Locking in BPF

Kumar Kartikeya Dwivedi

EPFL | Meta

Outline

01 Background

02 Problem Statement

03 Resilient Queued Spin Lock

04 Evaluation

05 Next Steps

bpf_spin_lock

- Helper functions were introduced in 2019.
- Allows updating map values atomically.

```
v = bpf_map_lookup_elem(&map, &key);
if (!v) { return 0; }
bpf_spin_lock(&v->lock);
v->val++;
bpf_spin_unlock(&v->lock);
```

bpf_spin_lock

- Helper functions were introduced in 2019.
- Allows updating map values atomically.
- One lock at a time, to prevent deadlocks.
- No function calls in the critical section.

```
v = bpf_map_lookup_elem(&map, &key);
if (!v) { return 0; }
bpf_spin_lock(&v->lock);
v->val++;
bpf_spin_unlock(&v->lock);
```

Graph Data Structures

- sched_ext led to introduction of linked lists and red-black trees.

```
n = bpf_obj_new(typeof(*n));
if (!n) { return 0; }
n->key = 5;
n->data = 10;

bpf_spin_lock(&lock);
bpf_rbtree_add(&root, &n->node, less);
bpf_spin_unlock(&lock);
```

Graph Data Structures

- sched_ext led to introduction of linked lists and red-black trees.
- Still restricted to one lock at a time.
- Only data structure operations supported inside the critical section.

```
n = bpf_obj_new(typeof(*n));
if (!n) { return 0; }
n->key = 5;
n->data = 10;

bpf_spin_lock(&lock);
bpf_rbtree_add(&root, &n->node, less);
bpf_spin_unlock(&lock);
```

Friction

- Expressing useful algorithms not feasible with “one lock at a time” constraint.

```
v1 = bpf_map_lookup_elem(&map, &key1);
v2 = bpf_map_lookup_elem(&map, &key2);
if (!v1 || !v2) {
  return 0;
}
bpf_spin_lock(&v1->lock);
bpf_spin_lock(&v2->lock); // AA or ABBA
migrate_task(p, v1->rbtree, v2->rbtree);
…
```

Friction

- Expressing useful algorithms not feasible with “one lock at a time” constraint.
- Restricting function calls within critical section is too prohibitive.

```
bpf_spin_lock(lock1);
bpf_map_lookup_elem(&map, &key); // No!
bpf_spin_unlock(lock1);
```

Friction

- Expressing useful algorithms not feasible with “one lock at a time” constraint.
- Restricting function calls within critical section is too prohibitive.

github.com/sched-ext/scx/blob/main/scheds/c/scx_pair.bpf.c

```
/* again, it'd be better to do all these with the lock held, oh well */
vptr = bpf_map_lookup_elem(&cgrp_q_idx_hash, &cgid);
if (!vptr) {
        scx_bpf_error("failed to lookup q_idx for cgroup[%llu]", cgid);
        return -ENOENT;
}
q_idx = *vptr;
```

Bugs

- syzbot regularly finds deadlocks in BPF maps even after implementing per-CPU counter protection.

```
============================================
WARNING: possible recursive locking detected
6.8.0-syzkaller-05242-g32fa4366cc4d #0 Not tainted
--------------------------------------------
syz-executor217/5072 is trying to acquire lock:
ffff88802a0fd9f8 (&trie->lock){....}-{2:2}, at: trie_delete_elem+0x96/0x6a0 kernel/bpf/lpm_trie.c:451

but task is already holding lock:
ffff88802a0fc9f8 (&trie->lock){....}-{2:2}, at: trie_update_elem+0xcb/0xc10 kernel/bpf/lpm_trie.c:324

other info that might help us debug this:
 Possible unsafe locking scenario:

       CPU0
       ----
  lock(&trie->lock);
  lock(&trie->lock);

 *** DEADLOCK ***
```

Bugs

- syzbot regularly finds deadlocks in BPF maps even after implementing per-CPU counter protection.
- Infeasible for verifier to perform full-kernel control flow analysis to prevent them statically.

```
============================================
WARNING: possible recursive locking detected
6.8.0-syzkaller-05242-g32fa4366cc4d #0 Not tainted
--------------------------------------------
syz-executor217/5072 is trying to acquire lock:
ffff88802a0fd9f8 (&trie->lock){....}-{2:2}, at: trie_delete_elem+0x96/0x6a0 kernel/bpf/lpm_trie.c:451

but task is already holding lock:
ffff88802a0fc9f8 (&trie->lock){....}-{2:2}, at: trie_update_elem+0xcb/0xc10 kernel/bpf/lpm_trie.c:324

other info that might help us debug this:
 Possible unsafe locking scenario:

       CPU0
       ----
  lock(&trie->lock);
  lock(&trie->lock);

 *** DEADLOCK ***
```

```
============================================
WARNING: possible recursive locking detected
6.13.0-rc5-syzkaller-00163-gab75170520d4 #0 Not tainted
--------------------------------------------
syz-executor174/5963 is trying to acquire lock:
ffff88802ea401e0 (&qs->lock){....}-{2:2}, at: __queue_map_get+0x2b6/0x360 kernel/bpf/queue_stack_maps.c:105

but task is already holding lock:
ffff8880239cb1e0 (&qs->lock){....}-{2:2}, at: __queue_map_get+0x2b6/0x360 kernel/bpf/queue_stack_maps.c:105

other info that might help us debug this:
 Possible unsafe locking scenario:

       CPU0
       ----
  lock(&qs->lock);
  lock(&qs->lock);

 *** DEADLOCK ***

 May be due to missing lock nesting notation
```

Bugs

- syzbot regularly finds deadlocks in BPF maps even after implementing per-CPU counter protection.
- Infeasible for verifier to perform full-kernel control flow analysis to prevent them statically.
- Endless game of whack-a-mole.

```
============================================
WARNING: possible recursive locking detected
6.8.0-syzkaller-05242-g32fa4366cc4d #0 Not tainted
--------------------------------------------
syz-executor217/5072 is trying to acquire lock:
ffff88802a0fd9f8 (&trie->lock){....}-{2:2}, at: trie_delete_elem+0x96/0x6a0 kernel/bpf/lpm_trie.c:451

but task is already holding lock:
ffff88802a0fc9f8 (&trie->lock){....}-{2:2}, at: trie_update_elem+0xcb/0xc10 kernel/bpf/lpm_trie.c:324

other info that might help us debug this:
 Possible unsafe locking scenario:

       CPU0
       ----
  lock(&trie->lock);
  lock(&trie->lock);

 *** DEADLOCK ***
```

```
============================================
WARNING: possible recursive locking detected
6.13.0-rc5-syzkaller-00163-gab75170520d4 #0 Not tainted
--------------------------------------------
syz-executor174/5963 is trying to acquire lock:
ffff88802ea401e0 (&qs->lock){....}-{2:2}, at: __queue_map_get+0x2b6/0x360 kernel/bpf/queue_stack_maps.c:105

but task is already holding lock:
ffff8880239cb1e0 (&qs->lock){....}-{2:2}, at: __queue_map_get+0x2b6/0x360 kernel/bpf/queue_stack_maps.c:105

other info that might help us debug this:
 Possible unsafe locking scenario:

       CPU0
       ----
  lock(&qs->lock);
  lock(&qs->lock);

 *** DEADLOCK ***

 May be due to missing lock nesting notation
```

```
=====================================================
WARNING: possible circular locking dependency detected
6.13.0-rc1-syzkaller-00025-gfeffde684ac2 #0 Not tainted
-----------------------------------------------------
syz-executor207/6807 is trying to acquire lock:
ffff88802632eca0 (&htab->lockdep_key#434){....}-{2:2}, at: htab_lock_bucket kernel/bpf/hashtab.c
ffff88802632eca0 (&htab->lockdep_key#434){....}-{2:2}, at: htab_lru_map_delete_elem+0x1c8/0x790

but task is already holding lock:
ffff888031440e20 (&htab->lockdep_key#435){....}-{2:2}, at: htab_lock_bucket kernel/bpf/hashtab.c
ffff888031440e20 (&htab->lockdep_key#435){....}-{2:2}, at: htab_lru_map_delete_elem+0x1c8/0x790

which lock already depends on the new lock.


the existing dependency chain (in reverse order) is:
```

02 Problem Statement

Deadlock Safety

Guaranteed forward progress for the kernel.

Intractable problem to use static analysis to detect deadlocks.

Deadlock Safety

Guaranteed forward progress for the kernel.

Intractable problem to use static analysis to detect deadlocks.

Scalability

BPF programs / maps are used in performance sensitive contexts.

Overhead to ensure runtime safety must be minimal.

Deadlock Safety

Guaranteed forward progress for the kernel.

Intractable problem to use static analysis to detect deadlocks.

Scalability

BPF programs / maps are used in performance sensitive contexts.

Overhead to ensure runtime safety must be minimal.

Fault Isolation

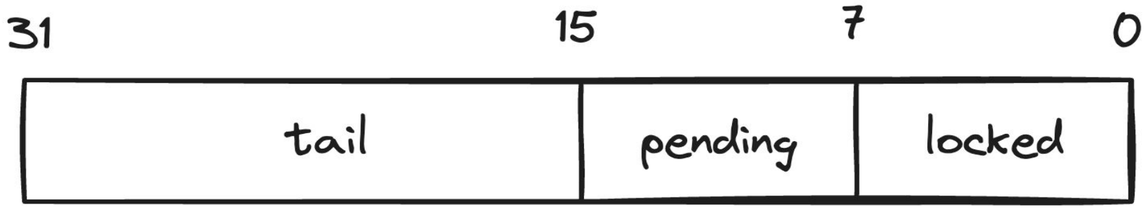
Only offending programs are affected by incorrectness.

Kernel must pinpoint the culprit program and recover itself quickly.

03 Resilient Queued Spin Lock

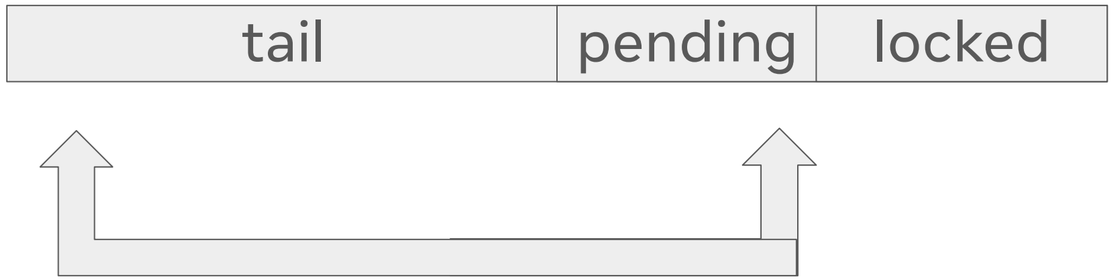
Queued Spin Lock - Primer

- 4-byte lock word.
- A single byte to indicate ownership status (locked).
- A pending ‘queue’ of size 1 for low contention (pending).
- Proper MCS queue of size NR_CPUS for high contention (tail).



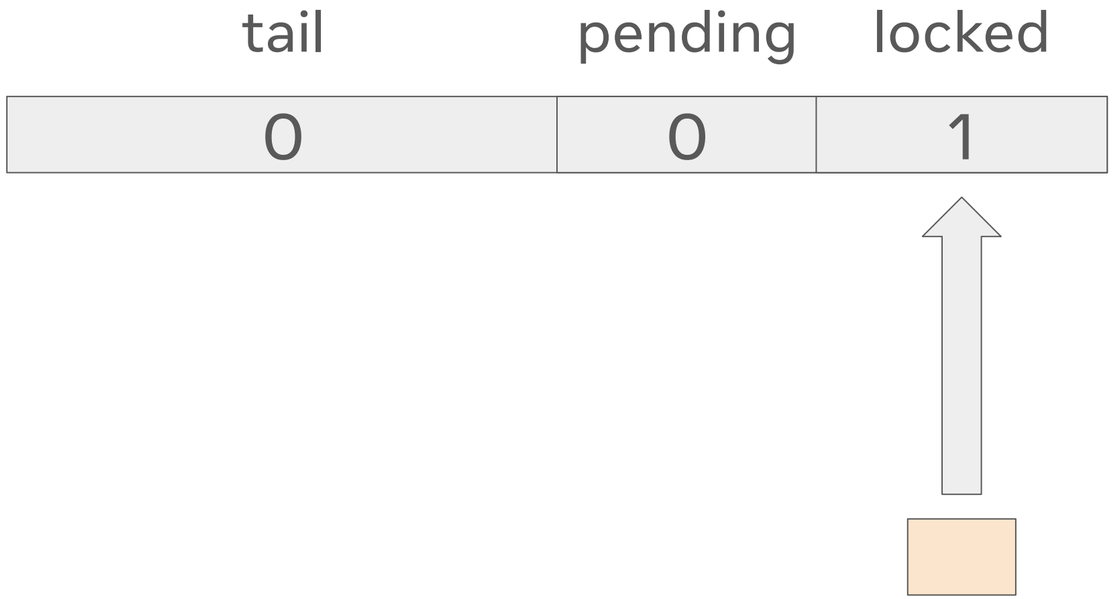
10,000 foot view

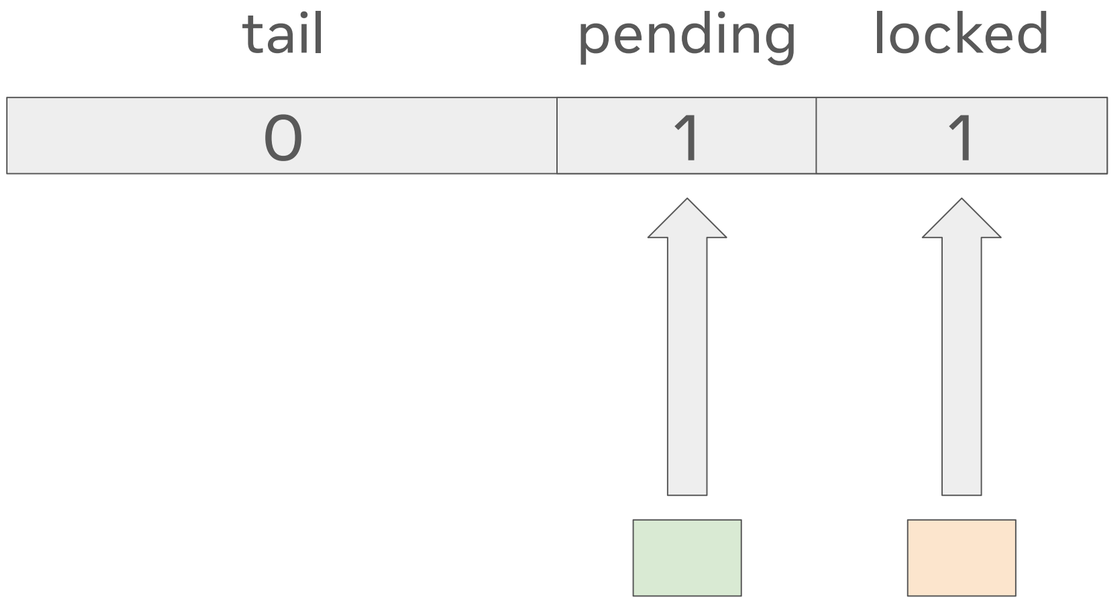
- Each CPU maintains a table of held locks.
- Both ‘pending’ waiter and ‘MCS queue’ head will check for deadlocks.
- If deadlock is detected (AA or ABBA), return error.
- If deadlock is not detected but enough time has passed, return error.

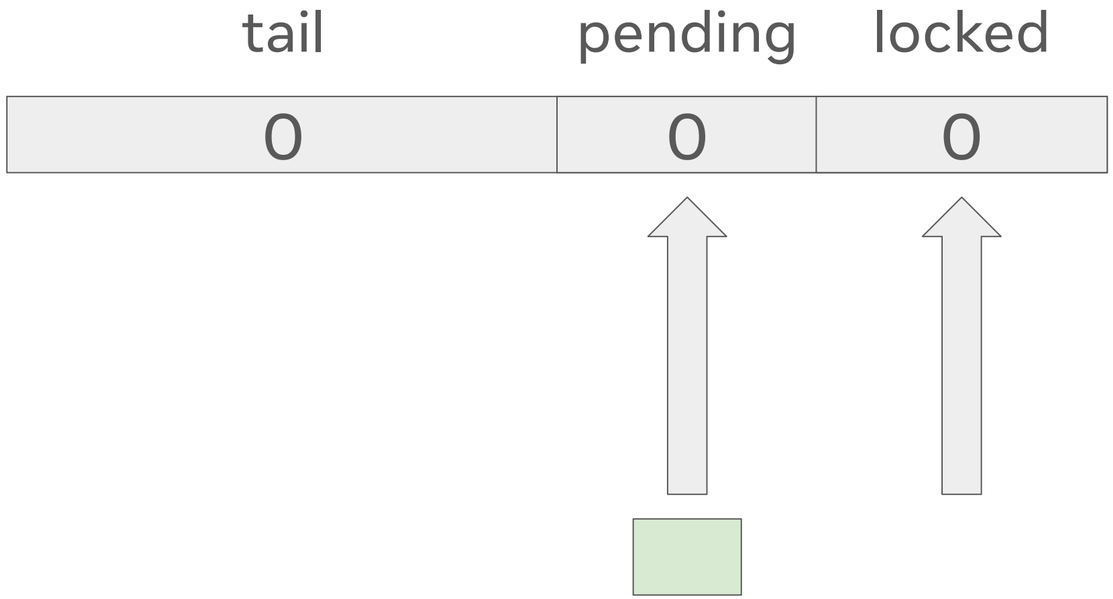


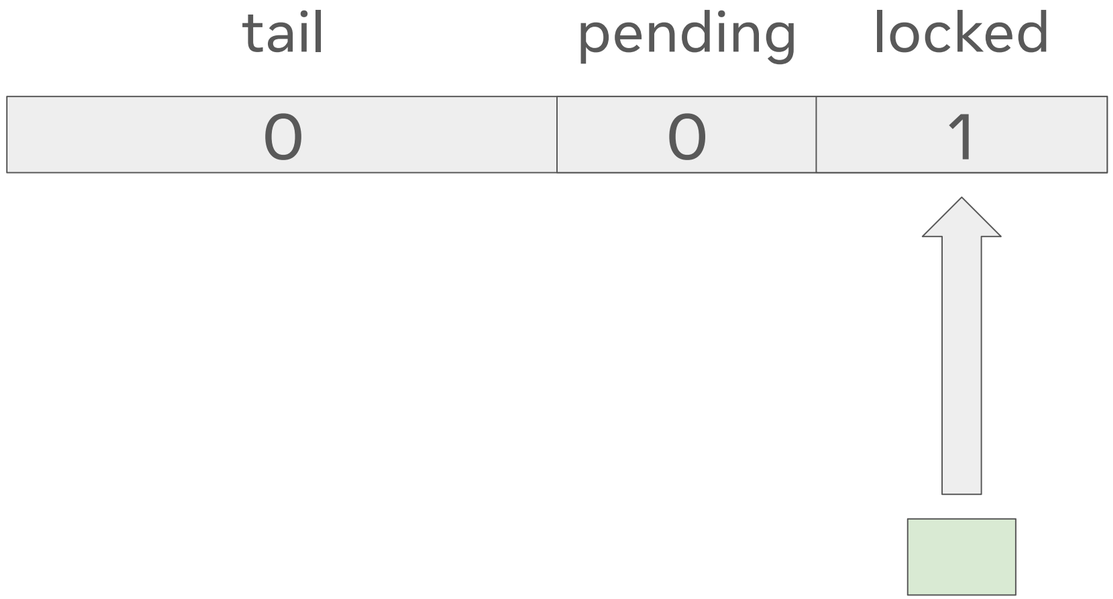
Contention

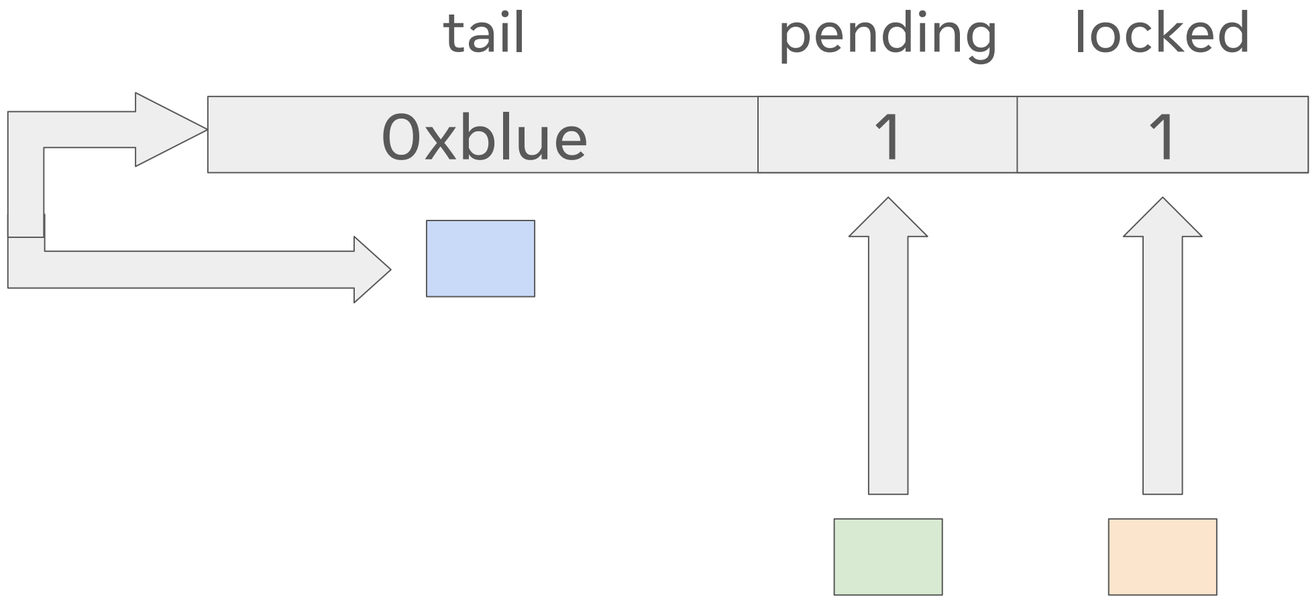

tail
pending
locked
0
0
0

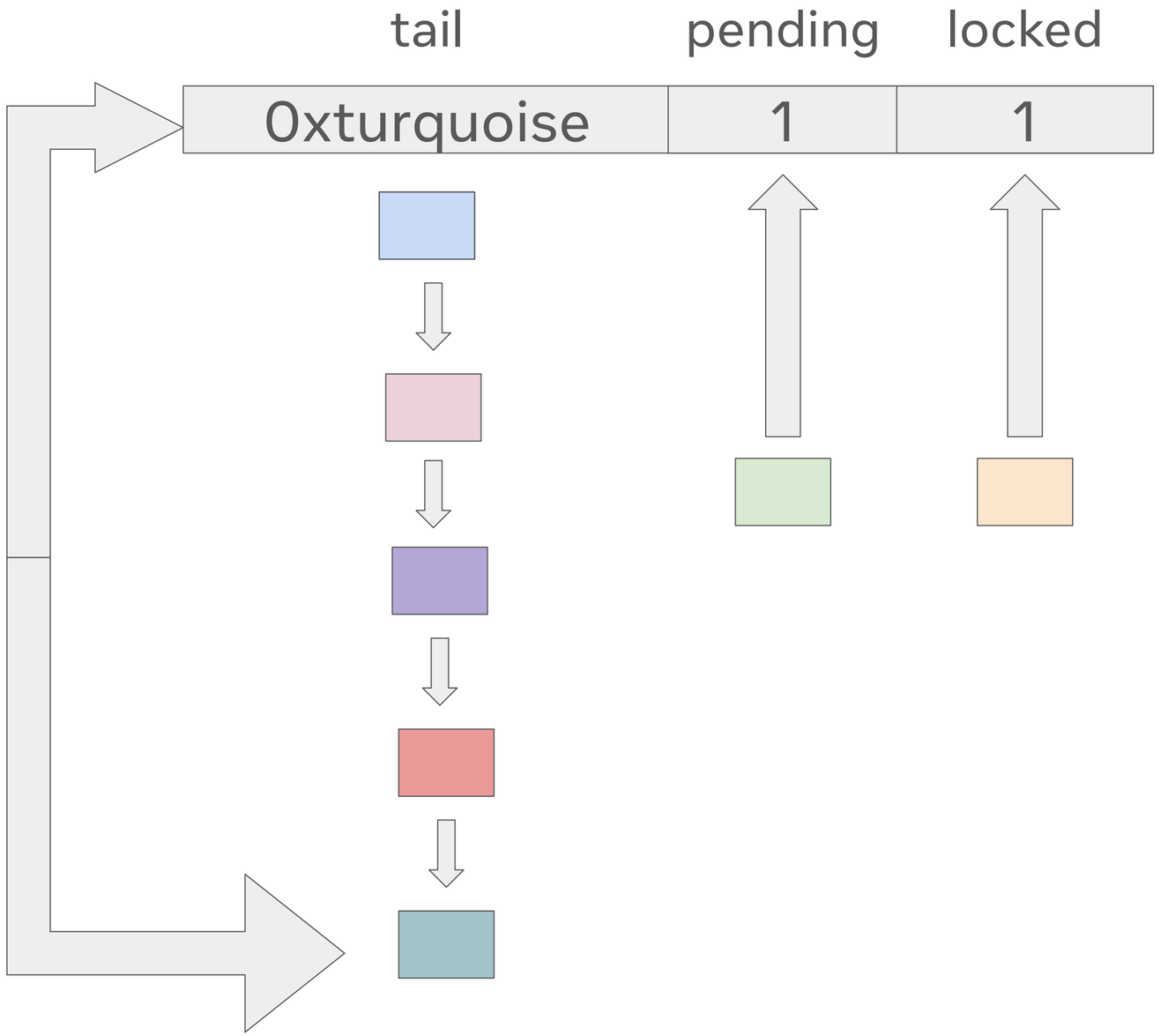

tail
pending
locked
0
0
1

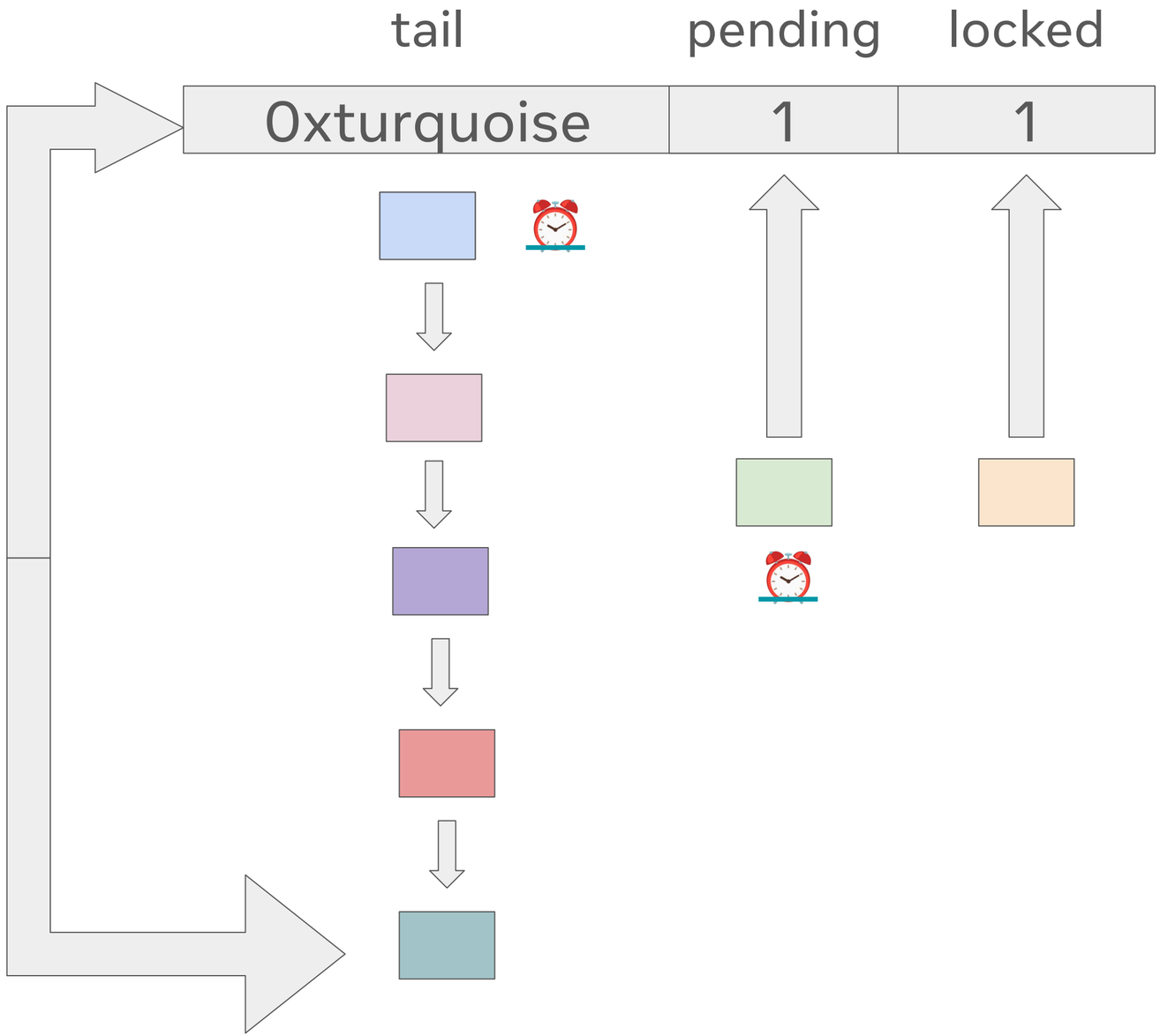

tail
pending
locked
0
1
1

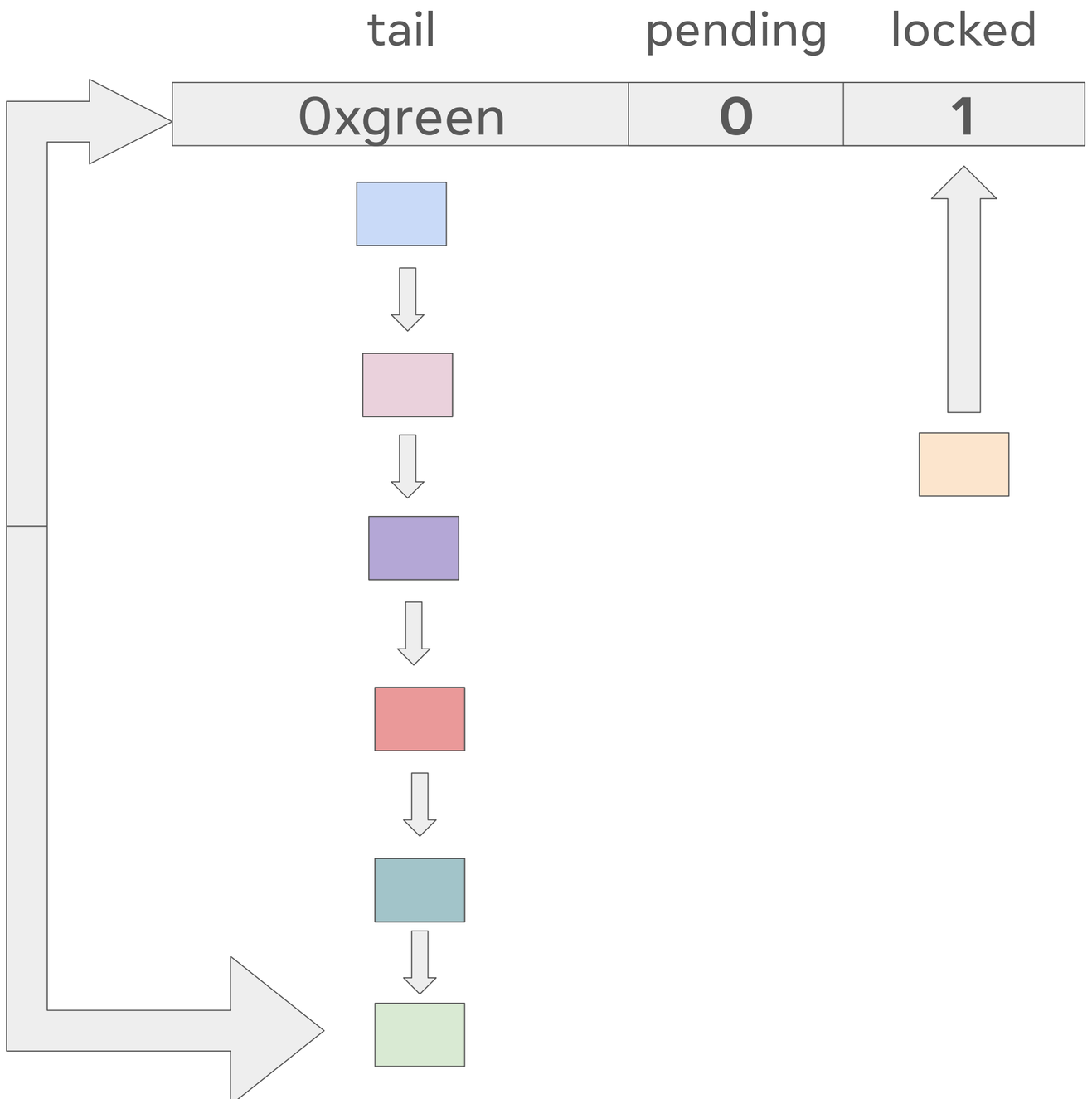

tail
pending
locked
0
0
0


tail
pending
locked
0
0
1


tail
pending
locked
0xblue
1
1


tail
pending
locked
0xturquoise
1
1


tail
pending
locked
0xturquoise
1
1

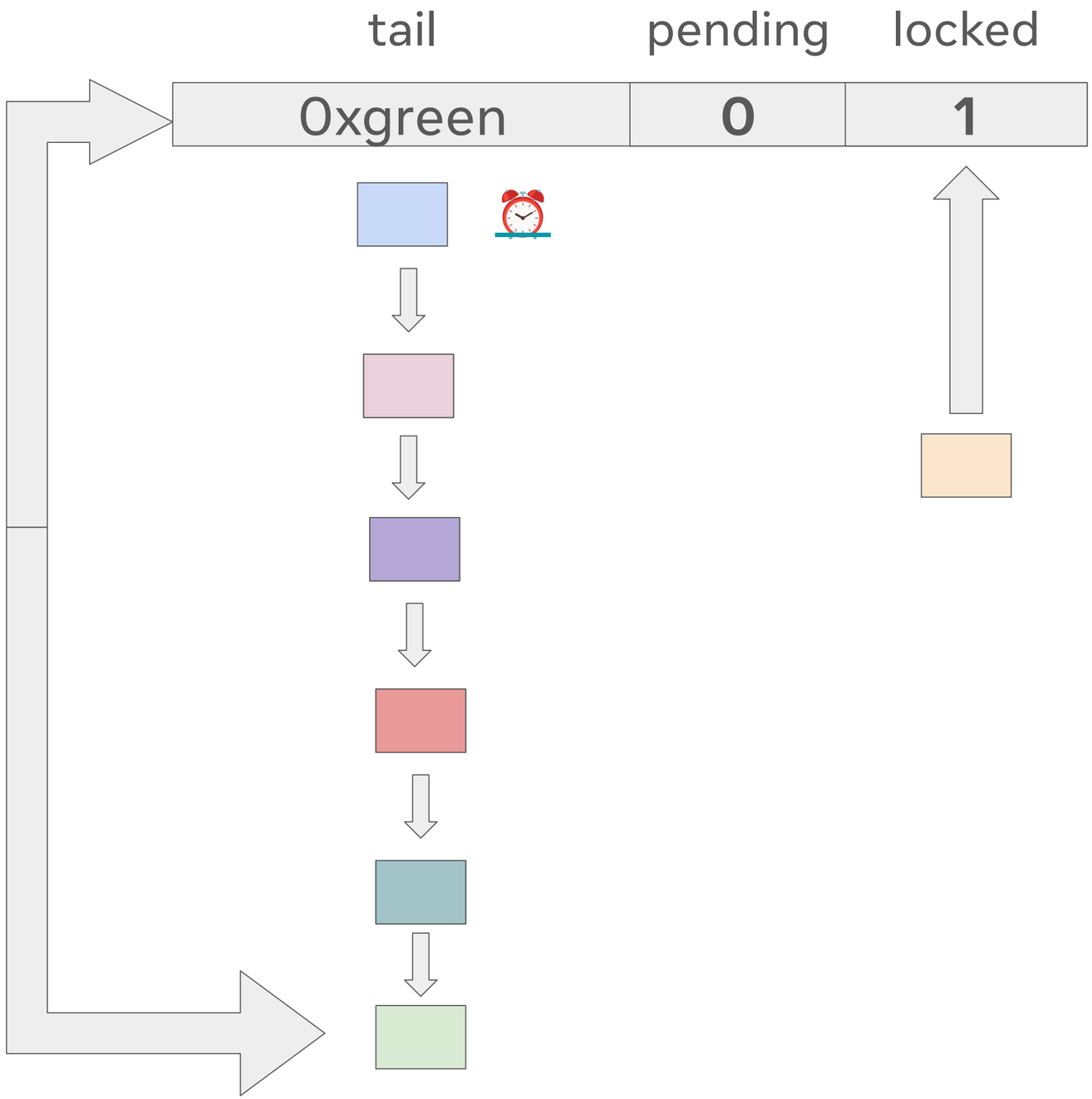

tail
pending
locked
0xgreen
0
1

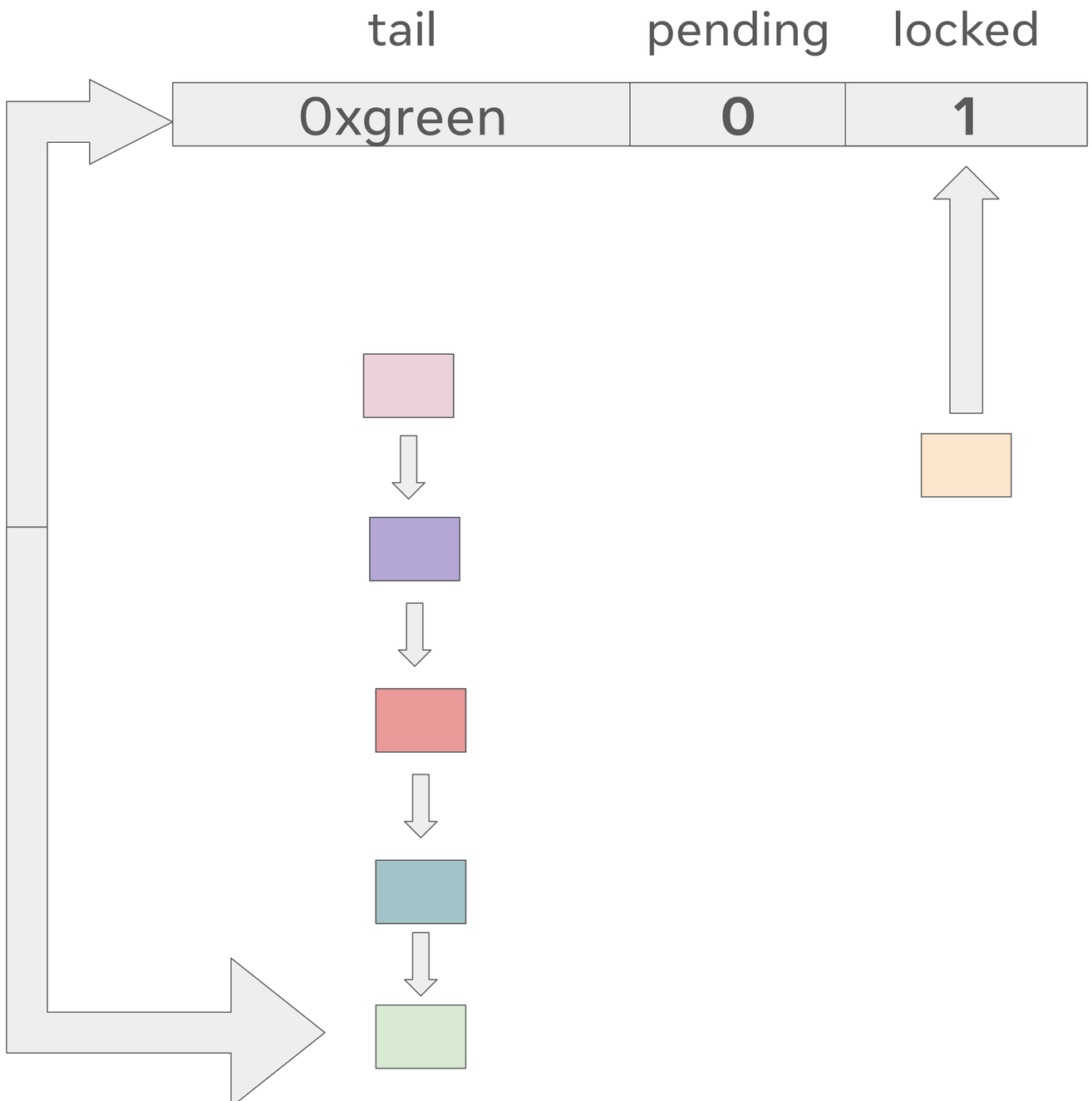
Recovery of queue

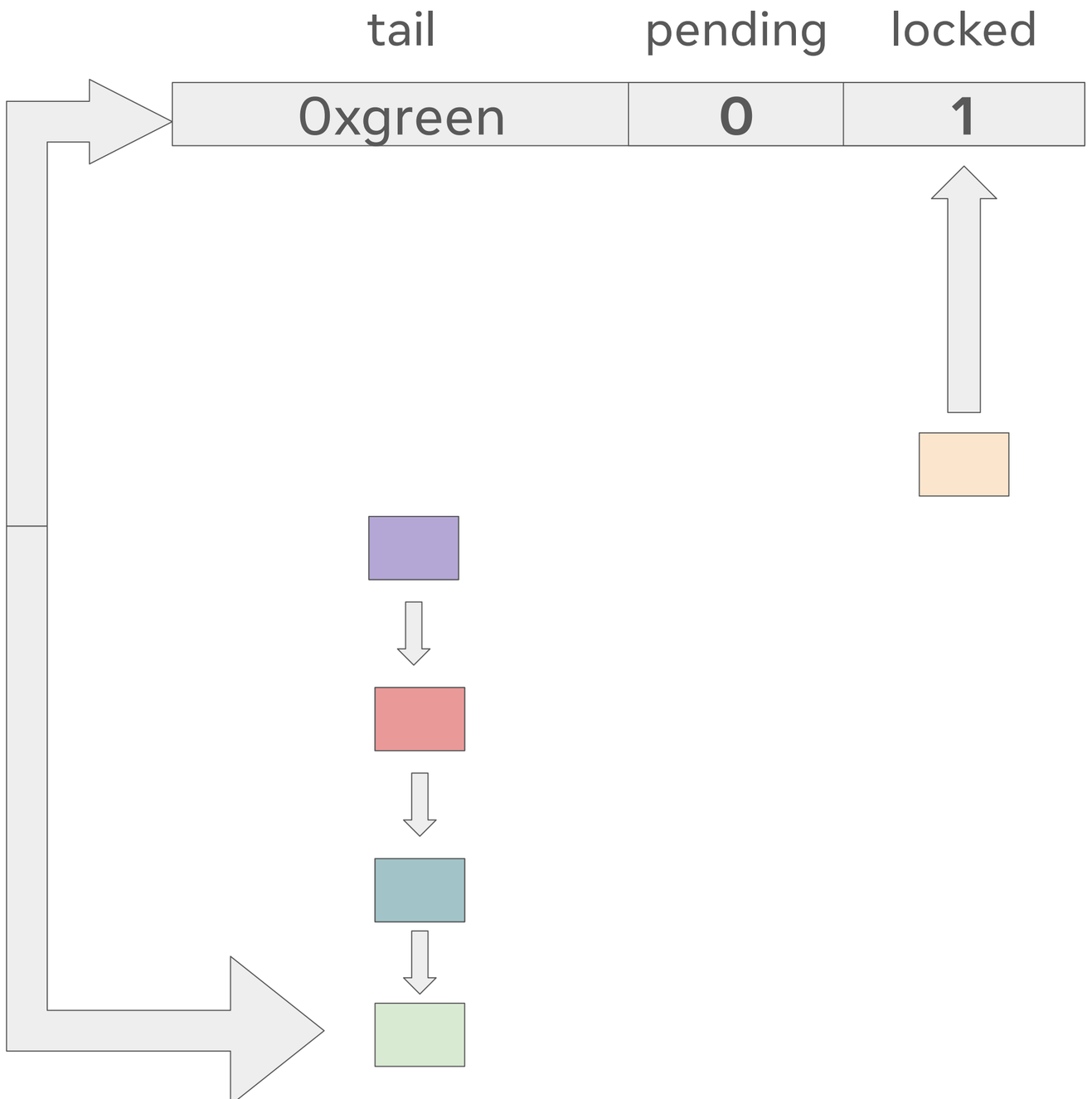
The head of the wait queue runs deadlock and timeout checks

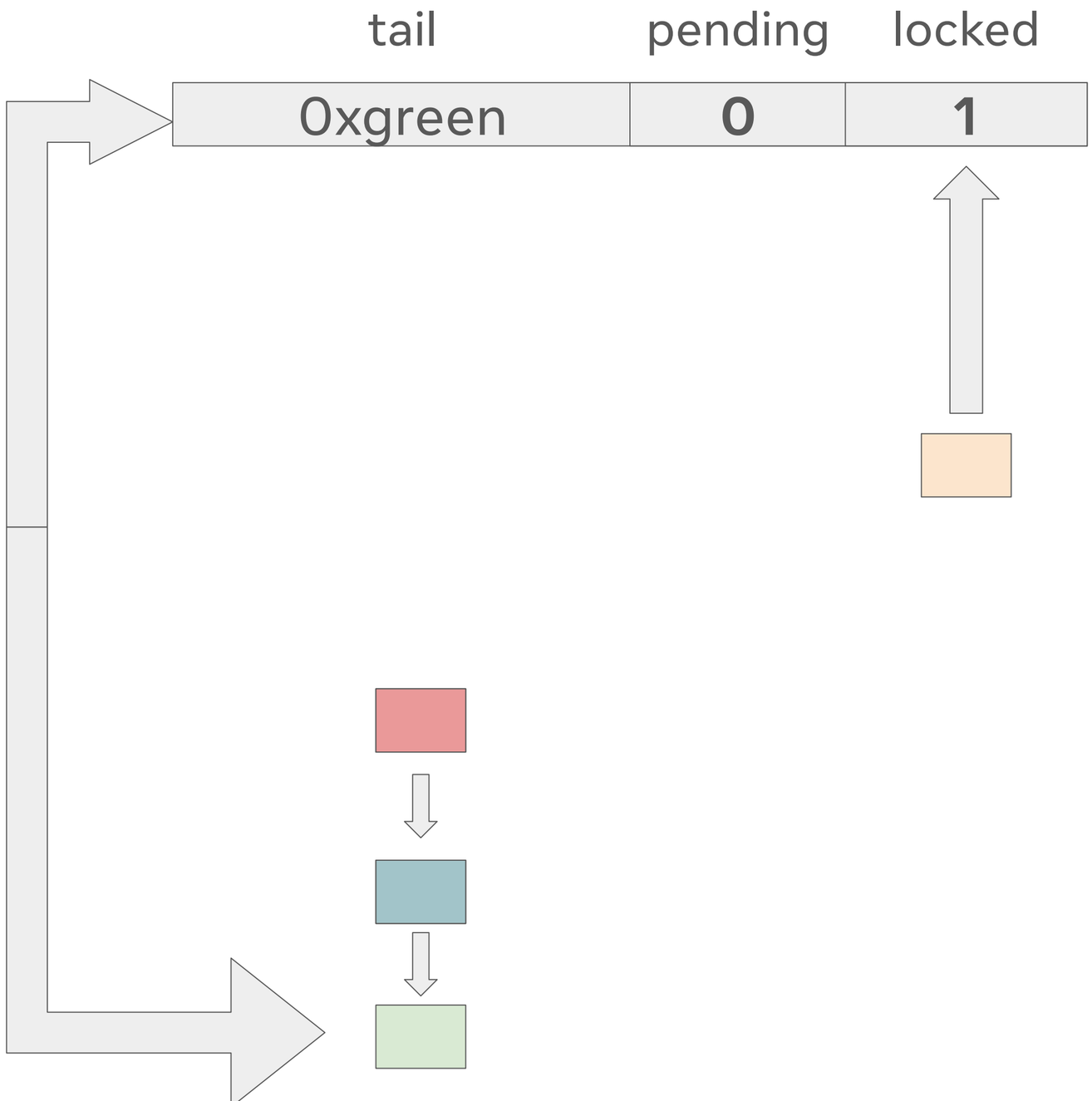
In case of timeout at head of queue:

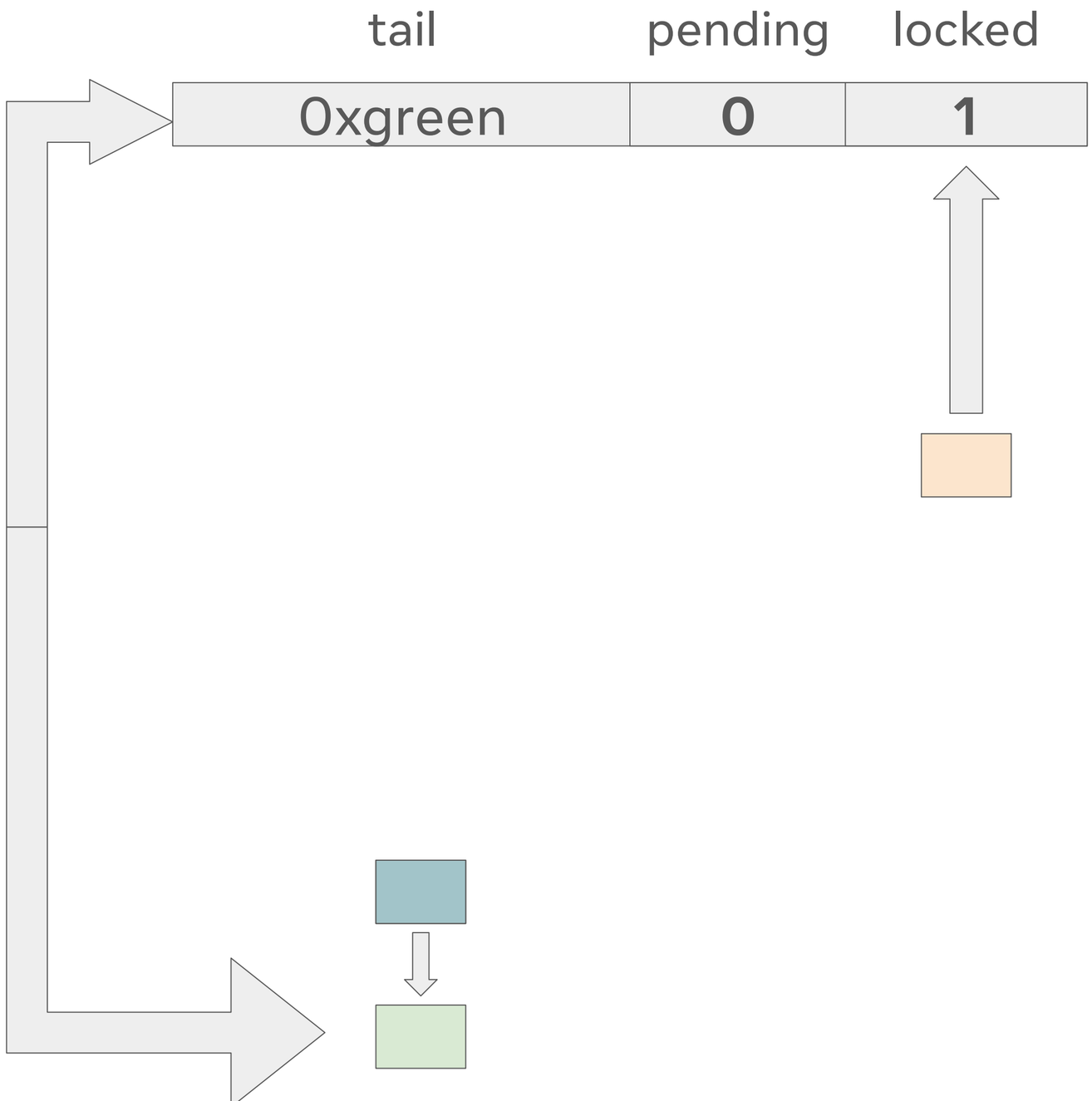
- Exit from wait queue is in FIFO order.
- No need to handle races of waiters in the middle of the queue randomly leaving.

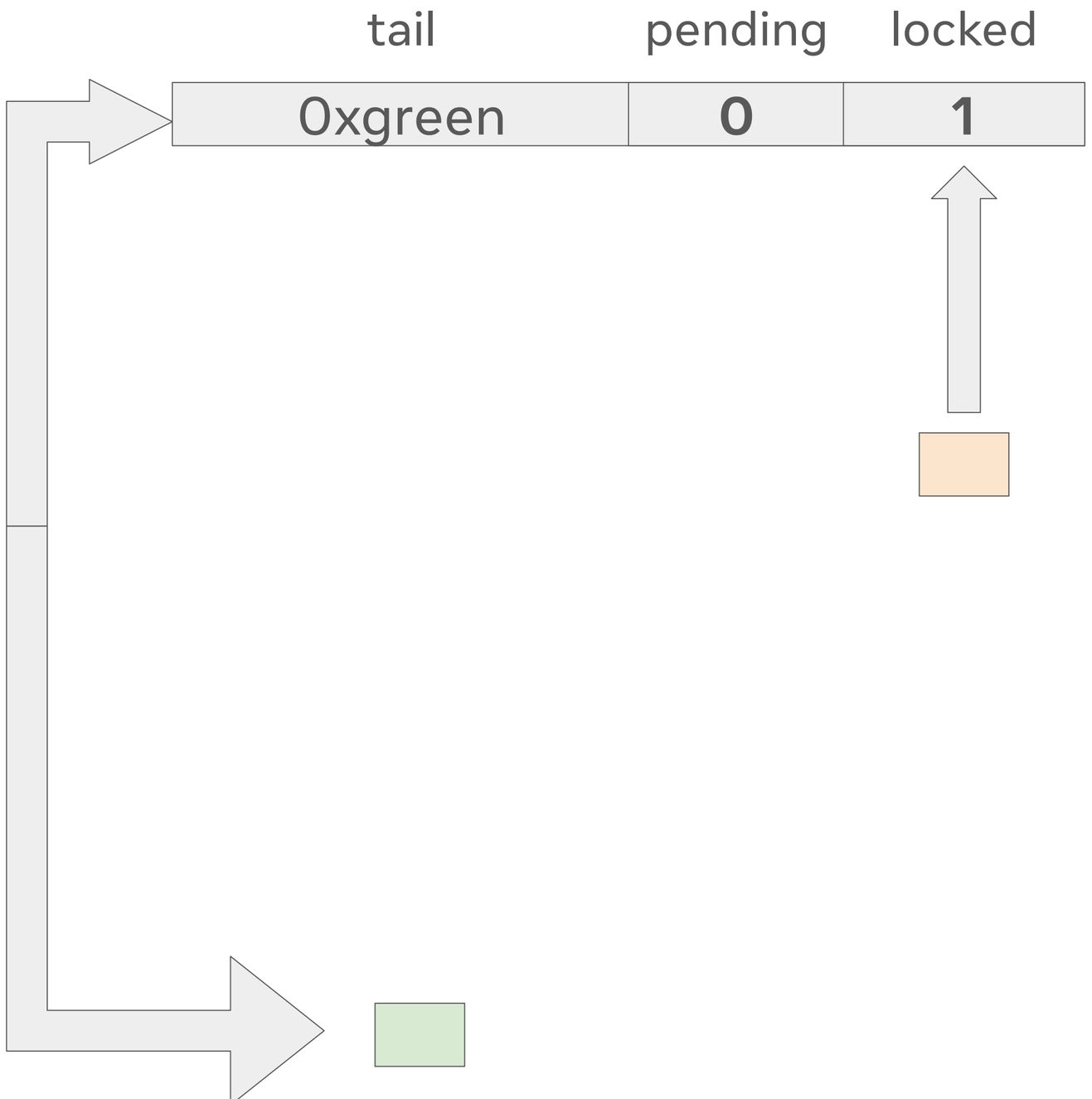

tail
pending
locked
0xgreen
0
1

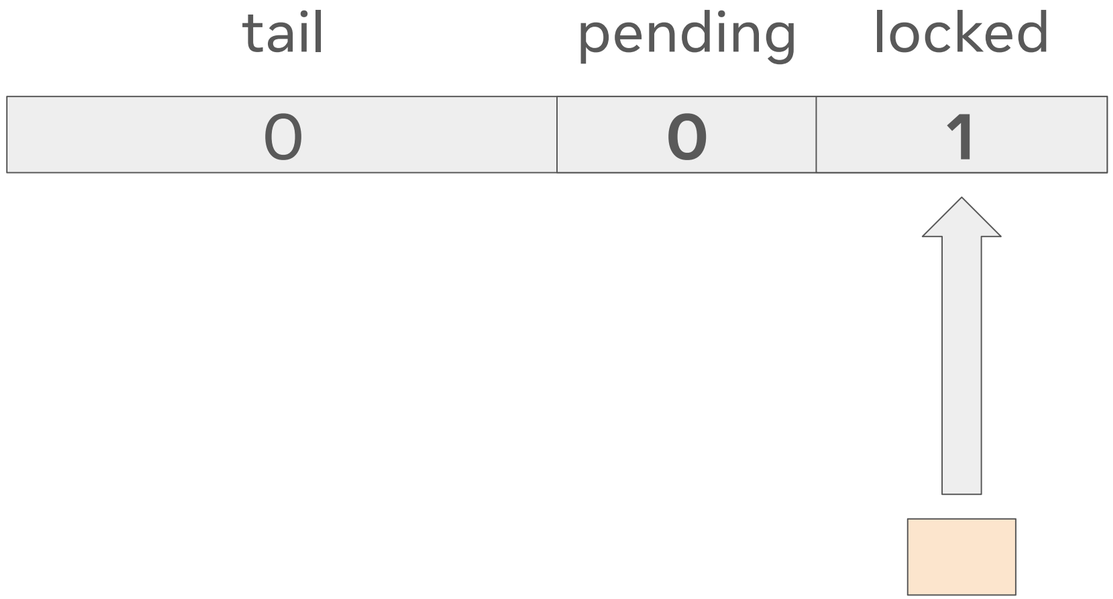

tail
pending
locked
0xgreen
0
1


tail
pending
locked
0xgreen
0
1


tail
pending
locked
0xgreen
0
1

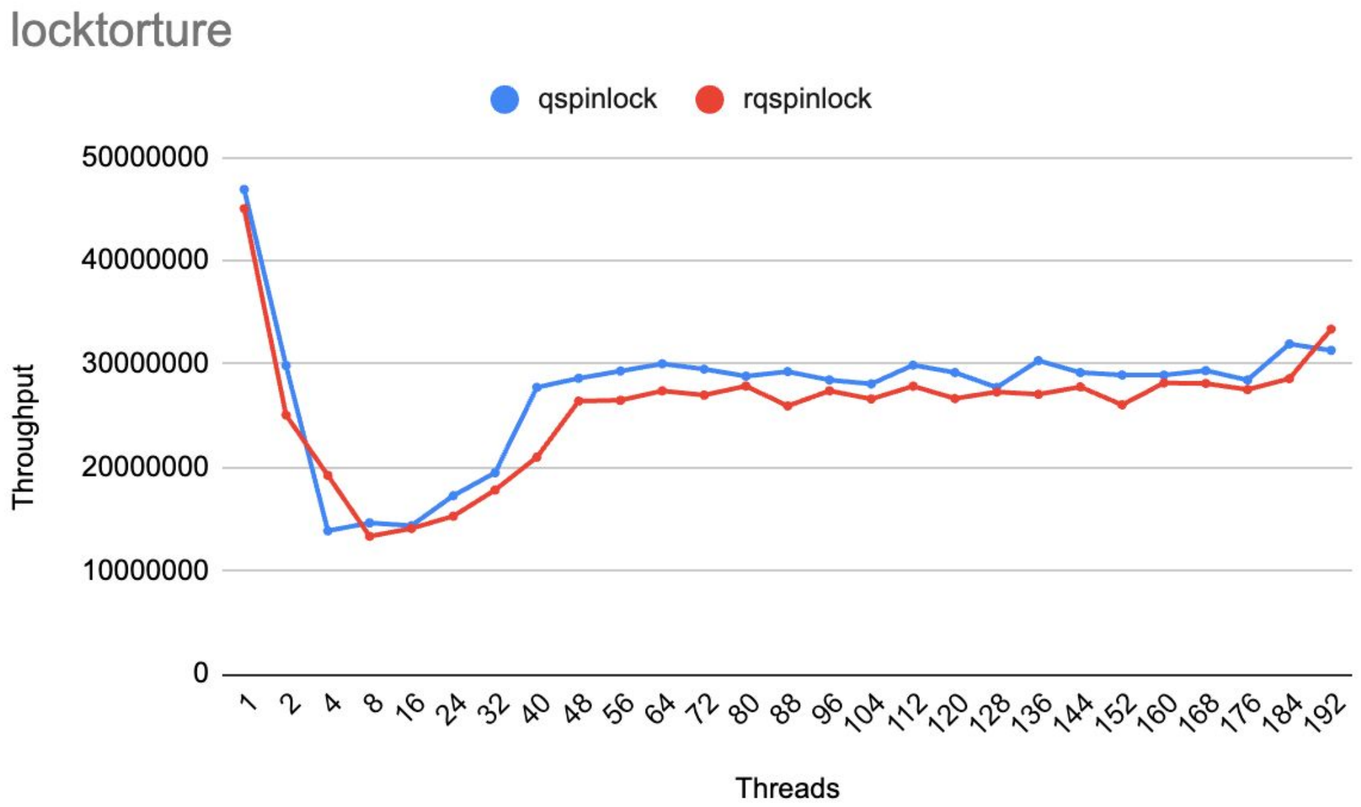

tail
pending
locked
0xgreen
0
1


tail
pending
locked
0xgreen
0
1

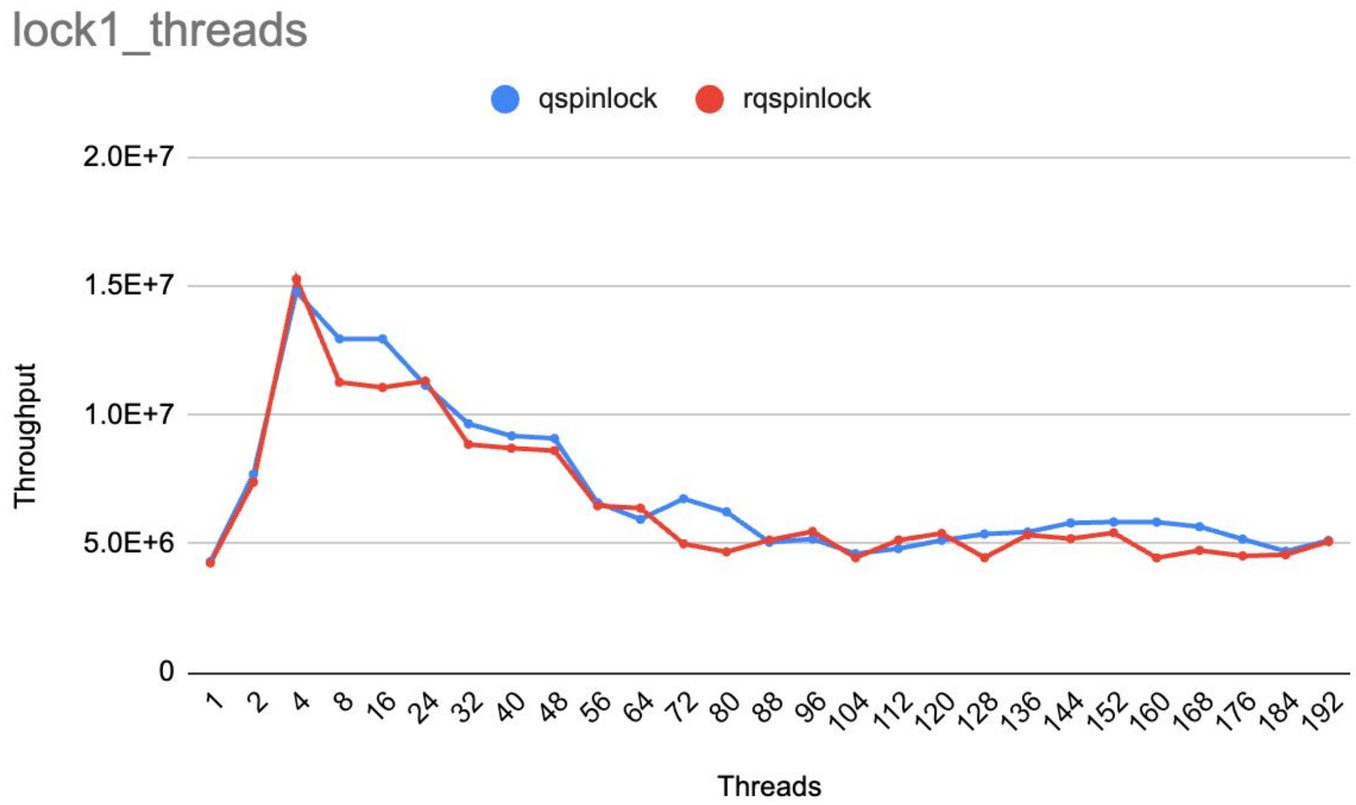

tail
pending
locked
0
0
1

04 Evaluation

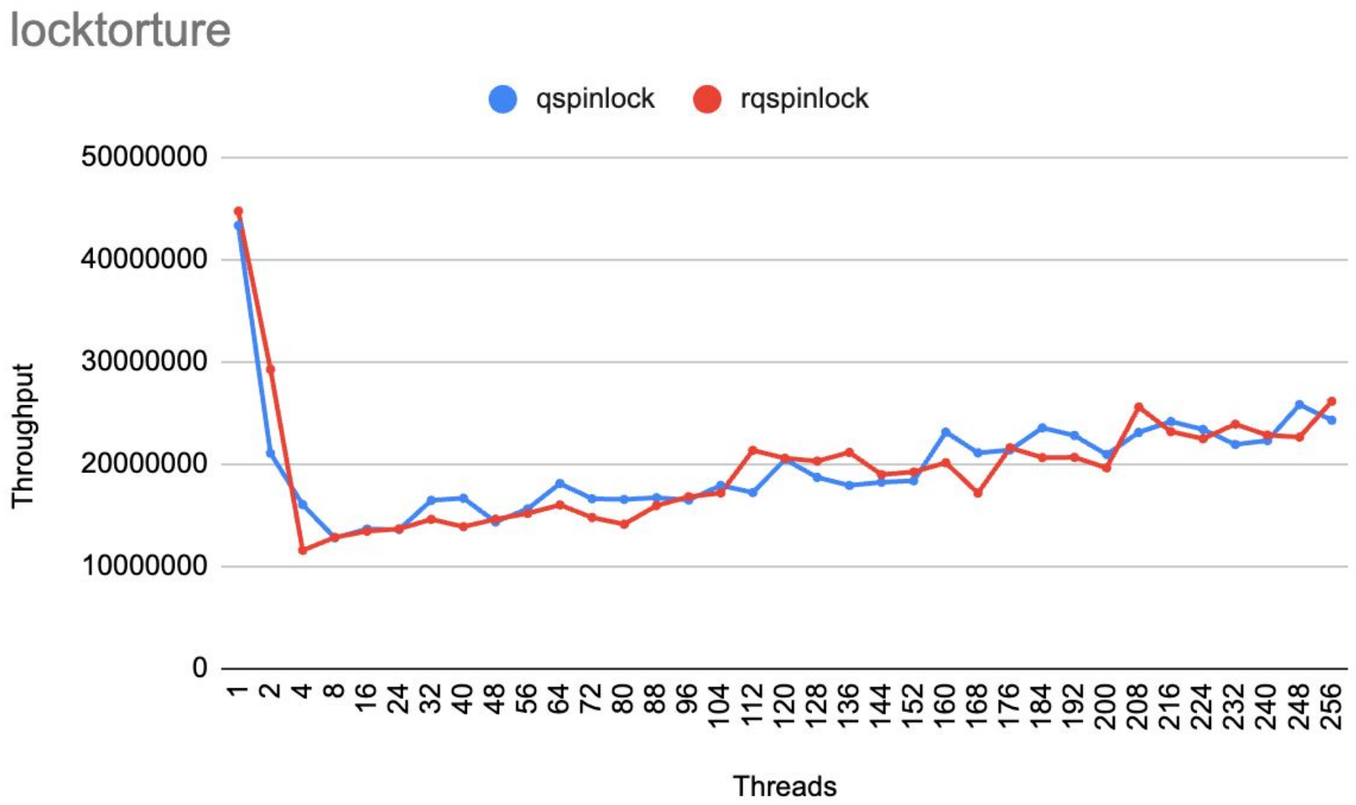
locktorture - x86



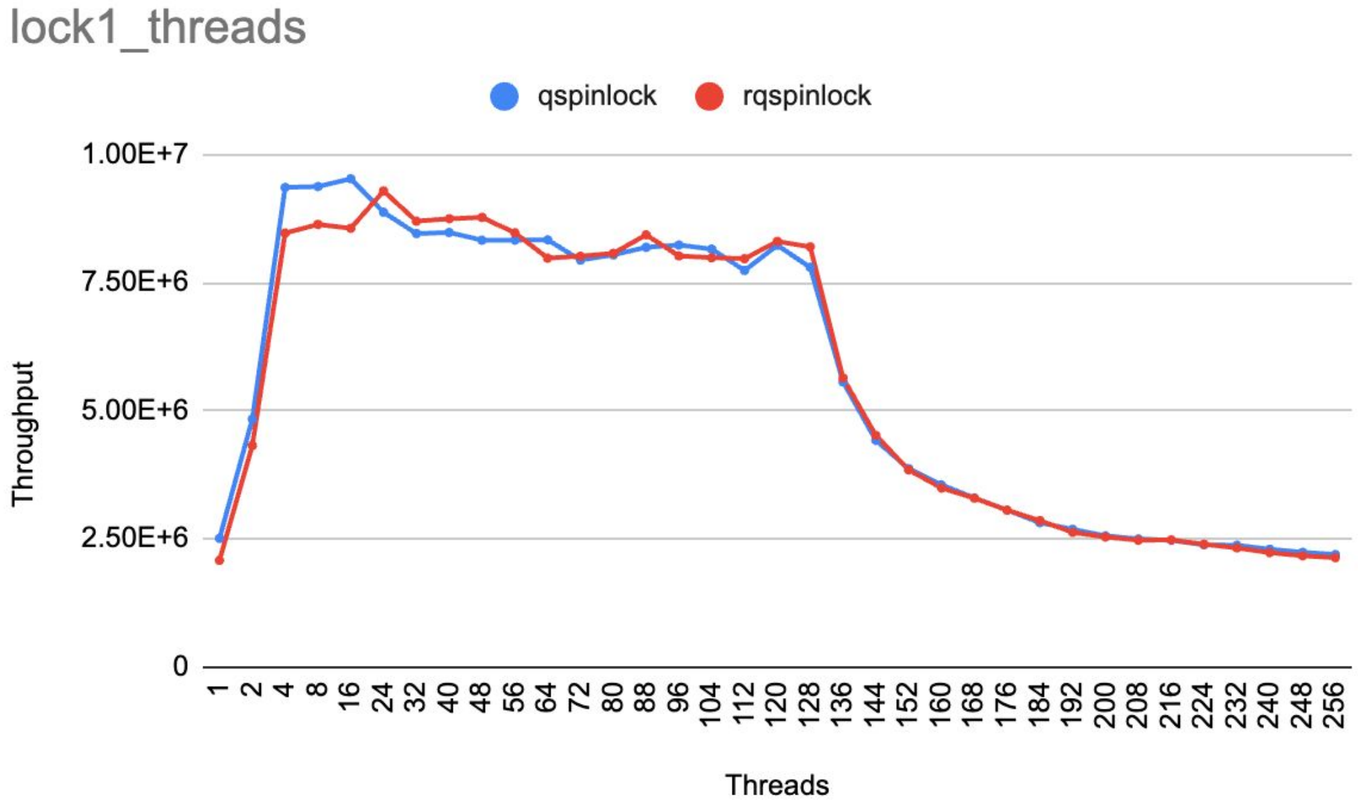
lock1_threads - will-it-scale - x86



locktorture - arm64



lock1_threads - will-it-scale - arm64



05 Next Steps

Use in BPF subsystem

Change BPF runtime to use the new lock to avoid and recover from deadlocks at runtime.

Use in BPF subsystem

Change BPF runtime to use the new lock to avoid and recover from deadlocks at runtime.

Relaxing restrictions

Relax restrictions on function calls and behavior inside critical sections for more flexibility.

Use in BPF subsystem

Change BPF runtime to use the new lock to avoid and recover from deadlocks at runtime.

Relaxing restrictions

Relax restrictions on function calls and behavior inside critical sections for more flexibility.

Reporting violations

Standard output interface per-program to report deadlock-safety violations to user space.

Links

Cover letter (with more numbers):
https://lore.kernel.org/bpf/20250316040541.108729-1-memxor@gmail.com

Algorithmic deep-dive:

https://github.com/kkdwivedi/rqspinlock/blob/main/rqspinlock.pdf


Meta