Binary search

doing it less wrong

Paul Khuong

October 30, 2014

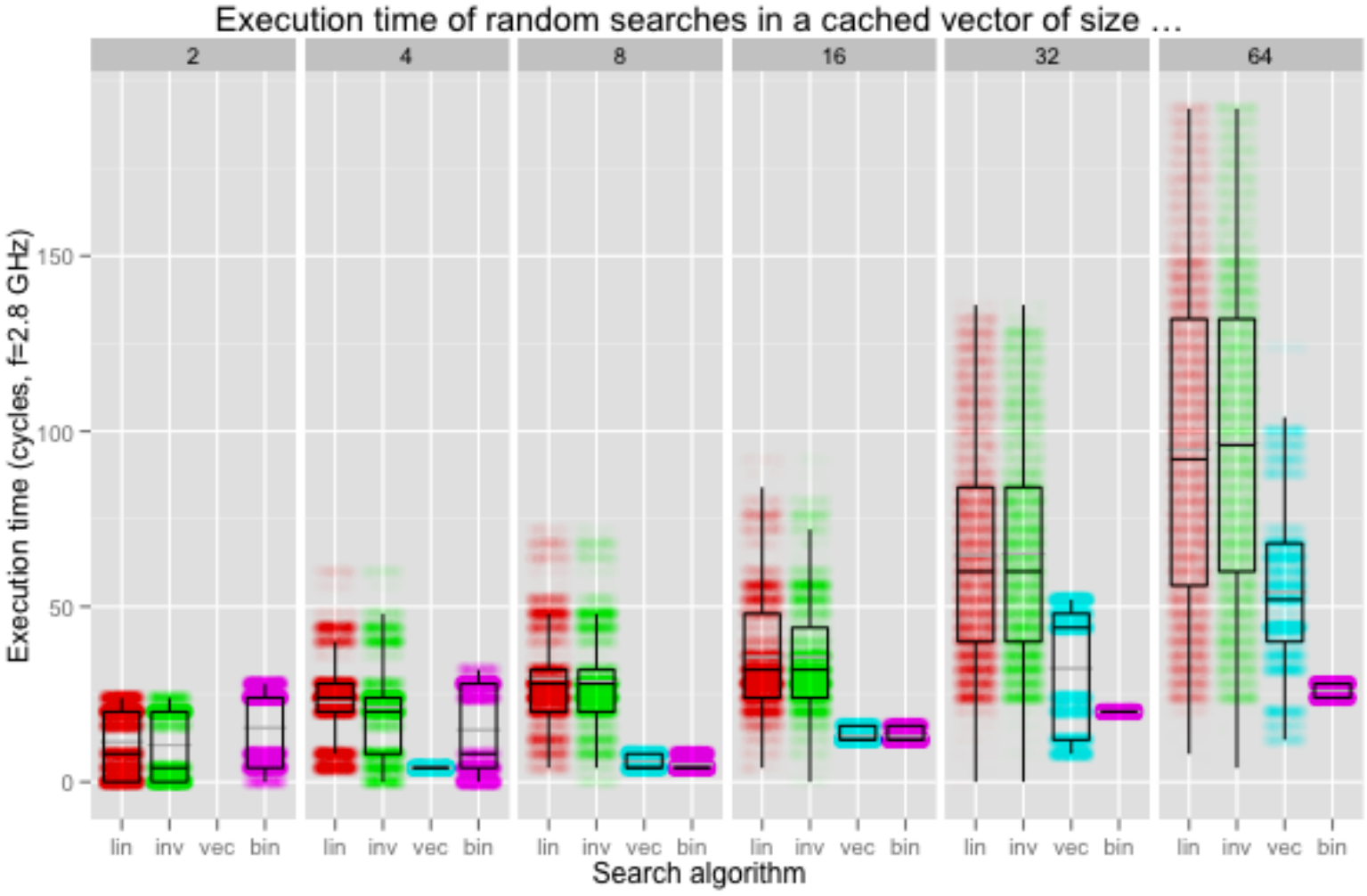
Adserver Engineer @ AppNexus

“Binary search is slow”

- “Linear search is faster for small n” (branches)
- “Fancy layouts scale better” (caches)

No branch, no misprediction

Data dependent conditional move.



http://pvk.ca/Blog/2012/07/03/binary-search-star-eliminates-star-branch-mispredictions/

Don't. . . look for early matches

```
if (*mid == needle) {
    return mid;
}
```

Don't. . . try to adjust bounds tightly

```
if (needle < *mid) {
    len = half;
} else {
    low = mid + 1;
    len -= half + 1;
}
```

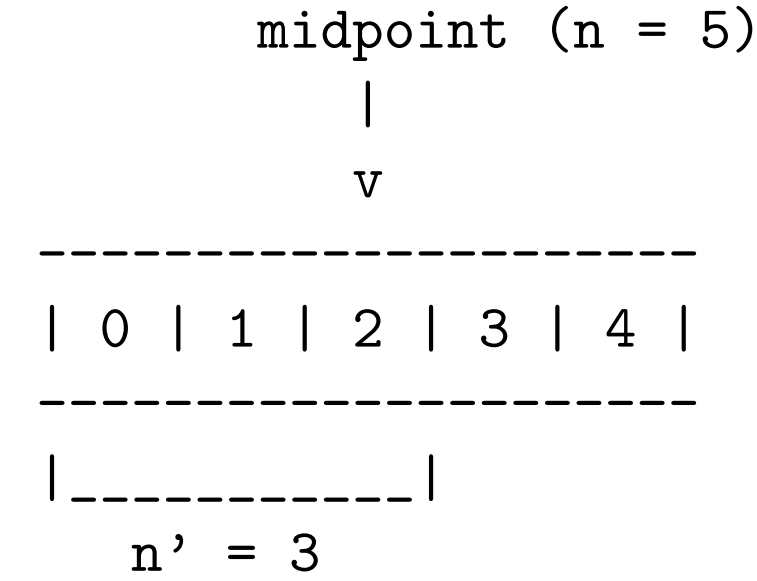
G++ STL

Don't. . . do both

```
if (comparison < 0) {
    high = mid;
} else if (comparison > 0) {
    low = mid + 1;
} else {
    return mid;
}
```

glibc, FreeBSD libc

Simple binary search

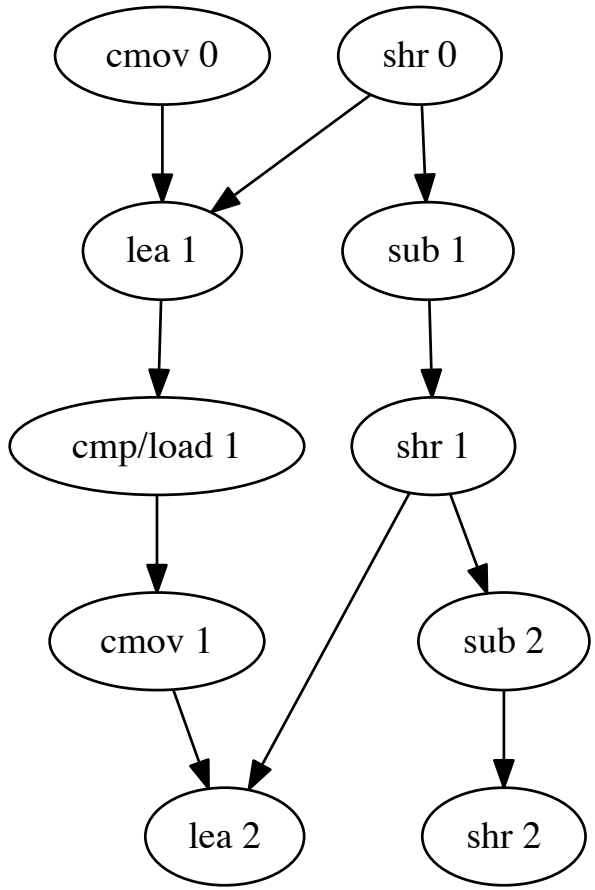



```
while ((half = n / 2) > 0) {
    mid = low + half;
    low = (*mid < needle) ? mid : low;
    n -= half;
}
```

So simple, it's AVX2-able!

Assume a decent compiler

```
loop:
 lea    (%rdx,%rcx,4), %rdi
 cmp    (%rdi), %esi
 cmovge %rdi, %rdx
 sub    %rcx, %rax
 mov    %rax, %rcx
 shr    %rcx
 jnz    loop
```



Microbenchmark

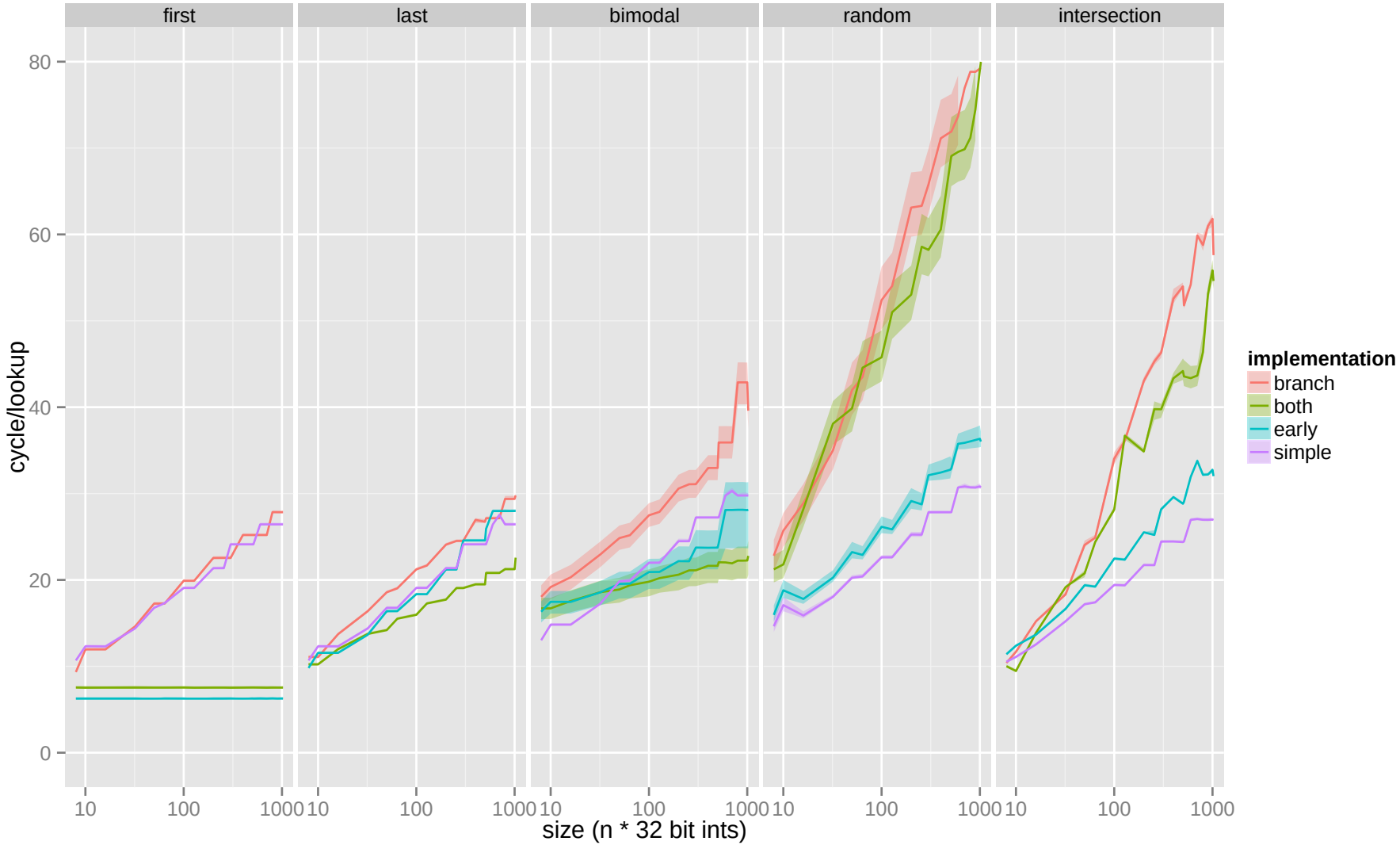
Implementations:

- branch (STL)
- both (libc)
- early (only early termination)
- simple (cmov)

Input: 32 bit ints (random, $\approx 5\%$ density)

- 8, 16, 32, . . . , 1024
- 10, 50, 100, 200, . . . , 1000

Report average of 128 lookups (median, 1st/99th percentile)


first
last
bimodal
random
intersection
cycle/lookup
0
20
40
60
80
10
100
1000
size (n * 32 bit ints)
implementation
branch
both
early
simple

Caching: $n = 2^k(-1) \Rightarrow$ aliasing issues

Midpoints:

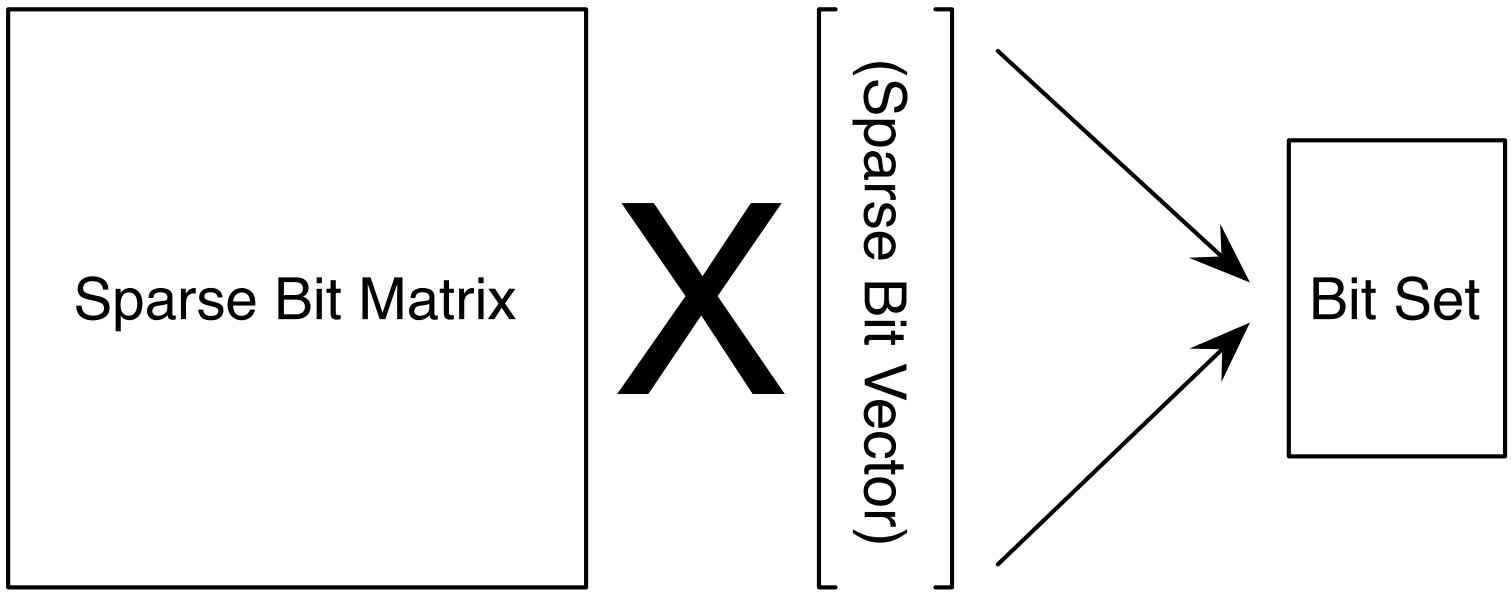
```
0x200000
0x100000
0x080000
0x040000
0x020000
0x010000
  ....
```

- Run proper microbenchmarks
- Offset "mid" point (`(n / 2) + (n / 64)`)
- 3-way ("ternary") search

In the wild: Bentley & Saxe dynamisation or $2^k \leq n < 2^{k+1}$.

`http://pvk.ca/Blog/2012/07/30/binary-search-is-a-pathological-case-for-caches/`

Practical use case: sparse bitmatrix mult + projection



a.k.a. (pre)sorted equijoin

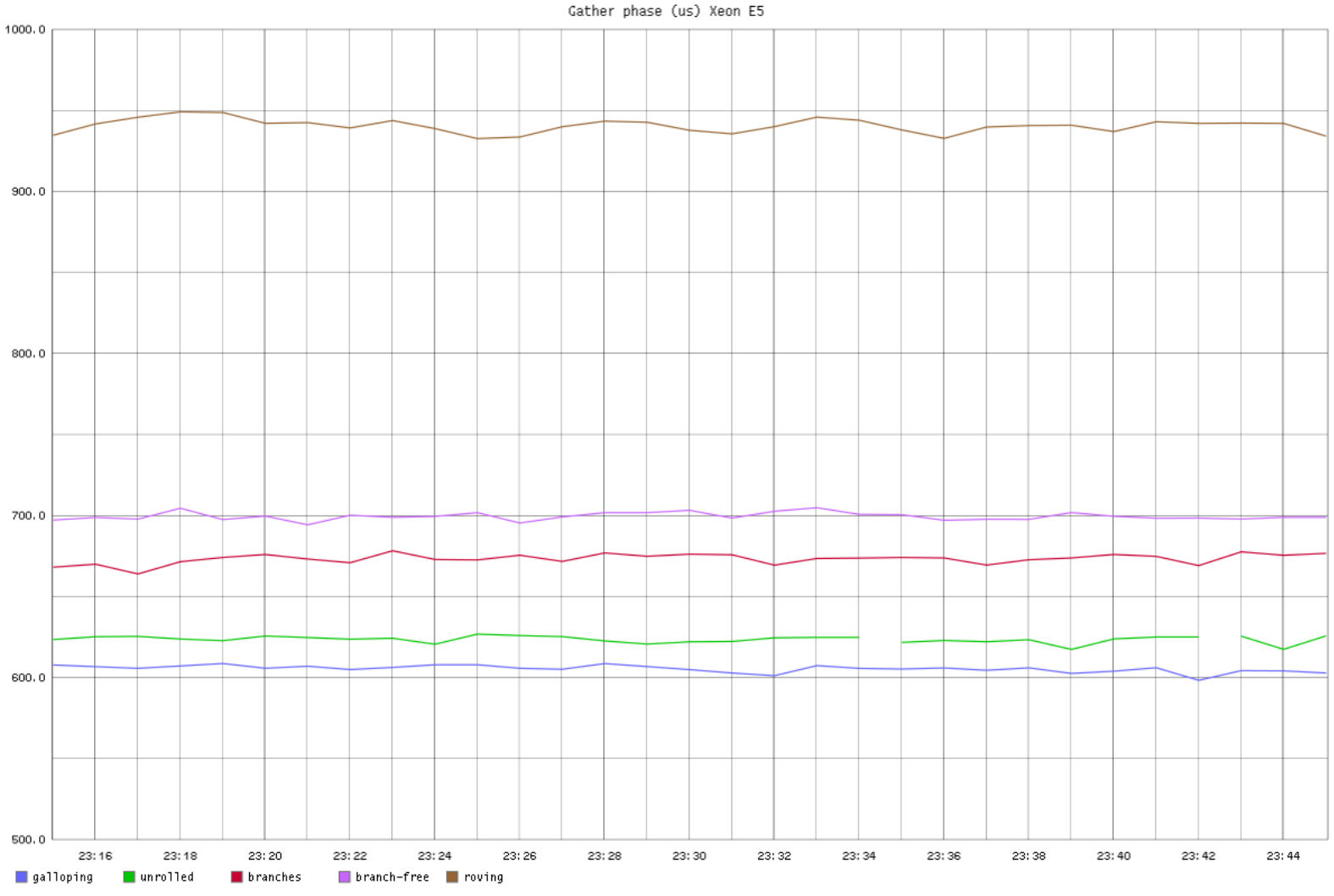
Inner loop:

- branch-free (simple)
- branches (STL)
- unrolled (3ary branch-free)

Reuse results

- roving lower bound
- galloping search

Gather phase around peak time



Sorted array search: a decent finger search

Let $\Delta = k_i - k_{i-1}$

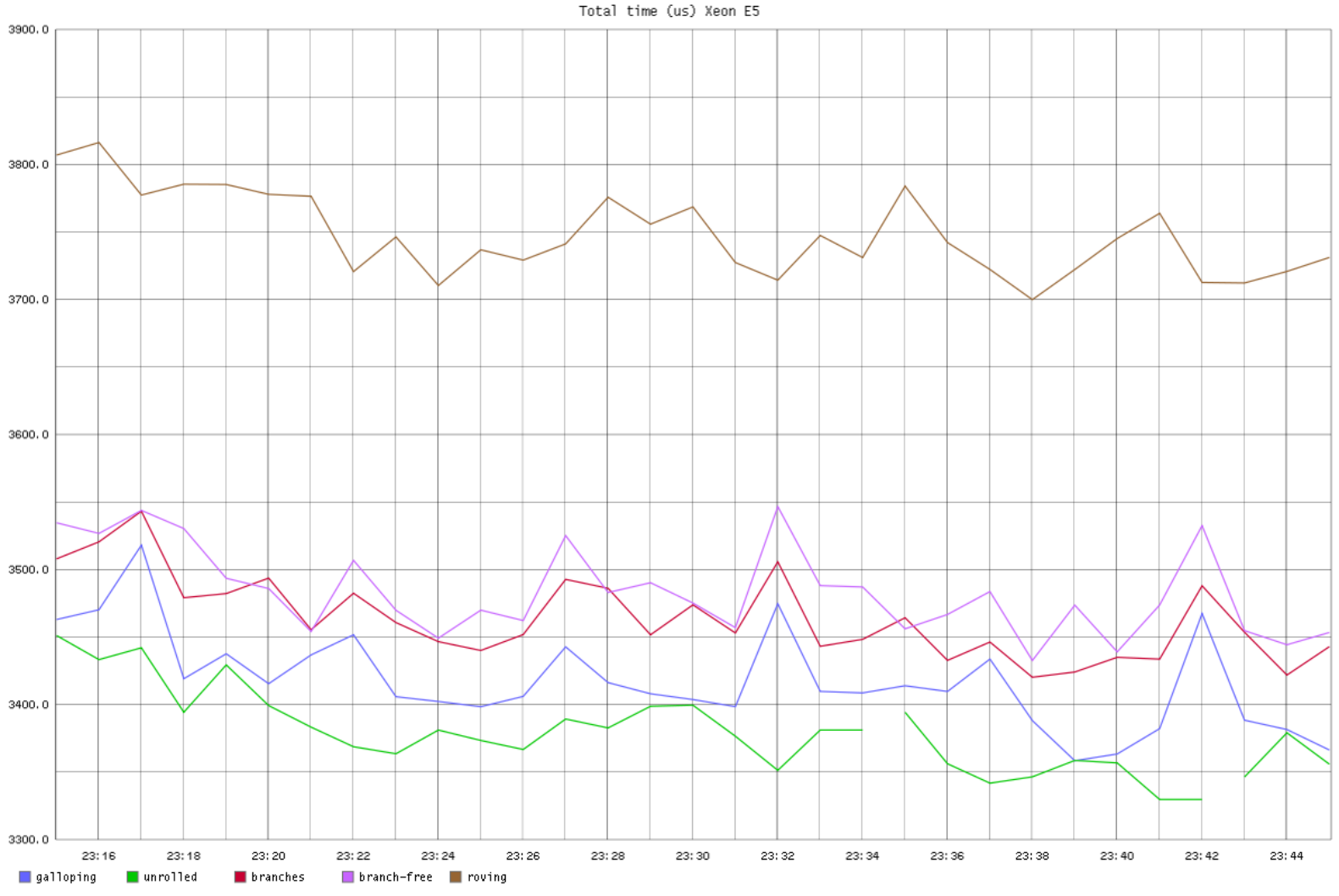
Galloping search: $\approx 2 \lg \Delta$ comparisons

3ary search: $\approx 2 \log_3 \Delta$ D$ misses

Roving search: $\approx \lg n$ D$ misses

```
       :
       :      000000000063f890 <sparse_tsearch_upto_14>:
  1.07 :        63f890:       48 c1 e6 06             shl    $0x6,%rsi
  0.03 :        63f894:       55                      push   %rbp
  0.43 :        63f895:       48 8d 84 3e 00 00 f8    lea    0xfffffffffff80000(%rsi,%rdi,1),%rax
  0.00 :        63f89c:       ff
  0.28 :        63f89d:       48 89 e5                mov    %rsp,%rbp
  0.20 :        63f8a0:       39 10                   cmp    %edx,(%rax)
  0.87 :        63f8a2:       48 0f 42 f8             cmovb  %rax,%rdi
  1.48 :        63f8a6:       3b 97 c0 aa 02 00       cmp    0x2aac0(%rdi),%edx
  3.67 :        63f8ac:       48 8d 8f 40 55 05 00    lea    0x55540(%rdi),%rcx
  0.00 :        63f8b3:       48 8d 87 c0 aa 02 00    lea    0x2aac0(%rdi),%rax
  0.03 :        63f8ba:       48 0f 46 c7             cmovbe %rdi,%rax
  1.48 :        63f8be:       3b 11                   cmp    (%rcx),%edx
  0.00 :        63f8c0:       48 0f 46 c8             cmovbe %rax,%rcx
  0.97 :        63f8c4:       3b 91 c0 e3 00 00       cmp    0xe3c0(%rcx),%edx
  3.46 :        63f8ca:       48 8d 81 00 c7 01 00    lea    0x1c700(%rcx),%rax
  0.15 :        63f8d1:       48 8d b1 c0 e3 00 00    lea    0xe3c0(%rcx),%rsi
  0.00 :        63f8d8:       48 0f 46 f1             cmovbe %rcx,%rsi
  1.73 :        63f8dc:       3b 10                   cmp    (%rax),%edx
  0.03 :        63f8de:       48 0f 46 c6             cmovbe %rsi,%rax
  1.45 :        63f8e2:       3b 90 00 4c 00 00       cmp    0x4c00(%rax),%edx
  3.89 :        63f8e8:       48 8d 88 c0 97 00 00    lea    0x97c0(%rax),%rcx
  0.00 :        63f8ef:       48 8d b0 00 4c 00 00    lea    0x4c00(%rax),%rsi
  0.00 :        63f8f6:       48 0f 46 f0             cmovbe %rax,%rsi
  1.86 :        63f8fa:       3b 11                   cmp    (%rcx),%edx
  0.10 :        63f8fc:       48 0f 46 ce             cmovbe %rsi,%rcx
  1.55 :        63f900:       3b 91 80 19 00 00       cmp    0x1980(%rcx),%edx
  4.30 :        63f906:       48 8d 81 80 32 00 00    lea    0x3280(%rcx),%rax
  0.10 :        63f90d:       48 8d b1 80 19 00 00    lea    0x1980(%rcx),%rsi
  0.00 :        63f914:       48 0f 46 f1             cmovbe %rcx,%rsi
  1.58 :        63f918:       3b 10                   cmp    (%rax),%edx
  0.05 :        63f91a:       48 0f 46 c6             cmovbe %rsi,%rax
  1.86 :        63f91e:       3b 90 80 08 00 00       cmp    0x880(%rax),%edx
  4.89 :        63f924:       48 8d 88 00 11 00 00    lea    0x1100(%rax),%rcx
  0.00 :        63f92b:       48 8d b0 80 08 00 00    lea    0x880(%rax),%rsi
  0.00 :        63f932:       48 0f 46 f0             cmovbe %rax,%rsi
  2.04 :        63f936:       3b 11                   cmp    (%rcx),%edx
  0.33 :        63f938:       48 0f 46 ce             cmovbe %rsi,%rcx
  1.58 :        63f93c:       3b 91 00 03 00 00       cmp    0x300(%rcx),%edx
  7.66 :        63f942:       48 8d 81 80 05 00 00    lea    0x580(%rcx),%rax
  0.05 :        63f949:       48 8d b1 00 03 00 00    lea    0x300(%rcx),%rsi
  0.00 :        63f950:       48 0f 46 f1             cmovbe %rcx,%rsi
  1.83 :        63f954:       3b 10                   cmp    (%rax),%edx
  1.20 :        63f956:       48 0f 46 c6             cmovbe %rsi,%rax
  1.25 :        63f95a:       3b 90 00 01 00 00       cmp    0x100(%rax),%edx
 11.30 :        63f960:       48 8d 88 00 02 00 00    lea    0x200(%rax),%rcx
  0.00 :        63f967:       48 8d b0 00 01 00 00    lea    0x100(%rax),%rsi
  0.03 :        63f96e:       48 0f 46 f0             cmovbe %rax,%rsi
  1.91 :        63f972:       3b 11                   cmp    (%rcx),%edx
  3.26 :        63f974:       48 0f 46 ce             cmovbe %rsi,%rcx
  1.12 :        63f978:       3b 91 80 00 00 00       cmp    0x80(%rcx),%edx
 13.82 :        63f97e:       48 8d b1 80 00 00 00    lea    0x80(%rcx),%rsi
  0.00 :        63f985:       48 0f 46 f1             cmovbe %rcx,%rsi
  2.21 :        63f989:       3b 56 40                cmp    0x40(%rsi),%edx
 11.99 :        63f98c:       48 8d 46 40             lea    0x40(%rsi),%rax
  0.00 :        63f990:       c9                      leaveq
  0.94 :        63f991:       48 0f 46 c6             cmovbe %rsi,%rax
```

And some spooky action at a distance



Sorted arrays work well on contemporary μarch

Don't be (too) clever:

- Careful with branches
- Avoid cache aliasing/bad benchmarks
- Reuse bounds when repeating searches

Cleverness is useful, but getting simple right goes a long way.