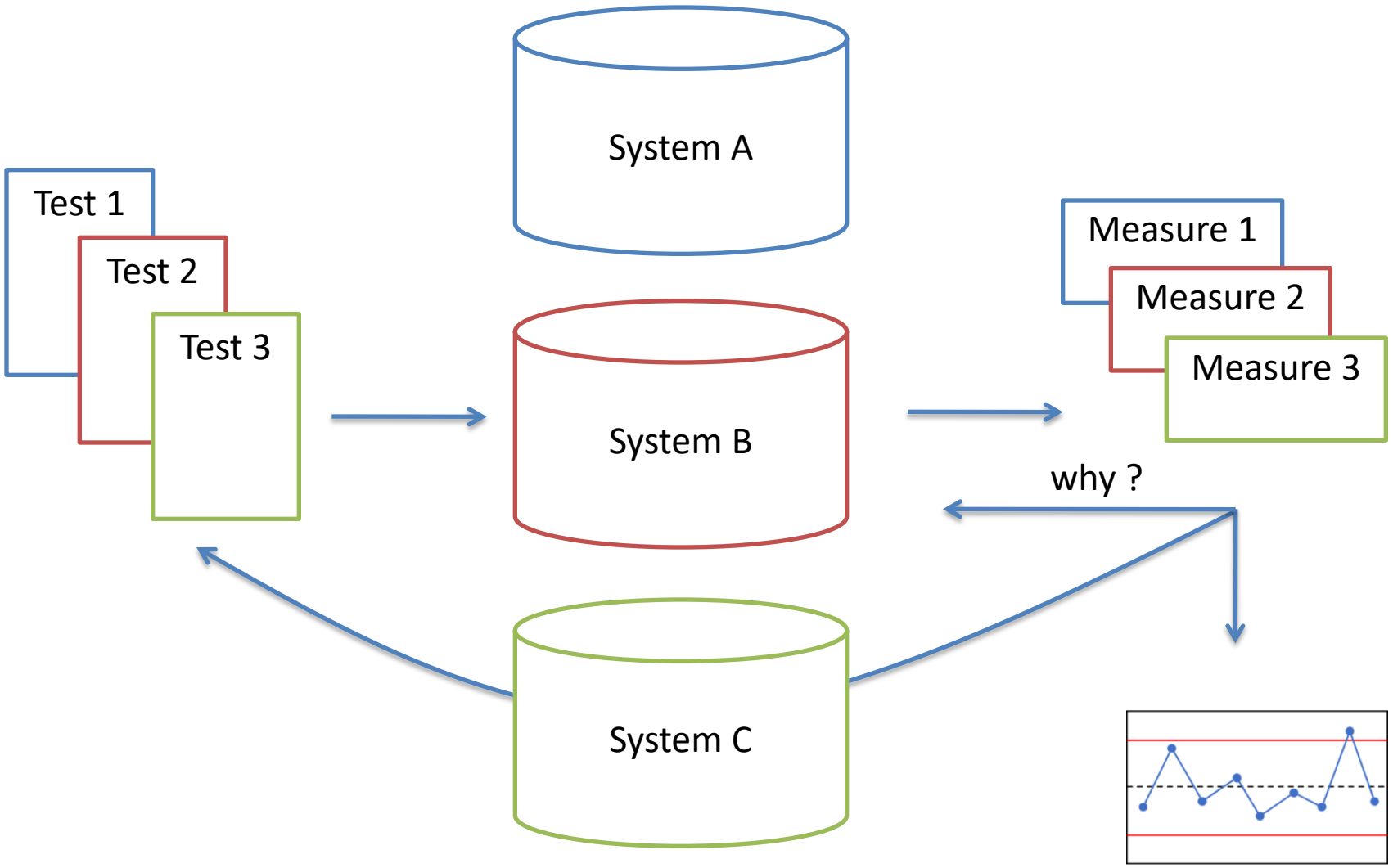
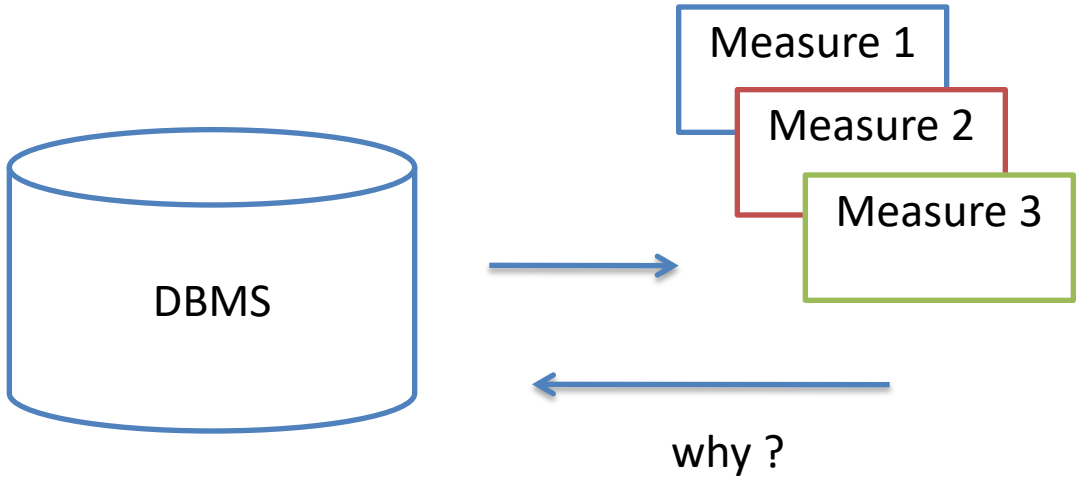
Performance Evaluation in Databases

Federico Ulliana

Slides collected from Ioana Manolescu, Stefan Manegold, Gunes Aluc, John Mellor-Crummey and Fabian Suchanek


System A
Test 1
Test 2
Test 3
System B
Measure 1
Measure 2
Measure 3
why ?
System C

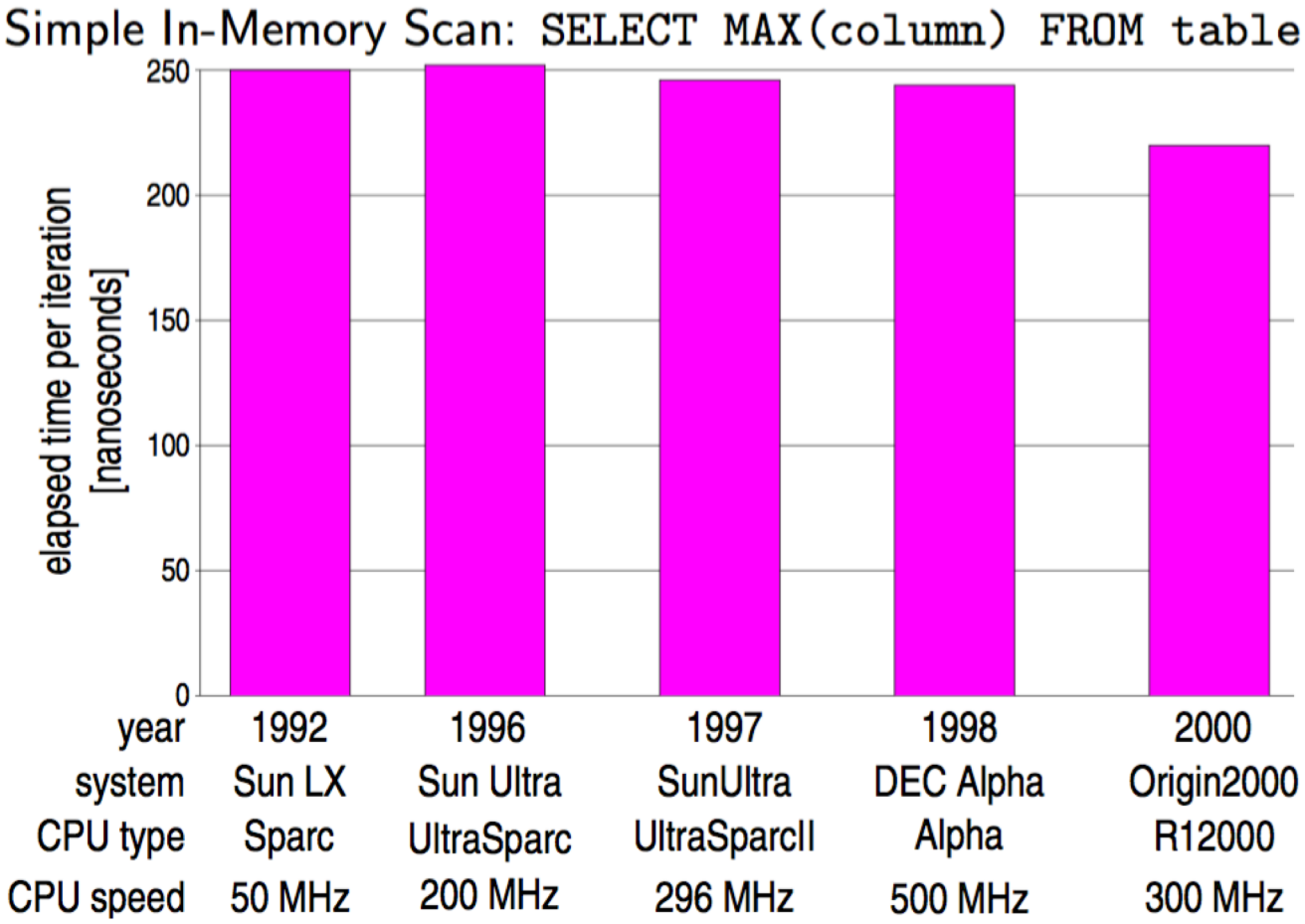
ANALYZING THE DATA CAN BE DIFFICULT, EITHER


Measure 1
Measure 2
Measure 3
DBMS
why ?

Understanding Experimental Results

- Experiments always bring surpring results
- Leads almost always to updating/upgrading your code
- Sometimes difficult to understand the cause of bad performances
- Need to use profilers to understand that

Do you know what happens?

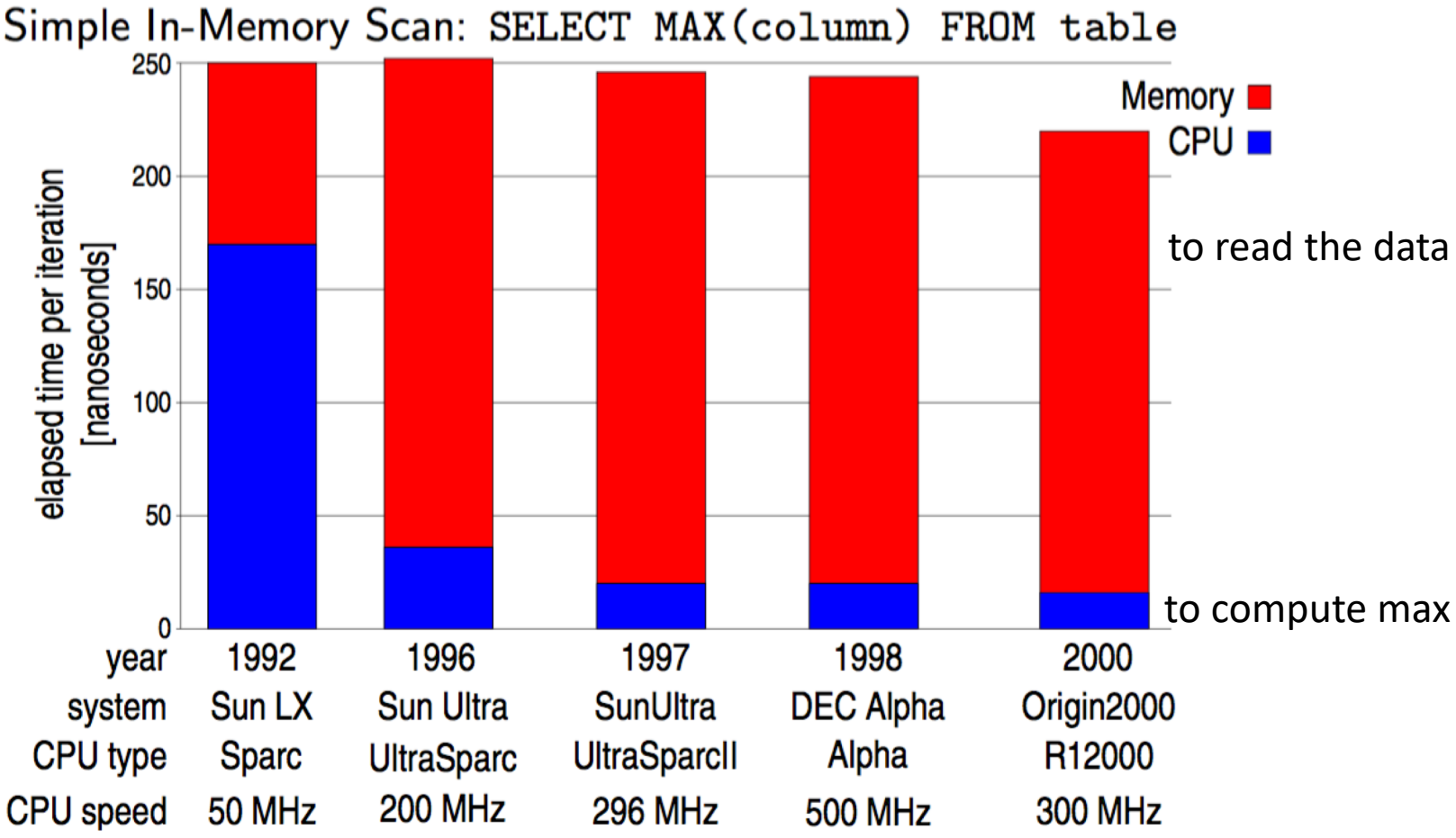


iteration = readinga tuple and compute max

Do you know what happens?

- Simple **In-Memory** Scan (No disk-I/O)
  - SELECT MAX(column) FROM table
- Up to 10x improvement in CPU clock-speed
  ⇒ *Yet hardly any performance improvement !??*
- Standard profiling (e.g., ‘gcc -gp‘ + ‘gprof‘) does not reveal more (in this case)
- Need to dissect CPU & memory access costs
  - Use hardware performance counters to analyze cache-hits, -misses & memory accesses
    - VTune, oprofile, perfctr, perfmon2, PAPI, PCL, etc.

Find out what happens !



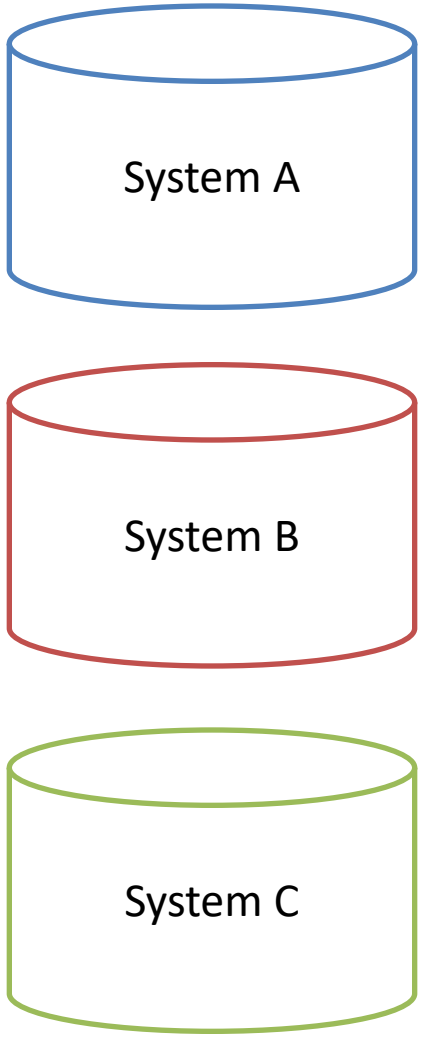
Find out what happens !

[Source : Understanding, Modeling, and Improving Main-Memory Database Performance]

- *“[…] Consequently, while our experiment was still largely CPU-bound on the Sun from 1992, it is dominated by **memory access costs** on the modern machines.*

  This can be attributed to the complex memory subsystem that comes with SMP architectures, resulting in a high memory latency […]”

COMPARATIVE ANALYSIS


System A
System B
System C

Comparing two systems A and B

- Systems should be run in almost identical conditions
  - Hardware : generally feasible, although some conditions (eg., limited or powerful machines) may favor one or the other (cannot run a server tool on an old laptop)
  - Software conditions : generally impossible
    - If system have different type of tunings, it may be very hard to find out which are the “closest” configurations for A and B
    - Systems may also have differnt type of exports / inports etc.. These issues should be adressed before starting the experiments
- Several configurations of a system may be run to better understand its behaviour.
- But in the project you have very similar systems ☺

Apples and Oranges (Comparative Analysis)

- Two colleagues A & B (at CWI) each implemented one version of an algorithm for Monet-DB, A the “old” version and B the improved “new” version
- They ran identical experiments on identical machines, each for his code.
- Though both agreed that B’s new code should be significantly better, results were consistently worse.
- They tested, profiled, analyzed, argued, wondered, fought for several days ...
- ... and eventually found out that A had compiled with C++ optimization enabled, while B had not .

Apples and Oranges

- A) `configure --enable-debug --enable-optimize --enable-assert CFLAGS = "-g [-O0]"`

- B) `configure --disable-debug --disable-optimize --disable-assert CFLAGS = " -O6 -fomit-frame-pointer -finline-functions -malign-loops=4 -malign-jumps=4 -malign-functions=4 -fexpensive-optimizations -funroll-all-loops -funroll-loops -frerun-cse-after-loop -frerun-loop-opt -DNDEBUG "`

Apples and Oranges
(Comparative Analysis)

- Compiler optimization ⇒ up to factor 2 performance difference
- DBMS configuration and tuning ⇒ factor x performance difference ($2 \leq x \leq 10$?)
- Default settings often too “conservative”
- Do you know all systems you use/compare equally well?

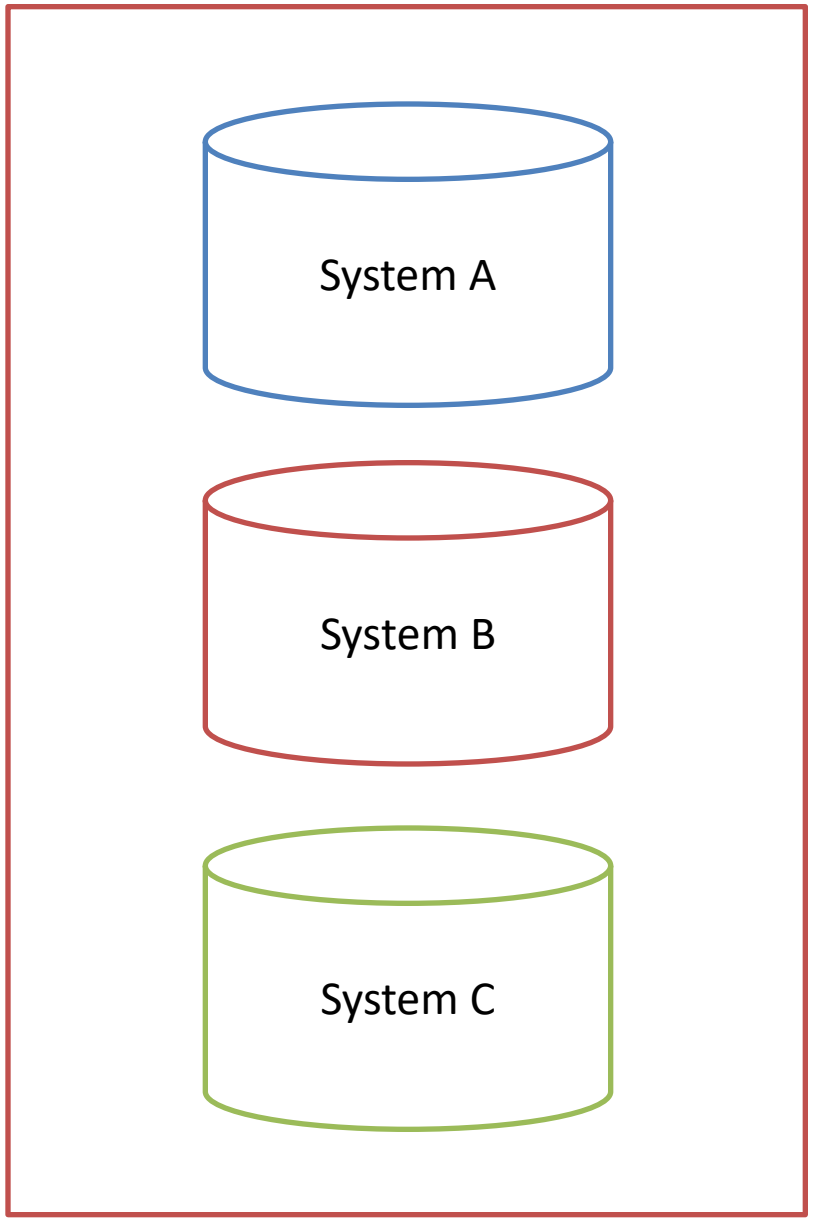
Is there any correct formulation ?

- "Absolutely fair" comparison of complex systems virtually impossible

- But, be at least aware of the the crucial factors and their impact, and document accurately and completely what you do.

*The joke : "Our problem-specific, hand-tuned, prototype **X** outperforms an out-of-the-box installation of a full-fledged off-the-shelf system **Y**, in particular when omitting query parsing, translation, optimization and result printing in **X** , while including them in **Y**"*

CHOOSING HARDWARE AND SOFTWARE


System A
System B
System C

Choosing the hardware

- Choice mainly depends on your problem, knowledge, background, resources, taste, etc.

- Whatever is required by / adequate for your problem
  - A laptop might not be the most suitable, representative database server...

Choosing the Software

- Operative system, DBMS, …
- Commercial
  - Require license
  - “Free” versions with limited capabilities?
  - Limitations on publishing results
  - No access to code
  - Optimizers
  - Analysis & Tuning Tools
- Open source
  - Freely available
  - No limitations on publishing results
  - Access to source code

Proprietary Software

*"Yes, I'll be happy to let you have a time-limited free license of **X** for [research] purposes. (is 12 months adequate?)*

*I only make one condition, which is that you give me the opportunity to comment on any factual statements about **X** before publication. "*

Choosing the software

- Other choices depend on your problem, knowledge, background, taste, etc.

- Operating system
- Programming language
- Compiler
- Scripting languages
- System tools
- Visualization tools

Specifying hardware environments

- We use a machine with 3.4 GHz.”

Underspecified

Specifying hardware environments

cat /proc/cpuinfo

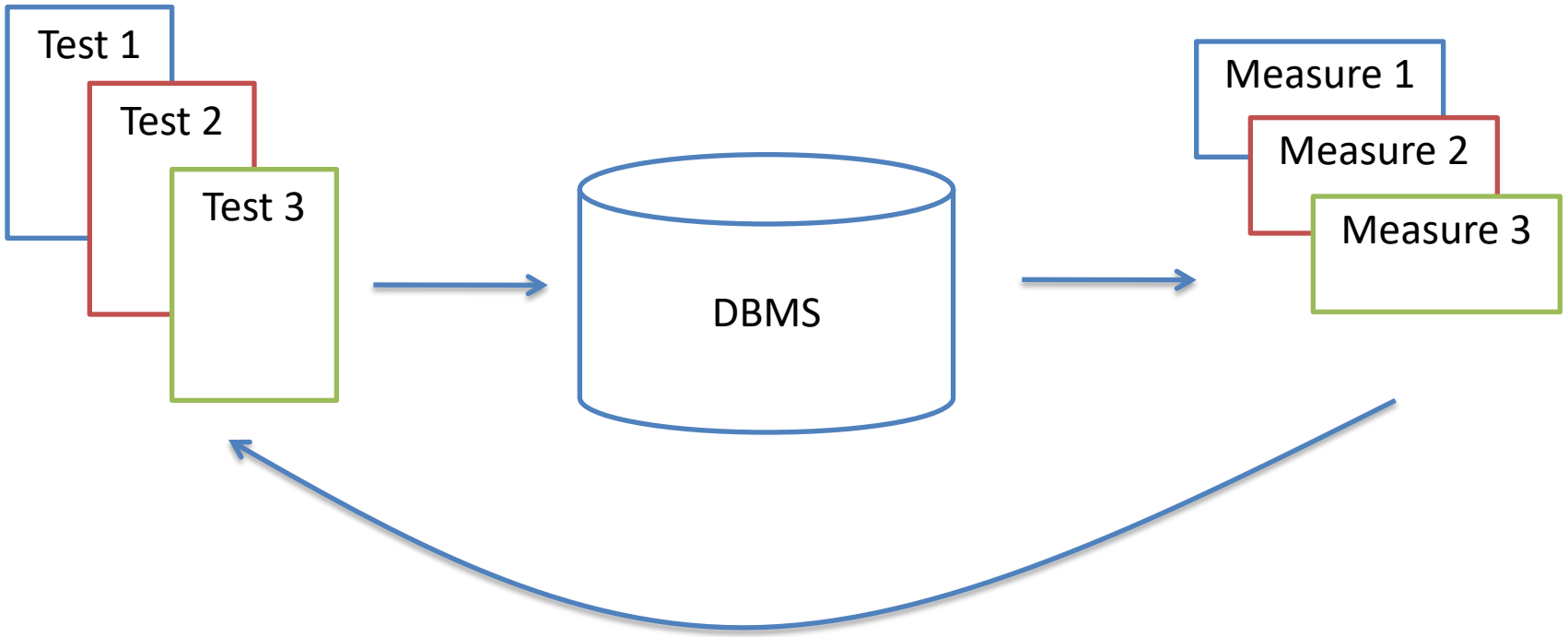
```
processor vendor_id
cpu family model
model name stepping
cpu MHz
cache size fdiv_bug hlt_bug f00f_bug coma_bug
fpu fpu_exception cpuid level wp
flags
bogomips clflush size
: 0
: GenuineIntel
: 6
: 13
: Intel(R) Pentium(R) M processor 1.50GHz : 6
: 600.000
: 2048 KB
: no
: no
: no
: no
: yes
: yes
: 2
```

Overspecified

Specifying hardware environments

- CPU: Vendor, model, generation, clockspeed, cache size(s):
  - 1.5 GHz Pentium M (Dothan), 32 KB L1 cache, 2 MB L2 cache
- Main memory: size
  - 2 GB RAM
- Disk (system): size & speed
  - 120 GB Laptop ATA disk @ 5400 RPM
    1 TB striped RAID-0 system (5x 200 GB S-ATA disk @ 7200 RPM

MAKING EXPERIMENTS REPEATABLE


Test 1
Test 2
Test 3
DBMS
Measure 1
Measure 2
Measure 3

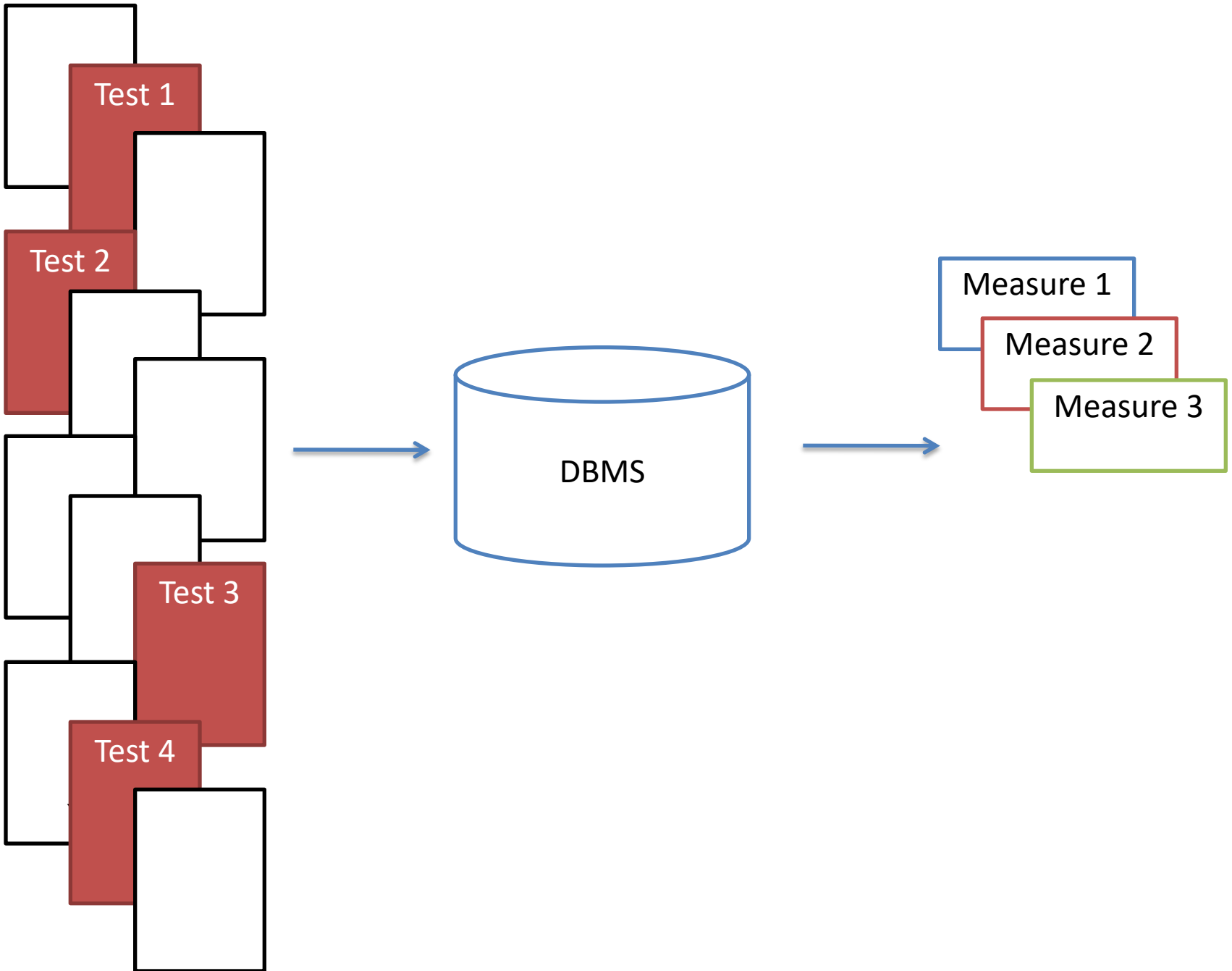
Making experiments repeatable

- Purpose: another human equipped with the appropriate software and hardware can repeat your experiments.
  - Your supervisor / people you supervise
  - Your colleagues
  - Yourself, 3 months later when you have a new idea
  - Yourself, 3 years later when writing a report or answering requests from people

Making experiments repeatable

- Making experiments portable and parameterizable
- Building a test suite and scripts
- Writing instructions

EXPERIMENT DESIGN (FINALLY SOME THEORY)


Test 1
Test 2
Test 3
Test 4
DBMS
Measure 1
Measure 2
Measure 3

Issues with experiment design

- Testing all possible level combinations is unfeasible.
- That there is never enough time to test and analyze all possible system configurations.
  - Example
    (2 CPU types)(3 systems)(4 memory sizes)(2 disk)
    (4 workloads) = 128 distinct experiments
- Ignoring important parameter leads to fragile analysis
- Goal of design : how to maximize information
  and minimize experiments

Terminology first...

- **Factors**: variables with alternatives that affect a measurement
  - e.g. CPU type, memory size, # disk drives, workload used, dataset used, DB used, system version
- **Levels**: values a factor can assume
  - e.g. memory size levels: 512MB, 1GB, 2GB, 4GB

Types of Experimental Designs

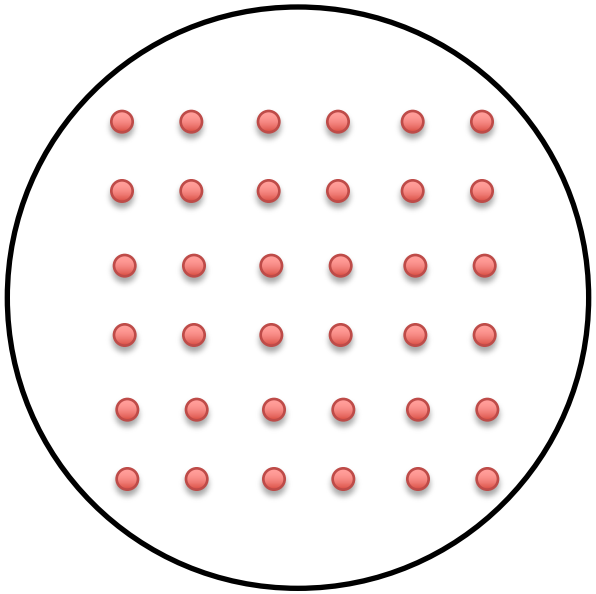
- Full factorial design
- Simple designs
- Fractional design

Full Factorial Design (Unfeasible)

- **Explore every possible combination** at all levels of factors
  - exponential number of experiments

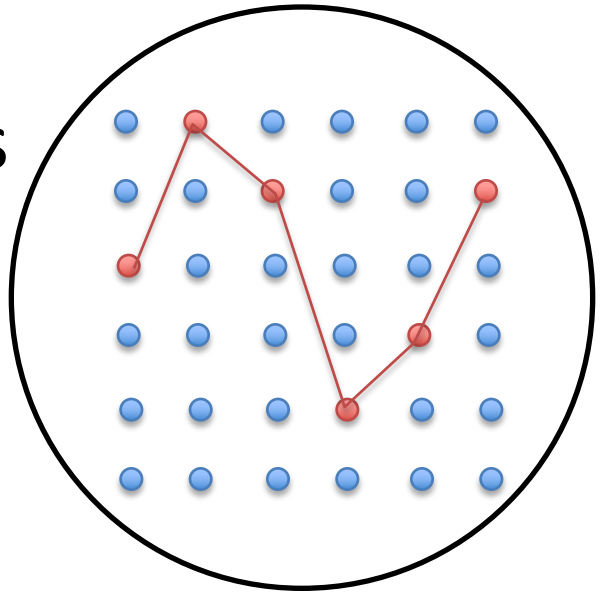
- Advantages
  - every possible configuration is examined

- Disadvantages
  - cost: too many experiments



Simple Design (Not Recommended)

- Start with typical configuration
- **vary one factor at a time** …
- … and see how performances change



Simple Design (Not Recommended)

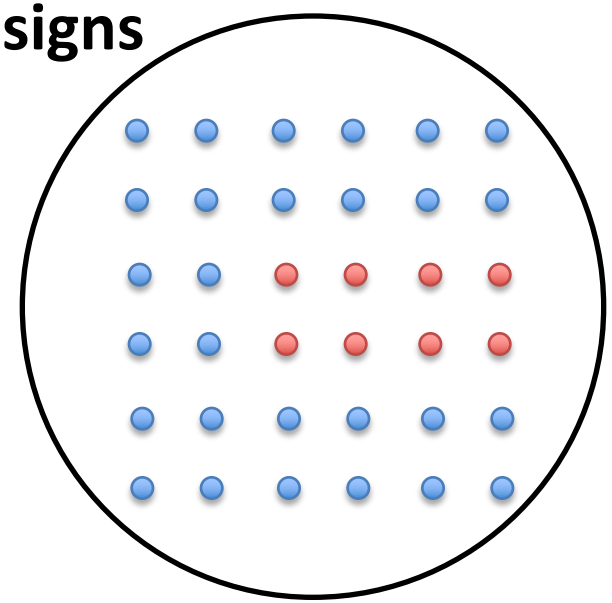
- Example: comparing workstation configurations
  - run typical configuration using a benchmark
  - run experiments to pick the best disk by varying disk only
  - using the best disk, run a set of experiments to find the minimum memory size yielding good performance
  - using the best (disk, memory) configuration, examine the impact of CPU on the benchmark's performance
- Drawbacks
  - if factors interact, may yield wrong conclusions
  - fails to make best use of # experiments

Fractional Design

- *Carefully chosen subset (fraction) of a full factorial design*

1. Reduce number of levels per factor
   - 2 is very popular: **2^k Factorial Designs**

1. Reduce # factors:
   - prune unimportant factors, then try more factors per level



Fractional Design

- (2 CPU types)(4 memory sizes)(2 disk RPMs)(5 workloads) = 80 distinct experiments

- $\mathbf{2^k}$ factorial design
  - (2 CPU types)(2 memory sizes)(2 disk RPMs)(2 workloads) = **16 exp.**

- Can even remove some tests
  - here a $2^{(4-1)}$ design
  - only 8 experiments
    - left ones maybe negligible ?
  - do a smart selection

CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload
CPU	Memory	Disk	Workload

2^k factorial design

- What are they?
  - determine effect of **k factors**, each with **2 levels**

- Why consider them?
  - easy to analyze
  - helps order factors based on impact
  - useful to identify factors that have significant impact
    - ⇒ study with full factorial design
  - factors have little impact
    - ⇒ not of interest for quantitative study

2^k Factorial Designs

- How to select 2 levels
  —factor effect is expected to be unidirectional

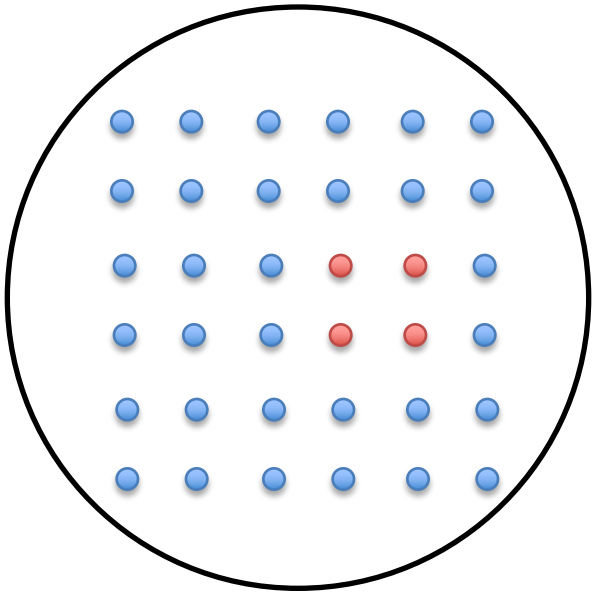
  $\Rightarrow$ select min (low), max (high)

- Performance increases or decreases w/ factor
  - e.g. performance improves with more memory

2^2 Factorial Designs

- Special case of 2^k factorial designs, k = 2
  - two factors at two levels

- Utility
  —2^2 designs simple to analyze with regression



Goal : Understand Factor Interaction

- Non-Interaction: level of A does not changes effect of level change of B

	Memory 2GB	Memory 8GB
DiskType HDD	10 (s)	8 (s)
DiskType SSD	5 (s)	4 (s)

Goal : Understand Factor Interaction

- Non-Interaction: level of A does not changes effect of level change of B

	Memory 2GB	Memory 8GB
DiskType HDD	10 (s)	8 (s)
DiskType SSD	5 (s)	4 (s)

- Interaction: when level of A changes effect of level change of B

	Memory 2GB	Memory 8GB
DiskType HDD	10 (s)	8 (s)
DiskType SSD	5 (s)	1 (s)

- If two factors do not interact, no need to test all combinations

Result Interpretation

- Example : impact of memory size and data size on a performance.

	Memory 1GB	Memory 2GB
Data Size 0.1GB	1000 (ms)	300 (ms)
Data Size 1GB	3500 (ms)	3000 (ms)

Mean of replicates

Treatment Combination	Replicate		
	I	II	III
A low, B low	28	25	27
A high, B low	36	32	32
A low, B high	18	19	23
A high, B high	31	30	29

Result Interpretation

- Example : impact of memory size and data size on a performance.

	Memory 1GB	Memory 2GB
Data Size 0.1GB	1000 (ms)	300 (ms)
Data Size 1GB	3500 (ms)	3000 (ms)

- Non-linear regression model to interpret result
  - Define variables accounting for low/high levels
  - x_A = -1 if 1GB memory +1 if 2GB memory
  - x_B = -1 if 0.1GB data +1 if 1GB data

Non-linear regression model

A=low B=low

A=any B=any

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

A=any B=low

A=any B=high

Non-linear regression model

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

- (low,low) mem=1G, data=0.1G, time=1000 (ms)

$$1000 = q_0 + q_A x_A + q_b x_B + q_{AB} x_A x_B$$

$$1000 = q_0 + q_A * -1 + q_b * -1 + q_{AB} -1 * -1$$

$$1000 = q_0 - q_A - q_b + q_{AB}$$

Non-linear regression model

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

- (high,low) mem=2G, data=0.1G, time=300(ms)

$$300 = q_0 + q_A x_A + q_b x_B + q_{AB} x_A x_B$$

$$300 = q_0 + q_A * +1 + q_b * -1 + q_{AB}\ 1 * -1$$

$$300 = q_0 + q_A - q_b - q_{AB}$$

Non-linear regression model

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

- (low,high) mem.=1G, data=1G, time=3500(ms)

$$3500 = q_0 + q_A x_A + q_b x_B + q_{AB} x_A x_B$$

$$3500 = q_0 + q_A * -1 + q_b * +1 + q_{AB}\ 1 * -1$$

$$3500 = q_0 - q_A + q_b - q_{AB}$$

Non-linear regression model

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B$$

- (high,high) mem.=2G, data=1G, time=3000(ms)

$$3000 = q_0 + q_A x_A + q_b x_B + q_{AB} x_A x_B$$

$$3000 = q_0 + q_A * +1 + q_b * +1 + q_{AB}\ 1*1$$

$$3000 = q_0 + q_A + q_b + q_{AB}$$

Non-linear regression model

$$1000 = q_0 - q_A - q_B + q_{AB}$$
$$300 = q_0 + q_A - q_B - q_{AB}$$
$$3500 = q_0 - q_A + q_B - q_{AB}$$
$$3000 = q_0 + q_A + q_B + q_{AB}$$

- 4 equations in 4 variables : unique solution

$$y = \mathbf{1950} - \mathbf{300}x_A + \mathbf{1300}x_B + \mathbf{50}x_Ax_B$$

Non-linear regression model

$$y = \mathbf{1950} - \mathbf{300}x_A + \mathbf{1300}x_B + \mathbf{50}x_Ax_B$$

Interpreted as:

- the base response time is **1950** ms
- effect of memory improves performances of **300**ms
  - regressive effect because of negative sign
- more data can lower performances of **1300**ms
- the interaction between memory and data accounts for **50**ms
  - the greater the factor (wrt mean) the greater the interaction

Shortcut to compute coefficients

- More generally

$$y_1 = q_0 - q_A - q_B + q_{AB} \qquad y_2 = q_0 + q_A - q_B - q_{AB}$$
$$y_3 = q_0 - q_A + q_B - q_{AB} \qquad y_4 = q_0 + q_A + q_B + q_{AB}$$

- Resolution always leads to

$$\mathbf{q_0} = \tfrac{1}{4}(y_1 + y_2 + y_3 + y_4)$$
$$\mathbf{q_A} = \tfrac{1}{4}(-y_1 + y_2 - y_3 + y_4)$$
$$\mathbf{q_B} = \tfrac{1}{4}(-y_1 - y_2 + y_3 + y_4)$$
$$\mathbf{q_{AB}} = \tfrac{1}{4}(+y_1 - y_2 - y_3 + y_4)$$

More Formulas

(Estimated) Variation due to memory

$$\frac{4(q_A)^2}{4(q_A)^2 + 4(q_B)^2 + 4(q_{AB})^2}$$

~ 5%

More Formulas

(Estimated) Variation due to data

$$\frac{4(q_B)^2}{4(q_A)^2 + 4(q_B)^2 + 4(q_{AB})^2}$$

~ 95%

More Formulas

(Estimated) Variation due to interaction

$$\frac{4(q_{AB})^2}{4(q_A)^2 + 4(q_B)^2 + 4(q_{AB})^2}$$

~ 0.1%

Conclusion on experiment design

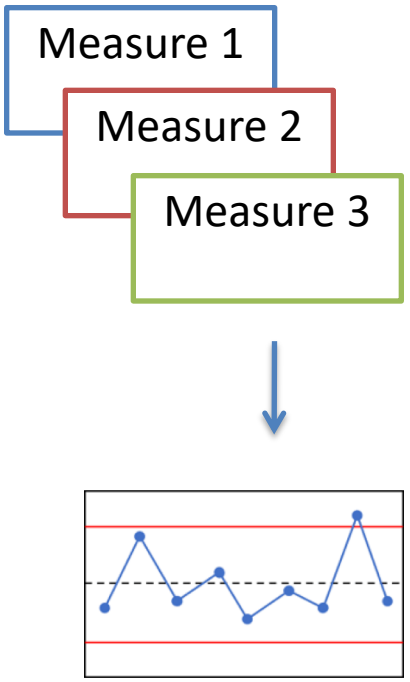
- Picking the factors, their levels, and the replication degree (number of repetitions)

- Design has a huge impact:
  - You don't know what you haven't tested
  - Ignoring important parameter leads to brittle results
  - Testing all possible level combinations is unfeasible

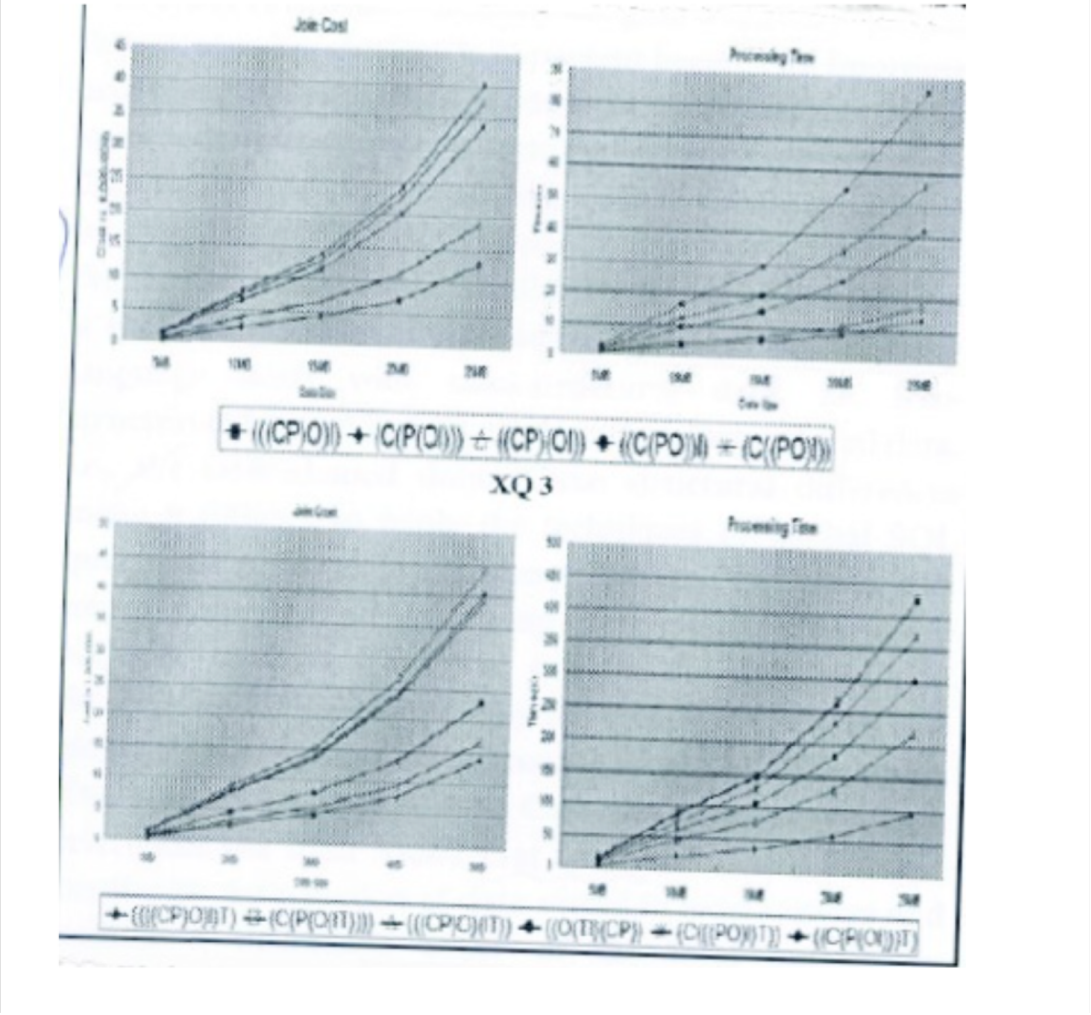
- There exist standard, well-founded procedures for getting the same information with less effort:
  - Run a $\mathbf{2^k}$ design
  - Evaluate factor importance
  - Pick important factors and possibly refine levels

REPORTING ON EXPERIMENTS

Performance Analysis

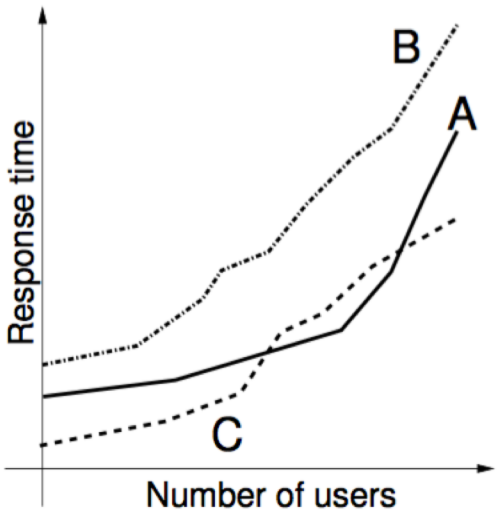


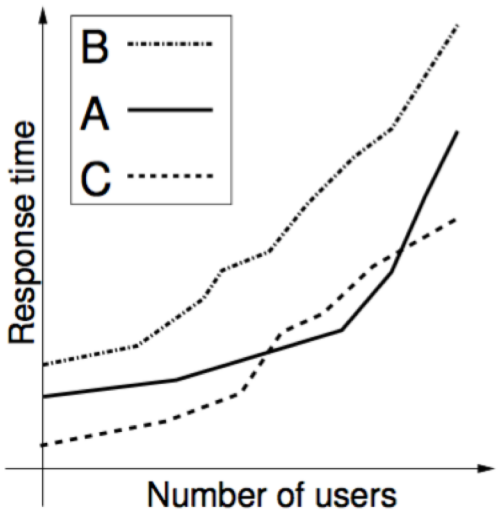
A picture worth a thousand words (maybe..)

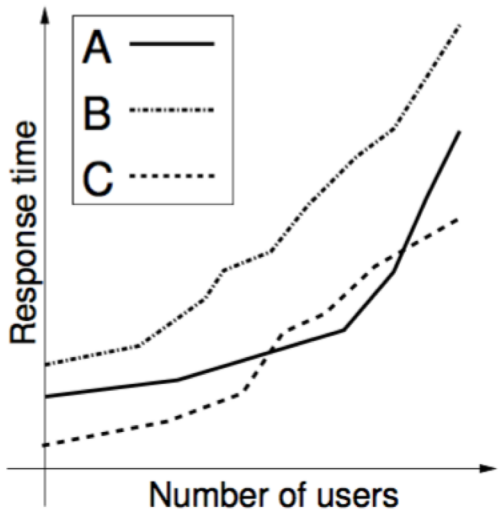


Guidelines for Graphical Charts

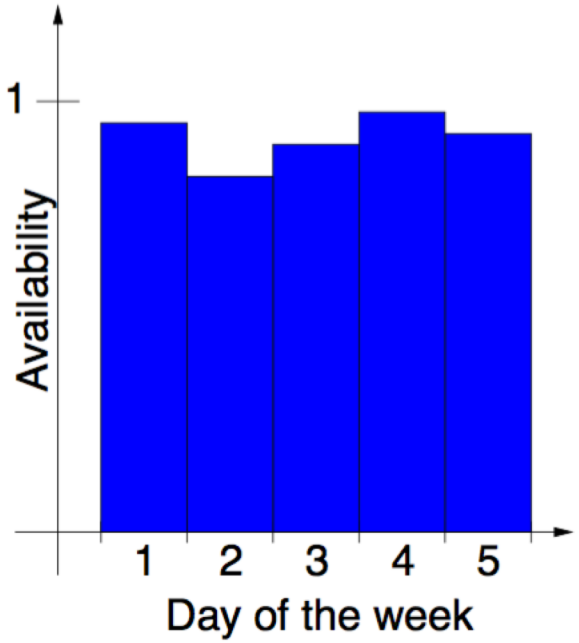
- Require minimum effort from the reader
  - Not the minimum effort from you !!





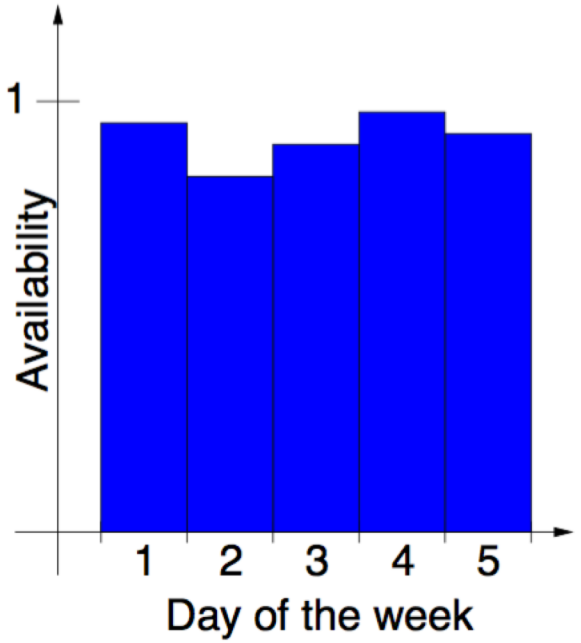


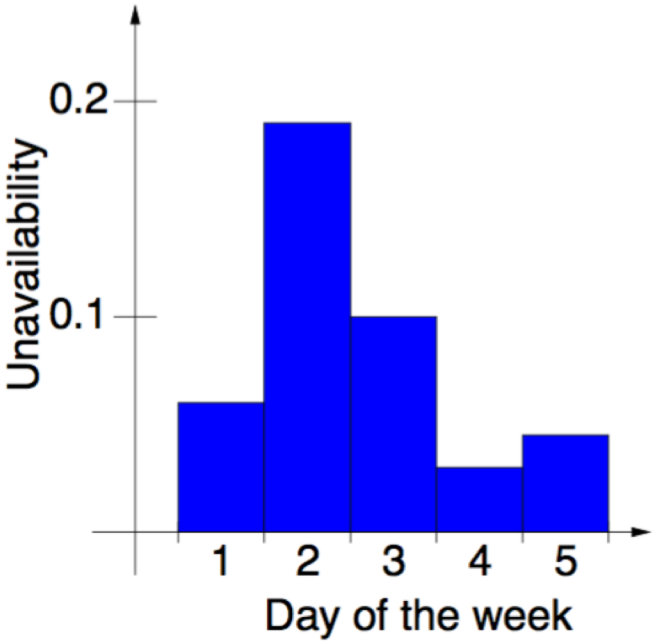
Guidelines for graphical charts



Guidelines for graphical charts

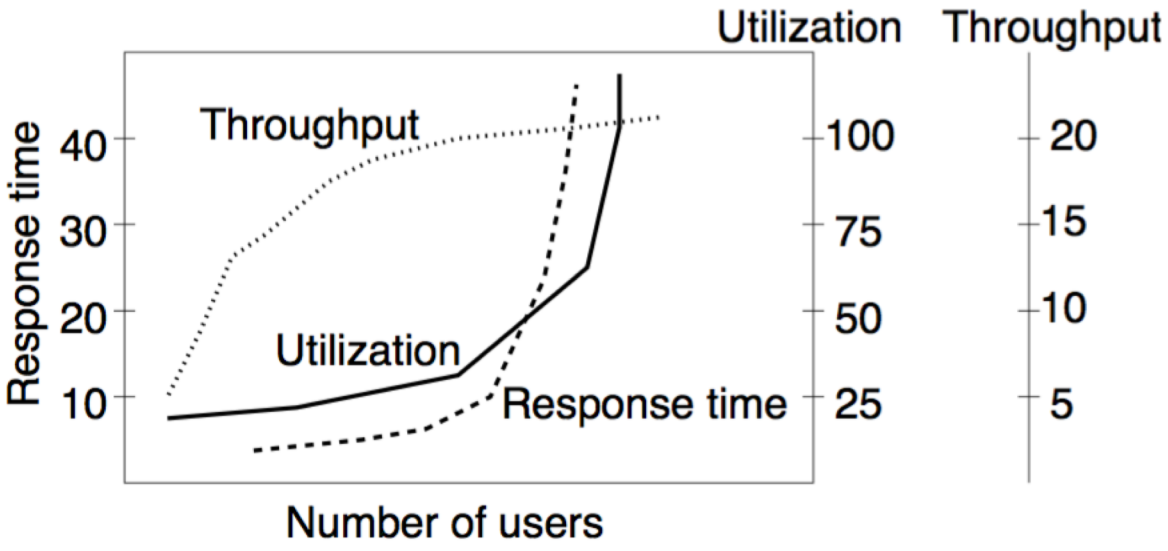
- Prefer the chart that minimize ink while maximizing information





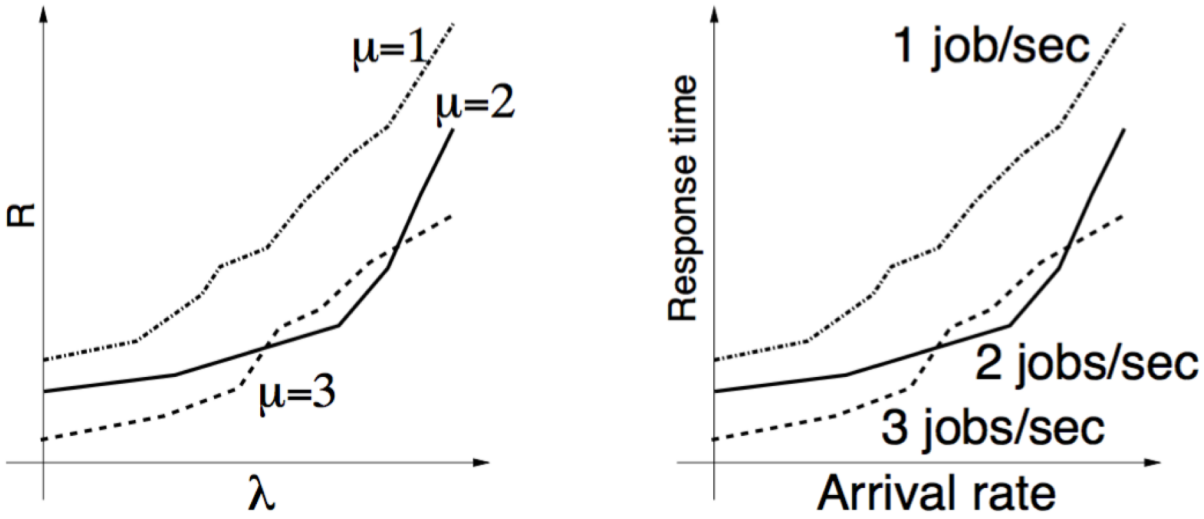
Common presentation mistakes

- Presenting many result variables on a single chart
  - (commonly done to fit into available space)



Common presentation mistakes

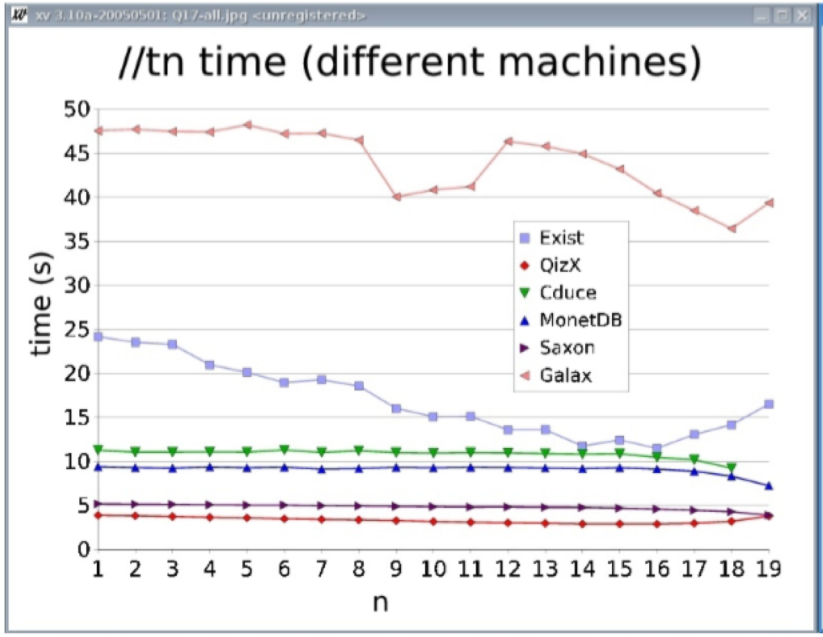
- Using symbols in place of text is bad

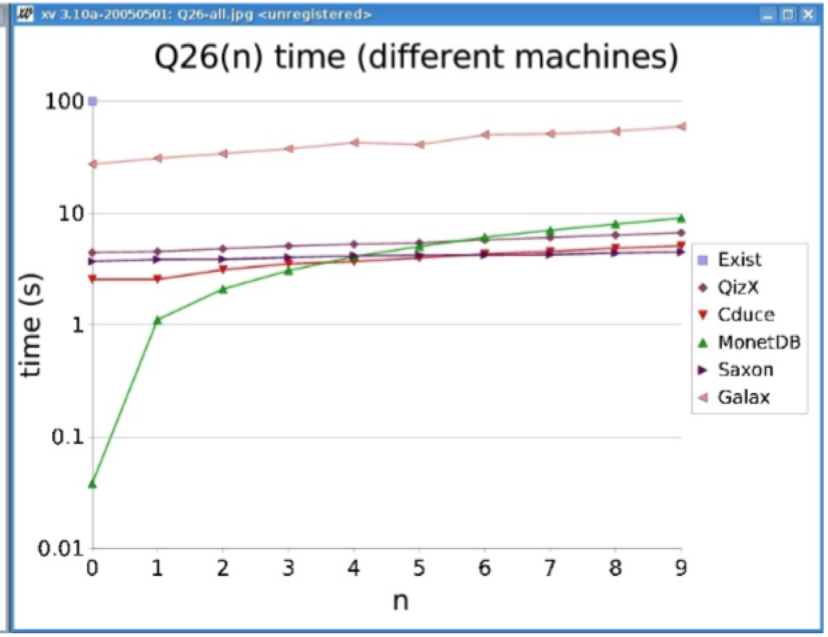


- Human brain is a poor join processor
- Humans get frustrated by computing joins

Common presentation mistakes

- Avoid using different scales for similar experiments





Guidelines for Graphical Charts

- Maximize information: try to make the graph self-sufficient (no missing descriptions/legend)

- Use keywords in place of symbols to avoid a join in the reader's brain

- Use informative axis labels:
  prefer "*Average I/Os per query*"
  to "*Average I/Os*" or "*I/Os*"

Guidelines for Graphical Charts

- Include units in the labels: prefer “CPU time (ms)” to “CPU time”
- Usually axes begin at 0, the factor is plotted on x, the result on y
- Usually scales are linear, increase from left to right, divisions are equal
  - Use exceptions as necessary

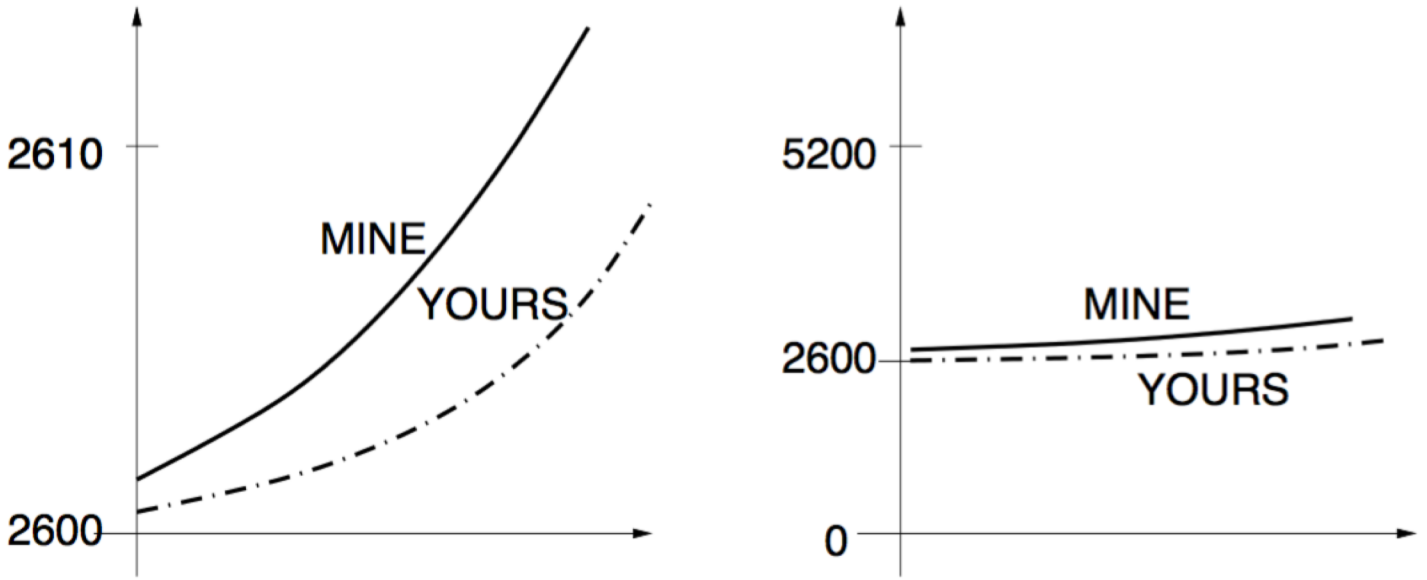
Common presentation mistakes

- Avoid presenting too many alternatives on a single chart

- Rules of thumb, to override with good reason:
  - A line chart should be limited at 6 curves
  - A column chart or bar should be limited to 10 bars
  - A pie chart should be limited to 8 components
  - Each cell in a histogram should have at least five data points

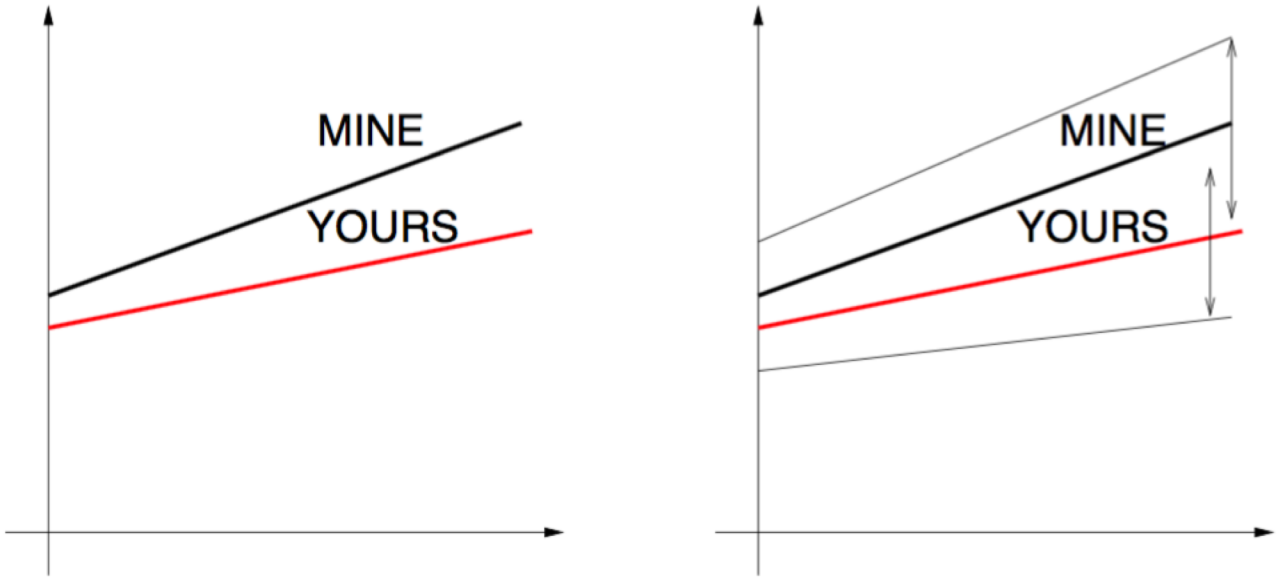
Pictorial Games

- MINE is better than YOURS! (Really?)



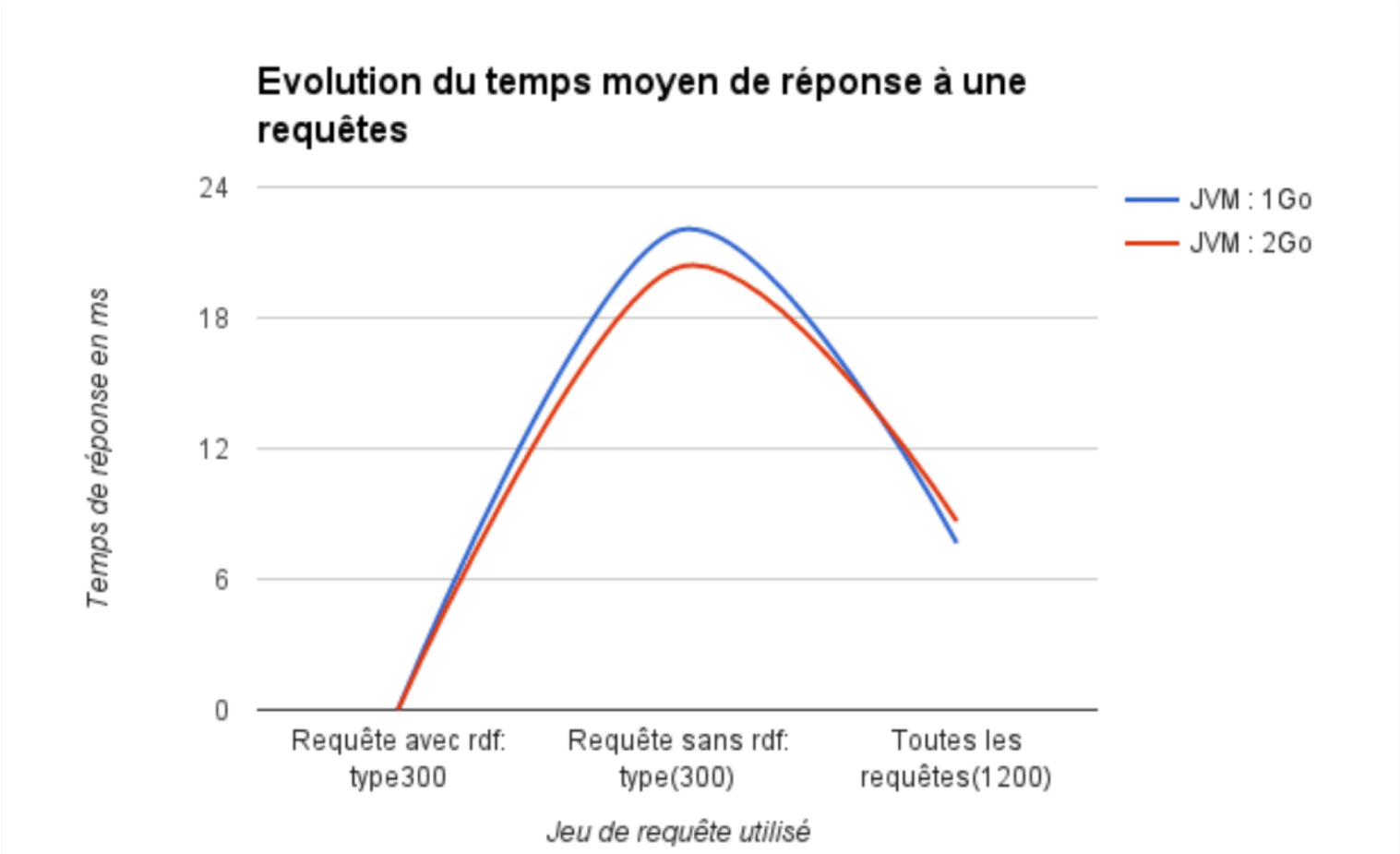
Pictorial Games

- Be careful plotting quantities w/out confidence intervals

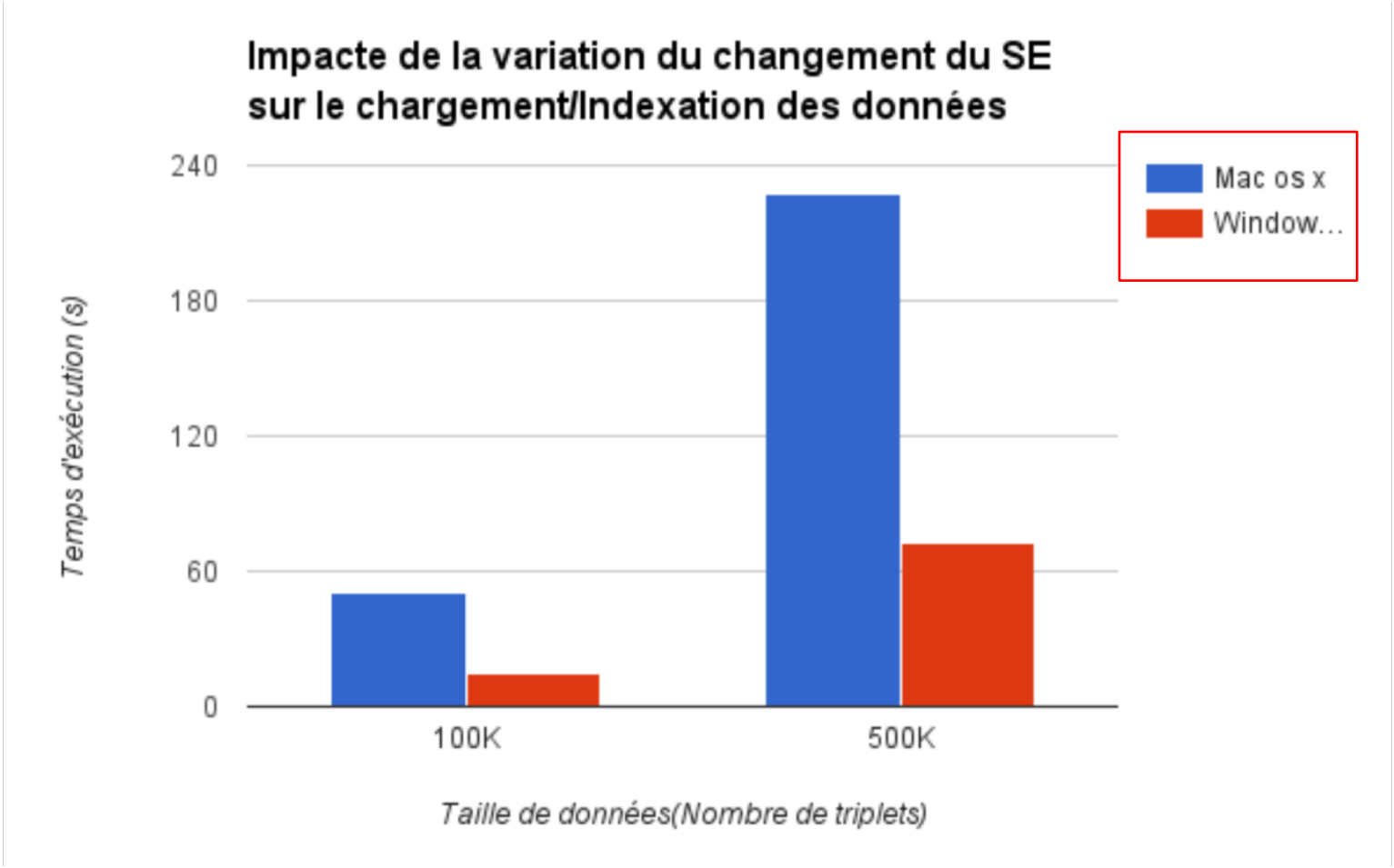


- Overlapping confidence intervals sometimes mean the two quantities are statistically indifferent

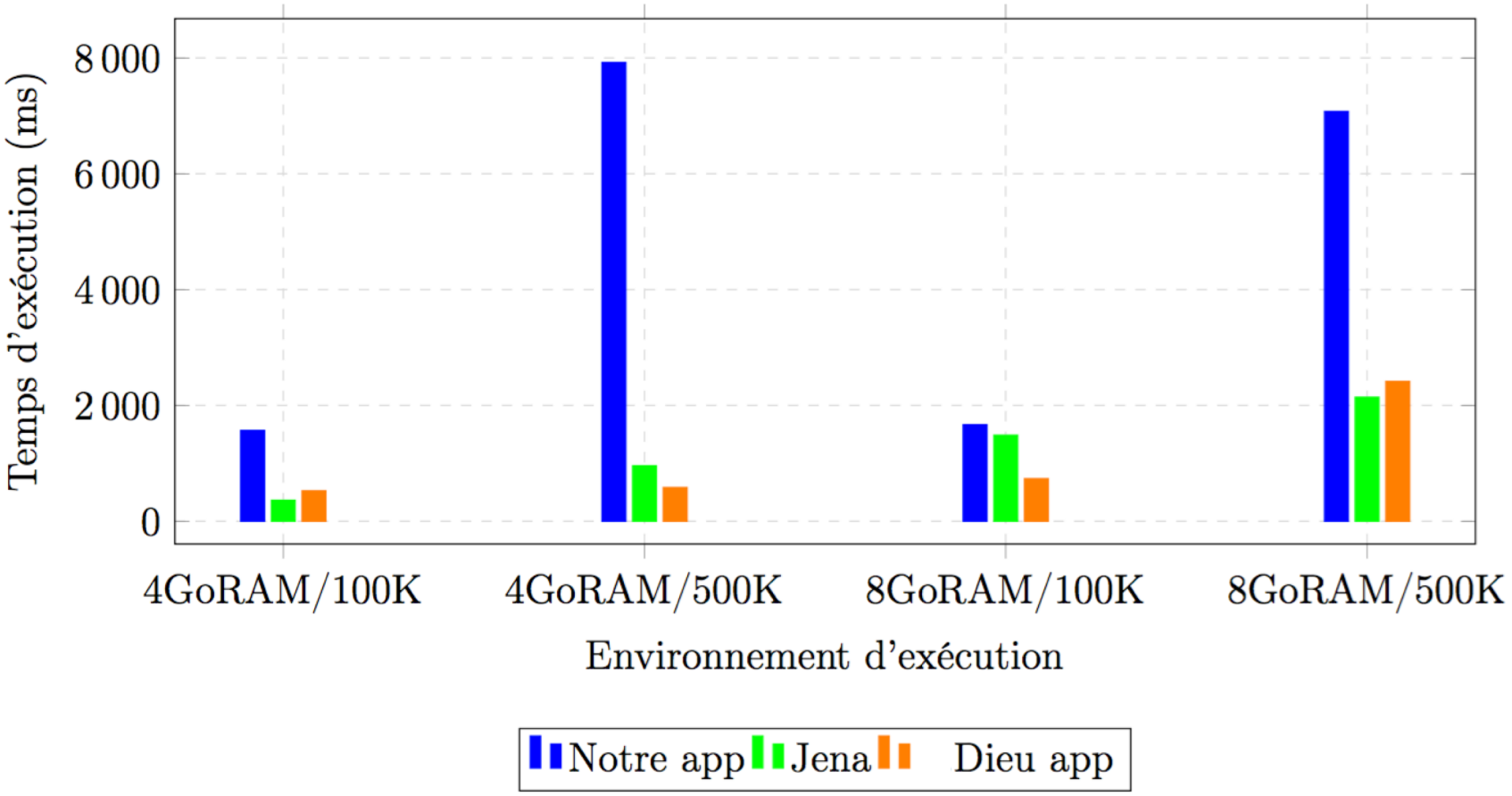
Never replace hystograms with interpolated curves!



You probably were not comparing OSX with Windows ! (but two whole machines HW+SW)

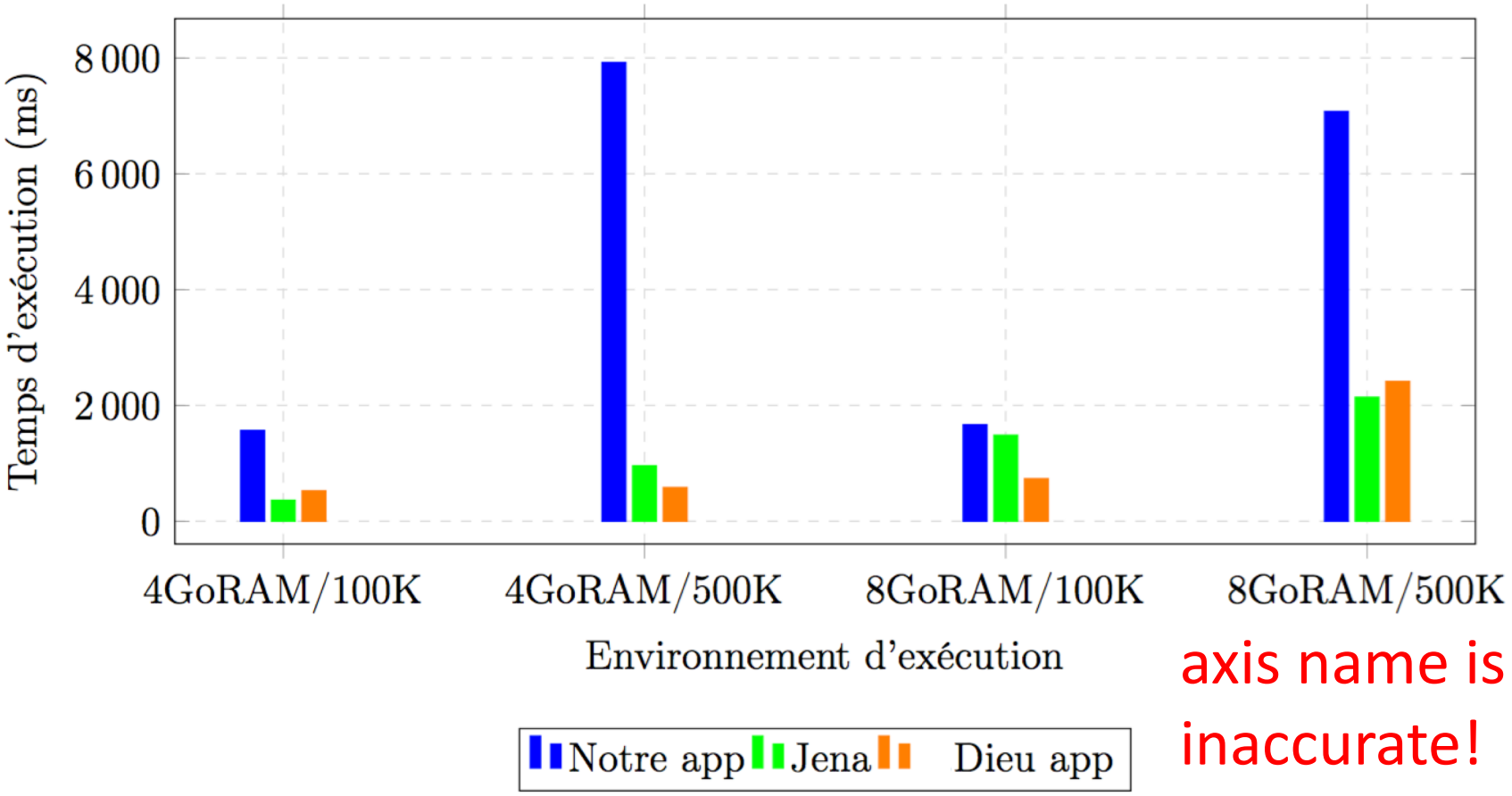


Using GNUPLOT for your charts is a good idea



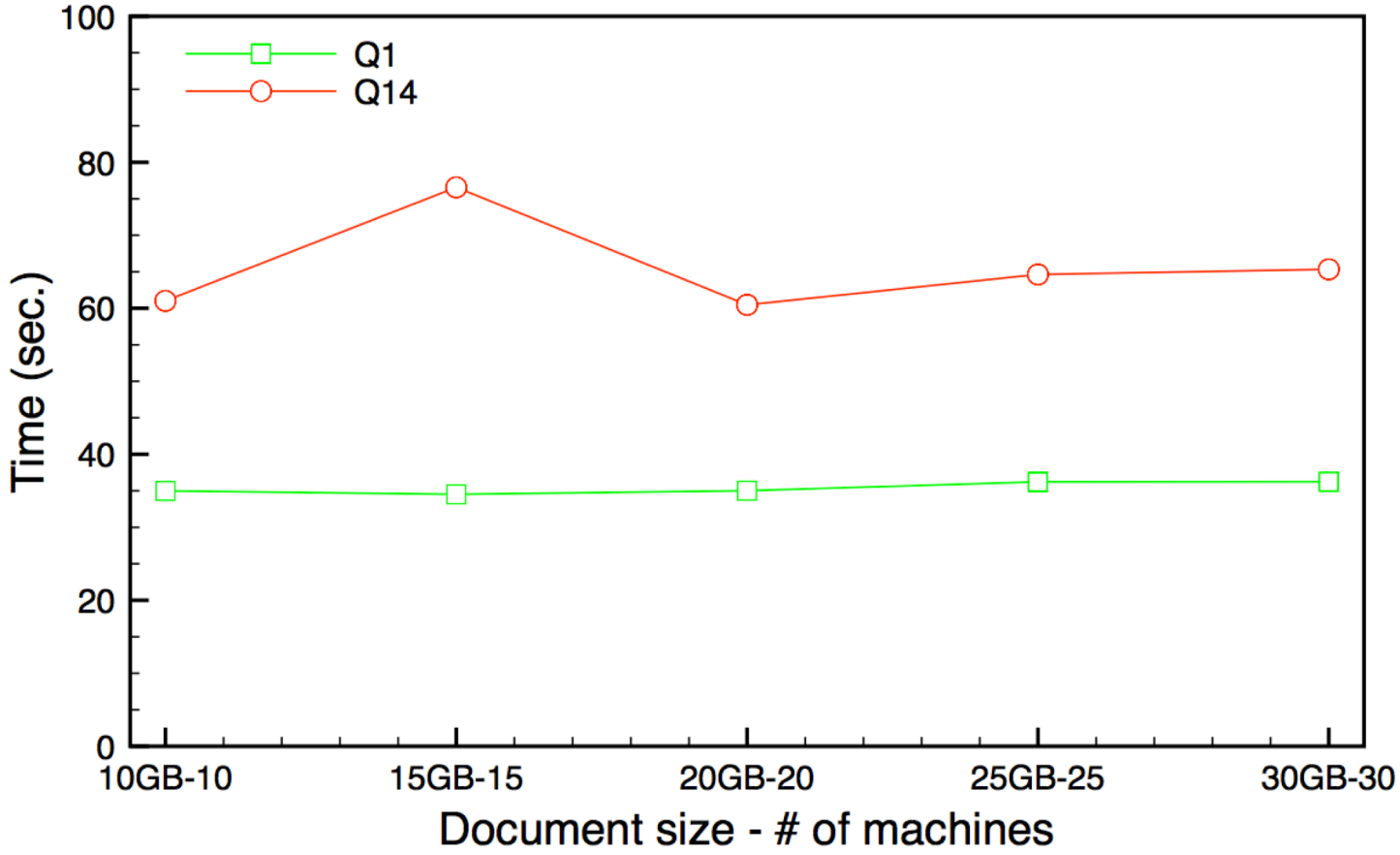
A nice chart but…

here is better to use seconds

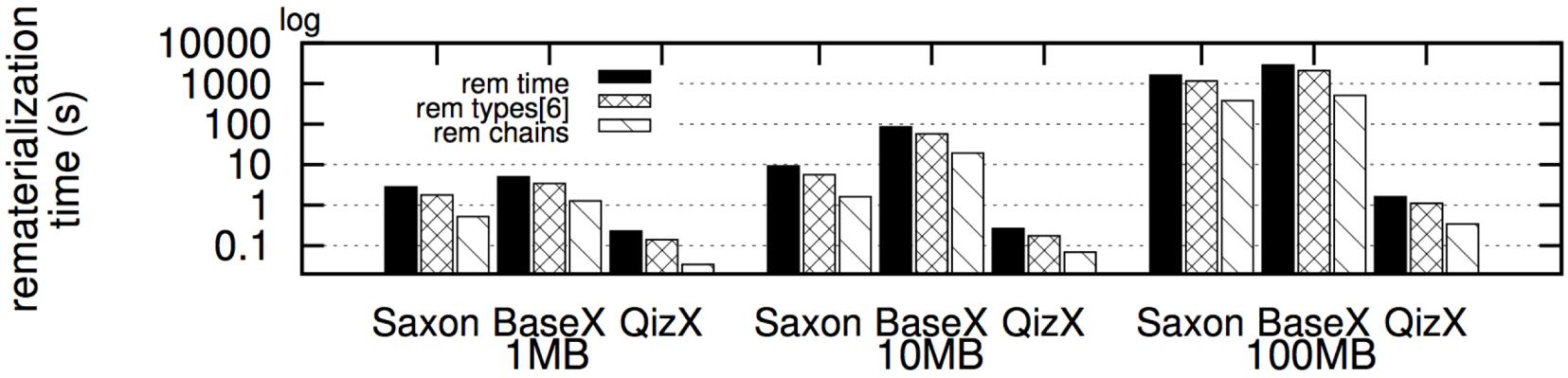


axis name is inaccurate!

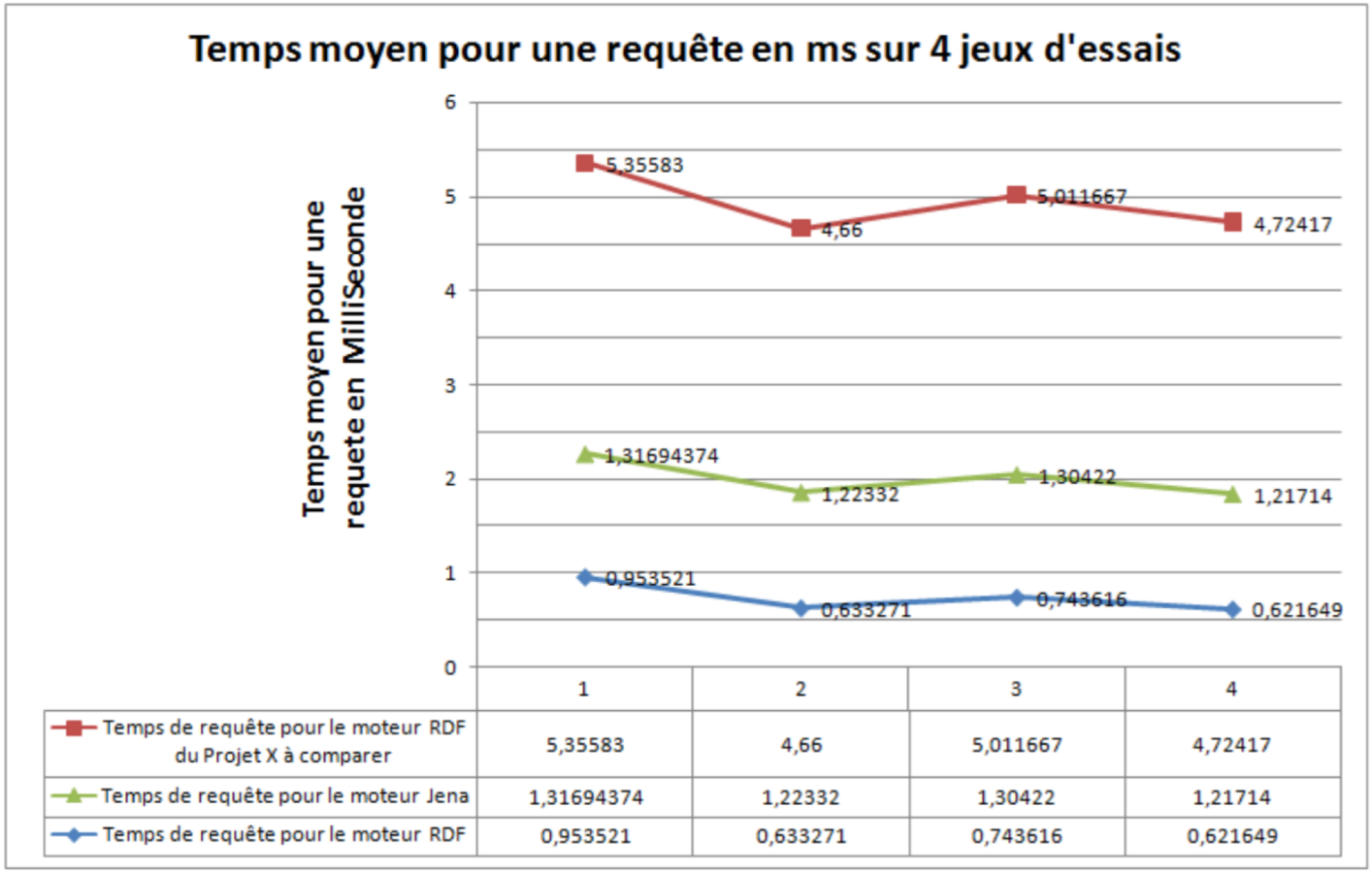
More accurate axis name



Advanced charts :
3 systems 3 techniques 3 datasets

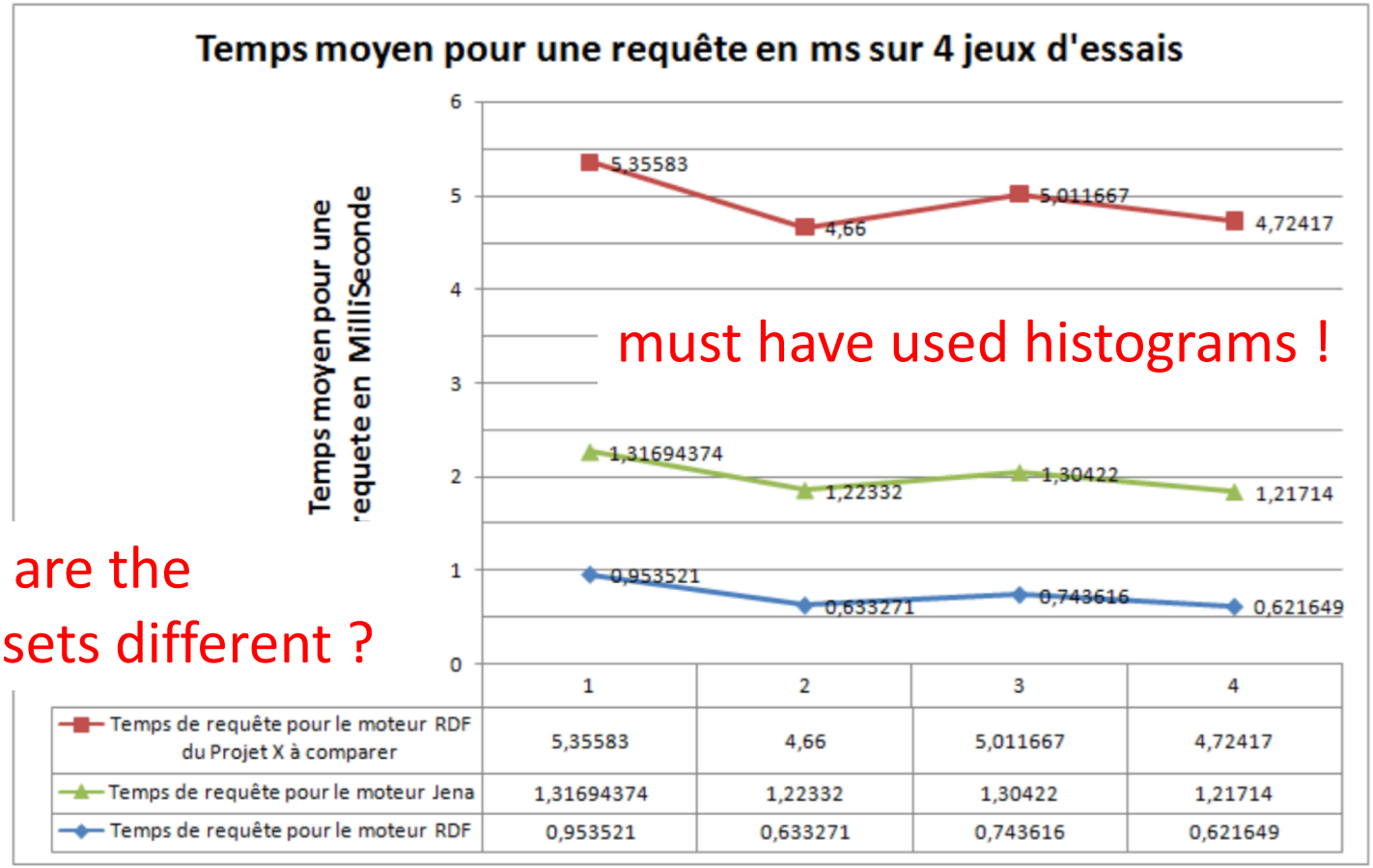


A (not so) nice chart…



A (not so) nice chart…

values repeated twice !

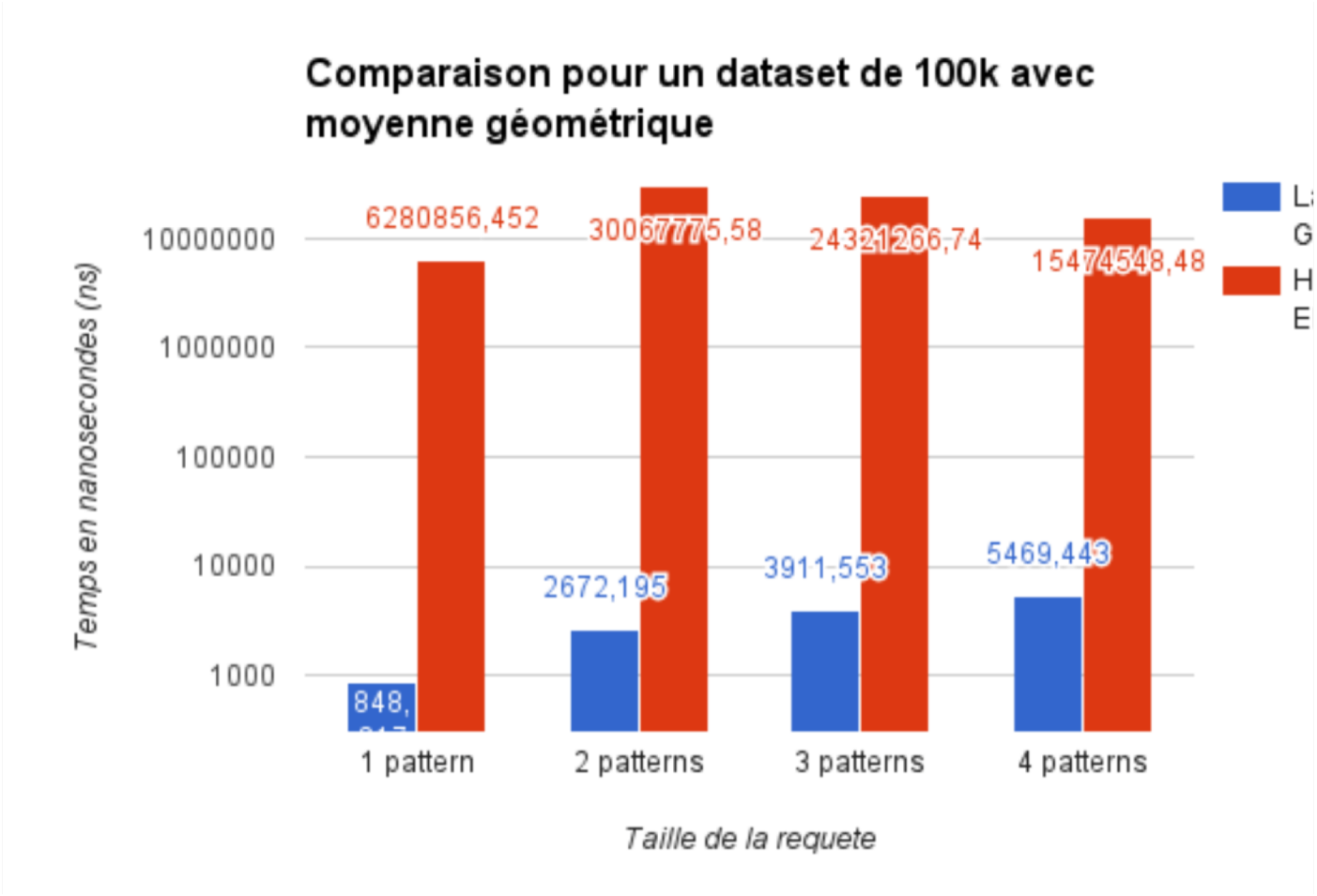


must have used histograms !

how are the datasets different ?

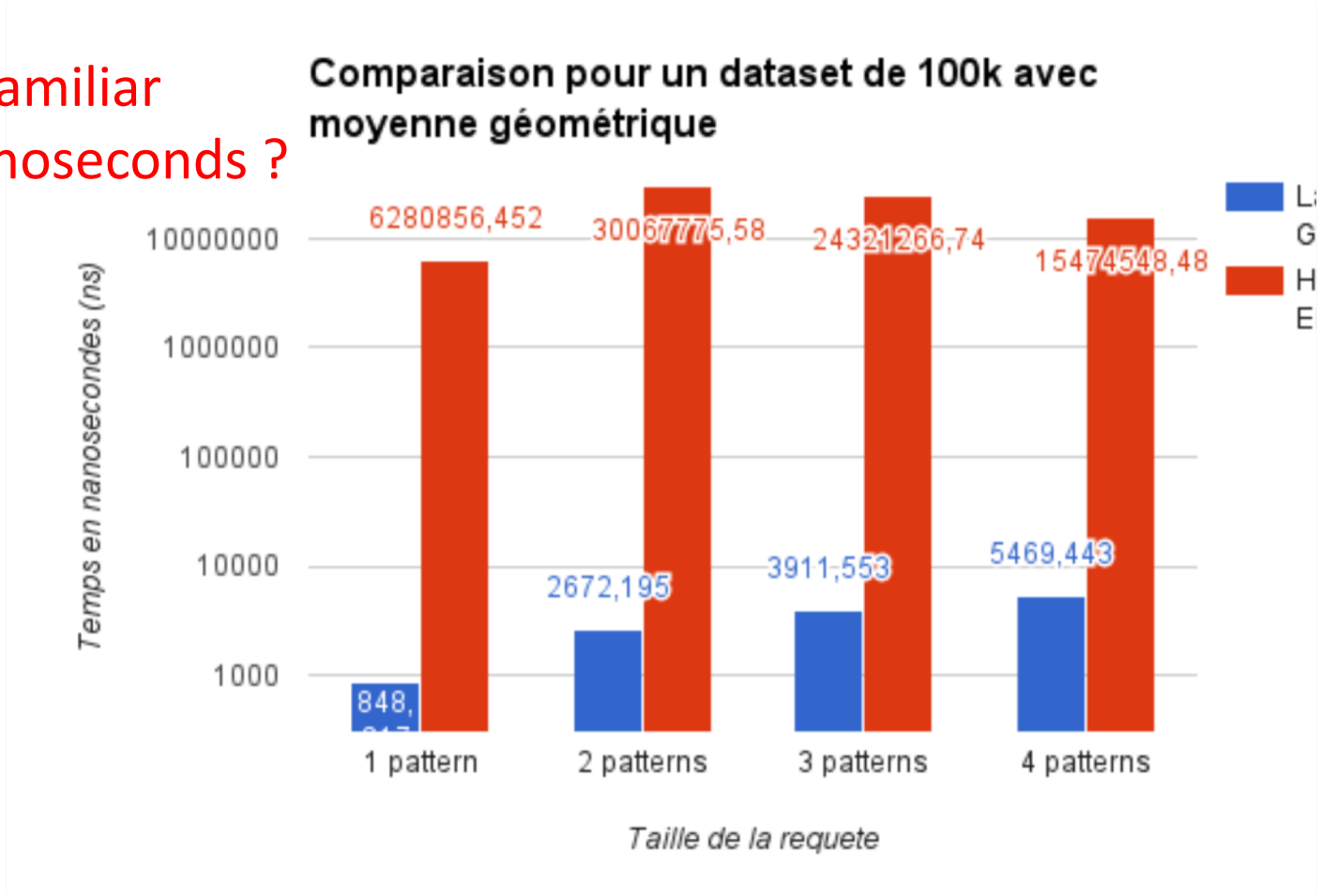
too many digits ! (especially for milliseconds)

A nice chart but…



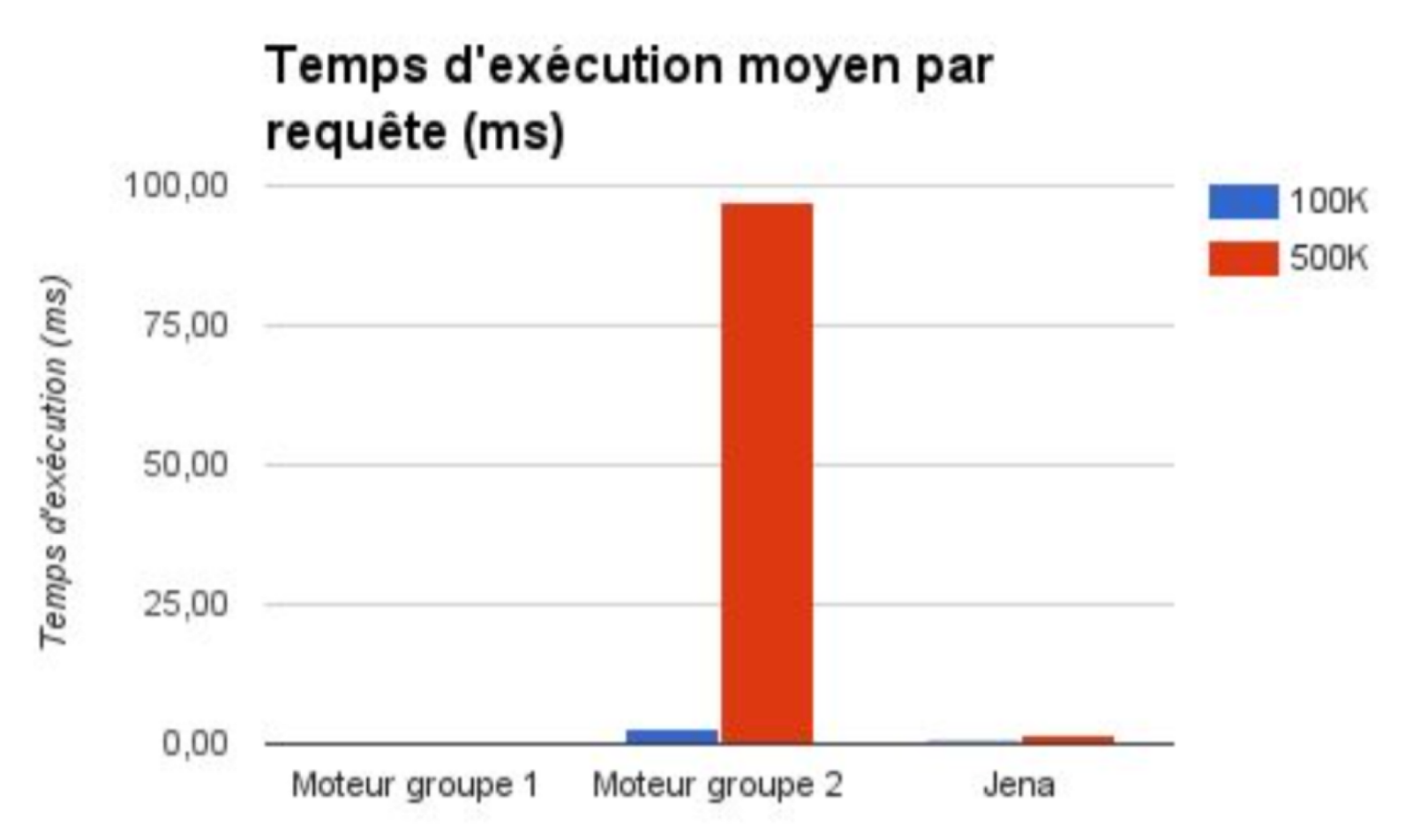
A nice chart but…

who is familiar
with nanoseconds ?

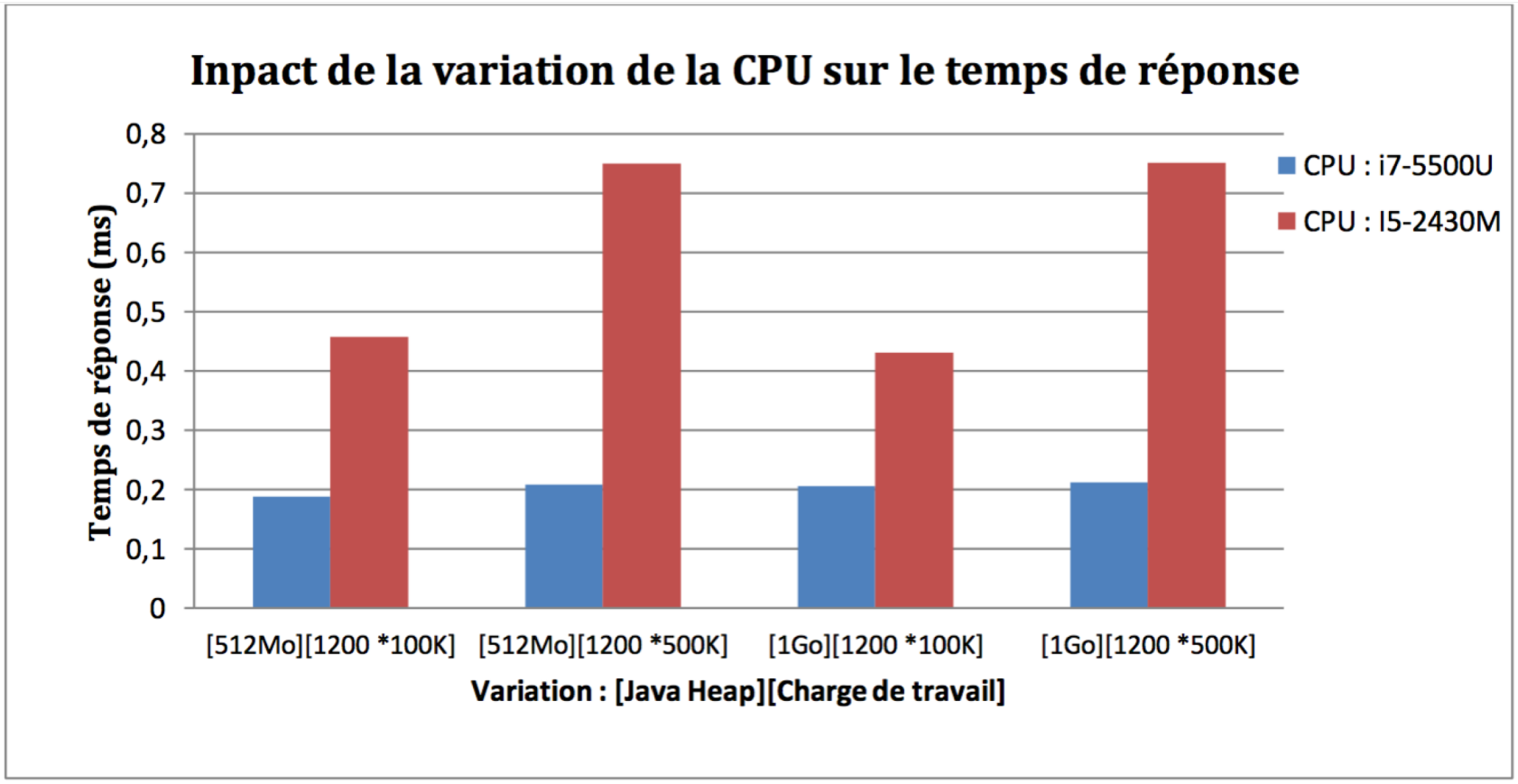


too many digits ! (especially for NANOseconds)

Linear scale does not always work best !

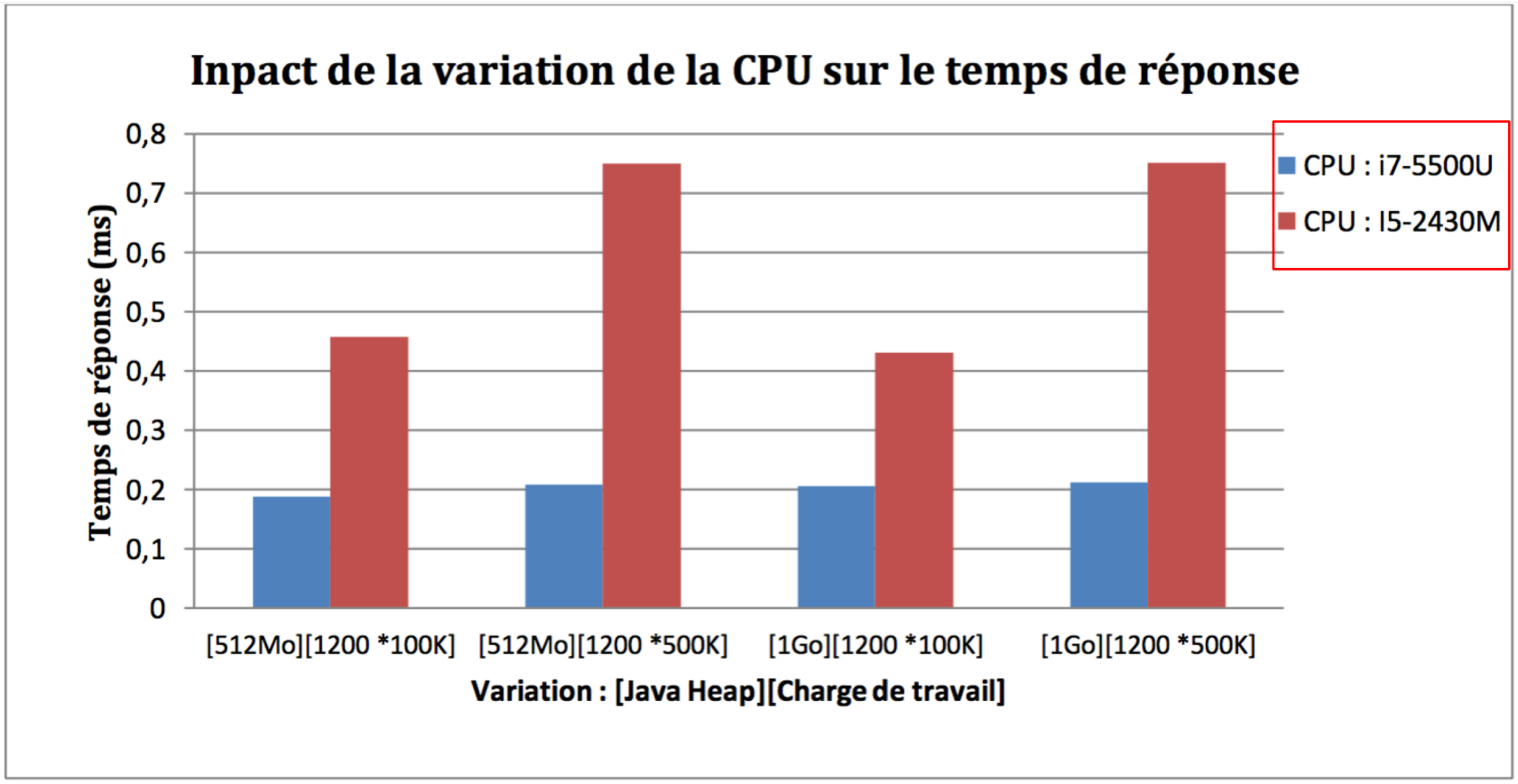


A nice chart but…



A nice chart but…

why caring about variation of performance by machine ??



You probably were not comparing two CPUs
(but two machines HW+SW)

CONCLUSION

Make experiments

Practical work requires experiments

Experiments should show
- that a system works
- that a system works better than another one

Saying that a solution is more sophisticated and hence better is NOT a valid argument!

In the experiments…

- use standard benchmark datasets if possible
- compare with best solutions in “state of the art”
- run on different data sets (at least 3)
- run different competitors (at least 1, better 3)

In the experiments…

- run with different parameter settings
- explain all datasets, metrics, and settings
- discuss reasons for good and bad performance

An irreproachable experiment is when we use exactly the same dataset, setting and metrics among competitors

Summary

- Good and repeatable performance evaluation and experimental assessment require no fancy magic but rather solid craftmanship
- Proper planning helps to keep you from “getting lost” and ensure repeatability
- Repeatable experiments simplify your own work (and help others to understand it better)

Conclusions

Database performance evaluation

- There is no single way how to do it right.
- There are many ways how to do it wrong.
- Now, it’s your turn !