Bi5444 Analysis of sequencing data

Introduction to NGS pipeline

Eva Budinska

Aims for today

- Introduce the general NGS analysis pipeline and touch (almost) all parts of analysis in order to get the general idea
- Explain how the raw read files are created and what is their format

What we learned so far about NGS data analysis…

1. There are tens to hundreds of different algorithms/SW solutions available for analysis of NGS data
2. There is no "one and the best" way to perform an analysis. The final selection/pipeline you use largely depends on:
   - Your experiment hypothesis and sample type
   - The latest review of similar methods for the analysis step you just read
   - The accessibility and comprehensibility of the algorithm/SW solution (in other words, never use something you do not understand!)
   - The compatibility of inputs/outputs between algorithms from different steps

....

3. There is NO tool, that can perform every analysis from the very beginning to the end -> like in any other analysis

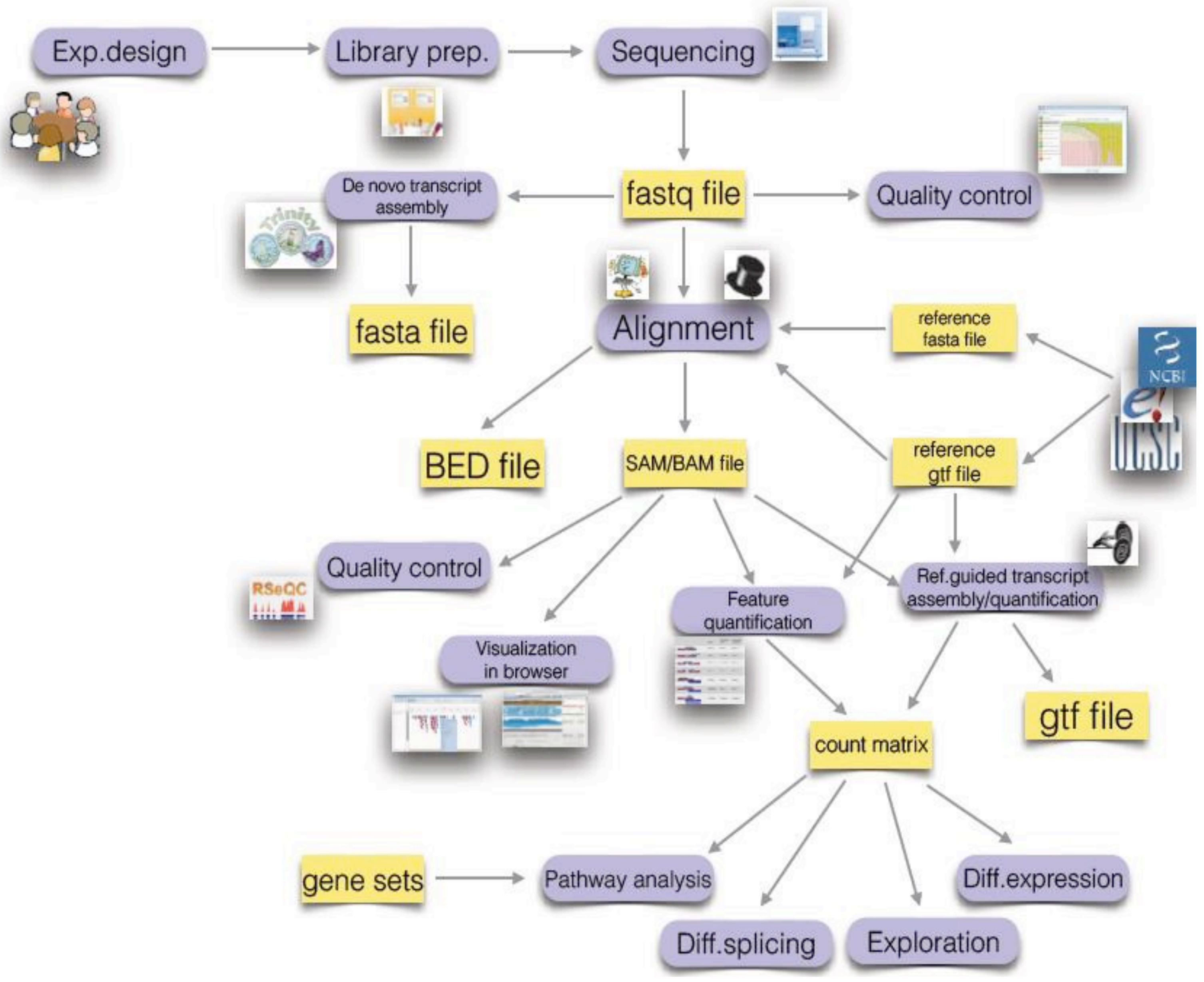
4. Most of the tools are command-line based – many work the best under Linux or MacOS environments

5. Windows is the worst environment you can use

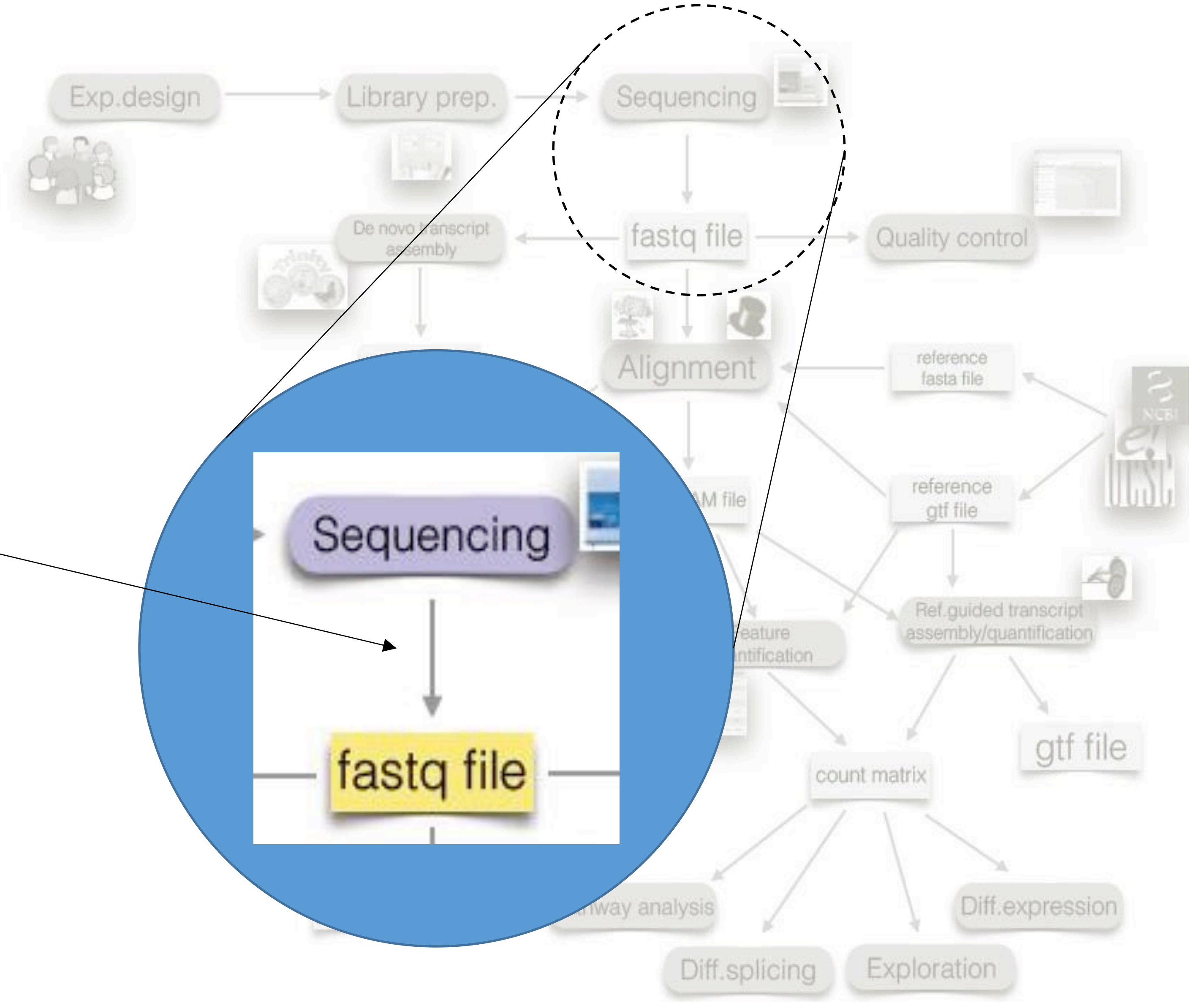
6. The tools are written in many different languages: Python, Java, Perl, C++, R…

7. You do not need to become expert in programming in these languages, however, you need to understand how the tools are installed and used

The NGS analysis pipeline

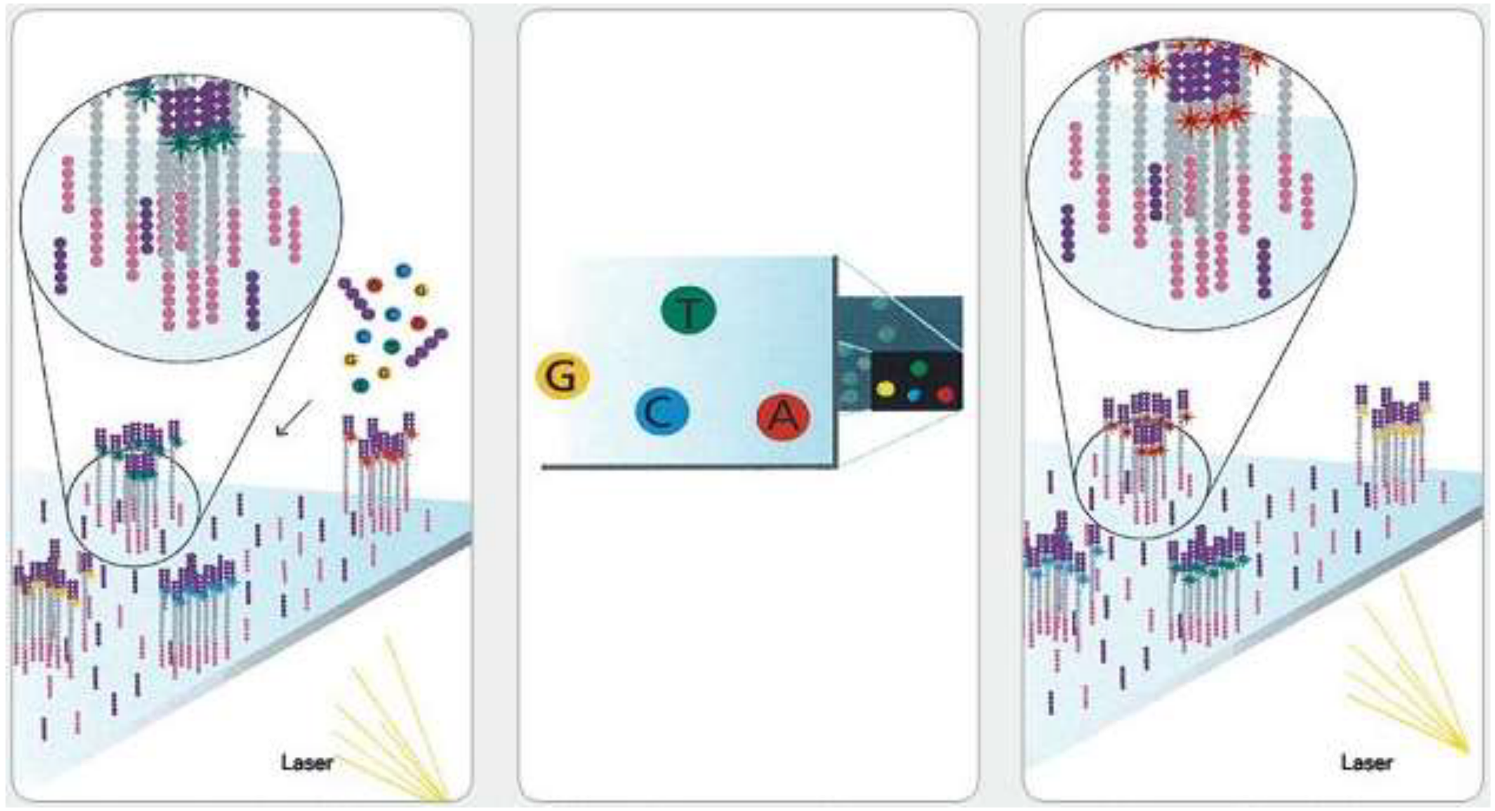


Step 0:
base calling (image analysis) + base quality control



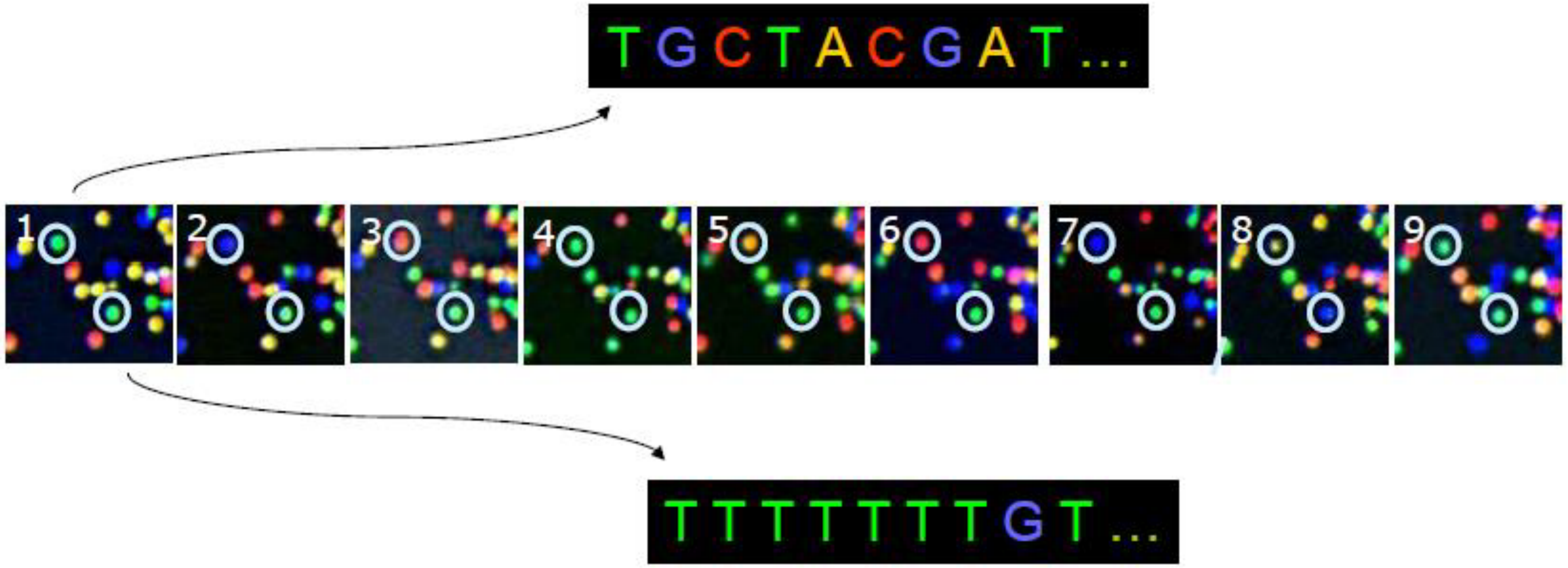
Sequencing by Synthesis - Fluorescently labeled Nucleotides (Illumina)

- During the process, **clusters of same sequences** are created

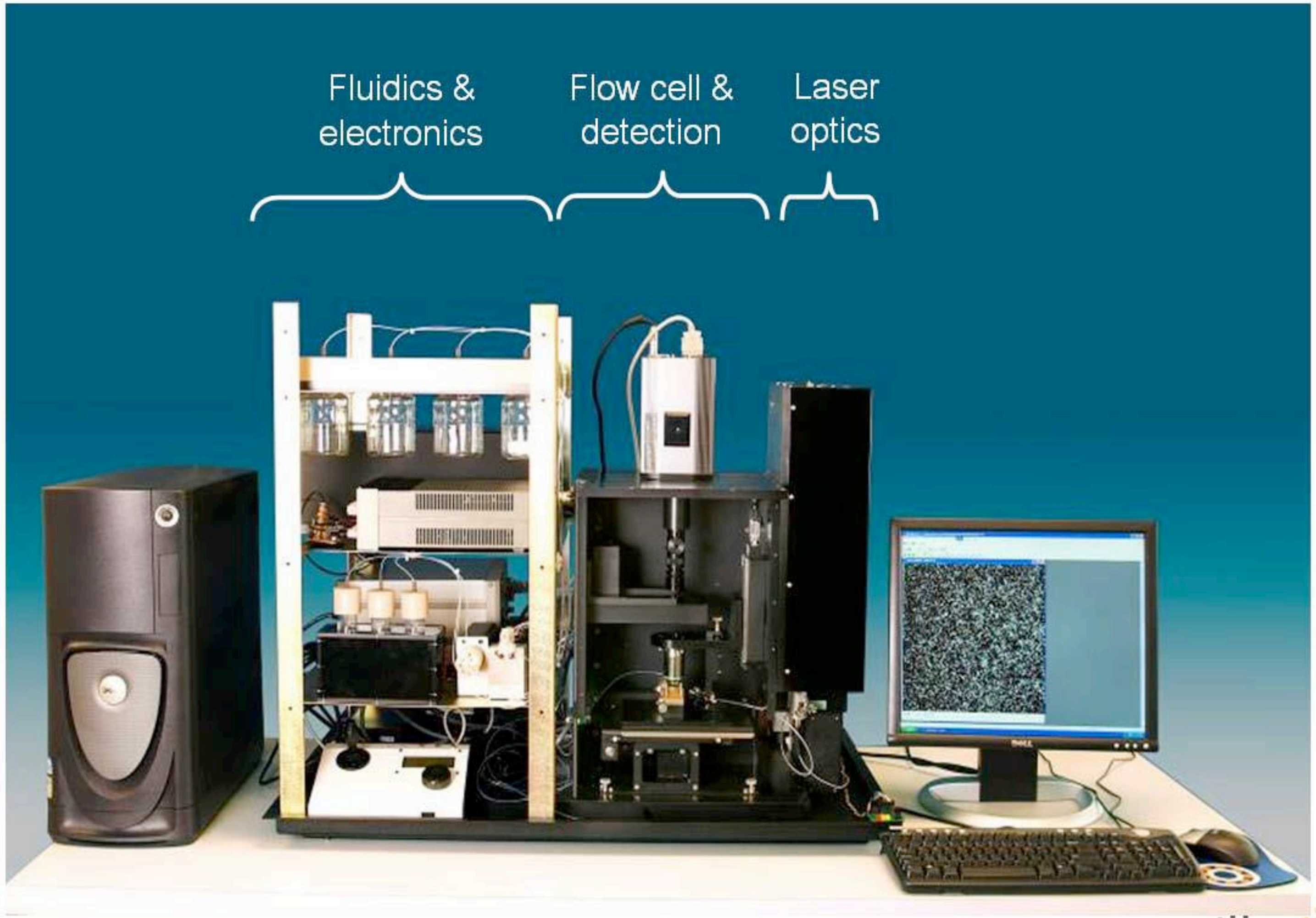


Step 0: base calling (image analysis)

- The identity of each base of a cluster is read off from **sequential images**
- One cycle -> one image

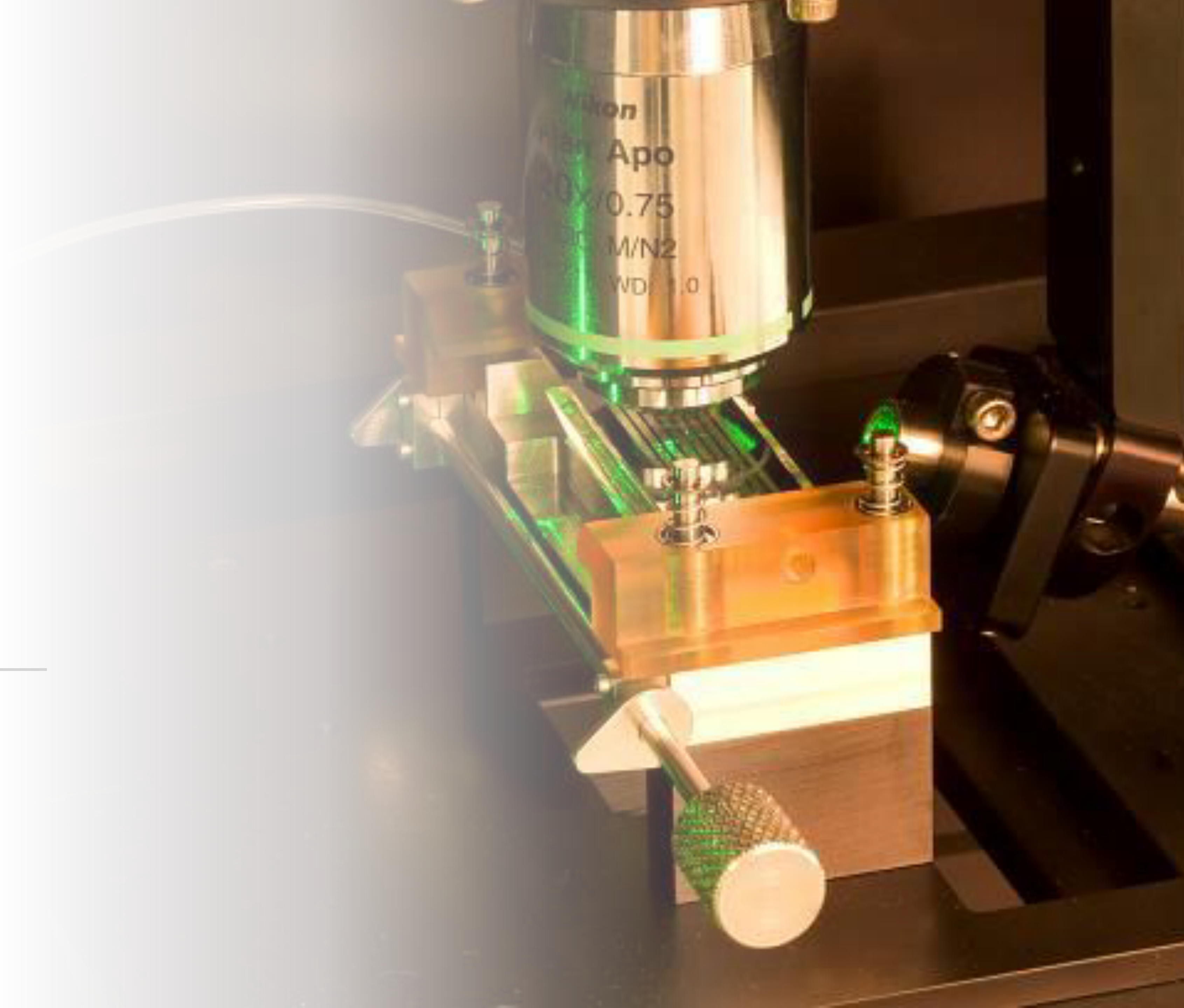


Instrument without Covers

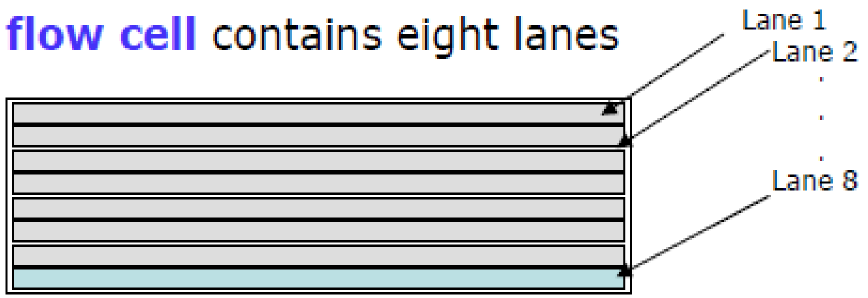


illumina

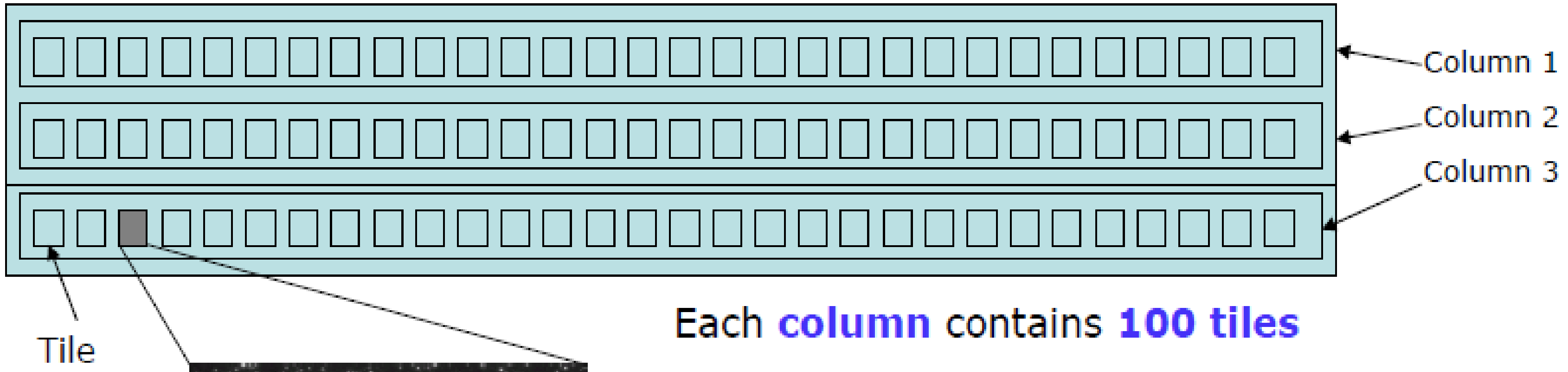
Flow-cell imaging



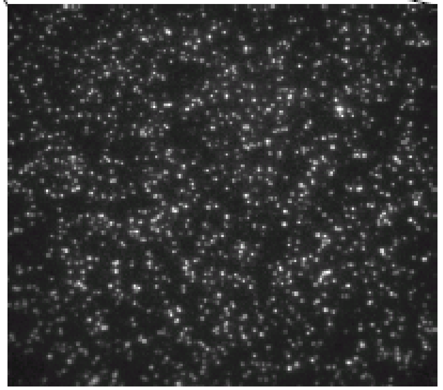
A **flow cell** contains eight lanes



Each **lane/channel** contains **three columns** of tiles



Each **column** contains **100 tiles**



20K-30K Clusters

350 X 350 µm

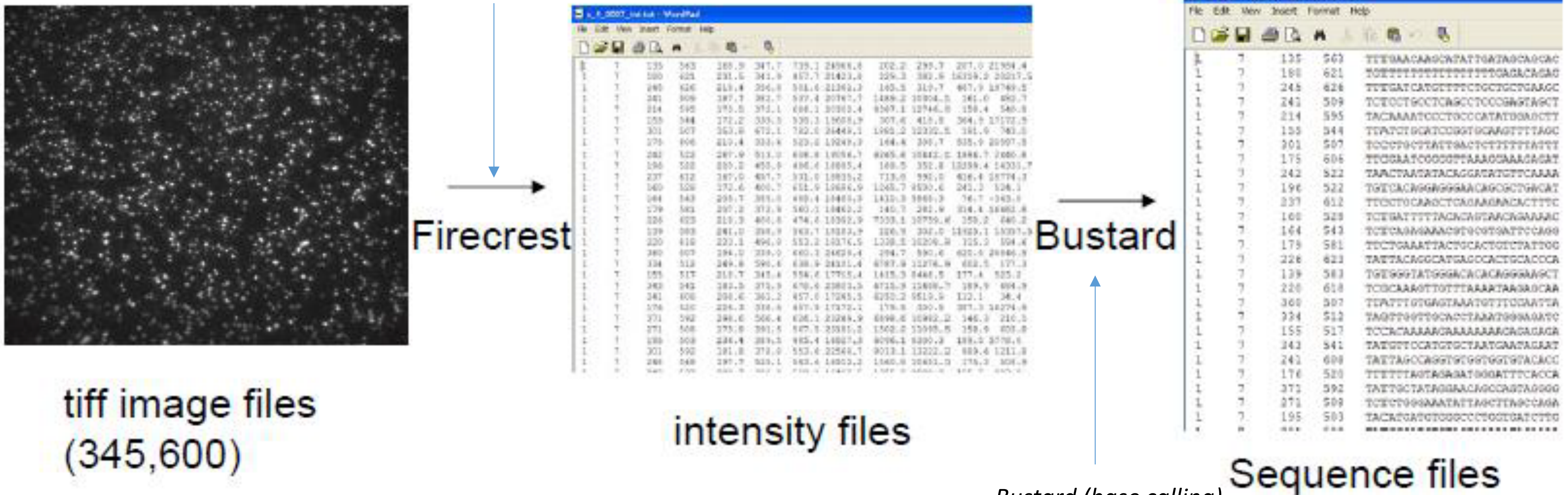
Each tile is imaged four times per cycle – one image per base.

345,600 images for a 36-cycle run

Getting the sequences from clusters

- Illumina pipeline

Firecrest (image analysis)
Locates clusters and calculates intensity and noise

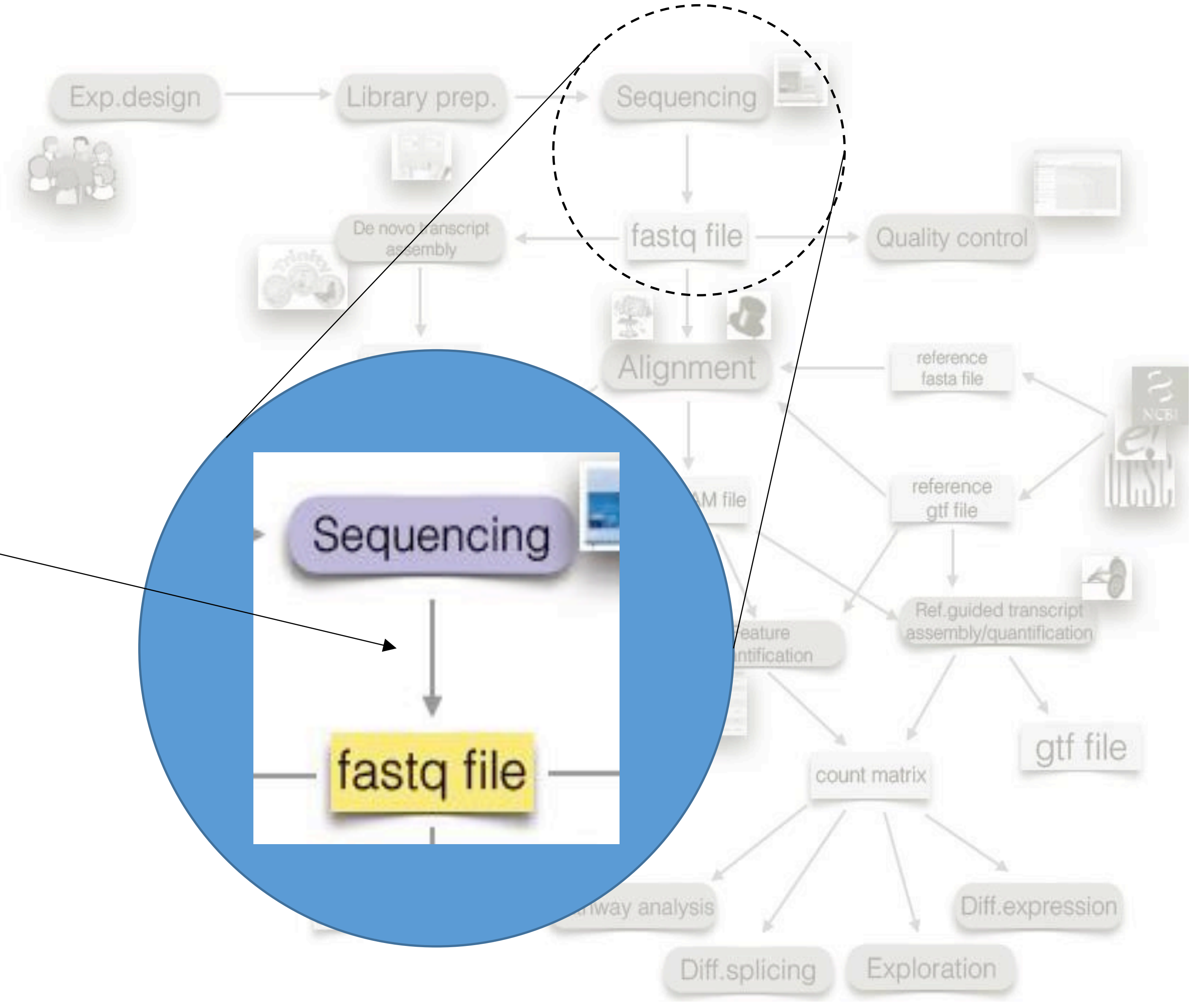


Bustard (base calling)
Deconvolutes signal and corrects for cross-talk, phasing

Image analysis data output

- 100 tiles per lane, 8 lanes per flow cell, 36 cycles
- 4 images (A,G,C,T) per tile per cycle = 115,200 images
- Each tiff image is ~ 7 MB = 806,400 MB of data
- 1.6 TB per 70 nt read, 3.2 TB for 70 nt paired-end read
- Most technologies are erasing intensities as they are sequencing, because of a too high amount of data

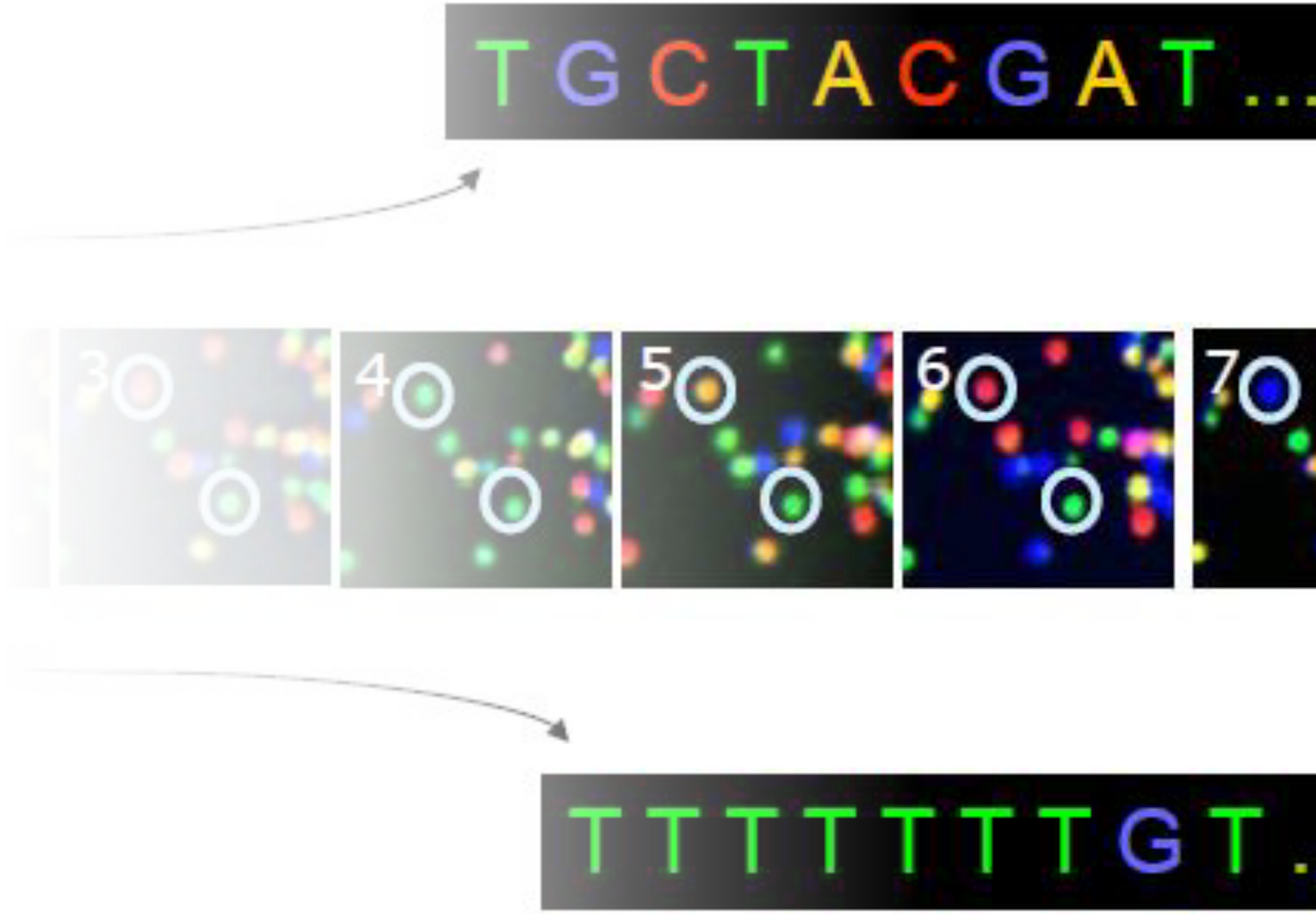
Step 0:
base calling
(image analysis)
+ **base quality control**



Base call quality control

- Quality control (QC) of each base call is automatically performed by the sequencing platform
- In other words: *For each letter in a read, we estimate the probability of it being erroneous (P).*

- QC per base is specialized for each platform – each platform must solve challenges unique to the underlying sequencing technology



Alternative base calling algorithms

- Multiple algorithms were proposed reporting improvements in sequence quality with respect to the manufacturer's algorithms
- See some reviews:
  - Cacho, Ashley & Smirnova, Ekaterina & Huzurbazar, Snehalata & Cui, Xinping. (2015). A Comparison of Base-calling Algorithms for Illumina Sequencing Technology. Briefings in bioinformatics. 17. 10.1093/bib/bbv088.
  - Ledergerber, Christian & Dessimoz, Christophe. (2011). Base-calling for next-generation sequencing platforms. Briefings in bioinformatics. 12. 489-97. 10.1093/bib/bbq077.

The PHRED score

$$Q_{phred} = - 10 \times \log_{10} P(\mathbf{error})$$

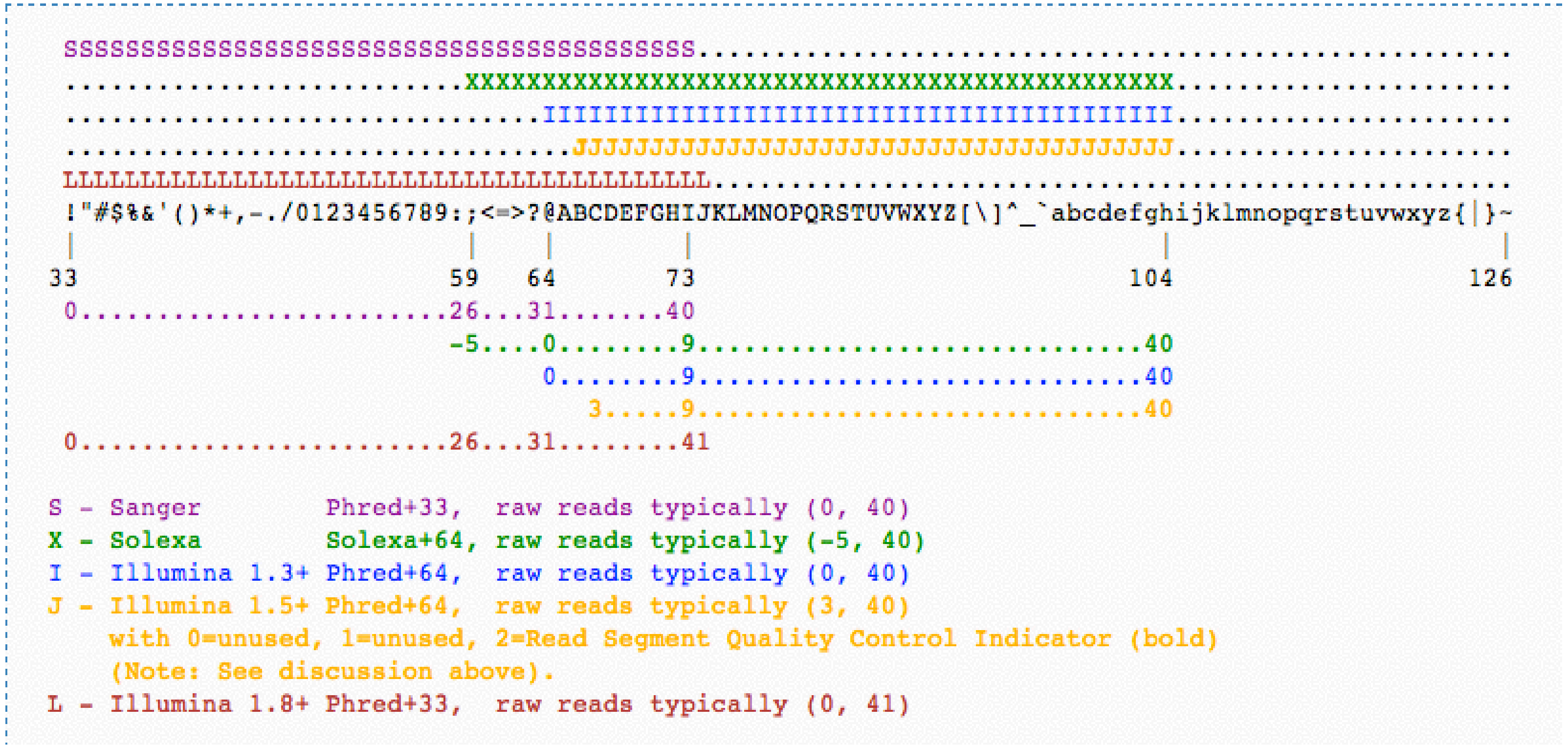
- The *Phred* quality score is the negative ratio of the error probability to the reference level of *P* = 1 expressed in Decibel (dB).
- The **error estimate** is based on **statistical model** providing measure of **certainty** of each base call in addition to the nucleotide itself
- These statistical models base their error estimate on:
  - Signal intensities from the recorded image
  - Number of the sequencing cycle
  - Distance to other sequence colonies
- *Phred* score is recoded using ASCII in fastq file

Phred score	Probability of incorrect base call	Base call
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10 000	99.99%
50	1 in 100 000	99.999%
60	1 in 1 000 000	99.9999%

Phred score encoding in ASCII

https://en.wikipedia.org/wiki/FASTQ_format

```
@D7MHBFN1:202:D1BUDACXX:4:1101:1340:1967 1:N:0:CATGCA
NATCTTCGGATCACTTTGGTCAAATTGAAACGATACAGAGAAGATTGTAAGTAACAATATTTACCAAGGTTCGAGTCATACTAACTCGTTGTCCTATAGT
+
#1=DDFFFHHHHHJJJJJJHIJIJJJIJIJGIIIJJJJJJIIJIJJJHIIFGIIIIJJJJJIIEHJIIHHGFFF@?ADFEDDEDCDDBDDBDCDDDDEC
```



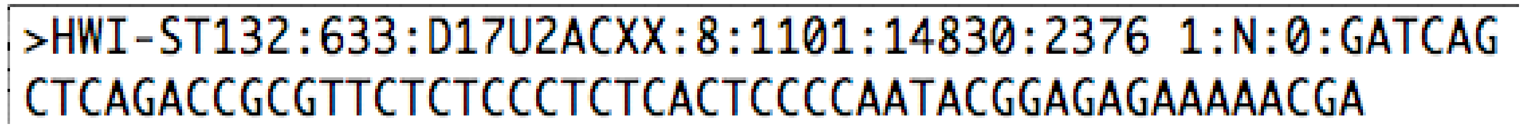
```
  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....................................................
  ..........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......................
  ...............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII......................
  .................................JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ......................
  LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL....................................................
  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
  |                         |    |        |                              |                     |
 33                        59   64       73                            104                   126
  0........................26...31.......40
                           -5....0........9.............................40
                                 0........9.............................40
                                    3.....9.............................40
  0........................26...31........41

 S - Sanger        Phred+33,  raw reads typically (0, 40)
 X - Solexa        Solexa+64, raw reads typically (-5, 40)
 I - Illumina 1.3+ Phred+64,  raw reads typically (0, 40)
 J - Illumina 1.5+ Phred+64,  raw reads typically (3, 40)
    with 0=unused, 1=unused, 2=Read Segment Quality Control Indicator (bold)
    (Note: See discussion above).
 L - Illumina 1.8+ Phred+33,  raw reads typically (0, 41)
```

FASTA and FASTQ formats

- The reads obtained from the sequencer are typically stored in **fasta** (just the sequences) or **fastq** (sequences + QC measure) format files.
- For **paired-end** reads, we usually obtain **two files**.
- **Reads** are *not* generally grouped by strand, only **by the order in which they were sequenced**.

FASTA format

- General format to represent sequences
- **Two lines per sequence** (read)
  - ID line (starting with >)
  - Sequence line
- Typical file extension: `.fa` or `.fasta`

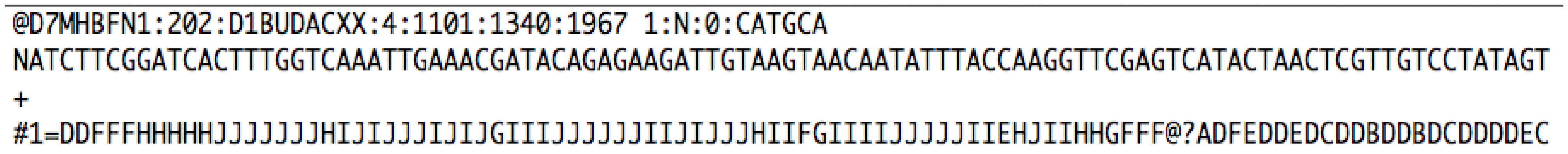


```
>HWI-ST132:633:D17U2ACXX:8:1101:14830:2376 1:N:0:GATCAG
CTCAGACCGCGTTCTCTCCCTCTCACTCCCCAATACGGAGAGAAAAACGA
```

- HWI-ST132 - unique instrument name
- 633 - run ID
- D17U2ACXX - flowcell ID
- 8 - flowcell lane
- 1101 - tile number within lane
- 14830 - x-coordinate of cluster within tile
- 2376 - y-coordinate of cluster within tile
- 1 - member of pair (1 or 2). Older versions: /1 and /2
- Y/N - whether the read failed quality control (Y = bad)
- 0 - none of the control bits are on
- CATGCA - index sequence (barcode)

FASTQ format

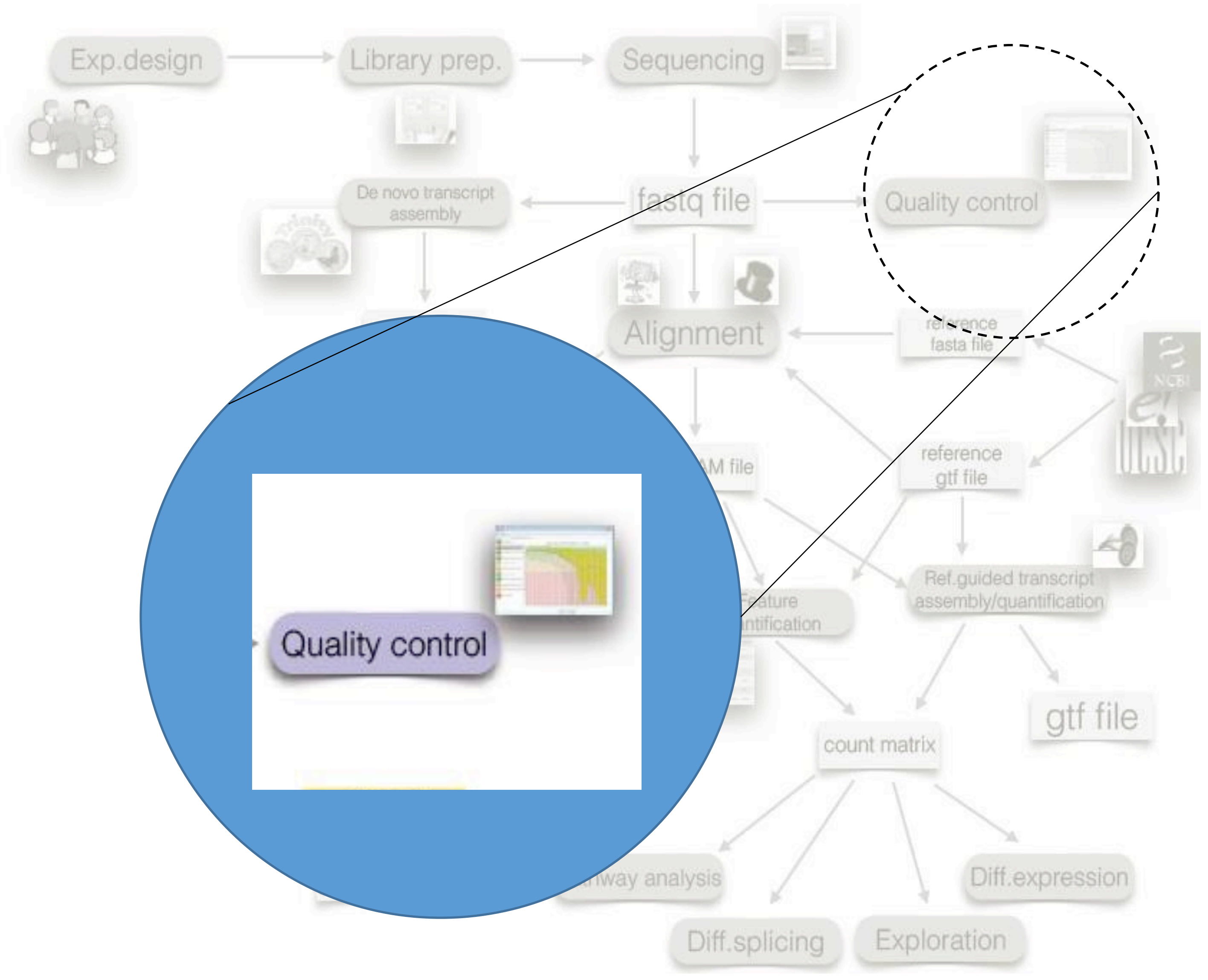
- Combines sequence and base call quality information.
- Typical file extension: `.fastq`



```
@D7MHBFN1:202:D1BUDACXX:4:1101:1340:1967 1:N:0:CATGCA
NATCTTCGGATCACTTTGGTCAAATTGAAACGATACAGAGAAGATTGTAAGTAACAATATTTACCAAGGTTCGAGTCATACTAACTCGTTGTCCTATAGT
+
#1=DDFFFHHHHJJJJJJHIJIJJJIJIJGIIIJJJJJJIIJIJJJHIIFGIIIIJJJJJIIEHJIIHHGFFF@?ADFEDDEDCDDBDDBDCDDDDEC
```

- Four lines per sequence (read):
  - ID (starting with @)
  - Sequence line
  - Another ID line (starting with +)
  - Base qualities (one for each letter in the sequence)

Step 1:
Read quality control and data filtering



Step 1: **Read quality control** and data filtering

- Uses the output file with information about the quality of base calls (.fastq)
- First step in the pipeline that **deals with actual sequencing data** in base or color space

- Several metrics are evaluated, not all of them use the Phred score information:
  - Distribution of quality scores at each sequence, Sequence composition, Per-sequence and per-read distribution of GC content, Library complexity, Overrepresented sequences

- Initial overview – already in base calling SW
- More quality overview – SW solutions `SolexaQA, FastQC`

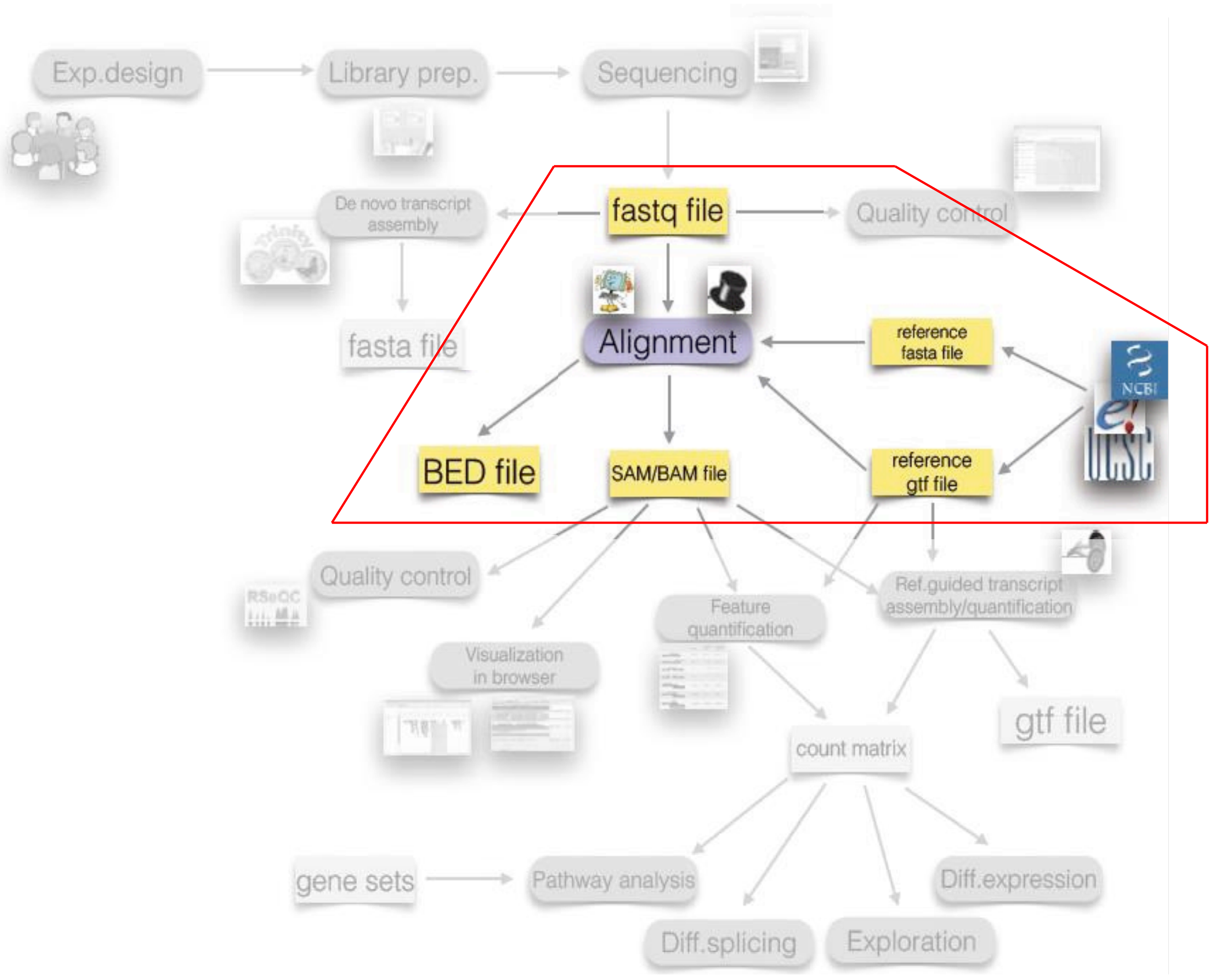
Step 1: Read quality control and **data filtering**

- Based on the quality measures, we decide to remove low quality bases and reads

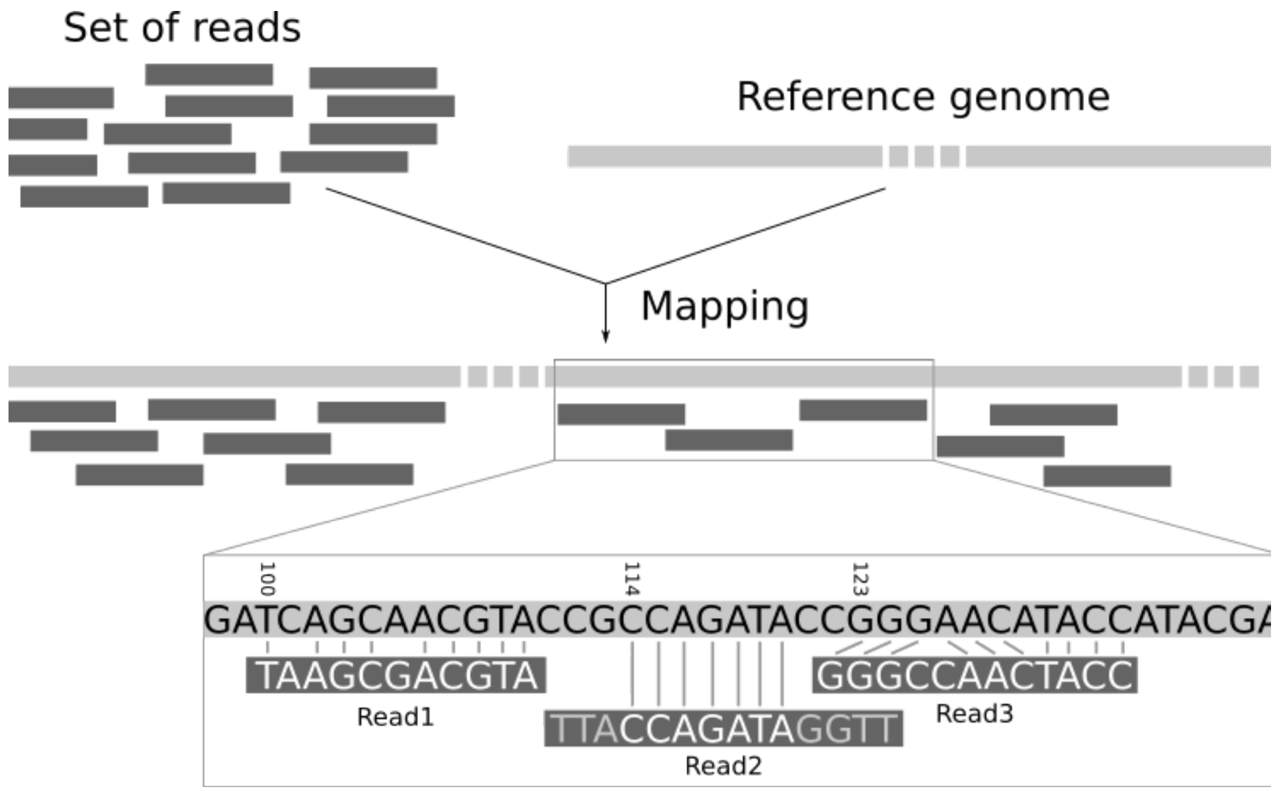
- **Trimming** – removes low quality or unwanted bases from reads, thus shortening them. Is applied to increase the number of mappable reads.

- **Filtering –** removes whole reads that do not meet quality standards (e.g. too short etc)

Step 2: Alignment (mapping)



Step 2: Alignment (mapping)



- To know, where the **short reads** (in our filtered .fastq file) come from (which part of the genome or transcriptome do they represent) they need to be (in most instances) aligned to a **reference sequence**

Reference sequence

- The reference sequence can be a genome, a transcriptome or a collection of specific sequences.
- Typically, the reference sequence(s) is given in a `.fa` or `.fasta` file
- An alternative is the GTF (gene transfer format) - stores gene structure
- BED format (designed for annotation tracks in genomic browsers)

(we will learn about where to get the reference sequences in one of the next lectures)

Step 2: Alignment (mapping)

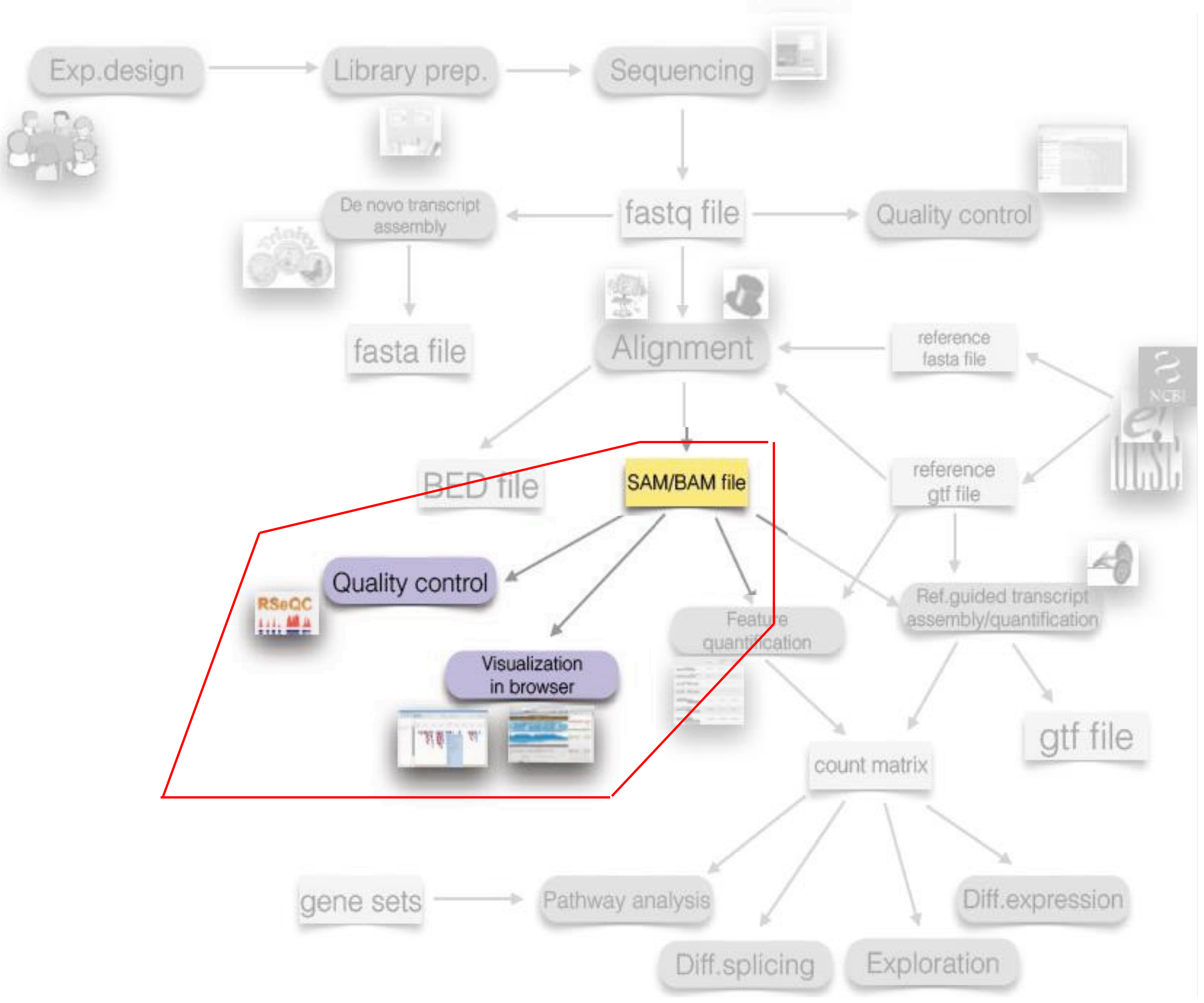
```
GTGCTCGCTGACACAGAAAGTTCGGCA
CTCAGACA
11111111
```

- Intuitively an easy task
- However, trying all the possible options (alignments), is very time consuming!
- Efficient algorithms (**aligners**) exist

- The result of mapping is stored by many algorithms in the **Sequence alignment/map (SAM) format**

- We will talk about mapping a in one of the future lectures

Step 3:
Post-alignment QC and visualization



Step 3: **Post-alignment QC** and visualization

- Necessary in order to see the **efficiency of the alignment**.

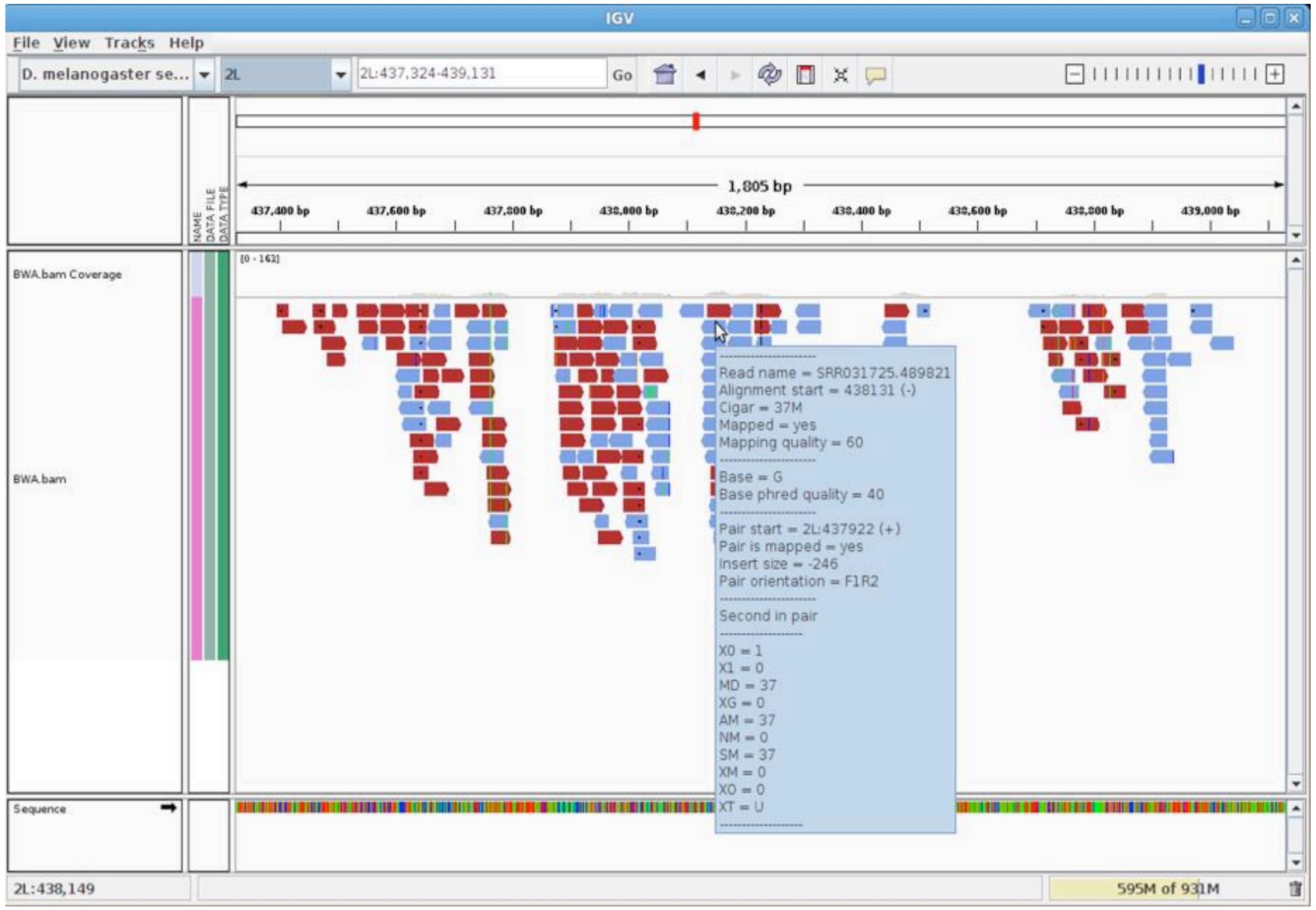
- During the alignment, not all the reads are aligned – but what proportion?
- If they were aligned – are there any errors?
- How well is the reference genome covered?

- Important in determining whether:
  - we can proceed with the analysis or some pre-processing needs to be done
  - we need to possibly redo the alignment
  - or we need to realign those unaligned reads

Step 3: Post-alignment QC and **visualization**

Allows us to get a detailed look on the **coverage** of a **given** region.

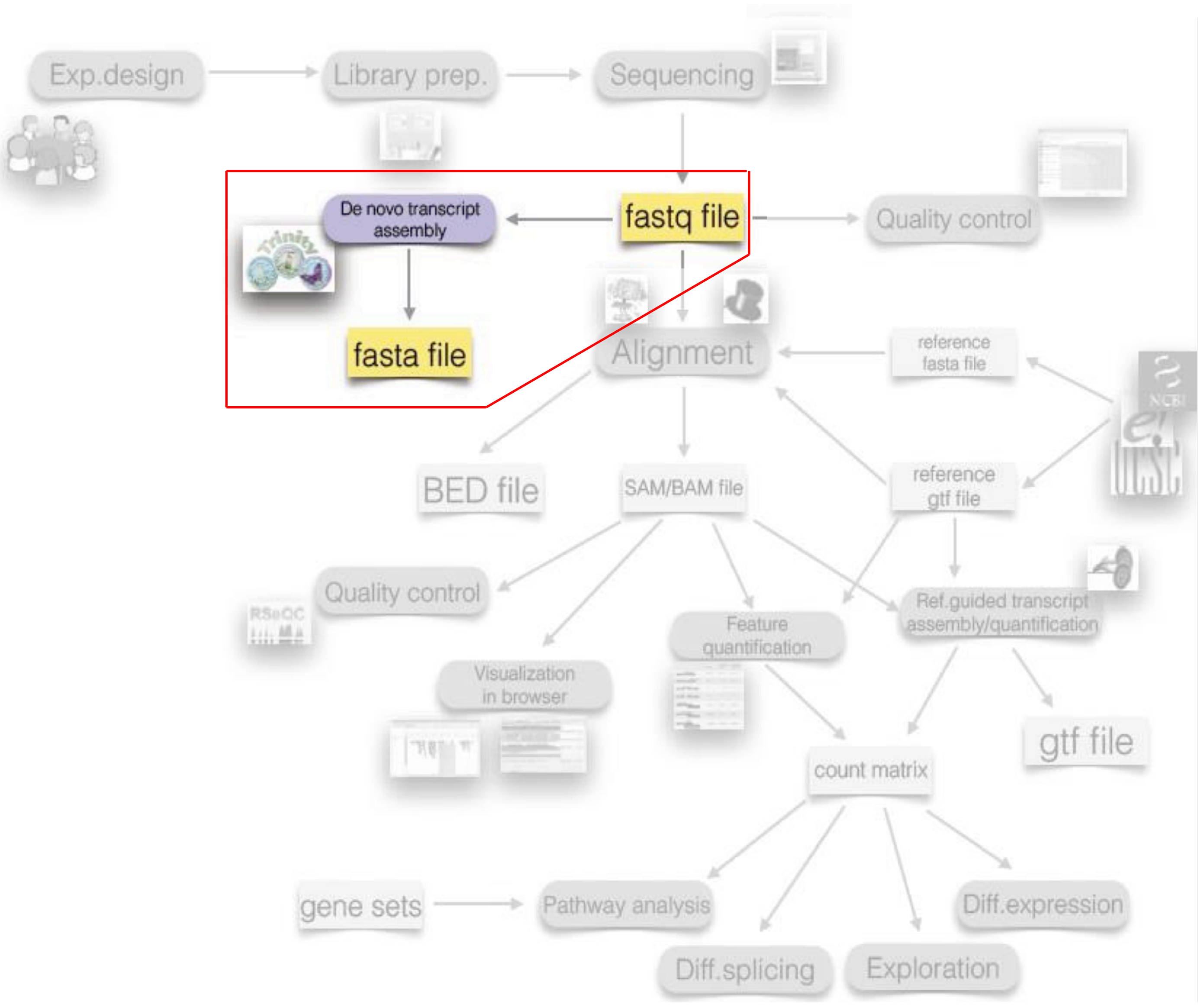
IGV genome browser



```
--------------------
Read name = SRR031725.489821
Alignment start = 438131 (-)
Cigar = 37M
Mapped = yes
Mapping quality = 60
--------------------
Base = G
Base phred quality = 40
--------------------
Pair start = 2L:437922 (+)
Pair is mapped = yes
Insert size = -246
Pair orientation = F1R2
--------------------
Second in pair
-----------------
X0 = 1
X1 = 0
MD = 37
XG = 0
AM = 37
NM = 0
SM = 37
XM = 0
XO = 0
XT = U
-----------------
```

http://software.broadinstitute.org/software/igv/

Alternative step 2: Genome/transcript (de-novo) assembly

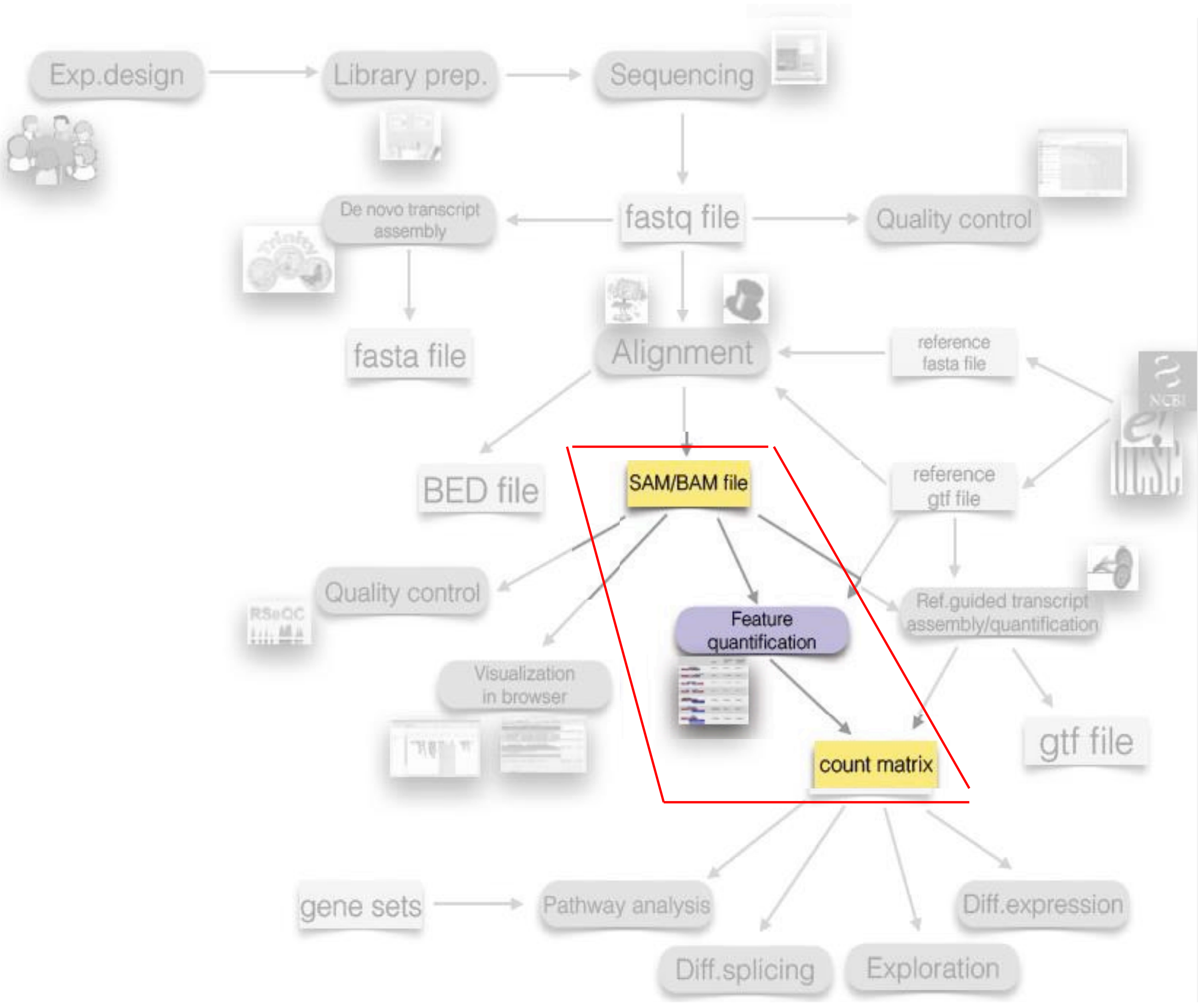


Alternative step 2: Genome/transcript (de-novo) assembly

- When the reference sequence does not exist

- Alignment is dependent on the existence of reference sequence.
- However – sometimes this reference does not exist! – ***de novo*** genome assembly – we need to practically create the reference genome.
- The assembly is sometimes preferred in order to identify large structural rearrangements even when reference genome is known. In transcriptomics we can use it to detect **alternative splicing** events

Step 4:
Feature detection (quantification)



Step 4: **Feature detection (quantification)**

- Creates the final table with read counts for further statistical analyses

- A feature of interest differs based on the experiment:
  - gene, exon, intron… (WGS, WES)
  - transcript, isoform (RNA-seq)
  - variant - SNP, insertion, deletion, CNV - (WGS, WES, targeted sequencing)
  - promotor sequence (ChIP-Seq)

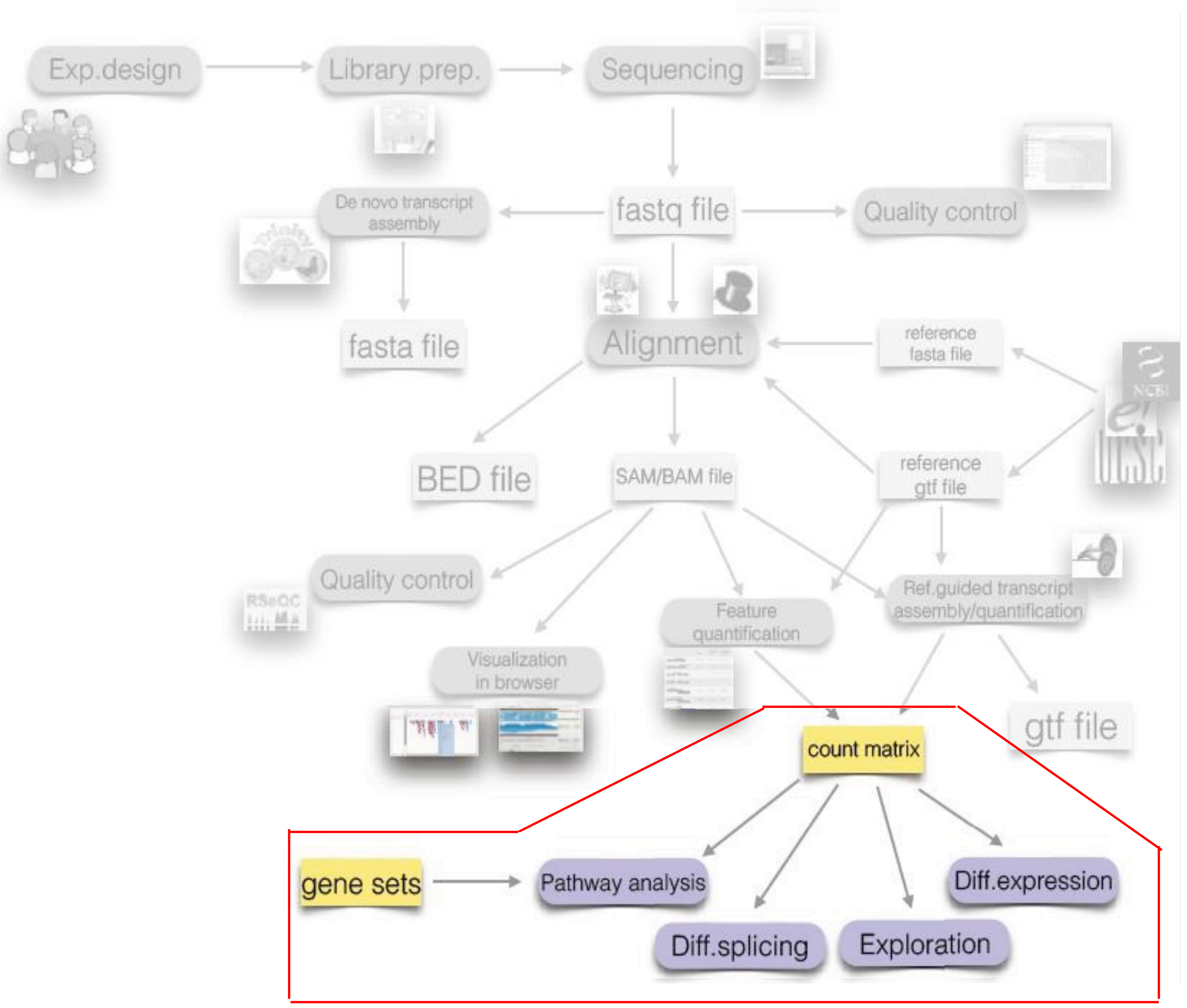
- In **transcriptomics** NGS experiments, the emphasis is on **quantification** of known transcripts (unless the aim is to get new isoforms) – we quantify the abundance of the RNA.
- In **genomic** NGS experiments, the emphasis is more on the **detection** of structural changes (the quantification is the % of alternative alleles found).

Step 4: **Feature detection (quantification)**

- Creates the final table with read counts for further statistical analyses

- The final output of this step is always a matrix with:
  - **Information** about the feature (ID, name, variant…)
  - **Quantification** of this feature in each of the samples

Step 5:
Statistical data analysis



Step 5: **Statistical data analysis**

- The final matrix is input to four main analysis types:

Group comparison (between groups of samples or groups of features)
•Differential gene expression / splicing
•Differential variants detection

Group discovery (within samples or features)

•Clustering of patients into unknown subtypes based on their sequencing profiles
•Searching for genes with similar expression

Group prediction (usually for samples)

•Finding genes for diagnosis…

Special analyses: pathway analysis, construction of gene networks, analysis of survival, …

Analyzing and writing the code

You cannot NGS analyze without scripting (writing of commands) and keeping track of it !

Why scripting and keeping track?

1. **Reproducibility** (you or anyone else must be able to reproduce your analysis step by step)
2. **Time saving** (if something in your data changes, you can simply run all the scripts again on new dataset)
3. **No one-size-fits-all solutions** (i. no program can cover all the possible combinations of tools; ii. it is easier to change something in the existing script than write it all over again)
4. **Batch-execution of commands** (high-performance cloud and cluster computing requires commands in batches)

Conversion, conversion, conversion

- … be prepared for never-ending format conversions …

(wrong format of input file is usually one of the most common reasons of errors)

- SAM to BAM,
- BAM to SAM,
- sorted SAM to BAM,
- BAM to sorted SAM,
- BAM to indexed BAM,
- aligned, realigned, indexed, ….

Examples of scripts for different analysis steps

- Quality control (using prinseq)

```
$ perl prinseq-lite.pl -fastq file1.fastq -graph_data file1.gd -out_good null -out_bad
null
```

- Alignment (using bwa)

```
$ bwa sampe -P hg19.fa file1.sai file2.sai \ file1.fastq file2.fastq > file_bwa.sam
```

- Variant calling

```
$ java -jar GenomeAnalysisTK.jar -T HaplotypeCaller \
-R hg19.fa \
-I file1.bam -I file2.bam -I file3.bam -I file4.bam \
-stand_call_conf 30 -stand_emit_conf 10 \
-o output.raw.snps.indels.vcf
```

Small first example

1. Download the toy example .fastq file *http://www.ebi.ac.uk/ena/data/view/SRR014849*

```
$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR014/SRR014849/SRR014849_1.fastq.gz
```

2. Unzip the file

```
$ gunzip SRR014849_1.fastq.gz
```

3. See the header of the file:

```
$ head SRR014849_1.fastq
```

4. Calculate total number of lines

```
$ wc -l SRR014849_1.fastq
```

5. Calculate total number of reads

```
$ wc -l SRR014849_1.fastq | awk '{print $1/4}'
```