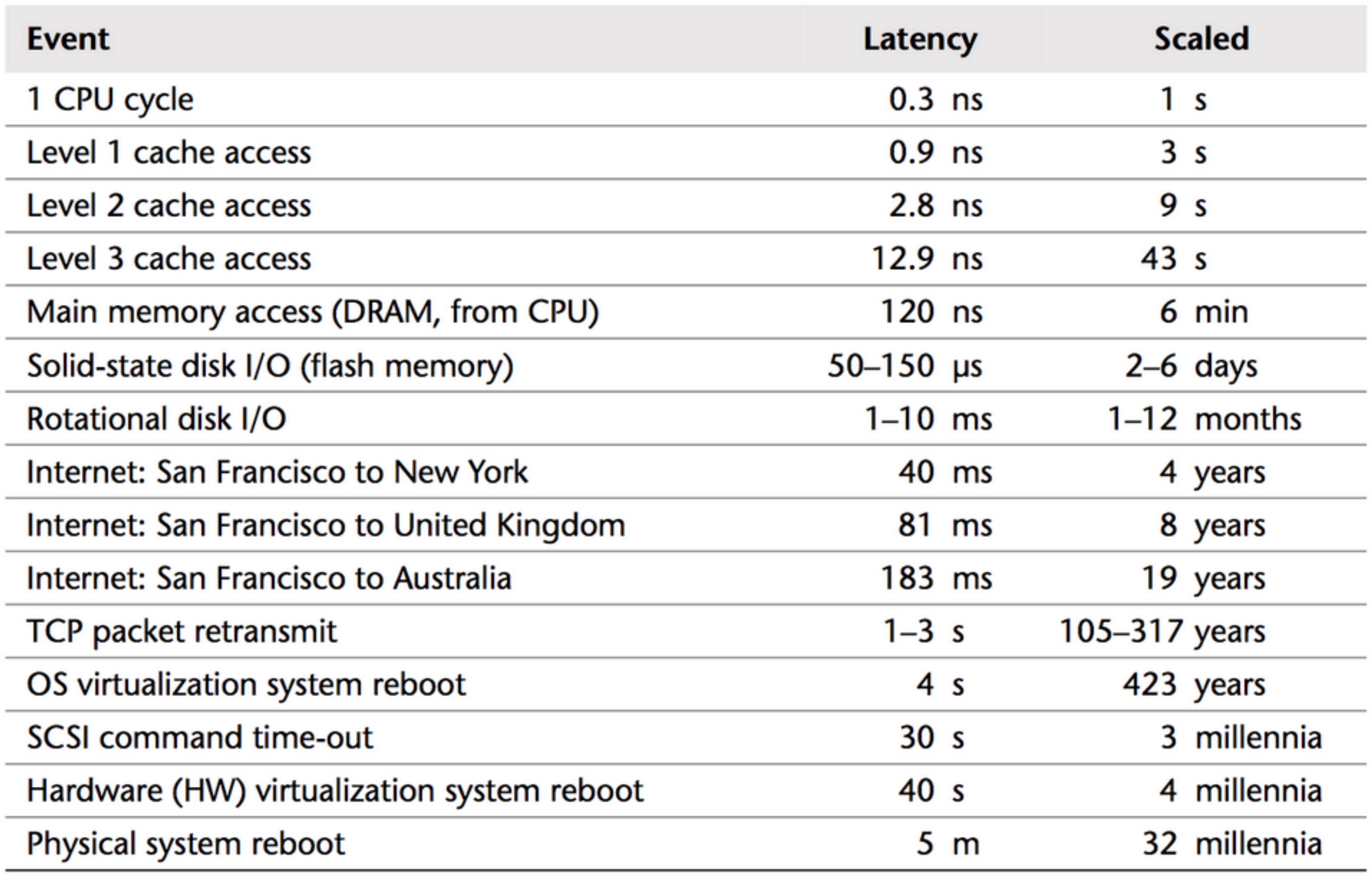
dablooms & dablooms_http

~B

understanding cost of quering

Table 2.2 Example Time Scale of System Latencies

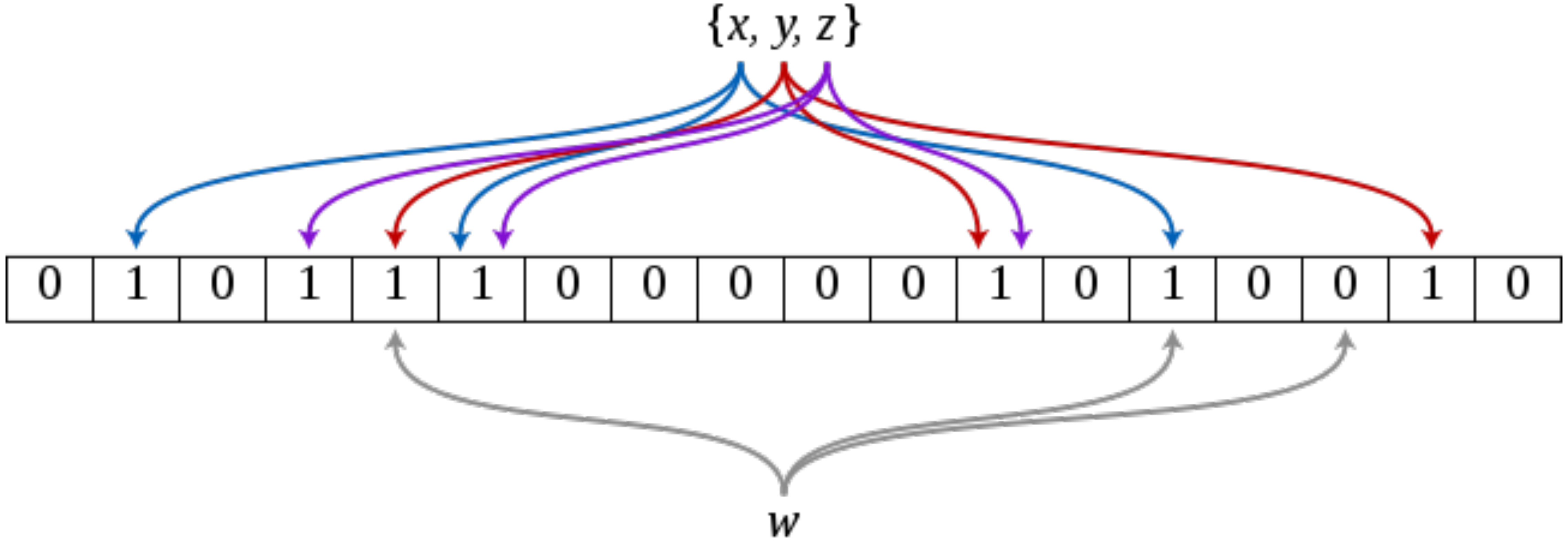


Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

why

- for every incoming profile id we need to check whether it exists in our DB.
- DB queries range from 20ms to 500ms or more when the DB is on another machine
- the event of creating a profile is once in the lifetime of that profile and does not change
- it’s almost like flipping a switch that never turns off

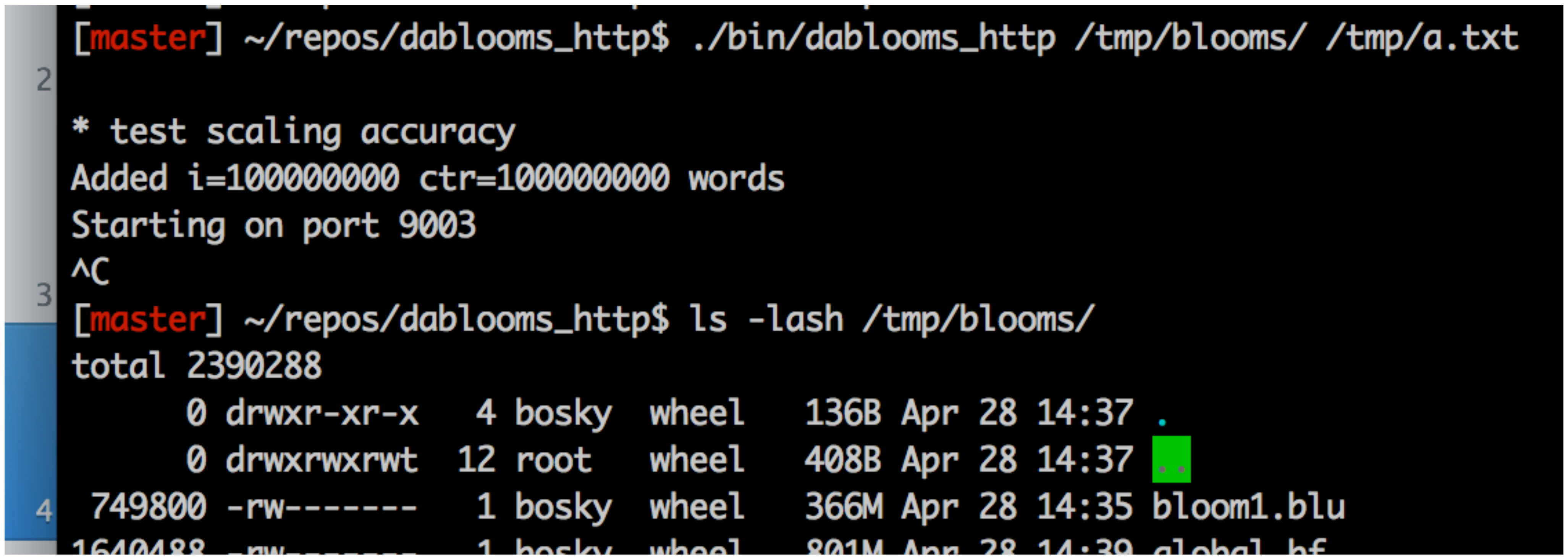
bloom filter



why

- storing 100 Million profiles of keysize 2 bytes =>
  => 190 MB

- storing the same in our bloom filter with capacity for 100M entries and error rate 0.05
  => **~1.6 MB**



```
[master] ~/repos/dablooms_http$ ./bin/dablooms_http /tmp/blooms/ /tmp/a.txt

* test scaling accuracy
Added i=100000000 ctr=100000000 words
Starting on port 9003
^C
[master] ~/repos/dablooms_http$ ls -lash /tmp/blooms/
total 2390288
      0 drwxr-xr-x   4 bosky  wheel   136B Apr 28 14:37 .
      0 drwxrwxrwt  12 root   wheel   408B Apr 28 14:37 ..
 749800 -rw-------   1 bosky  wheel   366M Apr 28 14:35 bloom1.blu
1640488 -rw-------   1 bosky  wheel   801M Apr 28 14:39 global.bf
```

scalable bloom filters

- Almeida, Baquero, Preguiça, Hutchison published a paper in 2006, on Scalable Bloom Filters
- suggested creating a list of bloom filters that act as one large bloom filter.
- When greater capacity is desired, a new filter is added to the list OR when a certain degree of additions also happen.

dablooms

- created by bit.ly for finding out if a url exists in their system already
- C, with wrappers in python, lisp, php, go
- instead of a bit, implements a 4-bit counter so that over time or queries, garbage collection occurs
- comes with implementation of murmur hash

dablooms_http

- C http wrapper over dablooms, created at Helpshift
- supports namespaces (each ns has its own *bf*)
- supports persisting multiple namespaces and reloading on next startup
- POST for adding entries
  GET for querying

how-to

```
$ make deps compile
```

```
Usage:
./bin/dablooms_http <bloom_dir>
<global_bloom_words_file>
```

```
$ mkdir /tmp/blooms

$ ./bin/dabloom_http
        /tmp/blooms/ /usr/share/dict/words
```

simple membership query

```
$ curl http://localhost:9003/?key=orange
0                            (NOTE: is plain/text )


$ curl -X POST -d “key=orange” http://localhost:9003/
{“ok":0}                     (NOTE: this is json )


$ curl http://localhost:9003/?key=orange
1
```

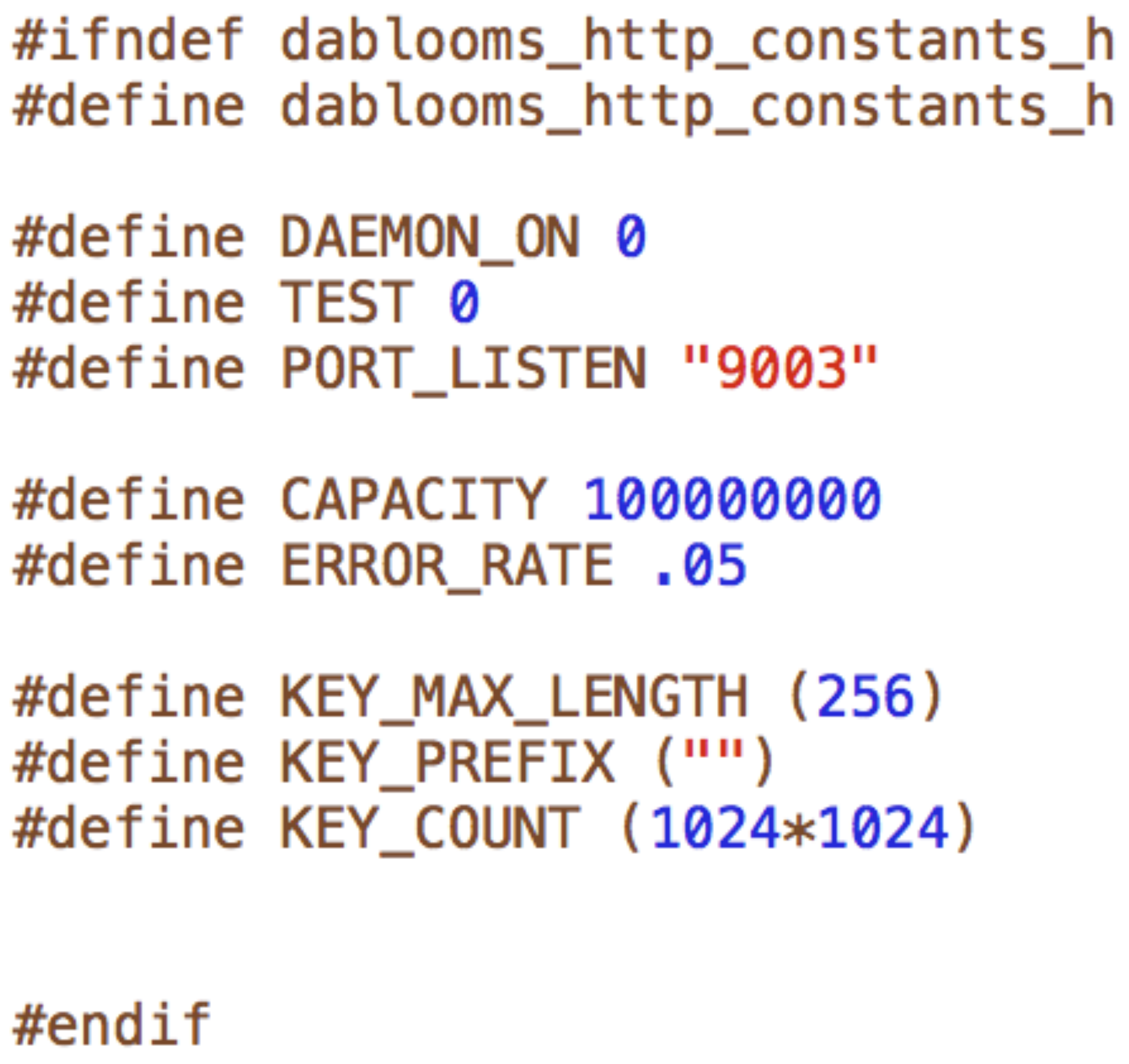
namespacing

```
$ curl -X POST -d “key=pune&ns=cities"
http://localhost:9003/
{“ok”:1}

$ curl
http://localhost:9003/?key=pune
0

$ curl
http://localhost:9003/?key=pune&ns=cities
1
```

examples

```
# is this profile known?
$ curl
http://localhost:9003/?key=profile1&ns=boomapp

# have we got visits from this ip
$ curl http://localhost:9003/?key=ip&ns=ip

# is this question encountered before?
$ curl
http://localhost:9003/?key=questionhash1&app=boom
```

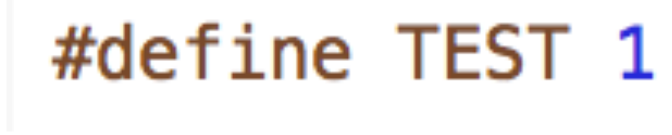
constants.h



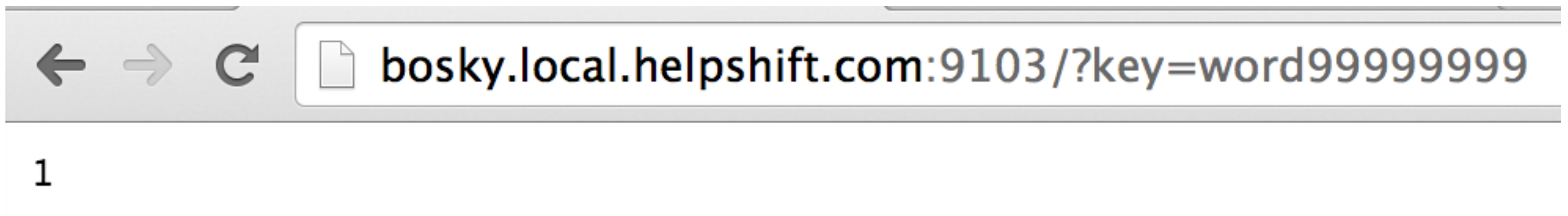
```
#ifndef dablooms_http_constants_h
#define dablooms_http_constants_h

#define DAEMON_ON 0
#define TEST 0
#define PORT_LISTEN "9003"

#define CAPACITY 100000000
#define ERROR_RATE .05

#define KEY_MAX_LENGTH (256)
#define KEY_PREFIX ("")
#define KEY_COUNT (1024*1024)


#endif
```

performance

after adding 10M entries



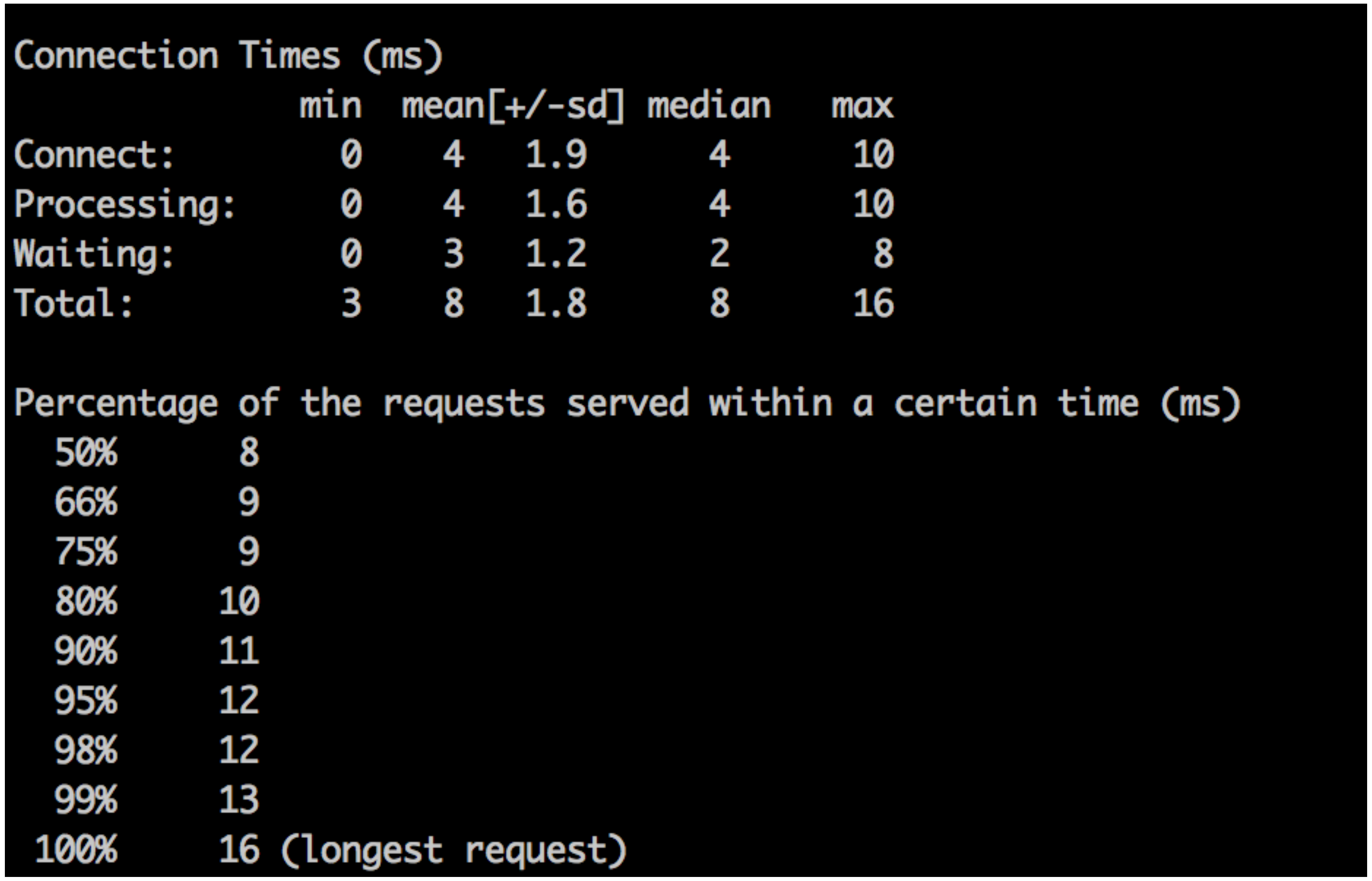
```
#define TEST 1
```



bosky.local.helpshift.com:9103/?key=word99999999

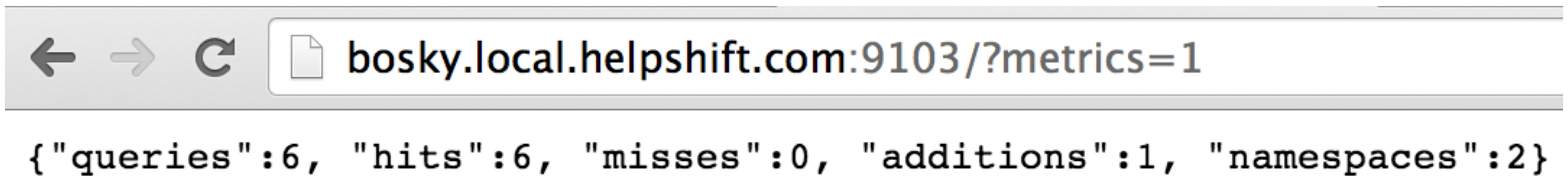
```
1
```

benchmarking 30k requests
with concurrency of 100



```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4   1.9      4      10
Processing:     0    4   1.6      4      10
Waiting:        0    3   1.2      2       8
Total:          3    8   1.8      8      16

Percentage of the requests served within a certain time (ms)
  50%      8
  66%      9
  75%      9
  80%     10
  90%     11
  95%     12
  98%     12
  99%     13
 100%     16 (longest request)
```

metrics



bosky.local.helpshift.com:9103/?metrics=1

```
{"queries":6, "hits":6, "misses":0, "additions":1, "namespaces":2}
```

references

- dablooms
  https://github.com/bitly/dablooms

- mongoose
  https://github.com/cesanta/mongoose

- hashmap
  https://github.com/petewarden/c_hashmap

- dablooms_http
  https://github.com/helpshift/dablooms_http

- Scalable Bloom Filters
  http://www.sciencedirect.com/science/article/pii/S0020019006003127