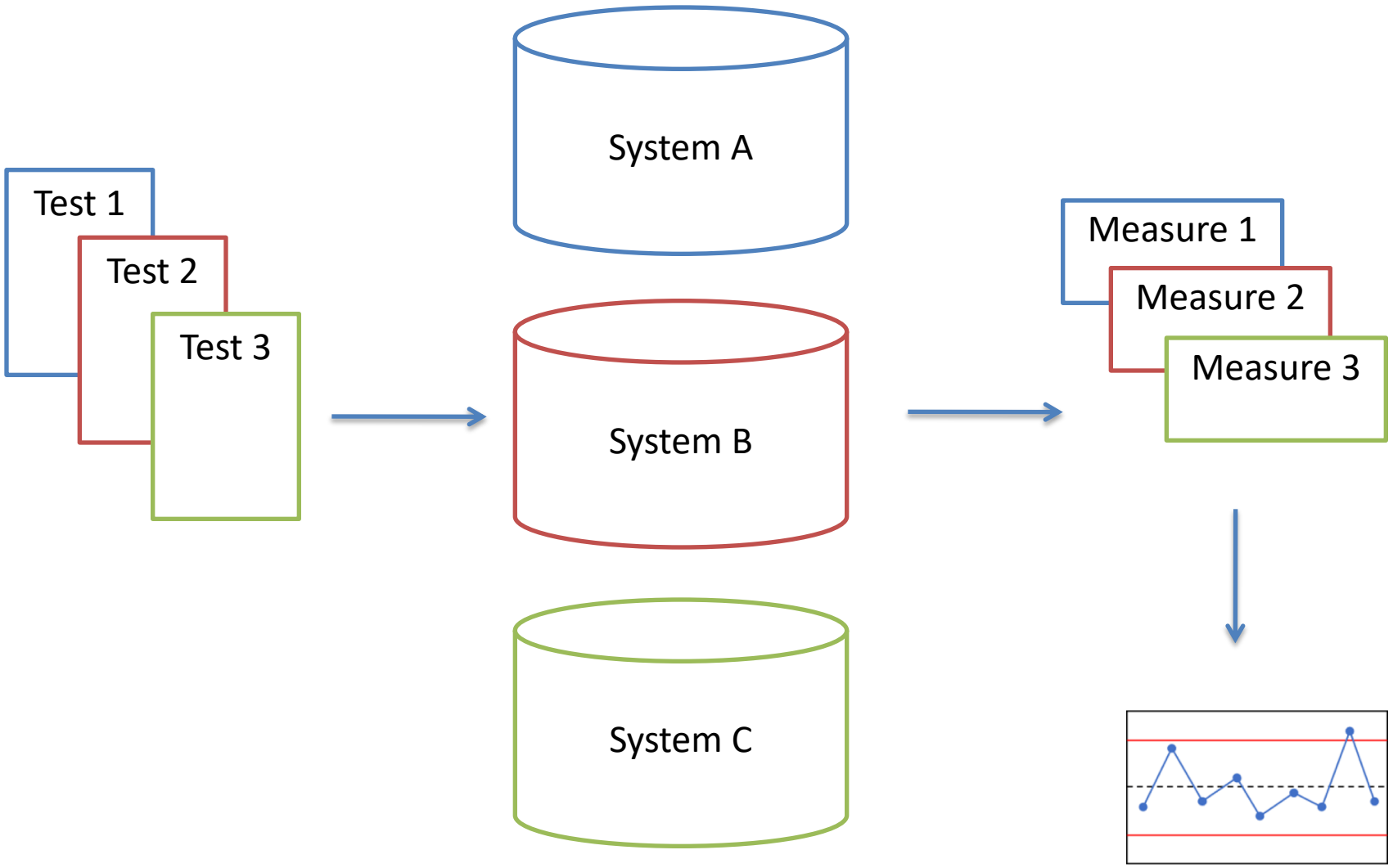
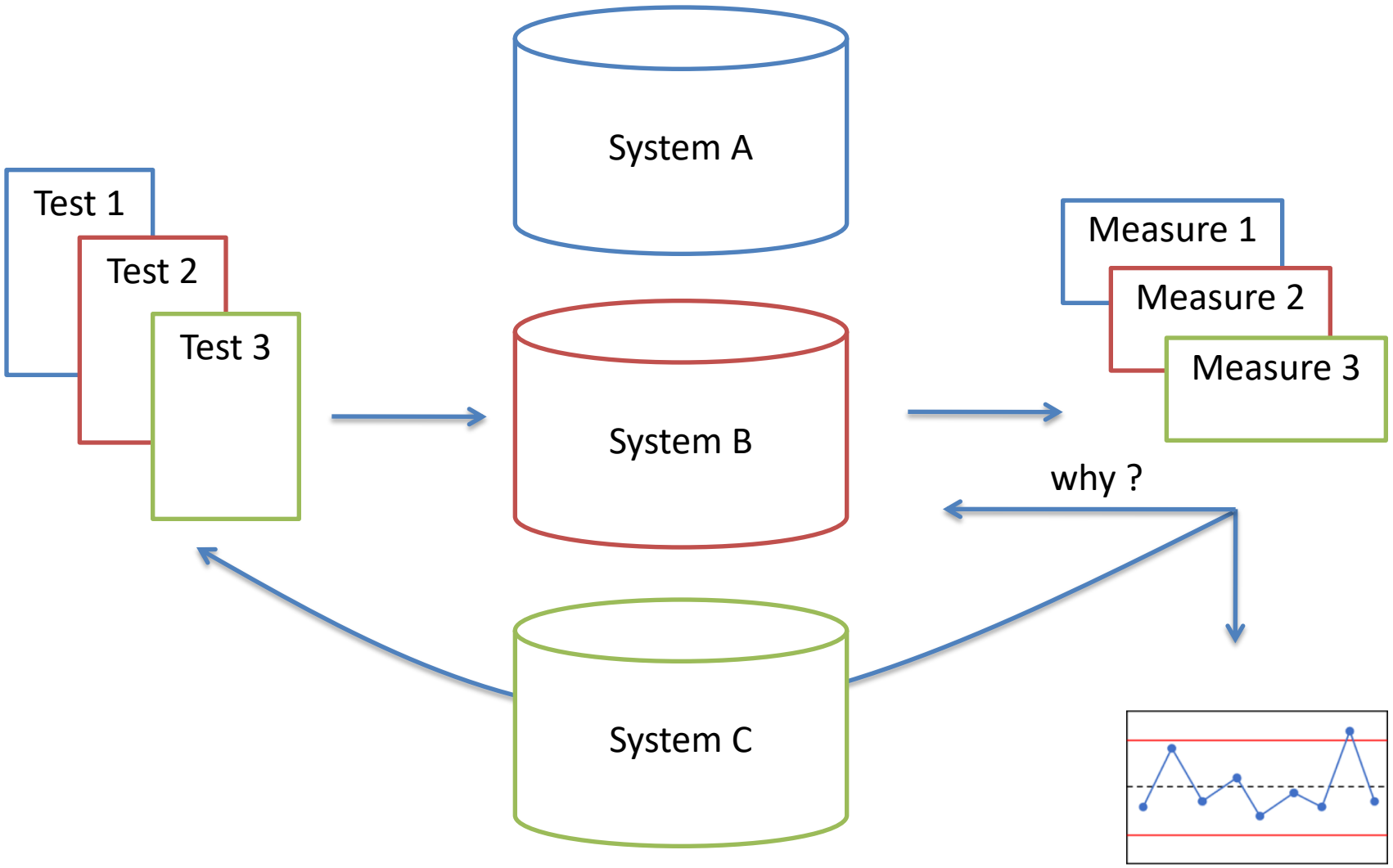
Performance Evaluation in Databases

Federico Ulliana

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

INTRO


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


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

Database Performance Evaluation

- Schematically simple, practically challenging.
  - There is no single way how to do it right.
  - There are many ways how to do it wrong.
  - You can make a carreer on performance analysis !
- This course is more a collection of anecdotes
  - providing some guidelines for what to/not-to do.

First, why making 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!

Warning : Making experiments is not science, is an art.

Questions we will try to answer today

Planning and Conducting **Database** Experiments

- Which data / data sets should be used?
- Which queries / workloads should be run?
- Which hardware / software should be used?

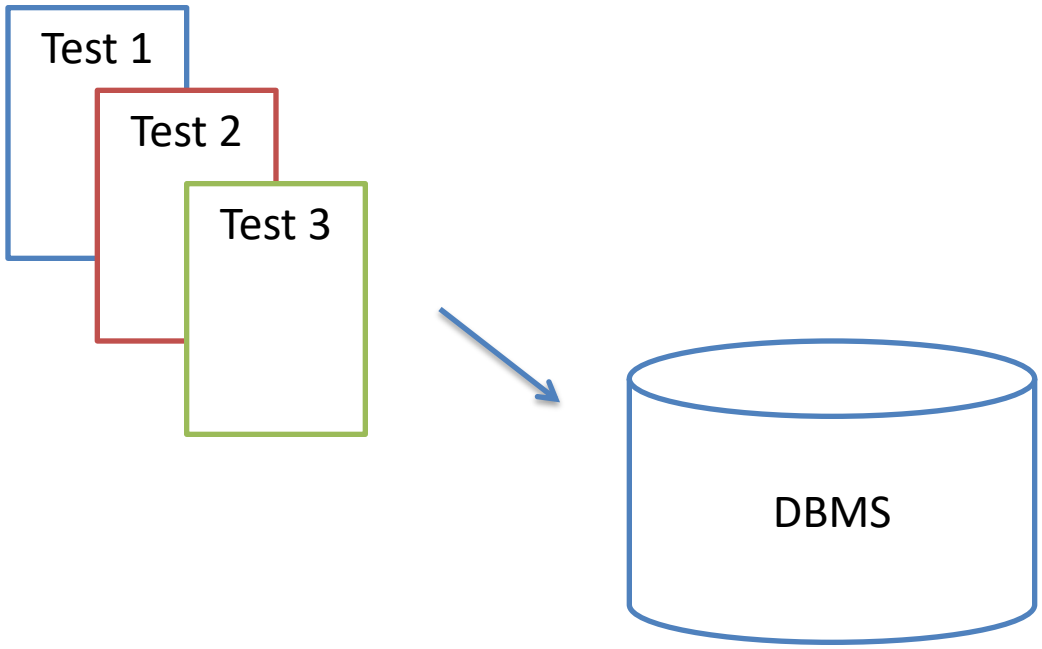
Metrics:

- What to measure ?
- How to measure? (Do not watch Youtube when you run experiments !)
- How to compare ?
- How to find out what is going on ?

How to report on experiments ?

BENCHMARKS :
WHICH INPUTS SHOULD WE USE FOR OUR TESTS ?

Performance Evaluation in Databases



What is a **Database** Benchmark ?

«A framework for assessing certain abilities of a database »

It typically contains

- data
- queries / updates (the workload)
- other specific operations

- if « synthethic » :
  - the generators of data / queries / etc..

There are 3 types of benchmarks

- Micro-benchmarks
- Standard benchmarks
- Real-life applications

To recognize them, ask yourself : who designs it ?

Micro-benchmarks

- Specific, stand-alone piece of code (think of a JUnit test, ususally written by the developers) isolating a particular piece of a larger system

- E.g., a single database functionality / operator
  - dictionnary creation
  - nested-loop
  - merge-join
  - aggregation
  - update

Dictionnary Correctness Test Example

```
TestData := [
            <Alice,WorksFor,EDF>
            <Alice,LivesIn,Paris> ]

TestResult := [ <1,2,3>,<1,4,5> ]

RunTestDico(TestResult,TestData)
```

Dictionnary Time Test Example

```
TestData := Watdiv100.rdf

RunTestDicoTime(TestData)
```

Micro-benchmarks : Pros

- Focused on a single problem
- Controllable workload and data characteristics
  - Data (synthetic & real) and size (scalability)
  - Value ranges and distribution correlation
  - Type of Queries and Workload size (scalability)
- Allow broad parameter range(s)
  - Useful for detailed, in-depth analysis
- Low setup threshold; easy to run

Micro-benchmarks : Cons

- Generalization of result sometimes difficult
- Neglect larger picture
  - e.g. contribution of local costs to global/total costs
- Neglect embedding in context/system at large
  - Application of insights in full systems / real-life applications no obvious

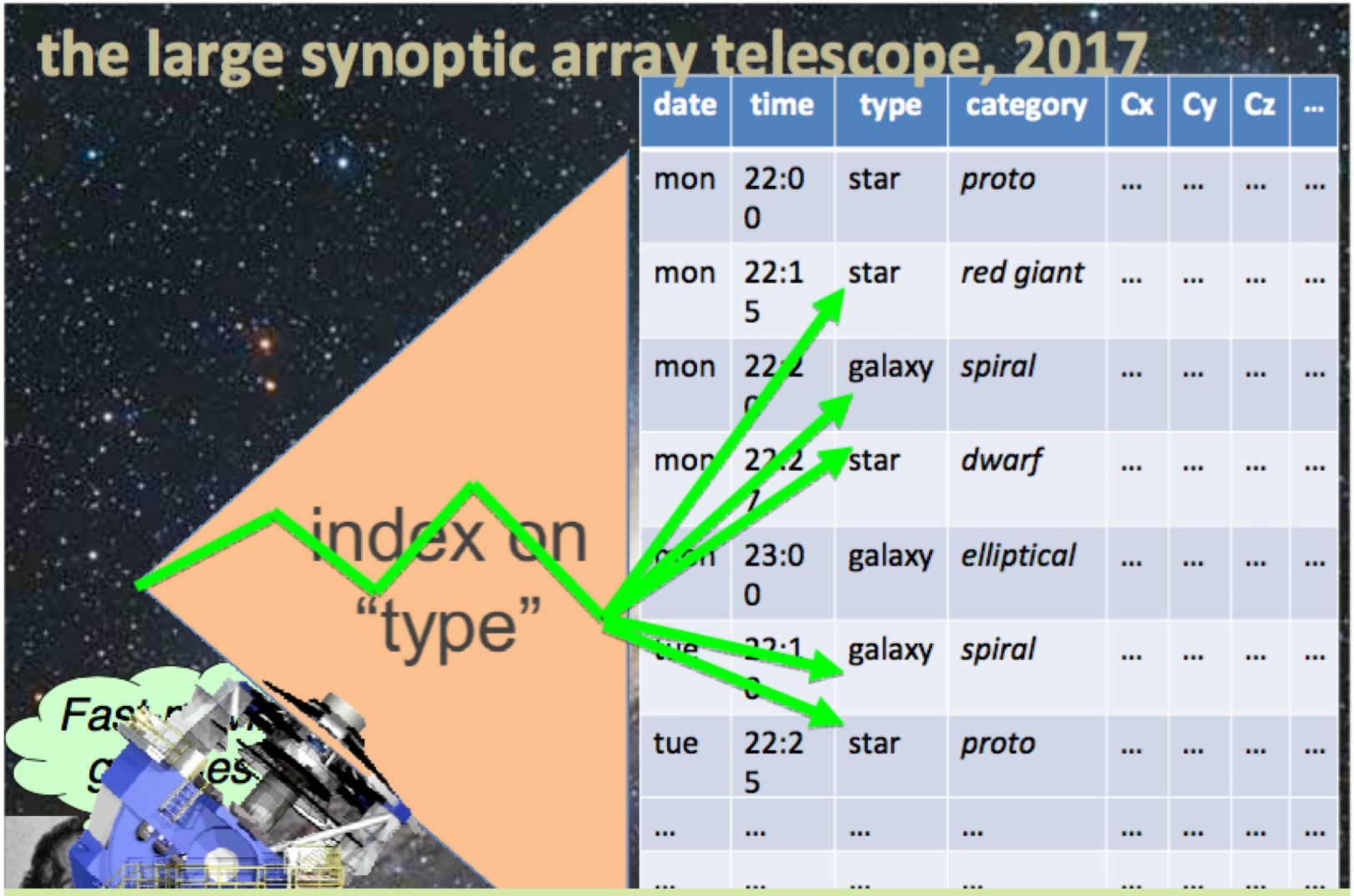
Real-life applications : Pros

- There is nothing better, at least in principle
- Existing problems and challenges

Real-life applications : Cons

- Proprietary datasets and workloads
- things are a bit evolving with open data but..

- Next example : obtaining scientific data
  - broader vision on scientific data exchange



date	time	type	category	Cx	Cy	Cz	...
mon	22:00	star	*proto*	...	...	...	...
mon	22:15	star	*red giant*	...	...	...	...
mon	22:2	galaxy	*spiral*	...	...	...	...
mon	22:2	star	*dwarf*	...	...	...	...
mon	23:00	galaxy	*elliptical*	...	...	...	...
tue	22:1	galaxy	*spiral*	...	...	...	...
tue	22:25	star	*proto*	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

100 billion objects, 20 PB/night

Meeting domain scientists

(from A. Ailamaki)

1. *"Hello, we're SO HAPPY to meet you. We have SO MUCH data! PLEASE come visit !"*
2. Visit lab, pretty pictures (we have SO MUCH data)
3. *"Let's have lunch!"* (we have SO MUCH data)
4. Revisit lab, receive promise to get some data
5. Ask for data, no reply
6. Play DBA (design/normalize schema,design data, write queries, rewrite queries, talk to tech staff)
7. Ask for data, receive 2 GB (telescope generates 20*10^6 GB/night)
8. Repeat (6), then ask again for data, receive 4GB

Reasons why collaboration (with scientists) can be hard
(from A. Ailamaki)

- Data is their achievement
  - They do not understand what we will do with it
  - They are afraid of what we may do with it
    - They may think that we will put it on the internet
  - They are not sure how we will help them
  - Do not recognize their problems in our demos

Standard Benchmarks

- Collaboratively-written benchmarks (by consortiums) aiming at testing a whole system
- RDBMS, OODBMS, ORDMBS: TPC-{A,B,C,H,R,DS}, OO7, ...
- XML, XPath, XQuery, XUF, SQL/XML: MBench, XBench, XMach-1, XMark, X007, TPoX, ...

Standard Benchmarks :

TPC (Transaction Processing Performance Council)

TPC-C - Ten Most Recently Published Results

As of 4-Nov-2015 at 1:40 PM [GMT]

Note 1: The TPC believes it is not valid to compare prices or price/performance of results in different currencies.

Date Submitted	Company	System	Performance (tpmC)	Price/tpmC	Watts/KtpmC	System Availability	Database
11/25/14	SAP	Dell PowerEdge T620	112,890	.19 USD	NR	11/25/14	SQL Anywhere 16
03/26/13	ORACLE	SPARC T5-8 Server	8,552,523	.55 USD	NR	09/25/13	Oracle 11g Release 2 Enterprise Edition with Oracle Partitioning
02/22/13	IBM	IBM System x3650 M4	1,320,082	.51 USD	NR	02/25/13	IBM DB2 ESE 9.7

'NR' in the Watts/Ktpmc column indicates that no energy data was reported for that benchmark.

Standard Benchmarks : Pros

- Mimic real-life scenarios
- Publicly available
- Well defined and scalable data sets and workloads (if well designed ...)
- Metrics well defined
- Easily comparable (?)

Standard Benchmarks : Cons

- Often “outdated” (standardization takes long)
- Often compromises
- Often very large and complicated to run
- Systems often optimized for the benchmarks !!!
- Limited dataset / workload variation

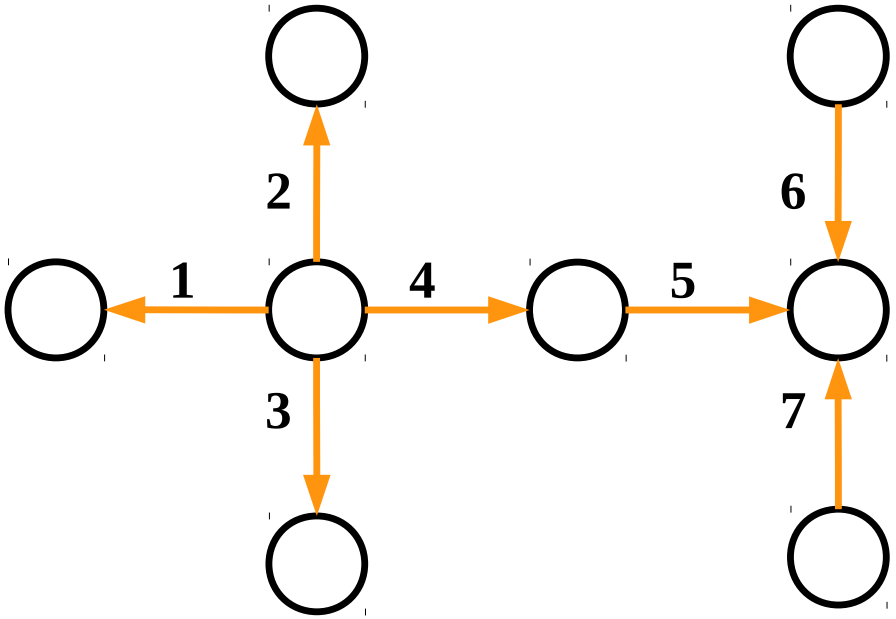
THE CASE FOR RDF BENCHMARKS (DE-FACTO STANDARDS)

Benchmark Analysis

- Often benchmarks are judged by the data they can produce
- But workloads are important as well !!
- We will survey some structural properties of queries and see whether they are testable by popular RDF benchmarks

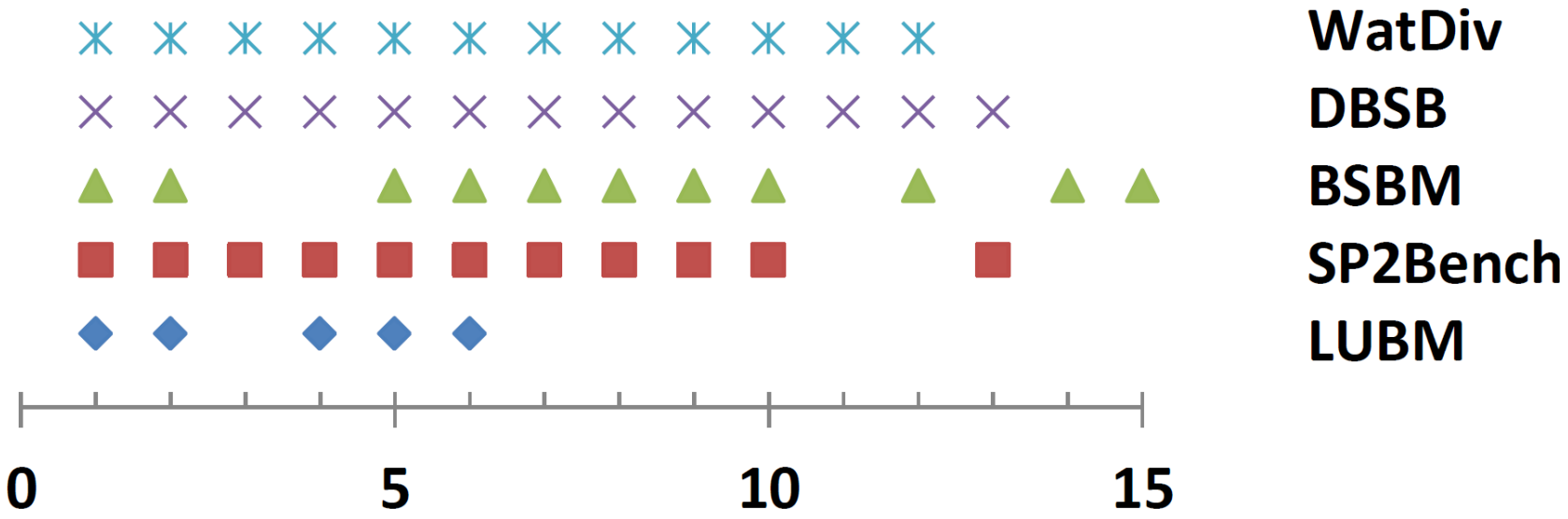
Structural Features

[Triple Pattern Count]



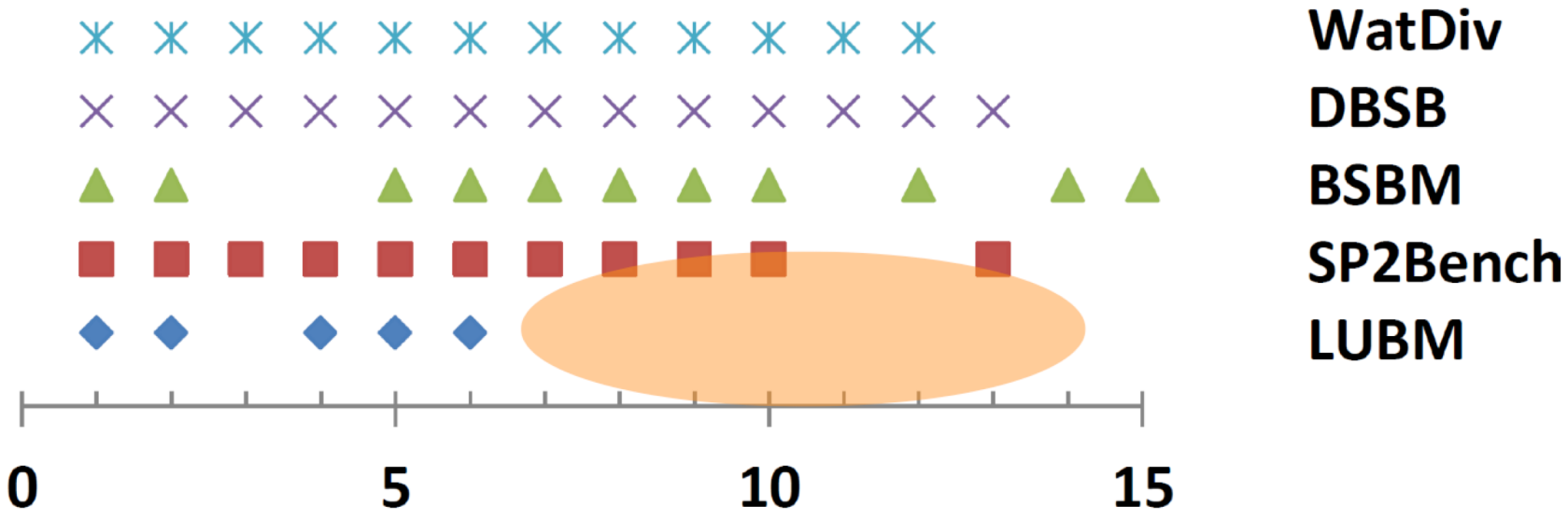
How Diverse are SPARQL Benchmarks?

[Triple Pattern Count]



How Diverse are SPARQL Benchmarks?

[Triple Pattern Count]



LUBM: A benchmark for OWL knowledge base systems

All Images News Videos Shopping More Settings Tools

About 13,800 results (0.52 seconds)

The **LUBM** features an ontology for the university domain, synthetic **OWL** data scalable to an arbitrary size, 14 extensional queries representing a variety of properties, and several performance metrics. The **LUBM** can be used to evaluate **systems** with different reasoning capabilities and storage mechanisms.

LUBM: A benchmark for OWL knowledge base systems ...

https://www.sciencedirect.com › science › article › pii

About Featured Snippets Feedback

[PDF]

LUBM: A Benchmark for OWL Knowledge Base Systems - SWAT

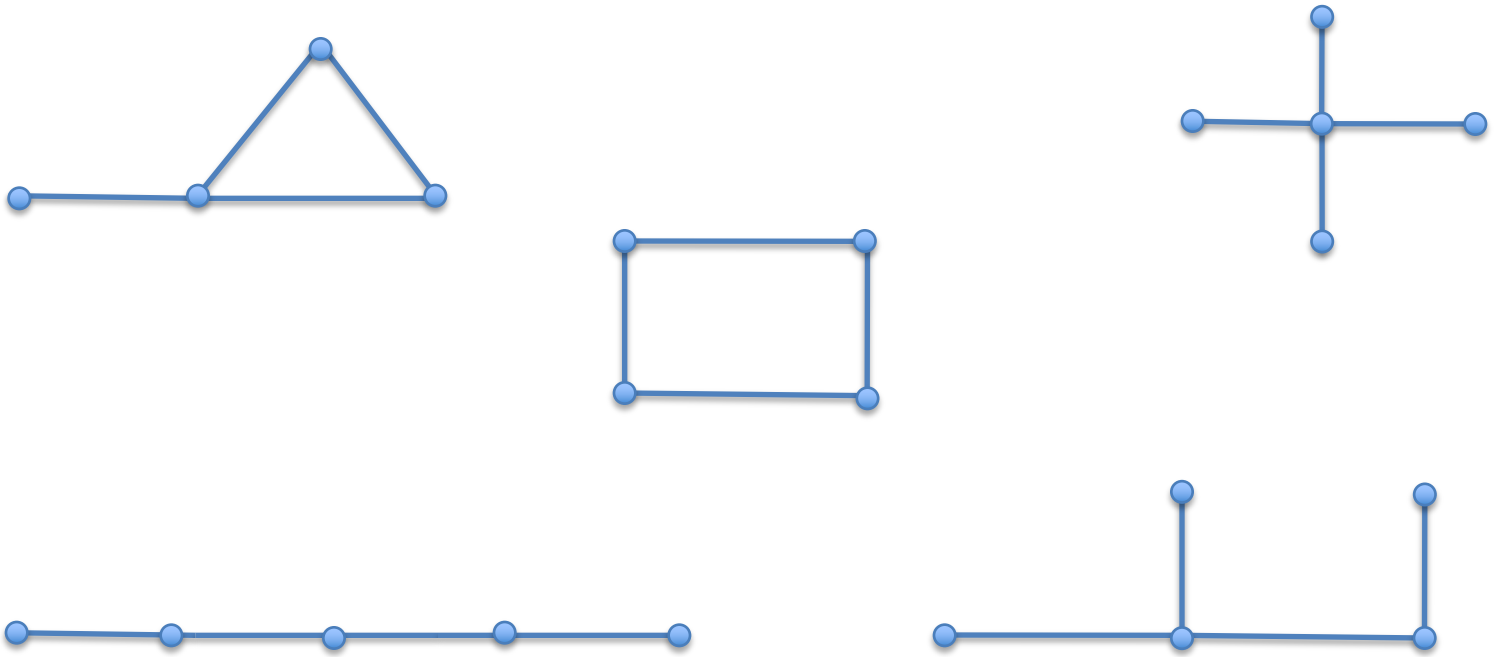
swat.cse.lehigh.edu › pubs

by Y Guo - Cited by 1468 - Related articles

The **LUBM** features an on- tology for the university domain, synthetic **OWL** data scalable to an arbitrary size, four- teen extensional queries representing a variety of properties, and several performance metrics. The **LUBM** can be used to evaluate **systems** with different reasoning capabili- ties and storage mechanisms.

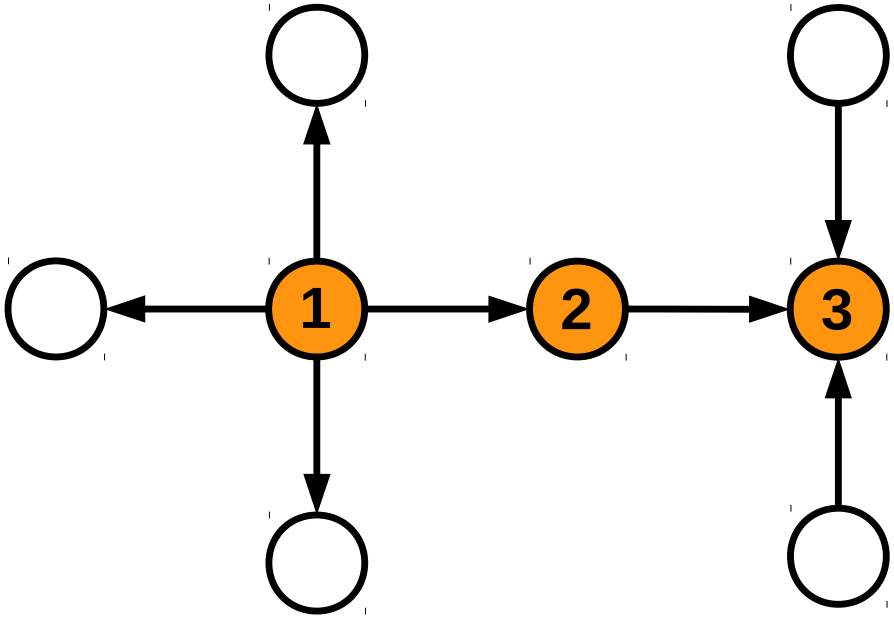
Triple Pattern Count is not Enough…

- Given a number of triple patterns, we can define multiple structures



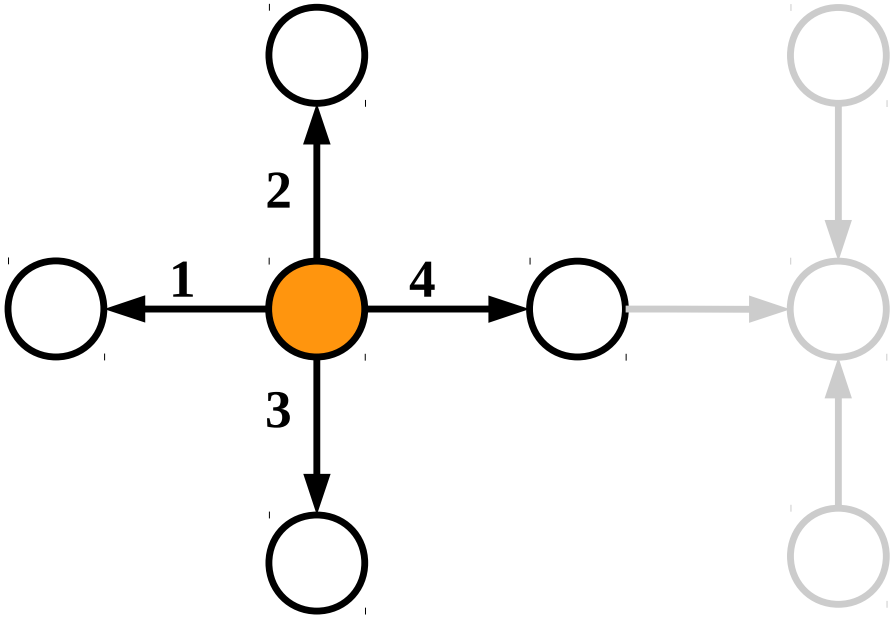
Structural Features

[Join Vertex Count]



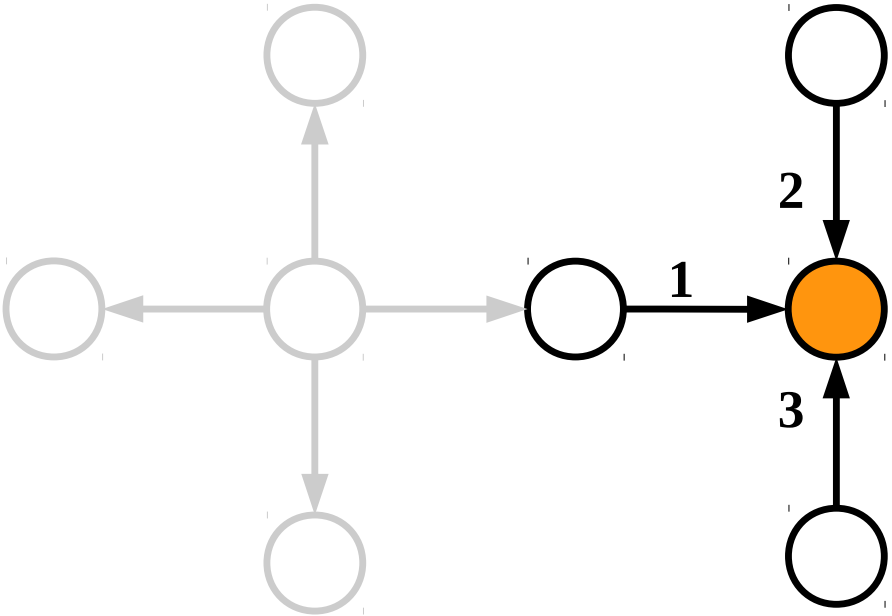
Structural Features

[Join Vertex Degree]



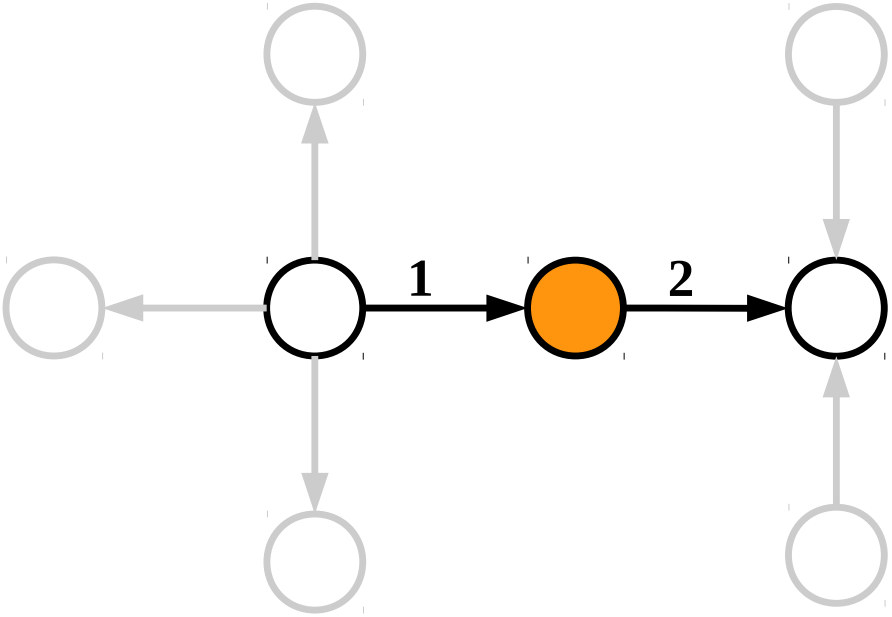
Structural Features

[Join Vertex Degree]

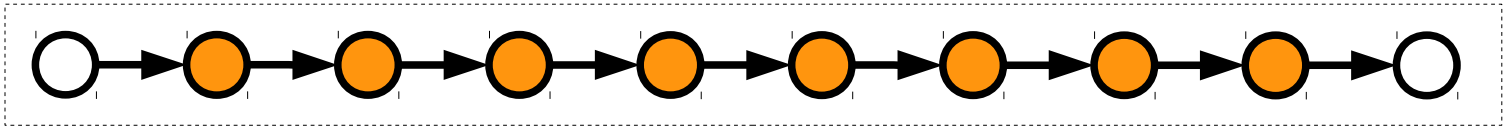


Structural Features

[Join Vertex Degree]

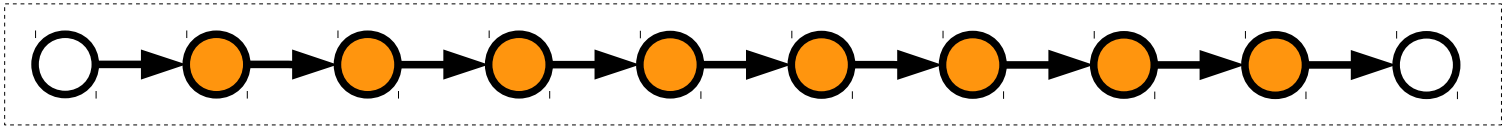


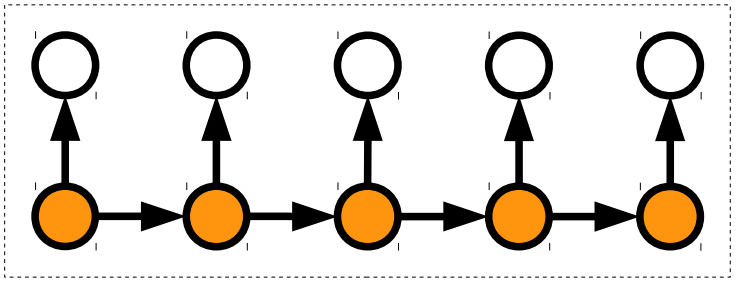
Structural Features



Join Vertex Count	Mean Join Vertex Degree
8	*2.0*

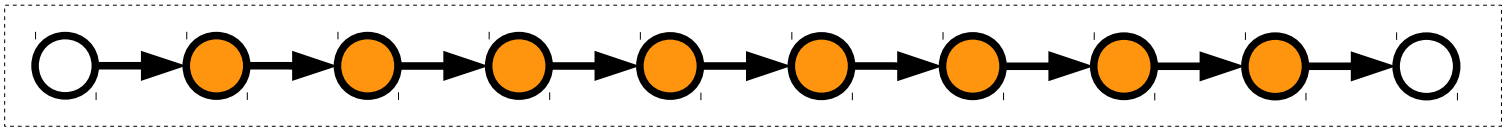
Structural Features

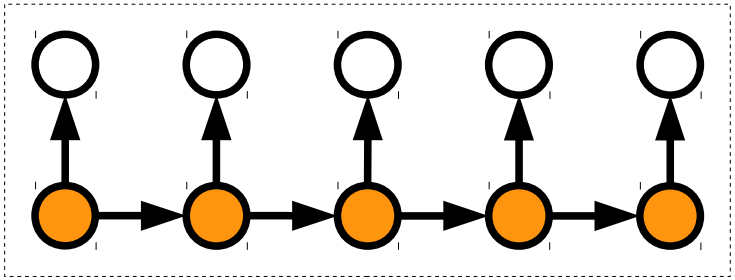


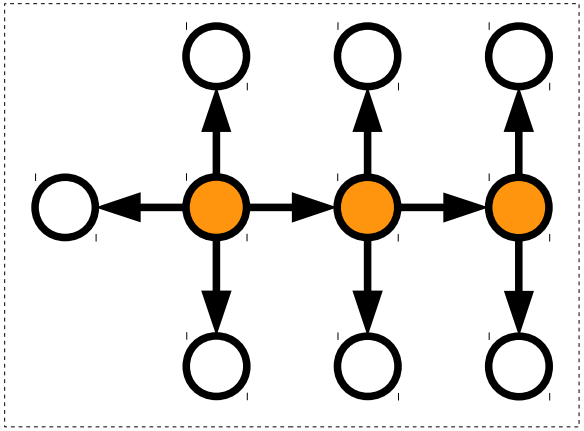


Join Vertex Count	Mean Join Vertex Degree
8	*2.0*
5	***2.6***

Structural Features

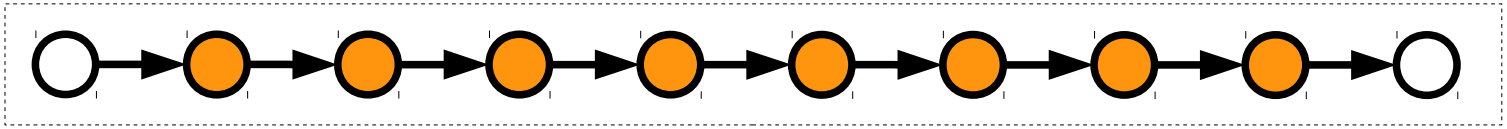


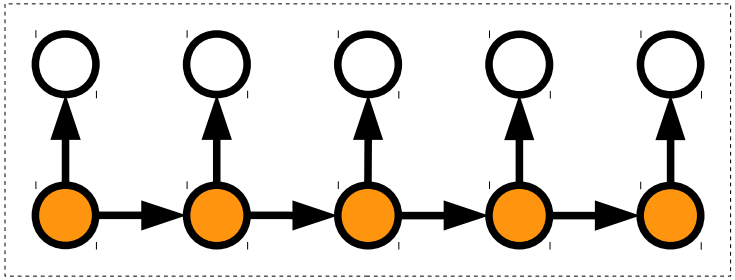


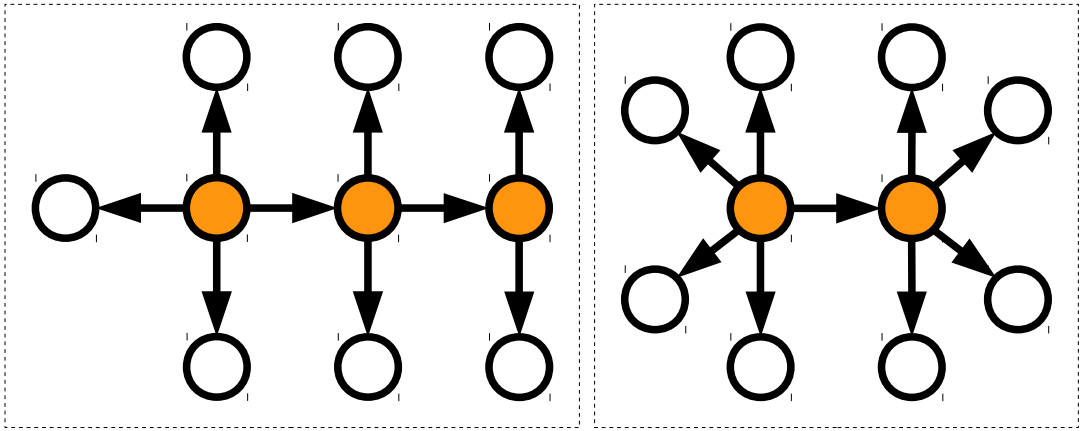


Join Vertex Count	Mean Join Vertex Degree
8	*2.0*
5	*2.6*
3	***~3.7***

Structural Features

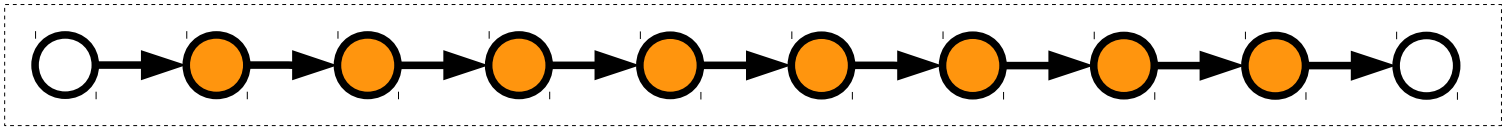


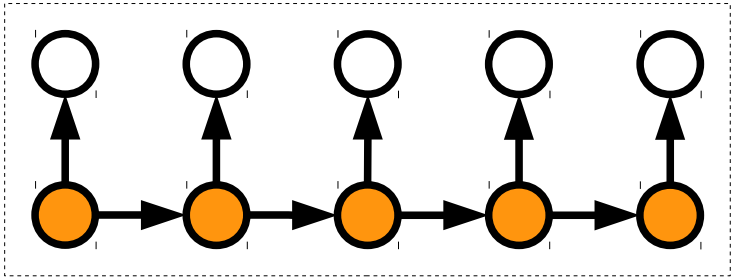


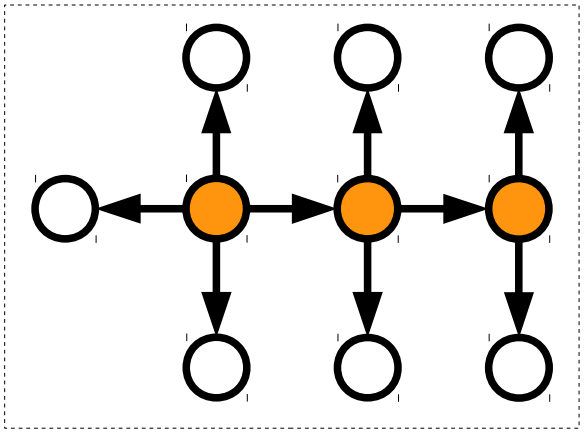


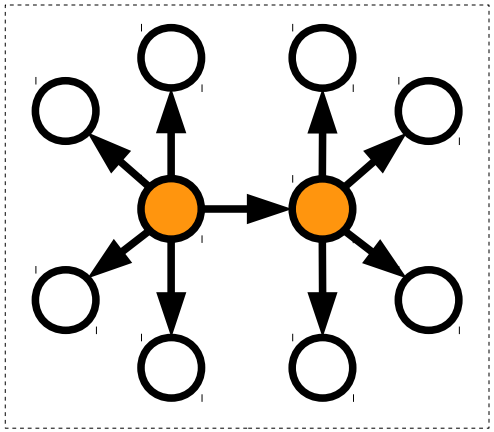
Join Vertex Count	Mean Join Vertex Degree
8	*2.0*
5	*2.6*
3	*~3.7*
2	***5.0***

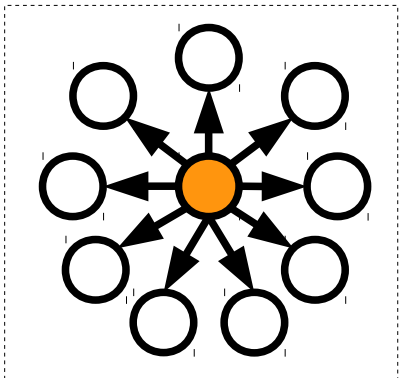
Structural Features







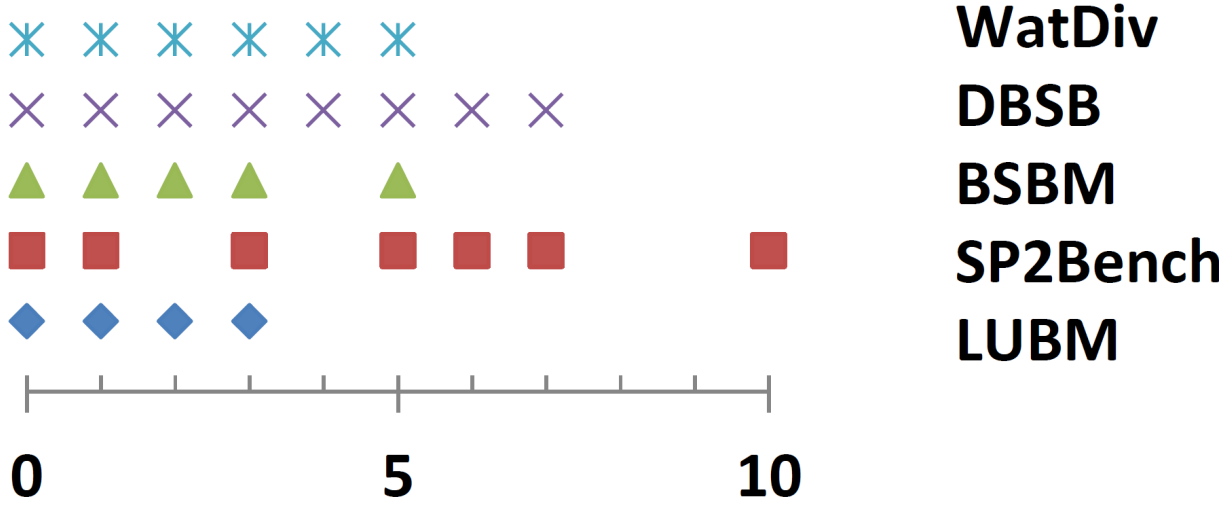




Join Vertex Count	Mean Join Vertex Degree
8	*2.0*
5	*2.6*
3	*~3.7*
2	*5.0*
1	***9.0***

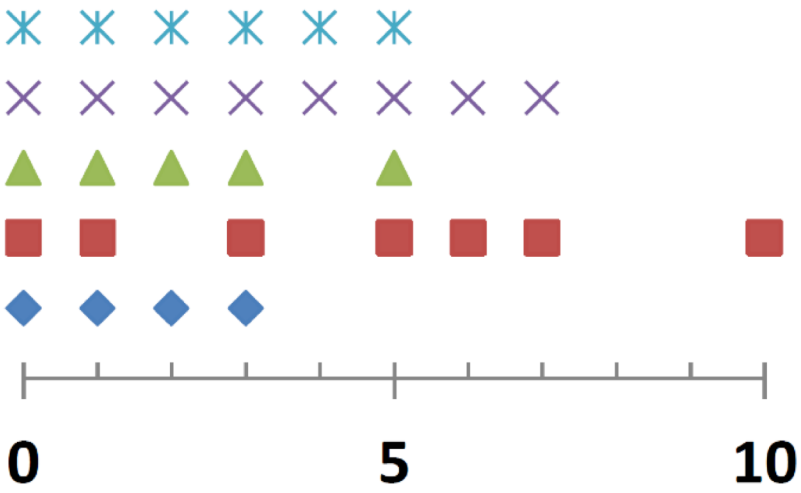
How Diverse are SPARQL Benchmarks?

[Join Vertex Count]



How Diverse are SPARQL Benchmarks?

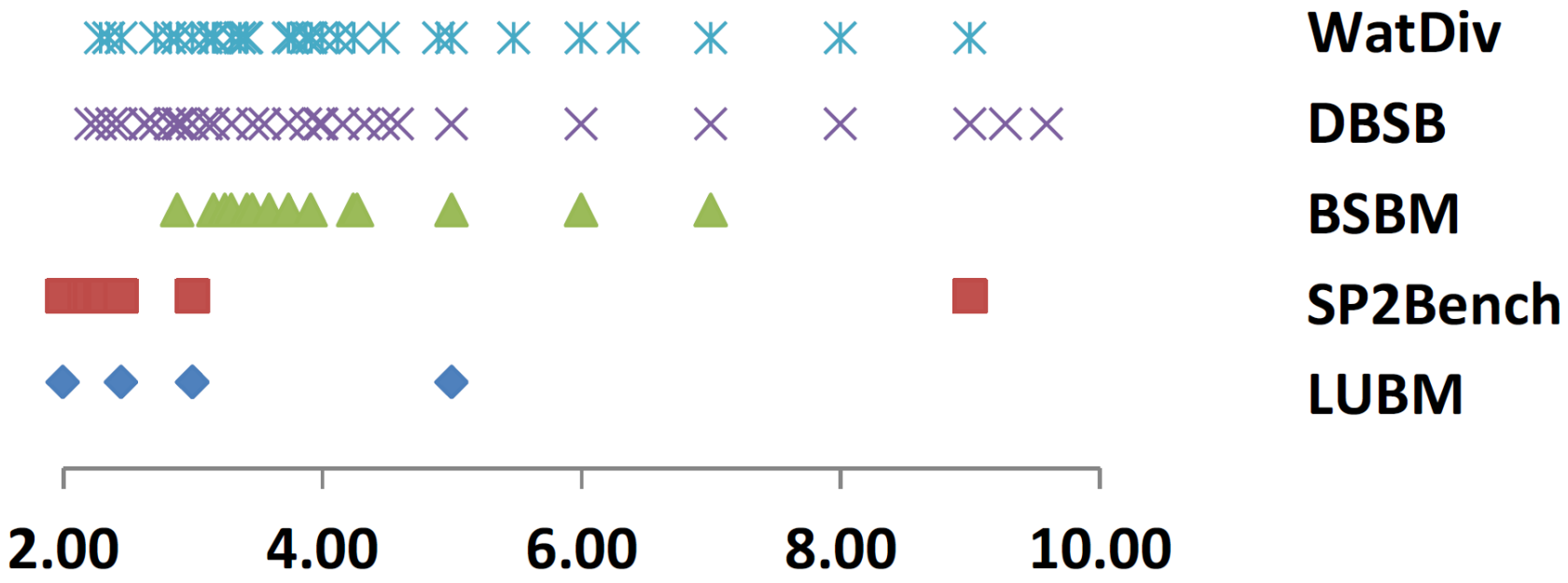
[Join Vertex Count]



WatDiv
DBSB
BSBM
SP2Bench
LUBM

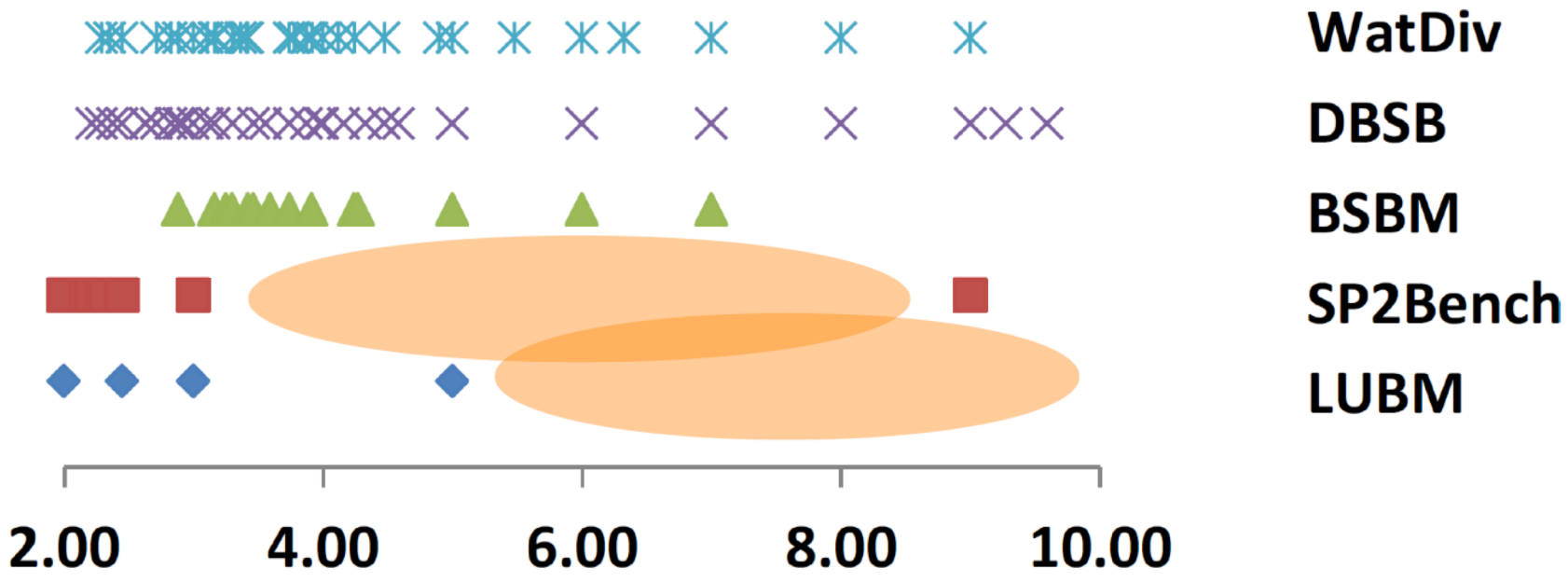
How Diverse are SPARQL Benchmarks?

[Join Vertex Degree – mean]



How Diverse are SPARQL Benchmarks?

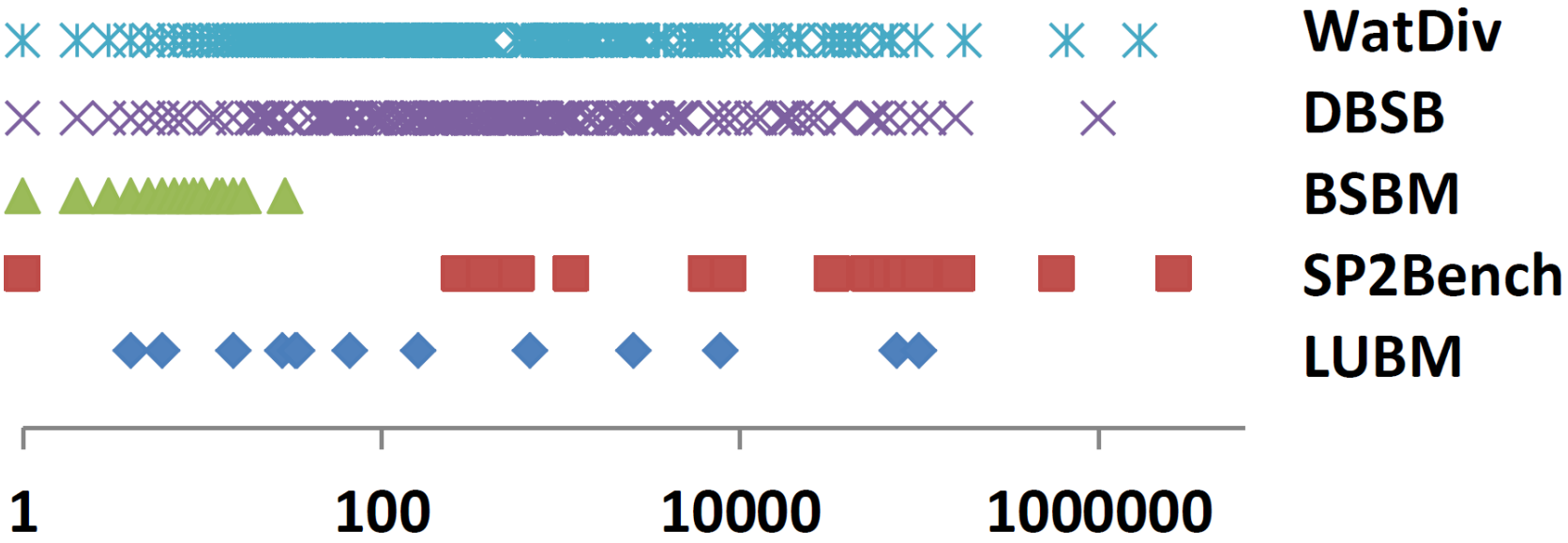
[Join Vertex Degree – mean]



- What about other interesting measures ?

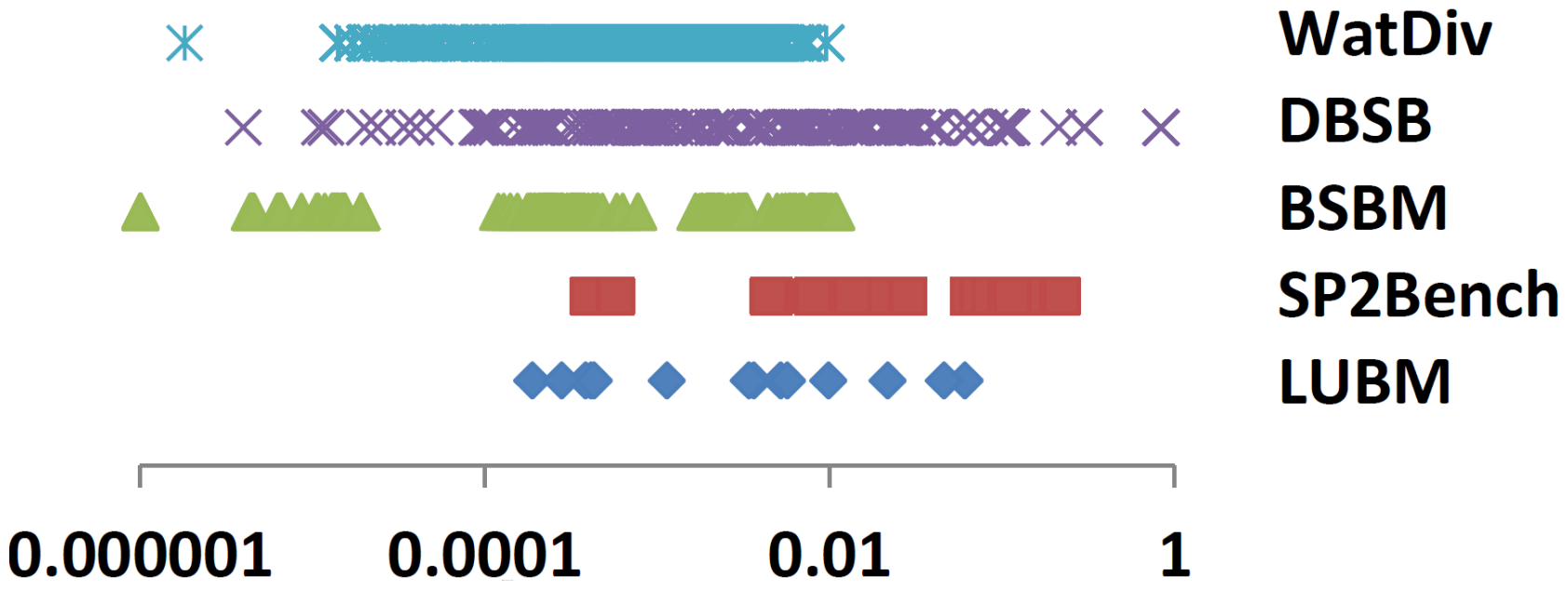
How Diverse are SPARQL Benchmarks?

[Result Cardinality]



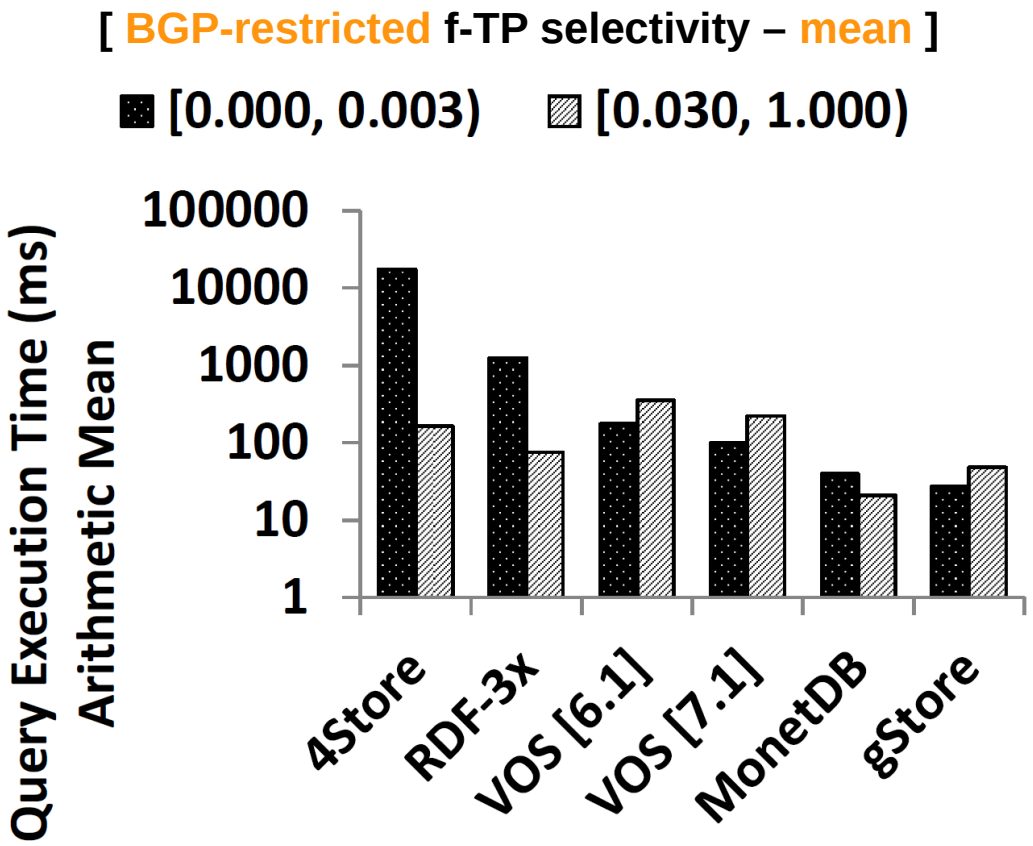
How Diverse are SPARQL Benchmarks?

[f-TP Selectivity – mean]



How Robust are Systems across WatDiv Workloads?

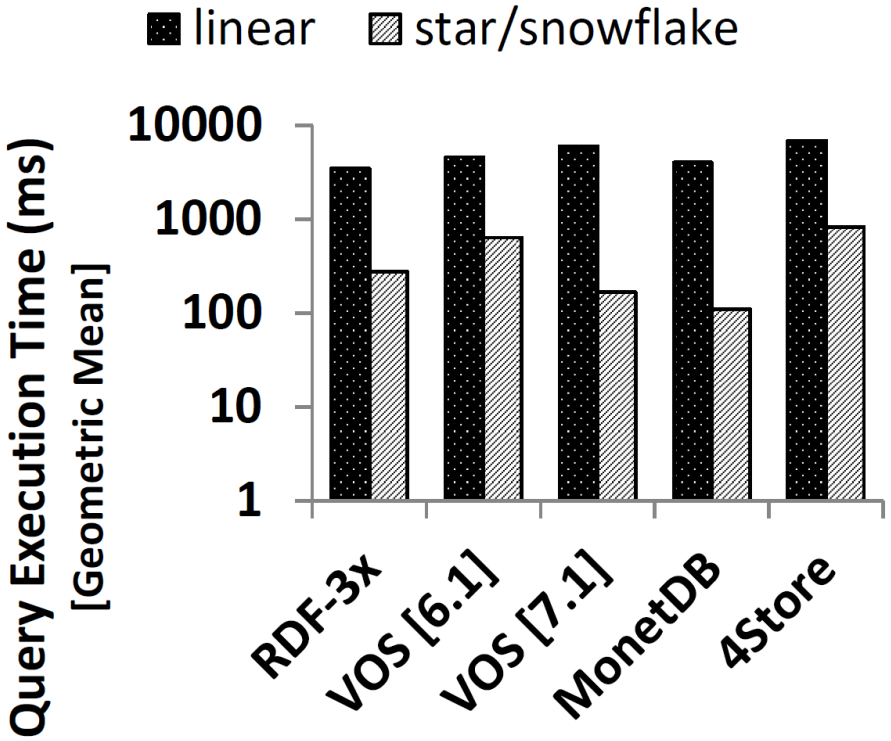
GOOD NEWS



WatDiv 100M triples, queries w/ single join vertex, result cardinality ≤ 2000

How Robust are Systems across WatDiv Workloads?

GOOD NEWS



WatDiv 10M triples

linear = { mean join vertex degree ≤ 3.0, join vertex count ≥ 3}

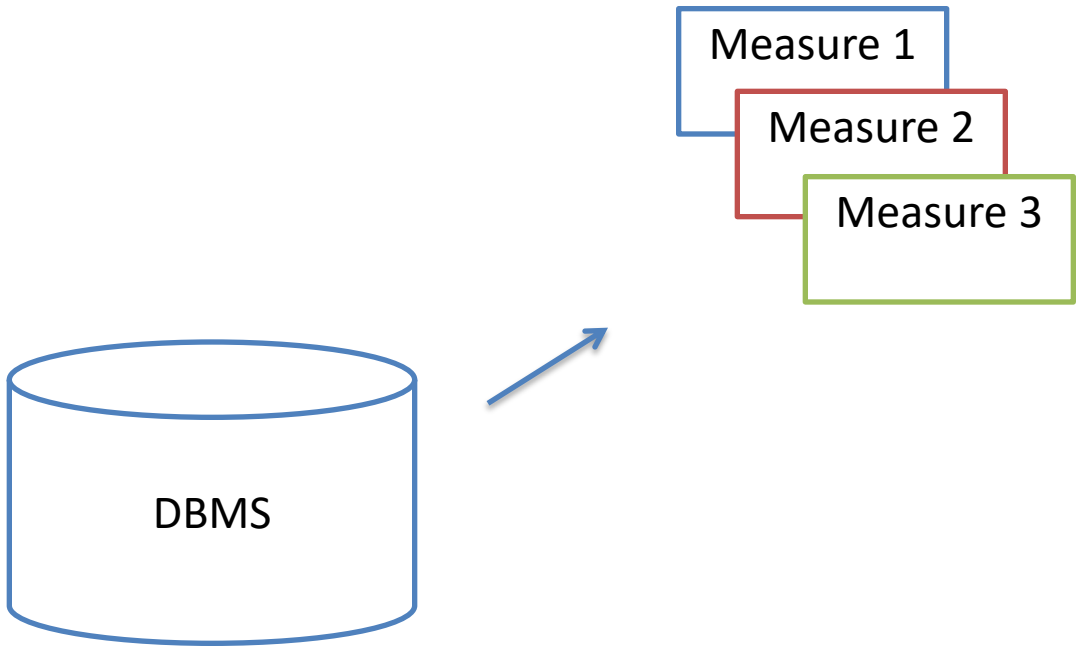
star/snowflake = { mean join vertex degree ≥ 5.0, join vertex count ≤ 2}

Standard Benchmarks : Summing Up

- Performance analysis needs standard benchmarks (microbenchmarks are limited, real data is rare)
- Impossible to make 100% fair comparison of two systems.
  - Every benchmark bests suits a system
- Most important :
  - Understand the reliability of the tests you choose
  - Understand what the test can really tell you about the system
    - Are you testing a limit/rare or typical case ?
    - Are you covering several important configurations or not ?

CHOOSING THE METRICS

Performance Evaluation in Databases

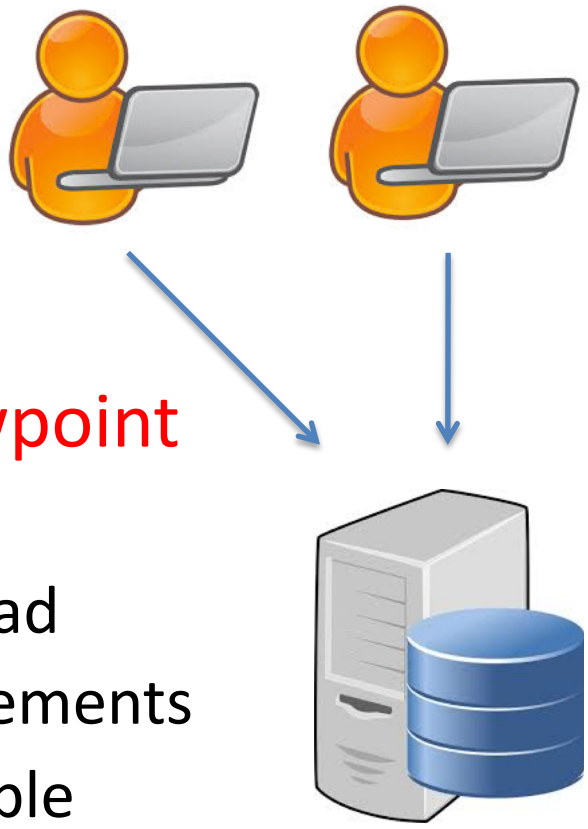


Metrics

“Standards of measurement by which efficiency, performance, progress, or quality of a plan, process, or product can be assessed”

Database Metrics : for whom ?

- Performance to an end-user
  - means response time
  - means quality of the answers



- Performance from a system viewpoint
  - means throughput
  - and capability to handle a given load
  - memory & storage usage & requirements
  - do more memory and storage enable scale-up (size of inputs) / speed-up (time reduction) ?

Database Metrics

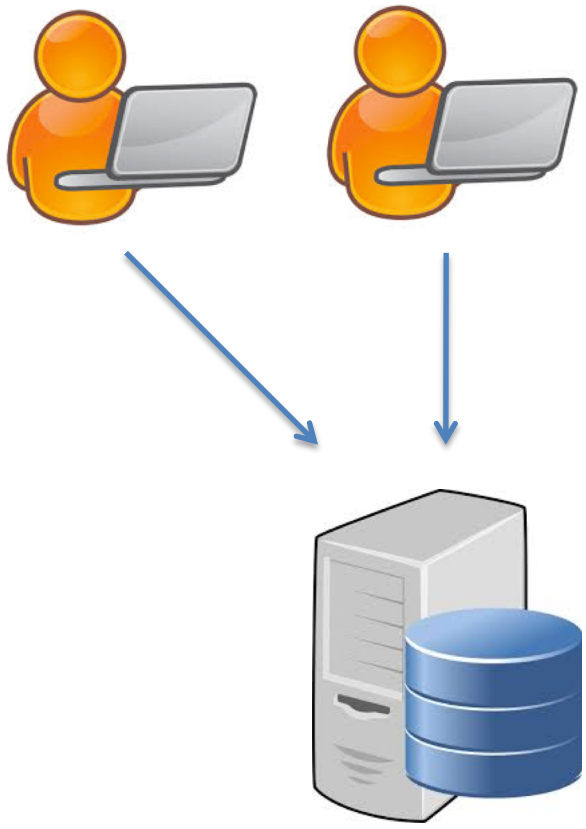
1. **Response time** : The time to get the answer to a query

2. **Quality of answers** : the % of correct and wrong solutions

1. **Throughput** : The number of queries that can be run in any time period.
   - e.g. Queries per second

2. **Usage/Requirements** : resources consumed by system (memory and storage size)

3. **Scalability** :
   - degradation of response time with more data / users
   - reduction of answer time with more ressources



Database Metrics

3 **Quality of answers** (to be measured against a trusted system O, the “oracle”)

- % **Soundness :** fraction of answers for our system that are also answers for O
- % **Completeness :** fraction of answers for O that are also answers for our system
- Incomplete and (even worse) unsound systems may be useless according to the domain application.

Project : Quality of answers

- **Quality of answers**
  - In your project : to be measured against Jena
  - You have to implement a soundess and completennes-check test in your project, for your execution times to make sense !

- **What do we want to avoid ?**
  - *Ex : fast response time with 30% less answers (incompleteness)*
  - *Unsound systems are even worse that in*

Matter of Time

- **Evaluation time**
  - wall-clock (“real”) CPU (“user”) I/O (“system”)
  - Server-side vs. client-side

Unix Time (User, Sys, Real)

- **User** is the amount of CPU time spent in user-mode code (outside the kernel) within the process.
  - When a program loops through an array, it is accumulating user CPU time.

- **Sys** is the amount of CPU time spent in the kernel within the process.
  - When a program executes a system call such as exec or fork, it is accumulating system CPU time

- Both are actual CPU time used in executing the process.

Unix Time

- **Real** time, is time from start to finish of the call.
  - This is all elapsed time including time slices used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete).

- The total CPU time (user time + sys time) may be *more* or *less* than real time.
  - A program may spend some time waiting.
  - On a multicore system, the user and/or sys time (as well as their sum) can actually exceed the real time.

MEASURING TIME

Mistake #1 : printing in system output

- Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk
- Benchmark : TPC-H (sf = 1)
- MonetDB/SQL v5.5.0/2.23.0

Q	server user	server real	client real	client real	result size	... time (milliseconds)
1	2830	3533	3534	3575	1.3 KB	
16	550	618	707	1468		

Beware of what you measure !

- Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk
- Benchmark : TPC-H (sf = 1)
- MonetDB/SQL v5.5.0/2.23.0

	server		client			
	user	real	real	real	result	... time (milliseconds)
Q	file	file	file	terminal	size	output went to ...
1	2830	3533	3534	3575	1.3 KB	
16	550	618	707	1468	1.2 MB	

Metrics : Beware of what you measure

1/11/2016

“[…] malheureusement je ne peux pas laisser le programme en marche car mon laptop a des problèmes de ventilation et ne supportera pas le processus pour une longue durée.

*Le programme mettra environ **23** heures. ”*

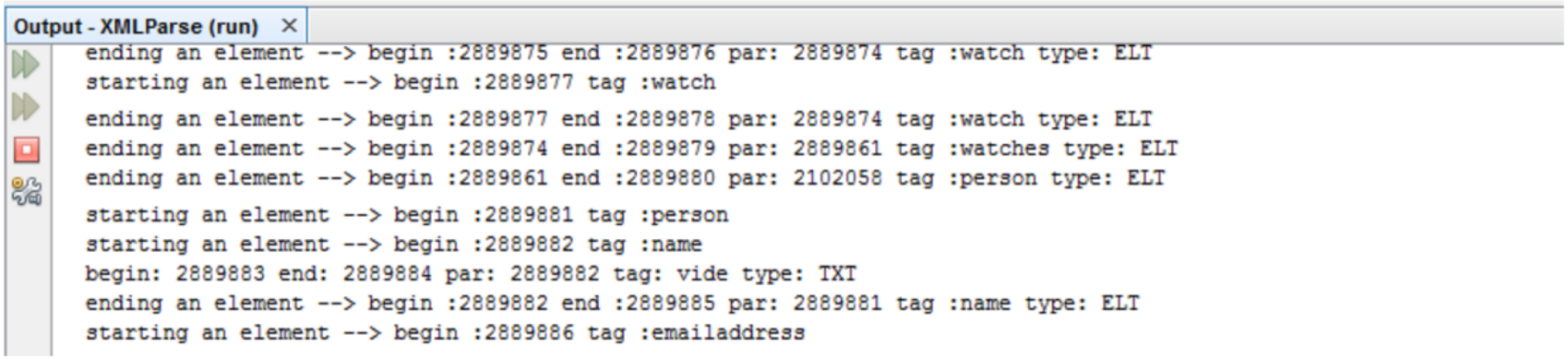
“[…] Traitement de 242641 balises en 45 minutes et intérrompu avant la fin. Ce temps de traitement démontre qu’il doit exister des pistes d’amélioration de l’efficacité du programme. ”

- Temps d'éxécution: Plus de 4h30
  Début de l'algorithme: 22h38
  Fin de l'algorithme: 3h26 et l'algorithme tournait encore

Metrics : Beware of what you measure

03/11/2015

“Mon pc est entrain de parser le document depuis le soir du 01/11/2015, suis-je sur le bon chemin ?”



Time without printing : in the order of seconds !
Size of the written output : several hundreds of MB

Time – Database Perf. Analysis

- Is almost always on seconds
- Use milliseconds if really needed
- Avoid nanoseconds or smaller units
- **If your query takes less than 1ms on average,** consider measuring the treatment of a whole workload of **100/1000/10000** queries (this reduce statistical errors)

Statistical Estimators

Round	Query time
1	22 ms
2	10 ms
3	16 ms
4	16 ms
5	15 ms
6	13 ms
7	553 ms
8	10 ms
9	5 ms
10	8 ms

Statistical Estimators

Round	Query time
1	22 ms
2	10 ms
3	16 ms
4	16 ms
5	15 ms
6	13 ms
7	**553** ms
8	10 ms
9	5 ms
10	8 ms

Avg = 66.8ms

AvgR = 13.75

Use Robust Mean

- Take 5 (or 7, or 10) measurements
  - 1 may be enough for dictionary creation…
  - …but not for queries
- Discard lower and higher values (Robust Mean)
- Average on the remaining values

Result Characterization

- **Arithmetic Mean** (caution, not always the best estimator)

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

AVG{0,10} = 5 = AVG{4,6}

- Add **variance** if result-set has large variability

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2$$

Result Characterization

- **Geometric Mean**

$$\left(\prod_{i=1}^{n} a_i\right)^{\frac{1}{n}} = \sqrt[n]{a_1 a_2 \cdots a_n}.$$

- A metric based on an **arithmetic** mean of query execution times may be dominated by the longest running queries.
- To assure that a query does not skew the results, a geometric mean is considered (small values more influent).

Execution time cannot be zero!

(it is always strictly greater than zero)

10 ms

1 ms

0 ms

2 ms

0 ms

0 ms

system.nanoSec() is inaccurate (time may be too short to be measured correctly!)

system.currentTimeMillis() can give you 0ms ! **Always round these values to 1ms**

Execution time cannot be zero!

(it is always strictly greater than zero)

10 ms	10 ms
1 ms	1 ms
0 ms	**1** ms
2 ms	2 ms
0 ms	**1** ms
0 ms	**1** ms

system.nanoSec() is inaccurate (time may be too short to be measured correctly!)

system.currentTimeMillis() can give you 0ms ! **Always round these values to 1ms**

HOW THE SYSTEM ENVIRONMENT INFLUENCES MEASUREMENTS : WARM VS COLD MEMORY

“We run all experiments in warm memory”

What to choose ? Warm vs Cold

- Depends on what you want to
  show /
  measure /
  analyze

- No formal definition, but “common sense”

Cold Run

- A cold run is a run of the query right after a DBMS is started and no (benchmark-relevant) data is preloaded into the system's main memory, neither by the DBMS, nor in file system caches.

- Such a clean state can be achieved via a system reboot or by running an application that accesses sufficient (benchmark-irrelevant) data to flush file system caches, main memory, and CPU caches.

Warm Run

- A hot run is a run of a query such that as much (query-relevant) data is available as close to the CPU as possible when the measured run starts.

- This can (e.g.) be achieved by running the query (at least) once before the actual measured run starts.

Cold vs Warm

- When a **cold** measure is interesting ?
  - to understand the worst case cost of a process
  - to understand tasks done once (eg. dictionnary)

- When a **warm** measure is interesting ?
  - to understand the average/real cost of a process
    - think of a query asked many times (i.e. keyword search)

Hot vs Warm & User vs. Real Time

- Laptop: 1.5 GHz Pentium M (Dothan), 2 MB L2 cache, 2 GB RAM, 5400 RPM disk
- TPC-H (sf = 1)
  MonetDB/SQL v5.5.0/2.23.0

Q	cold		hot		... time (milliseconds)
	user	real	user	real	
1	2930	13243	2830	3534	

Be aware *what* you measure!

«Cold» query time has simply no sense

	Temps 1 "Cold"	Temps 2 "Warm"	Temps 3 "Warm"	Temps 4 "Warm"	Temps 5 "Warm"
X	$1.5016ms$	$1.4775ms$	$1.3883ms$	$1.3558ms$	$1.3866ms$
Y	$1.0892ms$	$0.9892ms$	$1.1584ms$	$1.0261ms$	$0.97ms$
Z	$0.6388ms$	$0.5578ms$	$0.5476ms$	$0.4967ms$	$0.5459ms$

Also note that here times are essentially constant ! (up to fractions of millisenconds)

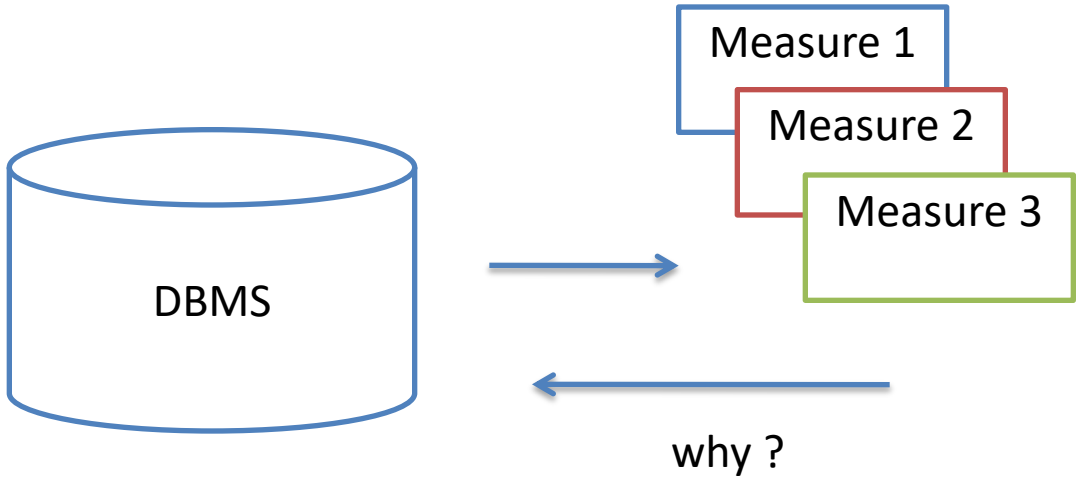
In this case, there was a mistake on the cold measurement that were in reality all warm

SUMMING UP ON MEASURING TIME

A reliable WARM experiment for query answering

1. Load all data
2. Load all queries in an Array
   - of java/c++ objects ; be sure you have a bit more of memory for this
3. Warm up the system with 30-50% random queries
4. Choose a permutation of the query Array
5. Evaluate the workload measuring start-end time

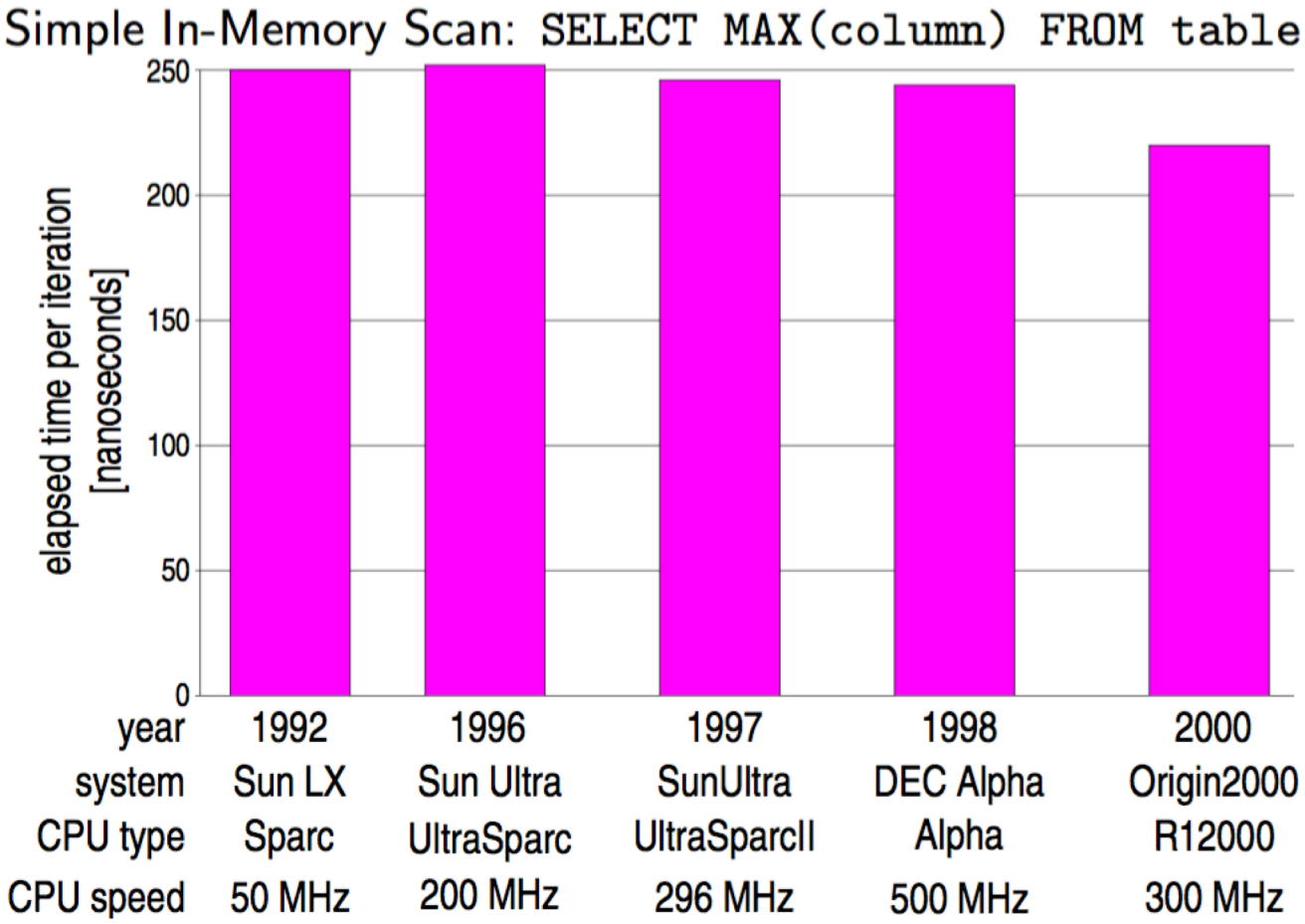
ANALYZING THE DATA CAN BE DIFFICULT, EITHER


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?

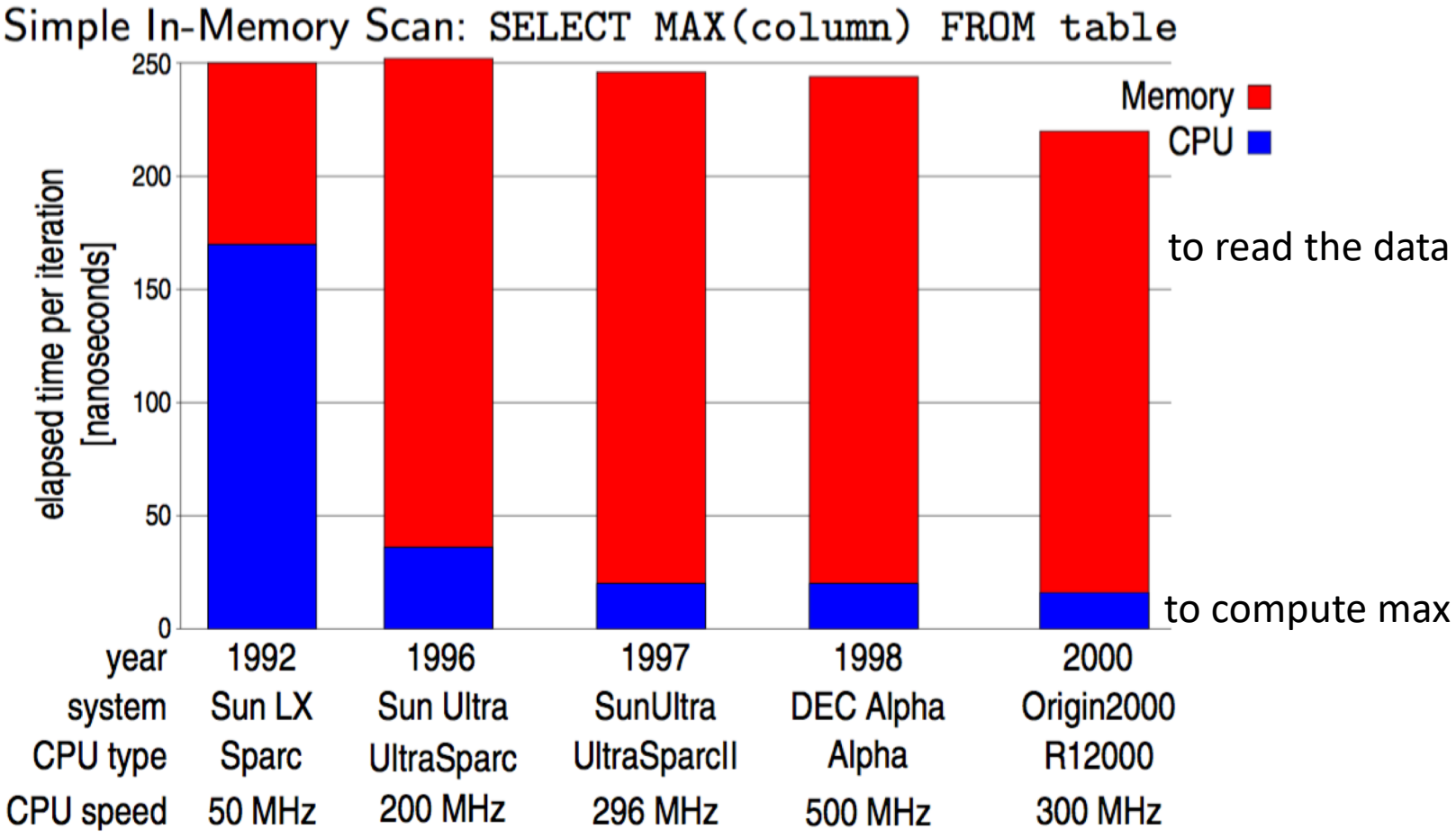


iteration = readinga tuple and compute max

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 […]”