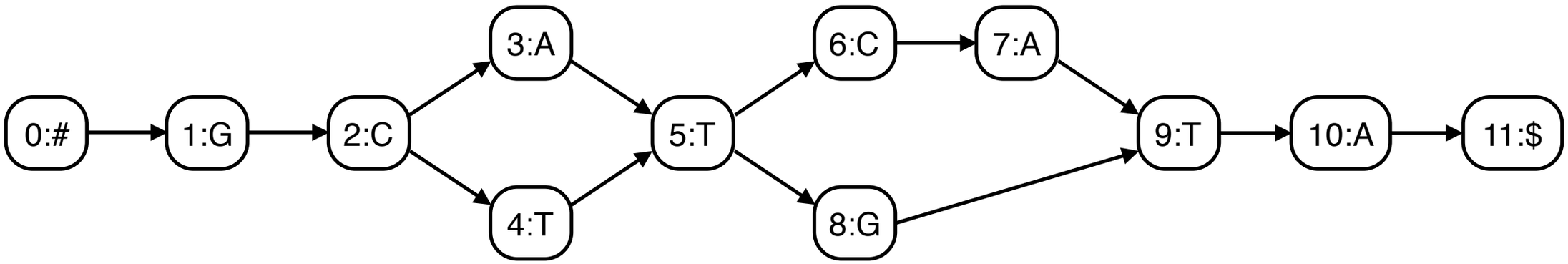
Indexing Variation Graphs

Jouni Sirén
Wellcome Trust Sanger Institute

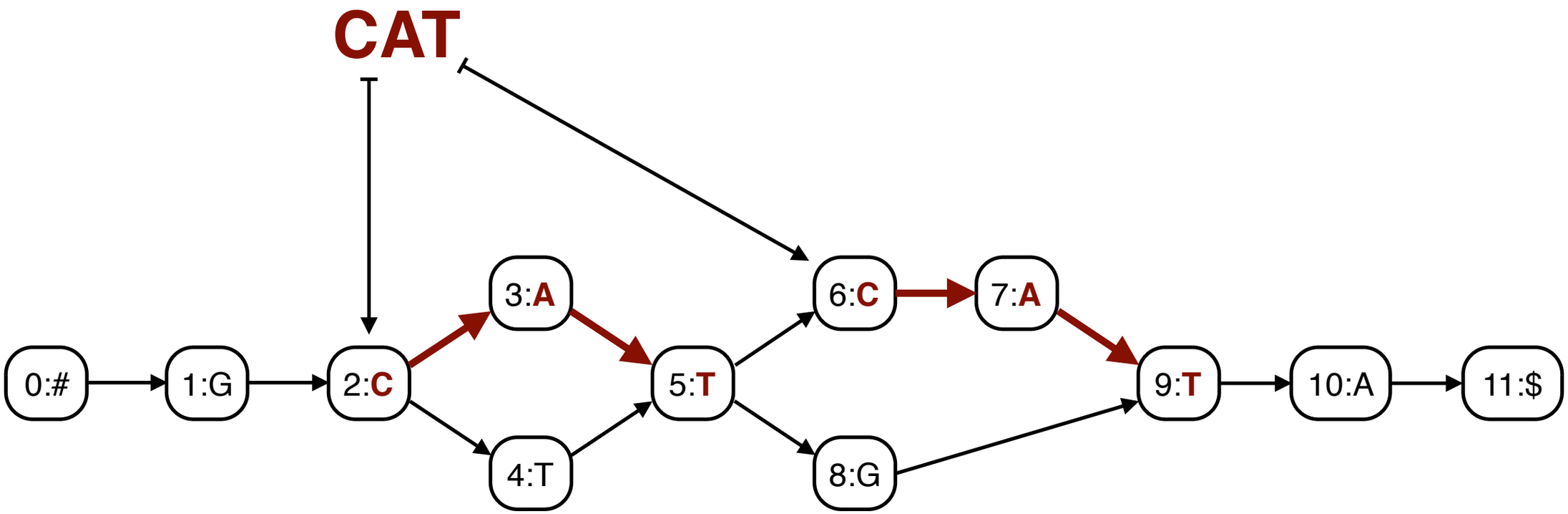


- Graphs are a natural way of representing genetic variation.

- Reference genomes could eventually become such graphs.

- The variation graph toolkit vg (Erik Garrison et al, https://github.com/vgteam/vg) is a community effort to develop tools for working with such graphs.

- This talk is about GCSA2, the path index used in vg.



- Path indexes are text indexes for the path labels in a graph.
- The index finds the start nodes of the paths labeled by the query string.
- Indexing the paths themselves is not cost-effective.

Trade-offs

- The number of kmers (substrings of length k) in a graph increases exponentially with k.

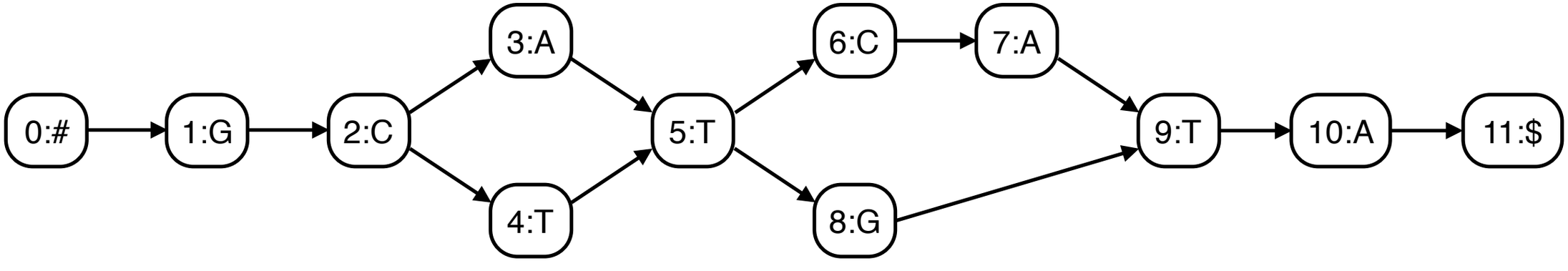
- k should be larger than the expected length of maximal exact matches.

- In one human variation graph, the number of kmers is $1.031^k \cdot 2.348$ billion, or 116 billion for $k = 128$.

- The design of a path index is a trade-off between index size, query performance, maximum query length, and ignoring complex regions of the graph.

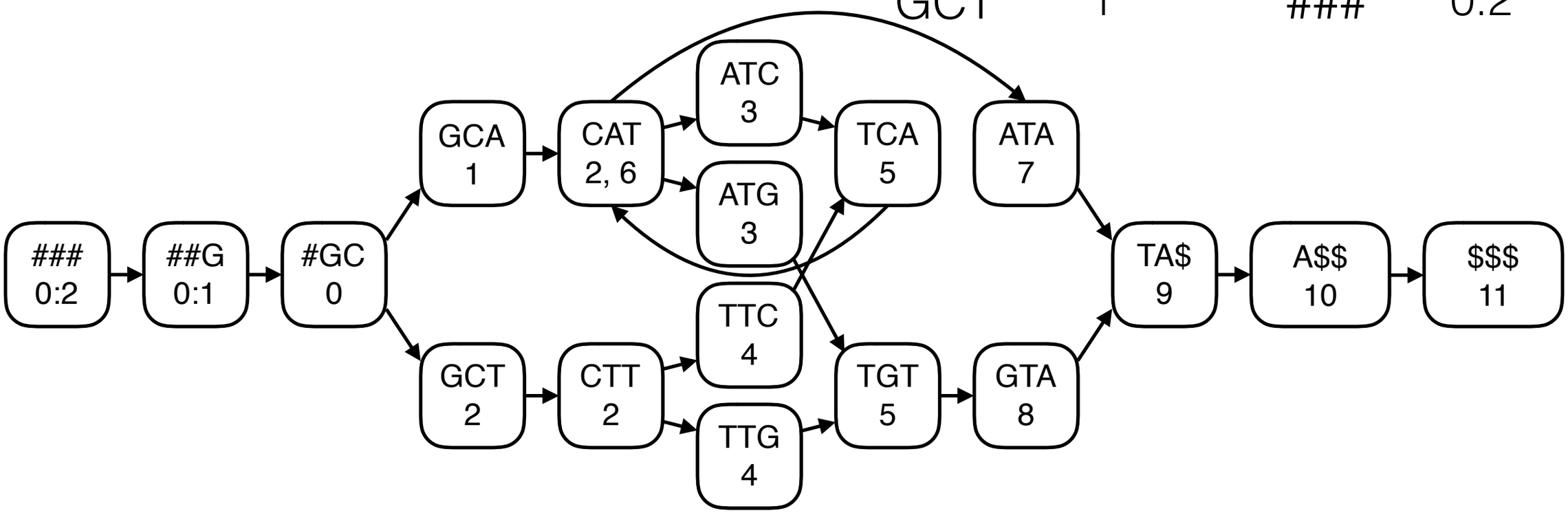
- The kmer index is a simple path index. It consists of key-value pairs.
- A hash table supports fast kmer queries.
- Binary search in a sorted array is slower but supports queries shorter than k.
- Index size: terabytes for 100 billion kmers.

Key	Value	Key	Value
$$$	11	GTA	8
A$$	10	TA$	9
ATA	7	TCA	5
ATC	3	TGT	5
ATG	3	TTC	4
CAT	2, 6	TTG	4
CTT	2	#GC	0
GCA	1	##G	0:1
GCT	1	###	0:2



- We can represent the kmer index as a de Bruijn graph.
- We label each node with the first character of the key.
- The de Bruijn graph approximates the input graph. There are no false negatives, and no false positives shorter than k+1.

Key	Value	Key	Value
$$$	11	GTA	8
A$$	10	TA$	9
ATA	7	TCA	5
ATC	3	TGT	5
ATG	3	TTC	4
CAT	2, 6	TTG	4
CTT	2	#GC	0
GCA	1	##G	0:1
GCT	1	###	0:2



FM-Index

Burrows-Wheeler transform

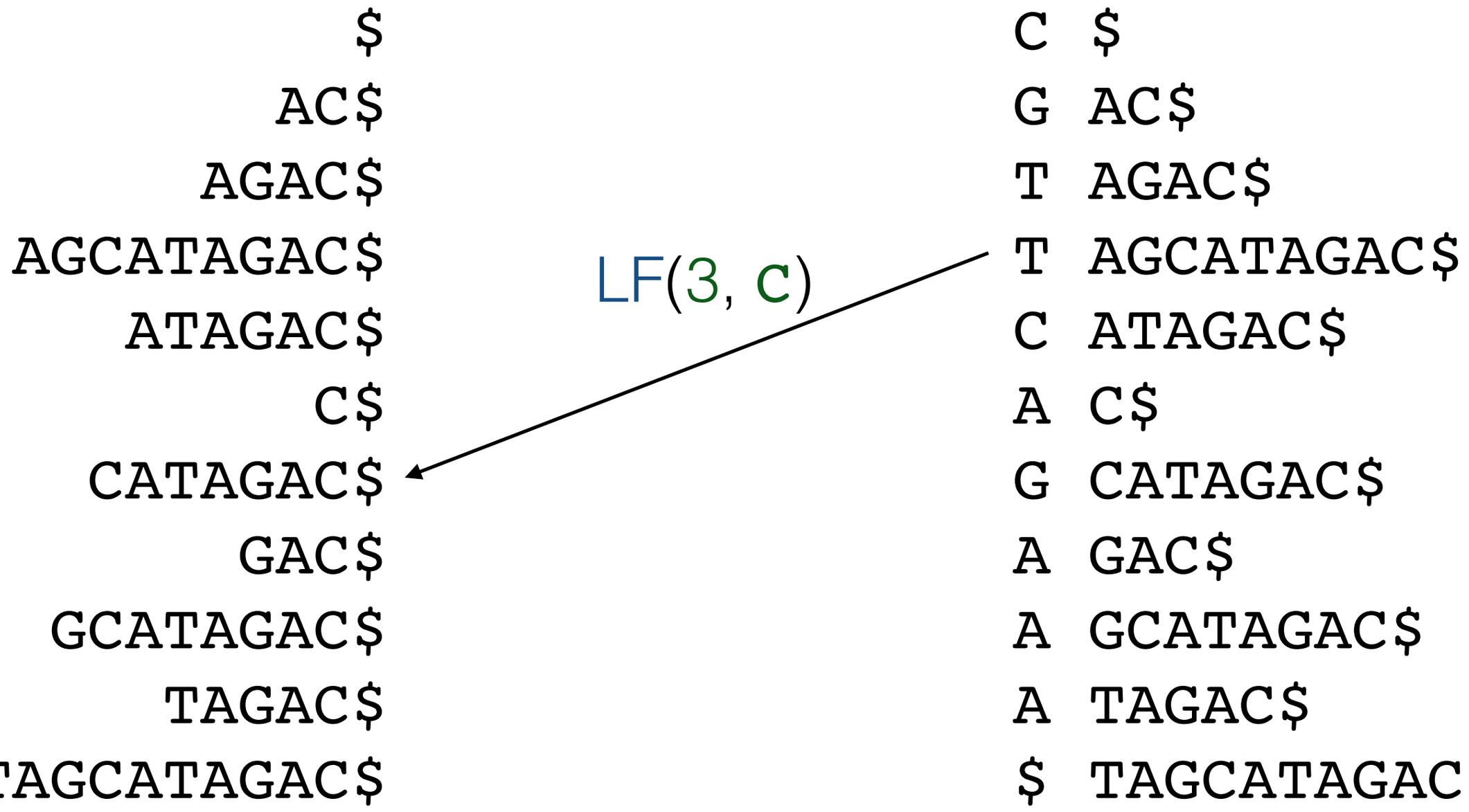
- Add a unique terminator ($) to the end of the text, sort the suffixes in lexicographic order, and output the preceding character for each suffix.

- 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).

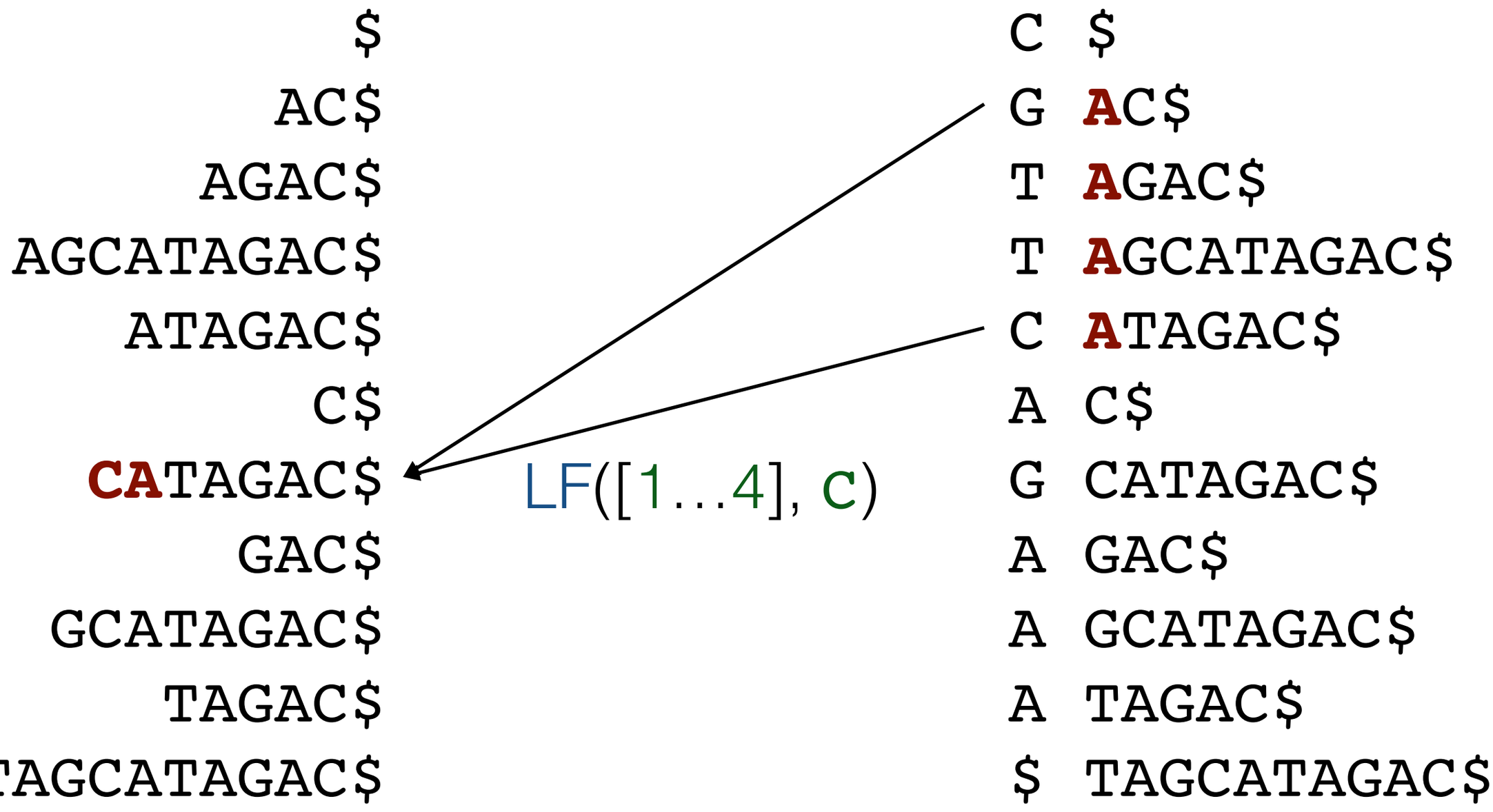
```
                TAGCATAGAC$

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

LF-mapping



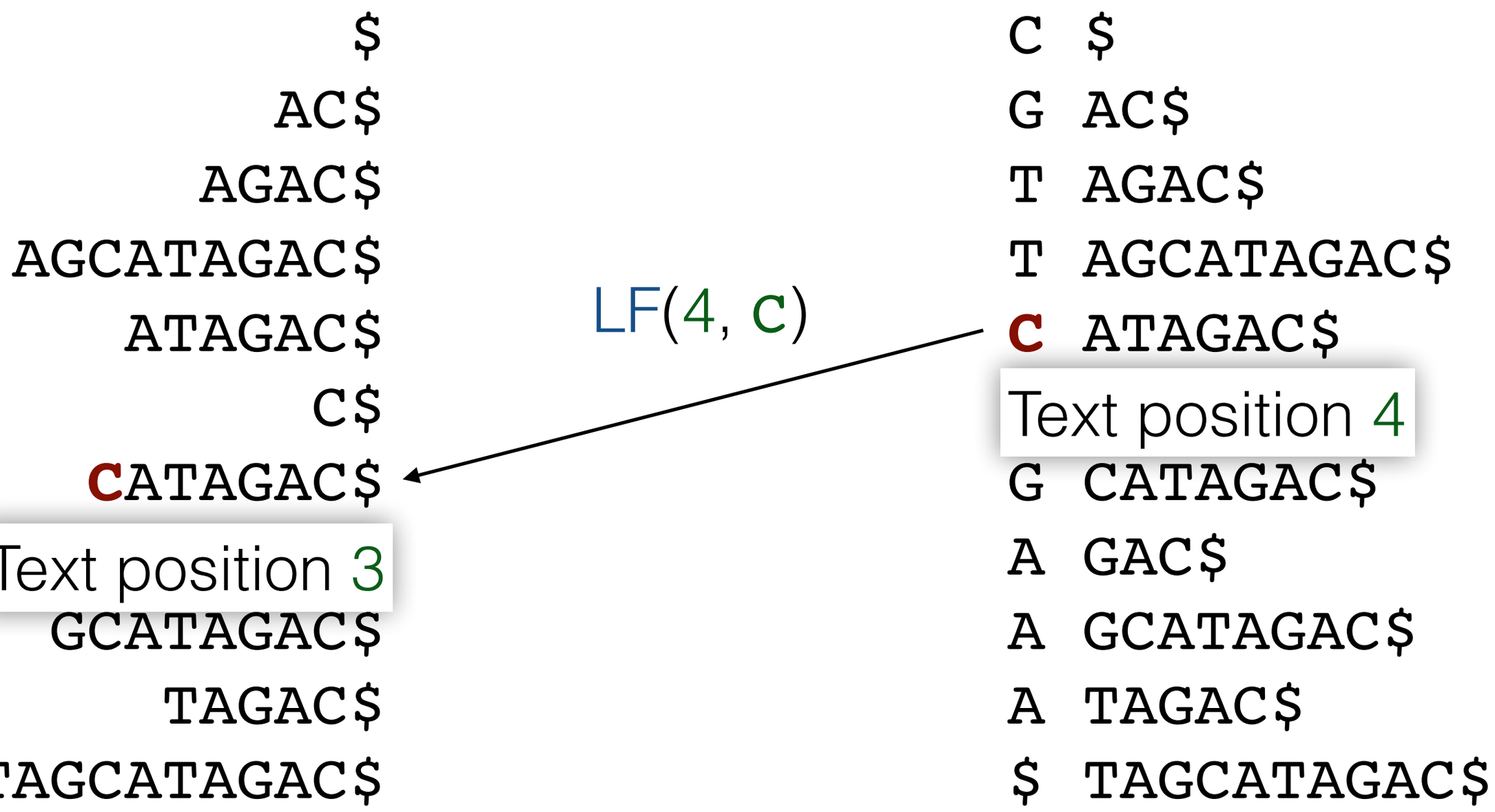
Interpretation: LF(i, c) = C[c] + BWT.rank(i, c) suffixes are **strictly before** the hypothetical suffix.

Backward searching



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

Locating the occurrences



We sample some suffix array pointers and iterate LF-mapping to derive the rest from the samples.

Succinct de Bruijn graphs

Node	BWT	IN	OUT
$$$	A	1	1
A$$	T	1	1
ATA	C	1	1
ATC	C	1	1
ATG	C	1	1
CAT	GT	01	001
CTT	G	1	01
GCA	#	1	1
GCT	#	1	1
GTA	T	1	1
TA$	AG	01	1
TCA	AT	01	1
TGT	AT	01	1
TTC	C	1	1
TTG	C	1	1
#GC	#	1	01
##G	#	1	1
###	$	1	1

- Sort the nodes, write the predecessor labels to BWT, and encode the indegrees and the outdegrees in unary to bitvectors IN and OUT.
- The result is an FM-index for de Bruijn graphs.
- Bowe et al: Succinct de Bruijn graphs. WABI 2012.
- Index size: hundreds of gigabytes for 100 billion kmers.

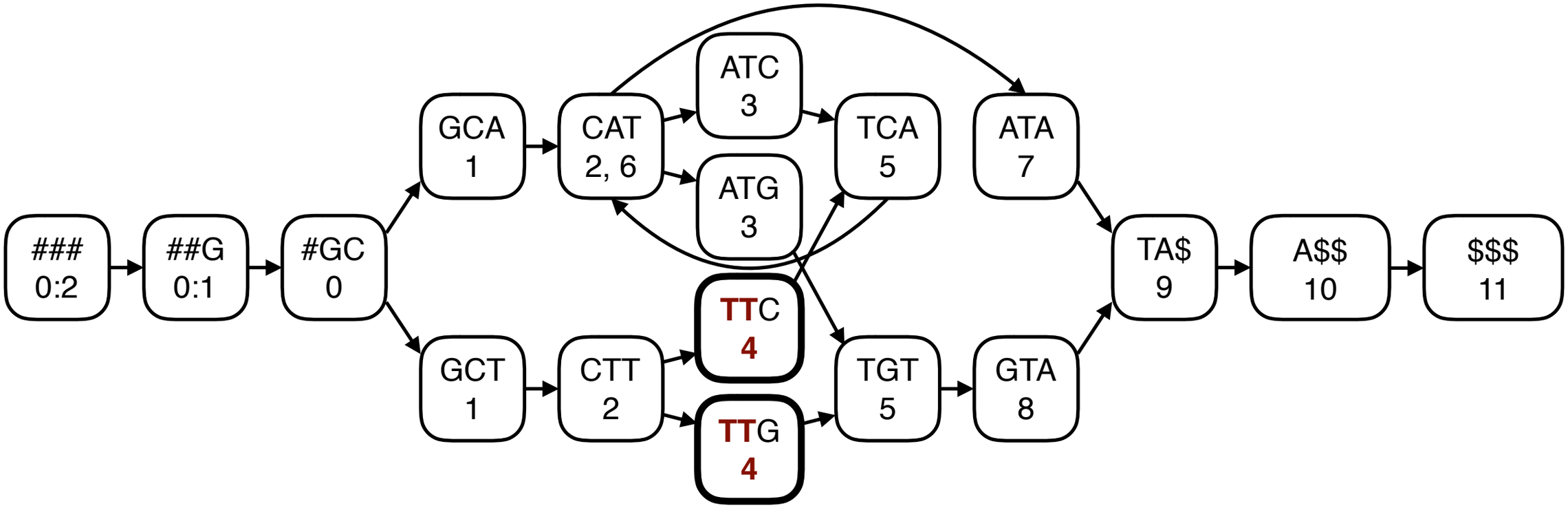
GCSA2

Path graphs

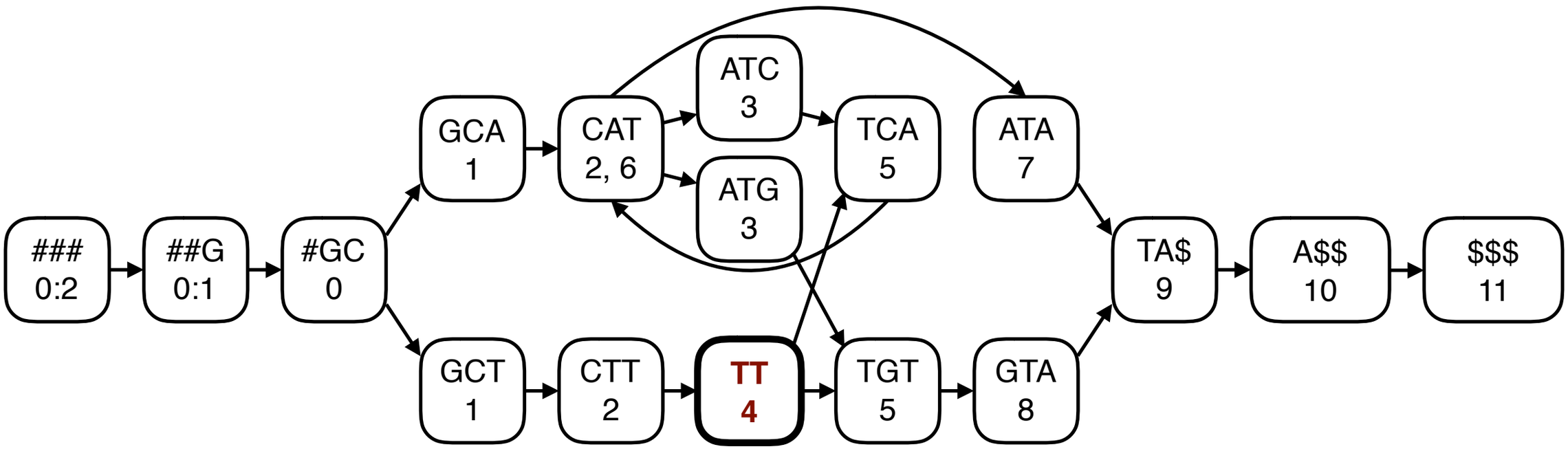
- Path graphs generalize de Bruijn graphs by using any prefix-free set of strings as keys.

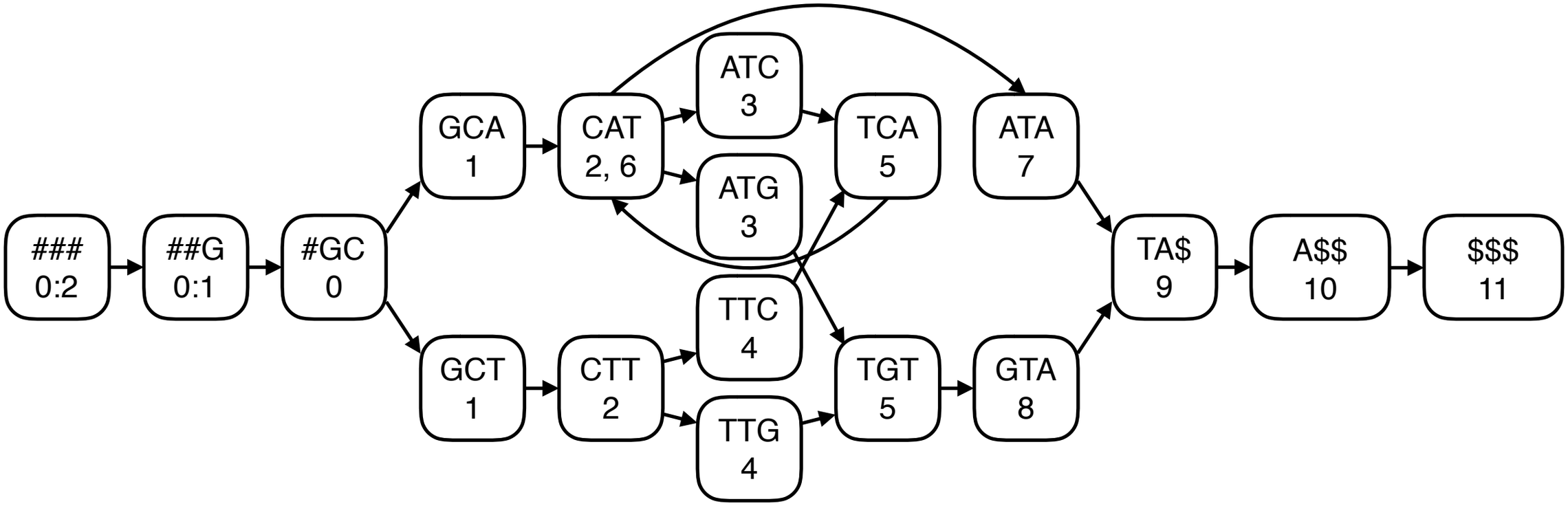
- They compress de Bruijn graphs structurally by merging redundant subgraphs.

- Inspired by: Sirén et al: Indexing Graphs for Path Queries with Applications in Genome Research. TCBB, 2014.

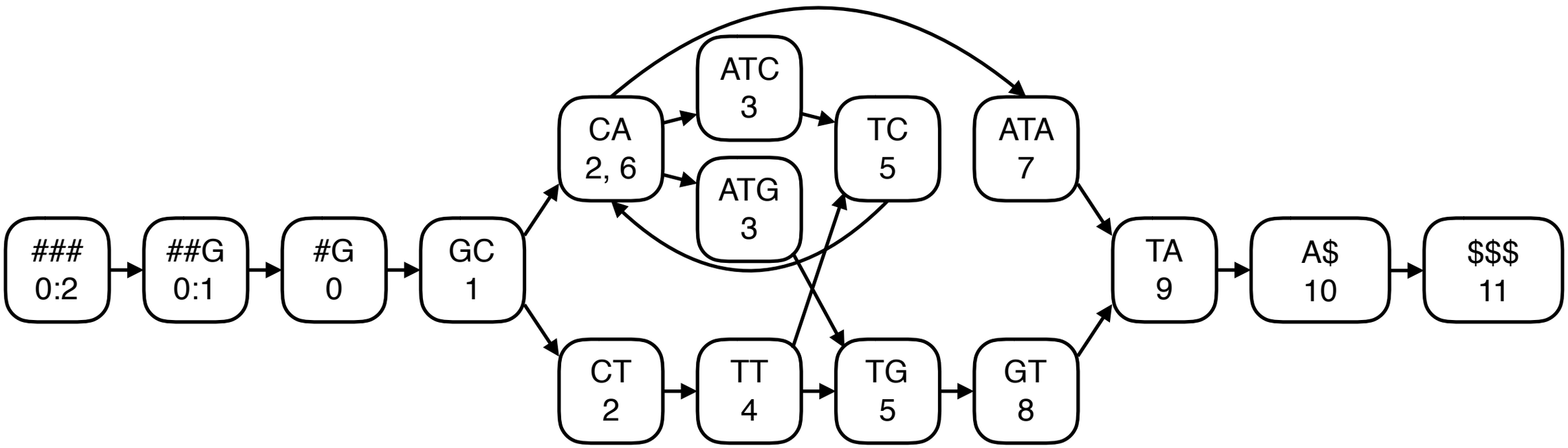


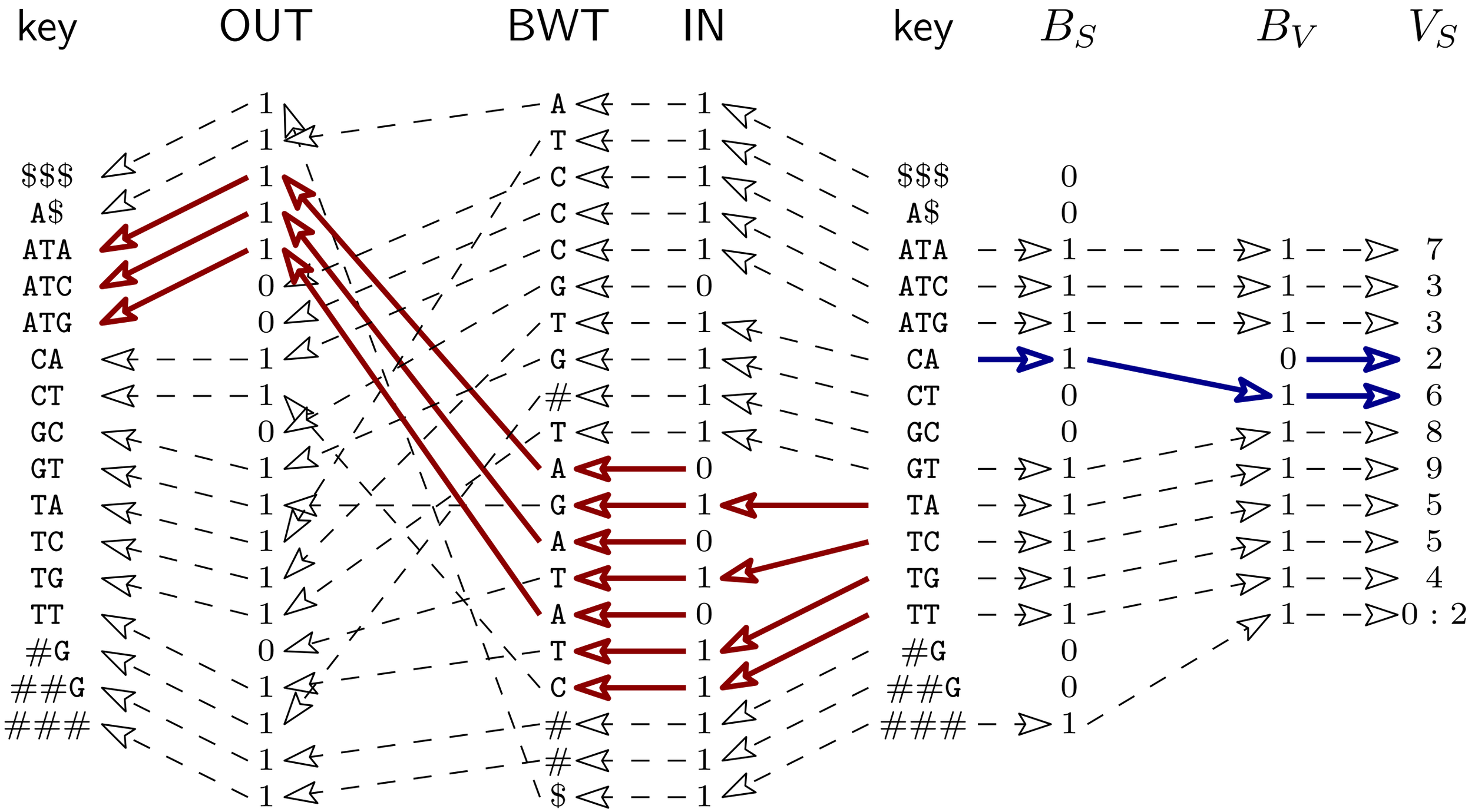
We can merge the nodes sharing a prefix of keys, if the value sets are identical.



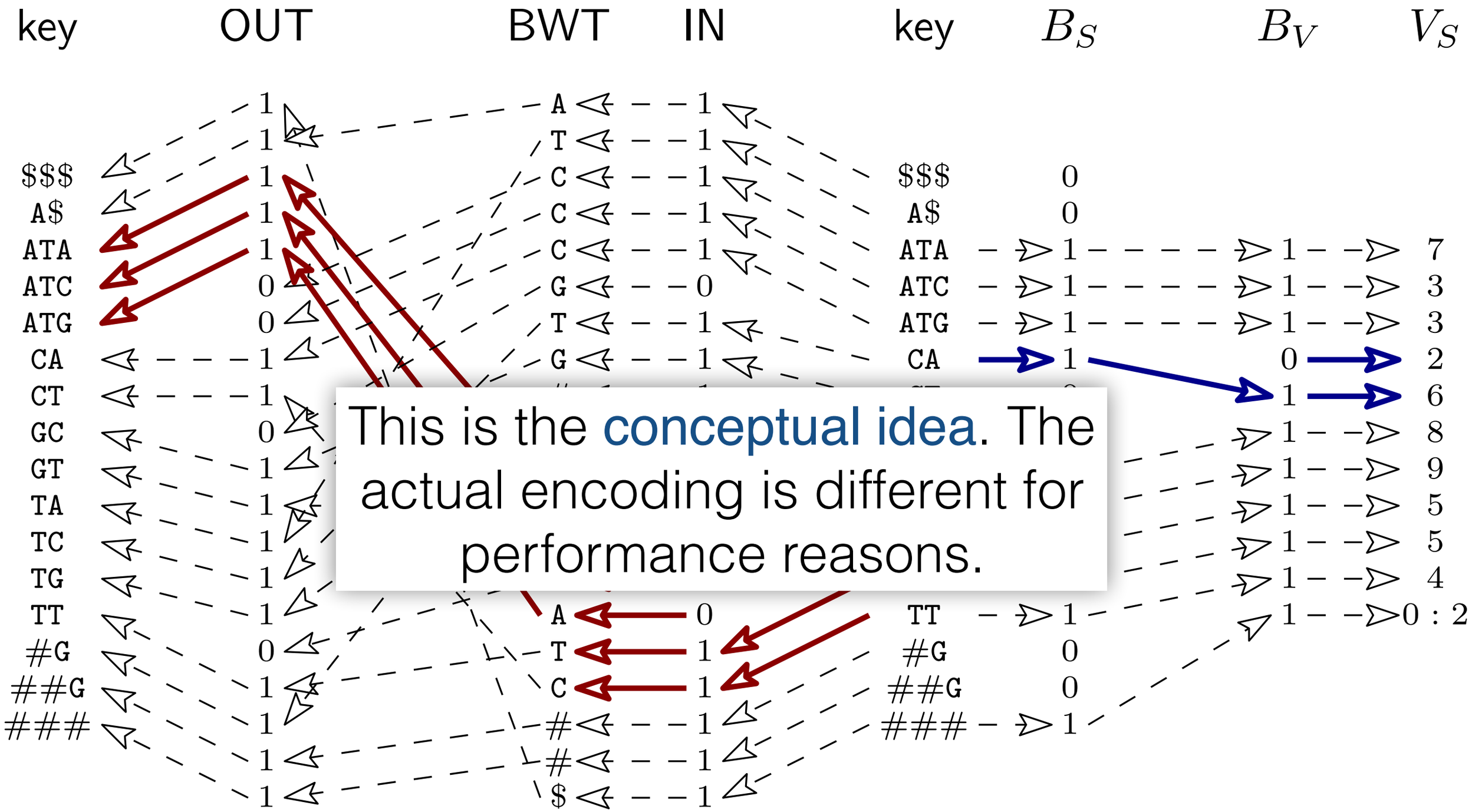


If we keep merging the nodes, we get a (maximally) pruned de Bruijn graph, which behaves intuitively.





We can encode the result in the same way as in the succinct de Bruijn graph.



We can encode the result in the same way as in the succinct de Bruijn graph.

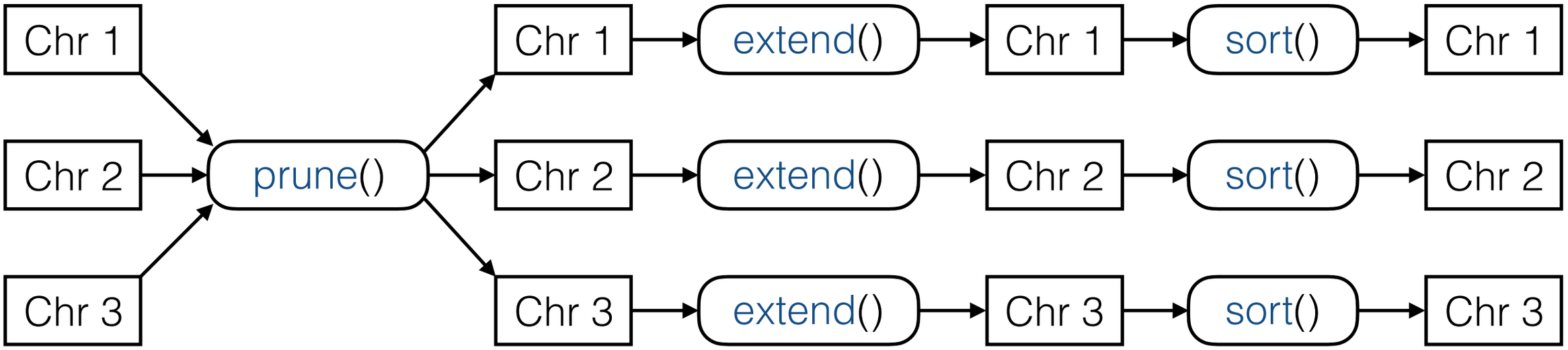
Path graph construction

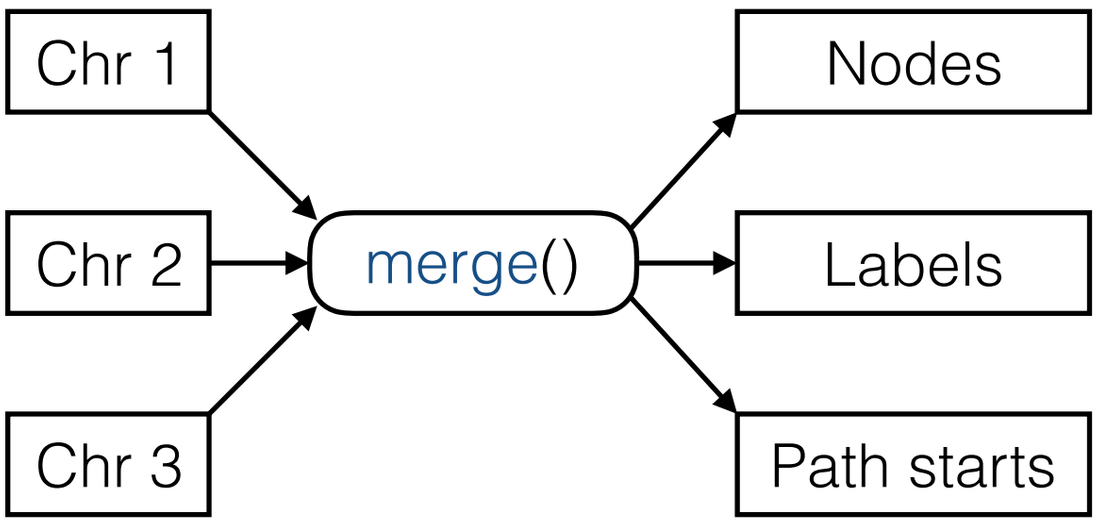
- Start from paths of length k and use a prefix-doubling algorithm to build the pruned de Bruijn graph.

- extend(): Double the path length by joining paths A→B and B→C into paths A→C.

- prune(): If all paths sharing a common prefix of their labels start from the same node, merge them.

- merge(): Merge all paths with the same label, and all paths sharing a prefix if their value sets are identical.





- prune() and merge() merge **sorted files** using a priority queue.
- extend() is done separately for each **chromosome**.
- Memory usage is often determined by extend() for the **most complex** chromosome.

GCSA2 construction

- Prefix-doubling gives us the nodes of the path graph.
- Index construction is essentially about determining the edges.
- There is an edge from X to Y, if Y has a predecessor with label $X[0]$ and one of X and $X[0]Y$ is a prefix of the other.
- One read pointer scans the destination nodes Y, while σ additional pointers scan the source nodes X starting with each character $c \in \sum$.

Path length	16→32	16→64	16→128
Kmers	6.20G	16.7G	**116G**
Nodes	4.37G	5.24G	**5.73G**
Index size	13.2 GB 18.2 bits / kmer	13.5 GB 6.99 bits / kmer	**14.6 GB** **1.08 bits / kmer**
Construction:			
Time	7.44 h	10.4 h	**14.1 h**
Memory	59.8 GB	51.9 GB	**52.3 GB**
Disk	387 GB	415 GB	**478 GB**
I/O:			
Read	1.37 TB	2.03 TB	**2.78 TB**
Write	0.88 TB	1.51 TB	**2.25 TB**

1000GP human variation (forward strand only)

`vg mod -p -l 16 -e 4 | vg mod -S -l 100`

32 cores, 256 GB memory, distributed Lustre file system

Index	kmers	Matched	find()	locate()
GCSA2	351584	347453	4.75 µs	5.85 µs
BWA	351584	320764	3.64 µs	4.65 µs
csa_wt	351584	301538	6.00 µs	2.43 µs

GCSA2: Order-128 index for the pruned variation graph

BWA: The FM-index from BWA v0.7.15 for the reference and its reverse complement

csa_wt: Fast FM-index from SDSL for the reference

Average time for find queries (per query) and locate queries (per distinct occurrence) with 16-mers extracted from the non-pruned variation graph.

Conclusions

- The design of a path index is a trade-off between index size, query performance, maximum query length, and ignoring complex regions of the graph.

- GCSA2 prioritizes performance and size, while supporting queries of length up to 128.

- It uses a de Bruijn graph as a kmer index, compresses it by merging redundant subgraphs, and encodes the result as an FM-index.

- The implementation is available at https://github.com/jltsiren/gcsa2.