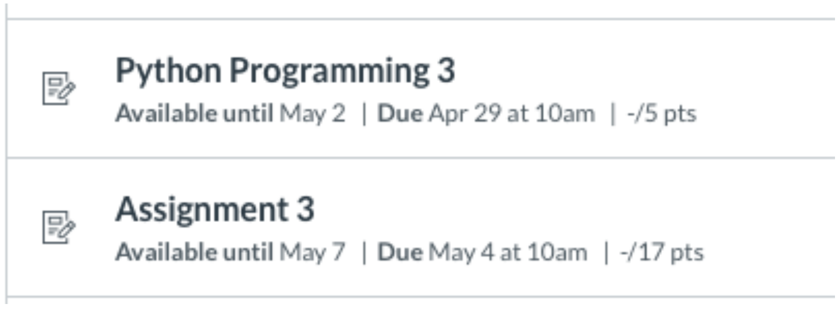
DIT 873 & DAT346 Techniques for Large-Scale Data

Lecture 7

Agenda for today:

- Bloom filters
- Suffix trees
- Birgit & Shirin: Information about project courses
- Sketching

Upcoming Deadlines



Python Programming 3
Available until May 2 | Due Apr 29 at 10am | -/5 pts

Assignment 3
Available until May 7 | Due May 4 at 10am | -/17 pts

Next lecture: April 29

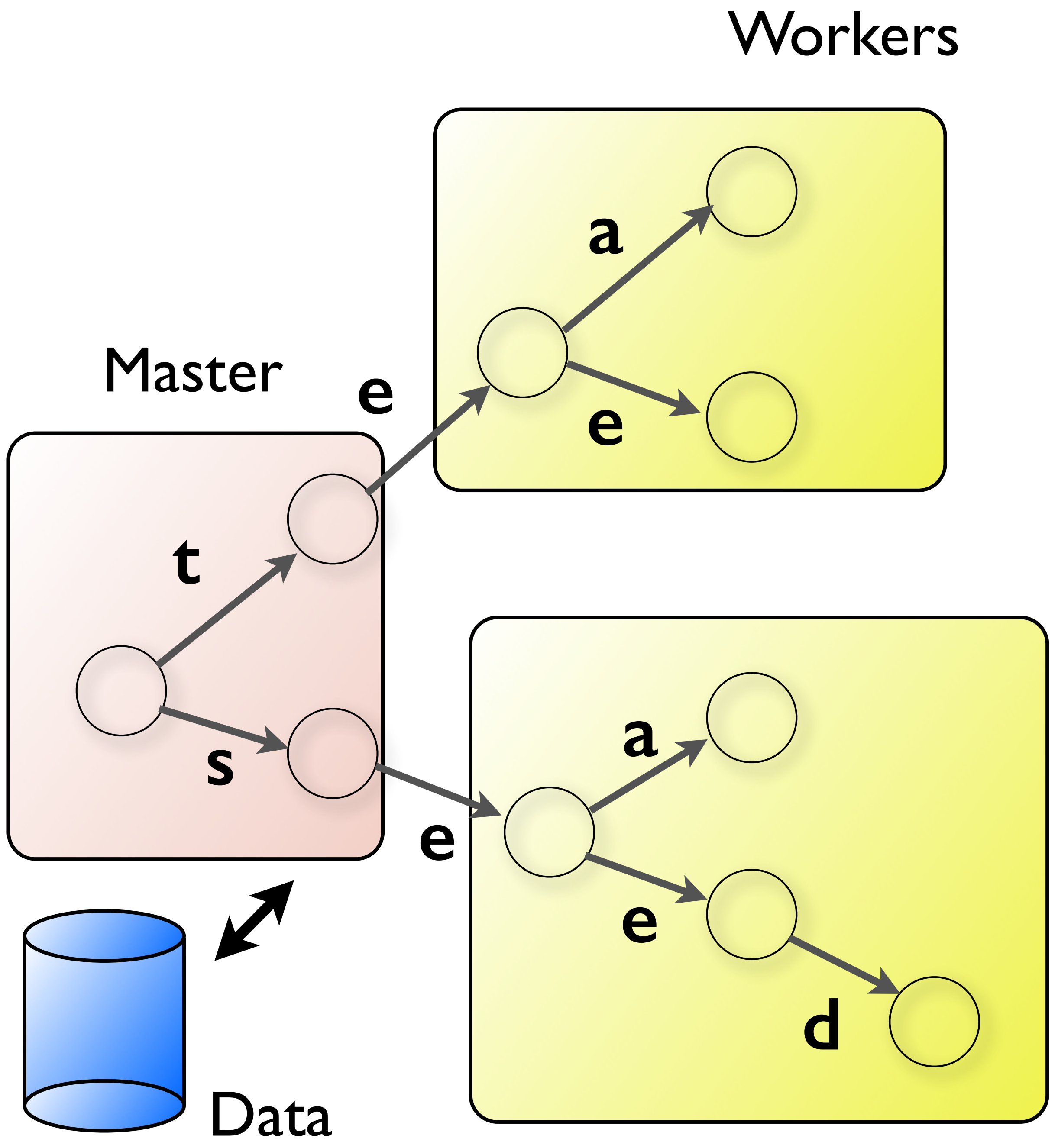
Background material for 2nd part of the course
https://chalmers.instructure.com/courses/9360/pages/advanced-databases-background

Recap

Application: Count Frequent Items

- Frequent items:
  - Heavy hitters
  - Top search queries
  - Most frequently requested items in a database
- Very important in Genomics/DNA sequencing

Trie: applications & parallelization



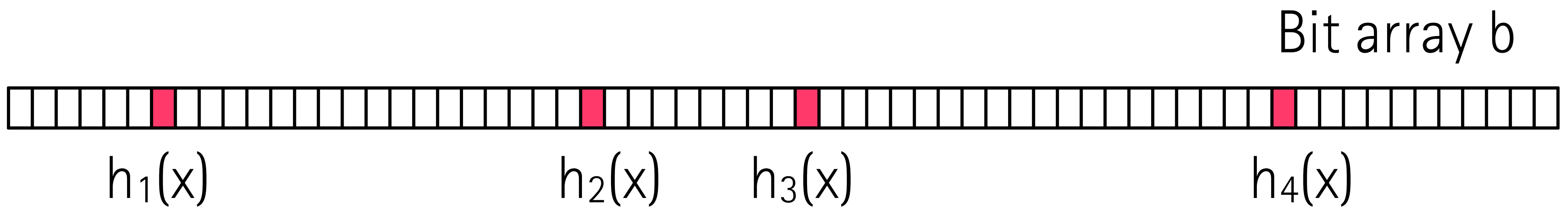
- Duplicates: Second visit to leaf
- Counting: Integer variables in leaf

Trie (Retrieval Tree)

Frequent items: problem sizes

- Billions of items (~ coverage x genome size)
  E.g. 264 Billions for an 80x Human sequencing data set
- Billions of frequent items (~genome size)
  > 3 Billion for Human
- Billions of infrequent items
  100s of Billions

Bloom filter



- Given k hash functions $h_1, h_2, \ldots, h_k$
- Insert: for item x set bits $h_1(x), h_2(x), \ldots, h_k(x)$ in a bit array b
- Query: Check whether bits $h_1(x), h_2(x), \ldots, h_k(x)$ are set in b
- No false negatives

Bloom filter example

	Text	Python Dict	Trie	Bloom Filter p = 0.01	Bloom filter p = 0.001	Bloom filter p = 0.0001
10 000	146 kb	2.5 MB	1.17 MB	12 KB	18 kB	24 kB
100 000	1.43 MB	19.2 MB	~11.44 MB	117 KB	175 kB	243 kB
1M	14.4 MB	179.9 MB	~112 MB	1.14 MB	1.71 MB	2.29 MB

Assuming 15 unique characters per string. E.g URL

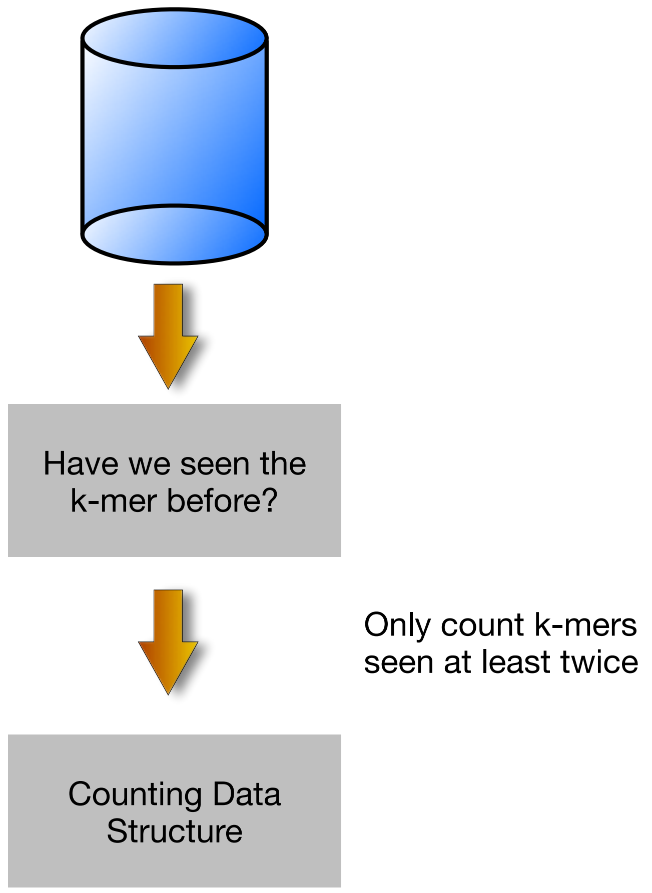
http://hur.st/bloomfilter

Continuing …

Counting with Bloom Filters?

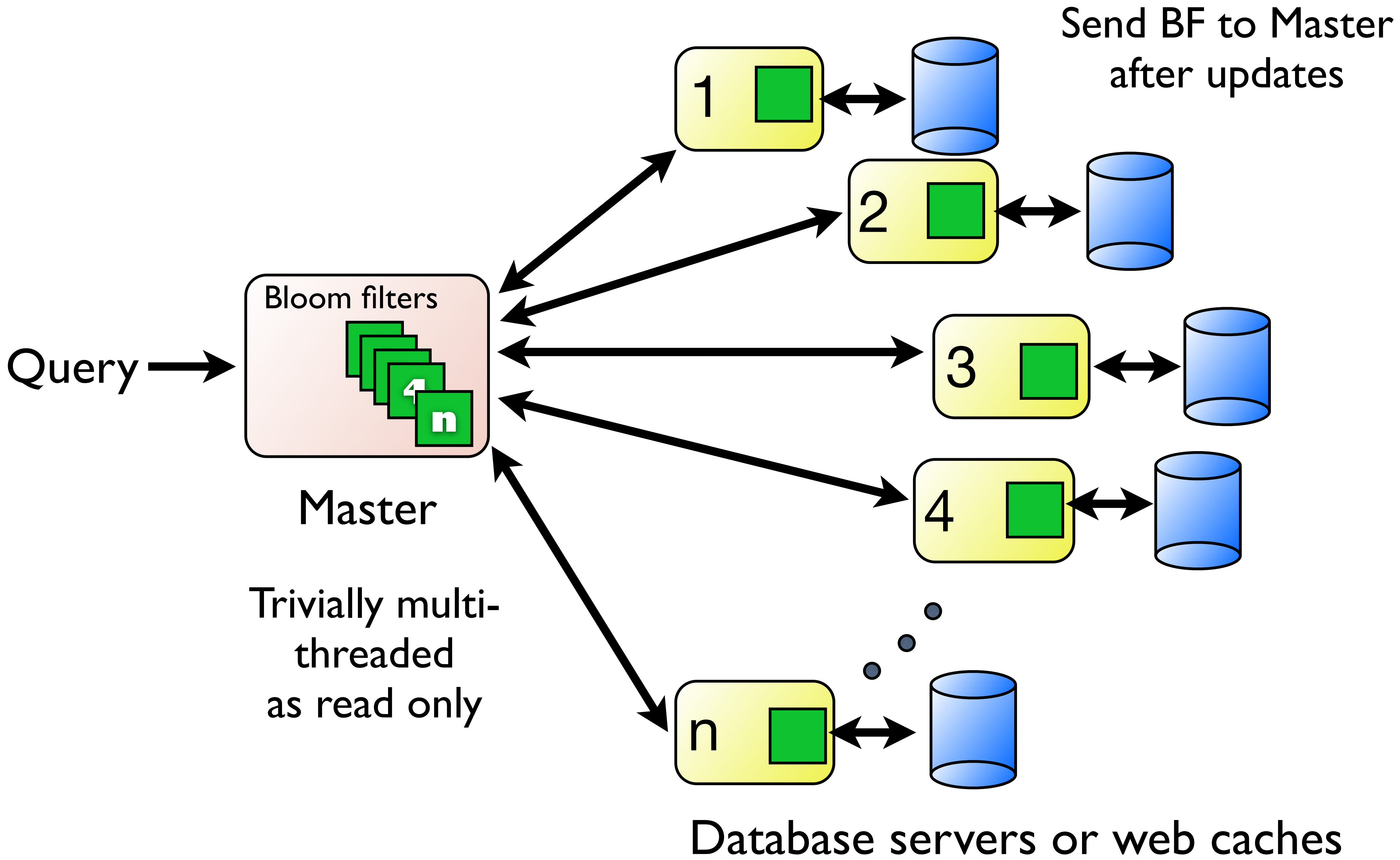
Counting with Bloom Filters

```
from pybloom import BloomFilter
f = BloomFilter(capacity=len(X),error_rate=p)
from collections import defaultdict
d = defaultdict(int)

for x in X:
    if x in f:
        d[x] += 1
    else:
        f.add(x)

for x in d.keys():
    if (d[x] + 1) > tau:
        print x, d[x]
```

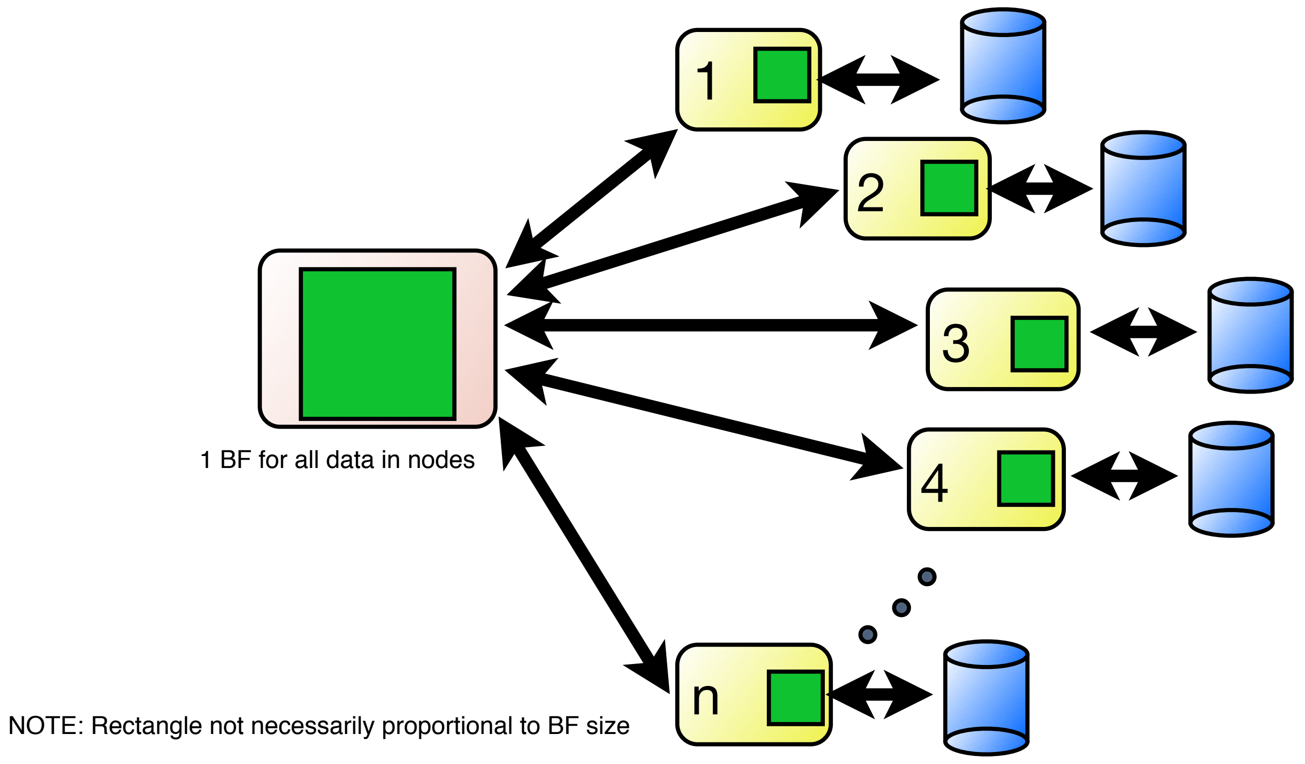

Have we seen the k-mer before?
Only count k-mers seen at least twice
Counting Data Structure

See https://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/io/BloomMapFile.Reader.html
for the same idea for files

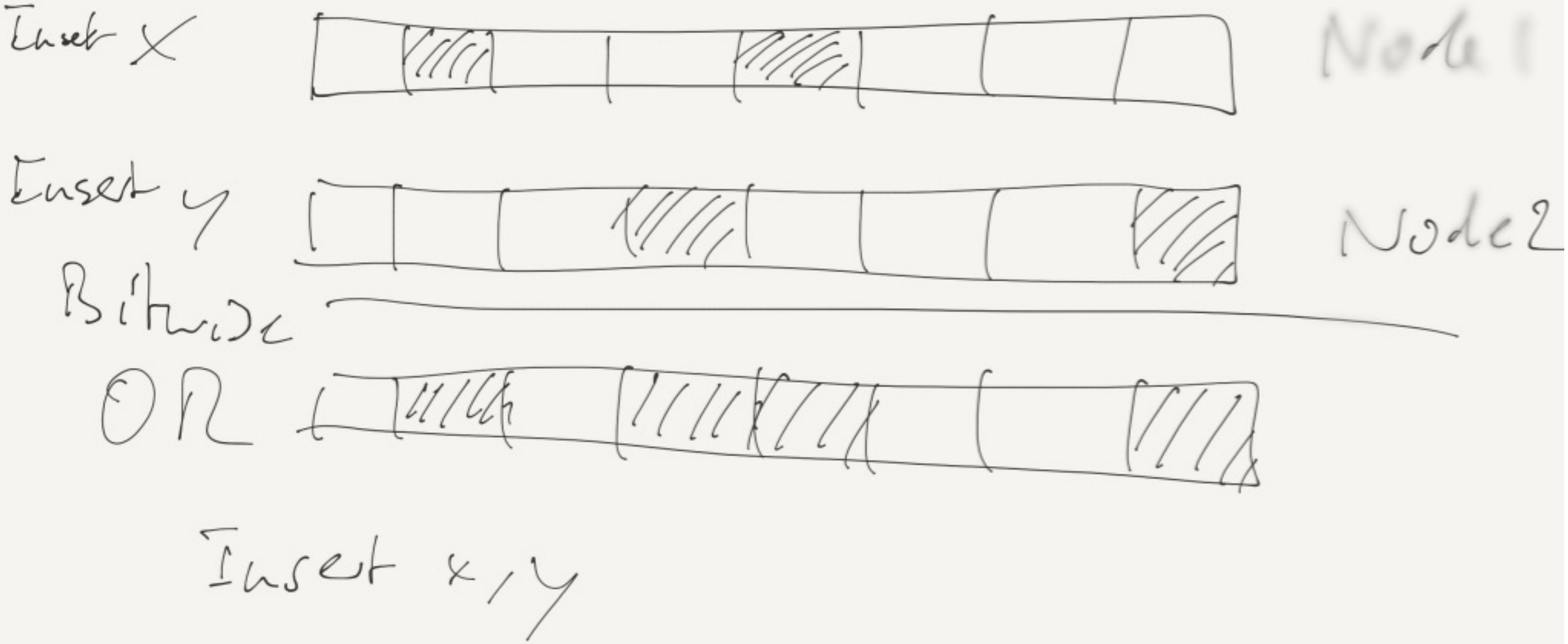
Distributed Data Store

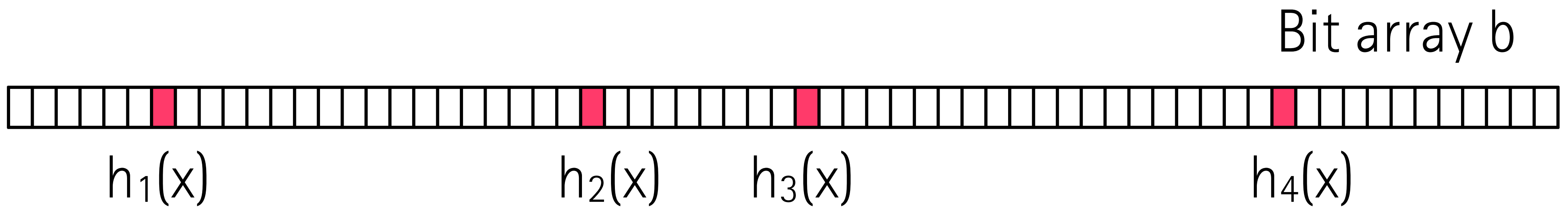


Can the creation of Bloom filters be efficiently distributed; e.g. with Map Reduce?


1
2
3
4
n
1 BF for all data in nodes
NOTE: Rectangle not necessarily proportional to BF size

A	B	C	D	E
True	False			

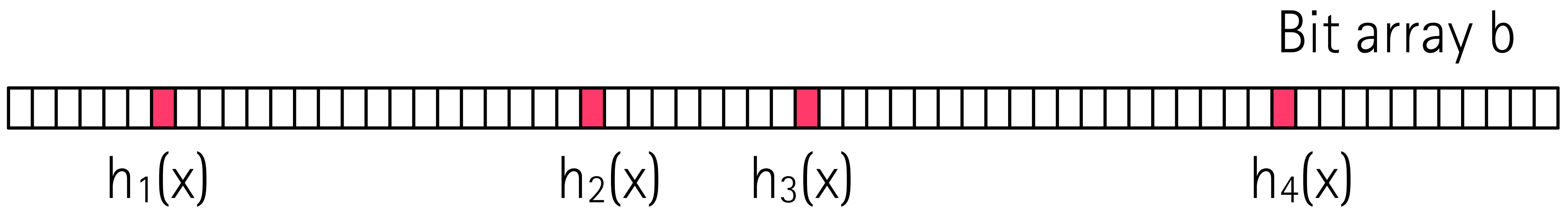

Insert X
Node1
Insert y
Node2
Bitwise OR
Insert x, y



For sufficiently large b, accessing each bit will likely cause a cache miss. The expected number of cache misses for insert and query operations, when the item is not in the Bloom filter is the same.

A	B	C	D	E
True	**False**			

Bloom filter: Cache Behaviour



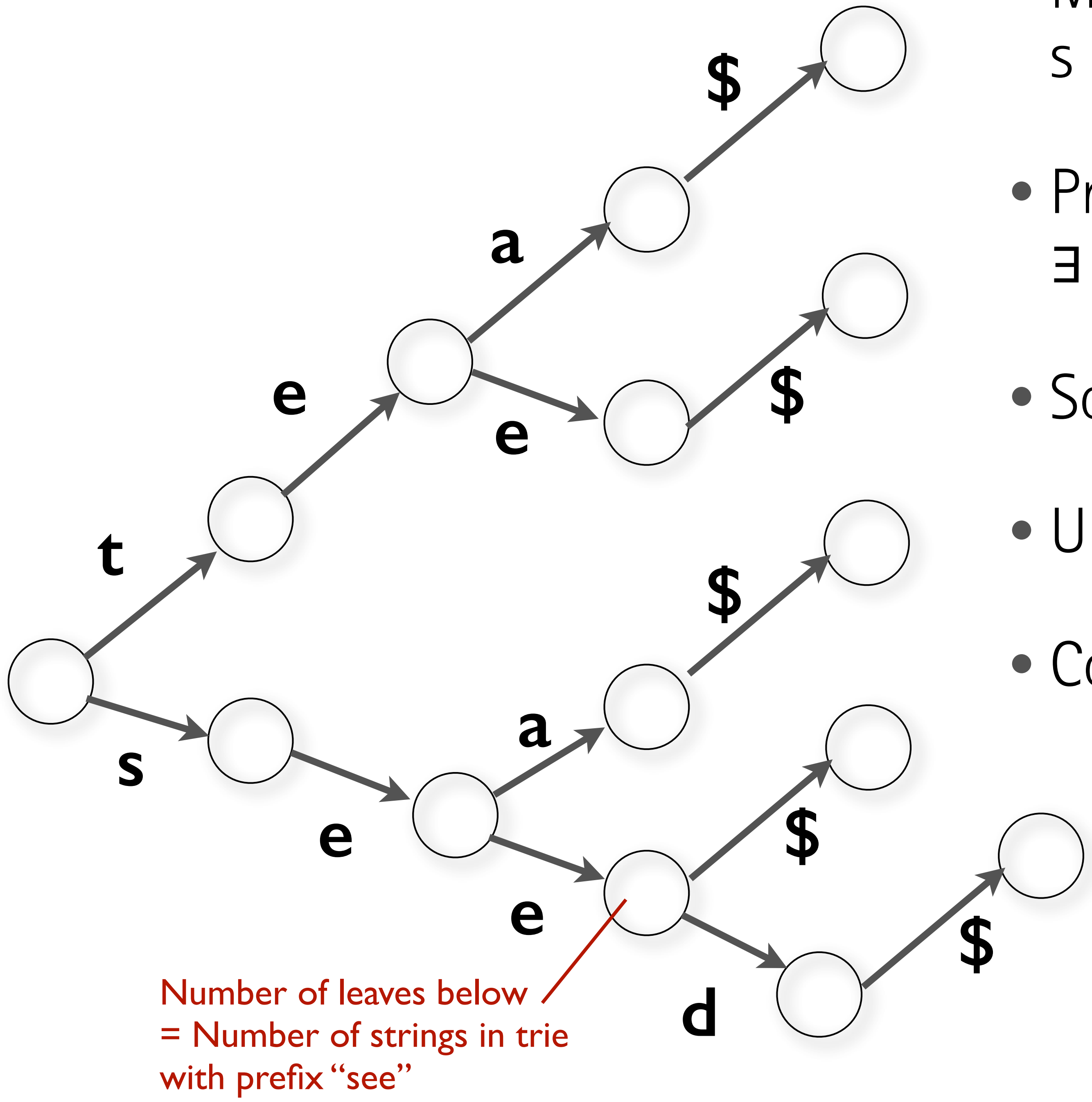
- Insert:
  - Worst-case: n cache misses (average close) per Insert
- Query:
  - item present / false positive: like insert
  - item not-present: average much lower than worst-case

Suffix trees

Data analysis on large texts

- Text T:
  - Natural language
  - Error or event logs (error codes/event types = alphabet)
  - Biological sequences, any other sequence of discrete observations
- Questions:
  - Is s a substring of T?
  - How often appears s in T?

Recall: Tries

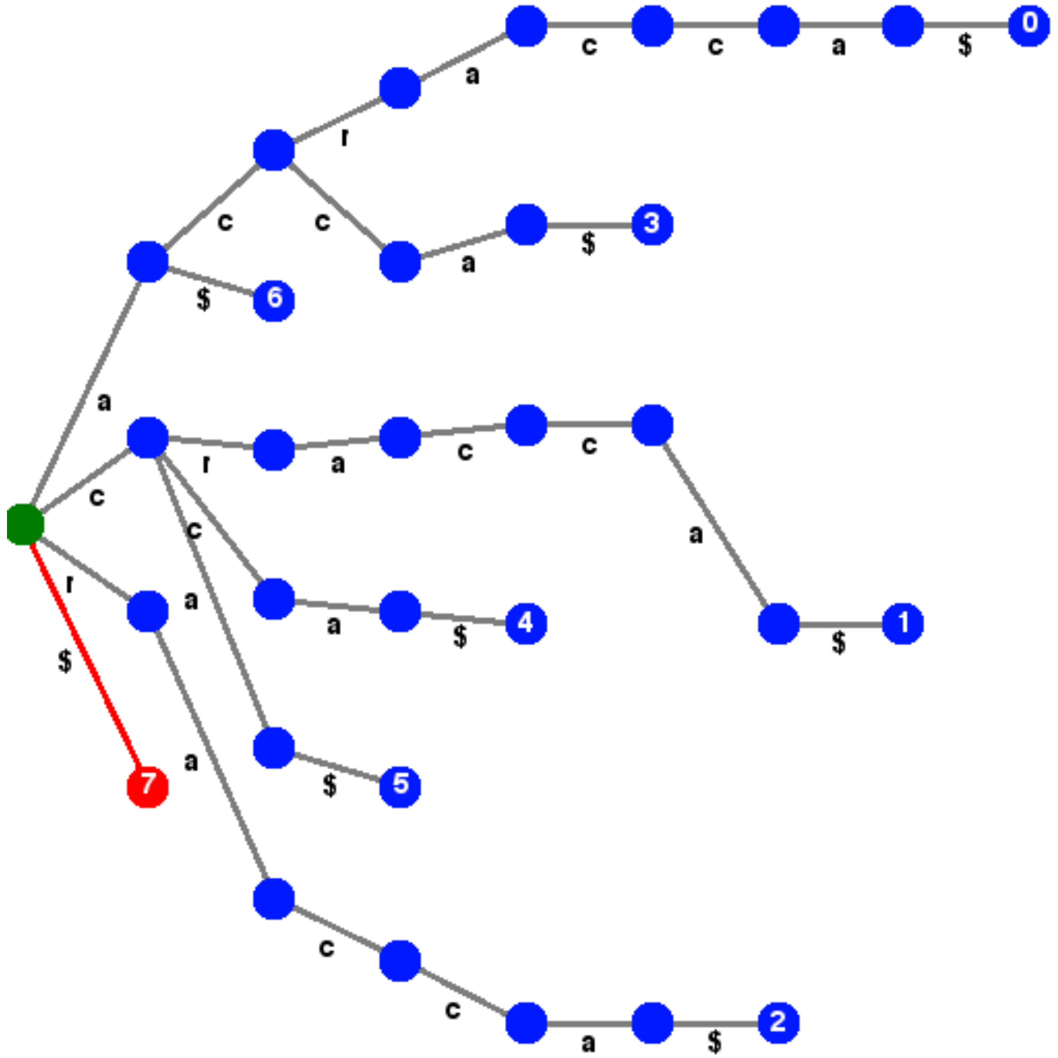


- Membership query:
  s ∈ S
- Prefix query:
  ∃ t ∈ S: s is prefix of t
- Sorting string
- Unique strings
- Count strings

Example: T = acracca$

```
acracca$
 cracca$
  racca$
   acca$
    cca$
     ca$
      a$
       $
```

Trie of all suffixes of T



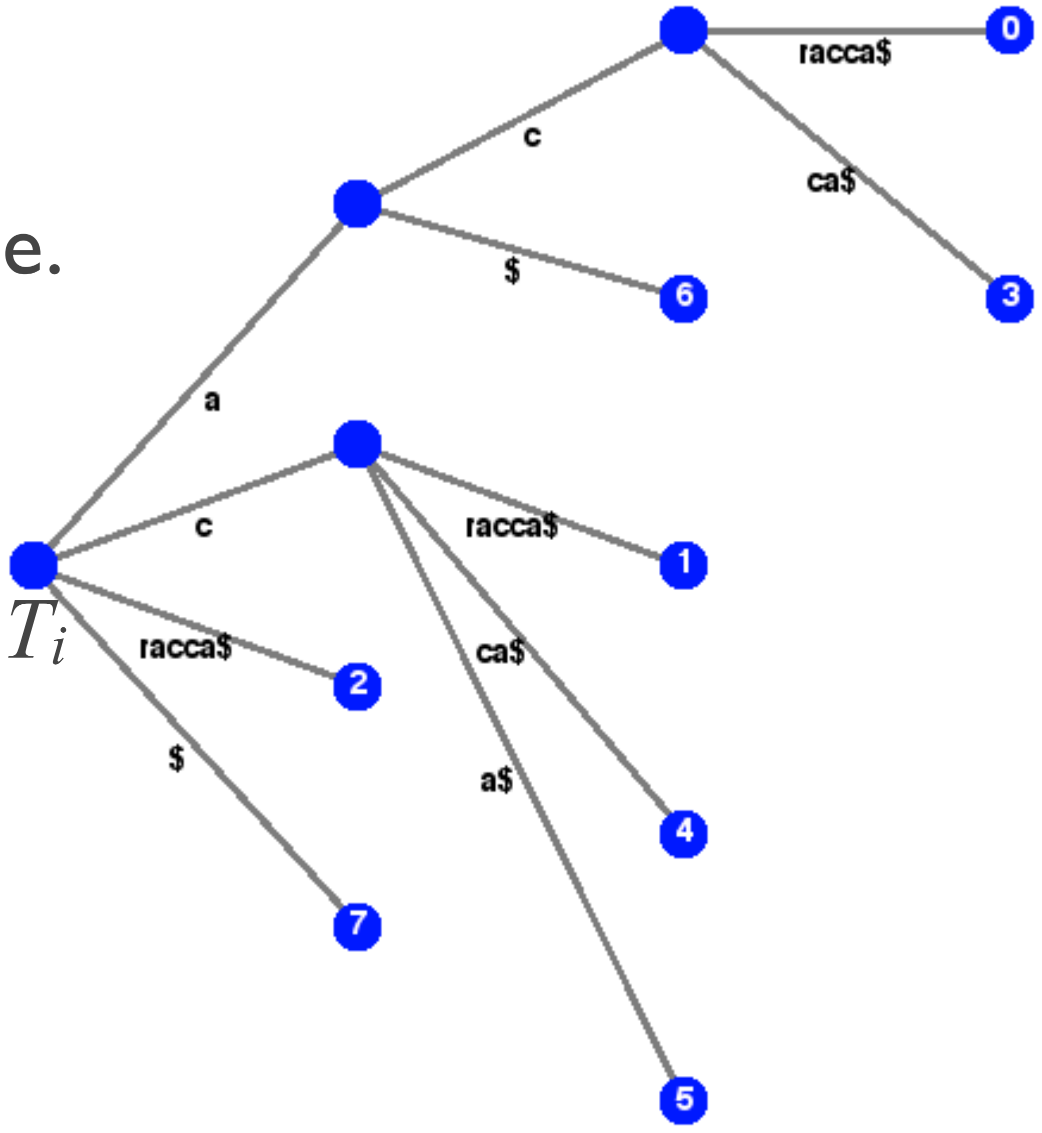
T = banana

Is "na" a substring of T

$\underline{\text{nana}}$
$\underline{\text{na}}$ are suffixes of banana

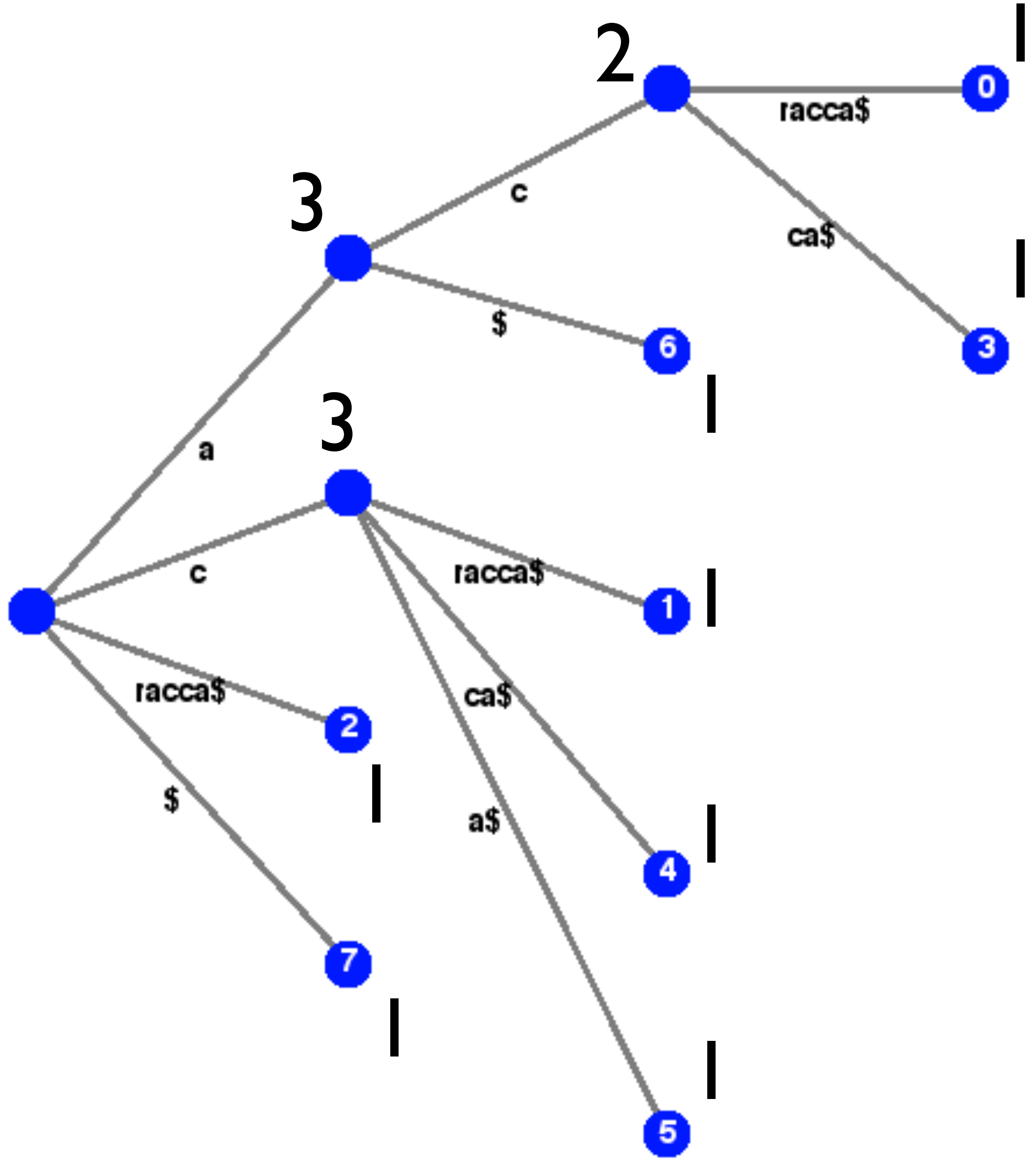
Definition: Suffix tree

- Given $T=t_1t_2\ldots t_n$ its suffix tree $\mathcal{T}_T$ is a rooted tree. $\mathcal{T}_T$ has
  - n leaves; leaf l_i corresponds to suffix T_i
  - edge label label(u,v): sub-string
- Can be constructed in O(|T|)



Application: #occurrences of s in T?

- Use tree-traversal to populate leaf-counts for all internal nodes (once!)
- If for some explicit or implicit vertex u, L[u] = s then s is a substring and its count is given by u (or its least explicit predecessor)
- Complexity: O(|s|)
- Independent of |T| !



Outlook:

- Suffix tree for Human Genome ~40GB
- Burrows-Wheeler-Transform (BWT) and FM-index support same operations with same complexity but index is compressed to ~4 GB

Accelerating computations

Sketches

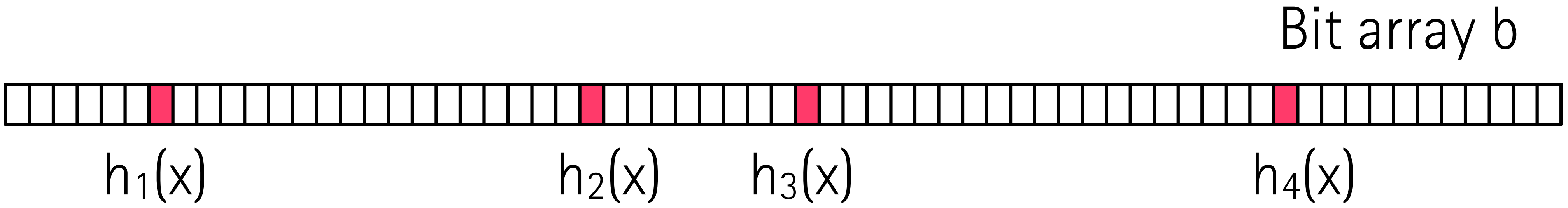
Counting observations

- Data: m elements from $\{1,2,...,n\}$, $m >> n$
- f_i frequency of i
- Query types:
  - Point query: frequency f_i of i
  - Range Query: for i, j estimate $f_i + f_{i+1} + ... + f_j$
  - Quantile query: find i s.t. $f_1 + f_2 + ... + f_i \sim x$
  - Heavy hitters: find i s.t. $f_i \geqq m\,x$

Subsampling

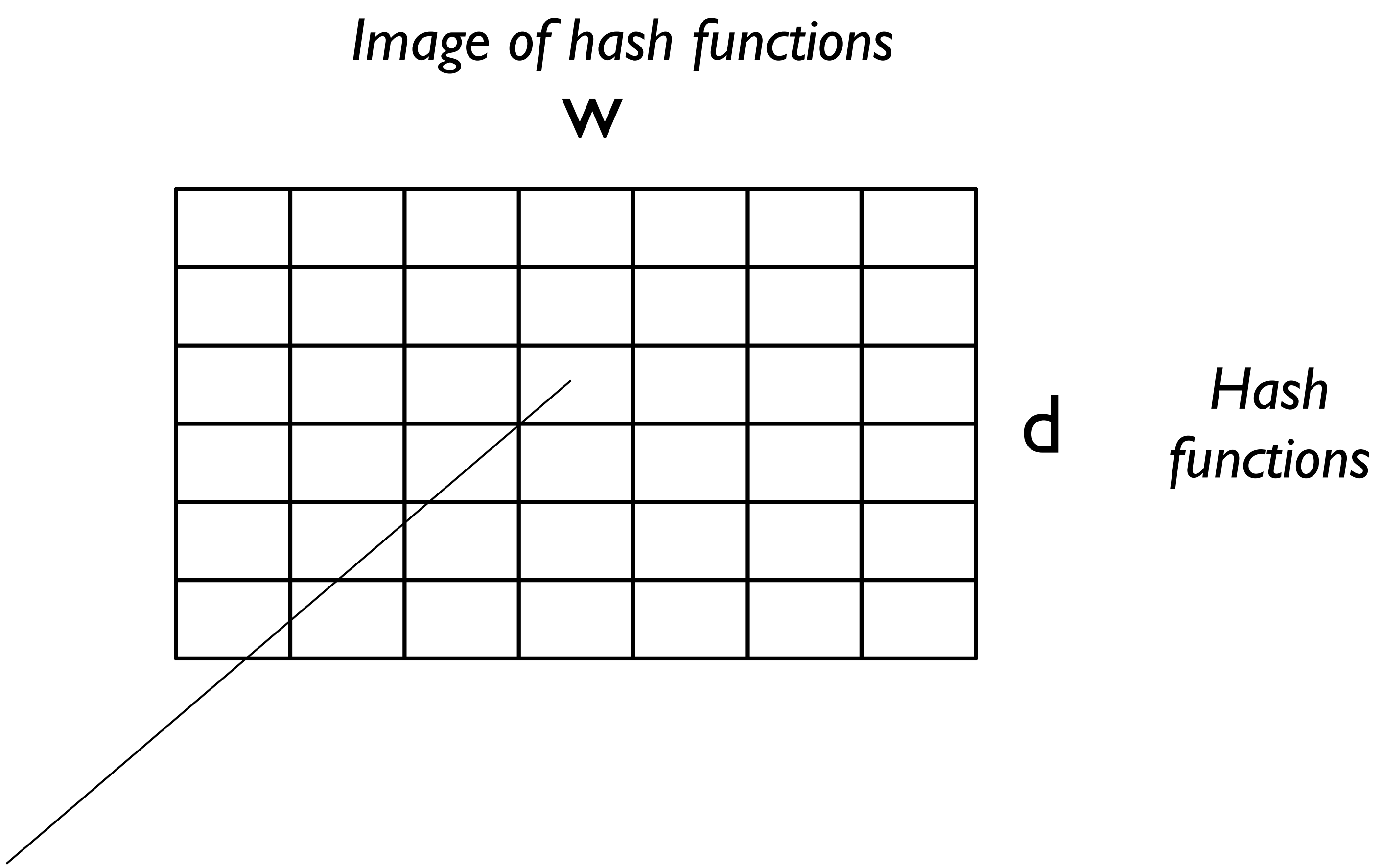
- Randomly select a small subset of the data
- Compute summary statistics on this

Bloom filter



How to change a Bloom Filter so that it can count?

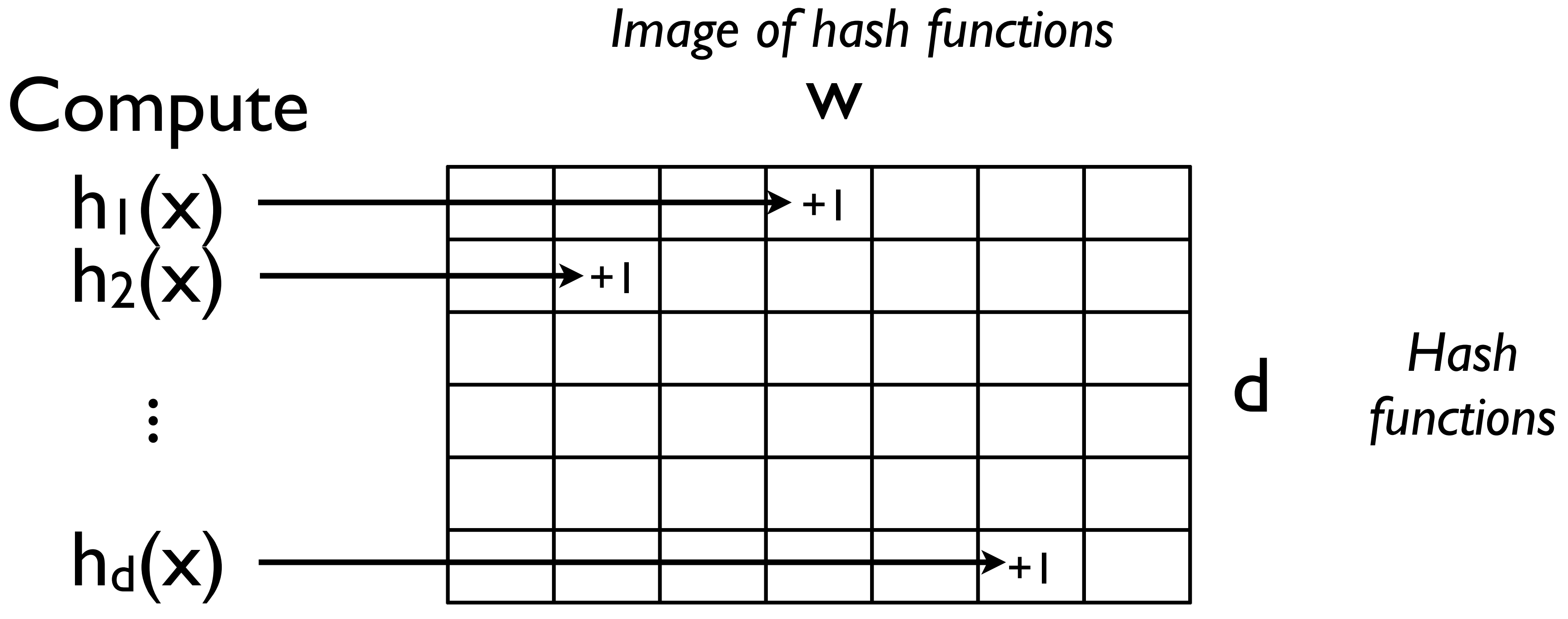
Count-min sketch



$c_{i,j}$ = # of elements with $h_i(x) = j$

e element in the data

Count-min sketch: Update



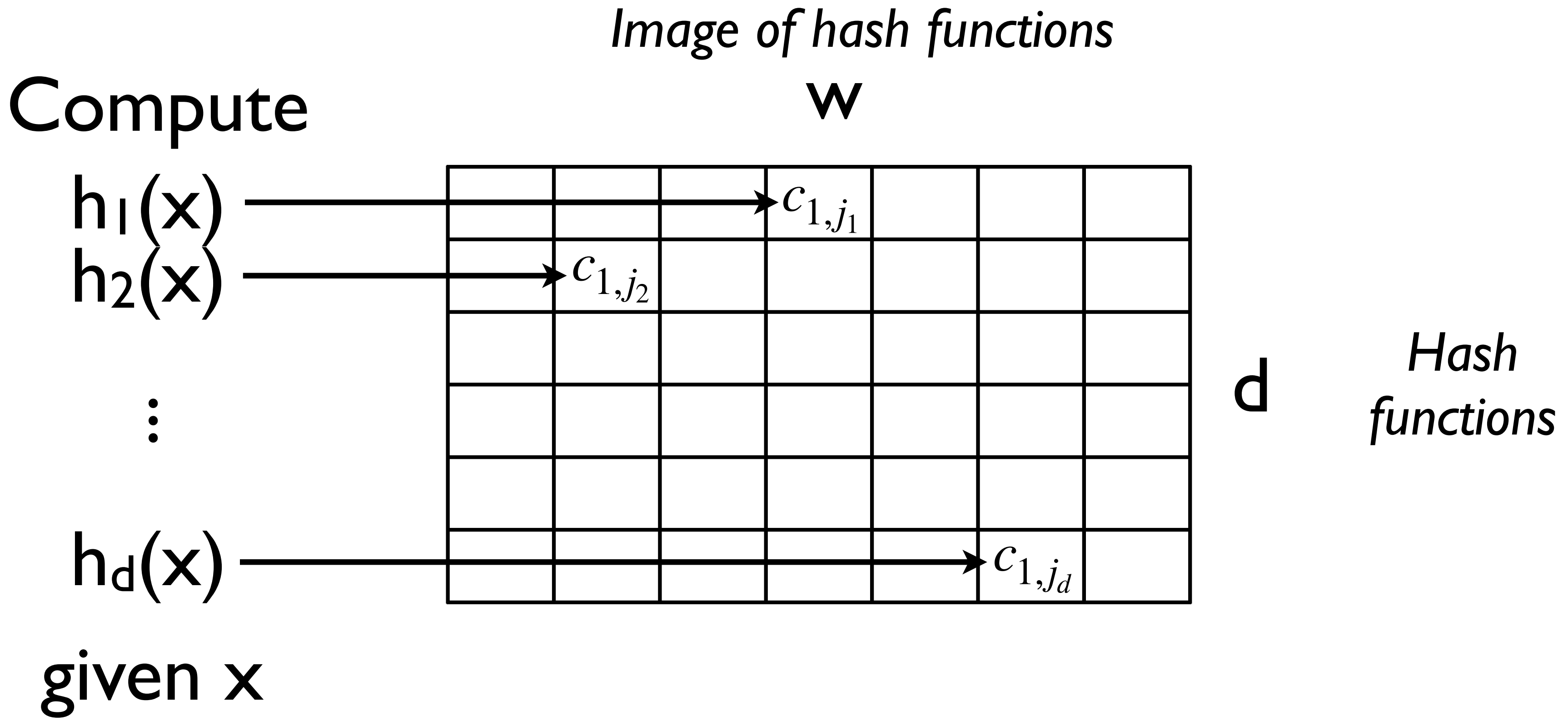
given x

Increment counters

Count min update can effectively be parallelized.

A	B	C	D	E
True	False			

Count-min sketch: Query



$$\tilde{f}_x = min\{c_{1,j_1}, c_{2,j_2}, \ldots, c_{d,j_d}\}$$

$$= min\{c_{1,H_1(x)}, c_{2,H_2(x)}, \ldots, c_{d,H_d(x)}\}$$

Count-min sketch:observations

- For all e, i: $c_{i,H_i(x)} \geq f_x$ so $\tilde{f}_x \geq f_x$
- d=1, w=n, H_1(x) = x: Exact counting using space linear in n
- Space: d x w x sizeof(counter)
- Increasing d?
- Decreasing w?

Count-min sketch: analysis

For w = 2/ε and d = $\log_2(1/\delta)$:

$$P[f_x \leq \tilde{f}_x \leq f_x + \epsilon m] \geq 1 - \delta$$

Need 2-way independent hash functions

$$P[h_i(x) = h_i(y)] \leq \frac{1}{w}$$

Count-min sketch: analysis

For w = 2/ε and d = $\log_2(1/\delta)$:

$$P[f_x \leq \tilde{f}_x \leq f_x + \epsilon m] \geq 1 - \delta$$

Error bound of estimate

Larger w, smaller ε

E.g. for error bound of
εm=2000 for m = 10^6
elements we need w = 1,000
at ε = 0.002

Count-min sketch: analysis

For $w = 2/\varepsilon$ and $d = \log_2(1/\delta)$:

$$P[f_x \leq \tilde{f}_x \leq f_x + \epsilon m] \geq 1 - \delta$$

Error bound of estimate
Larger w, smaller ε

Probability of staying within error bound
Larger d, smaller δ

E.g. for error bound of $\varepsilon m = 2000$ for $m = 10^6$ elements we need $w = 1{,}000$ at $\varepsilon = 0.002$

E.g. d=7 is needed for δ = 0.01

Memory usage: w * d * sizeof(counter) = 1000 * 7 * 4 bytes = 28kB

Accelerating computations

Index data structures

NOTE: The kd-trees + k-means part is for reference only. I mostly skipped it in the lecture. See the original nice paper.

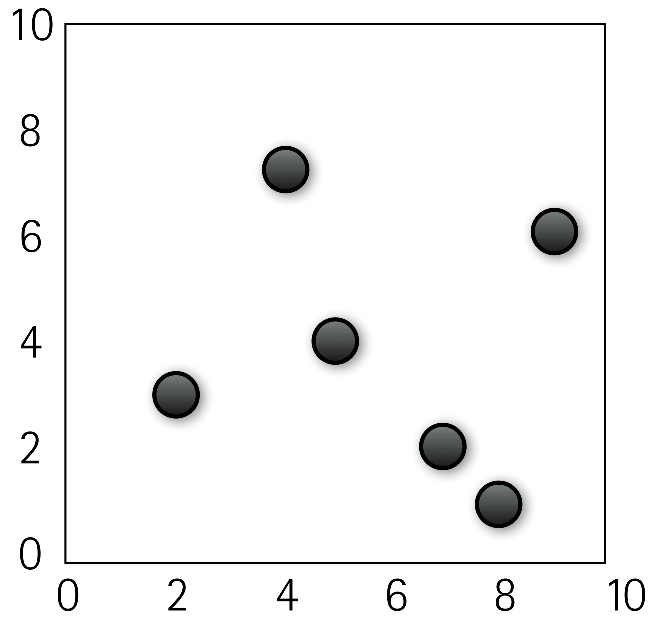
Pelleg, Moore: Accelerating exact k-means algorithms with geometric reasoning. KDD 1999.

Nearest neighbors

- Given a data base of n records with coordinates in Euclidean space (e.g. (x,z) or (x,y,z))
- For a query q, return the nearest neighbor in the database
- *Naive approach: O(n)*
- Application: GIS, computer games, dating websites/apps,...

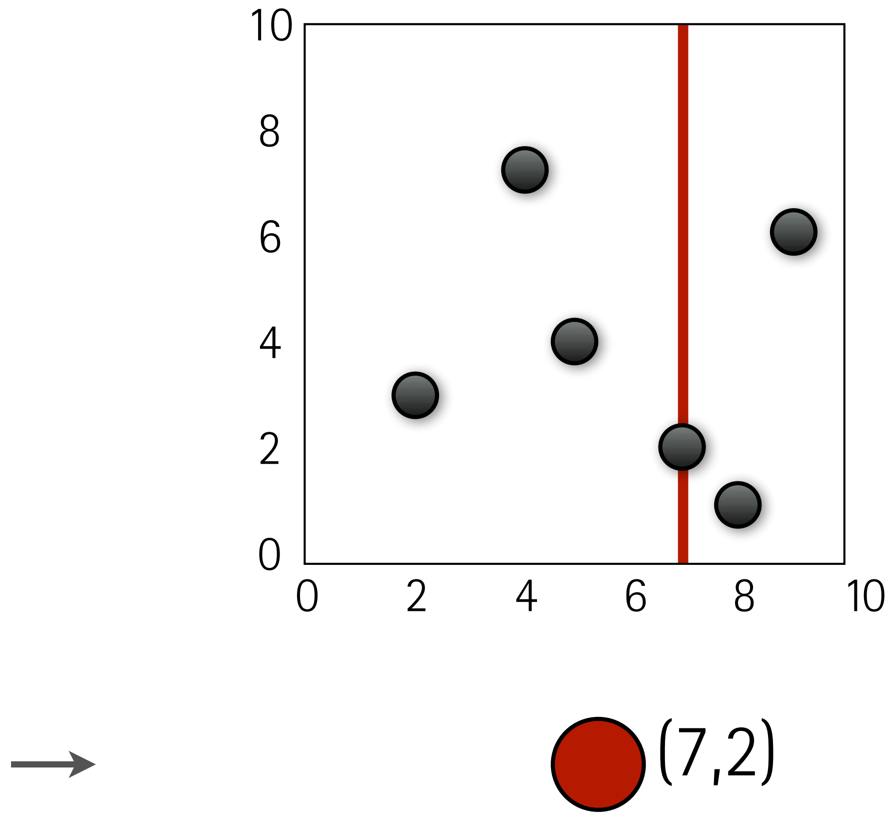
kd-trees

- Spatial index structure
- Balanced binary tree
- Fast nearest neighbor queries (dependent on instances)
- O(n log n) construction



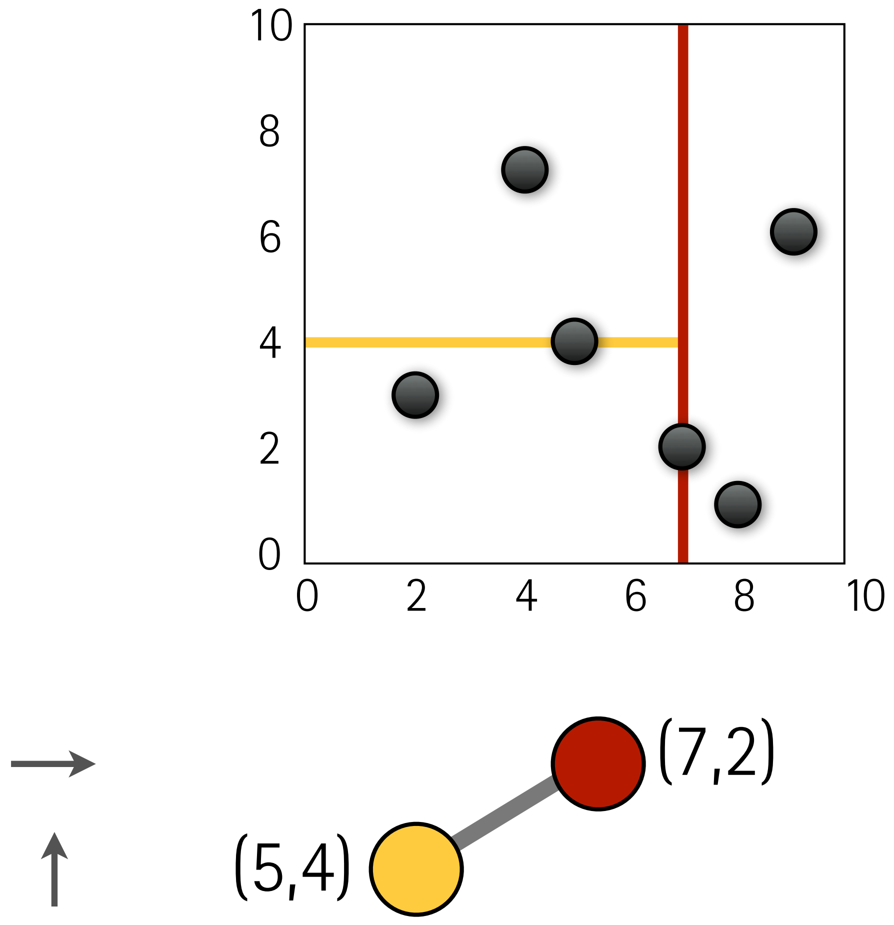
kd-trees

- Spatial index structure
- Balanced binary tree
- O(log n) nearest neighbor queries (dependent on instances)
- O(n log n) construction



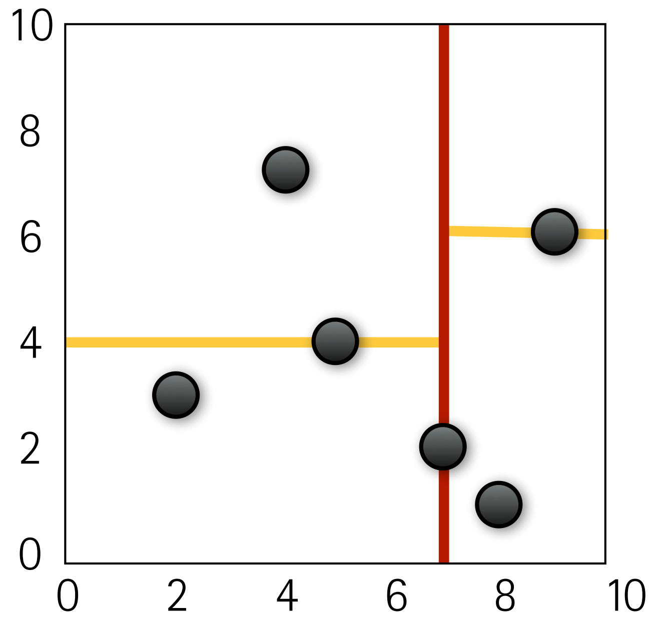
kd-trees

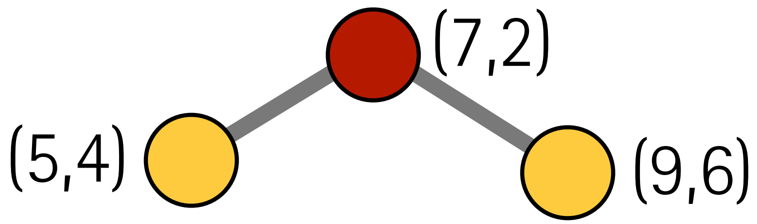
- Spatial index structure
- Balanced binary tree
- O(log n) nearest neighbor queries (dependent on instances)
- O(n log n) construction



kd-trees

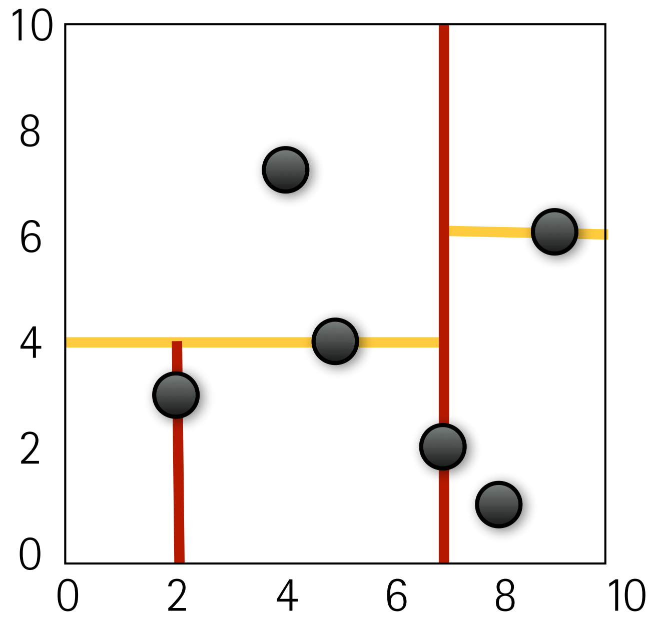
- Spatial index structure
- Balanced binary tree
- O(log n) nearest neighbor queries (dependent on instances)
- O(n log n) construction

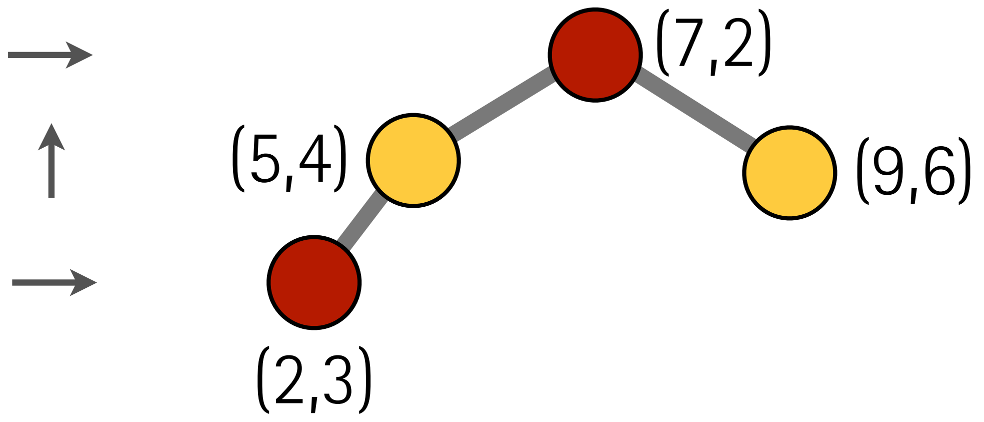




kd-trees

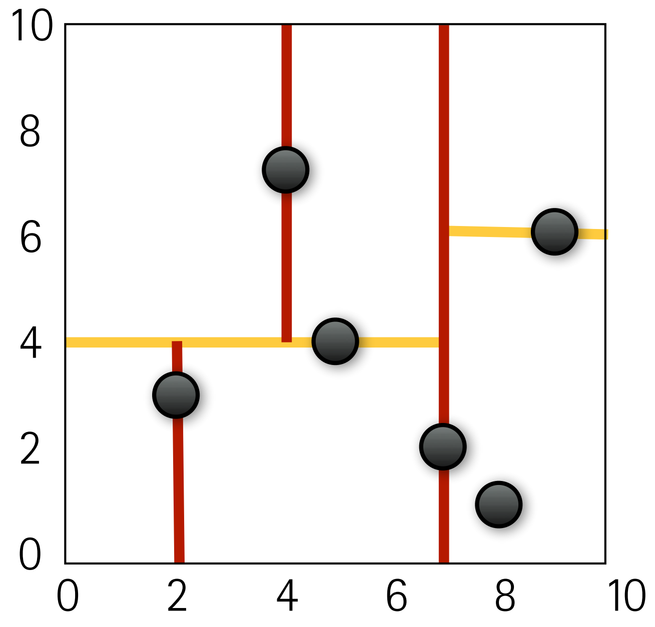
- Spatial index structure
- Balanced binary tree
- O(log n) nearest neighbor queries (dependent on instances)
- O(n log n) construction

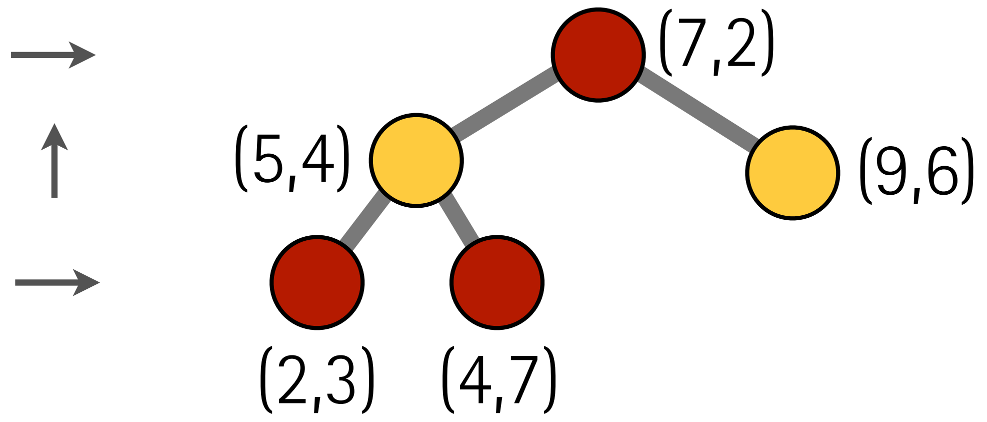




kd-trees

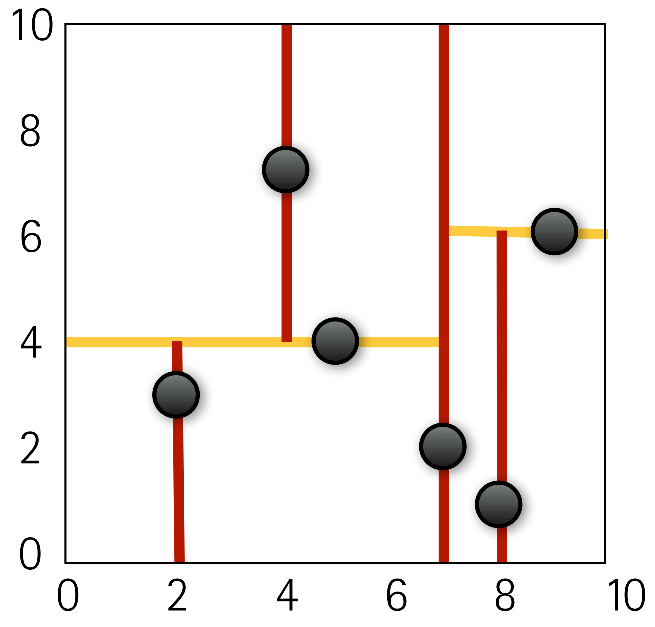
- Spatial index structure
- Balanced binary tree
- O(log n) nearest neighbor queries (dependent on instances)
- O(n log n) construction

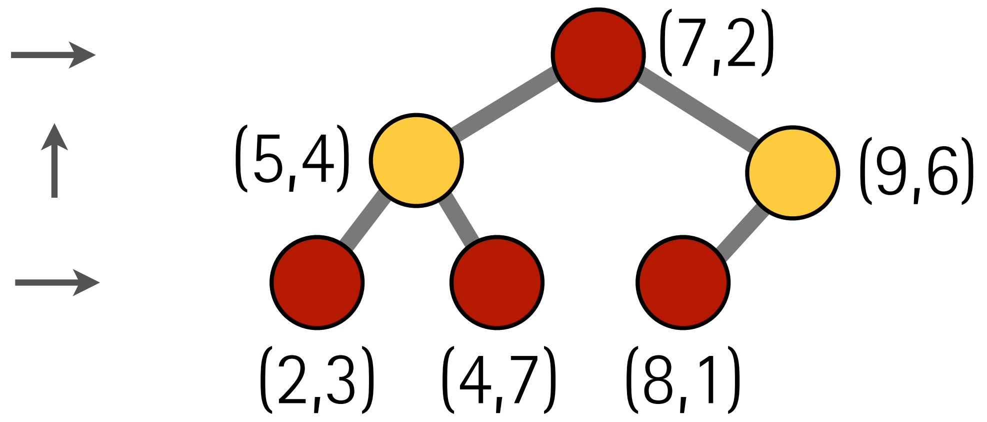




kd-trees

- Spatial index structure
- Balanced binary tree
- O(log n) nearest neighbor queries (dependent on instances)
- O(n log n) construction





k-means clustering

$$min_{C_1,C_2,\ldots,C_k}\{\sum_{k=1}^{K} W(C_k)\}$$

$$W(C_k) = \frac{1}{|C_k|}\sum_{i,j\in C_k} \| x_i - x_j \|^2$$

$\| x - y \|^2$ Euclidean distance

k-means clustering

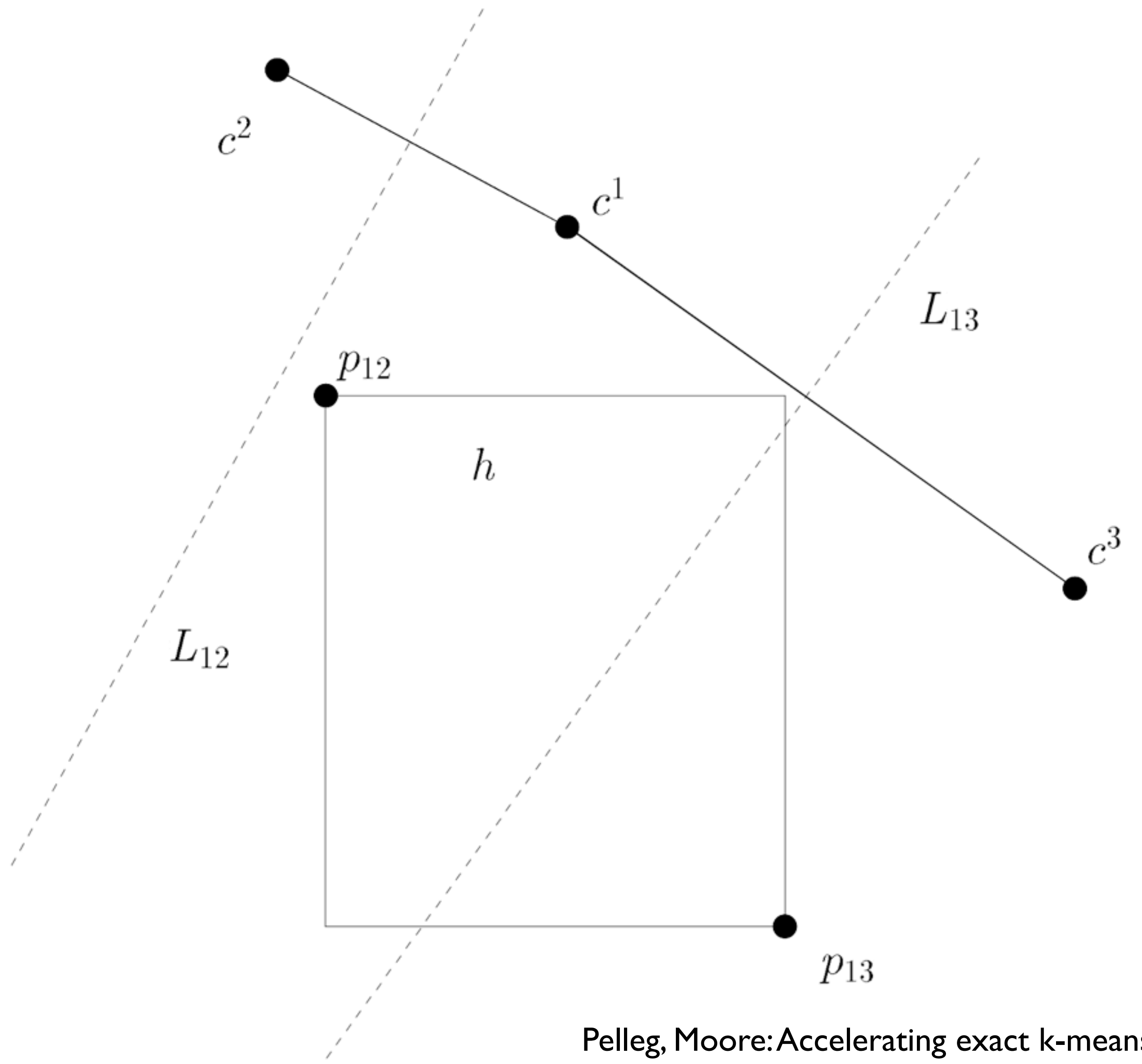
- Chose k centroids among data points
- Iterate:
  - assign data points to closest centroid $O(n \times k)$
  - recompute centroids $O(n)$

Accelerating k-means with kd-trees

- Idea: Avoid computing distance between every point and every centroid
- kd-trees partition space into hyper-rectangles

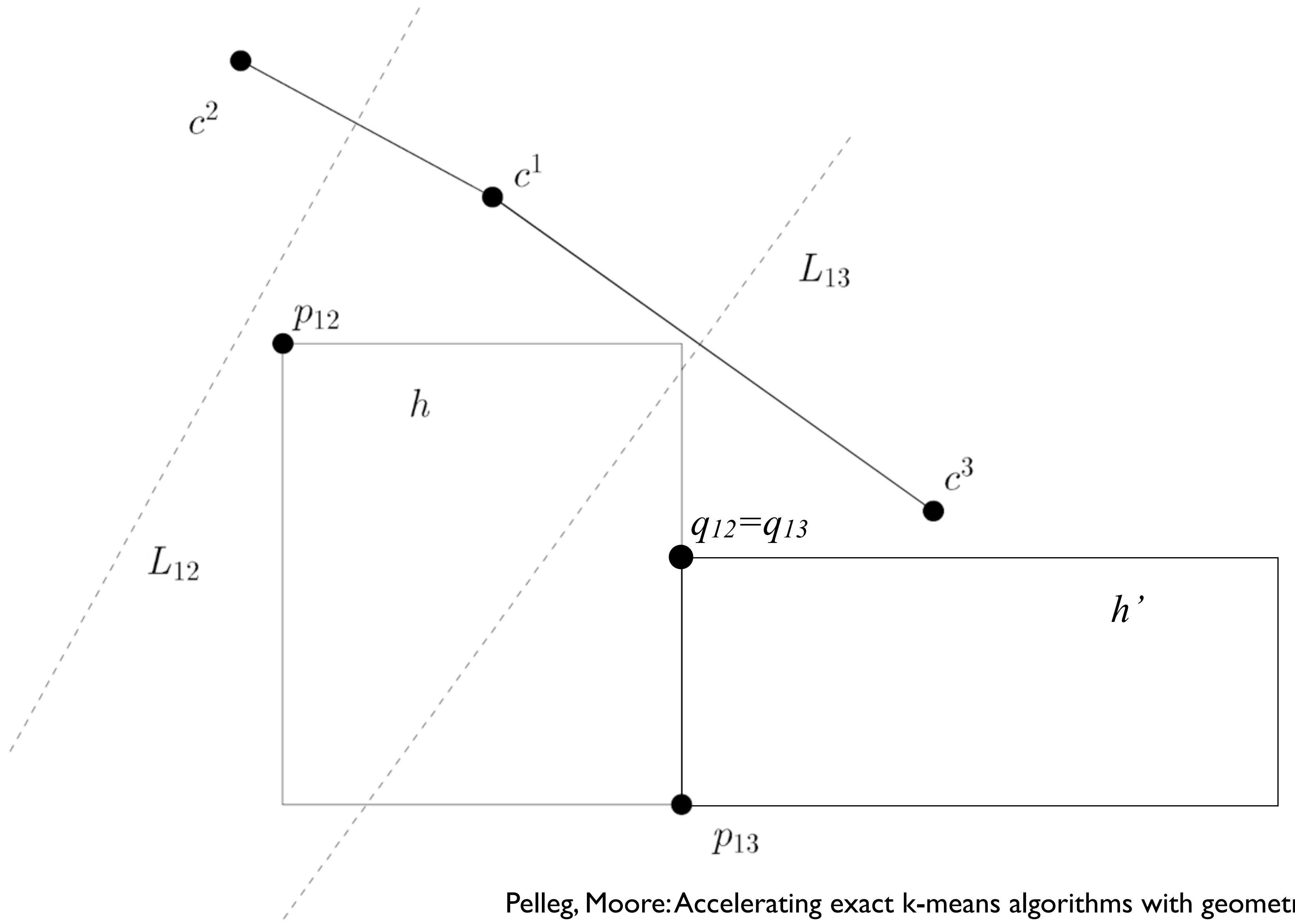
Pelleg, Moore: Accelerating exact k-means algorithms with geometric reasoning. KDD 1999.

Accelerating k-means with kd-trees



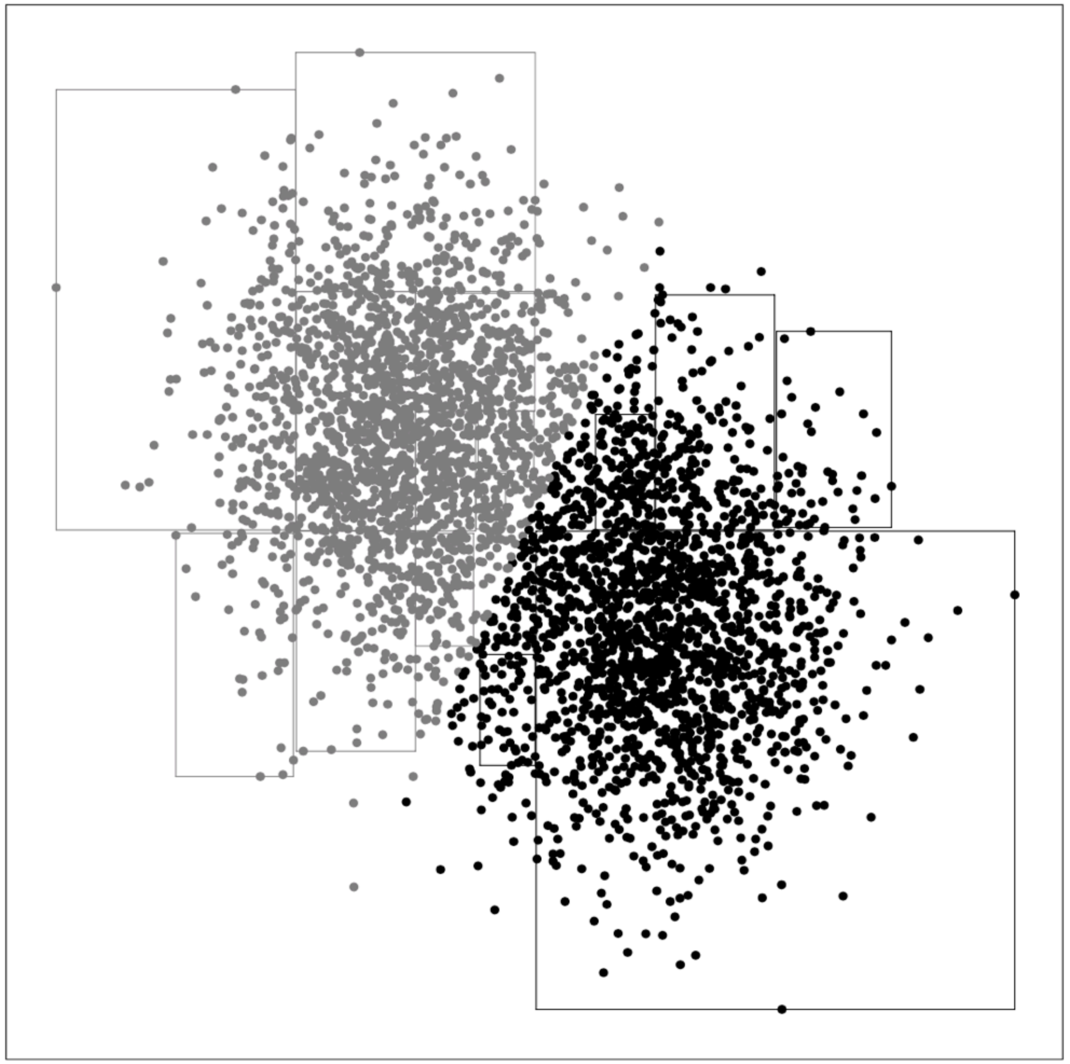
Pelleg, Moore: Accelerating exact k-means algorithms with geometric reasoning. KDD 1999.

Accelerating k-means with kd-trees



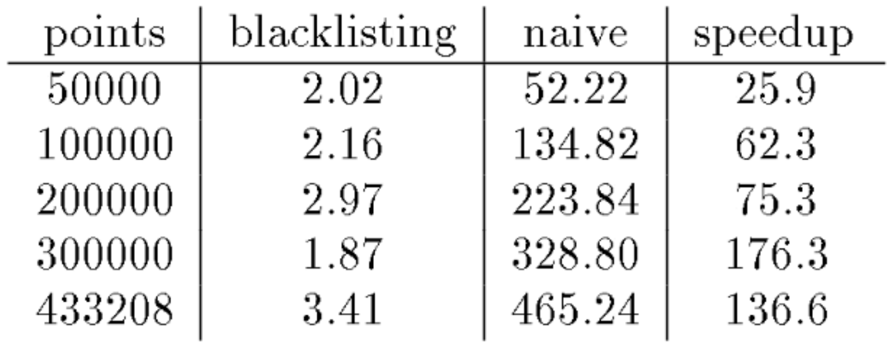
Pelleg, Moore: Accelerating exact k-means algorithms with geometric reasoning. KDD 1999.

Accelerating k-means with kd-trees



Pelleg, Moore: Accelerating exact k-means algorithms with geometric reasoning. KDD 1999.

Accelerating k-means with kd-trees



points	blacklisting	naive	speedup
50000	2.02	52.22	25.9
100000	2.16	134.82	62.3
200000	2.97	223.84	75.3
300000	1.87	328.80	176.3
433208	3.41	465.24	136.6

Table 1: Comparative results on real data.

Run-times of the naive and blacklisting algorithm, in seconds per iteration. Run-times of the naive algorithms also shown as their ratio to the running time of the blacklisting algorithm, and as a function of number of points. Results were obtained on random samples from the 2-D "petro" file using 5000 centers.

Pelleg, Moore: Accelerating exact k-means algorithms with geometric reasoning. KDD 1999.

Use in methods

- For dimension up to 10
  - k-means
  - mixture estimation
  - k-nearest neighbor classification
- Active research topic: higher dimensions; specialization for specific combinations of n, k, d

Elkan, Using the triangle inequality to accelerate k-means, ICML, 2003

Kurban and Dalkilic: A novel approach to optimization of iterative machine learning algorithms: over heap structure. Big Data 2017

How to apply all of this …?

Answer: reluctantly

General approach

- Implement Data Science solutions the framework you are most familiar with (e.g. Python / Pandas) without worrying about efficiency (or parallelism)
- Probably will be okay for < 100 GB on Laptop and <1 TB of data on desktop

  (assuming SSDs / RAID, enough RAM, multi-cores)

Cost of optimization

- Qualified technical employee: cost to employer 1.5M SEK/year for 1600 hours/year ⇒ ~1000 SEK/hour
- Most expensive Amazon EC2 on-demand instance is 250 SEK/hour (+storage).
- Not worth to invest an hour, if not saving at least four.
- Q: How frequently does a workload run?

What if my data is big?

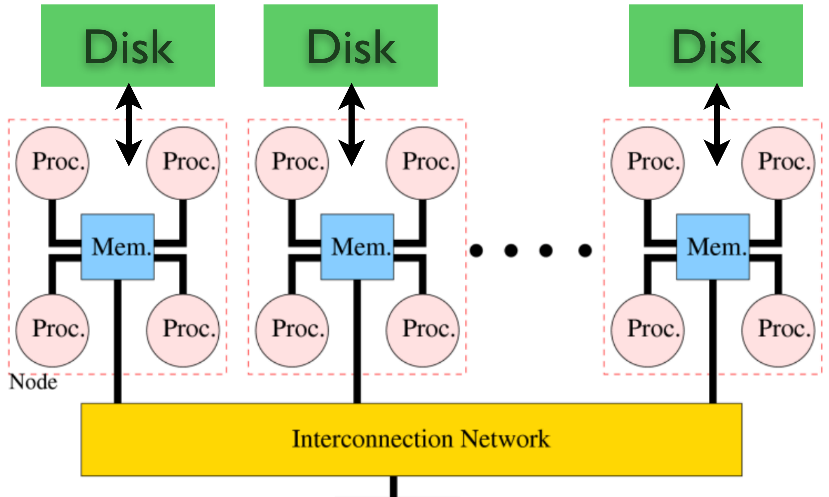
- Bandwith:
  - Single SSD: 0.5 GB/s
  - Large RAID: 1 GB/s
  - DDR4 RAM > 20 GB/s (peak)
- Reading 1 (resp. 100) TB data needs:
  - 33 min resp. 55h
  - 17 min resp. 28 h
  - ~ 1 min resp resp. 1:40 h

Is Input/Output the bottleneck?

- Times below are only a bottleneck if computations are fast
  - 33 min resp. 55h
  - 17 min resp. 28 h
  - ~ 1 min resp resp. 1:40 h
- Alternative: external-memory frameworks and algorithms.
  - E.g. https://dask.org/ Pandas frames too large for RAM

What if my data is big?

- Solutions:
  - Up to a point expensive hardware (e.g. RAIDs with SSDs)
  - Cluster with local disks & HDFS
- ⇒ You need to parallelize



What if my solution is too slow?

- General ideas:
  - Change the problem
  - Refactor
  - Compile
  - Optimize
  - Parallelize

Change the problem

- E.g. Descriptive statistics with MapReduce: Is the median needed?
- Identifying Duplicates: Are a few false positive duplicates acceptable?
- If a median (or quartiles are needed): are approximate estimates acceptable?
- If a classifier is used: Is a simpler algorithm (e.g. k-nearest neighbor instead of deep learning) acceptable?

Refactor

- Generally: short Python programs are fast Python programs.
- Use libraries: E.g. numpy and others offering C/C++-speed

```
def nearestCentroid(datum, centroids):
    # norm(a-b) is Euclidean distance, matrix - vector computes difference
    # for all rows of matrix
    dist = np.linalg.norm(centroids - datum, axis=1)
    return np.argmin(dist), np.min(dist)
```

- Might make some parallelization trivial

```
import mkl
import numpy as np

mkl.set_num_threads(4)
np.dot(x,y)
```

Compile your Python Code

Cython
https://cython.org

```
def primes(int nb_primes):
    cdef int n, i, len_p
    cdef int p[1000]
    if nb_primes > 1000:
        nb_primes = 1000

    len_p = 0  # The current number of elements in p.
    n = 2
    while len_p < nb_primes:
        # Is n prime?
        for i in p[:len_p]:
            if n % i == 0:
                break

        # If no break in the loop, we have a prime.
        else:
            p[len_p] = n
            len_p += 1
        n += 1
   # Let's return the result in a python list:
    result_as_list  = [prime for prime in p[:len_p]]
    return result_as_list
```

Some parallelization support OpenMP (not clusters though)
Allows calling of C libraries, use of C-structs etc.

Numba
https://numba.pydata.org/

```
from numba import jit
import random

@jit(nopython=True)
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

```
@numba.jit(nopython=True, parallel=True)
def logistic_regression(Y, X, w, iterations)
    for i in range(iterations):
        w -= np.dot(((1.0 /
              (1.0 + np.exp(-Y * np.dot(X, w
              - 1.0) * Y), X)
    return w
```

Also parallelizes incl. GPUs (not clusters though)

Optimize

- Measure running time and identify hot spots
- Better data structures and algorithms usually beats code optimization
  - Constants matter
  - Latency/memory hierarchy matters
  - ⇒ Lower computational complexity does not always win
- Sometimes optimizing makes sense

Parallelize

- Based on structure of workload: data flow, computational effort, hardware …
- Measure Amdahl's f
- SIMD instructions
- Multi-threading
- Message passing
- MapReduce, Spark, … and many big data computational frameworks.
- Question: Are there specialized frameworks for *your* problem?
- Avoid oversubscription …

Parallelize

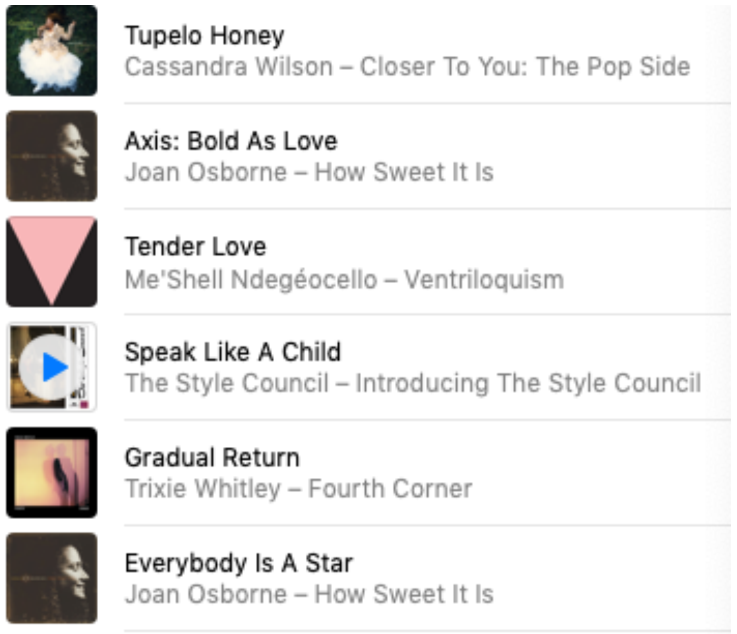
- Achieve scaling via:
  - Dedicated hardware: GPU, Tensor Units for deep learning
  - Clusters
  - Clouds …
  - *Volunteer Clouds (Folding at home…)*

Outlook

- Very active research field
  - Intel: https://github.com/IntelPython, https://www.youtube.com/watch?v=HKjM3peINtw (SciPy 2018 talk)
  - Amazon–Uber: https://arrow.apache.org/; see https://streamdata.io/blog/open-source-apache-big-data-projects/
  - Tool provider: Anaconda (e.g. Numba) with Intel, Nvidia and AMD

Q&A from Chat

Q: Where to find hash functions? Defined your own? A: Good hash functions are a science and art in itself. I always took the state-of-the-art for a specific task from the literature. Note that there is a tradeoff between speed and quality.



Tupelo Honey
Cassandra Wilson – Closer To You: The Pop Side

Axis: Bold As Love
Joan Osborne – How Sweet It Is

Tender Love
Me'Shell Ndegéocello – Ventriloquism

Speak Like A Child
The Style Council – Introducing The Style Council

Gradual Return
Trixie Whitley – Fourth Corner

Everybody Is A Star
Joan Osborne – How Sweet It Is