Systems Paper and Reproducible Experiments

Research Methods

Associate Professor Vincent Gramoli

https://gramoli.github.io

THE UNIVERSITY OF SYDNEY

We recognise and pay respect to the Elders and communities – past, present, and emerging – of the lands that the University of Sydney's campuses stand on. For thousands of years they have shared and exchanged knowledges across innumerable generations for the benefit of all.

THE UNIVERSITY OF SYDNEY

Syllabus

- What is a Systems Paper?
- Scientific Writing
- Reproducible Experiments
- Final Recommendations

What is a Systems Paper?

Systems Paper

- Simplified (naive?) view of computer science research
  - Theory: formal model, proofs, lower bounds, mathematic analysis, etc.
  - System: concrete implementation of software/system, empirical evaluation, etc.

- Systems does not only mean “operating” systems but also
  - Networked systems
  - Distributed systems
  - Database systems
  - …

- A “systems paper” is a scientific article that relates to this notion of system

Structure

- Abstract
- Introduction
    - context, motivation, background, contributions

("Problem" is sometimes in a separate Section)

- Solution in details
    - We implemented a system to solve the problem
- Evaluation/Results
    - Evaluation indicates that the solution actually works
- Related work
- Conclusion
    - Summary: you can repeat yourself once more
    - Future work: what is missing from your paper, new directions

Writing order:

1. Evaluation/Solution
    - Do not loose time: once you have an idea, try to validate it
    - If it works, you have something
2. Conclusion
    - Write down what you have, what this actually means
3. Related work
    - Explain what others did on the topic, where the difference is
4. Introduction
    - Summarized the above
5. Abstract
    - Summarized the introduction
6. Title
    - Only now should you be able to express you idea in few words

Abstract and Introduction

Abstract

Purpose: summarize the paper

- For the Program Committee (PC)
  - PC members usually bid on papers whose title+abstract interest them
  - Abstracts can be adjusted to target specific PC members (as the PC is public)
  - Carefully choose (key)words (avoid ambiguities with other fields)

- For you
  - To rapidly convince the reader there is something great in this paper
  - This paper has something new
  - This will have impact
  - The authors did a serious work

- It is ok to rewrite the abstract for a different audience once the paper is accepted

Abstract

Content

- What?
  - Explain the context
  - Make sure a researcher in computer science can identify the field of research
- Why?
  - Briefly explain the motivation of your work
  - Why it is relevant to the research community
- Results
  - Explain your achievements
  - Describe the impact this will have if you can

Abstract

- Avoid the generic boring abstract
  - P is very challenging problem that is not solved yet
  - This paper presents the system S to solve the problem P
  - We show experimentally that S outperforms S'

- Follow the same guidelines, but be specific:
  - Identify a common point of all existing approaches (except yours)
  - Prove that yours is novel precisely because of this distinction
  - Name one/two key ingredients: this is no magic, it works because of ingredient A
  - How much is it better than others? In what circumstances is it not better?

- Reader should learn something, e.g., "I did not expect A to be effective"

Introduction

The most important part of the paper!

1. Describe the **context** of the paper
   - Start with simple explanations
   - Then with technical content

2. **Motivate** the problem
   - Explain why it is relevant
   - Mention why it is not solved already

3. Identify a **gap** in the literature
   - Talk about the background, key related work
   - Why they do not solve your problem

4. State your **contributions**
   - Outline approach (it should fill the gap)
   - What ingredients allow you to solve the problem
   - Results

Introduction

No “Bait-and-switch”!

- Do not claim you solved everything
  - Be honest
  - It is normal for a solution to have limitations and/or assumptions
  - Try to emphasize the trickiness of the problem solved

- Solve all the problems you emphasized
  - Each single problem mentioned at the beginning should be solved, if not:
  - Narrow down the problem or
  - Complete your results, the paper is not ready

- Otherwise, reviewers would kill your paper

Introduction

*Review: "This paper presents a novel and **technically interesting solution** to an important problem, with **excellent results** shown through **rigorous studies**."*

No "Bait-and-switch"!

- Do not claim you solved everything
  - Be honest
  - It is normal for a solution to have limitations and/or assumptions
  - Try to emphasize the trickiness of the problem solved

- Solve all the problems you emphasized
  - Each single problem mentioned at the beginning should be solved, if not:
  - Narrow down the problem or
  - Complete your results, the paper is not ready

- Otherwise, reviewers would kill your paper

*Review: "The **message of the paper seems confused**: is it about transactions generally, or about polymorphism?"*

Scientific Writing

Scientific style

implementation of elastic transactions in these settings. Our evaluation on four benchmarks, i.e., a linked list and a hash table data structures as well as a bank and a MapReduce-like applications, indicates better scalability than locks and up to 20-fold speedup (relative to bare sequential code) when running 24 application cores.

Write scientifically!

- Top-down description
  - Starts by mentioning the idea, then describe it
  - ≠ bottom-up, e.g., novels where the reader discovers things progressively
  - It applies to the whole article, to a single section and to a paragraph

- Quantify
  - No superlatives: “very”, “high”, “significant”, “very high”, “super cool”…
  - Measure as much as possible: 10% higher, twice slower, super linear…
  - “Here are our three contributions”, “there are four results on the topic”

Scientific style

- Repeat yourself
  - About three times, for the reader to remember
  - Intro: we will talk about “bla”, Body: “bla” (Conclusion: we talked about “bla”)
  - It applies to the whole article, to a single section and to the abstract

- Structure your ideas
  - One key idea per paragraph, if not split the idea or the paragraph
  - Use linking words (“in addition”, “as a result”, “for example”…)
  - At most one contradiction “but”/”however”/”by contrast” per paragraph
  - First sentence of the paragraph should be a summary of the idea
  - Reading the 1st sentence of each paragraph must be enough to get the idea

Scientific style

Repetition example [EuroSys'12]

- Abstract
- Introduction
- Section 3
  (after Intro.&Pb.)
- Conclusion

This paper presents TM^{2}C, the first software Transactional Memory protocol for Many-Core systems. TM^{2}C ex-

We present in this paper TM^{2}C, the first TM system tailored for non-coherent many-core processors. TM^{2}C capitalizes the low latency of on-die message passing by being

3. TM^{2}C, Transactional Memory for Many-Cores

This section introduces the first *Transactional Memory for Many-Cores* (TM^{2}C). TM^{2}C allows programmers to exploit

10. Conclusions

We have proposed TM^{2}C, the first transactional memory for many-cores, the family of processors that promise to reconcile high performance with low energy consumption at the cost of trading hardware cache-coherence for message passing. TM^{2}C exports the standard transactional interface hid-

[Gramoli, Guerraoui, Trigonakis, TM2C, a Software Transactional Memory for Many-Cores. *EuroSys'12.*]

Scientific style

Paragraph example

- 1st sentence summary
"Evaluation on SCC"

- 2nd sentence details
"What SCC is"

- 3rd sentence details
"Evaluation"

- 4/5th sentence details
"More evaluations"

- 6th sentence details
"It was useful: we conjecture"

We evaluate TM^{2}C on the Intel®'s Single-chip Cloud Computer (SCC), a non-coherent message passing processor. The SCC is a 48-core experimental processor relying on a 6 × 4 two-dimensional mesh of tiles (two cores per tile) that is claimed to be "arbitrarily scalable" [29]. On a hash table data structure and a MapReduce example application TM^{2}C performs up to 20 and 27 times better than the corresponding bare (non-transactional) applications running on a single core. We also evaluated the importance of fair contention management by comparing our FairCM scheme with alternative ones on various workloads. Particularly, FairCM performs up to 9 times better than the others on a workload with a single core running long conflict prone transactions. Last but not least, we also elaborate on the portability of TM^{2}C to cache-coherent architectures. We show that TM^{2}C is also efficient on multi-cores and we conjecture on the possible causes of performance difference when running it on multi-cores and many-cores.

Scientific style

Think as the reader!

- Help the reader understanding your paper
  - Do not expect the reader to be an expert in your field
  - Do not think you can hide stuff from reviewers either
  - Define any keyword, put in italic the main ones first time you define them

- Get your paper accepted for publication
  - Program committee (PC) discussions are about convincing one another
  - In good conferences, you need someone to champion your paper
  - Make sure someone will like your paper
  - Give this reviewer some arguments to defend your paper

Scientific style

Definition example
- Many-core definition

2. The Many-Core Model

We consider a *many-core*, a processor that embeds from tens to thousands of simpler cores than a multi-core to maximize overall performance while minimizing energy consumption [6]. The backbone of the many-core is the network-on-

- This is not a new “buzzword” we came up with. Actually, a respectable researcher from Intel introduced it.

[6] Shekhar Borkar. Thousand core chips: a technology perspective. In *DAC*, pages 746–749, 2007.

Style

*Review: "The paper is nicely structured and written (I identified some **typos**, however)"*

Be diligent!

- Write in engaging style
  - Use active voice
  - Use present tense whenever possible (avoid future)
- Wording
  - Make sure you are clear and concise
  - Reuse same words again (as opposed to a novel)
- Proof-read
  - Spell-check, remove ambiguities, clarify and make the writing concise
  - Get native speaker to proof-read if you are not
  - Get outsider to read it, great to spot weaknesses

Evaluation

Evaluation

- Show that your solution actually works
  - Improvements in important situations
  - Slight or no degradations in other cases
  - Anticipate criticisms and test corner-cases where solution could fail

- Make sure your results are correct
  - Be honest!
  - Repeat benchmarks, take average, median…
  - Make sure standard deviation is low (even if you do not show it in the paper)

- Justify any choice you make (unless it makes perfect sense)
  - We tested on “up to four threads” because we only have “four cores”
  - We used Facebook traces (needless to say that these are realistic)

Evaluation

*Review: "This was a very interesting paper, revealing high technical sophistication, and with careful, **thorough** and **honest** experimental numbers".*

*Review: "I found the **lack of comparison** to Distributed Shared Memory concerning."*

- Show that your solution actually works
  - Improvements in important situations
  - Slight or no degradations in other cases
  - Anticipate criticisms and test corner-cases where solution could fail

- Make sure your results are correct
  - Be honest!
  - Repeat benchmarks, take average, median…
  - Make sure standard deviation is low (even if you do not show it in the paper)

- Justify any choice you make (unless it makes perfect sense)
  - We tested on "up to four threads" because we only have "four cores"
  - We used Facebook traces (needless to say that these are realistic)

Evaluation

Present your results

- Do not waste space
  - Figures can often be made smaller and convey the same idea
  - Use placeholders for figures, do not wait until you have all figures ready
  - Print graphics to make sure legends are readable (even on B&W printers)

- Follow formatting rules
  - Do not use small fonts, double-check appearance of pseudocode and figures
    - Some conferences have online checkers (check early)
  - Do not play with margins, space between lines (\baselineskip)
    - This is not clever, do you teach your grandma how to suck eggs?
    - It looks like you did not spend enough time (being concise is time-consuming)

Evaluation

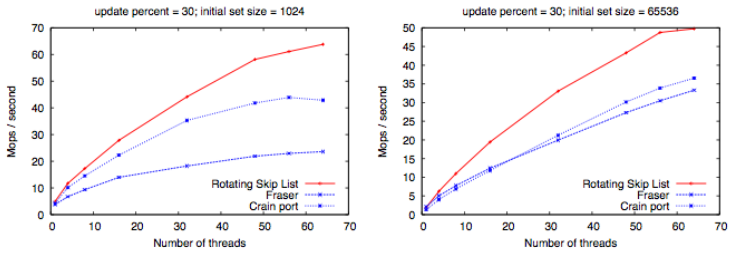


Fig. 7. The Rotating skip list does not suffer from contention hot spots

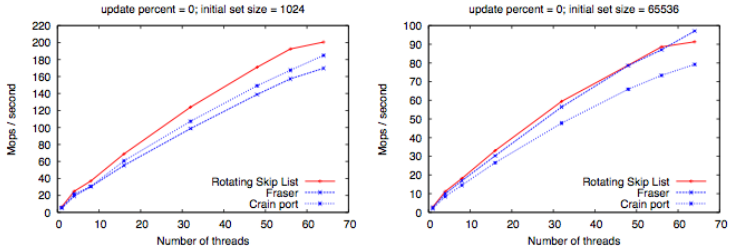


Fig. 8. The cache locality of the Rotating skip list boosts read-only workloads

update	0%	10%	30%
no hotspot	32,863K (0.28‰)	36,804K (0.45‰)	38,382K (0.50‰)
rotating	26,541K (0.24‰)	29,680K (0.36‰)	27,061K (0.36‰)
decrease	19%	19%	29%

TABLE II
UNLIKE THE NO HOTSPOT SKIP LIST, THE ROTATING SKIP LIST LIMITS THE CACHE MISSES ON LARGE DATASETS (2^{16}) CACHE MISS RATES ARE GIVEN IN PARENTHESIS

that initially populate the structure with (1024) values that are within a subrange ([0, 1024]) of the admitted range ([0, 32768]) of values.

We tested the performance of the three algorithms when running this biased workload, having each of the 64 threads inserting/deleting with probability 10% by picking uniformly a value within the range [0,32768]. **[V: Is it really the true that 64 threads insert/remove on all skip list algorithms?]** The result is that most inserts executed during the experiment fall to the right of the pre-existing towers, unbalancing the

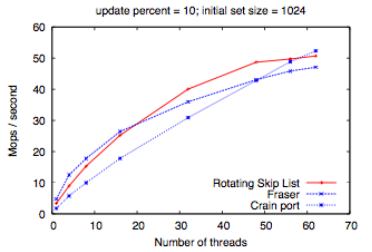


Fig. 9. The Rotating skip list rebalances effectively under biased workloads

Before

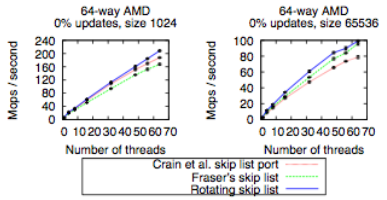


Figure 6. The data locality of the rotating skip list boosts read-only workloads

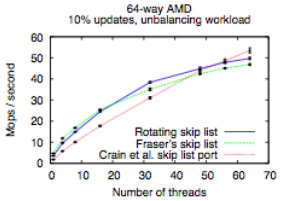


Figure 7. The rotating skip list rebalances effectively under biased workloads

presents lower performance than the rotating one. As no remove operations execute, this difference is expressed by the way nodes are represented in memory. The rotating skip list represents a wheel as an array so that nodes that are above each other get recorded in contiguous memory locations. The main advantage is that multiple nodes can be fetched from memory at once, when a single cache line is stored in the cache. Later, when the same thread traverses down the structure, accessing a lower node will likely produce a cache hit. Instead, Crain's skip list uses a linked structure to represent towers, so that traversing down the list requires to fetch disjoint memory locations that will not likely be in the cache already.

6.4 Cache miss rate

To confirm that the rotating skip list leverages caches more effectively than Crain's skip list, we measured the amount of cache misses. Table 3 depicts the number of cache misses (in parentheses), and the cache miss rate (that is, the proportion of cache misses out of all accesses) observed while running Crain's no hotspot skip list and the rotating skip list on a key-value store with 2^{16} elements. We can see that the rotating skip list decreases the cache miss rate substantially when compared to the no hot spot skip list, under each update rate. This confirms the impact of cache efficiency on performance we conjectured in the former sections.

update	0%	10%	30%
Crain port	(29,834K) 0.39‰	(35,457K) 0.83‰	(41,027K) 1.07‰
rotating	(28,061K) 0.29‰	(26,258K) 0.37‰	(24,094K) 0.38‰

Table 3. Compared to Crain's skip list, the rotating skip list limits the cache miss rate on datasets of size 2^{16} (absolute cache miss numbers in parentheses)

6.5 Stressing the maintenance thread

To reduce contention it is important that a single thread does the maintenance. To stress this aspect, we measure the performance under a skewed workload that unbalances the structure. We have extended Synchrobench with an additional -U parameter. This initially populates the structure with (1024) keys that are within a subrange ([0, 1024]) of the admitted range ([0, 32768]); application threads are then inserting/deleting with probability 10% by picking uniformly a key within the range [0, 32768]. The impact is that most inserts executed during the experiment fall to the right of the pre-existing towers, unbalancing the skip list.

Figure 7 depicts the performance results: it is clear that a single maintenance thread in the rotating skip list is enough to raise the new towers inserted by the application threads. In particular, the rotating skip list provides performance superior to Fraser's for all thread counts larger than or equal to 32. Although all three skip lists give similar performance, Crain's gives slightly higher performance for more than 56 threads, probably because it is the only one that does not support memory reclamation.

7. Extra Synchronizations and Structures

We also compared the performance of the rotating skip list against two different logarithmic data structures, a binary search tree and another balanced tree, as well as lock-based and transaction-based skip lists.

Synchronization techniques. Our lock-based skip list implements the Optimistic skip list algorithm [16]. Our transaction-based skip list uses a classic skip list [28] whose accesses were protected using elastic transactions as offered by the $\mathscr{E}$-STM software transactional library. (A mentioned previously, due to the limited L1 (data) cache size (32KiB) of the Intel Haswell processor we could not use hardware transactions.) Elastic transaction is a relaxed transaction model that was shown particularly promising on pointer-based structures [10], like skip lists.

Figure 8 depicts the performance of the lock-based skip list, the transaction-based skip list, Fraser's and the rotating one on the Intel machine. (We omitted the skip list port from Crain et al. as it has no memory reclamation.) We ran the same binary of the rotating skip list we tested on the AMD machine in Section 6. An interesting phenomenon is the vari-

After

Reproducible Experiments

Why Reproducibility?

nature

Explore content | About the journal | Publish with us | Subscribe

nature > special > key reads

Special | 18 October 2018

Challenges in irreproducible research

Science moves forward by corroboration – when researchers verify others' results. Science advances faster when people waste less time pursuing false leads. No research paper can ever be considered to be the final word, but there are too many that do not stand up to further study.

There is growing alarm about results that cannot be reproduced. Explanations include increased levels of scrutiny, complexity of experiments and statistics, and pressures on researchers. Journals, scientists, institutions and funders all have a part in tackling reproducibility. Nature has taken substantive steps to improve the transparency and robustness in what we publish, and to promote awareness within the scientific community. We hope that the articles contained in this collection will help.

Software

- Use version control system
  - Even for a single-author paper
  - Apache subversion (SVN)
  - Concurrent versions system (CVS)
  - …
- Shared documents
  - Avoid MS Word, MS researchers use LaTeX
  - Google docs: everyone sees your changes (no local copy)

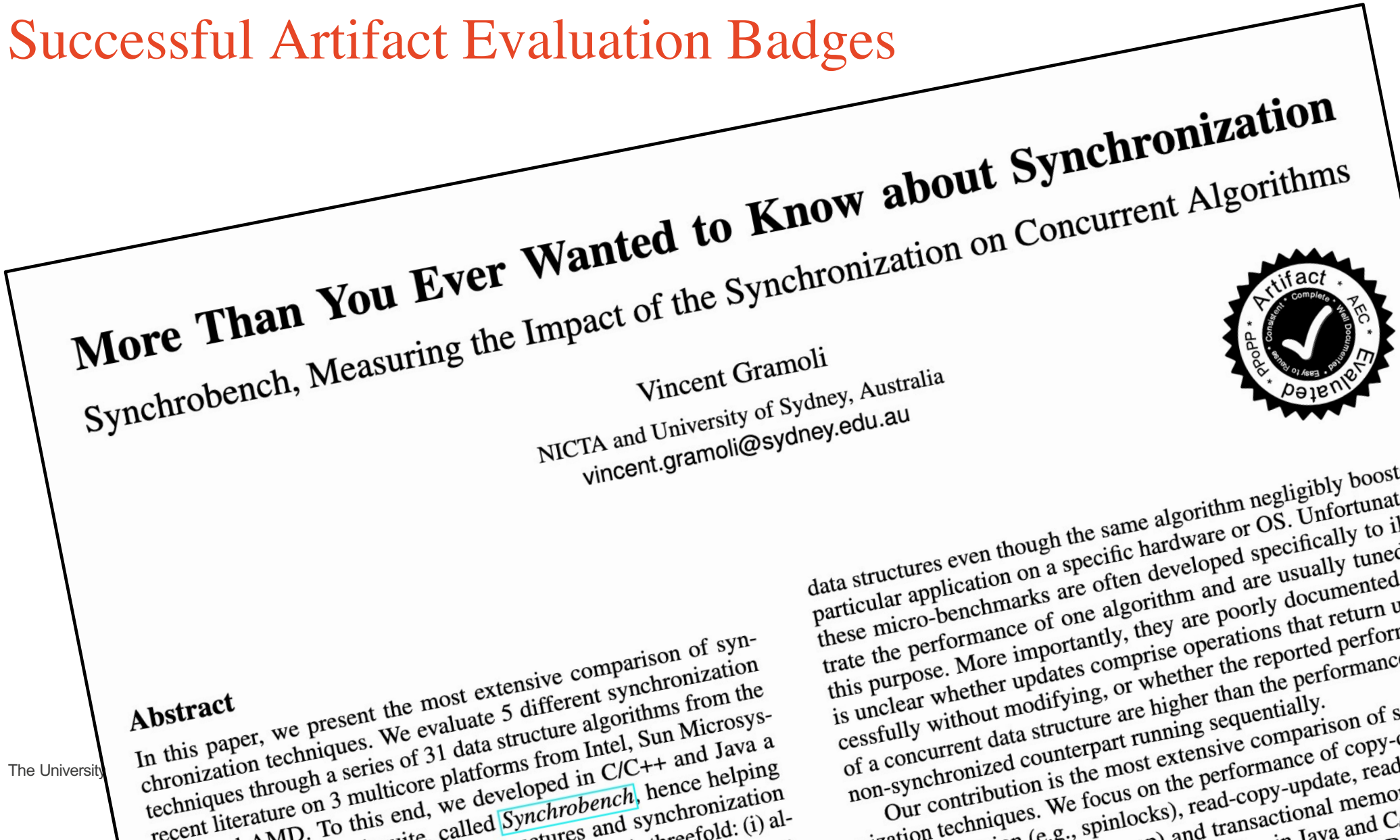
Artifact Evaluation

Originally coming from the Programming Language community, now the Systems community propose Artifact Evaluation Committee in addition to Program Committee at their conferences.

The Artifact Evaluation Committee evaluates the material used to produce experimental results of papers accepted to be published in the proceedings of the conference.

If successful, the Artifact Evaluation Committee add badges on the paper.

Successful Artifact Evaluation Badges



More Than You Ever Wanted to Know about Synchronization

Synchrobench, Measuring the Impact of the Synchronization on Concurrent Algorithms

Vincent Gramoli
NICTA and University of Sydney, Australia
vincent.gramoli@sydney.edu.au

Abstract

In this paper, we present the most extensive comparison of synchronization techniques. We evaluate 5 different synchronization techniques through a series of 31 data structure algorithms from the recent literature on 3 multicore platforms from Intel, Sun Microsystems and AMD. To this end, we developed in C/C++ and Java a benchmark suite, called *Synchrobench*, hence helping data structures and synchronization techniques. Our study is threefold: (i) algorithms even though the same algorithm negligibly boosts a particular application on a specific hardware or OS. Unfortunately, these micro-benchmarks are often developed specifically to illustrate the performance of one algorithm and are usually tuned for this purpose. More importantly, they are poorly documented as it is unclear whether updates comprise operations that return unsuccessfully without modifying, or whether the reported performance of a concurrent data structure are higher than the performance of its non-synchronized counterpart running sequentially.

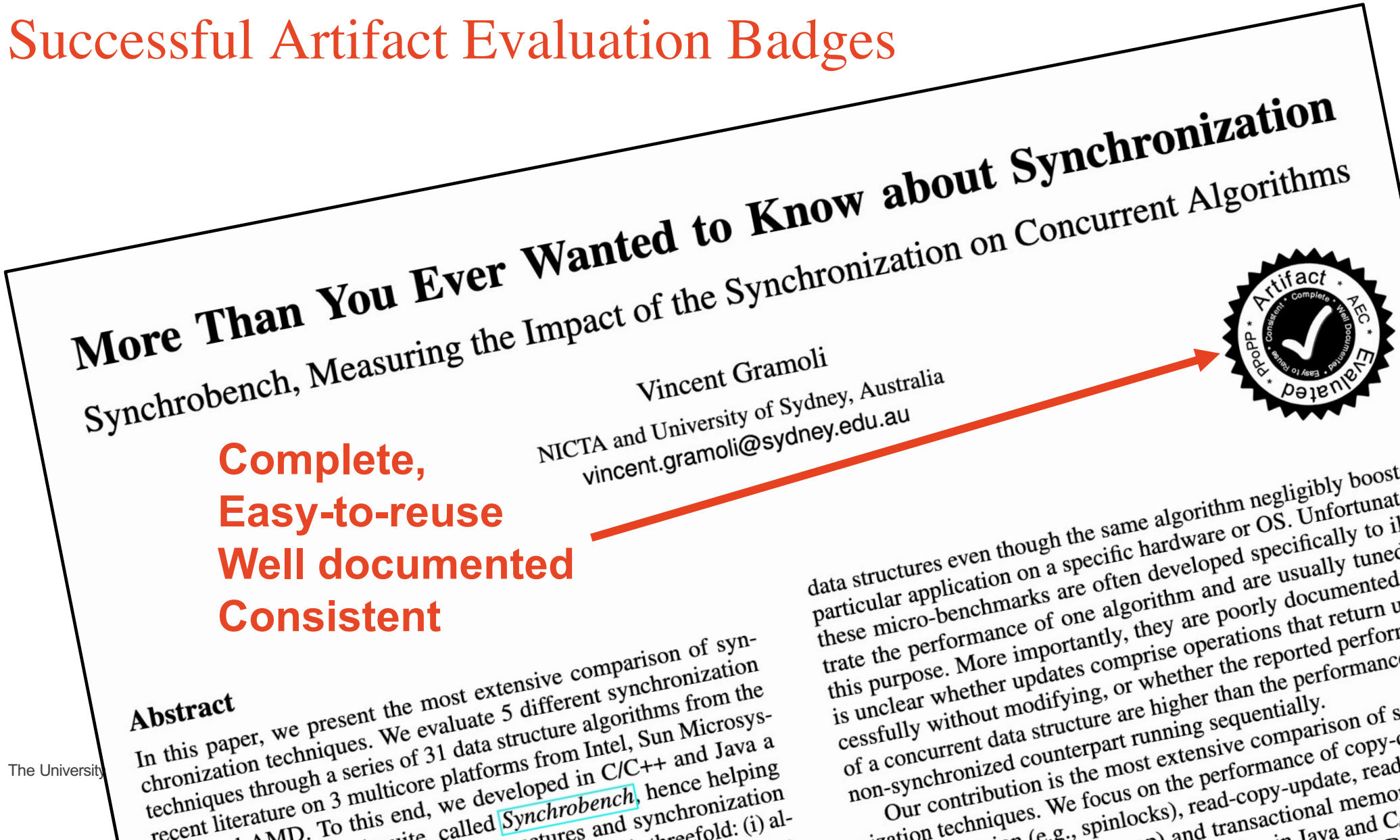
Our contribution is the most extensive comparison of synchronization techniques. We focus on the performance of copy-on-write, (e.g., spinlocks), read-copy-update, read-modify- and transactional memory to synin Java and C/C++, and T2 mul-

ee at

ntal

The University

Successful Artifact Evaluation Badges



More Than You Ever Wanted to Know about Synchronization

Synchrobench, Measuring the Impact of the Synchronization on Concurrent Algorithms

Vincent Gramoli
NICTA and University of Sydney, Australia
vincent.gramoli@sydney.edu.au

Complete,
Easy-to-reuse
Well documented
Consistent

Abstract

In this paper, we present the most extensive comparison of synchronization techniques. We evaluate 5 different synchronization techniques through a series of 31 data structure algorithms from the recent literature on 3 multicore platforms from Intel, Sun Microsystems and AMD. To this end, we developed in C/C++ and Java a benchmark suite, called *Synchrobench*, hence helping the fair comparison of data structures and synchronization techniques. Our study is threefold: (i) al-

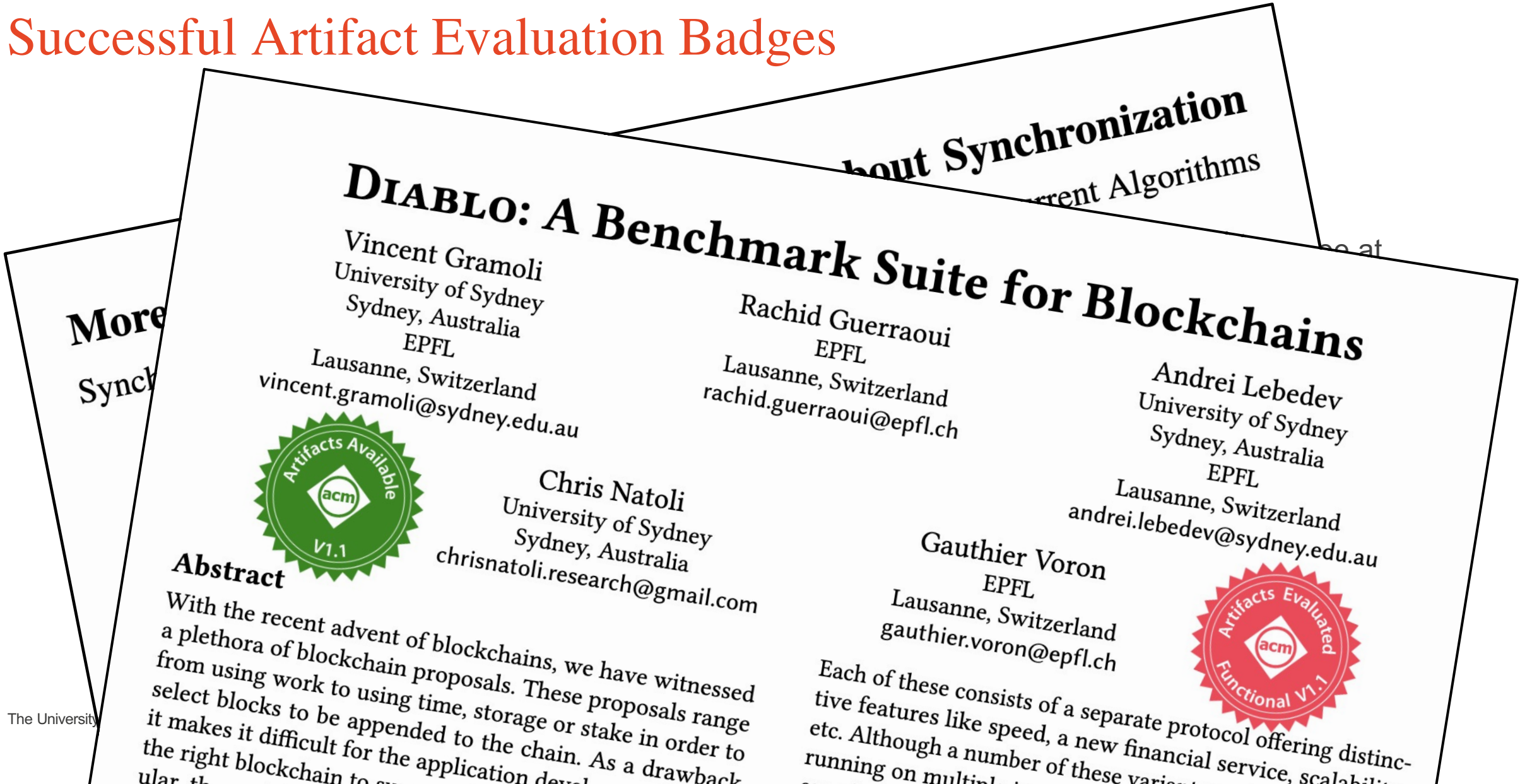
data structures even though the same algorithm negligibly boosts a particular application on a specific hardware or OS. Unfortunately, these micro-benchmarks are often developed specifically to illustrate the performance of one algorithm and are usually tuned for this purpose. More importantly, they are poorly documented as it is unclear whether updates comprise operations that return unsuccessfully without modifying, or whether the reported performance of a concurrent data structure are higher than the performance of its non-synchronized counterpart running sequentially.

Our contribution is the most extensive comparison of synchronization techniques. We focus on the performance of copy-on-write, (e.g., spinlocks), read-copy-update, read-modify- and transactional memory to synin Java and C/C++, and T2 mul-

ee at

ntal

Successful Artifact Evaluation Badges



DIABLO: A Benchmark Suite for Blockchains

Vincent Gramoli
University of Sydney
Sydney, Australia
EPFL
Lausanne, Switzerland
vincent.gramoli@sydney.edu.au

Rachid Guerraoui
EPFL
Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Andrei Lebedev
University of Sydney
Sydney, Australia
EPFL
Lausanne, Switzerland
andrei.lebedev@sydney.edu.au

Chris Natoli
University of Sydney
Sydney, Australia
chrisnatoli.research@gmail.com

Gauthier Voron
EPFL
Lausanne, Switzerland
gauthier.voron@epfl.ch

Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

Abstract

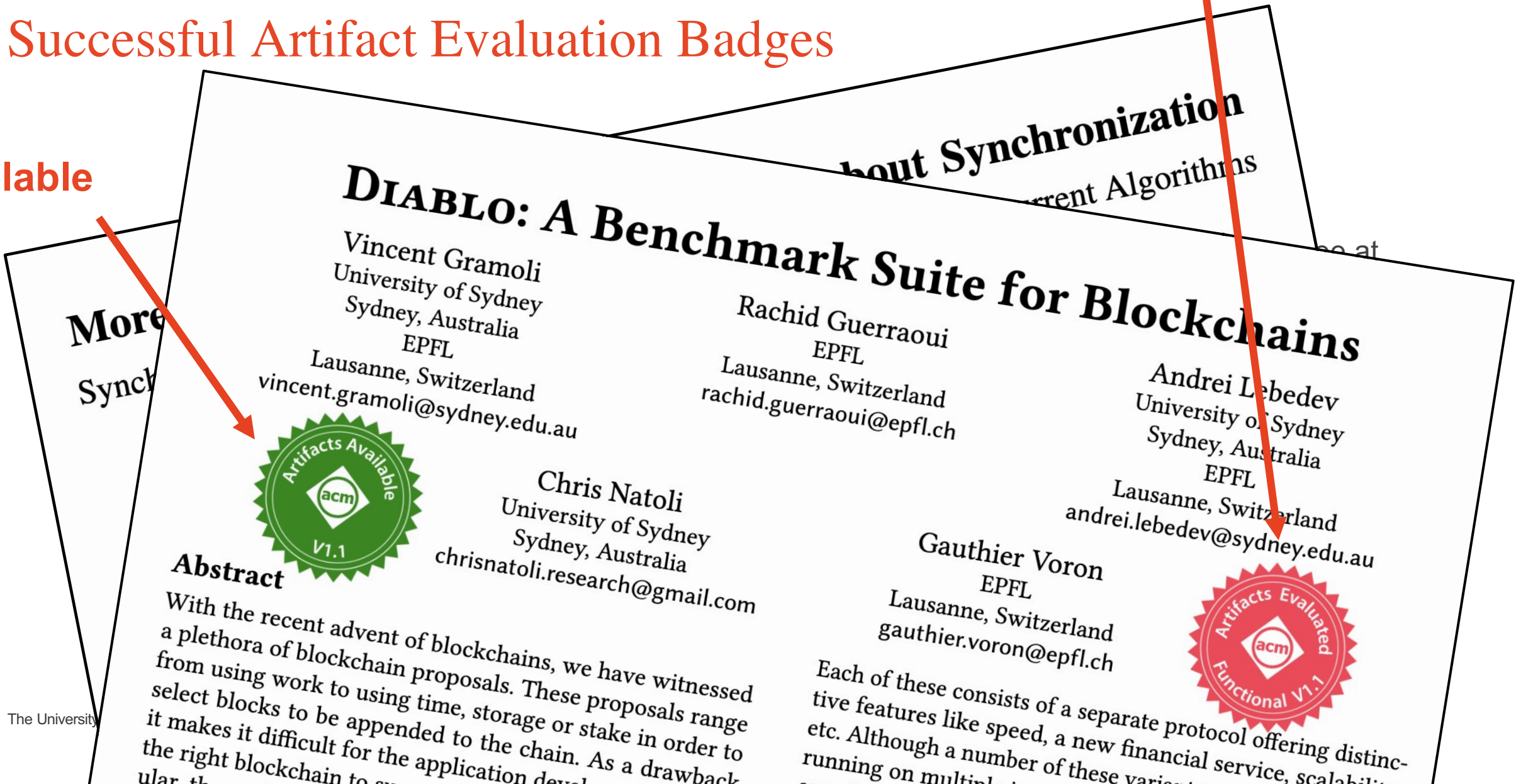
With the recent advent of blockchains, we have witnessed a plethora of blockchain proposals. These proposals range from using work to using time, storage or stake in order to select blocks to be appended to the chain. As a drawback it makes it difficult for the application

Each of these consists of a separate protocol offering distinctive features like speed, a new financial service, scalabi
etc. Although a number of these
running on multi

Successful Artifact Evaluation Badges

Functional

Available



Artifact Evaluation

Systems Conferences (e.g., EuroSys):

- **Available**: Author-created artifacts relevant to this paper have been placed on a publicly accessible archival repository. A DOI or link to this repository along with a unique identifier for the object is provided.
- **Functional**: The artifacts associated with the research are found to be documented, consistent, complete, exercisable, and include appropriate evidence of verification and validation.
- **Reproduced**: The main results of the paper have been obtained in a subsequent study by a person or team other than the authors, using, in part, artifacts provided by the author.

Programming Language Conferences (e.g., PPoPP):

- **Consistent**: does the artifact substantiate and help reproduce the claims of the paper.
- **Complete**: What is the fraction of the results that can be reproduced.
- **Well documented**: Does the artifact describe and demonstrate how to apply the presented method to a new input
- **Easy to reuse:** How easy is it to reuse the provided artifact?

Make your experiments reproducible

Document your experiments:

- Save compile flags (gcc -O3 –std=gnu11)
- Save environment variables (LD_PRELOAD, JAVA_HOME)
- Save runtime options (JVM Hotspot –server –Xoptimize –XX+UseBoundThreads)

Collect raw data before:

- Computing the median, mean
- Computing the standard deviation
- Computing the percentile

Keep the output logs
Keep the visualization scripts used to produce graphs
Use version management system

Recommendations

Aim for high quality venues

Few papers get highly cited but most papers get (almost) not cited.

Citations (and visibility) can be good for your career.

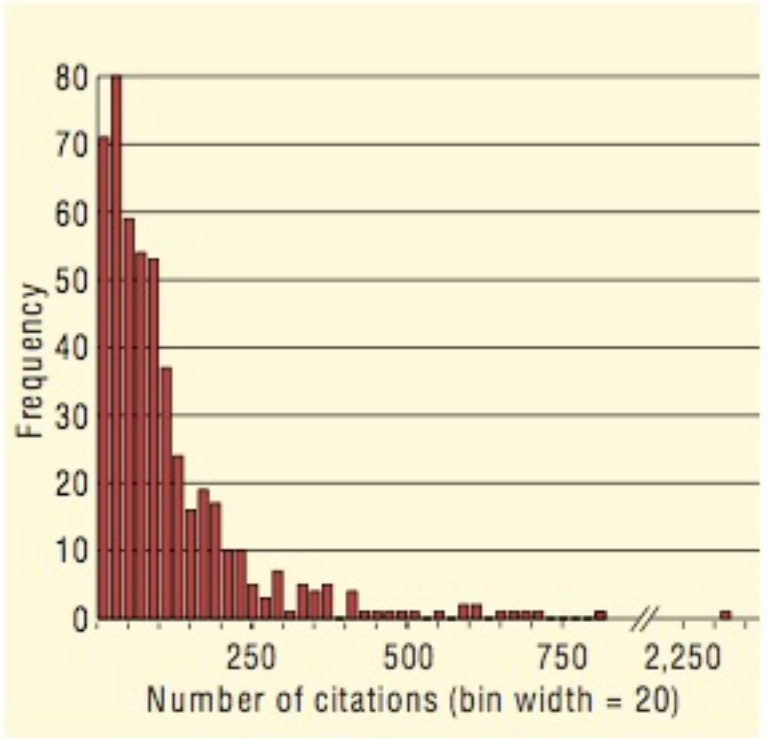
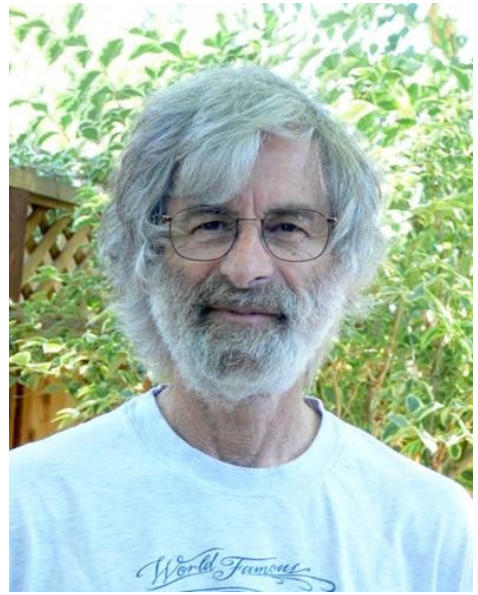


Figure 1 Distribution of the number of citations in five years for 500 biomedical papers published in *Nature*: 100 papers published in each of 1981, 1984, 1988, 1992 and 1996 were chosen at random, and for each paper the number of citations in the subsequent five years was counted. Data provided by Grant Lewison (Department of Information Science, City University, London EC1V 0HB, UK).

Rejection



- Good papers get rejected
  - Sometimes for the wrong reasons
  - Do not blame reviewers, try to explain better

- Even the best papers get rejected
  - Lamport's Paxos took 8 years to get published
  - This is not a failure:
    - Cited 4020 times since it appeared [TOCS'98]
    - Implemented by industrials (Google, IBM, MSR and NICTA)
    - Lamport received the Turing Award 2013 (highest distinction in computer science)

Conclusions

Conclusions

- Do not underestimate the writing time
  - This is crucial for your paper to get accepted (scientific style, reread)
  - Present and motivate the problem
  - Convince the reader that you solve it (repeat yourself, be honest)

- Make your experiments reproducible
  - Make sure no parameter is underestimated (architecture, OS, prog. lang.)
  - Share your dataset
  - Submit an artifact when possible

Additional materials

- An Evaluation of the Ninth SOSP Submissions –or- How (and How Not) to Write a Good Systems Paper. Roy Levin and David D. Redell, 9th SOSP Program Committee Co-chairmen
  https://www.usenix.org/legacy/event/samples/submit/advice.html

- Writing Reviews for Systems Conferences (Timothy Roscoe)
  https://people.inf.ethz.ch/troscoe/pubs/**review-writing**.pdf

- Avoid benchmarking crimes and write a system thesis:
  - http://gernot-heiser.org/benchmarking-crimes.html
  - http://gernot-heiser.org/style-guide.html

- Stories from Leslie Lamport about his own submissions
  http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html


THE UNIVERSITY OF
SYDNEY