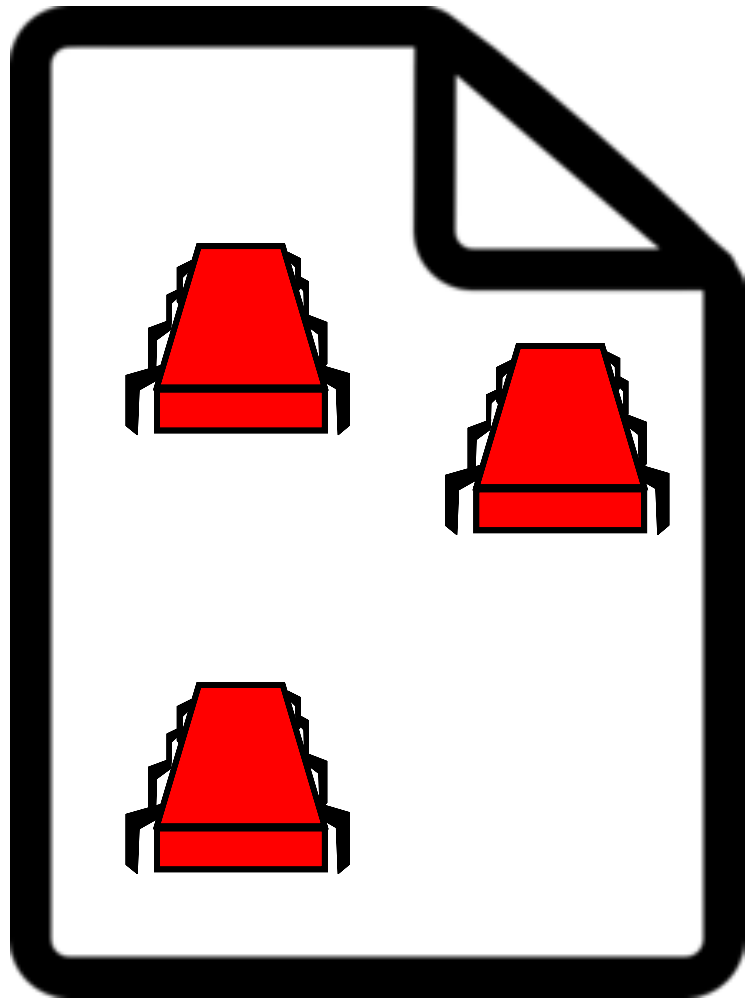
Scalable Range Locks for Scalable Address Spaces And Beyond

Alex Kogan   Dave Dice   **Shady Issa**

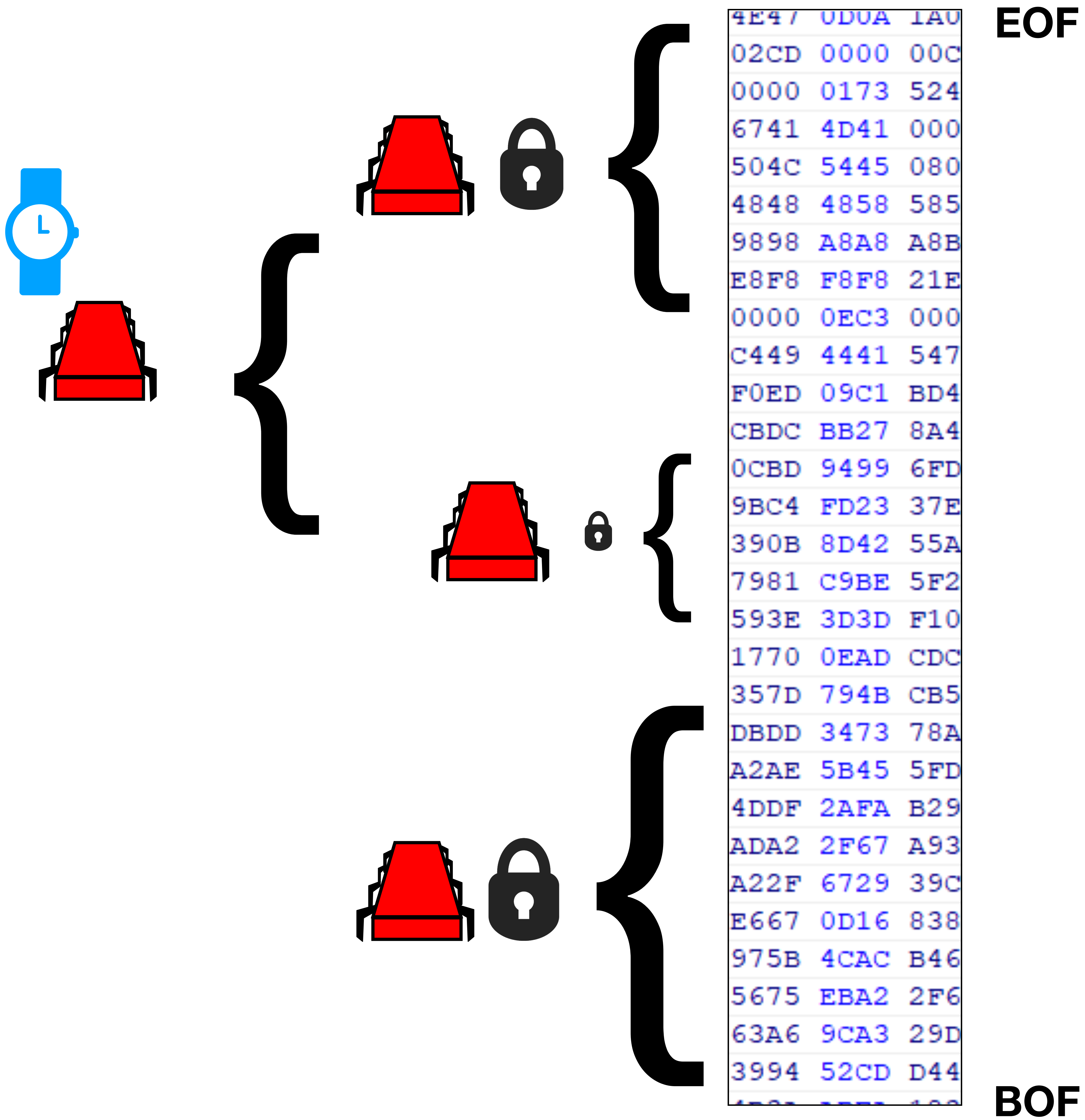
Oracle Labs   U. Lisboa & INESC-ID

Range Locks

- Conceived in parallel filesystems
- Allow concurrent access to shared resources
  - e.g.: writing to the same file



Range Locks



Linux kernel Scalability Bottleneck

How to get rid of mmap_sem

Please consider subscribing to LWN

Subscriptions are the lifeblood of LWN.net. If you appreciate this conte
and would like to see more of it, your subscription will help to ensure t
LWN continues to thrive. Please visit this page to join up and keep LW
the net.

[RFC 0/4] Replace mmap_sem by a range lock

Laurent Dufour | Wed, 19 Apr 2017 05:19:10 -0700

By **Jonathan Corbet**
May 8, 2019

LSFMM

The `mmap_sem` lock used in the memory-management subsystem has been a known scalability problem for years, but it has proved difficult to ren
management track of the 2019 Linux
`mmap_sem` and how it might be elimina
vague at best.

[RFC PATCH 00/64] mm: towards parallel address space operations

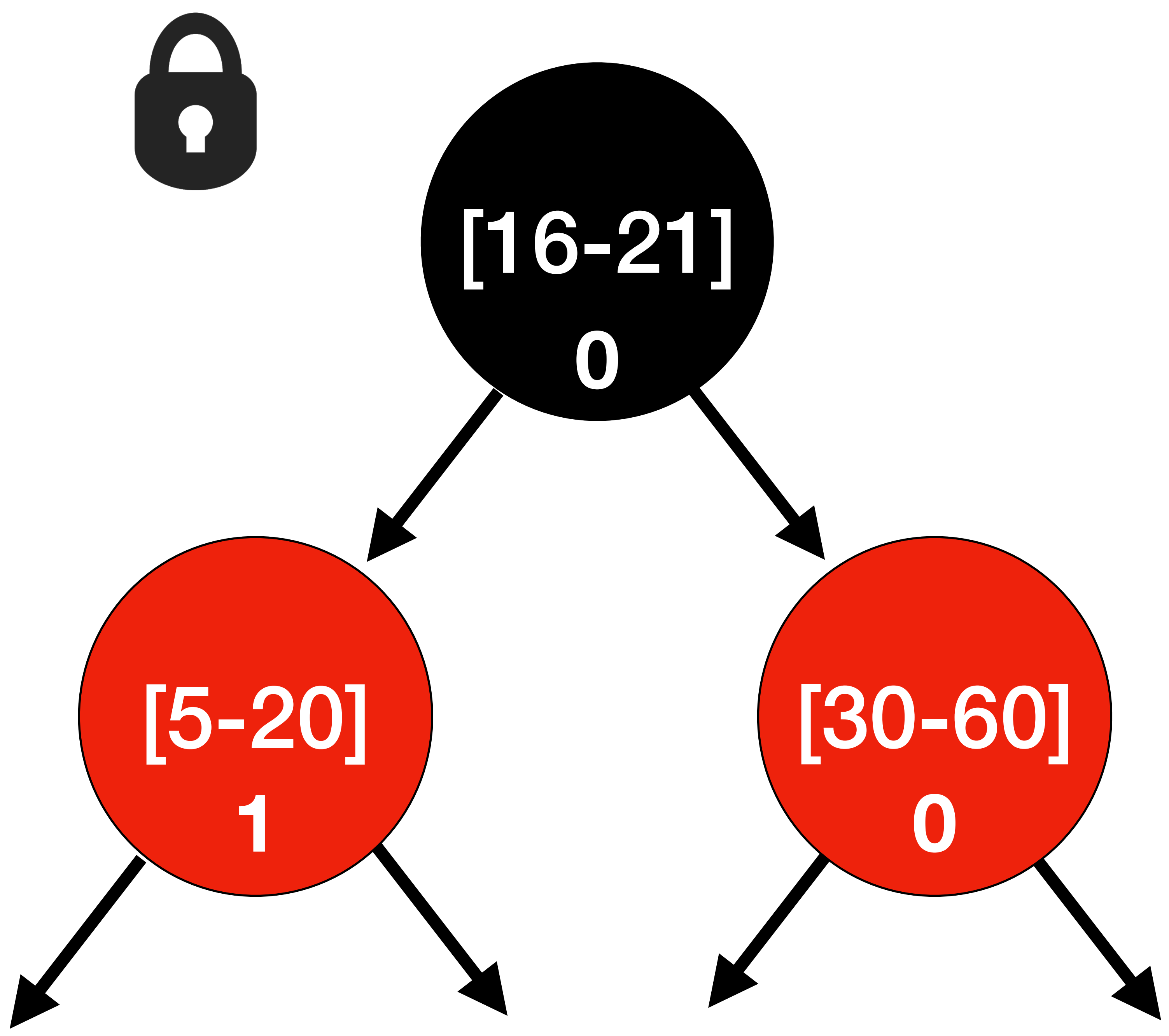
Davidlohr Bueso | Sun, 04 Feb 2018 17:30:07 -0800

[PATCH v3 -tip 0/6] locking: Introduce range reader/writer lock

Davidlohr Bueso | Mon, 15 May 2017 02:08:54 -0700

Existing Range Locks

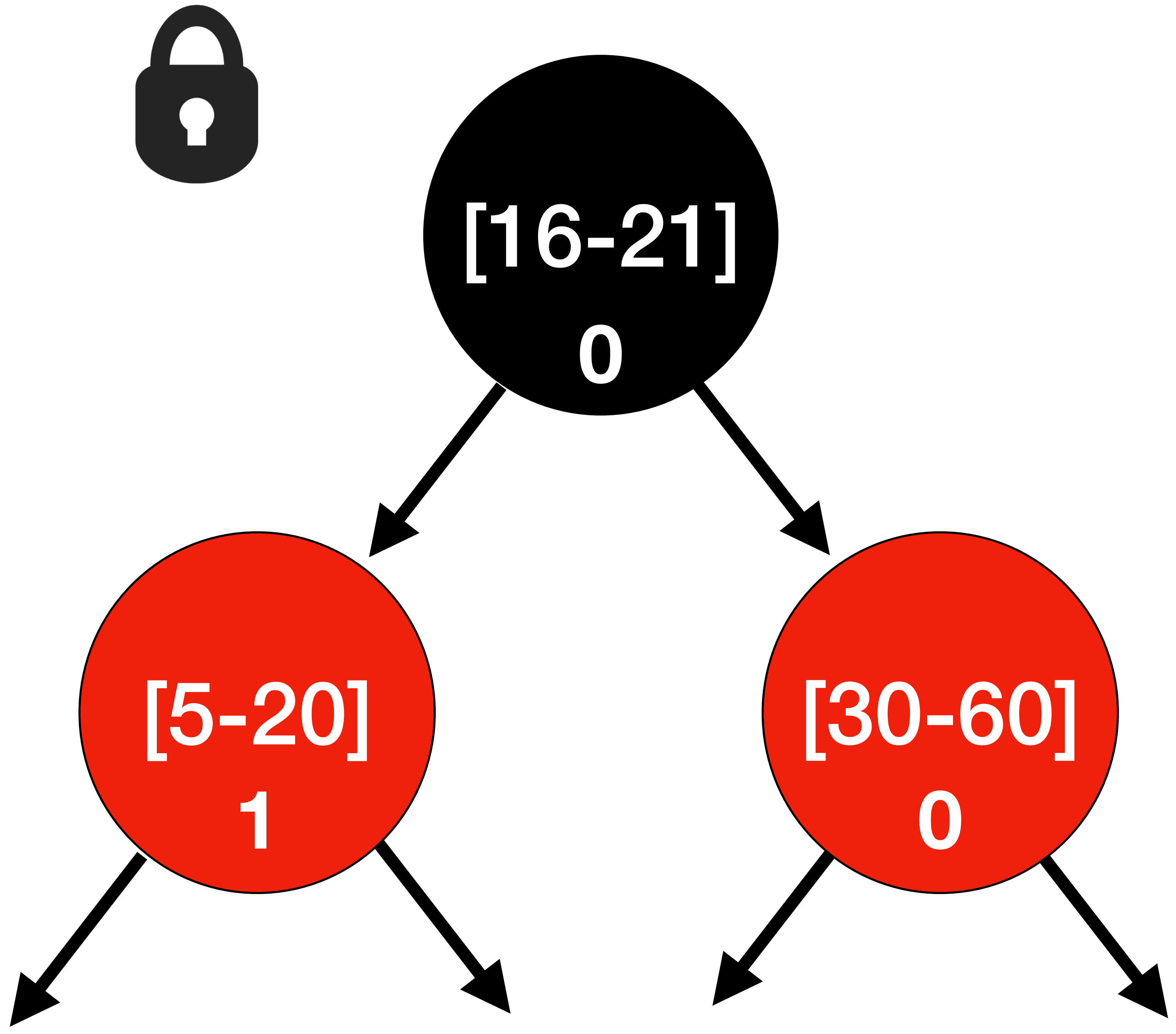
- Auxiliary red-black tree
  - Ranges sorted by starting address
- Protected by spin-lock
  - contention even for shared access



Existing Range Locks

Current RL are not scalable

- Aux… -tree
    - Ranges sorted by starting address
- Protected by spin-lock
    - contention even for shared access



Our Contributions

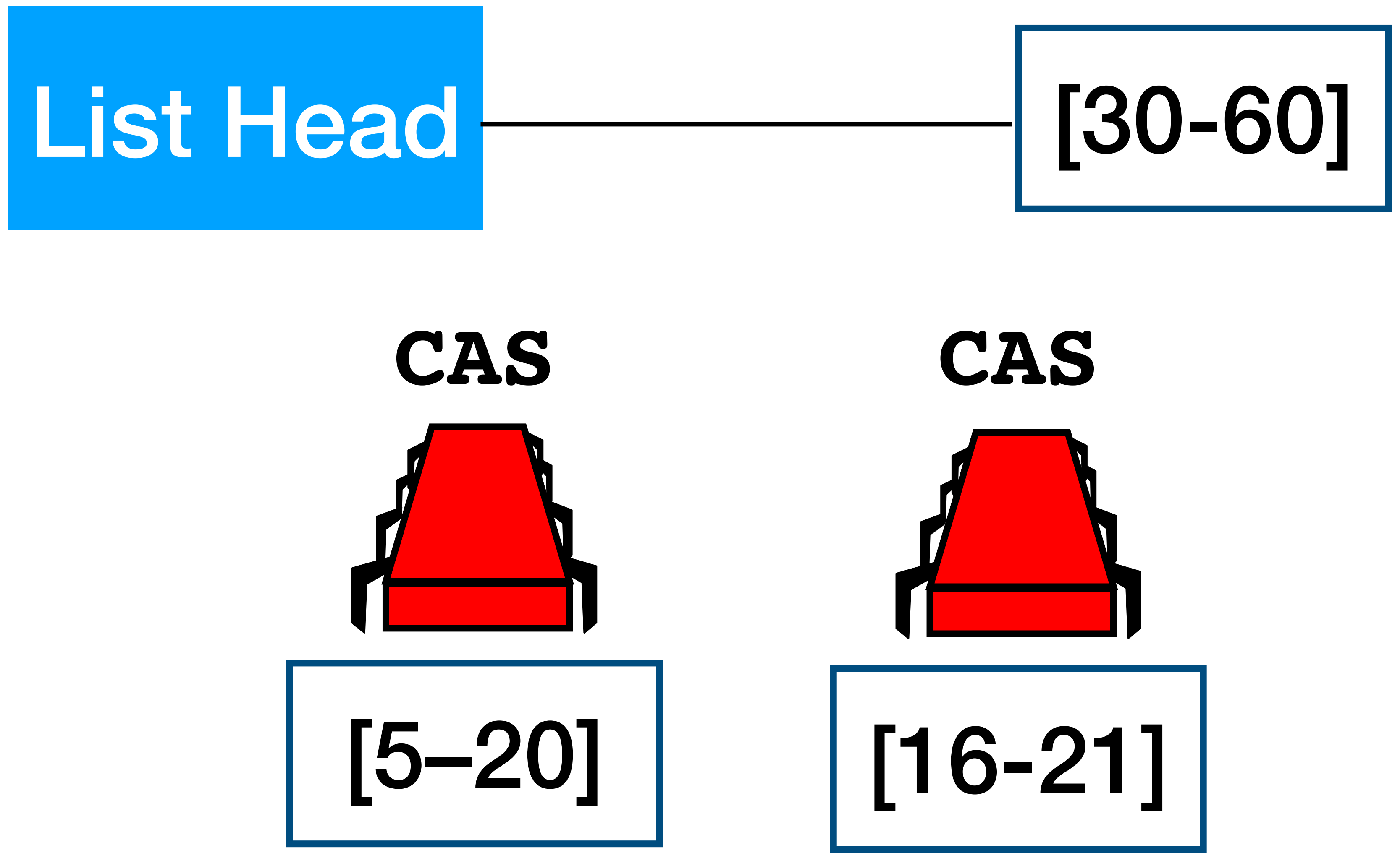
- New design for Range locks
  - Lock-free in the common case
  - Scales up to **144 threads**
- Speculative approach for VM operations in the Linux kernel
- Range locks for skip lists

List-based Range Locks

- A range lock is acquired once a range is inserted into a list
  - Sorted by their starting addresses

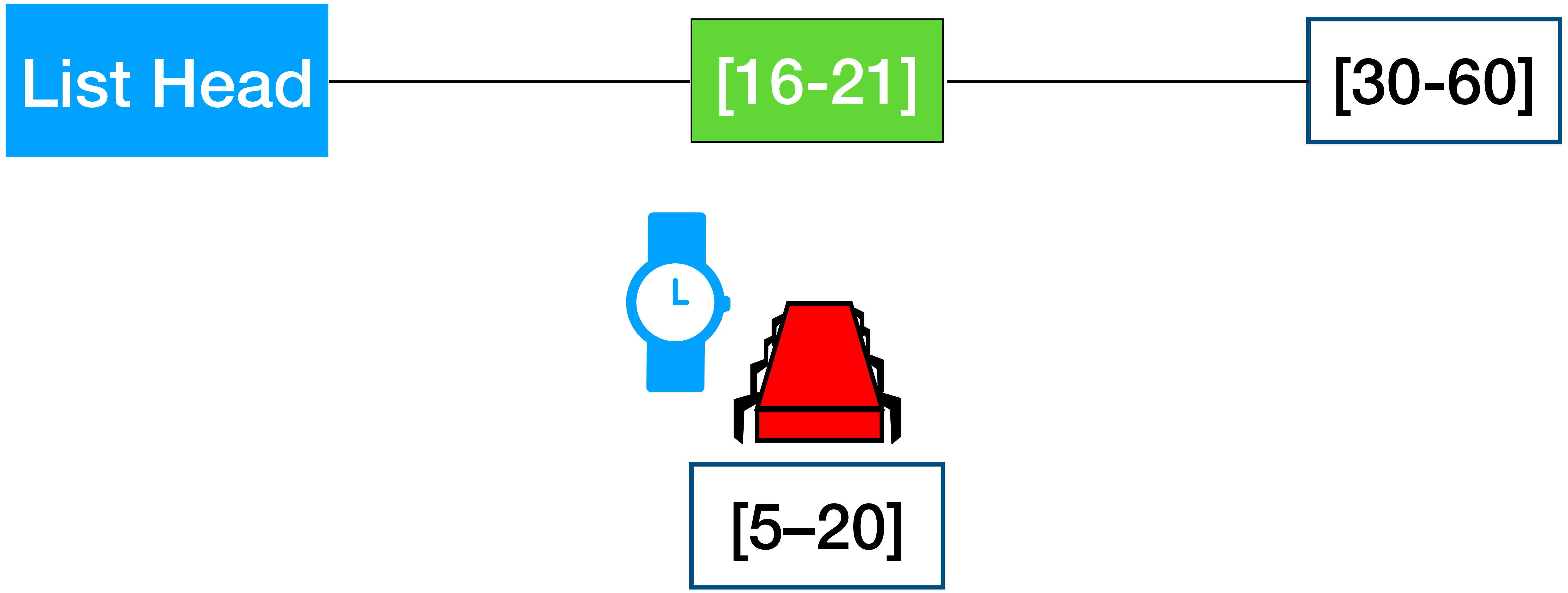
List-based Range Locks

- A range lock is acquired once a range is inserted into a list
  - Sorted by their starting addresses



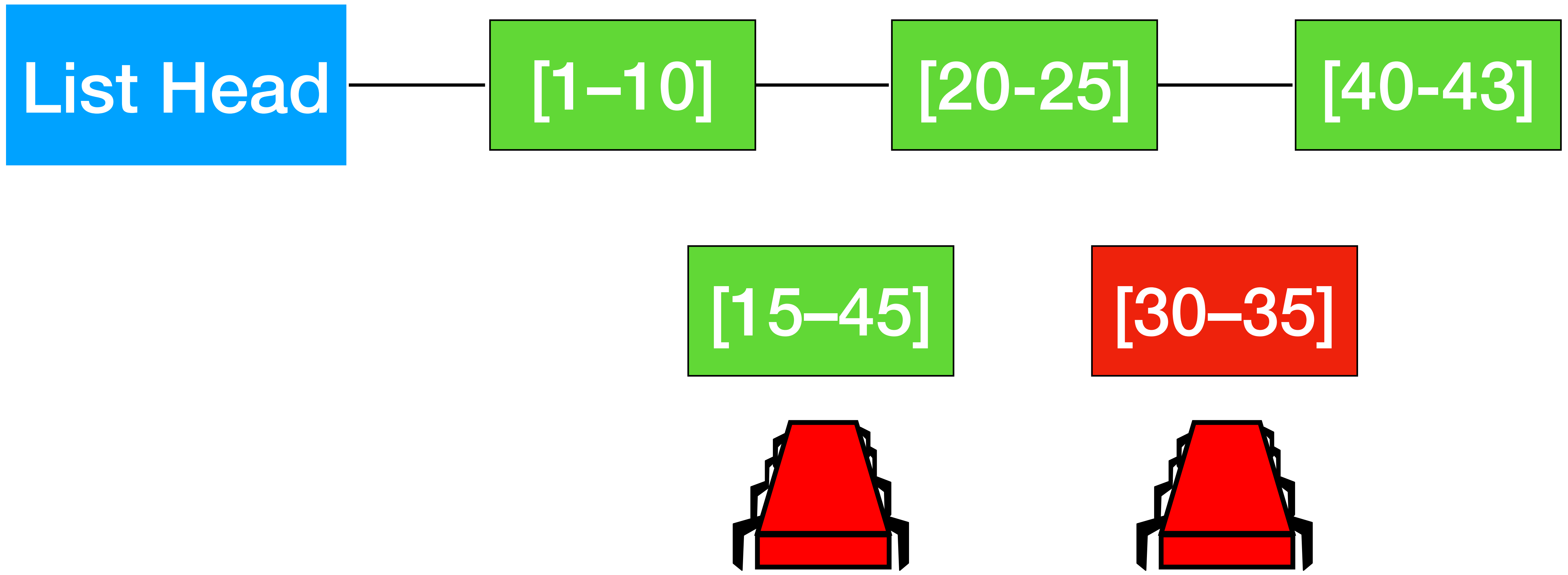
List-based Range Locks

- A range lock is acquired once a range is inserted into a list
  - Sorted by their starting addresses



List-based Range Locks

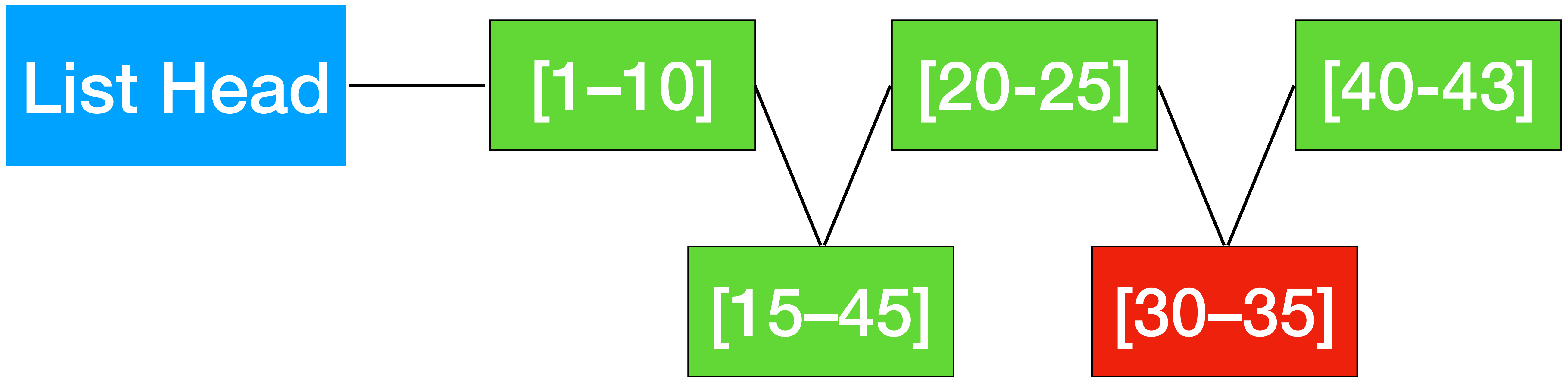
- A range lock is acquired once a range is inserted into a list
  - Sorted by their starting addresses



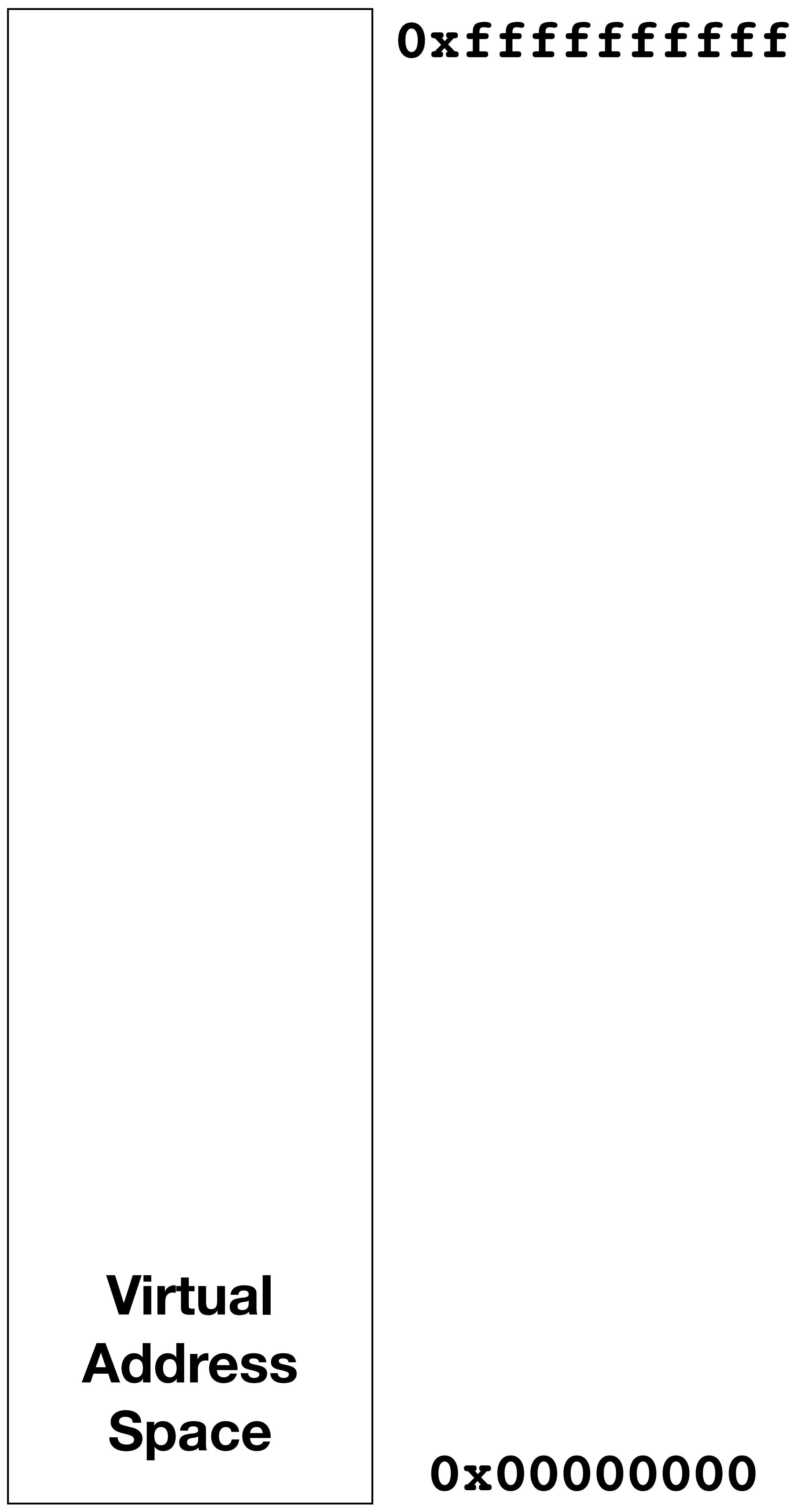
List-based Range Locks

- A range lock is acquired once a range is inserted into a list
  - Sorted by their starting addresses

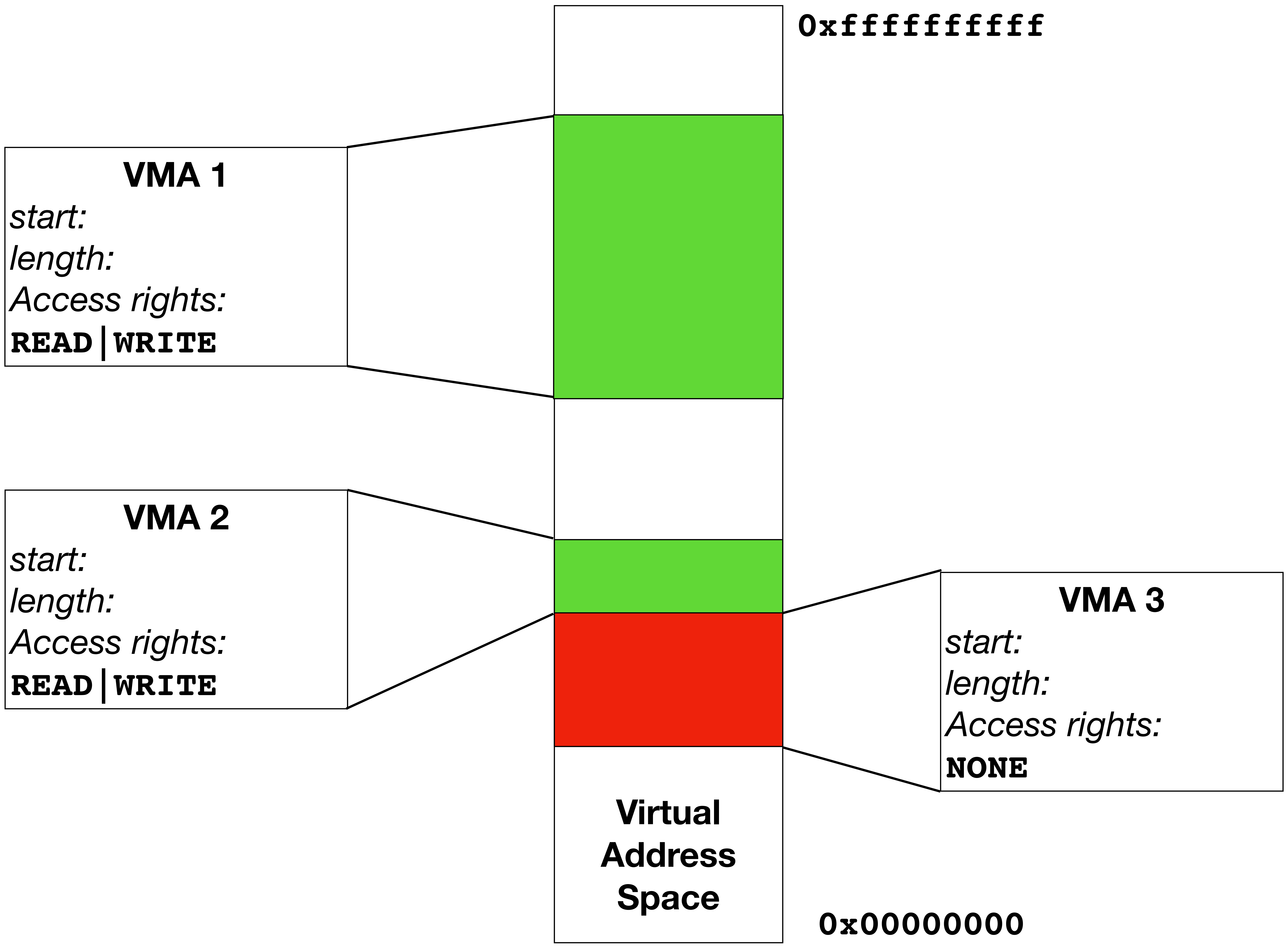
We only need an extra validation step for Read-Write semantics



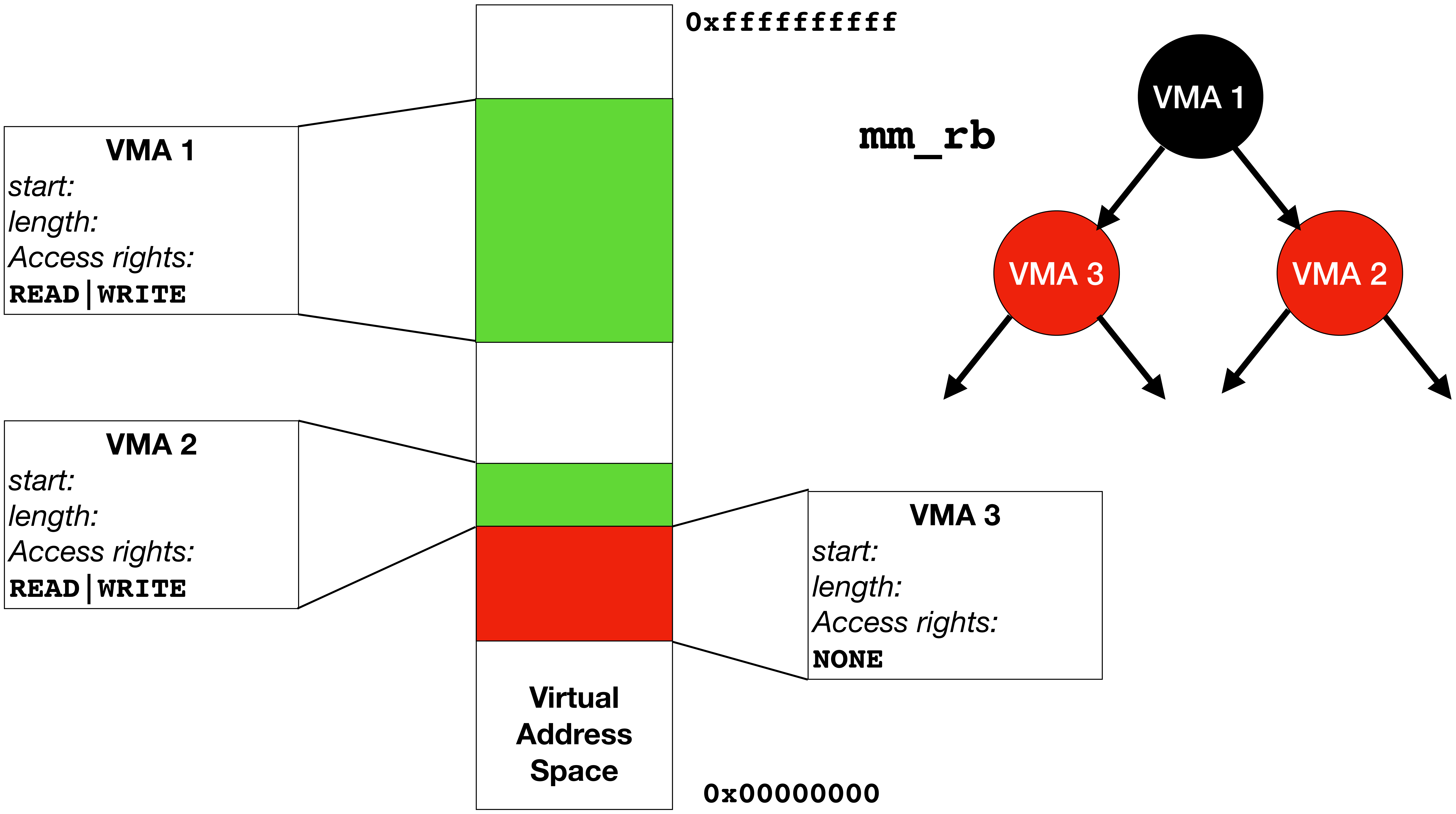
VM Management in the Kernel



VM Management in the Kernel



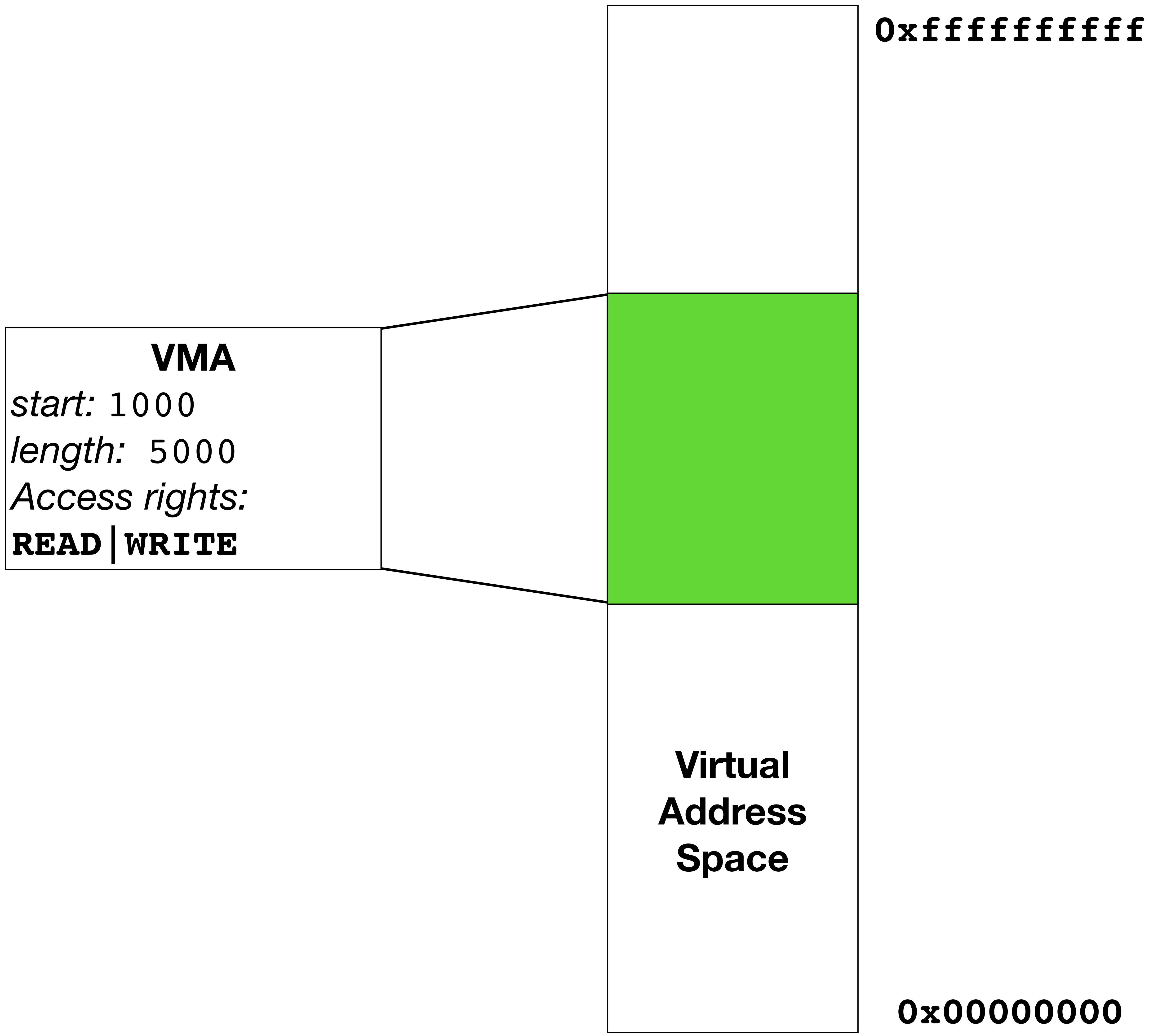
VM Management in the Kernel



VM Management in the Kernel

```
mprotect(1000, 100, READ|WRITE)
```

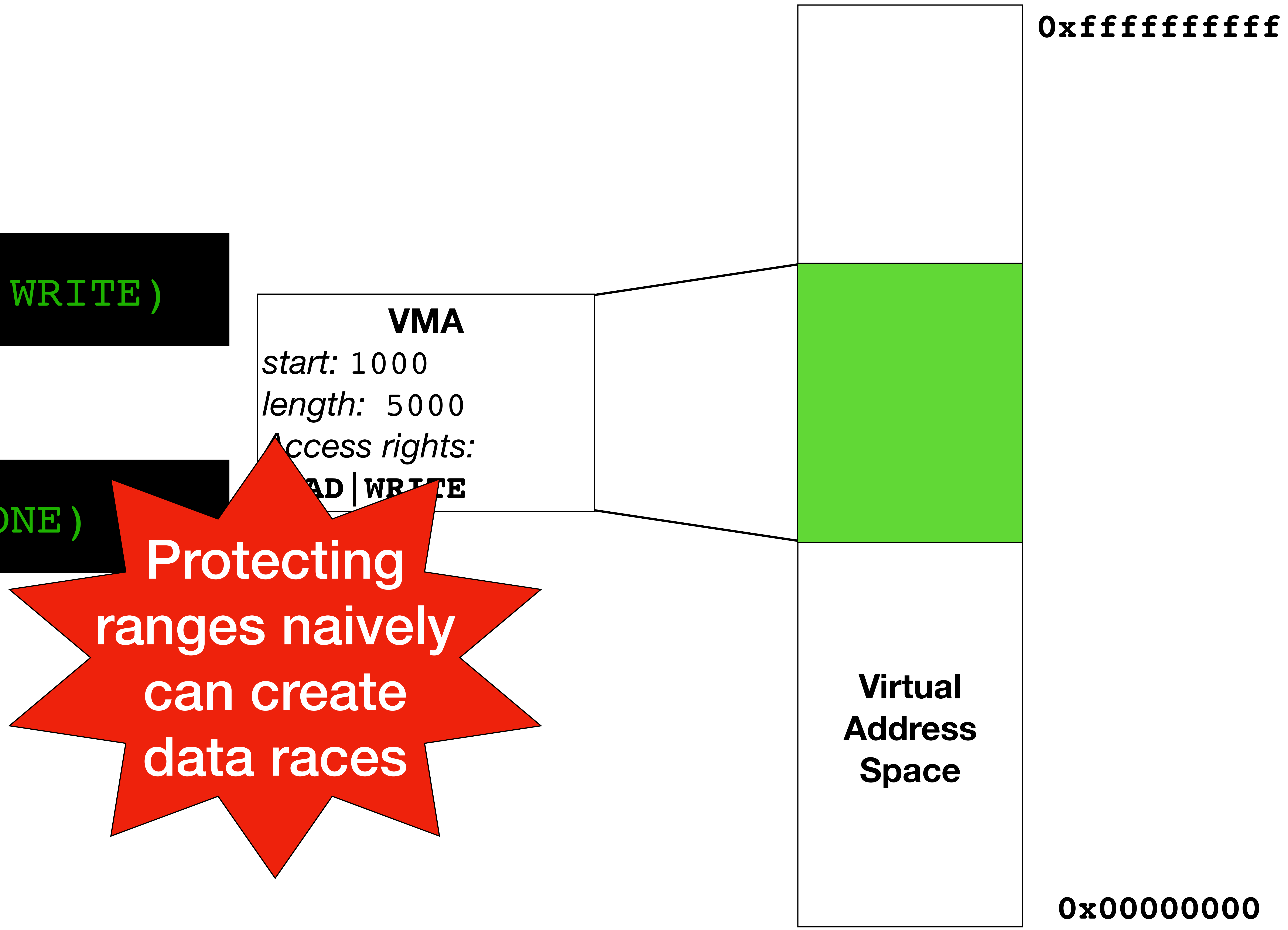
```
mprotect(3000, 100, NONE)
```



VM Management in the Kernel

```
mprotect(1000, 100, READ|WRITE)
```

```
mprotect(3000, 100, NONE)
```



Refined Ranges for VM

```
VM_Operation(start, length, args..){
  Acquire_mm_sem();
  VMA = find_vma(start);
  // operation logic
  …
  read_only operations
  Decide if structural modification is required
  …
  Release_mm_sem();
}
```

Refined Ranges for VM

```
VM_Operation(start, length, args..){
  Acquire_mm_sem();
  VMA = find_vma(start);
  // operation logic
  …
  read_only operations
  Decide if structural modification is required
  …
  Release_mm_sem();
}
```

Refined Ranges for VM

Traverses the red-black tree **mm_rb**

```
VM_Operation(start, length, args..){
  Acquire_mm_sem();
  VMA = find_vma(start);
  // operation logic
  …
  read_only operations
  Decide if structural modification is required
  …
  Release_mm_sem();
}
```

Refined Ranges for VM

Protect with range lock of input range

```
VM_Operation(start, length, args..){
  Acquire_mm_sem();
  Acquire_RL_Read(start, start+length);
  VMA = find_vma(start);
  Release_RL();
  // operation logic
  …
  read_only operations
  Decide if structural modification is required
  …
  Release_mm_sem();
}
```

Refined Ranges for VM

```
VM_Operation(start, length, args..){
  ~~Acquire_mm_sem();~~
  Acquire_RL_Read(start, start+length);
  VMA = find_vma(start);
  Release_RL();
  Acquire_RL_Write(VMA.start-x, VMA.end+x);
  // operation logic
  …
  read_only operations
  Decide if structural modification is required
  …
  Release_RL();
  ~~Release_mm_sem();~~
}
```

Protect with range lock of VMA range+Δ

check if the `mm_rb` changed meanwhile

Refined Ranges for VM

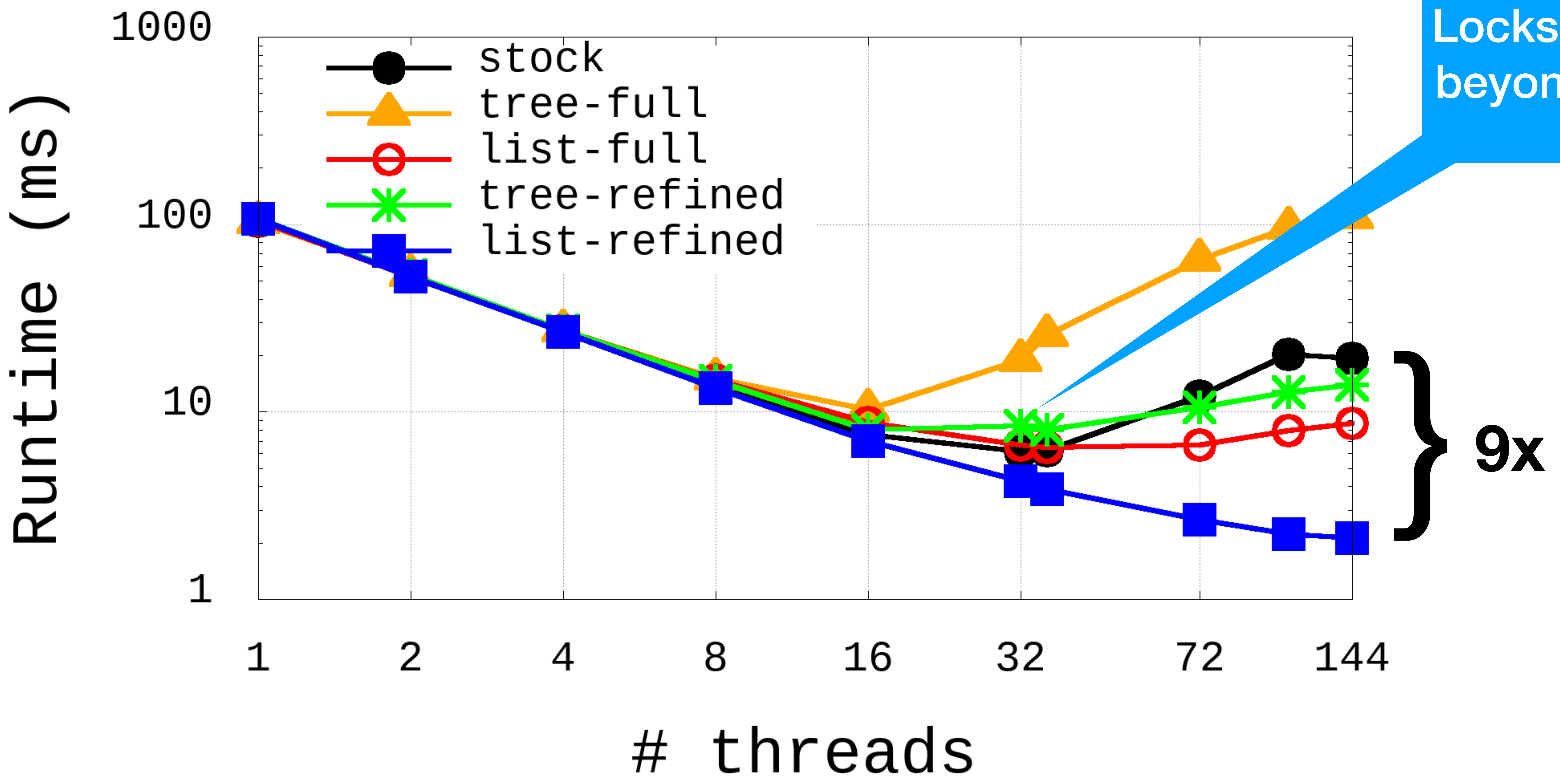
Acquire full range lock and retry

```
VM_Operation(start, length, args..){
  Acquire_mm_sem();
  Acquire_RL_Read(start, start+length);
  VMA = find_vma(start);
  Release_RL();
  Acquire_RL_Write(VMA.start-x, VMA.end)+x;
  // operation logic
  …
  read_only operations
  if structural modification is required{
      Release_RL();
      Acquire_RL_Write(0,2^63-1);
      retry();
  }
  …
  Release_RL();
  Release_mm_sem();
}
```

Evaluation

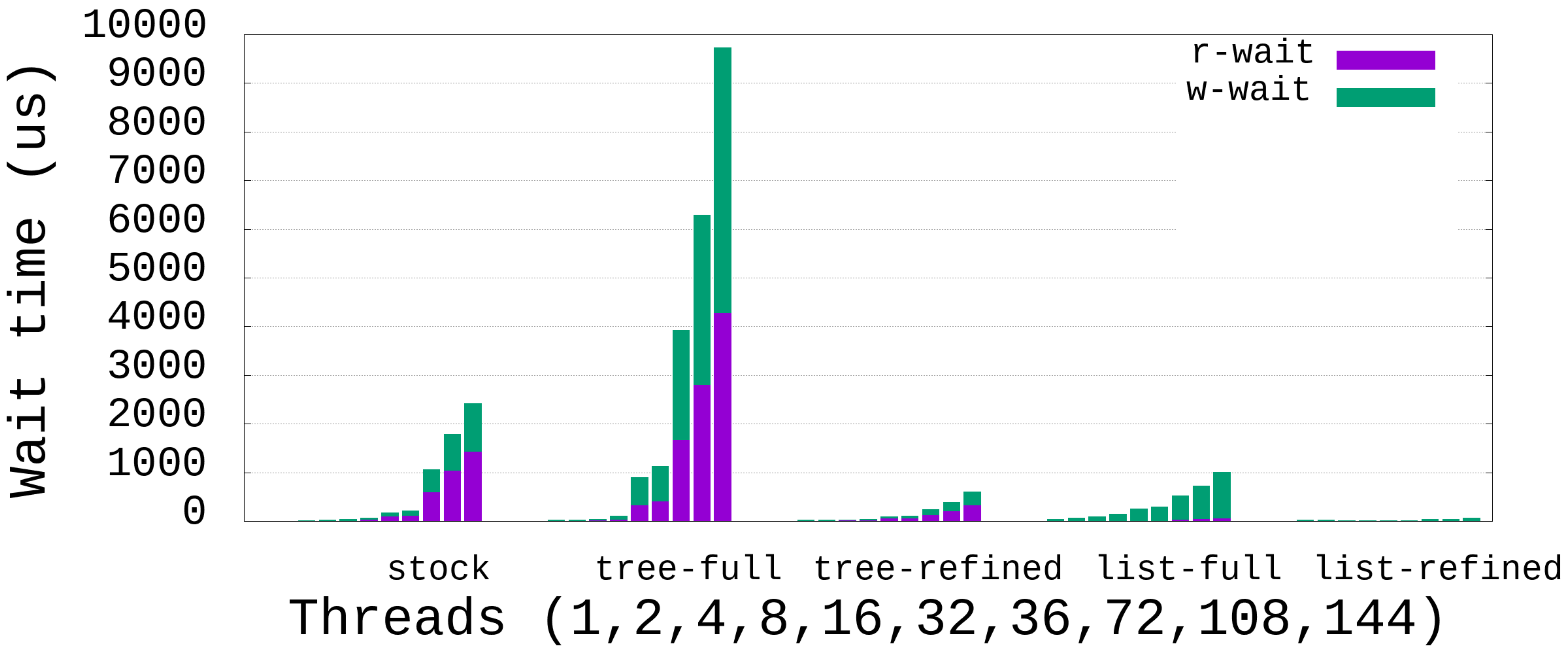
- Linux kernel 4.16.0-rc2
- 4 Intel Xeon E7-8895 v3 (144 threads)
- Metis benchmark (wrmem)
- Baselines:
    - Stock
    - Tree-based RL (with and w/out speculation)
    - List-based RL (with and w/out speculation)

Evaluation



Tree-based Range Locks do not scale beyond 32 threads

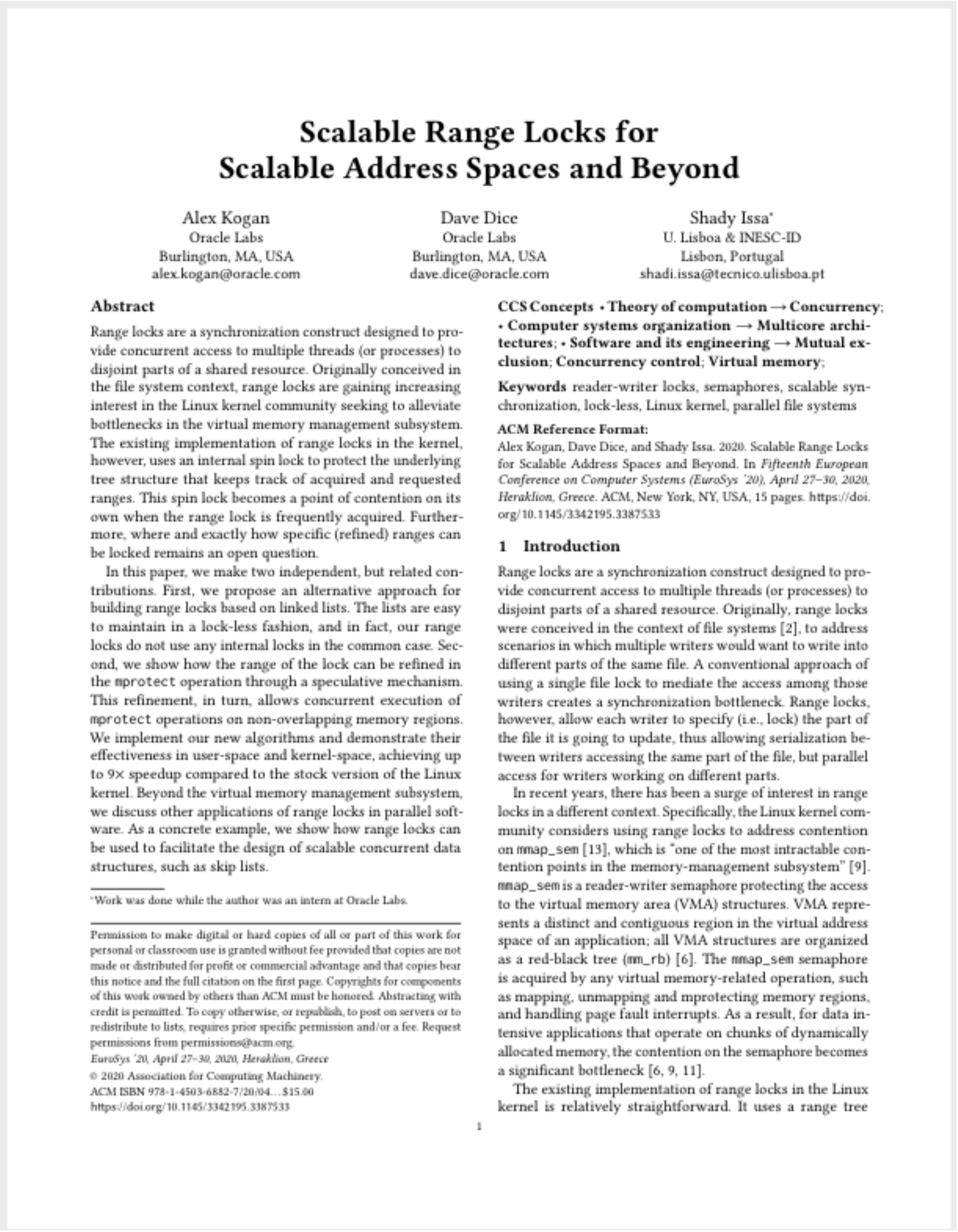
Evaluation



Collected using `lock_stats`

More in the paper…

- Evaluation:
  - More workloads
  - User-space applications
- Range Locks design
  - Fast path, avoiding starvation, memory reclamation
  - Range locks for skip lists



Scalable Range Locks for Scalable Address Spaces and Beyond

Alex Kogan
Oracle Labs
Burlington, MA, USA
alex.kogan@oracle.com

Dave Dice
Oracle Labs
Burlington, MA, USA
dave.dice@oracle.com

Shady Issa*
U. Lisboa & INESC-ID
Lisbon, Portugal
shadi.issa@tecnico.ulisboa.pt

Abstract

Range locks are a synchronization construct designed to provide concurrent access to multiple threads (or processes) to disjoint parts of a shared resource. Originally conceived in the file system context, range locks are gaining increasing interest in the Linux kernel community seeking to alleviate bottlenecks in the virtual memory management subsystem. The existing implementation of range locks in the kernel, however, uses an internal spin lock to protect the underlying tree structure that keeps track of acquired and requested ranges. This spin lock becomes a point of contention on its own when the range lock is frequently acquired. Furthermore, where and exactly how specific (refined) ranges can be locked remains an open question.

In this paper, we make two independent, but related contributions. First, we propose an alternative approach for building range locks based on linked lists. The lists are easy to maintain in a lock-less fashion, and in fact, our range locks do not use any internal locks in the common case. Second, we show how the range of the lock can be refined in the `mprotect` operation through a speculative mechanism. This refinement, in turn, allows concurrent execution of `mprotect` operations on non-overlapping memory regions. We implement our new algorithms and demonstrate their effectiveness in user-space and kernel-space, achieving up to 9× speedup compared to the stock version of the Linux kernel. Beyond the virtual memory management subsystem, we discuss other applications of range locks in parallel software. As a concrete example, we show how range locks can be used to facilitate the design of scalable concurrent data structures, such as skip lists.

*Work was done while the author was an intern at Oracle Labs.

EuroSys '20, April 27–30, 2020, Heraklion, Greece

ACM ISBN 978-1-4503-6882-7/20/04…$15.00
https://doi.org/10.1145/3342195.3387533

CCS Concepts • **Theory of computation** → **Concurrency**; • **Computer systems organization** → **Multicore architectures**; • **Software and its engineering** → **Mutual exclusion**; **Concurrency control**; **Virtual memory**;

Keywords reader-writer locks, semaphores, scalable synchronization, lock-less, Linux kernel, parallel file systems

ACM Reference Format:
Alex Kogan, Dave Dice, and Shady Issa. 2020. Scalable Range Locks for Scalable Address Spaces and Beyond. In *Fifteenth European Conference on Computer Systems (EuroSys '20), April 27–30, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3342195.3387533

1 Introduction

Range locks are a synchronization construct designed to provide concurrent access to multiple threads (or processes) to disjoint parts of a shared resource. Originally, range locks were conceived in the context of file systems [2], to address scenarios in which multiple writers would want to write into different parts of the same file. A conventional approach of using a single file lock to mediate the access among those writers creates a synchronization bottleneck. Range locks, however, allow each writer to specify (i.e., lock) the part of the file it is going to update, thus allowing serialization between writers accessing the same part of the file, but parallel access for writers working on different parts.

In recent years, there has been a surge of interest in range locks in a different context. Specifically, the Linux kernel community considers using range locks to address contention on `mmap_sem` [13], which is "one of the most intractable contention points in the memory-management subsystem" [9]. `mmap_sem` is a reader-writer semaphore protecting the access to the virtual memory area (VMA) structures. VMA represents a distinct and contiguous region in the virtual address space of an application; all VMA structures are organized as a red-black tree (`mm_rb`) [6]. The `mmap_sem` semaphore is acquired by any virtual memory-related operation, such as mapping, unmapping and mprotecting memory regions, and handling page fault interrupts. As a result, for data intensive applications that operate on chunks of dynamically allocated memory, the contention on the semaphore becomes a significant bottleneck [6, 9, 11].

The existing implementation of range locks in the Linux kernel is relatively straightforward. It uses a range tree

1

Conclusion

- Scalable linked list-based Range Locks
- New speculative approach for the Linux VM
- Using Range Locks for concurrent data structures