

# How Should We Think About Persistent Data Structures?

Michael L. Scott

University of Rochester

Rochester, NY, USA

[mlscott@acm.org](mailto:mlscott@acm.org)

 <https://orcid.org/0000-0001-8652-7644>

## ABSTRACT

For machines with nonvolatile memory (NVM) but volatile caches, most work on persistent data structures has assumed that every operation must be guaranteed to survive any crash that occurs after returning to the caller. Most programmers, however, don't want to persist existing transient structures: they want to avoid serializing and deserializing structures traditionally kept in block-structured files and databases. For these, programmers are accustomed to *buffered* semantics, which allow persistence to be delayed—either for a brief period of time or until the execution of an explicit *sync* operation. Experiments with Rochester's Montage system confirm that buffered persistence can approach the performance of nonpersistent structures placed in NVM—arguably the best one could hope for, and dramatically faster than systems with stricter semantics.

As NVM proliferates, there will be more and more opportunities to soften the traditional boundary between (transient, byte-addressable) memory and (persistent, block-structured) storage. As we seek to formalize the development of persistent data structures, the PODC/DISC community will want to consider not only alternative correctness criteria, but also evolving hardware characteristics. Issues to be considered include the choice between evicting and non-evicting write-back, the introduction of nonvolatile caches, and the possibility that NVM bandwidth and latency may vary greatly with access granularity, locality, and concurrent DRAM activity.

## CCS Concepts

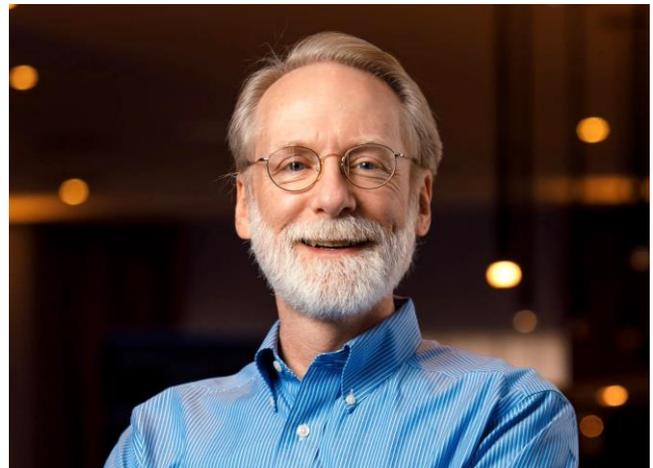
- Computer systems organization ~ Dependable and fault-tolerant systems and networks ~ Reliability
- Theory of computation ~ Design and analysis of algorithms ~ Concurrent algorithms

## Author Keywords

Durable linearizability; periodic persistence; nonvolatile memory

## BIOGRAPHY

Michael L. Scott is the Arthur Gould Yates Professor of Engineering and Chair of the Department of Computer Science at the University of Rochester. He received his Ph.D. from the University of Wisconsin–Madison in 1985. During the 2014–2015 academic year he was a Visiting Scientist at Google. He is best known for work in synchronization algorithms and concurrent data structures, in recognition of which he shared the 2006 SIGACT/SIGOPS Dijkstra Prize. His textbook on programming language design and implementation (*Programming Language Pragmatics*, 4<sup>th</sup> edition, Morgan Kaufmann, 2016) and his monograph on *Shared Memory Synchronization* (Morgan & Claypool, 2013) are standard references in the field. He has been named a Fellow of the ACM, the IEEE, and the AAAS. At the University of Rochester, he has received the lifetime achievement award of the Hajim School of Engineering and Applied Sciences, together with awards for both graduate and undergraduate teaching.



## ACKNOWLEDGEMENTS

Ideas described in this keynote reflect joint work with past and current students, including Joseph Izraelevitz, Haosen Wen, Wentao Cai, Mingzhe Du, and Chris Kjellqvist. This work was supported in part by NSF grants CCF-1717712, CNS-1900803, and CNS-1955498, and by a Google Faculty Research award.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author(s).

PODC '22, July 25–29, 2022, Salerno, Italy.

© 2022. Copyright is held by the owner/author(s).

ACM ISBN 978-1-4503-9262-4/22/07.

<https://doi.org/10.1145/3519270.3538455>