# BloomFilters

ESILV

nicolas.travers (at) devinci.fr

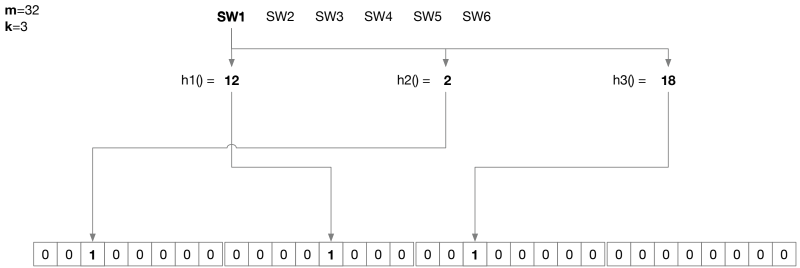
1. BloomFilters
   - Construction
   - Search
   - Properties
   - Extensions

# What is a BloomFilter?

- How to filter efficiently huge files?
- **Bloom Filter**[1]: data structure that says if data should be in
- Rely on a **bitmap** of size $m$ and $k$ distinct **hash functions**
- Each item is hashed $k$ times to set $k$ bits in the bitmap to 1

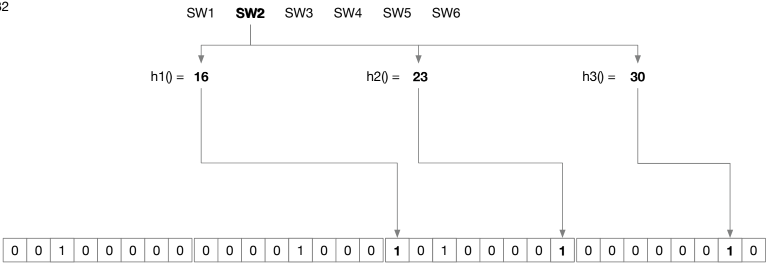
[1] *Burton Howard Bloom* in 1970

# BloomFilters Construction



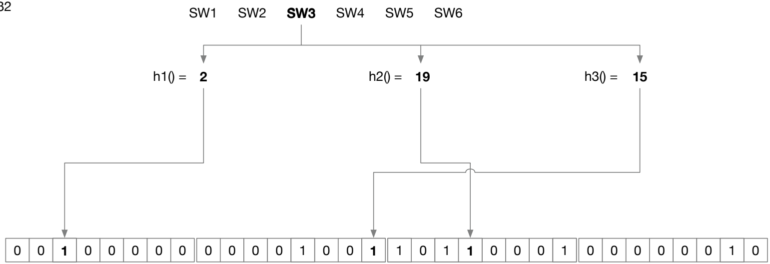
# BloomFilters Construction

**m**=32
**k**=3

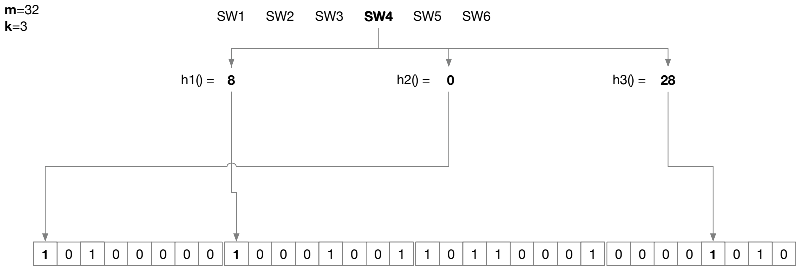


# BloomFilters Construction

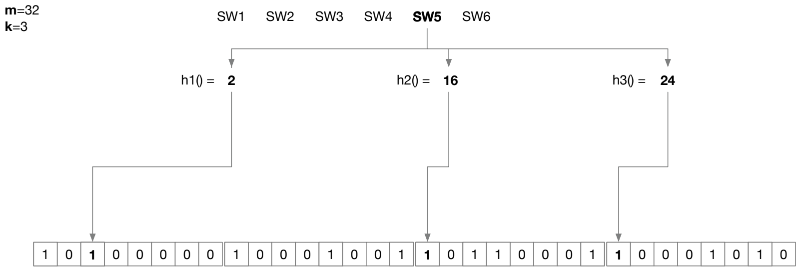
**m**=32
**k**=3



# BloomFilters Construction

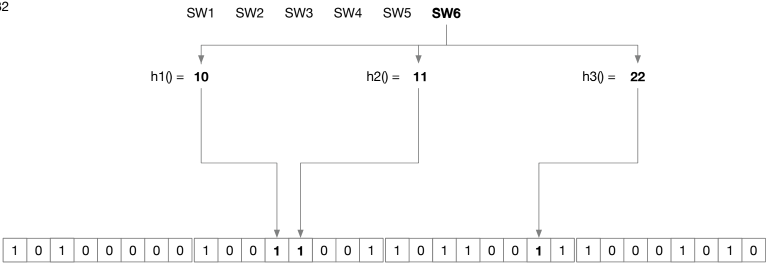


# BloomFilters Construction



# BloomFilters Construction

**m**=32
**k**=3



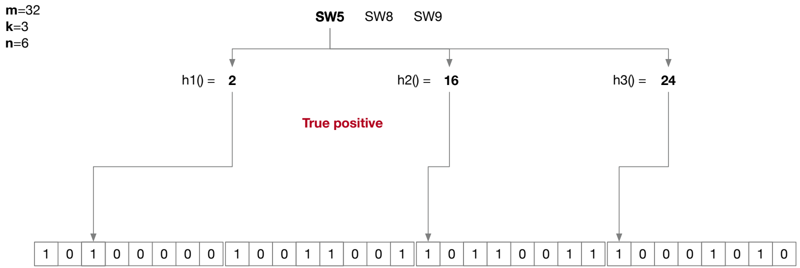
# Properties

- Lightweight: $m$ bits
- Efficient computation: $\mathcal{O}(k)$

# Searching in the BloomFilter

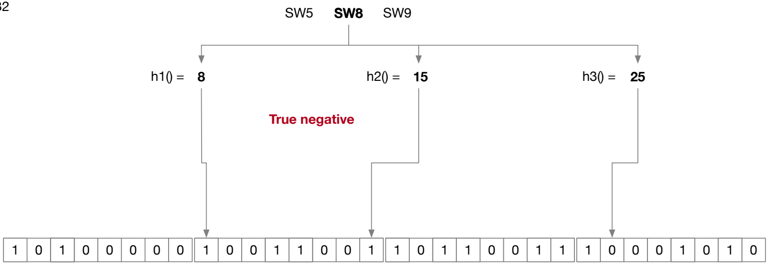
- Same process: $k$ hash values
- If and only if the $\boldsymbol{k}$ bits are *true* $\Rightarrow$ return *true*
  - True positives & True negatives $\Rightarrow$ Filter's the goal!
  - False positives $\Rightarrow$ *should be* in...
  - No false negatives
- Efficient search: $\mathcal{O}(k)$ + double check
  - Reduce unnecessary disk reads

# Search example



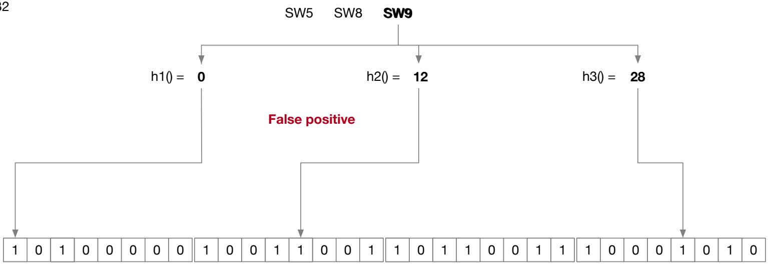
# Search example

**m**=32
**k**=3
**n**=6



# Search example

**m**=32
**k**=3
**n**=6



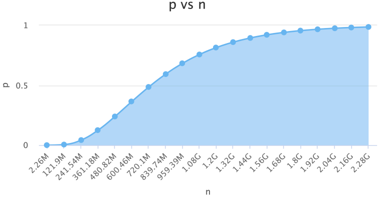
# Properties

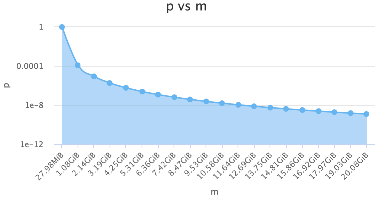
- $n$: Number of indexed values (stored)
- $m$: Bitmap size
  $$m = -\frac{n \ln(p)}{(\ln 2)^2}$$
- $k$: Number of hash functions
  $$k = \frac{m}{n} \ln(2)$$
- $p$: False positive rate
  $$p = (1 - e^{-kn/m})^k$$

$\Rightarrow$ Choose two params to configure (usually $n$ and another)

# Interesting relations

- Bigger collections ⇒ Needs larger bitmaps or get more false positives
- Larger bimaps ⇒ Less false positives or less hash functions
- Less hash functions ⇒ Faster operations...
- Simulator[2] with $n = 256M$, $m = 200MB$ and $k = 4$ (optimal)




[2] https://hur.st/bloomfilter/?n=1M

# BloomFilters extension

- *Counting bloom filters* (use a 4-bit counter instead of 1 bit)

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

- *Locality-sensitive hashing* (LSH): Hashing similar elements into the same bucket with high probability
- *Bloom joins*: Distributed joins
- *Decentralized aggregation*: Disseminator & Aggregator
- *Data synchronization*: Difference of counting BloomFilters
- *Layered Bloom Filters/HyperLogLog*: duplicate detection

# Usage

- *Programming languages*: `myCollection.mightContain(myObject)`
- *Databases*: PostgreSQL, BigTable, Cassandra, HBase
- *Web*: Check URL / cache control
- *Bitcoin*: Wallet synchronization
- *Ethereum*: Quick find logs in the blockchain
- *Biology*: Ckeck DNA sequences