Fast Algorithms for BIG DATA

(title means “I make slides according to the interests of audience)

Takeaki Uno
National Institute of Informatics
& Graduated School for Advanced Studies

14/Jan/2012 NII Shonan-meeting (open problem seminar)

Self Introduction, for Better Understanding

Takeaki Uno: National Institute of Informatics
(institute for activating joint research)

Research Area: Algorithms, (data mining, genome science)

+ Enumeration, Graph algorithms, Pattern mining, Similarity search
(Tree algorithms, Dynamic programming, NP-hardness, Reverse search,…)

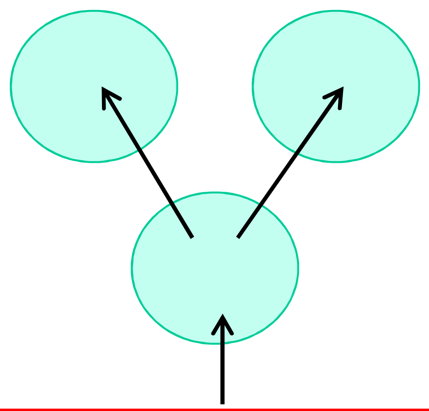
+ Implementations for Frequent Itemset Mining,
for String Similarity Search,…
+ Web systems for small business optimizations
+ Problem solver, or implementer, for research colleagues

Algorithm vs. Distributed Computation

- For fast computation, both “algorithm” and “distributed computation” are important

- However, sometimes they conflict

+ distributed algorithms sometimes doesn’t match latest algorithms, but is faster than the latest
+ new algorithms (or model) sometime vanish the existing distributed algorithms



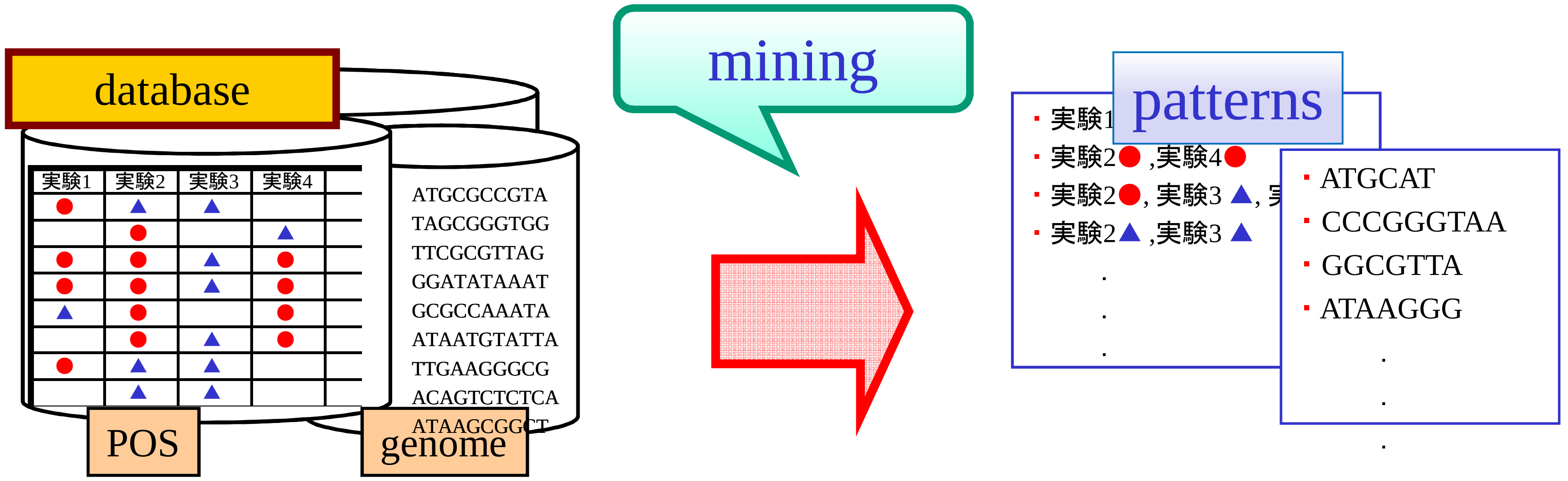
We should find “fundamental algorithms” that doesn’t change in future, and fit distributed computation

Frequent Itemset Mining

Frequent Itemset (pattern) Mining

- Problem of enumerating all frequently appearing patterns in big data (itemset = pattern that is a subset of the entire set)
- Nowadays, one of the fundamental problems in data mining
- We want to do this in (data) distributed system, but not easy

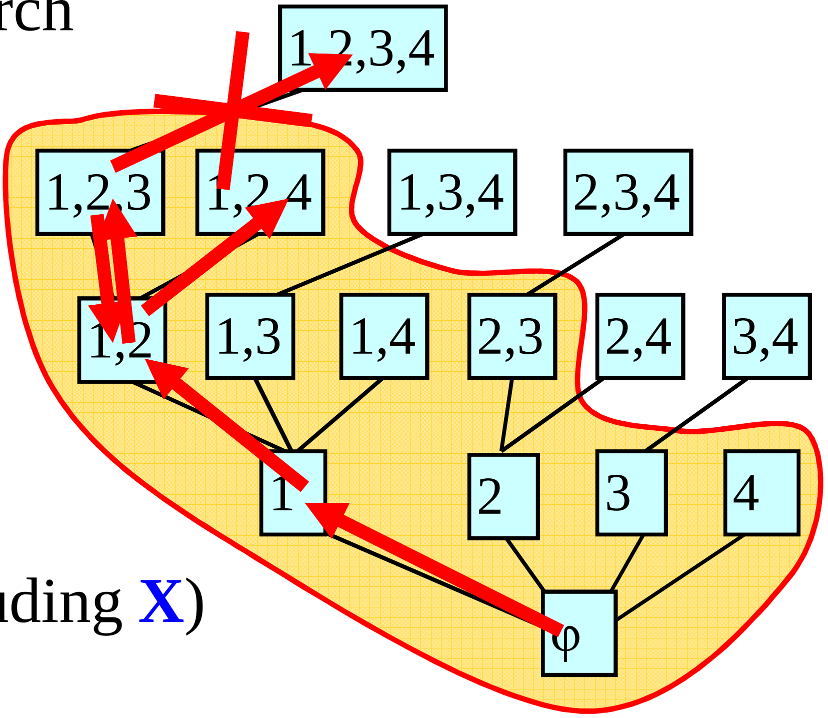
Observe the difficulties, and possibilities



History of Algorithms

- The beginning is early 90's, and many algorithms follow
- Several break-through
  - + breadth-first search → depth-first search
  - + naive frequency counting → recursive data compression
  - + heuristic pruning → reverse search

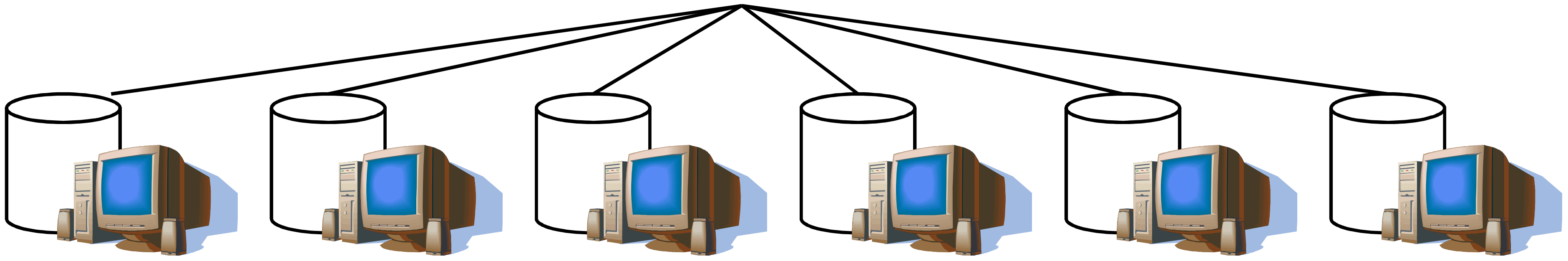
- Latest algorithm is, basically, composed of DFS + compression



```
MINING (X){
  output  X (,and compress data including X)
  for each X+e
    if  X+e is frequent  call MINING (X+e)
```

Work Stealing

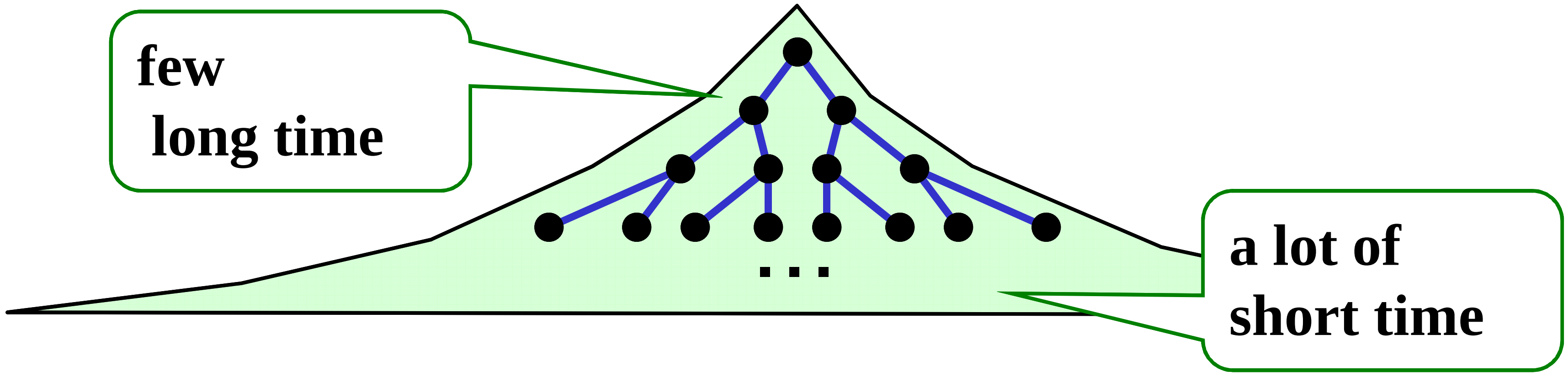
- DFS is a kind of Branch-and-Bound
  - ➔ Work Stealing would work!
- But,… it DOESN'T!
- To steal a work, we need to copy a big data!
  - ← this is the bottle neck
- If we copy the original data to everyone, many computers have to do the same operation, to generate the input to be moved



Bottom-wideness

- branch-and-bound explodes as going to deeper levels

  ➔ total computation time is dominated by those of deepest levels



Keeping deeper levels shorter time is important for fast enumeration

…so, data compression for recursive call (for deeper levels) is important , to reduce the time for "frequency counting"

How to Data Compression

- Reduce the database to speed up the bottom level iterations

In **MINING** (**X**), we do

(1) Delete items less than the maximum item in **X**

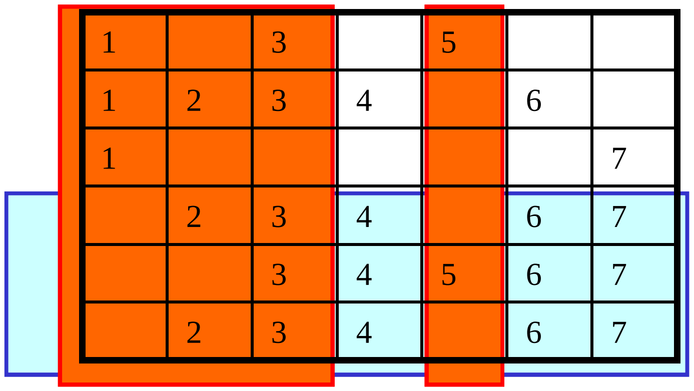
(2) Delete items being infrequent on the occurrence set database (since it never be added in the recursive call)

(3) unify the same transactions

X={1,3}, k=1, σ=4

- The database size is constant in the bottom levels in practice

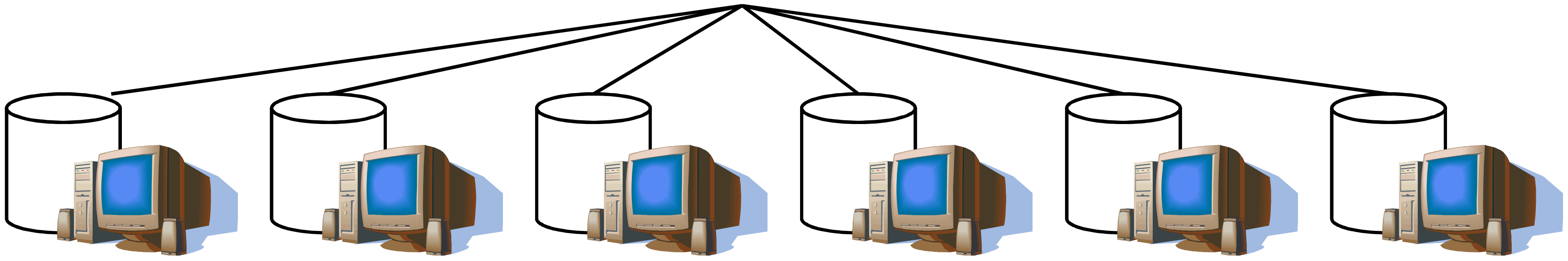
Bottom levels would not be a great bottleneck



1		3		5		
1	2	3	4		6	
1						7
	2	3	4		6	7
		3	4	5	6	7
	2	3	4		6	7

Data-Distributed Computing

- Work stealing doesn’t work, so move to data distributed
  ➔ Then, frequency counting has to be in parallel
- There are many works! including approximate counting
- But,… communication cost is too much
  ← at least **O(m)** with **m** machines
- On the other hand, bottom levels take **O(1)** time to generate a frequent pattern, since the database size is **O(1)**



WHAT can we do?

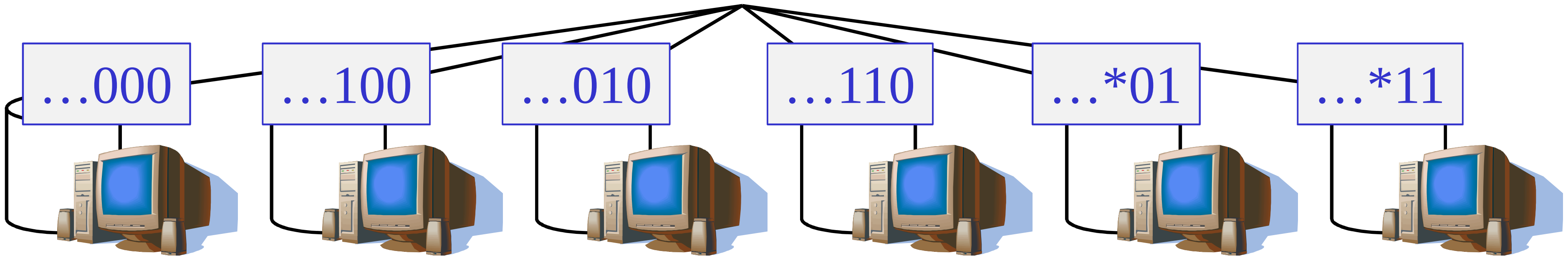
- Distributed computation, distributed data, both are not good can't we do something?

➔ Yes, we can do, if we can "shift" the focus little bit

- Suppose that we are allowed to define the data partition policy

- According to the most frequent **k** items, we define the database to belong, for each record

  ➔ All records in a data (computer) have the common "suffix"



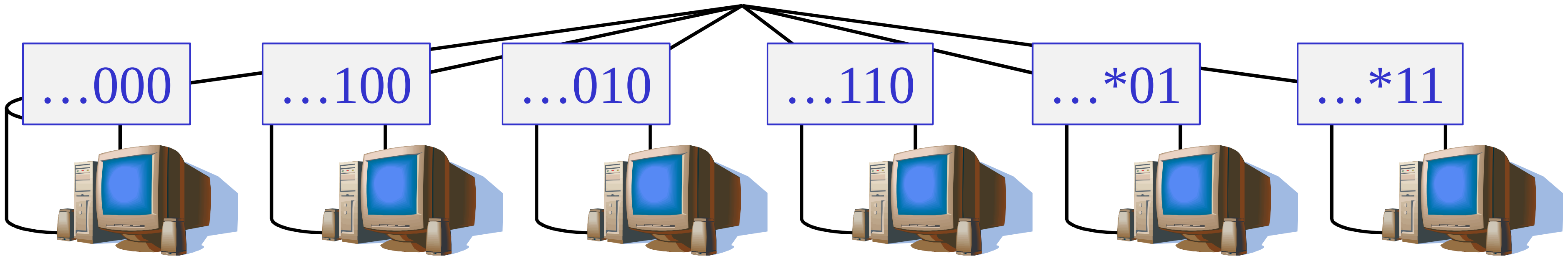
Distributed data&comp.

- In the bottom levels, compressed data is composed only of suffix
  ➔ By grouping records having the same suffix, communication is not needed

  ➔ Data partition policy reduces the communication cost!

- In upper levels, we need to communicate
  when we have **m** machines, suffix length is **$\log_2 m$**
  there are possibly **$O(2^{\log m})$** frequent itemsets

➔ Communication cost per frequent itemset would be constant



Is it a Tip?

- Of course, this is a **tip** (from algorithm theory, maybe)

- But, for distributed computation, there are not many tips,

  ➔ we should at least have tips

- **Tips** are important in applications

  (I think, algorithm society should give algorithm tips to other areas)

- We can abstract fundamental problems / extract global techniques from tips, and it would be a theoretical work

➔ Finding tips are also important

Partition Strategy?

- It seems that strategy for data partition would be important

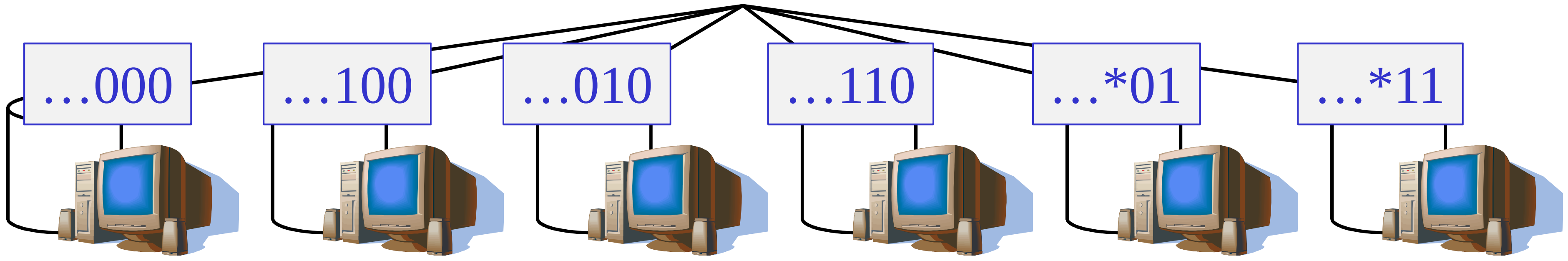
- However, the strategy is only for frequent itemset mining

  ➔ Moving data to other machines would take cost

  (but, it's OK if the mining cost is large)

  ➔ Can we determine the policy according only to this?

- We should know what kind of policies are there, from algorithmic view points, and which would be frequently used



To be Researched

- Basic algorithms are important to be researched
  ← what is basic?

- ”Basic” is, maybe, I would say,
  + a small/single part of an algorithm for basic problem,
    which is common to many algorithms
  + small but important “mutation” from usual problems
  + simple, not too much technical,
  + efficiency is clear (easy to understand why it is efficient)

- …would be fundamentals of large data analyzing algorithms

Similarity (neighbor) Search

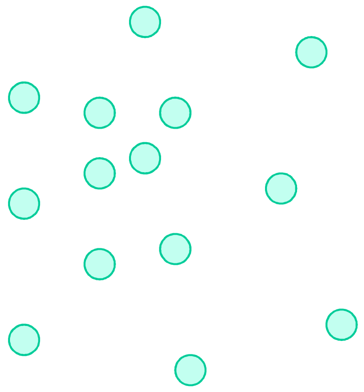
Similarity Search

• The problem to find records from the database that are similar to the given query record

• Different from exact search, approximation is not easy
 ← binary search paradigm doesn’t work

• On the other hand, we know that Local Sensitive Hash works
 → So the problem would be easier

However…



Combination of Bits

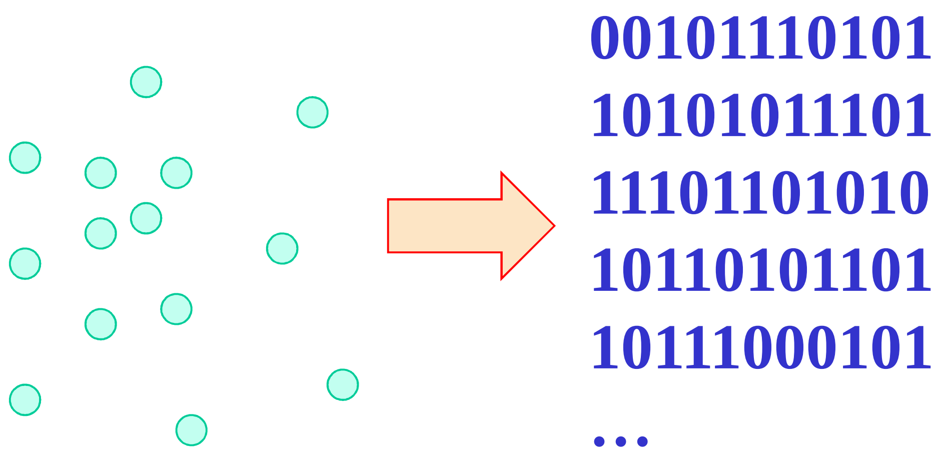
- Some application researchers say "LSH doesn't work!"

- LSH maps a record to 01-bit
  ← Similar records have the same bit in high probability
  ← We have to compare to the records having the same LSH bit

- To reduce #candidates, we combine some LSH bits to make a hash
  → For millions of records, we need 15-20 bits

- As a result, the probability to have same hash is small

→ we need many hashes, up to 100, to increase the chance



Reducing #hashes by Mismatches

- Data distribution by hash values can increase the performance
  ⇐ but, we need 100 of different data distributions

- To reduce #bits, we can introduce **“mismatches”**
  ➔ We compare with records having hash with mismatches
  ➔ 20 bit hash with 2 errors, we can reduce #hashes to about 8

- Finding records of mismatches at most small **d** can be done in, practically, short time by some algorithms

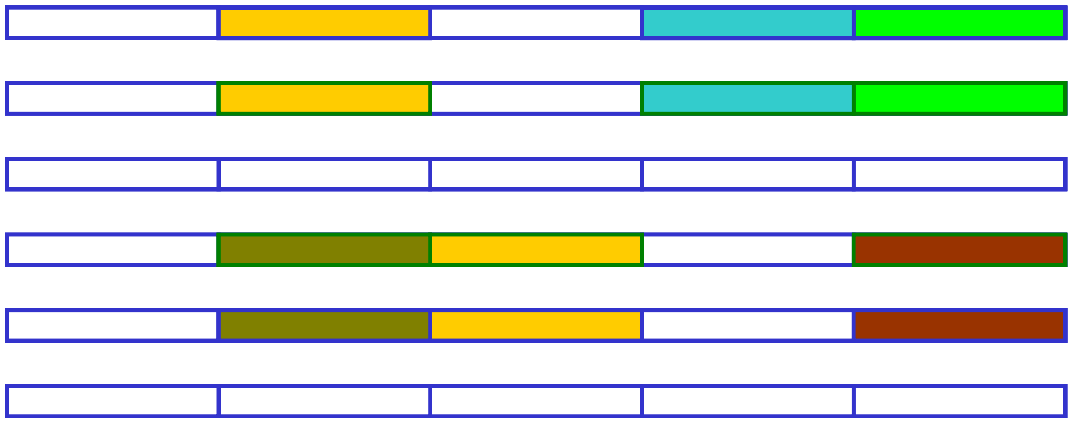
- We can at least reduce #duplicated data

00101011101
10101010101

Block Combination

- Consider partition of each string into **k(>d)** blocks
  - ➔ If two strings are similar, they share at least **k-d** same blocks
  - ➔ for each string, candidates of similar strings are those having at least **k-d** same blocks

- For all combinations of **k-d** blocks, we find the records having the same blocks (exact search)
  - ➔ we have to do several times, but not so many



Cost for Mismatches

- In a data distributed system, records of hashes with mismatches at most two are, generally, not in a single machine

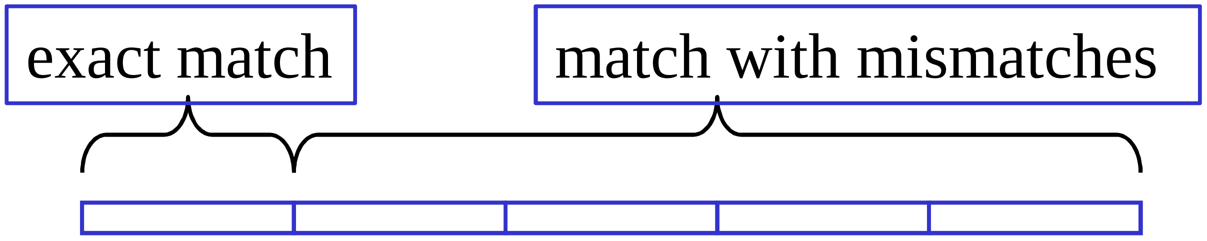
➔ we have to pay communication costs for search with mismatches

➔ we have a new difficulty

- For the problem, we can introduce a restriction of mismatching positions

➔ first **h** bits have to be the same, but we allow **d** mismatches in the remaining positions

➔ #necessary hashes is increased only a little bit



Again,

- Is this a tip? ➔ yes!

and…

+ Data partition strategy is important
  (my colleague says "partition is important for BFS")

+ Focus is on a small part (or key part) of the algorithm

+ Simple but would be efficient in practice

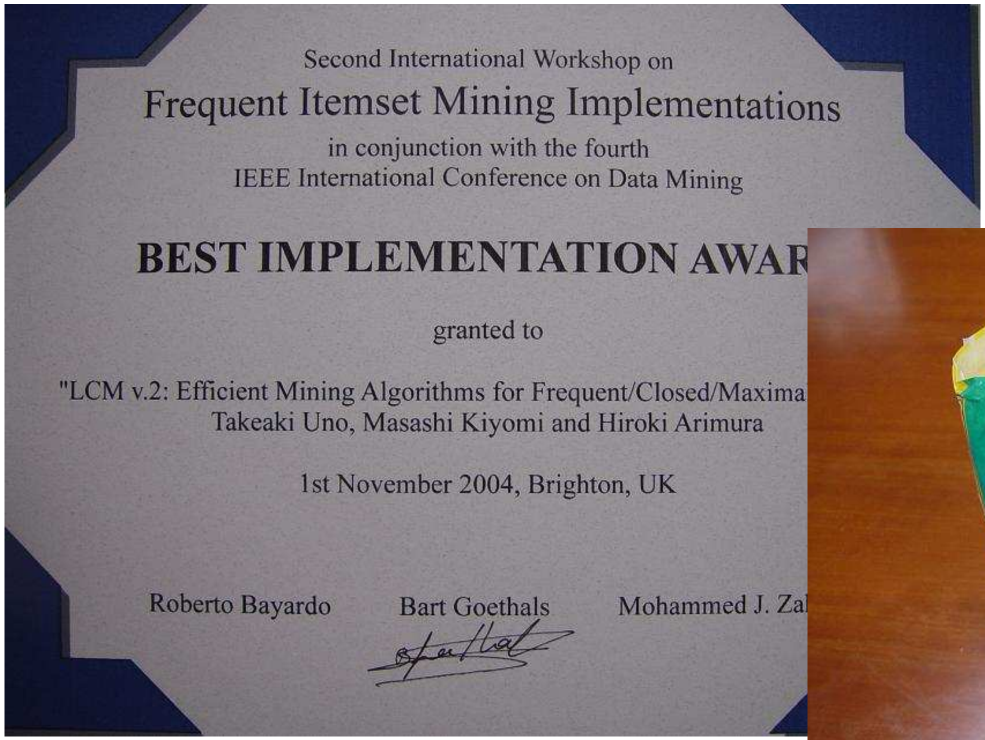
**Instead of fundamental (algorithmic) problems,
shall we start with fundamental fragments of algorithms?**

Other “Fragments”

- Extracting induced subgraph, for a vertex set
- Find (some) maximal clique
- Find (some) path
- Augment a matching
- Majority voting
- Frequency counting for sequence/graphs
- Range search
- All intersections of line segments
- Convex hull
- Huff transformation
- Dynamic Programming

…

Award: Frequent Itemset Mining Implementation



Second International Workshop on

Frequent Itemset Mining Implementations

in conjunction with the fourth

IEEE International Conference on Data Mining

BEST IMPLEMENTATION AWAR

granted to

"LCM v.2: Efficient Mining Algorithms for Frequent/Closed/Maxima

Takeaki Uno, Masashi Kiyomi and Hiroki Arimura

1st November 2004, Brighton, UK

Roberto Bayardo | Bart Goethals | Mohammed J. Za

[illegible]

Prize is {Beer, nappy}

…, which is the "Most Frequent Itemset"

Comparison; Human/Mouse Genome

Comparison of Mouse X and human X chromosome

(150MB for each, with 10% error)

15min. By PC

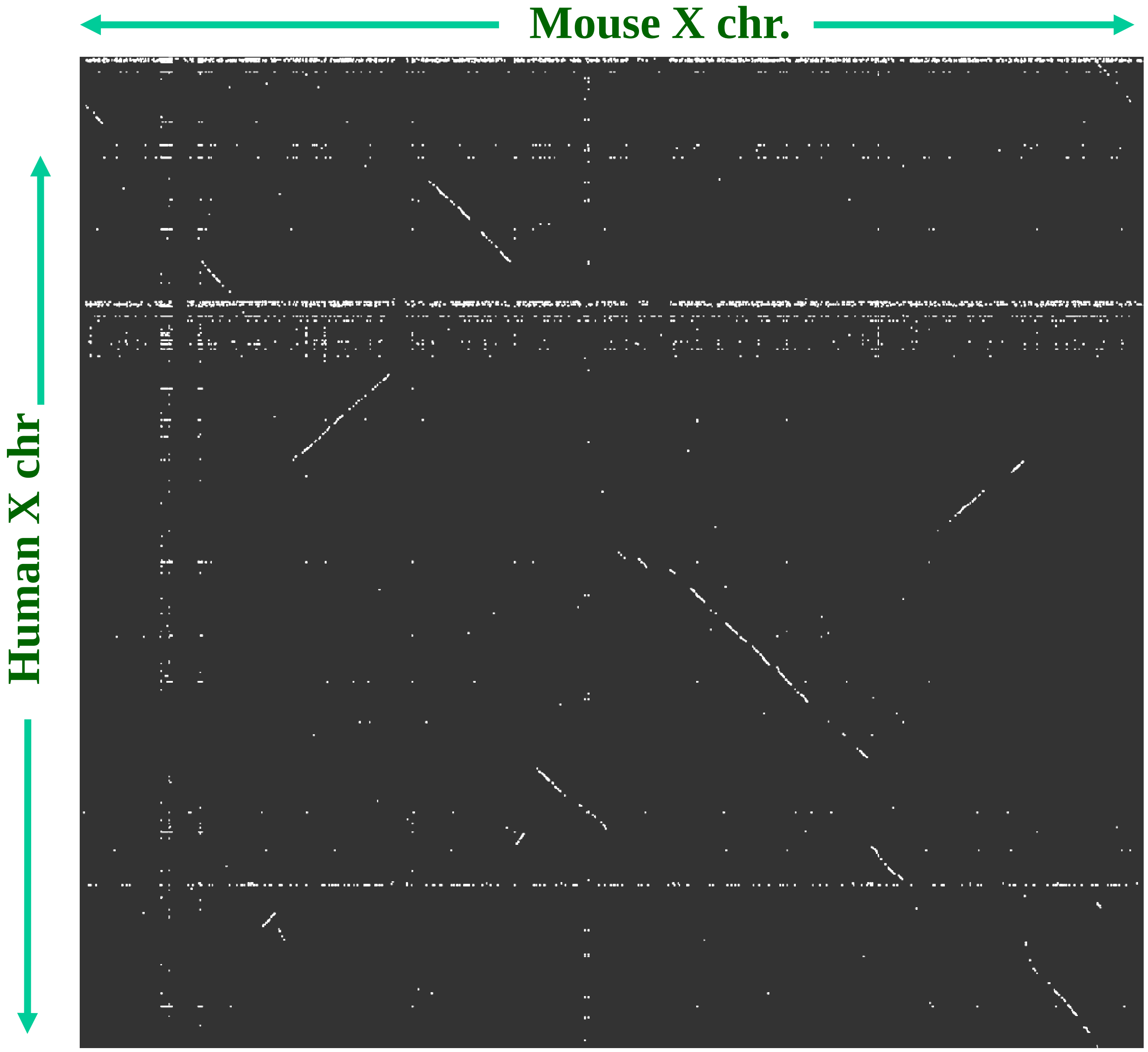
Note:

BLASTZ 2weeks

MURASAKI

2-3 hours with

1% error


Mouse X chr.
Human X chr