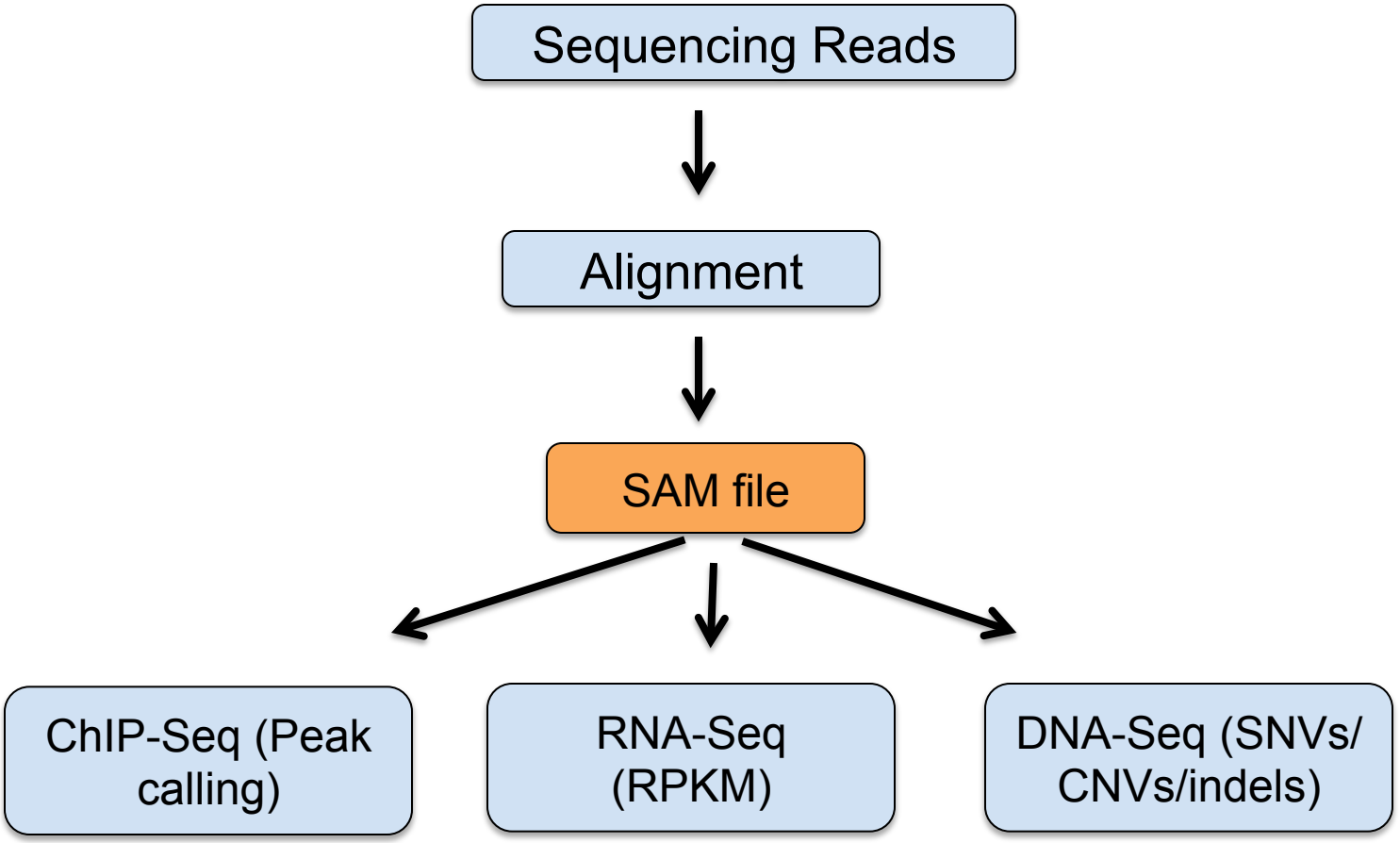
BIO306: Bioinformatics

Lecture 6

Reads Mapping and output format

Wenfei JIN PhD
jinwf@sustc.edu.cn
Department of Biology, SUSTech

The majority of data analysis split off after generating SAM file



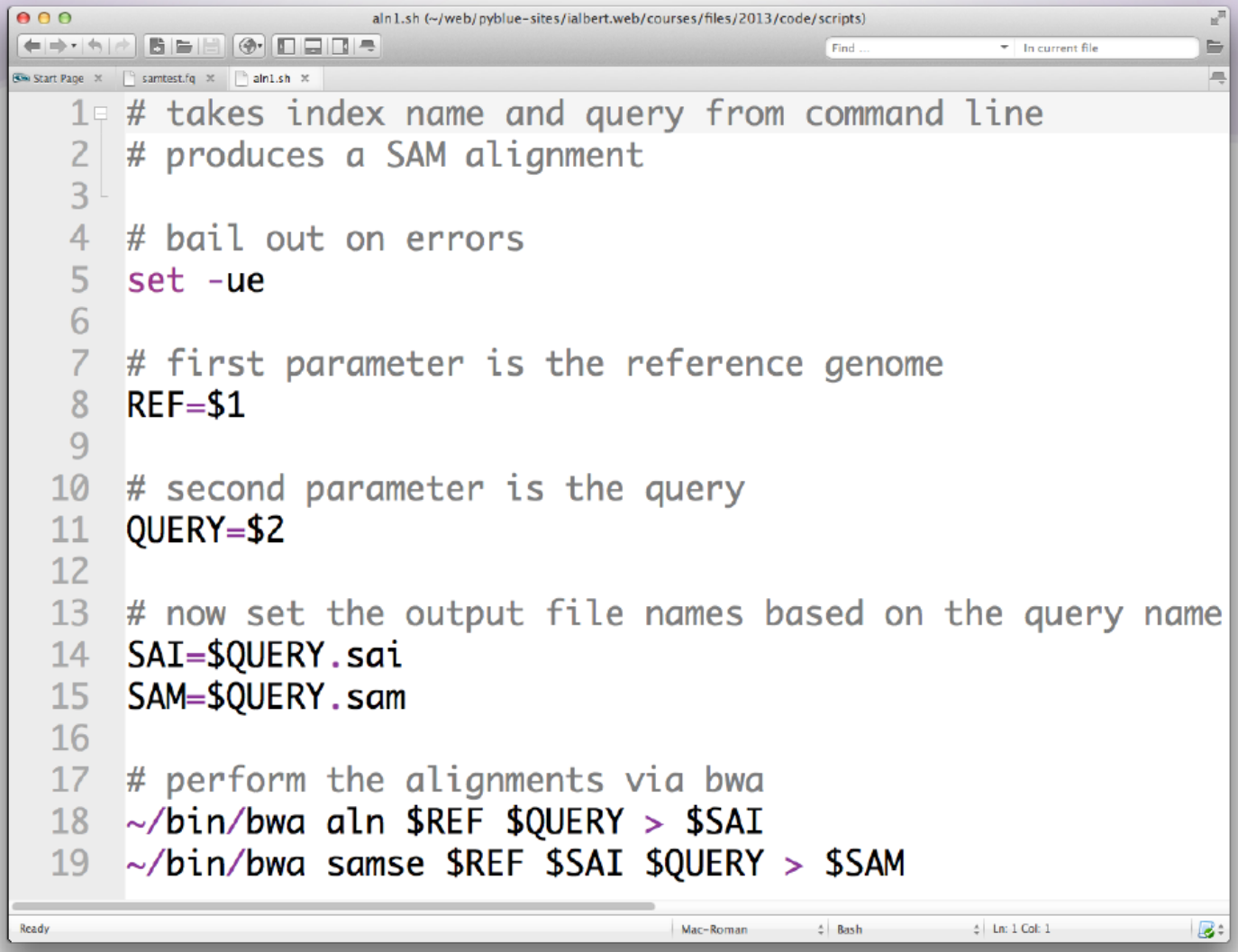
SAM format: Tabular text format

Published as

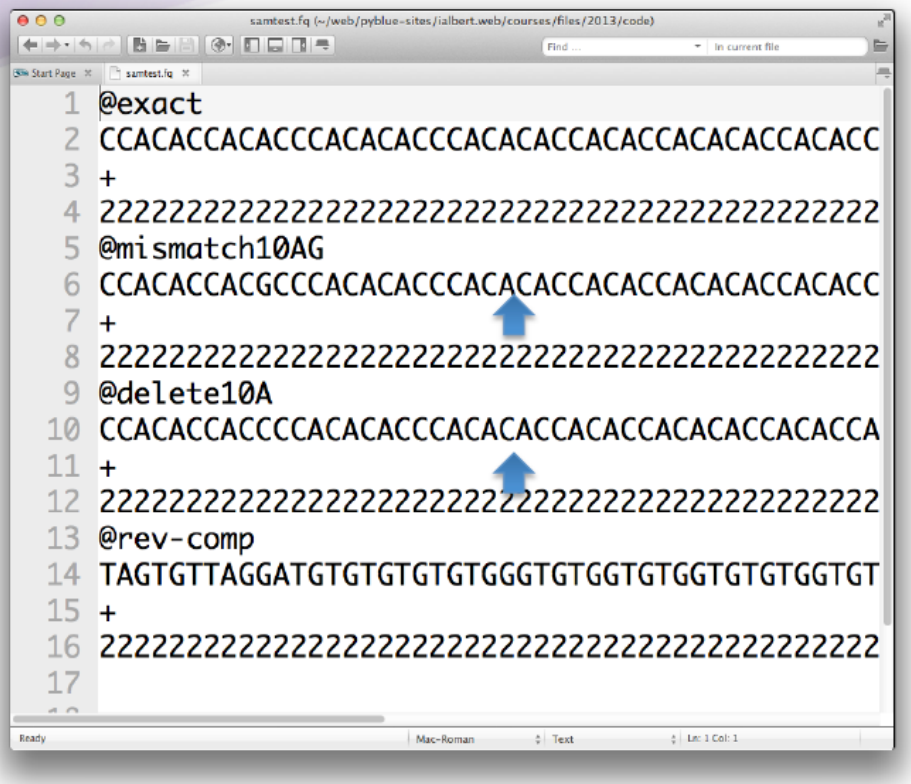
The Sequence Alignment/Map format and SAMtools by **Heng Li** et al Bioinformatics 25(16):2078-9, 2009

"A TAB delimited text format consisting of a header section, which is optional, and an alignment section."

Create a script that creates an alignment



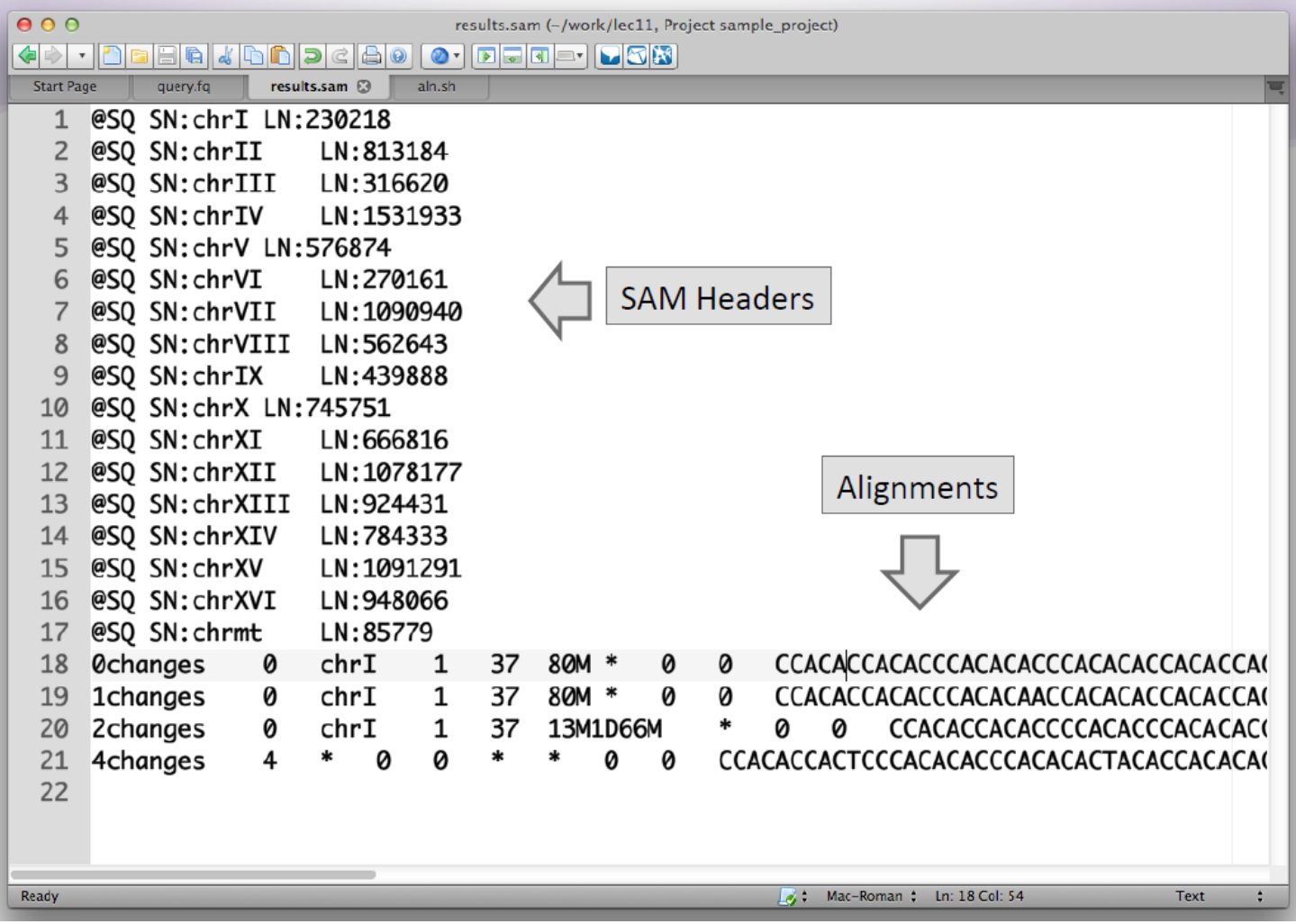
A test query file



The first line from the yeast genome – with a twist:

- 1st record is exact match
- 2nd record has a mismatch at position 10
- 3rd record has a deletion at position 10
- 4th record is the reverse complement of record 1

The structure of the SAM file



SAM format

- Sequence Alignment/Map format (SAM)
- Two sections
  - Header: All lines start with "@"; optional
    - @HD (VN*/SO/GO)
    - @SQ (SN*/LN*/AS…)
  - Alignments: All other lines

Resource: SAM specification
11 required column + optional fields

Sequence Alignment/Map Format Specification

The SAM/BAM Format Specification Working Group

3 Mar 2015

The master version of this document can be found at `https://github.com/samtools/hts-specs`.
This printing is version ec1fec2 from that repository, last modified on the date shown above.

1 The SAM Format Specification

SAM stands for Sequence Alignment/Map format. It is a TAB-delimited text format consisting of a header section, which is optional, and an alignment section. If present, the header must be prior to the alignments. Header lines start with '`@`', while alignment lines do not. Each alignment line has 11 mandatory fields for essential alignment information such as mapping position, and variable number of optional fields for flexible or aligner specific information.

An alignment consist of 11 tab delimited columns

1.4 The alignment section: mandatory fields

In the SAM format, each alignment line typically represents the linear alignment of a segment. Each line has 11 mandatory fields. These fields always appear in the same order and must be present, but their values can be '0' or '*' (depending on the field) if the corresponding information is unavailable. The following table gives an overview of the mandatory fields in the SAM format:

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

Column 1 and 2: QNAME and FLAG (Query name and bitwise flags)

1. QNAME: the name of query sequence

2. FLAG: bitwise FLAG. Each bit is explained in the following table:

Bit	Description
0x1	template having multiple segments in sequencing
0x2	each segment properly aligned according to the aligner
0x4	segment unmapped
0x8	next segment in the template unmapped
0x10	SEQ being reverse complemented
0x20	SEQ of the next segment in the template being reversed
0x40	the first segment in the template
0x80	the last segment in the template
0x100	secondary alignment
0x200	not passing quality controls
0x400	PCR or optical duplicate
0x800	supplementary alignment

Column 2: FLAG
the bitwise representation

1 = 00000001 → paired end read
2 = 00000010 → mapped as proper pair
4 = 00000100 → unmappable read
8 = 00001000 → read mate unmapped
16 = 00010000 → read mapped on reverse strand

The flag **11** → **1** + **2** + **8** = **0001011** (conditions 1, 2 and 8 satisfied)

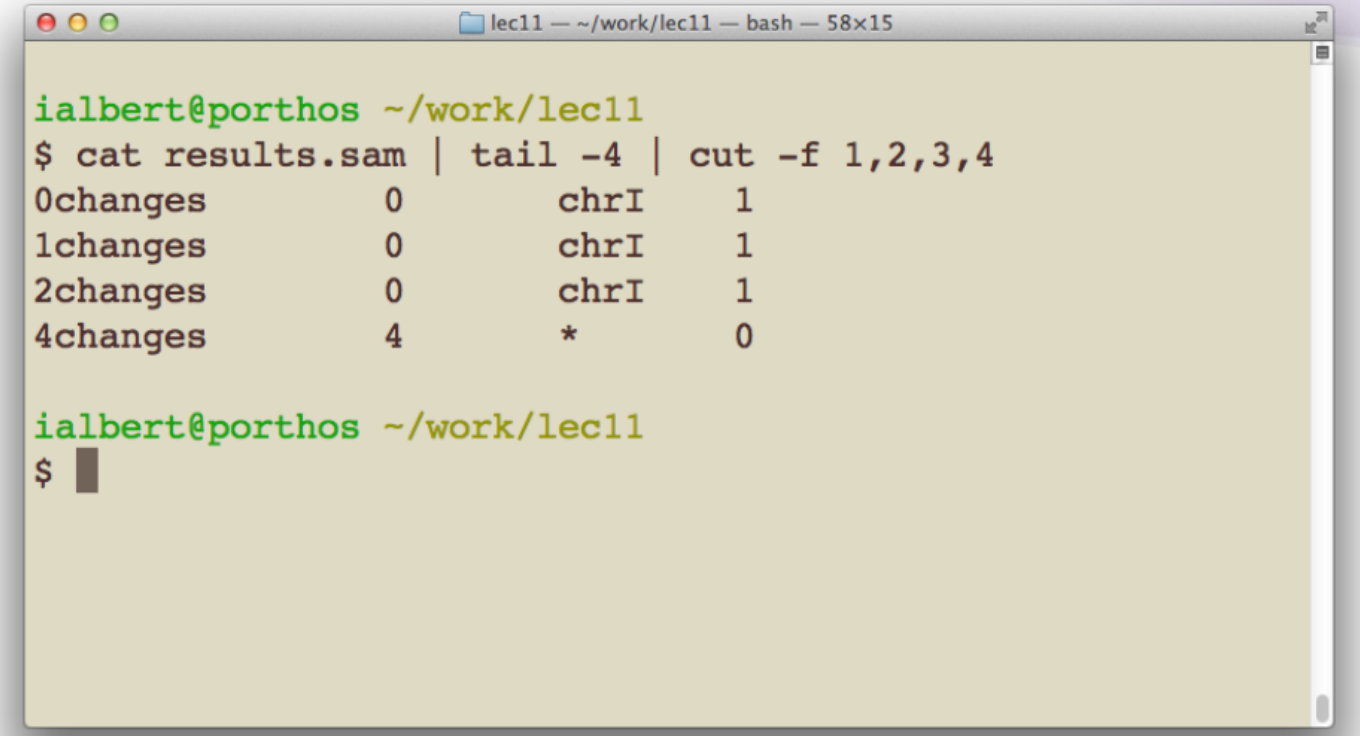
It is used to save space – but it does make things a bit more difficult.

Usually very few flags are needed in practice – 0, 4, 16 are the most generic ones

If you need to construct a more complex flag search for explain SAM flags:

http://picard.sourceforge.net/explain-flags.html

Columns 3, 4: RNAME and POS Reference and Position



Column 4 POS: **1-based leftmost mapping POSition of the first matching base**.

Very important to remember later when we need to find the 5' end (the actual start)

Column 5: MAPQ - Mapping Quality

- Phred score, identical to the quality measure in the fastq file. quality **Q**, probability **P**:

$$P = 10 \wedge (-Q / 10.0)$$

If **Q=30**, **P=1/1000** → on average, one of out 1000 alignments will be wrong

As good as this sounds it is not easy to compute such a quality.

$$Q=-10\log_{10}P$$

Details of the mapping quality computation – hard to find good answers

- Tool specific – there is no standard of what it should be
- The repeat structure of the reference. Reads falling in repetitive regions usually get very low mapping quality.
- The base quality of the read. Low quality means the observed read sequence is possibly wrong, and wrong sequence may lead to a wrong alignment.
- The sensitivity of the alignment algorithm. The true hit is more likely to be missed by an algorithm with low sensitivity, which also causes mapping errors.
- Paired end or not. Reads mapped in pairs are more likely to be correct.

(*from the MAQ manual*)

BWA specific high scores

A read alignment with a mapping quality 30 or above usually implies

- The overall base quality of the read is good.
- The best alignment has few mismatches.
- The read has few or just one `good' hit on the reference, which means the current alignment is still the best even if one or two bases are actually mutations or sequencing errors.

BWA specific low scores

Surprisingly difficult to track down the exact behavior

- Q=0 → if a read can be aligned equally well to multiple positions, BWA will randomly pick one position and give it a mapping quality zero.

- Q=25 → the edit distance equals mismatches and is greater than zero

Column 6: CIGAR

- CIGAR = Compact Idiosyncratic Gapped Alignment Report

6. CIGAR: CIGAR string. The CIGAR operations are given in the following table (set '*' if unavailable):

Op	BAM	Description
M	0	alignment match (can be a sequence match or mismatch)
I	1	insertion to the reference
D	2	deletion from the reference
N	3	skipped region from the reference
S	4	soft clipping (clipped sequences present in SEQ)
H	5	hard clipping (clipped sequences NOT present in SEQ)
P	6	padding (silent deletion from padded reference)
=	7	sequence match
X	8	sequence mismatch

- H can only be present as the first and/or last operation.
- S may only have H operations between them and the ends of the CIGAR string.
- For mRNA-to-genome alignment, an N operation represents an intron. For other types of alignments, the interpretation of N is not defined.

Columns 7, 8, 9: RNEXT, PNEXT, TLEN (used in paired end read sequencing)

- **RNEXT**: the name of the pair
- **PNEXT**: the position of the pair
- **TLEN**: the distance between the leftmost positions of the pairs

We will discuss these in more detail later.

These can show the position of reads that are distant – allow us to infer genomic variations

Column 10, 11

- SEQ: the query sequence
- QUAL: the phred encoded quality sequence

SEQ may contain the original sequence or the segment it was aligned to. Not all tools do the same thing.

Column 12 and beyond (optional)

1.5 The alignment section: optional fields

All optional fields follow the `TAG:TYPE:VALUE` format where `TAG` is a two-character string that matches `/[A-Za-z][A-Za-z0-9]/`. Each `TAG` can only appear once in one alignment line. A `TAG` containing lowercase letters are reserved for end users. In an optional field, `TYPE` is a single case-sensitive letter which defines the format of `VALUE`:

Type	Regexp matching `VALUE`	Description
A	`[!-~]`	Printable character
i	`[-+]?[0-9]+`	Signed integer[2]
f	`[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?`	Single-precision floating number
Z	`[ !-~]+`	Printable string, including space
H	`[0-9A-F]+`	Byte array in the Hex format[3]
B	`[cCsSiIf](,[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?)+`	Integer or numeric array

Samtools

SAM Tools provide various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format.

Binary SAM (BAM) files

SAM file:

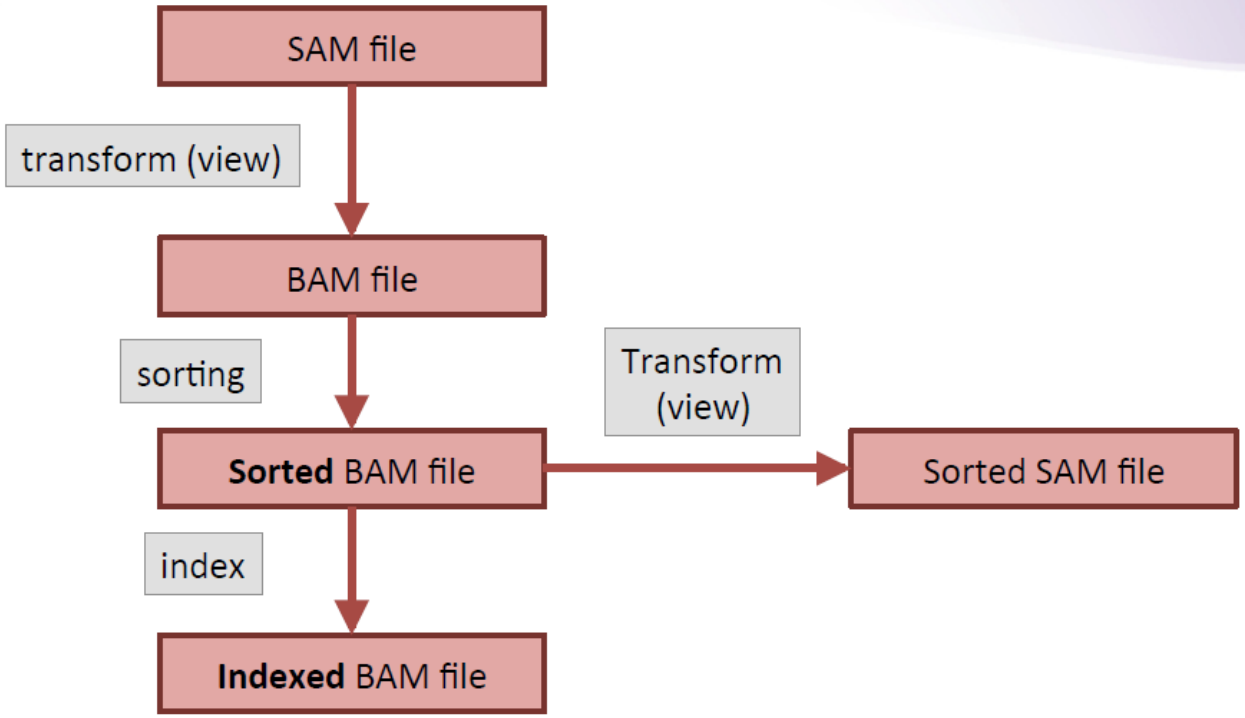
- information on the alignment of each read
- optimized for readability and sequential access

BAM (binary SAM):

- compression → saves space (optimized for size)
- may be **sorted** + **indexed** → location query (optimized for random access)
- the file is not readable by eye

Your default format should be BAM – only turn it into SAM when viewing the file

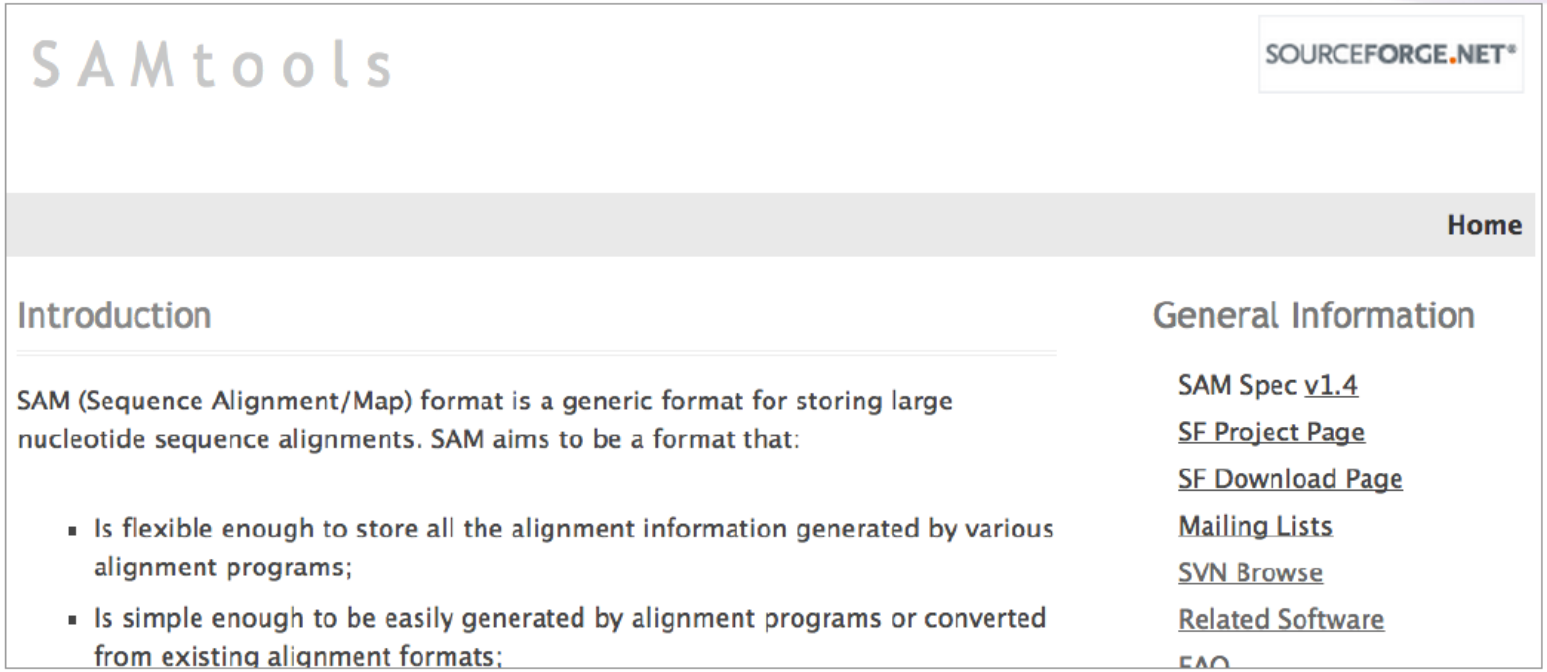
SAM/BAM hierarchy



Some tools have certain requirements of what type of SAM/BAM they take.

Your default data format should be a **sorted, indexed BAM** file!

Download and 'make' SAMTOOLS



http://samtools.sourceforge.net/

Samtools: is suite of commands

```
Usage:   samtools <command> [options]

Command: view         SAM<->BAM conversion
         sort         sort alignment file
         mpileup      multi-way pileup
         depth        compute the depth
         faidx        index/extract FASTA
         tview        text alignment viewer
         index        index alignment
         idxstats     BAM index stats (r595 or later)
         fixmate      fix mate information
         flagstat     simple stats
         calmd        recalculate MD/NM tags and '=' bases
         merge        merge sorted alignments
         rmdup        remove PCR duplicates
         reheader     replace BAM header
         cat          concatenate BAMs
         bedcov       read depth per BED region
         targetcut    cut fosmid regions (for fosmid pool only)
         phase        phase heterozygotes
         bamshuf      shuffle and group alignments by name
```

Most actions will provide help on their usage

```
$ samtools view

Usage:   samtools view [options] <in.bam>|<in.sam> [region1 [...]]

Options: -b       output BAM
         -h       print header for the SAM output
         -H       print header only (no alignments)
         -S       input is SAM
         -u       uncompressed BAM output (force -b)
         -1       fast compression (force -b)
         -x       output FLAG in HEX (samtools-C specific)
         -X       output FLAG in string (samtools-C specific)
         -c       print only the count of matching records
         -B       collapse the backward CIGAR operation
         -@ INT   number of BAM compression threads [0]
         -L FILE  output alignments overlapping the input BED FILE [null]
         -t FILE  list of reference names and lengths (force -S) [null]
         -T FILE  reference sequence file (force -S) [null]
         -o FILE  output file name [stdout]
         -R FILE  list of read groups to be outputted [null]
         -f INT   required flag, 0 for unset [0]
```

Default Operation

- By default **samtools** expects a **BAM** file as input and will produce a **SAM** file as output

- Every alignment result should be stored as a **sorted** and **indexed** BAM file

Transform SAM to BAM

transform to bam

```
samtools view -Sb input.sam > tempfile.bam
```

sort bam file

```
samtools sort -f tempfile.bam output.bam
```

Index bam file

```
samtools index output.bam
```

Filtering SAM/BAM files

Required flag (keep if matches)

```
samtools view -f
```

Filtering flag (remove if matches)

```
samtools view -F
```

Flags are using a bitwise representation

1 = 00000001 → paired end read
2 = 00000010 → mapped as proper pair
4 = 00000100 → unmappable read
8 = 00001000 → read mate unmapped
16 = 00010000 → read mapped on reverse strand

```
ialbert@porthos ~/work/lec12
$ ~/bin/samtools view -c -f 4 results.bam
1

ialbert@porthos ~/work/lec12
$ ~/bin/samtools view -c -F 4 results.bam
3
```

-c means to count the lines
-f <number> - keep reads that match
-F <number> - remove reads that match

```
# save on typing
alias samtools=~/bin/samtools

# how many reads in total
samtools view -c results.bam

# reads that cannot be mapped
samtools view -c -f 4 results.bam

# reads that can be mapped
samtools view -c -F 4 results.bam

# reads that map to reverse strand
samtools view -c -f 16 results.bam

# reads that map to forward strand
samtools view -c -F 16 results.bam

# reads that have a minimum mapping quality of 1
# note that for BWA this also means unique alignment!
samtools view -c -q 1 results.bam
```

Explore other commands

Flag statistics

```
samtools flagstat data.bam
```

Index stats

```
samtools idxstats data.bam
```

Depth of coverage

```
samtools depth data.bam | head
```

Querying a BAM file **name:start-end**

Samtools allows querying:

```
samtools view data.bam  chrV:1000-2000
```

Thank you for your attention