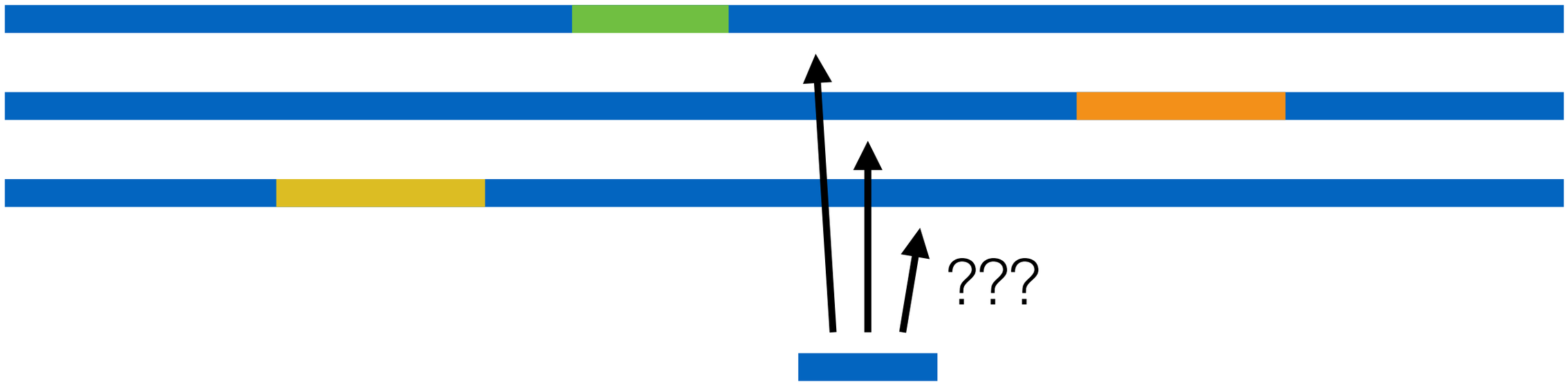
Haplotype-aware graph indexes

Jouni Sirén, Erik Garrison, Adam M. Novak,
Benedict Paten, and Richard Durbin

Reference sequence

- Reference sequences are easy to work with.
- When the sample diverges from the reference, the reference does not help and it may bias our results.

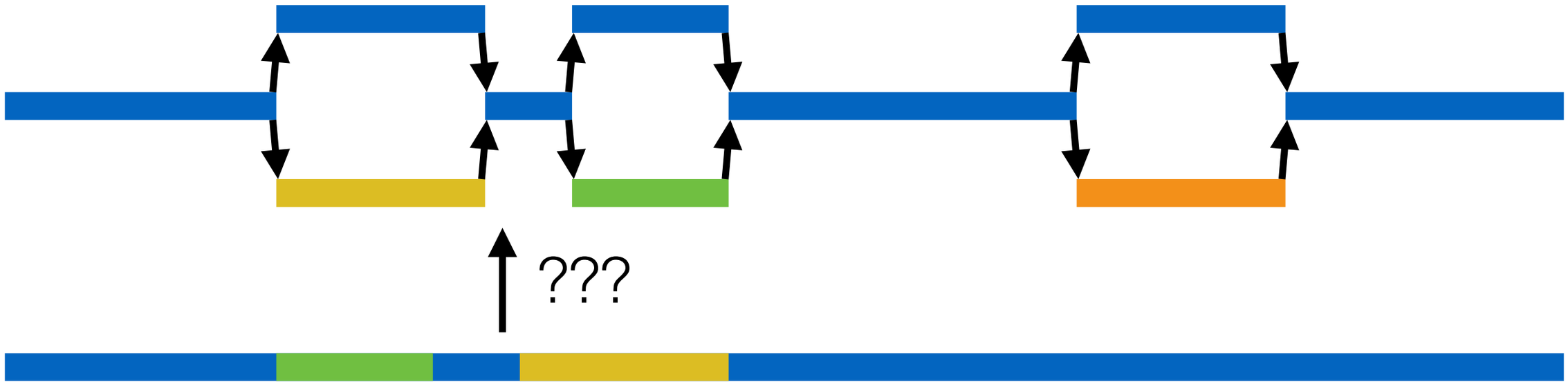
Collection of haplotypes



- We can try to reduce the reference bias by using a collection of haplotypes as the reference.

- How to deal with reads mapping to multiple haplotypes?

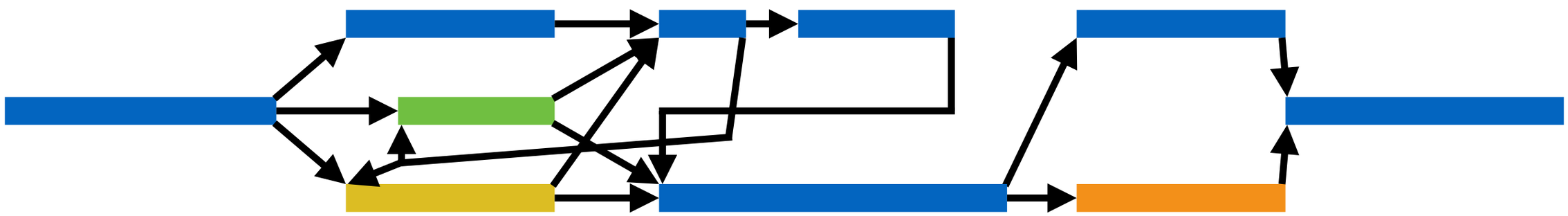
Global alignment / DAG



- A global alignment helps with reads mapping to multiple haplotypes. If we collapse shared regions, we get a directed acyclic graph.

- How to deal with structural variation?

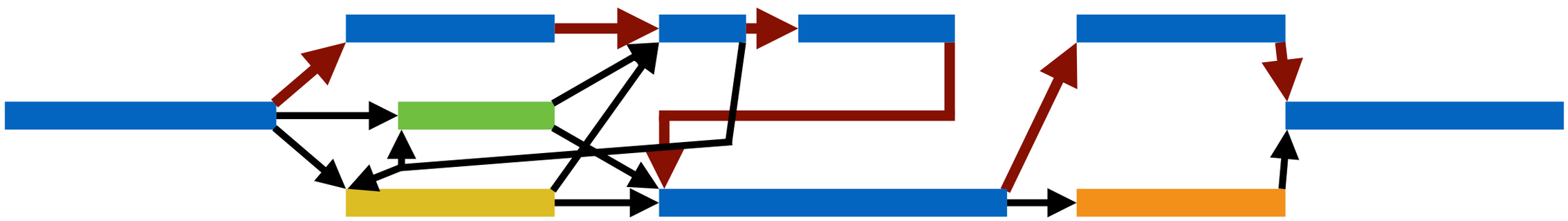
Local alignments



- If we use local alignments instead, we get assembly graphs that can handle structural variation.

- They contain nonsensical paths and lack a global coordinate system.

VG model



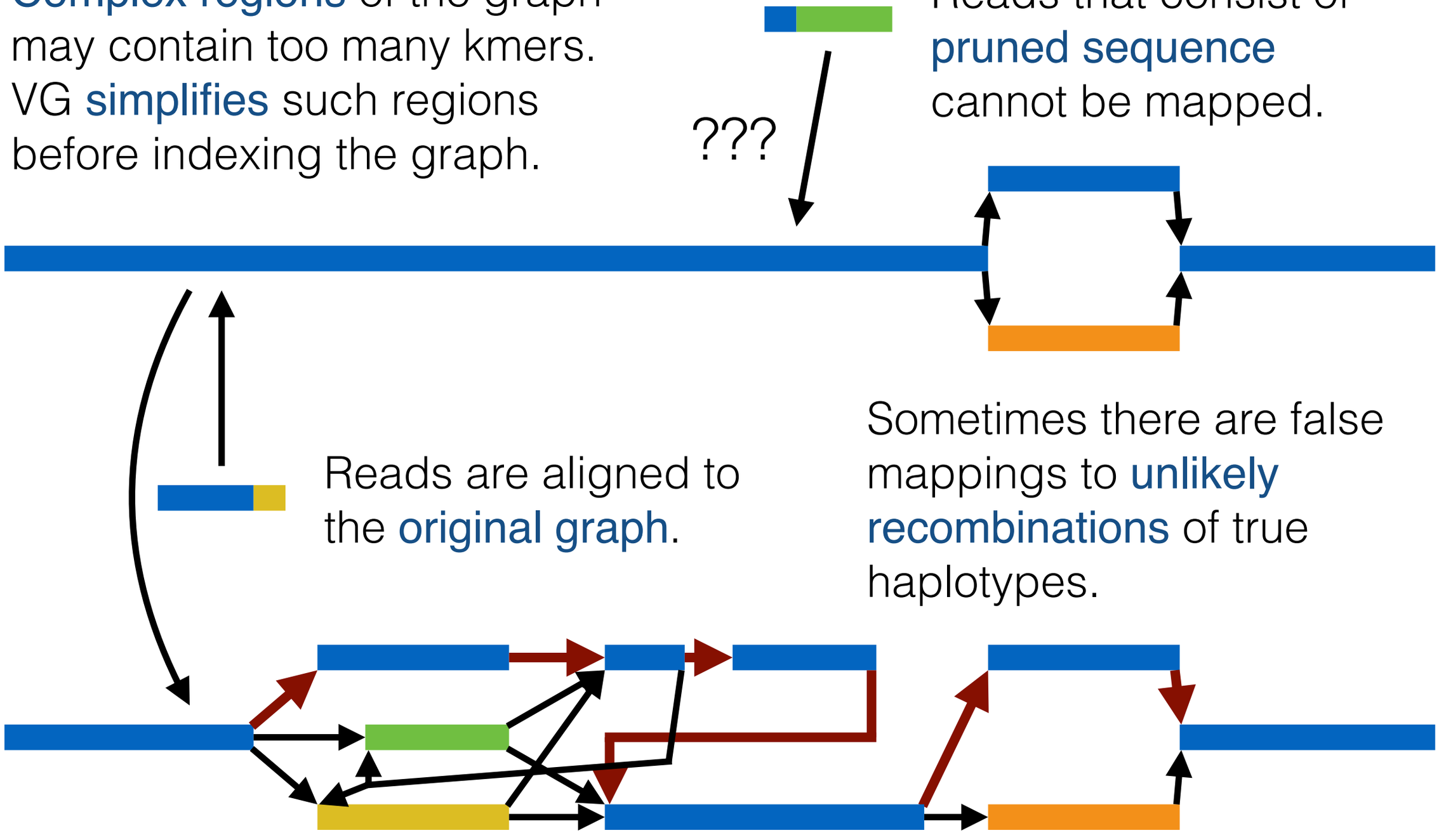
- The variation graph toolkit **VG** (Garrison et al, Nature Biotechnology, 2018; https://github.com/vgteam/vg) works with arbitrary graphs.

- A **primary path** provides a coordinate system.

- We still cannot deal with structural variation in DAGs or with nonsensical paths in assembly graphs.

Read mapping in VG

Complex regions of the graph may contain too many kmers. VG simplifies such regions before indexing the graph.

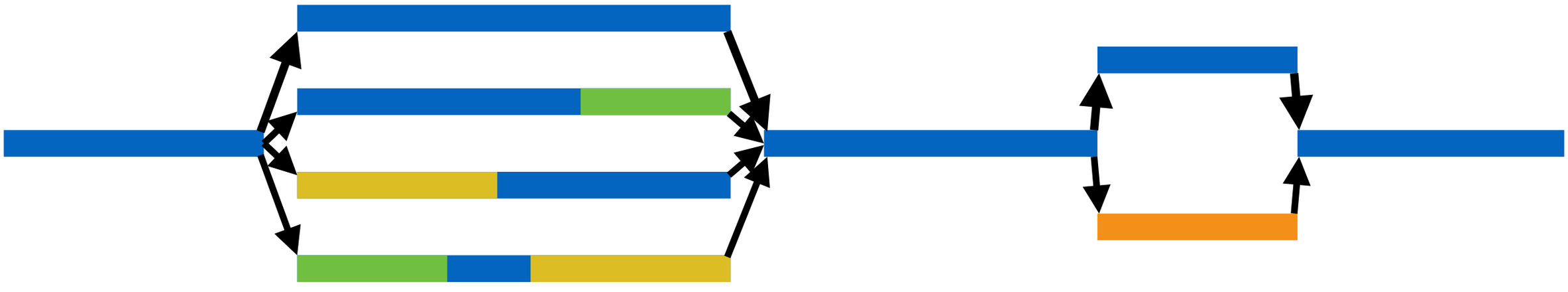


Reads that consist of pruned sequence cannot be mapped.

Reads are aligned to the original graph.

Sometimes there are false mappings to unlikely recombinations of true haplotypes.

Augmenting VG model



- Reference: graph + primary path + haplotype paths.
- Preserve haplotypes when simplifying the graph.
- Penalize recombinations when aligning reads.

This talk: VG infrastructure

- How to store and index the haplotypes as paths in the graph?
- A scalable version of the graph extension (Novak et al, 2017) of the positional BWT (Durbin, 2014).
- Tested with 5,000 human haplotypes; trying to scale up to 100,000 haplotypes.
- A subsequent paper will investigate the use of haplotype information in read mapping.

FM-index

Burrows–Wheeler transform

- Add a unique terminator ($) to the end of the text, sort the suffixes in lexicographic order, and output the preceding character for each suffix.

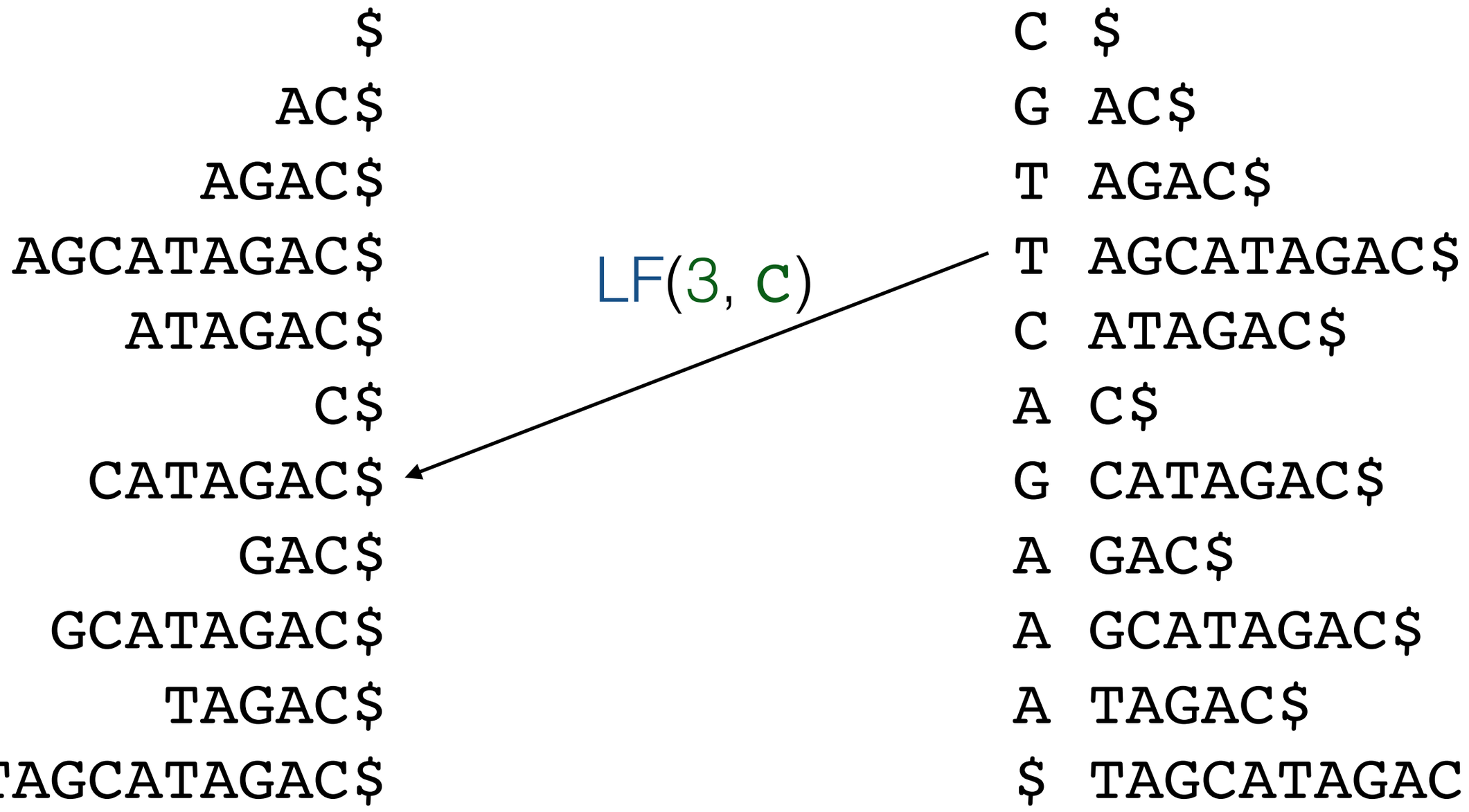
- The permutation is easily reversible and makes the text easier to compress (Burrows & Wheeler, 1994).

- The combinatorial structure is similar to the suffix array, which makes the BWT useful as a space-efficient text index (Ferragina & Manzini, 2000, 2005).

- There is a straightforward generalization to multiple strings by using distinct terminators during sorting.

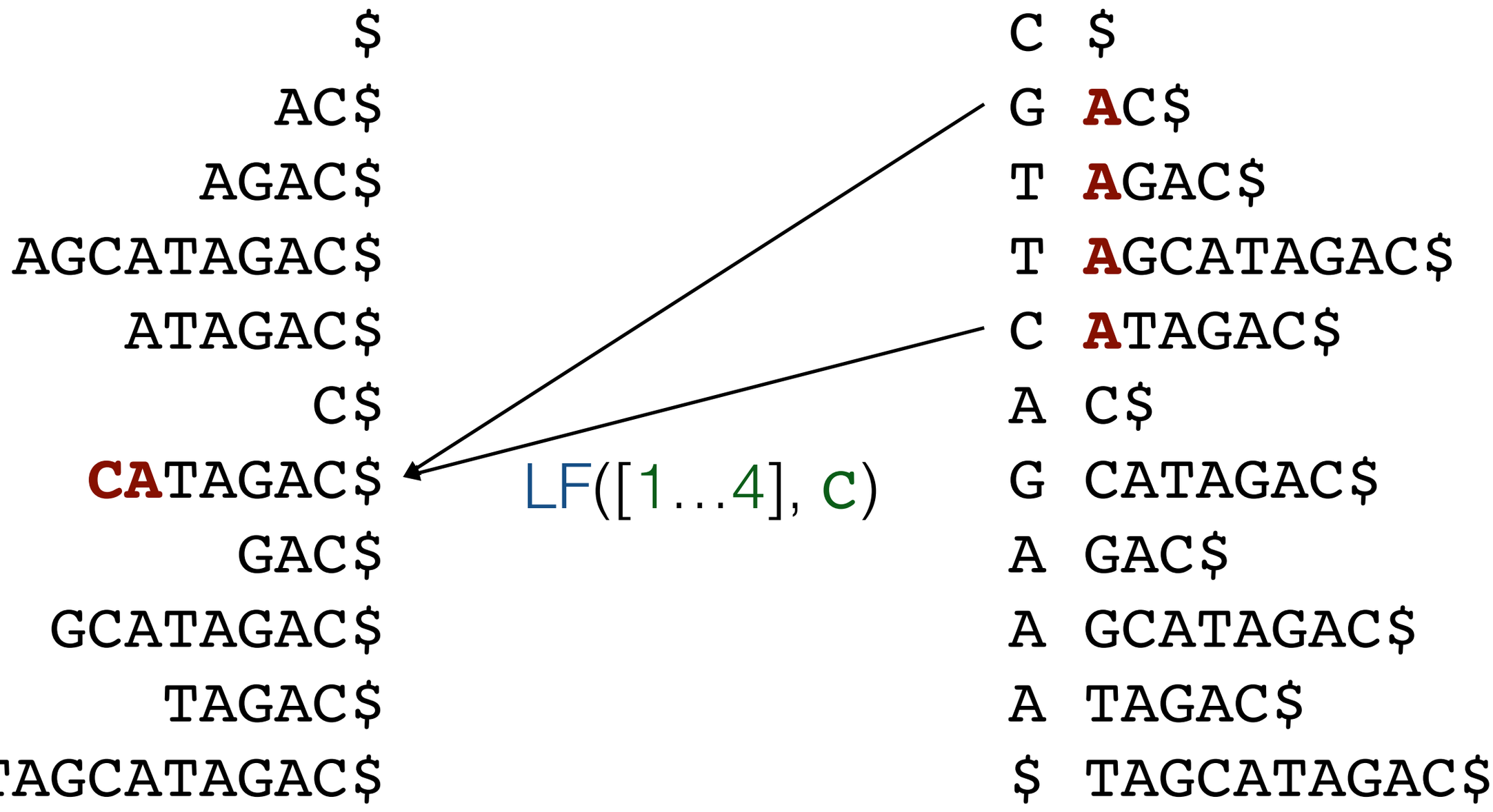
```
TAGCATAGAC$

C $
G AC$
T AGAC$
T AGCATAGAC$
C ATAGAC$
A C$
G CATAGAC$
A GAC$
A GCATAGAC$
A TAGAC$
$ TAGCATAGAC$
```

LF-mapping



Interpretation: LF(i, c) = C[c] + BWT.rank(i, c) suffixes are **strictly before** the hypothetical suffix.

Backward searching



LF([sp…ep], c) = [LF(sp, c)…LF(ep+1, c) −1]

Locating the occurrences

```
                $
               AC$
              AGAC$
           AGCATAGAC$
            ATAGAC$
                C$
              CATAGAC$
          GCATAGAC$
             TAGAC$
        TAGCATAGAC$
```

LF(4, C)

```
C $
G AC$
T AGAC$
T AGCATAGAC$
C ATAGAC$
```

Text position 4

```
G CATAGAC$
A GAC$
A GCATAGAC$
A TAGAC$
$ TAGCATAGAC$
```

Text position 3

SA[LF(i, BWT[i])] = SA[i] – 1

FMD-index

- In bioinformatics, the text and/or the patterns are often a mix of forward and reverse complement orientations.

- We can simplify the situation by indexing the text in both orientations in the same FM-index (Li, 2012).

- We can then:

  - search for both orientations of the pattern in both orientations of the text; and

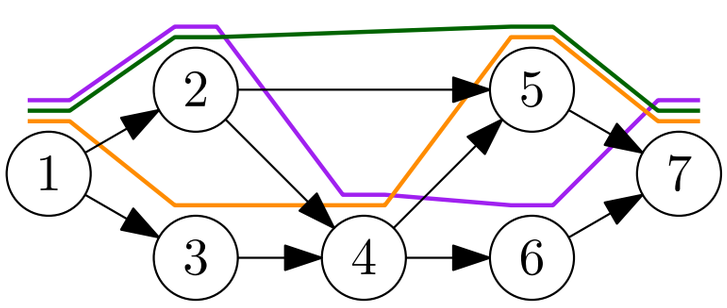
  - support bidirectional searching.

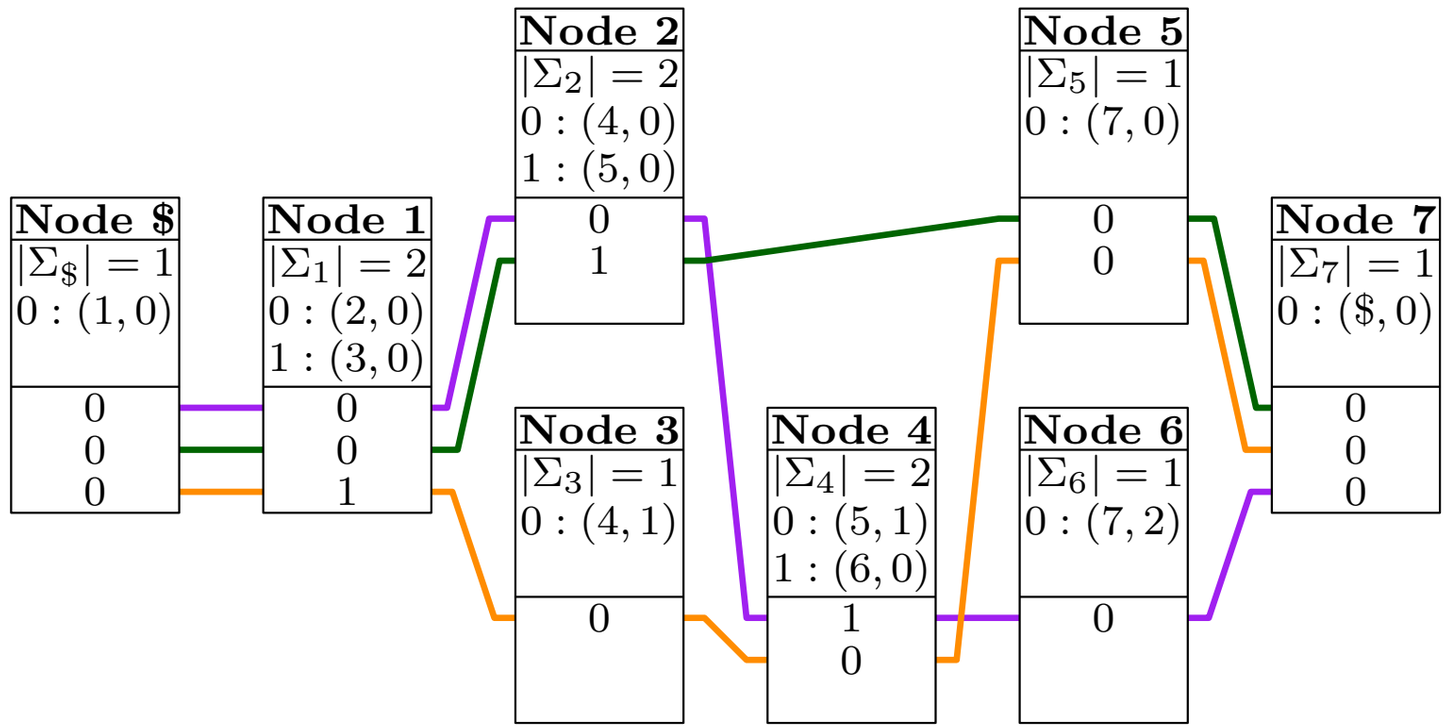
Graph BWT

Some assumptions

- We have a repetitive collection of paths in a large graph with a low average outdegree.
- The paths are represented as node sequences.
- The number of occurrences of almost every node is proportional to the number of samples.
- While there may be cycles, the graph is still mostly linear and topologically sorted.
- We index reverse paths, as it is more intuitive to have LF-mapping following the edges forward.

Records





- We have a separate record for each node (each character in the alphabet).
- The header stores the outgoing edges and the rank information needed for LF-mapping.
- The body stores the (run-length encoded) part of the BWT corresponding to the prefixes ending with the current node.

Node 4
Outdegree 2 0: node 5, offset 1 1: node 6, offset 0
10

Some consequences

- Query performance depends on the size of the local alphabet, not on the global alphabet.
- As the graph is (almost) topologically sorted, search tends to scan the BWT linearly instead of jumping around randomly.
- Because the rank structure is local, memory access takes almost constant time regardless of text size.
- We could even use a memory-mapped file for BWT.

GBWT construction

- An incremental algorithm based on BCR (Bauer, Cox & Rosone, 2013) and RopeBWT2 (Li, 2014).

- Insert a batch of sequences into a dynamic FM-index.

- Rewrite a record every time we update it.

- Larger batches use more memory but reduce the total number of rewrites.

- Some buffering is required, as we generate the sequences one variant site at a time.

Construction in VG

```
for each batch of 200 samples
    for each site in VCF
        for each sample in batch
            for each phase in sample
                if phaseBreak(phase, site)
                    GBWT.insert(S[phase])
                    S[phase].clear()
                S[phase].extend(site)
    for each sample in batch
        for each phase in sample
            GBWT.insert(S[phase])
            S[phase].clear()
```

- Separate process for each chromosome.
- Memory usage <1 GB / 10 Mbp with 1000GP data.
- Index both orientations to build an FMD-index.
- Sequences are buffered (size 100 million) and inserted in a background thread.
- The GBWTs for different chromosomes can be merged quickly, because the node ids do not overlap.

Benchmarks

Construction in the paper

- 1000GP data: 2504 samples, ~85 million variants.
- 29.3 million sequences of total length 1.62 trillion, alphabet size 493 million.
- AWS i3.8xlarge instance: 32 cores, 244 GB memory.
- 12 parallel jobs for 24 chromosomes + merging.
- Store sequence ids at one out of 1024 positions.
- 29 hours, final GBWT size 7.4 GB + 7.2 GB.

Faster construction

- We spent 29 hours for parsing the VCF files once for every 200 samples.
- Faster alternative: Parse each VCF only once and store the phasing information in a better format.
- Also some memory optimizations.
- 12 → 14 parallel jobs, 29.0 → 10.4 hours, 29.3 → 50.6 million sequences, 7.4+7.2 → 7.5+7.4 GB.

Why more sequences?

- 1000GP VCFs have issues with overlapping variants.
- In particular, a haplotype may both delete a base and replace it with another base.
- We accidentally prioritized SNPs over deletions; our new code treats such situations as phase breaks.
- We can also choose to ignore overlapping variants.
- 10.4 → 15.4 hours, 50.6 → 0.24 million sequences, 7.5+7.4 → 7.3+5.9 GB.

	Sequences	Construction	Size
Old	29.3M	29.0 h	7.4+7.2 GB
New	50.6M	10.4 h	7.5+7.4 GB
Ignore overlaps	240232	15.4 h	7.3+5.9 GB

- VCF parsing takes ~2 hours for the largest chromosomes.

- Indexing speed is >50M nodes/s (6 s/haplotype) with phase breaks and ~35M nodes/s (9 s/haplotype) when ignoring overlaps.

Conclusions

Conclusions

- We augmented the VG model with a collection of haplotypes.
- GBWT is an FM-index for repetitive collections of paths in low-degree graphs.
- We can easily index 5,000 human haplotypes.
- How to scale up to 100,000 haplotypes?