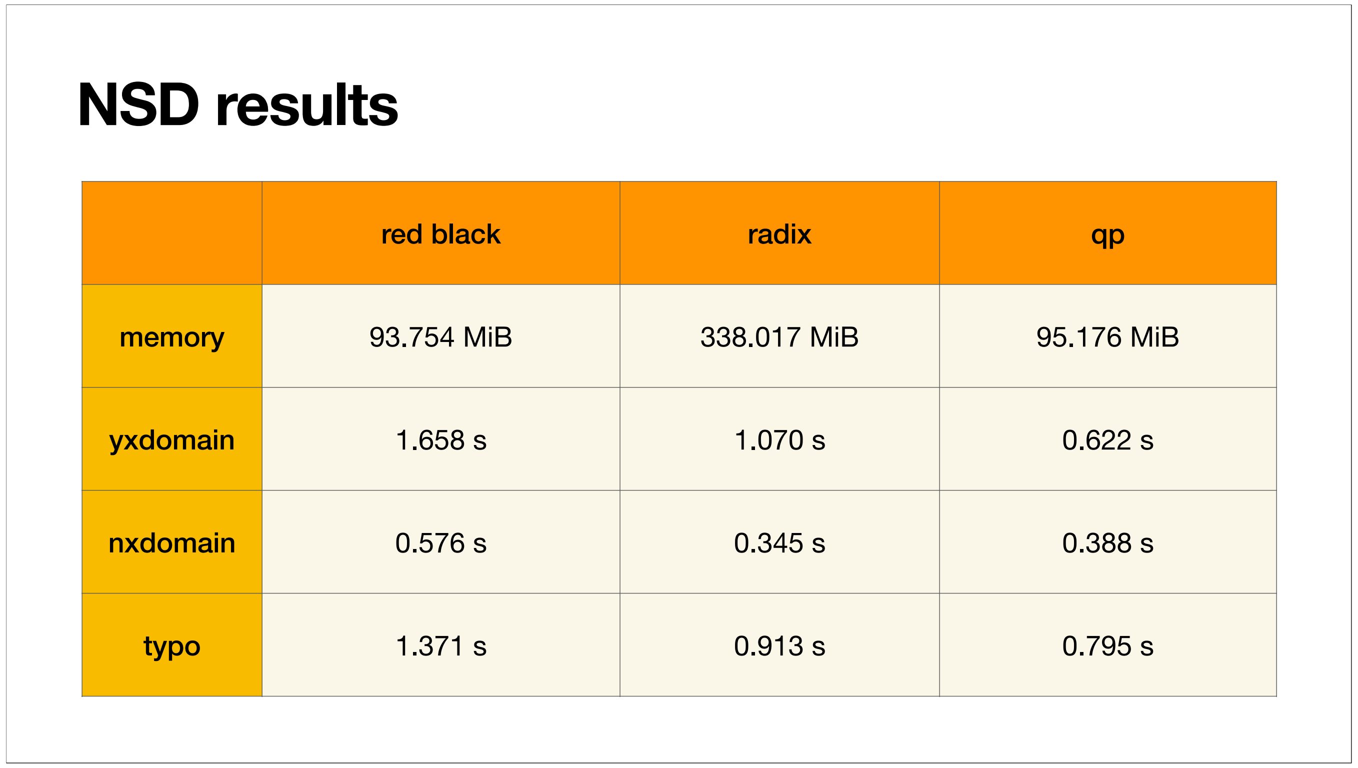
a qp-trie in BIND

why is it cool? what makes it hard?

Tony Finch <fanf@isc.org>



NSD results

	red black	radix	qp
memory	93.754 MiB	338.017 MiB	95.176 MiB
yxdomain	1.658 s	1.070 s	0.622 s
nxdomain	0.576 s	0.345 s	0.388 s
typo	1.371 s	0.913 s	0.795 s

qp-trie code from summer 2021

qp memory usage competitive with red-black tree

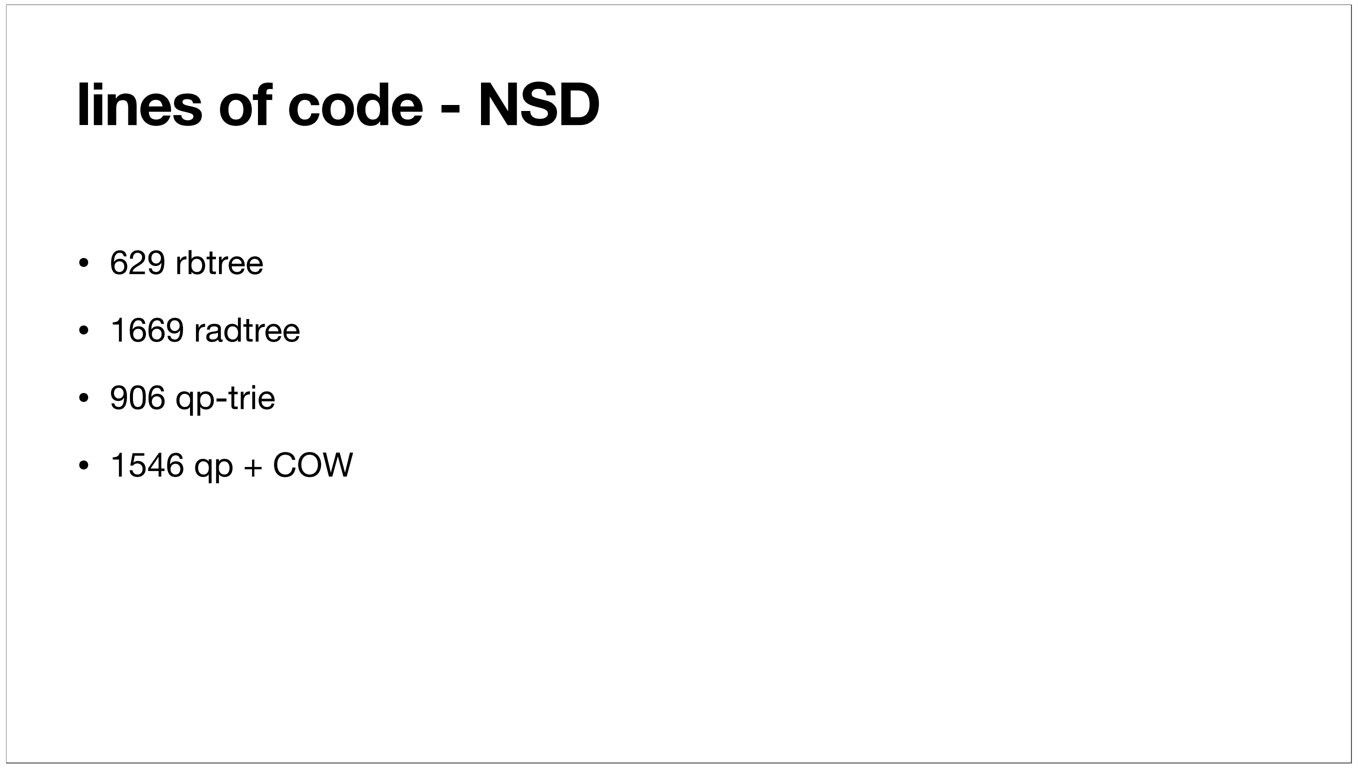
qp usually fastest

yxdomain = known to exist in tree

nxdomain = completely random

typo = small random change to known domain

qp nxdomain is a bit slower because it needs two traversals to find the nonexistence proof whereas the others can stop earlier



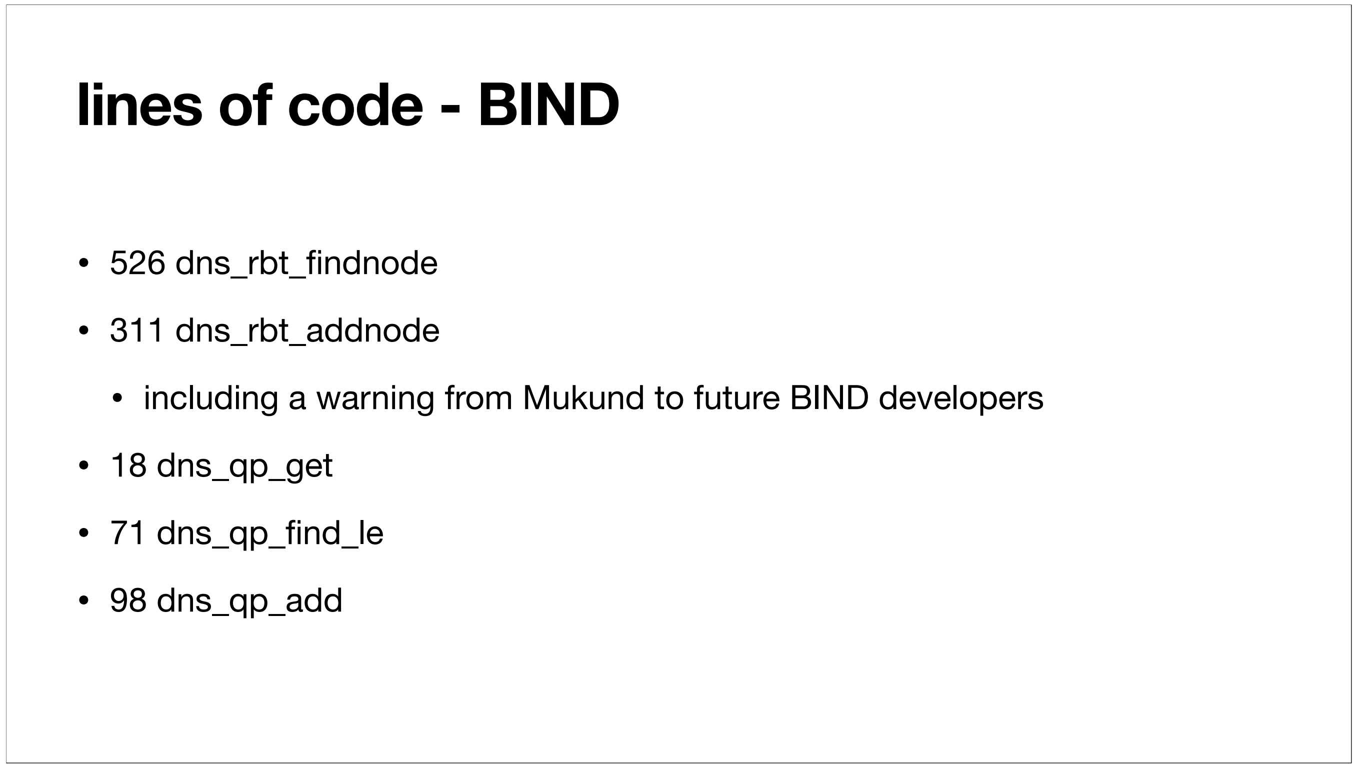
lines of code - NSD

- 629 rbtree
- 1669 radtree
- 906 qp-trie
- 1546 qp + COW

apples to apples comparison except the last one

total count for .c and .h files that implement each data structure

the benchmark used the COW version (the non-COW version uses about 150 MiB)



lines of code - BIND

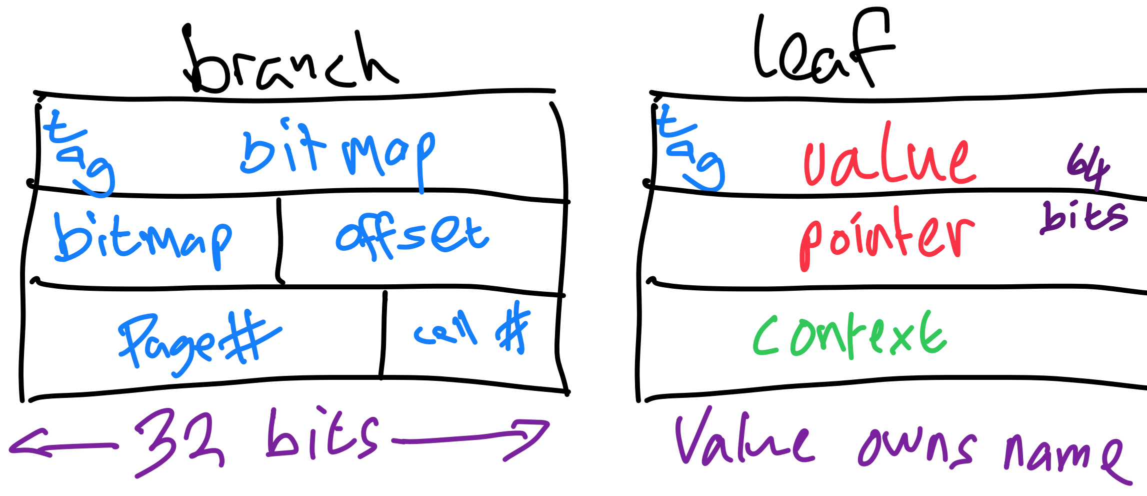
- 526 dns_rbt_findnode
- 311 dns_rbt_addnode
  - including a warning from Mukund to future BIND developers
- 18 dns_qp_get
- 71 dns_qp_find_le
- 98 dns_qp_add

apples and oranges, rbt does more?

just comparing a few functions

dns_qp_find_le is the function in the benchmark

qp node

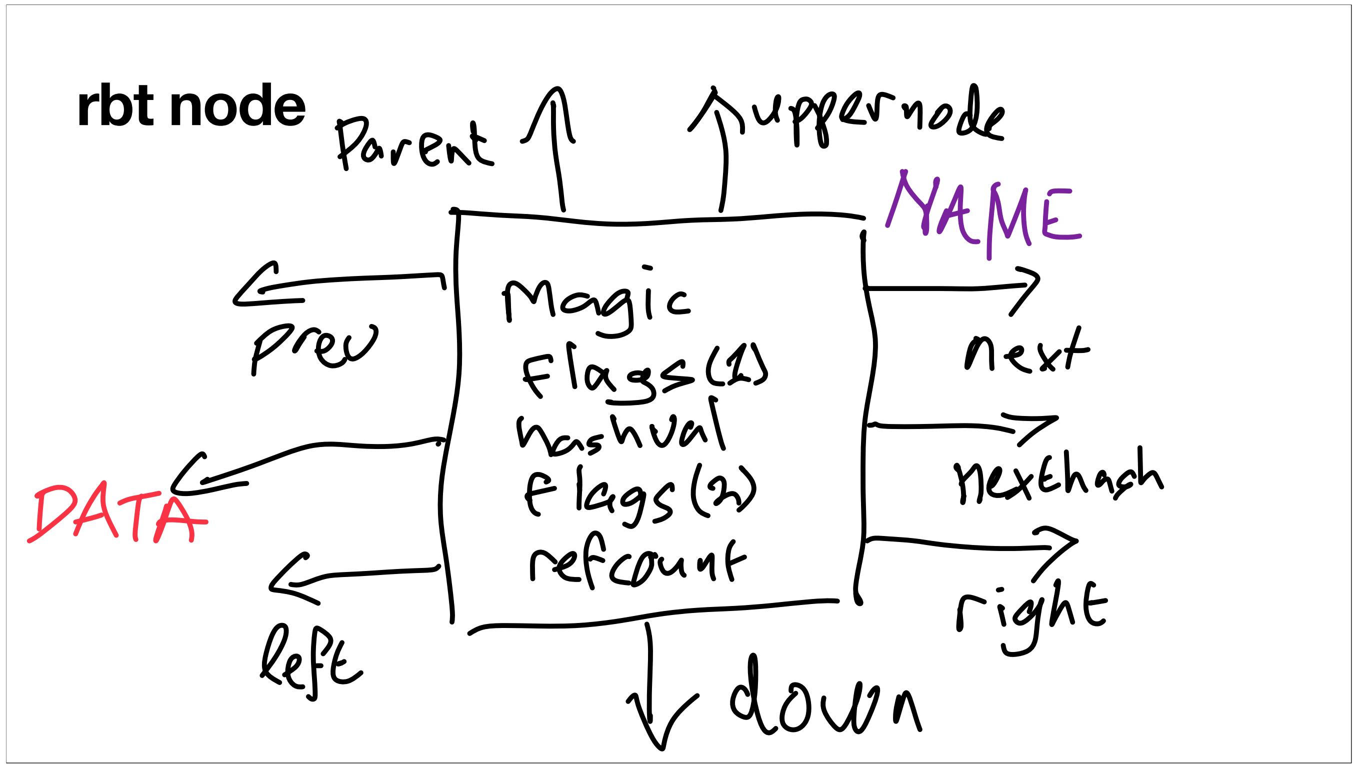


12 bytes per node

much less than one branch node per leaf node, so about 20 bytes per element, plus space for names

complete name per element

single threaded qp-trie root structure about 100 bytes, concurrent root is about 200 bytes



nine pointers (72 bytes)
5 x 32 bit words of other data (20 bytes)
plus the (partial?) name appended

100 bytes?

plus extra for nodes without data?

important question: what do we lose by throwing away all the extra stuff that an rbt node has? what do we need to move to the value structure?

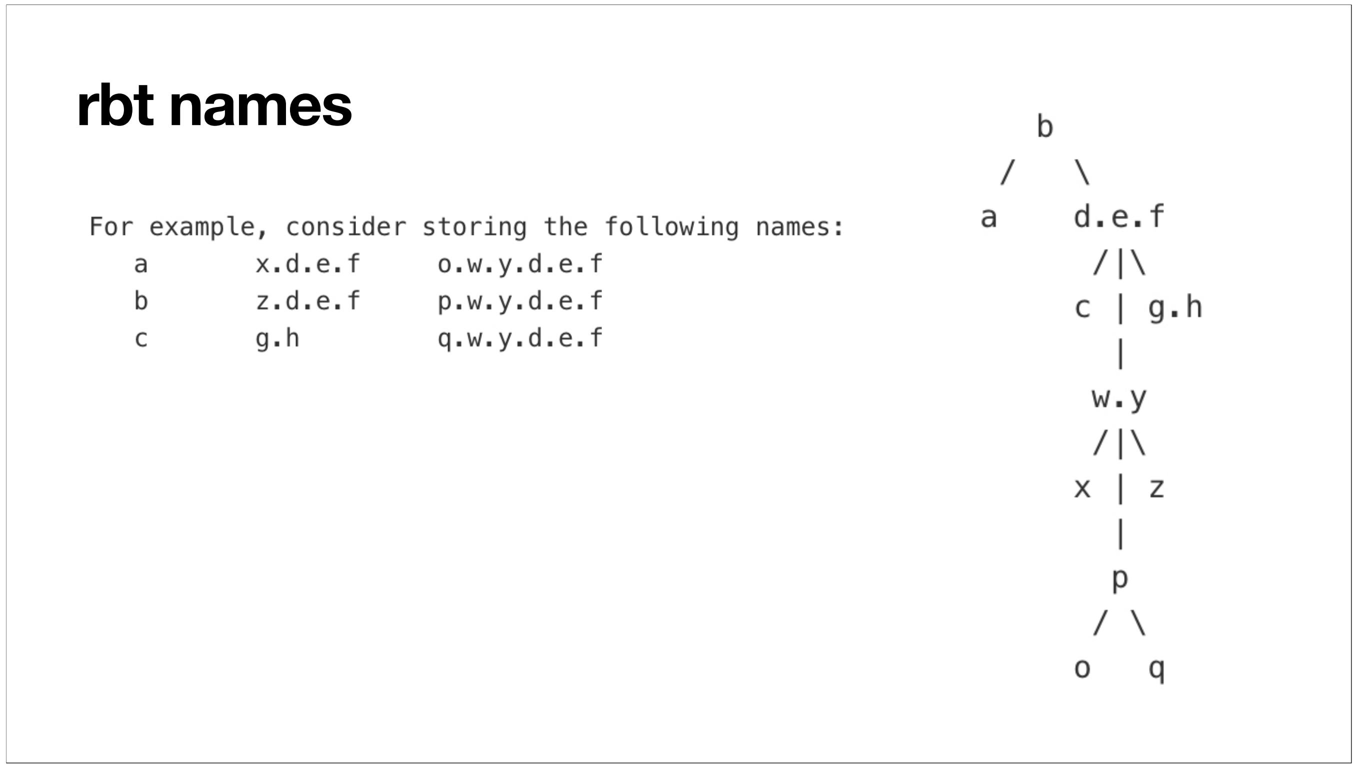
next/prev list for deferred cleanup - handled by the qp gc (1 pointer per page not 2 pointers per element)

nexthash - not needed because qp is fast?

left/right - incompatible with cow, use chains or more complicated search (dns_qp_find_neighbours?)

parent - incompatible with cow, use length of name suffix instead?

down - related to rbt tree-of-trees, not needed for qp



rbt names

```
For example, consider storing the following names:
   a       x.d.e.f     o.w.y.d.e.f
   b       z.d.e.f     p.w.y.d.e.f
   c       g.h         q.w.y.d.e.f
```

```
        b
      /   \
     a    d.e.f
           /|\
          c | g.h
            |
           w.y
           /|\
          x | z
            |
            p
           / \
          o   q
```

a major point: names are part of the rbt structure, and common suffixes are deduplicated

in the qp-trie, every leaf has its complete name (they can all be relative to the same suffix, eg the zone apex)

(i have a vague plan for how to fit names into the qp-trie nodes, but it makes things more complicated)

wip

- concurrent transactions
  - query (quick readonly)
  - snapshot (slow readonly, for xfer)
  - update - different flavours?
    - auth, infrequent, full compaction
    - cache, frequent, periodic cleanup
- small zone optimization
- leaf methods
  - get key
  - attach/detach for cow/gc
- locking, but can use urcu
- cleanup, isc style
- redo statistics counters

testing

- fuzzing with internal trie consistency checks
- mprotect() to verify cow readonly rules
- differential testing against rbt
  - needs api compatibility
  - supports unit level benchmarks
- … other ideas?

questions

- what kinds of lookup does bind need?
- is too much extra memory used for names?
- pointers between leaf values are a problem for cow
  - can the rbt chain machinery help?
- replace non-rbtdb rbt users first?
  - who would like to help?

give me the courage to change the things I can

the serenity to accept the things I cannot change

and the wisdom to know the difference