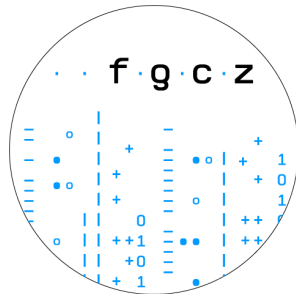


· f · g · c · z ·

# Mapping Reads

Dr. Hubert Rehrauer





## Sequence Alignment

- Optimization problem: Find the alignment with the highest score
- Global alignment (end-to-end): Needleman-Wunsch
- Local alignment: Smith-Waterman

Score matrix:

|   | a  | c  | g  | t  |
|---|----|----|----|----|
| a | +2 | -2 | -1 | -2 |
| c | -2 | +2 | -2 | -1 |
| g | -1 | -2 | +2 | -2 |
| t | -2 | -1 | -2 | +2 |

Gap score: -3

|   |   |    |    |    |    |    |    |    |   |
|---|---|----|----|----|----|----|----|----|---|
| c | c | t  | a  | g  | t  | t  | c  | a  | t |
|   |   |    |    | x  |    | x  |    |    |   |
| t | g | t  | a  | a  | t  | -  | c  | a  | c |
|   |   | +2 | +2 | -1 | +2 | -3 | +2 | +2 |   |

Alignment score = 6



## Local alignments ( Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap** (typically, mismatch penalties are set according to evolutionary knowledge)

Match: +1; Mismatch: -1; Gap: -2

|   | . | A | A | T | G | T |
|---|---|---|---|---|---|---|
| . | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 |   |   |   |   |   |
| T | 0 |   |   |   |   |   |
| G | 0 |   |   |   |   |   |
| A | 0 |   |   |   |   |   |
| C | 0 |   |   |   |   |   |

$$H(i, j) = \max \left\{ \begin{array}{l} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n$$

Match/Mismatch

Deletion

Insertion



## Local alignments ( Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap**

Match: +1; Mismatch: -1; Gap: -2

|   | . | A | A | T | G | T |
|---|---|---|---|---|---|---|
| . | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 |   |   |   |
| T | 0 |   |   |   |   |   |
| G | 0 |   |   |   |   |   |
| A | 0 |   |   |   |   |   |
| C | 0 |   |   |   |   |   |

$$H(i, j) = \max \left\{ \begin{array}{l} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n$$

Match/Mismatch

Deletion

Insertion



## Local alignments ( Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap**

Match: +1; Mismatch: -1; Gap: -2

|   | . | A | A | T | G | T |
|---|---|---|---|---|---|---|
| . | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 2 | 0 | 1 |
| G | 0 |   |   |   |   |   |
| A | 0 |   |   |   |   |   |
| C | 0 |   |   |   |   |   |

$$H(i, j) = \max \left\{ \begin{array}{l} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n$$

Match/Mismatch

Deletion

Insertion



## Local alignments ( Smith-Waterman)

Generally useful for local alignments, determining regions of similarity between two strings (here, DNA sequence)

Dynamic programming based algorithm: gives optimal alignment, with respect to scoring system

Need to set scores for **match**, penalties for **mismatch** and **gap**

Match: +1; Mismatch: -1; Gap: -2

|   | . | A | A | T | G | T |
|---|---|---|---|---|---|---|
| . | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 2 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 3 | 1 |
| A | 0 | 1 | 1 | 0 | 1 | 2 |
| C | 0 | 0 | 0 | 0 | 0 | 0 |

$$H(i, j) = \max \left\{ \begin{array}{l} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n$$

Match/Mismatch

Deletion

Insertion



## Local alignments ( Smith-Waterman)

Traceback:

Start with maximum score (here, 3)

Follow path that gives multiple score

|   | . | A | A | T | G | T |
|---|---|---|---|---|---|---|
| . | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 2 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 3 | 1 |
| A | 0 | 1 | 1 | 0 | 1 | 2 |
| C | 0 | 0 | 0 | 0 | 0 | 0 |

$$H(i, j) = \max \left\{ \begin{array}{l} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n$$

Match/Mismatch

Deletion

Insertion



## Local alignments ( Smith-Waterman)

Traceback:

Start with maximum score (here, 3)

Follow path that gives multiple score

This gives:

AATGT-

-ATGAC

|   | . | A | A | T | G | T |
|---|---|---|---|---|---|---|
| . | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 2 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 3 | 1 |
| A | 0 | 1 | 1 | 0 | 1 | 2 |
| C | 0 | 0 | 0 | 0 | 0 | 0 |

$$H(i, j) = \max \left\{ \begin{array}{l} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{array} \right\}, \quad 1 \leq i \leq m, 1 \leq j \leq n$$

Match/Mismatch

Deletion

Insertion





## Local alignments ( Smith-Waterman)

Other considerations:

Natural extension is to have a different *gap opening* and a *gap extension* penalty (former generally being larger)

GO penalty=-2; GE penalty=-1; Mismatch=-1; Match=1

With above penalties, which is the best scoring alignment?

AT-C-GT

ATC--GT

AT-C--GT

ATTTTGT

ATTTTGT

ATT-TTGT



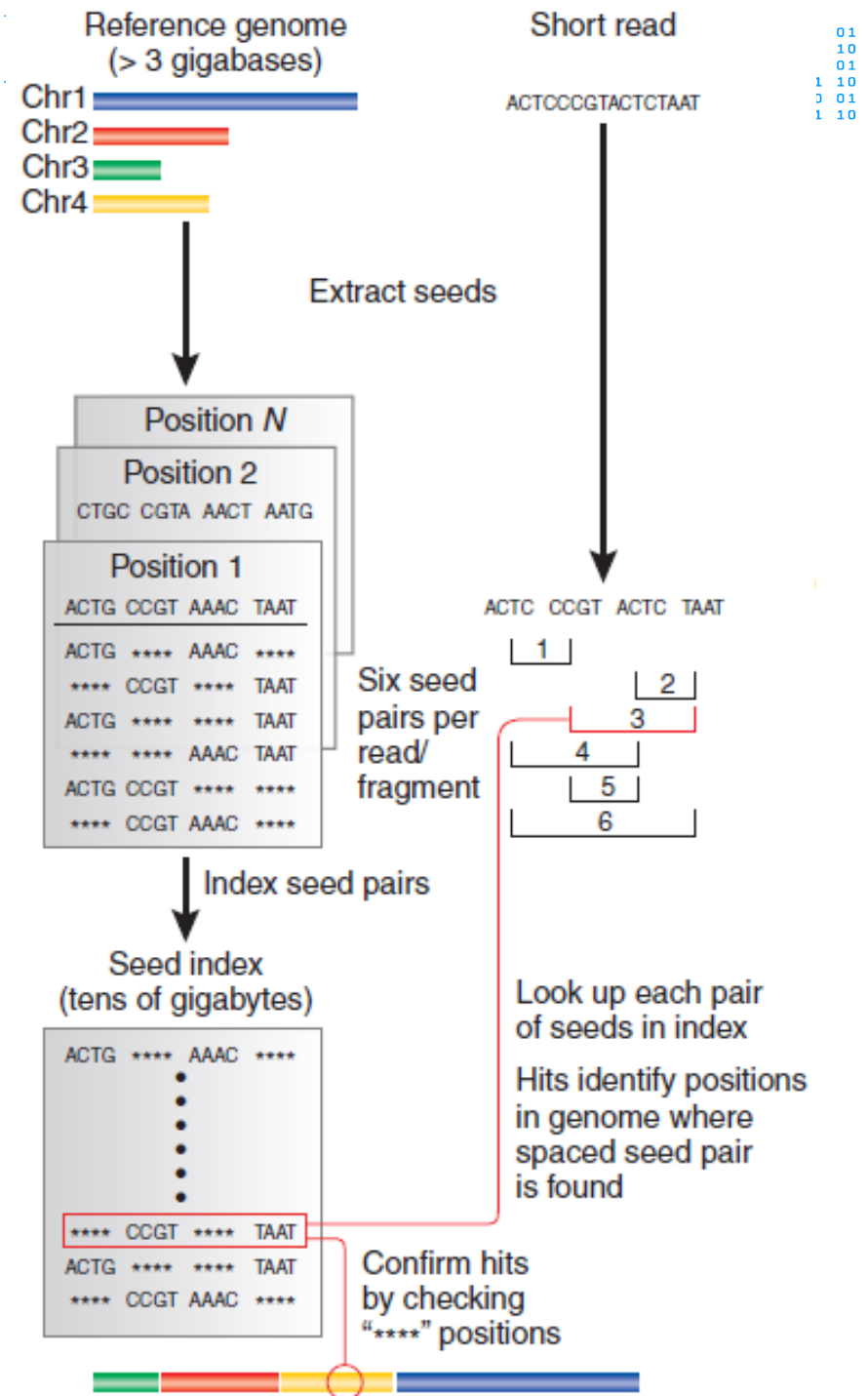
# Alignment Tools

- Dynamic programming is slow
- Speed-up with heuristics
  - e.g. exactly align short subsequences and extend these alignments
    - e.g. BLAST / BLAT
  - no longer guaranteed to find the best alignments
  - exact matches are found by index lookup



## Index Genome: Spaced Seeds

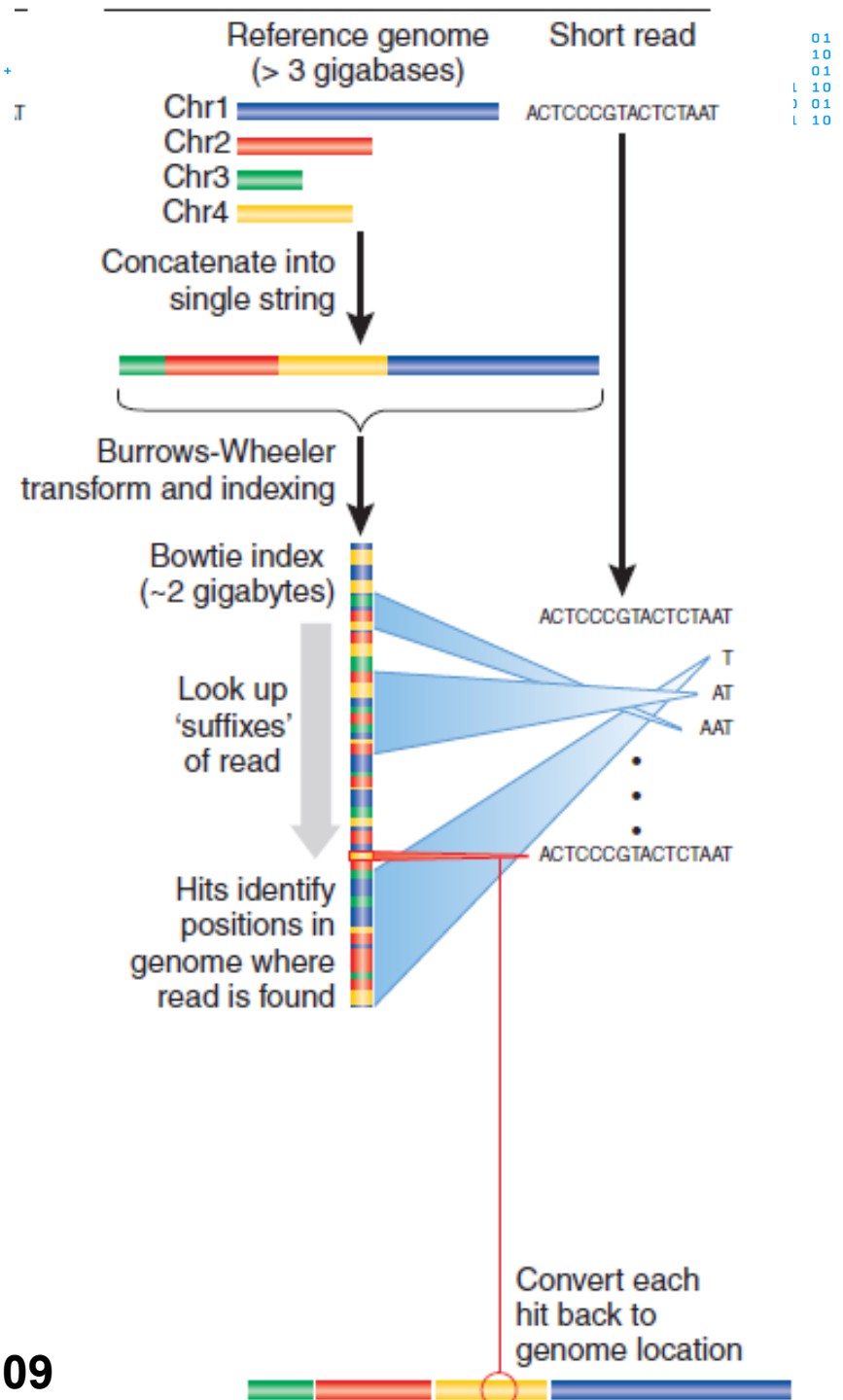
- Tags and tag-sized pieces of reference are cut into small “seeds.”
- Pairs of spaced seeds are stored in an index.
- Look up spaced seeds for each tag.
- For each “hit,” confirm the remaining positions.





## Index Genome: Burrows-Wheeler Transform

- Store entire reference genome.
- Align tag base by base from the end.
- When tag is traversed, all active locations are reported.
- If no match is found, then back up and try a substitution.





# The Burrows-Wheeler transform (1994; 1983)

c a c a a c g \$



|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| c  | a  | c  | a  | a  | c  | g  | \$ |
| a  | c  | a  | a  | c  | g  | \$ | c  |
| c  | a  | a  | c  | g  | \$ | c  | a  |
| a  | a  | c  | g  | \$ | c  | a  | c  |
| a  | c  | g  | \$ | c  | a  | c  | a  |
| c  | g  | \$ | c  | a  | c  | a  | a  |
| g  | \$ | c  | a  | c  | a  | a  | c  |
| \$ | c  | a  | c  | a  | a  | c  | g  |



|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| \$ | c  | a  | c  | a  | a  | c  | g  |
| a  | a  | c  | g  | \$ | c  | a  | c  |
| a  | c  | a  | a  | c  | g  | \$ | c  |
| a  | c  | g  | \$ | c  | a  | c  | a  |
| c  | a  | a  | c  | g  | \$ | c  | a  |
| c  | a  | c  | a  | a  | c  | g  | \$ |
| c  | g  | \$ | c  | a  | c  | a  | a  |
| g  | \$ | c  | a  | c  | a  | a  | c  |



g c c a a \$ a c



## The “Last-First mapping” property

The relative ordering of a particular character (say **c**) in column 1 is the same as that in the last column

$c_1$  a  $c_2$  a a  $c_3$  g \$

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| \$    | $c_1$ | a     | $c_2$ | a     | a     | $c_3$ | g     |
| a     | a     | $c_3$ | g     | \$    | $c_1$ | a     | $c_2$ |
| a     | $c_2$ | a     | a     | $c_3$ | g     | \$    | $c_1$ |
| a     | $c_3$ | g     | \$    | $c_1$ | a     | $c_2$ | a     |
| $c_2$ | a     | a     | $c_3$ | g     | \$    | $c_1$ | a     |
| $c_1$ | a     | $c_2$ | a     | a     | $c_3$ | g     | \$    |
| $c_3$ | g     | \$    | $c_1$ | a     | $c_2$ | a     | a     |
| g     | \$    | $c_1$ | a     | $c_2$ | a     | a     | $c_3$ |



# The “Last-First mapping” property

The relative ordering of a particular character (say **c**) in column 1 is the same as that in the last column

$c_1$  a  $c_2$  a a  $c_3$  g \$

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| \$    | $c_1$ | a     | $c_2$ | a     | a     | $c_3$ | g     |
| a     | a     | $c_3$ | g     | \$    | $c_1$ | a     | $c_2$ |
| a     | $c_2$ | a     | a     | $c_3$ | g     | \$    | $c_1$ |
| a     | $c_3$ | g     | \$    | $c_1$ | a     | $c_2$ | a     |
| $c_2$ | a     | a     | $c_3$ | g     | \$    | $c_1$ | a     |
| $c_1$ | a     | $c_2$ | a     | a     | $c_3$ | g     | \$    |
| $c_3$ | g     | \$    | $c_1$ | a     | $c_2$ | a     | a     |
| g     | \$    | $c_1$ | a     | $c_2$ | a     | a     | $c_3$ |

Proof:

Suppose  $c X$  and  $c Y$  are cyclic permutations of the input  $T$ .

Suppose  $c X < c Y$  (in lexicographical ordering)

Then  $X c < Y c$  (in lexicographical ordering)

The LF-mapping property follows.



# BWT is reversible

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| \$ | c  | a  | c  | a  | a  | c  | g  |
| a  | a  | c  | g  | \$ | c  | a  | c  |
| a  | c  | a  | a  | c  | g  | \$ | c  |
| a  | c  | g  | \$ | c  | a  | c  | a  |
| c  | a  | a  | c  | g  | \$ | c  | a  |
| c  | a  | c  | a  | a  | c  | g  | \$ |
| c  | g  | \$ | c  | a  | c  | a  | a  |
| g  | \$ | c  | a  | c  | a  | a  | c  |

g c c a a \$ a c



\$ a a a c c c g





# BWT is reversible

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| \$ | c  | a  | c  | a  | a  | c  | g  |
| a  | a  | c  | g  | \$ | c  | a  | c  |
| a  | c  | a  | a  | c  | g  | \$ | c  |
| a  | c  | g  | \$ | c  | a  | c  | a  |
| c  | a  | a  | c  | g  | \$ | c  | a  |
| c  | a  | c  | a  | a  | c  | g  | \$ |
| c  | g  | \$ | c  | a  | c  | a  | a  |
| g  | \$ | c  | a  | c  | a  | a  | c  |

g\$ ca ca aa ac \$c ac cg

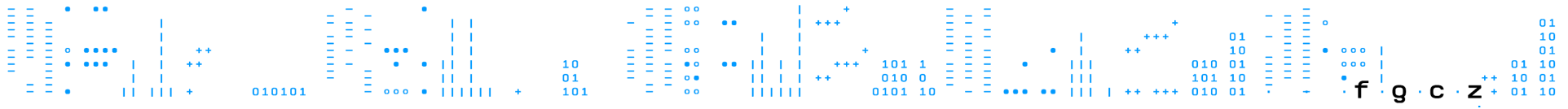


\$c aa ac ac ca ca cg g\$



**BWT is reversible**

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| \$ | c  | a  | c  | a  | a  | c  | g  |
| a  | a  | c  | g  | \$ | c  | a  | c  |
| a  | c  | a  | a  | c  | g  | \$ | c  |
| a  | c  | g  | \$ | c  | a  | c  | a  |
| c  | a  | a  | c  | g  | \$ | c  | a  |
| c  | a  | c  | a  | a  | c  | g  | \$ |
| c  | g  | \$ | c  | a  | c  | a  | a  |
| g  | \$ | c  | a  | c  | a  | a  | c  |



## Lookup AAC

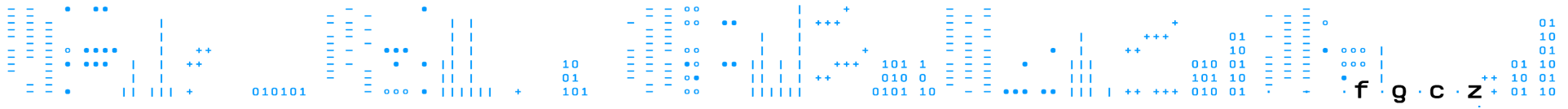
|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| \$ | c  | a  | c  | a  | a  | c  | g  |
| a  | a  | c  | g  | \$ | c  | a  | c  |
| a  | c  | a  | a  | c  | g  | \$ | c  |
| a  | c  | g  | \$ | c  | a  | c  | a  |
| c  | a  | a  | c  | g  | \$ | c  | a  |
| c  | a  | c  | a  | a  | c  | g  | \$ |
| c  | g  | \$ | c  | a  | c  | a  | a  |
| g  | \$ | c  | a  | c  | a  | a  | c  |

Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



# Lookup AAC

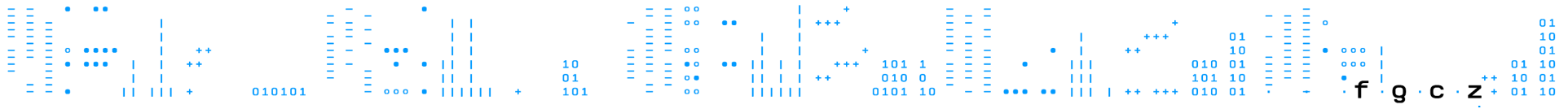
|   |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|
|   | \$ | c  | a  | c  | a  | a  | c  | g  |
|   | a  | a  | c  | g  | \$ | c  | a  | c  |
|   | a  | c  | a  | a  | c  | g  | \$ | c  |
|   | a  | c  | g  | \$ | c  | a  | c  | a  |
| → | c  | a  | a  | c  | g  | \$ | c  | a  |
|   | c  | a  | c  | a  | a  | c  | g  | \$ |
|   | c  | g  | \$ | c  | a  | c  | a  | a  |
| → | g  | \$ | c  | a  | c  | a  | a  | c  |

Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



## Lookup AAC

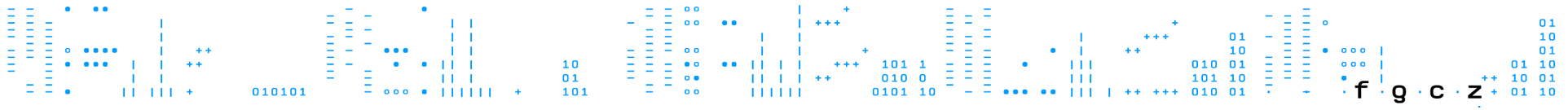
|   |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|
|   | \$ | c  | a  | c  | a  | a  | c  | g  |
|   | a  | a  | c  | g  | \$ | c  | a  | c  |
|   | a  | c  | a  | a  | c  | g  | \$ | c  |
|   | a  | c  | g  | \$ | c  | a  | c  | a  |
| → | c  | a  | a  | c  | g  | \$ | c  | a  |
|   | c  | a  | c  | a  | a  | c  | g  | \$ |
|   | c  | g  | \$ | c  | a  | c  | a  | a  |
| → | g  | \$ | c  | a  | c  | a  | a  | c  |

Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

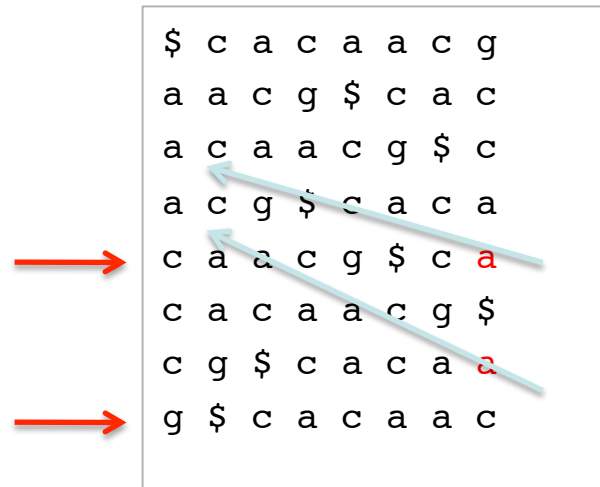
While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



# Lookup AAC

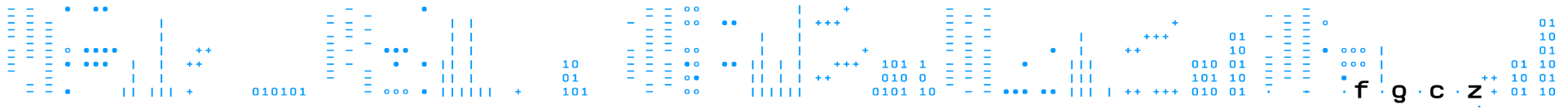


Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



## Lookup AAC

|   |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|
|   | \$ | c  | a  | c  | a  | a  | c  | g  |
|   | a  | a  | c  | g  | \$ | c  | a  | c  |
| → | a  | c  | a  | a  | c  | g  | \$ | c  |
| → | a  | c  | g  | \$ | c  | a  | c  | a  |
|   | c  | a  | a  | c  | g  | \$ | c  | a  |
|   | c  | a  | c  | a  | a  | c  | g  | \$ |
|   | c  | g  | \$ | c  | a  | c  | a  | a  |
|   | g  | \$ | c  | a  | c  | a  | a  | c  |

Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



# Lookup **A**AC

|   |    |    |    |    |    |    |    |          |
|---|----|----|----|----|----|----|----|----------|
| → | \$ | c  | a  | c  | a  | a  | c  | g        |
|   | a  | a  | c  | g  | \$ | c  | a  | c        |
| → | a  | c  | a  | a  | c  | g  | \$ | c        |
|   | a  | c  | g  | \$ | c  | a  | c  | <b>a</b> |
|   | c  | a  | a  | c  | g  | \$ | c  | a        |
|   | c  | a  | c  | a  | a  | c  | g  | \$       |
|   | c  | g  | \$ | c  | a  | c  | a  | a        |
|   | g  | \$ | c  | a  | c  | a  | a  | c        |

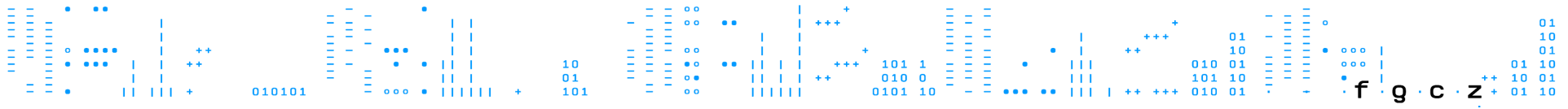
Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

While characters left (and nonzero range):

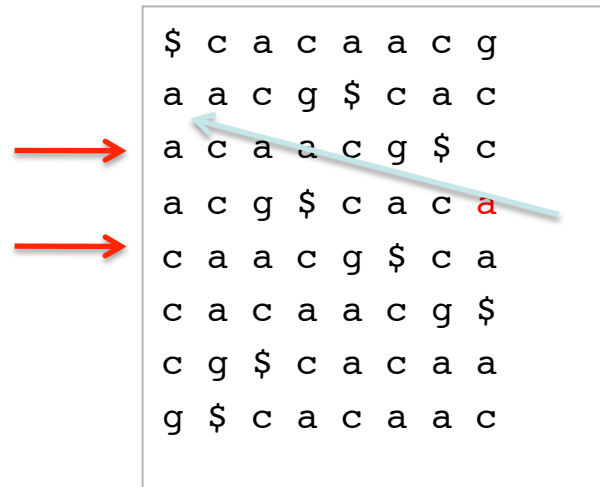
Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match





# Lookup **A**AC



Range  $\leftarrow$  range of last character in 1<sup>st</sup> column


While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



## Lookup **A**AC



|          |          |          |    |    |    |    |    |
|----------|----------|----------|----|----|----|----|----|
| \$       | c        | a        | c  | a  | a  | c  | g  |
| <b>a</b> | <b>a</b> | <b>c</b> | g  | \$ | c  | a  | c  |
| a        | c        | a        | a  | c  | g  | \$ | c  |
| a        | c        | g        | \$ | c  | a  | c  | a  |
| c        | a        | a        | c  | g  | \$ | c  | a  |
| c        | a        | c        | a  | a  | c  | g  | \$ |
| c        | g        | \$       | c  | a  | c  | a  | a  |
| g        | \$       | c        | a  | c  | a  | a  | c  |

Range  $\leftarrow$  range of last character in 1<sup>st</sup> column

While characters left (and nonzero range):

Lookup first and last match to preceding character in final column

Range  $\leftarrow$  LF-mapping of first and last match



## Comparison

### Spaced seeds

- Requires ~50Gb of memory.
- Runs 30-fold slower.
- More straightforward to program.
- Examples:
  - MAQ
  - Shrimp
- More tolerant to
  - sequence variations
  - sequencing errors

### Burrows-Wheeler

- Requires <2Gb of memory.
- Runs 30-fold faster.
- More complicated to program.
- Examples:
  - bowtie
  - BWA
  - tophat (uses bowtie)
  - STAR



## Alignment with Mismatches

- Mismatches can occur because of
  - sequencing error (error rate  $\sim 1/500$ )
  - mutation; (human mutation rate  $\sim 1/1e4$ )
  - $\rightarrow$  if reads are long ( $> 100\text{nt}$ ) reads with mismatches will not be rare; more than  $\sim 10\%$  of the reads may have a mismatch
- If there is a sequence mismatch the index lookup fails! How to find nevertheless an alignment?



## BWT Alignment with Mismatches

- Strategies:
  - If the BWT lookup fails at position k, try all different bases at position k
    - drawback: computing effort grows exponentially with the number of mismatches
    - implemented e.g. in bowtie
  - Chop reads in segments (seeds) and align those mismatch-free and stitch seed alignments together
    - implemented e.g. in bowtie



## Consideration: Sequencing Errors – PHRED quality

- Each called base is given a quality score Q
  - $Q = 10 * \log_{10}(\text{estimated probability that call is wrong})$ 
    - 10 prob = 0.1
    - 20 prob = 0.01
    - 30 prob = 0.001

[Q30 often used as a threshold for useful sequence data]
- down-weight low-quality bases when computing the alignment score
- These quality scores are called PHRED scores. They were first introduced for capillary sequencers.
  - PHRED scores are determined by the sequencer that directly rates how reliable the measured signal is

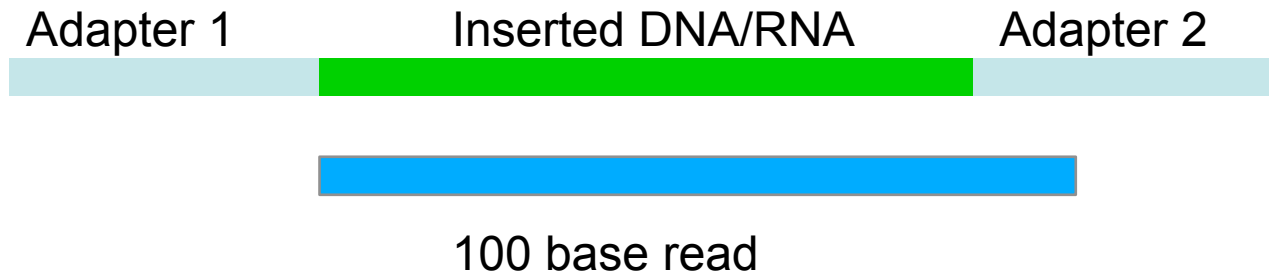


## Considerations: Read trimming

- Sequencers have systematic errors
- Illumina sequencers have a higher error rate at the first few bases (1-5)
- Basically all sequencers have increasing error rate towards the end of the read
- Hard trimming: trim a fixed number of bases from the beginning and/or end
- Quality trimming: cut the end of the read as soon as the base quality drops below a threshold



## Considerations: Read trimming



- If the inserted DNA/RNA fragment is too short, the read will contain part of the adapter
- Adapter trimming can be challenging if
  - if the insert is 90 – 100 bases
  - if the read has many sequencing errors





## Multiple Alignments

- A read may have multiple valid alignments with identical or similarly good alignment scores
- Aligners allow to choose different reporting strategies:
  - Randomly select one alignment from the top-scoring alignments
  - Report all alignments that are within *delta* of the top-scoring alignment; clip if more than *Nmax* alignments are found
  - Report only alignments if they are unique (no other alignment within *delta* of the alignment score)
  - Do not report anything if more than *Nmax* valid alignments are found
  - ...
- Whether a read has a **unique alignment** depends on
  - the read sequence and the sequence homology of the organism
  - the search algorithm of the aligner
  - the reporting strategy of the aligner



## Read Alignment Summary (I)

### How to map billions of reads?

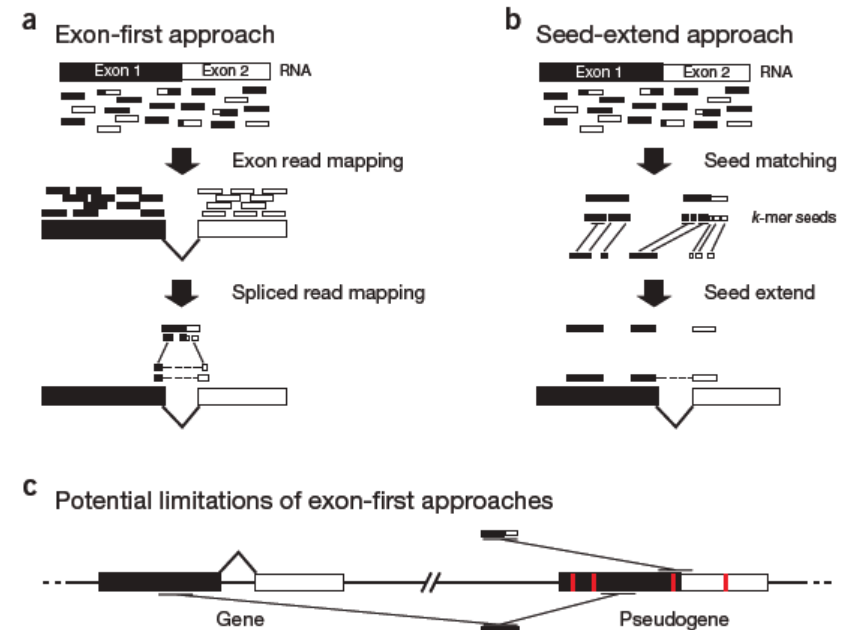
- The major aligners use the Burrows-Wheeler-Transform to create an index of the reference. Even for the human genome the index fits into 3GB RAM.
- Reads are aligned by **index lookup** not by sequence comparison
- The lookup of a perfect match read is faster than loading the read and writing the alignment coordinates to the output file!
- If the lookup fails because of SNPs or sequencing errors, an actual sequence comparison is done –and this can take time!

- ## Read Alignment Summary (II)



# RNA-seq Mapping

- Mapping targets:
  - transcriptome
  - splice junction library
  - genome
- Mouse retina 60+60bp reads:
  - 41 of 91 Mio map to junctions

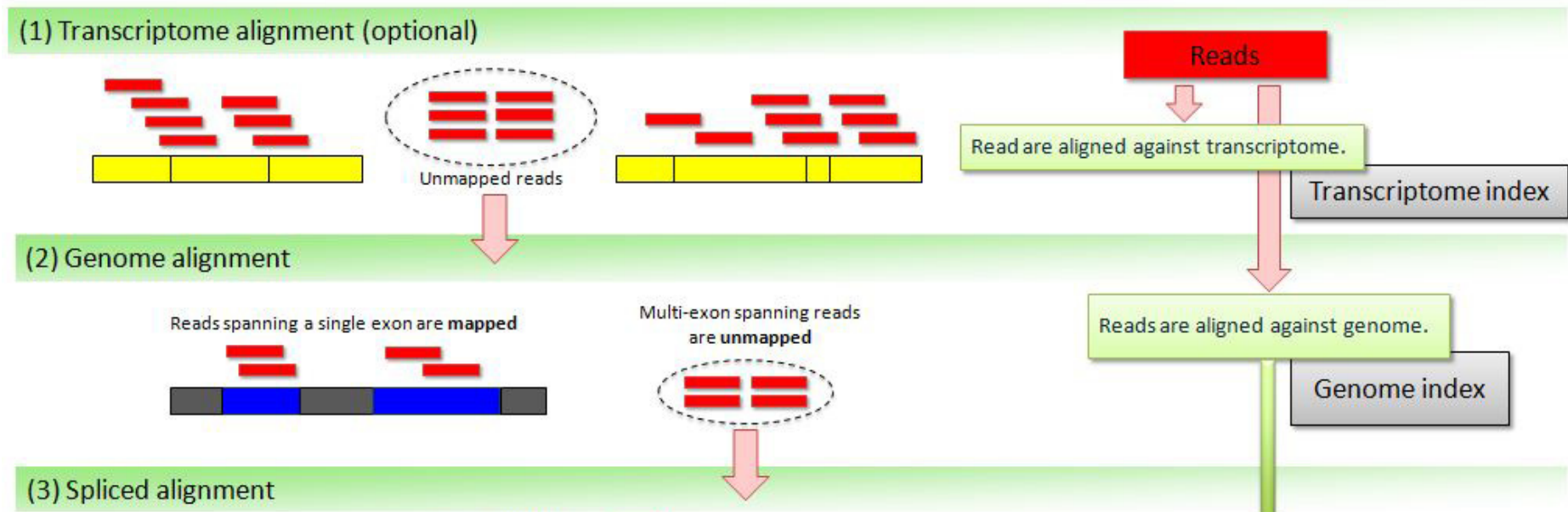


## Strategies:

- Exon-first: fast
- Seed-extend:
  - good with polymorphisms
  - simultaneous spliced/ unspliced mapping



# Tophat2 pipeline

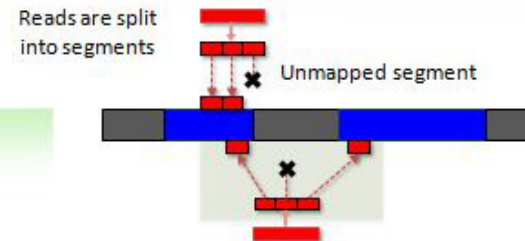




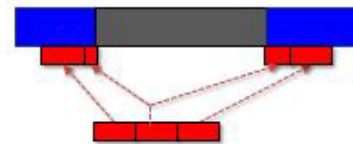
# Tophat2 pipeline

## (3) Spliced alignment

(3-1) Segment alignment to genome



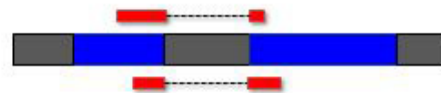
(3-2) Identification of splice sites (including indels and fusion break points)



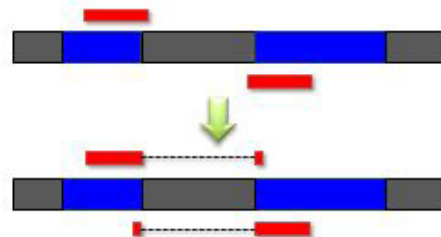
(3-3) Segments aligned to junction flanking sequences



(3-4) Segment alignments stitched together to form whole read alignments



(3-5) Re-alignment of reads minimally overlapping introns



Reads are split into smaller segments which are then aligned to the genome.

Genome index

Segment mappings are used to find potential splice sites usually when the distance between the mapped positions of the left and the right segments are longer than the length of the middle part of a read.

Sequences flanking a splice site are concatenated and segments are aligned to them.

Junction flanking index

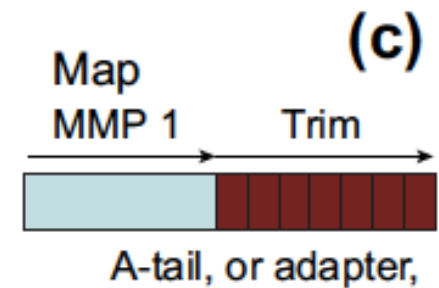
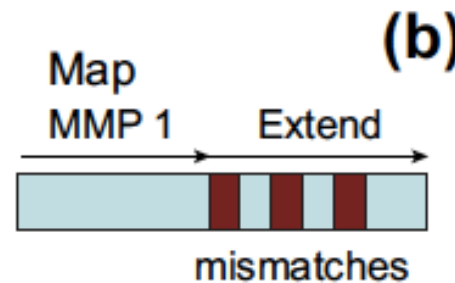
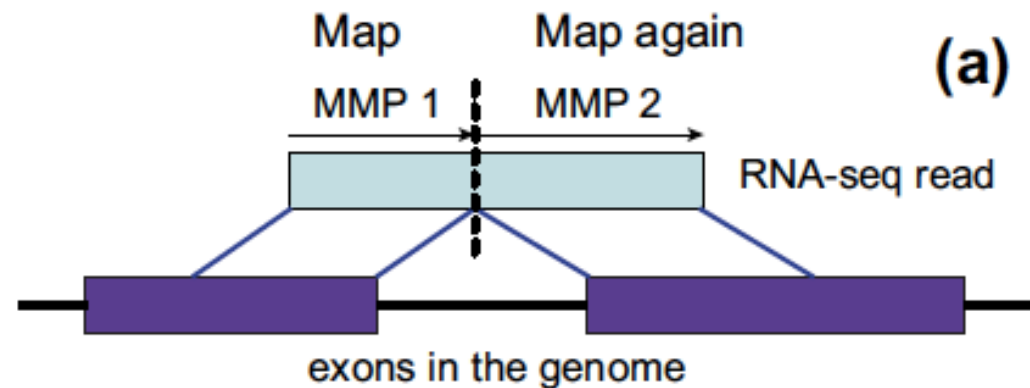
Mapped segments against either genome or flanking sequences are gathered to produce whole read alignments.

Genome mapped reads with alignments extending a few bases into introns are re-aligned to exons instead.



## STAR: universal RNA-seq aligner

- Designed to align the non-contiguous sequences directly to the reference genome
- Steps:
  - Search Maximal Mappable Prefix (MMP)
  - clustering/stitching/scoring





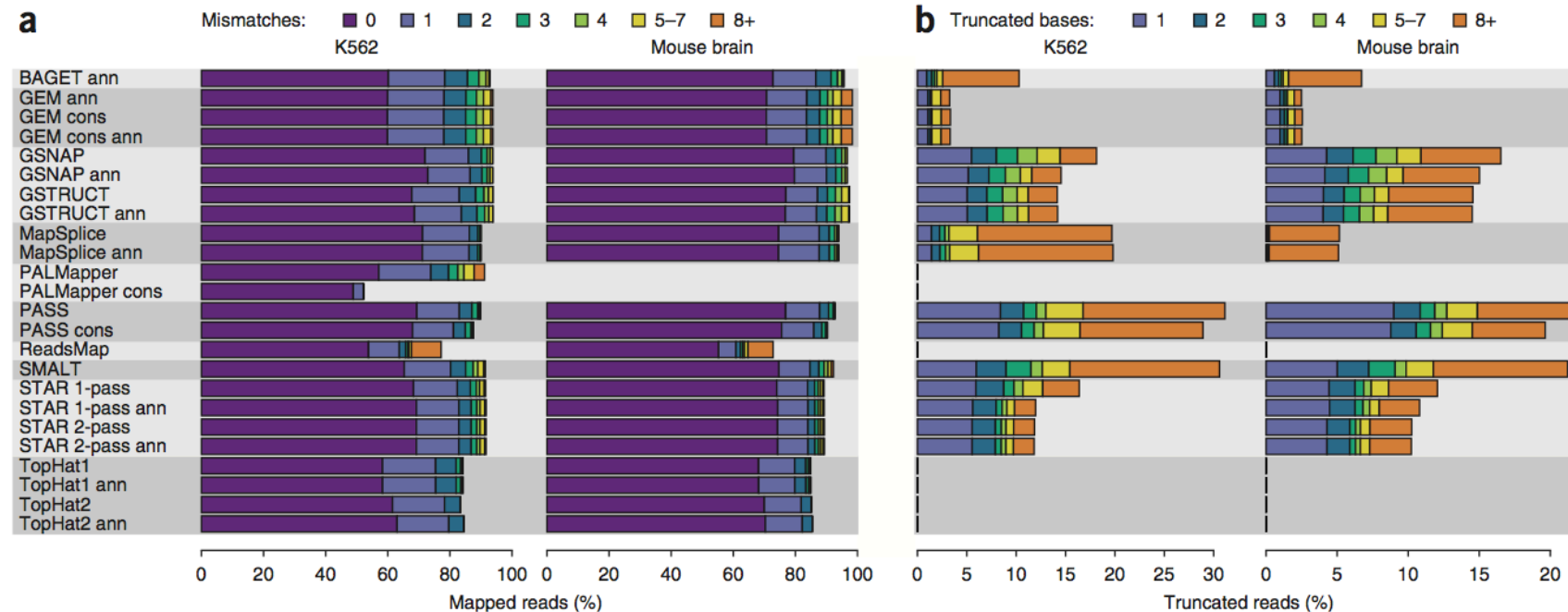
## Comparison of Aligners

- Tradeoff:
  - speed
  - RAM
  - accuracy
  - sensitivity
- <http://www.ecseq.com/support/benchmark>





# Performance comparison of RNA-seq mappers



**Figure 2** | Mismatch and truncation frequencies. (a) Percentage of sequenced reads mapped with the indicated number of mismatches. (b) Percentage of sequenced reads truncated at either or both ends. Bar colors indicate the number of bases removed.

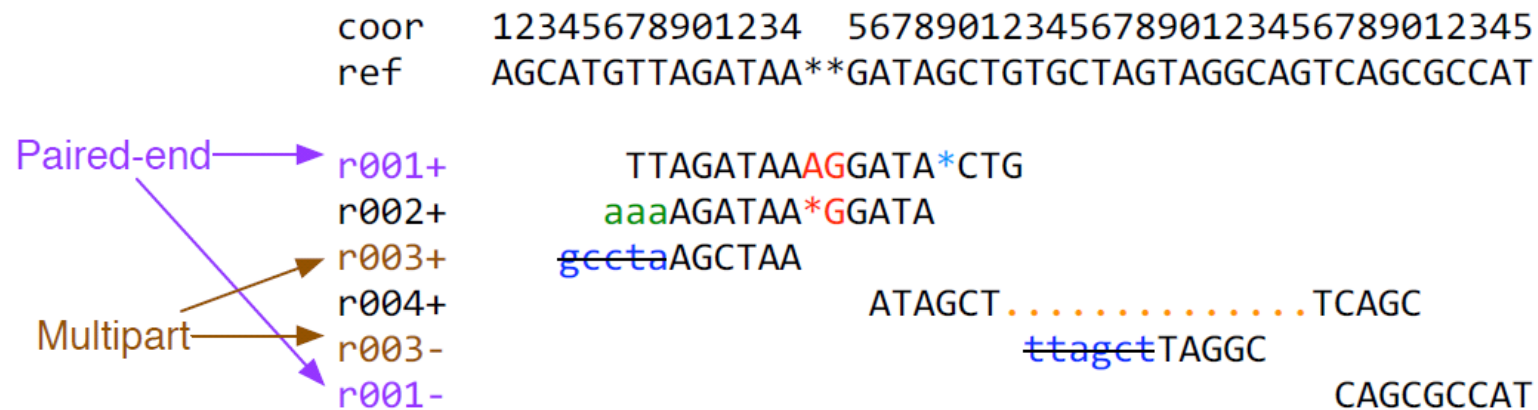


# Sequence / Alignment (SAM) files

- **SAM (Sequence Alignment/Map)**
  - Single unified format for storing read alignments to a reference genome
  - Large plain text file
- **BAM (Binary Alignment/Map)**
  - Binary equivalent of SAM
  - Compressed data plus index (bai)
  - Developed for fast processing/indexing



# An example of read mapping





## Corresponding SAM file

|                  |       |   |                                 |   |       |      |
|------------------|-------|---|---------------------------------|---|-------|------|
|                  | coord | 12345678901234                                  | 5678901234567890123456789012345 |   |       |      |
|                  | ref   | AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT |                                 |   |       |      |
| Paired-end       | r001+ | TTAGATAAAGGATA*CTG                              |                                 |   |       |      |
|                  | r002+ | aaaAGATAA*GGATA                                 |                                 |   |       |      |
| Multipart        | r003+ | gcctaAGCTAA                                     |                                 |   |       |      |
|                  | r004+ | ATAGCT.....TCAGC                                |                                 |   |       |      |
|                  | r003- | ttagctTAGGC                                     |                                 |   |       |      |
|                  | r001- | CAGCGCCAT                                       |                                 |   |       |      |
| @SQ SN:ref LN:45 |       |   |                                 |   |       |      |
| Ins & padding    | r001  | 163   | ref                             | 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTA * |       |      |
| Soft clipping    | r002  | 0   | ref                             | 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *      |       |      |
|                  | r003  | 0   | ref                             | 9 30 5H6M * 0 0 AGCTAA * NM:i:1             |       |      |
| Splicing         | r004  | 0   | ref                             | 16 30 6M14N5M * 0 0 ATAGCTTCAGC *           |       |      |
| Hard clipping    | r003  | 16  | ref                             | 29 30 6H5M * 0 0 TAGGC * NM:i:0             |       |      |
|                  | r001  | 83  | ref                             | 37 30 9M = 7 -39 CAGCGCCAT *                |       |      |
|                  | QNAME | RNAME   | MAPQ?                           | MRNM  | ISIZE | QUAL |
|                  | FLAG  | POS   | CIGAR?                          | MPOS  | SEQ   |      |

Slide modified from Heng Li's presentation



# CIGAR string - compact representation of an alignment

- M - match or mismatch
- I – insertion
- D – deletion
- S - soft clip
  - Clipped sequences stored in SAM
- H - hard clip
  - Clipped sequences not stored in SAM
- N – skipped reference bases, splicing

Match/mismatch, indels

Ref: ACGCA**B**TG--GT

Read: ATGCA-TG**C**AGT

Cigar: 5M1D2M2I2M

Soft clipping

REF: **A**TCGTGTAACCTGACTAGT**TAA**

READ: **ggg**GTGTAACC-GACTAG**ggg**

Cigar: 3S8M1D5M4S

Hard clipping

REF: **A**TCGTGTAACCTGACTAGT**TAA**

READ: **ggg**GTGTAACC-GACTAG**ggg**

Cigar: 3H8M1D5M4H