Hewlett Packard Enterprise

UNIVERSITY OF TORONTO

Be My Guest – MCS Lock Now Welcomes Guests

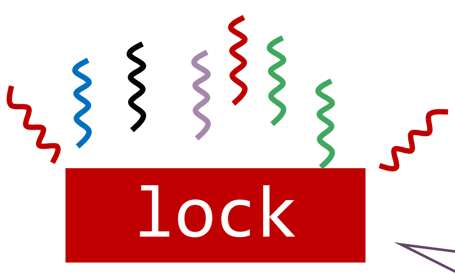
Tianzheng Wang, University of Toronto

Milind Chabbi, Hewlett Packard Labs

Hideaki Kimura, Hewlett Packard Labs

Protecting shared data using locks

```
foo() {
   lock.acquire();
   data = my_value;
   lock.release();
}
```



Centralized spin locks

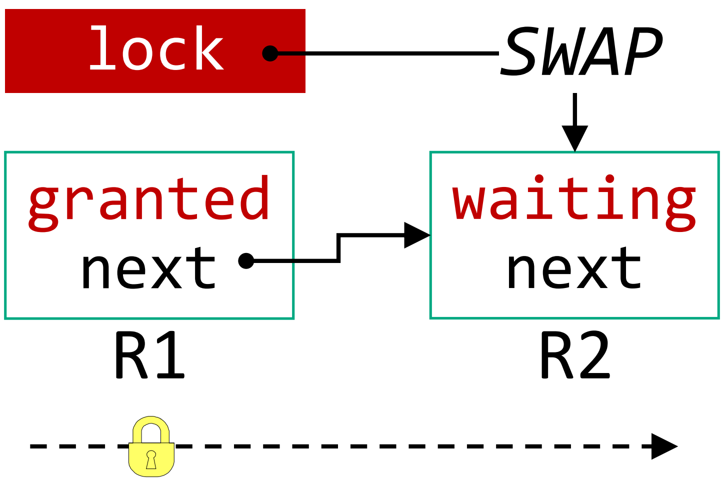
- Test-and-set, ticket, etc.
- Easy implementation
- Widely adopted
- Waste Interconnect traffic
- Cache ping-ponging

Contention on a centralized location

Hewlett Packard Enterprise

UNIVERSITY OF TORONTO

MCS Locks



– Local spinning

– FIFO order

Non-standard interface

```
foo(qnode) {
   lock.acquire(qnode);
   data = my_value;
   lock.release(qnode);
}
```

Queue nodes everywhere

"…it was especially complicated when the critical section spans multiple functions. That required having functions also accepting an additional MCS node in its parameter."

- Jason Low, HPE's Linux kernel developer

Not easy to adopt MCS lock with non-standard API

Hewlett Packard Enterprise

UNIVERSITY OF TORONTO

"…out of the 300+ places that make use of the dcache lock, 99% of the contention came from only 2 functions. Changing those 2 functions to use the MCS lock was fairly trivial…"

- Jason Low, HPE's Linux kernel developer

Not all lock users are created equal

Regular users

```
frequent_func(qnode) {
  lock.acquire(qnode);
  ...
  lock.release(qnode);
}
```

Guests

```
infrequent_func…(qnod
infrequent_func2(qnode
infrequent_func1(qnode){
  lock.acquire(qnode);
  ...
  lock.release(qnode);
}
```

– Transaction workers vs. DB snapshot composer

– Worker threads vs. daemon threads

Existing approaches

	Multi-process applications	Storage requirements
Thread-local queue nodes	Works	Bloated memory usage
K42-MCS	Queue nodes on the stack	Satisfies
Cohort locks	Works	Extra memory per node Possible data layout change

Hewlett Packard Enterprise

UNIVERSITY OF TORONTO

MCSg: best(MCS) + best(TAS)

Regular users

```
foo(qnode) {
  lock.acquire(qnode);
  ...
  lock.release(qnode);
}
```

Keeps all the benefits of MCS

Guests

```
bar() {
  lock.acquire();
  ...
  lock.release();
}
```

No queue node needed

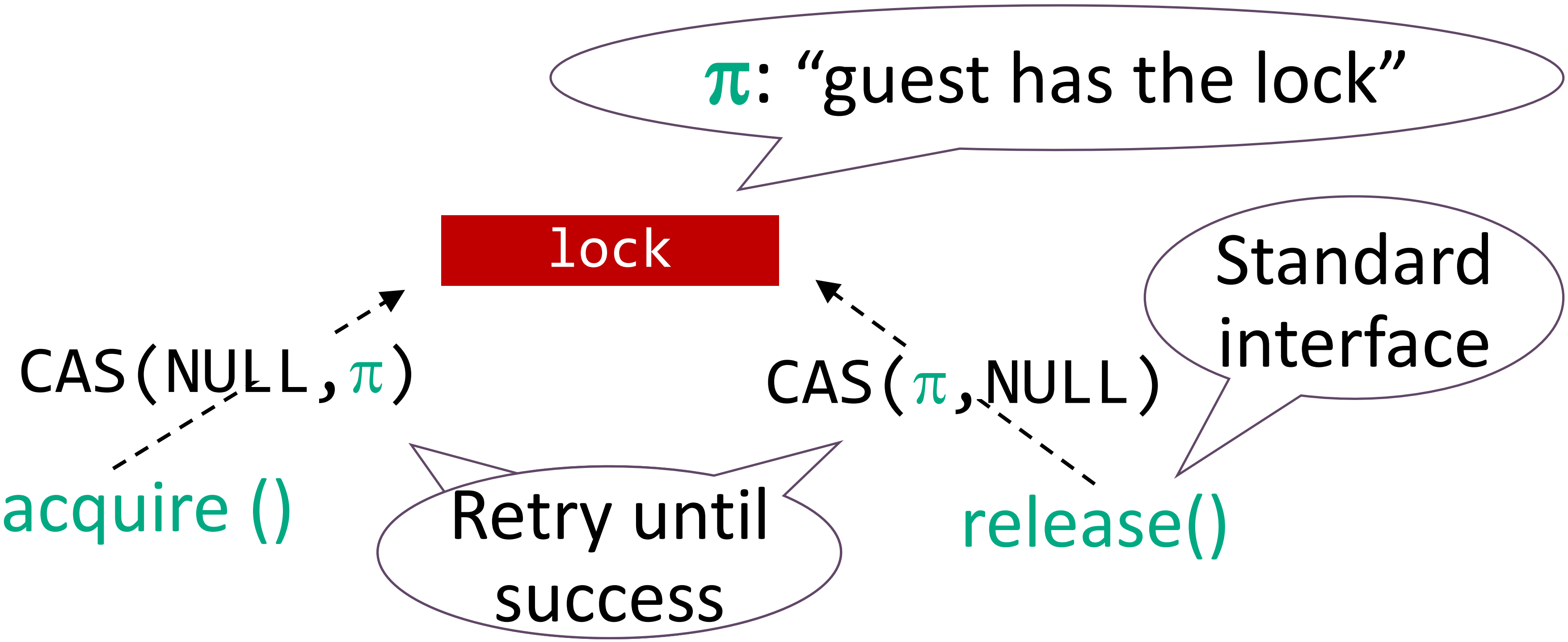
Hewlett Packard Enterprise

UNIVERSITY OF TORONTO

MCSg: use cases

- Drop-in replacement for MCS to support guests
- Replace a centralized spinlock for performance
  - Start from all guests,
  - Gradually identify regular users and adapt
- As a building block for composite locks
  - Same interface as MCS
  - Same storage requirement

Guests in MCSg



Guests: similar to using a centralized spin lock

Regular users – change in acquire()

π

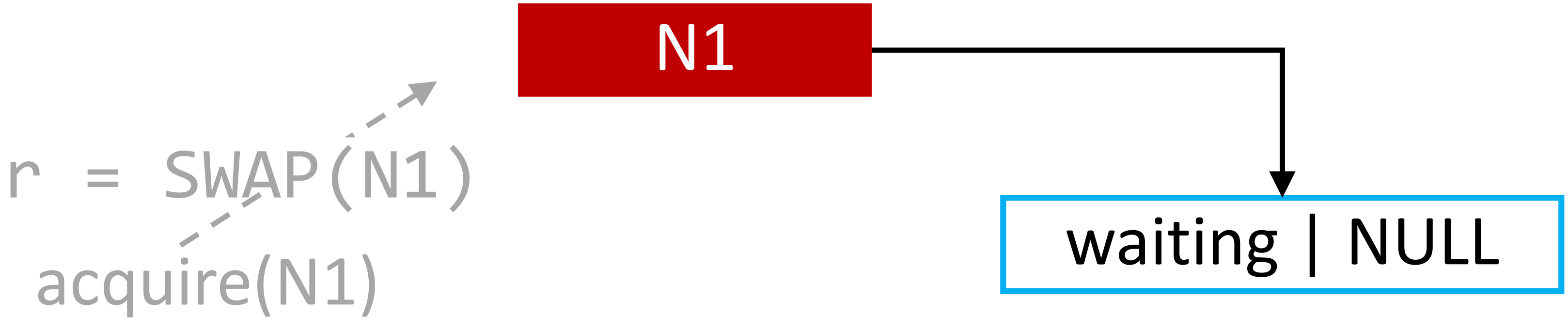
No guest:
same as MCS

```
r = SWAP(N1)
```

waiting | NULL

acquire(N1)

Regular users – change in acquire()



Regular users – change in acquire()

π

r = SWAP(N1)

acquire(N1)

t = SWAP(π)

waiting | NULL

r == π, return π for the guest to release the lock

t == N1/another ptr

Retry with r = SWAP(t)

r == NULL

Got lock

+5 LoC in acquire(...), *no change* in release(...)

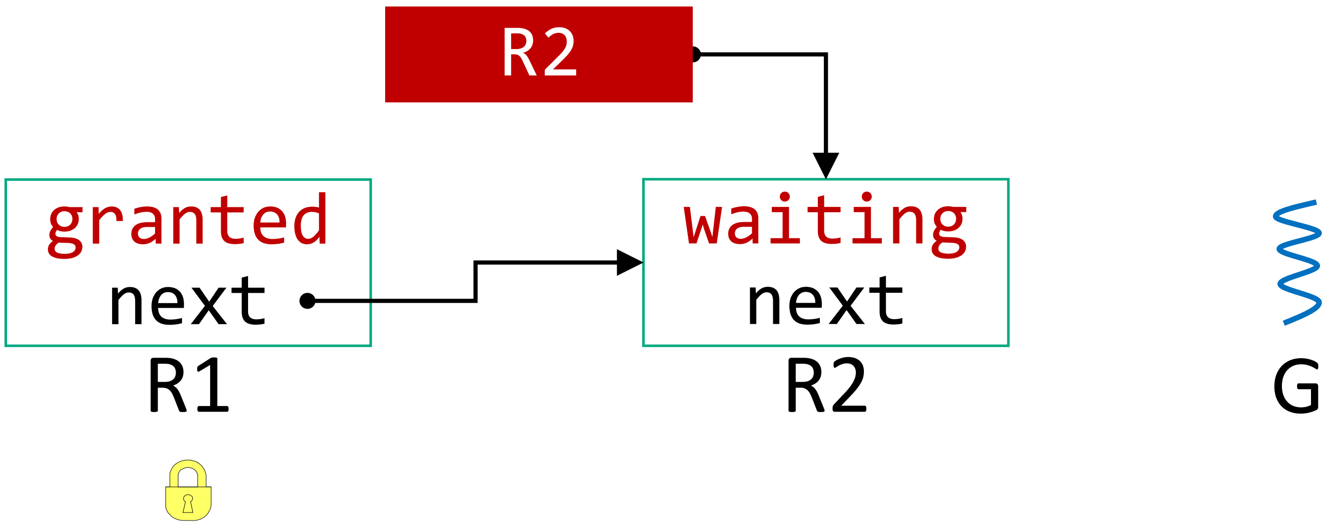
MCSg++ extensions

– Guest starvation
  – CAS: no guaranteed success in a bounded # of steps
  – Solution: attach the guest after a regular user
– FIFO order violations
  – Retrying XCHG might line up after a later regular user
  – Solution: retry with ticket

Hewlett Packard Enterprise

UNIVERSITY OF TORONTO

Reducing guest starvation



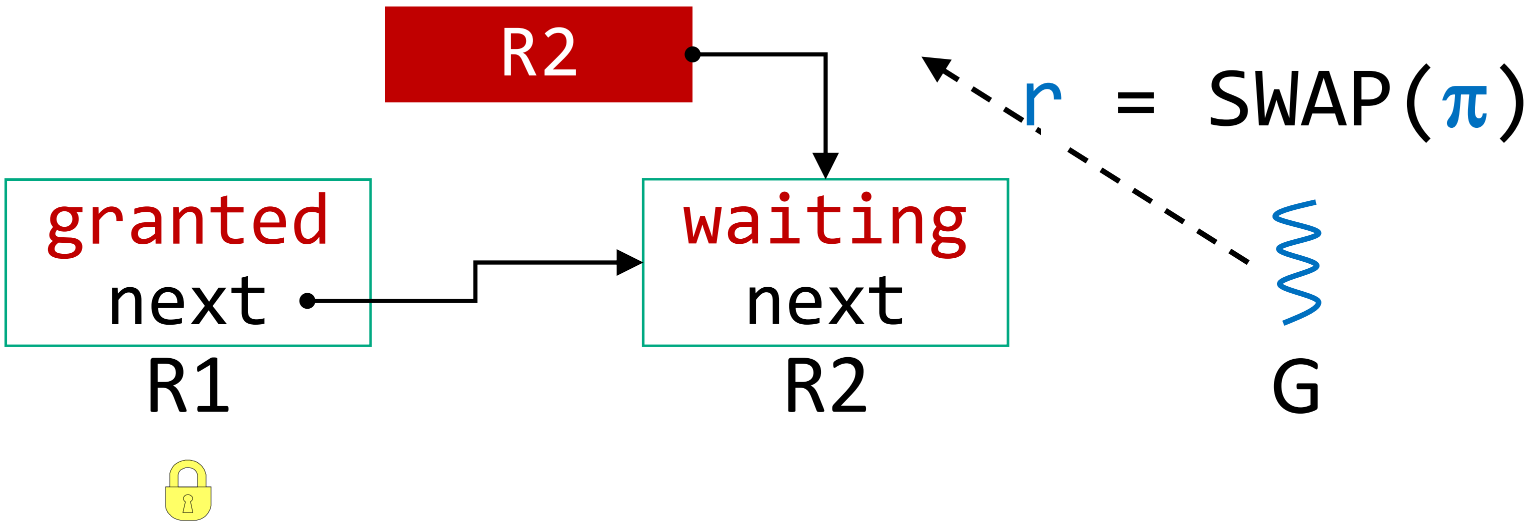
r = XCHG(π)

r.next = *Guest Waiting*

spin until r.next == *Guest Granted*

r.next = *Guest Acquired*

Reducing guest starvation



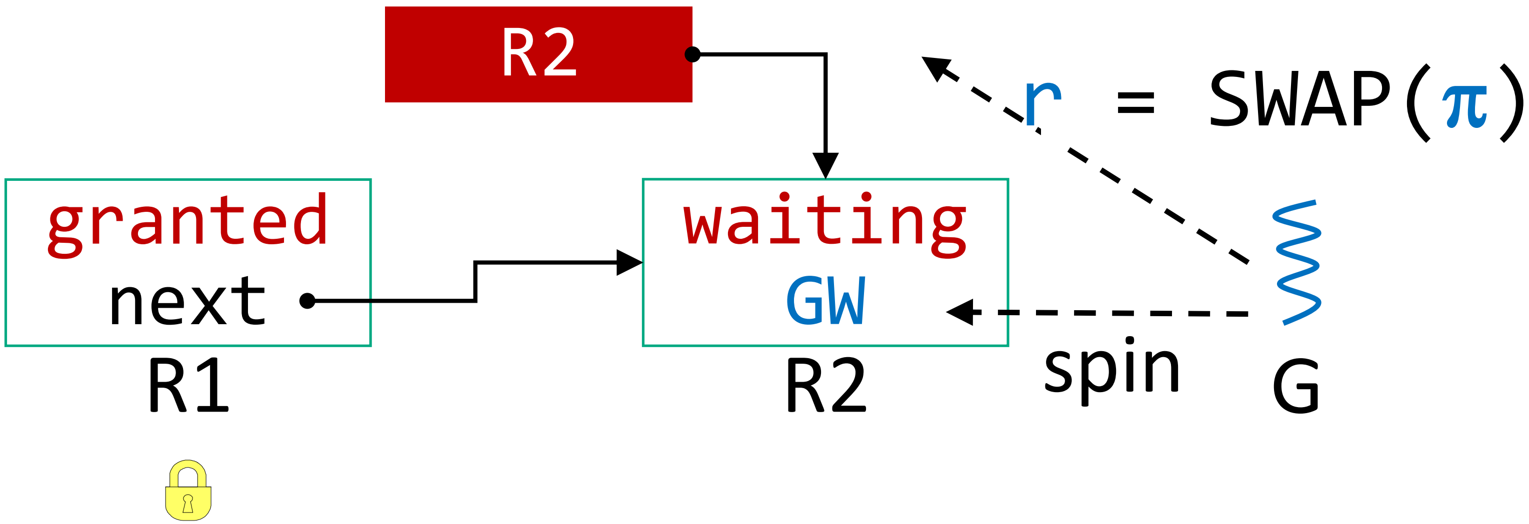
r = XCHG(π)

r.next = *Guest Waiting*

spin until r.next == *Guest Granted*

r.next = *Guest Acquired*

Reducing guest starvation



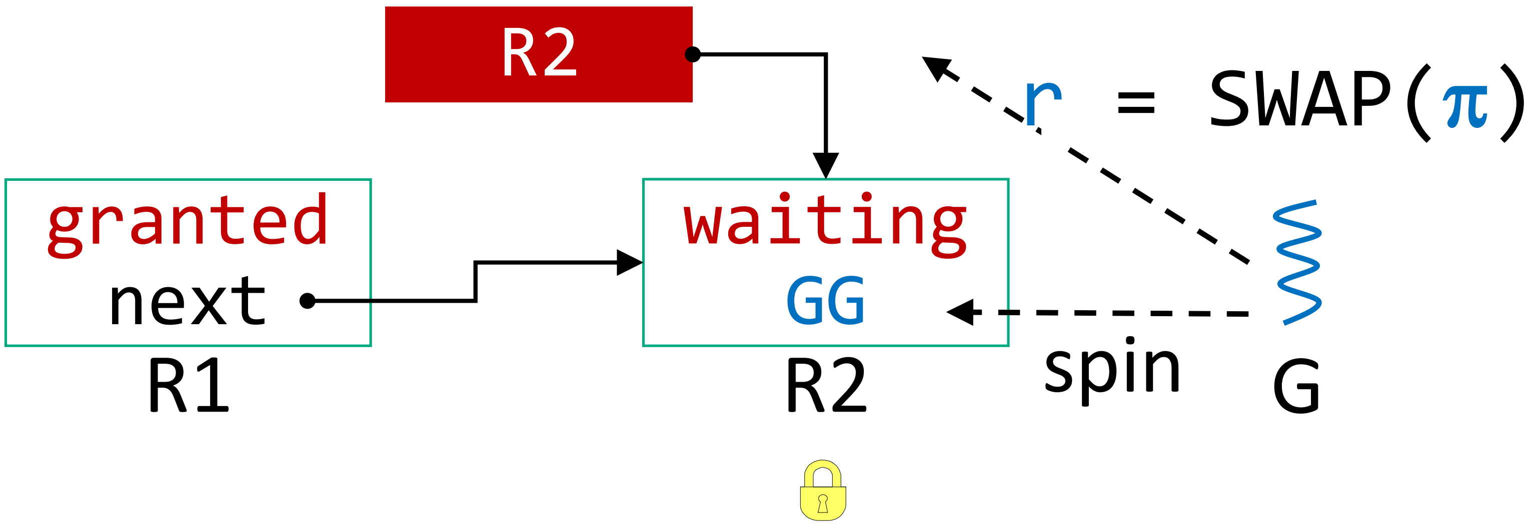
r = XCHG(π)

r.next = *Guest Waiting*

spin until r.next == *Guest Granted*

r.next = *Guest Acquired*

Reducing guest starvation



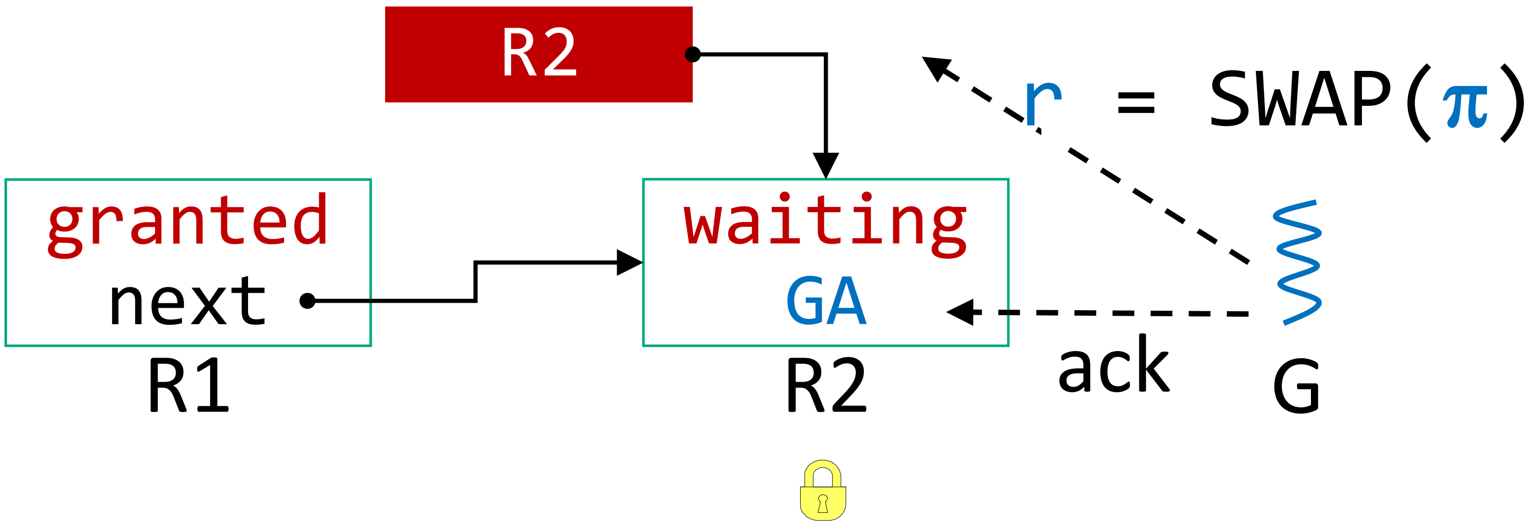
r = XCHG(π)

r.next = *Guest Waiting*

spin until r.next == *Guest Granted*

r.next = *Guest Acquired*

Reducing guest starvation



r = XCHG(π)

r.next = *Guest Waiting*

spin until r.next == *Guest Granted*

r.next = *Guest Acquired*

Evaluation

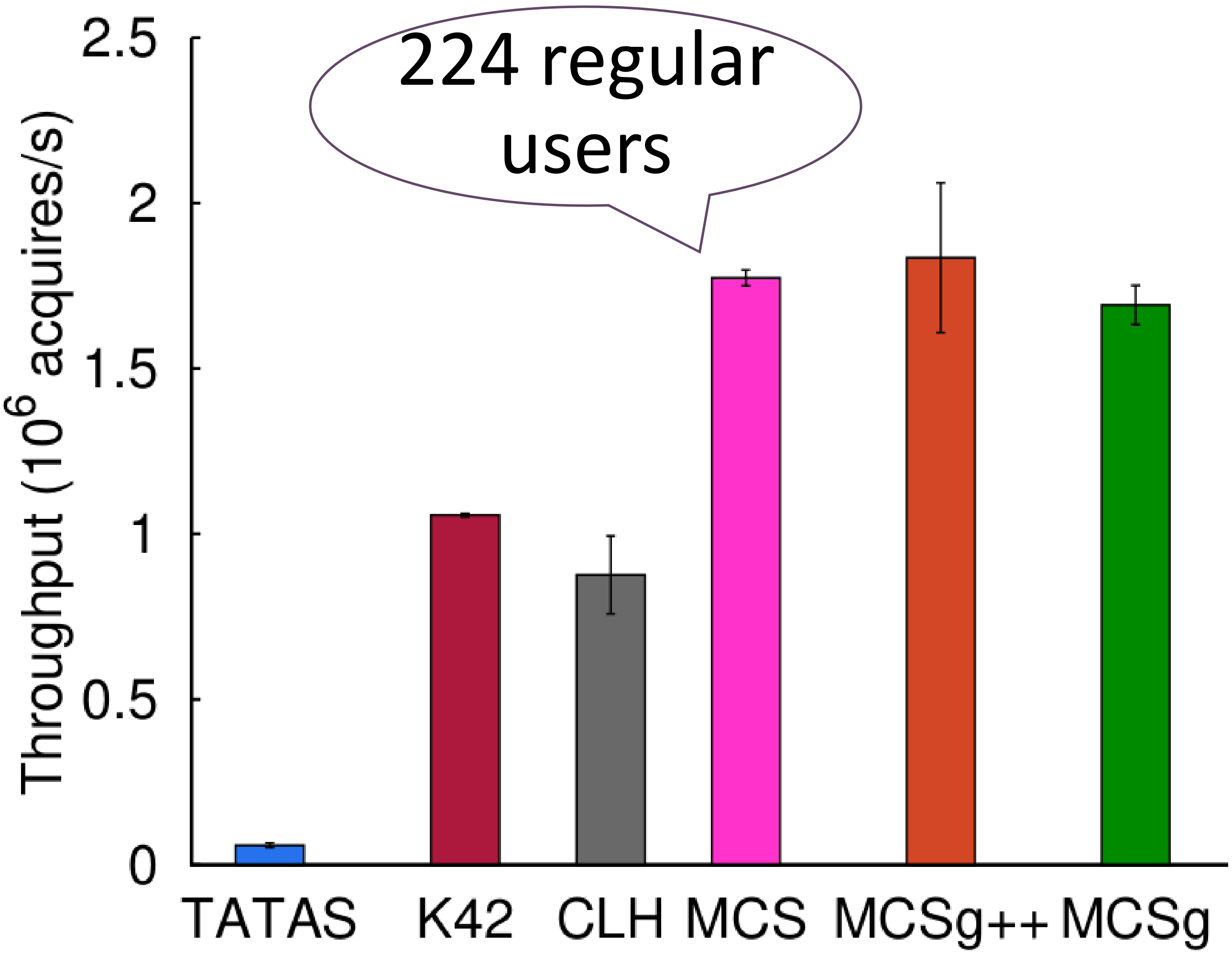
- HP DragonHawk
  - 15-core Xeon E7-4890 v2 @ 2.80GHz
  - 16 sockets → 240 physical cores
  - L2 256KB/core, L3 38MB/socket, 12TB DRAM
- Microbenchmarks
  - MCSg, MCSg++, CLH, K42-MCS, TATAS
  - Critical section: 2 cache line accesses, high contention
- TPC-C with MCSg in FOEDUS, an OSS database

Maintaining MCS's scalability

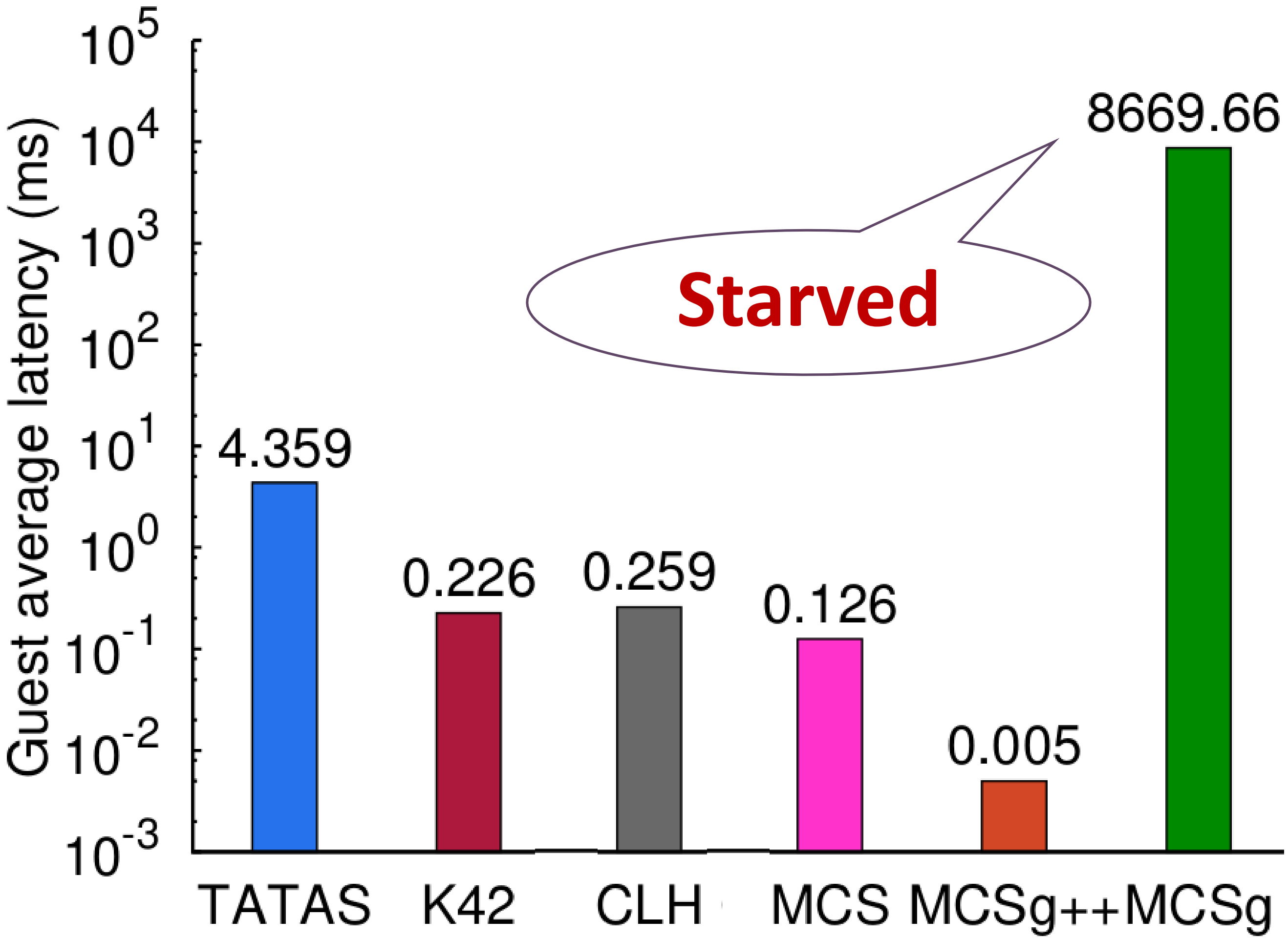
- TPC-C Payment
  - 192 workers
  - Highly contented – one warehouse

Lock	MTPS	STDEV
TATAS	0.33	0.095
MCS	0.46	0.011
MCSg	0.45	0.004

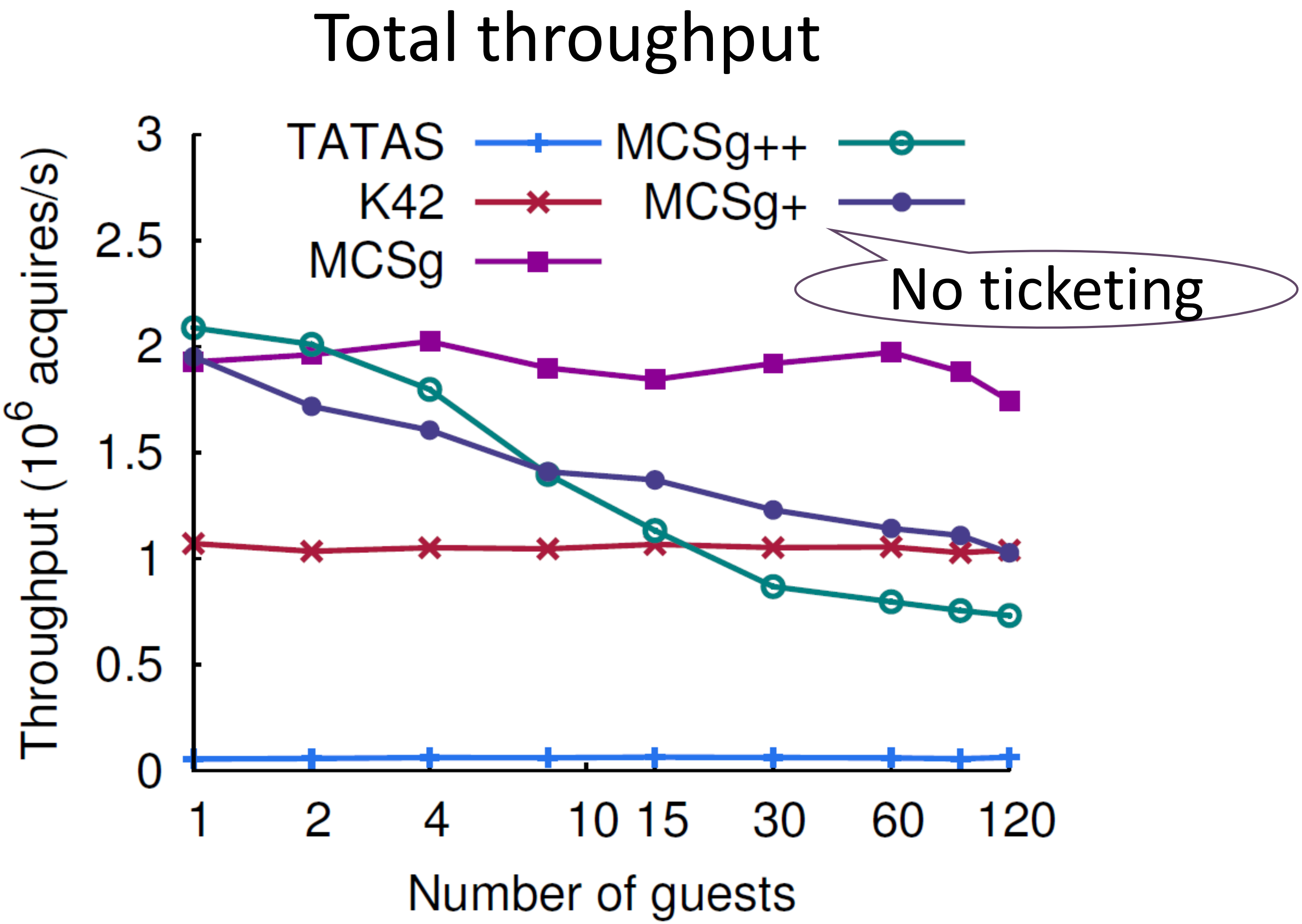
One guest + 223 regular users



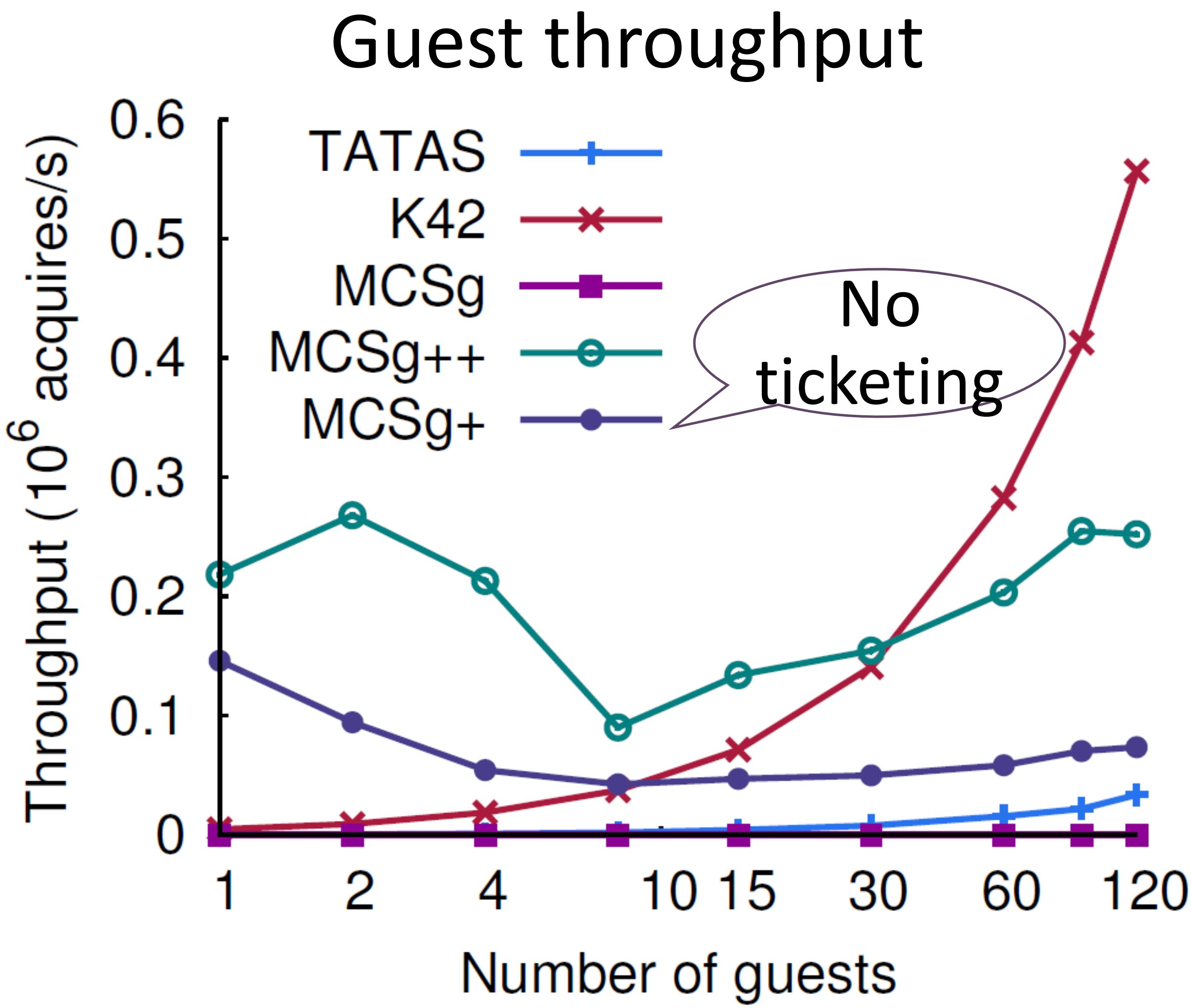
One guest + 223 regular users



Varying number of guests



Varying number of guests



Conclusions

– Not all lock users are created equal
  – Pervasive guests prevent easy adoption of MCS lock
– MCSg: dual-interface
  – Regular users: *acquire/release(lock, qnode)*
  – Infrequent guests: *acquire/release(lock)*
  – Easy-to-implement: ~20 additional LoC
  – As scalable as MCS (guests being minority at runtime)

Find out more in our paper!

Hewlett Packard Enterprise

UNIVERSITY OF TORONTO