Section 2:High-Throughput Sequencing Data

Maoying Wu

BI390 2019 Fall

Outline

- Sequencing technologies
  - From Sanger to 3rd generation sequencing
- Sequence representation & quality assessment
  - Fastq file
  - FASTQC: quality assessment
- Short-read apping algorithms
  - Spaced seed
  - Borrows Wheeler transformation & LF mapping
  - Suffix Tree and Suffix Array
- Sequence mapping representation
  - SAM / BAM formats
  - BED / BigBED formats
  - VCF / BCF format

Outline

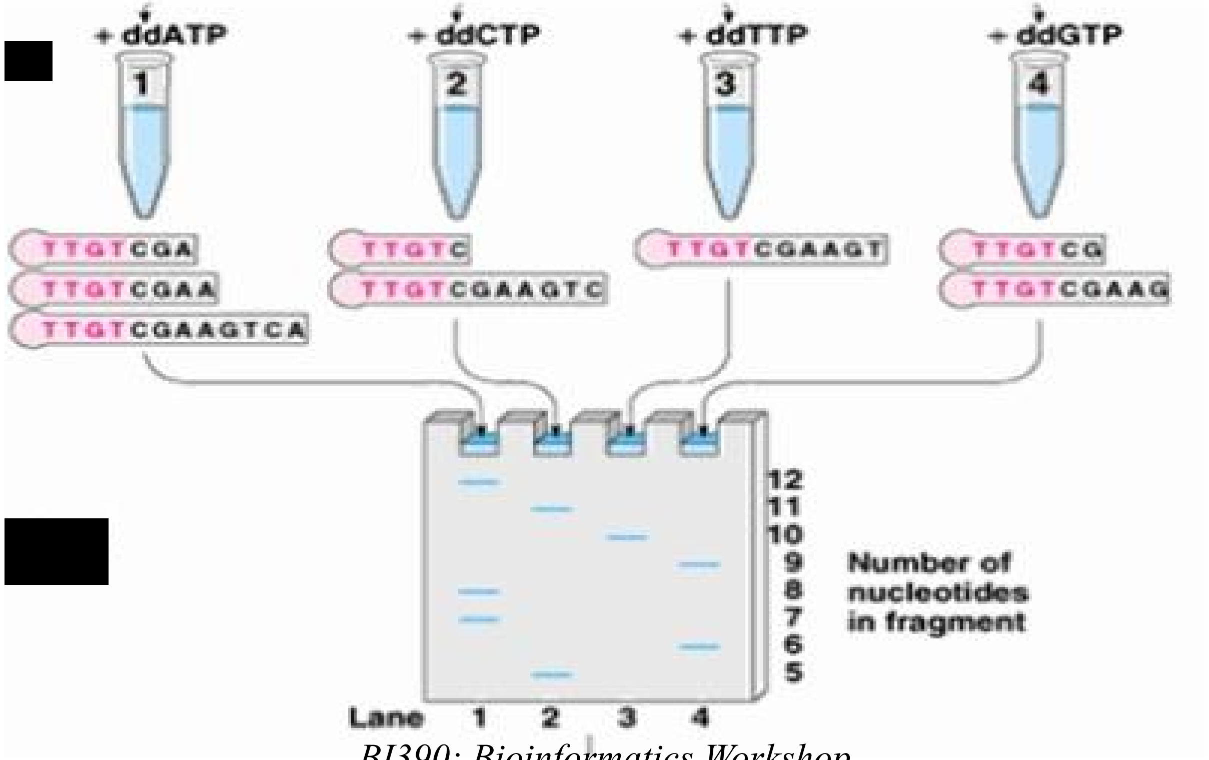
- Sequencing technologies.
- Fastq and FASTQC.
- Sequence mapping algorithms:
  - Spaced seed.
  - Borrows-Wheeler transformation & LF mapping.
  - Suffix Tree.
- Alignment output: SAM and BED.

Sanger sequencing – Step 1

- Add one-stranded DNA sequence to four test tubes.
- Each tube contain all d**N**TPs + one dd**N**TP.

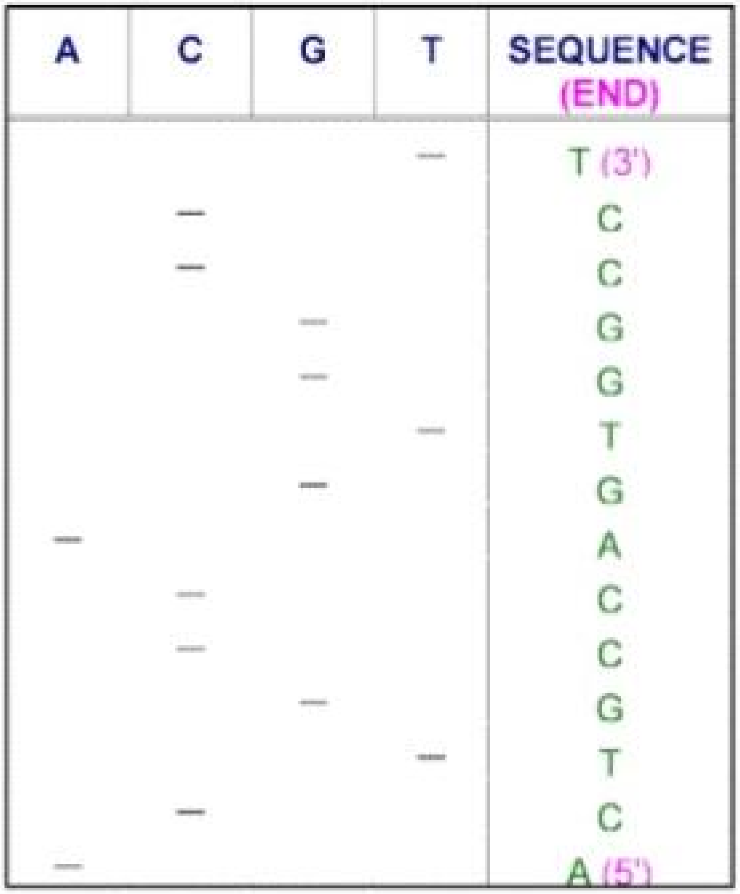
[redacted]

[redacted]

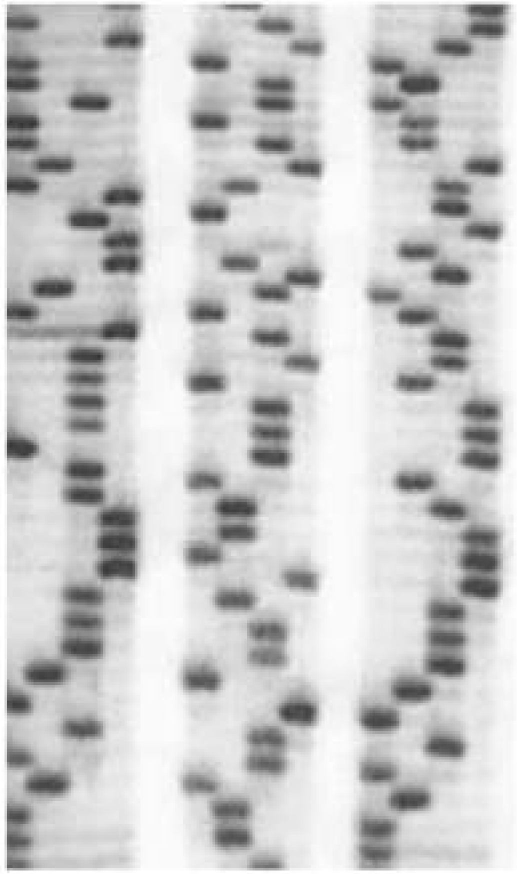


Sanger sequencing – Step 2

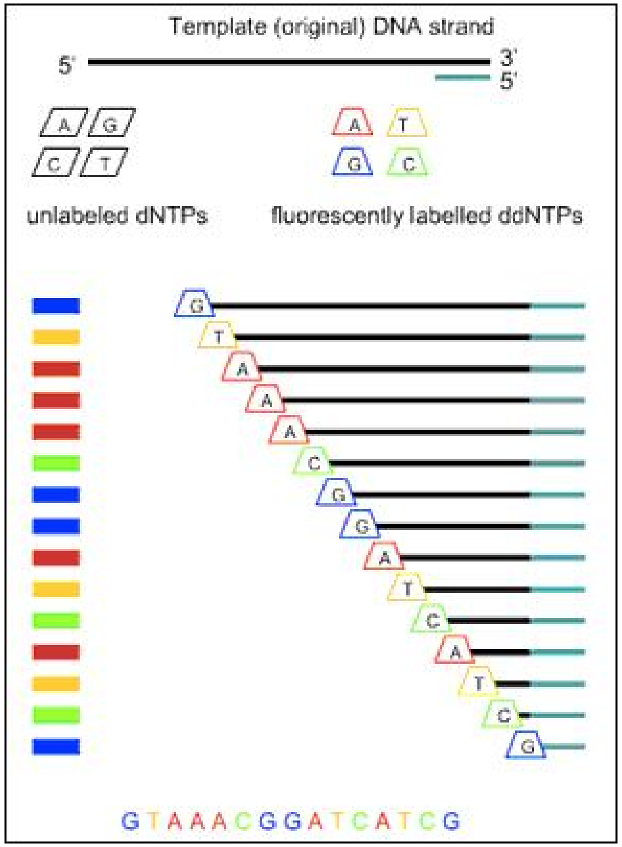
- Interpret results from gel eletrophoresis.

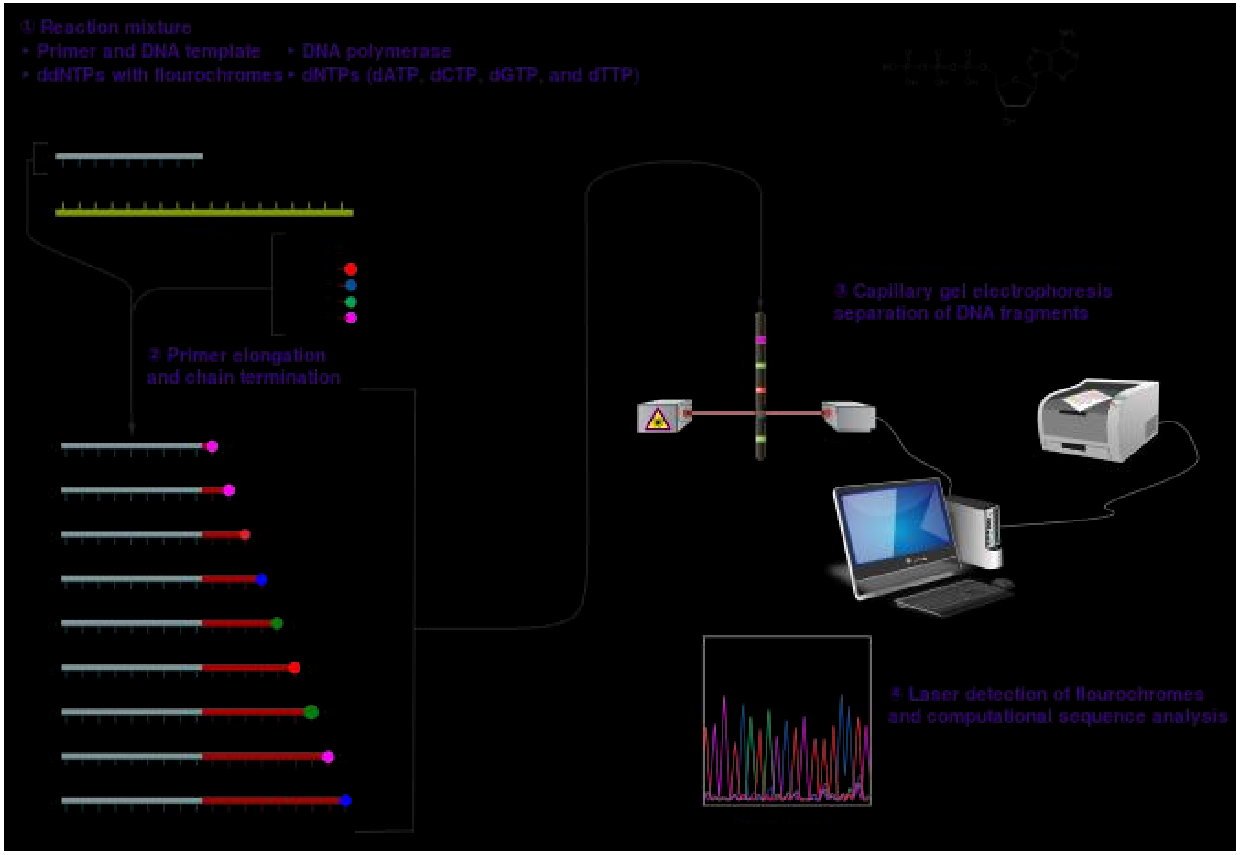


A	C	G	T	SEQUENCE (END)
			—	T (3')
	—			C
	—			C
		—		G
		—		G
			—	T
		—		G
—				A
	—			C
	—			C
		—		G
			—	T
	—			C
—				A (5')



Automated Sanger sequencing

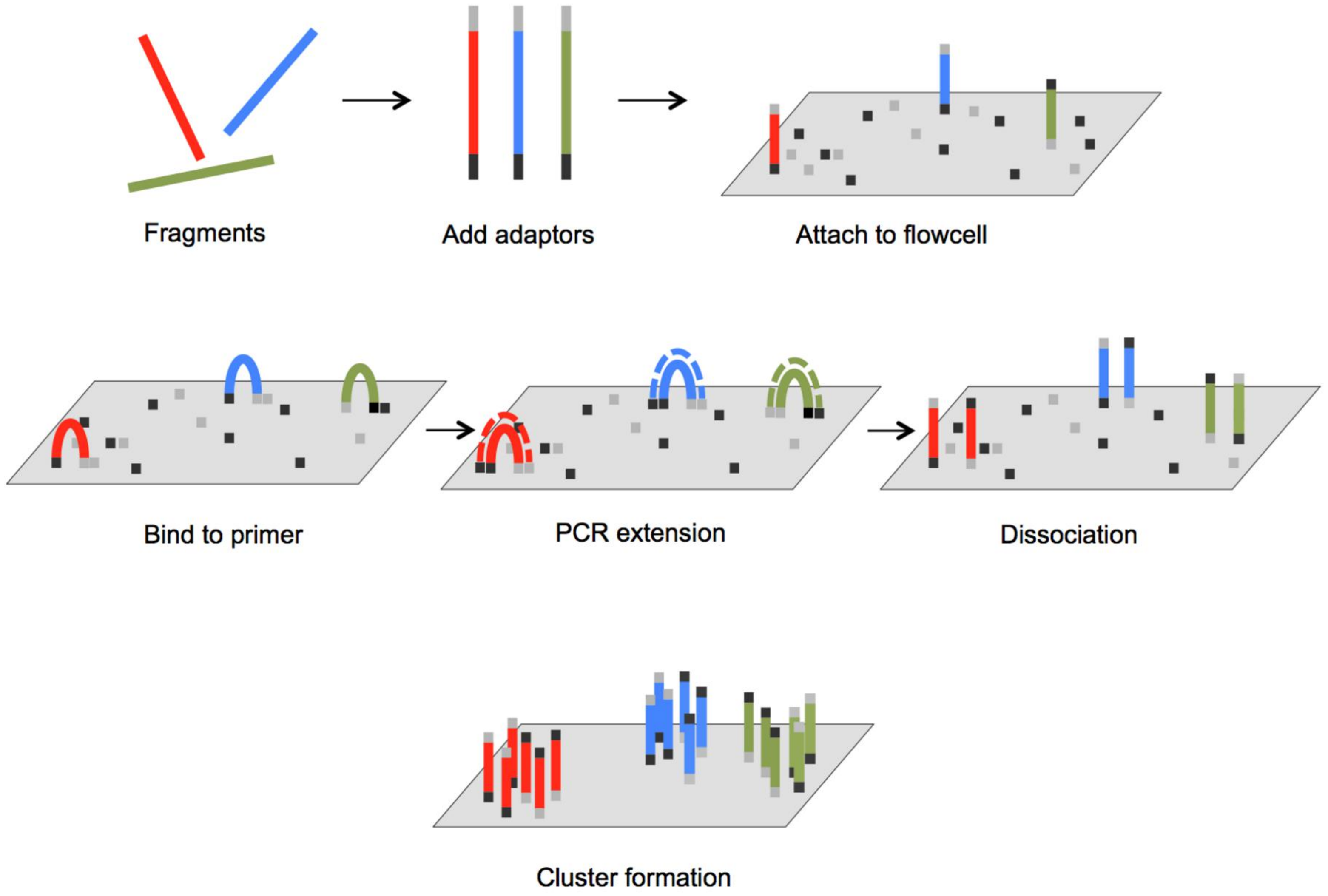




① Reaction mixture
• Primer and DNA template
• DNA polymerase
• ddNTPs with flourochromes
• dNTPs (dATP, dCTP, dGTP, and dTTP)
② Primer elongation and chain termination
③ Capillary gel electrophoresis separation of DNA fragments
④ Laser detection of flourochromes and computational sequence analysis

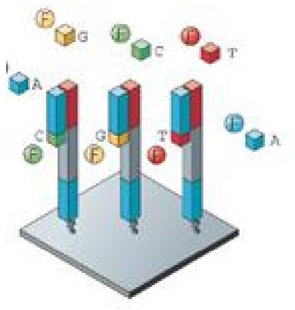
- Sanger Sequencing Summary: 384 * 1kb / 3 hours

Illumina – Cluster Generation



Illumina – Sequencing Process

1. Incorporate all 4 nucleotides, each label with a different dye

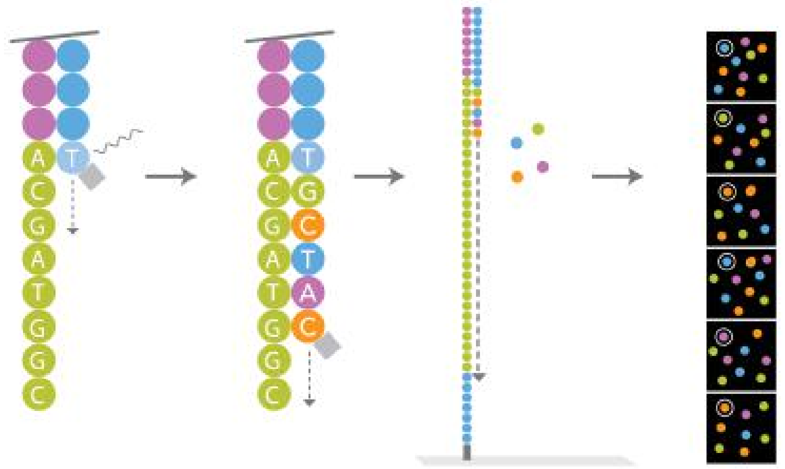


2. Wash, 4-color imaging

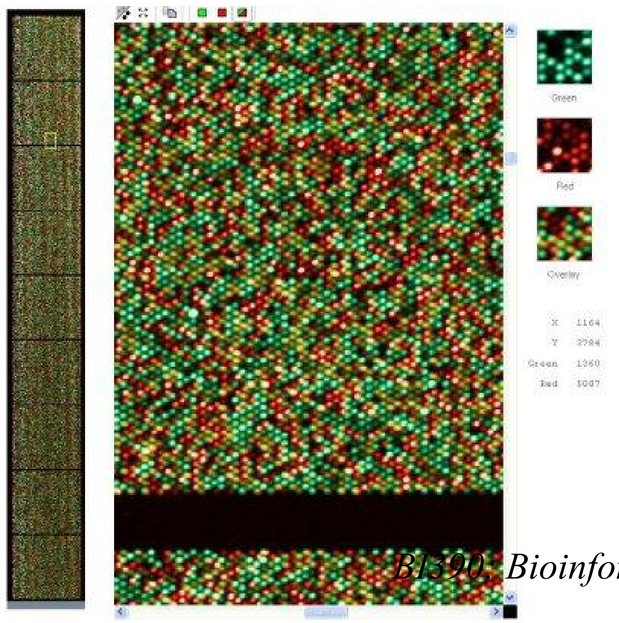
4. Repeat cycles

3. Cleave dye and terminating groups, wash

Illumina – Sequencing Process



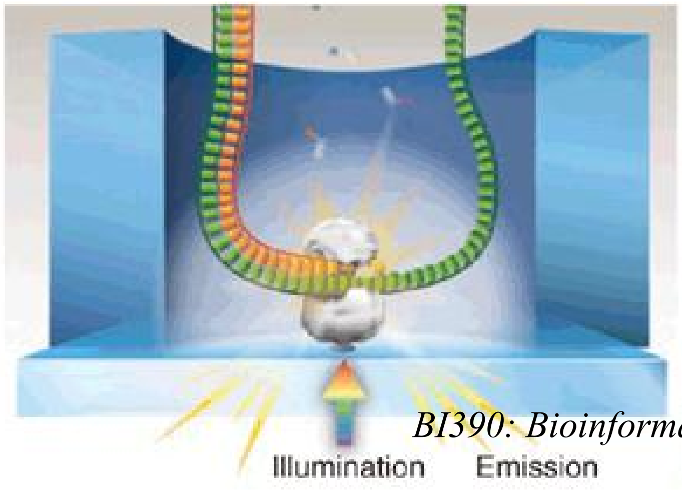
Cycle 1
2
3
4
5
6

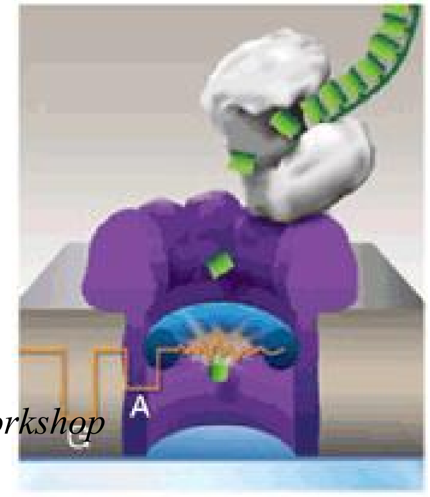


– https://www.youtube.com/watch?v=fCd6B5HRaZ8

Third Generation

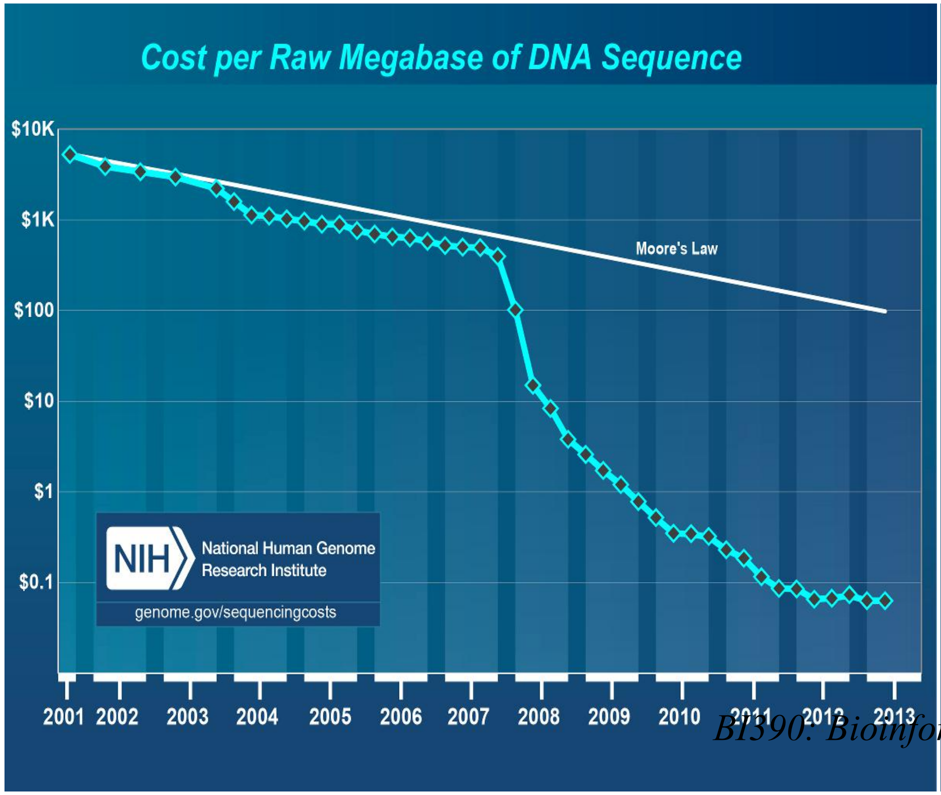
- Single molecule sequencing: no amplification.
- Fewer but much longer reads.
- Good for long reads, but not for read count applications.
- Still under development.
  - http://www.youtube.com/watch?v=v8p4ph2MAvI
  - https://www.youtube.com/watch?v=3UHw22hBpAk

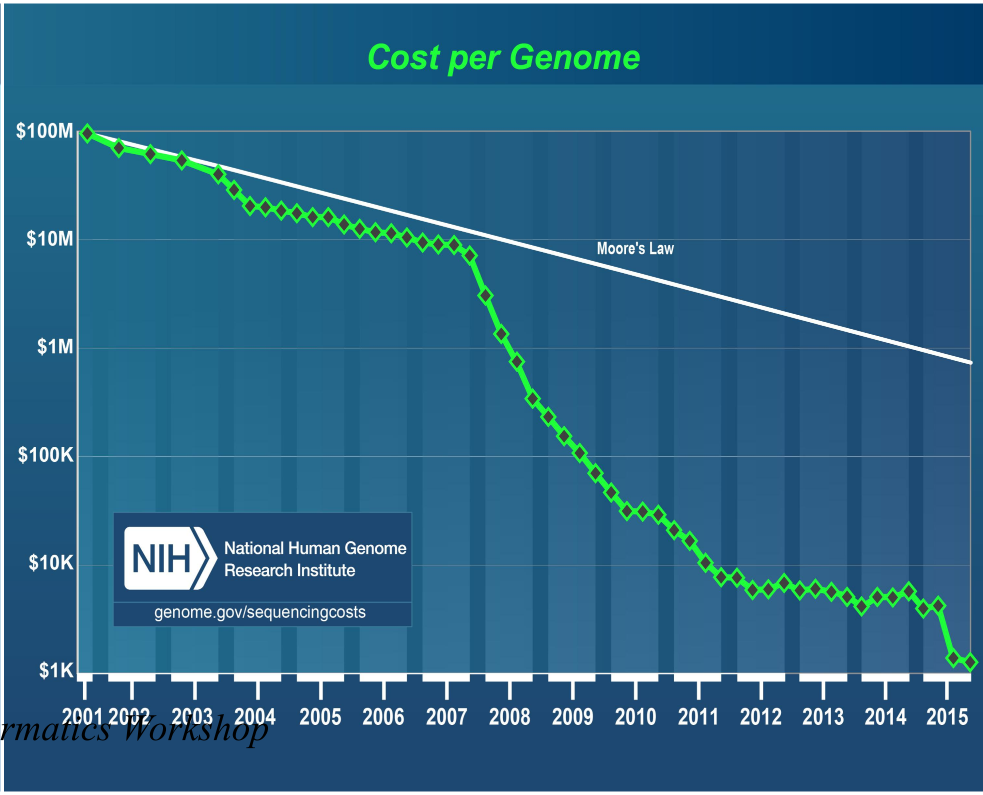




High Throughput Sequencing

- Big (data), fast (speed), cheap (cost), flexible (applications).
- Bioinformatic analyses become the bottleneck.





Outline

- Sequencing technologies.
- Fastq and FASTQC.
- Sequence mapping algorithms:
  - Spaced seed.
  - Borrows-Wheeler transformation & LF mapping.
  - Suffix Tree.
- Alignment output: SAM and BED.

FASTQ File

- Format:
  1. Sequence ID.
  2. Sequence.
  3. Quality ID.
  4. Quality score.

```
@HWI-EAS305:1:1:1:991#0/1
GCTGGAGGTTCAGGCTGGCCGGATTTAAACGT
AT
+HWI-EAS305:1:1:1:991#0/1
MVXUWVRKTWWULRQQMMWWBBBBBBBBBBBB
BB
@HWI-EAS305:1:1:1:201#0/1
AAGACAAAGATGTGCTTTCTAAATCTGCACTAA
T
+HWI-EAS305:1:1:1:201#0/1
PXX[[[[XTXYXTTWYYY[XXWWW[TMTVXWBBB
```

- Quality:
  - ASCII of: sequence quality + 33.
  - $-10 \log_{10}$ Pr(base is wrongly sequenced).

Worst quality Best quality

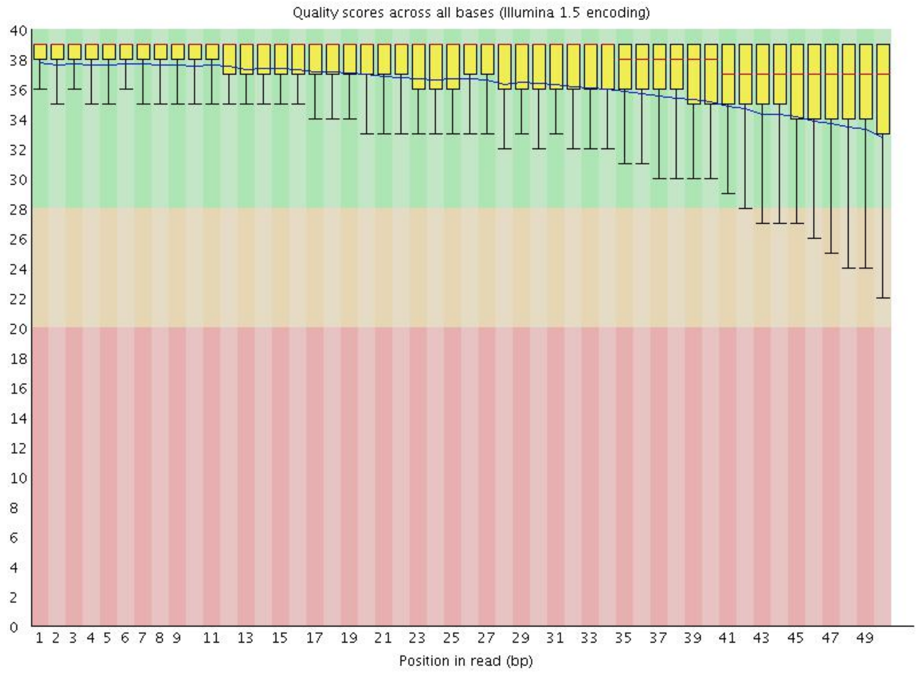
```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Why Quality Control?

- Sequencer output:
  - Sequence “reads” + quality = FASTQ file.
- Is the quality of my sequenced data OK?
- If something is wrong can I fix it?
- Problem: FASTQ are massive files!
- Common tool: FASTQC.
  - http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

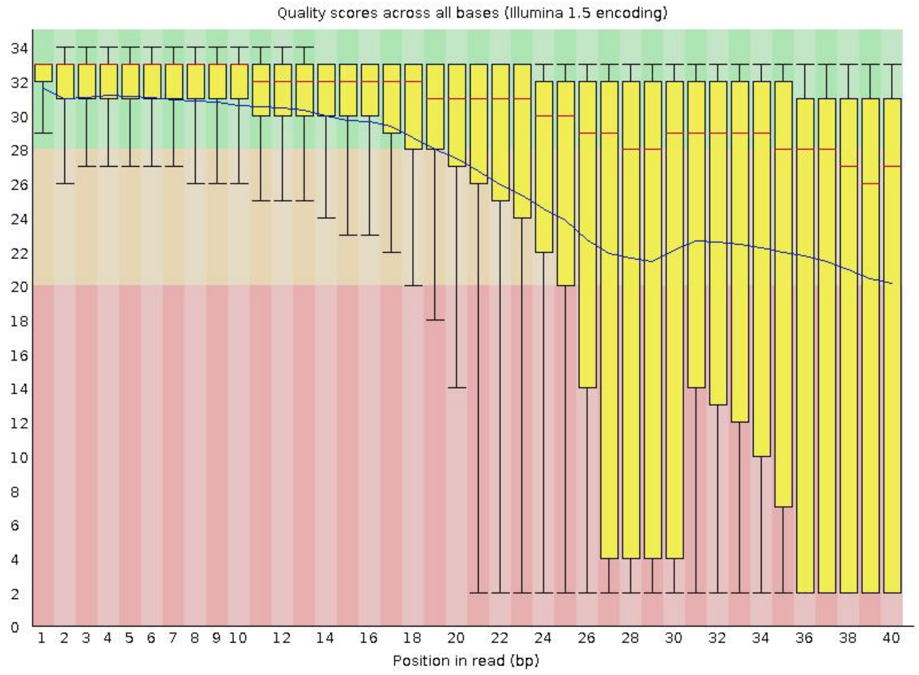
FASTQC: Per Base Sequence Quality

Good quality!



- Consistent.
- High-quality along the read.

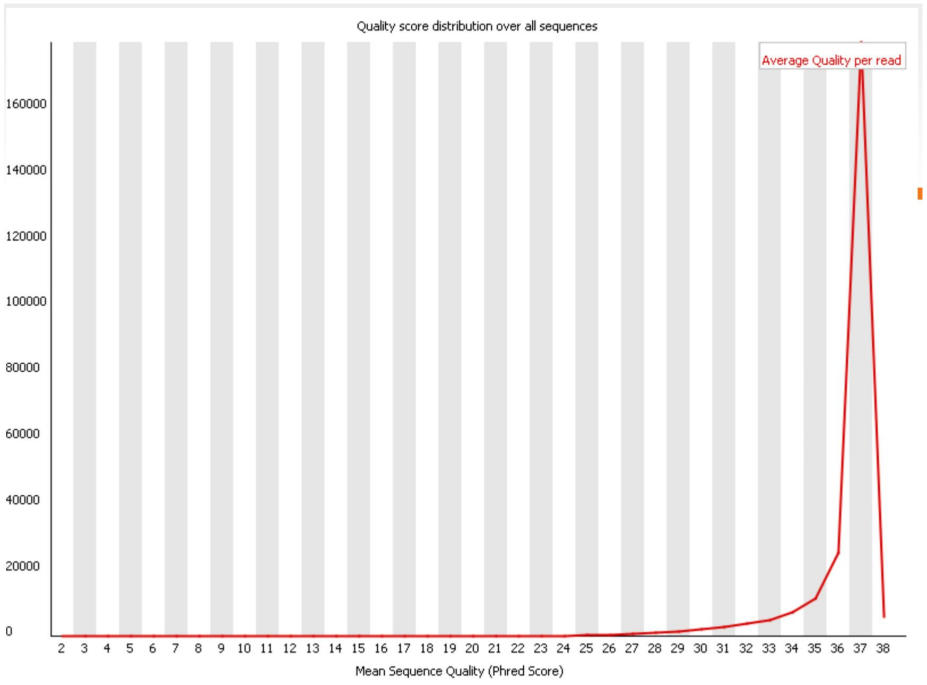
Poor quality!



- High Variance.
- Quality decreases at the 3'-end.

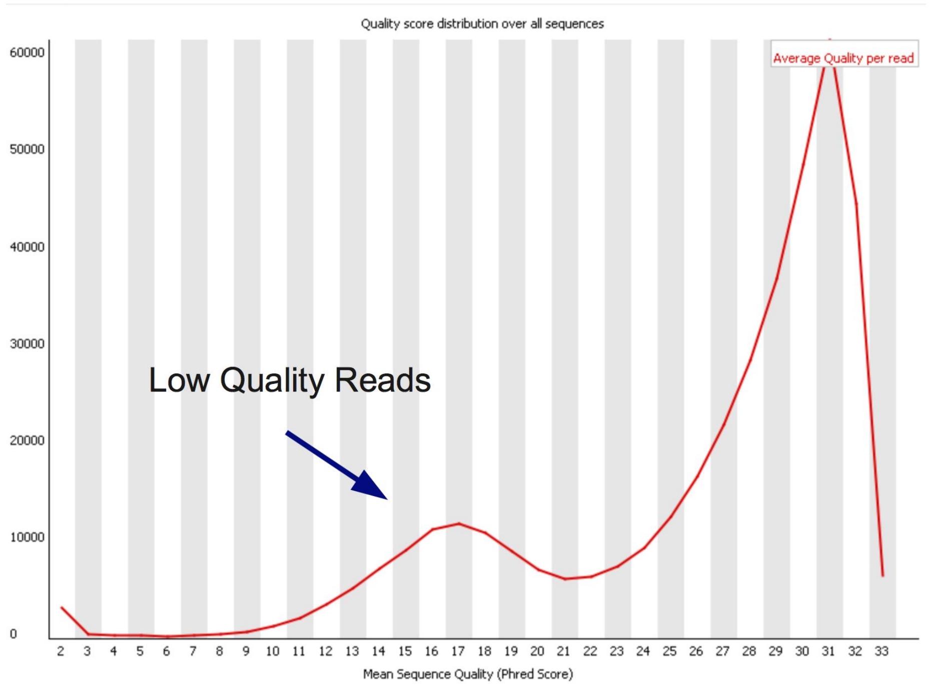
FASTQC: Per Sequence Quality Distribution

Good quality!



- Most are high-quality sequences.

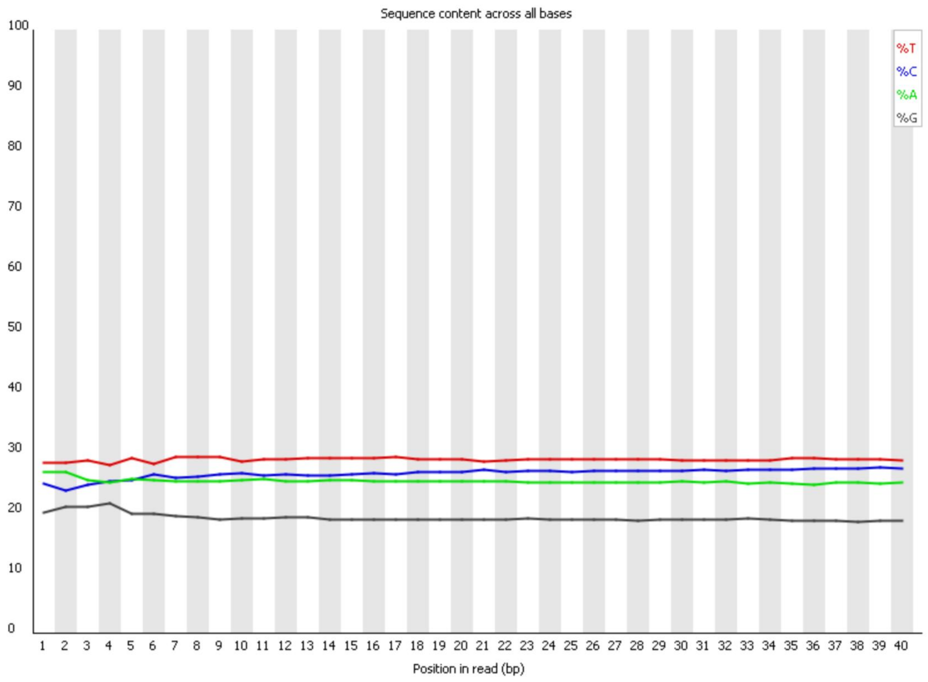
Poor quality!



- Distribution is not uniform.
- Presence of low quality reads.

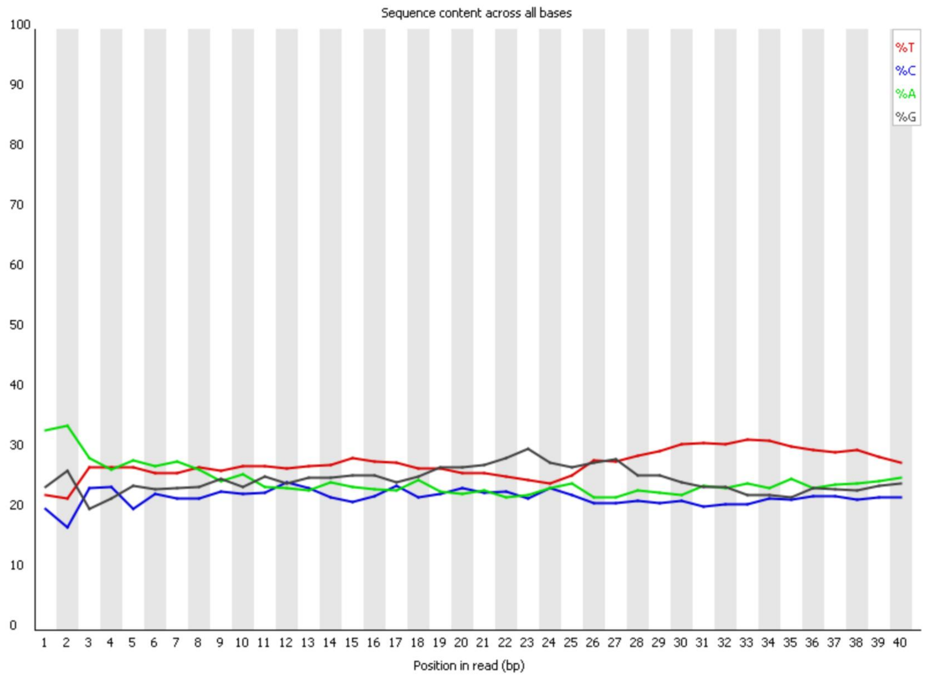
FASTQC: Nucleotide Content Per Position

Good quality!



- Smooth over length.

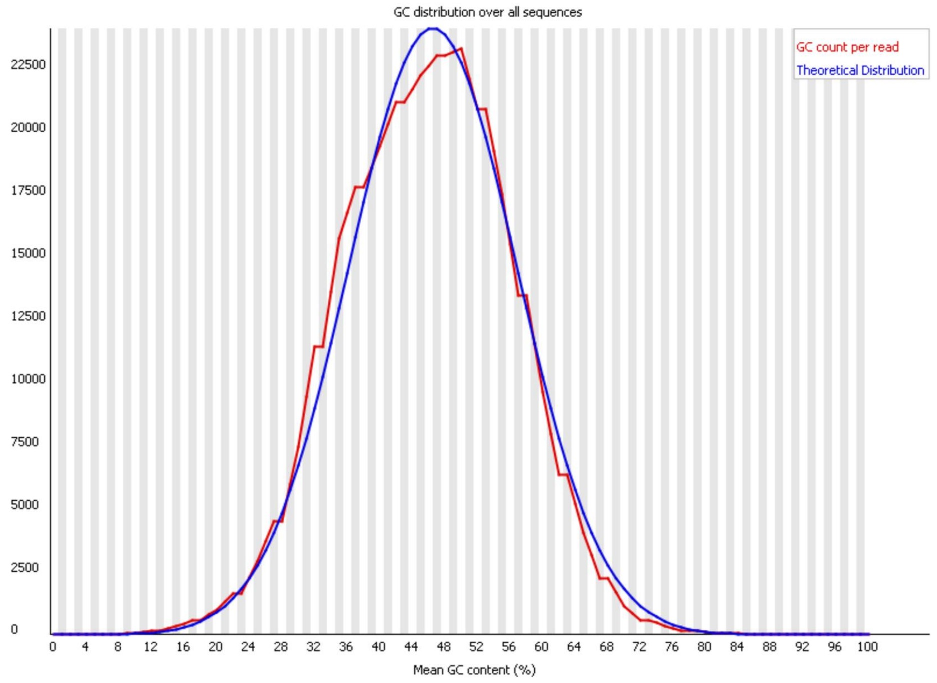
Poor quality!



- Sequence-position bias.

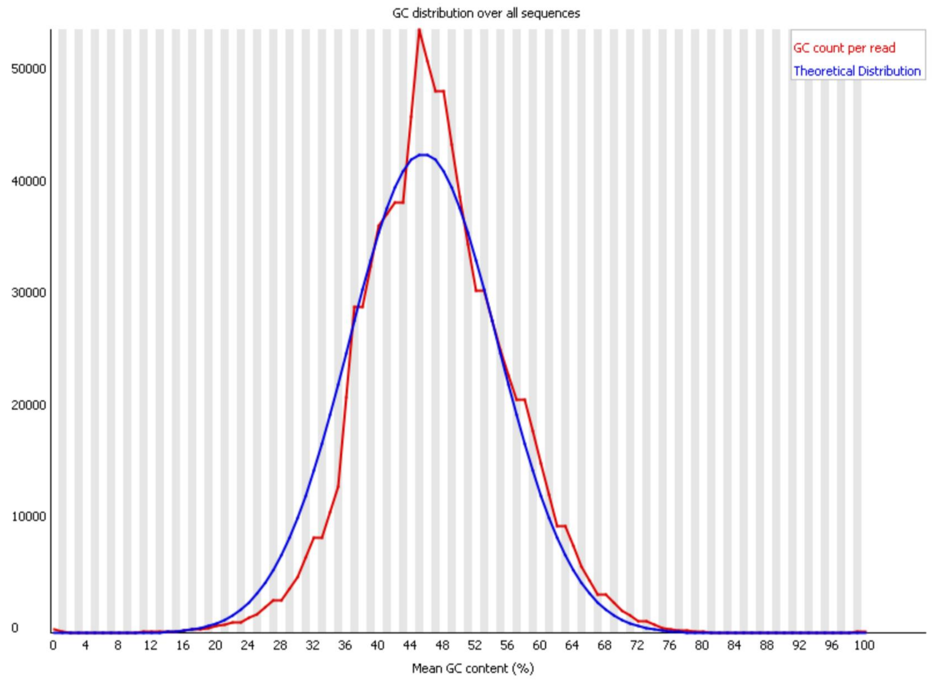
FASTQC: Per Sequence GC Content

Good quality!



- Fits with expectation.

Poor quality!

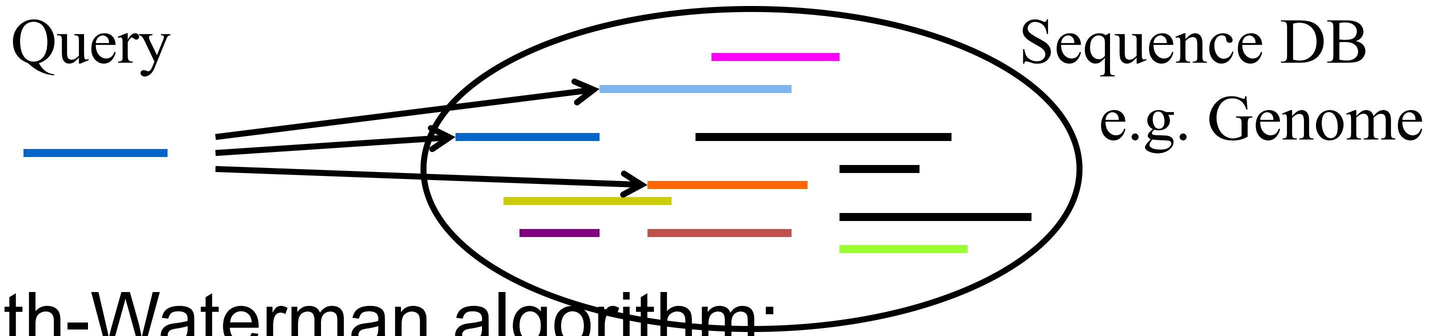


- Does not fit with expectation.

Outline

- Sequencing technologies.
- Fastq and FASTQC.
- Sequence mapping algorithms:
  - Spaced seed.
  - Borrows-Wheeler transformation & LF mapping.
  - Suffix Tree.
- Alignment output: SAM and BED.

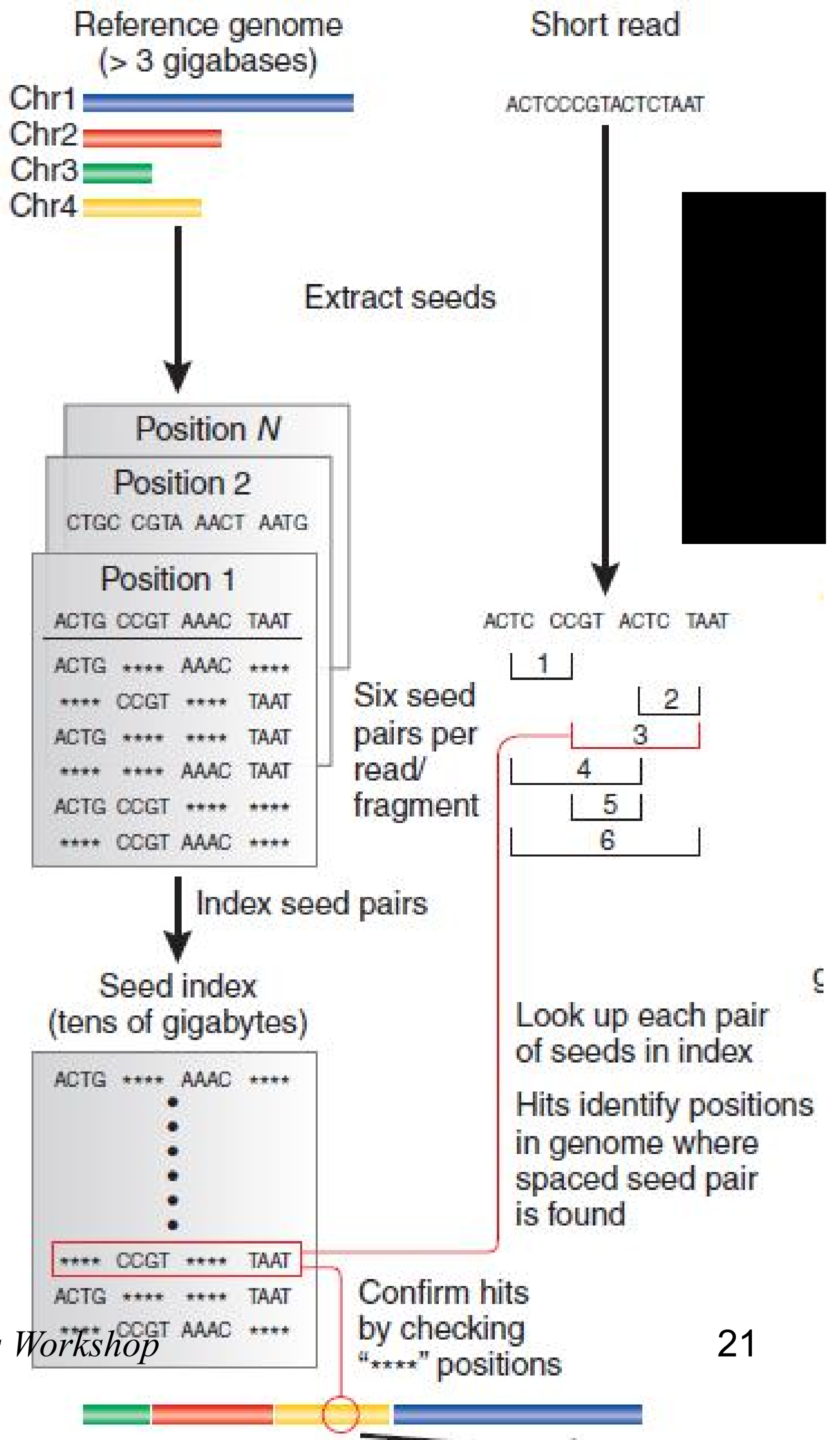
Read Mapping



- Smith-Waterman algorithm:
  - Deterministic approach using dynamic programming.
- Mapping hundreds of millions of reads back to the reference genome is both computation and memory intensive and thus slow.
- Most mappers allow ~2 mismatches within first 30bp (4^{28} could still uniquely identify most 30bp sequences in a 3GB genome), slower when allowing indels.

Spaced Seed Alignment

- 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.
- Report results to the user.

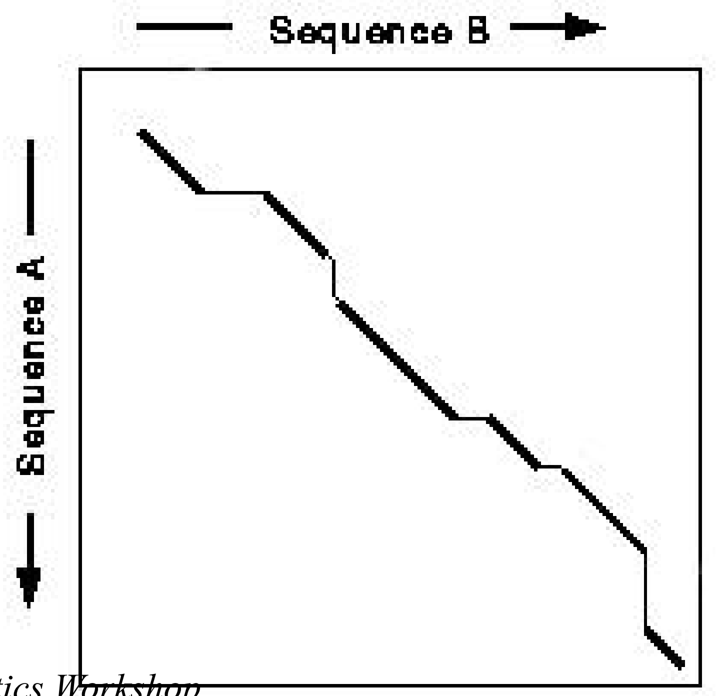


BLAST Algorithm Steps

- Altschul et al.
  - http://www.sciencedirect.com/science/article/pii/S0022283605803602
  - https://academic.oup.com/nar/article/25/17/3389/1061651/
- Break DB sequences into k-mer words and hash their locations to speed later searches.
- For each k-mer in query, find possible DB k-mers that matches well with it.
- Only words with $\geq$ T cutoff score are kept.

BLAST Algorithm Steps

- For each DB sequence with a high scoring word, try to extend it in both ends.
  - Form HSP (High-scoring Segment Pairs).
- Keep only statistically significant HSPs.
  - Based on the scores of aligning 2 random seqs.
- Use Smith-Waterman* algorithm to join the HSPs and get optimal alignment.



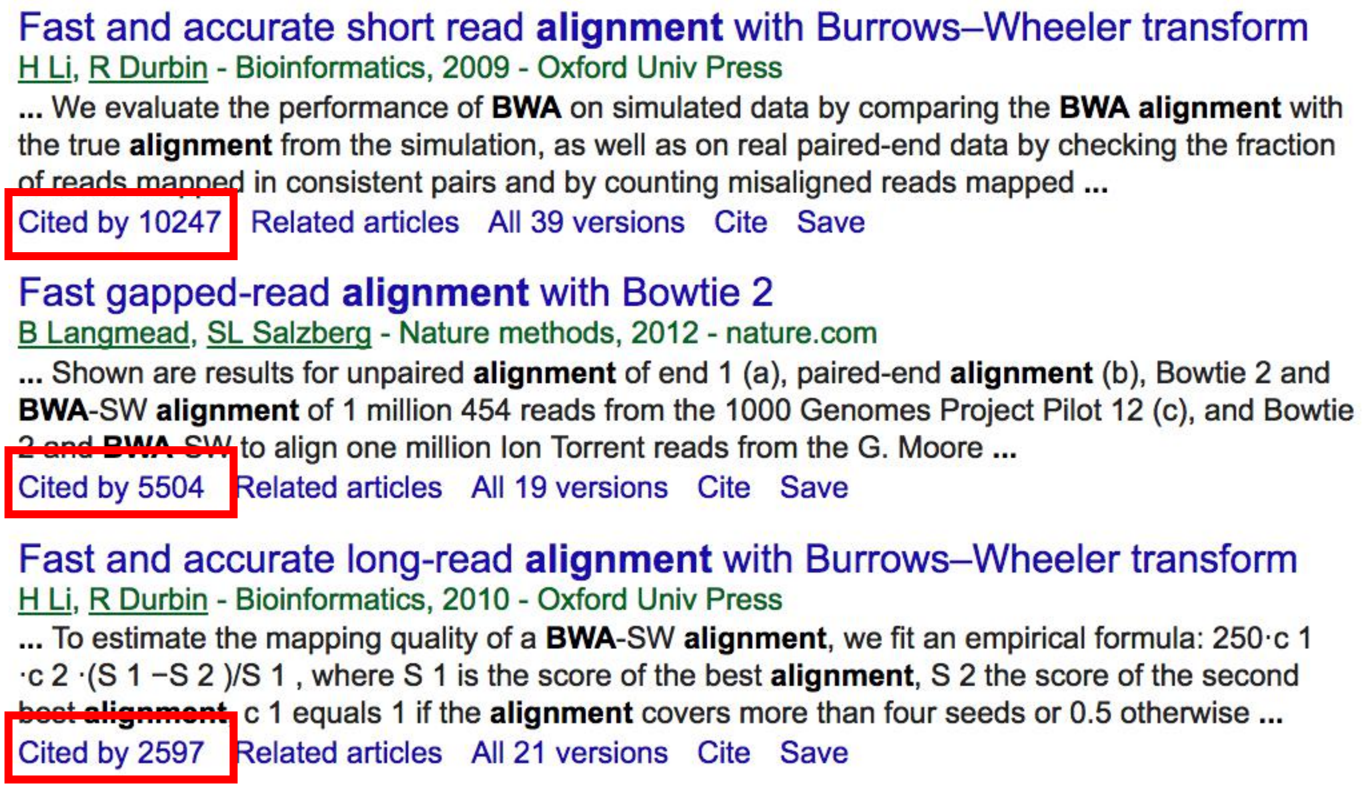
- *https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm
- *https://www.youtube.com/watch?v=lVRSFaGCGeE

Outline

- Sequencing technologies.
- Fastq and FASTQC.
- Sequence mapping algorithms:
  - Spaced seed.
  - Borrows-Wheeler transformation & LF mapping.
  - Suffix Tree.
- Alignment output: SAM and BED.

Burrows-Wheeler Alignment

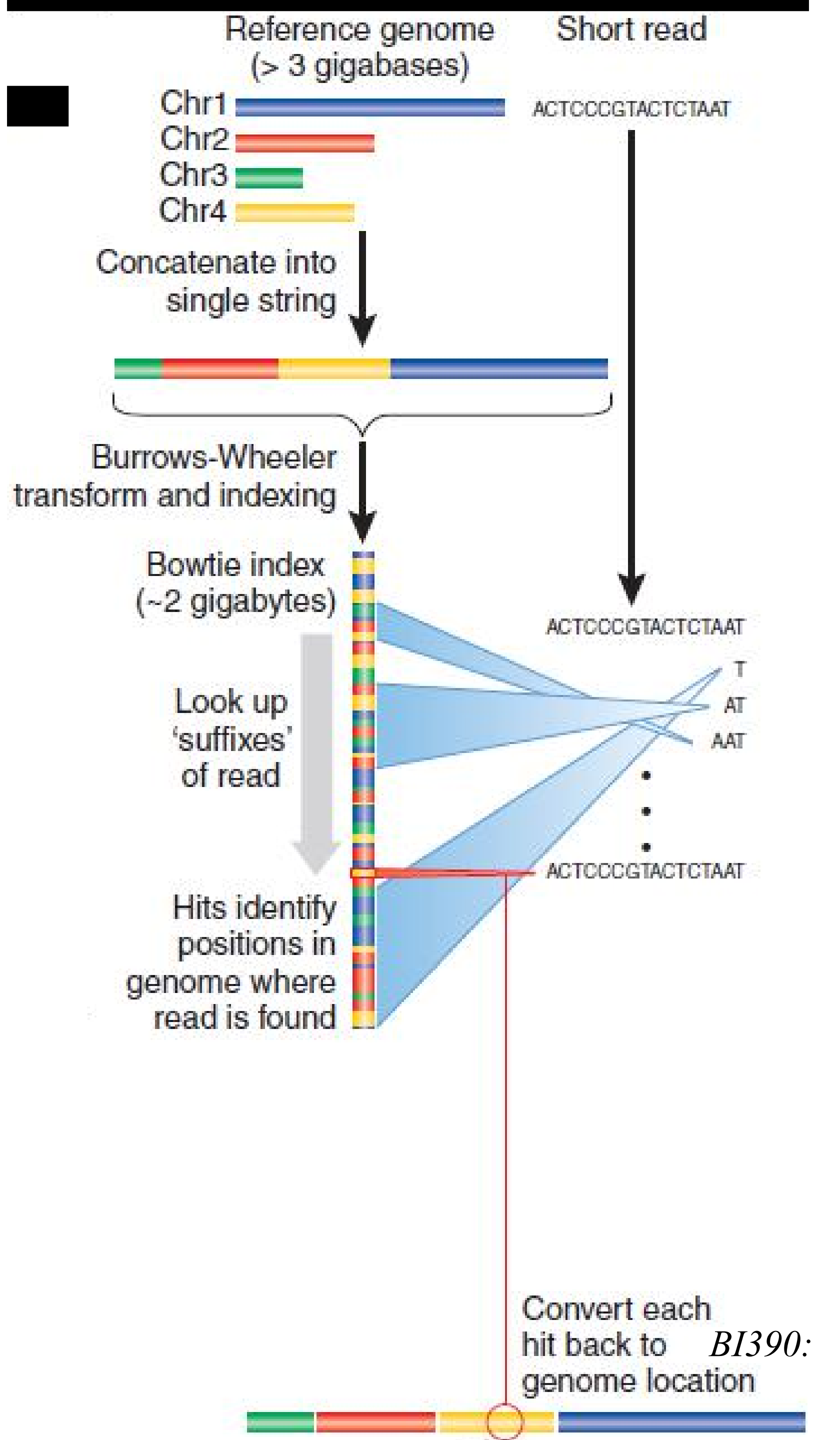
- Two most widely used tools:
  - bwa (http://bio-bwa.sourceforge.net/).
  - bowtie (http://bowtie-bio.sourceforge.net/bowtie2/index.shtml).



Fast and accurate short read **alignment** with Burrows–Wheeler transform
H Li, R Durbin - Bioinformatics, 2009 - Oxford Univ Press
... We evaluate the performance of **BWA** on simulated data by comparing the **BWA alignment** with the true **alignment** from the simulation, as well as on real paired-end data by checking the fraction of reads mapped in consistent pairs and by counting misaligned reads mapped ...
Cited by 10247 Related articles All 39 versions Cite Save

Fast gapped-read **alignment** with Bowtie 2
B Langmead, SL Salzberg - Nature methods, 2012 - nature.com
... Shown are results for unpaired **alignment** of end 1 (a), paired-end **alignment** (b), Bowtie 2 and **BWA**-SW **alignment** of 1 million 454 reads from the 1000 Genomes Project Pilot 12 (c), and Bowtie 2 and **BWA**-SW to align one million Ion Torrent reads from the G. Moore ...
Cited by 5504 Related articles All 19 versions Cite Save

Fast and accurate long-read **alignment** with Burrows–Wheeler transform
H Li, R Durbin - Bioinformatics, 2010 - Oxford Univ Press
... To estimate the mapping quality of a **BWA**-SW **alignment**, we fit an empirical formula: 250·c 1 ·c 2 ·(S 1 −S 2)/S 1 , where S 1 is the score of the best **alignment**, S 2 the score of the second best **alignment**, c 1 equals 1 if the **alignment** covers more than four seeds or 0.5 otherwise ...
Cited by 2597 Related articles All 21 versions Cite Save


Reference genome
(> 3 gigabases)
Short read
Chr1
Chr2
Chr3
Chr4
ACTCCCGTACTCTAAT
Concatenate into single string
Burrows-Wheeler transform and indexing
Bowtie index (~2 gigabytes)
ACTCCCGTACTCTAAT
T
AT
AAT
Look up 'suffixes' of read
ACTCCCGTACTCTAAT
Hits identify positions in genome where read is found
Convert each hit back to genome location

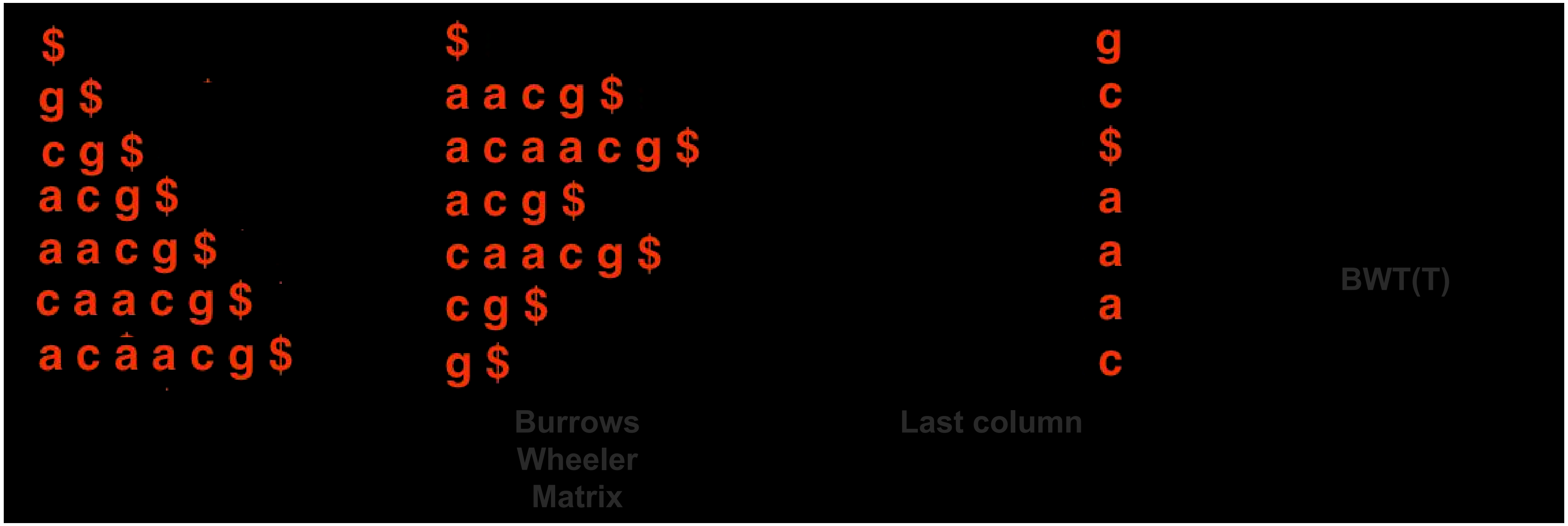
Burrows-Wheeler

- Use Burrows-Wheeler transform to store entire reference genome as a lookup index.
- Align tag base by base from the end.
- All active locations are reported.
- If no match is found, then back up and try a substitution.
- Ben Langmead videos:
  - https://www.youtube.com/watch?v=4n7NPk5lwbI
  - https://www.youtube.com/watch?v=kvVGj5V65io

Trapnell & Salzberg, Nat Biotech 2009.

Burrows-Wheeler Transform

- Reversible permutation used originally in compression
- Database sequence T = acaacg$



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

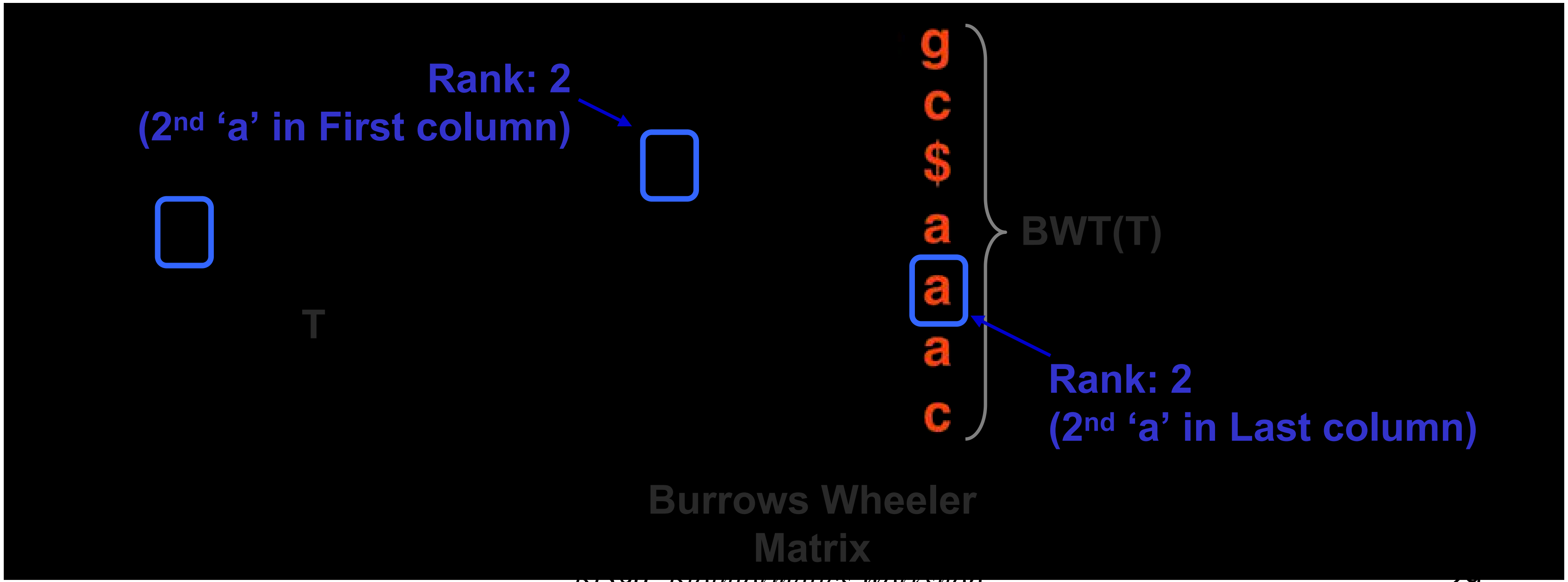
Burrows M, Wheeler DJ: **A block sorting lossless data compression algorithm**. *Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124; 1994

Burrows-Wheeler Transform

- Why BWT is useful for compression?
  - Once BWT(T) is built, everything else is discarded.
  - First column of BWM can be derived by sorting BWT(T).
  - Characters will tend to cluster together:
    - BWT(T) = gc$aaac -> compression -> gc$3ac
- How can we recreate T using BWT(T)?
  - LF mapping.
- How to use BWT(T) to retrieve alignments, given a query sequence Q?

BWT: LF Mapping

- Property that makes BWT(T) reversible is “LF Mapping”.
  - ith occurrence of a character in **L**ast column is the same *text* occurrence as the ith occurrence in **F**irst column.

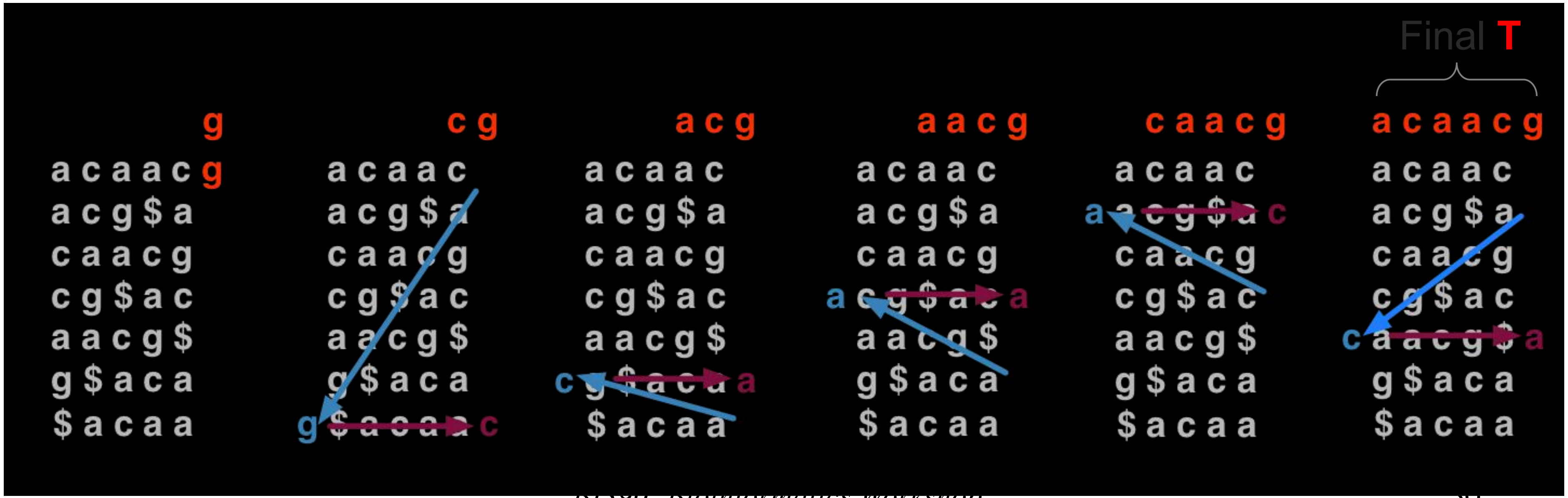


BWT: LF Mapping

- To recreate T from BWT(T), repeatedly apply rule:

  T = **BWT**[**LF**(i)] + **T**; i = **LF**(i)

  – Where **LF**(i) maps row i to row whose first char corresponds to i row's last char using LF Mapping.



Recovering T from bwt(T)

```
def recover(bwt):
        “””recover original string from its bwt transform
        pos = 0
        ans = endChar # $-terminated here
        for in range(1, bwt.length):
                ans = bwt.charAt(pos) + ans # update T
                pos = inverse(pos, bwt)    # update pos LF
        return ans
```

Recovering T from bwt(T)

```
def inverse(pos, bwt):
        “””update the position from the current position
        ch = bwt.charAt(pos)
        chCode = ch.chCodeAt(0)
        return rank[chCode] + occ(ch, bwt, pos)
```

rank[chCode] 表示的是bwt中字母表中在chCode之前的字母个数。

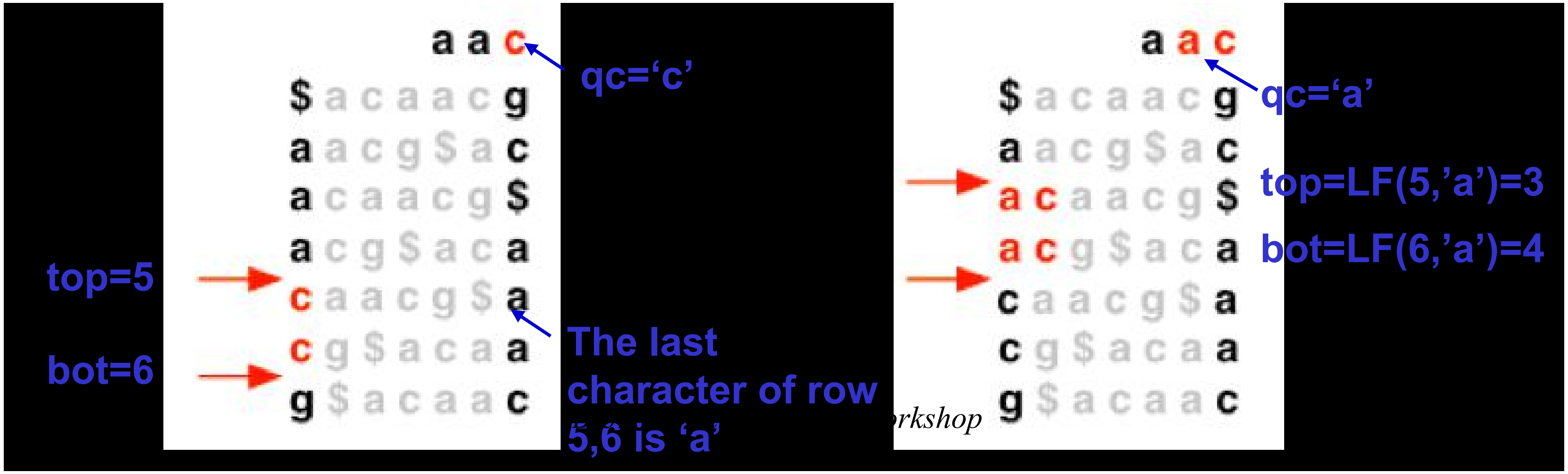
```
def occ(ch, bwt, pos):
        “”” return the occurrence of ch in bwt before pos
        pass
```

BWT(T) to retrieve alignments

- Query Q = aac; DB T = acaacg$; BWT(T) = gc$aaac
- To match Q in T using BWT(T), repeatedly apply rule:

  top = **LF**(**top**, **qc**); **bot** = **LF**(**bot**, **qc**)

  – Where **qc** is the next character in Q (right-to-left) and **LF**(i, **qc**) maps row i to the row whose first character corresponds to i's last character *as if it were* **qc**.

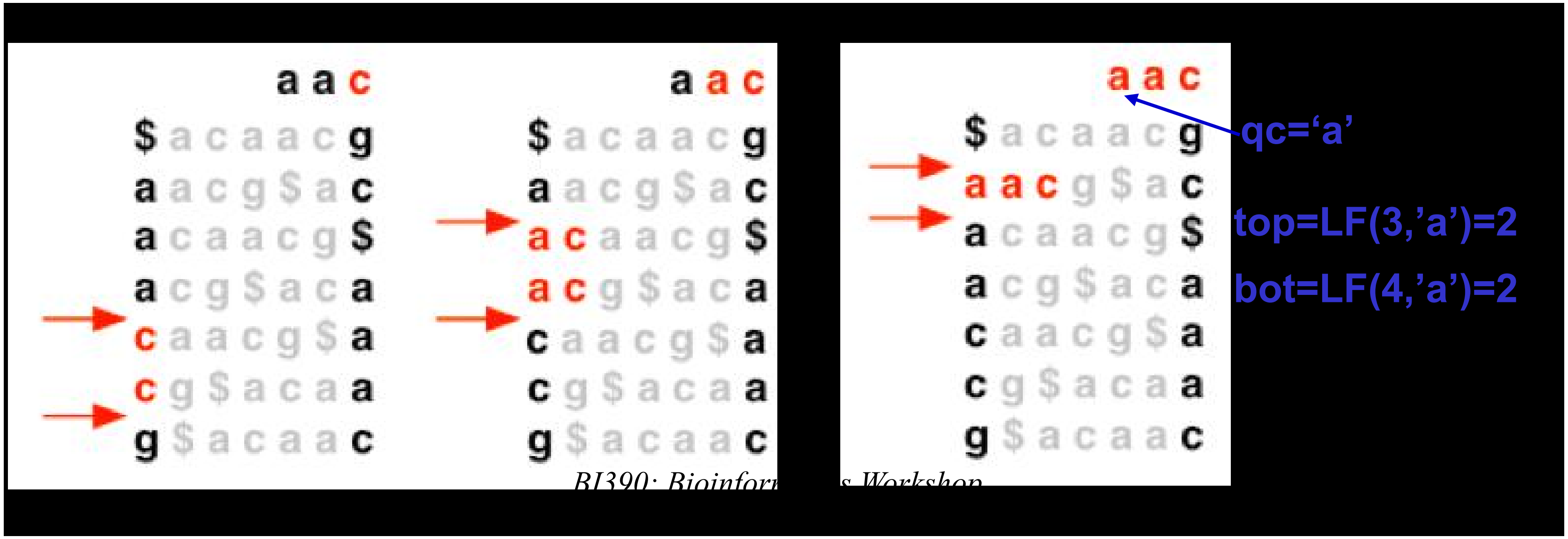


BWT(T) to retrieve alignments

- To match Q in T using BWT(T), repeatedly apply rule:

  top = **LF**(**top**, **qc**); **bot** = **LF**(**bot**, **qc**)

  – Where **qc** is the next character in Q (right-to-left) and **LF**(i, **qc**) maps row i to the row whose first character corresponds to i's last character *as if it were* ***qc***.



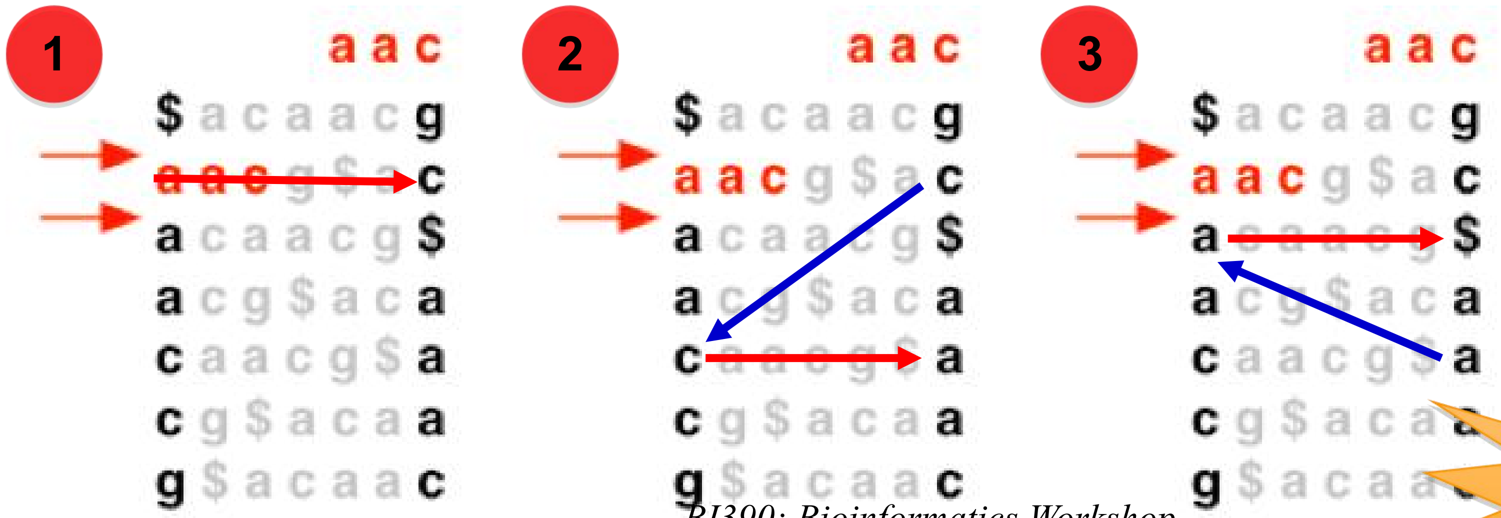
BWT(T) to retrieve alignments

- In progressive rounds, **top** & **bot** delimit the range of rows beginning with progressively longer suffixes of Q (from right to left).
- If range becomes empty the query suffix (and therefore the query) does not occur in the text.
- If no match, instead of giving up, try to “backtrack” to a previous position and try a different base (mismatch, much slower).

BWT(T) to retrieve alignments

- How to recover the query sequence (Q) alignment **position** in the reference sequence T?
  - LF mapping!

```
               3
T = acaacg$
        |||
Q =   aac
```

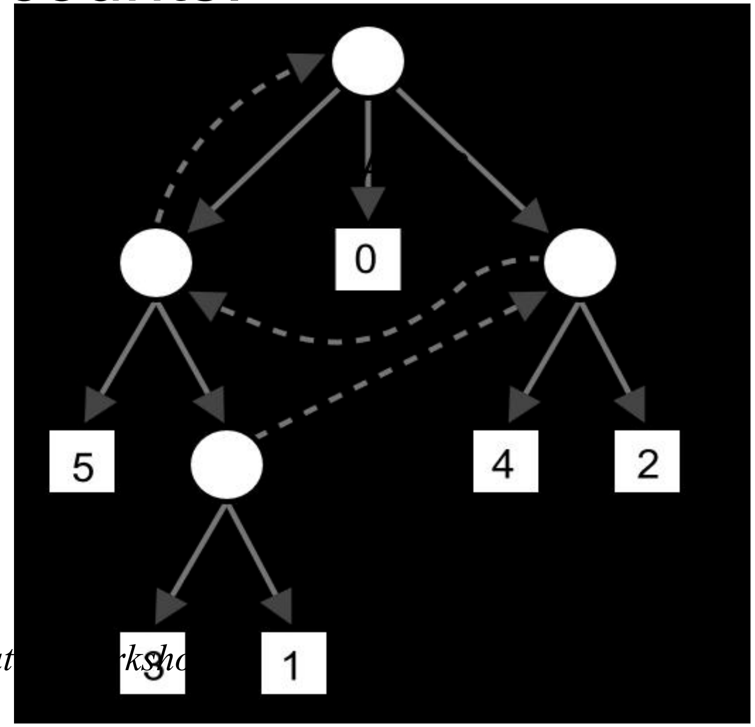


Outline

- Sequencing technologies.
- Fastq and FASTQC.
- Sequence mapping algorithms:
  - Spaced seed.
  - Borrows-Wheeler transformation & LF mapping.
  - Suffix Tree.
- Alignment output: SAM and BED.

Suffix Tree

- Used by alignment tools such as STAR:
  - https://academic.oup.com/bioinformatics/article/29/1/15/272537/
- Very fast and accurate for mapping “paired end” sequences and high read counts.
- O(n) time to build.
  - n = genome length.
- O($m\log_n$) time to search.
  - m = query length.
- Genome index is big.
  - ~15GB



Suffix Tree (Example)

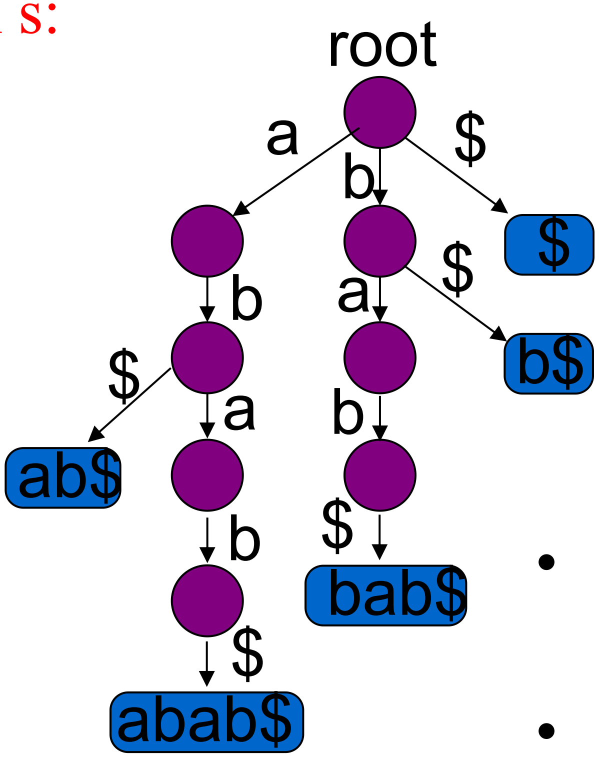
*https://en.wikipedia.org/wiki/Trie

Let s=abab, a suffix tree of s is a compressed trie* of all suffixes of s=abab$

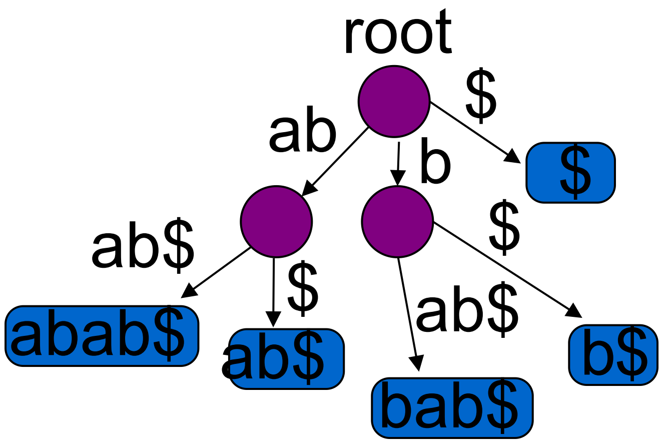
Suffixes from s:

{
$
b$
ab$
bab$
abab$
}

Suffixes trie:



Suffixes tree:



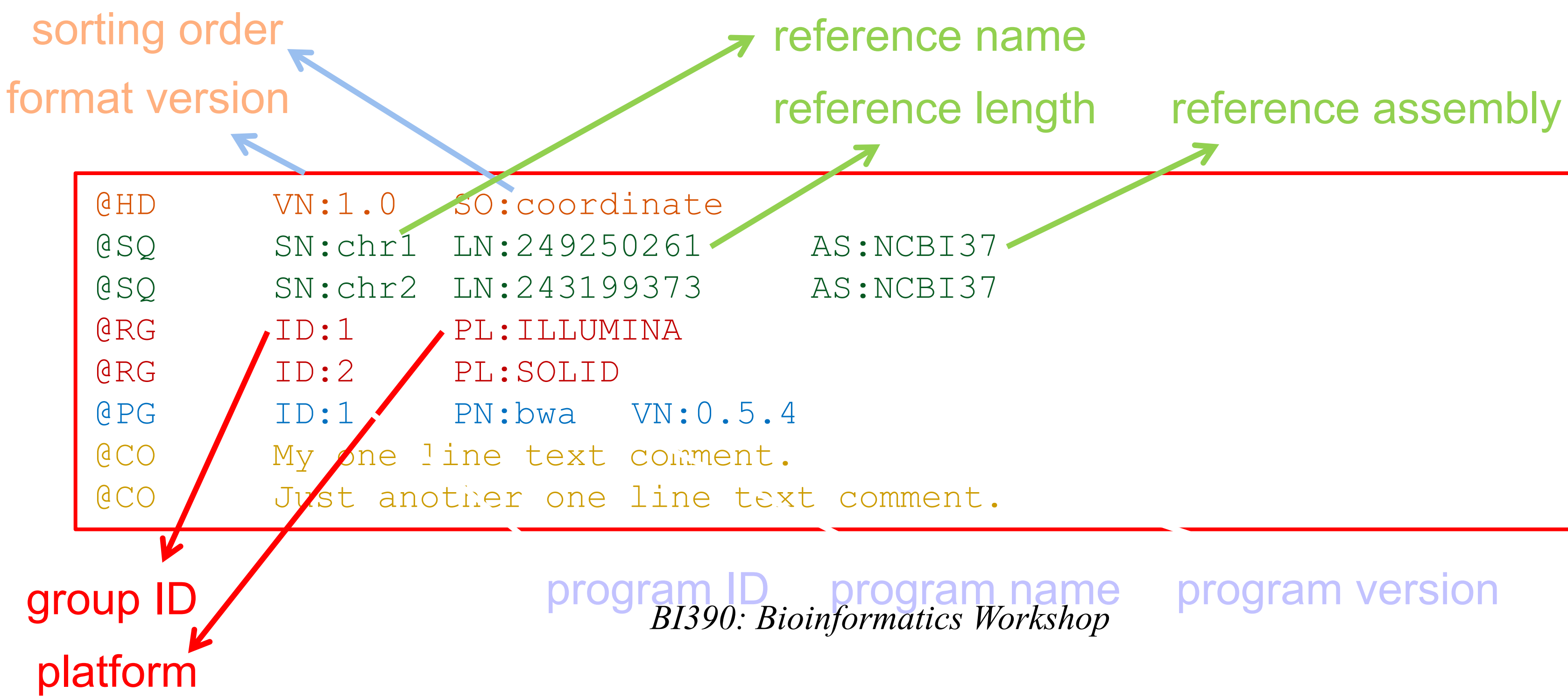
- Parallel between sufix trees and BWT.
- Ben Langmead videos:
  - https://www.youtube.com/watch?v=hLsrPsFHPcQ&t=23s

Outline

- Sequencing technologies.
- Fastq and FASTQC.
- Sequence mapping algorithms:
  - Spaced seed.
  - Borrows-Wheeler transformation & LF mapping.
  - Suffix Tree.
- Alignment output: SAM and BED.

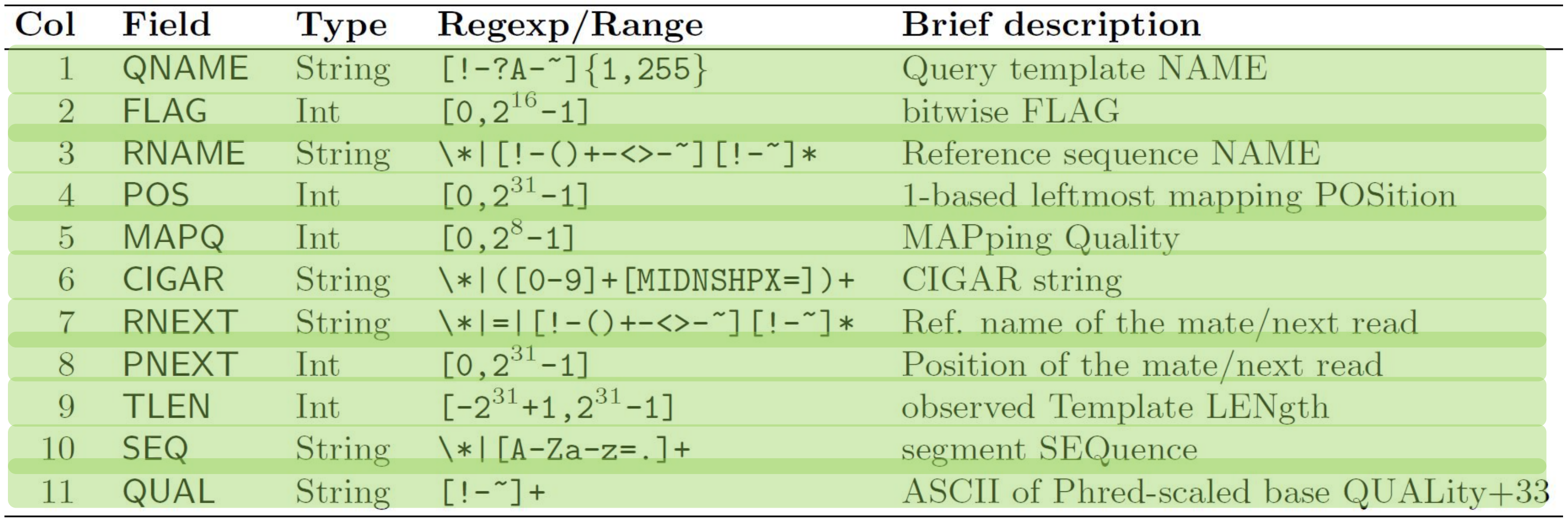
SAM File - Header

- @HD – Header line.
- @SQ – Reference genome information.
- @RG – Read group information.
- @PG – Program (software) information.
- @CO – Commentary line.



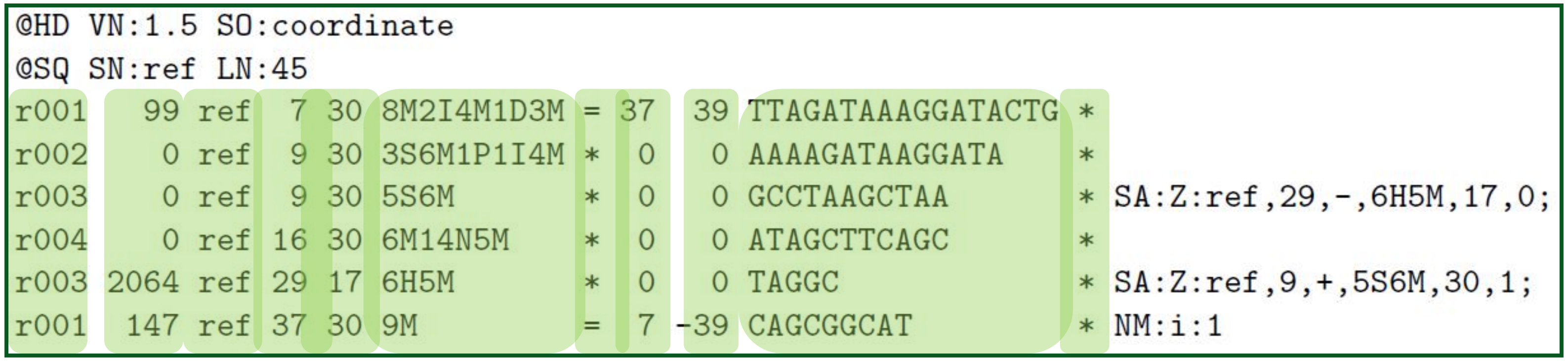
```
@HD      VN:1.0   SO:coordinate
@SQ      SN:chr1  LN:249250261     AS:NCBI37
@SQ      SN:chr2  LN:243199373     AS:NCBI37
@RG      ID:1     PL:ILLUMINA
@RG      ID:2     PL:SOLID
@PG      ID:1     PN:bwa   VN:0.5.4
@CO      My one line text comment.
@CO      Just another one line text comment.
```

SAM File – Fields



Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,255}`	Query template NAME
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME
4	POS	Int	$[0,2^{31}-1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next read
8	PNEXT	Int	$[0,2^{31}-1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31}+1,2^{31}-1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

- Example:



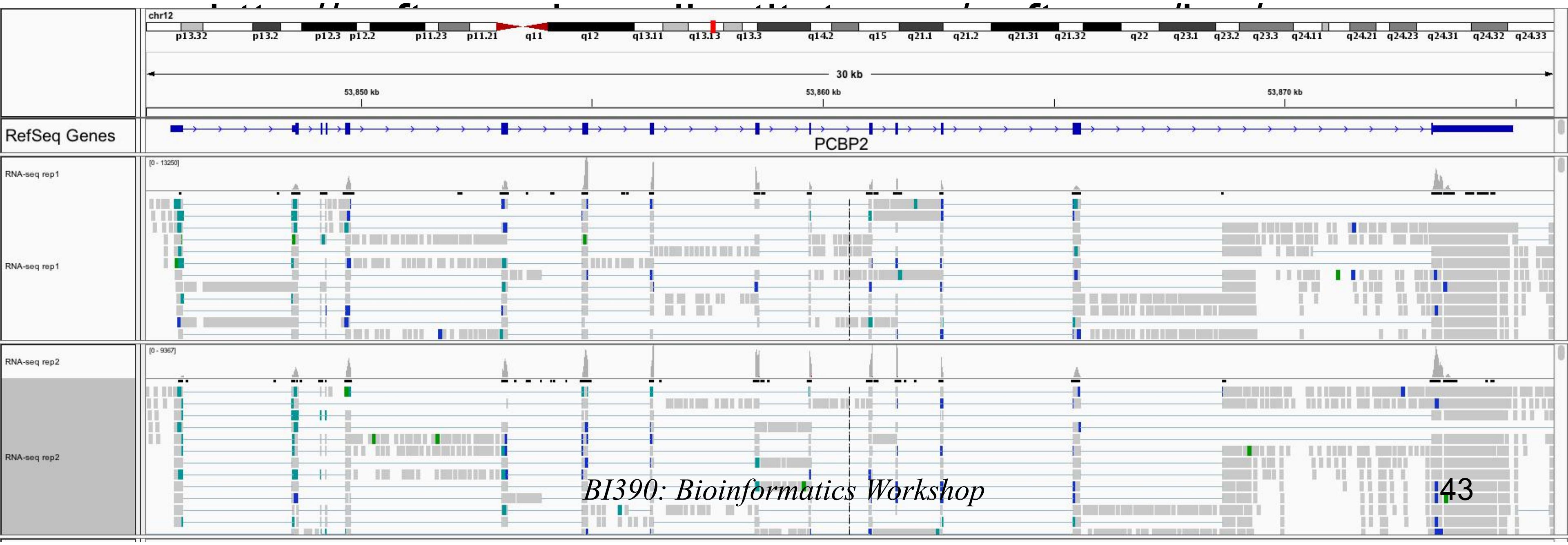
```
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC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

  – https://samtools.github.io/hts-specs/SAMv1.pdf

* Kent et al. (2010) *Bioinformatics*. 26(17):2204.

BAM File

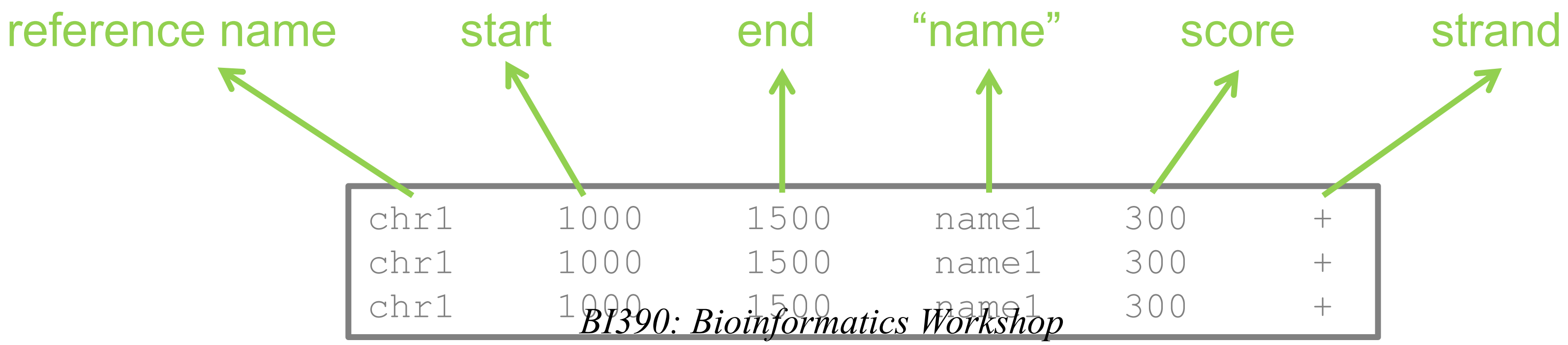
- Compression – BGZF Block Compression*.
- Efficient random access – UCSC Bin/Chunk Scheme*.
- BAI index files.
- Visualize BAM alignments in IGV software:



* Kent et al. (2010) *Bioinformatics*. 26(17):2204.

BED & BigBED Files

- Rarely used to store alignments: usually stores other types of genomic intervals.
- Bed specifications:
  - https://genome.ucsc.edu/FAQ/FAQformat#format1
- BigBed: Binary compressed & indexed BED file*.
- BigBed specifications:
  - https://genome.ucsc.edu/goldenPath/help/bigBed.html



```
chr1      1000      1500      name1      300      +
chr1      1000      1500      name1      300      +
chr1      1000      1500      name1      300      +
```

Summary

- Sequencing technologies: 1st, **2nd**, 3rd generation.
- Illumina has taken most of the market.
- Sequences are stored in FASTQ files.
- After sequencing, perform quality assessment (FASTQC).
- Sequenced “reads” need to be aligned back to reference genome.
- Aligned reads are stored in SAM/BAM files.