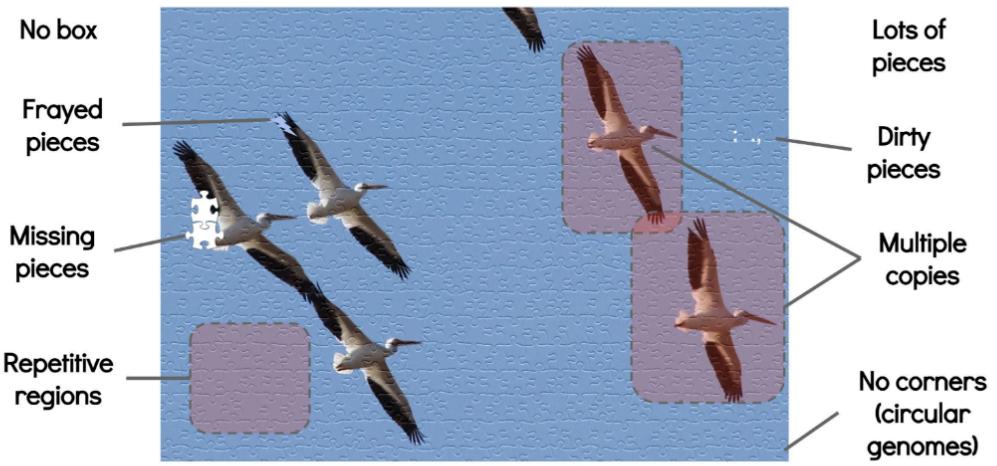
What makes a jigsaw puzzle hard?

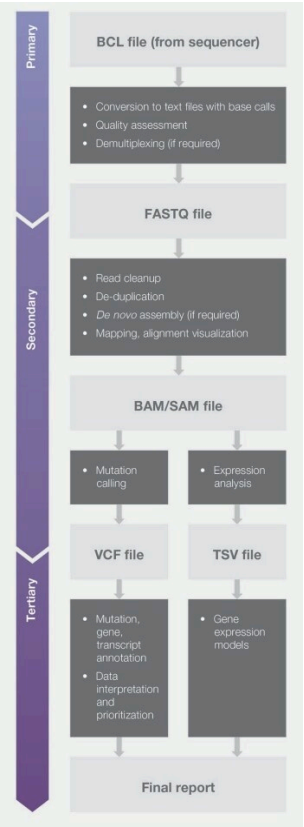


- What helps? Larger pieces (read length); fewer dirty or frayed pieces (errors in reads). fewer repeats and copies…

In a **jigsaw puzzle**, you:

1. **Start with the edges** – Find and assemble the corner and edge pieces to form the puzzle's boundary.

2. **Sort by color or pattern** – Group pieces by similar colors, textures, or parts of an image.



3. **Fit pieces together** – Match shapes and images to gradually complete sections of the picture.

4. **Combine sections** – Merge the completed clusters until the full image is formed.

It's a process of **visual analysis**, **pattern recognition**, and **problem-solving**, much like genome assembly in bioinformatics.

Simplified Overview of NGS Data Analysis Workflow

Next-generation sequencing (NGS) data analysis is crucial for accurate interpretation and conclusions. This process involves three main stages: primary, secondary, and tertiary analysis. Understanding the basics of these stages, common tools, and the general workflow can help plan NGS experiments effectively.

Key Considerations

NGS data analysis requires handling large files (1–3 GB), making access to powerful computing systems essential. Scripting skills in

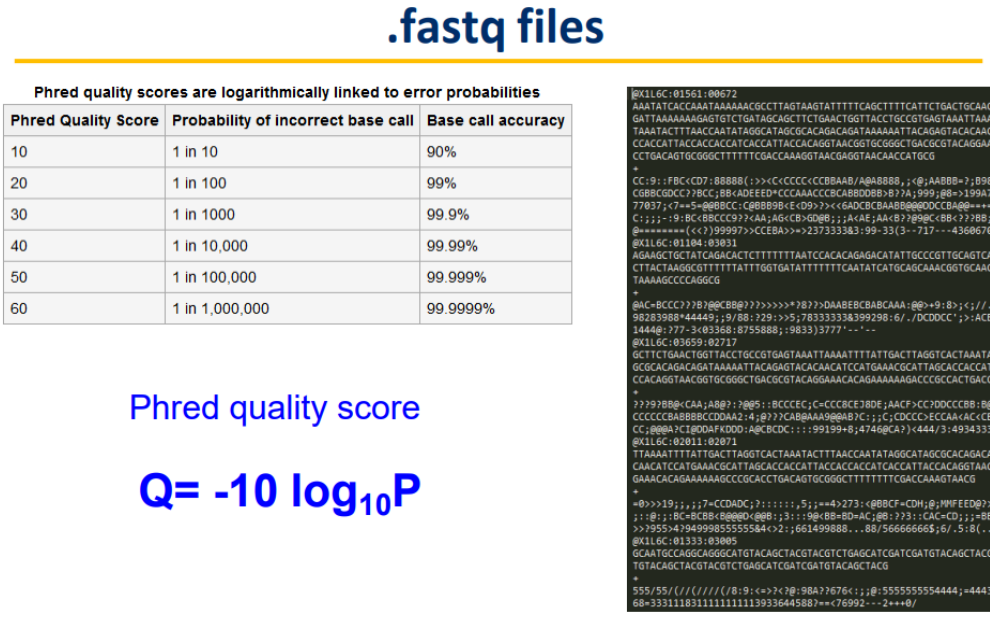
Python, Perl, R, or Bash are often necessary for processing the data. The three core analysis stages are:

- **Primary Analysis**: Quality check of raw data, usually done by sequencer software.
- **Secondary Analysis**: Conversion of data to results like alignment or expression using bioinformatics tools.
- **Tertiary Analysis**: Drawing conclusions about genetic features, mutations, or expression, typically leading to a final report.

1. Primary Analysis

Primary analysis assesses the quality of sequencing data. The raw data is converted from binary files (.bcl) to text-based FASTQ format. The key quality metrics include:

- **Yield**: Total base reads achieved.
- **Error Rate**: Incorrect base calls.
- **Phred Quality Score**: Measures base call accuracy.
- **% Aligned**: Percentage of sequences aligned to the reference genome.



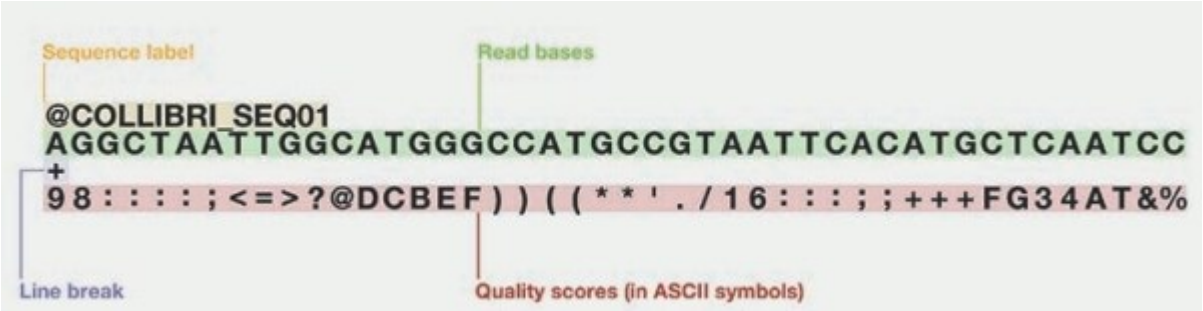
Phred score formula:

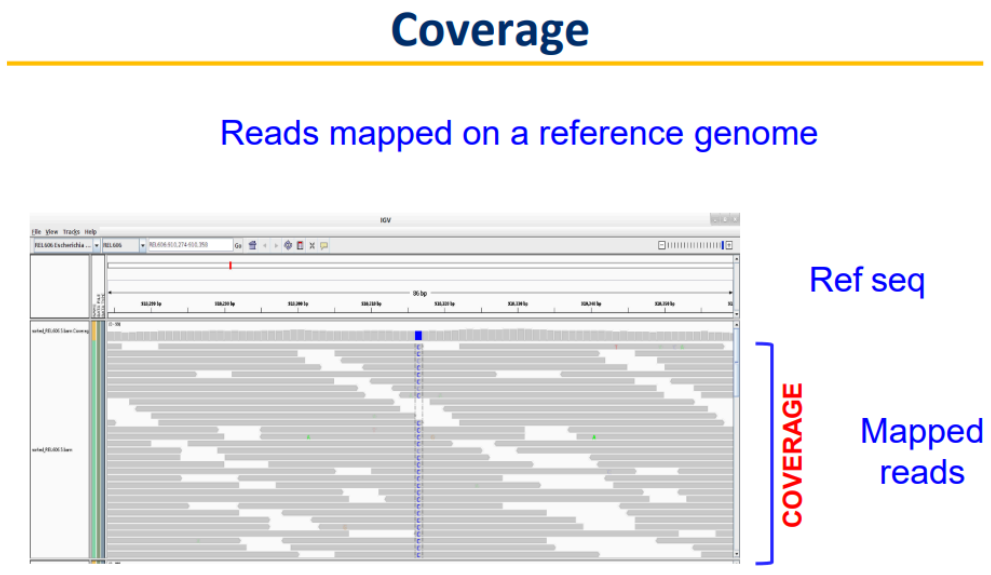
$Q = -10 \log_{10} P$, or
$P = 10^{(-Q/10)}$

Phred quality score (Q)	Probability of incorrect base call (P)	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1,000	99.9%
40	1 in 10,000	99.99%
50	1 in 100,000	99.999%
60	1 in 1,000,000	99.9999%

2. Demultiplexing

When multiple samples are sequenced together, demultiplexing separates them into individual FASTQ files based on index reads. These files are associated with specific samples.





🧬 Quality Scores (Phred-33 scale)

Each character in Line 4 maps to a **Phred quality score**, which reflects the probability of a base call being wrong. The formula is:

$$Q = -10 \times \log_{10}(P)$$

```
Quality encoding: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
                  |         |         |         |         |
   Quality score: 0........10........20........30........40
```

For example, a base with a score of 31 is very accurate (~0.08% error chance), but a score of 2 (common for 'N') means >50% error chance. **Bad reads have lots of low scores or Ns**.

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10,000	99.99%

Table 1. Summary of secondary analysis steps.

Step	Purpose
Read cleanup	This step removes low-quality sequence reads and/or portions of reads (also called "soft-clipping") from the data. The output from this step is a "cleaned" FASTQ file.
Alignment	This step matches, or maps, each base call to its corresponding location in the genome (or transcriptome for RNA) of the organism of interest. Although each individual organism generally has a unique genome, there are reference genomes and transcriptomes published for use as common alignment tools for the NGS community. The alignment output is a Binary Alignment Map (BAM) file that contains the base call alignments to the reference sequences.
Mutation calling	In this step, mutations, features, and/or other anomalies that do not match the reference genome of interest are identified (or called) from the BAM file; these calls are represented in a text-based tab-delimited file in VCF format.
Gene expression analysis	In the context of RNA sequencing, this can also include gene count and (differential) expression data. A standard file format (like VCF) does not exist for such data, but most tools will output in a tab-delimited format (e.g., Tab Separated Values (TSV) files) with each column representing samples, genes, amplicon IDs, raw counts, normalized counts, etc.

3. Secondary Analysis

Secondary analysis involves processing the demultiplexed FASTQ files, including:

- **Read Cleanup**: Removes low-quality reads and trims adapters.
  Sequence Alignment: Matches reads to a reference genome. Output is a BAM file.
  Mutation Calling: Identifies mutations from the aligned data.

Gene Expression Analysis: Quantifies gene expression in RNA sequencing.

Key Steps in Secondary Analysis:

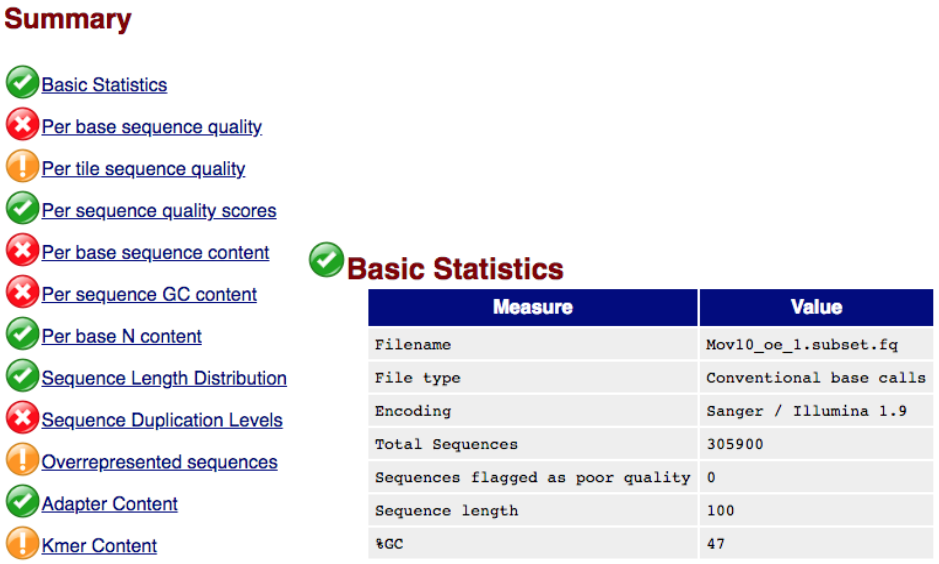
- **Read Cleanup**: Removes low-quality reads and short fragments, ensuring clean data for further analysis.
- **Sequence Alignment**: Aligns reads to a reference genome (e.g., GRCh38 for human). Tools like BWA or Bowtie 2 are commonly used.
- **Mutation Calling**: Identifies mutations by comparing the aligned data to a reference genome. The results are saved in a VCF format.
- **Gene Expression Analysis**: Measures gene expression, often with RNA-Seq, outputting results in formats like TSV.

✅ FastQC – Tool for Checking Sequence Quality

FastQC gives an overview of sequencing quality, showing whether data has any potential problems.

Key Features:

- Accepts FASTQ, BAM, SAM formats
- Highlights possible sequencing issues
- Visual summaries in an easy-to-understand HTML report



- ---

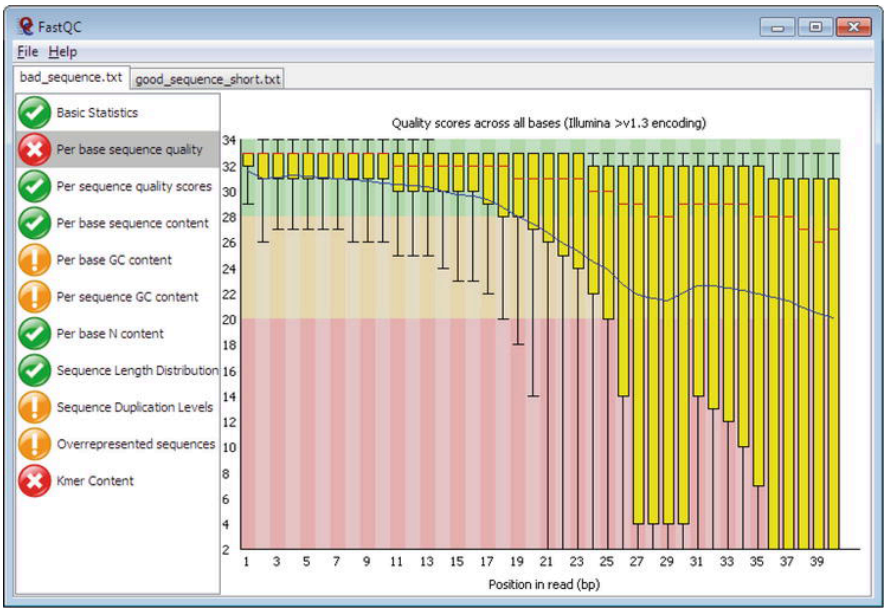
⚠️ Sequencing Error Sources

Expected:

- **Signal decay:** Fluorescent signal weakens over cycles.
  Phasing: Reads lose sync due to chemical inefficiencies, reducing quality at the end of the read.

Worrisome:

- **Overclustering:** Too many reads on the flow cell → mixed signals.
  Instrument failure: Sudden quality drops may require re-sequencing.



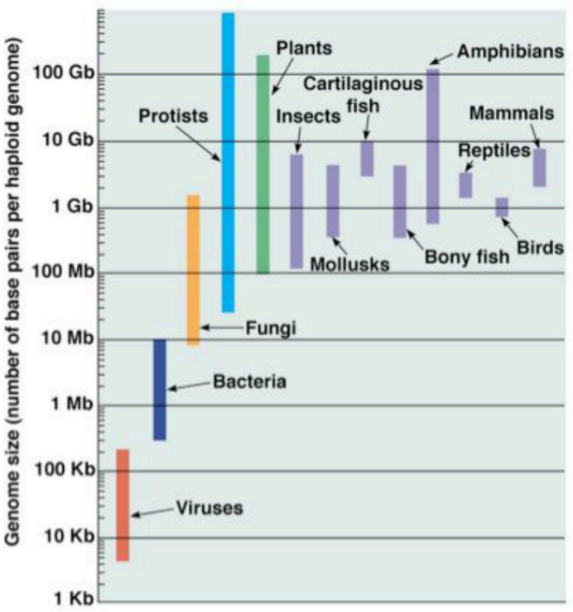
4. Tertiary Analysis

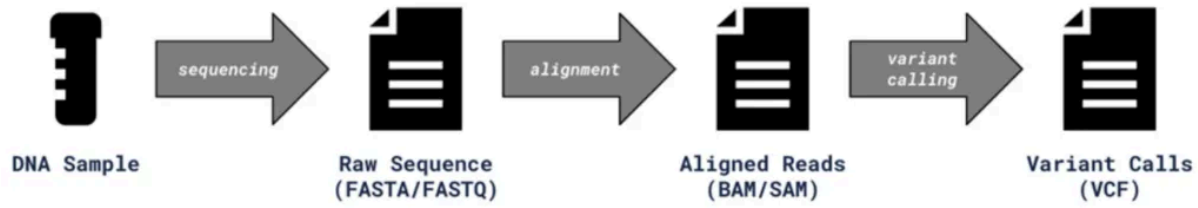
Tertiary analysis interprets the data, drawing conclusions about mutations, gene expressions, and genetic features, often forming the basis for further actions or reports.

Sure! Below is an updated version of the **bioinformatics pipeline workflow** that now includes the file formats you mentioned (FASTA, FASTQ, BAM, and VCF), along with their usage in the steps:

Key considerations

- sequencing coverage
- errors in reads
- reads lengths
- non-uniqueness of the genome (repeats)
- genome size
- genome heterozygosity and ploidy
- running time and memory





🧬 What is Mapping in Genomics?

When you sequence DNA or RNA, you get a bunch of short reads. But these reads have **no location info** — we don't know where in the genome they came from.

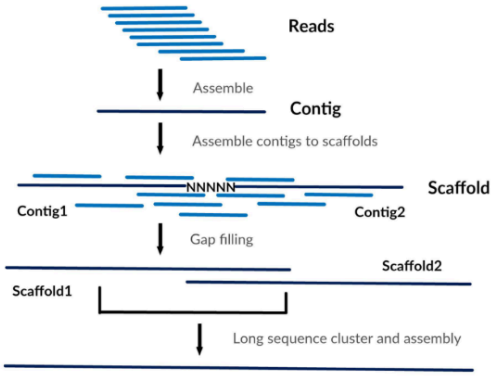
To figure that out, we use a **reference genome** (a known full sequence of a species) and align the reads to it. This step is called **mapping**, and it's a key part of analyzing sequencing data like RNA-Seq or ChIP-Seq.

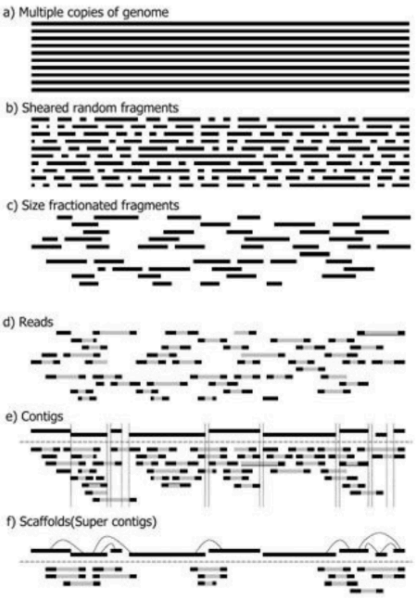
---

2. Challenges in WGS/WES Analysis

- High computational demands.
- Most tools focus on one part of the process.
- No single "gold standard" method for analyzing all diseases—strategies must be disease-specific

Terminology





Chromosome Improvements and impacts of GRCh38 human reference on high throughput sequencing data analysis

Yan Guo [a,*], Yulin Dai [a], Hui Yu [a], Shilin Zhao [a], David C. Samuels [b], Yu Shyr [c,*]

- contig: contiguous stretch of assembled sequence
- scaffold: ordered set of contigs with gaps (Ns) placed between

This is a Bioinfo class in LaunchED

This is a Bio

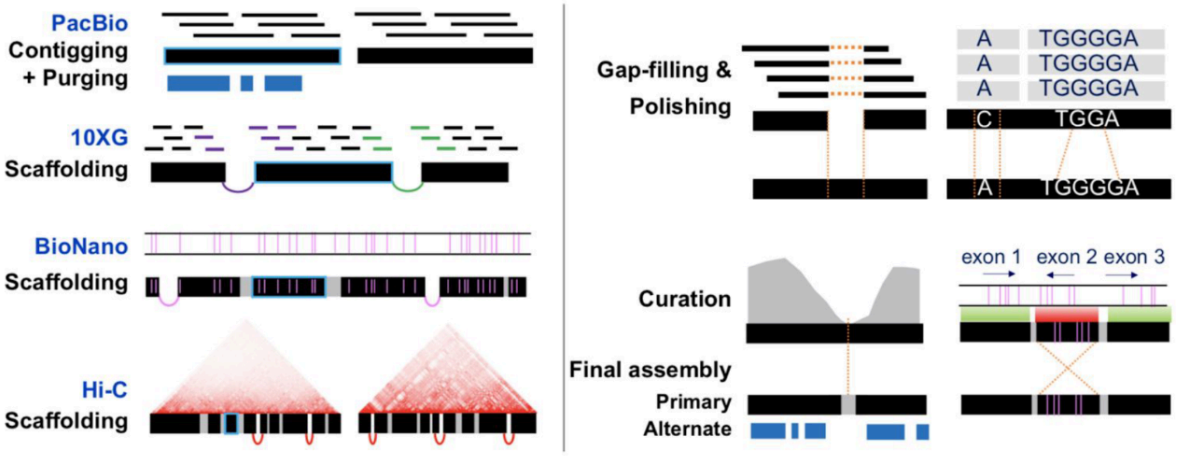
This is a Bioinfo

This is a Bioinfo class

in LaunchED

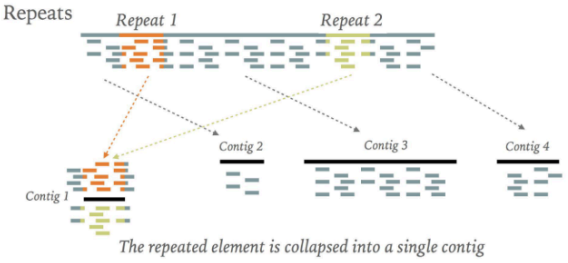
This is a Bioinfo class in LaunchED

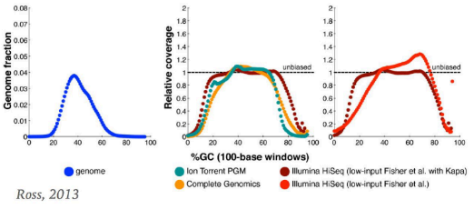
Example genome project workflow

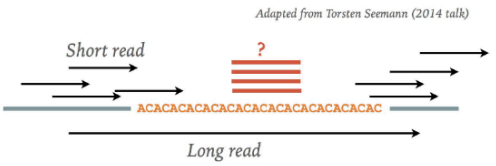


Hurdles

- Heterozygosity
- Sequencing errors
- Repeats
- Low complexity genomic regions
- Base composition and sequencing bias



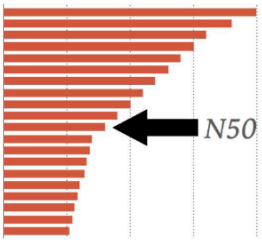




Assembly metrics

- total length
- number of sequences (contigs and scaffolds)
- average length (contigs and scaffolds)
- largest/smallest (contigs and scaffolds)
- N50 = X means 50% of the genome is in sequences larger than X
- NG50 (N50 scaled by the expected genome size)
- Gene content (% conserved core genes mapped)

N50 = what is the smallest contig at 50% of genome?



Scaffolding

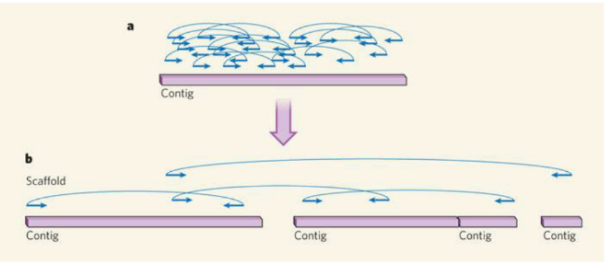
Goal: Order and orient the contigs into larger structure

Construct libraries of varying insert sizes

- Smaller size (2, 4 or 6 Kb), intermediate size (10-40 Kb), and libraries with large insert sequences (>100 Kb)
- Ends of these clones are sequenced, generating sequence reads.

Sources of evidence

- Mate-pair illumina libraries
- Fosmid ends
- Bacterial artificial chromosomes
- 10x Genomics linked reads
- Hi-C
- Optical maps



Gaps consisting of N (unknown) bases are inserted between contigs

(Pseudo) chromosome assignment

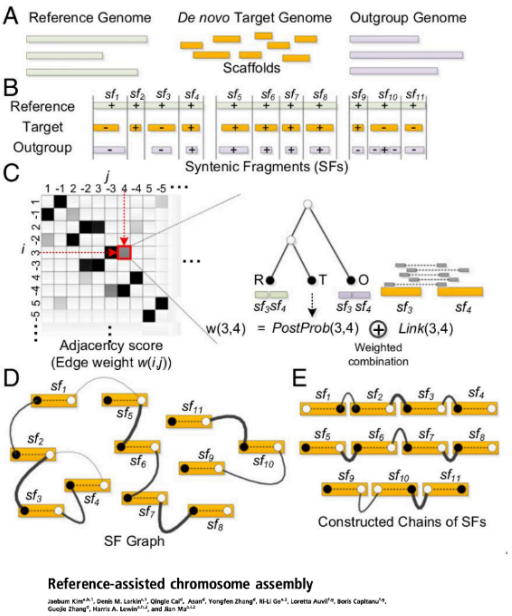
Goal: Assembly of chromosome-scale DNA fragments from scaffolds

Sources of evidence

- A close reference genome and/or outgroup genome
- Genetic markers or map

RACA software

- Input: Reference genome, de novo assembly, and one or more outgroups (A)
- Identify syntenic fragments (B)
- Calculate likelihood of adjacencies (C)



Bioinformatics Workflow

1. Raw Data Collection

1. FASTA (.fasta or .fa)

The **FASTA** format is used for representing nucleotide sequences (DNA or RNA) or protein sequences. Each sequence is preceded by a description line starting with a `>` symbol.

Example:

shell

CopyEdit

```
>Sequence_1
ATGCGTACGTTAGCTAGCTGACTG
>Sequence_2
GCTAGGTCAGTACGACTGA
```

In this example, each sequence starts with a description (e.g., `>Sequence_1`) followed by the sequence itself.

---

2. FASTQ (.fastq or .fq)

The **FASTQ** format is similar to FASTA, but it includes quality scores for each base in the sequence. It's commonly used to store high-throughput sequencing data. Each sequence is represented by four lines:

1. **Identifier line**: Begins with `@` symbol.
2. **Sequence line**: The nucleotide or protein sequence.
3. **Plus sign line (`+`)**: Sometimes followed by the sequence identifier again.
4. **Quality score line**: ASCII characters representing the quality score.

Example:

```
@Sequence_1
ATGCGTACGTTAGCTAGCTGACTG
+
IIIIIIIIIIIIIIIIIIIIIIII
@Sequence_2
GCTAGGTCAGTACGACTGA
+
IIIIIIIIIIIIIIIIIII
```

Here, `IIIIIIIIIIIIIIII` are the quality scores corresponding to each base in the sequence.

---

3. BAM (.bam)

The **BAM** (Binary Alignment Map) format is a binary version of the **SAM** (Sequence Alignment Map) format. BAM files store aligned sequence data in a compressed format, and they're used to represent the alignment of sequencing reads to a reference genome. BAM files are efficient for storage and faster for processing compared to their text-based SAM counterparts.

Example (SAM format for clarity):

css

CopyEdit

```
@SQ   SN:chr1    LN:249250621

r001  99    chr1  7     60    8S12M3S    =     37    39    AGCTGACAGTGA
      *     XS:A:R

r002  0     chr1  37    60    4S10M6S    =     73    39    AGCTGACTGACG
      *     XS:A:R
```

Here, `r001` and `r002` are sequence reads aligned to `chr1`, starting at position 7 and 37, respectively.

🔄 Why Sorting BAM Files Is Important

When sequencing reads are saved in a BAM file, they often aren't ordered in any useful way — usually by read name. But to analyze them properly, we usually need them **sorted by genomic position** (where they map in the genome).

- **Sorting is essential** because many tools expect reads to be in order by where they appear in the genome.
- Tools like **Samtools** and **Picard** are commonly used to sort BAM files.

---

4. VCF (.vcf)

The **VCF** (Variant Call Format) is used to represent variants (like SNPs, insertions, deletions) in a sequence compared to a reference genome. It includes metadata, column headers, and variant data. A **VCF (Variant Call Format)** file contains:

- Metadata about the file and software

Example:

shell

CopyEdit

```
##fileformat=VCFv4.2

##source=GenomeAnalysisTK

#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
```

```
1       10177   rs367896724     G       A       29.3    PASS
AC=1;AF=0.5;AN=2;DB;DP=14;FS=0.0;H2;H3

1       10235   rs367896725     G       T       30.6    PASS
AC=1;AF=0.5;AN=2;DB;DP=20;FS=0.0;H2
```

In this example, `rs367896724` is a known SNP with a reference base `G` and alternate base `A` on chromosome 1 at position 10177.

#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT sample

NC_045512.2 6512 . AGTT A 49.1 PASS ANN=A|disruptive_inframe_deletion|MODERATE|ORF1ab|GU280_gp01-|transcript|GU280_gp01|protein_coding|1/2|c.6248_6250delGTT|p.Ser2083_Leu2084delinsIle|6248/21291|6248/21291-|2083/7096|| GT:GQ:DP:AD:VAF:PL 1/1:46:128:0,128:1:49,49,0

NC_045512.2 8393 . G A 48.3 PASS ANN=A|missense_variant|MODERATE|ORF1ab|GU280_gp01|transcript-|GU280_gp01|protein_coding|1/2|c.8128G>A|p.Ala2710Thr|8128/21291|8128/21291|2710/7096|| GT:GQ:DP:AD:VAF:PL 1/1:44:189:0,189:1:48,45,0

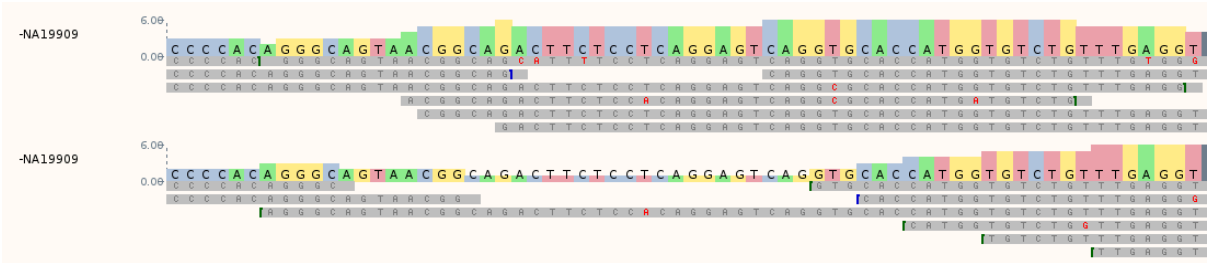
NC_045512.2 10029 . C T 52.1 PASS ANN=T|missense_variant|MODERATE|ORF1ab|GU280_gp01|transcript-|GU280_gp01|protein_coding|1/2|c.9764C>T|p.Thr3255Ile|9764/21291|9764/21291|3255/7096|| GT:GQ:DP:AD:VAF:PL 1/1:50:99:0,99:1:52,54,0

What is Variant Calling?

Variant calling is the process of finding genetic differences (variants) by comparing sequencing data to a reference genome.

Typical Workflow:

1. Sequence DNA (whole genome or exome) to get raw data (FASTQ files).
2. Align these sequences to a reference genome to create BAM or CRAM files.
3. Detect differences between aligned reads and the reference, and save these variants in a VCF file.



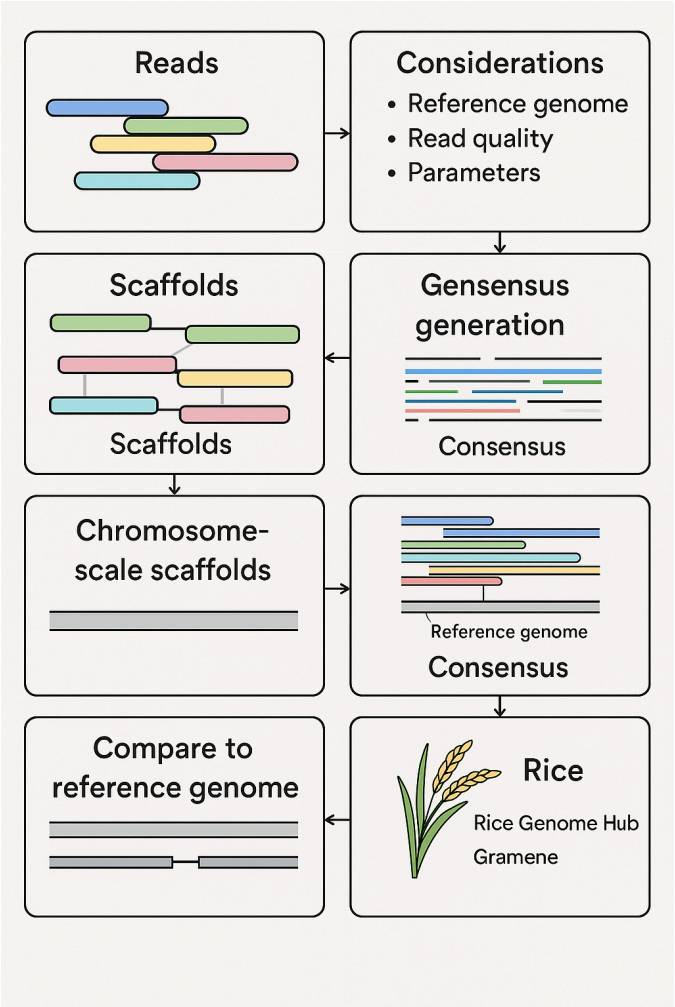
---

https://www.youtube.com/watch?v=UVGcPU8_aIU

https://www.youtube.com/watch?v=PwZMARY657A

https://www.youtube.com/watch?v=R8tqDTIkXOo

https://www.youtube.com/watch?v=l4BAfRekohk



https://www.youtube.com/watch?v=sFeK-25K5PE

The **computer requirements for performing NGS (Next-Generation Sequencing) analysis** depend on the **scale of data**, **type of analysis**, and **tools used**. Here's a breakdown for **small, medium, and large-scale** setups:

🖥️ Minimum Requirements (Small-scale / Education / Test data)

- **CPU**: 4 cores (e.g., Intel i5 or AMD Ryzen 5)
- **RAM**: 16 GB

- **Storage**: 500 GB SSD
- **OS**: Linux (Ubuntu preferred), macOS, or Windows with WSL
- **Use Case**: Single-end reads, small genomes (e.g., bacteria), tutorial-level datasets

---

💻 Recommended (Mid-scale / Standard Research Labs)

- **CPU**: 8–16 cores (Intel Xeon, AMD Ryzen 9)
- **RAM**: 64–128 GB
- **Storage**: 2–4 TB SSD (or hybrid SSD + HDD)
- **GPU**: Not essential, but helpful for some visualization
- **Use Case**: Whole exome/genome, RNA-Seq of human/mouse/rice, medium-size datasets

---

🖥️ High-performance (Large-scale / Core Facility / Institute-level)

- **CPU**: 32–64+ cores (dual processors)
- **RAM**: 256 GB – 1 TB
- **Storage**: 10–100 TB or more (fast I/O via RAID, NAS, or parallel file systems)
- **Cluster / HPC**: Slurm, Sun Grid Engine, or cloud (AWS, GCP, Azure)
- **Use Case**: Whole-genome sequencing, multiple samples, metagenomics, pangenomics

---

✅ Other Key Considerations

Component	Notes
Backup & Archival	Use external NAS, cold storage, or cloud backups
Software	Tools like BWA, Bowtie2, GATK, SAMtools, STAR, HISAT2, etc.

Parallelization	Use multi-threading or job schedulers for large pipelines
Cloud Option	AWS EC2, GCP Compute Engine, DNA Nexus, Terra.bio
I/O speed	Fast read/write speeds are crucial (especially for BAM/VCF handling)

🧬 Python Script Guidelines for NGS Analysis

🗂️ 1. Structure Your Project

```
ngs_analysis/
├── data/              # Raw and processed data files
├── scripts/           # Python scripts
├── results/           # Output files
├── logs/              # Log files
├── config.yaml        # Config file with paths and parameters
├── environment.yml    # Conda environment
└── main.py            # Entry point script
```

🗒️ 2. Sample Python Script Template

```
# scripts/align_reads.py

import subprocess
import argparse
import logging
from pathlib import Path

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

def align_reads(reference, reads_1, reads_2, output_bam):
    logging.info("Indexing reference genome...")
    subprocess.run(["bwa", "index", reference], check=True)

    logging.info("Aligning reads...")
    sam_file = output_bam.replace(".bam", ".sam")
    subprocess.run([
        "bwa", "mem", "-t", "4", reference, reads_1, reads_2
    ], stdout=open(sam_file, 'w'), check=True)

    logging.info("Converting SAM to BAM and sorting...")
```

```
    subprocess.run(["samtools", "view", "-Sb", sam_file], stdout=open("temp.bam", "wb"), check=True)
    subprocess.run(["samtools", "sort", "temp.bam", "-o", output_bam], check=True)
    subprocess.run(["samtools", "index", output_bam], check=True)

    Path("temp.bam").unlink()
    Path(sam_file).unlink()

    logging.info("Alignment complete.")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Align paired-end reads using BWA and SAMtools")
    parser.add_argument("-r", "--reference", required=True)
    parser.add_argument("-1", "--reads1", required=True)
    parser.add_argument("-2", "--reads2", required=True)
    parser.add_argument("-o", "--output", required=True)
    args = parser.parse_args()

    align_reads(args.reference, args.reads1, args.reads2, args.output)
```

⚙️ 3. Best Practices

Tip	Why It's Important
Use `argparse`	Enables command-line use and flexibility
Use `subprocess.run()`	Runs external tools (e.g., BWA, samtools) safely
Use `Pathlib`	Better path handling across OS
Add `logging`	Keeps logs instead of using print()
Write modular functions	Easier to reuse/test
Include `README.md`	Explain usage, dependencies
Use `Snakemake` or `Nextflow` later	For scaling up multi-sample analysis

📦 4. Recommended Libraries

Task	Library

File handling	`pandas`, `pyyaml`, `pathlib`
Bioinformatics tools	`Biopython`, `pysam`, `pyvcf`
Plotting/visualization	`matplotlib`, `seaborn`
Workflow management	`Snakemake`, `Click`, `Luigi`

🔁 5. Common NGS Analysis Tasks You Can Script in Python

- FASTQ QC parsing (e.g., parsing FastQC output)
- Read alignment wrapper (BWA, Bowtie2)
- BAM file handling and filtering (via `pysam`)
- Variant filtering and annotation (e.g., using `pyvcf`)
- Coverage calculation and visualization
- Report generation (HTML, PDF)

🧩 Other Genome Assembly Tools and Platforms

🔬 1. Geneious Prime

- **Type**: Commercial desktop software
- **Strengths**: Intuitive GUI, suitable for NGS assembly, primer design, annotation

- **Best for**: Labs looking for a polished, drag-and-drop alternative to DNASTAR
- **Link**: https://www.geneious.com

🧬 2. CLC Genomics Workbench (QIAGEN)

- **Type**: Commercial software with GUI + scripting
- **Strengths**: End-to-end workflows from read QC to variant calling and annotation
- **Best for**: Researchers with large datasets or clinical genomics
- **Link**: https://digitalinsights.qiagen.com

☁️ 3. BaseSpace (Illumina)

- **Type**: Cloud-based analysis platform
- **Strengths**: Seamless integration with Illumina sequencers, drag-and-drop apps
- **Best for**: Labs using Illumina instruments
- **Link**: https://basespace.illumina.com

🧠 4. Terra.bio (by Broad Institute)

- **Type**: Cloud-based platform
- **Strengths**: Workflow Description Language (WDL)-based scalable pipelines, GATK, Cromwell support
- **Best for**: Large-scale genomics (cancer, rare diseases, cohort studies)
- **Link**: https://terra.bio

🧪 5. Galaxy Europe / UseGalaxy.eu

- **Type**: European instance of Galaxy with more bioinformatics tools pre-installed
- **Strengths**: Stronger computing infrastructure for free public use, wide tool availability
- **Link**: https://usegalaxy.eu

🧱 6. iPlant/CyVerse Discovery Environment

- **Type**: Cloud-based, designed for plant genomics
- **Strengths**: Virtual machines + pipelines for large genome projects
- **Best for**: Plant genome assembly, annotation, gene expression
- **Link**: https://cyverse.org

🧬 7. DFAST (for Prokaryotic Annotation)

- **Type**: Online and standalone tool
- **Strengths**: Simple bacterial genome annotation with curated databases
- **Link**: https://dfast.nig.ac.jp

🧷 Summary Table

Platform/Tool	Type	Best Use Case	GUI Support	Free/Open?
DNASTAR	Desktop	Commercial labs, small genomes	✔️	❌
Galaxy	Web	Flexible workflows, open research	✔️	✔️

Geneious Prime	Desktop	Intuitive multi-function NGS suite	✔️	❌
CLC Genomics Workbench	Desktop	Large clinical and research projects	✔️	❌
BaseSpace	Cloud	Illumina data pipelines	✔️	❌ (free tier limited)
Terra.bio	Cloud	Scalable workflows for big data	❌ (workflow-based)	✔️
CyVerse	Cloud	Plant genomics, RNA-seq	✔️	✔️
DFAST	Web/Desktop	Bacterial annotation	✔️	✔️

🧪 Quality Control and Trimming

🔍 FastQC

- **Purpose**: Quality control for raw sequencing data
- **Link**: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

✂️ Cutadapt

- **Purpose**: Adapter trimming and quality filtering
- **Link**: https://cutadapt.readthedocs.io/

✂️ Trimmomatic

- **Purpose**: Flexible read trimming tool for Illumina NGS data
- **Link**: http://www.usadellab.org/cms/?page=trimmomatic

🧬 Alignment

🧷 BWA (Burrows-Wheeler Aligner)

- **Purpose**: Align short reads to a reference genome
- **Link**: http://bio-bwa.sourceforge.net/

🚀 STAR (Spliced Transcripts Alignment to a Reference)

- **Purpose**: Fast aligner for RNA-seq data
- **Link**: https://github.com/alexdobin/STAR

🎯 Bowtie2

- **Purpose**: Fast aligner for short reads, including gapped alignment
- **Link**: http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

---

📂 BAM File Processing

🧰 Samtools

- **Purpose**: Tools for working with SAM/BAM/CRAM files
- **Link**: http://www.htslib.org/

🍩 Picard

- **Purpose**: A set of Java-based tools for manipulating BAM files

- **Link**: https://broadinstitute.github.io/picard/

🧬 Variant Calling

🧪 Bcftools

- **Purpose**: Variant calling and manipulating VCF/BCF files
- **Link**: http://samtools.github.io/bcftools/

🧬 GATK (Genome Analysis Toolkit)

- **Purpose**: Comprehensive toolkit for variant discovery
- **Link**: https://gatk.broadinstitute.org/

⚖️ FreeBayes

- **Purpose**: Haplotype-based variant detector for NGS data
- **Link**: https://github.com/freebayes/freebayes

🔬 VCF Analysis and Visualization

🧬 IGV (Integrative Genomics Viewer)

- **Purpose**: Desktop application for visualizing genomics data
- **Link**: https://igv.org/

🧭 UCSC Genome Browser

- **Purpose**: Web-based browser for exploring reference genomes and annotations
- **Link**: https://genome.ucsc.edu/

Here's a **simplified version** of your text with **links intact** and the core ideas made clearer and more concise:

🧬 Common Variant Calling Tools

Frequently used tools for calling variants include:

- **BCFtools**
- **GATK HaplotypeCaller**
- **DeepVariant**
- **Strelka2**
- **SAMtools**
- **GraphTyper**
- **Nanopolish**

> For short-read data (e.g. Illumina), **GATK HaplotypeCaller**, **DeepVariant**, or **GraphTyper** are recommended.
> **DeepVariant** also works well with **PacBio** long-read data.
> For **Oxford Nanopore** data, use **Nanopolish**.

For a benchmark comparison, see the study by Barbitoff et al.

🔁 Variant Calling Workflow

Key steps in variant calling:

1. Identify candidate variant sites
2. Determine reference and alternate alleles
3. Assign genotypes (homozygous or heterozygous)
4. (For SARS-CoV-2, all variants are homozygous)
5. Perform quality control
6. Output as a **VCF** (Variant Call Format) file

🔍 *[Click here to enlarge Figure 2 – Overview of variant calling process]*

🧬 Joint vs Single-Sample Variant Calling

- **Single-sample calling** analyzes each BAM file individually.
- **Joint calling** analyzes all BAMs together, improving:
    - Detection of rare variants
    - Filtering of false positives
    - Distinction between true homozygous reference and missing data

📘 Learn more in this GATK article on joint calling

📄 Variant Call Format (VCF)

A **VCF** file summarizes called variants and includes:

- Genomic position
- Reference and alternate alleles
- Genotype info
- Optional annotations

🔗 VCF format details by SAMtools
📘 Detailed explanation by Danecek et al. (2021)

🔍 *[Click here to enlarge Figure 3 – Example VCF output]*

🧬 Variant Annotation

Variant annotation adds biological meaning to called variants, such as:

- Protein impact
- Population frequency

- Functional region (e.g. exonic, intronic)
- Known identifiers (e.g. dbSNP)

A popular tool for this is **snpEff**.

🔍 *[Click here to enlarge Figure 4 – Annotated VCF with snpEff]*

Mastering variant calling takes **practice, experience**, and a strong understanding of both **biology and computation**. This overview provides the **big picture**, but each step contains layers of complexity worth exploring deeply.
