



NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina

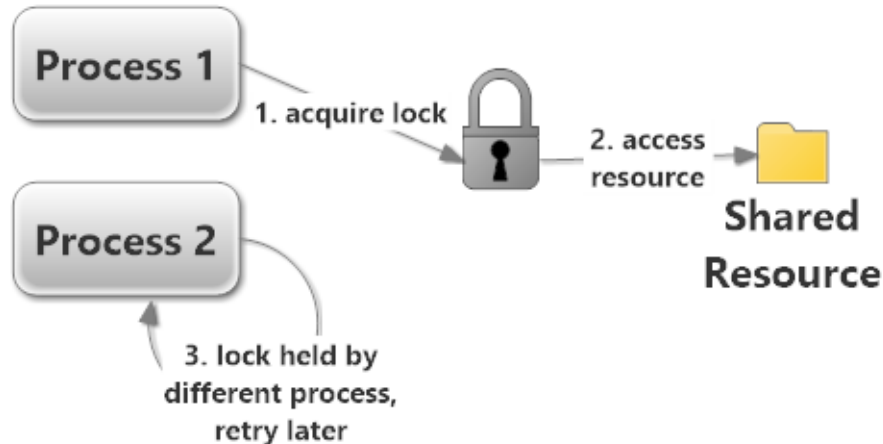
04.02.2015

NUMA Seminar

1. Recap: Locking
2. Locks in NUMA Systems
3. NUMA-aware RW Locks
4. Implementations
5. Hands On

Why Locking?

- Parallel tasks access shared resources
- Locks: Synchronization mechanism in concurrent environments
- Locks ensure mutual exclusion for critical section
- Preventing race conditions



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 3

Example: Race Condition

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Ideal execution leads
to **correct** result

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart 4

Example: Race Condition

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

No synchronization
may lead to **wrong**
result

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **5**

Example: Race Condition

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Critical section
requires mutual
exclusion

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart 6

Starvation

A thread never gets to execute critical section

Fairness

Some threads acquire lock more often than others

Deadlock

Two threads wait for each other to release a lock

Live Lock

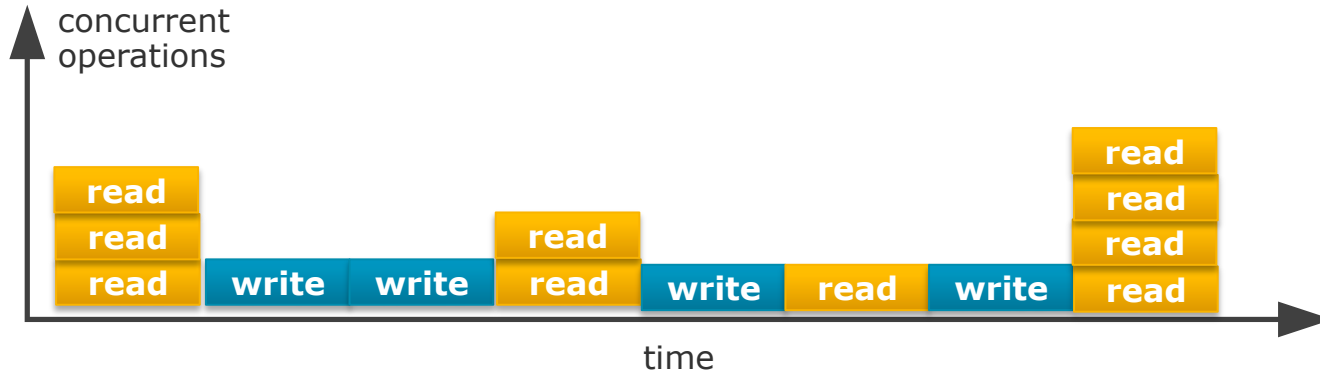
Two threads activate each other in an infinite loop

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart 7

Reader-Writer Locks

- Mutual exclusion can be too much!
- Allow concurrent read access
- Require exclusive access for write operations



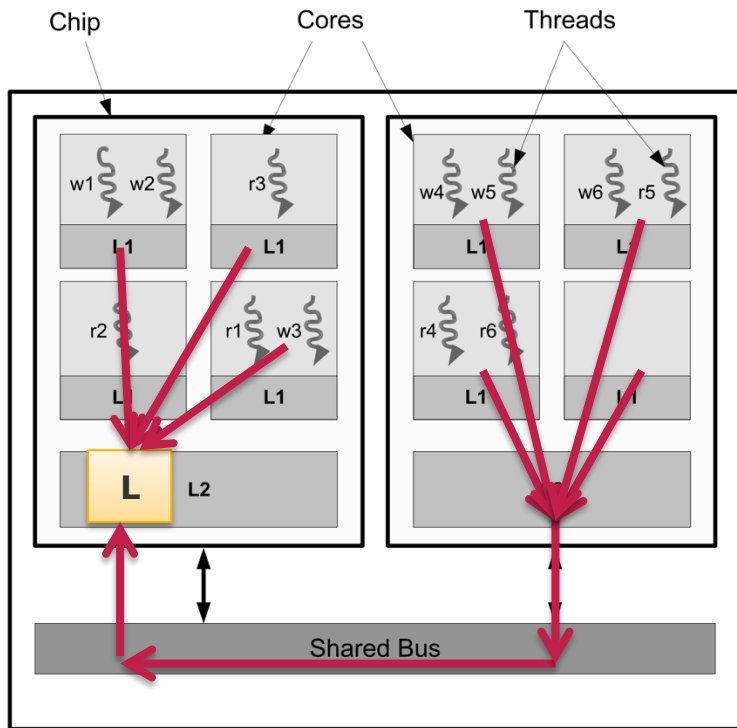
NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 8

A large, horizontal rectangular box with a gradient from yellow to orange, serving as a background for the title. It is framed by a dark red border on the left and bottom sides.

Locks in NUMA Systems

Locks on NUMA Systems



- Intra-node traffic is cheap
- Communication on interconnect is expensive!
- Where to put the lock?
- Problem: Cache coherency protocols lead to **lock migrations**

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 10

Cohort Detection

Lock owner can determine whether there are additional threads waiting to acquire the lock

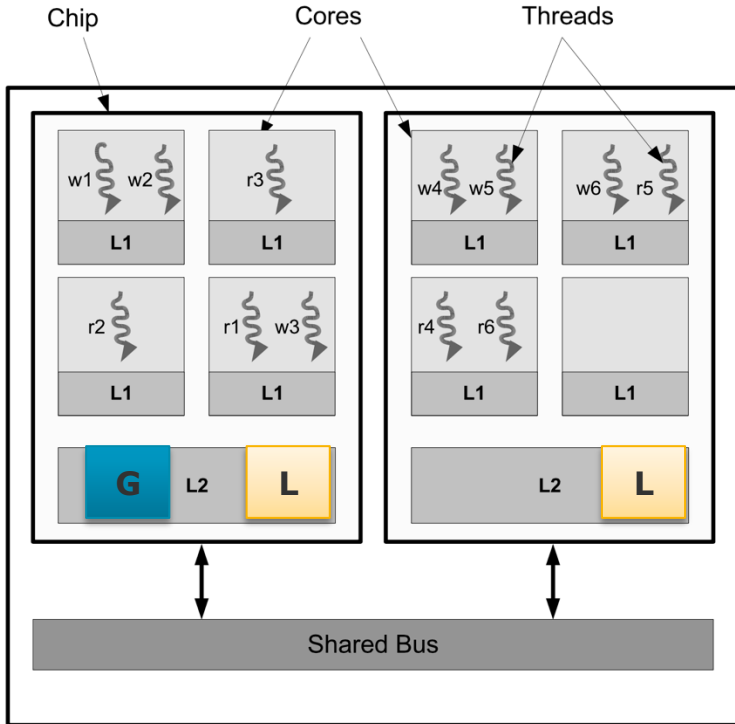
Thread Obliviousness

The lock can be acquired by one thread and released by any other thread

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **11**

Avoiding Lock Migrations: Cohort Locking



- Two locks:
 - **G** → global (thread-oblivious)
 - **L** → node-local

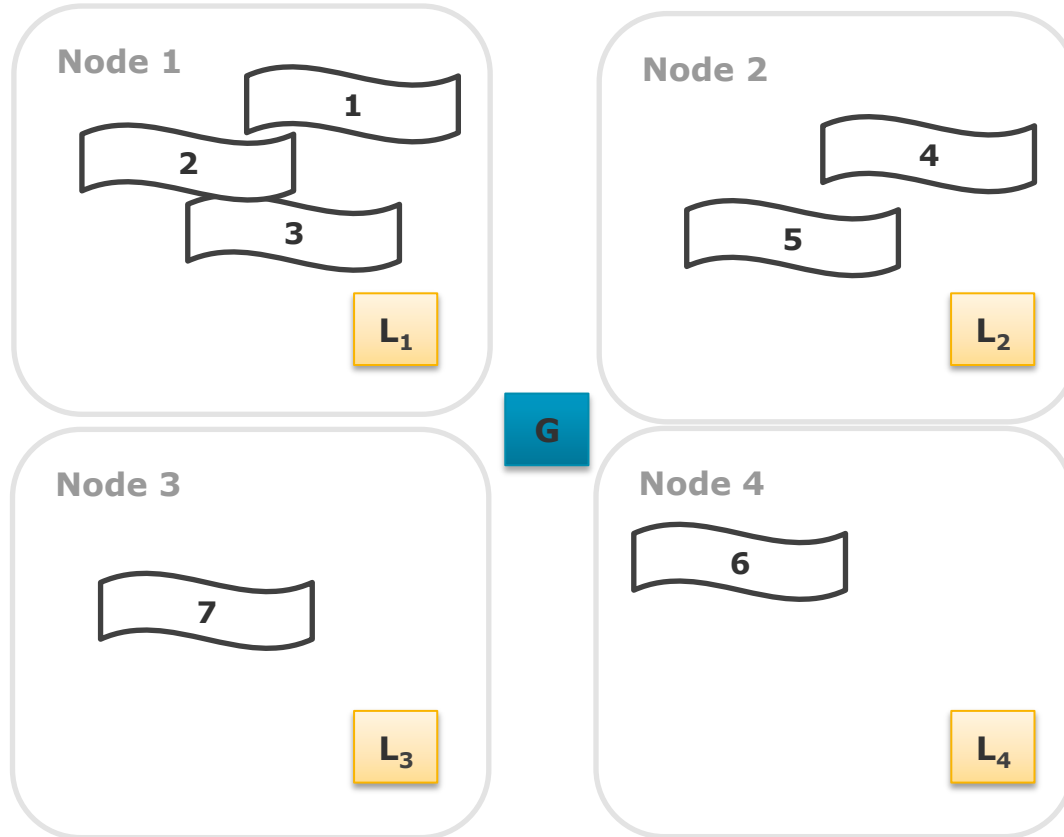
```
acquireCohortLock() {  
    L.acquire()  
    G.acquire()  
}
```

```
releaseCohortLock() {  
    L.release()  
    if(!L.hasWaitingThreads())  
        G.release()  
}
```

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart 12

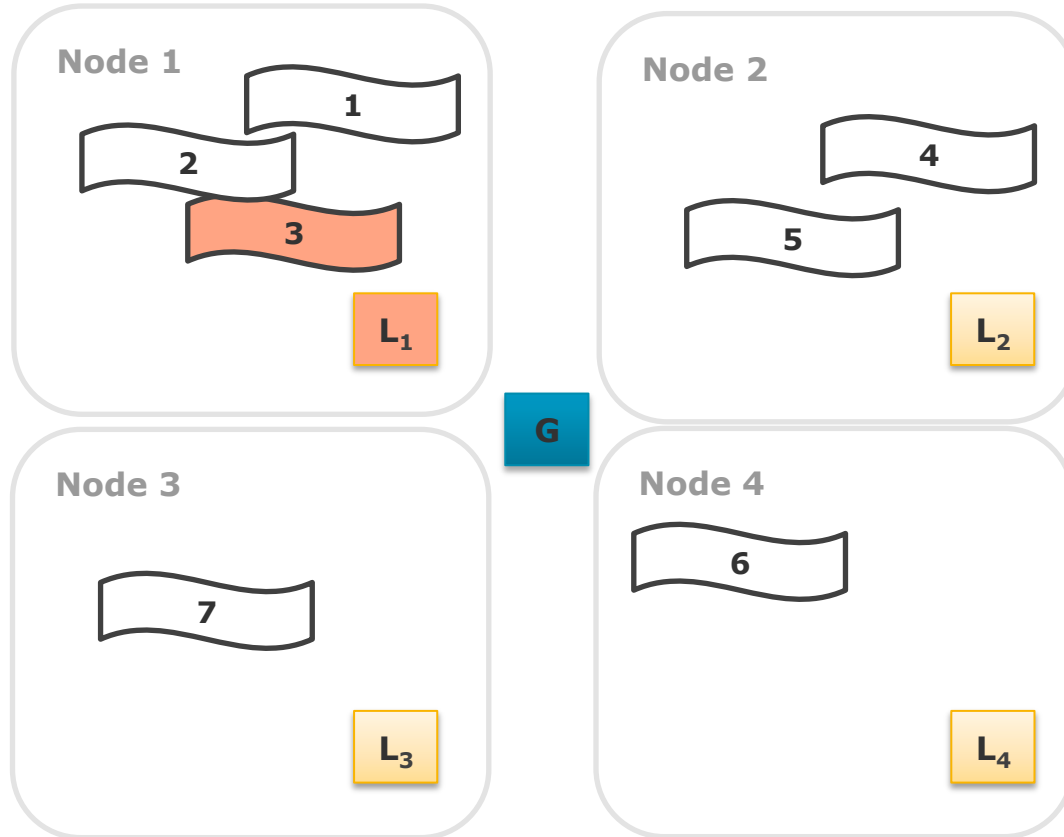
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 13

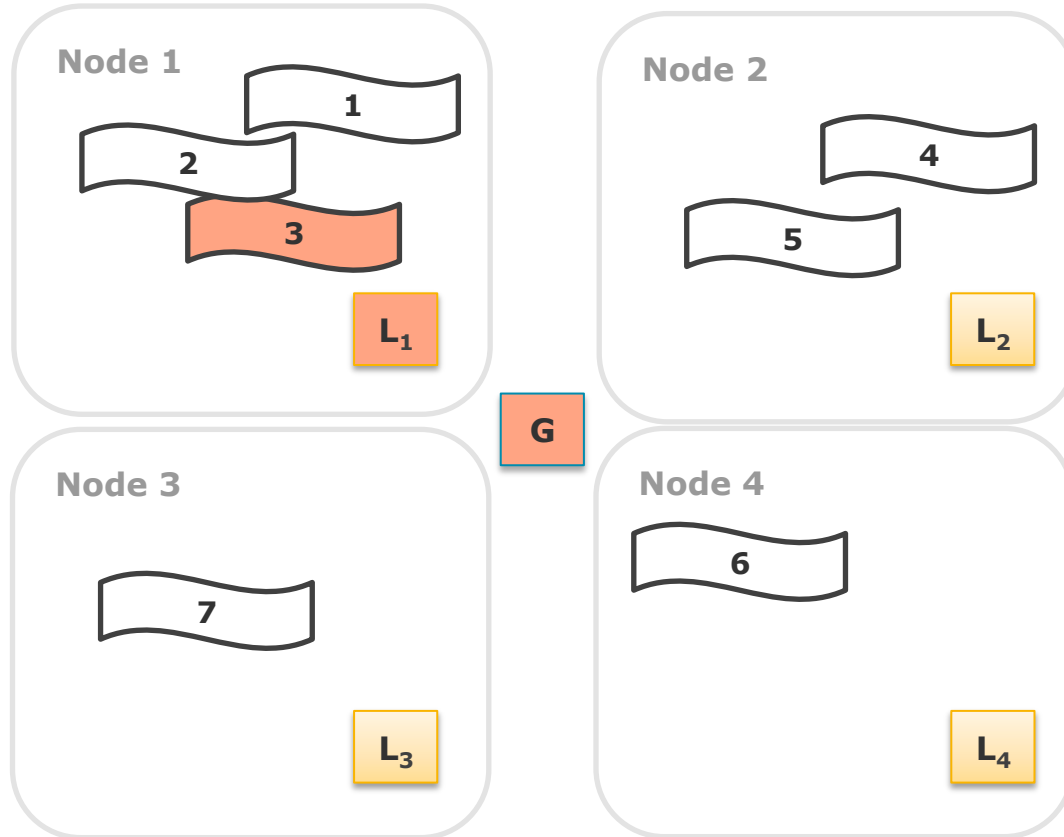
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 14

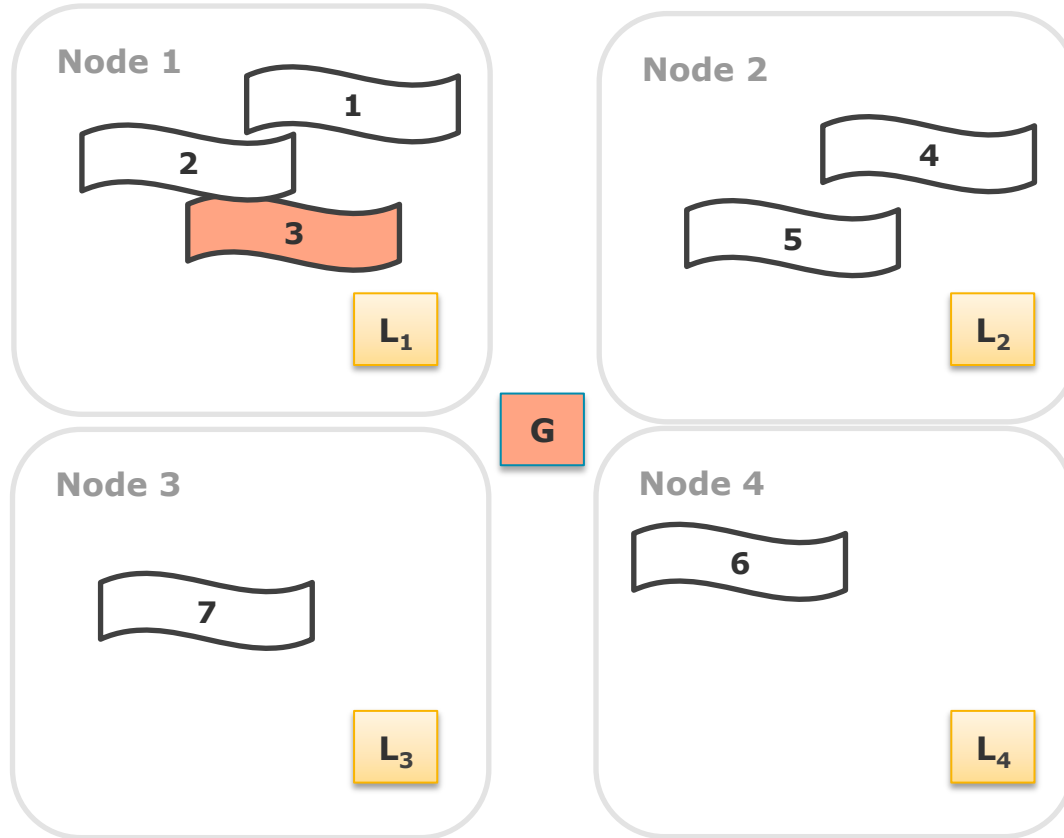
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 15

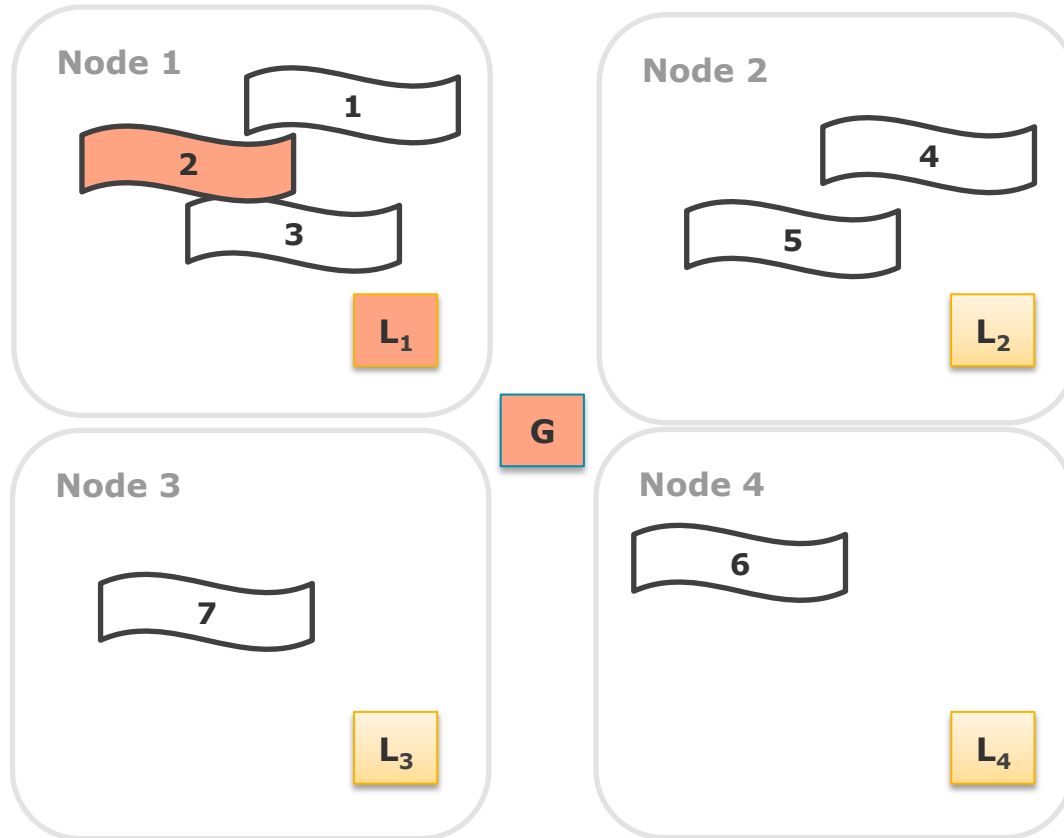
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 16

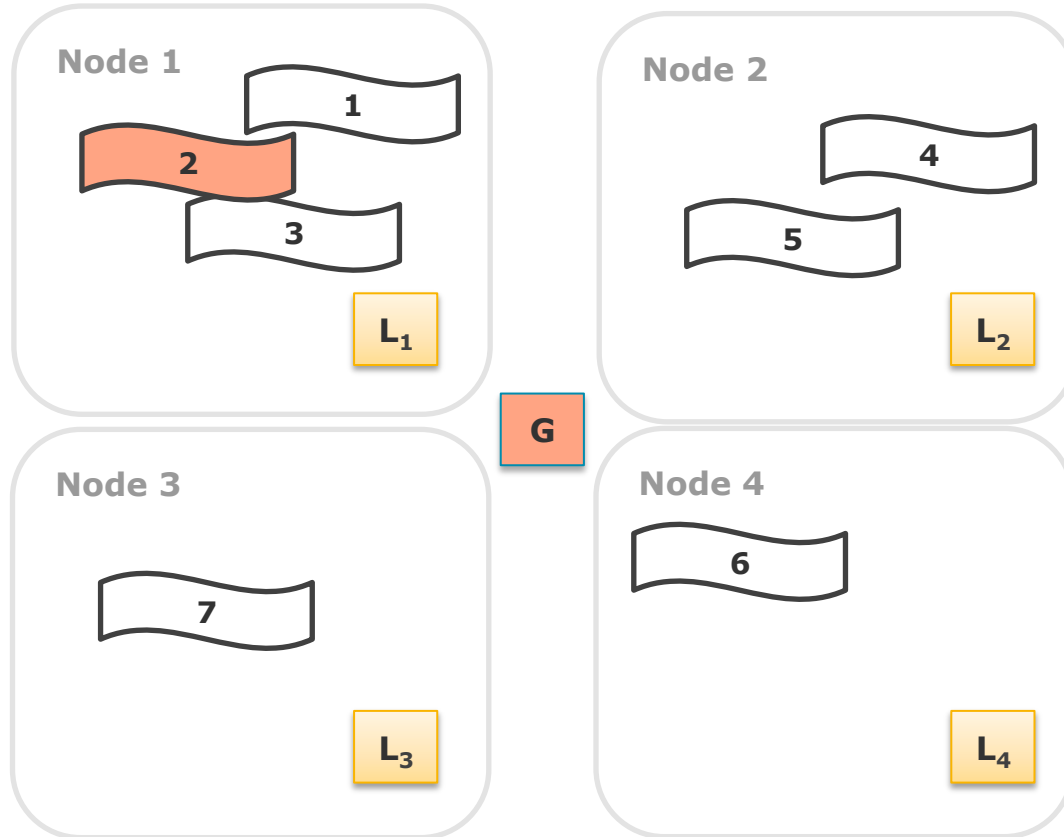
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 17

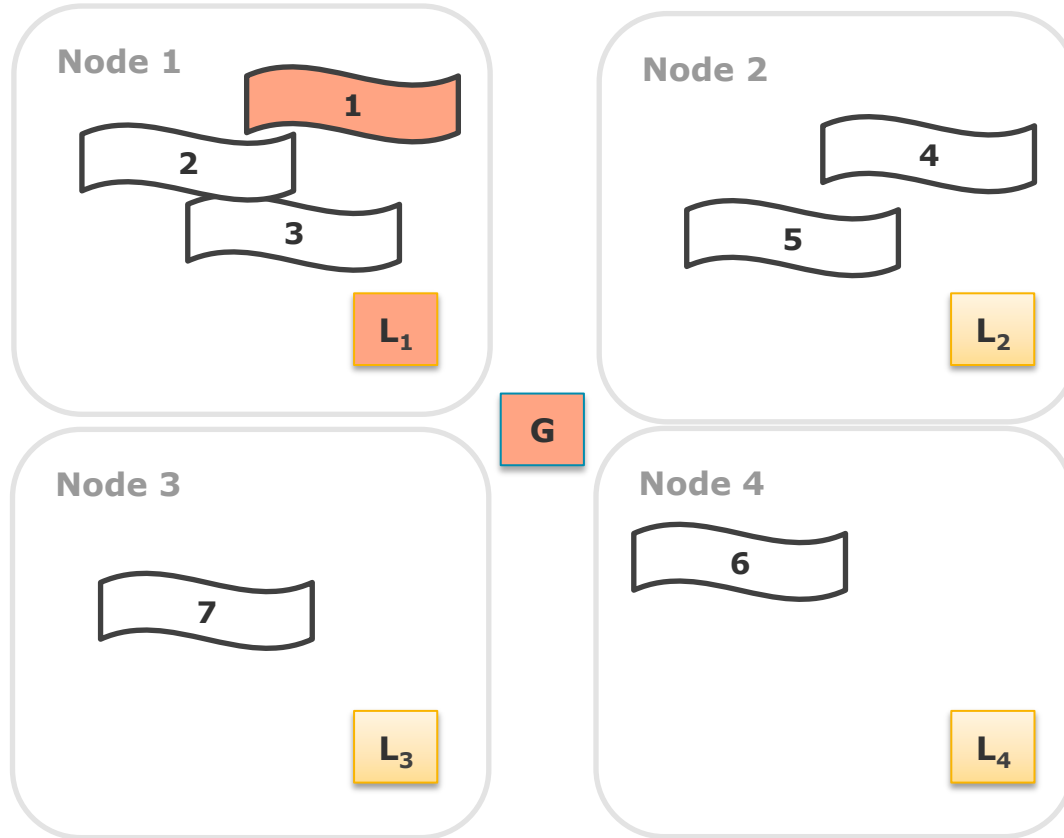
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 18

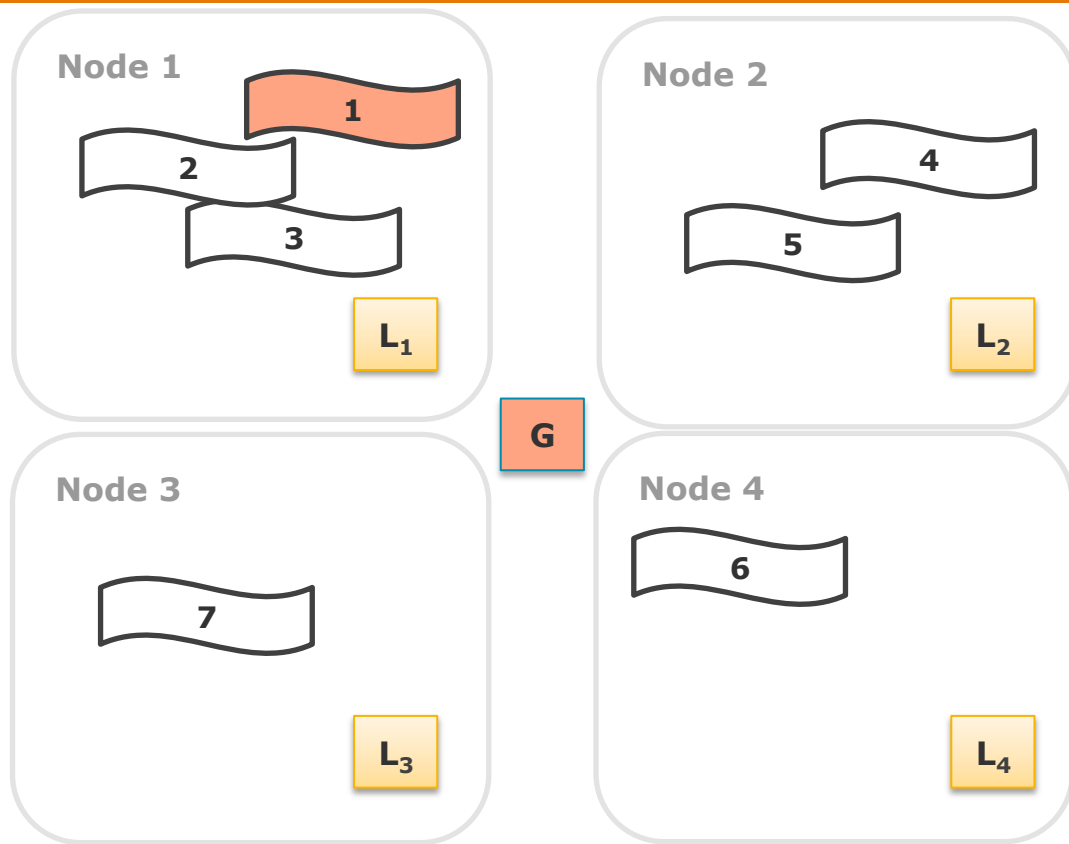
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 19

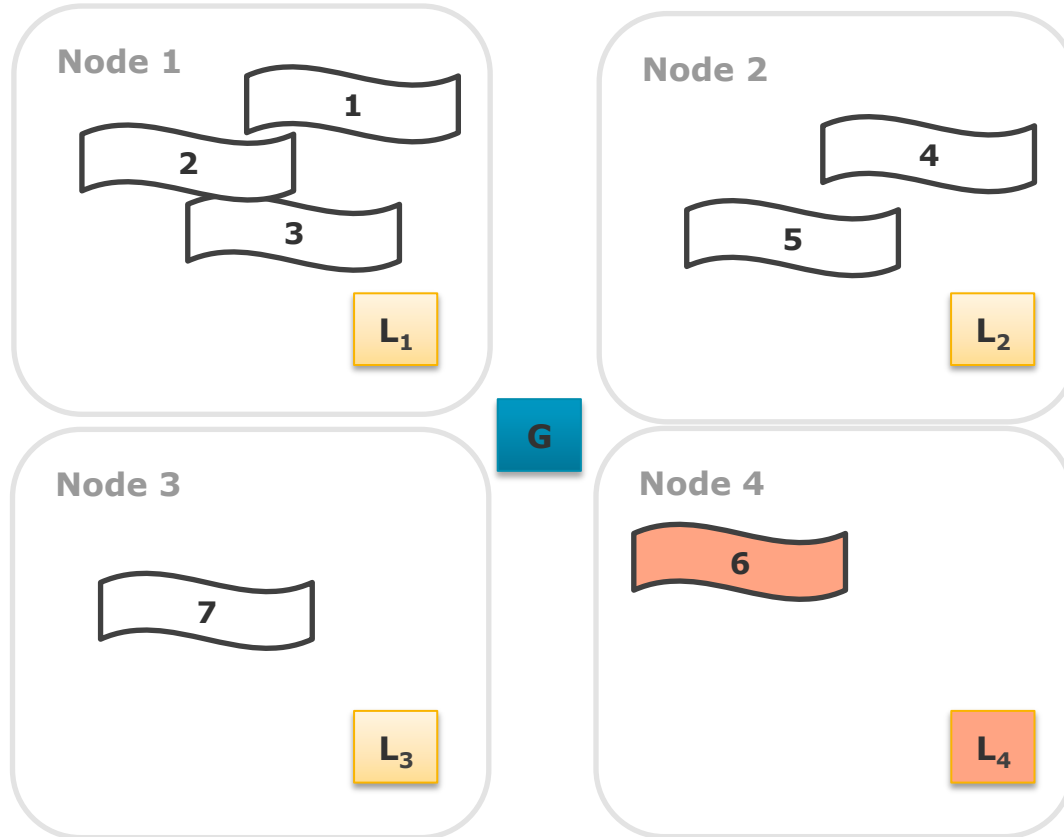
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 20

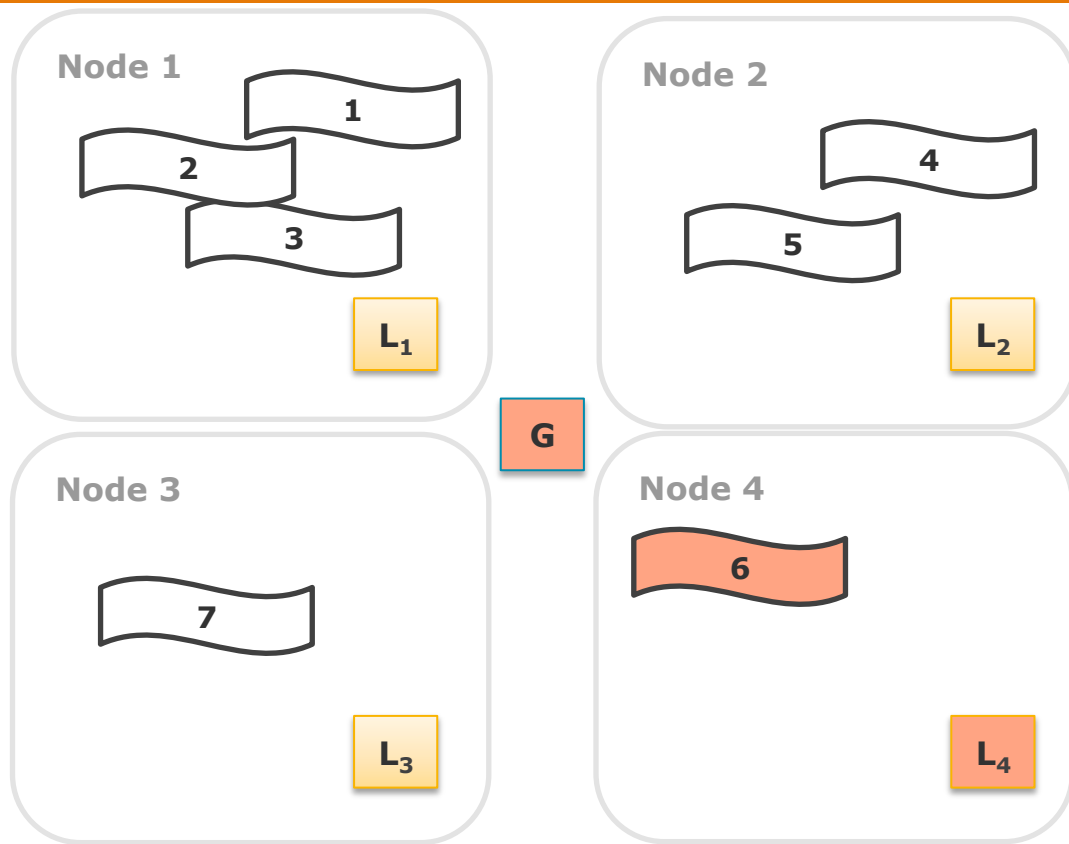
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 21

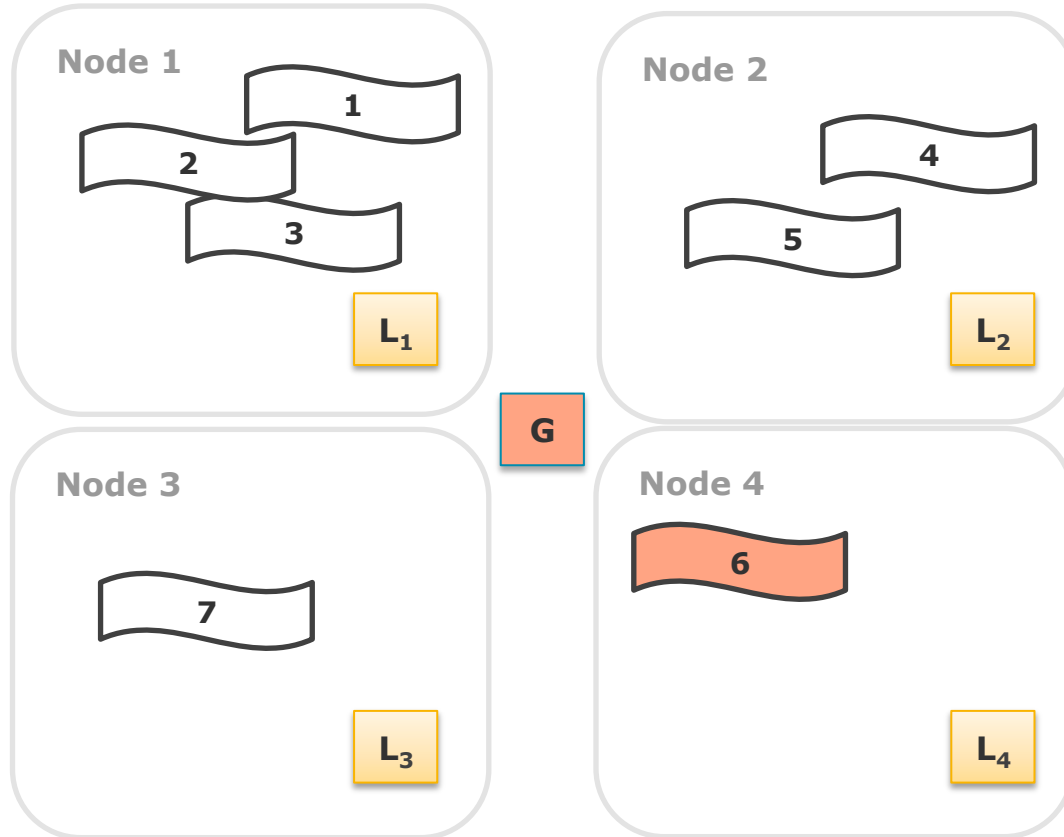
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 22

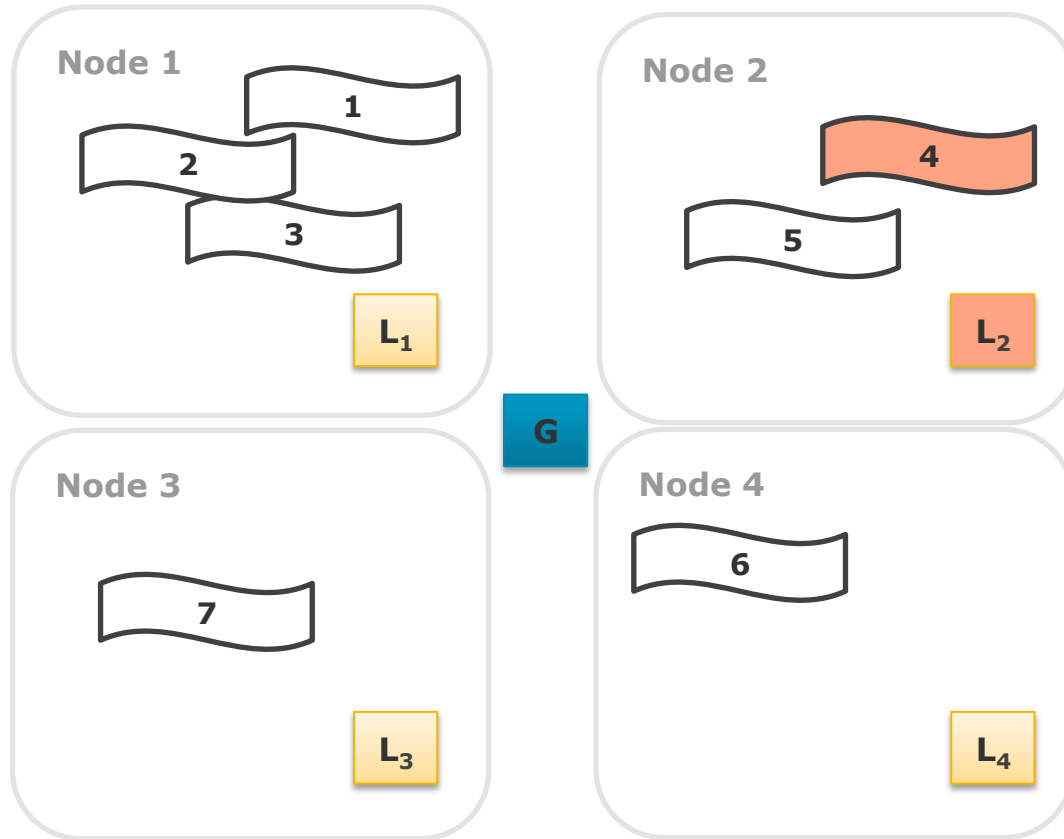
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 23

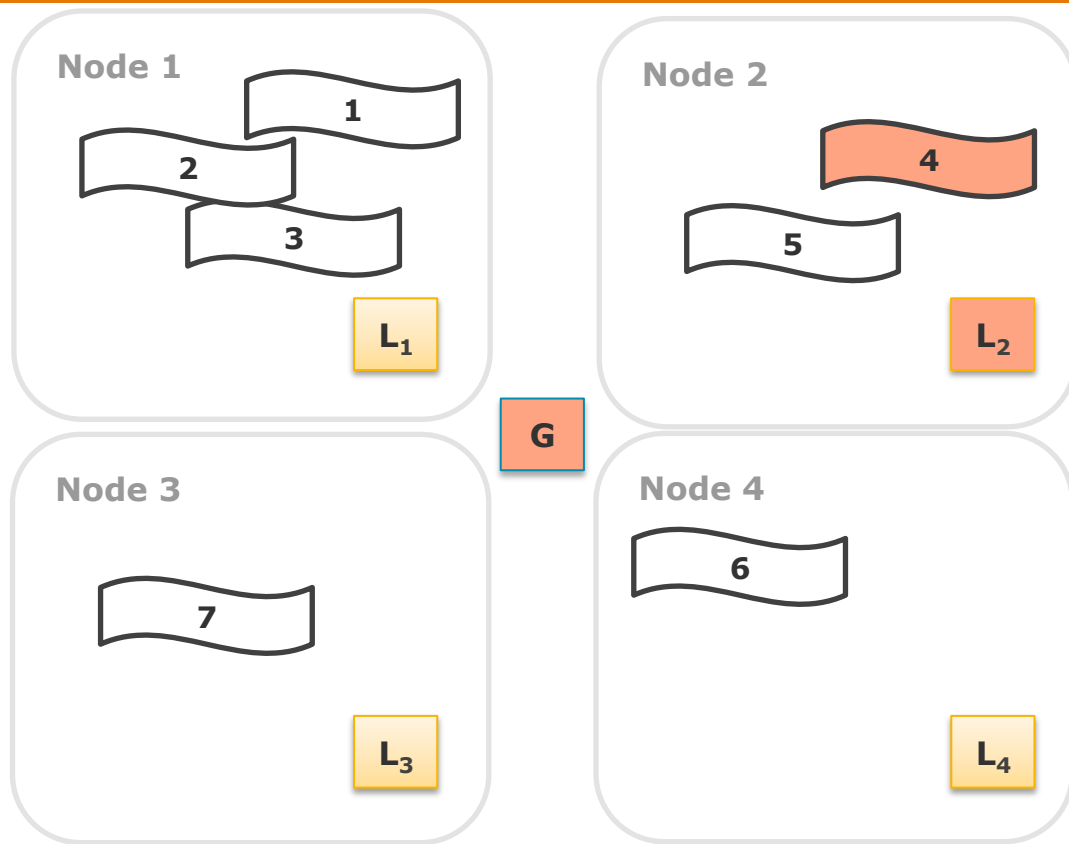
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 24

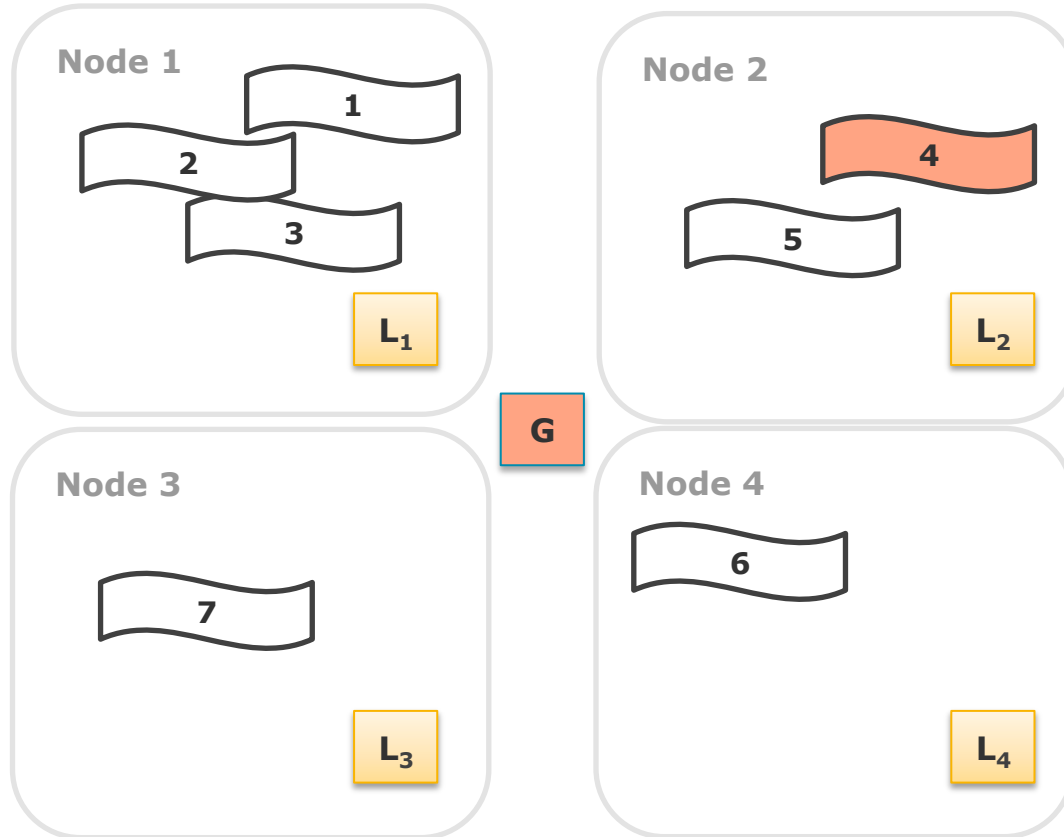
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 25

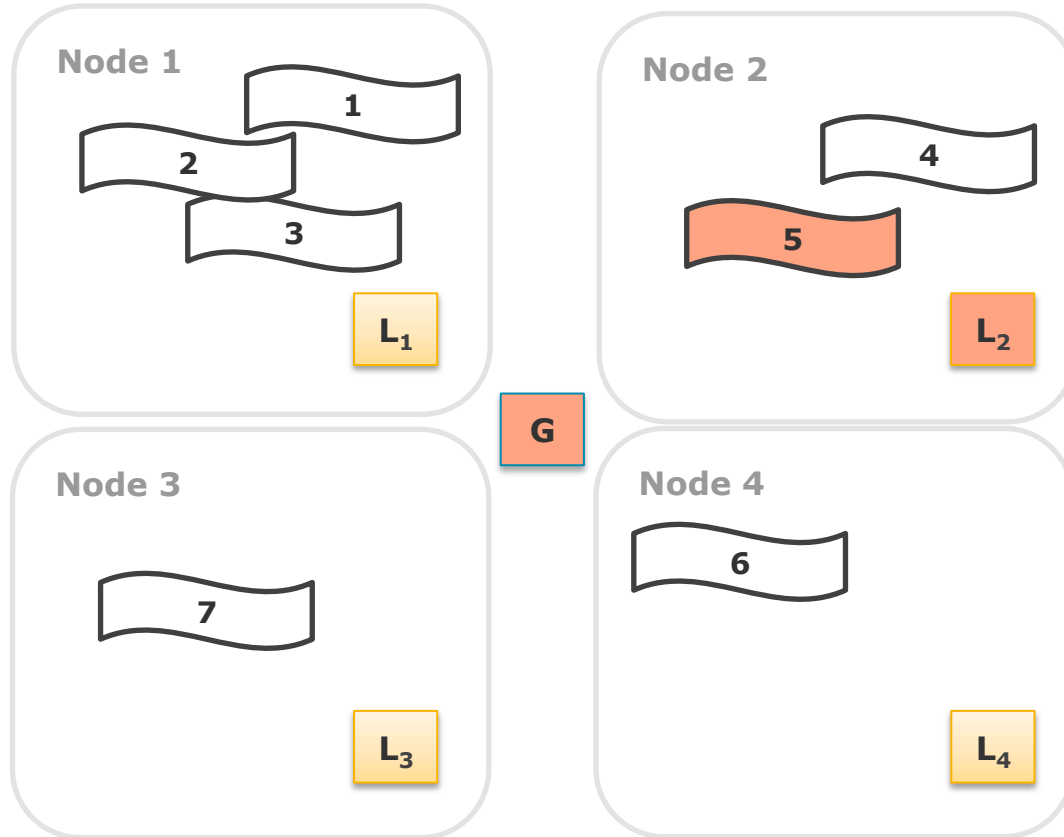
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 26

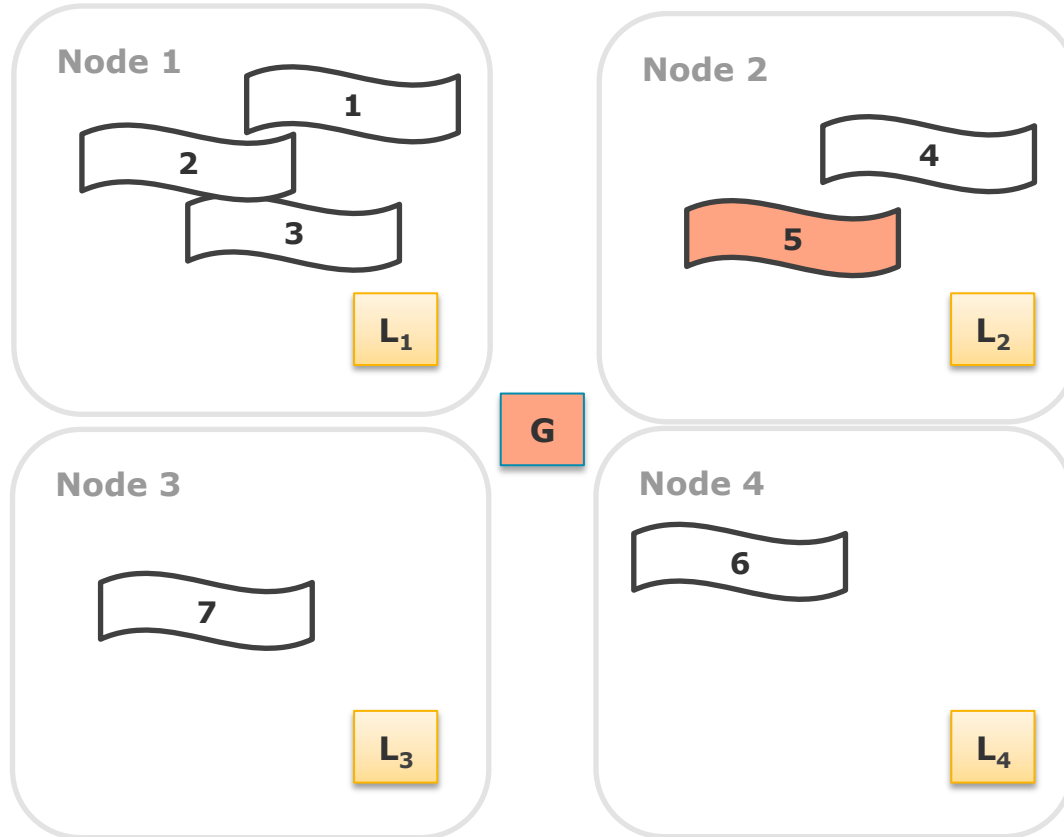
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 27

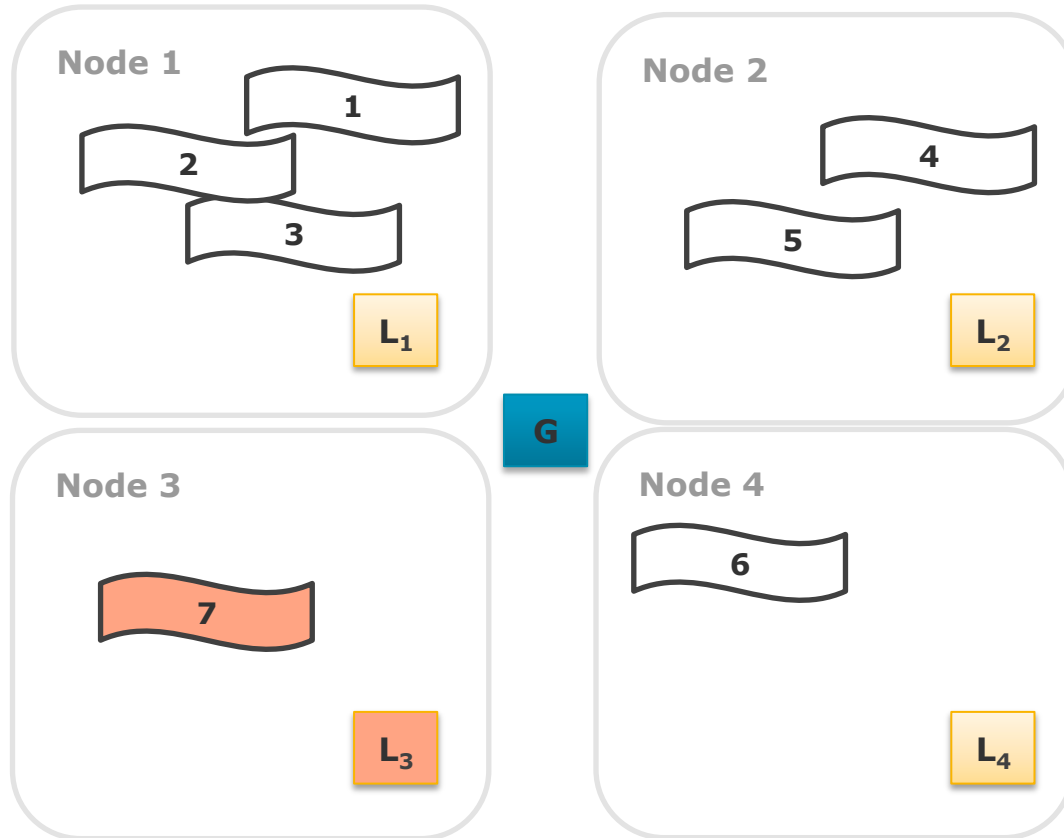
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 28

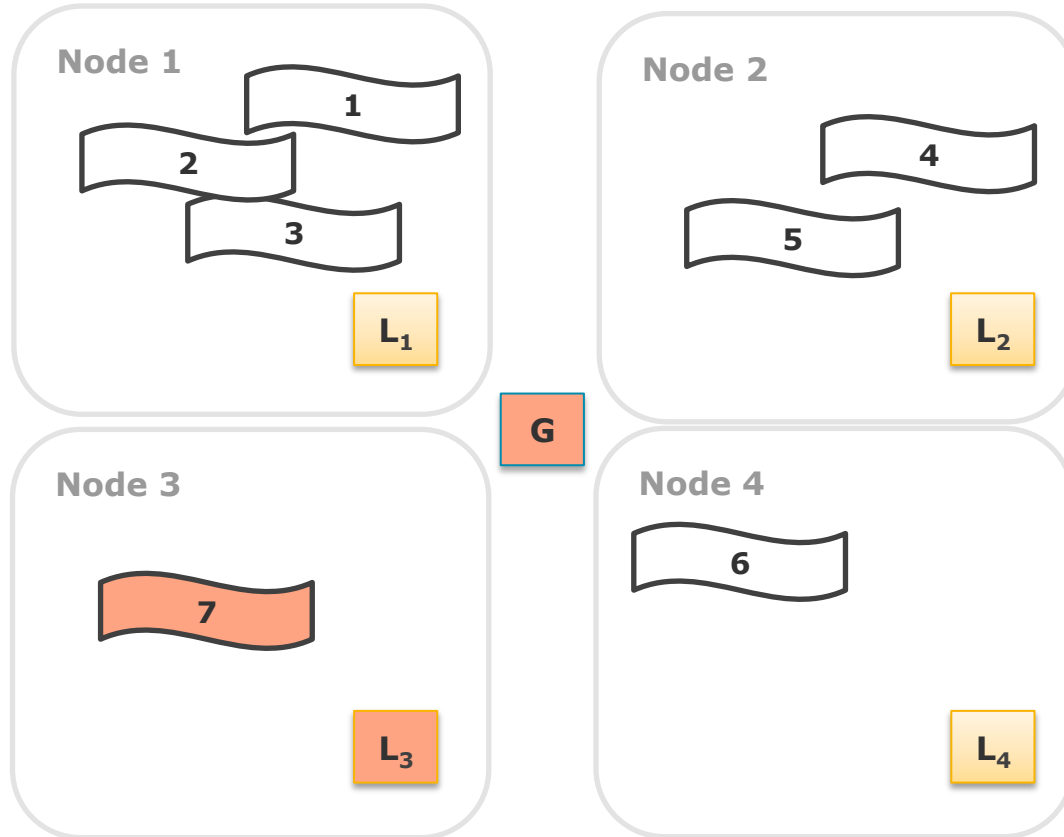
Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 29

Cohort Locks Example



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 30

Cohort Locks are a technique to compose NUMA-aware mutex locks from NUMA-oblivious mutex locks.

**NUMA-aware
Reader-Writer
Locks**

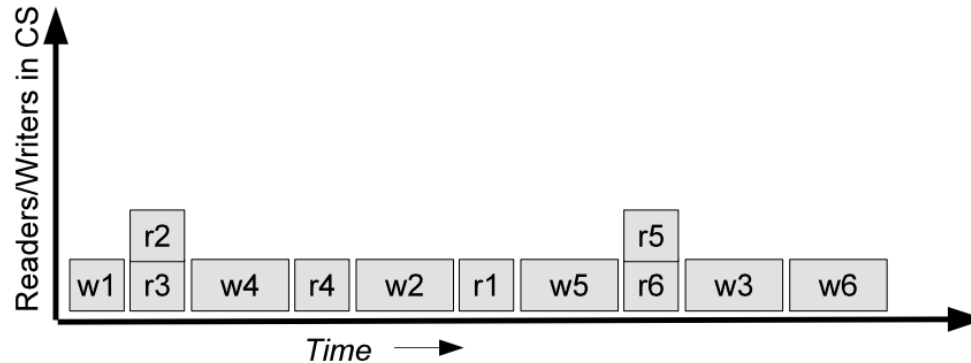
Tom Herold, Marco
Lamina
28.01.2015
Chart **31**

NUMA-aware Reader-Writer Locks

- Reduce lock migration frequency to generate better node-local locality by:
 - Batching read and write operations
 - Increase local cache hits
 - Trading “fairness” principle for maximized read concurrency

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **33**



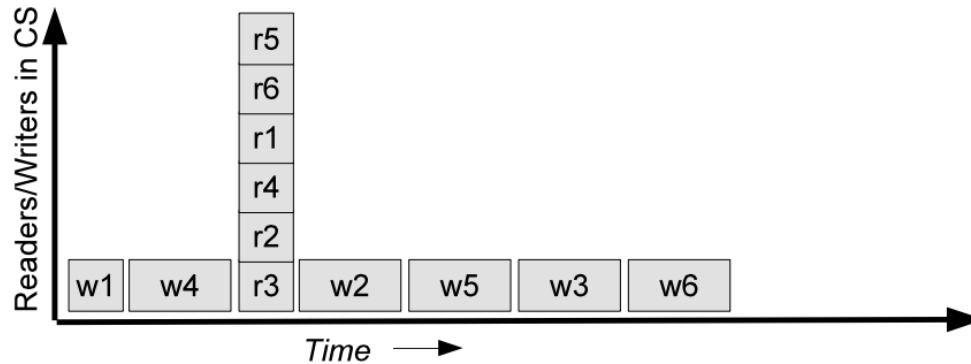
(a) Naïve reader-writer lock schedule

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 34

Source: [1]

Lock Design Goal II



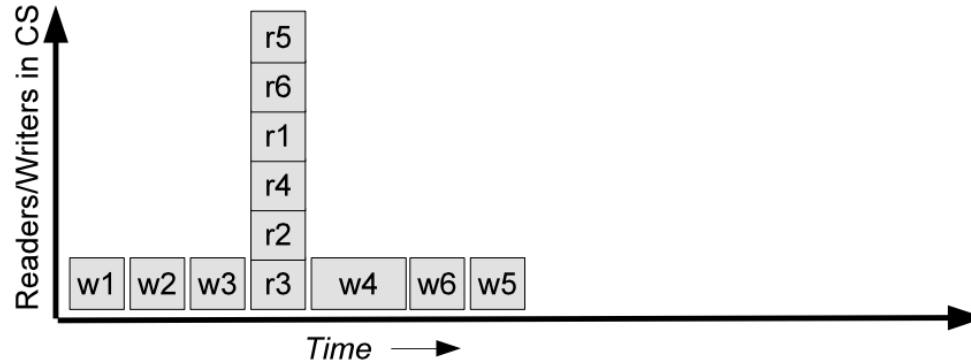
(b) Lock schedule with aggressive reader batching

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 35

Source: [1]

Lock Design Goal II



(c) Lock schedule with aggressive reader and writer batching

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 36

Source: [1]

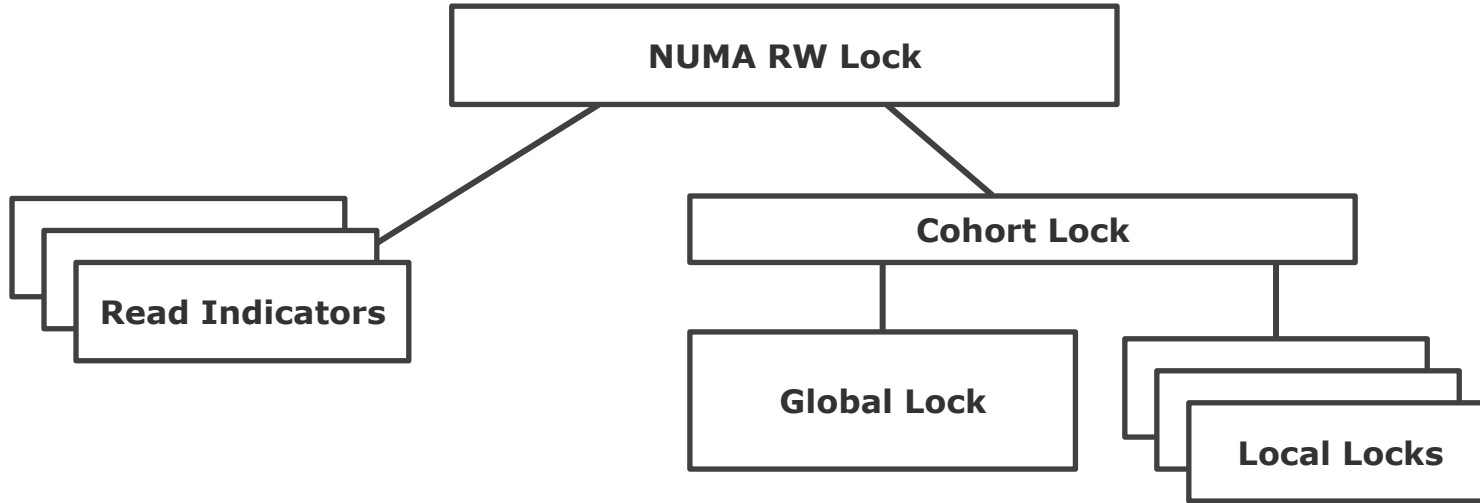
Classic NUMA

- Placement of memory relative to thread location
- CohortLocks
- About shared caches and their coherence state
- Reduce write invalidation, coherence misses and remote cache access

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **37**

A RW-Lock Instance



- One Read Indicator Counter per NUMA Node

- One global lock
- One local lock per NUMA Core

NUMA-aware Reader-Writer Locks

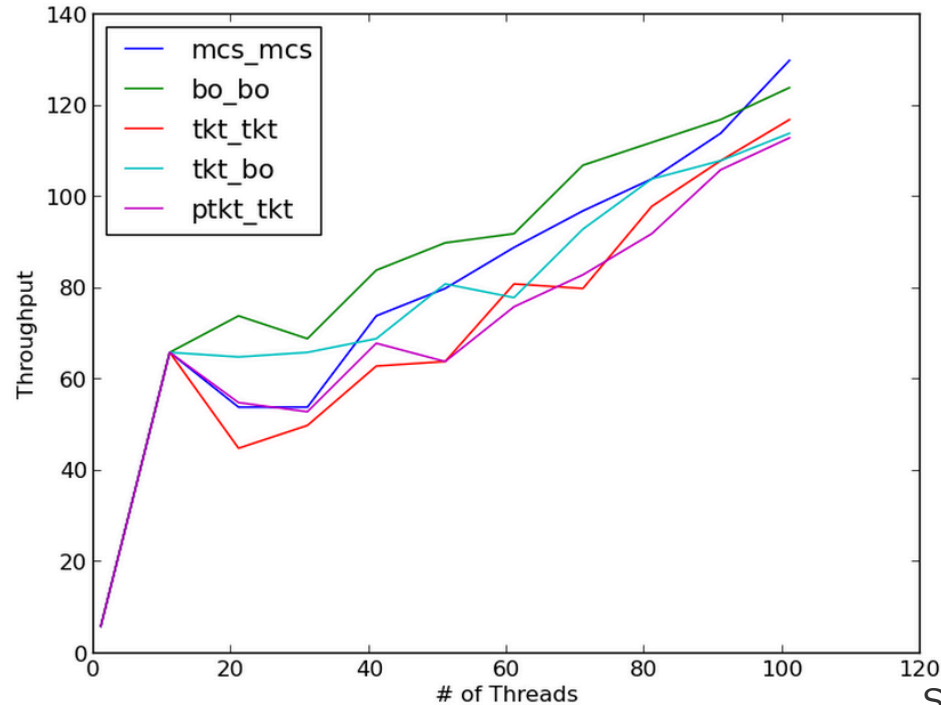
Tom Herold, Marco Lamina
28.01.2015
Chart 38

- NUMA-aware lock is oblivious of the underlying read indicator and mutex lock implementation
 - Backoff Locks
 - (Partitioned) Ticket Locks
 - MCS Locks
 - Simple Spin Locks
- Properties of the chosen locks influence and tune the RW lock
 - Abortion
 - Locality

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **39**

Performance of Different Implementations



Source: [4]

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 40

Lock Preferences - Neutral

```
1: reader:
2:     CohortLock.acquire()
3:     ReadIndr.arrive()
4:     CohortLock.release()
5:     <read-critical-section> ← Reader Concurrency
6:     ReadIndr.depart()
```

```
7: writer:
8:     CohortLock.acquire()
9:     while NOT(ReadIndr.isEmpty()) ← Wait for all readers
10:         Pause                               to finish
11:     <write-critical-section>
12:     CohortLock.release()
```

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 41

Problems with Neutral Preference

- Restriction of reader-reader concurrency by cohort lock acquisition
- High contention for central node under pressure → high interconnect bandwidth

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **42**

Assumptions

- Most RW locks are read-dominated
- Better throughput by batching read operations
- Bypass waiting readers

Benefits

- Increased scalability
- High Reader-reader concurrency

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **43**

```
1: reader:
2:   while RBarrier != 0
3:     Pause
4:   ReadIntr.arrive()
5:   while CohortLock.isLocked()
6:     Pause
7:   <read-critical-section>
8:   ReadIntr.depart()
```

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **44**

```
1: reader:
2:   while RBarrier != 0
3:     Pause
4:   ReadIntr.arrive()
5:   while CohortLock.isLocked()
6:     Pause
7:   <read-critical-section>
8:   ReadIntr.depart()
```

← Writers raise Barrier if out of patience

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 45

Problems with Reader Preference

- Neglect of writer throughput, yet high reader-reader concurrency
- Possible starvation of writers

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **46**

Benefits

- Writer preference leads to a build up of read locks, which are granted en masse
- Yet, most workers request a read lock and prevent starvation by an abundance of writers

Fairness

- Reader can turn “inpatient” and raise a barrier after a while

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **47**

```
1:writer:
2:   while WBarrier != 0
3:     Pause
4:   CohortLock.acquire()
5:   while NOT(ReadIndr.isEmpty())
6:     Pause
7:   <write-critical-section>
8:   CohortLock.release()
```

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **48**


```
1:writer:
2:   while WBarrier != 0
3:     Pause
4:   CohortLock.acquire()
5:   while NOT(ReadIndr.isEmpty())
6:     Pause
7:   <write-critical-section>
8:   CohortLock.release()
```

← Readers raise Barrier if out of patience

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 49

Lock Preferences – Writer

```
1: reader:
2:     bRaised = false // local flag
3:     start:
4:     ReadIndr.arrive()
5:     if CohortLock.isLocked()
6:         ReadIndr.depart()
7:         while CohortLock.isLocked()
8:             Pause
9:             if RanOutOfPatience AND ~bRaised
10:                 atomic_increment(Wbarrier)
11:                 bRaised = true
12:             goto start
13:     if bRaised
14:         atomical_decrement(Wbarrier)
15:     <read-critical-section>
16:     ReadIndr.depart()
```

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 50

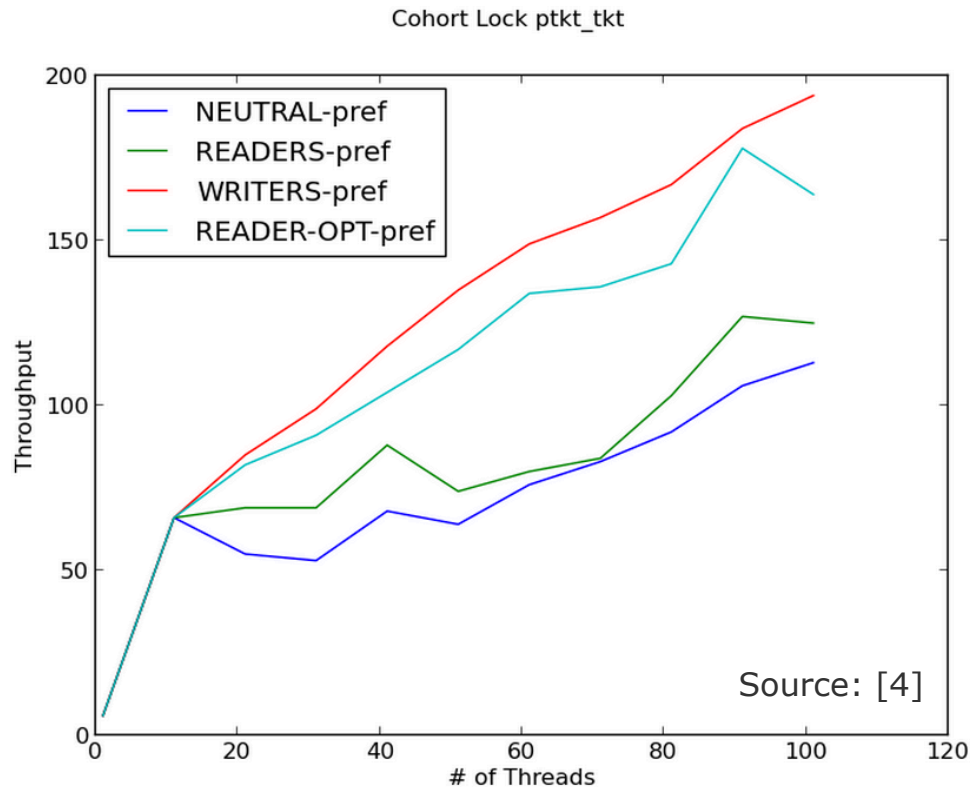
Lock Preferences – Writer

```
1: reader:
2:     bRaised = false // local flag
3:     start:
4:     ReadIndr.arrive()
5:     if CohortLock.isLocked()
6:         ReadIndr.depart()
7:         while CohortLock.isLocked()
8:             Pause
9:             if RanOutOfPatience AND ~bRaised
10:                 atomic_increment(Wbarrier)
11:                 bRaised = true
12:             goto start
13:     if bRaised
14:         atomical_decrement(Wbarrier)
15:     <read-critical-section>
16:     ReadIndr.depart()
```

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 51

NUMA-aware RW Locks: Benchmark of Different Preferences



NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 52

A large, horizontal decorative bar spans the width of the slide. It features a central orange rectangle with a yellow border on top and a red border on the bottom. The word 'Implementations' is centered within the orange area in white text.

Implementations

1. Oracle, MIT, Brown University
2. Brown University
3. University of Concepción / Azu Labs
4. Concurrency Kit

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **54**

First Implementation: Original Paper

NUMA-Aware Reader-Writer Locks

Irina Calciu
Brown University
irina@cs.brown.edu

Victor Luchangco
Oracle Labs
victor.luchangco@oracle.com

Dave Dice
Oracle Labs
dave.dice@oracle.com

Virendra J. Marathe
Oracle Labs
virendra.marathe@oracle.com

Yossi Lev
Oracle Labs
yossi.lev@oracle.com

Nir Shavit
MIT
shanir@csail.mit.edu

Abstract

Non-Uniform Memory Access (NUMA) architectures are gaining importance in mainstream computing systems due to the rapid growth of multi-core multi-chip machines. Extracting the best possible performance from these new machines will require us to revisit the design of the concurrent algorithms and synchronization primitives which form the building blocks of many of today's applications. This paper revisits one such critical synchronization primitive: the reader-writer lock.

"NUMA-Aware Reader-Writer Locks."
Calciu, Irina, et al.

coherent Non-Uniform Memory Access (NUMA) architectures.¹ These systems contain multiple nodes where each node has locally attached memory, a local cache and multiple processing cores. Such systems present a uniform programming model where all memory is globally visible and cache-coherent. The set of cache-coherent communications channels between nodes is referred to collectively as the interconnect. These inter-node links normally suffer from higher latency and lower bandwidth compared to the intra-node channels. To decrease latency and to conserve interconnect bandwidth, NUMA-aware policies encourage intra-node communication over inter-node communication.

Creating efficient software for NUMA systems is challenging because such systems present a naive uniform "flat" model of the relationship between processors and memory. The programmer must

relationship between processors and memory. The programmer must understand the underlying system topology and use special system-dependent library APIs to account for the system topology. NUMA-oblivious multithreaded programs suffer performance problems arising from long access bandwidth limits. Furthermore, inter-node and from interconnect bandwidth is a shared resource so coherence traffic generated by one thread can impede the performance of other unrelated threads because of queuing delays and channel contention. Concurrent data structures and synchronization constructs for modern multithreaded applications must be carefully designed to account for the underlying NUMA architectures. One key challenge is the underlying reader-writer (RW) lock. The lock

Source: [1]

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 55

First Implementation: Original Paper

NUMA-Aware Reader-Writer Locks

Irina Calciu
Brown University
irina@cs.brown.edu

Victor Luchangco
Oracle Labs
victor.luchangco@oracle.com

Dave Dice
Oracle Labs
dave.dice@oracle.com

Virendra J. Marathe
Oracle Labs
virendra.marathe@oracle.com

Yossi Lev
Oracle Labs
yossi.lev@oracle.com

Nir Shavit
MIT
shanir@csail.mit.edu

Abstract

Non-Uniform Memory Access (NUMA) architectures are gaining importance in mainstream computing systems due to the rapid growth of multi-core multi-chip machines. Extracting the best possible performance from these new machines will require us to re-visit the design of the concurrent algorithms and synchronization primitives that form the building blocks of many of today's applications. This paper presents a new set of critical synchronization primitives for NUMA-aware policies that encourage intra-node communication over inter-node communication.

coherent Non-Uniform Memory Access (NUMA) architectures. These systems contain multiple nodes where each node has locally attached memory, a local cache and multiple processing cores. Such systems present a uniform programming model where all memory is globally visible and cache-coherent. The set of cache-coherent communications channels between nodes is referred to collectively as the interconnect. These inter-node links normally suffer from higher latency and lower bandwidth compared to the intra-node channels. To decrease latency and to conserve interconnect bandwidth, NUMA-aware policies encourage intra-node communication over inter-node communication.

Creating efficient software for NUMA systems is challenging because such systems present a naive uniform "flat" model of the underlying architecture. In such systems, processors and memory are distributed across multiple nodes, and the performance of the system is limited by long access latencies caused by inter-node coherence traffic and from interconnect bandwidth limits. Furthermore, inter-node and from interconnect bandwidth is a shared resource so coherence traffic generated by one thread can impede the performance of other unrelated threads because of queuing delays and channel contention. Concurrent data structures and synchronization constructs at the level of modern multithreaded applications must be carefully designed to take advantage of the underlying NUMA architectures. One key challenge is the design of a reader-writer (RW) lock. The RW lock is a traditional mutual exclusion primitive that allows multiple readers to hold the lock

"Unfortunately the code has not been made open source, so we are unable to share it"
Dave Dice, Oracle

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015
Chart 56

First Implementation: Oracle

"Unfortunately the code has not been made open source, so we are unable to share it"

Dave Dice, Oracle

■ → Idea:

- Dynamically **inject** NUMA-aware locking implementation with LD_Preload on Linux
- **Overwrite** `pthread_rwlock_wrlock()`, `pthread_rwlock_rdlock()`
- **Delegate** calls to NUMA-aware RW lock implementation from "azulabs.com" (based on paper) [4]

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **57**

Second Implementation: Brown University Project Report

NUMA aware locks Implementation and Evaluation

Zhongyu Ma

ScM research project Spring 2012

Brown University Project Report "NUMA aware locks Implementation and Evaluation" Zhongyu Ma

1 Introduction

Programs running on NUMA machines are sensitive to memory access locality[3]. Accessing data on the local node is significantly faster than remote memory. Thus, designing locks that can take advantage of this property would improve performance. We review three papers related to this topic and study how different NUMA aware locks implement these locks in C++, and evaluate them on the different NUMA architectures in terms of performance.

2 Implementation

Generally, all the locks that we implemented are spin locks, including backoff locks and queue locks. First, we built the Backoff TTAS(Test and Set) lock, as well as the CLH and MCS queue lock. We built these simple locks not only because some of the NUMA aware locks are assembled by them, it is also to compare the NUMA aware locks with them. Some NUMA aware lock use only the backoff locks and some comprise both these two kind of locks. In the following section, we describe locks and how we implemented them in detail.

Source: [9]

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 58

Third Implementation: University of Concepción / Azu Labs



The screenshot shows the GitHub interface for the repository 'azu-labs / rw-numa-locks'. The repository name is at the top left. Below it, the description 'NUMA-Aware Reader-Writer Locks' is visible. The repository has 10 commits, 1 branch, and 0 releases. The latest commit is 4d9a46341f, made 8 months ago. The commit history shows several updates to README.md and LICENSE files, all made 8 months ago. The repository is watched by 1 person and has 1 contributor.

azu-labs / rw-numa-locks

NUMA-Aware Reader-Writer Locks

10 commits 1 branch 0 releases 1 contributor

branch: master rw-numa-locks / + latest commit 4d9a46341f

Update README.md 8 months ago

Update LICENSE 8 months ago

Update README.md 8 months ago

Update README.md 8 months ago

Update README.md 8 months ago

Update README.md 8 months ago

Update README.md 8 months ago

rw_mcs_mcs.c initial version

rw_ptkt_tkt.c initial version

University of Concepción, Chile
"NUMA-Aware Reader-Writer Locks Experiments"
Leo Ferres, Erick Elejalde

Source: [5]

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 59

Third Implementation: University of Concepción / Azu Labs

- Experimental Implementation in C
- Open Source on Github [10]
- Created mini-benchmarks to evaluate them:
 - N threads access shared array
 - Each thread has 10% probability of becoming a writer (is reader otherwise)
 - Critical section accesses array at random position
- Performance metric for mini-benchmark: **“Throughput”**
 - How many reads/writes were executed on the array

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **60**

Fourth Implementation: Concurrency Kit

← Manual Pages

CK_ARRAY_FOREACH(3)

iterate through an array

CK_COHORT_INIT(3)

initialize instance of a cohort type

CK_COHORT_INSTANCE(3)

declare an instance of a cohort type

CK_COHORT_LOCK(3)

acquire cohort lock

CK_COHORT_PROTOTYPE(3)

define cohort type with specified lock types

CK_COHORT_TRYLOCK(3)

try to acquire cohort lock

CK_COHORT_TRYLOCK_PROTOTYPE(3)

define cohort type with specified lock types

release cohort lock

invoke hash function with hash set seed

Open Source / Backtrace.io

<http://concurrencykit.org/>

Samy Al Bahra + Contributors

Source: [6]

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **61**

Fourth Implementation: Concurrency Kit

- Library with Concurrency primitives, non-blocking data structures in C:
 - Multiple locks
 - Atomic Operations
 - Ring Buffers
 - Cohort Locks
 - Individually configurable global and local locks
 - Very abstract MACRO programming
- Provides RW Cohort Lock(!)
- Active Community / Mailing List, but almost no documentation

**NUMA-aware
Reader-Writer
Locks**

Tom Herold, Marco
Lamina
28.01.2015
Chart **62**

A large, horizontal rectangular box with a gradient background, transitioning from dark red on the left to yellow on the right. The text 'Hands On' is centered within this box in a white, sans-serif font.

Hands On

```
#include <pthread.h>

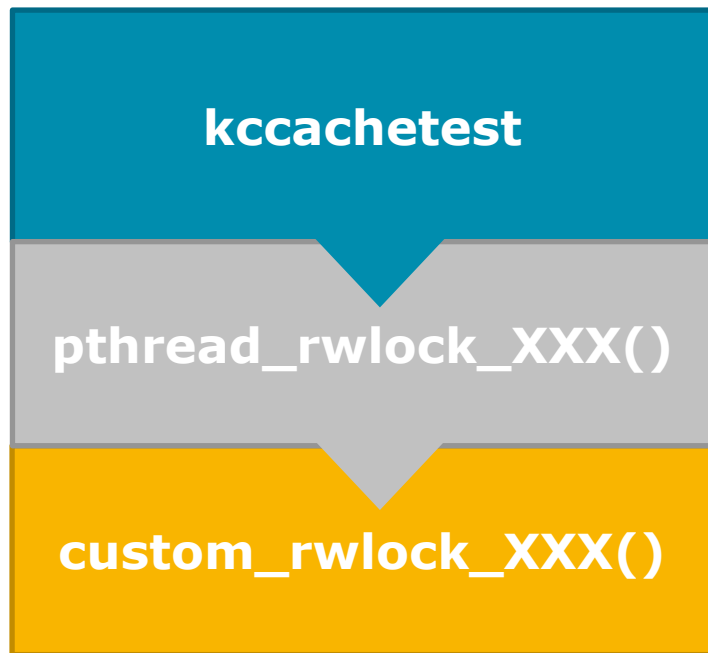
int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);

int pthread_rwlock_init(pthread_rwlock_t *rwlock,
                        const pthread_rwlockattr_t *restrict attr);
int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 64



- Kyoto-Cabinet's kccachetest
 - Open Source Database Implementation [4]
 - Heavy use of PThread's RW Locks
 - Benchmark random mixed operations to stress test the in-memory CacheDB

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart 65

- [1] Calciu, Irina, et al. "NUMA-aware reader-writer locks." ACM SIGPLAN Notices. Vol. 48. No. 8. ACM, 2013.
- [2] Dice, David, Virendra J. Marathe, and Nir Shavit. "Lock cohorting: a general technique for designing NUMA locks." ACM SIGPLAN Notices. Vol. 47. No. 8. ACM, 2012.
- [3] Dice, David, Virendra J. Marathe, and Nir Shavit. "Lock cohorting: a general technique for designing NUMA locks. An Extension" Unreleased Draft
- [4] <https://github.com/ldgeo/kyotocabinet>
- [5] http://azu-labs.com/numa_locks/
- [6] <http://concurrencykit.org/>
- [7] http://en.wikipedia.org/wiki/Race_condition
- [8] <http://blog.markedup.com/2014/07/easy-mode-synchronizing-multiple-processes-with-file-locks/>

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **66**

- [9] Ma, Zhongyu. “NUMA aware locks Implementation and Evaluation”, 2012
<http://cs.brown.edu/research/pubs/theses/masters/2012/ma.pdf> , visited 10.01.2014
- [10] <https://github.com/azu-labs/rw-numa-locks/>

NUMA-aware Reader-Writer Locks

Tom Herold, Marco
Lamina
28.01.2015
Chart **67**

Kccachetest with NUMA RW Lock with CK

```
tom.herold@ubuntu-numa0101:~/numa2014/rw-proxy$ make run
LD_PRELOAD=`pwd`/rwlock.so kccachetest wicked -th 24 -capsiz 2000000 100000
<Wicked Test>
  seed=1514826710  rnum=100000  thnum=24  itnum=1  opts=0  bnum=-1  capcnt=-1  capsiz=2000000  lv=
0
..... (00010000)
..... (00020000)
..... (00030000)
..... (00040000)
..... (00050000)
..... (00060000)
..... (00070000)
..... (00080000)
..... (00090000)
..... (00100000)
type: CacheDB (cache hash database) (type=0x20)
format version: 5 (libver=16.13) (chksum=0xFF)
path: *
status flags: (flags=0)
options: (opts=0)
opaque: 0
buckets: 1048583 (used=27640) (load=1.01)
count: 28018 (28.018 thousand) (capcnt=-1)
size: 10390452 (9.909 MiB) (capsiz=2000000)
memory: 13287424
time: 9.097 ←
ok
```

Presentation Title

Speaker, Job
Description, Date if
needed
Chart 68

Kccachetest with PThread RW Lock

```
tom.herold@ubuntu-numa0101:~/numa2014/rw-proxy$ make pthread
kccachetest wicked -th 24 -capsiz 2000000 100000
<Wicked Test>
  seed=1514925237 rnum=100000 thnum=24 itnum=1 opts=0 bnum=-1 capcnt=-1 capsiz=2000000 lv=
0
..... (00010000)
..... (00020000)
..... (00030000)
..... (00040000)
..... (00050000)
..... (00060000)
..... (00070000)
..... (00080000)
..... (00090000)
..... (00100000)
type: CacheDB (cache hash database) (type=0x20)
format version: 5 (libver=16.13) (chksum=0xFF)
path: *
status flags: (flags=0)
options: (opts=0)
opaque: 0
buckets: 1048583 (used=156993) (load=1.06)
count: 170331 (170.331 thousand) (capcnt=-1)
size: 20763985 (19.802 MiB) (capsiz=2000000)
memory: 24854528
time: 7.783 ←
ok
```

Presentation Title

Speaker, Job
Description, Date if
needed
Chart 69

Demo Results

“Perf mem rec” for NUMA RW Lock with CK

Samples: 89K of event 'cpu/mem-loads/pp', Event count (approx.): 942862

80,00%	88295	L1 hit
9,17%	524	LFB hit
7,35%	699	L2 hit
2,33%	141	L3 miss
0,82%	101	L3 hit
0,32%	181	Uncached hit

Presentation Title

Speaker, Job
Description, Date if
needed
Chart 70

Demo Results

“Perf mem rec” for Pthread RW Lock

Samples: 64K of event 'cpu/mem-loads/pp', Event count (approx.): 1322361

52,27%	58183	L1 hit
18,35%	1969	LFB hit
10,80%	787	L3 miss
10,02%	2362	L2 hit
5,38%	511	L3 hit
2,06%	86	Remote Cache (1 hop) hit
1,01%	555	Uncached hit
0,10%	3	Remote RAM (1 hop) hit

Presentation Title

Speaker, Job
Description, Date if
needed
Chart 71