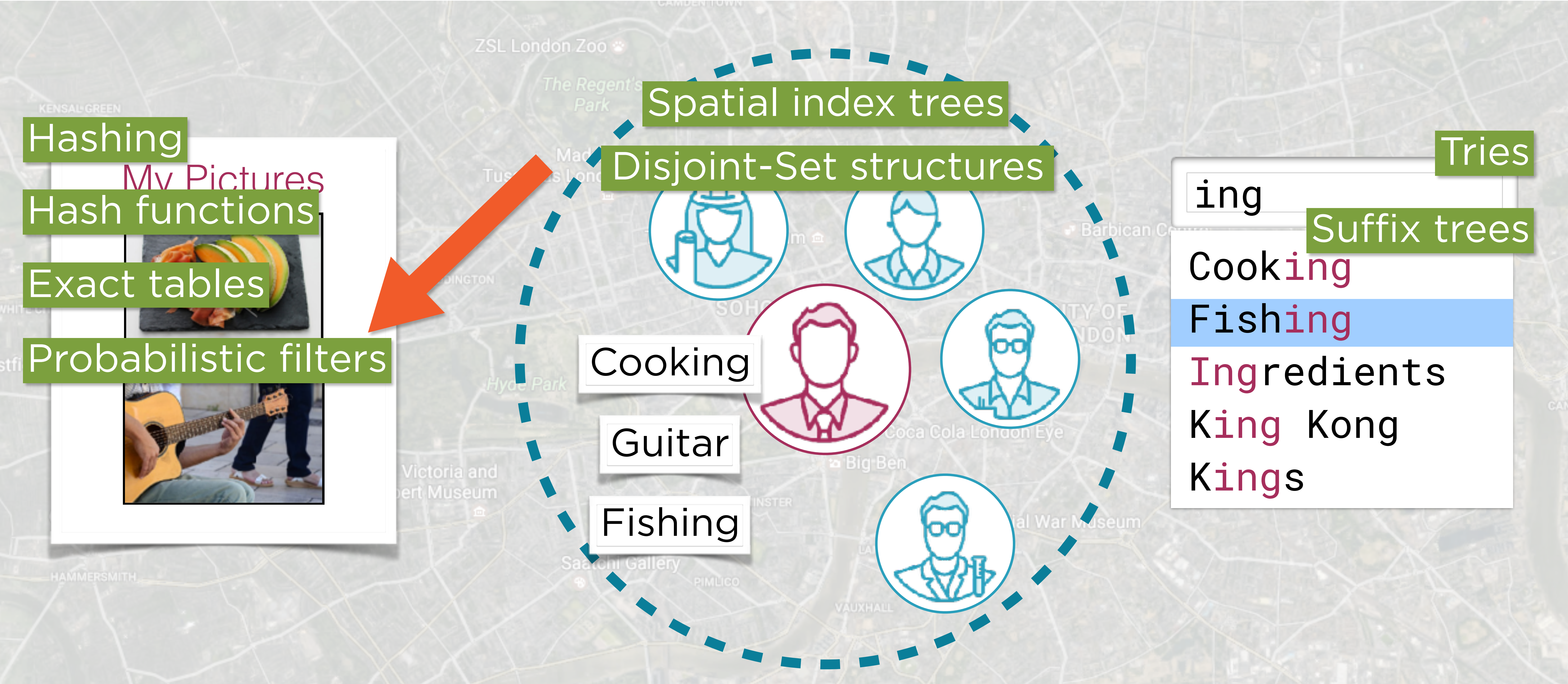
Hashing 2:
Saving Space with Bloom Filters

Rasmus Resen Amossen
SOLUTION ARCHITECT
rasmus.resen.org

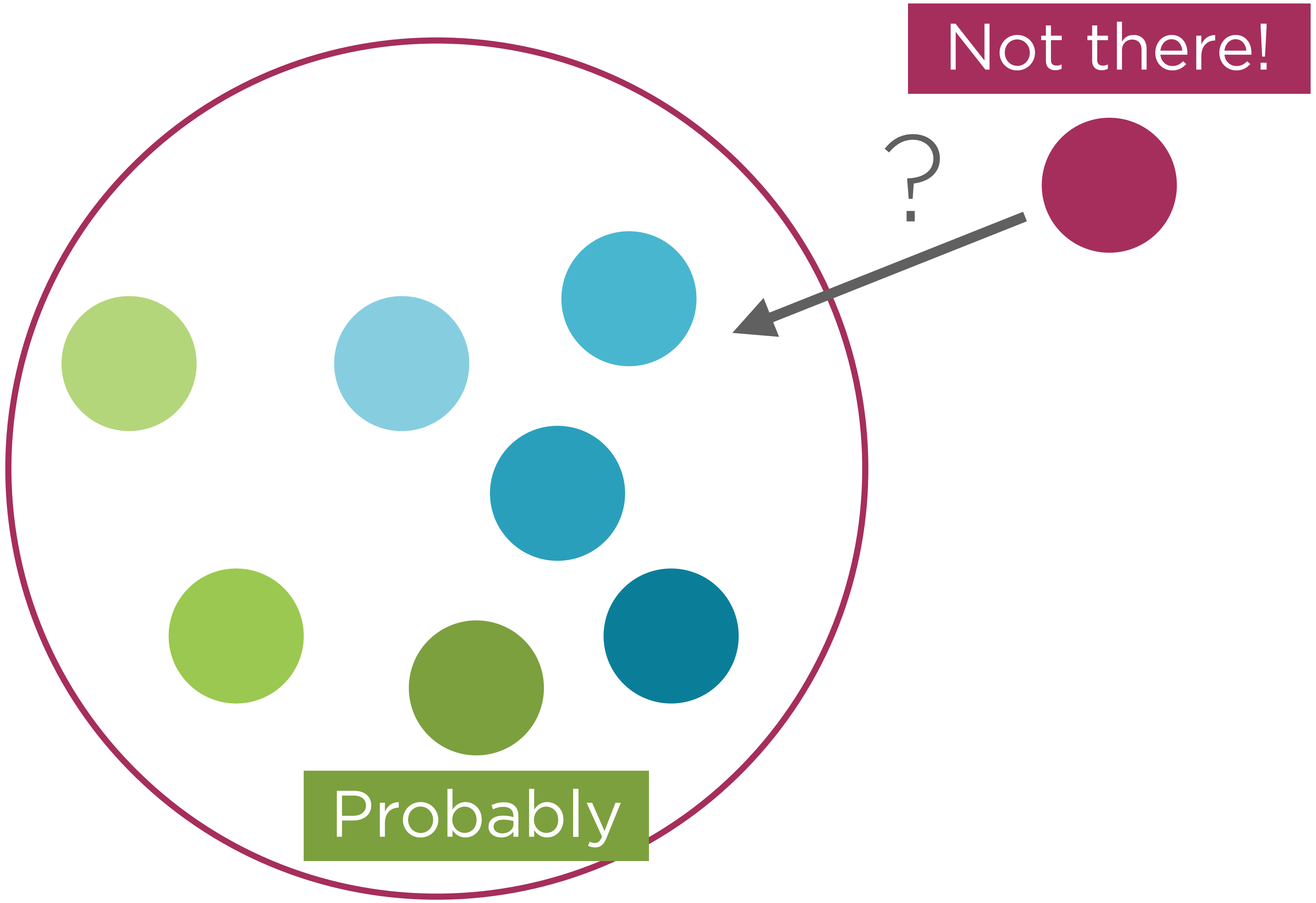
The Match Finder App



Demo

Wasting cache space

Bloom Filters



Bloom Filters

16 bits (2 bytes)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Use k independent hash functions: $h_1(\ldots)$, $h_2(\ldots)$, …, $h_k(\ldots)$

Set the bit specified by each hash function.

Key	h_1	h_2	h_3
"MyFirstLongKeyValue"	14	6	5
"MySecondLargeKeyValue"	2	10	6

Lookup

Filter is never full

Only increased rate of false positives

1	0	0	1	1	0	0	0	1	0	0	0	1	0
2	3	4	5	6	7	8	9	10	11	12	13	14	15

All k bits set: It's probably there.

Not all k bits set: Definitely not there.

Key	h_1	h_2	h_3	
"MyFirstLongKeyValue"	14	6	5	Probably
"MySecondLargeKeyValue"	2	10	6	Probably
"MyNonExistingKeyValue"	14	2	12	Not there!
"MyFourthLargeKeyValue"	[illegible]	[illegible]	14	Probably

False positive

Deletion

Not supported

...in this scheme (check "*counting bloom filters*")

0	0	1	0	0	1	1	0								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

All k bits set: It's probably there.

Not all k bits set: Definitely not there.

Key	h_1	h_2	h_3	
"MyFirstLongKeyValue"	14	6	5	Probably
"MySecondLargeKeyValue"	2	10	6	Probably
"MyNonExistingKeyValue"	14	2	12	Not there!
"MyFourthLargeKeyValue"			14	Probably

False positive

Sizing

Targets:

Minimize filter size in number of bits (m) **Deduce**

Minimize risk of false positives (p) **Given**

Other knobs:

Number of hash functions (k) **Deduce**

Number of elements (n) **Given**

$p = 10\%$	$p = 1\%$	$p = 0.1\%$
$m = 4.79 \cdot n$	$m = 9.59 \cdot n$	$m = 14.38 \cdot n$
$k = 4$	$k = 7$	$k = 10$

If n = 1M then...

m = 4.8M (0.6 MB)	m = 9.6M (1.2 MB)	m = 14.4M (1.8 MB)

If n = 100M then...

m = 480M (56 MB)	m = 959M (117 MB)	m = 1.4G (175 MB)

Trick #1

How do we get that many independent hash functions?

Share the bits!

$$\underbrace{\texttt{10000100101011110100}}_{h_1(\cdots)}\underbrace{\texttt{00010011100001000111}}_{h_2(\cdots)}\underbrace{\texttt{10111111110101110001}}_{h_3(\cdots)}$$

Bloom filter is *m* bits wide.

Requires $\log_2(m)$ bits in hash value to address.

Sizing

Targets:
- Minimize filter size in number of bits (m) **Deduce**
- Minimize risk of false positives (p) **Given**

Other knobs:
- …unctions (k) **Deduce**
- …ts (n) **Given**

Requires a 20 bit hash value

Requires a 30 bit hash value

$p = 10\%$	$p = 1\%$	$p = 0.1\%$
$m = 4.79 \cdot n$	$m = 9$…	$m = 14.38 \cdot n$
$k = 4$	$k = 7$…	$k = 10$

If n = 1M then…

m = 4.8M (0.6 MB)	m = 9.6M (1.2 MB)	m = 14.4M (1.8 MB)

If n = 100M then…

m = 480M (56 MB)	m = 959M (117 MB)	m = 1.4G (175 MB)

Trick #2

How do we get that many independent hash functions?

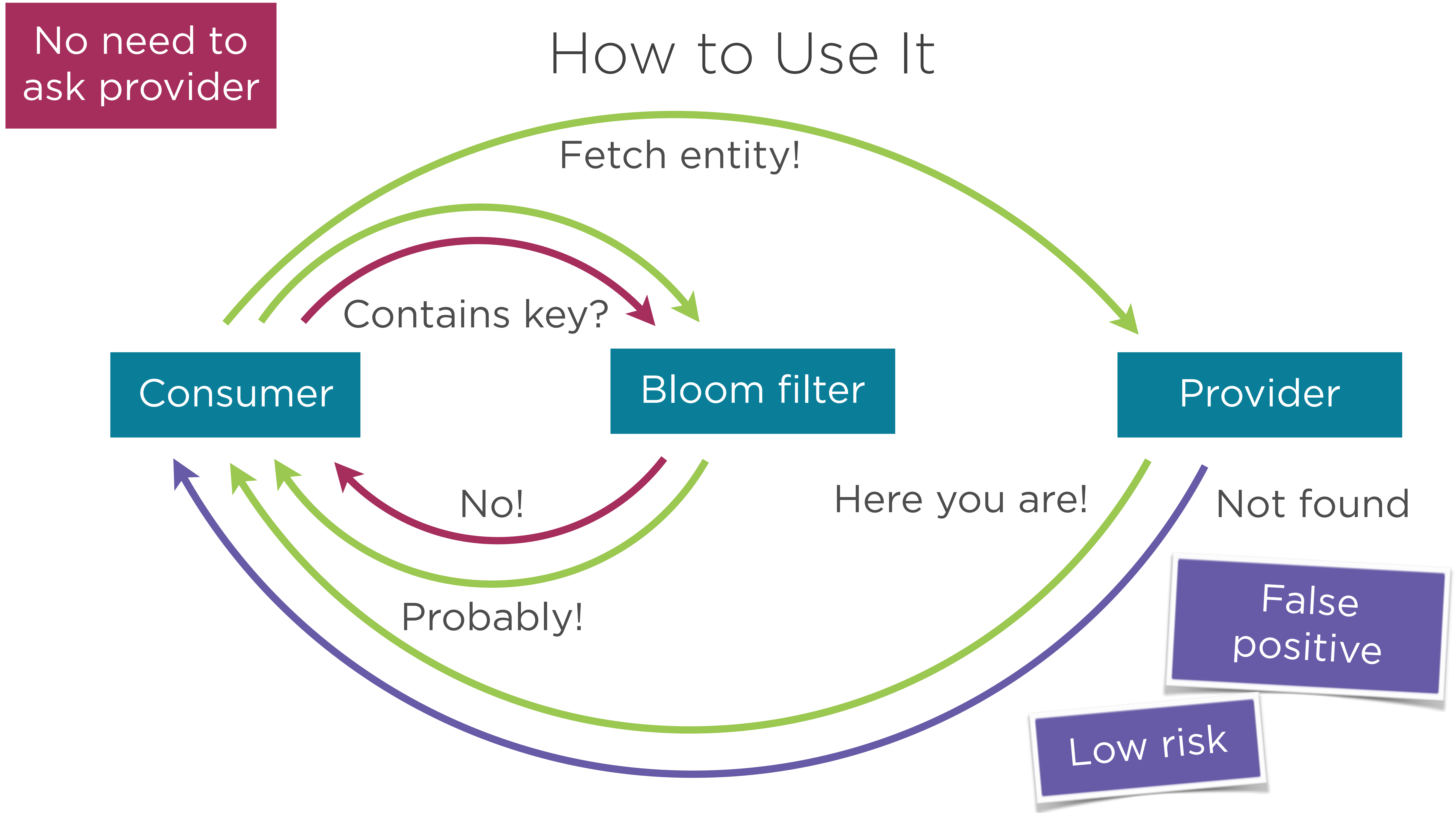
Combine two hash functions, $h_1(\cdots)$ and $h_2(\cdots)$.

$$h_i(\cdots) = h_1(\cdots) + i \cdot h_2(\cdots)$$

Example:

$$h_4(\cdots) = h_1(\cdots) + 4 \cdot h_2(\cdots)$$
$$h_5(\cdots) = h_1(\cdots) + 5 \cdot h_2(\cdots)$$
$$h_6(\cdots) = h_1(\cdots) + 6 \cdot h_2(\cdots)$$

How to Use It


No need to ask provider
Fetch entity!
Contains key?
Consumer
Bloom filter
Provider
No!
Here you are!
Not found
Probably!
False positive
Low risk

Applications

Cache saver

First-responder in NoSQL databases

Identify malicious URLs

Hy-phen-a-tion

Wikipedia: “Bloom filter”

Bloom filter by example: `tinyurl.com/bloomByExample`

In-depth details: `tinyurl.com/cuckoobloom`

Demo

Saving the cache

Lessons Learned

111001001010100111001001001010110100011110110011110101011001

Bits are set using multiple hash functions

Lookups: Check all bits specified by hash functions

False positives

No false negatives!

Size and number of hash functions depends on false positive rate