Scalable NUMA-aware Blocking Synchronization Primitives

Sanidhya Kashyap, Changwoo Min, Taesoo Kim

Georgia Institute of Technology

The rise of big NUMA machines

Press Release: November 22, 2011
Topics: Converged Infrastructure, Instant-On Enterprise

Share Print

HP to Transform Server Market with Single Platform for Mission-critical Computing

Expanded HP Converged Infrastructure delivers industry-leading choice, investment protection

PALO ALTO, Calif. -- HP today announced "Odyssey," a project to redefine the future of mission-critical computing with a development roadmap that will unify UNIX® and x86 server architectures to bring industry-leading availability, increased performance and uncompromising client choice to a single platform.

Organizations are challenged with increasingly stringent service-level agreements for their most demanding workloads, along with the pressure to be more efficient with their IT budgets and resources. They need the availability and resilience of UNIX-based platforms along with the familiarity and cost-efficiency of industry-standard platforms.

Using advanced technology across a common, modular HP BladeSystem architecture, HP is developing platforms to enable clients to choose the best environment aligned to their organizations' needs without compromise, helping ensure investment protection for the long term.

HP's new development roadmap includes ongoing innovations to HP Integrity servers, HP NonStop systems and the HP-UX and OpenVMS operating systems. The roadmap also includes delivering blades with Intel® Xeon® processors for the HP Superdome 2 enclosure (code name "DragonHawk") and the scalable c-Class blade enclosures (code named "HydraLynx"), while fortifying Windows® and Linux environments with innovations from HP-UX within the next two years.

With the availability of "DragonHawk," clients will be able to run mission-critical workloads on HP-UX on Intel Itanium®-based blades while simultaneously running workloads on Microsoft Windows or Red Hat Enterprise Linux on Intel Xeon-based blades in the same Superdome 2 enclosure.

"Clients have been asking us to expand the mission-critical experience that is delivered today with HP-UX on Integrity to an x86-based infrastructure," said Martin Fink, senior vice president and general manager, Business Critical Systems, HP. "HP plans to transform the server landscape for mission-critical computing by using the flexibility of HP BladeSystem and bringing

Find HP News

Related media contacts

Terri Molini, HP
terri.molini@hp.com

Lee Figora, Burson-Marsteller for HP
lee.figora@bm.com

View All Media Contacts

Resources

Newsroom
About us
Leadership
Investor relations

Social media

The rise of big NUMA machines

Press Release: November 22, 2011
Topics: Converged Infrastructure, Instant-On Enterprise

Share Print

Find HP News

HP to Transform Server Market with Single Platform for Mission-critical Computing

Expanded HP Converged Infrastructure delivers industry-leading choice, investment protection

Related media contacts

Terri Molini, HP
terri.molini@hp.com

PALO ALTO, Calif
critical computir
architectures to
uncompromising

Organizations ar
most demanding
and resources. T
familiarity and c

Using advanced
developing platf
organizations' n
term.

HP's new develc
NonStop system
delivering blade
"DragonHawk")
fortifying Windo
years.

With the availab
HP-UX on Intel I
Windows or Red
enclosure.

"Clients have be
with HP-UX on I
and general mar

Press Release

Oracle Announces Breakthrough Processor and Systems Design with SPARC M7

Dramatic Advancements in Memory Protection, Encryption Acceleration, and In-memory Database Processing Deliver End-to-End Security and Efficiency for Oracle Engineered Systems and Servers

ORACLE OPENWORLD, SAN FRANCISCO —Oct 26, 2015

Oracle today introduced an all-new family of SPARC systems built on the revolutionary 32-core, 256-thread SPARC M7 microprocessor. The systems feature Security in Silicon for advanced intrusion protection and encryption; SQL in Silicon that delivers unparalleled database efficiency; and world record performance spanning enterprise, big data, and cloud applications.

The new SPARC M7 processor-based systems, including the Oracle SuperCluster M7 engineered system and SPARC T7 and M7 servers, are designed to seamlessly integrate with existing infrastructure and include fully integrated virtualization and management for cloud. All existing commercial and custom applications will run on

The rise of big NUMA machines

Press Release: November 22, 2011
Topics: Converged Infrastructure, Instant-On Enterprise

Share Print

Find HP News

HP to Transform Server Market with Single Platform for Mission-critical Computing

Expanded HP Converged Infrastructure delivers industry-leading choice, investment protection

Related media contacts

Terri Molini, HP
terri.molini@hp.com

PALO ALTO, Calif
critical computi
architectures to
uncompromising

Organizations a
most demandin
and resources. T
familiarity and c

Using advanced
developing plat
organizations' n
term.

HP's new develc
NonStop system
delivering blade
"DragonHawk")
fortifying Windo
years.

With the availab
HP-UX on Intel I
Windows or Red
enclosure.

"Clients have be
with HP-UX on I
and general ma
landscape for m

Press Release

Oracle Announces Breakthrough Processor and Systems Design with SPARC M7

Dramatic Adv
Acceleration,
Security and

ORACLE OPENWO

Oracle today introdu
SPARC M7 micropro
encryption; SQL in S
spanning enterprise,

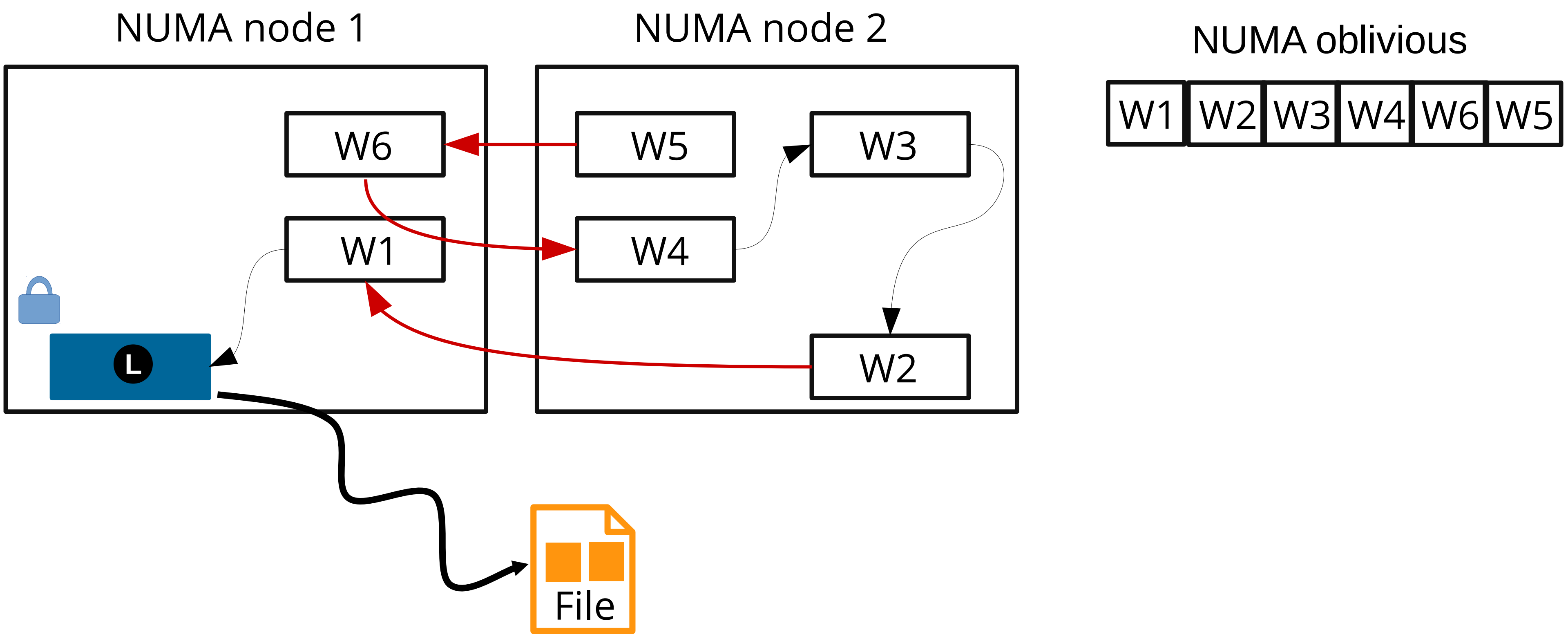
The new SPARC M7
SPARC T7 and M7 s
integrated virtualizati

Designed for large-scale, in-memory applications in the cloud

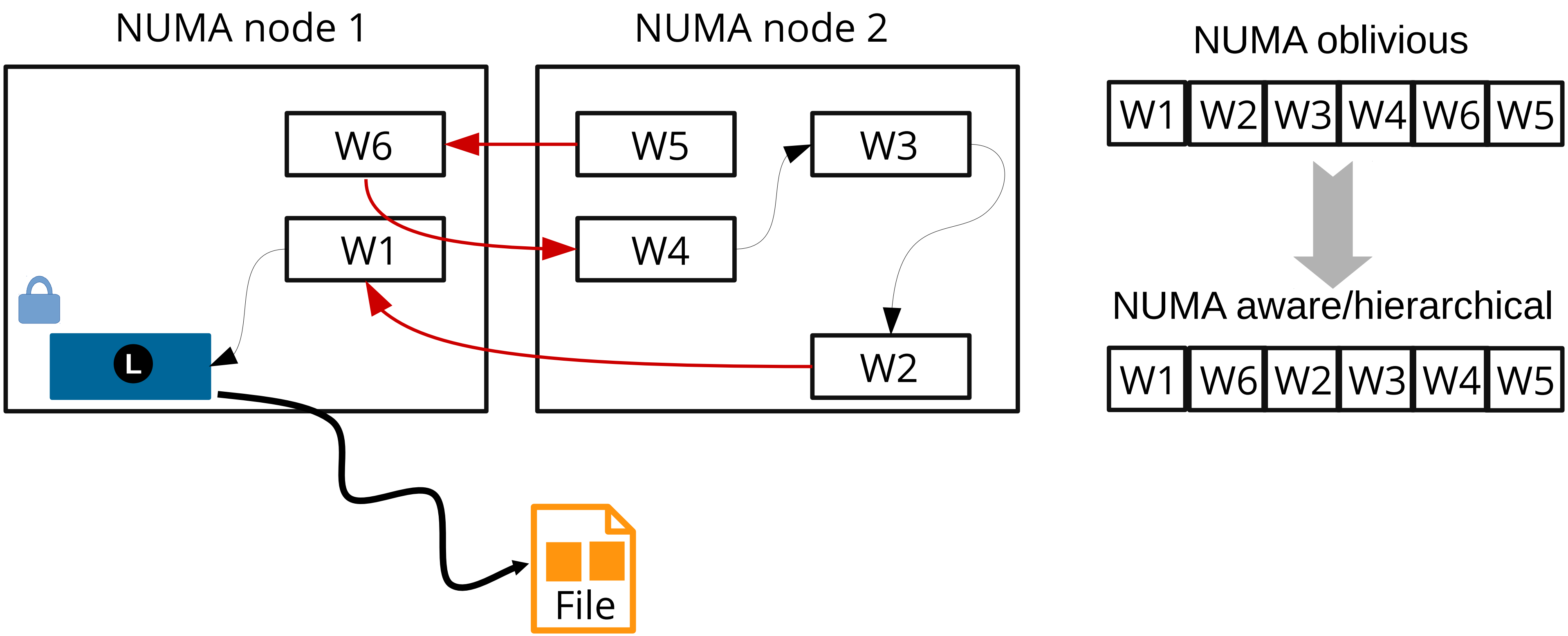
X1 Instances are a new addition to the Amazon EC2 memory-optimized instance family and are designed for running large-scale, in-memory applications and in-memory databases in the AWS cloud. X1 instances offer 1,952 GiB of DDR4 based memory, 8x the memory offered by any other Amazon EC2 instance. Each X1 instance is powered by four Intel® Xeon® E7 8880 v3 (Haswell) processors and offers 128 vCPUs.

Compared to other EC2 instances, X1 instances have the lowest price

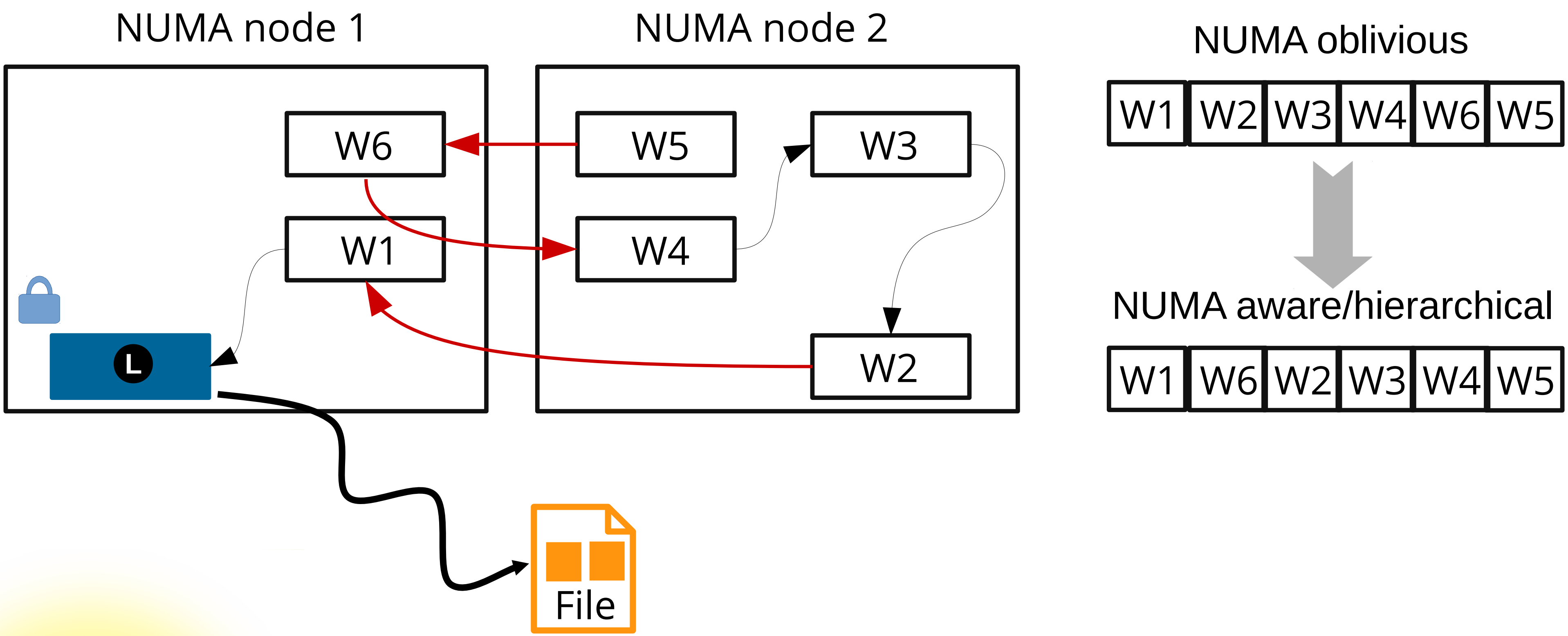
Importance of NUMA awareness



Importance of NUMA awareness



Importance of NUMA awareness



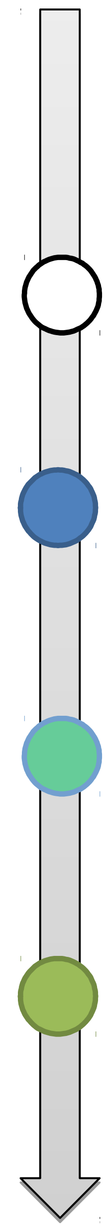
Idea:

Make synchronization primitives NUMA aware!

Lock's research efforts and their use

Lock's research efforts

- Dekker's algorithm (1962)
- Semaphore (1965)
- Lamport's bakery algorithm (1974)
- Backoff lock (1989)
- Ticket lock (1991)
- MCS lock (1991)
- HBO lock (2003)
- Hierarchical lock – HCLH (2006)
- Flat combining NUMA lock (2011)
- Remote Core locking (2012)
- Cohort lock (2012)
- RW cohort lock (2013)
- Malthusian lock (2014)
- HMCS lock (2015)
- AHMCS lock(2016)



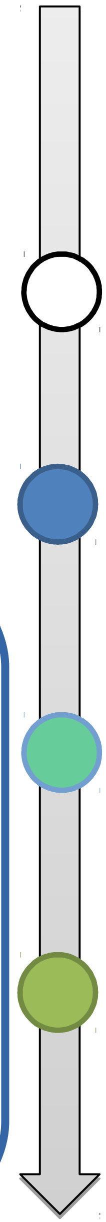
Linux kernel lock adoption / modification

Lock's research efforts and their use

Lock's research efforts

- Dekker's algorithm (1962)
- Semaphore (1965)
- Lamport's bakery algorithm (1974)
- Backoff lock (1989)
- Ticket lock (1991)
- MCS lock (1991)
- HBO lock (2003)
- Hierarchical lock – HCLH (2006)
- Flat combining NUMA lock (2011)
- Remote Core locking (2012)
- Cohort lock (2012)
- RW cohort lock (2013)
- Malthusian lock (2014)
- HMCS lock (2015)
- AHMCS lock(2016)

NUMA-aware locks



Linux kernel lock adoption / modification

Lock's research efforts and their use

Lock's research efforts

- Dekker's algorithm (1962)
- Semaphore (1965)
- Lamport's bakery algorithm (1974)
- Backoff lock (1989)
- Ticket lock (1991)
- MCS lock (1991)

NUMA-aware locks

- HBO lock (2003)
- Hierarchical lock – HCLH (2006)
- Flat combining NUMA lock (2011)
- Remote Core locking (2012)
- Cohort lock (2012)
- RW cohort lock (2013)
- Malthusian lock (2014)
- HMCS lock (2015)
- AHMCS lock(2016)

Linux kernel lock adoption / modification

1990s

Spinlock → TTAS
Semaphore → TTAS + block
Rwsem → TTAS + block

2011

Spinlock → ticket
Mutex → TTAS + block
Rwsem → TTAS + block

2014

Spinlock → ticket
Mutex → TTAS + spin + block
Rwsem → TTAS + spin + block

2016

Spinlock → qspinlock
Mutex → TTAS + spin + block
Rwsem → TTAS + spin + block

Lock's research efforts and their use

Lock's research efforts

Dekker's algorithm (1962)

Semaphore (1965)

Lamport's bakery algorithm (1974)

Linux kernel lock adoption / modification

Spinlock → TTAS
Semaphore → TTAS + block
1990s

Adopting NUMA aware locks is not easy

Remote Core locking (2012)

Cohort lock (2012)

RW cohort lock (2013)

Malthusian lock (2014)

HMCS lock (2015)

AHMCS lock(2016)

NUMA-aware locks

Mutex → TTAS + spin + block
Rwsem → TTAS + spin + block

Spinlock → qspinlock
Mutex → TTAS + spin + block
Rwsem → TTAS + spin + block
2016

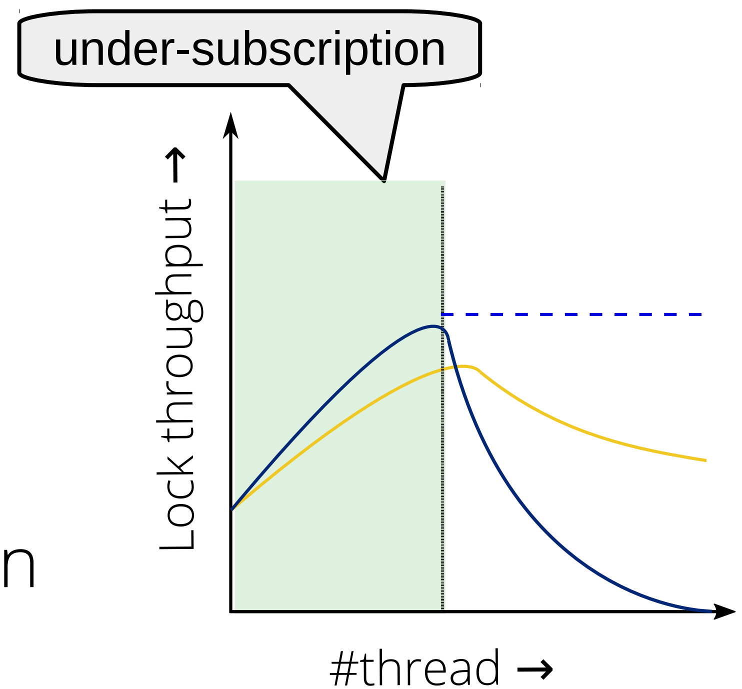
Issues with NUMA-aware primitives

- Memory footprint overhead
  - Cohort lock single instance: 1600 bytes
  - Example: 1–4 GB of lock space vs 38 MB of Linux's lock for 10 M inodes

- Does not support blocking/parking behavior

Blocking/parking approach

- Under subscription
  - #threads <= #cores
- Over subscription
  - #threads > #cores
- Spin-then-park strategy
  1) Spin for a certain duration
  2) Add to a parking list
  3) Schedule out (park/block)



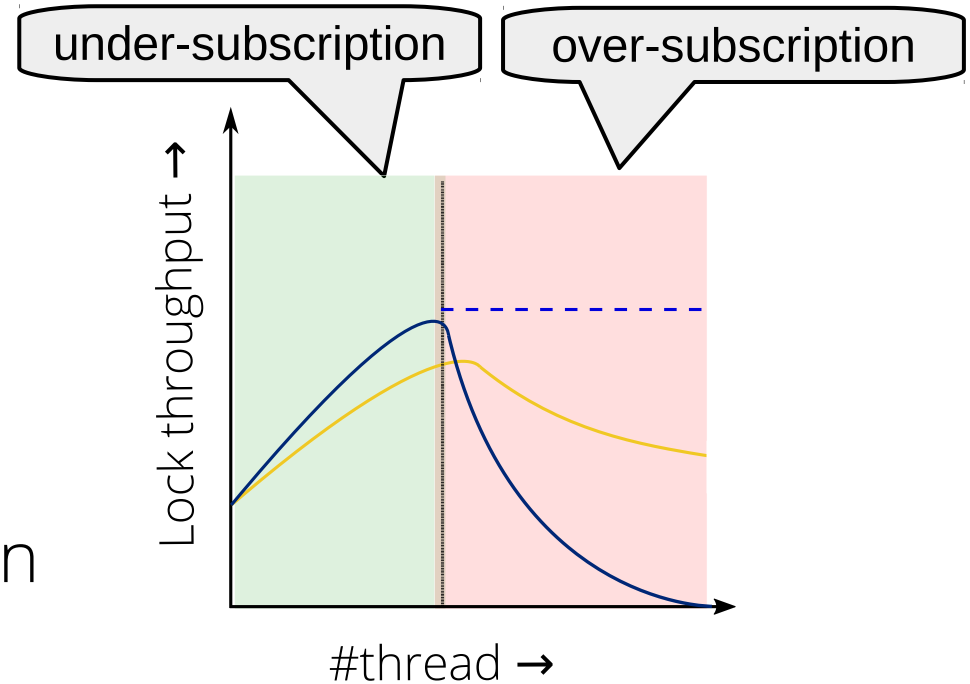
Blocking/parking approach

- Under subscription
  - #threads <= #cores
- Over subscription
  - #threads > #cores
- Spin-then-park strategy

  1) Spin for a certain duration

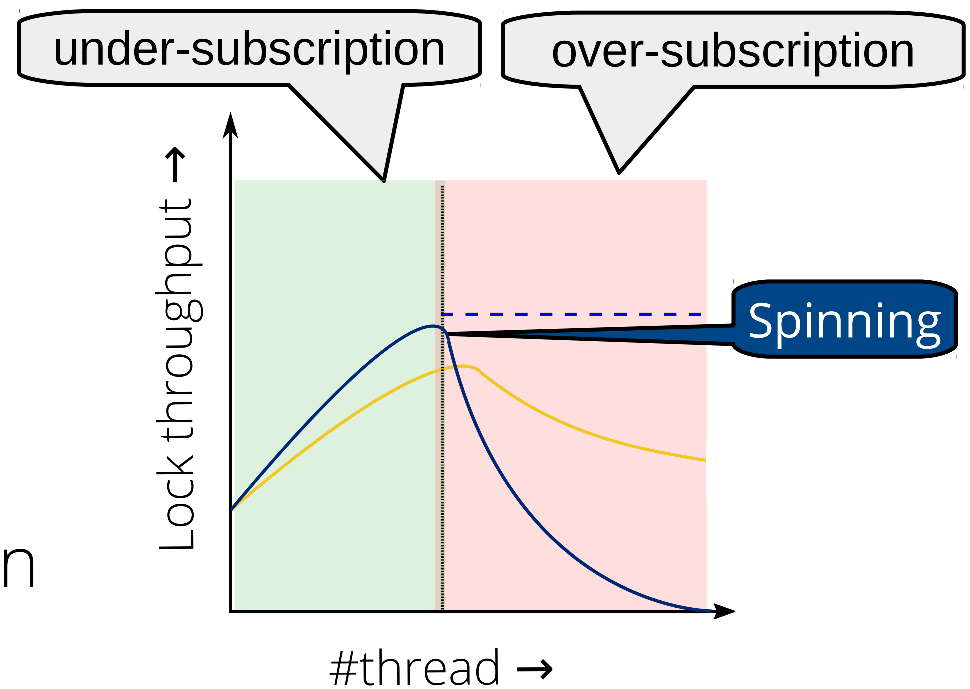
  2) Add to a parking list

  3) Schedule out (park/block)



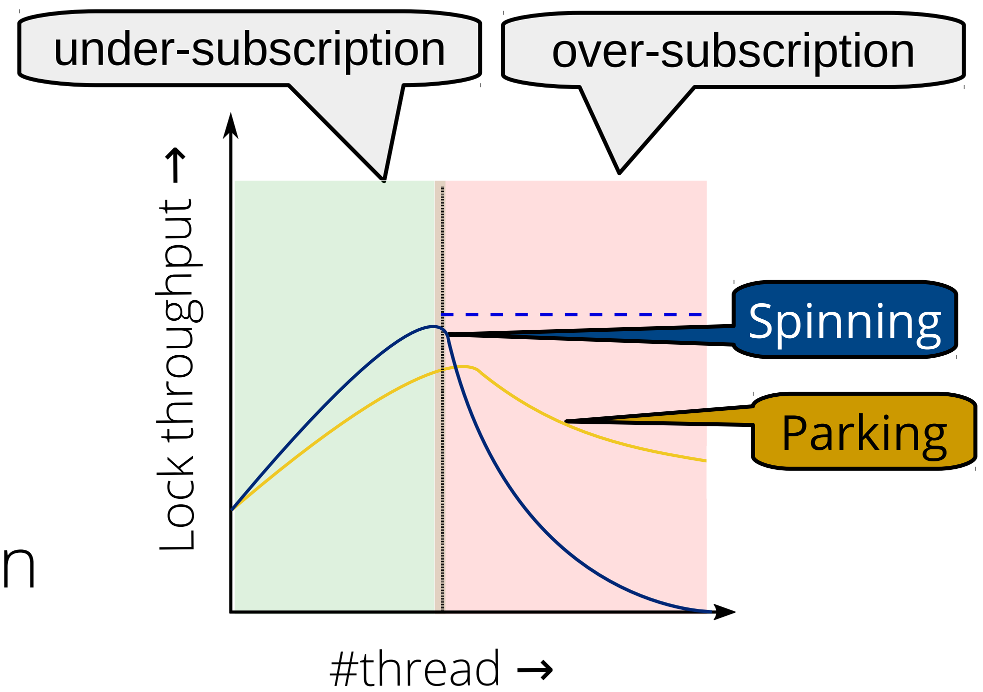
Blocking/parking approach

- Under subscription
  - #threads <= #cores
- Over subscription
  - #threads > #cores
- Spin-then-park strategy
  1) Spin for a certain duration
  2) Add to a parking list
  3) Schedule out (park/block)



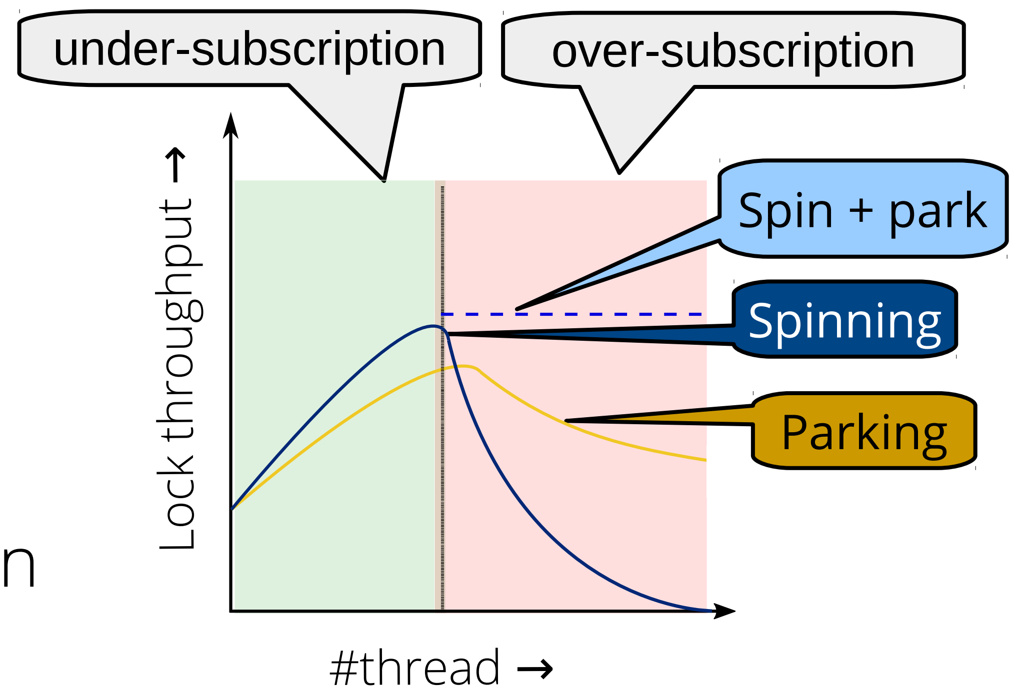
Blocking/parking approach

- Under subscription
  - #threads <= #cores
- Over subscription
  - #threads > #cores
- Spin-then-park strategy
  1) Spin for a certain duration
  2) Add to a parking list
  3) Schedule out (park/block)



Blocking/parking approach

- Under subscription
  - #threads <= #cores
- Over subscription
  - #threads > #cores
- Spin-then-park strategy
  1) Spin for a certain duration
  2) Add to a parking list
  3) Schedule out (park/block)



Issues with blocking synchronization primitives

- High memory footprint for NUMA-aware locks
- Inefficient blocking strategy
  - Scheduling overhead in the critical path
  - Cache-line contention while scheduling out

CST lock

- NUMA-aware lock
- Low memory footprint
  - Allocate socket specific data structure when used
  - 1.5–10X memory less memory consumption
- Efficient parking/wake-up strategy
  - Limit the spinning up to a waiter’s time quantum
  - Pass the lock to an active waiter
  - Improves scalability by 1.2–4.7X

CST lock design

- NUMA-aware lock
  - ➢ Cohort lock principle
  - \+ Mitigates cache-line contention and bouncing
- Memory efficient data structure
  - ➢ Allocate socket structure (snode) when used
  - ➢ Snodes are active until the life-cycle of the lock
  - \+ Does not stress the memory allocator

CST lock design

- NUMA-aware parking list
  - ➢ Maintain separate per-socket parking lists for readers and writers

  \+ Mitigates cache-line contention in over-subscribed scenario

  \+ Allows distributed wake-up of parked readers

CST lock design

- Remove scheduler intervention
  - ➢ Pass the lock to a spinning waiter
  - ➢ Waiters park themselves if more than one tasks are running on a CPU (system load)

  \+ Scheduler not involved in the critical path

  \+ Guarantees forward progress of the system

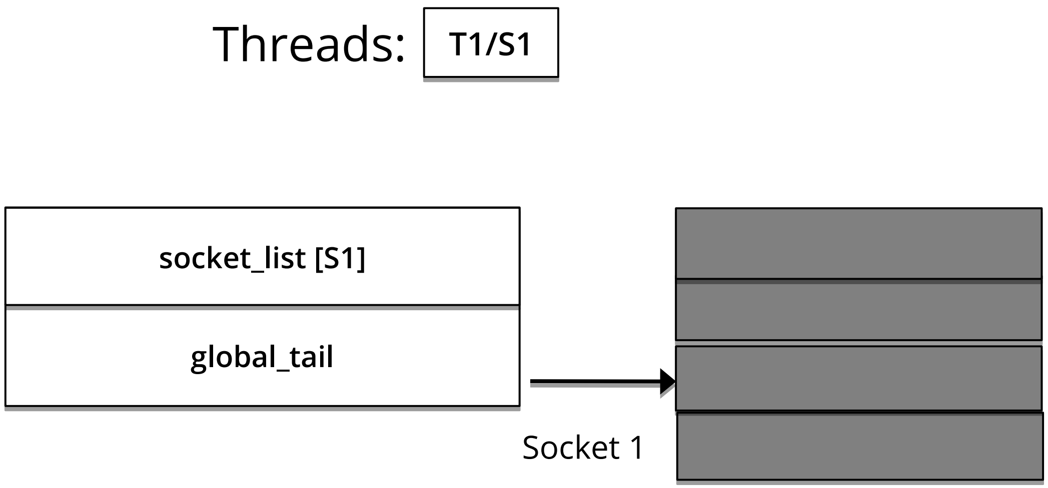
Lock instantiation

- Initially no snodes are allocated
- Thread in a particular socket initiates an allocation

Threads:

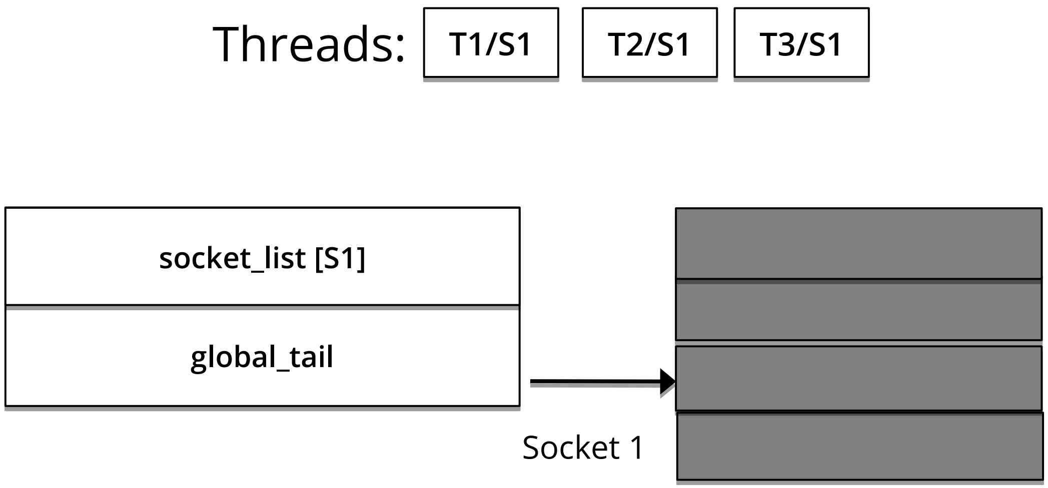
Lock instantiation

- Initially no snodes are allocated
- Thread in a particular socket initiates an allocation



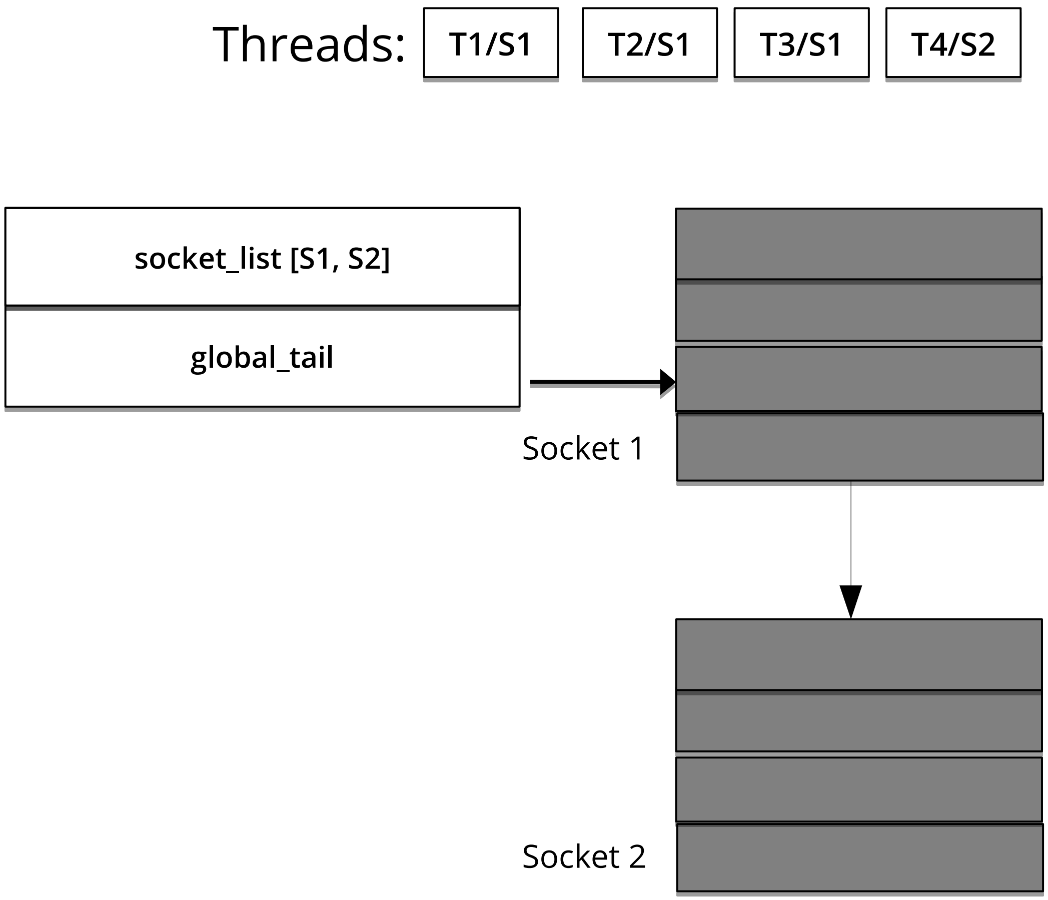
Lock instantiation

- Initially no snodes are allocated
- Thread in a particular socket initiates an allocation



Lock instantiation

- Initially no snodes are allocated
- Thread in a particular socket initiates an allocation



CST lock phase

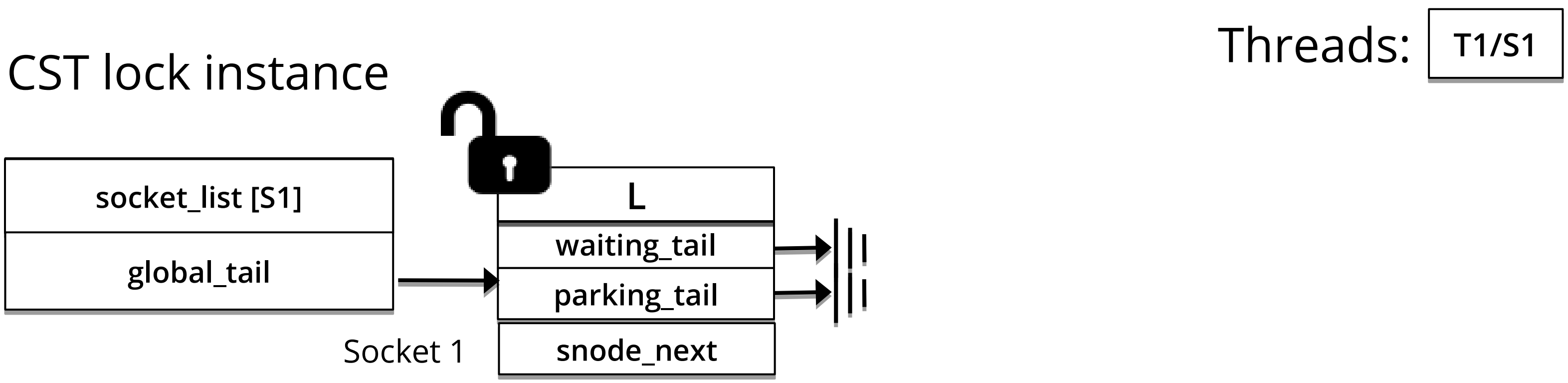
CST lock instance

socket_list
global_tail

Threads:

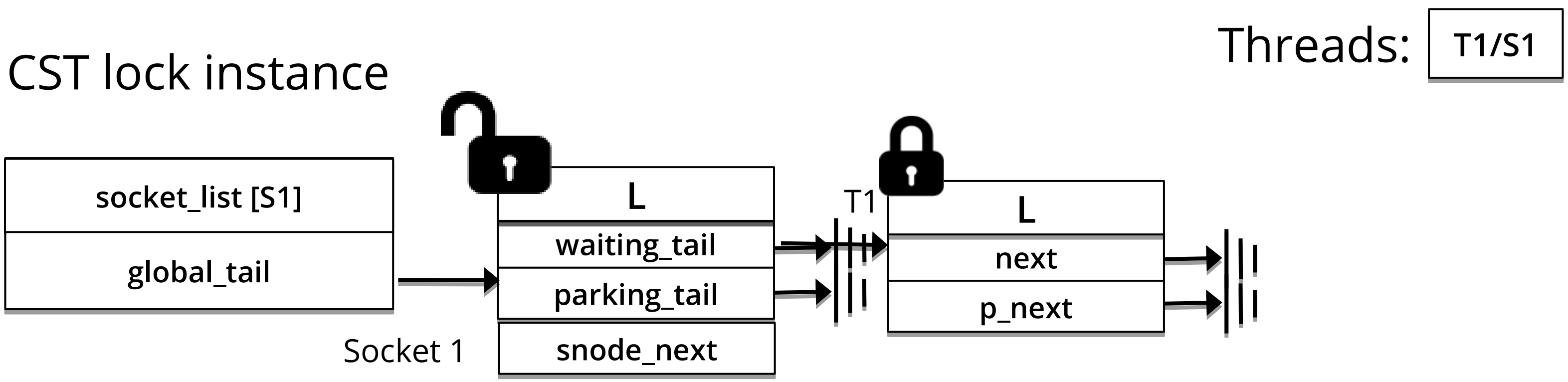
- Allocate thread specific structure on the stack
- Three states for each node
  - L → locked
  - UW → unparked/spinning waiter
  - PW → parked / blocked / scheduled out waiter

CST lock phase



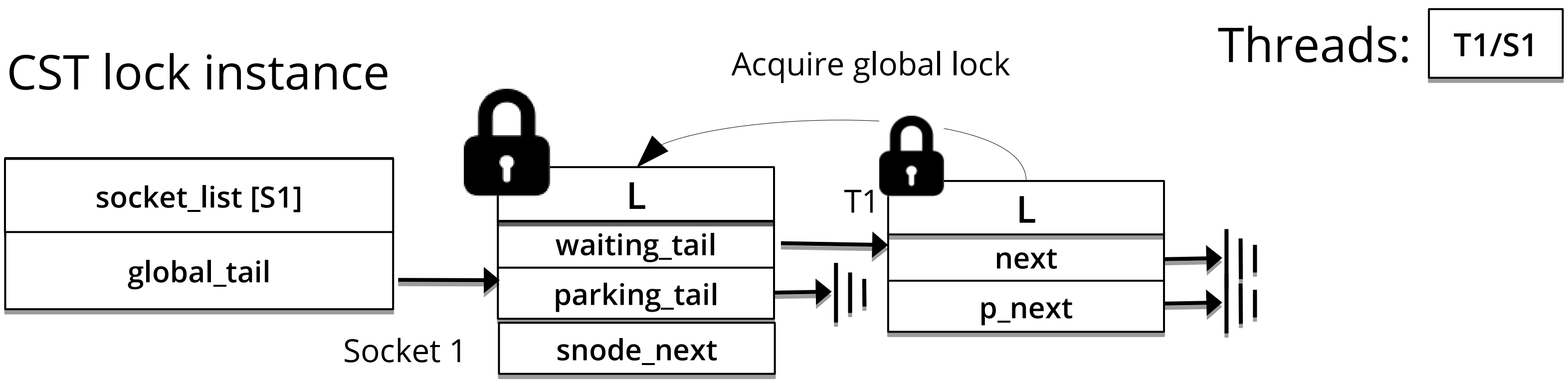
- Allocate thread specific structure on the stack
- Three states for each node
  - L → locked
  - UW → unparked/spinning waiter
  - PW → parked / blocked / scheduled out waiter

CST lock phase



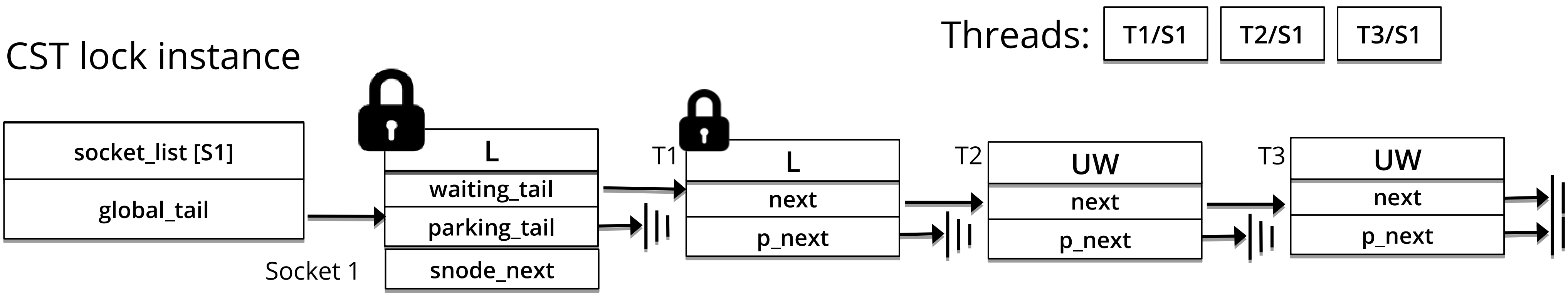
- Allocate thread specific structure on the stack
- Three states for each node
  - L → locked
  - UW → unparked/spinning waiter
  - PW → parked / blocked / scheduled out waiter

CST lock phase



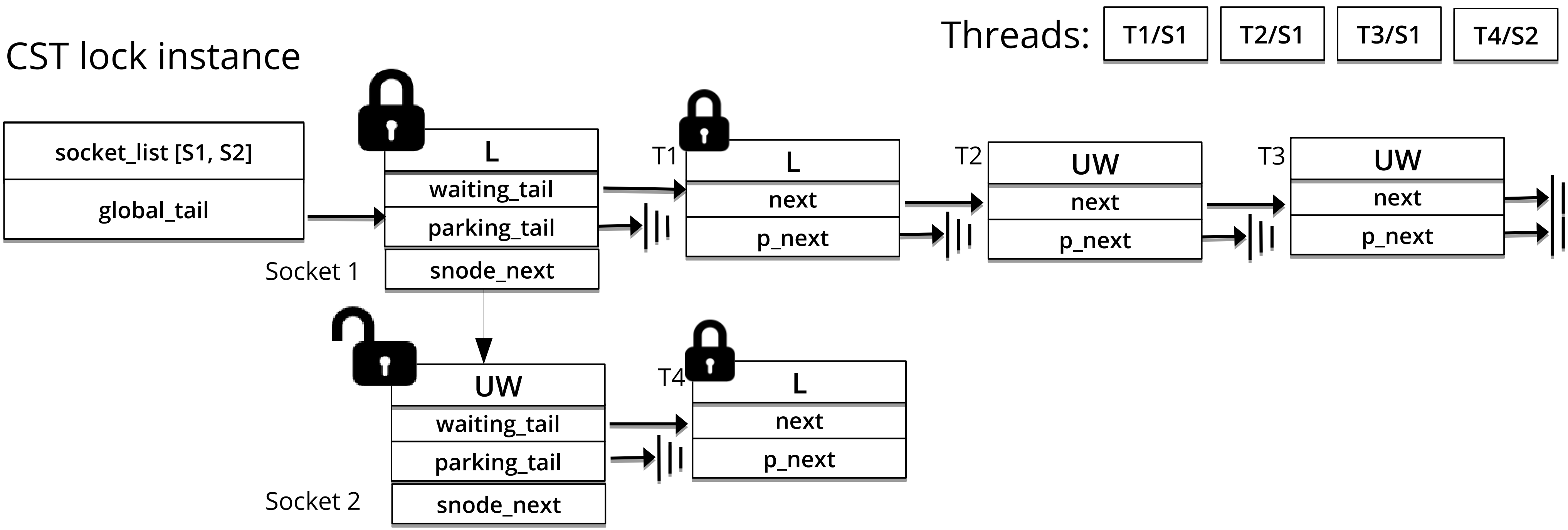
- Allocate thread specific structure on the stack
- Three states for each node
  - L → locked
  - UW → unparked/spinning waiter
  - PW → parked / blocked / scheduled out waiter

CST lock phase



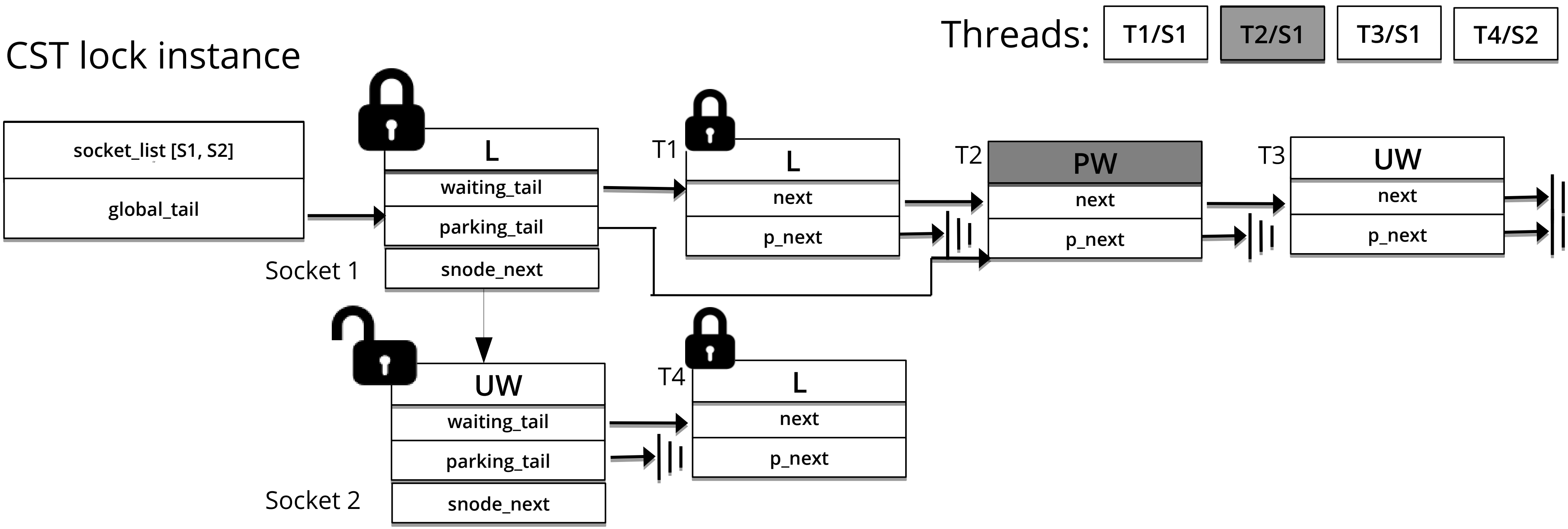
- Allocate thread specific structure on the stack
- Three states for each node
  - L → locked
  - UW → unparked/spinning waiter
  - PW → parked / blocked / scheduled out waiter

CST lock phase



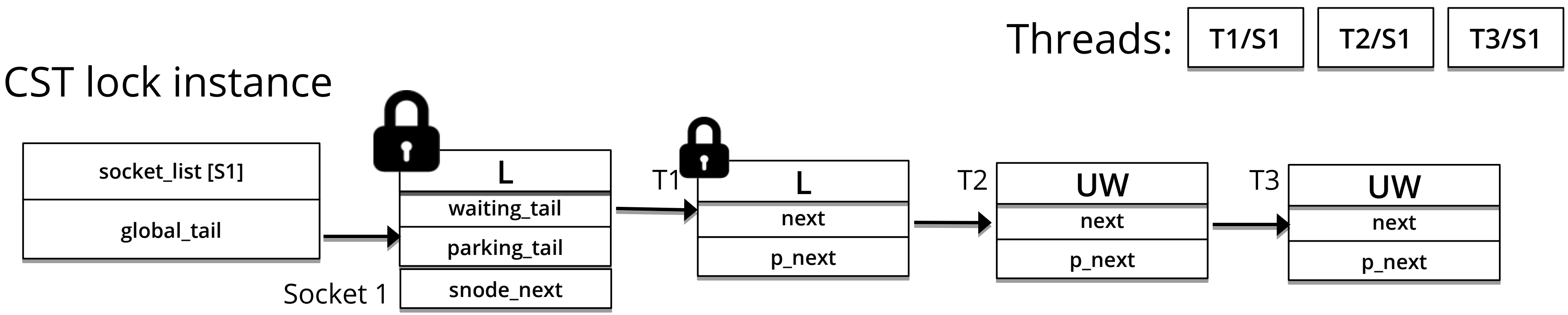
- Allocate thread specific structure on the stack
- Three states for each node
  - L → locked
  - UW → unparked/spinning waiter
  - PW → parked / blocked / scheduled out waiter

CST lock phase: blocking/parking



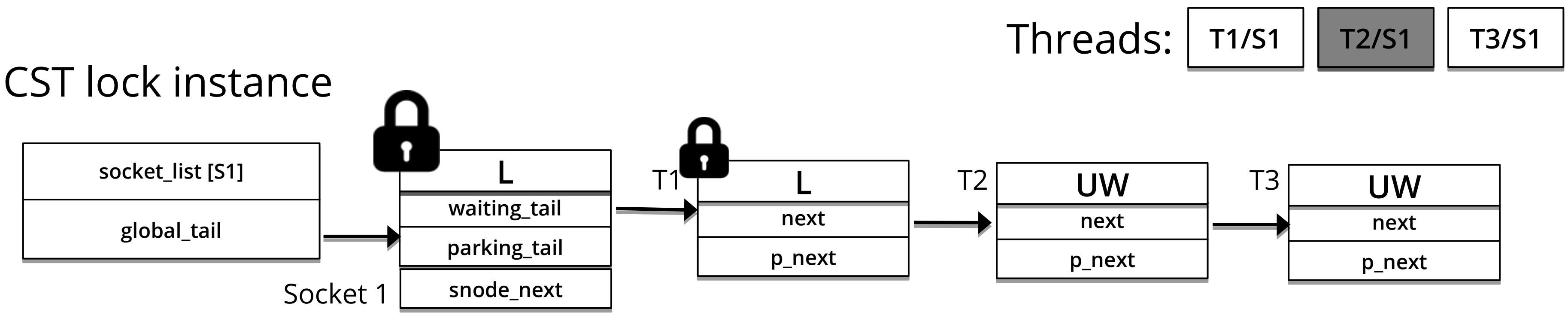
- Before scheduling out, waiters atomically
  - Update the status from UW to PW
  - Add themselves to the parking list

CST unlock phase



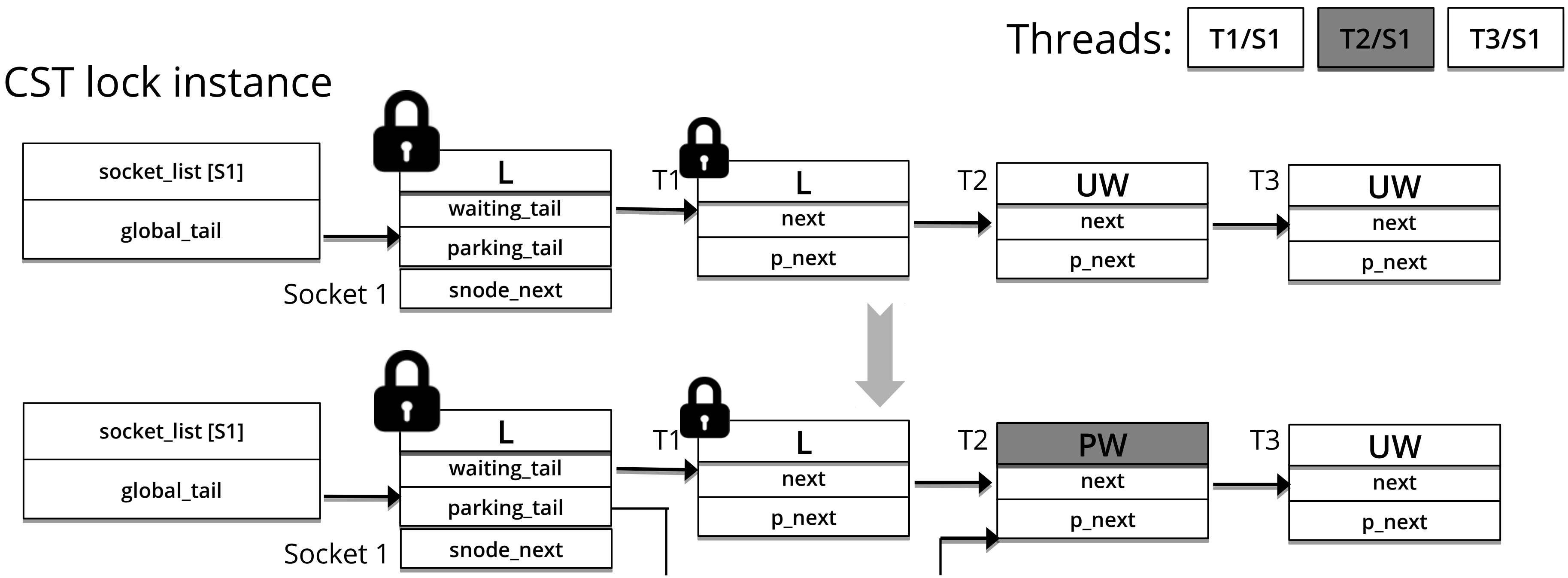
Pass the lock to a spinning waiter

CST unlock phase



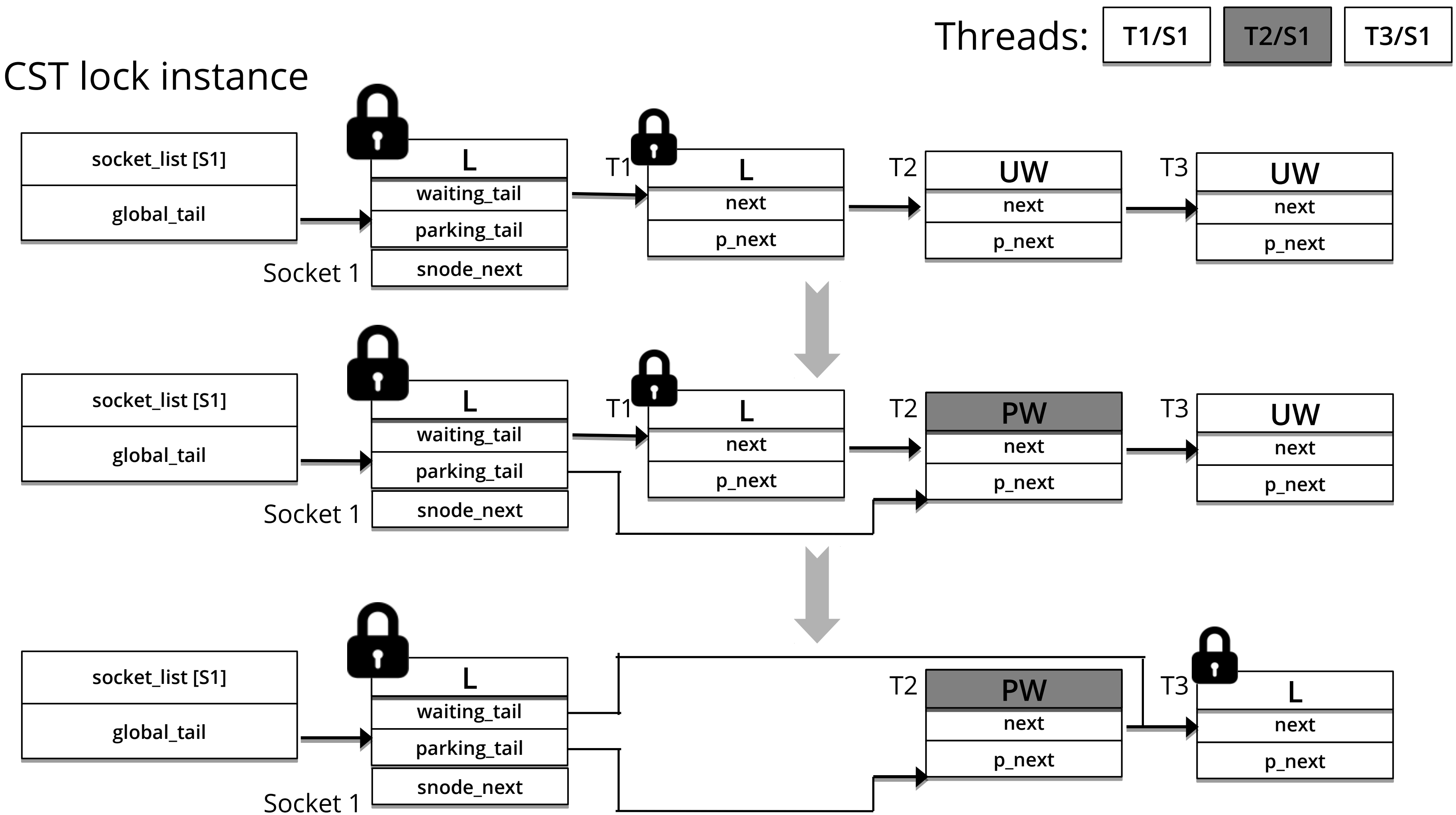
Pass the lock to a spinning waiter

CST unlock phase



Pass the lock to a spinning waiter

CST unlock phase



Pass the lock to a spinning waiter

Implementation

- Implemented in the Linux kernel
- Structures modified
  - File system: `inode`
  - Memory management: `mmap_sem`
- Please see our paper
  - Read-write lock
  - Pseudo code

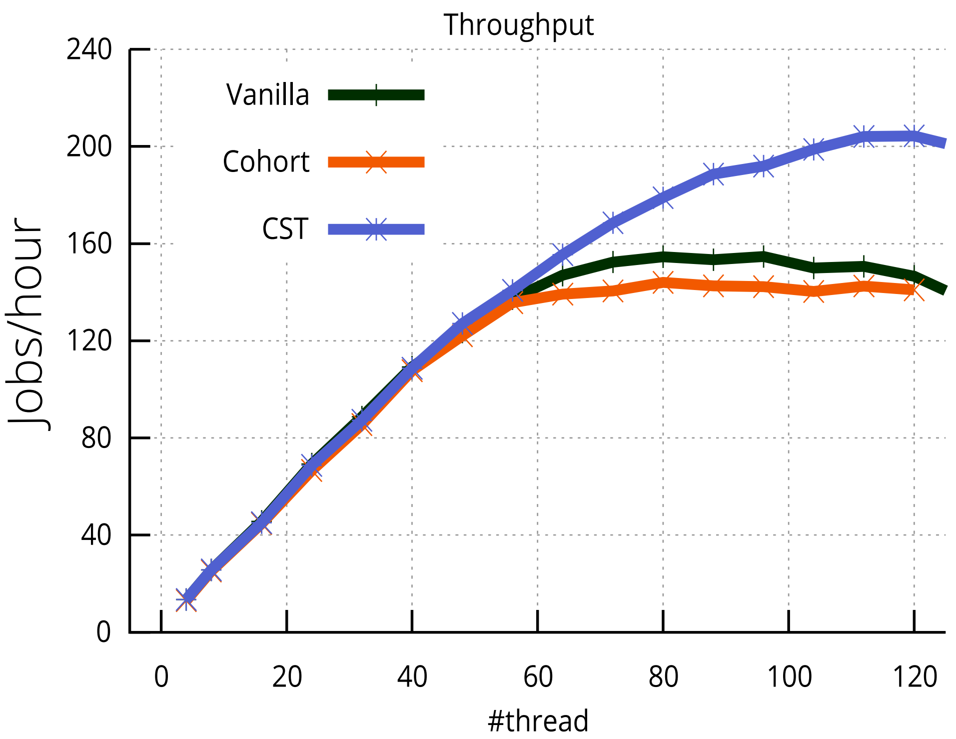
https://github.com/sslab-gatech/cst-locks

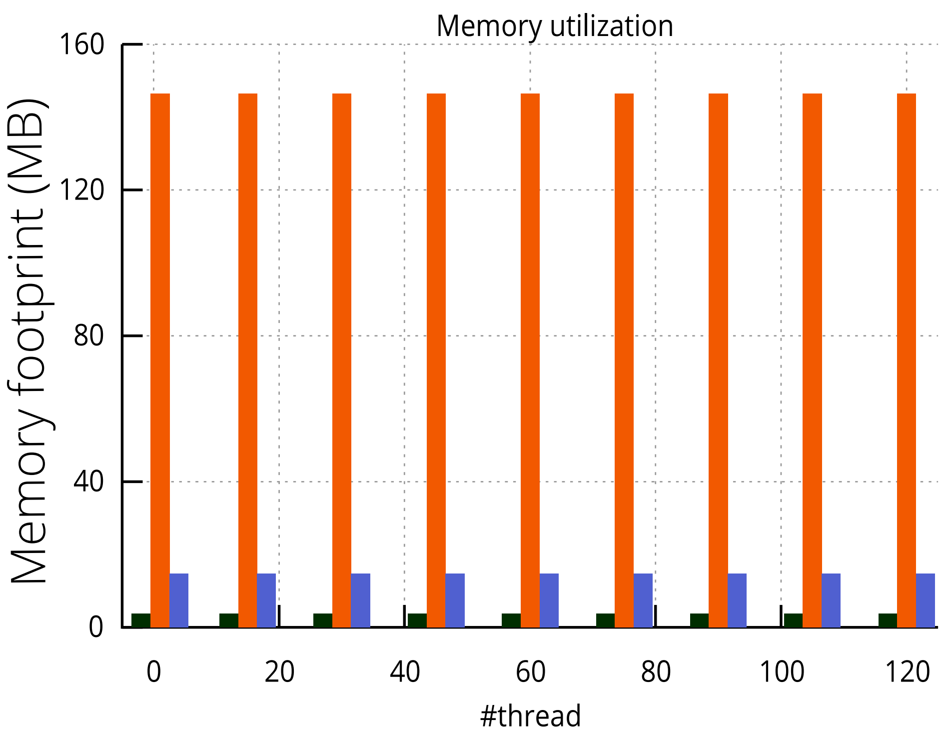
Evaluation

- Performance of locks in terms of scalability and memory footprint?
- Blocking/parking strategy effectiveness?

- Setup: 8-socket, 120-core NUMA machine

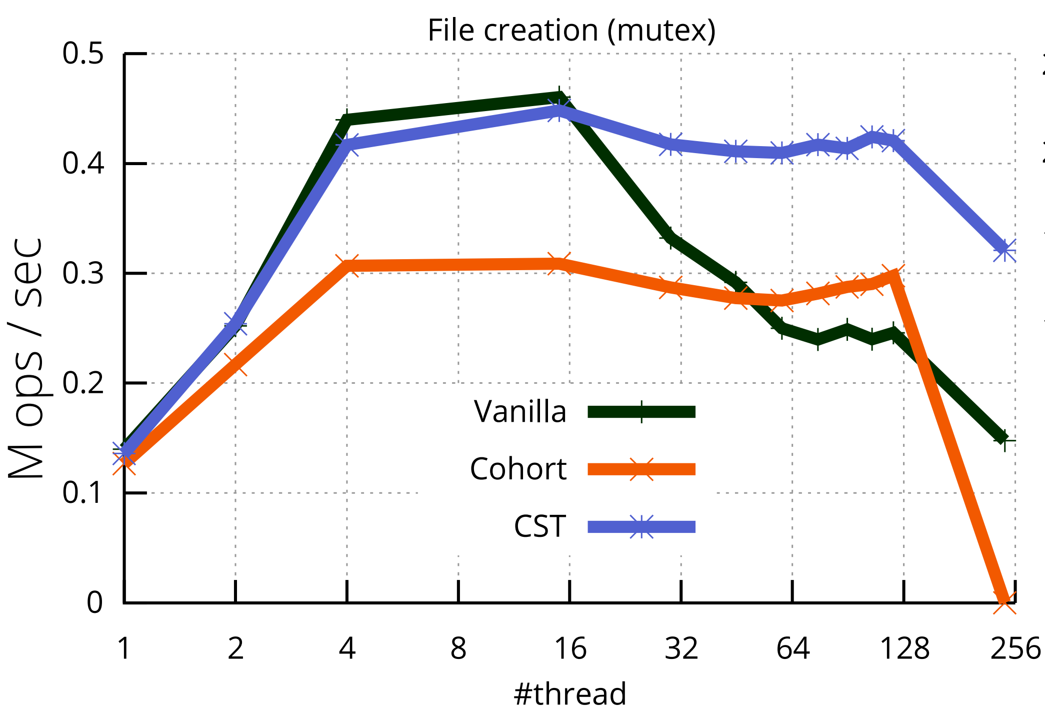
Case study: Psearchy

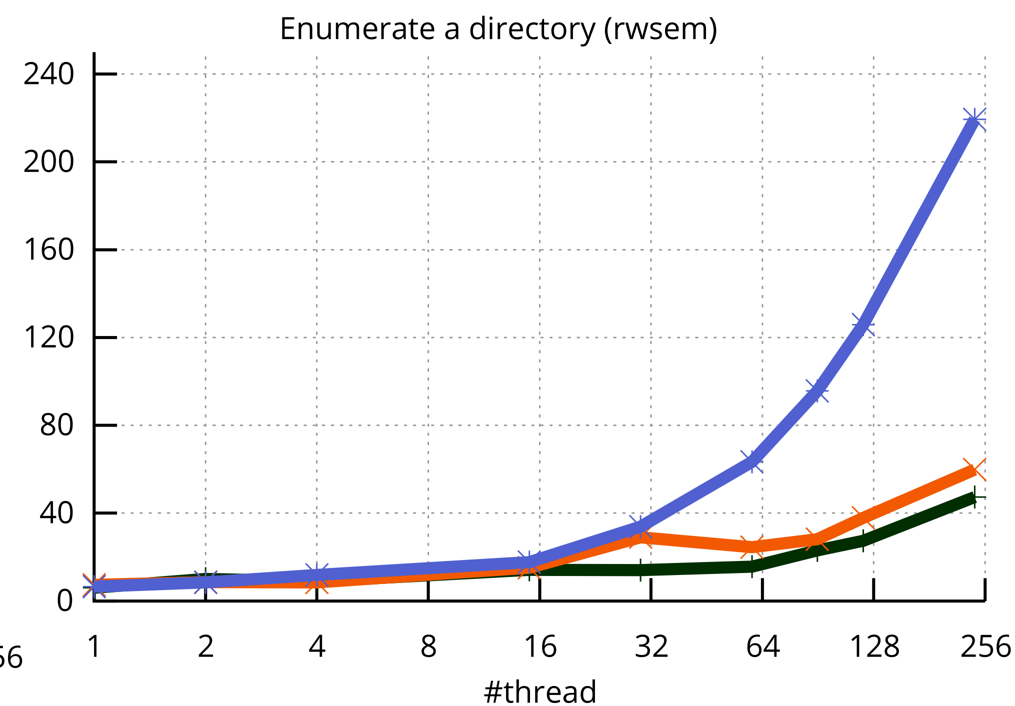




- Overcomes memory footprint and scheduling overhead
- Uses 1.5–9.1X less memory than the Cohort lock
- Improves throughput by 1.4–1.6X

Effective parking strategy





- Better performance for both under- and over-subscribed scenario
- Improves scalability by 1.3–3.7X

Conclusion

- Two blocking synchronization primitives
  - NUMA-aware mutex and read-write semaphore
- Dynamically allocated data structure
  - Resolve NUMA-aware lock's footprint issue
- Efficient spin-then-park strategy
  - Scheduling-aware parking/wake-up strategy
  - Mitigate scheduler interaction

Thank you!