PUGCHALLENGE EXCHANGE
AMERICAS

Time (and how to get rid of it)

Gus Björklund,

Head Groundskeeper, The Parmington Foundation,

Americas PUG Challenge
Manchester, NH, USA
4-7 June 2017

Abstract

In this talk, we examine the various ways in which time is used during the execution of a transaction by multiple concurrent users. One of these is "lock latency".

We then look at how latency can be reduced to quite small intervals by careful tuning.

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

Numbers you should know

(from Jeff Dean @ google)

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
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

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 Bunkers
  - database is about 12 GB
- 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

- 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

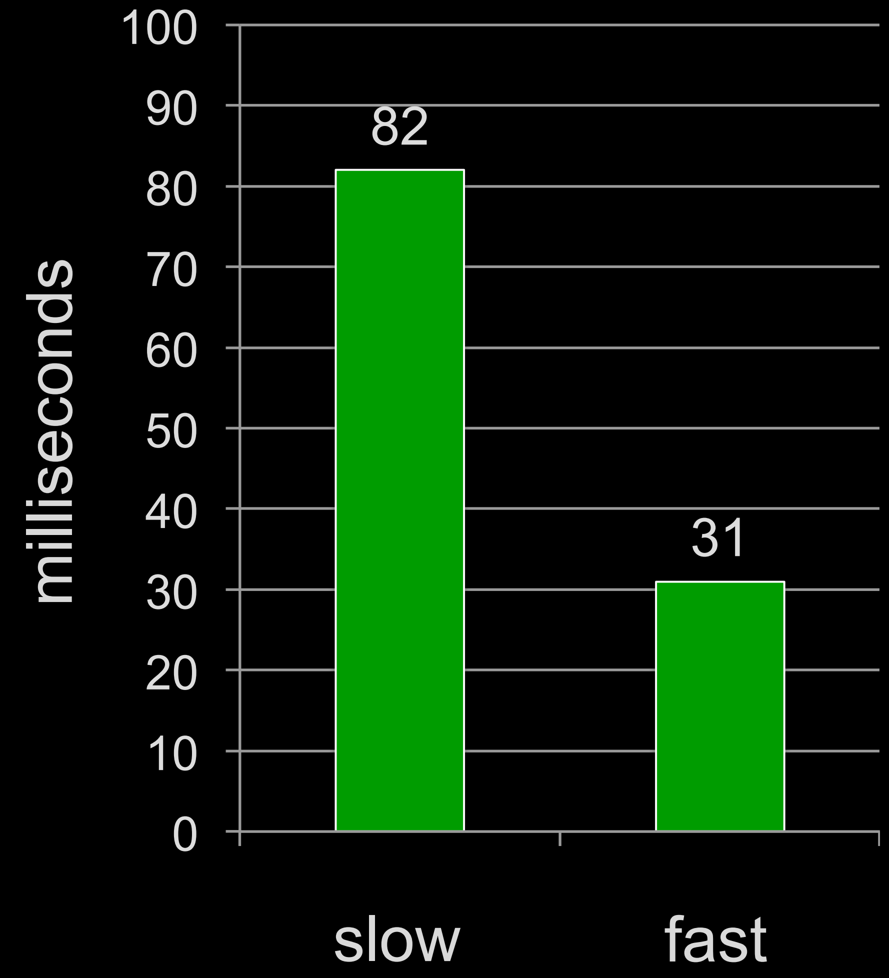
initial configuration

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
```

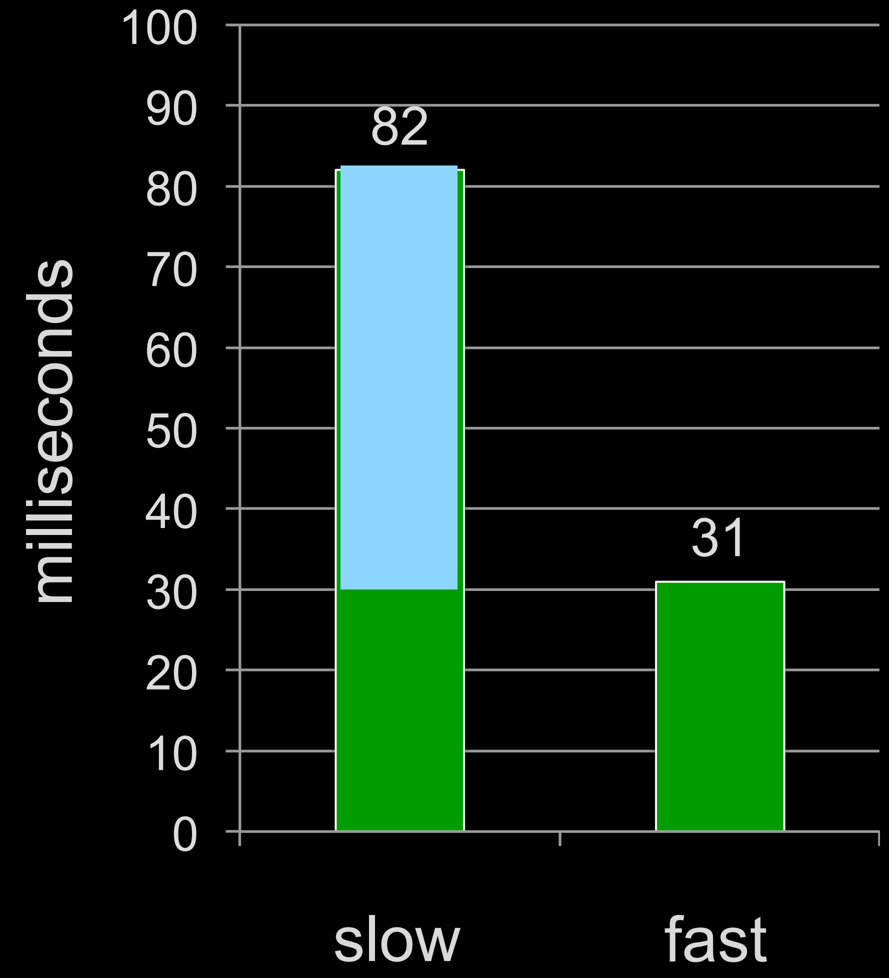
let's run some tests

transaction duration



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.

what can we do about it ??

The 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) get and release various kinds of locks

kinds of locks:

0) record locks

1) MTX lock

2) TXE lock

3) data buffer locks

4) bi buffer locks

5) latches

Latches are typically held for very short times.

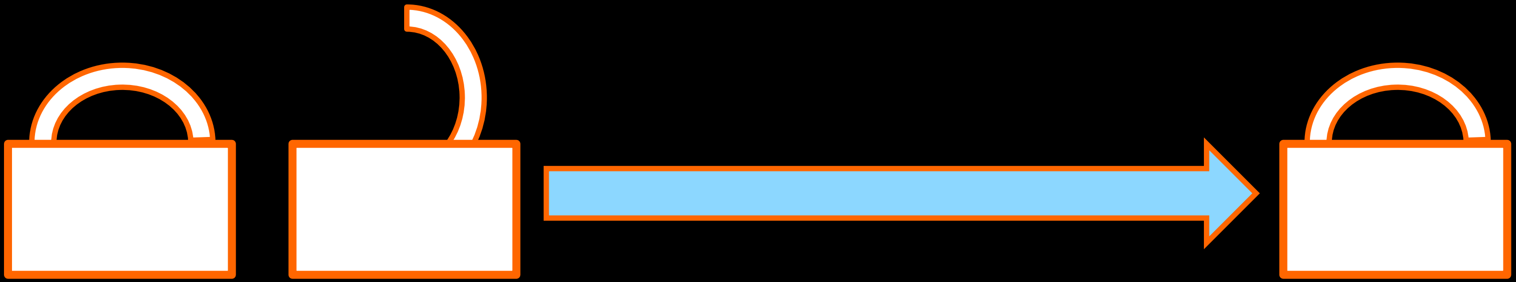
maybe 100 nanoseconds

on modern computers

Lock latency:

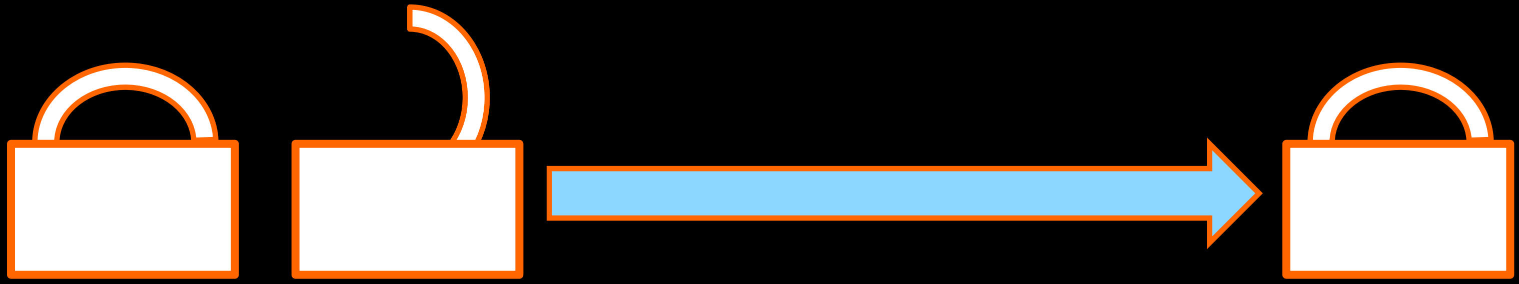
time from when holder releases lock
until waiting acquirer has locked it.

No useful work done while waiting.



Spinlock latches:

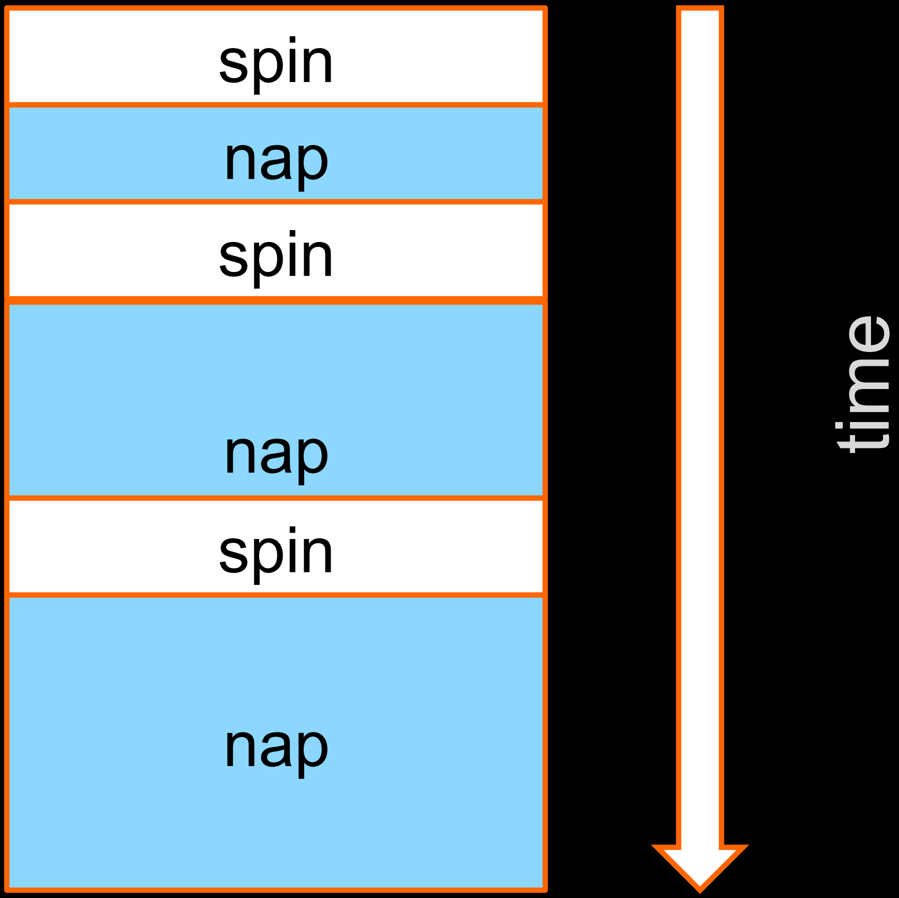
test and set
spin and test
take a nap
spin and test
nap longer
spin and test
nap even longer



-spin

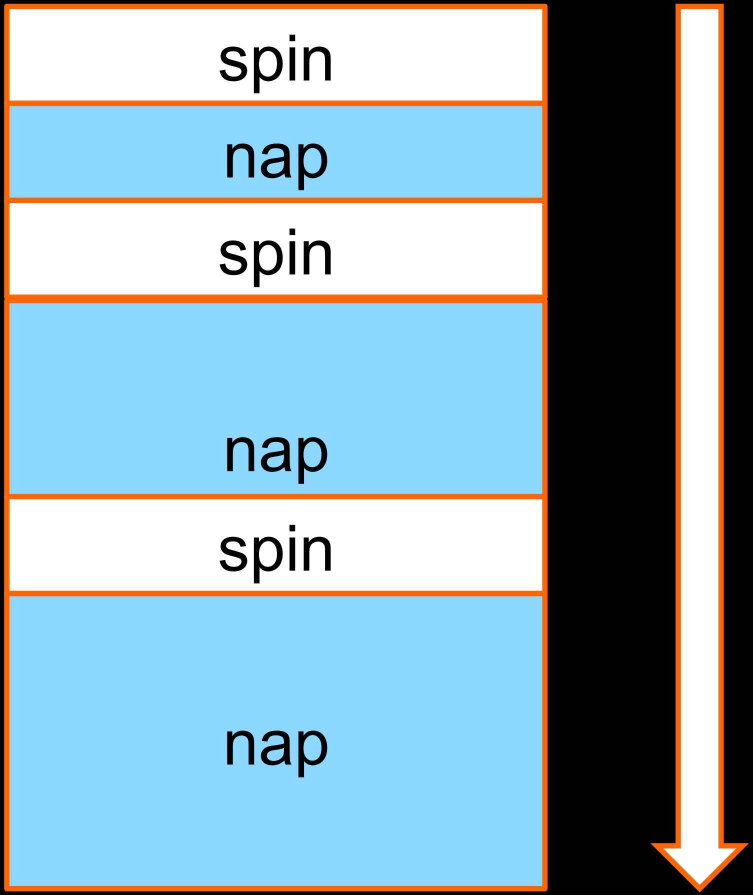
-nap

-napmax



Tuning

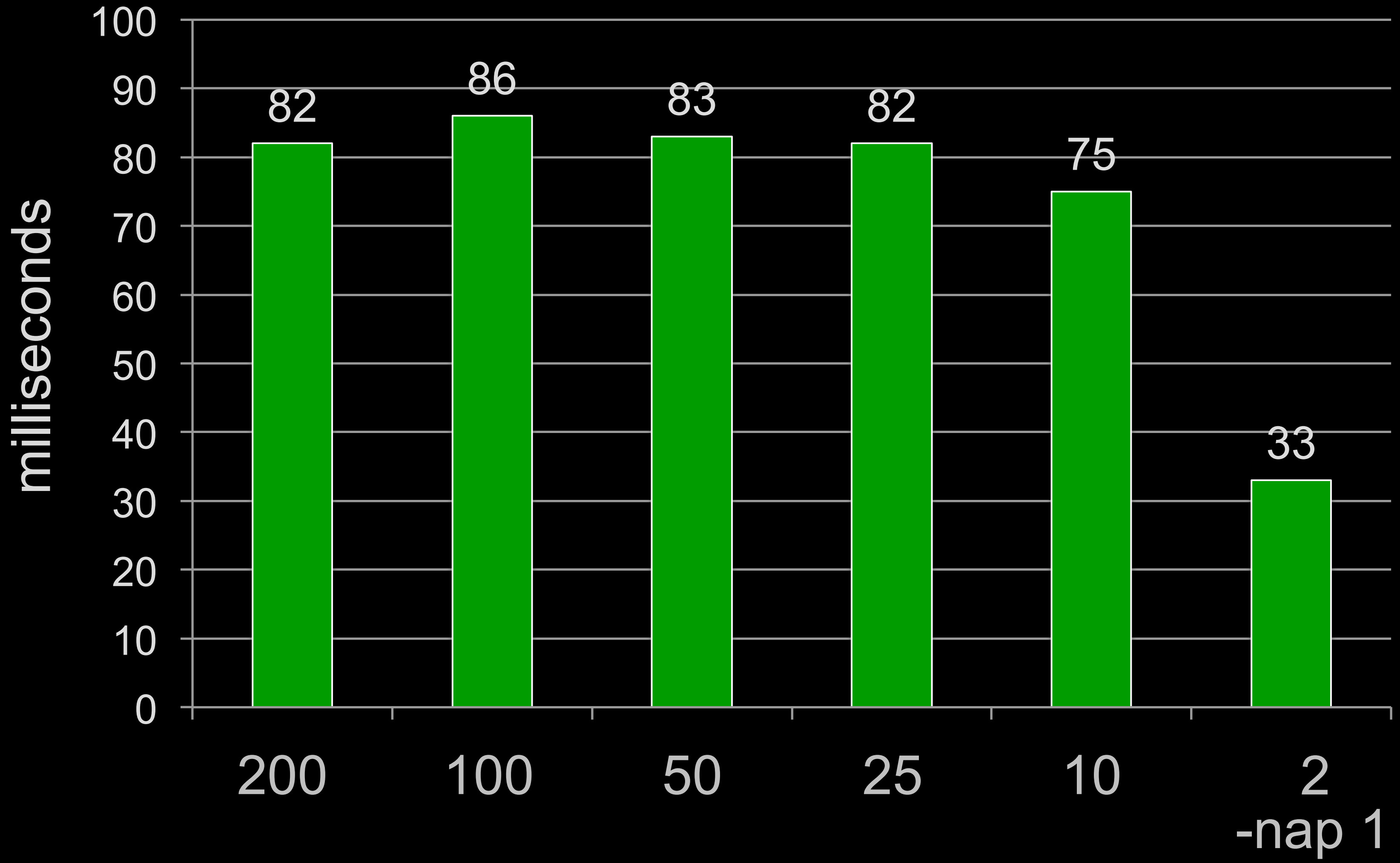
-napmax



The dawn rises only when the rooster crows.

Burmese proverb

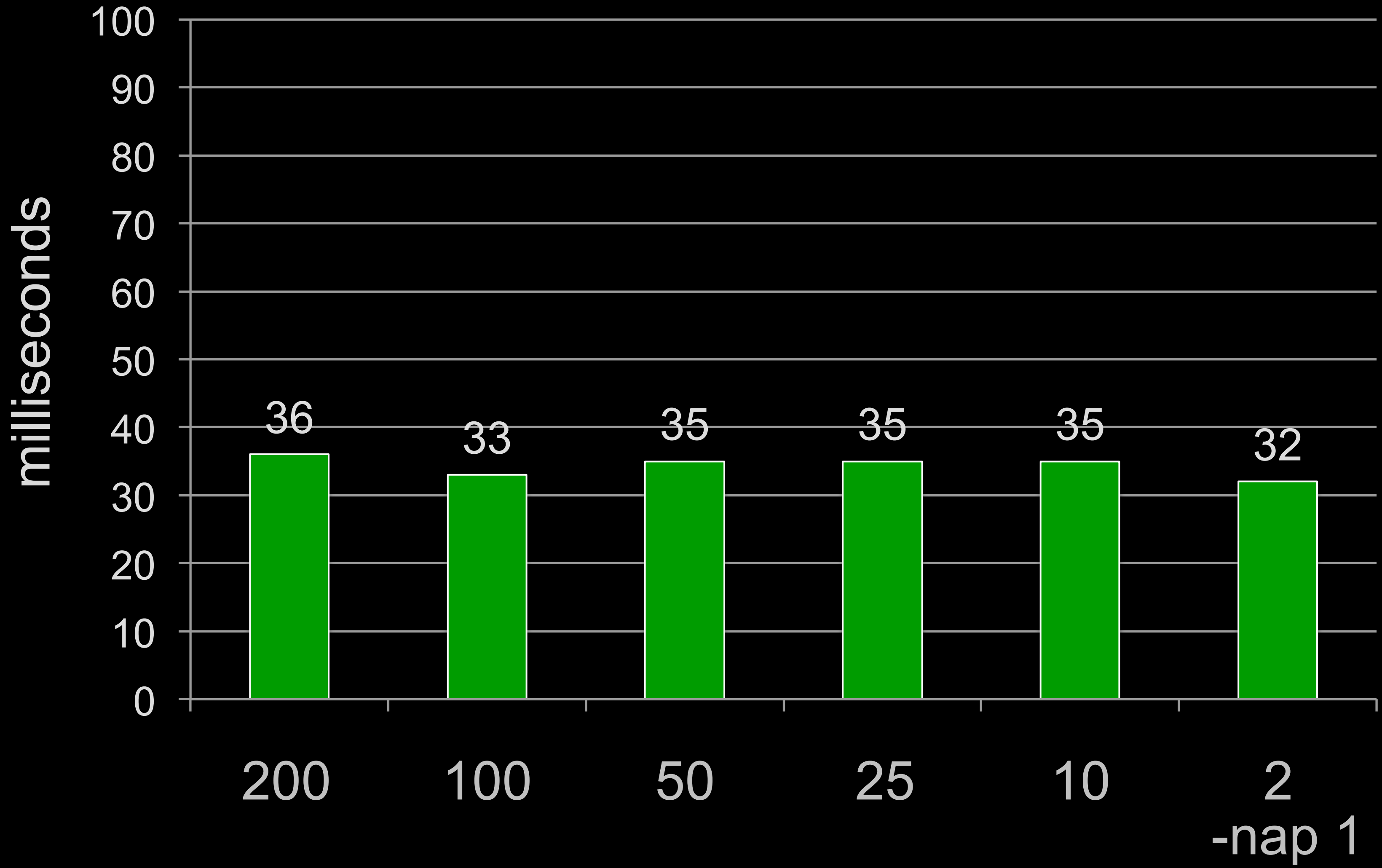
-spin 5,000 vary -napmax



Change -spin to 50,000

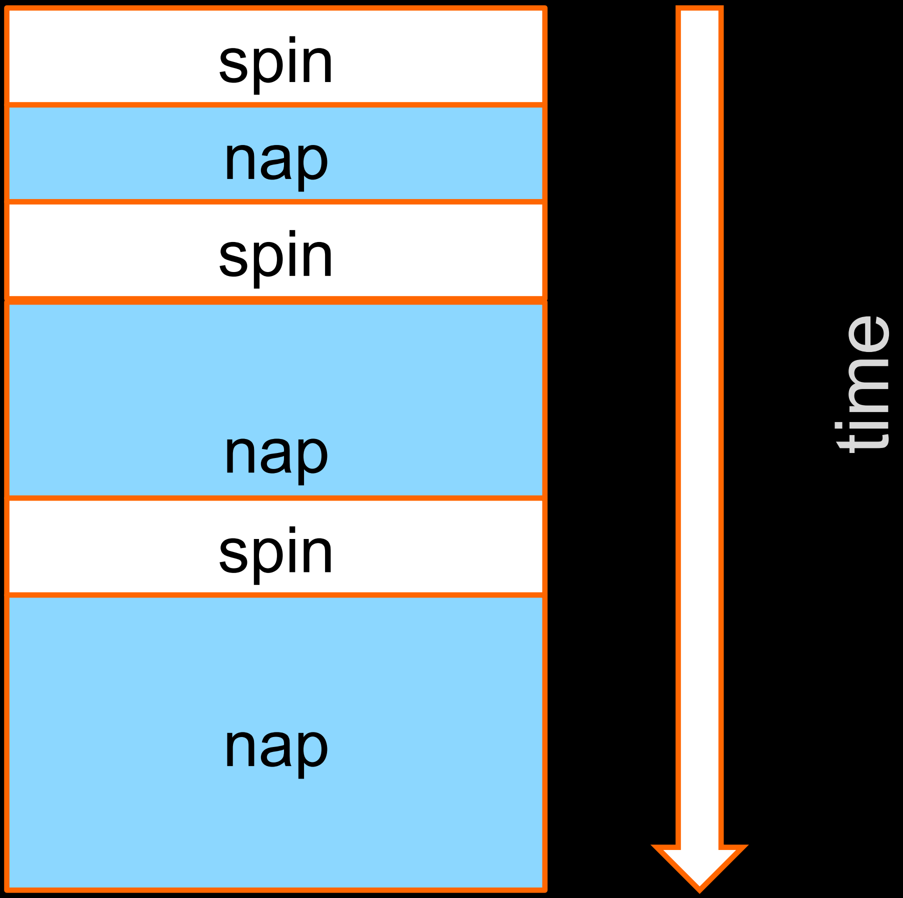
Tune –napmax again

-spin 50,000: vary -napmax

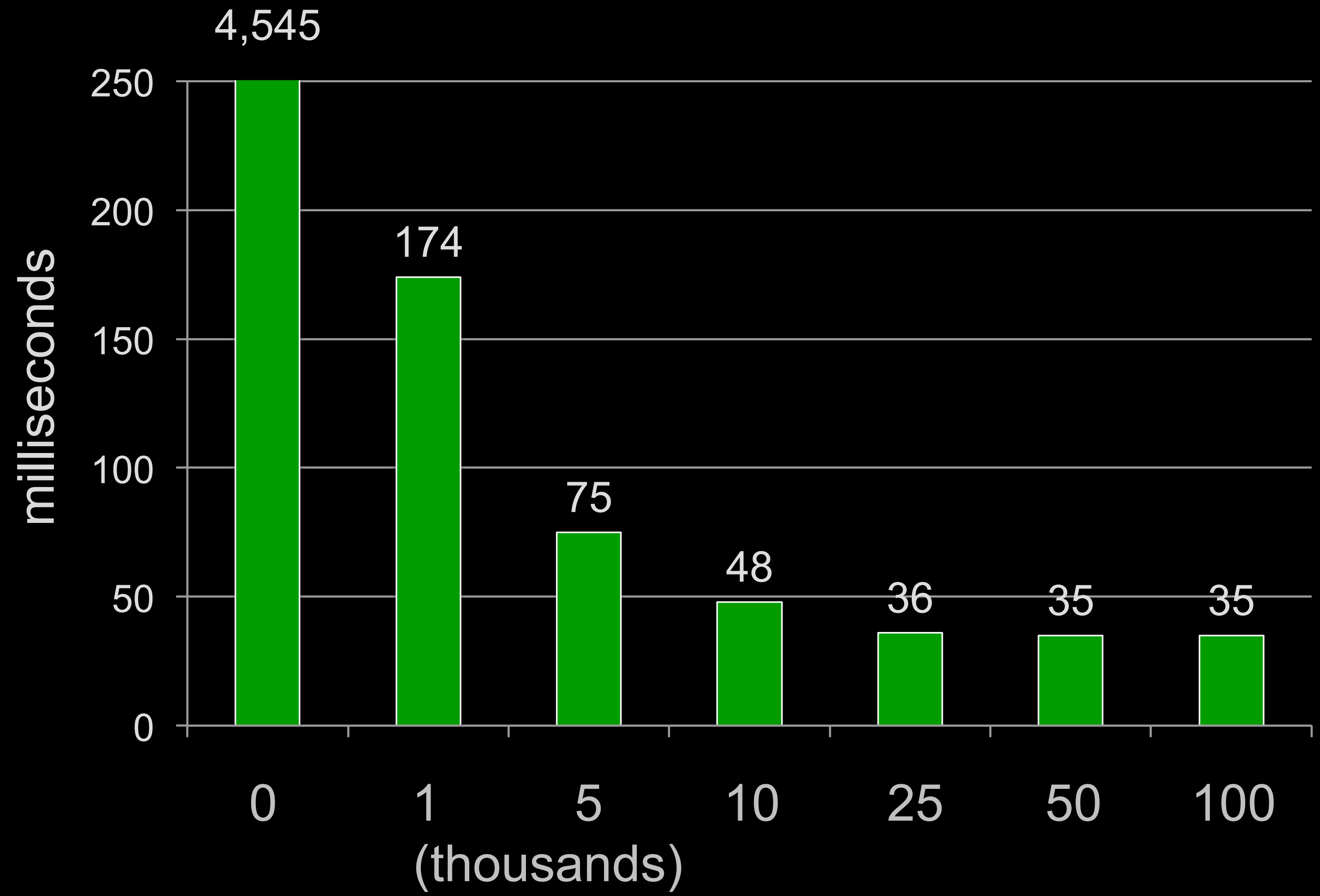


Tuning

-spin



-napmax 10: vary -spin

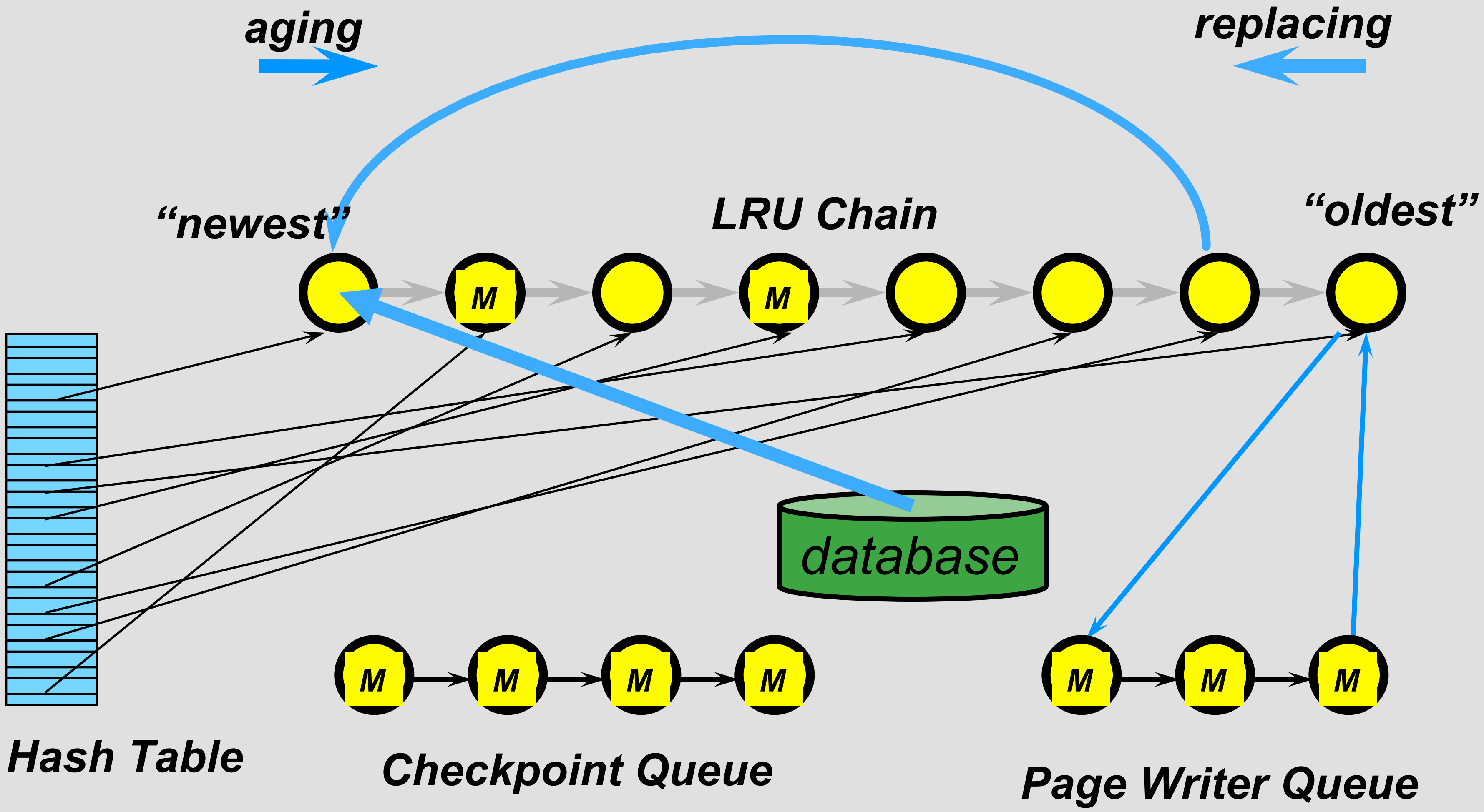


Longer nap times => higher latch latency

Higher spin => lower latch latency

Higher contention => higher latch latency

Buffer Pool LRU Chain



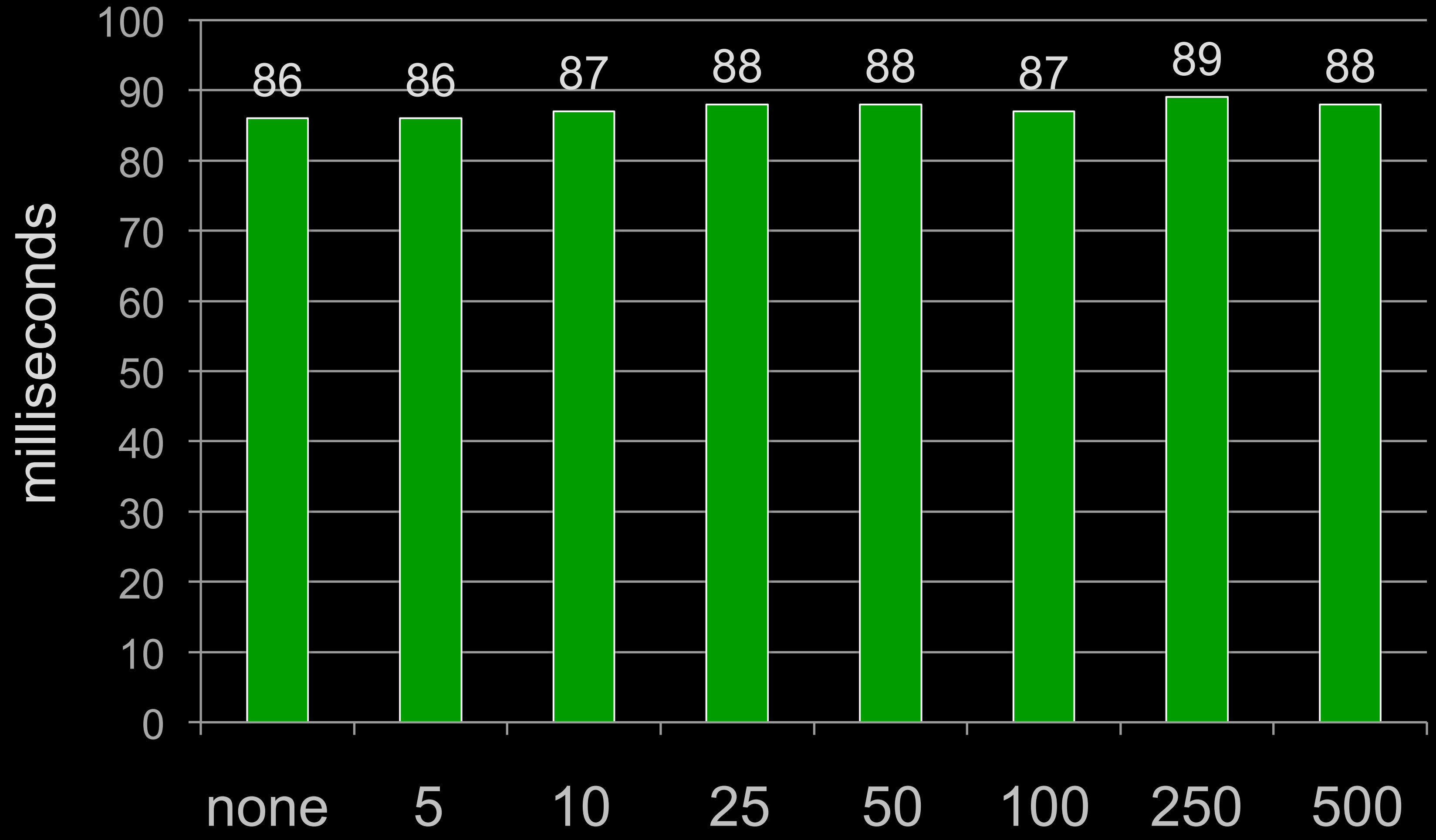
Every buffer access causes an LRU chain update

Can we reduce LRU chain overhead

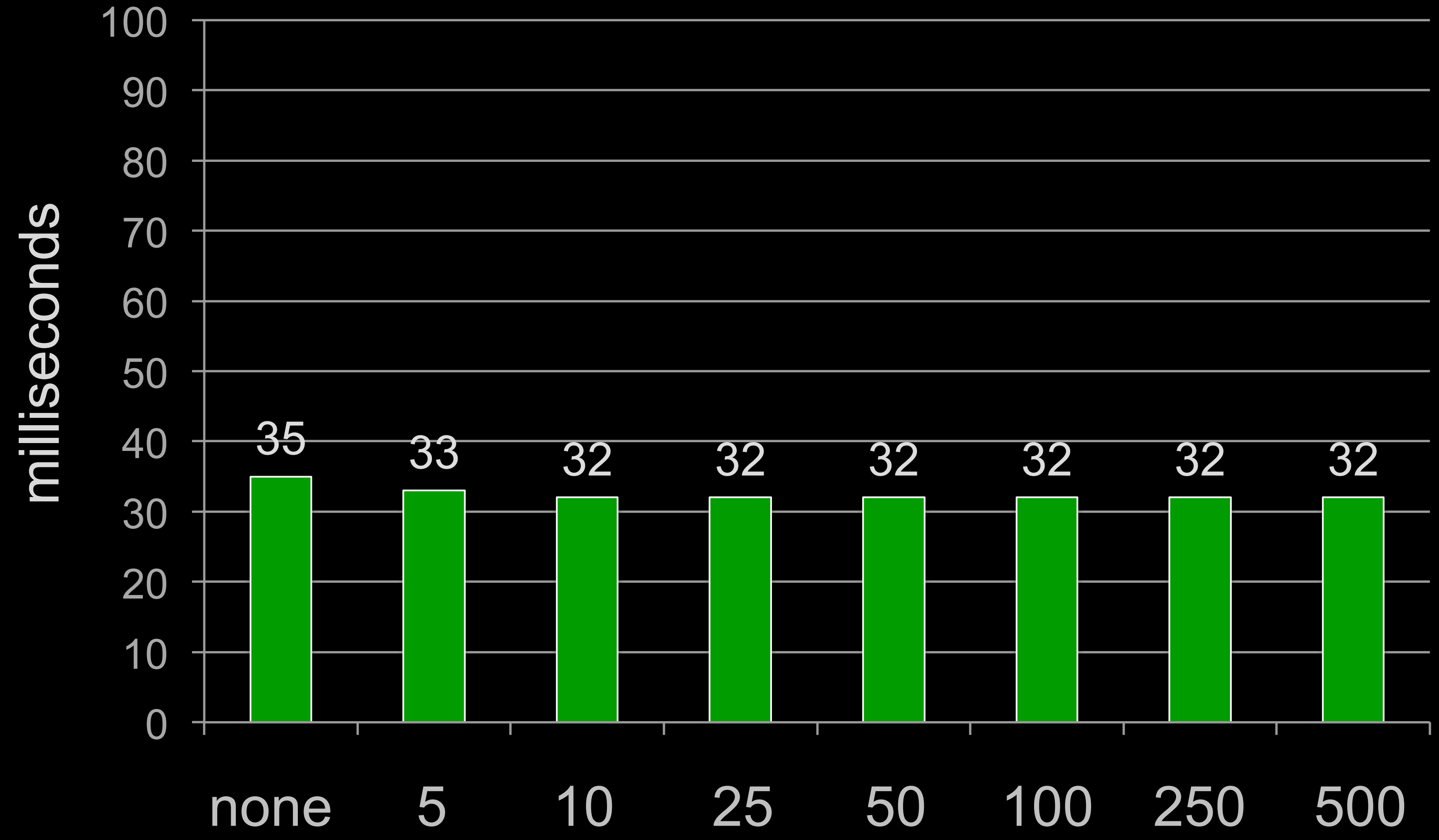
and associated latch contention?

Tuning -lruskips

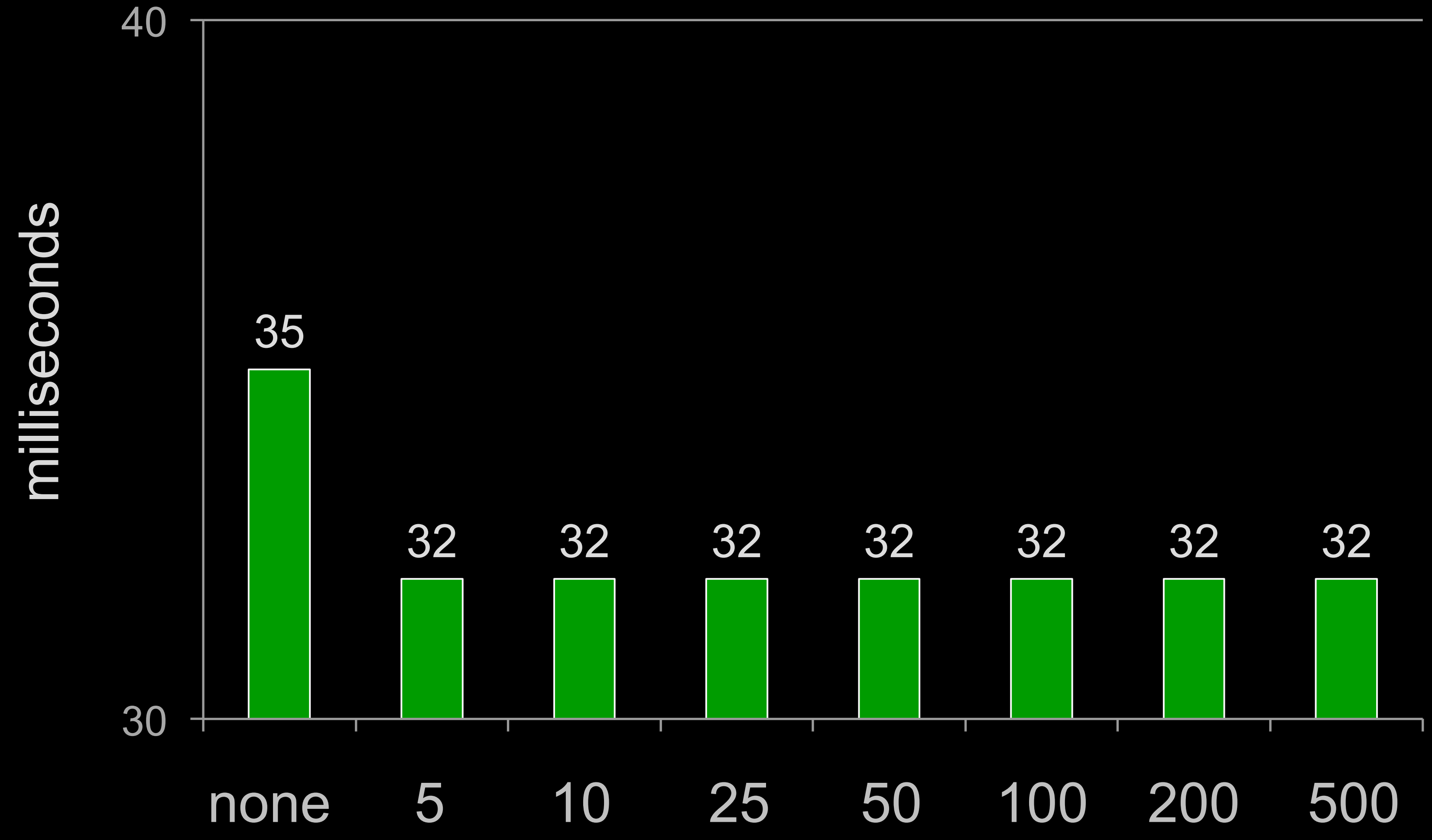
napmax 250 (default), spin 5,000: vary lruskips



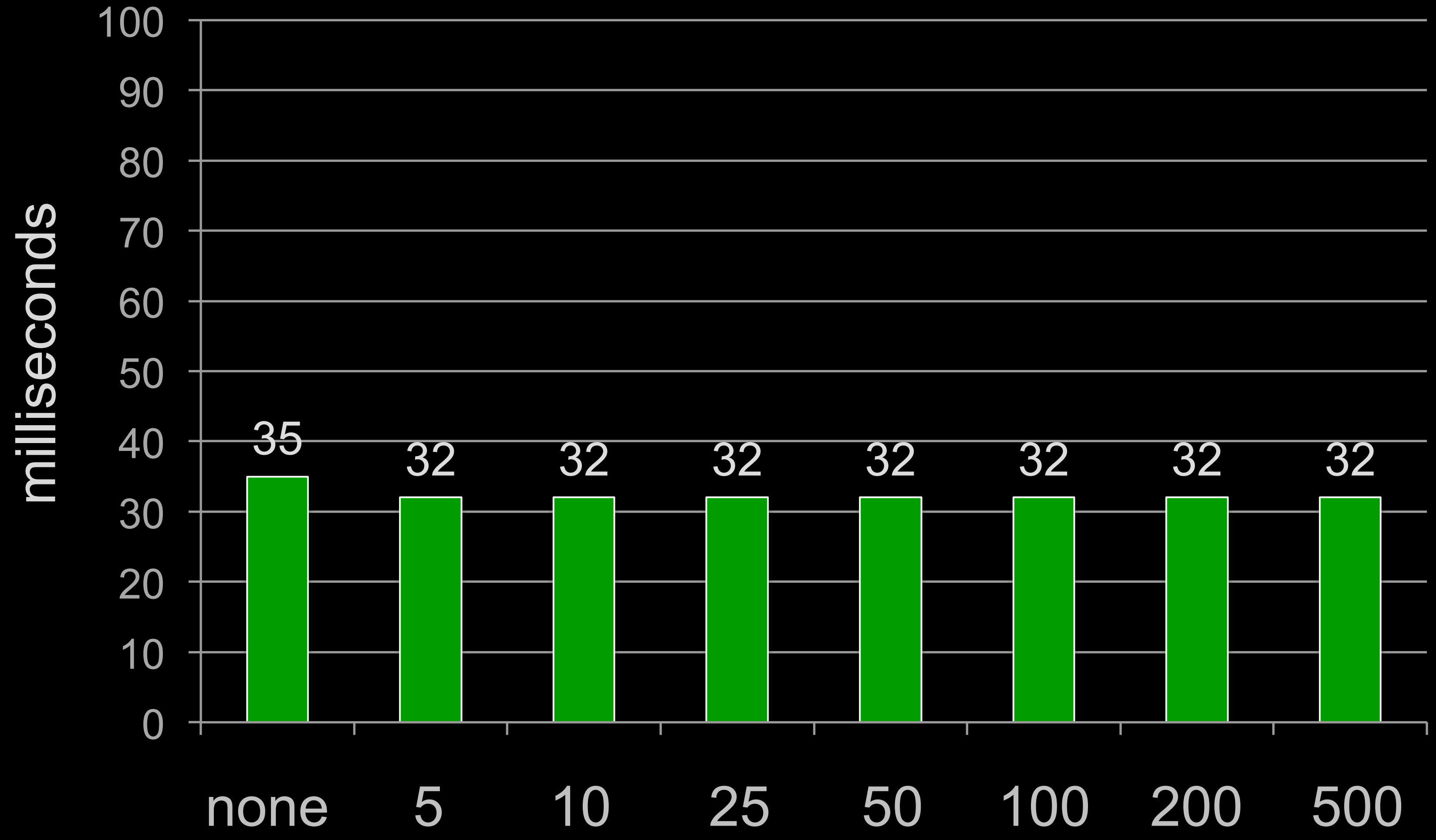
napmax 250 (default), spin 50,000: vary lruskips



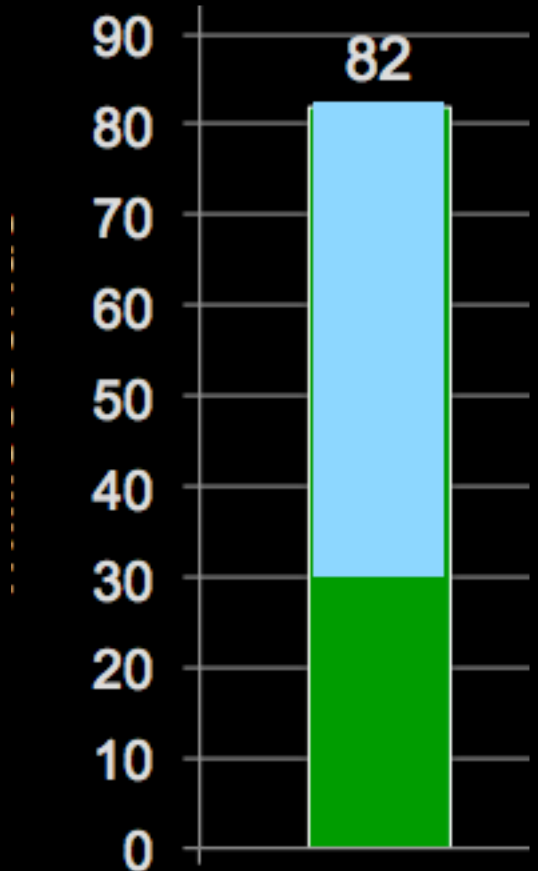
napmax 10, spin 50,000: vary lruskips



napmax 10, spin 50,000: vary lruskips



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



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

-- Vernon Sanders Law

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 be low
5) spin, napmax, lruskips interact
6) lruskips 25 to 100 seems sufficient

Want Answers ?

email:

gus642@gmail.com