Probabilistic structures for scalable computing

William Benton • @willb • willb@redhat.com

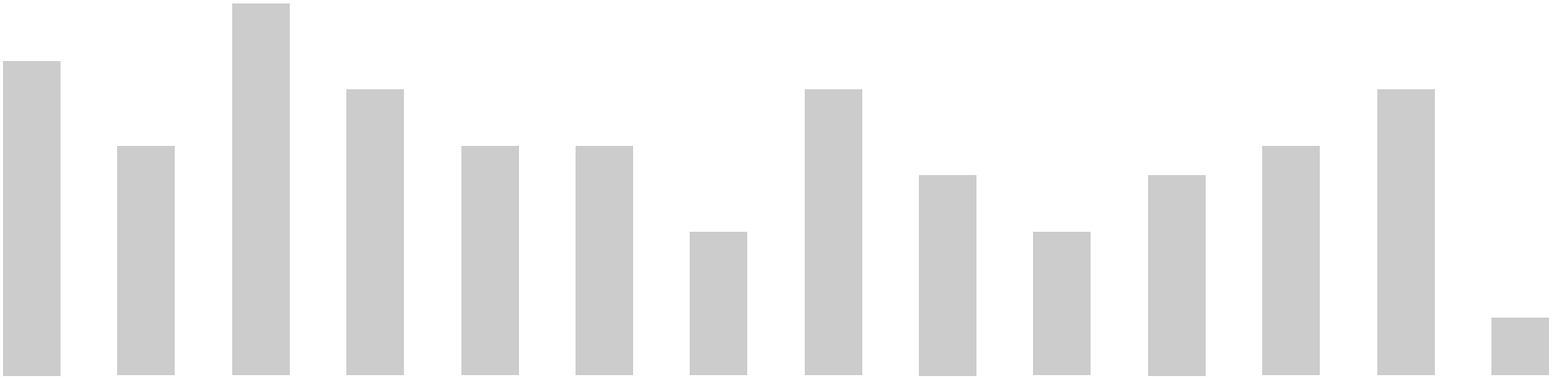
William Benton
@willb

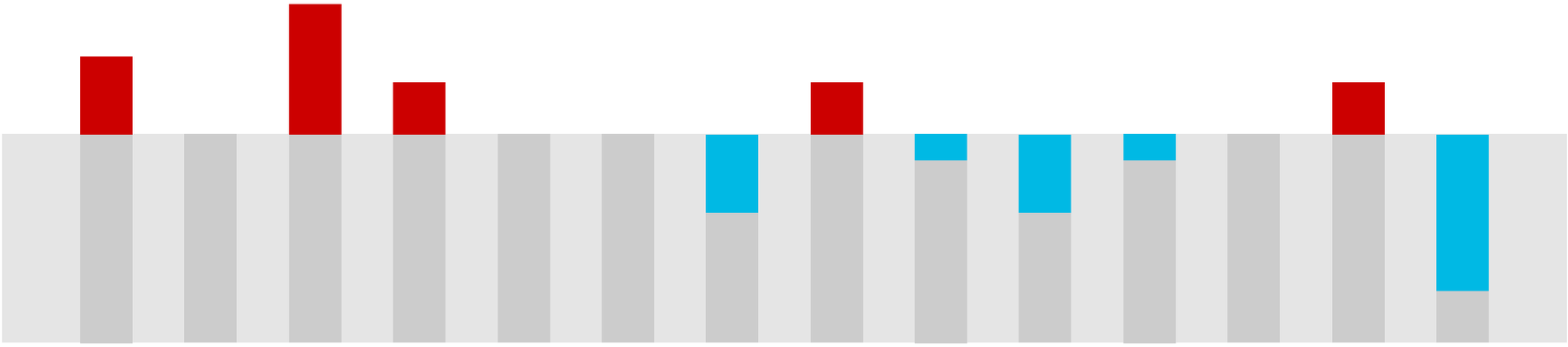
Idea: collect stories from engineers about the problem that first led them to realize the utility of a particular technique.

5:39 PM - 2 Feb 2017

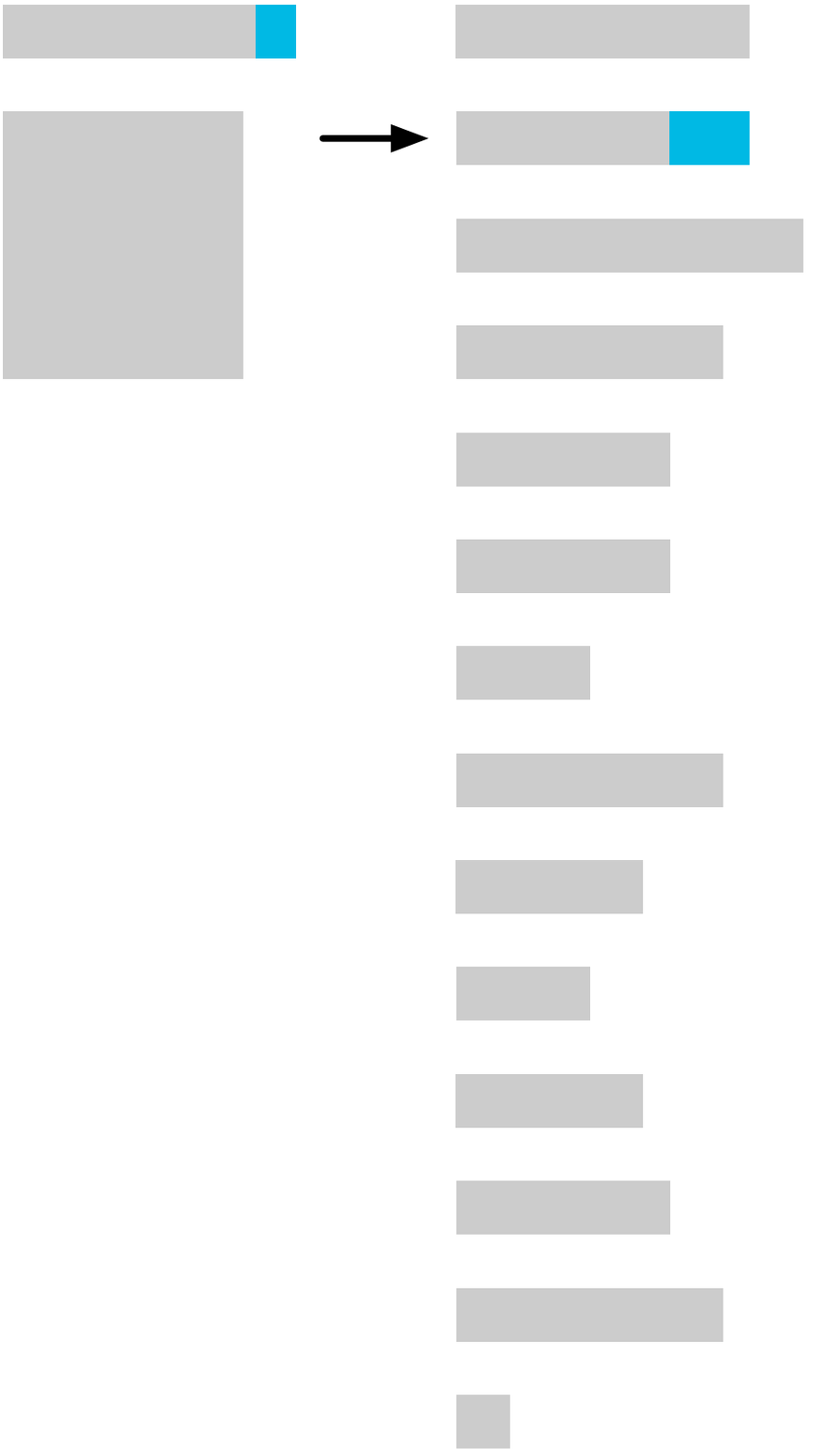
Mean and variance

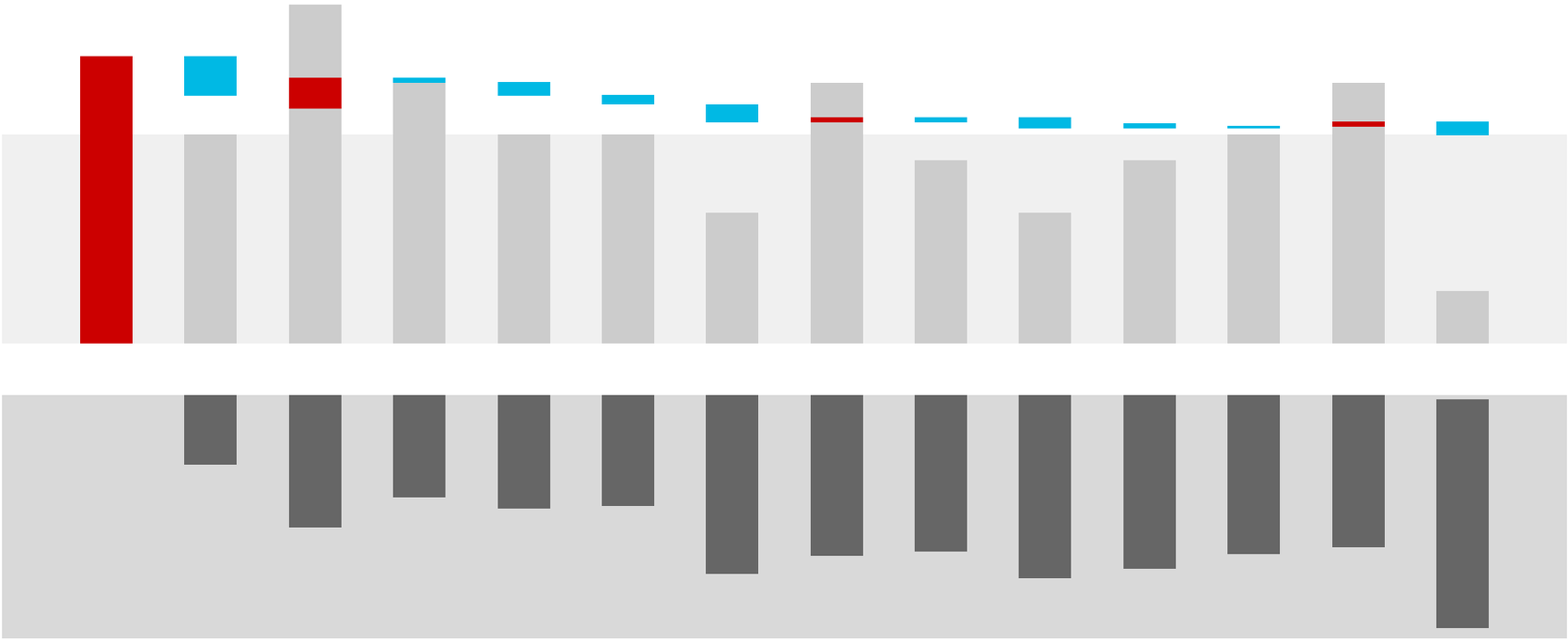
Textbook method

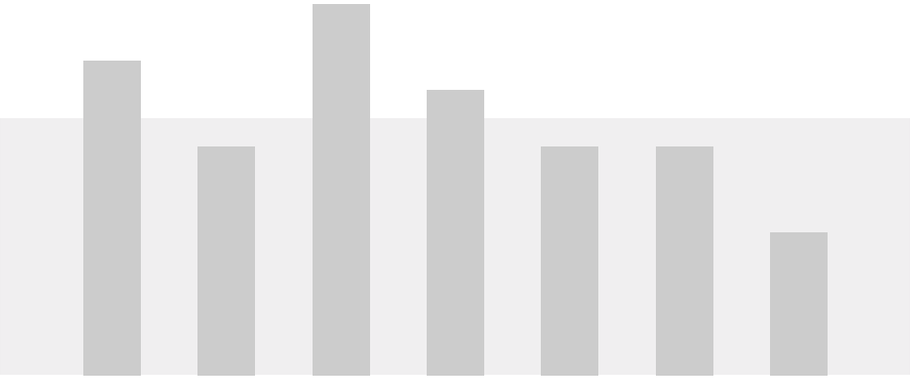


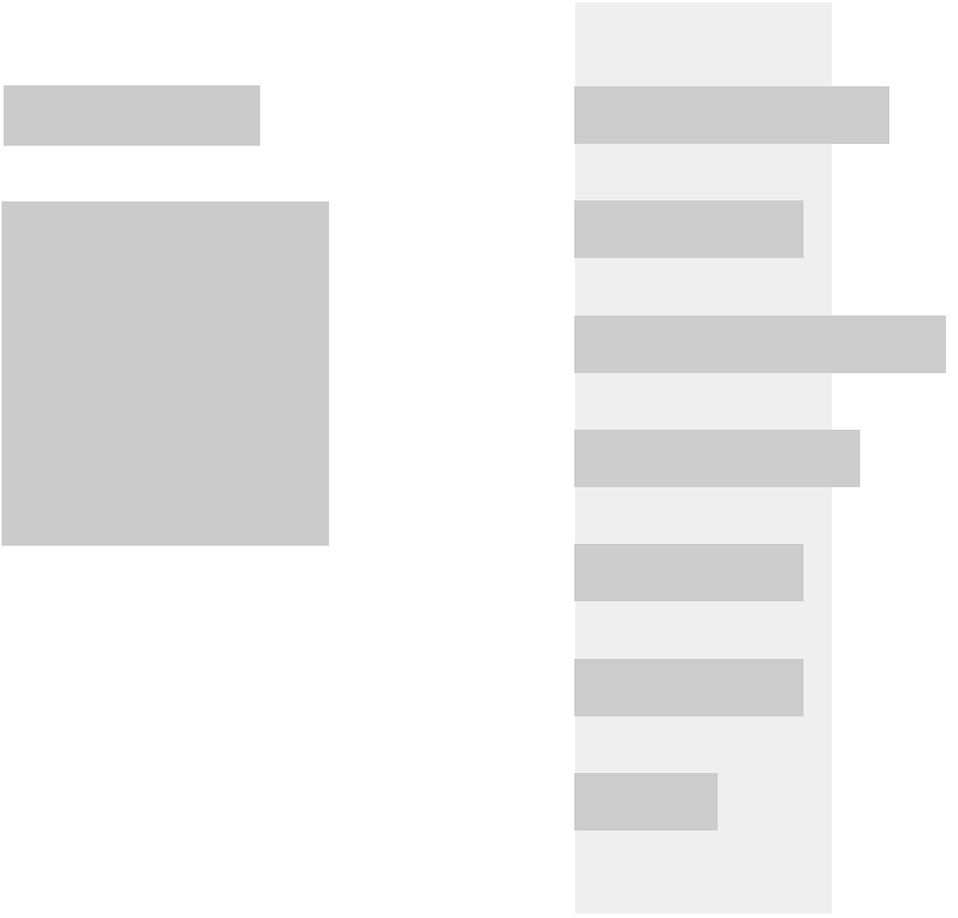


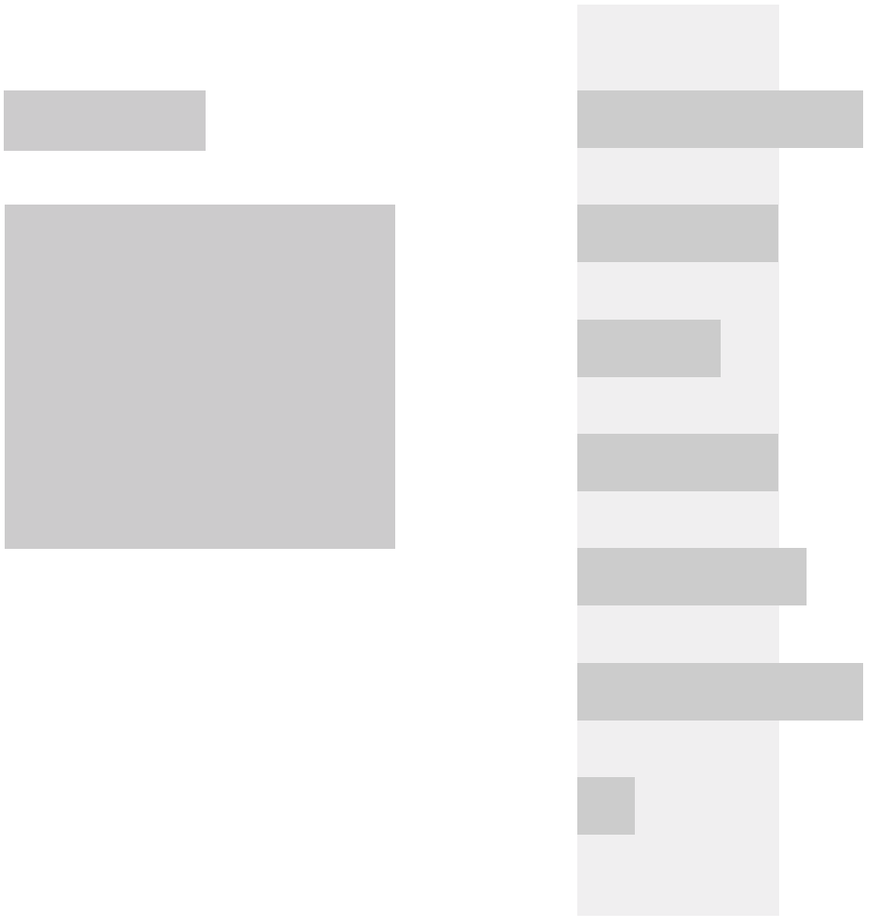
On-line estimates

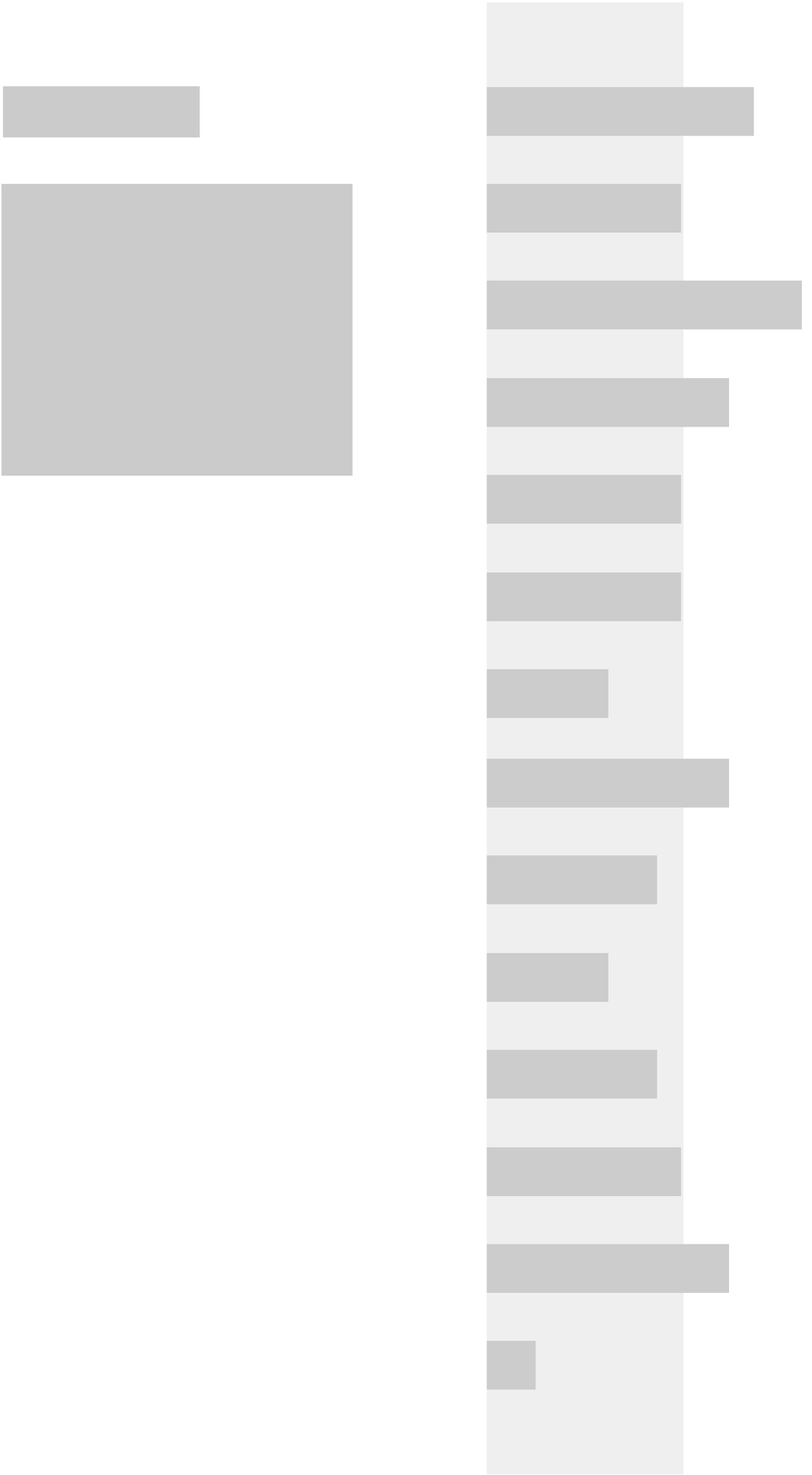


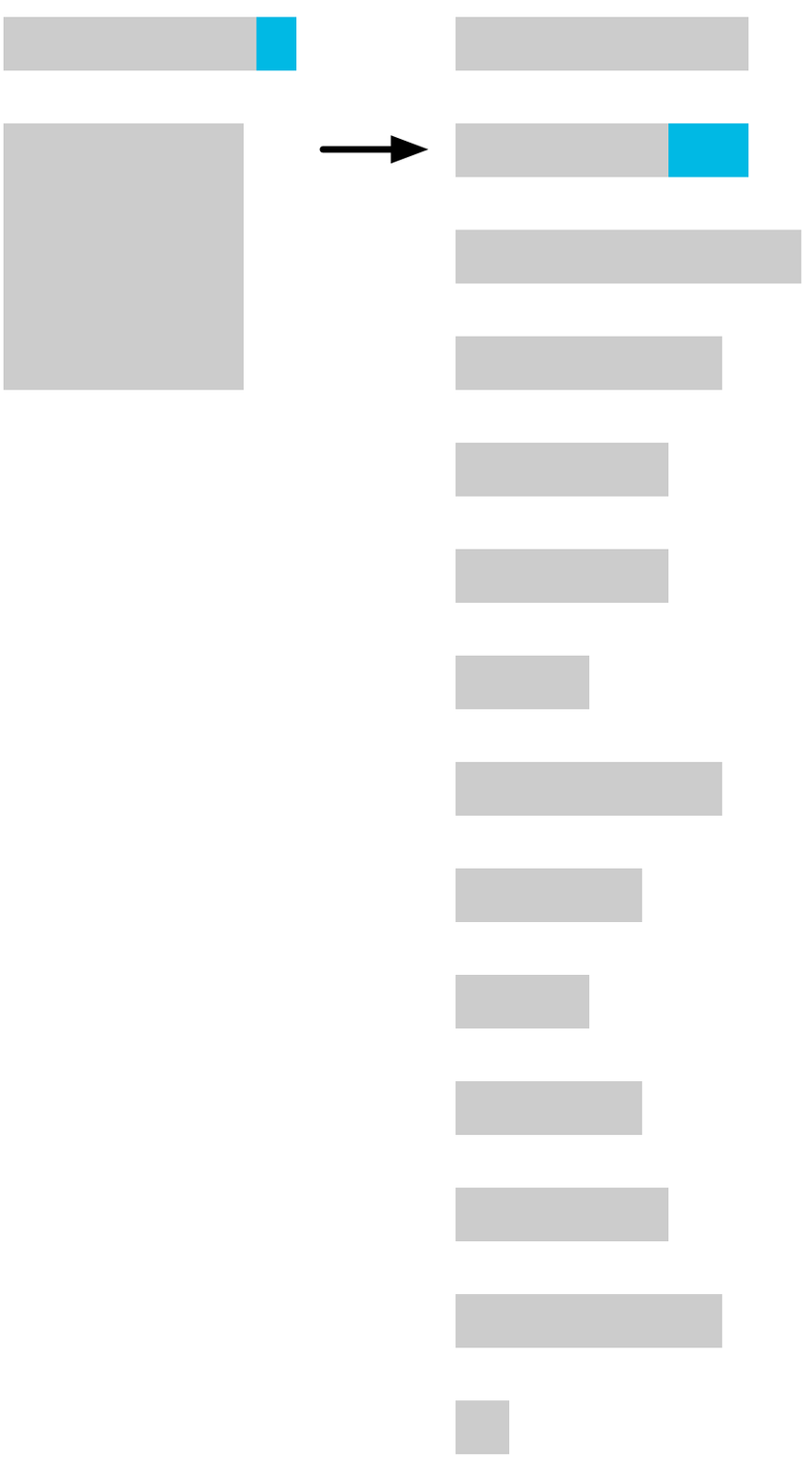


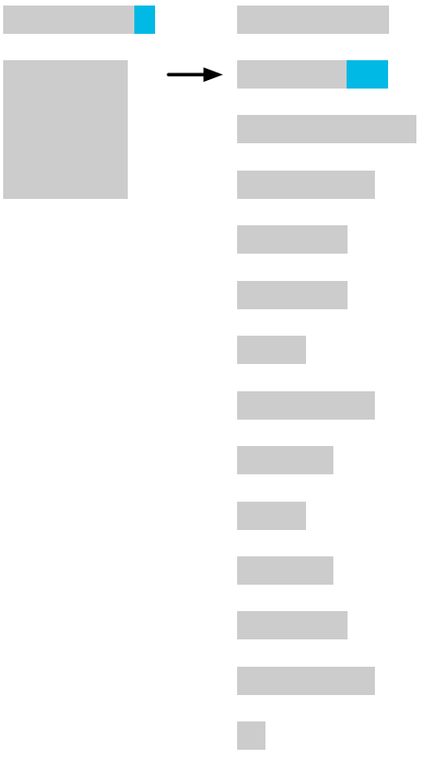


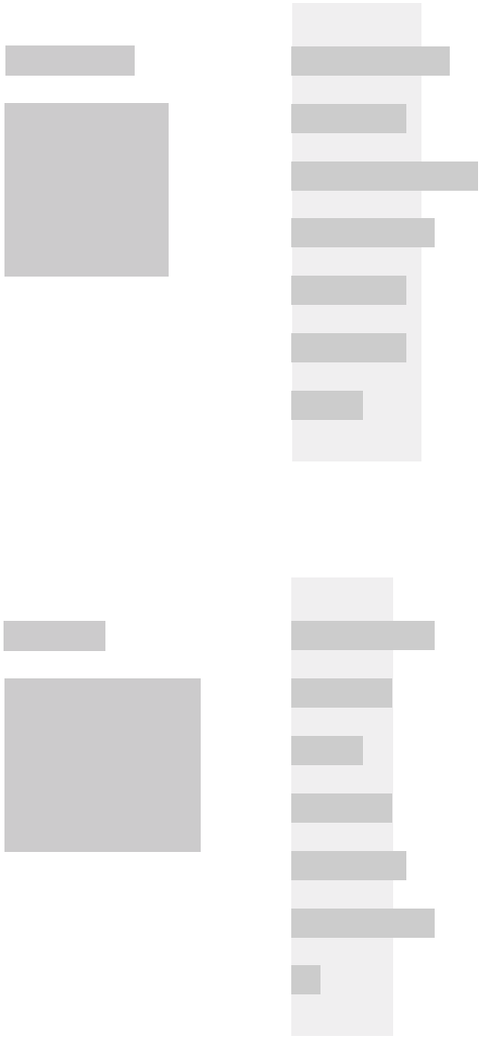


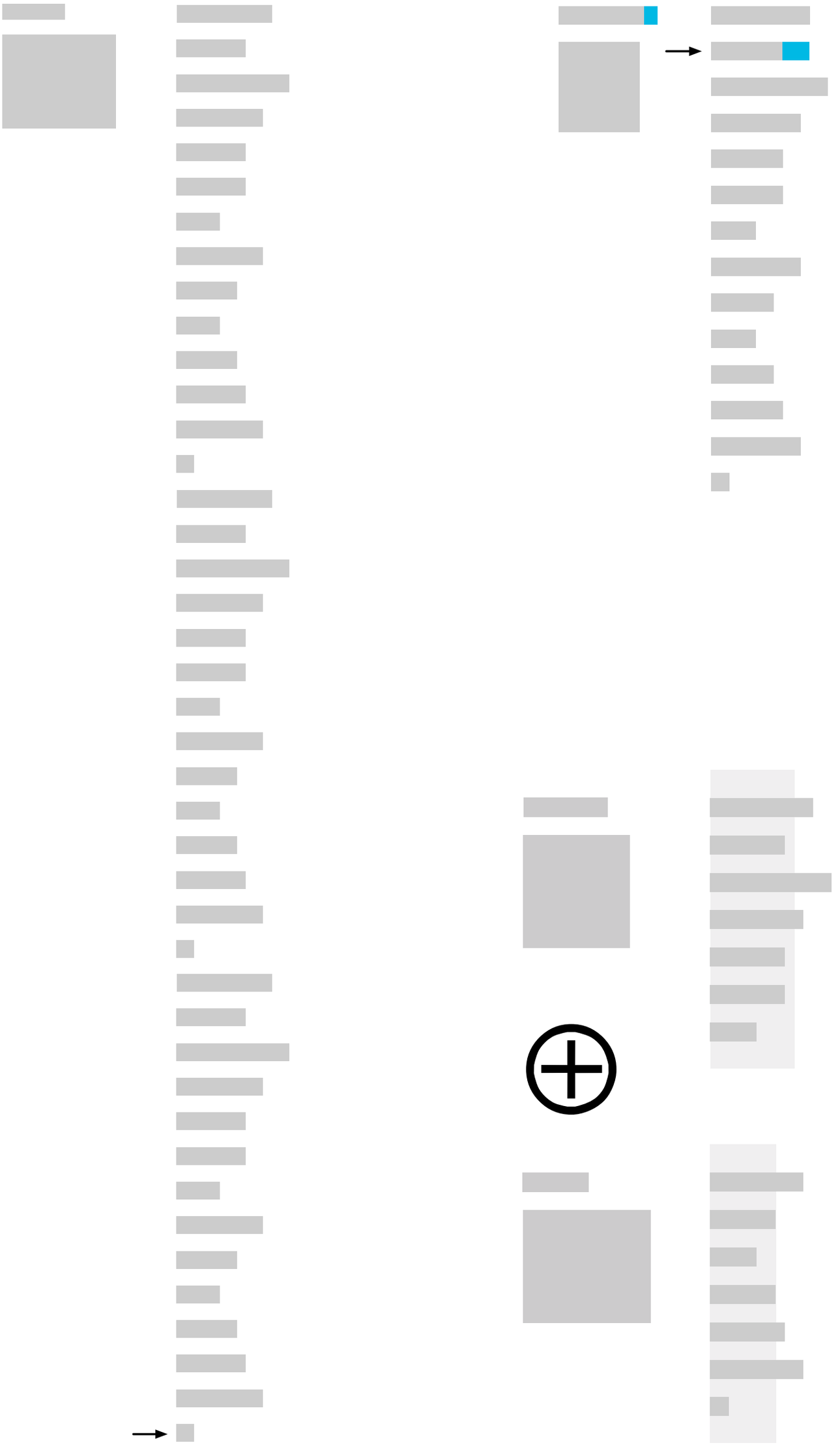






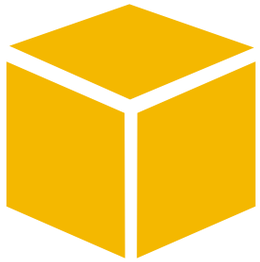


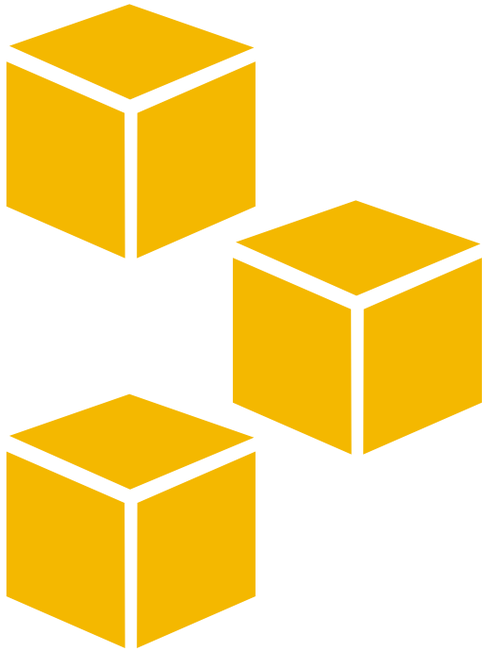


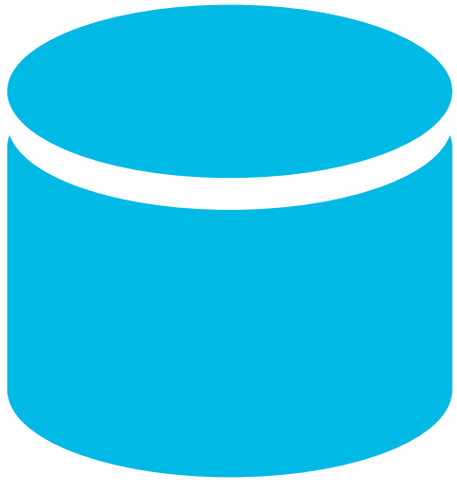


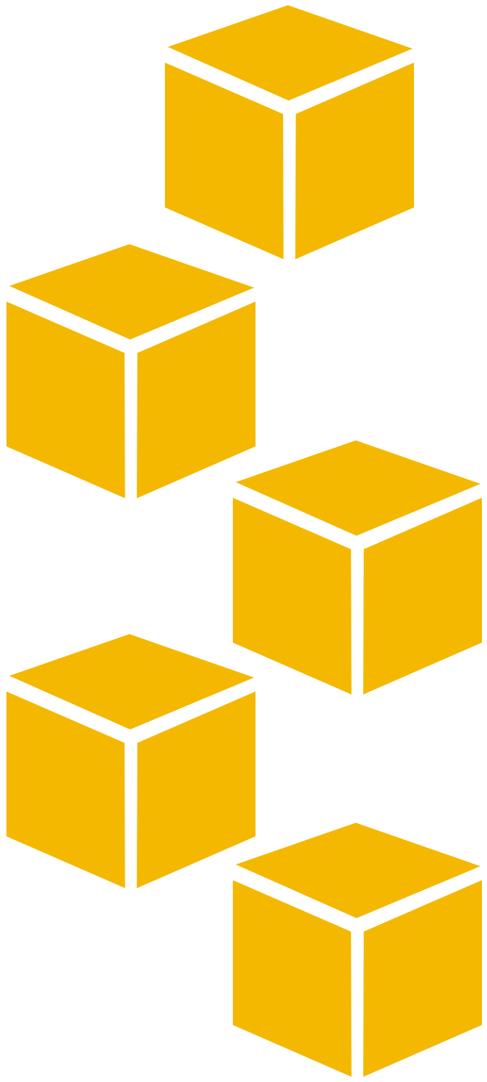
```python
class StreamMV(object):
    def __init__(self, count=0, m1=0.0, m2=0.0):
        (self.count, self.m1, self.m2) = (count, m1, m2)

    def __lshift__(self, sample):
        dev = sample - self.m1
        self.m1 = self.m1 + (dev / (self.count + 1))
        self.m2 = self.m2 + (dev * dev) * self.count / (self.count + 1)
        self.count += 1
        return self

    def mean(self): return self.m1

    def variance(self): return self.m2 / self.count

    def stddev(self): return math.sqrt(self.variance)
```

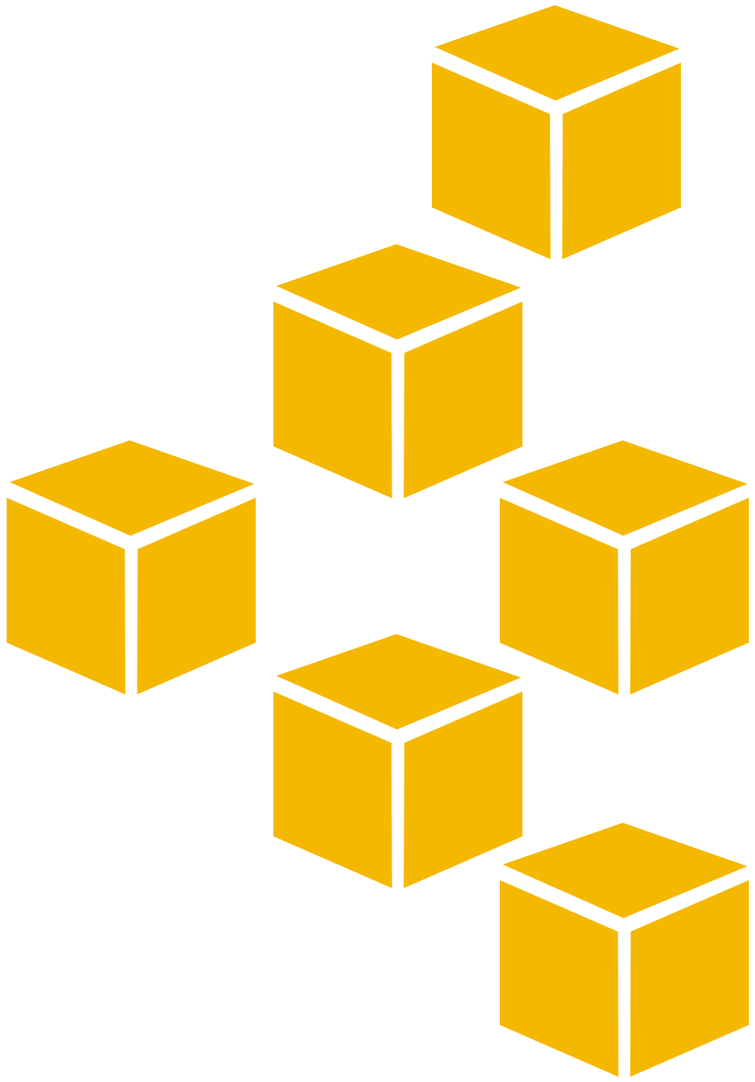
Set membership

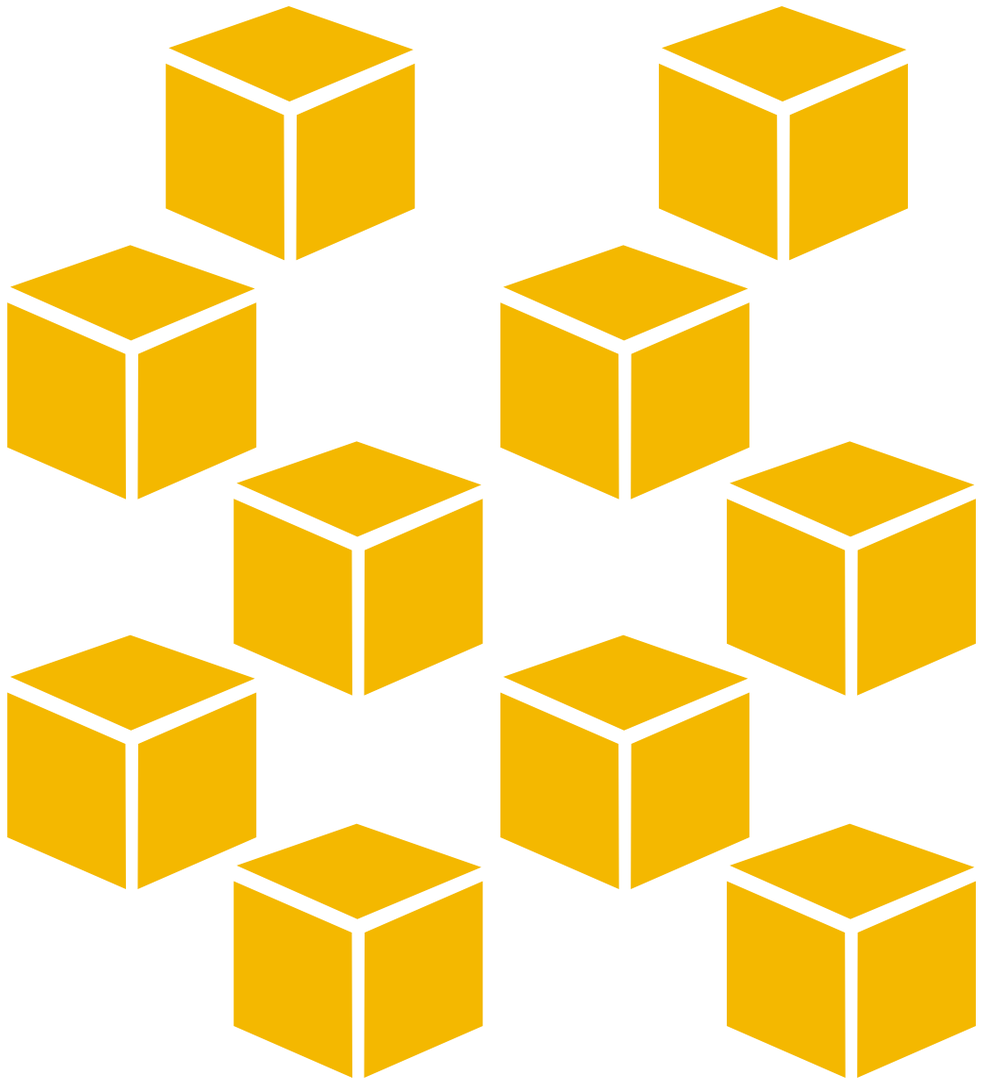


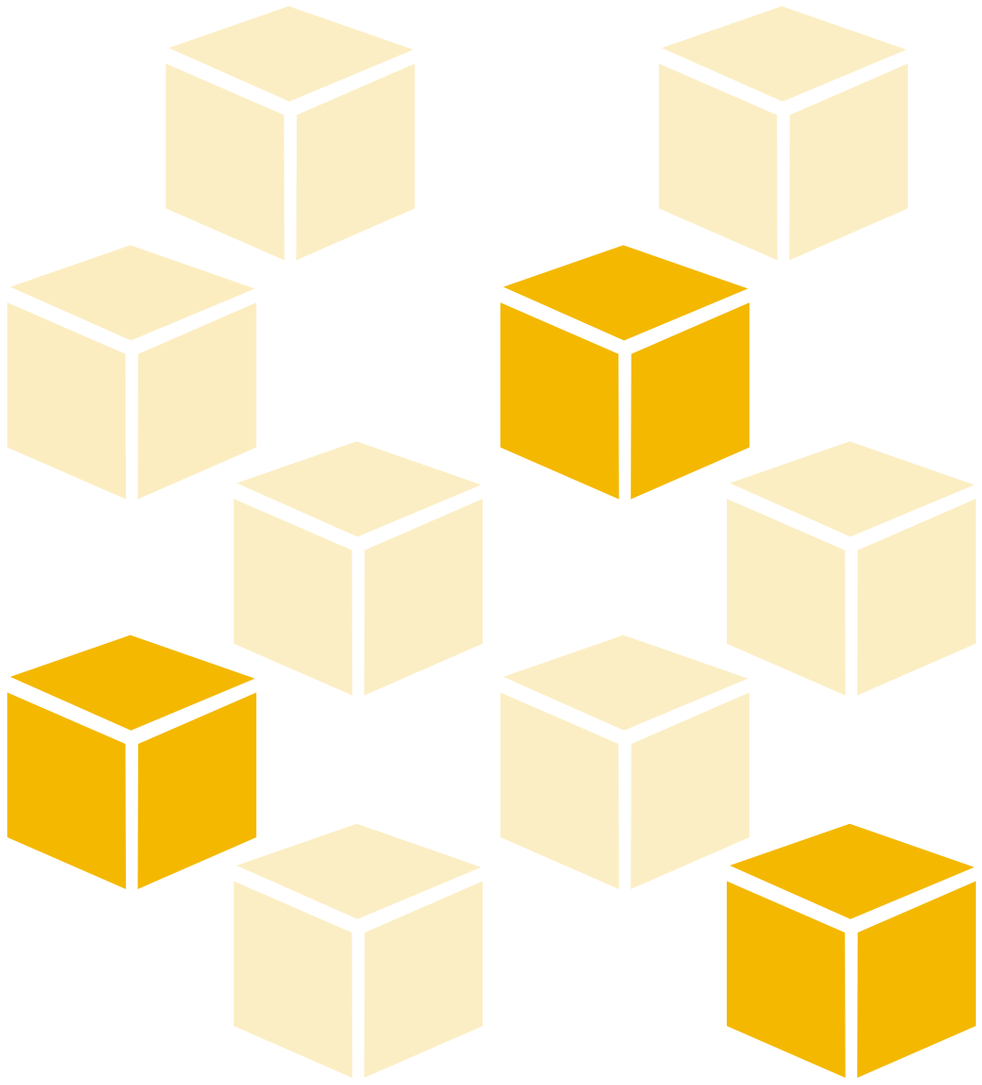




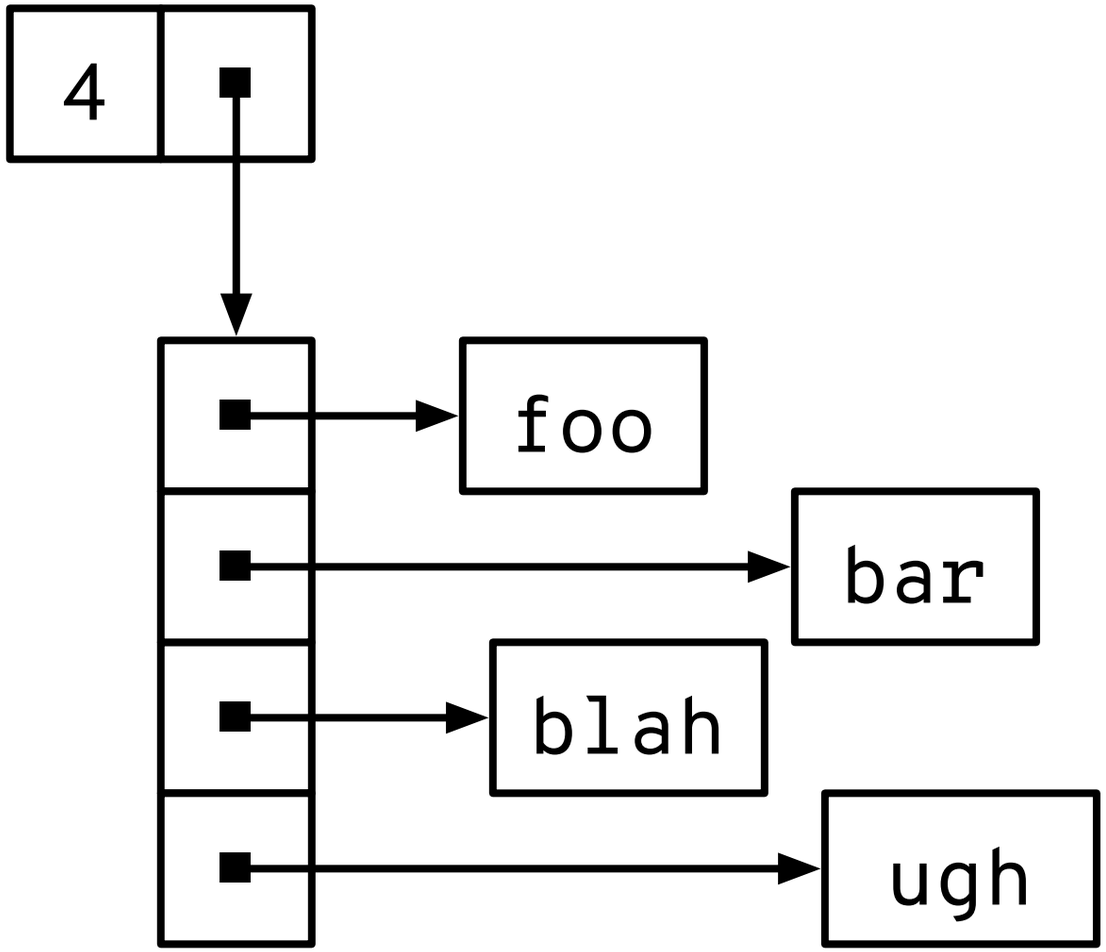


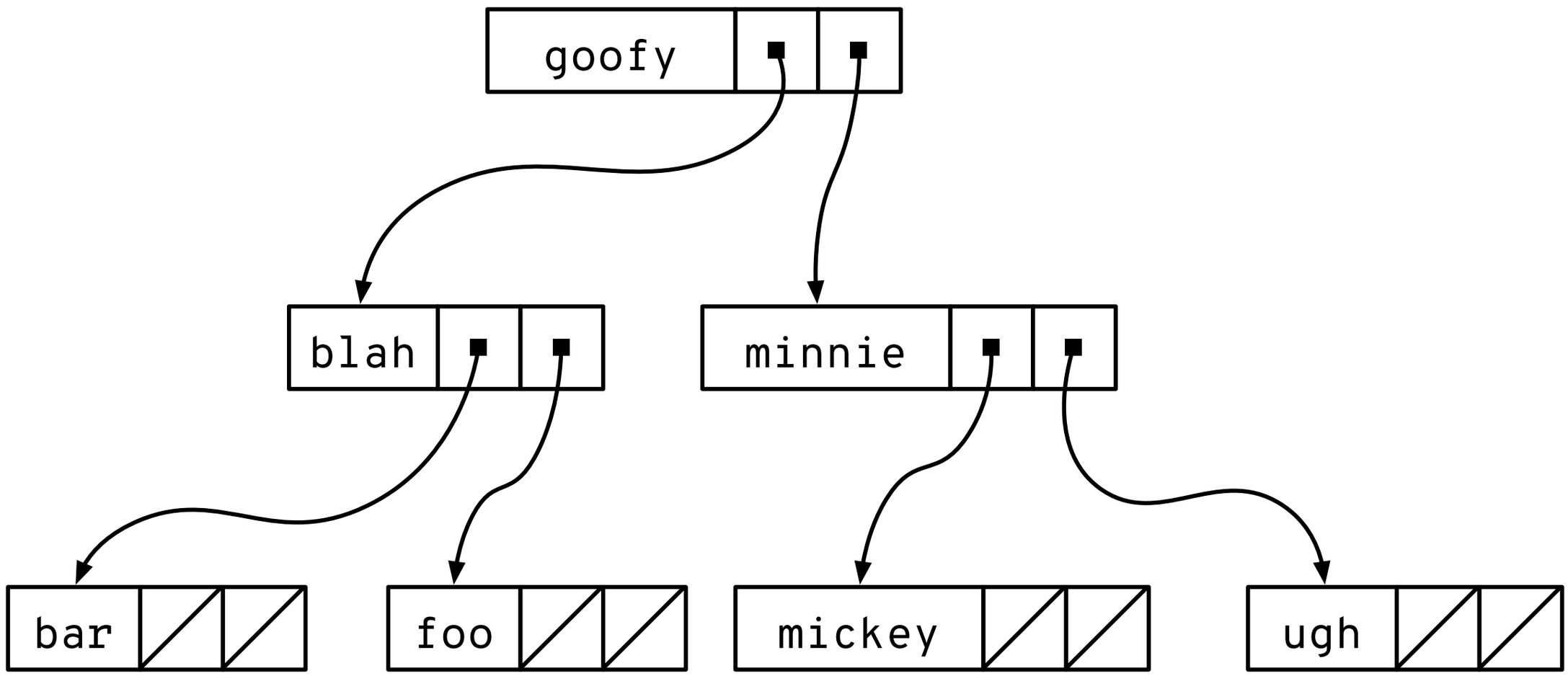


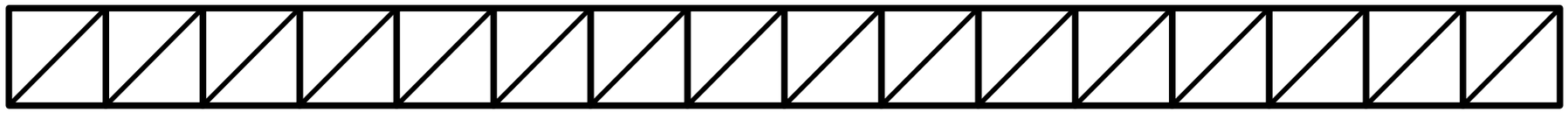


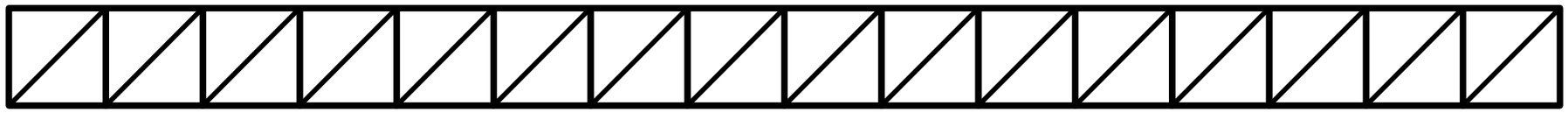


Precise structures

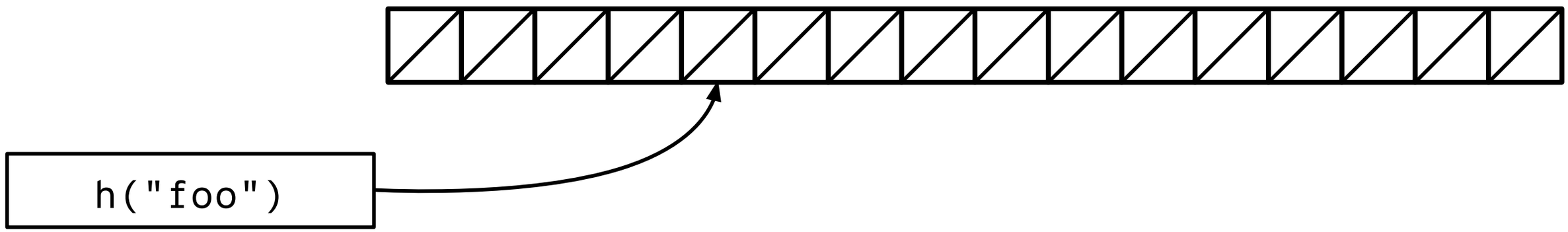

4
foo
bar
blah
ugh


goofy
blah
minnie
bar
foo
mickey
ugh

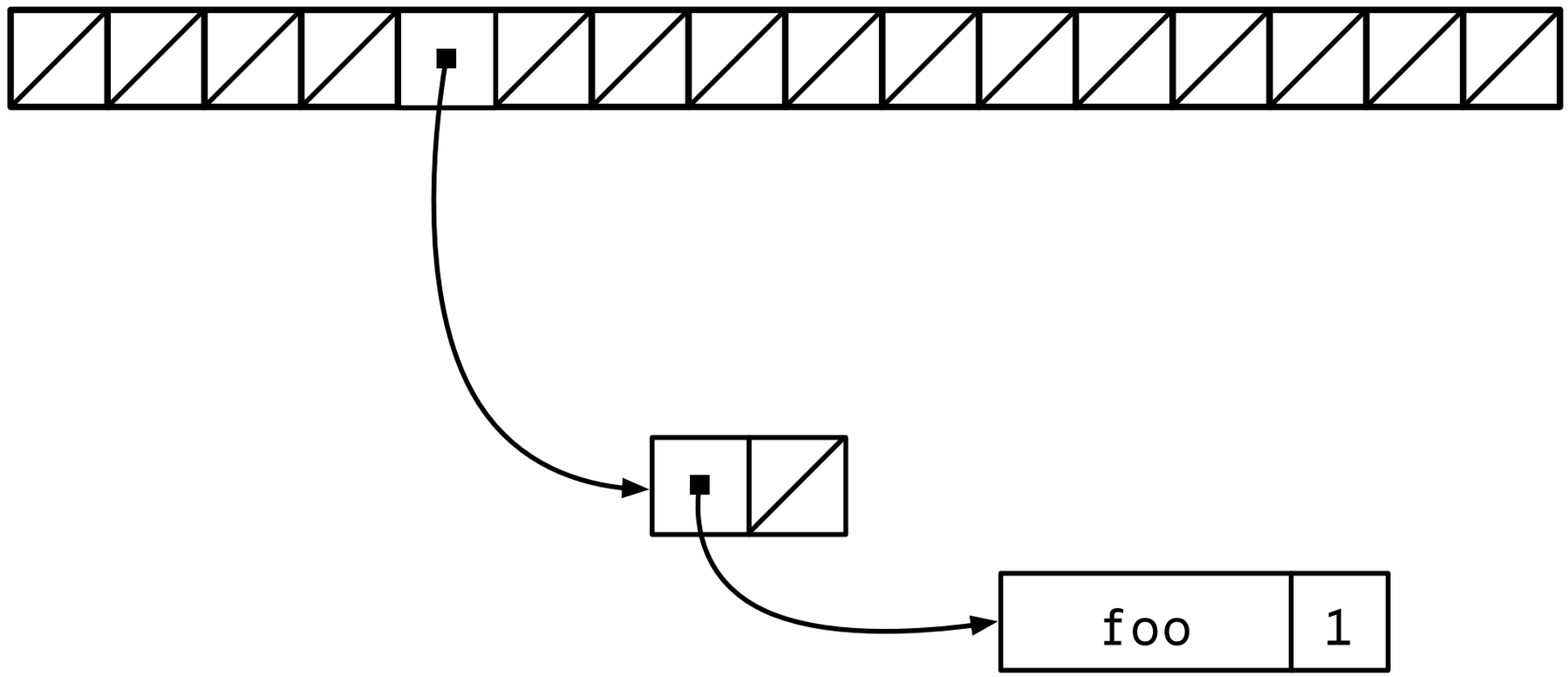




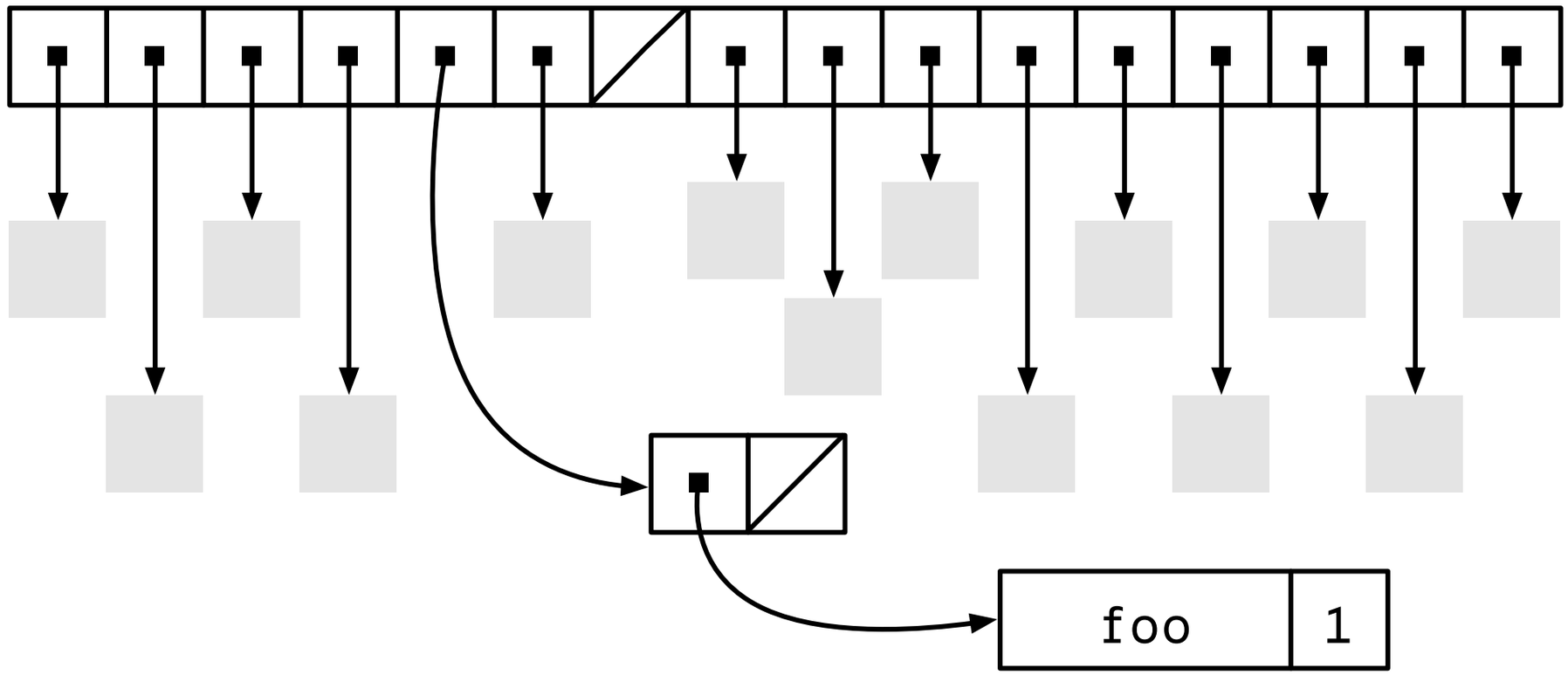
```
h.put("foo", 1)
```

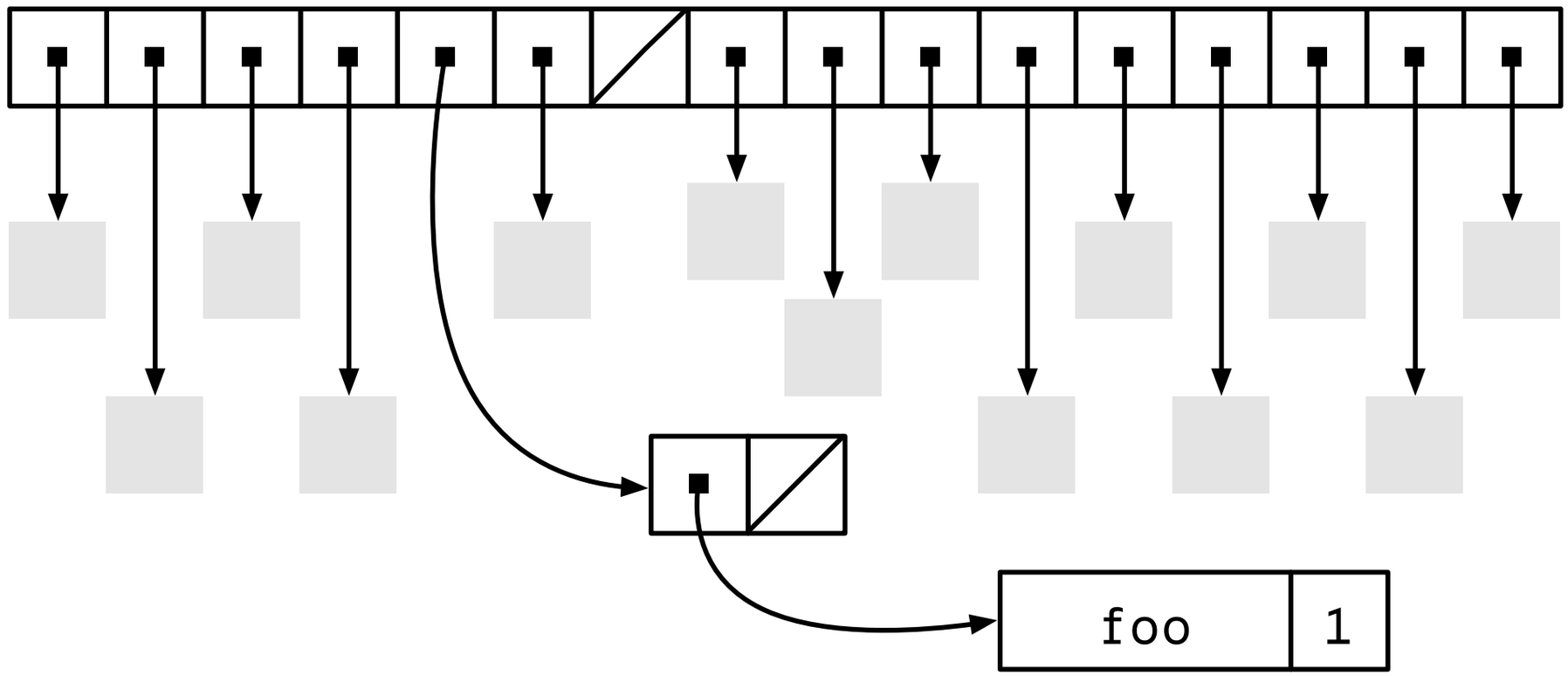



```
h.put("foo", 1)
```

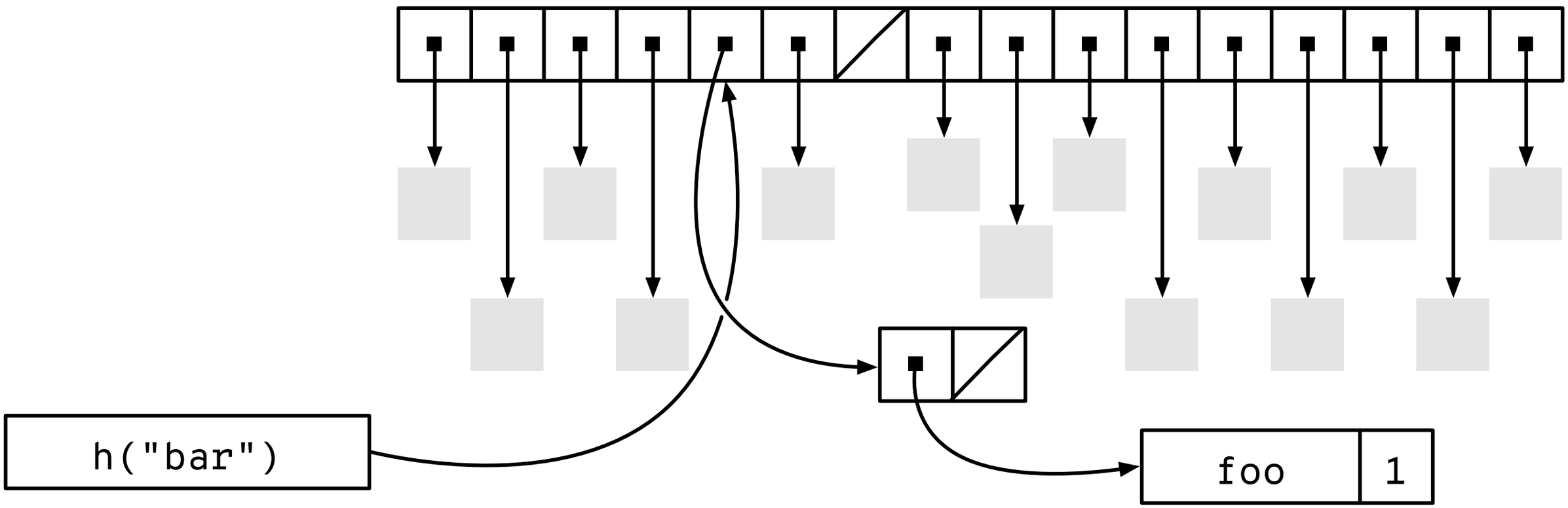



```
h.put("foo", 1)
```

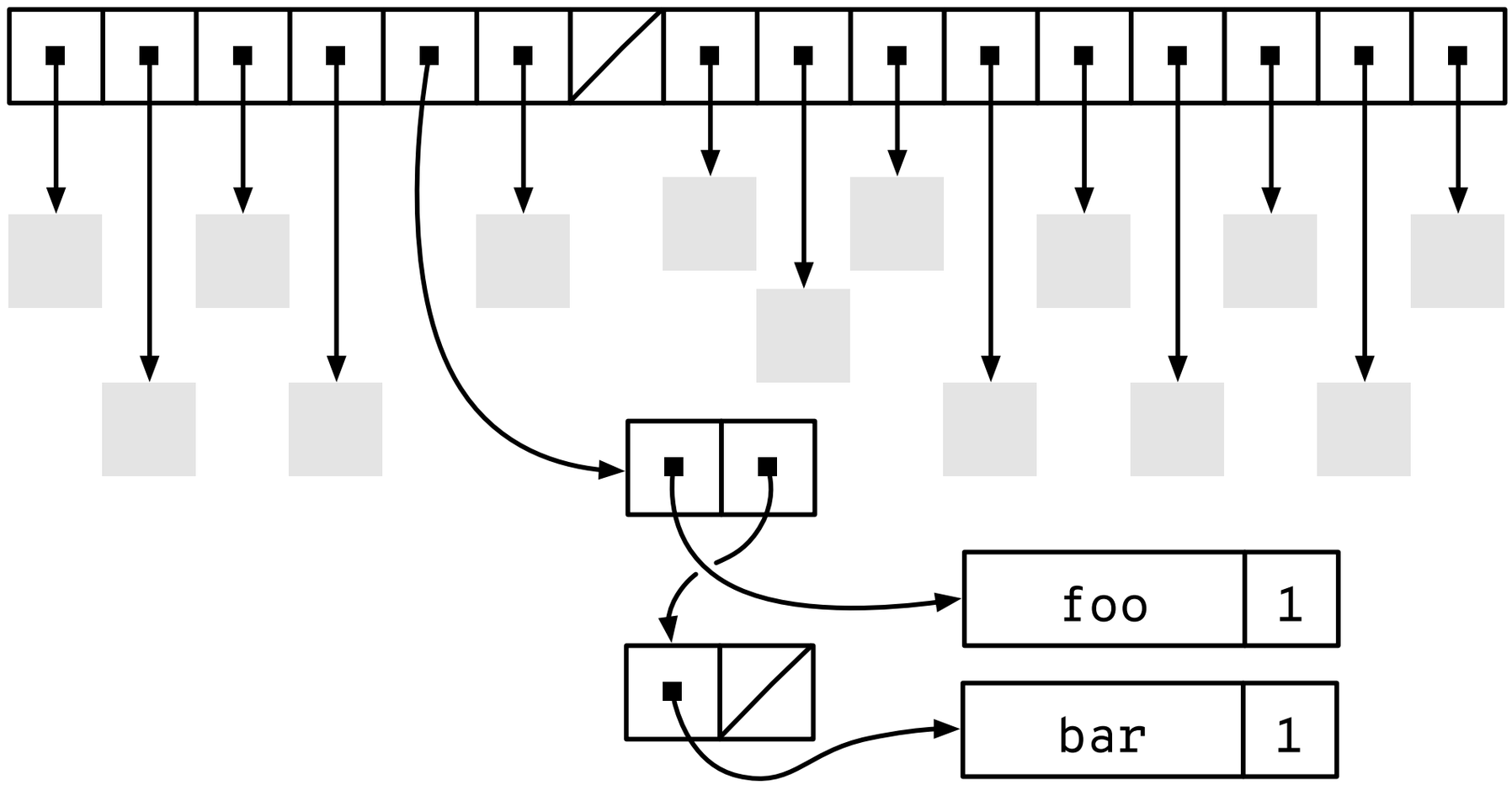

foo
1




```
h.put("bar", 1)
```

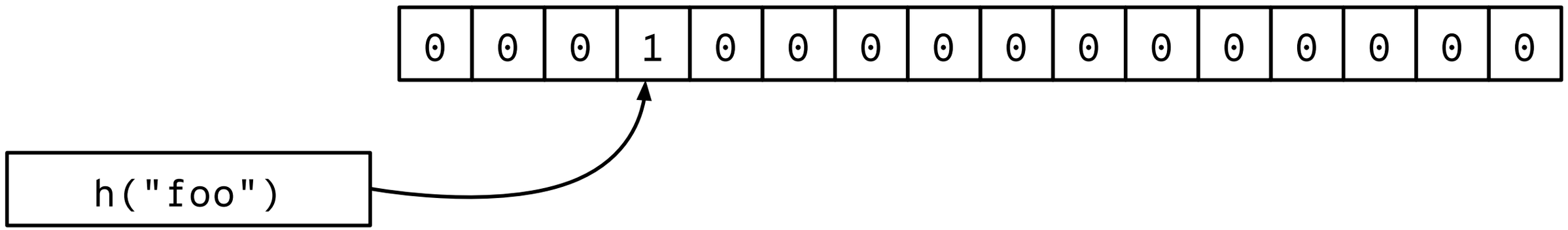


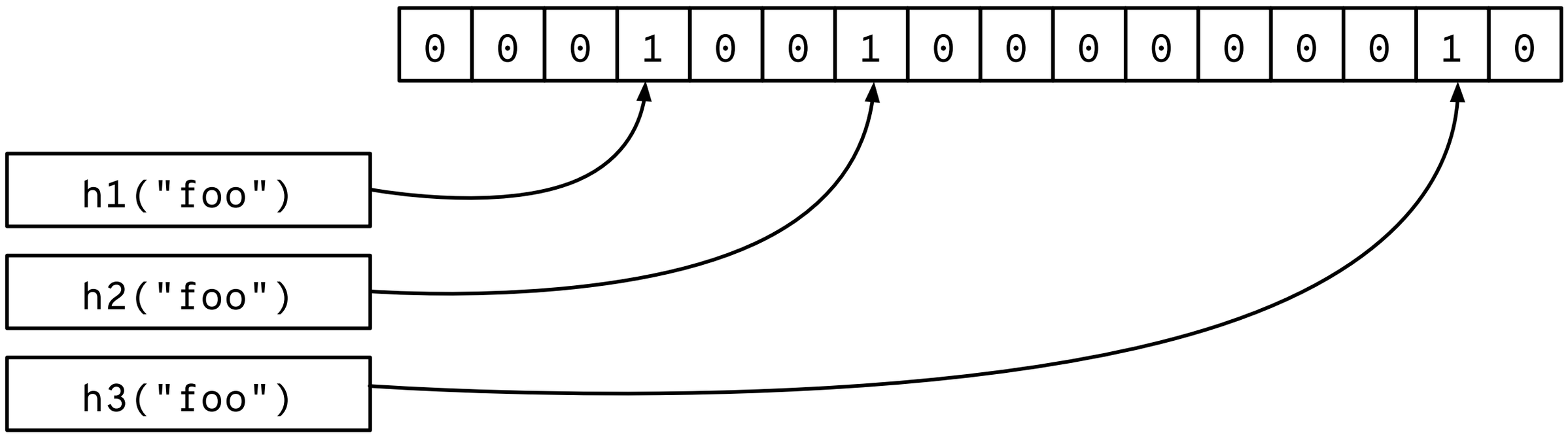
h.put("bar", 1)

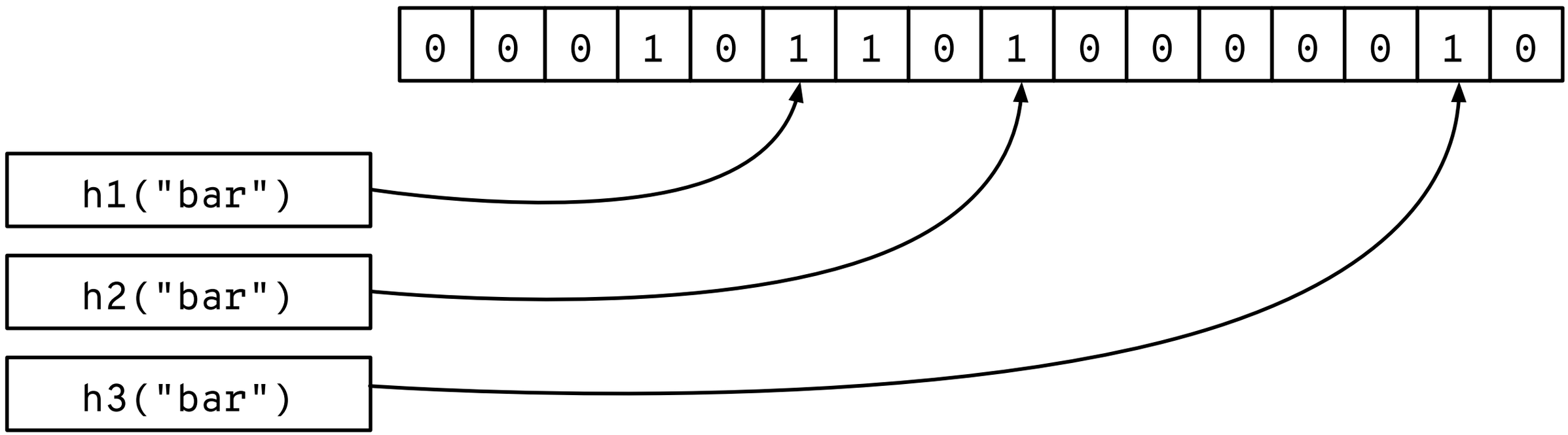



```
h.put("bar", 1)
```

Bloom filters

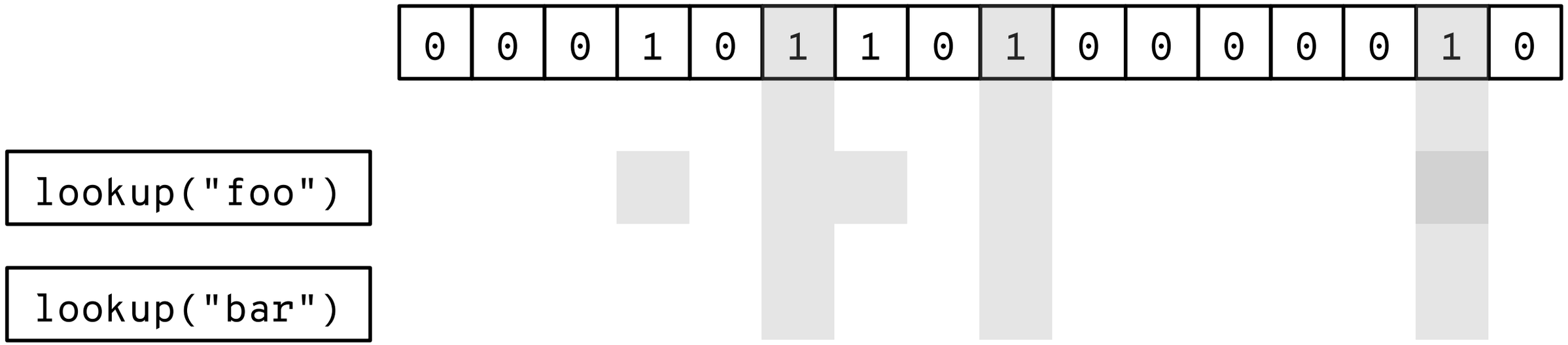

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
h("foo")

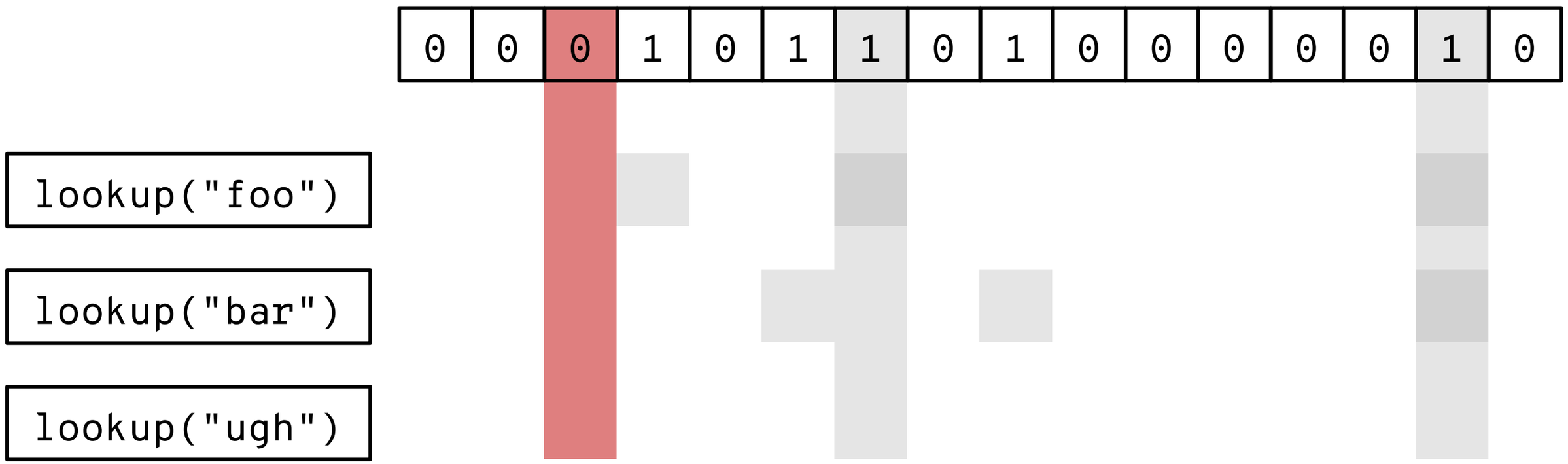

0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
h1("foo")
h2("foo")
h3("foo")

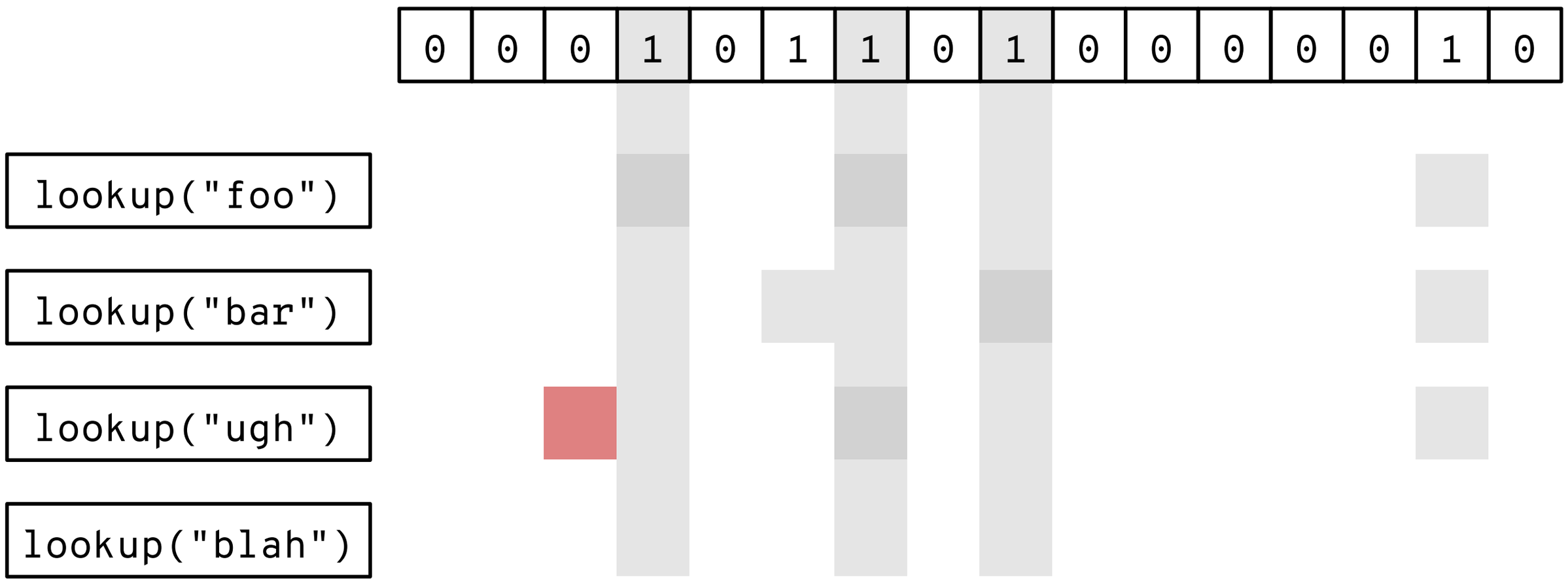

0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
h1("bar")
h2("bar")
h3("bar")

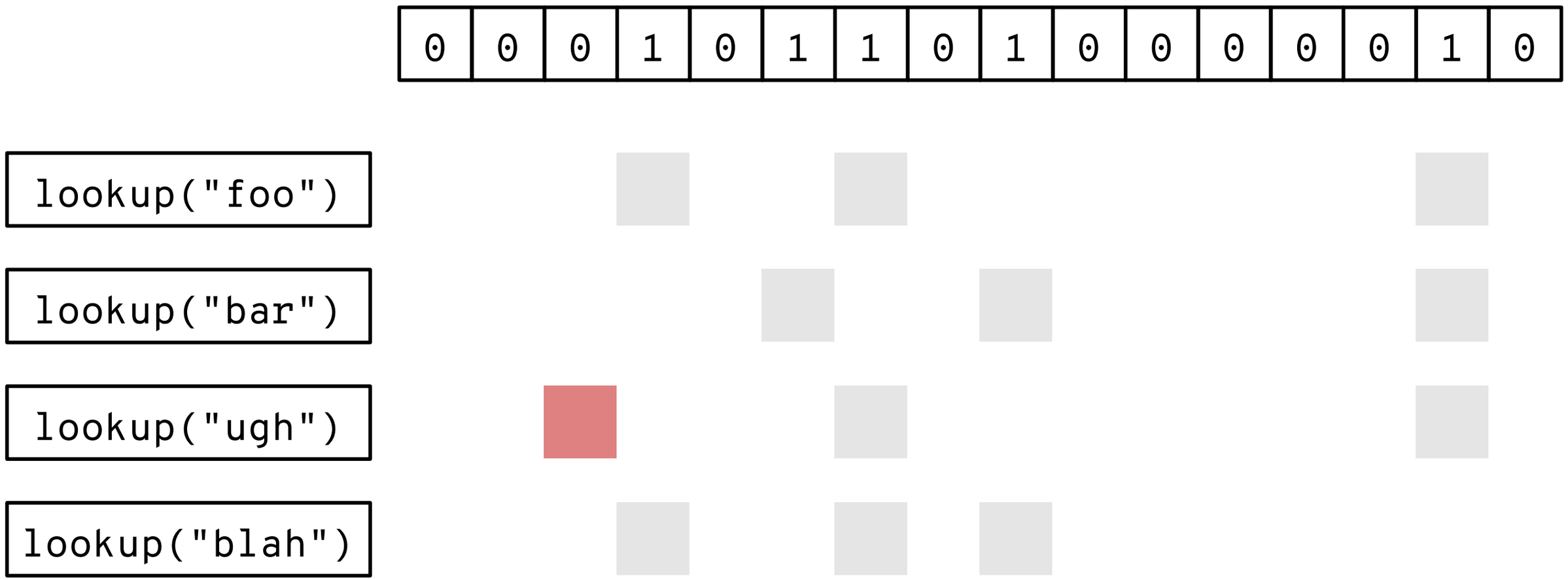
0	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0

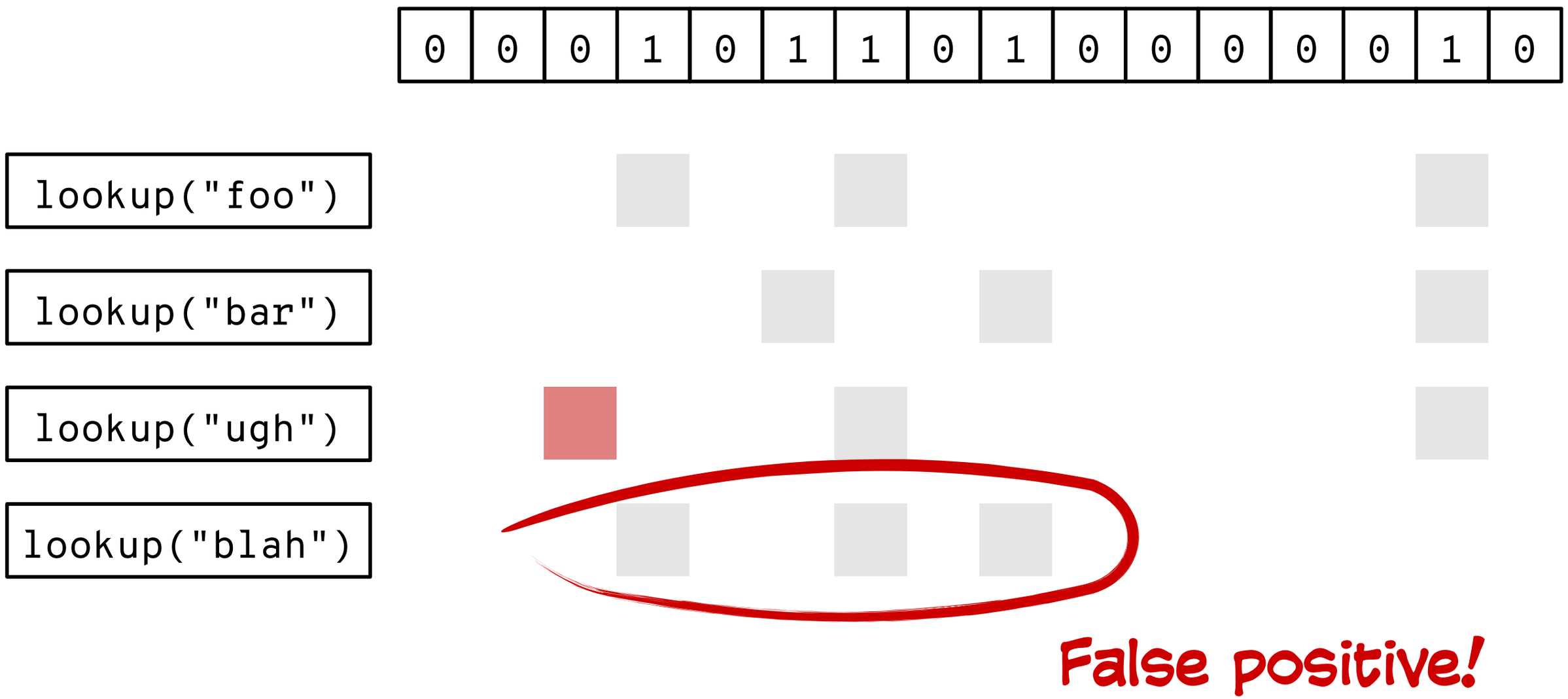
```
lookup("foo")
```


0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")

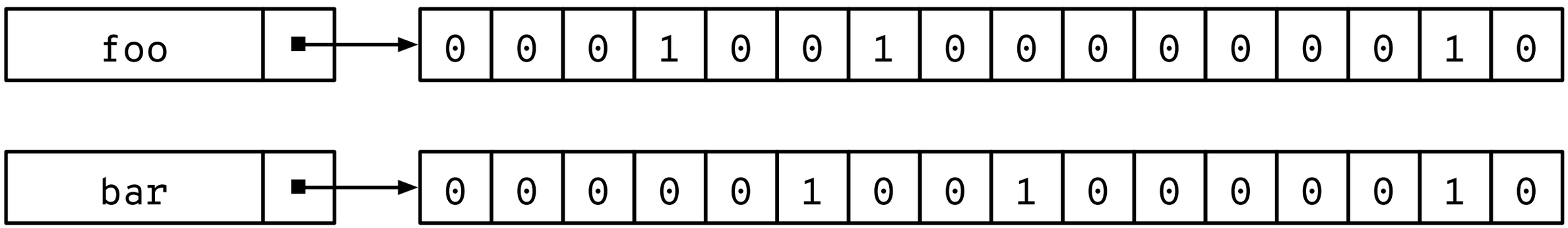

0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")
lookup("ugh")

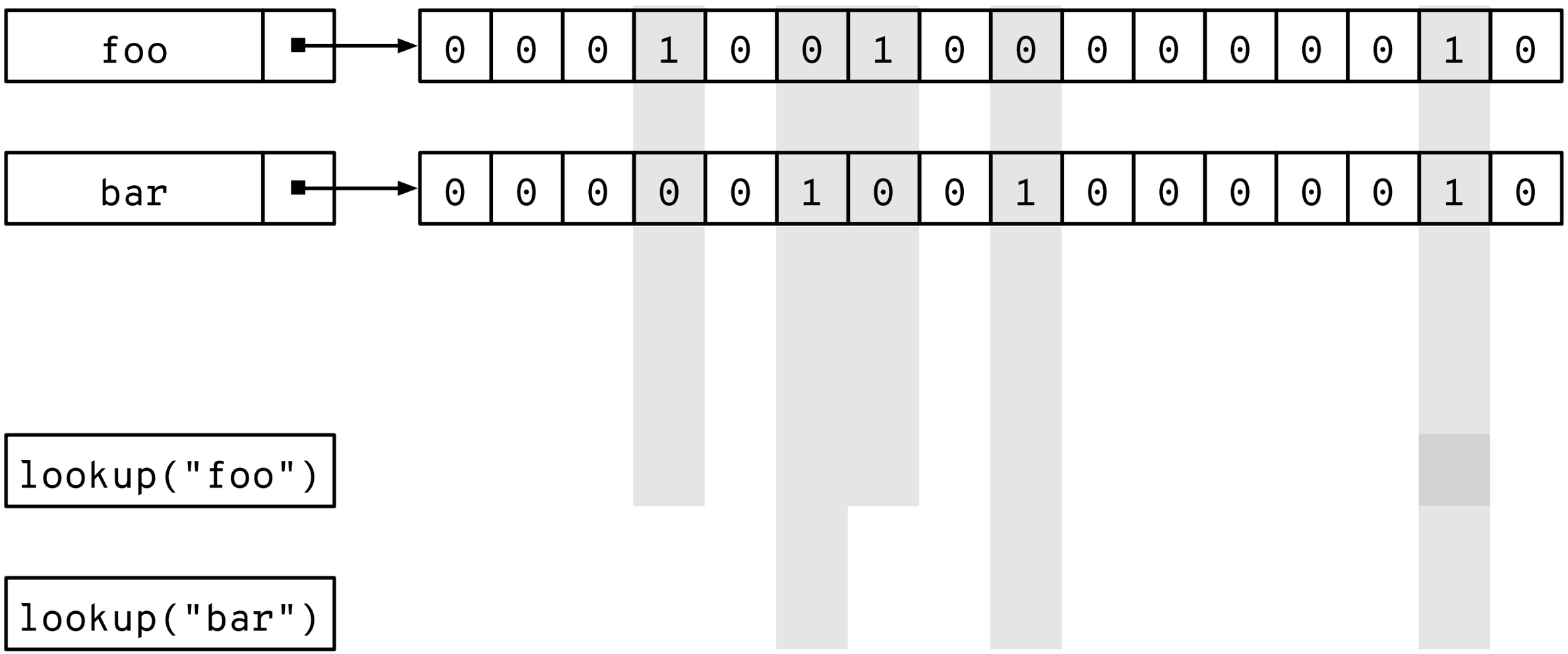

0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")
lookup("ugh")
lookup("blah")

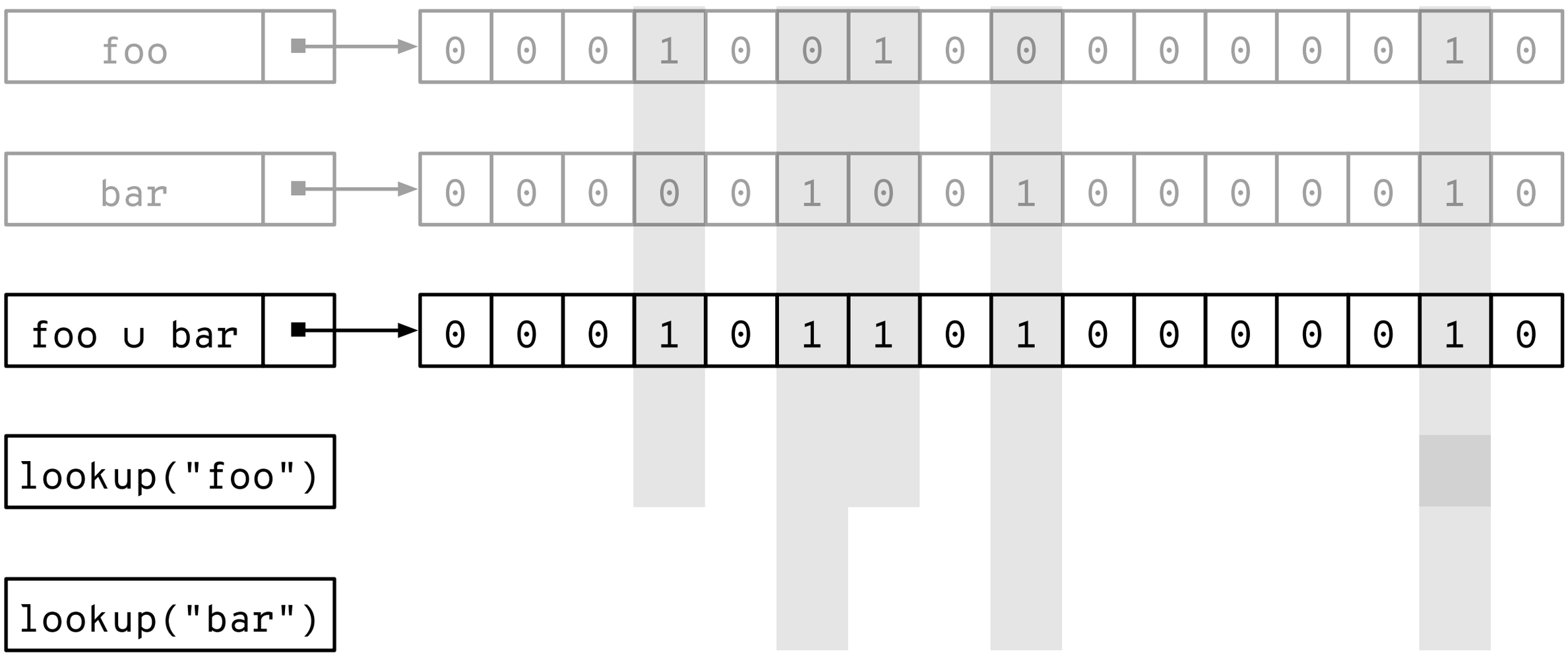

0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")
lookup("ugh")
lookup("blah")

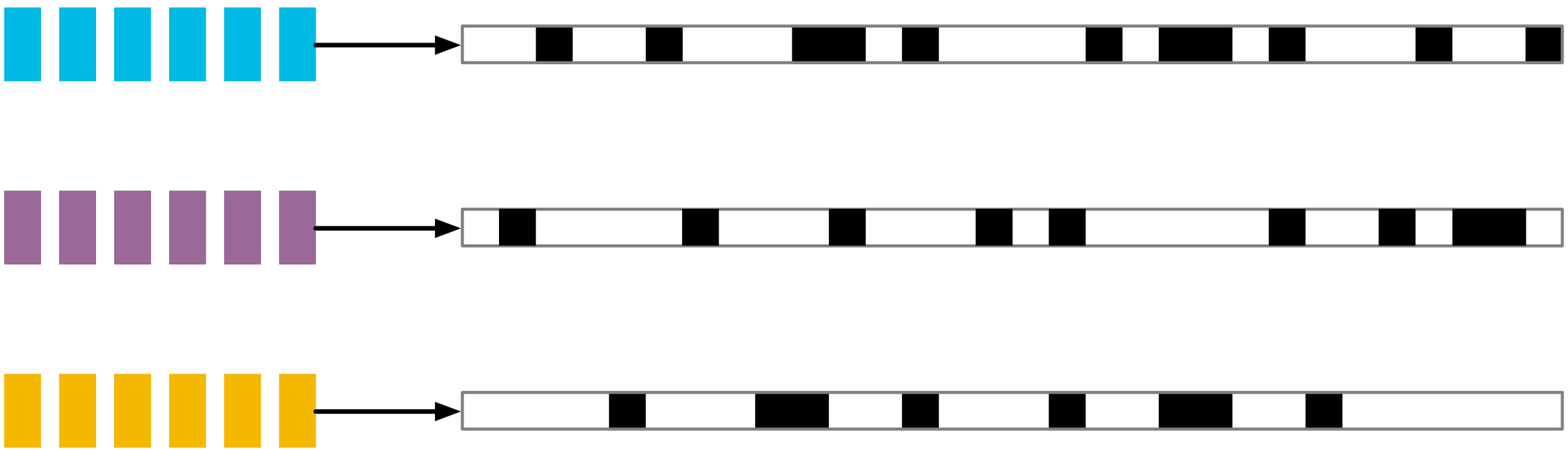

0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")
lookup("ugh")
lookup("blah")
False positive!

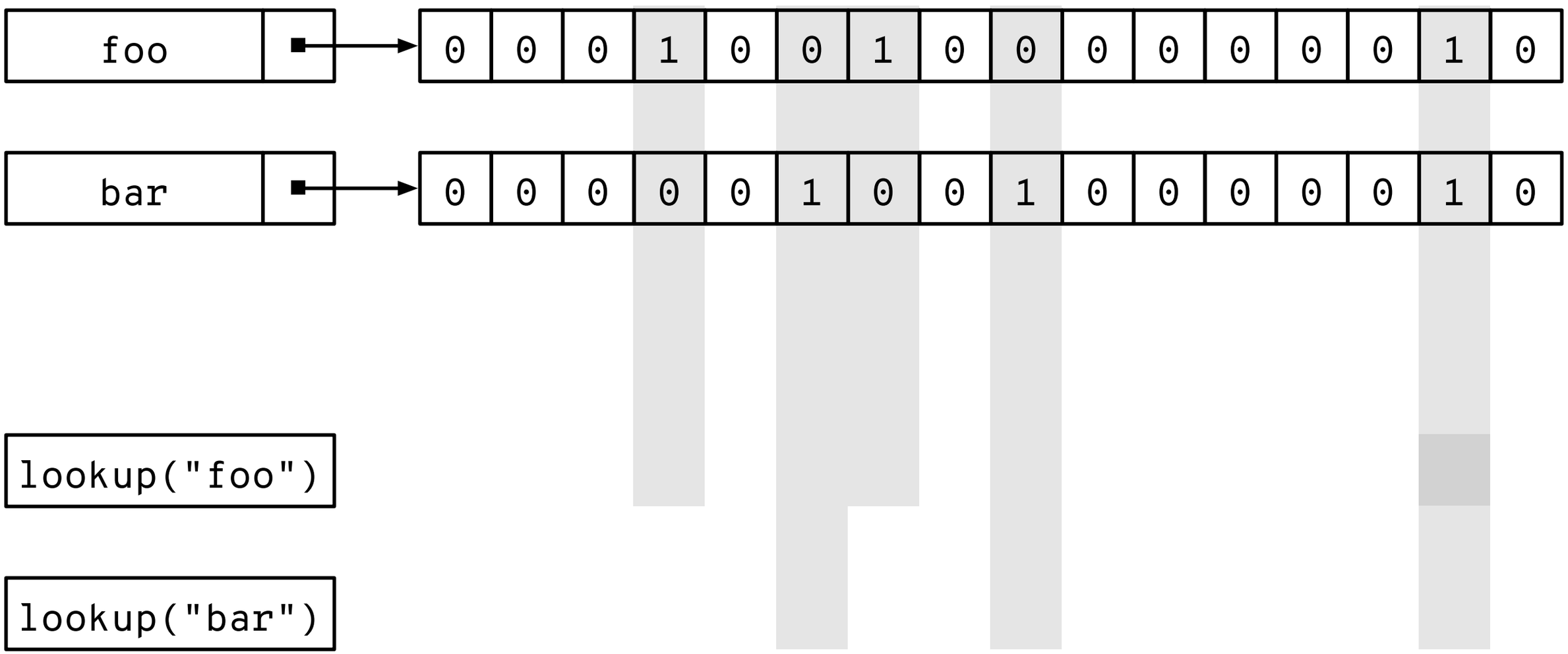
```python
class Bloom(object):
    def __init__(self, size, hashes):
        self.__buckets = BitVector(size)
        self.__size = len(self.__buckets)
        self.__hashes = lambda v: [f(v) for f in hashes[:]]

    def insert(self, value):
        for h in self.__hashes(value):
            self.__buckets[h % self.__size] = True

    def lookup(self, value):
        for h in self.__hashes(value):
            if self.__buckets[h % self.__size] == False:
                return False
        return True
```

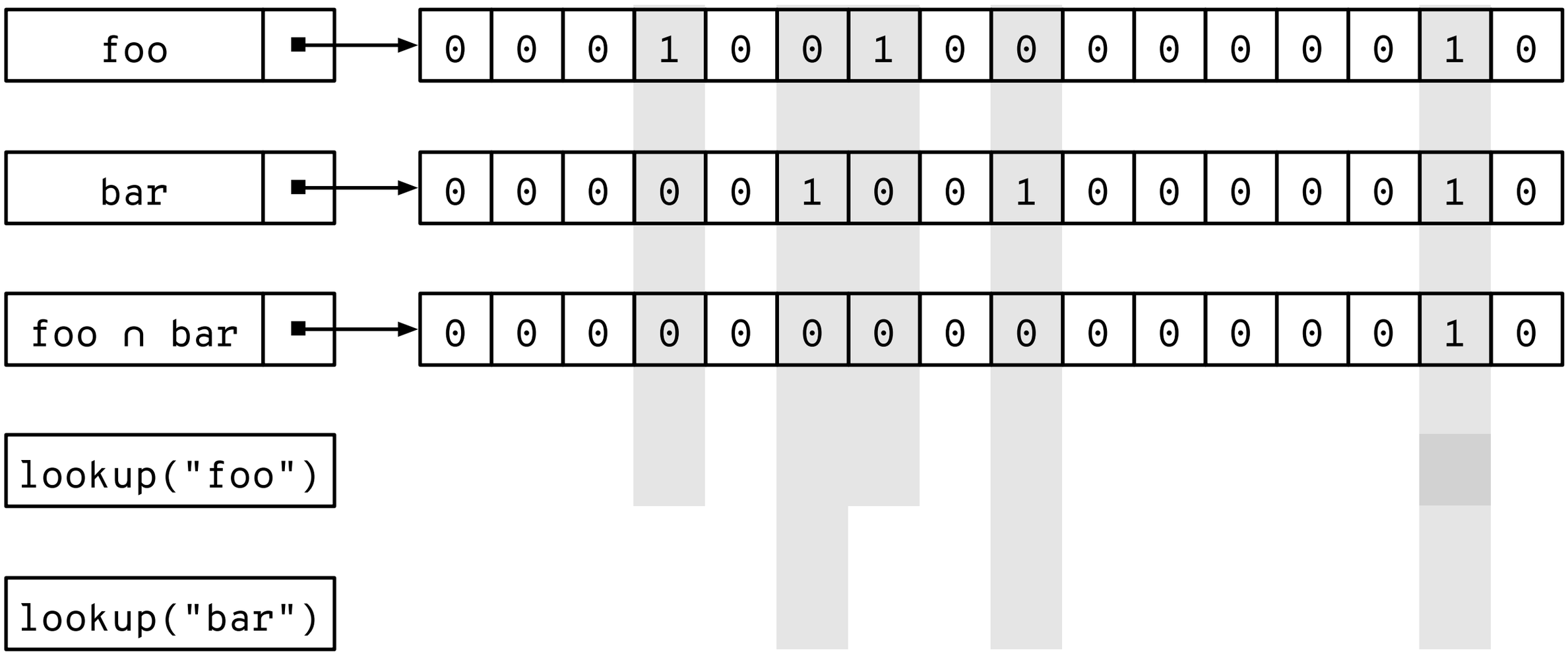

foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0

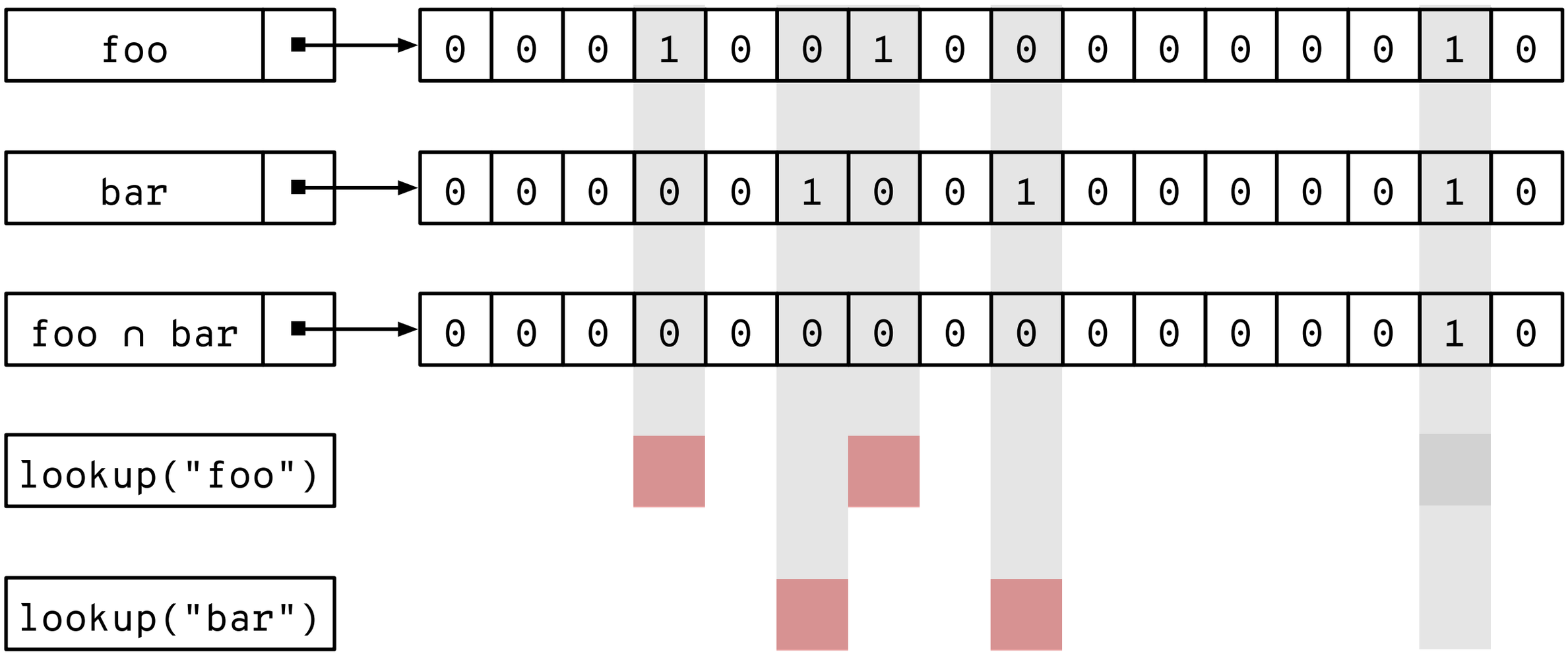

foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")


foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
foo ∪ bar
0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")

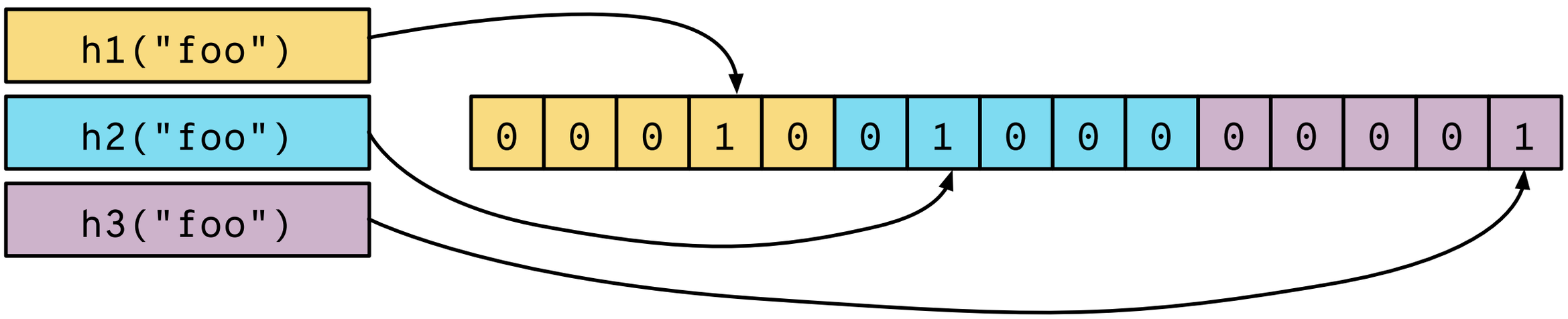



foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")


foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
foo ∩ bar
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")


foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
foo ∩ bar
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")

Partitioned filters


h1("foo")
h2("foo")
h3("foo")
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1

h1("foo")

h2("foo")

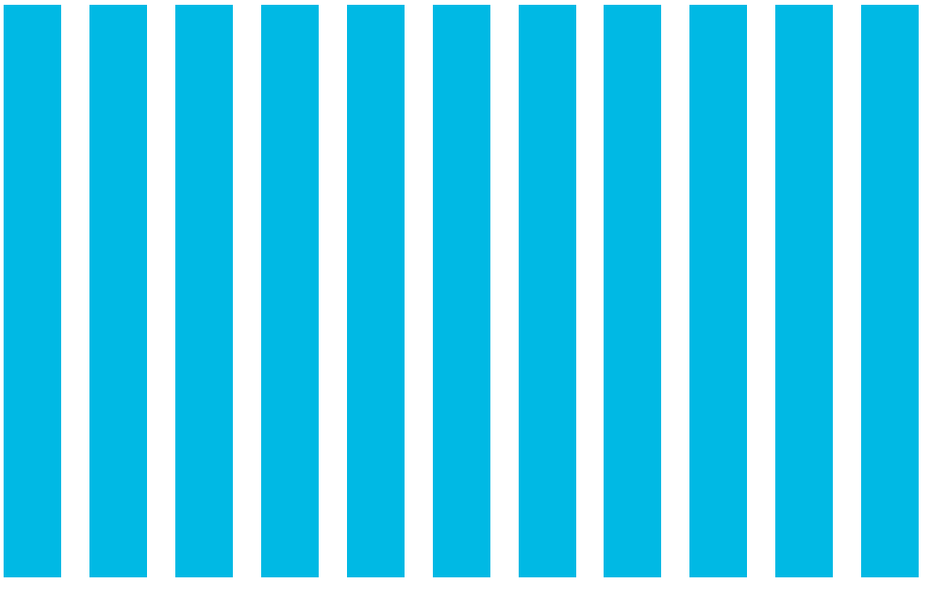
h3("foo")

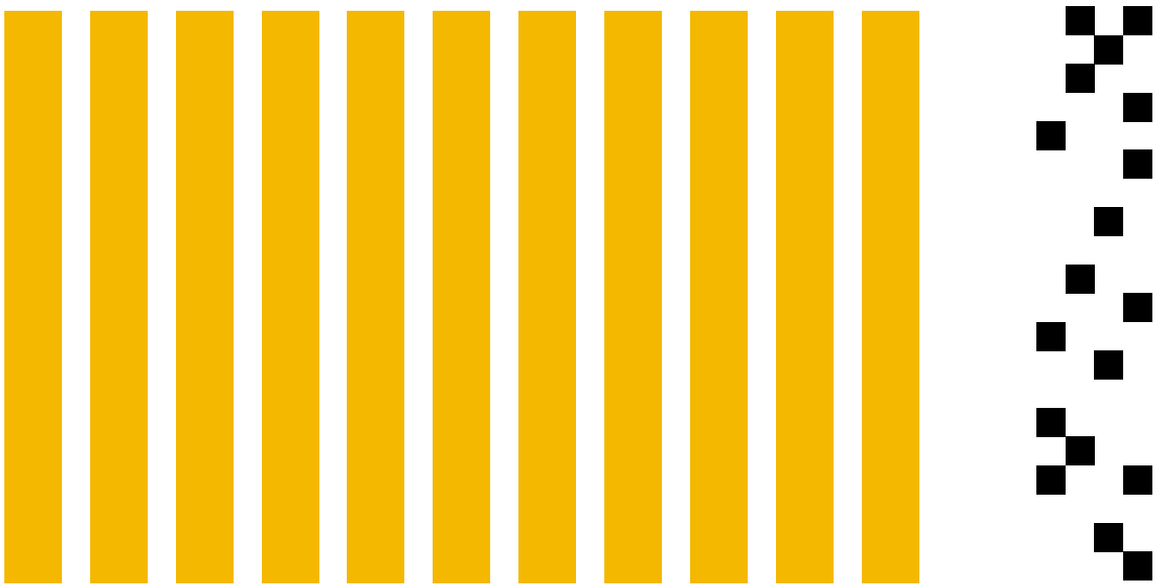
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

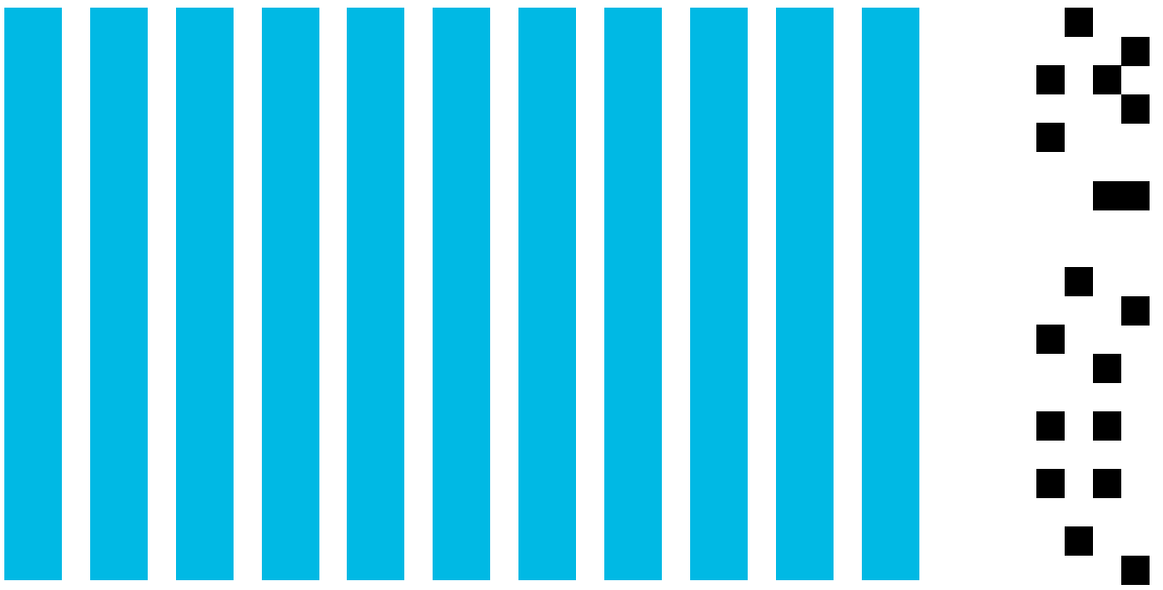
Applications

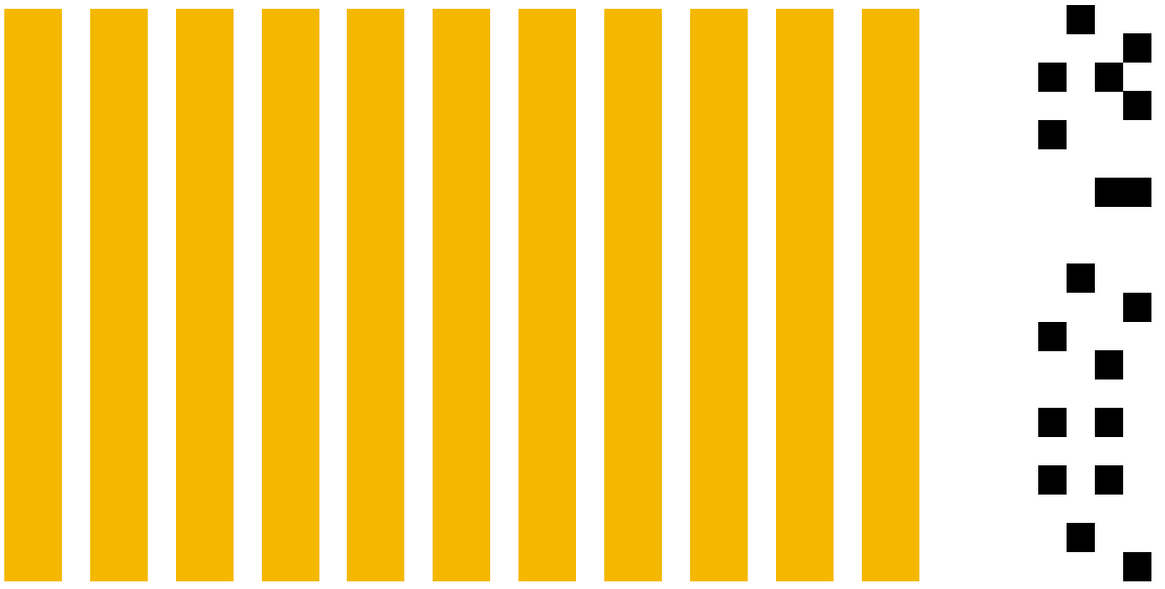
It’s prob•a•bly dif•fi•cult to imag•ine a time when a nat•u•ral lan•guage dic•tio•nary didn’t fit in main mem•ory.

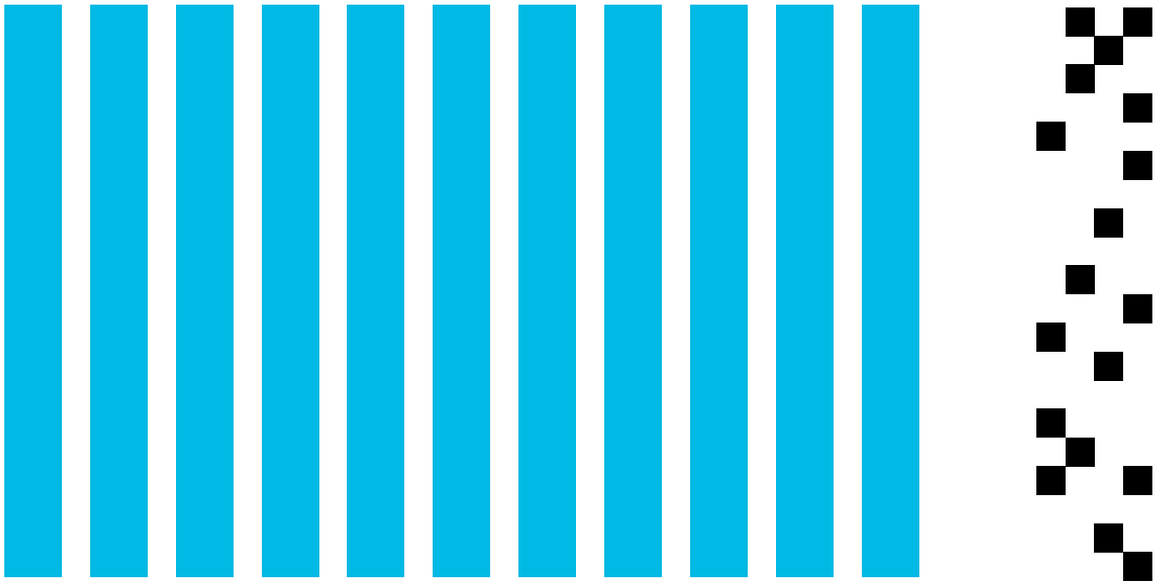
```
SELECT * FROM A,B
WHERE A.X = B.X
```

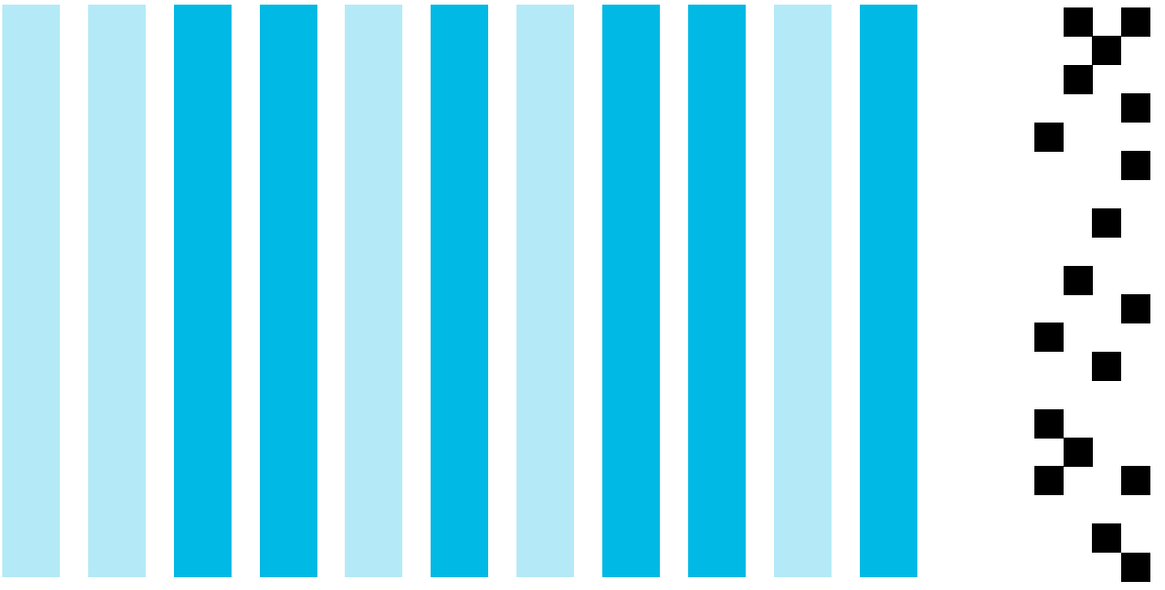


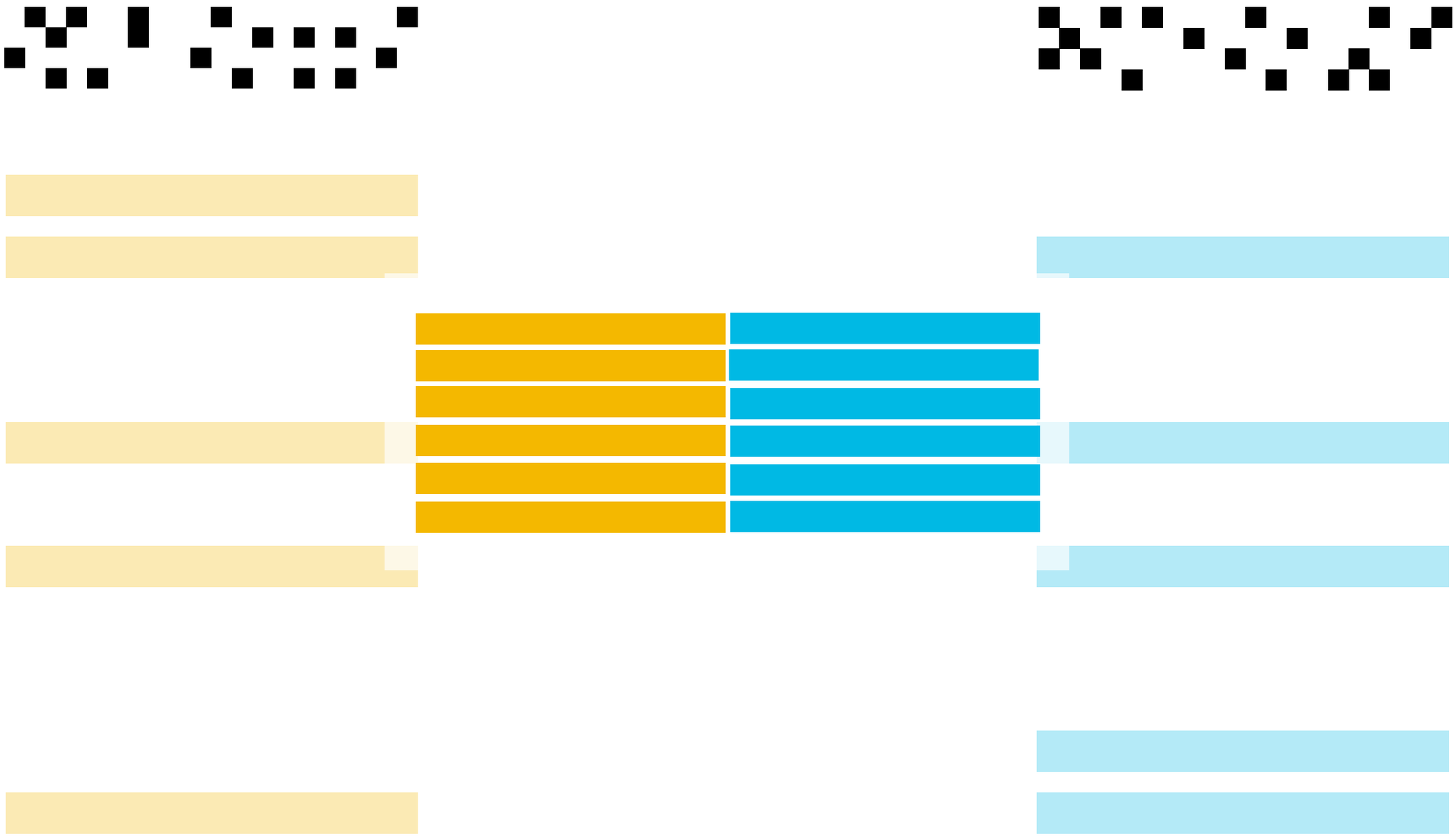


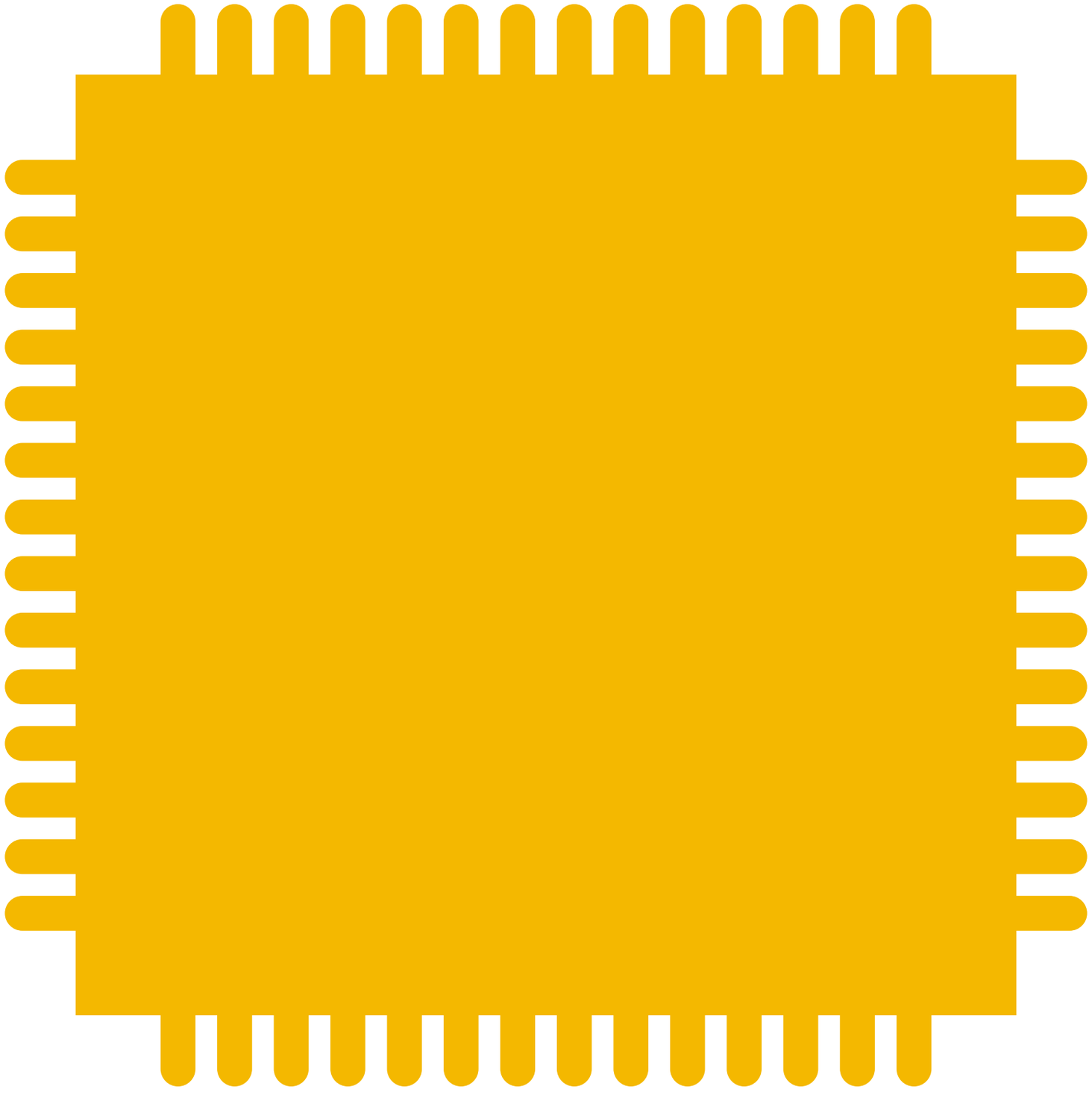












```
void a_inc(int *v, int c) {
  int i = 0;
  while (i < c) {
      v[i] = v[i] + 1;
      i++;
  }
}
```

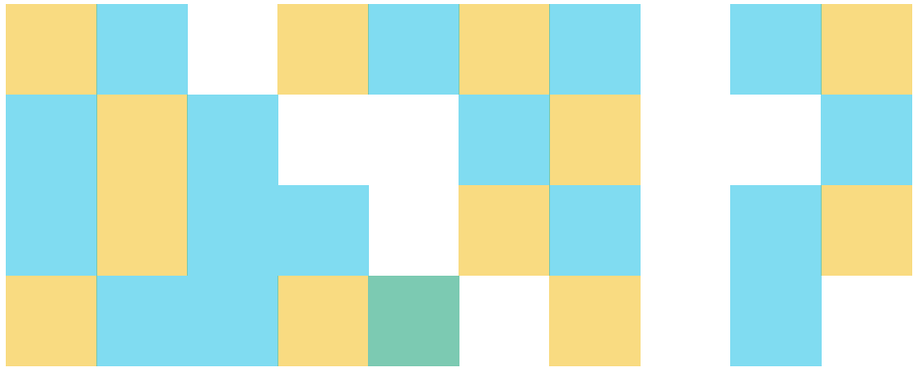
```c
void a_inc(int *v, int c) {
  int i = 0;
  while (i < c) {
    v[i] = v[i] + 1;
    i++;
  }
}

void test(int *v1, int c1, int *v2, int c2) {
  a_inc(v1, c1);
  a_inc(v2, c2);
}
```

```c
void a_inc(int *v, int c) {
  int i = 0;
  while (i < c) {
      v[i] = v[i] + 1;
      i++;
  }
}

void test(int *v1, int c1, int *v2, int c2) {
  a_inc(v1, c1);
  a_inc(v2, c2);
}
```

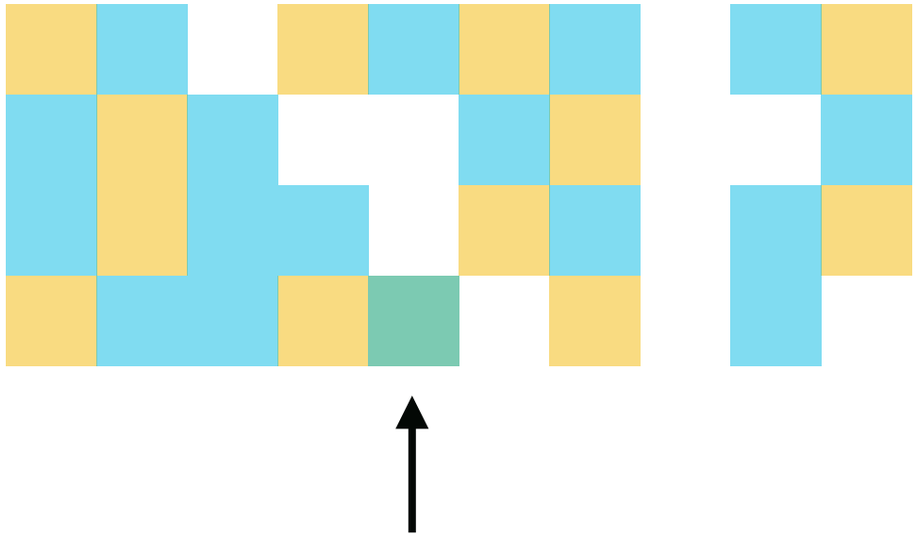
```c
void test(int *v1, int c1, int *v2, int c2) {
  a_inc(v1, c1);
  a_inc(v2, c2);
}
```

```
void test(int *v1, int c1, int *v2, int c2) {
  a_inc(v1, c1);
  a_inc(v2, c2);
}
```

```c
void test(int *v1, int c1, int *v2, int c2) {
  a_inc(v1, c1);
  a_inc(v2, c2);
}
```

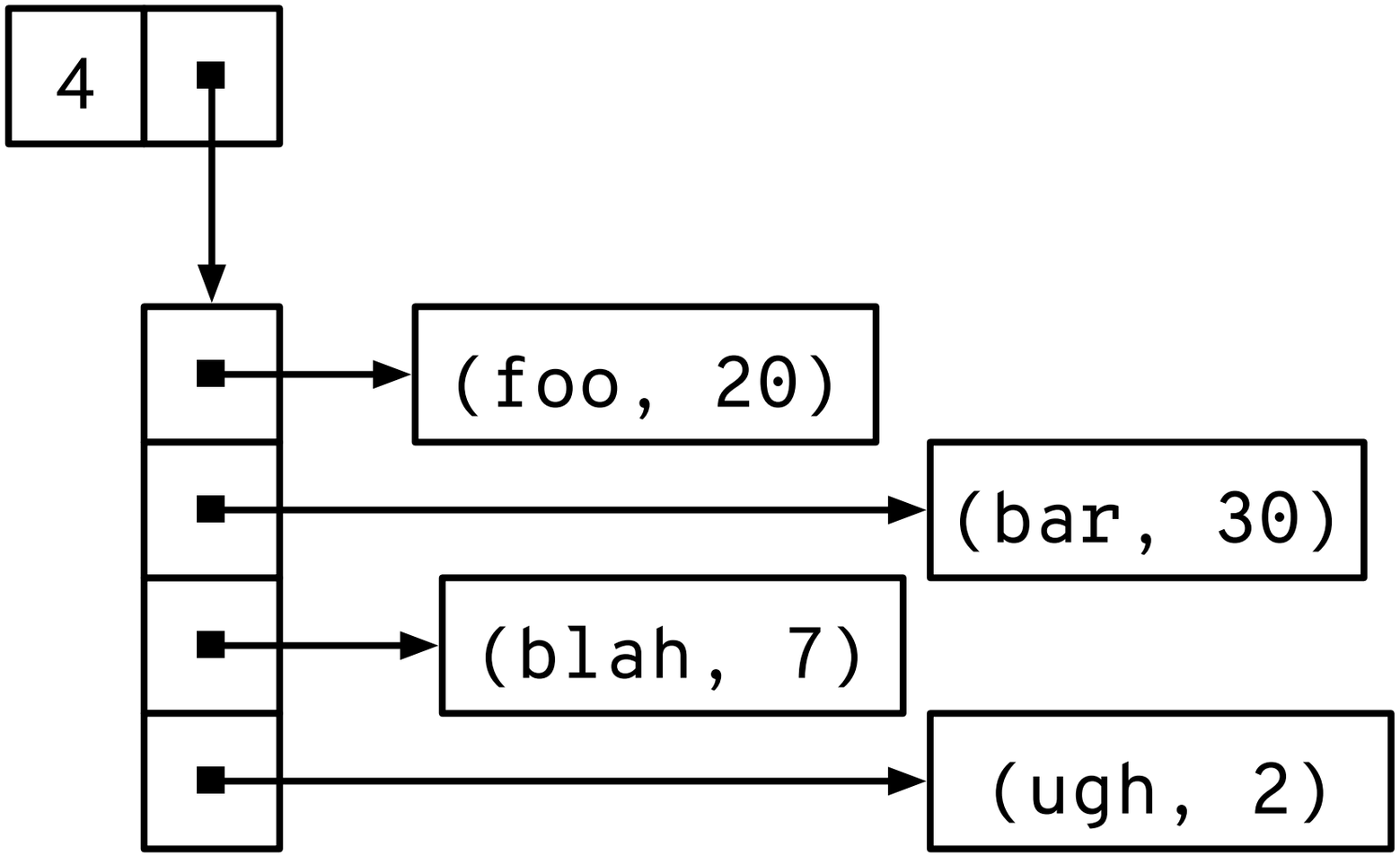


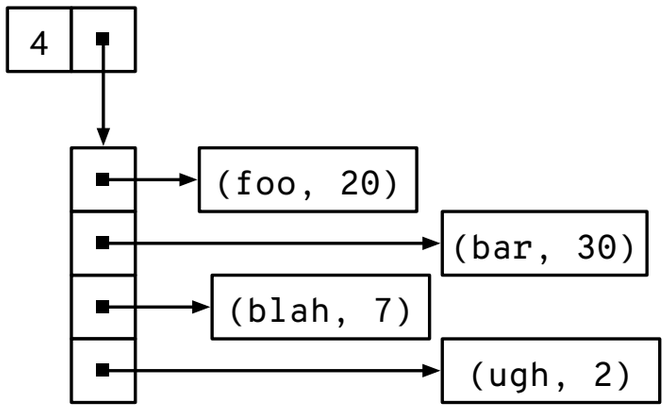
```
void test(int *v1, int c1, int *v2, int c2) {
  a_inc(v1, c1);
  a_inc(v2, c2);
}
```

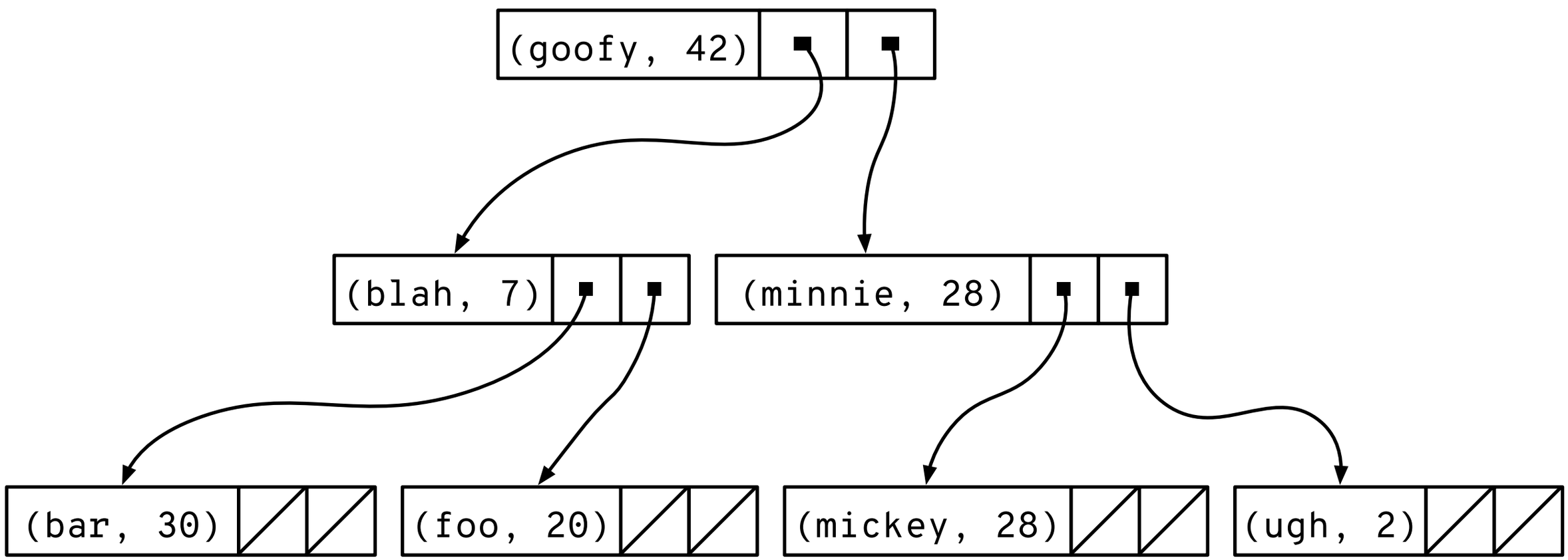


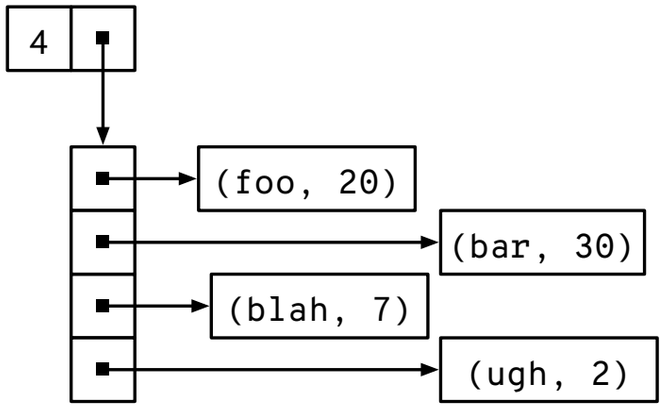
Event frequencies

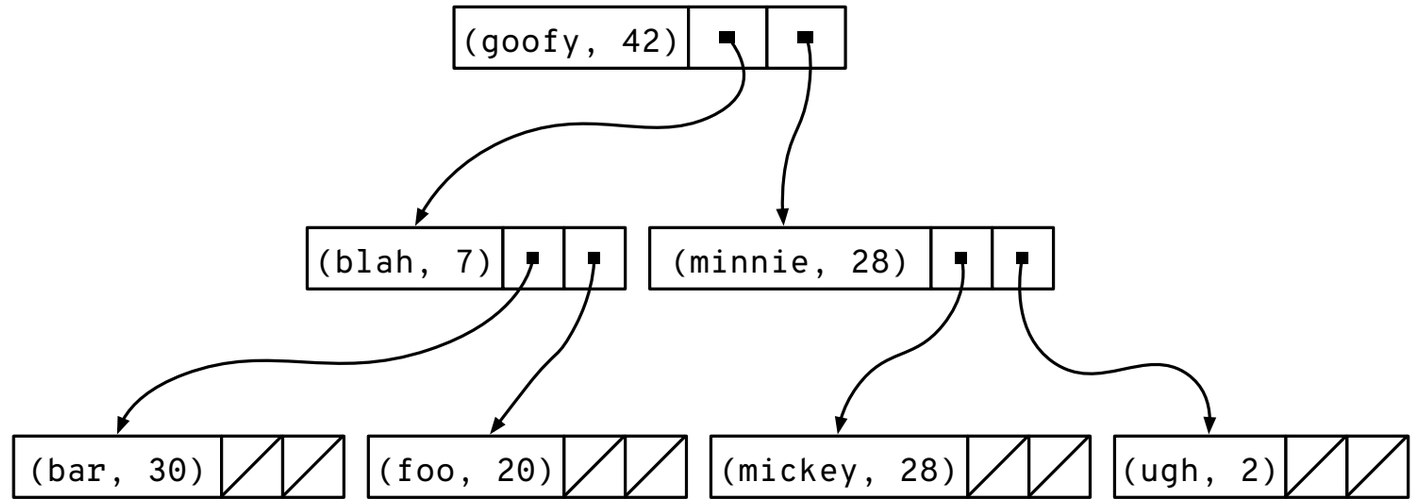
Precise structures

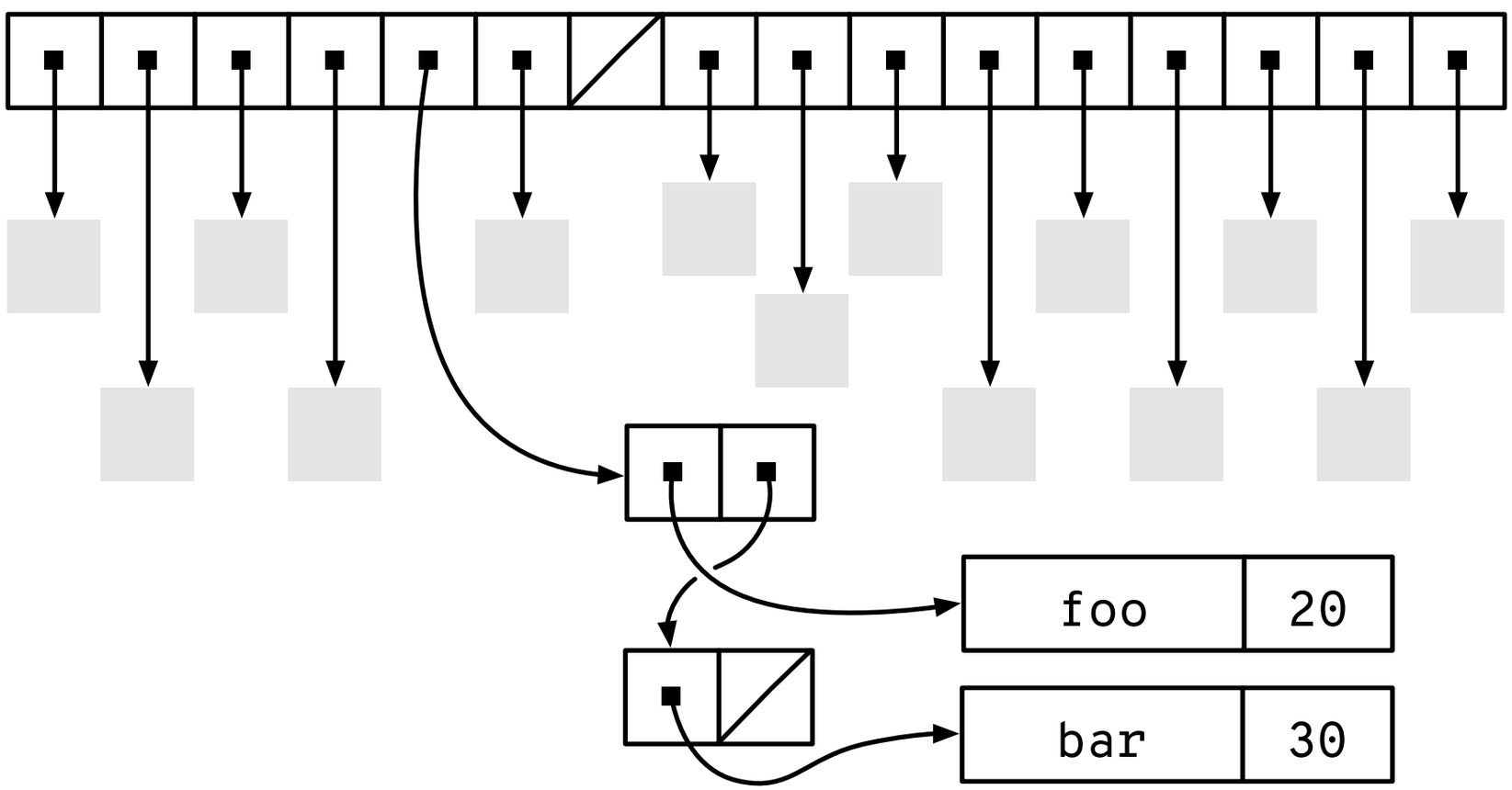

4
(foo, 20)
(bar, 30)
(blah, 7)
(ugh, 2)


4
(foo, 20)
(bar, 30)
(blah, 7)
(ugh, 2)


(goofy, 42)
(blah, 7)
(minnie, 28)
(bar, 30)
(foo, 20)
(mickey, 28)
(ugh, 2)


4
(foo, 20)
(bar, 30)
(blah, 7)
(ugh, 2)


(goofy, 42)
(blah, 7)
(minnie, 28)
(bar, 30)
(foo, 20)
(mickey, 28)
(ugh, 2)


foo
20
bar
30

Generalizing the Bloom filter: count-min sketch

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
put("foo")
```

h1("foo")

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

put("foo")

h1("foo")

h2("foo")

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

put("foo")

h1("foo")

h2("foo")

h3("foo")

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

put("foo")

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

```
lookup("foo")
```

h1("foo")

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

lookup("foo")

h1("foo")

h2("foo")

h3("foo")

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

lookup("foo")

h1("foo")

h2("foo")

h3("foo")

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

lookup("foo") = min(30, 20, 50)

```python
class CMS(object):
    import numpy as np
    def __init__(self, w, hashes):
        self._width = w
        self._hashes = lambda v: [int(f(v)) for f in hashes[:]]
        self._buckets = np.zeros((int(w), len(hashes)), np.uint64)

    def insert(self, value):
        """ Inserts a value into this sketch """
        for (row, col) in enumerate(self._hashes(value)):
            self._buckets[col % self._width][row] += 1

    def lookup(self, value):
        """ Returns a biased estimate of number of times
        value has been inserted in this sketch"""
        return min([self._buckets[col % self._width][row] for
           (row, col) in enumerate(self._hashes(value))])
```

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\oplus$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\oplus$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

0	1	1	3	0	0	2	1
1	1	2	0	1	2	0	1
0	2	2	1	0	0	2	0

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 +

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 +

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 +

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 +

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 + 1 +

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 + 1 +

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 + 1 + 1

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 + 1 + 1 + 2

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 + 1 + 1 + 2 = 7

subsystem: etcd

0	1	0	2	0	0	1	0
1	0	1	0	0	1	0	1
0	1	2	0	0	0	1	0

$\bullet$

severity: ERROR

0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0
0	1	0	1	0	0	2	0

2 + 1 + 1 + 1 + 2 = 7

Top-k elements

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

```
put("foo")
```

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

```
put("foo")
```

CMS

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

```
put("foo")
```

Priority queue

CMS

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

Priority queue

```
put("foo")
```

(20, foo)				

0	5	0	30	0	56	12	0	40	31	1	0	45	6	0	31
5	20	17	0	1	45	20	6	0	20	40	31	33	5	7	54
1	3	21	30	0	0	42	7	52	10	17	20	0	0	50	7

```
put("ugh")
```

(20, foo)				

0	5	0	30	0	56	12	0	40	31	2	0	45	6	0	31
5	20	17	0	2	45	20	6	0	20	40	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	50	7

```
put("ugh")
```

(20, foo)	(2, ugh)			

0	5	0	30	0	56	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	45	20	6	0	20	41	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	50	8

```
put("blah")
```

(20, foo)	(2, ugh)			

0	5	0	30	0	56	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	45	20	6	0	20	41	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	50	8

```
put("blah")
```

(20, foo)	(7, blah)	(2, ugh)		

0	5	0	30	0	56	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	45	20	6	0	20	41	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	50	8

```
put("goofy")
```

(20, foo)	(7, blah)	(2, ugh)		

0	5	0	30	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	45	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	51	8

```
put("goofy")
```

(20, foo)	(7, blah)	(2, ugh)		

0	5	0	30	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	45	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	51	8

```
put("goofy")
```

(42, goofy)	(20, foo)	(7, blah)	(2, ugh)	

0	5	0	30	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	45	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	42	7	52	10	17	20	0	0	51	8

```
put("bar")
```

(42, goofy)	(20, foo)	(7, blah)	(2, ugh)	

0	5	0	31	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	46	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	43	7	52	10	17	20	0	0	51	8

```
put("bar")
```

(42, goofy)	(20, foo)	(7, blah)	(2, ugh)	

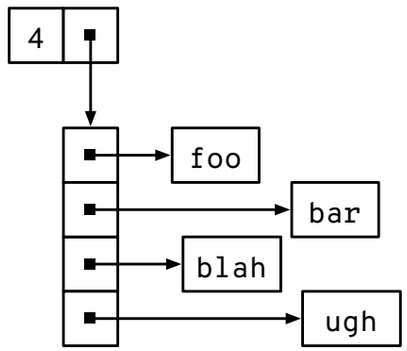
0	5	0	31	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	46	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	43	7	52	10	17	20	0	0	51	8

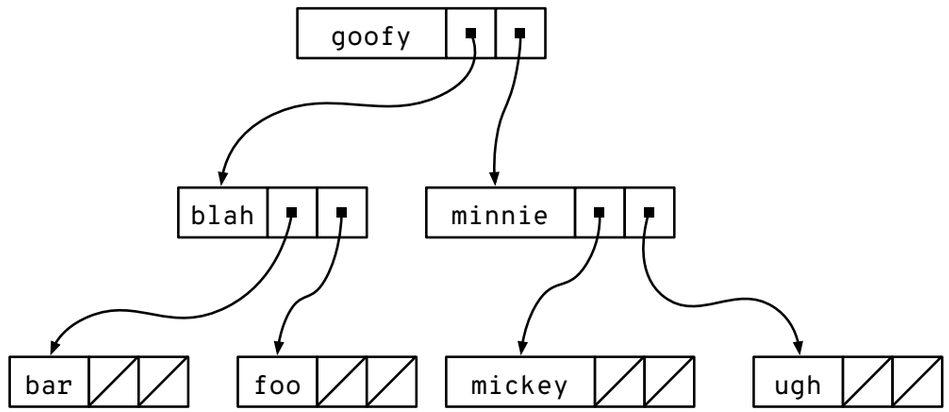
```
put("bar")
```

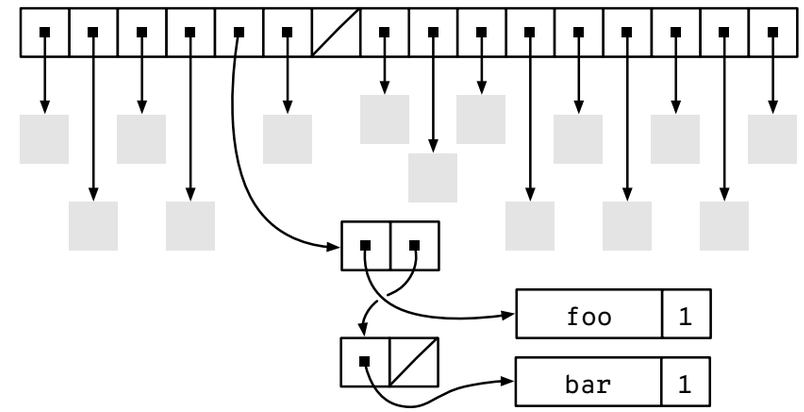
(42, goofy)	(31, bar)	(20, foo)	(7, blah)	(2, ugh)

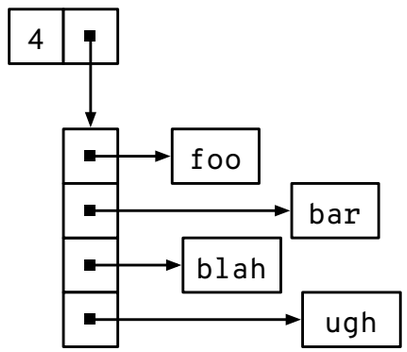
Counting distinct items

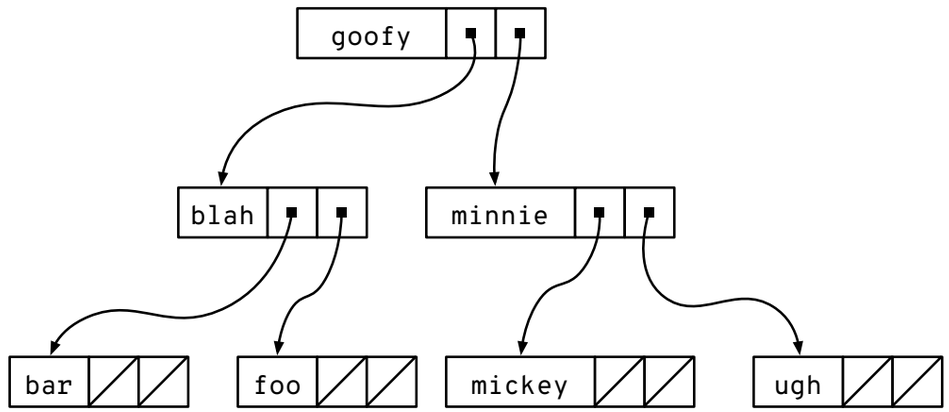
Precise approaches

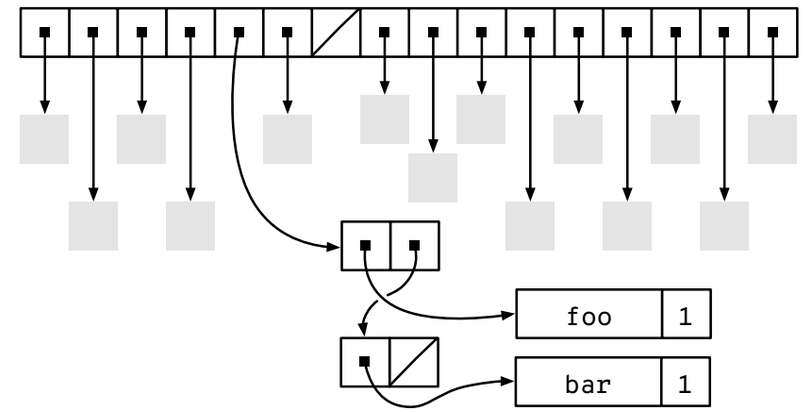

4
foo
bar
blah
ugh


goofy
blah
minnie
bar
foo
mickey
ugh


foo
1
bar
1


4
foo
bar
blah
ugh


goofy
blah
minnie
bar
foo
mickey
ugh


foo
1
bar
1

|s| = len(s)

Estimating cardinality with a Bloom filter

```python
def approx_cardinality(self):
    """ Estimates the cardinality of the
        set modeled by this filter.
        Uses a technique from Swamidass
        and Baldi (2007). """
    from math import log
    m = self.size() * self.partitions()
    k = self.partitions()
    X = self.__buckets.count_set_bits()
    return -(m / k) * log(1 - (X / m))
```

HyperLogLog

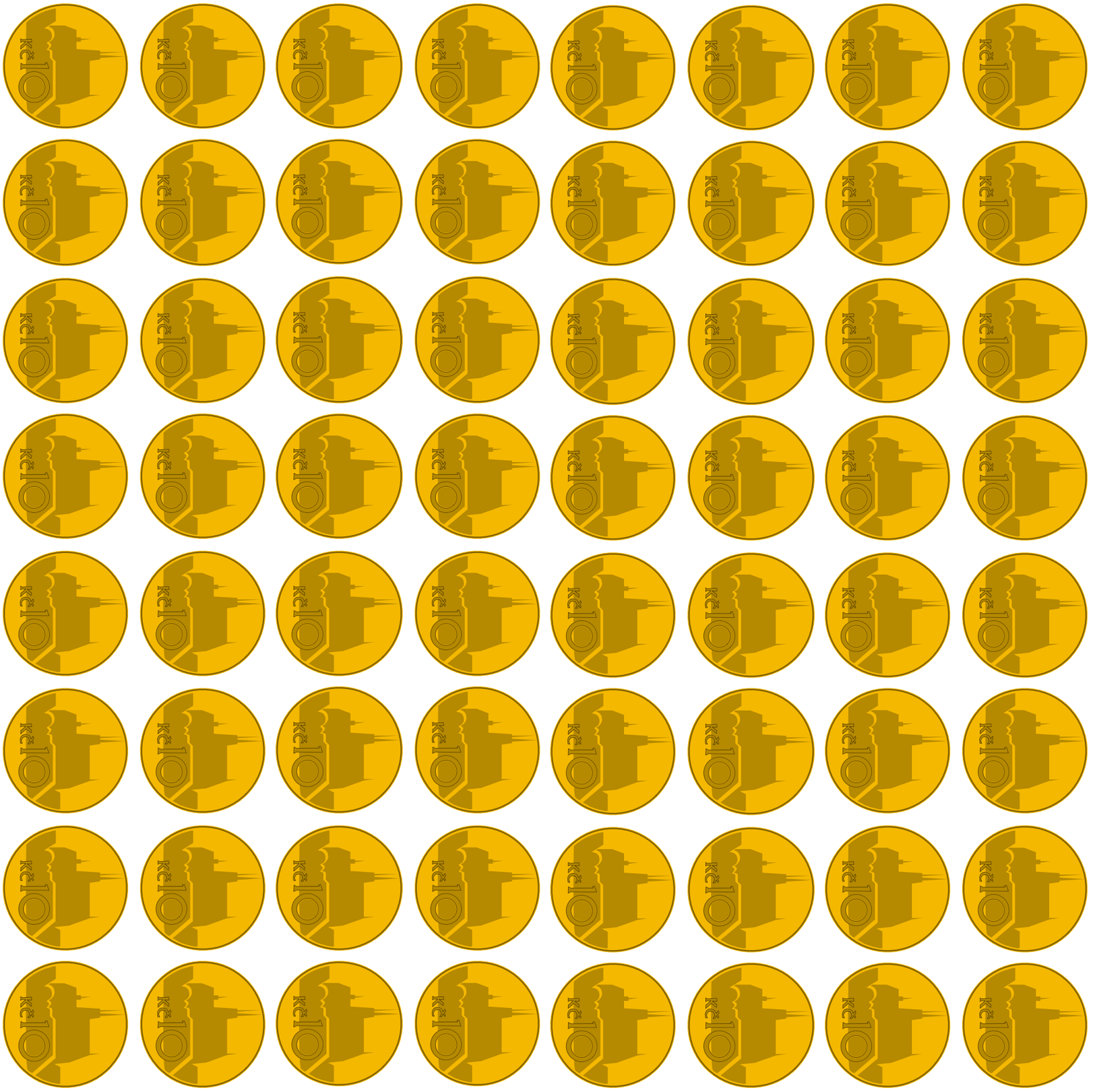

Kč 10


Kč 10


Kč 10


Kč 10


Kč 10

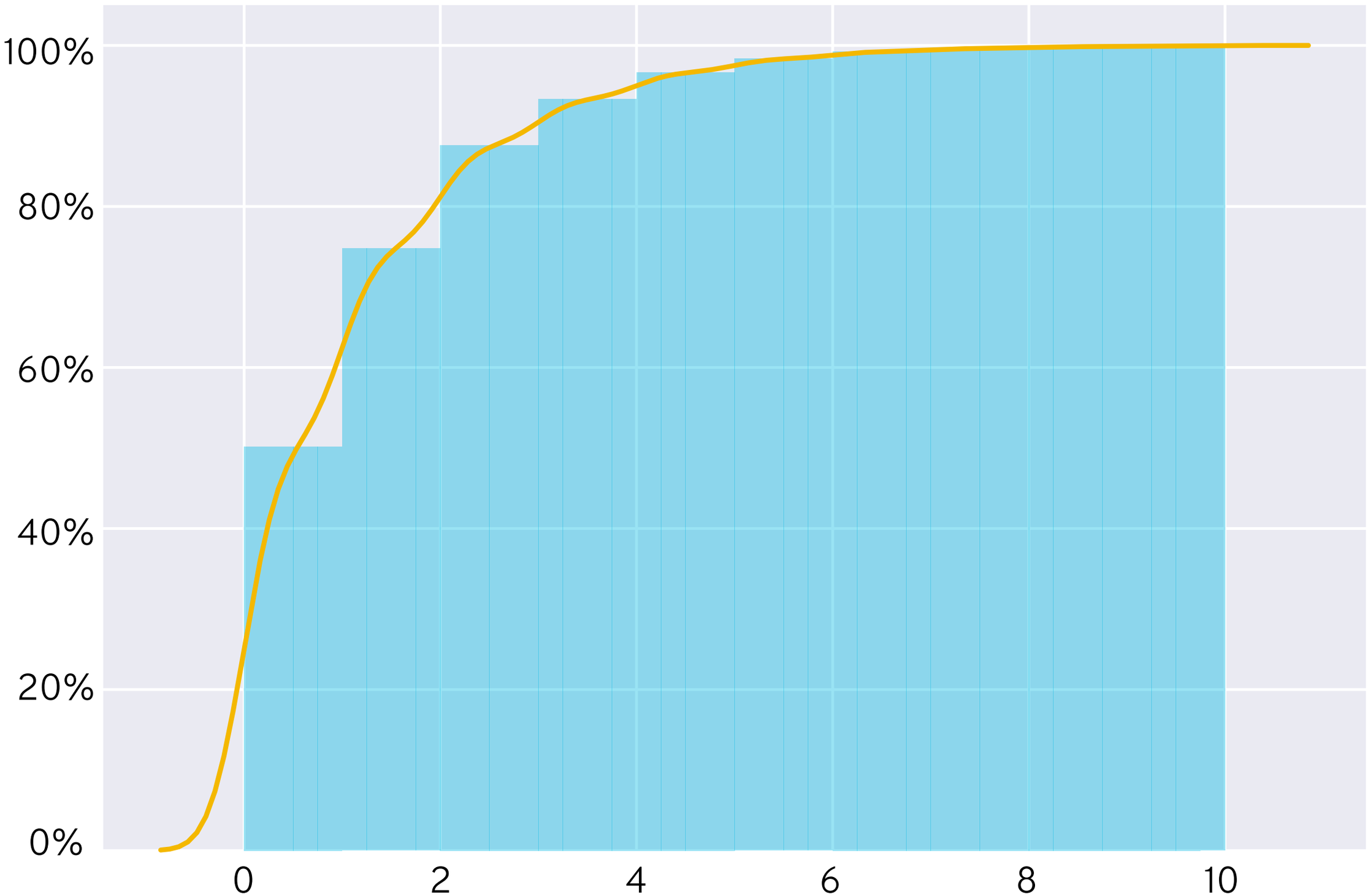

Kč 10

1	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0

0	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1

0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0

0	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0


100%
80%
60%
40%
20%
0%
0
2
4
6
8
10

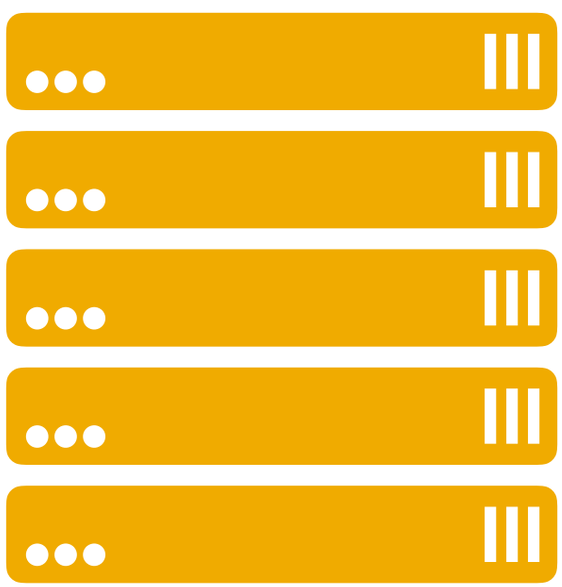
1	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0

0	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1

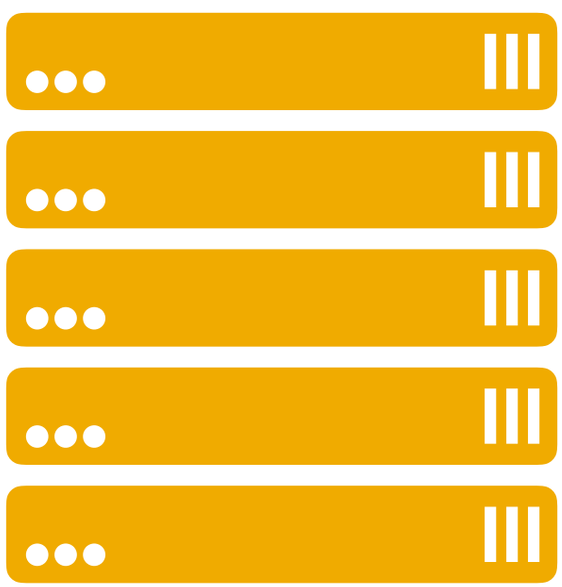
0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0

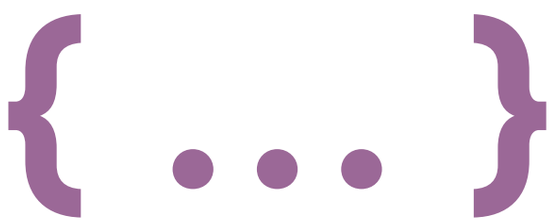
0	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0

1	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1
0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0



1	0	0	1	0	0	1	0	1	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1
0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0





{ … }

$$h(\{\ldots\}) = \begin{array}{|c|c|c|c|c|c|c|c|}\hline 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

$$h(\{\ldots\}) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline \end{array}$$

0	0	0	0	0	0	0	0

$$h(\{\ldots\}) =$$

0	1	0	0	0	1	1	0

0	0	0	0	0	0	0	0

h({ ... }) =

0	1	0	0	0	1	1	0

0	0	0	0	0	0	0	0

h({ ... }) =

0	1	0	0	0	1	1	0

0	0	0	0	0	2	0	0

h({ ... }) =

0	1	0	0	0	1	1	0

0	0	0	0	0	2	0	0

$$h(\{\ldots\}) =$$

1	1	0	1	0	0	1	0

0	0	0	0	0	2	0	0

h({ ... }) =

1	1	0	1	0	0	1	0

0	0	0	0	0	2	0	0

$$h(\{\ldots\}) =$$

1	1	0	1	0	0	1	0

1	3	2	1	3	2	1	2

$$h(\{\ldots\}) =$$

1	1	0	1	0	0	1	0

1	3	2	1	3	2	1	2

$$h(\{\ldots\}) =$$

1	1	0	1	0	0	1	0

$$|set| \approx 16$$

1	3	2	1	3	2	1	2

$\oplus$

1	1	4	1	2	3	2	1

1	3	2	1	3	2	1	2

$\oplus$

1	1	4	1	2	3	2	1

1	3	2	1	3	2	1	2

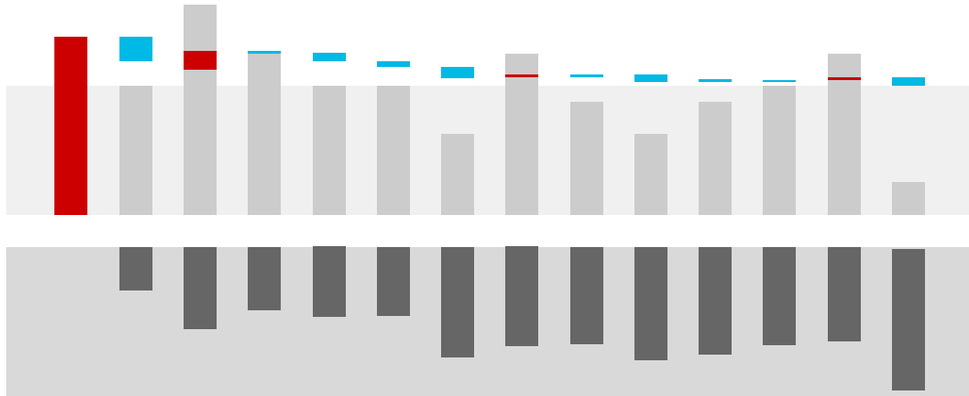
$\oplus$

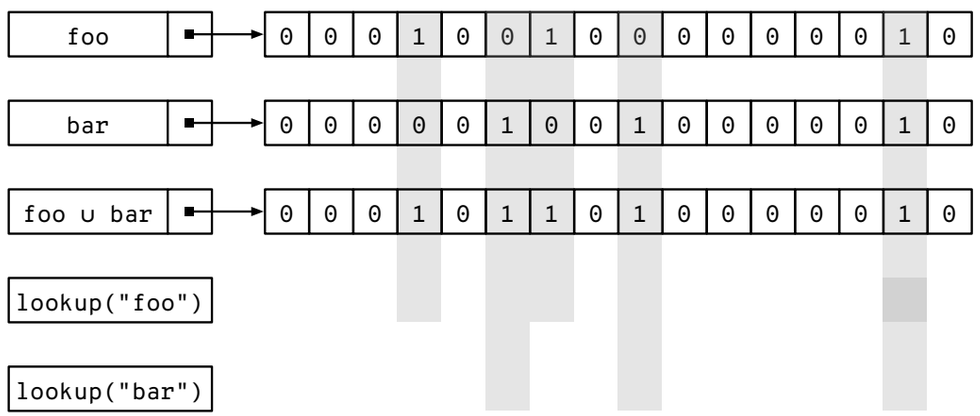
1	1	4	1	2	3	2	1

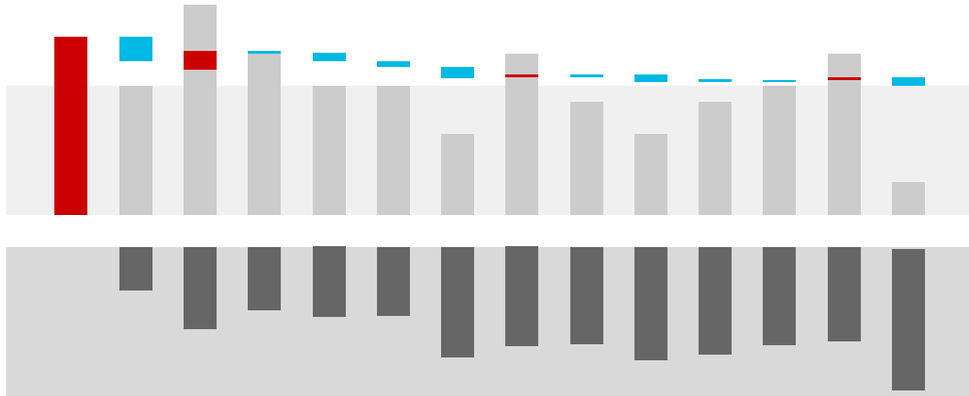
1	**3**	**4**	**1**	**3**	**3**	**2**	**2**

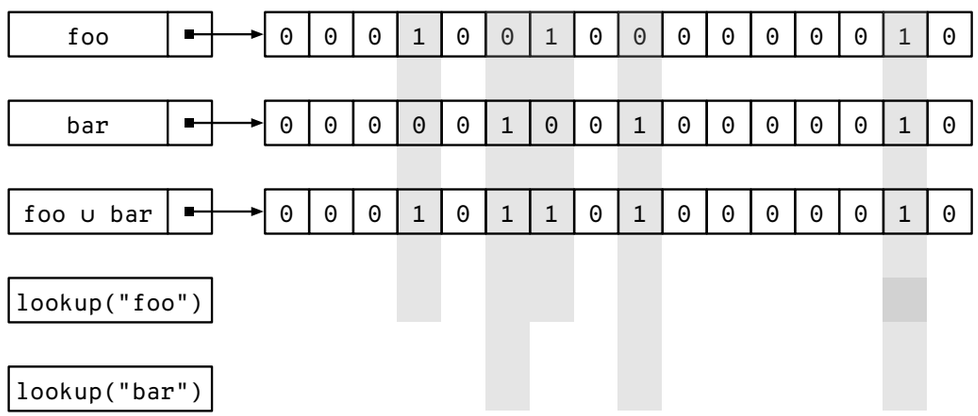
```python
import numpy as np
from scipy.stats import hmean

class HLL(object):
    def __init__(self, p=4):
        self.p = min(max(p, 4), 12)
        self.alpha = get_alpha(self.p)
        self._registers = np.zeros(int(2 ** self.p), np.uint8)

    def add(self, v):
        h = h64(v)
        idx = h & (len(self._registers) - 1)
        h >>= self.p
        fsb = first_set_bit(h, 64 - self.p)
        self._registers[idx] = max(self._registers[idx], fsb)

    def approx_count(self):
        m = len(self._registers)
        if self._registers.count_nonzero() < len(self._registers):
            return m * math.log(float(m) / self._zeros)
        else:
            return self.alpha * m * hmean(np.power(2.0, self._registers))
```

Putting it all together



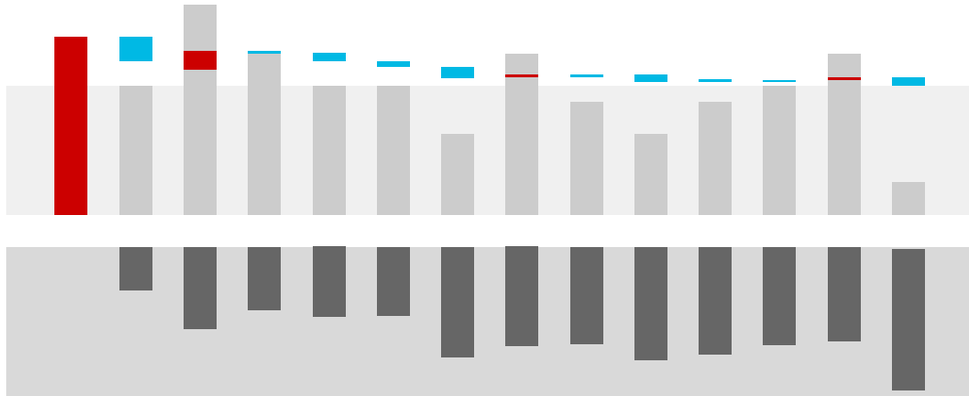

foo
0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0
bar
0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
foo ∪ bar
0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0
lookup("foo")
lookup("bar")





0	5	0	31	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	46	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	43	7	52	10	17	20	0	0	51	8

put("bar")



foo	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0
bar	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0
foo ∪ bar	0	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0

lookup("foo")

lookup("bar")

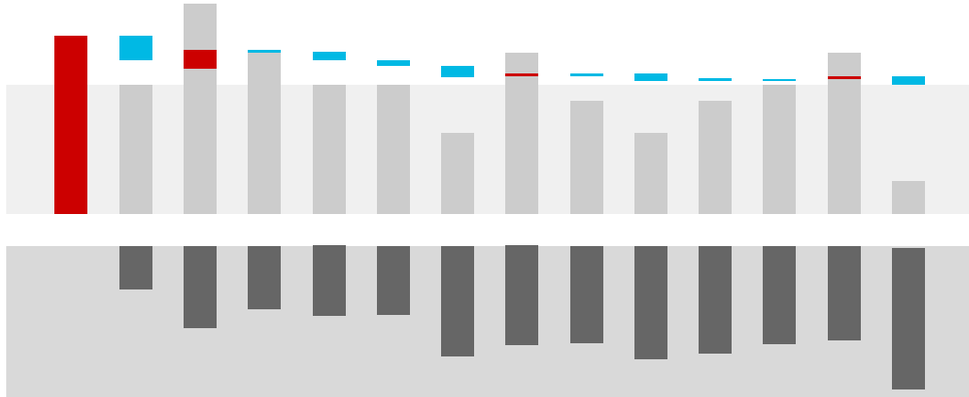
0	5	0	31	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	46	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	43	7	52	10	17	20	0	0	51	8

put("bar")

0	0	0	0	0	2	0	0

h({ ... }) =

0	1	0	0	0	1	1	0



foo	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0
bar	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0
foo ∪ bar	0	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0

lookup("foo")

lookup("bar")

0	5	0	31	0	57	12	0	40	31	2	0	45	7	0	31
5	20	17	0	2	46	20	6	0	20	42	31	33	5	7	54
1	4	21	30	0	0	43	7	52	10	17	20	0	0	51	8

put("bar")

0	0	0	0	0	2	0	0

h({ ... }) =

0	1	0	0	0	1	1	0

@willb

willb@redhat.com

https://radanalytics.io

https://chapeau.freevariable.com