Variant calling
Part 2

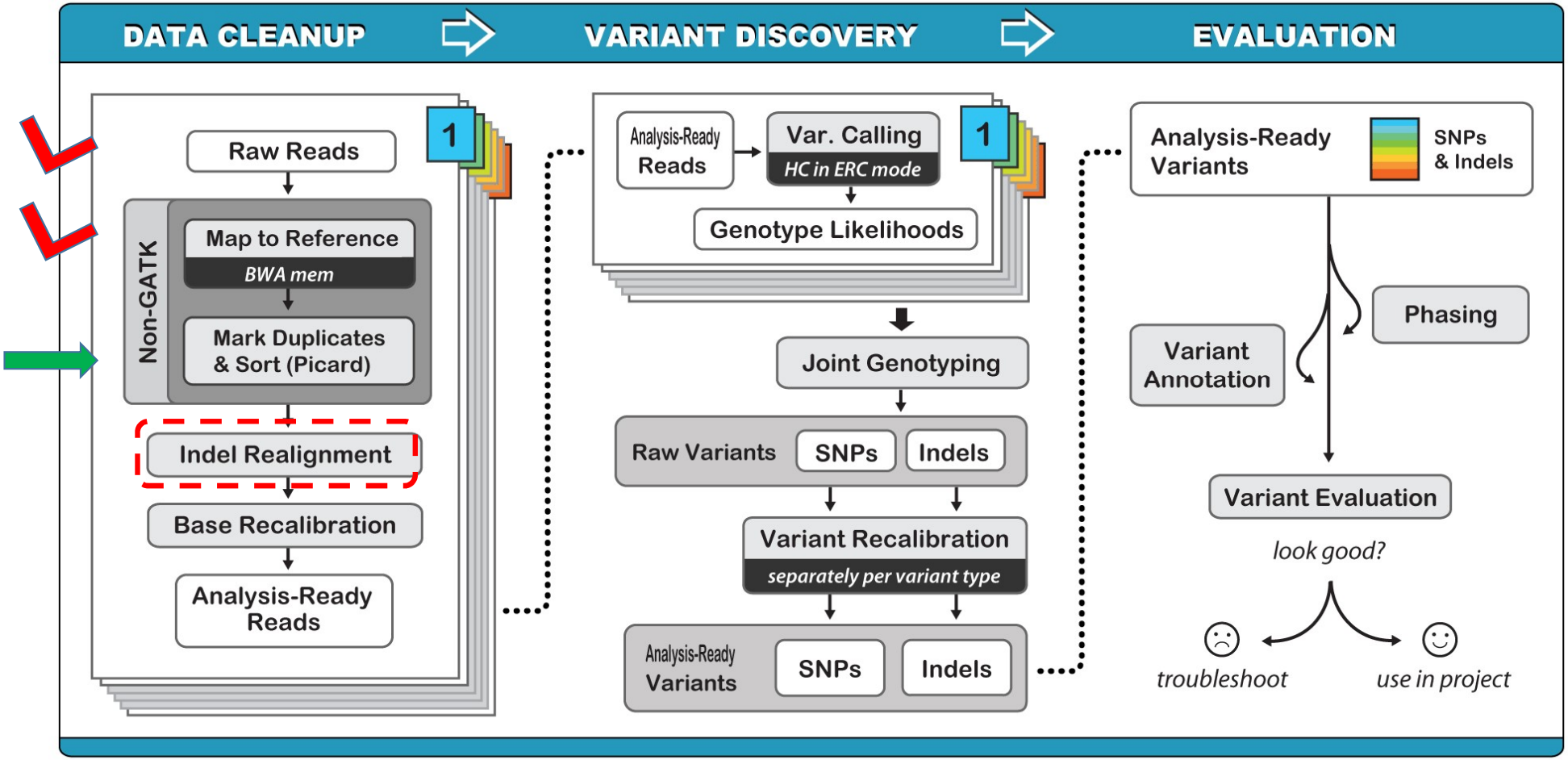
Robert Bukowski, Qi Sun, Minghui Wang
Bioinformatics Facility
Institute of Biotechnology

Slides: http://cbsu.tc.cornell.edu/lab/doc/Variant_workshop_Part2.pdf

Exercise instructions: http://cbsu.tc.cornell.edu/lab/doc/Variant_exercise2.pdf

Workshop contact: bukowski@cornell.edu

“Best Practices” for DNA-Seq variant calling



Duplicate reads (fragments)

- **Optical duplicates:** (Illumina) generated when a single cluster of reads is part of two adjacent tiles' on the same slide and used to compute two read calls separately

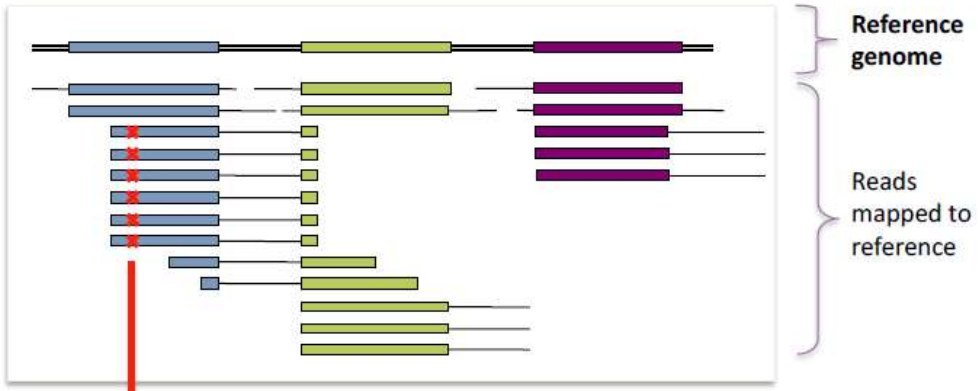
  - Very similar in sequence (except sequencing errors).
  - Identified where the first, say, 50 bases are identical between two reads and the read's coordinates are close

- **Library duplicates (aka PCR duplicates):** generated when the original sample is pre-amplified to such extent that initial unique targets are PCR replicated prior to library preparation and will lead to several independent spots on the Illumina slide.

  - do not have to be adjacent on the slide
  - share a very high level of sequence identity
  - align to the same place on reference
  - identified from alignment to reference

Why duplicates are bad for variant calling

✖ = sequencing error propagated in duplicates



FP variant call (bad)

After marking duplicates, the GATK will only see :

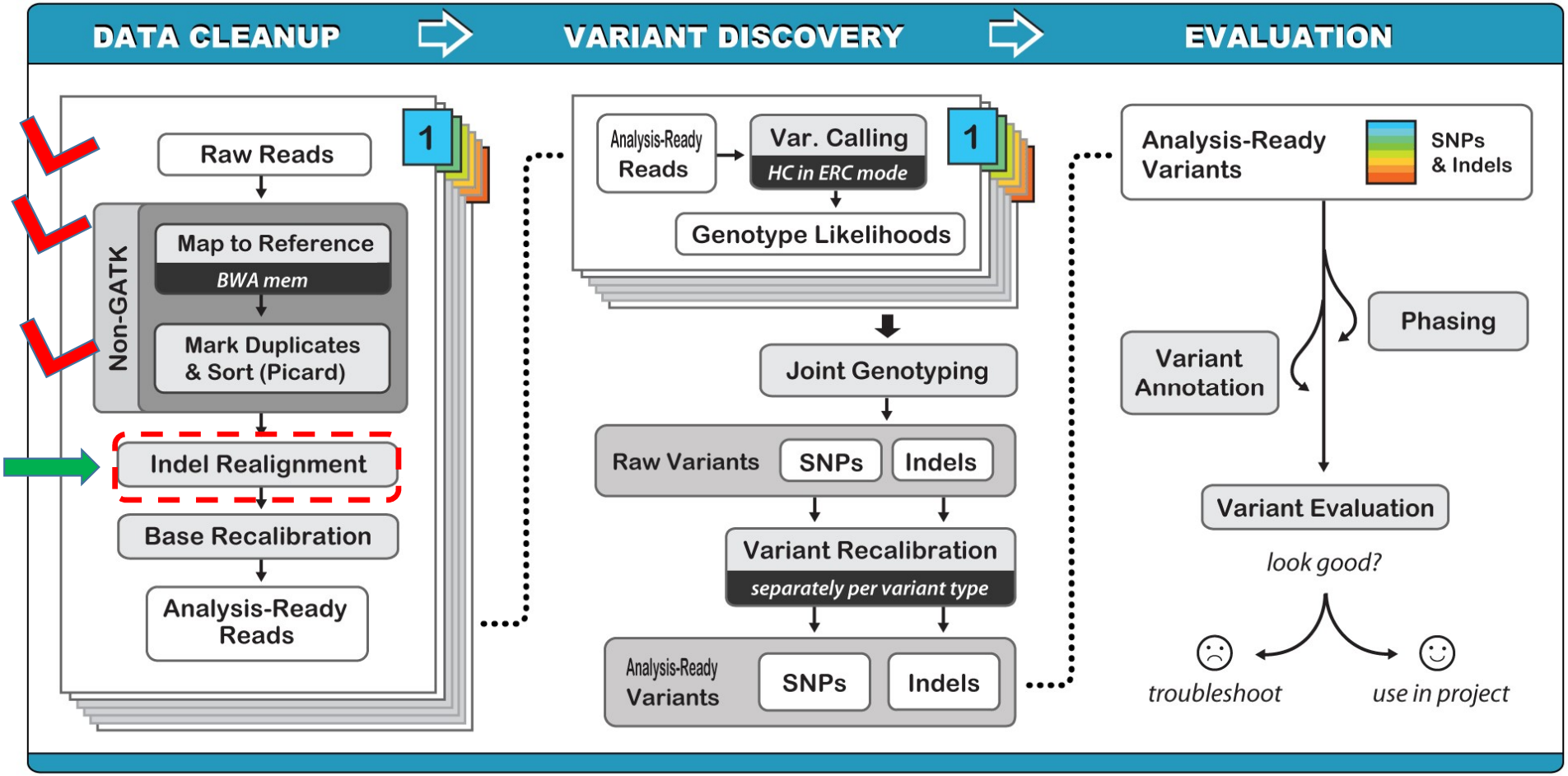
… and thus be more likely to make the right call

Removing (marking) duplicates with PICARD

```
java -jar $PICARDDIR/picard.jar \
MarkDuplicates \
INPUT=sample1.sorted.bam \
OUTPUT=sample1.sorted.dedup.bam \
METRICS_FILE=sample1.sorted.dedup.metrics.txt
```

- The metrics file will contain some stats about the de-duping
- In the resulting BAM file, only one fragment from each duplicate group survives unchanged, other duplicate fragments are given a flag 0x400 and will not be used downstream.
- Optimally, detection and marking of duplicate fragments should be done **per library**, i.e., over all read groups corresponding to a given library.
- In practice, often sufficient to do it per lane (read group).

“Best Practices” for DNA-Seq variant calling



Ambiguity of alignment at indel sites

Reference `CTTTAGTTTCTTTT----CTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC`

```
CTTTAGTTTCTTTT----GCCGCTTTCTTTCTTTCTT
CTTTAGTTTCTTTT----GCCGCTTTCTTTCTTTCTT
```

Reads

```
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
```

For these reads, aligner preferred to make a few SNPs rather than insertion

For these reads, insertion was a better choice

But we can try to shift things around a bit:

Reference `CTTTAGTTTCTTTT----CTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC`

```
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTT
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTT
```

Reads

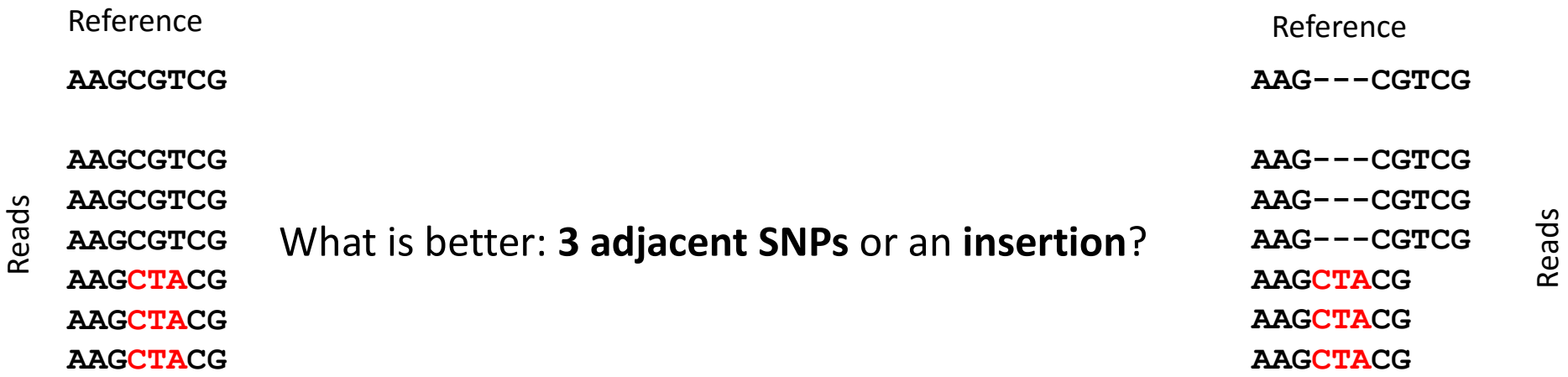
```
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
CTTTAGTTTCTTTTGCCGCTTTCTTTCTTTCTTTTTTTTTAAGTCTCCCTC
```

Aligner, like BWA, works on one read (fragment) at a time, does not see a bigger picture…)

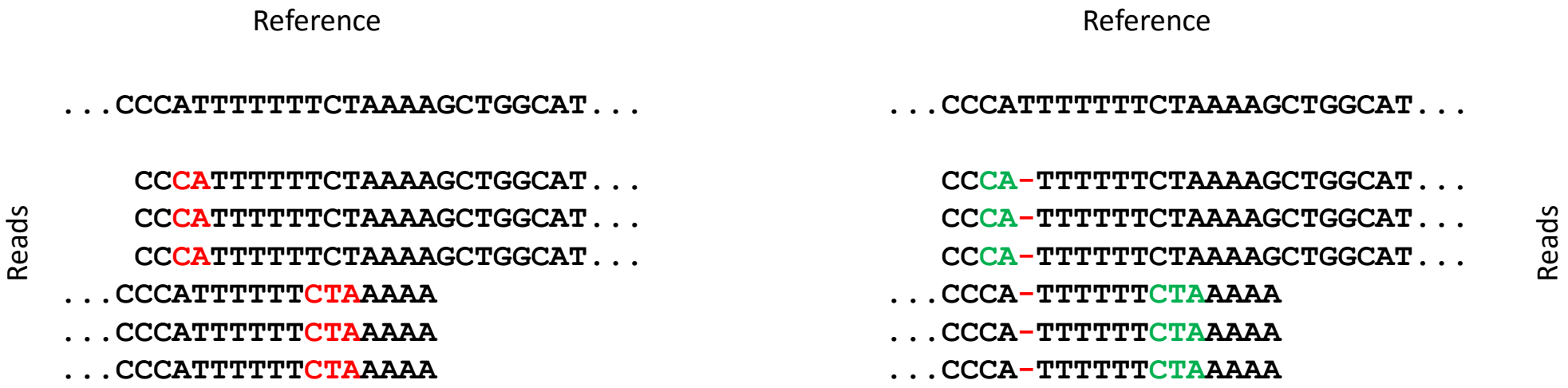
This looks better !

Only seen after aligning all (at least some) reads!

Ambiguity of alignment: around adjacent SNPs



Ambiguity of alignment: around homo-polymer runs flanked by adjacent SNPs



Remedy: local realignment

Generate intervals of interest from sample alignments

```
java -jar GenomeAnalysisTK.jar \
-T RealignerTargetCreator \
-nt 4 \
-R refgenome.fa \
-I sample1.sorted.dedup.bam \
-o realign.intervals
```

OR

Generate intervals of interest from known indels (once – will be good for all samples)

```
java -jar GenomeAnalysisTK.jar \
-T RealignerTargetCreator \
-R fergenome.fa \
-known known_indels.vcf
-o realign.intervals
```

Realign (multiple sequence alignment)

```
java -jar GenomeAnalysisTK.jar \
-T IndelRealigner  \
-R refgenome.fa\
-targetIntervals realign.intervals \
-I sample1.sorted.dedup.bam \
-o sample1.sorted.dedup.realigned.bam
```

Fix mate pair info in BAM (PICARD)

```
java -jar $PICARDDIR/picard.jar FixMateInformation \
INPUT=sample1.sorted.dedup.realigned.bam \
OUTPUT=sample1.sorted.dedup.realigned.fixmate.bam \
SO=coordinate \
CREATE_INDEX=true
```

Local realignment: when is it needed?

Local re-alignment is time-consuming!

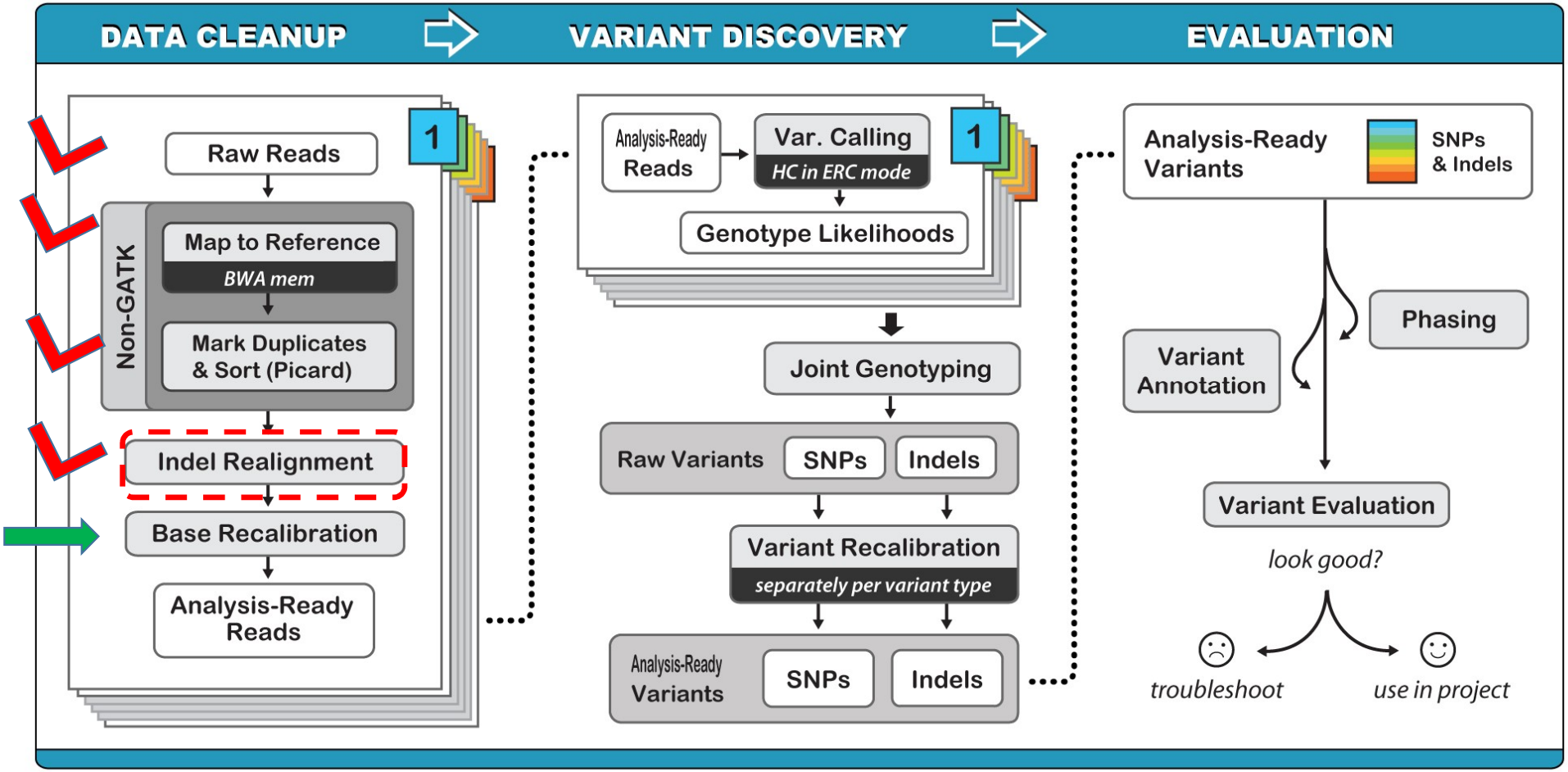
Re-alignment no longer recommended if the genotyping method used downstream involves **local haplotype assembly**

HaplotypeCaller (GATK)
FreeBayes
re-alignment implicit in the assembly algorithm

Still needed if genotypes called from allelic depths at individual sites

UnifiedGenotyper (GATK)
samtools

“Best Practices” for DNA-Seq variant calling



Base quality score recalibration

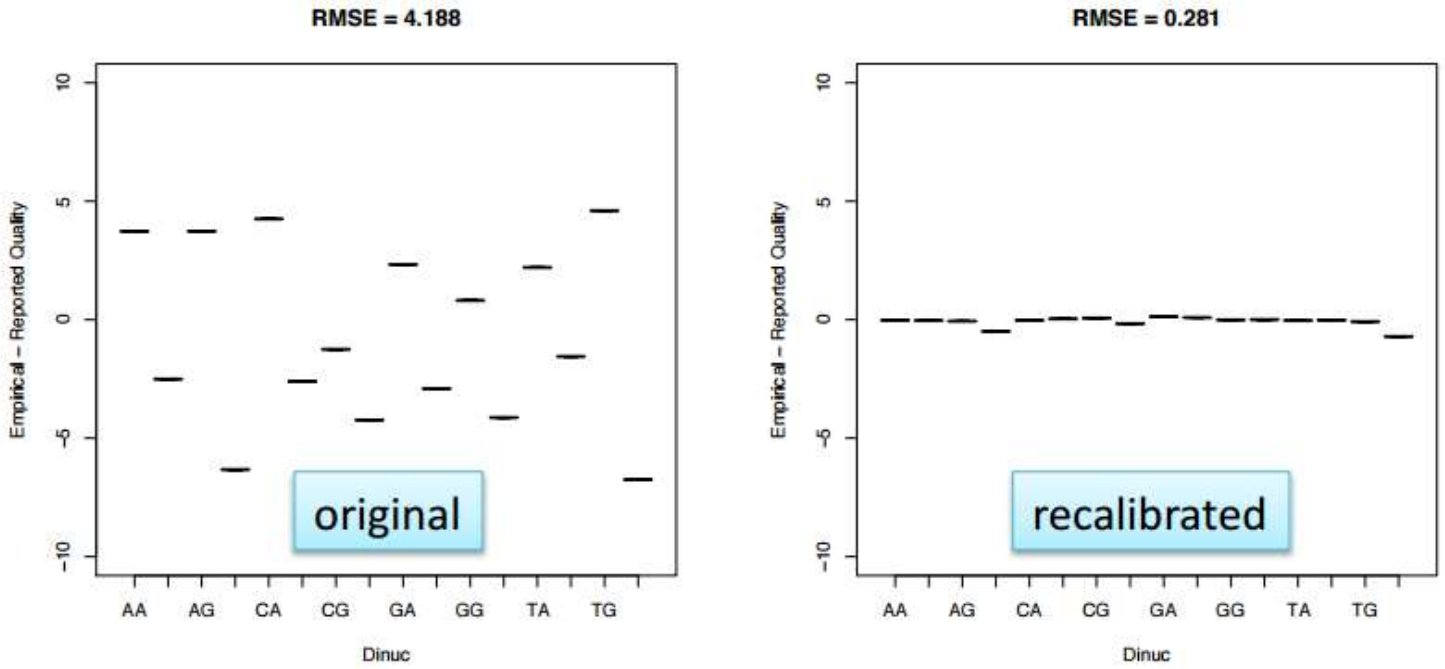
- Define “bins” in terms of covariates:
  - Lane
  - Original quality score (reported by Illumina pipeline)
  - Machine cycle (position on read)
  - Sequencing context (what bases are around)
- Scan all aligned reads (i.e., bases) in a given read group
  - Classify each base to a “bin”; decide whether it is a mismatch
- In each bin
  - count the number of mismatches (where read base != reference base)
  - Calculate **empirical quality score** from **#mismatches/#all_observed_bases**; compare to original
- Compile a **database** of corrections

- Scan all reads (i.e., bases) again (in a BAM file)
- For each base
  - Classify into a bin
  - Apply bin-specific correction to base quality scores (based on the database collected in previous step)

Caveats:
- Local indel realignment should be done before recalibration
- Known variation (SNPs and indels) have to be excluded (not a source of errors)

Base quality scores reported by a sequencer may be inaccurate and biased

Example: Bias in the qualities reported depending on nucleotide context



https://www.broadinstitute.org/gatk/guide/topic?name=methods

Base quality score recalibration pipeline

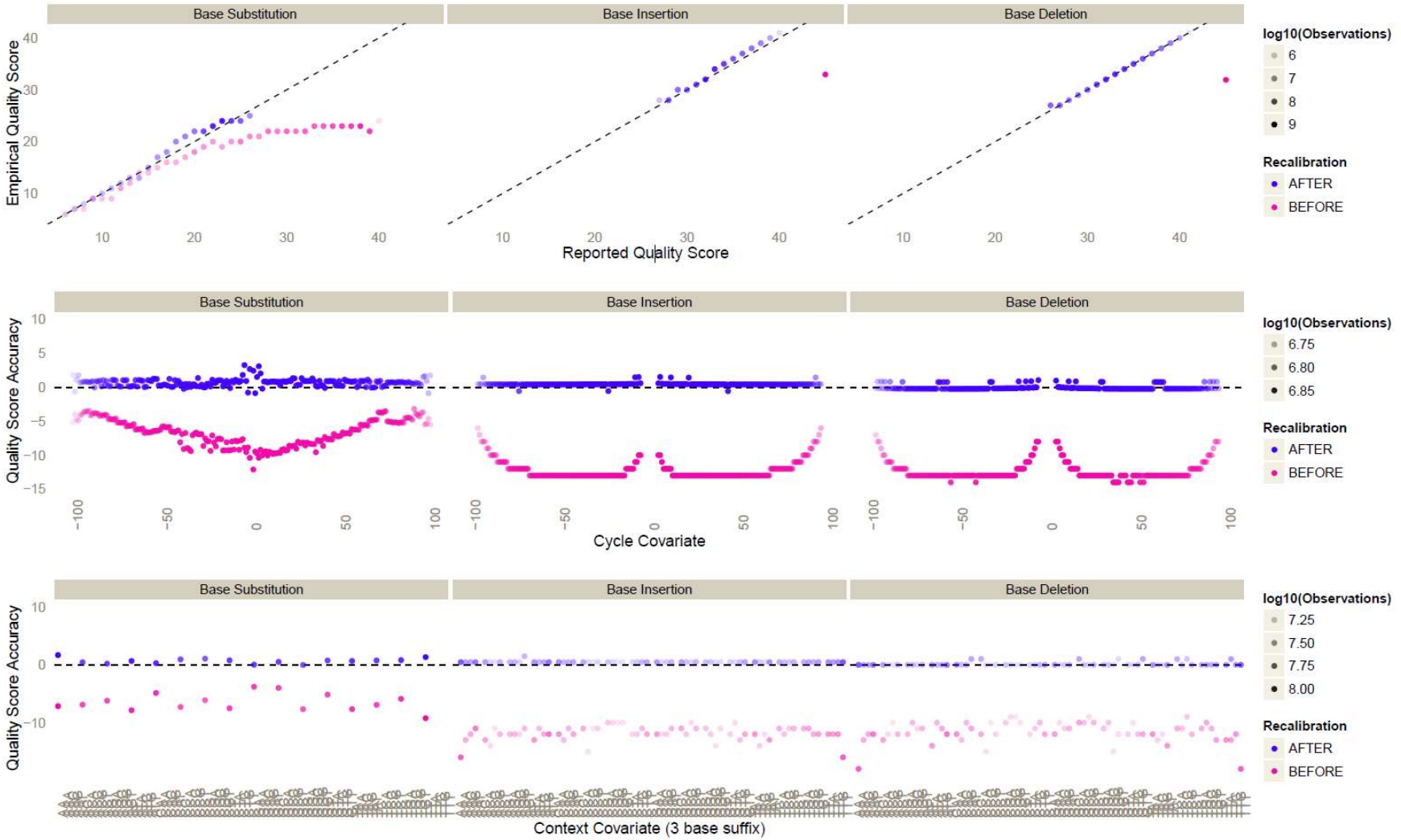
Collect mismatch statistics in bins

```
java -jar GenomeAnalysisTK.jar  \
-T BaseRecalibrator  \
-R refgenome.fasta\
-knownSites known_snps_indels.vcf  \
-I sample1.sorted.dedup.realigned.fixmate.bam \
-o sample1.sorted.dedup.realigned.fixmate.recal_data.table \
-cov ReadGroupCovariate  \
-cov QualityScoreCovariate \
-cov CycleCovariate
```

Recalibrate base qualities in the BAM file

```
java -jar GenomeAnalysisTK.jar  \
-T PrintReads  \
-R refgenome.fasta \
-BQSR sample1.sorted.dedup.realigned.fixmate.recal_data.table \
-I sample1.sorted.dedup.realigned.fixmate.bam \
-o sample1.sorted.dedup.realigned.fixmate.recal.bam
```

This is what recalibration results may look like



Running alignment preparation in parallel

Alignment

Multithreading in BWA mem works well up to 10-15 CPUs. On a machine with 24 CPUS, run 2 BWA mem jobs concurrently, each on 10 threads (**`bwa mem -t 10`** …) .

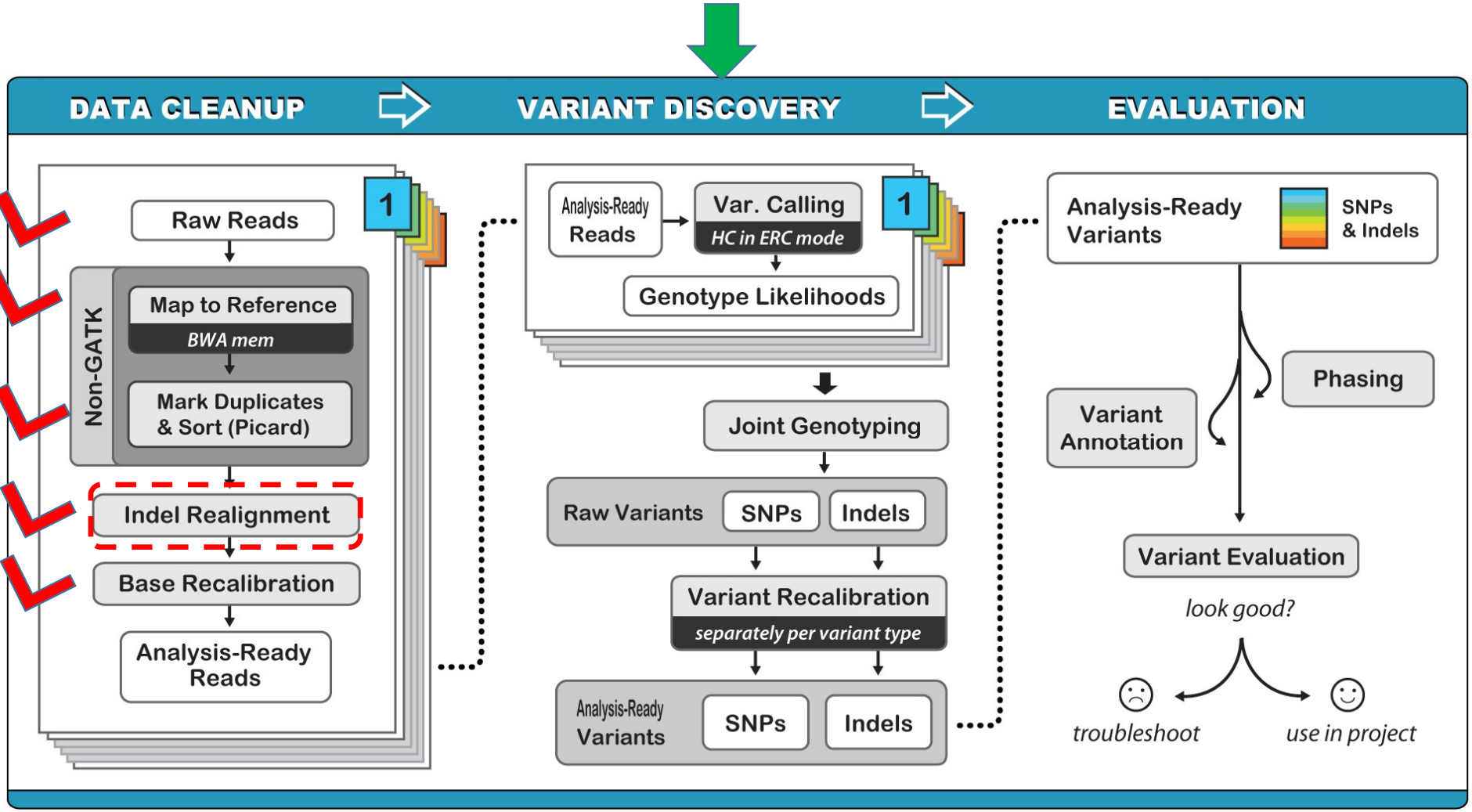
Mark Duplicates

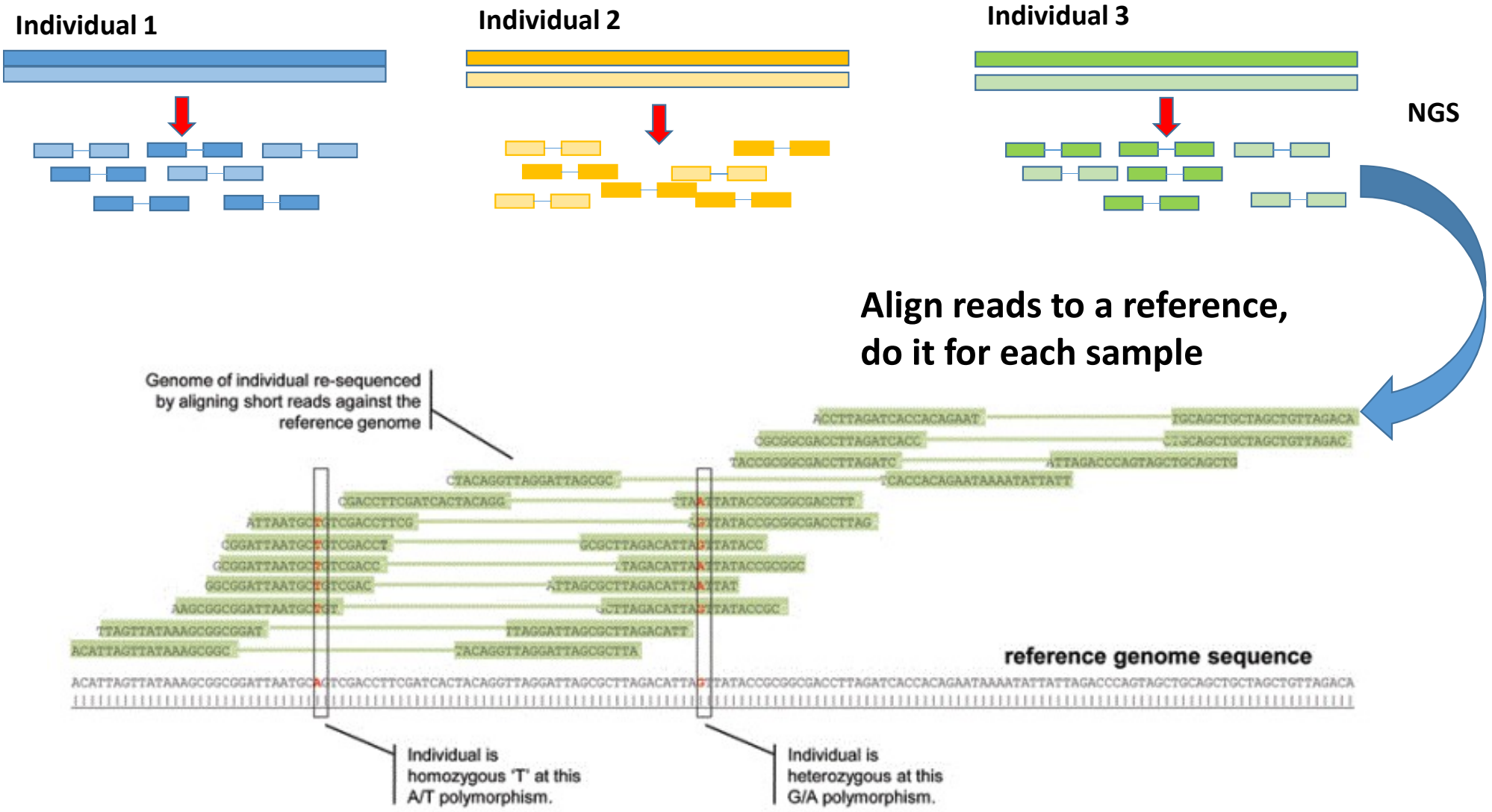
Realign

Recalibrate

- Multithreading non-existent or not too efficient
- best to execute this part of pipeline as multiple independent jobs (one per lane or sample/lane), run in parallel on one or multiple machines.
- Required memory and disk access bandwidth will determine the optimal number of concurrent jobs per machine. Experiment!
- These steps take **a long time** (may be comparable with alignment itself!)

“Best Practices” for DNA-Seq variant calling




Individual 1
Individual 2
Individual 3
NGS
Align reads to a reference,
do it for each sample
Genome of individual re-sequenced by aligning short reads against the reference genome
reference genome sequence
Individual is homozygous 'T' at this A/T polymorphism.
Individual is heterozygous at this G/A polymorphism.

Objectives…..

For each site on the genome determine

- Whether or not there is **any variation** at this site (across all samples)
- If there is variation, **assign a genotype** (for diploids: pair of alleles) to each sample
- Decide which sites can be considered **variant** from all samples' perspective and report these sites (positions, alleles, sample genotypes, …)

How it's done?

<u>The old days:</u>

Call variant base on thresholds (e.g., ratio of reference/non-reference bases)
Assign genotypes based on other thresholds

<u>Now-days: probabilistic framework</u>

- Calculate and report probabilities (or **likelihoods**) of various sample genotypes (given read alignments)
- Calculate and report **probability** of a site being a variant (given read alignments)
- Input: read alignments, read **base quality scores** (preferably **recalibrated**), read mapping quality

How to describe variants: Variant Call Format (VCF)

HEADER LINES: start with "##", describe all symbols found later on in FORMAT and ANNOTATIONS, e.g.,

```
##fileformat=VCFv4.1
##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt alleles in the order listed">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth (reads with MQ=255 or with bad mates are filtered)">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
……
```

SITE RECORDS:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	ZW155	ZW177
chr2R	2926	.	C	A	345.03	PASS	[ANNOTATIONS]	GT:AD:DP:GQ:PL	0/1:4,9:13:80:216,0,80	0/0:6,0:6:18:0,18,166
chr2R	9862	.	TA	T	180.73	.	[ANNOTATIONS]	GT:AD:DP:GQ:PL	1/1:0,5:5:15:97,15,0	1/1:0,4:4:12:80,12,0
chr2R	10834	.	A	ACTG	173.04	.	[ANNOTATIONS]	GT:AD:DP:GQ:PL	0/0:14,0:14:33:0,33,495	0/1:6,3:9:99:105,0,315

ID: some ID for the variant, if known (e.g., dbSNP)

REF, ALT: reference and alternative alleles (on forward strand of reference)

QUAL = -10*log(1-p), where p is the probability of variant being present given the read data

FILTER: whether the variant failed a filter (filters defined by the user or program processing the file)

How to describe variants: Variant Call Format (VCF)

```
[HEADER LINES]
#CHROM  POS    ID  REF   ALT   QUAL    FILTER     INFO            FORMAT            ZW155                    ZW177
chr2R   2926   .    C     A    345.03    PASS   [ANNOTATIONS]   GT:AD:DP:GQ:PL  0/1:4,9:13:80:216,0,80   0/0:6,0:6:18:0,18,166
chr2R   9862   .   TA     T    180.73     .     [ANNOTATIONS]   GT:AD:DP:GQ:PL  1/1:0,5:5:15:97,15,0     1/1:0,4:4:12:80,12,0
chr2R 10834    .    A  ACTG    173.04     .     [ANNOTATIONS]   GT:AD:DP:GQ:PL  0/0:14,0:14:33:0,33,495       ./.
```

GT (genotype):

0/0	reference homozygote
0/1	reference-alternative heterozygote
1/1	alternative homozygote
0/2, 1/2, 2/2, etc.	- possible if more than one alternative allele present
./.	missing data

AD: allele depths

DP: total depth (may be different from sum of AD depths, a the latter include only reads significantly supporting alleles)

PL: genotype likelihoods (phred-scaled), normalized to the best genotype, e.g.,

PL(0/1) = -10*log[Prob(data|0/1) / Prob(data|best_genotype)]

GQ: genotype quality – this is just PL of the second-best genotype

How to describe variants: Variant Call Format (VCF)

```
[HEADER LINES]
#CHROM  POS    ID  REF   ALT   QUAL    FILTER     INFO            FORMAT          ZW155                  ZW177
chr2R   2926   .    C     A    345.03    PASS   [ANNOTATIONS]   GT:AD:DP:GQ:PL  0/1:4,9:13:80:216,0,80  0/0:6,0:6:18:0,18,166
chr2R   9862   .   TA     T    180.73     .     [ANNOTATIONS]   GT:AD:DP:GQ:PL  1/1:0,5:5:15:97,15,0    1/1:0,4:4:12:80,12,0
chr2R 10834    .    A  ACTG   173.04     .     [ANNOTATIONS]   GT:AD:DP:GQ:PL  0/0:14,0:14:33:0,33,495 0/1:6,3:9:99:105,0,315
```

[ANNOTATIONS]: all kinds of quantities and flags that characterize the variant; supplied by the variant caller (different callers will do it differently)

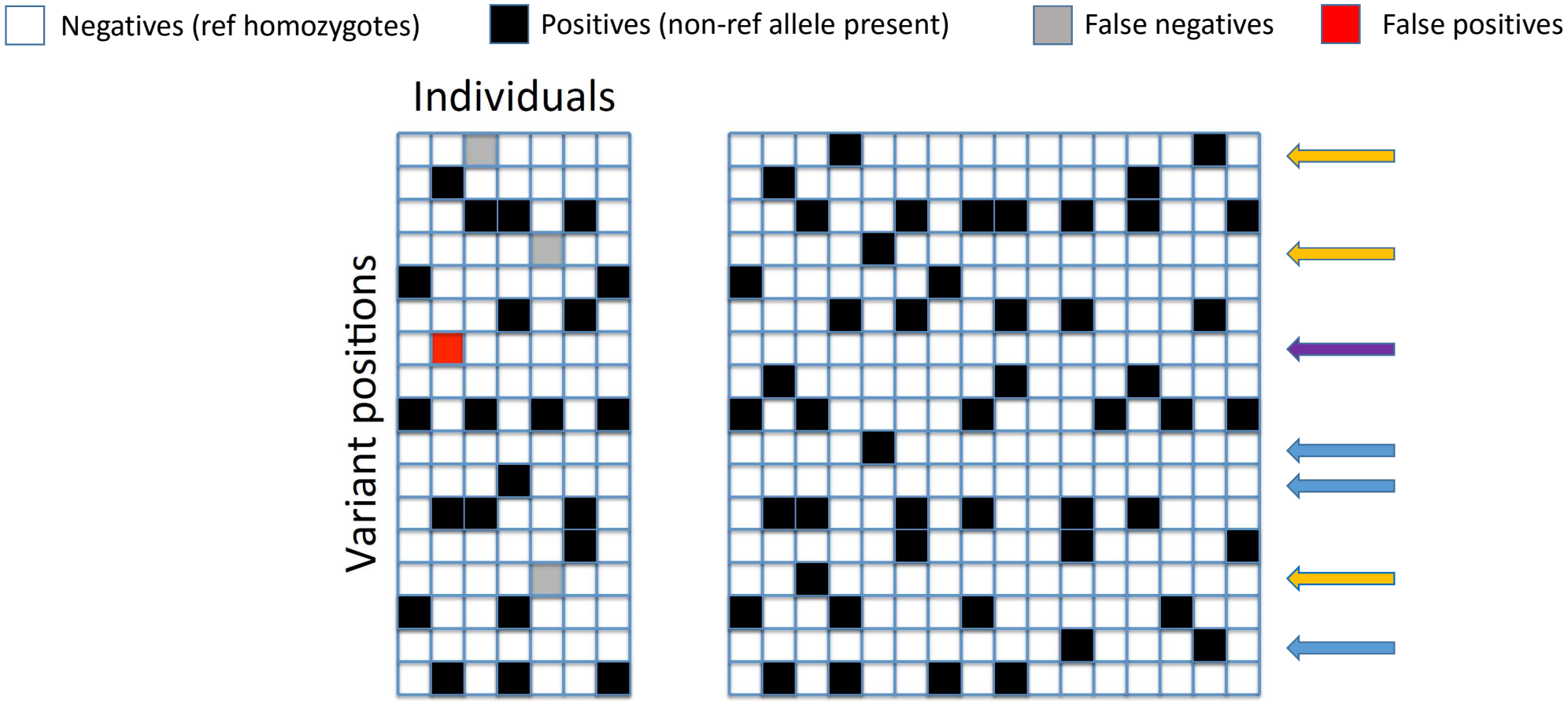
Example:

```
AC=2;AF=0.333;AN=6;DP=16;FS=0.000;GQ_MEAN=16.00;GQ_STDDEV=10.54;MLEAC=2;MLEAF=0.33
3;MQ=25.00;MQ0=0;NCC=1;QD=22.51;SOR=3.611
```

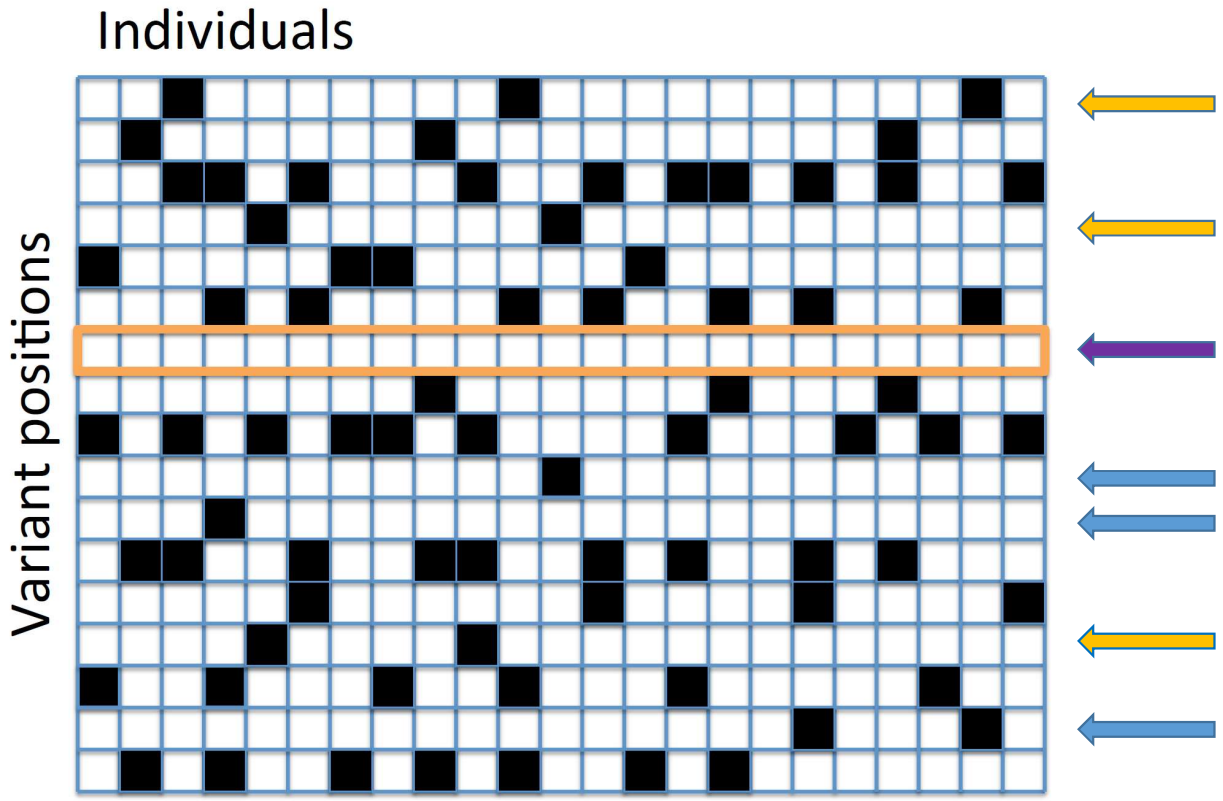
All ANNOTATION parameters are defined in the **HEADER LINES** on top of the file

```
…
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes, for each ALT allele, in the same order as listed">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency, for each ALT allele, in the same order as listed">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles in called genotypes">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth; some reads may have been filtered">
##INFO=<ID=FS,Number=1,Type=Float,Description="Phred-scaled p-value using Fisher's exact test to detect strand bias">
##INFO=<ID=GQ_MEAN,Number=1,Type=Float,Description="Mean of all GQ values">
##INFO=<ID=MQ,Number=1,Type=Float,Description="RMS Mapping Quality">
##INFO=<ID=NCC,Number=1,Type=Integer,Description="Number of no-called samples">
##INFO=<ID=QD,Number=1,Type=Float,Description="Variant Confidence/Quality by Depth">
##INFO=<ID=SOR,Number=1,Type=Float,Description="Symmetric Odds Ratio of 2x2 contingency table to detect strand bias">
…
```

Two batches of individuals considered separately



Two batches of individuals considered together



Sample-by-sample or joint (cohort-level) variant calling?

"Seeing" reads from multiple samples (mapped to a region of reference genome) allows smarter decisions about which alleles are real and which are sequencing or alignment errors…

More confidence in variant calling

Multiple samples data allow calling a variant even if individual sample calls are of low quality

Joint calling is better, but….

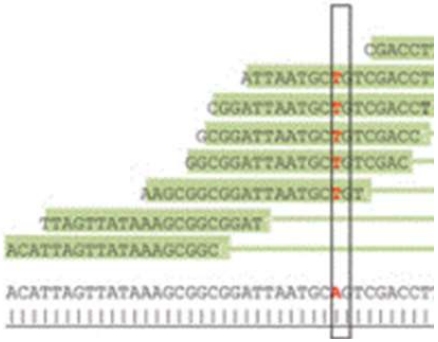
Scales badly with the number of samples

"N+1" problem: what if one more (or a few more) individuals are added?
Need to repeat the calling! (in finite time….)

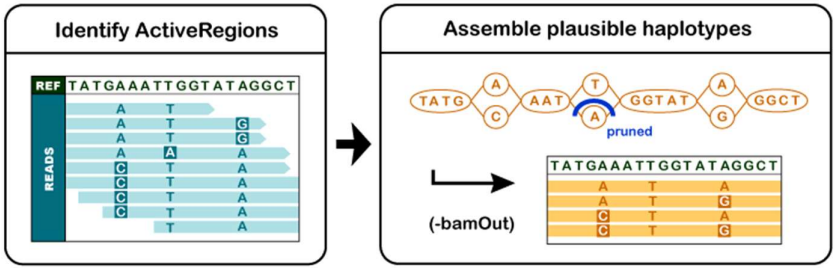
Variant detection, PL calculation

Joint calling strategy

Variants found from read pileup; PL from allelic read depths and base qualities



Variants found from locally assembled haplotypes; PL from read alignment to haplotypes



s1.bam s2.bam … sN.bam

↓

N-sample VCF file

Software:

- ❑ **GATK (UnifiedGenotyper)**
- ❑ **Sentieon (commercial GATK)**
- ❑ **samtools**

Software:

- ❑ **GATK (HaplotypeCaller)**
- ❑ **Sentieon (commercial GATK)**
- ❑ **FreeBayes** (Garrison E, Marth G. arXiv:1207.3907 [q-bio.GN] 2012)

RECOMMENDED

"N+1" problem: adding one or a few more samples means re-call of everything!

s1.bam **s1.bam** **… sN.bam**

↓ slow ↓ slow ↓

s1.g.vcf **s1.g.vcf** **… sN.g.vcf**

↘ ↓ fast ↙

N-sample VCF file

Software: **none**

Software:

- ❑ **GATK (HaplotypeCaller in gVCF mode)**
- ❑ **GATK (GenotypeGVCFs)**
- ❑ **Sentieon (commercial GATK)**

RECOMMENDED

No "**N+1" problem**: process extra BAMs into **g.vcf** and repeat the (**fast**) **joint calling**

`*.g.vcf`:

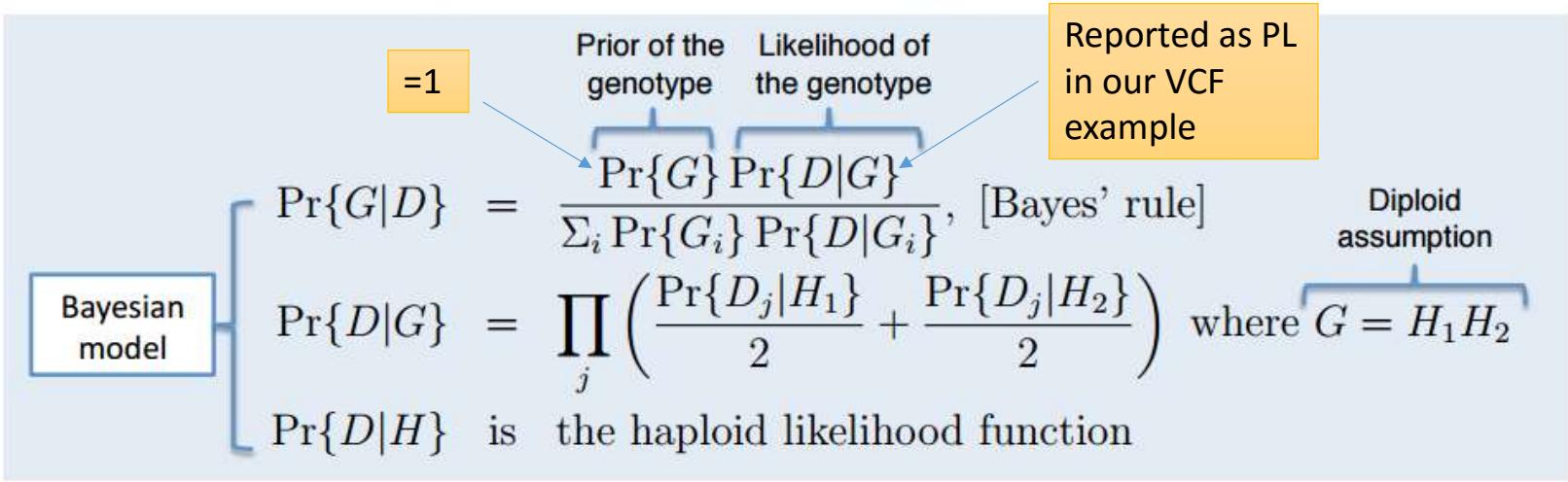
- Per-sample intermediate files summarizing read depths and genotype likelihoods for each site of genome
- Derived from BAM files (in fact – replace them)
- Format somewhat similar to VCF, but with invariant regions represented as single records

```
#CHROM  POS     ID      REF       ALT        QUAL    FILTER     INFO                FORMAT                SAMPLE_NAME
4       2503    .       G       <NON_REF>       .       .       END=2504        GT:DP:GQ:MIN_DP:PL      0/0:2:0:2:0,0,0
4       2505    .       C       <NON_REF>       .       .       END=2517        GT:DP:GQ:MIN_DP:PL      0/0:2:6:2:0,6,90
4       2518    .       C       T,<NON_REF>     0.01    .       DP=1;ExcessHet=3.0103;MLEAC=0,0;MLEAF=0.000,0.000;RAW_MQ=441.00
GT:AD:DP:GQ:PGT:PID:PL:SB     0/0:1,0,0:1:3:0|1:2518_C_T:0,3,26,3,26,26:0,1,0,0
4       2519    .       A       <NON_REF>       .       .       END=2531        GT:DP:GQ:MIN_DP:PL      0/0:2:6:2:0,6,90
4       2532    .       G       C,<NON_REF>     20.00   .       DP=1;ExcessHet=3.0103;MLEAC=1,0;MLEAF=0.500,0.000;RAW_MQ=441.00
GT:AD:DP:GQ:PGT:PID:PL:SB     1/1:0,1,0:1:3:0|1:2518_C_T:45,3,0,45,3,45:0,0,0,1
4       2533    .       A       <NON_REF>       .       .       END=2543        GT:DP:GQ:MIN_DP:PL      0/0:2:6:2:0,6,90
```

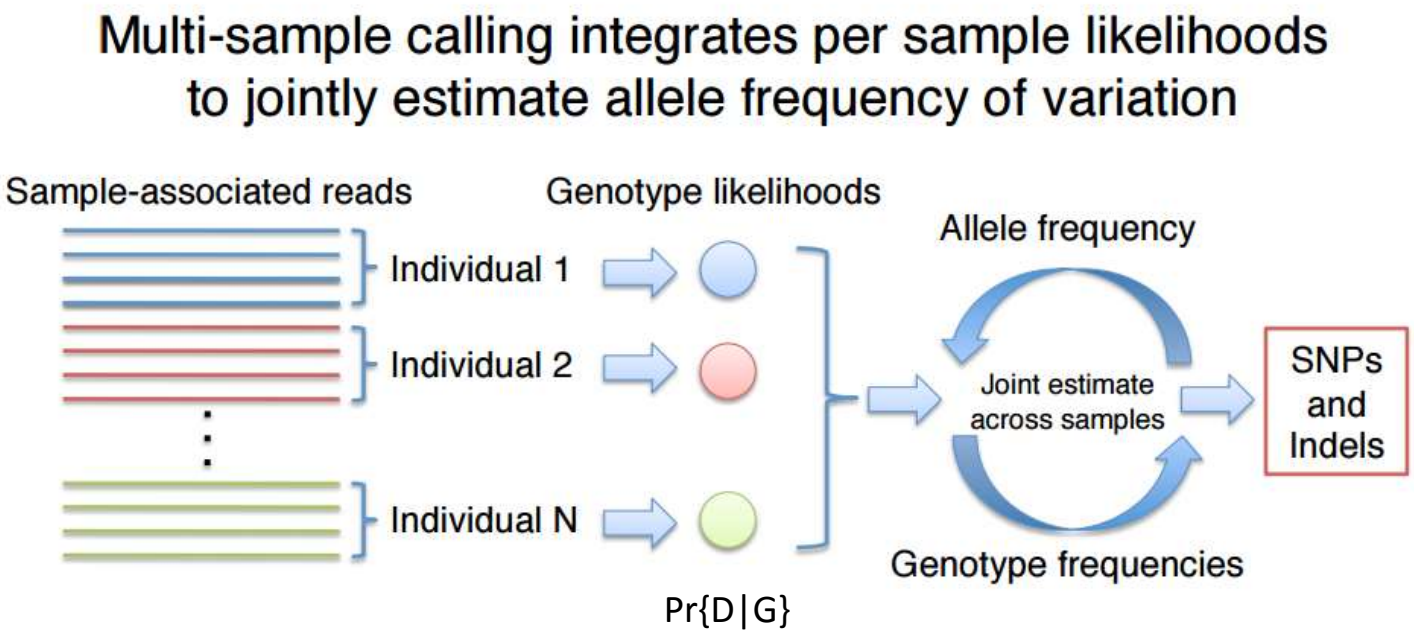
- Much smaller than BAM files
- Used later to make **joint calls** on the cohort (instead of the original BAM files)

This approach addresses the "**N+1" problem**: process one extra BAM into **g.vcf** and repeat the (fast) joint calling

SNP and Indel calling is a large-scale Bayesian modeling problem



- Inference: what is the genotype G of each sample given read data D for each sample?
- Calculate via Bayes' rule the probability of each possible G
- Product expansion assumes reads are independent
- Relies on a likelihood function to estimate probability of sample data given proposed haplotype



For each site, obtain distribution of count of non-reference allele:

Per sample **Genotype Likelihoods** + **Prior** → **Pr{AC=i | D}**

Prior: Pr{AC=i} = Het/i (where Het is population heterozygosity; or define your own prior)

QUAL = -10*log Pr{AC=0| D} (reported in VCF file)

Haplotype likelihood function in UnifiedGenotyper

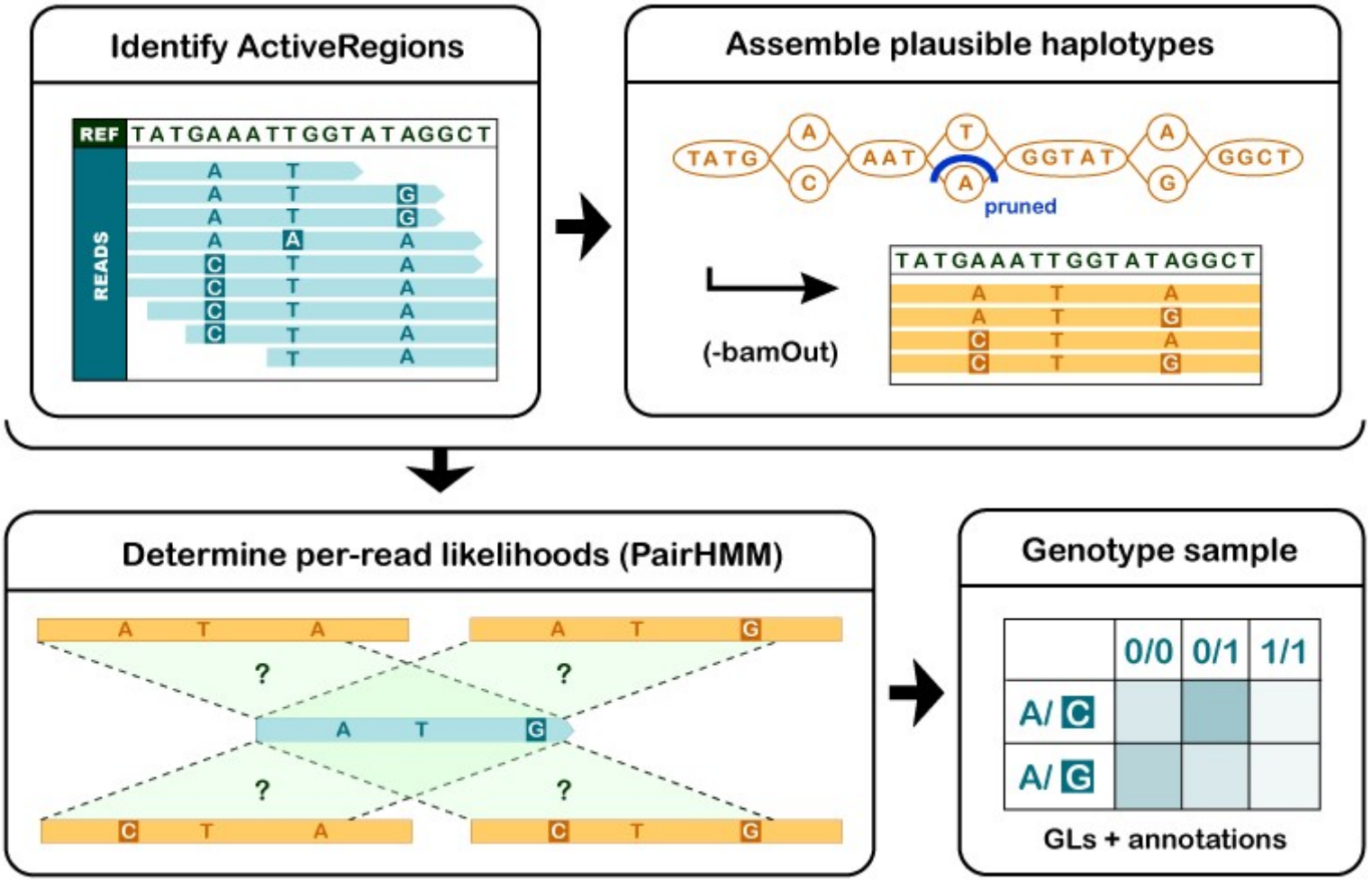
$$\Pr\{D_j|H\} = \Pr\{D_j|b\}, \text{ [single base pileup]}$$

$$\Pr\{D_j|b\} = \begin{cases} 1-\epsilon_j & D_j = b, \\ \epsilon_j & \text{otherwise.} \end{cases}$$

From base quality score

Substitution-specifc rates (confusion matrix) may also be used here

Haplotype likelihood function in HaplotypeCaller



P{D_j|H} determined from PairHMM scores of reads alignments to haplotypes (based on base qualities)

Haplotype caller: what does it do?

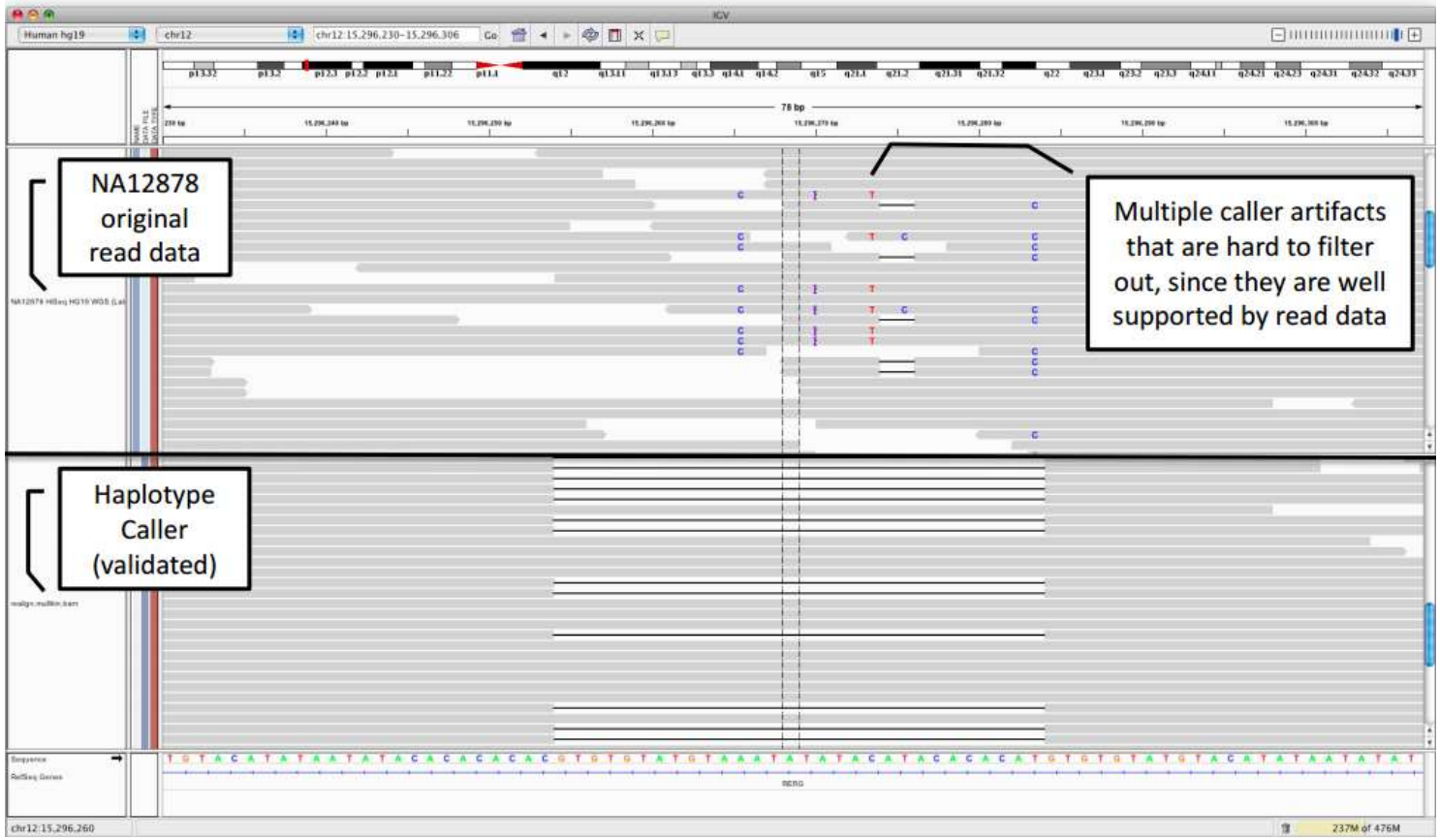
1. **Define active regions**
The program determines which regions of the genome it needs to operate on, based on the presence of significant evidence for variation.

2. **Determine haplotypes by re-assembly of the active region**
For each ActiveRegion, the program builds a De Bruijn-like graph to reassemble the ActiveRegion, and identifies what are the possible haplotypes present in the data. The program then realigns each haplotype against the reference haplotype using the Smith-Waterman algorithm in order to identify potentially variant sites.

3. **Determine likelihoods of the haplotypes given the read data P{Dj|H}**
For each ActiveRegion, the program performs a pairwise alignment of each read against each haplotype using the PairHMM algorithm. This produces a matrix of likelihoods of haplotypes given the read data. These likelihoods are then marginalized to obtain the likelihoods of alleles for each potentially variant site given the read data.

4. **Assign sample genotypes**
For each potentially variant site, the program applies Bayes' rule, using the likelihoods of alleles given the read data to calculate the likelihoods of each genotype per sample given the read data observed for that sample. The most likely genotype is then assigned to the sample.

Artifact SNPs and small indels caused by large indel is only recovered by local assembly

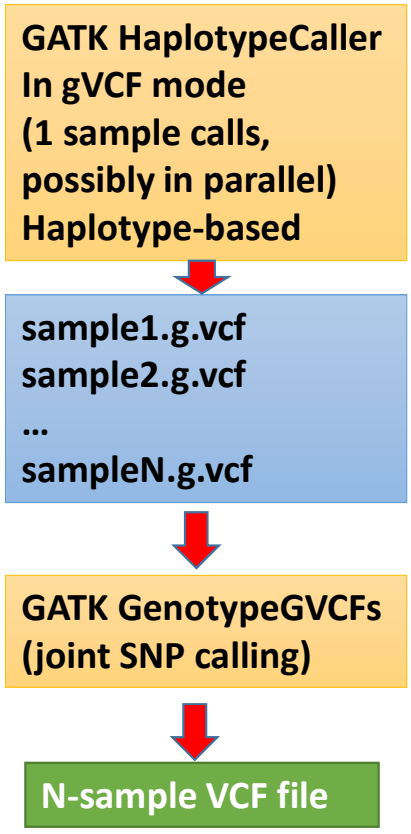


HaplotypeCaller in gVCF mode….

Run for each sample (on a multi-CPU machine, **run a few simultaneously**)

```
java -jar GenomeAnalysisTK.jar \
     -T HaplotypeCaller \
     -R genome.fa \
     -I sample1.sorted.dedup.realigned.fixmate.recal.bam  \
     --emitRefConfidence GVCF \
     -o sample1.g.vcf
```

Slow



….Followed by joint variant calling with GenotypeGVCFs

Run **once** after all *.g.vcf files are obtained

```
java -Xmx2g -jar GenomeAnalysisTK.jar \
   -T GenotypeGVCFs \
   -R genome.fa \
   --variant sample1.g.vcf \
   --variant sample2.g.vcf \
   --variant sample3.g.vcf \
   --variant sample4.g.vcf \
   -stand_call_conf 30 \
   -o 4samples.vcf
```

Fast

Running HaplotypeCaller (variant-only mode)

GATK HaplotypeCaller (joint SNP calling) Haplotype-based

N-sample VCF file

```
java -jar GenomeAnalysisTK.jar \
     -T HaplotypeCaller \
     -R genome.fa \
     -I sample1.sorted.dedup.realigned.fixmate.recal.bam  \
     -I sample2.sorted.dedup.realigned.fixmate.recal.bam  \
     -I sample3.sorted.dedup.realigned.fixmate.recal.bam  \
     -I sample4.sorted.dedup.realigned.fixmate.recal.bam  \
     -L chr2R \
     -stand_call_conf 30 \
     -o 4samples_joint_call.chr2R.vcf
```

May be parallelized by genome region (using –L option)
i.e., different regions run on different processors

Note:

Haplotype assembly uses reads from all samples (rather than one at a time), and so…

…resulting VCF file will not be exactly equivalent to that obtained from gVCF mode runs followed by GenotypeGVCFs

Running UnifiedGenotyper

GATK UnifiedGenotyper
(joint SNP calling)
Site-by-site

N-sample VCF file

```
java -Djava.io.tmpdir=$TMP -jar GenomeAnalysisTK.jar \
     -T UnifiedGenotyper \
     -R genome.fa \
     -I sample1.sorted.dedup.realigned.fixmate.recal.bam  \
     -I sample2.sorted.dedup.realigned.fixmate.recal.bam  \
     -I sample3.sorted.dedup.realigned.fixmate.recal.bam  \
     -I sample4.sorted.dedup.realigned.fixmate.recal.bam  \
     -L chr2R \
     -stand_call_conf 30 \
     -o 4samples.UG.chr2R.vcf
```

May be parallelized by genome region (using –L option)
i.e., different regions run on different processors

Broad recommends running HaplotypeCaller-based pipeline instead if this

However, still recommended for
Pooled sample cases
High ploidy cases

Options to pay attention to in HaplotypeCaller, GenotypeGVCFs, and UnifiedGenotyper

`-stand_call_conf [number]`

Variants with quality score (QUAL) less than **`[number]`** will not be written to VCF file. **Good to set this low** – better have too many raw variants than too few. Can always filter VCF file later. Default: 30.

`--num_cpu_threads_per_data_thread [number]`

Run on **`[number]`** threads (CPU cores). Default: 1 The program scales reasonably up to 8-10 threads.

`-dcov [int]`

Read depth at any locus will be capped at **`[number]`**; the goal is to provide even distribution of read start positions while removing excess coverage. For truly unbiased down-sampling, use **`-dfrac`** Defaults are usually high (250) – can be reduced to speed things up.

Alternatives to GATK

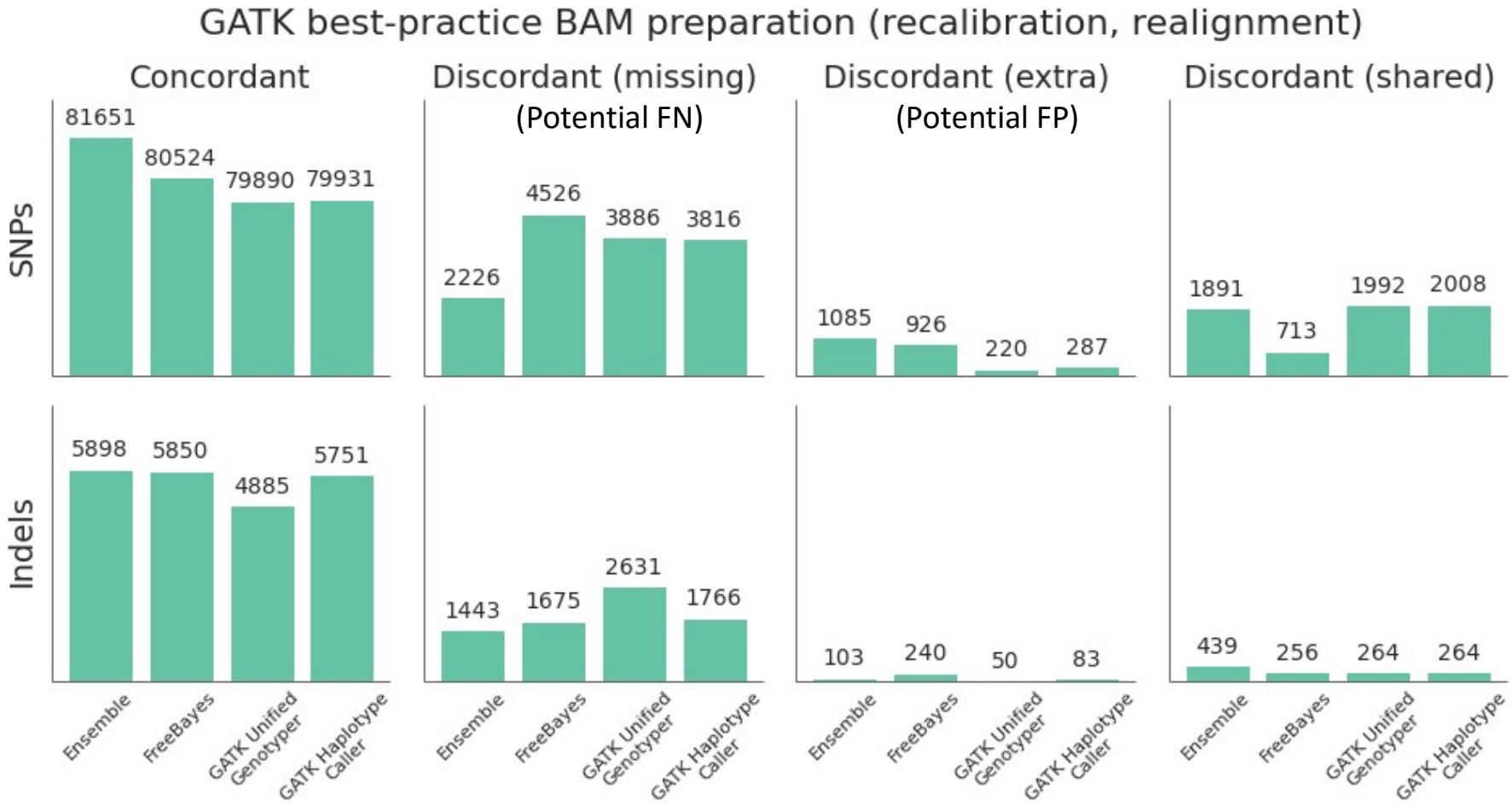
FreeBayes (Erik Garrison et al., https://github.com/ekg/freebayes)

- Haplotype-based variant detection (no re-alignment around indels needed)
- Better (than GATK's) Bayesian model, directly incorporating a number of metrics, such as read placement bias and allele balance
- **In our tests – order of magnitude faster than GATK HaplotypeCaller**!
- Still suffers from "N+1" problem

Sentieon (http://sentieon.com)

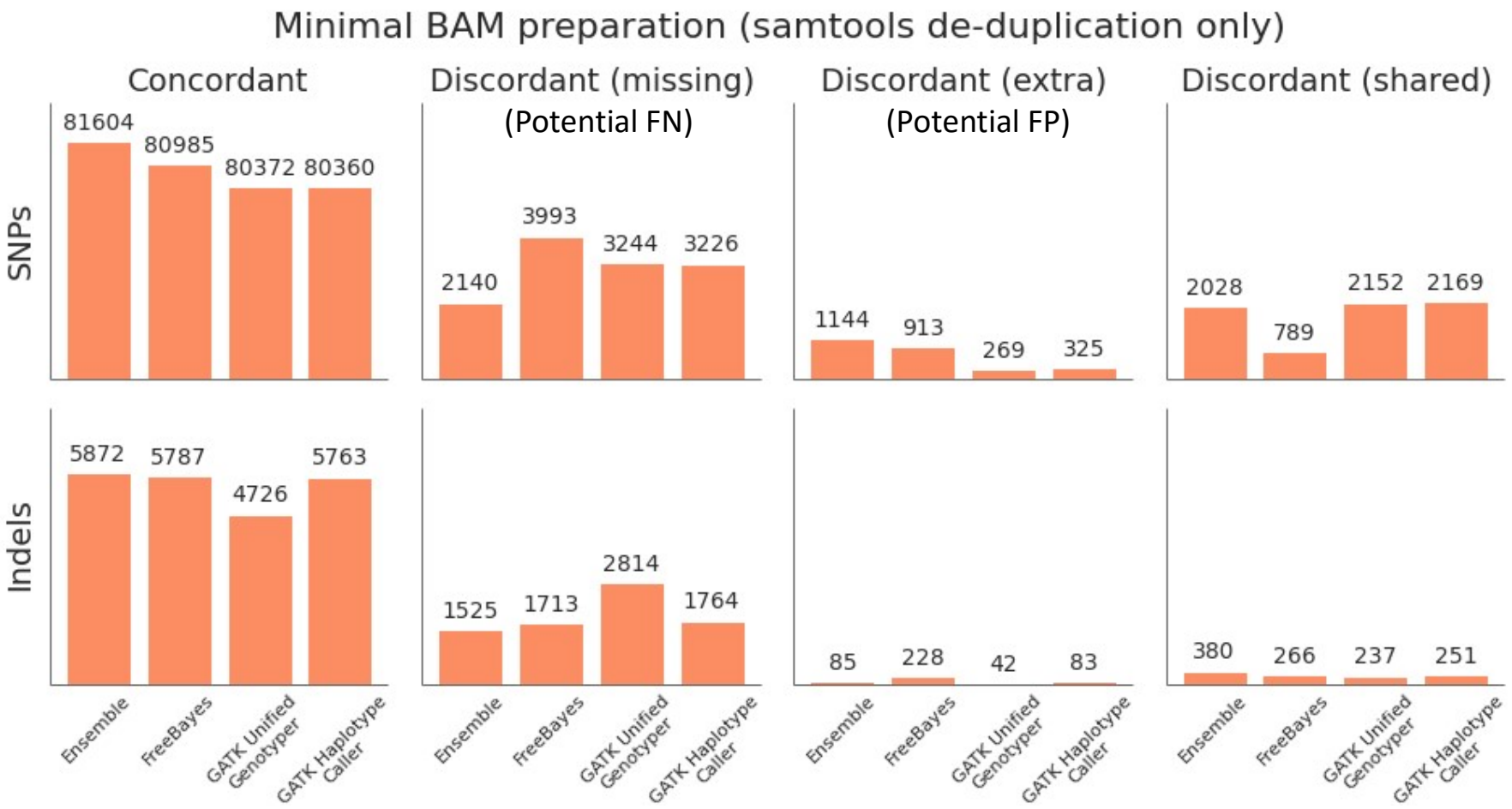
- Commercial version of GATK (currently equivalent to GATK 3.5)
- **10-30 times faster than GATK on most parts of the pipeline**
- Command syntax different from GATK (although functionality the same)
- Available on BioHPC Lab for $50/week (need to recover license cost)
  - License: 300 CPU cores of can run simultaneously (across all machines) at any time

Comparison of GATK and FreeBayes (how many known NA12878 SNPs/indels are called correctly/incorrectly)



Source: http://bcb.io/2013/10/21/updated-comparison-of-variant-detection-methods-ensemble-freebayes-and-minimal-bam-preparation-pipelines/

Comparison of GATK and FreeBayes (how many known NA12878 SNPs/indels are called correctly/incorrectly)



Source: http://bcb.io/2013/10/21/updated-comparison-of-variant-detection-methods-ensemble-freebayes-and-minimal-bam-preparation-pipelines/

- FreeBayes not much different than GATK HaplotypeCaller
    - somewhat better in detecting concordant SNPs and indels
    - somewhat worse with False Positives and False Negatives

- Expensive BAM file pre-processing (re-alignment around indels and base quality score recalibration) seems to have little impact, especially for haplotype-based callers (FreeBayes and HaplotypeCaller)

- However, Broad still recommends running base quality score recalibration

What to do with a freshly obtained set of called variants?

Simple linux tools help analyze a VCF file

Count variants:

```
grep –v ”#” hc.chr2R.vcf | wc –l
```

Extract sites located between positons 10000 and 20000 on chromosome chr2R and save them in a new VCF file:

```
head -1000 hc.chr2R.vcf | grep “#” > new_file.vcf
grep –v ”#” hc.chr2R.vcf | \
awk ’{if($1==”chr2R” && $2 >=10000 && $2 <=20000) print}’ >> new_file.vcf
```

Extract variants with quality (QUAL) greater than 100 (the resulting file will have no header!):

```
grep –v “#” hc.chr2R.vcf | awk ’{if($6>100) print}’ > good_variants
```

Useful tool: VariantFiltration – hard filtering on various criteria

Example:

```
java -jar GenomeAnalysisTK.jar  \
-T VariantFiltration \
-R genome.fa \
-filter "MQ0 >= 4 && ((MQ0 / (1.0 * DP)) > 0.1)"  \
-filter "FS>=10.0"   \
-filter "AN>=4" \
-filter "DP>100 || DP<4" \
-filterName HARD_TO_VALIDATE  \
-filterName SNPSBFilter \
-filterName SNPNalleleFilter  \
-filterName SNPDPFilter  \
-cluster 3 \
-window 10  \
--variant chr2R.4samples.vcf \
-o chr2R.4samples.filtered.vcf
```

Filtering options for SNPs may be different than for indels (see exercise)

Whenever any of the "-filter" conditions satisfied, the corresponding "-filterName" will be added to the FILTER field in VCF.

Commonly used filtering parameters (from GATK)

DP

Total depth of read coverage at the site (shouldn't be too low)

MQ0

Number of zero mapping quality reads spanning the site (should be low)

MQ

RMS mapping quality of reads spanning the site

FS

P-value (phred-scaled) of the strand bias contingency table (should be low)

QD

QUAL/(depth of non-reference reads) – should be large (e.g, >2)

ReadPosRankSum

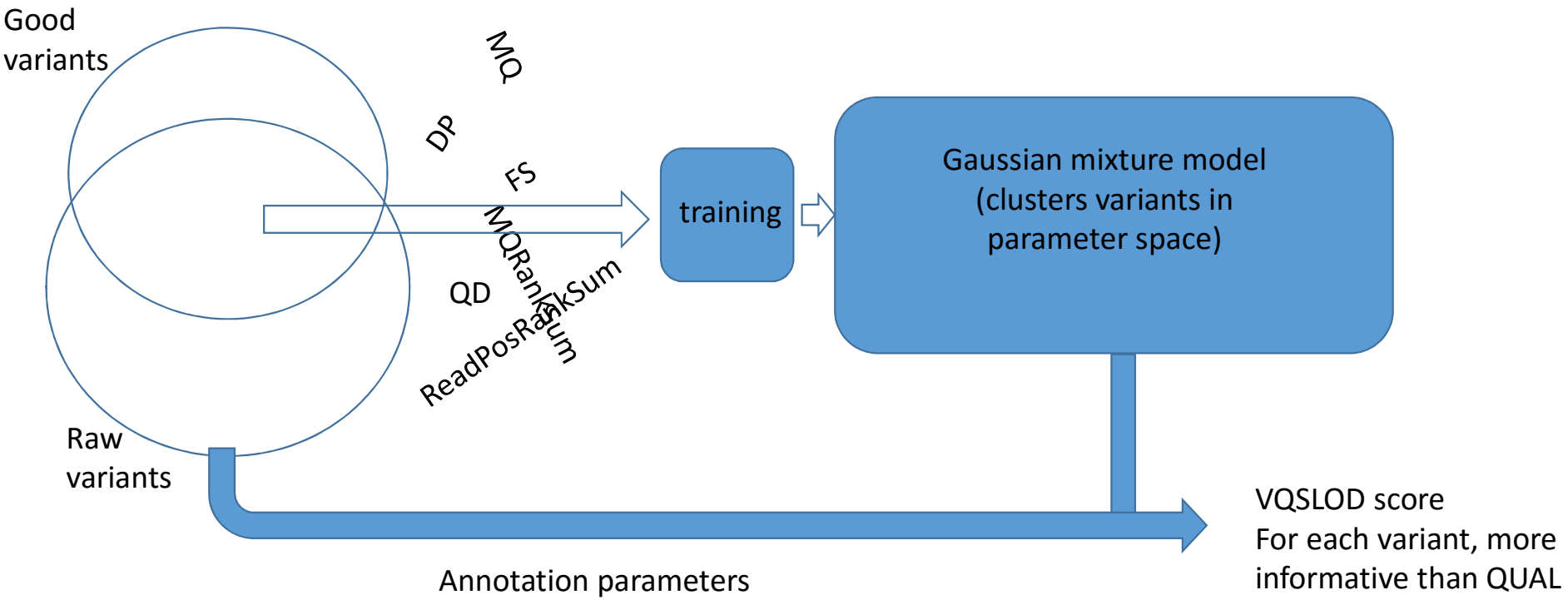
Parameter showing how close the variant site is to ends of reads (typically more positive for good variants) – available only for heterozygous sites

MQRankSum

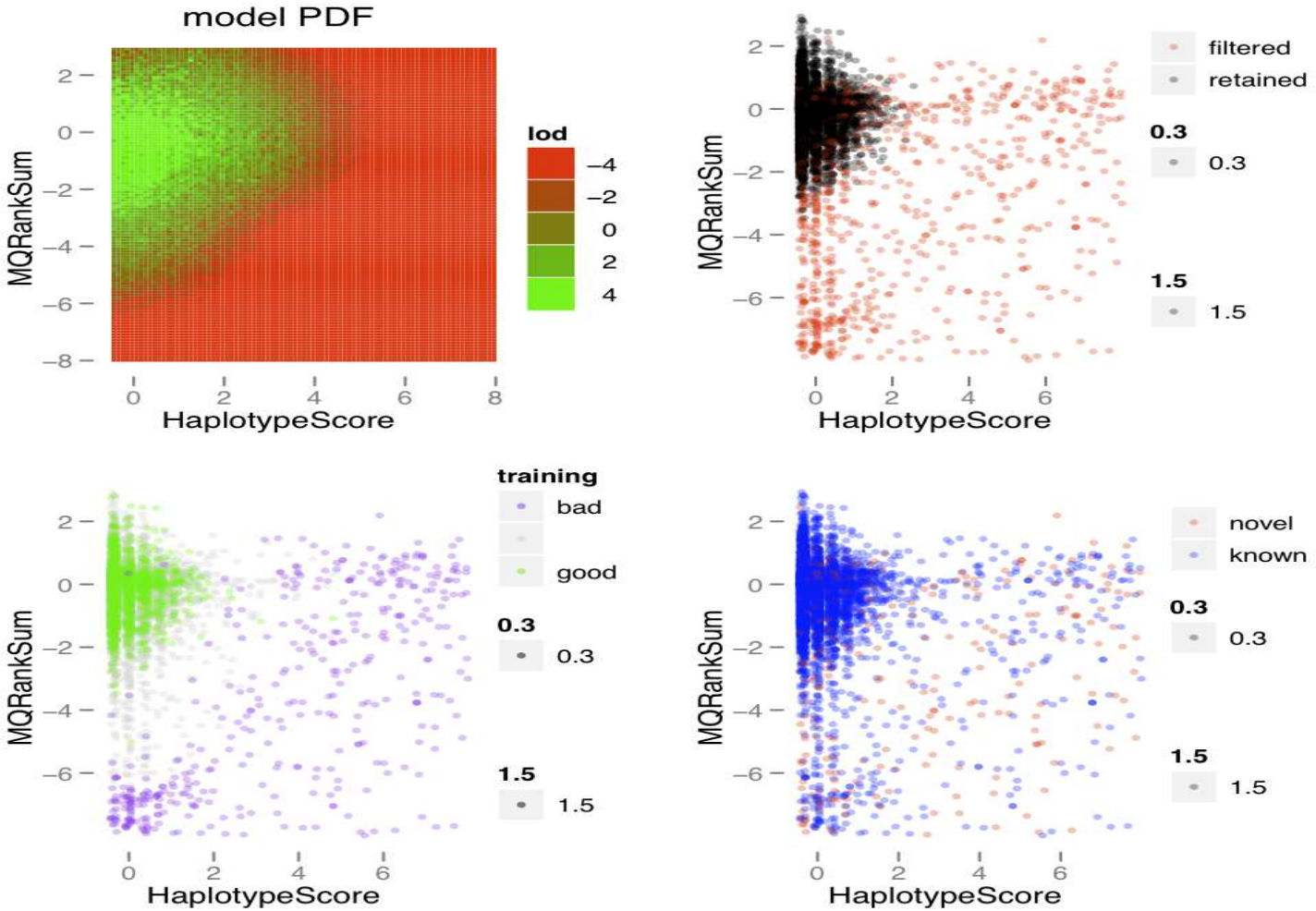
Parameter comparing mapping qualities of reads carrying an alternative allele to reference reads – available only for heterozygous sites (typically more positive for good variants).

Variant Quality Score Recalibration (VSQR)

Machine learning model, recommended instead of hard (threshold-based) filtering when a set of true, reliable variants is available.



2D cross-section though cluster of variants in multi-D parameters space



Useful tool: VariantEval – summary stats and comparison of callsets

```
java -Xmx2g -jar GenomeAnalysisTK.jar \
    -R genome.fa \
    -T VariantEval \
    -o file1.file2.comp.gatkreport \
    --eval:set1 file1.vcf  \
    --comp file2.vcf
```

Will summarize various properties of variants in **file1.vcf**

- Classes of variants
- Indel characteristics
- Ti/Tv
- Multi-allelic variants
- ….

Will compare to variants in file2.vcf

- Common variants and extra variants in file1.vcf (**compared to file2.vcf**)
- Concordance rate

Other VCF analysis and manipulation package: vcftools

vcftools (A. Auton, A. Amrcketta, http://vcftools.sourceforge.net/)

Obtain basis VCF statistics (number of samples and variant sites):

```
vcftools --vcf hc.chr2R.vcf
```

Extract subset of variants (chromosome chr2R, between positions 1M and 2M) and write tem a new VCF file

```
vcftools –vcf hc.chr2R.vcf --chr chr2R --from-bp 1000000 --to-bp 2000000
--recode –recode-INFO-all -c > subset.vcf
```

Get allele frequencies for all variants and write them to a file

```
vcftools --vcf hc.chr2R.vcf --freq -c > hc.chr2R.freqs
```

Compare two VCF files (will print out various kinds of compare info in files **`hc.ug.compare.*`**):

```
vcftools --vcf hc.chr2R.vcf --diff ug.chr2R.vcf --out hc.ug.compare
```

Vcftools can also compute

- LD statistics
- Fst between populations

Word of caution

All call optimization effort in GATK directed towards detection and removal of sequencing errors and small alignment errors

Reference genome assumed to be adequate (similar to those of re-sequenced individuals), i.e., reads assumed to be decently mapped to right locations possibly with small alignment ambiguities

Elaborate GATK pipeline will not help in case of massive misalignments (reads mapping to completely wrong locations) resulting from large diversity

What to do then?

Filter raw set of variants (most of them wrong) based on data for a large population (if you have one)

Identity by Descent (IBD):	exploit local identity within pairs of samples
local Linkage Disequilibrium (LD):	true variant should be in LD with nearby ones