Acceptably inaccurate

Probabilistic data structures

Hello

Today's talk

Motivation

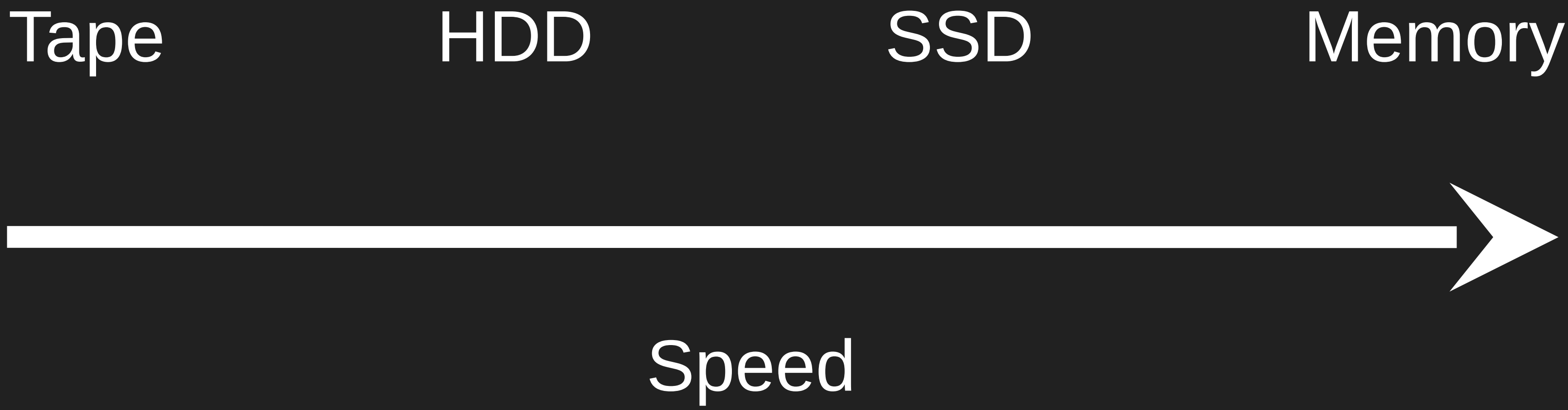
Bloom filters

Count-Min Sketch

HyperLogLog

Motivation

Tape HDD SSD Memory


Tape
HDD
SSD
Memory
Speed

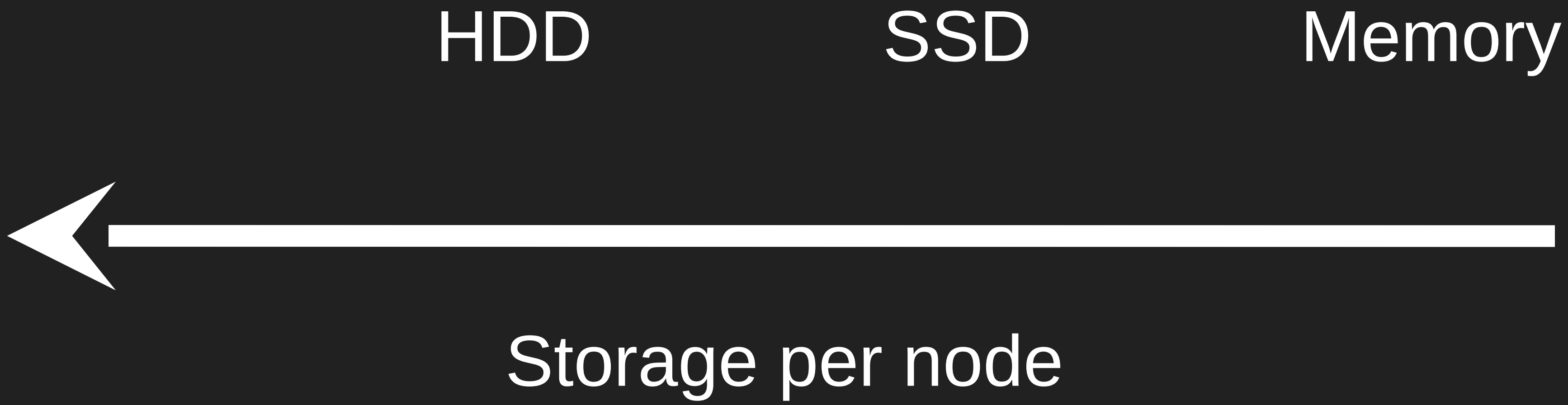

Tape
HDD
SSD
Memory
Cost

Tape HDD SSD Memory

Ease of use
(as a developer)

Tape | HDD | SSD | Memory

Set<String>


HDD
SSD
Memory
Storage per node

HDD SSD Memory

How can we do more here?

Probabilistic data structures

I, as a developer, accept a predictable level of inaccuracy.

Bloom filters

Bloom filters are for
set membership

```java
Set<String> visitors = new HashSet<>();

visitors.add("192.169.0.1");
visitors.add("74.245.10.1");
visitors.add("10.124.22.19");

visitors.contains("10.124.22.19"); // true
visitors.contains("999.999.999.999"); // false
```

Number of UUIDs	**JVM heap used (MB)**
10	< 2
100	< 2
1,000	3
10,000	9
100,000	37
1,000,000	264

What is a Bloom filter?

[Space/time trade-offs in hash coding with allowable errors (Bloom, 1970)]

A bit array of size *n*

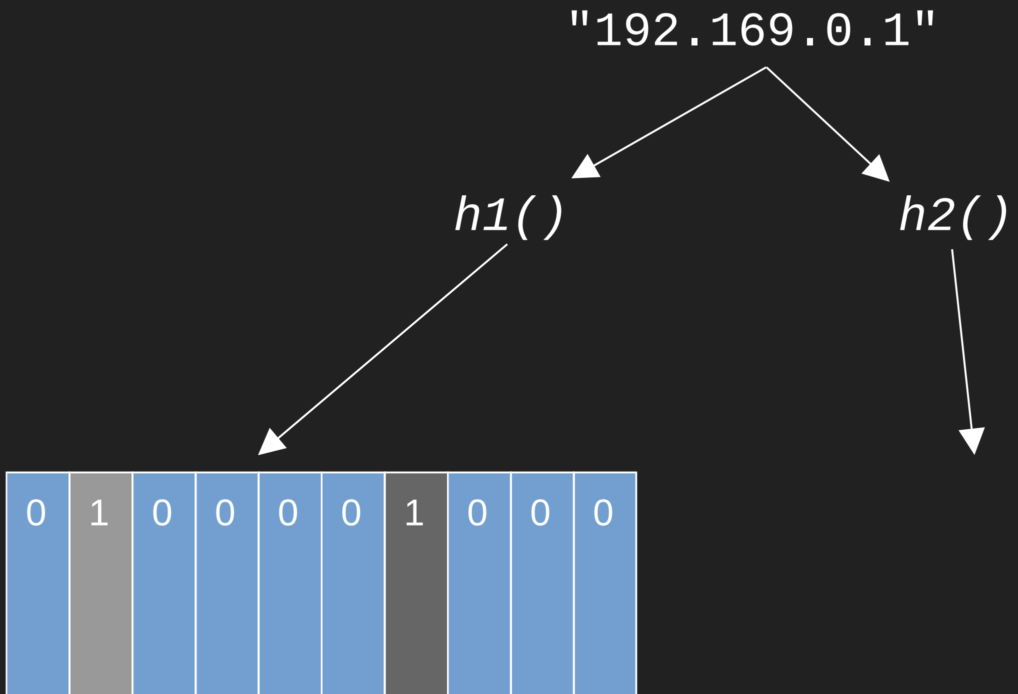
0	0	0	0	0	0	0	0	0	0

k hash functions

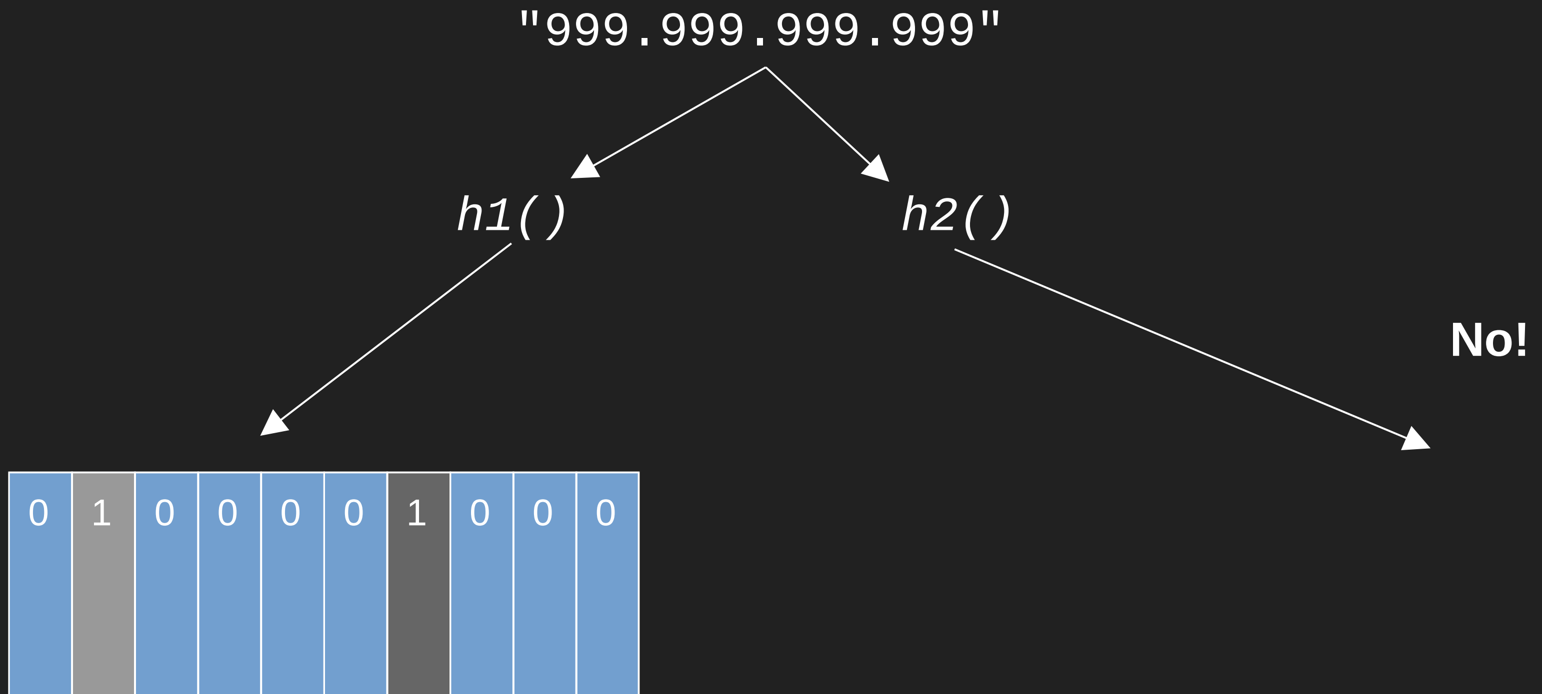
h1() *h2()*

0	0	0	0	0	0	0	0	0	0

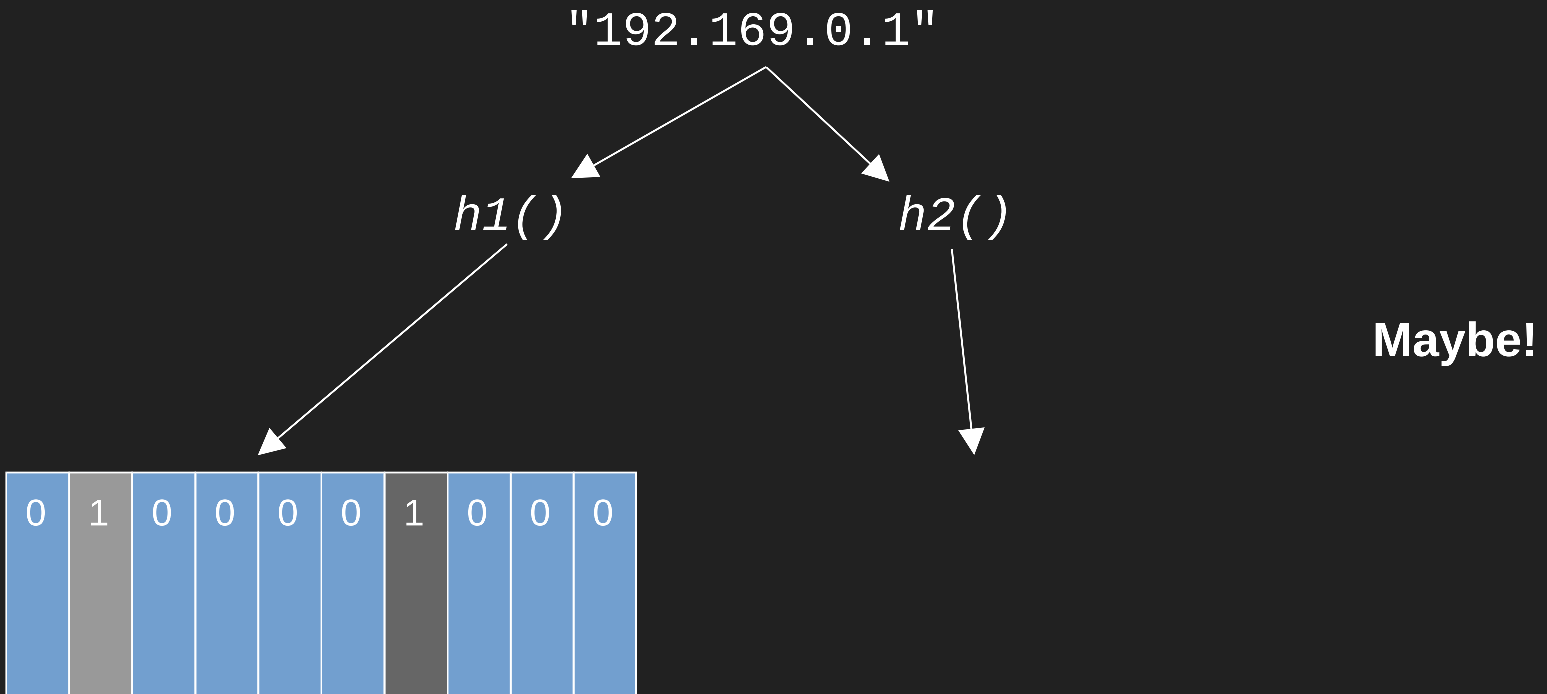
Adding an element



Checking for the presence of an element



Checking for the presence of an element



```xml
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>19.0</version>
</dependency>
```

```java
private BloomFilter<String> visitors =
    BloomFilter.create(
        Funnels.stringFunnel(Charset.forName("UTF-8")),
        10000);

visitors.add("192.169.0.1");
visitors.add("74.245.10.1");
visitors.add("10.124.22.19");

visitors.mayContain("10.124.22.19"); // true
visitors.mayContain("999.999.999.999"); // false
```

Number of UUIDs	Set JVM heap used (MB)	Bloom filter JVM heap used (MB)
10	< 2	0.01
100	< 2	0.01
1,000	3	0.01
10,000	9	0.02
100,000	37	0.1
1,000,000	264	0.9

Suggested size of Bloom filter	Bit count
10	40
100	378
1,000	3654
10,000	36231
100,000	361992
1,000,000	3619846

3% false positive probability by default.

```java
private BloomFilter<String> visitors =
    BloomFilter.create(
        Funnels.stringFunnel(Charset.forName("UTF-8")), 10000,
        0.005);
```

Suggested FPP of Bloom filter	Hash functions
3%	5
1%	7
0.1%	10
0.01%	13
0.001%	17
0.0001%	20

Use cases

"One hit wonders"

Avoiding lookups (HBase, Cassandra)

Real-time matching

Algorithmic nuggets in content delivery (Maggs, 2015)

Count-min sketch

Count-min sketch for count tracking

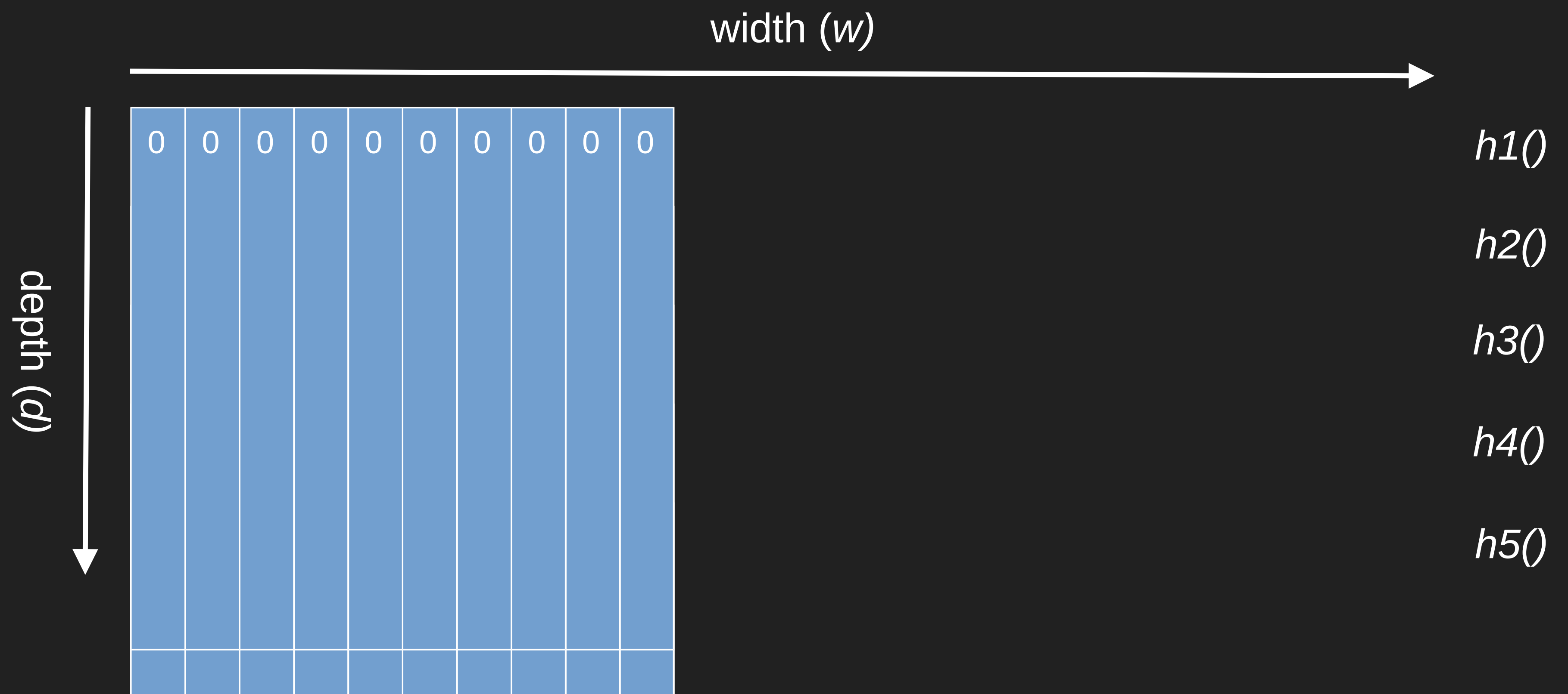
```java
Multiset<String> hits = HashMultiset.create();

hits.add("192.169.0.1");
hits.add("74.245.10.1");
hits.add("10.124.22.19");
hits.add("10.124.22.19");

hits.count("10.124.22.19"); // 2
hits.count("999.999.999.999"); // 0
```

Number of UUIDs	**JVM heap used (MB)**
10	< 2
100	< 2
1,000	3
10,000	9
100,000	39
1,000,000	234

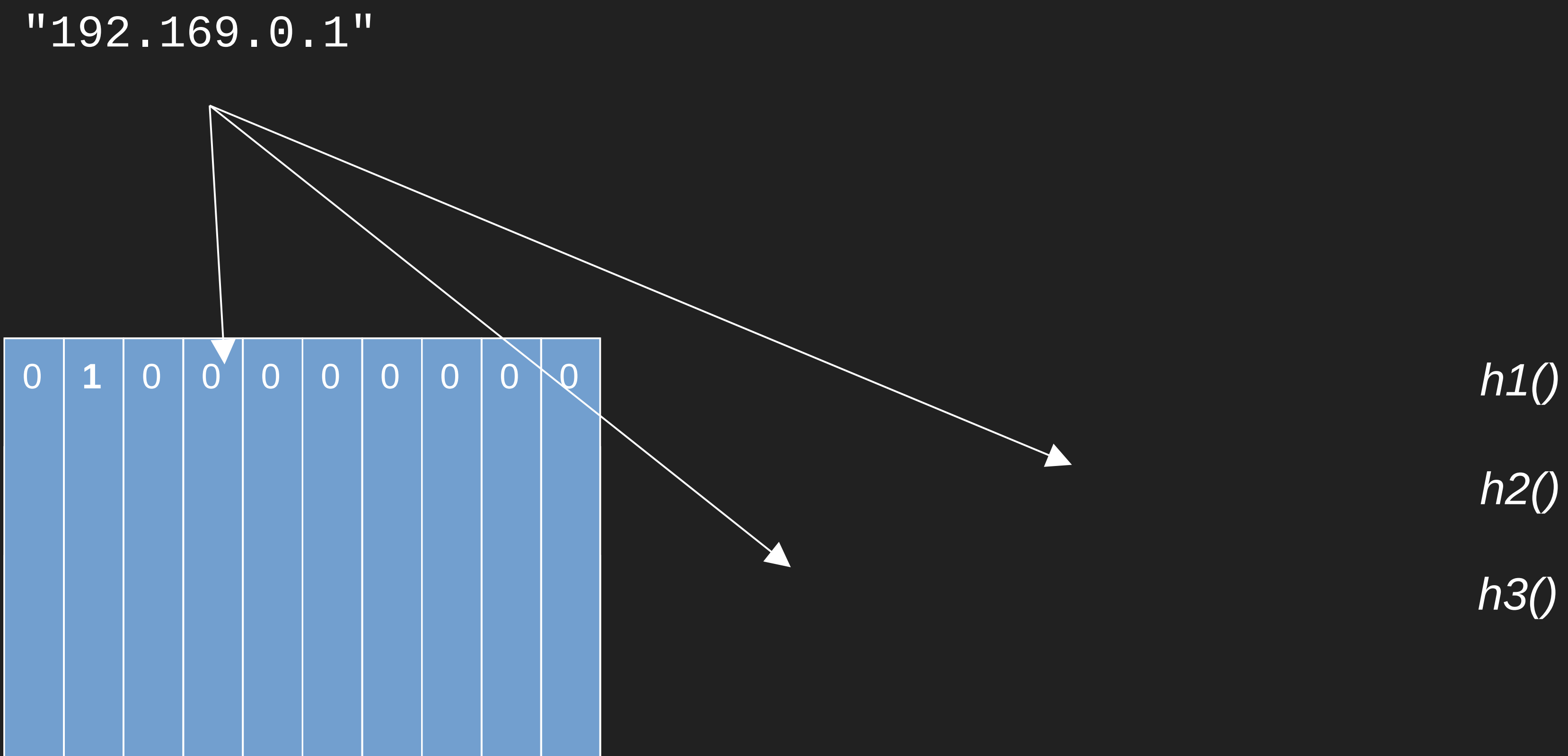
What is a Count-Min Sketch?

Approximating data with the count-min data structure (Cormode, Muthukrishnan, 2012)

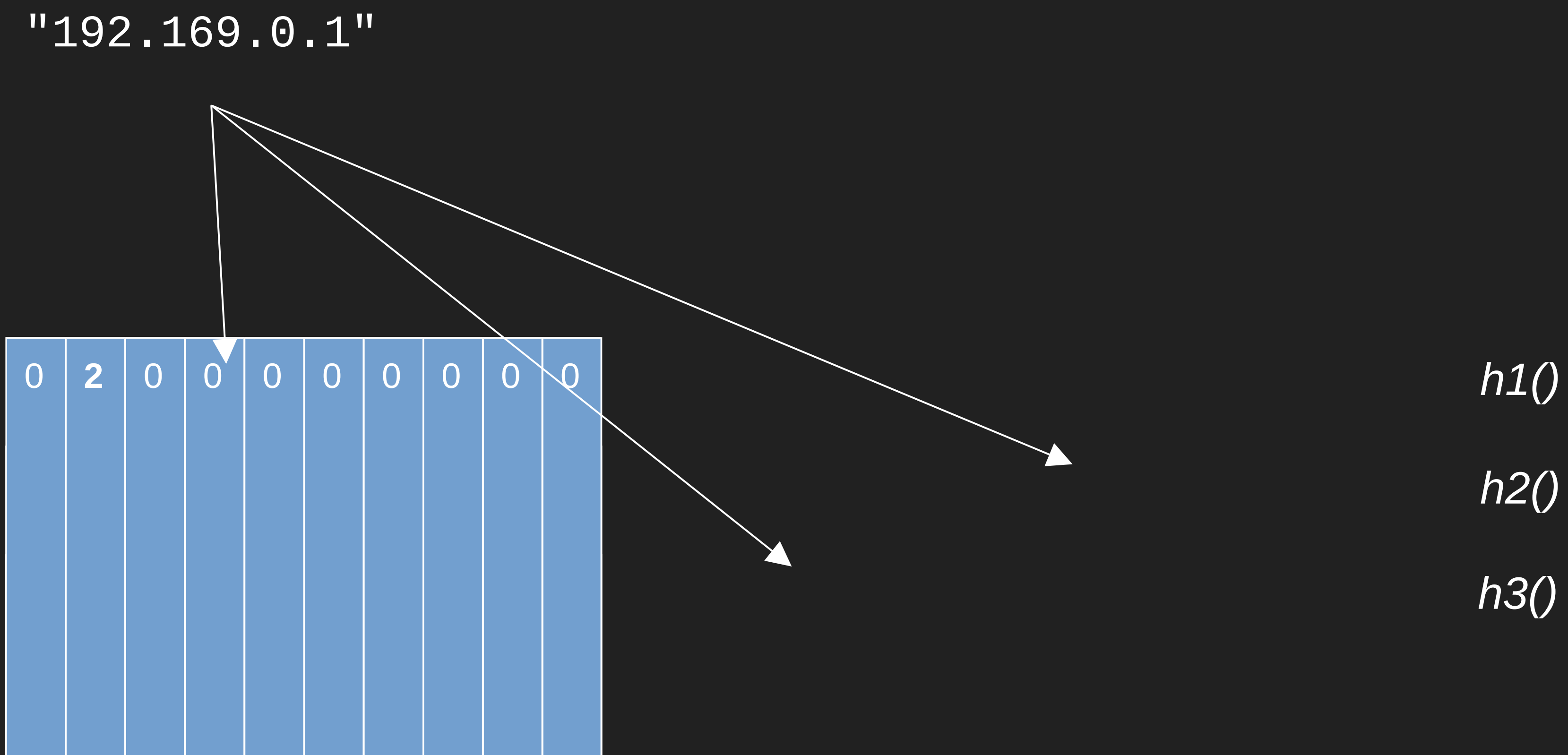
An array of counters



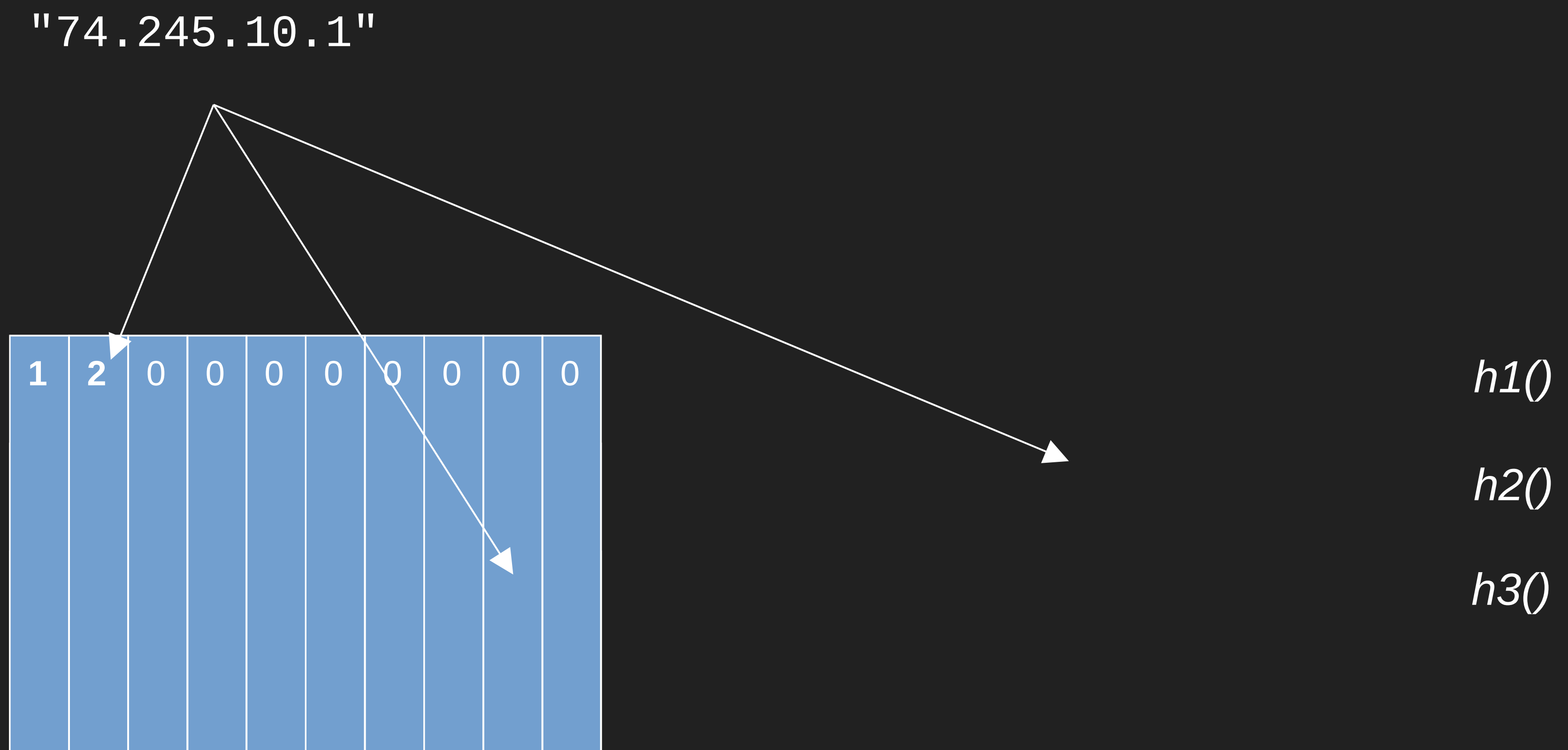
Adding an element



Adding the same element again



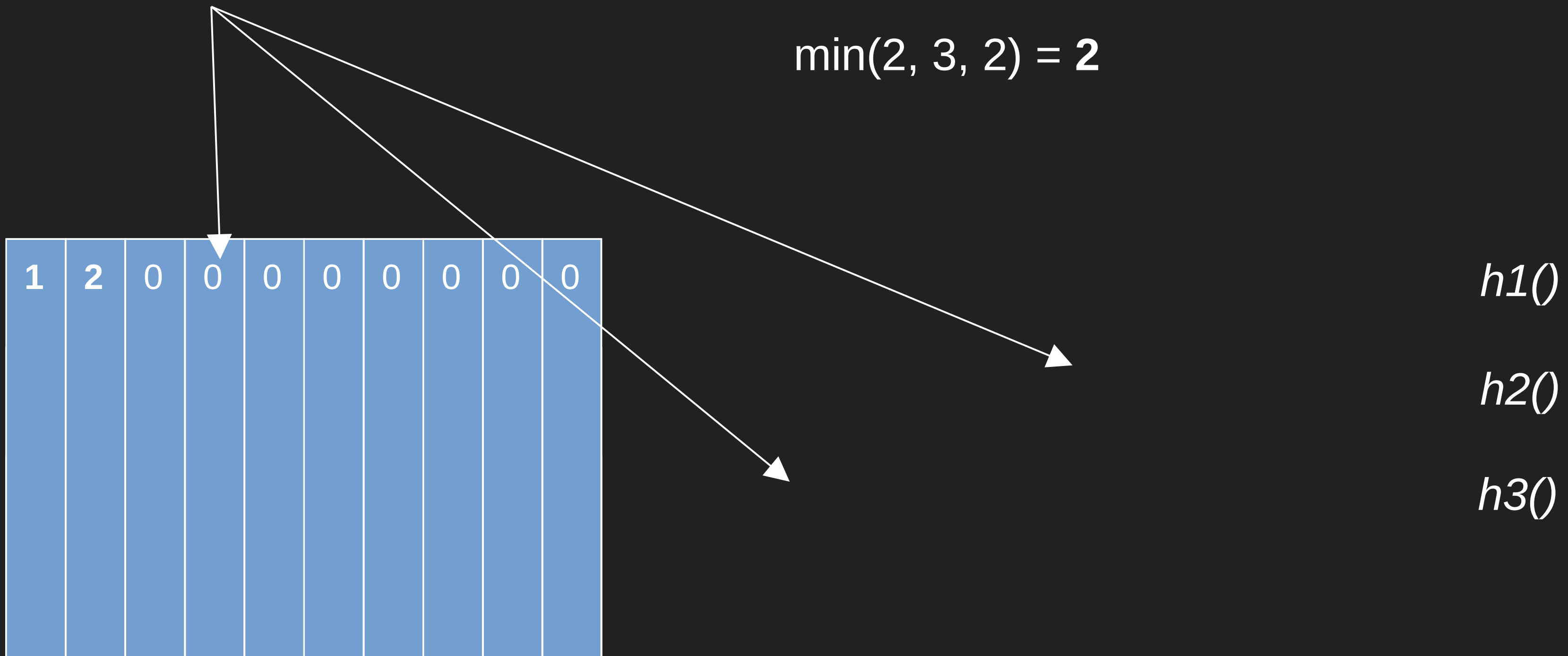
Adding a different element



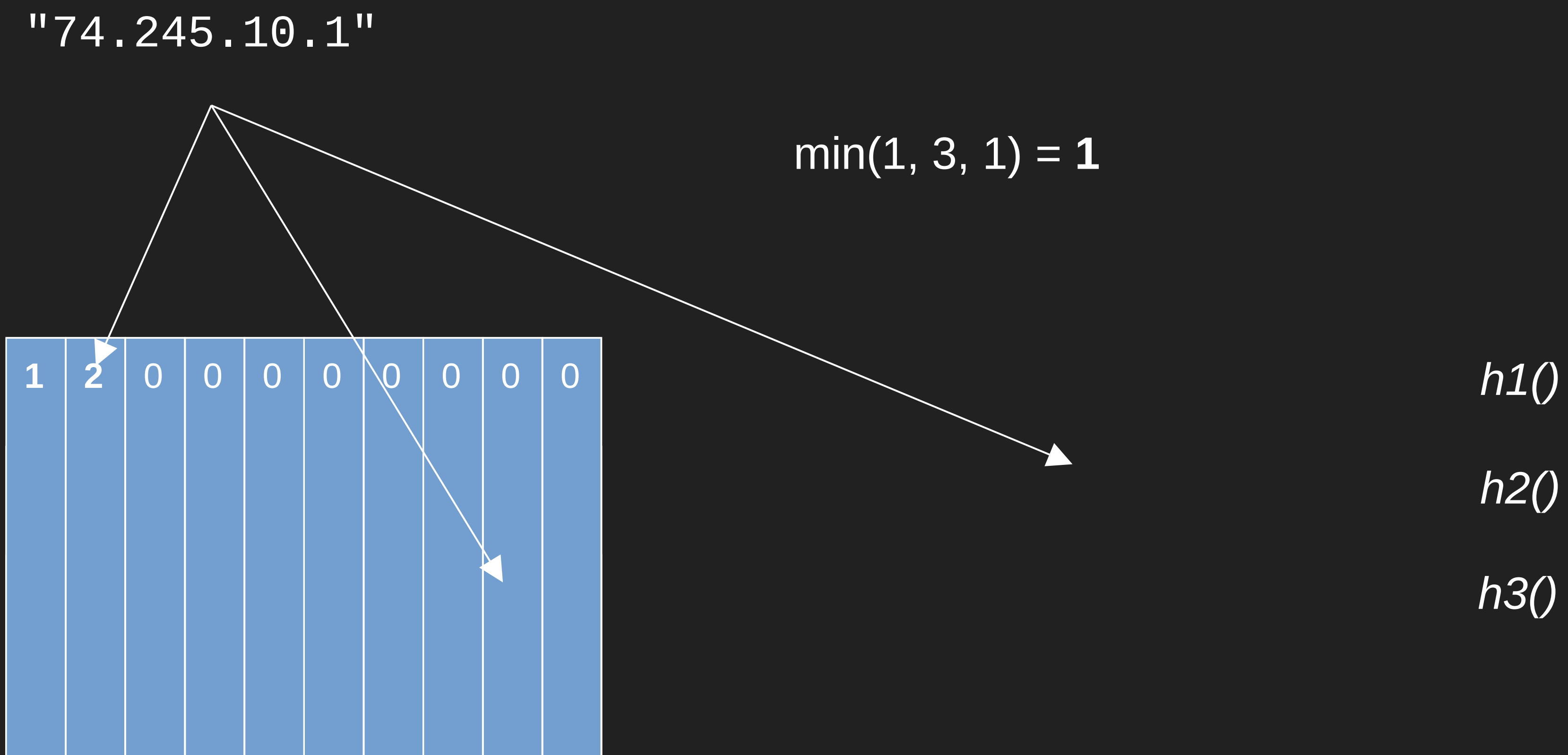
Getting the count of the first element

"192.169.0.1"

min(2, 3, 2) = **2**



Getting the count of the second element



Initialising a sketch

Epsilon: accepted error added to counts with each item

Delta: probability that estimate is outside accepted error

```
this.width = (int) Math.ceil(2 / epsilon);

this.depth = (int) Math.ceil(-Math.log(1 - delta) /
  Math.log(2))
```

```xml
<dependency>
    <groupId>com.clearspring.analytics</groupId>
    <artifactId>stream</artifactId>
    <version>2.9.2</version>
</dependency>
```

```java
private CountMinSketch cms = new CountMinSketch(0.001, 0.99,
1);

cms.add("192.169.0.1", 1);
cms.add("74.245.10.1", 1);
cms.add("10.124.22.19", 2);

cms.estimateCount("10.124.22.19"); // 2
cms.estimateCount("999.999.999.999"); // 0
```

Number of UUIDs	Multiset JVM heap used (MB)	CMS JVM heap used (MB)
10	< 2	N/A
100	< 2	N/A
1,000	3	N/A
10,000	9	N/A
100,000	39	N/A
1,000,000	234	N/A

Epsilon	Delta	Width	Depth	CMS JVM heap used (MB)
0.1	0.99	7	20	0.009
0.01	0.999	10	100	0.02
0.001	0.9999	14	2000	0.2
0.0001	0.99999	17	20000	2.7

Use cases

Any kind of frequency tracking!

NLP

Extension: Heavy-hitters

Extension: Range-query

HyperLogLog

HyperLogLog is for cardinality

```
Set<String> visitors = new HashSet<>();

visitors.add("192.169.0.1");
visitors.add("74.245.10.1");
visitors.add("10.124.22.19");
visitors.add("10.124.22.19");
visitors.add("10.124.22.19");

visitors.size(); // 3
```

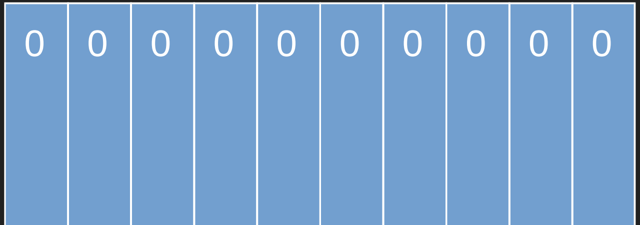
Number of UUIDs	**JVM heap used (MB)**
10	< 2
100	< 2
1,000	3
10,000	9
100,000	37
1,000,000	264

A gentle walk into HyperLogLog

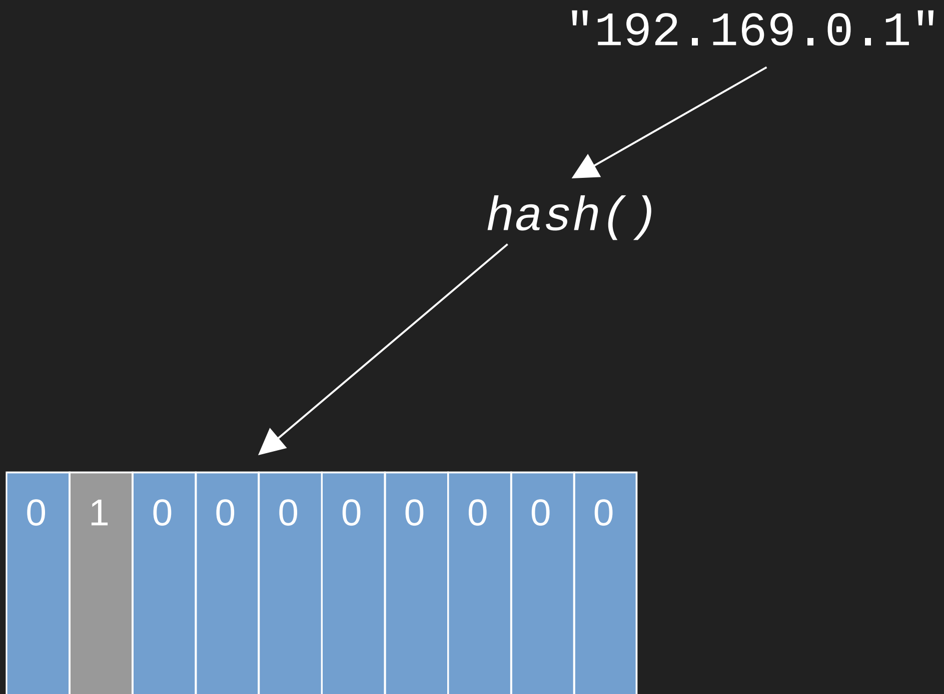
Linear counting

A linear-time probabilistic counting algorithm for database applications (Whang, 1990)

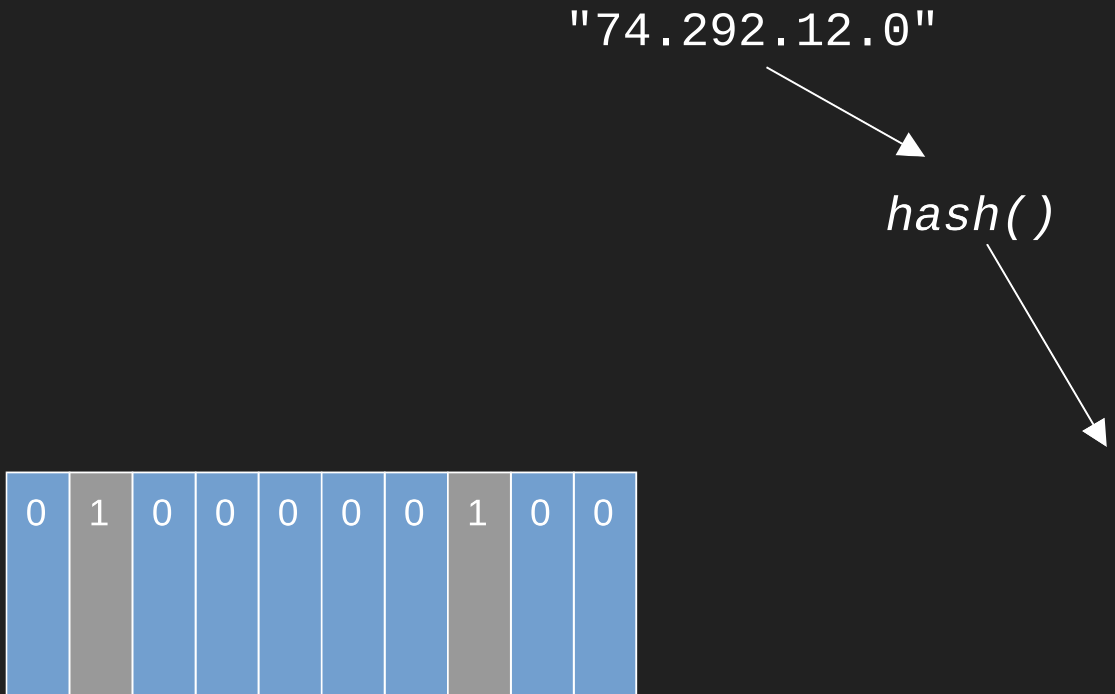
A bit array of size *m*



Adding an element



Adding another element



Estimating cardinality

Cardinality (estimate) = `-m * ln(m-w/m)` = 2.2

0	1	0	0	0	0	0	1	0	0

LogLog

LogLog counting of large cardinalities (Durand, Flajolet, 2003)

Flipping coins

Doing it with hashing

`"192.169.0.1"` → 10011011011 → 0

Doing it with hashing

`"192.169.0.1"` → 10011011011 → 0

`"74.292.12.0"` → 01010011010 → 1

Doing it with hashing

`"192.169.0.1"` →	10011011011 →	0
`"74.292.12.0"` →	01010011010 →	1
`"74.292.12.1"` →	10100011101 →	0

Doing it with hashing

"192.169.0.1" → 10011011011 → 0

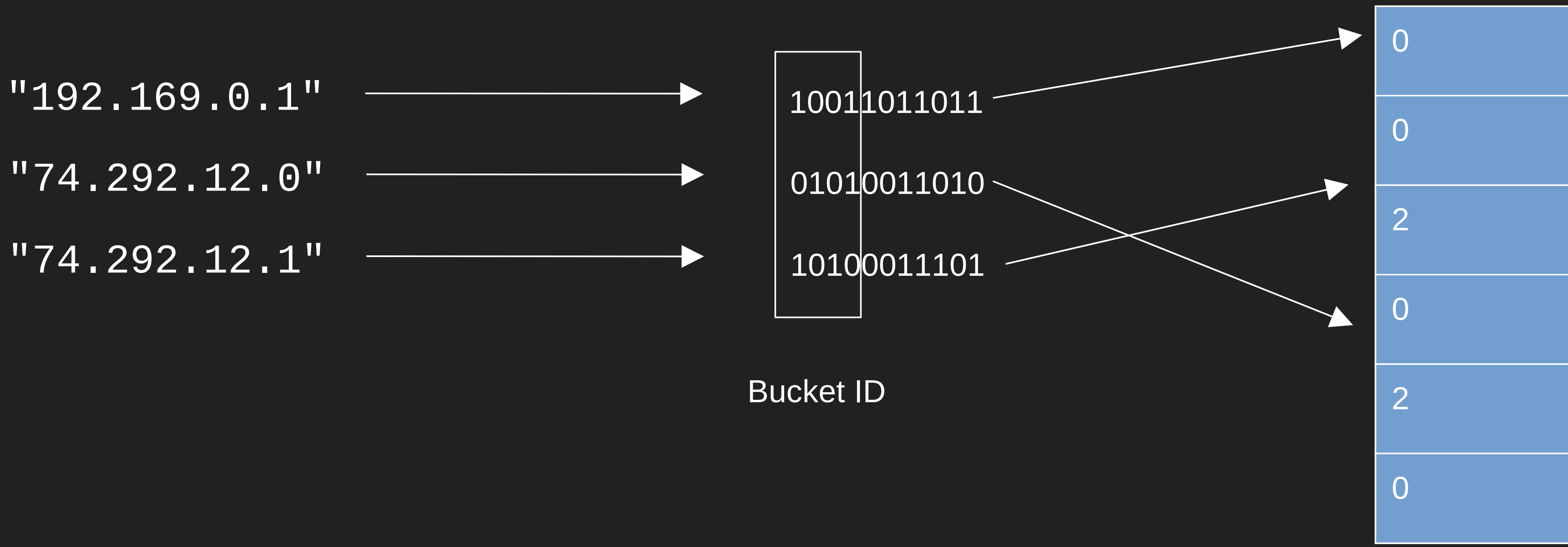
"74.292.12.0" → 01010011010 → 1

"74.292.12.1" → 10100011101 → 0

$$2^n = 2^1 = 2$$

Improving it

Stochastic averaging



$$2^{(\texttt{sum(max\_zeros) / buckets})} \texttt{ * buckets * ESTIMATION\_FACTOR}$$

Average error = 1.3/sqrt(buckets)

$$1.3/\text{sqrt}(1024) = 0.04$$

5 bits per bucket = 5.12K!

HyperLogLog

HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm (Flajolet, Fusy, Gandouet, 2007)

Average error = 1.05/sqrt(buckets)

$$1.05/\mathrm{sqrt}(1024) = 0.032$$

$1.04/\mathrm{sqrt}(1024) = 0.03$

```xml
<dependency>
    <groupId>com.clearspring.analytics</groupId>
    <artifactId>stream</artifactId>
    <version>2.9.2</version>
</dependency>
```

```
HyperLogLog visitors = new HyperLogLog(0.03);

visitors.offer("192.169.0.1");
visitors.offer("74.245.10.1");
visitors.offer("10.124.22.19");
visitors.offer("10.124.22.19");
visitors.offer("10.124.22.19");

visitors.cardinality(); // 3
```

Number of UUIDs	Set JVM heap used (MB)	HyperLogLog JVM heap used (MB)
10	< 2	0.005
100	< 2	0.005
1,000	3	0.005
10,000	9	0.005
100,000	37	0.005
1,000,000	264	0.005

Use cases

Anywhere you need cardinality in O(n)!

Unique site visitors

Estimates of massive tables

Streams of data

That's it!

We learned about

Bloom filters

Count-Min Sketch

HyperLogLog (and some of the story)