An Alternative Locking Mechanism for Multicore Operating Systems

or

How I Learned to Stop Worrying and Love Interprocessor Interrupts

Kurtis Heimerl

Intuition

- Locking is much harder in Multicore/SMP
  - We go through a lot of effort to avoid cache coherence traffic
    - MCS Locks
    - Semaphores on real-time systems
  - We also go through a lot of effort to get performance and latency out of lock structures
    - Fine-grained locking
    - Clustered Objects (Gamsa et al)

Intuition

- As multicore systems begin to resemble distributed systems, is there anything we can learn from their locking structures?
  - Chubby (Burrows et al) argues explicitly against fine grained locking, it causes too much network traffic.
    - Not focused on performance, rather reliability
  - However, the idea of using messages is intriguing. What advantages might this get us?

Messaging on Modern Chips

- Two Basic Types:
  - Shared Memory Queues
    - IPC-type communications
    - Well-supported by hardware
    - Likely to cause extra cache coherence traffic
  - Interprocessor Interrupts
    - Primarily used for uncommon events
      - I/O, errors
    - Not on fast path, hardware support poor
    - Will not cause extra coherence traffic
      - Even when using same bus, each message requires just two arbitrations

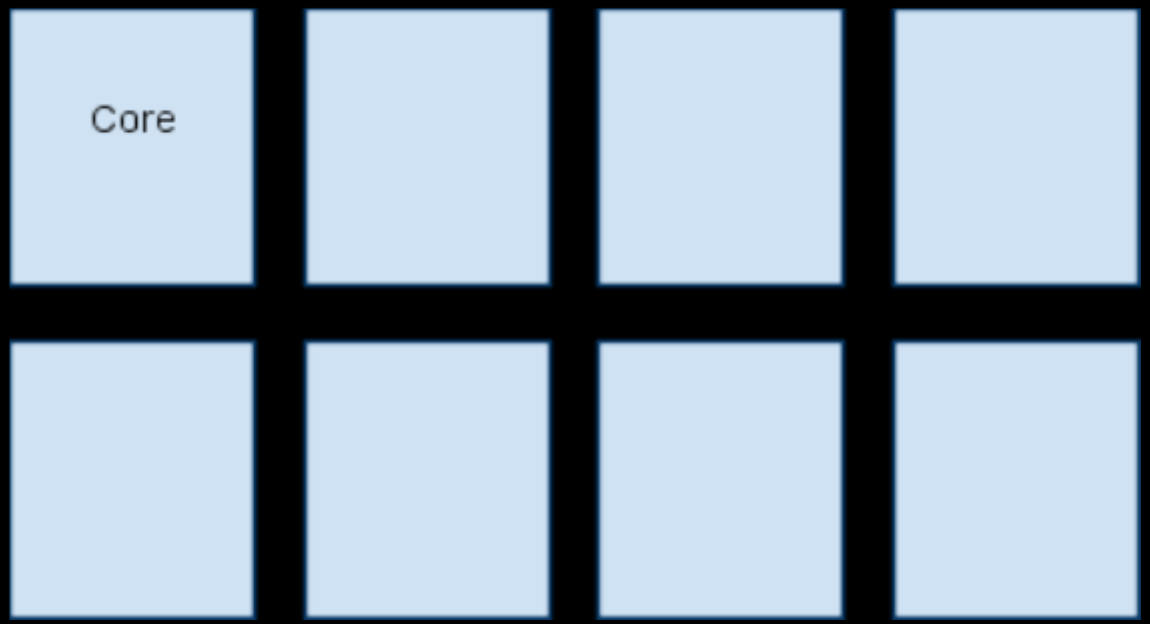
IPIs as Communication Primitives

- We focused on Kernel-mode IPIs.
  - Simpler
  - Purer (From a certain point of view...)
  - Distinct advantage on cache behavior.
  - Other messaging structures left as future work.

So how do we build locks out of IPIs?

Architecture

Assume an 8 core machine:

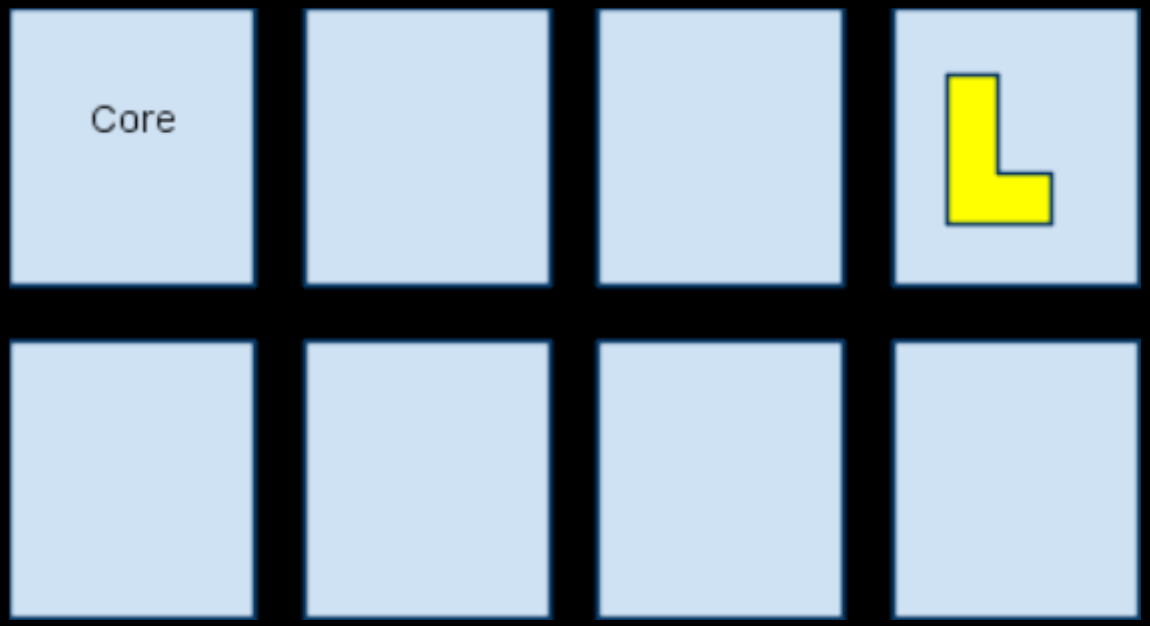


Architecture

One core initializes the lock, and acquires it. This core will manage this lock.

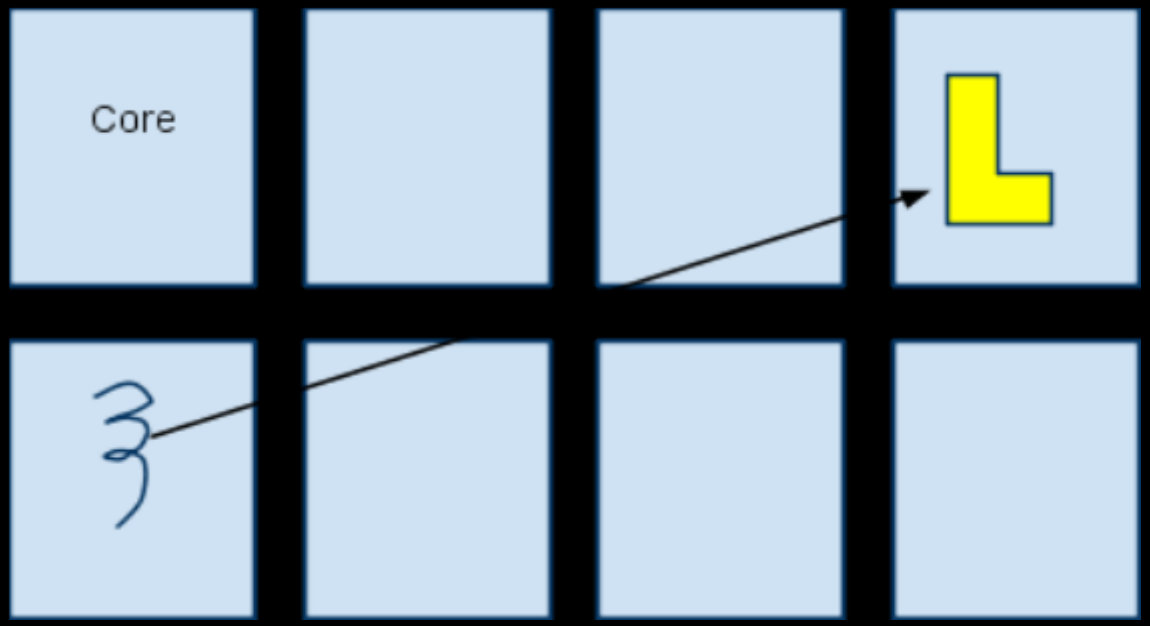
Architecture

This core does work using the lock, without having to check if anyone else needs it. If they do...



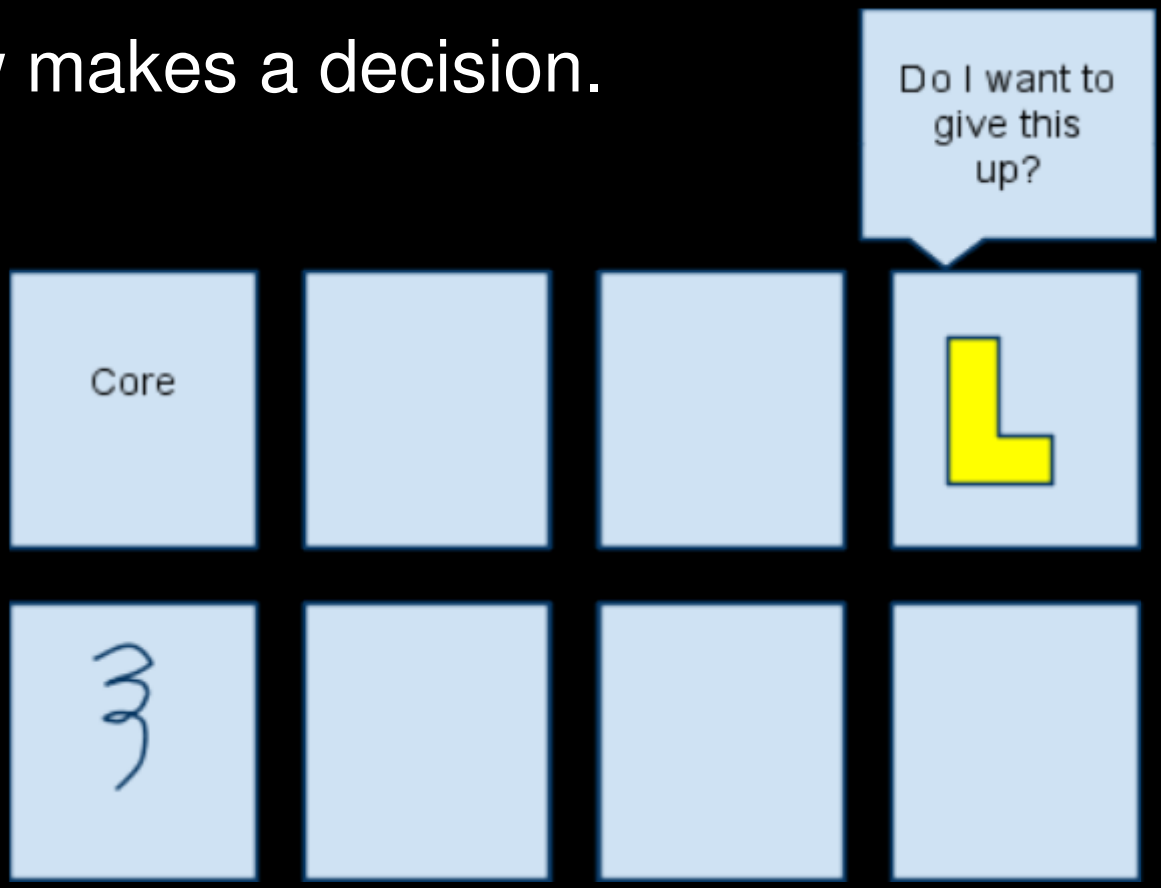
Architecture

They send an interrupt.



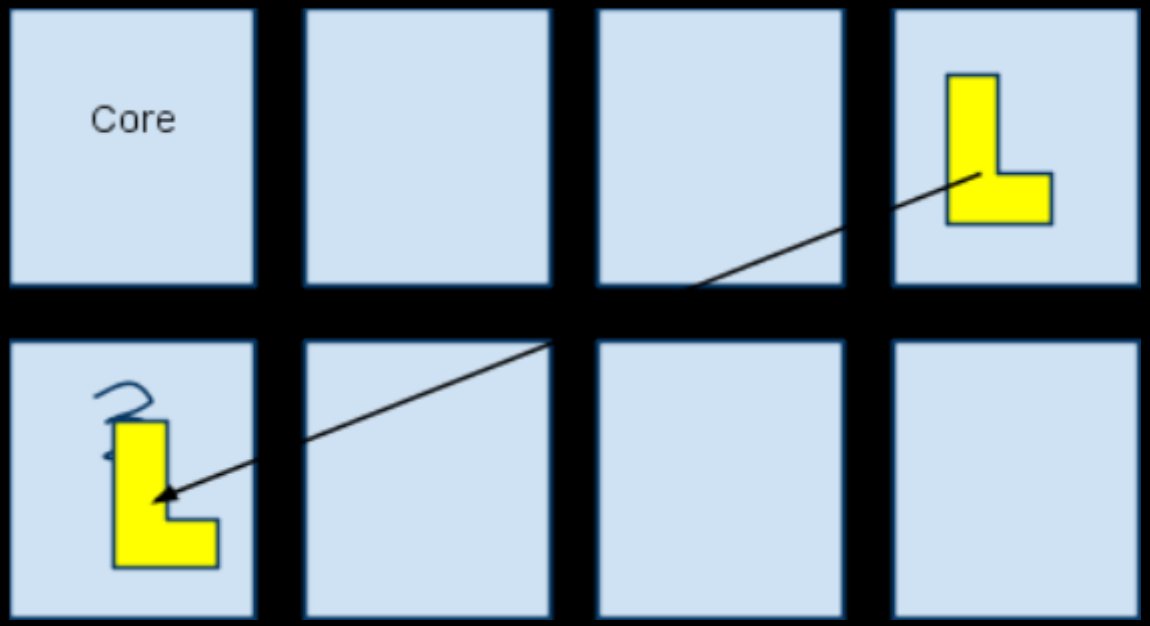
Architecture

Manager now makes a decision.



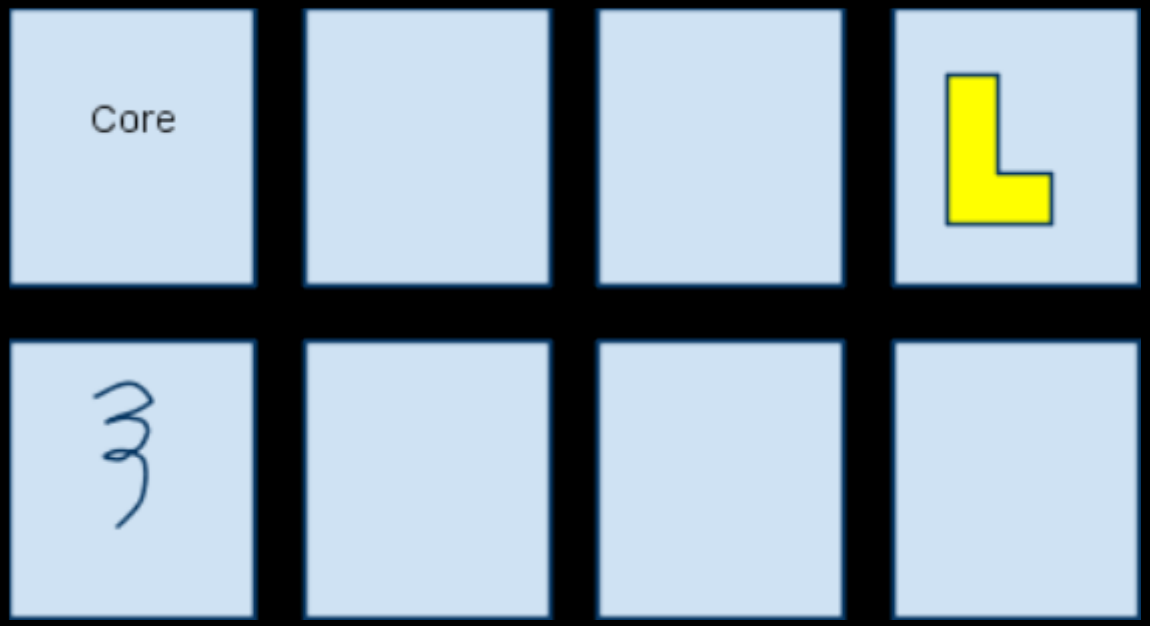
Architecture

They can give it up. The other thread can now do business.



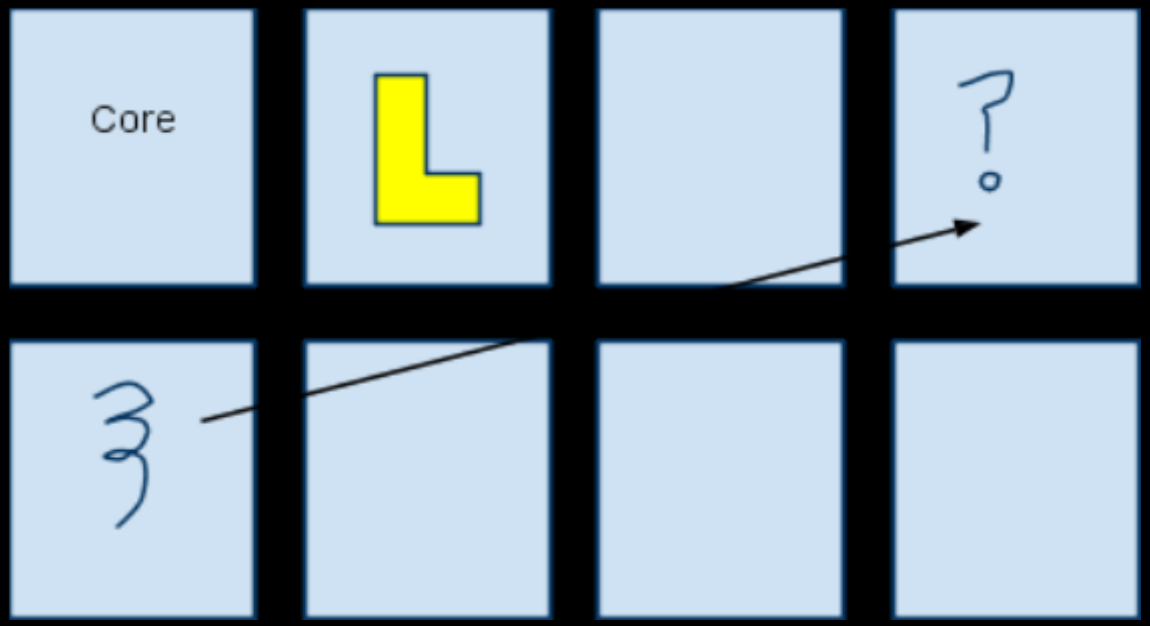
Architecture

Or they can hold it, and give it up when they're done.



Architecture

Lastly, the lock may be out. In this case, it may be revoked or the asking process may wait.



Architecture

- This is all extremely high-level.
  - There are a lot of details.
    - Revocation is hard.
      - Really hard.
    - Moving Lock Managers
    - What to do while waiting
      - We can work. Spinlocks can't. Semaphores can.
      - We'll be interrupted. Semaphores will not.
    - Many more.

Assuming this works, what are the tradeoffs?

Tradeoffs

	Latency	Startup	Polling Cost	Switch Cost	Schedulable	User Level
Spinlock (MCS) polling frequently	Low	Low	High	Low	No	Yes
Spinlock (MCS) polling infrequently	High	Low	Low	Low	No	Yes
Semaphore	High	High	None	High	yes	Yes
Interrupt Lock	Low	Medium	None	Medium	yes	No

- Two Key Use Cases:
  - Long-Held Spinlocks. Want low latency but do not want the high polling cost. Low-contention, but time critical.
  - Scheduling in Real-Time OS. MCS Locks use linked lists, these limit scheduling. Semaphores are too heavy-weight, Interrupt Locks can live in the middle for contentious locks.

Validation of Case 1

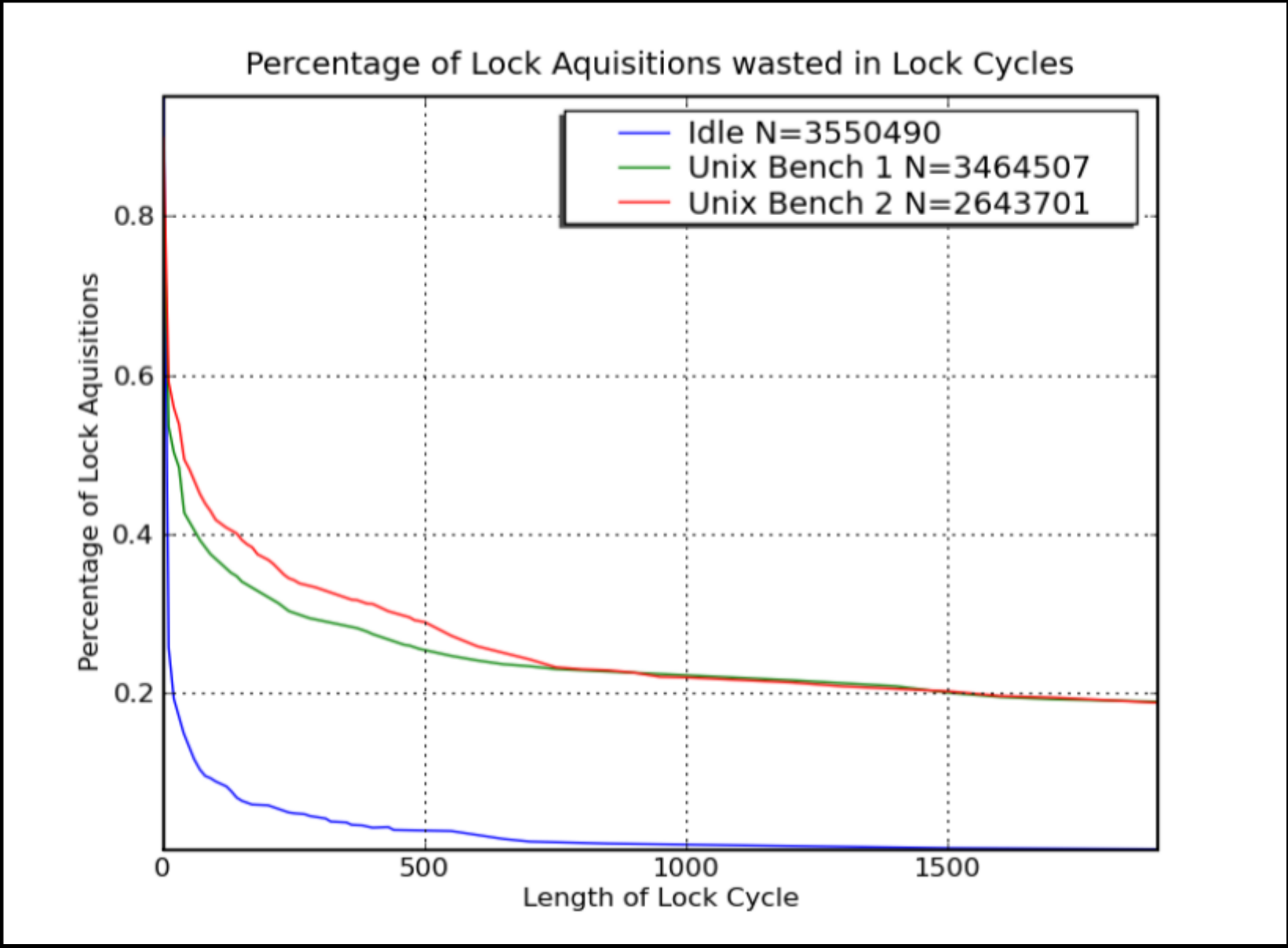
- This is built off a number of assumptions.
  - IPIs are cheap enough to compete.
  - There are processes that hold locks for long periods.
    - These processes constantly unlock for latency

Measurements

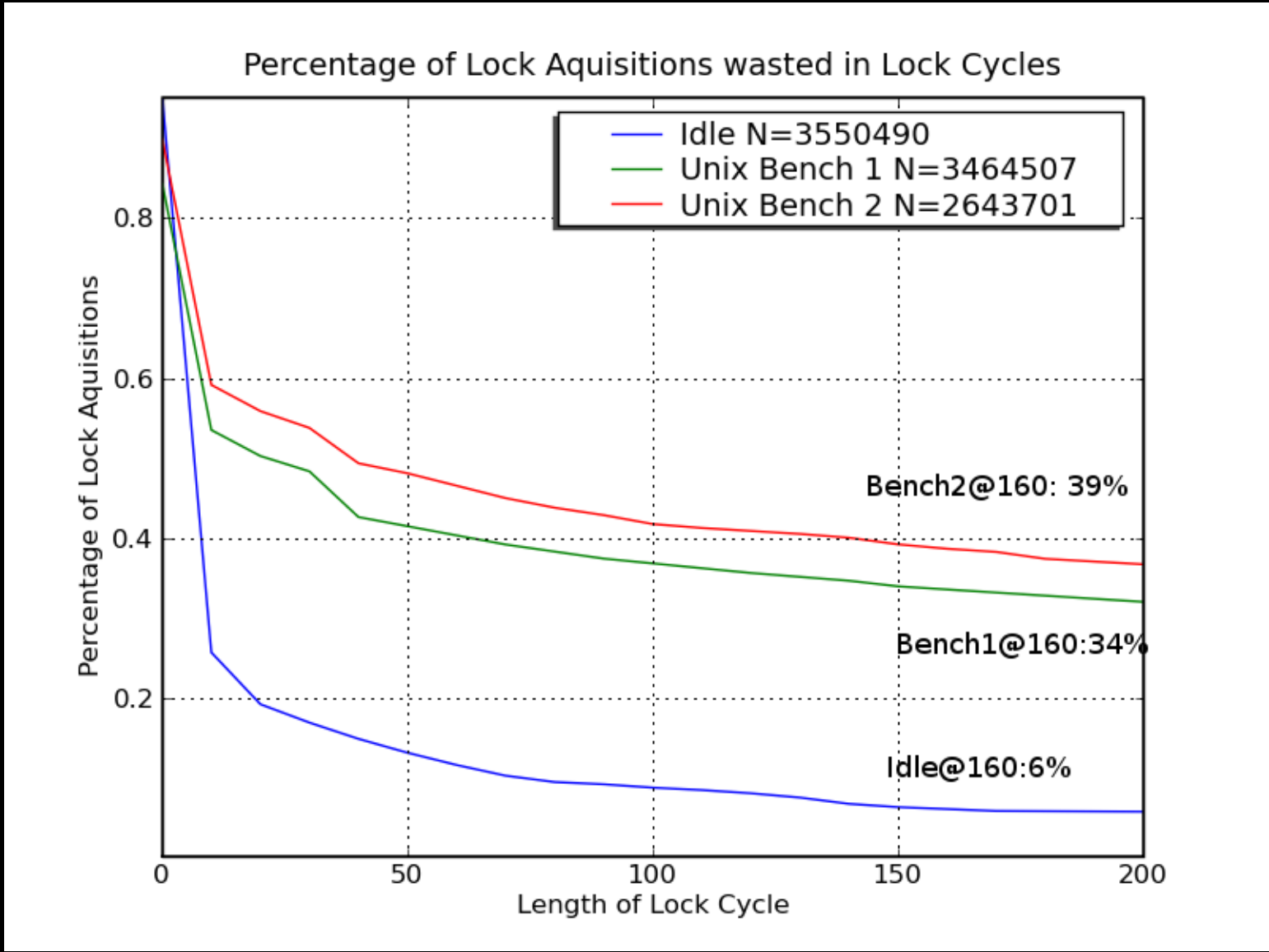
- Basic Spinlocks take 5-~20 cycles to complete, without contention.
  - Linux 2.6.29
  - Texas
- IPI Takes ~750 cycles
  - Nihalem processor, shared bus
- So we can expect [~(750*2) + Lock] cycles to use an Interrupt Lock.
  - 1600/(2*5) = 160 “Acquire/Release” lock cycles needed for us to win.

How often does the a kernel level process go through so many cycles?

Measurements



Measurements



Measurements

- Idle
  - Max: 3150 in a row
    - ~30000 and 60000 wasted cycles
  - 11 Processes contended for 3150 Lock
- Unix Bench
  - Max: 73347 (2nd place 43998)
    - ~700k and 1400k wasted cycles
  - Contention:
    - 3 Processes Contended for the 73347 Lock
    - 3 Processes Contended for the 43998 Lock

Conclusions

- Seems to be a good idea
  - Should provide performance benefit
  - Need to validate tradeoffs more clearly.
  - Scalability still an open issue: should be good.
- Implementation is reality
  - I avoid reality whenever I can.
  - I can't.
  - Left as future work.

Questions?