# Practical Example: NGS of Genomics Data using Illumina and Nanopore

**Ivan Gesteira Costa & Fabio Ticconi**
**IZKF Research Group Bioinformatics**

IZKF Interdisziplinäres Zentrum für Klinische Forschung | RWTH AACHEN UNIVERSITY

# Contact Information

Fabio Ticconi

fabio.ticconi@rwth-aachen.de

Room 3.03

# Overview

- Practical example of **Variant calling** with different NGS data:
  - Illumina HiSeq: industry standard, short reads, high depth
  - Nanopore MinION: relatively new, very long reads, low depth
- Data from the CAMDA 2017 Conference
  - **Salmonella Enterica** samples sequenced with both methods
  - NCBI genome entry: https://www.ncbi.nlm.nih.gov/nuccore/NC_003198.1
  - http://camda2017.bioinf.jku.at
- Slides available on course website

# Download Data

```
# download data
wget costalab.ukaachen.de/lecture3/data.tar.gz

# unpack it and change directory
tar xvfz data.tar.gz
cd data
```

# What's inside the data directory?

– **pipeline.sh** performs variant calling on both Illumina and Nanopore data

– open it in a text editor and run it line by line in a terminal, as the lecture progresses

– Reference genome:

```
salmonella_enterica_typhi_strCT18.fasta
```

– Illumina data:

```
Salmonella_enterica_hiseq_1
```

– Nanopore data:

```
Salmonella_enterica_minion_1
```

# Illumina data:
# from Alignment to Variant calling

# Illumina Pipeline

- Input:
  - reference genome in **FASTA** format:
    `salmonella_enterica_typhi_strCT18.fasta`
  - reads in **FASTQ** format:
    - `r1.fastq` and `r2.fastq` files (paired-end reads)
- Align to reference and create **SAM** file (bwa-mem)
- SAM-to-**BAM** (samtools)
- Add read groups and mark duplicates (picard tools)
- Variant calling to produce **VCF** file (GATK)

# FASTA File

– Stores DNA sequences in a text-based file

– Mainly used to store large genomic sequences

– Header (lines that start with '>') + DNA sequence

– DNA alphabet: A, C, G, T, N

```
>SEQ_1
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
>SEQ_2
AGCAGTTGGGGTTCATCGAATTTGGGGTTCATCCATTAAAGCAGAATCCATTTGATCAAT
```

# FASTQ File

– Also text-based. Mainly used to store short DNA sequences (reads) from NGS-based experiments.

– **Line 1:** Begins with a '@' character and is followed by a sequence identifier and an optional description (like a FASTA title line).
– **Line 2:** DNA sequence.
– **Line 3:** Begins with a '+' character and is optionally followed by the same sequence identifier (and any description) again.
– **Line 4:** Encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

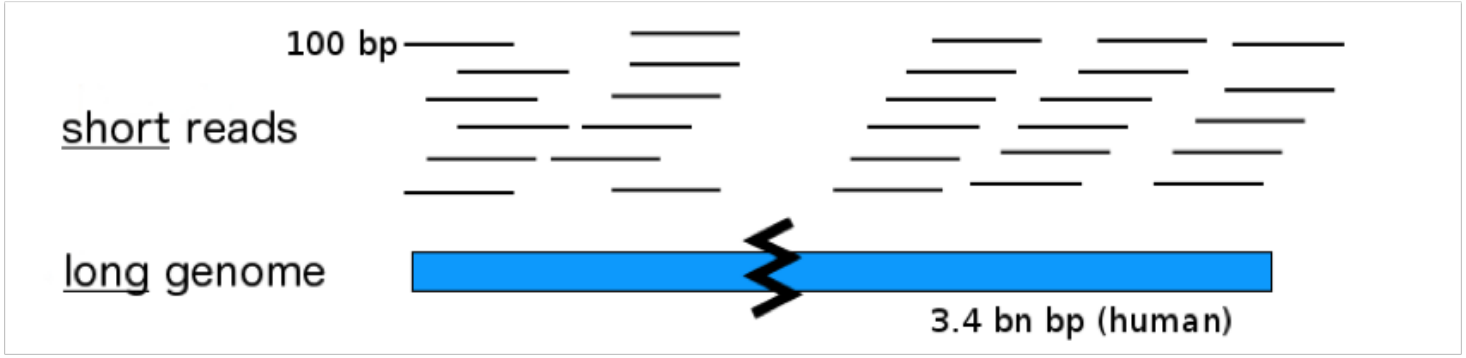
```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

# Alignment Problem

– A large reference sequence is given (genome)
  – up to billions of base pairs

– Query: reads (DNA sequences)

– Problem: find most probable position of the read in the genome (by inexact string matching)



# Burrows-Wheeler Aligner (BWA)

– Three available algorithms: backtrack (up to 100bp), sw (>= 70bp) and mem (>= 70bp)

– **bwa-mem** is the latest and recommended for data with reads >= 70bp

– More Information:

Paper: https://arxiv.org/abs/1303.3997
Website: http://bio-bwa.sourceforge.net

# Perform Alignment

– Align with BWA-mem:

**1. index:** creates index of reference for fast look-up (needed to quickly access potentially very big FASTA files)

```
bwa index ref.fa
```

**2. mem:** perform alignment (produces SAM file)

```
# paired-ends
bwa mem ref.fa r1.fastq r2.fastq > reads.sam

# single reads (always use paired-end data if possible)
bwa mem ref.fa reads.fastq > reads.sam
```

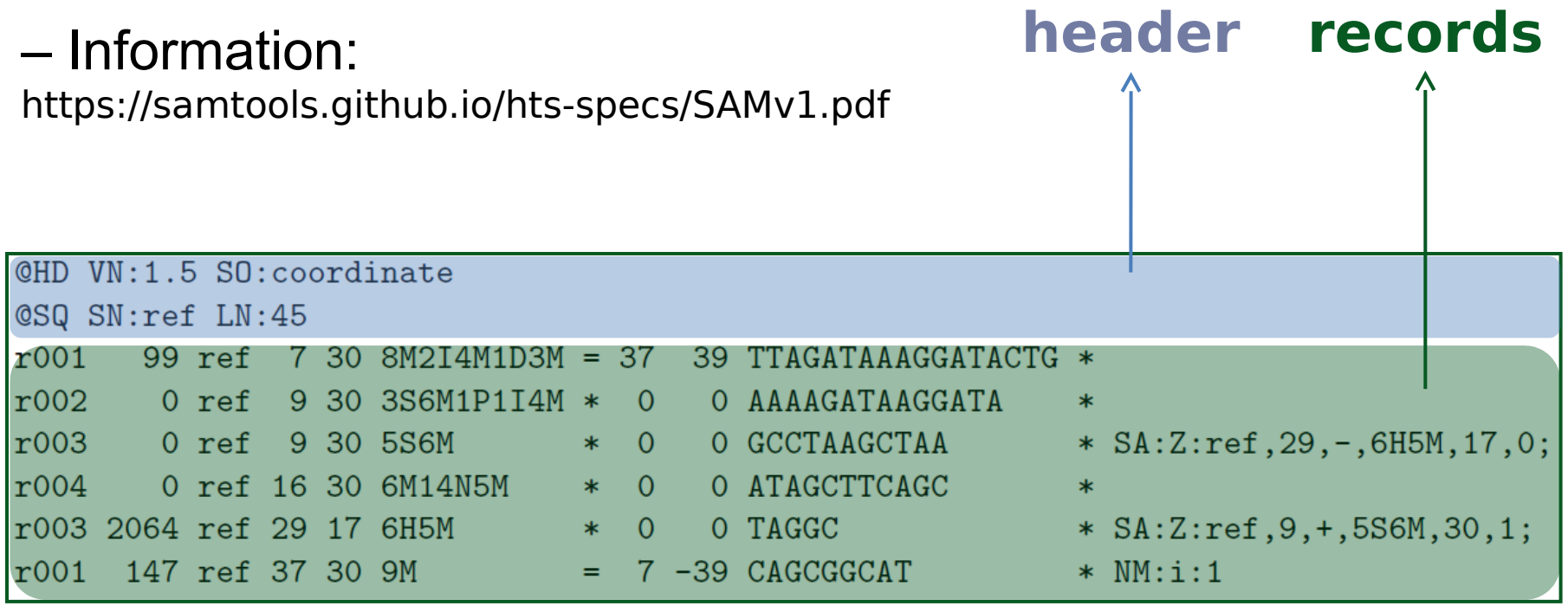
# SAM File

– Sequence Alignment/Map format.

– Text-based tab-delimited file.

– Header + records (aligned reads)

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



# SAM 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 |

```
@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
```

# BAM File

– Binary Alignment/Map format – compressed version of SAM.

– Compression: BGZF block compression.

– Efficient random access: UCSC bin/chunk scheme.

– BAI index files.

– More Information:

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2723002/
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC186604/

# Samtools

– Provides various utilities for manipulating alignments in the SAM format.

– Tools useful for quality check and bias correction.

– More Information:

Paper: http://www.ncbi.nlm.nih.gov/pubmed/19505943
Website: http://samtools.sourceforge.net/

# Convert SAM to BAM

– Using samtools:

1. **view:** shows binary format, which can be redirected to file

```
samtools view -hbS reads.sam > reads.bam
```

2. **sort:** sorts alignment by coordinates

```
samtools sort reads.bam > reads.sorted.bam
```

3. **index:** creates alignment's index for fast random access.

```
samtools index reads.sorted.bam
```

# Picard Tools

– Provides various utilities to manipulate SAM, BAM, CRAM and VCF files

– Complements GATK, which we'll also use

– We use it to add Read Groups and mark duplicates

– More Information:

Website: https://broadinstitute.github.io/picard

# Add Read Groups

– Read Groups are just metadata

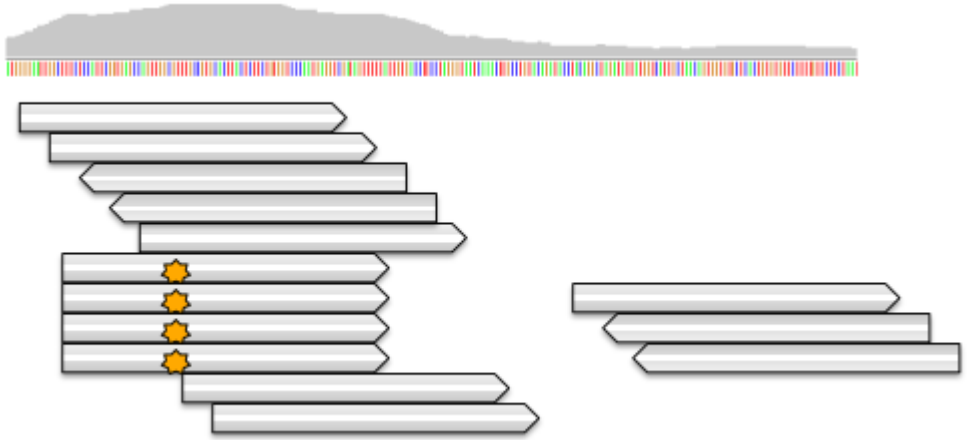
– Useless to us, but GATK fails without them

– Is useful if merging together BAMs from different experiments

– This command can be run on either SAM or BAM files:

```
java -jar $PICARD AddOrReplaceReadGroups \
          INPUT=reads.sorted.bam OUTPUT=reads.rg.sorted.bam \
          RGID=1 RGLB=library RGPL=ILLUMINA RGPU=unit1 RGSM=Sample1
```

# Mark Duplicates



```
java -jar $PICARD MarkDuplicates \
          I=reads.rg.sorted.bam O=reads.final.bam
```

Figure from Simon Rasmussen, Center for Biological Sequence Analysis, Denmark

# GATK

– Genome Analysis Toolkit

– Can be configured extensively with filters and preprocessing tasks

– Well documented

– Main task: find variants

– More Information:

Paper: http://genome.cshlp.org/content/20/9/1297
Website: https://software.broadinstitute.org/gatk

# Final step: SNP Calling

– Takes the final BAM and uses HaplotypeCaller to identify SNPs

– Must specify ploidy: by default it expects the data to come diploid organisms (ie, with two sets of chromosomes), but bacteria are haploid

– Produces VCF file with variants

```
java -jar $GATK -ploidy 1 -I reads.final.bam -R $REF \
          -T HaplotypeCaller -o reads.vcf
```

# VCF File

– Variant Call Format

– One variant per line (SNPs, Indels)

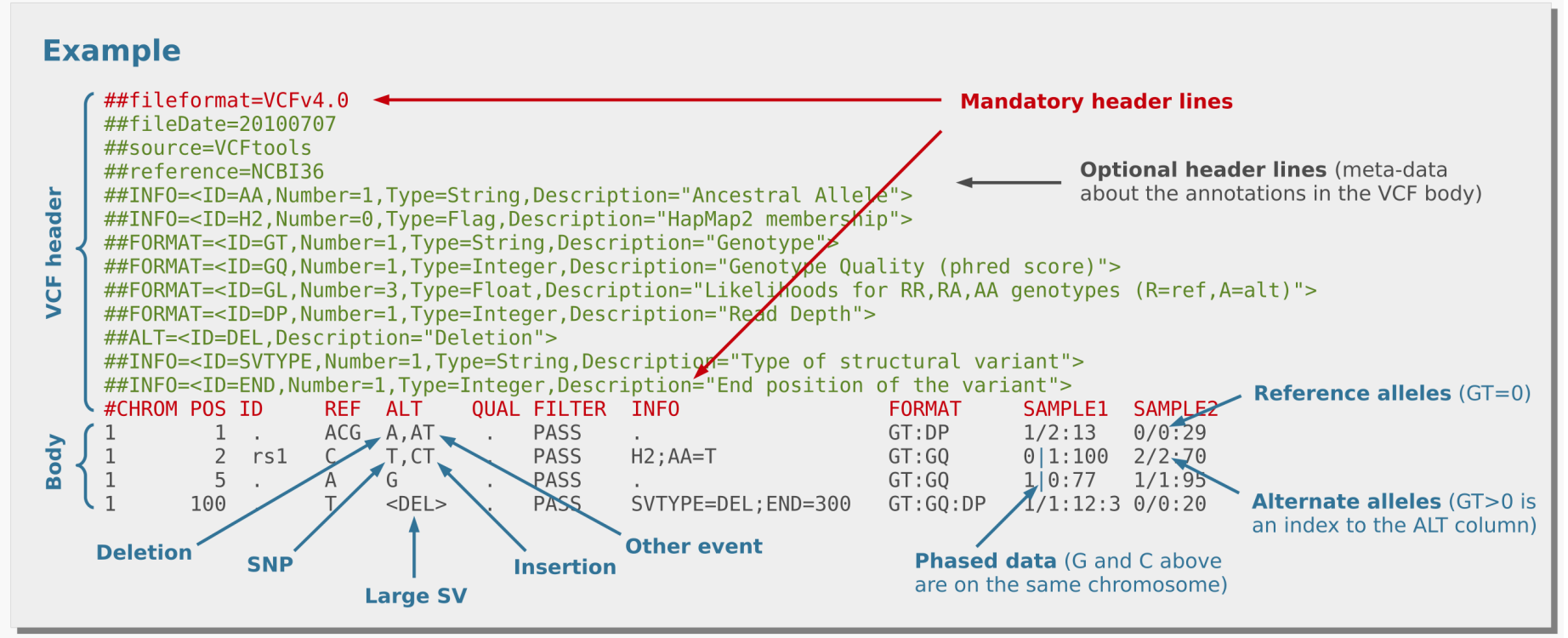



Figure from: http://vcftools.sourceforge.net/VCF-poster.pdf

# IGV

– Tool for visualising sequences, reads and/or variants

– We must setup the Salmonella genome then load the following files we have generated:

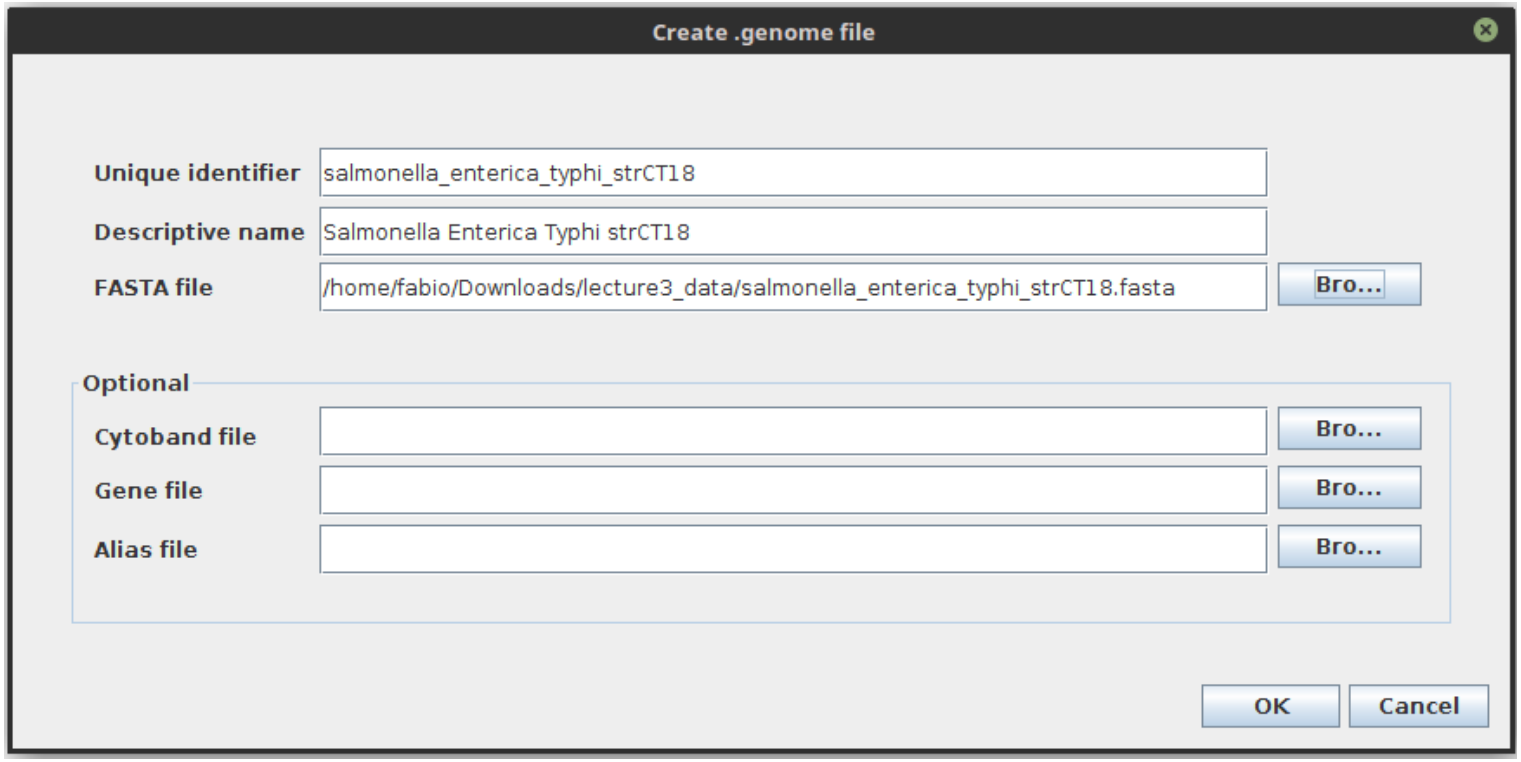
  – **reads.final.bam**
  – **reads.vcf**

– More Information:

  Paper: http://www.nature.com/nbt/journal/v29/n1/abs/nbt.1754.html
  Website: http://software.broadinstitute.org/software/igv/

# Create .genome file

– From menu: Genomes – Create .genome File



# Load tracks

– From menu: File – Load from File

– Illumina: select **reads.final.bam** and **reads.vcf**

– Can you locate the variants you have found?

# Nanopore:
# A slightly different pipeline

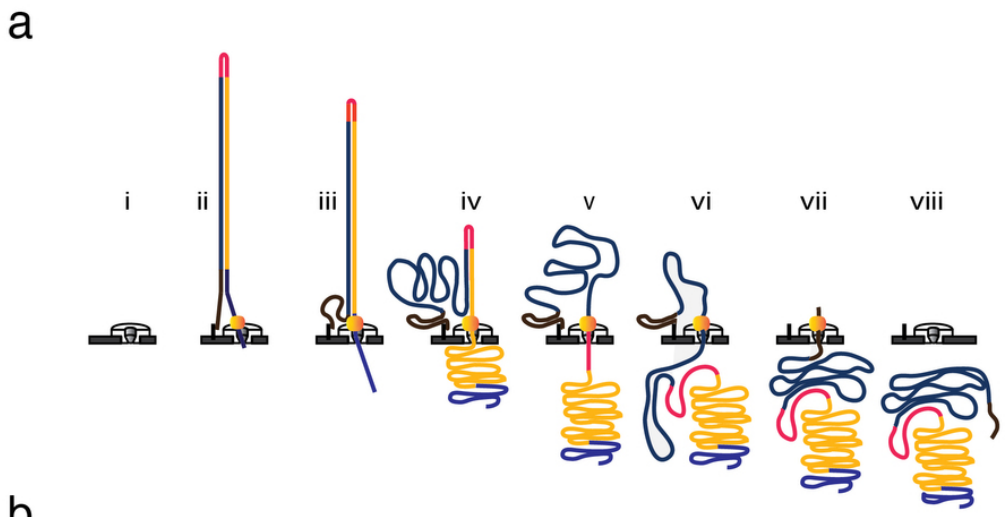
# FAST5 File

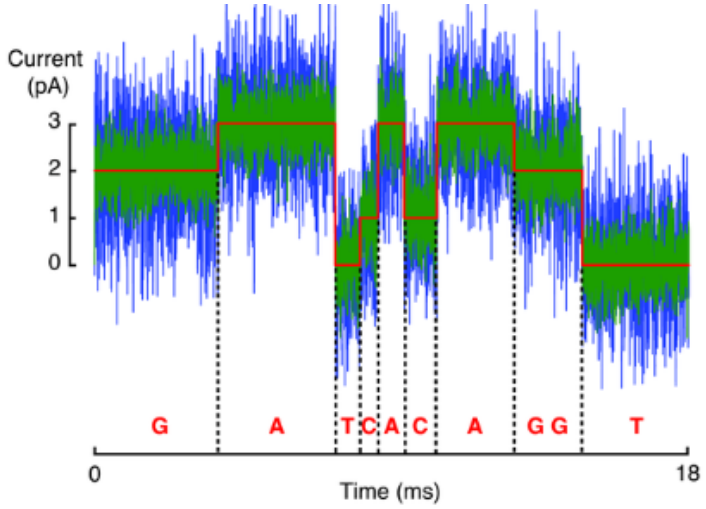
- Variation of HDF5 format, which is a compressed, hierarchical format for complex data

- contains reads as well as metadata
  (eg, temperature during sequencing, HMM states, 5-mer sequences)

- can be easily converted to FASTA or FASTQ

# Read Events




| strand | mean | start | stdv | length | model_state | p_A | p_C | p_G | p_T |
|---|---|---|---|---|---|---|---|---|---|
| template | 66 | 109444.6112 | 1.7 | 0.0148 | GG**A**AG | **0.29** | 0 | 0.05 | 0 |
| template | 68 | 109444.626 | 1.6 | 0.1738 | GG**A**AG | **0.06** | 0 | 0.01 | 0 |
| complement | 52 | 109452.6234 | 1.4 | 0.0138 | CT**A**TA | **0.53** | 0.29 | 0.04 | 0.11 |
| complement | 48 | 109452.6372 | 1.3 | 0.0364 | TA**T**AT | 0.27 | 0.20 | 0.00 | **0.51** |

**This is just an extract**. To get the full output (first 5 events):

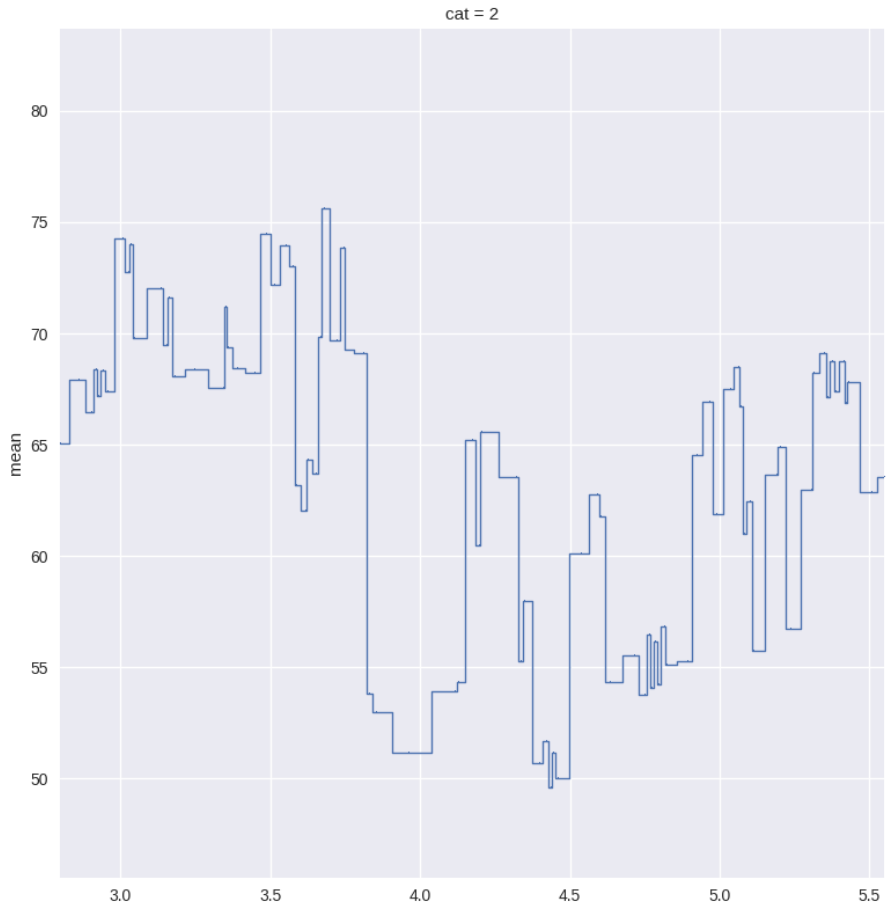
```
poretools events fast5/ | head -n 5 | column -t | less -S
```

# “Squiggle plots” to view signal

– on each fast5, you can run the command below to get a “squiggle plot”

– it shows the signal, eg the mean value of the current for the duration of the sequencing, for that specific pore

– it’s divided in 6 contiguous chunks (this one is the second, eg it doesn’t start from 0)




```
poretools squiggle fast5/file.fast5 --saveas png
```

# Temperature and more

– Other information can be extracted with the “index” command:

```
poretools index fast5/ | head -2 | column -t
```

– It looks like this (only a partial extract):

```
source_filename 2d_length  asic_id  asic_temp  heatsink_temp
file1.fast5     5398       33031    37.8       37.125
file2.fast5     None       33031    37.8       37.125
```

– Even more can be extracted with the “metadata” command (output not shown, try it):

```
poretools metadata --read fast5/file.fast5
```

# For more to dive into..

– See **Poretools documentation** for other plots you can generate out of FAST5 files:
http://poretools.readthedocs.io/en/latest

– You can align the signal k-mers to the reference using **Nanopolish**:
https://github.com/jts/nanopolish

– Look up the new chemistry for Nanopore (R9)
  – Neural Network for base calling instead of HMM
  – poretools and nanopolish should already work

# Convert FAST5 to FASTA

- We must convert the FAST5 files to FASTA (only **nanopore**)

- We don’t need FASTQ: quality has a different meaning, and bwa-mem doesn’t use it for mapping anyway

```
poretools fasta fast5/ --min-length 70 > reads.fa
```

- We enforce a minimum read length of 70, since bwa-mem underperforms with short reads. Several options available – explore!

- nanopolish is an alternative tool, which also allows to select only high quality 2D reads:

```
nanopolish extract --type 2d fast5/ > reads.fa
```

These are usually much lower in number than the total, so we won’t use it today

# Nanopore pipeline: go ahead!

- Input:
  - reference genome in **FASTA** format:
    `salmonella_enterica_typhi_strCT18.fasta`
  - reads in FASTA format
- Align to reference and create **SAM** file (bwa-mem)
- SAM-to-**BAM** (samtools)
- Add read groups (picard tools)
- We don’t mark duplicates
- Variant calling to produce **VCF** file (GATK)

# Variant calling: one difference

– With Nanopore, we want to retain as many reads as possible so we reduce the mapping quality filter (default 20)

– Also, since our reads were in FASTA, there are no base qualities. GATK needs them, so we set them to an arbitrary 30

```
java -jar $GATK -ploidy 1 -I reads.final.bam -R $REF \
          -T HaplotypeCaller -o reads.vcf \
          -mmq 5 --defaultBaseQualities 30
```

# Load in IGV

– From menu: File – Load from File

– select **reads.final.bam** and **reads.vcf**

# The End

Thanks for your attention!