Snarls, pangenome deconstruction, and read mapping with vg giraffe

GET-A-PAN

Xian Chang & Jean Monlong

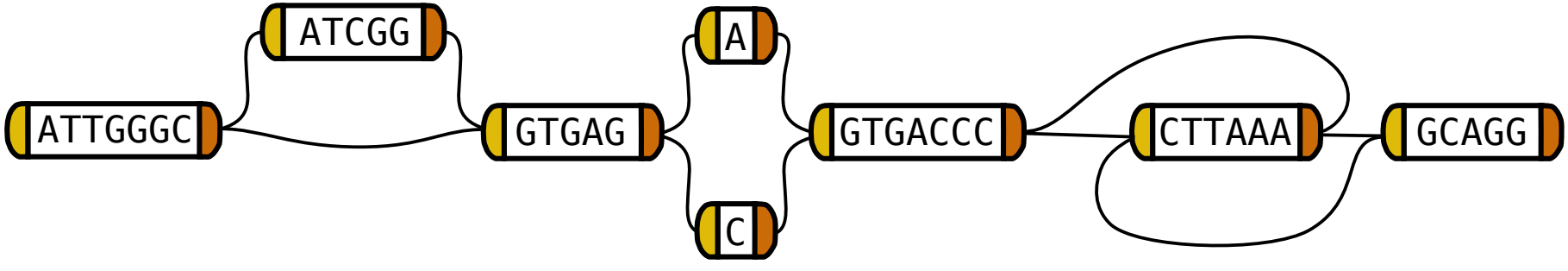
06/11/2025 - https://github.com/jmonlong/getapan2025

Pangenome

as a **bidirected** graph

ATTGGGC**ATCGG**GTGAG**A**GTGACCC**TTTAAG**GCAGG

ATTGGGC-----GTGAG**C**GTGACCC**CTTAAA**GCAGG

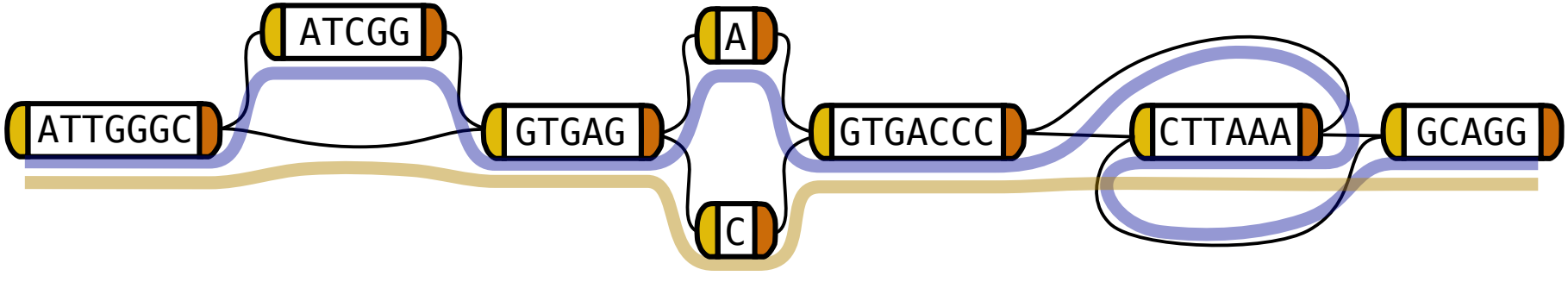


Pangenome

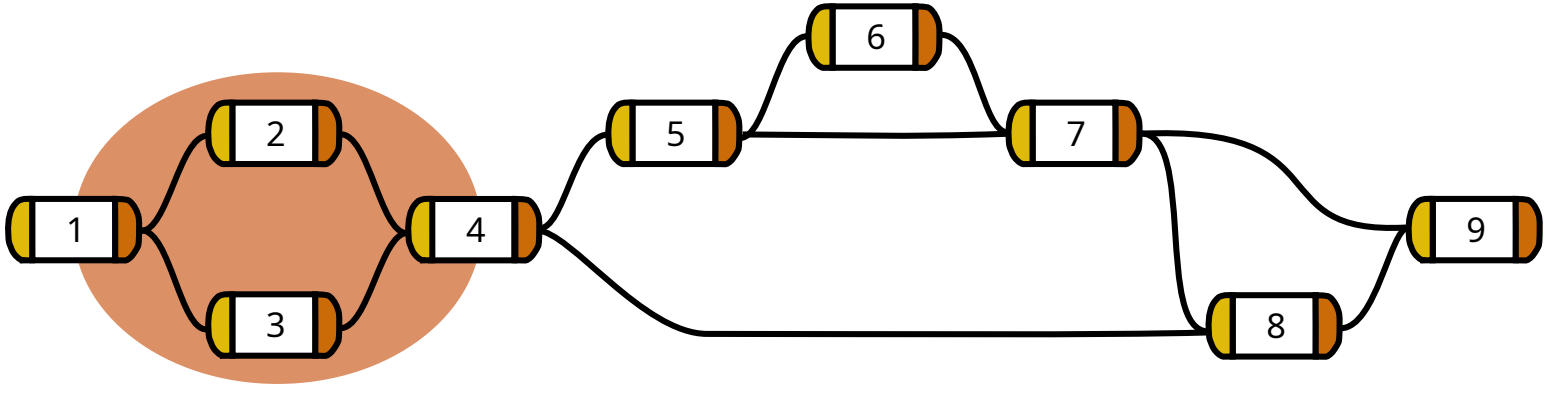
as a bidirected graph with **haplotype paths**

ATTGGGC**ATCGG**GTGAG**A**GTGACCC**TTTAAG**GCAGG

ATTGGGC-----GTGAG**C**GTGACCC**CTTAAA**GCAGG

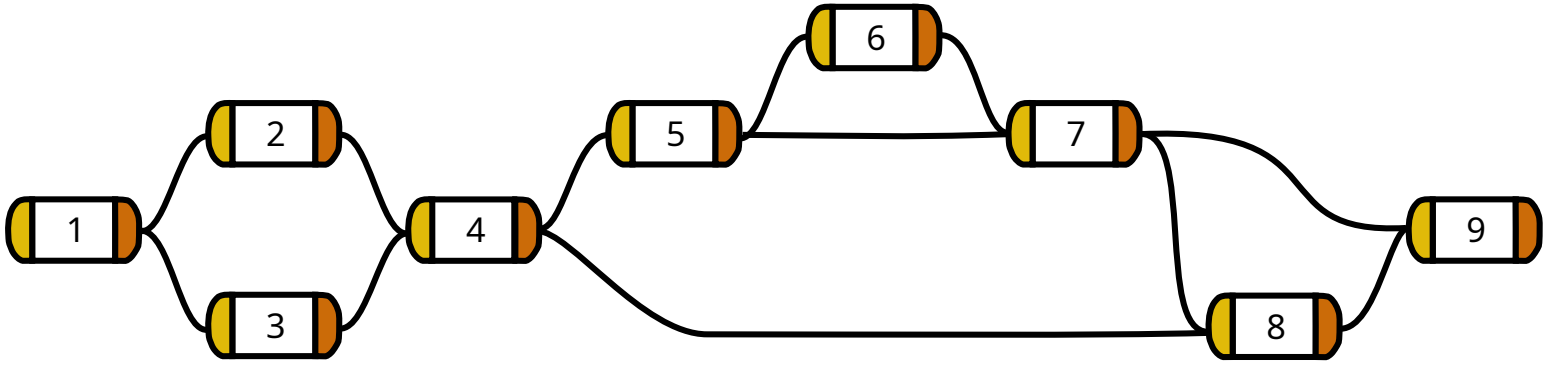


Snarls, intuitively a “bubble”



Snarls, formal definition

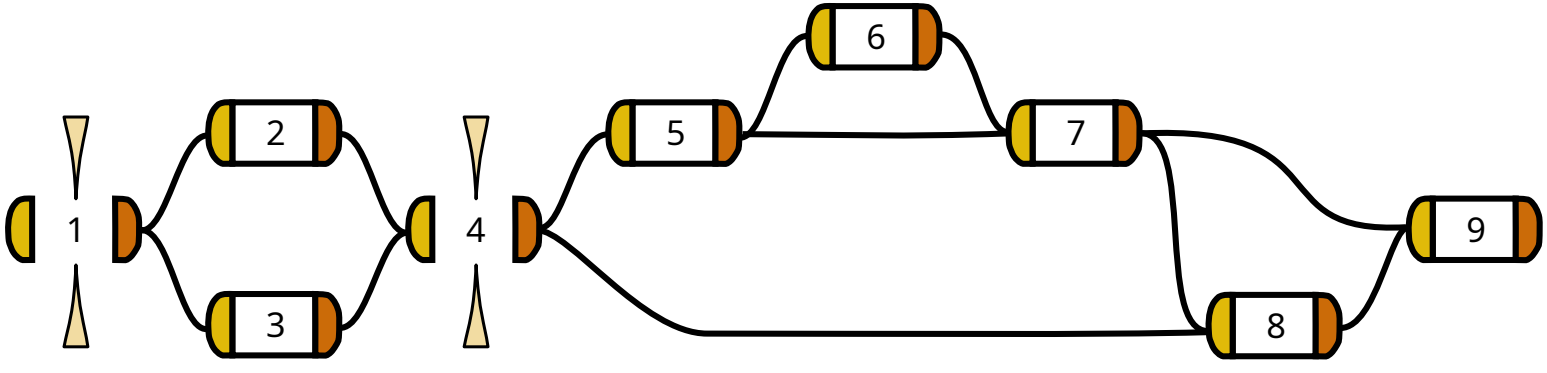
A snarl is a **subgraph** bounded by two **node sides** that are:



Snarls, formal definition

A snarl is a **subgraph** bounded by two **node sides** that are:

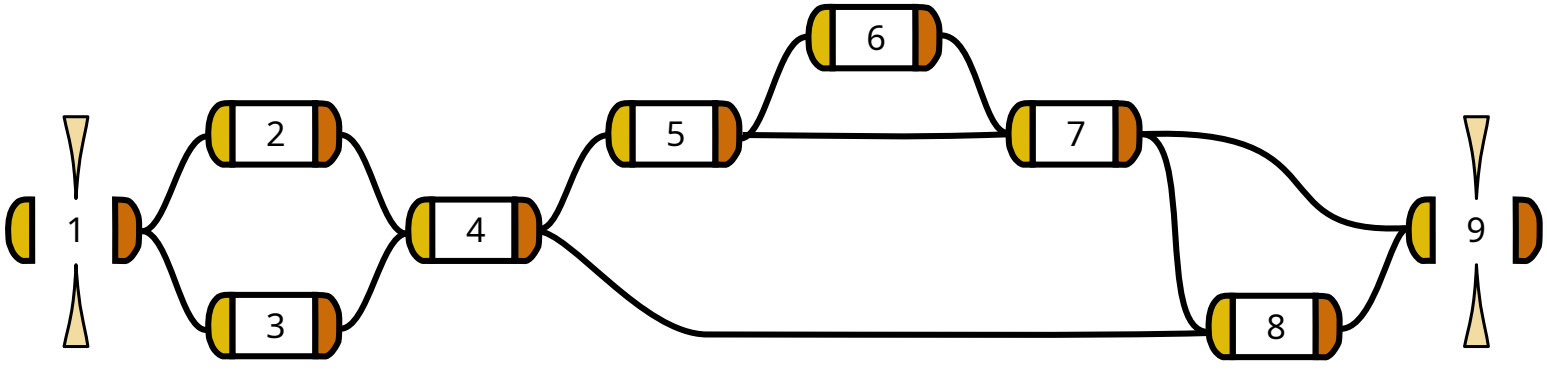
1. **Separable**: splitting the boundary nodes separates the snarl from the graph



Snarls, formal definition

A snarl is a **subgraph** bounded by two **node sides** that are:

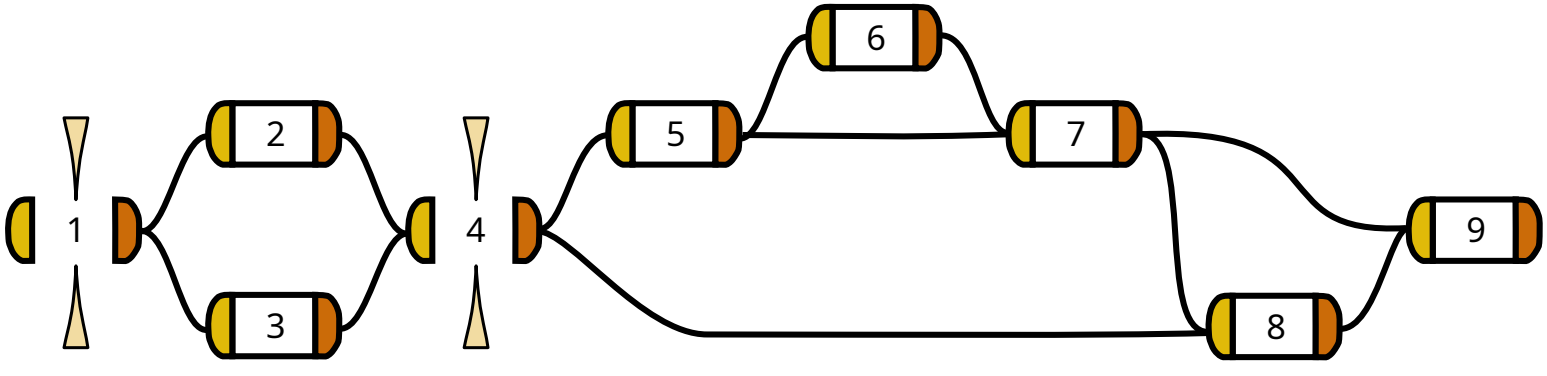
1. **Separable**: splitting the boundary nodes separates the snarl from the graph
2. **Minimal**: no nodes within the snarl are separable with either boundary node side



Snarls, formal definition

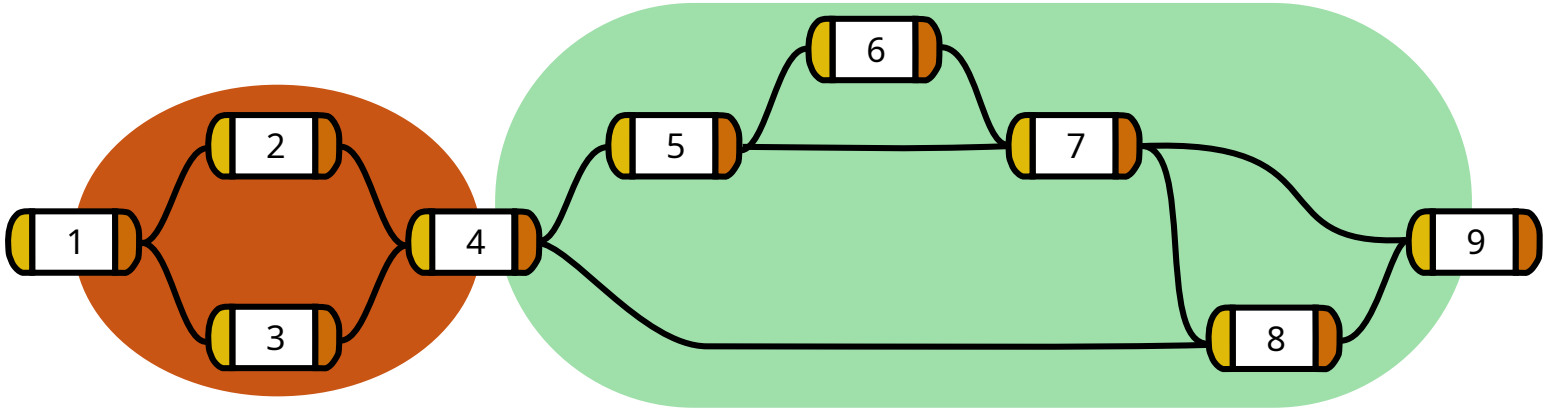
A snarl is a **subgraph** bounded by two **node sides** that are:

1. **Separable**: splitting the boundary nodes separates the snarl from the graph
2. **Minimal**: no nodes within the snarl are separable with either boundary node side



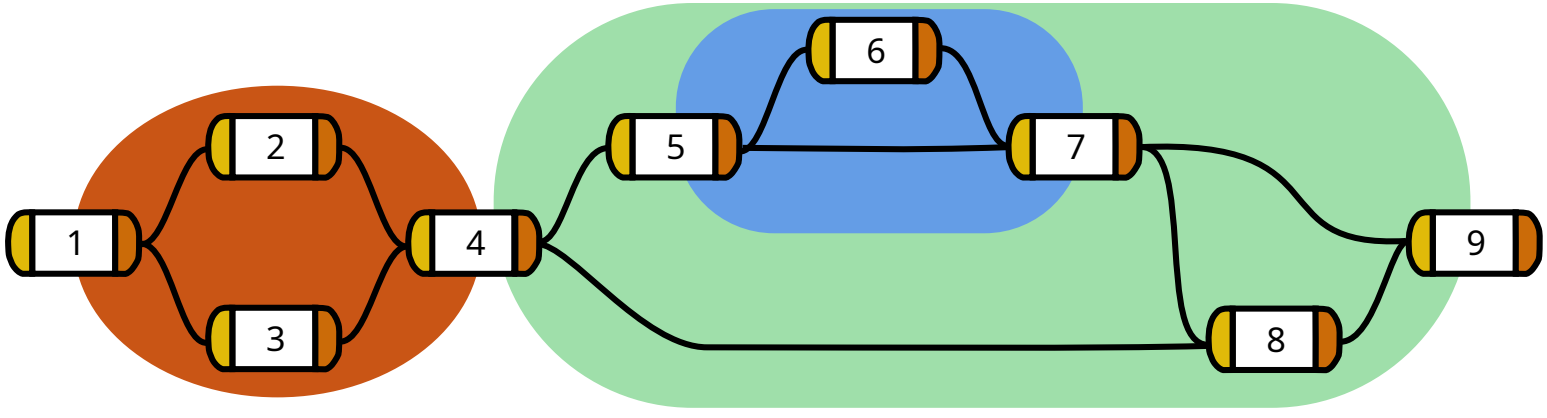
Chains

A run of consecutive snarls and nodes is called a **chain**.



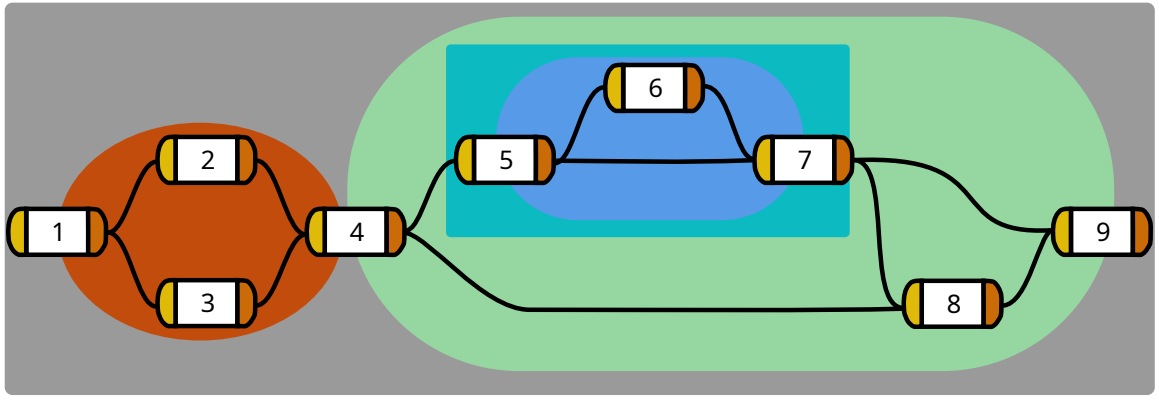
Snarl decomposition

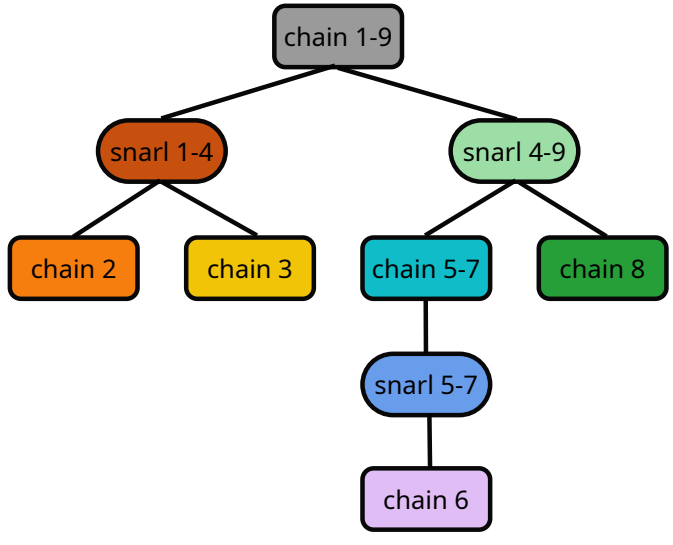
Snarls and chains can be **nested** inside of each other.



Snarls, chains, and nodes in a tree

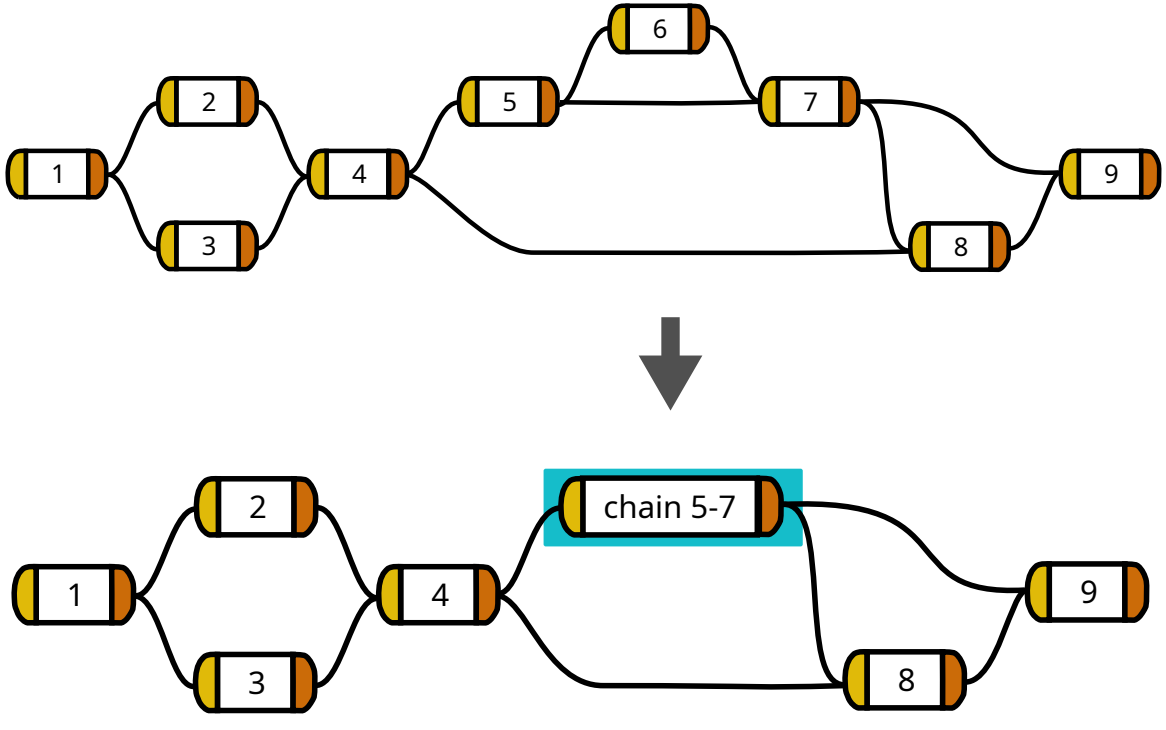
The nested relationship of snarls and chains is described by the **snarl tree**.

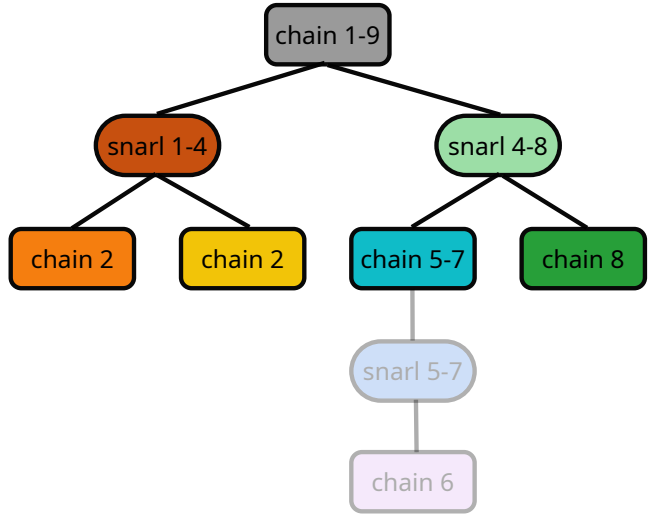




Netgraphs

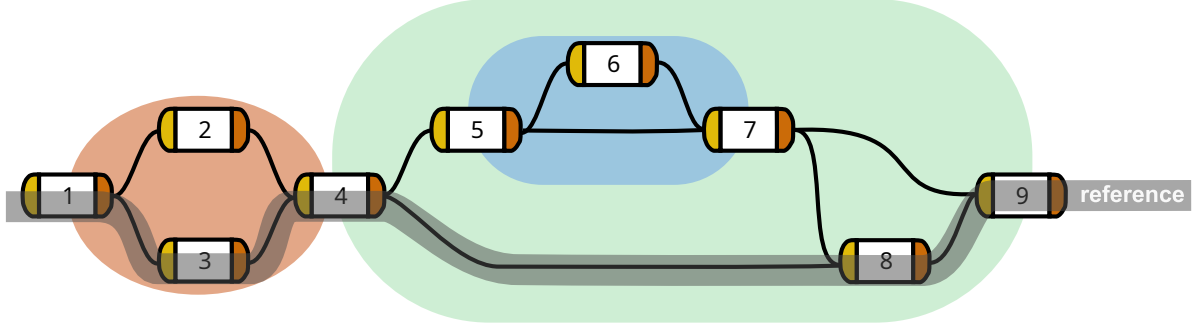
Netgraphs are a representation of snarls with their **child chains collapsed** into a single node

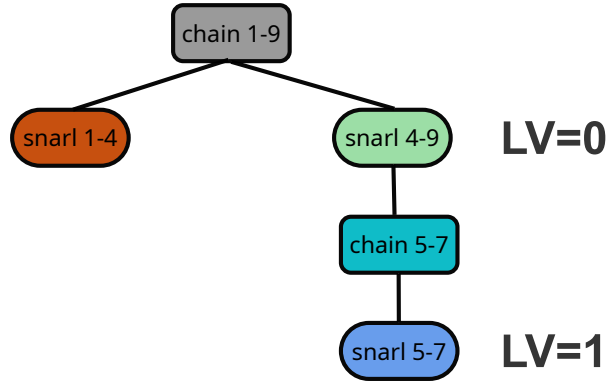




Deconstruct-ing a pangenome

Enumerate **alleles for each snarl** on **a reference path**. Before: all paths.

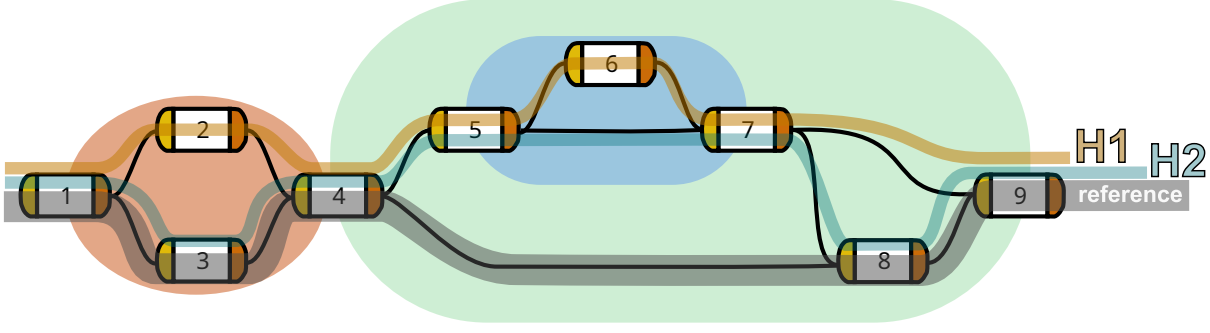


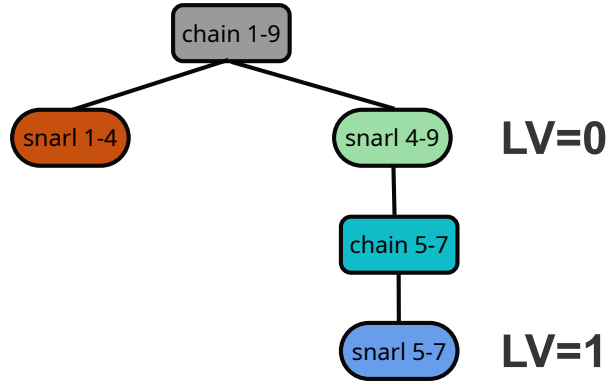


VARIANT	REF	ALT	LV
snarl 1-4	3	2	0
snarl 4-9	8	5 7 / 5 6 7 / 5 7 8 / 5 6 7 8	0

Deconstruct-ing a pangenome

Enumerate **alleles for each snarl** on **a reference path**. Now: only haplotypes.

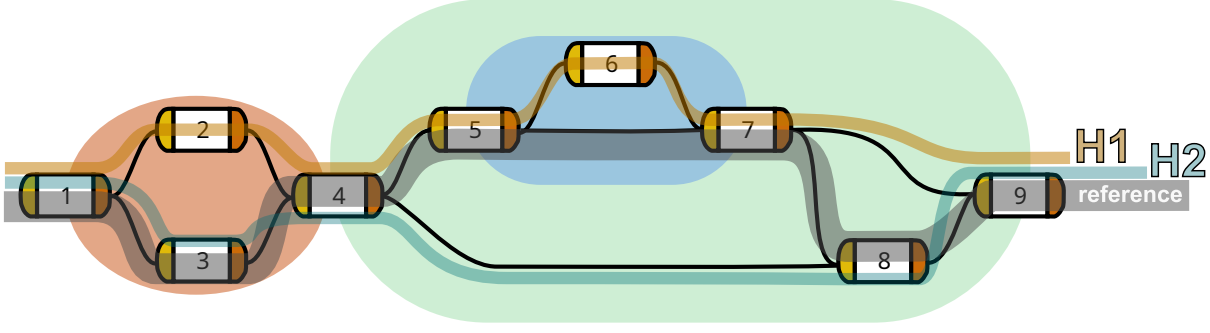


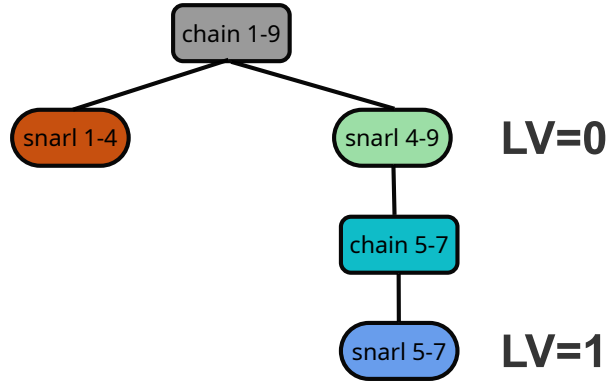


VARIANT	REF	ALT	LV	Genotype H1	Genotype H2
snarl 1-4	3	2	0	1	0
snarl 4-9	8	~~5 7~~ / 5 6 7 / 5 7 8 / 5 ~~6 7~~ 8	0	1	2

Deconstruct-ing a pangenome

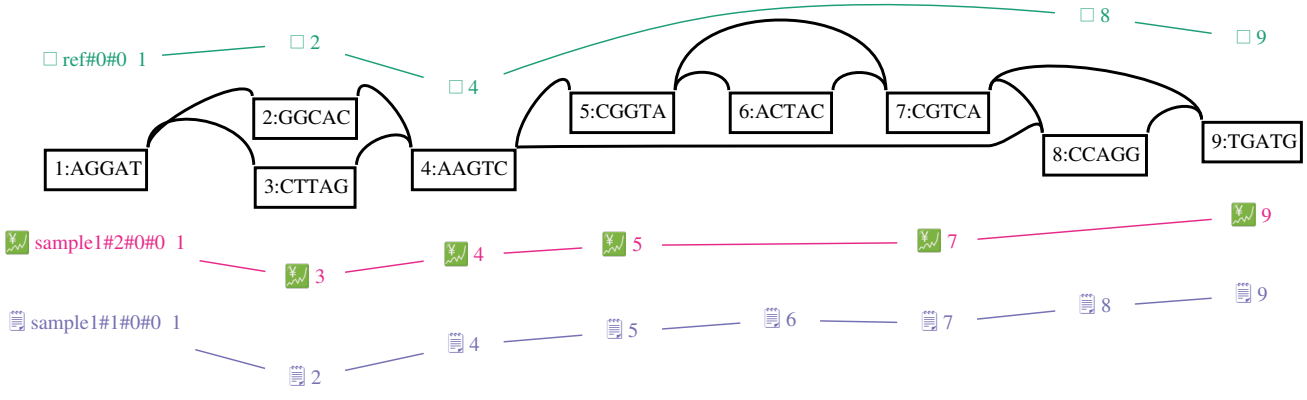
Enumerate **alleles for each snarl** on **a reference path** inc. nested snarls.



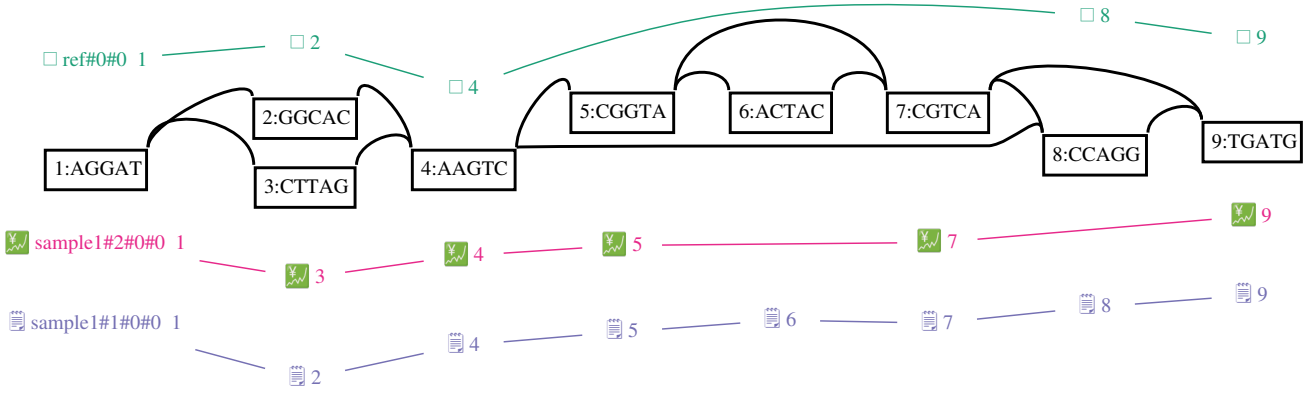


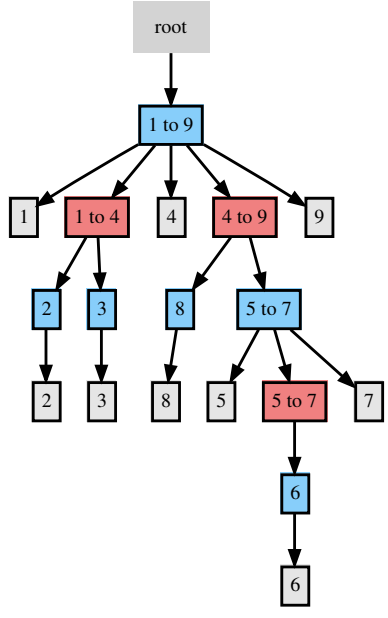
VARIANT	REF	ALT	LV	Genotype H1	Genotype H2
snarl 1-4	3	2	0	1	0
snarl 4-9	5 7 8	5 6 7 / 8	0	1	2
snarl 5-7	5	5 6	1	1	.

Snarl examples (`vg deconstruct`)



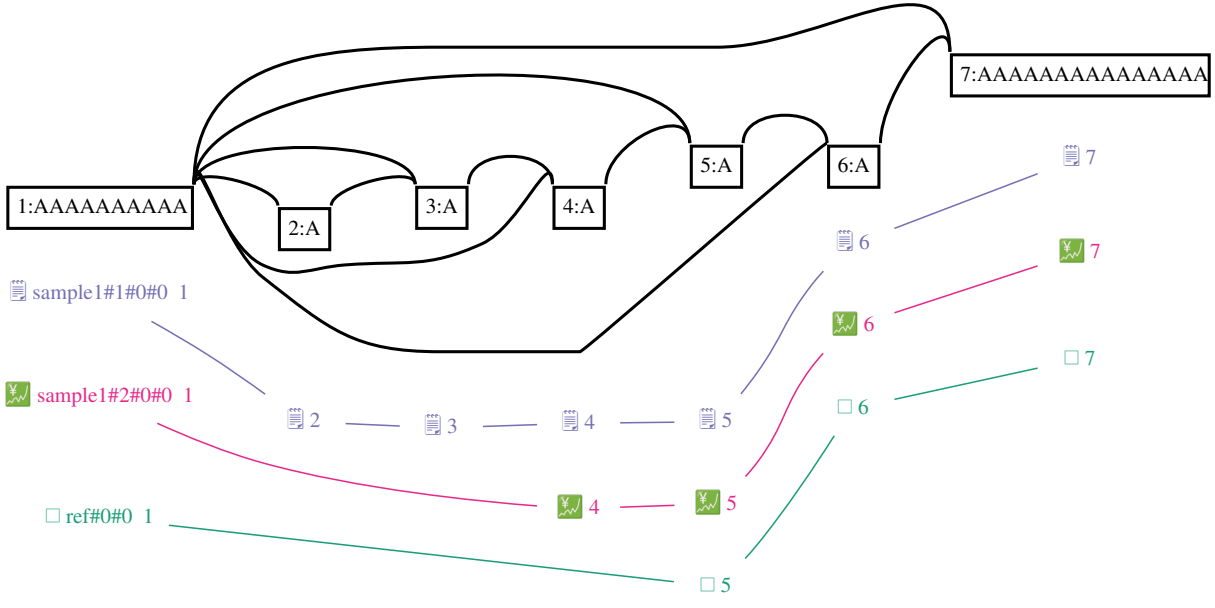
Snarl examples (`vg deconstruct`)



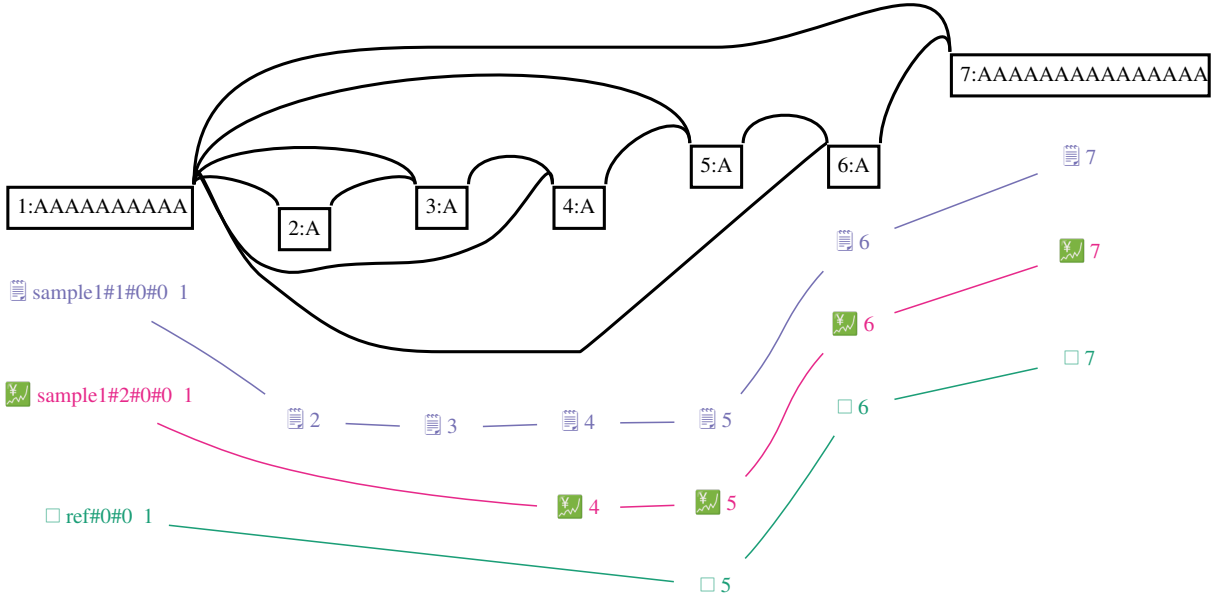


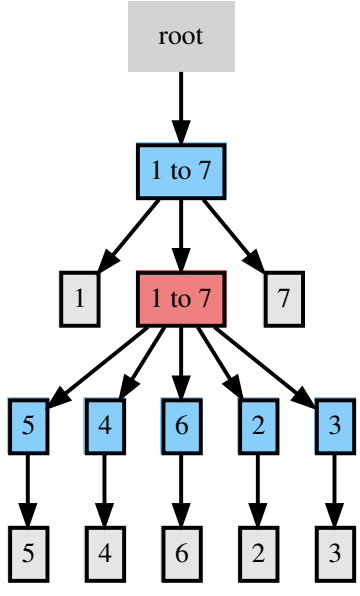

```
##INFO=<ID=LV,Number=1,Type=Integer,Description="Level in the snarl tree (0=top level)">
##INFO=<ID=AT,Number=R,Type=String,Description="Allele Traversal as path in graph">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  sample1
ref 6   >1>4    GGCAC   CTTAG   60  .   AT=>1>2>4,>1>3>4;LV=0   GT  0|1
ref 15  >4>9    CCCAGG  CCGGTAACTACCGTCACCAGG,CCGGTACGTCA   60  .   AT=>4>8>9,>4>5>6>7>8>9,>4>5>7>9;LV=0
GT  1|2
```

Snarl examples (`vg deconstruct`)



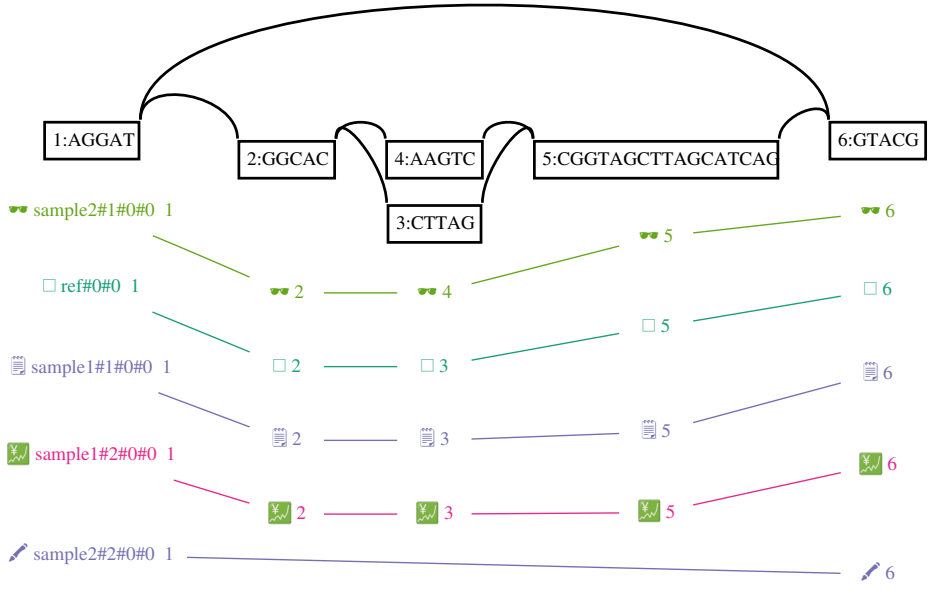
Snarl examples (`vg deconstruct`)



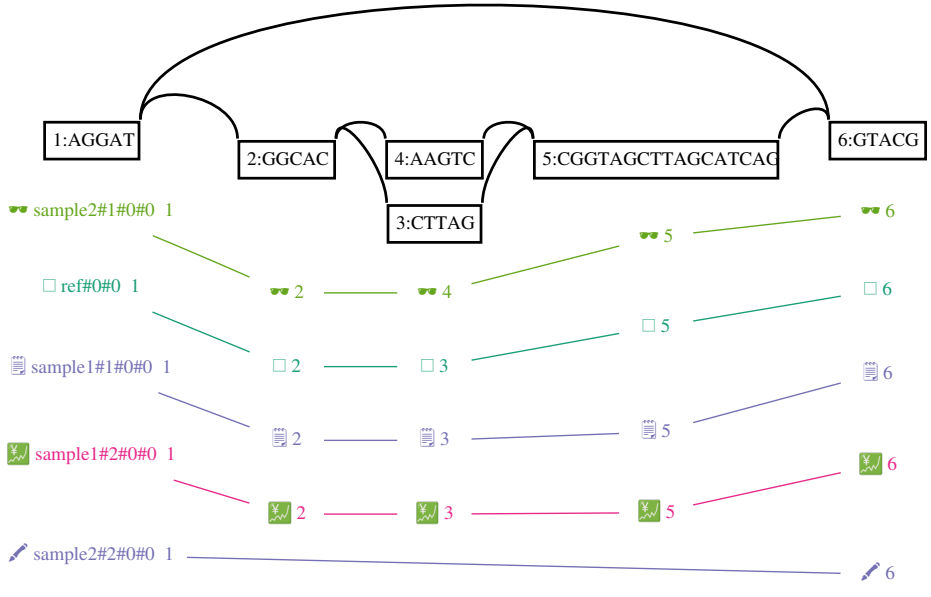


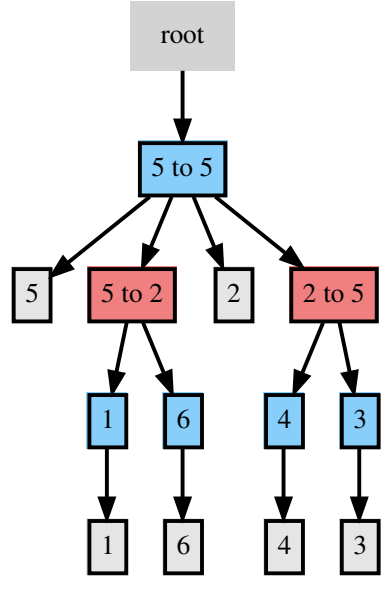

```
##INFO=<ID=LV,Number=1,Type=Integer,Description="Level in the snarl tree (0=top level)">
##INFO=<ID=AT,Number=R,Type=String,Description="Allele Traversal as path in graph">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  sample1
ref 10  >1>7    AAA AAAAAA,AAAA 60  .   AT=>1>5>6>7,>1>2>3>4>5>6>7,>1>4>5>6>7;LV=0  GT  1|2
```

Snarl (trick) examples (`vg deconstruct`)



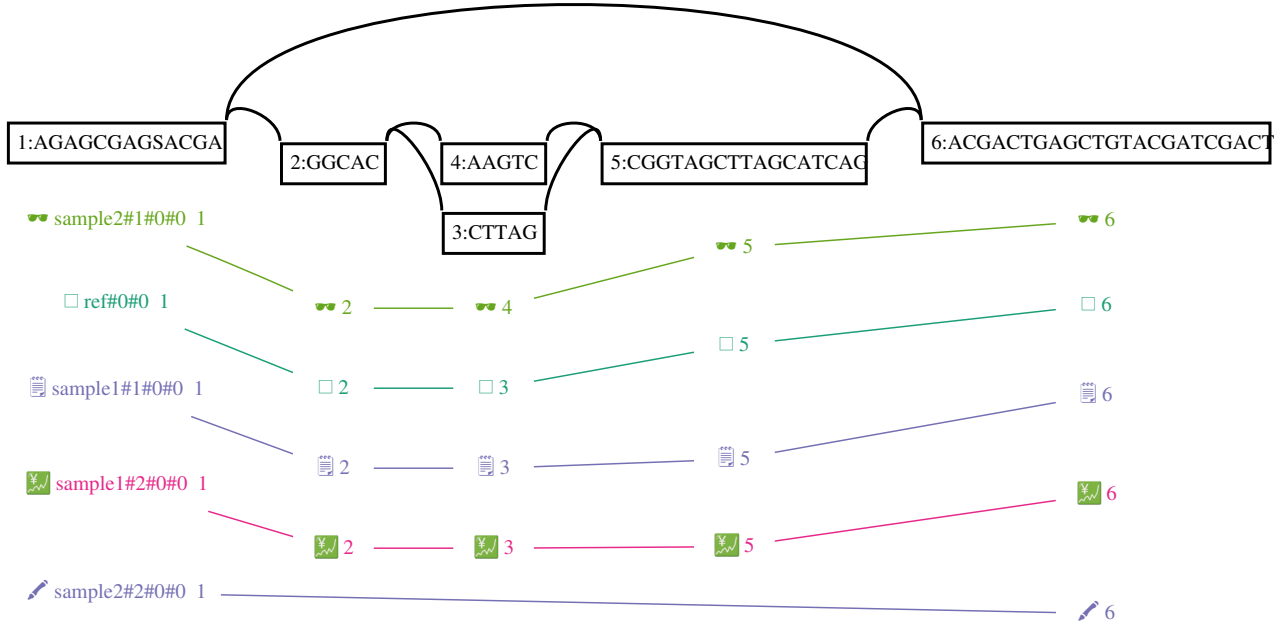
Snarl (trick) examples (`vg deconstruct`)

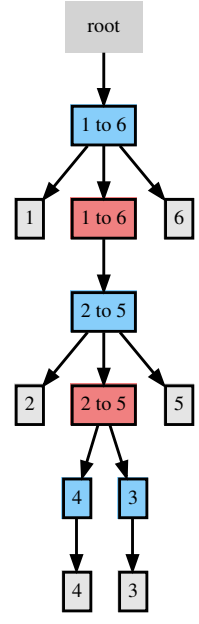





```
##INFO=<ID=LV,Number=1,Type=Integer,Description="Level in the snarl tree (0=top level)">
##INFO=<ID=AT,Number=R,Type=String,Description="Allele Traversal as path in graph">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  sample1 sample2
ref 11  >2>5    CTTAG   AAGTC   60  .   AT=>2>3>5,>2>4>5;LV=0   GT  0|0 1|.
```

Snarl examples (`vg deconstruct -a`)



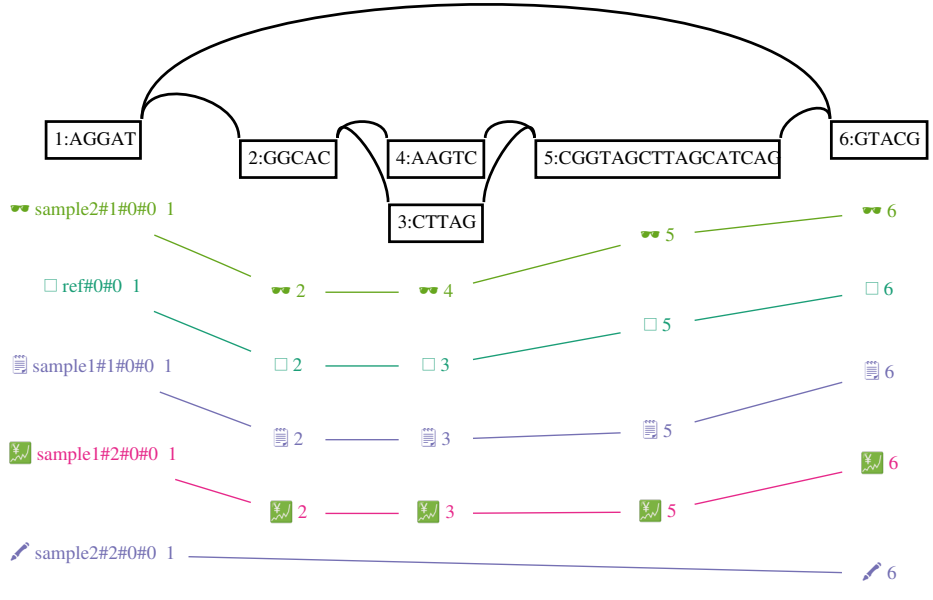


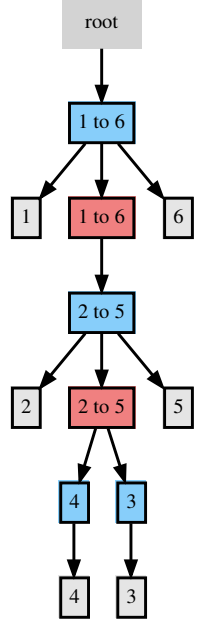

```
##INFO=<ID=LV,Number=1,Type=Integer,Description="Level in the snarl tree (0=top level)">
##INFO=<ID=PS,Number=1,Type=String,Description="ID of variant corresponding to parent snarl">
##INFO=<ID=AT,Number=R,Type=String,Description="Allele Traversal as path in graph">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  sample1 sample2
ref 13  >1>6    AGGCACCTTAGCGGTAGCTTAGCATCAG    AGGCACAAGTCCGGTAGCTTAGCATCAG,A  60  .
AT=>1>2>3>5>6,>1>2>4>5>6,>1>6;LV=0  GT  0|0 1|2
ref 19  >2>5    CTTAG   AAGTC   60  .   AT=>2>3>5,>2>4>5;LV=1;PS=>1>6   GT  0|0 1|.
```

Snarl examples

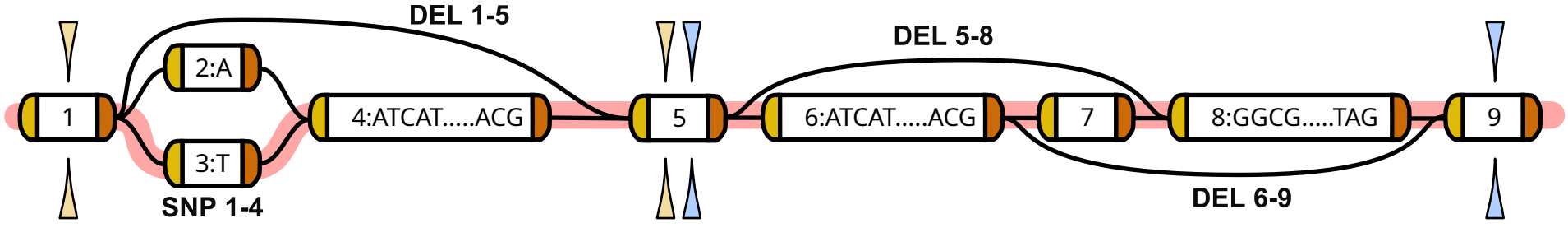
Trick for getting this snarl decomposition to look better (currently only for the distance index):

```
vg index -j [graph.dist] -w 6
```



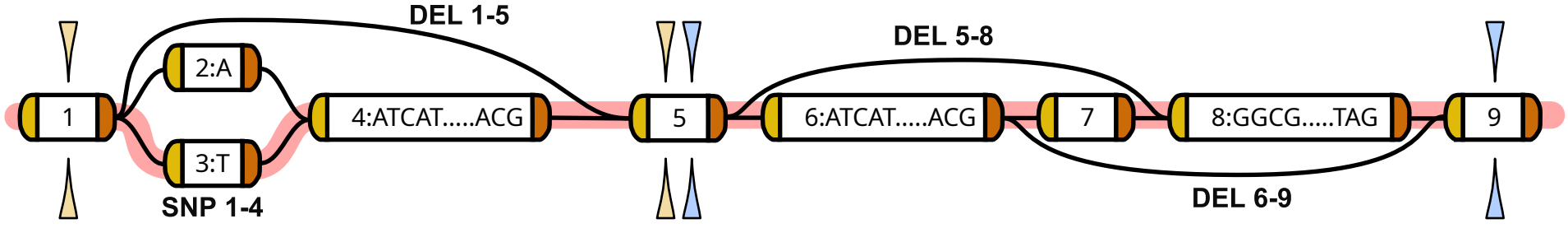


Snarl decomposition limitations



	Snarl 1-5	Snarl 5-9
REF	>1>3>4>5	>5>6>7>8>9
ALT1	>1>2>4>5	>5>8>9
ALT2	>1>5	>5>6>9

Snarl decomposition limitations



	Snarl 1-5	Snarl 5-9
REF	>1>3>4>5	>5>6>7>8>9
ALT1	>1>2>4>5	>5>8>9
ALT2	>1>5	>5>6>9

Solutions?

1. `vcfbub` keeps top-level snarls, then `vcfwave` aligns REF vs ALT(s) sequence to enumerate variants. Warning: inconsistent with the original pangenome.
2. in-house scripts that compares REF path vs ALT(s) paths (e.g. Pangenie).

Mapping reads with `vg giraffe`

Short reads

RESEARCH ARTICLE

GENOMICS

Pangenomics enables genotyping of known structural variants in 5202 diverse genomes

Jouni Sirén[1]†, Jean Monlong[1]†, Xian Chang[1]†, Adam M. Novak[1]†, Jordan M. Eizenga[1]†, Charles Markello[1], Jonas A. Sibbesen[1], Glenn Hickey[1], Pi-Chuan Chang[2], Andrew Carroll[2], Namrata Gupta[3], Stacey Gabriel[4], Thomas W. Blackwell[5], Aakrosh Ratan[6], Kent D. Taylor[7], Stephen S. Rich[6], Jerome I. Rotter[7], David Haussler[1,8], Erik Garrison[9], Benedict Paten[1]*

We introduce Giraffe, a pangenome short-read mapper that can efficiently map to a collection of haplotypes threaded through a sequence graph. Giraffe maps sequencing reads to thousands of human genomes at a speed comparable to that of standard methods mapping to a single reference genome. The increased mapping accuracy enables downstream improvements in genome-wide genotyping pipelines for both small variants and larger structural variants. We used Giraffe to genotype 167,000 structural variants, discovered in long-read studies, in 5202 diverse human genomes that were sequenced using short reads. We conclude that pangenomics facilitates a more comprehensive characterization of variation and, as a result, has the potential to improve many genomic analyses.

Long reads

bioRxiv

THE PREPRINT SERVER FOR BIOLOGY

HOME |

New Results

Follow this preprint

Rapid, accurate long- and short-read mapping to large pangenome graphs with vg Giraffe

Xian Chang, Adam M. Novak, Jordan M. Eizenga, Jouni Sirén, Jean Monlong, Shloka Negi, Francesco Andreace, Sagorika Nag, Konstantinos Kyriakidis, Glenn Hickey, Stephen Hwang, Emmanuèle C. Délot, Andrew Carroll, Kishwar Shafin, Pi-Chuan Chang, Faith Okamoto, Benedict Paten, the Human Pangenome Reference Consortium

doi: https://doi.org/10.1101/2025.09.29.678807

Read mapping algorithm overview

Input: Sequencing read and reference sequence

Read

GATACTAACGTA

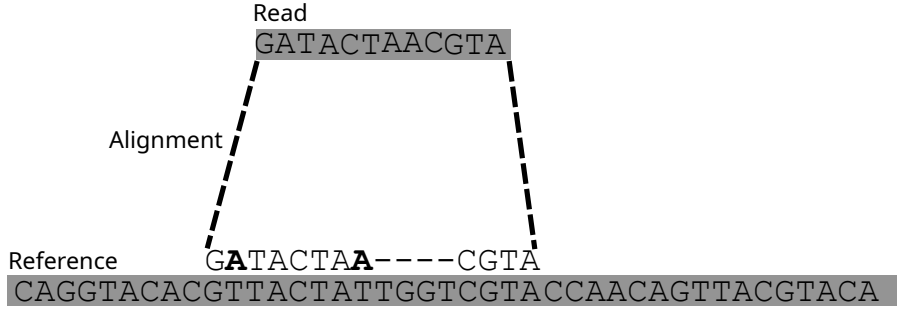
Reference

CAGGTACACGTTACTATTGGTCGTACCAACAGTTACGTACA

Read mapping algorithm overview

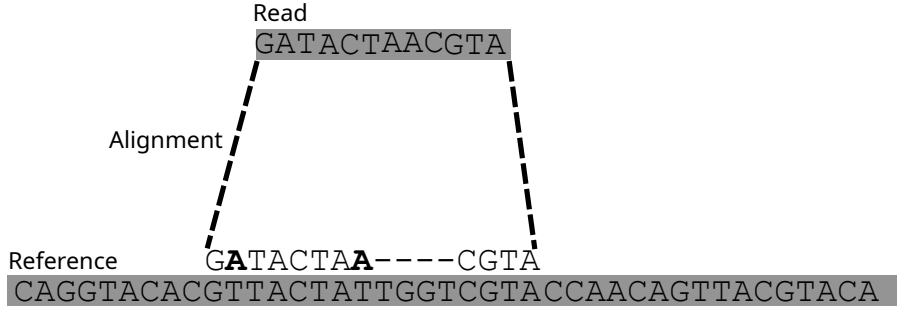
Input: Sequencing read and reference sequence

Output: Placement of the read on the reference and, usually, the edits between the sequences



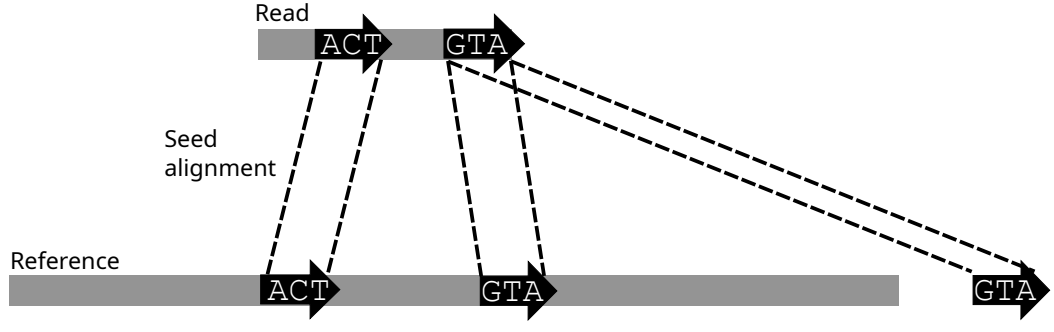
Read mapping algorithm overview

1. Seeding
2. Clustering/ chaining
3. Alignment



Read mapping algorithm overview

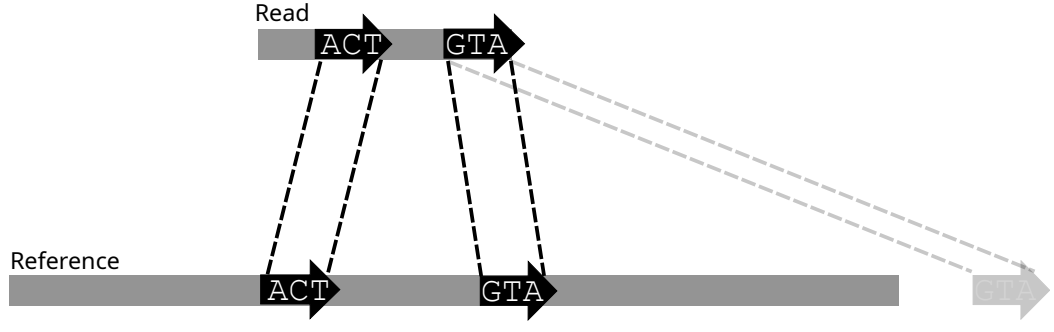
1. **Seeding**
2. Clustering/ chaining
3. Alignment



Find seeds (short, exact matches) between the read and reference using an index

Read mapping algorithm overview

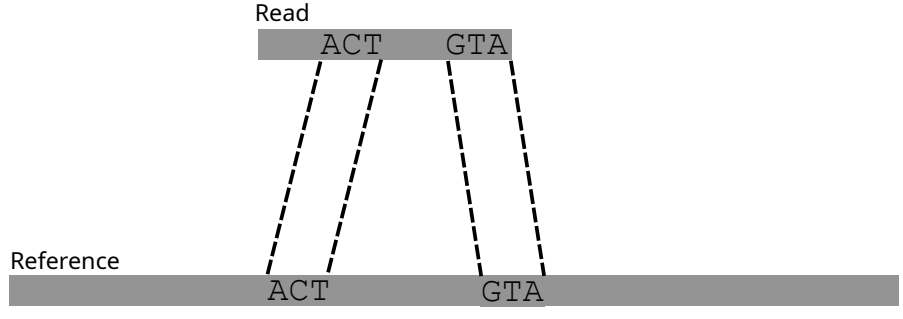
1. Seeding
2. **Clustering/ chaining**
3. Alignment



Find groups of seeds that may belong to the same alignment

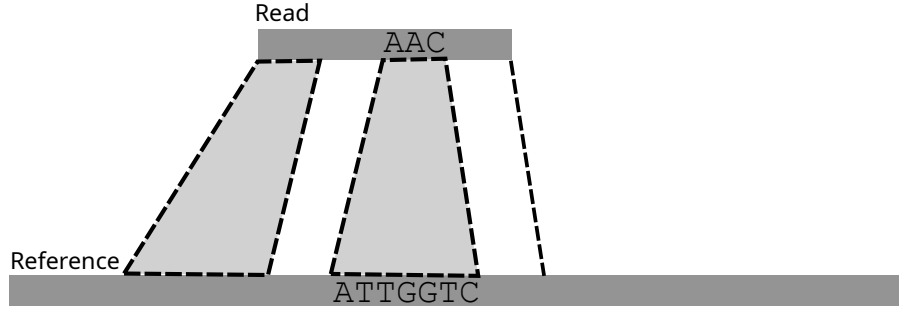
Read mapping algorithm overview

1. Seeding
2. **Clustering/ chaining**
3. Alignment



Read mapping algorithm overview

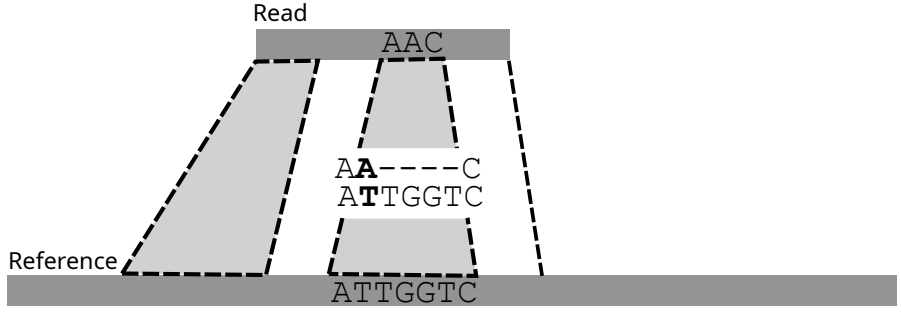
1. Seeding
2. Clustering/ chaining
3. **Alignment**



Extend the seed alignment using an alignment algorithm

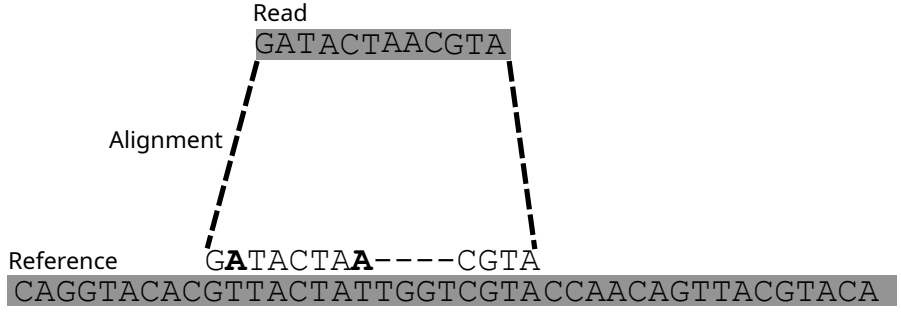
Read mapping algorithm overview

1. Seeding
2. Clustering/ chaining
3. **Alignment**



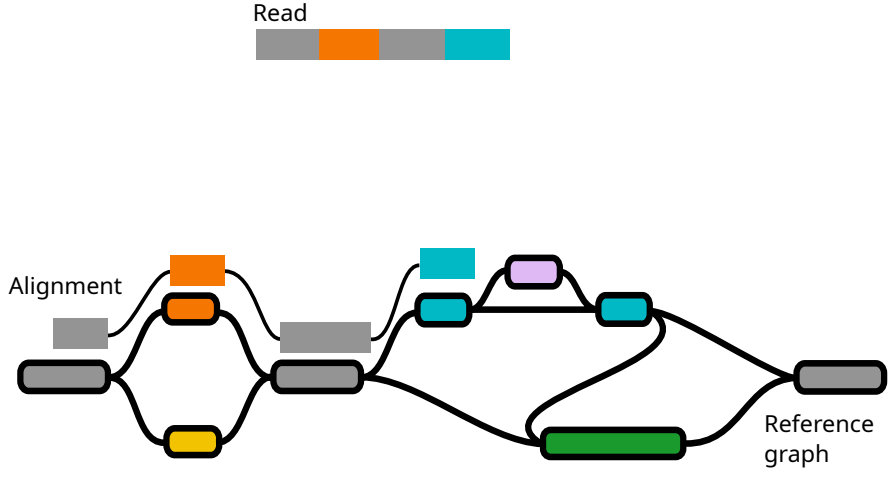
Read mapping algorithm overview

1. Seeding
2. Clustering/ chaining
3. Alignment



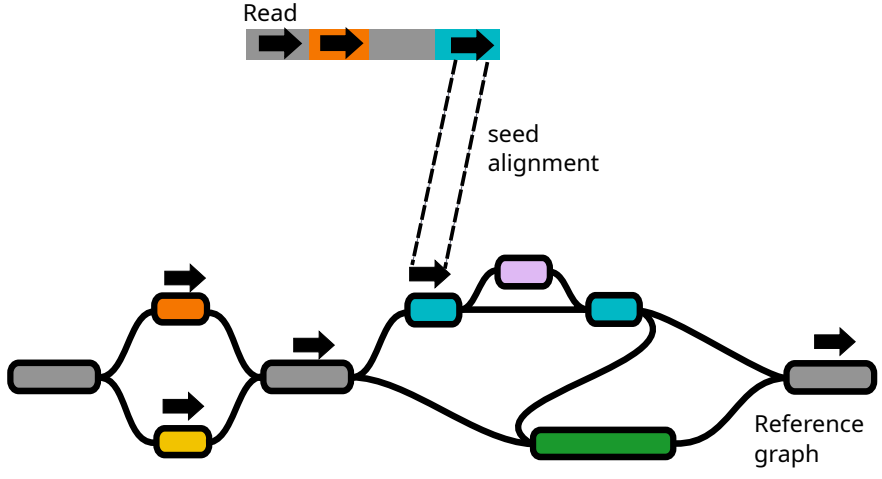
Short read giraffe algorithm

1. Seeding
2. Clustering/ chaining
3. Alignment



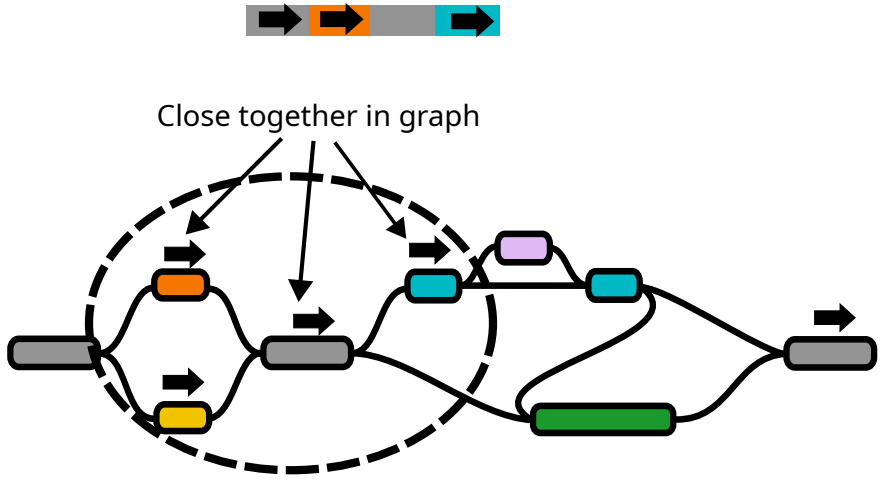
Short read giraffe algorithm

1. **Seeding** with **Minimizer Index**
2. Clustering/ chaining
3. Alignment



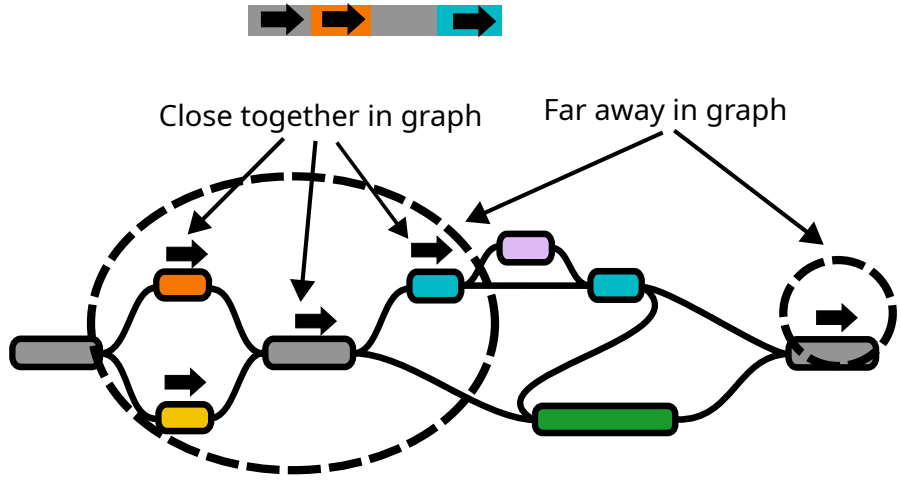
Short read giraffe algorithm

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment



Short read giraffe algorithm

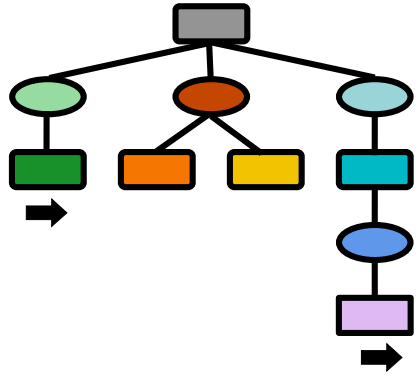
1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

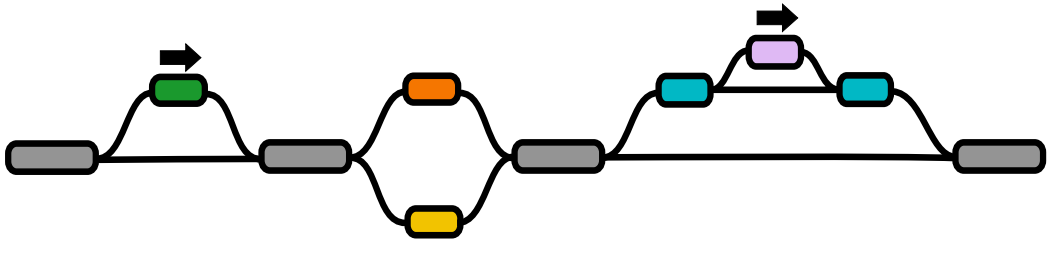


Short read giraffe algorithm (Distance index)

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

Distance index: used to find minimum distances using the snarl decomposition

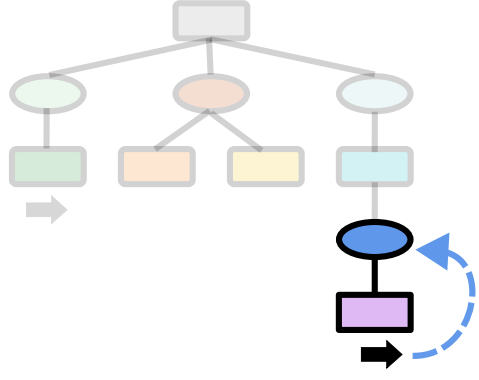


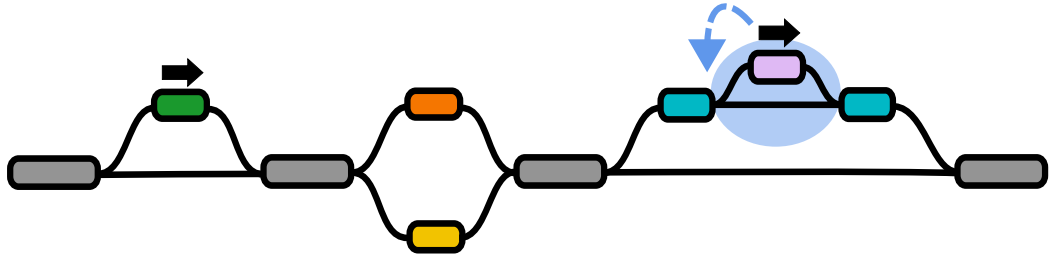


Short read giraffe algorithm (Distance index)

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

At each ancestor snarl and chain, find the minimum distance from the child to the boundary node

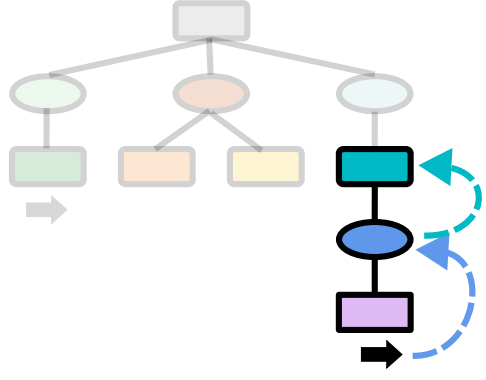


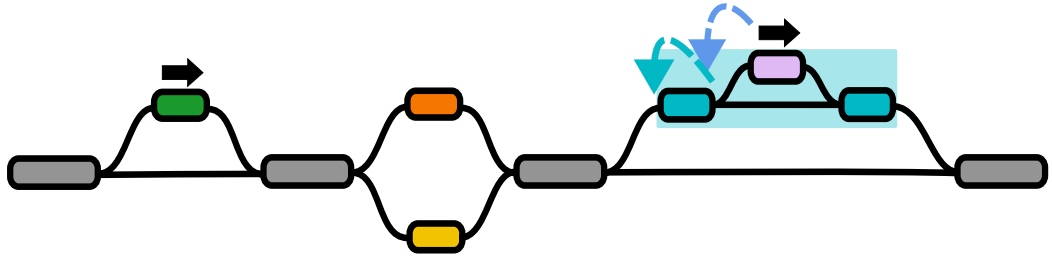


Short read giraffe algorithm (Distance index)

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

At each ancestor snarl and chain, find the minimum distance from the child to the boundary node

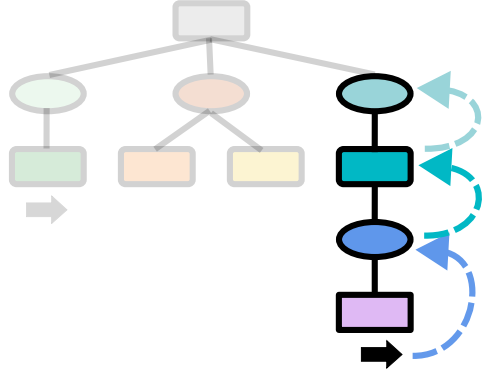


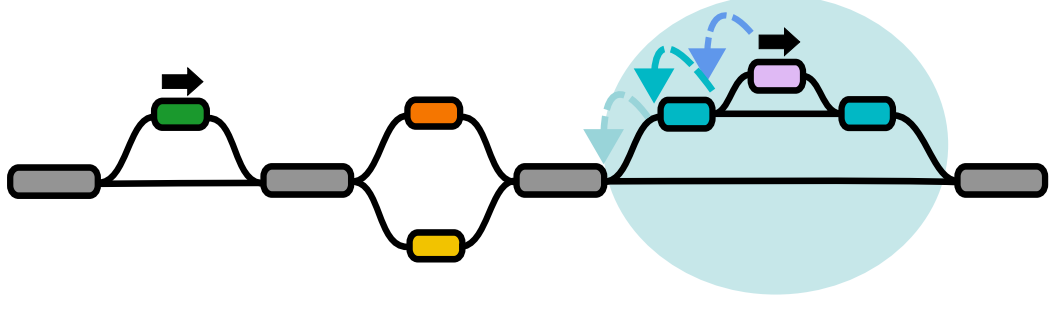


Short read giraffe algorithm (Distance index)

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

At each ancestor snarl and chain, find the minimum distance from the child to the boundary node

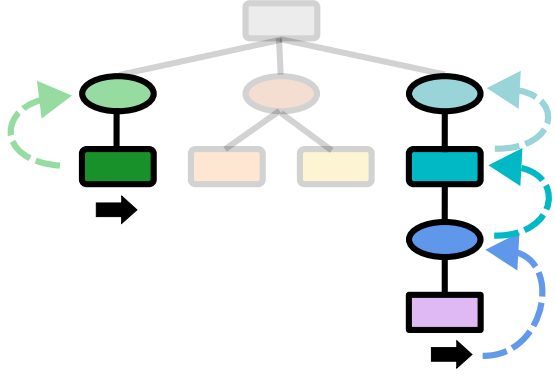


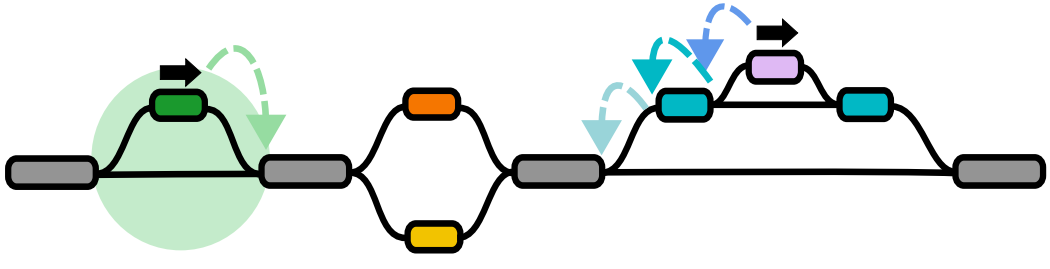


Short read giraffe algorithm (Distance index)

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

At each ancestor snarl and chain, find the minimum distance from the child to the boundary node



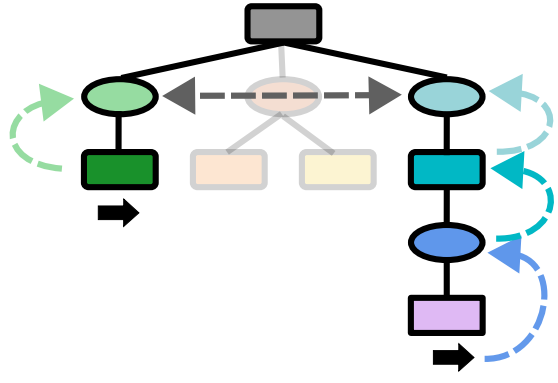


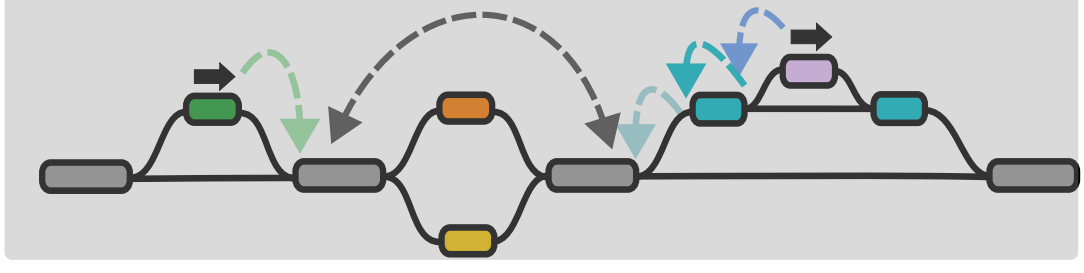
Short read giraffe algorithm (Distance index)

1. Seeding
2. **Clustering** with **Distance Index**
3. Alignment

At each ancestor snarl and chain, find the minimum distance from the child to the boundary node

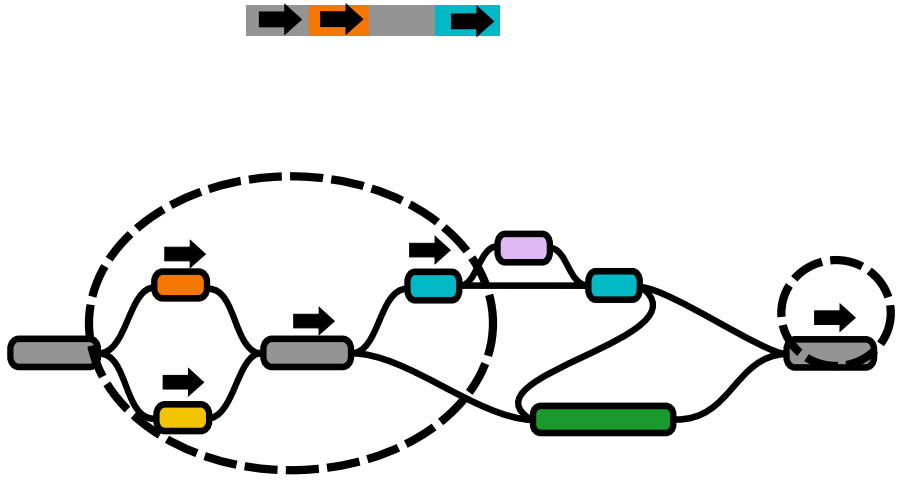
At the common ancestor, find the distance between children





Short read giraffe algorithm

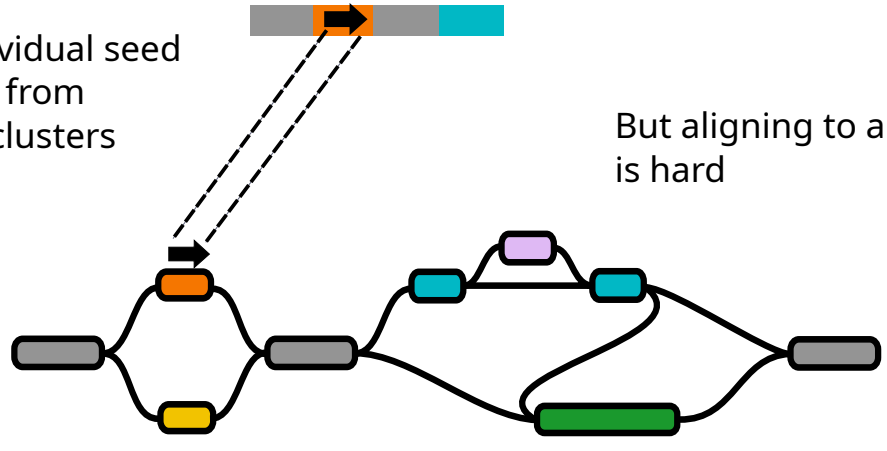
1. Seeding
2. Clustering
3. **Alignment**



Short read giraffe algorithm

1. Seeding
2. Clustering
3. **Alignment**

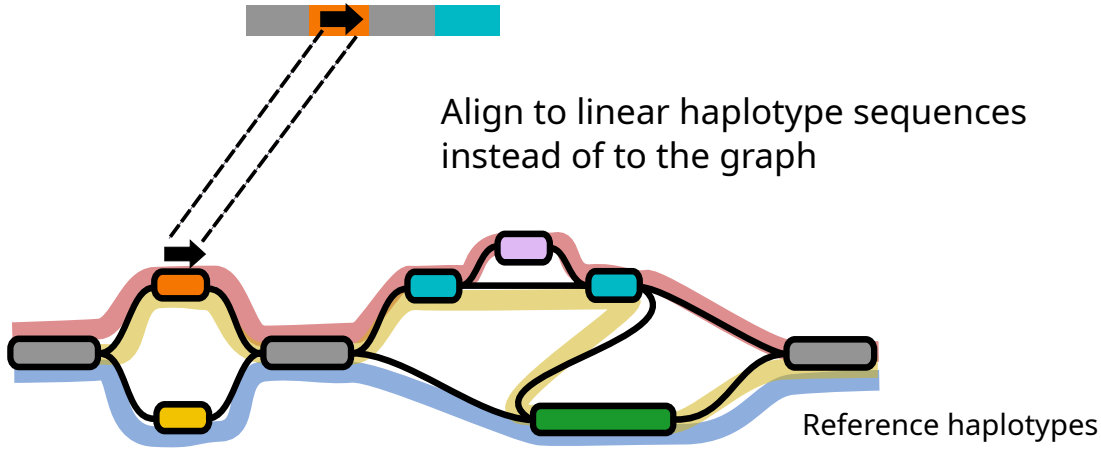
Extend individual seed alignments from promising clusters



But aligning to a graph is hard

Short read giraffe algorithm

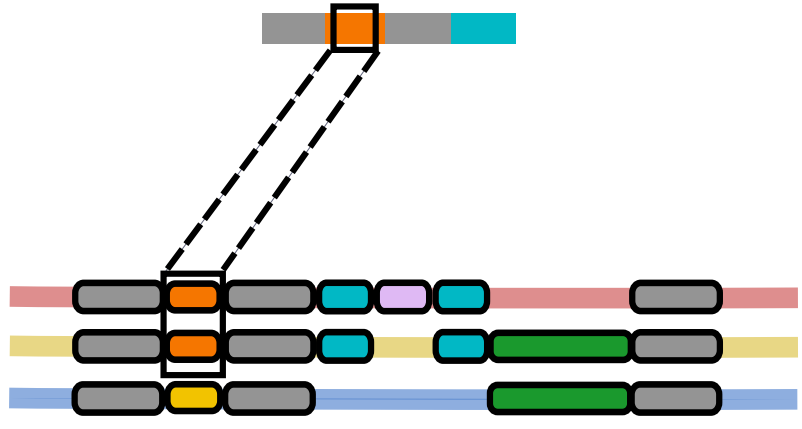
1. Seeding
2. Clustering
3. **Gapless extension**
4. Gapped alignment



Short read giraffe algorithm

1. Seeding
2. Clustering
3. **Gapless extension** with **GBWT**
4. Gapped alignment

Extend the alignment without allowing gaps



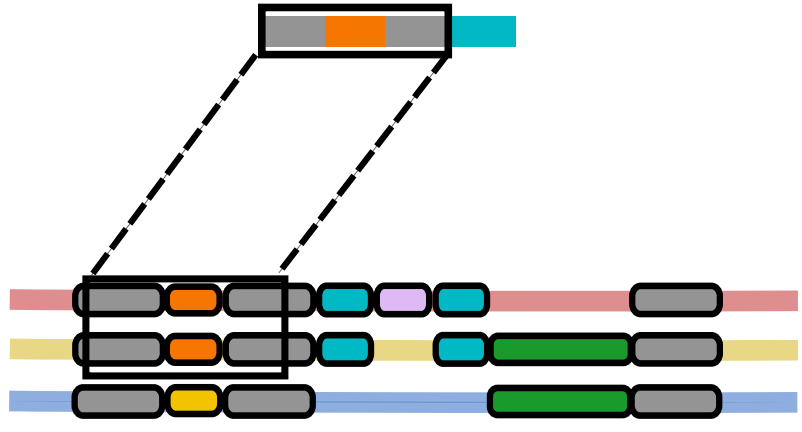
Reference haplotypes

Short read giraffe algorithm

1. Seeding
2. Clustering
3. **Gapless extension** with **GBWT**
4. Gapped alignment

Extend the alignment without allowing gaps

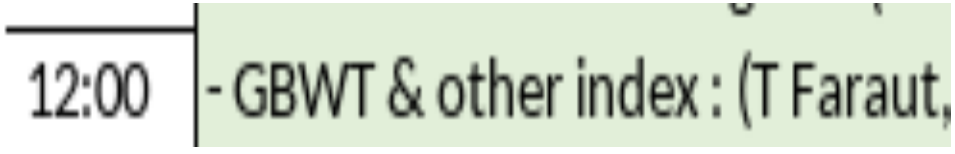
Gapless extension is fast and usually finds the whole length of the read

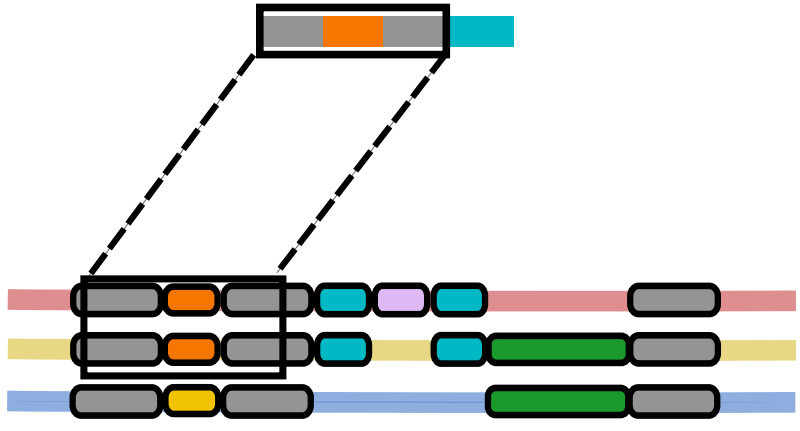


Reference haplotypes

Short read giraffe algorithm

1. Seeding
2. Clustering
3. **Gapless extension** with **GBWT**
4. Gapped alignment

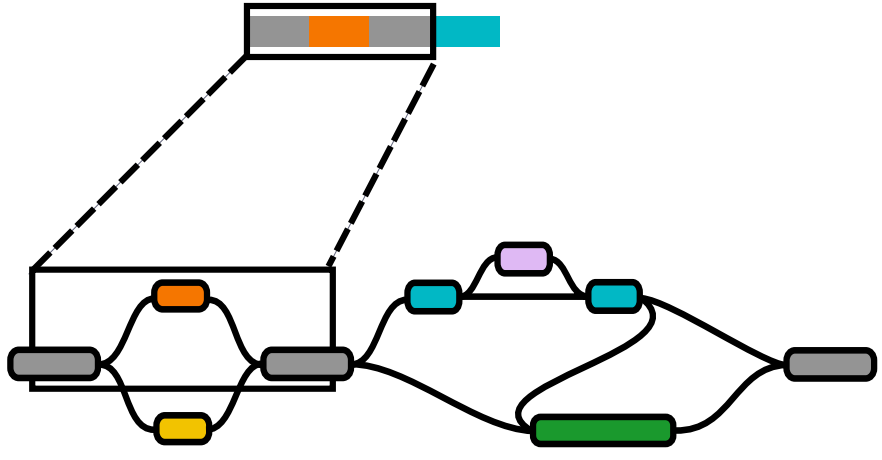




Reference haplotypes

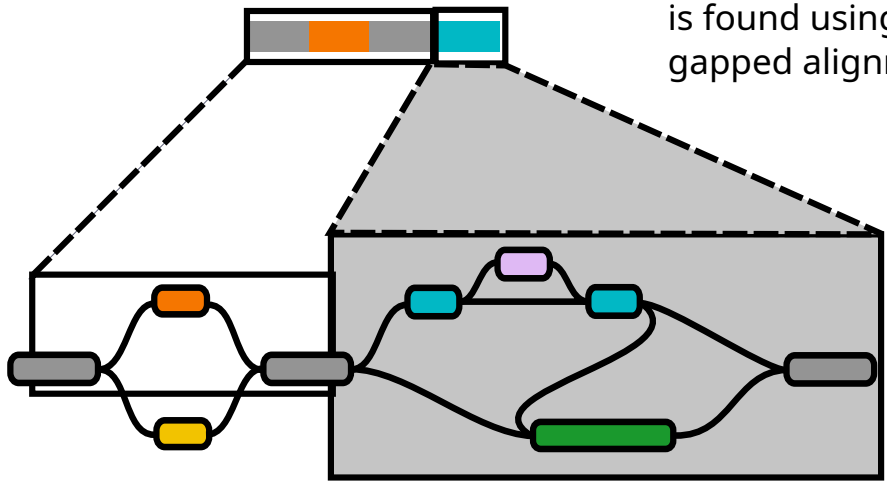
Short read giraffe algorithm

1. Seeding
2. Clustering
3. Gapless extension
4. **Gapped alignment** with graph



Short read giraffe algorithm

1. Seeding
2. Clustering
3. Gapless extension
4. **Gapped alignment** with graph

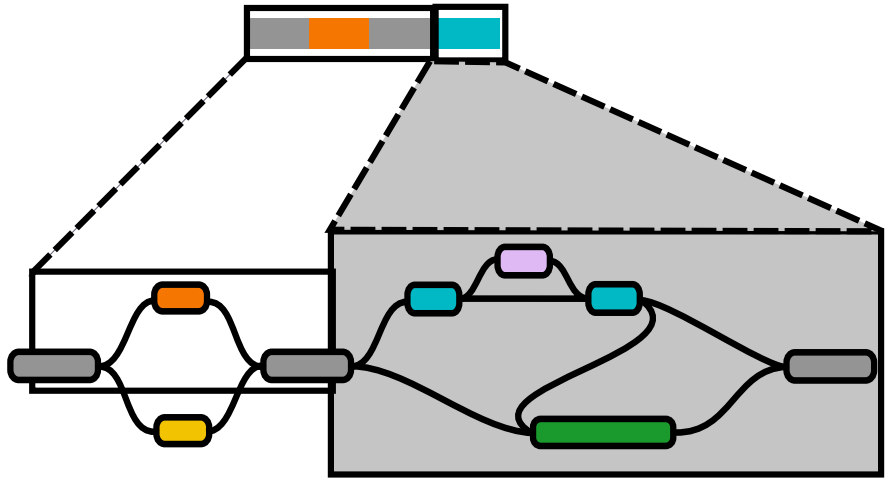


The remainder of the read is found using gapped alignment

Short read giraffe algorithm

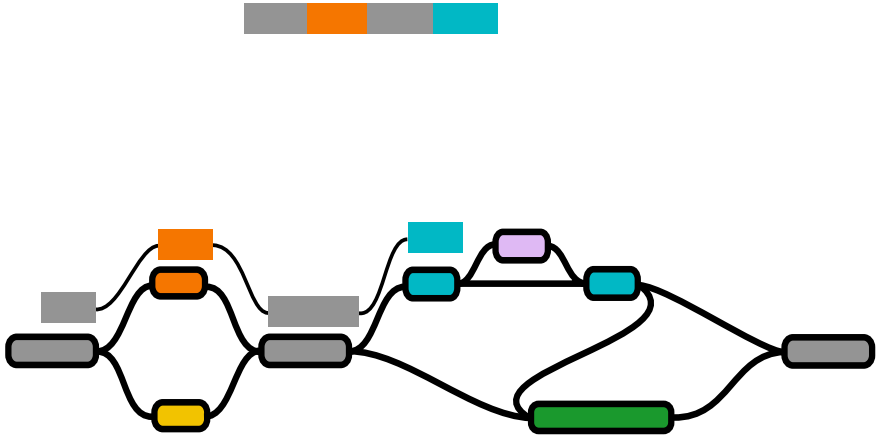
1. Seeding
2. Clustering
3. Gapless extension
4. **Gapped alignment** with graph

11:00	coffee break
	- Cactus & Wfmash : aligners (B. Linard)



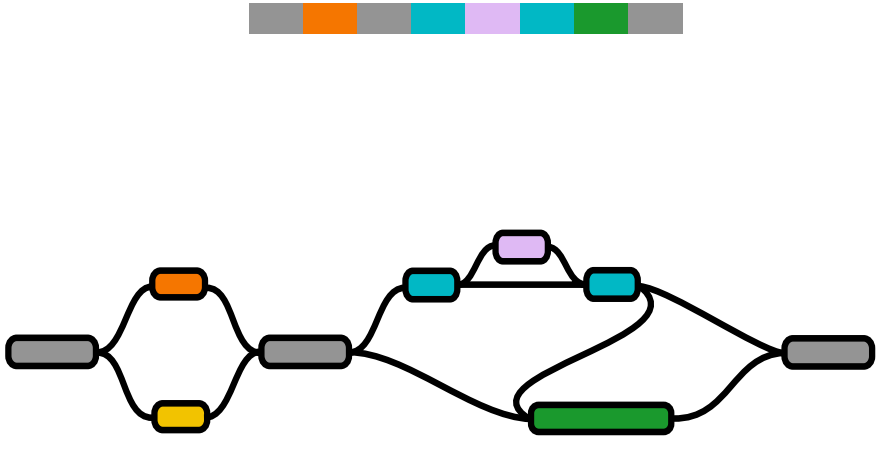
Short read giraffe algorithm

1. Seeding
2. Clustering
3. Gapless extension
4. Gapped alignment



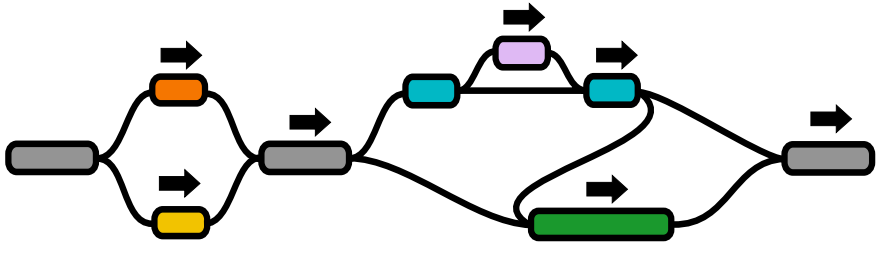
Long read giraffe algorithm

1. Seeding
2. Clustering/ chaining
3. Alignment



Long read giraffe algorithm

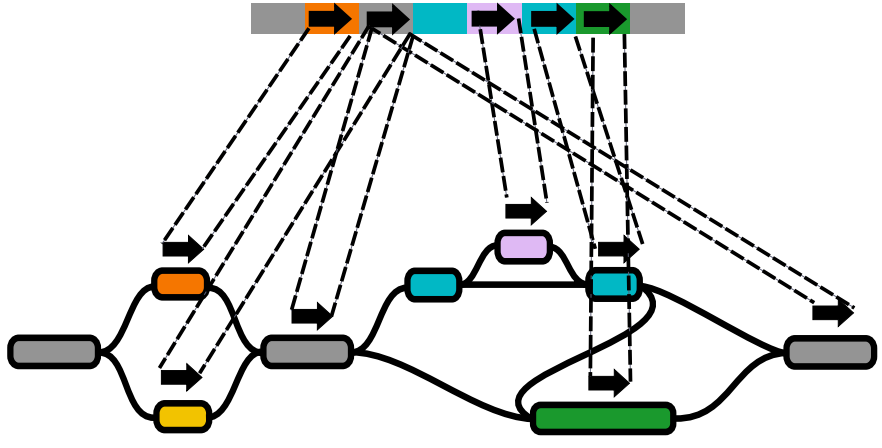
1. **Seeding** with **Minimizer Index**
2. Clustering/ chaining
3. Alignment



Long read giraffe algorithm

1. Seeding
2. **Chaining**
3. Alignment

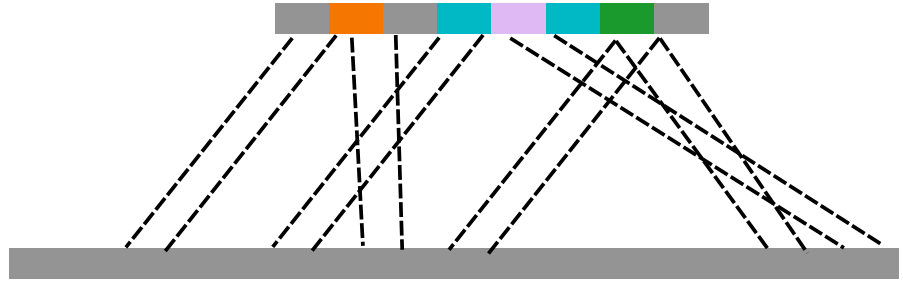
For long reads we do chaining instead of clustering



Long read giraffe algorithm

1. Seeding
2. **Chaining**
3. Alignment

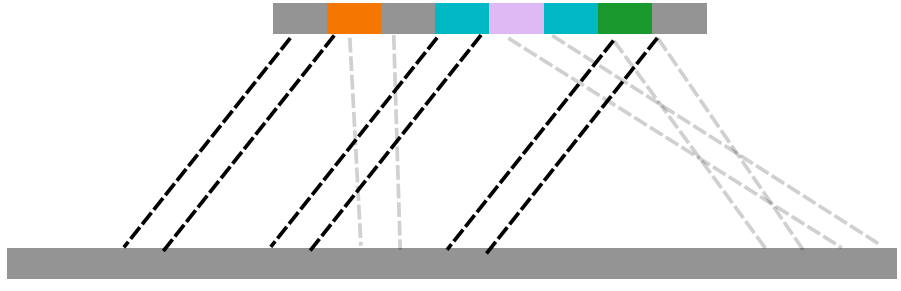
Seeds in a chain must be co-linear
(same order in read and reference)



Long read giraffe algorithm

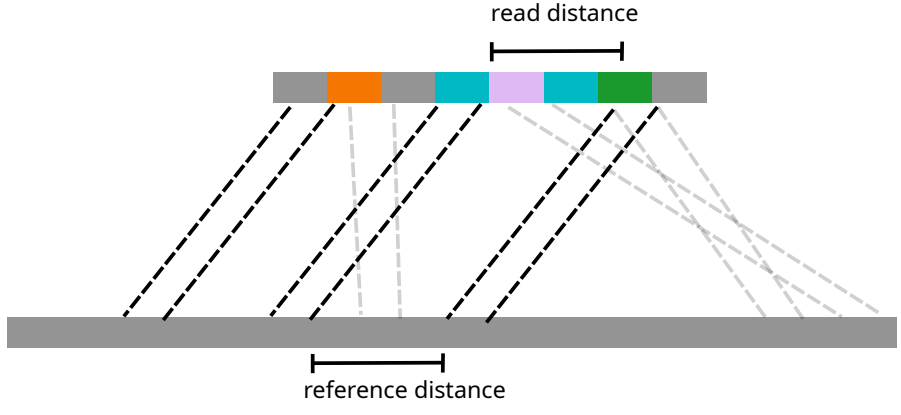
1. Seeding
2. **Chaining**
3. Alignment

Seeds in a chain must be co-linear
(same order in read and reference)



Long read giraffe algorithm

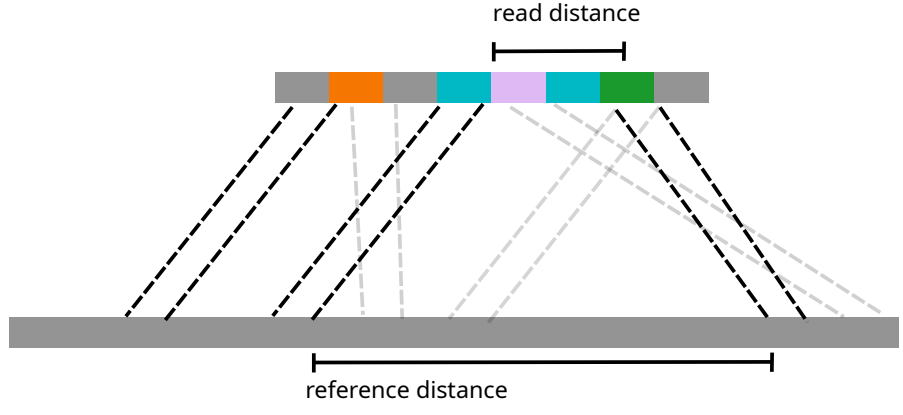
1. Seeding
2. **Chaining**
3. Alignment



Chains should minimize gap distances
(difference between read distance and reference distance)

Long read giraffe algorithm

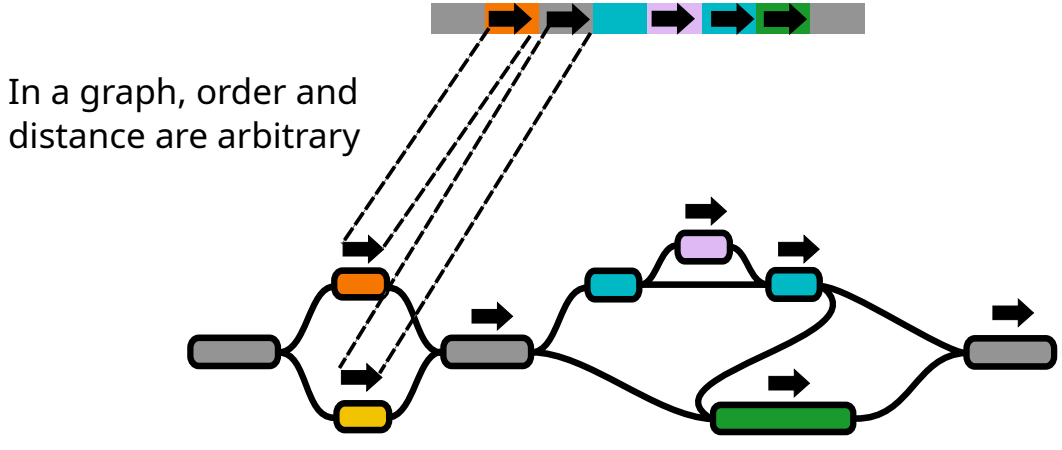
1. Seeding
2. **Chaining**
3. Alignment



Chains should minimize gap distances
(difference between read distance and reference distance)

Long read giraffe algorithm

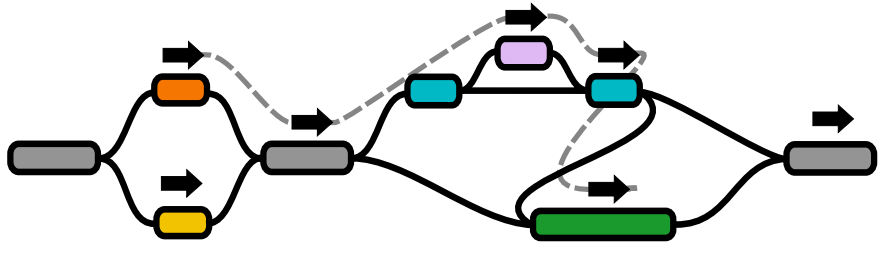
1. Seeding
2. **Chaining**
3. Alignment



Long read giraffe algorithm

1. Seeding
2. **Chaining**
3. Alignment

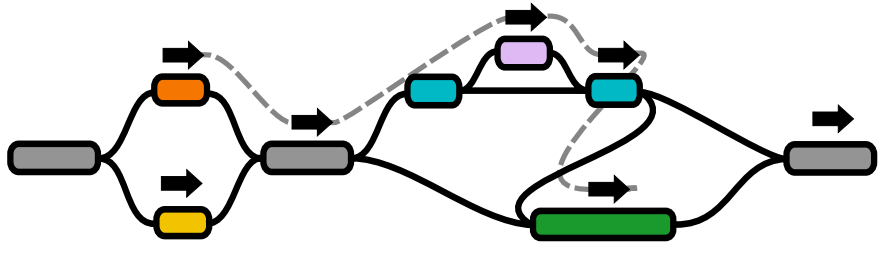
Consecutive seeds in a chain must be reachable



Long read giraffe algorithm

1. Seeding
2. **Chaining**
3. Alignment

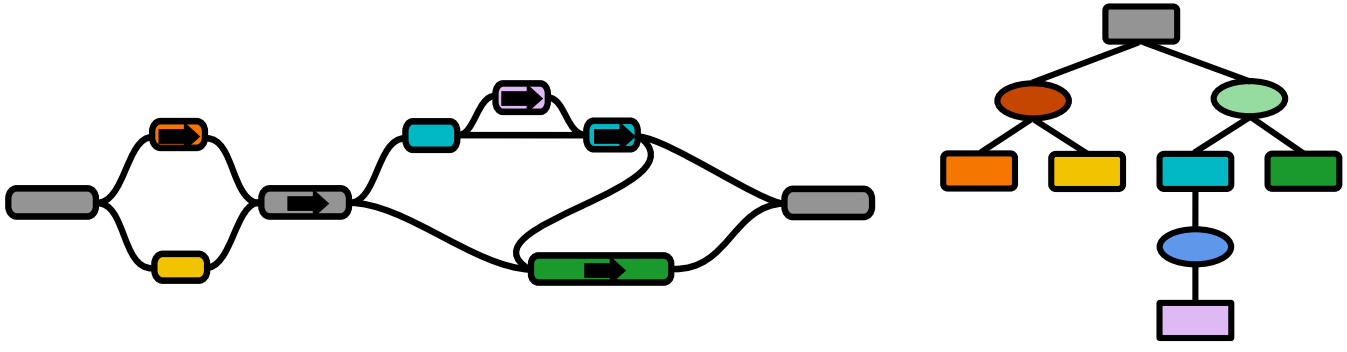
Consecutive seeds in a chain must be reachable



Distances are found using a **zip code tree**

Long read giraffe algorithm

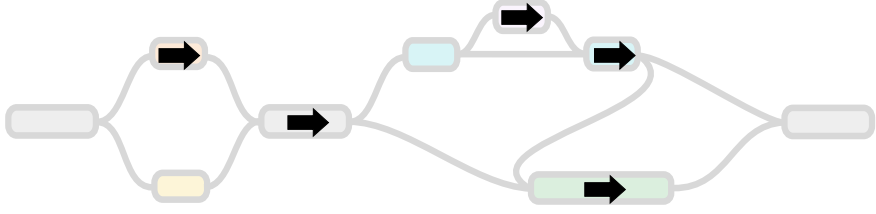
1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment

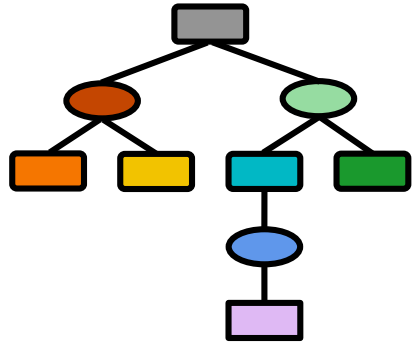


Zip code tree: new data structure for finding order and distance among seeds in a graph

Long read giraffe algorithm

1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment

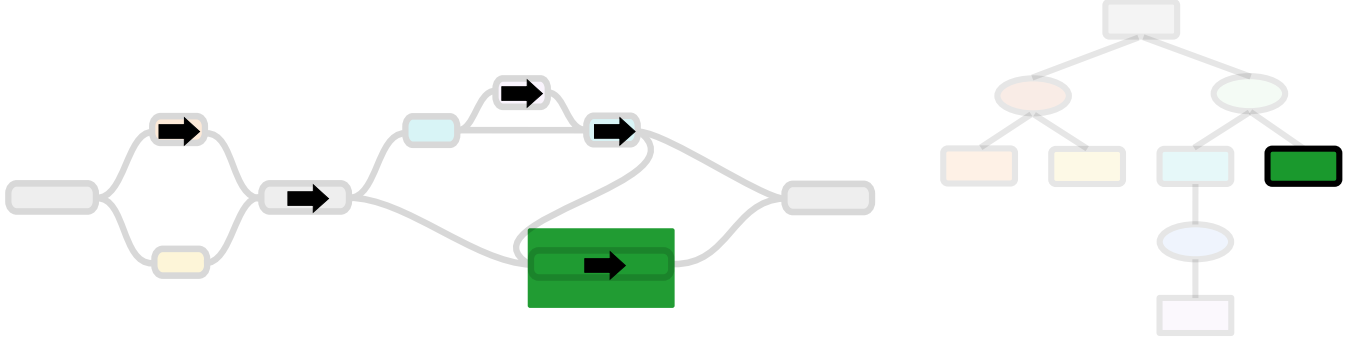




Each seed in the graph becomes a node in the zip code tree

Long read giraffe algorithm

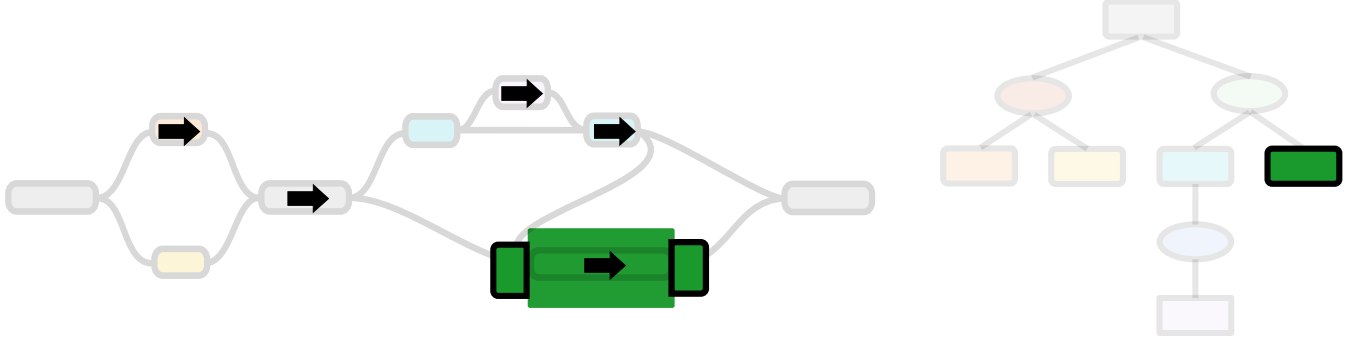
1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment



For each snarl and chain, add the bounds as nodes in the zip code tree

Long read giraffe algorithm

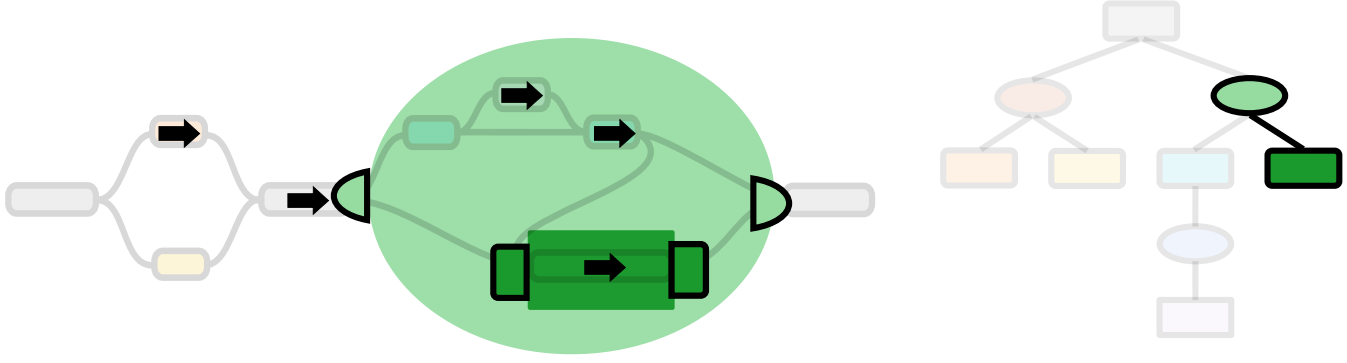
1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment



For each snarl and chain, add the bounds as nodes in the zip code tree

Long read giraffe algorithm

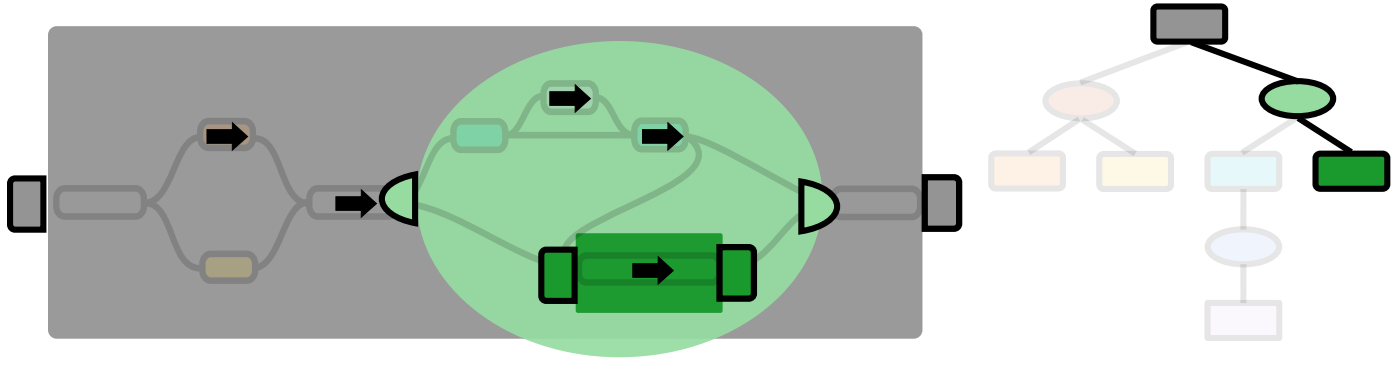
1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment



For each snarl and chain, add the bounds as nodes in the zip code tree

Long read giraffe algorithm

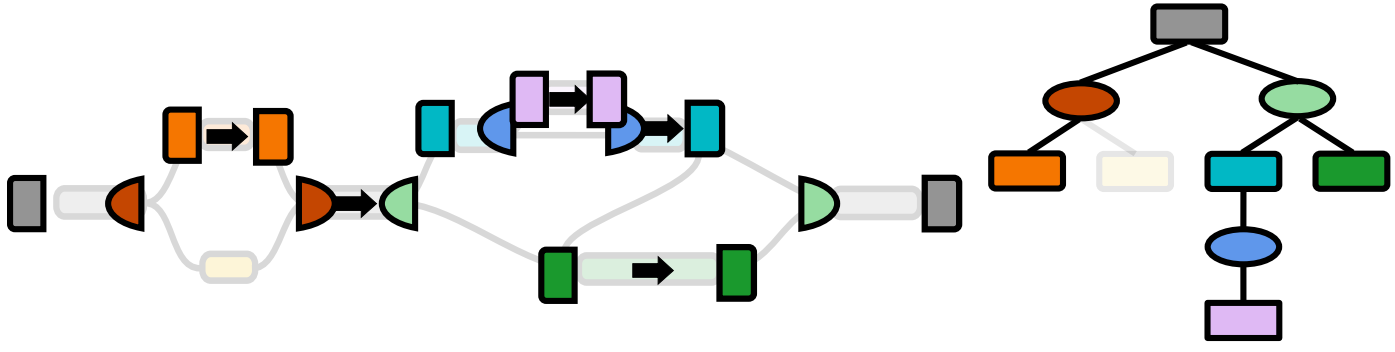
1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment



For each snarl and chain, add the bounds as nodes in the zip code tree

Long read giraffe algorithm

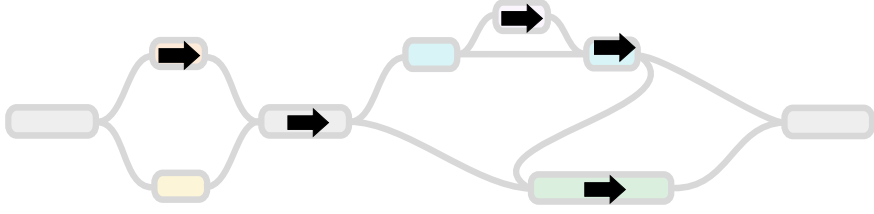
1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment

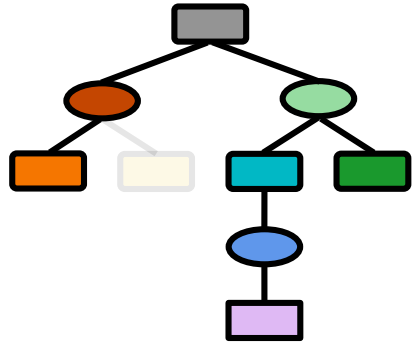


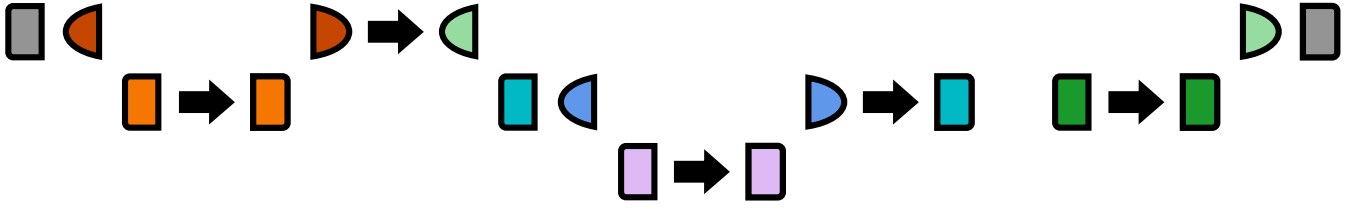
For each snarl and chain, add the bounds as nodes in the zip code tree

Long read giraffe algorithm

1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment

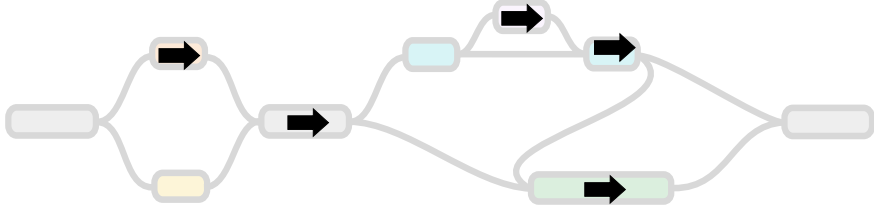


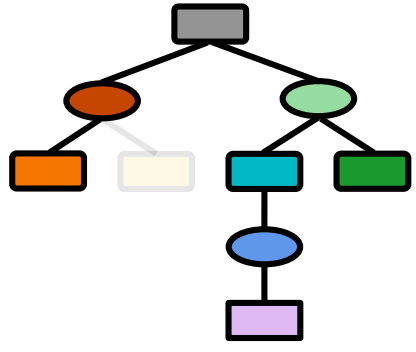


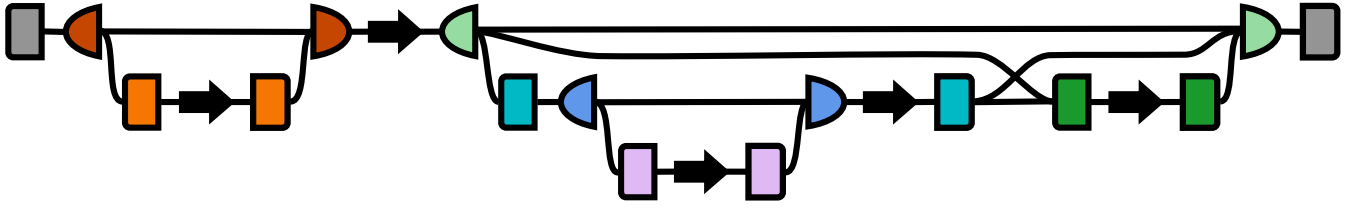


Long read giraffe algorithm

1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment



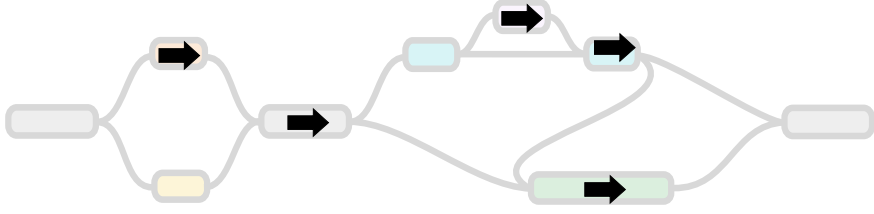


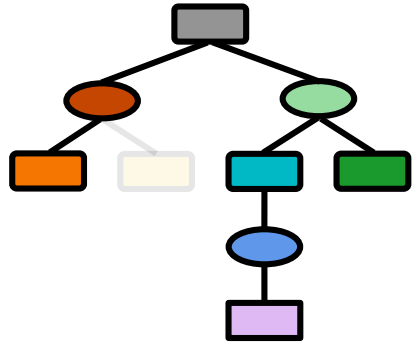


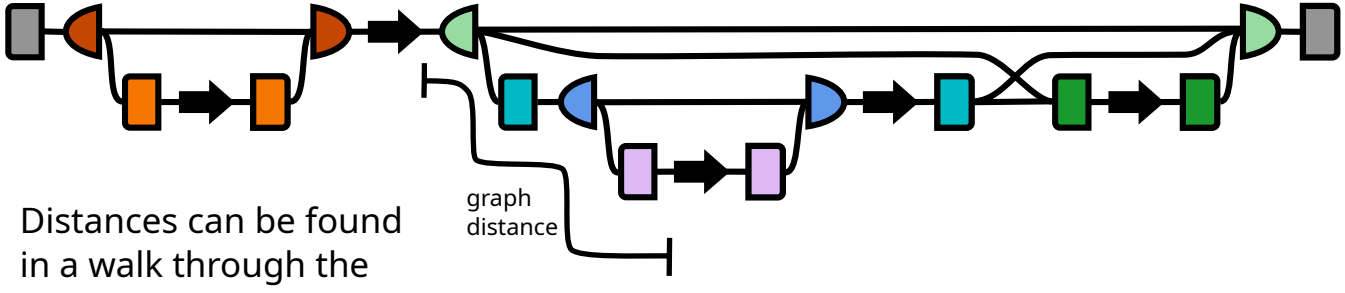
Add edges with distances between adjacent nodes

Long read giraffe algorithm

1. Seeding
2. **Zip code tree making** with **Distance Index**
3. Chaining with **Zip code trees**
4. Alignment



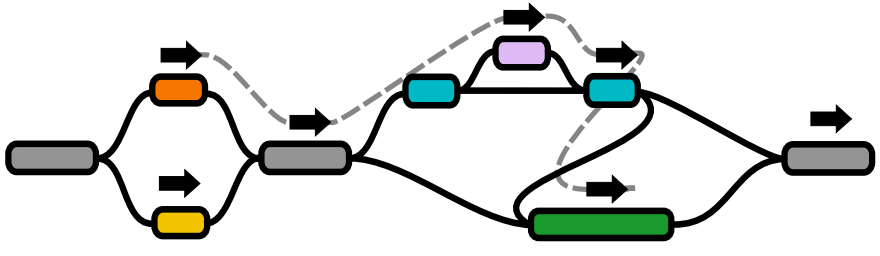




Distances can be found in a walk through the zip code tree

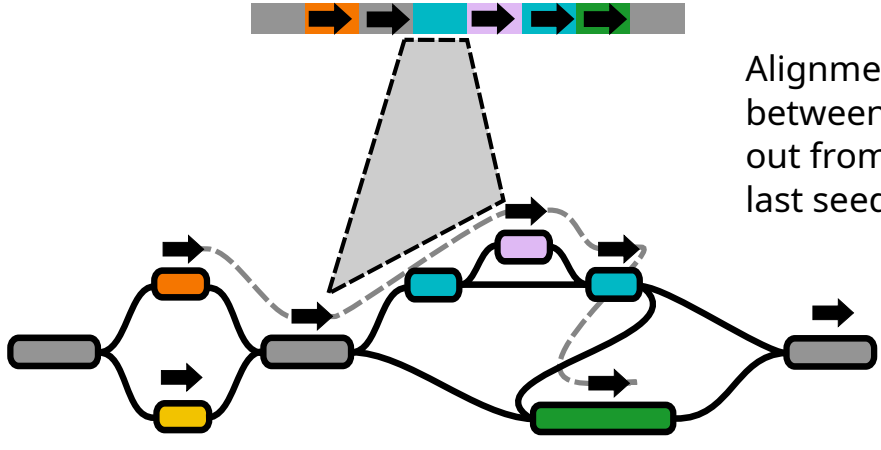
Long read giraffe algorithm

1. Seeding
2. Zip code tree making
3. Chaining
4. **Alignment** with graph



Long read giraffe algorithm

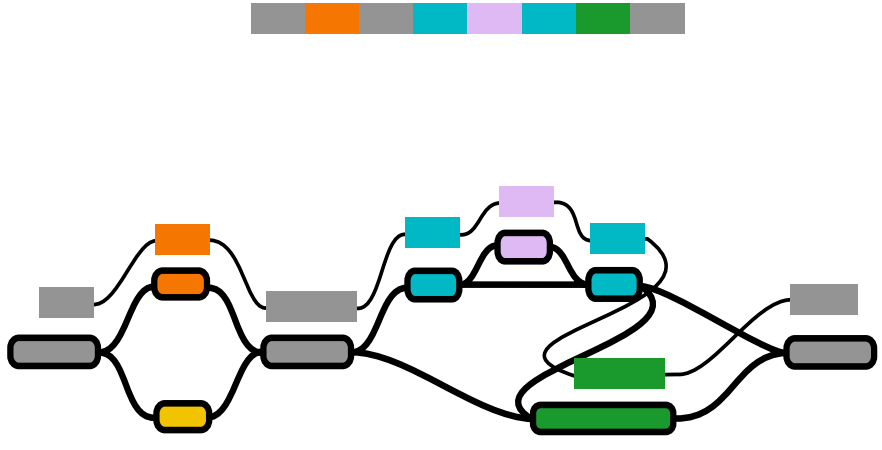
1. Seeding
2. Zip code tree making
3. Chaining
4. **Alignment** with graph



Alignment is done between seeds and out from the first and last seed

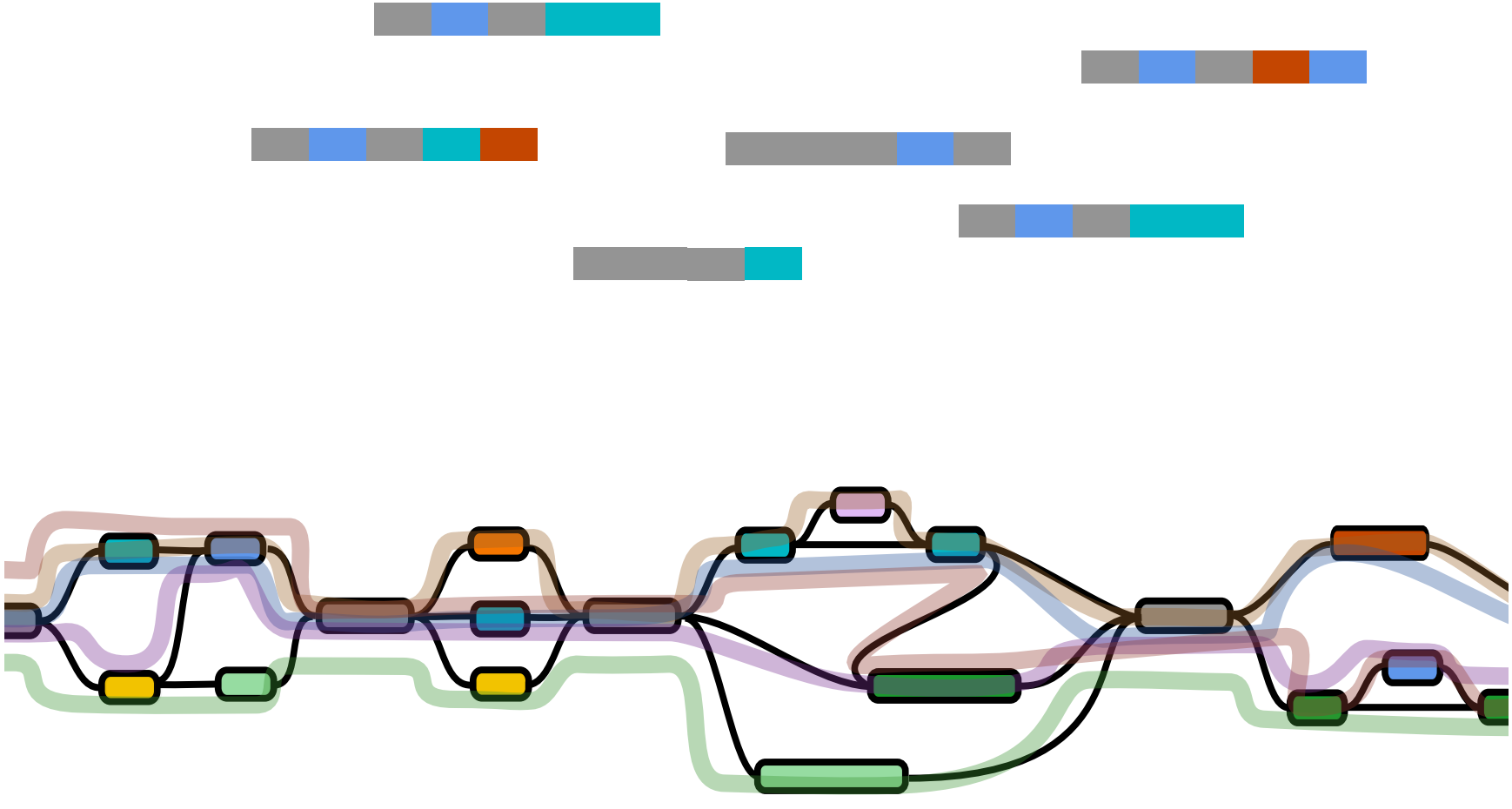
Long read giraffe algorithm

1. Seeding
2. Zip code tree making
3. Chaining
4. Alignment



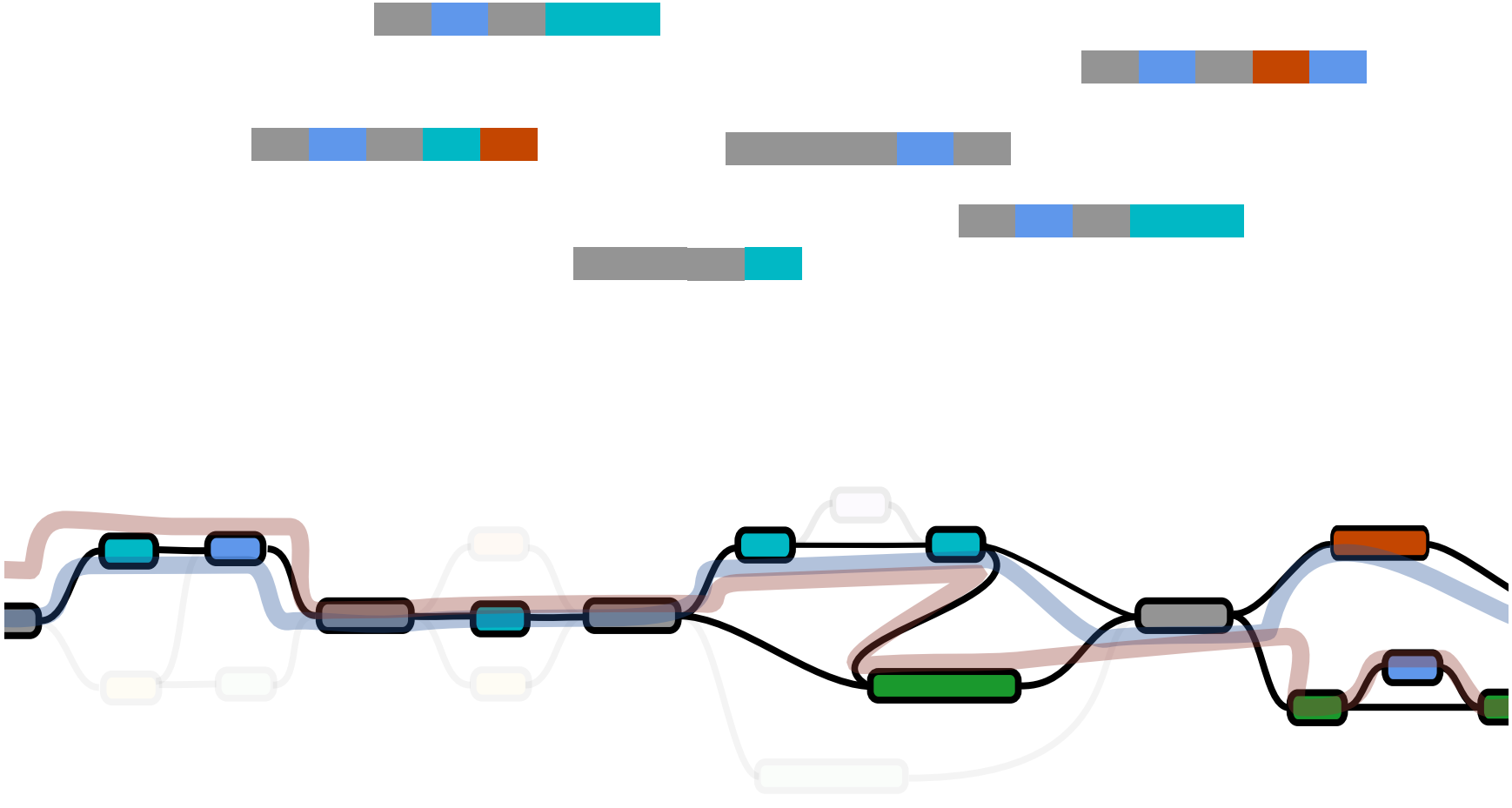
Haplotype sampling for mapping

Complex graphs can be slow to map to



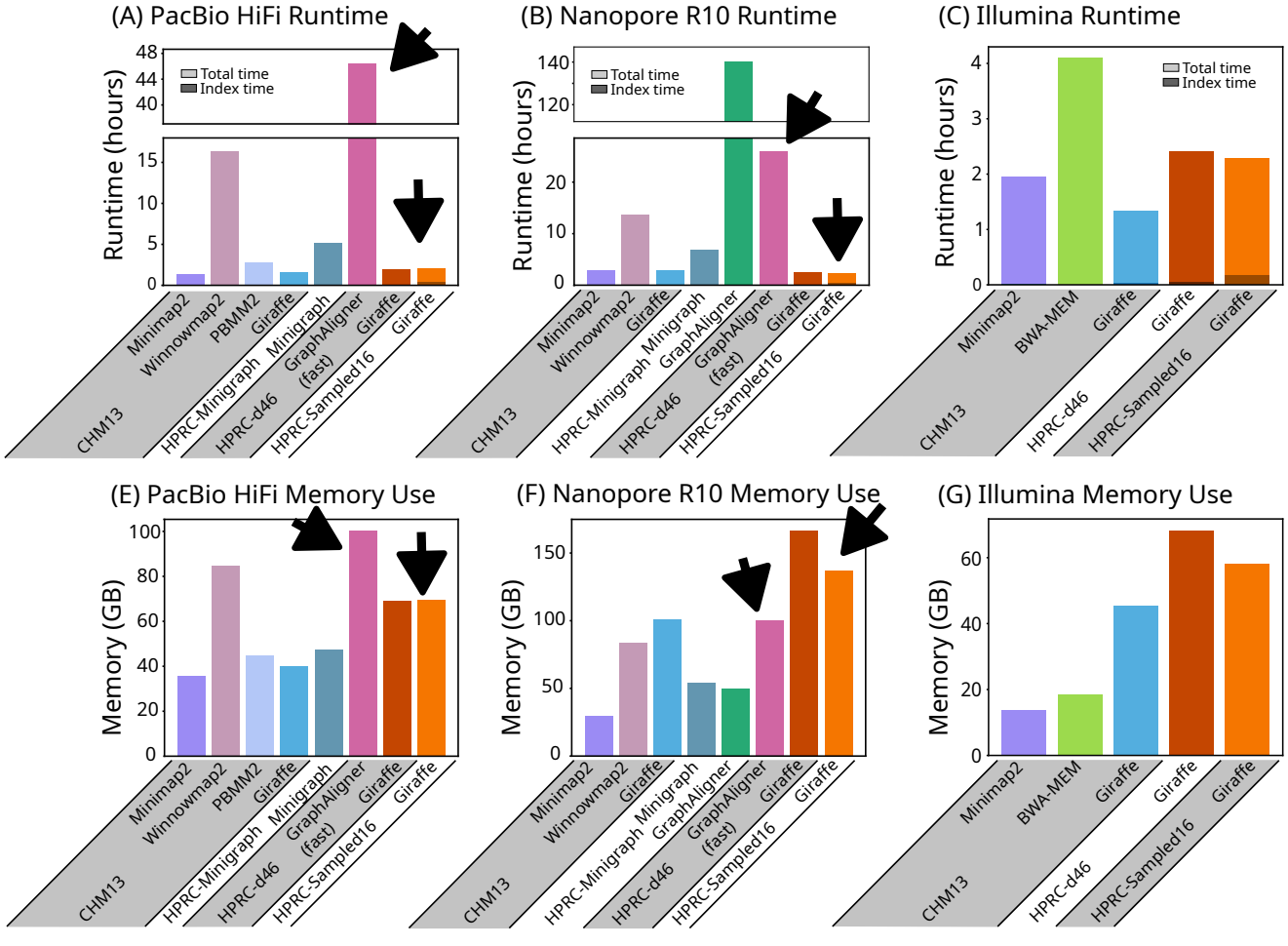
Haplotype sampling for mapping

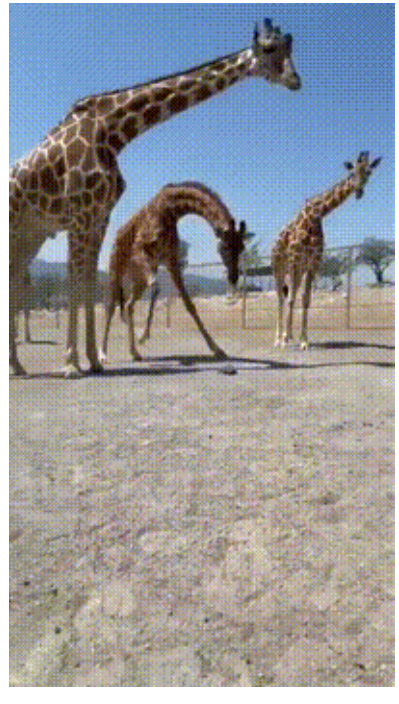
Simplify graphs by sampling haplotypes similar to reads



Giraffe results on HPRC v2 Minigraph-cactus

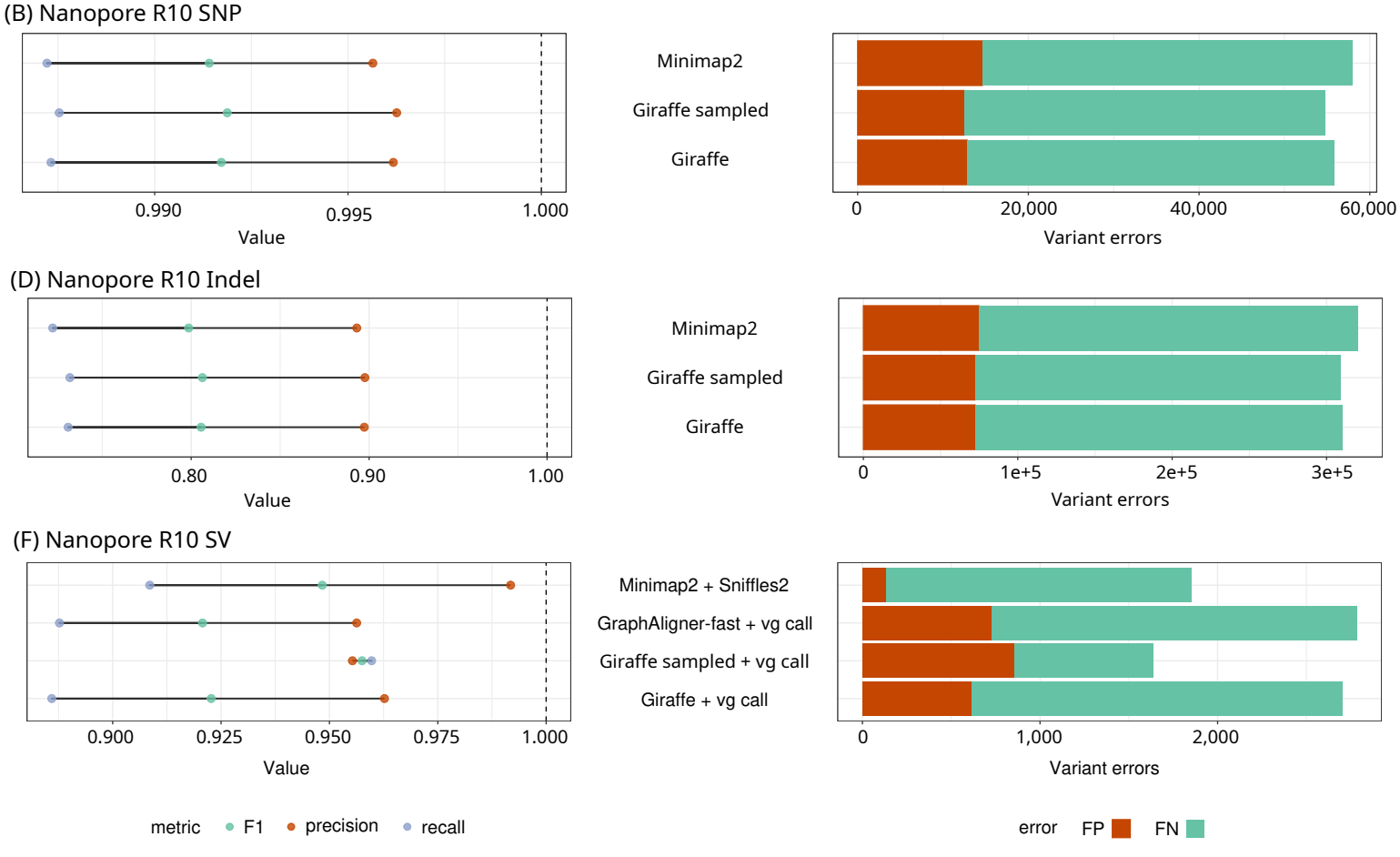
Pangenome with 462 haplotypes + 2 reference genomes





Giraffe results on HPRC v2 Minigraph-cactus

Small variant calling with DeepVariant, structural variant calling with Sniffles2 vs genotyping with `vg call`



vg graph formats and indexes

Indexes

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`
6. `.gcsa` (Generalized Compressed Suffix Array): substring index used by `vg map` and `vg mpmap`

Graphs

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`
6. `.gcsa` (Generalized Compressed Suffix Array): substring index used by `vg map` and `vg mpmap`

Graphs

1. `.gbz` (GBWT + GG): the graph induced by the GBWT

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`
6. `.gcsa` (Generalized Compressed Suffix Array): substring index used by `vg map` and `vg mpmap`

Graphs

1. `.gbz` (GBWT + GG): the graph induced by the GBWT
2. `.hg` (/`.vg`) (HashGraph): graph format optimized for speed

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`
6. `.gcsa` (Generalized Compressed Suffix Array): substring index used by `vg map` and `vg mpmap`

Graphs

1. `.gbz` (GBWT + GG): the graph induced by the GBWT
2. `.hg` (/`.vg`) (HashGraph): graph format optimized for speed
3. `.pg` (/`.vg`) (PackedGraph): graph format optimized for space efficiency

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`
6. `.gcsa` (Generalized Compressed Suffix Array): substring index used by `vg map` and `vg mpmap`

Graphs

1. `.gbz` (GBWT + GG): the graph induced by the GBWT
2. `.hg` (/`.vg`) (HashGraph): graph format optimized for speed
3. `.pg` (/`.vg`) (PackedGraph): graph format optimized for space efficiency
4. `.xg`: older graph format

`vg` graph formats and indexes

Indexes

1. `.gbwt` (Graph Burrows Wheeler Transform): haplotype paths
2. `.gg` (GBWT Graph): node sequences for a GBWT
3. `.dist` (Distance Index): snarl decomposition plus minimum distances
4. `.zipcodes`: per-node distance information used by `vg giraffe`
5. `.min` (Minimizer Index): minimizers used by `vg giraffe`
6. `.gcsa` (Generalized Compressed Suffix Array): substring index used by `vg map` and `vg mpmap`

Graphs

1. `.gbz` (GBWT + GG): the graph induced by the GBWT
2. `.hg` (/`.vg`) (HashGraph): graph format optimized for speed
3. `.pg` (/`.vg`) (PackedGraph): graph format optimized for space efficiency
4. `.xg`: older graph format
5. `.vg`: protobuf-based graph format

Resources

- These slides: https://github.com/jmonlong/getapan2025
- vg wiki
  - file types: https://github.com/vgteam/vg/wiki/File-Types
  - snarl explainer: https://github.com/vgteam/vg/wiki/Snarls-and-chains
  - deconstruct: https://github.com/vgteam/vg/wiki/VCF-export-with-vg-deconstruct
- vg manpage: https://github.com/vgteam/vg/wiki/vg-manpage
- Snarls paper doi:10.1089/cmb.2017.0251
- Short read giraffe paper doi:10.1126/science.abg8871
- Long read giraffe paper doi:10.1101/2025.09.29.678807

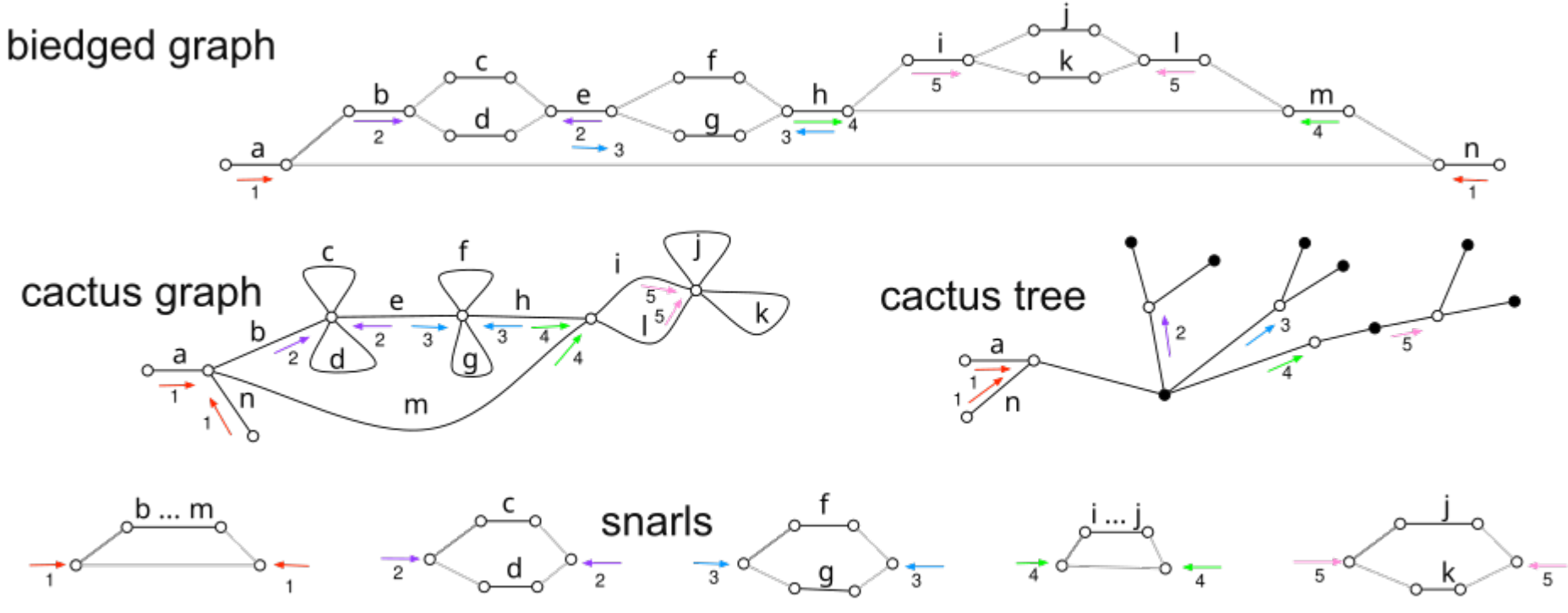
Acknowledgements

- Benedict Paten
- Jordan Eizenga
- Glenn Hickey
- Adam Novak
- Jouni Sirén

Snarl decomposition algorithm?

Vague understanding from Paten et al Journal of Computational Biology 2018.

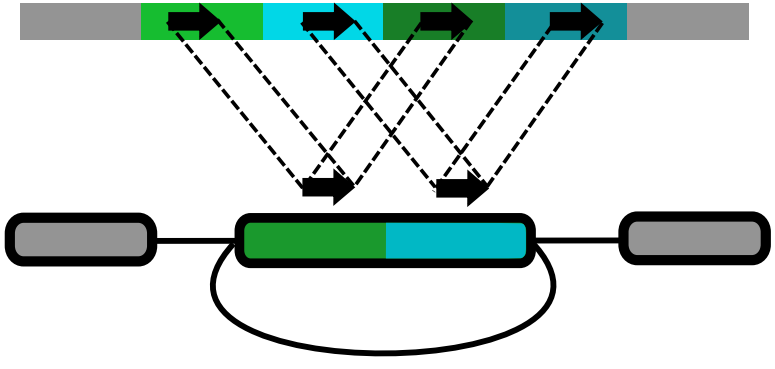
- **chain pair**: project to same vertex and within same cycle in Cactus graph.
- **bridge pair**: project to same "net" vertex in Cactus tree.



adapted from Paten et al. Journal of Computational Biology 2018

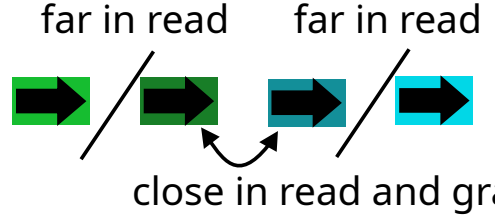
Duplications in zip code trees

Duplications in snarls



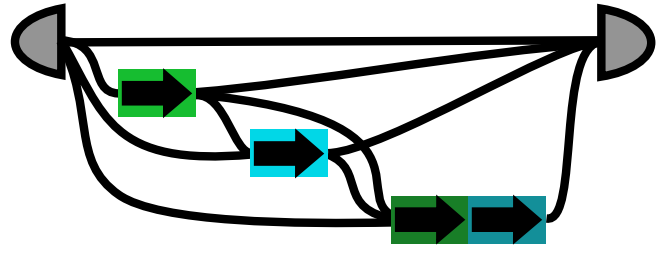
1. Sort seeds by chain coordinate

2. Split into runs by graph and read distance



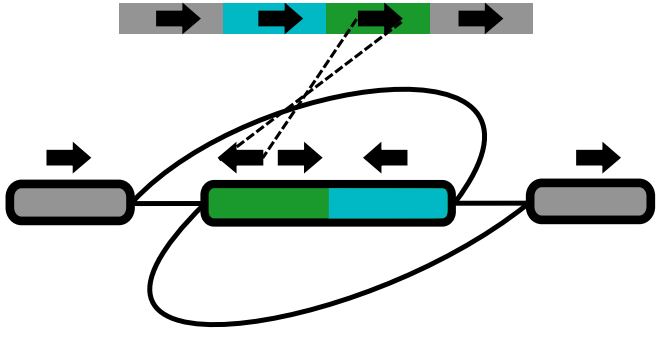
3. Sort runs by read coordinate

4. Build zip code tree with runs as chains



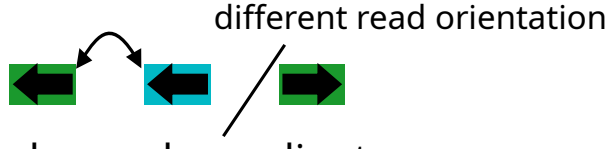
Inversions in zip code trees

Inversions in snarls



1. Sort seeds by chain coordinate

2. Split into runs by graph and read distance and orientation in read



3. Sort runs by read coordinate

4. Orient runs based on correlation of read positions in graph

5. Build zip code tree with runs as chains