# Analysis of Massive Data Sets

http://www.fer.hr/predmet/avsp

**Prof.dr.sc. Siniša Srbljić**

**Doc.dr.sc. Dejan Škvorc**

**Doc.dr.sc. Ante Đerek**

Faculty of Electrical Engineering and Computing
Consumer Computing Laboratory

# Detection of near-duplicate documents using locality sensitive hashing

*Otkrivanje sličnih dokumenata koristeći sažimanje neosjetljivo na lokalne promjene*

**Klemo Vladimir**

# Outline

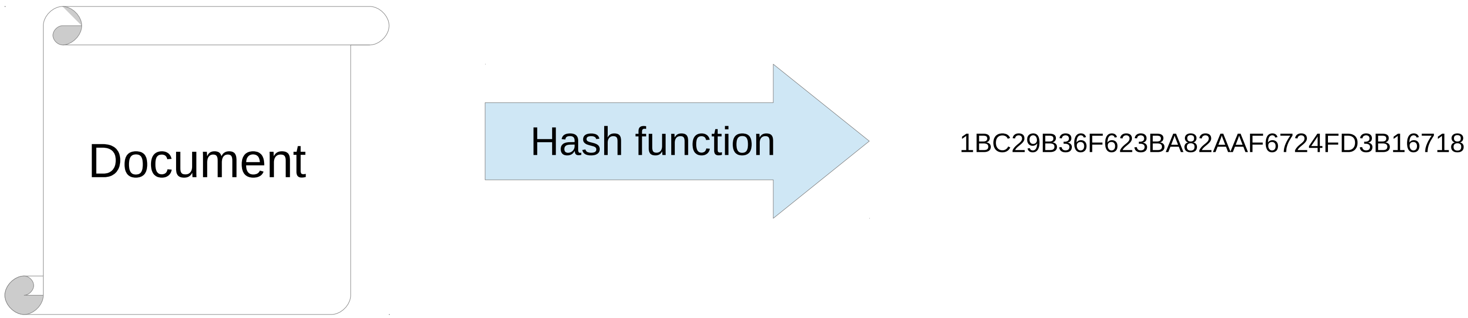
- Motivation
- Fingerprinting
  - Simhash algorithm
  - Rabin's rolling hash function
- Scalable queries
  - MapReduce sketch
- Literature

# Motivation

- Near-duplicate documents
  - Versioning
    - Different versions of single document
      - Revisions, different file formats, ...
  - Mirroring
    - published in more than one location
  - Plagiarism
    - Exact or “processed” copy
  - Malware
    - Viruses, spam, …
- Scalability
  - Documents and document respositories are large

# Exact copy analysis

- Checksumming
  - Cryptographic hash functions
  - MD5, SHA1, SHA2, ...




  - Catches the smallest edit
    - Great for detection of **exact** copies
    - Not so good for near-duplicate detection
      - Even the smallest change will result in totally different digest

# Near-duplicate detection

- Two families of methods
  - 1. *Fingerprinting*
    - Document hashing [*1]
    - Dimensionality reduction
  - 2. Ranking
    - Information retrieval techniques [*2]
    - High-dimensional vectors manipulation

# Fingerprinting

- Similarity preserving hashing
  - X, set of inputs
  - $d_x$, distance function on X
    - $x_1$, $x_2$ elements of X
  - similarity preserving hash function
    - **h: X→Y**
    - |Y| < |X|
  - $d_y$, distance function on Y

# Fingerprinting

- Similarity preserving hashing
  - Similar inputs have similar hashes

$$\text{if } d_x(x_1, x_2) < \varepsilon_x, \text{ then}$$
$$d_y(h(x_1), h(x_2)) < \varepsilon_y$$

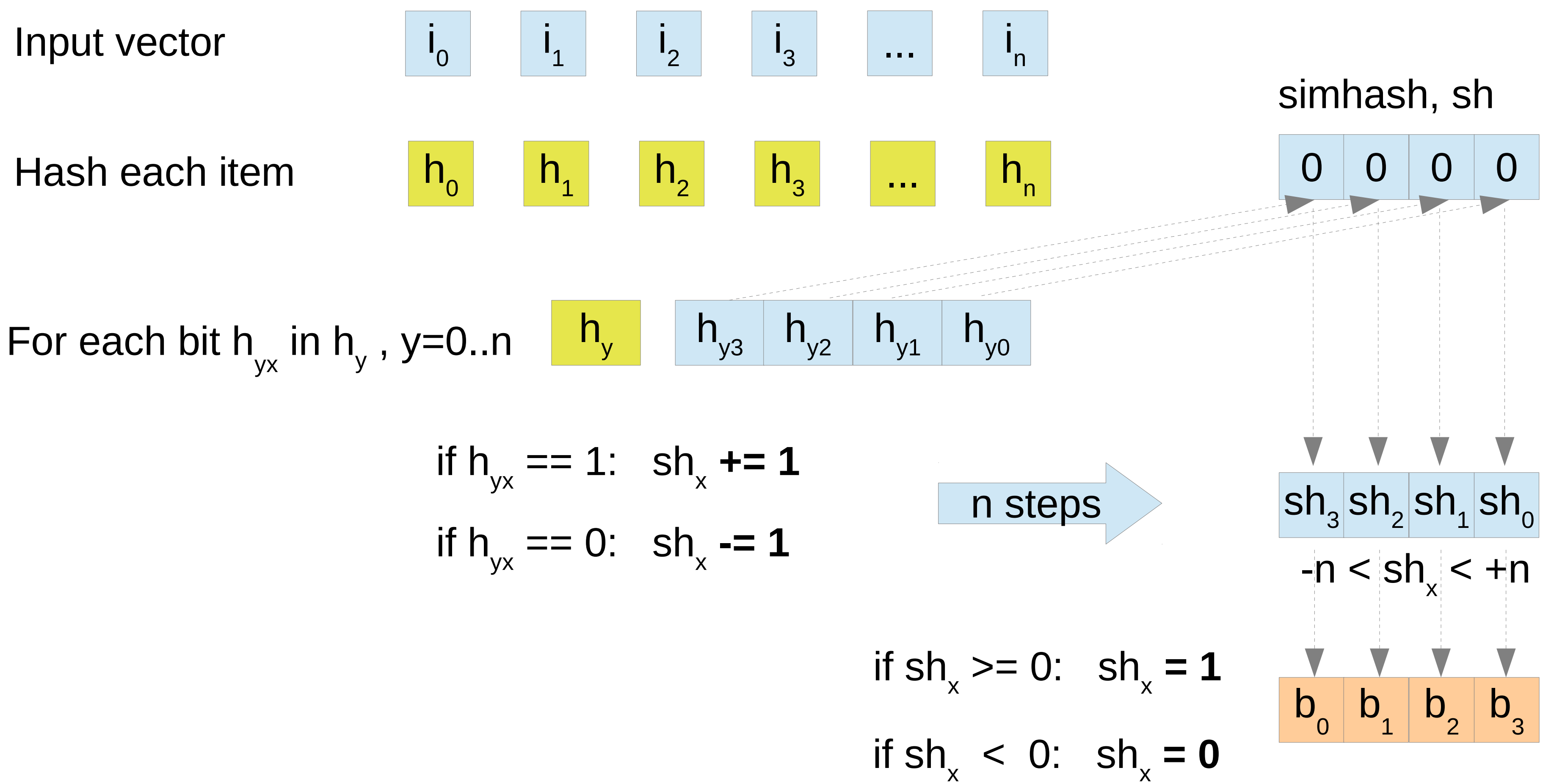
- Illustrative example
  - `h("1234") = 0xaaaf  h("5678") = 0xb115`
  - `h("1234") = 0xaaaf  h("1235") = 0xaaae`

# Fingerprinting

- **Simhash** algorithm [*3]
  - Author: M. Charikar, 2002
  - Fingerprinting technique
    - Fingerprints of near-duplicates differ in a small number of bit positions (**hamming** distance)
  - Dimensionality reduction
    - Maps high-dimensional vectors to small-sized fingerprints (f-bits)
  - Input
    - High-dimensional vector (strings, numbers, …)
  - Output
    - f-bit fingerprint
  - Fingerprint size (f-bits)
    - f is small and arbitrary
    - eg. f=64 for web sites

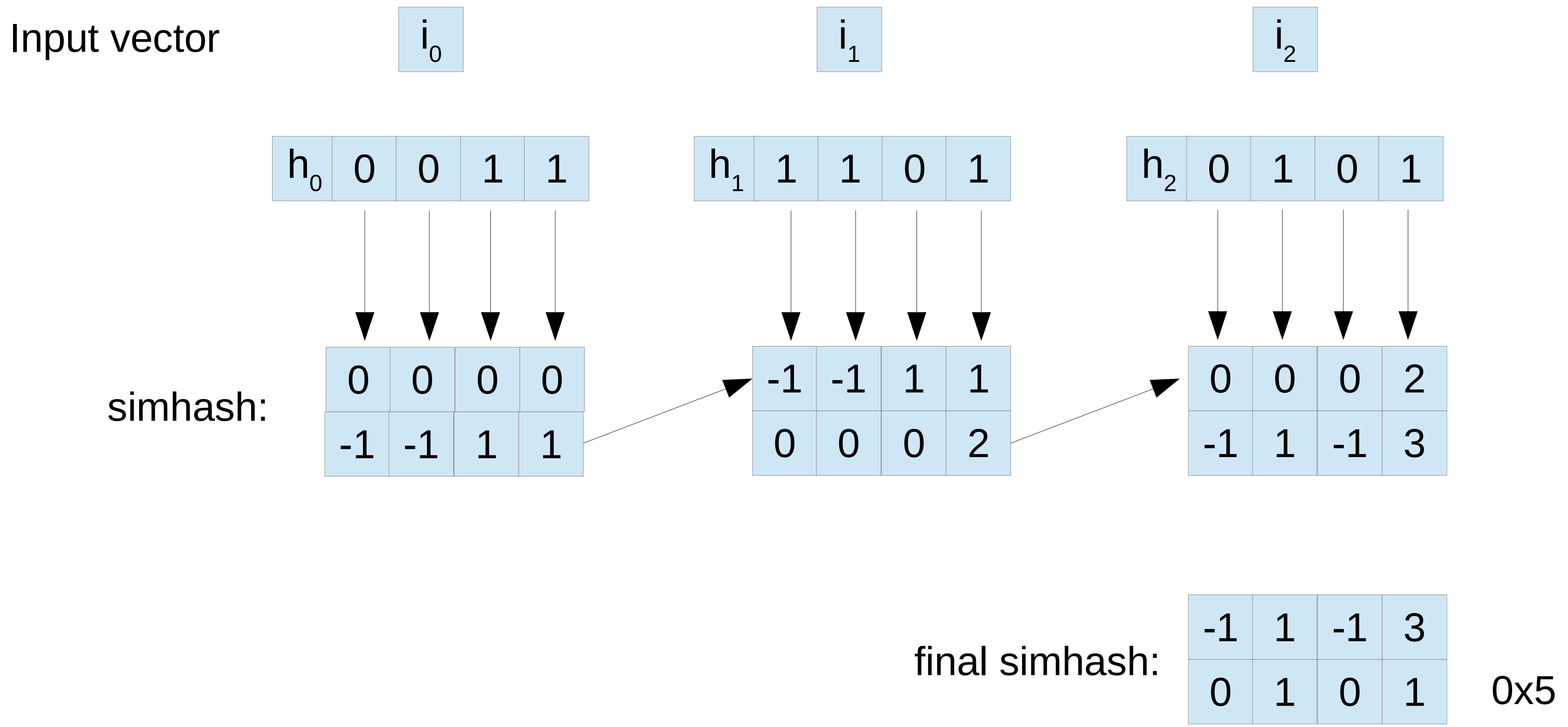
# Simhash

- Simhash computation, f=4



# Simhash

- Simhash example, f=4



# Simhash

- Weighted Simhash computation
  - Assign weight factor to each feature

| Input vector | $i_0$ | $i_1$ | $i_2$ | $i_3$ | ... | $i_n$ |
|---|---|---|---|---|---|---|
| | $w_0$ | $w_1$ | $w_2$ | $w_3$ | | $w_n$ |

if $h_{yx}$ == 1: $sh_x$ **+= $\mathbf{w_y}$**

if $h_{yx}$ == 0: $sh_x$ **-= $\mathbf{w_y}$**

# Simhash

- Choice of hash function *h*
  - Uniform distribution
  - Fast
  - Candidates
    - Cryptographic hash functions
      - MD5 (128-bit), SHA-1 (256-bit)
      - Problem: cryptographic hashing is slow
    - Rolling hash functions
      - Input is hashed by moving window element by element

# Simhash

- Choice of hash function *h*
  - Rabin rolling hash
    - Used in Rabin-Karp string searching algorithm
    - Input: string
    - *h(k) = kmodq*, q is some large prime number
  - Computation
    - substring coded as a number with base **d**
      - d = total number of possible characters
    - Coded string at position *i*:

$$x_i = s[i]d^{k-1} + s[i+1]d^{k-2} + \ldots + s[i+k-1]$$

# Simhash

- Choice of hash function *h*
  - Rabin rolling hash
    - Example: k=4, d=32
    - i=0

| 0 | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| l | o | r | e | m | | i | p | s | u | m |
| l | o | r | e | | | | | | | |

h(“lore”) = $h(x_o) = x_o \bmod q$

$x_o = \text{int}(‘l’)32^3 + \text{int}(‘o’)32^2 + \text{int}(‘r’)32 + \text{int}(‘e’)$

$x_{i+1}$=?

# Simhash

- Choice of hash function *h*
  - Rabin rolling hash
    - Example: k=4, d=32
    - i=1

| 0 | 1 | 2 | 3 | 4 | | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| l | o | r | e | m | | i | p | s | u | m |
| | o | r | e | m | | | | | | |

h(“orem”) = $h(x_1) = x_1 \bmod q$

$x_1 = (x_0 - \text{int}(\text{‘l’})d^3)d + \text{int}(\text{‘m’})$

$$\mathbf{x_{i+1} = (x_i - s[i]d^{k-1})d + s[i+k]}$$

# Simhash

- Choice of hash function *h*
  - Rabin rolling hash
    - Fingerprints represented using polynomials
    - Computes hash value of the next string from the previous one
      - Constant number of operations
      - Independent of string length

# Simhash

- Input vector
  - Focus on raw text documents
  - Convert document to a feature vector
  - Feature extraction
    - Tokenization
      - Unigram, 2-gram, 3-gram, ...
    - Stemming
    - Stopword removal
    - Phrase detection

# Simhash

- Tokenization examples

“lorem ipsum dolor sit amet”

**1. word tokens**

“lorem”, “ipsum”, “dolor”, “sit”, “amet”

**2. 2-word tokens**

“lorem ipsum”, “ipsum dolor”, “dolor sit”, “sit amet”

**3. character 3-grams**

“lor”, “ore”, “rem”, “em ”, “m i”, …

# Simhash

- Shingle
  - hash of k-gram
    - k-grams
      - Characters, words, phrases, sentences (or combination)
    - k = ?
      - Small k: dissimilar documents appear similar
      - Large k: similar documents appear dissimilar
  - Feature vector from IR output
    - Weighted (TF) with IDF (inverse document frequency)
    - Might change when collection changes!

# Fast Queries

- **F**, collection of f-bit fingerprints
- **Q**, query
  - single or set of fingerprints

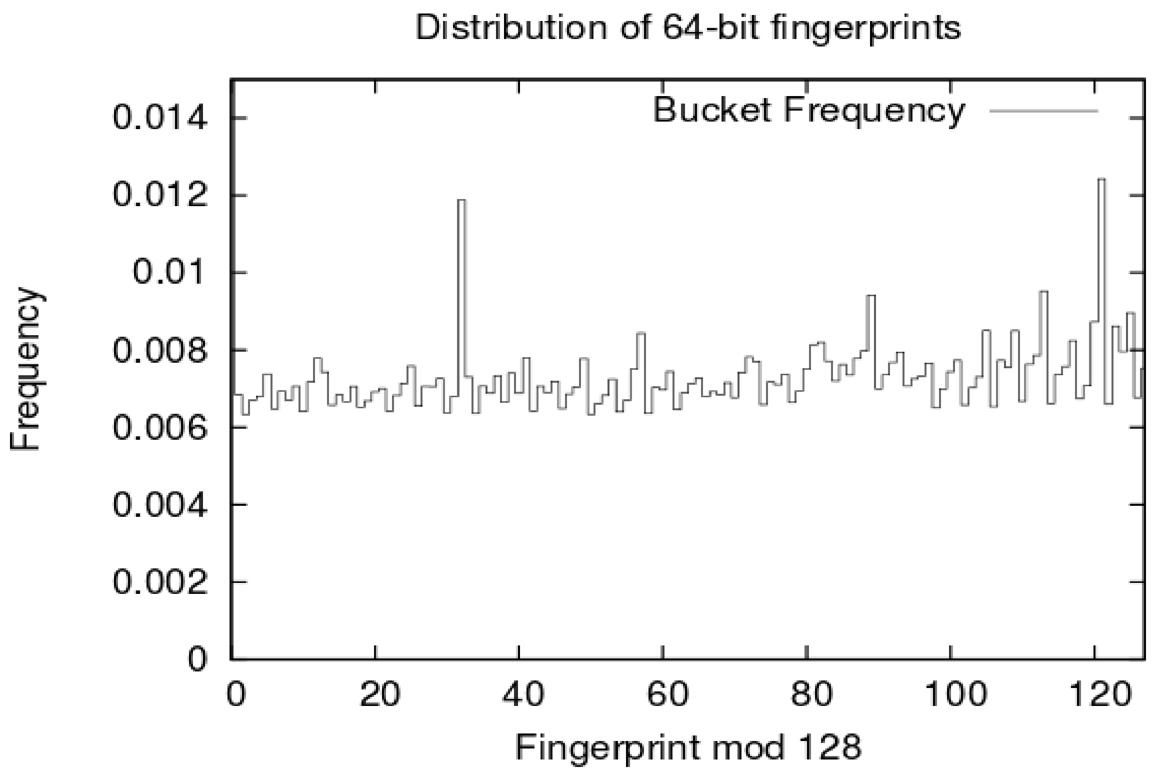
- Task
  - identify whether **Q** differs from any of the fingerprints in ***F*** in at most **k** bits

# Fast Queries

- Google numbers
  - 8B 64-bit fingerprints = 64GB
  - Online query
    - Q = single fingerprint
    - Restriction: few milliseconds
  - Batch query
    - Q = set of fingerprints
      - e.g. |Q| = 1M
    - Restriction: ~100seconds
      - 1B queries per day

# Simhash

- Distribution of fingerprints
  - 8B 64-bit fingerprints



# Fast Queries

- 1. approach
  - Build sorted table of **F**
  - Build list **Q'** with all fingerprints whose Hamming distance from **Q** is at most **k**

| Q |
|---|
| 100 |

k=1

| Q' | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |

| F | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

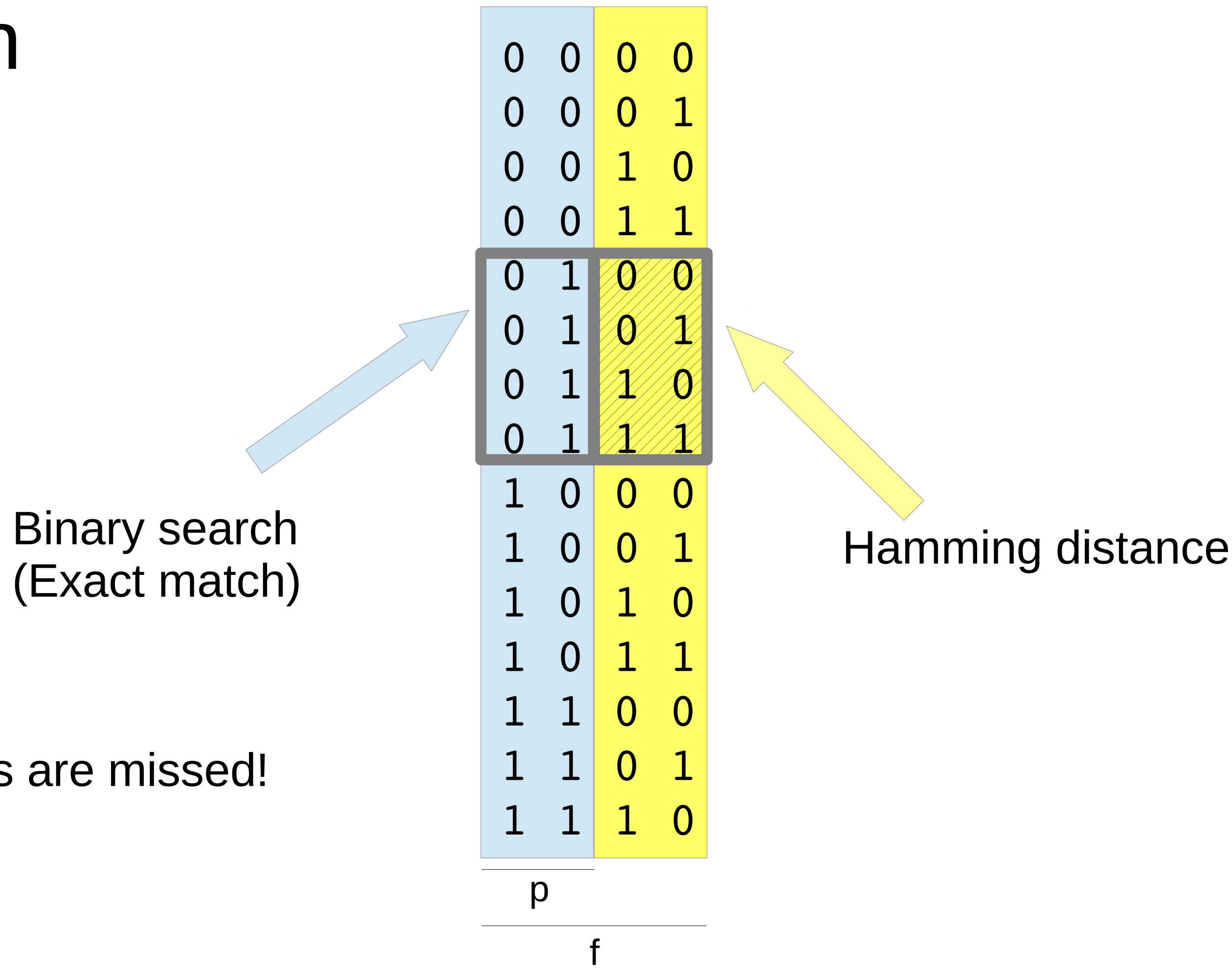
  - 8B 64-bit fingerprints (k=3)
    - $|Q'| = \binom{64}{3} = 41664$

# Fast Queries

- 2. approach
  - 1. Build sorted table of **F**
  - 2. Find set of fingerprints (**F'**) that have equal most significant part (p bits)
    - Sorted table – binary search O(p)
  - 3. Check Hamming distance for each fingerprint in **F'**
  - This approach will locate all fingerprints in **F** that differ in at most k bits
    - **Restricted to least significant f-p bits!**

# Fast Queries

- Illustration




Since $p < f$:
some true positives are missed!

# Fast Queries

- Increasing precision/recall
  - Generate additional table
    - Reversed positions

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |

+ sort

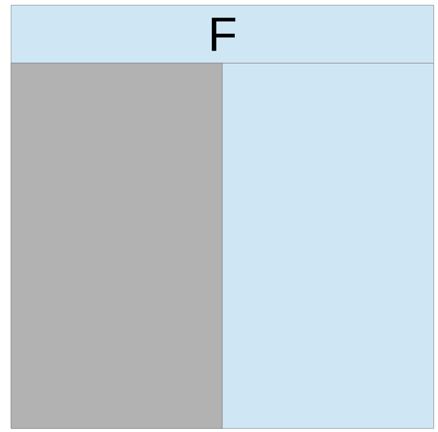
# Fast Queries

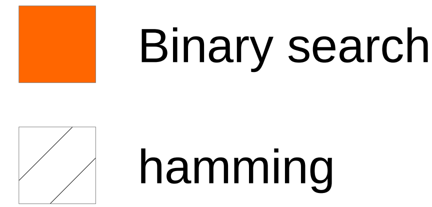
- Solution
  - Build additional tables
    - Each with *different permutation* of bits
    - Every table has different set of significant bits

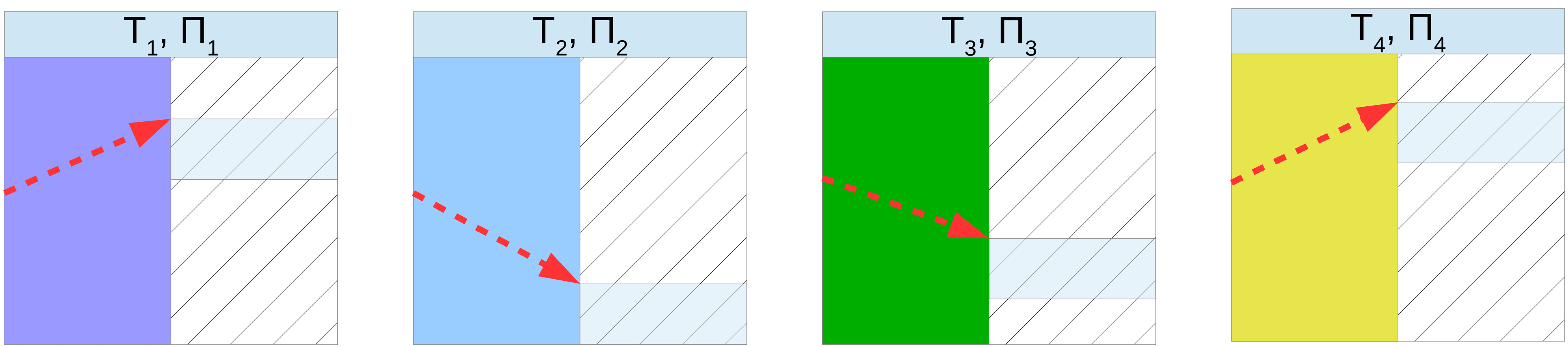
- Algorithm for fast (online) queries
  - Build **t** sorted tables of fingerprints: $T_1$, $T_2$, …, $T_t$
  - Each table $T_i$ also contains
    - $p_i$ – number of significant bits
    - $\Pi_i$ – random permutation
  - *Every fingerprint in $T_i$ is permuted with permutation $\Pi_i$*

# Fast Queries

- Illustration
  - Build tables
  - Query in parallel



# Fast Queries

- For given **Q** and **k**
  - Read each table (in parallel)
    - 1. Get fingerprints in $T_i$ whose significant $p_i$ bits match the significant $p_i$ bits of $\Pi_i(Q)$
      - $T'_i$
      - $O(p_i)$ steps (binary search)
    - 2. For each fingerprint in $T'_i$, check if it's Hamming distance is at most k bits from $\Pi_i(Q)$

# Fast Queries

- Example with t=20, f=64, k=3, |F| = 8B ($2^{34}$)
  - Split f into 6 blocks (4x11 + 2x10 bits)
  - Select 3 out of 6 blocks $\binom{6}{3}$ = 20 ways
  - Arrange those blocks as significant bits
  - p = sum of those bits
    - 31, 32, or 33
  - On average query returns $2^{34-31}$=8 fingerprints

# Fast Queries

- t and p parameters
  - t ~ p
  - Query time ~ 1/p
  - Storage requirements ~ p
  - Space/time tradeoff
    - Analytical solution for t [*1]

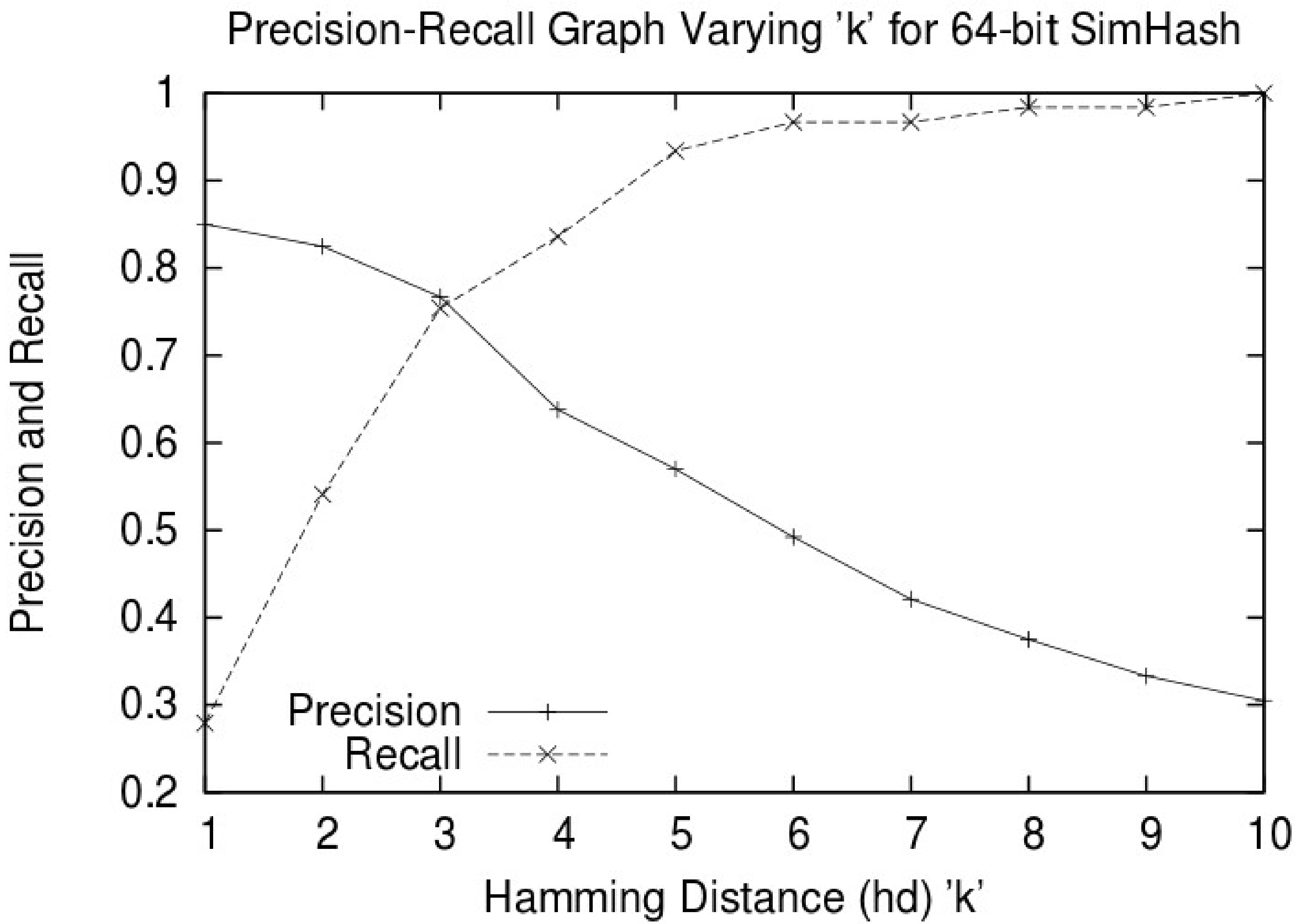
# Fast Queries

- Batch queries using **MapReduce** and **GFS**
  - F and Q are files in the GFS (with replication)
  - F ~ 64GB, Q ~ 8MB
  - F is stored in GFS chunks
  - Number of mappers = number of F chunks
  - Map:
    - Solves Hamming distance for chunk (64MB) and emits list of near-duplicates
  - Reduce
    - Remove duplicates

# Experimental results

- Detecting near-duplicate web pages
  - Web-crawling in Google
- Database
  - 8B fingerprints, k=1..10
- Manually tag experimental data set
  - True/false positive/negative
- Precision/recall graph
  - Precision: #tp / # returned results
  - Recall: #tp / # expected results

# Experimental results



# Papers


- [1] Manku, Gurmeet Singh, Arvind Jain, and Anish Das Sarma. **"*Detecting near-duplicates for web crawling.*"** In Proceedings of the 16th international conference on World Wide Web, pp. 141-150. ACM, 2007.
- [2] Hoad, Timothy C., and Justin Zobel. **"*Methods for identifying versioned and plagiarized documents.*"** Journal of the American society for information science and technology 54, no. 3 (2003): 203-215.
- [3] Charikar, Moses S. **"Similarity estimation techniques from rounding algorithms."** In Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 380-388. ACM, 2002.
- [4] Henzinger, Monika. "**Finding near-duplicate web pages: a large-scale evaluation of algorithms**." Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2006.