Online Generation of Locality Sensitive Hash Signatures

BENJAMIN VAN DURME & ASHWIN LALL

human language technology
center of excellence

JOHNS HOPKINS
UNIVERSITY

Georgia Tech

Data Overload

- Our access to data is growing fast

Data Overload

- Our access to data is growing faster than our ability to process it

Data Overload

- Our access to data is growing faster than our ability to process it

- Complementary solutions:
  - Distributed environments (e.g., MapReduce)
  - Streaming / Randomized Algorithms

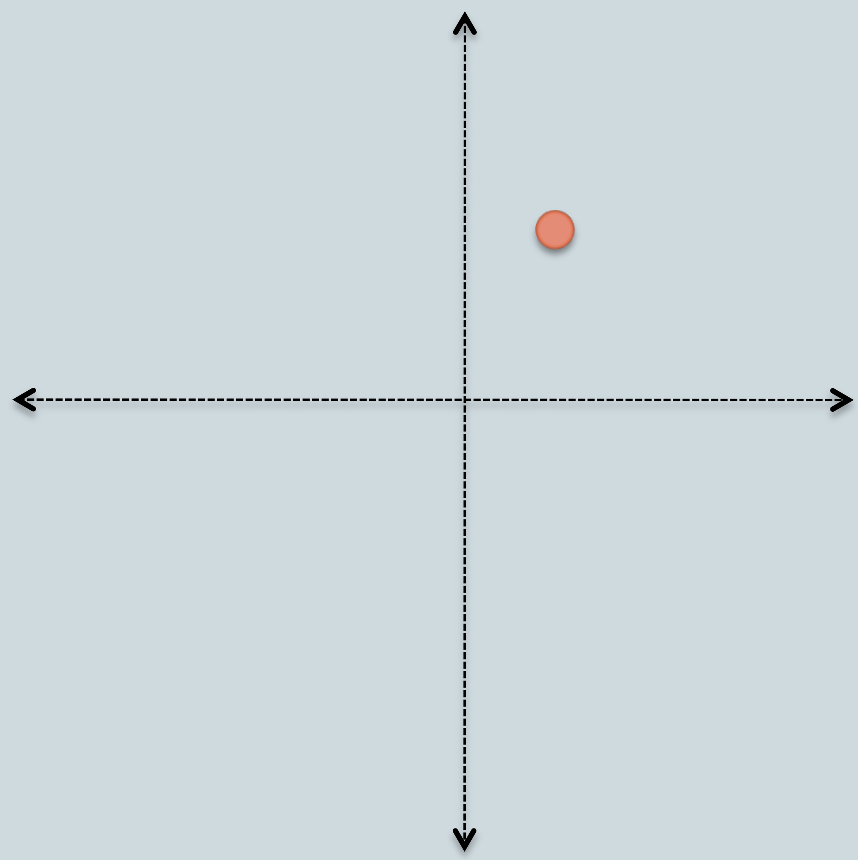
Locality Sensitive Hashing

- Goal: fast comparison between points in very high dimensional space

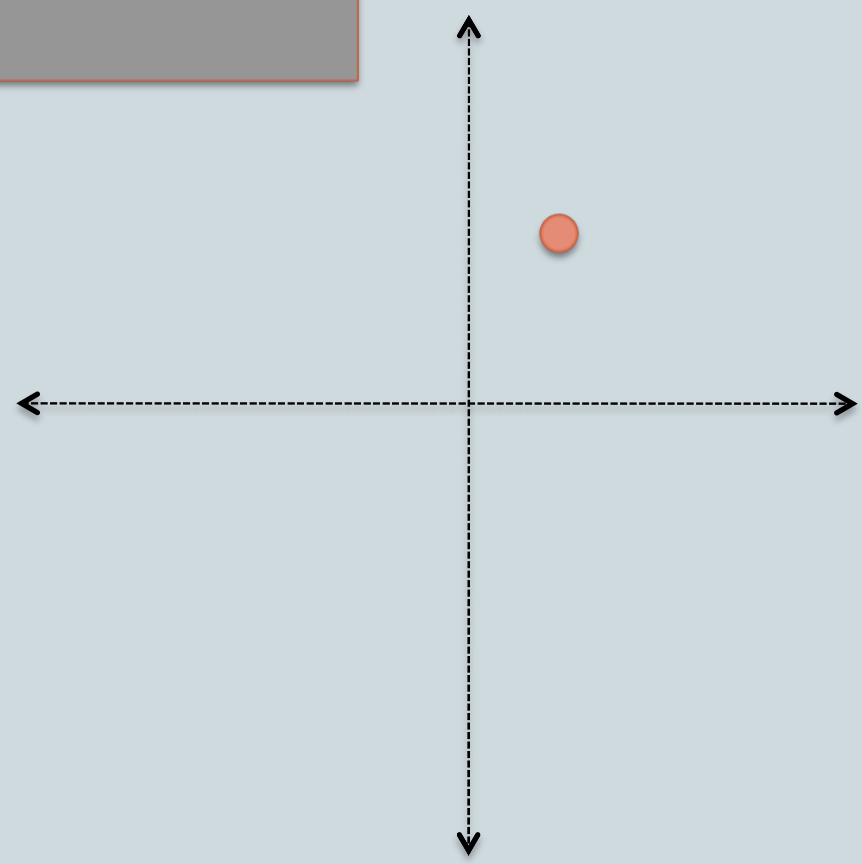
- Indyk & Motwani (‘98) => Charikar (’02)
  - Randomly project points to low dimensional *bit signatures* such that **cosine distance** is approximately preserved

- Example Applications in HLT
  - Noun clustering [Ravichandran *et al* ’05]
  - Topic Detection and Tracking (TDT) [Petrovic *et al* ‘10]

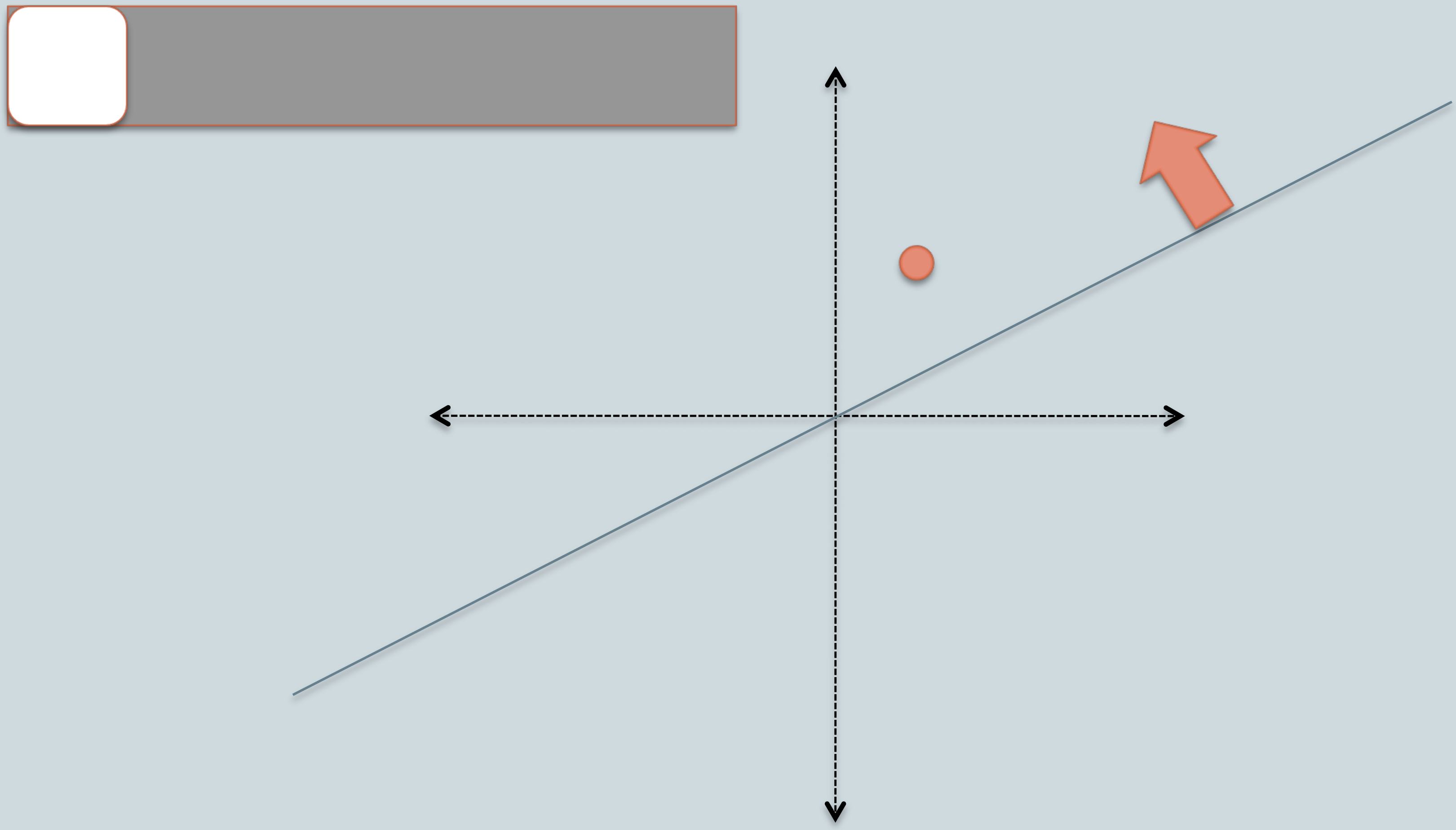
Locality Sensitive Hashing



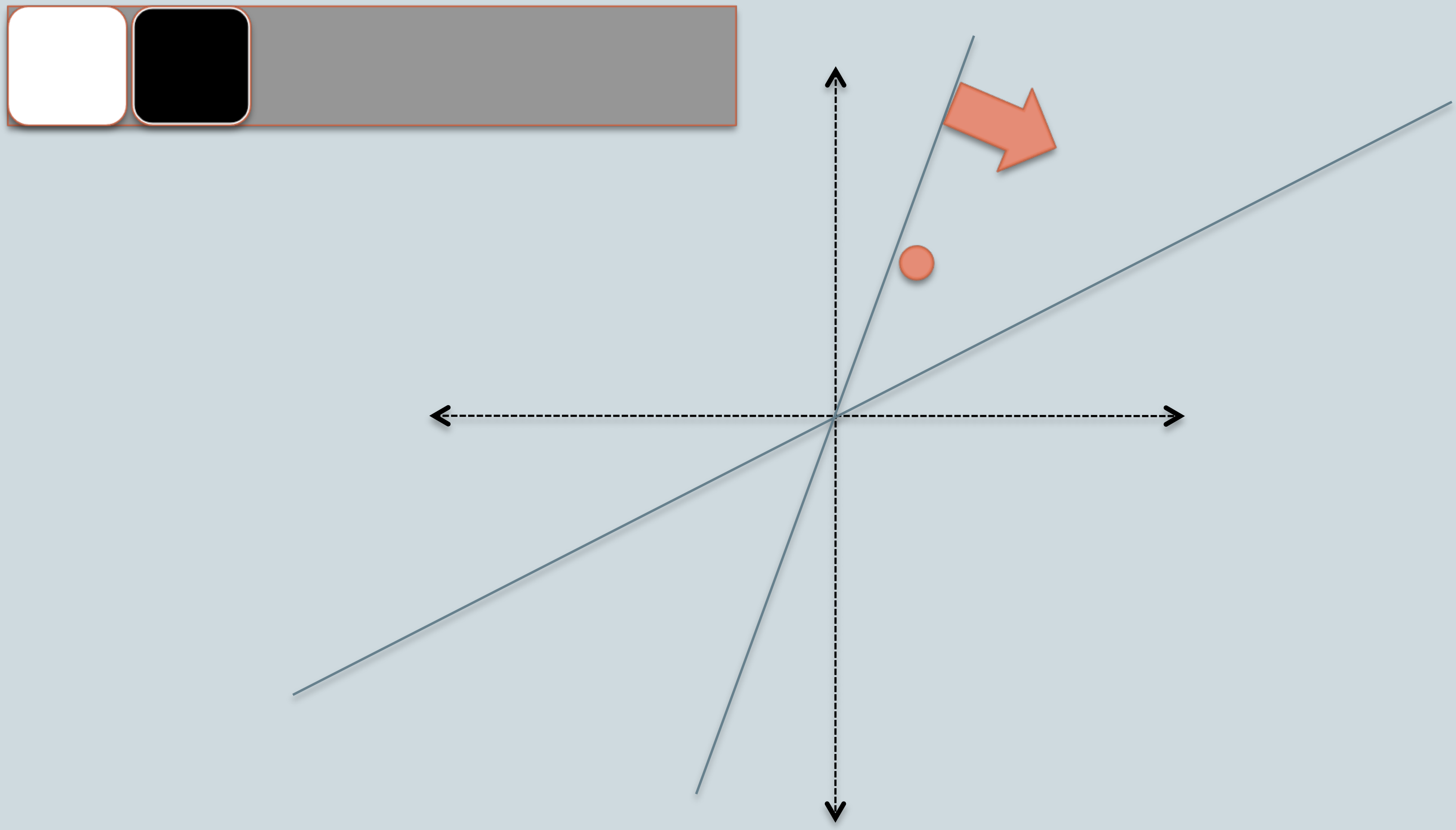
Locality Sensitive Hashing



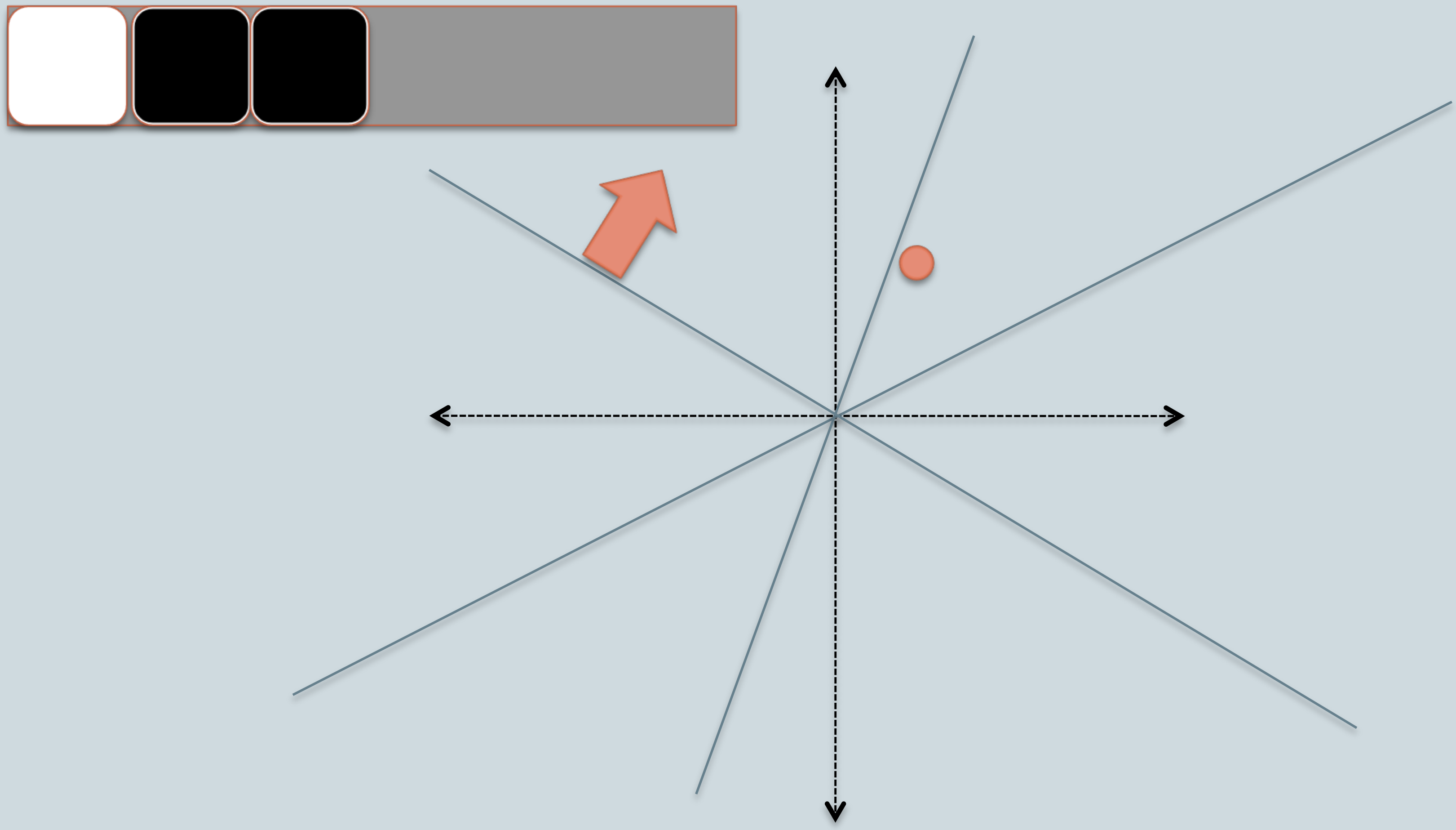
Locality Sensitive Hashing



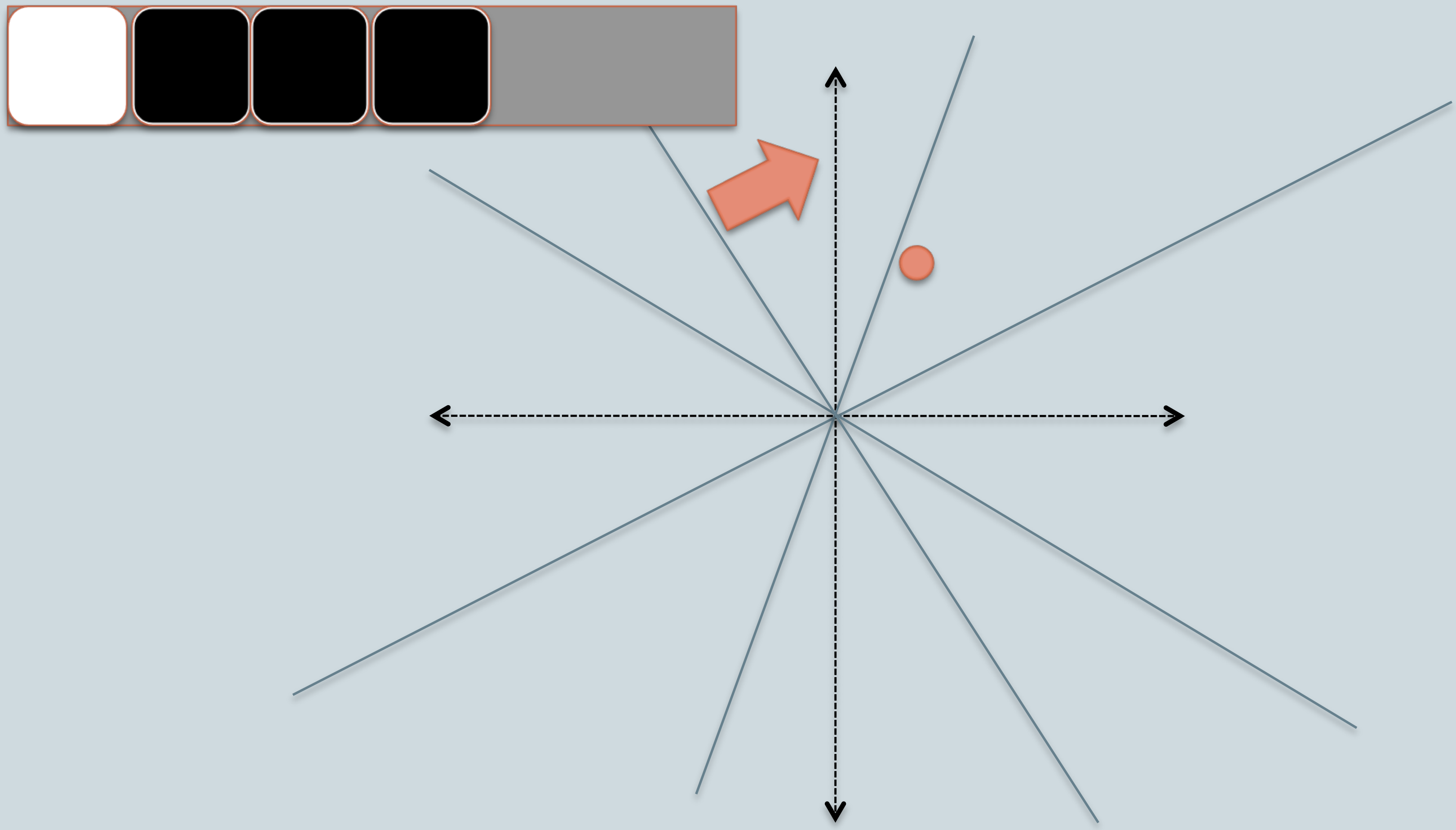
Locality Sensitive Hashing



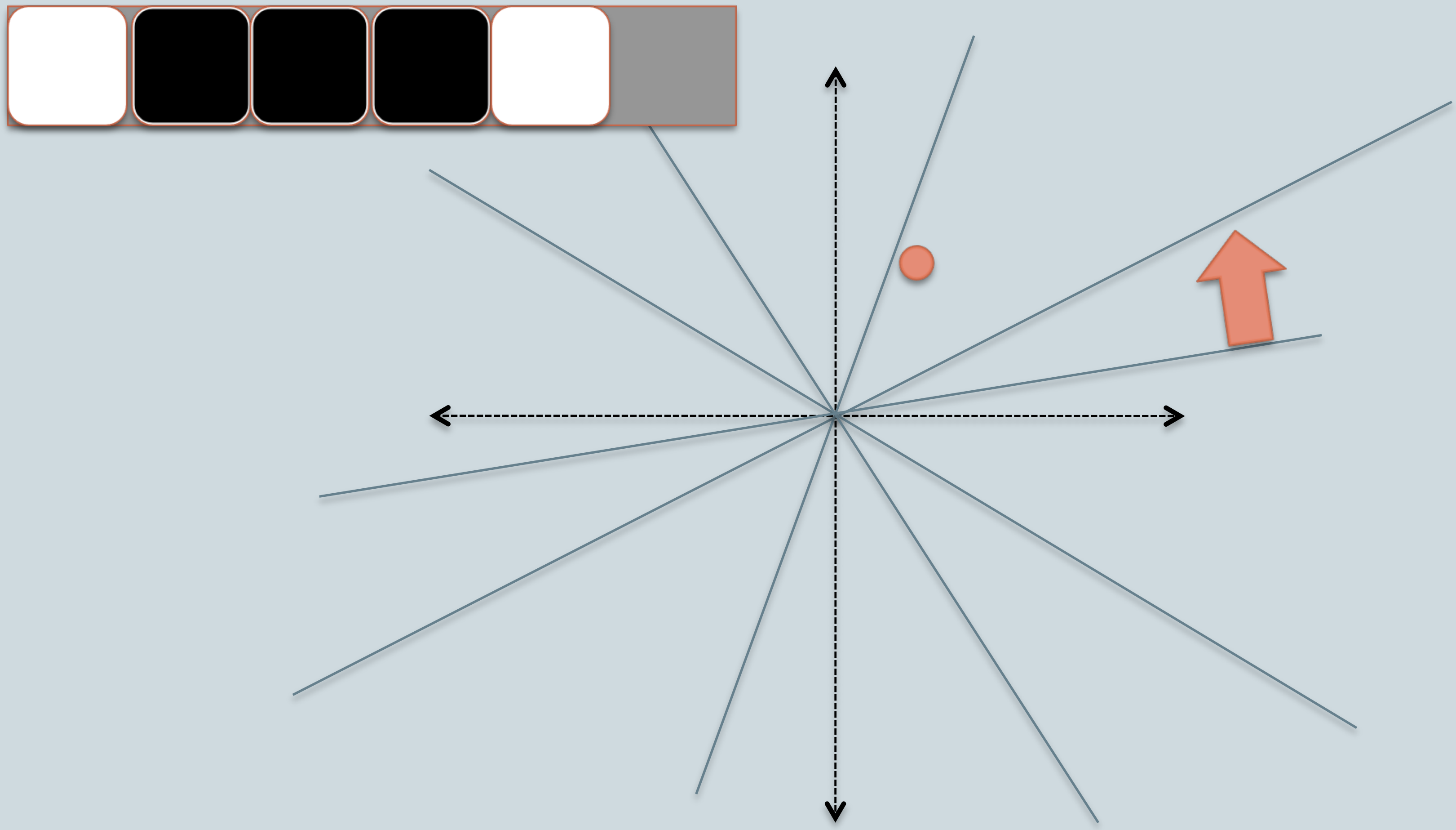
Locality Sensitive Hashing



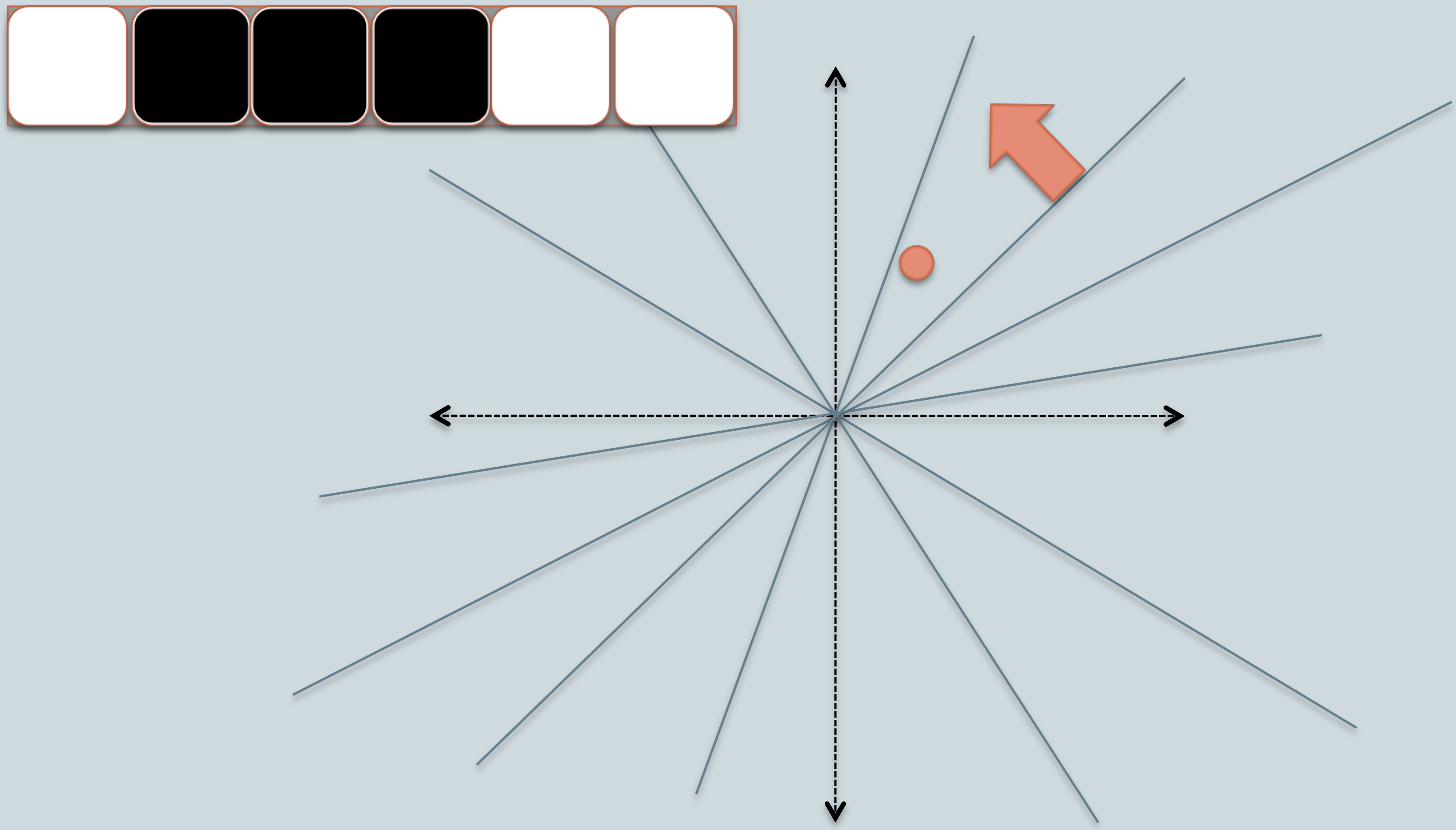
Locality Sensitive Hashing



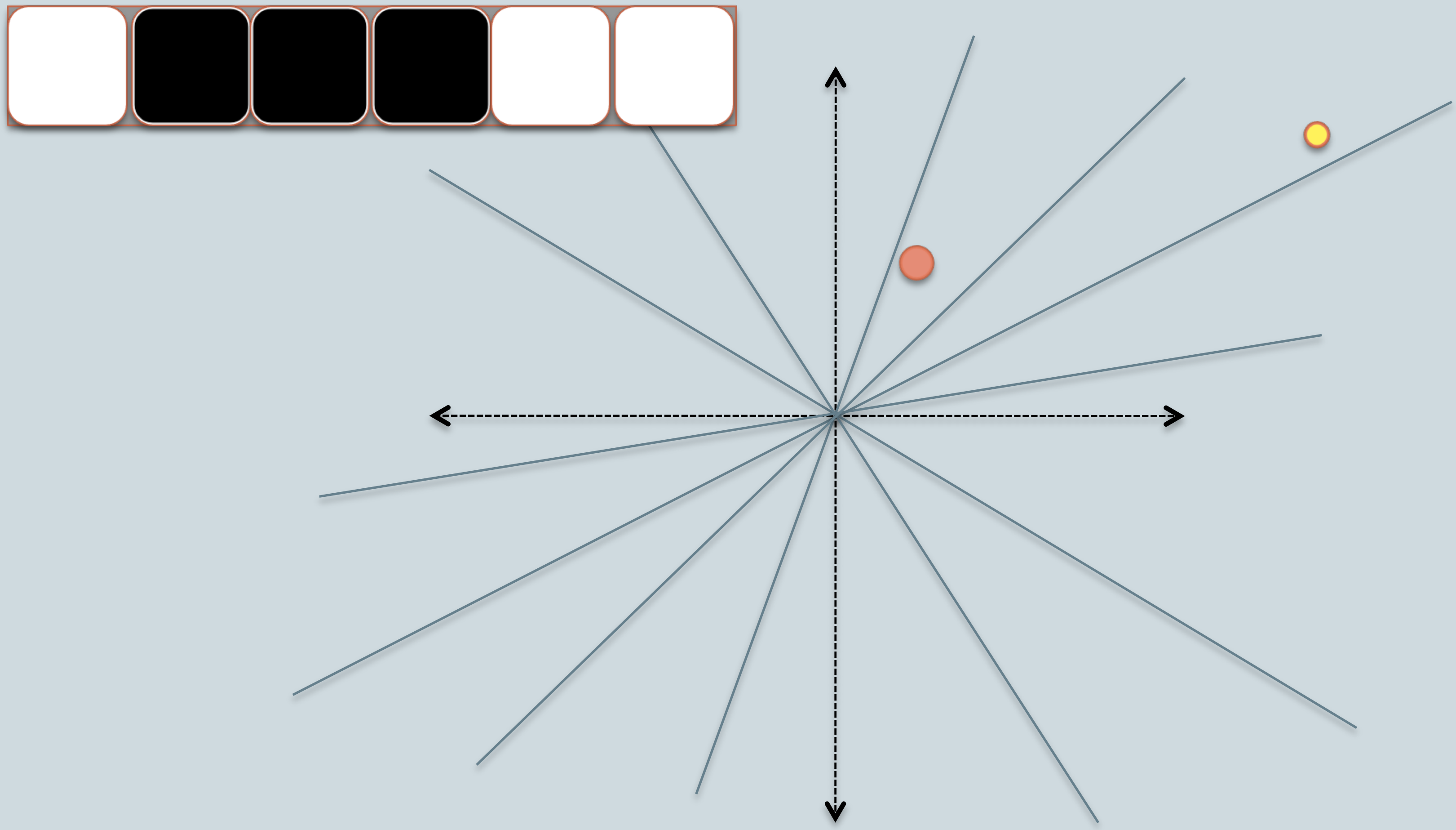
Locality Sensitive Hashing



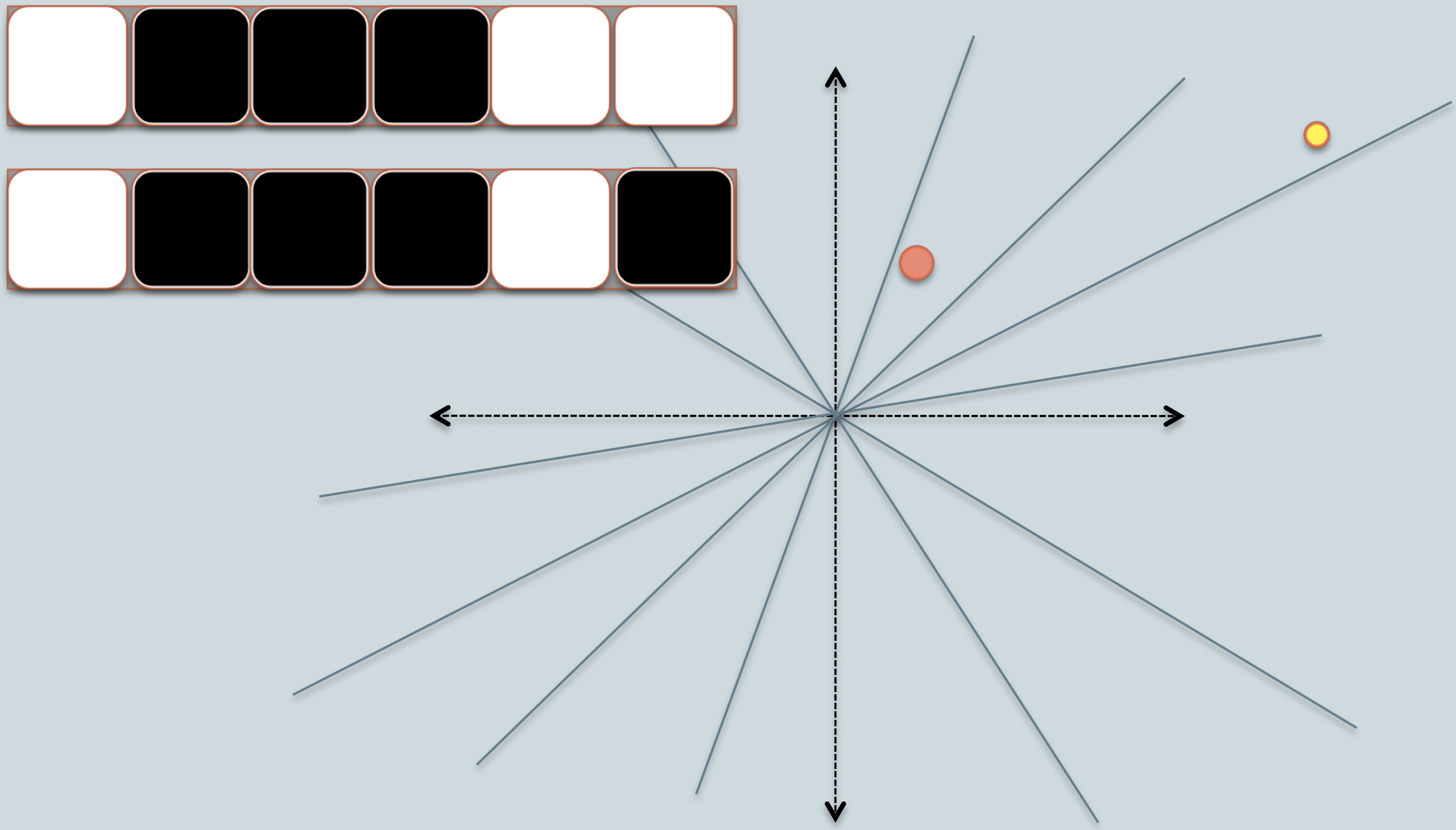
Locality Sensitive Hashing



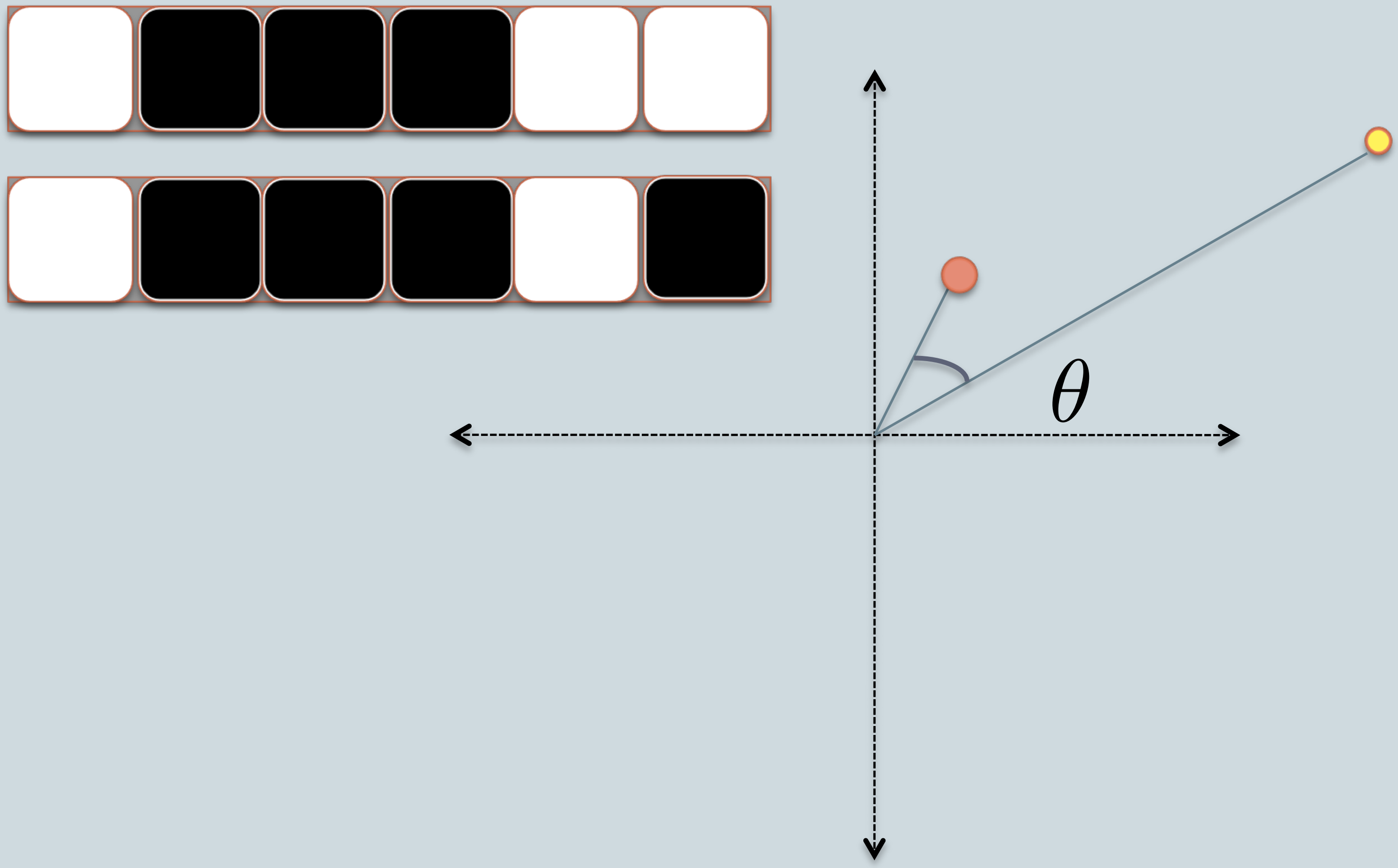
Locality Sensitive Hashing



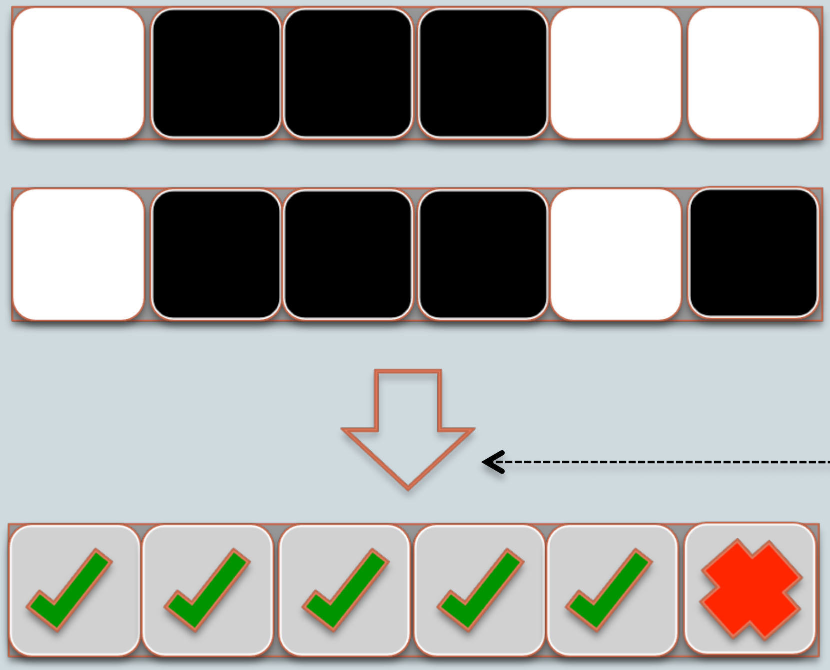
Locality Sensitive Hashing



Locality Sensitive Hashing

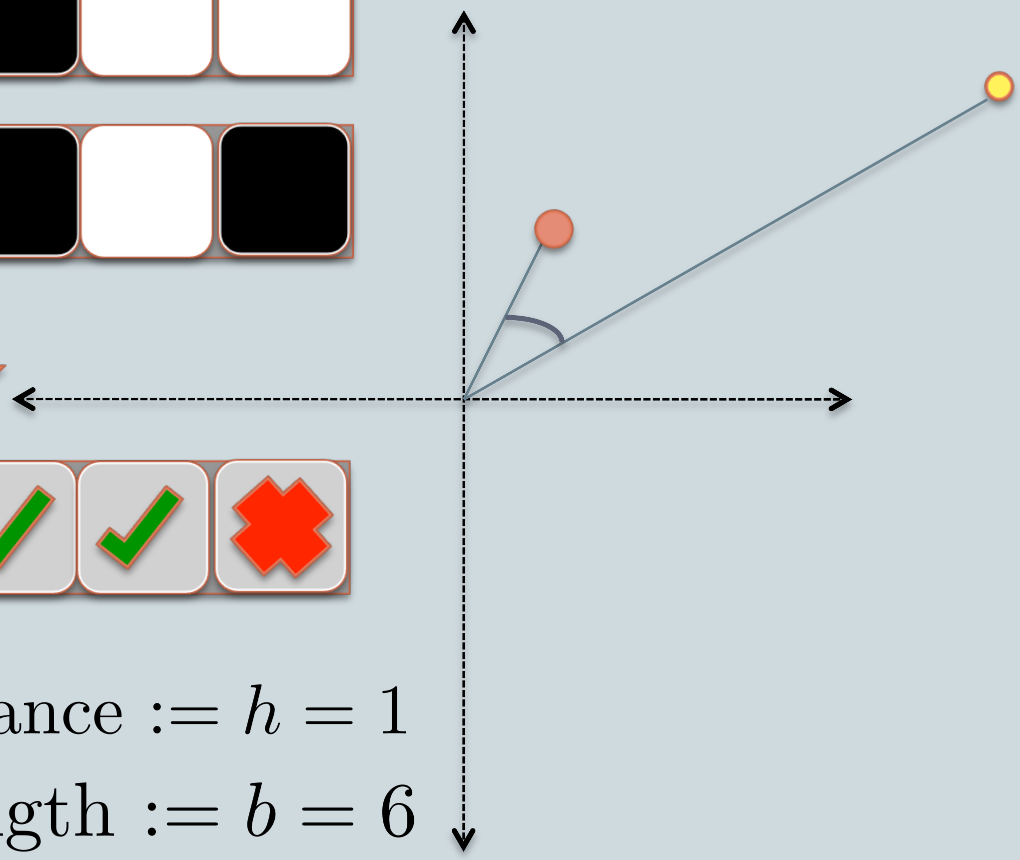


Locality Sensitive Hashing

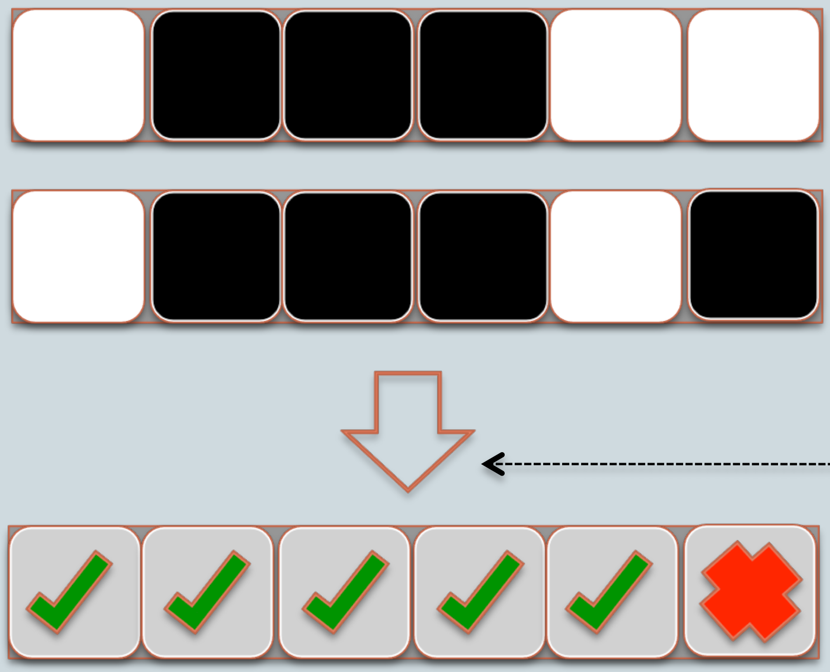


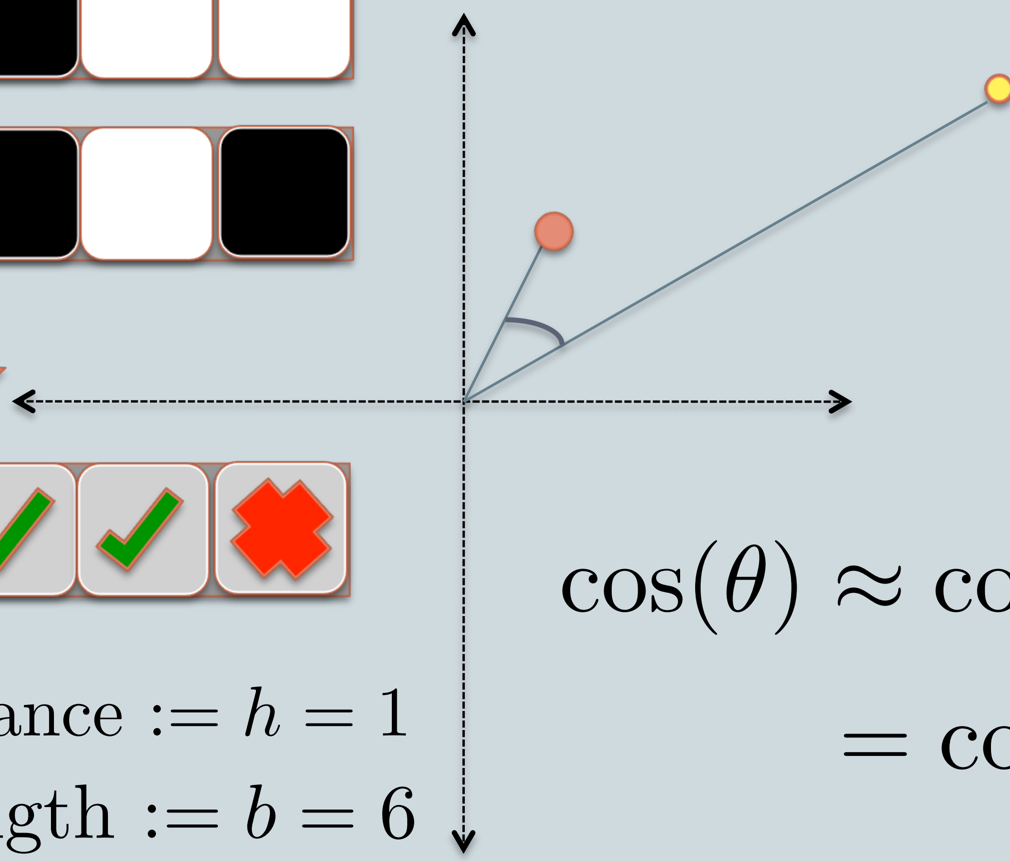
Hamming Distance $:= h = 1$

Signature Length $:= b = 6$



Locality Sensitive Hashing



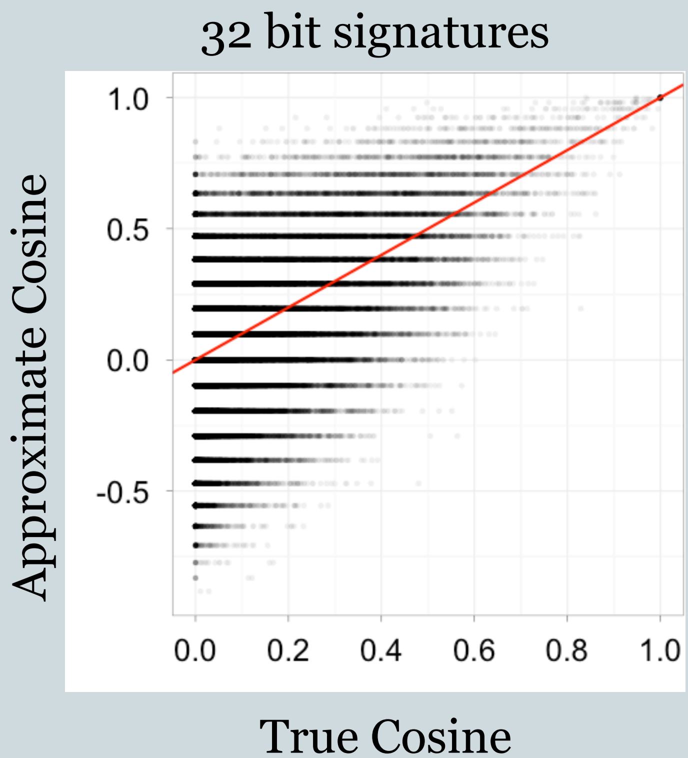


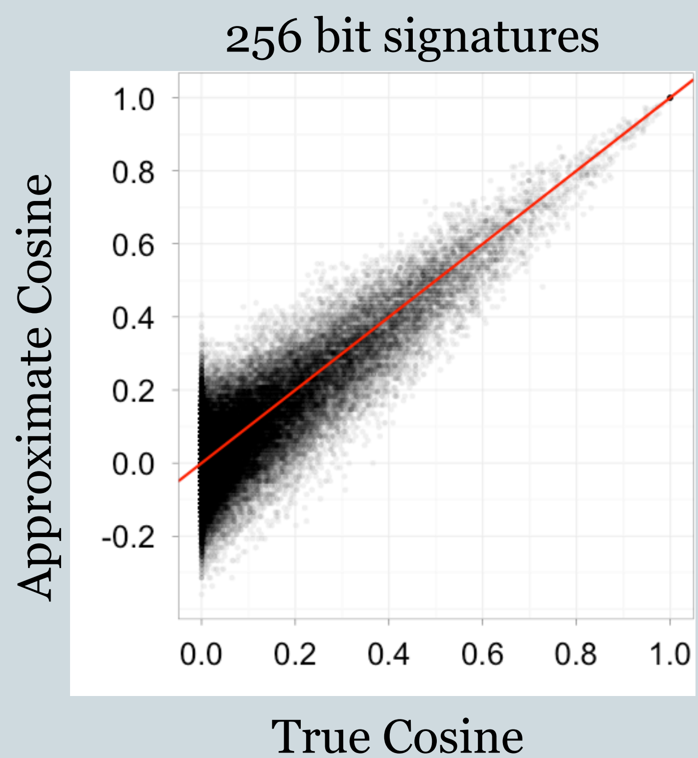
$$\cos(\theta) \approx \cos\left(\frac{h}{b}\pi\right)$$
$$= \cos\left(\frac{1}{6}\pi\right)$$

Hamming Distance $:= h = 1$

Signature Length $:= b = 6$

Accuracy as function of bit length





Cheap

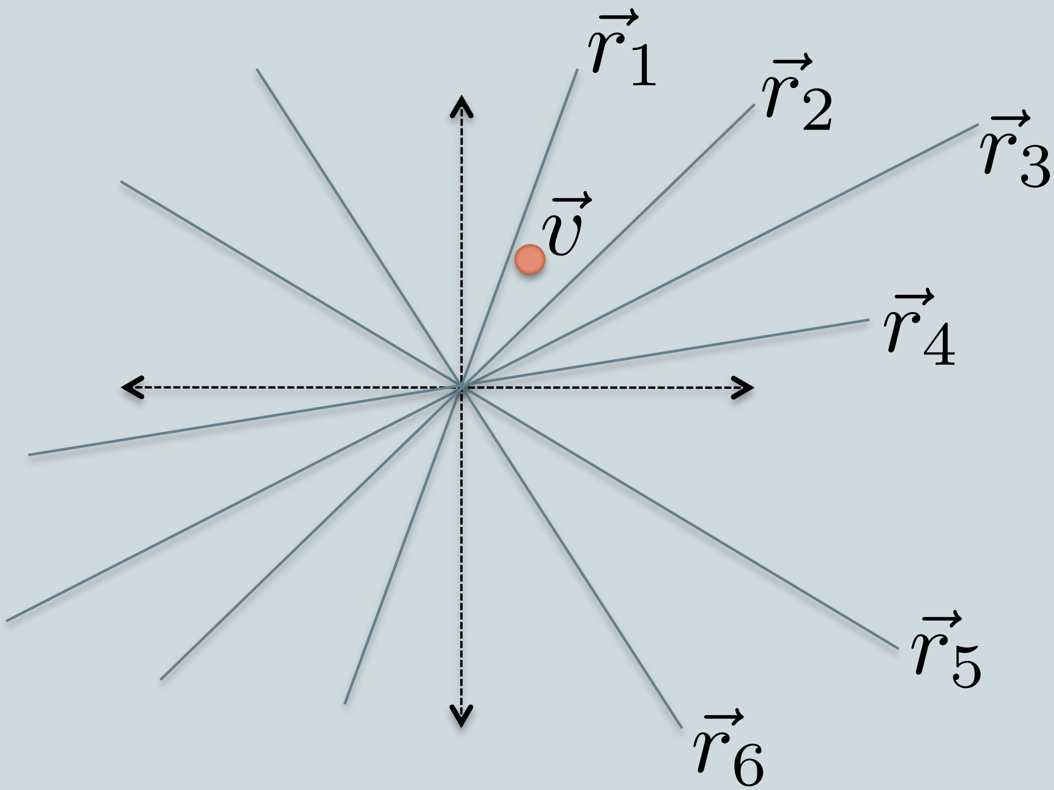
Accurate

Novel Points Here

1. Online hash function

2. Pooling trick

Online Hash Function



$$\vec{v} \in \mathbb{R}^d$$

$$\vec{r}_i \sim N(0,1)^d$$

Online Hash Function

$$\vec{v} \in \mathbb{R}^d$$

$$\vec{r}_i \sim N(0,1)^d$$

$$h_i(\vec{v}) = \begin{cases} 1 & \text{if } \vec{v} \cdot \vec{r}_i \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Online Hash Function

$$\text{if } \vec{v} = \Sigma_j \vec{v}_j$$
$$\text{then } \vec{v} \cdot \vec{r}_i = \Sigma_j \vec{v}_j \cdot \vec{r}_i$$

Break into local products

Online Hash Function

$$h_i(\vec{v}) = \begin{cases} 1 & \text{if } \vec{v} \cdot \vec{r}_i \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Offline

$$\text{if } \vec{v} = \Sigma_j \vec{v}_j$$
$$\text{then } \vec{v} \cdot \vec{r}_i = \Sigma_j \vec{v}_j \cdot \vec{r}_i$$

Online

$$h_{it}(\vec{v}) = \begin{cases} 1 & \text{if } \Sigma_j^t \vec{v}_j \cdot \vec{r}_i \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

The Pooling Trick

$$\vec{r}_i \sim N(0,1)^d$$

$$\begin{pmatrix} \vec{r}_1 \\ \dots \\ \vec{r}_b \end{pmatrix} = \begin{pmatrix} N(0,1) & \dots & N(0,1) \\ \dots & \dots & \dots \\ N(0,1) & \dots & N(0,1) \end{pmatrix}$$

The Pooling Trick

The random projection matrix can easily require gigabytes of memory

$$\begin{pmatrix} \vec{r}_1 \\ \dots \\ \vec{r}_b \end{pmatrix} = \begin{pmatrix} N(0,1) & \dots & N(0,1) \\ \dots & \dots & \dots \\ N(0,1) & \dots & N(0,1) \end{pmatrix}$$

The Pooling Trick

$$\vec{p} \sim N(0,1)^m$$

$$\vec{p} = \begin{pmatrix} N(0,1) & \ldots & N(0,1) \end{pmatrix}$$

$$\text{Define } \vec{r}_i[j] = \vec{p}[\text{hash}(i,j) \text{ MOD } m]$$

$$m \ll b \times d$$

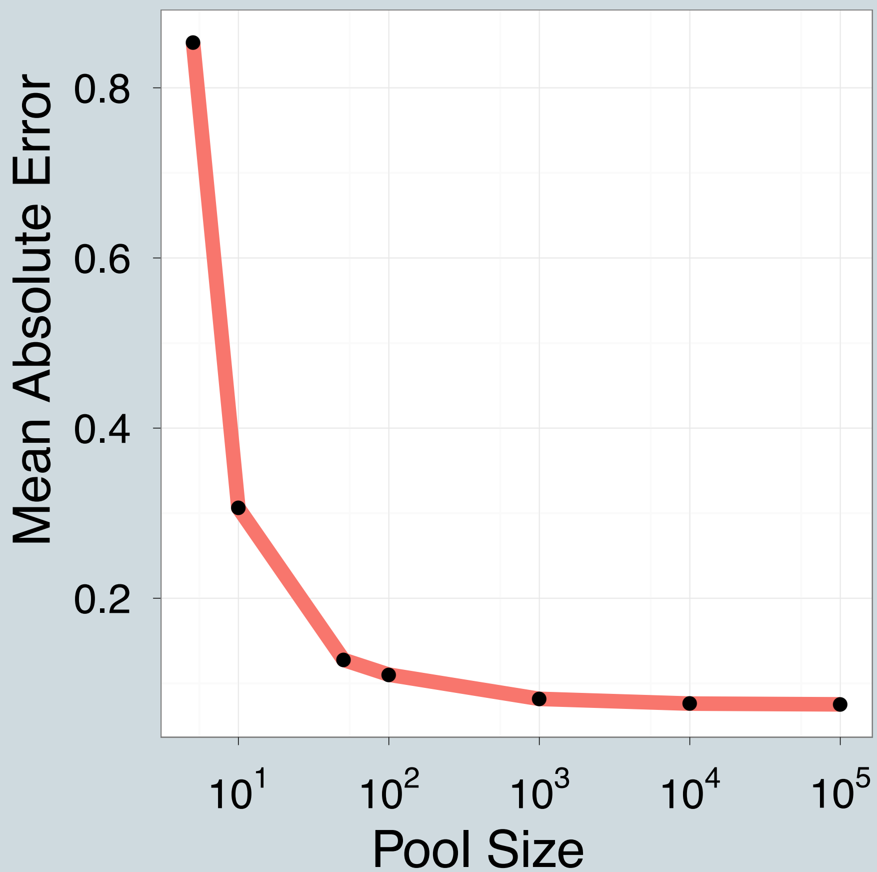
The Pooling Trick

$$\text{Define}\; \vec{r}_i[j] = \vec{p}\,[\text{hash}(i, j)\;\text{MOD}\; m]$$

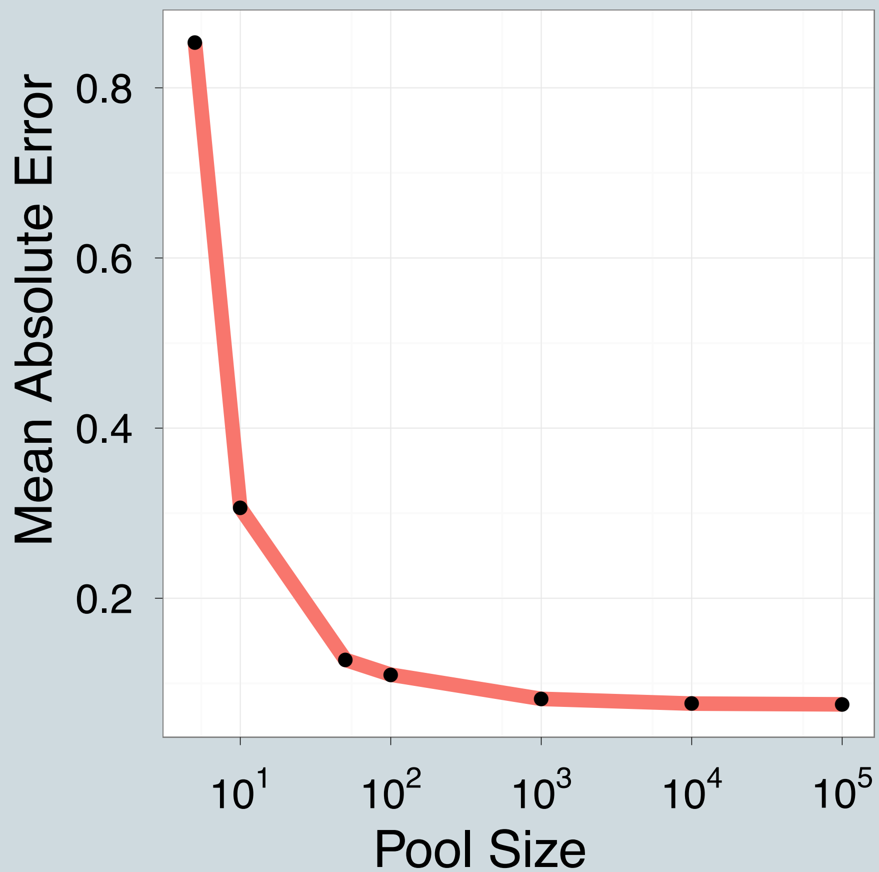
$$\begin{pmatrix} \ldots & \ldots & \ldots \\ \ldots & (i, j) & \ldots \\ \ldots & \ldots & \ldots \end{pmatrix}$$

$$\begin{pmatrix} N(0,1) & \ldots & N(0,1) \end{pmatrix}$$

The Pooling Trick



The Pooling Trick



That is 400 **kilobytes**

Example

London

Milan$_{.97}$, **Madrid**$_{.96}$, **Stockholm**$_{.96}$, **Manila**$_{.95}$, **Moscow**$_{.95}$
ASHER$_0$, Champaign$_0$, MANS$_0$, NOBLE$_0$, come$_0$
Prague$_1$, Vienna$_1$, suburban$_1$, synchronism$_1$, Copenhagen$_2$
Frankfurt$_4$, Prague$_4$, Taszar$_5$, Brussels$_6$, Copenhagen$_6$
Prague$_{12}$, Stockholm$_{12}$, Frankfurt$_{14}$, Madrid$_{14}$, Manila$_{14}$
Stockholm$_{20}$, Milan$_{22}$, Madrid$_{24}$, Taipei$_{24}$, Frankfurt$_{25}$

Accurate

Closest based on true cosine

London

Milan$_{.97}$, **Madrid**$_{.96}$, **Stockholm**$_{.96}$, **Manila**$_{.95}$, **Moscow**$_{.95}$
ASHER$_0$, Champaign$_0$, MANS$_0$, NOBLE$_0$, come$_0$
Prague$_1$, Vienna$_1$, suburban$_1$, synchronism$_1$, Copenhagen$_2$
Frankfurt$_4$, Prague$_4$, Taszar$_5$, Brussels$_6$, Copenhagen$_6$
Prague$_{12}$, Stockholm$_{12}$, Frankfurt$_{14}$, Madrid$_{14}$, Manila$_{14}$
Stockholm$_{20}$, Milan$_{22}$, Madrid$_{24}$, Taipei$_{24}$, Frankfurt$_{25}$

London

Milan$_{.97}$, **Madrid**$_{.96}$, **Stockholm**$_{.96}$, **Manila**$_{.95}$, **Moscow**$_{.95}$
ASHER$_0$, Champaign$_0$, MANS$_0$, NOBLE$_0$, come$_0$
Prague$_1$, Vienna$_1$, suburban$_1$, synchronism$_1$, Copenhagen$_2$
Frankfurt$_4$, Prague$_4$, Taszar$_5$, Brussels$_6$, Copenhagen$_6$
Prague$_{12}$, Stockholm$_{12}$, Frankfurt$_{14}$, Madrid$_{14}$, Manila$_{14}$
Stockholm$_{20}$, Milan$_{22}$, Madrid$_{24}$, Taipei$_{24}$, Frankfurt$_{25}$

Closest based on 32 bit sig.'s

Cheap

London

Milan$_{.97}$, **Madrid**$_{.96}$, **Stockholm**$_{.96}$, **Manila**$_{.95}$, **Moscow**$_{.95}$
ASHER$_0$, Champaign$_0$, MANS$_0$, NOBLE$_0$, come$_0$
Prague$_1$, Vienna$_1$, suburban$_1$, synchronism$_1$, Copenhagen$_2$
Frankfurt$_4$, Prague$_4$, Taszar$_5$, Brussels$_6$, Copenhagen$_6$
Prague$_{12}$, Stockholm$_{12}$, Frankfurt$_{14}$, Madrid$_{14}$, Manila$_{14}$
Stockholm$_{20}$, Milan$_{22}$, Madrid$_{24}$, Taipei$_{24}$, Frankfurt$_{25}$

Closest based on 256 bit sig.'s

Cheap-ish

No Questions.

See Poster.

Also: these algorithms distribute to the cloud.

human language technology
center of excellence

JOHNS HOPKINS
UNIVERSITY

Georgia Tech