Consensus with Max Registers

James Aspnes
Department of Computer Science, Yale University, New Haven, CT, USA
james.aspnes@gmail.com

He Yang Er
Department of Computer Science, Yale University, New Haven, CT, USA
erheyang@gmail.com

Abstract

We consider the problem of implementing randomized wait-free consensus from max registers under the assumption of an oblivious adversary. We show that max registers solve m-valued consensus for arbitrary m in expected $O(\log^* n)$ steps per process, beating the $\Omega\left(\frac{\log m}{\log\log m}\right)$ lower bound for ordinary registers when m is large and the best previously known $O(\log\log n)$ upper bound when m is small. A simple max-register implementation based on double-collect snapshots translates this result into an $O(n\log n)$ expected step implementation of m-valued consensus from n single-writer registers, improving on the best previously-known bound of $O(n\log^2 n)$ for single-writer registers.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases consensus, max register, single-writer register, oblivious adversary, shared memory

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.1

Funding *James Aspnes*: Supported in part by NSF grants CCF-1637385 and CCF-1650596.

1 Introduction

Most work on randomized wait-free shared-memory consensus, dating back to the initial papers of Abrahamson [1] and Chor, Israeli, and Li [13], assumes either single-writer or multi-writer atomic registers as the base object provided by the system. Atomic registers have **consensus number** 1 [19], meaning they can solve consensus deterministically for no more than one process [21]. They are also assumed to be available by default when computing the consensus number of an object [19, 20]. This makes them the weakest object that is usually considered when solving shared-memory consensus. But there are other objects that also have consensus number 1 that may provide better performance than registers when implementing *randomized* consensus for many processes.

We examine the power of consensus protocols built from **max registers**, which are register-like objects for which the `writeMax` operation writes values, and the `readMax` operation returns the largest value previously written [6]. We show that against an oblivious adversary, the expected individual step complexity of binary consensus can be improved from $O(\log\log n)$ [5] using standard registers to $O(\log^* n)$ using max registers, closing the gap between the best previously known upper bounds for consensus and the closely-related problem of test-and-set [16].

Consensus protocols built from max registers also tolerate more than two possible input values better than those built from atomic registers. We show that max registers can implement an m-valued **adopt-commit object** [2, 15, 22] in $O(1)$ steps using $O(1)$ max registers. Adopt-commit objects can be derived from any shared-memory consensus protocol, and there is a tight $\Omega(\log m/\log\log m)$ lower bound on the individual step complexity of implementations of adopt-commit objects from atomic registers that carries over to consensus [8]. Using max registers, we can avoid this lower bound and obtain m-valued consensus for any m while still getting $O(\log^* n)$ individual step complexity.

By implementing a max register from single-writer registers accessed via collects (in which a process sequentially reads n registers, one for each process), we can get consensus for single-writer registers in $O(n \log n)$ expected steps per process. This improves on the previous best known $O(n \log^2 n)$ upper bound [10] for single-writer registers, although at the cost of weakening the adversary from adaptive to oblivious. It also gets close to the $\Omega(n)$ lower bound on the worst-case expected individual step complexity that follows immediately from the need for the first process to check the registers belonging to all other processes. We suspect that further improvements may be possible through a more careful analysis of how maximum values interact with collects, but this analysis is beyond the scope of the present work.

1.1 Model

We consider the standard model of an asynchronous shared-memory system, where a collection of n **processes** communicate by applying operations to shared **objects**. Typically these objects are **atomic registers**, which support a `write`(v) operation that changes the state of the register to v (and returns nothing) and a `read` operation that returns the value of the most recent previous `write` operation, or a default initial value if there is no previous `write` operation. But we will also consider **max registers**, in which a `readMax` operation instead returns the *largest* value of any preceding `writeMax` operation, or a default initial value if there is none.

Concurrency is modeled by interleaving. An **execution** $C_0\pi_1C_1\pi_2\ldots$ consists of an alternating sequence of **configurations** C_i giving the state of all processes and objects in the system, and operations π_i, each of which is carried out by some process. An operation π is **enabled** in a configuration C if there is a process in C that chooses π as its next operation. The effect of applying the operation is to transition to a new configuration C' that updates the state of the process executing π and the object to which it is applied. At each step, an **adversary** chooses which of the enabled operations occurs.

For deterministic protocols with known initial configuration C_0, an execution can be summarized by giving a **schedule** $\pi_1\pi_2\ldots$, consisting only of the operations.

For randomized protocols, an **adaptive** adversary chooses π_i with full knowledge of C_i, including internal states of the processes. An **oblivious** adversary instead fixes a sequence of process ids $p_{i_1}p_{i_2}\ldots$ at the start of the execution, and p_{i_j} carries out the j-th step in all executions. In both cases, the execution obtained is a random variable that depends on both the adversary's strategy and the outcomes of any random choices made by the processes.

The **total step complexity** of an execution is the number of operations carried out by all processes during the execution. The **individual step complexity** is the maximum number of operations carried out by any individual process during the execution.

The **space complexity** of a protocol is the total number of objects in the system, whether they are used in a particular execution or not. This does not depend on the capacity of each object: a register that holds one bit and a register that holds an unbounded number of bits both count as a single object for this purpose. The **allocate-on-use space complexity** [9] of an execution is the number of objects to which at least one operation is applied during the execution.

2 Consensus using max registers

Consensus [23] has three requirements:

Termination All processes finish.

Validity The output value of every process is the input value of some process.

Agreement All processes return the same output value.

We will be building **wait-free** consensus, meaning that the termination condition must hold for each process in all executions, regardless of the relative speeds of the other processes.

For our consensus protocol, we follow the modular construction suggested in [4]. This construction alternates **conciliator** objects with **adopt-commit** objects, and gives step complexity and space complexity proportional to the sum of the corresponding complexities of the conciliator and adopt-commit objects.

A **conciliator** object [4] is similar to a consensus protocol, but it replaces the agreement condition with a **probabilistic agreement** condition, that says that all processes return the same output with some minimum probability $\delta > 0$.

An **adopt-commit** object [3, 22] abstracts the adopt-commit protocol of Gafni [15], which is used to detect and enforce agreement obtained through other means. In addition to an output value, it also supplies a tag adopt or commit. Termination and validity hold as in consensus, but the agreement condition is now replaced by two new conditions:[1] **convergence**, which says that if all inputs are equal to some value v then all processes return $\langle \mathsf{commit}, v\rangle$; and **coherence**, which says that if any process returns $\langle \mathsf{commit}, v\rangle$ then all processes return either $\langle \mathsf{commit}, v\rangle$ or $\langle \mathsf{adopt}, v\rangle$. The significance of these conditions is that once all processes have the same input (for example, as the result of a preceding conciliator successfully producing agreement), then all will commit, and if any process commits even without all process having the same input, it will force all processes to agree on their outputs, allowing the first process to safely return knowing that any others will return the same value after the next adopt-commit.

Each execution of a conciliator has a δ chance of resulting in agreement, meaning that a protocol using this construction finishes in $1/\delta + 1$ expected phases. For constant δ, this makes the expected cost of the consensus protocol a constant multiple of the sum of the costs of the conciliator and adopt-commit objects. So our goal will be to use max registers to make both of these objects cost as little as possible.

2.1 Conciliator using max registers

To implement a conciliator from max registers, we construct a **sifter** [2] supplemented by the **persona** construction of [5]. A sifter is a mechanism for getting rid of processes quickly. The persona construction allows losers in this process to effectively take over the role of winners by copying sufficient control information (the *persona*) to duplicate the winning process's behavior.

The intuition behind our construction is that if a sequence of s distinct random ranks is written to a max register, only the left-to-right maxima will actually appear in the register. If the ranks are chosen independently of each other and of the order of the writes, there are $\sum_{i=1}^{s} \frac{1}{i} = H_s \leq 1 + \ln s$ such maxima on average, and we get a reduction from s to $O(\log s)$ values each time we apply this trick. This remains true if correlated duplicate ranks appear in the sequence, so long as the random ranks are independent of the order in which they *first* appear. This is enforced by having each process associate a sequence of independent random ranks (the *persona*) with its input value at the start of the protocol. When another process adopts the same value, it also adopts the same random ranks. This allows a fast process to pick up the value of a slow process and carry it to victory – a necessary condition for wait-freedom, since processes cannot leave until a winner is determined – without creating correlations that the adversary can exploit in the sifter mechanism.

[1] Our choice of names for these conditions follows [8].

Pseudocode for the conciliator is given in Algorithm 2.1. It uses $\ell = 4 + \log^* n$ rounds, with an independent random rank associated with each process's input value for each round. The ranks can be chosen from a variety of distributions, but for simplicity we assume that each rank is an independent uniform random integer in the range 1 through n^3.

Algorithm 2.1 Conciliator using max registers.

```
1 procedure conciliator(v)
2 |   Choose random ranks r_1 ... r_ℓ
3 |   for i ← 1 ... ℓ do
4 |   |   writeMax(M_i, ⟨r_i, ..., r_ℓ, v⟩)
5 |   |   ⟨r_i, ..., r_ℓ, v⟩ ← readMax(M_i)
6 |   end
7 |   return v
8 end
```

▶ **Lemma 1.** *Algorithm 2.1 implements a conciliator with a constant probability of agreement for sufficiently large n.*

Proof. Termination and validity are immediate from inspection of the code. Probabilistic agreement requires more work.

Define the persona of process p after 0 rounds as its initial tuple $\langle r_1, \ldots, r_\ell, v\rangle$ and after i rounds as the tuple $\langle r_{i+1}, \ldots, r_\ell, v\rangle$ that it stores after completing its i-th `readMax` operation in Line 5.

Let D be the event that all r_i are distinct from each other for each i. For r_i chosen uniformly in the range 1 through n^3 we will have $\Pr\left[\overline{D}\right] \le n^{-3}\binom{n}{2}\ell = O(n^{-1}\log^* n)$ by the union bound.

Let X_i be the number of distinct personae that appear across all processes after round i. We will argue by induction that $\mathrm{E}\left[X_i \mid D\right] \le f^{(i)}(n)$, where $f(x) = 1 + \ln x$ and $f^{(i)}$ is the i-fold iteration of f defined by $f^{(0)}(x) = x$ and $f^{(i+1)}(x) = f(f^{(i)}(x))$.

Initially, $\mathrm{E}\left[X_0 \mid D\right] = n = f^0(n)$. For the induction step, observe that any persona after $i > 0$ rounds is read from M_i and thus must be a left-to-right maximum of the values written to M_i. The adversary is oblivious, so it cannot observe r_i when scheduling the operations on M_i; at best, it knows only which processes share the same persona. But any particular persona $\langle r_i, \ldots, r_\ell, v\rangle$ is a left-to-right maximum if and only if the first occurrence of this persona in the sequence is a left-to-right maximum. There are X_{i-1} such distinct personae, and by symmetry a persona that appears first after $j-1$ other distinct personae has a $1/j$ chance of being a left-to-right maximum. This gives $\mathrm{E}\left[X_i \mid D, X_{i-1}\right] = \sum_{j=1}^{X_{i-1}} \frac{1}{j} = H_{X_{i-1}} \le 1 + \ln X_{i-1} = f(X_{i-1})$. Because f is concave, Jensen's inequality gives $\mathrm{E}\left[X_i \mid D\right] = \mathrm{E}\left[\mathrm{E}\left[X_i \mid D, X_{i-1}\right] \mid D\right] \le \mathrm{E}\left[f(X_{i-1}) \mid D\right] \le f(\mathrm{E}\left[X_{i-1} \mid D\right]) = f(f^{(i-1)}(n)) = f^{(i)}(n)$.

A straightforward numerical calculation shows that $f(x) \le \log_2 x$ for $x \ge 10$, which implies that $f^{(i+1)}(n) \le \log_2^{(i+1)}(n)$ as long as $f^{(i)}(n) \ge 10$. It follows that $f^{(i)}(n) < 10$ when $i = \log^* n$. Applying f four more times gives $f^{(4+\log^* n)} < f^{(4)}(10) < 1.59$. So $\mathrm{E}\left[X_\ell \mid D\right] < 1.59$.

Because X_i is always at least 1, $X_\ell - 1 \ge 0$, so Markov's inequality applies to $X_\ell - 1$, giving $\Pr\left[X_\ell \ge 2 \mid D\right] = \Pr\left[X_\ell - 1 \ge 1 \mid D\right] \le \mathrm{E}\left[X_\ell - 1 \mid D\right] < 0.59$. We can remove the conditioning to get $\Pr\left[X_\ell \ge 2\right] \le 0.59 + \Pr\left[D\right] = 0.59 + o(1)$. This completes the proof of probabilistic agreement. ◀

2.2 Adopt-commit using max registers

Pseudocode for an adopt-commit object implemented from max registers is given in Algorithm 2.2. The structure is similar to the implementation of adopt-commit from a generic **conflict detector** given in [8, Algorithm 2].

Here a pair of max registers min and max serve as the conflict detector, which detects when multiple distinct values have arrived at the adopt-commit. The mechanism for this is that min holds the smallest input value (negated to turn the max register into a min register), while max holds the largest, so that distinct values will cause these to diverge.

To obtain a commit, a process must successfully write its value to the proposal register before any process with a distinct value finishes writing to both max registers. This will cause later processes to see and adopt the same value.

Algorithm 2.2 Adopt-commit using max registers.

```
1  procedure adoptCommit(v)
2      writeMax(min, −v)
3      writeMax(max, v)
4      if proposal ≠ ⊥ then
5          v ← proposal
6      end
7      proposal ← v
8      if readMax(min) = −v and readMax(max) = v then
9          return ⟨commit, v⟩
10     else
11         return ⟨adopt, v⟩
12     end
13 end
```

▶ **Lemma 2.** *Algorithm 2.2 implements an adopt-commit object.*

Proof. Termination and validity are immediate from inspection of the code.

For convergence, suppose that all calls to adoptCommit have the same input v. Then no value other than v is ever written to proposal or max, and no value other than $-v$ is ever written to min. It follows that (a) every process has value v when it reaches Line 8, and (b) all processes read $-v$ from min and v from max in this line, causing them to return $\langle \text{commit}, v \rangle$.

For coherence, suppose that some process p returns $\langle \text{commit}, v \rangle$. Then at the point where p starts executing Line 8, proposal $\neq \perp$, Process p subsequently reads $-v$ from min and v from max, which implies that at this same point no process p' with a value $v' < v$ has yet written to min and no process with a value $v' > v$ has yet written to max. Since a process must do both before checking proposal in Line 4, any such process will read v from proposal and adopt this value. ◀

The proposal register is assumed to be a standard atomic register, but if a max-register-only implementation is desired, proposal can be replaced by a max register without affecting the complexity or proof of correctness.

2.3 Full result

Combining Lemmas 1 and 2 gives:

▶ **Theorem 3.** *Alternating Algorithms 2.1 and 2.2 implements randomized consensus for any number of input values against an oblivious adversary, with expected individual step complexity $O(\log^* n)$ and expected space complexity $O(\log^* n)$ in the allocate-on-use model.*

3 Consensus using collects

If max registers are not available, we can replace them with an array of n ordinary atomic registers over which we perform **collect** operations, a non-atomic operation that involves a process reading all n registers in some arbitrary order. We use a variant of the classic double-collect technique [24] to ensure that the value returned is in fact the maximum value in the array at some point during the execution. We will show that this does not impose too much additional cost and does not create opportunities for the adversary to bias the outcome.[2]

For the adopt-commit object, we skip the max register implementation entirely and simply use Gafni's original implementation of adopt-commit from two collects [15, §4.2].

Pseudocode for the conciliator is given in Algorithm 3.1. We assume that each $A[i][j]$ is an atomic register that initially holds a default value $\perp$ that is treated as smaller than all non-default values.

Algorithm 3.1 Conciliator using collect. Code for process j.

```
1  procedure conciliator(v)
2      Choose random ranks r_1 ... r_ℓ
3      for i ← 1 ... ℓ do
4          A[i][j] ← ⟨r_i, ..., r_ℓ, v⟩
5          cur ← ⊥
6          repeat
7              prev ← cur
               // Perform collect on A[i]
8              cur ← A[i][1 ... n]
9          until prev ≠ ⊥ and max(cur) = max(prev)
10         ⟨r_i, ..., r_ℓ, v⟩ ← max(cur)
11     end
12     return v
13 end
```

[2] An alternative might be to use an existing implementation of bounded max registers from atomic registers, such as the original linearizable implementation of [6] or the strongly linearizable implementation of [18]. This runs into two difficulties. First, these implementations assume multi-writer registers, and there are implementations of conciliators directly from multi-writer registers that run in $O(\log \log n)$ steps [5], much less than the $\Omega(\log n)$ steps needed for a single operation of a polynomially-bounded max register [6]. Second, neither linearizability nor strong linearizability is enough to guarantee that an implementation can be used in place of the original atomic object without affecting the behavior of the algorithm when scheduling is controlled by an oblivious adversary [17]. Though the stronger condition of **uniform linearizability** can allow such composition [14], we are not aware of any uniformly linearizable implementations of max registers.

To show that this works, we must first argue that the maximum value computed in Line 10 corresponds to the actual maximum value in $A[i][1 \ldots n]$ at some point in the execution.

▶ **Lemma 4.** *Let $x = \langle r_i, \ldots, r_\ell, v \rangle$ be computed in Line 10 during some execution of Algorithm 3.1. Then at some point during the execution, x is the maximum value in registers $A[i][1]$ through $A[i][n]$.*

Proof. First observe that each $A[i][j]$ can only increase over time, and thus the same holds for the maximum value in $A[i]$.

Because x is the maximum of the non-overlapping collects that provided prev and cur, we know that (a) the maximum value at the end of the first collect is at least x, and (b) the maximum value at the start of the second collect is at most x. So throughout the interval between these collects, the maximum value is exactly x. ◀

Applying the same argument as in Lemma 1, if X_i distinct personae with distinct random ranks appear across all processes after i rounds, then on average there are H_{X_i} left-to-right maxima in the sequence of values written to $A[i]$, giving $X_{i+1} \leq H_{X_i} \leq 1 + \ln X_i$ on average. Since each repetition of the inner loop of Algorithm 3.1 by some process requires a new maximum, this means that (a) on average, only $O(\log n)$ collects are performed during the first round, and (b) on average, at most $O(\log \log n)$ collects are performed during subsequent rounds. The first round dominates, giving an expected $O(\log n)$ collects per process, for an expected individual step complexity of $O(n \log n)$. We can similarly argue that after ℓ rounds there is only one surviving persona with constant probability.

As written, the algorithm uses $O(n \log^* n)$ registers. However, we can trivially have each process consolidate its $O(\log^* n)$ registers $A[1][j]$ through $A[\ell][j]$ into a single register divided into ℓ fields, giving an implementation with only n registers.

Applying the appropriate corrections to deal with the small chance of duplicate ranks, we get:

▶ **Lemma 5.** *Algorithm 3.1 implements a conciliator with constant agreement probability with expected individual step complexity $O(n \log n)$ and space complexity n.*

Since it is possible to implement adopt-commit using $O(1)$ write operations and two collects [15], and since we can consolidate all registers used by a particular process, even across rounds, into a single register, we get

▶ **Theorem 6.** *There is an implementation of randomized consensus that achieves consensus against an oblivious adversary with expected individual step complexity $O(n \log n)$ using n single-writer registers.*

The step complexity of this algorithm compares favorably with the best previously known implementation of consensus from single-writer registers, the $O(n \log^2 n)$ algorithm of [10]. It should be noted however that the price of this improvement is that we have assumed an oblivious adversary instead of an adaptive adversary.

4 Conclusion and open problems

We have shown that using max registers as a base object allows for a significant improvement in the time and space complexity of randomized consensus over previous solutions based on atomic registers. We have also shown that this approach gives improvements even for single-writer atomic registers, by implementing max registers from collects over arrays of single-writer registers.

At present there is no known non-trivial lower bound on the expected individual step complexity of randomized consensus with an oblivious adversary, even for ordinary registers, although a lower bound on the distribution of the individual step complexity is known [12]. So it may be that a more sophisticated algorithm could reduce the expected time for max-register-based consensus from $O(\log^* n)$ to as little as $O(1)$.

Similarly, there is no stronger lower bound on the individual step complexity of consensus implemented from single-writer registers than the trivial $\Omega(n)$ bound. Here we suspect that a more careful analysis of the possible set of maximum values returned by concurrent *single* collects could reduce the gap between this lower bound and the $O(n \log n)$ upper bound obtained using double collects.

It is also interesting to consider what happens in a message-passing system. It is known that the classic Attiya-Bar-Noy-Dolev (ABD) implementation of an atomic register from asynchronous message-passing with fewer than $n/2$ failures [11] extends in a straightforward way to give a linearizable implementation of a max register in $O(1)$ time using $O(n)$ messages per operation [7]. However, the same problem of linearizability interacting poorly with randomization that required using double collects in Algorithm 3.1 applies here, and it is not clear if simply using ABD in Algorithm 2.1 would prevent even an oblivious adversary from preserving more values than expected. As in the case of single collects, further analysis is needed.

References

1 Karl Abrahamson. On Achieving Consensus Using a Shared Memory. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 291–302, 1988.

2 Dan Alistarh and James Aspnes. Sub-logarithmic test-and-set against a weak adversary. In *Distributed Computing: 25th International Symposium, DISC 2011*, volume 6950 of *Lecture Notes in Computer Science*, pages 97–109. Springer-Verlag, September 2011.

3 Dan Alistarh, Seth Gilbert, Rachid Guerraoui, and Corentin Travers. Of Choices, Failures and Asynchrony: The Many Faces of Set Agreement. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *ISAAC*, volume 5878 of *Lecture Notes in Computer Science*, pages 943–953. Springer, 2009. doi:10.1007/978-3-642-10631-6_95.

4 James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. *Distributed Computing*, 25(2):179–188, May 2012.

5 James Aspnes. Faster randomized consensus with an oblivious adversary. *Distributed Computing*, 28(1):21–29, February 2015.

6 James Aspnes, Hagit Attiya, and Keren Censor-Hillel. Polylogarithmic concurrent data structures from monotone circuits. *jacm*, 59(1):2:1–2:24, February 2012.

7 James Aspnes and Keren Censor-Hillel. Atomic Snapshots in $O(\log^3 n)$ Steps Using Randomized Helping. In Yehuda Afek, editor, *Distributed Computing: 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14–18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 254–268. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-41527-2_18.

8 James Aspnes and Faith Ellen. Tight Bounds for Adopt-Commit Objects. *Theory of Computing Systems*, 55(3):451–474, 2014. doi:10.1007/s00224-013-9448-1.

9 James Aspnes, Bernhard Haeupler, Alexander Tong, and Philipp Woelfel. Allocate-On-Use Space Complexity of Shared-Memory Algorithms. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15–19, 2018*, volume 121 of *LIPIcs*, pages 8:1–8:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.8.

10 James Aspnes and Orli Waarts. Randomized consensus in expected $O(n \log^2 n)$ operations per processor. *SIAM J. Comput.*, 25(5):1024–1044, October 1996.

11 Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing Memory Robustly in Message-Passing Systems. *jacm*, 42(1):124–142, 1995. doi:10.1145/200836.200869.

12 Hagit Attiya and Keren Censor-Hillel. Lower Bounds for Randomized Consensus under a Weak Adversary. *SIAM J. Comput.*, 39(8):3885–3904, 2010. doi:10.1137/090751906.

13 Benny Chor, Amos Israeli, and Ming Li. Wait-Free Consensus Using Asynchronous Hardware. *SIAM J. Comput.*, 23(4):701–712, 1994.

14 Oksana Denysyuk and Philipp Woelfel. Are Shared Objects Composable Under an Oblivious Adversary? In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, pages 335–344, New York, NY, USA, 2016. ACM. doi:10.1145/2933057.2933115.

15 Eli Gafni. Round-by-Round Fault Detectors: Unifying Synchrony and Asynchrony (Extended Abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 143–152, 1998. doi:10.1145/277697.277724.

16 George Giakkoupis and Philipp Woelfel. On the time and space complexity of randomized test-and-set. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 19–28. ACM, 2012. doi:10.1145/2332432.2332436.

17 Wojciech Golab, Lisa Higham, and Philipp Woelfel. Linearizable Implementations Do Not Suffice for Randomized Distributed Computation. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 373–382, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993687.

18 Maryam Helmi, Lisa Higham, and Philipp Woelfel. Strongly Linearizable Implementations: Possibilities and Impossibilities. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 385–394, New York, NY, USA, 2012. ACM. doi:10.1145/2332432.2332508.

19 Maurice Herlihy. Wait-free Synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, January 1991. doi:10.1145/114005.102808.

20 Prasad Jayanti. Robust wait-free hierarchies. *J. ACM*, 44(4):592–614, 1997. doi:10.1145/263867.263888.

21 Michael C. Loui and Hosame H. Abu-Amara. Memory Requirements for Agreement Among Unreliable Asynchronous Processes. In Franco P. Preparata, editor, *Parallel and Distributed Computing*, volume 4 of *Advances in Computing Research*, pages 163–183. JAI Press, 1987.

22 Achour Mostefaoui, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. The Combined Power of Conditions and Information on Failures to Solve Asynchronous Set Agreement. *sicomp*, 38(4):1574–1601, 2008.

23 M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *jacm*, 27(2):228–234, April 1980.

24 Gary L Peterson and James E Burns. Concurrent reading while writing II: the multi-writer case. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 383–392. IEEE, 1987.