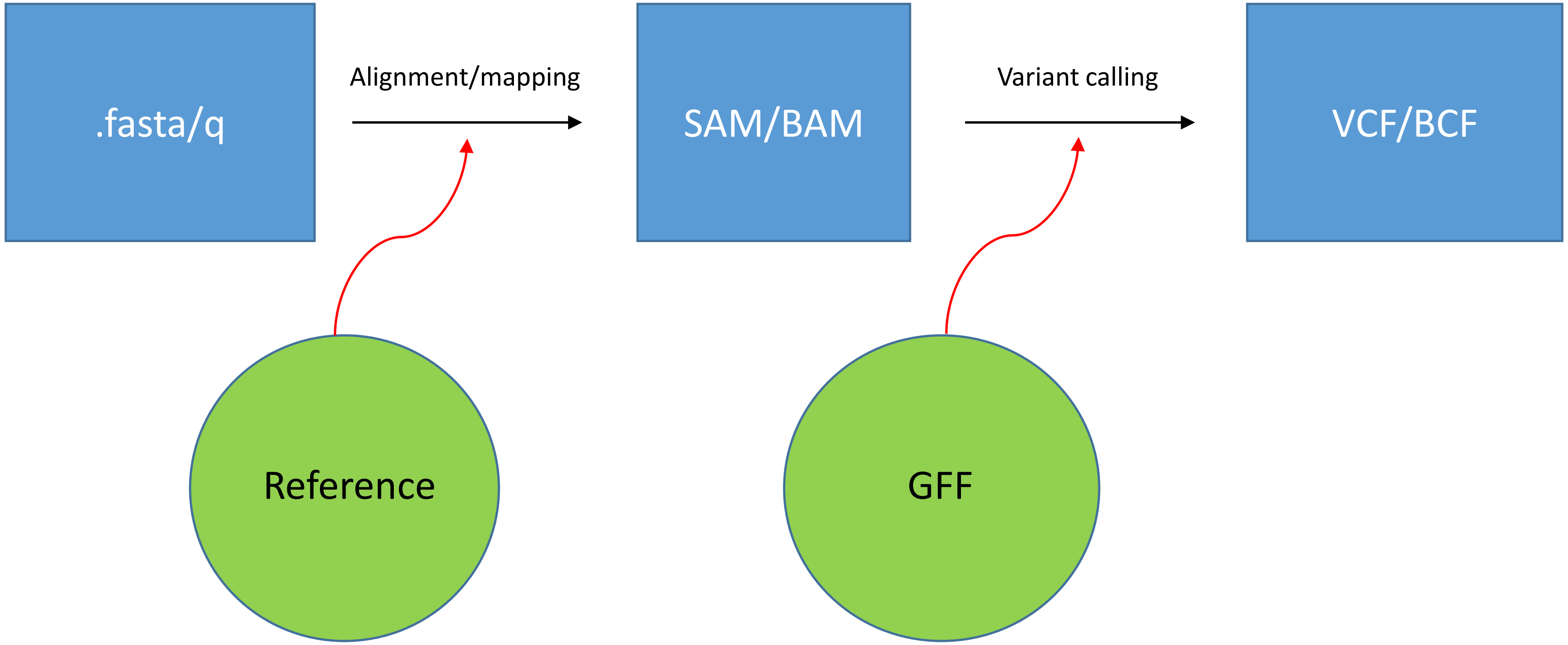
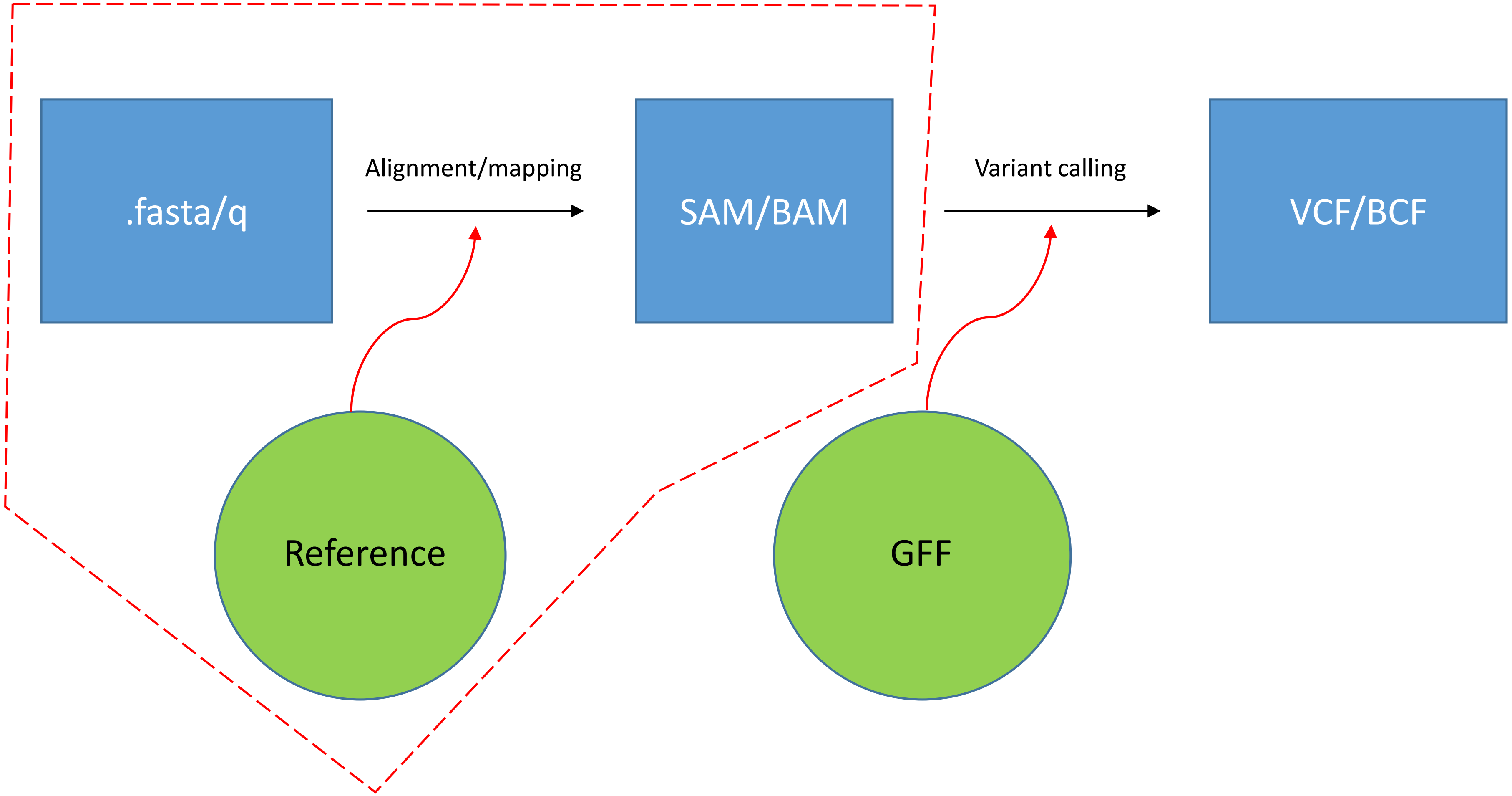
From BAM to BCF & beyond

Dr. Paolo Vatta M.S.S.E.
Foodborne & Neglected Parasites group
Dept. of Infectious Diseases
Istituto Superiore di Sanità

Overview


.fasta/q
Alignment/mapping
SAM/BAM
Variant calling
VCF/BCF
Reference
GFF

Overview


.fasta/q
Alignment/mapping
SAM/BAM
Variant calling
VCF/BCF
Reference
GFF

BAM (SAM)

- Sequence Alignment/Mapping (SAM)
- The output of any type of alignment to a reference* done by programs such as BWA or BOWTIE, their mapping is fundamentally different than BLAST alignment and is based on the «FM-Index» based on the Burroughs-Wheeler Transform (no math I promise!)

  1 sample of 15 million reads, BLAST 0.1 secs x read = 17 days!!!

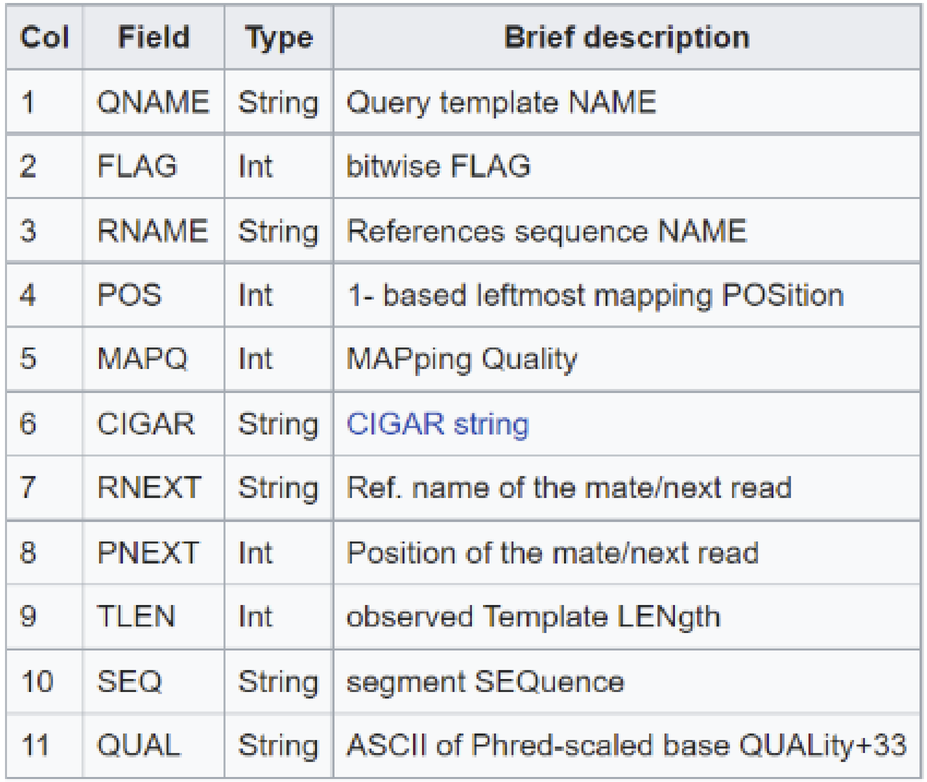
- SAM is the standardized output obtained from the alignment of fasta and fastq sequences (the «reads») to a reference
- It is the data that we can then use to understand anything we wish from our NGS experiment

* I specifically do not use the words «genomic reference»

BAM (SAM)

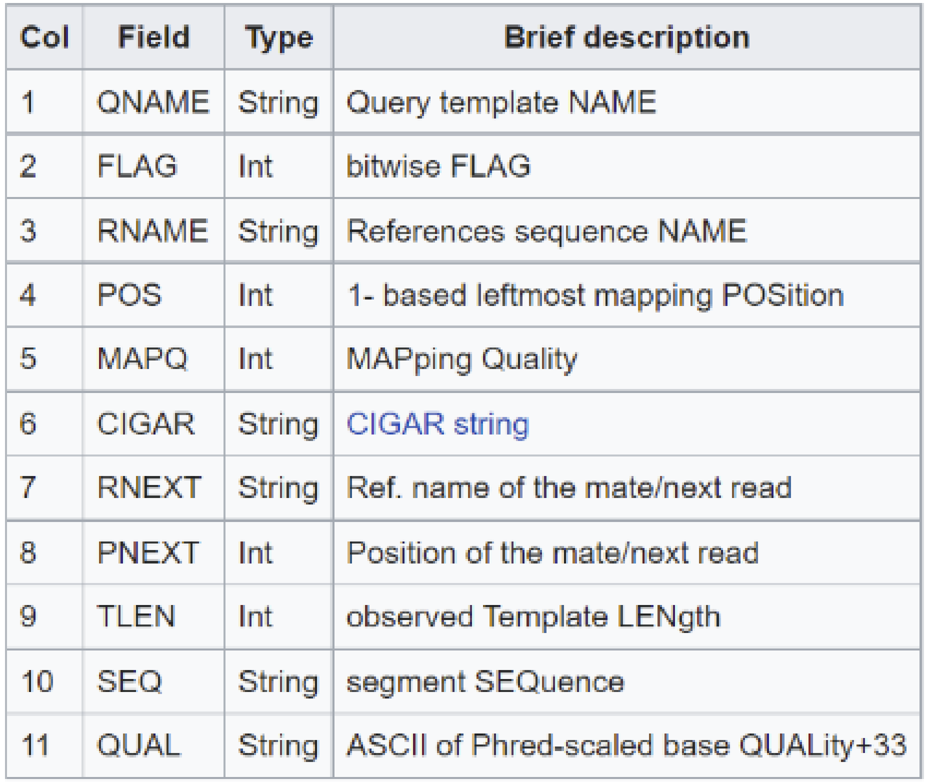
- Feeding our fasta/q files and the reference to the aligners will produce results in a .sam file that contains, in text form, the standardized results of each fragment mapping known as SAM format
- SAM is the text form of standardized output obtained from the alignment of fasta and fastq sequences (the «reads») to a reference. It will contain one row of data for each fragment. Since it has much more information than the .fasta/q file. It is huge! Often in the range of 10s to 100s of Gb
- To reduce the footprint of this file SAMs are usually transformed into BAM (Binary Alignment/Mapping) files

What is the BAM (SAM) format

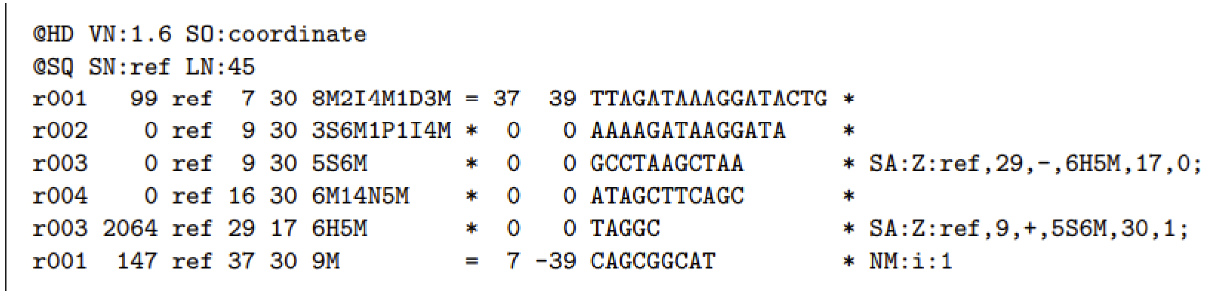


Col	Field	Type	Brief description
1	QNAME	String	Query template NAME
2	FLAG	Int	bitwise FLAG
3	RNAME	String	References sequence NAME
4	POS	Int	1- based leftmost mapping POSition
5	MAPQ	Int	MAPping Quality
6	CIGAR	String	CIGAR string
7	RNEXT	String	Ref. name of the mate/next read
8	PNEXT	Int	Position of the mate/next read
9	TLEN	Int	observed Template LENgth
10	SEQ	String	segment SEQuence
11	QUAL	String	ASCII of Phred-scaled base QUALity+33

What is the BAM (SAM) format

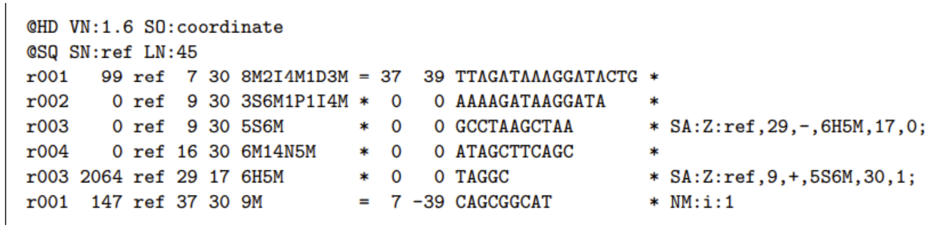


Col	Field	Type	Brief description
1	QNAME	String	Query template NAME
2	FLAG	Int	bitwise FLAG
3	RNAME	String	References sequence NAME
4	POS	Int	1- based leftmost mapping POSition
5	MAPQ	Int	MAPping Quality
6	CIGAR	String	CIGAR string
7	RNEXT	String	Ref. name of the mate/next read
8	PNEXT	Int	Position of the mate/next read
9	TLEN	Int	observed Template LENgth
10	SEQ	String	segment SEQuence
11	QUAL	String	ASCII of Phred-scaled base QUALity+33

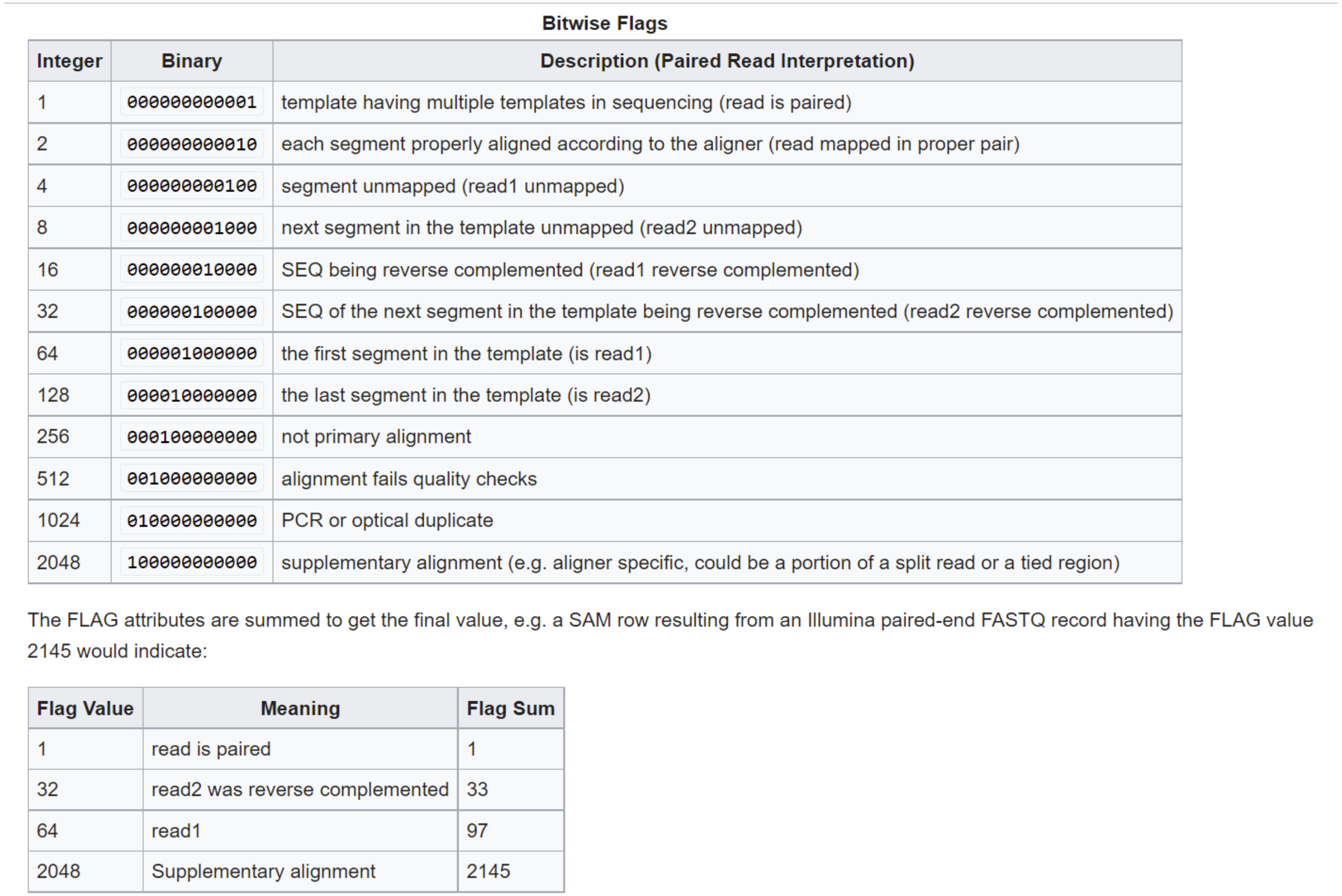


```
@HD VN:1.6 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
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
```

SAM FLAG field



```
@HD VN:1.6 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
```



Bitwise Flags

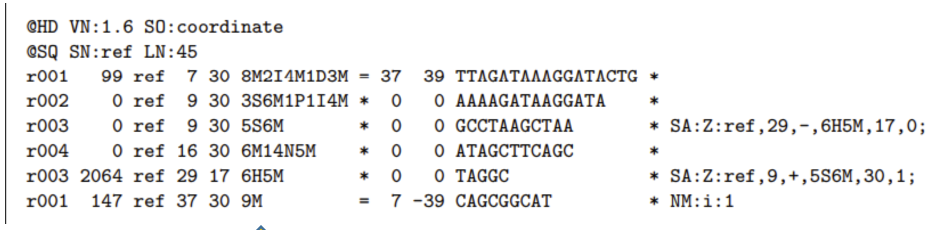
Integer	Binary	Description (Paired Read Interpretation)
1	`000000000001`	template having multiple templates in sequencing (read is paired)
2	`000000000010`	each segment properly aligned according to the aligner (read mapped in proper pair)
4	`000000000100`	segment unmapped (read1 unmapped)
8	`000000001000`	next segment in the template unmapped (read2 unmapped)
16	`000000010000`	SEQ being reverse complemented (read1 reverse complemented)
32	`000000100000`	SEQ of the next segment in the template being reverse complemented (read2 reverse complemented)
64	`000001000000`	the first segment in the template (is read1)
128	`000010000000`	the last segment in the template (is read2)
256	`000100000000`	not primary alignment
512	`001000000000`	alignment fails quality checks
1024	`010000000000`	PCR or optical duplicate
2048	`100000000000`	supplementary alignment (e.g. aligner specific, could be a portion of a split read or a tied region)

The FLAG attributes are summed to get the final value, e.g. a SAM row resulting from an Illumina paired-end FASTQ record having the FLAG value 2145 would indicate:

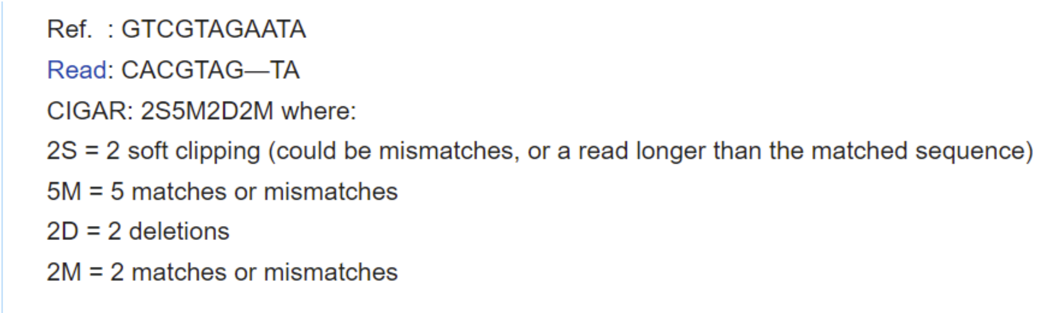
Flag Value	Meaning	Flag Sum
1	read is paired	1
32	read2 was reverse complemented	33
64	read1	97
2048	Supplementary alignment	2145

CIGAR string

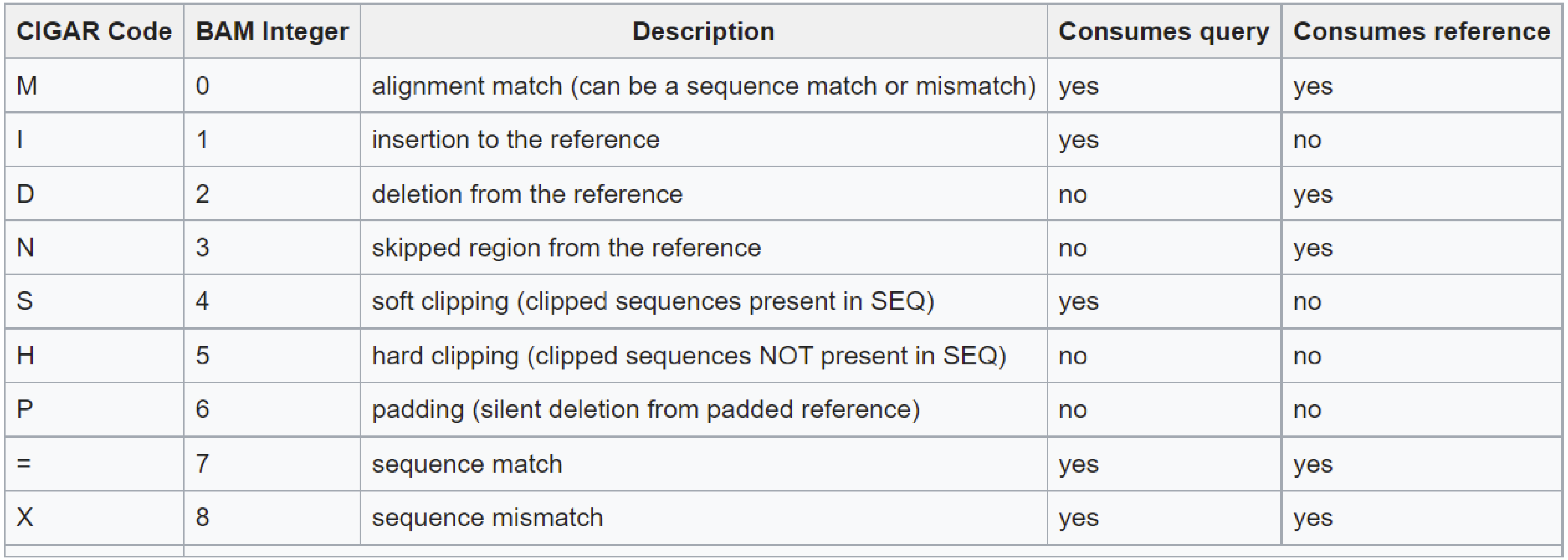
CIGAR (Compact Idiosyncratic Gapped Alignment Report) string



```
@HD VN:1.6 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
```



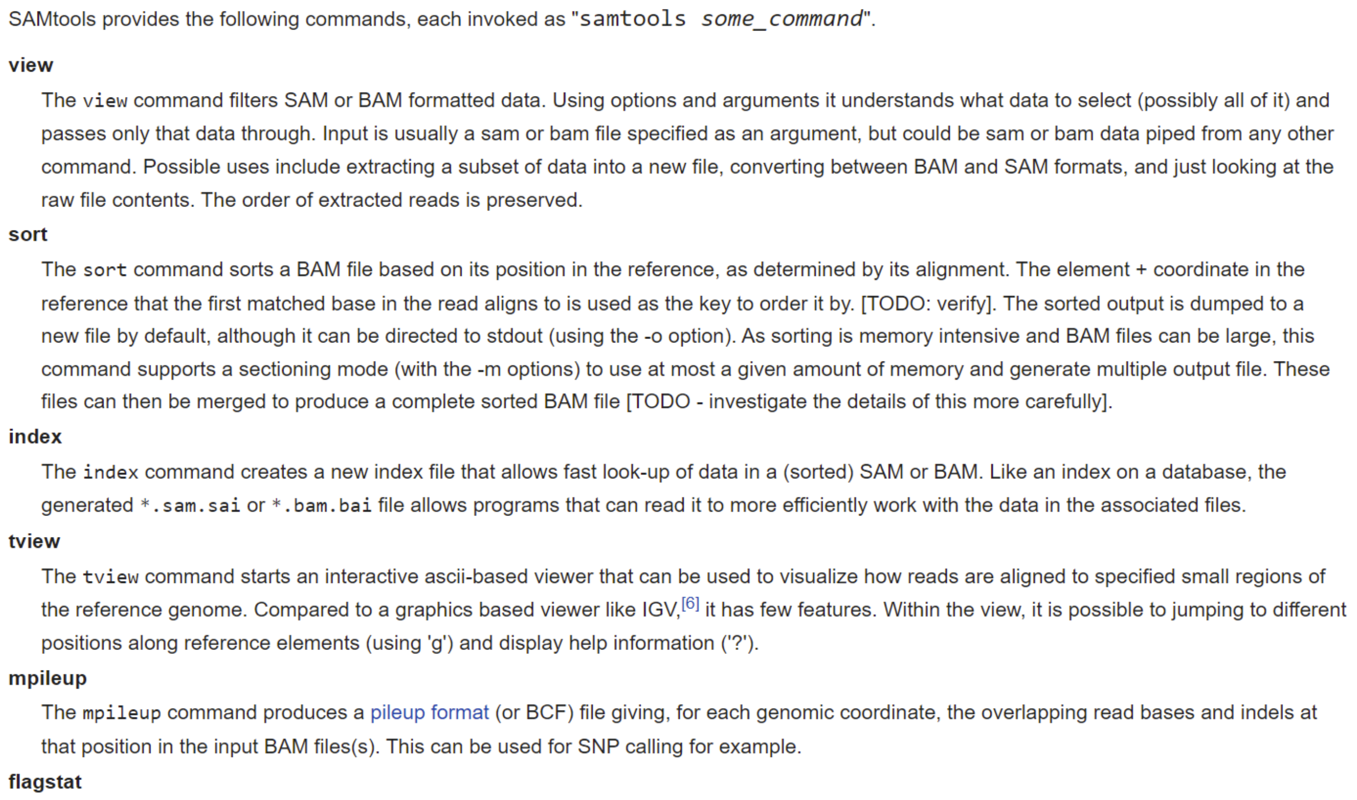
Ref. : GTCGTAGAATA
Read: CACGTAG—TA
CIGAR: 2S5M2D2M where:
2S = 2 soft clipping (could be mismatches, or a read longer than the matched sequence)
5M = 5 matches or mismatches
2D = 2 deletions
2M = 2 matches or mismatches



CIGAR Code	BAM Integer	Description	Consumes query	Consumes reference
M	0	alignment match (can be a sequence match or mismatch)	yes	yes
I	1	insertion to the reference	yes	no
D	2	deletion from the reference	no	yes
N	3	skipped region from the reference	no	yes
S	4	soft clipping (clipped sequences present in SEQ)	yes	no
H	5	hard clipping (clipped sequences NOT present in SEQ)	no	no
P	6	padding (silent deletion from padded reference)	no	no
=	7	sequence match	yes	yes
X	8	sequence mismatch	yes	yes

SAMtools - BAMtools

- Suite of programs («commands») used to interrogate, extract, manipulate, sort, find, munge, classify, ask questions of data contained in SAM/BAM files
- Can be used as command line programs, or often are included in GUI based container programs such as Galaxy



SAMtools provides the following commands, each invoked as "`samtools` *`some_command`*".

view

The `view` command filters SAM or BAM formatted data. Using options and arguments it understands what data to select (possibly all of it) and passes only that data through. Input is usually a sam or bam file specified as an argument, but could be sam or bam data piped from any other command. Possible uses include extracting a subset of data into a new file, converting between BAM and SAM formats, and just looking at the raw file contents. The order of extracted reads is preserved.

sort

The `sort` command sorts a BAM file based on its position in the reference, as determined by its alignment. The element + coordinate in the reference that the first matched base in the read aligns to is used as the key to order it by. [TODO: verify]. The sorted output is dumped to a new file by default, although it can be directed to stdout (using the -o option). As sorting is memory intensive and BAM files can be large, this command supports a sectioning mode (with the -m options) to use at most a given amount of memory and generate multiple output file. These files can then be merged to produce a complete sorted BAM file [TODO - investigate the details of this more carefully].

index

The `index` command creates a new index file that allows fast look-up of data in a (sorted) SAM or BAM. Like an index on a database, the generated `*.sam.sai` or `*.bam.bai` file allows programs that can read it to more efficiently work with the data in the associated files.

tview

The `tview` command starts an interactive ascii-based viewer that can be used to visualize how reads are aligned to specified small regions of the reference genome. Compared to a graphics based viewer like IGV,[6] it has few features. Within the view, it is possible to jumping to different positions along reference elements (using 'g') and display help information ('?').

mpileup

The `mpileup` command produces a pileup format (or BCF) file giving, for each genomic coordinate, the overlapping read bases and indels at that position in the input BAM files(s). This can be used for SNP calling for example.

flagstat

SAMtools - BAMtools

View

```
samtools view sample.bam > sample.sam
```

Convert a bam file into a sam file.

```
samtools view -bS sample.sam > sample.bam
```

Convert a sam file into a bam file. The -b option compresses or leaves compressed input data.

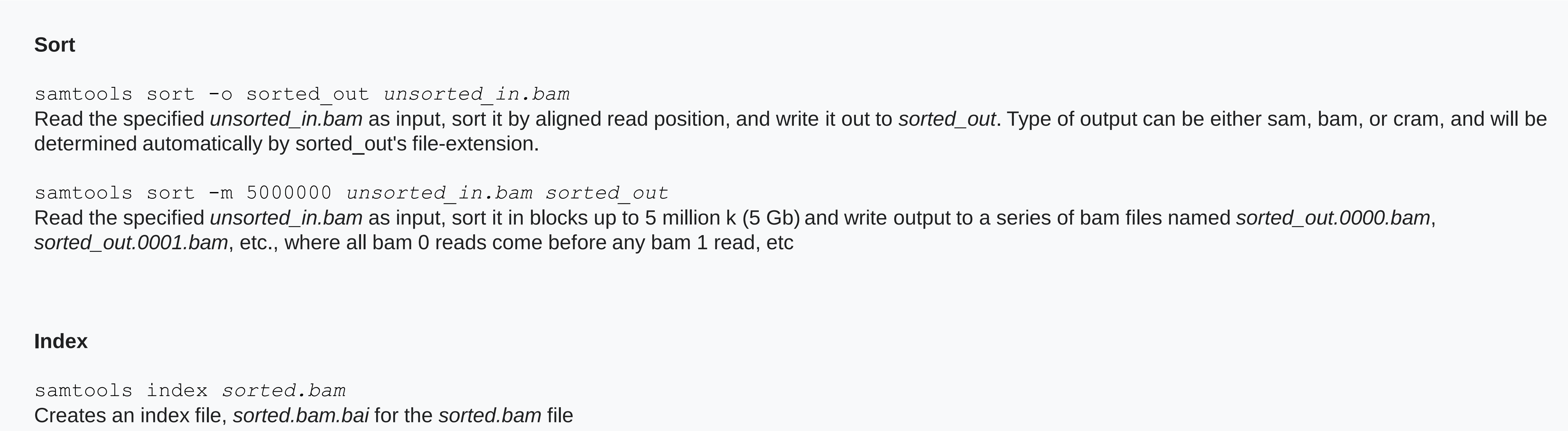
```
samtools view sample_sorted.bam "chr1:10-13"
```

Extract all the reads aligned to the range specified, which are those that are aligned to the reference element named *chr1* and cover its 10th, 11th, 12th or 13th base.
The results is saved to a BAM file including the header. An index of the input file is required for extracting reads according to their mapping position in the reference genome, as created by *samtools index*.

```
samtools view -h -b sample_sorted.bam "chr1:10-13" > tiny_sorted.bam
```

Extract the same reads as above, but instead of displaying them, writes them to a new bam file, *tiny_sorted.bam*. The -b option makes the output compressed and the -h option causes the SAM headers to be output also. These headers include a description of the reference that the reads in *sample_sorted.bam* were aligned to and will be needed if the *tiny_sorted.bam* file is to be used with some of the more advanced SAMtools commands. The order of extracted reads is preserved.

SAMtools - BAMtools



Sort

```
samtools sort -o sorted_out unsorted_in.bam
```

Read the specified *unsorted_in.bam* as input, sort it by aligned read position, and write it out to *sorted_out*. Type of output can be either sam, bam, or cram, and will be determined automatically by sorted_out's file-extension.

```
samtools sort -m 5000000 unsorted_in.bam sorted_out
```

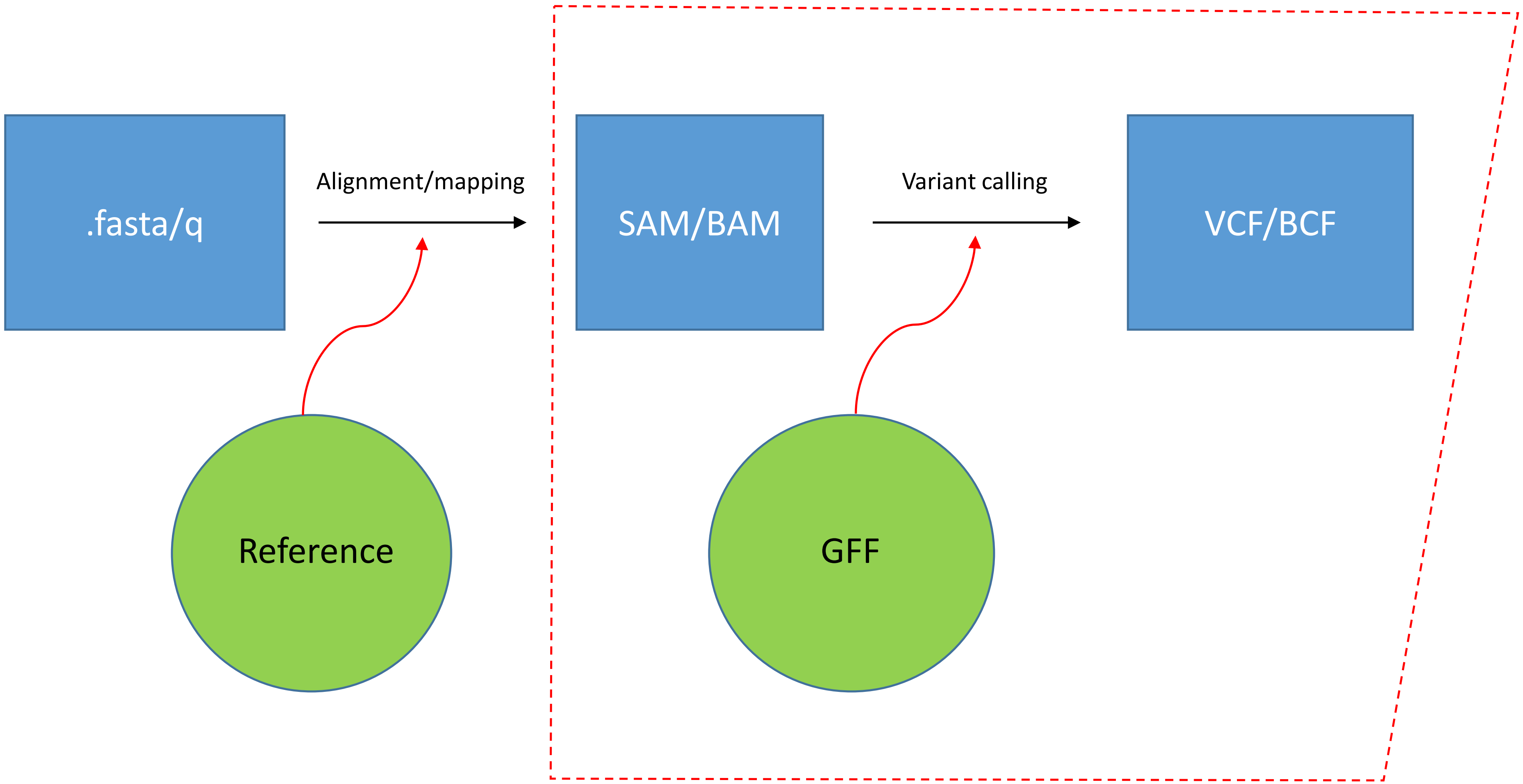
Read the specified *unsorted_in.bam* as input, sort it in blocks up to 5 million k (5 Gb) and write output to a series of bam files named *sorted_out.0000.bam*, *sorted_out.0001.bam*, etc., where all bam 0 reads come before any bam 1 read, etc

Index

```
samtools index sorted.bam
```

Creates an index file, *sorted.bam.bai* for the *sorted.bam* file

Overview


.fasta/q
Alignment/mapping
SAM/BAM
Variant calling
VCF/BCF
Reference
GFF

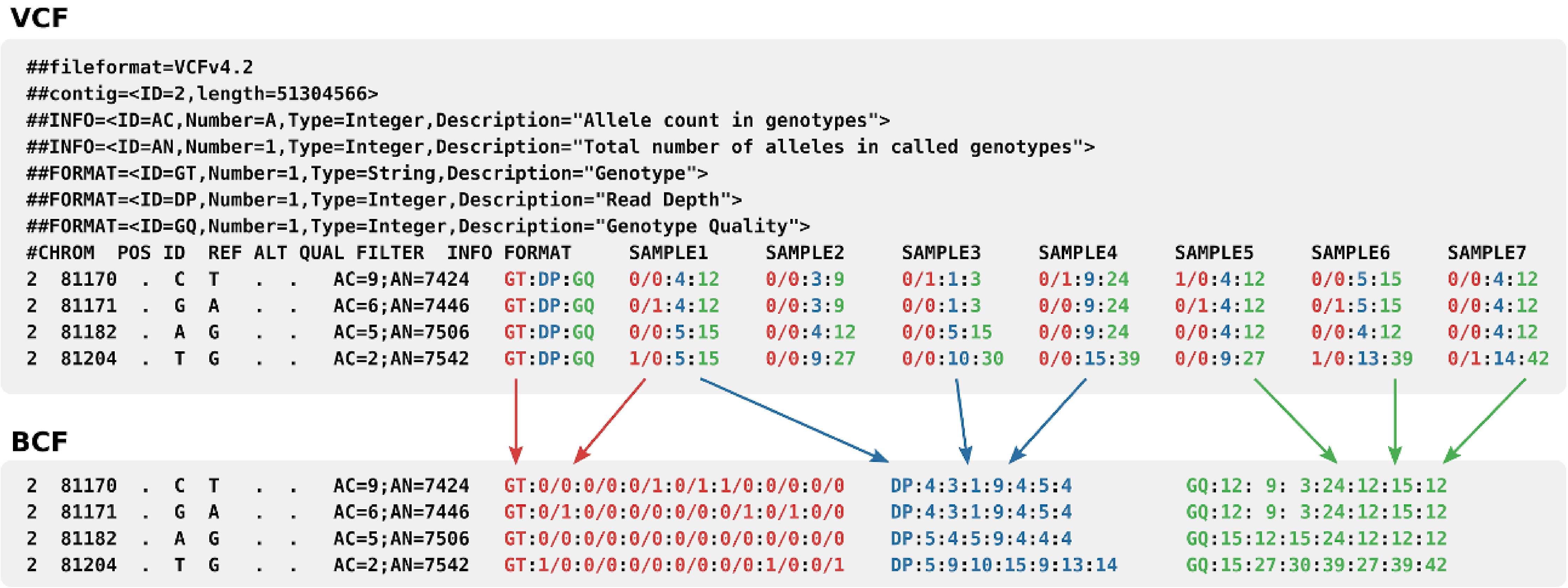
VCF/BCF

- **Variant Call Format** (**VCF**) specifies the format of a text file used in for storing sequence variations
- **bcftools mpileup [OPTIONS] -f ref.fa in.bam [in2.bam [...]]**

- Generate VCF or BCF containing genotype likelihoods for one or multiple alignment (BAM or CRAM) files.

- This is based on the original **samtools mpileup** command producing genotype likelihoods in VCF or BCF format

VCF/BCF



VCF

```
##fileformat=VCFv4.2
##contig=<ID=2,length=51304566>
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
#CHROM  POS ID  REF ALT QUAL FILTER  INFO FORMAT     SAMPLE1     SAMPLE2     SAMPLE3     SAMPLE4     SAMPLE5     SAMPLE6     SAMPLE7
2   81170  .  C  T   .  .   AC=9;AN=7424  GT:DP:GQ  0/0:4:12    0/0:3:9     0/1:1:3     0/1:9:24    1/0:4:12    0/0:5:15    0/0:4:12
2   81171  .  G  A   .  .   AC=6;AN=7446  GT:DP:GQ  0/1:4:12    0/0:3:9     0/0:1:3     0/0:9:24    0/1:4:12    0/1:5:15    0/0:4:12
2   81182  .  A  G   .  .   AC=5;AN=7506  GT:DP:GQ  0/0:5:15    0/0:4:12    0/0:5:15    0/0:9:24    0/0:4:12    0/0:4:12    0/0:4:12
2   81204  .  T  G   .  .   AC=2;AN=7542  GT:DP:GQ  1/0:5:15    0/0:9:27    0/0:10:30   0/0:15:39   0/0:9:27    1/0:13:39   0/1:14:42
```

BCF

```
2   81170  .  C  T   .  .   AC=9;AN=7424  GT:0/0:0/0:0/1:0/1:1/0:0/0:0/0   DP:4:3:1:9:4:5:4         GQ:12: 9: 3:24:12:15:12
2   81171  .  G  A   .  .   AC=6;AN=7446  GT:0/1:0/0:0/0:0/0:0/1:0/1:0/0   DP:4:3:1:9:4:5:4         GQ:12: 9: 3:24:12:15:12
2   81182  .  A  G   .  .   AC=5;AN=7506  GT:0/0:0/0:0/0:0/0:0/0:0/0:0/0   DP:5:4:5:9:4:4:4         GQ:15:12:15:24:12:12:12
2   81204  .  T  G   .  .   AC=2;AN=7542  GT:1/0:0/0:0/0:0/0:0/0:1/0:0/1   DP:5:9:10:15:9:13:14     GQ:15:27:30:39:27:39:42
```

BCFTools

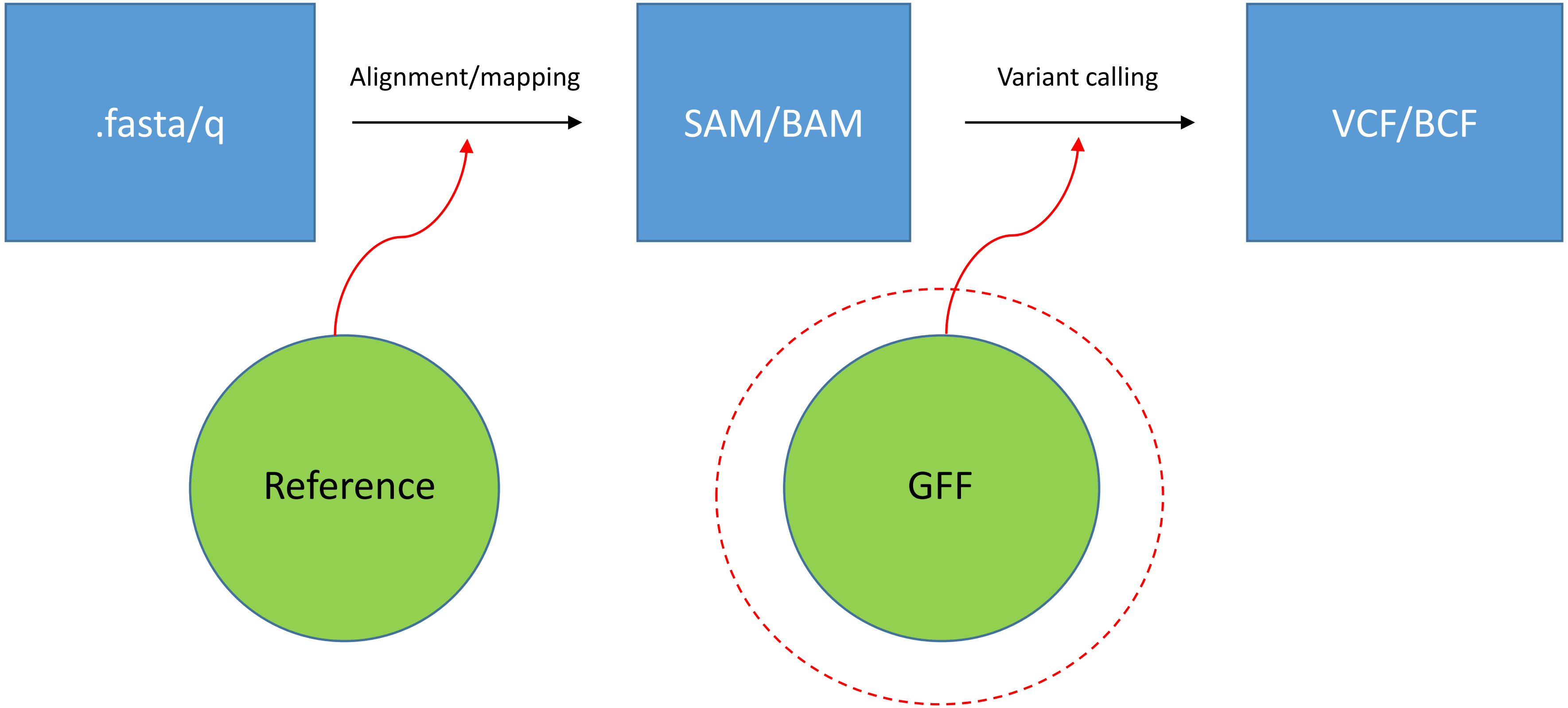
- BCFtools is a set of utilities that manipulate variant calls in the Variant Call Format (VCF) and its binary counterpart BCF. All commands work transparently with both VCFs and BCFs, both uncompressed and BGZF-compressed.
- Most commands accept VCF, bgzipped VCF and BCF with filetype detected automatically even when streaming from a pipe. Indexed VCF and BCF will work in all situations. Un-indexed VCF and BCF and streams will work in most, but not all situations. In general, whenever multiple VCFs are read simultaneously, they must be indexed and therefore also compressed

BCFTools

- **annotate** .. edit VCF files, add or remove annotations
- **call** .. SNP/indel calling (former "view")
- **cnv** .. Copy Number Variation caller
- **concat** .. concatenate VCF/BCF files from the same set of samples
- **consensus** .. create consensus sequence by applying VCF variants
- **convert** .. convert VCF/BCF to other formats and back
- **csq** .. haplotype aware consequence caller
- **filter** .. filter VCF/BCF files using fixed thresholds
- **gtcheck** .. check sample concordance, detect sample swaps and contamination
- **index** .. index VCF/BCF
- **isec** .. intersections of VCF/BCF files
- **merge** .. merge VCF/BCF files files from non-overlapping sample sets
- **mpileup** .. multi-way pileup producing genotype likelihoods
- **norm** .. normalize indels
- **plugin** .. run user-defined plugin
- **polysomy** .. detect contaminations and whole-chromosome aberrations
- **query** .. transform VCF/BCF into user-defined formats
- **reheader** .. modify VCF/BCF header, change sample names
- **roh** .. identify runs of homo/auto-zygosity
- **sort** .. sort VCF/BCF files
- **stats** .. produce VCF/BCF stats (former vcfcheck)
- **view** .. subset, filter and convert VCF and BCF files

VCFStats

Overview



.fasta/q

Alignment/mapping

SAM/BAM

Variant calling

VCF/BCF

Reference

GFF

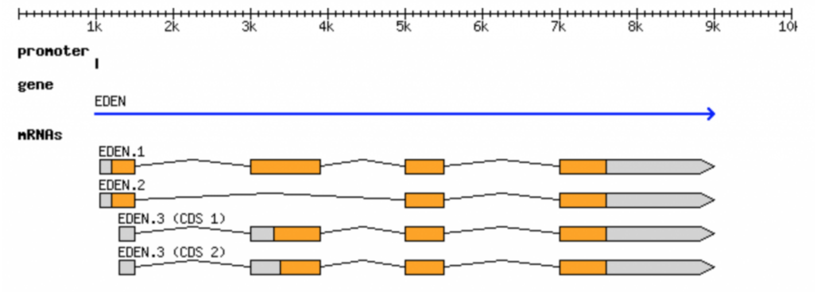
GFF General Feature Format

- Describes genes and other features of DNA, RNA and protein sequences for the genome and contains one line per feature with 9 tab-separated columns of data

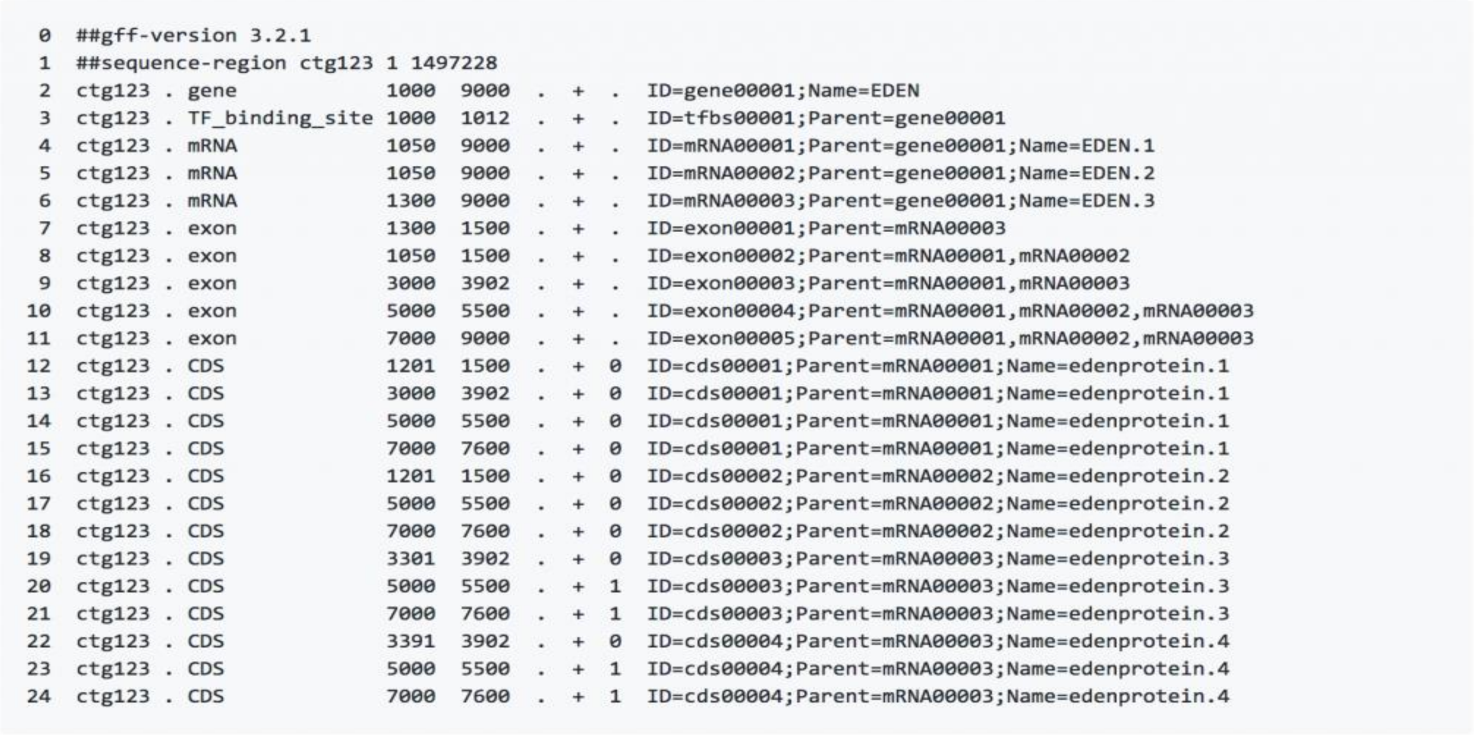
- Created by «variant calling» on a BAM file with a GFF (Generic Feature File)

Existing formats for genetic data such as General feature format (GFF) stored all of the genetic data, much of which is redundant because it will be shared across the genomes. By using the variant call format only the variations need to be stored along with a reference genome.

GFF General Feature Format



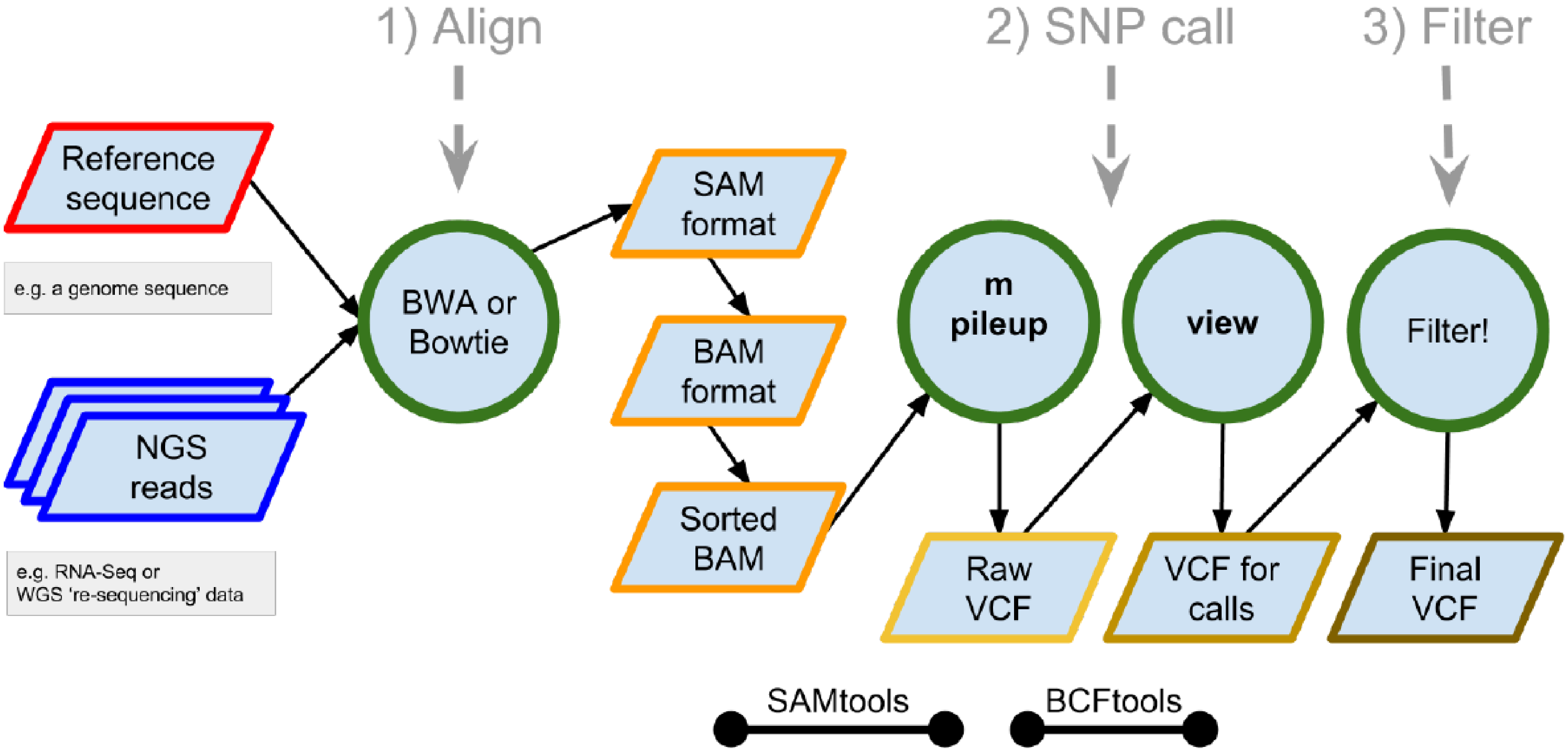
1. Sequence ID
2. Source
3. Feature Type
4. Feature Start
5. Feature End
6. Score
7. Strand
8. Phase
9. Attributes



```
##gff-version 3.2.1
##sequence-region ctg123 1 1497228
ctg123 . gene            1000  9000  .  +  .  ID=gene00001;Name=EDEN
ctg123 . TF_binding_site 1000  1012  .  +  .  ID=tfbs00001;Parent=gene00001
ctg123 . mRNA            1050  9000  .  +  .  ID=mRNA00001;Parent=gene00001;Name=EDEN.1
ctg123 . mRNA            1050  9000  .  +  .  ID=mRNA00002;Parent=gene00001;Name=EDEN.2
ctg123 . mRNA            1300  9000  .  +  .  ID=mRNA00003;Parent=gene00001;Name=EDEN.3
ctg123 . exon            1300  1500  .  +  .  ID=exon00001;Parent=mRNA00003
ctg123 . exon            1050  1500  .  +  .  ID=exon00002;Parent=mRNA00001,mRNA00002
ctg123 . exon            3000  3902  .  +  .  ID=exon00003;Parent=mRNA00001,mRNA00003
ctg123 . exon            5000  5500  .  +  .  ID=exon00004;Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . exon            7000  9000  .  +  .  ID=exon00005;Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . CDS             1201  1500  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             3000  3902  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             5000  5500  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             7000  7600  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             1201  1500  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS             5000  5500  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS             7000  7600  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS             3301  3902  .  +  0  ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS             5000  5500  .  +  1  ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS             7000  7600  .  +  1  ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS             3391  3902  .  +  0  ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
ctg123 . CDS             5000  5500  .  +  1  ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
ctg123 . CDS             7000  7600  .  +  1  ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
```

crypto

Pipeline overview



1) Align reads to reference (using BWA)

1. Index the reference (genome) sequence

```
bwa index my.fasta
# The various index files are output in the CWD
```

2. Perform the alignment

```
bwa aln [opts] my.fasta my.fastq > my.sai
```

3. Output results in SAM format (single end)

```
bwa samse my.fasta my.sai my.fastq > my.sam
```

1) Align reads to reference (using BWA)

1. Index the reference (genome) sequence

```
bwa index my.fasta
# The various index files are output in the CWD
```

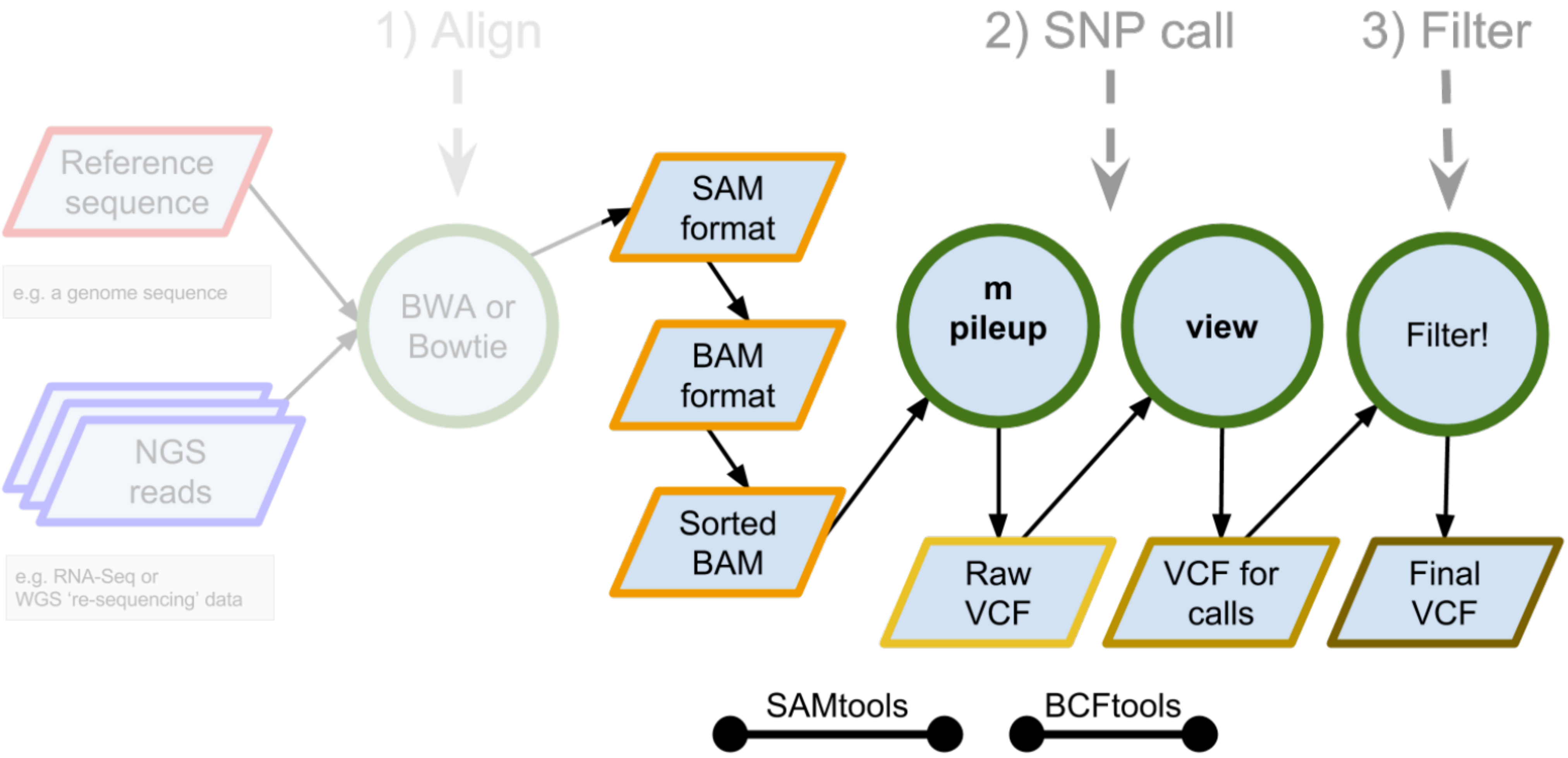
2. Perform the alignment

```
bwa aln [opts] my.fasta my.fastq > my.sai
```

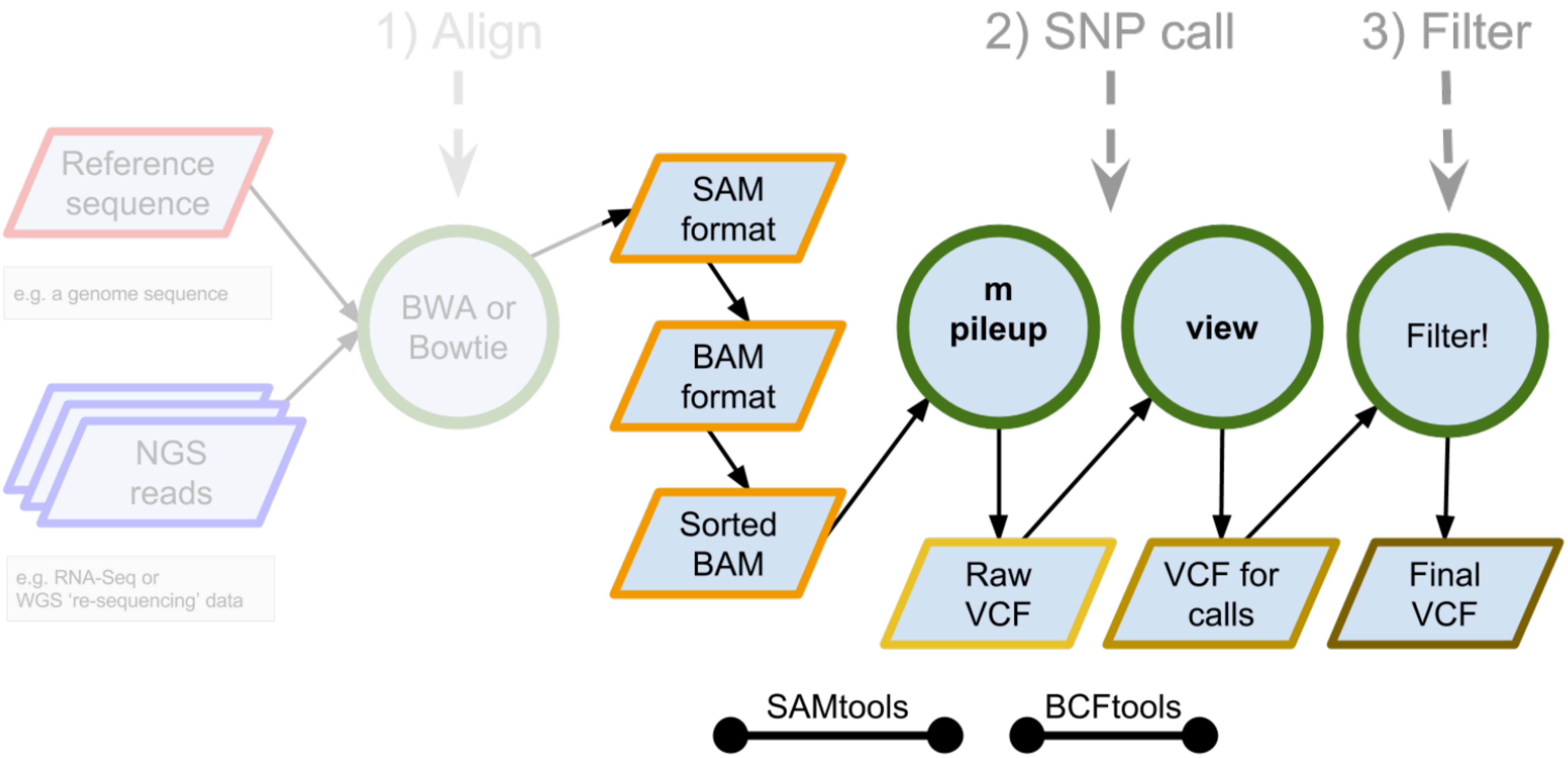
3. Output results in SAM format (single end)

```
bwa samse my.fasta my.sai my.fastq > my.sam
```

Alignment is done! Next, SNP calling!!



Alignment is done! Next, SNP calling!!



First... convert alignments (using SAMtools)

1. Convert SAM to BAM for sorting

```
samtools view -S -b my.sam > my.bam
```

2. Sort BAM for SNP calling

```
samtools sort my.bam my-sorted
```

Alignments are both:

- compressed for long term storage and
- sorted for variant discovery.

2) Call SNPs (using SAMtools)

1. Index the genome assembly (again!)

   ➢
   ```
   samtools faidx my.fasta
   ```

2. Run 'mpileup' to generate VCF format

   ➢
   ```
   samtools mpileup -g -f my.fasta my-sorted-1.bam my-sorted-2.bam my-sorted-n.bam > my-raw.bcf
   ```

NB: All we did so far (roughly) is to perform a format conversion from BAM to VCF!

2) Call SNPs (using bcftools)

3. Call SNPs...
   - ➢ `bcftools view -bvcg my-raw.bcf > my-var.bcf`

Again...

- `samtools mpileup`
  - Collects summary information in the input BAMs, computes the likelihood of data given each possible genotype and stores the likelihoods in the BCF format.
- `bcftools view`
  - Applies the prior and does the actual calling.

resources

- VEuPathDB
- cryptoDB