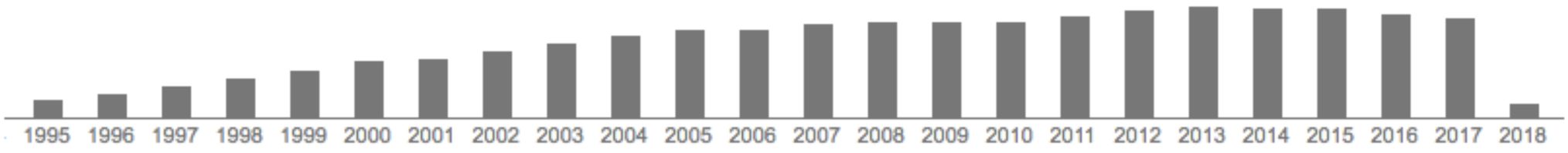
BLAST:
Basic Local Alignment Search Tool

Altschul et al. J. Mol Bio. 1990.

One of the highest cited papers in history

Basic local alignment search tool

Authors	Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, David J Lipman
Publication date	1990/10/5
Journal	Journal of molecular biology
Volume	215
Issue	3
Pages	403-410
Publisher	Elsevier Science
Description	A new approach to rapid sequence comparison, basic local alignment search tool (BLAST), directly approximates alignments that optimize a measure of local similarity, the maximal segment pair (MSP) score. Recent mathematical results on the stochastic properties of MSP scores allow an analysis of the performance of this method as well as the statistical significance of alignments it generates. The basic algorithm is simple and robust; it can be implemented in a number of ways and applied in a variety of contexts including straight-forward DNA and protein sequence database searches, motif searches, gene identification searches, and in the analysis of multiple regions of similarity in long DNA sequences. In addition to its flexibility and tractability to mathematical analysis, BLAST is an order of magnitude faster than existing sequence comparison tools of comparable sensitivity.
Total citations	Cited by 127612

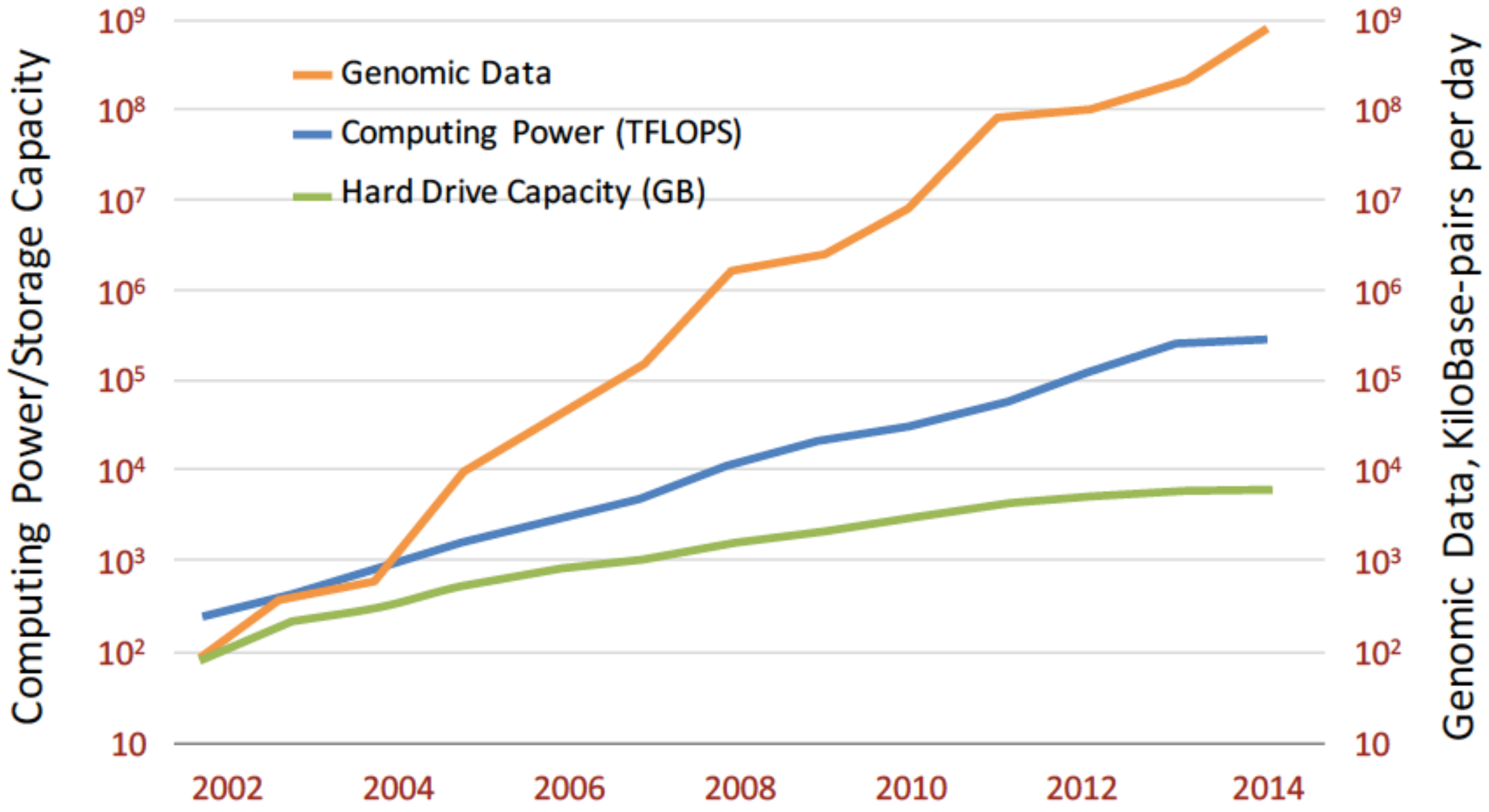


Basic **L**ocal **A**lignment **S**earch **T**ool

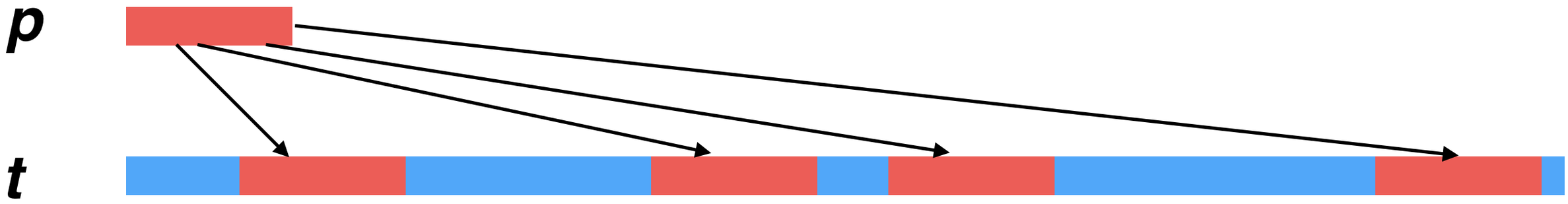
- Rapidly compare a sequence Q to a database to find all sequences in the database with an score above some cutoff S.
  - Which protein is most similar to a newly sequenced one?
  - Where does this sequence of DNA originate?

- Speed achieved by using a procedure that typically finds "most" matches with scores > S.
  - Tradeoff between sensitivity and specificity/speed
    - Sensitivity – ability to find all related sequences
    - Specificity – ability to reject unrelated sequences

Why is database search difficult?



Consider a simpler problem



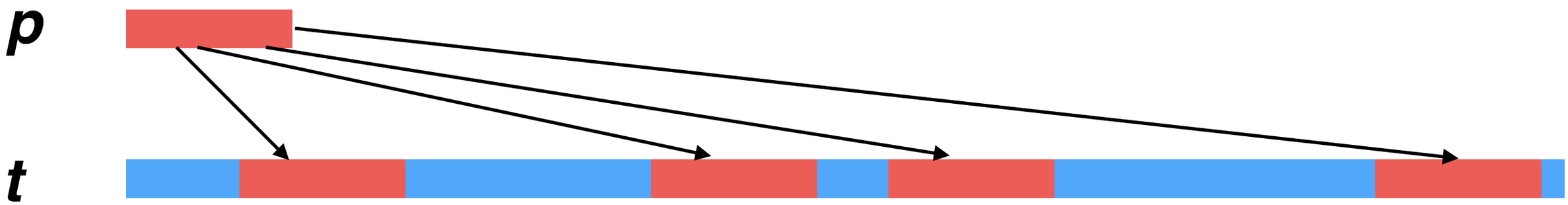
Goal: *Find all occurrences of a pattern in a text*

Input: Pattern $\boldsymbol{p} = p_1 \ldots p_n$ and text $\boldsymbol{t} = t_1 \ldots t_m$

Output: All positions $1 \leq i \leq (m - n + 1)$ such that the n-letter substring of $\boldsymbol{t}$ starting at i matches $\boldsymbol{p}$

Motivation: Searching database for a known pattern

Further simplified version

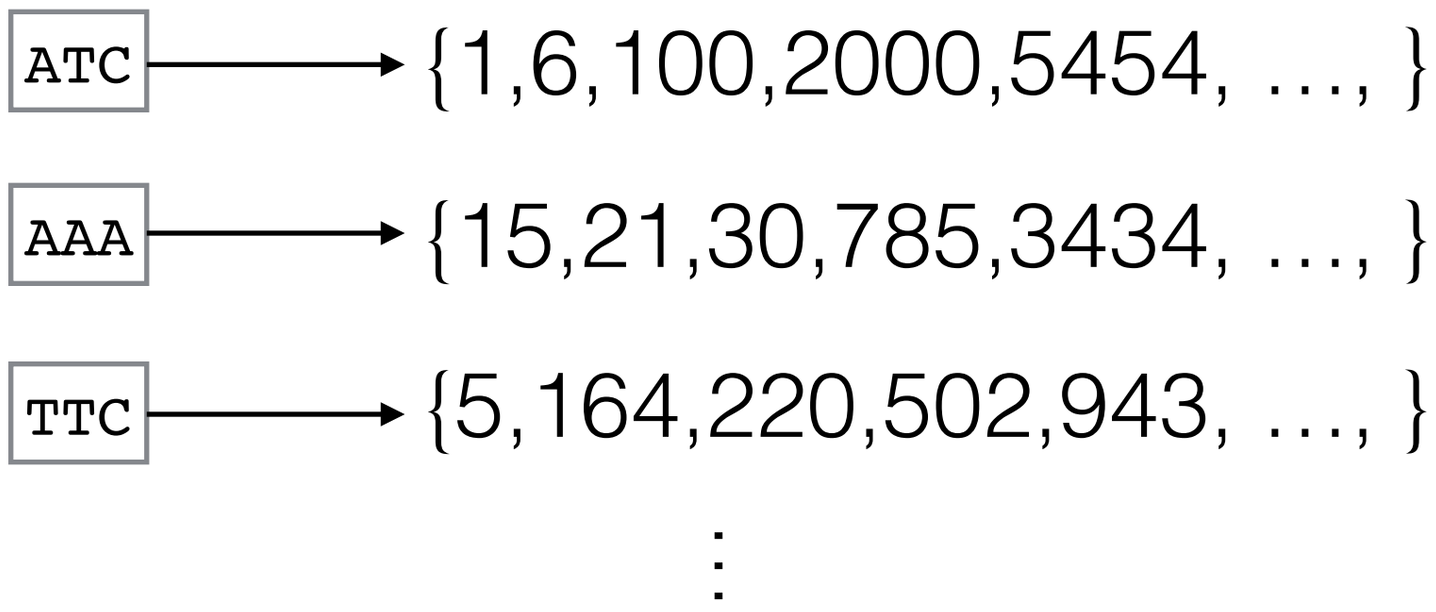


p = **ATC** or **AAA** or **TTC** or …

Text = Human genome (3 billion basepairs)
Pattern = a 3-letter word
Output: All positions $1 \leq i \leq (m - n + 1)$ such that the n-letter substring of ***t*** starting at i matches ***p***

Key idea: preprocessing

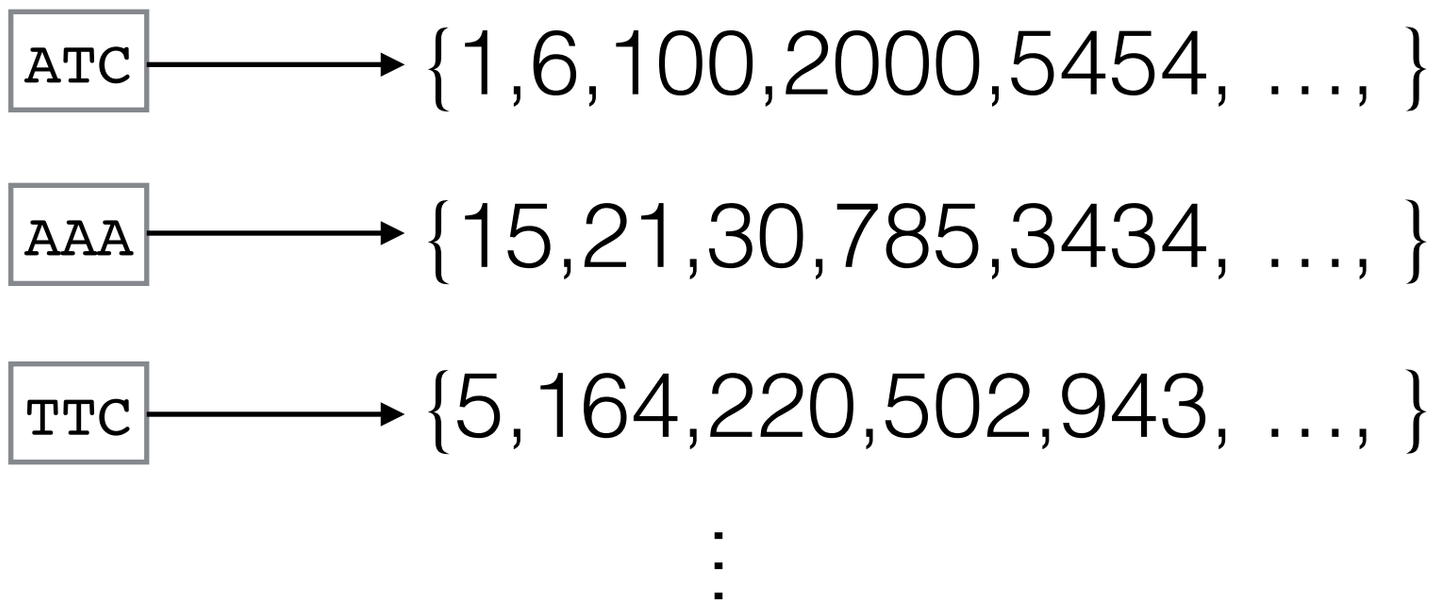
Preprocessing:
store exact matches of all short patterns on the text



After preprocessing, a large part of the computation is already finished before we search for similarities.
For biological sequences of length n there are 4^n (for the DNA-alphabet $\Sigma = \{A, G, C, T\}$) and/or 20^n (for the amino acid alphabet) different strings.
For small $|\Sigma|$ and n, it is possible to store all in a table.

Key idea: preprocessing

Preprocessing:
store exact matches of all short patterns on the text

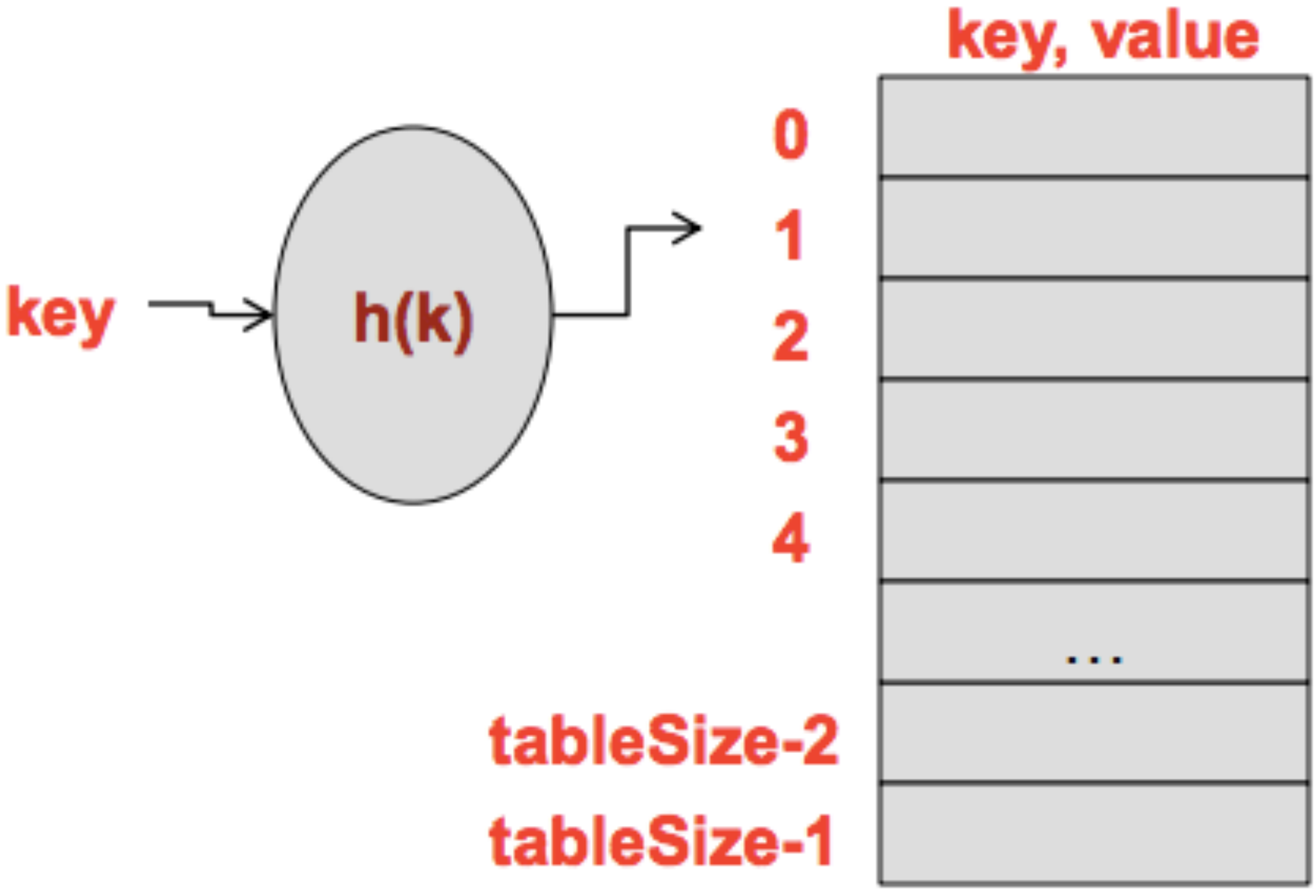


After preprocessing, a large part of the computation is already finished before we search for similarities.
For biological sequences of length n there are 4^n (for the DNA-alphabet $\Sigma = \{A, G, C, T\}$) and/or 20^n (for the amino acid alphabet) different strings.
For small $|\Sigma|$ and n, it is possible to store all in a table.

what if n is big?

Hashing

A hash function maps a key to a value



Hash table

- Hash table is a data structure: a way to store key-value pairs, and a way to retrieve them

- Based on the idea of a hash function. This maps a key or an object (e.g., a string, or a more complex record) to an integer, the “address”

- The value of the key is then stored at that address in memory

Hashing: an example

- Key: (AAACGTAT, 1234321)
  - i.e., a 8 bp-string and its location in genome
- We want to store many such strings and their locations
  - and later retrieve all locations of a particular string really quickly
- Hash function h(AAACGTAT) = 435

Hashing: an example

- Let's assume that there are 4^8 = 64K memory locations available.

- The first time we see (AAACGTAT, *), we store it at address h(AAACGTAT) = 435.

- The next time we see (AAACGTAT, *), we compute h(AAACGTAT), go to 435, find it already occupied. A collision!

How to handle collisions

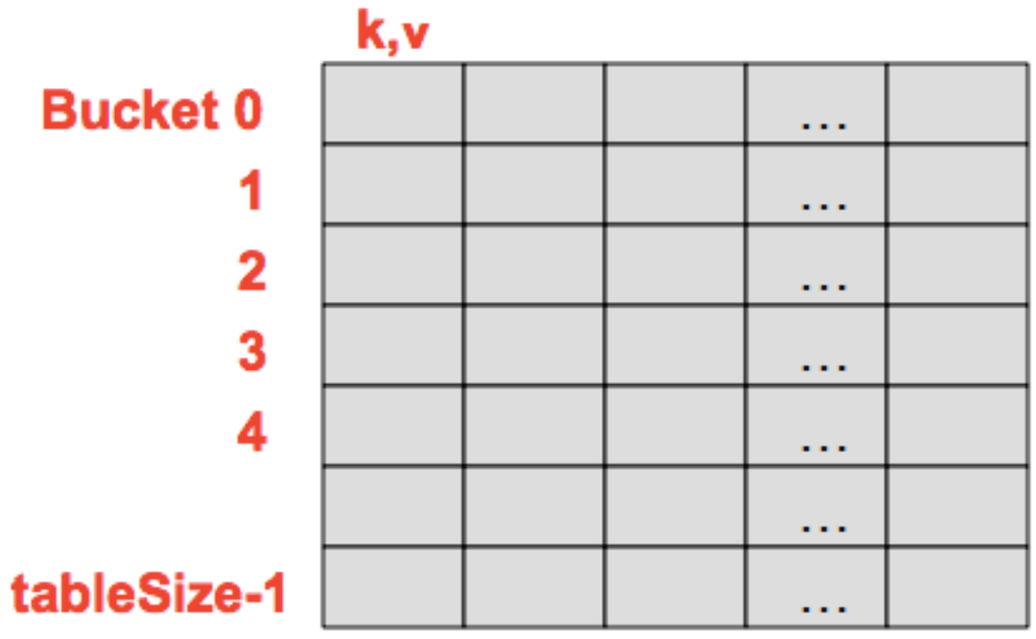
- Buckets: Address 435 can store multiple keys/ objects (e.g., as a linked list)

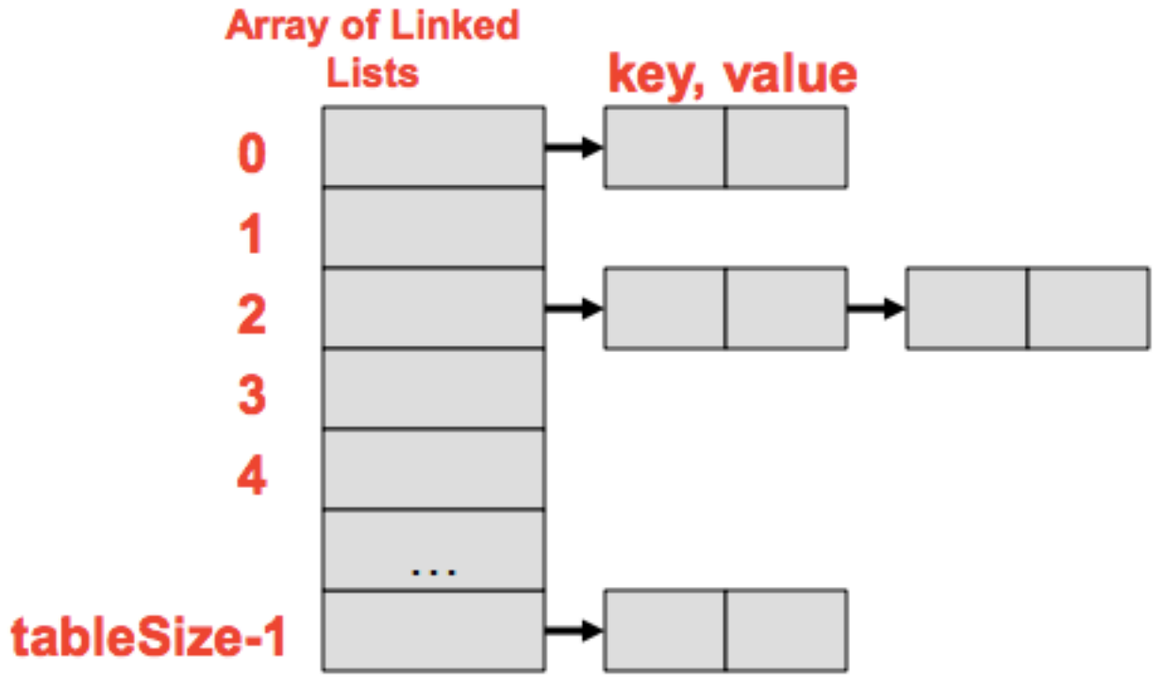
- Linear probing: If an address is occupied, store the key/object in next available location

- Multiple hashing: have an army of hash functions. If the first one (“h”) led to a collision, try another hash function (“h2”)

Bucketing and Chaining

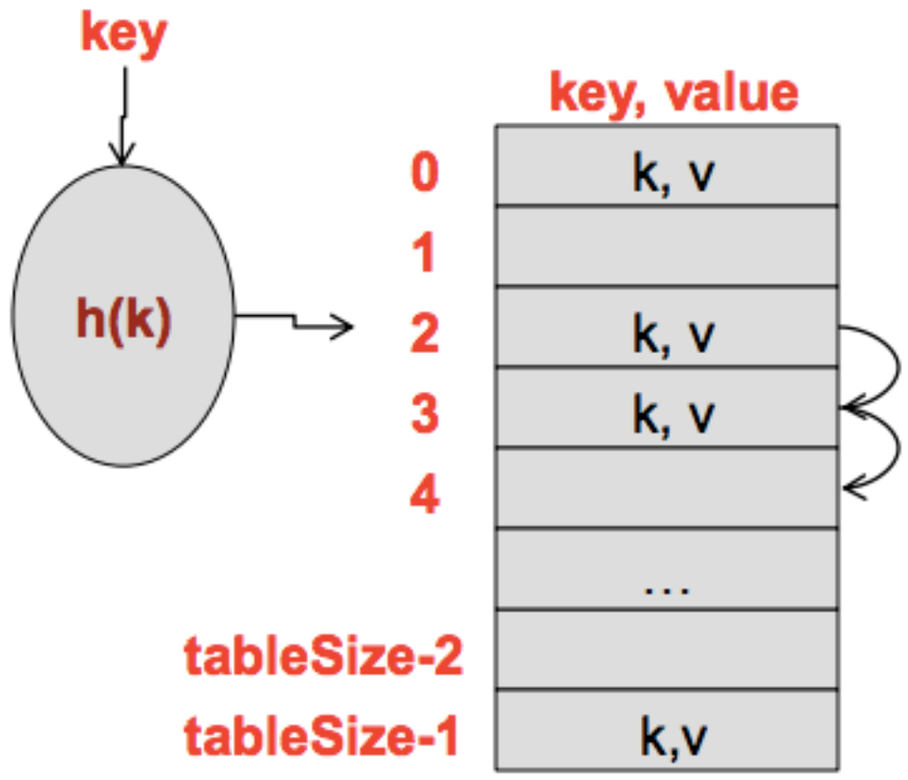
- Rather than searching for a free entry, make each entry in the table an ARRAY (bucket) or LINKED LIST (chain) of items/entries
- Buckets
  - How big should you make each array?
  - Too much wasted space
- Chaining
  - Each entry is a linked List





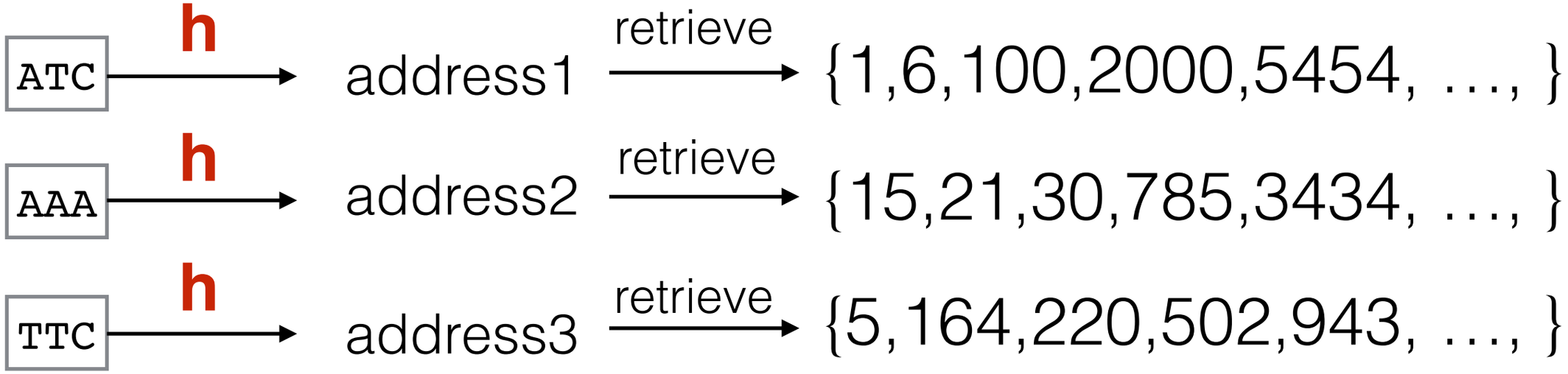
Open addressing and linear probing

- Open addressing means an item with key, k, may not be located at h(k)
- Assume, location 2 is occupied with another item
- If a new item hashes to location 2, we need to find another location to store it
- Linear Probing
  - Just move on to location h(k)+1, h(k)+2, h(k)+3,…



Preprocessing and hash

Preprocessing:
store exact matches of all short patterns on the text by a hash table



BLAST: finding maximal segment pairs

- Given two sequences of same length, the similarity score of their alignment (without gaps) is the sum of similarity values for each pair of aligned residues

- Maximal segment pair (MSP): Highest scoring pair of identical length segments from the two sequences being compared (“query” and “subject”)

- The similarity score of an MSP is called the MSP score

- BLAST heuristically **aims** to find them

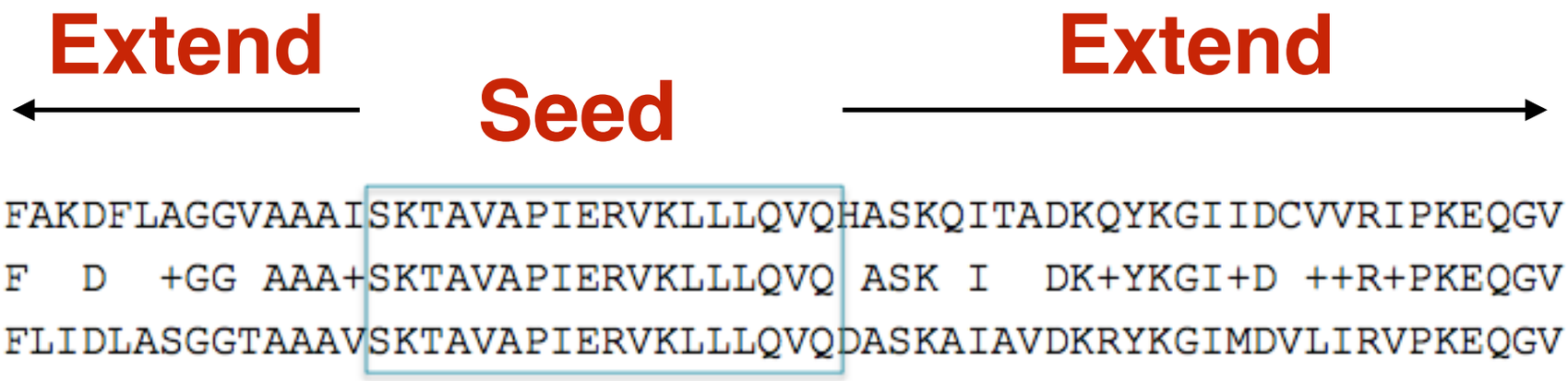
Maximal segment pairs and High scoring pairs

```
Query: HBA_HUMAN Hemoglobin alpha subunit
Sbjct: SPAC869.02c [Schizosaccharomyces pombe]

 Score = 33.1 bits (74),  Expect = 0.24
 Identities = 27/95 (28%), Positives = 50/95 (52%), Gaps = 10/95 (10%)

Query   30   ERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAH   89
             ++M  ++P      P+F+ +H  +     + +A AL N   ++DD+  +LSA  D
Sbjct   59   QKMLGNYPEV---LPYFNKAHQISL--SQPRILAFALLNYAKNIDDL-TSLSAFMDQIVV  112

Query   90   K---LRVDPVNFKLLSHCLLVTLAAHLPAEF-TPA  120
             K    L++    ++ ++ HCLL T+   LP++  TPA
Sbjct   113  KHVGLQIKAEHYPIVGHCLLSTMQELLPSDVATPA  147
```

- Goal: report database sequences that have MSP score above some threshold S.
  - Thus, sequences with at least one locally maximal segment pair that scores above S.

A quick way to find MSPs

- **Homologous sequences tend to have very similar or even identical substrings, also called seeds.**
- **From a seed, it is possible to construct a local HSP/MSP by extending to flanking regions.**



Efficient algorithm?