Index assisted matching

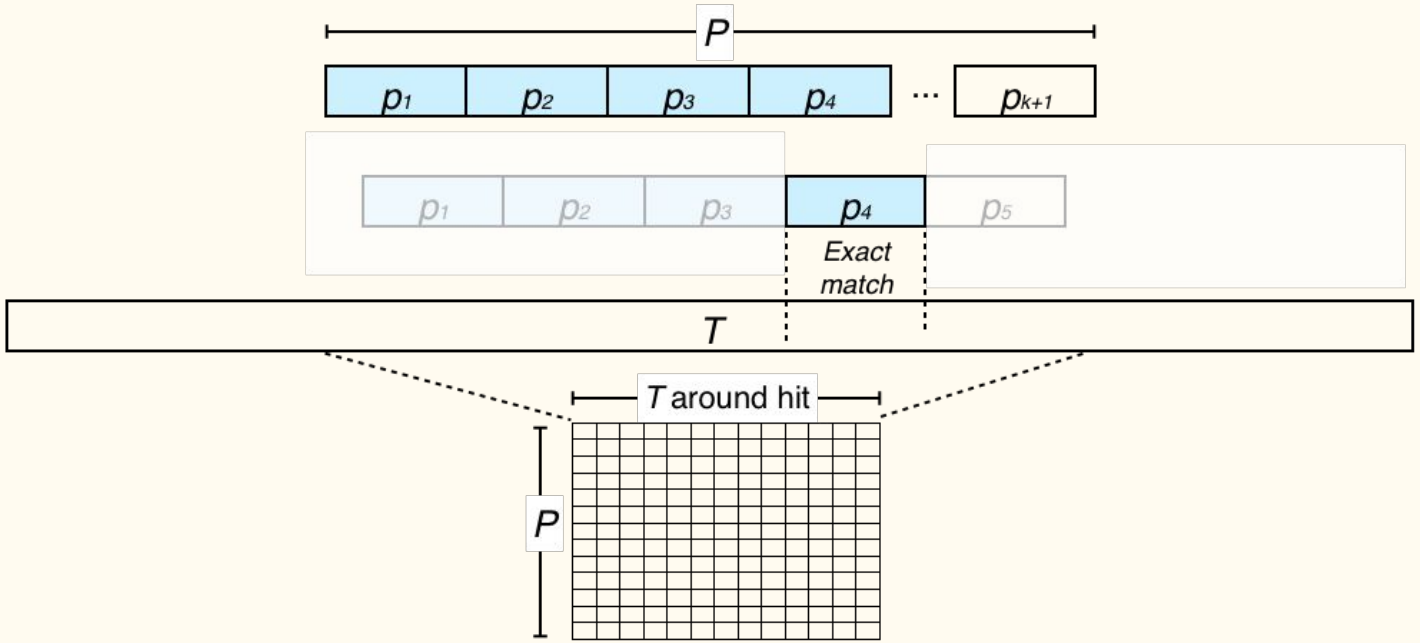
Index assisted matching

Use index for exact-matching subproblems, follow up with DP

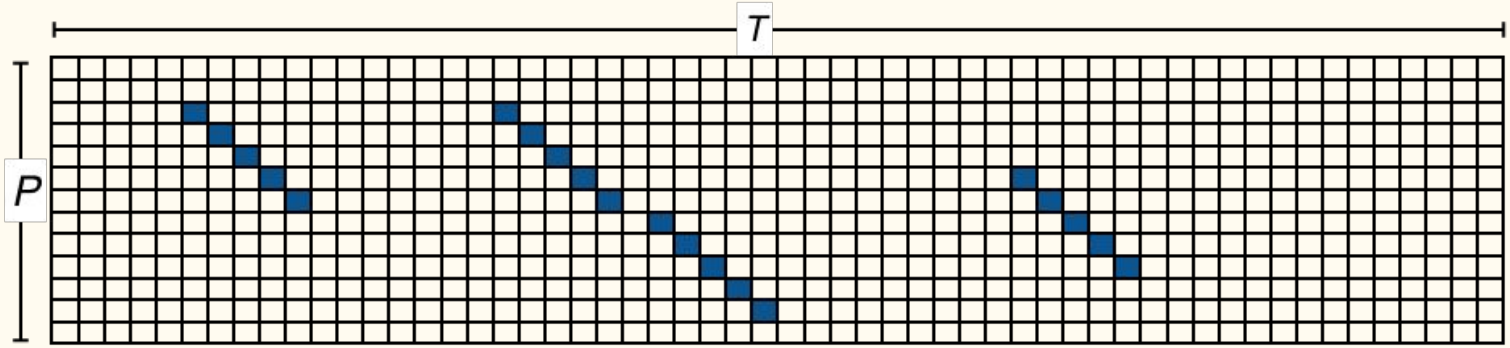
Partition P, like for pigeonhole

Index finds exact partition matches (hits)

Use DP in vicinity of exact matches



Index assisted matching



Index is identifying diagonal stretches of matches.

These are likely to be part of a high-scoring alignment.

Many stretches within a few diagonals of each other are even more likely to be part of a high-scoring alignment.

Seed-and-extend approach

—

Example with Bowtie2

Two-step aligners

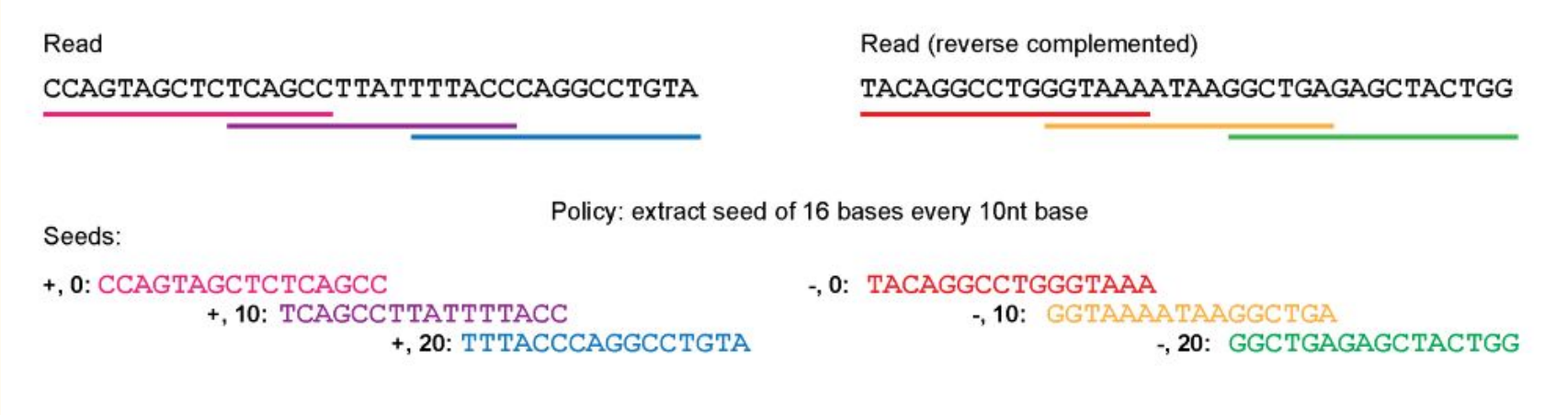
Most modern aligners take a two-step approach (also called **“seed and extend”**):

1. Find “coarse” alignments or seeds
2. Do fine detailed alignments in the vicinity of the seeds and chose the best scoring fine grain alignment

Unpaired alignment

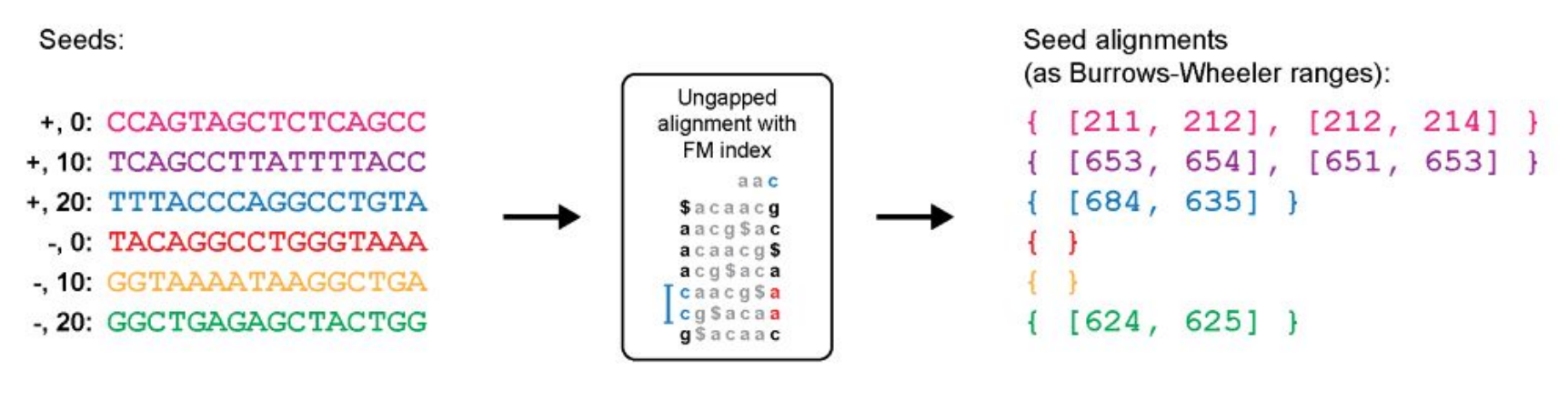
Bowtie 4-steps workflow:

step 1: Extracting substrings (“seed” strings) from the read and its reverse complement



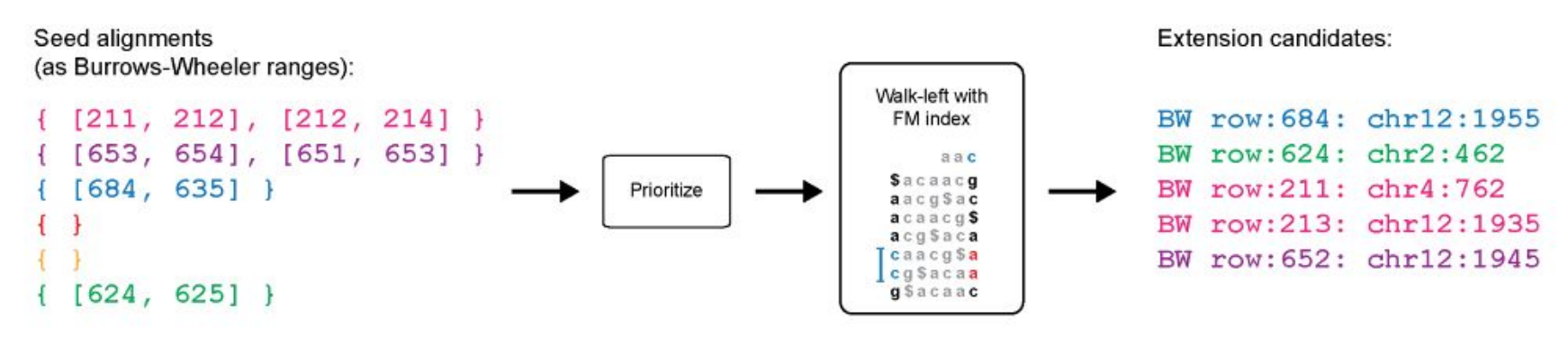
Unpaired alignment

step 2: Ungapped alignment of ‘seed’ strings as in Bowtie 1



Unpaired alignment

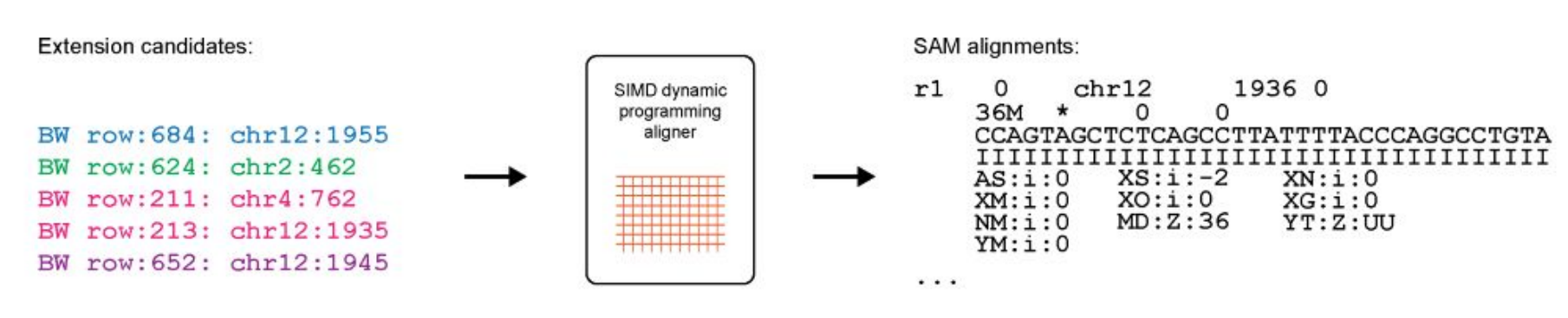
step 3: Prioritization and offset resolving



Bowtie 2 assigns a priority to each row (i.e. locus) equal to $1/r^2$ where r is the total number of rows in the range.

Unpaired alignment

step 4: SIMD-accelerated dynamic programming (“extend”)



Seeding step

- Find a set of possible coordinates in reference genome
- Many false-positives, but it is usually fast

Read:

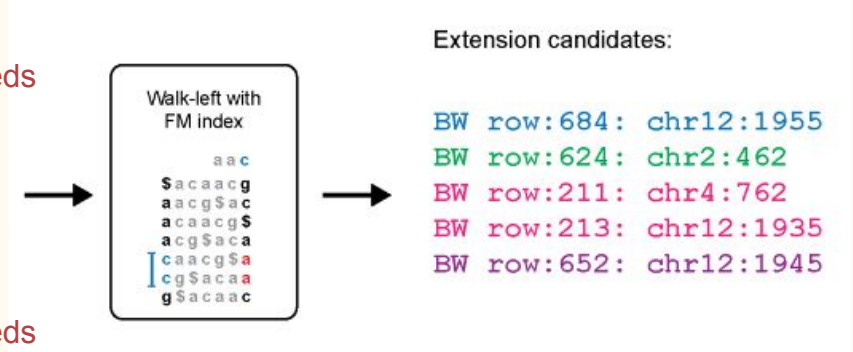
```
CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
CCAGTAGCTCTCAGCC
            TCAGCCTTATTTTACC
                        TTTACCCAGGCCTGTA
```

Seeds

Read (reverse complemented):

```
TACAGGCCTGGGTAAAATAAGGCTGAGAGCTACTGG
TACAGGCCTGGGTAAA
            GGTAAAATAAGGCTGA
                        GGCTGAGAGCTACTGG
```

Seeds



Extending step

- Calculates a precise sequences match score
- Often use a lot of RAM (related to sequence size)
- The extend step also needs to produce the information on **how** the sequences match
- Mostly based on **dynamic programming**

Dynamic programming

Recursive definition:

$$S(i,j) = max \begin{cases} S(i-1,j-1) + \sigma(x_i, y_j), \\ S(i-1,j) + \gamma, \\ S(i,j-1) + \gamma. \end{cases}$$

$\sigma(x_i, y_j)$: Match score if letters match, otherwise mismatch penalty

γ : Gap penalty

Dynamic programming matrix

Dynamic programming matrix:

j → (sequence **y**)

i (sequence **x**)		0	1 **T**	2 **G**	3 **C**	4 **T**	5 **C**	6 **G**	7 **T**	8 = N **A**
0		0	−6	−12	−18	−24	−30	−36	−42	−48
1	**T**	−6	5	−1	−7	−13	−19	−25	−31	−37
2	**T**	−12	−1	3	−3	−2	−8	−14	−20	−26
3	**C**	−18	−7	−3	8	2	3	−3	−9	−15
4	**A**	−24	−13	−9	2	6	0	1	−5	−4
5	**T**	−30	−19	−15	−4	7	4	−2	6	0
M = 6	**A**	−36	−25	−21	−10	1	5	2	0	11

Optimum alignment scores 11:

T	**–**	**–**	**T**	**C**	**A**	**T**	**A**
T	**G**	**C**	**T**	**C**	**G**	**T**	**A**
+5	−6	−6	+5	+5	−2	+5	+5

- Calculate scores for each field
- Remember the path to each field
- Backtrack from maximum to zero
- Diagonal step is match/mismatch
- Horizontal step is a deletion
- Vertical step is an insertion

Alignment output

```
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001  147 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

SAM format specification: https://genome.sph.umich.edu/wiki/SAM

CIGAR string

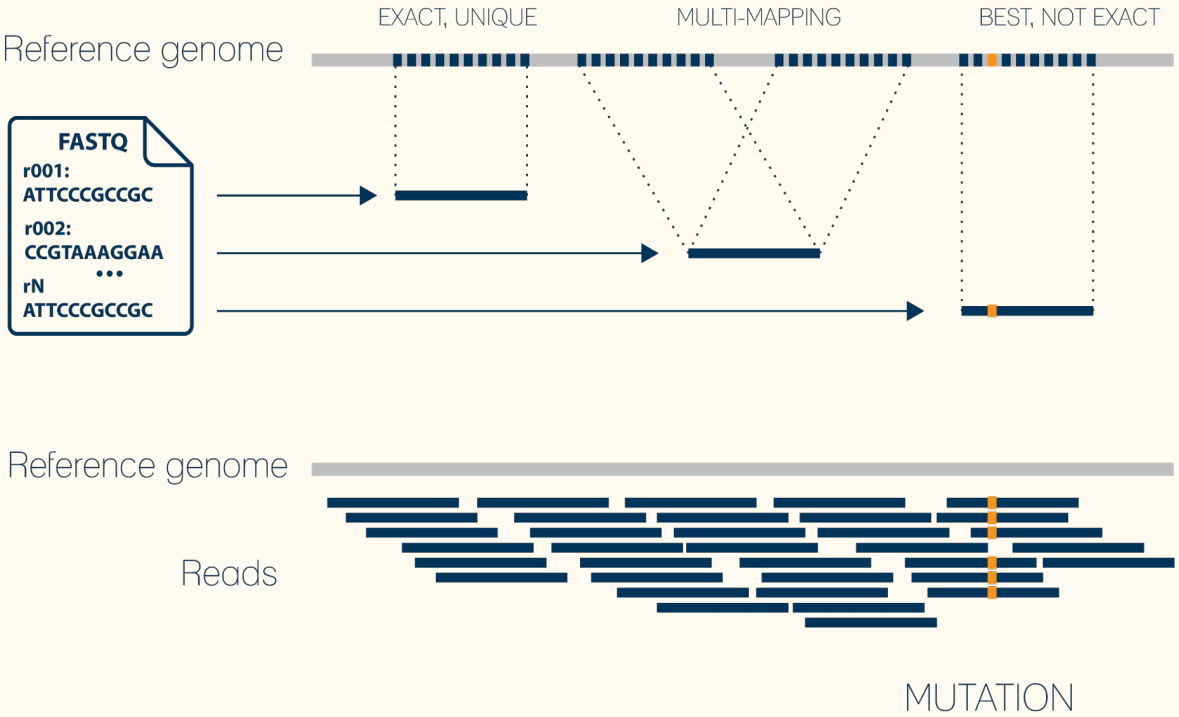
- CIGAR (Concise Idiosyncratic Gapped Alignment Report) Codes:
  - **M** - alignment match (Match or Mismatch)
  - **I** - insertion
  - **D** - Deletion
  - **S** - soft clip
  - **N** - skipping (usually) introns
  - H, X, =, P - rare

Example:

TGCTC - GTA
TCC - CTGT -

=> 3M1D1M1I2M1S

Where are we?


EXACT, UNIQUE
MULTI-MAPPING
BEST, NOT EXACT
Reference genome
FASTQ
r001:
ATTCCCGCCGC
r002:
CCGTAAAGGAA
•••
rN
ATTCCCGCCGC
Reference genome
Reads
MUTATION

BLAST

—

Basic Local Alignment Search Tool

BLAST

BLAST (Altschul et al., 1990) and its variants are some of the most common sequence search tools in use.

Roughly, the basic BLAST has three parts:

1. Find segment pairs between the query sequence and a database sequence above score threshold (”seed hits”)
2. Extend seed hits into locally maximal segment pairs
3. Estimating the statistical significance of found matches

Gapped BLAST introduced in 1997 allows for gaps in alignments

Finding seed hits

First, we generate a set of neighborhood sequences for given k, match score matrix and threshold T.

I = GCATCGGC, J = CCATCGCCATCG, k = 5, match score 1, mismatch score 0, T = 4. This allows for one mismatch in each k-word.
The neighborhood of the first k-word of I, GCATC, is GCATC and there are 15 possible sequences for this k-mer.

$$\begin{cases} \texttt{A} \\ \texttt{CCATC,G} \\ \texttt{T} \end{cases} \begin{cases} \texttt{A} \\ \texttt{GATC,GC} \\ \texttt{T} \end{cases} \begin{cases} \texttt{C} \\ \texttt{GTC,GCA} \\ \texttt{T} \end{cases} \begin{cases} \texttt{A} \\ \texttt{CC,GCAT} \\ \texttt{G} \end{cases} \begin{cases} \texttt{A} \\ \texttt{G} \\ \texttt{T} \end{cases}$$

We repeat this for every k-mer. Aho-Corasick algorithm is used for this part.

Extend seed hits

After finding seed hits in database, we do extension of seed sequencing wia local alignment. We extend seeds until the alignment value drops off below certain threshold.

We call these High-scoring Segment Pairs (HSP).

Estimate statistical significance of found hits

Last, we estimate how likely it that these hits happened by chance (E-value). The more E-value is closer to zero, the less likely is that hit happended by chance.

The BLAST algorithm has three parameters: t

- The word size W. → k-mer size
- The word similarity threshold T.
- The minimum match score x (cutoff score x). → for extension

The choice of the parameters in BLAST is guided by the Altschul-Dembo-Karlin statistics which is also used for computing statistical significance of hits.