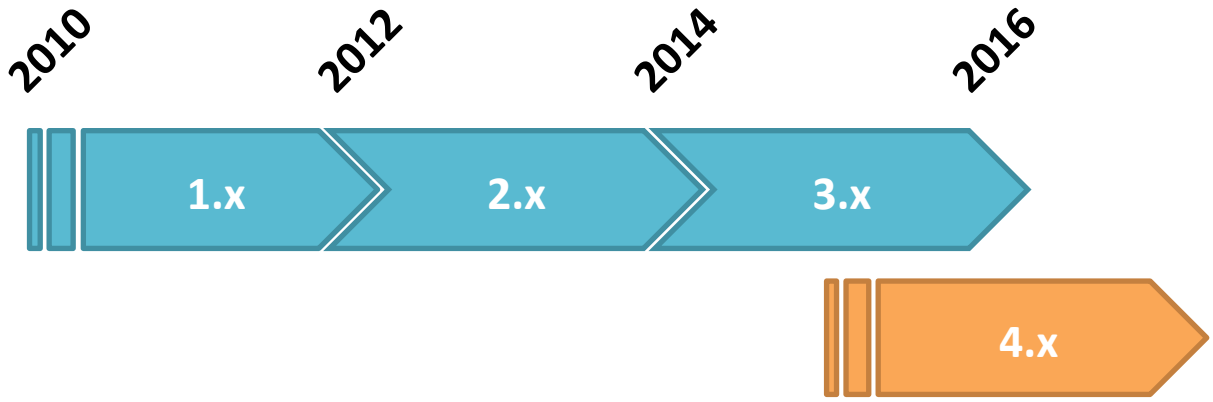
Somatic Copy Number Variation

Coming soon in GATK4 alpha:
New implementation of ReCapSeg

GATK development roadmap



Alpha GATK 4 : cloud-friendly and more scalable (Apache Spark)
+ extended functionality (CNVs, Picard)

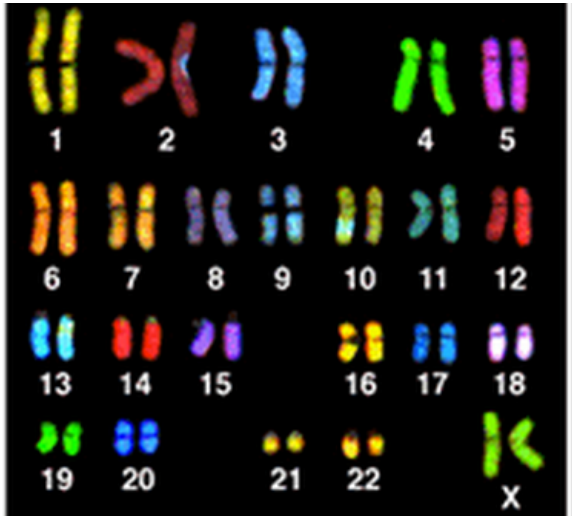
https://github.com/broadinstitute/gatk

First new feature in GATK 4 is a reimplementation of ReCapSeg

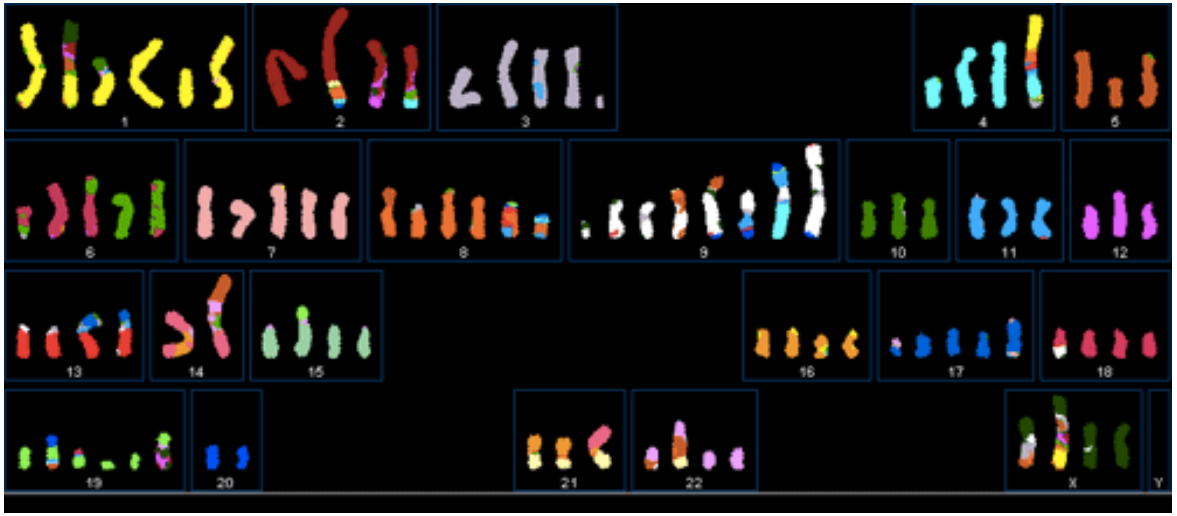
Copy Number Variations (=Alterations) can be dramatic

per genome (3 x 10^9 bp)	per Mbp	somatic alteration type	acronym
1000s to 10,000s	0.33 to >1000	single nucleotide variations	SNV or SNP
100s to 1,000s	< 1	small insertions & deletions	Indel
100s to 1,000s	< 1	structural variations	SV
100s to 1,000s	< 1	copy number alterations	CNA or CNV

Normal Cell



Cancer Cell Line HCC1954



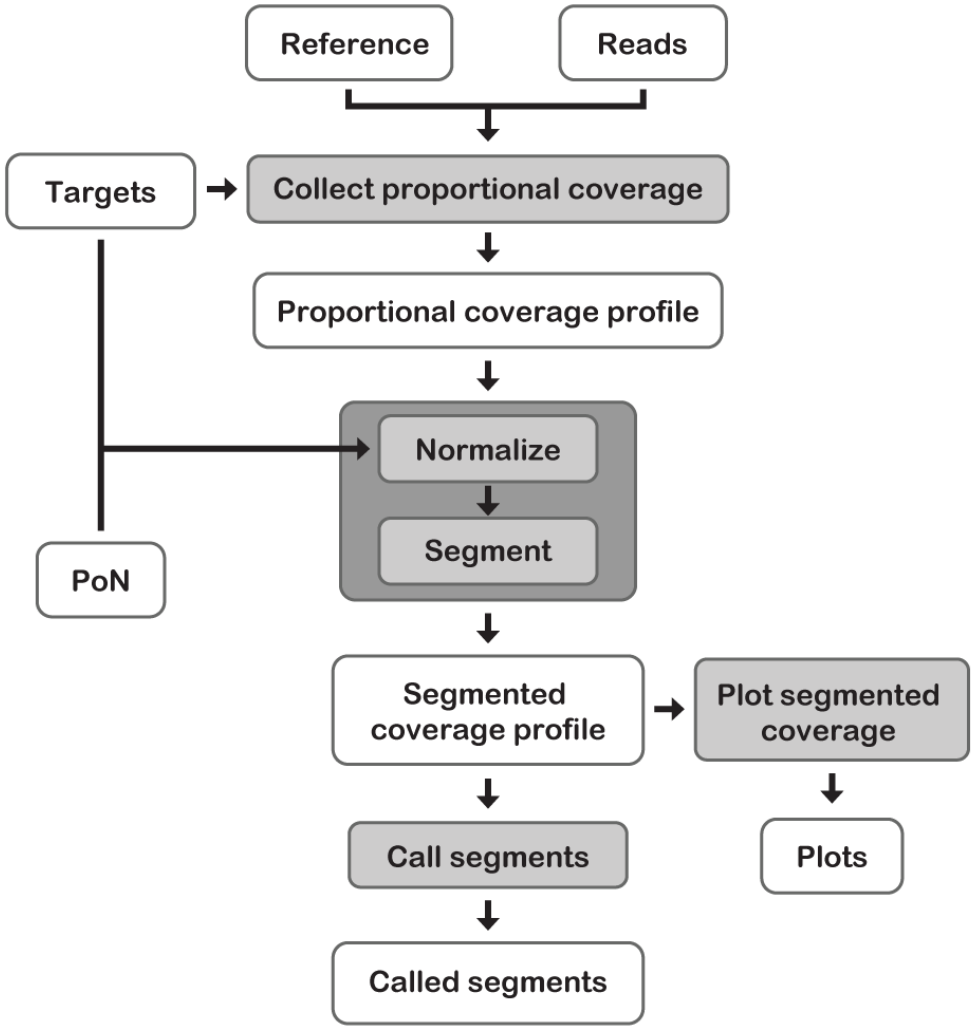
- Spectral karyotyping paints each chromosome pair with a color
- Alterations can vary dramatically between cancers and within cancers

Overview of the somatic CNV discovery workflow

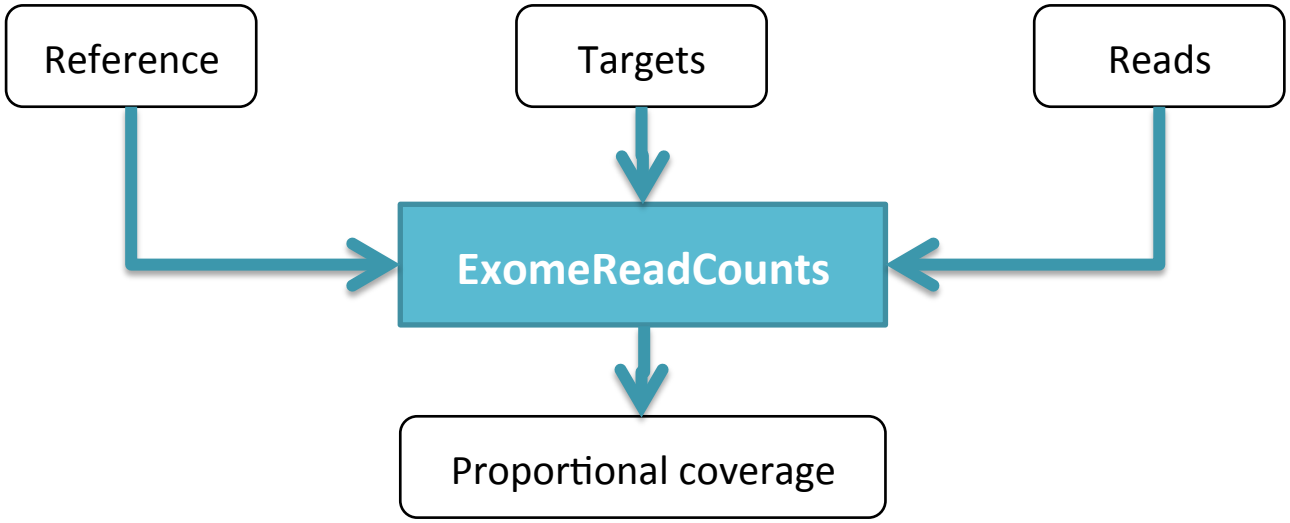
Underlying method (CapSeg) has been around for years and has been cited in many papers

Start:
- Genome reference
- Reads (BAM file)
- Targets (exome or gene panel)
- Panel of normals (PoN)

End: Amplification/deletion calls and copy ratio for each segment



Step 1: Collect proportional coverage




```
java -jar GATK4.jar ExomeReadCounts \
    -R <ref_genome> \
    -I <input_bam_file> \
    -O <pcov_output_file_path> \
    -exome <target_BED> \
    -transform PCOV -exonInfo FULL \
    -groupBy SAMPLE –keepdups
```

Collecting proportional coverage is just a matter of counting reads

First collects coverage per read group at each target:

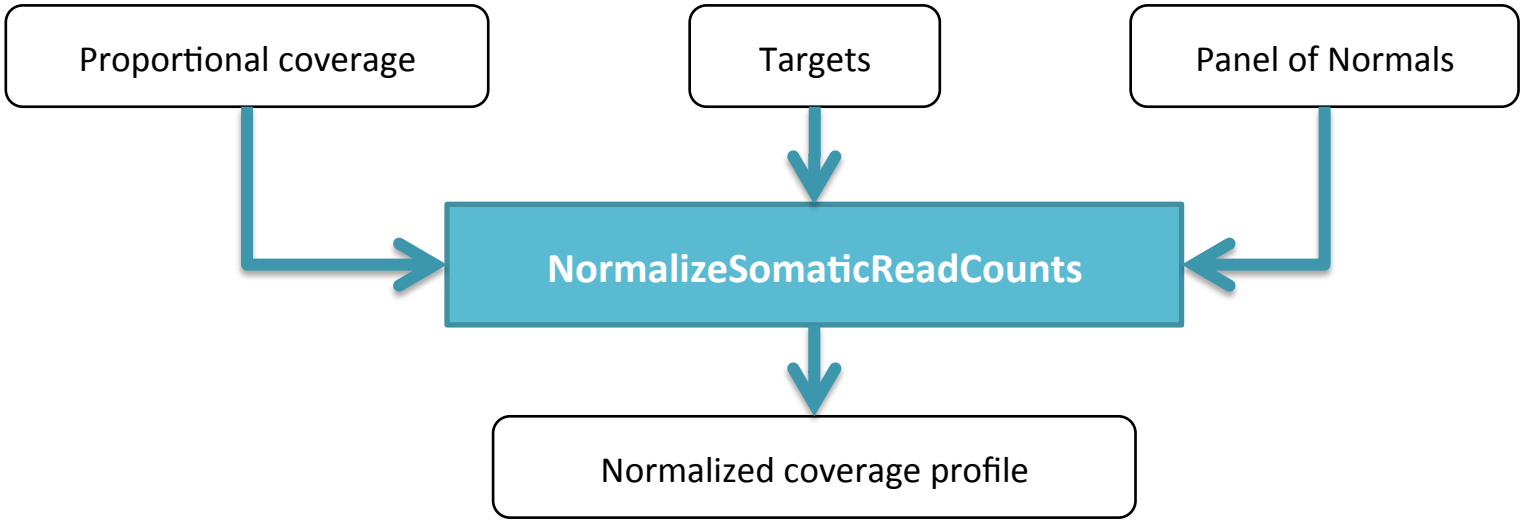
Dark region is target. Purple reads are counted, green are not.

Then divides coverage by total number of reads per sample.

Proportional coverage file

```
##fileFormat  = tsv
##commandLine =
org.broadinstitute.gatk.tools.exome.ExomeReadCounts  ...snip...
##title       = Read counts per target and sample
CONTIG  START   END     NAME    SAMPLE1
1       12200   12275   target1    1.150e-05
1       13505   13600   target2    1.500e-05
1       31000   31500   target3    7.000e-05
....snip....
```

Step 2: Create normalized coverage profile



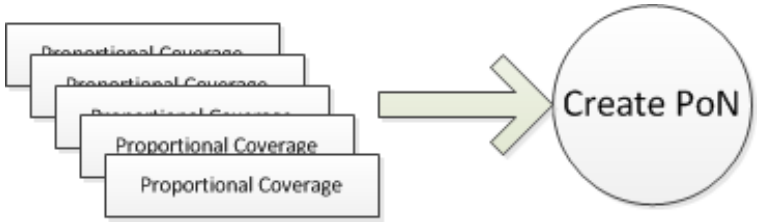

```
java -jar GATK4.jar NormalizeSomaticReadCounts \
     -I <pcov_input_file_path> \
     -T <target_BED> \
     -pon <pon_file> \
     -O <output_target_cr_file> \
     -FNO <output_target_fnt_file> \
     -BHO <output_beta_hats_file> \
     -PTNO <output_pre_tangent_normalization_cr_file>
```

Also requires `-Djava.library.path=<hdf_jni_native_dir>` *(omitted for simplicity)*

The PoN is used for normalization

- The PoN is created by collecting proportional coverage from each Normal sample

ExomeReadCounts (on each Normal)
-> CombineReadCounts
-> CreatePanelOfNormals

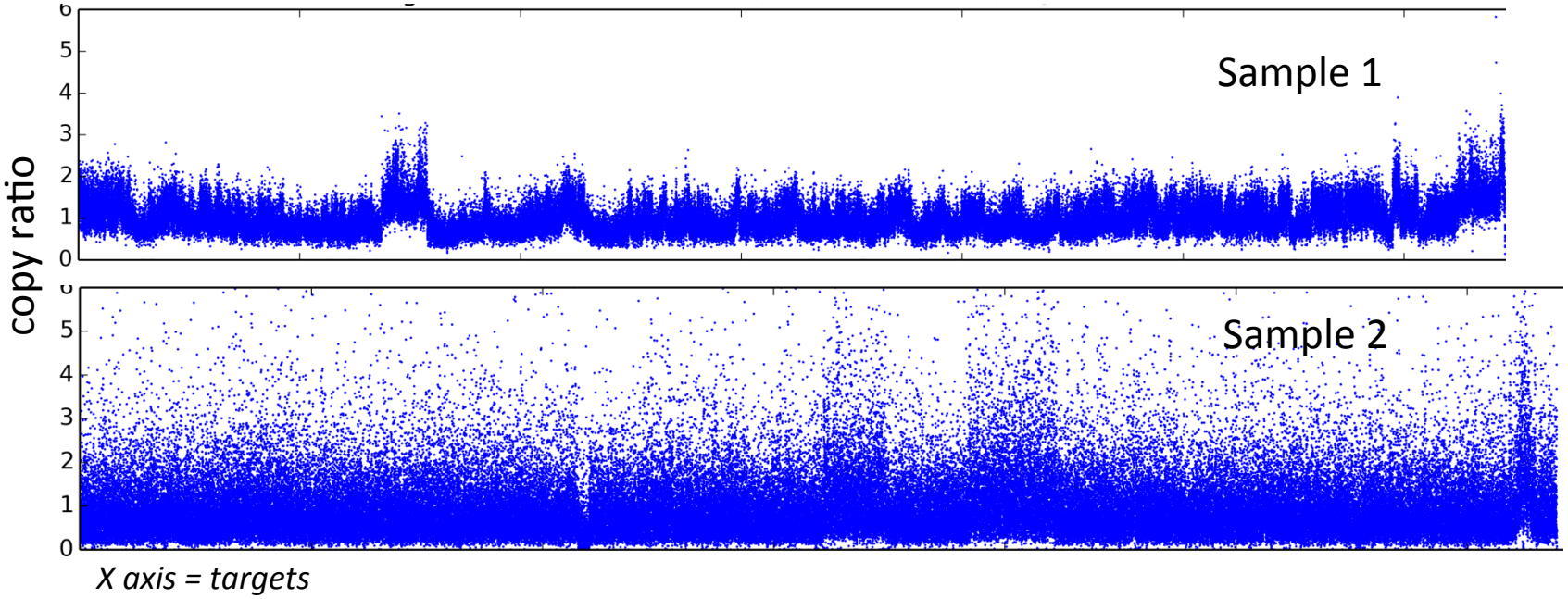


- PoN stores median proportional coverage per target across the panel (target factors)

- Case sample is normalized against the target factors

=> First estimate at copy ratio per target in a case sample

Normalized coverage profiles are still very noisy



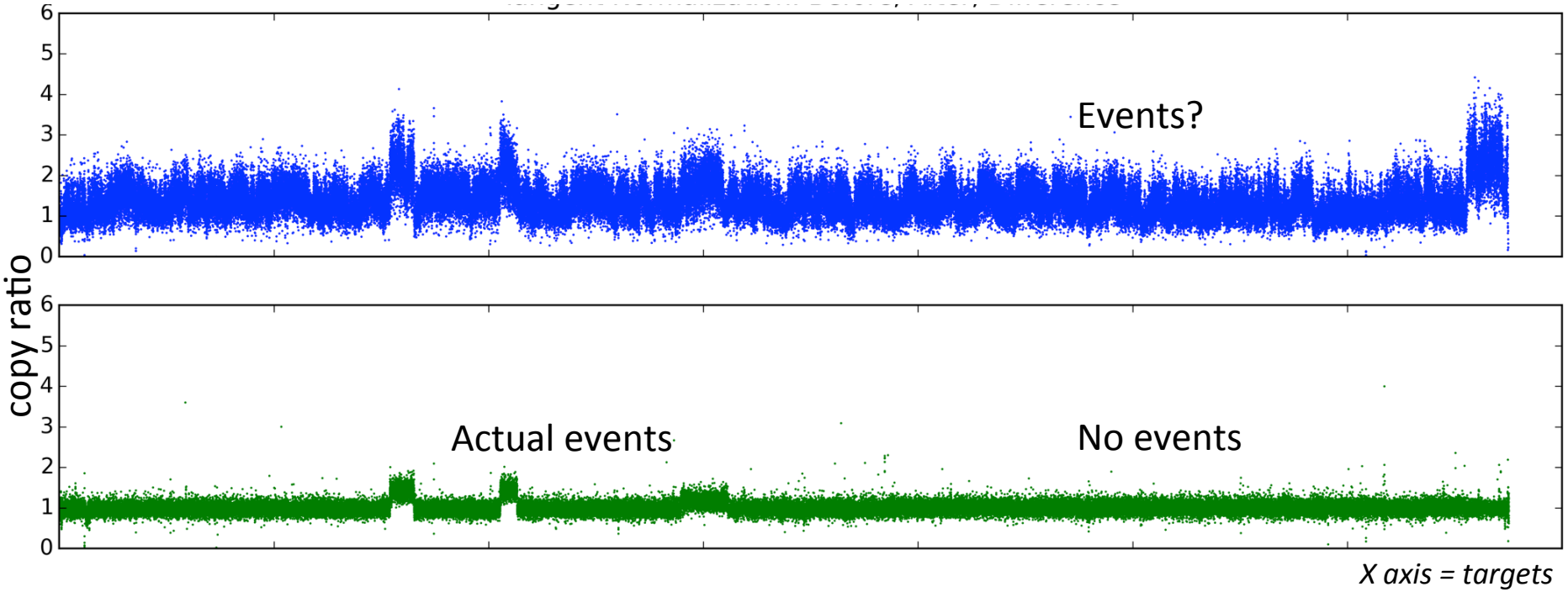
- Some substructure contributed by sequencing artifacts
  => would lead to hypersegmentation

- Lower coverage → increased noise

Important to choose appropriate Normal samples for PoN

- Use separate PoNs for different sequencing platforms
  - Protocol change in sample prep
  - Target region change
- Identify the samples
  - Blood normals without large events (see PoN QC)
  - Young subjects without bloodborne cancers
  - How many?
    - Recommend 40 or more
    - Depends on noisiness of data – more never hurts
- QC steps included to eliminate samples with large CNVs

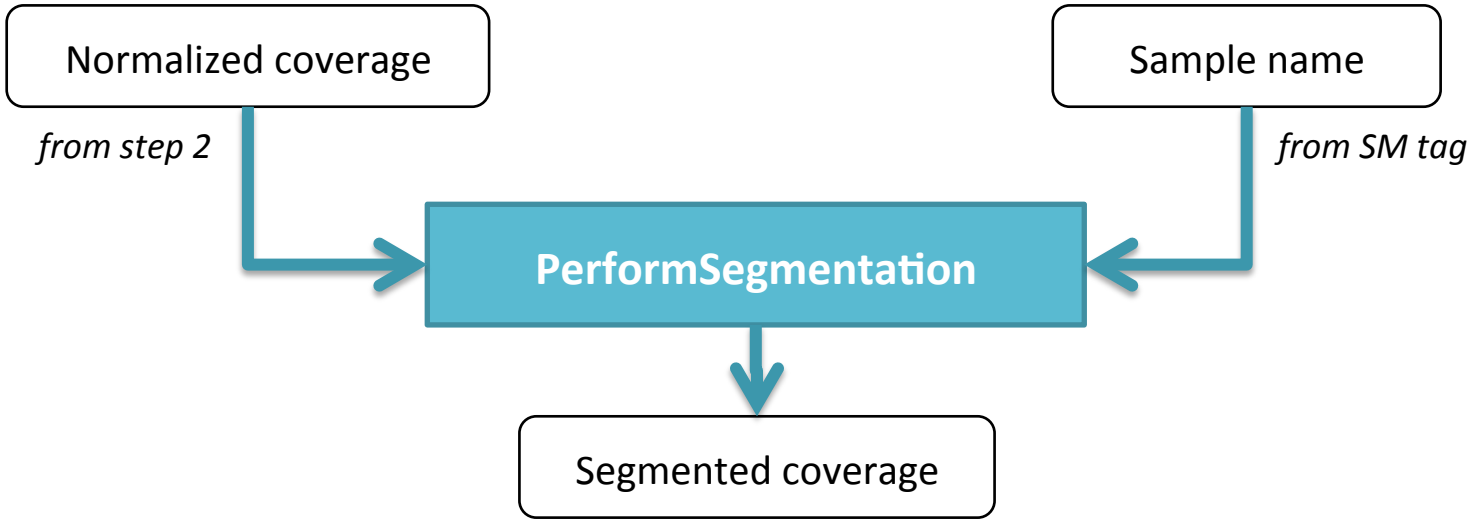
Second round of (tangent) normalization cleans them up



Normalized coverage profile

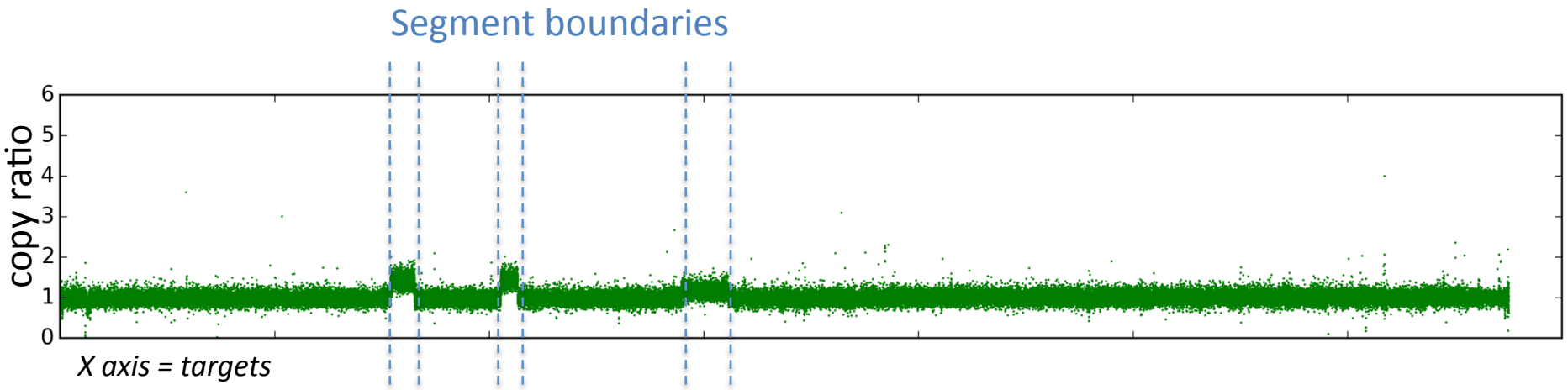
```
#fileFormat = tsv
#commandLine = ....snip....
#title = ....snip....
name    contig  start   stop    SAMPLE1
target1    1       12200   12275   -0.5958351605220968
target2    1       13505   13600   -0.2855054918109098
target3    1       31000   31500   -0.11450116047248263
....snip....
```

Step 3: Segment coverage profile




```
java -jar GATK4.jar PerformSegmentation \
    -S <sample_name> \
    -T <normalized_coverage_file> \
    -O <output_seg_file> \
    -log
```

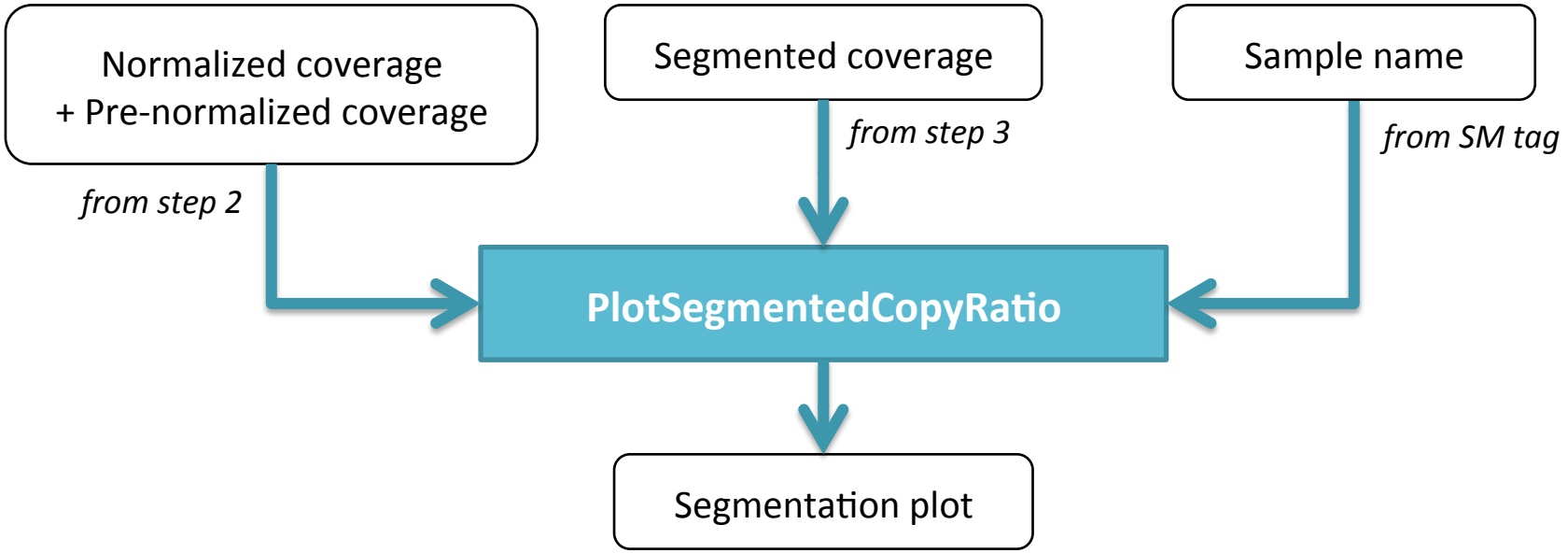
Segmentation produces… segments!



Segmented coverage file

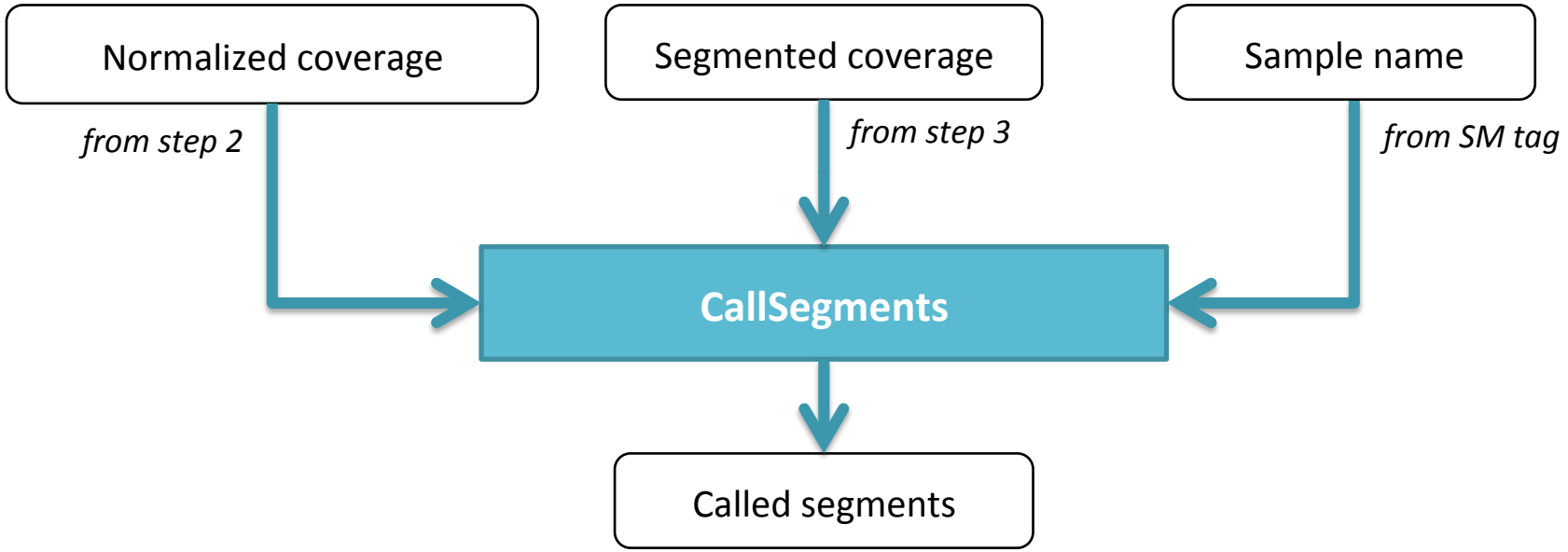
```
Sample  Chromosome      Start   End     Num_Probes      Segment_Mean
SAMPLE1         1       12200   70000   18       0.841235
SAMPLE1         1       300600  1630000 337     1.23232323
....snip....
```

Step 4: Plot segmented coverage profile



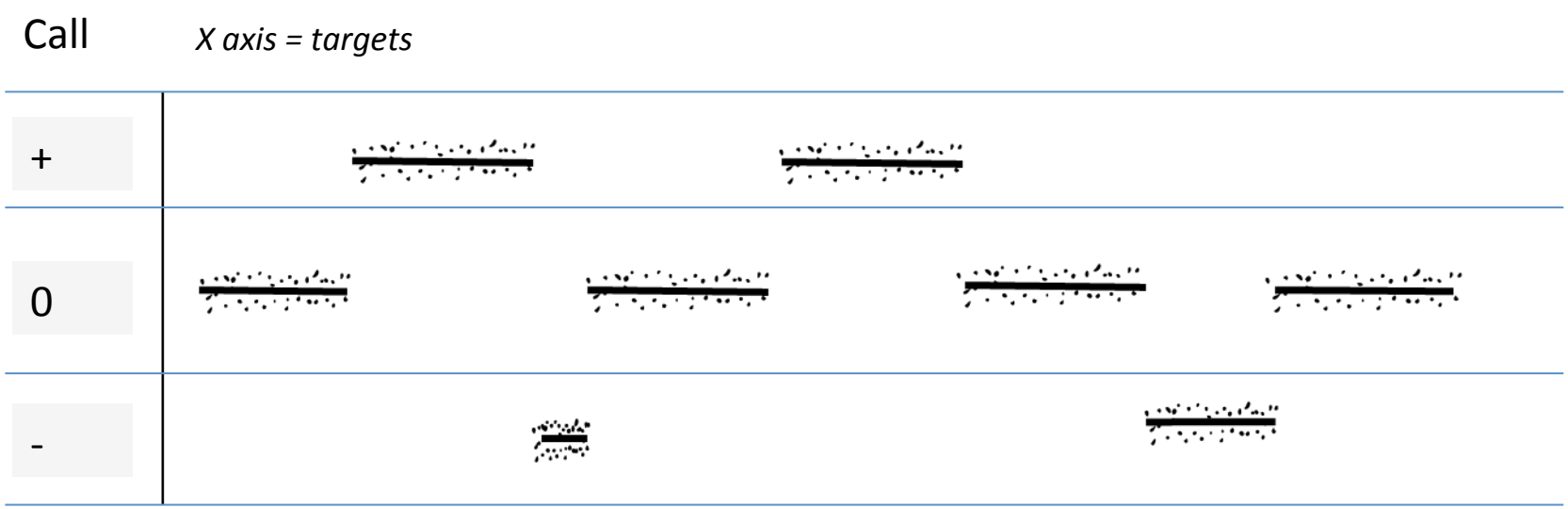

```
java -jar GATK4.jar PlotSegmentedCopyRatio \
    -S <sample_name> \
    -T <normalized_coverage_file> \
    -P <pre_normalized_coverage_file> \
    -seg <segmented_coverage_file> \
    -O <output_seg_plot> \
    -log
```

Step 5: Call segments




```
java -jar GATK4.jar CallSegments \
    -T <normalized_coverage_file> \
    -S <seg_file> \
    -O <output_called_seg_file> \
    -sample <sample_name>
```

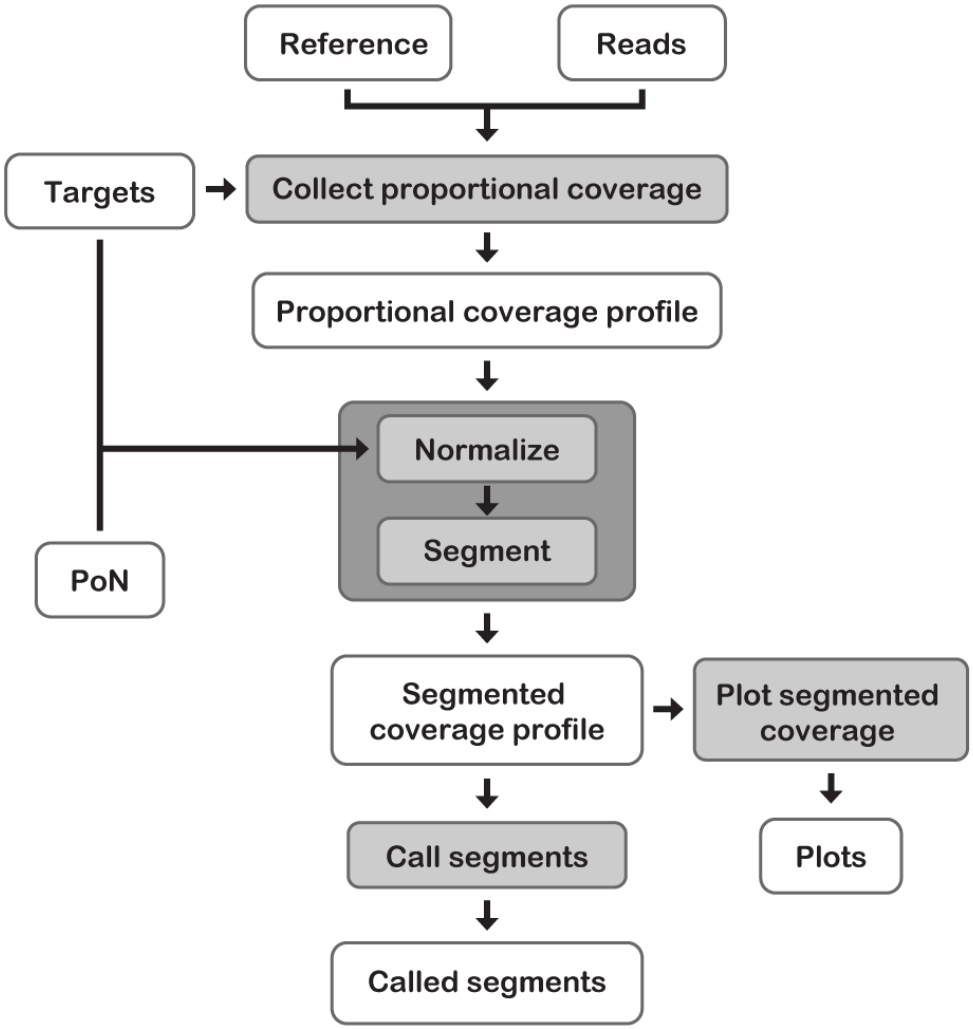
Each segment is given a call (3 possibilities)



Called segments

Sample	Chromosome	Start	End	Num_Probes	Segment_Mean	Segment_Call
SAMPLE1	1	12200	70000	18	0.841235	-
SAMPLE1	1	300600	1630000	337	1.23232323	0
....snip....						

Recap of the CNV calling workflow



Further reading

http://gatkforums.broadinstitute.org/categories/recapseg-documentation

https://github.com/broadinstitute/hellbender-protected#the-cnv-case-and-pon-workflows-description-and-examples