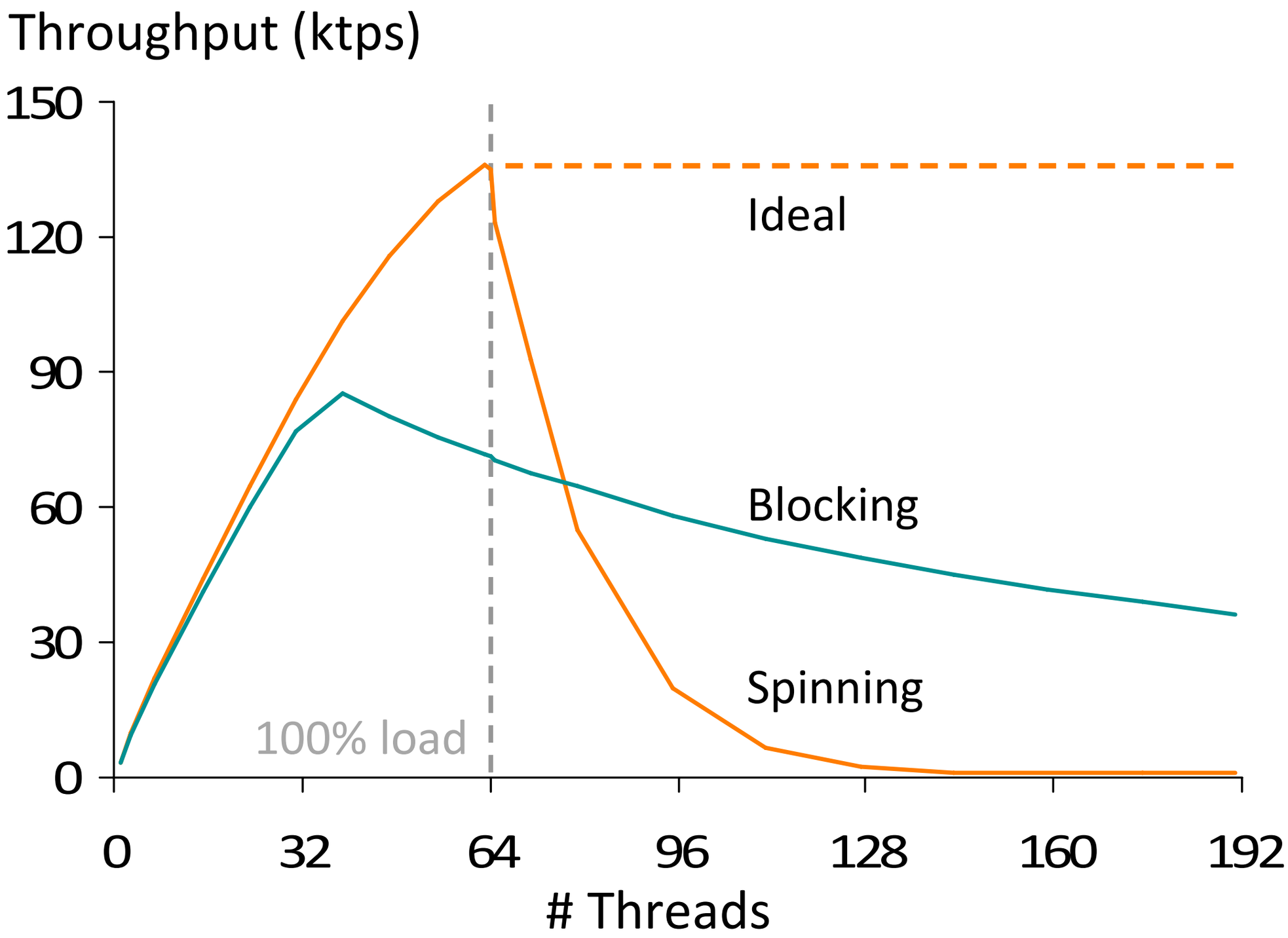
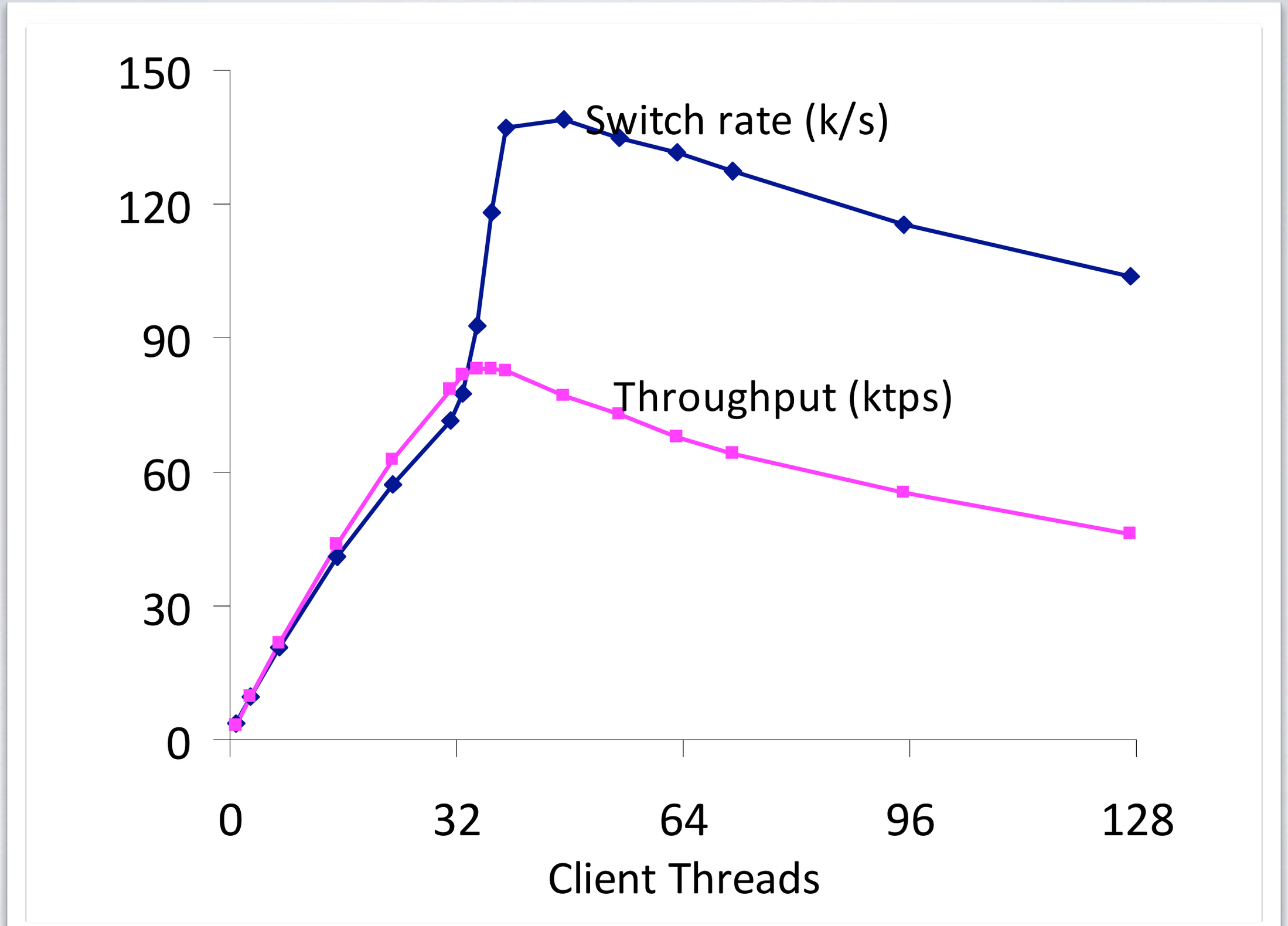
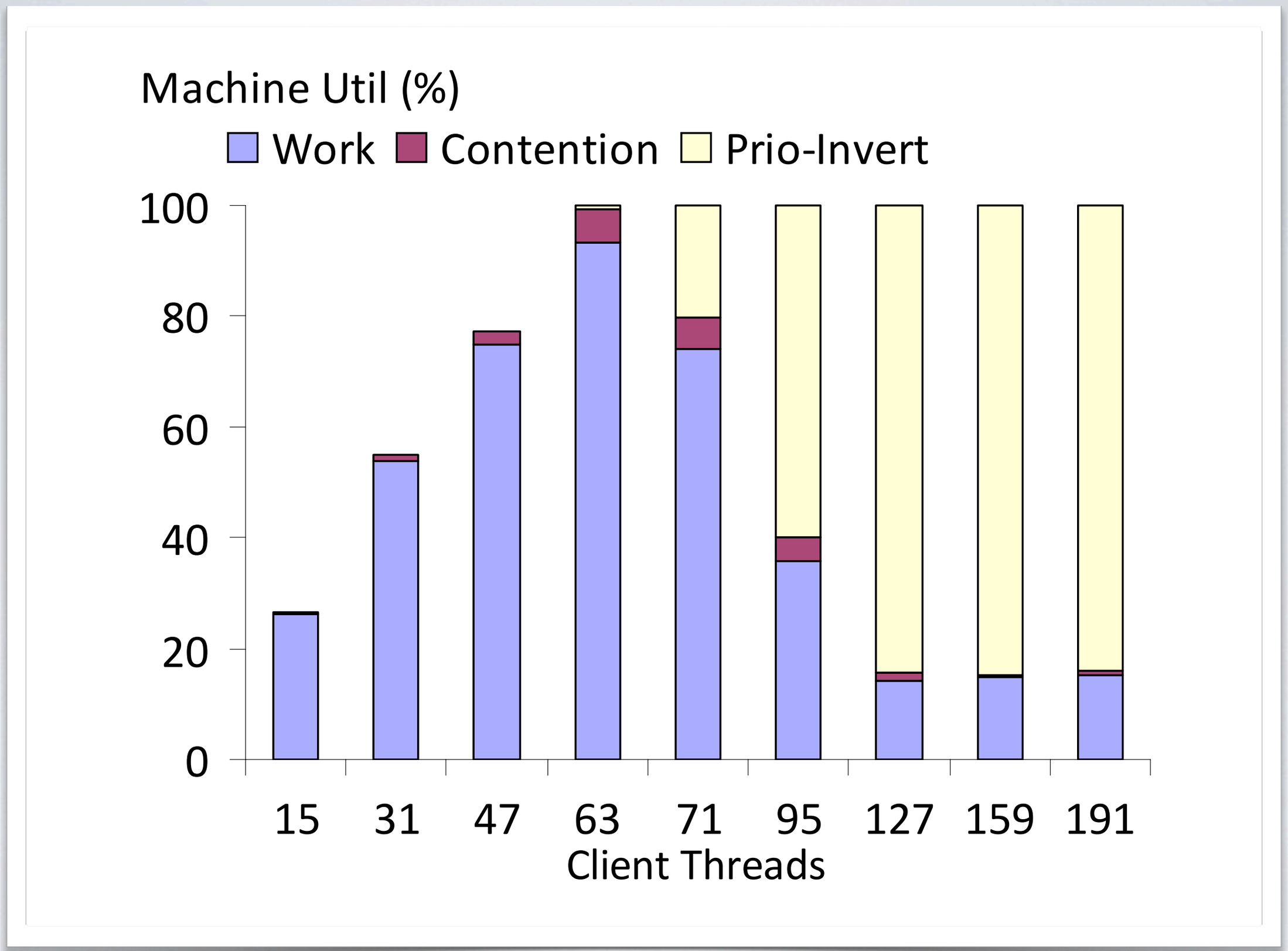
# DECOUPLING CONTENTION MANAGEMENT FROM SCHEDULING

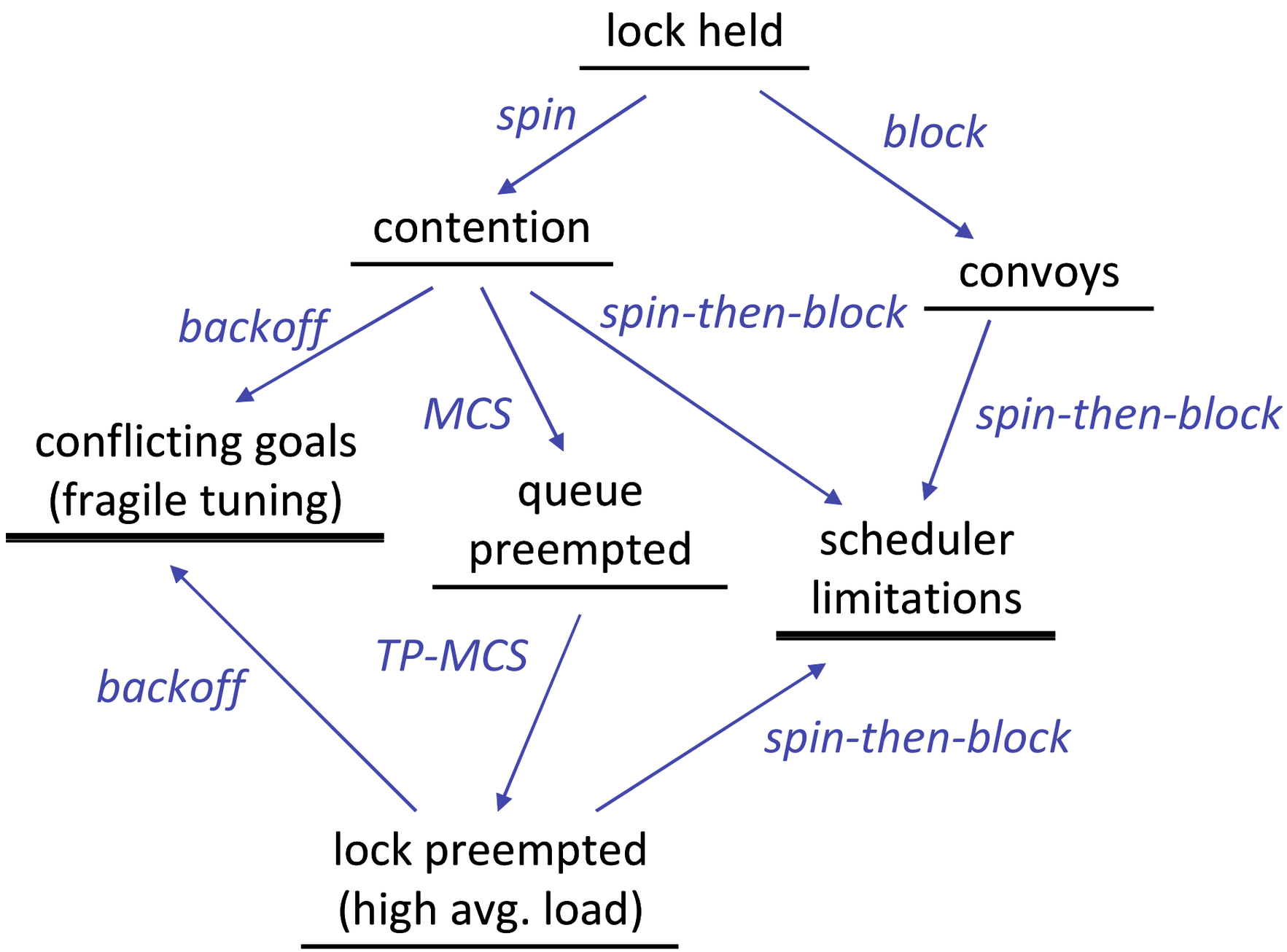
F. Ryan Johnson
Anastasia Ailamaki

Radu Stoica
Todd C. Mowry


Throughput (ktps)
150
120
90
60
30
0
Ideal
Blocking
Spinning
100% load
0
32
64
96
128
160
192
# Threads

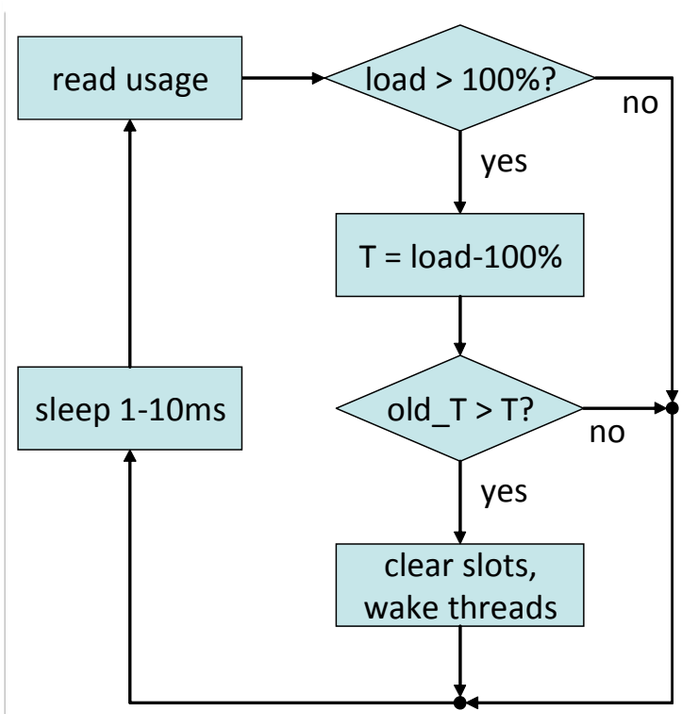

150
120
90
60
30
0
Switch rate (k/s)
Throughput (ktps)
0
32
64
96
128
Client Threads

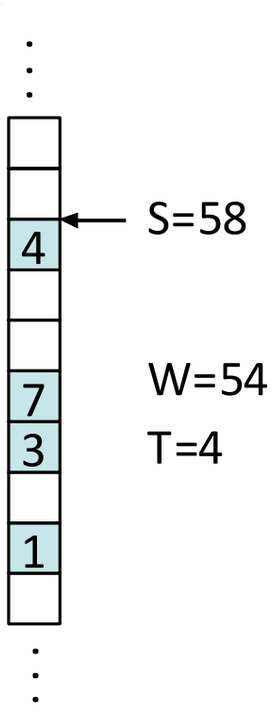

Machine Util (%)
Work
Contention
Prio-Invert
100
80
60
40
20
0
15
31
47
63
71
95
127
159
191
Client Threads

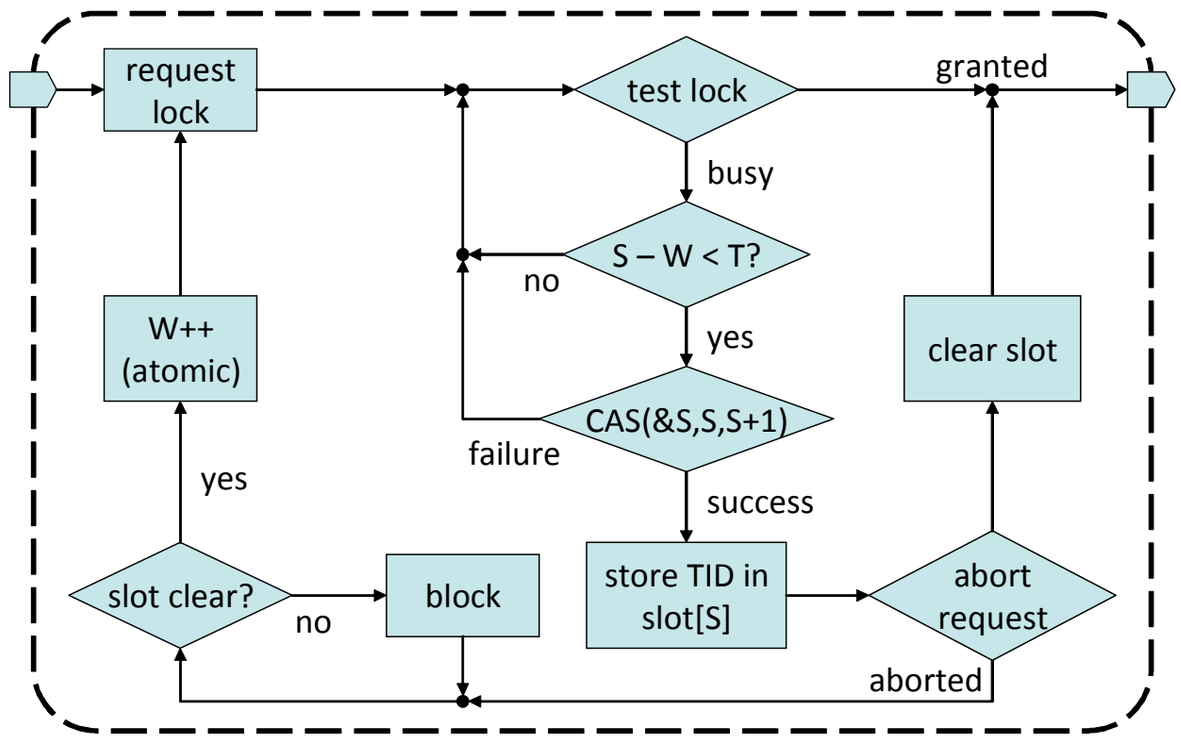

lock held
spin
block
contention
convoys
backoff
spin-then-block
MCS
spin-then-block
conflicting goals
(fragile tuning)
queue
preempted
scheduler
limitations
TP-MCS
backoff
spin-then-block
lock preempted
(high avg. load)

# DECOUPLING

| **contention management** | **load control** |
| --- | --- |
| spinning | blocking |
| fast lock handoff | scheduler overhead |
| on critical path | off critical path |


read usage
load > 100%?
no
yes
T = load-100%
old_T > T?
no
yes
clear slots,
wake threads
sleep 1-10ms


4
7
3
1
S=58
W=54
T=4


request
lock
test lock
granted
busy
S – W < T?
no
yes
W++
(atomic)
CAS(&S,S,S+1)
failure
success
clear slot
yes
slot clear?
no
block
store TID in
slot[S]
abort
request
aborted

# OS SUPPORT

- schedule daemon periodically & independent from system tick

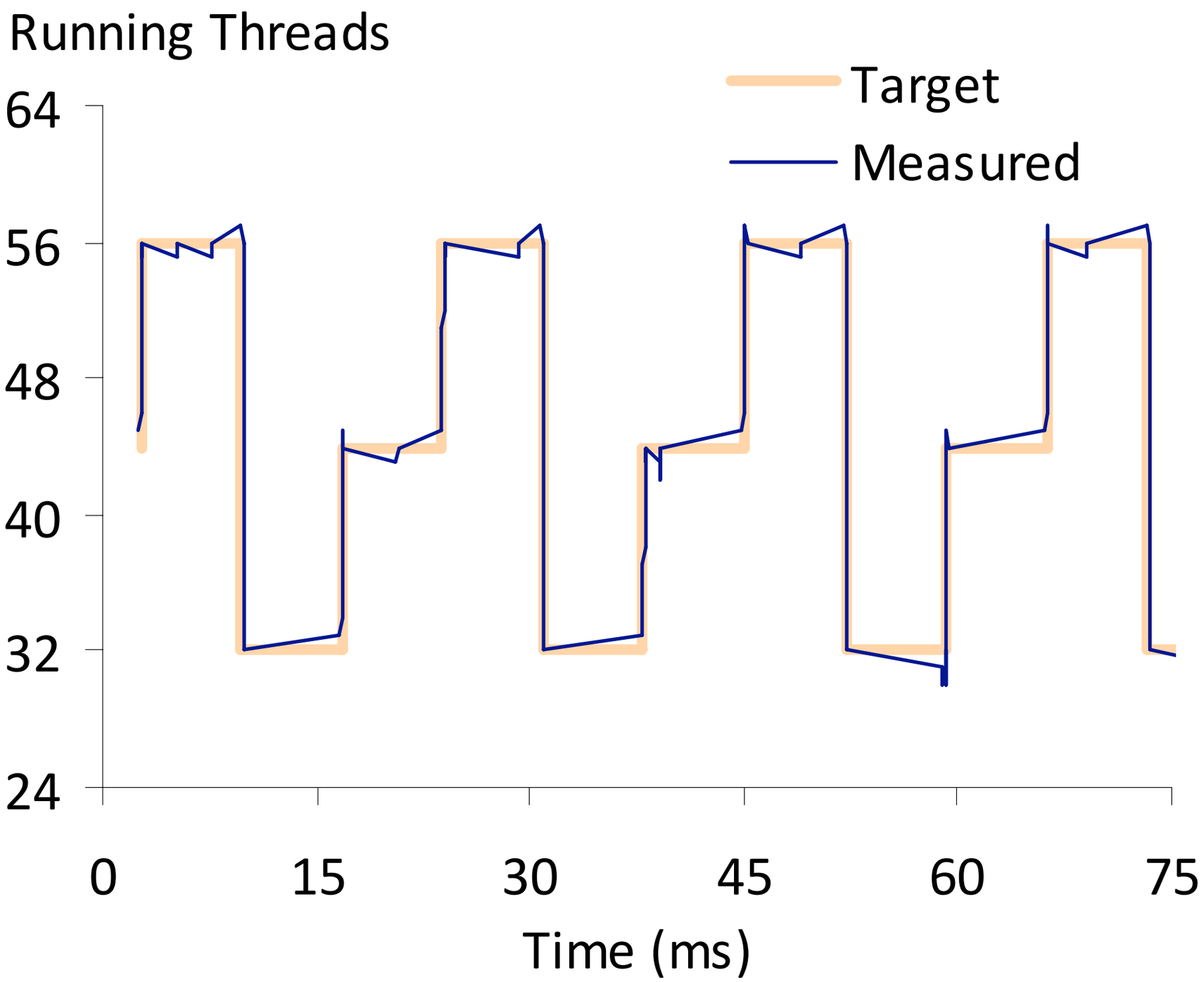
**high-res timer**

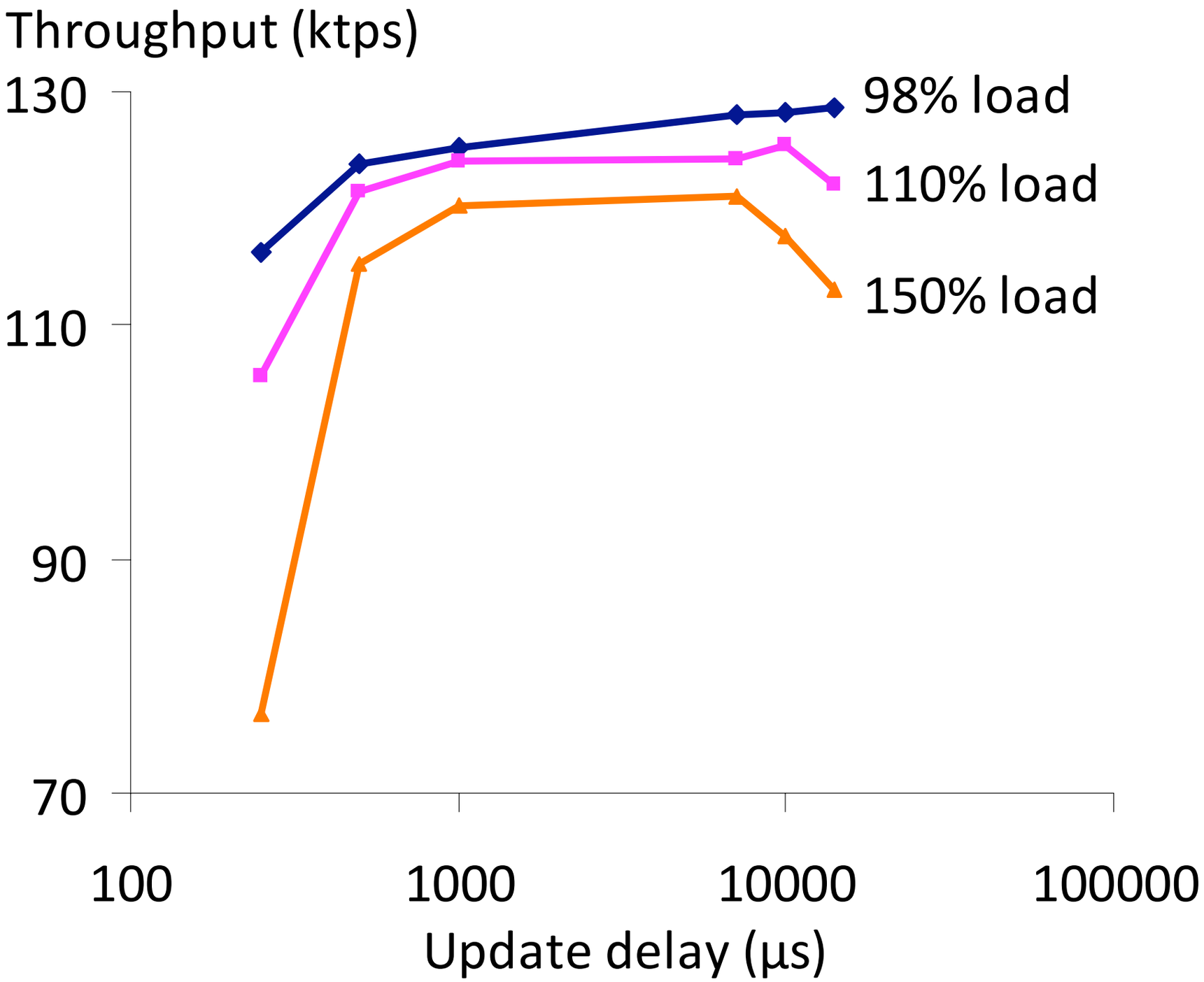
- measure load accurately and with high resolution

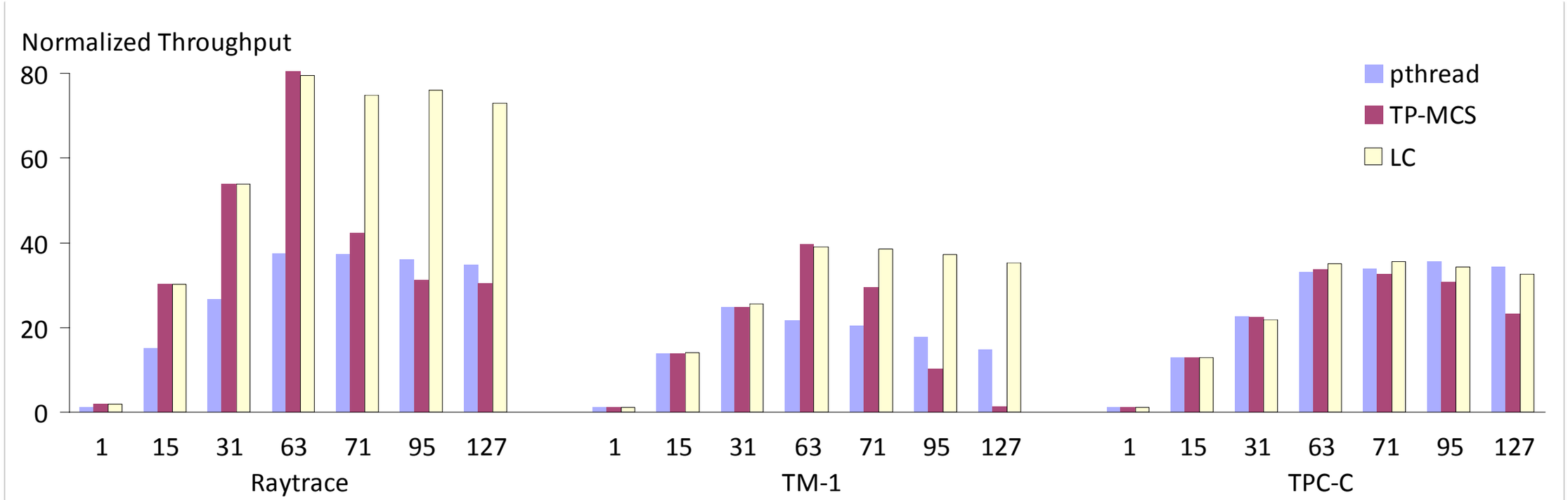
**Solaris microstate**

- efficient deschedule and wake

**futex, lwp_park**


Running Threads
Target
Measured
64
56
48
40
32
24
0
15
30
45
60
75
Time (ms)


Throughput (ktps)
130
110
90
70
98% load
110% load
150% load
100
1000
10000
100000
Update delay (µs)


Normalized Throughput
80
60
40
20
0
pthread
TP-MCS
LC
1
15
31
63
71
95
127
Raytrace
TM-1
TPC-C

# LIMITATIONS

- large, transient changes in load
  - apply some control theory
- nested critical sections
- load measurement cost linear in thread count
  - load-change notification instead of polling

# CONCLUSION

- unwanted interaction between scheduling and contention management lead to poor performance
- decouple the two
  - use spinning in response to contention
  - use blocking to control the number of runnable threads
- implementation is transparent to the application

# DISCUSSION

- Did you fully understand the presented algorithm?
- What about the increased energy consumption due to spinning?
- Do state-of-the-art spinlocks use MONITOR-MWAIT?
- Is this an argument for parallelism, where the number of threads is determined by the OS, not the application?