Four Russians Trick (Arlazarov, Dinic, Kronrod, Faradzev)
- Cover the DP table with t X t 1-overlapping blocks for t = logn / 4, since alphabet size is 4.
- Create alignment lookup for every possible score pair and the sequences for a specific block.
- Filter useless ones & create binary encoding for each posibility based on consecutive relationships between scores: 1 if difference, 0 otherwise. Lookup size: 2^t scores & 4^t strings, so O(n^(3/2)).
- Compute last row & column of every block based on first row & column encodings from the lookup & the sequences for that block.

Bit-parallel Alignment
For edit (*Vladimir Iosifovich Levenshtein*) distance where match=0, mismatch=1, gap=1.
- Encode the DP matrix using 1-word long bit vectors (32 or 64 bit based on the system).,
- Parallelize DP matrix using word-operations and resolve dependencies.

horizontal adjacency property $\Delta h_{i,j} = C_{i,j} - C_{i,j-1} \in \{-1, 0, +1\}$
vertical adjacency property $\Delta v_{i,j} = C_{i,j} - C_{i-1,j} \in \{-1, 0, +1\}$
diagonal property $\Delta d_{i,j} = C_{i,j} - C_{i-1,j-1} \in \{0, +1\}$

The delta vectors are encoded as bit-vectors by the following boolean variables:
- $VP_{ij} \equiv (\Delta v_{i,j} = +1)$, the vertical positive delta vector
- $VN_{ij} \equiv (\Delta v_{i,j} = -1)$, the vertical negative delta vector
- $HP_{ij} \equiv (\Delta h_{i,j} = +1)$, the horizontal positive delta vector
- $HN_{ij} \equiv (\Delta h_{i,j} = -1)$, the horizontal negative delta vector
- $D0_{ij} \equiv (\Delta d_{i,j} = 0)$, the diagonal zero delta vector

Encode the DP matrix C

The deltas and bits are defined such that

$$\Delta h_{i,j} = HP_{i,j} - HN_{i,j}$$
$$\Delta v_{i,j} = VP_{i,j} - VN_{i,j}$$
$$\Delta d_{i,j} = 1 - D0_{ij}$$

These values "encode" the entire DP matrix C[0..m, 0..n] by $C(i, j) = \sum_{r=1}^{i} \Delta v_{r,j}$

		A	N	N	E	A	L	I	N	G
	0	0	0	0	0	0	0	0	0	0
A	1	0	1	1	1	0	1	1	1	1
N	2	1	0	1	2	1	1	2	2	2
N	3	2	1	0	1	2	2	2	2	2
U	4	3	2	1	1	2	3	3	3	3
A	5	4	3	2	2	1	2	3	4	4
L	6	5	4	3	3	2	1	2	3	4

		A	N	N	E	A	L	I	N	G
	0	0	0	0	0	0	0	0	0	0
A	1	0	1	1	1	0	1	1	1	1
N	1	1	-1	0	1	1	0	1	1	1
N	1	1	1	-1	-1	1	1	0	0	0
U	1	1	1	1	0	0	1	1	1	1
A	1	1	1	1	1	-1	-1	0	1	1
L	1	1	1	1	1	1	-1	-1	-1	0

1. Instead of computing C we compute the Δ values, which in turn are represented as bit-vectors.
2. We compute the matrix column by column.
3. We maintain the value $score_j$ using the fact that $score_0 = m$ and $score_j = score_{j-1} + \Delta h_{m,j}$.

Multiple Sequence Alignment (MSA)
If classic alignment extended to k-strings, traverse n-dim space, O(2^k * n^k).
- p(y,j): Frequency of char y occuring in col j of profile C
- S(x,j): score of aligning x with col j of profile C
*** Sum over all char y in column j, score function(x,y) * p(y,j)
- V(0,j): Sum over col k <= j, S('-',k)
- V(i,0): Sum over all row k <= i, score func(kth char of sequence s, '-')

$$V(i,j) = max \begin{cases} V[i-1,j-1] + S(S_1(i), j) \\ V[i-1,j] + \delta(S_1(i), '-') \\ V[i,j-1] + S('-', j) \end{cases}$$

Alignment Strategies
Greedy: Iteratively align the most similar pair & expand the profile with seq having highest score w.r.t. the profile.
Progressive & ClustalW: Construct pairwise alignments, build phylogenetic tree using neighbourhood joining algorithm, progressive alignment guided by the tree.

Center Star: Choose the seq i having largest pairwise alignment score sum as center star, iteratively build alignment with the current best pairwise alignment with center star. Update the profile after every step.
Partial order alignment: Directed, acyclic graph (DAG) G where you have parents from waay before or all over the DP table. When computing alignment, do topological sort over the graph & visit the nodes in this order when computing DP entries.

Alignment Quality
- Sum of pairwise scores imposed by the MSA
- Entropy: sum for every column,
Sum over every char in column j, -1 * frequency of char c * log(frequency of char c)
If pure, entropy = 0, if all unique, it's 2.
- Number of matches, longest common subsequence of MSA

Phylogenetic Trees
- If distance Dij = tree distance between i & j, D is fitting.
- For 3 sequences, triangular shape with a node in center.
- Expending to n sequences gives (n chooses 2) equations with 2n-3 variables, not always possible to solve optimally when n > 3.

UPGMA
Ultrametric tree: distance from root to any leaf is the same.
- Compute distance between clusters as average pairwise distance between cluster elements.
- Merge the clusters closest by removing the clusters from the table and creating a new cluster at distance / 2.
- Update the distance between remaining clusters and the new cluster.

Neighbour Joining
Distance matrix D is *additive* if node distances in tree equal to the distance in D.
Degenerate triplets: Dij + Djk = Dik
- If any, remove j & reduce problem size
Four-Point Condition,
Dij + Dkl, Dik + Djl, Dil + Djk
Two are the same & one is less
If this is true for all triplets, additive

Least-squares distance phylogeny problem: Finding best appropriation tree T for non-additive matrix D, which is NP-hard.
Max Parsimony: Determine what chars at internal nodes would best explain character strings for n observed species.
Parsimony score: Sum of cost of all mutations found in the tree.

Small parsimony problem: Given a tree with each leaf labeled m-char string, find the tree minimizing the parsimony score.
Weighted small parsimony problem: Includes a scoring matrix to show penalty for mutations between chars. Otherwise, use Hamming distance.
Sankoff's Algorithm
s_t,parent = min_i{s_i(left_child) + penalty_i,t)} + min_j{s_j(right_child) + penalty_j,t)}

- Scores at the root are computed by going up. After the lowest scored char selected for root, backtrack and assign chars contributing to the character of a parent from the children & assign it to the child.
- Remember that left & right children parsimony scores are computed separately.

Fitch's Algorithm: Assign a letter set to each vertex bottom to top. If intersection of children' sets is empty, take their union. Otherwise, use the intersection. Afterwards, arbitrarily assign a letter to the root from its set, go down the tree. If the letter of the root is in the node's label set, select it. Otherwise, choose arbitrarily.
- Sankoff & Fitch both O(nk) & identical if Sankoff uses Hamming distance.

Large Parsimony Problem: Problem of constructing a tree T with minimum parsimony score. (2n-3)! rooted & (2n-5)! unrooted possible trees. Problem is *NP-complete*. Branch-and-bound or heuristics for solution.

Similarity Search
Dot matrix – character-equalities shown with marks, empty otherwise. Afterwards, find the longest diagonal w/o mismatches & gaps.
- Approximate Pattern Matching: Find all approximate occurrences of a pattern in a text, runs in quadratic
- Query Matching Problem: Match substrings in a query to a text with at most k mismatches.

Instead of finding approximate matches, find short exact matches and expand along the diagonals instead. Assume k-mismatch problem. Split the pattern P into (k+1) regions of short l-mers. By *pigeonhole principle*, one region has to perfectly match the text & this is the basis of how longer matches can be found. Although the longer K gets the less performance you get due to computational overhead.

FAST-P
- Find short good k-mer matches
- Determine good k-mer matches & their endpoints
- Use DP to align the good matches
FAST-P extract 5 best diagonals and scores them using PAM250, w/o considering indels.

FAST-A: Instead of 5, choose 10 best diagonals. Eliminate the diagonals below a threshold before chaining.
BLAST: Search for short local alignments between k-mers of the pattern and the text, extend locally around the alignment. Original BLAST doesn't allow gaps, gapped BLAST does & usually finds better alignments at the cost of searching a larger local extension space.

FAST-A Variations: TFASTAX, TFASTAY, FASTAX, FASTAY
BLAST Variations: blastn, blastp, blastx, tblastn, tblastx, PSI-BLAST, Megablast, WU-BLAST (Wash U BLAST)

Minimap2
Indexing all k-mers for query matching is expensive since O(4^k) k-mers for DNA. Instead, use minimizer k-mers: hash only certain k-mers to reduce memory usage.
- Minimizer-based binning fragments the query into a few pieces. Based on sequence continuity some k-mer will occur in many long segments which can be used for binning.
- Each k-mer has a unique k-mer minimizer & will always to go the same bin.

Algorithm
- Collect minimizers from the text & create a hash table.
- For each query in database collect its minimizers and find exact matches in the text to be used as anchors.
- Apply *chaining* to merge overlapping (co-linear) anchors.
- Use DP to extend the ends of the chains & close the regions between the chains.

One-Dimensional Chaining
Endpoints of every interval sorted left to right. Process each point in sequence:
- If left end, max value of obtainable by this segment is current max + value of the interval.
- If right end, max = max of (cur_max, max value of obtainable by this segment)
Backtrack to retrieve the interval & cur_max reports the max value obtainable at the end.

Two-Dimensional Chaining
Assume you have 2-d local alignments. Get left & right ends of each rectangle and sort similar to 1-d case. Also, in a separate list get lowest y coordinate, value of the local alignment & the segment, sort w.r.t. lowest y-coordinate. Process x-list in ascending sorted order:

- If left-end, let this rectangle be k. Find the triplet for rectangle j that has the smallest y value which is greater than the largest y value of k. (First retangle to the left and above of k). Then, max possible value obtainable by k is V(k) = v(k) + V(j).
- If right-end, search for the first j that has highest y value less than y value of current rectangle k. Then, delete rectangles having largest y value less than or equal to this j (since you need to go low-right direction).

Maximal Unique Matches (MUMs): The longest substring that occurs in both texts exacly once. Has linear solution with generalized suffix trees.
- Extract MUMs in both genomes
- Sort matches in MUM aligment & extract the longest set of matches that occur in the same order in both genomes using *longest increasing subsequence* problem.
- Close the gaps using local alignments around the ends

- Compared to FASTA, MUM-based alignment is much less noisy.
- <u>MUMs can also be used for MSA:</u> iteratively split the sequences into smaller chunks until MSA becomes feasible & merge the sequences.

Maximal Exact Matches (MEMs): Exact matches that cannot be extended in either direction without breaking the equality. Relaxation over MUMs since uniqueness isn't a requirement.
<u>Burrows-Wheeler Alignment MEMs (BWA-MEM):</u> BWT with MEMs
- Based on Super-MEMs, MEMs not contained in other MEMs.
- Each query results in longest match covering the target position
- Apply greedy chaining while querying & extend using banded affine-gap alignment

K-mer structures
Genome assembly, error correction in sequencing, detecting similarity & distance between sequences and datasets, alignments & pseudoalignments, etc. *~10^10* k-mers for average sequence.
- Supports construction, membership, and iteration operations
- Tries: O(k) all, if depth is O(k)

Bloom Filters: A long bit array with multiple hash functions.
- <u>Insertion:</u> set all positions given by hash functions to 1
- <u>Query/membership:</u> Check if all positions given by the hash functions are 1
*O(#hash functions * k-mer size k) for insertion, deletion, & query*
- <u>Need an estimate for # k-mers n</u>

Possible false positives, no false negatives.
Percentage of FP: **(1 – e^-(d * n / m))^d**
m: length of bit array, d: # hash functions, n: estimated # k-mers
<u>With a bloom-filter, do 2 forward passes</u>. If a k-mer is seen at least twice, put it in the hash table. Second pass detects and removes Fps.

<u>Counting Quotient Filter:</u> Hybrid hash table + bloom filter for approximate membership. Hash table of bloom filters. O(k) operations, k: length of k-mer.
<u>Bloom Filter Tries:</u> Cut k-mers into chunks & insert them into a burst trie. Bloom filter for speed. During insertion, if partial overlap, burst & create a new child. For applications in pan-genomics where a k-mer belongs to many sets.
<u>De Bruijn Graphs:</u> n-dimensional directed graph with m-symbols, m^n vertices consisting of all length-n sequences of m symbols. An edge between two nodes u,v if v can be generated by shifting u by one & inserting a new char.

For DNA assembly, 4^k possible vertices but in reality this size bounded by genome size. Application is short sequence readers: Euler, ALLPATHS-LG, Velvet, AbySS, SOAPdenovo.
- Divide the reads into k-mers & build the graph based on these. Put an edge if k-1 base-pair prefix-suffix matches between two k-mers.

- Apply error correction & use Eulerian-like paths to construction.
- Graph construction & error correction is common to all variations. Corrections based on assumption that less frequent k-mers are caused by errors, so these paths can be removed.

Capillary(Sanger) Sequencing

Sanger (1977) and Gilbert (1977) methods.

Sequencing by Hybridization (SBH): 1988 first occurrence, 1991 Light directed polymer synthesis by Steve Fodor, 1994 first 64-kb DNA microarray by Affymetrix.

- Attach all possible DNA probes of length l to a flat surface, each probe at a distinct and known location. This is the DNA array.
- Apply a solution containing fluorescently labeled DNA fragment to the array.
- The DNA fragment hybridizes with those probes that are complementary to substrings of length l of the fragment.

Using a spectroscopic detector, determine which probes hybridize to the DNA fragment to obtain the l–mer composition.

l–mer composition: Spectrum(s,l) is an unorder multiset of (n-l+1) l-mers of string s of length n. Set, so order doesn't matter. Do a Hamiltonian path (each vertex visited once) to construct predecessor relationships. Or do Eulerian path (each edge visited once) based on 1 shift & 1 insert (De Bruijn) difference.

- Fidelity of Hybridization: difficult to detect small errors
- Array Size: Fidelity can be addressed with longer l-mers, but array size increases exponentially in l, which is limited by technology.
- Not practical, but spearheaded further techniques.

Gel electrophoresis: Start at primer (restriction Site), Grow DNA chain, Include dideoxynucleotide (modified a, c, g, t), Stops reaction at all possible points, separate products with length using gel electrophoresis, challenging to read due to signal errors.

PHRED & PHRAP (by Phil Green): Do signal operations to filter the output, apply DP for PHRED; use genome assembler in addition for PHRAP.

Capillary sequencing can do ~1000 letters at a time

- Double-barrelled sequencing: Circular shape that reads both ends separated by a known separator sequence, outputs error probabilities as ASCII char values as -10log_10 (probability of error).
- About > 7*fold overlap redundancy, 10 is enough to reduce error to 1 in every 1,000,000-base

Sanger sequencing analysis: long reads(1000 base-pairs), low error(<0.1%), cloning libraries applicable, BUT expensive & cloning is hard and time consuming.

High-Throughput Sequencing (HTS)

Random cuts with various sizes, paired-end sequencing and read-mapping to find appropriate place for the piecec based on reference genome (HGP)

- Pyrosequencing (by 454 Life Sciences): The first, acquired by Roche
- Illumina (Solexa): Sequencing by synthesis. GAIIx, HiSeq2000-2500-3000-4000, X Ten, NextSeq, MiSeq, NovaSeq, NovaSeq2
- SOLiD (color-space reads) and Ion Torrent (by Applied Biosystems)
- PacBio (by Pacific Biosciences)
- ONT (by Oxford Nanopore)

- Short sequence reads
  - –~500 bp: 454 (Roche)
  - – 100 – 150 bp Solexa(Illumina), SOLiD(AB)
- Longer
  - •PacBio: 8-20 Kb
  - •ONT: 10-100 Kb
- •Huge amount of sequence per run
  - –Gigabases per run (4 Tbp for Illumina/HiSeq4000)
- Huge number of reads per run
  - •Up to billions
- Bias against high and low GC content (Illumina and Ion Torrent)
  - •GC% = (G + C) / (G + C + A + T)
- Higher error (compared with Sanger)
  - –Different error profiles
  - –10% PacBio, 1% PacBio CCS (HiFi), 5% ONT

Application Areas: Genome re-sequencing: somatic mutation detection, organismal SNP discovery, mutational profiling, structural variation discovery, de novo assembly (constructing genomes from DNA fragments, with no a prior knowledge of sequence or order of the fragments), DNA-protein interaction analysis (ChiP-Seq), novel transcript discovery, quantification of gene expression, epigenetic analysis (methylation profiling)

Fundamental challanges: Interpreting machine readouts - base calling, base error estimation; data visualization, storage and data management for large genomes; SNP, indel, and structural variation discovery; de novo assembly

Illumina (Solexa): Current market leader, Based on sequencing by synthesis, Current read 100-150bp up to 300 with errors, paired end reading, ~0.1% error from mismatches, 6 Tera base-pair reads in a single 2-day run, cheapest so far

Ion Torrent: Sequencing on a microprocessor that measuring pH level w.r.t. bases, indel & homopolymers dominated ~1% error, matepair sequencing possible but difficult

PacBio: 'Third generation' single molecule real time sequencing (SMRT), no replication with PCR, labeled phosphates watching artificial slow DNA polymerase, long reads ~ 10-80 Kbase, indel-dominated ~10% error

Any enzyme reads 70 Kbase median sequence, CLR: single read,CCS: circular reads of 5-6 times renamed HiFi based on MSA for error correction, 10 Kbase median with ~1% error

Nanopore Sequencing: 2 Mega base-pair reads, ~5% indel dominated error, recurrent net to guess the bases, works in real-time, 100 Kbase with ~20% error reduced to 5% nowadays (MinION, SmidgION, PromethION)

HTS Challenges

- Data is large, compression.
- Read mapping – finding segment location
- Variations discovery
- De novo assembly, especially if high error

Compression

Reference based: Embedding rather than compression, high compression rate, fast to encode slow to decode, needs reference genome which can be infeasible & needs mapping, lossy. CRAMtools, SlimGene, etc.

Reference-free: Less compression, no reference, slow to compress but faster to decompress, lossy or lossless. gzip, bzip2, 7zip etc. Specialized FASTQ compressors like CALCE, ReCoil, G-SQZ, etc. Algorithm is concat the text & run Lempel-Ziv algorithm. Locality needed!

Lempel-Ziv algorithm: Iteratively build a lookup of previously observed keywords & iteratively encode afterwards. High locality ensures lookup fills slowly & improves performance.

Post mapping; SAM format:

Read name	Flag	Map	Map quality	CIGAR		
FCB01H4ABXX:6:2103:15210:113744	137	chr1 10001	0	90M	=	10001 0

TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCC
CTAACCCTAACCCCAACCCCAACCCCAACCC *Read sequence*

HHHHHGEEEGHHHGGBFGGGHGHHBEE?GECHHFHG9FFGF<DBFGGG<GGGGGAFGG
GGAEDFEDADA########################## *Read quality*

X0:i:350 MD:Z:72T5T5T5 RG:Z:1 XG:i:0 AM:i:0 NM:i:3 SM:i:0 XM:i:3 XO:i:0
XT:A:R

Keep: **137** ; **chr1:10001** ; **0** ; **90M**; **72T5T5T5** ; **(#,#,#)**
Add a layer of Huffman encoding

- Accuracy
  - Due to repeats, we need a confidence score in alignment
- Sensitivity
  - Don't lose information
- Speed
- Think of the memory usage
- Output
  - Keep all needed information, but don't overflow your disks
- All read mapping algorithms perform alignment at some point (read vs. reference)

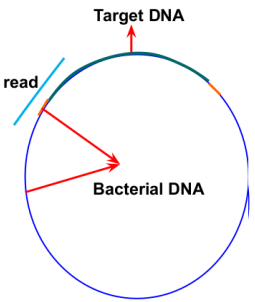
- One human genome
  - 40X coverage
  - 134 GB gzipped = 479 GB raw text
  - Mapped with BWA; >1 day with 200 CPUs
  - SAM format converted to BAM file: 112 GB
  - BAM to CRAM: 7.5 GB
  - Decode CRAM to BAM: 33 GB (lossy!!!)

Read Mapping

- When we have a reference genome & reads from DNA sequencing, which part of the genome does it come from?
- Challenges:
  - Sanger sequencing
    - Cloning vectors
    - Millions of long (~1000 bp reads)
  - HTS sequencing:
    - Billions of short reads with low error
    - OR: hundreds of millions of long reads with high error
  - Common: sequencing errors
    - More prevalent in HTS
  - Common: contamination
    - Typically ~2-3% of reads come from different sources; i.e. human resequencing contaminated with yeast, E. coli, etc.
  - Common: Repeats & Duplications

Sanger vs HTS: cloning vectors

- Sanger reads may contain sequence from the cloning vector; thus mapping needs *local alignment.*
- No cloning vectors in HTS, *global alignment* is fine.



Mapping Reads

Problem: We are given a read, *R*, and a reference sequence, *S*. Find the best or all occurrences of *R* in *S*.

Example:
R = AAACGAGTTA
S = TTAATGC*AAACGAGTTA*CCCAATATATAT*AAACCAGTTA*TT

Considering no error: one occurrence.
Considering up to 1 substitution error: two occurrences.
Considering up to 10 substitution errors: many meaningless occurrences!

Don't forget to search in both forward and reverse strands!!!

Variations:

- Sequencing error
  - No error: *R* is a perfect subsequence of *S*.
  - Only substitution error: *R* is a subsequence of *S* up to a few substitutions.
  - Indel and substitution error: *R* is a subsequence of *S* up to a few short indels and substitutions.
- Junctions (for instance in alternative splicing)
  - Fixed order/orientation
    $R = R_1R_2...R_n$ and R_i map to different non-overlapping loci in *S*, but to the same strand and preserving the order.
  - Arbitrary order/orientation
    $R = R_1R_2...R_n$ and R_i map to different non-overlapping loci in *S*.

Mapping algorithms

- Two main "styles":
  - Hash based seed-and-extend (hash table, suffix array, suffix tree)
    - Index the k-mers in the genome
      - Continuous seeds and gapped seeds
    - When searching a read, find the location of a k-mer in the read; then extend through alignment
    - Requires large memory; this can be reduced with cost to run time
    - More sensitive, but slow
  - Burrows-Wheeler Transform & Ferragina-Manzini Index based aligners
    - BWT is a data compression method used to compress the genome index
    - Perfect hits can be found very quickly, memory lookup costs increase for imperfect hits
    - Reduced sensitivity
  - Today's standard: hybrid
    - Seed with BWT-FM then extend

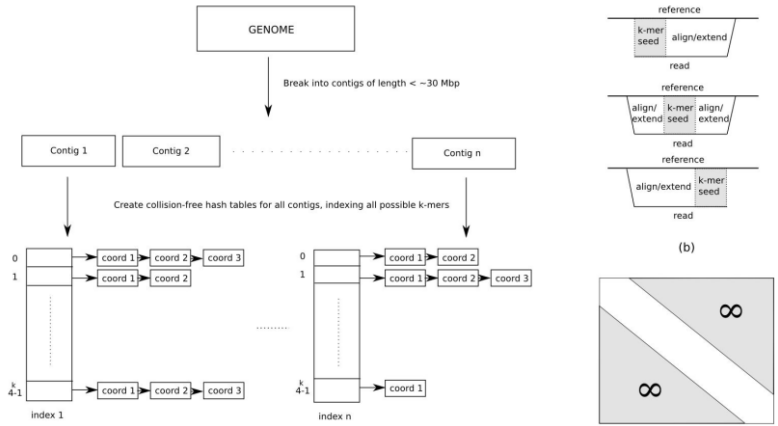
Short read mappers

- BWT-FM based
  - Illumina: BWA, Bowtie, SOAP2
  - Human genome can be compressed into a 2.3 GB data structure through BWT
  - Extremely fast for perfect hits
  - Increased memory lookups for mismatch
    - Indels are found in postprocessing when paired-end reads are available
  - GPGPU implementations: SOAP3 (poor performance due to memory lookups)
- Hybrid: BWA-MEM

Long read mappers

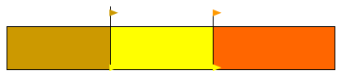
- PacBio and ONT:
  - BLASR (suffix-tree based indexing)
  - MashMap and Minimap2 (minimizers + chaining + Smith-Waterman)
    - Paper presentation candidate
  - NGM-LR (hash table + chaining + alignment w/ convex gap penalty model
    - Paper presentation candidate

Hash Based Aligners



Seed and extend

- Break the read into *n* segments of k-mers.
  - For perfect sensitivity under edit distance *e*
    - There is at least one *l*-mer where l = floor(*L*/(e+1)); *L*=read length
    - For fixed *l*=*k*; *n* = *e*+1 and k ≤ *L* / *n*
  - Large k -> large memory
  - Small k -> more hash hits
- Lets consider the read length is 36 bp, and k=12.



- if we are looking for 2 edit distance (mismatch, indel) this would guaranty to find all of the hits

Mapping Quality

- MAPQ = -10 * $\log_{10}$(Prob(mapping is wrong))

For reference sequence *x;* read sequence *z:*
p(z | x,u) **=** probability that *z* comes from position *u*
= multiplication of p_e of mismatched bases of *z*

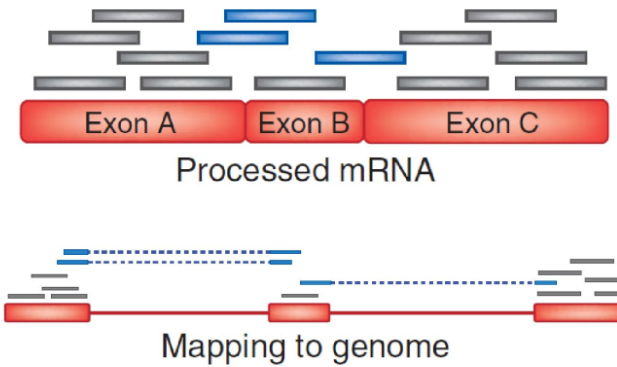
For posterior probability **p(u | x,z)** assume uniform prior distribution **p(u|x)**
L=|x| and *l*=|z|. Apply Bayesian formula:

$$p_s(u|x,z) = \frac{p(z|x,u)}{\sum_{v=1}^{L-l+1} p(z|x,v)}$$

$$Q_s(u|x,z) = -10\log_{10}[1 - p_s(u|x,z)].$$

Calculated for one "best" hit Li et al., Genome Research, 2008

Spliced-read mapping



- Used for processed mRNA data
- Reports reads that span introns.
- Examples: TopHat, ERANGE