Version 23, *18* sep 2024 19:33

# Time

## (and how to get rid of it)

*research from the parmington foundation*

Gus Björklund,

Head Groundskeeper,

The Parmington Foundation


PUG CHALLENGE

**Westin Hotel**

**Waltham MA, USA**

**30 sep 2024**
**02 oct 2024**

sign on a building in London

# Notices

Please ask questions as we go

YMMV (Your mileage may vary, transportation, meals, and accomodations not included).

"Time is what we want most,
but... what we use worst."

-- William Penn

# Time you should know

| thing | time |
|---|---|
| Read or write L1 cache memory | 0.5 ns |
| Branch mispredict | 5 ns |
| Mutex lock/unlock | 100 ns |
| Read 1 byte from main memory | 100 ns |
| 1 microsecond | 1,000 ns |
| Send 2K bytes over 1 Gbps network | 20,000 ns |
| Read 1 MB sequentially from memory | 250,000 ns |
| Round trip packet within same datacenter | 500,000 ns |
| 1 millisecond | 1,000,000 ns |
| Disk seek | 10,000,000 ns |
| Read 1 MB sequentially from network | 10,000,000 ns |
| Read 1 MB sequentially from disk | 30,000,000 ns |
| Send packet CA -> Netherlands -> CA | 150,000,000 ns |
| 1 second | 1,000,000,000 ns |

(from Jeff Dean @ google)

*More numbers you should know.*
*Trust the big B !!!*

| Layer | Time (sec) | # of Recs | # of Ops | Time per op (nsec) | Relative |
|---|---|---|---|---|---|
| **4GL to –B** | 0.96 | 100,000 | 203,473 | 4,718 | 1 |
| -B to FS Cache | 10.24 | 100,000 | 26,711 | 383,362 | 81 |
| FS Cache to SAN | 5.93 | 100,000 | 26,711 | 222,006 | 47 |
| -B to SAN Cache** | 11.17 | 100,000 | 26,711 | 418,180 | 89 |
| SAN Cache to Disk | 200.35 | 100,000 | 26,711 | 7,500,655 | 1590 |
| -B to Disk | 211.52 | 100,000 | 26,711 | 7,918,834 | 1678 |

*** Used concurrent IO to eliminate FS cache effects*

actual measurements made by Tom Bascom on customer AIX system

# Test environment: ATM

## Same as the one in Secret Bunker 2017

- but database is only 12 GB instead of 100

## Simulates ATM withdrawal transaction

## 150 concurrent users

- execute as many transactions as possible in given time
- result reported as "transactions per second".

## Highly update intensive

- fetch 3 rows
- update 3 rows
- create 1 row with 1 index entry

# our test machine, bunker15

4 quad-core 2.4 GHz intel processors

64 GB memory

16 x 300 GB 10,000 rpm sas drives in RAID 10

Centos 6 Linux (2.6.32-504.12.2.el6.x86_64)

OpenEdge 11.7

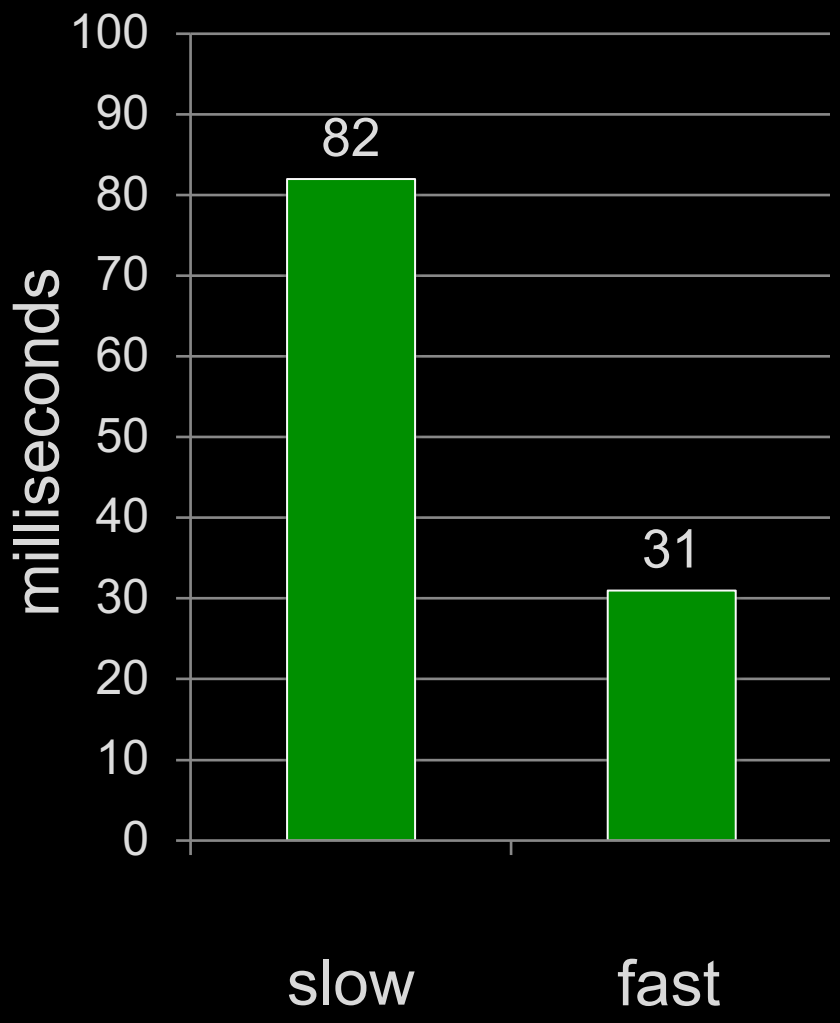
ATM 7

initial configuration

OE 11.7
database size 12 GB
150 self-serving clients

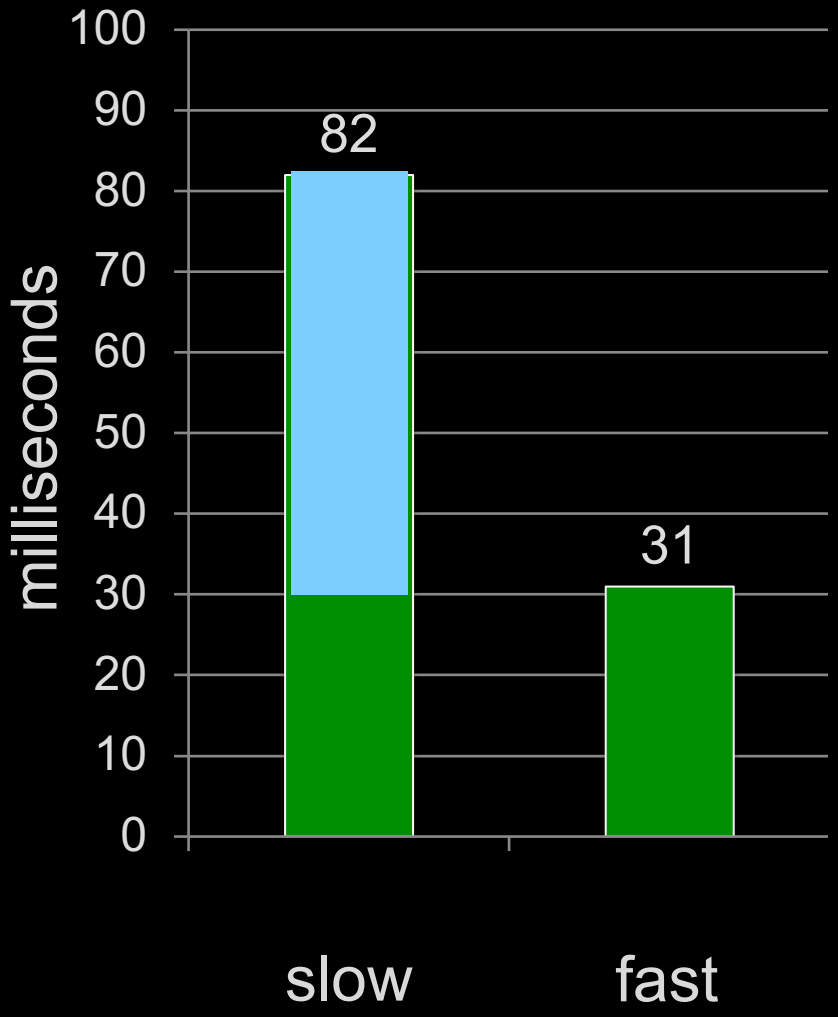
-db atm
-maxAreas 50
-omsize 4096
-n 200
-spin 5000
-L 10240
-B 64000
-bibufs 64

let's run some tests


ATM transaction duration
(millseconds)
100
90
80
70
60
50
40
30
20
10
0
milliseconds
82
31
slow
fast

# ATM transaction duration

## what is going on for 51 of 82 milliseconds ?



nothing at all.

for more than half the time.

nothing at all.

for more than half the time.

can we get rid of this "nothing" time ??

The ATM transaction does the following (for 150 users):

0) execute 4GL code

1) fetch records from db, reading from cache

2) generate BI notes

3) update and create records

4) create index entries

5) write to disk

6) get and release various kinds of locks

kinds of locks:

0) record locks

1) record-get locks

2) MTX lock

3) TXE lock

4) data buffer locks

5) bi buffer locks

6) Various latches

Latches are typically held for very short times.

maybe as little as 100 nanoseconds

on modern computers

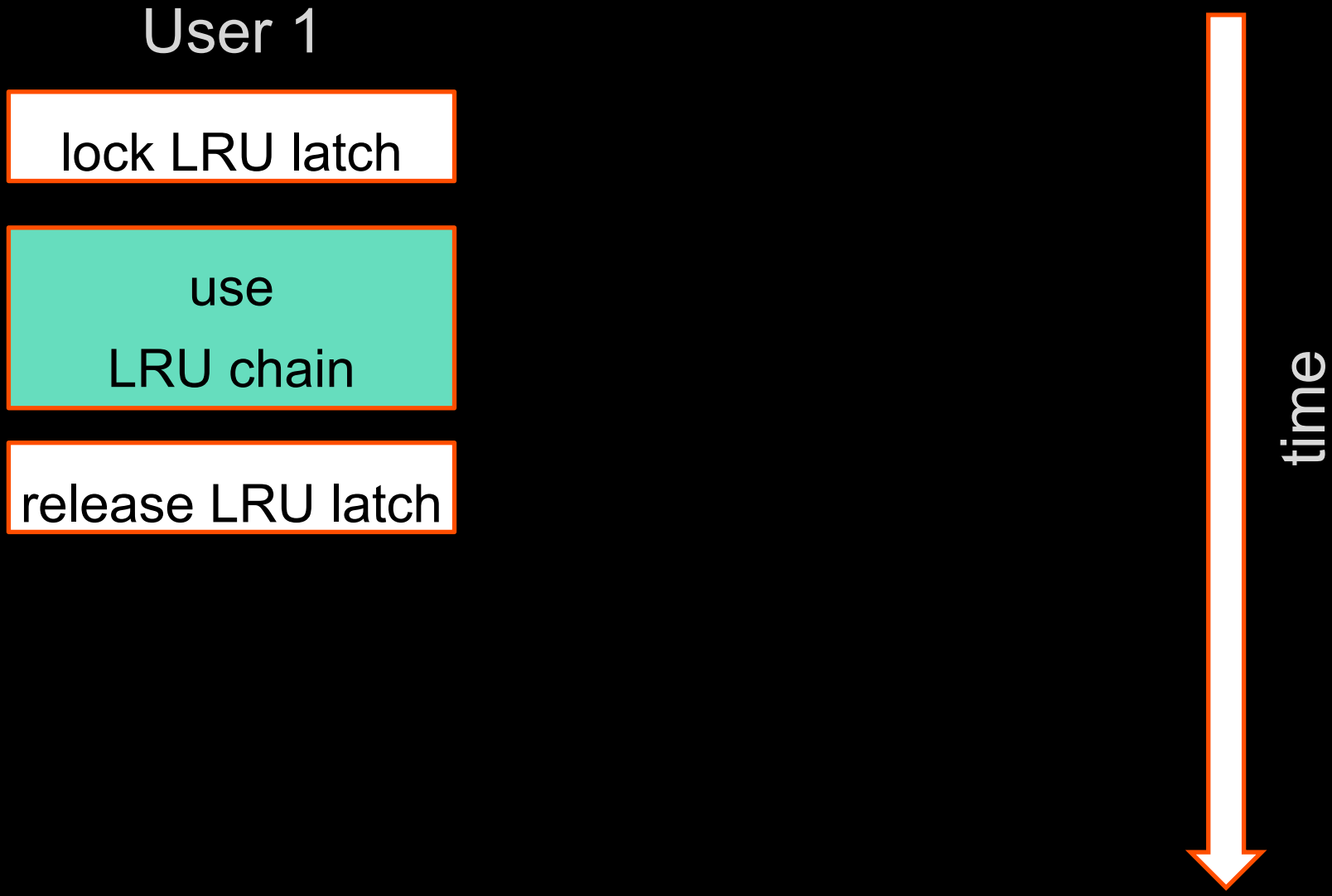
btw: with 2.4 GHz processor, 200 nsec is 479 clocks

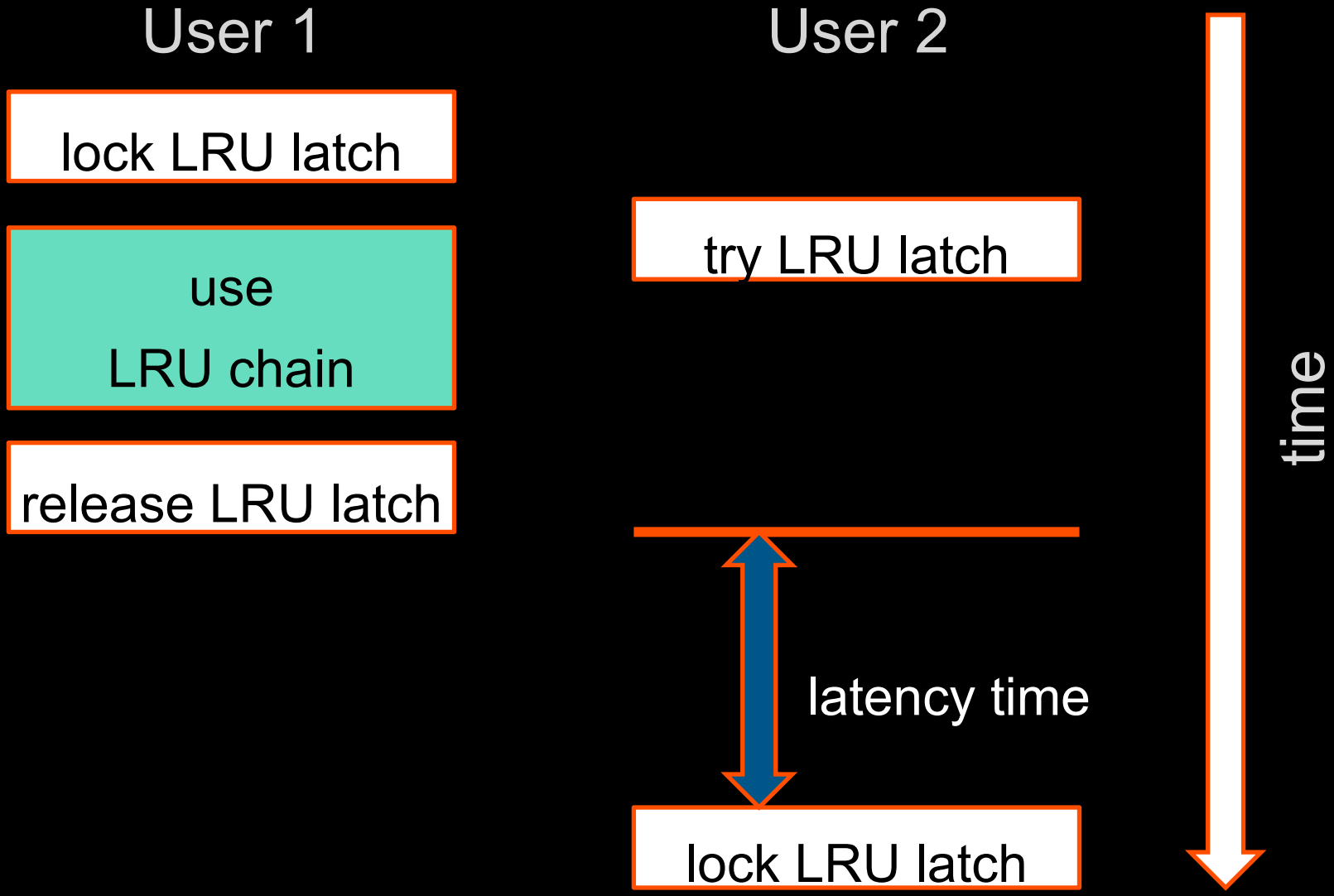
Lock latency:

time when lock holder releases lock

until waiting acquirer has locked it.

No useful work done.


User 1
lock LRU latch
use
LRU chain
release LRU latch
time


User 1
User 2
lock LRU latch
use
LRU chain
release LRU latch
try LRU latch
latency time
lock LRU latch
time

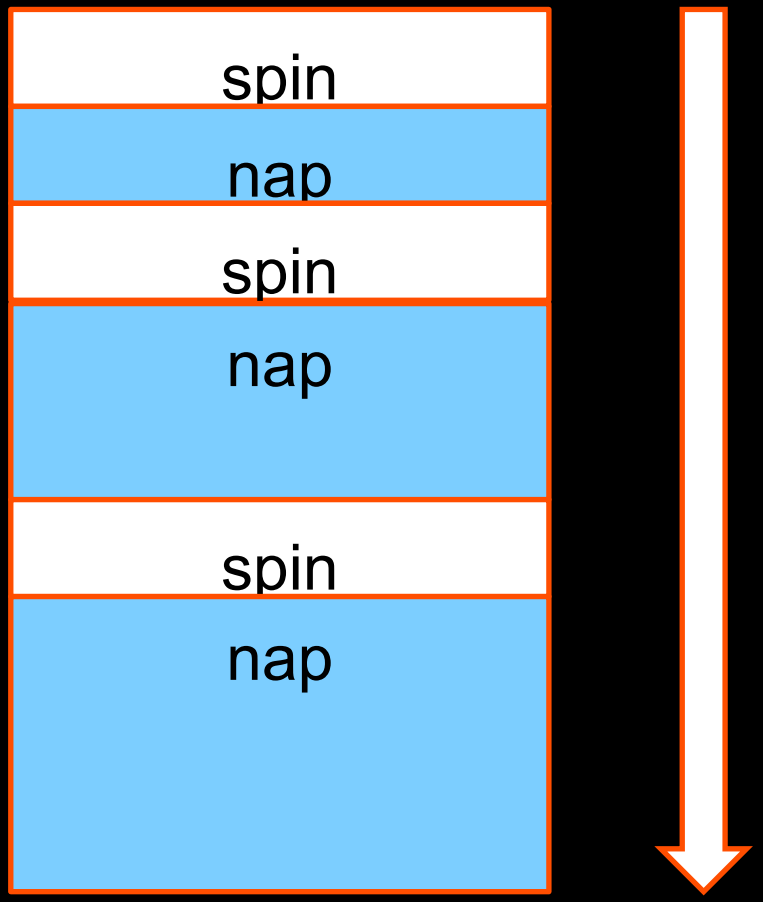
Spinlock latches:

test and set
spin and test
take a nap
spin and test
nap longer
spin and test
nap even longer

-spin

-nap

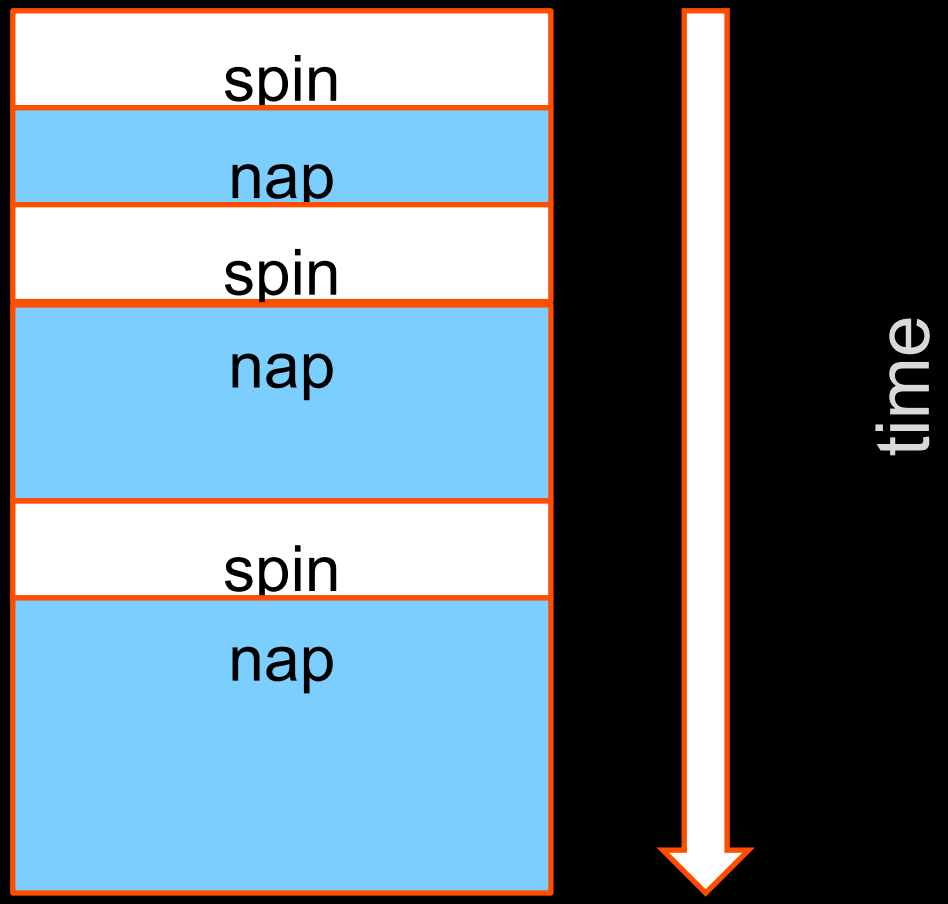
-napmax



-spin limits duration of white boxes

-nap sets minimum duration of blue boxes

-napmax sets maximum duration of blue boxes

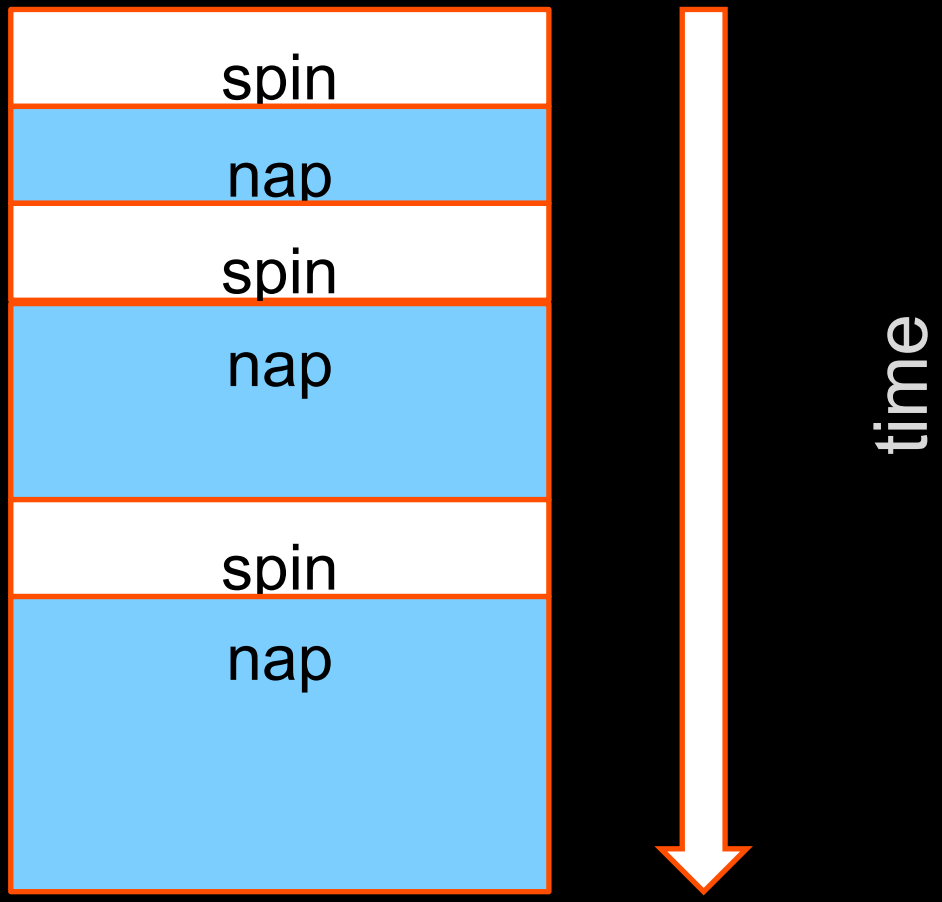


Tuning

-napmax

(max size of

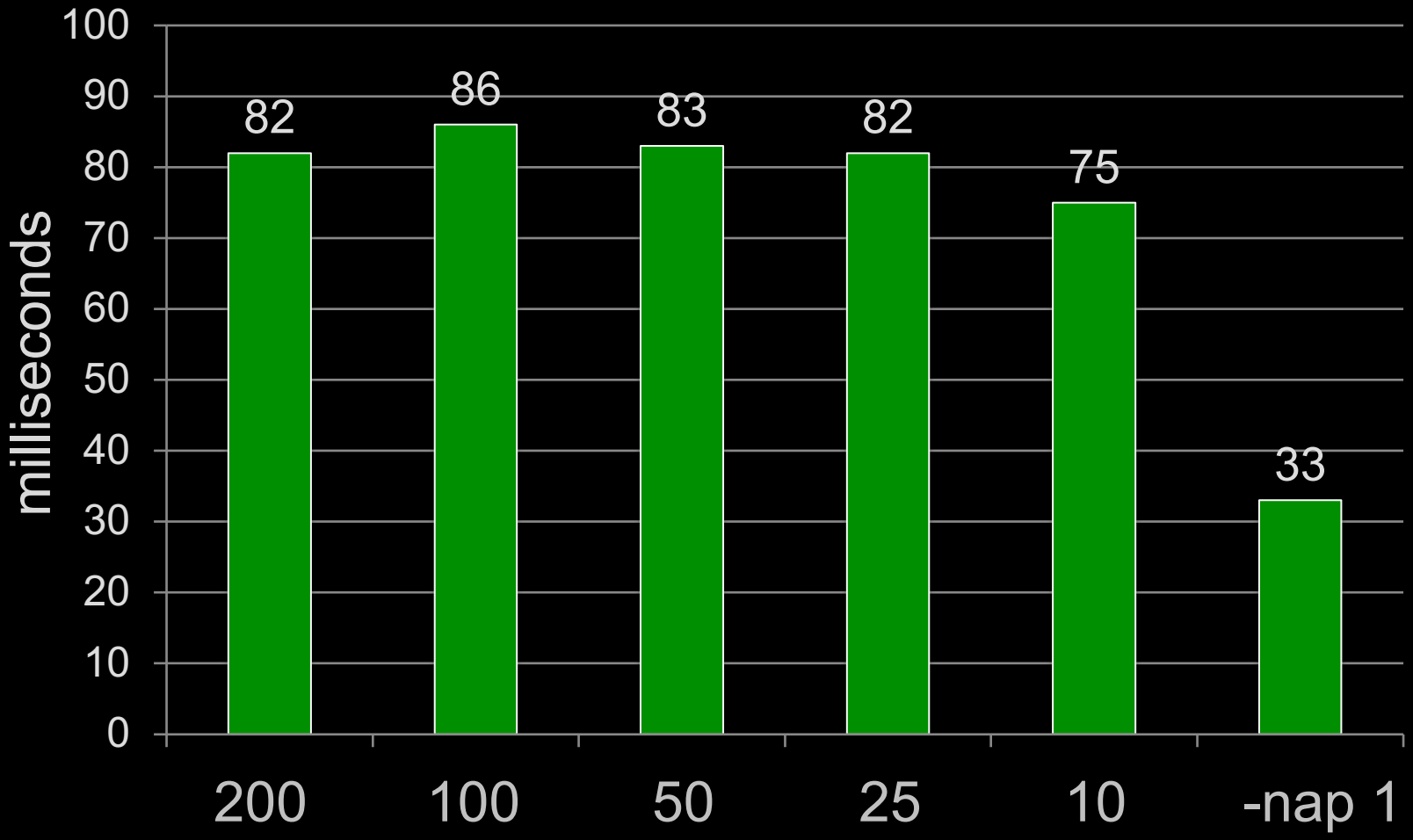
blue boxes)



The dawn rises only when the rooster crows.

Burmese proverb

# -spin 5,000  vary -napmax



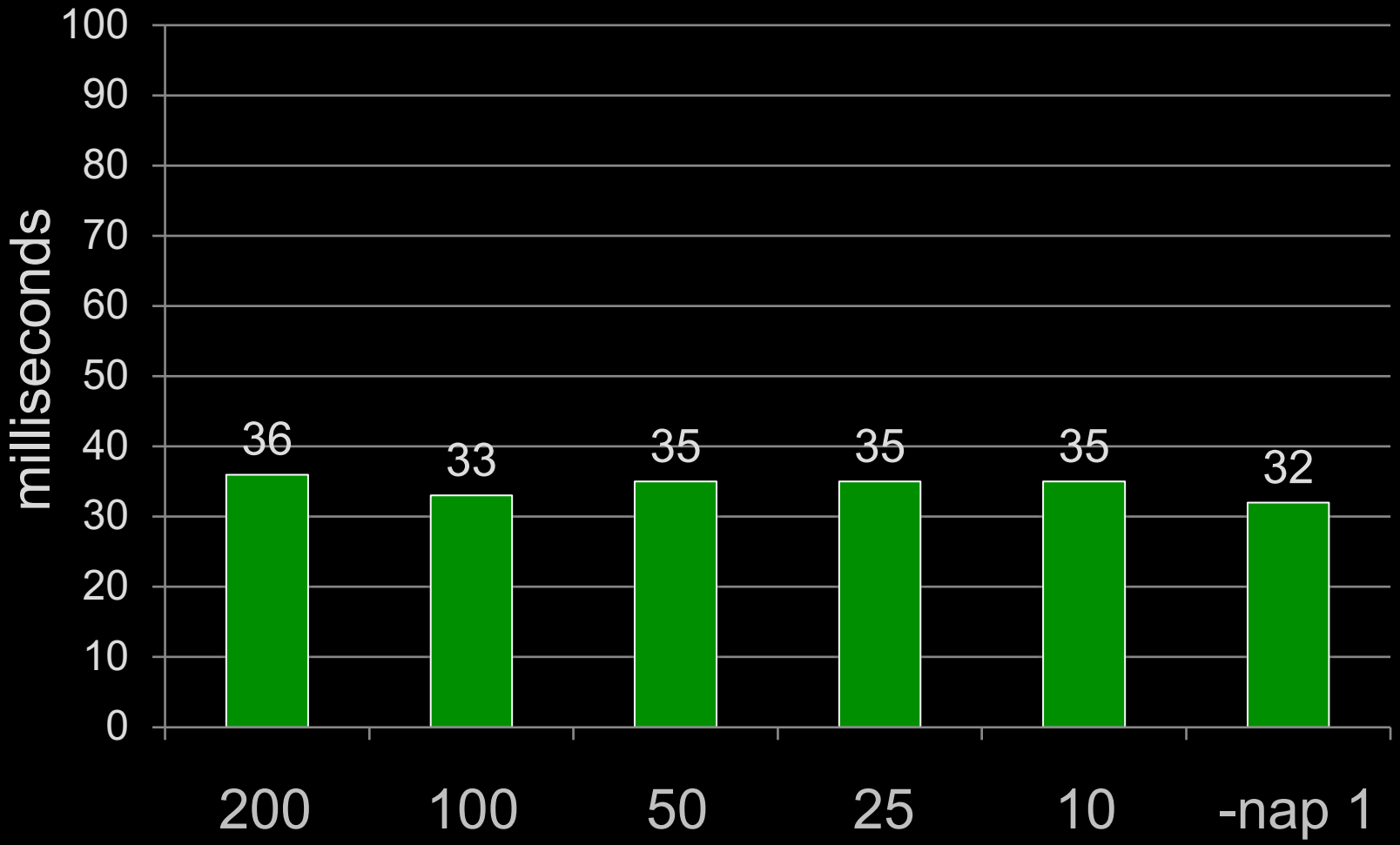
Change -spin to 50,000

Tune –napmax again

Does this mean everyone should set -napmax to 10?

NO!

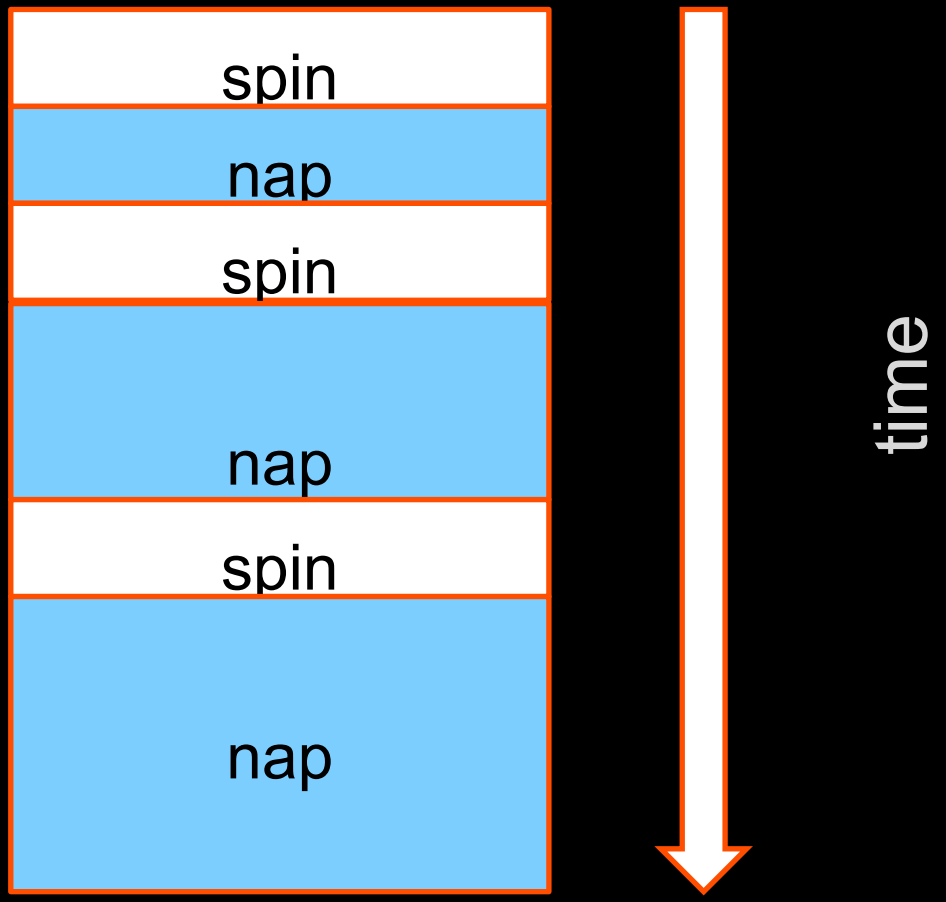
# -spin 50,000:  vary -napmax



Tuning

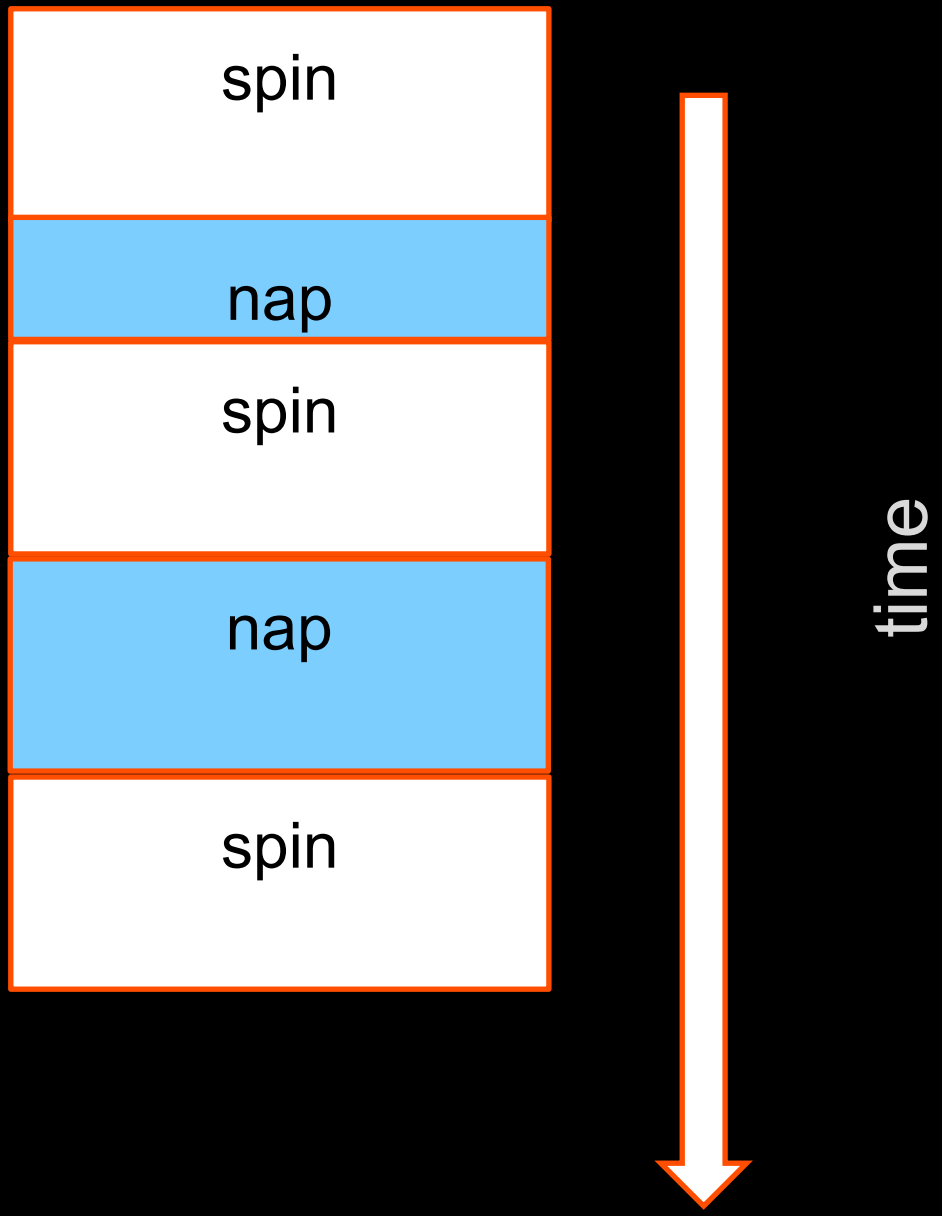
-spin

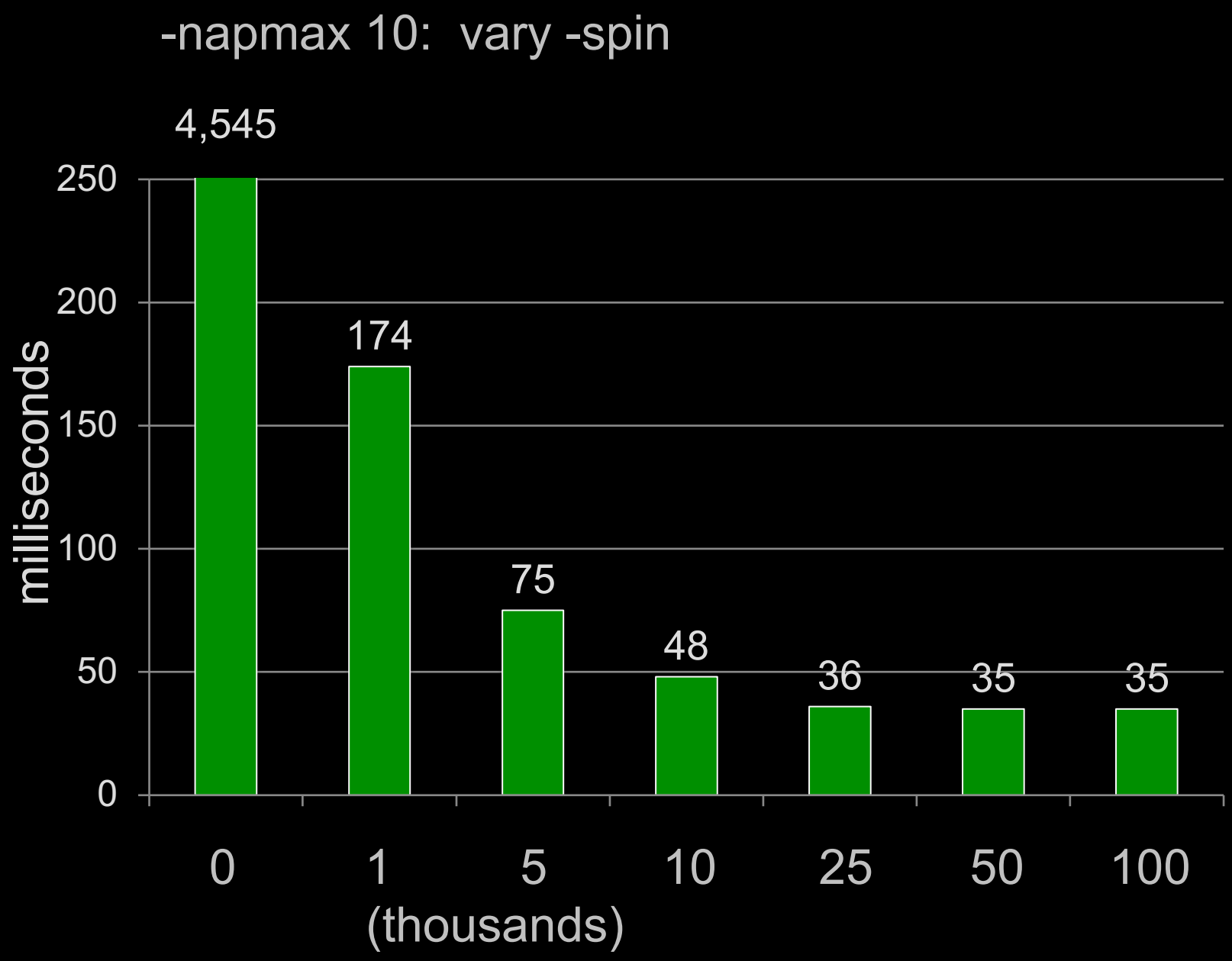
(white boxes)



Spin longer

before naps




-napmax 10: vary -spin
4,545
174
75
48
36
35
35
250
200
150
100
50
0
milliseconds
0
1
5
10
25
50
100
(thousands)

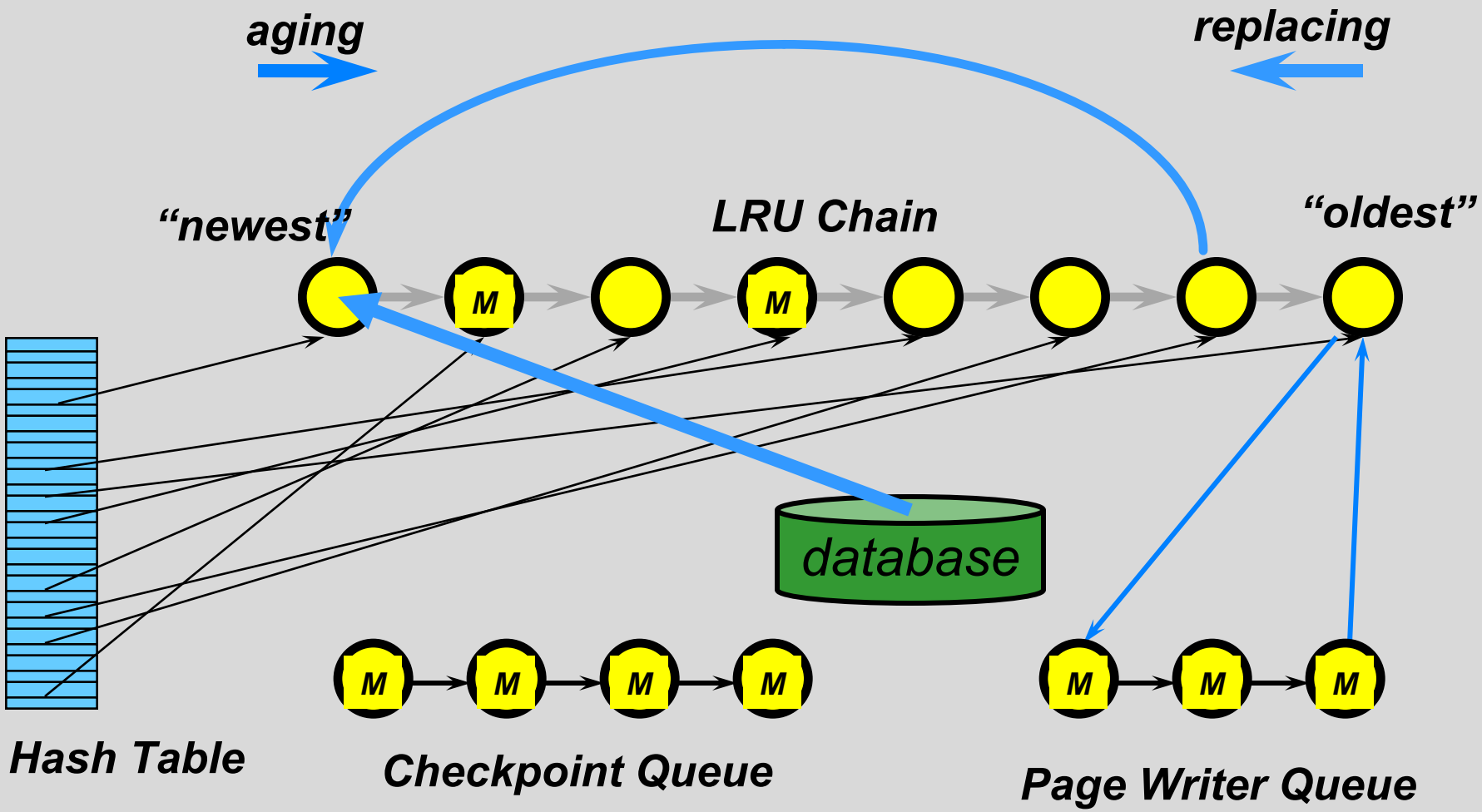
Longer nap times => higher latch latency

Higher spin => lower latch latency

Higher contention => higher latch latency

# LRU chain maintenance

# Buffer Pool LRU Chain



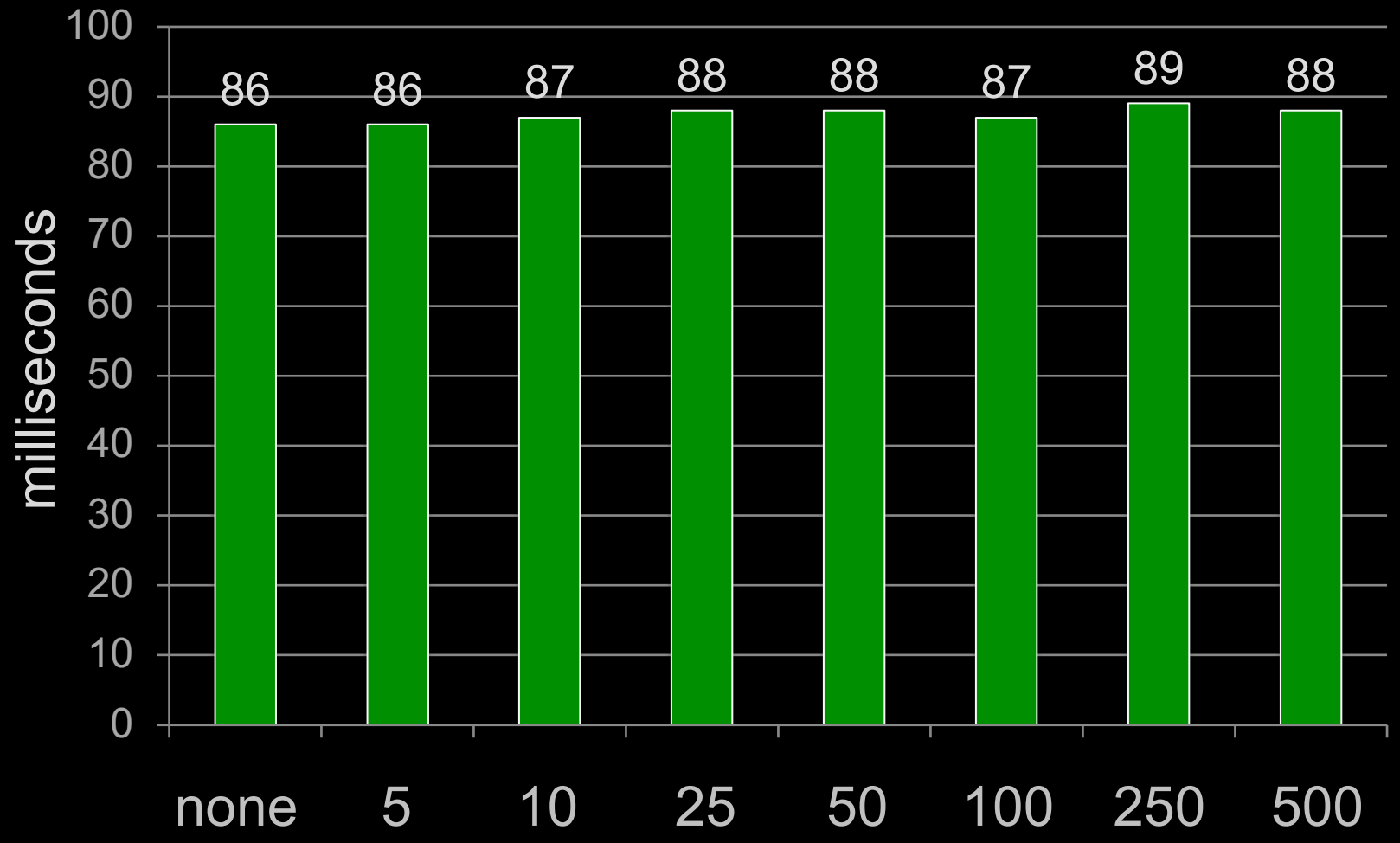
Every buffer access causes an LRU chain update

Can we reduce LRU chain overhead

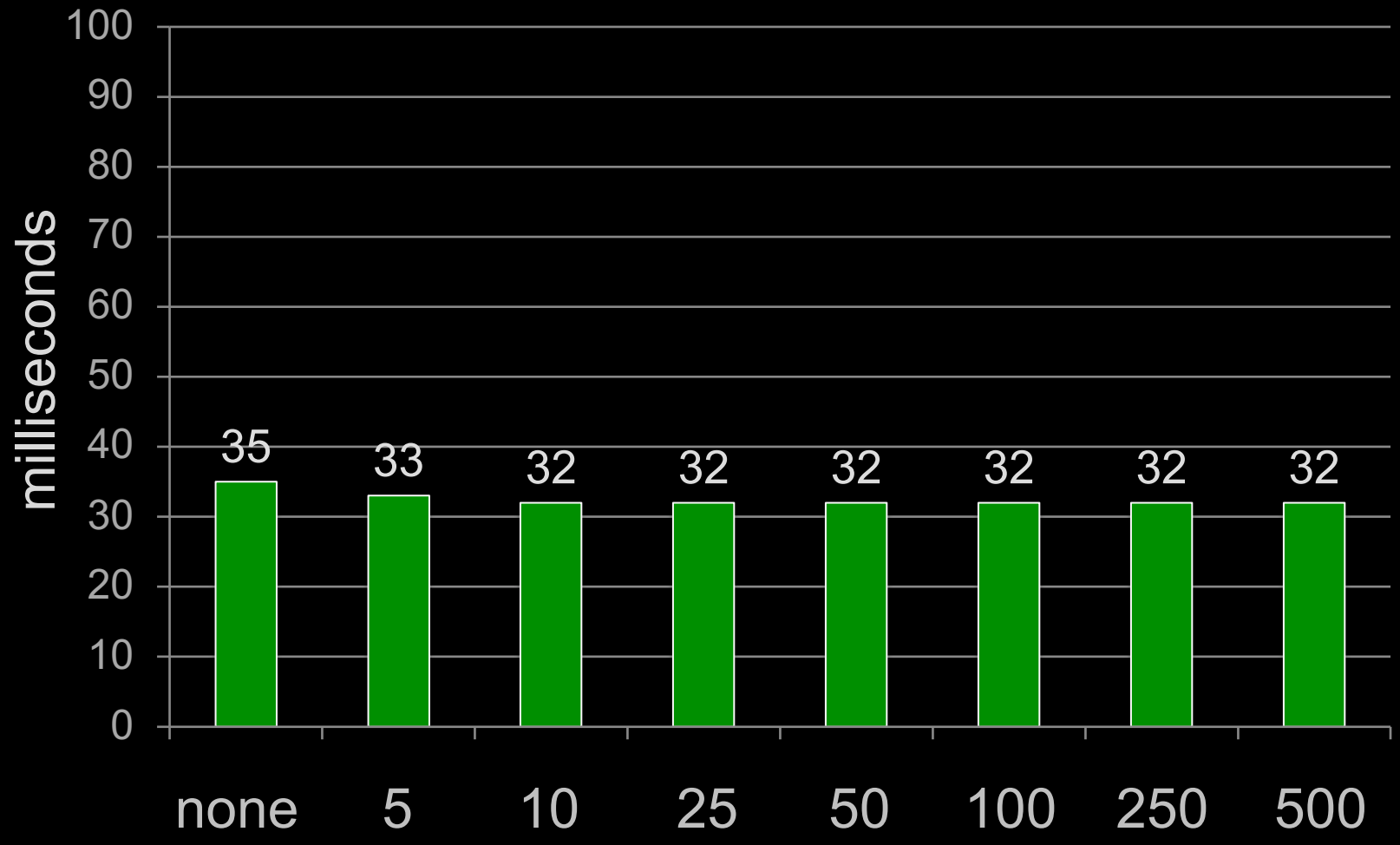
and associated latch contention?

# Tuning -lruskips

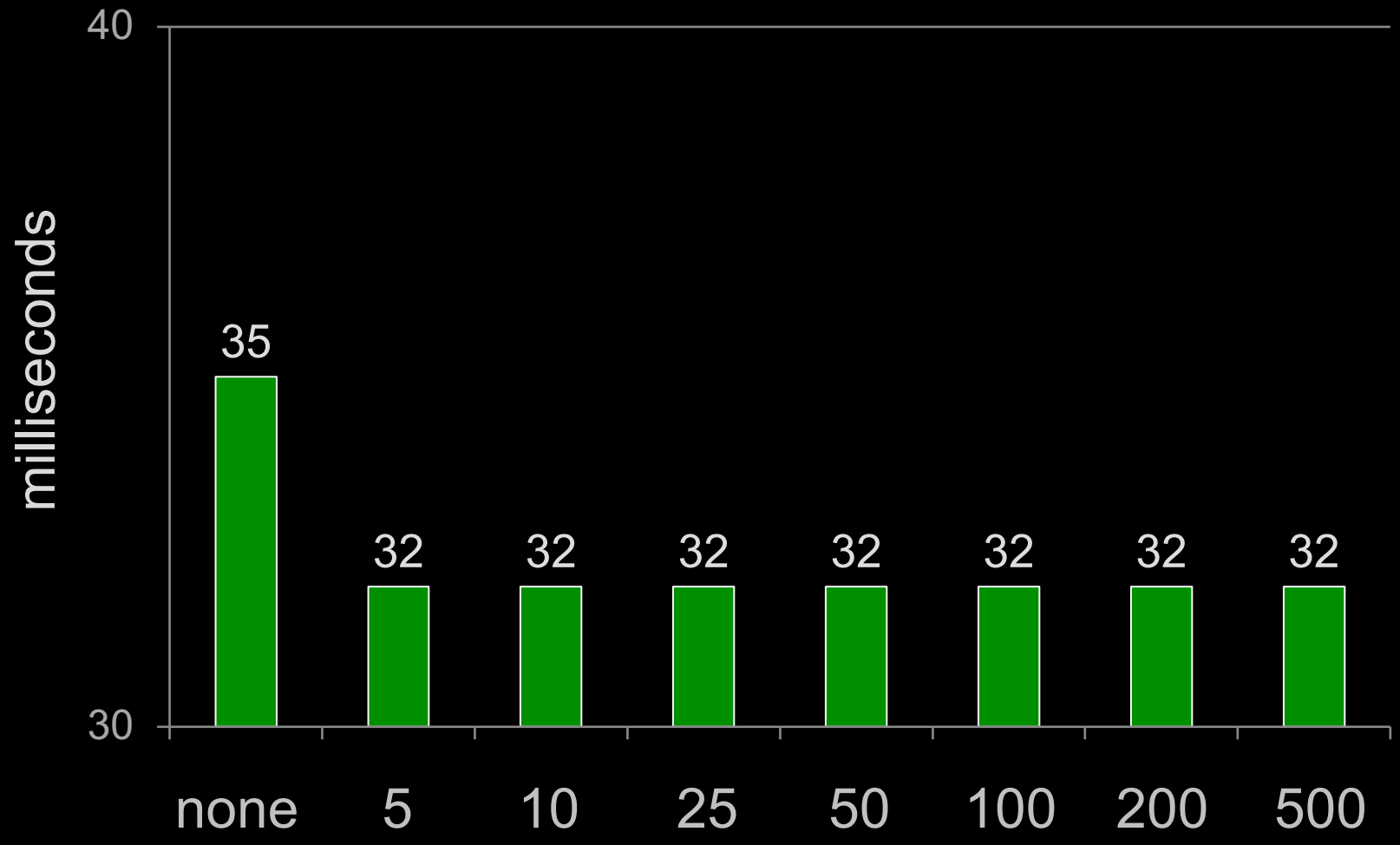
# napmax 250 (default), spin 5,000: vary lruskips



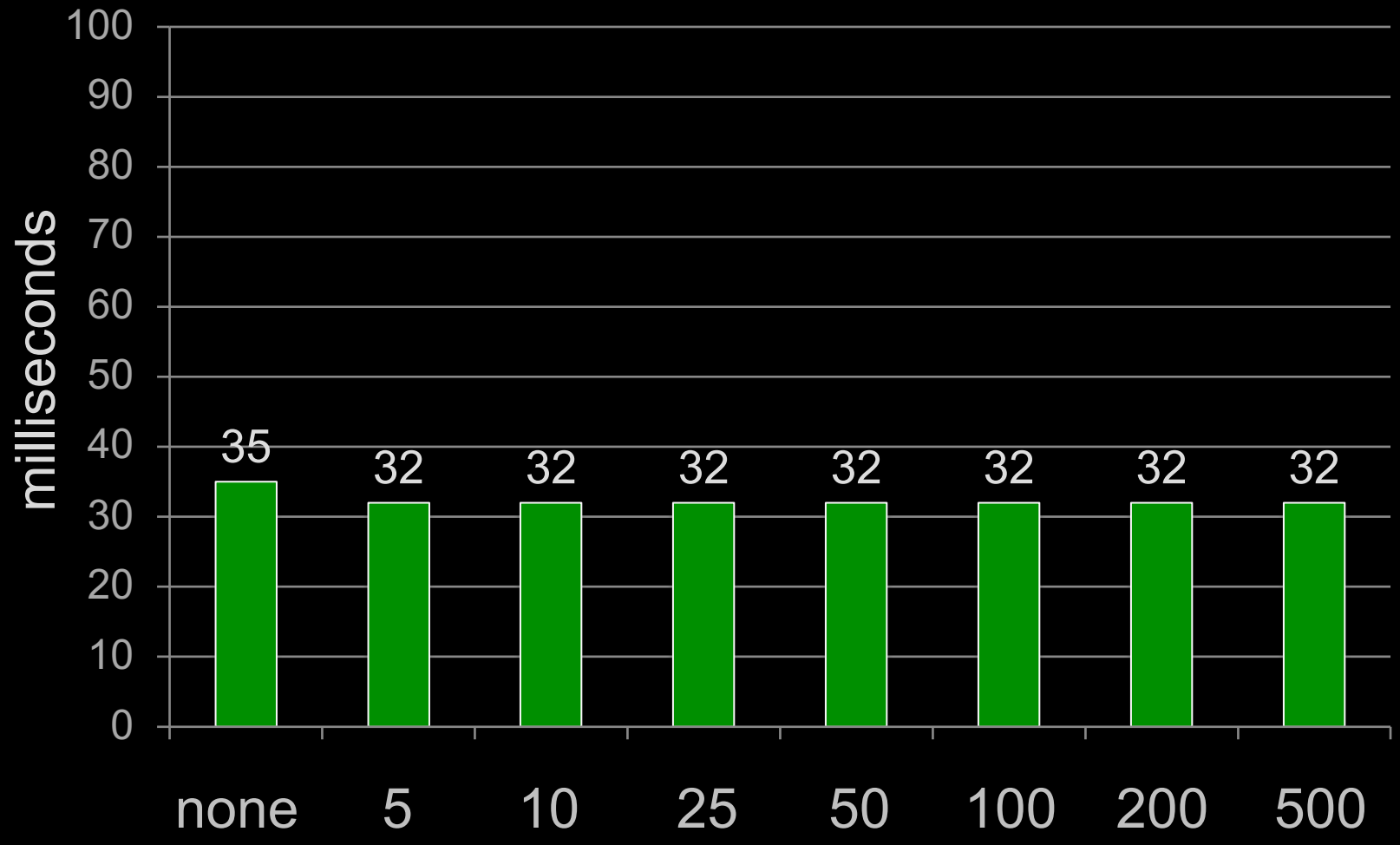
# napmax 250 (default), spin 50,000: vary lruskips



# napmax 10, spin 50,000: vary lruskips



# napmax 10, spin 50,000:  vary lruskips

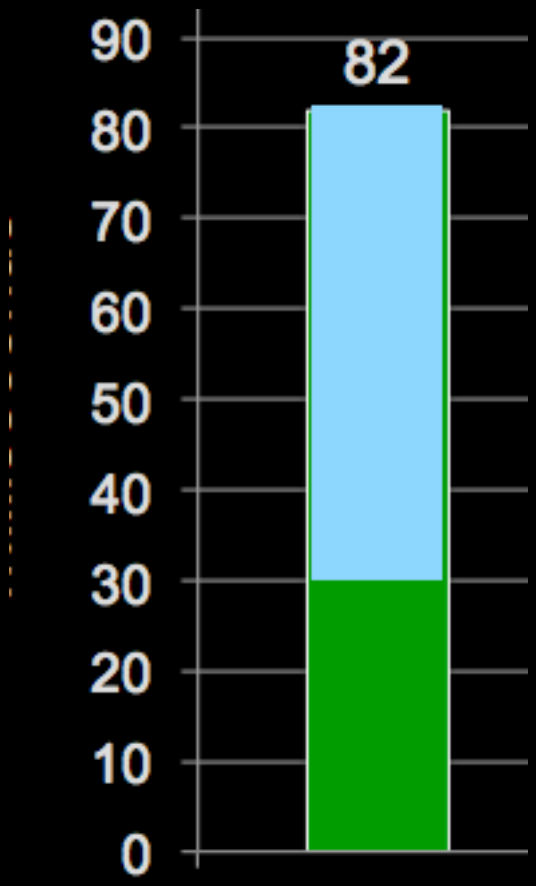


"Experience is a brutal teacher because
she gives the test first and the lesson afterwards."

-- Vernon Sanders Law

By tuning, we got rid of 51 milliseconds of time.

wasted time.



What do we learn from all this?

0) small changes have small effects

1) sometimes big changes have small effects

2) proper use of -spin has yuuge effects

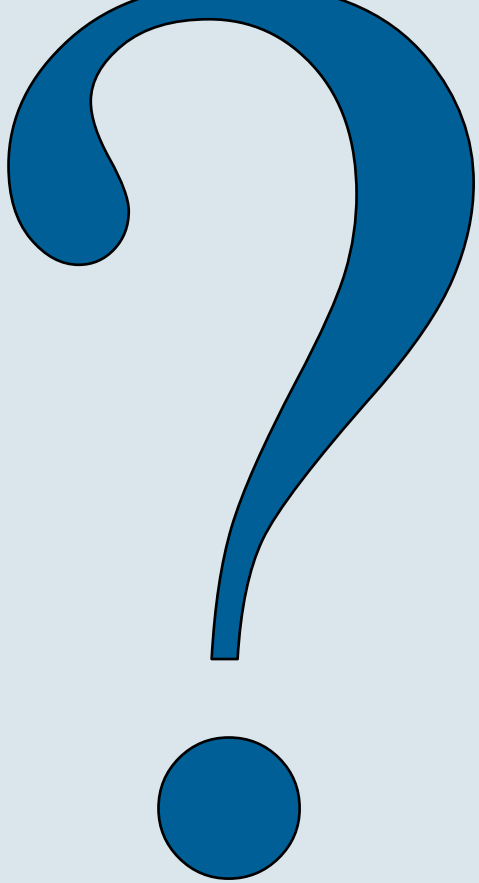
3) -spin should be higher than we thought

4) -napmax should probably be low
   (but watch out!)

5) spin, napmax, lruskips interact

6) lruskips 25 to 100 seems sufficient

# Want Answers



email:

gus642@gmail.com