

# Large-Scale Similarity Joins With Guarantees

Rasmus Pagh  
IT University of Copenhagen

EDBT/ICDT  
March 25, 2015



## SCALABLE SIMILARITY SEARCH

# Similarity join example 1 (record linkage)

| Country | Name      |
|---------|-----------|
| USA     | IBM       |
| USA     | Microsoft |
| Germany | SAP       |
| China   | Baidu     |

| Token     | ID |
|-----------|----|
| Mircosoft | 1  |
| SAP SE    | 2  |
| I.B.M.    | 3  |
| baidu.com | 4  |

# Similarity join example 1 (record linkage)

| Country | Name      | Token     | ID |
|---------|-----------|-----------|----|
| USA     | IBM       | Mircosoft | 1  |
| USA     | Microsoft | SAP SE    | 2  |
| Germany | SAP       | I.B.M.    | 3  |
| China   | Baidu     | baidu.com | 4  |

The diagram illustrates the process of record linkage between two datasets. The left dataset contains four rows with columns 'Country' and 'Name'. The right dataset contains four rows with columns 'Token' and 'ID'. Lines connect the names in the left 'Name' column to the tokens in the right 'Token' column, indicating a similarity join. Specifically, 'IBM' is linked to 'Mircosoft', 'Microsoft' is linked to 'SAP SE', 'SAP' is linked to 'I.B.M.', and 'Baidu' is linked to 'baidu.com'.

# Similarity join example 2 (classification)

| Class    | Name   |
|----------|--------|
| Mammalia | Cat    |
| Mammalia | Dog    |
| Reptilia | Snake  |
| Aves     | Parrot |

| Image   | ID |
|---|----|
|   | 1  |
|  | 2  |
|  | 3  |
|  | 4  |

# Similarity join example 2 (classification)

| Class    | Name   |
|----------|--------|
| Mammalia | Cat    |
| Mammalia | Dog    |
| Reptilia | Snake  |
| Aves     | Parrot |

| Features | Image   | ID |
|----------|---|----|
| 0101101  |   | 1  |
| 1001000  |  | 2  |
| 1101101  |  | 3  |
| 1101110  |  | 4  |

# Similarity join example 2 (classification)

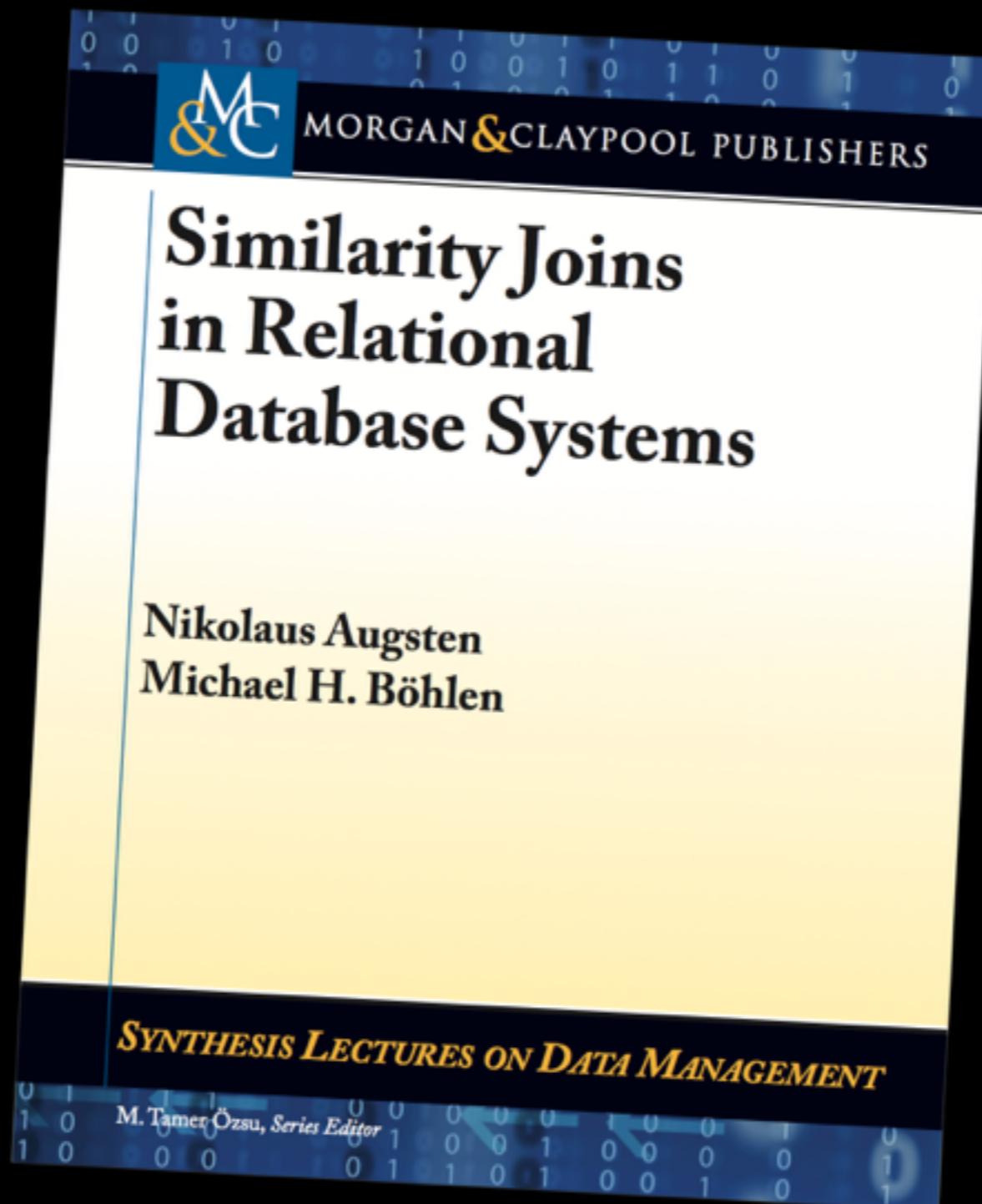
| Class    | Name   | Features |
|----------|--------|----------|
| Mammalia | Cat    | 1100101  |
| Mammalia | Dog    | 1101101  |
| Reptilia | Snake  | 1011000  |
| Aves     | Parrot | 1010111  |

| Features | Image   | ID |
|----------|---|----|
| 0101101  |   | 1  |
| 1001000  |  | 2  |
| 1101101  |  | 3  |
| 1101110  |  | 4  |

# Similarity join example 2 (classification)

| Class    | Name   | Features | Features | Image   | ID |
|----------|--------|----------|----------|---|----|
| Mammalia | Cat    | 1100101  | 0101101  |   | 1  |
| Mammalia | Dog    | 1101101  | 1001000  |  | 2  |
| Reptilia | Snake  | 1011000  | 1101101  |  | 3  |
| Aves     | Parrot | 1010111  | 1101110  |  | 4  |

# More examples



# Similarity Search in High Dimensions via Hashing

ARISTIDES GIONIS<sup>\*</sup> PIOTR INDYK<sup>†</sup> RAJEEV MOTWANI<sup>‡</sup>  
Department of Computer Science  
Stanford University  
Stanford, CA 94305  
[{gionis,indyk,rajeev}@cs.stanford.edu](mailto:{gionis,indyk,rajeev}@cs.stanford.edu)

## Abstract

The nearest- or near-neighbor query problems arise in a large variety of database applications, usually in the context of similarity searching. Of late, there has been increasing interest in building search/index structures for performing similarity search over high-dimensional data, e.g., image databases, document collections, time-series databases, and genome databases. Unfortunately, all known techniques for solving this problem fall prey to the "curse of dimensionality." That is, the data structures scale poorly with data dimensionality; in fact, if the number of dimensions exceeds 10 to 20, searching in k-d trees and related structures involves the inspection of a large fraction of the database, thereby doing no better than brute-force linear search. It has been suggested that since the selection of features and the choice of a distance metric in typical applications is rather heuristic, determining an approximate nearest neighbor should suffice for most practical purposes. In this paper, we examine a novel scheme for approximate similarity search based on hashing. The basic idea is to hash the points

from the database so as to ensure that the probability of collision is much higher for objects that are close to each other than for those that are far apart. We provide experimental evidence that our method gives significant improvement in running time over other methods for searching in high-dimensional spaces based on hierarchical tree decomposition. Experimental results also indicate that our scheme scales well even for a relatively large number of dimensions (more than 50).

## 1 Introduction

A similarity search problem involves a collection of objects (e.g., documents, images) that are characterized by a collection of relevant features and represented as points in a high-dimensional attribute space; given queries in the form of points in this space, we are required to find the nearest (most similar) object to the query. The particularly interesting and well-studied case is the  $d$ -dimensional Euclidean space. The problem is of major importance to a variety of applications; some examples are: data compression [20]; databases and data mining [21]; information retrieval [11, 16, 38]; image and video databases [15, 17, 37, 42]; machine learning [7]; pattern recognition [9, 13]; and, statistics and data analysis [12, 27]. Typically, the features of the objects of interest are represented as points in  $\mathbb{R}^d$  and a distance metric is used to measure similarity of objects. The basic problem then is to perform indexing or similarity searching for query objects. The number of features (i.e., the dimensionality) ranges anywhere from tens to thousands. For example, in multimedia applications such as IBM's QBIC (Query by Image Content), the number of features could be several hundreds [15, 17]. In information retrieval for text documents, vector-space representations involve several thousands of dimensions, and it is considered to be a dramatic improvement that dimension-reduction techniques, such as the Karhunen-Loeve transform [26, 30] (also known as principal components analysis [22] or latent semantic indexing [11]), can reduce the dimensionality to a mere few hundreds!

\*Supported by NAVF N00014-96-1-1221 grant and NSF Grant IIS-9811904.

†Supported by Stanford Graduate Fellowship and NSF NYI Award CCR-9357849.

‡Supported by ARO MURI Grant DAAH04-96-1-0007, NSF Grant IIS-9811904, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.  
Proceedings of the 25th VLDB Conference,  
Edinburgh, Scotland, 1999.

# Similarity Search in High Dimensions via Hashing

ARISTIDES GIONIS<sup>\*</sup> PIOTR INDYK<sup>†</sup> RAJEEV MOTWANI<sup>‡</sup>  
Department of Computer Science  
Stanford University  
Stanford, CA 94305  
[{gionis,indyk,rajeev}@cs.stanford.edu](mailto:{gionis,indyk,rajeev}@cs.stanford.edu)

## Abstract

from the database so as to ensure that the probability of collision is much higher for objects that are close to each other than for those that are far apart. We provide experimental evidence that our algorithm achieves significant improvement in running

## [PDF] Similarity search in high dimensions via hashing

A Gionis, P Indyk, R Motwani - VLDB, 1999 - cs.princeton.edu

**Abstract** The nearest-or near-neighbor query problems arise in a large variety of database applications, usually in the context of similarity searching. Of late, there has been increasing interest in building search index structures for performing similarity search over high- ...

Cited by 1723 Related articles All 46 versions Cite Save More

than brute-force linear search. It is often suggested that since the selection of features and the choice of a distance metric in typical applications is rather heuristic, determining an approximate nearest neighbor should suffice for most practical purposes. In this paper, we examine a novel scheme for approximate similarity search based on hashing. The basic idea is to hash the points

\*Supported by NAVF N00014-96-1-1221 grant and NSF Grant IIS-9811904.

†Supported by Stanford Graduate Fellowship and NSF NYI Award CCR-9357849.

‡Supported by ARO MURI Grant DAAH04-96-1-0007, NSF Grant IIS-9811904, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.  
Proceedings of the 25th VLDB Conference,  
Edinburgh, Scotland, 1999.

item is of interest. Some examples are: data compression [29]; and data mining [21]; information retrieval [11, 16, 38]; image and video databases [15, 17, 37, 42]; machine learning [7]; pattern recognition [9, 13]; and, statistics and data analysis [12, 27]. Typically, the features of the objects of interest are represented as points in  $\mathbb{R}^d$  and a distance metric is used to measure similarity of objects. The basic problem then is to perform indexing or similarity searching for query objects. The number of features (i.e., the dimensionality) ranges anywhere from tens to thousands. For example, in multimedia applications such as IBM's QBIC (Query by Image Content), the number of features could be several hundreds [15, 17]. In information retrieval for text documents, vector-space representations involve several thousands of dimensions, and it is considered to be a dramatic improvement that dimension-reduction techniques, such as the Karhunen-Loeve transform [26, 30] (also known as principal components analysis [22] or latent semantic indexing [11]), can reduce the dimensionality to a mere few hundreds!

# Beyond databases

- Association rule mining
- Automation
- Bio-chemistry (finding motifs)
- Bio-informatics (homology search)
- Clustering
- Computer vision and pattern recognition
- Data cleaning
- Data stream computation
- Data privacy
- First story detection (with application to twitter)
- Identifying trends in time series
- Linear algebra
- Motion planning for robots
- Near-duplicate detection
- News personalization  
(collaborative filtering)
- Privacy preserving data mining
- Search engines for 3D models
- Sensor networks
- ...

# Talk outline

- The algorithmic problem.
- Curse of dimensionality and approximation.
- Techniques for candidate set generation

# Talk outline

- The algorithmic problem.
- Curse of dimensionality and approximation.
- Techniques for candidate set generation
  - A. Locality-sensitive hashing

# Talk outline

- The algorithmic problem.
- Curse of dimensionality and approximation.
- Techniques for candidate set generation
  - A. Locality-sensitive hashing
  - B. Cache-efficiency via recursion

# Talk outline

- The algorithmic problem.
- Curse of dimensionality and approximation.
- Techniques for candidate set generation
  - A. Locality-sensitive hashing
  - B. Cache-efficiency via recursion
  - C. Exact similarity join: Achieving 100% recall

# Algorithmic problem

- Given distance function  $d$  and tolerance  $r$  compute:

$$Q \bowtie_r S = \{(q, x) \in Q \times S \mid d(q, x) \leq r\}$$

# Algorithmic problem

- Given distance function  $d$  and tolerance  $r$  compute:

$$Q \bowtie_r S = \{(q, x) \in Q \times S \mid d(q, x) \leq r\}$$

- This talk:

Consider  $n$  vectors in  $\{0,1\}^D$  and Hamming distance.

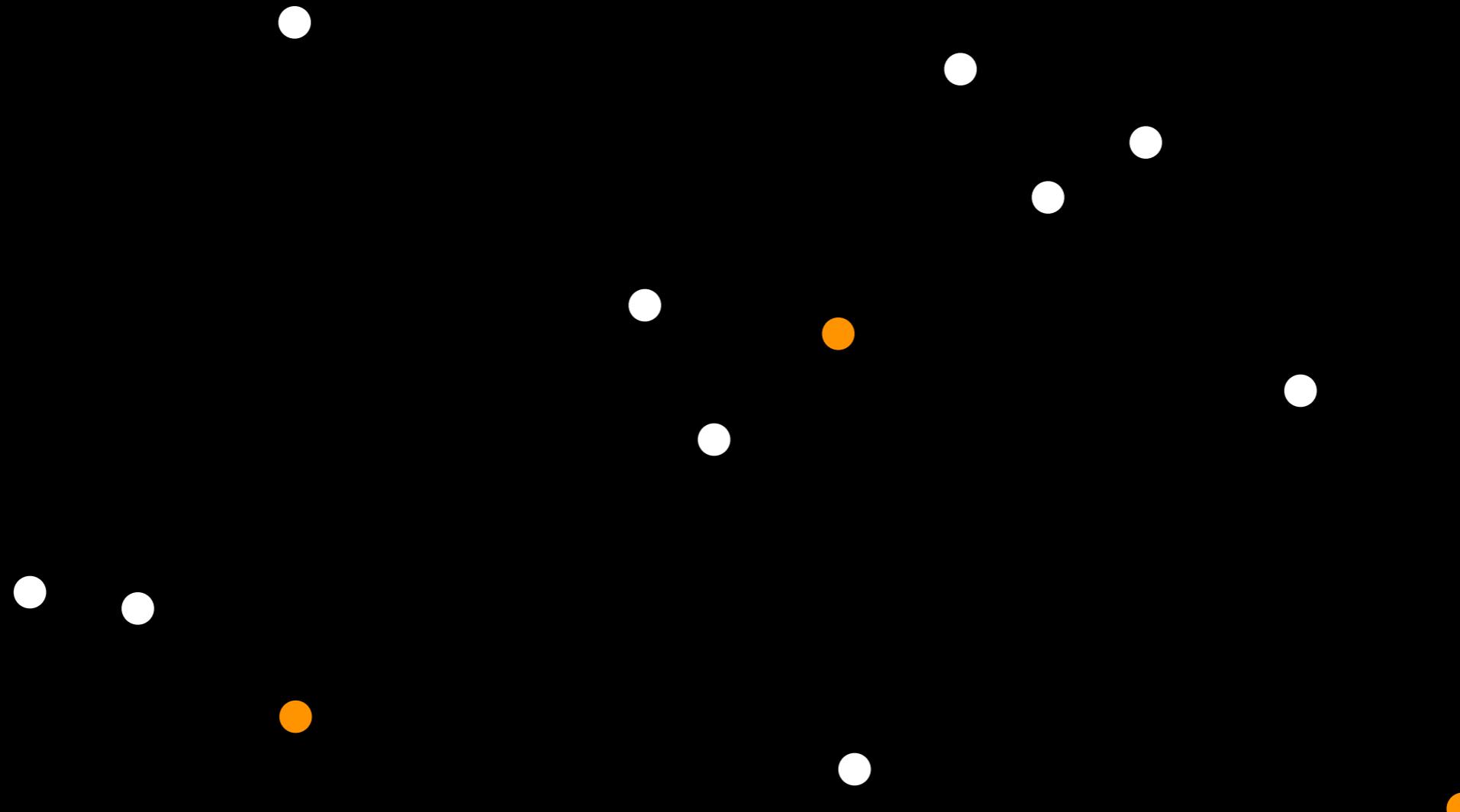
$$q = 110\textcolor{orange}{0}101$$

$$x = 110\textcolor{orange}{1}101$$

$$d(q, x) = 1$$

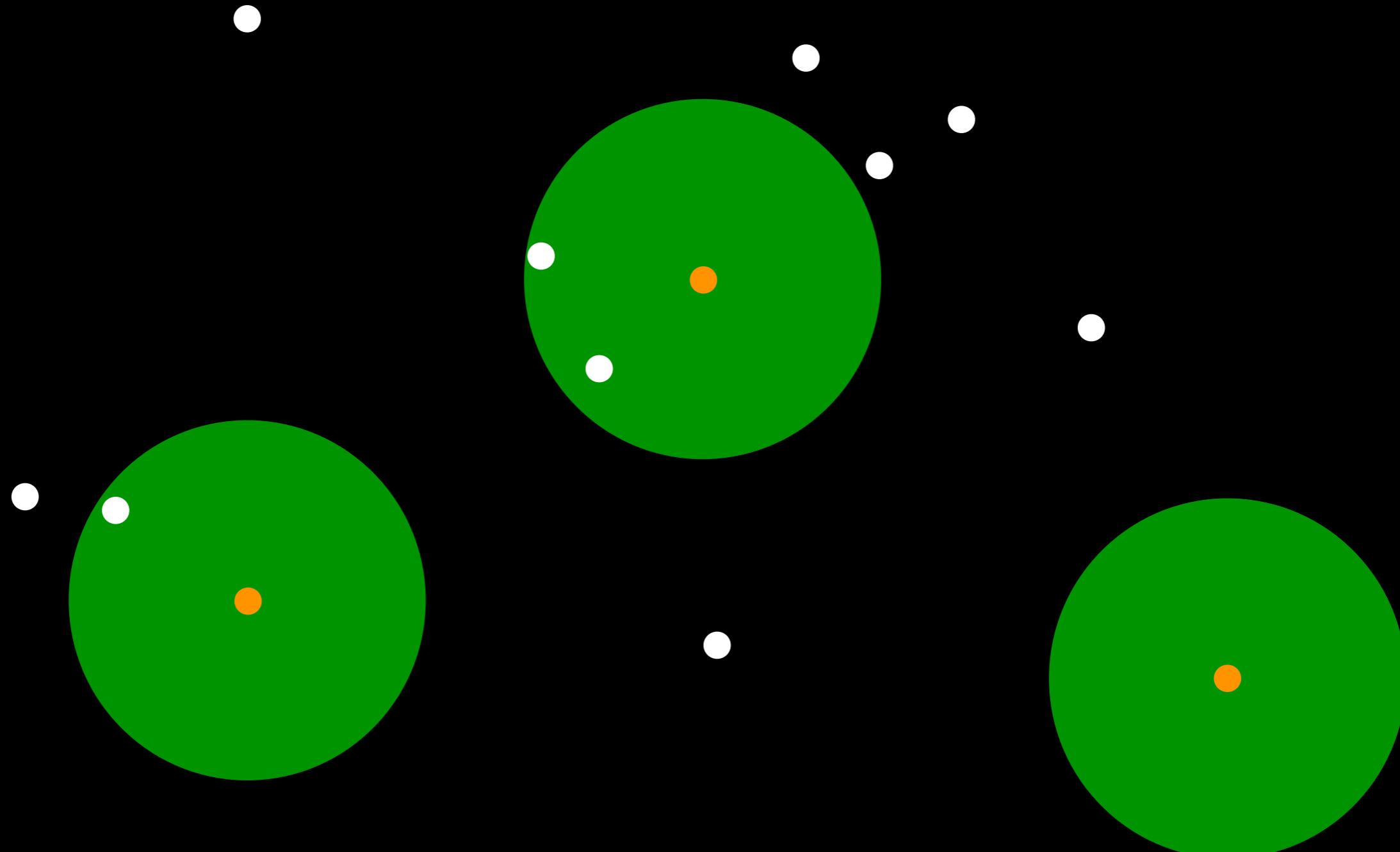
# Similarity join in a picture

$$R = Q \bowtie_r S$$



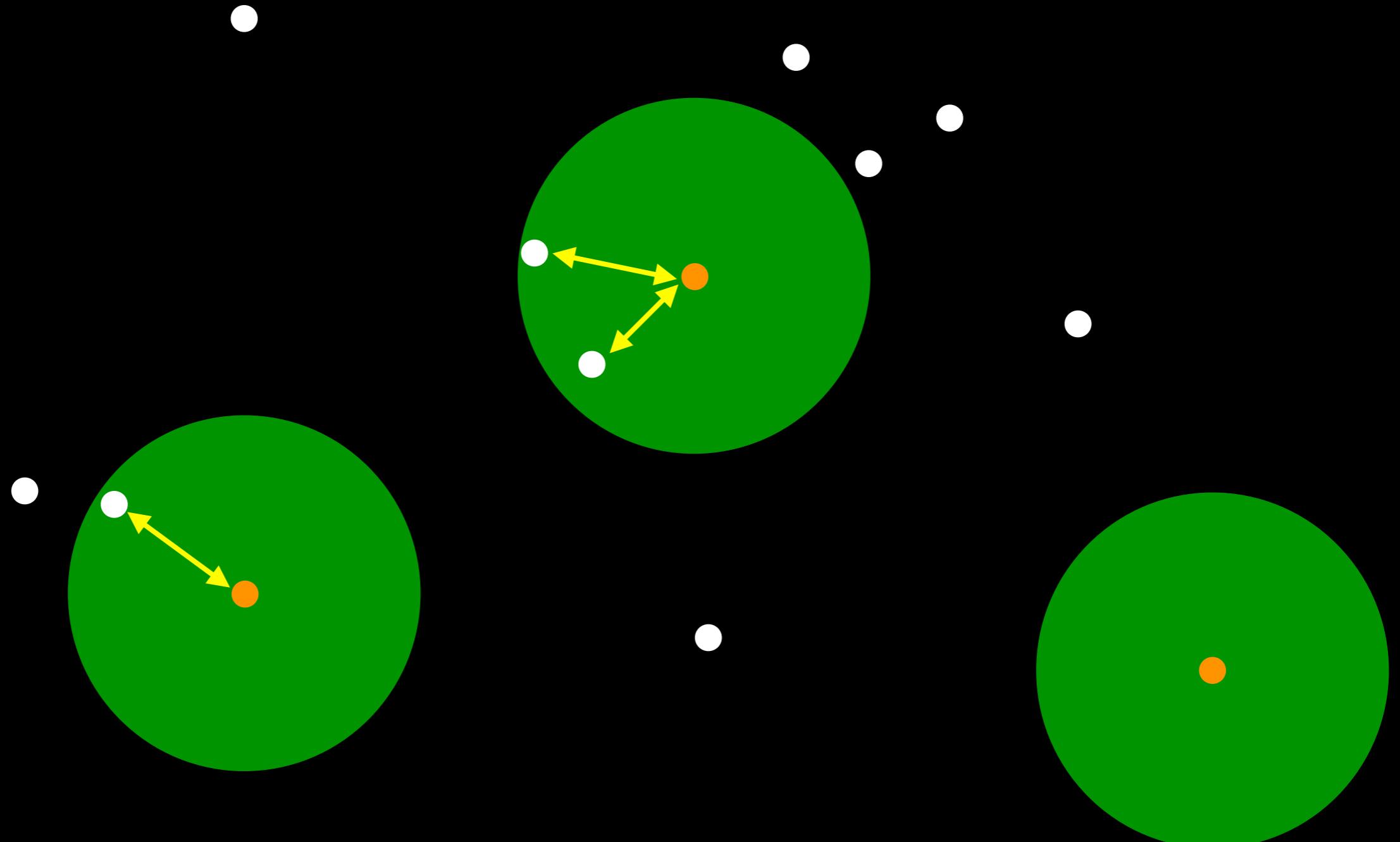
# Similarity join in a picture

$$R = Q \bowtie_r S$$



# Similarity join in a picture

$$R = Q \bowtie_r S$$



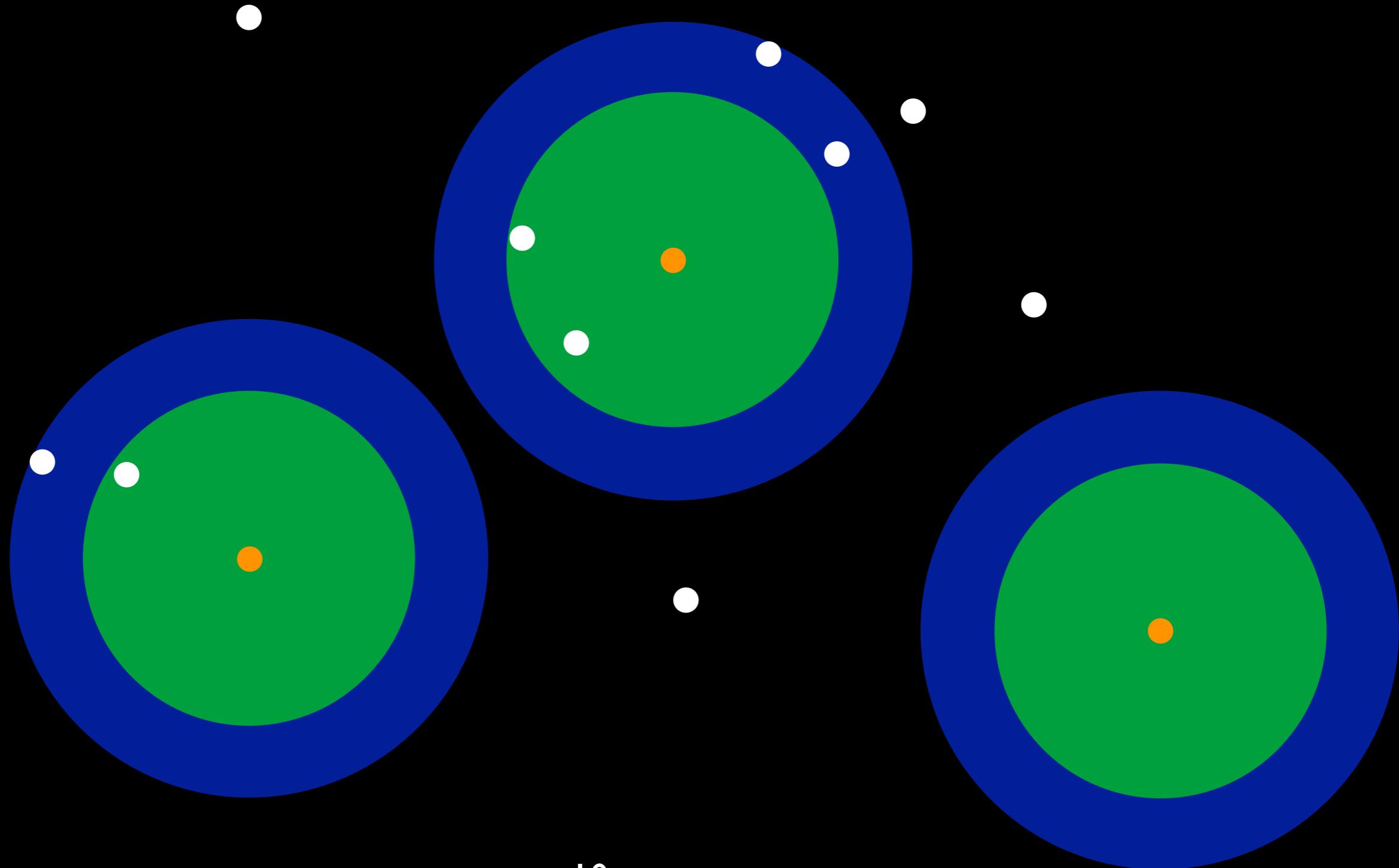
# Similarity join in a picture in high dimensional space ( $D=80$ )

```
10001110001001101100100011110111110000000100001101001111100000011010100
10110011000110001111011000110110110111000111001100110011001100110011000100
01011001001101100001110001000001011011110000001010011100111011011011001011
0111010010010011000111001111101100010110100110001101100011110001111000110101
1111010111010110010001101001011101010010110101011110010011000000111101000110
00011001101100001001010011001100101111111100111100001001100100110010001111
000111011100001000010010100000101100011011111000110110110011010110110110110
001011011101100101100100101010010110010101110100101010101010000110001110000010
11101011000010100100101011101001101111001000010011011111011110001100111100000
01111100101101100011010111001011110011101101100110110011011011011011011110111
001100101100011010111011001111010101011111001110011011100110110011001110011010
0001110001100100001101101101011011111110011001100111101011001101100101111101010
111100011100110001001101101000010001101110000101110000101110100101010010100001001000
1110110110110100010100110110110111100000001001010011110001111101010011001111010010111
111101100100011001000110011000110010001100110001100110011001100110011001111000011110
1011000001011011000110101111100000010001011101101110000010111011011000010110000100000111
00101110100110011001111100000100001011100100011100010001100010001010101011000111000111
0110100100000101001110001100001110000010011000100100010010101110011001100111001110011
010111010000011110100111000111001110001010010110110110011001010101110110011100111001110
0010010001010011000111110000010100111000010110001100011000101000001011101100111001100110
10111110011110101011000001011110010000001010010101000111000011011011000011011011001010
11011010110110000011010110011101011110011111001111101110100001011110000110010011
111001010001001100101110010000011101101110010101101100101101111010110011110101100111010
011011000000111101111010000001101011000011100001100010100011010110011001011100100101
101000110111000110100111101011110011100001110001100011010110000110100010100010101101100
```

01001101110110000110100100100100101110100100110110101110100100110100001101001100001011
1010010001010010010001111110011010111001111011011101011101010100010011011001001101100010
1100001100100011110111010001101011011011100011100011001100101001100101001100110011001000110

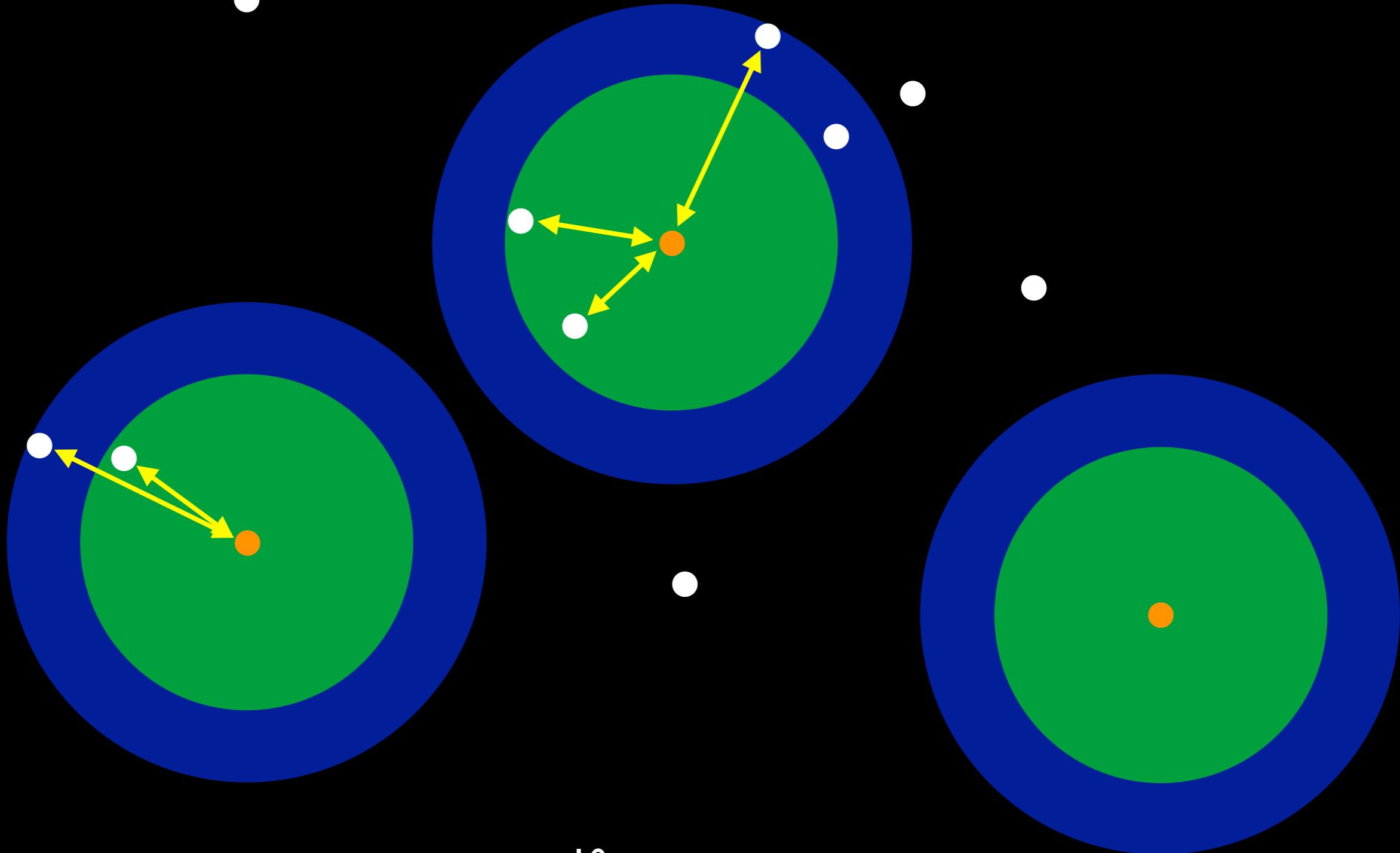
# $c$ -approximate similarity join

$$C \supseteq Q \bowtie_r S$$

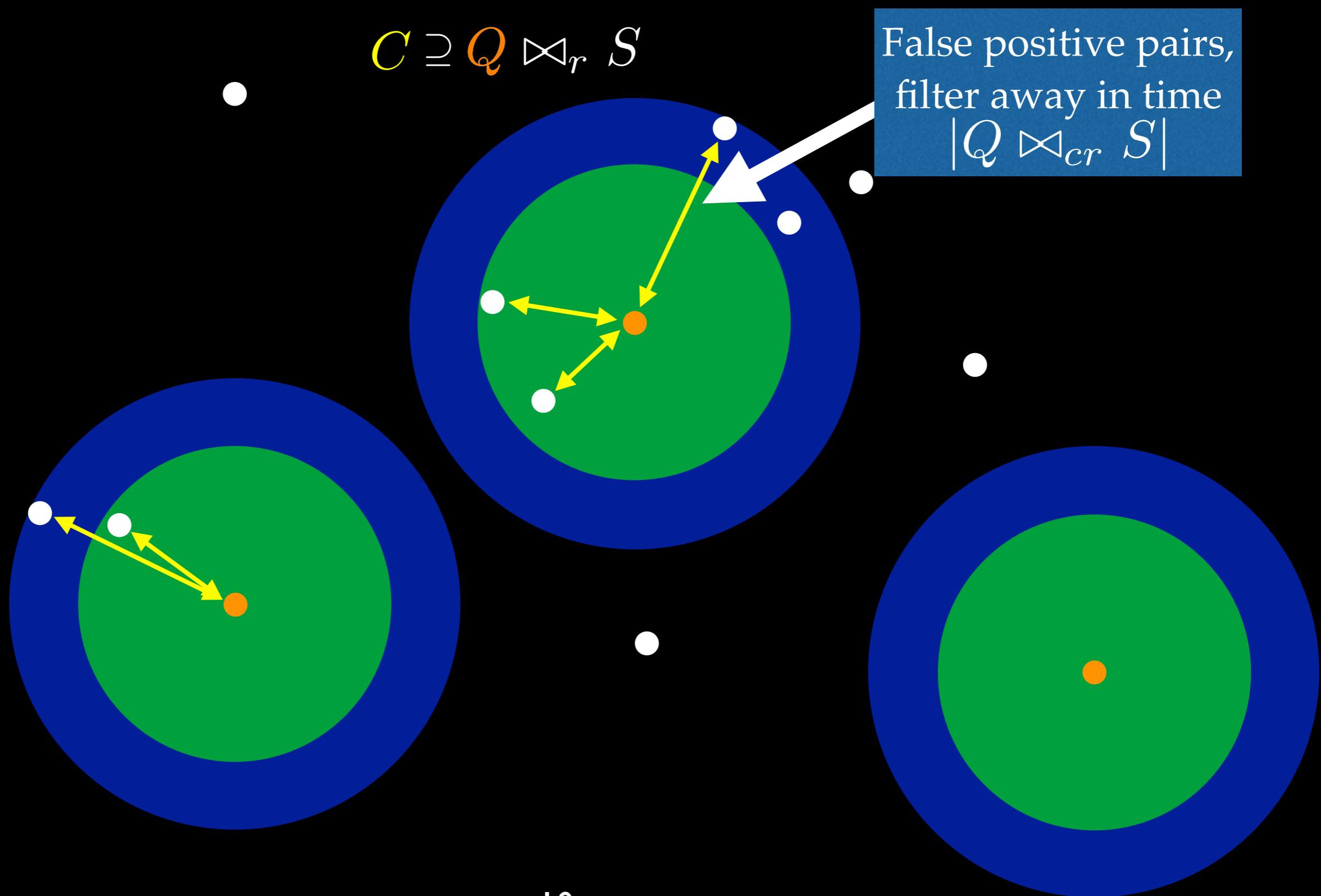


# $c$ -approximate similarity join

$$C \supseteq Q \bowtie_r S$$



# $c$ -approximate similarity join



# Why approximation?

# Why approximation?

Because of the  
CURSE of  
Dimensionality!

of dimensionality

# Why approximation?

- Silvestri et al. 2015:  
 $\exists$  similarity join algorithm using time  $n^{1+o(1)} 2^{o(D)}$   
 $\Rightarrow$   
 $k$ -SAT w.  $n$  variables can be solved in time  $2^{n/2+o(1)}$

# Why approximation?

- Silvestri et al. 2015:

$\exists$  similarity join algorithm using time  $n^{1+o(1)} 2^{o(D)}$

$\Rightarrow$

$k$ -SAT w.  $n$  variables can be solved in time  $2^{n/2+o(1)}$

Huge research effort has been devoted to  $k$ -SAT,  
but so far failed to even reach  $1.99^n$  for general  $k$ .

# Why approximation?

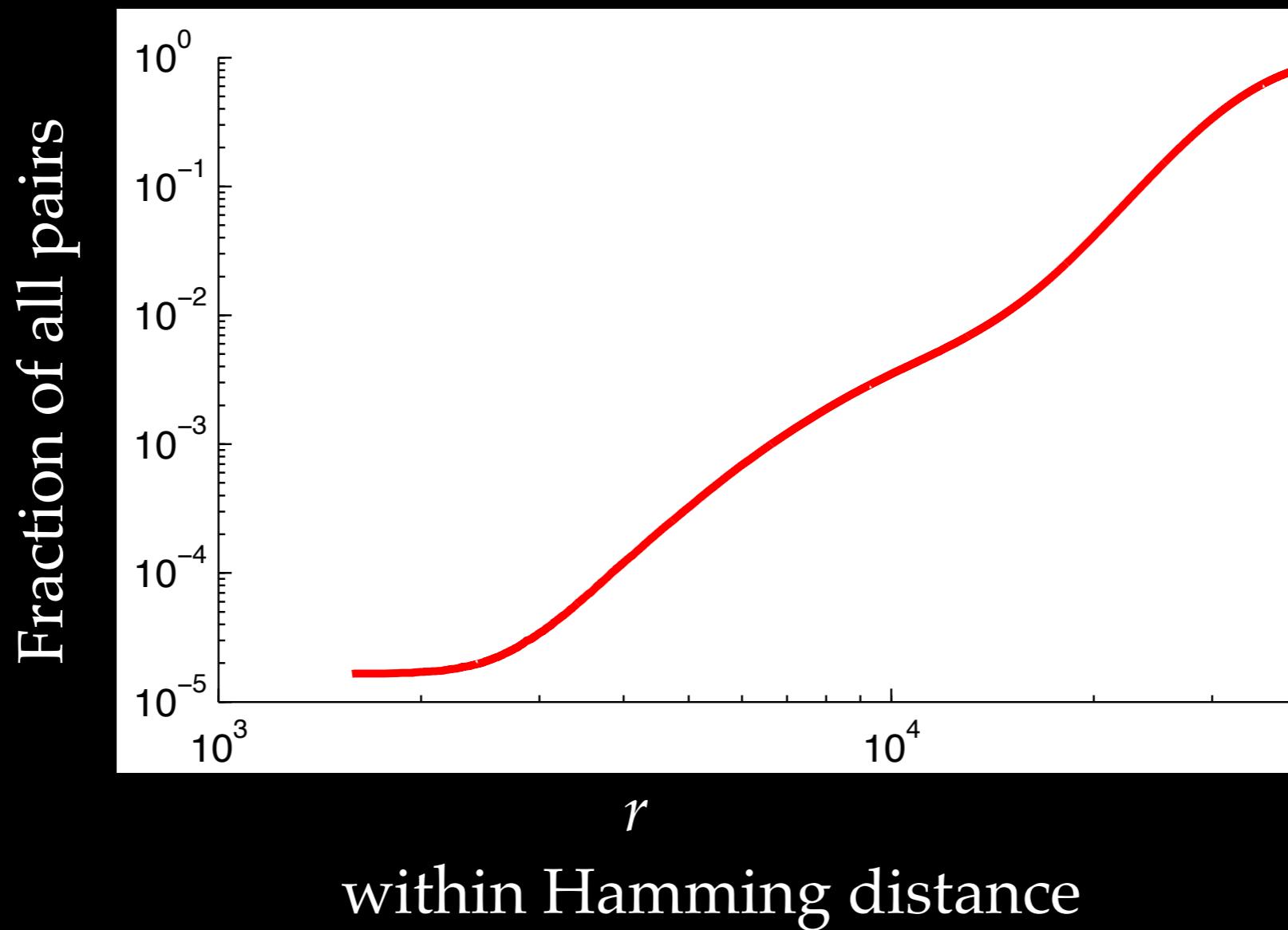
Conclusion: A highly efficient algorithm for similarity join with no approximation of distances would be very surprising.

- Silvestri et al. 2015:  
 $\exists$  similarity join algorithm using time  $n^{1+o(1)} 2^{o(D)}$   
 $\Rightarrow$   
 $k$ -SAT w.  $n$  variables can be solved in time  $2^{n/2+o(1)}$

Huge research effort has been devoted to  $k$ -SAT,  
but so far failed to even reach  $1.99^n$  for general  $k$ .

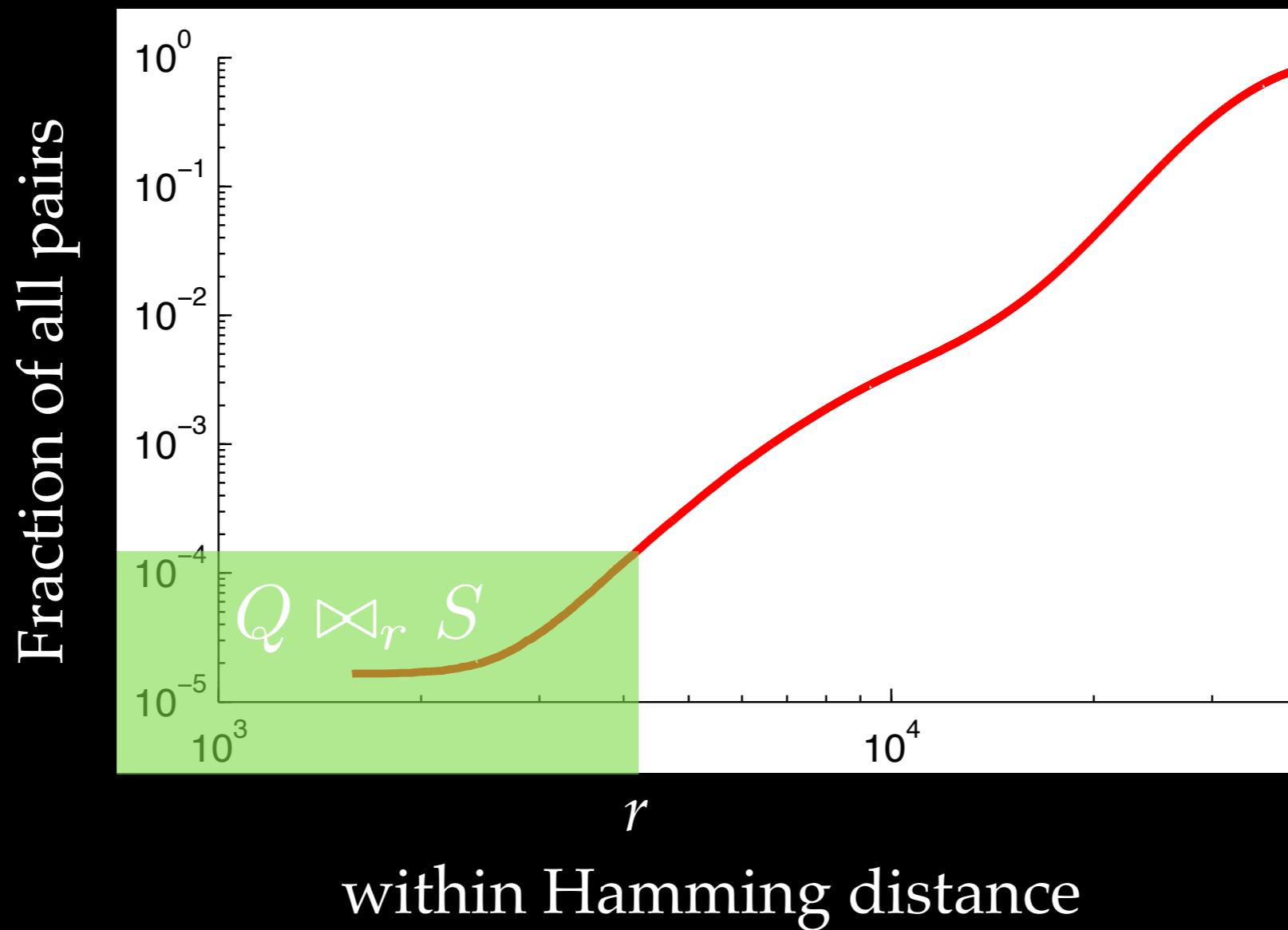
# Effect of approximation

MNIST data set (unary encoding)



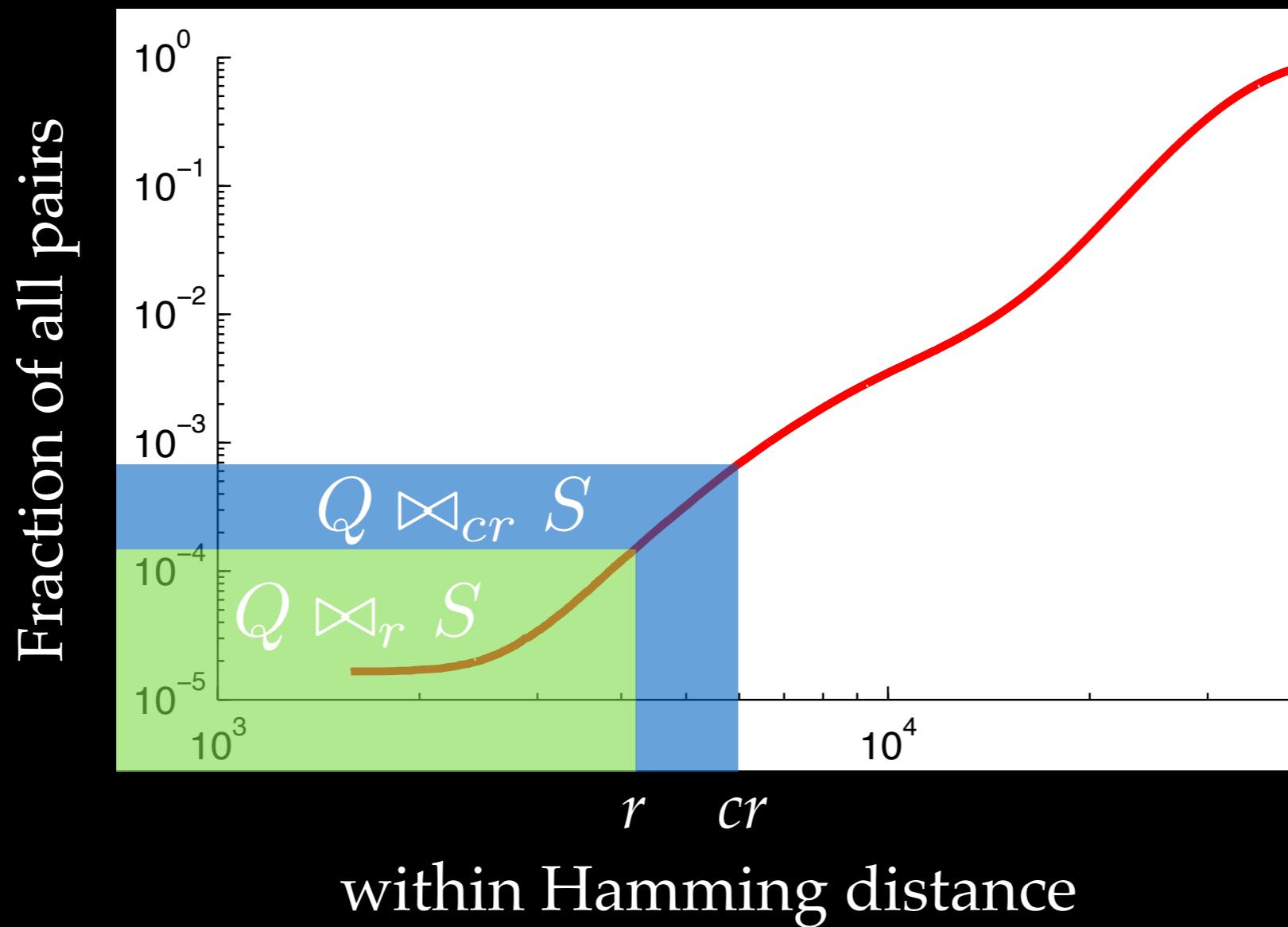
# Effect of approximation

MNIST data set (unary encoding)



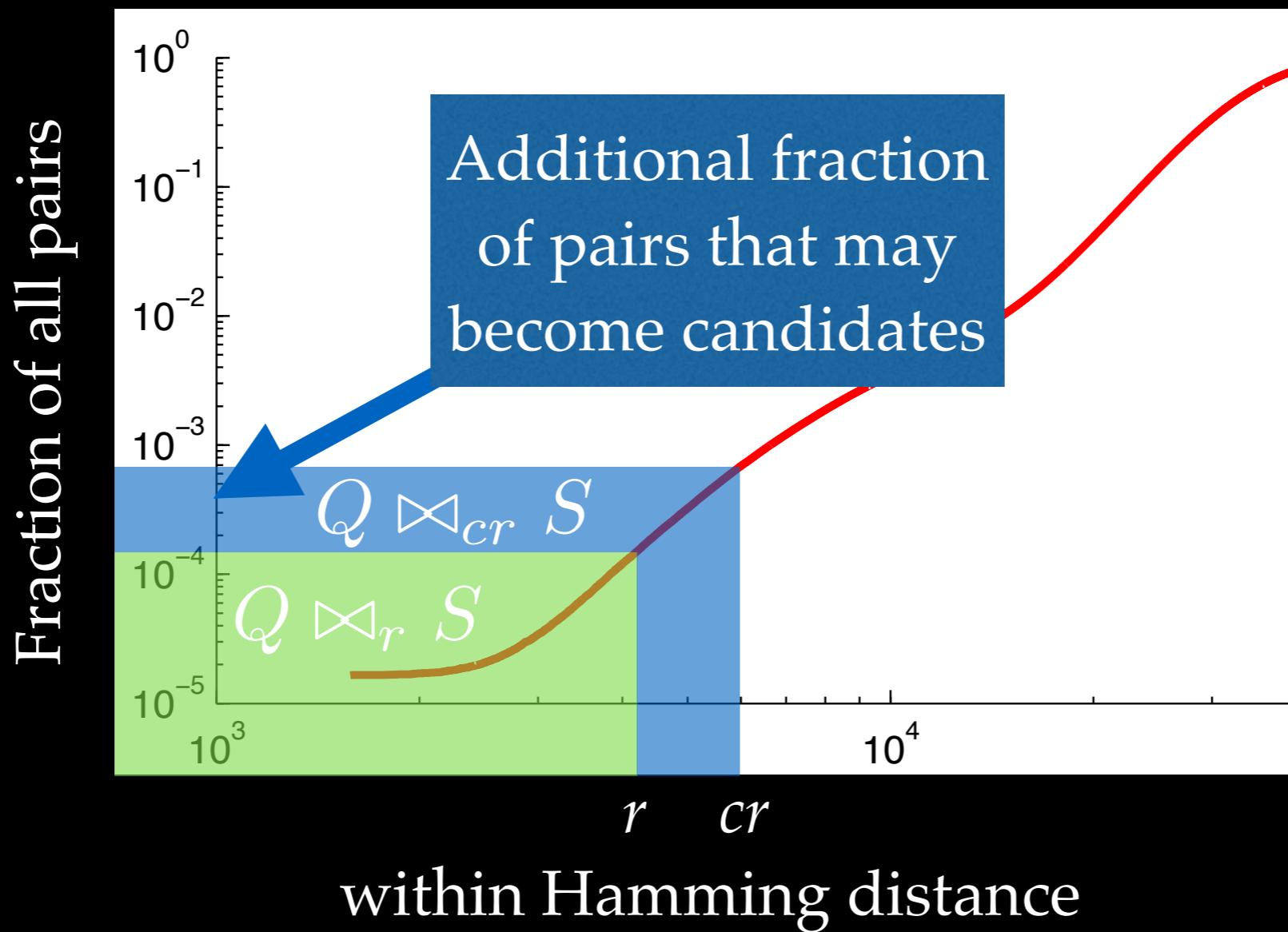
# Effect of approximation

MNIST data set (unary encoding)



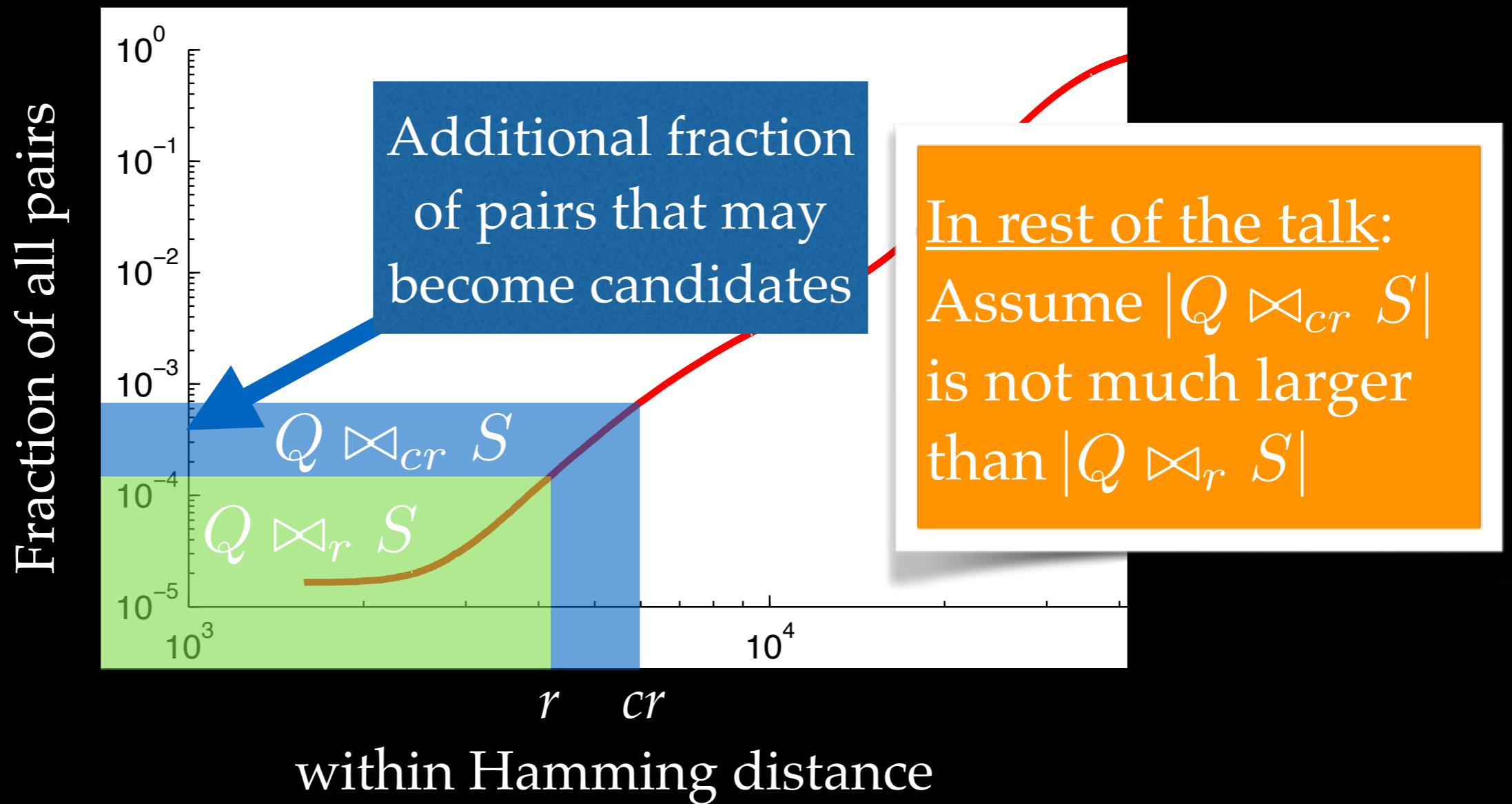
# Effect of approximation

MNIST data set (unary encoding)



# Effect of approximation

MNIST data set (unary encoding)



Part A.

# Locality-sensitive hashing (LSH)

# Locality-sensitive hashing

```
100011100010011011001000111101111000000010000110100111100000110101000  
1011001100011000111011010001101101101110001110011001100110011001001100100  
0101100100110110000111000100000101101110000010100111001110110110011001011  
01110100100100110001110011110101100010110100110001101100011110001111000110101  
111101011101011000010000110101101010010110101011110010011000000111101000110  
000110011011000010010100110001010111111100011110000100110010001111  
0001110111000010000100101000001011000110111110001101101100011010110110110  
0010110111011001011001001001011001010111010010101010101010000110001110000010  
11101011000010100100101011101001101111001000010011011111011110001100111100000  
011111001011011000011010111000101111000111011011001101100101111010111101111  
0011001011000110101110110011101010010111100110011100110011001100011100110100  
000111000110010000111011011010110111110011001100111100110011110101111010101
```

Idea: Consider projection onto  
a random subset of dimensions,  
each chosen with probability  $p$

```
011011100010111010011001101101110010000111011011100101011100100101010001001000  
11000000001001010011110001110000101100010001111001111101001010111  
010001010010000001001111011011000010110001000100011110001111101001010111  
00001000101111011101100001011000100010001000100010001000100010001000100010001  
101110010001110001000100010001000100010001000100010001000100010001000100010001  
100000100110100010010001001000100100010001000100010001000100010001000100010001  
011100101001010110110101110011001100110011001100110011001100110011001100110011  
1011100111110110110110110110110110110110110110110110110110110110110110110110110  
10000000101001010010100101001010010100101001010010100101001010010100101001010010  
1110011111101110100010111100010111100010111100010111100010111100010111100010111100  
110001100100011110111010001101111000111100011110001111000110011001100110011001100110  
01001101110110001101001001001001001001001001001001001001001001001001001001001001001011  
1010010001010010010001111110011011100111101101111011011110110111101101111011011110110111101  
110000110010001111011101000110111011011110001111000111100011001100110011001100110011001100110
```

# Locality-sensitive hashing

```
100011100010011011001000111101111100000010000110100111100000101000  
10110011000110001110110100011010110111000111001100011001100101001100100  
0101100100110110000111000100000101101110000010100111001101110011001011  
0111010010010011000111001111010110001011001100011011000111100011001101  
11110101110101100100011010011101000101101001011011100110000001111000110  
00011001101100001001001011000110011001011111100011110000100110010001111  
0001110111000010000100101000001011000110111110001101100011010110110110  
001011011101100101100100100101100101011101001010101010101000011000111000010  
1110101100001010010010101110100111100100000100110111110111100011001111000000  
01111100101101000110101110001011110001110110110011011001101101101101111010111  
00110010110001101011101100111101010010111100110011110001101100110100011100100  
00011100011001000111011011010111100111110011011001101100110101111010101
```

Idea: Consider projection onto a random subset of dimensions, each chosen with probability  $p$

```
011011100010111011011000011101101110010101110010111010010100001001000  
100000000111101110100000110101100001110010100011011001001100100011000111  
0001000101110110111011000010111101100001011110110000101111011000010000111  
10111001000111000100011100010001110001000111000100011100010001110001000111  
00000100110001000111000100011100010001110001000111000100011100010001110001  
011100101010010101101101101101101101101101101101101101101101101101101101  
101100111101101101101101101101101101101101101101101101101101101101101101  
0000000101001010100100011100001101101101101101101101101101101101101101  
111001111101101101101101101101101101101101101101101101101101101101101101  
111001010010011101101101101101101101101101101101101101101101101101101101  
011011000000111101110100000110101100001110010100011011001001101100100100101  
101000110111000110100111101101101101101100011110001100110101100001101000101101100  
  
01001101110110001101001001001001001001001001001001001000001101001001100001011000  
10100100010100100100011111100110101100111101101101101101101010100010011011000100010  
11000011001010001111011010001101011011011100011100011001101011000011001011001101000110
```

# Locality-sensitive hashing

$$h(x) = x \wedge a$$

```
int lsh(int x) {return x & a}
```

```
10001110000100  
10110011000110  
01011001001101  
01110100100100  
1111010111010  
00011001101100  
00011101110000  
00101101111011  
11101011000010  
01111100101100  
00110010110001  
00011100110001
```

```
00000110101000  
11001101000100  
11010011001011  
01111000110101  
00111010001110  
00000011000000  
00000000100100  
01000101000000  
00010001011101  
10111001000111  
00000100100010  
01110010100110  
01110011110110  
00000010100100  
01100111110110  
00000001011100  
11100111110100  
01101000000111  
10100011001110  
01000110010011
```

Idea: Consider projection onto a random subset of dimensions, each chosen with probability  $p$

```
111001010010011001011100100000110110111001010000011011011100101011011001011101101110101  
01101100000001111011110100000011010110000111001010001101011001001100101011100100100101  
10100011011100011010011110101110110001111000011010110000110100011010001101000101101100  
0100110111011000110100100100100100100101011101011001111000110101101011010101000110100110001  
10100100010100100100100100100100101100111100011010110101101010101000100100110110001  
1100001100100011110110001101011011100011110001100110001100101011001100010101100110001100
```

# Locality-sensitive hashing

$$h(x) = x \wedge a$$

```
int lsh(int x) { return x & a }
```

```
1000111000100  
10110011000110  
01011001001101  
01110100100100  
1111010111010
```

```
00000110101000  
11001101000100  
11010011001011  
01111000110101  
00111010001111  
00011101100001
```

```
001011011101101010010010010110010101100101011000100001100000000010  
111010110001010010010110010110010110000100110111101110000000000000  
0111110010110110000110101110001011000001011001100110110110111101011  
00110010110001101011101100111010110000011010001101100110100011100100  
000111000110010000111011011011010110000000000000000000000000000000000
```

Idea: Consider projection onto a random subset of dimensions, each chosen with probability  $p$

Candidate pair

```
111001010010011001011100100000110100000000000000000000000000000000000  
01101100000001111011101000000110101010101010101010101010101010101010  
1010001101110001101001111010111011011001100000000000000000000000000000
```

```
01001101110110110001101001001001010111010010011010101000001101001100001  
101001000101001001001000111111010110110011110110110110110101010001001101100  
110000110010001110111010001101011011100011100110001100101001011001101000110
```

# Locality-sensitive hashing

```
1 0 0 0 1 1 1 0 0 0 0 1 0 0  
1 0 1 1 0 0 1 1 0 0 0 1 1 0  
0 1 0 1 1 0 0 1 0 0 1 1 0 1  
0 1 1 1 0 1 0 0 1 0 0 1 0 0  
1 1 1 1 0 1 0 1 1 1 1 0 1 0
```

$$h(x) = x \wedge a$$

```
int lsh(int x) {return x & a}
```

Idea: Consider projection onto a random subset of dimensions, each chosen with probability  $p$

Repeat enough times that pairs at distance  $r$  produce at least one collision

```
1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 1  
0 1 1 0 1 1 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 0 1 1  
1 0 1 0 0 1 1 0 1 1 1 0 0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 0 1 1 1 0 1
```

# Locality-sensitive hashing

$$h_i(x) = x \wedge a_i$$

```
int lsh(int x, int i) {return x & a[i];}
```

10001110  
10110011  
01011001  
01110100  
11110101

```
00011001101100001001010110001100101111111111000110000100110010001111  
0001110111000010001001010000010110001101111000011011011011000110110110  
00101101110110110010010100100100100101100010101101001010101000011001110000010  
1110101100010101010110010110001011000101100010110001011000101100010110000000  
01111110010110101000110101110001010111001101100101100110111010111010111010111  
0011001011000110101110101101100110101110111001101100110110110011010001101100110100  
0001110001100100001101101110101101101101101101101101101101101101101101101101101101101
```

Idea: Consider projection onto a random subset of dimensions, each chosen with probability  $p$

## Candidate pair

Repeat enough times that pairs at distance  $r$  produce at least one collision

```
0100110110110000110100100100100100101110100100110100110100000110100110000101  
1010010001010010100100011111100110101100111011011011011011010110101010  
110000110010100011110111010001101011011011110001111001100011001010011001101000110
```

# Chance of collision?

$$x \wedge a_1 \stackrel{?}{=} q \wedge a_1$$

$$x \wedge a_2 \stackrel{?}{=} q \wedge a_2$$

$$x \wedge a_3 \stackrel{?}{=} q \wedge a_3$$

⋮ ⋮

$$x \wedge a_t \stackrel{?}{=} q \wedge a_t$$

# Chance of collision?

$$x \wedge a_1 \stackrel{?}{=} q \wedge a_1 \quad \text{prob. } (1-p)^{d(q,x)}$$

$$x \wedge a_2 \stackrel{?}{=} q \wedge a_2 \quad \text{prob. } (1-p)^{d(q,x)}$$

$$x \wedge a_3 \stackrel{?}{=} q \wedge a_3 \quad \text{prob. } (1-p)^{d(q,x)}$$

$$\vdots \qquad \vdots$$

$$x \wedge a_t \stackrel{?}{=} q \wedge a_t \quad \text{prob. } (1-p)^{d(q,x)}$$

# Chance of collision?

$$x \wedge a_1 \stackrel{?}{=} q \wedge a_1 \quad \text{prob. } (1-p)^{d(q,x)}$$

$$x \wedge a_2 \stackrel{?}{=} q \wedge a_2$$

Probability of  
no collision:

$$x \wedge a_3 \stackrel{?}{=} q \wedge a_3$$

$$(1-(1-p)^{d(q,x)})^t$$

:

:

$$x \wedge a_t \stackrel{?}{=} q \wedge a_t$$

$$\text{prob. } (1-p)^{d(q,x)}$$

# Chance of collision?

$$x \wedge a_1 \stackrel{?}{=} q \wedge a_1 \quad \text{prob. } (1-p)^{d(q,x)}$$

$$x \wedge a_2 \stackrel{?}{=} q \wedge a_2$$

$$x \wedge a_3 \stackrel{?}{=} q \wedge a_3$$

 $\vdots$  $\vdots$ 

Probability of  
no collision:

$$(1-(1-p)^{d(q,x)})^t$$

From math book:  $(1-x)^{1/x} \rightarrow 1/e$  as  $x \rightarrow 0$

# Summary of analysis

Analysis (GIM '99): Each bit of  $a_i$  is 1 with probability  $p$ .

- Expect  $c_p \leq n^2(1-p)^{cr}$  candidate pairs outside  $Q \bowtie_{cr} S$
- Probability of  $h(q)=h(x)$  when  $d(q,x) = r$  is  $(1-p)^r$ , so need to hash  $t_p=(1-p)^{-r}$  times to achieve constant recall.
- Choose  $p$  to balance cost of hashing  $t_p$  times and checking  $t_p c_p$  candidate pairs not in  $Q \bowtie_{cr} S$

# Summary of analysis

Analysis (GIM '99): Each bit of  $a_i$  is 1 with probability  $p$ .

- Expect  $c_p \leq n^2(1-p)^{cr}$  candidate pairs outside  $Q \bowtie_{cr} S$
- Probability of  $h(q)=h(x)$  when  $d(q,x) = r$  is  $(1-p)^r$ , so need to hash  $t_p=(1-p)^{-r}$  times to achieve constant recall.
- Choose  $p$  to balance cost of hashing  $t_p$  times and checking  $t_p c_p$  candidate pairs not in  $Q \bowtie_{cr} S$

# Summary of analysis

Analysis (GIM '99): Each bit of  $a_i$  is 1 with probability  $p$ .

- Expect  $c_p \leq n^2(1-p)^{cr}$  candidate pairs outside  $Q \bowtie_{cr} S$
- Probability of  $h(q)=h(x)$  when  $d(q,x) = r$  is  $(1-p)^r$ , so need to hash  $t_p=(1-p)^{-r}$  times to achieve constant recall.
- Choose  $p$  to balance cost of hashing  $t_p$  times and checking  $t_p c_p$  candidate pairs not in  $Q \bowtie_{cr} S$

# Locality-sensitive hashing

Analysis (GIM '99): Each bit of  $a_i$  is 1 with probability  $p$ .

- Expect  $c_p \leq n^2(1-p)^{cr}$  candidate pairs outside  $Q \bowtie_{cr} S$
- Probability of  $h(q)=h(x)$  when  $d(q,x) = r$  is  $(1-p)^r$ , so need to hash  $t_p=(1-p)^{-r}$  times to achieve constant recall.
- Choose  $p$  to balance cost of hashing  $t_p$  times and checking  $t_p c_p$  candidate pairs not in  $Q \bowtie_{cr} S$

# Locality-sensitive hashing

Analysis (GIM '99): Each bit of  $a_i$  is 1 with probability  $p$ .

- Expect  $c_p \leq n^2(1-p)^{cr}$  candidate pairs outside  $Q \bowtie_{cr} S$
- Probability of  $h(q)=h(x)$  when  $d(q,x) = r$  is  $(1-p)^r$ , so need to hash  $t_p = (1-p)^{-r}$  times to achieve constant recall.
- Choose  $p$  to balance cost of hashing  $t_p$  times and checking  $t_p c_p$  candidate pairs not in  $Q \bowtie_{cr} S$

Number of operations is  $O(n^{1+1/c}D)$ , expected;  
Can be implemented in  $O(n^{1+1/c}D/B)$  I/Os.

# Can LSH work for large data?

- Quote from an ICDT paper:  
*“LSH needs large memory space and long processing time to achieve good performance when searching a massive dataset”.*

# Can LSH work for large data?

- Quote from an ICDT paper:  
*“LSH needs large memory space and long processing time to achieve good performance when searching a massive dataset”.*

For similarity join,  
space is linear:  
Process one hash  
function at a time.

# Can LSH work for large data?

- Quote from an ICDT paper:  
*“LSH needs large memory space and long processing time to achieve good performance when searching a massive dataset”.*
- Other issues:
  - LSH parameters are pessimistic; chosen to work for worst-case data set.

For similarity join,  
space is linear:  
Process one hash  
function at a time.

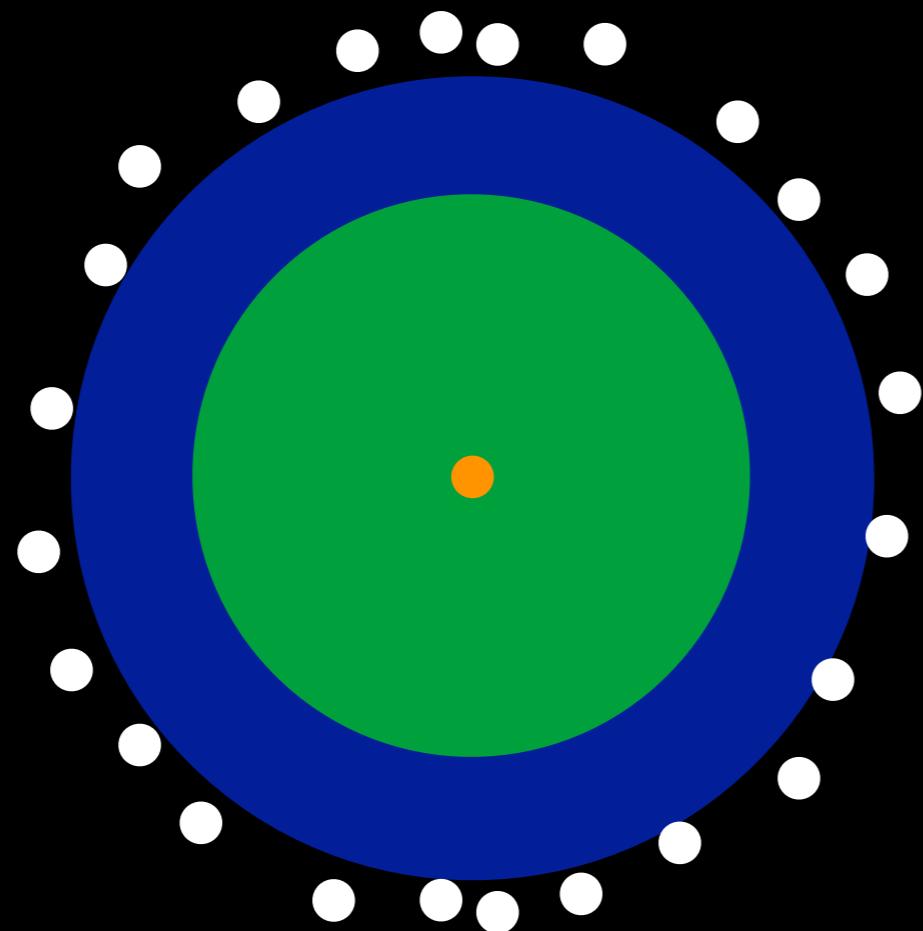
# Can LSH work for large data?

- Quote from an ICDT paper:  
*“LSH needs large memory space and long processing time to achieve good performance when searching a massive dataset”.*
- Other issues:
  - LSH parameters are pessimistic; chosen to work for worst-case data set.
  - Poor use of internal memory: A simple nested loop join that uses  $O(D^2n^2 / (MB))$  I/Os is often better.

For similarity join,  
space is linear:  
Process one hash  
function at a time.

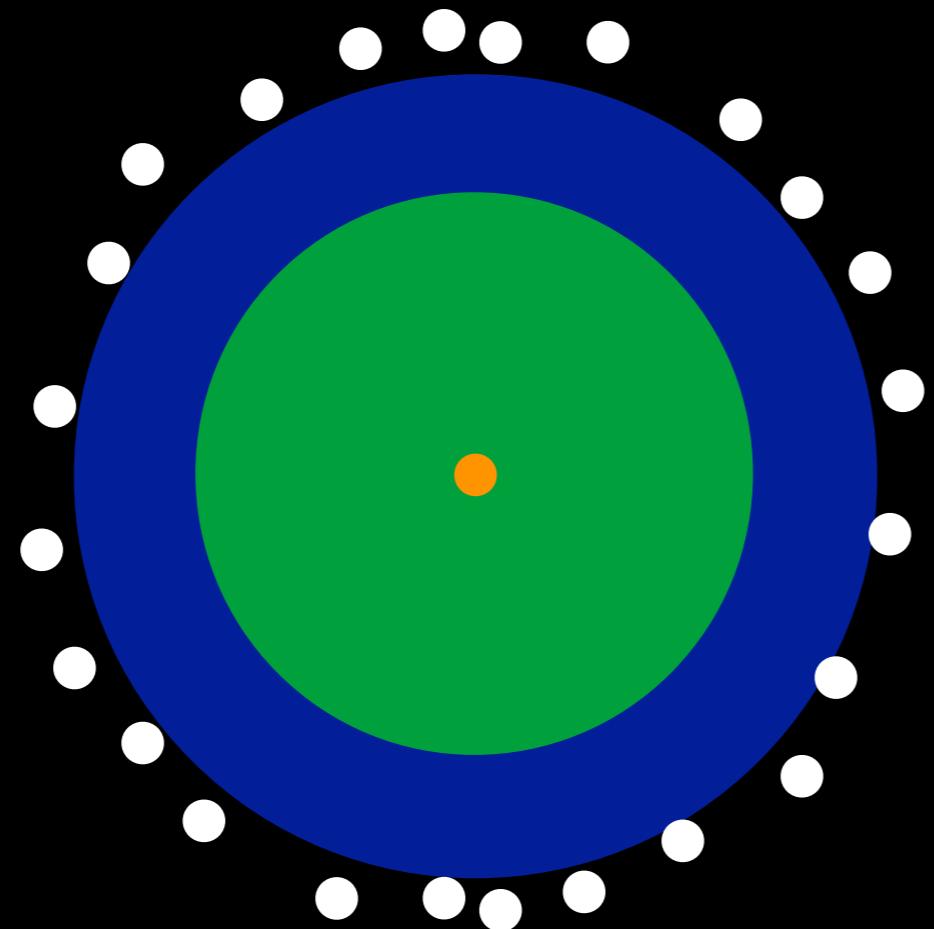
# On pessimism

- LSH chosen to ensure few collisions at distance  $cr$ , even in worst-case scenarios:



# On pessimism

- LSH chosen to ensure few collisions at distance  $cr$ , even in worst-case scenarios:



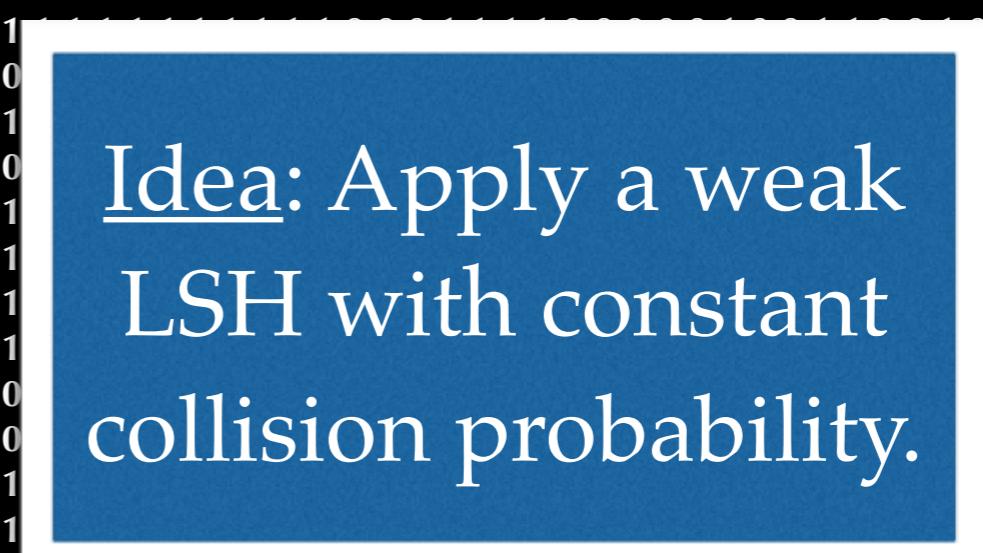
Price paid:  
Also small collision probability at distance  $r$ , so need many repetitions.

# Part B.

# Cache-efficiency via recursion

# Cautious LSH

Joint work with Pham, Silvestri, and Stöckel



Idea: Apply a weak LSH with constant collision probability.

10001110001001101100100011110111110000000100001101001111100000011010100  
101100110001100011110110001101101101110001110001100110001100100110011000100  
010110010011011000011100010000010110111100000010100111001110110100110011001011  
01110100100100110001110011110101100010110100110001101100011110001111000110101  
1111010111010110100100001101011101010010110100101101011110010011000000111101000110  
00011001101110000100101001011000110010111100001101111001001100011110001100110101  
00011101110000100001001010000001011000110  
00101101110110010110100100100101100101  
1110101100001010010010111010011011110010  
01111100101101100001101011100010111100011  
001100101100011010111011100111010100101  
000111000110010000110110111010110111111  
111100011100110001000100111010000010001101  
11101101101101000101001001101101011111000  
111101100100011001000011000110100010001000  
1011000001011011000111010110111110000001  
00101110100110100111111000001000010111  
01101001000001010011110011000011110000010000  
0101110100001111010011000111010011110001010  
00100100010100110001111110000010111000001011100  
10111110011110101011000001011110010000001010010000010111000110111000110111001010  
1101101011011000001101011001110101111100111110110100001011110000110111000110010011  
11100101000100110010111100000111011011001011101011111010110110111110101111101011101  
011011000000111101110100000110101100001110010100011010110011001011001100101011100100101  
1010001101110001101001111010111100001111000110101100011010001101000110100010101101100  
01001101110110000110100100100100100100101110000110100100100100100100101110000110100110001011  
1010010001010010010001111110011010111001111010111010111010110110101010001001101100010  
1100001100100011110111010001101011011100011100011001100110010100110010011001101000110

# Cautious LSH

Joint work with Pham, Silvestri, and Stöckel



100011100010011011001000111101111000000100001101001111000001101000  
101100110001100011101101000110110111000111001100110011001100100  
01011001001101100001110001000001011011110000010100111100111011011011  
0111010010010011000111001111010110001011010011000110110001111000110101  
111101011101011010010001101011101010010110101001011011100100000111101000110  
0001100110110000100101001011000110010111010111001000001111010001111  
000111011100001000010010100001011000110010111001000001111010001111  
001011011101100101101001001001001011000110010111001010100101  
111010110000101001001010111010011011110010  
011111001011011000110101110010111100011  
0011001011000110101110111001111010100101  
00011100011001000011011011101011011111  
111100011100110001000100111010000010001101  
1110110110110100010100100110110101111000  
11110110010001100100001110001100110001000  
1011000001011011000111010111011110000001  
001011101001101100111111000001000010111  
01101001000001010011110011000011110000  
010111010000111101001100011101001111011001  
0010010001010011001111110000101111011011101  
1011111001111010101100000010111100100000010100  
11011010110110000110101100111010111110011101000010111000110010011  
1110010100010011001011110000111011011011110101  
011011000000111101110100000011010110001101011001001  
101000110110001101001111010111100001111000110101000101101100  
01001101110110001101001001001001001001001001001011  
10100100010100100100011111001101110011110101110100010011011000110  
11000011001000111101110100110101111000111100011001010011001101000110

Idea: Apply a weak  
LSH with constant  
collision probability.

# Cautious LSH

Joint work with Pham, Silvestri, and Stöckel

```
100011100010011011001000111101111100000010000111010011111000001101000  
10110011000110001110110100011010110111000111001100110011001010011001000100  
01011001001101100001110001000001011011110000010100111001110110110011001011  
1111010111101011010010000110101110100010110111100000110011110110011001100110  
00011001101110000100100101100011010001011111111000111100000100110010001111  
00011101110000100001001010000010110001101111100001101101100011010110110110  
11101011000101001001011101001101111001000010011011110111000110011110001100111  
10111110011110101011000001011110010000001010010101000111000011011011100011011  
11011010101101100000110101100111010111110011111011101010000101111000011001001  
1110010100010011001011110010000111011100101111001011011001011111010110011110  
01101100000011110111010000011010110001110010100011011001011001100101110010010  
1010001101110001101001110101101110011100011110001101000011010001010110110  
1111000111001100010011101000001001110000101110000101110100101010010100001001000  
0010111101001101100111111000001000010111001000111000100010101011000111100111  
0111010010010011001111010110001110100010110001011000110001111000111100011010  
0010110111011001011001011000101100010110001011000101100011000111100011110000010  
0111111001011011000110101110001011110001110100110110011011001101101101101110111  
001100101100011010111011010010111100011101000101111000110011110001100111100110  
000111000110010000111011011101011111110011001001111100011001111000110011111010  
111011011011000101001101101101111000000010010101011110001111100011111000111110  
11110110010001100100001110001000100010001000100010001000100010001000100010001000  
101100000101101100011101011111100000010001011110111011100000101100001011000010000  
0110101001000010100111100110000100110000100110000100110000101111000110011110001100  
010111010000111101001110100111100011001111000110011110001100111100011001111000110  
00100100010100100111110001011111000110011110001011111000101111100010111110001110  
110000110010001110110001001110001011111000101111100010111110001100110011001100110  
0100110111011000110100100100100101011101001001001010100101011101001000011010011000  
10100100010100100011111000110111100010111110001101111000110111100010100010011100010
```

# Cautious LSH

Joint work with Pham, Silvestri, and Stöckel

**RECURSIVE SUBPROBLEM**  $Q_0 \bowtie_r S_0$

**RECURSIVE SUBPROBLEM**  $Q_1 \bowtie_r S_1$

# Cautious LSH

Joint work with Pham, Silvestri, and Stöckel

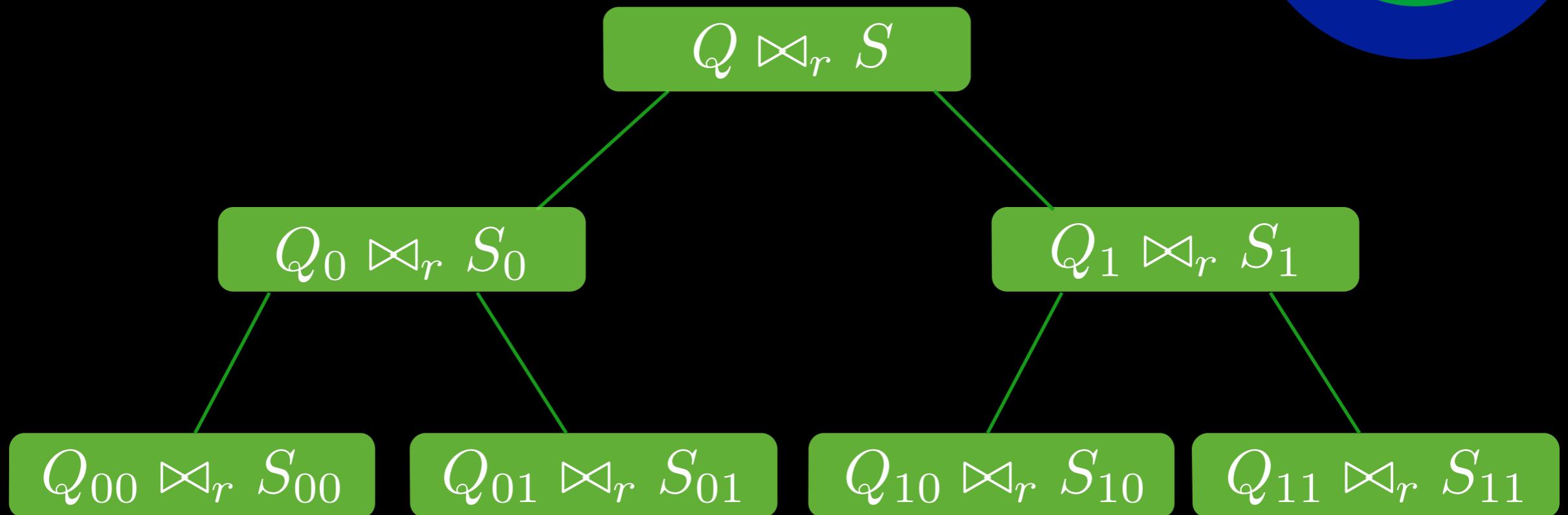
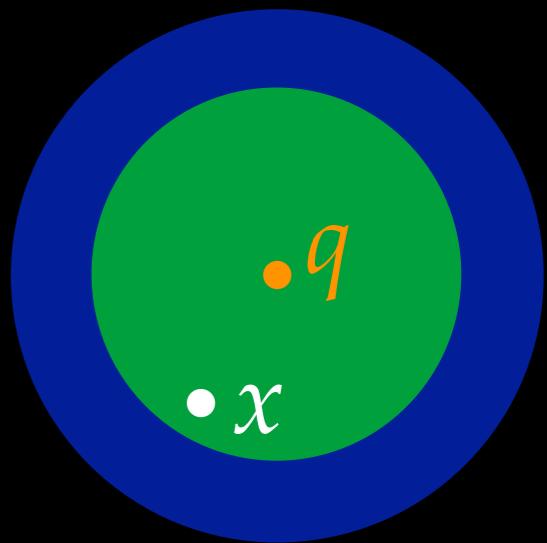
**RECURSIVE SUBPROBLEM**  $Q_0 \bowtie_r S_0$

Subproblems of size  $M$  or less require no further I/Os

**RECURSIVE SUBPROBLEM**  $Q_1 \bowtie_r S_1$

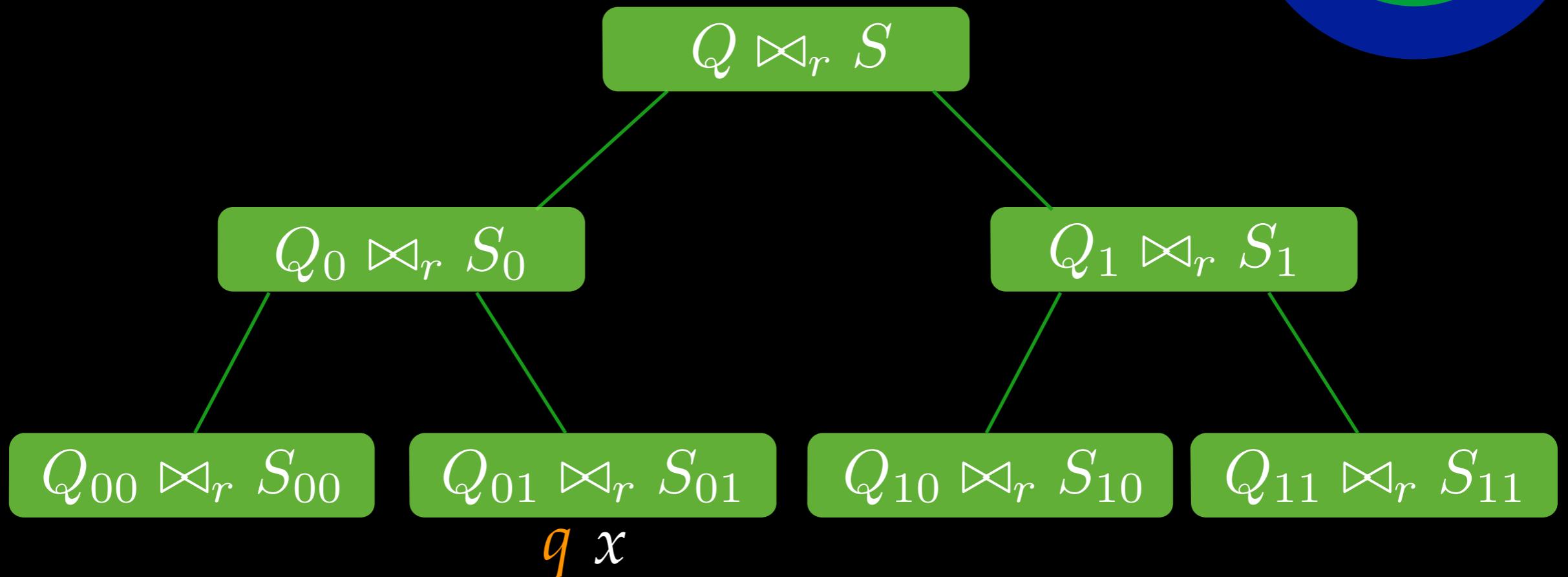
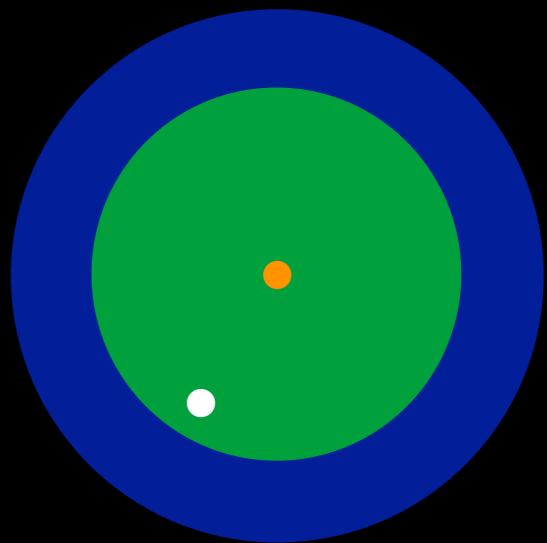
# Example, cautious LSH

- Assume weak LSH collision prob. 0.9.
- Prob. that  $q$  and  $x$  collide  $i$  times is  $0.9^i$ .
- If depth is  $\log n$ , success probability is  $\approx n^{-0.152}$ .



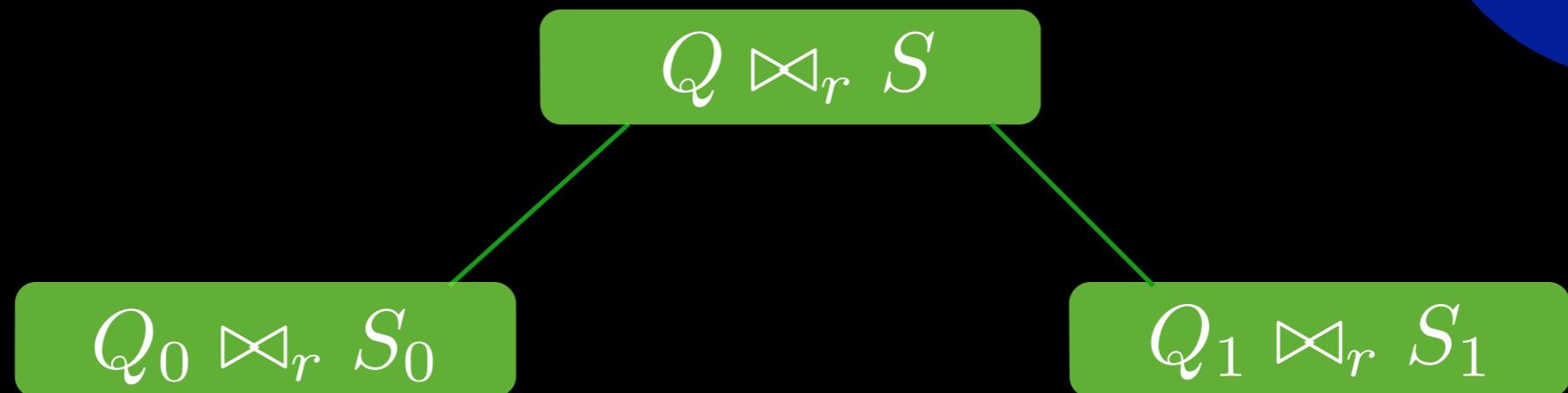
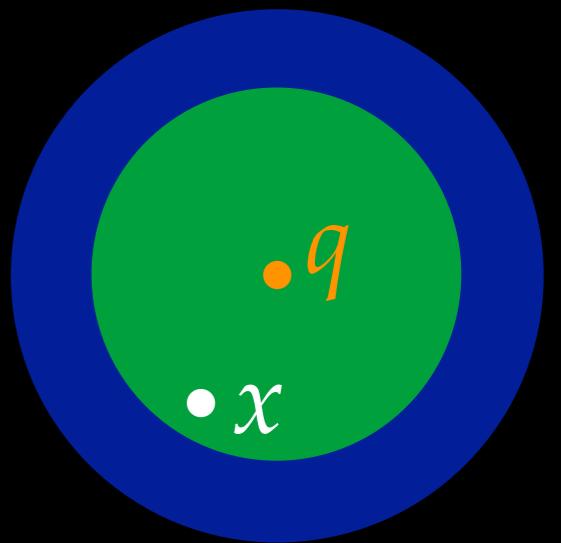
# Example, cautious LSH

- Assume weak LSH collision prob. 0.9.
- Prob. that  $q$  and  $x$  collide  $i$  times is  $0.9^i$ .
- If depth is  $\log n$ , success probability is  $\approx n^{-0.152}$ .



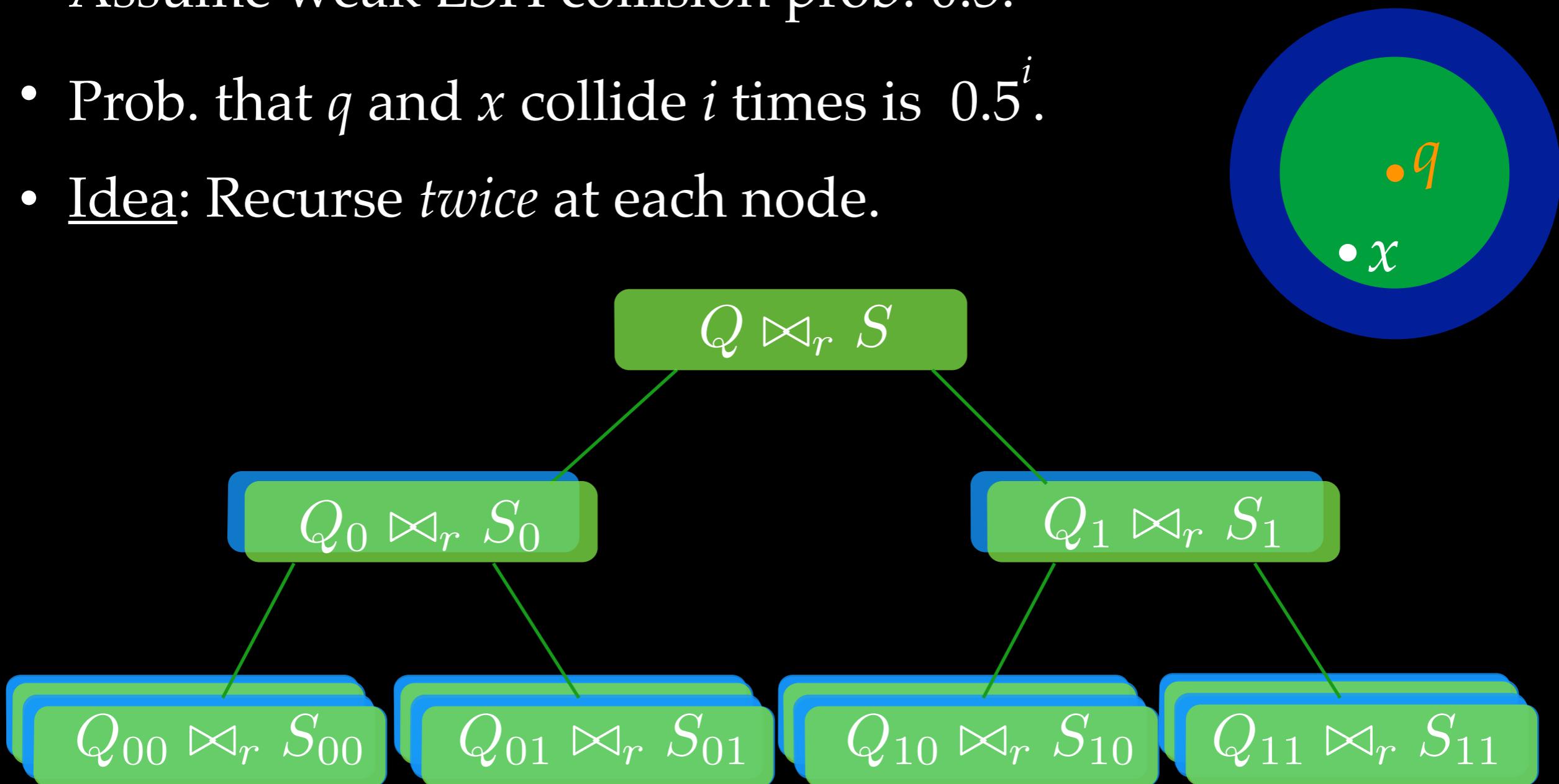
# Example 2, cautious LSH

- Assume weak LSH collision prob. 0.5.
- Prob. that  $q$  and  $x$  collide  $i$  times is  $0.5^i$ .



# Example 2, cautious LSH

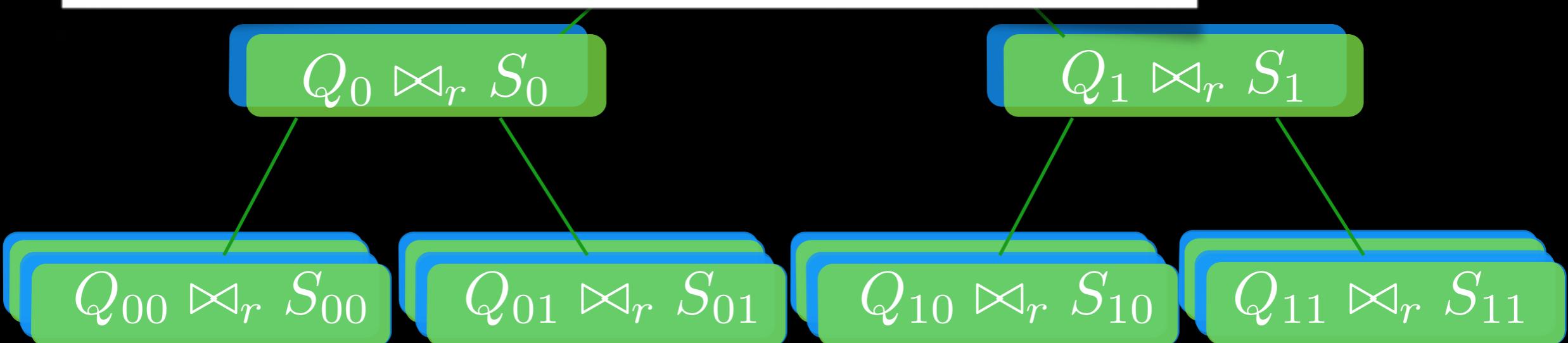
- Assume weak LSH collision prob. 0.5.
- Prob. that  $q$  and  $x$  collide  $i$  times is  $0.5^i$ .
- Idea: Recurse *twice* at each node.



# Example 2, cautious LSH

- Assume weak LSH collision prob. 0.5.
- Prob. that  $q$  and  $x$  collide  $i$  times is  $0.5^i$ .
- Idea: Recurse *twice* at each node.

#subproblems containing  $q$  and  $x$ :  
1 at each level, expected!



Aside:  
Don't be fooled by great expectations

# Aside: Don't be fooled by great expectations

*What is the expected TNT equivalent  
of asteroid impacts during this talk?*

# Aside: Don't be fooled by great expectations

*What is the expected TNT equivalent  
of asteroid impacts during this talk?*

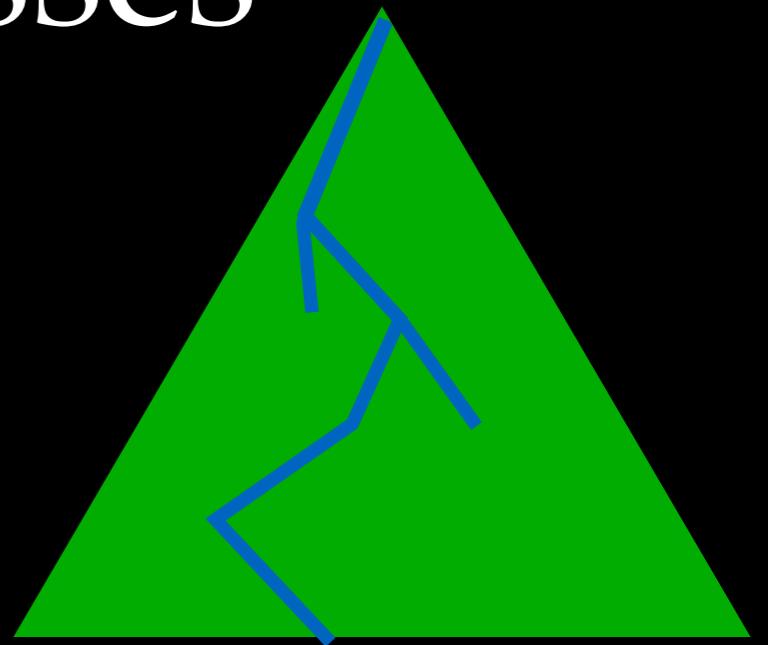
100 Teratons /  $66 \times 10^6$  years

48.05 kg/s (kilograms per second)

$\approx 0.3 \times$  rate of trash production by New York City ( $\approx 1 \times 10^7$  kg/day)

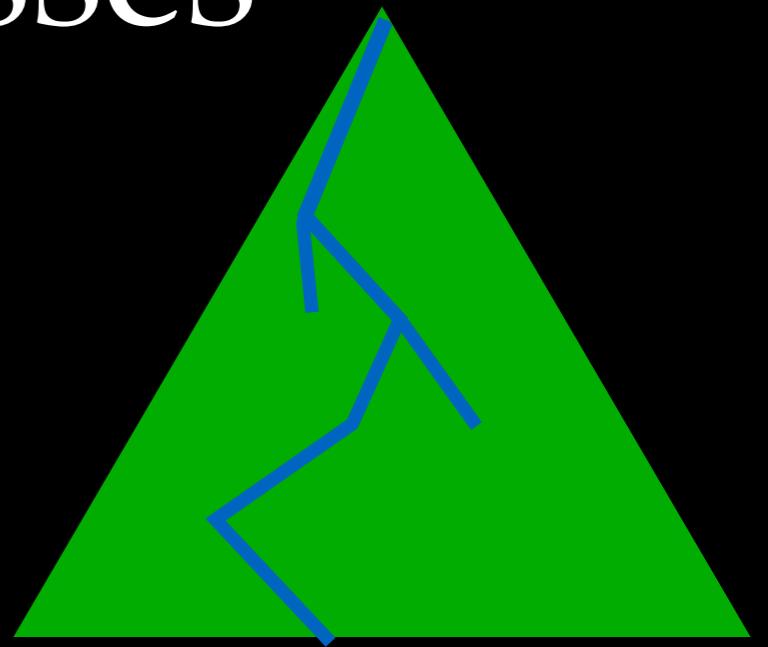
# Branching processes

- Abstract setting:
  - One pair  $(q,x)$  at root problem.
  - Generate  $t$  subproblems such that  $(q,x)$  is “reproduced” in each with probability  $\geq 1/t$ .



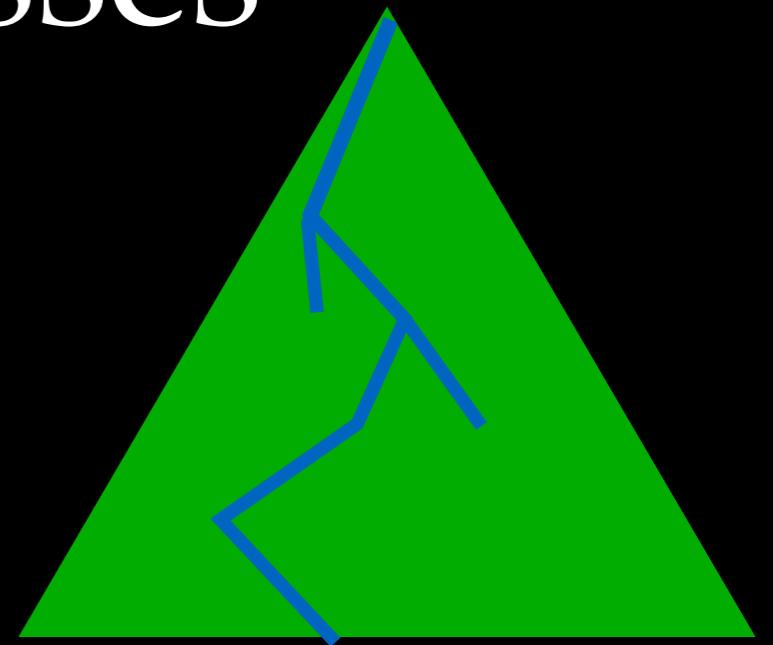
# Branching processes

- Abstract setting:
  - One pair  $(q,x)$  at root problem.
  - Generate  $t$  subproblems such that  $(q,x)$  is “reproduced” in each with probability  $\geq 1/t$ .
- What is the probability that  $(q,x)$  is *extinct* at recursive level  $i$ ?

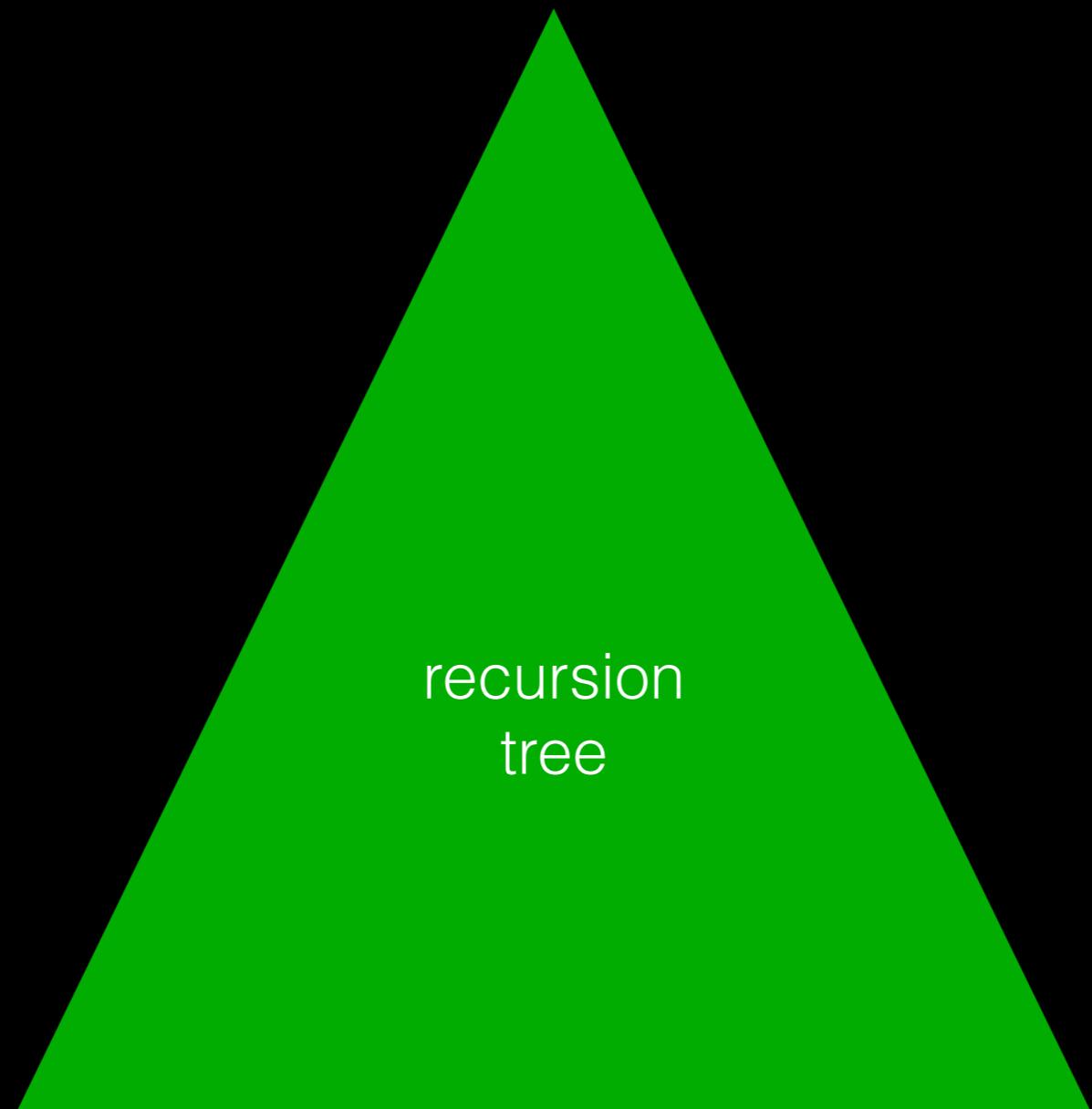


# Branching processes

- Abstract setting:
  - One pair  $(q,x)$  at root problem.
  - Generate  $t$  subproblems such that  $(q,x)$  is “reproduced” in each with probability  $\geq 1/t$ .
- What is the probability that  $(q,x)$  is *extinct* at recursive level  $i$ ?
  - From theory of branching processes:  $\Omega(1/\sqrt{i})$



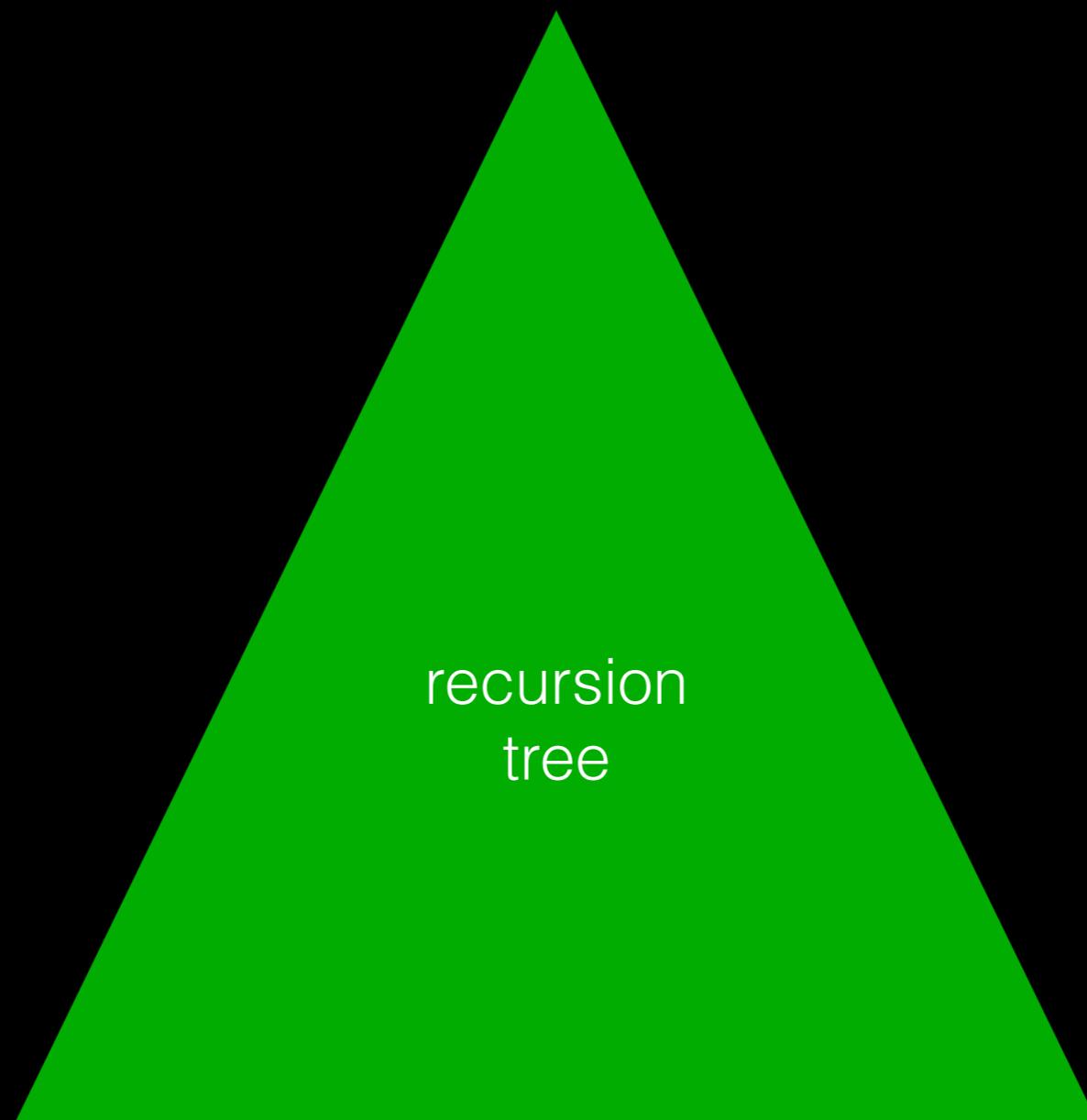
# I/O complexity, sketch



# I/O complexity, sketch

Size of all subproblems

$n$



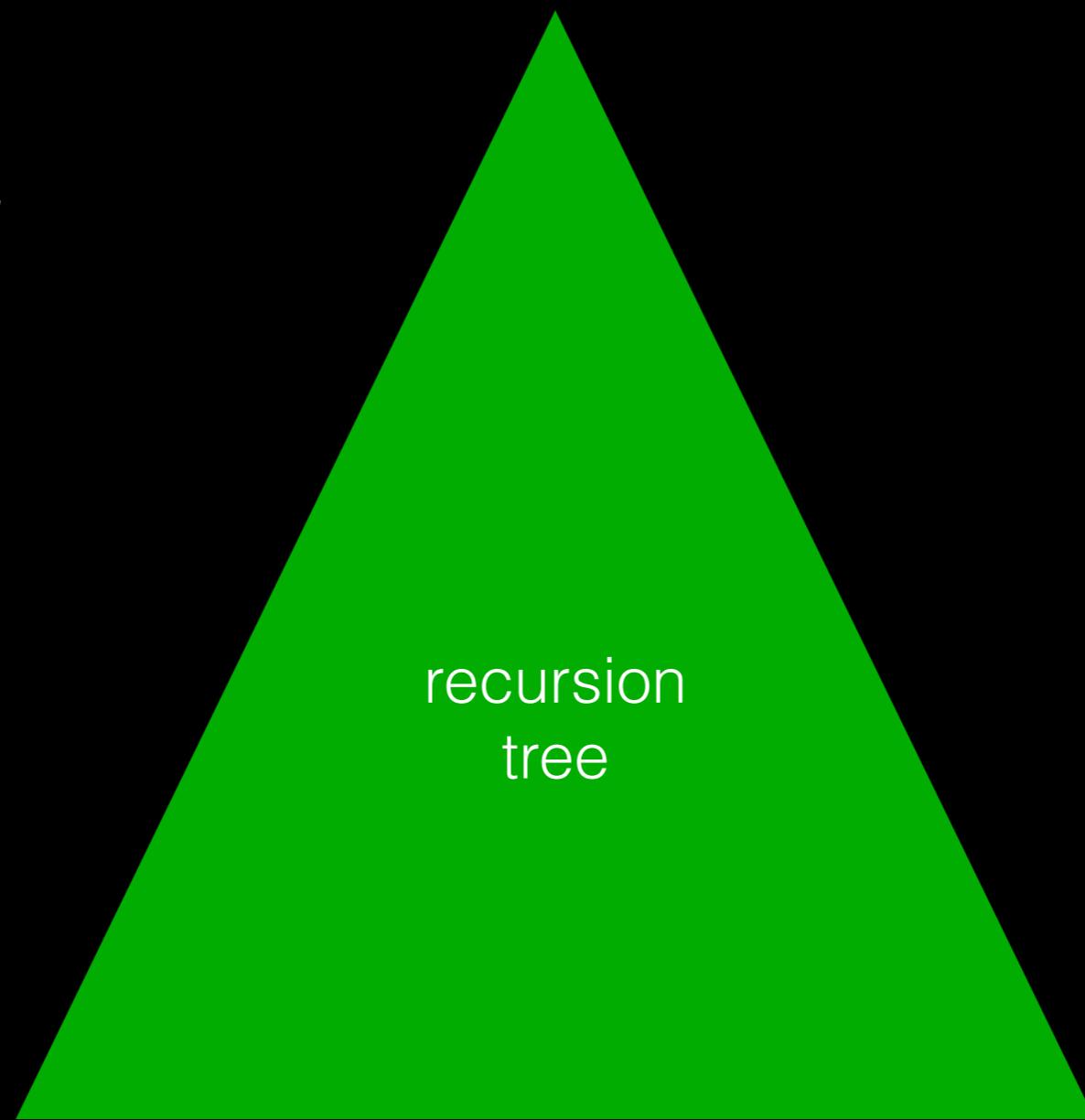
# I/O complexity, sketch

Size of all subproblems

$n$

$nt$

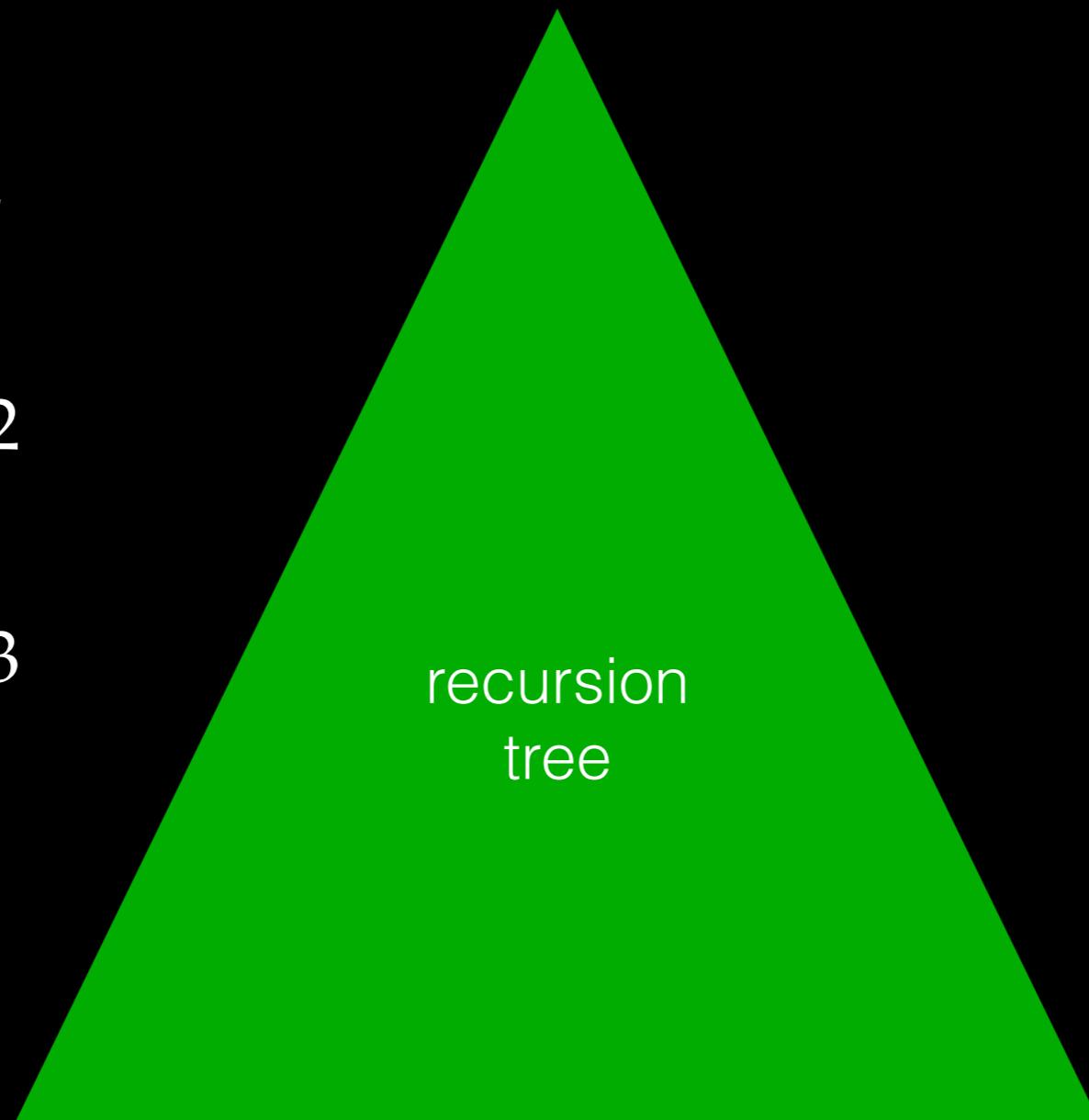
recursion  
tree



# I/O complexity, sketch

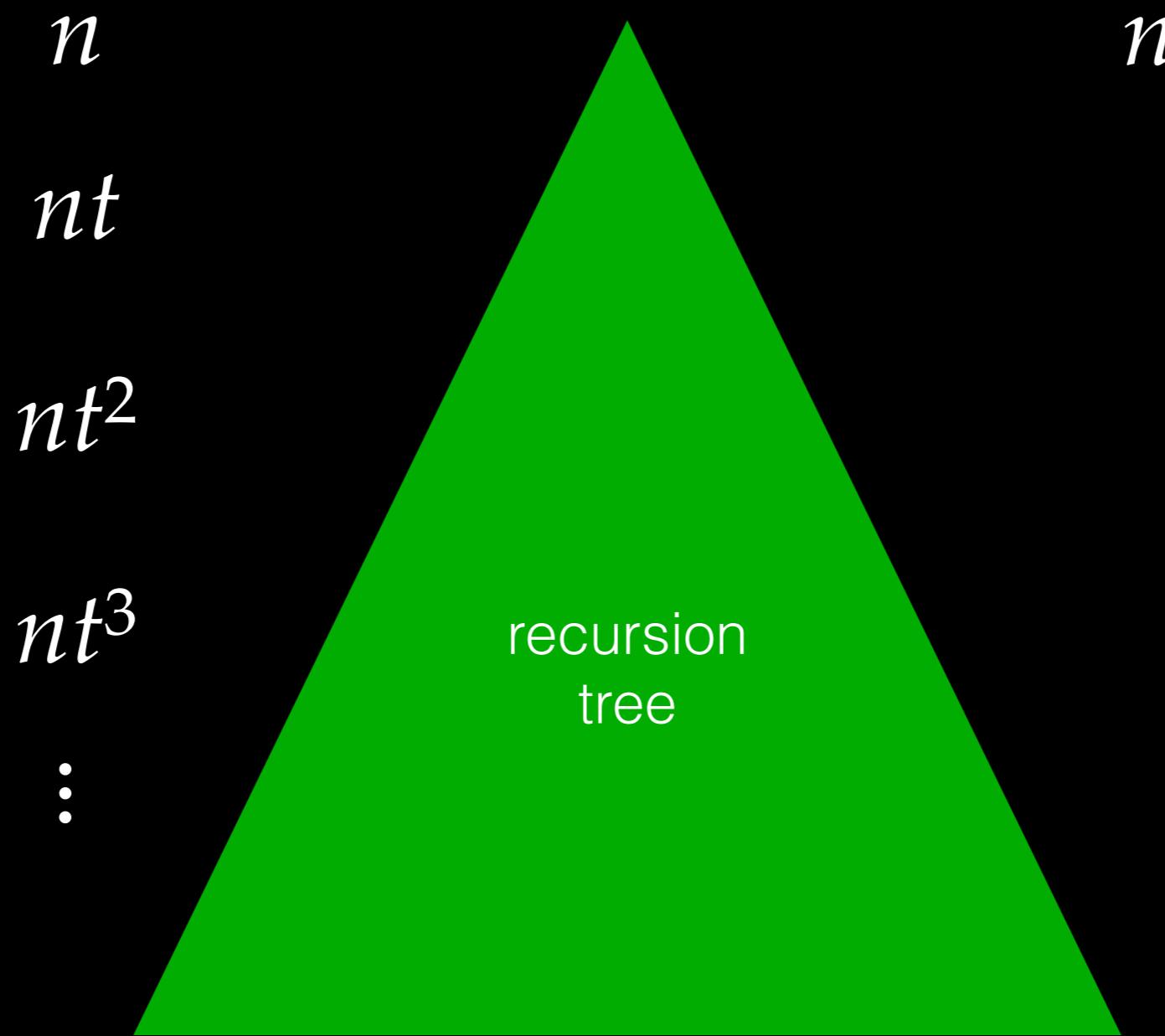
Size of all subproblems

$n$   
 $nt$   
 $nt^2$   
 $nt^3$   
⋮



# I/O complexity, sketch

Size of all subproblems



Expected average  
# points at distance  $> cr$ .

# I/O complexity, sketch

Size of all subproblems

$n$

$nt$

$nt^2$

$nt^3$

$\vdots$

recursion  
tree

$n$

$n/t^c$

Expected average  
# points at distance  $> cr$ .

# I/O complexity, sketch

Size of all subproblems

$n$

$nt$

$nt^2$

$nt^3$

⋮

recursion  
tree

$n$

$n/t^c$

$n/t^{2c}$

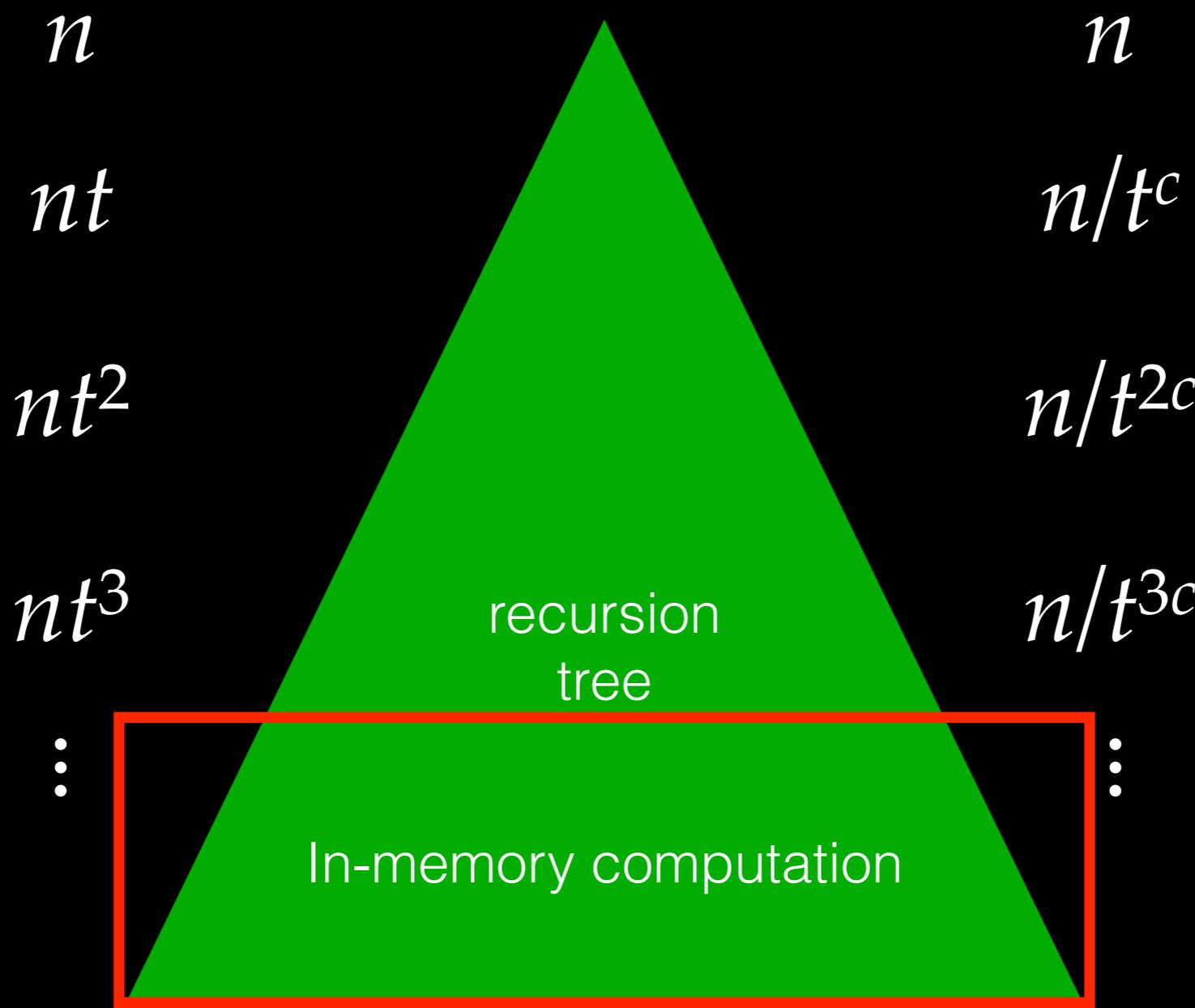
$n/t^{3c}$

⋮

Expected average  
# points at distance  $> cr.$

# I/O complexity, sketch

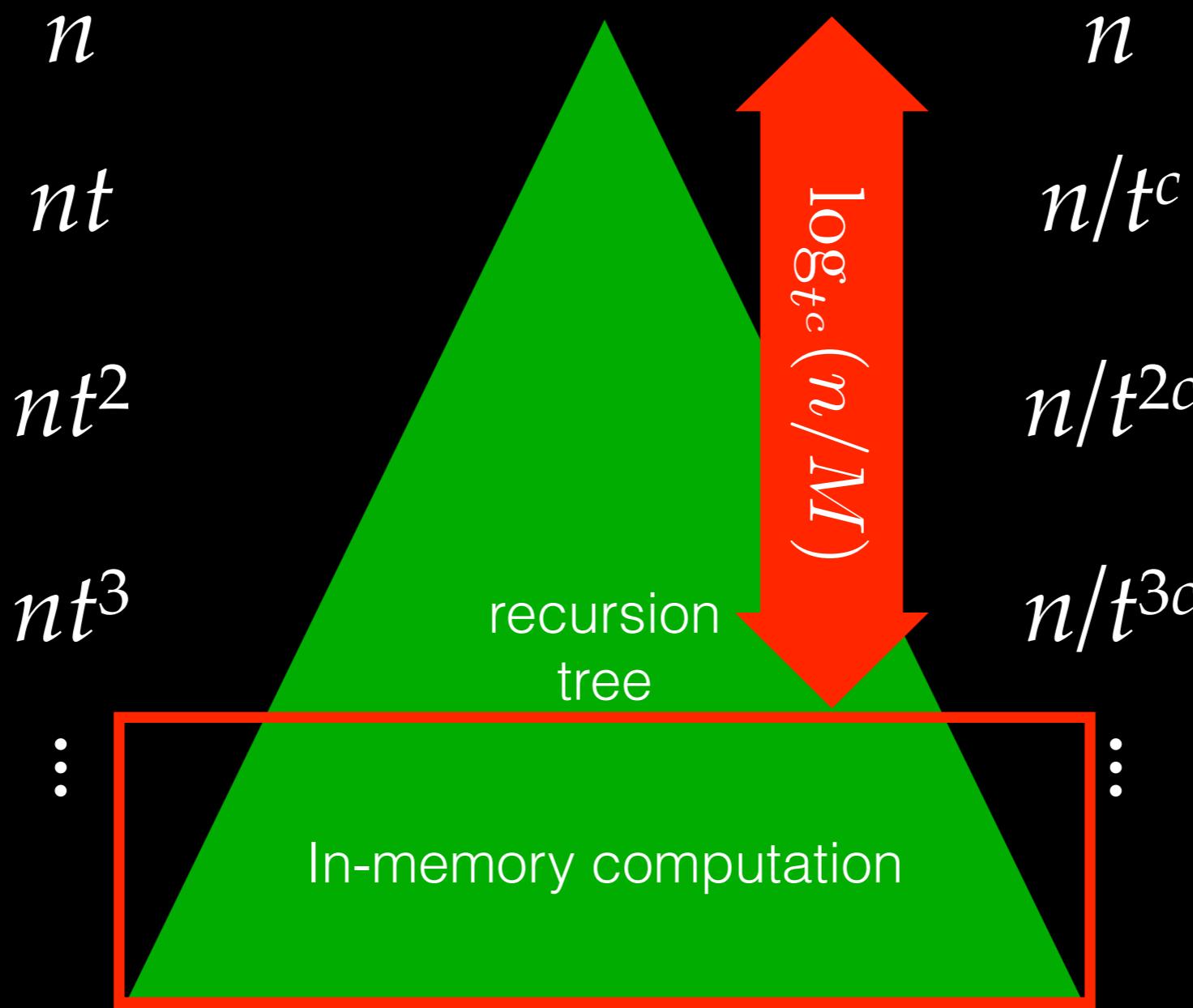
Size of all subproblems



Expected average  
# points at distance  $> cr$ .

# I/O complexity, sketch

Size of all subproblems



Expected average  
# points at distance  $> cr$ .

# I/O complexity

- Simplified:

$$O\left(\left(\frac{n}{M}\right)^{1/c} \left(\frac{n}{B} + \frac{|Q \bowtie_{\leq r} S|}{MB}\right)\right) \text{ I/Os}$$

Assuming  $|Q \bowtie_{cr} S|$   
is not much larger  
than  $|Q \bowtie_r S|$

# I/O complexity

Cost of  
reading input

- Simplified:

$$O\left(\left(\frac{n}{M}\right)^{1/c} \left(\frac{n}{B} + \frac{|Q \bowtie_{\leq r} S|}{MB}\right)\right) \text{ I/Os}$$

Assuming  $|Q \bowtie_{cr} S|$   
is not much larger  
than  $|Q \bowtie_r S|$

# I/O complexity

- Simplified:

$$O\left(\left(\frac{n}{M}\right)^{1/c} \left( \frac{n}{B} + \frac{|Q \bowtie_{\leq r} S|}{MB} \right)\right) \text{ I/Os}$$

Cost of  
reading input

Cost of  
generating  
output

Assuming  $|Q \bowtie_{cr} S|$   
is not much larger  
than  $|Q \bowtie_r S|$

# I/O complexity

The diagram illustrates the components of I/O complexity. Three blue boxes are arranged horizontally: 'Overhead' on the left, 'Cost of reading input' in the middle, and 'Cost of generating output' on the right. Arrows point from each box to a common formula below. The formula is:

$$O\left(\left(\frac{n}{M}\right)^{1/c} \left( \frac{n}{B} + \frac{|Q \bowtie_{\leq r} S|}{MB} \right)\right) \text{ I/Os}$$

Assuming  $|Q \bowtie_{cr} S|$  is not much larger than  $|Q \bowtie_r S|$

# I/O complexity

- Simplified:

$$O\left(\left(\frac{n}{M}\right)^{1/c} \left(\frac{n}{B} + \frac{|Q \bowtie_{\leq r} S|}{MB}\right)\right) \text{ I/Os}$$

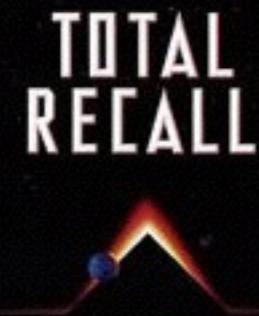
- In general:

$$\tilde{O}\left(\left(\frac{n}{M}\right)^{1/c} \left(\frac{n}{B} + \frac{|Q \bowtie_{\leq r} S|}{MB}\right) + \frac{|Q \bowtie_{\leq cr} S|}{MB}\right) \text{ I/Os}$$

# Part C: Exact similarity join

# Total recall?

Get ready for the ride  
of your life.



- False negative problem of classical LSH:  
No collision for a close pair with prob.  $P > 0$ .

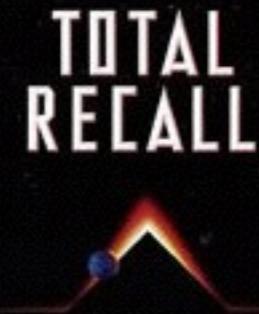
# Total recall?



- False negative problem of classical LSH:  
No collision for a close pair with prob.  $P > 0$ .
- Recall can be made  $1-P^k$  by repeating  $k$  times.

# Total recall?

Get ready for the ride  
of your life.

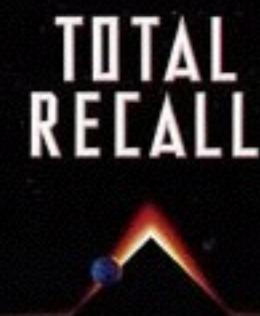


- False negative problem of classical LSH:  
No collision for a close pair with prob.  $P > 0$ .
- Recall can be made  $1-P^k$  by repeating  $k$  times.
- Many recent papers deal with deterministic, *exact* similarity joins that achieve 100% recall.

# Total recall?

- False negative problem of classical LSH:  
No collision for a close pair with prob.  $P > 0$ .
- Recall can be made  $1-P^k$  by repeating  $k$  times.
- Many recent papers deal with deterministic, *exact* similarity joins that achieve 100% recall.

Get ready for the ride  
of your life.



A Primitive Operator for Similarity Joins in Data Cleaning

Efficient Similarity Joins for Near Duplicate Detection

WWW 2007 / Track: Data Mining

Session: Similarity Search

Scaling Up All Pairs Similarity Search

# Correlated LSH

- Idea: Choose hash functions  $h_1, h_2, \dots$  to ensure at least one collision if distance is at most  $r$ .

## Efficient Exact Set-Similarity Joins

Arvind Arasu  
Microsoft Research  
One Microsoft Way  
Redmond, WA 98052  
[arvinda@microsoft.com](mailto:arvinda@microsoft.com)

Venkatesh Ganti  
Microsoft Research  
One Microsoft Way  
Redmond, WA 98052  
[vganti@microsoft.com](mailto:vganti@microsoft.com)

Raghav Kaushik  
Microsoft Research  
One Microsoft Way  
Redmond, WA 98052  
[skaushi@microsoft.com](mailto:skaushi@microsoft.com)

### ABSTRACT

Given two input collections of sets, a *set-similarity join* (SSJoin) identifies all pairs of sets, one from each collection, that have high similarity. Recent work has identified SSJoin as a useful primitive operator in data cleaning. In this paper, we propose new algorithms for SSJoin. Our algorithms have two important features: They are exact, i.e., they always produce the correct answer, and they carry

| <i>City</i>   | <i>State</i> |
|---------------|--------------|
| Los Angeles   | CA           |
| Palo Alto     | CA           |
| San Diego     | CA           |
| Santa Barbara | CA           |
| San Francisco | CA           |
| Seattle       | WA           |
| ...           | ...          |

| <i>City</i>   | <i>State</i> |
|---------------|--------------|
| Los Angeles   | California   |
| San Diego     | California   |
| Santa Barbara | California   |
| San Francisco | California   |
| Sacramento    | California   |
| Seattle       | Washington   |
| ...           | ...          |

# Basic correlated LSH partitioning

```
100011100010011011001000111101111000000100001101001111000001101000
101100110001100011101101000110110111000111001100110011001010011001000100
010110010011011000011100010000010110111100000101001111001110111011011001011
011101001001001100011100111101011000101100110001101100011110001111000110101
1111010111010110010001101011010100101101011110011001111000000111101000110
00011001101100001001010011001100101111110011110000100110010001100110010001111
0001110111000010000100101000001011000110111110001101101100011010110110110110
00101101110110010110010010101001011001010111100110010000110001110000011000111000
111010110000101001001011101001101111001000001001101111101111000110011110000000
01111100101101100011010111001111000111100110110011011001011011011011011011110111
00110010110001101011101100111101010011110011001111001100111101011000111000110100
00011100011001000111011011011011111100110011001111010110010111100010101111010101
1111000111001000100110110000100011011110000100011011100001011101001010000110010000
1110110110110001010011011001101111000000010010100111100011111010011001111010010111
1111011001000110010100011000110011001100111100000001001000000100111101001111100011110
10110000010110110001110101111100000100001011100100011100010001011000100011000010000111
00101110100110110011111000001000010111001000111000100010110001000101010110001111000111
0110101001000010100111000110000111000001001100010010001010111100011001111000101110011
01011101000011110100111000111001110011001011011100010001010101110001100111100011110110
001001000101001100111110000101001111000101100111100110111010110110000101110110011100110
10111110011110101011000001011110010000001010010101000111000011011011000011011011001010
11011010110110000110101100111010111110011111001111101110100001011110000110010011
111001010001001100111100000111011011100101101101100101101101101100110011110101110101
01101100000011110111000001101011000011100101100011100010101100110011001011100100101
10100011011000110100111101011110011100001111000110100011010001101000110100010101101100
```

# Basic correlated LSH partitioning

```

10001110001001101101001000111101111000000100001101001111000001101000
10110011000110001111011000110110111000111000110011000110011001001100100
010110010011011000001110001000001011011110000010100111001110110110011011
01110100100100110001110011110101100010110011000110110001111000111100011010
11110101110101100100011010010111010100101101011110010011000000111101000110
000110011011000001001010010110001011111100011110000100110010001111
0001110111000001000010010100000010110001101111100011011011000110110110110
001011011101100101100101100100100101100101011101001010101010000110001110000010
1110101100001010010010101110100110111100100000100110111110111100011001111000000
01111100101101100001101011100111100011110011011001101100110110010110111010111
001100101100011010111011001111010100111100011001110011001100110011001100110010
00011100011001000011011011011011111100110011001111001100110011001100111101010
1111000111001100010011011000100111010000010001101110000101110100101010010100001001000
11101101101101000101001101101101111000000010010100111100011111010011001111010010111
1111011001000110010001100110011001100110011001100110011001100110011001111000111100001110
1011000001011011000111010111110000001000101110110000010001011101100000101100001000000111
001011101001101100111111000001000010111001000111000010011001000100101010110001111000111
01101010010000010100111001100001110000010011000100100010010101111001100110011110011110011
010111010000111101001110011001111001100110011001100110011001100110011110001111000111101101
0010010001010011001100111110000101001111000101100111100011001100110011001100001011101100110
10111110011110101011000000101111001000000101001010100011100001101101100001101101100001101101010
1101101011011000001101011001110101111100111110011111001111101011100001011110000110010011
111001010001001100101110010000011101101110010101101100101101100101101111010110011110101
01101100000011110111010000001101011000111000110010100011010011001011001100101110010010
1010001101100011010011110101111001110000111100011001100011010001101000110100010101101100
0100110111011000011010010010010010010010111010010010010010010010010010010010000110100110001011
10100100010100100100011111100110101110011110011011011011011011010100010011001001101100010
11000011001000111101110100011010110110111000111000110011000110100011010011001010011001100100110

```

$h_1$

# Basic correlated LSH partitioning

```
100011100010011011011001000111101111100000010000111010011111000001101000  
10110011000110001110111010001101101101101110001110001100110001100110010011000100  
010110010011011000011100010000010110111100000101001111001110011001100110011001011  
011101001001001100011100111110101100010110100110001101100011110001111000110101  
1111010111010110010001101011011010001011011010001011011011110011000000111101000110  
000110011011000010010010110001100101111111001111000010011000100110010001111  
00011101110000100001001010000010110001101111100011110000100110010110110110110  
001011011110110010110010010010110010101110100101011101001010101010000110001110000010  
11101011000010100100101011101001101111001000001001101111101111000110011110000000  
01111100101101100011010111000101111000111011011001101100101101101101101101110111  
001100101100011010111011001111000101111000111011001110011100110011100110001110010  
00011100011001000011011011010111110011001111000110011110011001111001011111010101  
1111000111001000100110100010001101101000001000110110000101110000101110001001000010010000  
1110110110110001010010011011010111100000001001010011110001111101001100111100011001111000000  
11110110010001100100011001100110011001100110011001100110011001100110011001100110011001100111  
11110110010001100100011001100110011001100110011001100110011001100110011001100110011001100111  
101100000101101100011101011111000000100010111011100000100010111011100000101100001000000111  
001011101001101100111111000001000010111000100011100000100011100010001010101011000111000111  
011010010000010001110001100001110000010001100001000110000100011000010001100001000111000111  
010111010000111100011000111000111000110001100011000110001100011000110001100011000110001110001  
00100100010100010001100011000110001100011000110001100011000110001100011000110001100011000110  
1011111001110101011000000101111001000000010100000001010000000101000000010100000001010000000101000000  
11011010110110000110101100111010111110011111001111100111110011111001100001011110000110010011  
111001010001001100111100000111011011011001100110011001100110011001100110011001100110011001100110011  
01101100000011110111010000011010110000110101100001101011000011010110000110101100001101011000011010110000  
101000110110001101011101011110000110001110001100011000110001100011000110001100011000110001100011000110000
```

01001101110110000110100100100100100101011101001001001001010100100100001101001100001101001100001011  
1010010001010010010001111110011010110011100011110001111000111100011110001111000111100011110001111000110  
11000011001000111101110100011010110110111100011110001111000111100011110001111000111100011110001111000110

$h_2$

# Basic correlated LSH partitioning

100011100001001101101100100011110111110000001000011101001111100000011010011111000001101000  
1011001100011000111101100100011010110111000111001100011001100110010100110011000100  
0101100100110110000111000100000101101110000010100111001100111001100111011001100101  
01110100100100110011110101100101101001101001101001101100111100011110001111000110101  
11110101110101101001000011010110100101101001101111001001100110000001111000110  
0001100110111000010010010110001100101111111100011110000010011001000111100011110001111  
000111011100001000010010100000010110001101111000011011011011000110101101110110110  
001011011110110010110100100101100101101001010111101001010101010100000110001110000010  
111010110000101001001011010011011110010000100110111110111000110011110001100111100000  
01111100101101100011010111000101111000111011010110011011001011011011011011101110111  
0011001011000110101110110011101001011110011100111100110010011110011011001011000111001100  
000111000110010000110110110101111111001100100111100110010011110101100101111101011110101  
1111000111001100010011011010000100011011100001011100001011101001010100100101000010010000  
1110110110110001000100110110101111100000001001010100111110001111101001111101010010111  
111101100100011001010000110001100011000100010100100000010011110111010011111000111100011110  
1011000001011011000110101111100000010001011110111011101100001011000010110000100000111  
0010111010011011001111100000100001011100100011100010001001010101011000111100011110001111  
01101001000001010011110011000011100001001100100010010101010111001111000111100011110001111  
010111010000011101001110100111100110010010101010110110110011110001111000111100011110001111  
0010010001010011000111111100010111001111100110111110110111101011000010111011001111000111100  
10111110011110101011000001011110010000001010010101001111000011011011100001101101110001010  
1101101010110000011010110011101011111001111100111110111010100001011110000110011110000110010011  
111001010001001100101111000001110110111001101110010110110010110111101011111010111101011110101  
011011000000011101110100000011010110000111000101001000110101100110011011001100101111000100101  
1010001101110001101001111010111100111000011110001101011000110110001101000110100010101101100  
  
010011011101100001101001001001010111010010010011010111010010010011010101000001101001100001011  
10100100010100100010001111111001101100111110110111011011101011101010100010011001101100010  
11000011001010011110111010001101011101101111000111000111000110001100101010110011001100110100110

h3

# Basic correlated LSH partitioning

```

100011100010011011001000111101111000000100001101001111100000010101000
101100110001100011101100011011001101101110001110011000110011001100100100
0101100100110110000111000100000101101111000000101001110011110111010011001011
011101001001001100011100111101011000101100110001101100011110001111000110101
1111010111010110010001101011010100101101011110000001011110011000000111101000110
000110011011000010010100110011001011111100011110000001001100100110010001111
000111011100001000010010100000010110001101111100001101101100011010110110110110
001011011101100101100100101010010010110010101110100101010101000001100011100000010
11101011000010100100101011101001101111000000100110111110111100011001111000000
0111110010110110001101011100111100011110011011001101100110110011011011011110111
001100101100011010111001110011010101111000110011100110011100110110001110011010
00011100011001000011011011011011011111001100110011100110110010111110101010
1111000111001100010011011000000100011011100000101110000101110010010010100001001000
1110110110110100010100110110110111100000001001010011110001111010011001010111
1111011001000110010001100100011001000110010001100100011001000110010001100100011110
10110000010110110001101011111000000100010111011100000100010111011100000100000111
00101110100110110011111000001000010111001000111000010001110001000101010101100011100111
011010100100000101001110001100001110000010011000100100101011100110011001011100111
01011101000011110100111000111000111000101011011100110011100110011100111001110111
001001000101001100111110001100011110001011000111100010110000010111101001111000011110
10111110011110101011000000101111001000000010100101010001110000110111011001111001010
110110101101100000110101100111010111110011111001111101110100001011110000110010011
1110010100010011001011110000011101101110010110110110110111101011111010110011110101
0110110000001111011110100000011010110000111000101000110101100110011001011100100101
10100011011000110100111101011110011100001111000110101100001101000110100010101101100

```

$h_4$

# Basic correlated LSH partitioning

For Hamming distance  $\leq 3$ ,  
a collision is guaranteed!

```
100011100010011011001000111101111000000100001101001111000001101000  
1011001100011000111101100011011001101110001110011000110011001001100100  
01011001001101100000111000100000101101111000001010011100110111010011011  
0111010010010011000111000100000101101111000001010011100110111010011011  
11110101110101100100100  
00011001101100000100100  
0001101110000010000100  
0010110111011001001011  
11101011000010100100101  
0111110010110110000110  
00110010110001101011101  
000111000110010000111011011110101111101111011011110111101111011110111  
11110001110011000100111010000010001101110000101110100101010010100001001000  
11101101101110001010011011011110000000100101011110001111101001010111  
1111101100110001100100001100011010001100010010000001001111010011111000011110  
101100000101101100011101011111000000100010111011100001011000010110000100000111  
001011110100110110011111000001000010111001000111000100010101011000111100111  
01101001000001001111000110000111000001001100010010101111001100111100111100111  
0101110100001111010011101001111001100101101110100101111010011110011001111011101  
0010010001010011001111100011000111100010111001101110101110100001011101100111011  
1011111001111010101100000101111001000000101001010100011100001101101110011011101010  
1101101011011000011010110011101011111001111101111101110100001011110000110010011  
111001010001001100111100000111011011100101110110111110110111110110111110110111101  
0110110000001111011110000011010110000111000111001010011011001011001100101111001001  
1010001101110001101001111010111100001111000111100011010001101000110100010101101100
```

# Basic correlated LSH partitioning

For Hamming distance  $\leq 3$ ,  
a collision is guaranteed!

To bound probability of  
collision for distance  $> 3$   
randomly permute the  
dimensions

# Basic correlated LSH enumeration

```

10001110001001101100100011110111110000001000011010011110000001101000
10110011000110001111011000110110110111000111000110011000110011001100100
01011001001101100001110001000001011011100000101001110011011001100110011011
011101001001001100011100111110101100010110100110001101100011110001111000110101
111101011101011010010000110101101101000101101000101101011110010011000000111101000110
0001100110110000100101001100011001011111100111111110011110000100110010001111
0001110111000010000100101000001011000110111110001101101100011010110110110110
00101101110110010110010010010010110001010111010010101010101010000110001110000010
1110101100001010010010101110100110111100100000100110111110111100011001111000000
011111001011011000011010111000110111100011101101100110110010110110110110111010111
001100101100011010111011001111010100110111100111110011100111100110011101100111010
0001110001100100001101101101101111100111110011001111010111101001111010010111101010
1111000111001100010010011011000010001101110000101110000101110001001001010100001001000
111011011011010001010011011011011110000000100101011110001111100011001111010010111
1111011001000110010001100110001100110001100110001100000010011110001111100011001111000
101100000101101100011010111110000010000101110001000111011100001011000010110000100000111
00101110100110110011111000001000010111000100011100010001110001000101010110001111000111
01101001000001010011100011000011110000010011000100110001001001010111100011001111000111
010111010000111101001100011100011100011100011100011100011100011100011100011100011100011
0010010001010011000111110000101001100011111000010100110001111100001010011000111110000110
101111100111101010110000010111100010000001010000001010000001010000001010000001010000001010
11011010110110000011010110011101011111001111110011111100111111001111110010000101111000011001
1110010100010011001011110000011101101100110011001011101100101110110010111110110011110101
0110110000001111011110000011010110000110101100001101011000011010110000110101100001101011000101
10100011011100011010011110101111000011010110000110101100001101011000011010001101000101101100

```

$h_{12}$

# Basic correlated LSH enumeration

```

1000111000100110110100100011110111110000001000011010011111000000101000
10110011000110001110110100011010110111000111001100011001100110010100110011000100
010110010011011000001110001000001011011110000001010011100111001100110011001100110011011
0111010010010011000111001111010110001011010011000110110001111000111100011110001111000110101
11110101110101100100011000110011110101100010110100101101011110010011000000111101000110
000110011011100001001010010110001100111111110001111000010011001000110010001111
00011101110000010000100101000001011000110111110001101101101100011010110110110110
001011011101100101101010010010010110010101110100101010111010010101010000110001110000010
11101011000010100100101011101001101111001000001001101111101111000110011110000000
0111110010110110000110101110001101111000111100111011011001101100101101101101101110111
0011001011000110101110100111101010101011110001101111000110110011110001101100011100110
000111000110010000110110110110110111110011001100111100011011001111000110110010111101010
1111000111001100010011011001100110000010001101110000101110000101110100101010010100001001000
111011011011010001010011010011011011110000000100101001111000111110100101010111100011111010010111
111101100100011001000110011001100000100010001100000001001000000100111100011111000111110001111
10110000010110110001110101111100000010001011110111011000001011011101100000101100001000000111
00101110100110011111000000100001011100100011100000010011001000111000100010101011000111000111
0110100100000101001110001100000010000000100110000001001000100101011100011000000101011100011100011
0101110100000111101001110000001100000011000000110000001100000011000000110000001100000011000000110
0010010001010001100111110000001000000010000000100000001000000010000000100000001000000010000000100
101111100111101010101000000101110010000000101000000010100000001010000000101000000010100000001010
1101101011011000001101011001110101111100111110011111001111100111110011111000001011110000110010011
1110010100010011001011100100000011011011001110110010101011011001000101101111010111101011110101101
011011000000111101110100000011010110000001101000000110100000011010000001101000000110100000011010000001
101000110111000110100111101011110011100000111000000111000000111000000111000000111000000111000000110
01001101110110000110100100100100100101011101001001001001010111010010010010101110100100100101011011
101001000101001001000111111001101011100111100111100111100110110101110101010100010011010110001100010
11000011001000111101110100011010110110110111000011100011100011000110001100011000110001100011000110

```

$h_{13}$

# Basic correlated LSH enumeration

```

100011100010011011010010001111011110000001000011010011111000000110100100111110000001101000100
101100110001100011110110001101101110001110001100110001100110011001100110011001100110011001100
0101100100110110000011100010000010110111100000010100111100110011001100110011001100110011001100
01110100100100110001110011111010110001011010011000110110001111000111100011110001111000110101
11110101111010110100100001100011011001111010110001011010011011110011000000111101000110
000110011011100001001010010110001101111111100011110000010011110000100110010001111
00011101110000010000100101000000101100011011111111000111100000100110011010110110110
0010110111101100101101001001001001011000101011101001010101010101000001100011100000010
11101011000010100100101011101001101111001000001001101111101111000110011110000000
011111001011011000011010111000110111100011110001101100110110011011001101101101101110
00110010110001101011101100111101010111100011011110001101100110110011011001110011010
0001110001100100001101101100111101011111100011011110001001111010110010111110101010
111100011100110001000110110110000001000110111100001011101001010100101000010010000
1110110110110100010100110110110111100000001001010011110001111100011001111000110010
11110110010001100100011010001100000000100010001000000010011110001111100011001111000
10110000010110110001101011111000000100010111101110000001000101111011100000010110000
00101111010011001111110000010000101110010001110000100011100010001010101011000111000
01101001000001010011100011000011100000100110001001000100101010111001100101110010110
0101110100000111101001110000111000001110000011100000111000001110000011100000111000
00100100010100110011000001100000011000000110000001100000011000000110000001100000011000
1011111001111010101100000010111100100000001010010101000111000001101100000101110000
110110101101100000110101100111010111110011111001111101110100001011110000011001001
1110010100010011001011100100000111011011100101011011011011110101100111101011001110
0110110000000111101110100000011010110000111000001100000011000000110000001100000011000
10100011011100011010011110101111001110000111100001111000011010110000110100001100000
01001101110110000110100100100100100100101110100100100100100100100100000110100110000
10100100010100100100011111100110101110011110101110101110101110101110101000010011000
1100001100100011110111010001101011011100011110001100110001100110011001100110011000

```

$h_{14}$

# Basic correlated LSH enumeration

10001110001001101101001000111101111000000100001101001111100000010101000  
101100110001100011110110001101101110001110011000110011001100110010100100  
01011001001101100000111000100000101101111000000101001111001111011001100101  
011101001001001100011100111101011000101100110001101100011110001111000110101  
11110101110101100100011000110110010110010110010110011110001100000011110100110  
00011001101110000100101001011000110111111000111100000100110010001111  
00011101110000010000100101000001011000110111110001101100011010110110110  
001011011110110010110010110010010010110010111010010101010100000110001110000010  
11101011000010100100101011110001100  
0111110010110110000110  
00110010110001101011101  
00011100011001000011101  
11110001110011000100010  
11101101101110100010100  
1111011001000110010100  
10110000010110111000111  
0010111010011011001111  
0110100100000101001111  
01011101000011110100111  
0010010001010011001100  
1011111001111010101010  
11011010110110000011010  
11100101000100110011110  
01101100000011110111000001101011000111000101100110011001011001100100101  
101000110111000110100111101011110001111000011110001101011000011010001101000101100  
010011011101100001101001001001001001001001011101001001100100100110010011000011010001011  
101001000101001001000111111001101011100111101011101011101011001000100110110001001101100010  
11000011001010001111011101000110101101101110001111000110011000110010100100110011000110

For Hamming distance  $\leq 2$ ,  
a collision is guaranteed in

$h_{12}, h_{13}, h_{14}, h_{23}, h_{24}, h_{34}$

$h_{14}$

# Result in LSH framework

- The Gionis et al. LSH achieves:

$$\Pr[h(q) = h(x) \mid d(q, x) = cr] \leq \Pr[h(q) = h(x) \mid d(q, x) = r]^c$$

# Result in LSH framework

- The Gionis et al. LSH achieves:

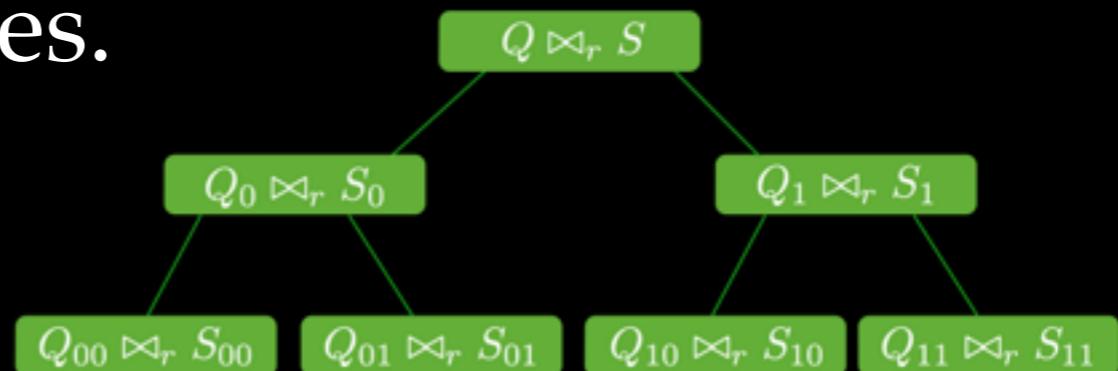
$$\Pr[h(q) = h(x) \mid d(q, x) = cr] \leq \Pr[h(q) = h(x) \mid d(q, x) = r]^c$$

- Arasu et al. construction combining partitioning, enumeration, and random permutation:

$$\Pr[h(q) = h(x) \mid d(q, x) = cr] \leq \Pr[h(q) = h(x) \mid d(q, x) = r]^{0.36c}$$

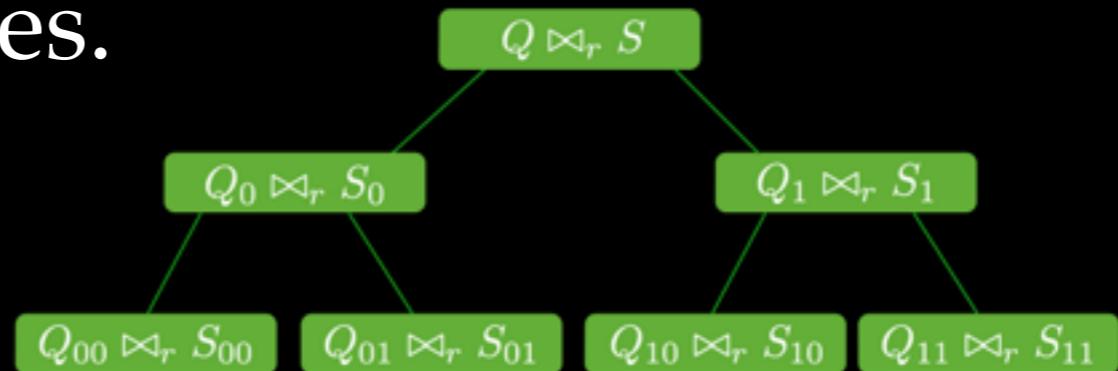
# Conclusions

- Recursive application of LSH is a promising approach to large-scale similarity joins with theoretical guarantees.



# Conclusions

- Recursive application of LSH is a promising approach to large-scale similarity joins with theoretical guarantees.



- Guarantees can include “total recall”, though with present constructions this requires a constant  $c > 3$ .

Get ready for the ride  
of your life.

TOTAL

# Some open questions

- Practicality of recursive approach
  - Many choices made to enable theoretical reasoning should be reconsidered.
  - How about parallelism?

# Some open questions

- Practicality of recursive approach
  - Many choices made to enable theoretical reasoning should be reconsidered.
  - How about parallelism?
- Better correlated LSH
  - Ideally matching the best performance possible without 100% recall guarantee.

# Thank you!

IT UNIVERSITY OF COPENHAGEN

DET FRIE FORSKNINGSRÅD



**European Research Council**

Established by the European Commission

**Supporting top researchers  
from anywhere in the world**

“There's nothing an agnostic can't do if he doesn't know whether he believes in anything or not.”

*—Monty Python*