Alignment, variant calling, and filtering

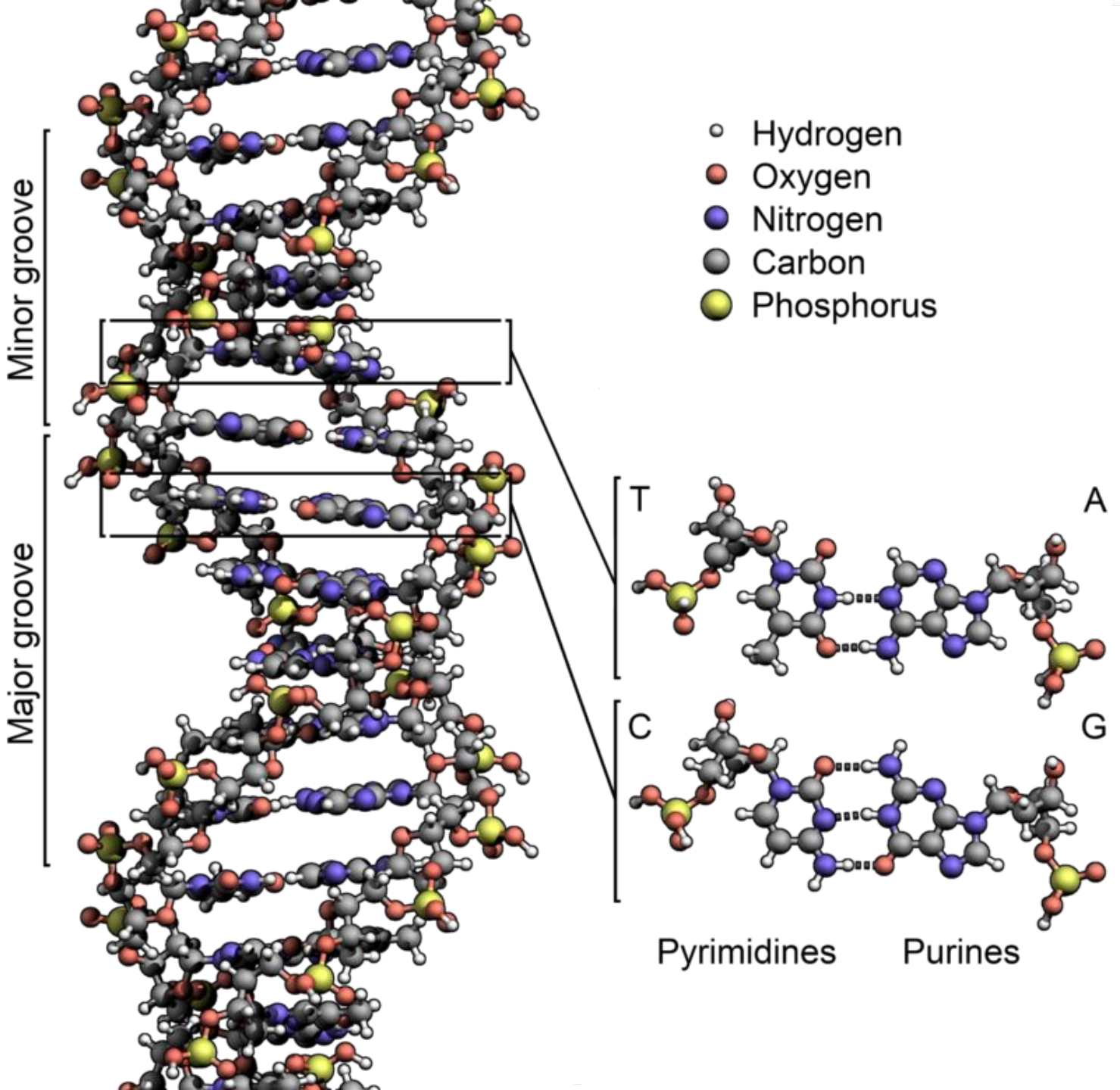
Erik Garrison

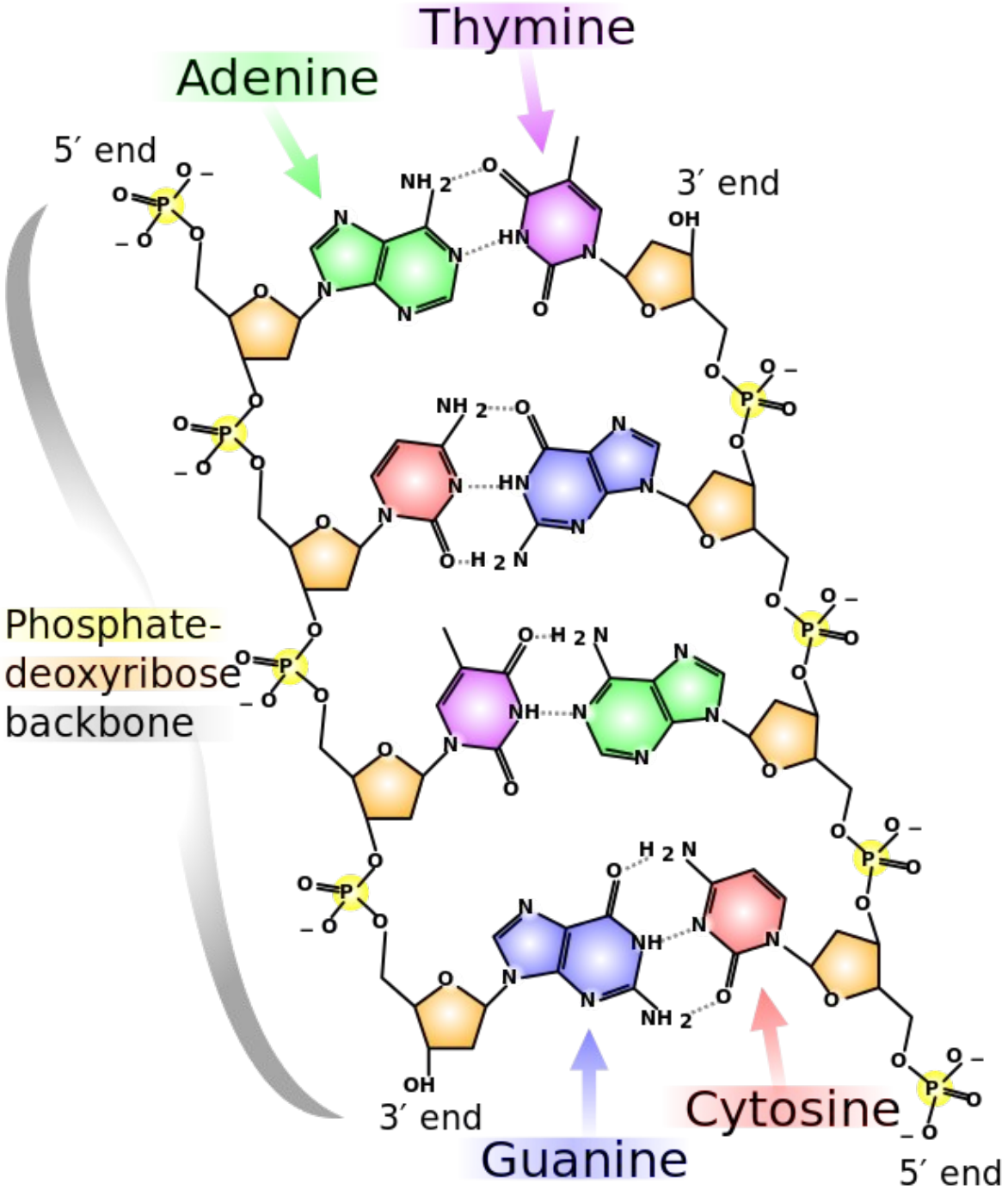
B@G Winter School
March 1, 2016

Overview

1. **Genesis of variation (SNPs and indels)**
2. Causes of sequencing error
3. Sequence alignment
4. Alignment-based variant detection
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

DNA


Hydrogen
Oxygen
Nitrogen
Carbon
Phosphorus
Minor groove
Major groove
T
A
C
G
Pyrimidines
Purines



DNA

“twisted” view of ladder

A SNP

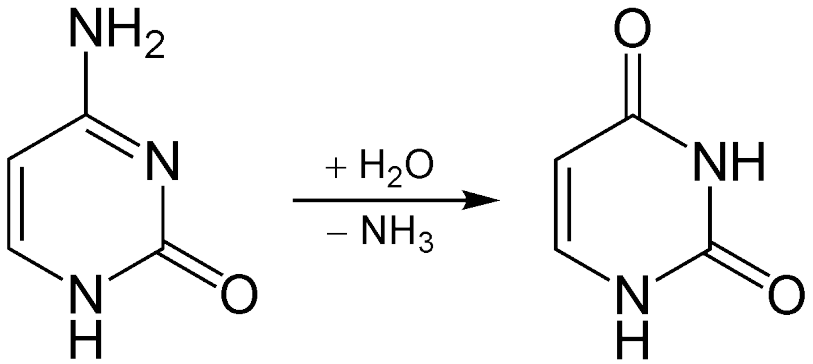
A point mutation in which one base is swapped for another.

AATTAG*C*CATTA

AATTAG*T*CATTA

(some) causes of SNPs

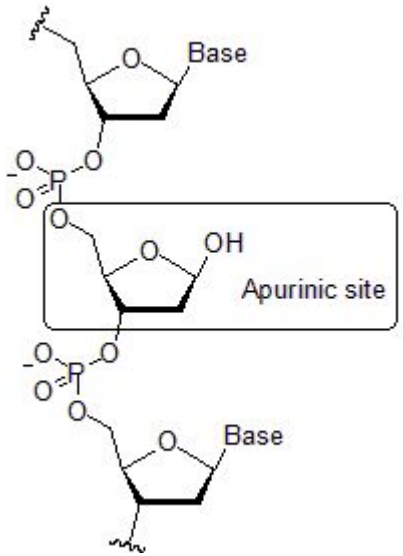
- Deamination
  - cytosine → uracil
  - 5-methylcytosine → thymine
  - guanine → xanthine (mispairs to A-T bp)
  - adenine → hypoxanthine (mispairs to G-C bp)

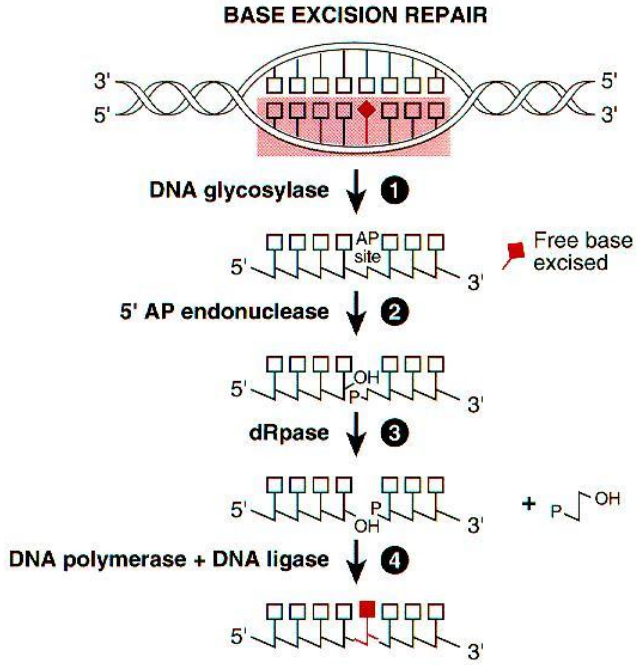


Deamination of Cytosine to Uracil
http://en.wikipedia.org/wiki/Deamination

(some) causes of SNPs

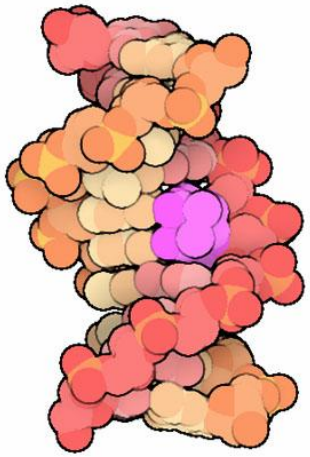
- Depurination
  - purines are cleaved from DNA sugar backbone (5000/cell/day, pyrimidines at much lower rate)
  - Base excision repair (BEP) can fail → mutation

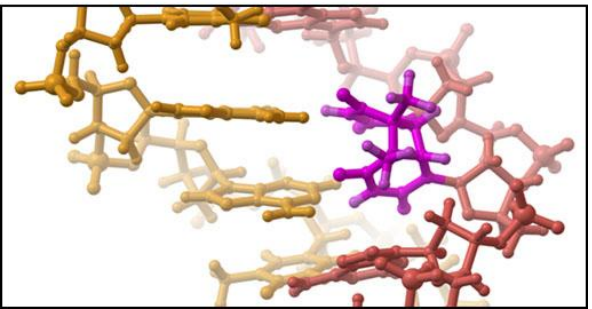


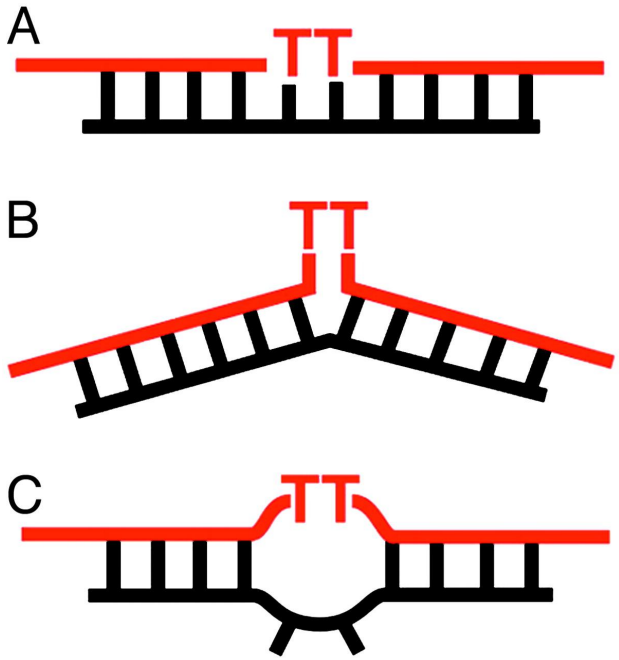


Multi-base events (MNPs)

- MNPs
  - thymine dimerization (UV induced)
  - other (e.g. oxidative stress induced)



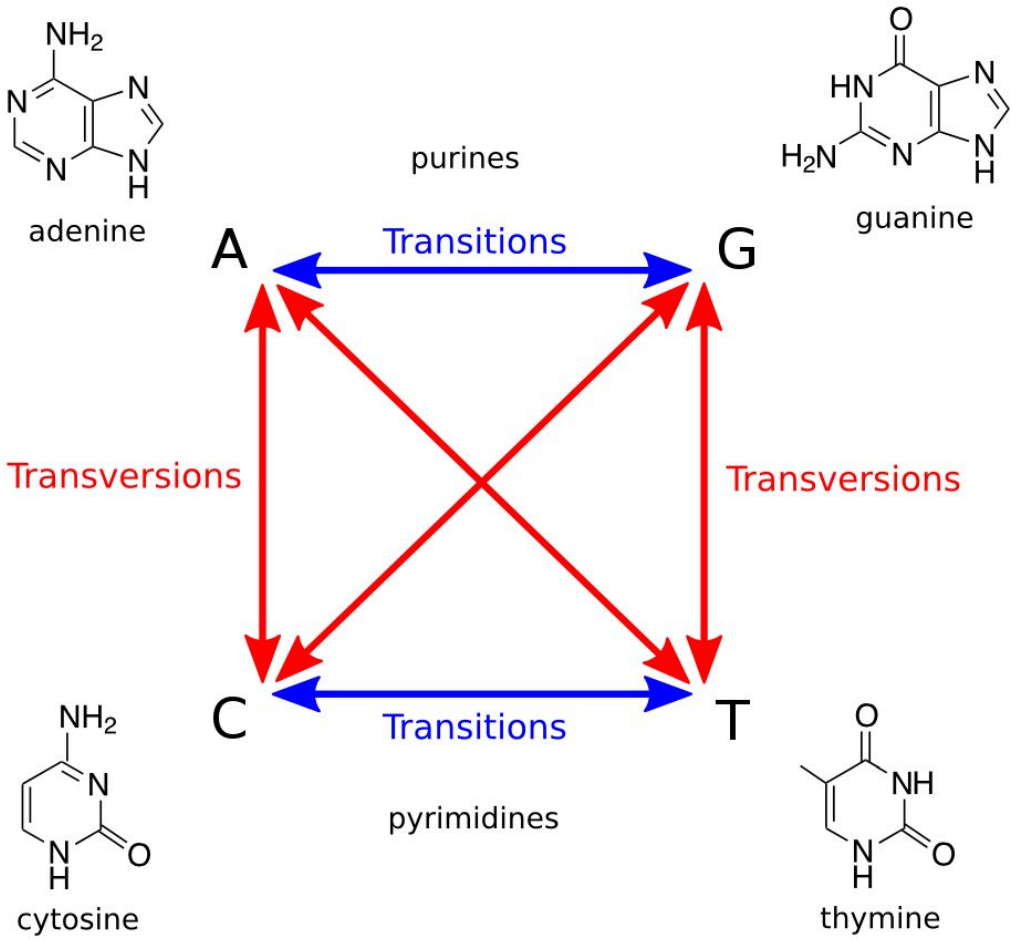




http://www.rcsb.org/pdb/101/motm.do?momID=91

Transitions and transversions

In general **transitions are 2-3 times more common than transversions**. (But this depends on biological context.)



https://upload.wikimedia.org/wikipedia/commons/3/35/Transitions-transversions-v3.png

An INDEL

A mutation that results from the gain or loss of sequence.

AATTAGCCATTA

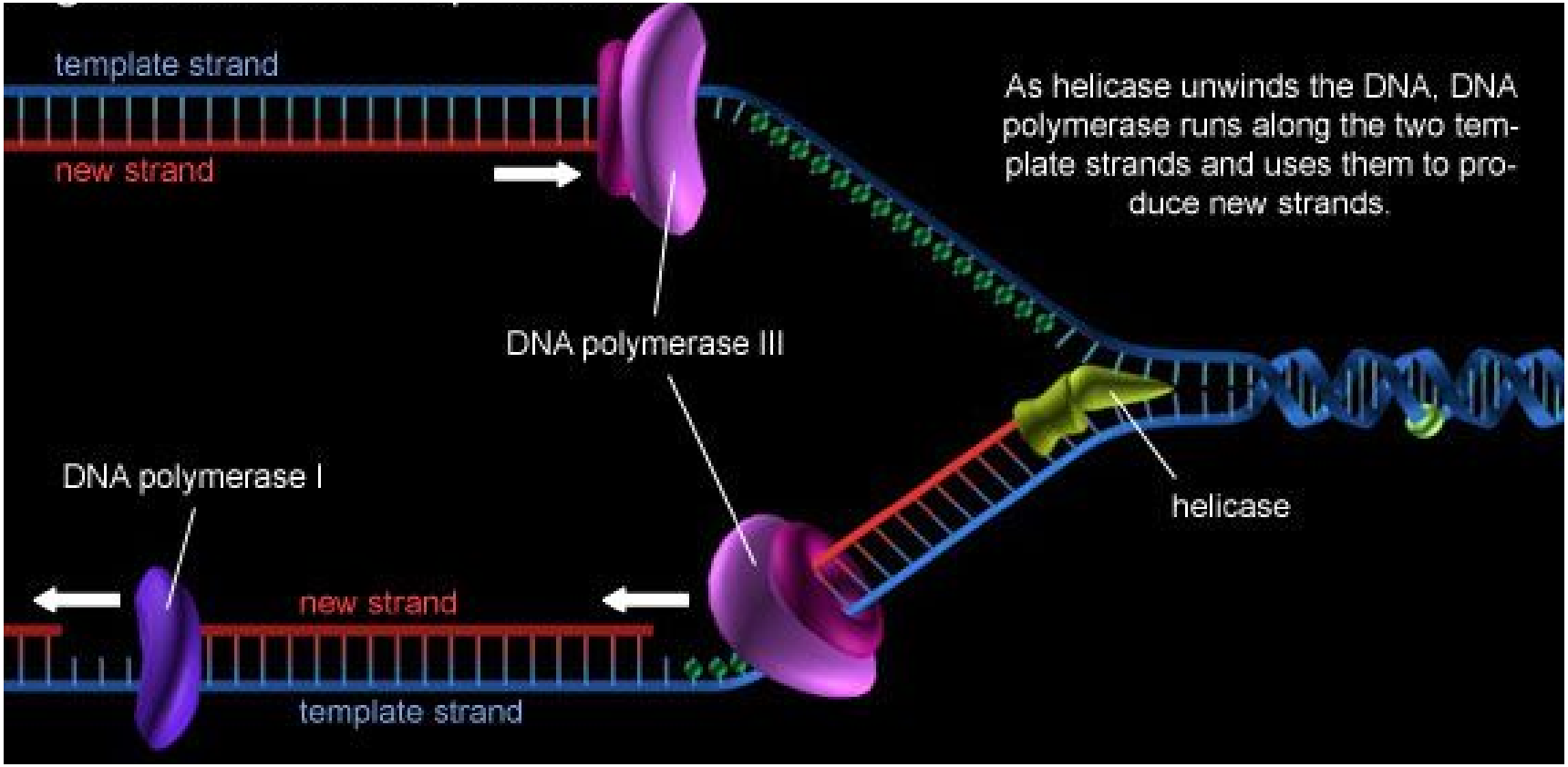
AATTA--CATTA

INDEL genesis

A number of processes are known to generate insertions and deletions in the process of DNA replication:

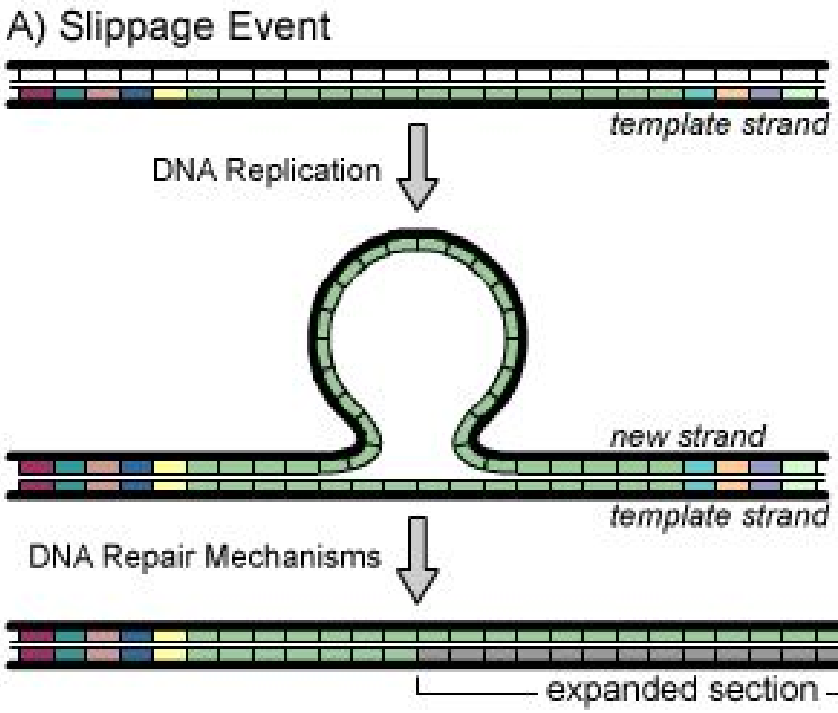
- Replication slippage
- Double-stranded break repair
- Structural variation (e.g. mobile element insertions, CNVs)

DNA replication



http://www.stanford.edu/group/hopes/cgi-bin/wordpress/2011/02/all-about-mutations/

Polymerase *slippage*

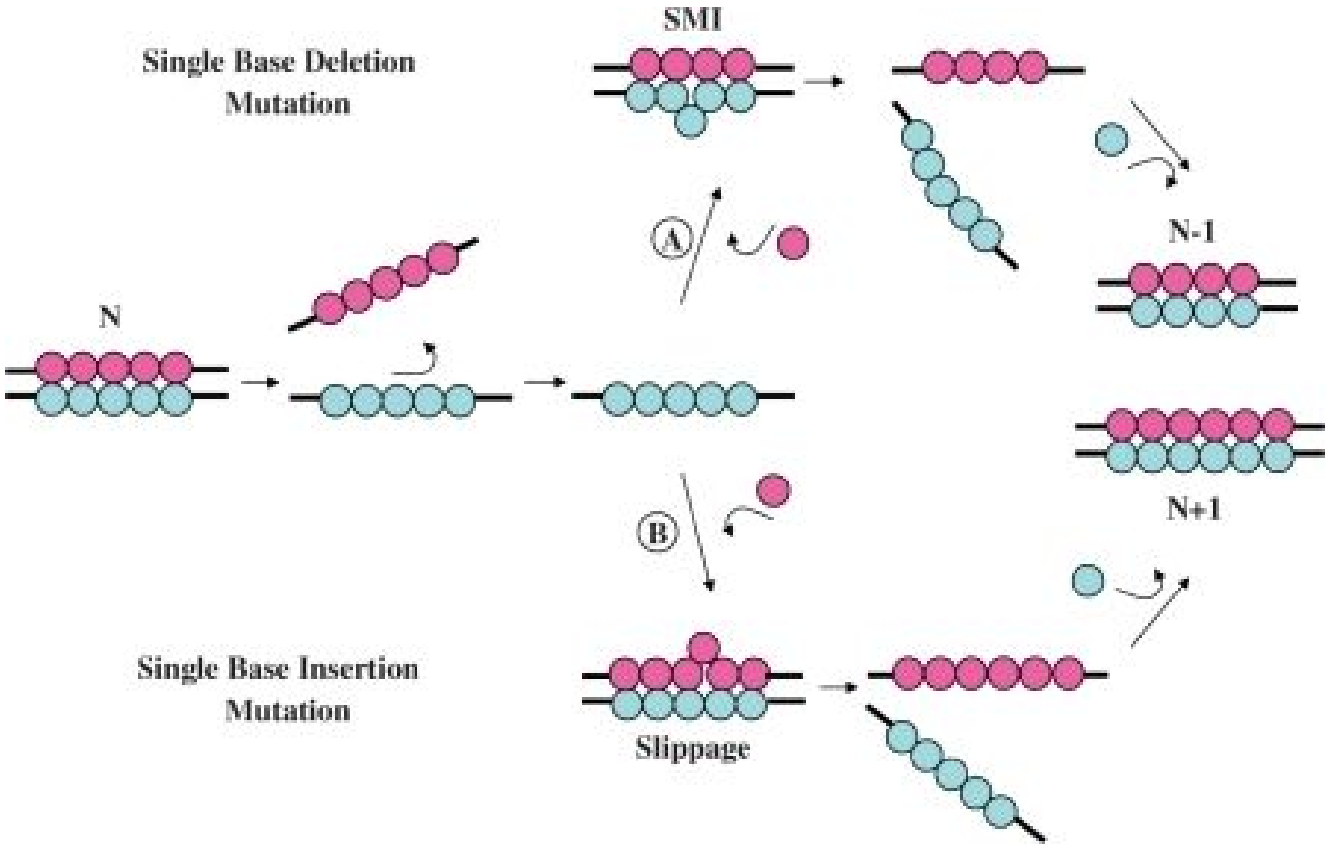


(A) During replication, polymerase slippage and subsequent reattachment may cause a bubble to form in the new strand. Slippage is thought to occur in sections of DNA with repeated patterns of bases (such as CAG), represented here by matching colors. Then, DNA repair mechanisms realign the template strand with the new strand and the bubble is straightened out. The resulting double helix is thus expanded.

(B) Polymerase slippage, as theorized, cannot occur in DNA without repeating patterns of bases.

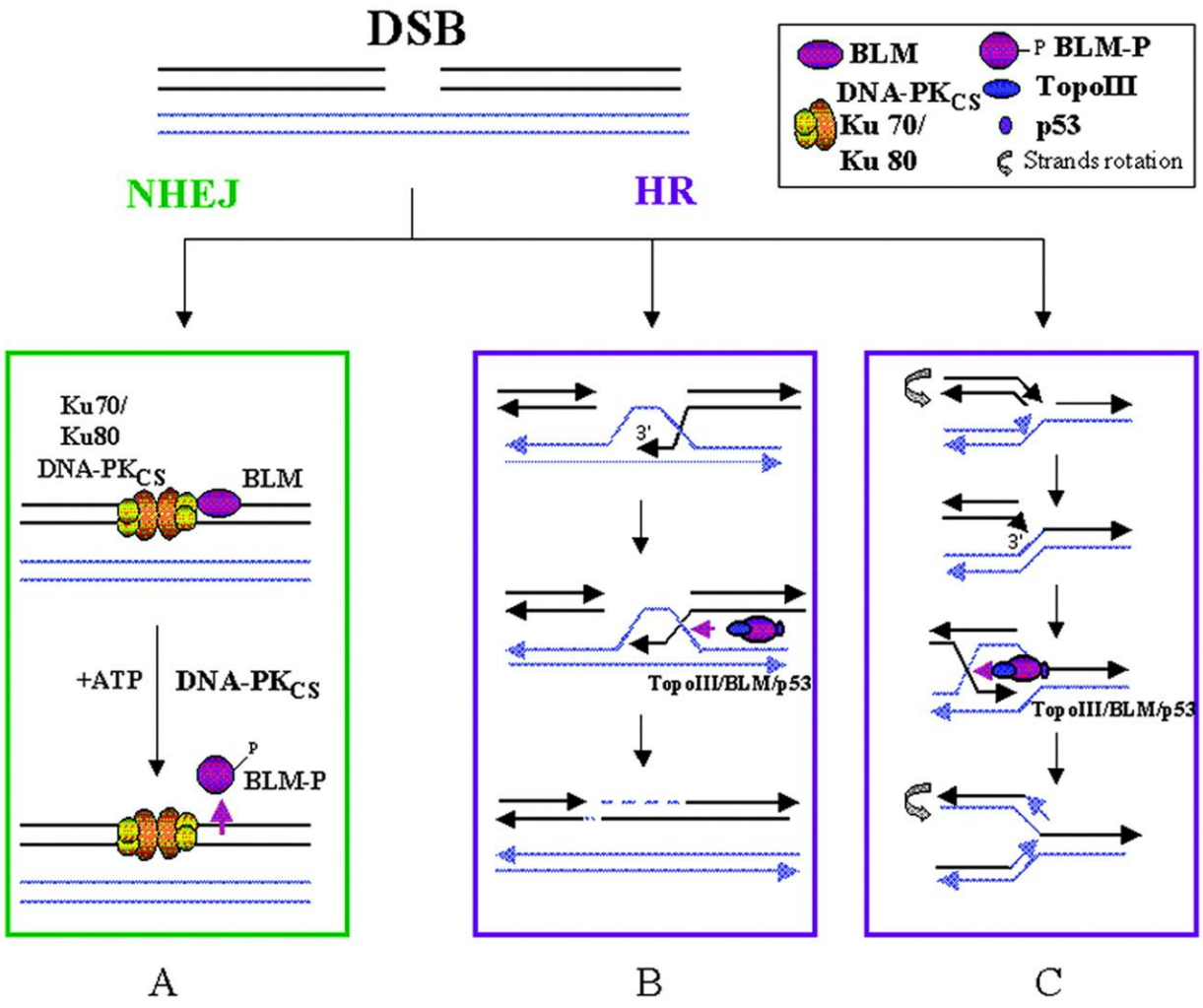
http://www.stanford.edu/group/hopes/cgi-bin/wordpress/2011/02/all-about-mutations/

Insertions and deletions via slippage



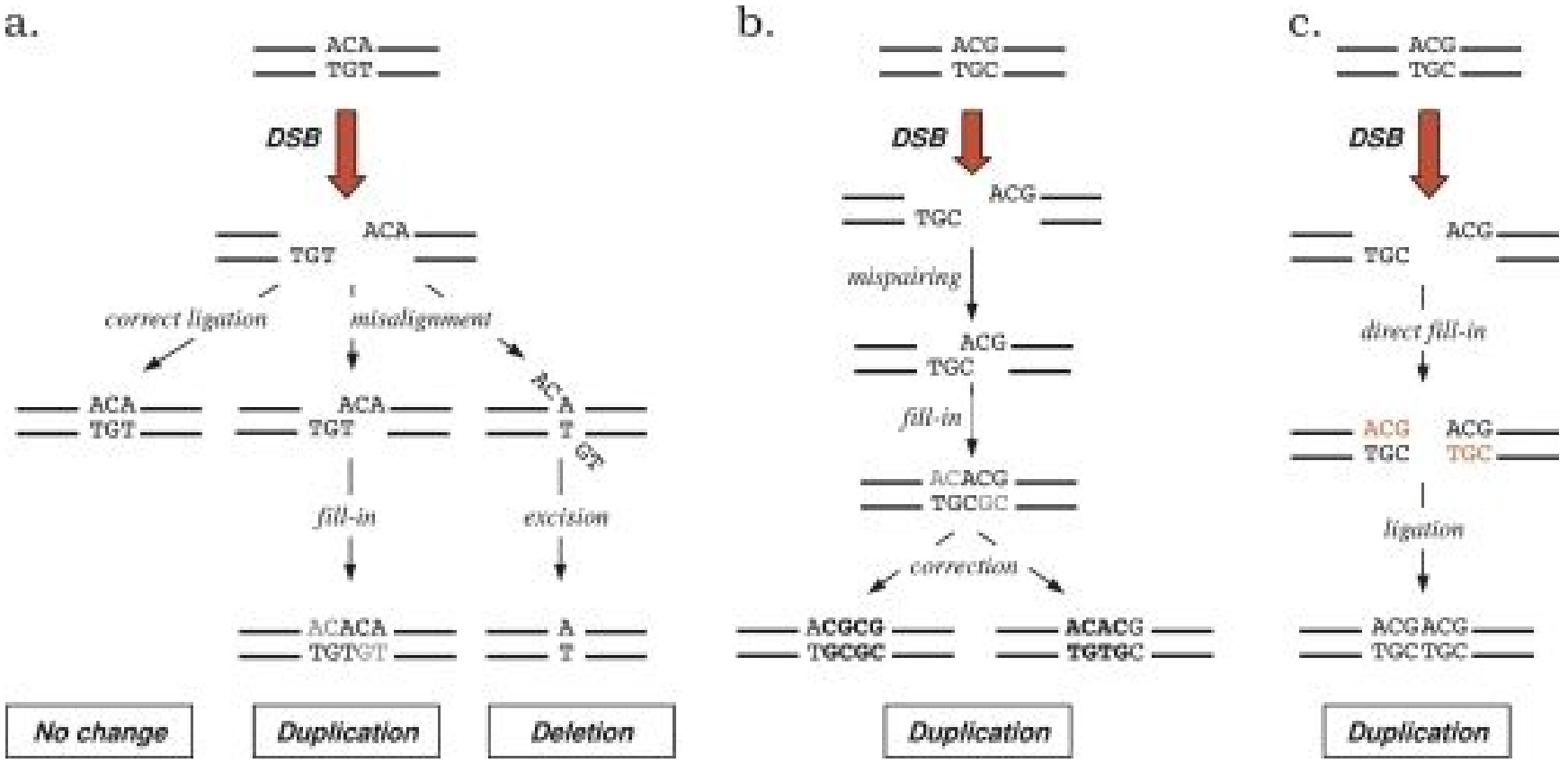
Energetic signatures of single base bulges: thermodynamic consequences and biological implications. Minetti CA, Remeta DP, Dickstein R, Breslauer KJ - Nucleic Acids Res. (2009)

Double-stranded break repair



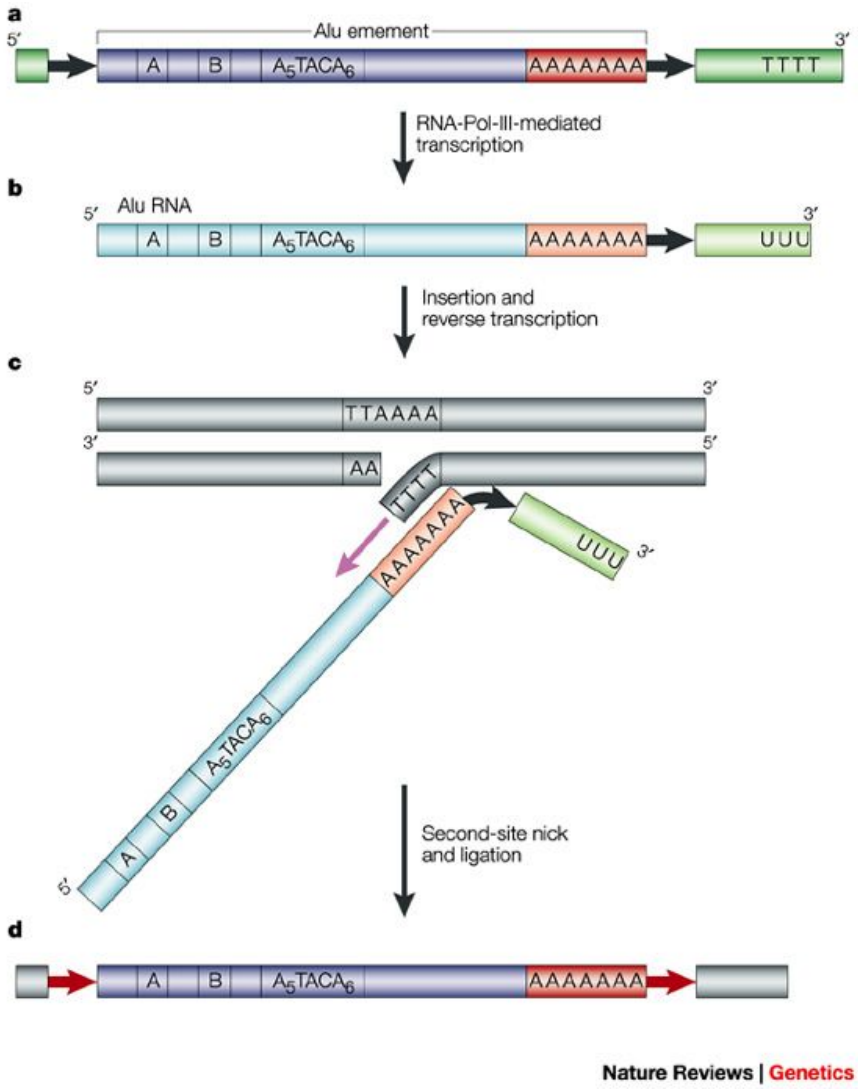
Possible anti-recombinogenic role of Bloom's syndrome helicase in double-strand break processing. doi: 10.1093/nar/gkg834

NHEJ-derived indels



DNA Slippage Occurs at Microsatellite Loci without Minimal Threshold Length in Humans: A Comparative Genomic Approach. Leclercq S, Rivals E, Jarne P - Genome Biol Evol (2010)

Structural variation (SV)



Nature Reviews | Genetics

Transposable elements (in this case, an Alu) are sequences that can copy and paste themselves into genomic DNA, causing insertions.

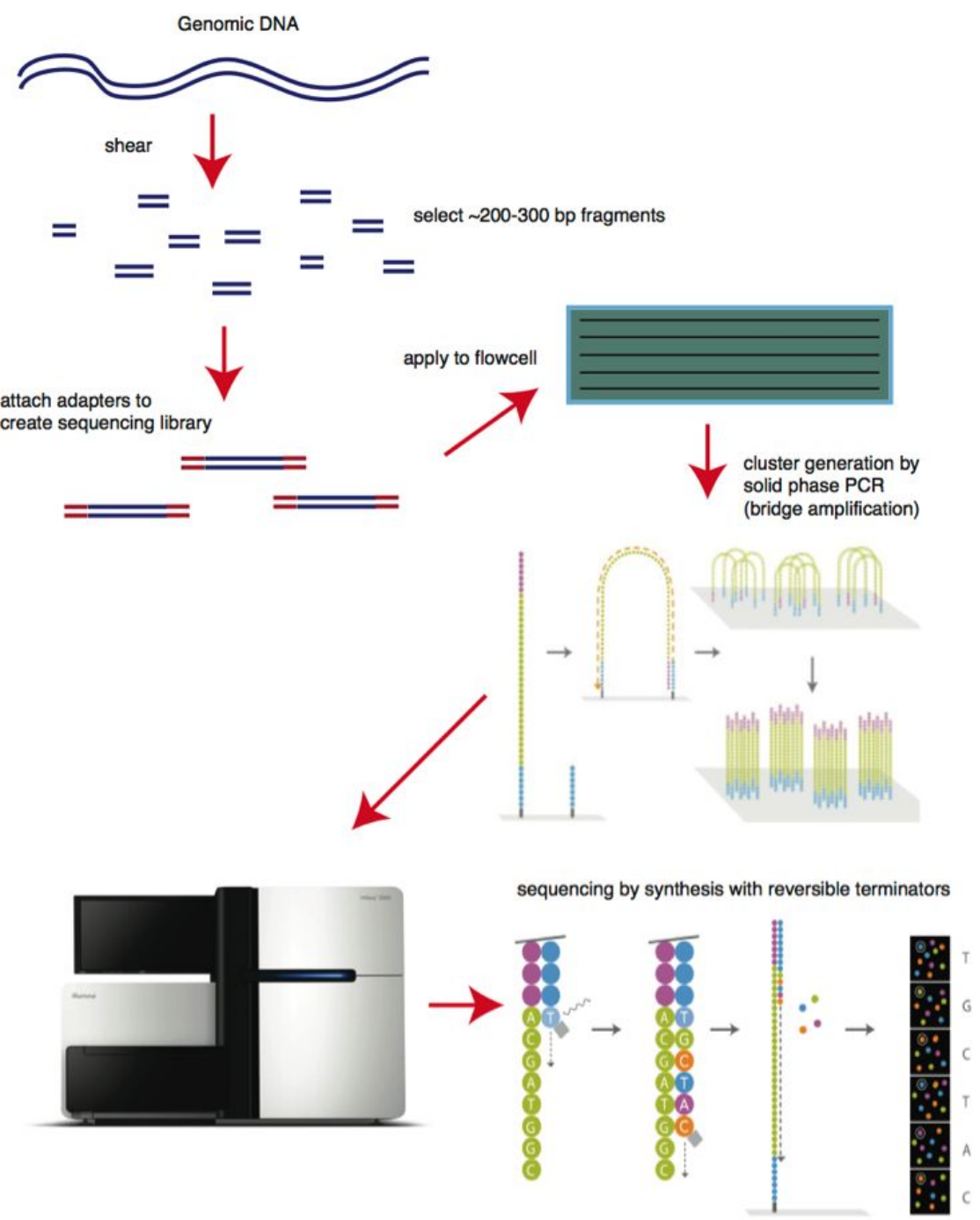
Deletions can also be mediated by these sequences via other processes.

http://www.nature.com/nrg/journal/v3/n5/full/nrg798.html

Overview

1. Genesis of variation (SNPs and indels)
2. **Causes of sequencing error**
3. Sequence alignment
4. Alignment-based variant detection
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

Sequencing by synthesis (Illumina)

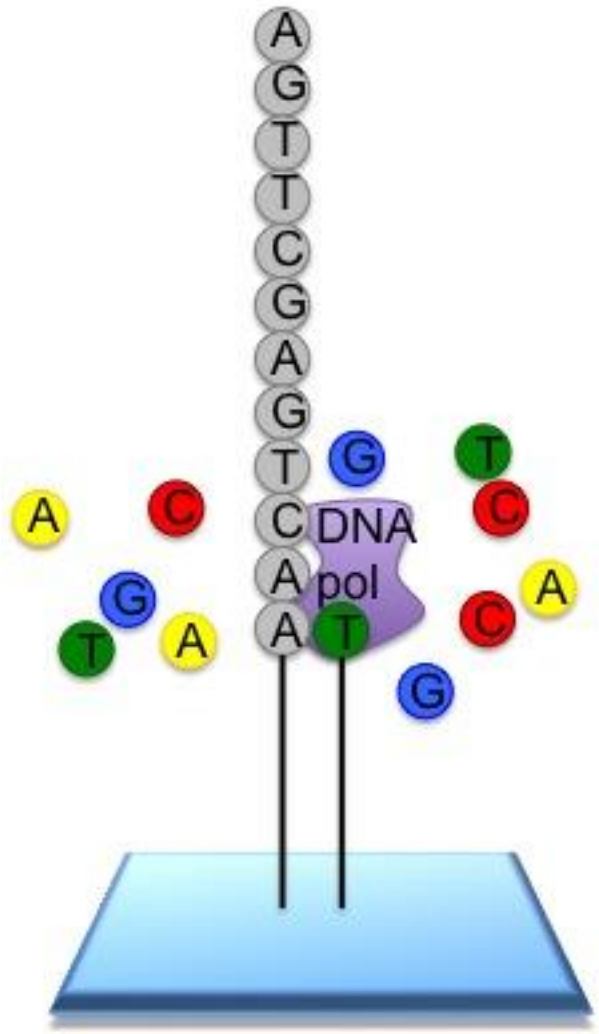


http://bitesizebio.com/13546/sequencing-by-synthesis-explaining-the-illumina-sequencing-technology/

Illumina sequencing process

(1)

Fluorescently labeled dNTPs with reversible terminators are incorporated by polymerase into growing double-stranded DNA attached to the flowcell surface.



http://www.ebi.ac.uk/training/online/course/ebi-next-generation-sequencing-practical-course/what-next-generation-dna-sequencing/illumina-

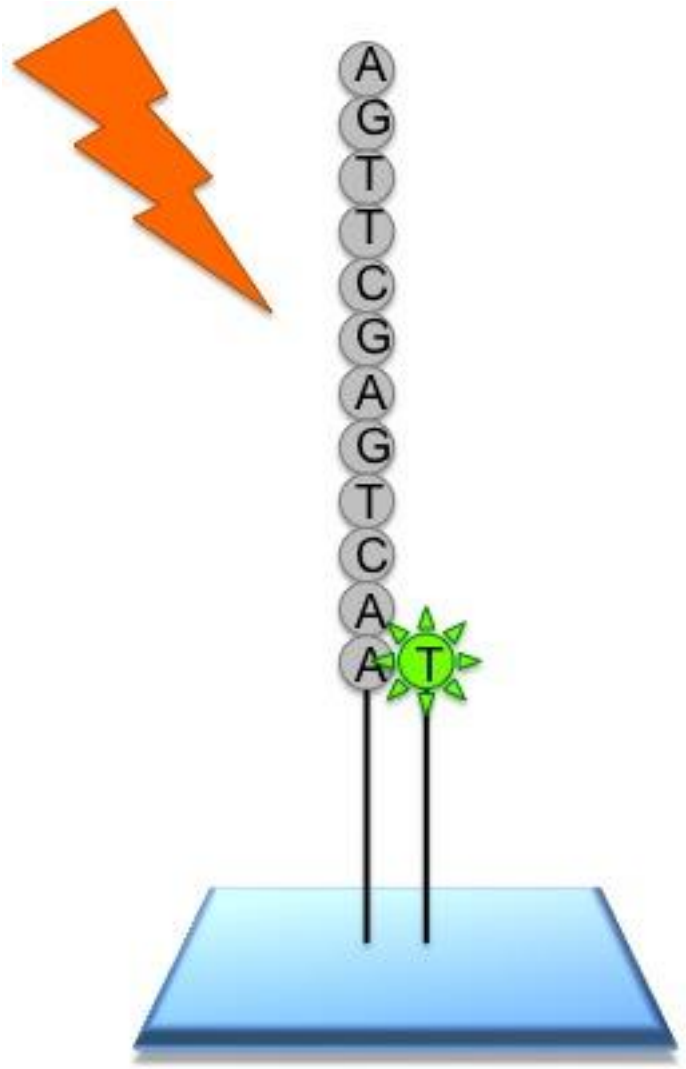
Illumina sequencing process

(2) A round of imaging determines the predominant base in each cluster at the given position.

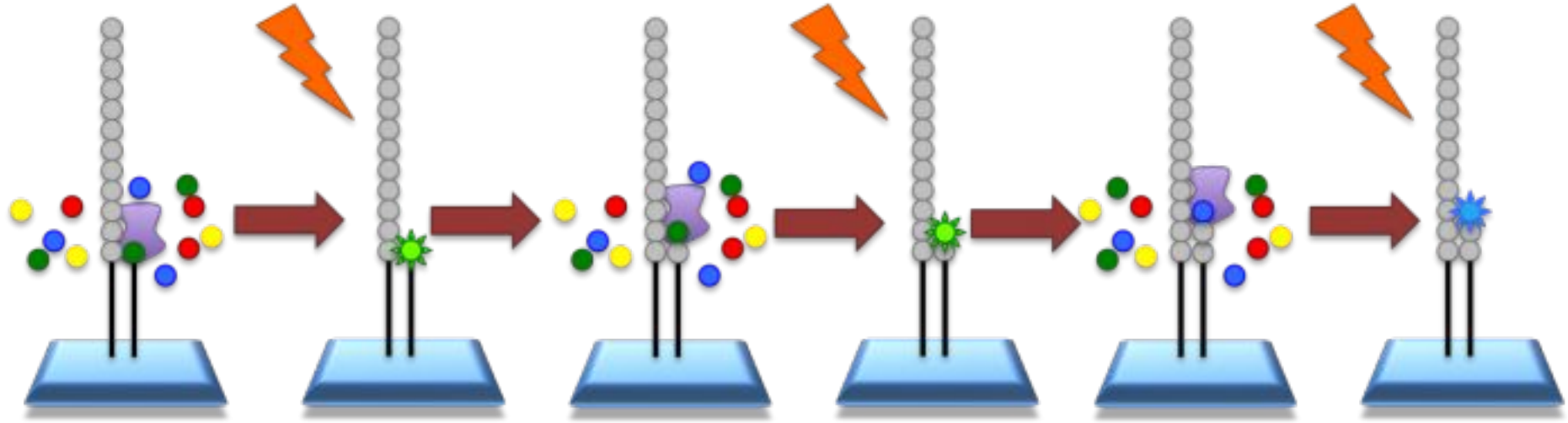
(3) “reverse” terminator by washing, goto **(1)**

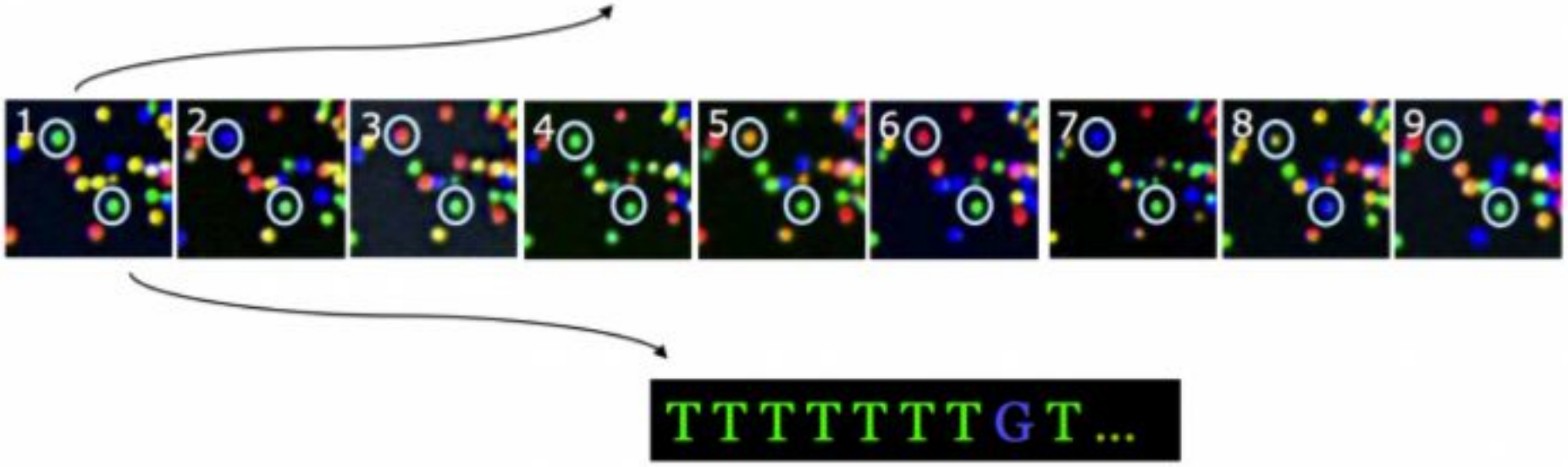
(...)

Build up images, process to determine sequence



http://www.ebi.ac.uk/training/online/course/ebi-next-generation-sequencing-practical-course/what-next-generation-dna-sequencing/illumina-





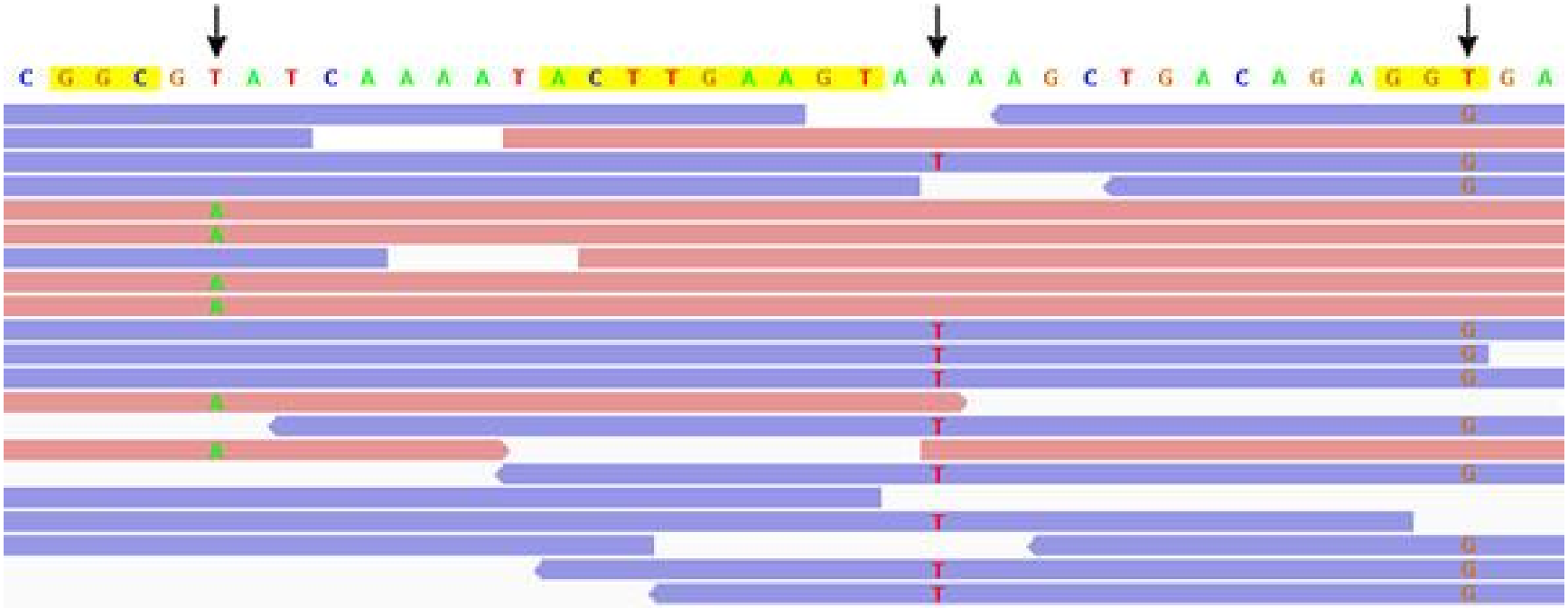
http://www.ebi.ac.uk/training/online/course/ebi-next-generation-sequencing-practical-course/what-next-generation-dna-sequencing/illumina-

What can go wrong?

1. input artifacts, problems with library prep
   a. replication in PCR has no error-correction (→ SNPs)
   b. no quaternary structures (e.g. clamp) to prevent slippage (→ indels)
   c. chimeras…
   d. duplicates (worse if they are errors)
2. Sequencing-by-synthesis
   a. phasing of step
      i. synthesis reaction efficiency is not 100%
      ii. particularly bad in A/T homopolymers
   b. certain *context specific errors*
      i. vary by sequencing protocol, device
      ii. often strand-specific

Context specific errors

Show up as strand-specific errors:



http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3622629/

Context specific errors (motifs)

Rank	Context	FER [%]	RER [%]	ERD [%]
1	ACGGCGGT	26.1	0.5	25.6
2	GTGGCGGT	25.1	0.7	24.4
3	GCGGCGGT	22.9	0.7	22.2
4	GTGGCTGT	22.4	0.6	21.8
5	ATGGCGGT	21.2	1.0	20.3
6	NCGGCGGT	20.0	0.7	19.3
7	GTGGCTTG	20.2	1.2	19.0
8	GNGGCGGT	19.2	0.7	18.5
9	GCGGCTGT	18.8	0.7	18.1
10	ACGGCTGT	18.6	0.8	17.7

← forward and reverse error rates for the ten most-common CSEs on a variety of illumina systems

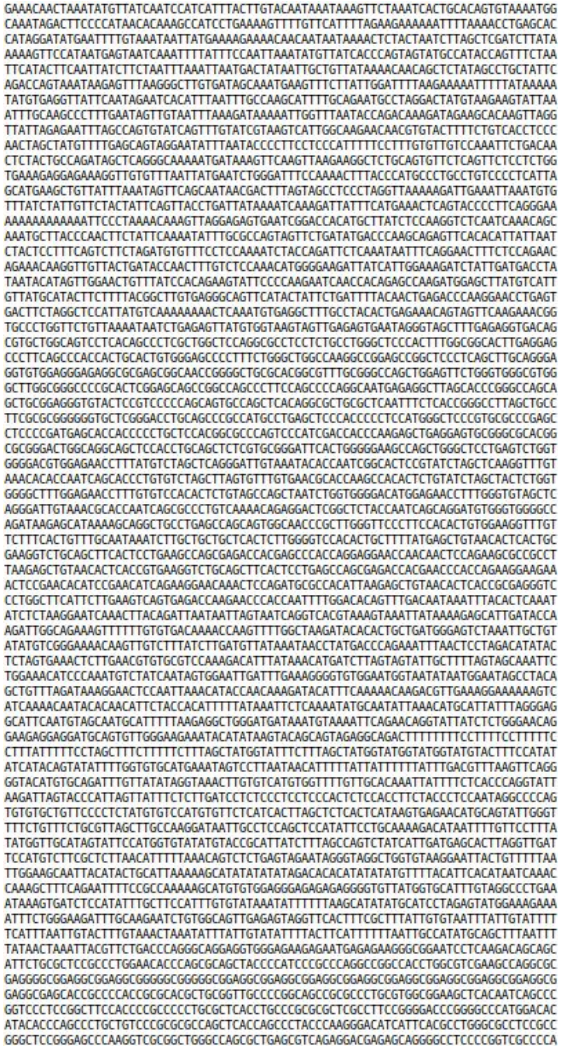
(often GC-rich)

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3622629/

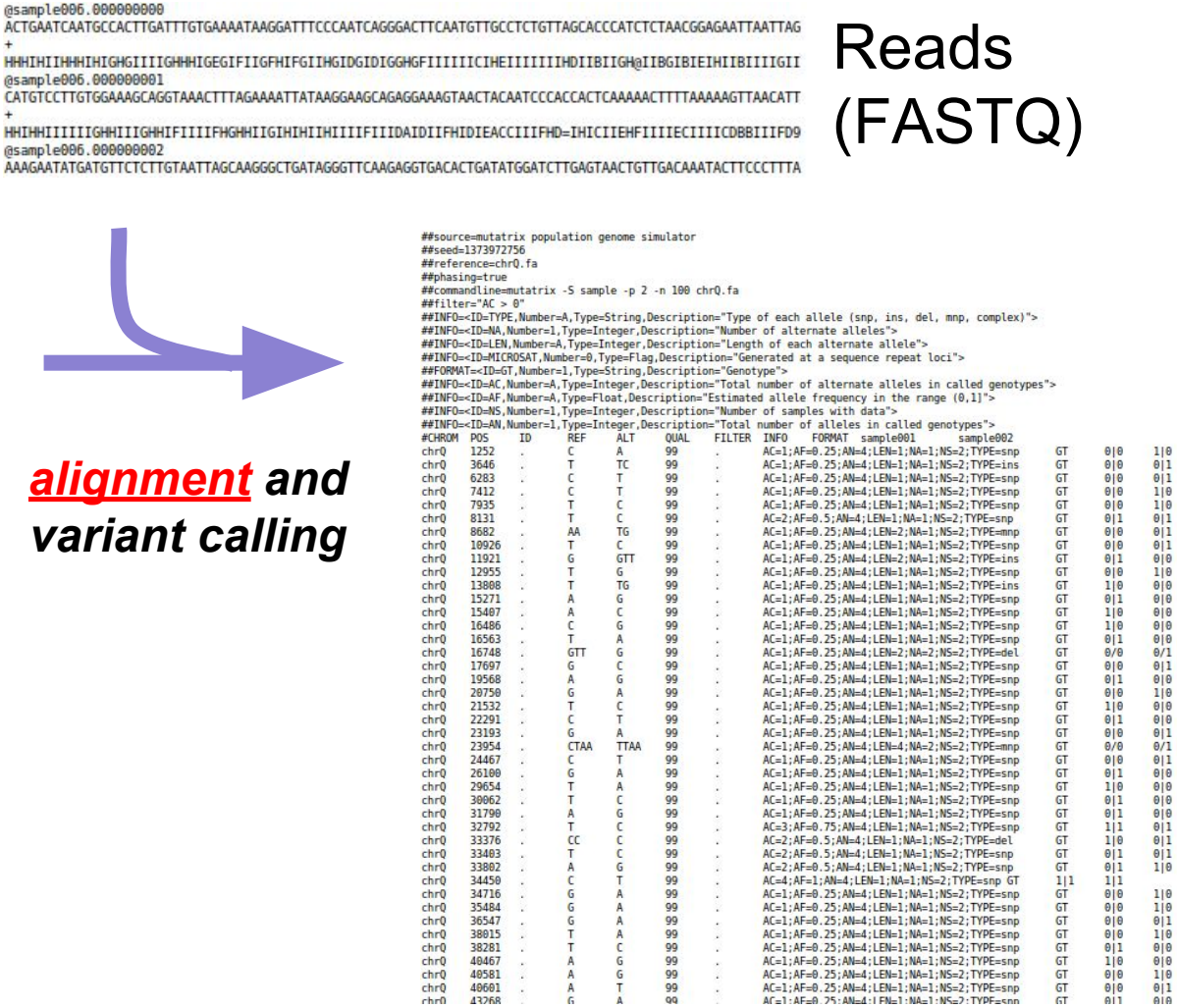
Overview

1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. **Sequence alignment**
4. Alignment-based variant detection
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

Objective: From reads to variants



Genome (FASTA)



Reads (FASTQ)

alignment* and *variant calling

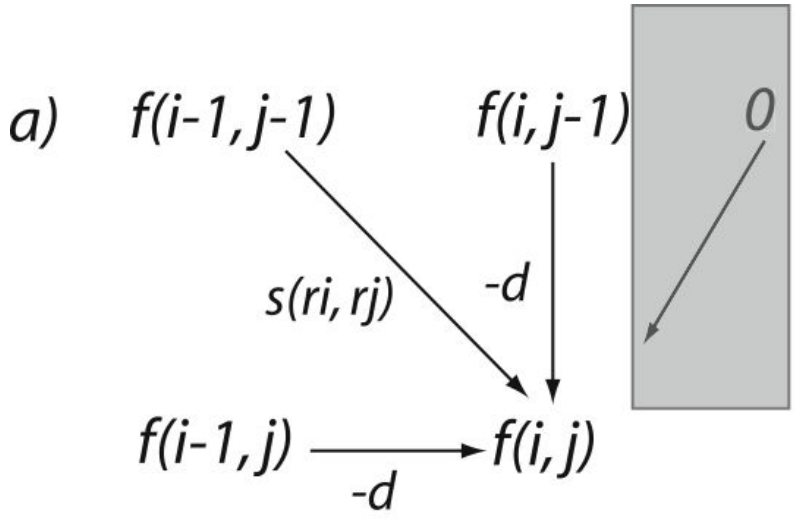
Variation (VCF)

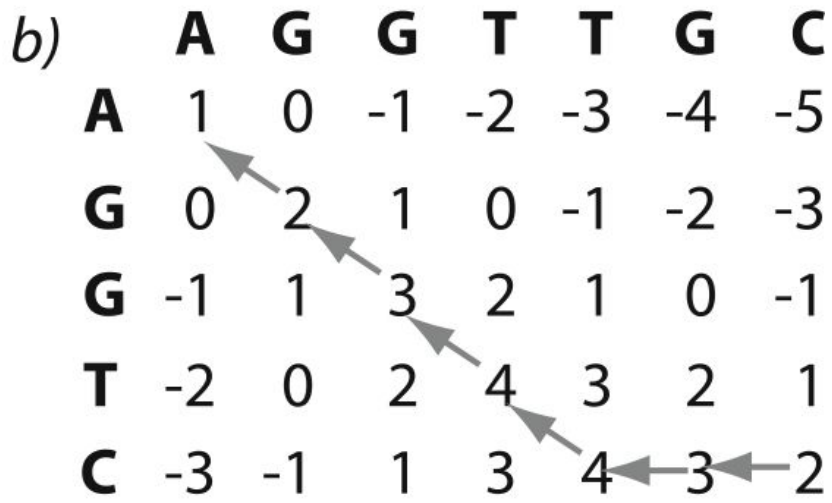
Alignment

We can produce a minimal representation of the relationship between two sequences given some scoring of mismatches and gaps via sequence alignment algorithms.

Exhaustive evaluation of all possible linear alignments between two sequences is *O(NM)* (*M* is our genome size, *N* is our query size). Various approaches allow us to do better.

Local alignment



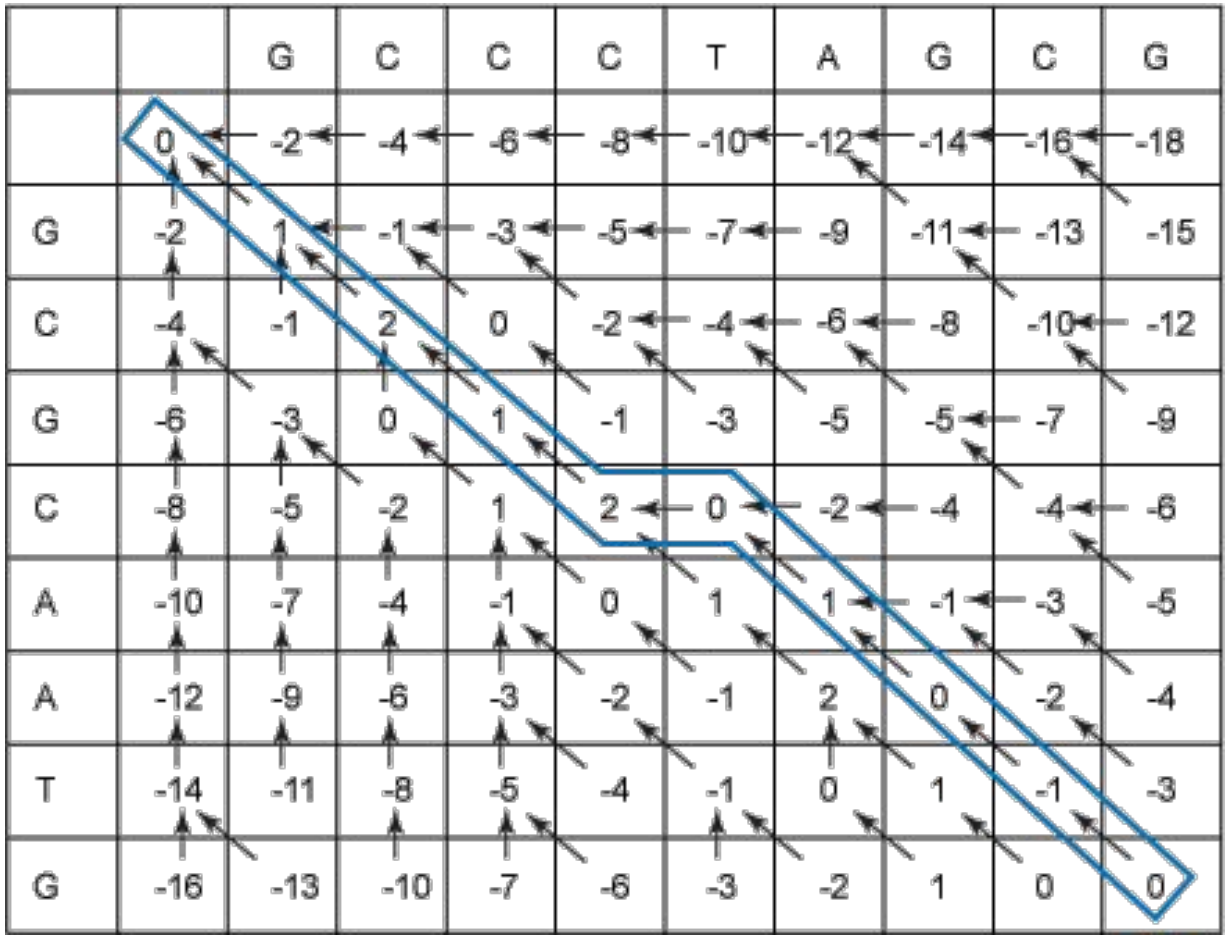


Schematic of Needleman and Wunsch algorithm. Smith and Waterman bounds scores at 0, producing locally optimal alignments. http://www.hiv.lanl.gov/content/sequence/HIV/REVIEWS/2006_7/ABECASIS/abecasis.html

A G G T T G C
A G G T - - C

Needleman-Wunsch

Filled-in Needleman-Wunsch table with traceback

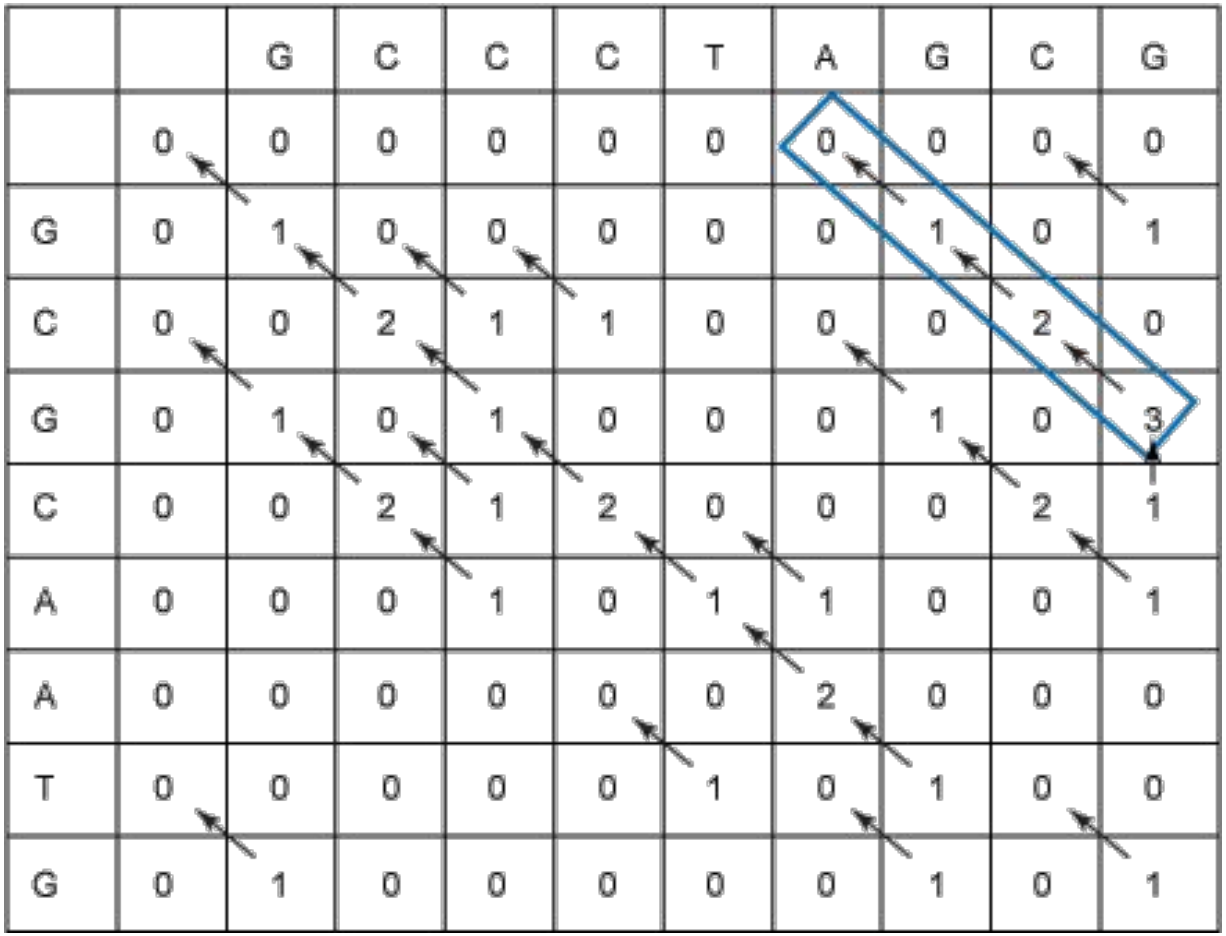


		G	C	C	C	T	A	G	C	G
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18
G	-2	1	-1	-3	-5	-7	-9	-11	-13	-15
C	-4	-1	2	0	-2	-4	-6	-8	-10	-12
G	-6	-3	0	1	-1	-3	-5	-5	-7	-9
C	-8	-5	-2	1	2	0	-2	-4	-4	-6
A	-10	-7	-4	-1	0	1	1	-1	-3	-5
A	-12	-9	-6	-3	-2	-1	2	0	-2	-4
T	-14	-11	-8	-5	-4	-1	0	1	-1	-3
G	-16	-13	-10	-7	-6	-3	-2	1	0	0

http://www.ibm.com/developerworks/library/j-seqalign/

Smith-Waterman Gotoh

Filled-in SWG matrix with traceback

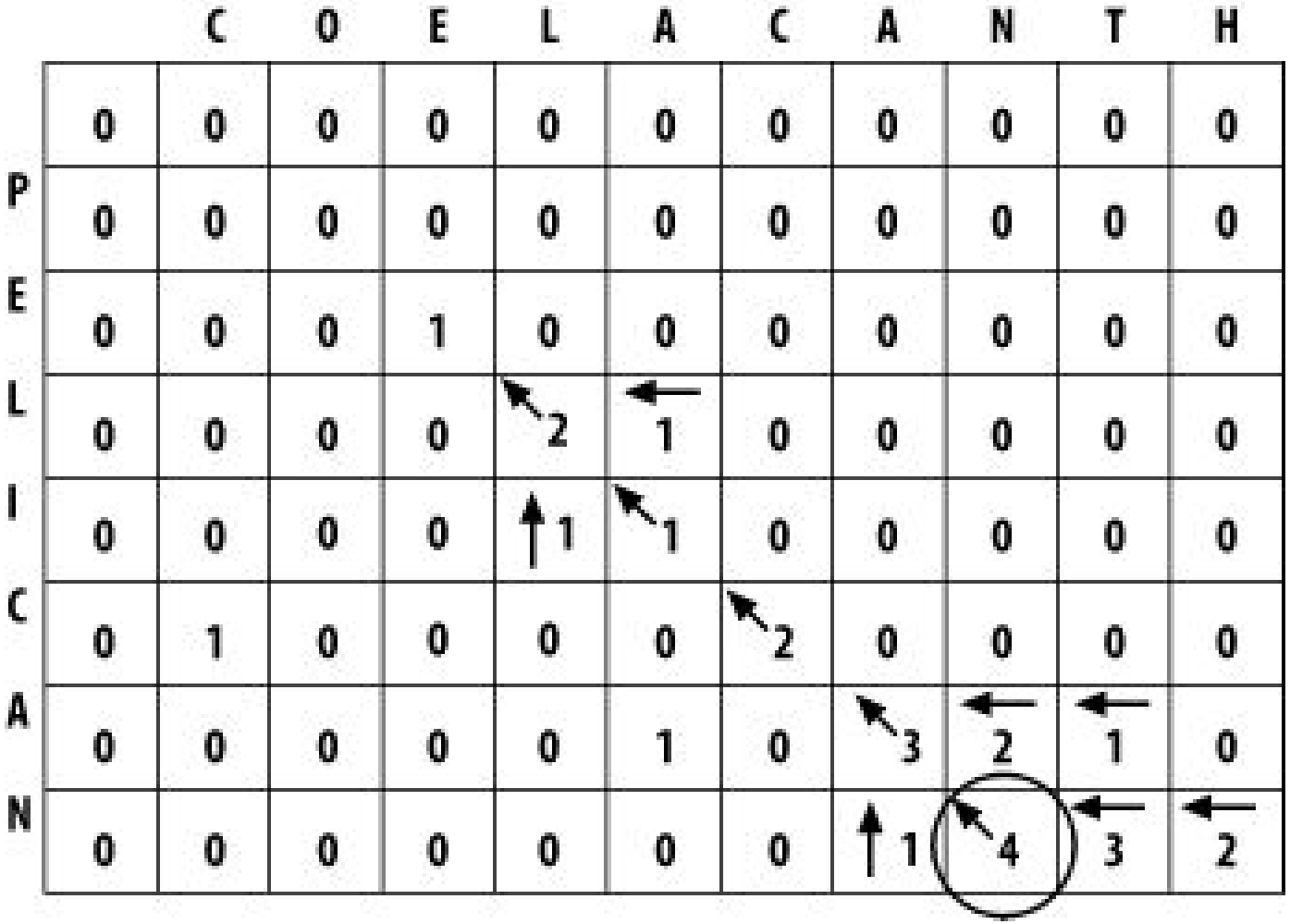


		G	C	C	C	T	A	G	C	G
	0	0	0	0	0	0	0	0	0	0
G	0	1	0	0	0	0	0	1	0	1
C	0	0	2	1	1	0	0	0	2	0
G	0	1	0	1	0	0	0	1	0	3
C	0	0	2	1	2	0	0	0	2	1
A	0	0	0	1	0	1	1	0	0	1
A	0	0	0	0	0	0	2	0	0	0
T	0	0	0	0	0	1	0	1	0	0
G	0	1	0	0	0	0	0	1	0	1

http://www.ibm.com/developerworks/library/j-seqalign/

Smith-Waterman Gotoh

Another example.



Global alignment

Problem: genomes are big. Matching sequences by evaluating all possible alignments is too **£¥€$Ƀ**

Idea: reduce the *M* in our *O(NM)* by finding a few candidate mappings quickly.

Methods:

1. k-mer based seeding
2. compressed suffix arrays / FM-indexes

k-mer based alignment

Let's break our reference genome(s) into sequences of a fixed size *k*.

Now we can make a (big) hash table:

k-mer → positions

and look for sequences of length *k* efficiently.

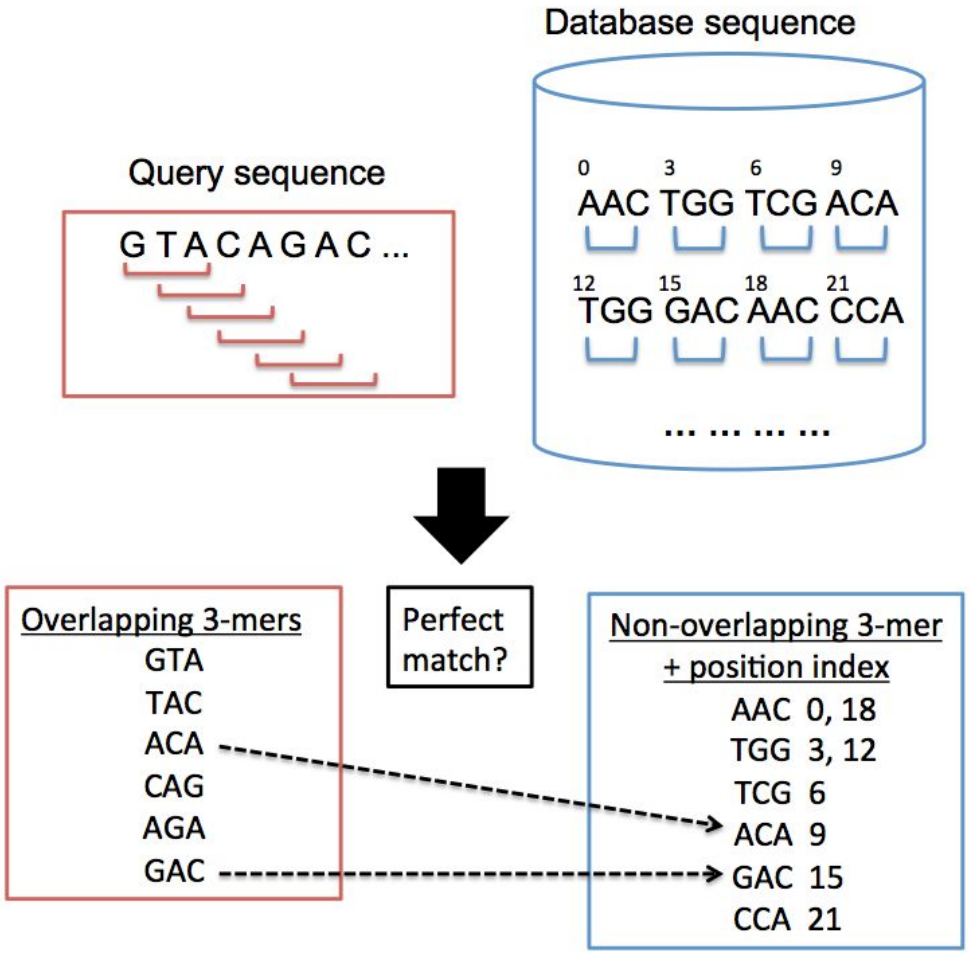
k-mer based mappers

Approach:

- Make kmers in query, match to positions in database of kmers.
- Use positional clustering to drive further alignment.

Implemented in:

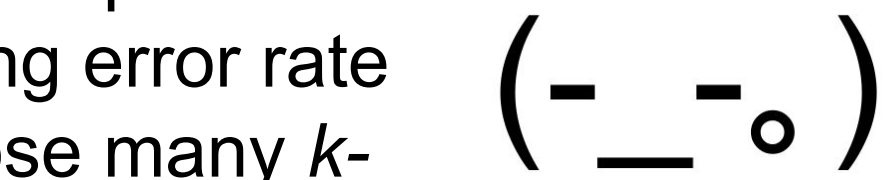
BLAT (pictured), Mosaik, Novoalign, *many* student projects.



https://en.wikipedia.org/wiki/BLAT_(bioinformatics)

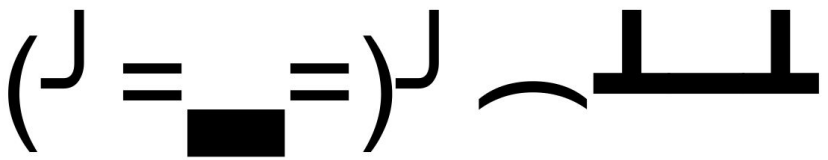
k-mer mapper limitations

If diversity between sample and genome or sequencing error rate is high enough, we lose many *k*-mer matches.

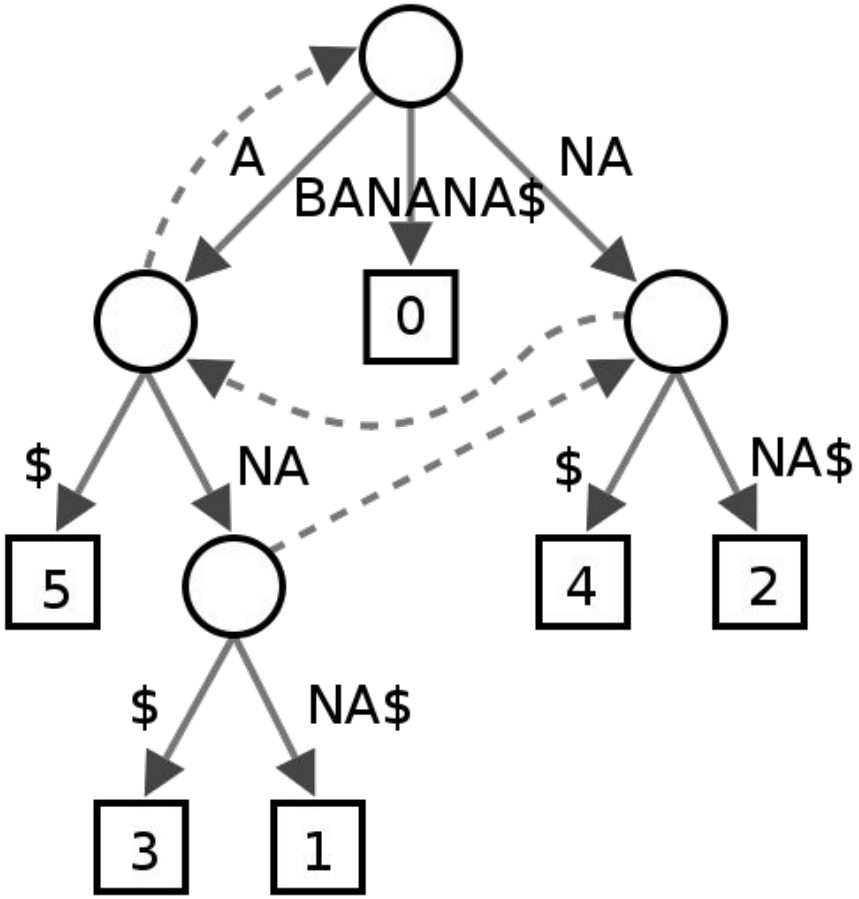


Short *k*-mers can be extremely redundant: What do we do with millions of hits?

k is a hard limit: Looking up long sequences requires stringing together many hits.



Suffix trees



A tree containing all suffixes of a string (here: **BANANA**) as paths from the root to the tree to the leaves.

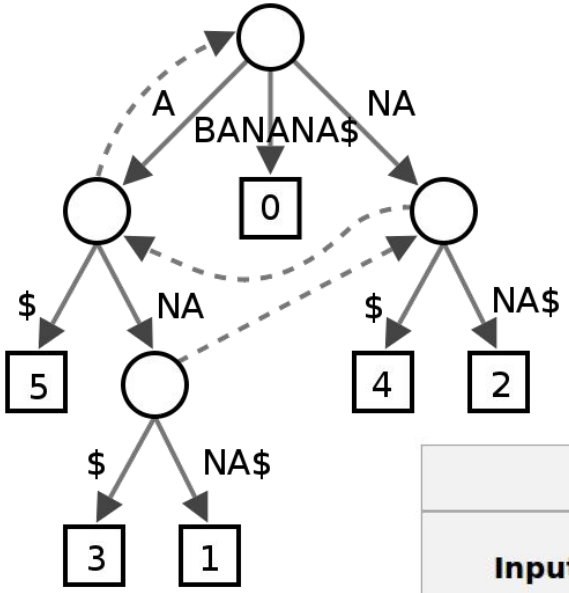
We can look up any subsequence of the string in O(|subseq|) operations!

Problem: the tree requires a *lot* of memory! Practically, 10-20X the input text.

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

*Note that we use a sentinel character $ to indicate the end of the string. The dashed lines are suffix links, which are relevant for construction but not this presentation.

Compressed suffix arrays



Suffix tree ~ Suffix array ~ BWT

BWT=Burrows-Wheeler transform

BWTs are highly compressible because they exploit repetitions in the text, but preserve the original sequence. Size: ~1X the input seq!

Transformation				
Input	**All Rotations**	**Sorting All Rows into Lex Order**	**Taking Last Column**	**Output Last Column**
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA	ANANA|^B ANA|^BAN A|^BANAN BANANA|^ NANA|^BA NA|^BANA ^BANANA| |^BANANA	BNN^AA|A

https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform

BWT searching

"This matrix has a property called 'last first (LF) mapping'. The ith occurrence of character X in the last column corresponds to the same text character as the ith occurrence of X in the first column."

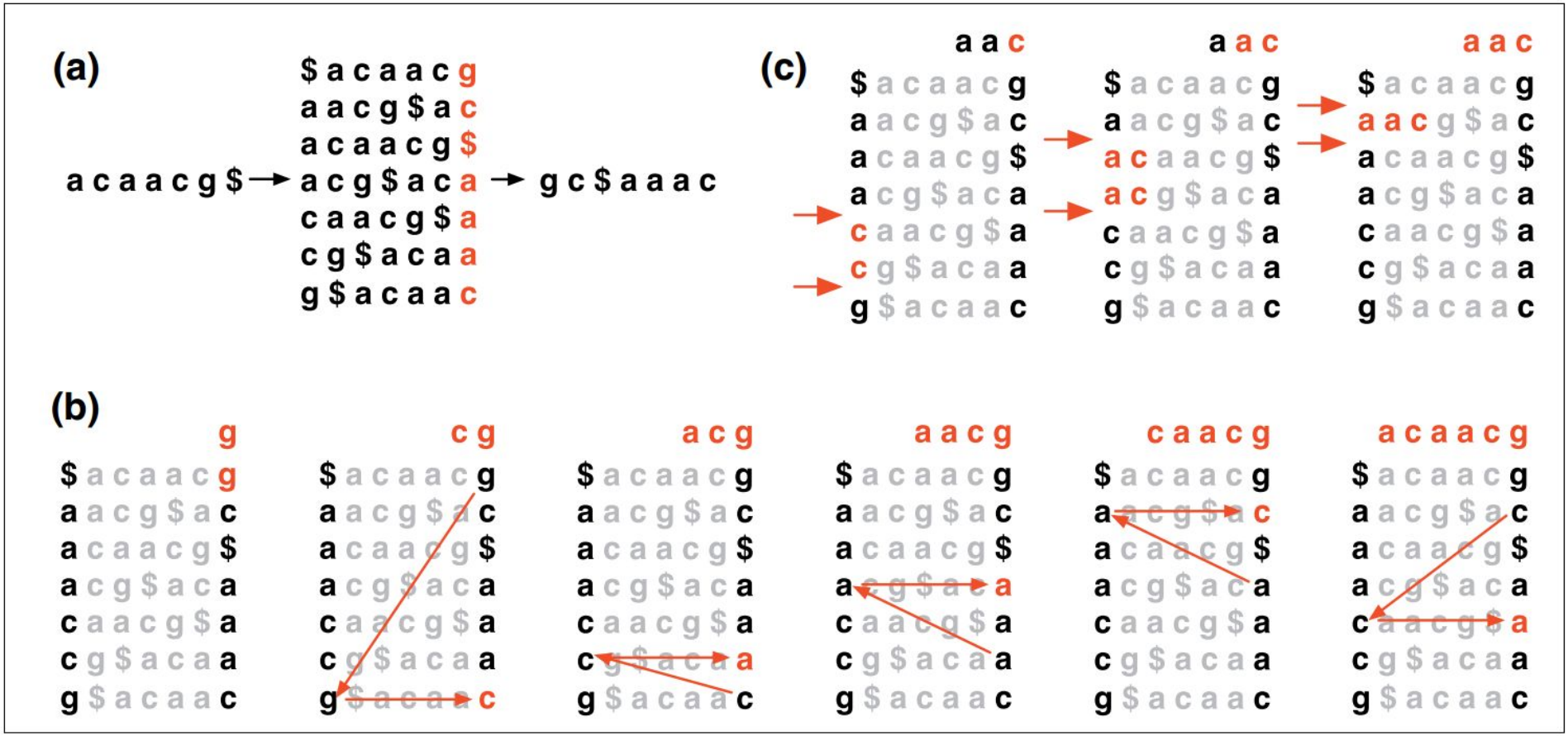


Figure 1
Burrows-Wheeler transform. **(a)** The Burrows-Wheeler matrix and transformation for 'acaacg'. **(b)** Steps taken by EXACTMATCH to identify the range of rows, and thus the set of reference suffixes, prefixed by 'aac'. **(c)** UNPERMUTE repeatedly applies the last first (LF) mapping to recover the original text (in red on the top line) from the Burrows-Wheeler transform (in black in the rightmost column).

BWT inexact matching

If the matched range reaches 0 before we reach the end of the query, then it implies a mismatch between our read and the reference.

We then backtrack and try different characters at the mismatched position until we find a match.

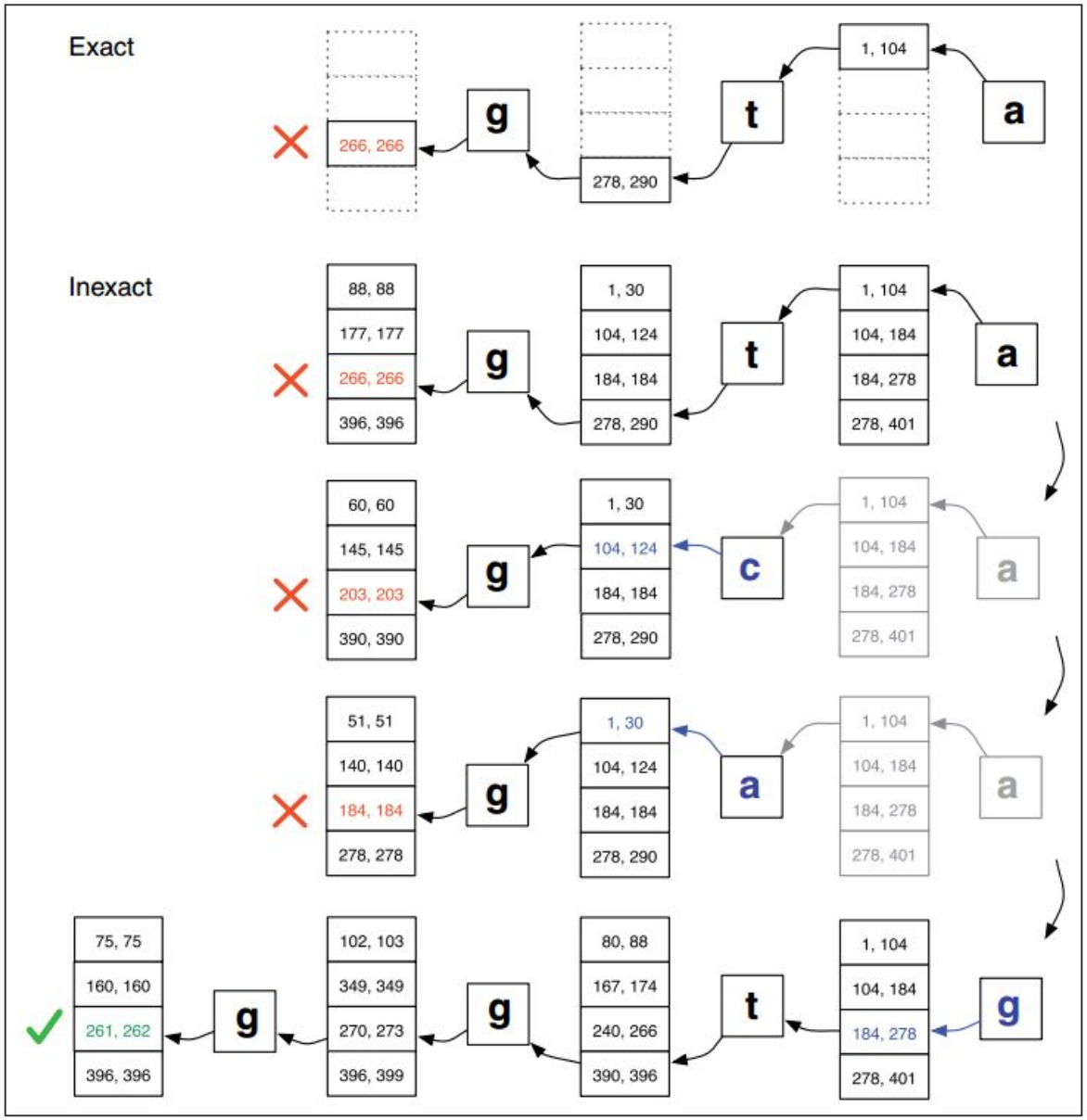


Figure 2
Exact matching versus inexact alignment. Illustration of how EXACTMATCH (top) and Bowtie's aligner (bottom) proceed when there is no exact match for query 'ggta' but there is a one-mismatch alignment when 'a' is replaced by 'g'. Boxed pairs of numbers denote ranges of matrix rows beginning with the suffix observed up to that point. A red X marks where the algorithm encounters an empty range and either aborts (as in EXACTMATCH) or backtracks (as in the inexact algorithm). A green check marks where the algorithm finds a nonempty range delimiting one or more occurrences of a reportable alignment for the query.

Bowtie, Langmead et al
http://www.genomebiology.com/2009/10/3/R25

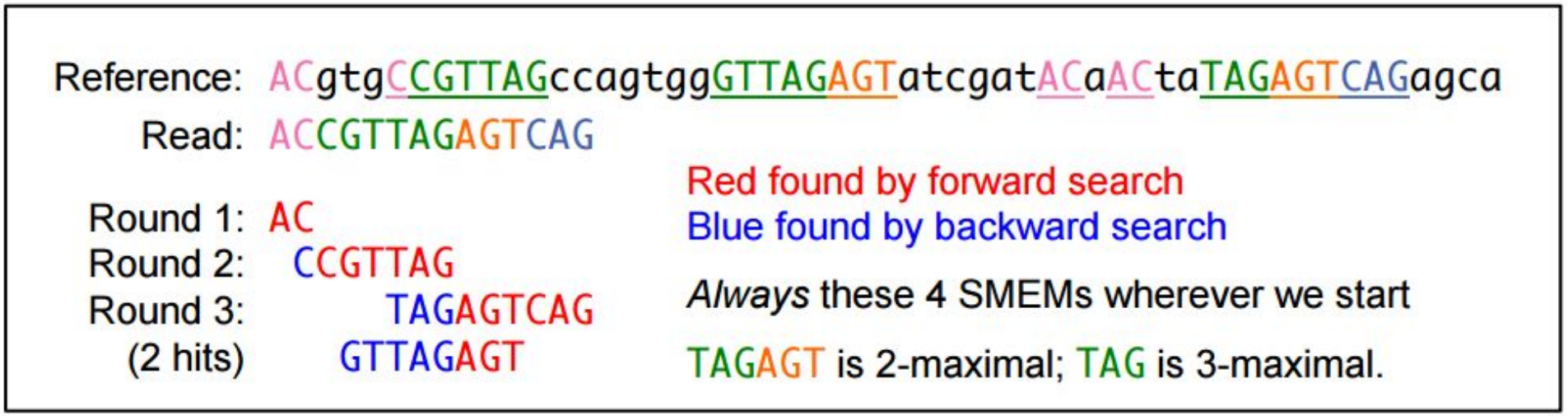
MEM mapping

Problem: can't backtrack with high error rates, long reads

Solution: find MEMs using the FM-index (bwa mem)

Maximal exact match (MEM): an exact match that cannot be extended further in either direction.

Super-maximal exact match (SMEM): a MEM that is not contained in any other MEMs on the query coordinate (Li, 2012). At any query position, the longest exact match covering the position must be a SMEM.



http://lh3lh3.users.sourceforge.net/download/mem-poster.pdf

Alignment recap

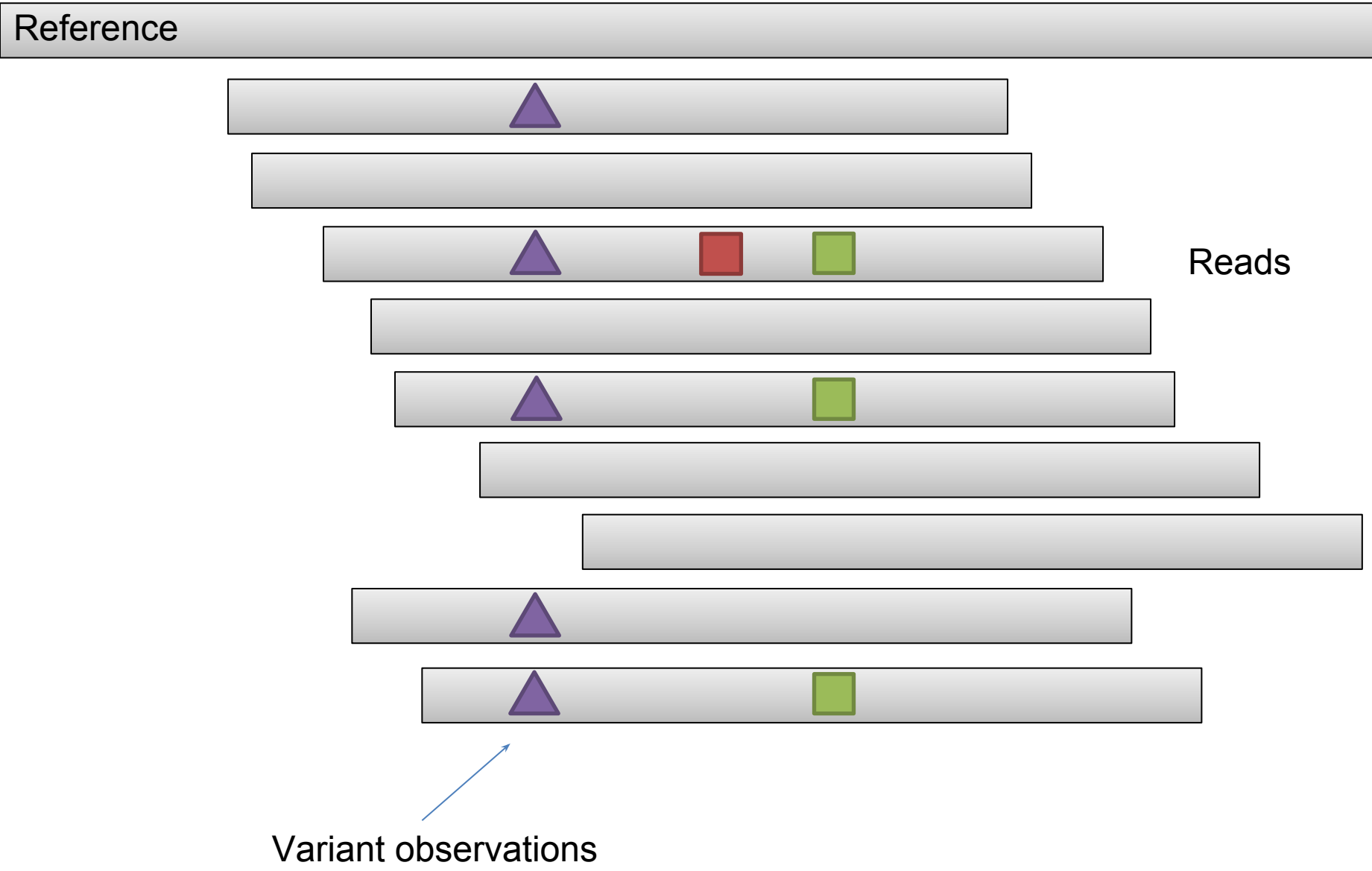
Problem: We have good algorithms for alignment, but they don't scale to large genomes (e.g. Smith-Waterman-Gotoh) or handle error gracefully (exact matching).

Solution: We use exact matching techniques to get the subset of the genome we are likely to align to. Then, we apply more expensive local alignment methods to derive a sensitive description of the relationship between query and reference.

Overview

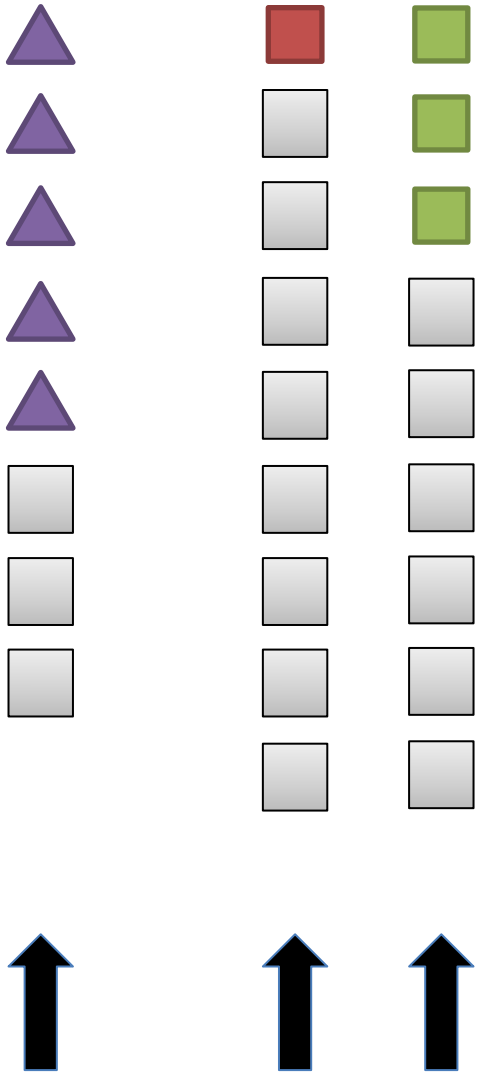
1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. Sequence alignment
4. **Alignment-based variant detection**
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

Alignments to candidates



The data exposed to the caller

Reference



Alignment-based variant calling

For each candidate locus, collect information about support for each allele. Use heuristics (e.g. varscan) or a bayesian model (e.g. samtools, GATK, freebayes) to assign maximum-likelihood alleles and genotypes:

(more on this later…)

$$Pr\left(D|G=\{A_1,A_2\}\right)=\prod_{i=1}^{M}Pr\left(b_i|G=\{A_1,A_2\}\right)$$

$$=\prod_{i=1}^{M}\left(\frac{1}{2}p\left(b_i|A_1\right)+\frac{1}{2}p\left(b_i|A_2\right)\right),$$

$$p\left(b|A\right)=\begin{cases}\frac{e}{3} & b\neq A\\ 1-e & b=A.\end{cases}$$

But wait!

If we work directly from our alignments, we may assign too much significance to the particular way our reads were aligned.

(Discuss this, then back to Bayesian model…)

Example: calling INDEL variation

Can we quickly design a process to detect indels from alignment data?

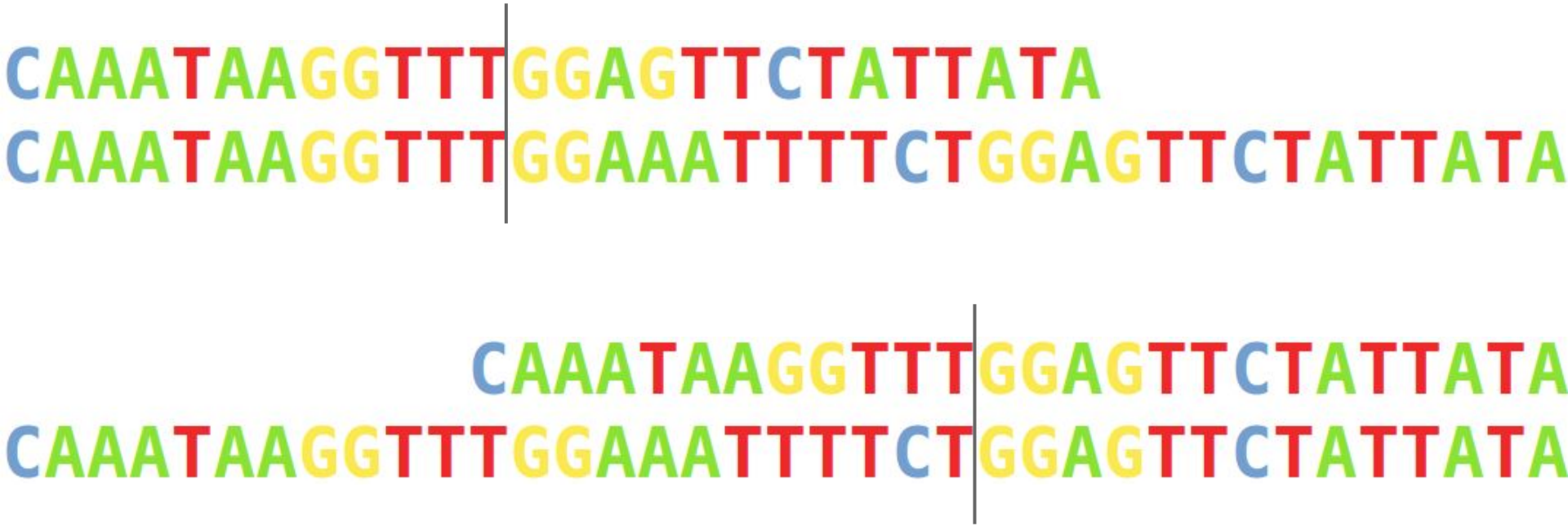
What are the steps you'd do to find the indel between these two sequences?

CAAATAAGGTTTGGAGTTCTATTATA

CAAATAAGGTTTGGAAATTTTCTGGAGTTCTATTATA

Indel finder

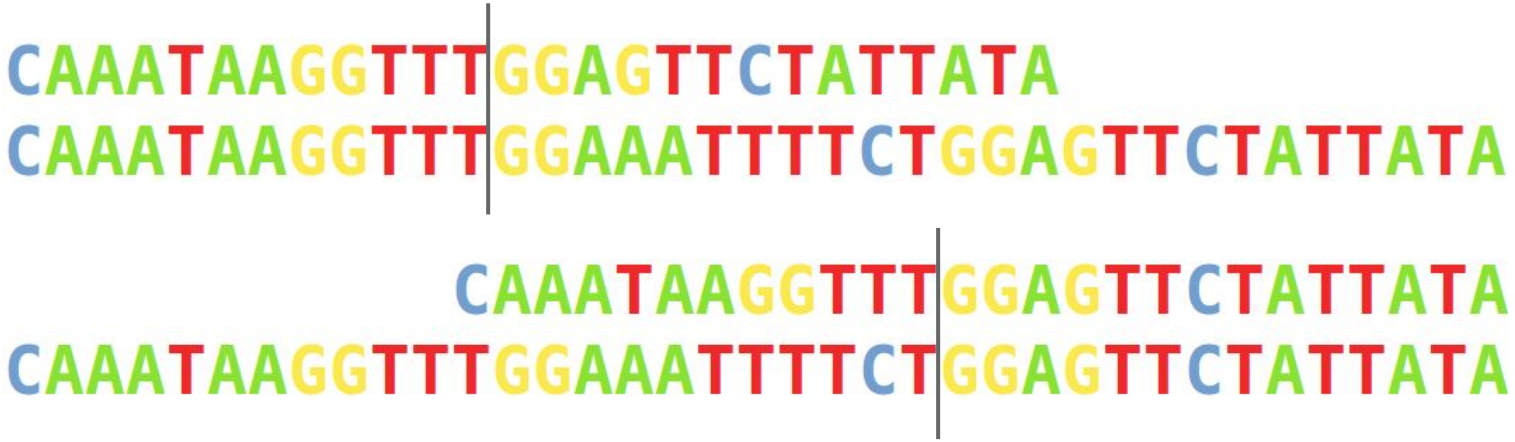
We could start by finding the long matches in both sequences at the start and end:



Indel finder

We can see this more easily like this:

CAAATAAGGTTTGGAGTTCTATTATA
CAAATAAGGTTT*GGAAATTTTCT*GGAGTTCTATTATA



Indel finder

The match structure implies that the sequence that doesn't match was inserted in one sequence, or lost from the other.

```
CAAATAAGGTT----------TGGAGTTCTATTATA
CAAATAAGGTTTGGAAATTTTCTGGAGTTCTATTATA
```

So that's easy enough….

Something more complicated

These sequences are similar to the previous ones, but with different mutations between them.

CAAATAAGGAAATTTTCTGGAGTTCTATTATA
CAAATAAGGTTTGCTATCTAGGTTATTATA

They are still (kinda) homologous but it’s not easy to see.

Pairwise alignment

One solution, assuming a particular set of alignment parameters, has 3 indels and a SNP:

```
CAAATAAGGAAATTT----TCTGGAGTTCTATTATA
CAAATAAGG---TTTGCTATCT--AGGT-TATTATA
```

But if we use a higher gap-open penalty, things look different:

```
CAAATAAGGAAATTT--TCTGGAGTTCTATTATA
CAAATAAGG---TTTGCTATCTAGGT-TATTATA
```

Alignment as interpretation

Different parameterizations can yield different results.

Different results suggest “different” variation.

What kind of problems can this cause? (And how can we mitigate these issues?)

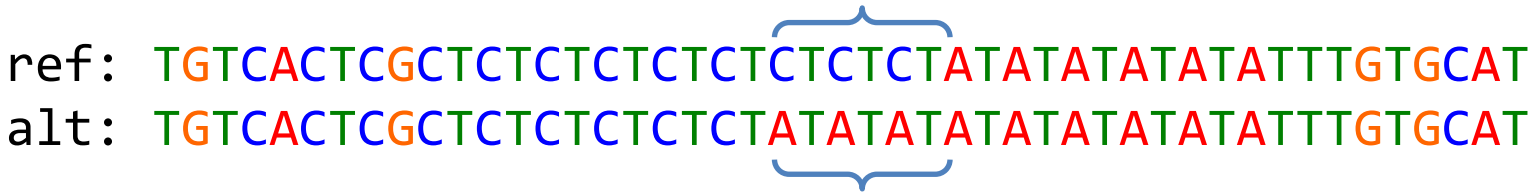
INDELs have multiple representations and require normalization for standard calling

Left alignment allows us to ensure that our representation is consistent across alignments and also variant calls.

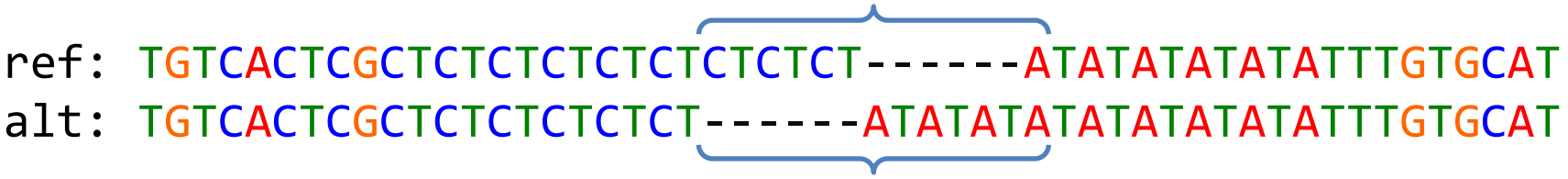
https://www.biostars.org/p/66843/ user sa9

example: 1000G PhaseI low coverage

chr15:81551110, ref:CTCTC alt:ATATA



Interpreted as 3 SNPs



Interpreted as microsatellite expansion/contraction

example: 1000G Phasel low coverage
chr20:708257, ref:AGC alt:CGA

Overview

1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. Sequence alignment
4. Alignment-based variant detection
5. **Assembly-based variant detection**
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

Problem: inconsistent variant representation makes alignment-based variant calling difficult

If alleles are represented in multiple ways, then to detect them correctly with a single-position based approach we need:

1. An awesome normalization method
2. Perfectly consistent filtering (so we represent our entire context correctly in the calls)
3. Highly-accurate reads

Solution: assembly and haplotype-driven detection

We can shift our focus from the specific interpretation in the alignments:

- this is a SNP
- whereas this is a series of indels

… and instead focus on the underlying sequences.

Basically, we use the alignments to localize reads, then process them again with assembly approaches to determine candidate alleles.

Variant detection by assembly

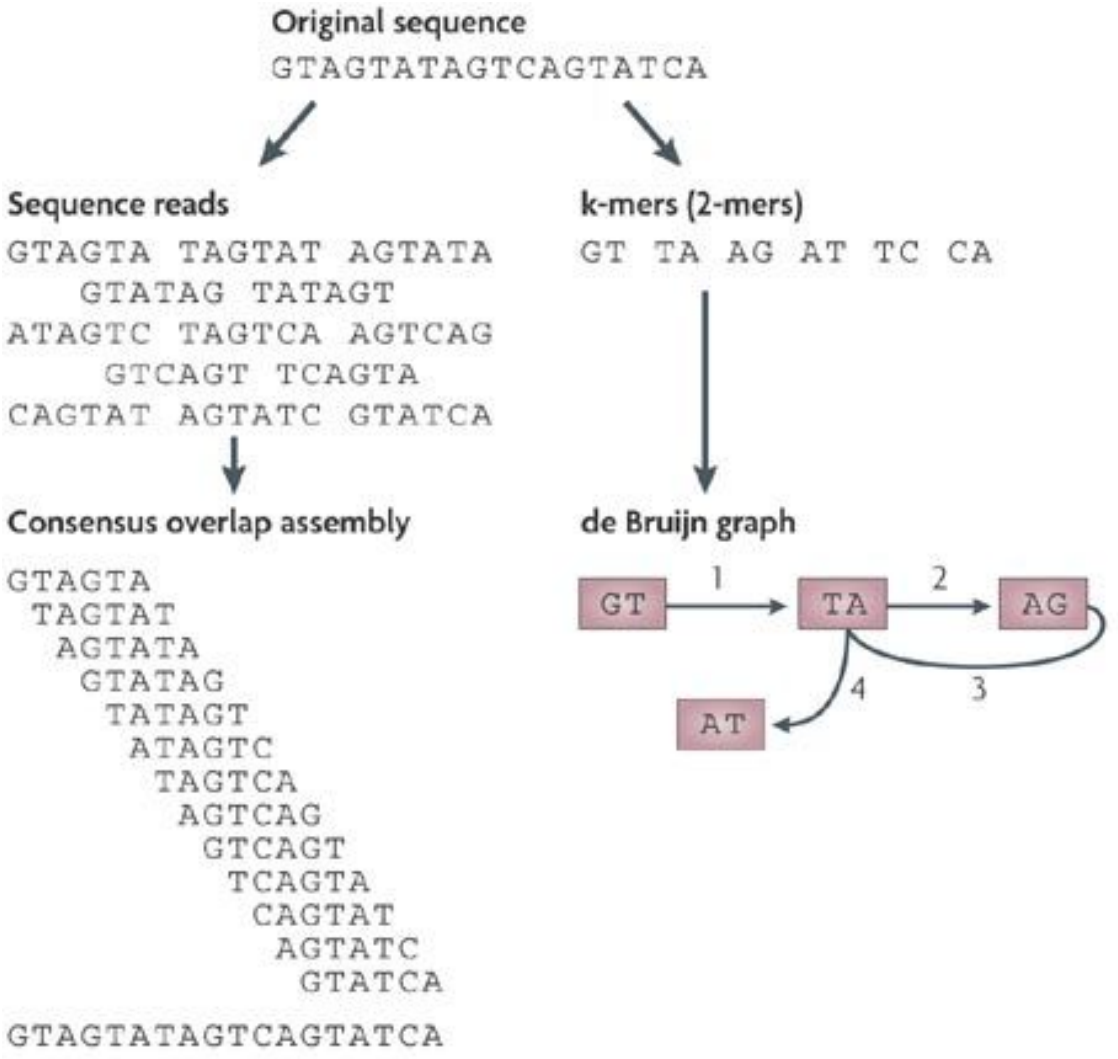
Multiple methods have been developed by members of the 1000G analysis group:

- Global joint assembly
  - cortex
  - SGA (localized to 5 megabase chunks)
- Local assembly
  - Platypus (+cortex)
  - GATK HaplotypeCaller
- k-mer based detection
  - FreeBayes (anchored reference-free windows)

Assembly

A very popular approach is the de Bruijn graph, which is a graph with *k*-mer labels on the nodes and edges where the tail of one sequence is equal to the head of the next.

An assembly can be produced in O(|E|) as a Eulerian path through the graph. (Fast but not necessarily accurate.)



Nature Reviews | Microbiology

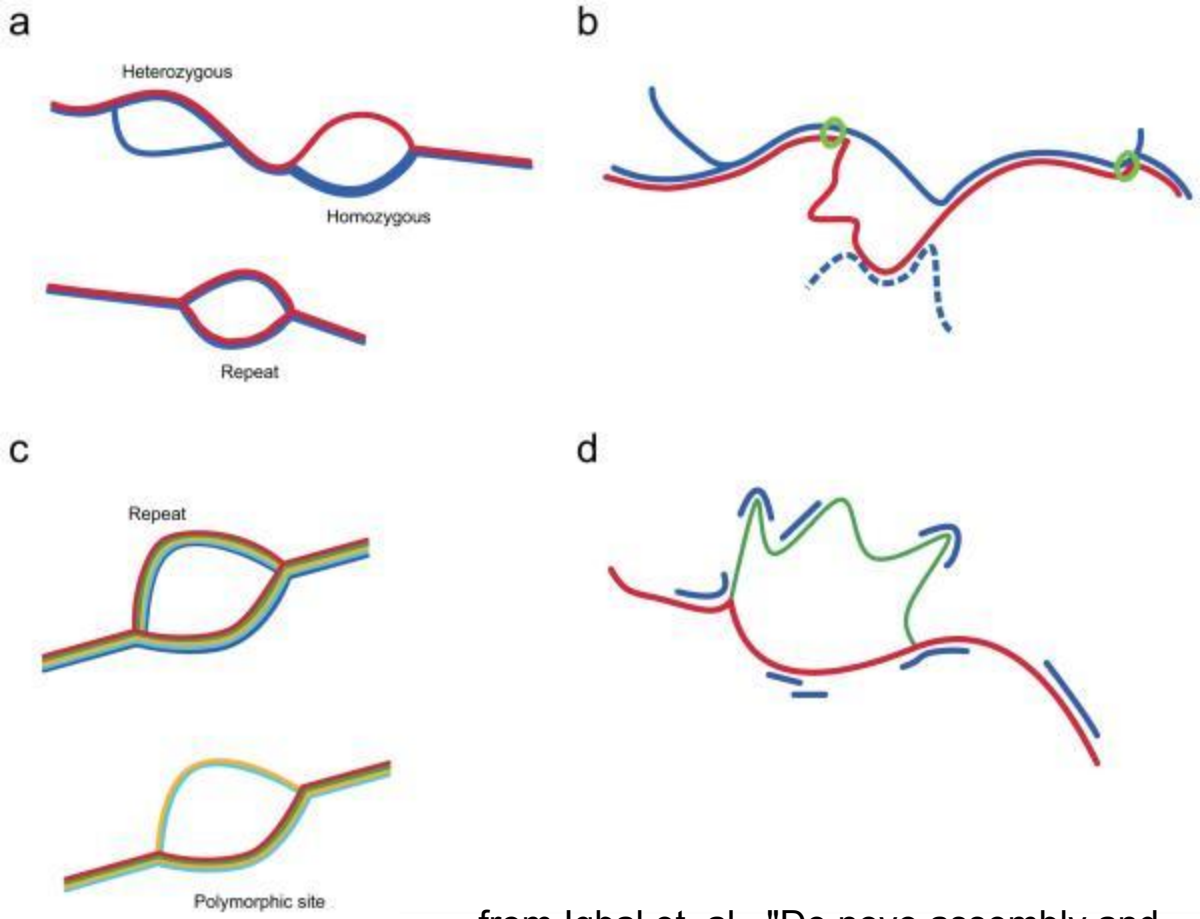
http://www.nature.com/nrmicro/journal/v7/n4/full/nrmicro2088.html

Using colored graphs (Cortex)

Variants can be called using bubbles in deBruijn graphs.

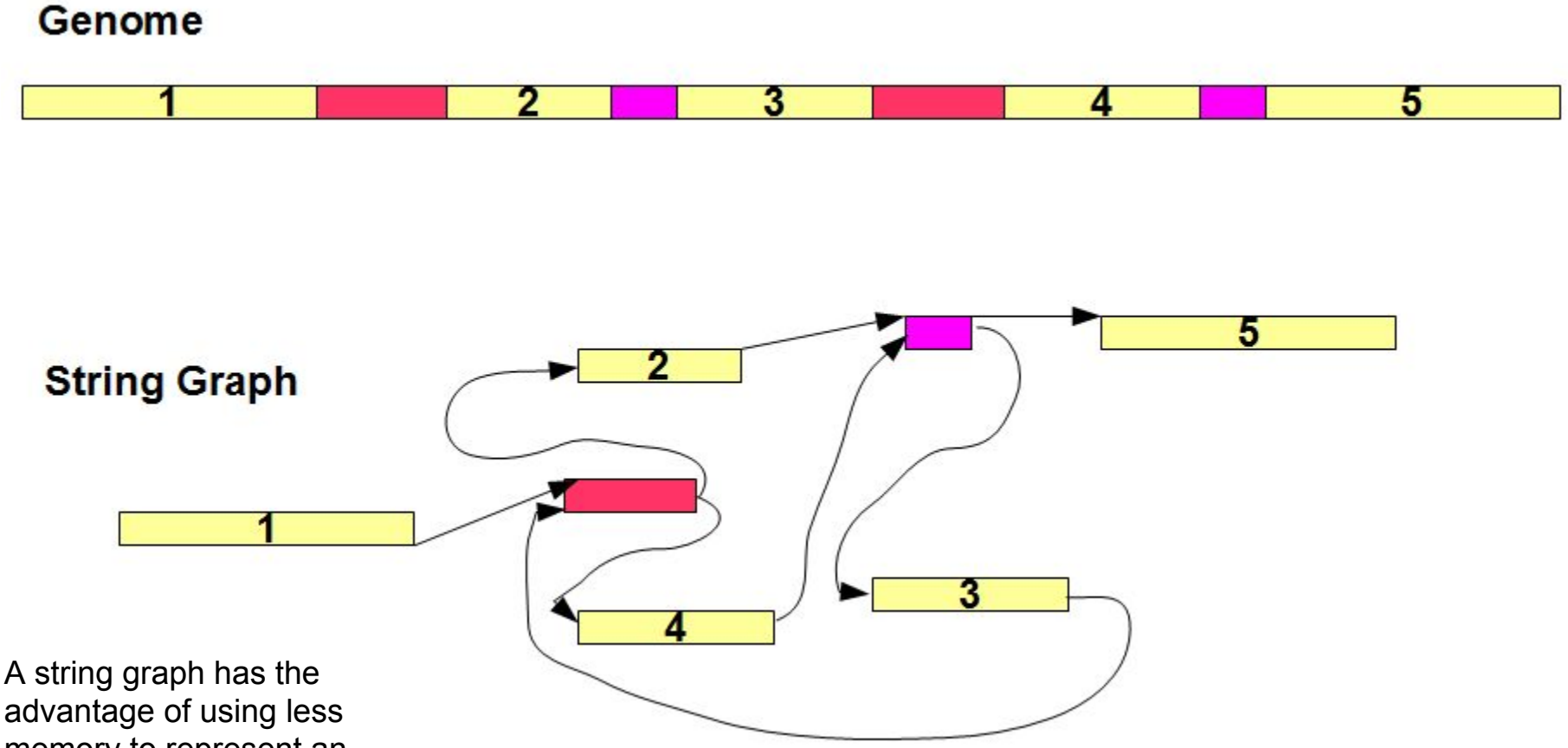
Method is completely reference-free, except for reporting of variants. The reference is threaded through the colored graph.

Many samples can be called at the same time.



from Iqbal et. al., "De novo assembly and genotyping of variants using colored de Bruijn graphs." (2012)

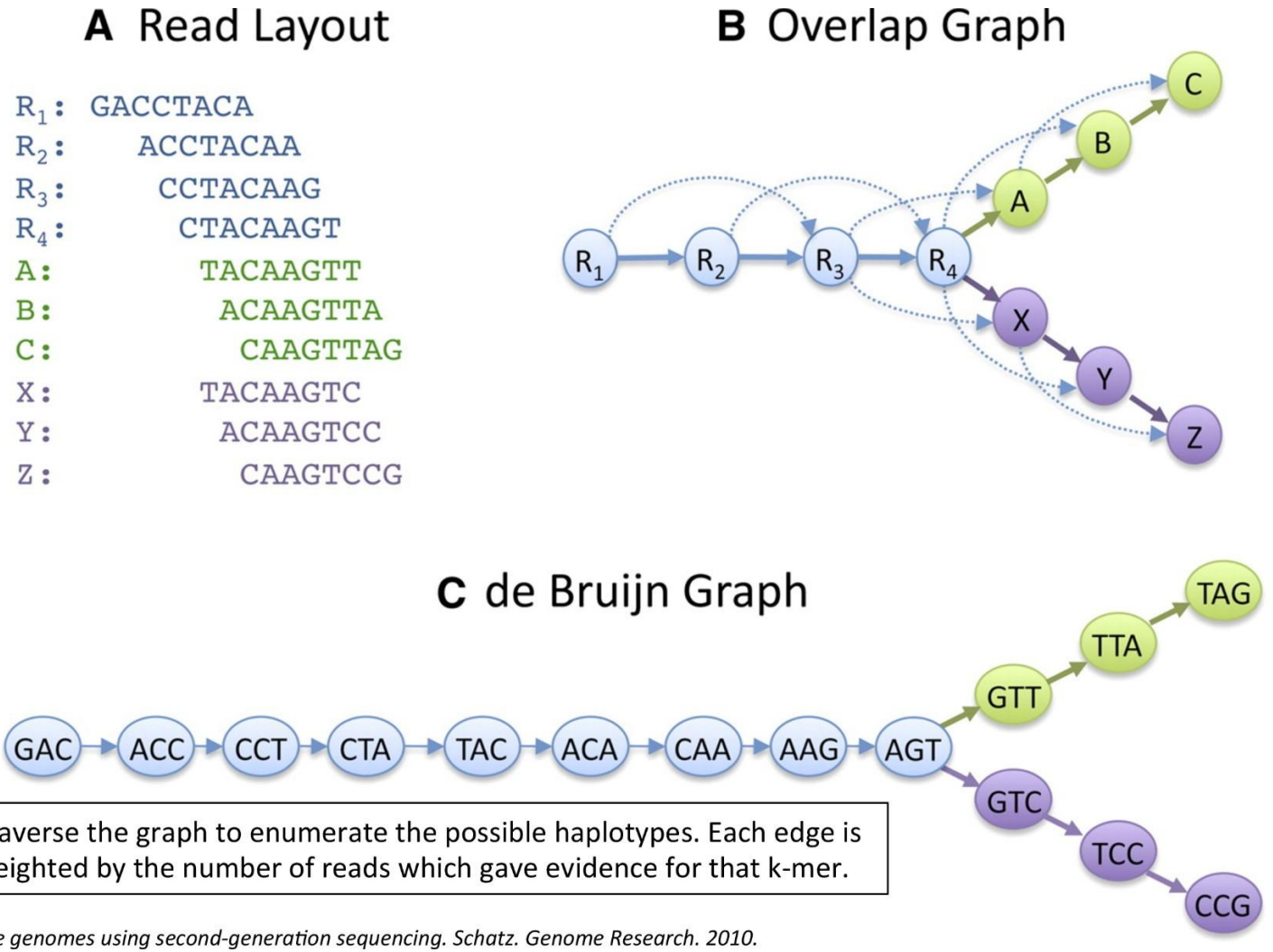
String graphs (SGA)



A string graph has the advantage of using less memory to represent an assembly than a de Bruijn graph. In the 1000G, SGA is run on alignments localized to ~5mb chunks.

http://www.homolog.us/blogs/2012/02/13/string-graph-of-a-genome/

Discovering alleles using graphs (GATK HaplotypeCaller)

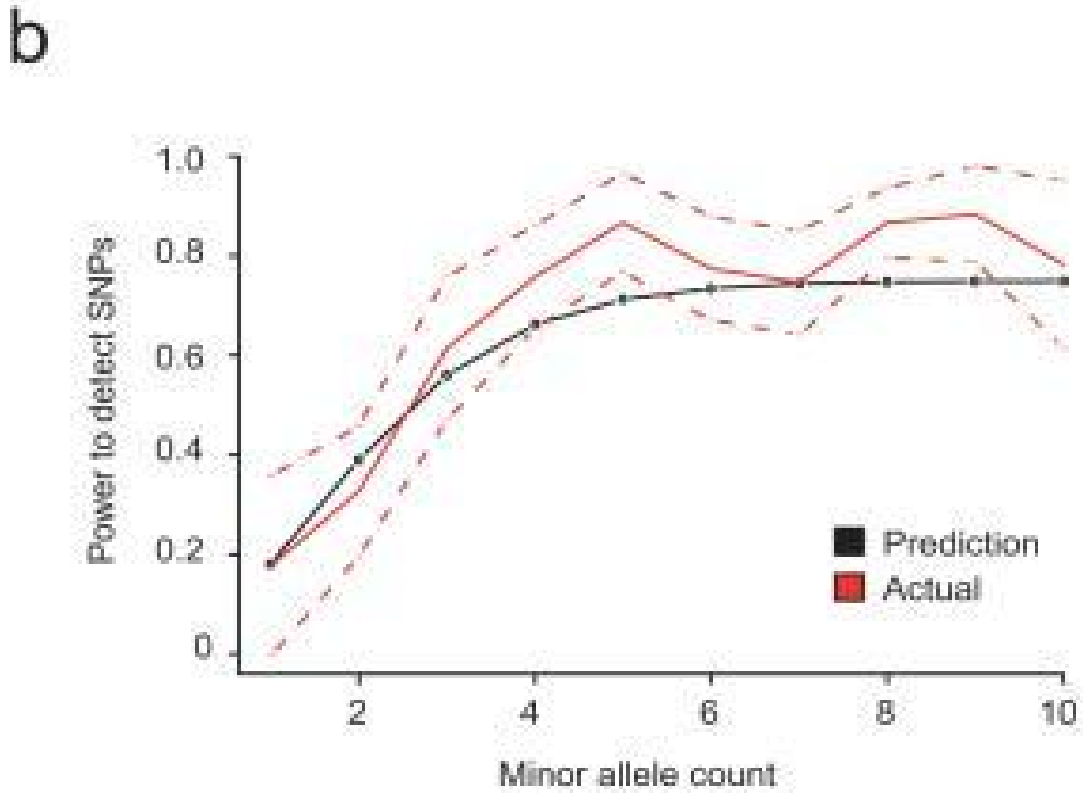


Traverse the graph to enumerate the possible haplotypes. Each edge is weighted by the number of reads which gave evidence for that k-mer.

Assembly of large genomes using second-generation sequencing. Schatz. Genome Research. 2010.

Why don't we just assemble?

Assembly-based calls tend to have high specificity, but sensitivity suffers.



The requirement of exact kmer matches means that errors disrupt coverage of alleles.

Existing assembly methods don't just detect point mutations--- they detect haplotypes.

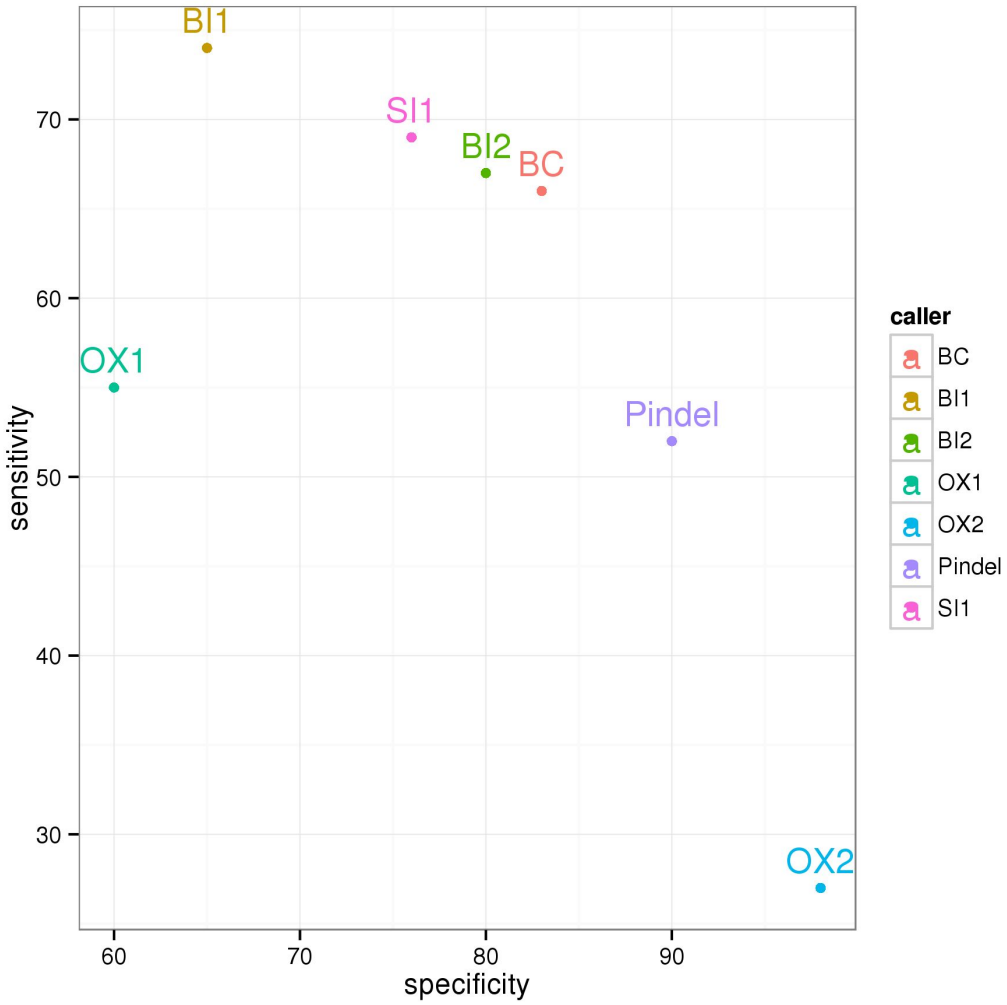
from Iqbal et. al., "De novo assembly and genotyping of variants using colored de Bruijn graphs." (2012)

Indel validation, 191 AFR samples

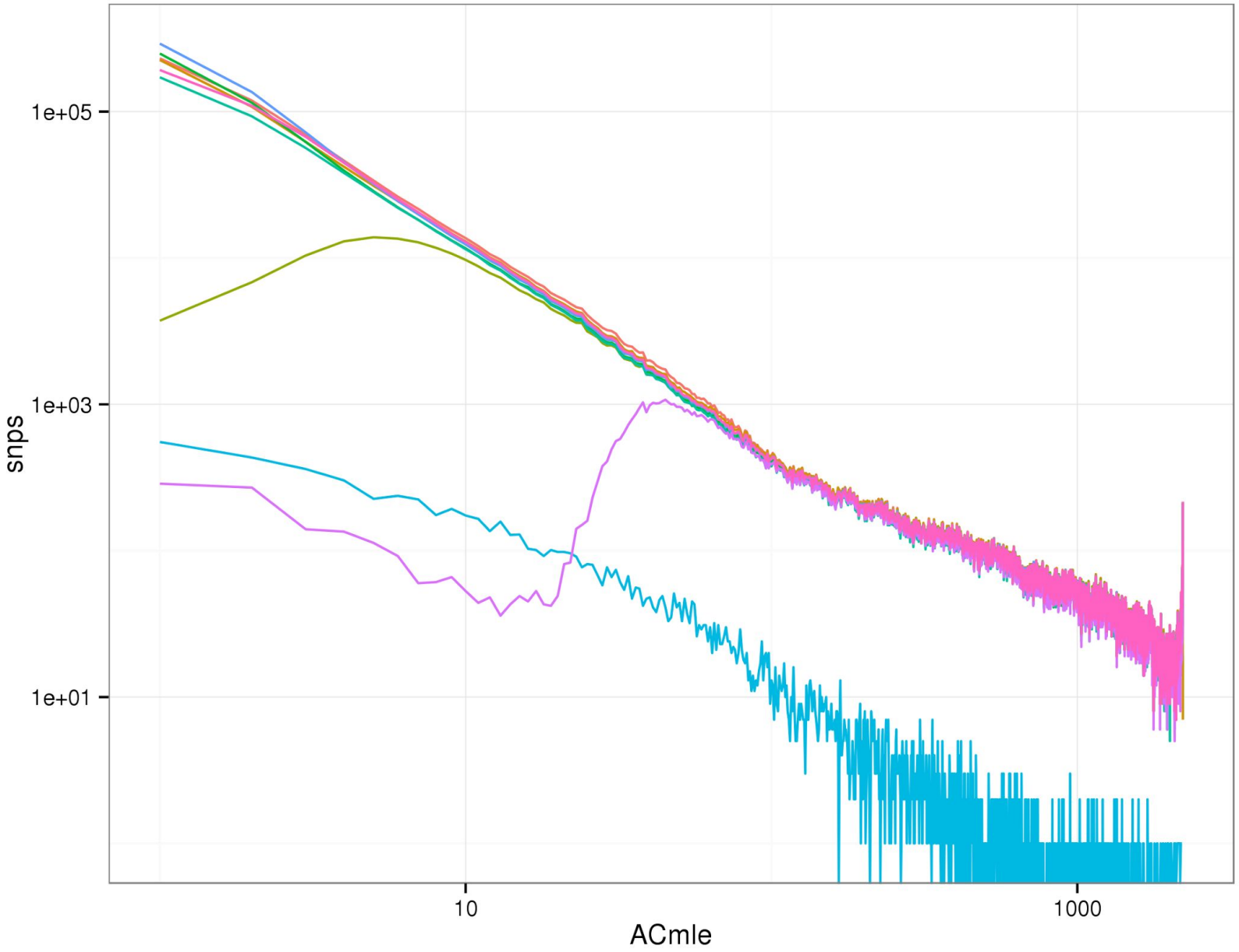
High-depth miSeq sequencing-based validation on 4 samples.

Local assembly methods (BI2, BC, SI1)* have higher specificity than baseline mapping-based calls (BI1), but lower sensitivity. Global assembly (OX2) yielded very low error, but also low sensitivity.

*The local assembly-based method Platypus (OX1) had a genotyping bug which caused poor performance.



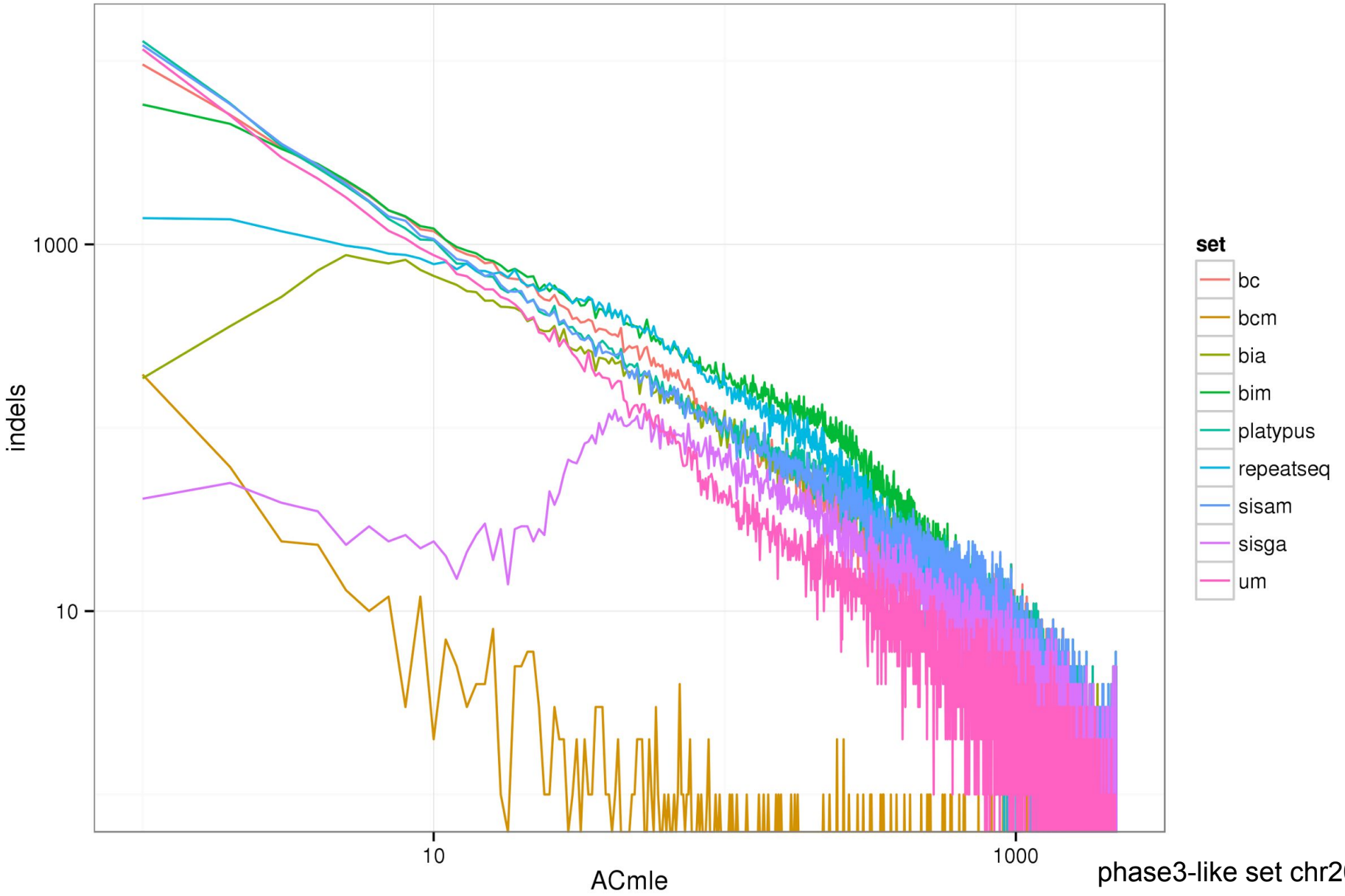
Site-frequency spectrum, SNPs



phase3-like set
chr20

*RepeatSeq is a microsatellite caller

Site-frequency spectrum, indels



phase3-like set chr20

Overview

1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. Sequence alignment
4. Alignment-based variant detection
5. Assembly-based variant detection
6. **Haplotype-based variant detection**
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

Finding haplotype polymorphisms

Two reads

AGAACCCAGTGCTCTTTCTGCT

AGAACCCAGTGGTCTTTCTGCT

a SNP

AGAACCCAGTG C/G TCTTTCTGCT

Their alignment

Another read showing a SNP on the same haplotype as the first

AGAACCCAGTGCTCTATCTGCT

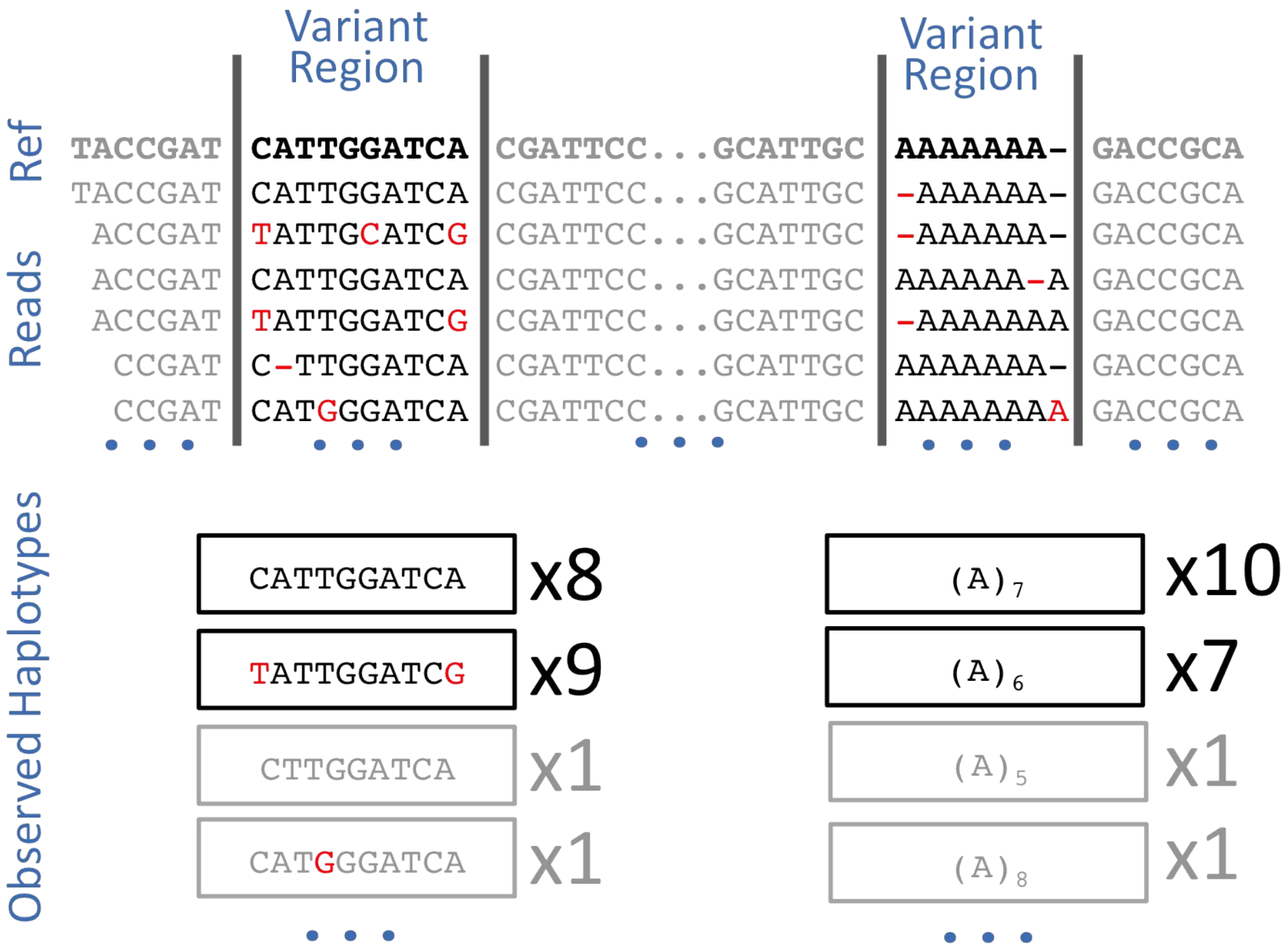
AGAACCCAGTG CTCTA/GTCTT TCTGCT

A variant locus implied by alignments

Direct detection of haplotypes

Direct detection of haplotypes from reads resolves differentially-represented alleles (as the sequence is compared, not the alignment).

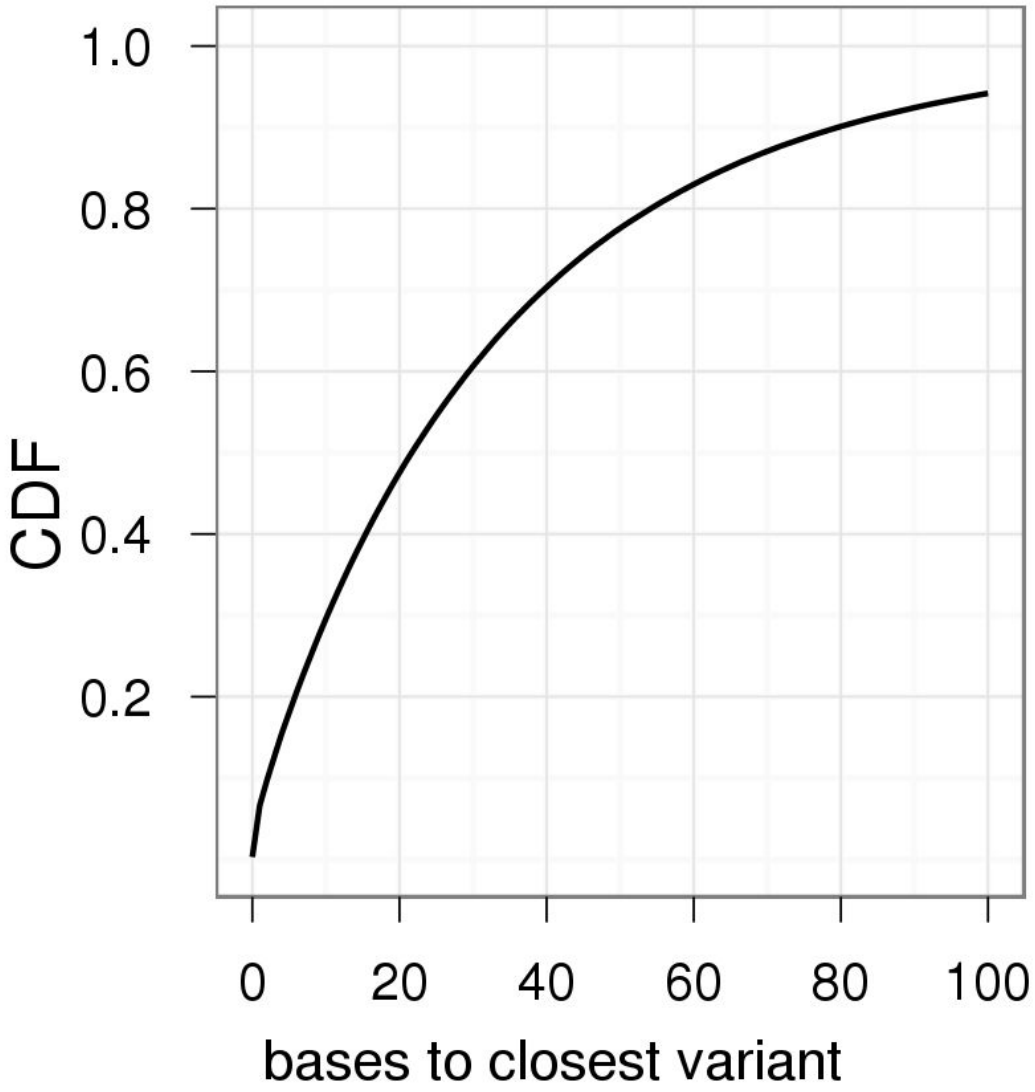
Allele detection is still alignment-driven.


Variant Region
Variant Region
Ref
Reads
TACCGAT CATTGGATCA CGATTCC...GCATTGC AAAAAAA- GACCGCA
TACCGAT CATTGGATCA CGATTCC...GCATTGC -AAAAAA- GACCGCA
ACCGAT TATTGCATCG CGATTCC...GCATTGC -AAAAAA- GACCGCA
ACCGAT CATTGGATCA CGATTCC...GCATTGC AAAAAA-A GACCGCA
ACCGAT TATTGGATCG CGATTCC...GCATTGC -AAAAAAA GACCGCA
CCGAT C-TTGGATCA CGATTCC...GCATTGC AAAAAAA- GACCGCA
CCGAT CATGGGATCA CGATTCC...GCATTGC AAAAAAAA GACCGCA
Observed Haplotypes
CATTGGATCA x8
TATTGGATCG x9
CTTGGATCA x1
CATGGGATCA x1
(A)7 x10
(A)6 x7
(A)5 x1
(A)8 x1

Why haplotypes?

- Variants cluster.
- This has functional significance.
- Observing haplotypes lets us be more certain of the local structure of the genome.
- We can improve the detection process itself by using haplotypes rather than point mutations.
- We get the sensitivity of alignment-based approaches with the specificity of assembly-based ones.

Sequence variants cluster



In ~1000 individuals, ½ of variants are within ~22bp of another variant.

Variance to mean ratio (VMR) = 1.4.

The functional effect of variants depends on other nearby variants on the same haplotype

```
reference:  AGG GAG CTG
            Arg Glu Leu

apparent:   AGG TAG CTG
            Arg Ter ---

actual:     AGG TTG CTG
            Arg Leu Leu
```

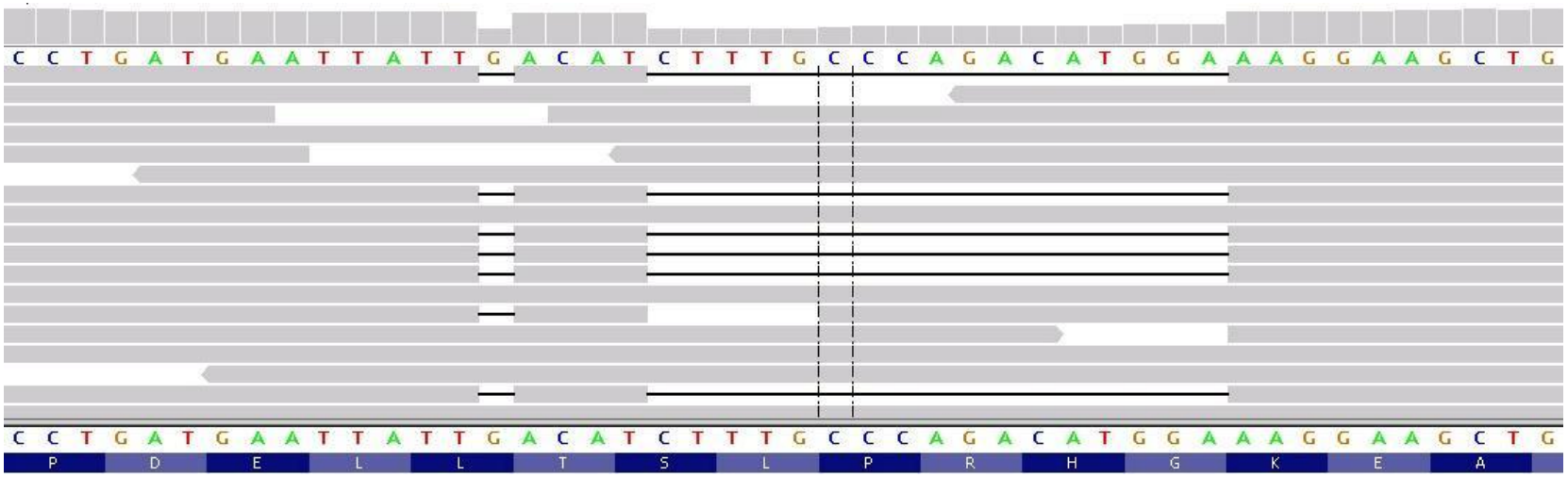
OTOF gene – mutations cause profound recessive deafness

Apparent nonsense variant, one YRI homozygote

Actually a block substitution that results in a missense substitution

(Daniel MacArthur)

Importance of haplotype effects: frame-restoring indels



(in NA12878)

- Two apparent frameshift deletions in the *CASP8AP2* gene (one 17 bp, one 1 bp) on the same haplotype
- Overall effect is in-frame deletion of six amino acids

(Daniel MacArthur)

Frame-restoring indels in 1000 Genomes Phase I exomes

chr6:117113761, GPRC6A **(~10% AF in 1000G)**

```
ref: ATTGTAATTCTCA--TA--TT--TGCCTTTGAAAGC
alt: ATTGTAATTCTCAGGTAATTTCCTGCCTTTGAAAGC
```

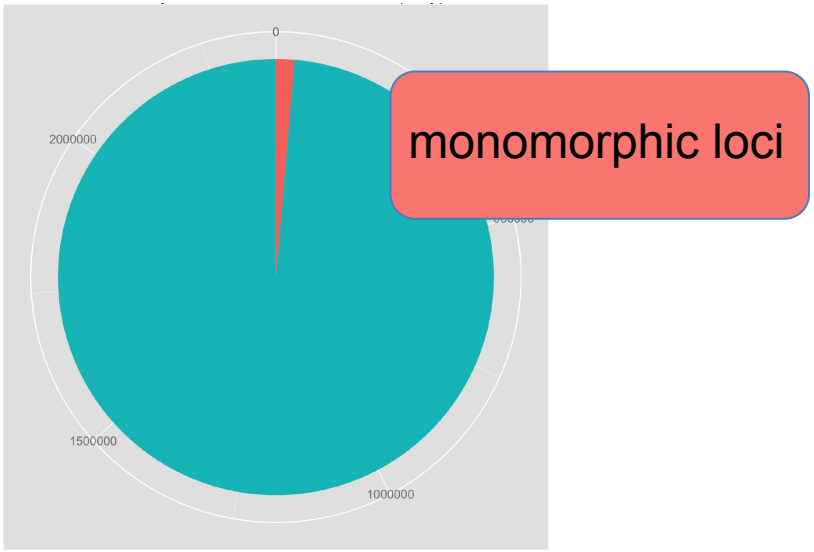
chr6:32551935, HLA-DRB1 **(~11% AF in 1000G)**

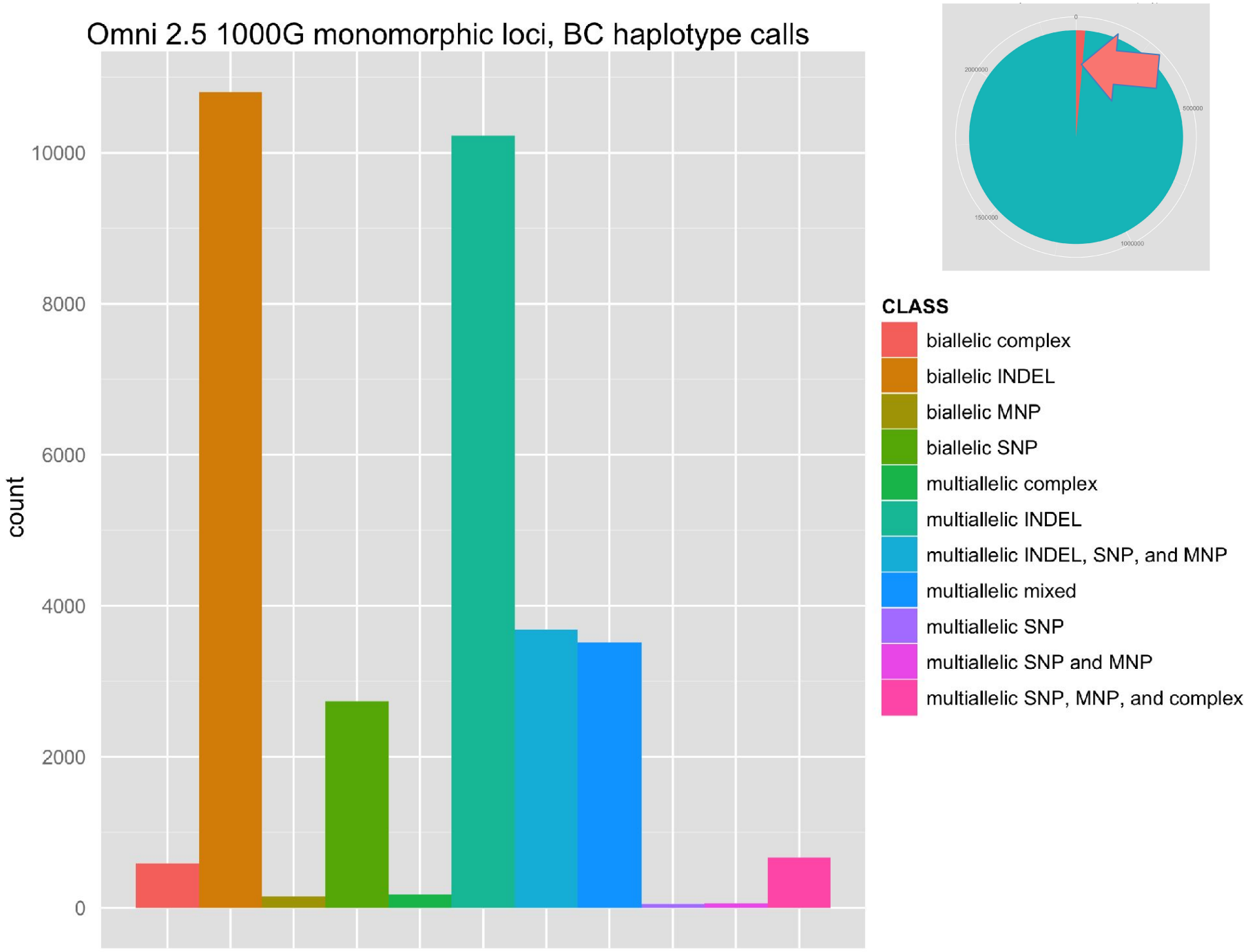
```
 ref: CCACCGCGGCCCGCGCCTG-C-TCCAGGATGTCC
 Alt: CCACCGCGG--CGCGCCTGTCTTCCAGGAGGTCC
```

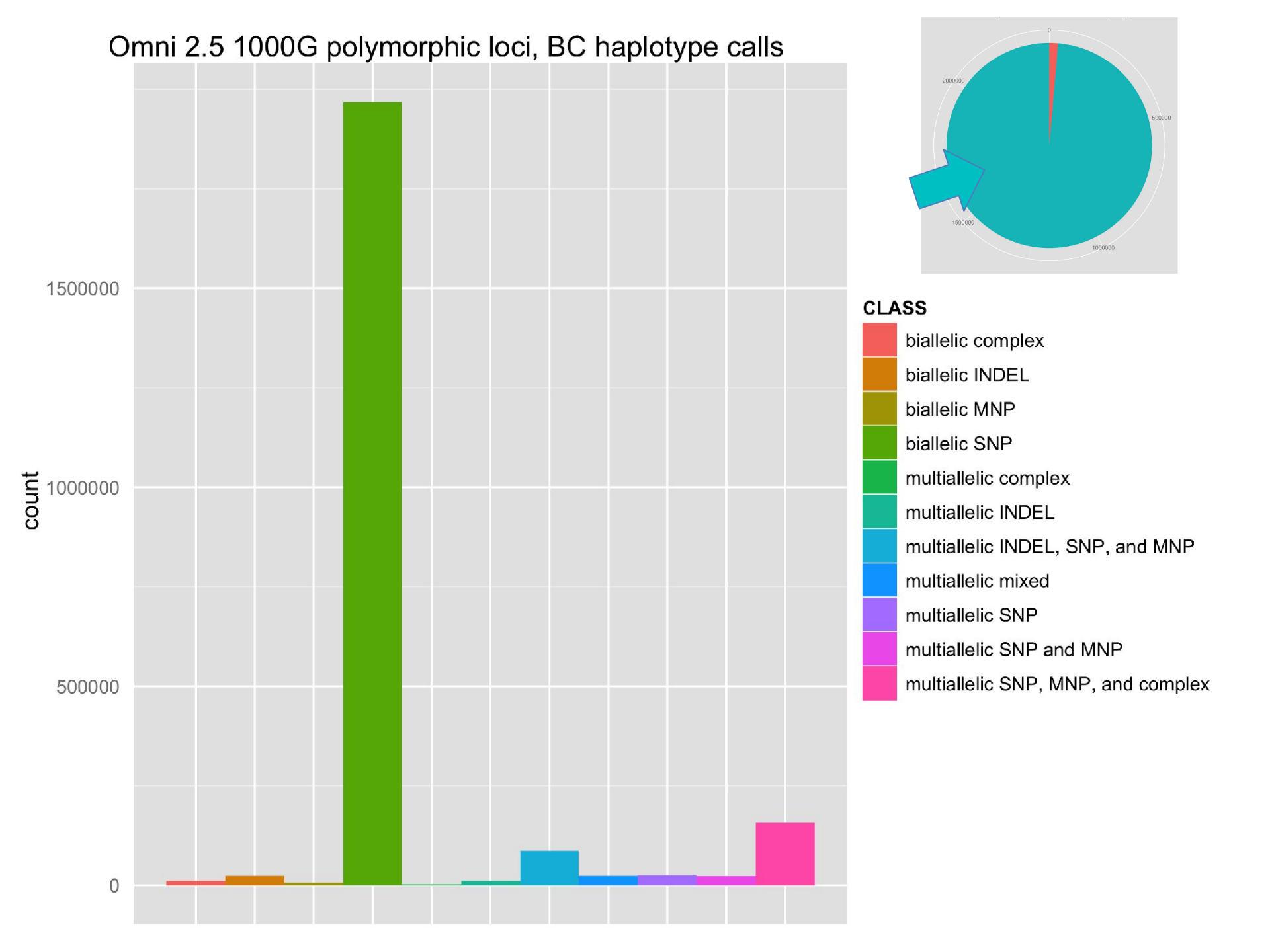
Impact on genotyping chip design

- Biallelic SNPs detected during the 1000 Genomes Pilot project were used to design a genotyping microarray (Omni 2.5).

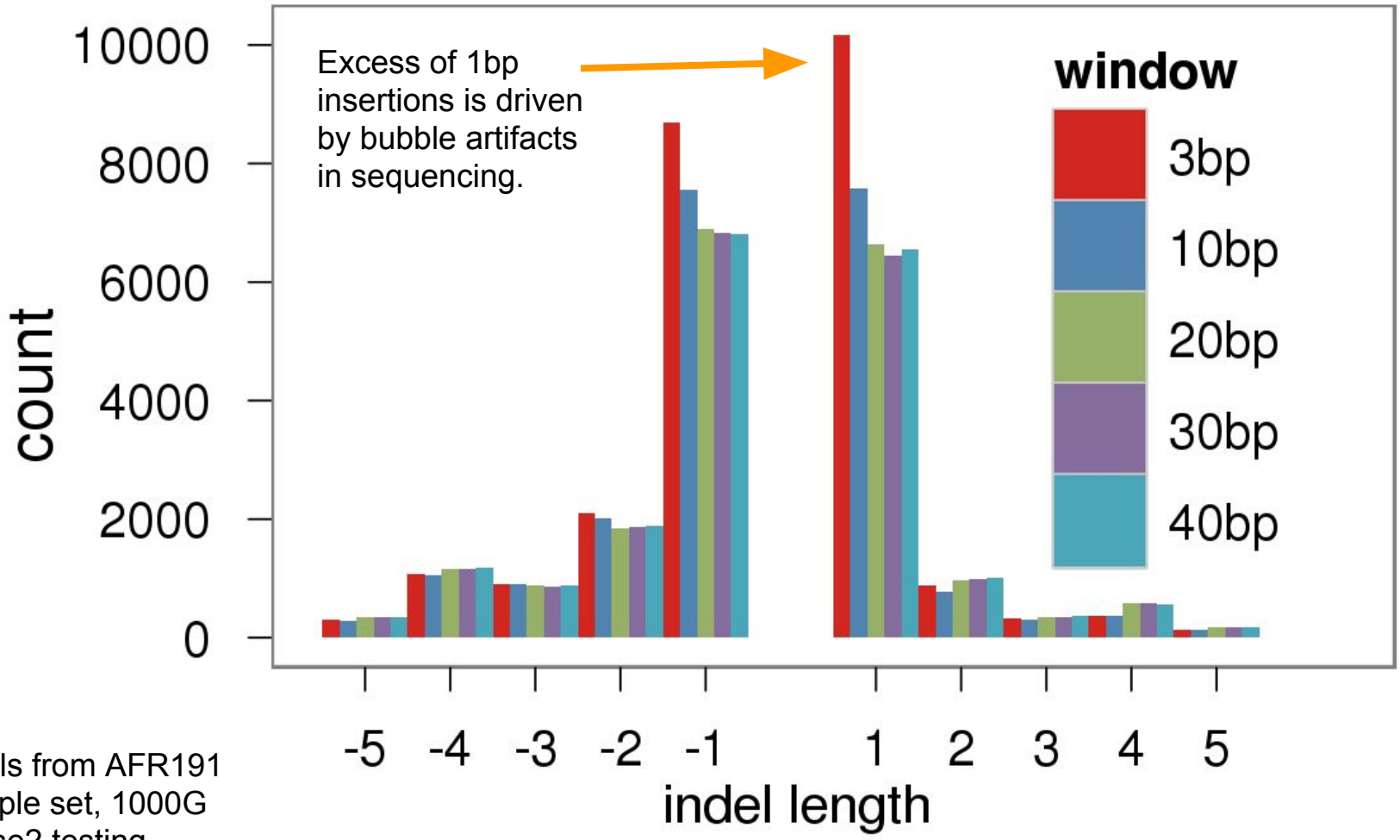
- When the 1000 Genomes samples were genotyped using the chip, 100k of the 2.5 million loci showed no polymorphism (monomorphs).




Omni 2.5 1000G monomorphic loci, BC haplotype calls
count
10000
8000
6000
4000
2000
0
CLASS
biallelic complex
biallelic INDEL
biallelic MNP
biallelic SNP
multiallelic complex
multiallelic INDEL
multiallelic INDEL, SNP, and MNP
multiallelic mixed
multiallelic SNP
multiallelic SNP and MNP
multiallelic SNP, MNP, and complex

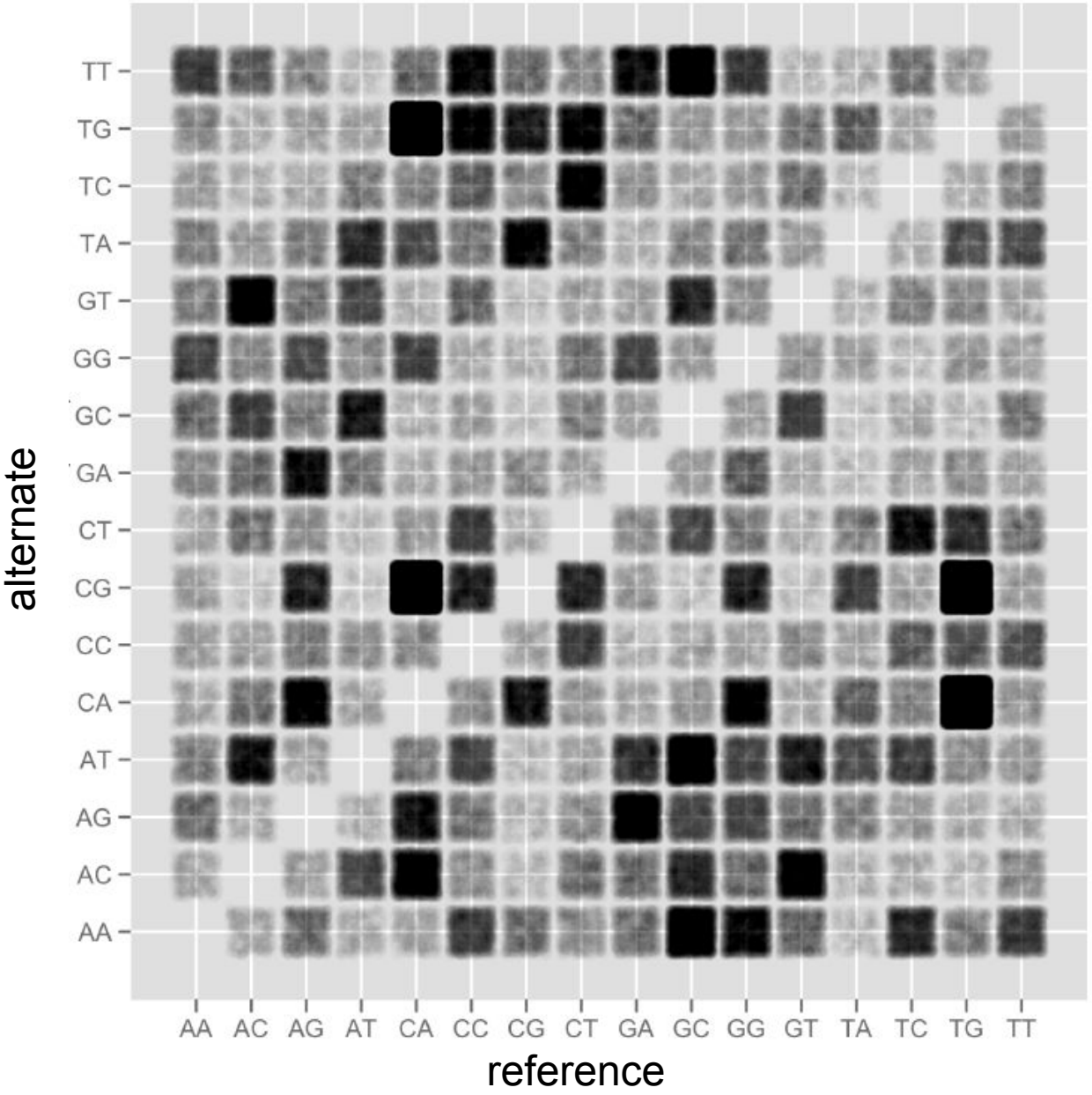

Omni 2.5 1000G polymorphic loci, BC haplotype calls
count
1500000
1000000
500000
0
CLASS
biallelic complex
biallelic INDEL
biallelic MNP
biallelic SNP
multiallelic complex
multiallelic INDEL
multiallelic INDEL, SNP, and MNP
multiallelic mixed
multiallelic SNP
multiallelic SNP and MNP
multiallelic SNP, MNP, and complex

Measuring haplotypes improves specificity

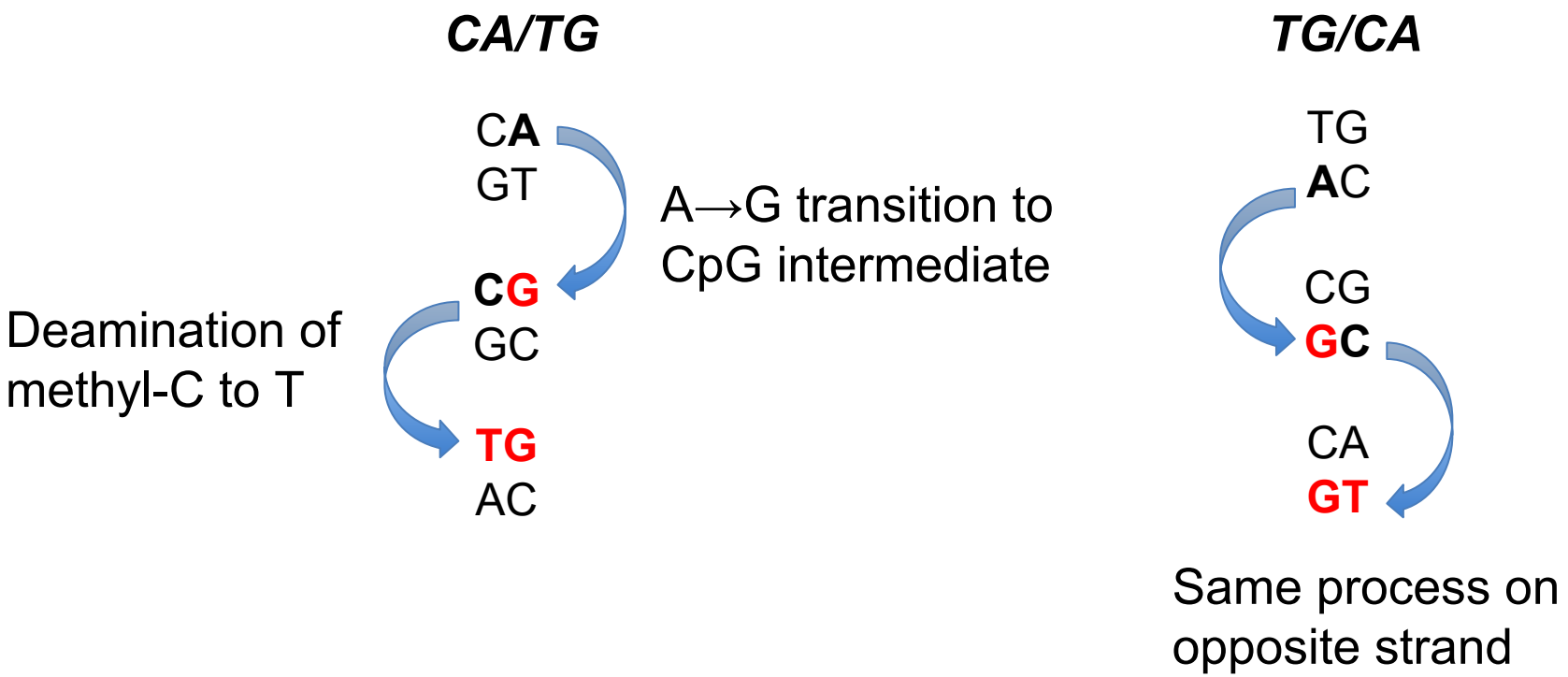


Indels from AFR191 sample set, 1000G phase2 testing.

2bp MNPs and dinucleotide intermediates



Direct detection of haplotypes can remove directional bias associated with alignment-based detection



Overview

1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. Sequence alignment
4. Alignment-based variant detection
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. **Primary filtering: Bayesian callers**
8. Post-call filtering: Hard filters, SVM
9. Genome variation graphs

Filtering Variation

Sequencing error rates are high.

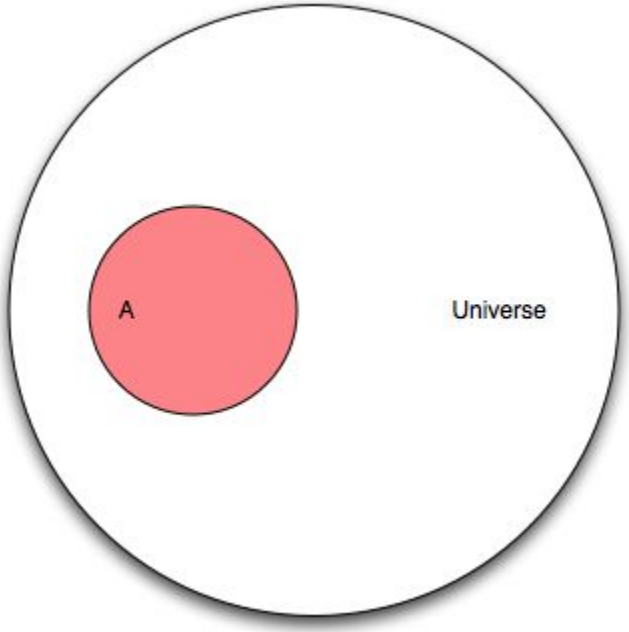
So, we need to filter.

The standard filter of NGS is the Bayesian variant caller.

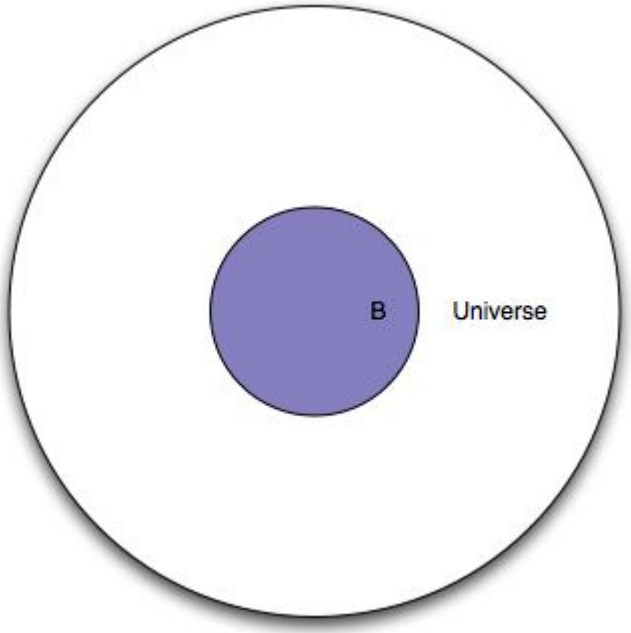
Combines population-based priors and data from many samples to make high-quality calls.

Bayesian (visual) intuition

We have a universe of individuals.



A = samples with a variant at some locus



B = putative observations of variant at some locus

Figures from http://oscarbonilla.com/2009/05/visualizing-bayes-theorem/

probability(A|B)

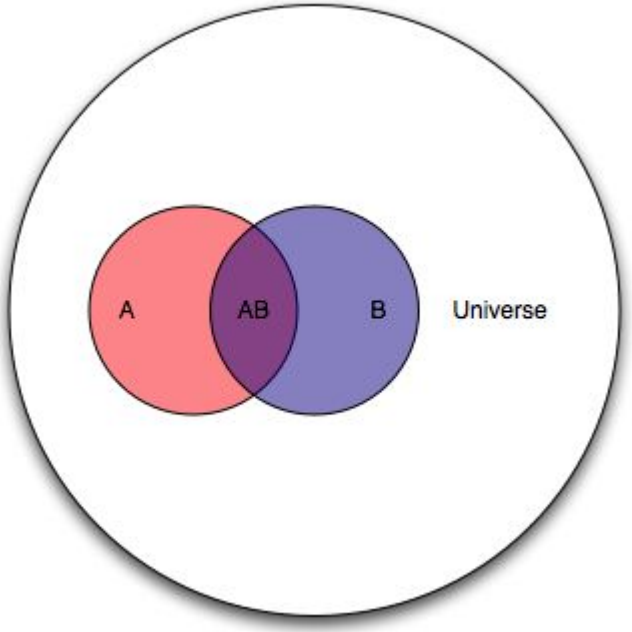
We want to estimate the probability that we have a real polymorphism "A" given "|" that we observed variants in our alignments "B".

$$P(A|B) = \frac{|AB|}{|B|}$$

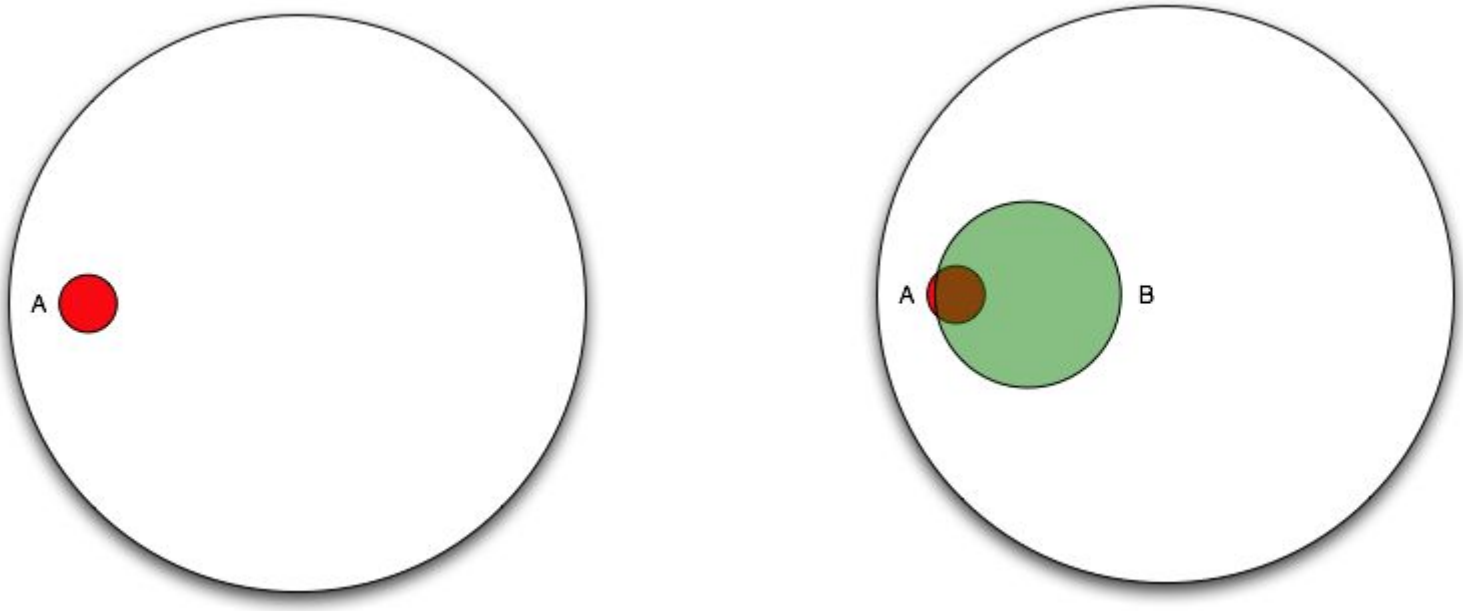
$$P(A|B) = \frac{P(AB)}{P(B)}$$

$$P(B|A) = \frac{P(AB)}{P(A)}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$



In our case it's a bit more like this...



Observations (B) provide pretty good sensitivity, but poor specificity.

The model

- Bayesian model estimates the probability of polymorphism at a locus given input data and the population mutation rate (~pairwise heterozygosity) and assumption of “neutrality” (random mating).
- Following Bayes theorem, the probability of a specific set of genotypes over some number of samples is:
  - **P(G|R) = (P(R|G) P(G)) / P(R)**
- Which in FreeBayes we extend to:
  - **P(G,S|R) = (P(R|G,S) P(G)P(S)) / P(R)**
  - **G** = genotypes, **R** = reads, **S** = locus is well-characterized/mapped
  - **P(R|G,S)** is our data likelihood, **P(G)** is our prior estimate of the genotypes, **P(S)** is our prior estimate of the mappability of the locus, **P(R)** is a normalizer.

Handling non-biallelic/diploid cases

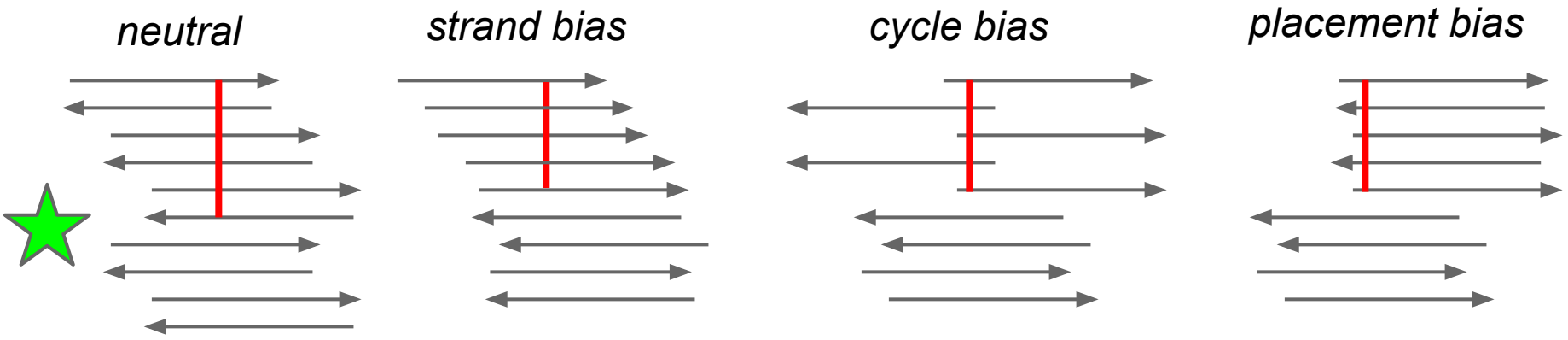
We compose our data likelihoods, **P(Reads|Genotype)** using a discrete multinomial sampling probability:

$$P(reads|genoytpe) = \binom{|reads|}{|reads = A|, |reads = B| \ldots} \times \prod_{\forall alleles \in genotype} freq(allele \in genotype) \times \prod_{\forall reads} P(correct(read))$$

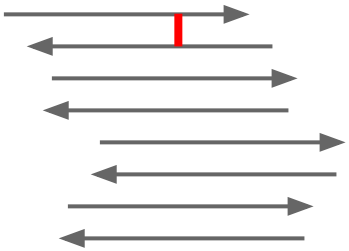
Our priors, **P(Genoypes)**, follow the Ewens Sampling Formula and the discrete sampling probability for genotypes.

Are our locus and alleles sequenceable?

In WGS, biases in the way we observe an allele (placement, position, strand, cycle, or balance in heterozygotes) are often correlated with error. We include this in our posterior **P(G,S|R)**, and to do so we need an estimator of **P(S)**.

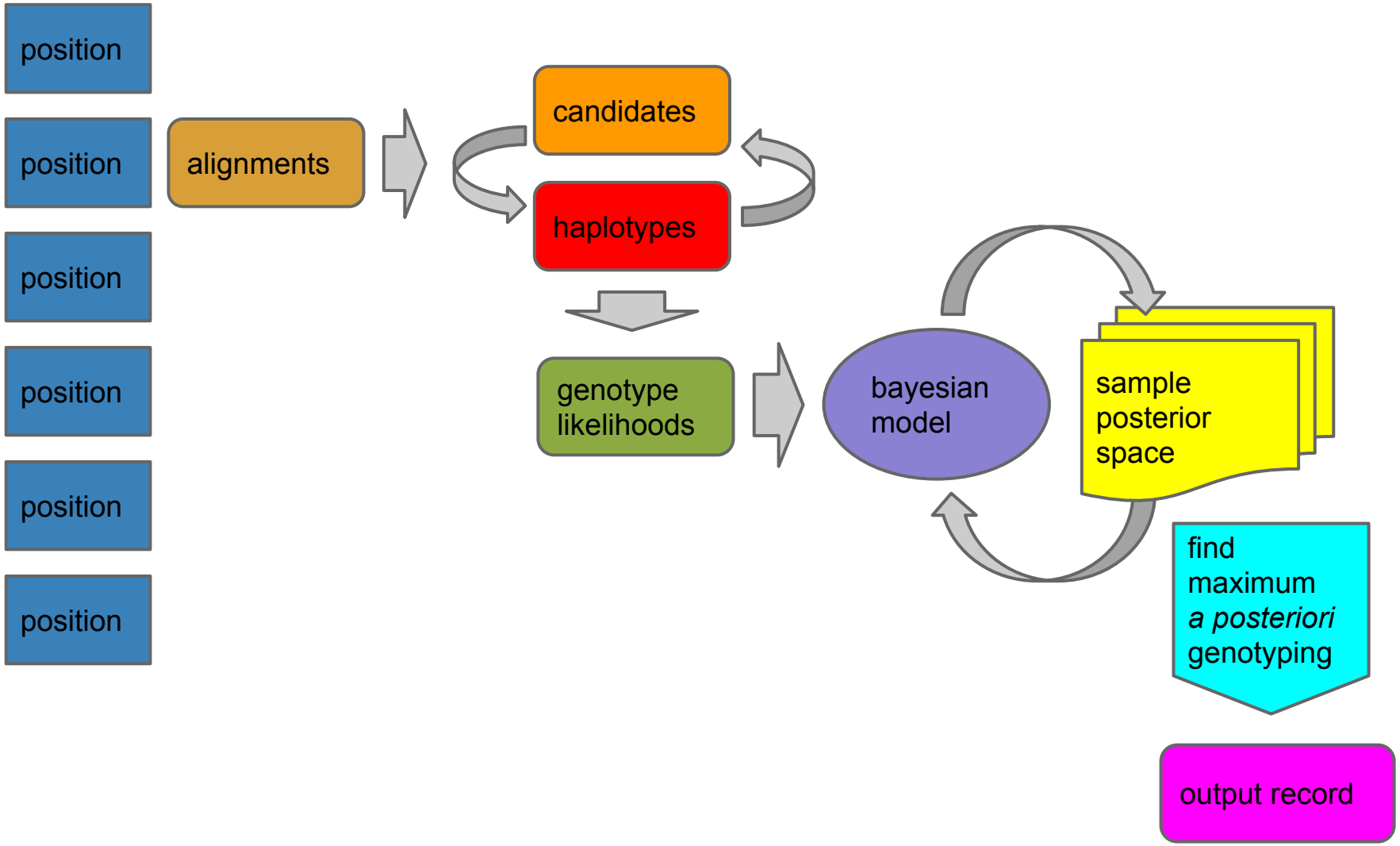


allele imbalance



$$P(S) \propto \times \prod_{\forall b \in \{B\}} multinom([|R \equiv b|\forall b_1, \ldots, b_K]; |\{R\}|, f_i, \ldots, f_K)$$
$$binom(|forwardStrand(\{R \equiv b\})|; |\{R \equiv b\}|, 1/2)$$
$$\times binom(|placedLeft(\{R \equiv b\})|; |\{R \equiv b\}|, 1/2)$$
$$\times binom(|placedRight(\{R \equiv b\})|; |\{R \equiv b\}|, 1/2)$$

The detection process



Variant detector lineage

PolyBayes– original Bayesian variant detector (Gabor Marth, 1999); written in perl

GigaBayes– ported to C++

BamBayes– “modern” formats (BAM)

FreeBayes– 2010-present

FreeBayes-specific developments

FreeBayes model features (~in order of introduction):

- Multiple alleles
- Indels, SNPs, MNPs, complex alleles
- Local copy number variation (e.g. sex chromosomes)
- Global copy-number variation (e.g. species-level, genome ploidy)
- Pooled detection, both discrete and continuous
- Many, many samples (>30k exome-depth samples)
- Genotyping using known alleles (hints, haplotypes, or alleles)
- Genotyping using a reference panel of genotype likelihoods
- Direct detection of haplotypes from short-read sequencing
- Haplotype-based consensus generation (clumping)
- Allele-length-specific mapping bias
- Contamination-aware genotype likelihoods

Design/development methodology

- Portable C++
- MIT license
- Modular codebase (multiple git submodules)
  - BAM library (bamtools)
  - VCF library (vcflib)
  - FASTA library (fastahack)
- Streams > rewriting files
- *A priori* models > technology-specific recalibration
- Probabilities > hard filters
- Simulations > ts/tv, bulk metrics, array concordance
- Annotations for classification and filtering of variants (vcfsom)
- Produce internally-consistent VCF output
- Run *fast*

Overview

1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. Sequence alignment
4. Alignment-based variant detection
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. **Post-call filtering: Hard filters, SVM**
9. Genome variation graphs

Variant filtering

Variant detection is hard.

A priori models can't capture all types of error.

We can use *hard filters* to cut out parts of the data.

We can use *classifiers* like Support Vector Machines (SVM) to further improve results.

Hard filters

What calls do we “know” are poor based on what we know about sequencing error?

(w.r.t. freebayes VCF annotations).

1. low-quality calls (QUAL < N)
    a. also, maybe QUAL is high but QUAL / AO is low
2. loci with low read depth (DP < N)
3. alleles that are only seen on one strand
    a. remove by “SAF > 0 & SAR > 0”
4. alleles that are only observed by reads placed to the left or right
    a. remove by “RPL > 0 & RPR > 0”

suggested freebayes hard filters

For diploid, humans…

pipe your VCF through something like:

```
vcffilter -f "QUAL > 1 & QUAL / AO > 10 & SAF > 0 & SAR > 0 & RPR > 1 & RPL > 1"
```

- QUAL > 1
  - removes horrible sites
- QUAL / AO > 10
  - additional contribution of each obs should be 10 log units (~ Q10 per read)
- SAF > 0 & SAR > 0
  - reads on both strands
- RPR > 1 & RPL > 1
  - at least two reads “balanced” to each side of the site

When hard filters aren't enough

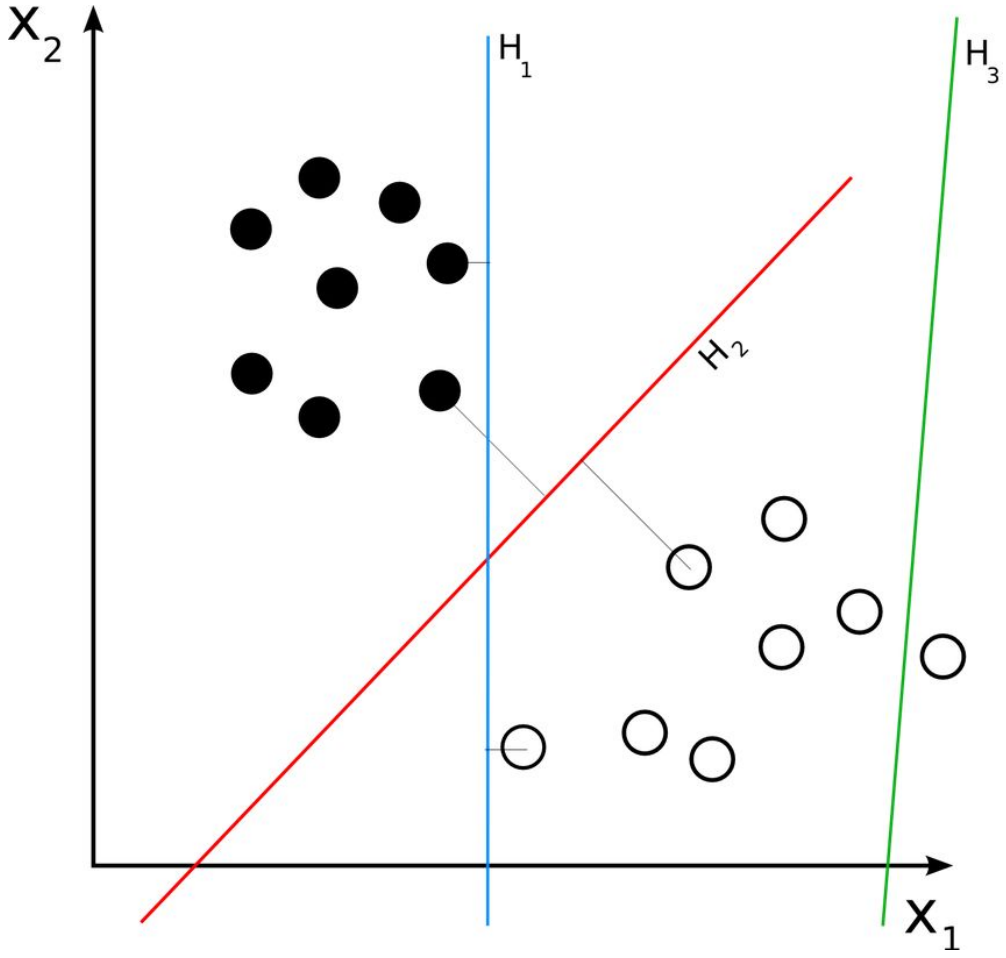
We can use validation data to improve our filtering beyond what's possible with *a priori* models.

Example:

Calling indels in the 1000 Genomes Phase 3.

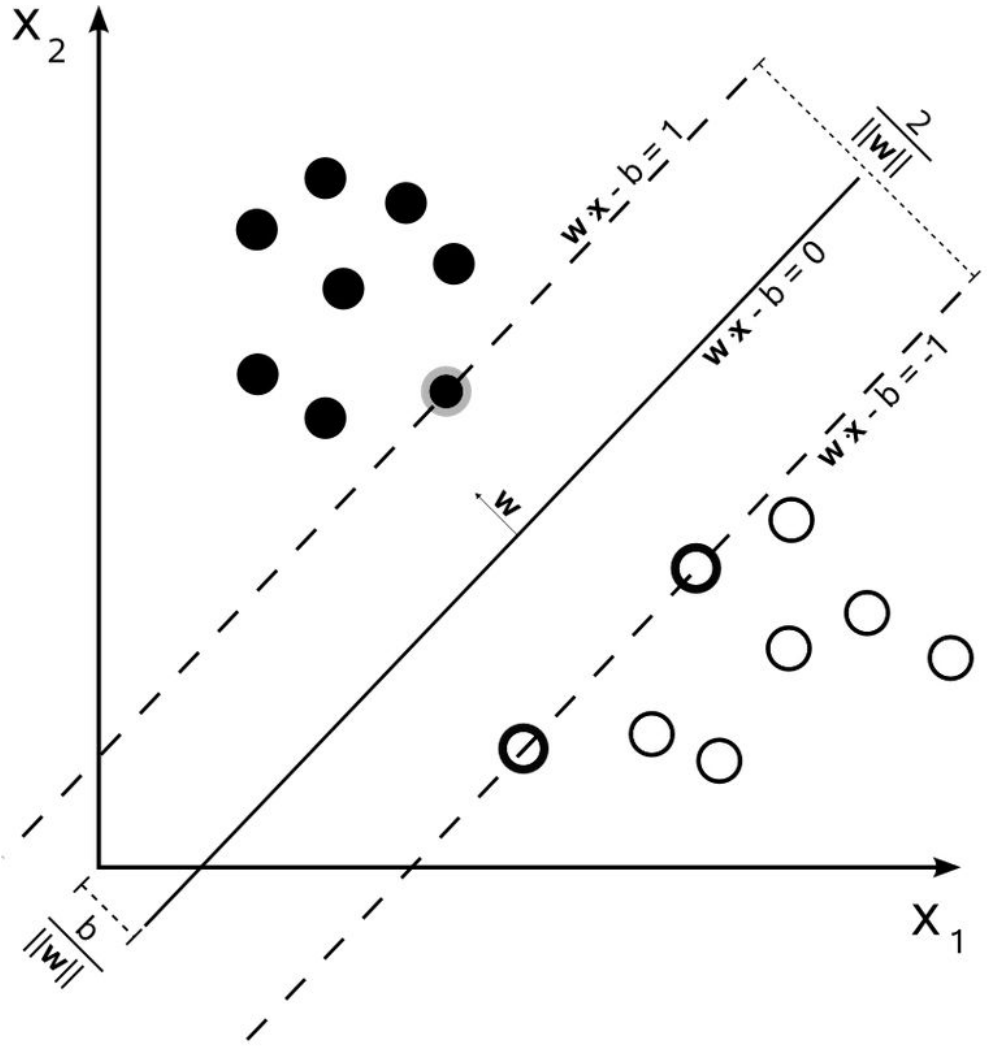
SVM classifier

Find a hyperplane (here a line in 2D) which separates observations.



SVM classifier

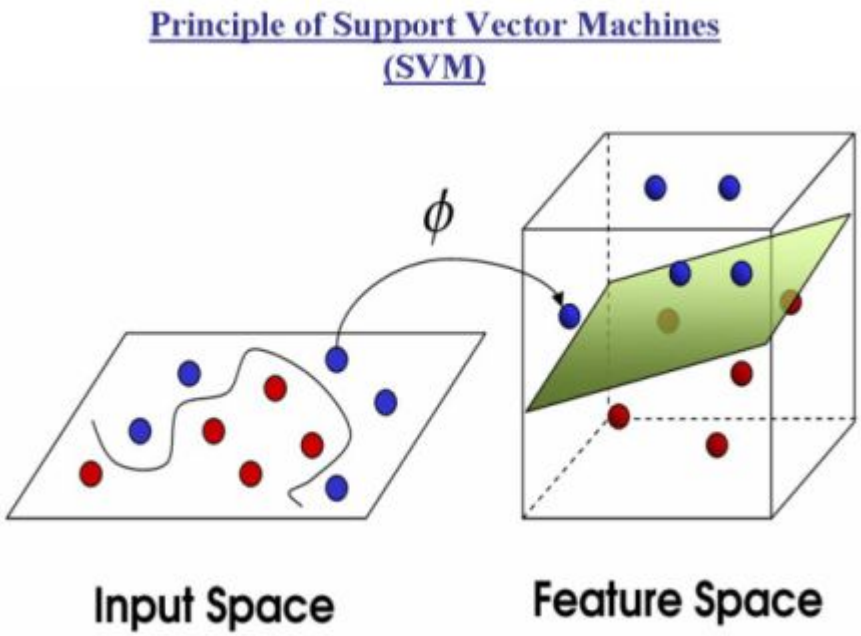
The best separating hyperplane is determined by maximum margin between groups we want to classify.



SVM classifier: but...

We can't always linearly separate the data points. So, in practice we apply transformations to "lift" the classes apart in a higher-dimensional space.

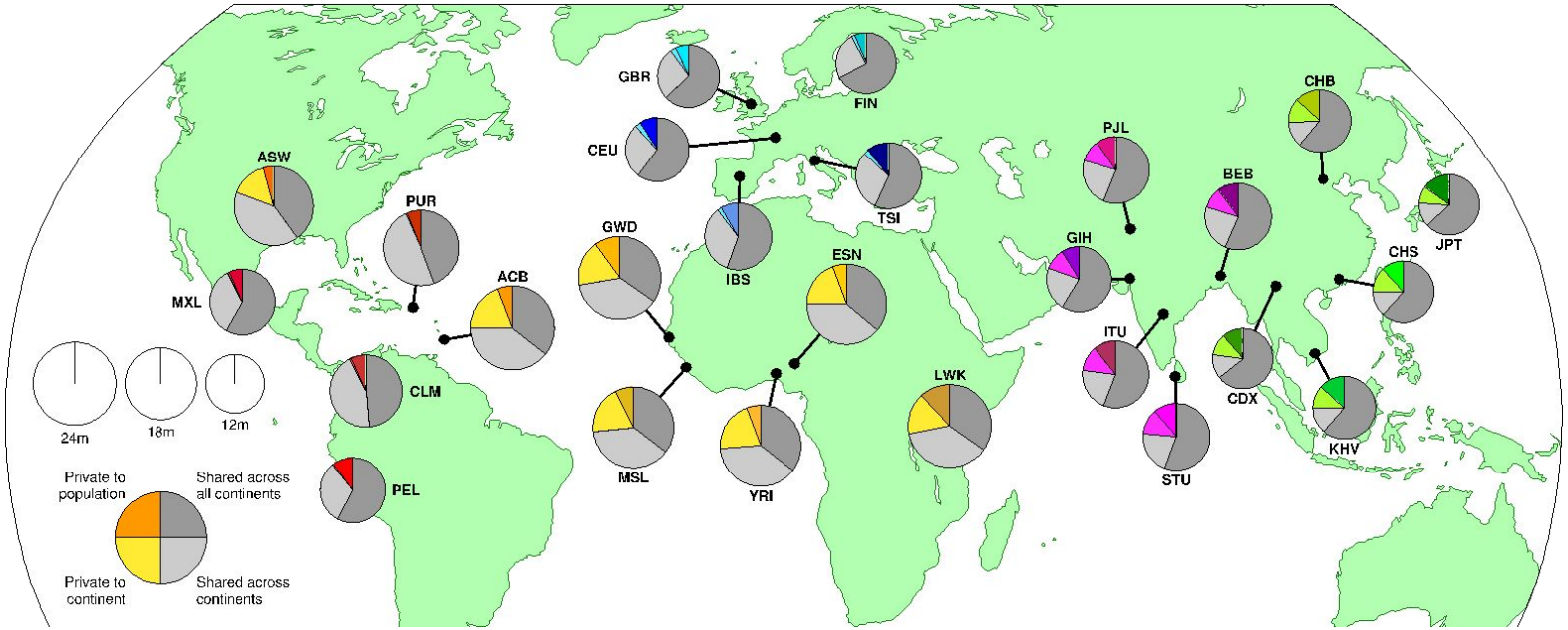
Apply a radial basis function (RBF) kernel ϕ. Two hyperparameters (c and gamma) control the bias and variance of the projection. We find them via grid search.



http://article.sapub.org/10.5923.j.computer.20110101.04.html

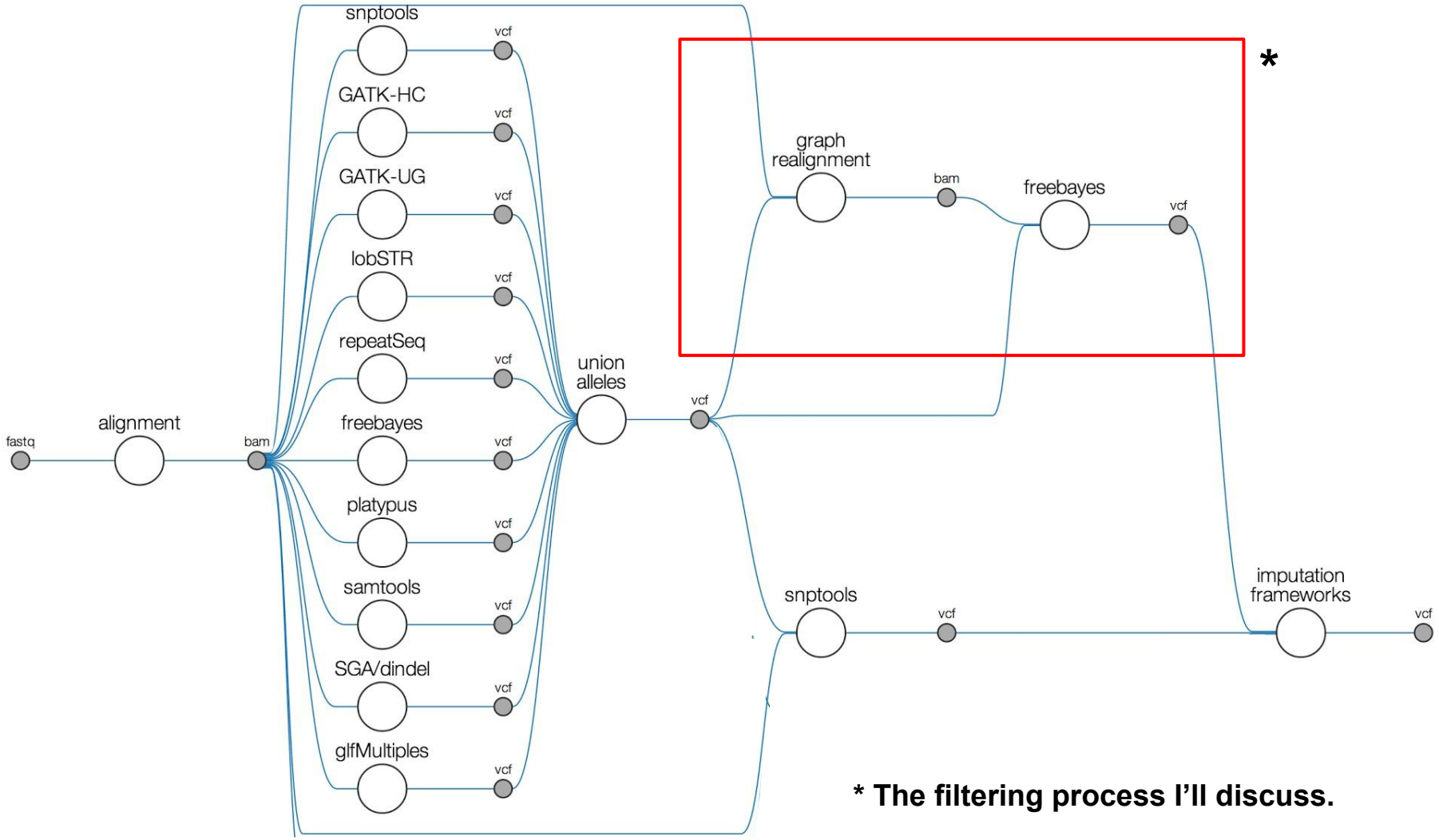
https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine

Indel filtering in the 1000 Genomes



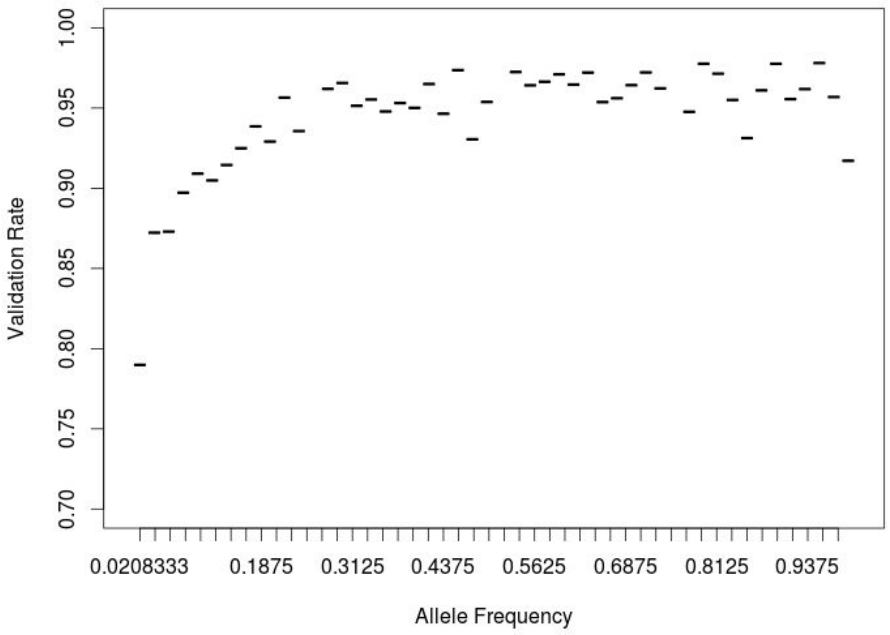
25 human populations X ~100 samples each.

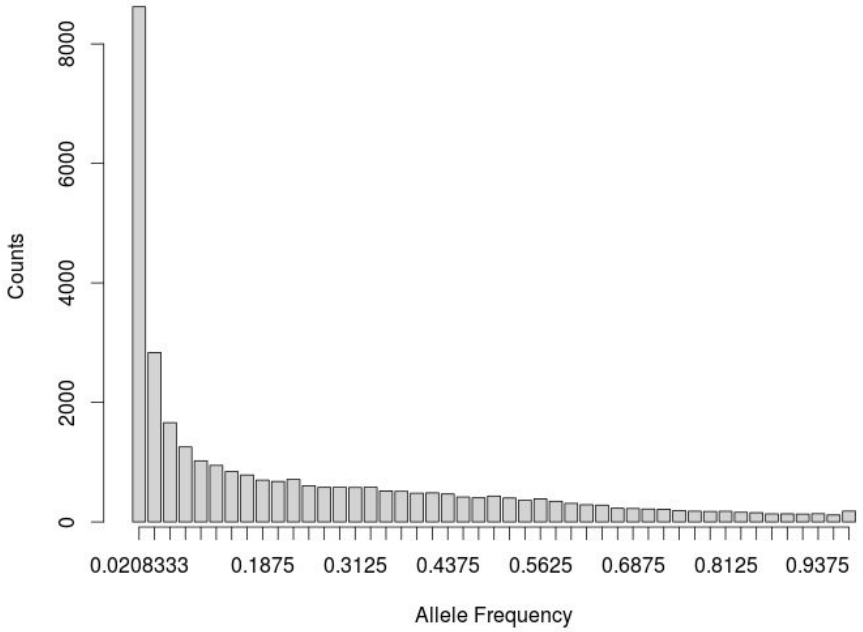
1000G data integration process



*** The filtering process I'll discuss.**

1000G INDELs had a high error rate





Raw validation rates of indels in 1000G phase 3, “MVNCall” set. Validation is provided using *k*-mer matching between results and PCR-free sequencing reads from a subset of the samples.

Anthony Marcketta and Adam Auton

SVM approach for INDEL filtering

Extract features that we expect to vary with respect to call quality:

- call QUALity
- read depth
- sum of base qualities
- inbreeding coefficient
- entropy of sequence at locus
- mapping quality
- allele frequency in population
- read pairing rate

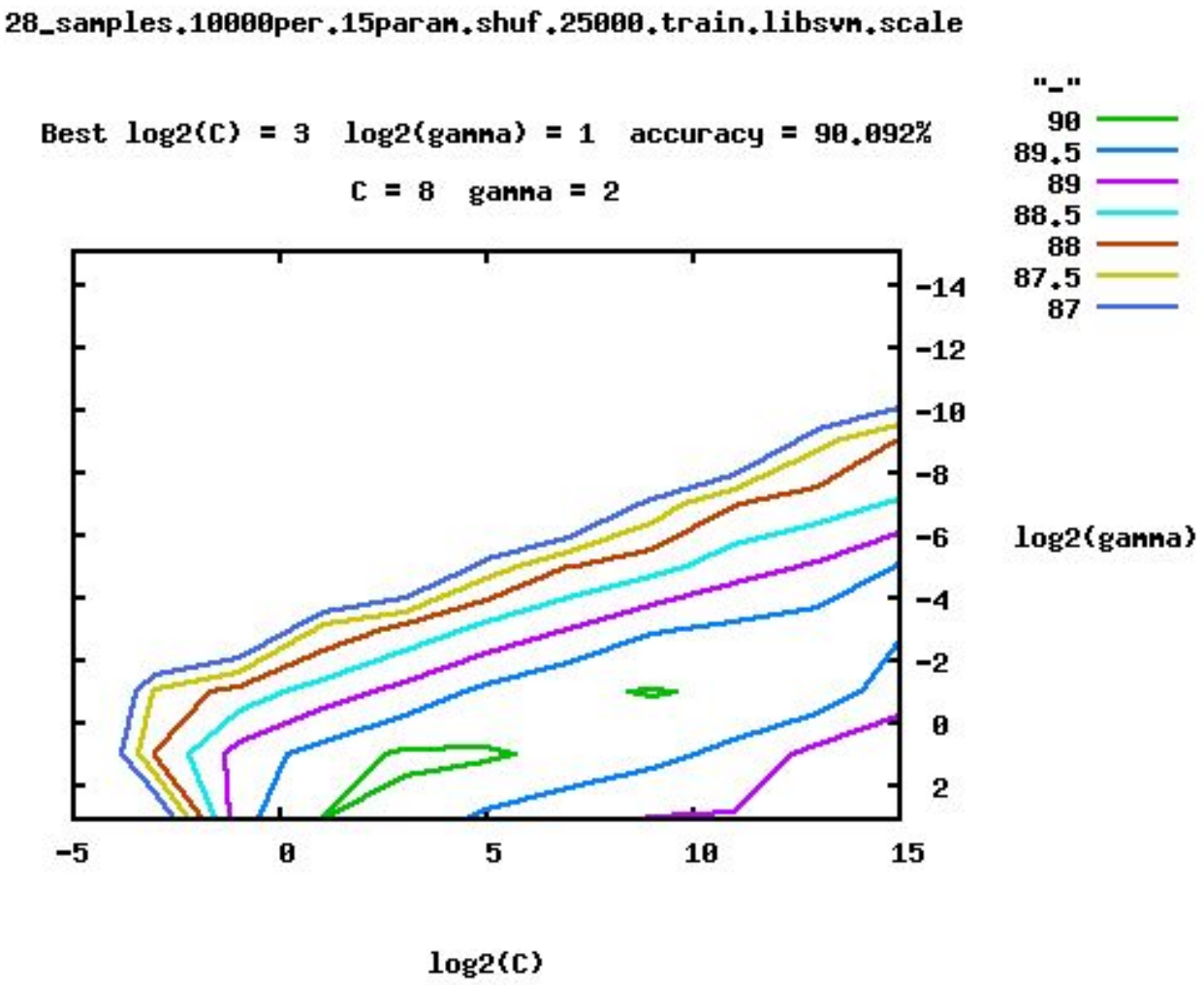
SVM approach for INDEL filtering

1. Now, use overlaps in validation samples or sites to determine likely errors and true calls.
2. Use this list + annotations of the calls to train an SVM model using a small subset of the data from our validation samples.
3. Then apply the model to all the calls, filter, and measure validation rate of the whole set.
4. Use a probabilistic estimate of class to allow post-hoc decision making about filtering.

Model training: hyperparameters

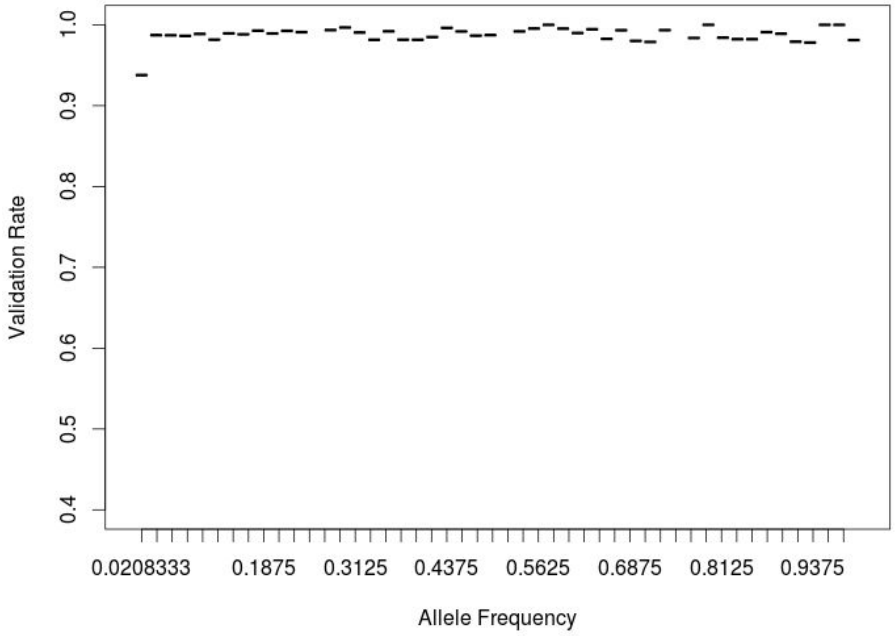
A visual hint at the process of training an SVM.

This plot shows the results of a grid search over our model hyperparameters gamma and C.

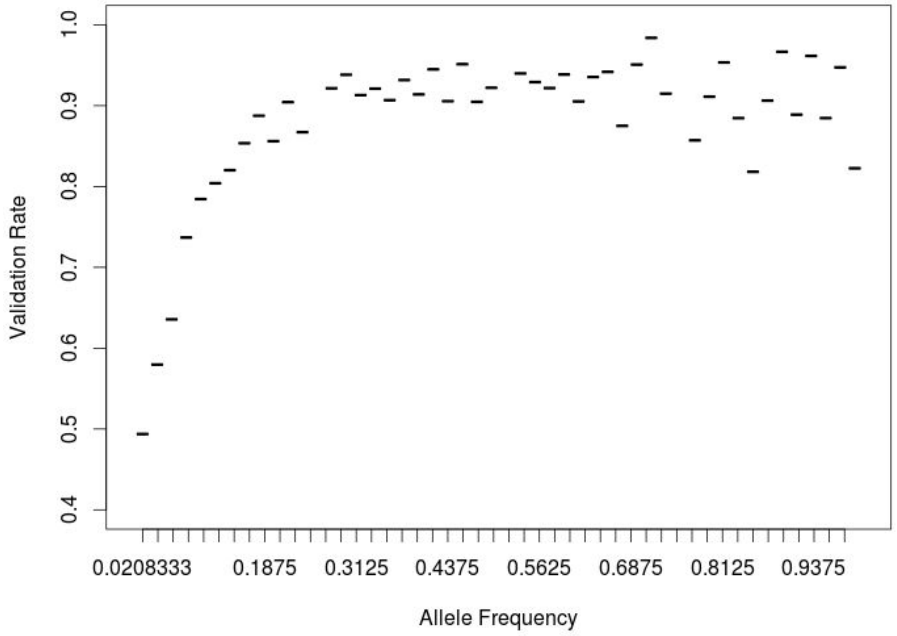


Application of SVM to 1000G INDELs

Passing SVM



Failing SVM

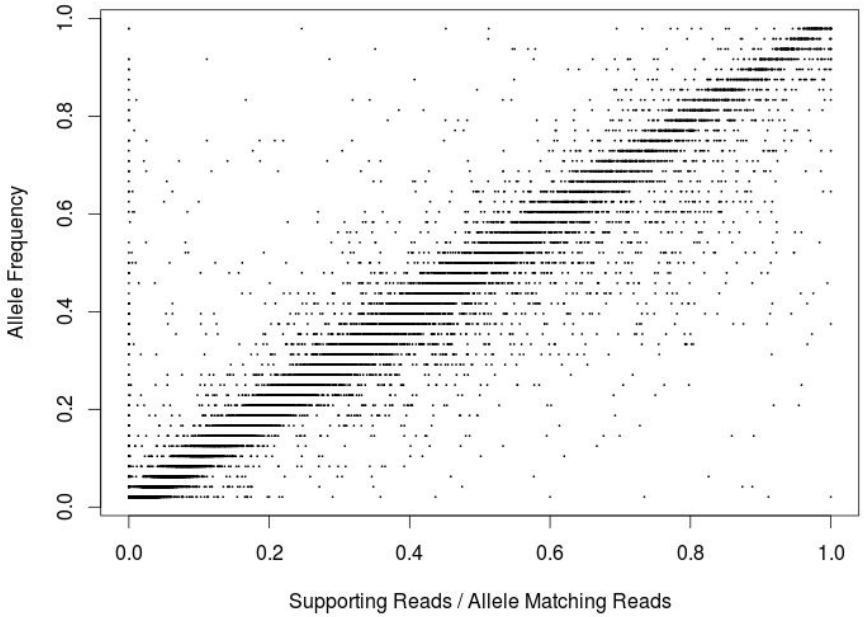


Filtering results, using SVM-based method.

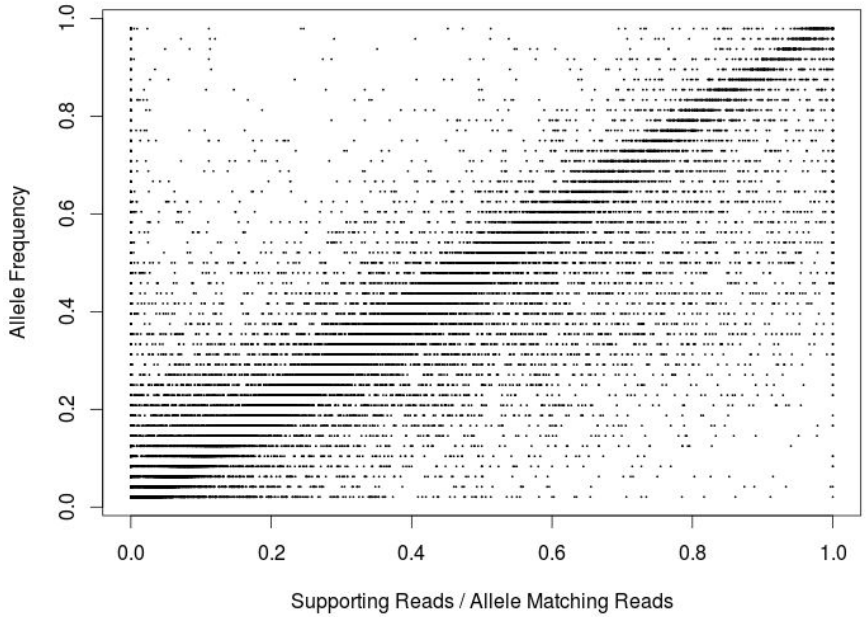
Anthony Marcketta and Adam Auton

Application of SVM to 1000G INDELs

Passing SVM



Failing SVM



Correlation between allele frequency and observation counts.

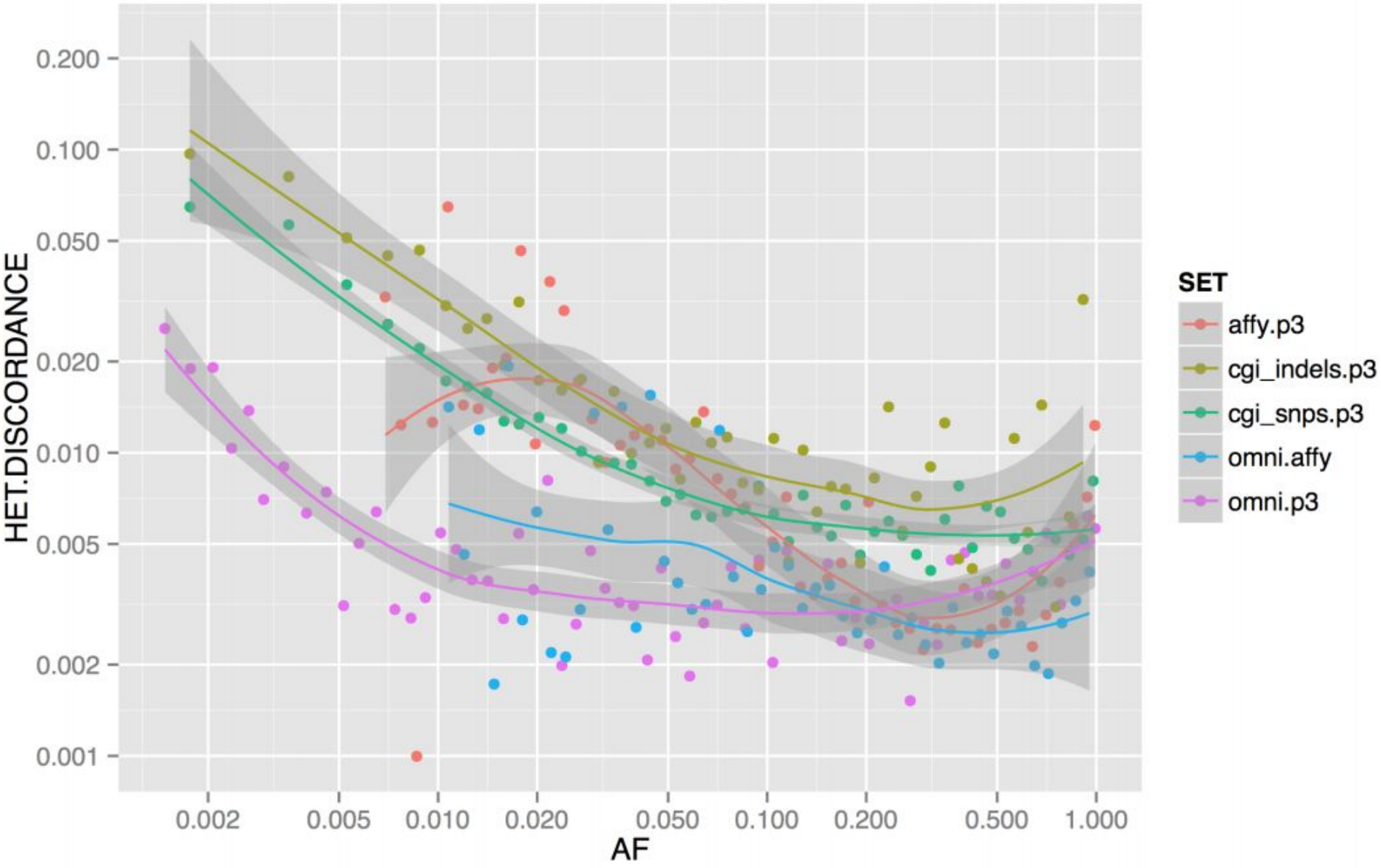
Anthony Marcketta and Adam Auton

Indel results from 1000G

Gold	Eval	R/R	R/A	A/A	All	NonRef
CGI SNPs	Phase3	0.9998	0.9930	0.9983	0.9994	0.9920
CGI Indels	Phase3	0.9990	0.9889	0.9923	0.9982	0.9805

Comparing the phase3 results to the genotypes for indels in the subset of samples for which we also had high-quality, high-coverage genomes from Complete Genomics.

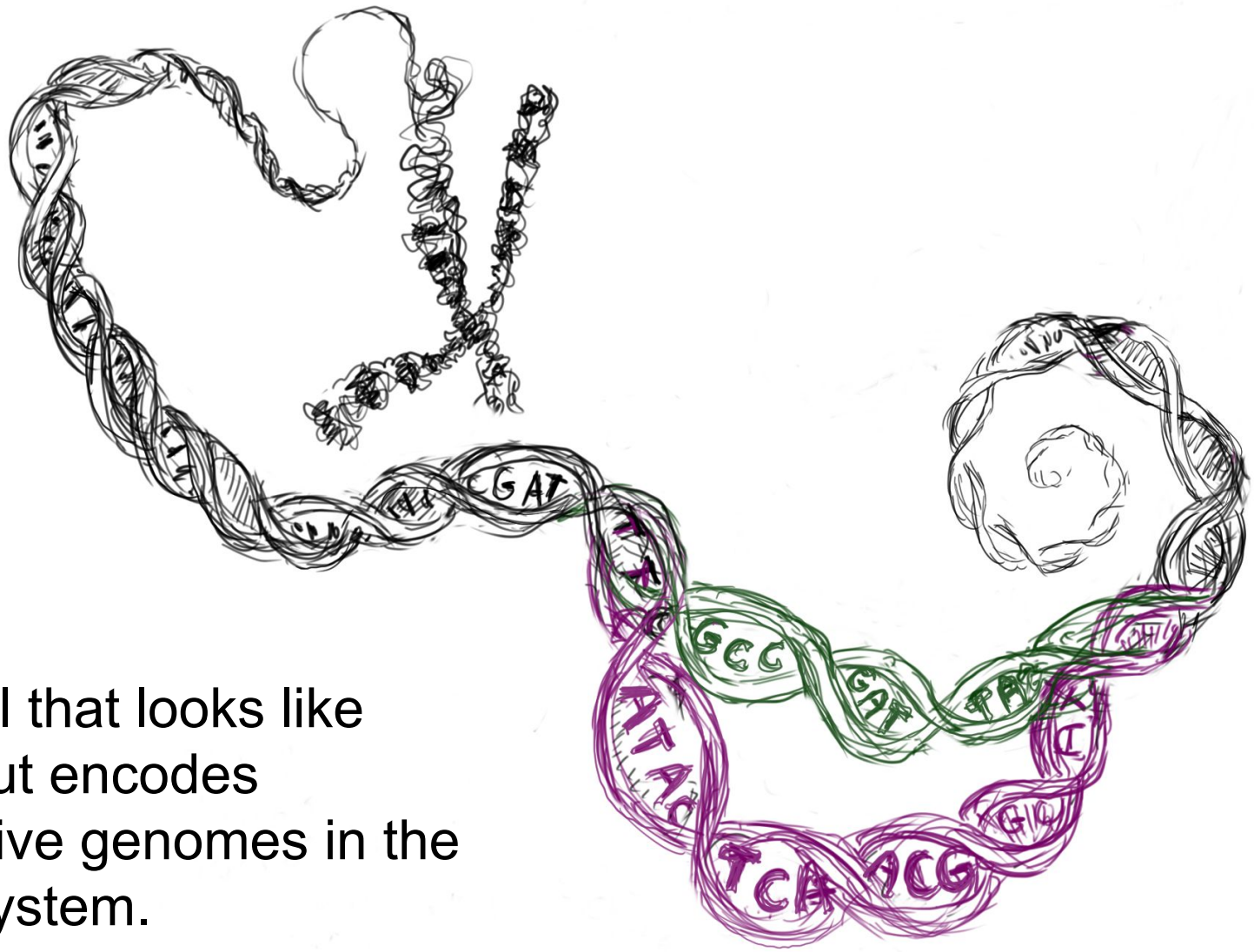
Genotype Accuracy by Allele Frequency



Overview

1. Genesis of variation (SNPs and indels)
2. Causes of sequencing error
3. Sequence alignment
4. Alignment-based variant detection
5. Assembly-based variant detection
6. Haplotype-based variant detection
7. Primary filtering: Bayesian callers
8. Post-call filtering: Hard filters, SVM
9. **Genome variation graphs**

Genome variation graphs

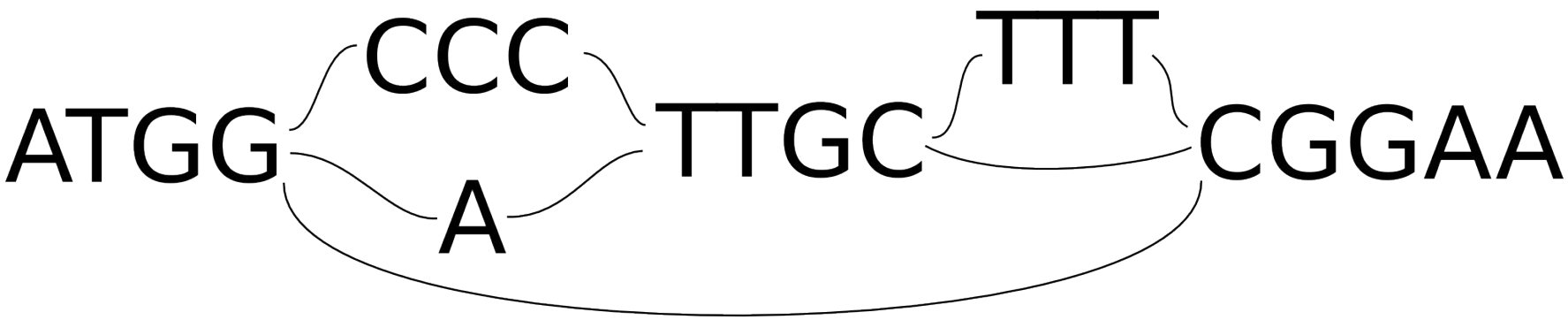


A model that looks like DNA, but encodes alternative genomes in the same system.

Maciej Smuga-Otto http://www.smuga-otto.com/mso/

Genome variation graphs

A variation graph represents many genomes in the same context.



Nodes contain sequence and directed edges represent potential links between successive sequences.

Multiple sequence alignments ~ variation graphs

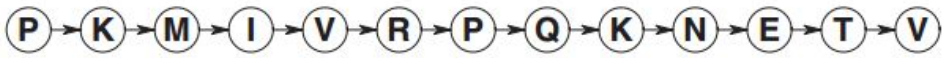
traditional MSA

(a)

. . P K M I V R P Q K N E T V .
T H . K M L V R . . . N E T I M

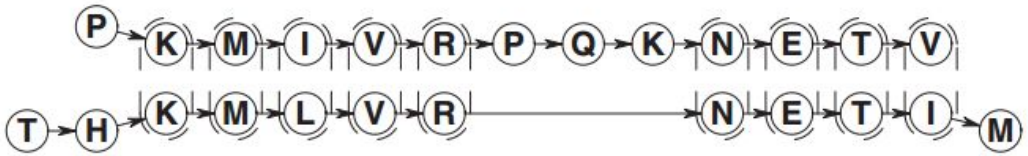
consensus sequence

(b)



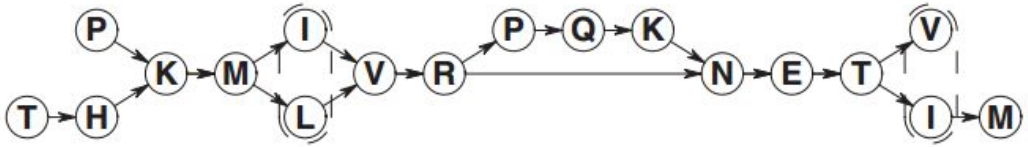
positionally-matching regions aligned

(c)



multiple sequence alignment

(d)



Christopher Lee, Catherine Grasso, Mark F. Sharlow. **Multiple sequence alignment using partial order graphs.** *Bioinformatics*, 2002.

Assembly graphs ~ variation graphs

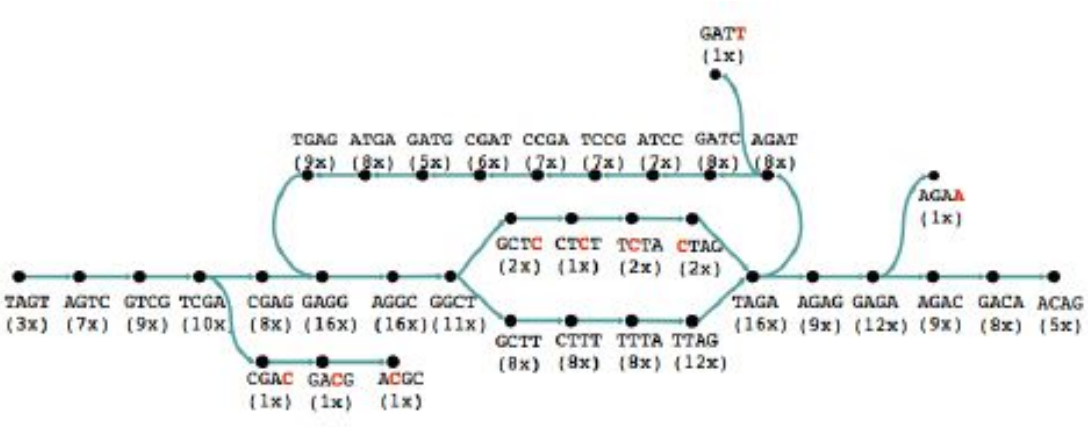
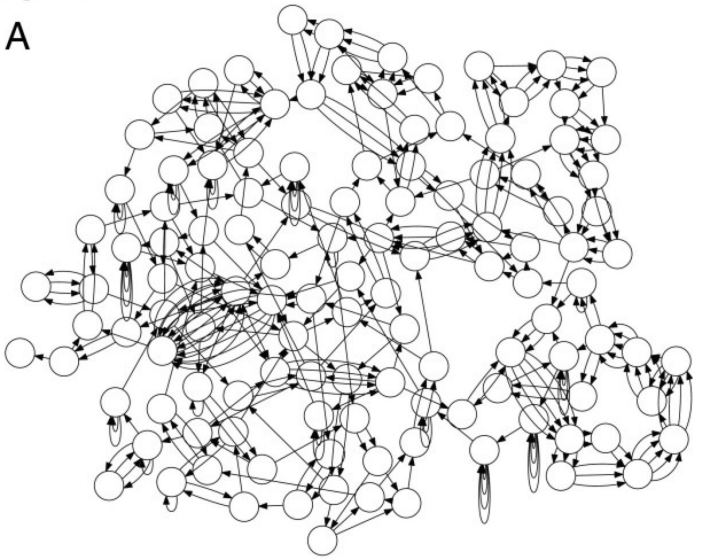
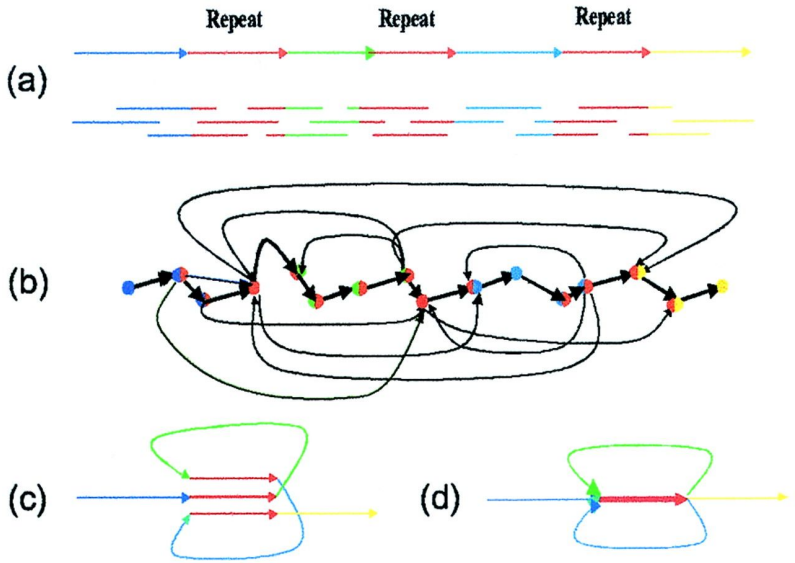


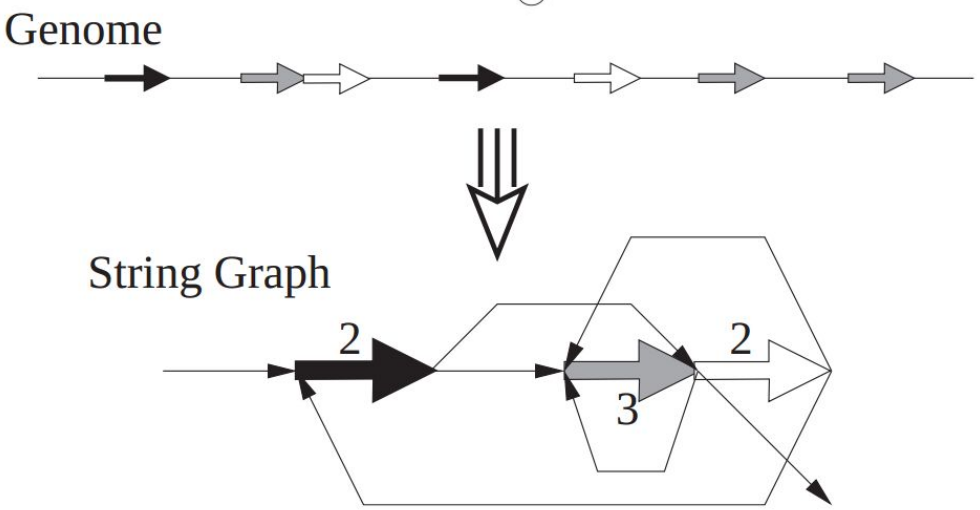
Figure 1.

A



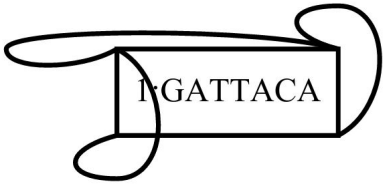
http://plus.maths.org/content/os/issue55/features/sequencing/index, credit Daniel Zerbino





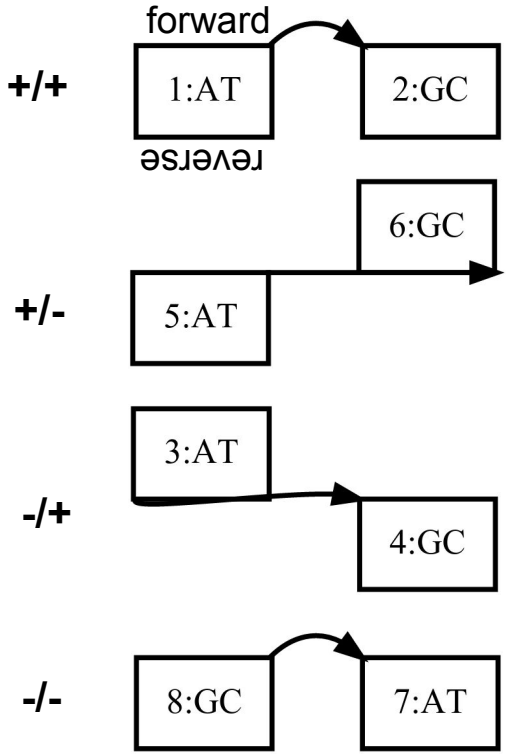
Eugene Myers. **The fragment assembly string graph**. *Bioinformatics*, 2005.

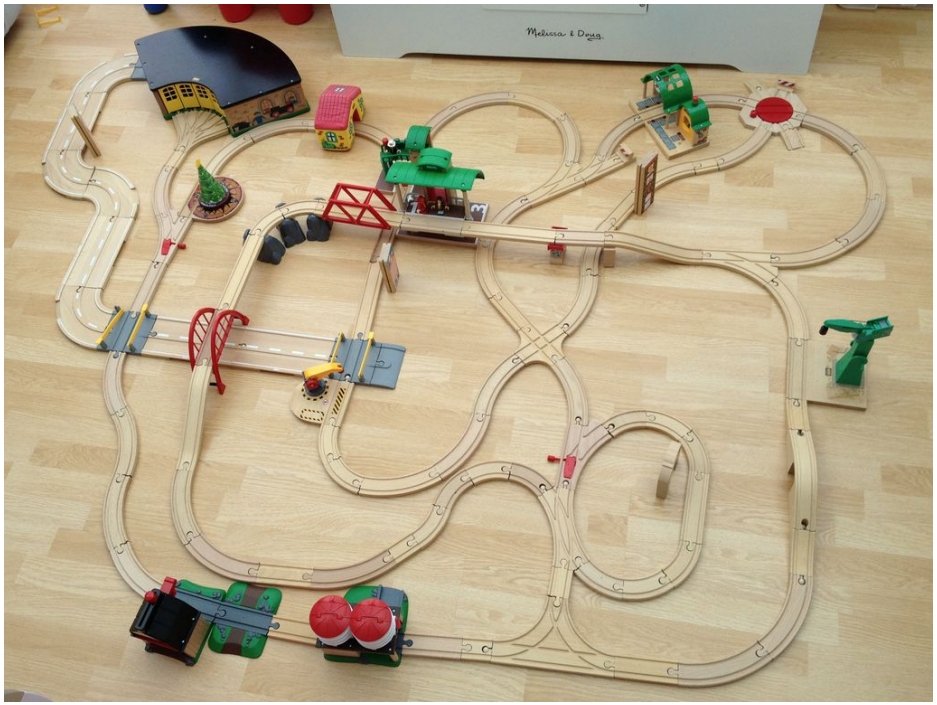
Train track graphs

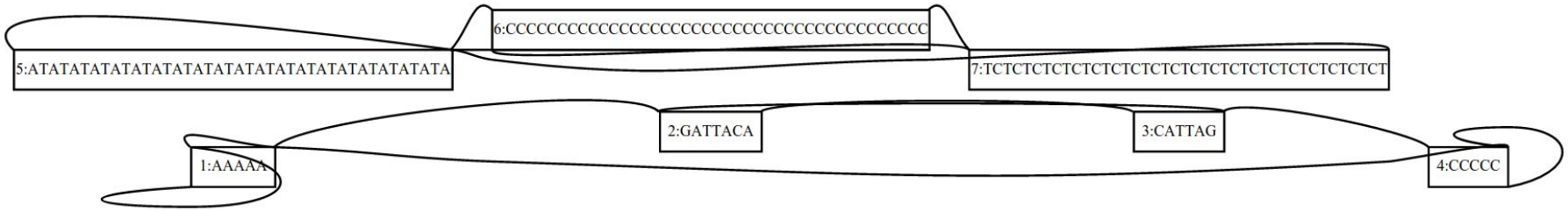


The graph is implicitly bidirectional, encoding both the forward and reverse complement.

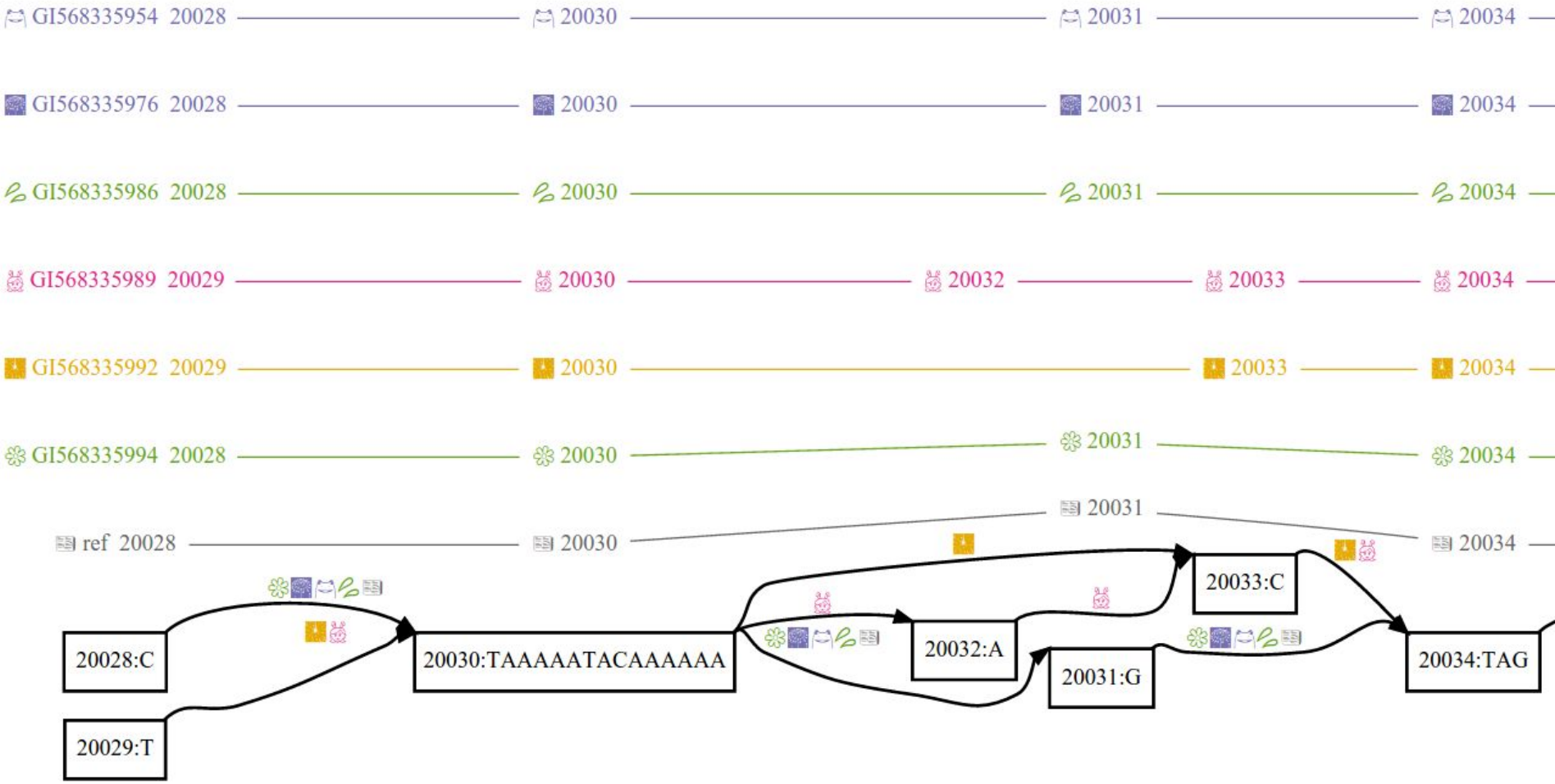
Edges switching from the forward (+) to reverse (-) represent inversions.







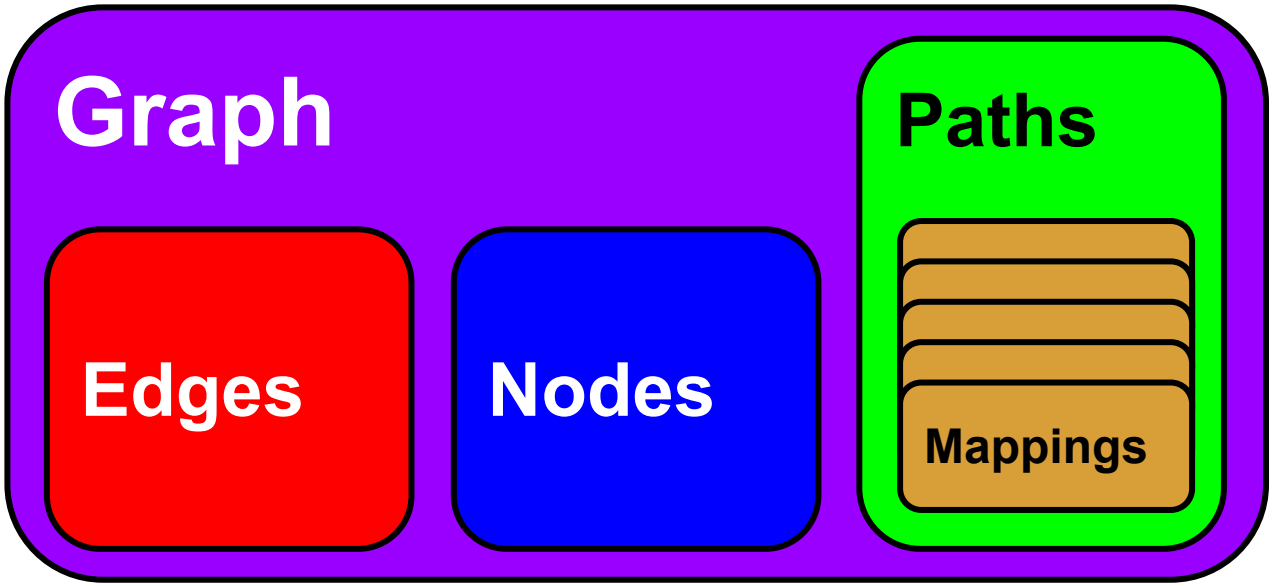
Essential components of a VG



*a fragment of the MHC

Model

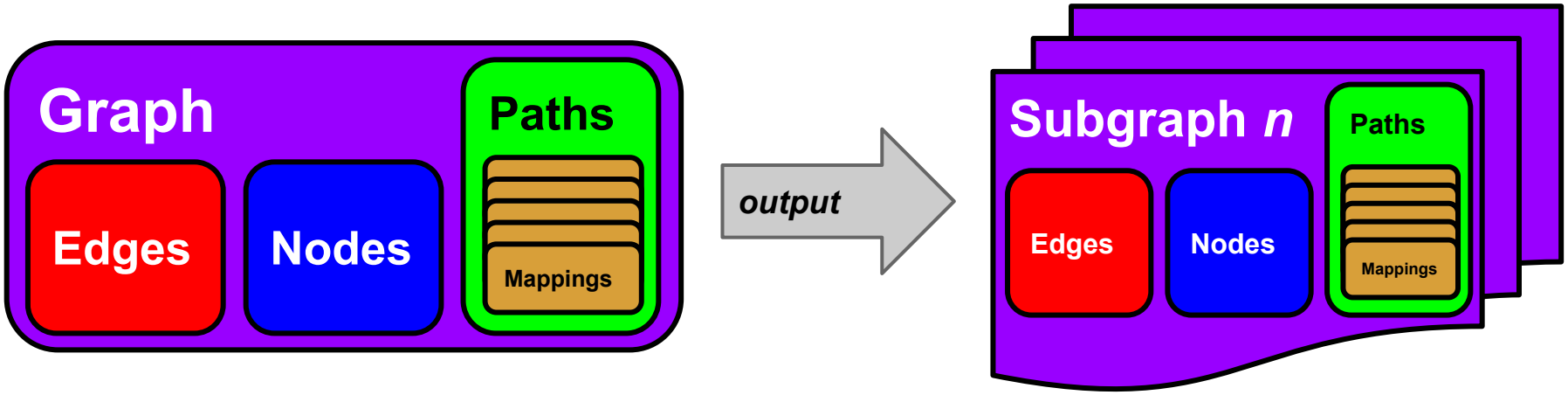
Basic entity is a *Graph*:



Implemented in protobuf / JSON / RDF.

Serialization

To serialize the graph, we generate a stream of sub-graphs that can be reassembled into the whole.



in memory **serialized stream**

How to VG

Make a graph: ***vg construct /msga /view***

Index it: ***vg index***

Query it: ***vg find***

Sample it: ***vg sim***

Map to it: ***vg map***

Call variants: ***vg call***

POS ID REF ALT
...

Construction

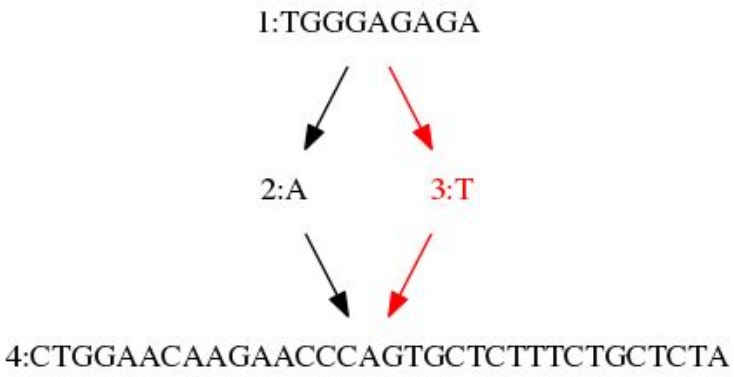
1:TGGGAGAGAACTGGAACAAGAACCCAGTGCTCTTTCTGCTCTA

For each variant

1. cut the reference path around the variant
2. add the novel (ALT) sequence to the graph

Construction

```
POS     ID      REF     ALT
10      .       A       T
...
```



For each variant

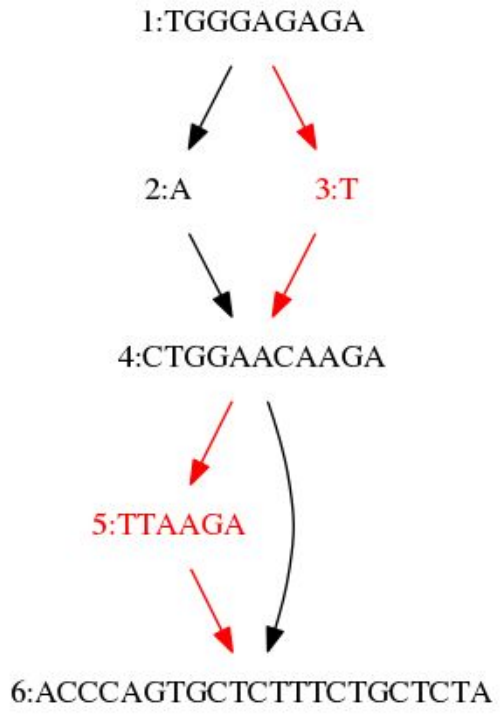
1. cut the reference path around the variant
2. add the novel (ALT) sequence to the graph

Construction

POS	ID	REF	ALT
10	.	A	T
21	.	A	ATTAAGA
...			

For each variant

1. cut the reference path around the variant
2. add the novel (ALT) sequence to the graph



Construction

POS	ID	REF	ALT
10	.	A	T
21	.	A	ATTAAGA
31	.	TCTTT	T

For each variant

1. cut the reference path around the variant
2. add the novel (ALT) sequence to the graph

vg construct

tiny.fa

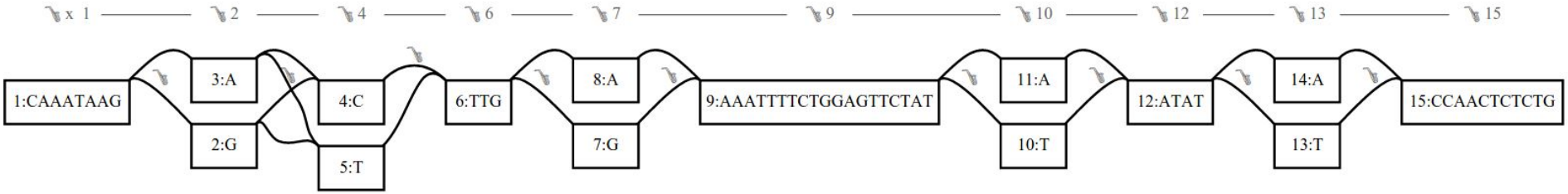
```
1:CAAATAAGGCTTGGAAATTTTCTGGAGTTCTATTATATTCCAACTCTCTG
```

tiny.vcf.gz

#CHROM	POS	REF	ALT
x	9	G	A
x	10	C	T
x	14	G	A
x	34	T	A
x	39	T	A

```
vg construct \
    -v tiny/tiny.vcf.gz \
    -r tiny/tiny.fa >tiny.vg
```

tiny.vg



Constructing a whole human genome variation graph

Constructed 1000G phase3 + GRCh37 variation graph using `vg construct`.

5h20m on 32-core system

3.07G on disk

3.181 Gbp of sequence in graph

286 million nodes (11 bp/node)

376 million edges (8.5 bp/edge)

Indexing the variation graph

As graphs grow large, memory-inefficient storage of the graph and its kmers in hash tables becomes infeasible.

The whole genome graph requires >500G if loaded directly into vg.

To efficiently query the graph, we transform it into a set of succinct databases.

Succinct variation graphs (xg)

A *succinct data structure*

- uses an amount of space close to the minimum information-theoretic lower bound,
- but does so while allowing efficient queries!

Core concept is the rank/select dictionary, a bit vector, e.g.

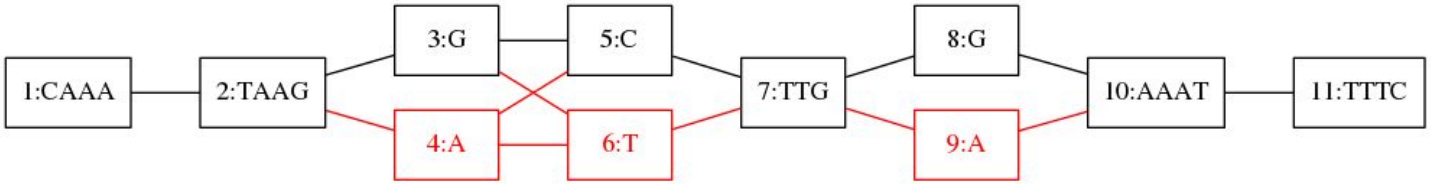
```
1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
```

That (given q ∈ {0,1}) supports functions:

$$\mathbf{rank}_q(x) = |\{k \in [0 \ldots x] : B[k] = q\}|$$

$$\mathbf{select}_q(x) = \min\{k \in [0 \ldots n) : \mathbf{rank}_q(k) = x\}$$

xg: nodes and edges



nodes

label	CAAA TAAG G A C T TTG G A AAAT TTTC
nodes	1000 1000 1 1 1 1 100 1 1 1000 1000
id	1 2 3 4 5 6 7 8 9 10 11

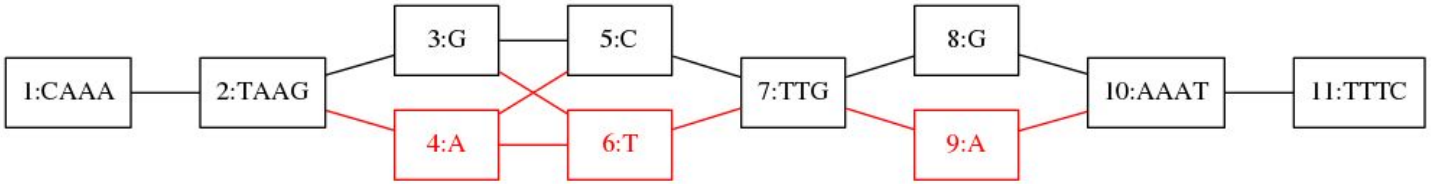
Graph storage is straightforward, and complicated only be the fact that nodes have variable length.

edges

The from and to integer vectors store [id][edges from or to id...][id+1][edges]

from	1 2 2 3 4 3 5 6 4 5 6 5 7 6 7 7 8 9 8 10 9 10 10 11 11
...	1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
to	1 2 1 3 2 4 2 5 3 4 6 3 4 7 5 6 8 7 9 7 10 8 9 11 10
...	1 1 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1 0

xg: positional paths



paths* (*for each path)

members `11110110000110011011001111`

a collection of nodes and edges--- we can use this structure alone for annotating subgraphs.

ordered node ids `1 2 3 5 7 8 10 11`

node path offsets `0 4 8 9 10 13 14 18`

node starts `10001000111001100010000`

In conjunction these structures allow navigating the graph using path-relative coordinates.

We can find the node at a particular path position by rank_1(i) on the node starts bit vector, and we can find the position of a node in a path using the ordered node ids, which is indexed using a wavelet tree.

xg: index construction

Construction is memory-intensive due to poor implementation, but feasible on our systems:

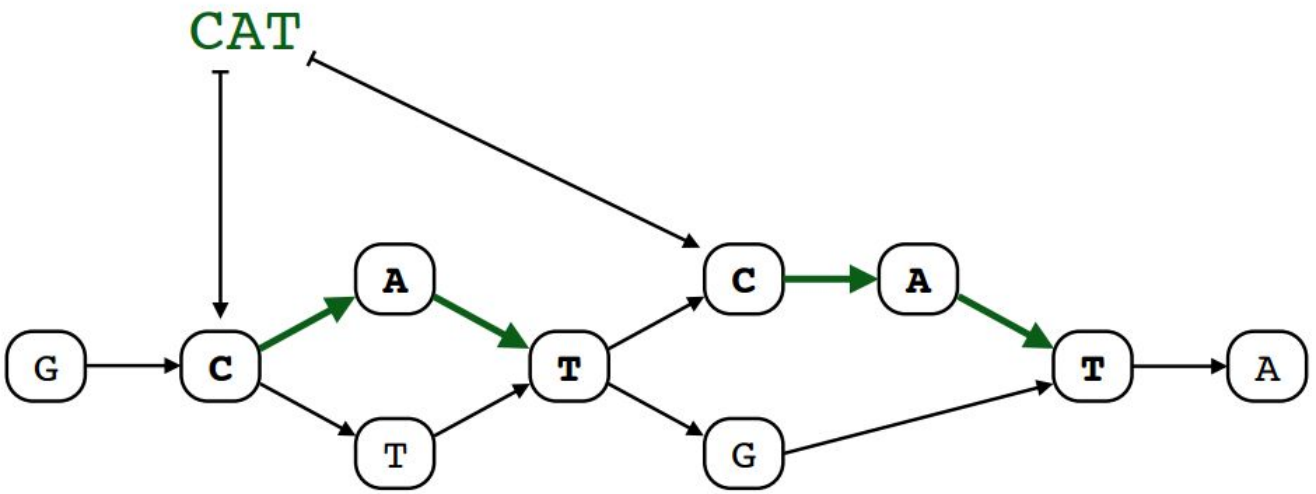
~125G of RAM and 1.5 hours on a single CPU.

The resulting index is 12G on disk and in memory.

Indexing path sequences: GCSA2

Given a graph where paths are labeled by strings, a path index is a text index for the strings.

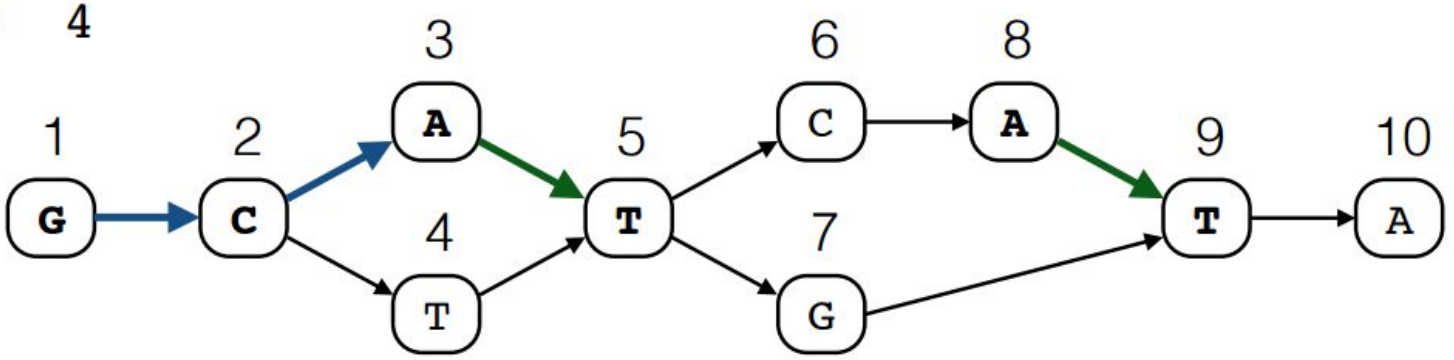
A path query finds the (start nodes of) the paths labeled by a kmer.



Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf

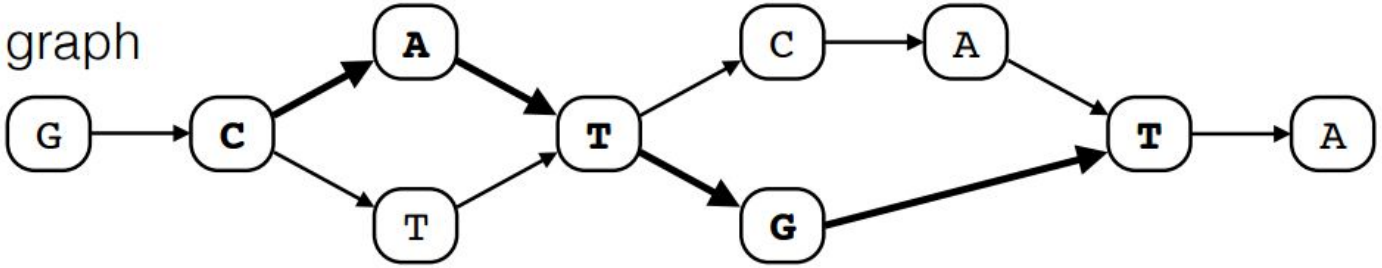
```
A$$   10
ATA   8
ATC   3
ATG   3
CAT   2, 6
CTT   2
GCA   1
GCT   1
GTA   7
TA$   9
TCA   5
TGT   5
TTC   4
TTG   4
```

- A kmer index based on a hash table supports queries of length k efficiently.
- If we sort the kmers, we can use them as a suffix array-like index for shorter queries.
- The kmer index can also simulate a de Bruijn graph.

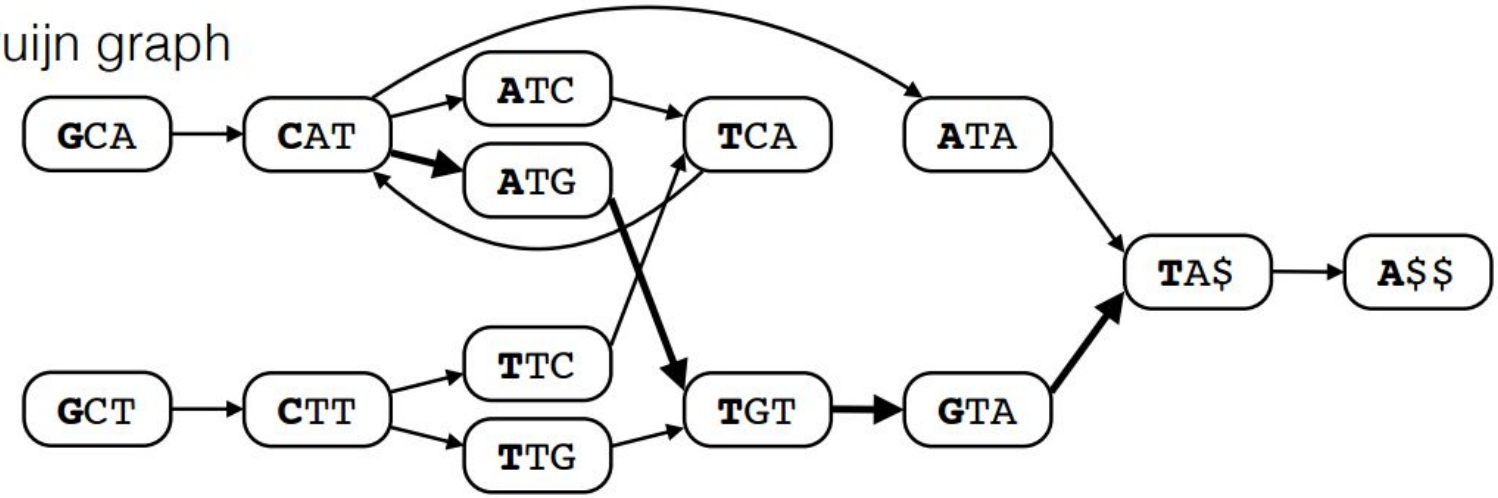


Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf

Original graph

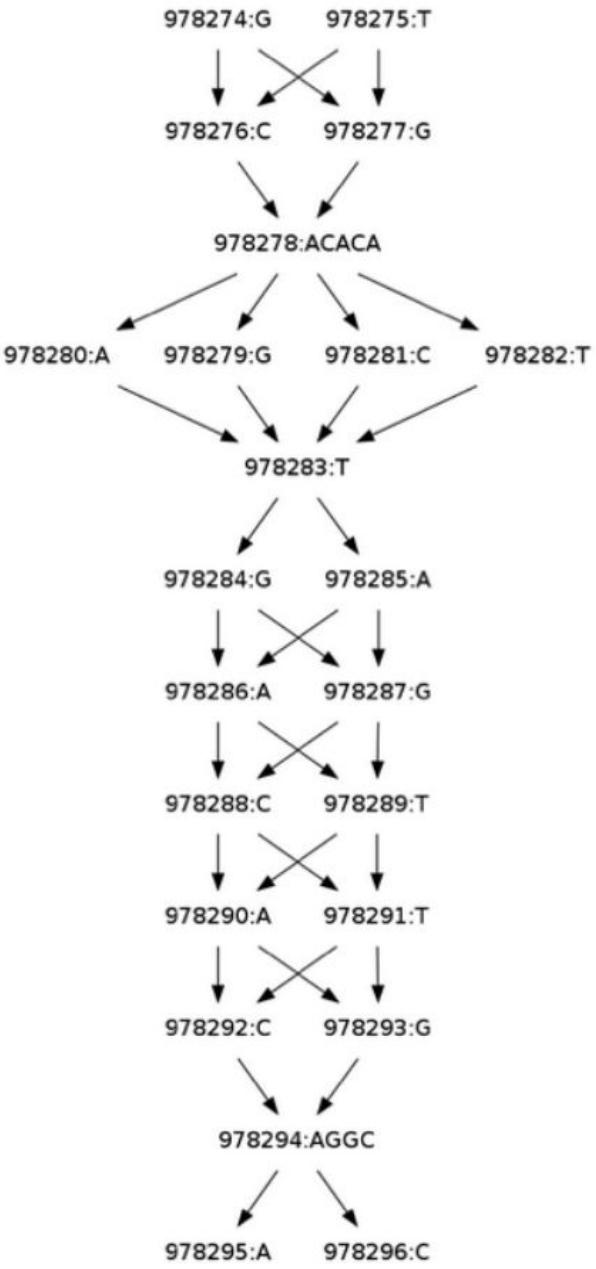


de Bruijn graph



We can search for longer patterns by representing the kmer index as a de Bruijn graph.

Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf

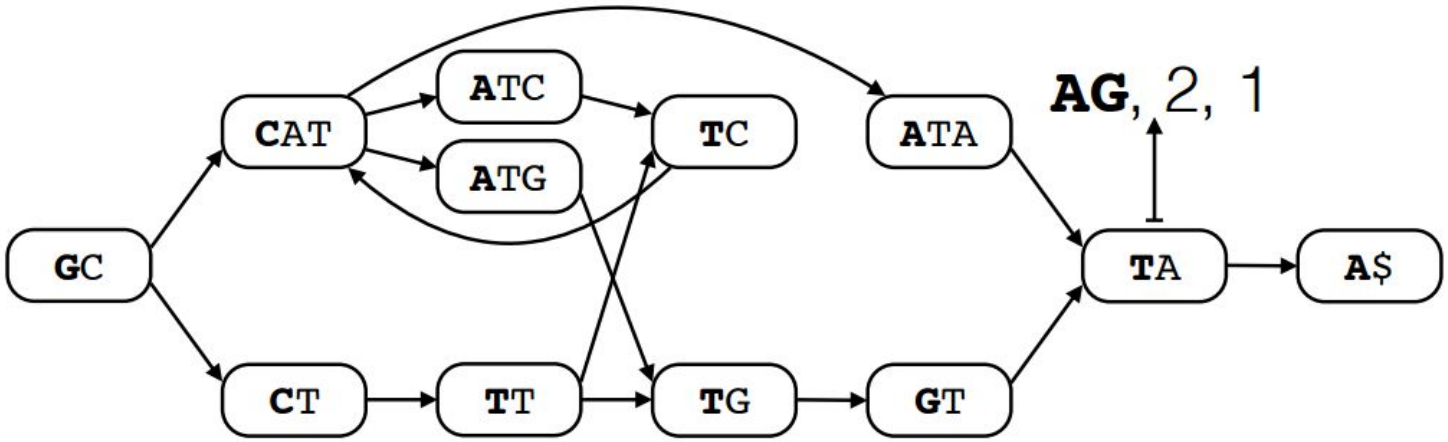


Some parts of the original graph may have **too many paths** through them. Those parts must be **pruned** before indexing.

The de Bruijn graph can also be pruned by **merging** the nodes with a **common prefix** of the label, if:

1. the shorter label **uniquely defines** the start node in the original graph; or
2. the start nodes **cannot be distinguished** by length-k extensions of the label.

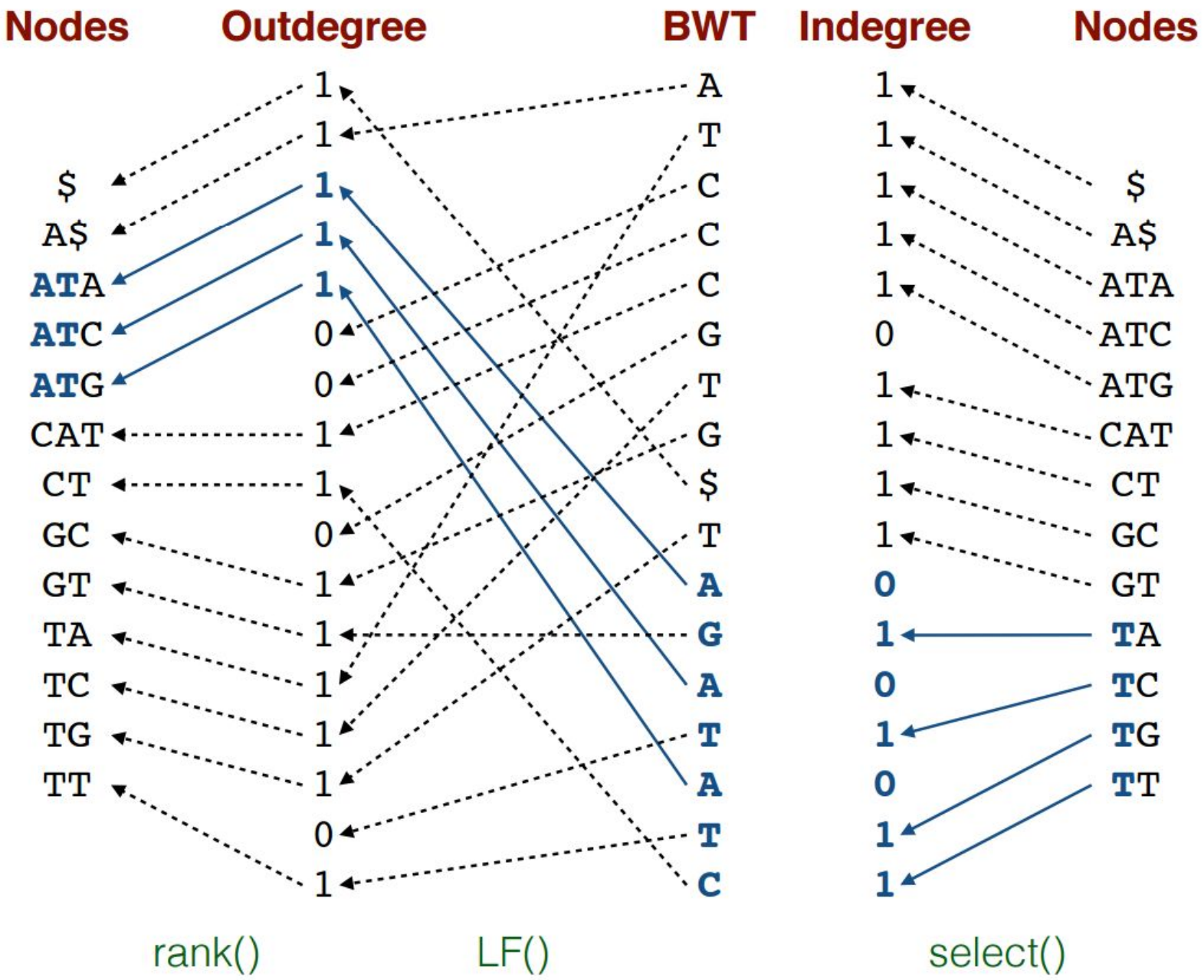
Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf



We store predecessor labels, indegree, and outdegree for each node. For the nodes at the beginning of unary paths, we also store pointers to the original graph. Edges can be determined if the nodes are stored in sorted order.

The encoding is similar to the Burrows-Wheeler transform and the FM-index. Typical space usage is 1–2 bytes/node.

Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf



Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf

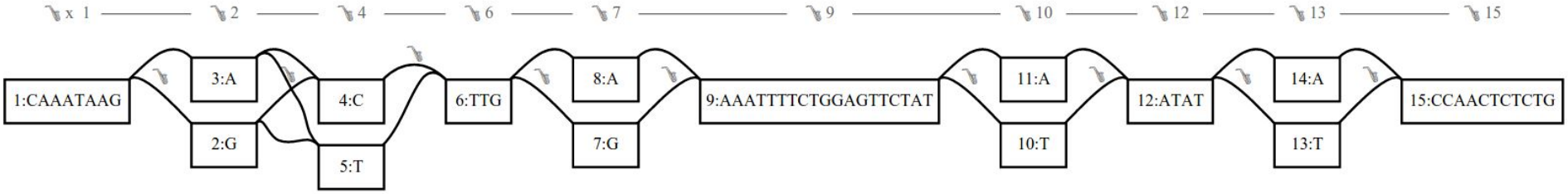
Path length	16→32	16→64	16→128
Nodes:			
de Bruijn graph	6.23G	16.9G	118G
Pruned	4.39G	5.27G	5.76G
Index size:			
Full index	9.99 GB	9.22 GB	9.23 GB
Without pointers	4.10 GB	4.84 GB	5.27 GB
Construction:			
Time	7.20 h	11.4 h	15.5 h
Memory	43.8 GB	43.8 GB	43.8 GB
Disk	401 GB	424 GB	489 GB
I/O:			
Read	1.43 TB	2.11 TB	2.89 TB
Write	1.05 TB	1.71 TB	2.47 TB

1000GP human variation, vg mod -p -l 16 -e 4 | vg mod -S -l 100
32 cores, 256 GB memory, distributed Lustre file system

Jouni Sirén http://jltsiren.kapsi.fi/files/sendai2016.pdf

vg index

tiny.vg




```
vg index tiny.vg \
    -x tiny.xg \
    -g tiny.gcsa -k 16
```

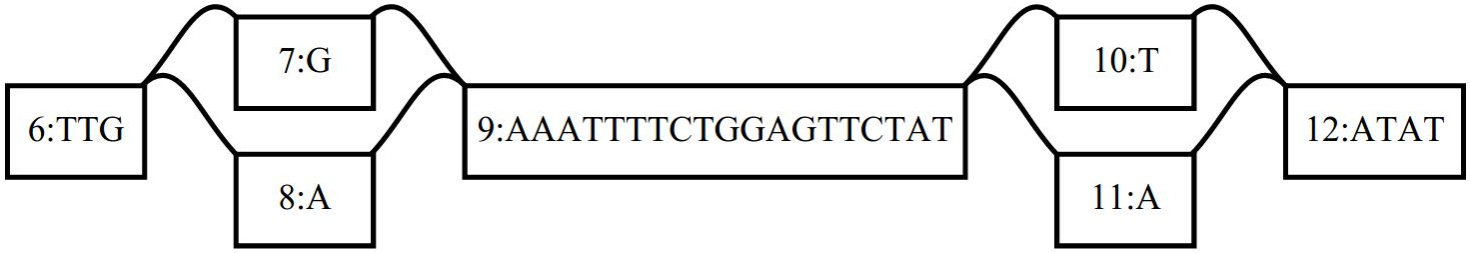
```
 200 tiny.vg
5.1K tiny.xg
4.4K tiny.gcsa
```

```
vg kmers -gk 16 tiny.vg | head -50
CTTGAAAATTTTCTGG        4:0     A,G     A       9:11
ACTTGAAAATTTTCTG        3:0     G       G       9:10
CTTGGAAATTTTCTGG        4:0     A,G     A       9:11
GCCTTATTTG$$$$$$        4:-0    A       $       17:6
ACTTGGAAATTTTCTG        3:0     G       G       9:10
GTCTTATTTG$$$$$$        4:-0    A       $       17:6
ATTTGAAAATTTTCTG        3:0     G       G       9:10
ATTTGGAAATTTTCTG        3:0     G       G       9:10
TCTTATTTG$$$$$$$        3:-0    A,G     $       17:7
CCTTATTTG$$$$$$$        2:-0    A,G     $       17:7
GCTTGAAAATTTTCTG        2:0     G       G       9:10
GCTTGGAAATTTTCTG        2:0     G       G       9:10
GTTTGAAAATTTTCTG        2:0     G       G       9:10
GTTTGGAAATTTTCTG        2:0     G       G       9:10
ACCTTATTTG$$$$$$        5:-0    A       $       17:6
ATCTTATTTG$$$$$$        5:-0    A       $       17:6
TTTGAAAATTTTCTGG        5:0     A,G     A       9:11
TTTGGAAATTTTCTGG        5:0     A,G     A       9:11
CCAAACCTTATTTG$$        7:-0    T       $       17:2
```

vg find

```
vg find -x tiny.xg \
    -p x:20-25 -c 1 \
   | vg view -d -
```

Query the nodes around x:20-25 in the reference path “x”.



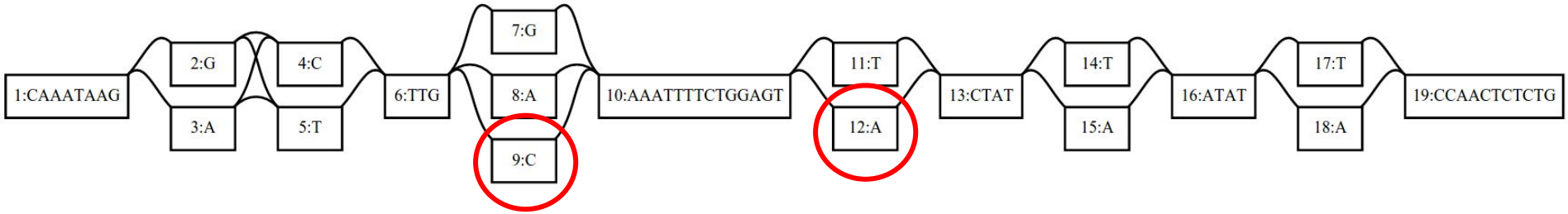

```
vg find -g tiny.gcsa \
    -S TCCAGAAAATTTTCAA
→ 9:-7
```

Query the position of a particular sequence in the GCSA2 index.

vg sim

Use a haplotype representing some variants relative to the tiny.vg to build a new graph:

```
vg msga -g tiny.vg -Nz \
    -s CAAATAAGGTTTGCAAATTTTCTGGAGTACTATAATATTCCAACTCTCTG \
    >truth.vg
```



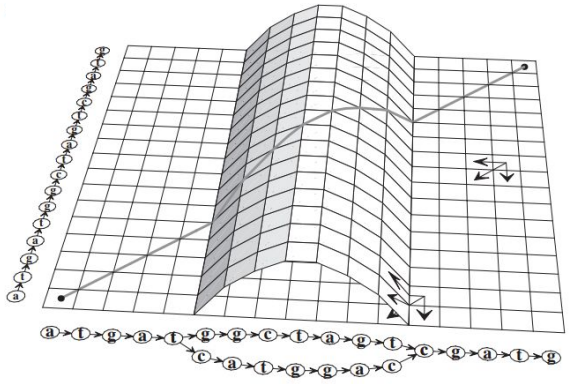
We can then use it as a generative model and sample reads from it:

```
vg sim -l 50 -n 10 -s 1337 truth.vg >truth.reads
```

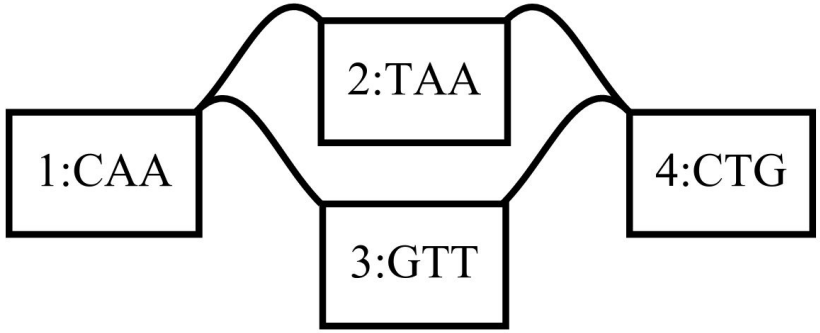
Partial order alignment

query:
CAAATTCT

	T	A	A
C	0	0	0
A	0	2	2
A	3	2	4
A	4	5	4
T	6	3	3
T	4	4	1
C	2	2	2
T	2	0	0



	C	A	A
C	**2**	0	0
A	0	**4**	2
A	0	2	**6**
A	0	2	4
T	0	0	2
T	0	0	1
C	2	0	0
T	0	0	0



	C	T	G
C	2	0	0
A	0	0	0
A	1	0	0
A	2	0	0
T	2	4	1
T	5	4	3
C	**10**	7	6
T	7	**12**	9

	G	T	T
C	0	0	0
A	0	0	0
A	3	2	1
A	**4**	1	0
T	2	**6**	3
T	0	4	**8**
C	0	2	5
T	0	2	4

1. fill the score matrixes
2. find the maximum score
3. trace back for alignment

scores:
match = 2
mismatch = 2
gap_open = 3
gap_extension = 1

vg map

```
vg map -x tiny.xg -g tiny.gcsa -r truth.reads -k 16 >aln.gam
```

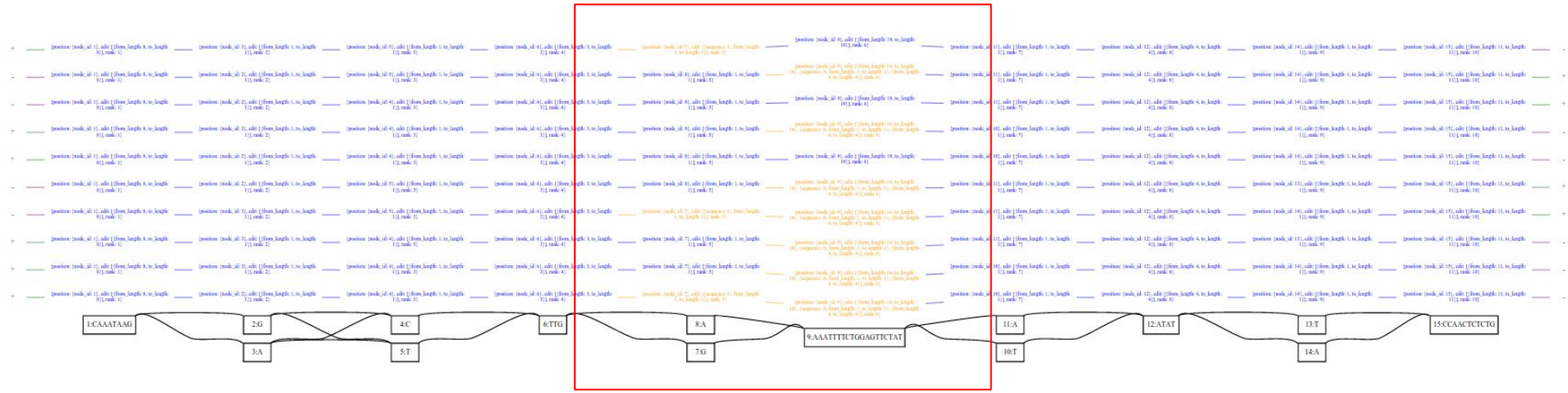
```
vg view -a aln.gam
```


```
{
  "sequence": "CAGAGAGTTGGTATATTATAGTACTCCAGAAAATTTGCAAATCTTATTTG",
  "path": {
    "mapping": [
      {
        "position": {
          "node_id": 15,
          "is_reverse": true
        },
        "edit": [
          {
            "from_length": 11,
            "to_length": 11
          }
        ],
        "rank": 1
      },
      {
        "position": {
          "node_id": 14,
          "is_reverse": true
        },
        "edit": [
          {
            "from_length": 1,
            "to_length": 1
          }
        ],
```

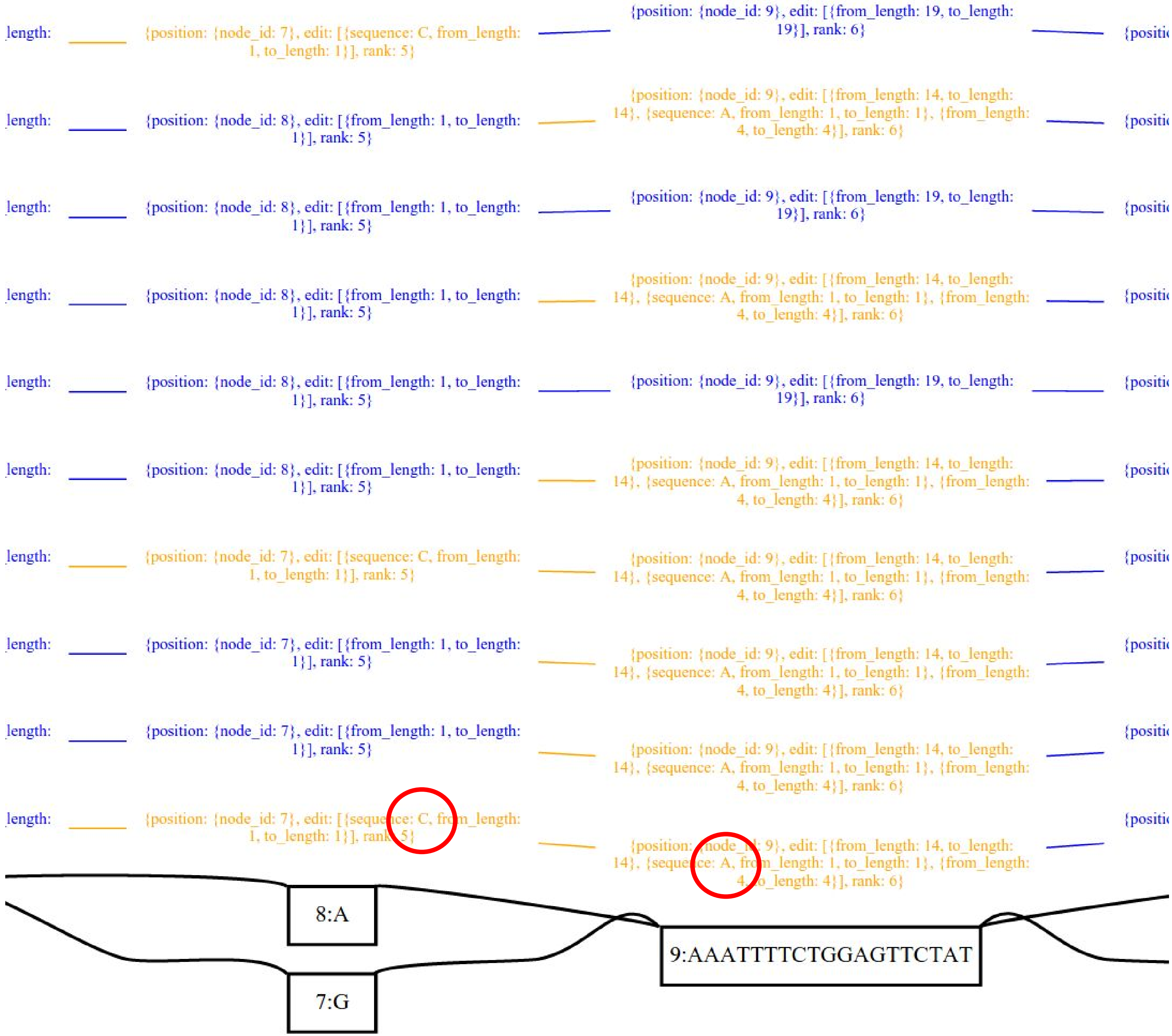
alignment viz

We can also visualize alignments against the graph:

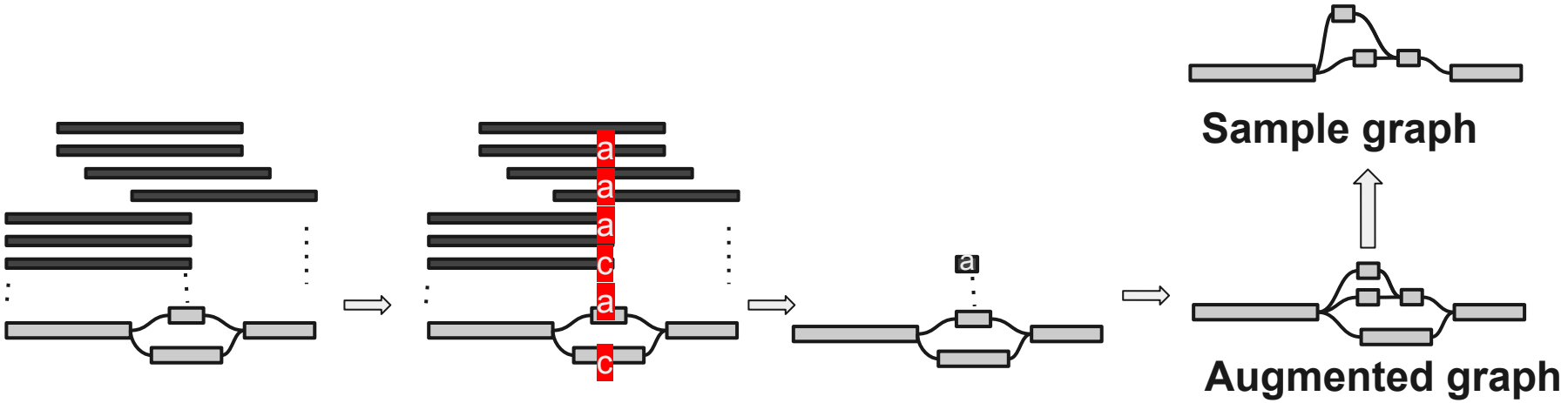
```
vg view -dA aln.gam tiny.vg
```



Blue represents perfect match.
Yellow represents a mismatch.


length:
{position: {node_id: 7}, edit: [{sequence: C, from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 19, to_length: 19}], rank: 6}
{positi
length:
{position: {node_id: 8}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
length:
{position: {node_id: 8}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 19, to_length: 19}], rank: 6}
{positi
length:
{position: {node_id: 8}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
length:
{position: {node_id: 8}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 19, to_length: 19}], rank: 6}
{positi
length:
{position: {node_id: 8}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
length:
{position: {node_id: 7}, edit: [{sequence: C, from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
length:
{position: {node_id: 7}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
length:
{position: {node_id: 7}, edit: [{from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
length:
{position: {node_id: 7}, edit: [{sequence: C, from_length: 1, to_length: 1}], rank: 5}
{position: {node_id: 9}, edit: [{from_length: 14, to_length: 14}, {sequence: A, from_length: 1, to_length: 1}, {from_length: 4, to_length: 4}], rank: 6}
{positi
8:A
7:G
9:AAATTTTCTGGAGTTCTAT

vg calling pipeline



Read alignment

- GAM format
- from `vg map`

Graph Pileup

- VG pileup format
- from `vg pileup`

Variants

- VG format
- from `vg call`

Calling pipeline is very similar to samtools, but also rudimentary (only toy examples at present).

Glenn Hickey

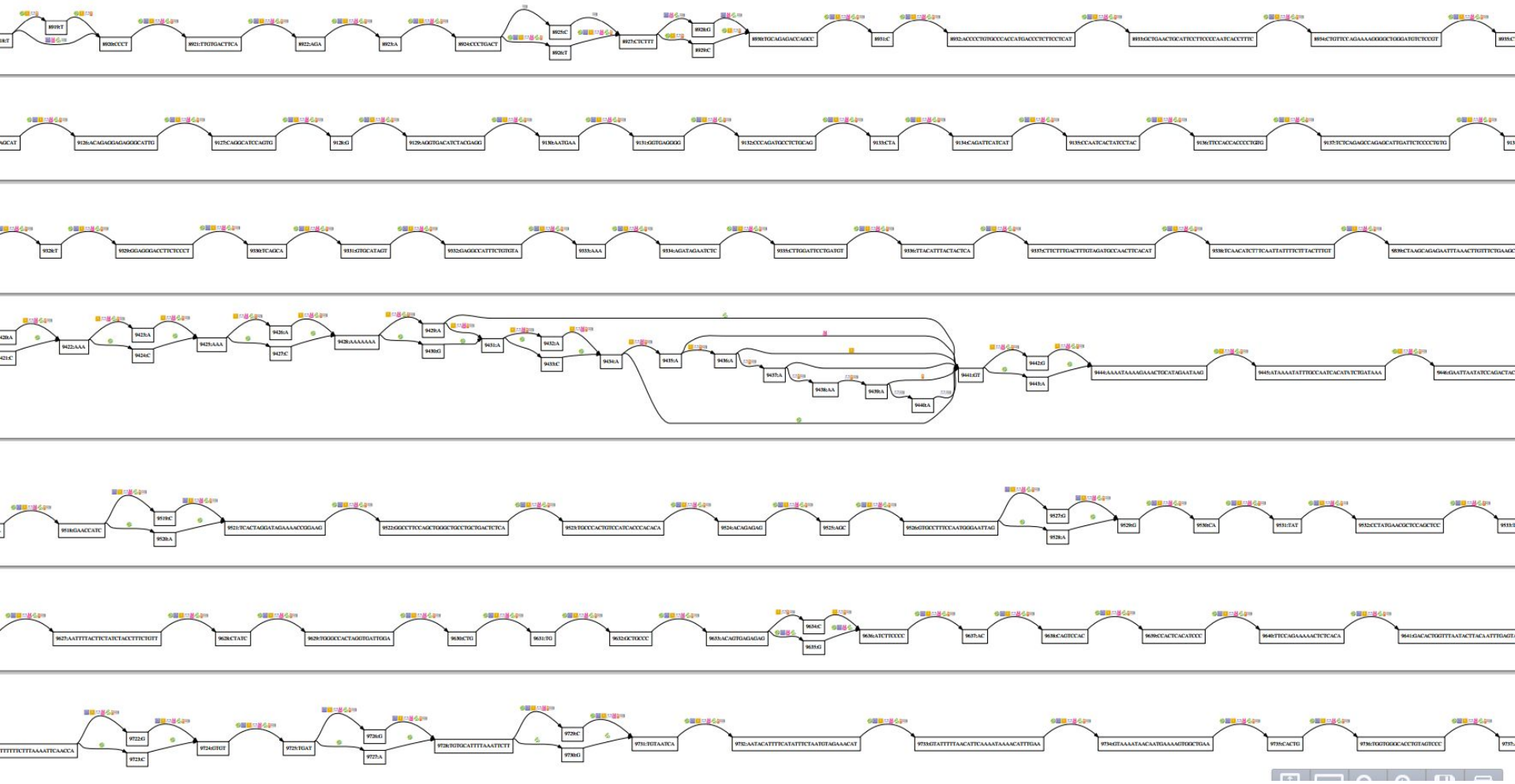
Long read assembly

vg can be used to make assemblies from sets of finished genomes:

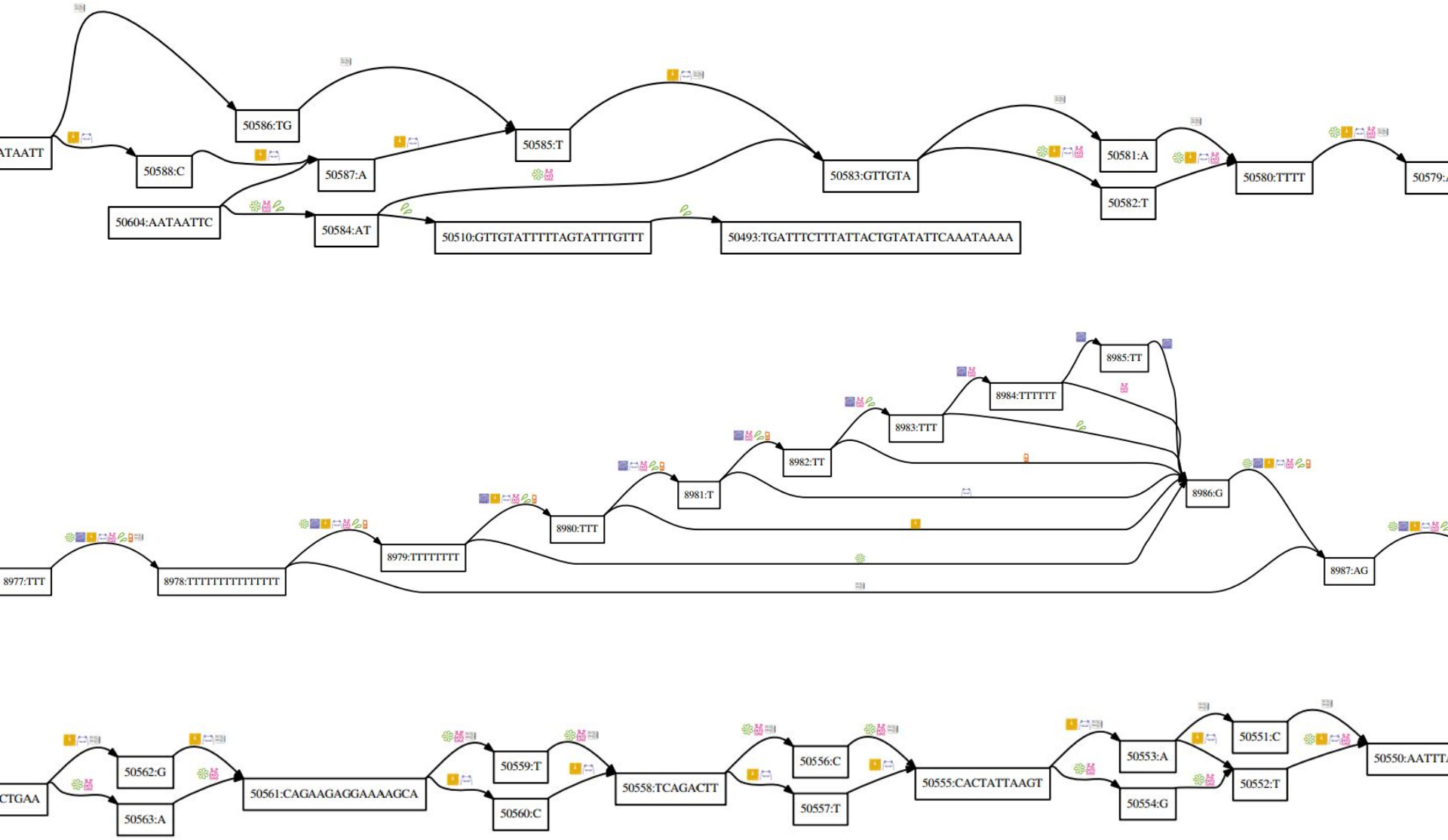
1. index (make our xg/gcsa supports)
2. align (map a sequence to the graph)
3. edit (include alignment in graph)
4. repeat …

Next slides: bits of the MHC.

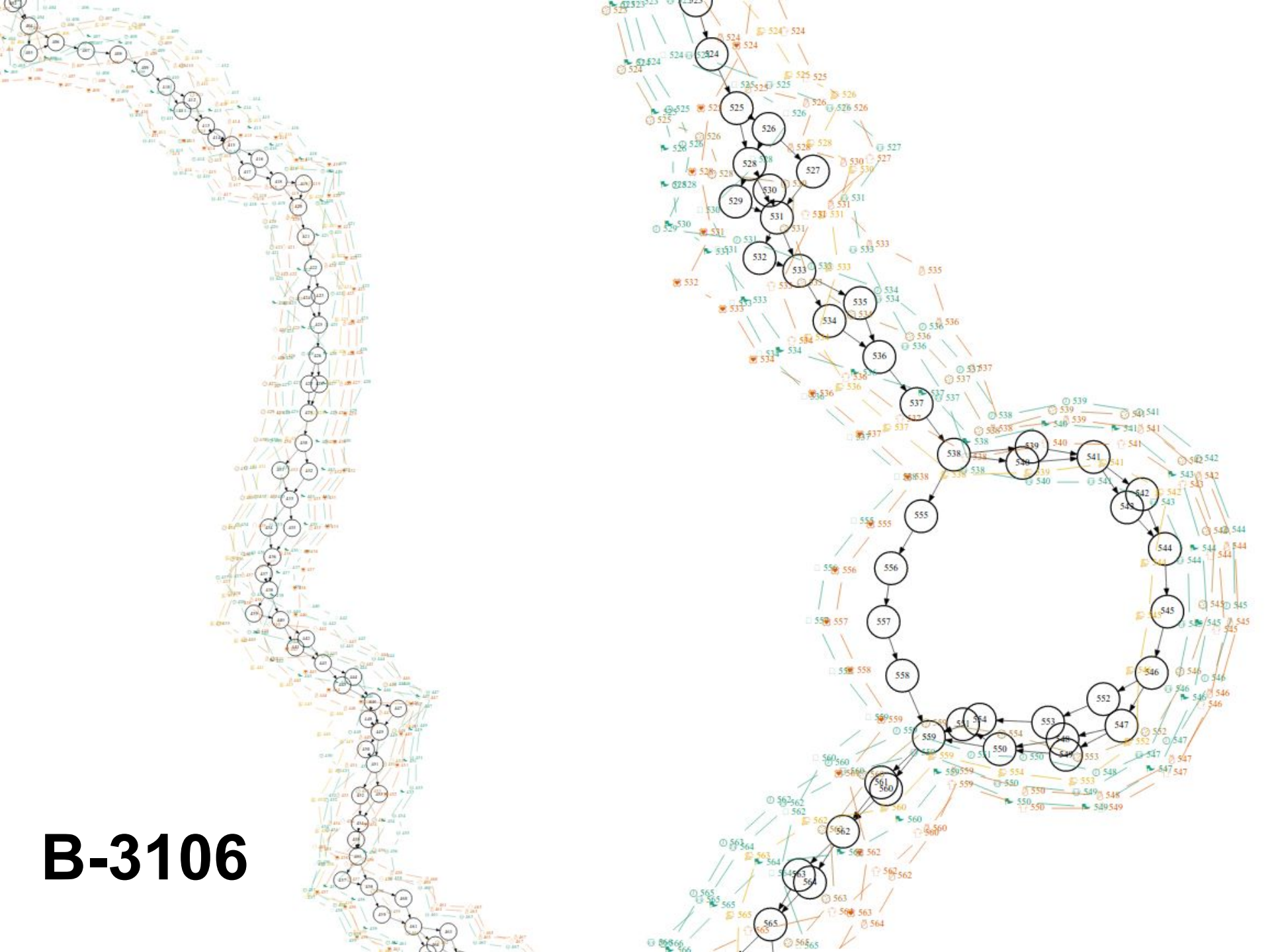
a 1000s of bp view of the MHC



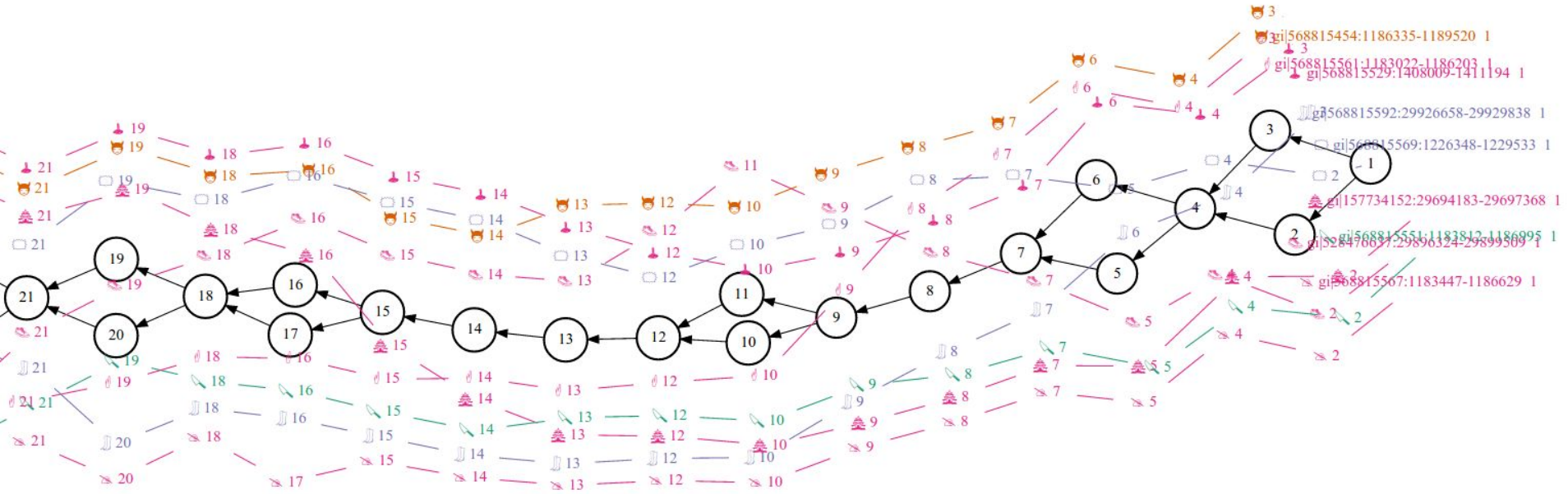
from 100s of bps


ATAATT
50586:TG
50588:C
50587:A
50585:T
50604:AATAATTC
50584:AT
50510:GTTGTATTTTAGTATTTGTTT
50493:TGATTTCTTTATTACTGTATATTCAAATAAAA
50583:GTTGTA
50581:A
50582:T
50580:TTTT
50579:
8977:TTT
8978:TTTTTTTTTTTTTT
8979:TTTTTTTT
8980:TTT
8981:T
8982:TT
8983:TTT
8984:TTTTTT
8985:TT
8986:G
8987:AG
CTGAA
50562:G
50563:A
50561:CAGAAGAGGAAAAGCA
50559:T
50560:C
50558:TCAGACTT
50556:C
50557:T
50555:CACTATTAAGT
50553:A
50554:G
50551:C
50552:T
50550:AATTT

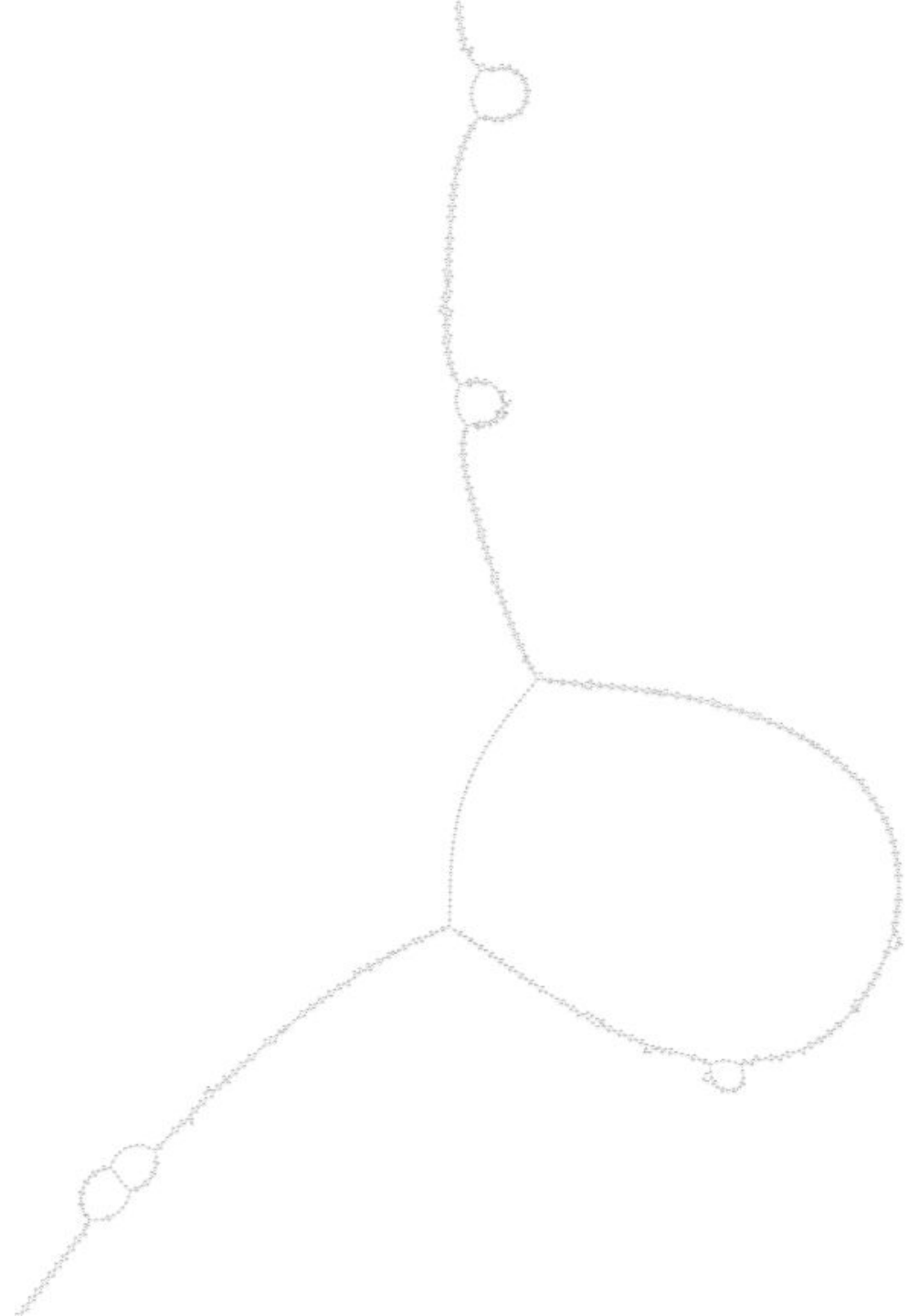
up close


B-3106

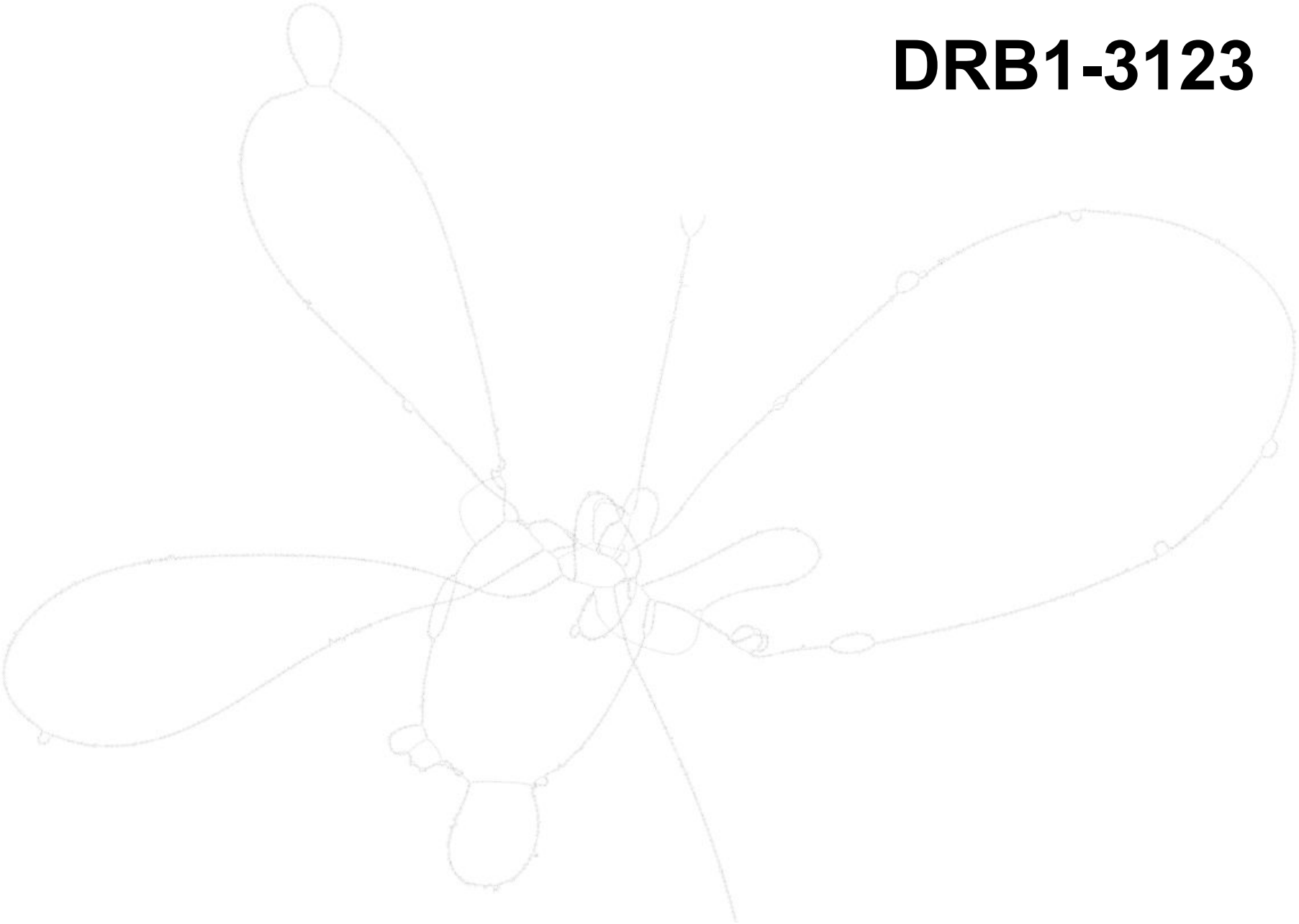
K-3138


gi|568815454:1186335-1189520 1
gi|568815561:1183022-1186203 1
gi|568815529:1408009-1411194 1
gi|568815592:29926658-29929838 1
gi|568815569:1226348-1229533 1
gi|157734152:29694183-29697368 1
gi|568815551:1183812-1186995 1
gi|528476637:29896324-29899509 1
gi|568815567:1183447-1186629 1

DQA1-3117



DRB1-3123



DRB1-3123

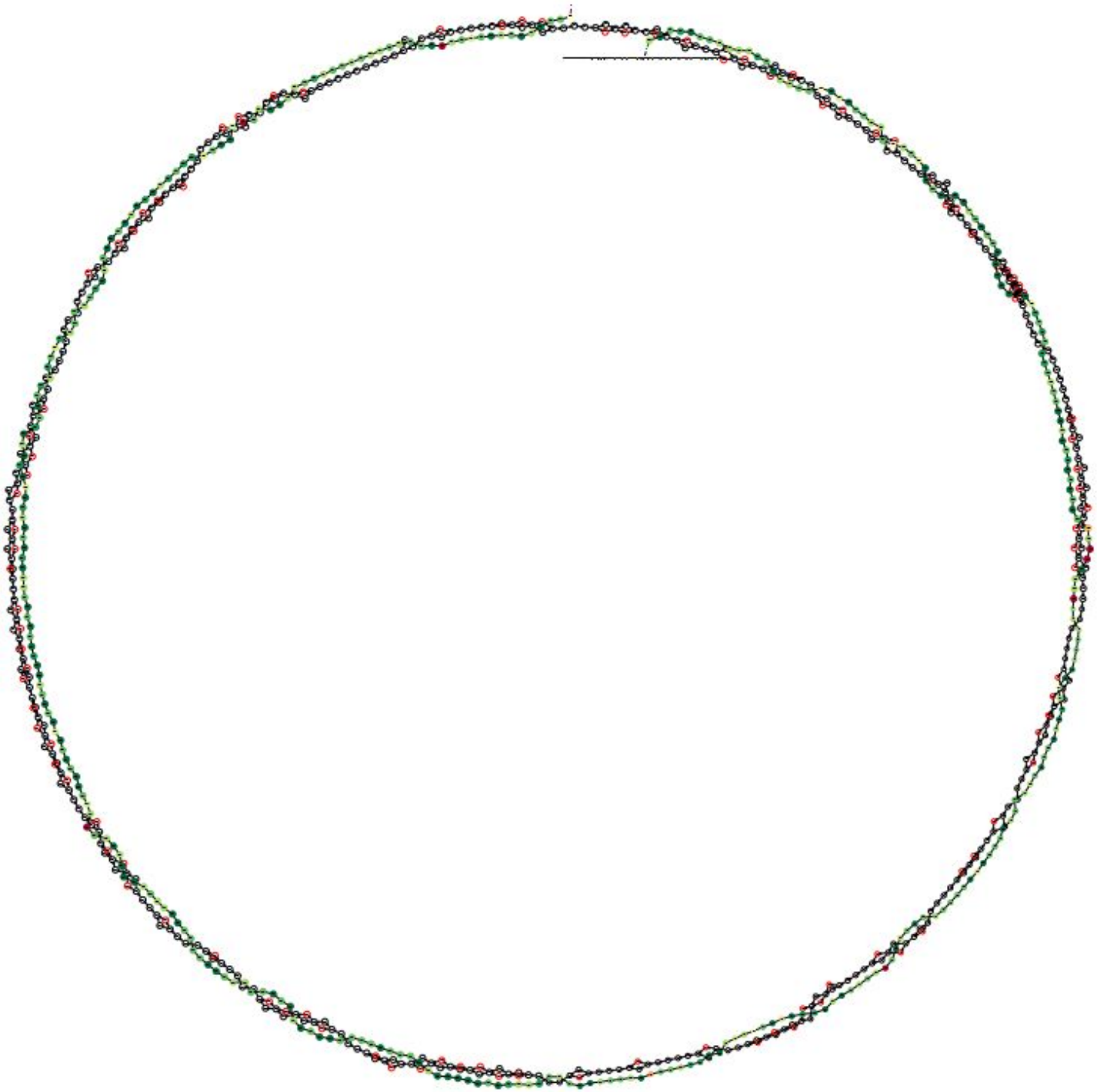
HPV nanopore sequencing

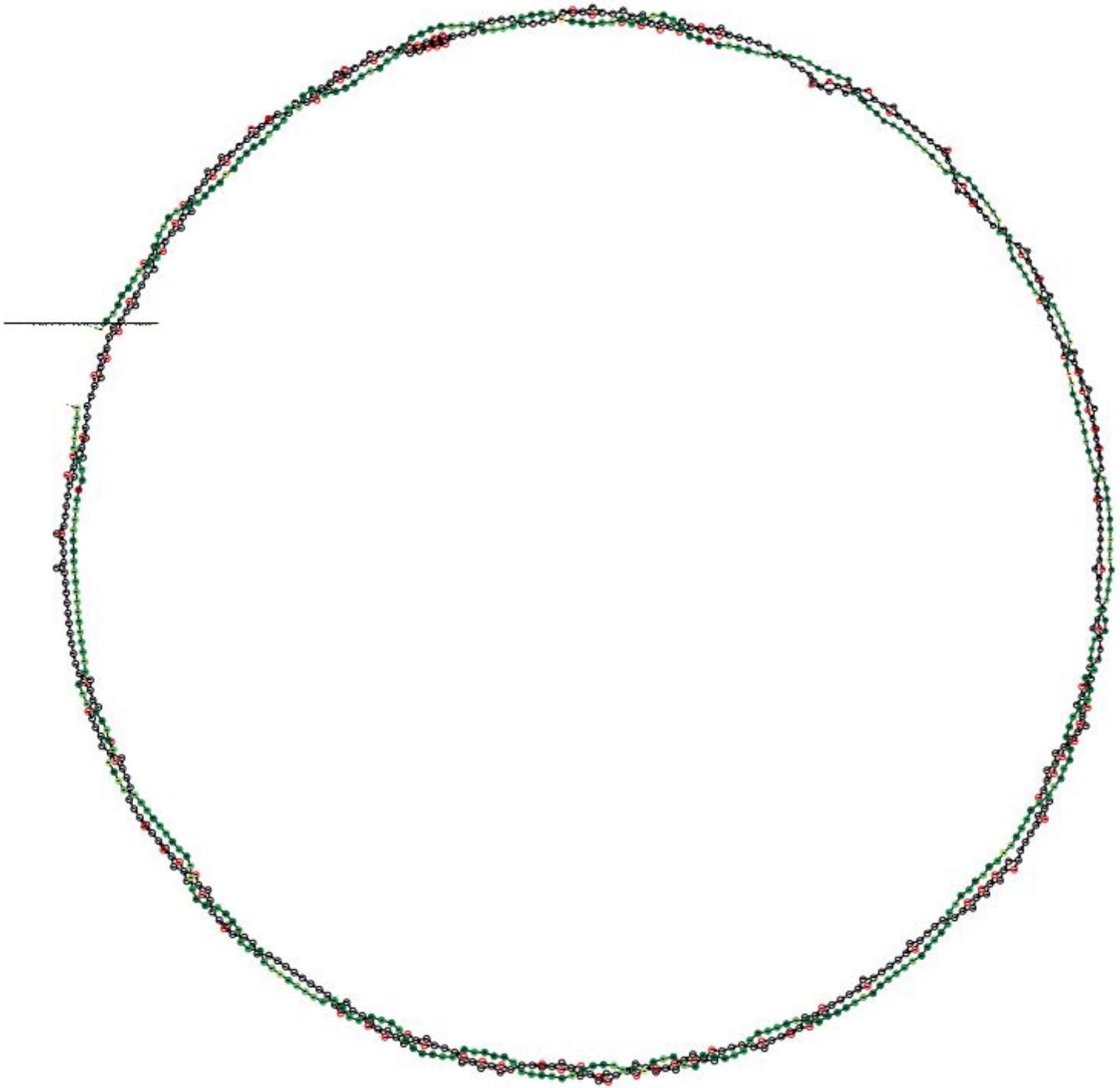
Project: given a pool of HPV sequences, try to determine the fractions of two disparate strains.

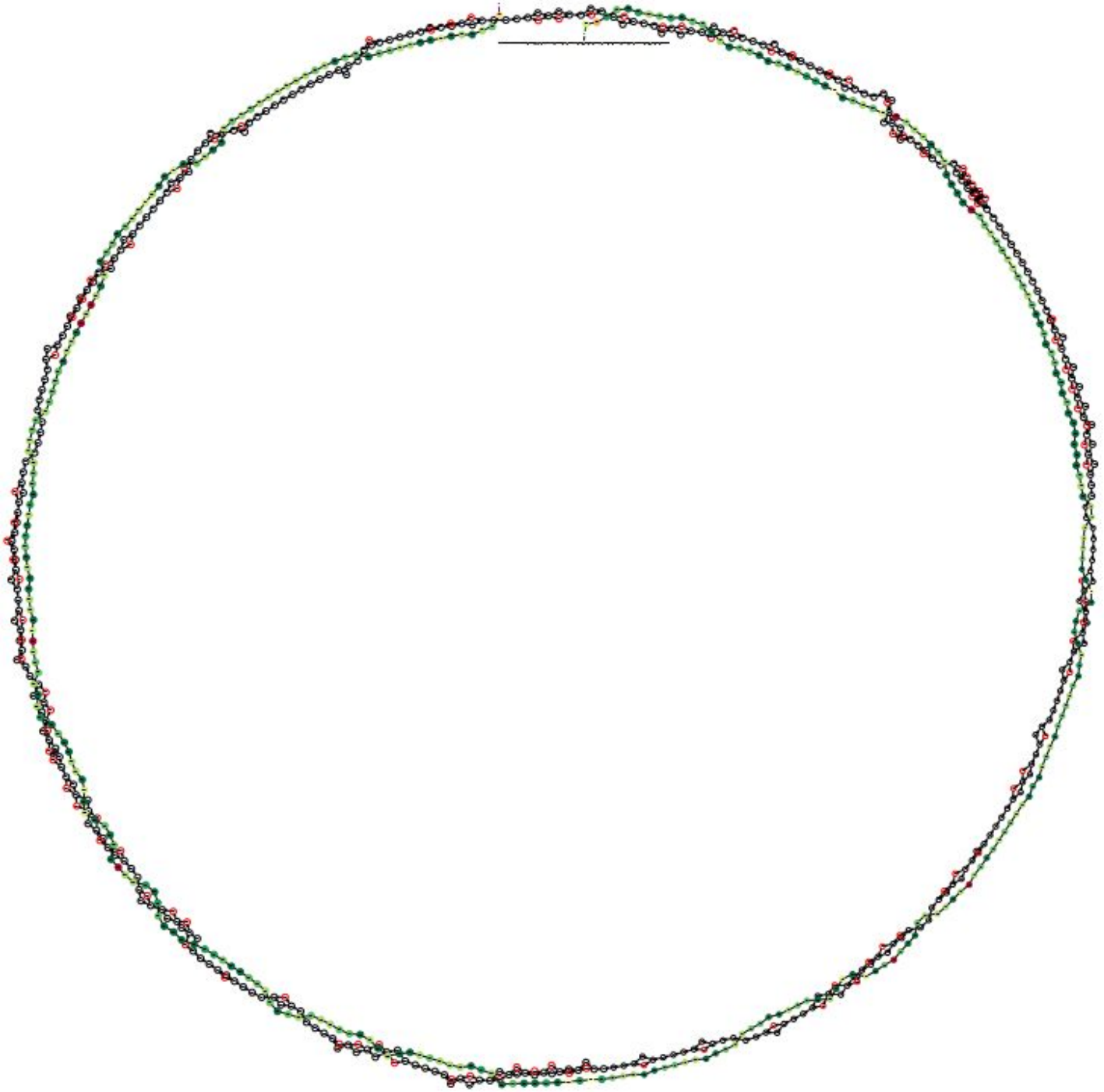
Problems with standard approaches.

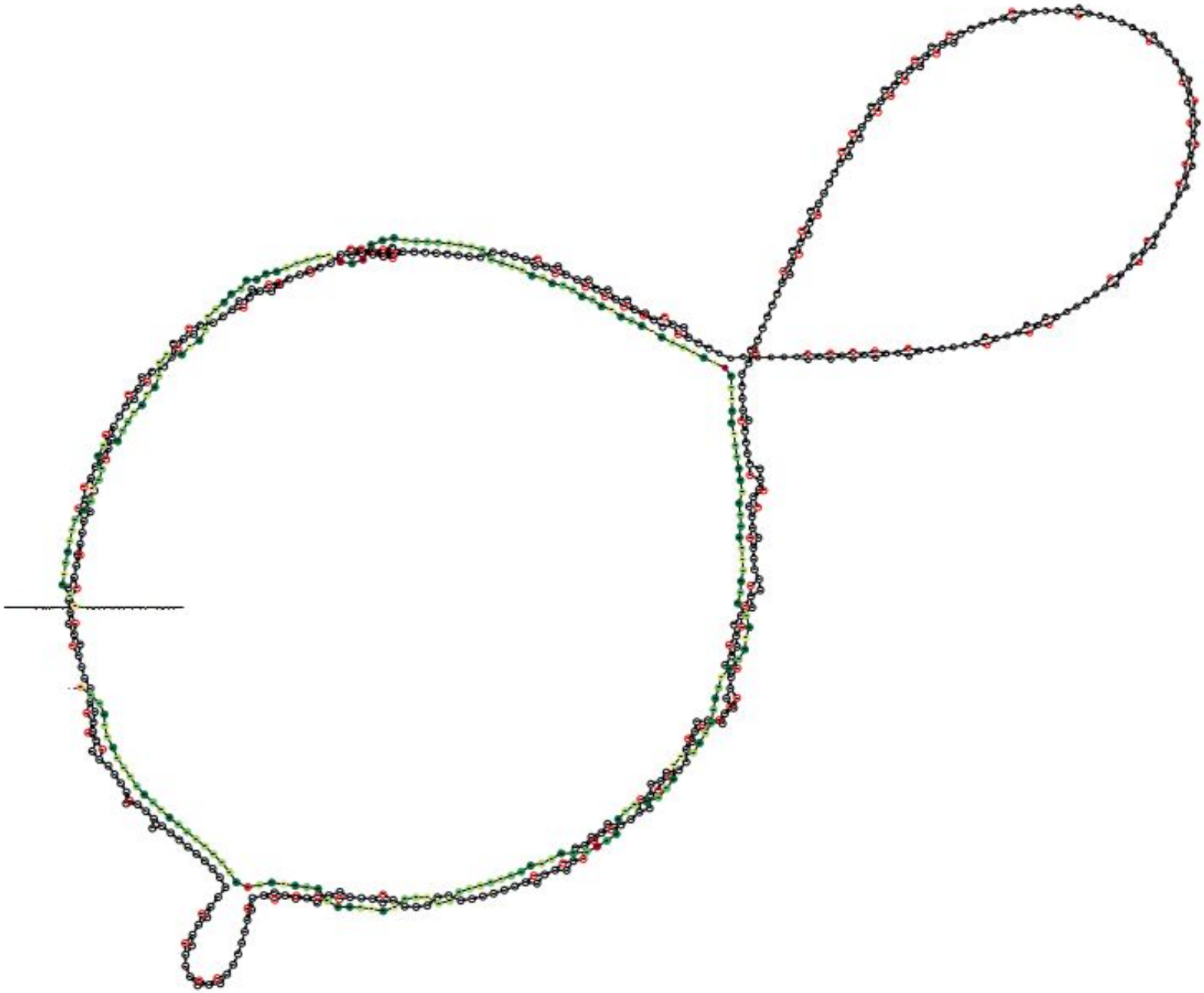
1. Genome is circular (FASTA files are not)
2. Disparate genomes → two separate references?

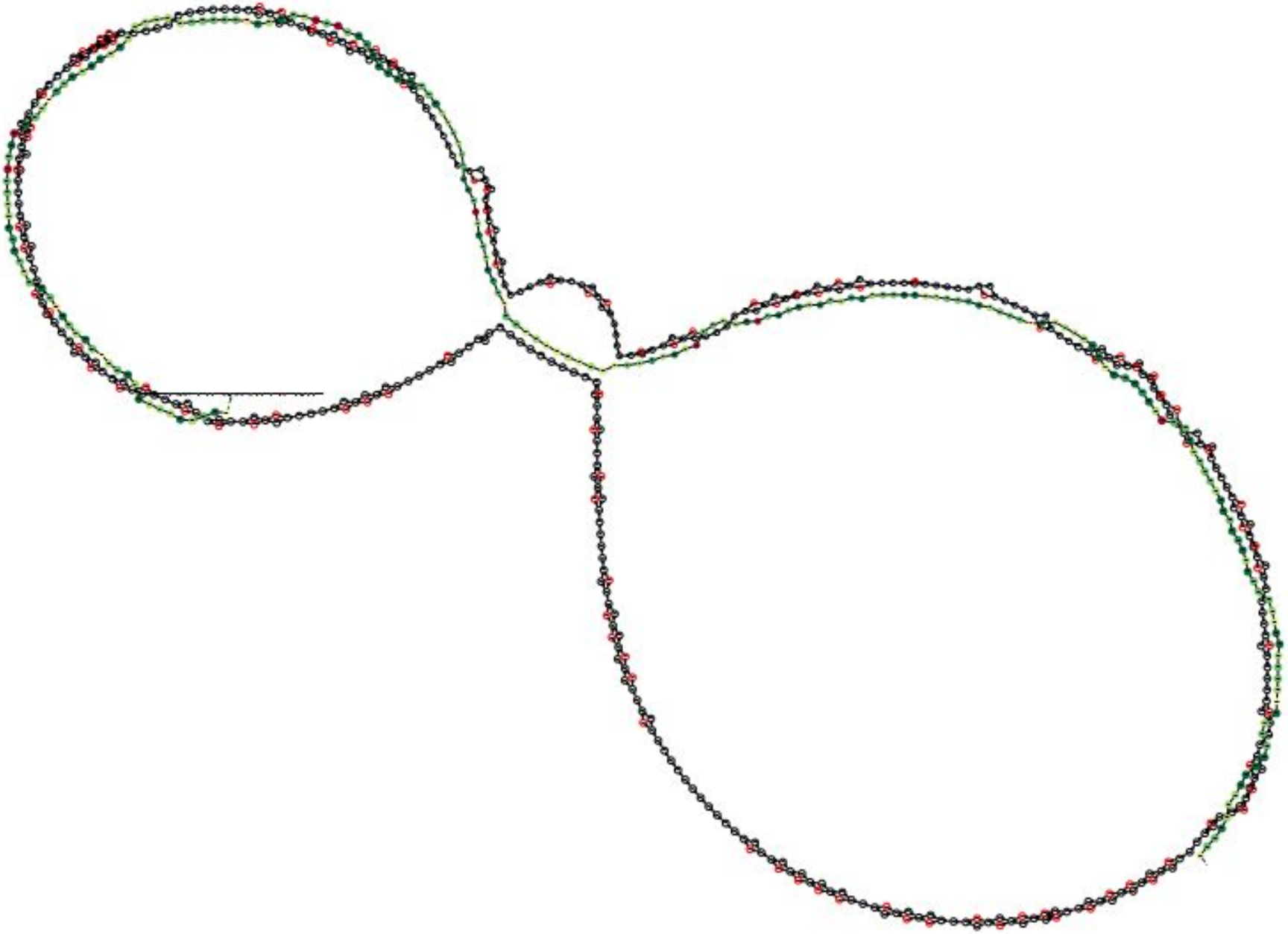
We make one reference and align everything!

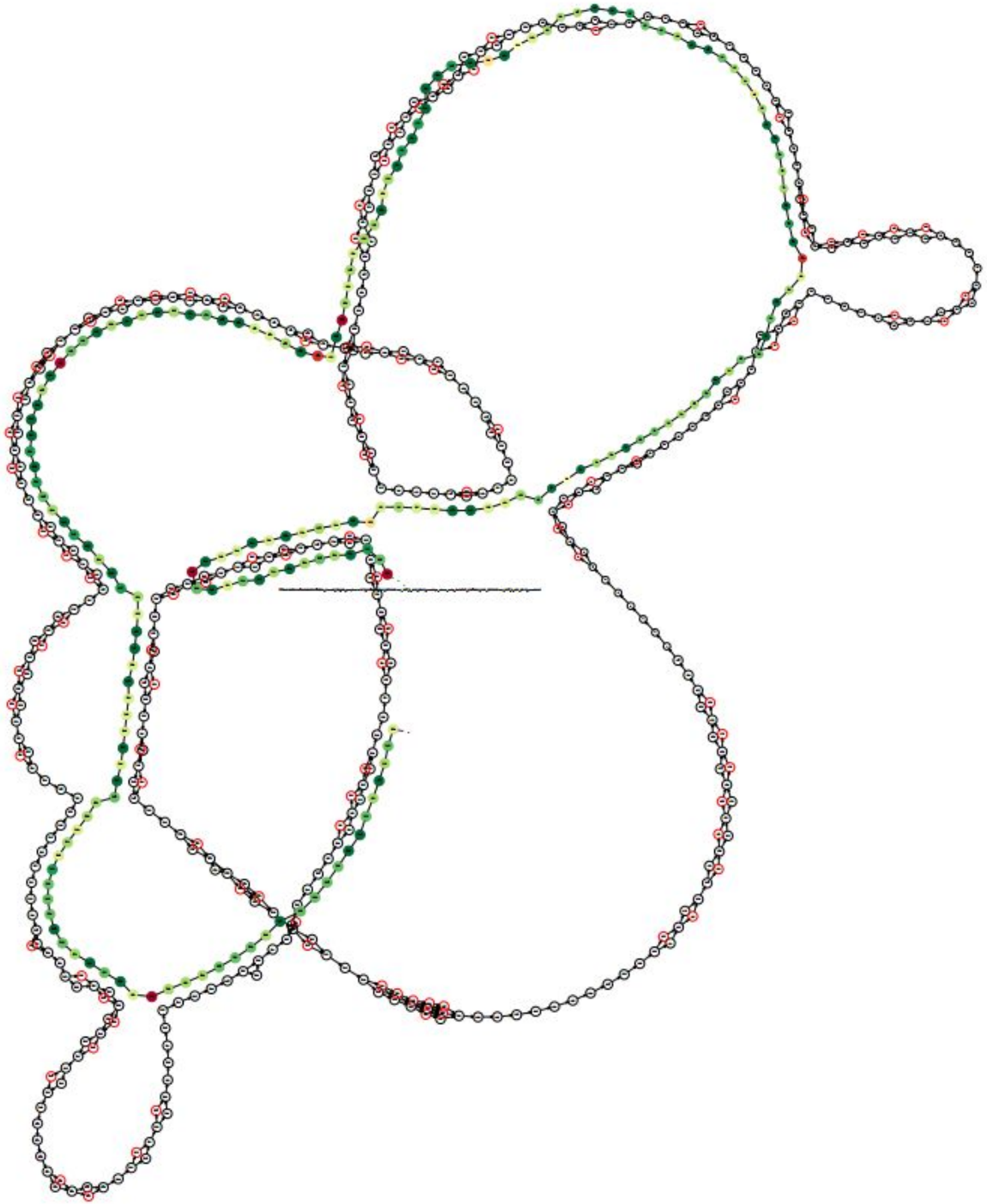












Practical!

https://github.com/ekg/alignment-and-variant-calling-tutorial

Practical: data

We'll use a few data sources:

- E. Coli (for testing alignment)
- NA12878 (human cell line with truth set)
- simulated data

Practical:
alignment and variant calling

1. get our reference genome
2. index it (BWT/FM-index generation)
3. align
4. mark duplicates
5. call variants
6. simple filters

Practical:
filtering

Using a fragment of chr20 and alignments from NA12878, we'll explore filtering using the Genome in a Bottle truth set.

Practical:
things you can do with freebayes

1. defaults
2. parallelism
3. haplotype-based techniques
4. filtering
5. quality assessment
6. pooled
7. polyploid

(We can use simulated data for this.)

Questions?

…