U LISBOA | UNIVERSIDADE DE LISBOA

Ciências ULisboa

LASIGE

# Getting Started on LSH

Vinicius Vielmo Cogo

Navtalks – November 18, 2016

# ~~Gentle~~ Introduction

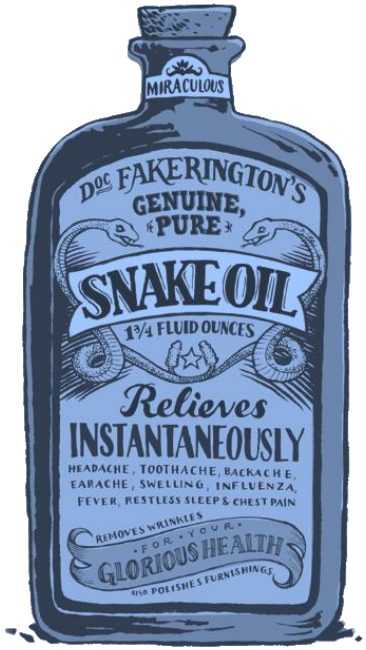
# Locality Sensitive Hashing

SPOILER ALERT!

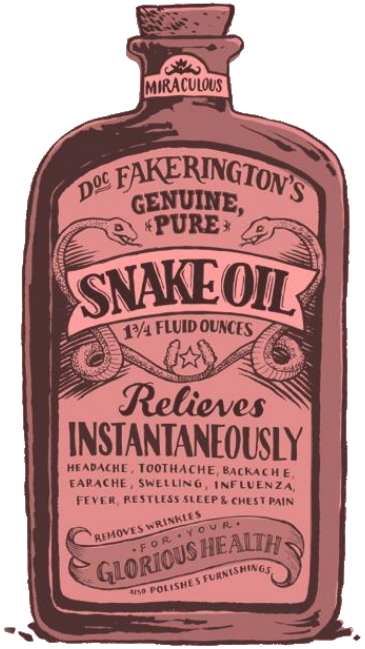
LSH is an efficient algorithm
to **find similar objects using hashes**

# Recommendation Algorithms

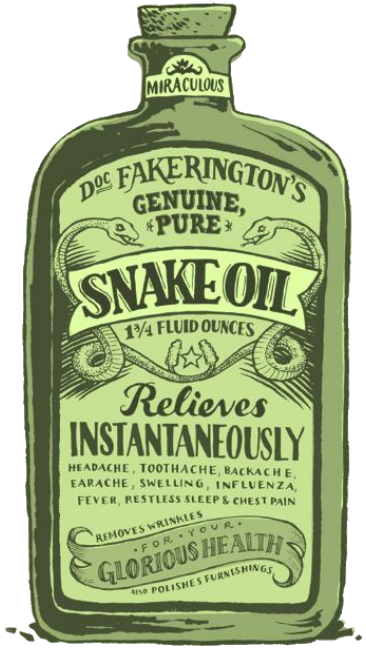
## Customers who bought this object also bought …




★★★★★ 50€




★★★★ 35€




★★★ 20€

# Recommendation Algorithms


→ … **people** you may know


→ … **people** you may like


→ … **videos** you may like


→ … **movies** you may like


→ … **music** you may like


→ … **products** you may like

# Recommendation Algorithms

Example:
Suggest something to users.

1) Find **similar users** (with similar preferences)
Comparing the list of things they like

2) Suggest what one likes and the other doesn't know yet

# Example: Music Recommendation

P1


P2


P3

# Example: Music Recommendation

P1


P2


P3


Jaccard Distance*

$$\text{Similarity} = \frac{|A \cap B|}{|A \cup B|}$$

* See extra slides for other distances

# Example: Music Recommendation

P1

A B C D E

$$\frac{|P1 \cap P2|}{|P1 \cup P2|} = \frac{4}{6} = 0.667$$

P2

A B D E F

$$\frac{|P2 \cap P3|}{|P2 \cup P3|} = \frac{1}{8} = 0.125$$

P3

A V X Y Z

$$\frac{|P1 \cap P3|}{|P1 \cup P3|} = \frac{1}{8} = 0.125$$

# Example: Music Recommendation

P1


P2


P3


(P1, P2) are more similar than (P1, P3) and (P2, P3)

Suggestions:
F -> P1
C -> P2

# Problems

**Millions of users** that listen
**thousands** different **songs** each

# Problems

Millions of users:

**Users** are the **objects** to **compare** – $O(n^2)$

Inserting a **new** user = compare 1 to all

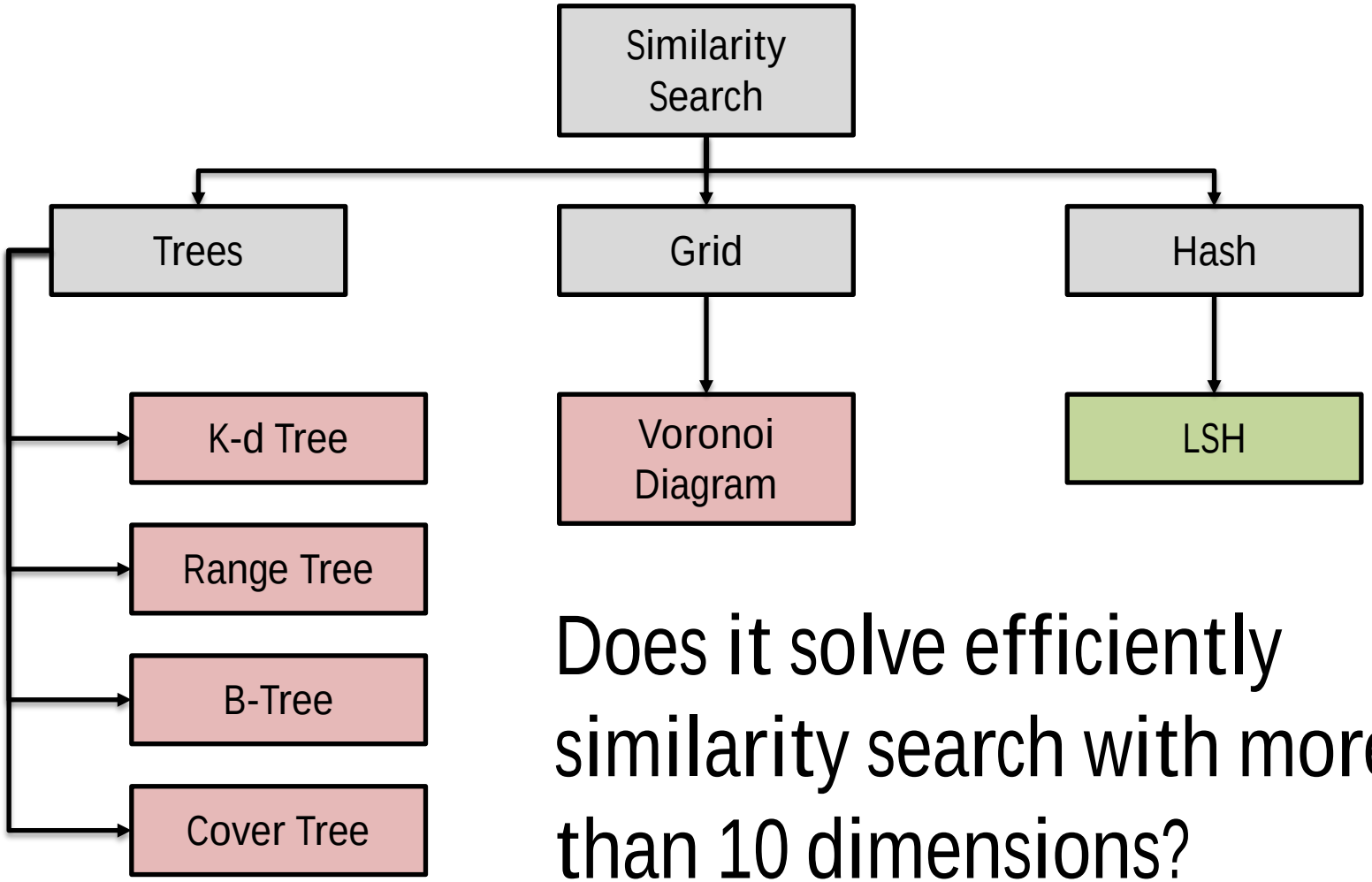
Thousands songs:

Each **song** is a **dimension*** to compare

**Curse of dimension**

* See extra slides for other dimension examples

# Problems




Does it solve efficiently similarity search with more than 10 dimensions?

# Locality Sensitive Hashing

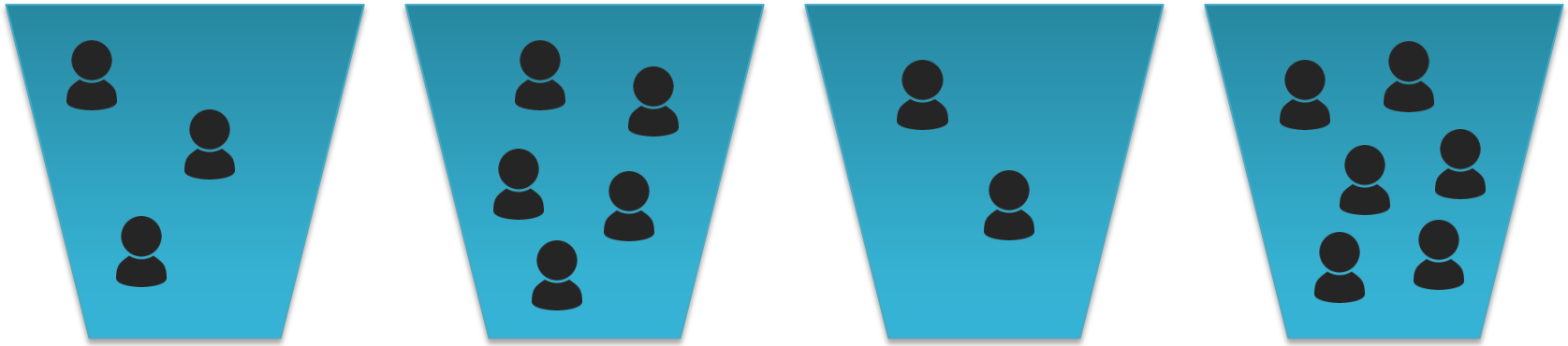
NOT A SPOILER ANYMORE!

LSH is an efficient algorithm
to **find similar objects using hashes**

# Locality Sensitive Hashing

Cluster similar objects into hash buckets

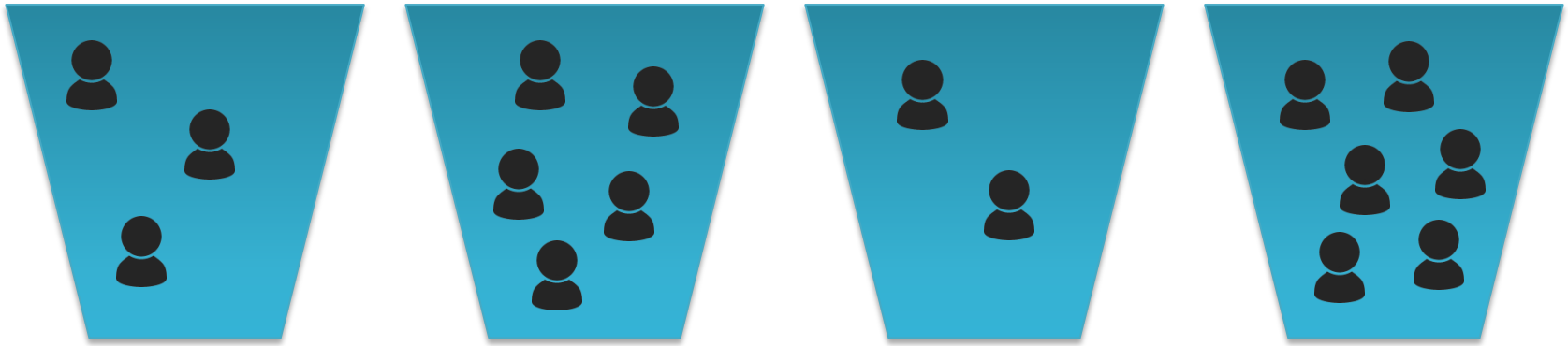
(with a similarity threshold)



# Locality Sensitive Hashing

Crypto hashes:
Similar objects -> very different hashes

Locality Sensitive Hashing:
**Similar objects -> similar hashes**



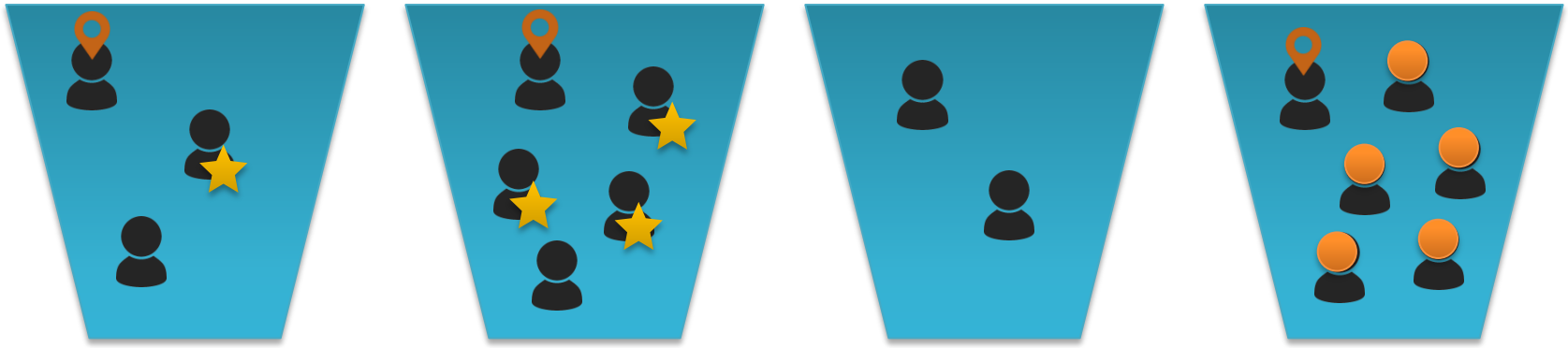
# Locality Sensitive Hashing

Calculate the distance between objects within the same bucket only

# Locality Sensitive Hashing

Queries (search similarity):

- Nearest neighbor
- Near neighbors
- Clustering



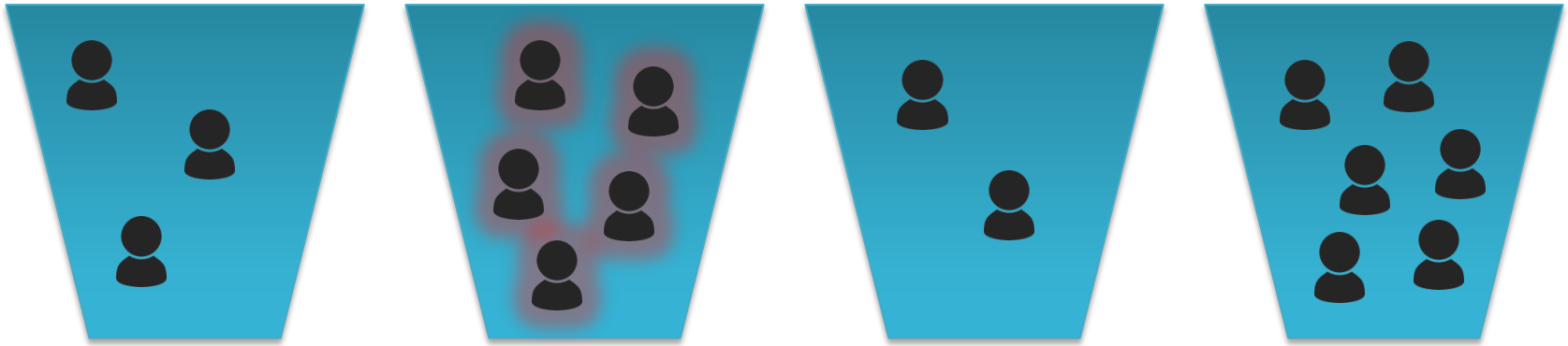
Nearest neighbor

Near neighbors

Clustering

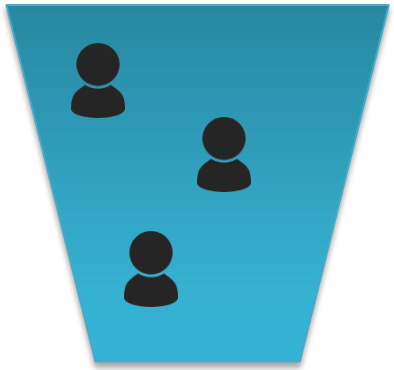
# Locality Sensitive Hashing

We need to calculate the distance within a bucket to validate the distances

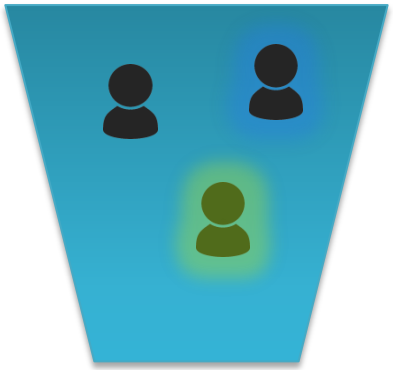


# Locality Sensitive Hashing

False positives and false negatives may happen (configurable)



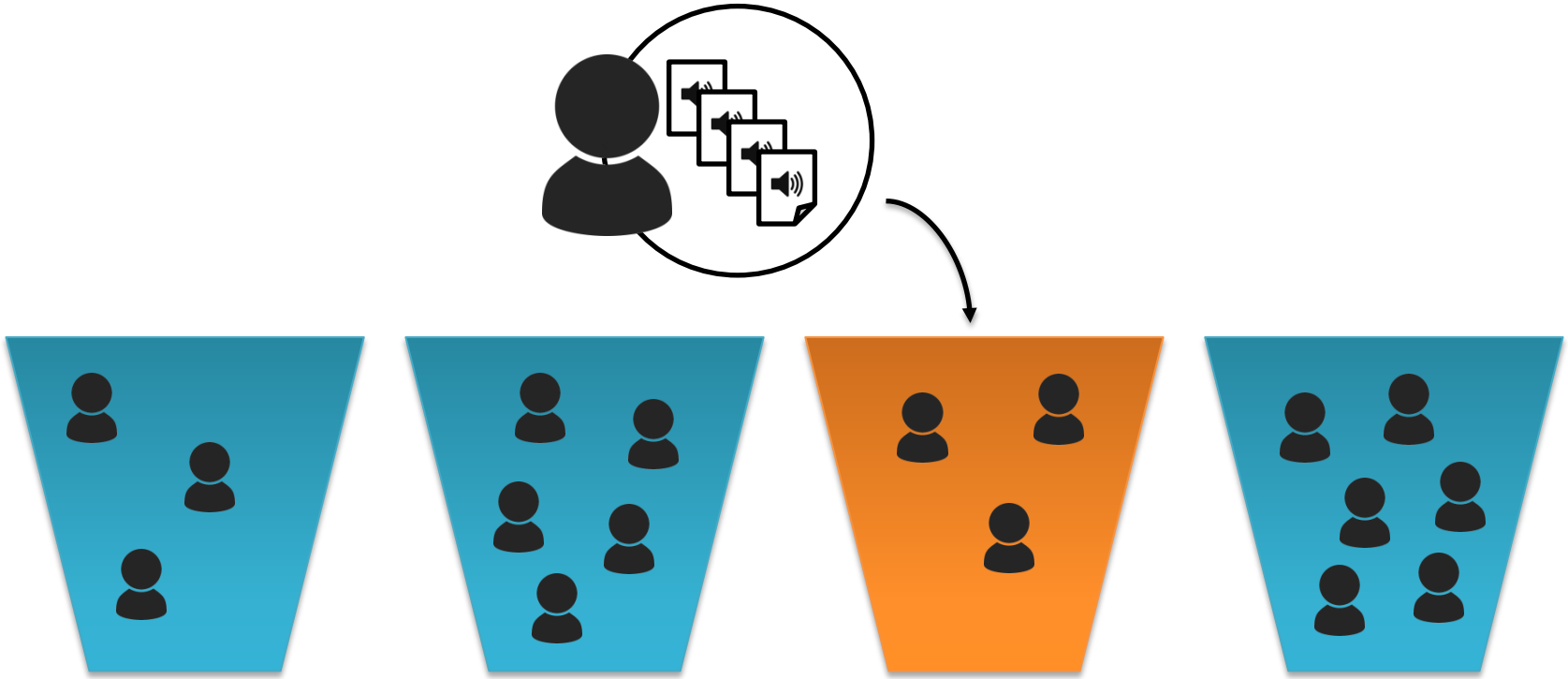
False positives
Computing overhead



False negative
Reduces recall

# Locality Sensitive Hashing

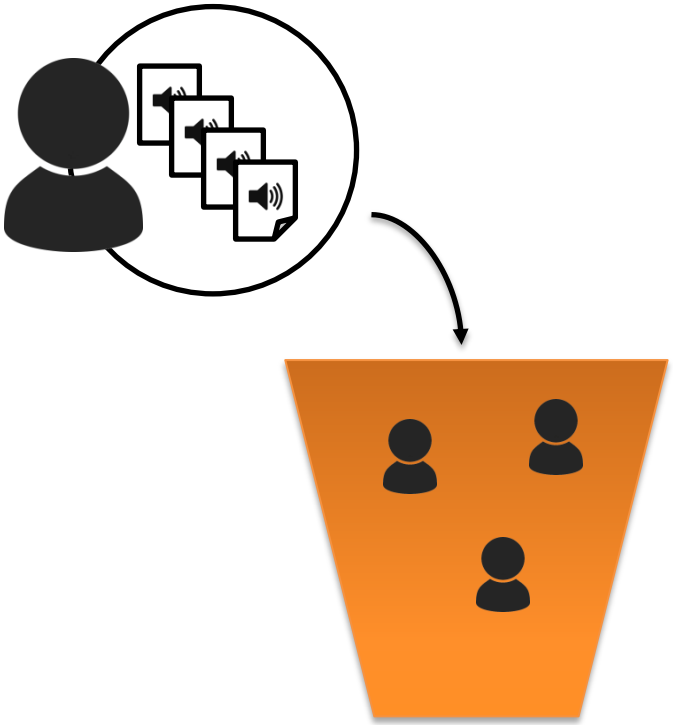
Insert: hash dimensions of new object + compare within bucket



# Locality Sensitive Hashing

LSH = MinHash + MultiMap[ ]

# MinHash



Hashing is crucial!

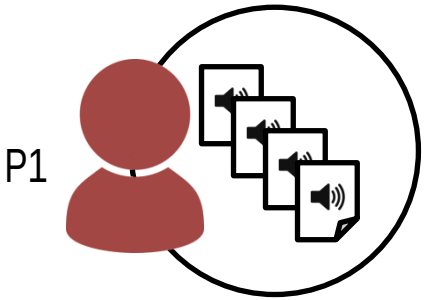
For each distance
there is a different hash family*

Jaccard Distance -> MinHash*

* See extra slides for other hash families

# MinHash

- An array with the minimal hashes from all dimensions for each hash function



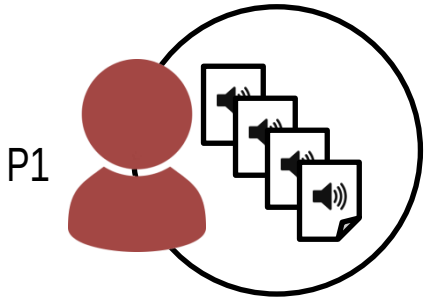

minHash(P1)

| 23 | 12 | 5 | 45 | 7 | 34 | 12 | 76 | 87 | ... |
|---|---|---|---|---|---|---|---|---|---|

```
minHash(Object o):
     num_hashes <- 200 //defined based on a similarity threshold
     hashes <- new hash[num_hashes] //200 different hash functions
     minHash <- new int[num_hashes] //MinHash of the object
     for i in 0..hashes: // for each hash function
          for d in object.dimensions: //for each dimension (music)
                    hi <- hashes[i](d) //calculate the hash of d
                    minHash[i] <- min(minHash[i], hi) // store the min
```

- Converts **variable** number of dimensions **to** a **fixed** configurable number
- Using the **same order** of hash functions is important **to find similar** objects

# Locality Sensitive Hashing



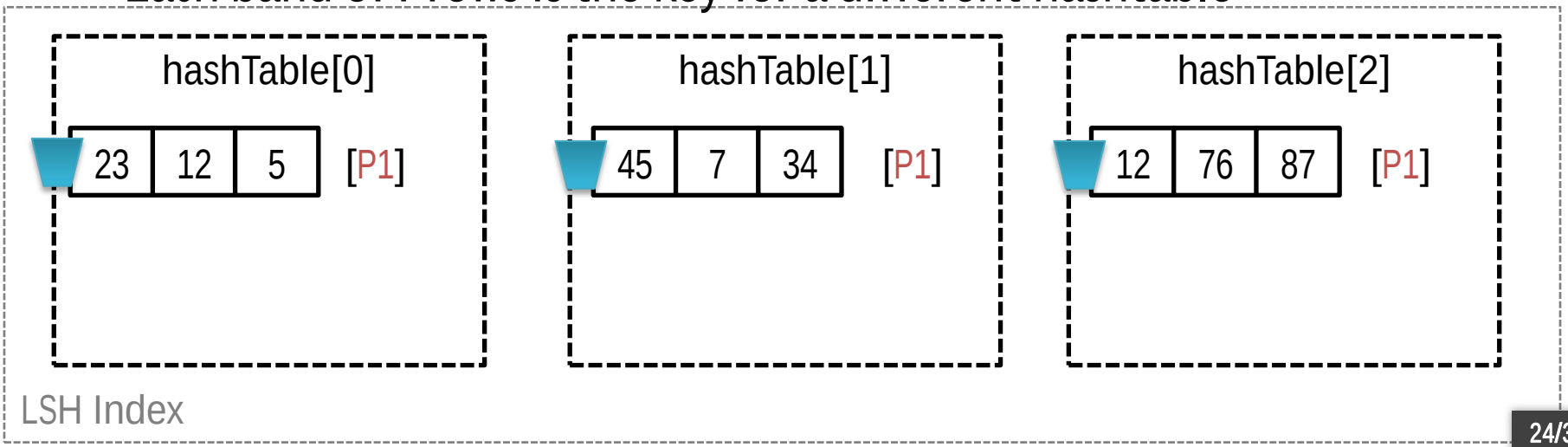

minHash(P1)

| 23 | 12 | 5 | 45 | 7 | 34 | 12 | 76 | 87 | ... |
|---|---|---|---|---|---|---|---|---|---|

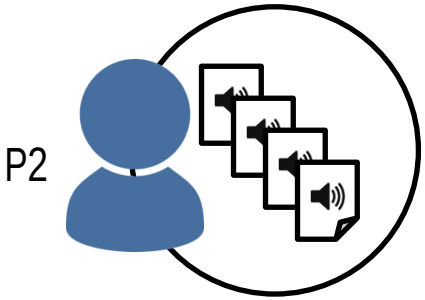
break it into b bands and r rows (based also on the desired similarity threshold)

| 23 | 12 | 5 | | 45 | 7 | 34 | | 12 | 76 | 87 | | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Each band of r rows is the key for a different hashtable



# Locality Sensitive Hashing



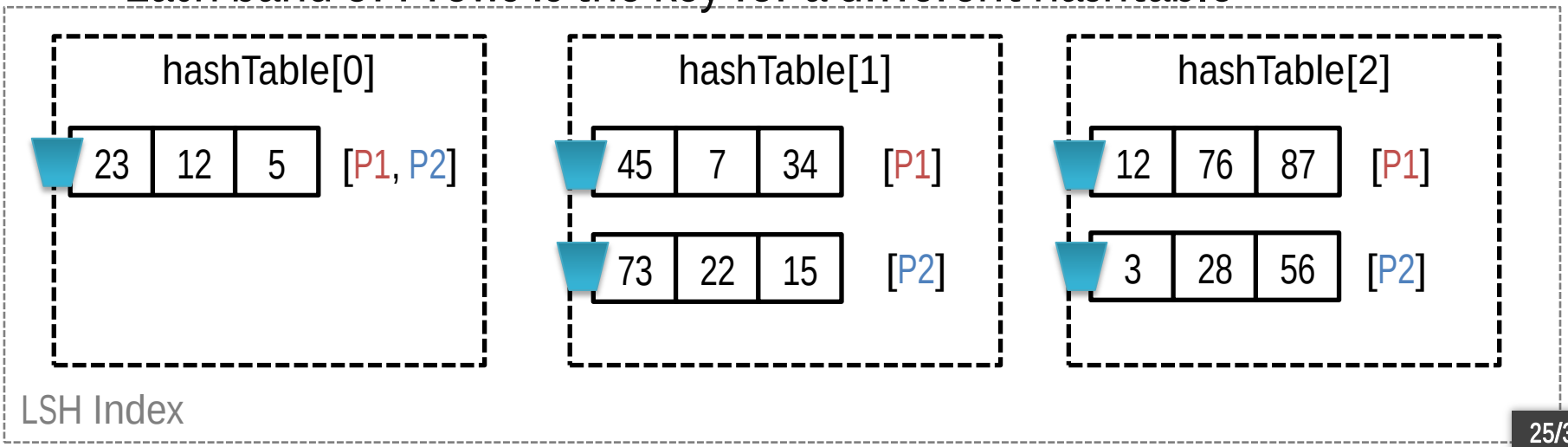

`minHash(P2)`

| 23 | 12 | 5 | 73 | 22 | 15 | 3 | 28 | 56 | … |
|---|---|---|---|---|---|---|---|---|---|

break it into b bands and r rows (based also on the desired similarity threshold)

| 23 | 12 | 5 | | 73 | 22 | 15 | | 3 | 28 | 56 | | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Each band of r rows is the key for a different hashtable



# Interfaces

```
distance(Object o1, Object o2)

insert(Object o)

query(Object o)
```


nearestNeighbor(Object o)


nearNeighbors(Object o, int maxNeighbors)


clustering(Object o)

# Challenges: Implementing

- Generic to any **object**
- Providing multiple hash function families
  (generic to all **distances**)
- Being **efficient** (space and time)
- **Durability**

# Challenges: Scaling up

- **MultiMaps** (1:n)
- **Off-heap** implementation (avoid garbage collection)
- **Bigger** than memory (e.g., using RAM + SSD disk space)
- **Multi-threaded** (fine-grain locks or non-blocking)
- Using **primitives** (avoid space overhead)

# Challenges: Scaling out

- Distributing hash tables in several machines

  hashTable[0] -> s1
  hashTable[1] -> s2
  hashTable[2] -> s3
  hashTable[3] -> s4
  hashTable[4] -> s5

- Partitioning keys (require to inform hashTable number)

  Keys [0 – 1,000,000] -> s1 (hashTable[0-4])
  Keys [1,000,000–2,000,000] -> s2 (hashTable[0-4])
  Keys [2,000,000–3,000,000] -> s3 (hashTable[0-4])
  Keys [3,000,000–4,000,000] -> s4 (hashTable[0-4])

# Available LSH implementations

- OpenLSH (https://github.com/singhj/locality-sensitive-hashing)
- Datasketch (https://github.com/ekzhu/datasketch)
- TarsosLSH (https://github.com/JorenSix/TarsosLSH)
- E2LSH (https://github.com/JorenSix/TarsosLSH)
- Many others

# Some LSH papers

- Similarity Search in High Dimensions via Hashing
- Locality-Preserving Hashing in Multidimensional Spaces
- Approximate Nearest Neighbors: Towards Removing the Curse of dimensionality
- Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions
- Fast Search in Hamming Space with Multi-Index Hashing
- b-Bit Minwise Hashing
- LSH forest: self-tuning indexes for similarity search

# Who uses LSH for what?


Detect near-duplicate web pages

Detecting Near-Duplicates for Web Crawling

Google News recommendations

Google News Personalization: Scalable Online Collaborative Filtering


Detect very similar routes

https://spark-summit.org/2016/events/locality-sensitive-hashing-by-spark/


Detect spam and malicious messages for event organizers

https://www.eventbrite.com/engineering/multi-index-locality-sensitive-hashing-for-fun-and-profit/


Clustering People

http://www.freepatentsonline.com/y2015/0213112.html


Spotify recommender system

LSH forest - ANNOY

And others

# Take outs

- LSH solves **similarity search**
- LSH is very **useful** for several applications
- **Similarity search** is usually **a step** to something bigger
- Think **what do with** the similarity **knowledge**

# Take outs

- **Implementing** basic specific cases is **simple**
- **Being generic** is **not**
- **Scaling** requires **good engineering** and optimizations
- Take time to **experiment the best parameters** to your case

U LISBOA | UNIVERSIDADE DE LISBOA

Ciências ULisboa

LASIGE

# Thank you!

Navtalks – November 18, 2016

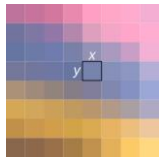
# Other Objects and Dimensions

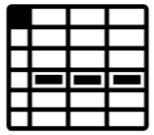
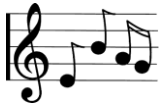
**1 Dimension**
- Binary values: 0 or 1
- Numbers: age, height, weight, etc.

**2 Dimensions**
- Cartesian coordinates: (x, y)
- Tuples: (k, v)

**3 Dimensions**
- 3D coordinates: (x, y, z)
- Animation: (time, x, y)

**N Dimensions**
- Characters in a string: “abcdefgh”
- Substrings of a string: “abc”, “bcd”, “cde”…
- Bits in a Byte array: 0011 1101
- Words in a sentence: “Foo bar bar foo”
- Sentences in a document
- Pixels in an image
- Notes in a music
- Properties in an object
- Columns in a DB row
- Minutiae of fingerprints




# Distances and LSH families

| Distance | Description | LSH family |
|---|---|---|
| Euclidean | Distance between two vectors | Random projections |
| Jaccard | len(intersection)/len(union) | MinHash |
| Cosine | Angular distance between vectors | SimHash |
| Hamming | Number of Substitutions | BitSampling |
| Levenshtein | Minimal number of substitutions, insertions and deletions | |

# Links to used resources

- **Presentations:**

http://www.slideshare.net/j_singh/mining-of-massive-datasets-using-locality-sensitive-hashing-lsh
http://www.slideshare.net/j_singh/open-lsh-a-framework-for-locality-sensitive-hashing-45912645
http://www.slideshare.net/SameeraHorawalavithana/locality-sensitive-hashing (Tree on taxonomy)
http://www.slideshare.net/DmitriySelivanov/finding-similar-items-in-high-dimensional-spaces-locality-sensitive-hashing
http://www.slideshare.net/jsuchal/minhashing-fast-similarity-search
http://www.slideshare.net/SparkSummit/locality-sensitive-hashing-by-spark (Uber on similar routes)
http://www.slideshare.net/InfoQ/approximate-methods-for-scalable-data-mining-25589794
https://speakerdeck.com/polyfractal/going-organic-genomic-sequencing-in-elasticsearch
http://www.slideshare.net/huitseeker/a-gentle-introduction-to-locality-sensitive-hashing-with-apache-spark

- **Blog posts:**

https://www.eventbrite.com/engineering/multi-index-locality-sensitive-hashing-for-fun-and-profit/

- **Videos:**

https://www.youtube.com/watch?v=dgH0NP8Qxa8
https://www.youtube.com/watch?v=bQAYY8INBxg
https://www.youtube.com/watch?v=Arni-zkqMBA
https://www.youtube.com/watch?v=t_8SpFV0I7A
https://www.youtube.com/watch?v=LqcwaW2YE_c
https://www.youtube.com/watch?v=Ha7_Vf2eZvQ
https://www.youtube.com/watch?v=Dkomk2wPaoc