LINUTRONIX

Formalizing Kernel Synchronization Primitives with PREEMPT_RT

Linux Plumbers Conference
September 20-24, 2021

Refereed Track Session

Ahmed S. Darwish <a.darwish@linutronix.de>

Expected Audience – 1

Kernel and driver developers

- Get a glimpse of locking subsystem development
- See how core subsystems adapt to competing call-site needs

PREEMPT_RT advanced users

- See PREEMPT_RT development behind the scenes ("how the sausages gets made")
- Appreciate the complexity of the task at hand

Expected Audience – 2

Subsystem Maintainers

- See how PREEMPT_RT implicitly and explicitly helps your subsystem :)

Locking Subsystem Maintainers

- Hmm, You already know everything in this talk ;-)

PREEMPT_RT Preliminaries – 1

The Big Picture

- Transform the Linux Kernel, a General Purpose OS, to a hard-realtime system
- while maintaining full user-space ABI compatibility
- and maintaining full in-kernel API comptability; e.g., for all the existing mainline Linux drivers
- That is, you *can* have your cake and eat it too...

PREEMPT_RT Preliminaries – 2

Minimize sources of scheduling latency / interference

Allow scheduling to happen *almost* anywhere. Limit the effects of:

- Interrupts
- Soft interrupts (softirqs)
- Interrupt disable regions
- Preemption disable regions
- Concurrency mechanisms
- ...

PREEMPT_RT Preliminaries – 3

Interrupt handlers

- Force interrupt handlers to run in thread context – thus scheduler and priority controlled

Soft interrupts (softirqs)

- Force softirqs to run in thread context – thus also scheduler and priority controlled

PREEMPT_RT Preliminaries – 4

Spinning locks

- Require preemption, and sometimes irqs, disabled
- Substitute with RT-Mutexes (except raw_spinlock special cases)

Blocking locks

- Use RT-Mutexes with Priority Inheritance (not all blocking locks are covered)
- Documentation/locking/locktypes.rst

Formalizing Kernel Synchronization Primitives with PREEMPT_RT

Sequence Counters & Sequential Locks

Sequence Counters: Definition & Usage

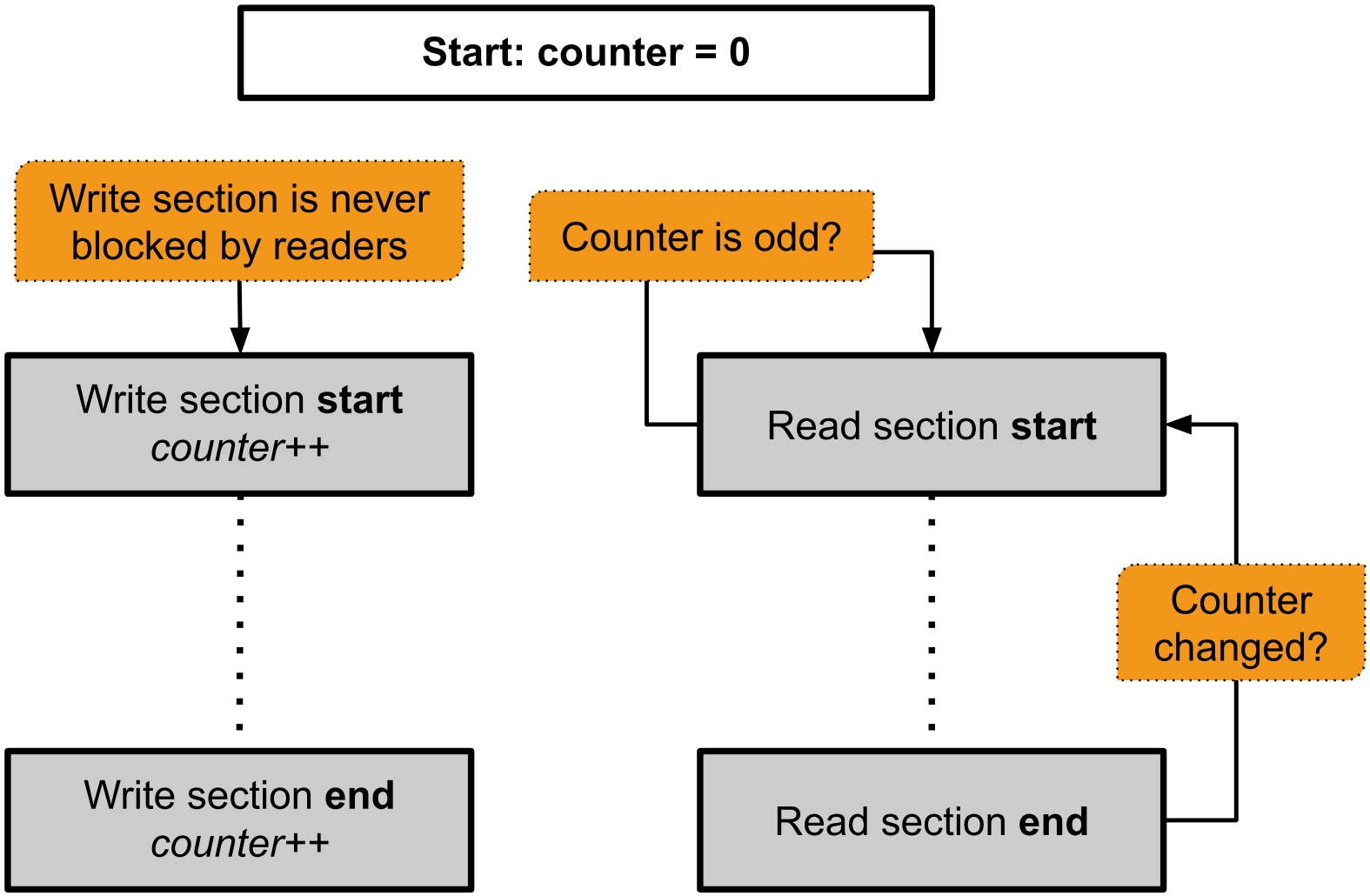
- A reader-writer consistency mechanism with lockless readers (read-only retry loops), and no writer starvation.

- Reader wants a consistent set of information and is willing to retry if that information changes.

Sequence Counters: Definition & Usage (2)

- Usually used for data that's rarely written to (e.g. system time, statistics, ...). Can support an arbitrarily large number of concurrent readers, but only one writer at a time.
- Also commonly used as a cheap trylock mechanism in hot kernel code paths
- Documentation/locking/seqlock.rst

Sequence Counters: Working mechanism



Sequence Counters: Abridged seqlock.h (1)

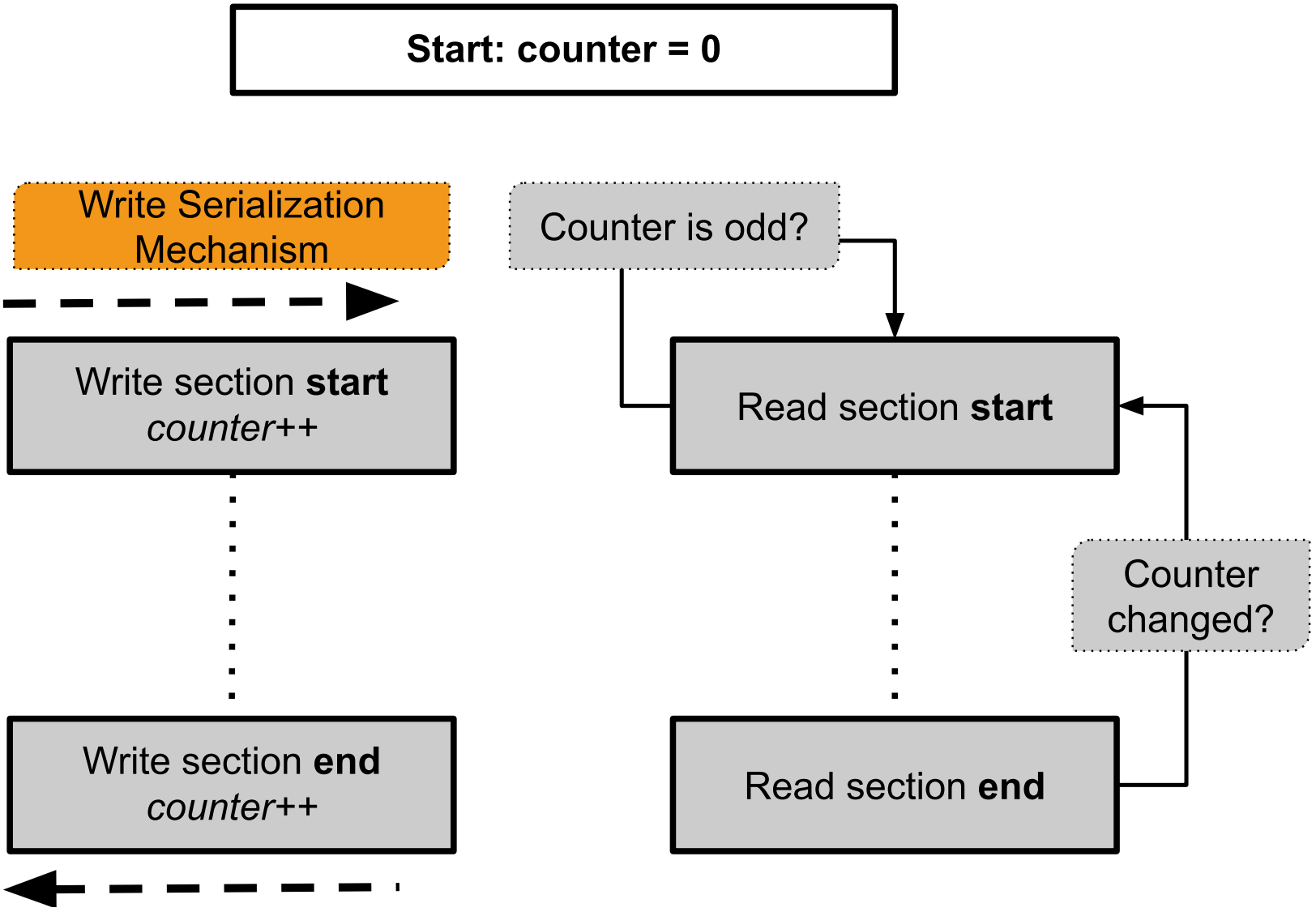
```
static inline void
write_seqcount_begin(seqcount_t *s) {
      s->sequence++;

      // Pairs with smp_rmb() at read_seqcount_retry()
      smp_wmb();
}


static inline void
write_seqcount_end(seqcount_t *s) {
      // Pairs with smp_rmb() at read_seqcount_begin()
      smp_wmb();

      s->sequence++;
}
```

Sequence Counters: Abridged seqlock.h (2)

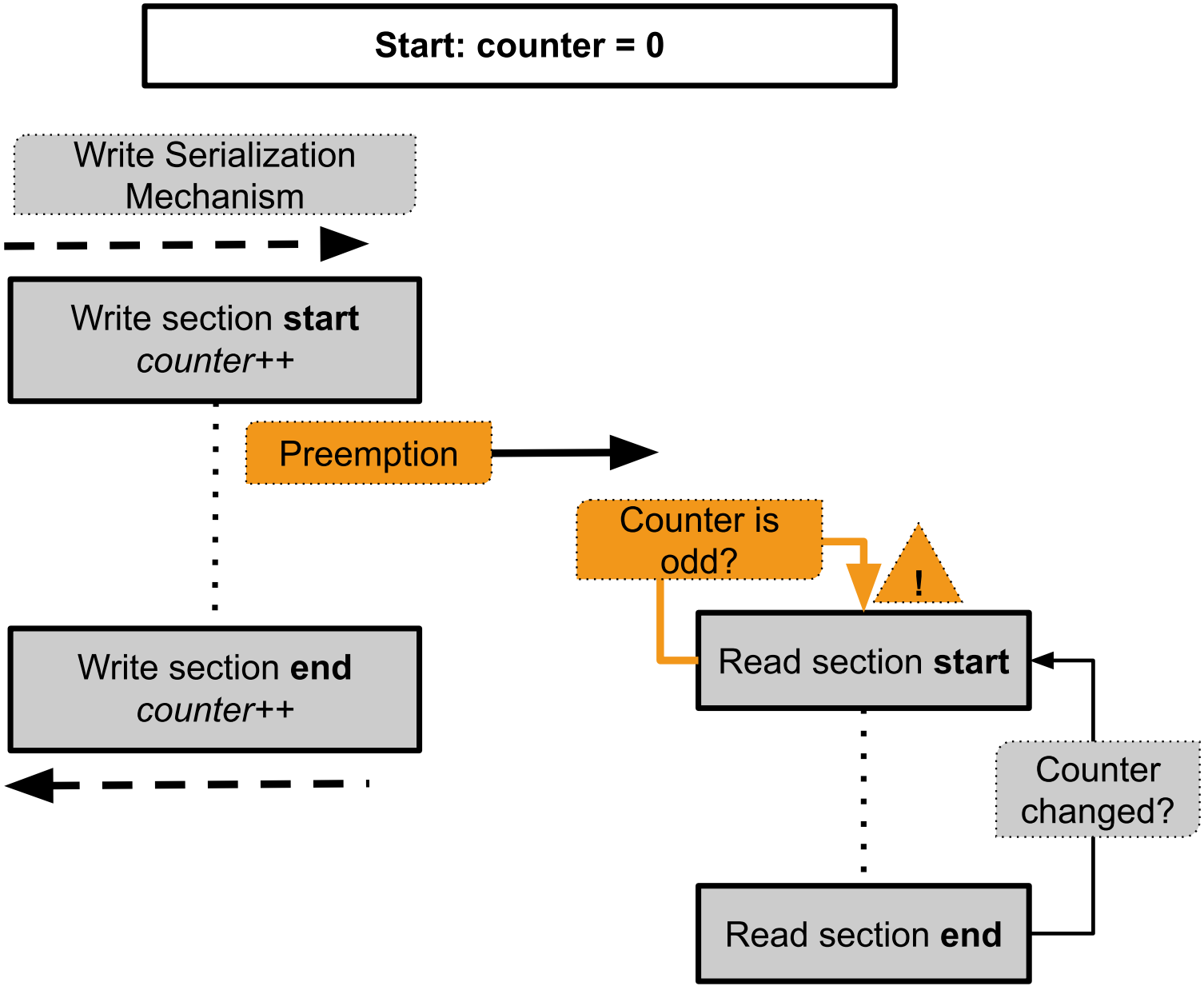
```
static unsigned
read_seqcount_begin(seqcount_t *s) {
      while ((__seq = s->sequece) & 1))
            cpu_relax();
      smp_rmb();
      return __seq;
}
```

```
static int
read_seqcount_retry(seqcount_t *s, unsigned start) {
      smp_rmb();
      return unlikely(s->sequence != start);
}
```

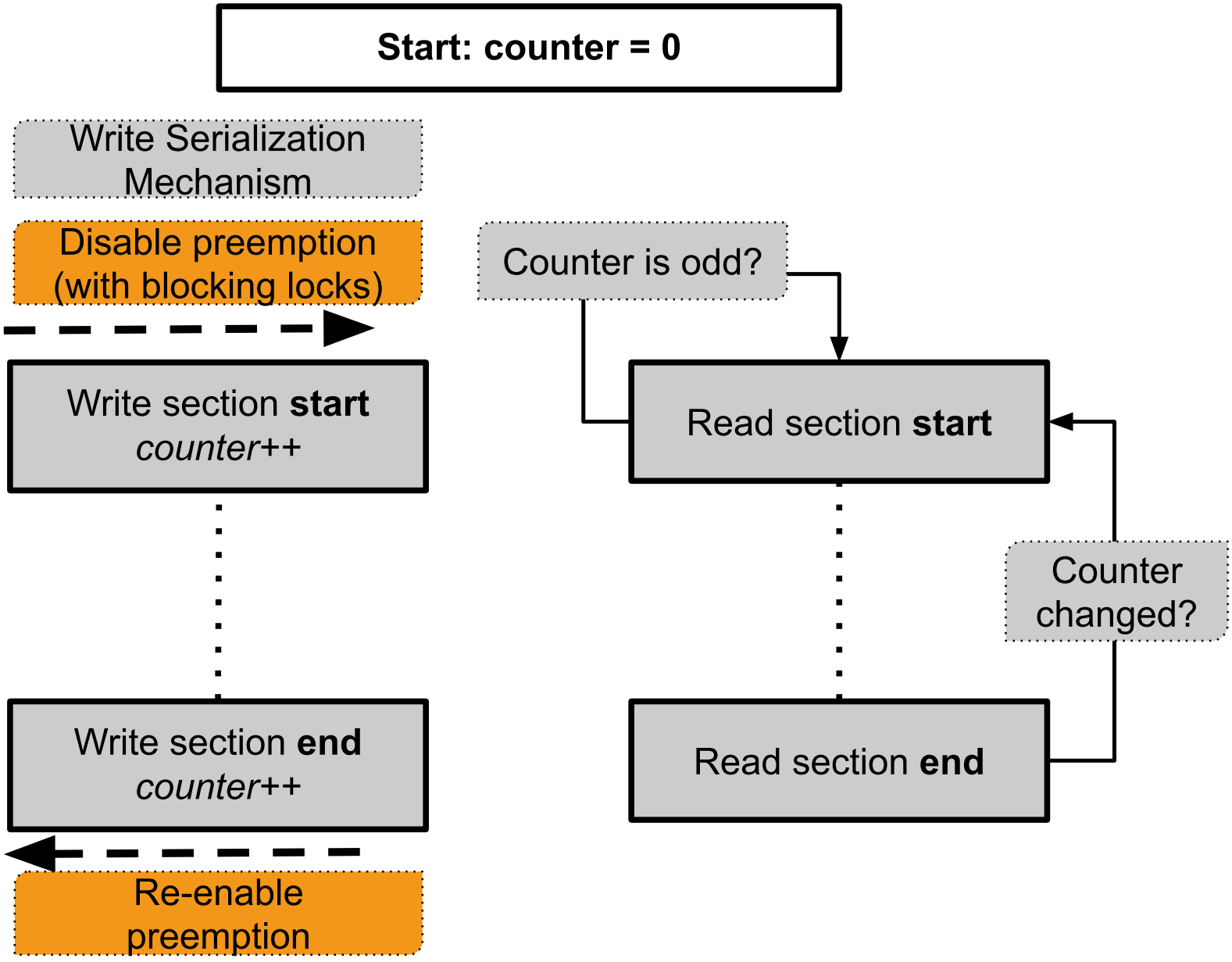
Seqcount Req. 1: Write Serialization



Seqcount Req. 2: Preemption Disable



Seqcount Req. 2: Preemption Disable (2)



Sequence Counters: Example Usage

Write side

```
mutex_lock(&x);
…
preempt_disable();
write_seqcount_begin(&foo_seqcount);
…
write_seqcount_end(&foo_seqcount);
preempt_enable();
…
mutex_unlock(&x);
```

Sequence Counters: Example Usage (2)

Write side

```
spin_lock(&x);    // or spin_lock_irqsave/bh()
…
write_seqcount_begin(&foo_seqcount);
…
write_seqcount_end(&foo_seqcount);
…
spin_unlock(&x);
```

Sequence Counters: Example Usage (3)

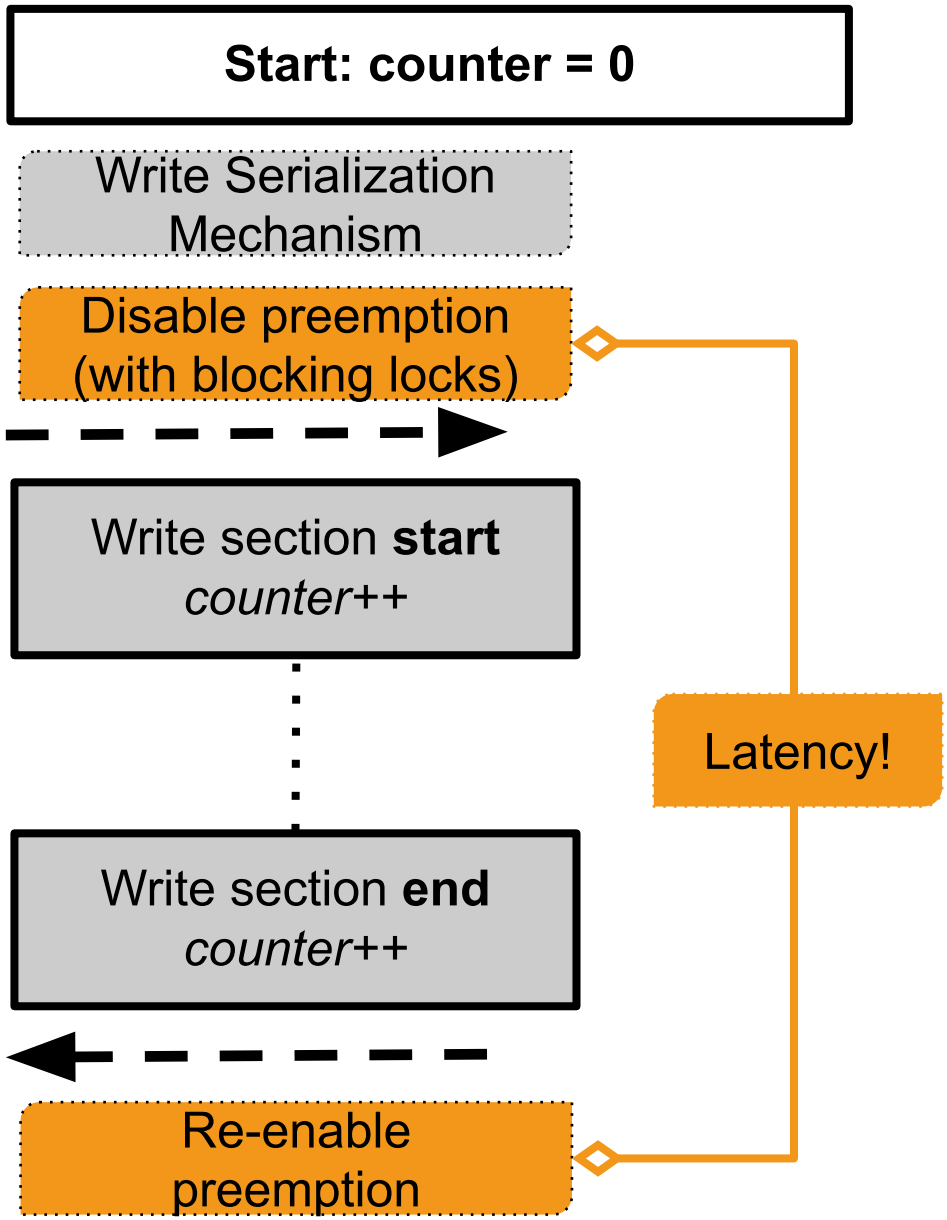
Read side

```
do {
        seq = read_seqcount_begin(&foo_seqcount);
        ...
} while (read_seqcount_retry(&foo_seqcount, seq));
```

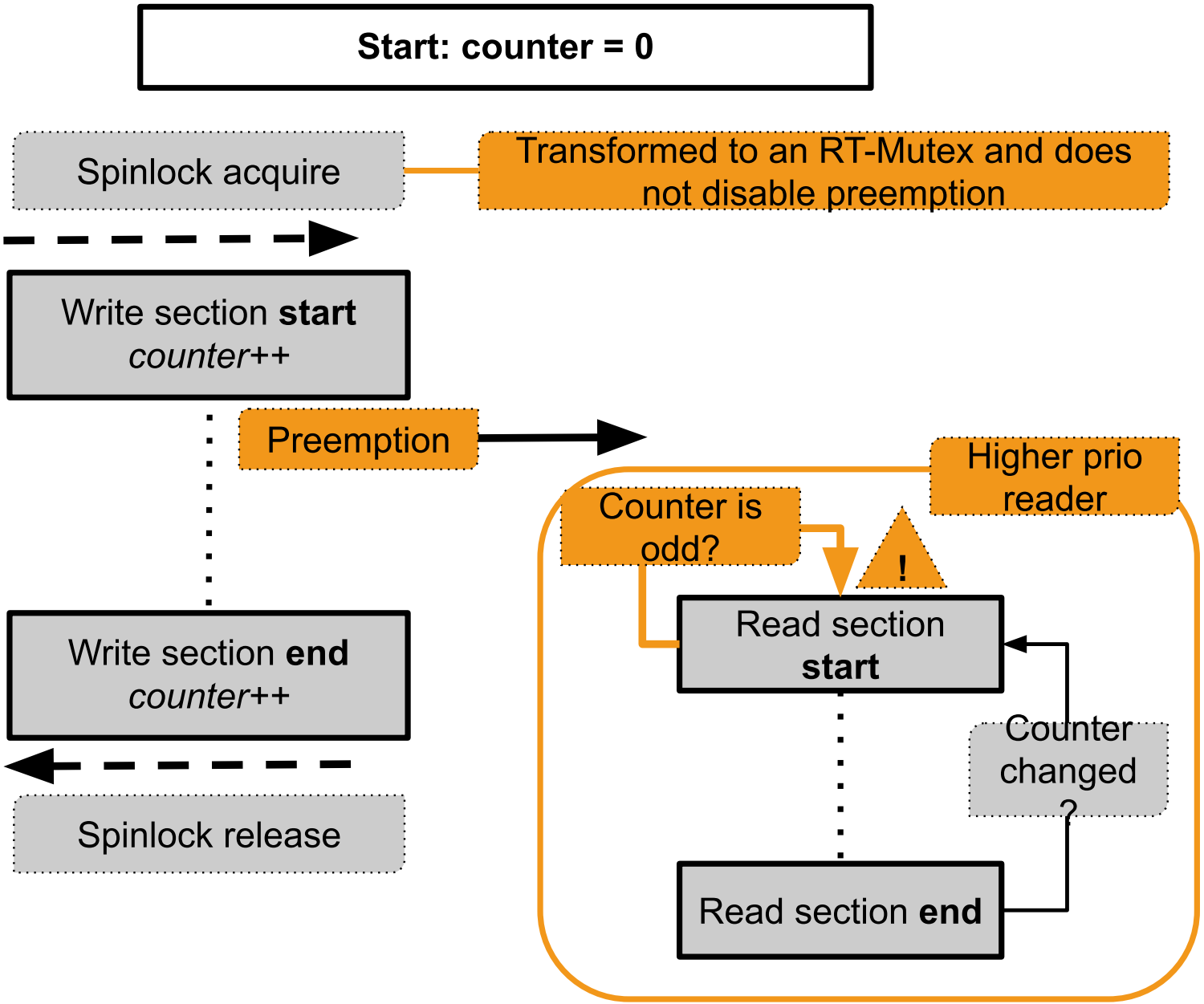
Sequence Counters

Problems for PREEMPT_RT

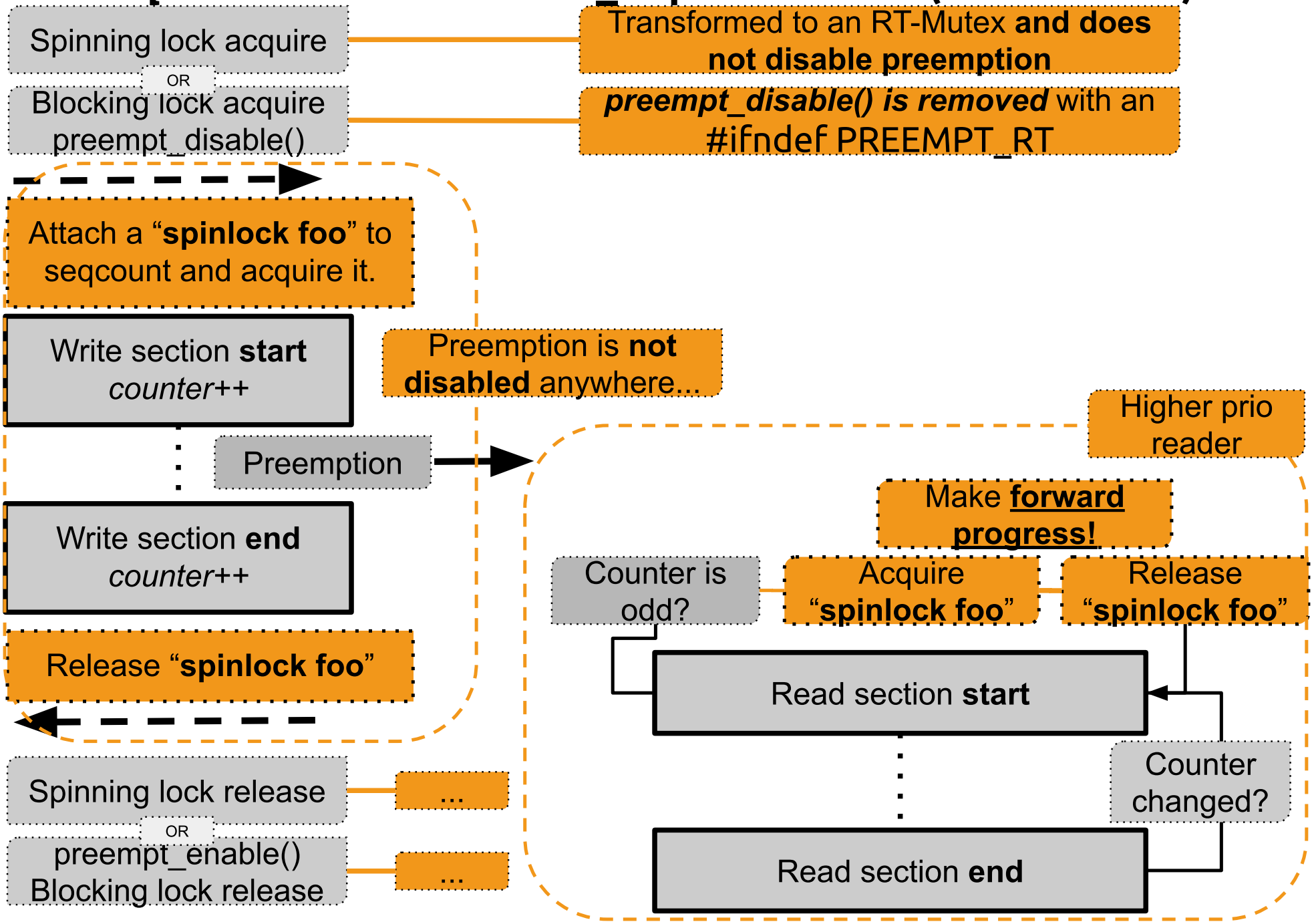
Sequence Counters: Problems for PREEMPT_RT (1)



Sequence Counters: Problems for PREEMPT_RT (2)



Seqcount: Previous PREEMPT_RT patch solution (non-mainline)



Advantages and Disadvantages of Previous Solution

Advantages

- Solves the latency and reader infinite loops issues

Disadvantages

- Requires call-site modifications when blocking locks are used for writer serialization. Polluting drivers and subsystem code with “#ifdef PREEMPT_RT” is *not* acceptable.
- Introduces an extra lock, which can make performance-sensitive subsystems’ maintainers hesitant (even though RT is not about throughput, *minimally-intensive* solutions are always preferred).

Searching for solutions...

Do we really need the extra lock?

- The extra lock attached to the seqcount was only added in the PREEMPT_RT patch to accomplish the lock-unlock operation for reader forward progress.
- Why? Because the seqlock.h code does not have a reference to the write serialization lock.
- If we attach the write serialization lock to the seqcount, that extra lock will not be needed.
- Thomas Gleixner (tglx; RT lead) asked for a survey of all seqcount_t call sites...

Seqcount_t Call-sites Survey (1)

Survey Purpose

- Get an understanding of call-sites seqcount usage and behavior
- Verify that “associating the write serialization lock to the seqcount_t” solution will be applicable to almost all call sites
- Survey is attached to presentation materials folder: “seqcount_call-sites_survey.ods”.

Seqcount_t Call-sites Survey (2)

Survey results

- All call-sites were analyzed and surveyed; *26 in total*.
- Multiple call-sites forgot to manually disable preemption when using blocking locks for writer serialization
- Some call-sites abused sequence counters API or used them in wrong contexts. Such cases were substituted with alternative mechanisms.

Seqcount_t Call-sites Survey (3)

Resulting bug-fixes and cleanups to multiple subsystems:

- **mm/swap: Do not abuse the seqcount_t latching API**
- **net: core: device_rename: Use rwsem instead of a seqcount**
- u64_stats: Document writer non-preemptibility requirement
- **net: mdiobus: Disable preemption upon u64_stats update**
- **block: nr_sects_write(): Disable preemption on seqcount write**
- net: phy: fixed_phy: Remove unused seqcount
- seqlock: lockdep assert non-preemptibility on seqcount_t write
- dma-buf: Remove custom seqcount lockdep class key
- ...

Seqcount_t Call-sites Survey (4)

Add guards to avoid future call-site bugs:

- Add lockdep preemption context assertions
- Add lockdep “serialization lock held” assertions
- Add explicit kernel-doc for all seqcount and seqlock exported APIs
- Add “big picture” documentation under Documentation/locking

Seqcount_t Call-sites Survey (5)

Final conclusion

- After all the call-site bugfixes and cleanups, especially after converting sites abusing the seqcount_t API, it was apparent that remaining call sites can be converted to a “sequence counters with associated writer serialization lock” design.

New sequence counters API: seqcount_LOCKNAME_t

Differs by writer serialization type

- seqcount_spinlock_t
- seqcount_raw_spinlock_t
- seqcount_rwlock_t
- seqcount_mutex_t
- seqcount_ww_mutex_t

New sequence counters API: seqcount_LOCKNAME_t (2)

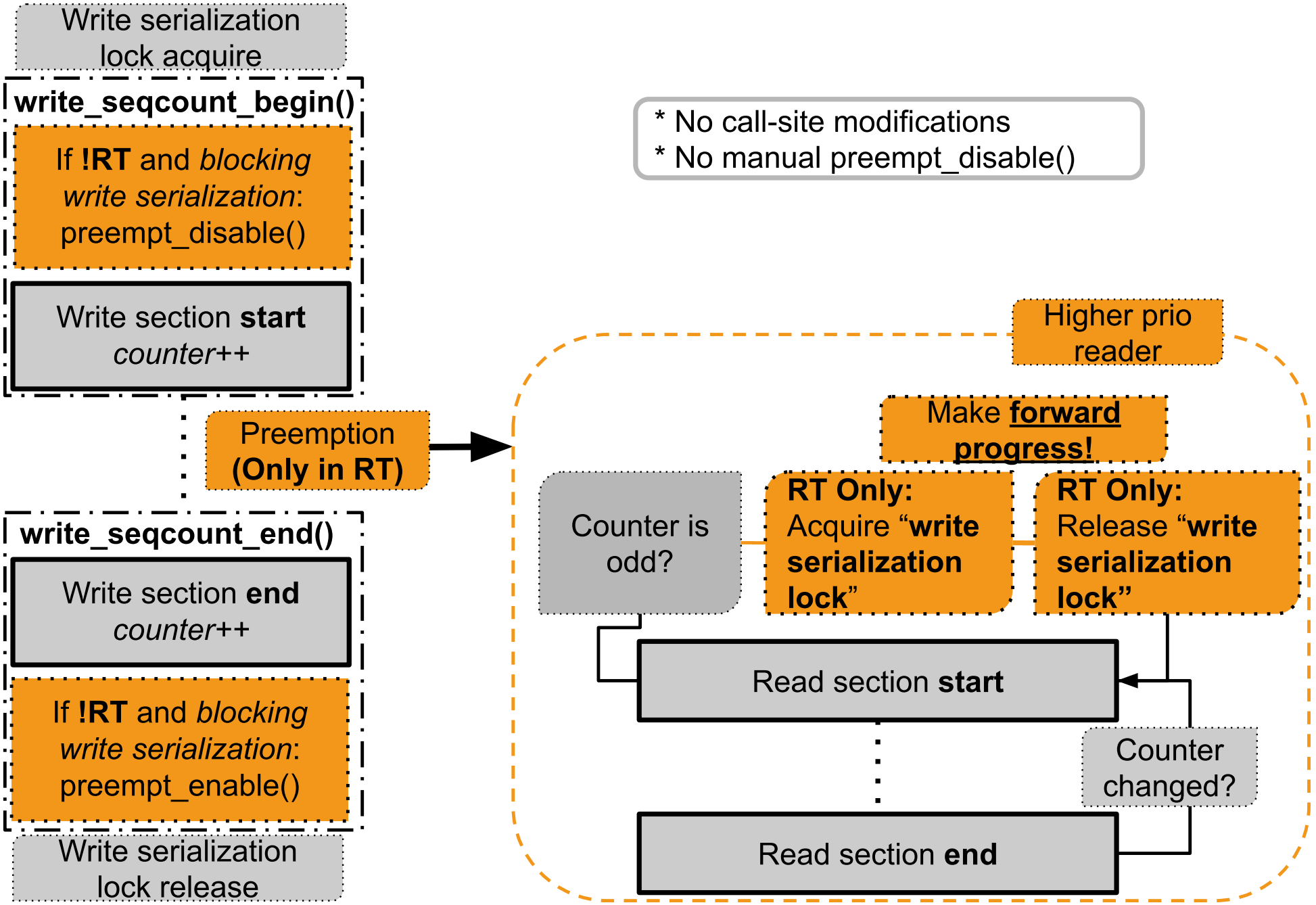
Benefits

- Internally asserts that the write serialization lock is always held for all write sections...
- More reliability: in case of blocking write serialization locks, job of disabling preemption is moved from call-sites to “seqlock.h” internal implementation.
- Last point is *critical* for RT: preempt_disable() should no longer be done by call-sites on their own. It is “seqlock.h” responsibility. No more latency impacts.

New sequence counters API: seqcount_LOCKNAME_t (3)

```
seqcount_mutex_t foo_seq;
struct mutex mut;

mutex_init(&mut);
seqcount_mutex_init(&foo_seq, &mut);

/* triggers lockdep fail (mutex not acquired) */
write_seqcount_begin(&foo_seq);
write_seqcount_end(&foo_seq);

/* Acquire write serialization lock */
mutex_lock(&mut);

/* Inside the sequence counter write section, preemption will be
 * automatically disabled for !RT. For RT, preemption is kept
 * enabled */
write_seqcount_begin(&seq);
...
write_seqcount_end(&seq);

/* Release write serialization lock */
mutex_unlock(&mut);
```

seqcount_LOCKNAME_t: Summary (3)



Questions / Comments

Thank you for your attention.

a.darwish@linutronix.de
info@linutronix.de

Appendix: Latch sequence counters

Summary

Sequence counters with multiversion concurrency where the read section can safely preempt or interrupt the write section.

Formalization

Implementation was earlier done through manual read and write accessors. Proper abstractions were created as part of the sequence counters PREEMPT_RT work.

Jon perfectly covered it in the LWN article: The seqcount latch lock type

After the formalizations, two new call-sites now exist at core printk code :-)

Appendix: Mainline status

Status

All formalizations mentiond in this talk are already merged mainline.

Submissions

Since patch series cover letters provide even more context, here is a list of the most relevant discussions:

- [PATCH v1→v4 00/25] seqlock: Extend seqcount API with associated locks
- [PATCH v2 0/6] seqlock: seqcount_t call sites bugfixes
- [PATCH v2 0/5] seqlock: Introduce PREEMPT_RT support
- [PATCH v1 0/8] seqlock: Introduce seqcount_latch_t
- [PATCH -tip v1 0/3] seqlock: assorted cleanups

Pull Requests

Pull requests typically provide a “big picture” context. For this work:

- [GIT pull] locking/urgent for 5.9-rc1
- [GIT PULL] locking changes for v5.10

Appendix: Further readings

Synchronization Primitives Development

- Is Parallel Programming Hard, And, If So, What Can You Do About It?
  Paul E. McKenney

- A Primer on Memory Consistency and Cache Coherence, 2nd Edition
  Vijay Nagarajan et al. *Synthesis Lectures on Computer Architecture*

PREEMPT_RT

- A guided tour through the Preempt-RT castle
  Thomas Gleixner. *ELISA May 2021 special*

- Real Time is Coming to Linux; What Does that Mean to You?
  Steven Rostedt. *Embedded Linux Conference 2018 Europe*