Open Problem Resolved: The "Two" in Existing Multiprocessor PI-Blocking Bounds is Fundamental

Shareef Ahmed
University of North Carolina at Chapel Hill, USA

James H. Anderson
University of North Carolina at Chapel Hill, USA

Abstract

The goal of a real-time locking protocol is to reduce any *priority-inversion* blocking (pi-blocking) a task may incur while waiting to access a shared resource. For mutual-exclusion sharing on an m-processor platform, the best existing lower bound on per-task pi-blocking under suspension-oblivious analysis is a (trivial) lower bound of $(m-1)$ request lengths under any job-level fixed-priority (JLFP) scheduler. Surprisingly, most *asymptotically* optimal locking protocols achieve a per-task pi-blocking upper bound of $(2m-1)$ request lengths under JLFP scheduling, even though a range of very different mechanisms are used in these protocols. This paper closes the gap between these existing lower and upper bounds by establishing a lower bound of $(2m-2)$ request lengths under global fixed-priority (G-FP) and global earliest-deadline-first (G-EDF) scheduling. This paper also shows that worst-case per-task pi-blocking can be arbitrarily close to $(2m-1)$ request lengths for locking protocols that satisfy a certain property that is met by most (if not all) existing locking protocols. These results imply that most known asymptotically optimal locking protocols are *almost* truly optimal (not just asymptotic) under G-FP and G-EDF scheduling.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases Real-Time Systems, Real-Time Synchronization, Multiprocessors

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.9

Funding Supported by NSF grants CPS 2038960, CPS 2038855, CNS 2151829, and CPS 2333120.

1 Introduction

A real-time locking protocol is *asymptotically optimal* if it provides asymptotically optimal bounds on per-task priority-inversion blocking (pi-blocking). Such a *per-task* bound upper bounds pi-blocking for any job (invocation) of said task. For mutual-exclusion (mutex) sharing, a number of suspension-based multiprocessor real-time locking protocols are known that are asymptotically optimal under any job-level fixed-priority (JLFP) scheduler [1,3,7,8]. Under the assumption that each job contains one lock request, most of these locking protocols provide a per-task pi-blocking bound of $2m-1$ request lengths on an m-processor platform under suspension-oblivious (s-oblivious) schedulability analysis, where suspension time is analytically treated as computation time [7–9]. The lone exception is a recently presented protocol that provides a per-task pi-blocking bound of $m-1$ request lengths under first-in-first-out (FIFO) scheduling [1].

In contrast to most protocols, the best existing lower bound under s-oblivious analysis is a (trivial) lower bound of $m-1$ request lengths [7]. This lower bound, which applies to any job-level fixed-priority (JLFP) scheduler, is obtained by examining the simple scenario where m lock requests are active on m processors and noting the pi-blocking time of the last request to be satisfied. Thus, the locking protocol given in [1] is the only known *truly optimal* suspension-based locking protocol, and it is optimal only when employed with FIFO scheduling. Therefore, a gap exists between the existing lower and upper bounds on s-oblivious pi-blocking under all non-FIFO JLFP schedulers. This raises a natural question:

is an s-oblivious pi-blocking bound of $2m - 1$ *request lengths fundamental under any non-FIFO JLFP scheduler?* This question has been an open problem since the discovery of the first asymptotically optimal suspension-based locking protocol in 2010 [7].

In this paper, we demystify the optimality of suspension-based locking protocols under s-oblivious analysis by improving upon the existing lower-bound result. In particular, we show that worst-case per-task s-oblivious pi-blocking is at least $2m - 2$ request lengths under global fixed-priority (G-FP) and global earliest-deadline-first (G-EDF) scheduling, as well as a broader subset of JLFP schedulers. This result implies that most existing asymptotically optimal locking protocols are within a single request length of being *truly* optimal under G-FP and G-EDF scheduling. We also show that, under G-FP and G-EDF scheduling, worst-case per-task s-oblivious pi-blocking can be arbitrarily close to $2m - 1$ request lengths when the employed locking protocol satisfies a certain property. This property is satisfied by most (if not all) existing locking protocols.

When "thinking asymptotically," a factor of two may seem insignificant. However, use cases exist where doubling pi-blocking costs in schedulability analysis can have serious negative consequences. For example, a common approach for predictably sharing a hardware accelerator such as a graphics processing unit (GPU) is to use a mutex locking protocol to ensure that each task has exclusive access when performing an accelerator operation. In the case of a GPU, the corresponding critical sections can be rather large, so doubling pi-blocking costs in analysis can easily make a system unschedulable.

Pi-blocking under FIFO vs. non-FIFO JLFP scheduling. As mentioned earlier, a recently proposed locking protocol achieves a pi-blocking bound of $m - 1$ request lengths under FIFO scheduling [1]. The key benefit FIFO scheduling offers is that, once a job achieves high enough priority to be scheduled, it continues to have high enough priority to be scheduled until its completion. This is not true under any non-FIFO JLFP scheduler. In fact, a job's priority can repeatedly switch between being high enough to be scheduled and being too low to be scheduled until its completion. Our lower-bound proof exploits this phenomenon to show that a set of jobs can gradually accumulate pi-blocking so that one job in that set must be pi-blocked for $2m - 2$ request lengths.

Contributions. Our contributions are twofold.

First, we give a lower bound of $2m - 2$ request lengths on per-task s-oblivious pi-blocking under G-FP and G-EDF scheduling, as well as a broader subset of JLFP schedulers. In particular, we show that there exists a task system with a large enough task count where a job must incur s-oblivious pi-blocking for the duration of at least $2m - 2$ request lengths.

Second, we show that, when a locking protocol adheres to certain conditions, which most protocols would naturally adhere to, worst-case per-task s-oblivious pi-blocking can be arbitrarily close to $2m - 1$ request lengths.

Organization. In the rest of this paper, we provide needed background (Sec. 2), present the above-mentioned lower-bound results (Secs. 3 and 4), more fully review related work (Sec. 5), and conclude (Sec. 6).

2 Background

In this section, we provide needed definitions. Tbl. 1 summarizes the notation we use.

Table 1 Notation summary.

Symbol	Meaning	Symbol	Meaning
n	Number of tasks	ℓ	a shared resource
m	Number of processors	L_i	Maximum request length for ℓ by τ_i
τ_i	i^{th} task	L_{max}	$\max_{1\le i\le n}\{L_i\}$
$J_{i,j}$	j^{th} job of τ_i	$\mathcal{R}$	A request
T_i	Period of τ_i	$r_{i,j}$	Release time of $J_{i,j}$
C_i	WCET of τ_i	$f_{i,j}$	Finish time of $J_{i,j}$
D_i	Relative deadline of τ_i	Y_i	RPP of τ_i
u_i	Utilization of τ_i	$y_{i,j}$	PP of $J_{i,j}$
T_{min}	$\min_{1\le i\le n}\{T_i\}$	T_{max}	$\max_{1\le i\le n}\{T_i\}$

Task model. We consider a system of n sporadic *tasks* $\tau_1, \tau_2, \ldots, \tau_n$ to be scheduled on m identical processors. Each task τ_i releases a potentially infinite sequence of *jobs* $J_{i,1}, J_{i,2}, \ldots$. (We omit job indices if they are irrelevant.) The *period* T_i of task τ_i is the minimum separation between the release times of two consecutive jobs of τ_i. Each task has a *relative deadline* D_i. We assume that each task τ_i has an implicit deadline.[1] Thus, $D_i = T_i$ holds. The maximum and minimum period among all tasks are denoted by T_{max} and T_{min}, respectively. Task τ_i has a *worst-case execution time* (WCET) denoted C_i. The *utilization* of τ_i is defined as $u_i = C_i/T_i$. The *release time* (resp., *finish time*) of a job $J_{i,j}$ is given by $r_{i,j}$ (resp., $f_{i,j}$). $J_{i,j}$ is *pending* at time t if $r_{i,j} \le t < f_{i,j}$. A pending job is either *ready* (when it can be scheduled) or *suspended* (when it cannot be scheduled).

Multiprocessor scheduling. Under *job-level fixed-priority* (JLFP) scheduling, each job has a fixed priority throughout its execution, but a task's priority may change over time. Common JLFP schedulers include the earliest-deadline-first (EDF), FIFO, and fixed-priority (FP) scheduling algorithms. In applying such schedulers, a *clustered* scheduling approach is often assumed where the m processors are divided into c equal-sized clusters of m/c processors each. Each task is assigned to a specific cluster and may execute on any processor in that cluster. *Partitioned* ($c = m$) and *global* ($c = 1$) scheduling are special cases of clustered scheduling. The results of this paper pertain to global JLFP schedulers. Under such a scheduler, the m highest-priority ready jobs (if that many exist) are scheduled at any time.

Global-EDF-like scheduling. Global-EDF-like (GEL) scheduling [12] is a special case of JLFP scheduling that generalizes both G-EDF and FIFO scheduling. Under GEL scheduling, each task has a *relative priority point* (RPP) Y_i. The *priority point (PP)* of a job $J_{i,k}$, denoted by $y_{i,k}$, is defined as

$$y_{i,k} = r_{i,k} + Y_i. \tag{1}$$

Under GEL scheduling, a job's priority is determined by its PP. If $y_{i,k} < y_{j,\ell}$, then job $J_{i,k}$ has higher priority than job $J_{j,\ell}$. The lower-bound results of this paper apply not only to G-EDF but also to a subset of non-FIFO GEL scheduler. (They apply to G-FP as well.)

[1] The results of this paper do not depend on the choice of deadline constraints. Implicit deadlines are assumed for simplicity.

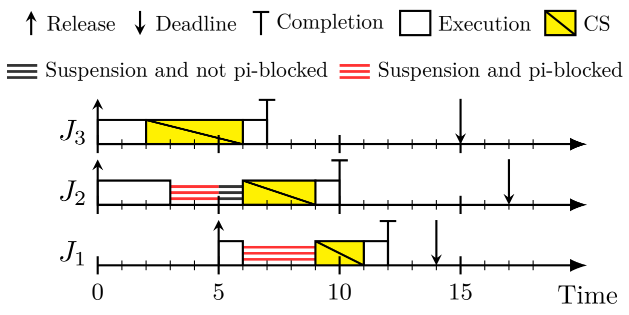

Figure 1 A schedule illustrating s-oblivious pi-blocking.

Resource model. We consider a system that has a single shared resource ℓ.[2] Resource ℓ has a *mutual exclusion* (mutex) sharing constraint: ℓ can be held by at most one job at any time. When a job J_i requires resource ℓ, it *issues* a *request* $\mathcal{R}$ for ℓ. $\mathcal{R}$ is *satisfied* as soon as J_i *holds* ℓ, and *completes* when J_i *releases* ℓ. $\mathcal{R}$ is *active* from its issuance to its completion. J_i must *wait* until $\mathcal{R}$ can be satisfied if it is held by another job. Generally speaking, it may do so either by *busy-waiting* (or *spinning*) in a tight loop, or by being *suspended* by the operating system until $\mathcal{R}$ is satisfied. However, in this paper, we consider only suspension-based waiting. We assume that a job J_i must be scheduled on a processor at any time instant when it holds resource ℓ and has a high-enough priority to be scheduled.[3] A resource access is called a *critical section* (CS).

As only one resource exists in the systems we consider, we implicitly assume that each job can request or hold at most one resource at a time, *i.e.*, resource requests are non-nested. To ease calculating pi-blocking across an entire job, we assume that each job includes at most one request for ℓ. Letting L_i denote the maximum length of all requests by τ_i's jobs, we define $L_{max} = \max_{1 \le i \le n}\{L_i\}$. We assume all L_i's to be constant.

Priority inversions. *Priority-inversion blocking* (or *pi-blocking*) occurs when a job's execution is delayed and this delay cannot be attributed to the execution of higher-priority jobs. On multiprocessors, the formal definition of pi-blocking actually depends on how schedulability analysis is done. Of relevance to suspension-based locks, schedulability analysis may be either *suspension-oblivious* (*s-oblivious*) or *suspension-aware* (*s-aware*) [7]. Under s-oblivious analysis (the focus of this work), pi-blocking is formally defined as follows.

▶ **Definition 1** ([7]). *Under s-oblivious schedulability analysis, a job J_i incurs s-oblivious pi-blocking at time t if J_i is* ***pending*** *but not scheduled and fewer than m higher-priority jobs are* ***pending****.*

By Def. 1, a job suffers s-oblivious pi-blocking at any time when it is one of the m highest priority jobs but it is not scheduled. In the rest of the paper, by the term "pi-blocking," we will mean s-oblivious pi-blocking unless otherwise specified.

▶ **Example 2.** Fig. 1 illustrates a G-EDF schedule of three jobs J_1, J_2, and J_3 on two processors. Job J_3 acquires resource ℓ at time 2. Job J_2 issues a request for ℓ at time 3, and it is suspended from time 3 to time 6. Since there is only one job with higher priority than

[2] To establish a lower bound, we do not need to consider multiple shared resources. Nonetheless, our results apply when multiple shared resources are present.

[3] This is a common assumption in work on synchronization. It is needed for shared data, but may be pessimistic for other shared resources such as I/O devices.

Table 2 Asymptotically optimal locking protocols for mutex locks under s-oblivious analysis.

Scheduling	Protocol	Release blocking	Request blocking
Global JLFP	OMLP [7]	0	$(2m-1)L_{max}$
Clustered JLFP	C-OMLP [8]	mL_{max}	$(m-1)L_{max}$
Clustered JLFP	OMIP [3]	0	$(2m-1)L_{max}$
Clustered FIFO	OLP-F [1]	0	$(m-1)L_{max}$

J_2 during time interval $[3, 5)$, by Def.1, J_2 incurs s-oblivious pi-blocking during this time interval. At time 5, job J_1, which has higher priority than both J_2 and J_3, is released. Since there are $m = 2$ jobs with higher-priority than J_2 during time interval $[5, 6)$, J_2 does not incur s-oblivious pi-blocking during this time interval. ◀

Request vs. release blocking. Under a given real-time locking protocol, a job may experience pi-blocking each time it requests a resource—this is called *request blocking*. Additionally, a job may suffer pi-blocking upon its release and each time it releases a resource—this is called *release blocking*. Release blocking occurs due to the usage of certain *progress mechanisms* that ensure that a resource-holding job is sure to be scheduled if it pi-blocks some other job. Non-preemptive execution is a simple example of such a mechanism. However, more complex such mechanisms exist. A notable example is *priority donation*, which avoids problematic preemptions of resource-holding jobs by carefully manipulating job priorities [8].

Blocking complexity. Request lengths are unavoidable in assessing maximum pi-blocking, as a request-issuing job may have to wait for the whole duration of a current resource-holder's resource access. As a consequence, maximum pi-blocking bounds are usually expressed as an integer multiple of the maximum request length, *i.e.*, the number of requests that are satisfied while a resource-requesting job is pi-blocked.

Asymptotically optimal locking protocols. For mutex locks, a lower bound of $m-1$ request lengths on per-request s-oblivious pi-blocking under any JLFP scheduler is known [7]. Thus, under s-oblivious analysis, an asymptotically optimal locking protocol achieves $O(m)$ per-job pi-blocking (taking the number of resource requests per job as a constant). Locking protocols such as the OMLP [7], the OMIP [3], and the C-OMLP [8] are asymptotically optimal under any JLFP scheduling algorithm. The OLP-F [1] achieves per-request pi-blocking of $m-1$ request lengths under FIFO scheduling. Thus, the OLP-F is an optimal locking protocol under FIFO scheduling. Tbl. 2 provides a summary of existing asymptotically optimal locking protocols. Note that, for the C-OMLP, the pi-blocking bound of $2m-1$ request lengths, as mentioned in Sec. 1, comes from a combination of release and request blocking. Note also that global scheduling is a special case of clustered scheduling, so any lower bound obtained assuming global scheduling is applicable to a protocol designed for clustered scheduling.

3 General Lower Bound On Pi-Blocking

In this section, we give a lower bound of $2m-2$ request lengths on pi-blocking under a set of JLFP schedulers that includes G-EDF and G-FP scheduling. Specifically, we demonstrate the existence of a task system and a corresponding release sequence such that a job incurs pi-blocking for at least $2m-2$ request lengths. In the rest of the section, we first describe said task system (Sec. 3.1), then show how a job in that system incurs pi-blocking for at least

$2m-2$ request lengths assuming a certain assignment of job priorities (Sec. 3.2), and finally show how that priority assignment can be realized under different schedulers (Sec. 3.3).

3.1 Task System

Let Γ be a set of n tasks that are scheduled on m processors. The tasks in Γ consists of m disjoint groups of tasks. The first group of tasks consists of $2m-2$ tasks $\{\tau_1^1, \tau_2^1, \ldots, \tau_{2m-2}^1\}$. Each of the remaining $m-1$ groups of tasks consists of m tasks. We denote the set of tasks in the i^{th} group ($i > 1$) by $\{\tau_1^i, \tau_2^i, \ldots, \tau_m^i\}$. Thus, the total number of tasks is $n = (2m-2) + (m-1)m = m^2 + m - 2$.

Each job of each task issues a request for resource ℓ as soon as the job is released. The request length of each request is L. Each job completes as soon as its request for resource ℓ completes. Thus, $C_i = L$ holds. To establish our lower bound, we do not need any specific assignment of periods to the tasks in Γ, unless a period assignment is required to assign job priorities (as in G-EDF). We will show how periods can be assigned (if needed) in Sec. 3.3.

Feasibility of Γ. In the following lemma, we show that Γ can be feasibly scheduled under any JLFP scheduler when the minimum period of all tasks is large enough.

▶ **Lemma 3.** *If $T_{min} \geq nL$, then there exists a locking protocol under which Γ can be feasibly scheduled under any JLFP scheduler.*

Proof. We show that Γ is schedulable under any JLFP scheduler when lock requests are satisfied in FIFO order. Assume for a contradiction that Γ is not schedulable in a JLFP schedule $\mathcal{S}$ where lock requests are satisfied in FIFO order. Assume that job release times are totally ordered by introducing consistent tie-breaking if needed. Let J be the job with the earliest release time that misses its deadline in $\mathcal{S}$. Let t_r be the time instant when J is released. By the definition of J, no job with a deadline at or before time t_r misses its deadline. Since each task has an implicit deadline, there is at most one pending job per task at time t_r. Thus, there are at most n pending jobs (including J) at time t_r. Since requests are satisfied in FIFO order, J's request completes by the time all these n requests are complete. Since JLFP scheduling is work-conserving, these n requests complete by time $t_r + nL$. Since each job finishes execution when the request it issues completes, J completes execution by time $t_r + nL \leq t_r + T_{min}$. Thus, J completes execution by its deadline, a contradiction. ◀

Release sequence. We now describe a release sequence Γ_{seq} for tasks in Γ. Our lower-bound proof only requires one job of each task. For ease of notation, we denote the job of τ_i^j by J_i^j. These jobs are released according to the following rules.

R1 Jobs $G^1 = \{J_1^1, J_2^1, \ldots, J_{2m-2}^1\}$ are released at time 0.
R2 Let t_i be the time instant when the i^{th}-satisfied request is complete. We define $t_0 = 0$. At time t_{km-1}, jobs $G^{k+1} = \{J_1^{k+1}, J_2^{k+1}, \ldots, J_m^{k+1}\}$ are released.
R3 No task releases a new job until all jobs in $G^1 \cup G^2 \cup \cdots \cup G^m$ complete execution.

Note that Rules R1–R3 do not require an unsatisfied request to be satisfied immediately after a request completion. Thus, at time t_i, a locking protocol may insert delay before satisfying the next request.

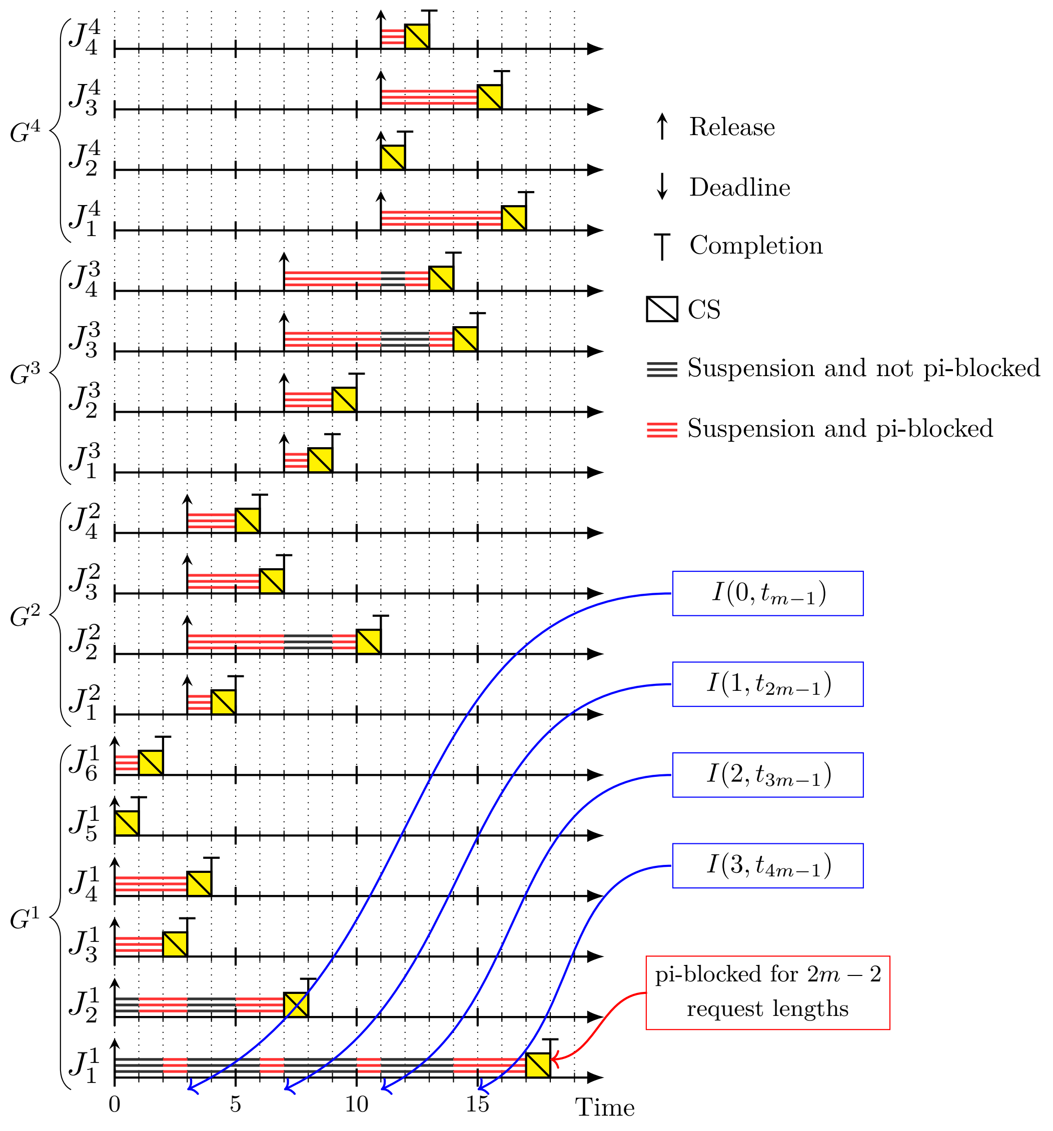

Figure 2 Release sequence by Rules R1–R3 for $m = 4$. Job priorities increase from bottom to top.

Job priorities. We assume job priorities satisfy the following rules. We will later illustrate how this priority ordering can be achieved under different scheduling algorithms in Sec. 3.3.

P1 For any $u > v$, job J_i^u has higher priority than job J_j^v, *i.e.*, the jobs in G^u have higher priority than the jobs in G^v

P2 For any $i > j$, job J_i^u has higher priority than job J_j^u.

▶ **Example 4.** Fig. 2 depicts a release sequence according to Rules R1– R3 for $m = 4$. By Rule R1, jobs J_1^1–J_6^1 are released at time 0. Since the $(m-1)^{st} = 3^{rd}$ satisfied request (J_3^1's request) completes at time 3, by Rule R2, jobs J_1^2–J_4^2 are released at time 3. Similarly, by Rule R2, jobs J_1^3–J_4^3 are released at time 7, as the $(2m-1)^{st} = 7^{th}$ satisfied request (J_3^2's request) completes then.

In Fig. 2, the time intervals when a job experiences s-oblivious pi-blocking are marked red. For example, during $[0, 1)$, jobs J_6^1, J_4^1, and J_3^1 suffer pi-blocking, as they are among the top $m = 4$ jobs by priority during this time interval (by Rule P2). In contrast, a job does not experience pi-blocking during black-marked intervals. For example, jobs J_1^1 and J_2^1 are suspended but not pi-blocked during time interval $[0, 1)$. Note how their suspension time here is "negated" as pi-blocking time by the presence of $m = 4$ higher-priority jobs that are either executing or suspended. ◀

3.2 Lower-Bound Proof

In this section, we prove the following theorem.

▶ **Theorem 5.** *There is a job J_i^j in Γ_{seq} that incurs pi-blocking for at least $(2m-2)L$ time units when job priorities are determined by Rules P1 and P2.*

To prove Theorem 5, our goal is to show that there exists a time instant when $m-1$ jobs are pending with unsatisfied requests and each such job has already incurred pi-blocking of at least mL time units. If no higher-priority jobs are released at or after such a time instant, then at least one job must incur pi-blocking for at least an additional $(m-2)L$ time units (as any locking protocol must impart some ordering of these requests). To prove this, we show that an invariant holds at times $t_{m-1}, t_{2m-1}, \ldots, t_{m^2-1}$. We first define some notation.

▶ **Definition 6.** *Let $B(u,t)$ denote the number of pending jobs at time t that have incurred pi-blocking of at least uL time units by time t.*

Using $B(u,t)$, we define the following predicate.

$$I(p,t) \equiv \bigwedge_{u=m,v=p}^{u=1,v=m+p-1} B(u,t) \geq \min\{v, m-1\} \tag{2}$$

Thus, $I(p,t)$ is true if and only if $B(m,t) \geq \min\{p, m-1\} \wedge B(m-1,t) \geq \min\{p+1, m-1\} \wedge \cdots \wedge B(1,t) \geq \min\{m+p-1, m-1\}$. This means, at time t, there exist $\min\{p, m-1\}$ pending jobs that have incurred at least mL time units of pi-blocking, $\min\{p+1, m-1\}$ pending jobs that have incurred at least $(m-1)L$ time units of pi-blocking, and so on.

▶ **Example 4** (Cont'd). Consider the schedule in Fig. 2. At time 3, the only pending jobs that were released previously (and thus could have been pi-blocked) are J_4^1, J_2^1, and J_1^1. These jobs have incurred pi-blocking for $3L, 2L$, and L time units, respectively. Thus, there is one (resp., two, three) job(s) that has (have) incurred pi-blocking for $3L$ (resp., $2L$, L) time units by time 3 (and none that have experience pi-blocking for $4L$ time units). Thus, $B(4,3) = 0, B(3,3) = 1, B(2,3) = 2$, and $B(1,3) = 3$ each hold. Hence, $I(0,3)$ holds. Similarly, by time 7, J_2^2, J_2^1, and J_1^1, which were all released previously and are still pending, have incurred pi-blocking for $4L, 4L$, and $2L$ time units, respectively. Thus, $B(4,7) = 2$, as there are two pending jobs at time 7 that have incurred pi-blocking for at least $4L$ time units by time 7. Similarly, $B(3,7) = 2$, $B(2,7) = 3$, and $B(1,7) = 3$ each hold. Thus, we have $B(4,7) \geq \min\{1,3\} \wedge B(3,7) \geq \min\{2,3\} \wedge B(2,7) \geq \min\{3,3\} \wedge B(1,7) \geq \min\{4,3\}$. Since $m = 4$, by (2), $I(1,7)$ holds. ◀

To prove Theorem 5, our goal is to show that $I(m-1,t)$ holds at some time instant t in Γ_{seq} under any suspension-based locking protocol. By (2), this would imply that $B(m,t) \geq (m-1)$ holds. Thus, there exists a time instant when $m-1$ pending jobs have already incurred mL time units of pi-blocking. In the following lemma, we first show that there exists a time instant t when $I(0,t)$ holds.

▶ **Lemma 7.** *$I(0, t_{m-1})$ holds.*

Proof. Time instant $t_{m-1} = t_3$ in Fig. 2 illustrates this lemma. By (2), we need to prove that $B(m, t_{m-1}) \geq 0 \wedge B(m-1, t_{m-1}) \geq 1 \wedge \cdots \wedge B(1, t_{m-1}) \geq m-1$ holds. Since no jobs in $G^2 \cup G^3 \cup \cdots \cup G^m$ are released before time t_{m-1}, only jobs in G^1 can incur pi-blocking before time t_{m-1}. By Rule R2, the first i satisfied requests and their corresponding jobs complete by time t_i. Thus, at time t_{m-1}, there are $2m-2-(m-1) = m-1$ pending jobs of G^1. Let $J_{hp(i)}$ be the i^{th} highest-priority pending job among the jobs of G^1 at time t_{m-1}. We prove the lemma by first establishing the following claim.

▷ Claim 8. Job $J_{hp(i)}$ incurs pi-blocking for at least $(m-i)L$ time units by time t_{m-1}.

Proof. We first show that job $J_{hp(i)}$ is one of the m highest-priority pending jobs during $[t_{i-1}, t_{m-1})$. Note that $i \leq m-1$ (hence, $i-1 < m-1$) holds, as there are $m-1$ pending jobs (including $J_{hp(i)}$) of G^1 at time t_{m-1}. By the definition of $J_{hp(i)}$, among the $m-1$ pending jobs of G at time t_{m-1}, exactly $m-1-i$ jobs have lower priority than $J_{hp(i)}$.

By Rules R1 and R2, no job is released during $(0, t_{m-1})$. Thus, by the definition of time instant t_{i-1}, there are $2m-2-(i-1) = 2m-i-1$ pending jobs at time t_{i-1}. Since $J_{hp(i)}$ has higher priority than $m-i-1$ jobs of G^1 at time t_{m-1}, $J_{hp(i)}$ is among the $2m-i-1-(m-i-1) = m$ highest-priority pending jobs at time t_{i-1}. $J_{hp(i)}$ remains one of the m highest-priority pending jobs during $[t_{i-1}, t_{m-1})$, as no jobs are released during $[t_{i-1}, t_{m-1})$.

During the time interval $[t_{i-1}, t_{m-1})$, the $i^{th}, (i+1)^{st}, \ldots, (m-1)^{st}$ satisfied requests are satisfied and complete. Thus, $t_{m-1} - t_{i-1} \geq (m-1-i+1)L = (m-i)L$. Note that $t_m - t_{i-1}$ can be greater than $(m-i)L$ if a locking protocol inserts delay between two consecutive satisfied requests. Since $J_{hp(i)}$ is pending at time t_{m-1} and it is among the top-m jobs by priority during $[t_{i-1}, t_{m-1})$, it is continuously pi-blocked during $[t_{i-1}, t_{m-1})$. Thus, $J_{hp(i)}$ incurs pi-blocking for at least $(m-i)L$ time units by time t_{m-1}. ◀

Continuing the proof of the lemma, we now show that, for any $i < m$, $B(m-i, t_{m-1}) \geq i$. Consider the set of jobs $\{J_{hp(1)}, J_{hp(2)}, \ldots, J_{hp(i)}\}$. By Claim 8, each job in $\{J_{hp(1)}, J_{hp(2)}, \ldots, J_{hp(i)}\}$ incurs at least $(m-i)L$ time units of pi-blocking by time t_{m-1}. Thus, $B(m-i, t_{m-1}) \geq i$ holds for each $i < m$. Moreover, $B(m, t_m) \geq 0$ holds trivially. Thus, $I(0, t_m) = \wedge_{u=m,v=0}^{u=1,v=m-1} B(u, t_{m-1}) \geq v$ holds. ◀

In the following three lemmas, we show that each job that is pending but unscheduled during time intervals $[t_{km-2}, t_{km-1})$ (for any $2 \leq k \leq m$) incurs pi-blocking during this time interval. This allows the system to steadily reach a time instant t when $I(m-1, t)$ holds. In Fig. 2, the time intervals $[t_{km-2}, t_{km-1})$ refer to time intervals $[6, 7), [10, 11)$, and $[14, 15)$. Note that, during each of these intervals, exactly $m = 4$ jobs are pending.

▶ **Lemma 9.** *For any integer $2 \leq k \leq m$, there are m pending jobs during time interval $[t_{km-2}, t_{km-1})$.*

Proof. By Rule R1, $2m-2$ jobs are released at time 0. By Rule R2, for each time instant t_{im-1} with $1 \leq i \leq m-1$, m jobs are released. By Rules R1 and R2, at or before time t_{km-2}, jobs are released only at time instants $0, t_{m-1}, t_{2m-1}, \ldots, t_{(k-1)m-1}$. Thus, the number of jobs that are released at or before time t_{km-2} is $2m-2+(k-1)m = (k+1)m-2$. By the definition of time instant t_{km-2}, the first $km-2$ satisfied requests are complete and these request-issuing jobs finish execution by time t_{km-2}. Thus, the number of pending jobs at time t_{km-2} is $(k+1)m-2-km+2 = m$. Since no job is released during $[t_{km-2}, t_{km-1})$, the lemma follows. ◀

▶ **Lemma 10.** *For any integer $2 \leq k \leq m-1$, there are $m-1$ pending jobs of $G^1 \cup G^2 \cup \cdots \cup G^k$ at time t_{km-1}.*

Proof. By Rules R1 and R2, only jobs in $G^1 \cup G^2 \cup \cdots \cup G^k$ are released at or before time t_{km-2}. By Lemma 9, there are m pending jobs during time interval $[t_{km-2}, t_{km-1})$. By the definition of time instants t_{km-2} and t_{km-1}, a request completes and the request-issuing job finishes execution at time t_{km-1}. Thus, there are $m-1$ pending jobs of $G^1 \cup G^2 \cup \cdots \cup G^k$ at time t_{km-1}. ◀

▶ **Lemma 11.** *For any integer $2 \leq k \leq m$, all but one job that is pending at time t_{km-2} incurs pi-blocking throughout the time interval $[t_{km-2}, t_{km-1})$.*

Proof. By Rule R2, no jobs in G^i with $i > k$ are released at or before time t_{km-2}. By Lemma 9, there are m pending jobs during the time interval $[t_{km-2}, t_{km-1})$. Among these m jobs, the job whose request is complete at time t_{km-1} must be scheduled during $[t_{km-2}, t_{km-1})$ Thus, each of the remaining $m-1$ pending jobs suffers pi-blocking throughout $[t_{km-2}, t_{km-1})$. ◀

Using Lemma 11, we can show the following lemma. Informally, since $m-1$ jobs incur pi-blocking during time interval $[t_{km-2}, t_{km-1})$, the number of pending jobs at time t_{km-1} that have already incurred L units of pi-blocking is at least $m-1$.

▶ **Lemma 12.** *For any integer $2 \leq k \leq m$, $B(1, t_{km-1}) \geq m-1$.*

Proof. By Lemma 9, there are m pending jobs during $[t_{km-2}, t_{km-1})$. By Lemma 11, $m-1$ jobs among these m pending jobs incur pi-blocking throughout $[t_{km-2}, t_{km-1})$. By the definition of time instants t_i, $t_{km-1} - t_{km-2} \geq L$. Thus, there are at least $m-1$ pending jobs at time t_{km-1} that have incurred pi-blocking for the duration of at least L time units, and the lemma follows. ◀

We now show that there exist time instants t such that $I(k,t)$ holds for each $k \leq m-1$. In Fig. 2, these time instants are times $3, 7, 11$, and 15 when $I(0,3), I(1,7), I(2,11)$, and $I(3,15)$ hold, respectively. Specifically, we show that if $I(k-2, t_{(k-1)m-1})$ holds, then $I(k-1, t_{km-1})$ will also hold.

▶ **Lemma 13.** *For any integer $1 \leq k \leq m$, $I(k-1, t_{km-1})$ holds.*

Proof. We use Fig. 3 to illustrate the proof. By Lemma 7, $I(0, t_{m-1})$ holds. We thus prove the lemma for $k \geq 2$. Assume for a contradiction that $p \geq 2$ is the smallest integer for which $I(p-1, t_{pm-1})$ does not hold. Thus, by the definition of p, we have

$$I(p-2, t_{(p-1)m-1}) \wedge \neg I(p-1, t_{pm-1}). \tag{3}$$

Since $\neg I(p-1, t_{pm-1})$ holds, by (2), we have

$$\bigvee_{u=m, v=p-1}^{u=1, v=m+p-2} B(u, t_{pm-1}) < \min\{v, m-1\}. \tag{4}$$

Note that the index u decreases from m to 1 in (4). Assume that $m-q$ (with $0 \leq q \leq m-1$) is the largest index u in (4) such that $B(u, t_{pm-1}) < \min\{v, m-1\}$ holds. In (4), the index v equals $p-1$ when u equals m, and v increases as u decreases. Thus, when $u = m-q$, we have $v = p-1+q$. Therefore,

$$B(m-q, t_{pm-1}) < \min\{p-1+q, m-1\}. \tag{5}$$

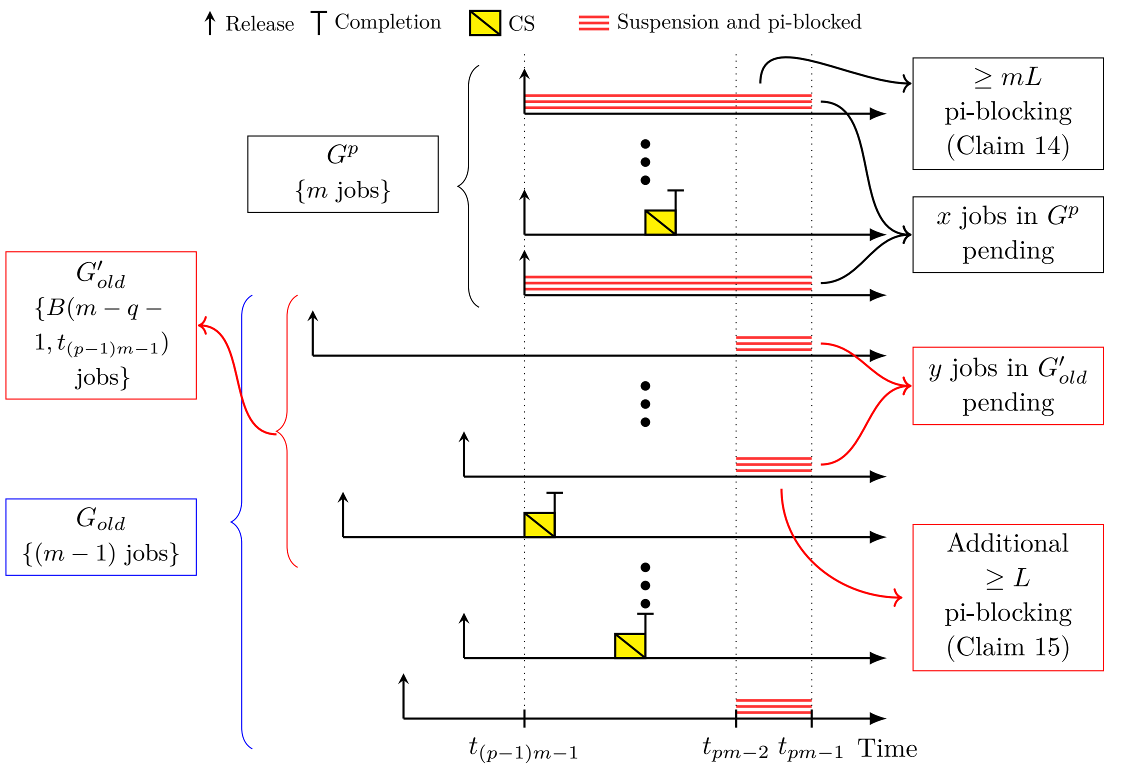
Release
Completion
CS
Suspension and pi-blocked
$\geq mL$ pi-blocking (Claim 14)
x jobs in G^p pending
G^p $\{m$ jobs$\}$
G'_{old} $\{B(m-q-1, t_{(p-1)m-1})$ jobs$\}$
y jobs in G'_{old} pending
G_{old} $\{(m-1)$ jobs$\}$
Additional $\geq L$ pi-blocking (Claim 15)
$t_{(p-1)m-1}$
t_{pm-2}
t_{pm-1}
Time

Figure 3 Illustration of the proof of Lemma 13.

To reach a contradiction, we will show that (5) cannot hold by considering the requests that are satisfied during $[t_{(p-1)m-1}, t_{pm-1})$. By Rule R2, m jobs of G^p are released at time $t_{(p-1)m-1}$ (see Fig. 3). By Lemma 9, there are m pending jobs of $G^1 \cup G^2 \cup \cdots \cup G^{p-1}$ during $[t_{(p-1)m-2}, t_{(p-1)m-1})$. By the definition of time instant $t_{(p-1)m-1}$, one of these m pending jobs during $[t_{(p-1)m-2}, t_{(p-1)m-1})$ completes at time $t_{(p-1)m-1}$. Thus, at time $t_{(p-1)m-1}$, there are $m-1$ pending jobs of $G^1 \cup G^2 \cup \cdots \cup G^{p-1}$. Let G_{old} be the set of these $m-1$ jobs of $G^1 \cup G^2 \cup \cdots \cup G^{p-1}$ that are pending at time $t_{(p-1)m-1}$ (see Fig. 3). We now consider two cases depending on the value of $m-q$.

Case 1. $m-q=1$. Replacing $q=m-1$ in (5), we have $B(1, t_{pm-1}) < \min\{p-1+m-1, m-1\}$. Since $p \geq 2$, we have $p-1+m-1 \geq 2-1+m-1 = m$. Thus, $B(1, t_{pm-1}) < \min\{m, m-1\} = m-1$, contradicting Lemma 12.

Case 2. $m-q>1$. Thus, $m-q-1 \geq 1$ holds. By Def. 6, there are $B(m-q-1, t_{(p-1)m-1})$ pending jobs at time $t_{(p-1)m-1}$ that have incurred pi-blocking for the duration of $(m-q-1)L \geq L$ time units. Thus, each of these $B(m-q-1, t_{(p-1)m-1})$ jobs are from G_{old}, as they must be released before time $t_{(p-1)m-1}$. Let $G'_{old} \subseteq G_{old}$ be these $B(m-q-1, t_{(p-1)m-1})$ jobs (see Fig. 3).

We now lower bound the number of jobs in $G^p \cup G'_{old}$. Since G^p and G'_{old} are disjoint, we have $|G^p| + |G'_{old}| = m + B(m-q-1, t_{(p-1)m-1})$. Thus, to lower bound $|G^p| + |G'_{old}|$, we derive a lower bound on $B(m-q-1, t_{(p-1)m-1})$ using $I(p-2, t_{(p-1)m-1})$ (which holds by (3)). Since $m-q-1 \geq 1$ and $I(p-2, t_{(p-1)m-1})$ holds, by (2), we have $B(m-q-1, t_{(p-1)m-1}) \geq \min\{p-2+q+1, m-1\}$. Using this lower bound on $B(m-q-1, t_{(p-1)m-1})$, we get

$$|G^p| + |G'_{old}| = m + B(m-q-1, t_{(p-1)m-1}) \geq m + \min\{p+q-1, m-1\}. \tag{6}$$

By Lemma 10, there are $m-1$ pending jobs of $G^1 \cup G^2 \cup \cdots \cup G^p$ at time t_{pm-1}. Thus, $m-1$ jobs in $G_{old} \cup G^p$ are pending at time t_{pm-1}. Assume that, at time t_{pm-1}, among these $m-1$ pending jobs of $G_{old} \cup G^p$, x jobs are from G^p, y jobs are from G'_{old}, and the remaining $m-1-x-y$ jobs are from $G_{old} \setminus G'_{old}$ (see Fig. 3). Thus, $|G^p| - x$ jobs from G^p, and $|G'_{old}| - y$ jobs from G'_{old} complete execution by time t_{pm-1}. Since (by Rule R2) a total of m jobs complete execution during $[t_{(p-1)m-1}, t_{pm-1})$, we have

$$|G^p| - x + |G'_{old}| - y \leq m,$$

which implies

$$\begin{aligned} x + y &\geq |G^p| + |G'_{old}| - m \\ &\geq \{\text{By (6)}\} \\ &\quad m + \min\{p+q-1, m-1\} - m \\ &= \min\{p+q-1, m-1\}. \end{aligned} \tag{7}$$

We now show that these $x+y$ pending jobs of $G^p \cup G'_{old}$ incur at least $(m-q)L$ time units of pi-blocking by time t_{pm-1}. By Def. 6 and (7), this implies that $B(m-q, t_{pm-1}) \geq x + y \geq \min\{p+q-1, m-1\}$, contradicting (5). The following two claims help to establish this.

▷ Claim 14. Each of the x jobs of G^p that that are pending at time t_{pm-1} incurs pi-blocking for at least mL time units by time t_{pm-1}.

Proof. Let J be one of the x jobs of G^p that are pending at time t_{pm-1}. Since $J \in G^p$ and $G_{old} \subseteq (G^1 \cup G^2 \cup \cdots \cup G^{(p-1)})$, by Rule P1, J has higher priority than each job in G_{old}. Since only jobs in $G^p \cup G_{old}$ are pending during $[t_{(p-1)m-1}, t_{pm-1})$ and $|G^p| = m$ (by Rule R2), J is among the top-m pending jobs by priority throughout $[t_{(p-1)m-1}, t_{pm-1})$. Thus, J incurs pi-blocking throughout $[t_{(p-1)m-1}, t_{pm-1})$. During $[t_{(p-1)m-1}, t_{pm-1})$, m requests complete execution. Thus, J incurs pi-blocking for at least mL time units. ◀

▷ Claim 15. Each of the y jobs of G'_{old} that are pending at time t_{pm-1} incurs pi-blocking for at least L time units during time interval $[t_{(p-1)m-1}, t_{pm-1})$.

Proof. By Lemma 11, each of the y jobs of G'_{old} that are pending at time t_{pm-1} incurs pi-blocking throughout the time interval $[t_{pm-2}, t_{pm-1})$. By the definition of t_i, $t_{pm-1} - t_{pm-2} \geq L$. Thus, the claim holds. ◀

By the definition of G'_{old}, each of the y pending jobs of G'_{old} has incurred at least $(m-q-1)L$ time units of pi-blocking by time $t_{(p-1)m-1}$. By Claim 15, each such job incurs pi-blocking for at least L time units during $[t_{(p-1)m-1}, t_{pm-1})$. Thus, these y jobs incur pi-blocking for at least $(m-q)L$ time units by time t_{pm-1}. By Claim 14, each of the x jobs of G^p that are pending at time t_{pm-1} incurs at least $mL \geq (m-q)L$ time units of pi-blocking by time t_{pm-1}. Thus, at time t_{pm-1}, there are $x+y$ pending jobs that have incurred pi-blocking for at least $(m-q)L$ time units. Therefore, by (7), $B(m-q, t_{pm-1}) \geq x+y \geq \min\{p+q-1, m-1\}$, contradicting (5).

Thus, in both cases, the lemma holds. ◀

We now prove Theorem 5.

Proof of Theorem 5. By Lemma 13, $I(m-1, t_{m^2-1})$ holds. Thus, by (2), at time t_{m^2-1}, there are $m-1$ pending jobs that have incurred pi-blocking for at least mL time units. By Lemma 10, there are $m-1$ pending jobs of $G^1 \cup G^2 \cup \cdots \cup G^m$ at time t_{m^2-1}. By Rule R3, no new job is released before these $m-1$ jobs are complete. Thus, each of these pending jobs incurs pi-blocking until its request is satisfied. Since any locking protocol must satisfy these $m-1$ pending jobs' requests in some order, the job whose request is satisfied last incurs at least an additional $(m-2)L$ time units of pi-blocking. Therefore, there exists a job that incurs pi-blocking for at least $mL + (m-2)L = (2m-2)L$ time units. ◀

3.3 Job Priority Assignment

In this section, we show how the lower-bound proof in Sec. 3.2 applies under different schedulers. We do so by showing how jobs can be assigned priorities under these schedulers so that Rules P1 and P2 hold.

G-FP schedulers. The following theorem shows that the lower-bound proof in Sec. 3.2 applies to any G-FP scheduler.

▶ **Theorem 16.** *A job in Γ_{seq} incurs pi-blocking for at least $(2m-2)$ request lengths under any G-FP scheduler.*

Proof. Consider a G-FP scheduler $\mathcal{F}$. We re-index the tasks in Γ based on the task priority assignment under $\mathcal{F}$. For each $u > v$, τ_i^u has higher priority than τ_j^v. Also, for each $i > j$, τ_i^u has higher priority than τ_j^u. Thus, job priorities under $\mathcal{F}$ satisfy Rules P1 and P2. The theorem follows from Theorem 5. ◀

GEL schedulers. We now show that the lower-bound proof in Sec. 3.2 also applies under a class of GEL schedulers. The GEL schedulers in this class assign RPPs to tasks in Γ as follows. Task τ_i^u is assigned an RPP Y_i^u that satisfies the following constraints.[4]

$$\forall \tau_m^u : 2 \le u \le m-1 :: Y_m^u > Y_1^{u+1} + m(2m-2)L \tag{8}$$

$$\forall \tau_i^u : 2 \le u \le m-1 \land 1 \le i \le m-1 :: Y_i^u > Y_{i+1}^u \tag{9}$$

$$Y_{2m-2}^1 > Y_1^2 + (m-1)(2m-2)L \tag{10}$$

$$\forall \tau_i^1 : 1 \le i \le 2m-3 :: Y_i^1 > Y_{i+1}^1 \tag{11}$$

▶ **Theorem 17.** *A job in Γ_{seq} incurs pi-blocking for at least $(2m-2)$ request lengths under any GEL scheduler that satisfies (8)–(11).*

Proof. Assume that each job in Γ_{seq} incurs pi-blocking for less than $(2m-2)L$ time units under a GEL scheduler satisfying (8)–(11). We first prove the following claim.

[4] We chose constraints (8) and (10) for conciseness. Less restrictive constraints are also applicable by replacing $m(2m-2)$ and $(m-1)(2m-2)$ with $(4m-4)$.

▷ Claim 18. For $i \geq 0$, $t_{i+1} - t_i < (2m-2)L$.

Proof. By the definitions of t_{i+1} and t_i, a request is satisfied and complete during time interval $[t_i, t_{i+1})$. If $t_{i+1} - t_i \geq (2m-2)L$ holds, then a pending job during time interval $[t_i, t_{i+1})$ incurs pi-blocking for at least $(2m-2)L$ time units, a contradiction. ◀

Using Claim 18, we show that the jobs in Γ_{seq} satisfy Rules P1 and P2. We first show that P2 holds. Let y_i^u denote the PP of job J_i^i. Consider two jobs J_i^u and J_j^u with $i > j$. By (1) and Rules R1 and R2, we have $y_i^u = r_i^u + Y_i^u = r_j^u + Y_i^u$. Since $i > j$ holds, applying (9) and (11), we have $y_i^u < r_j^u + Y_j^u = y_j^u$. Thus, J_i^u has an earlier PP (hence, higher priority) than J_j^u, satisfying P2. We now consider PPs of jobs J_i^u and J_j^{u+1} with $2 \leq u \leq m-1$. By (1), we have

$$\begin{aligned}
y_j^{u+1} &= r_j^{u+1} + Y_j^{u+1} \\
&= \{\text{By Rule R2}\} \\
&\quad t_{um-1} + Y_j^{u+1} \\
&= t_{um-1} - t_{(u-1)m-1} + t_{(u-1)m-1} + Y_j^{u+1} \\
&< \{\text{By Claim 18}\} \\
&\quad m(2m-2)L + t_{(u-1)m-1} + Y_j^{u+1} \\
&\leq \{\text{By Rule R2 and (9)}\} \\
&\quad m(2m-2)L + r_i^u + Y_1^{u+1} \\
&< \{\text{By (8)}\} \\
&\quad m(2m-2)L + r_i^u + Y_m^u - m(2m-2)L \\
&\leq \{\text{By (9)}\} \\
&\quad r_i^u + Y_i^u \\
&= \{\text{By (1)}\} \\
&\quad y_i^u.
\end{aligned}$$

Thus, job J_j^{u+1} has an earlier PP (hence, higher priority) than job J_i^u where $2 \leq u \leq m-1$. Similarly, using (10) and (11) in the above calculation, we can show that J_j^2 has higher priority than J_i^1. Thus, both Rules P1 and P2 are satisfied. By Theorem 5, there is a job that is pi-blocked for at least $(2m-2)$ request lengths, a contradiction. ◀

▶ **Corollary 19.** *A job in Γ_{seq} incurs pi-blocking for at least $(2m-2)$ request lengths under G-EDF scheduling.*

Proof. We let $T_m^m \geq (m^2+m-2)L = nL$. We assign periods to each task so that the constraints (8)–(11) are satisfied. Note that $T_m^m = T_{min}$ holds. Thus, by Lemma 3, Γ is feasible. Since $Y_i^u = T_i^u$ holds under G-EDF, by Theorem 17, the corollary holds. ◀

The case of FIFO scheduling. Our lower-bound proof heavily relies on being able to release m jobs that have higher-priority than any earlier-released jobs (Rule R2 and P1). The proof does not apply for any scheduler that prevents higher-priority job releases. FIFO, which is a GEL scheduler, is one such example as it prioritizes jobs by their release times (a future job cannot have an earlier release time). This property enabled the design of the OLP-F (see Tbl. 2) that achieves $(m-1)L$ pi-blocking bound under FIFO scheduling.

Request vs. release blocking. Recall from Sec. 2 that a job may experience both release blocking and request blocking under a locking protocol. Also, recall from Tbl. 2 that the C-OMLP achieves a per-job pi-blocking bound of $(2m-1)L$ time units through a combination of a request-blocking bound of $(m-1)L$ time units and a release-blocking bound of mL time units. By the construction of Γ and Γ_{seq}, it may appear that the lower bound of $(2m-2)L$ time units of pi-blocking applies only for request blocking (as each job issues a request as soon as it is released), contradicting the C-OMLP's request-blocking bound. However, our bound actually applies for the total per-job pi-blocking (the sum of request and release blocking). A locking protocol like the C-OMLP may choose to decompose the worst-case per-job pi-blocking of $(2m-2)$ request lengths into seperate release blocking and request blocking terms.

4 Improved Lower Bound Under An Additional Assumption

In this section, we improve the lower bound established in Sec. 3 for a class of locking protocols that satisfy a certain property. We begin by introducing some terms that we use to define this class of locking protocols.

▶ **Definition 20.** *Consider a job J that issues a request $\mathcal{R}$ at time $t_a(\mathcal{R})$ that is satisfied at time $t_s(\mathcal{R})$. Define $t_h(\mathcal{R})$ as follows: if J ever becomes one of the m highest-priority pending jobs in $[t_a(\mathcal{R}), t_s(\mathcal{R}))$, then let $t_h(\mathcal{R})$ denote the first such time; otherwise, let $t_h(\mathcal{R}) = \infty$.*

Using the above definition, we define *reorder-bounded* locking protocols.

▶ **Definition 21.** *Let J_i and J_j be two jobs that issue request $\mathcal{R}_i$ and $\mathcal{R}_j$, respectively. A* reorder-bounded *locking protocol decides the order in which $\mathcal{R}_i$ and $\mathcal{R}_j$ are satisfied (relative to each other) no later than time* $\max\{t_h(\mathcal{R}_i), t_h(\mathcal{R}_j)\}$.

To the best of our knowledge, most (if not all) existing locking protocols are reorder-bounded. Such protocols utilize a deterministic queue structure (which can be a hybrid of different queues). Such a queue structure maintains an order of requests, which typically depends on the time instant when each request enters into this structure and job-specific attributes that are fixed. Once a request enters into the queue structure, the order in which it will be satisfied relative to existing requests in the queue structure becomes fixed. In the rest of this section, we limit attention to locking protocols that are reorder-bounded.

We now show that there exists a task system and a corresponding release sequence such that the pi-blocking incurred by a job can be arbitrarily close to $(2m-1)$ request lengths. We organize the proof in a similar structure as in Sec. 3.

4.1 Task System

Let $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of n tasks that are scheduled on m processors. Each job of each task issues a request of length L for a resource ℓ as soon as it is released. Each job completes as soon as its request for resource ℓ completes. Thus, $C_i = L$. Let $\epsilon > 0$ be a constant (which can be arbitrarily close to 0) such that $\epsilon < L$ holds. We assume that $n = (2m-1) + (2m-1)L/\epsilon$. For ease of notation, we assume that $(2m-1)L/\epsilon$ is an integer.[5] Below, we show that Γ is schedulable under any JLFP scheduler when the minimum period T_{min} is large enough.

[5] This assumption can be removed by taking $n = (2m-1) + \lceil (2m-1)L/\epsilon \rceil$.

▶ **Lemma 22.** *If $T_{min} \geq nL$, then there exists a suspension-based locking protocol under which Γ is schedulable under any JLFP scheduler.*

Proof. We show that Γ is schedulable under any JLFP scheduler when lock requests are satisfied in FIFO order. Assume that Γ is not schedulable in a JLFP schedule $\mathcal{S}$ where lock requests are satisfied in FIFO order. Let J be the job with the earliest release time, call it t_r, that misses its deadline in $\mathcal{S}$. By the definition of J, no job with a deadline at or before time t_r misses its deadline. Since each task has an implicit deadline, there is at most one pending job per task at time t_r. Thus, there are at most n pending jobs (including J) at time t_r. Since requests are satisfied in FIFO order, J's request is complete by the time these n requests are complete. Since JLFP scheduling is work-conserving, these n requests complete by time $t_r + nL$. Since each job finishes execution when its request completes, J completes execution by time $t_r + nL \leq t_r + T_{min}$. Thus, J does not miss its deadline, a contradiction. ◀

Release sequence for Γ. We now give a release sequence Γ_{seq} for Γ. We only give the release time for one job J_i of each task τ_i. We denote job J_i's request by $\mathcal{R}_i$. These jobs are released according to the following rules. (An example release sequence is given in Fig. 4, which we cover in detail below.)

S1 Jobs $\{J_1, J_2, \cdots, J_m\}$ are released at time 0. Without loss of generality, assume that, $\mathcal{R}_1$ is the last satisfied request among $\{\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_m\}$.

S2 Let t_k be the time instant when the k^{th}-satisfied request is satisfied. Job J_{m+k} is released at time $t_k + \epsilon$ if *one* of the following conditions is met.

S2.1 $k = 1$ holds.

S2.2 $1 < k \leq n - m$ holds and $\mathcal{R}_{m+k-1}$ is ordered (hence, satisfied) before $\mathcal{R}_1$.

S3 Let $h = \min\{n - m - k, m - 1\}$. Jobs $\{J_{m+k+1}, J_{m+k+2}, \cdots, J_{m+k+h}\}$ are released at time $t_k + \epsilon$ if $\mathcal{R}_{m+k}$ is ordered (hence, satisfied) after $\mathcal{R}_1$.

S4 If some jobs are released by Rule S3, then no task releases a new job until all jobs released by Rule S3 complete execution.

Note that Rules S2 and S3 require that, when a job J_i with $i > m$ is released, the order in which the prior job J_{i-1}'s request will be satisfied with respect to $\mathcal{R}_1$ is known. In Lemma 26, we will show that this ordering has already been finalized when J_{i-1} is released under any reorder-bounded locking protocol. Finally, when a job J_{m+k} is released whose request is ordered after $\mathcal{R}_1$, h new jobs are instantaneously released at the same time by Rule S3. For simplicity, we assume such instantaneous releases are possible. This assumption can be removed by delaying the release of the h new jobs by another ϵ time units.

Job priorities. We assume job priorities satisfy the following rule. In Sec. 4.3, we will describe how such priorities can be assigned under different schedulers.

Q For any $i > j$, job J_i has higher priority than job J_j.

▶ **Example 23.** Fig. 4 depicts a release sequence according to Rules S1–S4 for $m = 4$, $L = 3$, and $\epsilon = 1$. By Rule S1, jobs J_1–J_4 are released at time 0. At time 0, J_2's request is satisfied. Thus, $t_1 = 0$ holds. At time $t_1 + \epsilon = 1$, by Rule S2, job $J_{m+1} = J_5$ is released. $t_2 = 3$ holds because the second request to be satisfied (J_4's request) is satisfied at time 3. Assuming J_5's request is ordered before $\mathcal{R}_1$, by Rule S2, J_6 is released at time $t_2 + \epsilon = 4$. At time 7, J_7 is released. Job J_7's request is ordered after $\mathcal{R}_1$. Thus, by Rule S3, J_8–J_{10} are released at time 7.

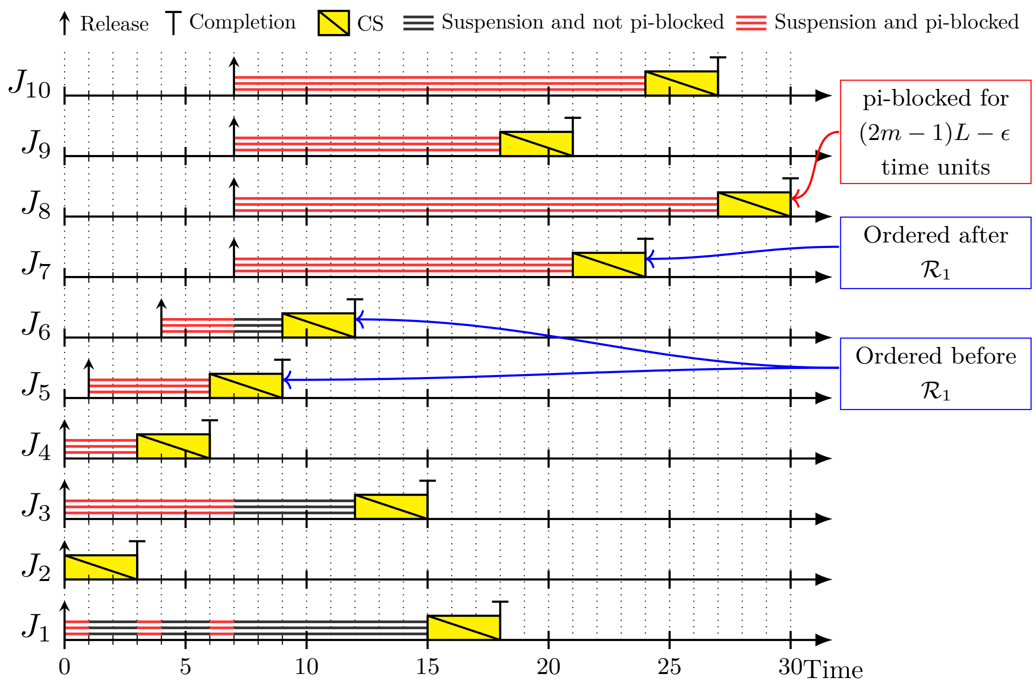

Figure 4 Release sequence by Rules S1–S4 for $m = 4$, $L = 3$, and $\epsilon = 1$.

In Fig. 4, the time intervals when a job experience s-oblivious pi-blocking are marked red. During time interval $[0, 1)$, J_1 is one of the top $m = 4$ jobs by priority. Thus, it experience pi-blocking during this interval. However, due to release of J_5 and Rule Q, J_1 is not one of the top $m = 4$ jobs by priority during time interval $[2, 3)$. Thus, it does not experience pi-blocking during this interval. ◀

4.2 Lower-Bound Proof

In this section, we prove the following theorem.

▶ **Theorem 24.** *Under a reorder-bounded locking protocol, there exists a job in Γ_{seq} that incurs pi-blocking for at least $(2m-1)L-\epsilon$ time units when job priorities satisfy Rule Q.*

We prove Theorem 24 using the following two lemmas.

▶ **Lemma 25.** *If no job is released by Rule S3 at or before time t_i where $1 \le i \le n-m+1$, then there are m pending jobs at time t_i.*

Proof. We first determine the number of jobs released at or before time t_i. By Rule S1, m jobs are released at time 0. By Rule S2, only job J_{m+k-1} is released during $[t_{k-1}, t_k)$ for any $2 \le k \le i$. Note that J_{m+k-1} is valid because $m+k-1 \le m+i-1 \le m+n-m+1-1 = n$ holds. Thus, the number of released jobs by time t_i is $m+i-1$. By the definition of t_i (Rule S2), the i^{th}-satisfied request is satisfied but not complete at time t_i. Thus, exactly $i-1$ jobs complete execution by time t_i. Therefore, the number of pending jobs at time t_i is $m+i-1-(i-1) = m$. ◀

▶ **Lemma 26.** *The relative order in which each request $\mathcal{R}_i$ (with $i > 1$) is satisfied with respect to $\mathcal{R}_1$ is determined when $\mathcal{R}_i$ is issued.*

Proof. By Rule S1, each request $\mathcal{R}_i$ with $1 \leq i \leq m$ is issued at time 0 when J_i is one of the top-m jobs by priority. Thus, $t_h(\mathcal{R}_i) = t_a(\mathcal{R}_i) = 0 \geq t_a(\mathcal{R}_1)$ for each $1 \leq i \leq m$. By Rule Q, job J_i has higher priority than any job J_j with $j < i$. By Rules S2 and S3, at most m jobs are released at any time. Thus, for any job J_i with $i > m$, $t_h(\mathcal{R}_i) = t_a(\mathcal{R}_i) > 0 = t_a(\mathcal{R}_1) = t_h(\mathcal{R}_1)$ holds. Therefore, for any i, $t_a(\mathcal{R}_i) = \max\{t_h(\mathcal{R}_i), t_h(\mathcal{R}_1)\}$. By Def. 21, the relative order in which $\mathcal{R}_i$ is satisfied with respect to $\mathcal{R}_1$ is determined at time $t_a(\mathcal{R}_i)$. ◀

We now prove Theorem 24.

Proof of Theorem 24. For a contradiction, we assume the following.

(A1) Each job in Γ_{seq} incurs pi-blocking for less than $(2m-1)L - \epsilon$ time units.

In the following claim, we show that a job must exist whose request is satisfied later than $\mathcal{R}_1$ to satisfy (A1). Consequently, when such a job is released, h new jobs are also released by Rule S3.

▷ Claim 27. There exists a request $\mathcal{R}_{m+q}$ with $1 \leq q \leq n - 2m + 1$ that is satisfied after $\mathcal{R}_1$.

Proof. Assume that each request $\mathcal{R}_{m+q}$ with $1 \leq q \leq n - 2m + 1$ is satisfied before $\mathcal{R}_1$. By Rule S1, $\mathcal{R}_1$ is the last satisfied request among $\{\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_m\}$. Thus, $\mathcal{R}_1$ is the last satisfied request among all requests in $\{\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_{m+n-2m+1}\}$. Hence, by the definition of time instant t_i (Rule S2), $\mathcal{R}_1$ is not satisfied before time $t_{m+n-2m+1} = t_{n-m+1}$.

Since each request $\mathcal{R}_{m+q}$ with $1 \leq q \leq n - 2m + 1$ is satisfied before $\mathcal{R}_1$, no job J_{m+q} with $1 \leq q \leq n - 2m + 1$ is released by Rule S3. Consider any time instant t_q with $1 \leq q \leq n - 2m + 1$. By Lemma 25, there are m pending jobs at time t_q. By Rule S2, no job is released during $(t_q, t_q + \epsilon)$. Thus, the number of pending jobs throughout $[t_q, t_q + \epsilon)$ is m. Since $\mathcal{R}_1$ is not satisfied before time t_{n-m+1}, J_1 is pending and pi-blocked during all intervals $[t_q, t_q + \epsilon)$ with $1 \leq q \leq n - 2m + 1$. Thus, J_1 incurs pi-blocking for at least $(n - 2m + 1)\epsilon = (2m - 1 + (2m-1)L/\epsilon - 2m + 1)\epsilon = (2m-1)L$ time units, contradicting (A1). ◀

Let $\mathcal{R}_{m+q}$ be the request with smallest $(m+q)$ value that is ordered after $\mathcal{R}_1$ (J_7's request in Fig. 4). Claim 27 guarantees the existence of such request. Therefore, the following holds.

(A2) Each request $\mathcal{R}_i$ with $i < m + q$ and $i \neq 1$ is satisfied before $\mathcal{R}_1$. (Requests of jobs J_2–J_6 in Fig. 4.)

By Rule S3, in addition to J_{m+q}, $\min\{n-m-q, m-1\}$ new jobs are released at time $t_q + \epsilon$. By Claim 27, $q \leq n - 2m + 1$ holds. Thus, $n - m - q \geq n - m - (n - 2m + 1) = m - 1$ and $\min\{n - m - q, m - 1\} = m - 1$. Therefore, including J_{m+q}, a total of $m - 1 + 1 = m$ new jobs are released at time $t_{q+\epsilon}$. Thus, the following holds.

(A3) There are m active requests $\mathcal{R}_i$ at time $t_q + \epsilon$ with $i \geq m + q$. (Requests of jobs J_7–J_{10} in Fig. 4 at time 7.)

Since $t_q + \epsilon$ is the first time instant when jobs are released by Rule S3, by Lemma 25, there are m pending jobs J_i with $i < m + q$ at time t_q. By the definition of t_q, a request from one of these m jobs is satisfied at time t_q. Since $\epsilon < L$, the satisfied request is not complete by time $t_q + \epsilon$. Thus, these m pending jobs with $i < m + q$ are also pending at time $t_q + \epsilon$. Therefore, we have the following.

(A4) There are m active requests $\mathcal{R}_i$ at time $t_q + \epsilon$ with $i < m + q$. (Requests of jobs J_1, J_3, J_5, and J_6 in Fig. 4 at time 7.)

By (A3) and (A4), there are total $2m$ pending requests at time $t_q + \epsilon$. Let $\mathcal{R}_i$ be the last satisfied request among these $2m$ requests. By the definition of $\mathcal{R}_{m+q}$, $i \neq 1$ holds, as $\mathcal{R}_{m+q}$ is ordered after $\mathcal{R}_1$. Therefore, by (A2), $i \geq m + q$ holds. By Rule Q, J_i has higher priority than each of the m jobs J_j with $j < m + q$. By Rule S4, no new jobs will be released until $\mathcal{R}_i$ is complete. Thus, J_i experience pi-blocking until it is satisfied, which occurs after the completion of the other $2m - 1$ requests. Since one of these $2m - 1$ requests is satisfied at time t_q and J_i is released at time $t_q + \epsilon$, J_i incurs pi-blocking for at least $(2m - 1)L - \epsilon$ time units, contradicting (A1). ◀

4.3 Job Priority Assignment

In this section, we show how the lower-bound proof in Sec. 4.2 applies under different schedulers by demonstrating how jobs can be assigned priorities under these schedulers so that Rule Q holds.

G-FP schedulers. The following theorem shows that the lower-bound proof in Sec. 4.2 is valid under any G-FP scheduler.

▶ **Theorem 28.** *Under any reorder-bounded locking protocol, a job in Γ_{seq} incurs pi-blocking for at least $(2m - 1)L - \epsilon$ time units under any G-FP scheduler.*

Proof. Consider a G-FP scheduler $\mathcal{F}$. We re-index the tasks in Γ based on the task priority assignment under $\mathcal{F}$. For each $i > j$, τ_i has higher priority than τ_j. Thus, job priorities under $\mathcal{F}$ satisfies Rule Q. The theorem follows from Theorem 24. ◀

GEL schedulers. We now show that the lower-bound proof in Sec. 4.2 also applies under a class of GEL schedulers. The GEL schedulers in this class assign RPPs to tasks in Γ satisfying the following constraints.

$$\forall\ i : 1 \leq i \leq n - 1 :: Y_i > Y_{i+1} + (2m - 1)L \tag{12}$$

▶ **Theorem 29.** *Under any reorder-bounded locking protocol, a job in Γ_{seq} incurs pi-blocking for at least $(2m - 1)L - \epsilon$ time units under any GEL scheduler that satisfies (12).*

Proof. Assume that each job in Γ_{seq} incurs pi-blocking for less than $(2m - 1)L - \epsilon$ time units under a GEL scheduler that assigns task RPPs according to (12). We first claim that the following hold.

$$\forall\ i \geq 1 : t_{i+1} - t_i < (2m - 1)L \tag{13}$$

$$\forall\ i \geq 1 : t_1 < (2m - 1)L \tag{14}$$

If either (13) or (14) does not hold, then a job must incur at least $(2m - 1)L$ time units of pi-blocking during $[t_i, t_{i+1})$, a contradiction.

We now show that (12) satisfies Rule Q. We show this by considering two consecutive jobs J_i and J_{i+1}. We first show that $y_{i+1} - y_i < 0$ by considering the following three cases.

Case 1. $i = m$. In this case, by Rules S1 and S2, J_i and J_{i+1} are released at times 0 and t_1, respectively. By (1), we have $y_{i+1} - y_i = r_{i+1} + Y_{i+1} - r_i - Y_i = t_1 + Y_{i+1} - Y_i$. Thus, by (12) and (14), we have $y_{i+1} - y_i < (2m - 1)L - (2m - 1)L = 0$.

Case 2. $i < m$ or J_{i+1} is released by Rule S3. In this case, by Rules S1 and S3, both jobs J_i and J_{i+1} are released at the same time, *i.e.*, $r_{i+1} = r_i$ holds. By (1) and (12), we have $y_{i+1} - y_i = r_{i+1} + Y_{i+1} - r_i - Y_i = Y_{i+1} - Y_i < -(2m-1)L < 0$.

Case 3. $i > m$ and J_{i+1} is not released by Rule S3. In this case, both jobs J_i and J_{i+1} are released by Rule S2. By Rule S2, J_{m+k} is released at time $t_k + \epsilon$. Thus, $J_i = J_{m+(i-m)}$ is released at time $t_{i-m} + \epsilon$. Similarly, $J_i = J_{m+(i+1-m)}$ is released at time $t_{i+1-m} + \epsilon$. Thus, $r_{i+1} = t_{i+1-m} + \epsilon$, and $r_i = t_{i-m} + \epsilon$. Hence, by (13), we have $r_{i+1} - r_i = t_{i-m+1} - t_{i-m} < (2m-1)L$. Now, by (1) and (12), we have $y_{i+1} - y_i = r_{i+1} + Y_{i+1} - r_i - Y_i = r_{i+1} - r_i + Y_{i+1} - Y_i < (2m-1)L - (2m-1)L = 0$.

In all three cases, $y_{i+1} < y_i$ holds. Therefore, J_{i+1} has an earlier PP (hence, higher priority) than J_i. Hence, Rule Q is satisfied. Thus, by Theorem 24, there is a job that incurs pi-blocking for at least $(2m-1)L - \epsilon$ time units, a contradiction. ◀

▶ **Corollary 30.** *Under any reorder-bounded locking protocol, a job in Γ_{seq} incurs pi-blocking for at least $(2m-1)L - \epsilon$ time units G-EDF scheduling.*

Proof. Let $T_n \geq (2m-1)L + (2m-1)L^2/\epsilon = nL$, and for each $i < n$, let $T_i = T_{i+1} + 2mL$. Thus, (12) is satisfied, as $Y_i = T_i$ holds under G-EDF. Note that $T_n = T_{min}$ holds. Thus, by Lemma 22, Γ is feasible. By Theorem 29, the corollary holds. ◀

5 Related Work

A wealth of work has been done on suspension-based multiprocessor real-time locking protocols (*e.g.*, [3, 7–11, 13–17, 19, 20]). Interested readers can find an excellent recent survey in [6]. Below, we comment further on a few specific relevant protocols.

Locking protocols such as the FMLP [2] and MPCP-VS [14] were presented and considered under s-oblivious analysis even before the term was formally defined by Brandenburg and Anderson [7]. The OMLP is the first known asymptotically optimal locking protocol under such analysis. Later, variants of the OMLP were introduced that are asymptotically optimal under s-oblivious analysis under clustered (and hence both partitioned and global) scheduling [8]. Later, the OMIP [3] was presented; it is asymptotically optimal under s-oblivious analysis while maintaining an *independence preserving* property (which isolates tasks from unrelated critical sections) under clustered JLFP scheduling. Recently, the OLP-F [1] was proposed, which achieves optimal s-oblivious pi-blocking under FIFO scheduling.

Many locking protocols have been studied under *suspension-aware* (s-aware) analysis where suspension time is treated as time not executing. Many of these protocols (*e.g.*, the MPCP [19], the PPCP [11], the PIP [18], *etc.*) were inspired by classical uniprocessor locking protocols. Under s-aware analysis, an $\Omega(n)$ lower bound on pi-blocking has been established [7]. The FMLP$^+$ [5] is an extension of the FMLP that achieves asymptotically optimal s-aware pi-blocking under clustered JLFP scheduling. Later, linear-programming techniques were shown to improve the s-aware analysis of various protocols, including the PIP, the PPCP, and the FMLP, under global and partitioned fixed-priority scheduling [4, 21].

6 Conclusion

In this paper, we have closed a long-standing open problem concerning pi-blocking optimality. In particular, we have presented lower-bound results that show that the factor of two present in the pi-blocking bounds of most real-time multiprocessor mutex protocols that are asymptotically optimal under s-oblivious analysis is fundamental. In presenting these results, we have assumed global scheduling. As global scheduling is a special case of clustered scheduling, our results are applicable to locking protocols that target clustered scheduling as well. Extensions to our analysis would be required to obtain more refined lower bounds that take cluster sizes into account. Extensions would also be needed to reach conclusions about optimal locking under partitioned scheduling. The literature on optimal multiprocessor synchronization is not limited to mutex sharing. In particular, reader/writer sharing (both read requests and write requests are supported and only the latter require mutual exclusion) and k-exclusion sharing (k simultaneous lock holders are allowed) have been considered. A reader/writer lock can be used to support mutex sharing by considering only writers, and a k-exclusion lock reduces to a mutex lock when $k = 1$. Thus, our results have implications for the design of optimal protocols for reader/writer and k-exclusion sharing as well. However, as before, it might be possible to obtain more refined lower-bound results by taking into account the specific nature of these sharing constraints.

References

1 S. Ahmed and J. Anderson. Optimal multiprocessor locking protocols under FIFO scheduling. In *ECRTS'23*, pages 16:1–16:21, 2023.

2 A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In *RTCSA'07*, pages 47–56, 2007.

3 B. Brandenburg. A fully preemptive multiprocessor semaphore protocol for latency-sensitive real-time applications. In *ECRTS'13*, pages 292–302, 2013.

4 B. Brandenburg. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *RTAS'13*, pages 141–152, 2013.

5 B. Brandenburg. The FMLP+: an asymptotically optimal real-time locking protocol for suspension-aware analysis. In *ECRTS'14*, pages 61–71, 2014.

6 B. Brandenburg. Multiprocessor real-time locking protocols. In *Handbook of Real-Time Computing*, pages 347–446. Springer, 2022.

7 B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. In *RTSS'10*, pages 49–60, 2010.

8 B. Brandenburg and J. Anderson. Real-time resource-sharing under clustered scheduling: Mutex, reader-writer, and k-exclusion locks. In *EMSOFT'11*, pages 69–78, 2011.

9 B. Brandenburg and J. Anderson. The OMLP family of optimal multiprocessor real-time locking protocols. *Des. Autom. Embed.*, 17(2):277–342, 2014.

10 C. Chen, S. Tripathi, and A. Blackmore. A resource synchronization protocol for multiprocessor real-time systems. In *ICPP'94*, pages 159–162, 1994.

11 A. Easwaran and B. Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *RTSS'09*, pages 377–386, 2009.

12 J. Erickson, J. Anderson, and B. Ward. Fair lateness scheduling: reducing maximum lateness in G-EDF-like scheduling. *Real-Time Syst.*, 50(1):5–47, 2014.

13 D. Faggioli, G. Lipari, and T. Cucinotta. Analysis and implementation of the multiprocessor bandwidth inheritance protocol. *Real Time Syst.*, 48(6):789–825, 2012.

14 K. Lakshmanan, D. Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *RTSS'09*, pages 469–478, 2009.

15 F. Nemati, M. Behnam, and T. Nolte. Independently-developed real-time systems on multicores with shared resources. In *ECRTS'11*, pages 251–261, 2011.

16 F. Nemati and T. Nolte. Resource hold times under multiprocessor static-priority global scheduling. In *RTCSA'11*, pages 197–206, 2011.

17 F. Nemati and T. Nolte. Resource sharing among real-time components under multiprocessor clustered scheduling. *Real Time Syst.*, 49(5):580–613, 2013.

18 R. Rajkumar. *Synchronization In Real-Time Systems – A Priority Inheritance Approach.* Kluwer Academic Publishers, 1991.

19 R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *RTSS'88*, pages 259–269, 1988.

20 Z. Tong, S. Ahmed, and J. Anderson. Overrun-resilient multiprocessor real-time locking. In *ECRTS'22*, pages 9:1–9:23, 2022.

21 M. Yang, A. Wieder, and B. Brandenburg. Global real-time semaphore protocols: A survey, unified analysis, and comparison. In *RTSS'15*, pages 1–12, 2015.