SAM, pileup and related formats

Sequence Alignment/ Map (SAM)
Binary SAM (BAM)
pileup
CRAM
Compact Idiosyncratic Gapped Alignment Report (CIGAR)

This presentation

- *- is intended to :*
  - *give a fairly detailed description of the types of sequence alignment file formats used for "maps" of short-reads aligned to longer reference sequences*
  - *builds on the introduction to those formats in the "Sequence Alignments (1)" presentation*
  - *provides more in-depth background for those attempting to do the SAMtools tutorial*
- *- but it is not essential for that tutorial*

- These formats are most useful for storing information like this:
  - i.e. the location of many **sequence reads** which have been 'mapped' to a longer reference sequence

Mapping data files

- The mapping procedure which produced this information will have been performed by one of many different software packages
- The SAM, pileup etc. data file doesn't care how it was done
- Or how good the mapping data is
  - E.g. how reliable each mapped read location is
  - (but it can store scores produced by the mapping program)

Mapped-read file formats

- SAM format (and its compressed, "binary" form, BAM) is used commonly
- Both can be easily manipulated by dedicated software (SAMTOOLS package)
- Various other software can read SAM/BAM files
  - E.g. visual browsers, such as:
  - the Integrated Genomics Viewer (Robinson *et al.*, 2011; Thorvaldsdóttir *et al.*, 2013)
  - The Integrated Genome Browser (Freese *et al.*, 2016)
- Conversion to other formats such as pileup, BCF – done with SAMTOOLs

- VCF/BCF are not covered here
  - VCF/BCF effectively summarise a mapping exercise in terms of identified **variants**, such as SNPs (as opposed to any different base calls such as errors)

pileup format

- **pileup format*** specifies differences between each read and the sequence of that part of the reference to which it is mapped
- Also specifies quality of mapping, and quality of read
- Useful for visually identifying SNPs etc
- Does not save space, as each base of each read is explicitly specified
- Relatively easy to understand and visualise

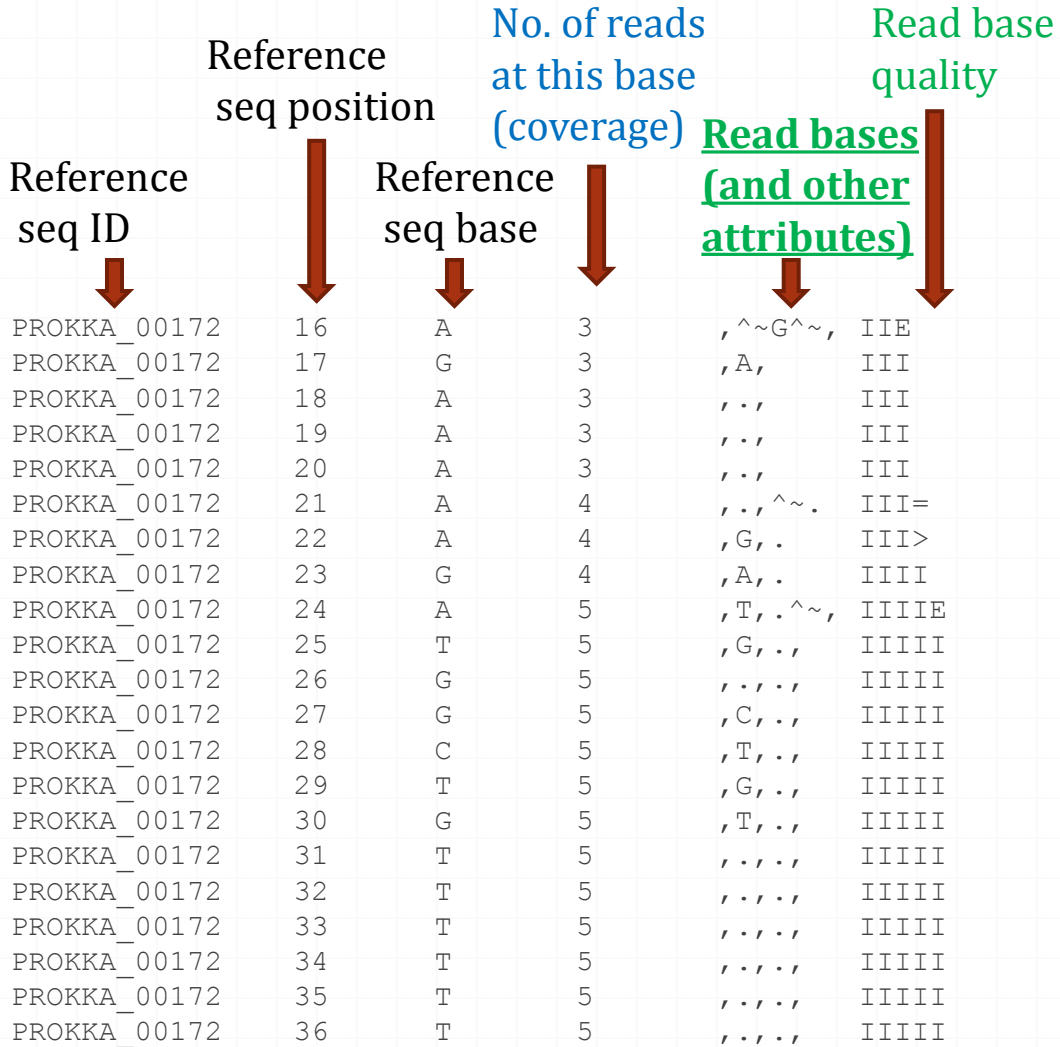
- * *For those of a certain age: 'pileup' is not to be confused with the (very old) alignment program of the same name or its output (in GCG package); this in fact was the origin of the MSF alignment format (which is still widely recognised)*

Example pileup

Reference seq ID	Reference seq position	Reference seq base	No. of reads at this base (coverage)	Read bases (and other attributes)	Read base quality
PROKKA_00172	16	A	3	,^~G^~,	IIE
PROKKA_00172	17	G	3	,A,	III
PROKKA_00172	18	A	3	,.,	III
PROKKA_00172	19	A	3	,.,	III
PROKKA_00172	20	A	3	,.,	III
PROKKA_00172	21	A	4	,.,^~.	III=
PROKKA_00172	22	A	4	,G,.	III>
PROKKA_00172	23	G	4	,A,.	IIII
PROKKA_00172	24	A	5	,T,.^~,	IIIIE
PROKKA_00172	25	T	5	,G,.,	IIIII
PROKKA_00172	26	G	5	,.,.,	IIIII
PROKKA_00172	27	G	5	,C,.,	IIIII
PROKKA_00172	28	C	5	,T,.,	IIIII

Introduction to sequence quality data

Example pileup



Read bases (and other attributes)

Symbols	meaning
.	matches the reference base
,	matches the reverse complement of the reference base
ACGTN	a base call which does not match
^***qX***	*X* (see above) is the first base of this read; *q* represents the quality of the **mapping location of the whole read**
$***X***	*X* (see above) is the last base of this read
+***n***	The next *n* symbols represent inserted bases in that read
-***n***	The next *n* symbols represent deleted bases from that read

pileup in more detail

```
PROKKA_00172    16      A       3       ,^~G^~,  IIE
PROKKA_00172    17      G       3       ,A,      III
PROKKA_00172    18      A       3       ,.,      III
PROKKA_00172    19      A       3       ,.,      III
PROKKA_00172    20      A       3       ,.,      III
PROKKA_00172    21      A       4       ,.,^~.   III=
PROKKA_00172    22      A       4       ,G,.     III>
PROKKA_00172    23      G       4       ,A,.     IIII
PROKKA_00172    24      A       5       ,T,.^~,  IIIIE
PROKKA_00172    25      T       5       ,G,.,    IIIII
PROKKA_00172    26      G       5       ,.,.,    IIIII
...
...
PROKKA_00172    113     A       23      ,$.,,,,.....,,,,,^~.^~.    IIIIIIIIIIIIIIIIIIIBB
PROKKA_00172    114     A       22      ..,.,,.....,,,,,,..  IIIIIIIIIIIIIIIIIIEE
```

- In essence, reads are represented by vertical columns.
- Three reads are highlighted.
- The 2nd and 3rd reads (yellow and green) both begin at position 16.
- A 4th begins at 21, and a 5th at 24
- By the time position 113 is reached, there are 23 reads covering that position.
- The 2nd (yellow) read ends at 113.
- The 1st and 3rd reads match the reverse complement
- The 2nd read matches the forward sequence (with a few mismatching bases)

SAM data files

- The SAM data is concerned primarily with:
  - Where each read is mapped
  - Does it 'map perfectly' i.e. can be aligned without interruptions
    - i.e. without insertions or deletions ("indels")
  - If not, then where are the indels
  - Do the actual sequences match at each base position? Where do they differ?

SAM format

- A SAM format file consists of one line per read
- Each line has many data fields
- Some of these can have null values (usually '*')
  - - It depends on the application
- One of the fields consists of a series of symbols (i.e. a **string**) in a code called **CIGAR**
- The CIGAR string specifies properties of each alignment (mapping of 1 read)

CIGAR format

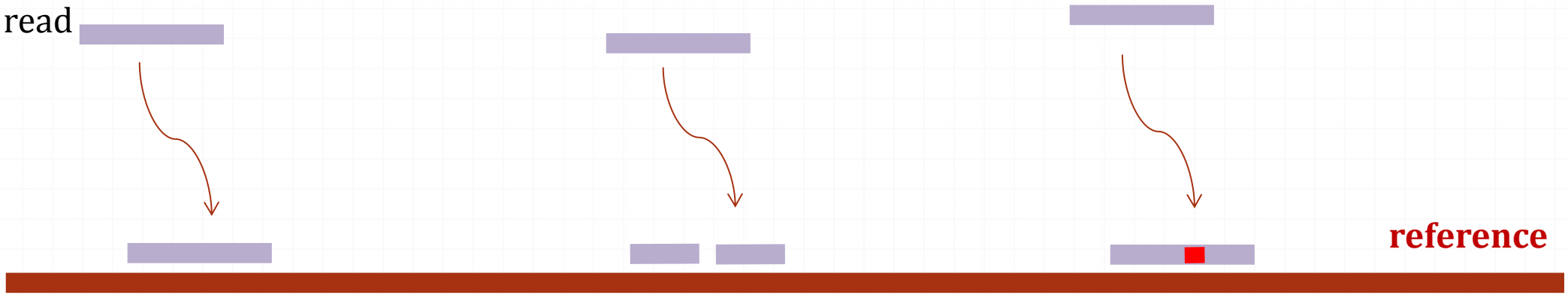
- **CIGAR** format ("Compact Idiosyncratic Gapped Alignment Report")
  - (and it *is* idiosyncratic)
- Strictly speaking, this specifies the ***mapping*** characteristics, not the full details of the ***sequence*** alignment
- What does that mean?
- It means that the **location** of **all the bases** of the sequence read, relative to the reference, are specified; which is the same as:
  - Where the read **begins**, and where it **ends**
  - And where any **insertions** and **deletions** are
  - ***Mismatched bases are not explicitly stored in the CIGAR string***

CIGAR format

- One CIGAR 'string' per alignment (read ↔ reference):
  - Example: '**6=1X7=2X6=**'
- Original CIGAR specification was designed to specify for each alignment:
  - **How long the alignment is**
  - **Where the insertions and deletions (indels) are**
- Later additions to CIGAR also permitted specification of:
  - ***Where*** **mismatching bases occur**
  - But ***NOT*** what those mismatches actually are
  - -it was simply not the original purpose of the tools which used CIGAR

- Example alignment 1:
  - Reference:…**TACGGAAGGTCCGGGCGTTATC**…
  - Read: **TACGGAAGGTCCGGGCGTTATC**
  - CIGAR string: **22M** (means '22 alignment matches')

- Example alignment 2:
  - Reference:…**TACGGAAGGTCCGGGCGTTATC**…
  - Read: **TACGGAGGGTCCGGTTGTTATC**
  - CIGAR string (*early* CIGAR spec): **22M** (because there are no indels)
  - CIGAR string (later CIGAR spec): **6=1X7=2X6=**
  - There are ***still*** '22 alignment matches' but '=' means a **sequence match** and 'X' means a **sequence mismatch**



Perfect mapping
Perfect sequence match
(all bases the same
as reference)

Alignment can be fully specified with CIGAR string (+ ref. sequence + start position)

Imperfect mapping
(there is a deletion
in the read – a base
in the reference is
absent from the read)
All aligned bases match
perfectly

Alignment can be fully specified with CIGAR string (+ ref. sequence + start position)

"Perfect" mapping
Imperfect sequence match
(one base mismatches)

Alignment cannot be fully specified with only CIGAR string (+ ref. sequence + start position) -additional fields are required

SAM format

- Each read is represented on one line (1 line = 1 'record')
- Each line has a field specifying the position on the reference sequence
- And a field in CIGAR format
- Then it gets complicated…
- ***Also additional optional fields*** – these can be used to state:
  - explicit base differences
  - (and many other things)
- SAM format **can** explicitly state the sequence of each read
  - And often does (i.e. no space saved)
  - But does **not** have to (saves space)

SAM format

- So if you want to do **additional** things like:
- Given the **read sequence** and the CIGAR string
  - infer the aligned segment of the **reference**
- Or given the **reference sequence** and the CIGAR string
  - infer the sequence of the **read**
- Then it depends on:
  - If a non-ancient version of CIGAR is used
  - Which optional fields are present in the SAM record
  - Whether there are any base mismatches in the first place
  - In some cases, it may simply be necessary to state the read sequence explicitly

Evolution of bioinformatics formats

- This is an example of bioinformatics format specifications which have become revised over the years
- The original design may have been to achieve something very specific
- Additional functionality has been enabled by additional information included in the data files
- New fields “bolted on” to a simpler, earlier spec
- Complete revisions are generally uncommon because of the need to retain backward compatibility

Some context

- Example: You want to know **how many reads** are aligned to each part of the genome

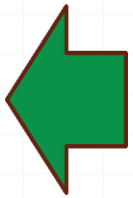
Some context

- Example: You want to know **how many reads** are aligned to each part of the genome
- E.g. transcriptomics (RNAseq) – thus, gene expression levels
  - Which genes are most represented?
  - You don't care if there are a small number of mismatches due to sequence errors
  - You don't even care what the sequences of the alignments in each region actually ***are***
  - **All you want to know is where the alignment/mapping program (whatever that may be) mapped the reads**
  - **Only minimal CIGAR/SAM information is required for this**
  - (N.B. SAM format can optionally store scores produced by the alignment program)

Example SAM data

```
@HD VN:1.3 SO:coordinate
@SQ SN:ref LN:45
r001 163 ref 7 30 8M2I4M1D3M  = 37 39 *    *
r002 0   ref 9 30 3S6M1P1I4M  *  0  0 *    *
r003 0   ref 9 30 5H6M        *  0  0 *    * NM:i:1
```

Header lines



Data lines (records) : 1 per read

Read ID

Mapping position on reference sequence

CIGAR string

Read sequence (absent in this case)

Optional field

Other fields above specify information on:

- The quality of the bases (unspecified in this case)
- The quality of the mapping location
- The reference sequence ID
- Mate pair reads
- Numerous other attributes

```
@HD VN:1.3 SO:coordinate
@SQ SN:ref LN:45
r001 163  ref 7 30 8M2I4M1D3M = 37 39   TTAGATAAAGGATACTG   *
r002 0    ref 9 30 3S6M1P1I4M * 0 0     AAAAGATAAGGATA      *
r003 0    ref 9 30 5H6M       * 0 0     AGCTAA              * NM:i:1
```

- Similar data, but with explicit read sequences

This field stores the **quality scores** for each base
- so are blank in this example

Introduction to sequence quality data

SAMtools

- SAMtools tools:
  - `tview` – display alignments in a text-based viewer
  - `view` – extract/display some (or all) sub alignments (e.g. for one region)
    - also converts between formats
  - `sort` – sort reads by location on reference (or by other criteria)
  - `index` – create indices which enable fast access to any part of the data
  - `mpileup` – convert to pileup or BCF format, eg. for SNP calling
  - `merge` – merge multiple sorted alignments
  - `rmdup` – remove potential PCR duplicates
  - Various other utilities
- For more info: http://samtools.sourceforge.net/samtools.shtml

BAM format

- BAM = Binary SAM
  - "binary" meaning not a plain-text file
  - A compressed form of SAM
- Can be interconverted with SAMTOOLS
  - E.g.: **`samtools view ……`**
  - **`samtools view -S myfile1.sam -o myfile1.bam`**
  - **`Samtools view myfile2.bam -o myfile2.sam`**
- Some operations require the files to have been **indexed** (refer to samtools manual/tutorial)

Other formats

- SAM/BAM are widely used and there have been several methods devised to still more efficiently compress these, e.g.
  - SAMZIP (Sakib *et al.*, 2011)
  - Goby (Campagne *et al.*, 2013) – latest version provides direct support for BAM and CRAM formats
  - Samcomp (Bonfield & Mahoney, 2013)

References (1)

- SAM/BAM/pileup/SAMtools
  - Li *et al.* Bioinformatics (2009) **25** (16): 2078-2079
  - Canonical specification http://samtools.github.io/hts-specs/SAMv1.pdf
  - Standard optional fields http://samtools.github.io/hts-specs/SAMtags.pdf
  - SAM/BAM v1.5 extensions http://biorxiv.org/content/biorxiv/early/2015/05/29/020024.full.pdf
- SAMZIP: Sakib *et al.* (2011) PLoS ONE **6** (12): e28251
- Goby: Campagne *et al.* (2013) PLoS ONE **8** (11): e79871
- CRAM
  - Fritz *et al.* (2011) Genome Res. **21** (5): 734-740
  - Specification http://samtools.github.io/hts-specs/CRAMv3.pdf
  - Tutorial http://www.ebi.ac.uk/ena/support/cram-tutorial

References (2)

- Samcomp and FASTQ-compressors (Fqzcomp, Fastqz)
  - Bonfield & Mahoney (2013) PLoS ONE **8** (3): e59190
- Integrative Genomics Viewer (IGV)
  - Robinson *et al.* (2011) Nature Biotechnology **29**: 24-26
  - Thorvaldsdóttir *et al.* (2013) Briefings in Bioinformatics **14** (2): 178-192
  - http://software.broadinstitute.org/software/igv/
- Integrated Genome Browser (IGB)
  - Freese *et al.* (2016) Bioinformatics **32** (14): 2089-2095
  - http://bioviz.org/igb/