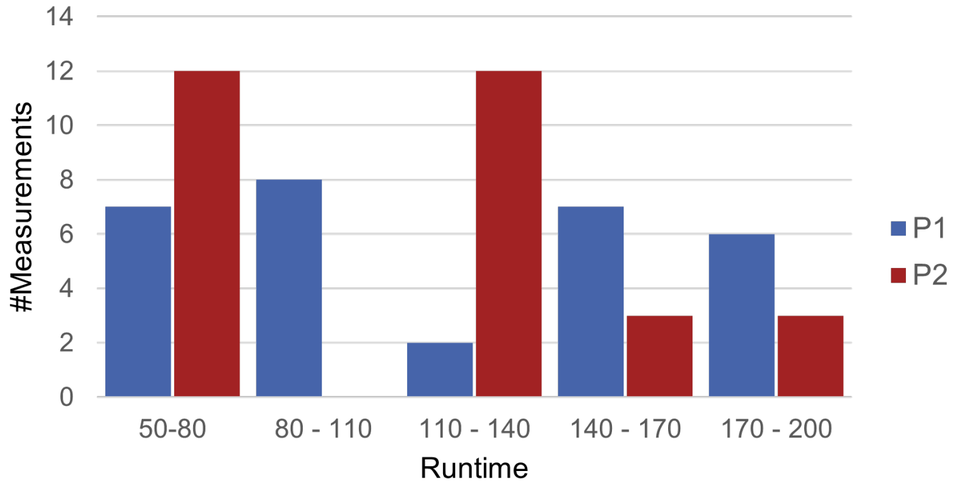
👍 Provide high-level inforation
👍 See where bulk of data is
👍 Spot outliers

👎 Simplicity limits information density (e.g., shape of distribution)



Presenting Results

- Histograms
  - 👍 Spot peaks of the distributions
  - 👍 Tell if it is skewed, symmetric, ...
  - 👍 Spot outliers
  - 👍 Explore large data sets
  - 👎 Hard to draw conclusions about relationships of multiple variables
  - 👎 Bin sizing



Benchmarking Crimes

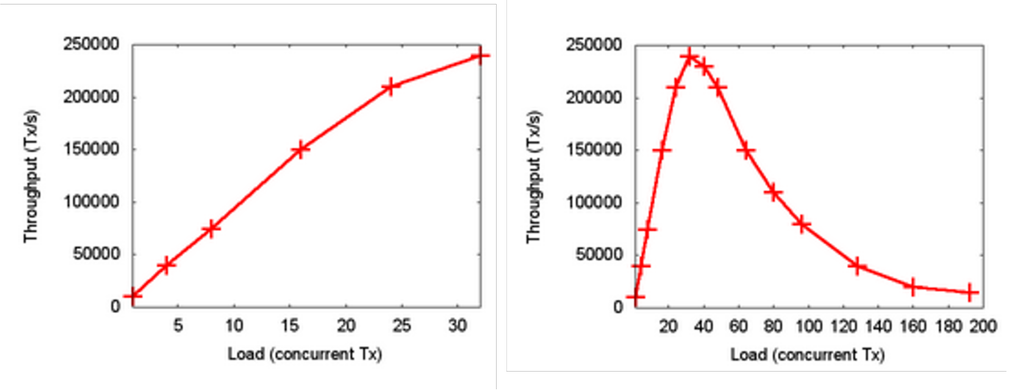
Benchmarking Crimes

Selective Benchmarking

- Using a biased set (e.g., a sub-set) of benchmarks to "prove" a point
  - When sub-setting, justify your choices!
- A fair evaluation must demonstrate two things:
  1. Performance does improve (significantly) in area of interest (progressive)
  2. Performance does not degrade elsewhere (conservative)

Selective Data/ Instances

- Make system look good but range/ parameter selection is not representative on real workloads



- *Shown: troughput of a database system; Left: linear scaling as long as there is at most one client per core*

Benchmarking Crimes

Improper Result Handling

- Microbenchmarks probe particular aspects
  - May suffice if this aspect is critical
- Macro-/ Application benchmarks provide more realistic pictures of overall performance
- Don't confuse percentage with percentage points!
  - E.g., CPU utilization: 26% to 46% is nearly doubling it, not increasing it by 20%
- Don't report raw averages without variance (to indicate significance)
- Consider geometric mean instead of arithmetic mean

Using the Wrong Benchmark

- E.g., uniprocessor benchmark for multiprocessor scalability
- Simluated/ synthetic instances contain assumptions not necessarily present in real world
- Using same dataset for calibration, optimization and validation

Benchmarking Crimes

Improper Comparison of Results

- No proper baseline (instead: state-of-the-art solution, virtually best solution, ...)
- Evaluation against yourself only
- Unfair benchmarking of competitor *(it's easy to run someone else's system sub-optimally...)*

Improper Use of Statistics

- Same set of data for multiple tests (Bonferroni correction!)
- Too little runs
- Assuming normal distribution where it is not the case *(You don't need to run a statistical test, but eyeball your data at least)*

Missing Information

- Missing specification of the evaluation platform (hardware and software; reproducibility!)
- Reporting relative numbers only
- Exercise: Count benchmarking crimes in "https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4784874"

Benchmarking Crimes

Best Practices

- Document what you are doing (assumptions, choices, results), try to explain what you are doing
- Do several runs
- Check standard deviation, especially for abnormal variance
- Do statistical tests (e.g., outlier-detection, ...) only when necessary

Test Data and Results

- Verify your data *(at least after one complete run)*
- Never reuse the same data
- If it is hard to get new (good) test data, consider randomizing what you have
- Use lots of iterations to improve statistics and remove clock granularity
- Perform sanity checks

Thank You!