BIOINFORMATICS

How do we locate disease causing mutation?

Marco Beccuti

Università degli Studi di Torino

Dipartimento di Informatica

April 2019

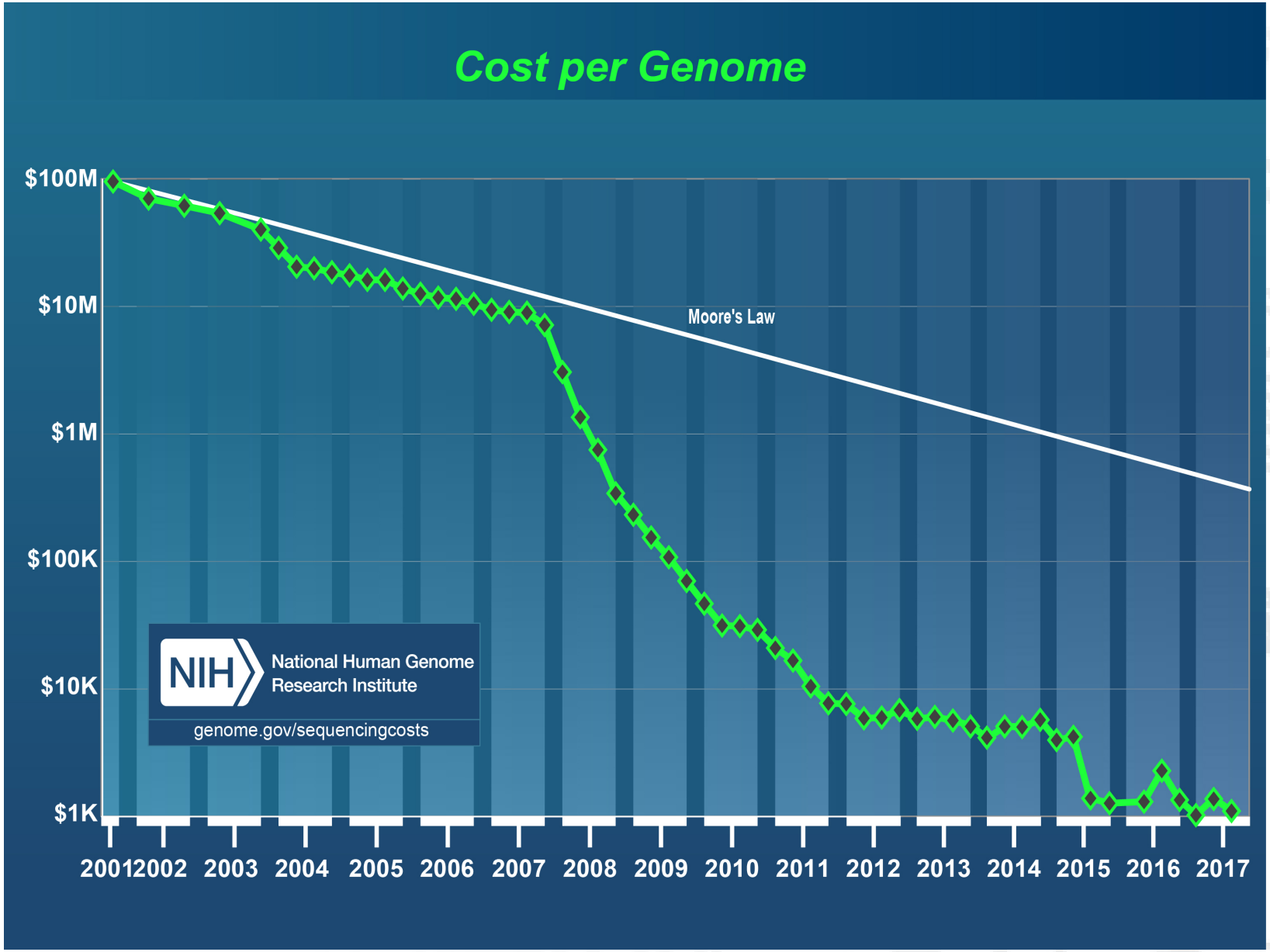
Outline

1. Why do we map reads?
2. How can we map reads?
3. Can we deal with the computational challenge?

Part 1

Why do we map reads?

Sequencing Costs are decreasing exponentially



Sequencing throughput of some current platforms

	iSeq 100 System	MiniSeq System	MiSeq Series	NextSeq Series
Popular Applications & Methods	Key Application	Key Application	Key Application	Key Application
Large Whole-Genome Sequencing (human, plant, animal)				●
Small Whole-Genome Sequencing (microbe, virus)	●	●	●	●
Exome Sequencing				●
Targeted Gene Sequencing (amplicon, gene panel)	●	●	●	●
Whole-Transcriptome Sequencing				●
Gene Expression Profiling with mRNA-Seq				●
Targeted Gene Expression Profiling	●	●	●	
Long-Range Amplicon Sequencing*	●	●	●	
miRNA & Small RNA Analysis	●	●	●	●
DNA-Protein Interaction Analysis			●	●
Methylation Sequencing				●
16S Metagenomic Sequencing		●	●	●
Run Time	9–17.5 hrs	4–24 hours	4–55 hours	12–30 hours
Maximum Output	1.2 Gb	7.5 Gb	15 Gb	120 Gb
Maximum Reads Per Run	4 million	25 million	25 million †	400 million
Maximum Read Length	2 × 150 bp	2 × 150 bp	2 × 300 bp	2 × 150 bp

From Species to Personal Genomes

- Comparison among species:

- Studying single personal genome:

From Species to Personal Genomes

```
CTGATGATGGACTACGCTA
CTACTGCTAGCTGTATTAC
GATCAGCTACCACATCGTA
GCTACGATGCATTAGCAAG
CTATCGATCGATCGATCGA
TTATCTACGATCGATCGAT
CGATCACTATACGAGCTAC
TACGTACGTACGATCGCGG
GACTATTATCGACTACAGA
TAAAACATGCTAGTACAAC
AGTATACATAGCTGCGGGA
TACGATTAGCTAATAGCTG
ACGATATCCGAT
```

```
CTGATGATGGACTACGCTA
CTACTGCTAGCTGTATTAC
GATCAGCTACAACATCGTA
GCTACGATGCATTAGCAAG
CTATCGATCGATCGATCGA
TTATCTACGATCGATCGAT
CGATCACTATACGAGCTAC
TACGTACGTACGATCGCGT
GACTATTATCGACTACAGA
TGAAACATGCTAGTACAAC
AGTATACATAGCTGCGGGA
TACGATTAGCTAATAGCTG
ACGATATCCGAT
```

```
CTGATGATGGACTACGCTA
CTACTGCTAGCTGTATTAC
GATCAGCTACTACATCGTA
GCTACGATGCATTAGCAAG
CTATCGATCGATCGATCGA
TTATCTACGATCGATCGAT
CGATCACTATACGAGCTAC
TACGTACGTACGATCGCGA
GACTATTATCGACTACAGA
TCAAACATGCTAGTACAAC
AGTATACATAGCTGCGGGA
TACGATTAGCTAATAGCTG
ACGATATCCGAT
```

From Species to Personal Genomes

- Nicholas Volcker: first person to have life saved by genome sequencing (2010);
- His intestine had been dangerously inflamed, necessitating a hundred surgeries including the removal of his colon;
- Personal genomics can help us to understand the genetic basis of diseases.

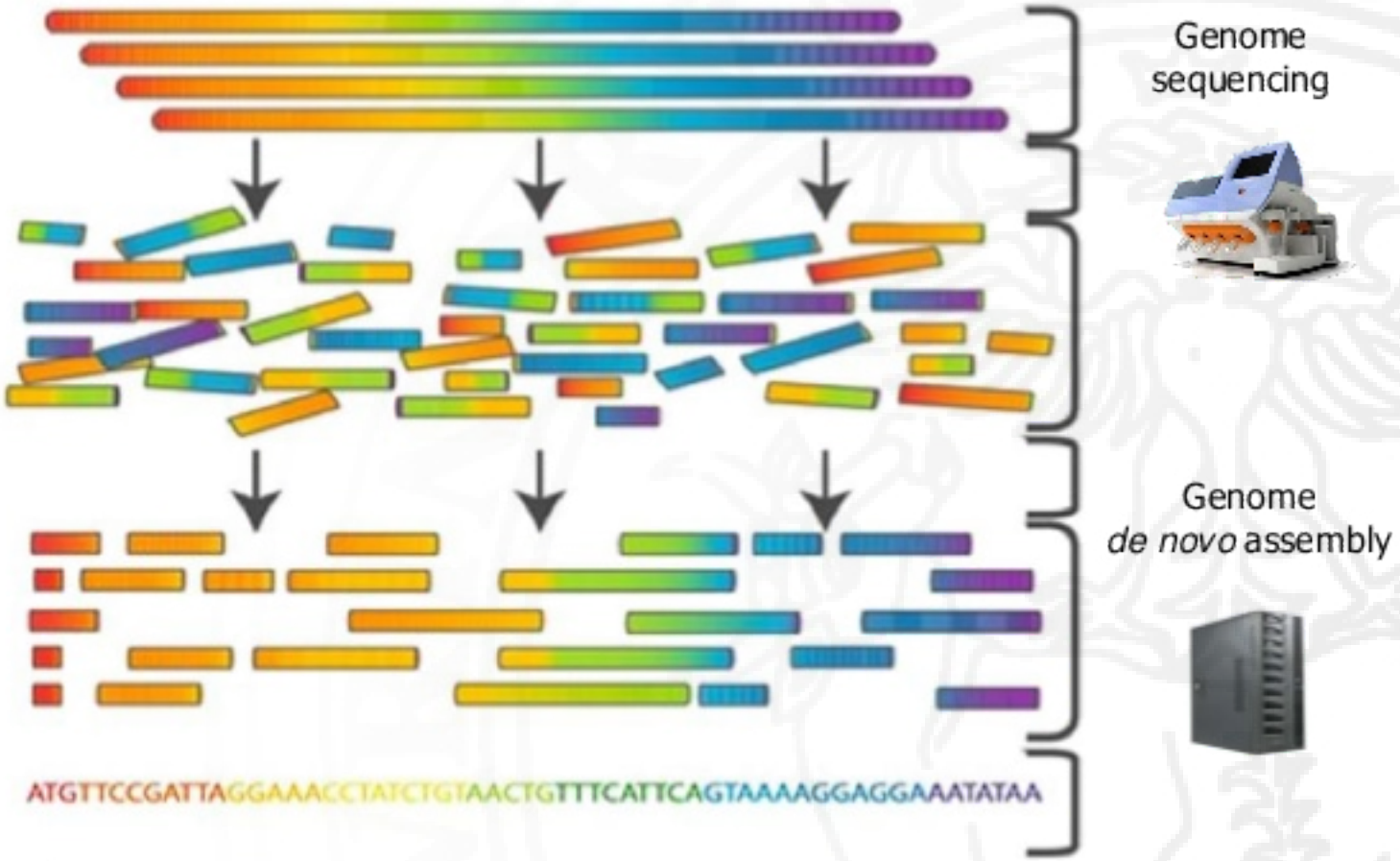
Genomes Meet the Crowd

- Personal Genome Project UK (2013): 100,000 human genomes;
- Ethical issues?

How can we map reads?

Why Not Use Assembly?

- Assembly algorithms (e.g. de Bruijn graph) are too computational expensive;



- **Idea**: use existing structure of reference genome to help us to sequence a patient's genome.

Toward a Computational Problem

- **Reference genome**: database genome used for comparison;
- Question: How can we assemble individual genomes efficiently using a reference?

CTGA**T**GATGGACTACGCTACTACTG**C**TAGCTGT**A**T	Individual
CTGA**G**GATGGACTACGCTACTACTG**A**TAGCTGT**T**T	Reference

Read Mapping

- **Read mapping**: determine where each read has high similarity to the reference genome.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT   Reference
  GAGGA      CCACG          TGA-A     Reads
```

- **local alignment** or **pattern matching** algorithms can be exploited.

Can we deal with the computational challenge?

Can we use local alignment algorithm?

- Recent implementations can calculate 3 billion of node score per second;
- Human genome $\sim$ 3 billion $*$ Read length $\sim$ 250 $*$ Read Number $\sim$ 100 million $\rightarrow \sim 150 * 100$ million of seconds;
- How to deal with this:
  - Parallel computing;
  - Filtering phase before local alignment algorithm;
  - Quasi alignment approaches.

Can we use local alignment algorithm?

- Recent implementations can calculate $\sim$ 3 billion of node score per second;
- Human genome $\sim$ 3 billion $*$ Read length $\sim$ 250 $*$ Read Number $\sim$ 100 million $\rightarrow \sim 150 * 100$ million of seconds;
- How to deal with this:
  - Parallel computing;
  - **Filtering phase before local alignment algorithm;**
  - Quasi alignment approaches.

Read alignment process

- The read alignment process can be decomposed into three steps:
  - to identify the potential areas of similarity between reads and genome using **k-mer** and **hashing** (i.e. we look for exact substring matches);
  - to validate the similarity between read and each potential area exploiting **local alignment**;
  - to evaluate the alignments with similar score in different positions.

Introduction to k-mers

- **Definition:** Given a string S then a k-mer is a substring of S with length k.

ATGCTGCTGATCGTCATGCTGCGCTAGCTAGCTAGCT

k = 7

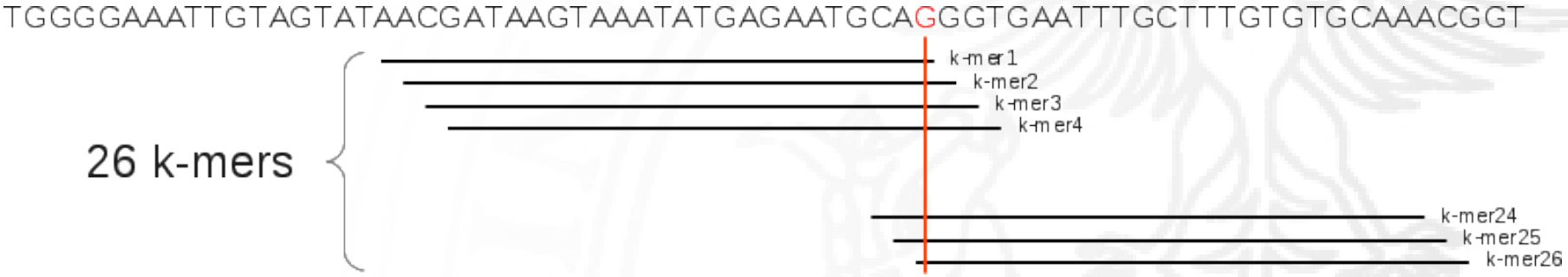
- **k-mer generation:** All the k-mers of a strings can be generated using a sliding window approach.

ATGCTGCTGATCGTCATGCTGCGCTAGCTAGCTAGCT

#k-mer = Read length – k +1 (WORST CASE)

Read alignment process

- Observe that a mismatch will affect *k* k-mers. For instance assuming k=26 we will have:



Introduction to k-mers

- **Associating information with k-mers.**

Different information can be associated with k-mers as:

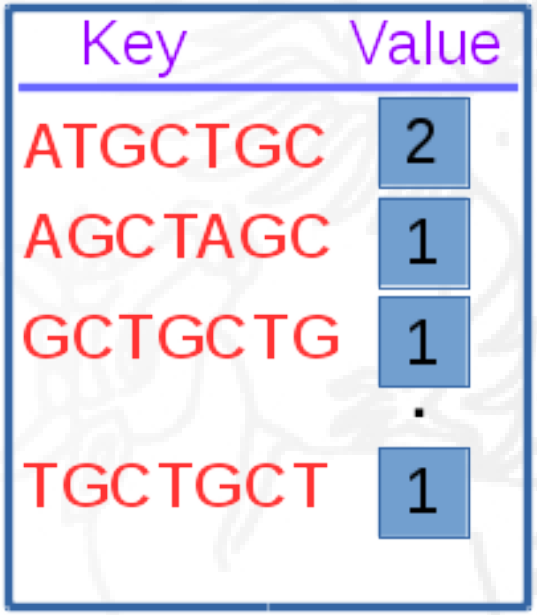
k-mer	Frequency
ATGCTGC	2
TGCTGCT	1
GCTGCTG	1
.....	....
TAGCTAG	1
AGCTAGC	1

k-mer	Position(s)
ATGCTGC	1,16
TGCTGCT	2
GCTGCTG	3
.....	....
TAGCTAG	28
AGCTAGC	29

Associative array

How to efficiently store k-mers and their information: Associative array is abstract data type composed of a collection of $\langle key, value \rangle$ pairs, such that each possible key appears just once in the collection.
Informally, an associative array is an array having an index that is not necessarily an integer, and can be sparsely populated.

k-mer	Frequency
ATGCTGC	2
TGCTGCT	1
GCTGCTG	1
.....	....
TAGCTAG	1
AGCTAGC	1



Associative array

Hash table in a nutshell

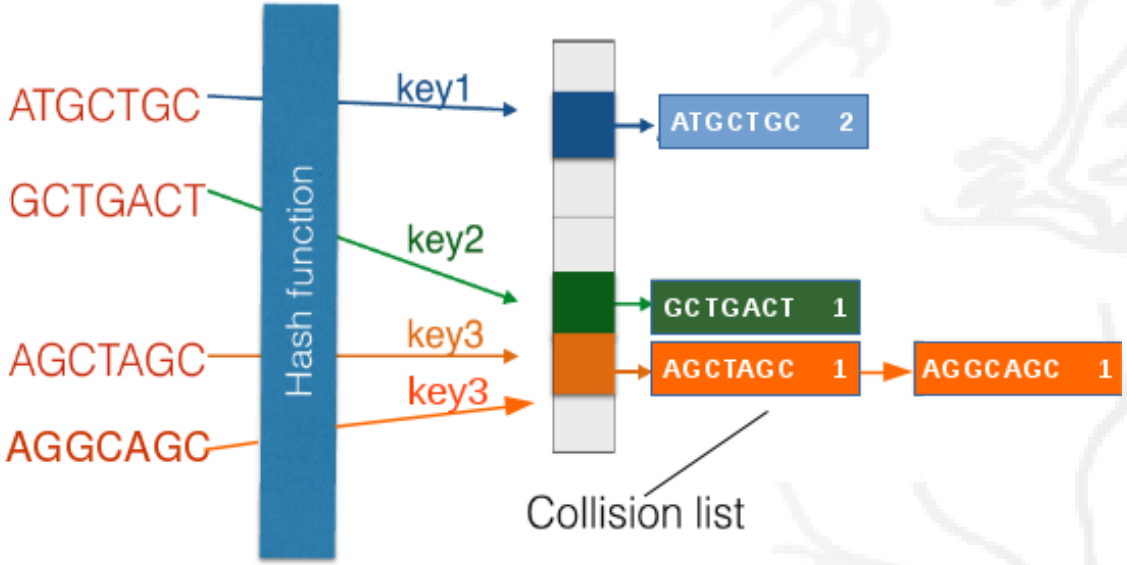
An associative array can be efficiently implemented using **hash table**. Hash table is a good compromise in terms of memory and search cost. It requires:

- **hash function**: is a function used to map data of arbitrary size to a index.

Example (A trivial hash function)

A trivial hash function is $(\sum_{i=1}^{k} f(m[i]) mod n$ *where* $m[i]$ *returns the nucleotide in position* i *in* m*, and* f *is a function mapping a integer value to a nucleotide type.*

- **collision policy**: two or more k-mers can have the same hash value. Collision can be solved using a chaining: all elements with same hash value are stored in linked list.



Hash table in a nutshell

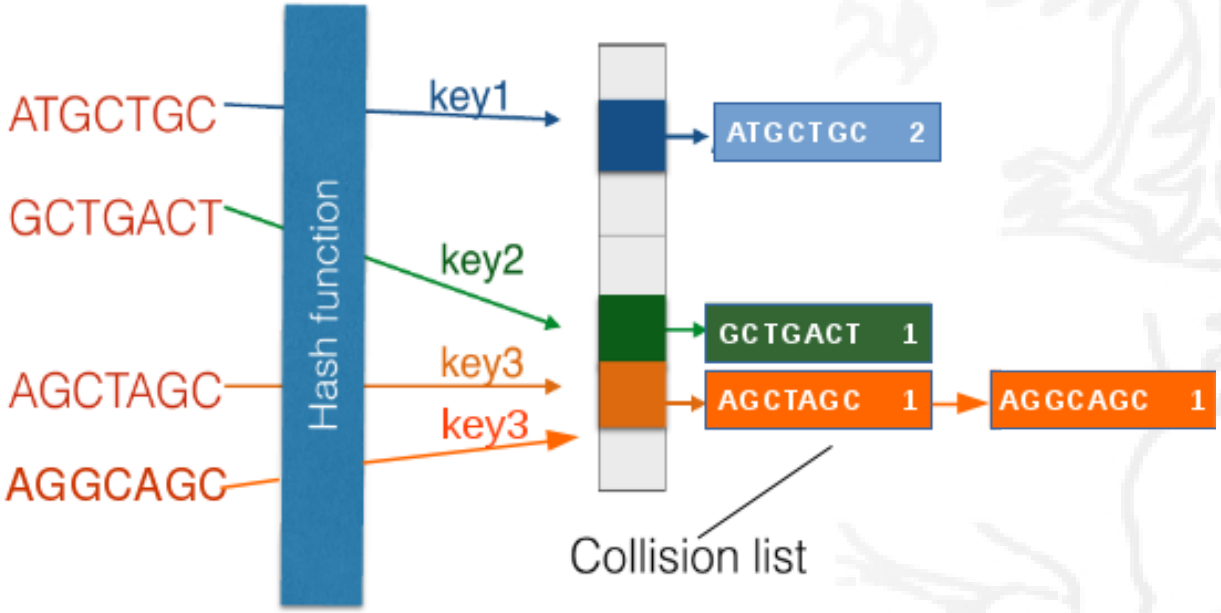
To search a k-mer requires:

- to identify the right collision list computing the hash value for such k-mer;
- to scan the collision list.

It is important to select a hash function which provides uniform distribution of hash values.

To insert a k-mer requires:

- to search the k-mer;
  - if it is in the hash table then only its information are updated;
  - otherwise the k-mer is inserted into the right collision list.



How to deal with multiple mismatches

```
ATCAGCGCAAATGCTCAAGA
ATCAGC
 TCAGCG
  CAGCGC
   AGCGCA
    GCGCAA
     CGCAAA
      GCAAAT
       CAAATG
        AAATGC
         AATGCT
          ATGCTC
           TGCTCA
            GCTCAA
             CTCAAG
              TCAAGA
```

- In this case, assuming $k = 6$ and orange bases mutated then no exact matches are found;
- How to cope with this?

Multiple mismatches and space seed

- **space seed** can be used;
- The idea is to used a special mask to generate k-mer;

$$111 * 1 * 11$$

```
ATCAGCGCAAATGCTCAAGA
ATC G GC
 TCA C CA
  CAG G AA
   AGC C AA
    GCG A AT
     CGC A TG
      GCA A GC
       CAA T CT
        AAA G TC
         AAT C CA
          ATG T AA
           TGC C AG
            GCT A GA
```

Multiple mismatches and space seed

- Different spaced seed can be used to detect different homologies;
- Some spaced seeds can detect more homologies than others;
- A set of spaced seeds can be simultaneously used to hit more homologies.