An Improved Lower Bound for the Time Complexity of Mutual Exclusion*

(Extended Abstract)

James H. Anderson and Yong-Jik Kim
Department of Computer Science
University of North Carolina at Chapel Hill
{anderson, kimy}@cs.unc.edu

ABSTRACT

We establish a lower bound of $\Omega(\log N/\log\log N)$ remote memory references for N-process mutual exclusion algorithms based on reads, writes, or comparison primitives. Our bound improves an earlier bound of $\Omega(\log\log N/\log\log\log N)$ established by Cypher.

1. INTRODUCTION

Recent work on mutual exclusion [6] has focused on the design of "scalable" local-spin algorithms. Good scalability is achieved in such algorithms by using spin loops in which shared variables are read that are stored in a local cache or memory module. Early work on local-spin algorithms focused on "queue locks" implemented with fetch-and-ϕ primitives. Several such algorithms were proposed in which only $O(1)$ remote memory references (*i.e.*, references requiring a traversal of the processor-to-memory interconnect) are required per critical-section access [3, 7, 8].

In later work, Yang and Anderson initiated a study of time complexity bounds for local-spin algorithms, where "time" is measured by only counting remote memory references [2, 11]. One of their main contributions was an N-process mutual exclusion algorithm with $\Theta(\log N)$ time complexity that uses only read and write operations [11]. In addition, they proved several lower bounds for systems in which *write-* or *access-contention* is restricted (*i.e.*, systems for which a bound exists on the number of processes that may be concurrently enabled to write or access the same shared variable). For example, they showed that if access-contention is c, then $\Omega(\log_c N)$ remote operations are required for a process to enter its critical section.

Yang and Anderson's lower bounds apply only to systems with limited write- or access-contention. In particular, they left open the question of whether $\Omega(\log N)$ is a lower bound for mutual exclusion under read/write atomicity with arbitrary write- and access-contention; such a bound would imply that their $\Theta(\log N)$ algorithm is time-optimal.

Cypher was the first to attempt to address this open question [5]. He established a lower bound of $\Omega(\log\log N/\log\log\log N)$, which surprsingly is applicable not only to algorithms based on reads and writes, but also to algorithms that use comparison primitives such as test-and-set and compare-and-swap.[1] Given the fact that fetch-and-ϕ primitives can be used to implement mutual exclusion in $O(1)$ time, this result pointed to an unexpected weakness of compare-and-swap, which is still widely regarded as being the most useful of all primitives to provide in hardware.

Contributions of this paper. In this paper, we show that Cypher's lower bound can be improved to $\Omega(\log N/\log\log N)$. Thus, we have *almost* succeeded in establishing the optimality of Yang and Anderson's $\Theta(\log N)$ algorithm. Our lower bound suggests that it is likely that, from an asymptotic standpoint, compare-and-swap and test-and-set are no better than reads and writes when implementing local-spin mutual exclusion algorithms. Moreover, the time complexity gap that exists between comparison primitives and fetch-and-ϕ primitives is actually quite wide. Thus, comparison primitives may not be the best choice to provide in hardware if one is interested in scalable synchronization.

The rest of the paper is organized as follows. In Sec. 2, we present our model of shared-memory systems. Our proof is then sketched in Sec. 3 and presented in detail in Sec. 4 for systems with read/write atomicity. Comparison primitives are then considered in Sec. 5. We conclude in Sec. 6.

2. DEFINITIONS

Our system model is similar to that used in [2].

Shared-memory systems. A *shared-memory system* $\mathcal{S} = (C, P, V)$ consists of a set of computations C, a set of processes P, and a set of variables V. A *computation* is a finite sequence of events.

An *event* e is denoted $[R, W, p]$, where $p \in P$. The sets R and W consist of pairs (v, α), where $v \in V$. This notation represents an event of process p that reads the value α from

*Work supported by NSF grants CCR 9732916, CCR 9972211, CCR 9988327, and ITR 0082866.

[1]A *comparison primitive* conditionally updates a shared variable after first testing that its value meets some condition.

variable v for each pair (v, α) in R, and writes the value α to variable v for each pair (v, α) in W. Each variable in R (or W) is assumed to be distinct. We define $Rvar(e)$ ($Wvar(e)$) to be the set of variables read (written) by e, *i.e.*, $Rvar(e) = \{v : (v, \alpha) \in R\}$ and $Wvar(e) = \{v : (v, \alpha) \in W\}$. We define $var(e)$, the set of all variables accessed by e, to be $Rvar(e) \cup Wvar(e)$. We say that event e *accesses* each variable in $var(e)$. We say that process p is the *owner* of e, denoted $owner(e) = p$. For brevity, we sometimes use e_p to denote an event owned by process p.

Each variable is *local* to at most one process and is *remote* to all other processes. (Note that a variable may be remote to *all* processes.) An *initial value* is associated with each variable. An event is *local* if it does not access any remote variable, and *remote* otherwise.

We use $\langle e, \ldots \rangle$ to denote a computation that begins with the event e, and $\langle\rangle$ to denote the empty computation. We use $H \circ G$ to denote the computation obtained by concatenating computations H and G. The value of variable v at the end of computation H, denoted $value(v, H)$, is the last value written to v in H (or the initial value of v if v is not written in H). The last event to write to v in H is denoted $writer\_event(v, H)$,[2] and its owner is denoted $writer(v, H)$. If v is not written by any event in H, then we let $writer(v, H) = \perp$ and $writer\_event(v, H) = \perp$.

For a computation H and a set of processes Y, $H \,|\, Y$ denotes the subcomputation of H that contains all events in H of processes in Y. Computations H and G are *equivalent with respect to* Y iff $H \,|\, Y = G \,|\, Y$. A computation H is a Y-*computation* iff $H = H \,|\, Y$. For simplicity, we abbreviate the preceding definitions when applied to a singleton set of processes (*e.g.*, $H \,|\, p$ instead of $H \,|\, \{p\}$).

The following properties apply to any shared-memory system.

(P1) If $H \in C$ and G is a prefix of H, then $G \in C$.

(P2) If $H \circ \langle e_p \rangle \in C$, $G \in C$, $G \,|\, p = H \,|\, p$, and if $value(v, G) = value(v, H)$ holds for all $v \in Rvar(e_p)$, then $G \circ \langle e_p \rangle \in C$.

(P3) If $H \circ \langle e_p \rangle \in C$, $G \in C$, $G \,|\, p = H \,|\, p$, then $G \circ \langle e'_p \rangle \in C$ for some event e'_p such that $Rvar(e'_p) = Rvar(e_p)$ and $Wvar(e'_p) = Wvar(e_p)$.

(P4) For any $H \in C$, $H \circ \langle e_p \rangle \in C$ implies that $\alpha = value(v, H)$ holds, for all $(v, \alpha) \in Rvar(e_p)$.

For notational simplicity, we make the following assumption, which requires each remote event to be either an atomic read or an atomic write. Later, in Sec. 5, we show that this assumption can be relaxed to allow comparison primitives.

Atomicity Assumption: Each event of process p may either read or write (but not both) at most one variable that is remote to p. $\square$

Mutual exclusion systems. We now define a special kind of shared-memory system, namely mutual exclusion systems, which are our main interest.

A *mutual exclusion system* $\mathcal{S} = (C, P, V)$ is a shared-memory system that satisfies the following properties. Each process $p \in P$ has a local variable $stat_p$ that ranges over $\{ncs,\ entry,\ exit\}$.[3] The variable $stat_p$ is initially ncs and is accessed only by the following events: $Enter_p = [\{\},\ \{(stat_p,\ entry)\},\ p]$, $CS_p = [\{\},\ \{(stat_p,\ exit)\},\ p]$, and $Exit_p = [\{\},\ \{(stat_p,\ ncs)\},\ p]$. The allowable transitions of $stat_p$ are as follows: for all $H \in C$,

$$H \circ \langle Enter_p \rangle \in C \text{ iff } value(stat_p, H) = ncs;$$
$$H \circ \langle CS_p \rangle \in C \text{ only if } value(stat_p, H) = entry;$$
$$H \circ \langle Exit_p \rangle \in C \text{ only if } value(stat_p, H) = exit.$$

We henceforth assume each computation contains at most one $Enter_p$ event for each process p, because this is sufficient for our proof. The remaining requirements of a mutual exclusion system are as follows.

Exclusion: For all $H \in C$, if both $H \circ \langle CS_p \rangle \in C$ and $H \circ \langle CS_q \rangle \in C$ hold, then $p = q$.

Progress (starvation freedom): For all $H \in C$, if $value(stat_p, H) \neq ncs$, then there exists an X-computation G such that $H \circ G \circ \langle e_p \rangle \in C$, where $X = \{q \in P : value(stat_q, H) \neq ncs\}$ and e_p is either CS_p (if $value(stat_p, H) = entry$) or $Exit_p$ (if $value(stat_p, H) = exit$). $\square$

Cache-coherent systems. On cache-coherent systems, remote variables may sometimes be accessed without causing interconnect traffic. Our proof applies to such systems without modification. This is because we do not count every remote event, but only critical events, as defined below.

Definition: Let $\mathcal{S} = (C, P, V)$ be a mutual exclusion system. Let e_p be an event in $H \in C$. Then, we can write H as $F \circ \langle e_p \rangle \circ G$, where F and G are subcomputations of H. We say that e_p is a *critical event* in H iff one of the following conditions holds:

Transition event: e_p is one of $Enter_p$, CS_p, or $Exit_p$.

Critical read: There exists $v \in Rvar(e_p)$, remote to p, that is not read by p in $F \,|\, p$.

Critical write: There exists $v \in Wvar(e_p)$, remote to p, such that $writer(v, F) \neq p$. $\square$

Although state transition events do *not* actually cause cache misses, categorizing them as critical allows us to combine certain cases in our proof. A process executes only three transition events per critical section execution, so this has no asymptotic impact.

According to the above definition, a remote read of v by p is critical if it is the first read of v by p. A remote write of v by p is critical if **(i)** it is the first write of v by p; or **(ii)** some other process has written to v since p's last write of v.

Note that if p both reads and writes v, then both its first read of v and first write of v are considered critical. Depending on the system implementation, the latter of these two events might not generate a cache miss. However, even in such a case, the first such event will always generate a cache miss, and hence at least half of all such critical reads and

[2] We assume that multiple instances of the same event are distinguishable from each other. (For simplicity, we do not extend our notion of an event to include an additional identifier for distinguishability.)

[3] Each critical-section execution of p is captured by the single event CS_p, so $stat_p$ changes directly from $entry$ to $exit$.

writes will actually incur real global traffic. Hence, our lower bound remains asymptotically unchanged for such systems.

In a *write-through* cache scheme, writes always generate a cache miss. On the other hand, with a *write-back* scheme, a remote write to a variable v may create a cached copy of v, so that subsequent writes to v do not cause cache misses. In the definition above, if e_p is not the first write to v by p, then it is considered critical only if $writer(v, F) = q \neq p$ holds, which implies that v is stored in a local cache line of another process q.[4] In such a case, e_p must either invalidate or update the cached copy of v (depending on the system), thereby generating global traffic.

Note that the above definition of a critical event depends on the particular computation that contains the event. Therefore, when saying that an event is (or is not) critical, the computation containing the event must be specified.

3. PROOF STRATEGY

Our proof focuses on a special class of computations called "regular" computations. A regular computation consists of events of two groups of processes, "active processes" and "finished processes." Informally, an active process is a process in its entry section, competing with other active processes; a finished process is a process that has executed its critical section once, and is in its noncritical section. (These properties follow from (RF4), given later in this section.)

Definition: Let $\mathcal{S} = (C, P, V)$ be a mutual exclusion system, and H be a computation in C. We define $\mathrm{Act}(H)$, the set of *active processes* in H, and $\mathrm{Fin}(H)$, the set of *finished processes* in H, as follows.

$$\begin{aligned}\mathrm{Act}(H) &= \{p \in P : H \,|\, p \neq \langle\rangle \text{ and } \langle \mathit{Exit}_p\rangle \notin H\}\\ \mathrm{Fin}(H) &= \{p \in P : H \,|\, p \neq \langle\rangle \text{ and } \langle \mathit{Exit}_p\rangle \in H\}\end{aligned} \qquad \square$$

In our proof, several other process groups are considered as well. As explained below, $\mathrm{Act}(H)$ is partitioned into two sets of processes, the "invisible" and "promoted" processes, respectively. If H is regular, then all processes in $\mathrm{Act}(H)$ are invisible; if H is non-regular, then $\mathrm{Act}(H)$ may include both invisible and promoted processes. The set of promoted processes is described below. For a process to be invisible, it must be the case that no other process has knowledge of it. Because all active processes in a regular computation are invisible, we can "erase" any active process from such a computation (*i.e.*, remove its events from the computation) and still get a valid computation. The set of processes that have been "erased" in this way constitutes another of the processes groups of importance in our proof. These process groups are depicted in Fig. 2 (the roll-forward set is discussed below).

At the end of this section, a detailed overview of our proof is given. Here, we give cursory overview, so that the definitions that follow will make sense. Our proof is based on a combination two strategies, rolling forward and erasing (also used elsewhere [1, 4, 9]), in which longer and longer regular computations are inductively constructed. In each induction step, we append to each active process (except at most one) one future critical event. Such events may introduce information flow among active processes, which may be eliminated by erasing some active processes. Sometimes erasing alone doesn't leave enough active processes for the next induction step. In that case, some active processes are "promoted" and allowed to "roll forward" by executing together until they all reach their noncritical sections (NCSs). We erase some other active processes so that the remaining active processes remain invisible. The invisible processes (that are not erased) will constitute the set of active processes for the next regular computation in the induction. Although invisible processes may have knowledge of promoted (and finished) processes, once all promoted processes have finished execution, the regularity condition holds again (*i.e.*, all active processes are invisible).

The promoted and finished processes together constitute a "roll-forward set," which must meet the conditions defined below. Informally, (RF1) ensures that an invisible process is not known to any other processes; (RF2) and (RF3) bound the number of possible conflicts caused by appending a critical event; (RF4) ensures that the invisible, promoted, and finished processes behave as explained above; (RF5) ensures that we can erase any invisible process.

Definition: Let $\mathcal{S} = (C, P, V)$ be a mutual exclusion system, H be a computation in C, and RFS be a subset of P such that $\mathrm{Fin}(H) \subseteq \mathit{RFS}$ and $H \,|\, p \neq \langle\rangle$ for each $p \in \mathit{RFS}$. We say that RFS is a valid *roll-forward set* (*RF-set*) of H iff the following conditions hold.

(RF1) Assume that H can be written as $E \circ \langle e_p\rangle \circ F \circ \langle f_q\rangle \circ G$. If $p \neq q$ and there exists a variable $v \in \mathit{Wvar}(e_p) \cap \mathit{Rvar}(f_q)$ such that F does not contain a write to v (*i.e.*, $\mathit{writer\_event}(v, F) = \perp$), then $p \in \mathit{RFS}$. — *Informally, if a process p writes to a variable v, and if another process q reads that value from v, then $p \in \mathit{RFS}$ holds.*

(RF2) For any event e_p in H and any variable v in $\mathit{var}(e_p)$, if v is local to another process q ($\neq p$), then either $q \notin \mathrm{Act}(H)$ or $\{p, q\} \subseteq \mathit{RFS}$. — *Informally, if a process p accesses a variable that is local to another process q, then either q is not an active process in H, or both p and q belong to the roll-forward set RFS.*

(RF3) Consider a variable $v \in V$ and two different events e_p and f_q in H. Assume that both p and q are in $\mathrm{Act}(H)$, $p \neq q$, there exists a variable v such that $v \in \mathit{var}(e_p) \cap \mathit{var}(f_q)$, and there exists a write to v in H. Then, $\mathit{writer}(v, H) \in \mathit{RFS}$ holds. — *Informally, if a variable v is accessed by more than one processes in $\mathrm{Act}(H)$, then the last process in H to write to v (if any) belongs to RFS.*

(RF4) For any process p such that $H \,|\, p \neq \langle\rangle$, $\mathit{value}(\mathit{stat}_p, H)$ is

$$\begin{cases} \mathit{entry} & \text{if } p \in \mathrm{Act}(H) - \mathit{RFS} \\ \mathit{entry} \text{ or } \mathit{exit} & \text{if } p \in \mathrm{Act}(H) \cap \mathit{RFS} \\ \mathit{ncs} & \text{otherwise } (\textit{i.e.}, p \in \mathrm{Fin}(H)). \end{cases}$$

Moreover, if $p \in \mathrm{Fin}(H)$, then the last event of p in H is Exit_p. — *Informally, if a process p participates in H ($H \,|\, p \neq \langle\rangle$), then at the end of H, one of the following holds:* **(i)** *p is in its entry section and has not yet executed its critical section ($p \in \mathrm{Act}(H) - \mathit{RFS}$);* **(ii)** *$p$ is in the process*

[4]Effectively, we are assuming an idealized cache of infinite size: a cached variable may be updated or invalidated but it is never replaced by another variable. (Note that $\mathit{writer}(v, F) = q$ implies that q's cached copy of v has not been invalidated.) Cache size and associativity limitations can only *increase* the number of cache misses.

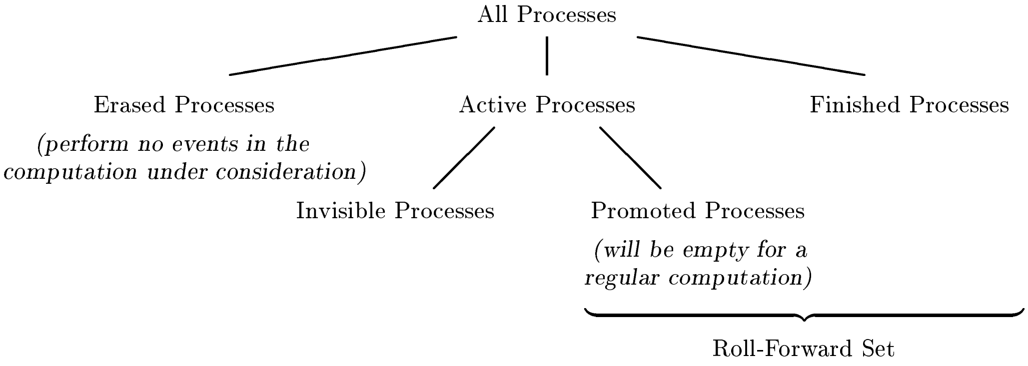

Figure 1: Process groups.

of "rolling forward" and is in its entry or exit section ($p \in \mathrm{Act}(H) \cap RFS$)*; or* **(iii)** p *has already finished its execution and is in its noncritical section* (*i.e.*, $p \in \mathrm{Fin}(H)$).

(RF5) Assume that H can be written as $E \circ \langle e_p \rangle \circ F \circ \langle f_p \rangle \circ G$. If there exists a variable v, remote to p, such that $v \in Wvar(e_p) \cap Wvar(f_p)$ and $writer(v, F) = q$, where $q \neq p$ and $q \neq \perp$, then there exists a process r in RFS such that $F \,|\, r$ contains a write to v. — *Informally, if an event f_p is a critical write to a variable v, and if f_p is* not *the first event of p to write to v, then there exists a write to v by some process r in RFS between the previous write to v by p (= event e_p) and event f_p. This condition is used to show that the property of being a critical write is conserved when considering certain related computations.* □

The invisible and promoted processes (which partition the set of active processes) are defined as follows.

Definition: Let $\mathcal{S} = (C, P, V)$ be a mutual exclusion system, H be a computation in C, and RFS be a valid RF-set of H. We define $\mathrm{Inv}_{RFS}(H)$, the set of *invisible processes* in H, and $\mathrm{Pmt}_{RFS}(H)$, the set of *promoted processes* in H, as follows.

$$\begin{aligned} \mathrm{Inv}_{RFS}(H) &= \mathrm{Act}(H) - RFS \\ \mathrm{Pmt}_{RFS}(H) &= \mathrm{Act}(H) \cap RFS \end{aligned}$$ □

For brevity, we sometimes omit the specific RF-set if it is obvious from the context, and simply use the notation $\mathrm{Inv}(H)$ and $\mathrm{Pmt}(H)$. Finally, the regularity condition can be defined succinctly as "all the processes we wish to roll forward have finished execution."

Definition: A computation H in C is *regular* iff $\mathrm{Fin}(H)$ is a valid RF-set of H. □

Detailed proof overview. The structure of our proof is depicted in Fig. 2. Initially, we start with H_1, in which $\mathrm{Act}(H_1) = P$, $\mathrm{Fin}(H_1) = \{\}$, and each process has one critical event. At the j^{th} induction step, we consider a computation H_j such that $\mathrm{Act}(H_j)$ consists of n processes, each of which executes j critical events in H_j. We show that if $j > c \log n$, where c is a constant to be specified later, then the lower bound has already been attained. Thus, in the inductive step, we assume $j \leq c \log n$. We construct a regular computation H_{j+1} such that $\mathrm{Act}(H_{j+1})$ consists of $\Omega(n/\log^2 n)$ processes,[5] each of which executes $j+1$ critical events in H_{j+1}. The construction method, formally described in Lemma 5, is explained below.

We show in Lemma 3 that, among the n processes in $\mathrm{Act}(H_j)$, at least $n-1$ processes can execute an additional critical event before entering its critical section. We call these events "future" critical events, and denote the corresponding set of processes by Y.

The processes in Y collectively execute at most $cn \log n$ critical events in H_j. Let $\mathcal{E}$ be the set of all these events plus all the future critical events. Clearly, $|\mathcal{E}| \leq (c \log n + 1)n$. Among the variables accessed by these events, we identify V_{HC}, the set of variables that experience "high contention," that is, those that are remotely accessed by at least $d \log^2 n$ critical events in $\mathcal{E}$, where d is another constant to be specified. Because of the Atomicity Assumption, each event can access at most one remote variable, so we have $|V_{\mathrm{HC}}| \leq \frac{(c \log n + 1)n}{d \log^2 n} \leq \frac{(c+1)n}{d \log n}$. Next, we partition the processes in Y depending on whether their future critical events access a variable in V_{HC}, as follows: $P_{\mathrm{HC}} = \{y \in Y : y\text{'s future critical event accesses some variable in } V_{\mathrm{HC}}\}$, and $P_{\mathrm{LC}} = Y - P_{\mathrm{HC}}$.

Because H_j is regular and $Y \subseteq \mathrm{Act}(H_j)$, we can erase any process in Y. Hence, we can erase the smaller of P_{HC} and P_{LC} and still have $\Omega(n)$ remaining active processes. We consider these cases separately.

Erasing strategy. Assume that we erased P_{HC} and saved P_{LC}. Define V_{LC} as the set of variables remotely accessed by the future critical events of P_{LC}. Then clearly, every variable in V_{LC} is a "low contention" variable, and hence is accessed by at most $d \log^2 n$ different critical events (and hence, different processes). Therefore, a future event by a process in P_{LC} can conflict with at most $d \log^2 n$ processes. By generating a "conflict graph" and applying Turán's theorem (Theorem 1), we can find a set of processes Z such that $|Z| = \Omega(n/\log^2 n)$ and their critical events do not conflict with each other. By retaining Z and erasing all other active processes, we can eliminate all conflicts. Thus, we can construct H_{j+1}.

Roll-forward strategy. Assume that we erased P_{LC} and saved P_{HC}. Every future event by a process in P_{HC} accesses a variable in V_{HC}. For each variable $v \in V_{\mathrm{HC}}$, we arrange the future critical events that access v by placing all writes before all reads. Thus, the only information flow between processes in P_{HC} is that from the "last writer" of each such

[5] We use $\log^2 n$ to denote $(\log n)^2$.

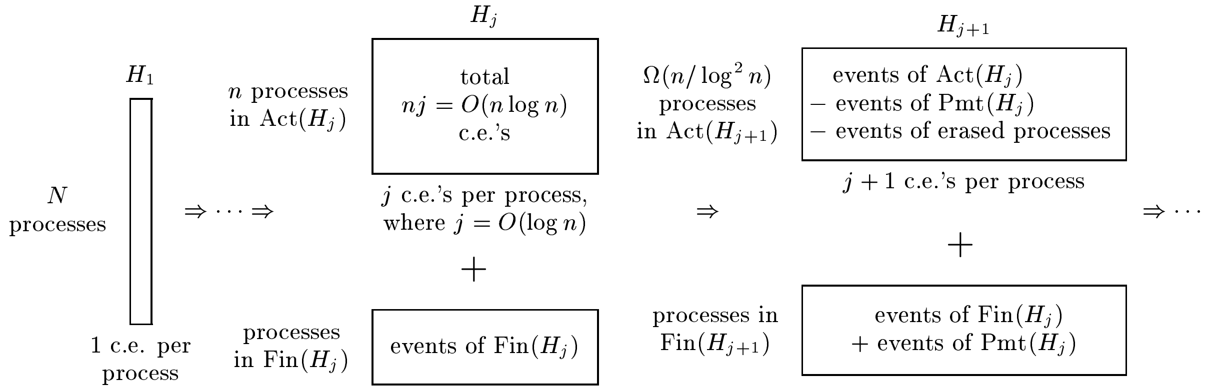

Figure 2: Proof strategy. In this figure, "c.e." stands for "critical event."

v to all the subsequent readers of v. Denote the set of last writers by LW. Because $|V_{\mathrm{HC}}| \leq \frac{(c+1)n}{d \log n}$, we have $|LW| \leq |V_{\mathrm{HC}}| \leq \frac{(c+1)n}{d \log n}$. We then roll the processes in LW forward (*i.e.*, promote their execution) by generating a computation $\overline{G}$ from H_j such that $\mathrm{Inv}(\overline{G}) = P_{\mathrm{HC}} - LW$ and $LW \cup \mathrm{Fin}(H_j)$ is a valid RF-set of $\overline{G}$.[6]

If any process p in $\mathrm{Pmt}(\overline{G})$ $(= LW)$ executes at least $\log n$ critical events before returning to its NCS, then the lower bound easily follows. (It can be shown that $\log n = \Omega(\log N / \log\log N)$. The formal argument is presented in Theorem 2.) Therefore, we can assume that each process in LW performs fewer than $\log n$ critical events while being rolled forward. Because $|LW| \leq \frac{(c+1)n}{d \log n}$, it follows that all the processes in LW can be rolled forward with a total of $O(n)$ critical events.

Appending to $\overline{G}$ a critical event of LW may generate information flow only if it reads a variable v that is written by another process in $\overline{G}$. (RF3) guarantees that if there are multiple writers to v, the last writer in $\overline{G}$ belongs to the RF-set. Because information flow inside the RF-set is allowed, a conflict arises only if there is a single process that writes to v in $\overline{G}$. Thus, each critical event of LW can conflict with, and thus erase, at most one process in $\mathrm{Inv}(\overline{G})$. (Appending a noncritical event to $\overline{G}$ cannot cause any processes to be erased. In particular, if a noncritical remote read by a process p is appended, then p must have previously read the same variable. By (RF3), if the last writer is another process, then that process is in the RF-set.)

Therefore, the entire roll-forward procedure erases $O(n)$ processes from $\mathrm{Inv}(\overline{G})$. Because $|\mathrm{Inv}(\overline{G})| = \Theta(n) - |LW| = \Theta(n)$, we can adjust constant coefficients so that $\Omega(n)$ processes of $\mathrm{Inv}(\overline{G})$ survive after the entire procedure. Thus, we can construct H_{j+1}.

4. LOWER BOUND PROOF

In this section, we present our lower-bound theorem for systems satisfying the Atomicity Assumption, *i.e.*, systems under read/write atomicity. A corresponding lower bound for systems with comparison primitives is established in Sec. 5. We begin by stating several lemmas. Proof sketches for Lemmas 1–4 can be found in an appendix. Lemma 1 states that we can safely "erase" any invisible process.

[6]In fact, the proof of Lemma 5 shows that $\mathrm{Inv}(\overline{G})$ is *not* exactly the same as $P_{\mathrm{HC}} - LW$, because of minor conflicts that can erase $O(n/\log n)$ processes. However, this does not change the asymptotic argument.

Lemma 1 *Consider a computation H in C with a valid RF-set RFS. For any set Y of processes such that $RFS \subseteq Y$, the following hold: $H \mid Y \in C$; RFS is a valid RF-set of $H \mid Y$; an event e in $H \mid Y$ is a critical event iff it is also a critical event in H.* □

The next lemma gives us means for extending a computation by appending noncritical events.

Lemma 2 *Consider a computation H in C with a valid RF-set RFS and a set of processes $Y = \{p_1, p_2, \ldots, p_m\}$, where $Y \subseteq \mathrm{Inv}_{RFS}(H)$. Assume that, for each p_j in Y, there exists a p_j-computation L_{p_j}, such that $H \circ L_{p_j} \in C$ and L_{p_j} has no critical events in $H \circ L_{p_j}$.*

Define L to be $L_{p_1} \circ L_{p_2} \circ \cdots \circ L_{p_m}$. Then, the following hold: $H \circ L \in C$, RFS is a valid RF-set of $H \circ L$, and L contains no critical events in $H \circ L$. □

The next lemma states that if n active processes are competing for entry into their critical sections, then at least $n-1$ of them execute at least one more critical event before entering their critical sections.

Lemma 3 *Let H be a regular computation in C. Define $n = |\mathrm{Act}(H)|$. Then, there exists a subset Y of $\mathrm{Act}(H)$, where $n - 1 \leq |Y| \leq n$, satisfying the following: for each process p in Y, there exist a p-computation L_p and an event e_p of p such that*

- *$H \circ L_p \circ \langle e_p \rangle \in C$;*
- *L_p contains no critical events in $H \circ L_p$;*
- *$e_p \notin \{Enter_p, CS_p, Exit_p\}$;*
- *e_p is a critical event of p in $H \circ L_p \circ \langle e_p \rangle$;*
- *$\mathrm{Fin}(H)$ is a valid RF-set of $H \circ L_p$.* □

According to the next lemma, if there exist promoted processes, we can extend the computation with one more critical event of some promoted process, and erase at most one invisible proess while doing that.

Lemma 4 *Let H be a computation in C with a valid RF-set RFS satisfying $\mathrm{Fin}(H) \subsetneq RFS$ (i.e., $\mathrm{Fin}(H)$ is a proper subset of RFS). Then, there exists a computation G satisfying the following.*

- *RFS is a valid RF-set of G.*
- *G can be written as $H \mid (Y \cup RFS) \circ L \circ \langle e \rangle$, for some choice of Y, L, and e such that*

- Y *is a subset of* $\text{Inv}(H)$ *such that* $|\text{Inv}(H)| - 1 \le |Y| \le |\text{Inv}(H)|$;
- $\text{Inv}(G) = Y$;
- L *is a* Z*-computation that has no critical events in* G*, where* $Z = \text{Pmt}(H)$;
- e *is an event of some process in* Z*, such that* e *is a critical event in* G.

- $\text{Pmt}(G) \subseteq \text{Pmt}(H)$.
- *An event in* $H \,|\, (Y \cup RFS)$ *is a critical event iff it is also a critical event in* H. □

The following theorem is due to Turán [10].

Theorem 1 (Turán) *Let* $\mathcal{G} = (V, E)$ *be an undirected graph, with vertex set* V *and edge set* E*. If the average degree of* $\mathcal{G}$ *is* d*, then an independent set*[7] *exists with at least* $\lceil |V|/(d+1) \rceil$ *vertices.* □

The following lemma provides the induction step that leads to the lower bound in Theorem 2.

Lemma 5 *Let* $\mathcal{S} = (C, P, V)$ *be a mutual exclusion system, and let* H *be a regular computation in* C*. Define* $n = |\text{Act}(H)|$*. Assume that* $n > 1$ *and*

- *each process in* $\text{Act}(H)$ *executes exactly* c *critical events in* H*, where* $c \le \log n - 1$. (1)

Then, one of the following propositions is true.

(Pr1) *There exist a process* p *in* $\text{Act}(H)$ *and a computation* F *in* C *such that*
- $F \circ \langle Exit_p \rangle \in C$;
- F *does not contain* $\langle Exit_p \rangle$;
- p *executes at least* $(c + \log n)$ *critical events in* F.

(Pr2) *There exists a regular computation* G *in* C *such that*
- $\text{Act}(G) \subseteq \text{Act}(H)$;
- $|\text{Act}(G)| \ge \min(\frac{n}{4} - \frac{n}{2\log n} - \frac{1}{2},\ \frac{n-1}{2\cdot(8\log^2 n+1)})$; (2)
- *each process in* $\text{Act}(G)$ *executes exactly* $(c+1)$ *critical events in* G.

Proof: Because H is regular, by Lemma 3, there exists a subset Y of $\text{Act}(H)$ such that

$$n - 1 \le |Y| \le n, \tag{3}$$

and for each process $p \in Y$, there exist a p-computation L_p and an event e_p such that

- $H \circ L_p \circ \langle e_p \rangle \in C$; (4)
- L_p contains no critical events in $H \circ L_p$; (5)
- $e_p \notin \{Enter_p, CS_p, Exit_p\}$; (6)
- e_p is a critical event of p in $H \circ L_p \circ \langle e_p \rangle$; (7)
- $\text{Fin}(H)$ is a valid RF-set of $H \circ L_p$. (8)

If (Pr1) is satisfied by any process in Y, then the theorem is clearly true. Thus, we will assume, throughout the remainder of the proof, that there is no process in Y that satisfies (Pr1). Define $\mathcal{E}_H$ as the set of critical events in H of processes in Y.

$$\mathcal{E}_H = \{e \text{ in } H : e \text{ is critical in } H \text{ and } owner(e) \in Y\}$$

[7] An *independent set* of a graph $\mathcal{G} = (V, E)$ is a subset $V' \subseteq V$ such that no edge in E is incident to two vertices in V'.

Define $\mathcal{E} = \mathcal{E}_H \cup \{e_p : p \in Y\}$, *i.e.*, the set of all "past" and "next" critical events of processes in Y. From (1) and (3), it follows that

$$|\mathcal{E}| = (c+1)|Y| \le (c+1)n. \tag{9}$$

Now define V_{HC}, the set of variables that experience "high contention" (*i.e.*, those that are accessed by "sufficiently many" events in $\mathcal{E}$), as $V_{\text{HC}} = \{v \in V :$ there are at least $4\log^2 n$ events in $\mathcal{E}$ that remotely access $v\}$. Since each event in $\mathcal{E}$ can access at most one remote variable, from (1) and (9), we have

$$|V_{\text{HC}}| \le \frac{|\mathcal{E}|}{4\log^2 n} \le \frac{(c+1)n}{4\log^2 n} \le \frac{n}{4\log n}. \tag{10}$$

Define P_{HC}, the set of processes whose "next" event accesses a variable in V_{HC}, as follows.

$$P_{\text{HC}} = \{p \in Y : e_p \text{ accesses a variable in } V_{\text{HC}}\} \tag{11}$$

We now consider two cases, depending on $|P_{\text{HC}}|$.

Case 1: $|P_{\text{HC}}| < \frac{1}{2}|Y|$ (erasing strategy)

— *In this case, we start with* $Y' = Y - P_{\text{HC}}$*, which consists of at least* $(n-1)/2$ *active processes. We construct a "conflict graph"* $\mathcal{G}$*, made of the processes in* Y'*. By applying Theorem 1, we can find a subset* Z *of* Y' *such that their critical events do not conflict with each other.*

Let $Y' = Y - P_{\text{HC}}$. Then, by (3), we have

$$|Y'| \ge |Y|/2 \ge (n-1)/2. \tag{12}$$

We construct a graph $\mathcal{G} = (Y', E_{\mathcal{G}})$, where each vertex is a process in Y'. To each process y in Y' and each variable $v \in var(e_y)$ that is remote to y, we apply the following rules: **(R1)** if v is local to a process z in Y', then introduce edge (y, z); and **(R2)** if there exists an event $f \in \mathcal{E}$ that remotely accesses v, and if $owner(f) \in Y'$, then introduce edge $(y, owner(f))$.

Because each variable is local to at most one process, and since an event can access at most one remote variable, rule (R1) can introduce at most one edge per process. Since $y \in Y'$, we have $y \notin P_{\text{HC}}$, which, by (11), implies $v \notin V_{\text{HC}}$. Hence, by the definition of V_{HC}, it follows that there are at most $4\log^2 n - 1$ events in $\mathcal{E}$ that access v. Therefore, since an event can access at most one remote variable, rule (R2) can introduce at most $4\log^2 n - 1$ edges per process.

Combining (R1) and (R2), at most $4\log^2 n$ edges are introduced per process. Thus, the average degree of $\mathcal{G}$ is at most $8\log^2 n$. Hence, by Theorem 1, there exists an independent set $Z \subseteq Y'$ such that

$$|Z| \ge \frac{|Y'|}{(8\log^2 n + 1)} \ge \frac{n-1}{2\cdot(8\log^2 n + 1)},$$

where the latter inequality follows from (12).

Next, we construct a computation G, satisfying (Pr2), such that $\text{Act}(G) = |Z|$.

Let $H' = H \,|\, (Z \cup \text{Fin}(H))$. For each $z \in Z$, (4) implies $H \circ L_z \in C$. Hence, by (8), and applying Lemma 1 with 'H' $\leftarrow H \circ L_z$, 'RFS' $\leftarrow \text{Fin}(H)$, and 'Y' $\leftarrow Z \cup \text{Fin}(H)$, we have the following:

- $H' \circ L_z \in C$ (which, by (P1), implies $H' \in C$), and
- an event in $H' \circ L_z$ is critical iff it is also critical in $H \circ L_z$. (13)

By (5), the latter also implies that L_z contains no critical events in $H' \circ L_z$.

Let $m = |Z|$ and index the processes in Z as $Z = \{z_1, z_2, \ldots, z_m\}$. Define $L = L_{z_1} \circ L_{z_2} \circ \cdots \circ L_{z_m}$. By applying Lemma 2 with 'H' $\leftarrow H'$, 'RFS' $\leftarrow \mathrm{Fin}(H)$, and '$Y$' $\leftarrow Z$, we have the following:

- $H' \circ L \in C$,
- $\mathrm{Fin}(H)$ is a valid RF-set of $H' \circ L$, and
- L contains no critical events in $H' \circ L$. (14)

By the definition of H' and L, we also have

- for each $z \in Z$, $(H' \circ L) \,|\, z = (H \circ L_z) \,|\, z$. (15)

Therefore, by (4) and Property (P3), for each $z_j \in Z$, there exists an event e'_{z_j}, such that

- $G \in C$, where $G = H' \circ L \circ E$ and $E = \langle e'_{z_1}, e'_{z_2}, \ldots, e'_{z_m} \rangle$;
- $Rvar(e'_{z_j}) = Rvar(e_{z_j})$, $Wvar(e'_{z_j}) = Wvar(e_{z_j})$, and $owner(e'_{z_j}) = owner(e_{z_j}) = z_j$.

Conditions (RF1)–(RF5), with 'RFS' $\leftarrow \mathrm{Fin}(H)$, can be individually checked to hold in G, which implies that G is a regular computation. Since $Z \subseteq Y' \subseteq Y \subseteq \mathrm{Act}(H)$, combining (1), (13), and (14), it follows that each process in Z executes exactly c critical events in $H' \circ L$.

We now show that every event in E is critical in G, which implies that G is a computation that satisfies (Pr2). Note that, by (7), e_z is a critical event in $H \circ L_z \circ \langle e_z \rangle$. By (6), e_z is not a transition event. By (15), the events of z are the same in both $H \circ L_z$ and $H' \circ L$. Thus, if e_z is a critical read or a "first" critical write in $H \circ L_z \circ \langle e_z \rangle$, then it is also a critical in G. The only remaining case is that e_z writes some variable v remotely, and is critical in $H \circ L_z \circ \langle e_z \rangle$ because of a write to v prior to e_z by another process not in G. However, (RF5) ensures that in such a case there exists some process in $\mathrm{Fin}(H)$ that writes to v before e_z, and hence e_z is also critical in G.

Case 2: $|P_{\mathrm{HC}}| \geq \frac{1}{2}|Y|$ (roll-forward strategy)

— In this case, we start with P_{HC}, which, by (3), consists of at least $(n-1)/2$ active processes. We first erase the processes in K, defined below, to satisfy (RF2). Appending the critical events e_p for each p in $S = P_{\mathrm{HC}} - K$ gives us a non-regular computation $\overline{G}$. We then select a subset LW of P_{HC} as the set of promoted processes. We roll the processes in LW forward, inductively generating a sequence of computations $\overline{G} = G_0, G_1, \ldots, G_k$, where the last computation G_k is regular. We erase at most $n/4$ processes during the procedure, which leaves $\Theta(n)$ active processes in G_k.

Define K, the erased (or "killed") processes, S, the "survivors," and H', the resulting computation, as follows.

$$
\begin{aligned}
K &= \{p \in P : \text{there exists a variable } v \in V_{\mathrm{HC}} \\
&\qquad \text{such that } v \text{ is local to } p\} && (16)\\
S &= P_{\mathrm{HC}} - K && (17)\\
H' &= H \,|\, (S \cup \mathrm{Fin}(H)) && (18)
\end{aligned}
$$

Because each variable is local to at most one process, from (10) and (16), we have

$$|K| \leq n/(4 \log n). \tag{19}$$

Using $S \subseteq P_{\mathrm{HC}} \subseteq Y \subseteq \mathrm{Act}(H)$, we also have

$$\mathrm{Act}(H') = S \subseteq \mathrm{Act}(H) \;\wedge\; \mathrm{Fin}(H') = \mathrm{Fin}(H) \tag{20}$$

Because H is regular, by Lemma 1,

- $H' \in C$, (21)
- $\mathrm{Fin}(H)$ is a valid RF-set of H', and (22)
- an event in H' is a critical event iff it is also a critical event in H. (23)

We can arbitrarily index V_{HC} as $\{v_1, v_2, \ldots, v_s\}$, where $s = |V_{\mathrm{HC}}|$. For $1 \leq j \leq s$, define $Reader_j = \{p \in S : v_j \in Rvar(e_p)\}$, and $Writer_j = \{p \in S : v_j \in Wvar(e_p)\}$. (16) and (17) imply that no variable in V_{HC} is local to any process in S. By the Atomicity Assumption, the sets $\{Reader_j : 1 \leq j \leq s\}$ and $\{Writer_j : 1 \leq j \leq s\}$ form a disjoint partition of S. Without loss of generality, we can assume that the processes in S are indexed from y_1 to y_m, where $m = |S|$, with the following order: processes in $Writer_1$, processes in $Reader_1$, processes in $Writer_2$, processes in $Reader_2$, $\ldots$, processes in $Writer_s$, processes in $Reader_s$.

For each $y \in S$, let $F_y = (H \circ L_y) \,|\, (S \cup \mathrm{Fin}(H))$. (4) implies $H \circ L_y \in C$. Hence, by (8), and applying Lemma 1 with 'H' $\leftarrow H \circ L_y$, 'RFS' $\leftarrow \mathrm{Fin}(H)$, and '$Y$' $\leftarrow S \cup \mathrm{Fin}(H)$, we have the following: $F_y \in C$, and an event in F_y is a critical event iff it is also a critical event in $H \circ L_y$. Since $y \in S$ and L_y is a y-computation, we have $L_y \,|\, (S \cup \mathrm{Fin}(H)) = L_y$. Thus, by (18), $F_y = H' \circ L_y$. Hence, by (5), we have

- $H' \circ L_y \in C$, and (24)
- L_y does not have a critical event in $H' \circ L_y$. (25)

Let $L = L_{y_1} \circ L_{y_2} \circ \cdots \circ L_{y_m}$. We now use Lemma 2, with 'H' $\leftarrow H'$, 'RFS' $\leftarrow \mathrm{Fin}(H)$, and '$Y$' $\leftarrow S$. The antecedent of the lemma follows from (20), (21), (22), (24), and (25). This gives us the following.

- $H' \circ L \in C$;
- $\mathrm{Fin}(H)$ is a valid RF-set of $H' \circ L$;
- L contains no critical events in $H' \circ L$. (26)

By (18) and the definition of L, we also have

- for each $y \in S$, $(H' \circ L) \,|\, y = (H \circ L_y) \,|\, y$. (27)

Therefore, by (4) and Property (P3), for each $y_j \in S$, there exists an event e'_{y_j}, such that

- $\overline{G} \in C$, where $\overline{G} = H' \circ L \circ E$ and $E = \langle e'_{y_1}, e'_{y_2}, \ldots, e'_{y_m} \rangle$;
- $Rvar(e'_{y_j}) = Rvar(e_{y_j})$, $Wvar(e'_{y_j}) = Wvar(e_{y_j})$, and $owner(e'_{y_j}) = owner(e_{y_j}) = y_j$.

By (6) and (26), $L \circ E$ does not contain any transition events. Moreover, by the definition of L and E, $(L \circ E) \,|\, p \neq \langle\rangle$ implies $p \in S$, for each process p. Combining these assertions with (20), we have

$$
\begin{aligned}
&\mathrm{Act}(\overline{G}) = \mathrm{Act}(H' \circ L) = \mathrm{Act}(H') = S \;\wedge \\
&\mathrm{Fin}(\overline{G}) = \mathrm{Fin}(H' \circ L) = \mathrm{Fin}(H') = \mathrm{Fin}(H). && (28)
\end{aligned}
$$

We now state and prove two claims regarding $\overline{G}$.

Claim 1: There exists a valid RF-set RFS of $\overline{G}$ such that $|\mathrm{Pmt}_{RFS}(\overline{G})| \leq n/(4 \log n)$.

Proof of Claim 1: Define LW, the "last writers," as $\{y \in$

$S : y = \mathit{writer}(v, \overline{G})$ for some variable $v \in V_{\mathrm{HC}}\}$, and define $\mathit{RFS} = \mathit{LW} \cup \mathrm{Fin}(H)$. We claim that RFS is a valid RF-set of $\overline{G}$, in which case LW is the set of promoted processes. The conditions (RF1)–(RF5) can be individually checked to hold in $\overline{G}$. We now prove that RFS satisfies Claim 1. Since, by (28), $\mathrm{Fin}(\overline{G}) = \mathrm{Fin}(H)$, $\mathrm{Pmt}_{\mathit{RFS}}(\overline{G}) = \mathit{LW}$ holds by definition. By the definition of LW, we have $|\mathit{LW}| \leq |V_{\mathrm{HC}}|$. Hence, from (10), we have

$$|\,\mathrm{Pmt}_{\mathit{RFS}}(\overline{G})| = |\mathit{LW}| \leq n/(4 \log n). \tag{29}$$

Thus, LW satisfies Claim 1. □

Claim 2: Each process in S $(= \mathrm{Act}(\overline{G}))$ executes exactly $c+1$ critical events in $\overline{G}$.

Proof of Claim 2: Combining (1), (20), (23), and (26), it follows that each process in S executes exactly c critical events in $H' \circ L$. On the other hand, by (7), e_y is a critical event in $H \circ L_y \circ \langle e_y \rangle$. By (27), and using an argument that is similar to that at the end of Case 1, we can prove that each event e'_y in E is a critical event in $\overline{G}$. □

We now let processes in LW finish their execution by inductively appending a critical event of some process in LW, thus generating a sequence of computations $\overline{G} = G_0,\ G_1,\ \ldots,\ G_k$ such that at most one invisible process is erased at each induction step (33), and the last computation G_k is regular (34). G_k is obtained by inductively applying Lemma 4 until $\mathrm{Fin}(G_j) = \mathit{RFS}$ is established; thus, we have the following for each j:

- $G_j \in C$;
- RFS is a valid RF-set of G_j; (30)
- each process in $\mathrm{Inv}_{\mathit{RFS}}(G_j)$ executes exactly $c+1$ critical events in G_j; (31)
- the processes in LW collectively execute exactly $|\mathit{LW}| \cdot (c+1) + j$ critical events in G_j; (32)
- $|\,\mathrm{Inv}_{\mathit{RFS}}(G_{j+1})| \geq |\,\mathrm{Inv}_{\mathit{RFS}}(G_j)| - 1$ if $j < k$; (33)
- $\mathrm{Fin}(G_j) \subsetneq \mathit{RFS}$ if $j < k$, and $\mathrm{Fin}(G_j) = \mathit{RFS}$ if $j = k$. (34)

We now show that $k < n/4$. Assume otherwise. By applying (32) to G_k, it follows that there exists a process $p \in \mathit{LW}$ such that p executes at least $c+1+k/|\mathit{LW}|$ critical events in G_k. Because $k \geq n/4$, by (29), p executes at least $c+1+\log n$ critical events in G_k. From (34) and $p \in \mathit{LW} \subseteq \mathit{RFS}$, we get $p \in \mathrm{Fin}(G_k)$. Hence, by (RF4), the last event of p is Exit_p. Therefore, G_k can be written as $F \circ \langle \mathit{Exit}_p \rangle \circ \cdots$, where F is a prefix of G_k. But then p and F satisfy (Pr1), a contradiction.

Finally, we show that G_k satisfies (Pr2). The following derivation establishes (2).

$$\begin{aligned}
&|\,\mathrm{Act}(G_k)| \\
&\quad = |\,\mathrm{Inv}_{\mathit{RFS}}(G_k)| \\
&\qquad\quad \text{— by (34), } \mathit{RFS} = \mathrm{Fin}(G_k)\text{, which implies that } \mathrm{Act}(G_k) = \mathrm{Inv}_{\mathit{RFS}}(G_k). \\
&\quad \geq |\,\mathrm{Inv}_{\mathit{RFS}}(G_0)| - k \\
&\qquad\quad \text{— by repeatedly applying (33).} \\
&\quad = |\,\mathrm{Act}(\overline{G}) - \mathit{RFS}| - k \\
&\qquad\quad \text{— by the definition of Inv; note that } \overline{G} = G_0. \\
&\quad = |S - \mathit{RFS}| - k \\
&\qquad\quad \text{— by (28).} \\
&\quad = |S - (\mathit{LW} \cup \mathrm{Fin}(H))| - k \\
&\qquad\quad \text{— because } \mathit{RFS} = \mathit{LW} \cup \mathrm{Fin}(H). \\
&\quad = |S - \mathit{LW}| - k \\
&\qquad\quad \text{— because } S \cap \mathrm{Fin}(H) = \{\}. \\
&\quad = |(P_{\mathrm{HC}} - K) - \mathit{LW}| - k \\
&\qquad\quad \text{— by (17).} \\
&\quad \geq |P_{\mathrm{HC}}| - |K| - |\mathit{LW}| - k \\
&\quad \geq |Y|/2 - n/(4\log n) - n/(4 \log n) - n/4 \\
&\qquad\quad \text{— by Case 2, (19), (29), and } k < n/4. \\
&\quad \geq (n-1)/2 - 2 \cdot n/(4 \log n) - n/4 \\
&\qquad\quad \text{— by (3).} \\
&\quad = \frac{n}{4} - \frac{n}{2 \log n} - \frac{1}{2}
\end{aligned}$$

Moreover, (30) and (34) imply that G_k is regular, and (31) implies that each process in $\mathrm{Act}(G_k) = \mathrm{Inv}_{\mathit{RFS}}(G_k)$ executes exactly $c+1$ critical events in G_k. Therefore, G_k satisfies (Pr2). □

Theorem 2 *For any mutual exclusion system $\mathcal{S} = (C, P, V)$, there exist a process p in P and a computation H in C such that $H \circ \langle \mathit{Exit}_p \rangle \in C$, H does not contain Exit_p, and p executes $\Omega(\log N / \log\log N)$ critical events in H, where $N = |P|$.*

Proof: Let $H_1 = \langle \mathit{Enter}_1,\ \mathit{Enter}_2,\ \ldots,\ \mathit{Enter}_N \rangle$, where $P = \{1, 2, \ldots, N\}$. By the definition of a mutual exclusion system, $H_1 \in C$. It is obvious that H_1 is regular and each process in $\mathrm{Act}(H) = P$ has exactly one critical event in H_1. Starting with H_1, we repeatedly apply Lemma 5 and construct a sequence of computations $H_1,\ H_2,\ \ldots,\ H_k$, such that each process in $\mathrm{Act}(H_j)$ has j critical events in H_j. The construction algorithm is shown in Fig. 3.

For each computation H_j such that $1 \leq j < k$, we have the following inequality:

$$n_{j+1} \geq \frac{cn_j}{\log^2 n_j} \geq \frac{cn_j}{\log^2 N},$$

where c is some fixed constant. This in turn implies

$$\log n_{j+1} \geq \log n_j - 2 \log\log N + \log c. \tag{35}$$

By iterating over $1 \leq j < k$, and using $n_1 = N$, (35) implies

$$\log n_k \geq \log N - 2(k-1) \log\log N + (k-1) \log c. \tag{36}$$

We now consider two possibilities, depending on how the algorithm terminates. First, suppose that H_k satisfies $k > \log n_k - 1$. Combining this inequality with (36), we have

$$k > \frac{\log N + 2 \log\log N - \log c - 1}{2 \log\log N - \log c + 1} = \Theta\left(\frac{\log N}{\log\log N}\right).$$

Therefore, each process in $\mathrm{Act}(H_k)$ executes $\Omega(\log N/ \log\log N)$ critical events in H_k. By the Progress property, we can extend H_k to construct a computation that satisfies the theorem.

The other possibility is that $k \leq \log n_k - 1$ holds and H_k satisfies (Pr1). In this case, a process p and a computation F exist such that $F \circ \langle \mathit{Exit}_p \rangle \in C$, F does not contain $\langle \mathit{Exit}_p \rangle$, and p executes at least $k + \log n_k$ critical events in F. By

$H_1 := \langle Enter_1,\ Enter_2,\ \ldots,\ Enter_N \rangle$; $n_1 := N$; $j := 1$;
repeat forever
LOOP INVARIANT: H_j is regular, $n_j = |\,\mathrm{Act}(H_j)|$, and each process in $\mathrm{Act}(H_j)$ executes j critical events in H_j.
  if $j > \log n_j - 1$ **then**
    let $k := j$, and exit the algorithm
  else /* $j \leq \log n - 1$ */
    apply Lemma 5 to H_j;
    if (Pr1) holds **then**
      let $k := j$, and exit the algorithm
    else /* (Pr2) holds */
      — *There exists $Z \subseteq Y$, and a regular comp. G in C, such that* $Z = \mathrm{Act}(G)$, $|Z| = \Omega(n_j / \log^2 n_j)$, and each process in Z executes $j+1$ critical events in G.
      $H_{j+1} := G$; $n_{j+1} := |Z|$; $j := j+1$
  fi fi
od

Figure 3: Algorithm for constructing $H_1, H_2, \ldots, H_k$.

combining $k \leq \log n_k - 1$ with (36), it follows that $\log n_k$ is at least

$$\frac{\log N + 4\log\log N - 2\log c}{2\log\log N - \log c + 1} = \Theta\left(\frac{\log N}{\log\log N}\right).$$

Therefore, computation F satisfies the theorem. □

5. LOWER BOUND FOR SYSTEMS WITH COMPARISON PRIMITIVES

In this section, we show how to adapt our lower bound proof to apply to systems in which processes invoke comparison primitives. A comparison primitive is an atomic operation on a variable v defined as follows.

$Compare\_and\_fg(v, old, new)$
  $temp := v$;
  if $v = old$ **then** $v := f(old, new)$ **fi**;
  return $g(temp, old, new)$

For example, compare-and-swap can be defined such that $f(old,\ new) = new$ and $g(temp,\ old,\ new) = (temp = old)$. We call an invocation of such a primitive a *comparison event*.

When defining critical events, comparison events can be treated much like ordinary writes. For example, the first comparison event on a variable v by a process p is critical; a subsequent comparison event on v by p is critical if v has been updated (by a write or comparison event) by another process since p's last update (by a write or comparison event) of v.

Since a comparison event can access only one remote variable, all of the reasoning in the previous section remains unchanged, except Case 2 of Lemma 5, where processes are rolled forward. Consider the problem of appending to a computation H a set of comparison events $\mathcal{C}$, all on the same variable v. The comparison events in $\mathcal{C}$ can be ordered so that at most once succeeds in updating v. Effectively, that event can be treated like a remote write, and the other events in $\mathcal{C}$ like remote reads. It follows that all information flow is either from $writer(v, H)$ or from the successful comparison event (if any) in $\mathcal{C}$. Hence, we can roll forward at most *two* processes per variable to satisfy (Pr2) of Lemma 5, and the asymptotic bound remains unchanged. Thus, we have the following theorem.

Theorem 3 *For any mutual exclusion system $\mathcal{S} = (C, P, V)$ in which each remote event is either an atomic read, an atomic write, or a comparison primitive invocation, there exist a process p in P and a computation H in C such that $H \circ \langle Exit_p \rangle \in C$, H does not contain $Exit_p$, and p executes $\Omega(\log N / \log\log N)$ critical events in H, where $N = |P|$.* □

6. CONCLUDING REMARKS

We have improved the $\Omega(\log\log N / \log\log\log N)$ lower bound of Cypher to $\Omega(\log N / \log\log N)$. We conjecture that $\Omega(\log N)$ is actually a tight lower bound for the class of algorithms considered in this paper, which would imply that the algorithm of [11] is optimal. We leave this conjecture for future research.

Acknowledgement: We are grateful to Faith Fich for her comments on an earlier draft of this paper.

7. REFERENCES

[1] Y. Afek, P. Boxer, and D. Touitou. Bounds on the shared memory requirements for long-lived and adaptive objects. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pp. 81–89. July 2000.

[2] J. Anderson and J.-H. Yang. Time/contention tradeoffs for multiprocessor synchronization. *Information and Computation*, 124(1):68–84, Jan. 1996.

[3] T. Anderson. The performance of spin lock alternatives for shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):6–16, Jan. 1990.

[4] J. Burns and N. Lynch. Mutual exclusion using indivisible reads and writes. In *Proceedings of the 18th Annual Allerton Conference on Communication, Control, and Computing*, pp. 833–842, 1980.

[5] R. Cypher. The communication requirements of mutual exclusion. In *Proceedings of the Seventh Annual Symposium on Parallel Algorithms and Architectures*, pp. 147–156, 1995.

[6] E. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, 1965.

[7] G. Graunke and S. Thakkar. Synchronization algorithms for shared-memory multiprocessors. *IEEE Computer*, 23:60–69, June 1990.

[8] J. Mellor-Crummey and M. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, Feb. 1991.

[9] E. Styer and G. Peterson. Tight bounds for shared memory symmetric mutual exclusion. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pp. 177–191. Aug. 1989.

[10] P. Turán. On an extremal problem in graph theory (in Hungarian). *Mat. Fiz. Lapok*, 48:436–452, 1941.

[11] J.-H. Yang and J. Anderson. A fast, scalable mutual exclusion algorithm. *Distributed Computing*, 9(1):51–60, Aug. 1995.

Appendix: Proof Sketches for Lemmas 1-4

In this appendix, proof sketches are presented for Lemmas 1–4.

Lemma 1 *Consider a computation H in C with a valid RF-set RFS. For any set Y of processes such that $RFS \subseteq Y$, the following hold: $H \,|\, Y \in C$; RFS is a valid RF-set of $H \,|\, Y$; an event e in $H \,|\, Y$ is a critical event iff it is also a critical event in H.*

Proof sketch: Because H satisfies (RF1), if a process p is not in RFS, no process other than p reads a value written by p. Therefore, $H \,|\, Y \in C$. The conditions (RF1)–(RF5) can be individually checked to hold in $H \,|\, Y$, which implies that RFS is a valid RF-set of $H \,|\, Y$.

To show that an event e in $H \,|\, Y$ is a critical event iff it is also a critical event in H, it is enough to consider critical writes. (The other two cases are straightforward.) The only problematic case for cricial writes is when an event e, which writes v remotely, is critical in H because of a write to v prior to e by another process that does not participate in $H \,|\, Y$ (and hence does not write to v in $H \,|\, Y$). However, (RF5) ensures that in such a case there exists some process in RFS that writes to v before e. $\square$

Lemma 2 *Consider a computation H in C with a valid RF-set RFS and a set of processes $Y = \{p_1, p_2, \ldots, p_m\}$, where $Y \subseteq \mathrm{Inv}_{RFS}(H)$. Assume that, for each p_j in Y, there exists a p_j-computation L_{p_j}, such that $H \circ L_{p_j} \in C$ and L_{p_j} has no critical events in $H \circ L_{p_j}$.*

Define L to be $L_{p_1} \circ L_{p_2} \circ \cdots \circ L_{p_m}$. Then, the following hold: $H \circ L \in C$, RFS is a valid RF-set of $H \circ L$, and L contains no critical events in $H \circ L$.

Proof sketch: For each j, define L^j to be $L_{p_1} \circ L_{p_2} \circ \cdots \circ L_{p_j}$. The lemma can be proved by induction on j. At each induction step, it is assumed that $H \circ L^j \in C$, RFS is a valid RF-set of $H \circ L^j$, and L^j contains no critical events in $H \circ L^j$. Because $L_{p_{j+1}}$ contains no critical events in $H \circ L_{p_{j+1}}$, it can be appended to $H \circ L^j$ and get $H \circ L^{j+1}$ for the next induction step. (As mentioned at the end of Sec. 3, appending a noncritical event cannot cause any undesired information flow from invisible processes to processes in RFS.) $\square$

Lemma 3 *Let H be a regular computation in C. Define $n = |\mathrm{Act}(H)|$. Then, there exists a subset Y of $\mathrm{Act}(H)$, where $n - 1 \leq |Y| \leq n$, satisfying the following: for each process p in Y, there exist a p-computation L_p and an event e_p of p such that*

- *$H \circ L_p \circ \langle e_p \rangle \in C$;* (37)
- *L_p contains no critical events in $H \circ L_p$;* (38)
- *$e_p \notin \{Enter_p, CS_p, Exit_p\}$;* (39)
- *e_p is a critical event of p in $H \circ L_p \circ \langle e_p \rangle$;*
- *$\mathrm{Fin}(H)$ is a valid RF-set of $H \circ L_p$.*

Proof sketch: First, we construct, for each process p in $\mathrm{Act}(H)$, a computation L_p and an event e_p that satisfy (37) and (38). Then, we show that every event e_p thus constructed, except at most one, satisfies (39). The other conditions can be easily proved and will be omitted here.

Define $H_p = H \,|\, (\{p\} \cup \mathrm{Fin}(H))$. Because H is regular, $\mathrm{Fin}(H)$ is a valid RF-set of H. Hence, by Lemma 1, H_p is in C and $\mathrm{Fin}(H)$ is a valid RF-set of H_p. Since $p \in \mathrm{Act}(H)$, we have $\mathrm{Act}(H_p) = \{p\}$ and $\mathrm{Fin}(H_p) = \mathrm{Fin}(H)$. Therefore, by the Progress property, there exists a p-computation F_p such that $H_p \circ F_p \circ \langle CS_p \rangle \in C$. If F_p has a critical event, then let e'_p be the first critical event in F_p, and let L_p be the prefix of F_p that precedes e'_p (*i.e.*, $F_p = L_p \circ \langle e'_p \rangle \circ \cdots$). Otherwise, define L_p to be F_p and e'_p to be CS_p.

Now we have a p-computation L_p and an event e'_p of p, such that $H_p \circ L_p \circ \langle e'_p \rangle \in C$, in which L_p has no critical events and e'_p is a critical event. Because L_p has no critical events in $H_p \circ L_p$, it can be shown that $H \circ L_p \in C$ and that L_p has no critical events in $H \circ L_p$. Because $H \circ L_p$ and $H_p \circ L_p$ are equivalent with respect to p, by Property (P3), there exists an event e_p of p such that $Rvar(e_p) = Rvar(e'_p)$, $Wvar(e_p) = Wvar(e'_p)$, and $H \circ L_p \circ \langle e_p \rangle \in C$.

We now claim that at most one process in $\mathrm{Act}(H)$ does not satisfy (39). Because $p \in \mathrm{Act}(H)$ and H is regular, e_p cannot be $Enter_p$ or $Exit_p$. By the Exclusion property, there can be at most one $p \in \mathrm{Act}(H)$ such that $e_p = CS_p$. $\square$

Lemma 4 *Let H be a computation in C with a valid RF-set RFS satisfying $\mathrm{Fin}(H) \subsetneq RFS$ (i.e., $\mathrm{Fin}(H)$ is a proper subset of RFS). Then, there exists a computation G satisfying the following.*

- *RFS is a valid RF-set of G.*
- *G can be written as $H \,|\, (Y \cup RFS) \circ L \circ \langle e \rangle$, for some choice of Y, L, and e such that: Y is a subset of $\mathrm{Inv}(H)$ such that $|\mathrm{Inv}(H)| - 1 \leq |Y| \leq |\mathrm{Inv}(H)|$; $\mathrm{Inv}(G) = Y$; L is a Z-computation that has no critical events in G, where $Z = \mathrm{Pmt}(H)$; and e is an event of some process in Z, such that e is a critical event in G.*
- $\mathrm{Pmt}(G) \subseteq \mathrm{Pmt}(H)$.
- *An event in $H \,|\, (Y \cup RFS)$ is a critical event iff it is also a critical event in H.*

Proof sketch: Arbitrarily choose a process $r \in Z$. Let $H' = H \,|\, RFS$. Then, by the definition of an active process, $\mathrm{Act}(H') = \mathrm{Act}(H) \cap RFS = \mathrm{Pmt}(H) = Z$. By Lemma 1, $H' \in C$ and RFS is a valid RF-set of H'. Thus, by (RF4), $value(stat_r, H')$ is either $entry$ or $exit$, and for any $q \in \mathrm{Fin}(H) = \mathrm{Fin}(H')$, $value(stat_q, H') = ncs$ holds.

Therefore, by applying the Progress property, we can construct a Z-computation F such that $H' \circ F \circ \langle \bar{e} \rangle \in C$, where $\bar{e}$ is either CS_r or $Exit_r$. If F has a critical event, then let e' be the first critical event in F, and let L be the prefix of F that precedes e' (*i.e.*, $F = L \circ \langle e' \rangle \circ \cdots$). Otherwise, define L to be F and e' to be $\bar{e}$. Let $owner(e') = p$. Because F is a Z-computation, $p \in Z$.

Now we have a Z-computation L and an event e' of $p \in Z$, such that $H' \circ L \circ \langle e' \rangle \in C$, L has no critical events in $H' \circ L \circ \langle e' \rangle$, and e' is a critical event in $H' \circ L \circ \langle e' \rangle$. It can be shown that $H \circ L \in C$ and that L has no critical events in $H \circ L$. (Informally, this follows because H and H' are equivalent with respect to Z.) Because $H \circ L$ and $H' \circ L$ are equivalent with respect to p, by Property (P3), there exists an event e'' of p such that $Rvar(e'') = Rvar(e')$, $Wvar(e'') = Wvar(e')$, and $H \circ L \circ \langle e'' \rangle \in C$.

Because e' is a critical event in $H' \circ L \circ \langle e' \rangle$ and e'' accesses the same variables as e', it can be shown that e'' is a critical event in $H \circ L \circ \langle e'' \rangle$. Let v be the remote variable accessed by e''. If v is local to a process q in $\mathrm{Inv}(H)$, or if $q = writer(v, H \circ L)$ is in $\mathrm{Inv}(H)$, then we can "erase" process q and construct a computation G that satisfies the consequent of Lemma 4. (If both conditions hold simultaneously, then by (RF2), q is identical in both cases.) $\square$