Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina

04.02.2015

NUMA Seminar

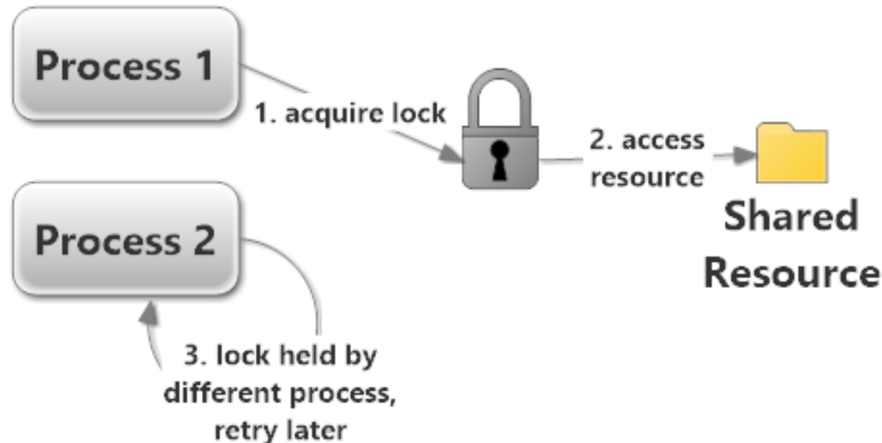
Agenda

Hasso Plattner Institut

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



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

HPI Hasso Plattner Institut

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**

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

Locking Basics

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

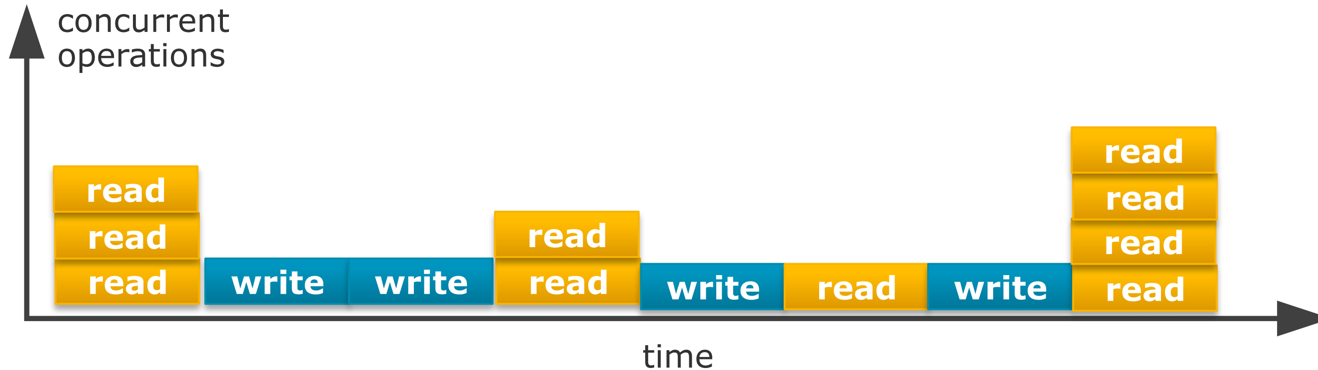
Live Lock

Two threads activate each other in an infinite loop

Reader-Writer Locks

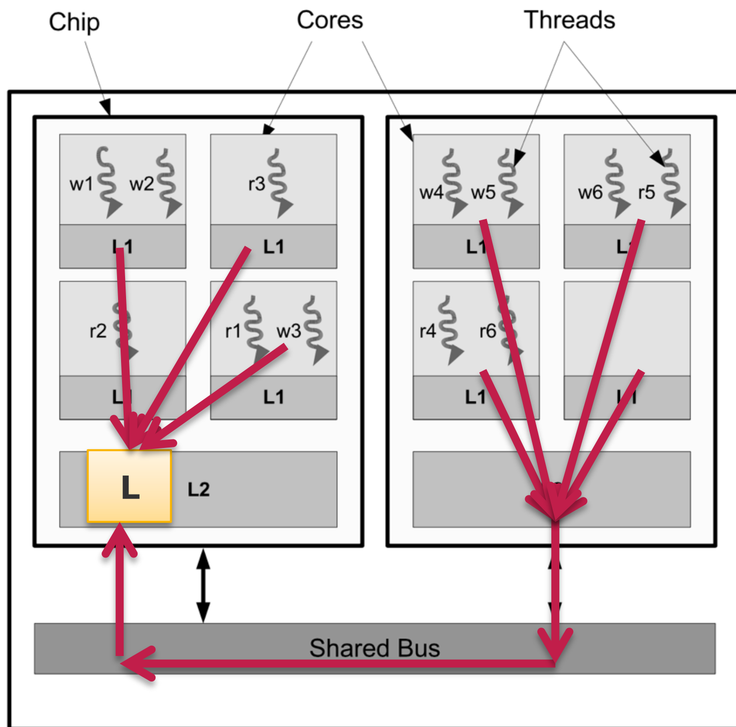
HPI Hasso Plattner Institut

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



HPI Hasso Plattner Institut

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**

Cohort Lock Properties

HPI Hasso Plattner Institut

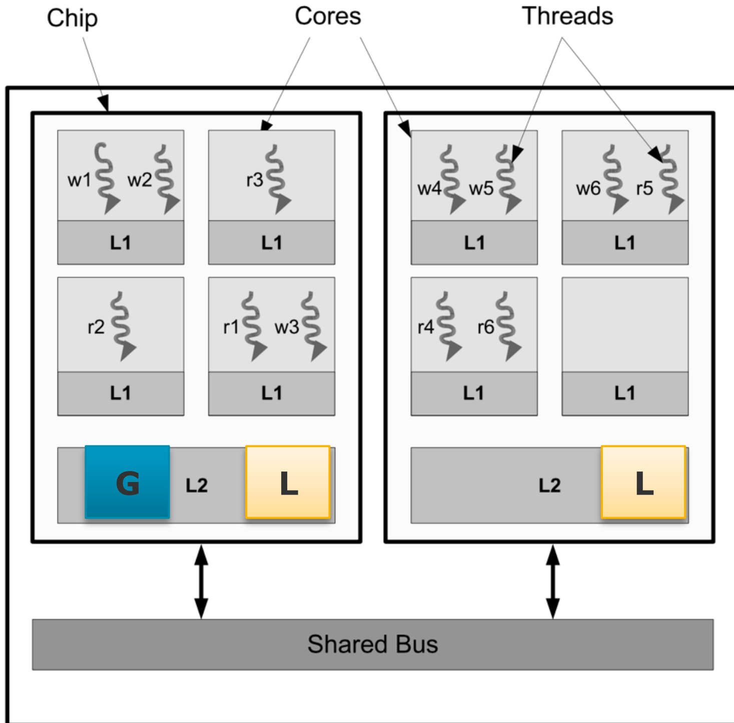
Cohort Detection

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

Avoiding Lock Migrations: Cohort Locking

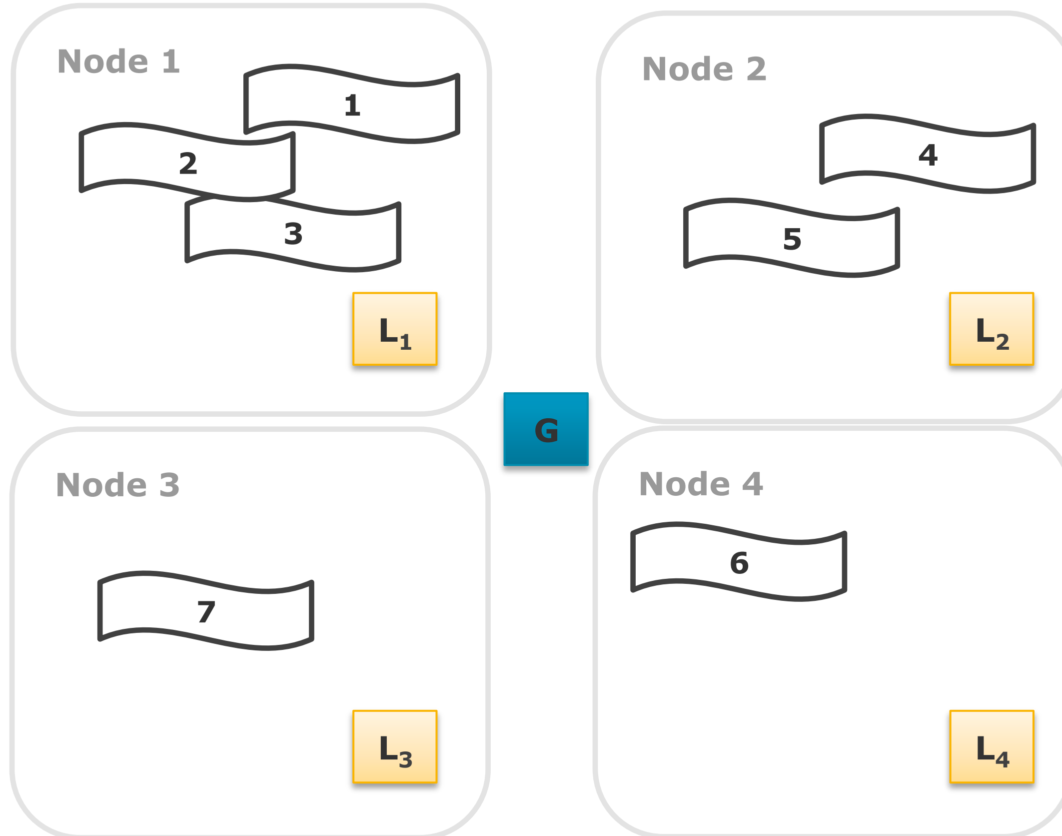


- Two locks:
  - **G** → global (thread-oblivious)
  - **L** → node-local

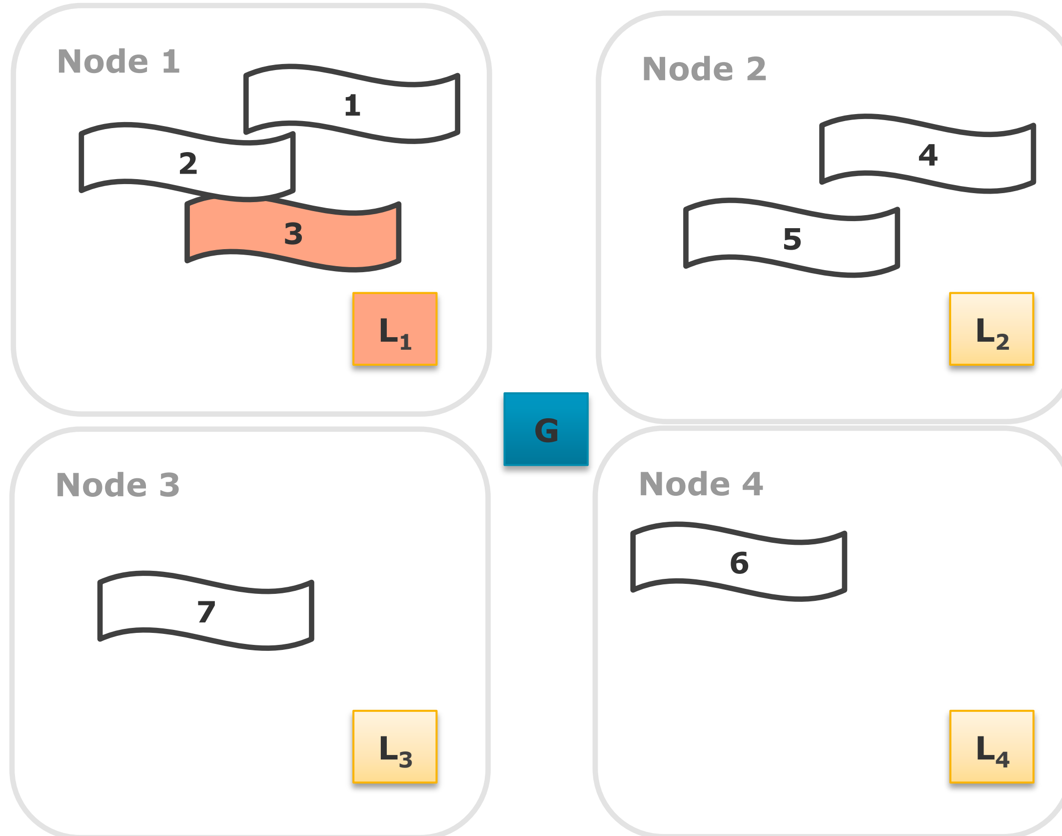
```
acquireCohortLock() {
    L.acquire()
    G.acquire()
}

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

Cohort Locks
Example

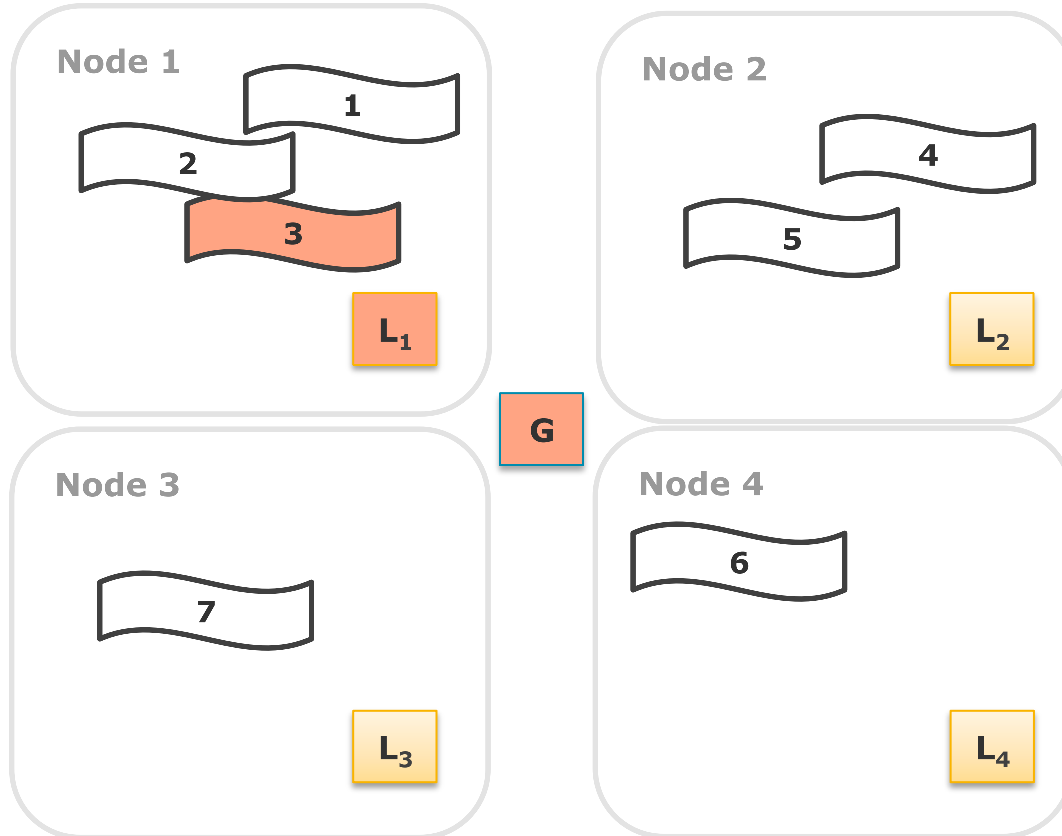


Cohort Locks
Example

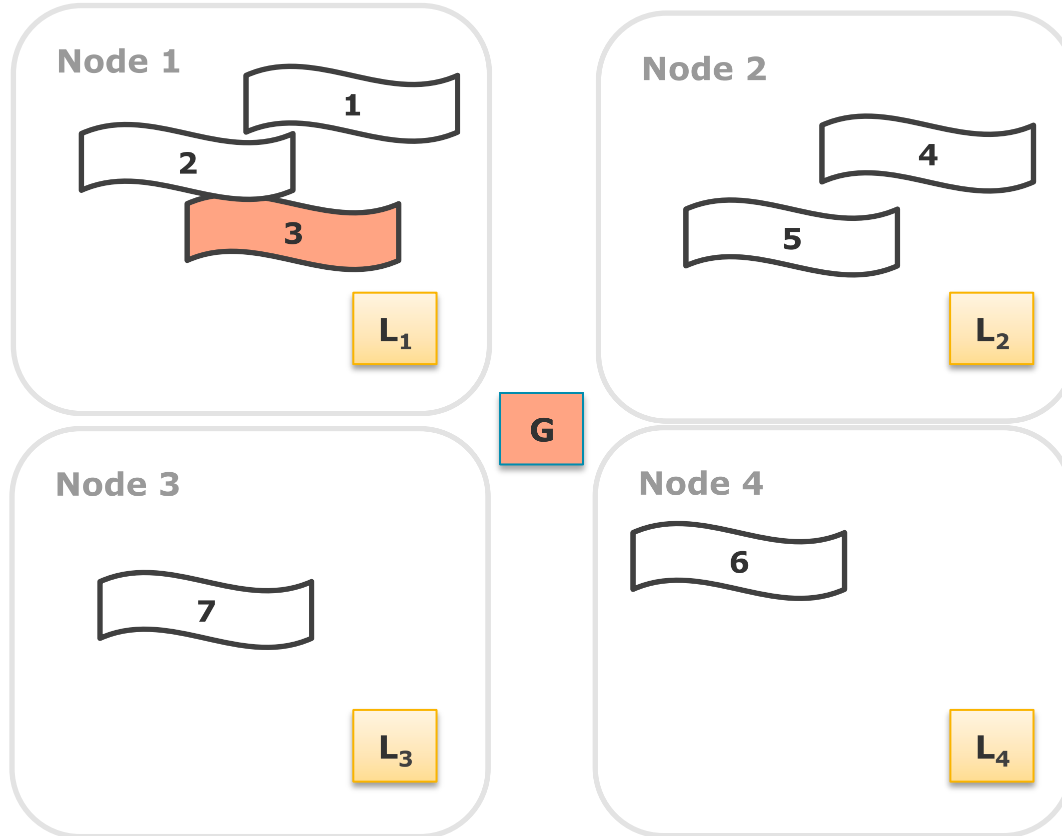


HPI Hasso Plattner Institut

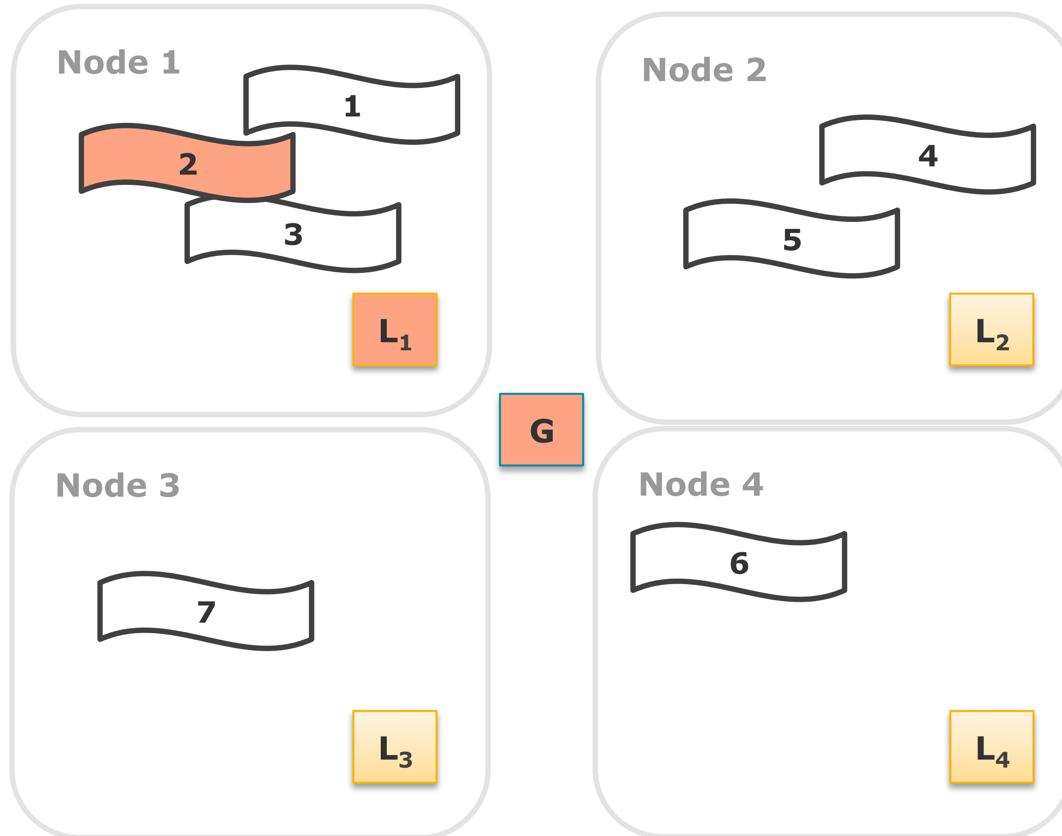
Cohort Locks
Example



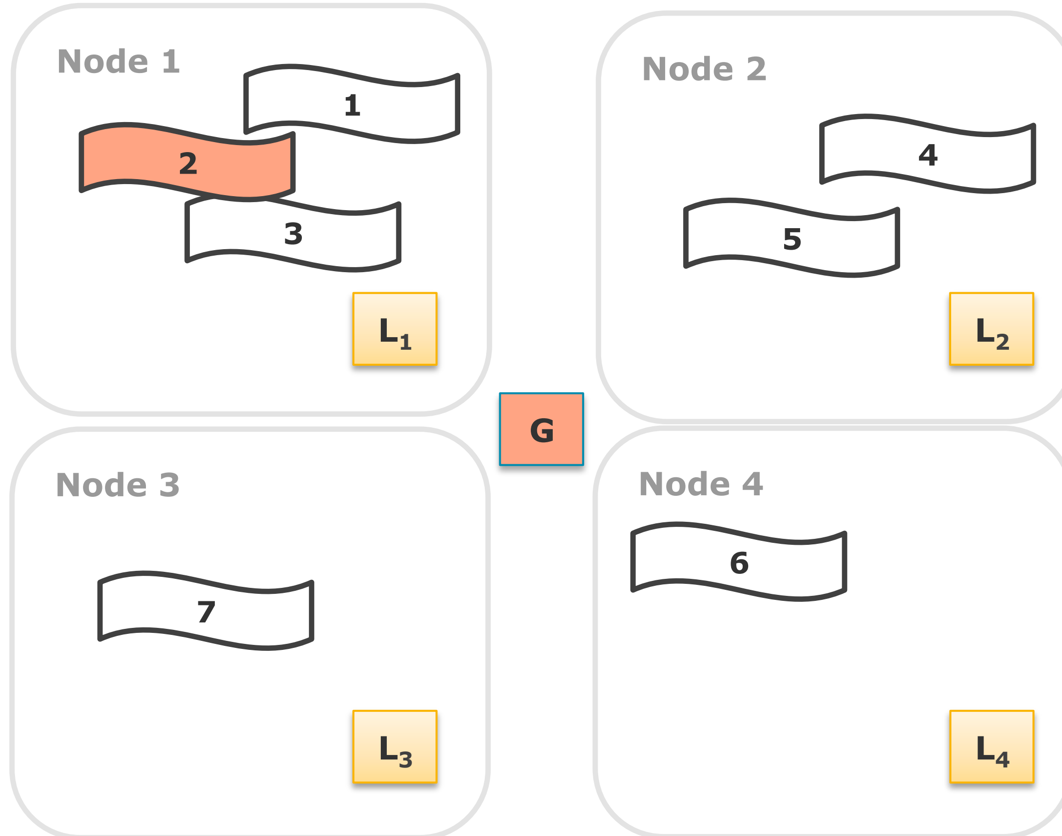
Cohort Locks
Example



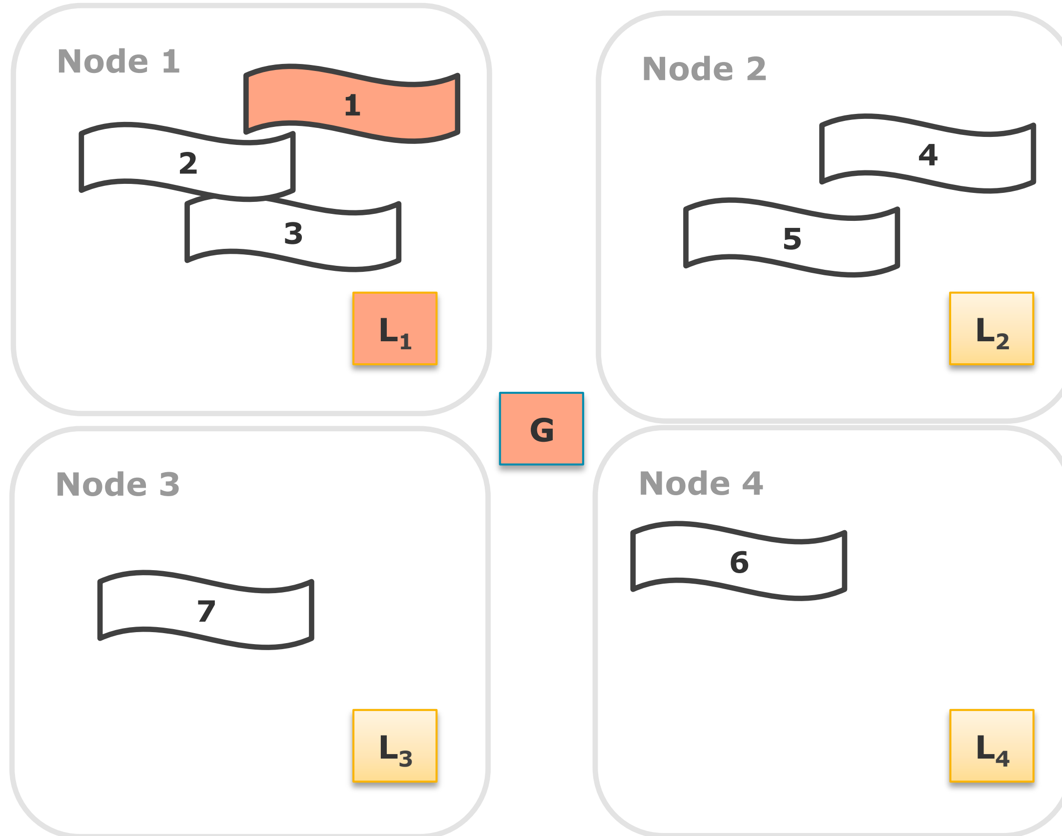
Cohort Locks
Example



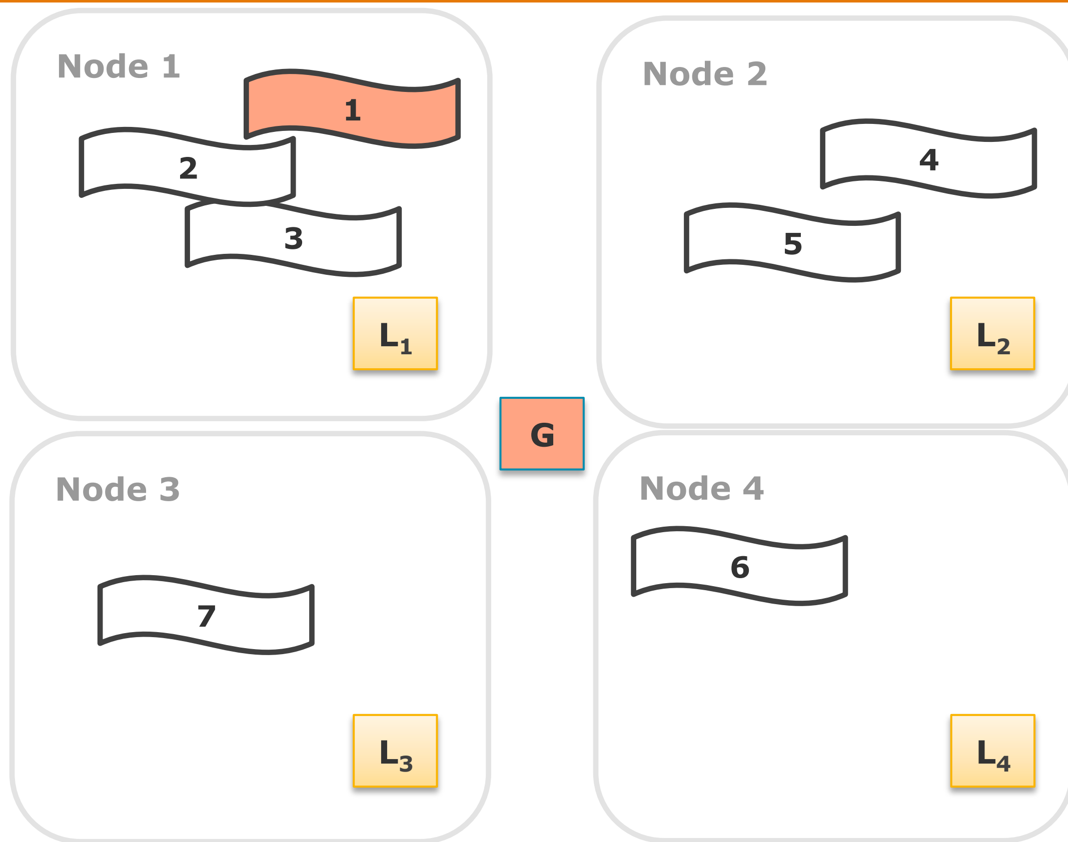
Cohort Locks
Example



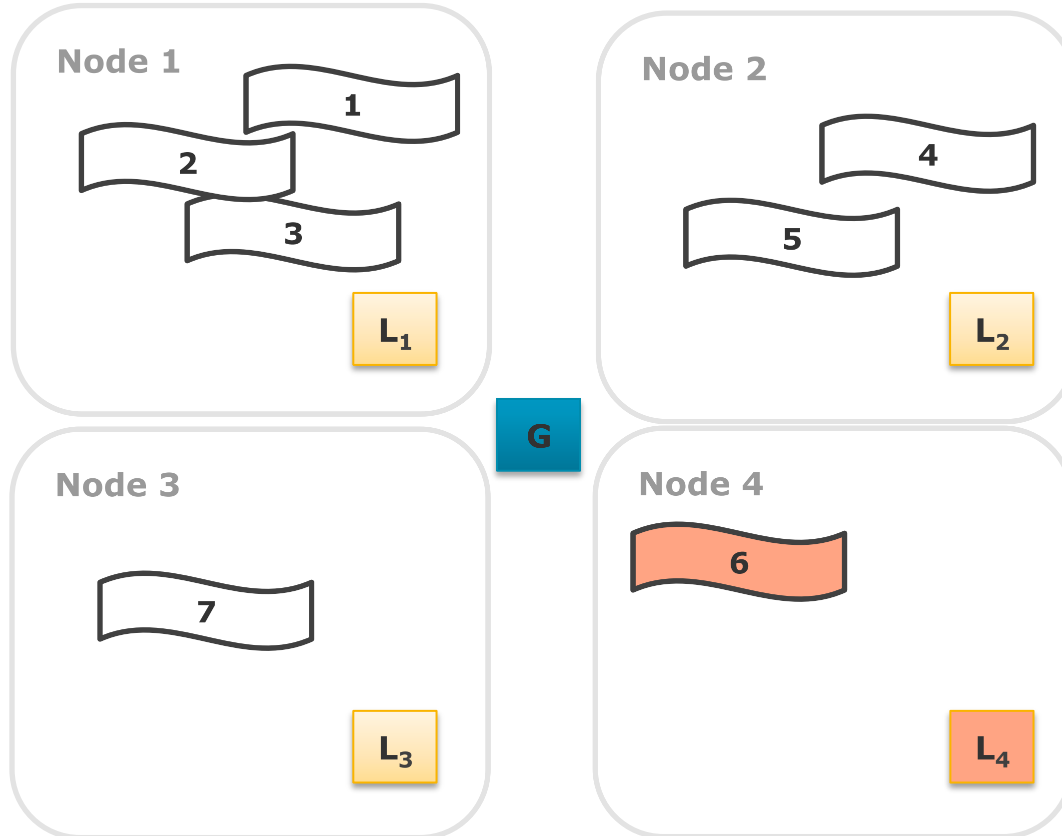
Cohort Locks
Example



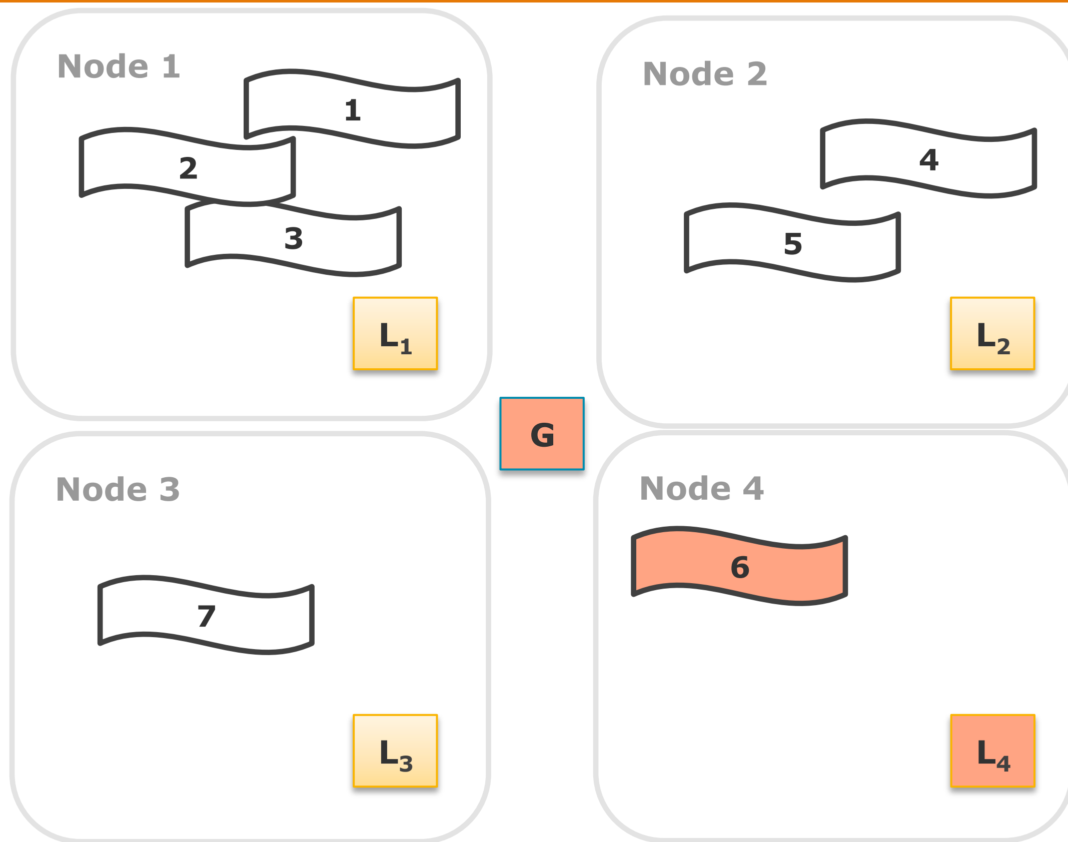
Cohort Locks
Example



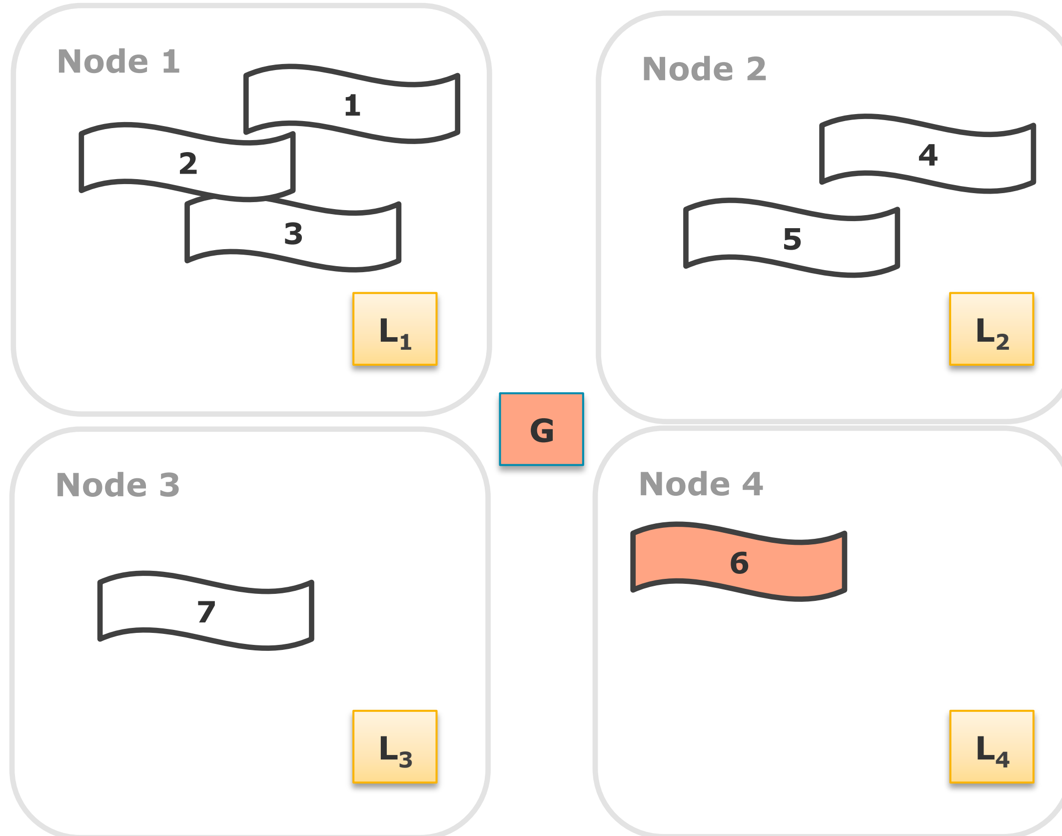
Cohort Locks
Example



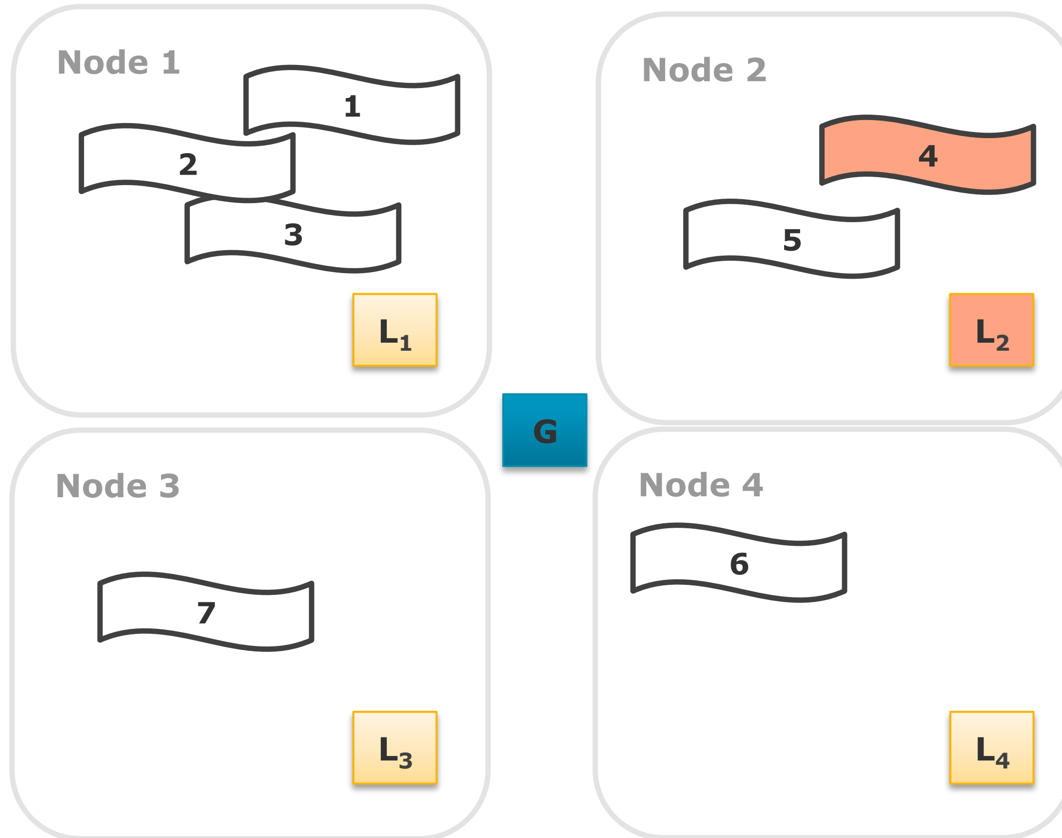
Cohort Locks
Example



Cohort Locks Example

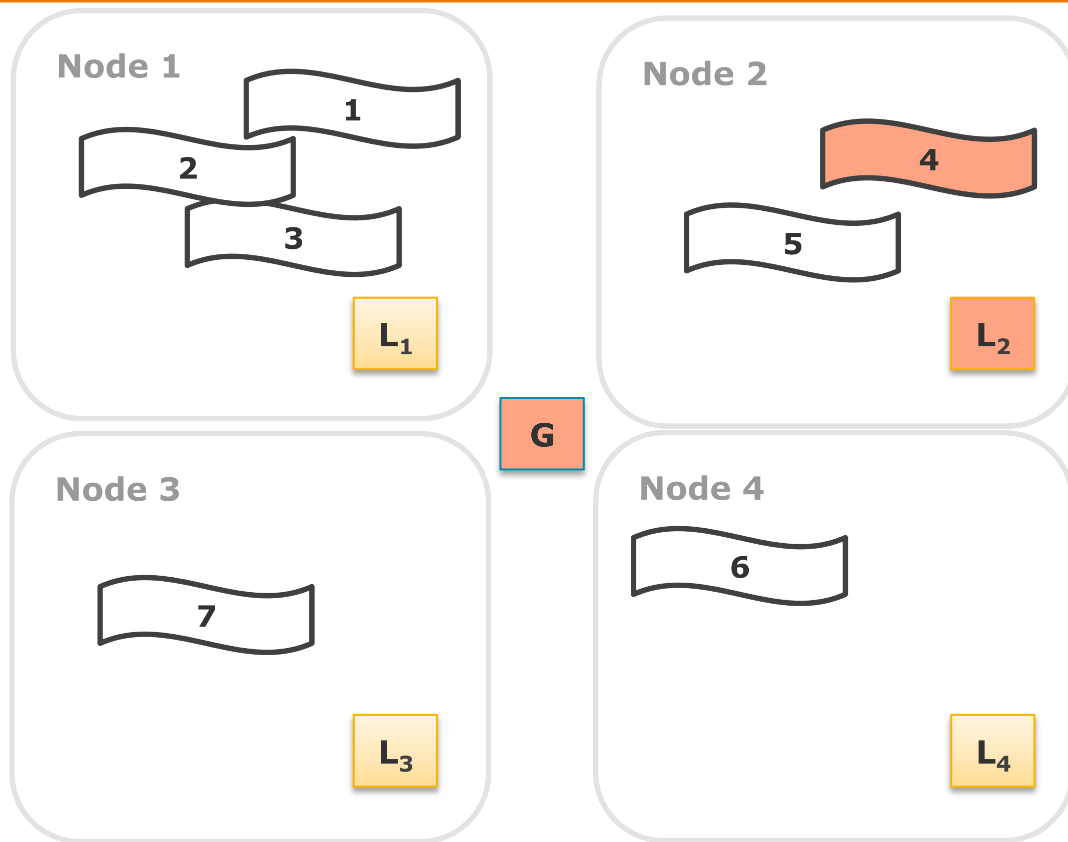


Cohort Locks
Example

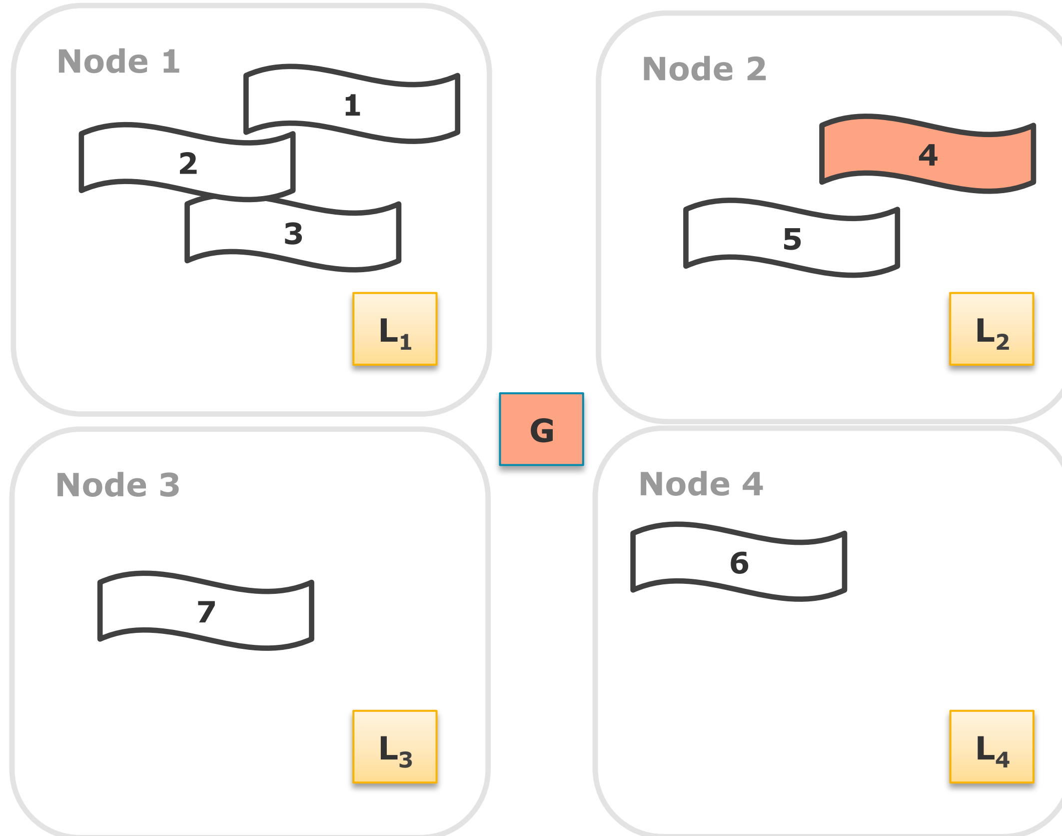


Hasso Plattner Institut

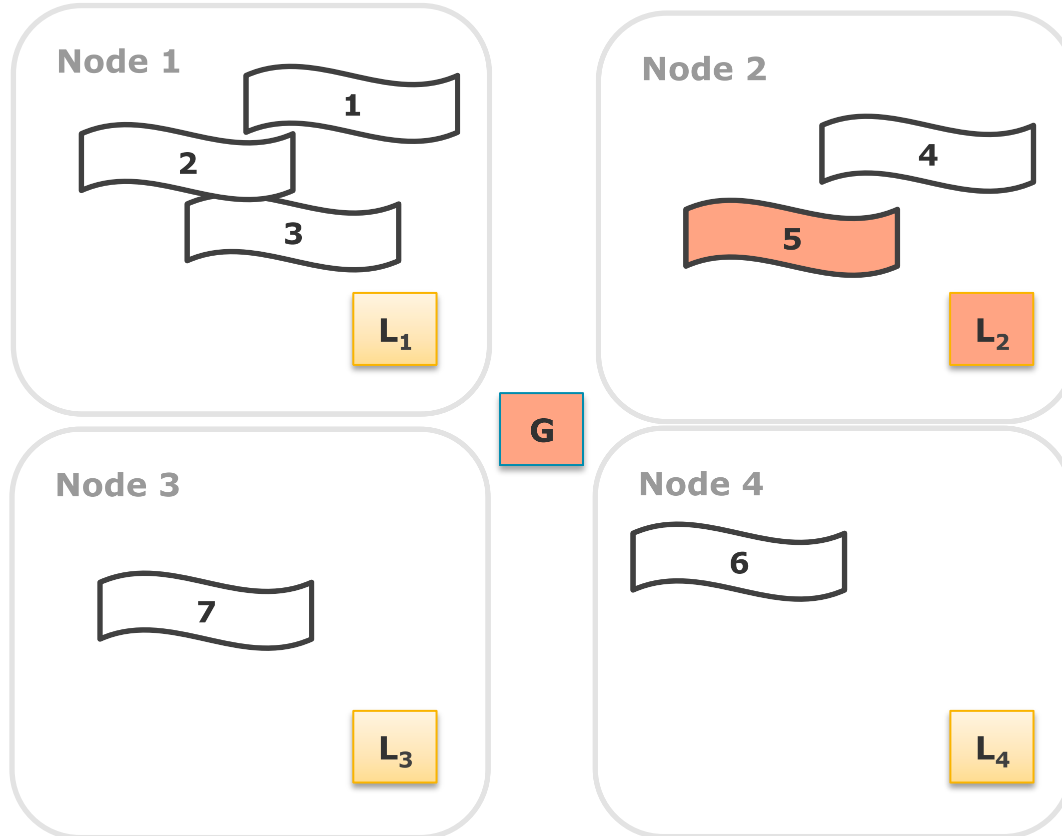
Cohort Locks
Example



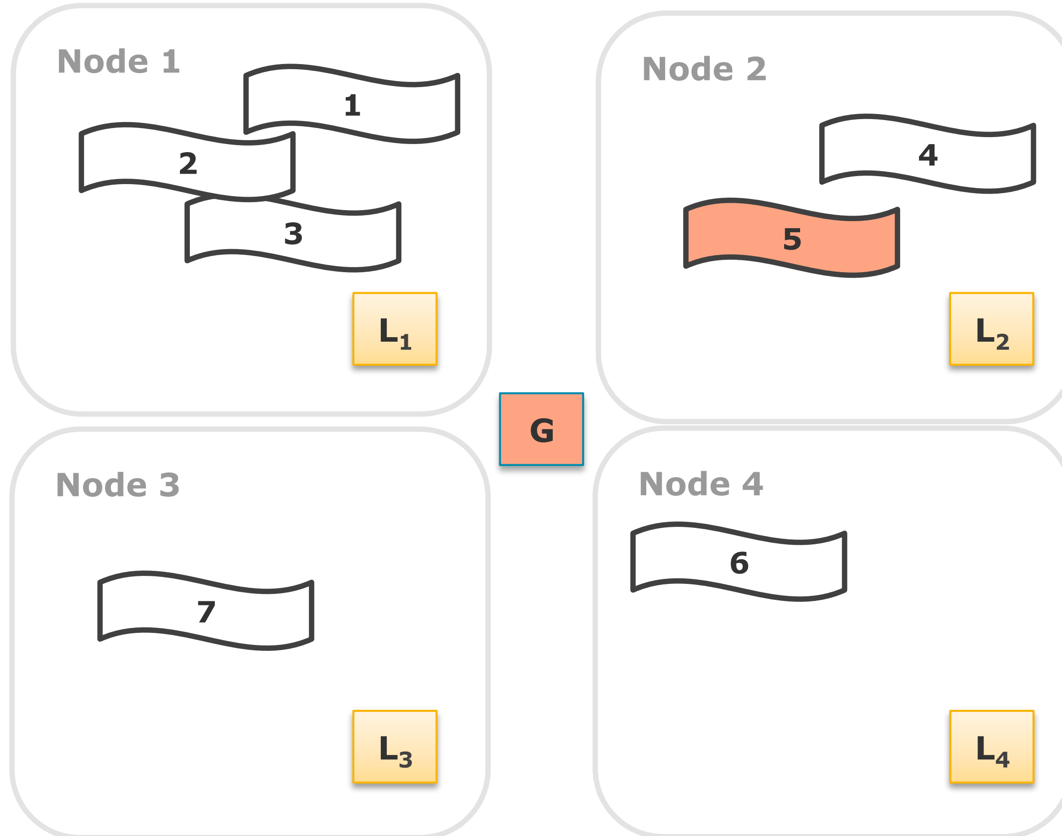
Cohort Locks
Example



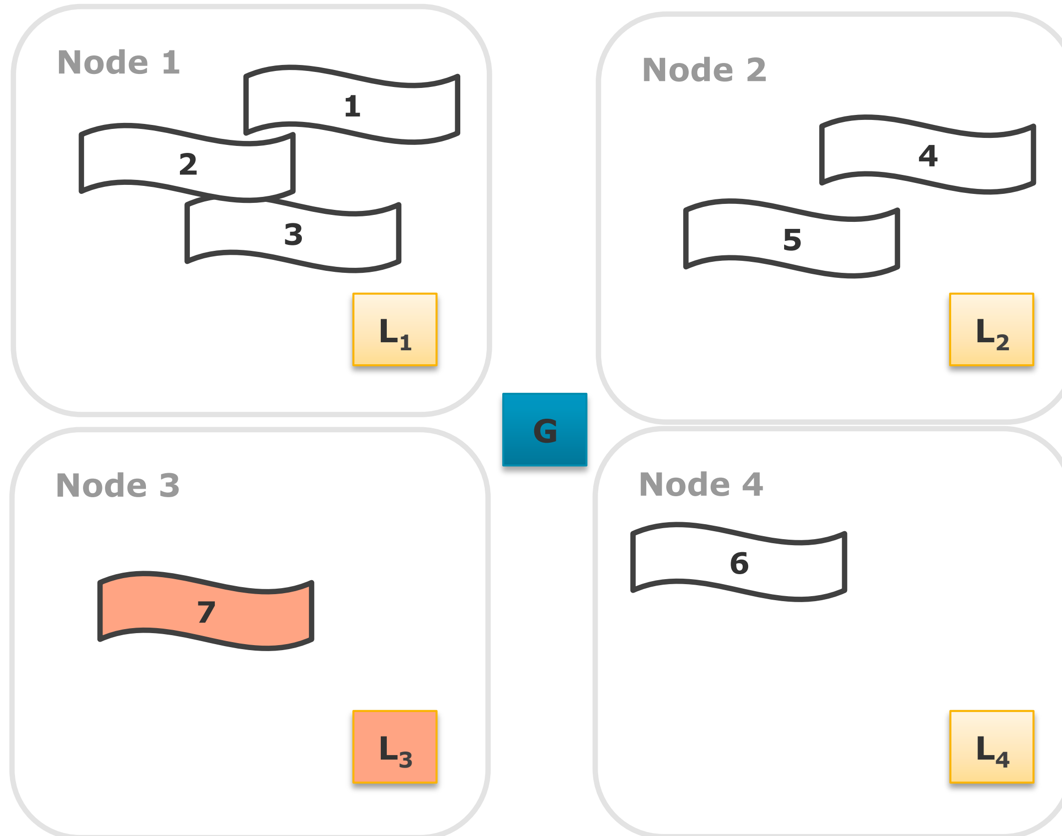
Cohort Locks
Example



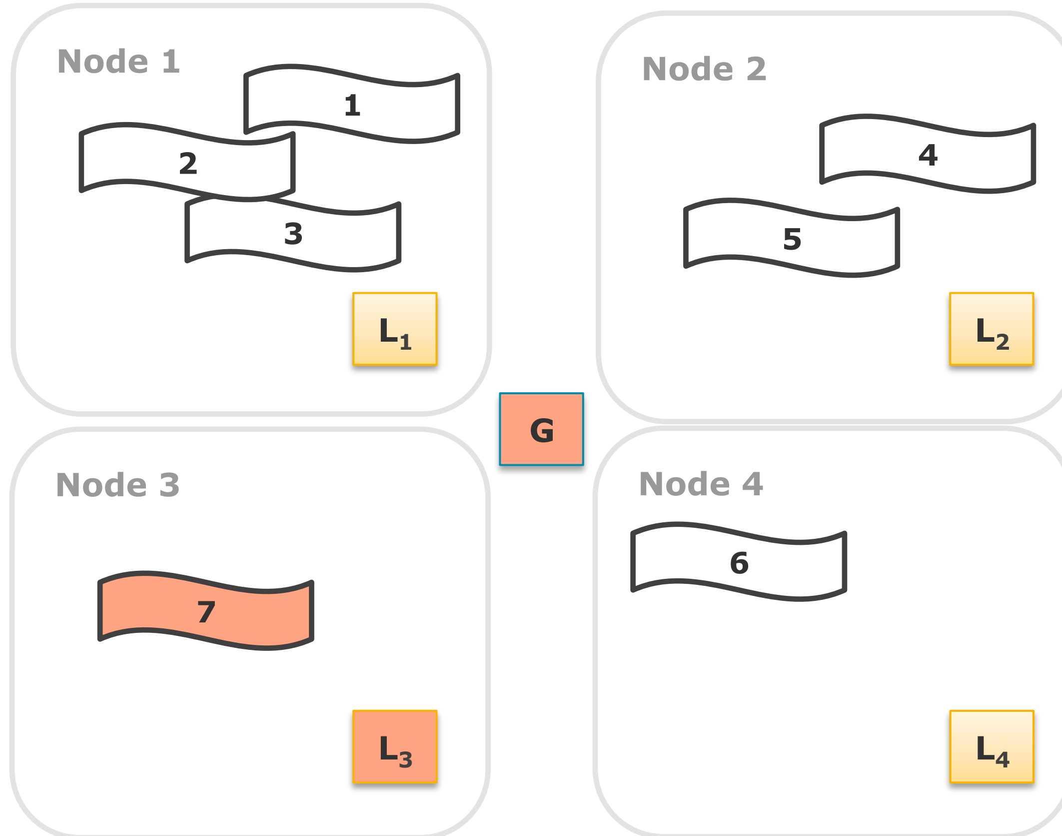
Cohort Locks
Example



Cohort Locks
Example



Cohort Locks
Example



Definition: Cohort Locks

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

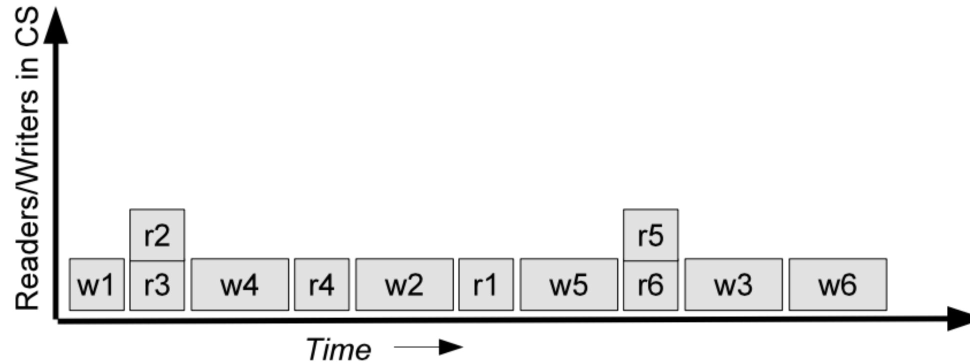
Hasso Plattner Institut

NUMA-aware Reader-Writer Locks

Lock Design Goal

- 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

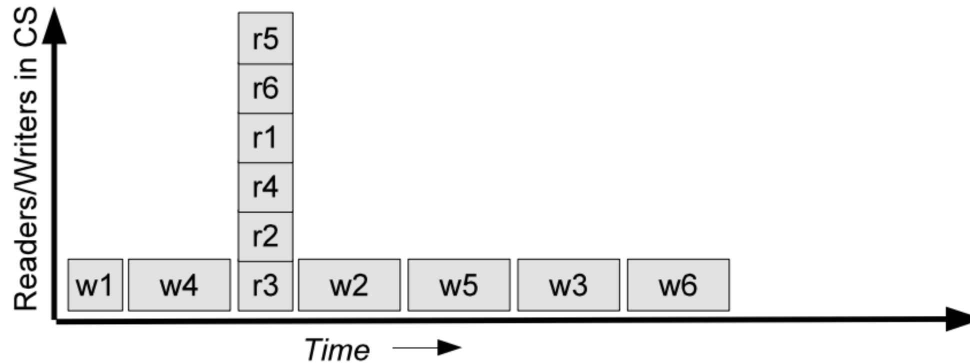
Lock Design Goal II



(a) Naïve reader-writer lock schedule

Source: [1]

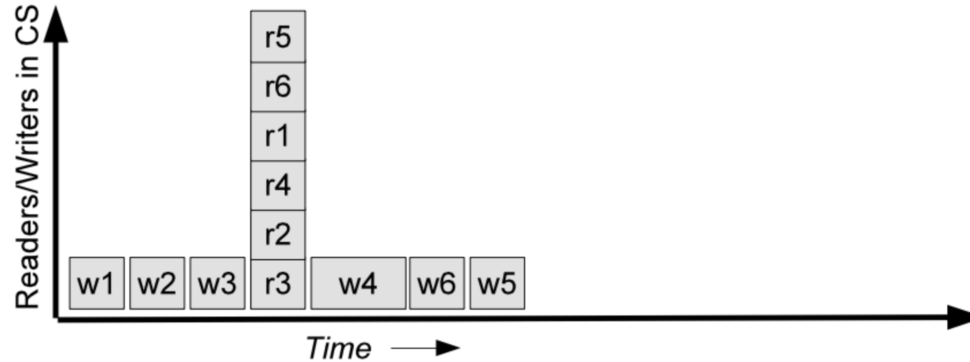
Lock Design Goal II



(b) Lock schedule with aggressive reader batching

Source: [1]

Lock Design Goal II



(c) Lock schedule with aggressive reader and writer batching

Source: [1]

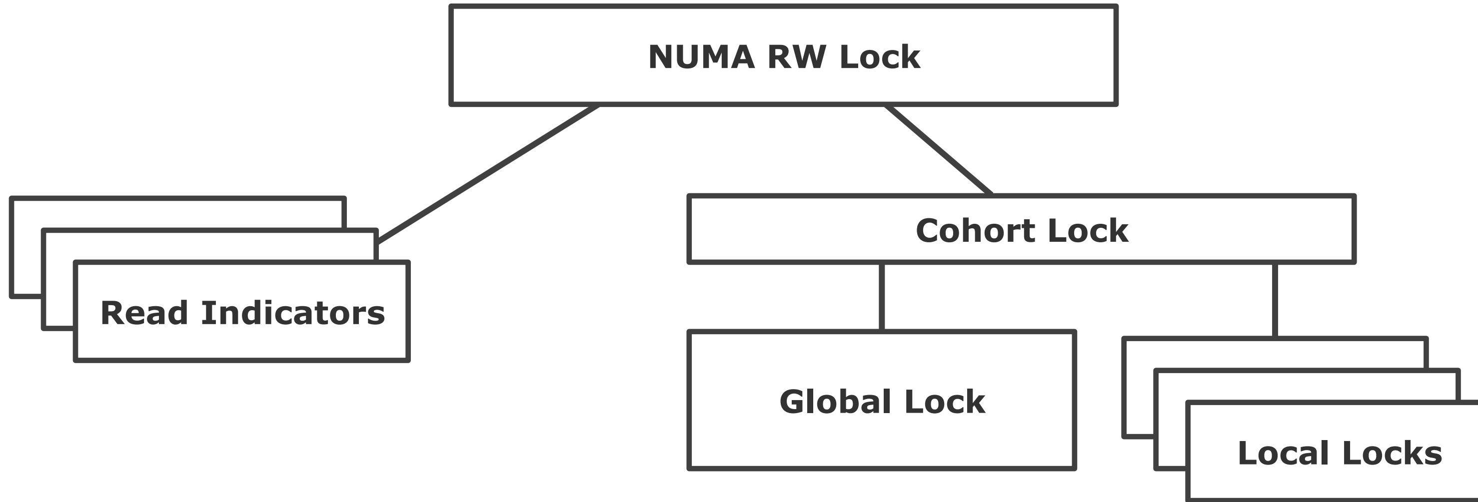
NUMA?

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

A RW-Lock Instance



- One Read Indicator Counter per NUMA Node

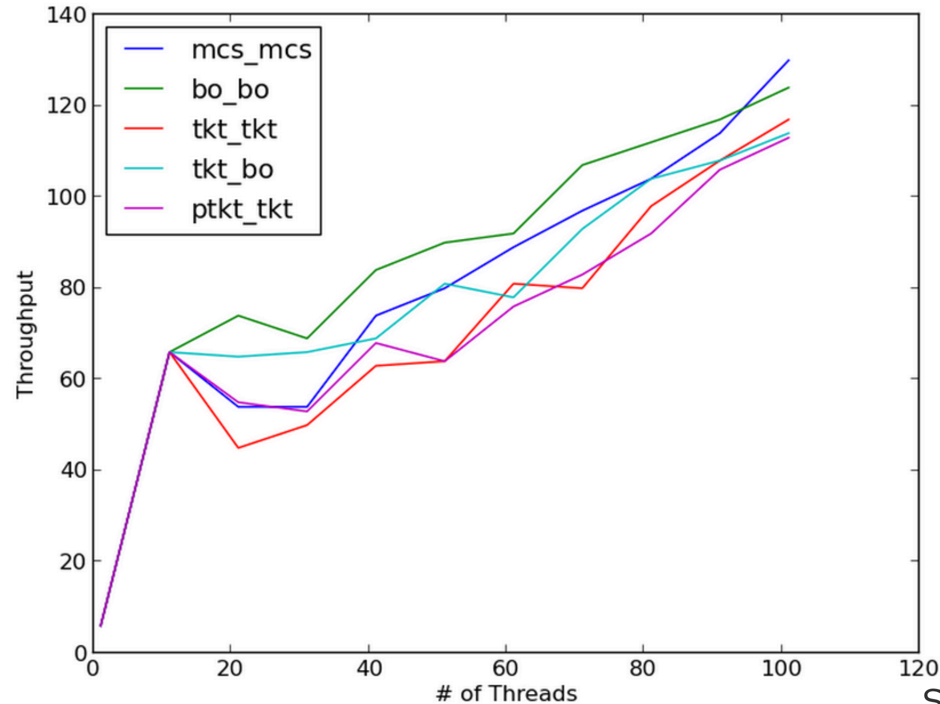
- One global lock
- One local lock per NUMA Core

HPI Hasso Plattner Institut

Benefit from Lock Generalization

- 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

Performance of Different Implementations



Source: [4]

NUMA-aware Reader-Writer Locks

Tom Herold, Marco Lamina
28.01.2015

Lock Preferences - Neutral


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

Reader Concurrency

```
7: writer:
8:       CohortLock.acquire()
9:       while NOT(ReadIndr.isEmpty())
10:              Pause
11:      <write-critical-section>
12:      CohortLock.release()
```

Wait for all readers to finish

Lock Preferences - Neutral

HPI Hasso Plattner Institut

Problems with Neutral Preference

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

Lock Preferences - Reader

HPI Hasso Plattner Institut

Assumptions

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

Benefits

- Increased scalability
- High Reader-reader concurrency

Lock Preference - Reader

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

Lock Preference - Reader

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

Writers raise Barrier if out of patience

Lock Preferences - Reader

Problems with Reader Preference

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

Lock Preferences - Writer

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

Lock Preferences – Writer

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

Lock Preferences – Writer

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

Readers raise Barrier if out of patience

Lock Preferences – Writer

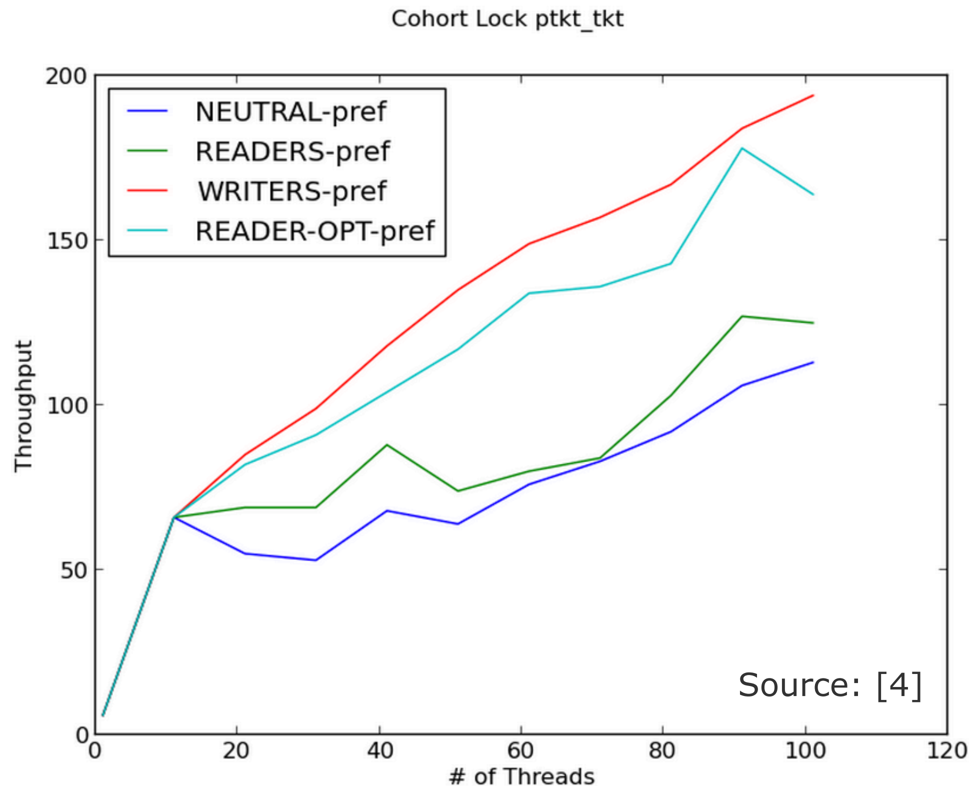
```
1:reader:
2:     bRaised = false // local flag
3:   start:
4:     ReadIndr.arrive()
5:     if CohortLock.isLocked()
6:         ReadIndr.depart()
7:         while CohortLock.isLocked()
8:              Pause
9:              if RanOutOfPatience AND ~bRaised
10:                atomic_increment(Wbarrier)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


HPI Hasso Plattner Institut

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 RW Locks: Benchmark of Different Preferences



HPI Hasso Plattner Institut

HPI Hasso Plattner Institut

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

First Implementation: Original Paper

NUMA-Aware Reader-Writer Locks

Irina Calciu
Brown University
irina@cs.brown.edu

Dave Dice
Oracle Labs
dave.dice@oracle.com

Yossi Lev
Oracle Labs
yossi.lev@oracle.com

Victor Luchangco
Oracle Labs
victor.luchangco@oracle.com

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

Nir Shavit
MIT
shanir@csail.mit.edu

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

Source: [1]

First Implementation:
Original Paper

NUMA-Aware Reader-Writer Locks

Irina Calciu
Brown University
irina@cs.brown.edu

Dave Dice
Oracle Labs
dave.dice@oracle.com

Yossi Lev
Oracle Labs
yossi.lev@oracle.com

Victor Luchangco
Oracle Labs
victor.luchangco@oracle.com

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

Nir Shavit
MIT
shanir@csail.mit.edu

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

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_rw_wrlock(), pthread_rw_rdlock()*
  - **Delegate** calls to NUMA-aware RW lock implementation from "azu-labs.com" (based on paper) [4]

Second Implementation:
Brown University Project Report

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

Source: [9]

Third Implementation: University of Concepción / Azu Labs

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

Source: [5]

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

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
CK_COHORT_UNLOCK(3)	release cohort lock

Open Source / Backtrace.io
http://concurrencykit.org/
Samy Al Bahra + Contributers

Source: [6]

Fourth Implementation: Concurrency Kit

HPI Hasso Plattner Institut

- 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

Hands On

Pthread RW Locks

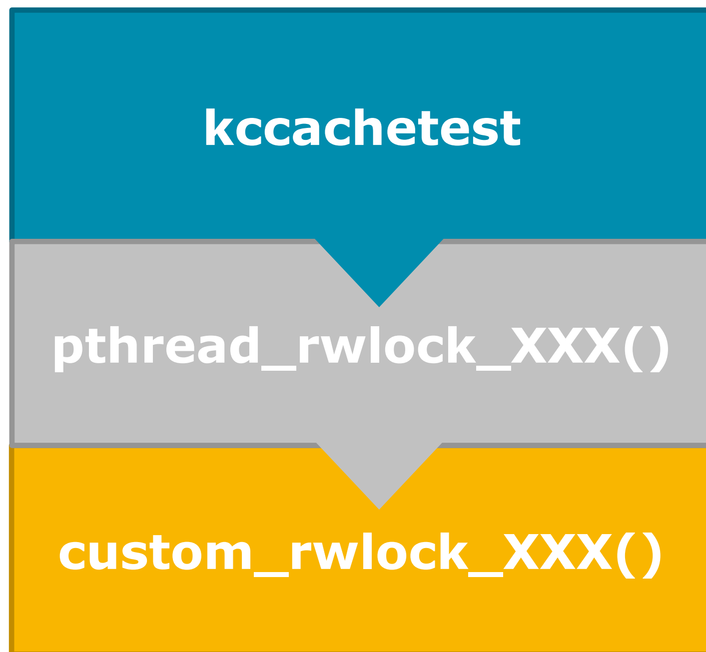
```
#include <pthread.h>

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);
```

Pthread RW Lock Proxy



- 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

HPI Hasso Plattner Institut

Sources

- [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/

Sources

- [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/

Demo Results

Hasso Plattner Institut

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
```

Demo Results

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
```

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
```

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
```