GATK Hands-On Tutorial:
Variant Discovery with GATK

This GATK workshop focuses on key steps of the GATK Best Practices for Variant Discovery.

Our main purpose in this tutorial is to show you **how to examine and pre-process datasets of different experimental designs**, including whole genome (WGS), exome (ICE and Nexome), and RNAseq data for use in variant discovery analysis, and **how to call variants in DNA and in RNAseq** with HaplotypeCaller and other tools.

Along the way we will also demonstrate some important features of the HaplotypeCaller, including the **local assembly of haplotypes and realignment of reads** which enables it to produce superior indel calls compared to position-based callers such as the UnifiedGenotyper. We'll also show you some tips and tricks for **troubleshooting variant calls** with HaplotypeCaller and the IGV genome browser.

We will also cover the **GVCF workflow for joint variant analysis** applied to a trio of DNA samples. Hopefully we will convince you that this workflow has **substantial practical advantages** over a joint analysis that is achieved by calling variants simultaneously on all samples, while producing **results that are just as good** or even better.

The tutorial dataset will be made available for public download from the GATK website here.

1 WORKING WITH DATASETS FROM DIFFERENT EXPERIMENTAL DESIGNS

1.1 Get familiar with the datasets in IGV

1.1.1 The genome reference: b37

We are using a version of the b37 human genome reference containing only chromosome 20, which we prepared specially for this tutorial in order to provide a reasonable bundle size for download. It is accompanied by its index and sequence dictionary.

```
▫ ref/
      ▫human_g1k_b37_20.fasta              genome reference
      ▫human_g1k_b37_20.fai.fasta          fasta index
      ▫human_g1k_b37_20.dict               sequence dictionary
```

When you first open up IGV, you need to start by loading a genome reference file, as shown in the screenshots below. IGV offers the possibility to load common references from a remote server, or a specific reference from a local file. Here we could load either our special chromosome 20-only version of b37 (`ref/human_g1k_b37_20.fasta`), or just use the b37 human reference available on the IGV server -- both will work. In this tutorial we will use the custom 20-only reference for GATK commands but in IGV we will use the IGV server's b37 genome so that we can see the gene track that comes preloaded with it.

This is how you load a genome: use "from File" to load our custom reference, or "from Server" to load the Human (1kg, b37+decoy) genome from the IGV server.

1.1.2 The test sample: NA12878

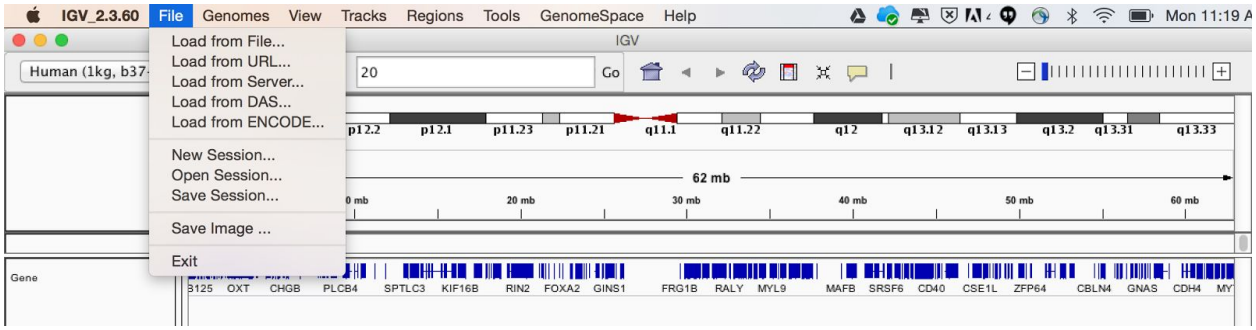
The biological sample from which the example sequence data was obtained comes from individual NA12878, a member of a 17 sample collection known as *CEPH Pedigree 1463*, taken from a family in Utah, USA. This trio is often referred to as the CEU Trio and is widely used as an evaluation standard (*e.g.* in the Illumina Platinum Genomes dataset). Note that an alternative trio constituted of the mother (NA12878) and her parents is often also referred to as a CEU Trio. Our trio corresponds to the 2nd generation and one of the 11 grandchildren.

In this tutorial we will work with the following BAM files derived from NA12878: (1) DNA dataset generated by paired end 250 bp whole genome sequencing (WGS) on Illumina HiSeqX and fully pre-processed according to the GATK Best Practices for germline DNA (`NA12878_wgs_20.bam`); (2) RNAseq dataset generated by paired end 75 bp transcriptome sequencing on Illumina HiSeqX and aligned using STAR 2-pass according to the GATK Best Practices for RNAseq (`NA12878_rnaseq_20.bam`); and (3) the same RNAseq dataset fully pre-processed according to the GATK Best Practices for RNAseq (`NA12878_rnaseq_pp_20.bam`). All three BAM files are accompanied by an index file.

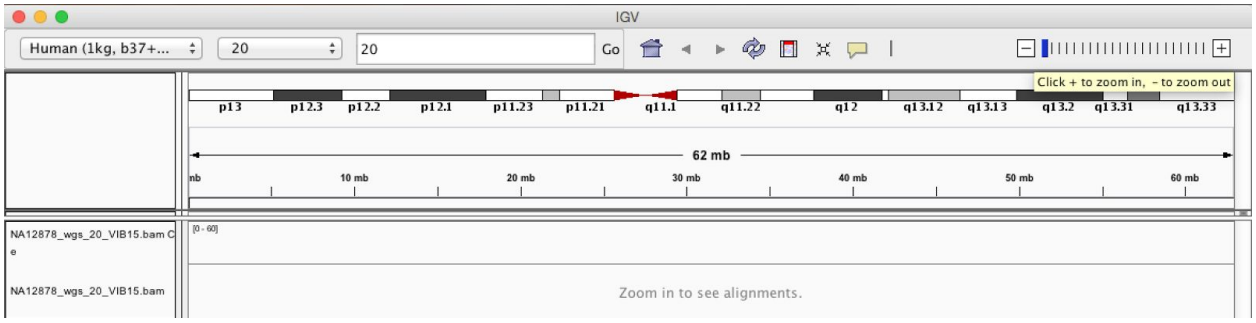
The sequence data files have been specially prepared as well to match our custom chromosome 20-only reference. They only contain data on chromosome 20, in two pre-determined intervals of interest ranging from positions 20:10,000,000-10,200,000 and 20:15,800,000-16,100,00 in the interest of keeping file sizes down.

- `bams/exp_design/`
  - `NA12878_wgs_20.bam` — DNA WGS fully pre-processed
  - `NA12878_ICE_20.bam` — ICE exome fully pre-processed
  - `NA12878_NEX_20.bam` — NEXOME fully pre-processed
  - `NA12878_rnaseq_20.bam` — RNAseq aligned with STAR
  - `NA12878_rnaseq_pp_20.bam` — RNAseq fully pre-processed

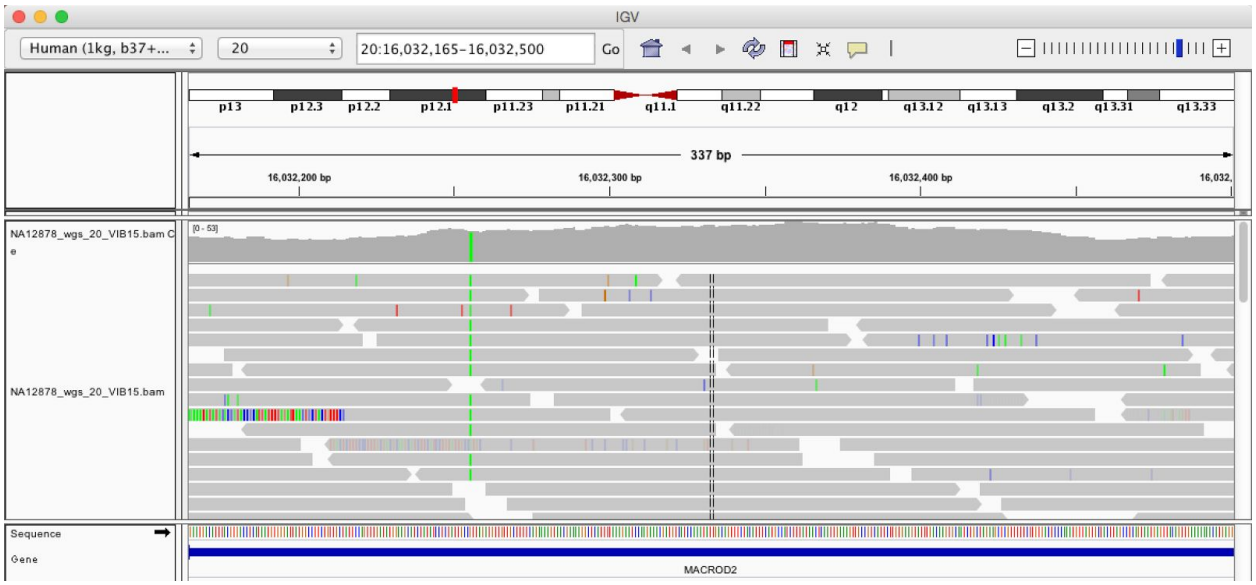
Let's start by loading the DNA WGS sample of NA12878 (`bams/exp_design/NA12878_wgs_20.bam`), as shown in the screenshots below.



Initially you will not see any data displayed. You need to zoom in to a smaller region for IGV to start displaying reads. You can do that by using the -/+ zoom controls, or by typing in some genome regions coordinates in the coordinates box (where it says "20" in the screenshot below).



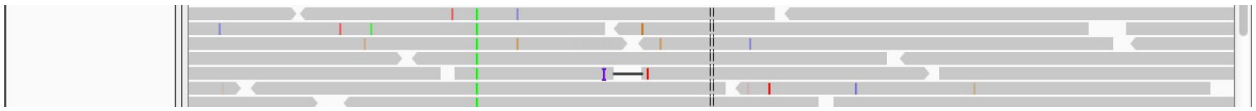
Here, we'll zoom into a predetermined interval of interest: 20:16,032,165-16,032,500. Once you hit the `Go` button, you should see something like this:



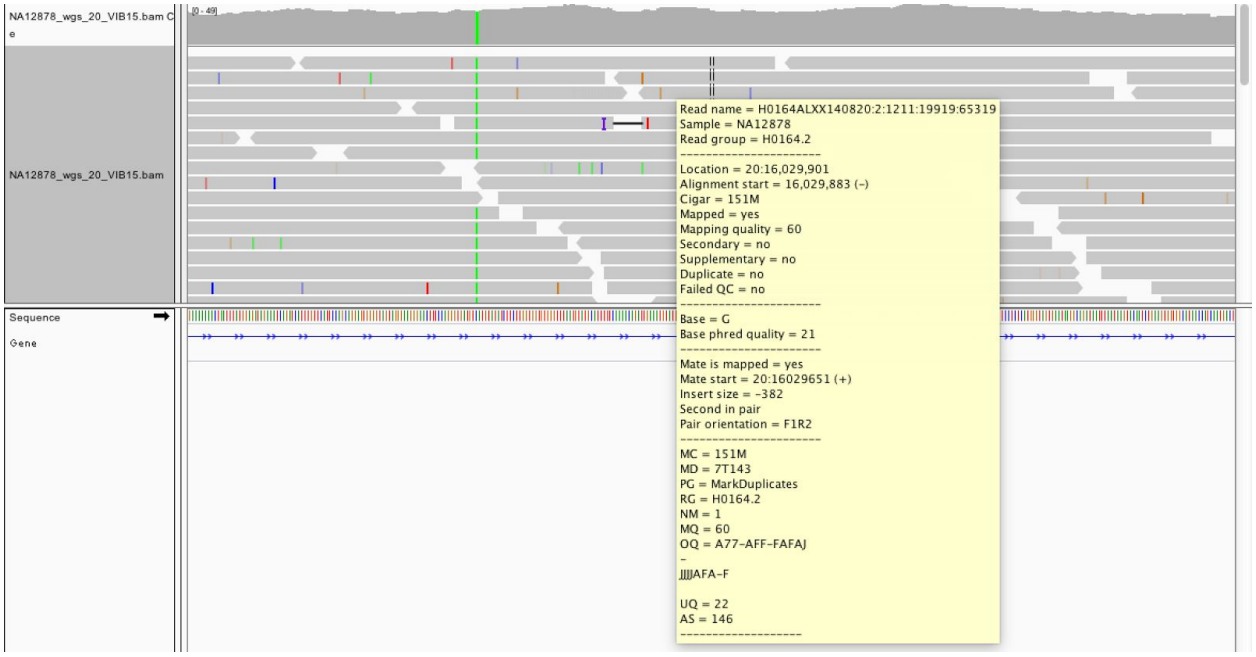
The top track shows depth of coverage, ie the amount of sequence reads present at each position.

The mostly grey horizontal bars filling the viewport are the reads. Grey means that those bases match the reference, while colored stripes or base letters (depending on your zoom level) indicate mismatches.

If you jump to another predetermined interval of interest: 20:16,029,744-16,030,079 you will see some reads with mapping insertions and deletions, indicated by purple symbols and crossed-out gaps, respectively.

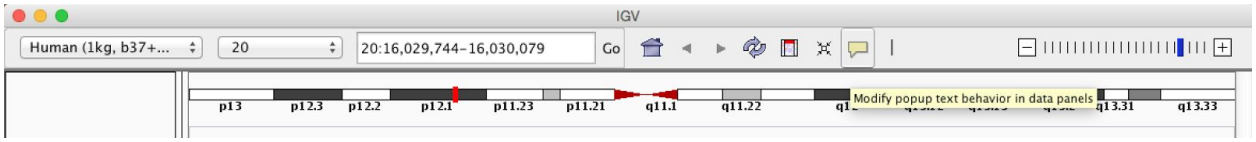


By default, when your pointer hovers over a read, IGV will display a box containing all the information encoded in that read's SAM record, as shown below.

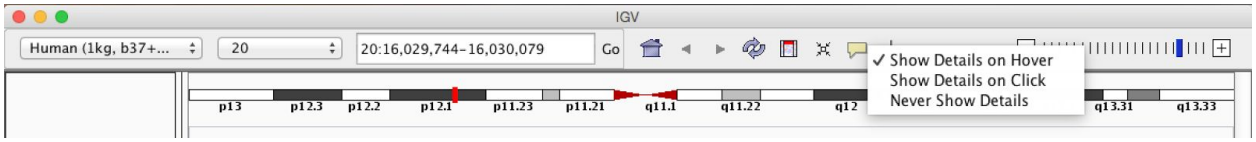


This is a very useful feature when you're troubleshooting a difficult region because you can easily see all the properties of each read instead -- much easier than looking at the BAM files directly with samtools.

However while the popup behavior may seem convenient at first, it gets annoying really quickly, so you can change it to display the SAM record only when you actually click a read. You do this using the yellow speech bubble icon in the tool menu above the viewport:



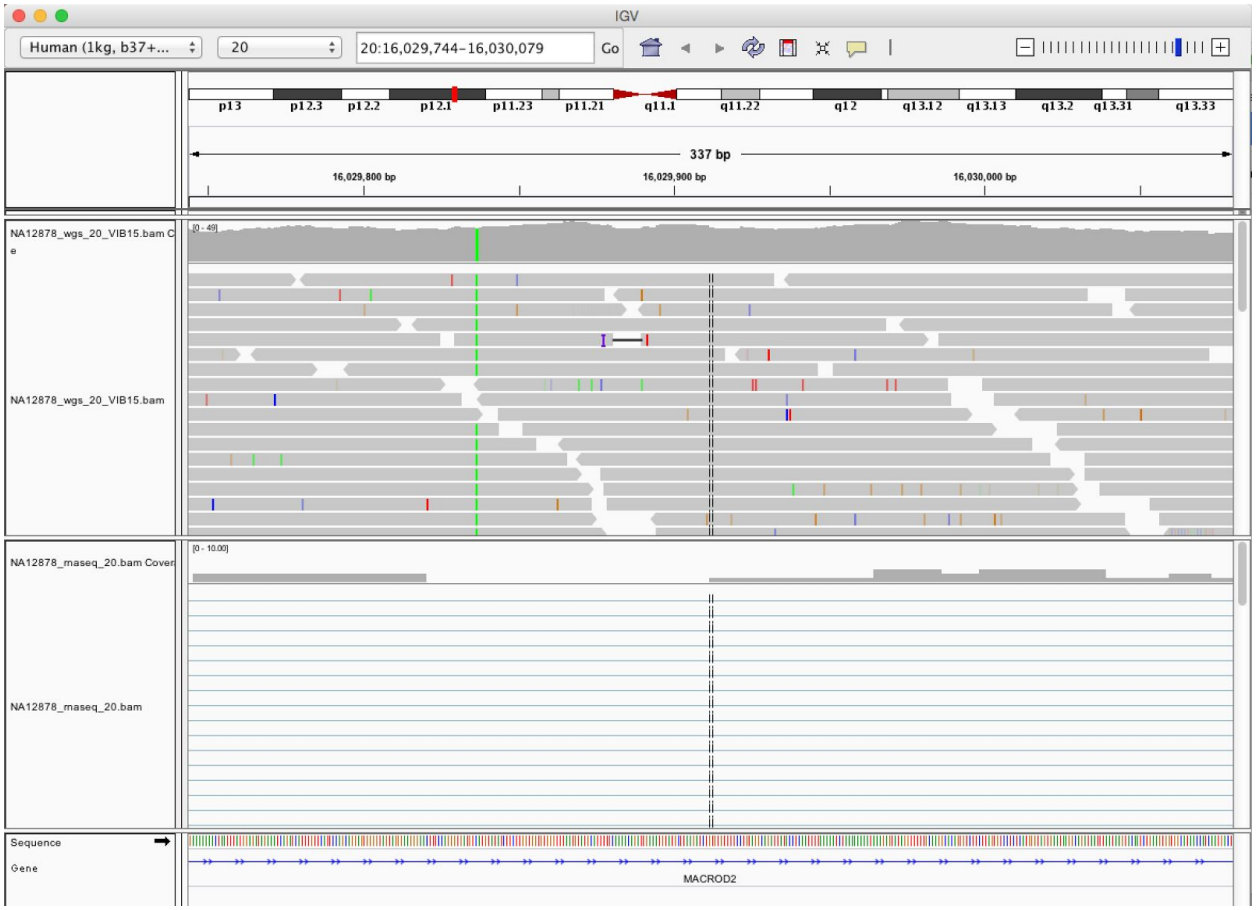
Click on it and change the selection from "Show details on Hover" to "Show details on Click".



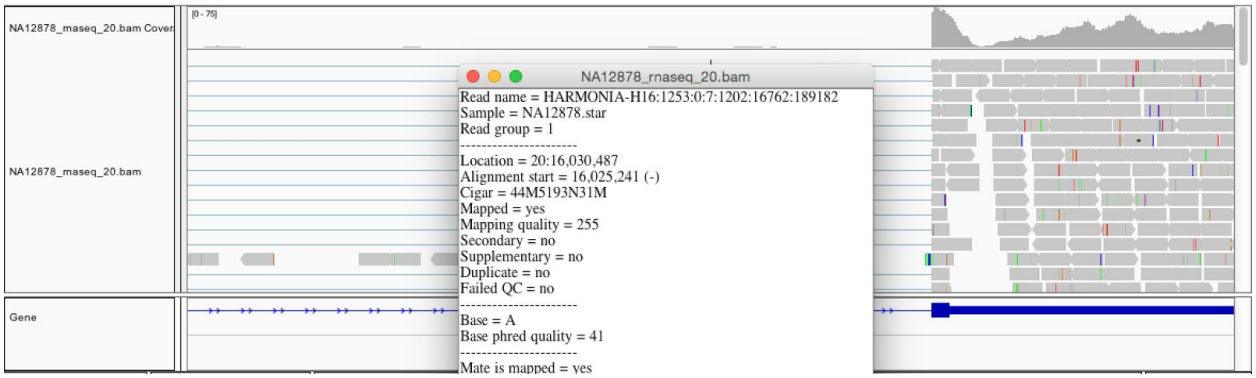
Once you've done that, the detailed information will only be shown if you click on a read (or variant in the variant track). When you do, the info will be shown in a window box that you have to close manually to make it go away.

NEWLY ADDED BIT (details not in worksheet -- be sure to follow along!)
We load the two exome datasets we have in addition to the WGS in the bams/exp_design/ directory. Look at [20:15,873,697-15,875,416] in collapsed view. We look especially at coverage distribution.

Next, let's load the aligned (but not fully processed) RNAseq dataset that we have for NA12878 (`NA12878_rnaseq_20.bam`). Just use the File menu and select "Load from File" just like we did for the DNA BAM file. It will be loaded as a new track below the previous one.



If you're still in the second interval we looked at, you will only see pale blue lines instead of reads. This is because it's an intronic region! Zoom out a few times to see the coverage in the nearby exon region. The blue lines connect to reads that are located in the exon. Click on one to see the N operator in the CIGAR string: in the example below, 44M5193N31M indicates that the read covers a 5193 bp intron.



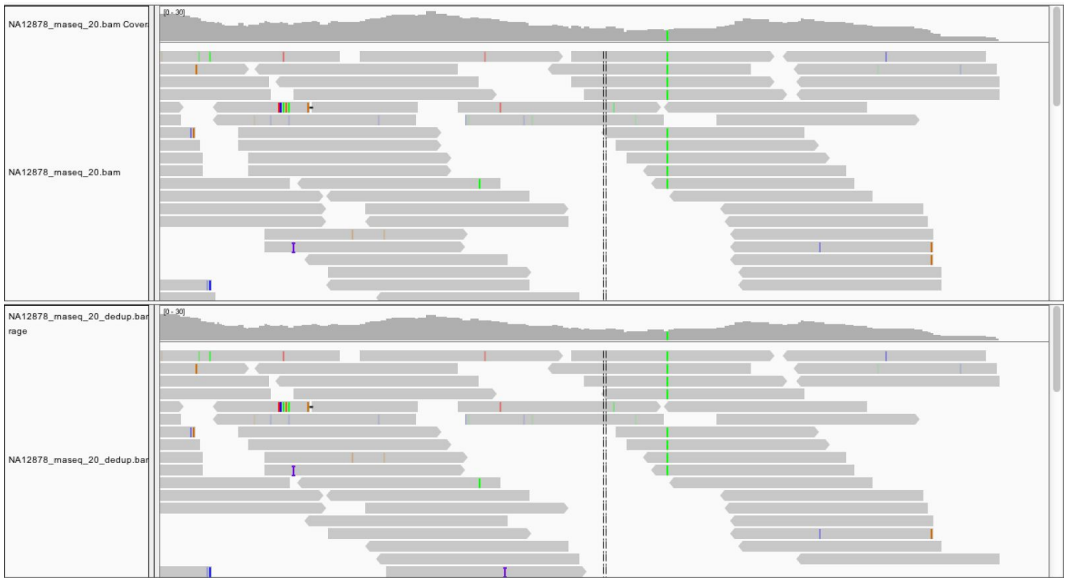
2 PRE-PROCESSING EXERCISES

2.1 Mark duplicates

Run Picard MarkDuplicates to identify duplicates and mark them so that GATK tools will automatically ignore them (they are not removed from the dataset unless requested).

```
java -jar picard.jar MarkDuplicates \
      I=bams/exp_design/NA12878_rnaseq_20.bam \
      O=sandbox/NA12878_rnaseq_20_dedup.bam \
      CREATE_INDEX=true \
      M=output.metrics
```

Now let's load the output BAM in IGV. Because of how IGV tiles reads, the difference between the original and the "dedupped" data is easiest to see at the end of a covered region, so let's navigate to the interval 20:16,033,512-16,033,847 using the coordinates box.



The metrics file that is produce by MarkDuplicates contains various statistics about how much duplication was observed (here, about 6% overall which is very reasonable), plus a histogram that can be used to plot duplication rate.

```
## METRICS CLASS      picard.sam.DuplicationMetrics

LIBRARY UNPAIRED_READS_EXAMINED      READ_PAIRS_EXAMINED   UNMAPPED_READS
UNPAIRED_READ_DUPLICATES     READ_PAIR_DUPLICATES  READ_PAIR_OPTICAL_DUPLICATES
PERCENT_DUPLICATION   ESTIMATED_LIBRARY_SIZE

bar     0       23902   0       0       1439    830     0.060204        429317
```

2.2 Split'N'Trim (RNAseq only)

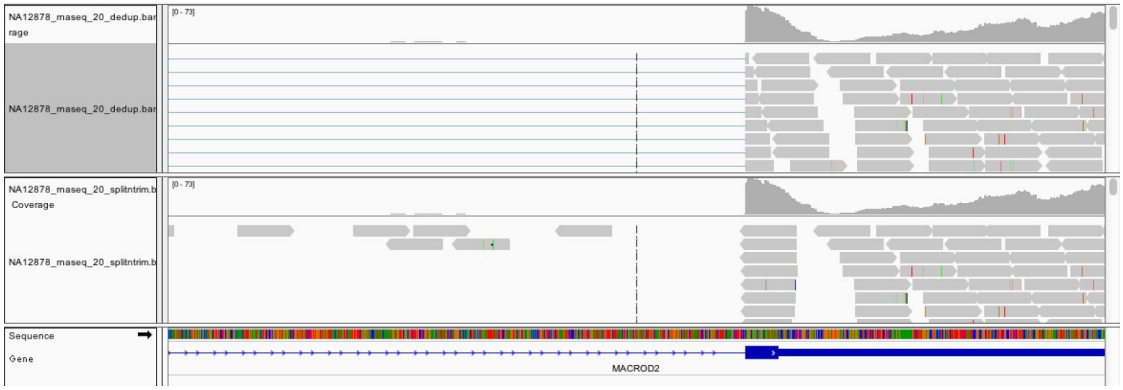
Next, we use a GATK tool called SplitNCigarReads developed specially for RNAseq, which splits reads into exon segments (getting rid of Ns but maintaining grouping information) and hard-clip any sequences overhanging into the intronic regions.

At this step we also add one important tweak: we need to reassign mapping qualities, because STAR assigns good alignments a MAPQ of 255 (which technically means "unknown" and is therefore meaningless to GATK). So we use the GATK's ReassignOneMappingQuality read filter to reassign all good alignments to the default value of 60. This is not ideal, and we hope that in the future RNAseq mappers will emit meaningful quality scores, but in the meantime this is the best we can do. In practice we do this by adding the ReassignOneMappingQuality read filter to the splitter command.

Finally, be sure to specify that reads with N cigars should be allowed. This is currently still classified as an "unsafe" option due to some GATK legacy code, but this classification will change to reflect the fact that this is now a supported option for RNAseq processing.

```
java -jar GenomeAnalysisTK.jar -T SplitNCigarReads \
      -R ref/human_g1k_b37_20.fasta \
      -I sandbox/NA12878_rnaseq_20_dedup.bam \
      -o sandbox/NA12878_rnaseq_20_splitntrim.bam \
      -rf ReassignOneMappingQuality -RMQF 255 -RMQT 60 \
      -U ALLOW_N_CIGAR_READS
```

Once that has run, we load the output BAM in IGV and jump to coordinates 20:16,029,645-16,030,991. We see that the reads that previously spanned the intronic regions have been split into sets of separate reads that border the intron. The split reads are hard-clipped; this can be seen in the CIGAR strings.



2.3 Indel Realignment & BQSR [not done hands-on]

Both steps are run the same way for DNA and RNAseq (with the same known sites resource files), without any special arguments. We do not run this in hands-on tutorials because the algorithm needs to see a lot of data -- more than can be processed in the small amount of time available in a tutorial.

3 VARIANT DISCOVERY

3.1 HaplotypeCaller basics: effect of HC assembly / realignment step

Before we tackle the specifics of calling variants on RNAseq, let's go through the basic operation and options of HaplotypeCaller as applied to discovery of germline variants in DNA.

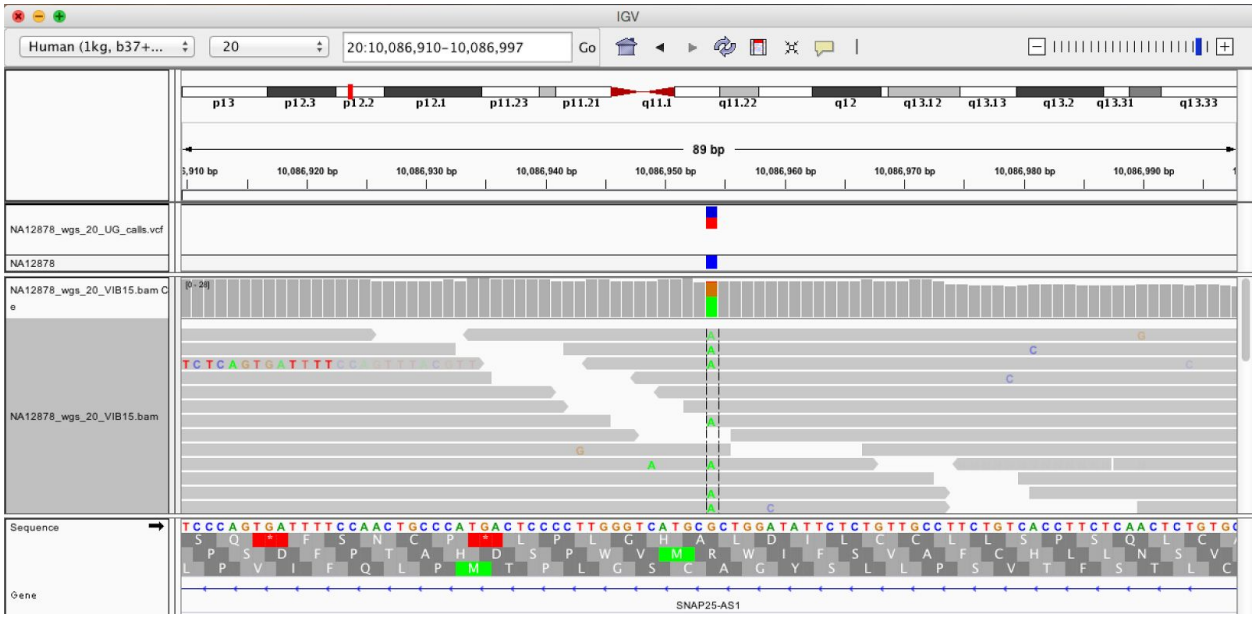
3.1.1 Call variants with a position-based caller: UnifiedGenotyper

The UnifiedGenotyper is an older variant caller that operates per genome position, essentially the same way as Samtools mpileup. It calls SNPs and indels separately. We don't recommend using UG anymore because it produces inferior results to more recent tools, but it makes a good starting point for learning.

Run this basic UG command:

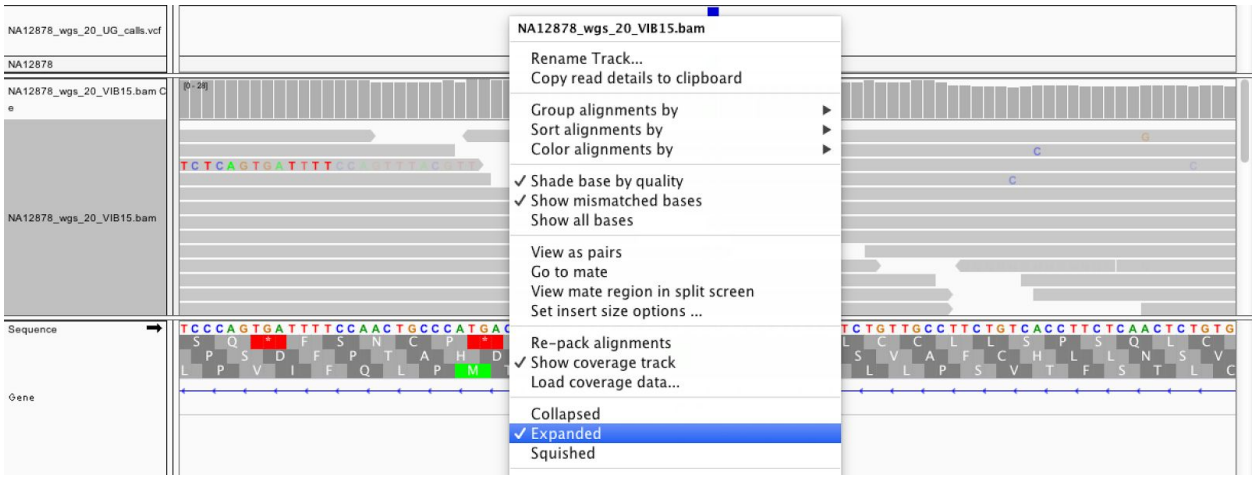
```
java -jar GenomeAnalysisTK.jar -T UnifiedGenotyper \
      -R ref/human_g1k_b37_20.fasta \
      -I bams/exp_design/NA12878_wgs_20.bam \
      -o sandbox/NA12878_wgs_20_UG_calls.vcf \
      -glm BOTH \
      -L 20:10,000,000-10,200,000
```

Load the BAM file (`bams/exp_design/NA12878_wgs_20.bam`) and the output VCF, which contains the variant calls (`sandbox/NA12878_wgs_20_UG_calls.vcf`) in IGV using the same menu command as we did earlier for BAM files. Jump to coordinates 20:10,086,910-10,086,997 and look at this randomly chosen heterozygous G/A SNP variant call.

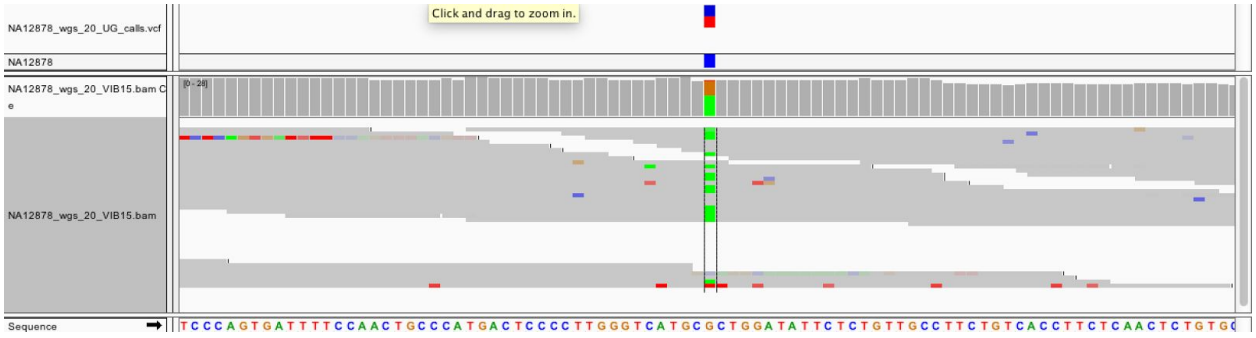


The variant track shows the site-level call information first, then each sample represented in the VCF is on a separate line below, if there are multiple samples (but we're only showing a single sample in this case). As with read data, you can click on a variant to have IGV display the details of the variant call.

The supporting information for this call looks fine, but sometimes we want to get a sense of what all the read data in the region looks like, which can be difficult with the default IGV view parameters, unless you have a huge monitor. For a different view option, right-click the area with the reads of the file name on the left to bring up a menu with some options, and choose "Collapsed" or "Squished" instead of "Expanded" (bottom of the screenshot, about halfway down the popup menu):

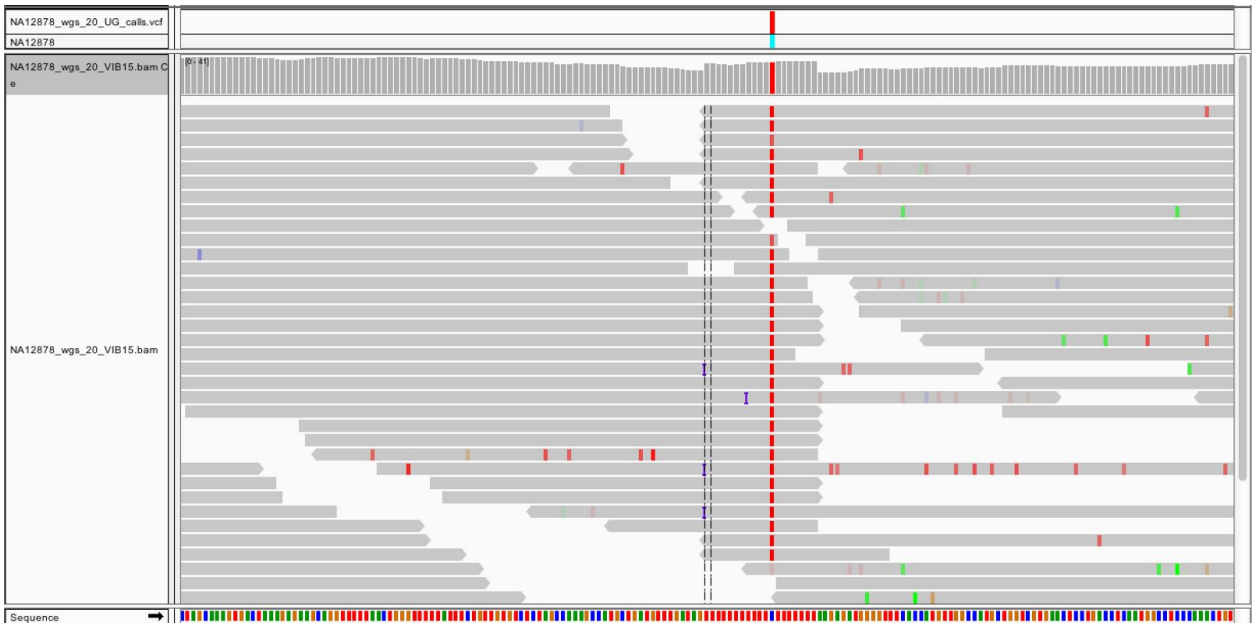


Now the view should look something like this:



This gives us a better overview of what the data looks like in this region: good even coverage, not too much noise in the region, and reasonable allele balance (half and half). This is probably a true heterozygous SNP and we can expect it to pass filters later on (unless the filters uncover some biases or flaws that are not immediately obvious in IGV). Position-based callers like UnifiedGenotyper tend to produce good results on easy sites like this.

Now let's jump to coordinates 20:10,002,371-10,002,546 to look at a different kind of site.



It looks like there's a nice clean homozygous C/T SNP, slightly right of center in the viewport. For the rest, we see quite a bit of noise, probably related to the homopolymer runs (see the sequence below) but nothing else that looks convincing. Accordingly, UG doesn't call any other variants here. Well, we'll see in the next exercise that there's actually more than meets the eye in this region.

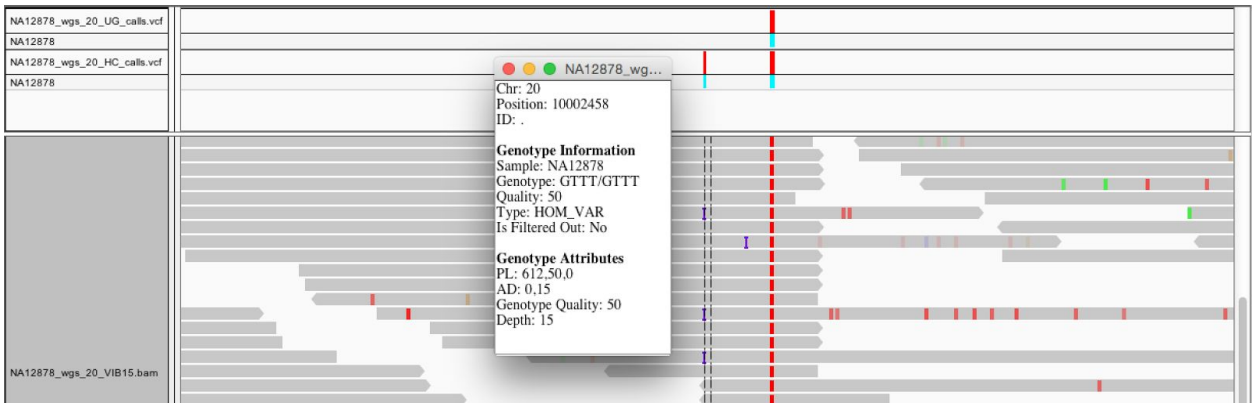
3.1.2 Call variants with HaplotypeCaller

The HaplotypeCaller is a more sophisticated variant caller that calls SNPs and indels simultaneously via local de-novo assembly of haplotypes in an active region. In other words, whenever the program encounters a region showing signs of variation, it discards the existing mapping information and completely reassembles the reads in that region. This allows the HaplotypeCaller to be more accurate when calling regions that are traditionally difficult to call, for example when they contain different types of variants close to each other. It also makes the HaplotypeCaller much better at calling indels than position-based callers like UnifiedGenotyper.

Run this basic HC command:

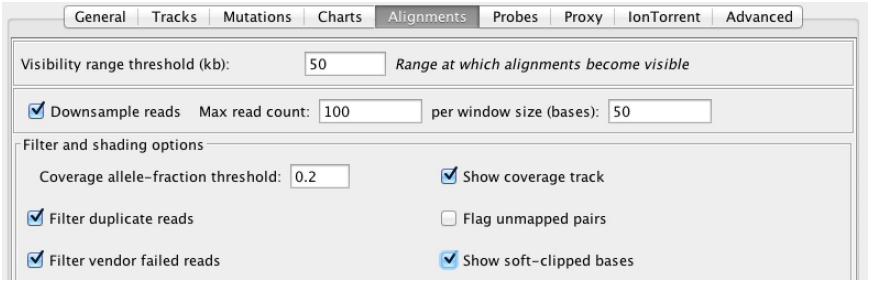
```
java -jar GenomeAnalysisTK.jar -T HaplotypeCaller \
      -R ref/human_g1k_b37_20.fasta \
      -I bams/exp_design/NA12878_wgs_20.bam \
      -o sandbox/NA12878_wgs_20_HC_calls.vcf \
      -L 20:10,000,000-10,200,000 -L 20:15,800,000-16,100,000
```

Load the output VCF (`sandbox/NA12878_wgs_20_HC_calls.vcf`) in IGV, still zoomed in on coordinates 20:10,002,371-10,002,546.

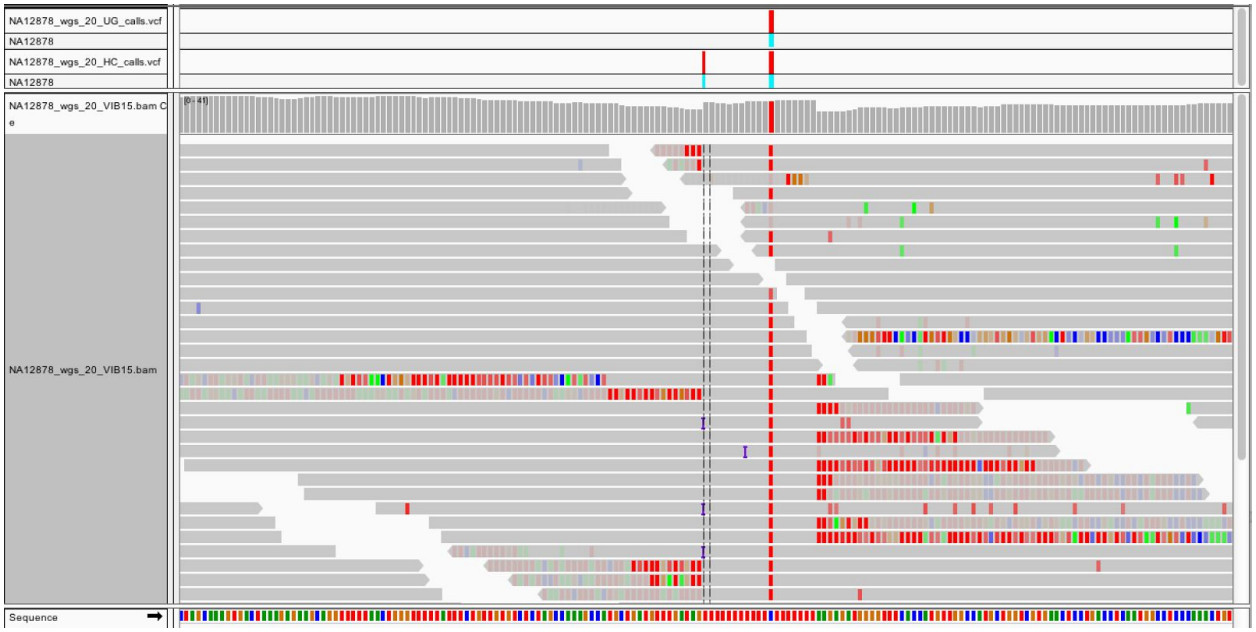


We see that HC called the same C/T SNP as UG, but it also called another variant, a homozygous variant insertion of three T bases. How is this possible when so few reads seem to support an insertion at this position?

Pro Tip: when you encounter indel-related weirdness, turn on the display of soft-clips, which IGV turns off by default. Go to View > Preferences > Alignments and select “Show soft-clipped bases”:



Once you've turned on the soft-clips display, the region lights up like a Christmas tree.



This tells us that the aligner (here, BWA mem) had a lot of trouble mapping reads in this region, and it suggests that the HaplotypeCaller may have seen things differently after performing its local graph assembly step.

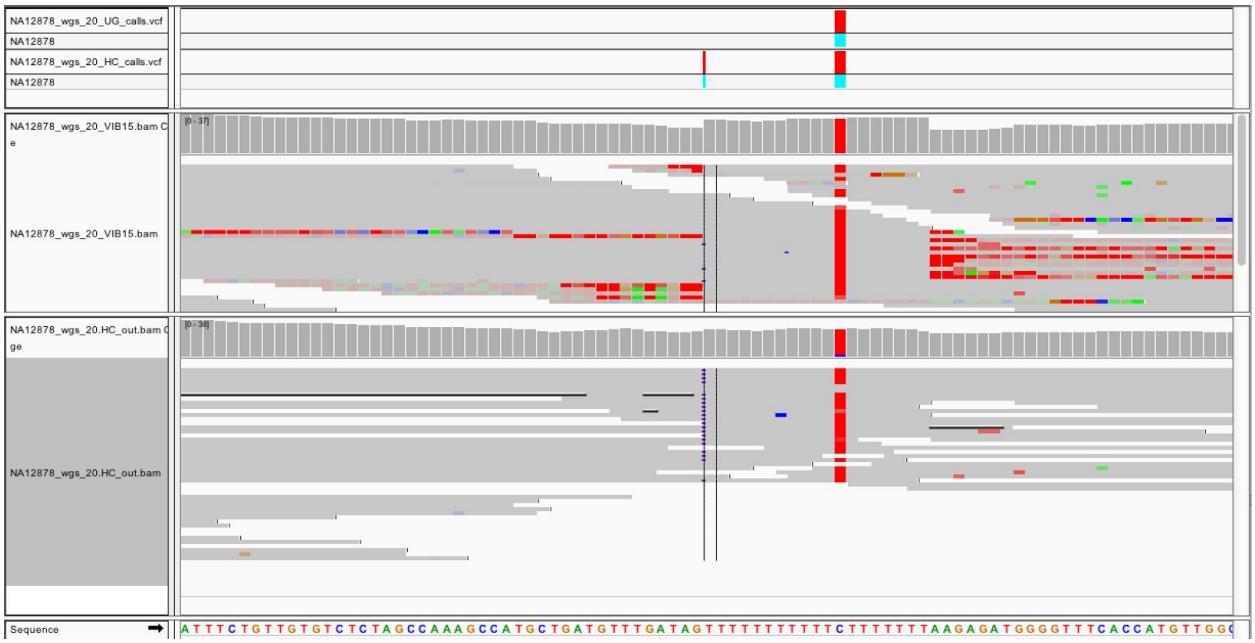
3.1.3 View realigned reads and assembled haplotypes

The assembly and realignment step performed by HC aims to correct mapping errors made by the original aligner, and it is what enables HaplotypeCaller to make more accurate indel calls than UnifiedGenotyper. We can view how the reads have been realigned using the `-bamout` argument. The output of the `-bamout` argument is a bam file containing the realigned reads. You can compare the bamout file to the original bam file and diagnose unexpected calls.

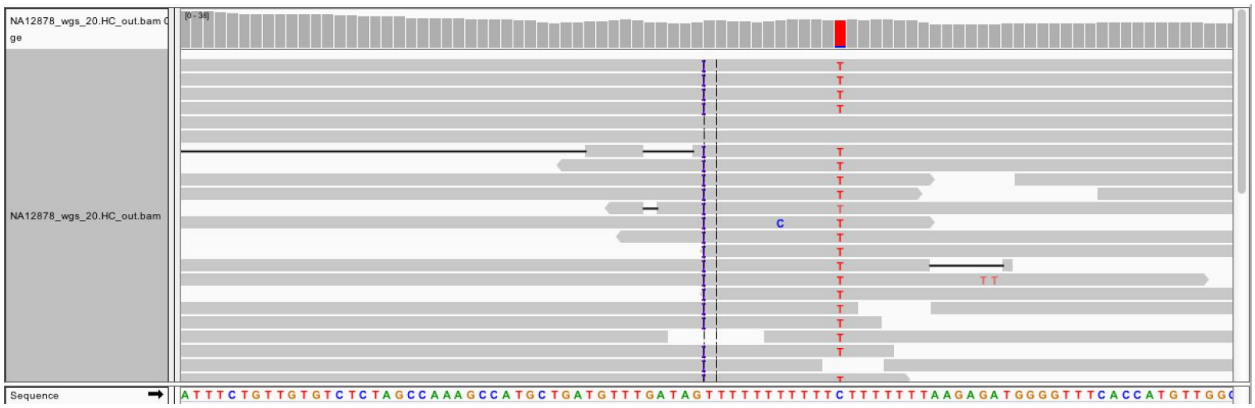
Run HaplotypeCaller with debugging arguments:

```
java -jar GenomeAnalysisTK.jar -T HaplotypeCaller \
       -R ref/human_g1k_b37_20.fasta \
       -I bams/exp_design/NA12878_wgs_20.bam \
       -o sandbox/NA12878_wgs_20_HC_calls_debug.vcf \
       -bamout sandbox/NA12878_wgs_20.HC_out.bam \
       -forceActive -disableOptimizations \
       -L 20:10,002,371-10,002,546 -ip 100
```

Load the output BAM (`sandbox/sandbox/NA12878_wgs_20.HC_out.bam`) in IGV, still zoomed in on coordinates 20:10,002,371-10,002,546. Let's view the data in Collapsed mode to see the overall picture better.



After realignment by HaplotypeCaller, almost all the reads show the insertion, and the region in general is a lot cleaner.

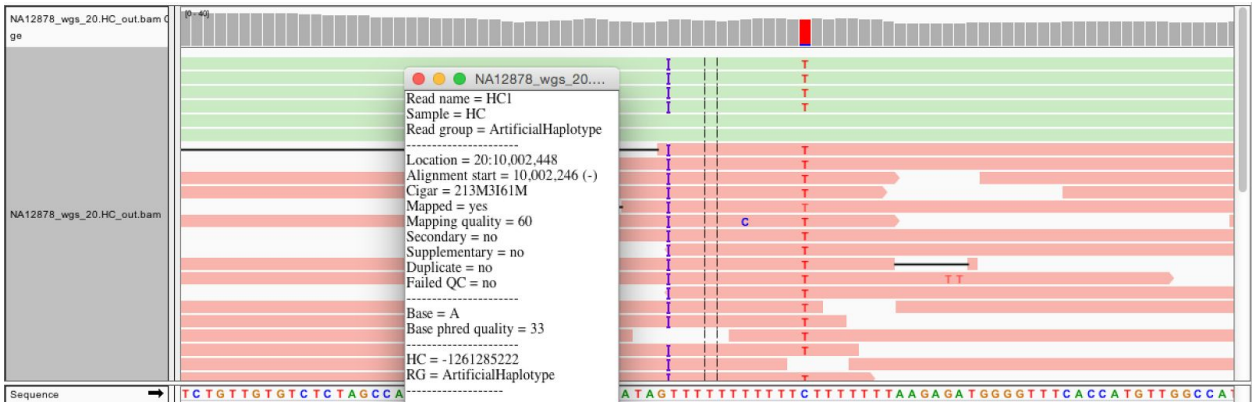


If we expand the reads in the output BAM, we see that all the insertions are in phase with the C/T SNP. When HC is run in -ERC GVCF mode, the relevant phasing information will be output by default.

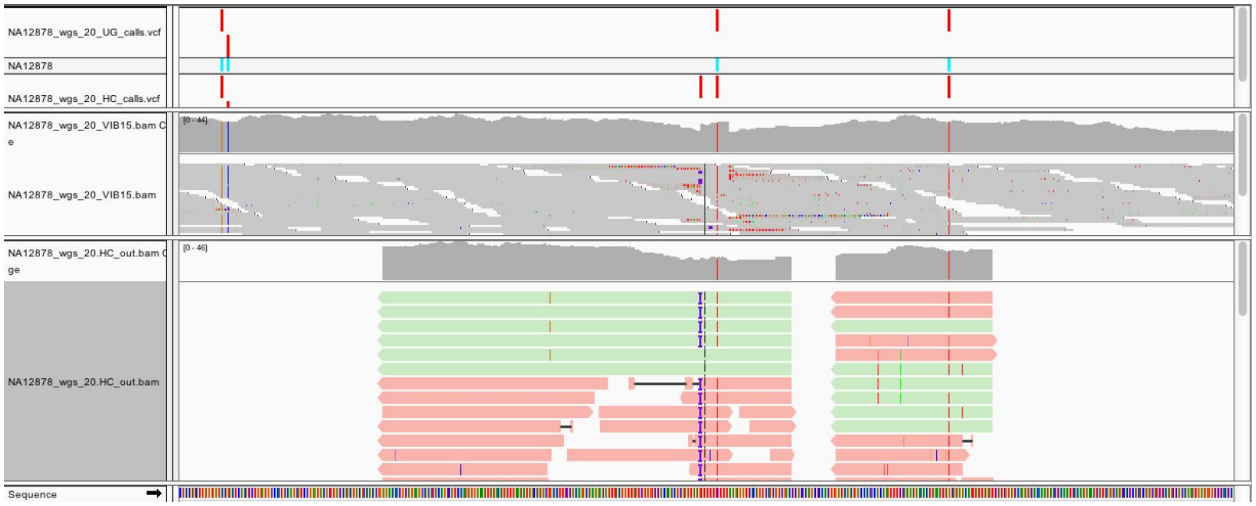
Note that some of the first set of reads shown at the top of the pile are not real reads, they are artificial haplotypes constructed by HaplotypeCaller. They are tagged with a special read group identifier ("Artificial Haplotype") so that they can be recognized and visualized in IGV. To do so, right-click on the reads area to bring up the view options menu, select "Color alignments by", and select "read group". You can also group and/or sort the alignments by sample.

This makes it obvious which are the artificial haplotypes (here, the green reads). We can click on one to see details.

Note that it is possible to change most of IGV color settings if desired for better differentiation (e.g. for colorblind people).



We see that HaplotypeCaller considered six possible haplotypes, because there is one more called variant in the same ActiveRegion, which we can see by zooming out a bit further -- in fact we can see that two ActiveRegions were examined within the scope of the interval we provided (with padding).



3.2 Joint analysis of multiple DNA samples via GVCF workflow

Starting in GATK version 3 and above, the GATK includes a workflow that is optimized for calling variants in multiple samples efficiently and incrementally. This workflow was developed because running HaplotypeCaller on multiple samples scales very badly (because the computational complexity increases exponentially) so it will take a very long time to run, and it may even time out. This is a problem in itself. To make matters worse, you may be able to finish running HaplotypeCaller on all your samples, but then, what happens if you get another sample to add to your cohort? You will have to run HaplotypeCaller again on all your samples, from scratch! This costs time and money. The so-called GVCF workflow solves both of these problems. In this section, we demonstrate how to apply the GVCF workflow by calling variants per-sample then performing joint genotyping to generate a final VCF for the trio.

The first step in the GVCF workflow is to run HaplotypeCaller in GVCF mode on each individual bam file. This is basically running HaplotypeCaller in normal mode, but with `-ERC GVCF` added to the command. We will only run HaplotypeCaller in GVCF mode on the NA12878 bam. In the interest of time, we have supplied the other sample GVCFs in the bundle, but normally you would run them individually in the same way as the first.

3.2.1 Run HaplotypeCaller on a single bam file in GVCF mode

```
java -jar GenomeAnalysisTK.jar \
      -T HaplotypeCaller \
      -R ref/human_g1k_b37_20.fasta \
      -I bams/NA12878_wgs_20.bam \
      -o sandbox/NA12878_wgs_20.g.vcf \
      -ERC GVCF \
      -L 20:10000000-10200000
```

3.2.2 View resulting GVCF file in the terminal

```
more sandbox/NA12878_wgs_20.g.vcf
```

Note the NON_REF allele defined in the header.

```
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
```

Also note the GVCF blocks defined in the header. You can tell how the reference blocks are grouped by GQ.

```
##GVCFBlock0-1=minGQ=0(inclusive),maxGQ=1(exclusive)
##GVCFBlock1-2=minGQ=1(inclusive),maxGQ=2(exclusive)
##GVCFBlock10-11=minGQ=10(inclusive),maxGQ=11(exclusive)
##GVCFBlock11-12=minGQ=11(inclusive),maxGQ=12(exclusive)
```

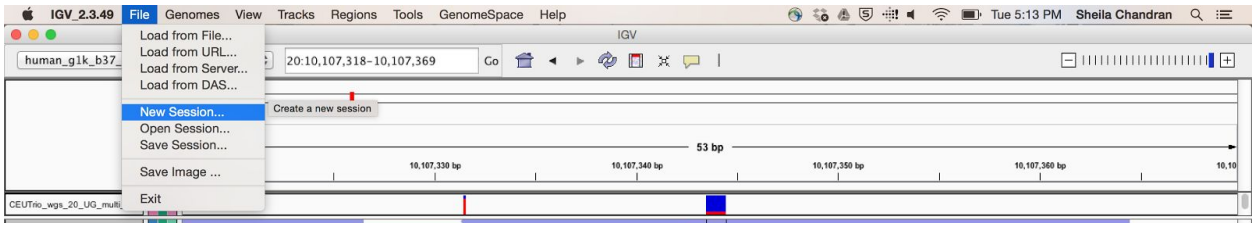
Finally, while scrolling through the records, we can see the reference blocks and variant sites

```
20      10000115        .       G       <NON_REF>       .       .       END=10000116    GT:DP:GQ:MIN_
DP:PL       0/0:25:69:25:0,69,1035
20      10000117        .       C       T,<NON_REF>     262.77  .       BaseQRankSum=-0.831;ClippingR
ankSum=-0.092;DP=23;MLEAC=1,0;MLEAF=0.500,0.00;MQ=60.47;MQRankSum=1.446;ReadPosRankSum=0.462        GT
:AD:DP:GQ:PL:SB       0/1:11,12,0:23:99:291,0,292,324,327,652:9,2,9,3
20      10000118        .       T       <NON_REF>       .       .       END=10000123    GT:DP:GQ:MIN_
DP:PL       0/0:25:63:24:0,63,945
```

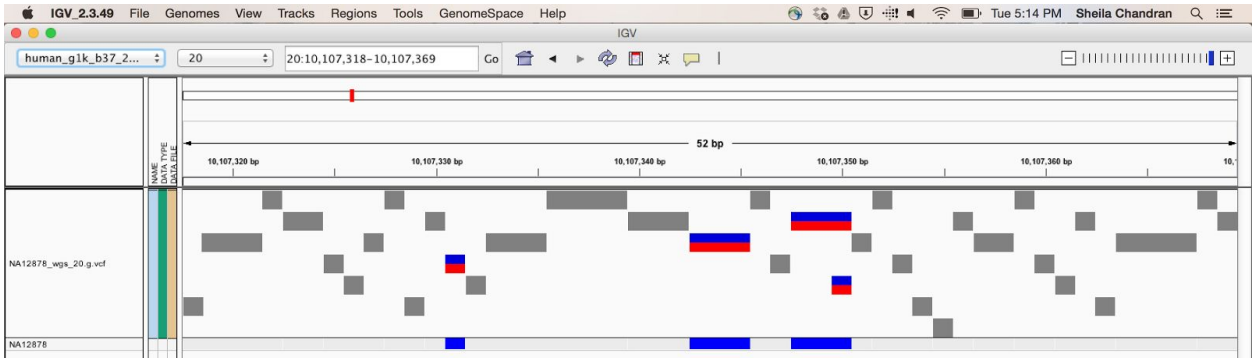
The GVCF is designed to be very sensitive to any potential variants, so they may be rescued if we see other samples having the same variant.

3.2.3 View variants in IGV

Because our IGV screen is full, let's start a new session.



Now, we can load the NA12878 GVCF and see something like this:



Notice anything different from the VCF? Along with the colorful variant blocks, we can see some gray blocks in the GVCF. Those gray blocks represent the non-variant blocks in the GVCF. Notice some of the gray blocks are next to each other, but are not grouped together. This is because they belong to different GQ blocks.

Now, let's add the other two samples GVCFs (previously generated) to our NA12878 GVCF. To perform joint genotyping on all the GVCFs, we run Genotype GVCFs. Do you think the final vcf will be the same as the final vcf from Haplotype Caller in normal mode?

3.2.4 View GVCFs of CEU Trio samples (already generated previously) in IGV

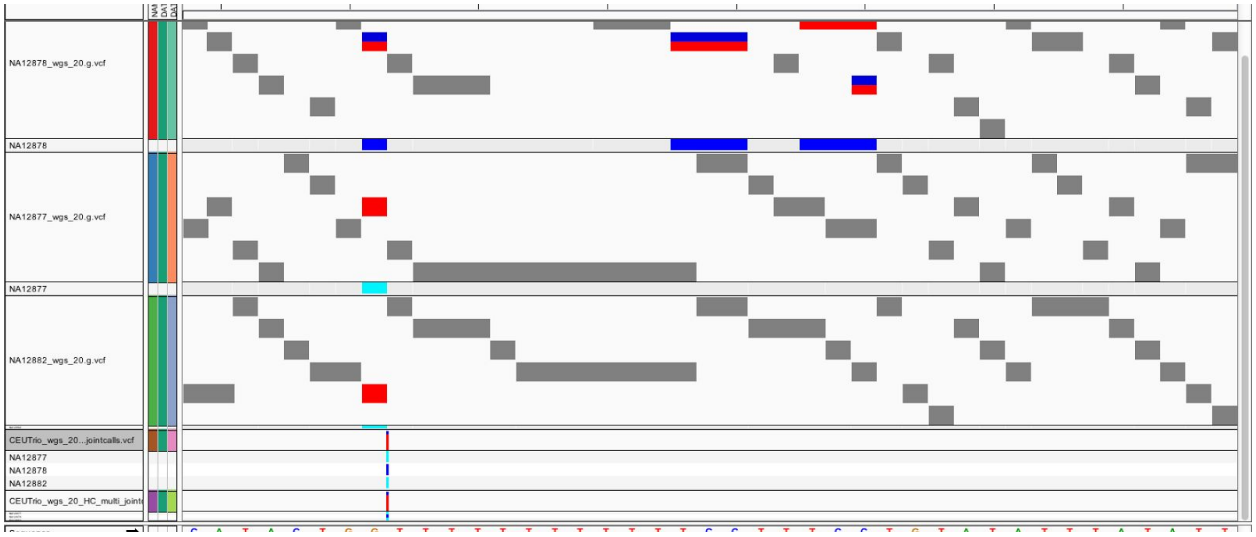
It looks like NA12878 has some potential variants that the other two samples do not have. Let's find out if they make it from the GVCF to the VCF. Remember, the GVCF is very sensitive to any potential variants even if we do not have much confidence in them. The final vcf will only contain variants we have high confidence in.

3.2.5 Run joint genotyping on the CEU Trio GVCFs to generate the final VCF

```
▫ java -jar GenomeAnalysisTK.jar \
      -T GenotypeGVCFs \
      -R ref/human_g1k_b37_20.fasta \
      -V sandbox/NA12878_wgs_20.g.vcf \
      -V gvcfs/NA12877_wgs_20.g.vcf \
      -V gvcfs/NA12882_wgs_20.g.vcf \
      -o sandbox/CEUTrio_wgs_20_GGVCFs_jointcalls.vcf \
      -L 20:10000000-10200000
```

3.2.6 View variants in IGV and compare callsets

Load the final VCF from HaplotypeCaller in normal mode and GenotypeGVCFs. You should see something like this. The final VCFs are at the bottom.



Notice the final VCFs from GenotypeGVCFs and from HaplotypeCaller run in normal multisample mode contain the same calls.

The final VCFs obtained from GenotypeGVCFs and HaplotypeCaller run in multisample mode are essentially equivalent (although there may be some marginal differences in borderline calls). The intermediate GVCF is very sensitive and reports all potential variants, even if we are not confident enough to emit them in the final VCF. The GVCF workflow allows for better scalability and ease of adding samples to your cohort when calling variants.

The next step would be to filter the resulting callset with VQSR (preferred) or hard-filtering methods (if VQSR is not possible).

3.3 Calling variants in RNAseq

The GVCF workflow has not yet been validated for use in RNAseq data so we do not cover it in this tutorial, but in theory there is no major technical obstacle to applying it to RNAseq in the same way as to DNA datasets.

In our tutorial we only show how to run HaplotypeCaller in its default mode, per-sample, on RNAseq data. There are only a few arguments to add, compared to the basic HC command, in order to run HaplotypeCaller on the RNAseq sample of NA12878.

The most important, "`-dontUseSoftClippedBases`" causes HC to disregard soft-clipped bases, which are often left overhanging splice junctions, despite the trimming operations done as part of the pre-processing. So applying it reduces the amount of false positive calls associated with these overhangs. Unfortunately, this does reduce HC's sensitivity to large insertions.

The other two, "`-stand_call_conf 20.0`" and "`-stand_emit_conf 20.0`" lower the default quality thresholds that are applied as a pre-filter before emitting variant calls to VCF. They are not specific to RNAseq but our analyses suggest that the amount of evidence found in RNAseq datasets per call on average is lower than what we see in WGS data, for which our default thresholds are tuned. These thresholds can be lowered further to boost sensitivity, at the expense however of specificity.

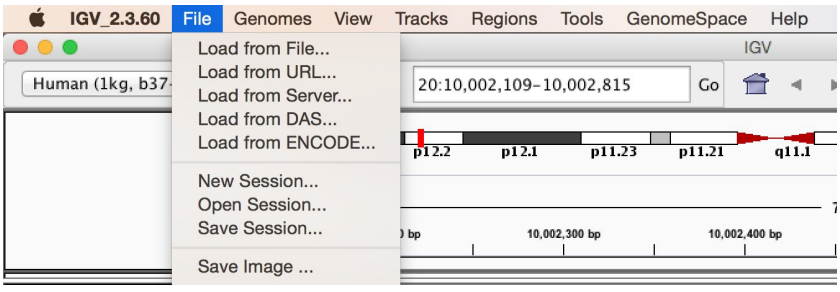
Run this HC command on the fully pre-processed RNAseq dataset :

```
java -jar GenomeAnalysisTK.jar -T HaplotypeCaller \
      -R ref/human_g1k_b37_20.fasta \
      -I bams/exp_design/NA12878_rnaseq_pp_20.bam \
      -o sandbox/NA12878_rnaseq_20_HC_calls.vcf \
      -L 20:15,800,000-16,100,000 \
      -dontUseSoftClippedBases \
      -stand_call_conf 20.0 -stand_emit_conf 20.0
```

The results will be output in the same way as for DNA datasets; you can open the VCF and compare the calls to the read data in the same way to get a first impression of what it looks like. There are no substantial differences in how you would work with the data. The more challenging part lies in interpreting results and filtering calls to identify variants of interest -- all of which should be done relative to a matched DNA sample if possible.

3.3.1 Comparing DNA and RNAseq results

Pro Tip: In IGV, start a new session to clean up the view quickly and easily. The reference selection and all our changes to the view preferences will be retained, as will the reference selection, but the datasets will be removed.

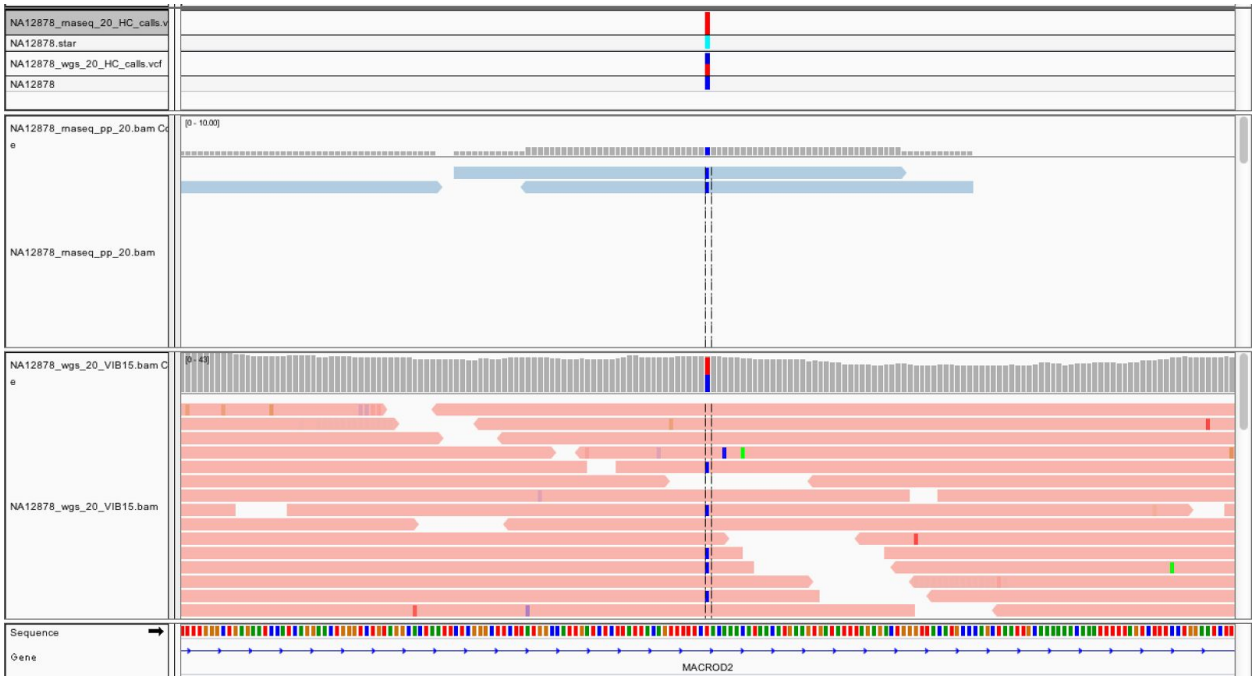


Load the fully pre-processed RNAseq dataset (`bams/exp_design/NA12878_rnaseq_pp_20.bam`) and the output VCF (`sandbox/NA12878_rnaseq_20_HC_calls.vcf`) in IGV, as well as the DNA dataset and the corresponding callset. Jump to coordinates 20:16,029,782-16,032,611, at the start of the last exon in the MACROD2 gene.

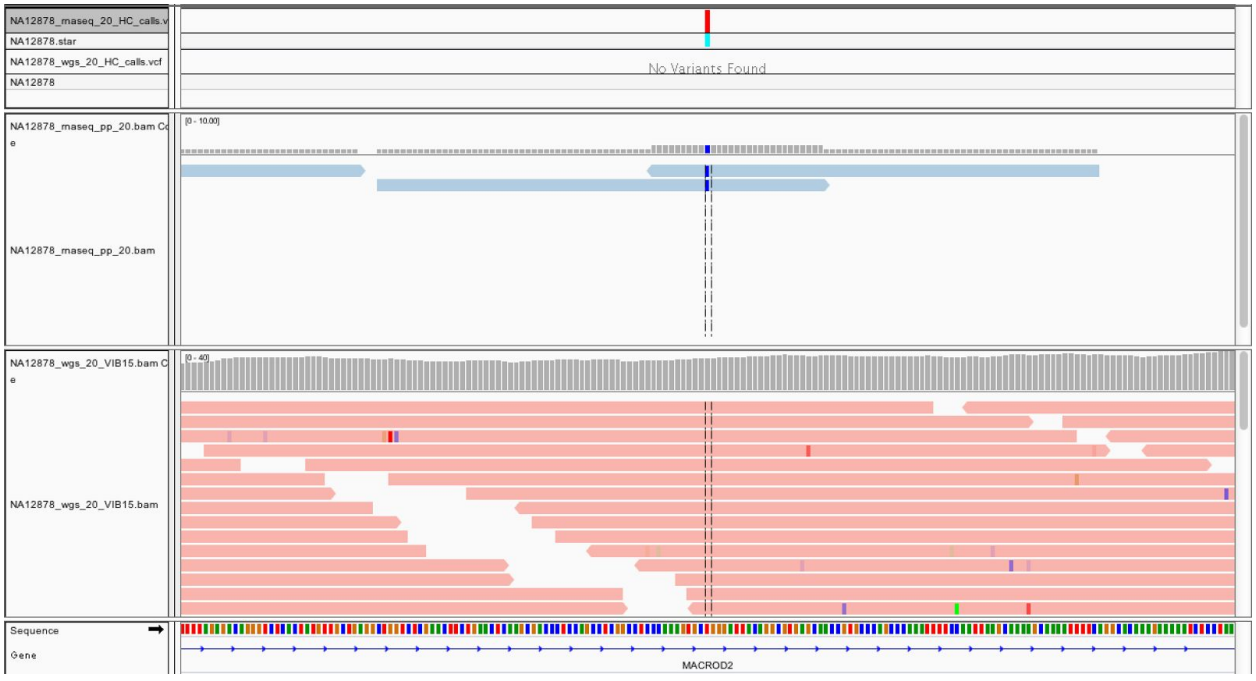
We see two variants in the expressed portion of the gene that is visible, both called homozygous in both the DNA and the RNAseq callsets. So at this level there's nothing really interesting going on in the RNAseq.

Now let's look at a few other positions. Caveat: some of these have very low coverage in the RNAseq, and are located in intronic regions, so simply could be artifacts.

20:15,962,110-15,962,285 - Different genotype calls



20:15,884,843-15,885,018 - Variant only in RNAseq



To make it easier to compare what is called in one set but not the other, we can use GATK's variant manipulation tools, CombineVariants and SelectVariants.

Run **CombineVariants** to produce a single VCF file with records annotated with set information:

```
java -jar GenomeAnalysisTK.jar -T CombineVariants \
      -R ref/human_g1k_b37_20.fasta \
      -V:RNA sandbox/NA12878_rnaseq_20_HC_calls.vcf \
      -V:DNA sandbox/NA12878_wgs_20_HC_calls.vcf \
      -o sandbox/combined_calls.vcf \
      --genotypemergeoption UNIQUIFY \
      -L 20:15,800,000-16,100,000
```

The output VCF has records that look like this:

```
20      15884931        .       T       C       21.77   .
AC=2;AF=1.00;AN=2;DP=2;ExcessHet=3.0103;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=60.00;QD=10.88;SOR=0.693;set=RNA
GT:AD:DP:GQ:PL  ./.     1/1:0,2:2:6:49,6,0

20      15885525        .       CAAAAAAAAAAA    C       293.73  .
AC=1;AF=0.500;AN=2;BaseQRankSum=-1.733;ClippingRankSum=1.733;DP=13;ExcessHet=3.0103;FS=0.000;MLEAC=1;ML
EAF=0.500;MQ=60.00;MQRankSum=0.898;QD=29.37;ReadPosRankSum=-0.480;SOR=0.473;set=DNA     GT:AD:DP:GQ:PL
0/1:2,8:10:59:331,0,59  ./.

20      15888362        .       G       A       55.28   .
AC=4;AF=1.00;AN=4;DP=36;ExcessHet=3.0103;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=60.00;set=Intersection
GT:AD:DP:GQ:PL  1/1:0,33:33:99:1074,99,0        1/1:0,3:3:9:83,9,0
```

Run **SelectVariants** to subset a VCF directly based on discordance relative to a callset:

```
java -jar GenomeAnalysisTK.jar -T SelectVariants \
      -R ref/human_g1k_b37_20.fasta \
      -V sandbox/NA12878_rnaseq_20_HC_calls.vcf \
      --discordance sandbox/NA12878_wgs_20_HC_calls.vcf \
      -o sandbox/inRNA_notInDNA.vcf \
      -L 20:15,800,000-16,100,000
```

Various other tools and options exist to evaluate callset overlap, concordance at the site level and at the genotype level, summarize counts and statistics.

The **VariantsToTable** tool is also very useful to extract key statistics for further manipulations that are difficult to do directly from a VCF.