ORACLE®

NUMA-Aware Reader-Writer Locks

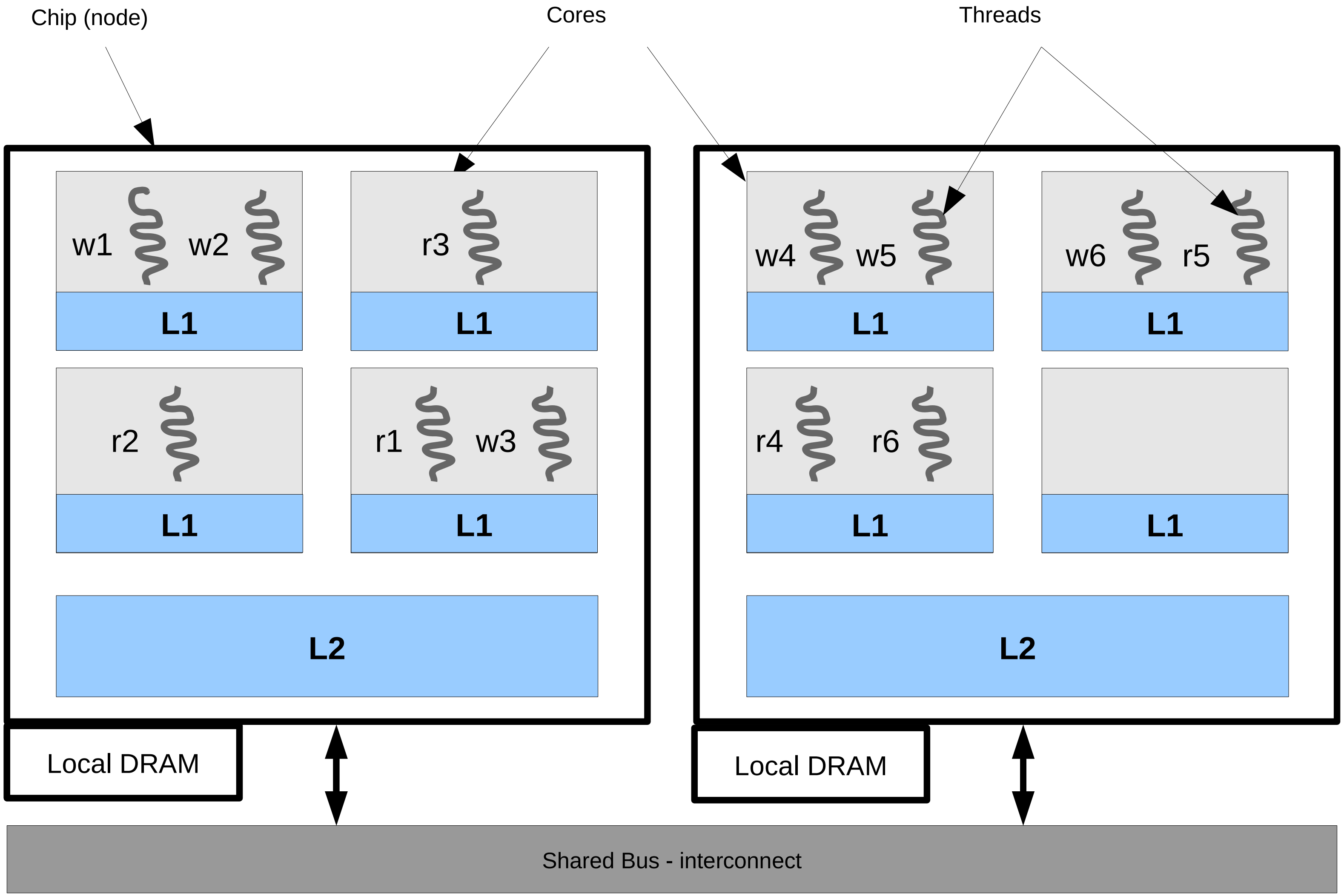
PPoPP 2013

Irina Calciu
Brown University

BROWN

Authors

- Irina Calciu @ Brown University
- Dave Dice
- Yossi Lev
- Victor Luchangco
- Virendra J. Marathe
- Nir Shavit @ MIT



Typical NUMA system

NUMA

- Interconnect is growing most slowly of all interfaces
- Critical bottleneck on large systems
- Classic NUMA programming :
  - Avoid cold & capacity misses served from remote node
  - Concern : home node of memory vs node of thread accessing that memory

ORACLE

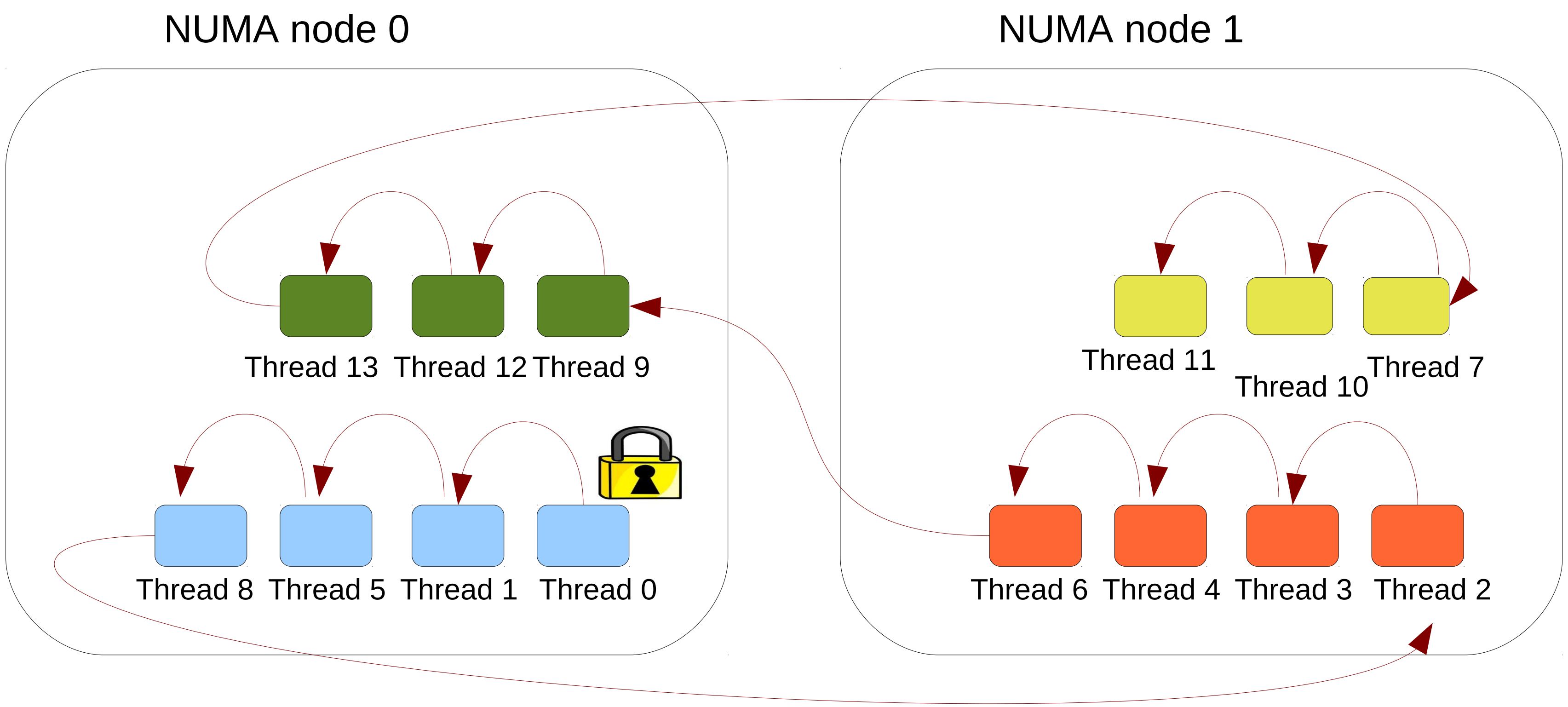
NUMA

- Our concern : contended locks
  - Coherence misses & communication
  - Minimize cache-to-cache coherence transfers
  - Location of thread accessing a line
    - Caches that have that line & states

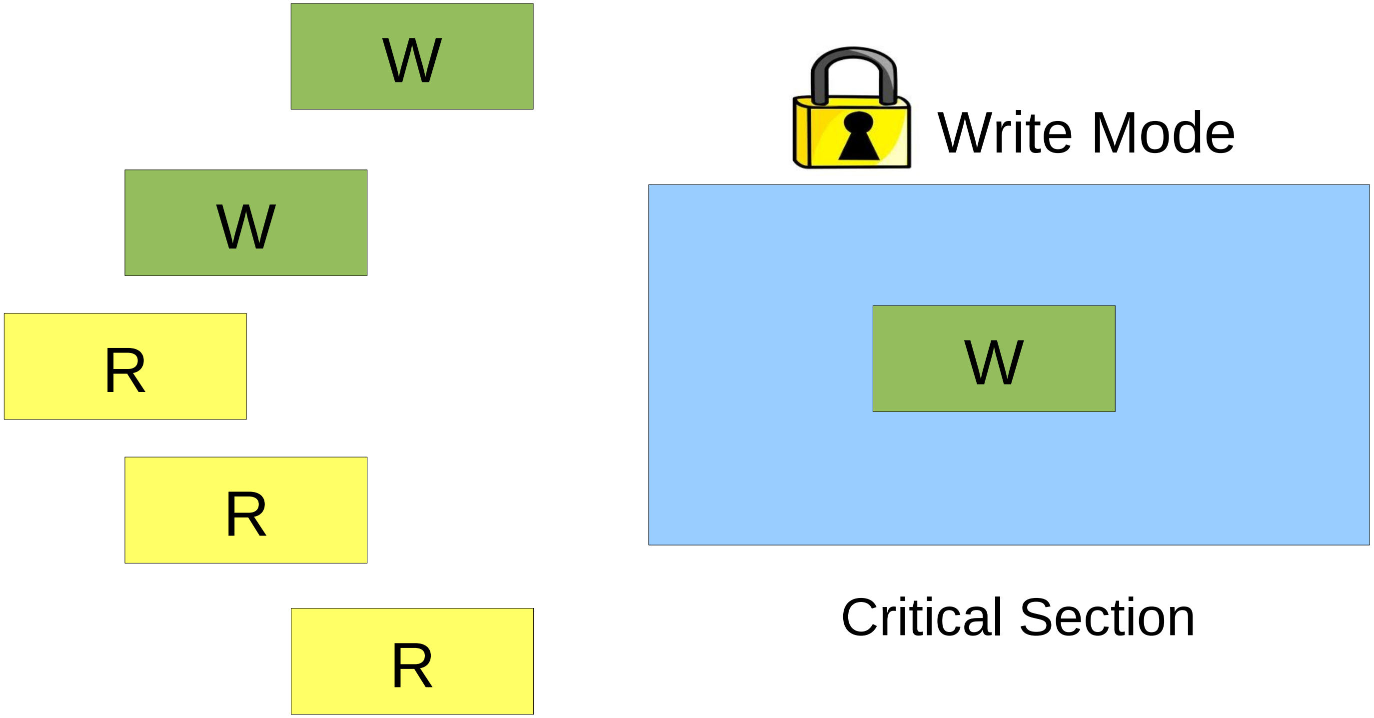
ORACLE

Background : cohort locks

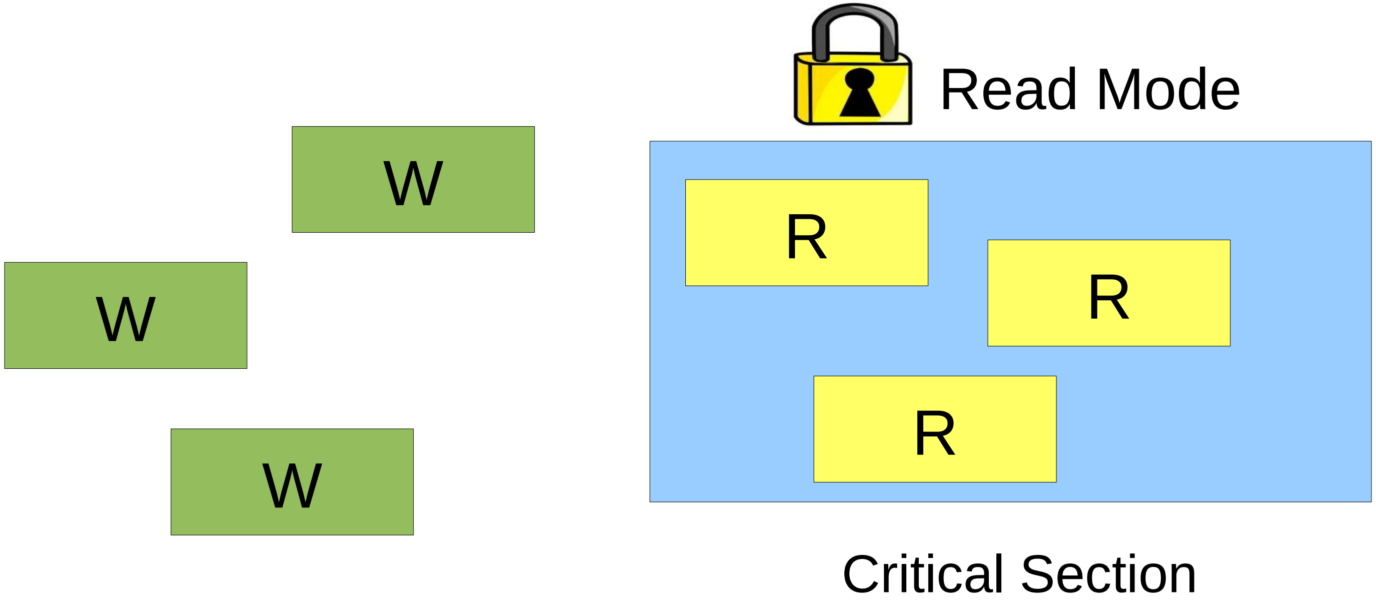
- Non-FIFO : trade short-term fairness for aggregate throughput
- [PPoPP 2012]



Reader-Writer Locks



Reader-Writer Locks

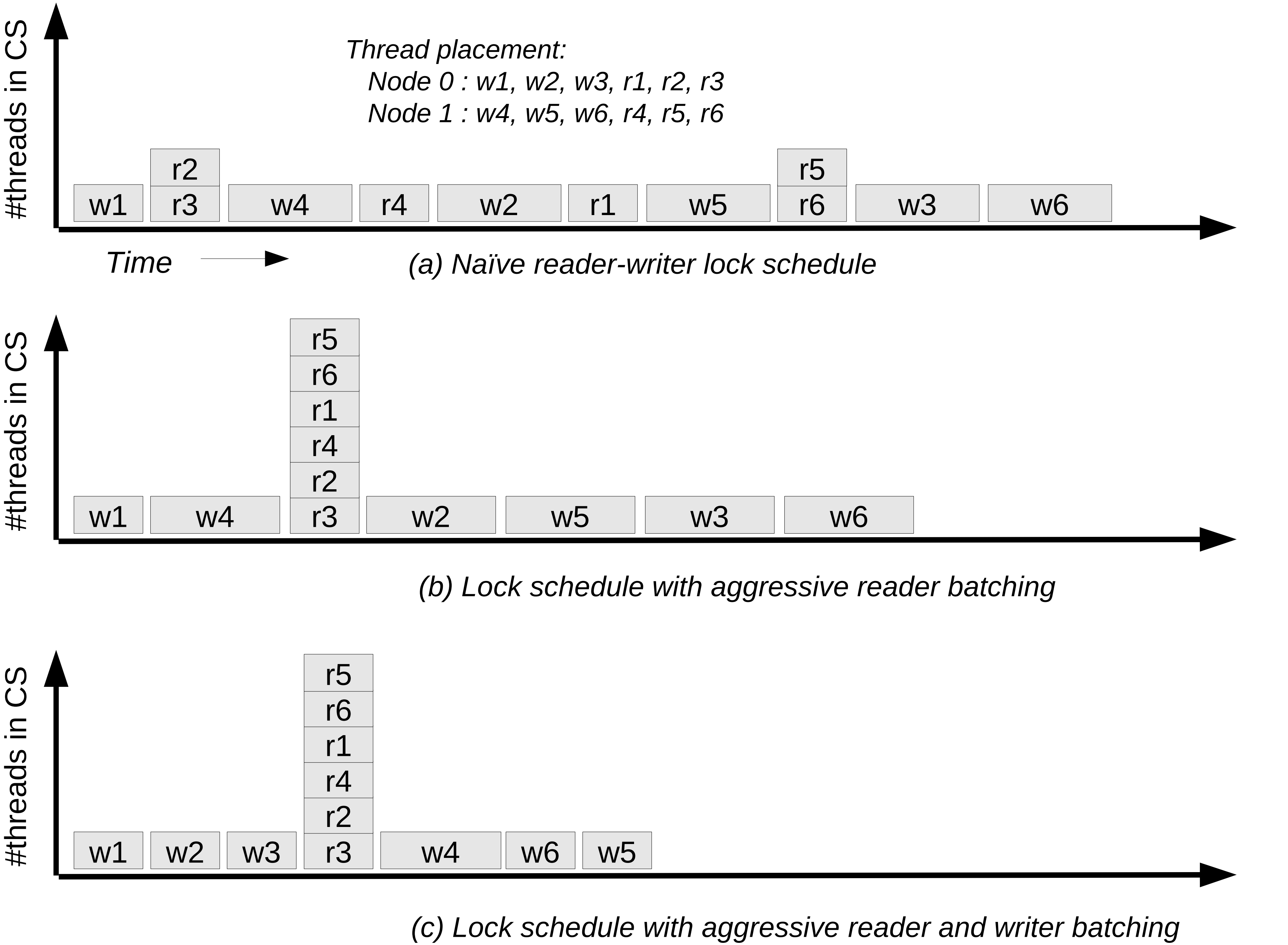


Reader-Writer Locks

- Maximize size of R-groups
- Minimize R-W alternation
- Used in : databases, operating systems, STM
- Alternative roles : Stop-the-world Garbage collection
  - “read” confers RW access to heap
  - “write” confers ability of collector to move

Admission Policy - Variations

- Include Read/Write in scheduling decision
- Reader-preference
- Writer-preference
- FIFO : R-groups form from ambient order


#threads in CS
Thread placement:
Node 0 : w1, w2, w3, r1, r2, r3
Node 1 : w4, w5, w6, r4, r5, r6
w1
r2
r3
w4
r4
w2
r1
w5
r5
r6
w3
w6
Time
(a) Naïve reader-writer lock schedule
#threads in CS
w1
w4
r5
r6
r1
r4
r2
r3
w2
w5
w3
w6
(b) Lock schedule with aggressive reader batching
#threads in CS
w1
w2
w3
r5
r6
r1
r4
r2
r3
w4
w6
w5
(c) Lock schedule with aggressive reader and writer batching

Problems with existing RW locks

- Path length
  - Longer relative to a mutex
- Lock meta-data accesses
  - Centralized : NUMA-oblivious
  - Coherency communication costs
- Simple mutex often yields better results
  - For relatively short critical sections
  - Despite lack of R-R parallelism
- RW lock : benefits of R-R parallelism don't overcome additional overhead

ORACLE

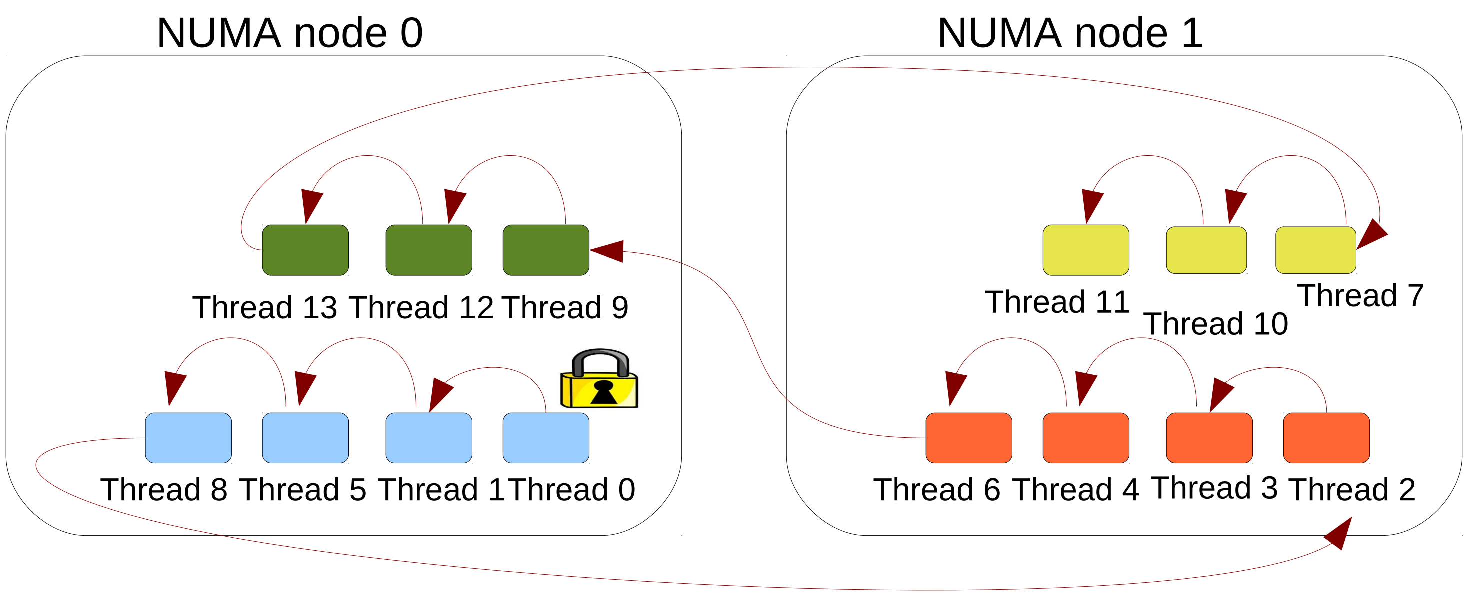
Our design

- Trade short-term fairness for throughput
  - Similar to Cohort Locks

- Presume reads dominate
  - Shift burden of work from reader lock path to writer path

Our design: Writers

- **Single centralized write lock (WL)**
  - Abstraction : Lock; Unlock; IsLocked
  - W-vs-W conflicts
  - Best implementation

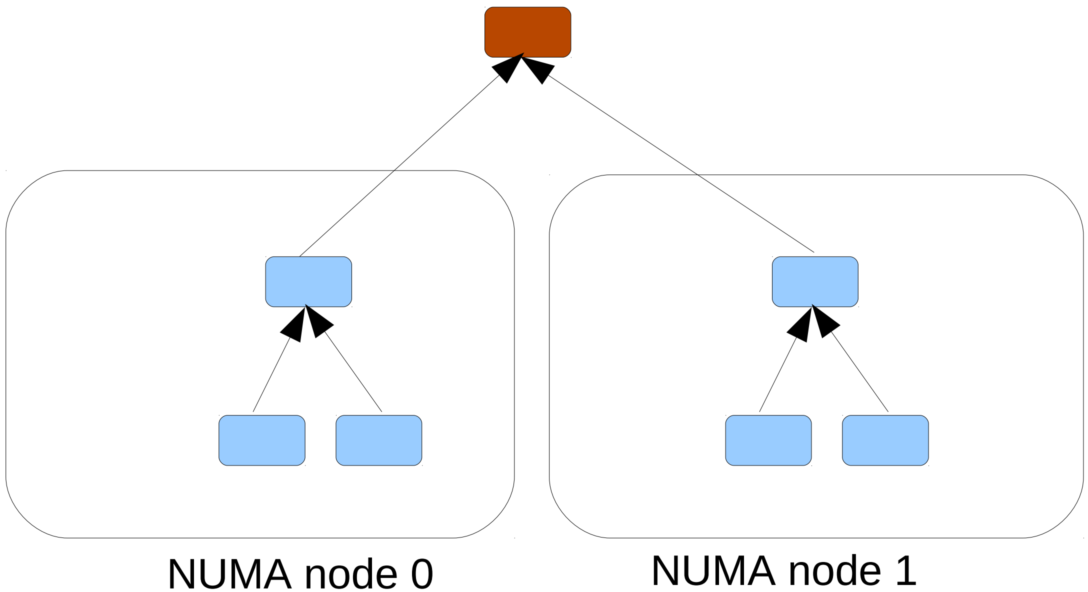


Our design: Readers

- **Reader indicators (RI)**
  - Publish intent to read to writers
  - Abstraction : Arrive; Depart; IsZero
  - Conceptually : counter

Reader Indicators

- **Global counter**
  - Atomic increment and decrement
  - OK uniprocessor, horrible on NUMA
- **SNZI**



Reader Indicators

- **Per-node distributed counters :**
  - Local writes only

- **Per-node pairs : ingress and egress fields**
  - Arrive : increment ingress
  - Depart : increment egress
  - Reduces intra-node fetch-and-add contention
  - *Preferred implementation*

Our design: Readers and Writers

- IsLocked and IsZero :
  - Detect and resolve R-vs-W conflicts

Reader:

```
start:
    RI.Arrive()
    // Check for writers
    if WL.isLocked():
        RI.Depart()
        while WL.isLocked():
            Pause()
        goto start
<read-critical-section>
RI.Depart()
```

Writer:

```
WL.Acquire()
// Check for readers
while not RI.isZero():
    Pause()
<write-critical-section>
WL.release()
```

ORACLE

Impatience (I)

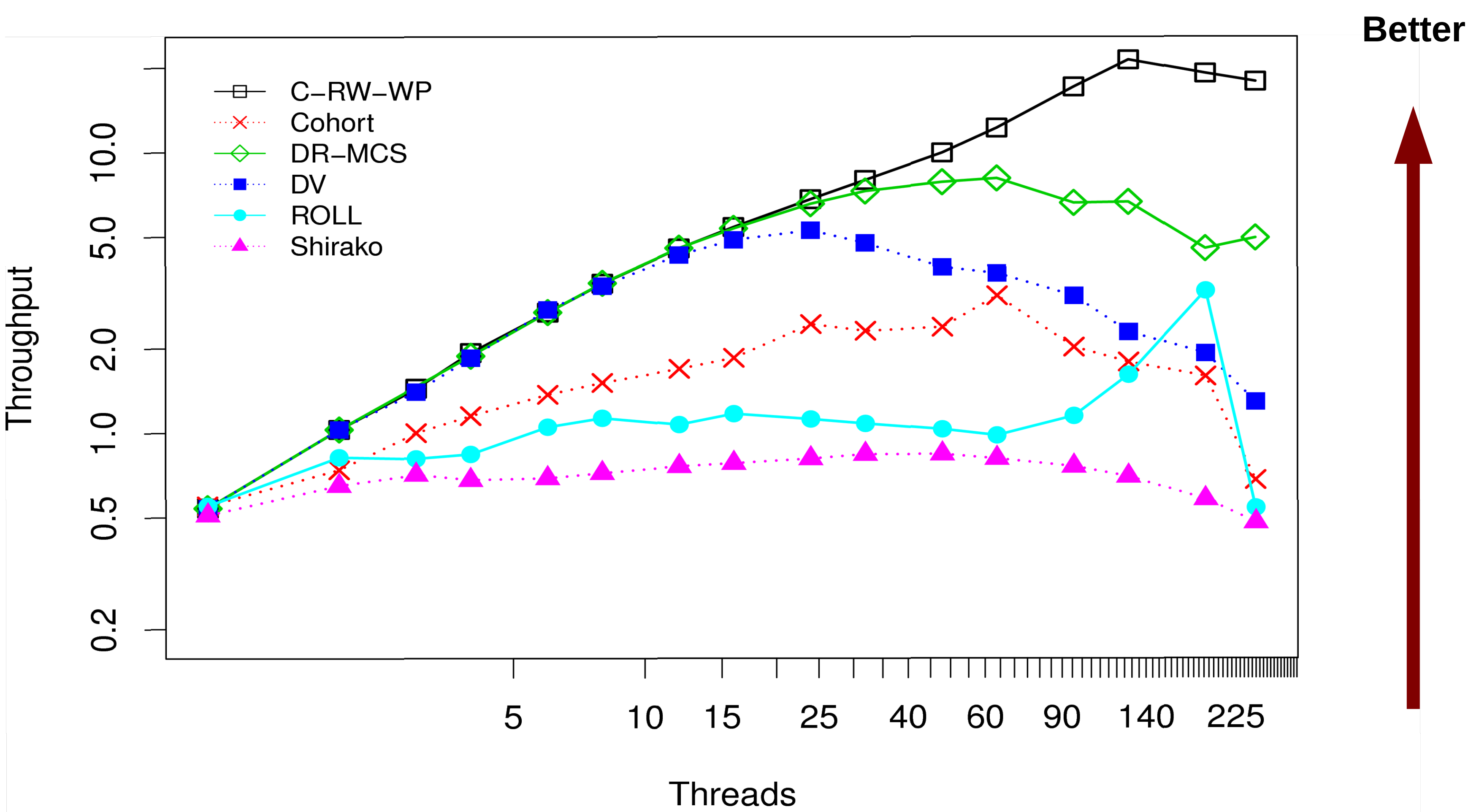
- Adaptive RP-WP policy
- Start with writer-preference lock – C-RW-WP
- Writers acquire WL and wait for RI to reach 0
- Readers increment RI and check WL
  - If locked, decrement and defer to writers

Impatience (II)

- Readers initially patient but can become impatient
  - block inflow of newly arriving writers – erect barrier
  - avoids reader starvation
- Bounded bypass : writers can bypass patient readers

Impatience (III)

- Effectively : toggling preference policy to avoid starvation
- Promotes large R-groups
- Long chains of writers leverage cohort locks
- Adaptive admission policy



98% reads, 2% writes

Observations

- Distributed RIs beat SNZI
  - Flat array of RI better, at least for 4 or 8 node systems
  - SNZI expected to win at some N
- NUMA-like behavior on-chip
  - Core-local L2 caches
  - Treat each core as if a NUMA node
- Fixed thread roles vs variable
  - Variable : models use of thread pools
  - Fixed : our lock family still yields best results

Summary (I)

- Family NUMA-friendly RW locks
- Trivial to substitute RI or WL implementations
- High aggregate throughput
- Fair over long-term for : threads; R/W roles; NUMA nodes

Summary (II)

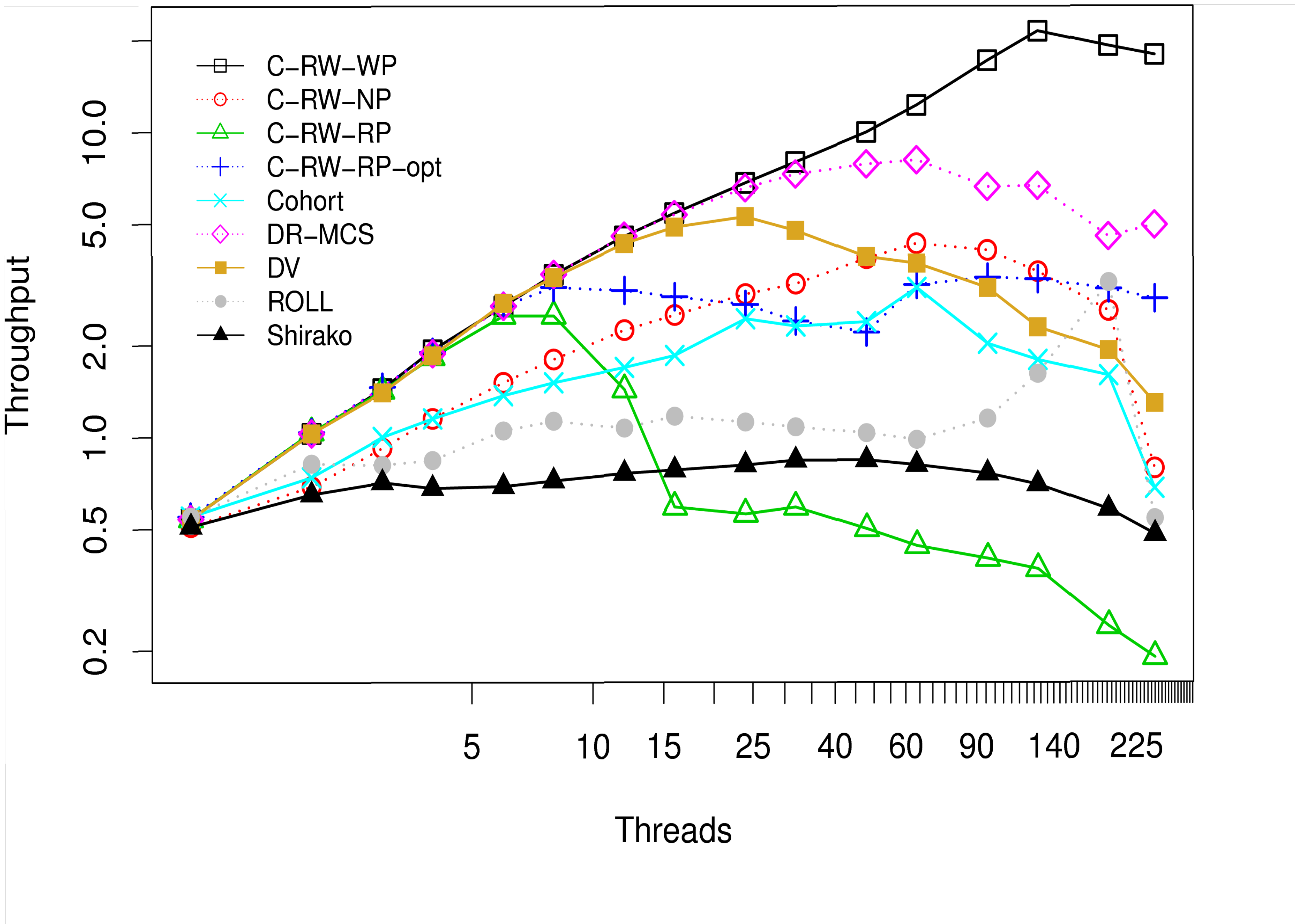
- Long critical sections
  - Quality of scheduling is critical
  - R-group formation
- Short critical sections
  - Lock overheads can dominate
  - Consider a NUMA-friendly mutex
- Fixed preference policies can be problematic
  - Adaptive to avoid starvation
  - Non-preferred role can become impatient

ORACLE

Thank you!

- http://cs.brown.edu/~irina
- http://blogs.oracle.com/dave

ORACLE



98% reads, 2% writes