Compact NUMA-aware Locks*

Alex Kogan
Oracle Labs
alex.kogan@oracle.com

joint work with Dave Dice (Oracle Labs)

* patch series "**Add NUMA-awareness to qspinlock**" (https://lwn.net/Articles/856387)

Locks: Quick Background

Protect access to the shared data

Remain the most popular synchronization technique
… and the topic of extensive research

Performance of parallel software often depends on the efficiency of the locks it employs

Locks: Quick Background

Many flavors:

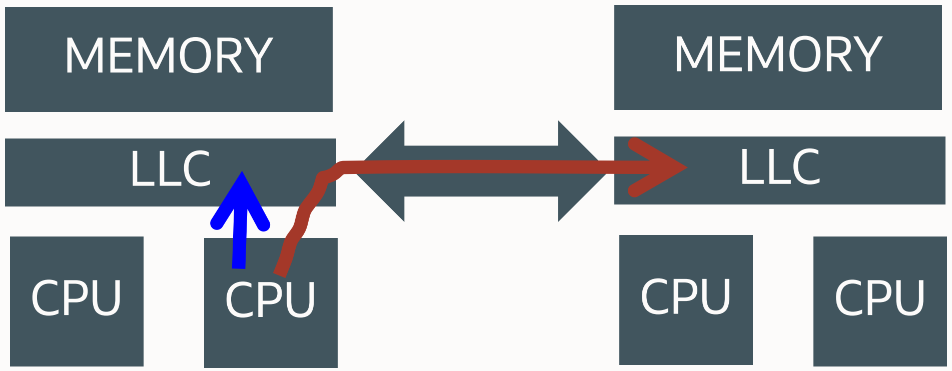
- exclusive / reader-writer
- spinning / blocking
- strictly fair / unfair / long-term fair
- …

The focus of this talk: exclusive, fair, spinning lock (aka **qspinlock**)

Evolve with the evolution of computing architectures

- we live in the era of multi-socket architectures with NUMA effects → we need *NUMA-aware* locks

NUMA-aware Locks



Access by a core to a local memory or local cache is faster that accesses to a remote memory or remote cache

- known as Non-Uniform Memory Access (NUMA) effect

Keep the lock ownership ***within the same node***

- decrease remote cache misses and inter-node communication
  - for lock state access as well as data accessed in the critical section
- non-FIFO and unfair over the short term
  - but usually preserve fairness over the longer term

➢ trade-off short-term fairness for better performance

qspinlock in the Kernel

Certain critical requirements

- compact
  - must occupy at most 4 bytes
- fair (strictly fair/FIFO ?)
- perform well under both low and high contention

Keeps evolving

- test-set → ticket → MCS (slow path + fast path test-set)

Still **not** NUMA-aware!

- existing NUMA-aware locks tend to use space proportional to #nodes
  - 100s bytes on a typical multi-node system

CNA: Compact NUMA-aware Lock

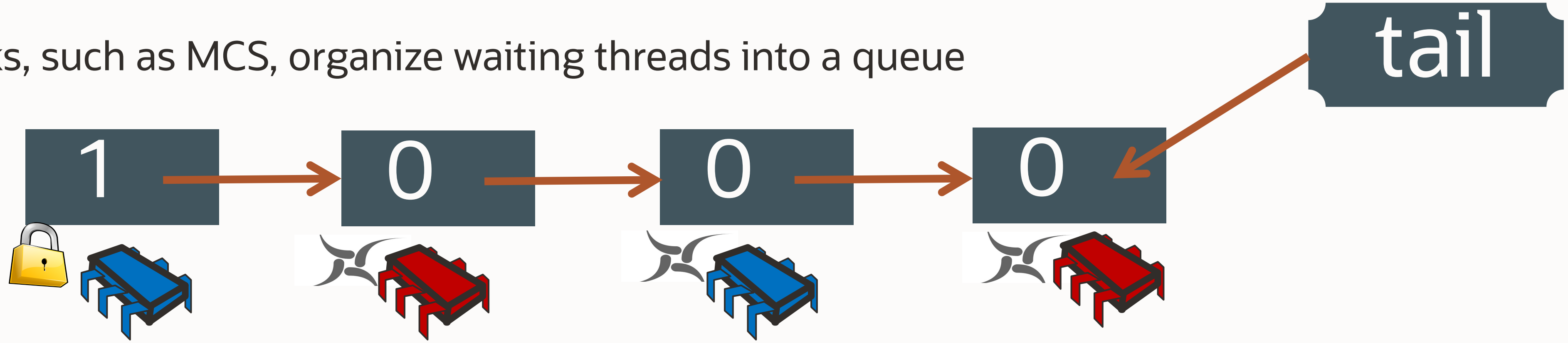
- ✓ Requires 4 bytes of memory
  - like existing qspinlock
  - or just one word (pointer) when implemented in user-space

- ✓ Variant of a (NUMA-oblivious) MCS lock
  - inherits its performance features
    - local spinning, one atomic operation per acquisition, …
  - requires minor changes to existing MCS implementations
    - including qspinlock

- ✓ Performance on-par with MCS under no contention, on-par with state-of-the-art hierarchical NUMA-aware locks when contended
  - up to ~3x throughput increase on a highly contended (4 node) system

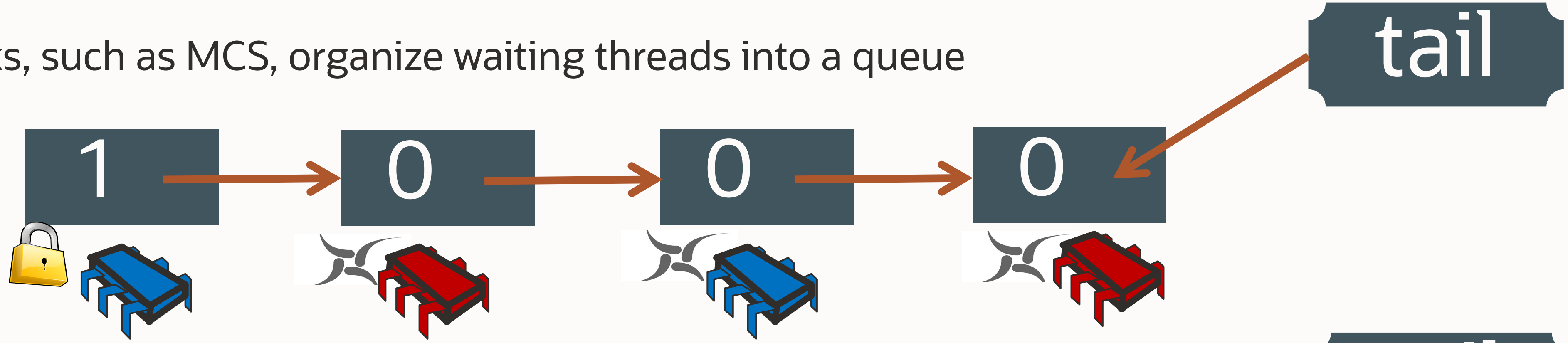
How Does CNA Do That?
(Or: What is the Trick?)

Queue-based spin locks, such as MCS, organize waiting threads into a queue



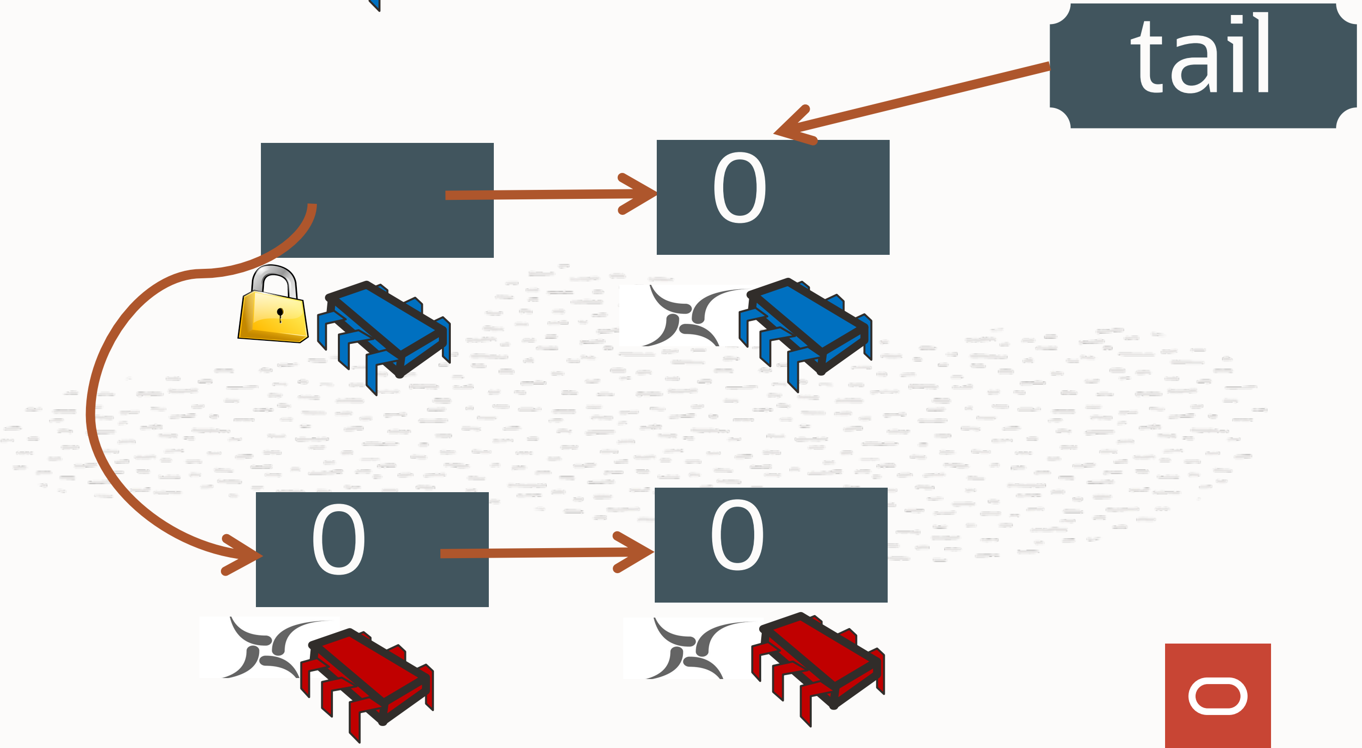
How Does CNA Do That? (Or: What is the Trick?)

Queue-based spin locks, such as MCS, organize waiting threads into a queue



CNA uses **two queues**:

- **primary**: threads running on the same node as the lock holder
- **secondary**: everyone else



How Does CNA Do That?
(Or: What is the Trick?)

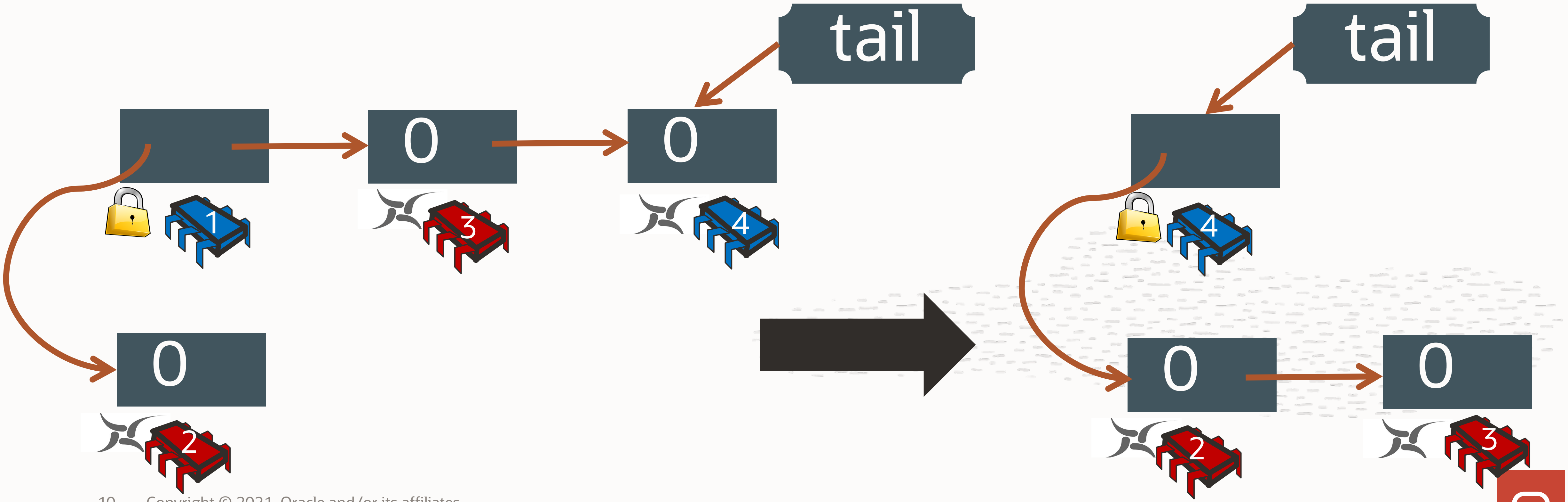
MCS lock holder checks whether the next waiter in the primary queue is running on the same NUMA node

- if not, it detaches that waiter from the primary queue and moves it to the tail of the secondary one

How Does CNA Do That?
(Or: What is the Trick?)

MCS lock holder checks whether the next waiter in the primary queue is running on the same NUMA node

- if not, it detaches that waiter from the primary queue and moves it to the tail of the secondary one

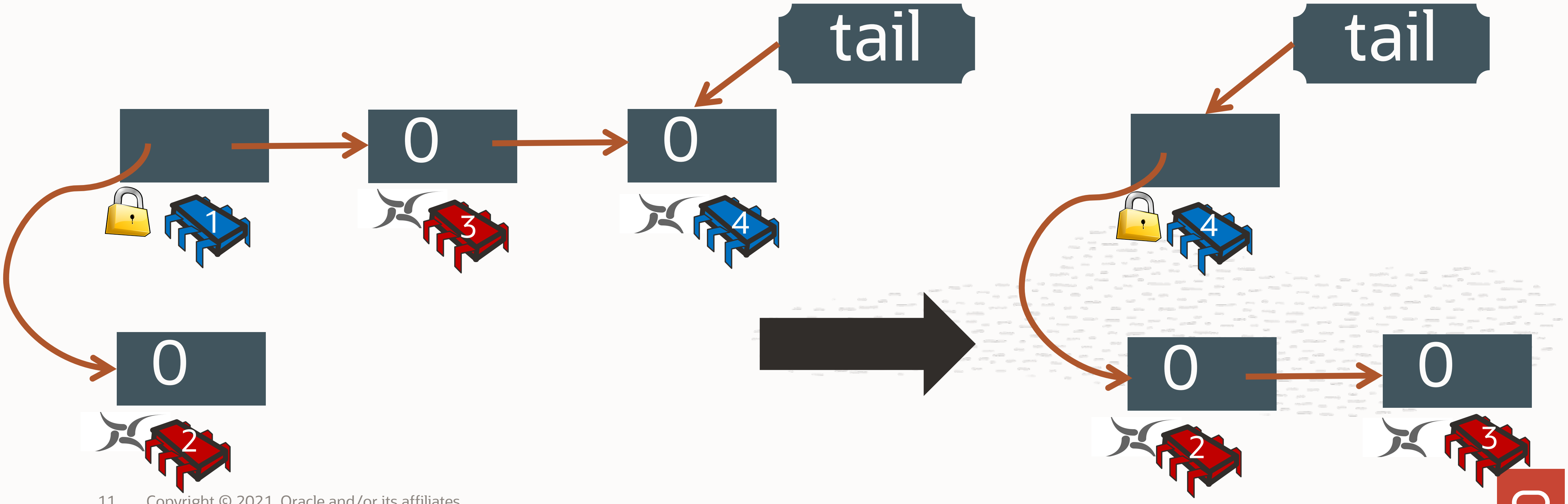


How Does CNA Do That?
(Or: What is the Trick?)

MCS lock holder checks whether the next waiter in the primary queue is running on the same NUMA node

- if not, it detaches that waiter from the primary queue and moves it to the tail of the secondary one

➢ gradually filter the primary queue, leaving only waiters running on the same (“preferred”) NUMA node



Avoiding Starvation
(Or: What about Fairness / FIFO?)

To ensure long-term fairness, flush the secondary queue back into the primary one after a *certain period of time (or number)* of “intra-node” handovers

After certain time has passed since the first thread has been moved to the secondary queue

- How much time?
- Tunable parameter, default value – 1ms
- Can be tweaked on the fly (`module_param()`)

CNA trades FIFO / short-term fairness for better performance

Performance Evaluation

Kernel-space:

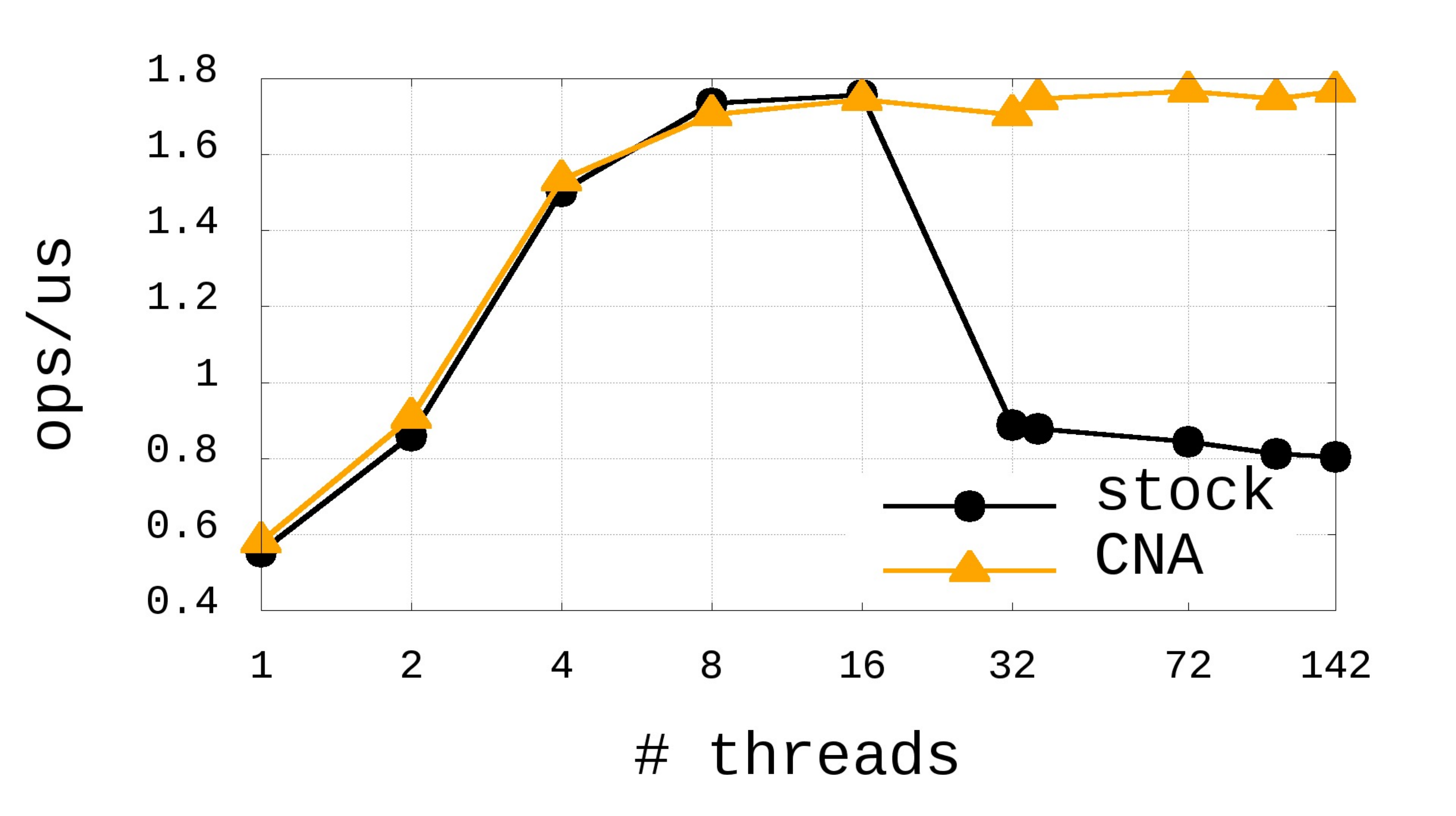
- Integrated into the slow path of qspinlock

User-space:

- Implemented CNA as a user-level library
- Compared to MCS, SOTA (hierarchical) NUMA-aware locks (cohort lock C-BO-MCS & HMCS lock)

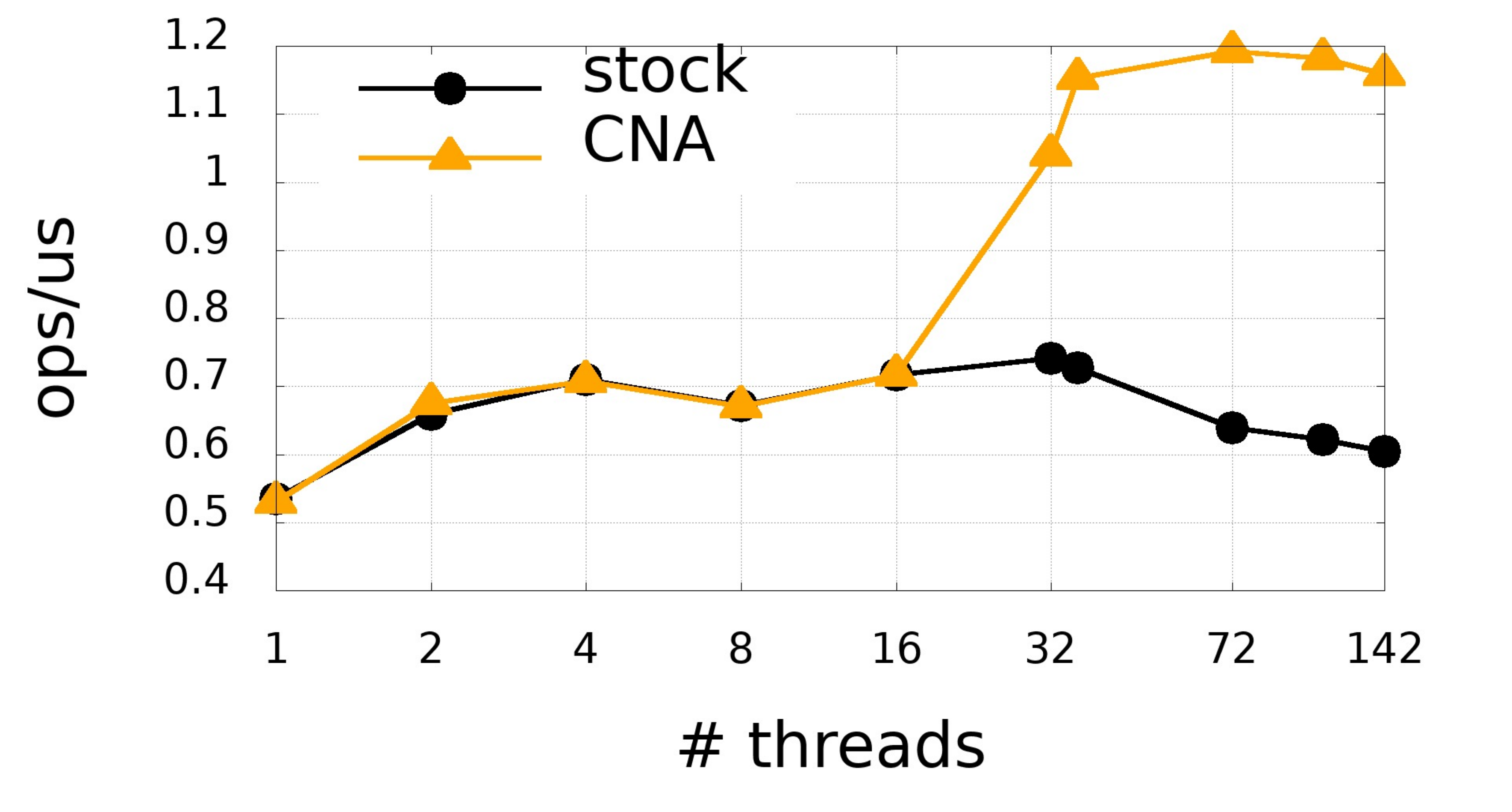
HW:
4-socket x86 system (Intel Xeon E7-8895 v3 @ 2.60GHz), with 18 hyper-threaded cores per sockets

will-it-scale/open1_threads



LevelDB/readrandom

CNA accelerates contended user-land pthread locks by increasing throughput over the futex chains



More results

In the patch description and on the MLs, e.g.:

https://lists.01.org/hyperkitty/list/lkp@lists.01.org/thread/HGVOCYDEE5KTLYPTAFBD2RXDQOCDPFUJ/
[locking/qspinlock] 0e8d8f4f12: fsmark.files_per_sec 213.9% improvement

https://lists.01.org/hyperkitty/list/lkp@lists.01.org/thread/OUPS7MZ3GJA2XYWM52GMU7H7EI25IT37/
[locking/qspinlock] 0dd6d5b8c0: vm-scalability.throughput 102.9% improvement

https://lists.01.org/hyperkitty/list/lkp@lists.01.org/thread/DNMEQPXJRQY2IKHZ3ERGRY6TUPWDTFUN/
[locking/qspinlock] 372cdd28b7: aim7.jobs-per-min 76.7% improvement

Summary

CNA reduces remote cache misses while preserving long-term fairness

CNA achieves the best of both worlds:

- ✓ as efficient as MCS at low contention
  - but better at high contention by 40-200%
- ✓ as performant as state-of-the-art NUMA-aware locks at high contention
  - but its state requires only four bytes of memory

Kernel patch “**Add NUMA-awareness to qspinlock”** at https://lwn.net/Articles/856387

Patch Status

15 rounds of revisions

- big **thank you** to everyone who provided feedback, evaluated, etc.
- more feedback / evaluation results are welcome!

Do we really need this?

"Shouldn't we be spending our time breaking [contended] locks [instead]?"

Probably. If you can rewrite your software and avoid lock contention, do so! But

- more efficient locks help us to "buy time" for rewrite
  - sometimes, rewrite is not really an option (e.g., legacy software)
- some locks are inherently contended
  - a "hot" file accessed by many clients concurrently
- by ignoring NUMA, we leave up to ~3x performance on the table

Resources

"Compact NUMA-aware Locks" at ACM Eurosys'19: https://dl.acm.org/doi/10.1145/3302424.3303984

- Also available on arxiv: https://arxiv.org/abs/1810.05600

CNA patch (latest revision): https://lwn.net/Articles/856387

LWN article: https://lwn.net/Articles/852138

Some performance reports from kernel test robot:
https://lists.01.org/hyperkitty/list/lkp@lists.01.org/thread/HGVOCYDEE5KTLYPTAFBD2RXDQOCDPFUJ/
https://lists.01.org/hyperkitty/list/lkp@lists.01.org/thread/OUPS7MZ3GJA2XYWM52GMU7H7EI25IT37/
https://lists.01.org/hyperkitty/list/lkp@lists.01.org/thread/DNMEQPXJRQY2IKHZ3ERGRY6TUPWDTFUN/

Thank you!
Questions?
alex.kogan@oracle.com