Thinking about performance

Search: a case study

Perf: speed/power/etc.

Perf: why do we care?

“Premature optimization is the root of all evil”

“We should forget about small efficiencies, say about 97% of the time”

Different designs:
100x - 1000x perf difference

Working code attracts people who want to code. Design documents attract people who want to talk.

RETWEETS
538

LIKES
561

“Coding feels like real work”

Whiteboard: 1h/iteration
Implementation: 2yr/iteration

Scale

(precursor to perf discussion)

10k; 10M; 10G

(5kB per doc)

What’s the actual problem?

AND queries

10k; 10M; 10G

(5kB per doc)

One person’s email
One forum

10k

5kB * 10k = 50MB

10k

50MB is small!

10k

$50 phone => 1GB RAM

Naive algorithm

```
for loop over all documents {
  for loop over terms in document {
    // matching logic here.
  }
}
```

10k

10k; **10M**; 10G

(5kB per doc)

~Wikipedia sized

10M

5kB * 10M = 50GB

10M

$2000 for 128GB server

(Broadwell single socket Xeon-D)

10M

25 GB/s memory bandwidth

10M

50GB / 25 GB/s = 2s

(½ query per sec (QPS))

10M

Is 2s latency ok?

10M

Is 1/2 QPS ok?

10M

Larger service

Latency == $$$

10M

Latency == $$$

http://assets.en.oreilly.com/1/event/29/Keynote%20Presentation%202.pdf

http://www.bizreport.com/2016/08/mobify-report-reveals-impact-of-mobile-website-speed.html

http://assets.en.oreilly.com/1/event/29/The%20User%20and%20Business%20Impact%20of%20Server%20Delays,%20Additional%20Bytes,%20and%20HTTP%20Chunking%20in%20Web%20Search%20Presentation.pptx

10M

http://assets.en.oreilly.com/1/event/27/Varnish%20-%20A%20State%20of%20the%20Art%20High-Performance%20Reverse%20Proxy%20Presentation.pdf

Google: 400ms extra latency

0.44% decrease in searches per user

10M

Google: 400ms extra latency

0.44% decrease in searches per user

0.76% after six weeks

10M

Google: 400ms extra latency

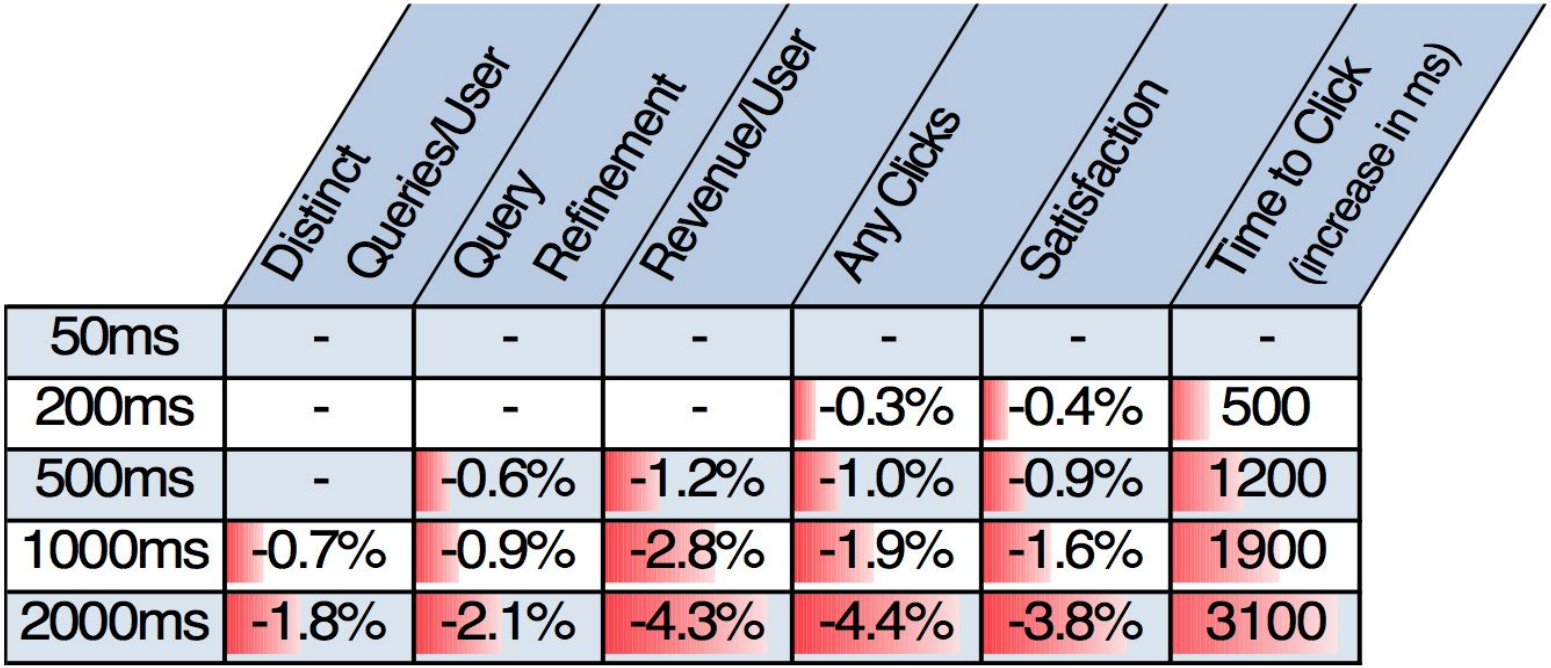
0.44% decrease in searches per user

0.76% after six weeks

0.21% decrease after delay removed

10M

Bing



	Distinct Queries/User	Query Refinement	Revenue/User	Any Clicks	Satisfaction	Time to Click (increase in ms)
50ms	-	-	-	-	-	-
200ms	-	-	-	-0.3%	-0.4%	500
500ms	-	-0.6%	-1.2%	-1.0%	-0.9%	1200
1000ms	-0.7%	-0.9%	-2.8%	-1.9%	-1.6%	1900
2000ms	-1.8%	-2.1%	-4.3%	-4.4%	-3.8%	3100

- Means no statistically significant change

10M

Mobify

100ms home load => 1.11% delta in conversions

10M

Mobify

100ms home load => 1.11% delta in conversions

100ms checkout page speed => 1.55% delta in conversions

10M

All Images News Maps Videos More ▾ Search tools

About 71,100 results (0.59 seconds)

10M

To hit 500ms round trip...

...budget ~10ms for search

10M

Larger service

Latency == $$$

Need to handle more than ½ QPS

10M

Use an index?

Salton; The SMART Retrieval System (1971); work originally done in early 60s

10M

30 - 30,000 QPS

(we'll talk about figuring this out later)

http://www.anandtech.com/show/9185/intel-xeon-d-review-performance-per-watt-server-soc-champion/14

Haque et al.; Few-to-Many: Incremental Parallelism for Reducing Tail Latency in Interactive Services (ASPLOS, 2015)

10M

10k; 10M; **10G;**

(5kB per doc)

10B

5kB * 10G = 50TB

Horizontal scaling

(use more machines)

Easy to scale

(different docuemnts on different machines)

Horizontal scaling

10G docs / (10M docs / machine) = 1k machines

Redmond-Dresden: 150ms

Horizontal scaling

10G docs / (10M docs / machine) = 1k machines

1k machines * 10 clusters = 10k machines

“[With 1800 machines, in one year], it’s typical that 1,000 individual machine failures will occur; thousands of hard drive failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; 5 racks will “go wonky,” with half their network packets missing in action; and the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span”

Horizontal scaling

10G docs / (10M docs / machine) = 1k machines

1k machines * 10 clusters = 10k machines

10k machines * 3 redundancy = 30k machines

Horizontal scaling

10G docs / (10M docs / machine) = 1k machines

1k machines * 10 clusters = 10k machines

10k machines * 3 redundancy = 30k machines

30k machines * $1k/yr/machine = $30M / yr

2x perf: $15m/yr

2% perf: $600k/yr

Horizontal scaling

10G docs / (10M docs / machine) = 1k machines

1k machines * 10 clusters = 10k machines

10k machines * 3 redundancy = 30k machines

30k machines * $1k/yr/machine = $30M / yr

Machine time vs. dev time

10B

Search Algorithms

What’s the problem again?

Posting list

INDEX RERVM PRÆCIPVARVM HVIVS ATLANTIS.

See http://nlp.stanford.edu/IR-book/ for implementation details

HashMap[term] => list[docs]

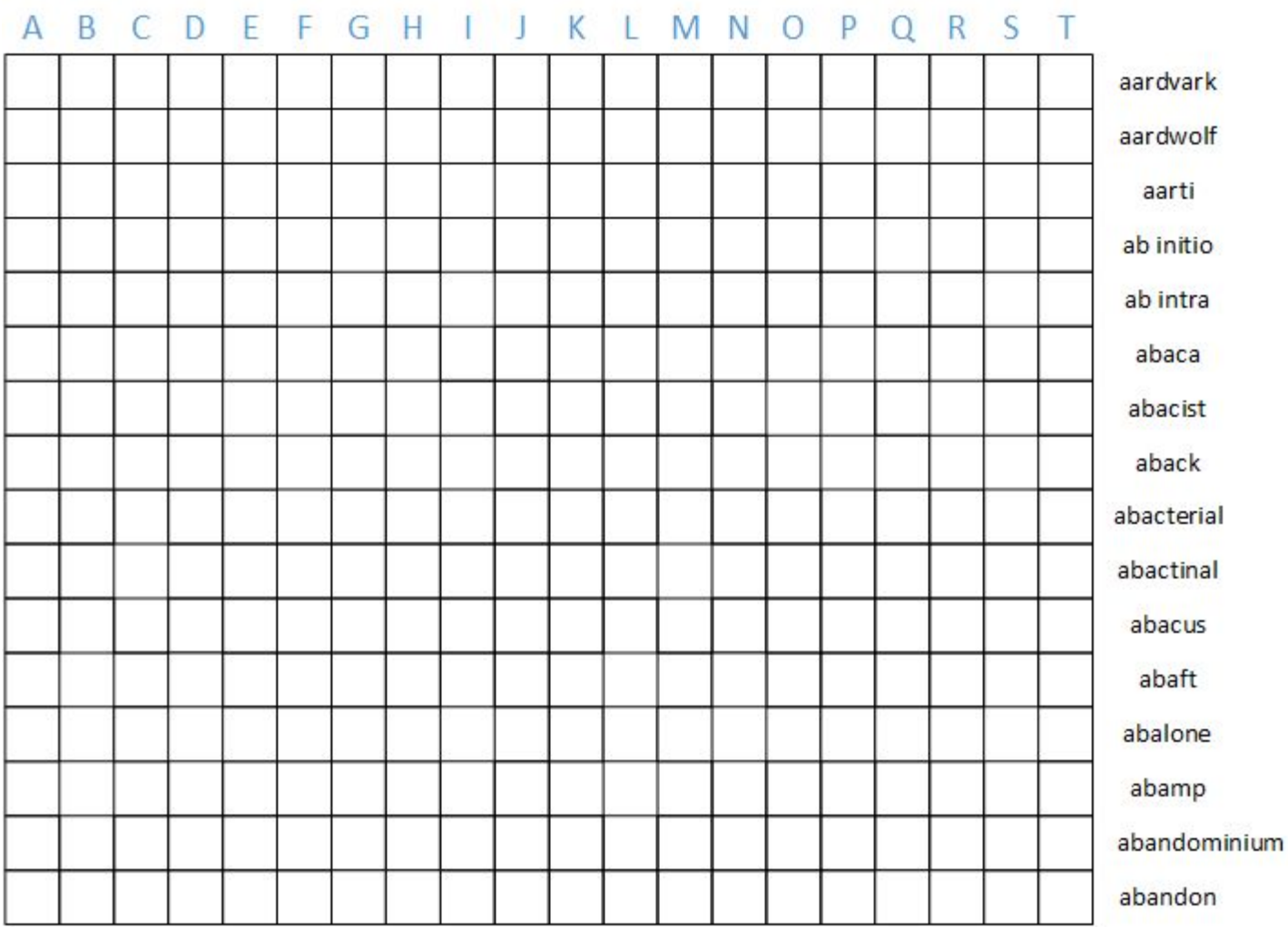
Bloom filter

Algorithms: bloom filter

BitFunnel

Algorithms: bloom filter

What about an array?


A B C D E F G H I J K L M N O P Q R S T
aardvark
aardwolf
aarti
ab initio
ab intra
abaca
abacist
aback
abacterial
abactinal
abacus
abaft
abalone
abamp
abandominium
abandon

How many terms?

Crista Lopes
@cristalopes

Follow

The largest C++ file we found in GitHub has 528Mb, 57 lines of code. Contains the first 50,847,534 primes, all hard coded in an array.

RETWEETS 1,491 LIKES 1,882

4:53 PM - 15 Sep 2016

1.5K 1.9K

Algorithms: bloom filter

One site has 37B primes

Algorithms: bloom filter

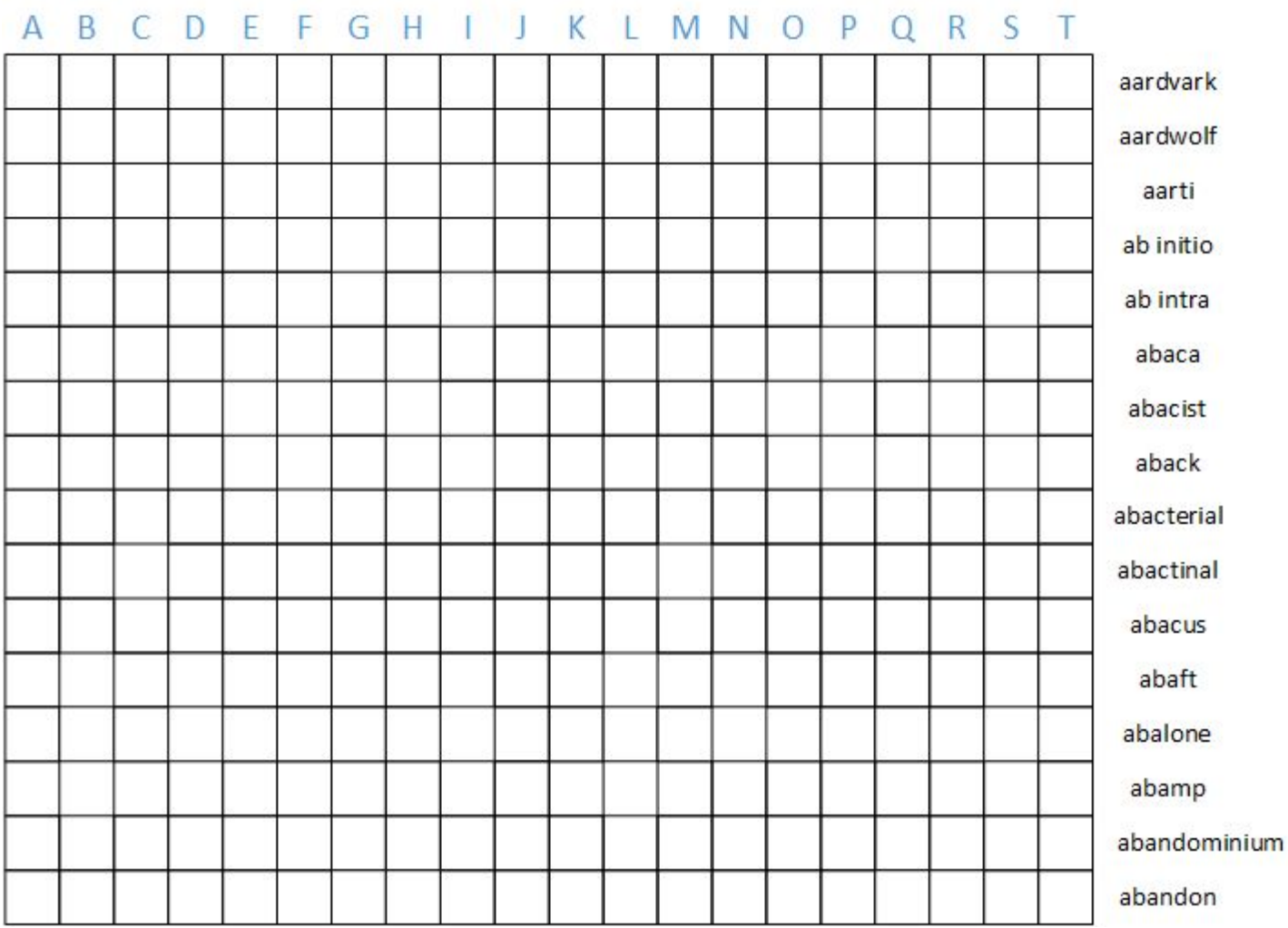
GUIDs, timestamps, DNA, etc.

Algorithms: bloom filter

Why index that stuff?

Algorithms: bloom filter

GTGACCTTGGGCAAGTTACTTA
ACCTCTCTGTGCCTCAGTTTCCT
CATCTGTAAAATGGGGATAATA

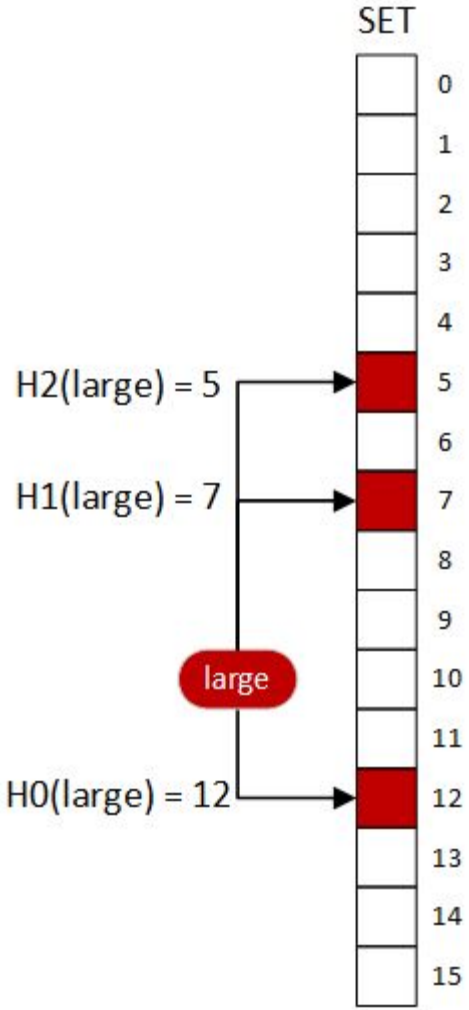

A B C D E F G H I J K L M N O P Q R S T
aardvark
aardwolf
aarti
ab initio
ab intra
abaca
abacist
aback
abacterial
abactinal
abacus
abaft
abalone
abamp
abandominium
abandon

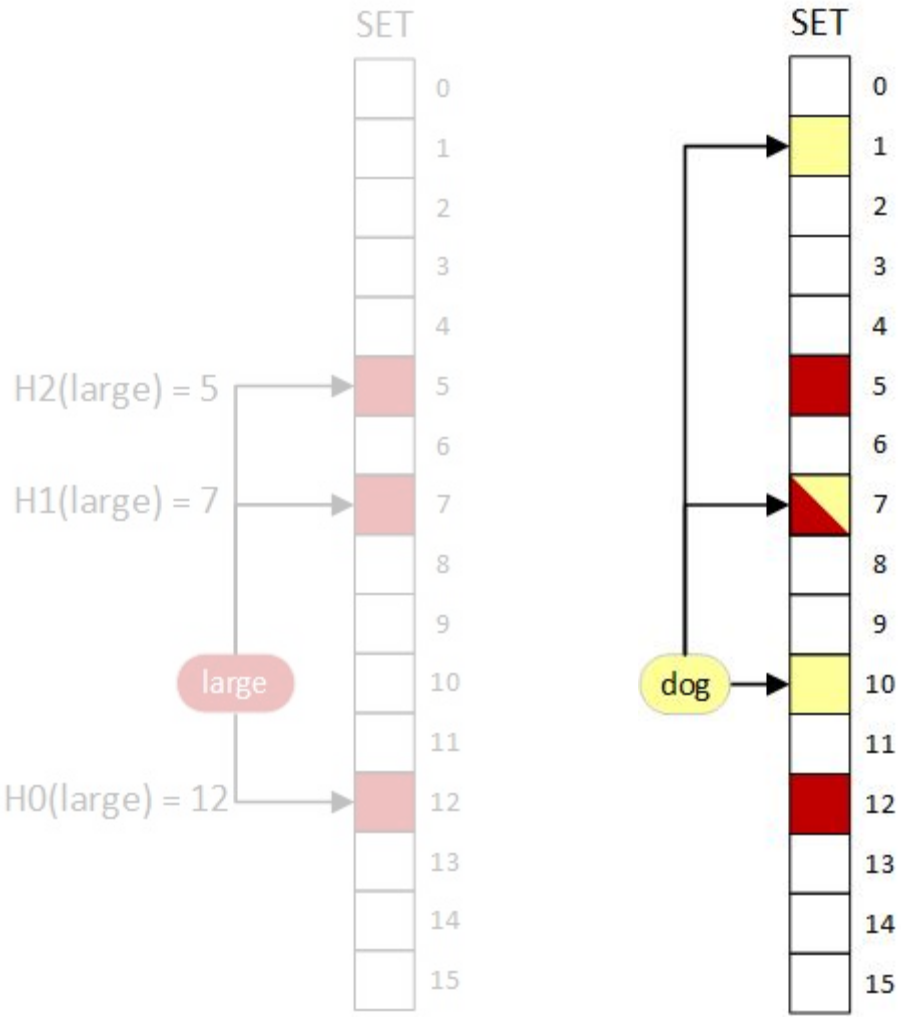
Most terms aren’t in most docs => use hashing

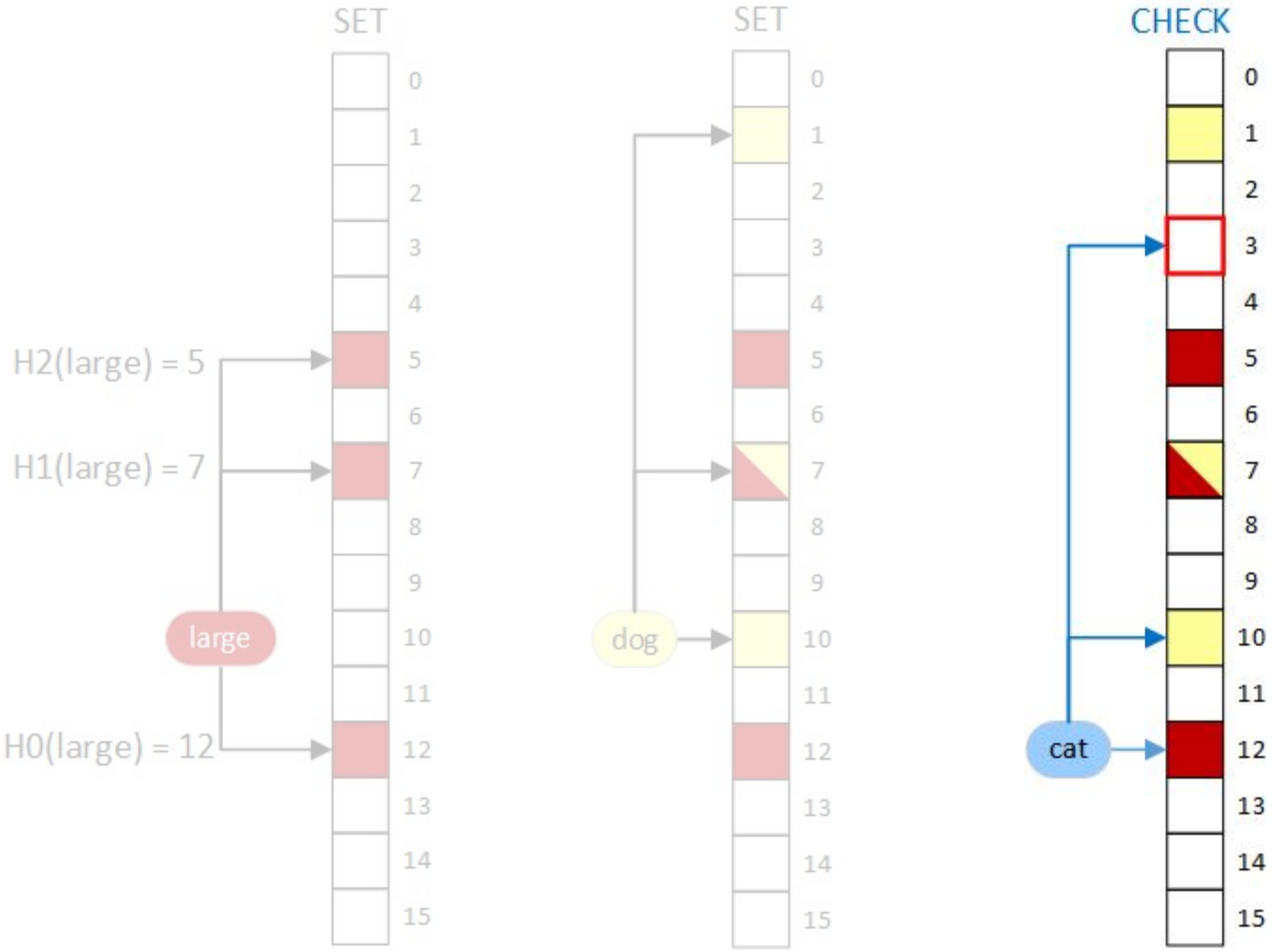
Algorithms: bloom filter

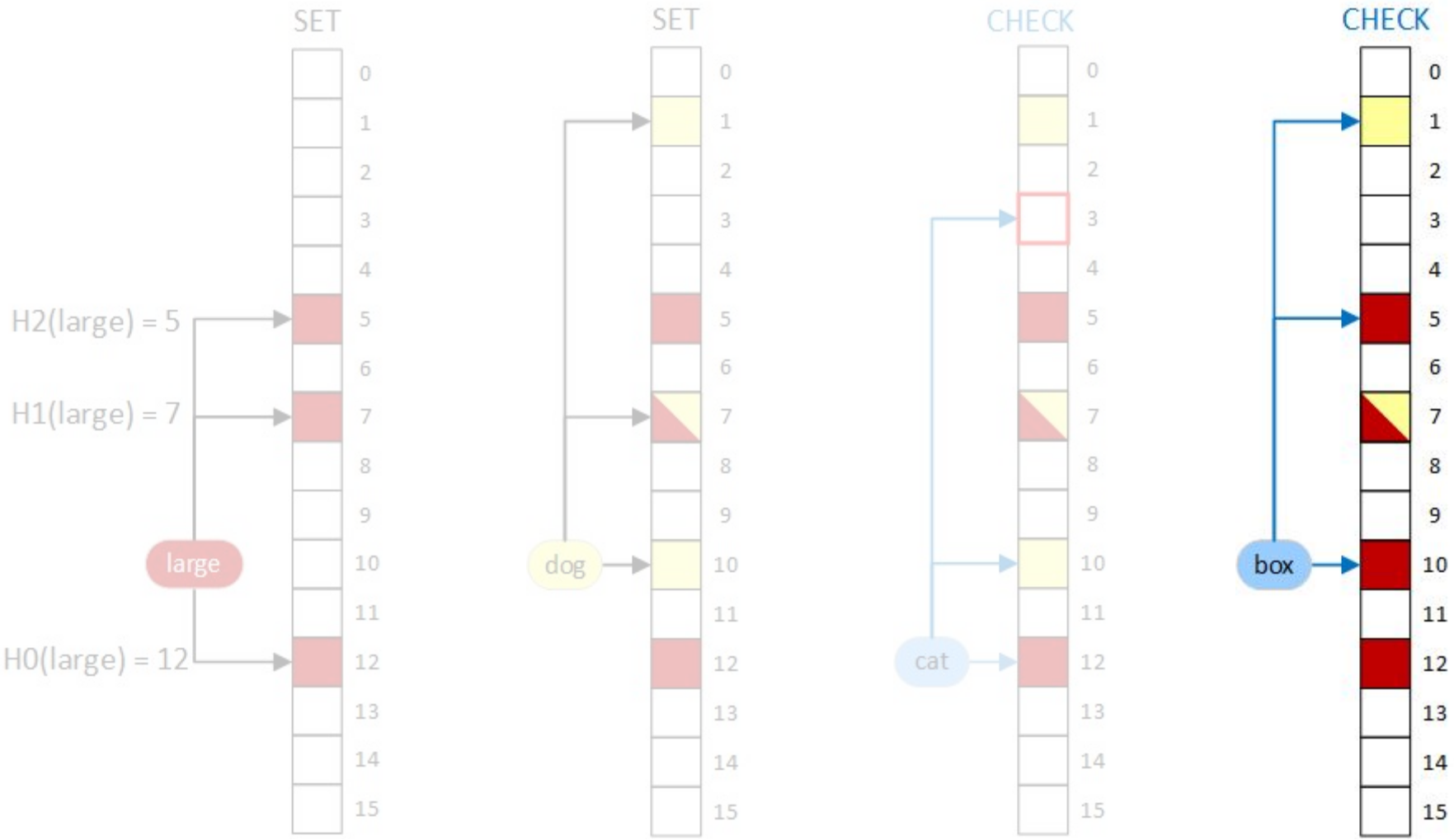
Bloom Filters

Algorithms: bloom filter


SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12


SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12
SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
dog


SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12
SET
dog
CHECK
cat


SET
H2(large) = 5
H1(large) = 7
large
H0(large) = 12
SET
dog
CHECK
cat
CHECK
box
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Probability of false positive?

(assume 10% bit density)

1 location: .1 = 10% false positive rate

(assume 10% bit density)

1 location: .1 = 10% false positive rate

2 locations: .1 * .1 = 1% false positive rate

(assume 10% bit density)

1 location: .1 = 10% false positive rate

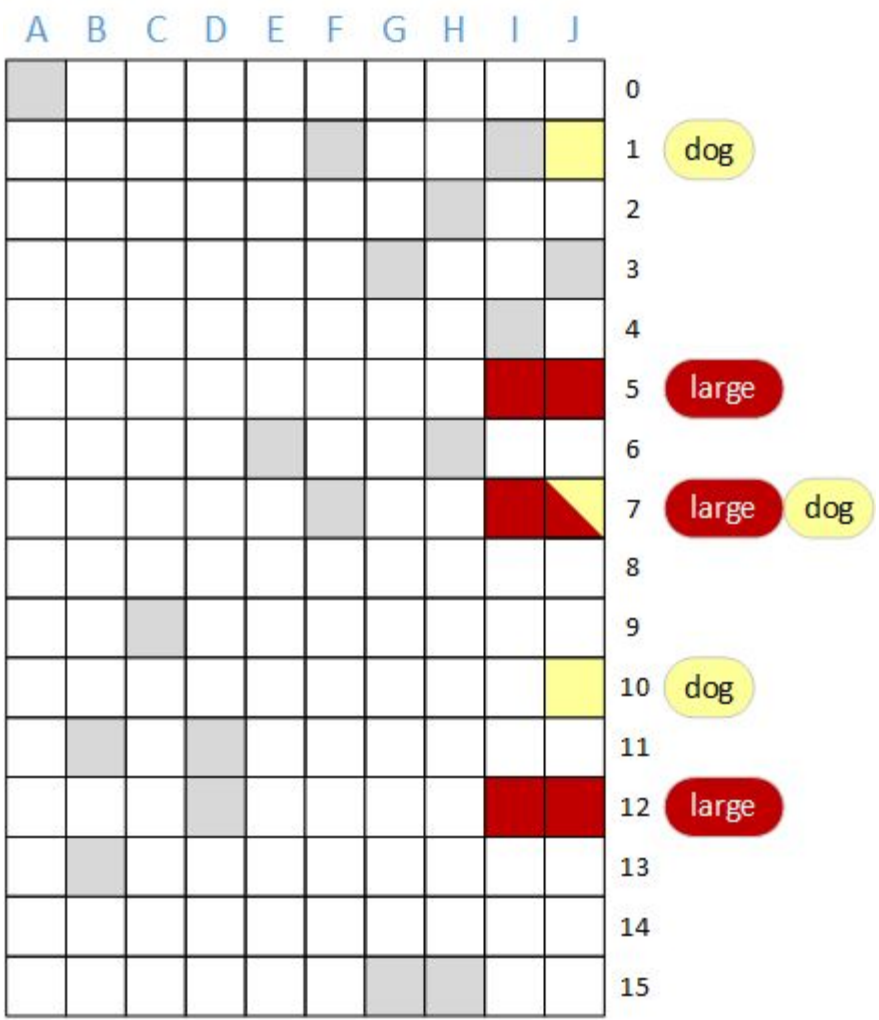
2 locations: .1 * .1 = 1% false positive rate

3 locations: .1 * .1 * .1 = 0.1% false positive rate

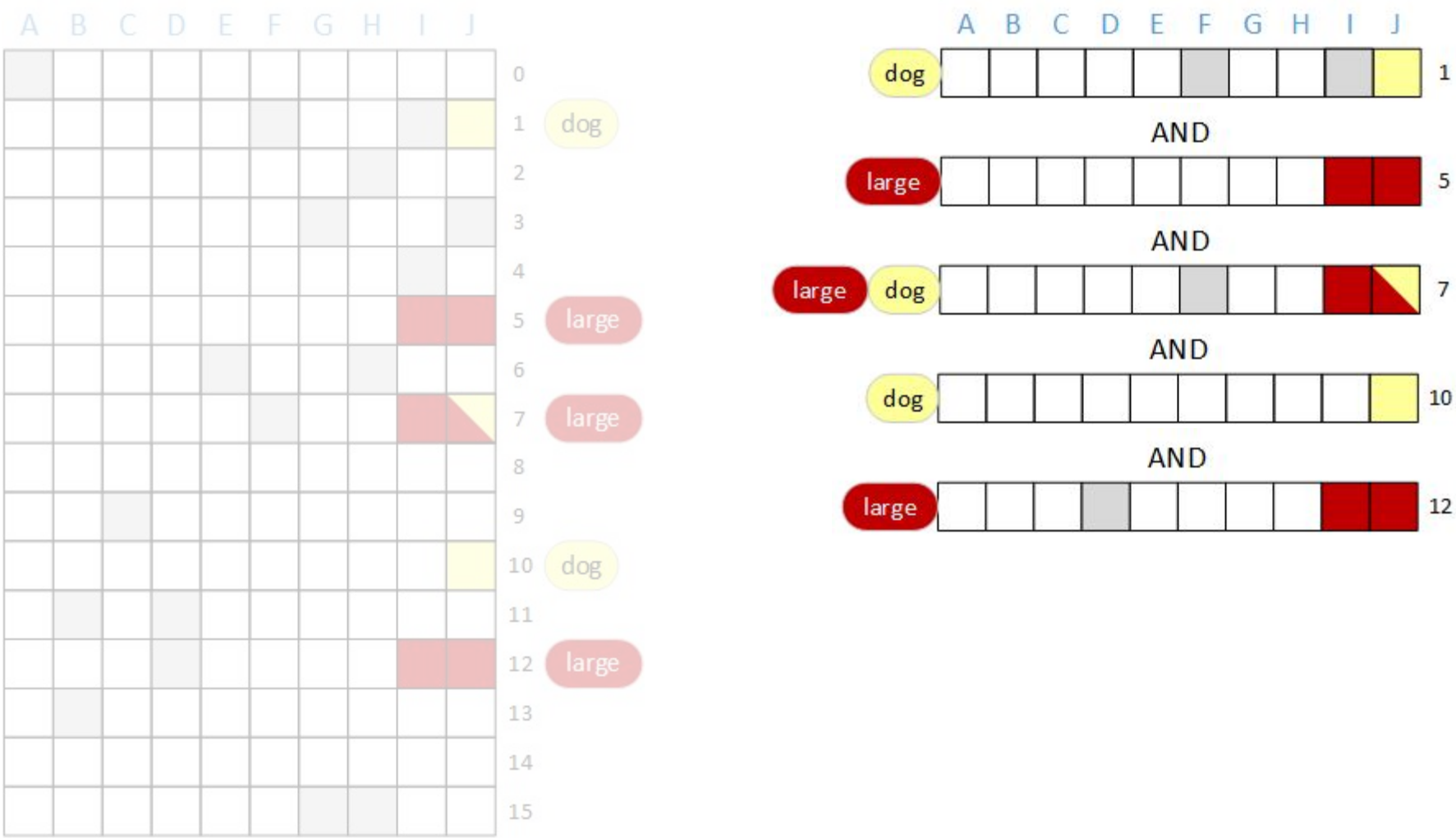
Linear cost
Exponential benefit

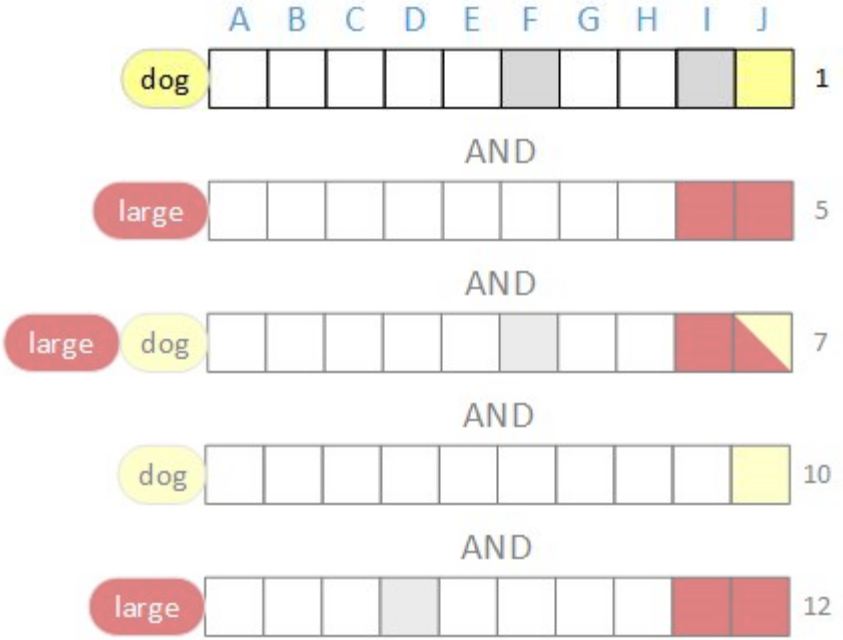
Algorithms: bloom filter

Multiple Documents Multiple Bloom Filters

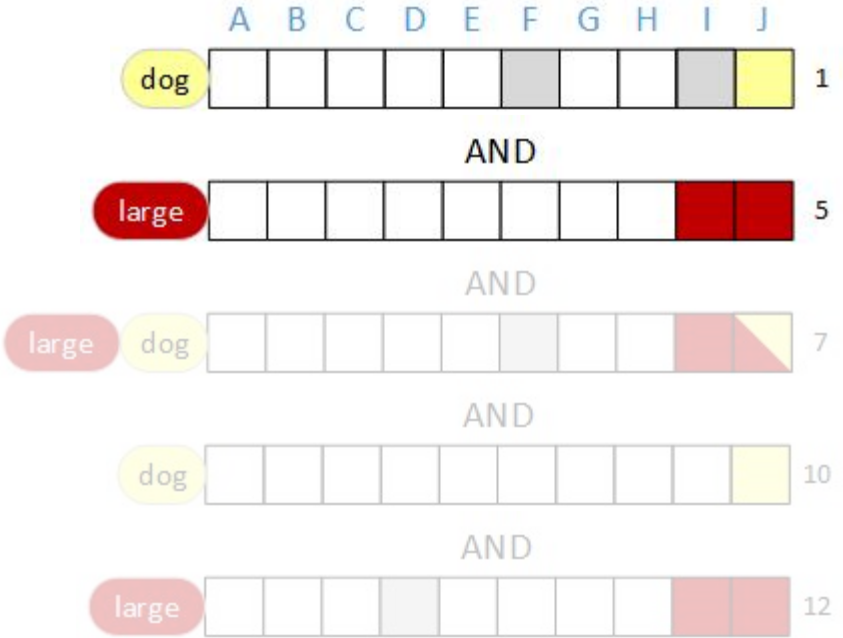

A B C D E F G H I J
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
dog
large
large
dog
dog
large

Do comparisons in parallel!

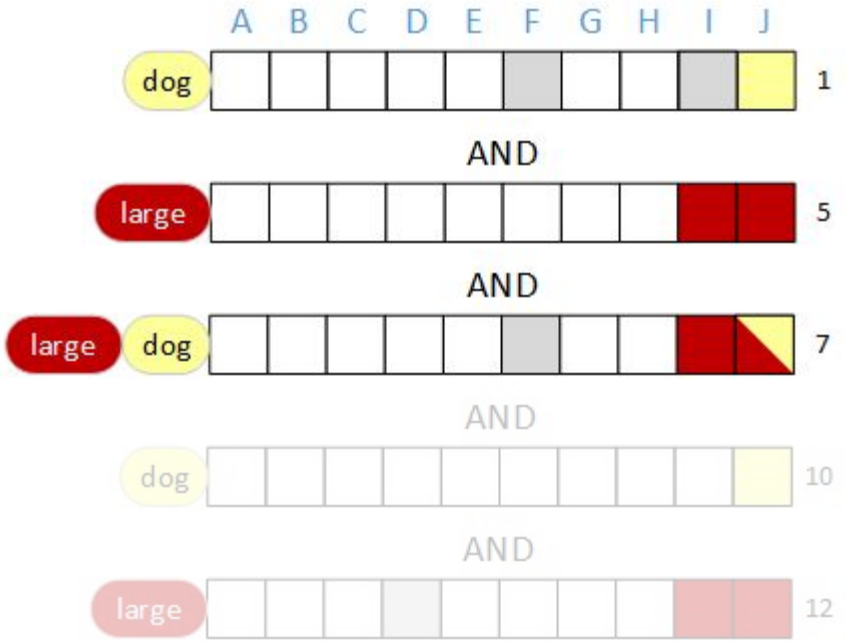

A B C D E F G H I J
0
1
dog
2
3
4
5
large
6
7
large
8
9
10
dog
11
12
large
13
14
15
A B C D E F G H I J
dog
1
AND
large
5
AND
large
dog
7
AND
dog
10
AND
large
12



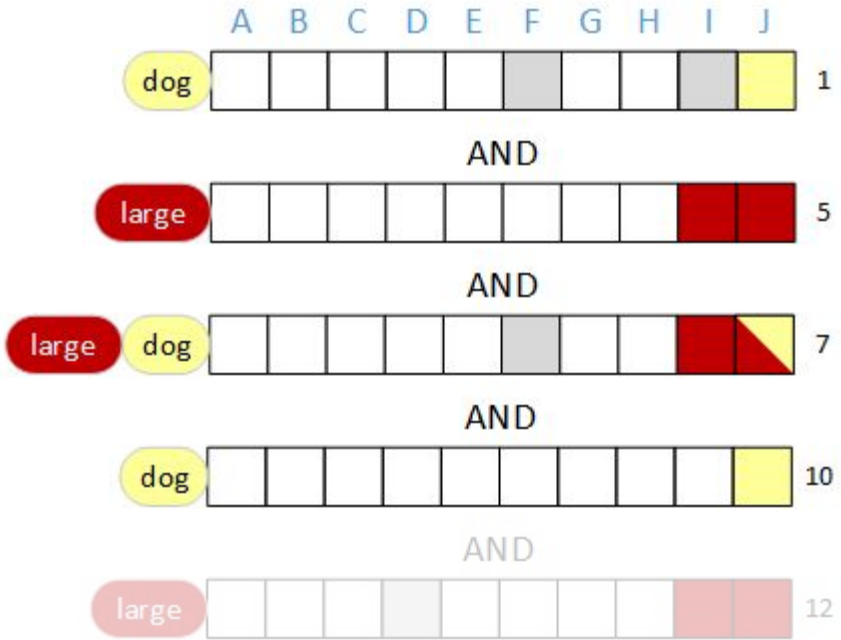
Algorithms: bloom filter



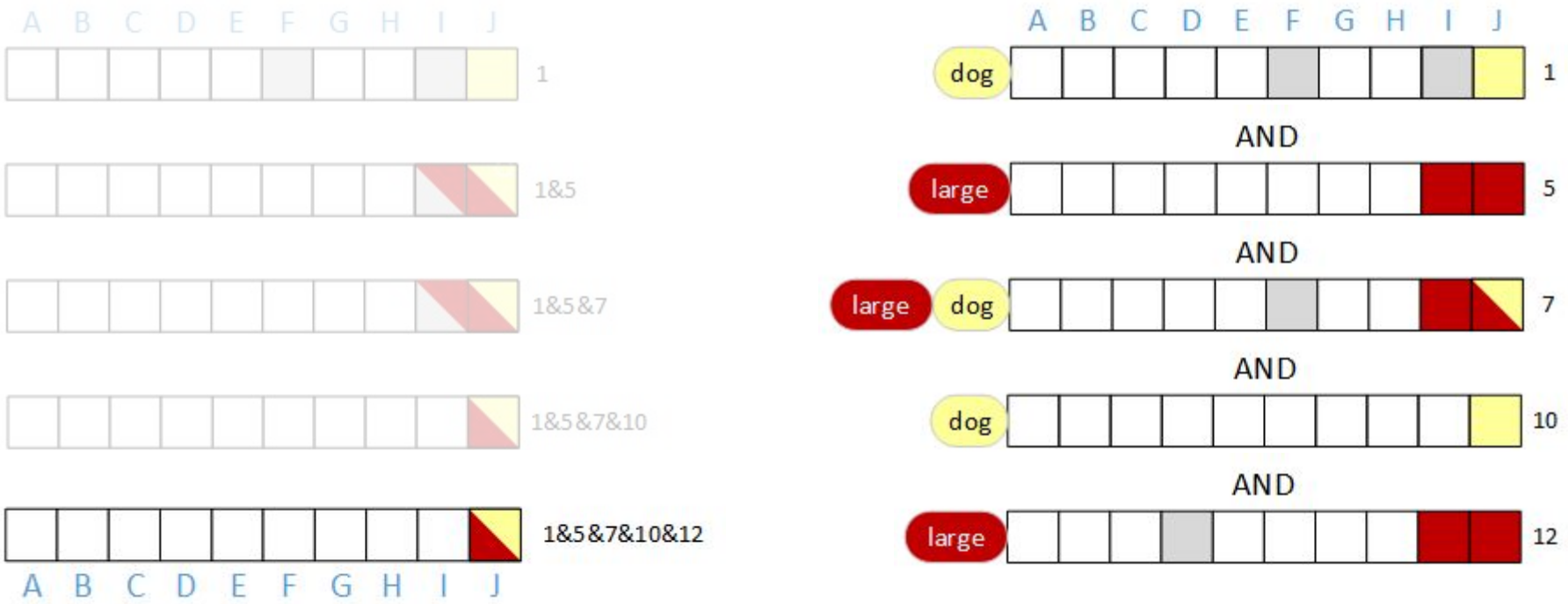
Algorithms: bloom filter



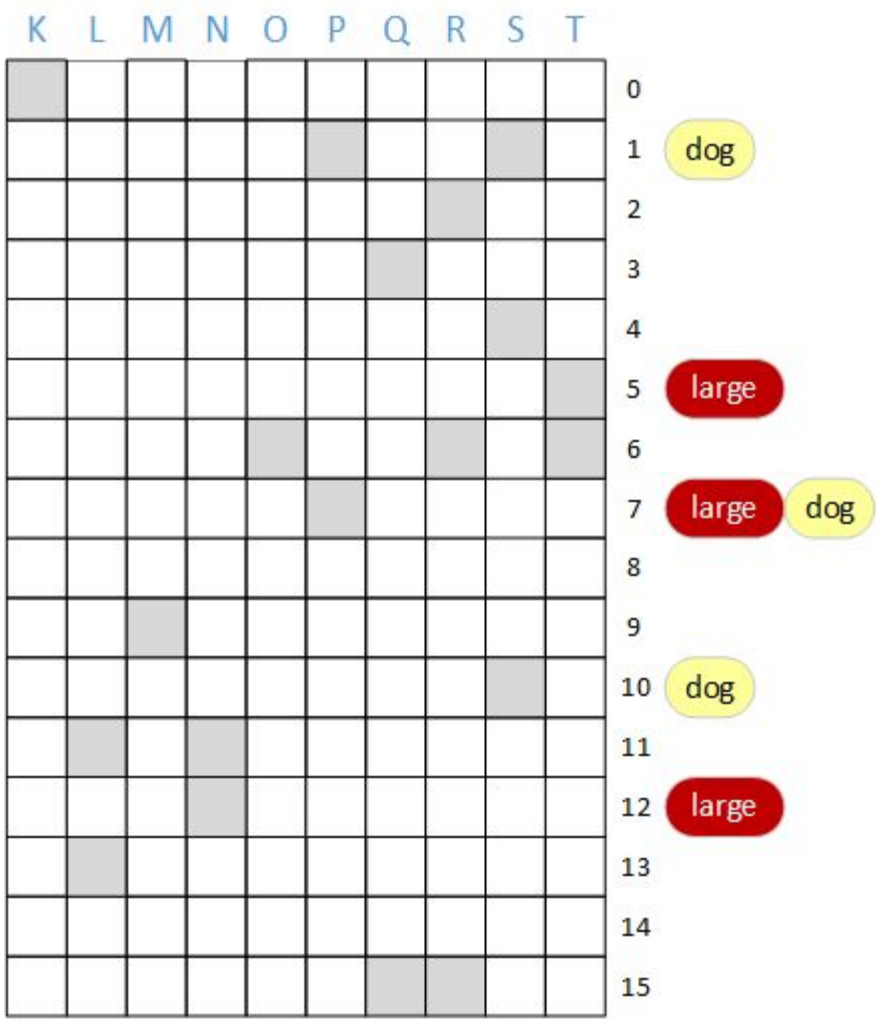
Algorithms: bloom filter

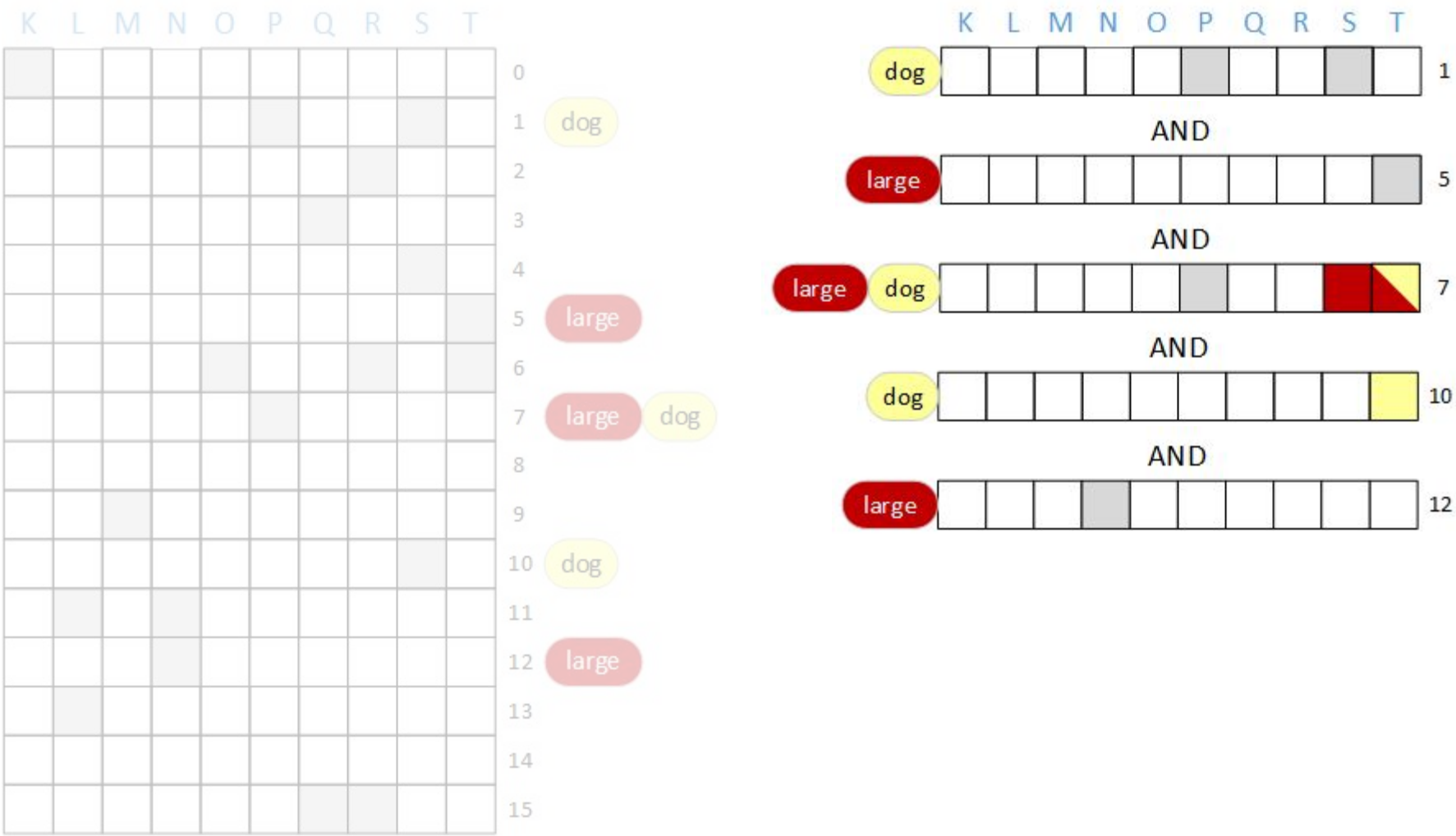


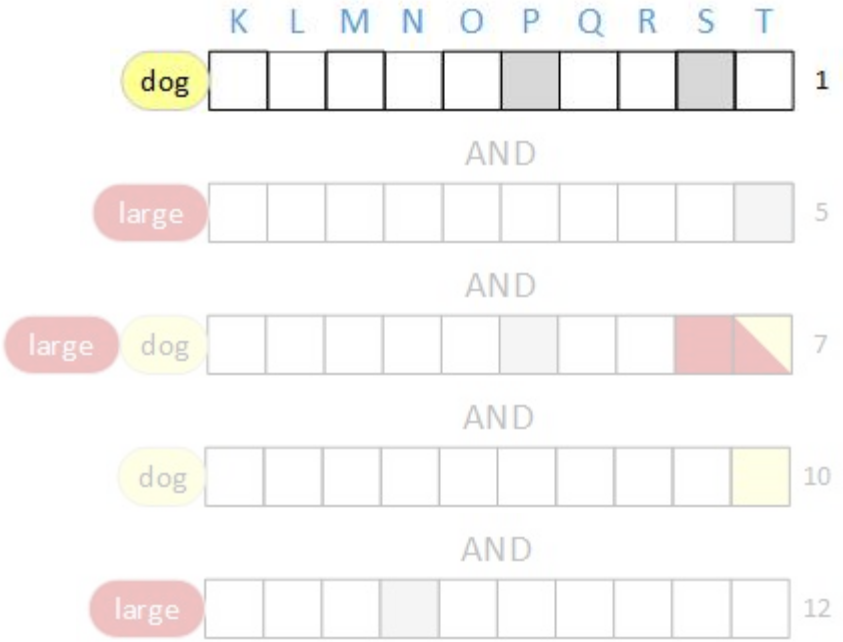
Algorithms: bloom filter



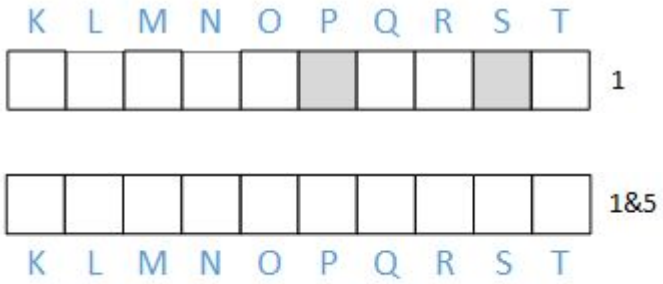
Algorithms: bloom filter

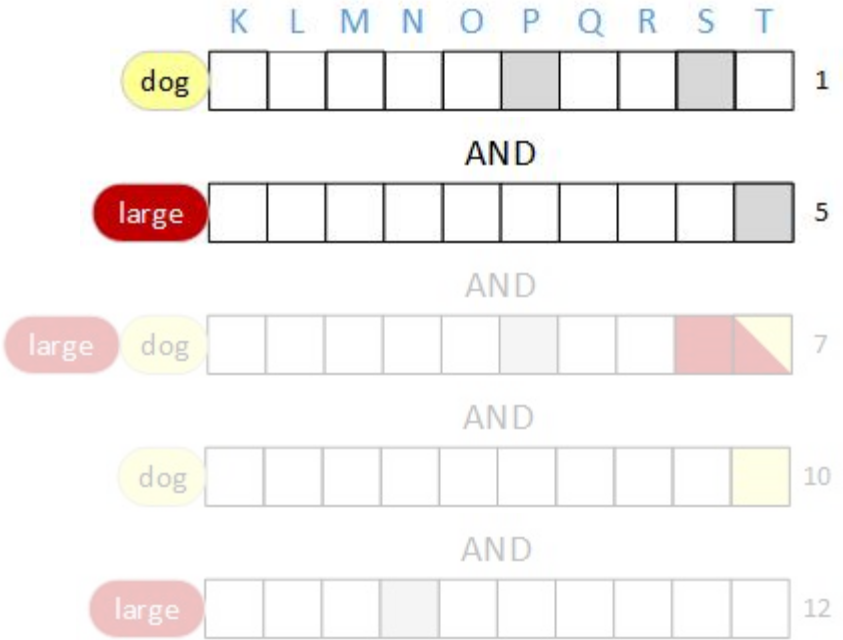

K L M N O P Q R S T
0
1
dog
2
3
4
5
large
6
7
large
dog
8
9
10
dog
11
12
large
13
14
15


K L M N O P Q R S T
0
1
dog
2
3
4
5
large
6
7
large
dog
8
9
10
dog
11
12
large
13
14
15
K L M N O P Q R S T
dog
1
AND
large
5
AND
large
dog
7
AND
dog
10
AND
large
12



Algorithms: bloom filter





Algorithms: bloom filter

How do we estimate perf?

Cost model
Number of operations

512-bit “blocks”

(pay for memory accesses)

How many memory accesses per block?

$$\begin{aligned}
\mathbf{P}\{X > i\} &= \mathbf{P}\{\text{the accumulator at level } i \text{ is not all zeros}\} \\
&= \mathbf{P}\left\{\bigcap_i \neq \vec{0}\right\} \\
&= 1 - \mathbf{P}\left\{\bigcap_i = \vec{0}\right\} \\
&= 1 - \prod_{j=1}^{512} \mathbf{P}\left\{j^{th} \text{ bit in } \bigcap_i \text{ is } 0\right\}
\end{aligned}$$

http://bitfunnel.org

Perf estimation

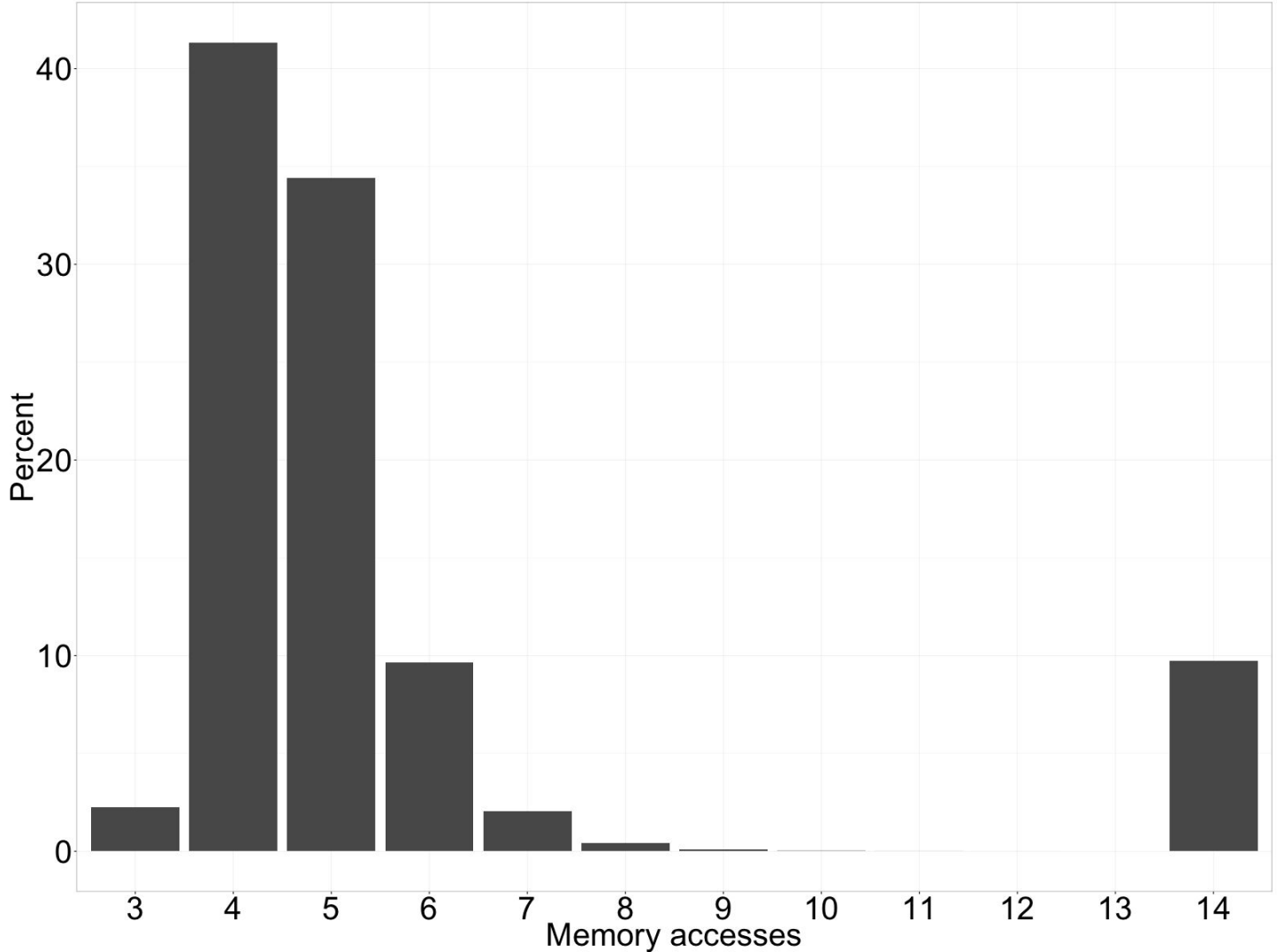
Conditioning on the whether document j is a match:

$$\begin{aligned}
\mathbf{P}\left\{ j^{th} \text{ bit in } \bigcap_i \text{ is } 0 \right\} &\overset{=}{\geq} \begin{cases} 0, & j^{th} \text{ document is a match} \\ 1-(d-s)^i, & \text{otherwise} \end{cases} \\
&\overset{=}{\geq} \begin{cases} 0, & \text{w.p. } s \\ 1-(d-s)^i & \text{w.p. } 1-s \end{cases} \\
&\geq \left(1-(d-s)^i\right) \cdot (1-s)
\end{aligned}$$

Returning to the CDF of X:

$$\mathbf{P}\{X > i\} \leq 1 - \left(\left(1-(d-s)^i\right) \cdot (1-s)\right)^{512}$$

Perf estimation


40
30
20
10
0
Percent
3
4
5
6
7
8
9
10
11
12
13
14
Memory accesses

Why do we have so many rows?

Term rewriting

Term Rewriting

“Large yellow dog”

Term Rewriting

“Large yellow dog” ||

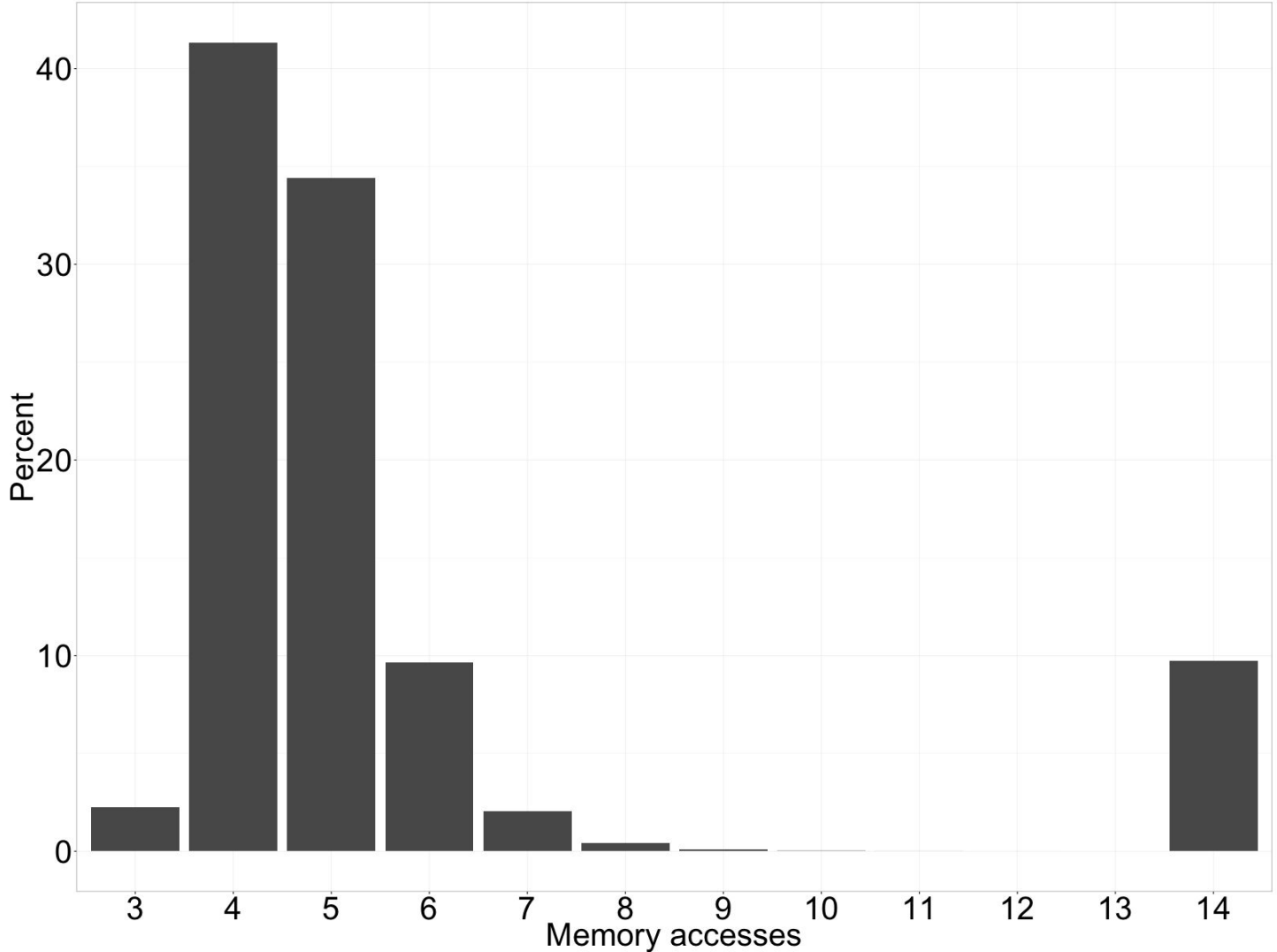
“Golden Retriever”

Term Rewriting

“Large yellow dog” ||

“Golden Retriever” ||

“Old Yeller” ||


40
30
20
10
0
Percent
3
4
5
6
7
8
9
10
11
12
13
14
Memory accesses

Expected performance?

10 M docs / 512 bits per block = 20k “blocks”

10 M docs / 512 bits per block = 20k “blocks”

20 k-blocks * 5 transfers per block = 100 kT

10 M docs / 512 bits per block = 20k “blocks”

20 k-blocks * 5 transfers per block = 100 kT

25 GB/s / 512 bits per transfer = 390 MT/s

Perf estimation

10 M docs / 512 bits per block = 20k “blocks”

20 k-blocks * 5 transfers per block = 100 kT

25 GB/s / 512 bits per transfer = 390 MT/s

390 MT/s / 100 kT = 3900 QPS (with rounding)

Actual performance?

Actual performance ~similar

Small factors

Large factors

Ranking results

Ingestion

(faster than querying)

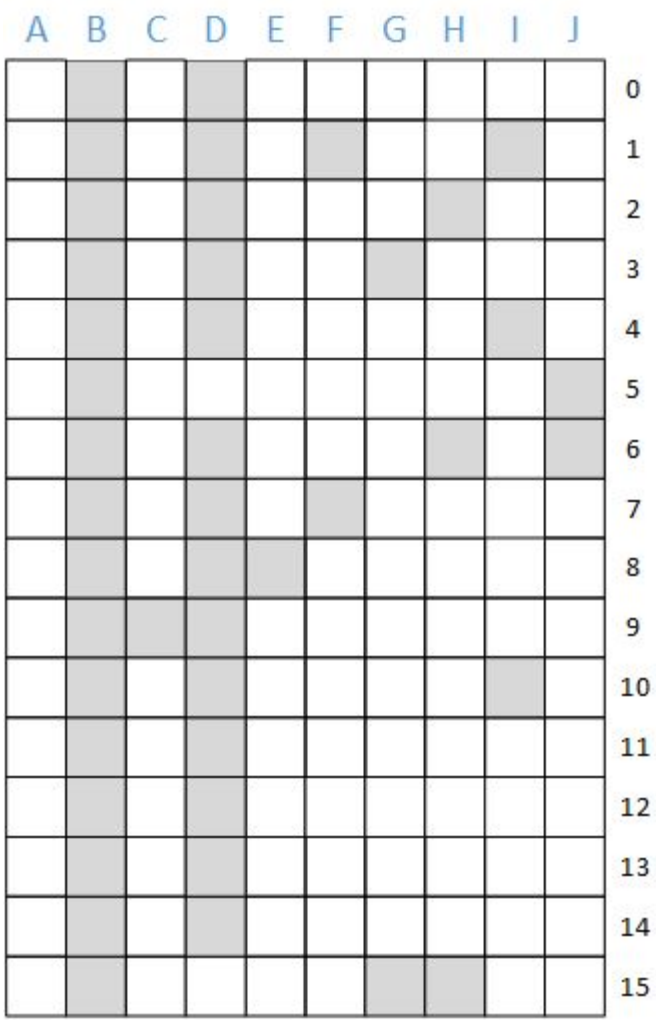
Perf estimation

Ingestion is just setting bits

Hierarchical bloom filters

Perf estimation

Complicating issues?


A B C D E F G H I J
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Conclusions?

False conclusions

Search is simple

False conclusions

Search is simple

Bloom filters are better than posting lists

Zobel et al., Inverted files versus signature files for text indexing; TODS 1998

False conclusions

Search is simple

Bloom filters are better than posting lists

You can easily reason about all performance

Zobel et al., Inverted files versus signature files for text indexing; TODS 1998

Conclusions!

You can reason about perf

It’s often just arithmetic

Acknowledgements

Thanks to Leah Hanson, Mike Hopcroft, Julia Evans, Hari Angepat, David Turner, Danielle Sucher, Ikhwan Lee, Tejas Sapre, Raul Jara, Rich Ercolani, Bert Muthalaly, Harsha Nori, Jeshua Smith, Bill Barnes, Gary Bernhardt, Marek Majkowski, Tom Crayford, Gina Willard, Laura Lindzey, Larry Marbuger, Siddarth Anand, Eric Lemmon, Tom Ballinger, and [Anonymous Reviewer] for feedback.

bitfunnel.org/strangeloop

github.com/bitfunnel/bitfunnel

danluu.com

Unused slides

(thar be dragons)

SLIDE FOR HOMEWORK. TODO: USE DIFFERENT TEMPLATE

Why are posting lists standard?

Literature on alternatives

“**Signatures files** were proposed in [23] and **shown to be inferior** to inverted indexing in [24]. “

“ Inverted indexes have been benchmarked as the most generalisable, and well performing structure (Zobel et al., 1998). **The experiments in this thesis are therefore conducted solely on an inverted index system**.”

“While this technique provides a relatively low computation overhead, studies by Zobel et al. [1998] have shown that inverted files significantly outperform signature files. **We will now focus the analysis on inverted files as it is generally considered to be the most efficient indexing method for most IR systems**.”

“The other two mechanisms are usually adopted in certain applications even if, recently, **they have been mostly abandoned in favor of inverted indexes because some extensive experimental results** [194] have shown that: Inverted indexes offer better performance than signature files and bitmaps, in terms of both size of index and speed of query handling [188]”

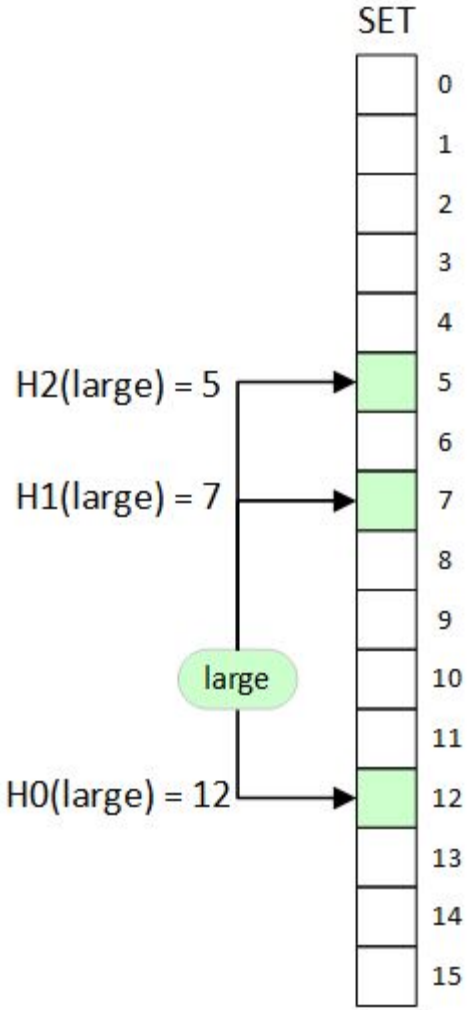
“Zobel et al. [16] compared inverted files and signature files with respect to query response time and space requirements. They found that the inverted files evaluated queries in less time than the signature files and needed less space. Their results showed that the **signature files were much larger, more expensive to construct and update, their response time was unpredictable, they support ranked queries only with difficulty, they did not scale well and they were slow**“

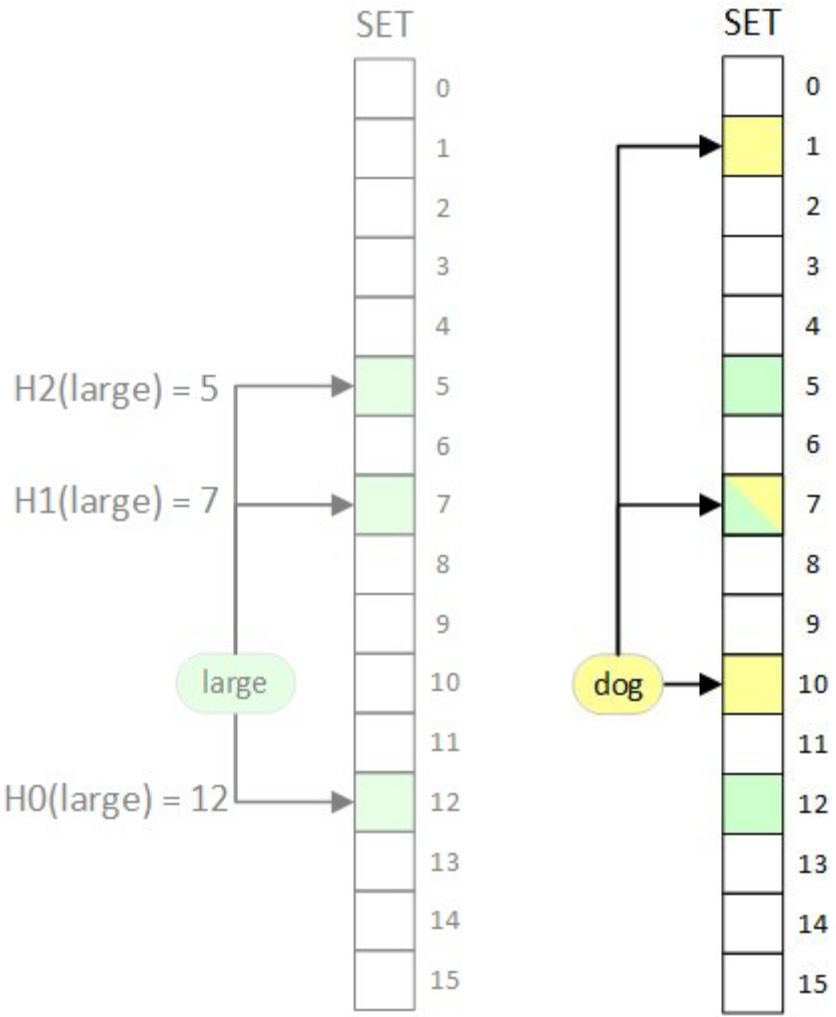
Zobel et al., actual quotes

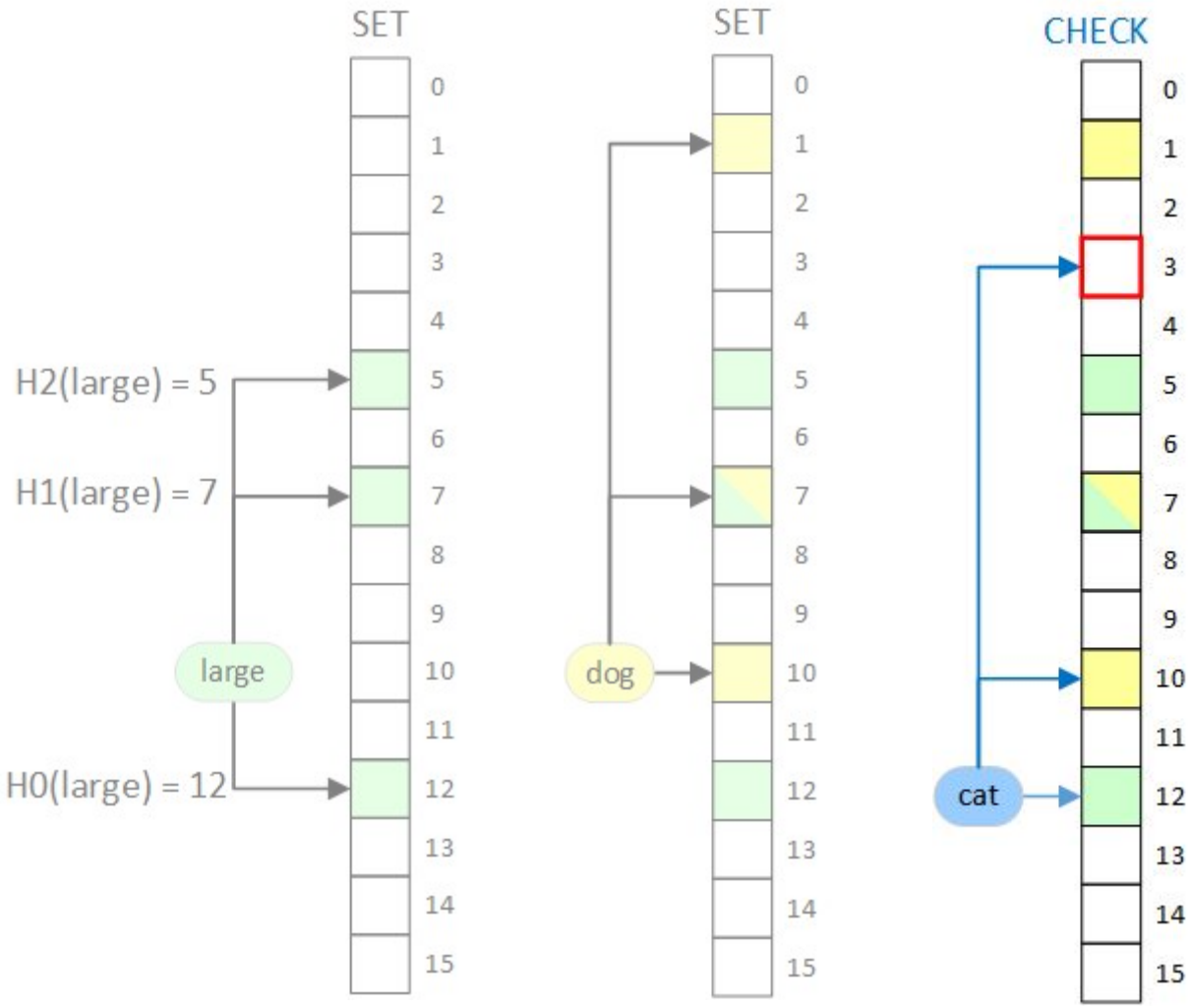
“Inverted file indexes with in-memory search structures **require no more disk accesses** to answer a conjunctive query than do bitsliced signature files.”

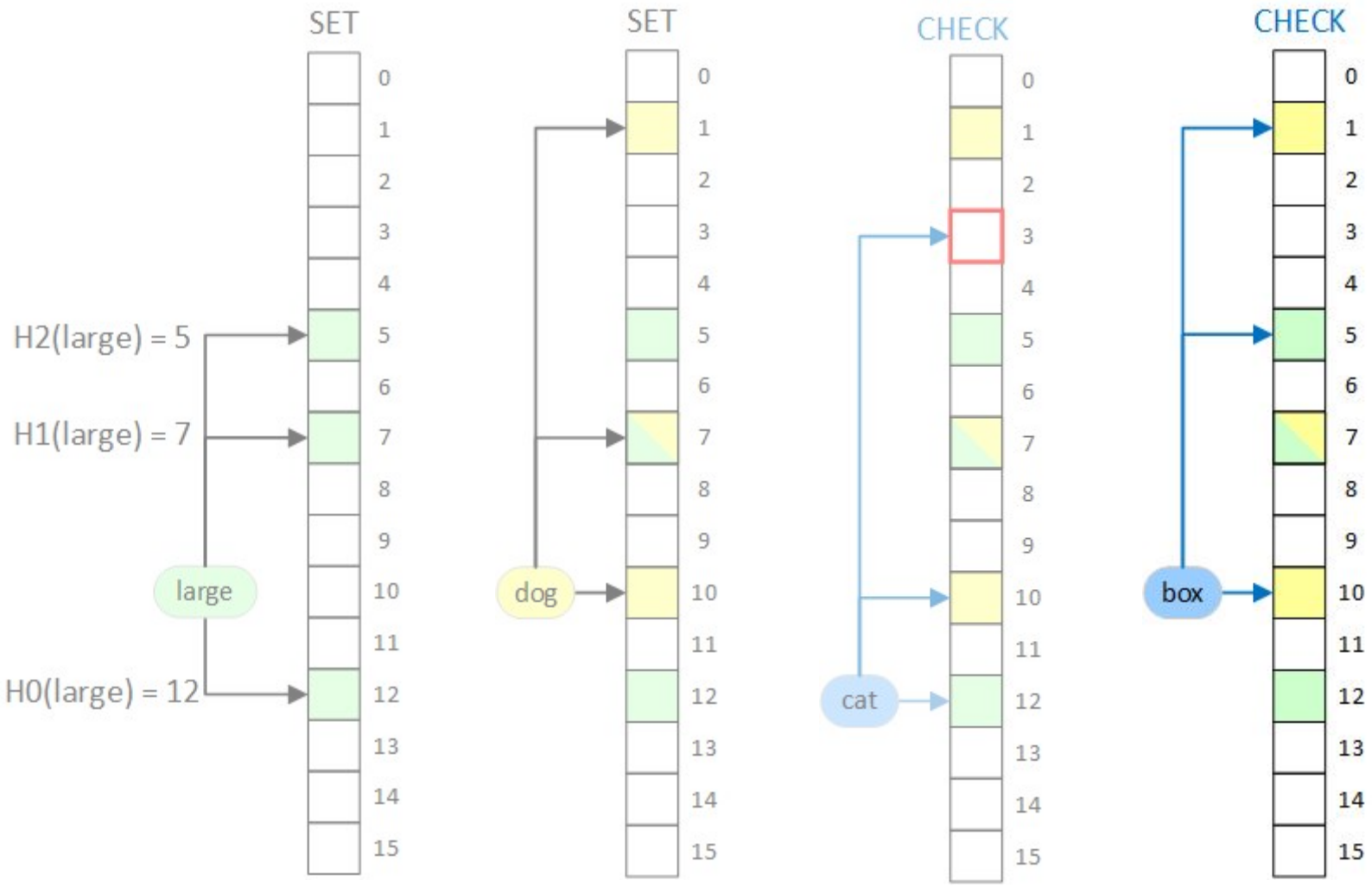
“One of the difficulties in the comparison of inverted files and signature files is that many variants of signature file techniques have been proposed, and **it is possible that some combination of parameters and variants will result in a better method**.”

Citations are lossy

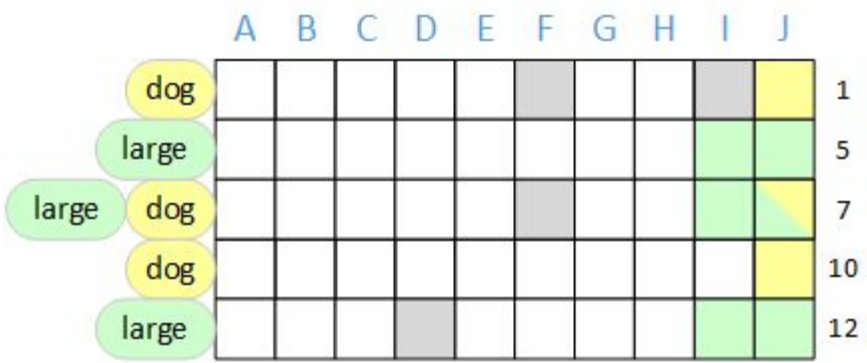

SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12

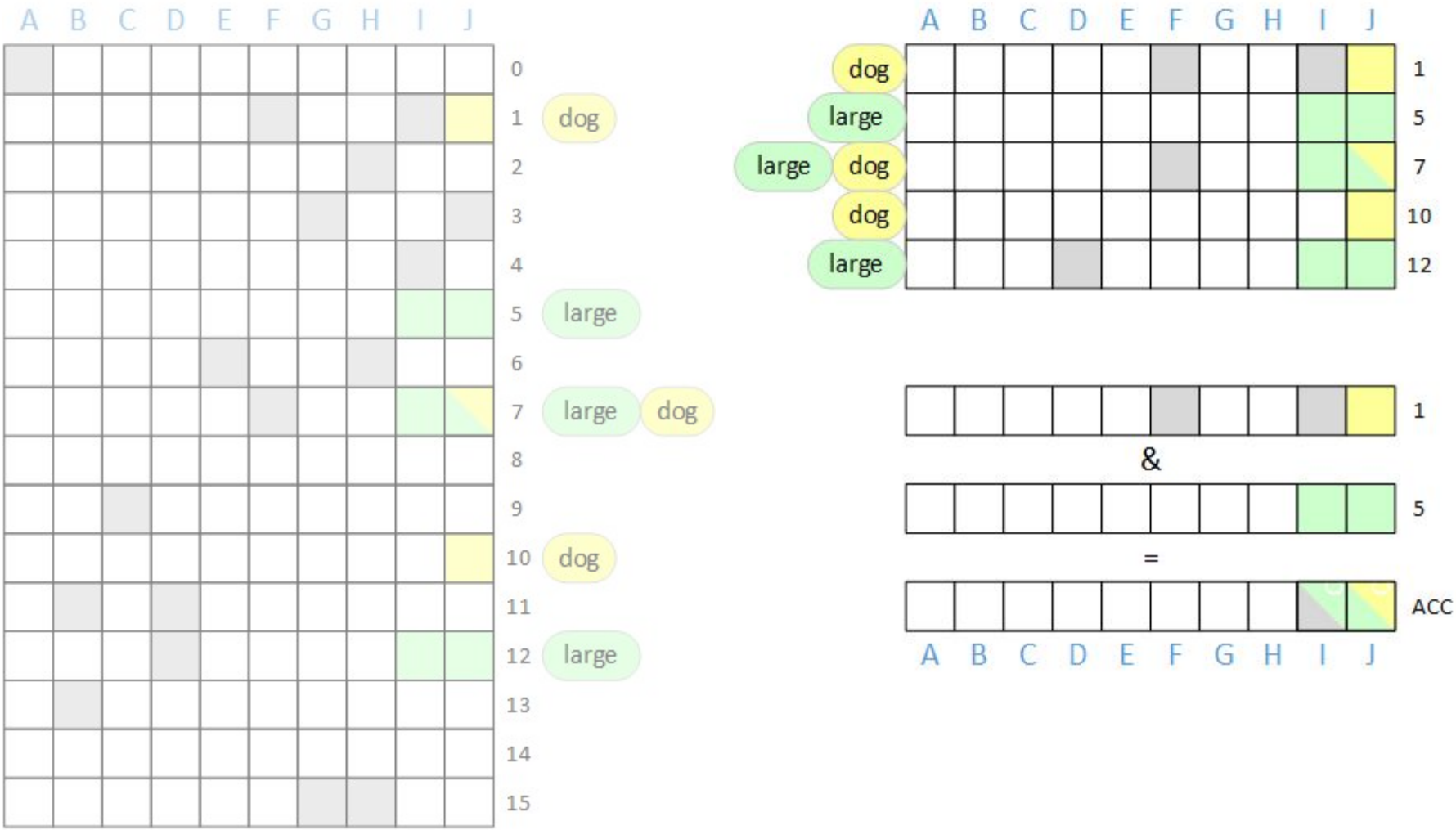

SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12
SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
dog

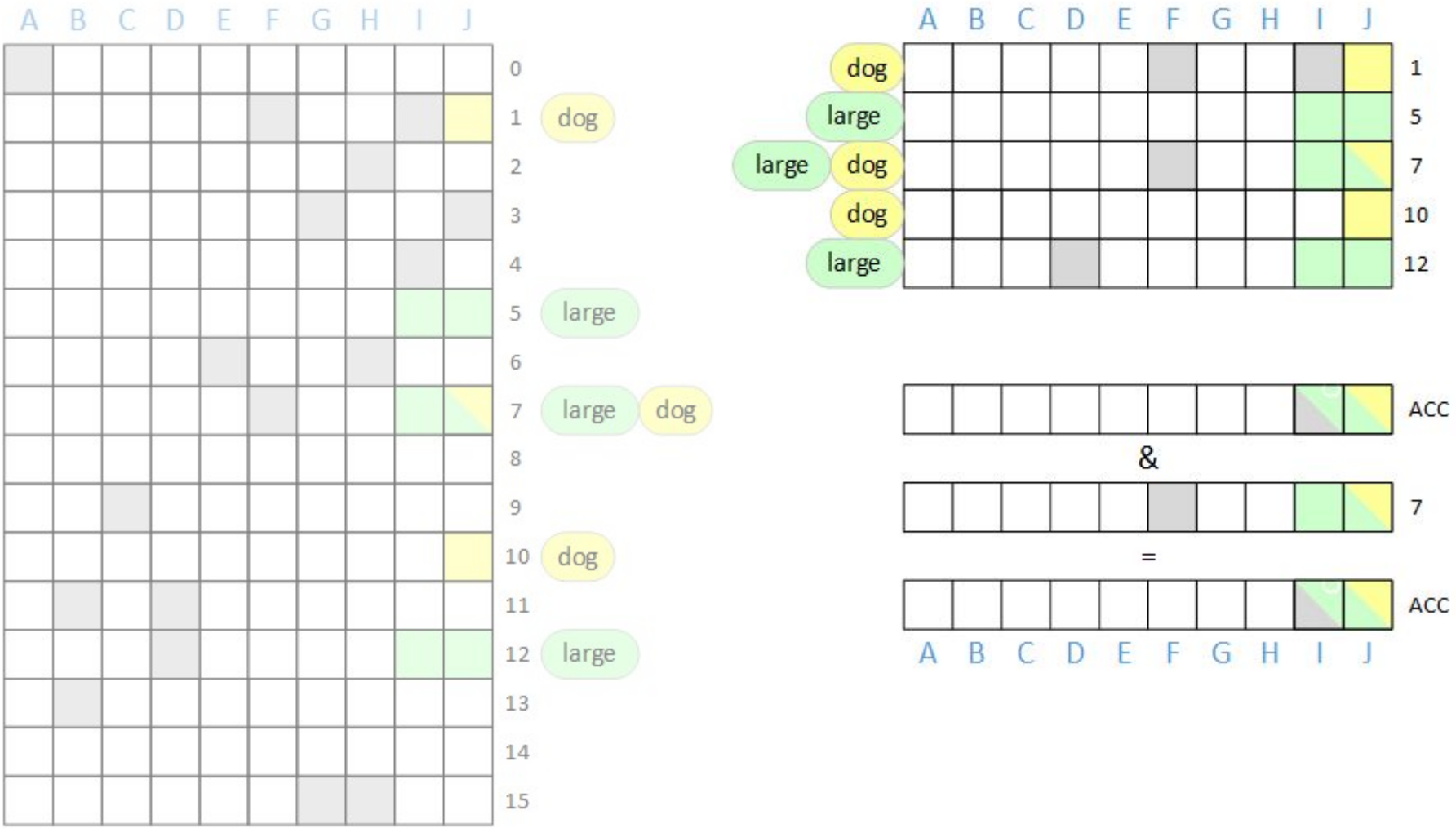

SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12
SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
dog
CHECK
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
cat


SET
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
H2(large) = 5
H1(large) = 7
large
H0(large) = 12
SET
dog
CHECK
cat
CHECK
box

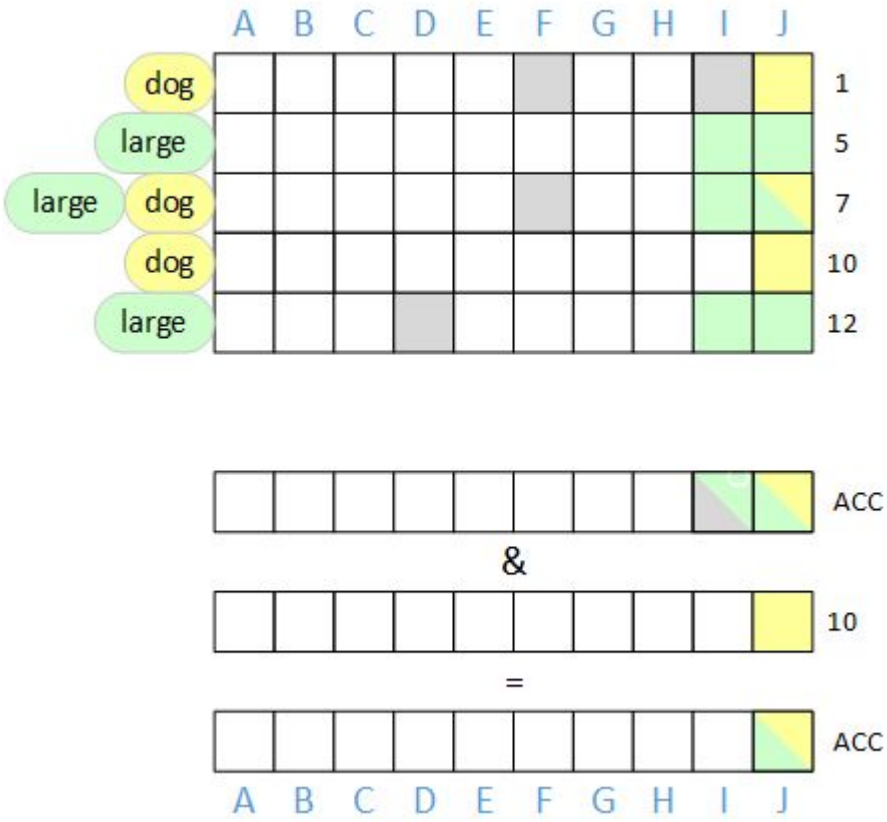

A B C D E F G H I J
0
1 dog
2
3
4
5 large
6
7 large dog
8
9
10 dog
11
12 large
13
14
15


A B C D E F G H I J
dog 1
large 5
large dog 7
dog 10
large 12

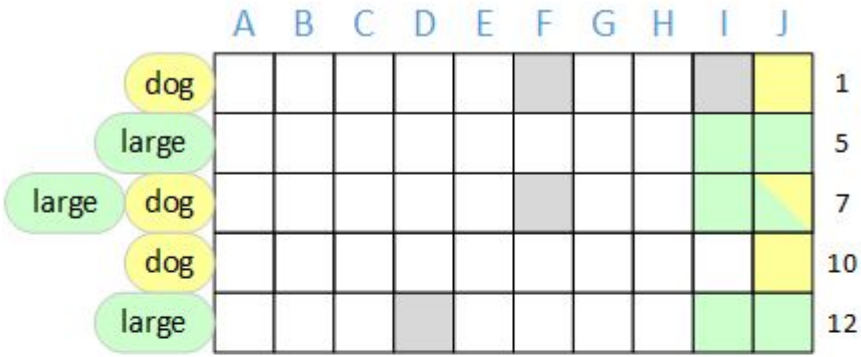

A B C D E F G H I J
0
1 dog
2
3
4
5 large
6
7 large dog
8
9
10 dog
11
12 large
13
14
15
A B C D E F G H I J
dog
large
large dog
dog
large
1
5
7
10
12
1
&
5
=
ACC
A B C D E F G H I J

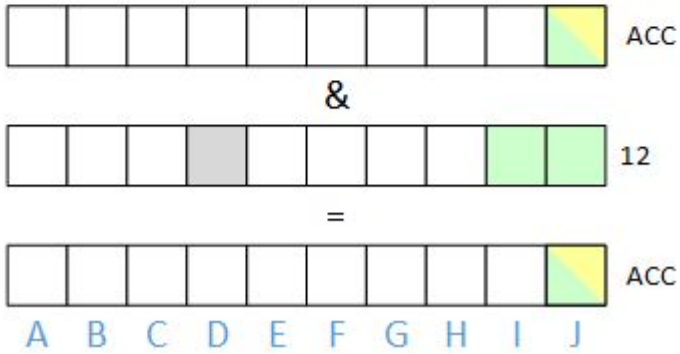

A B C D E F G H I J
0
1 dog
2
3
4
5 large
6
7 large dog
8
9
10 dog
11
12 large
13
14
15
A B C D E F G H I J
dog 1
large 5
large dog 7
dog 10
large 12
ACC
&
7
=
ACC
A B C D E F G H I J

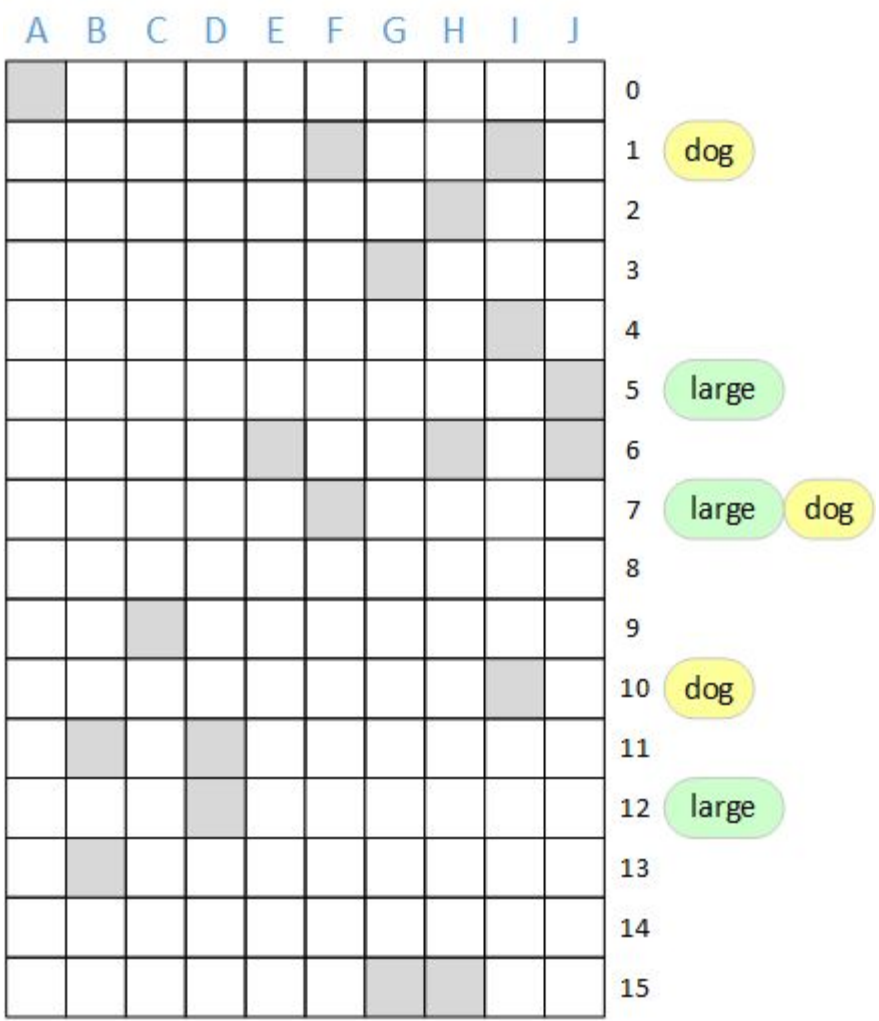

A B C D E F G H I J
0
1 dog
2
3
4
5 large
6
7 large dog
8
9
10 dog
11
12 large
13
14
15

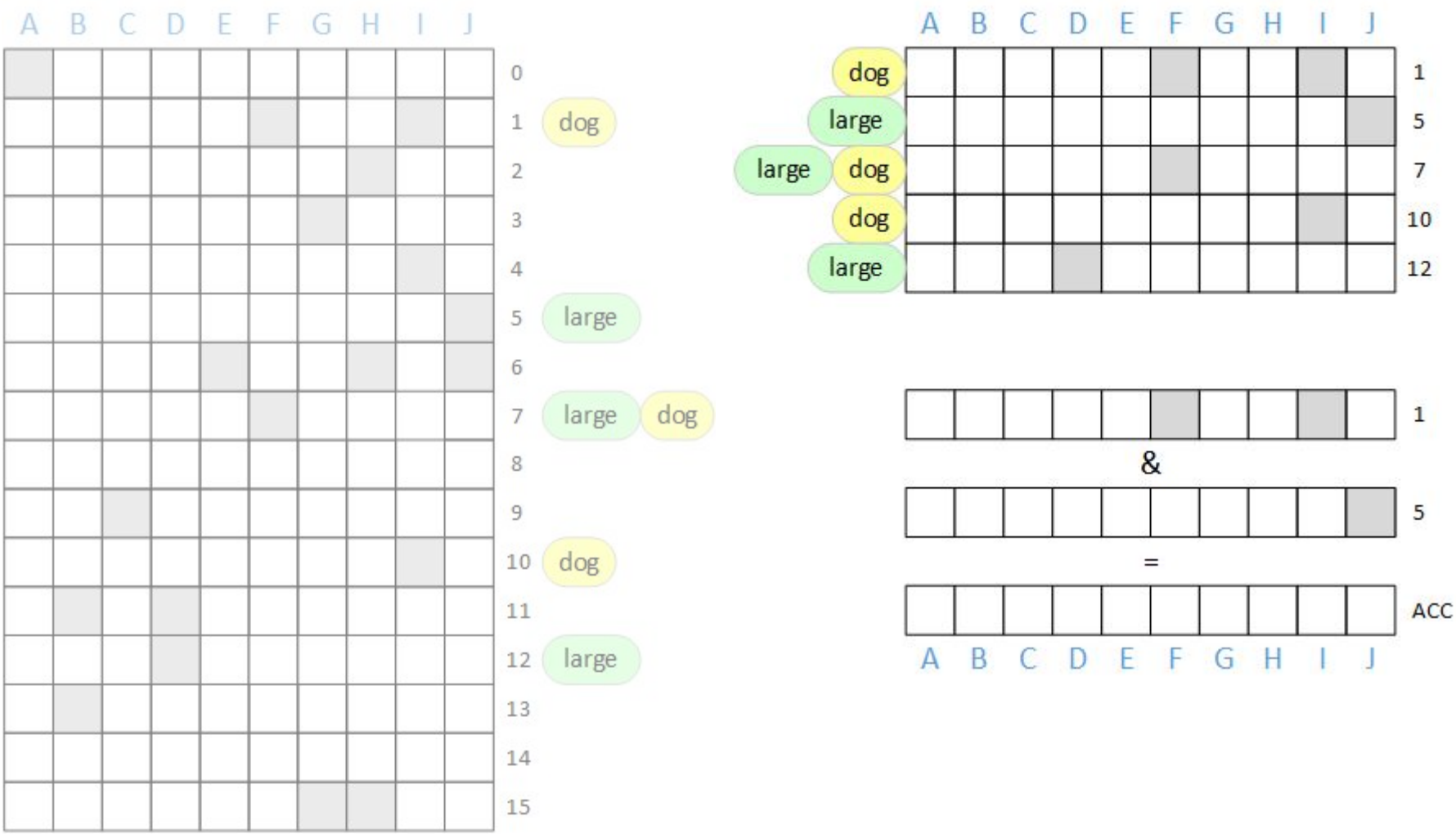

A B C D E F G H I J
dog
large
large dog
dog
large
1
5
7
10
12
ACC
&
10
=
ACC
A B C D E F G H I J


A B C D E F G H I J
0
1 dog
2
3
4
5 large
6
7 large dog
8
9
10 dog
11
12 large
13
14
15


A B C D E F G H I J
dog
large
large dog
dog
large
1
5
7
10
12


ACC
&
12
=
ACC
A B C D E F G H I J


A B C D E F G H I J
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
dog
large
large
dog
dog
large


A B C D E F G H I J
0
1 dog
2
3
4
5 large
6
7 large dog
8
9
10 dog
11
12 large
13
14
15
A B C D E F G H I J
dog
large
large dog
dog
large
1
5
7
10
12
1
&
5
=
ACC
A B C D E F G H I J

Search: why do we care?

$20M/yr * 2% savings

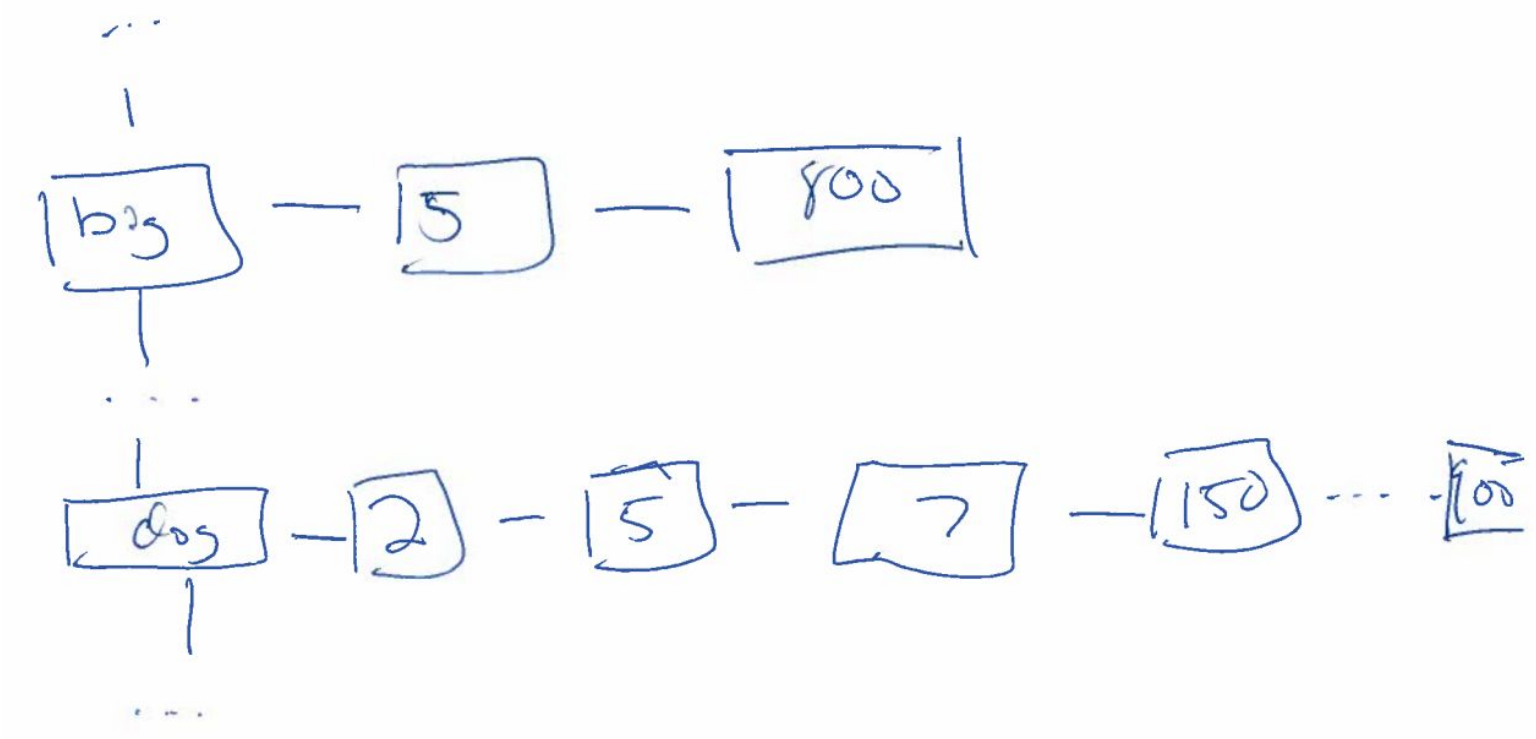
=

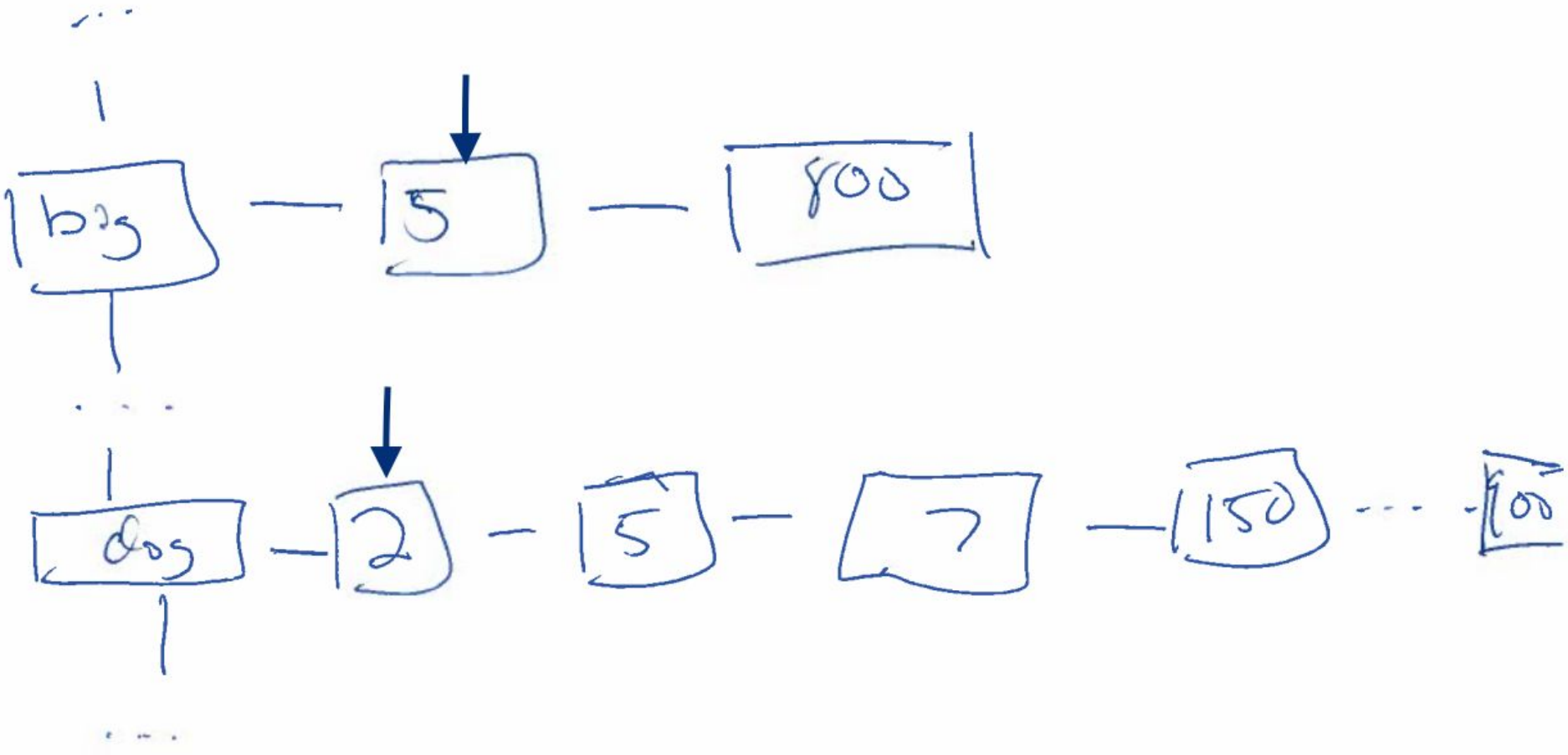
$400k/yr

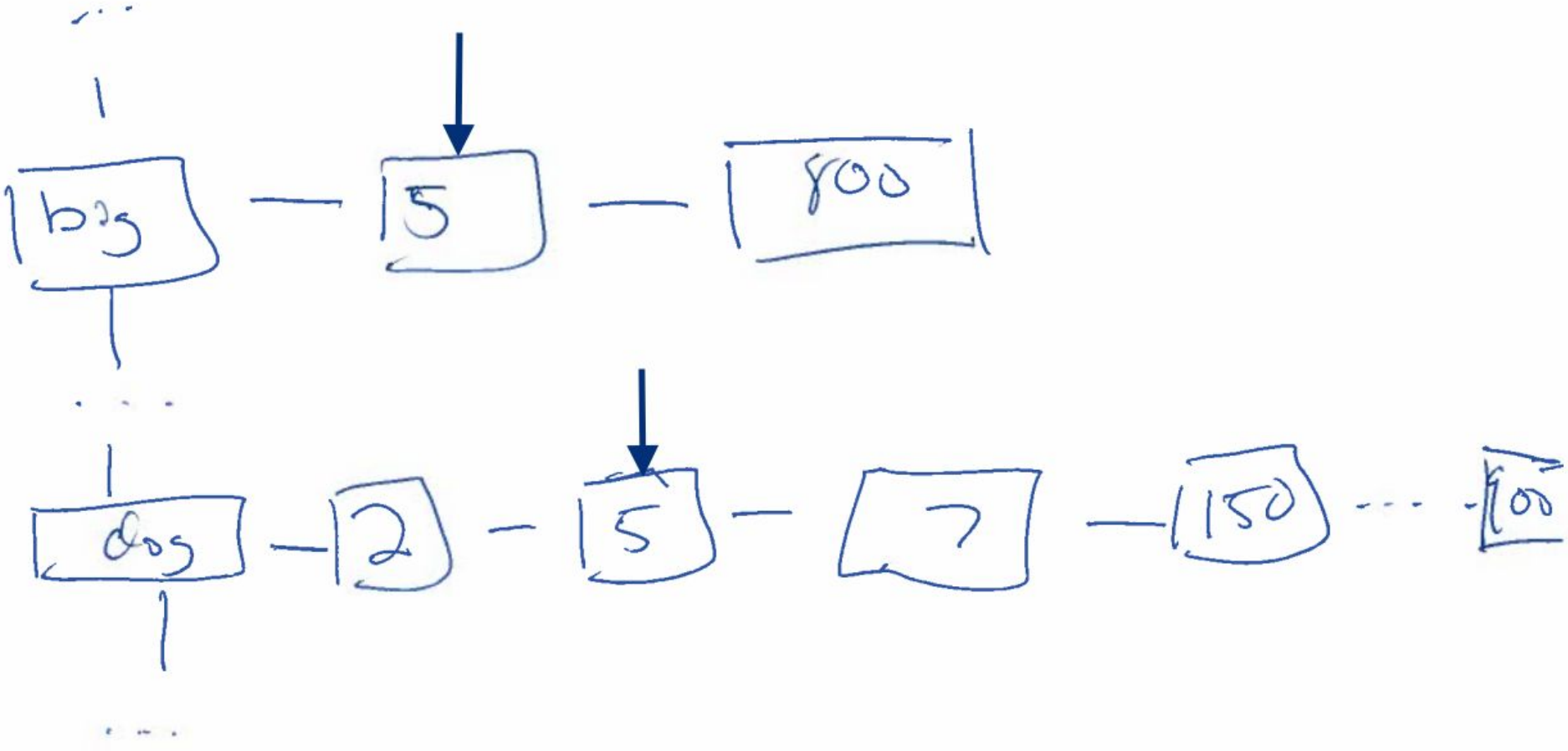
How things fit together

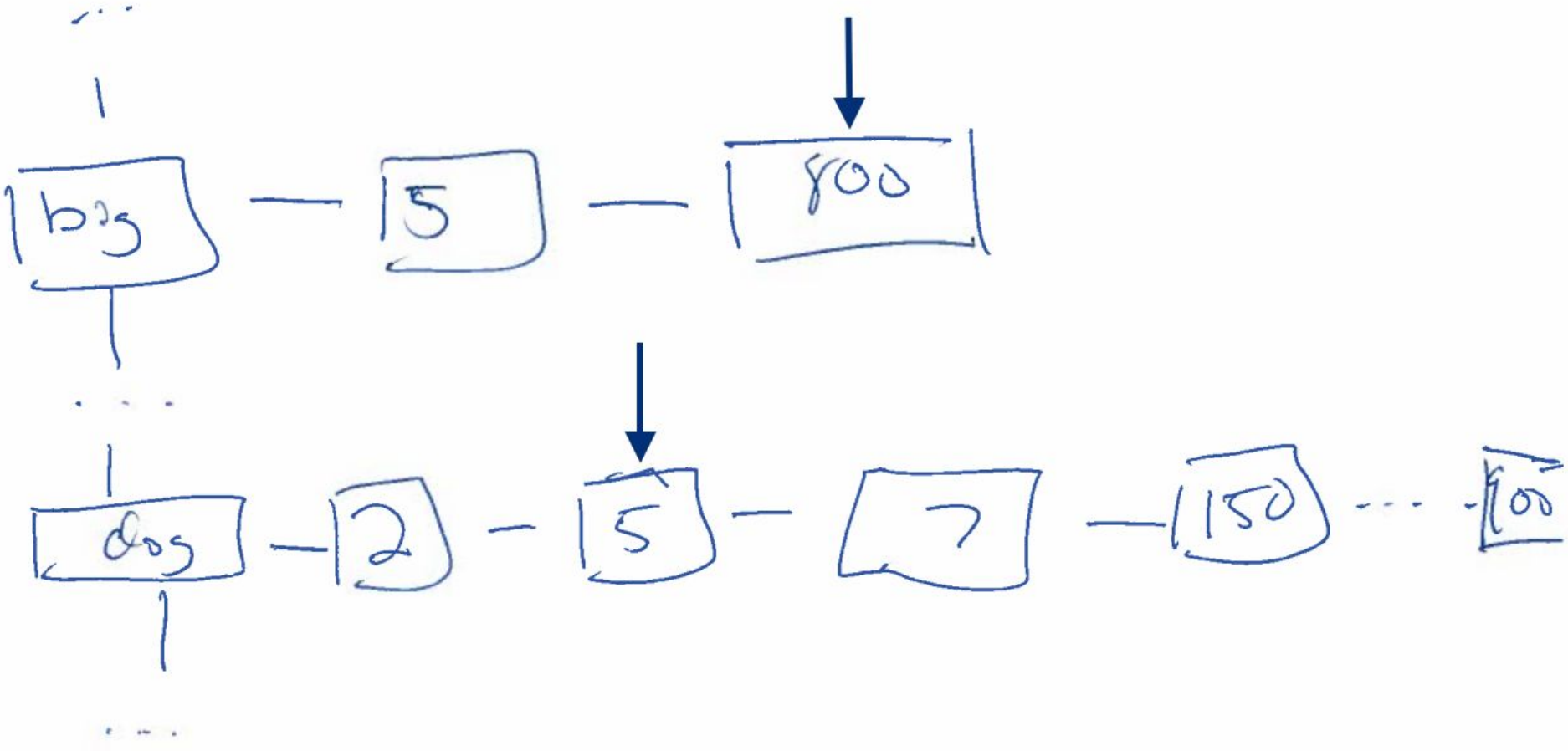
TODO: add diagram

Posting list




1
big
5
800
dog
2
5
7
150
100


1
big
5
800
dog
2
5
7
150
100


big
5
800
dog
2
5
7
150
100

How many terms?

TODO: pseudo-code

TODO: diagram about how bits drop out

TODO: search is a high dynamic range problem.

TODO: higher rank rows

TODO: sharding by document length

TODO: diagram of how things fit together. Could just be concentric circles

Posting lists are standard

Coursera:
Term Vocabulary and Postings Lists (IIR Ch. 2)
Notes:
[powerpoint] [PDF/6] [PDF/1]
Readings
IIR Ch. 2
MG Ch. 3.6 4.3
MIR 7.2
Porter's stemmer (MIR), Porter stemming algorithm (Official)
A skip list cookbook (Pugh 1990)
Fast phrase querying with combined indexes (Williams, Zobel, Bahle 2004)
Efficient phrase querying with an auxiliary index (Bahle, Williams, Zobel 2002)

Coursera:
Index Compression (IIR Ch. 5)
Notes:
[powerpoint] [PDF/6] [PDF/1]
Readings:
IIR Ch. 5
MG 3.3, 3.4
Compression of inverted indexes for fast query evaluation (Scholer et al. 2002)
Inverted index compression using word-aligned binary codes (Anh and Moffat 2005)

Posting list optimizations

Skip list

Delta compression

etc.

Search

Perf: how to think about it?

Performance

Search is BIG

Parsing / Tokenization

Harder than it sounds

Search is a big problem

Tokenization

- Some languages mix alphabets, are partially left-to-right and right-to-left, etc.
- Can’t drop non-alphanumeric characters (C# vs C++)

Multi-language queries

Ranking / Relevance

Distributed Systems

etc.