Short Read Alignment Algorithms

Raluca Gordân

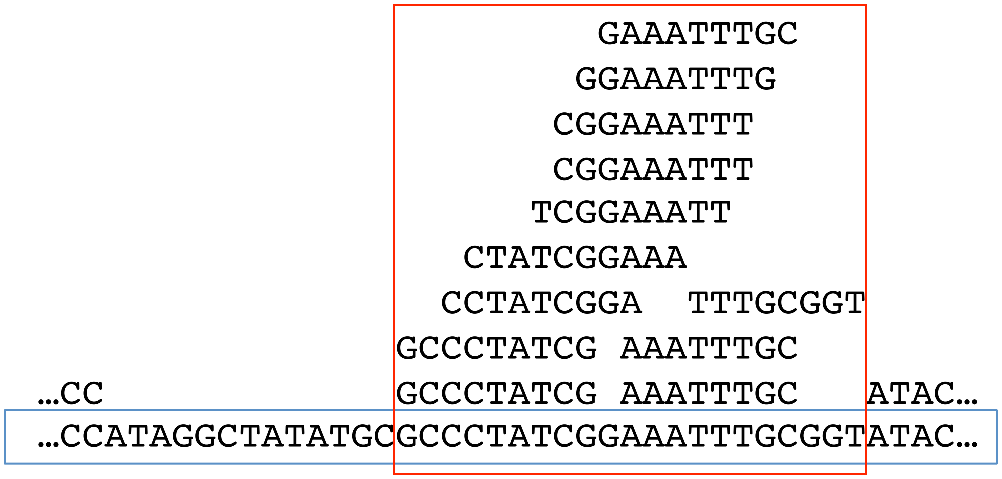
Department of Biostatistics and Bioinformatics
Department of Computer Science
Department of Molecular Genetics and Microbiology
Center for Genomic and Computational Biology
Duke University

July 23, 2018

Sequencing applications

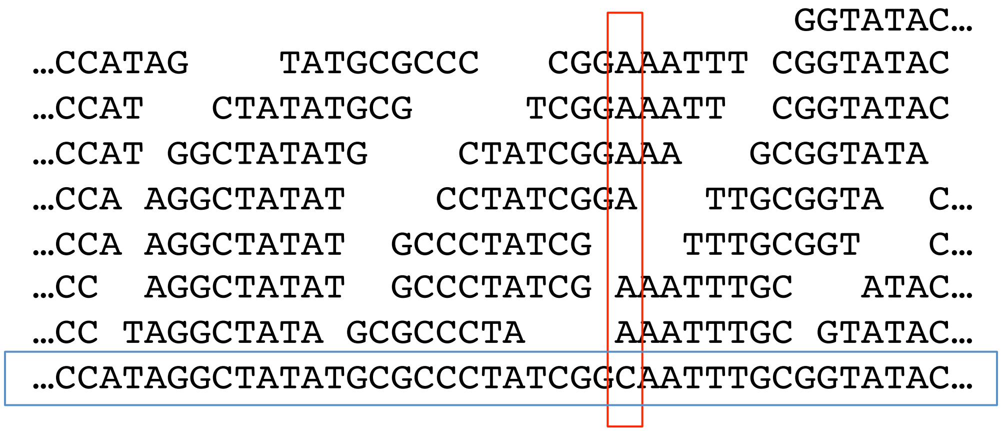
RNA-seq – this course

ChIP-seq: identify and measure significant peaks

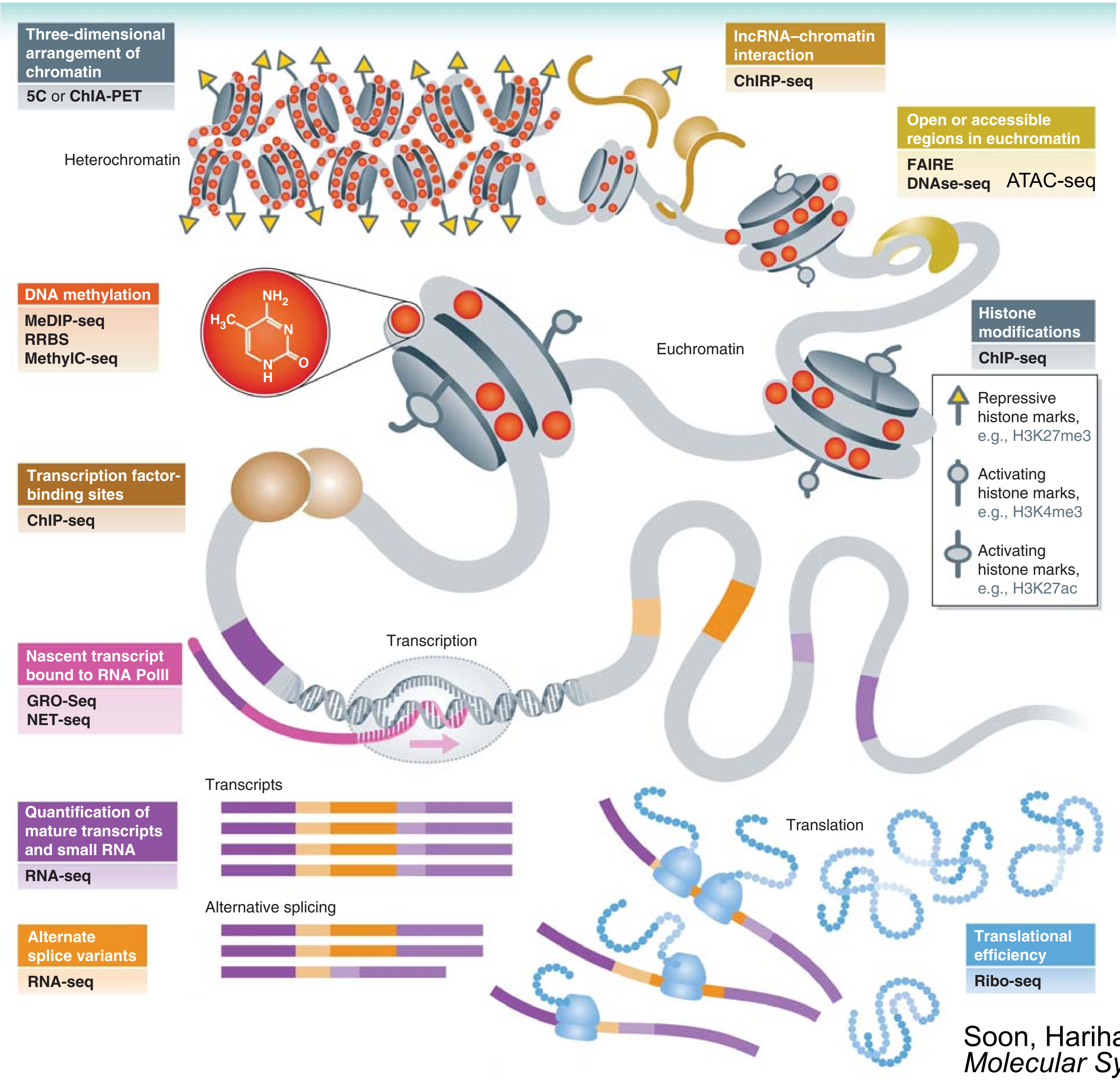


```
                         GAAATTTGC
                        GGAAATTTG
                       CGGAAATTT
                       CGGAAATTT
                      TCGGAAATT
                   CTATCGGAAA
                  CCTATCGGA  TTTGCGGT
                GCCCTATCG AAATTTGC
…CC             GCCCTATCG AAATTTGC   ATAC…
…CCATAGGCTATATGCGCCCTATCGGAAATTTGCGGTATAC…
```

Genotyping: identify variations



```
                                  GGTATAC…
…CCATAG    TATGCGCCC   CGGAAATTT CGGTATAC
…CCAT   CTATATGCG     TCGGAAATT  CGGTATAC
…CCAT GGCTATATG    CTATCGGAAA   GCGGTATA
…CCA AGGCTATAT    CCTATCGGA   TTGCGGTA  C…
…CCA AGGCTATAT  GCCCTATCG    TTTGCGGT   C…
…CC  AGGCTATAT  GCCCTATCG AAATTTGC   ATAC…
…CC TAGGCTATA GCGCCCTA    AAATTTGC GTATAC…
…CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC…
```



Three-dimensional arrangement of chromatin
5C or ChIA-PET

lncRNA–chromatin interaction
ChIRP-seq

Open or accessible regions in euchromatin
FAIRE
DNAse-seq ATAC-seq

DNA methylation
MeDIP-seq
RRBS
MethylC-seq

Histone modifications
ChIP-seq

Repressive histone marks, e.g., H3K27me3

Activating histone marks, e.g., H3K4me3

Activating histone marks, e.g., H3K27ac

Transcription factor-binding sites
ChIP-seq

Nascent transcript bound to RNA PolII
GRO-Seq
NET-seq

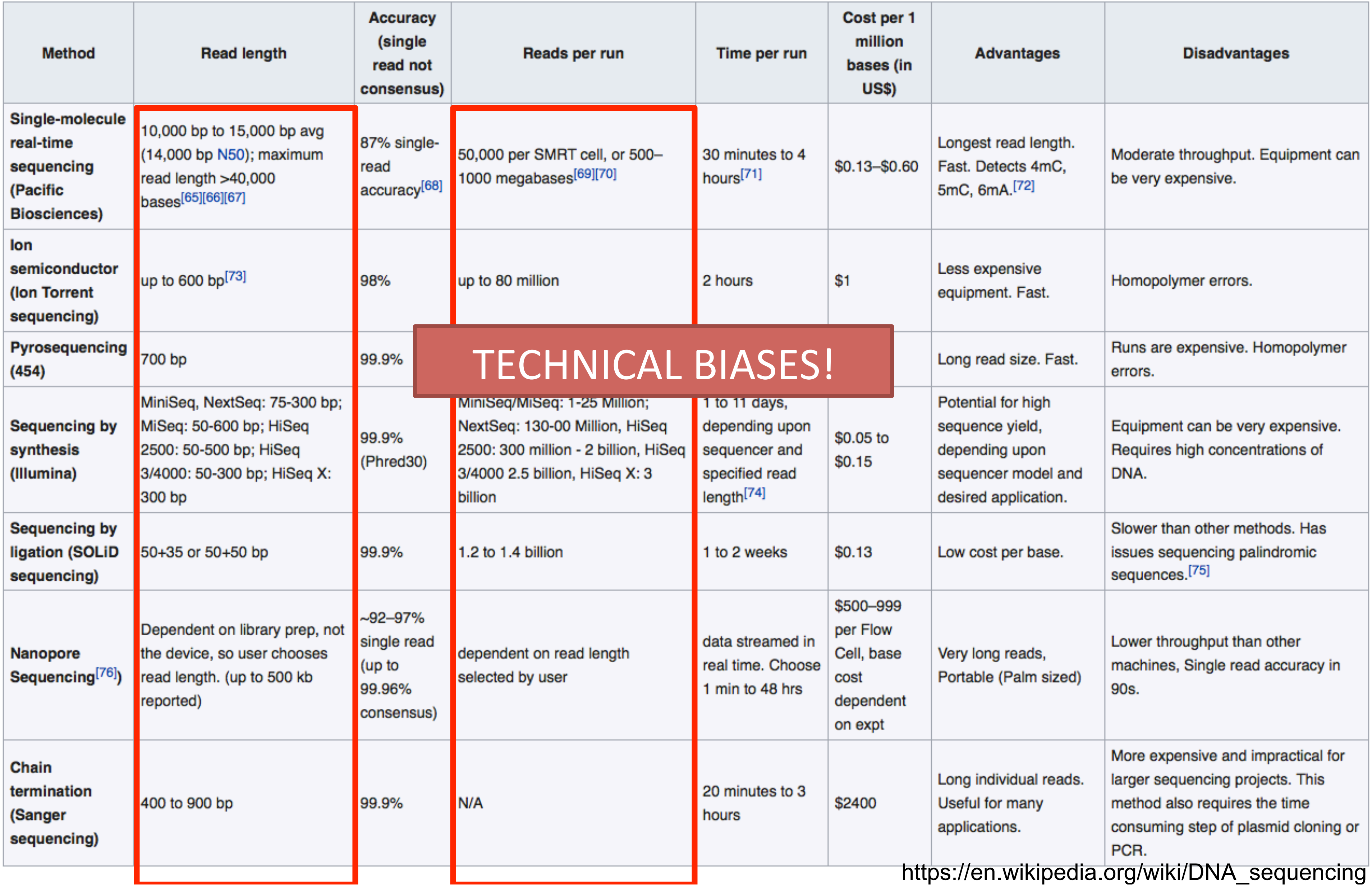
Quantification of mature transcripts and small RNA
RNA-seq

Alternate splice variants
RNA-seq

Translational efficiency
Ribo-seq

Soon, Hariharan, Snyder
Molecular Systems Biology 2012

Sequencing technologies



Method	Read length	Accuracy (single read not consensus)	Reads per run	Time per run	Cost per 1 million bases (in US$)	Advantages	Disadvantages
Single-molecule real-time sequencing (Pacific Biosciences)	10,000 bp to 15,000 bp avg (14,000 bp N50); maximum read length >40,000 bases[65][66][67]	87% single-read accuracy[68]	50,000 per SMRT cell, or 500–1000 megabases[69][70]	30 minutes to 4 hours[71]	$0.13–$0.60	Longest read length. Fast. Detects 4mC, 5mC, 6mA.[72]	Moderate throughput. Equipment can be very expensive.
Ion semiconductor (Ion Torrent sequencing)	up to 600 bp[73]	98%	up to 80 million	2 hours	$1	Less expensive equipment. Fast.	Homopolymer errors.
Pyrosequencing (454)	700 bp	99.9%	[illegible]	[illegible]	[illegible]	Long read size. Fast.	Runs are expensive. Homopolymer errors.
Sequencing by synthesis (Illumina)	MiniSeq, NextSeq: 75-300 bp; MiSeq: 50-600 bp; HiSeq 2500: 50-500 bp; HiSeq 3/4000: 50-300 bp; HiSeq X: 300 bp	99.9% (Phred30)	MiniSeq/MiSeq: 1-25 Million; NextSeq: 130-00 Million, HiSeq 2500: 300 million - 2 billion, HiSeq 3/4000 2.5 billion, HiSeq X: 3 billion	1 to 11 days, depending upon sequencer and specified read length[74]	$0.05 to $0.15	Potential for high sequence yield, depending upon sequencer model and desired application.	Equipment can be very expensive. Requires high concentrations of DNA.
Sequencing by ligation (SOLiD sequencing)	50+35 or 50+50 bp	99.9%	1.2 to 1.4 billion	1 to 2 weeks	$0.13	Low cost per base.	Slower than other methods. Has issues sequencing palindromic sequences.[75]
Nanopore Sequencing[76])	Dependent on library prep, not the device, so user chooses read length. (up to 500 kb reported)	~92–97% single read (up to 99.96% consensus)	dependent on read length selected by user	data streamed in real time. Choose 1 min to 48 hrs	$500–999 per Flow Cell, base cost dependent on expt	Very long reads, Portable (Palm sized)	Lower throughput than other machines, Single read accuracy in 90s.
Chain termination (Sanger sequencing)	400 to 900 bp	99.9%	N/A	20 minutes to 3 hours	$2400	Long individual reads. Useful for many applications.	More expensive and impractical for larger sequencing projects. This method also requires the time consuming step of plasmid cloning or PCR.

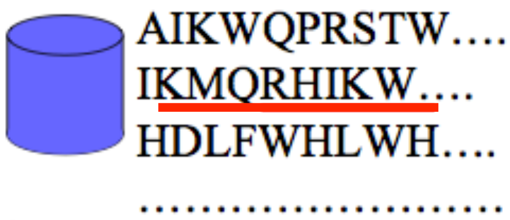
TECHNICAL BIASES!

https://en.wikipedia.org/wiki/DNA_sequencing

Sequence alignment

Heuristic local alignment
(**BLAST**)

- INPUT:
  - Database



AIKWQPRSTW….
IKMQRHIKW….
HDLFWHLWH….
……………………

  - Query: PSKMQRGIKWLLP
- OUTPUT:
  - sequences similar to query

Global/local alignment
(Needleman-Wunsch,
Smith-Waterman)

- INPUT:
  - Two sequences
    $X = x_1x_2\ldots x_m$
    $Y = y_1y_2\ldots y_n$
- OUTPUT:
  - Optimal alignment between X and Y (or substrings of X and Y)

Short read alignment

- INPUT:
  - A few million short reads, with certain error characteristics (specific to the sequencing platform)
    - Illumina: few errors, mostly substitutions
  - A reference genome
- OUTPUT:
  - Alignments of the reads to the reference genome

- Can we use BLAST?
  - Assuming BLAST returns the result for a read in 1 sec
  - For 10 million reads: 10 million seconds = 116 days

- Algorithms for exact string matching are more appropriate

Algorithms for exact string matching

- Search for the substring ANA in the string BANANA

Brute Force

```
BANANA
BAN
 ANA
  NAN
   ANA
```

Naive

Slow & Easy

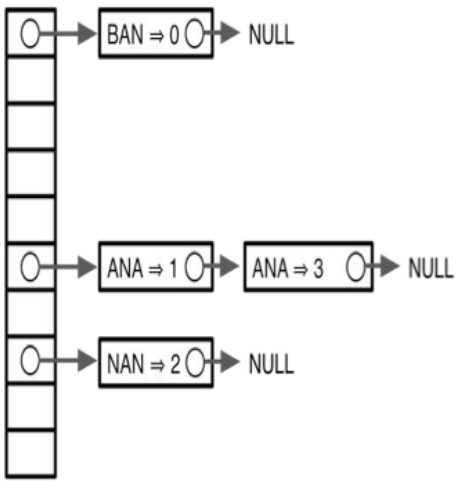
Suffix Array

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Binary Search

PacBio Aligner (BLASR); Bowtie

Hash (Index) Table



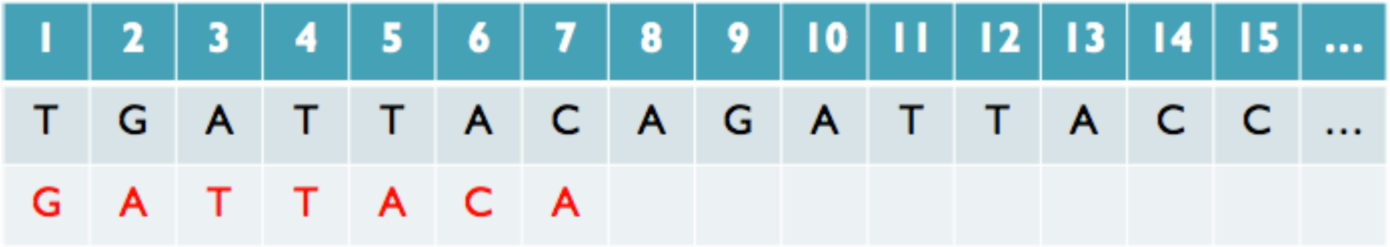
Seed-and-extend

BLAST, BLAT

Time complexity versus space complexity

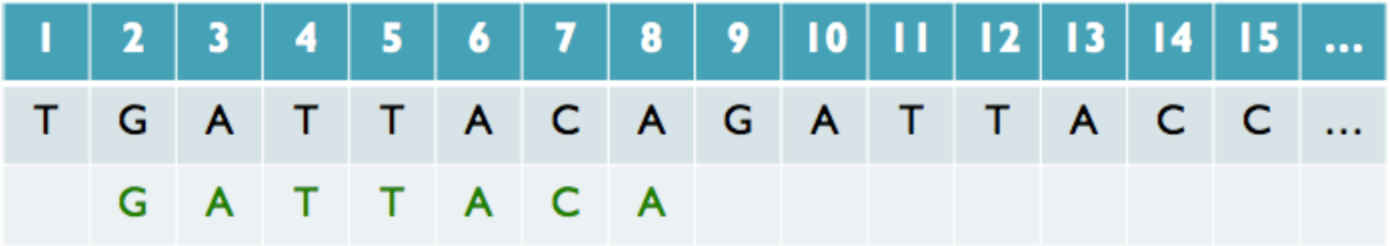
Brute force search for GATTACA

- Where is GATTACA in the human genome?



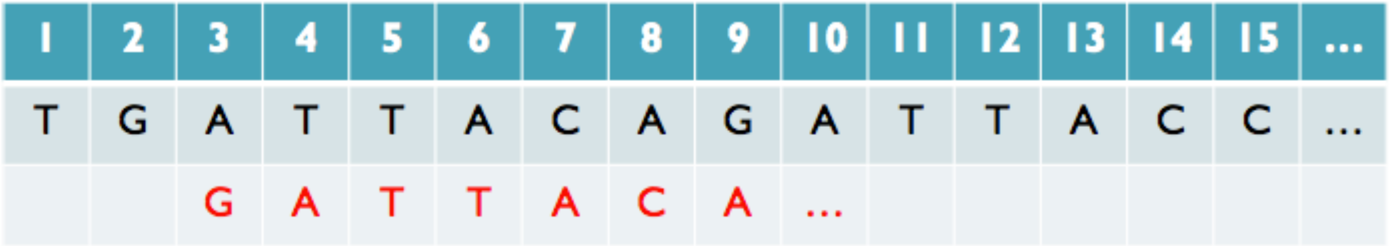
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
G	A	T	T	A	C	A									

No match at offset 1



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
	G	A	T	T	A	C	A								

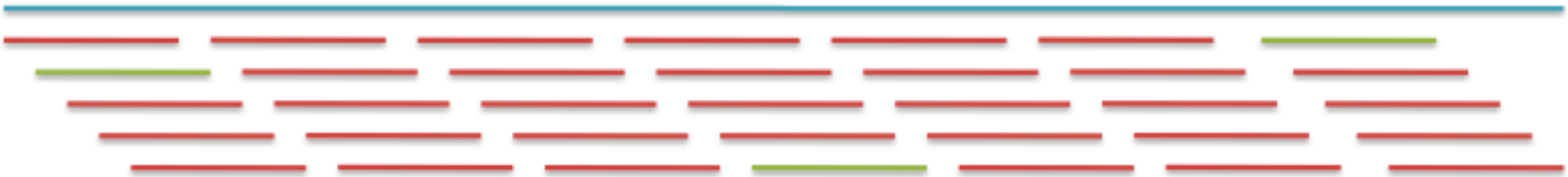
Match at offset 2



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
		G	A	T	T	A	C	A	...						

No match at offset 3...

Brute force search for GATTACA

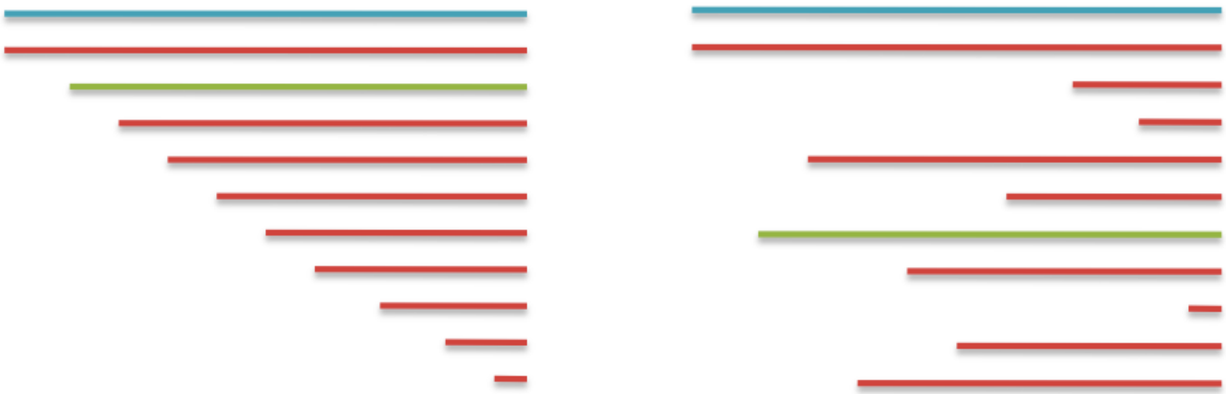


- Simple, easy to understand
- Analysis
  – Genome length = n = 3,000,000,000
  – Query length = m = 7
  – Comparisons: (n-m+1) * m = 21,000,000,000
- Assuming each comparison takes 1/1,000,000 of a second…
- … the total running time is 21,000 seconds = 0.24 days
- … for one 7-bp read

Suffix arrays

- Preprocess the genome
  - Sort all the suffixes of the genome

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
	G	A	T	T	A	C	A								



Split into suffixes

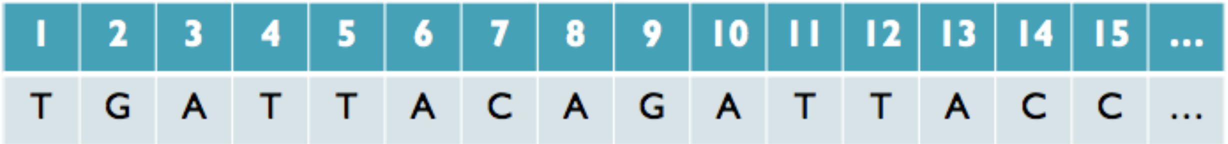
Sort suffixes alphabetically

Suffix array

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

- Use binary search

Suffix arrays



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...

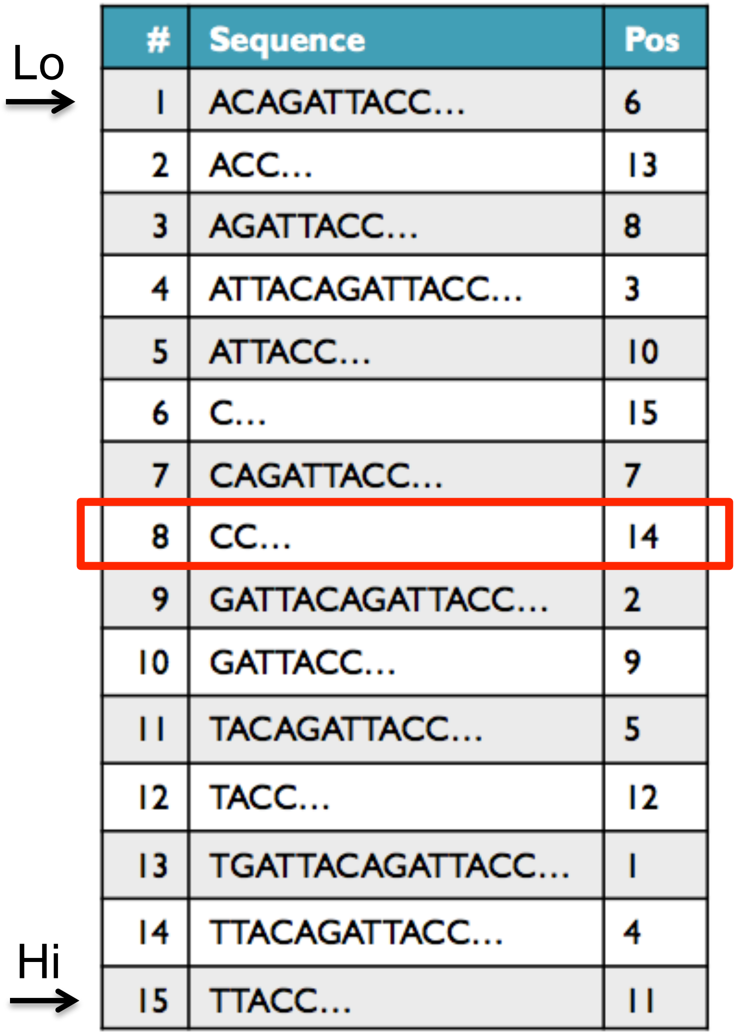
Lo = 1; Hi = 15

Mid = (1+15)/2 = 8

Middle = Suffix[8] = CC

Compare GATTACA to CC => Higher

Lo = Mid + 1



Lo

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Hi

Suffix arrays - search for GATTACA

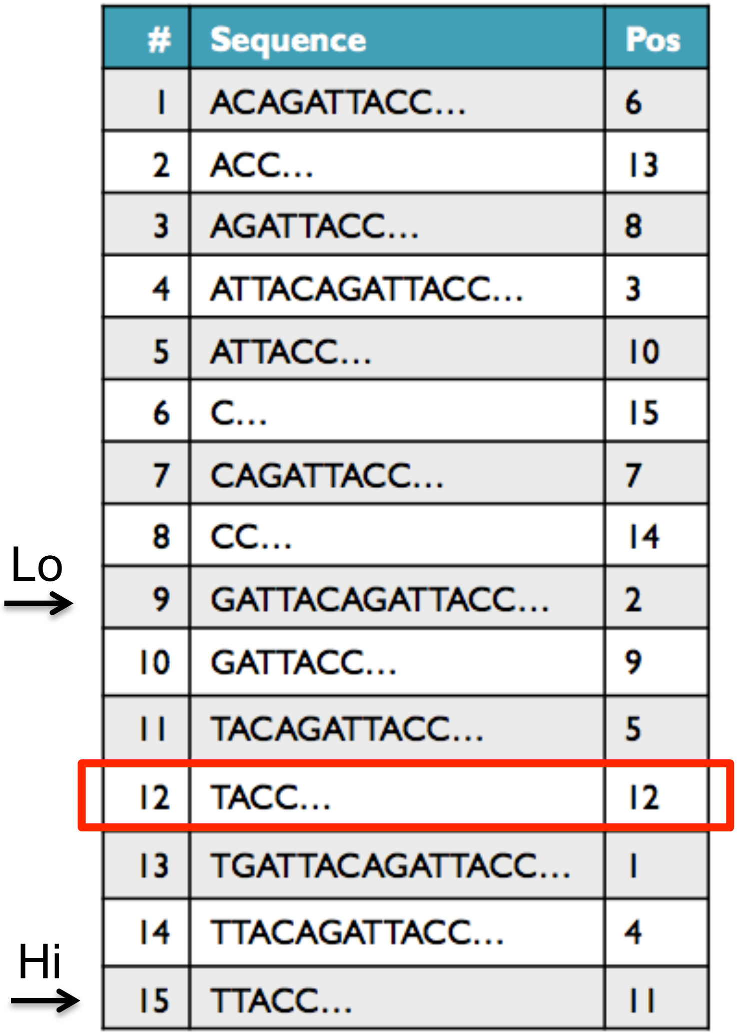
Lo = 9; Hi = 15

Mid = (9+15)/2 = 12

Middle = Suffix[12] = TACC

Compare GATTACA to TACC => Lower

Hi = Mid - 1



#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Suffix arrays - search for GATTACA

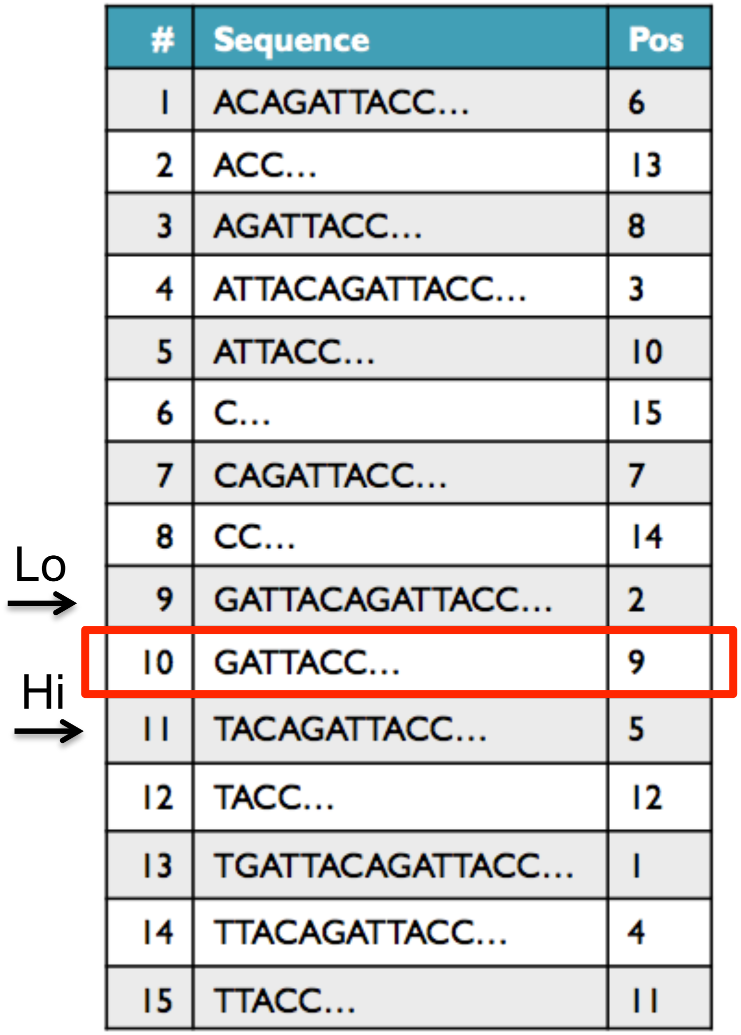
Lo = 9; Hi = 11

Mid = (9+11)/2 = 10

Middle = Suffix[10] = GATTACC

Compare GATTACA to GATTACC => Lower

Hi = Mid - 1



#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Suffix arrays - search for GATTACA

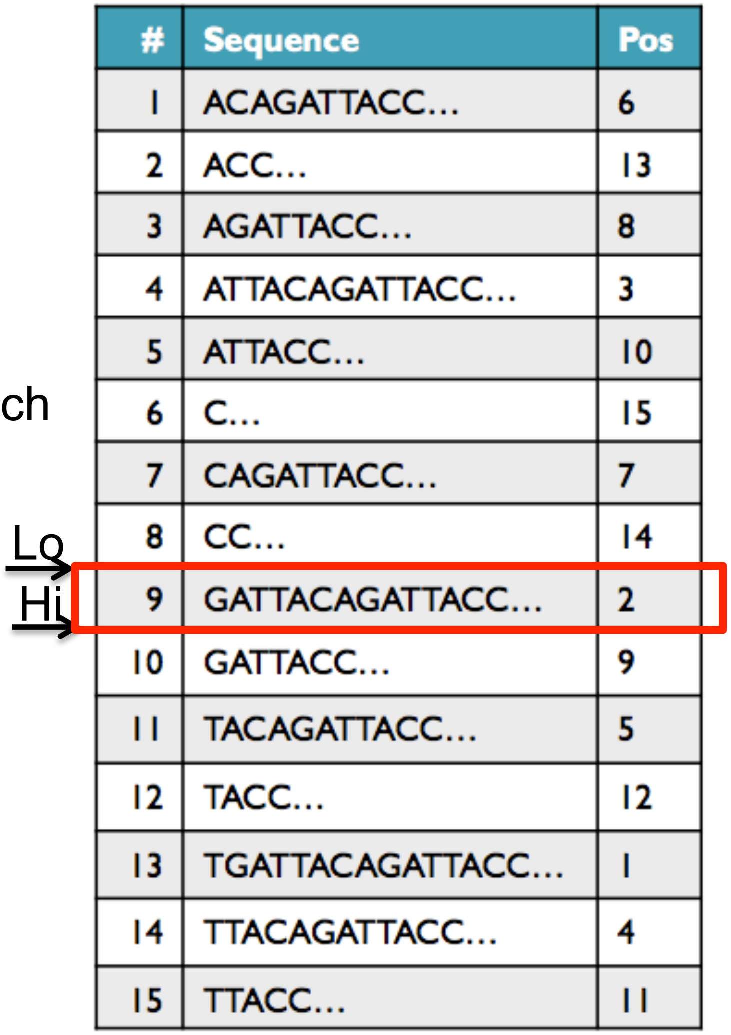
Lo = 9; Hi = 9

Mid = (9+9)/2 = 9

Middle = Suffix[9] = GATTACAG…

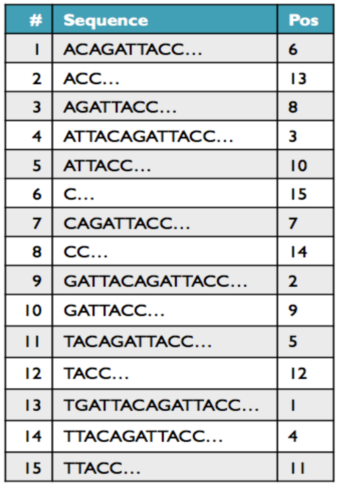
Compare GATTACA to GATTACAG… => Match

Return: match at position 2



#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Suffix arrays - analysis



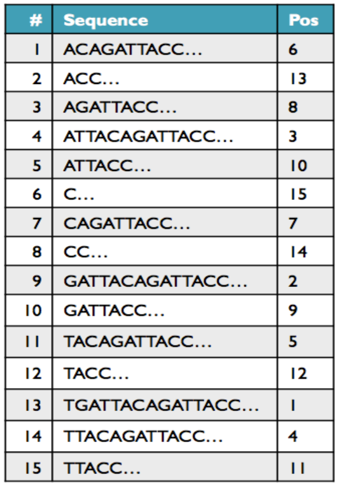
#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

- Word (query) of size **m = 7**
- Genome of size **n = 3,000,000,000**
- Bruce force:
  - approx. **m x n = 21,000,000,000** comparisons
- Suffix arrays:
  - approx. **m x $\log_2$(n) = 7 x 32 = 224** comparisons

- Assuming each comparison takes 1/1,000,000 of a second…
- … the total running time is **0.000224 seconds** for one 7-bp read
- Compared to **0.24 days** for one 7-bp read in the case of brute force search

- For 10 million reads, the suffix array search would take
  2240 seconds = **37 minutes**

Suffix arrays - analysis



#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

- Word (query) of size **m = 7**
- Genome of size **n = 3,000,000,000**

- For 10 million reads, the suffix array search would take 2240 seconds = **37 minutes**

- Problem? Time complexity versus space complexity

- Total characters in all suffixes combined:

  1+2+3+…+n = n(n+1)/2

- For the human genome:

  4.5 billion billion characters!!!

Algorithms for exact string matching

- Search for the substring ANA in the string BANANA

Brute Force

```
BANANA
BAN
 ANA
  NAN
   ANA
```

Naive

Slow & Easy

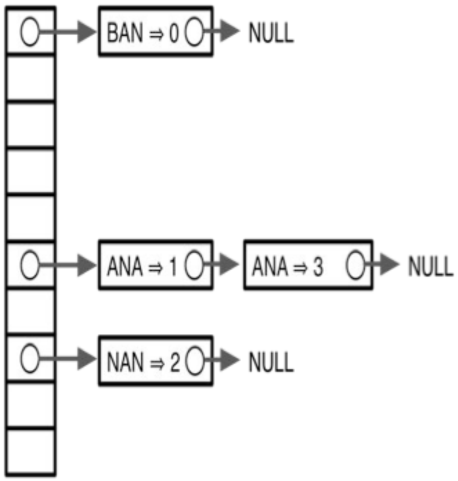
Suffix Array

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Binary Search

PacBio Aligner (BLASR); Bowtie

Hash (Index) Table



Seed-and-extend

BLAST, BLAT

Time complexity versus space complexity

Hashing

- Where is GATTACA in the human genome?

  - Build an inverted index of every k-mer in the genome

- How do we access the table?

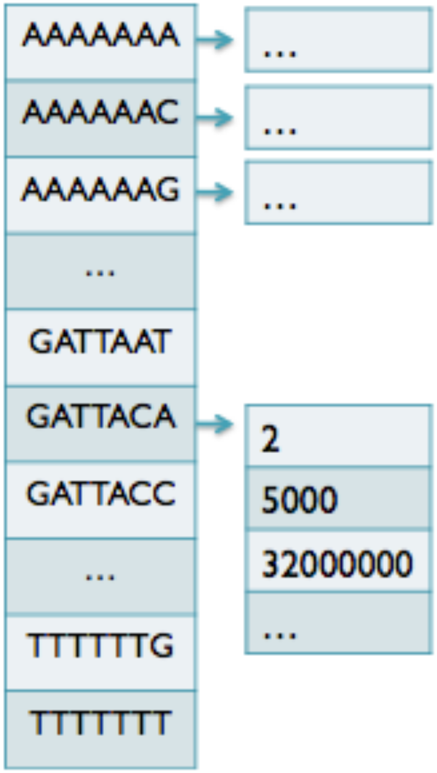
  - We can only use numbers to index

  - Encode sequences as numbers

  Simple: A=0,C=1,G=2,T=3 => GATTACA=2,033,010

  Smart: A = 00_2, C = 01_2, G = 10_2, T = 11_2
  => GATTACA=10001111000100_2=9156_{10}

- Lookup: very fast
- But constructing an optimal hash is tricky


AAAAAAA → ...
AAAAAAC → ...
AAAAAAG → ...
...
GATTAAT
GATTACA → 2, 5000, 32000000, ...
GATTACC
...
TTTTTTG
TTTTTTT

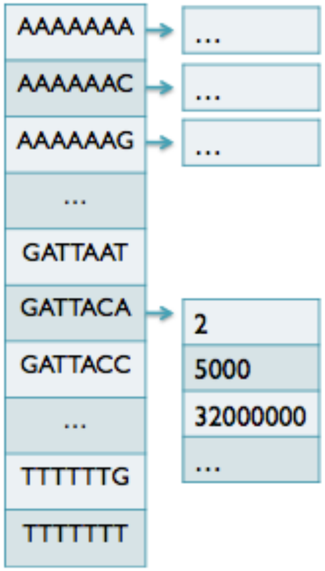
Hashing

- Number of possible sequences of length k is 4^k
- K=7 => 4^7 = 16,384 (easy to store)
- K=20 => 4^{20} = 1,099,511,627,776 (impossible to store directly in RAM)

  - There are only 3B 20-mers in the genome

  - Even if we could build this table, 99.7% will be empty

  - But we don't know which cells are empty until we try



AAAAAAA → ...

AAAAAAC → ...

AAAAAAG → ...

...

GATTAAT

GATTACA → 2, 5000, 32000000, ...

GATTACC

...

TTTTTTG

TTTTTTT

Hashing

- Number of possible sequences of length k is 4^k
- K=7 => 4^7 = 16,384 (easy to store)
- K=20 => 4^{20} = 1,099,511,627,776 (impossible to store directly in RAM)

  - There are only 3B 20-mers in the genome

  - Even if we could build this table, 99.7% will be empty

  - But we don't know which cells are empty until we try
- Use a **hash function** to shrink the possible range

  - Maps a number **n** in [**0,R**] to **h** in [**0,H**]

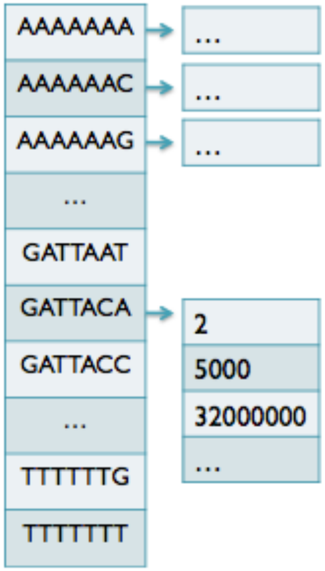
    - Use **128** buckets instead of **16,384**

  - Division: hash(n) = H*n/R;

    - hash(GATTACA)= 128 * 9156/16384 = 71

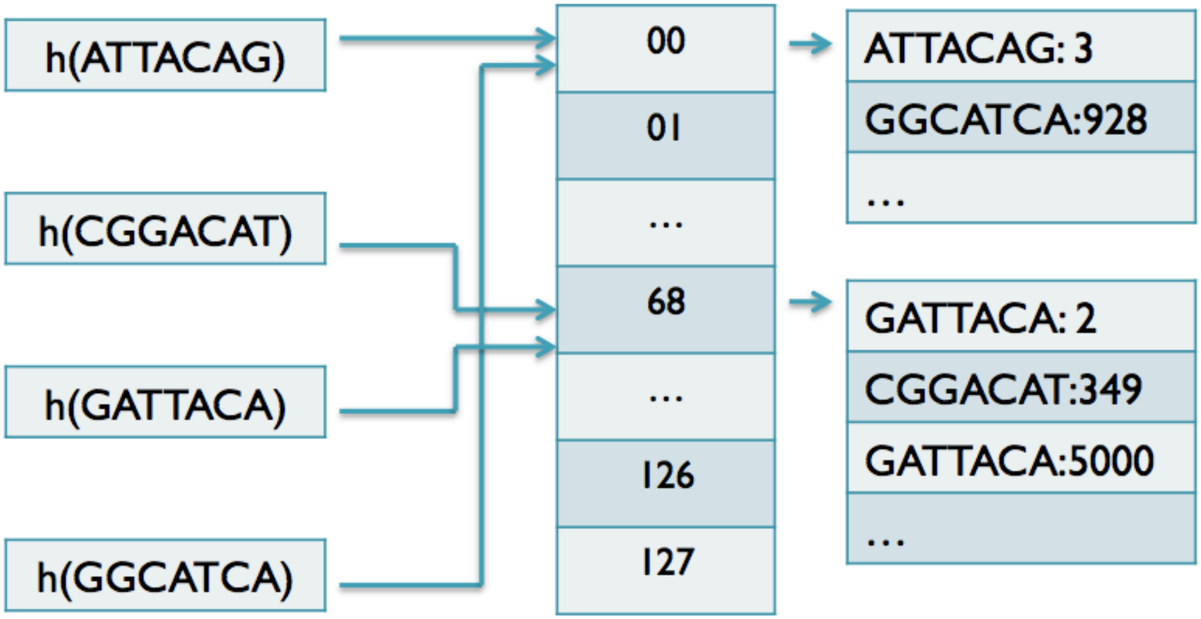
  - Modulo: hash(n) = n % H

    - hash(GATTACA)= 9156 % 128 = 68



Hashing

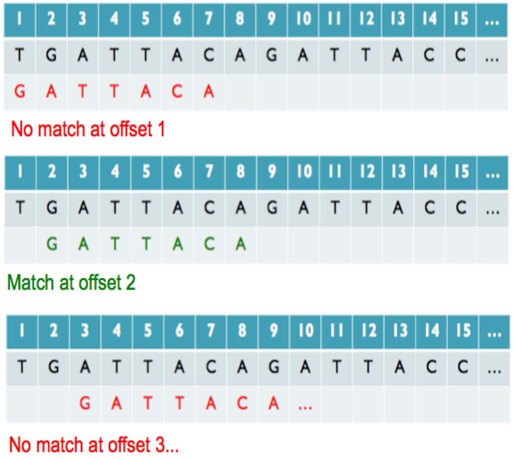
- By construction, **multiple keys have the same hash value**
  - Store elements with the same key in a bucket chained together
    - A good hash evenly distributes the values: R/H have the same hash value
  - Looking up a value scans the entire bucket



Algorithms for exact string matching

- Search for the substring GATTACA in the genome

Brute Force



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
G	A	T	T	A	C	A									

No match at offset 1

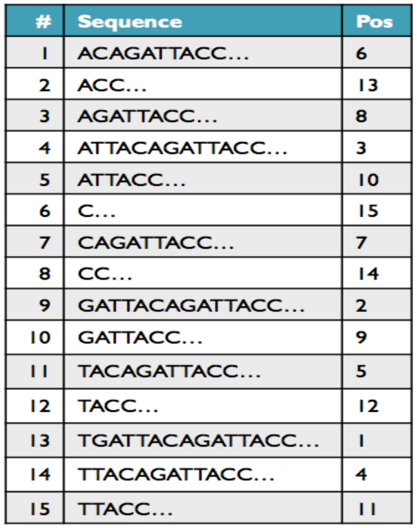
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
	G	A	T	T	A	C	A								

Match at offset 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	...
		G	A	T	T	A	C	A	...						

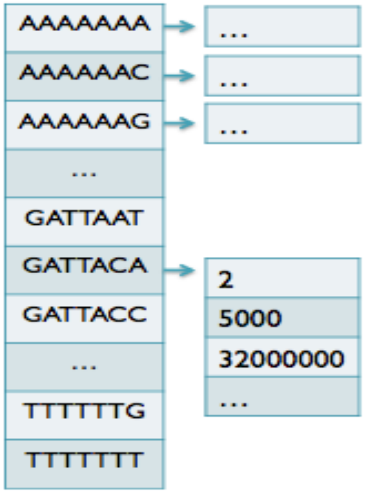
No match at offset 3...

Suffix Array



#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Hash (Index) Table



Easy

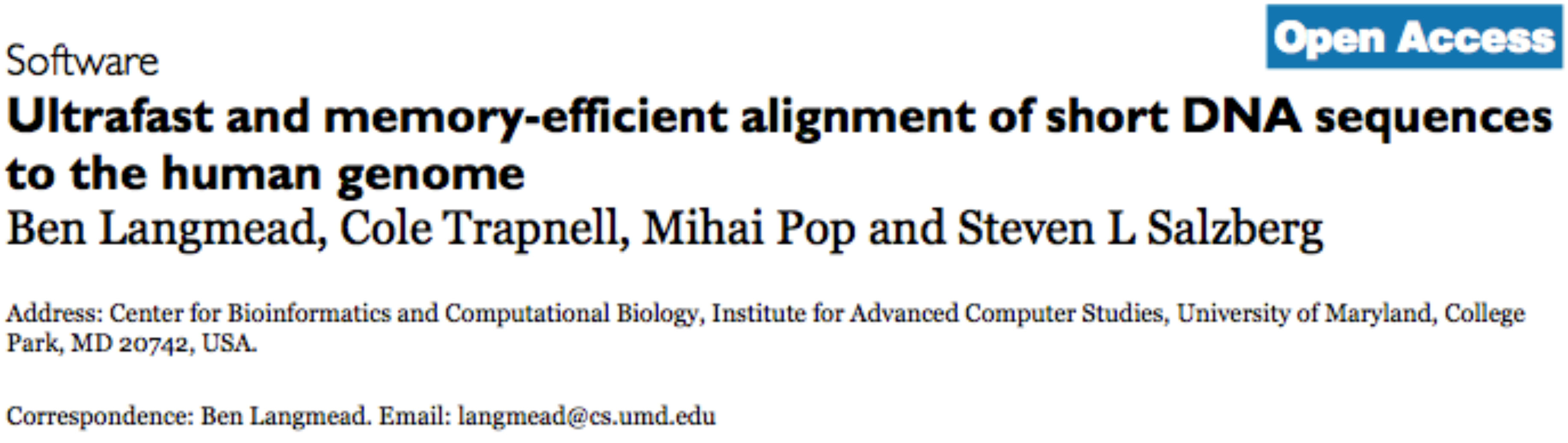
Slow

Fast (binary search)

High space complexity

Fast

Tricky to develop hash function



Software

Open Access

Ultrafast and memory-efficient alignment of short DNA sequences to the human genome

Ben Langmead, Cole Trapnell, Mihai Pop and Steven L Salzberg

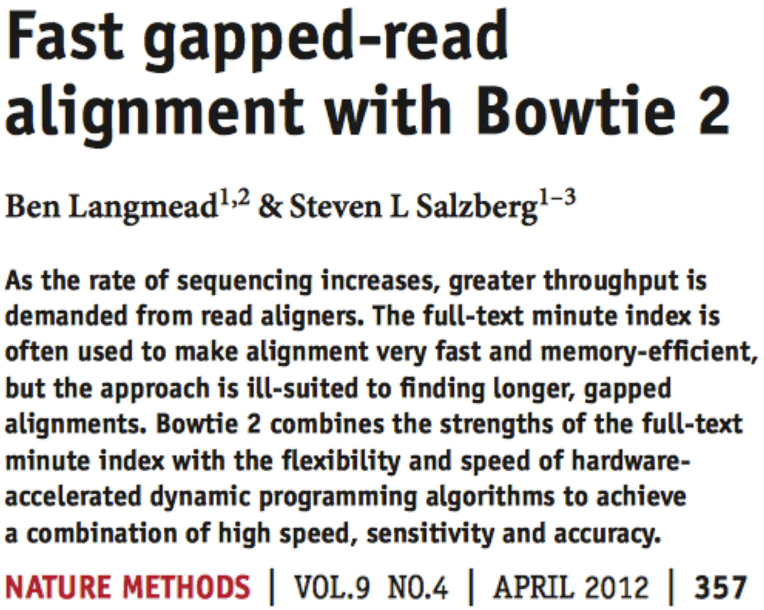
Address: Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA.

Correspondence: Ben Langmead. Email: langmead@cs.umd.edu

Published: 4 March 2009

Genome ***Biology*** 2009, **10:**R25 (doi:10.1186/gb-2009-10-3-r25)

Bowtie is an ultrafast, memory-efficient alignment program for aligning short DNA sequence reads to large genomes. For the human genome, Burrows-Wheeler indexing allows Bowtie to **align more than 25 million reads per CPU hour with a memory footprint of approximately 1.3 gigabytes**. Bowtie extends previous Burrows-Wheeler techniques with a novel quality-aware backtracking algorithm that permits **mismatches**. Multiple processor cores can be used simultaneously to achieve even greater alignment speeds. Bowtie is open source http://bowtie.cbcb.umd.edu.



Fast gapped-read alignment with Bowtie 2

Ben Langmead[1,2] & Steven L Salzberg[1–3]

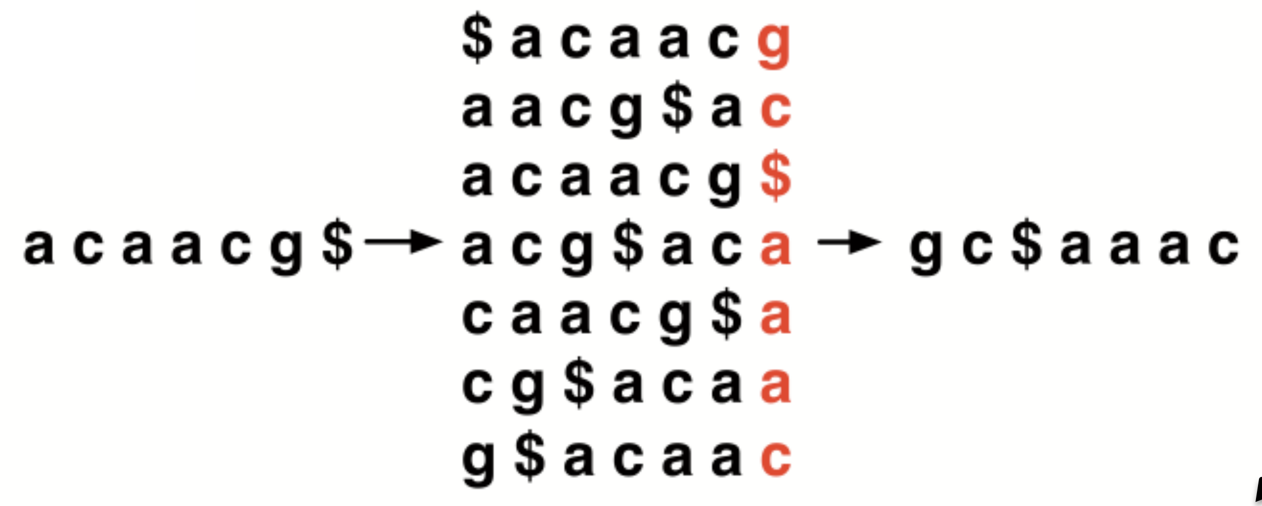
As the rate of sequencing increases, greater throughput is demanded from read aligners. The full-text minute index is often used to make alignment very fast and memory-efficient, but the approach is ill-suited to finding longer, gapped alignments. Bowtie 2 combines the strengths of the full-text minute index with the flexibility and speed of hardware-accelerated dynamic programming algorithms to achieve a combination of high speed, sensitivity and accuracy.

NATURE METHODS | VOL.9 NO.4 | APRIL 2012 | 357

- Bowtie indexes the genome using a scheme based on the Burrows-Wheeler transform (**BWT**) and the Ferragina-Manzini (**FM**) index

Burrows-Wheeler transform

- The BWT is a reversible permutation of the characters in a text
- BWT-based indexing allows large texts to be searched efficiently in a small memory footprint



```
                $acaacg
                aacg$ac
                acaacg$
acaacg$  →      acg$aca  →  gc$aaac
                caacg$a
                cg$acaa
                g$acaac
```

All cyclic rotations of T$, sorted lexicographically

Same size as T

T

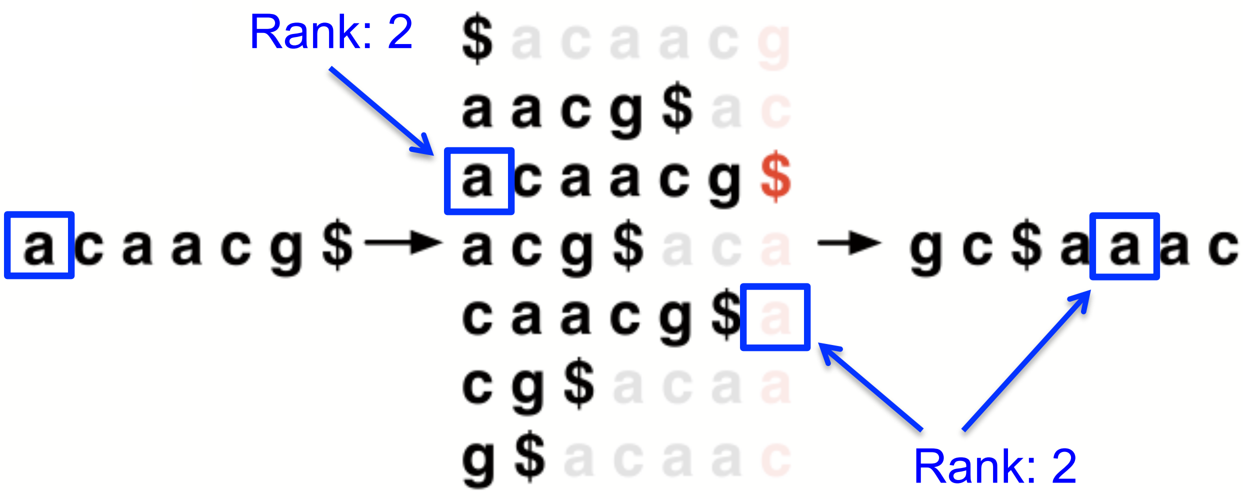
Burrows-Wheeler matrix of T

Burrows-Wheeler transform of T: BWT(T)

Last first (LF) mapping

- The BW matrix has a property called **last first (LF) mapping**:

  The ith occurrence of character X in the last column corresponds to the same text character as the ith occurrence of X in the first column
- This property is at the core of algorithms that use the BWT index to search the text



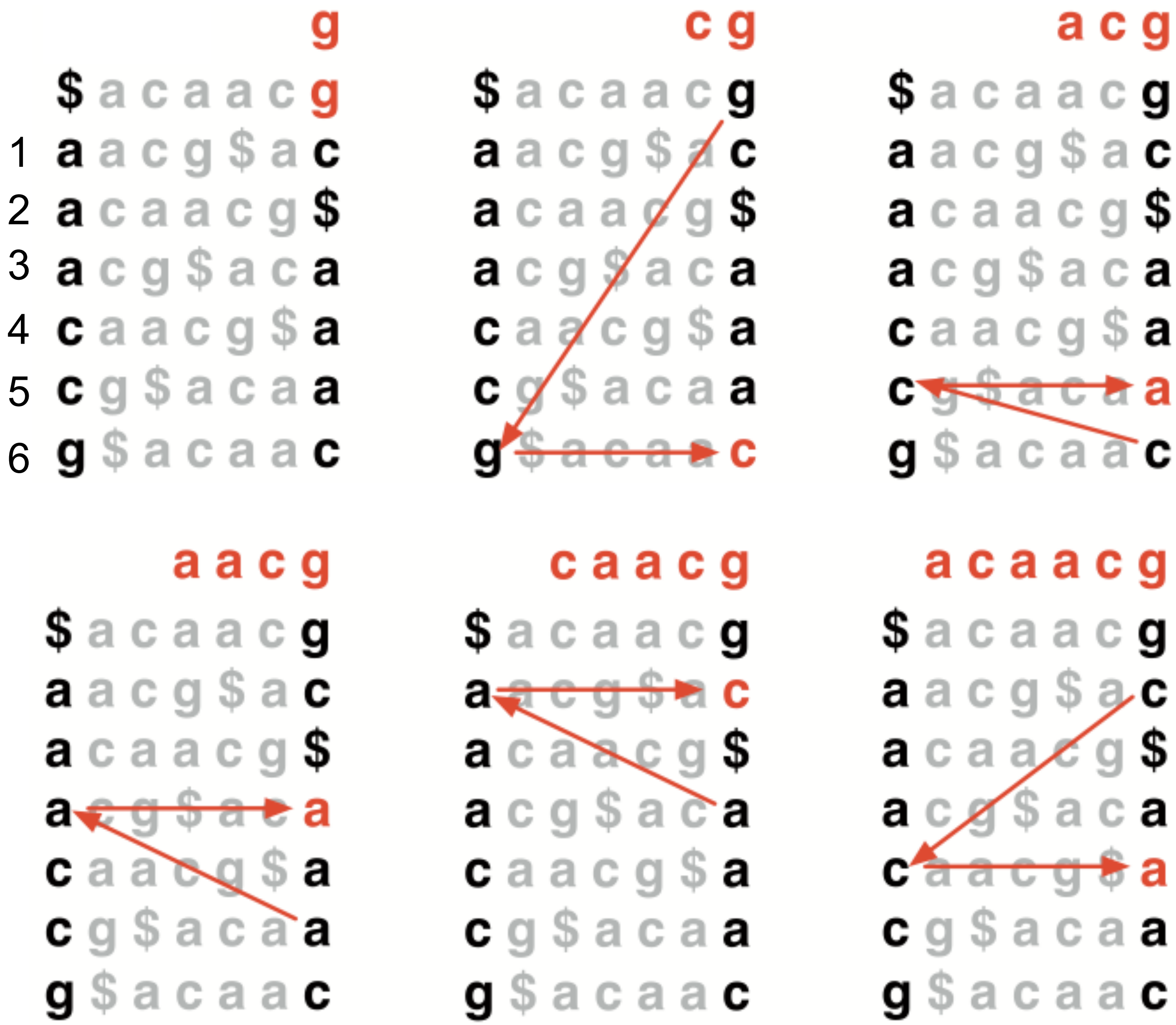
LF property implicitly encodes the Suffix Array

Last first (LF) mapping

We can repeatedly apply LF mapping to **reconstruct T from BWT(T)**

UNPERMUTE algorithm

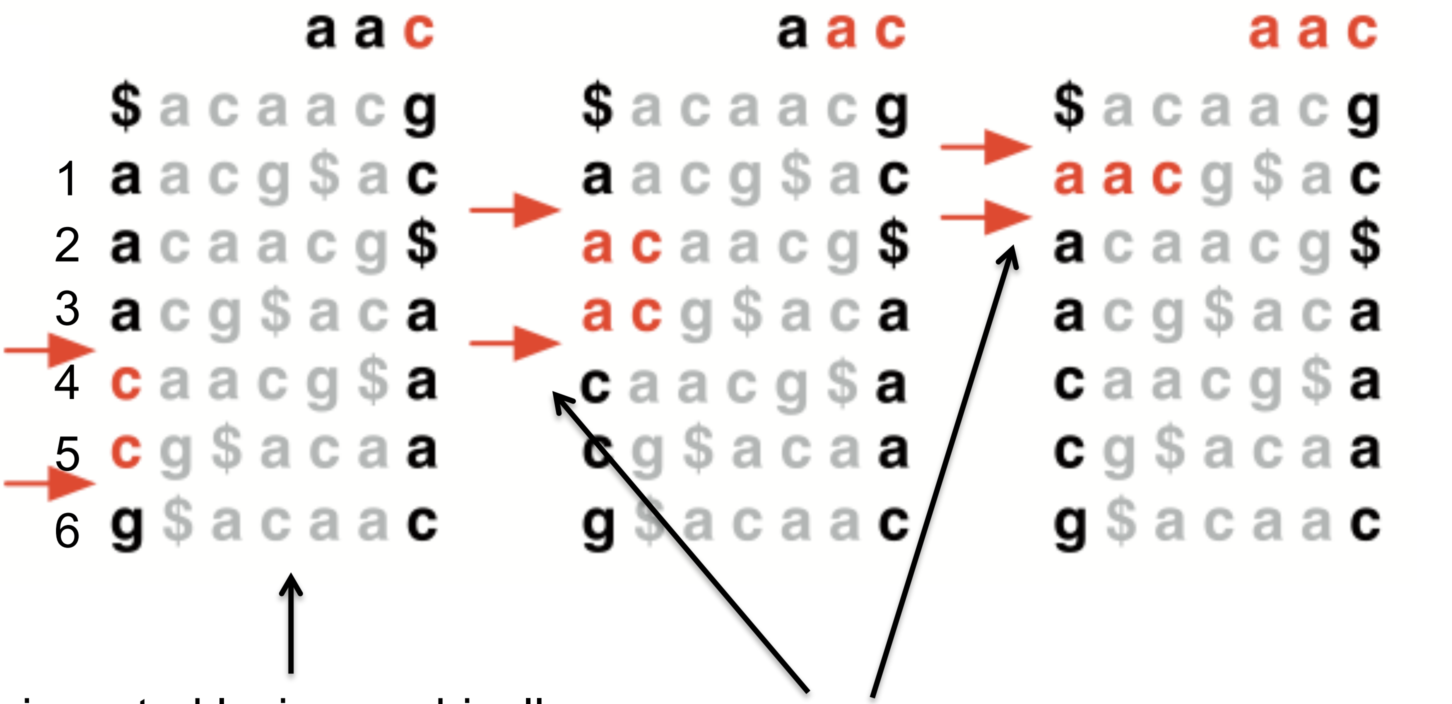
(Burrows and Wheeler, 1994)



LF mapping and exact matching

EXACTMATCH algorithm (Ferragina and Manzini, 2000) - calculates the range of matrix rows beginning with successively longer suffixes of the query

Reference: acaacg. Query: aac



aac aac aac

	$acaacg	$acaacg	$acaacg
1	aacg$ac	aacg$ac	aacg$ac
2	acaacg$	acaacg$	acaacg$
3	acg$aca	acg$aca	acg$aca
4	caacg$a	caacg$a	caacg$a
5	cg$acaa	cg$acaa	cg$acaa
6	g$acaac	g$acaac	g$acaac

the matrix is sorted lexicographically

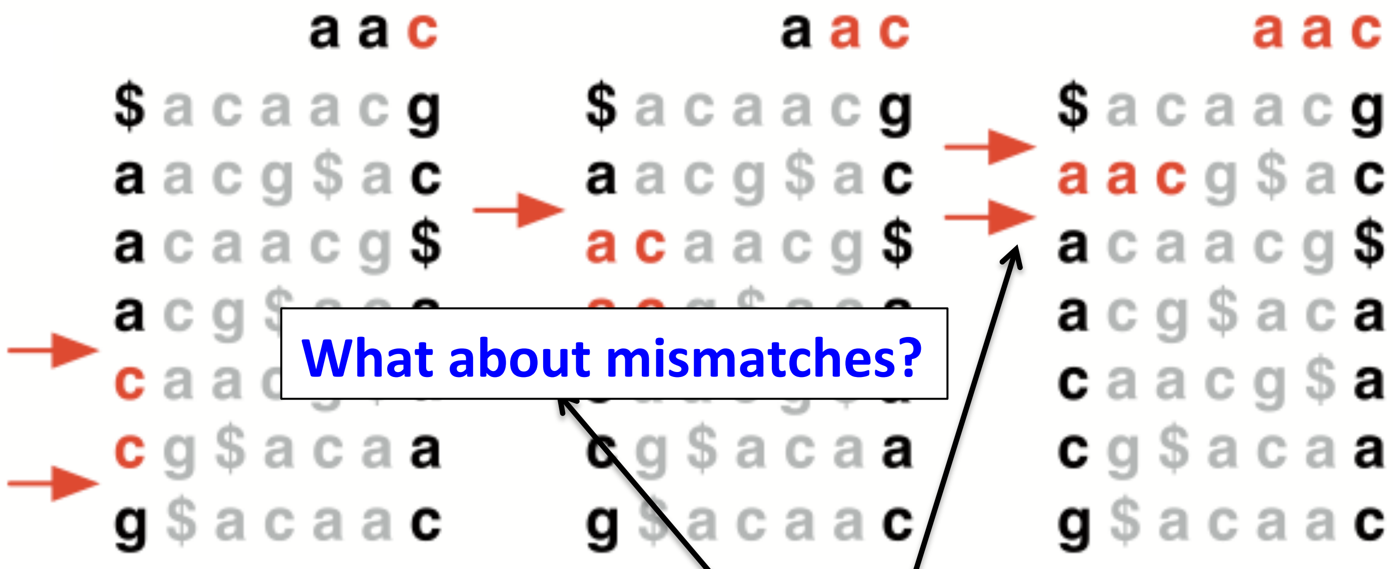
rows beginning with a given sequence appear consecutively

At each step, the size of the range either shrinks or remains the same

LF mapping and exact matching

EXACTMATCH algorithm (Ferragina and Manzini, 2000) - calculates the range of matrix rows beginning with successively longer suffixes of the query

Reference: acaacg. Query: aac



What about mismatches?

the matrix is sorted lexicographically

rows beginning with a given sequence appear consecutively

At each step, the size of the range either shrinks or remains the same

Mismatches?

- EXACTMATCH is insufficient for short read alignment because alignments may contain mismatches
- What are the main causes for mismatches?

  - sequencing errors

  - differences between reference and query organisms

Bowtie – mismatches and backtracking search

- EXACTMATCH is insufficient for short read alignment because alignments may contain mismatches
- Bowtie conducts a **backtracking search** to quickly find alignments that satisfy a specified alignment policy
- Each character in a read has a **numeric quality value**, with lower values indicating a higher likelihood of a sequencing error
- Example: Illumina uses Phred quality scoring

  Phred score of a base is: $Q_{phred} = -10*\log_{10}(e)$ where e is the estimated probability of a base being wrong
- Bowtie alignment policy allows a **limited number of mismatches** and prefers alignments where the **sum of the quality values at all mismatched positions is low**

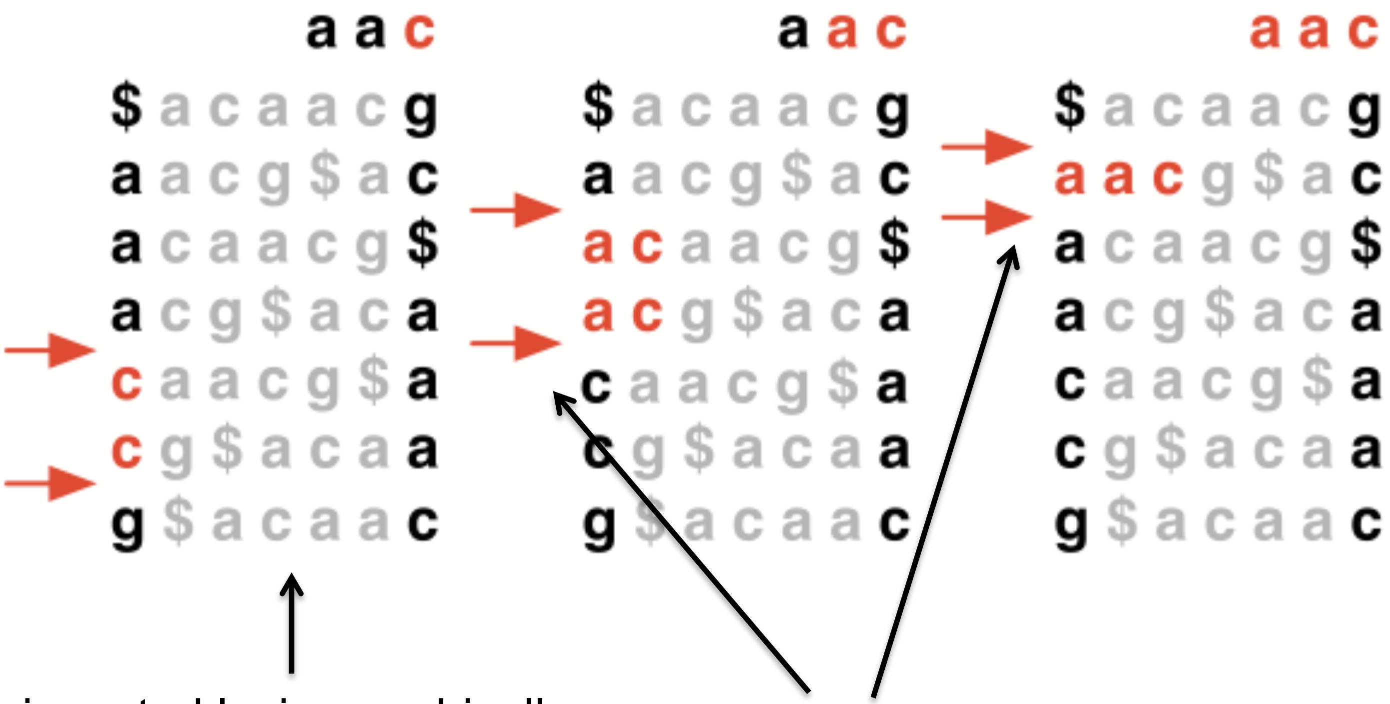
Bowtie - backtracking search

- The search is similar to EXACTMATCH
- It calculates matrix ranges for successively longer query suffixes

LF mapping and exact matching

EXACTMATCH algorithm (Ferragina and Manzini, 2000) - calculates the range of matrix rows beginning with successively longer suffixes of the query

Reference: acaacg. Query: aac



```
aac          aac          aac
$acaacg      $acaacg      $acaacg
aacg$ac      aacg$ac      aacg$ac
acaacg$      acaacg$      acaacg$
acg$aca      acg$aca      acg$aca
caacg$a      caacg$a      caacg$a
cg$acaa      cg$acaa      cg$acaa
g$acaac      g$acaac      g$acaac
```

the matrix is sorted lexicographically

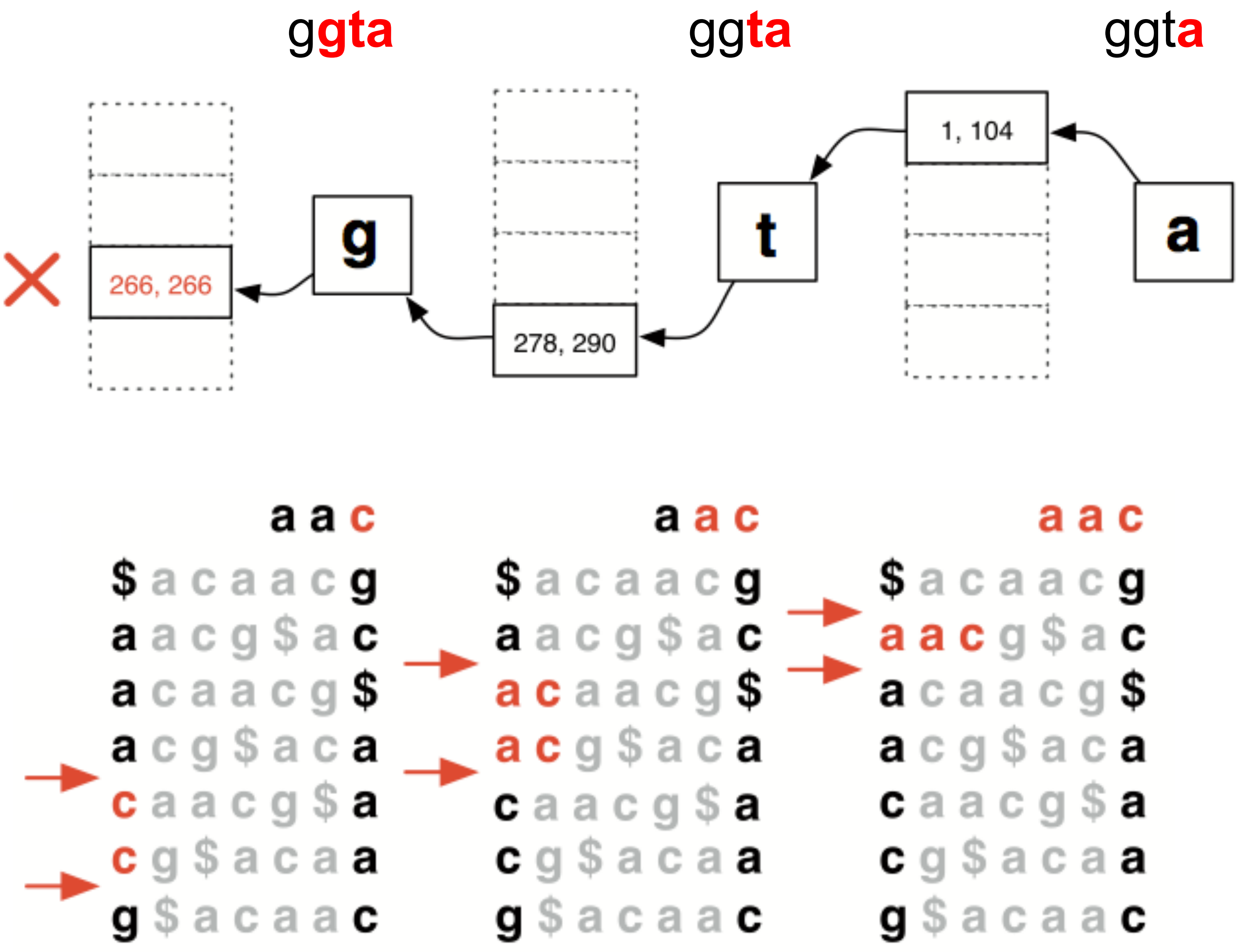
rows beginning with a given sequence appear consecutively

At each step, the size of the range either shrinks or remains the same

Bowtie - backtracking search

- The search is similar to EXACTMATCH
- It calculates matrix ranges for successively longer query suffixes
- **If the range becomes empty** (a suffix does not occur in the text), then the algorithm may select an already-matched query position and substitute a different base there, introducing a **mismatch** into the alignment
- The EXACTMATCH search resumes from just after the substituted position
- The algorithm selects only those substitutions that are **consistent with the alignment policy**

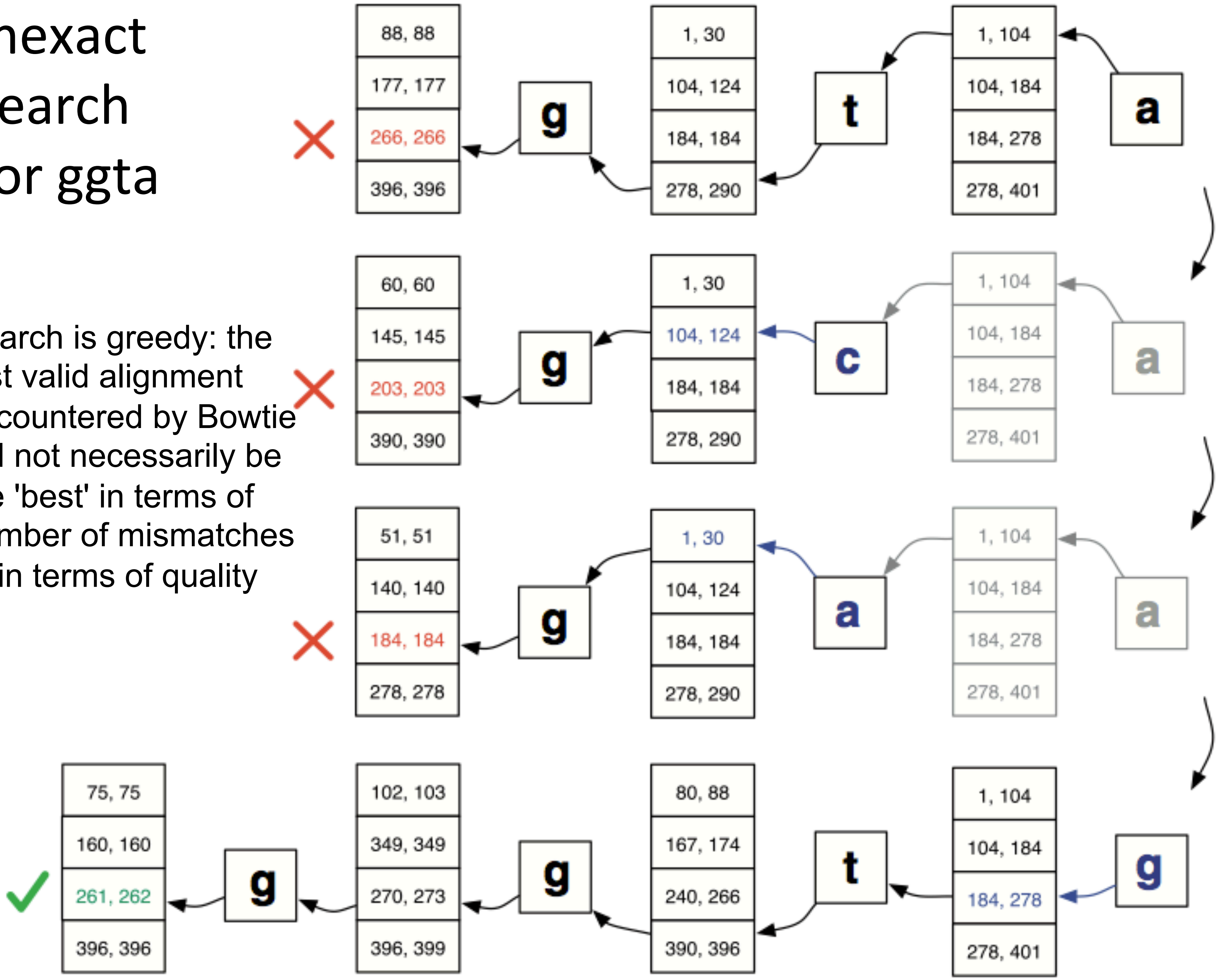
Exact search for ggta



```
      a a c             a a c             a a c
$ a c a a c g     $ a c a a c g     $ a c a a c g
a a c g $ a c     a a c g $ a c     a a c g $ a c
a c a a c g $     a c a a c g $     a c a a c g $
a c g $ a c a     a c g $ a c a     a c g $ a c a
c a a c g $ a     c a a c g $ a     c a a c g $ a
c g $ a c a a     c g $ a c a a     c g $ a c a a
g $ a c a a c     g $ a c a a c     g $ a c a a c
```

Inexact search for ggta

Search is greedy: the first valid alignment encountered by Bowtie will not necessarily be the 'best' in terms of number of mismatches or in terms of quality



Bowtie - backtracking search

- This standard aligner can, in some cases, encounter sequences that cause excessive backtracking
- Bowtie mitigates excessive backtracking with the novel technique of **double indexing**
  - Idea: create 2 indices of the genome: one containing the BWT of the genome, called the **forward index**, and a second containing the BWT of the genome with its sequence reversed (not reverse complemented) called the **mirror index**.
- Let's consider a matching policy that allows one mismatch in the alignment (either in the first half or in the second half)
- Bowtie proceeds in two phases:

  1. load the **forward index** into memory and invoke the aligner with the constraint that it may *not* substitute at positions in the query's **right half**

  2. load the **mirror index** into memory and invoke the aligner on the *reversed query*, with the constraint that the aligner may *not* substitute at positions in the reversed query's right half (the original query's **left half**).
- The constraints on backtracking into the right half prevent excessive backtracking, whereas the use of two phases and two indices maintains full sensitivity

Bowtie - backtracking search

- Base quality varies across the read
- Bowtie allows the user to select

  - the number of mismatches permitted in the high-quality end of a read (default: 2 mismatches in the first 28 bases)

  - maximum acceptable quality of mismatched positions over the alignment (default: 70 PHRED score)
- The first 28 bases on the high-quality end of the read are termed the **seed**
- The seed consists of two halves:

  - the 14 bp on the high-quality end (usually the 5' end) = the **hi-half**

  - the 14 bp on the low-quality end = **lo-half**
- Assuming 2 mismatches permitted in the seed, a reportable alignment will fall into one of four cases:

  1. no mismatches in seed;

  2. no mismatches in hi-half, one or two mismatches in lo-half

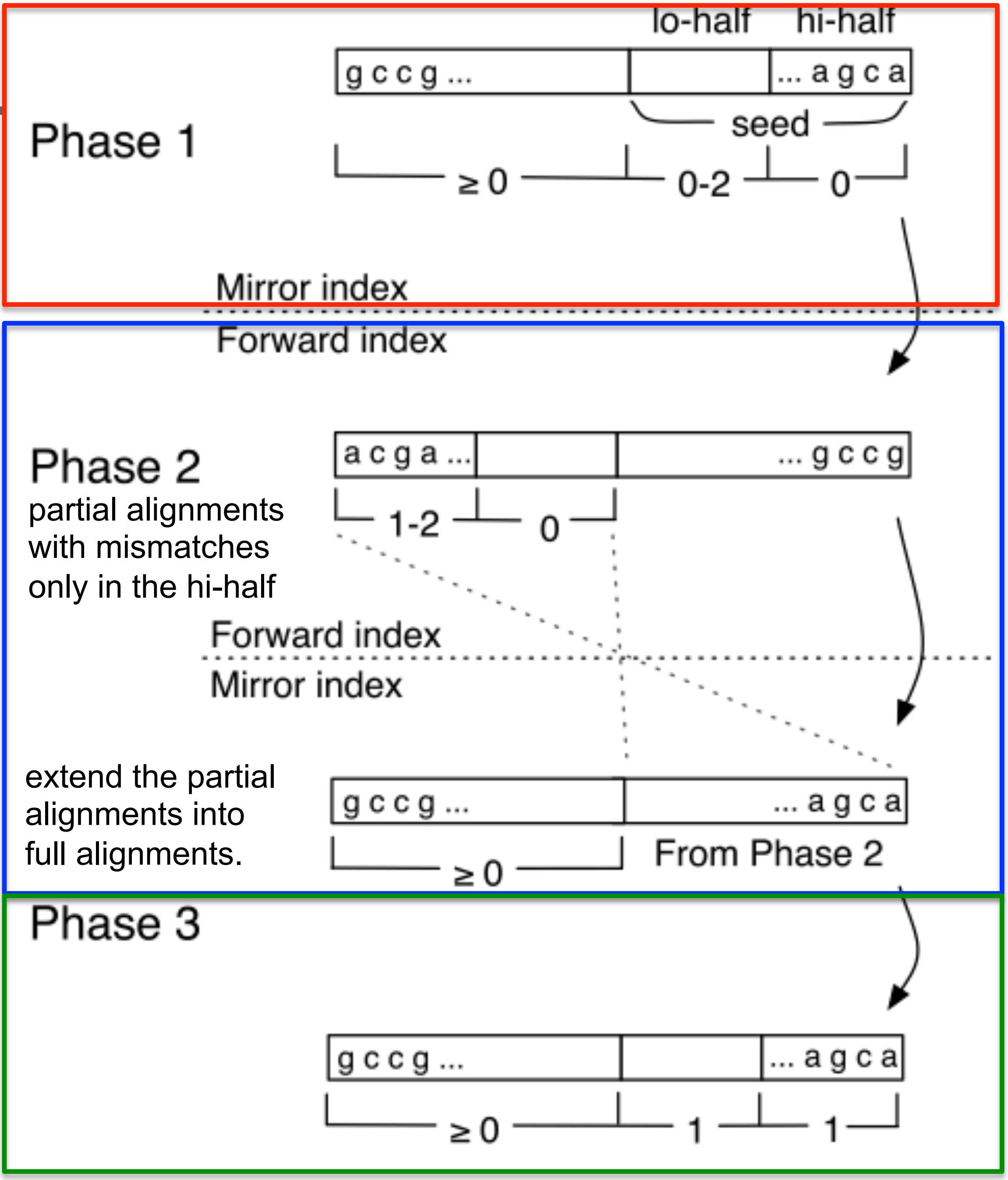
  3. no mismatches in lo-half, one or two mismatches in hi-half

  4. one mismatch in hi-half, one mismatch in lo- half

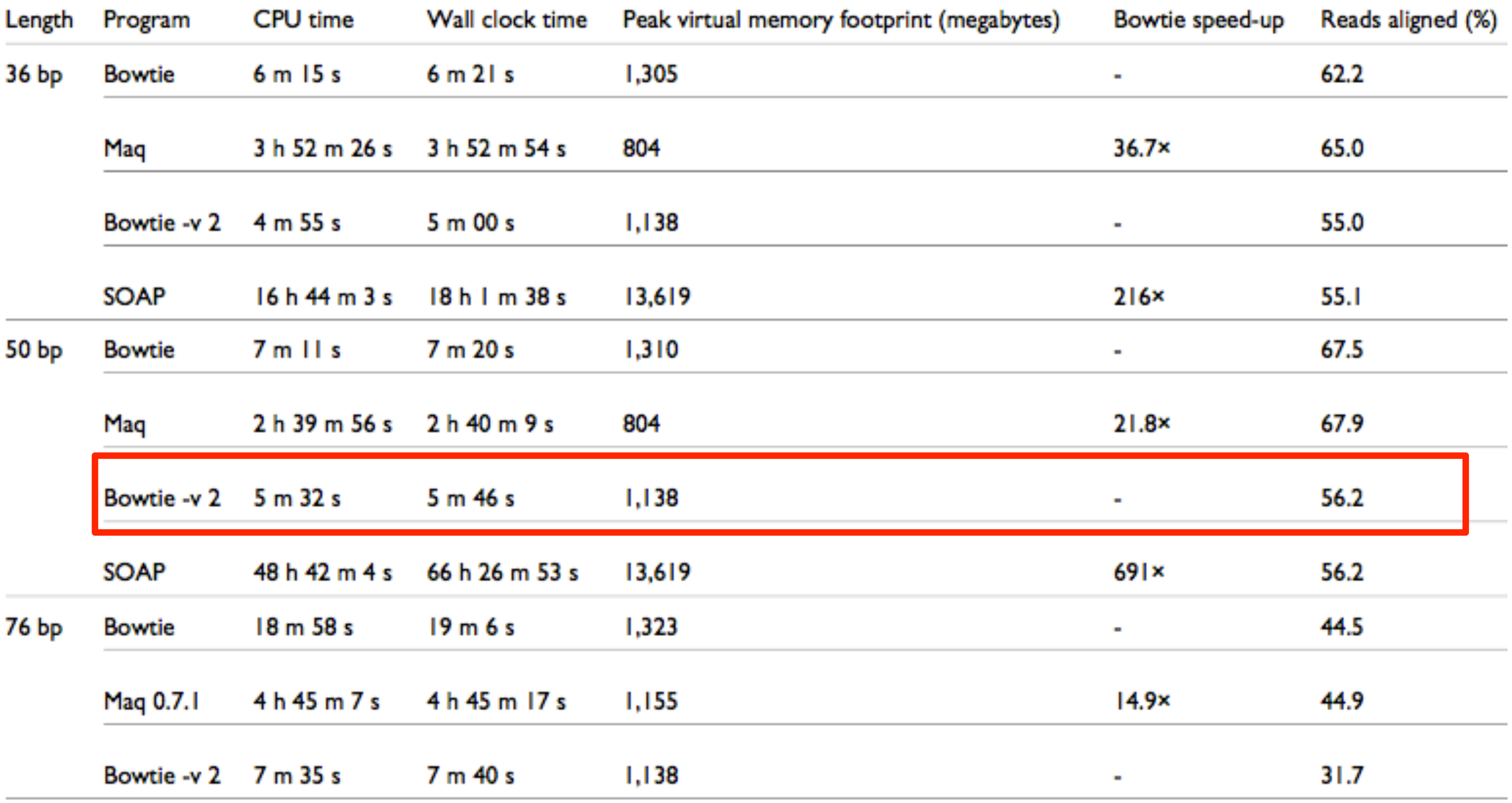
Bowtie

- The Bowtie algorithm consists of three phases that alternate between using the forward and mirror indices

1. no mismatches in seed
2. no mismatches in hi-half, one or two mismatches in lo-half
3. no mismatches in lo-half, one or two mismatches in hi-half
4. one mismatch in hi-half, one mismatch in lo-half



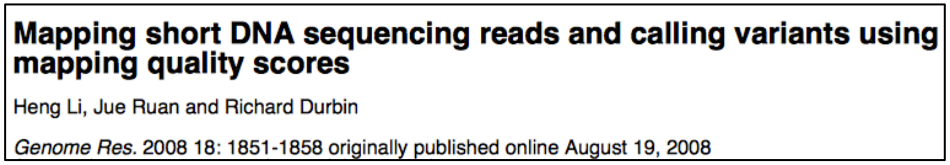
Aligning 2 million reads to the human genome



Length	Program	CPU time	Wall clock time	Peak virtual memory footprint (megabytes)	Bowtie speed-up	Reads aligned (%)
36 bp	Bowtie	6 m 15 s	6 m 21 s	1,305	-	62.2
	Maq	3 h 52 m 26 s	3 h 52 m 54 s	804	36.7×	65.0
	Bowtie -v 2	4 m 55 s	5 m 00 s	1,138	-	55.0
	SOAP	16 h 44 m 3 s	18 h 1 m 38 s	13,619	216×	55.1
50 bp	Bowtie	7 m 11 s	7 m 20 s	1,310	-	67.5
	Maq	2 h 39 m 56 s	2 h 40 m 9 s	804	21.8×	67.9
	Bowtie -v 2	5 m 32 s	5 m 46 s	1,138	-	56.2
	SOAP	48 h 42 m 4 s	66 h 26 m 53 s	13,619	691×	56.2
76 bp	Bowtie	18 m 58 s	19 m 6 s	1,323	-	44.5
	Maq 0.7.1	4 h 45 m 7 s	4 h 45 m 17 s	1,155	14.9×	44.9
	Bowtie -v 2	7 m 35 s	7 m 40 s	1,138	-	31.7

Maq: Mapping and Assembly with Qualities

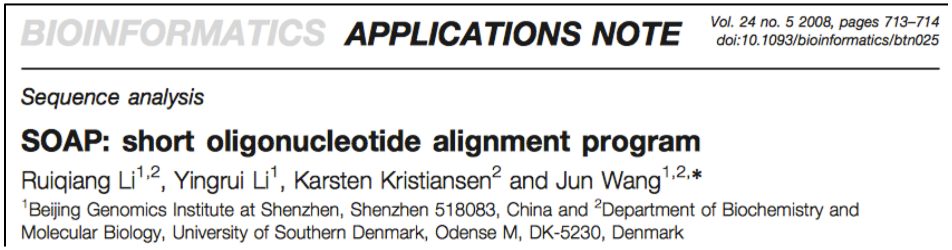
SOAP = Short Oligonucleotide Analysis Package



Mapping short DNA sequencing reads and calling variants using mapping quality scores

Heng Li, Jue Ruan and Richard Durbin

Genome Res. 2008 18: 1851-1858 originally published online August 19, 2008



BIOINFORMATICS ***APPLICATIONS NOTE***

Vol. 24 no. 5 2008, pages 713–714
doi:10.1093/bioinformatics/btn025

Sequence analysis

SOAP: short oligonucleotide alignment program

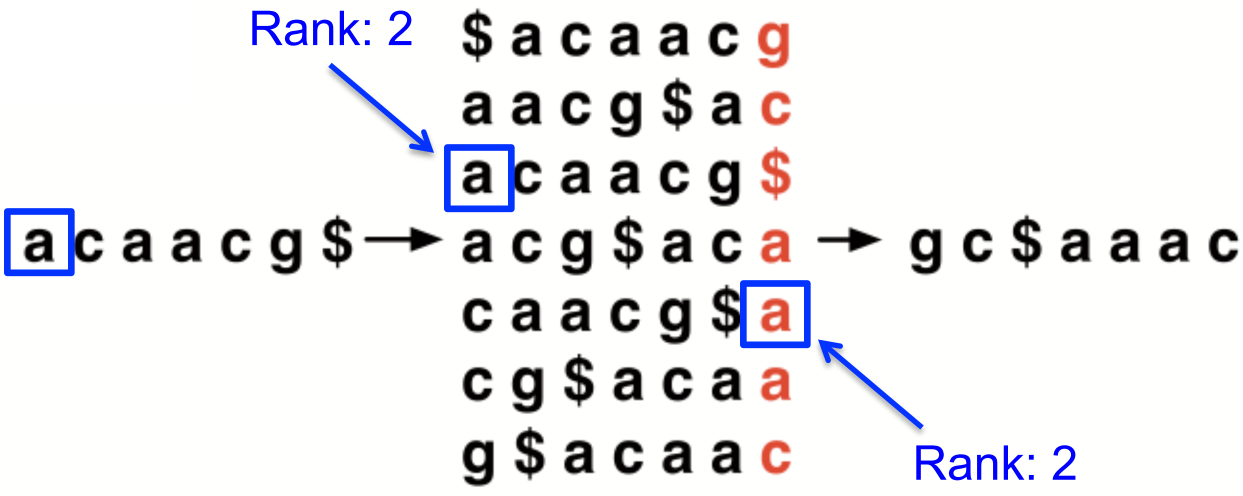
Ruiqiang Li[1,2], Yingrui Li[1], Karsten Kristiansen[2] and Jun Wang[1,2,*]

[1]Beijing Genomics Institute at Shenzhen, Shenzhen 518083, China and [2]Department of Biochemistry and Molecular Biology, University of Southern Denmark, Odense M, DK-5230, Denmark

Last first (LF) mapping

- The BW matrix has a property called **last first (LF) mapping**:

  The ith occurrence of character X in the last column corresponds to the same text character as the ith occurrence of X in the first column
- This property is at the core of algorithms that use the BWT index to search the text



Rank: 2

```
          $ a c a a c g
          a a c g $ a c
          a c a a c g $
a c a a c g $ →  a c g $ a c a  →  g c $ a a a c
          c a a c g $ a
          c g $ a c a a
          g $ a c a a c
```

Rank: 2

$$BWT[i] = \begin{cases} T[SA[i]-1] & SA[i] \neq 0 \\ \$ & SA[i] = 0 \end{cases}$$

LF property implicitly encodes the Suffix Array

Constructing the index

- How do we construct a BWT index?
- Calculating the BWT is closely related to building a suffix array
- Each element of the **BWT** can be derived from the corresponding element of the **suffix array**:

$$BWT[i] = \begin{cases} T[SA[i]-1] & SA[i] \neq 0 \\ \$ & SA[i] = 0 \end{cases}$$

- One could generate all suffixes, sort then to obtain the SA, then calculate the BWT in a single pass over the suffix array
- However, constructing the entire suffix array in memory requires at least **~12 gigabytes** for the human genome
- Instead, Bowtie uses a **block-wise strategy**: builds the suffix array and the BWT block-by-block, discarding suffix array blocks once the corresponding BWT block has been built
- Bowtie can build the full index for the human genome in about **24 hours** in less than **1.5 gigabytes of RAM**
- If **16 gigabytes of RAM** or more is available, Bowtie can exploit the additional RAM to produce the same index in about **4.5 hours**

Constructing the index

- How do we construct a BWT index?
- Calculating the BWT is closely related to building a suffix array
- Each element of the **BWT** can be derived from the corresponding element of the **suffix array**:

$$BWT[i] = \begin{cases} T[SA[i]-1] & SA[i] \neq 0 \\ \$ & SA[i] = 0 \end{cases}$$

- One could generate all suffixes, sort then to obtain the SA, then calculate the BWT in a single pass over the suffix array
- However, constructing the entire suffix array in memory requires at least **~12 gigabytes** for the human genome
- Instead, Bowtie uses a **block-wise strategy**: builds the suffix array and the BWT block-by-block, discarding suffix array blocks once the corresponding BWT block has been built
- Bowtie can build the full index for the human genome in about **24 hours** in less than **1.5 gigabytes of RAM**
- If **16 gigabytes of RAM** or more is available, Bowtie can exploit the additional RAM to produce the same index in about **4.5 hours**

Storing the index

- The largest single component of the Bowtie index is the BWT sequence. Bowtie stores the BWT in a **2-bit-per-base** format
- A Bowtie index for the assembled human genome sequence is about **1.3 gigabytes**
- A full Bowtie index actually consists of pair of equal-size indexes, the **forward** and **mirror** indexes, for any given genome, but it can be run such that only one of the two indexes is ever resident in memory at once (using the –z option)

- What about gaps?

Bowtie 2

- Bowtie: very efficient **ungapped** alignment of short reads based on BWT index
- Index-based alignment algorithms can be quite inefficient when gaps are allowed
- Gaps can results from
  - sequencing errors
  - true insertions and deletions
- Bowtie 2 extends the index-based approach of Bowtie to permit gapped alignment
- It divides the algorithm into two stages

  1. an initial, ungapped **seed-finding stage** that benefits from the speed and memory efficiency of the full-text index

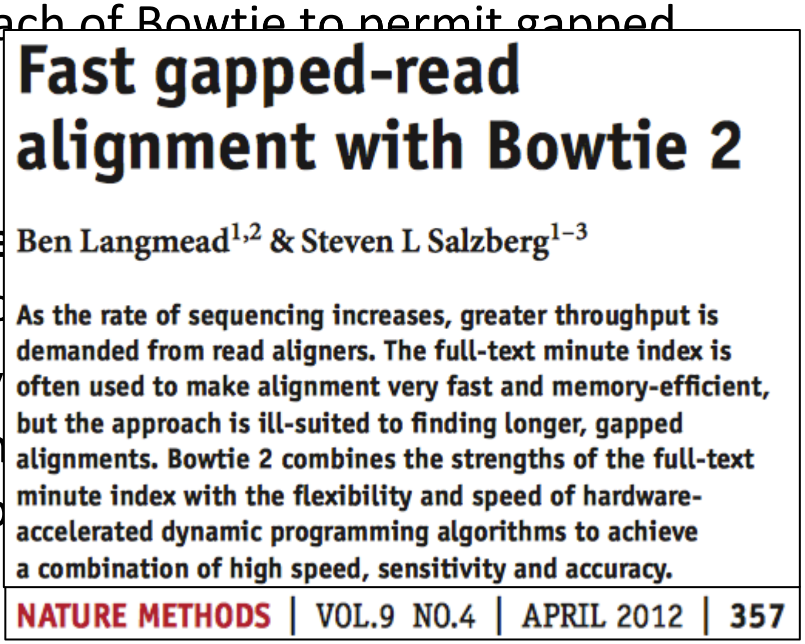
  2. a **gapped extension stage** that uses dynamic programming and benefits from the efficiency of single-instruction multiple-data (SIMD) parallel processing available on modern processors

Bowtie 2

- Bowtie: very efficient **ungapped** alignment of short reads based on BWT index
- Index-based alignment algorithms can be quite inefficient when gaps are allowed
- Gaps can results from
  - sequencing errors
  - true insertions and deletions
- Bowtie 2 extends the index-based approach of Bowtie to permit gapped alignment
- It divides the algorithm into two stages

  1. an initial, ungapped **seed-finding stage** and memory efficiency of the full-text inc

  2. a **gapped extension stage** that uses dy from the efficiency of single-instruction n processing available on modern processo

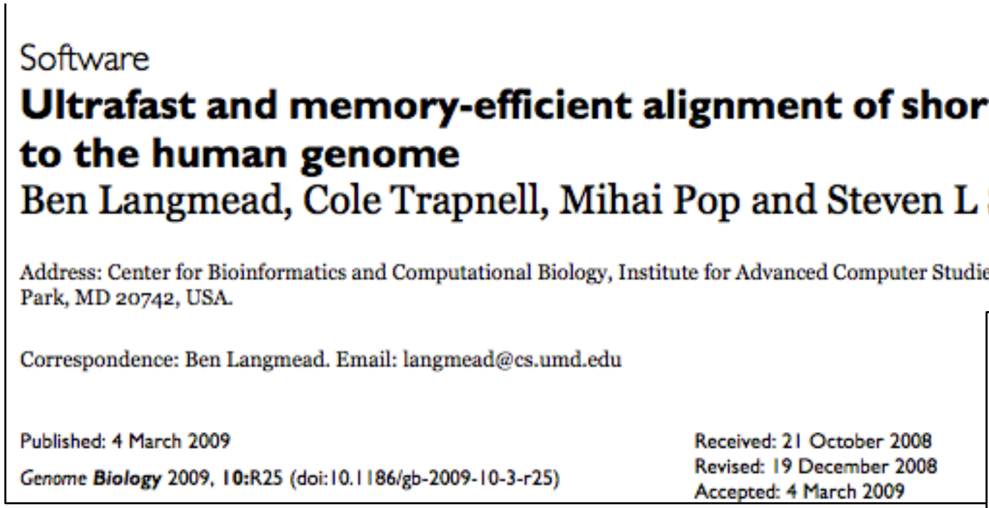


Fast gapped-read alignment with Bowtie 2

Ben Langmead[1,2] & Steven L Salzberg[1–3]

As the rate of sequencing increases, greater throughput is demanded from read aligners. The full-text minute index is often used to make alignment very fast and memory-efficient, but the approach is ill-suited to finding longer, gapped alignments. Bowtie 2 combines the strengths of the full-text minute index with the flexibility and speed of hardware-accelerated dynamic programming algorithms to achieve a combination of high speed, sensitivity and accuracy.

NATURE METHODS | VOL.9 NO.4 | APRIL 2012 | 357



Software

Open Access

Ultrafast and memory-efficient alignment of short DNA sequences to the human genome

Ben Langmead, Cole Trapnell, Mihai Pop and Steven L Salzberg

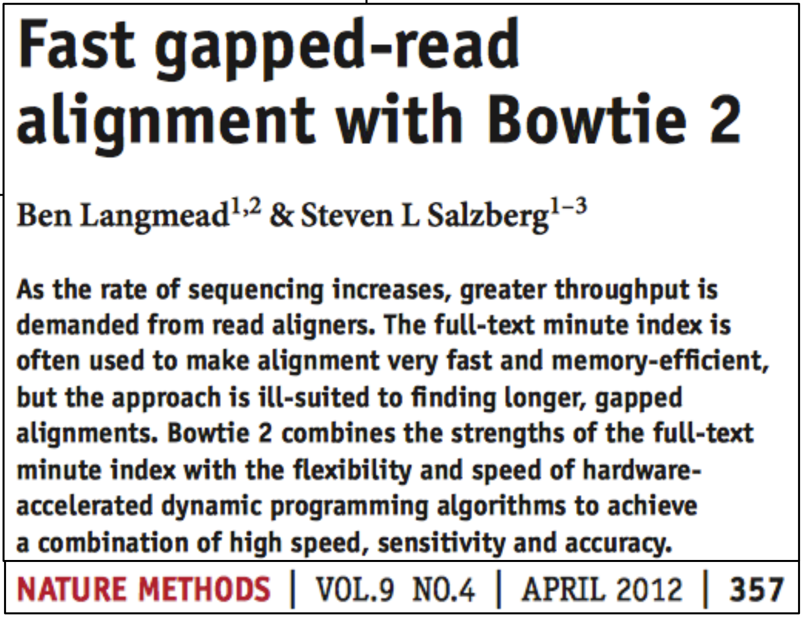
Address: Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA.

Correspondence: Ben Langmead. Email: langmead@cs.umd.edu

Published: 4 March 2009

Genome ***Biology*** 2009, **10:**R25 (doi:10.1186/gb-2009-10-3-r25)

Received: 21 October 2008
Revised: 19 December 2008
Accepted: 4 March 2009

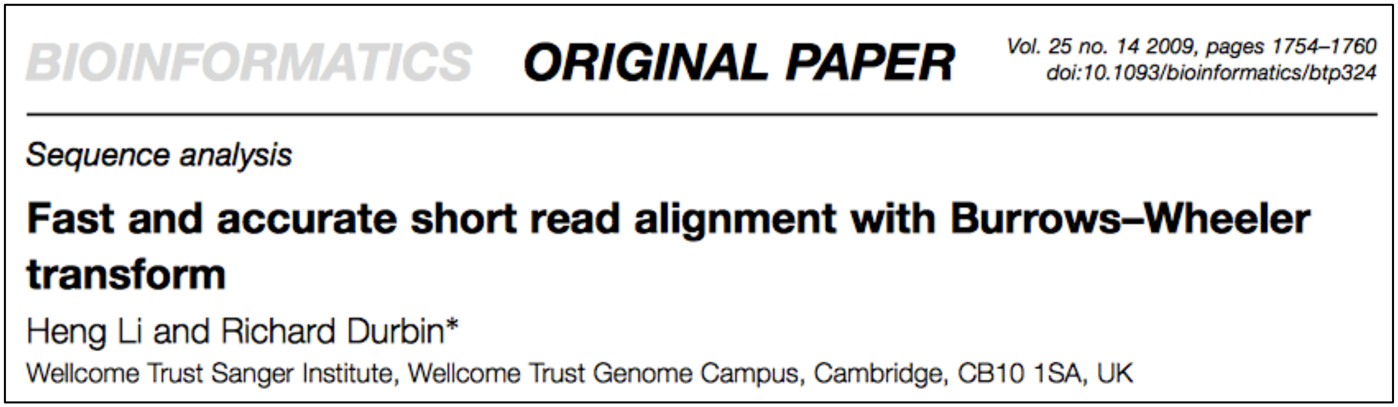


Fast gapped-read alignment with Bowtie 2

Ben Langmead[1,2] & Steven L Salzberg[1–3]

As the rate of sequencing increases, greater throughput is demanded from read aligners. The full-text minute index is often used to make alignment very fast and memory-efficient, but the approach is ill-suited to finding longer, gapped alignments. Bowtie 2 combines the strengths of the full-text minute index with the flexibility and speed of hardware-accelerated dynamic programming algorithms to achieve a combination of high speed, sensitivity and accuracy.

NATURE METHODS | VOL.9 NO.4 | APRIL 2012 | 357



BIOINFORMATICS ***ORIGINAL PAPER***

Vol. 25 no. 14 2009, pages 1754–1760
doi:10.1093/bioinformatics/btp324

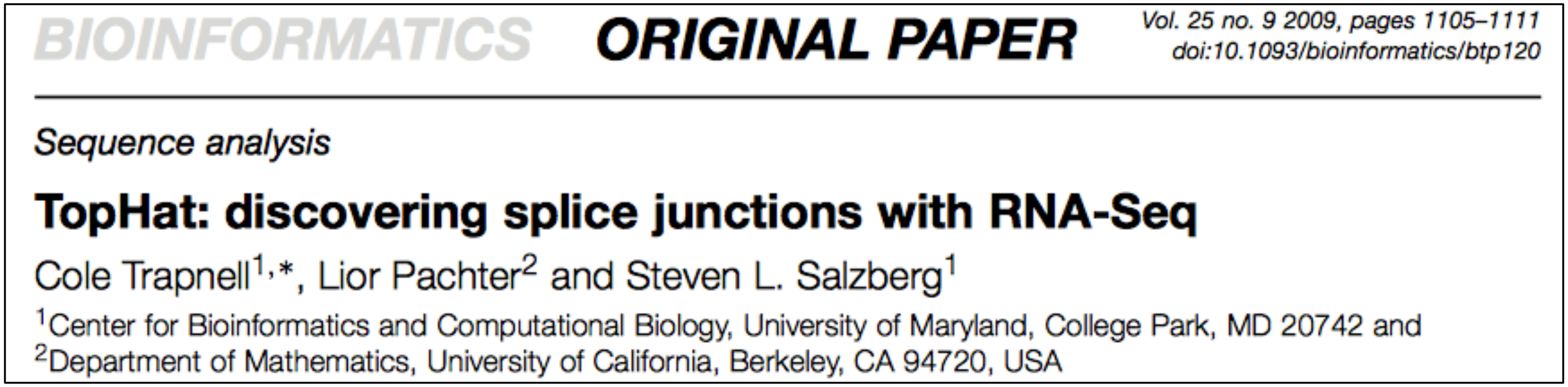
Sequence analysis

Fast and accurate short read alignment with Burrows–Wheeler transform

Heng Li and Richard Durbin*

Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Cambridge, CB10 1SA, UK

BWA



BIOINFORMATICS ORIGINAL PAPER

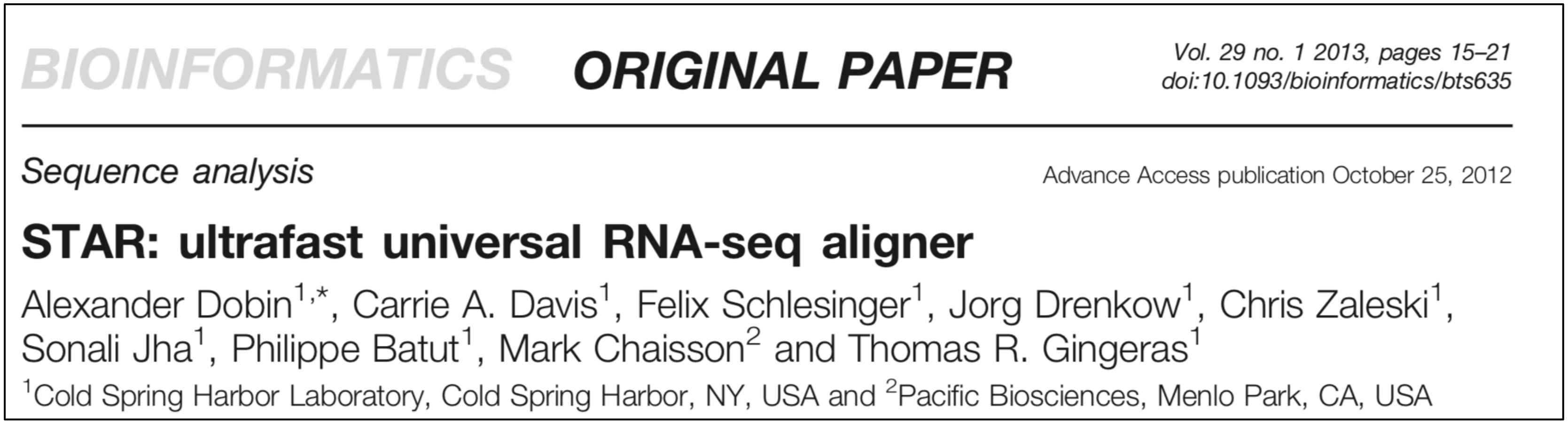
Vol. 25 no. 9 2009, pages 1105–1111
doi:10.1093/bioinformatics/btp120

Sequence analysis

TopHat: discovering splice junctions with RNA-Seq

Cole Trapnell[1,*], Lior Pachter[2] and Steven L. Salzberg[1]

[1]Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD 20742 and [2]Department of Mathematics, University of California, Berkeley, CA 94720, USA



BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 1 2013, pages 15–21
doi:10.1093/bioinformatics/bts635

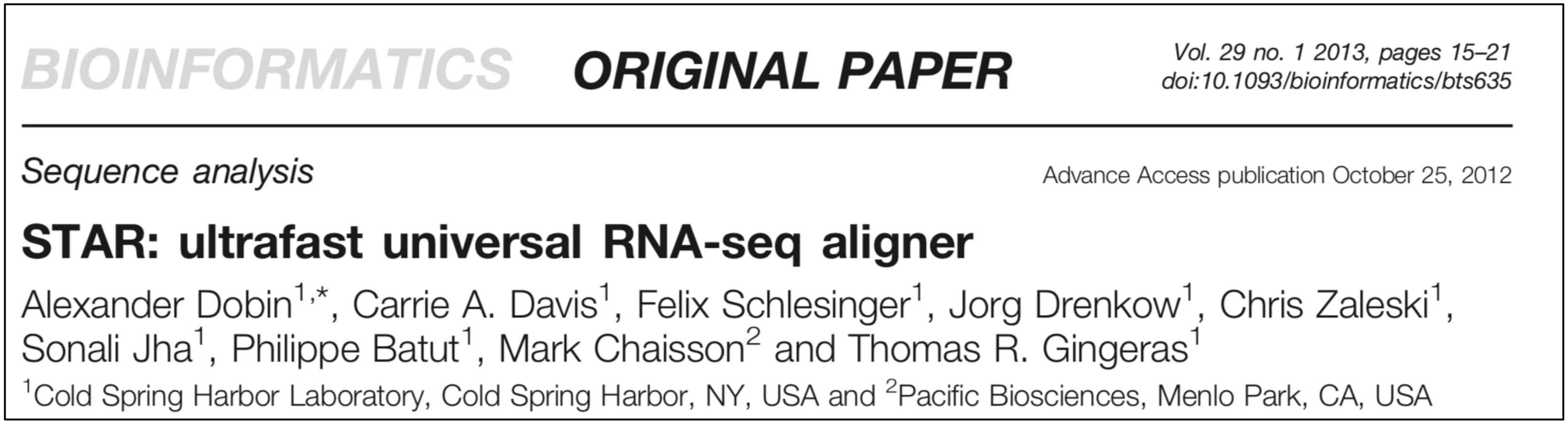
Sequence analysis

Advance Access publication October 25, 2012

STAR: ultrafast universal RNA-seq aligner

Alexander Dobin[1,*], Carrie A. Davis[1], Felix Schlesinger[1], Jorg Drenkow[1], Chris Zaleski[1], Sonali Jha[1], Philippe Batut[1], Mark Chaisson[2] and Thomas R. Gingeras[1]

[1]Cold Spring Harbor Laboratory, Cold Spring Harbor, NY, USA and [2]Pacific Biosciences, Menlo Park, CA, USA



BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 1 2013, pages 15–21
doi:10.1093/bioinformatics/bts635

Sequence analysis

Advance Access publication October 25, 2012

STAR: ultrafast universal RNA-seq aligner

Alexander Dobin[1,*], Carrie A. Davis[1], Felix Schlesinger[1], Jorg Drenkow[1], Chris Zaleski[1], Sonali Jha[1], Philippe Batut[1], Mark Chaisson[2] and Thomas R. Gingeras[1]

[1]Cold Spring Harbor Laboratory, Cold Spring Harbor, NY, USA and [2]Pacific Biosciences, Menlo Park, CA, USA

“Accurate alignment of high-throughput RNA-seq data is a challenging and yet unsolved problem because of the

- non-contiguous transcript structure,
- relatively short read lengths and
- constantly increasing throughput of the sequencing technologies.”

“Currently available RNA-seq aligners suffer from

- high mapping error rates,
- low mapping speed,
- read length limitation and
- mapping biases.”

Solution:

- sequential maximum mappable seed search in uncompressed suffix arrays
- followed by seed clustering and stitching procedure.

Suffix arrays - search for GATTACA

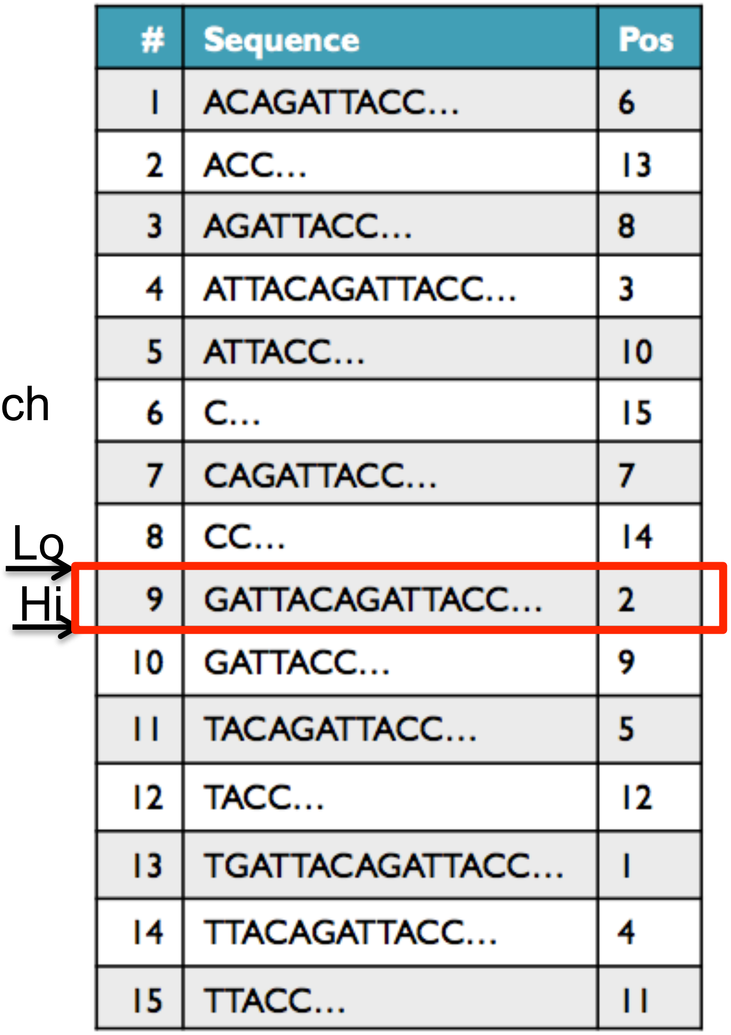
Lo = 9; Hi = 9

Mid = (9+9)/2 = 9

Middle = Suffix[9] = GATTACAG…

Compare GATTACA to GATTACAG… => Match

Return: match at position 2



#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Suffix arrays - search for GATTACA

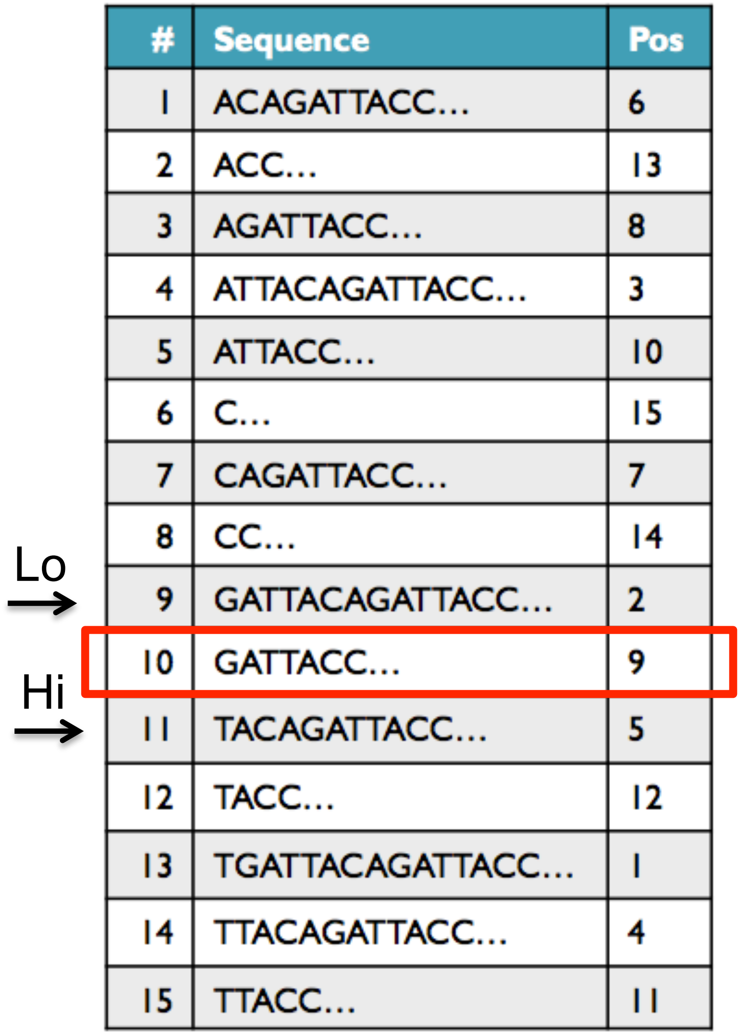
Lo = 9; Hi = 11

Mid = (9+11)/2 = 10

Middle = Suffix[10] = GATTACC

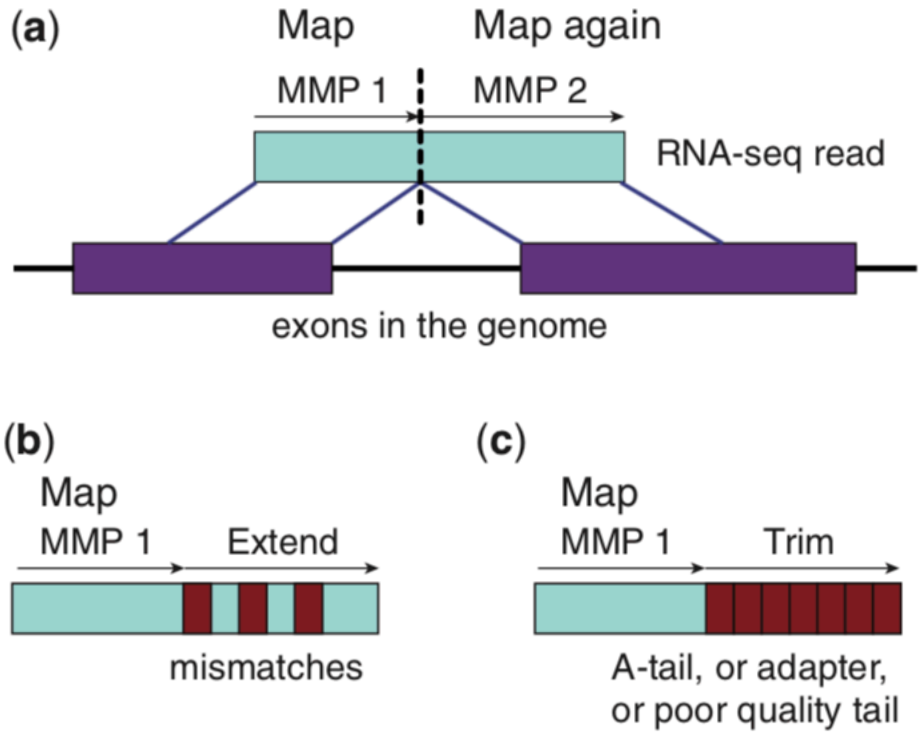
Compare GATTACA to GATTACC => Lower

Hi = Mid - 1



#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11

Maximum Mappable Prefix (MMP) search



- Using uncompressed suffix arrays leads to increased speed (compared to BWT)
- This speed advantage is traded off against the increased memory usage

Maximum Mappable Prefix (MMP) search

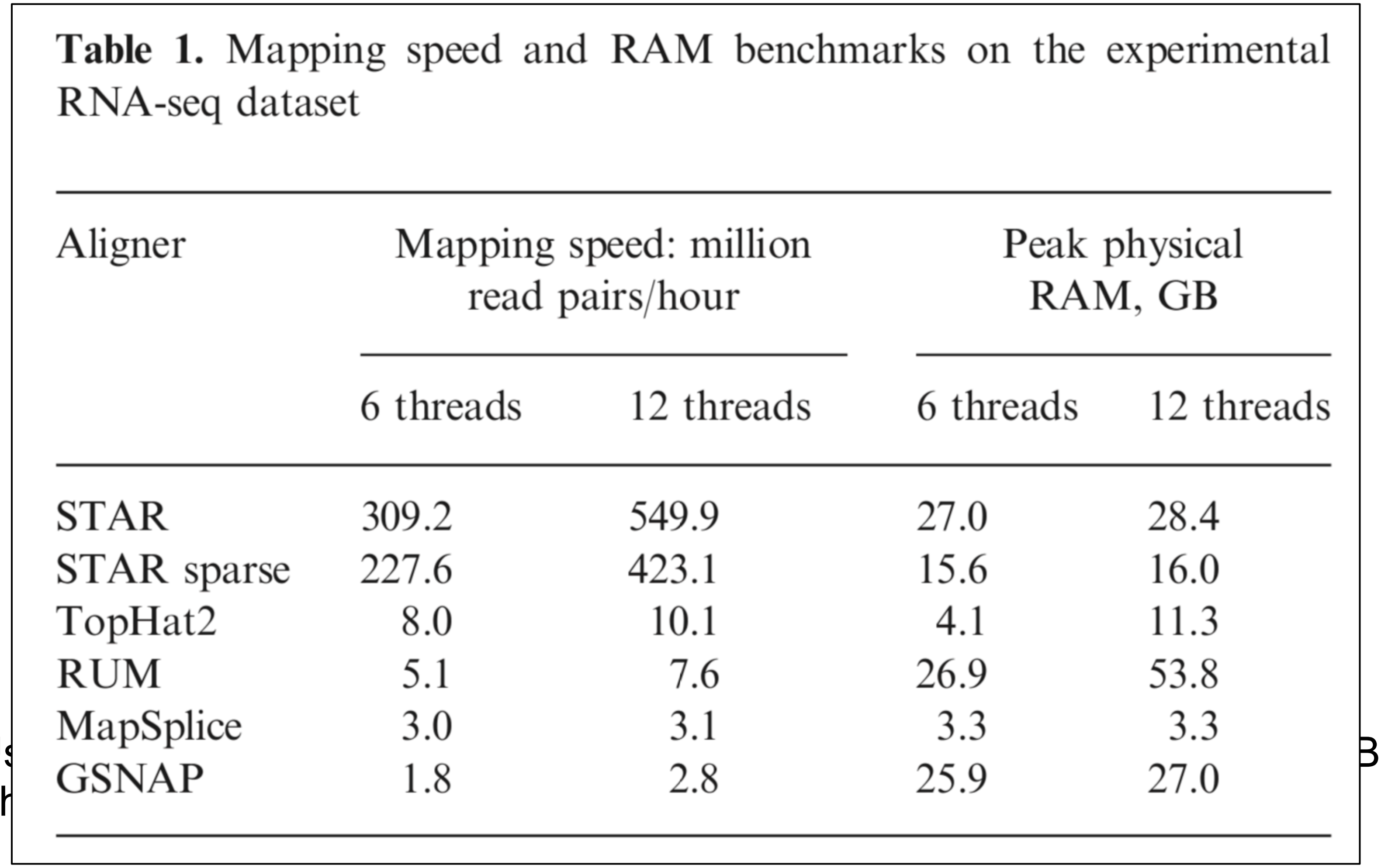


Table 1. Mapping speed and RAM benchmarks on the experimental RNA-seq dataset

Aligner	Mapping speed: million read pairs/hour		Peak physical RAM, GB	
	6 threads	12 threads	6 threads	12 threads
STAR	309.2	549.9	27.0	28.4
STAR sparse	227.6	423.1	15.6	16.0
TopHat2	8.0	10.1	4.1	11.3
RUM	5.1	7.6	26.9	53.8
MapSplice	3.0	3.1	3.3	3.3
GSNAP	1.8	2.8	25.9	27.0

- Us BWT)
- Th