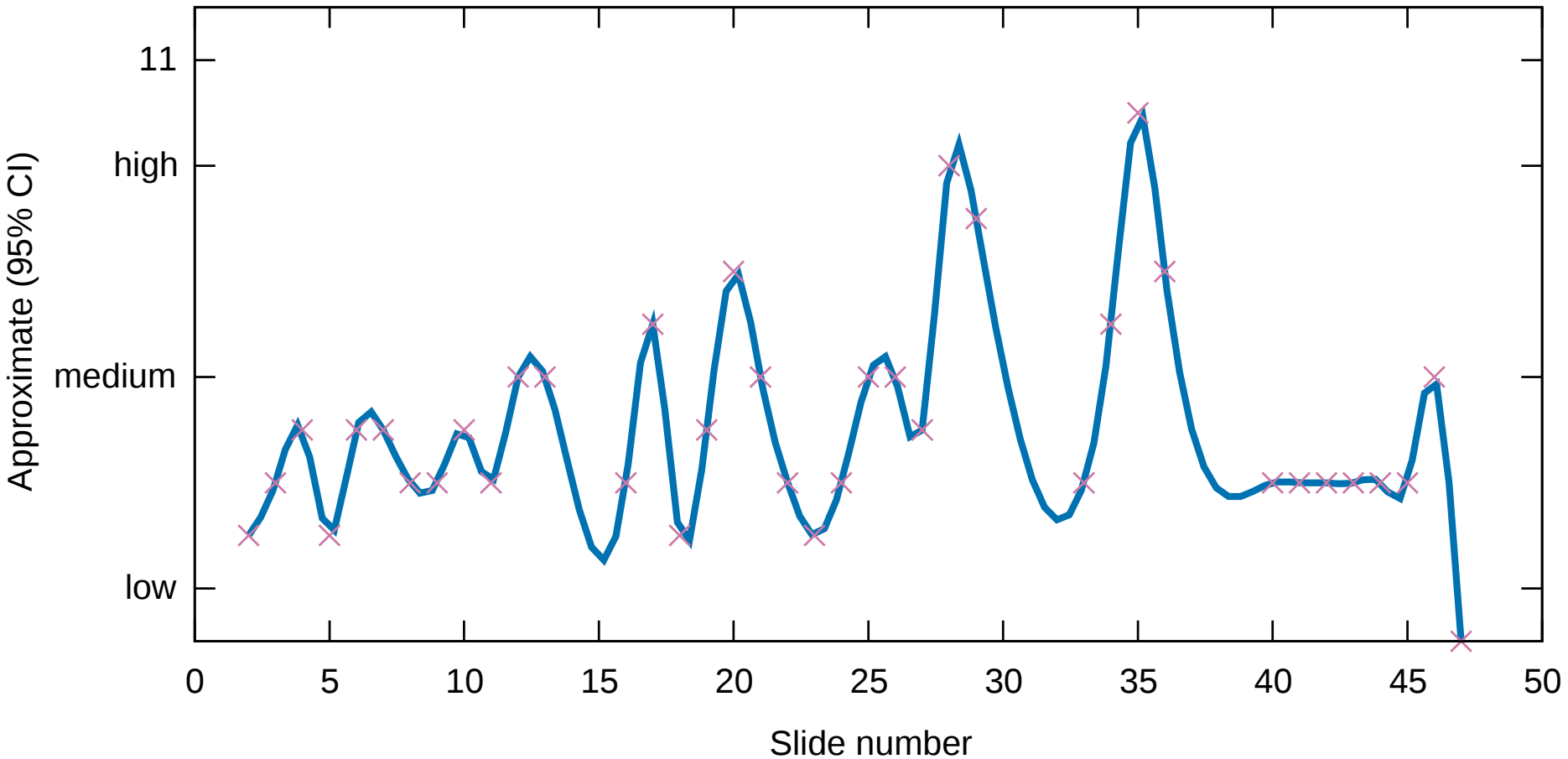
Solr sparse faceting

Everything counts in large amounts

@TokeEskildsen
State and University Library, Denmark
https://tokee.github.io/lucene-solr/

Presentation tech level



Nothing To Fear

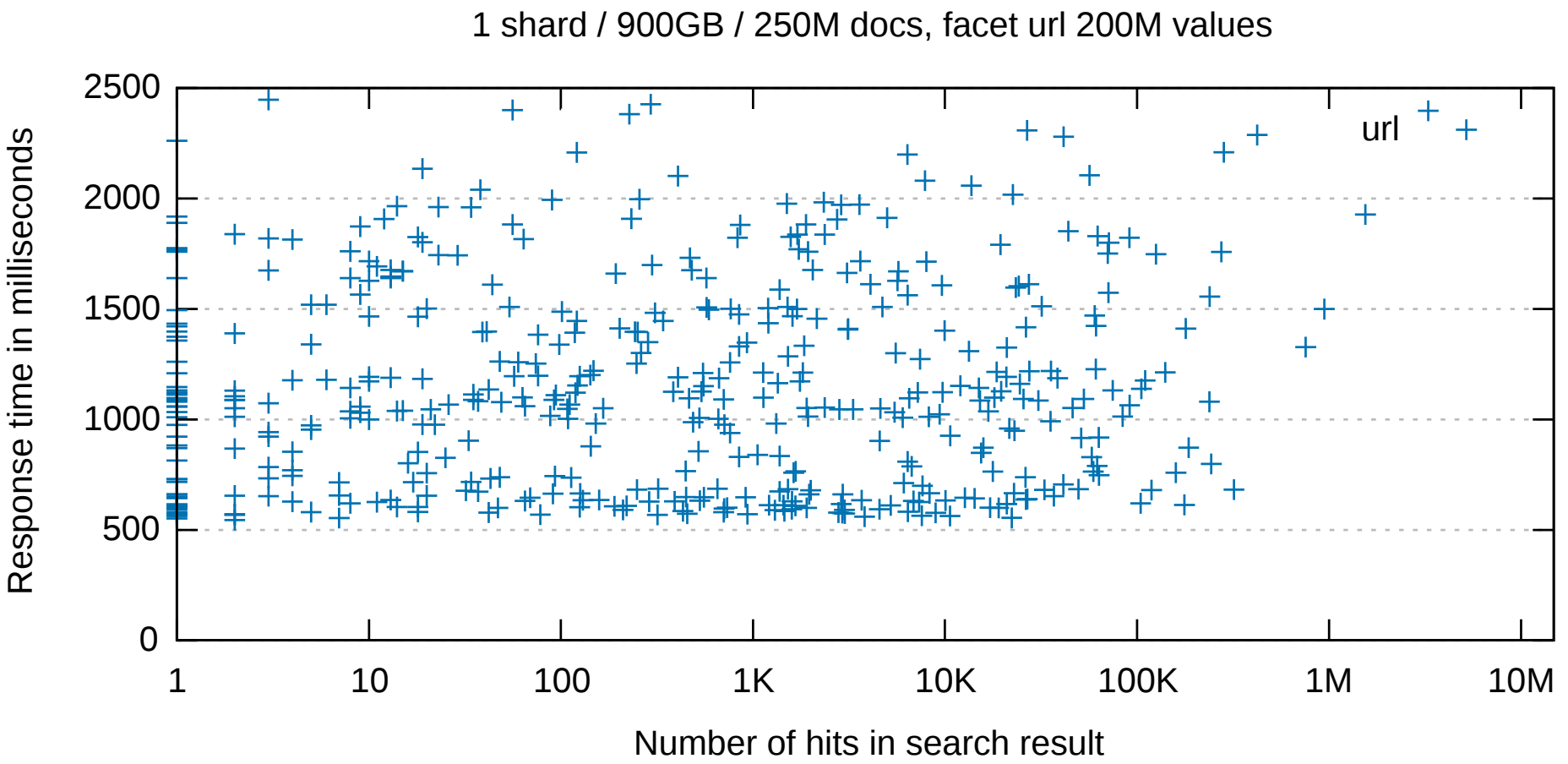
- 500TB+ web resources from Danish Net Archive
- Estimated 50TB Solr index data when finished
- 3 machines of 16 CPU cores, 256GB RAM, 25 * 900GB SSD
  - Each machine holds: 25 Solrs
    - Each Solr holds: 1 optimized shard with 900GB / 250M docs
- Shards build externally, one at a time

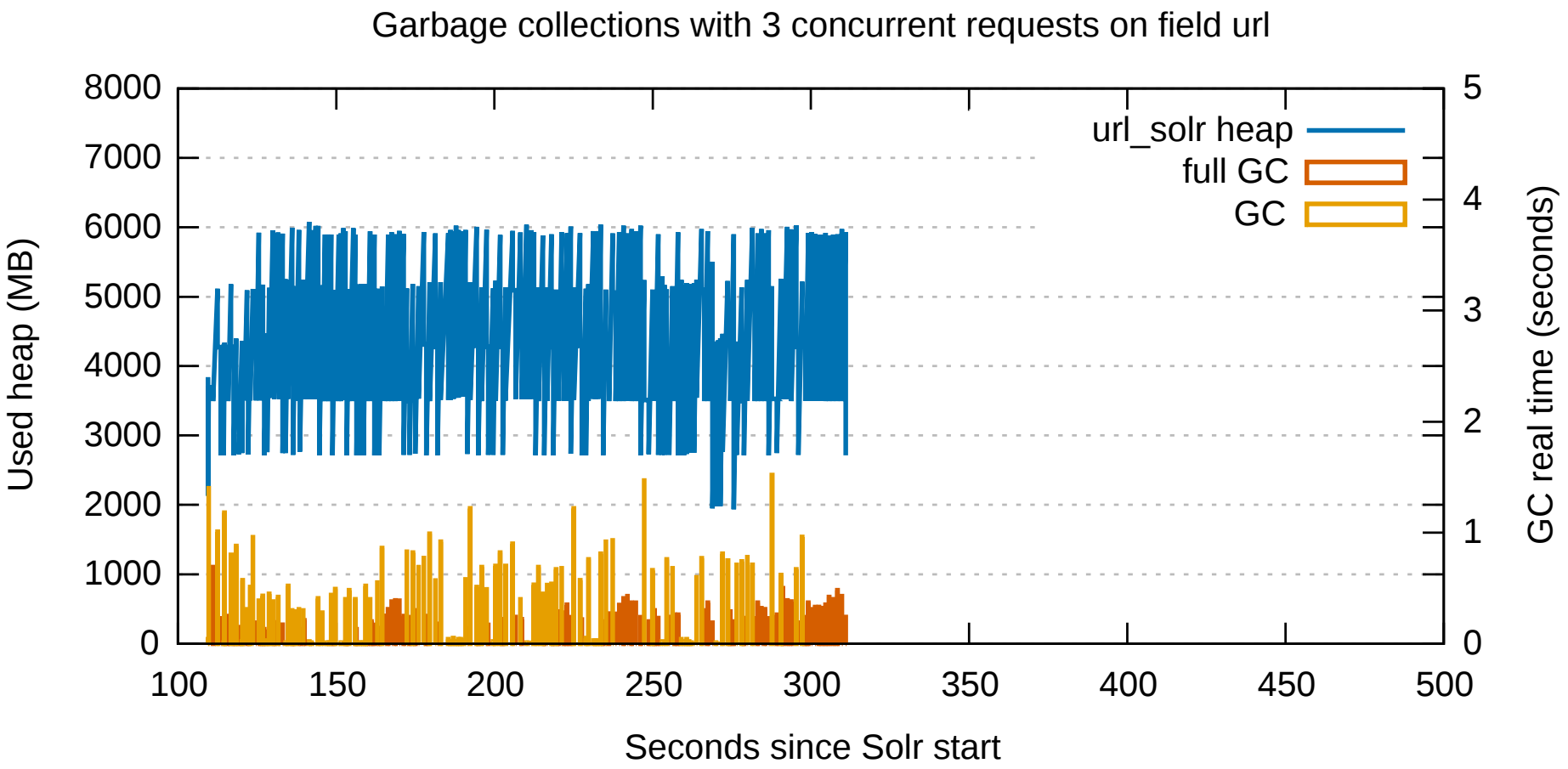
- (Optimizations also relevant for smaller setups)

Pipeline

```
counter = new int[ordinals]
for docID: result.getDocIDs()
  for ordinal: getOrdinals(docID)
    counter[ordinal]++

for ordinal = 0 ; ordinal < counters.length ; ordinal++
  priorityQueue.add(ordinal, counter[ordinal])
```

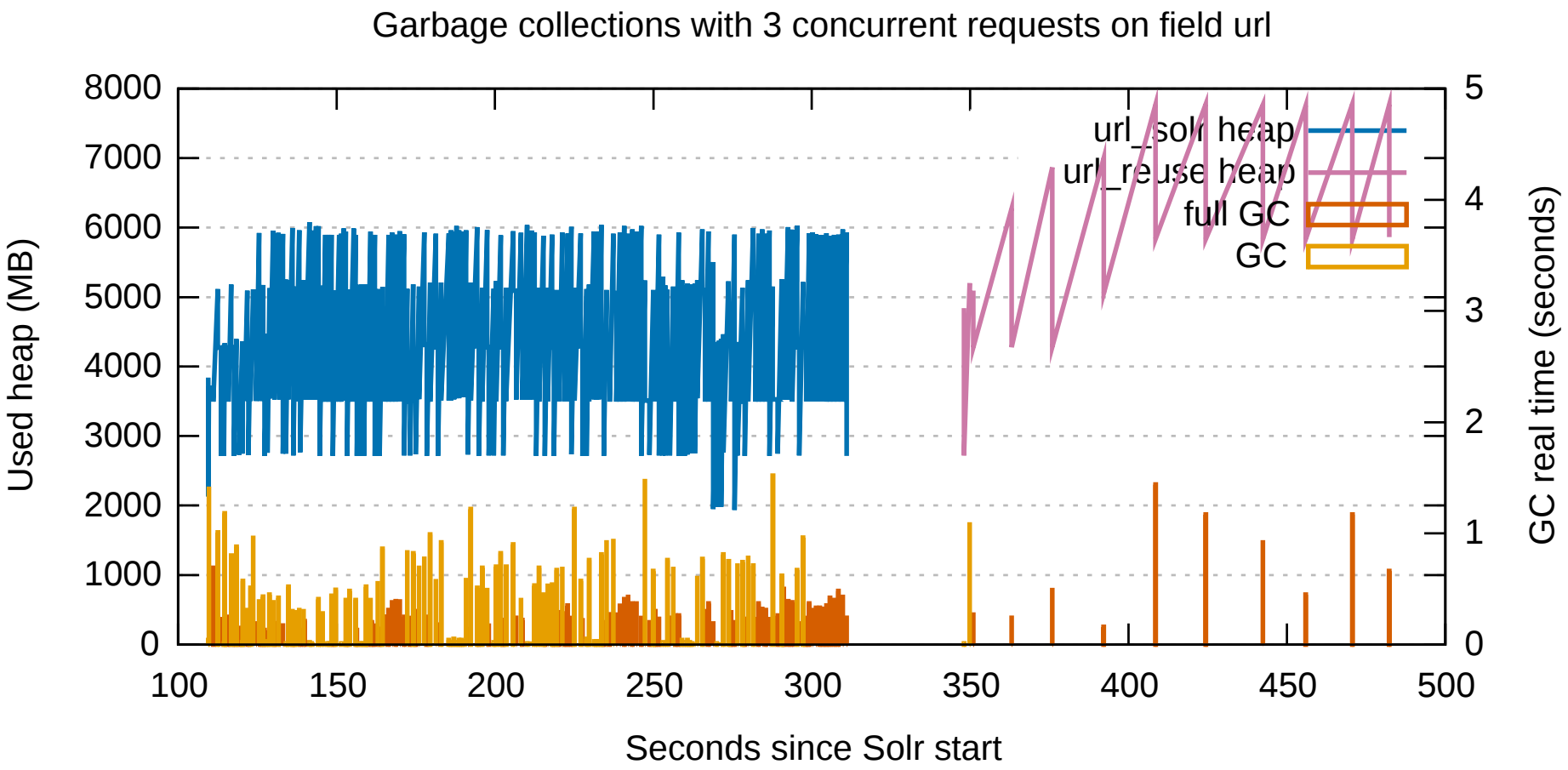
ord	term	counter
0	A	0
1	B	3
2	C	0
3	D	1006
4	E	1
5	F	1
6	G	0
7	H	0
8	I	3

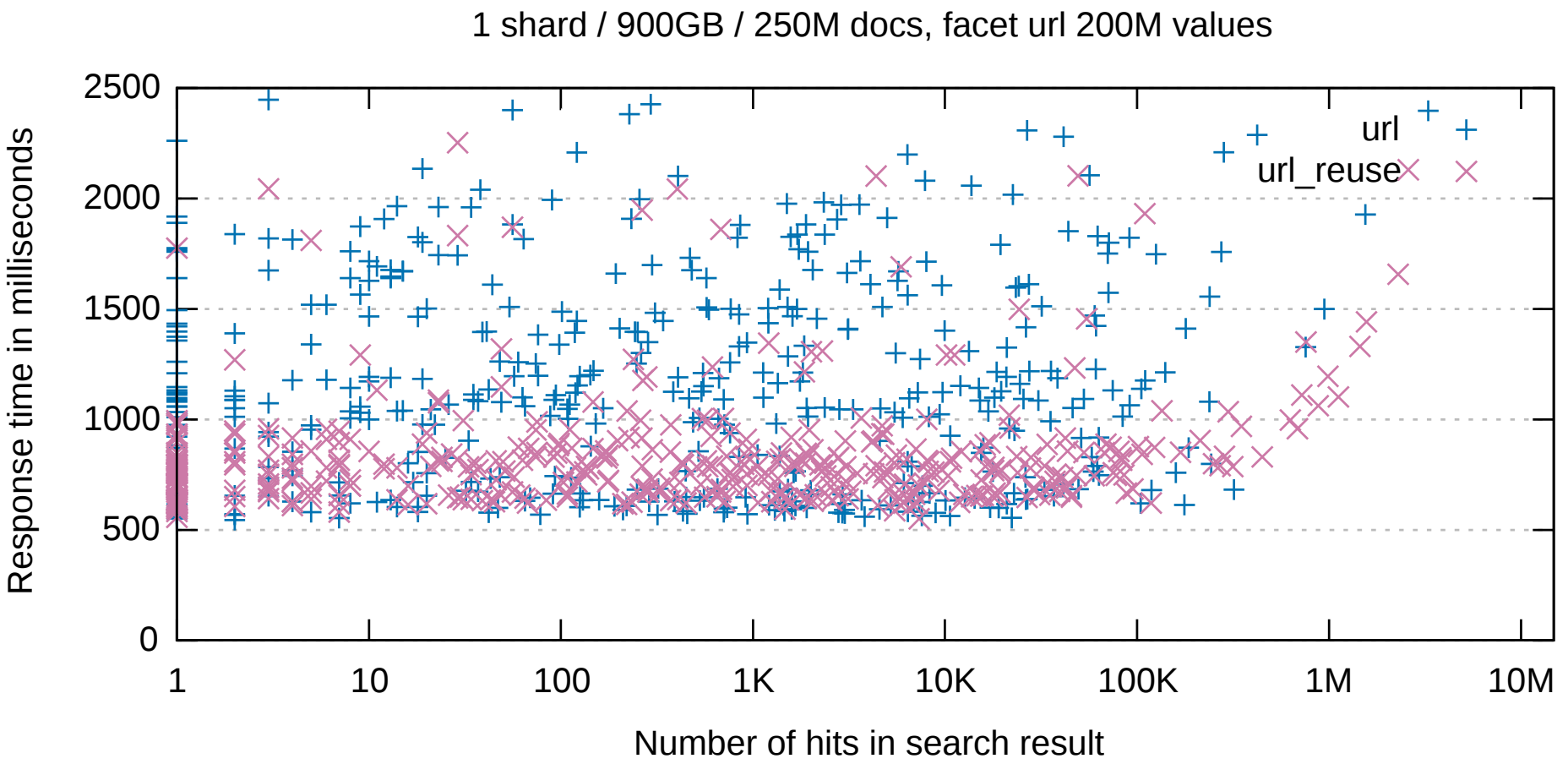

1 shard / 900GB / 250M docs, facet url 200M values
Response time in milliseconds
2500
2000
1500
1000
500
0
1
10
100
1K
10K
100K
1M
10M
Number of hits in search result
url

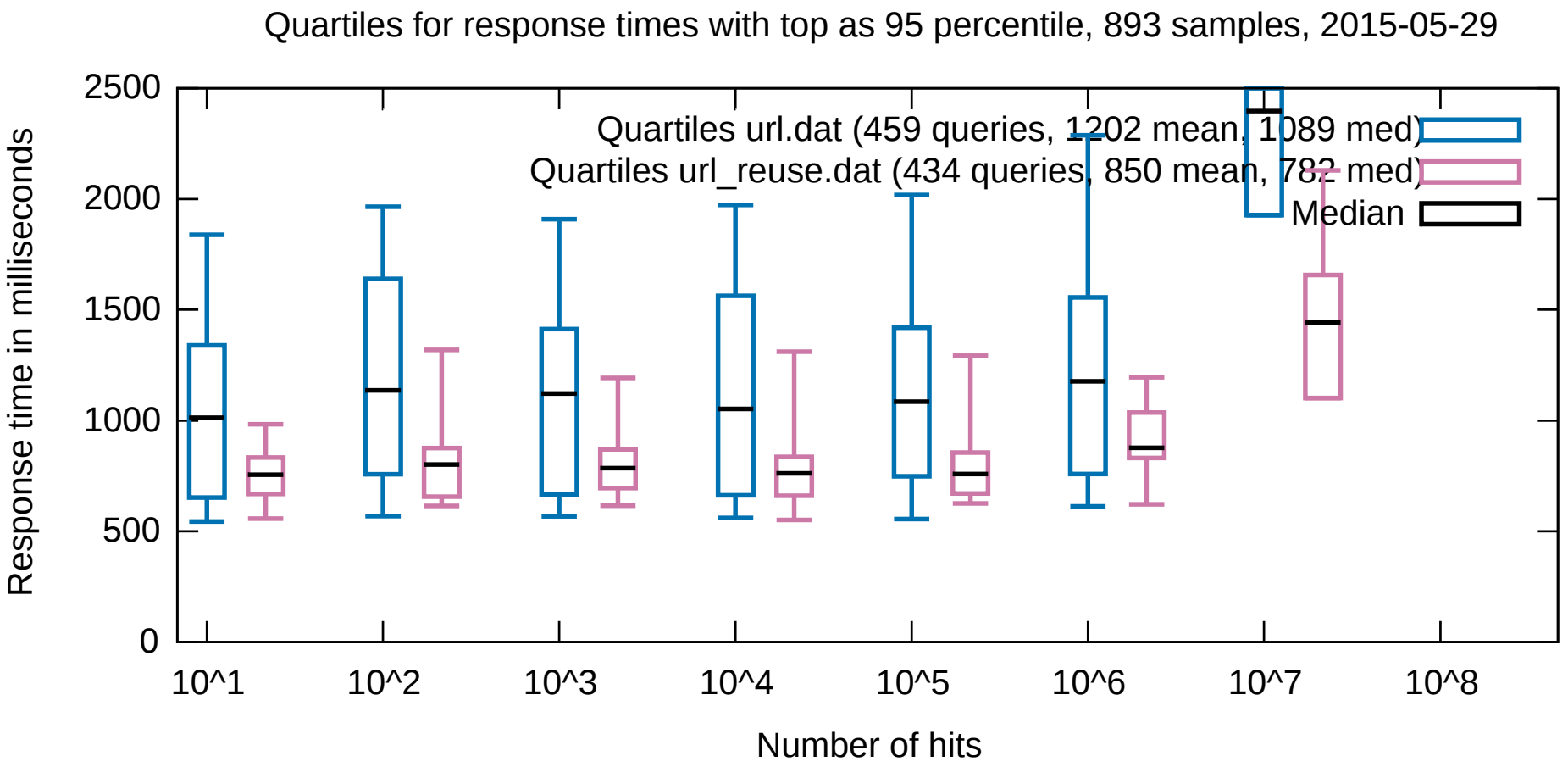

Garbage collections with 3 concurrent requests on field url
Used heap (MB)
GC real time (seconds)
Seconds since Solr start
url_solr heap
full GC
GC
8000
7000
6000
5000
4000
3000
2000
1000
0
5
4
3
2
1
0
100
150
200
250
300
350
400
450
500

Recycle

```
counter = pool.getCounter()
for docID: result.getDocIDs()
  for ordinal: getOrdinals(docID)
    counter[ordinal]++

for ordinal = 0 ; ordinal < counters.length ; ordinal++
  priorityQueue.add(ordinal, counter[ordinal])

pool.release(counter)
```


Garbage collections with 3 concurrent requests on field url
Used heap (MB)
GC real time (seconds)
Seconds since Solr start
url_solr heap
url_reuse heap
full GC
GC
8000
7000
6000
5000
4000
3000
2000
1000
0
5
4
3
2
1
0
100
150
200
250
300
350
400
450
500


1 shard / 900GB / 250M docs, facet url 200M values
Response time in milliseconds
2500
2000
1500
1000
500
0
1
10
100
1K
10K
100K
1M
10M
Number of hits in search result
url
url_reuse


Quartiles for response times with top as 95 percentile, 893 samples, 2015-05-29
Quartiles url.dat (459 queries, 1202 mean, 1089 med)
Quartiles url_reuse.dat (434 queries, 850 mean, 782 med)
Median
Response time in milliseconds
2500
2000
1500
1000
500
0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
Number of hits

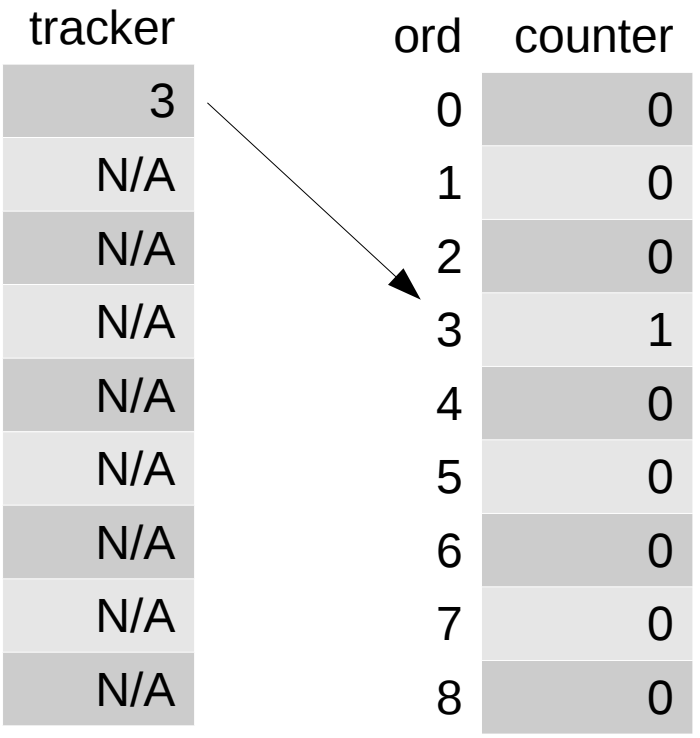
Counting

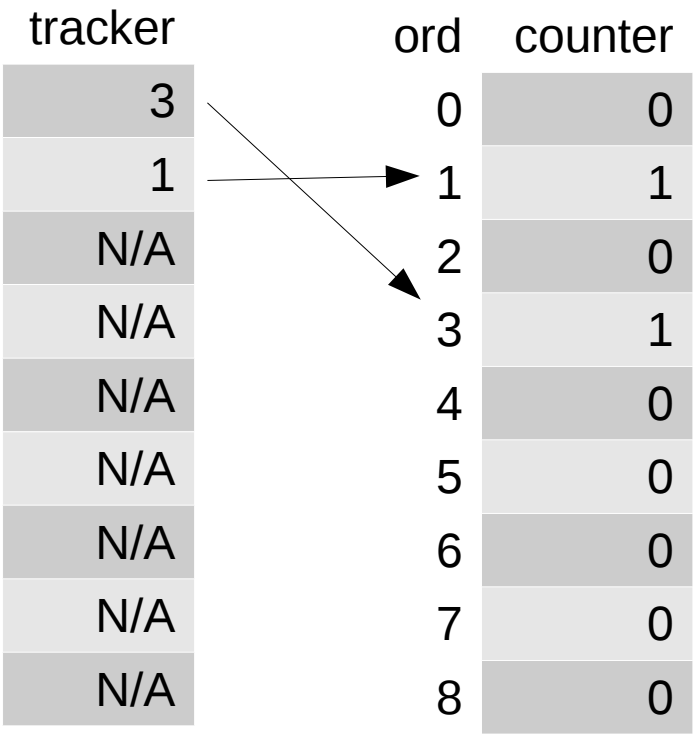
```
counter = pool.getCounter()
for docID: result.getDocIDs()
  for ordinal: getOrdinals(docID)
    counter[ordinal]++

for ordinal = 0 ; ordinal < counters.length ; ordinal++
  priorityQueue.add(ordinal, counter[ordinal])

pool.release(counter)
```

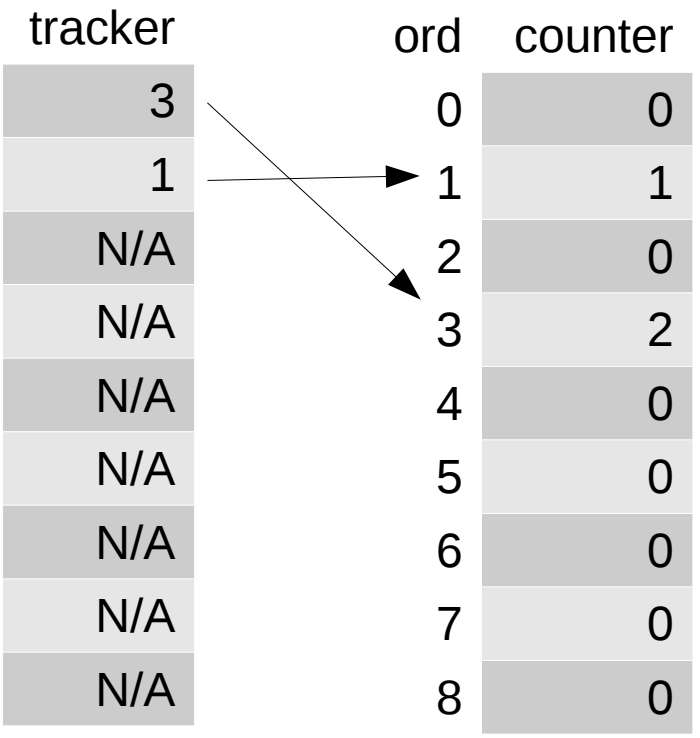
ord	term	counter
0	A	0
1	B	3
2	C	0
3	D	1006
4	E	1
5	F	1
6	G	0
7	H	0
8	I	3

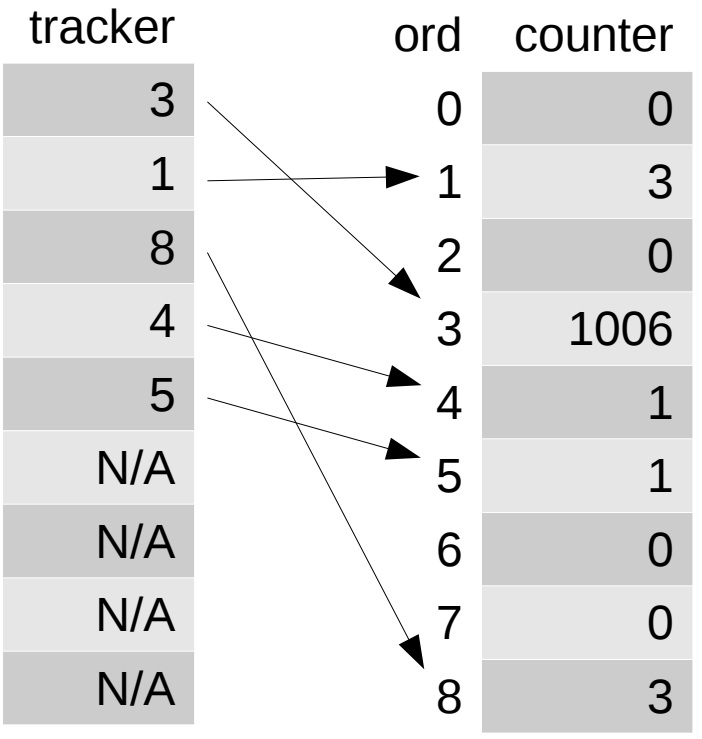
tracker
N/A
N/A
N/A
N/A
N/A
N/A
N/A
N/A
N/A

ord	counter
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0


tracker
3
N/A
N/A
N/A
N/A
N/A
N/A
N/A
N/A
ord
0
1
2
3
4
5
6
7
8
counter
0
0
0
1
0
0
0
0
0

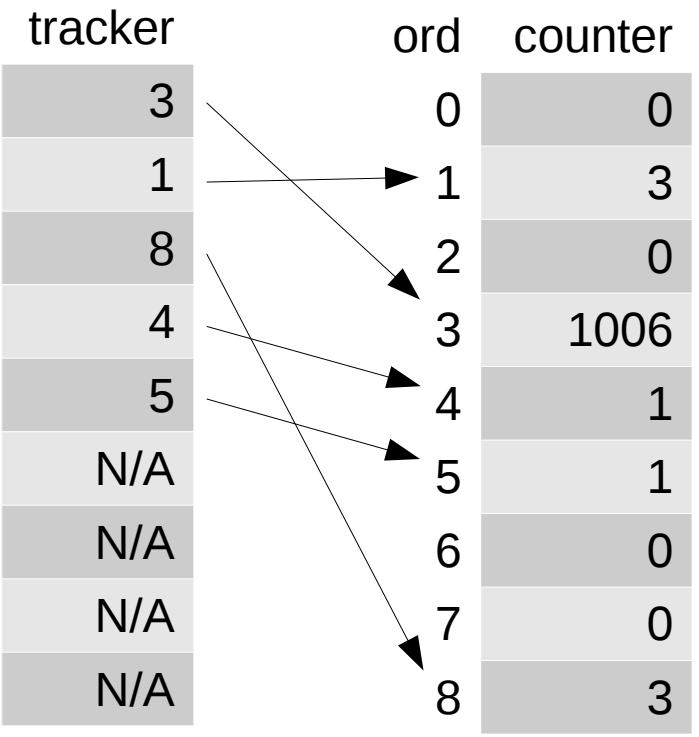

tracker
3
1
N/A
N/A
N/A
N/A
N/A
N/A
N/A
ord
0
1
2
3
4
5
6
7
8
counter
0
1
0
1
0
0
0
0
0

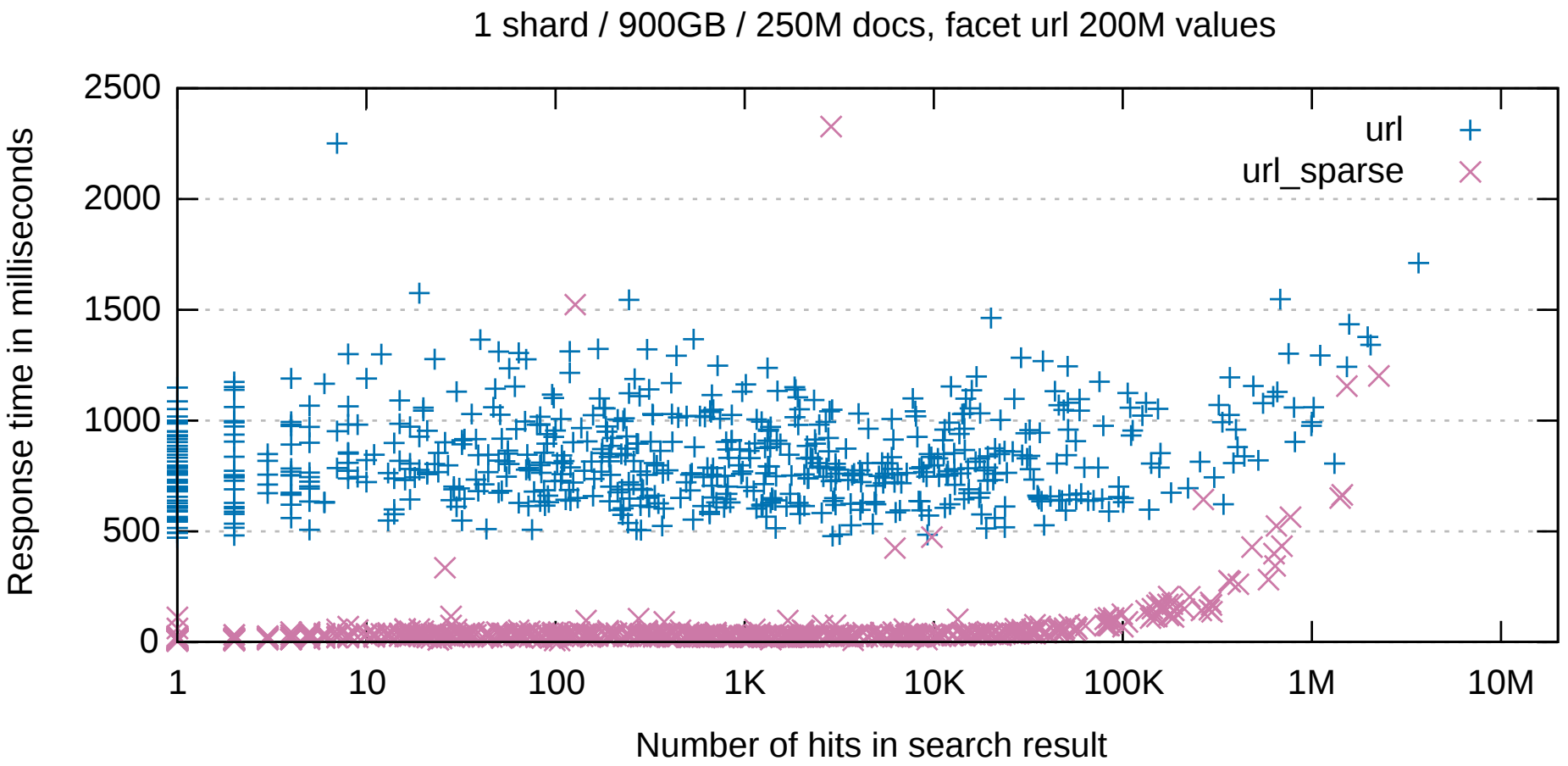

tracker
3
1
N/A
N/A
N/A
N/A
N/A
N/A
N/A
ord
0
1
2
3
4
5
6
7
8
counter
0
1
0
2
0
0
0
0
0


tracker
3
1
8
4
5
N/A
N/A
N/A
N/A
ord
0
1
2
3
4
5
6
7
8
counter
0
3
0
1006
1
1
0
0
3

Sparse counting

```
counter = pool.getCounter()
for docID: result.getDocIDs()
  for ordinal: getOrdinals(docID)
    if counter[ordinal]++ == 0 && tracked < maxTracked
      tracker[tracked++] = ordinal
if tracked < maxTracked
  for i = 0 ; i < tracked ; i++
    priorityQueue.add(tracker[i], counter[tracker[i]])
else
  for ordinal = 0 ; ordinal < counter.length ; ordinal++
    priorityQueue.add(ordinal, counter[ordinal])
```




1 shard / 900GB / 250M docs, facet url 200M values
Response time in milliseconds
2500
2000
1500
1000
500
0
1
10
100
1K
10K
100K
1M
10M
Number of hits in search result
url
url_sparse

Get the Balance Right

Phase 1) All shards perform faceting

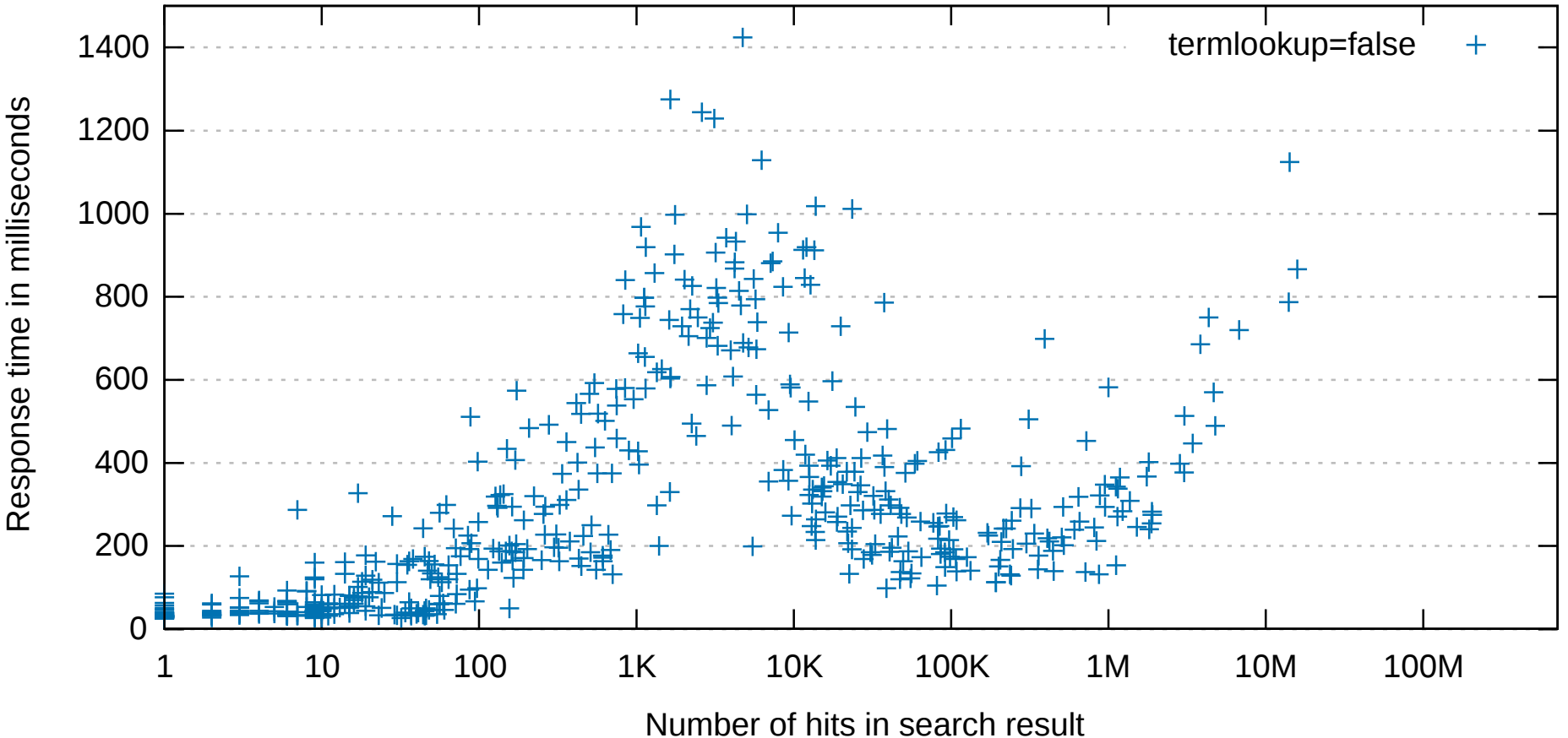
The Merger calculates top-X terms

Phase 2) The term counts are requested from the shards that did not return them in phase 1

```
for term: query.getTerms()
  result.add(term, searcher.numDocs(
    query(field:term), base.getDocIDs()
  ).hitCount)
```

Pit of Pain™

9 shards / 7TB / 2.3G docs, 256GB RAM, 16 cores, facet 1.1M values/shard



Alternative fine counting

```
counter = pool.getCounter()
for docID: result.getDocIDs()
  for ordinal: getOrdinals(docID)
    counter.increment(ordinal)
```

} Same as phase 1

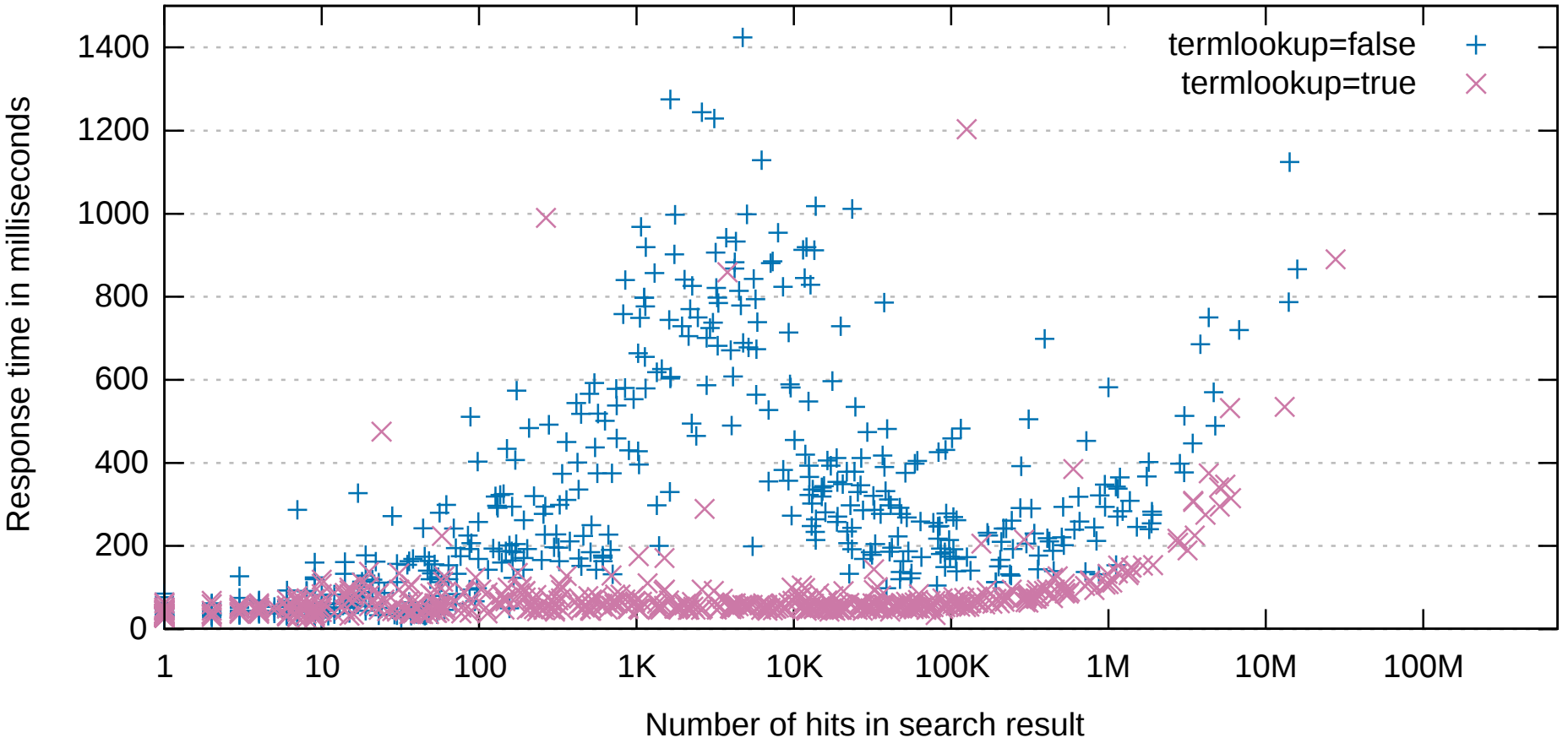
```
for term: query.getTerms()
  result.add(term, counter.get(getOrdinal(term)))
```

Stripped

```
counter = pool.getCounter(key)

for term: query.getTerms()
  result.add(term, counter.get(getOrdinal(term)))
```

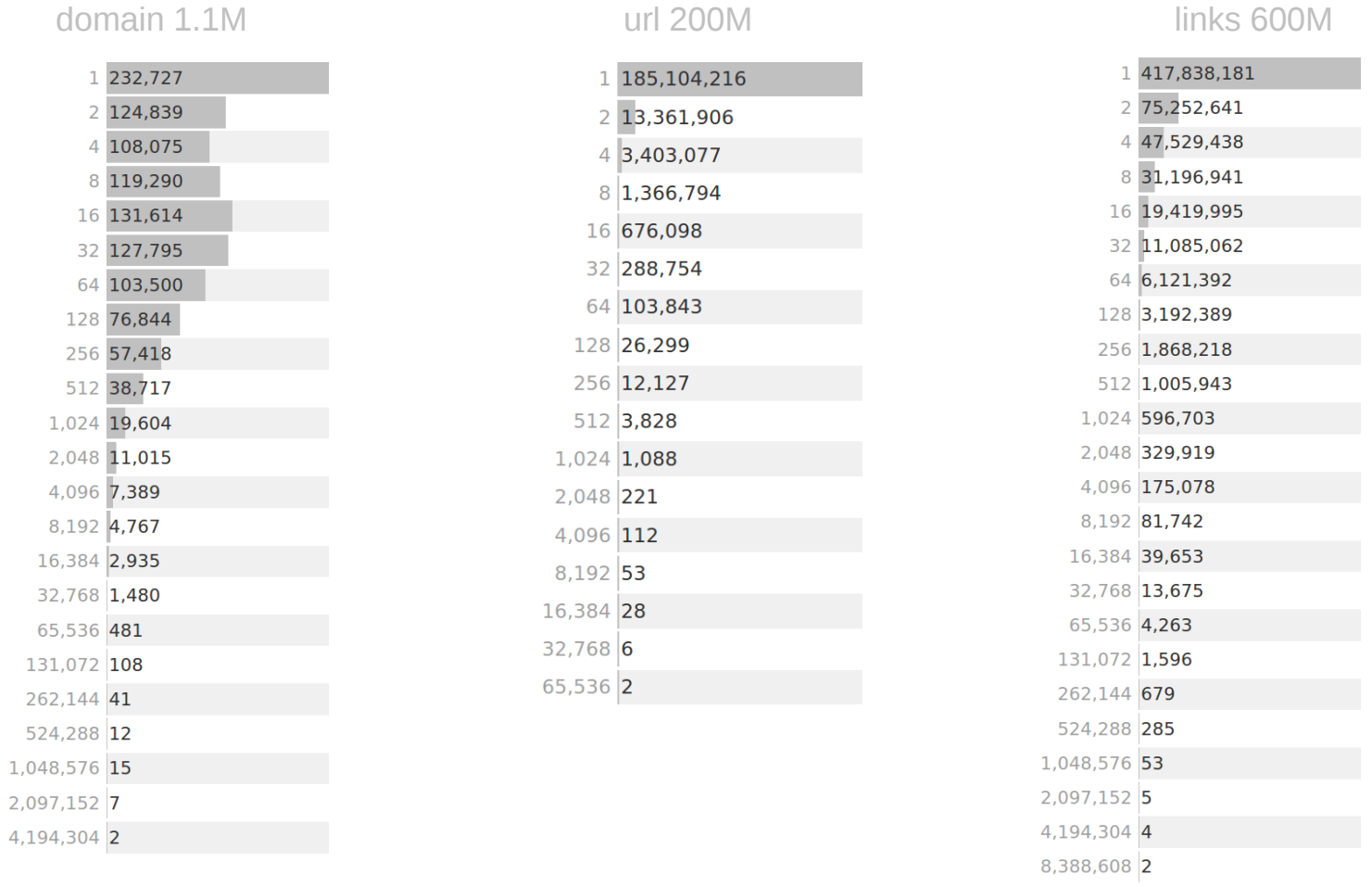
Plain of Platitude™

9 shards / 7TB / 2.3G docs, 256GB RAM, 16 cores, facet 1.1M values/shard

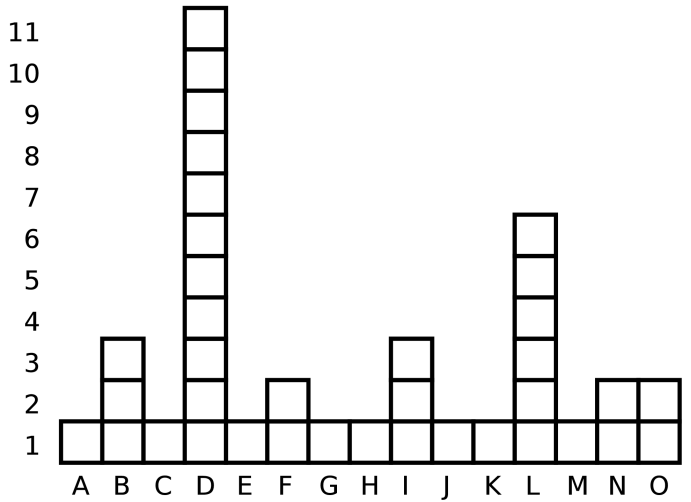


250,000,000 docs / 900GB, optimized			
Field	References	Max docs/term	Terms
domain	250,000,000	3,000,000	1,100,000
url	250,000,000	56,000	200,000,000
links	5,800,000,000	5,000,000	610,000,000

Term distributions

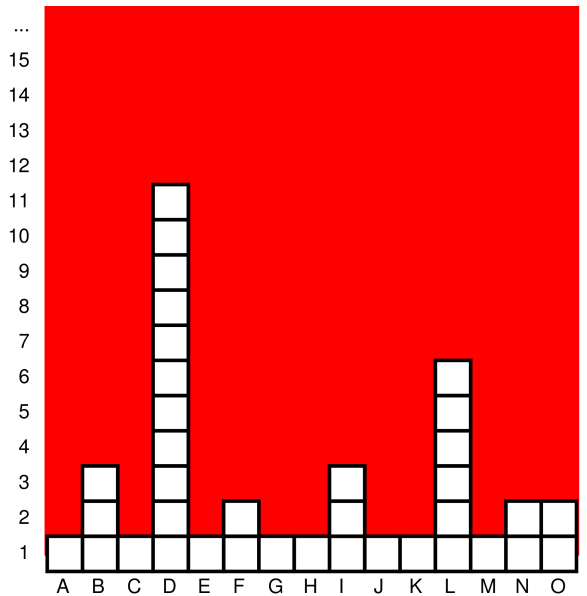


Clean



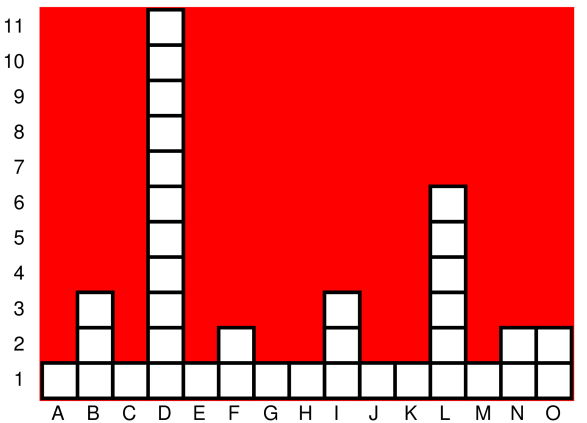
World Full Of Nothing

int[ordinals]




```
domain:    4 MB
url:     780 MB
links:  2350 MB
```

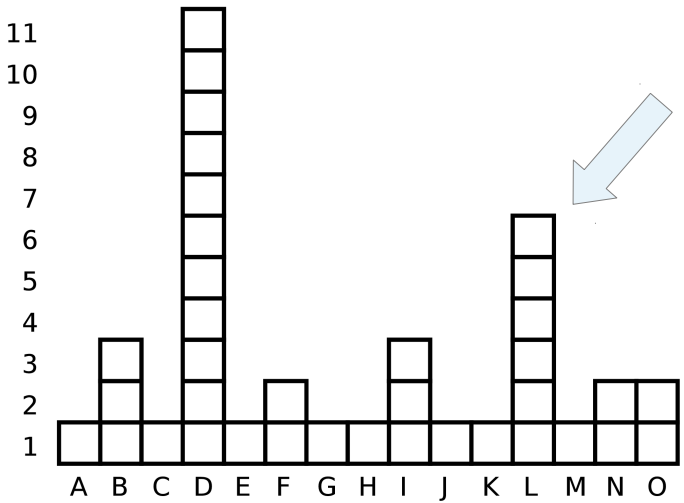
PackedInts(ordinals, maxBPV)



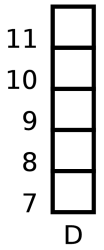

```
domain:    3 MB (72%)
url:     420 MB (53%)
links:  1760 MB (75%)
```

Construction Time Again

Platonic ideal

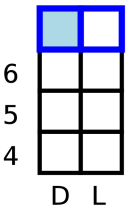


Plane 4

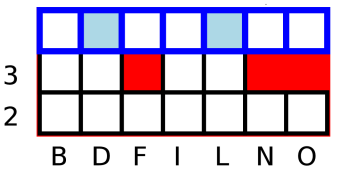


Harsh reality

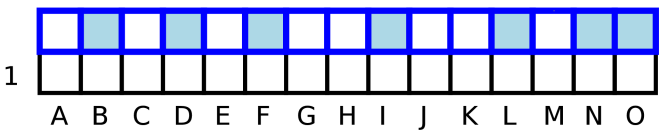
Plane 3



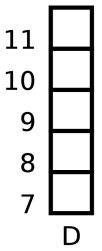
Plane 2



Plane 1

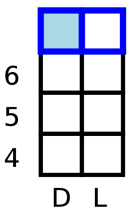


Plane 4

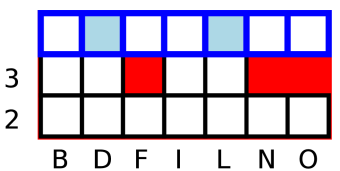


L: 0 ≡ 000000

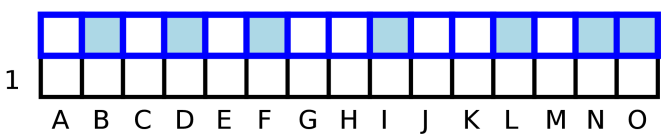
Plane 3



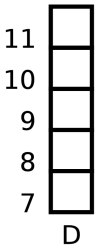
Plane 2



Plane 1

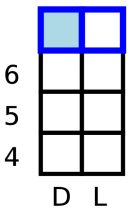


Plane 4

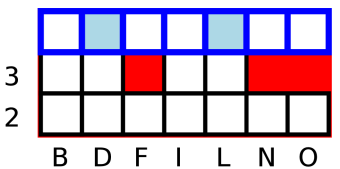



```
L:  0 ≡ 000000
L:  1 ≡ 000001
```

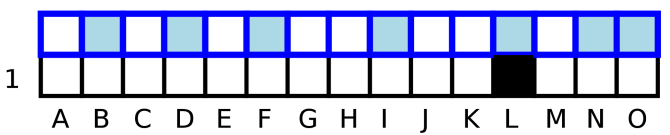
Plane 3



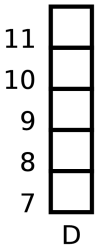
Plane 2



Plane 1

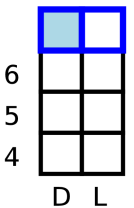


Plane 4

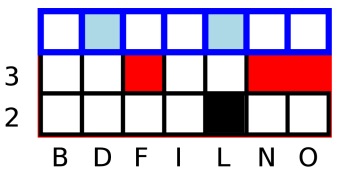



```
L:  0 ≡ 000000
L:  1 ≡ 000001
L:  2 ≡ 000010
```

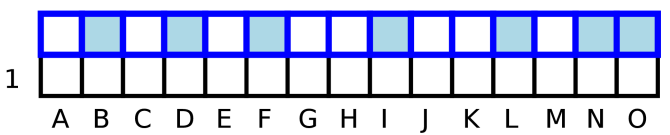
Plane 3



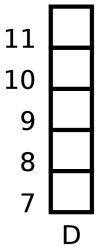
Plane 2



Plane 1

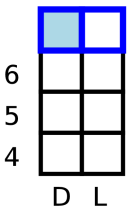


Plane 4

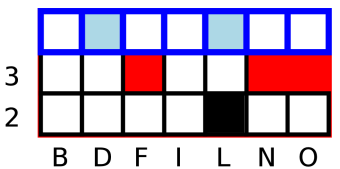



```
L:  0 ≡ 000000
L:  1 ≡ 000001
L:  2 ≡ 000010
L:  3 ≡ 000011
```

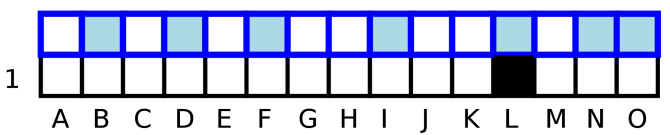
Plane 3



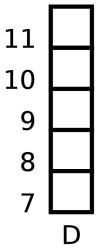
Plane 2



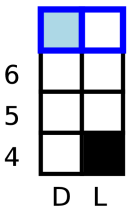
Plane 1



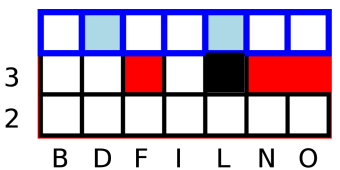
Plane 4



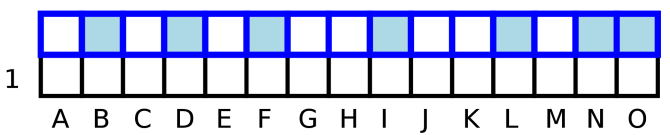
Plane 3



Plane 2



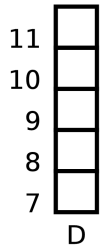
Plane 1



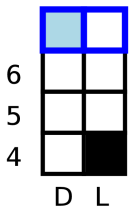

```
L:  0 ≣ 000000
L:  1 ≣ 000001
L:  2 ≣ 000010
L:  3 ≣ 000011
L:  4 ≣ 000100
L:  5 ≣ 000101
L:  6 ≣ 000110
L:  7 ≣ 000111
...
L: 12 ≣ 001100
```

```
if counter[ordinal]++ == 0 && tracked < maxTracked
  tracker[tracked++] = ordinal
```

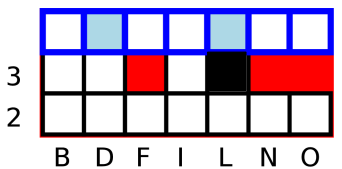
Plane 4



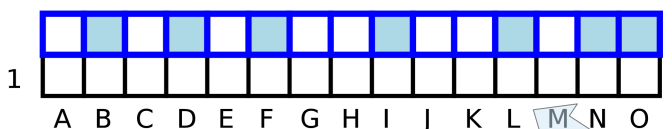
Plane 3



Plane 2



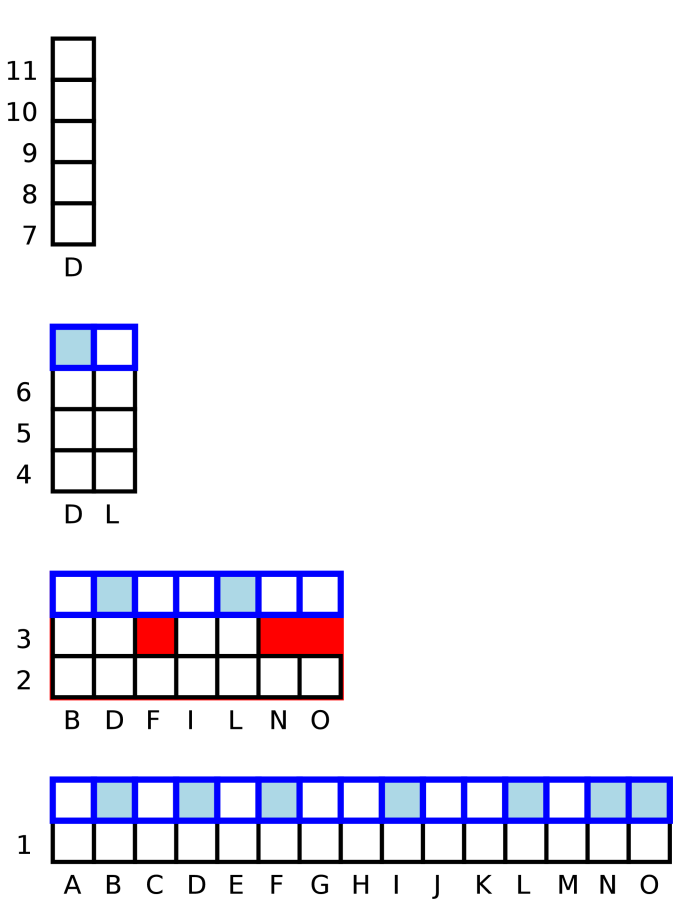
Plane 1



?

```
L:  0 ≡ 000000
L:  1 ≡ 000001
L:  2 ≡ 000010
L:  3 ≡ 000011
L:  4 ≡ 000100
L:  5 ≡ 000101
L:  6 ≡ 000110
L:  7 ≡ 000111
...
L: 12 ≡ 001100
```

Now This is Fun

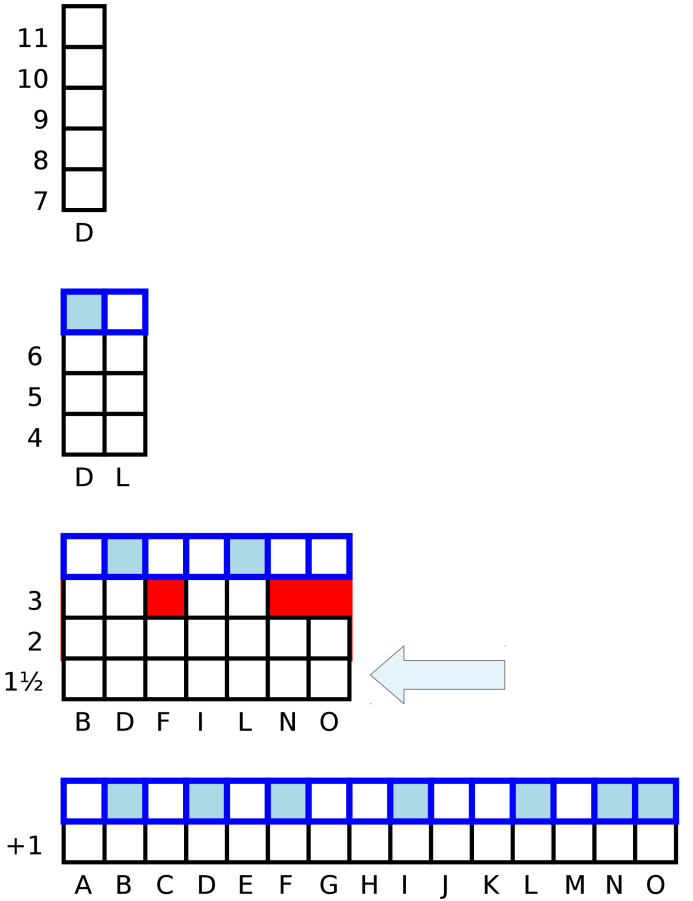


Plane 4

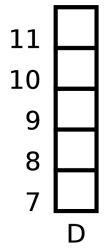
Plane 3

Plane 2

Plane 1

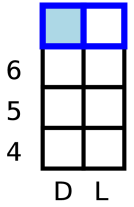


Plane 4

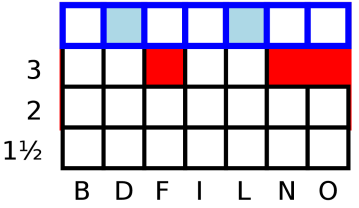


L: 0 ≡ 000000

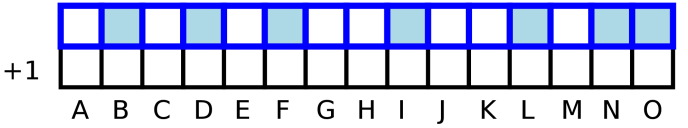
Plane 3



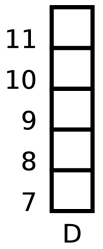
Plane 2



Plane 1

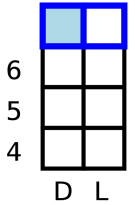


Plane 4

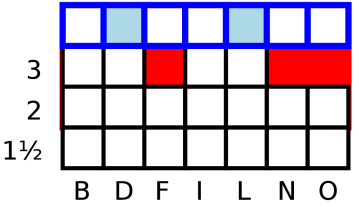


L: 0 ≡ 000000
L: 1 ≡ 000001

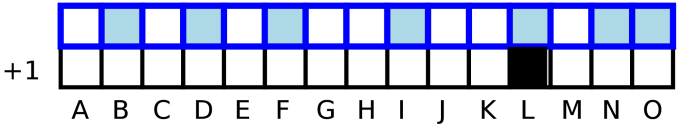
Plane 3



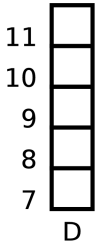
Plane 2



Plane 1

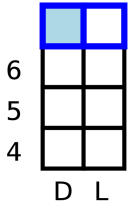


Plane 4

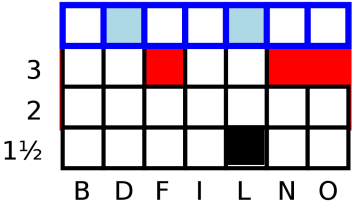



```
L:  0 ≣ 000000
L:  1 ≣ 000001
L:  2 ≣ 000011
```

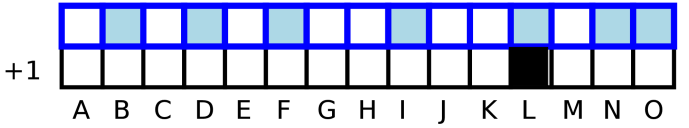
Plane 3



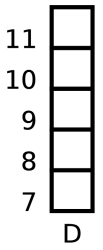
Plane 2



Plane 1

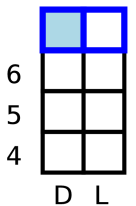


Plane 4

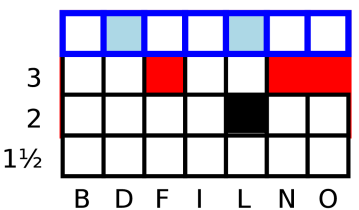



```
L:  0 ≣ 000000
L:  1 ≣ 000001
L:  2 ≣ 000011
L:  3 ≣ 000101
```

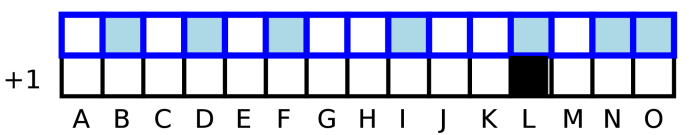
Plane 3



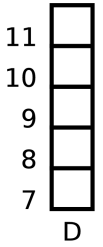
Plane 2



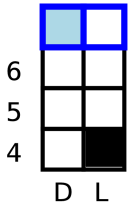
Plane 1



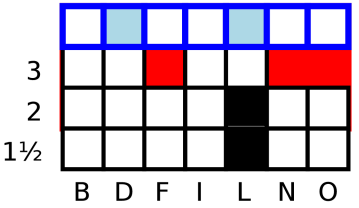
Plane 4



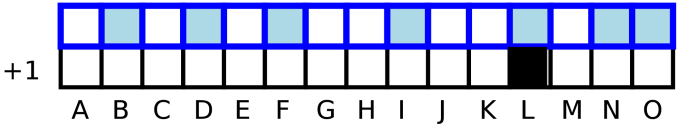
Plane 3



Plane 2



Plane 1

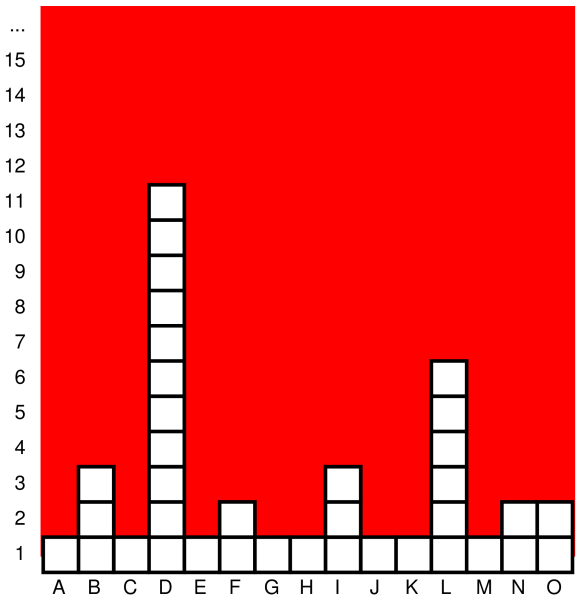



```
L:  0 ≣ 000000
L:  1 ≣ 000001
L:  2 ≣ 000011
L:  3 ≣ 000101
L:  4 ≣ 000111
L:  5 ≣ 001001
L:  6 ≣ 001011
L:  7 ≣ 001101
...
L: 12 ≣ 010111
```

The Bottom Line

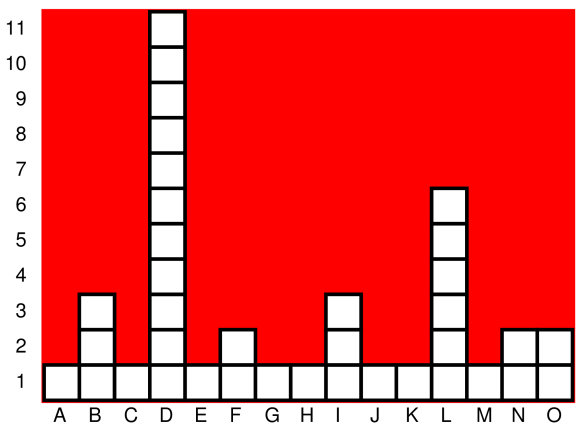
```
domain:     4 MB
url:      780 MB
links:   2350 MB
```

int[ordinals]



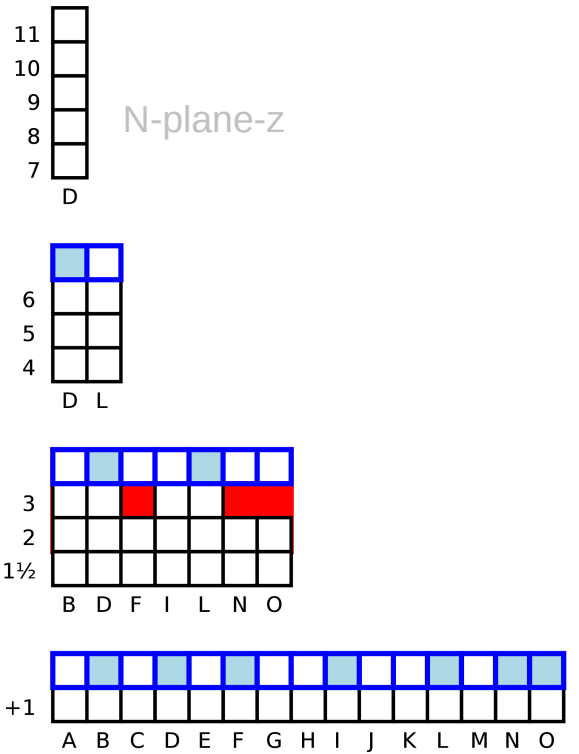

```
domain:     3 MB (72%)
url:      420 MB (53%)
links:   1760 MB (75%)
```

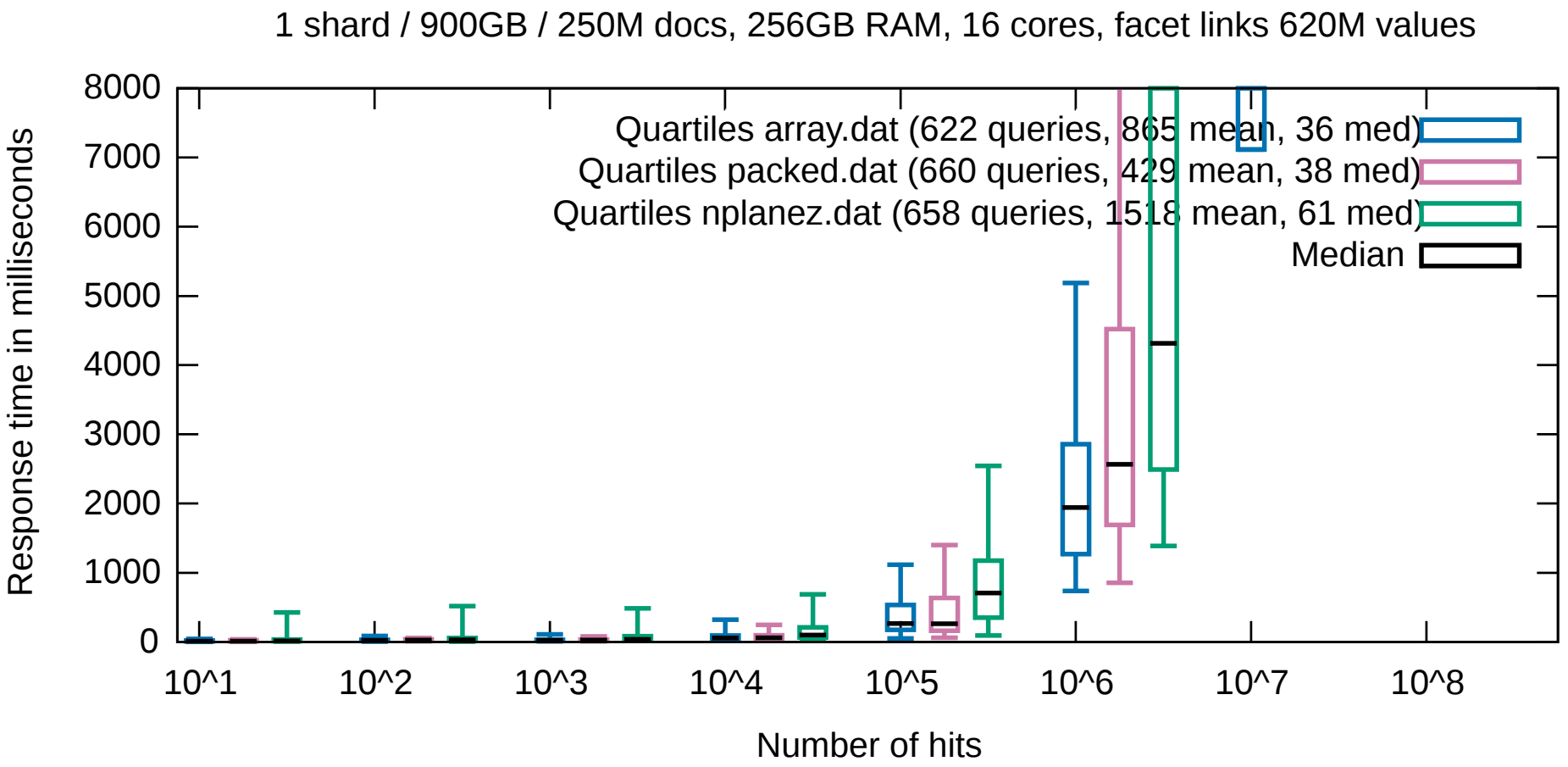
PackedInts(ordinals, maxBPV)




```
domain:     1 MB (30%)
url:       66 MB ( 8%)
links:    311 MB (13%)
```

N-plane-z



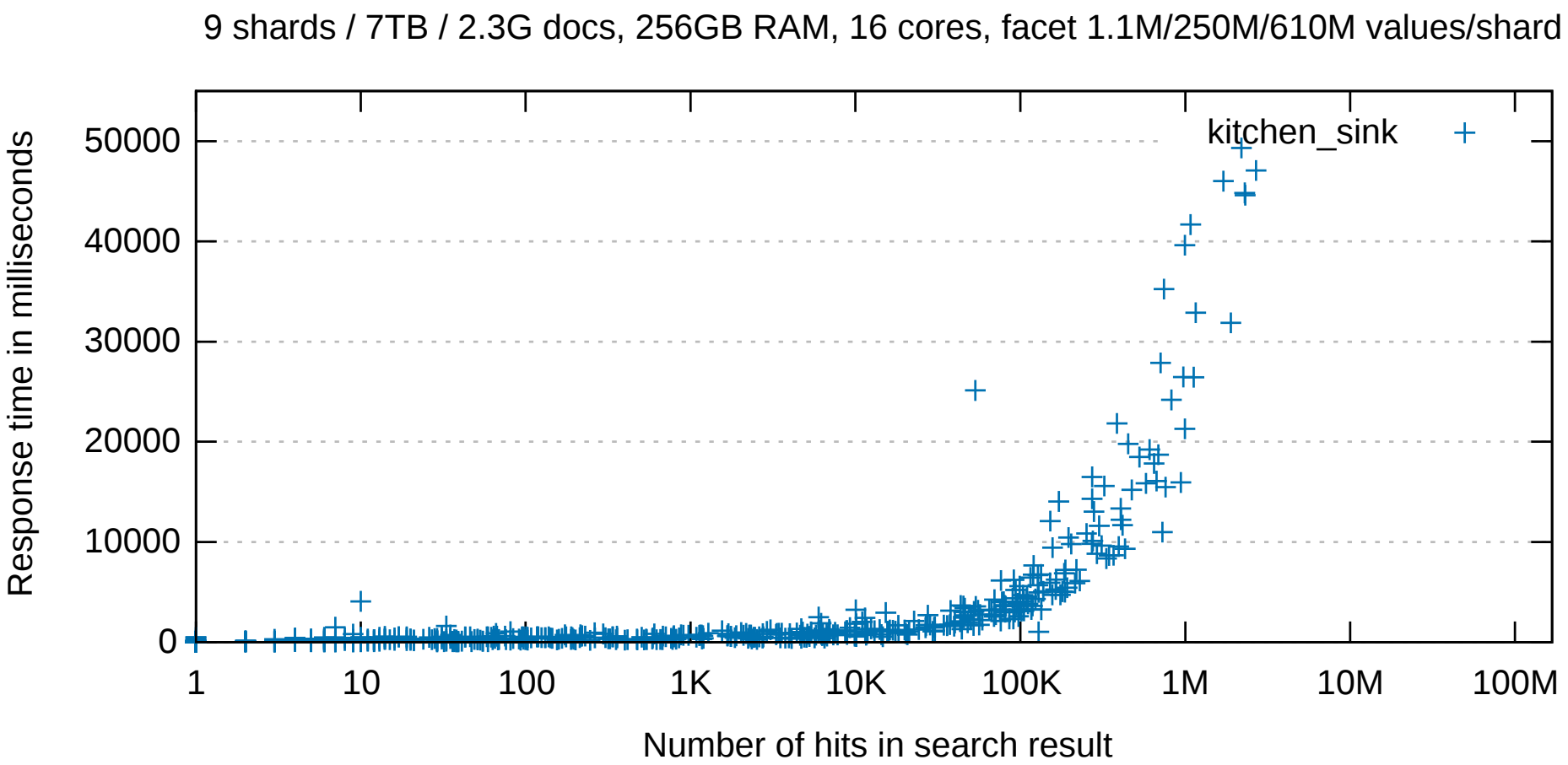

1 shard / 900GB / 250M docs, 256GB RAM, 16 cores, facet links 620M values
Response time in milliseconds
8000
7000
6000
5000
4000
3000
2000
1000
0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
Number of hits
Quartiles array.dat (622 queries, 865 mean, 36 med)
Quartiles packed.dat (660 queries, 429 mean, 38 med)
Quartiles nplanez.dat (658 queries, 1518 mean, 61 med)
Median

Kitchen sink

250,000,000 docs / 900GB, optimized			
Field	References	Max docs/term	Terms
domain	250,000,000	3,000,000	1,100,000
url	250,000,000	56,000	200,000,000
links	5,800,000,000	5,000,000	610,000,000

x 9 shards
x 3 concurrent requests

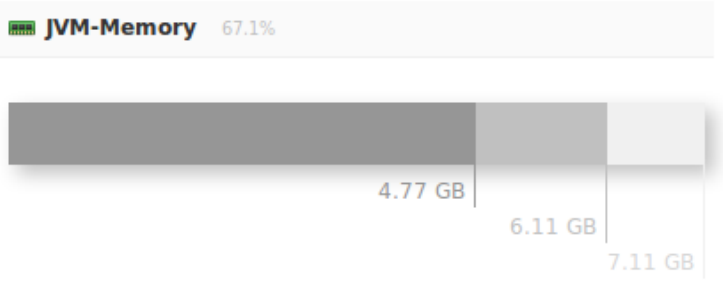
Shouldn't Have Done That



Some Great Reward

JVM

Runtime	Oracle Corporation Java HotSpot(TM) 64-Bit Server VM (1.7.0_76 24.7...
Processors	32
Args	-Djava.io.tmpdir=/home/summanet/tomcat-solr8a/temp
	-Dcatalina.home=/home/summanet/tomcat-solr-master
	-Dcatalina.base=/home/summanet/tomcat-solr8a
	-Djava.endorsed.dirs=/home/summanet/tomcat-solr-master/endorsed
	-Dlog4j.configuration=file:///home/summanet/services/conf/tomcat-so...
	-Dsb.tomcat.maxthreads=200
	-DzkHost=localhost:52001,localhost:52002,localhost:52003
	-Dsb.solr.httpport=52308
	-Dsb.solr.shutdownport=52508
	-Dsb.solr.instanceid=8a
	-Xmx8192m
	-Djava.security.egd=file:/dev/./urandom
	-Dsun.net.inetaddr.ttl=300
	-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
	-Djava.util.logging.config.file=/home/summanet/tomcat-solr8a/conf/lo...



8GB heap per 900GB shard

Dream On

- Threaded counting
- Monotonically increasing tracker for nplane-z
- Regexp filtering
- Fine count skipping
- Counter capping

Nothing's Impossible