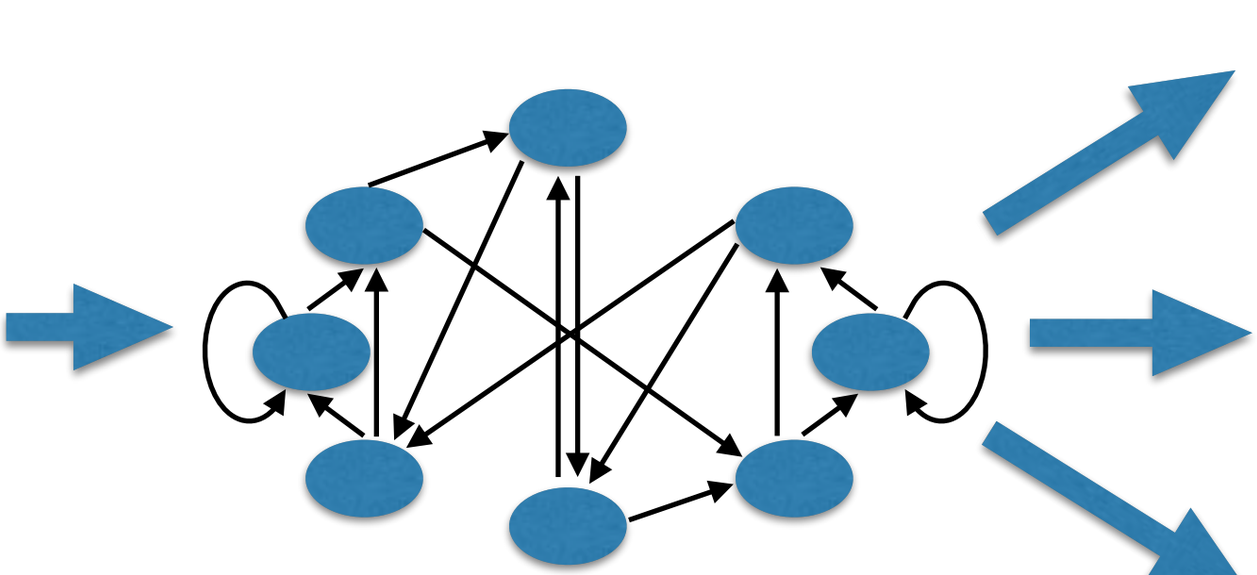
Compact Representations of Annotated de Bruijn Graphs

Prashant Pandey
Stony Brook University, NY, USA

De Bruijn graphs are ubiquitous

[Pevzner et al. 2001, Zerbino and Birney, 2008, Simpson et al. 2009, Grabherr et al. 2011, Compeau et al. 2011, Schulz et al. 2012, Chang et al. 2015, Kannan et al. 2016, Liu et al. 2016, Carvalho et al. 2016, Salmela et al. 2016, Koren et al. 2017]

Raw sequencing data



De Bruijn graph

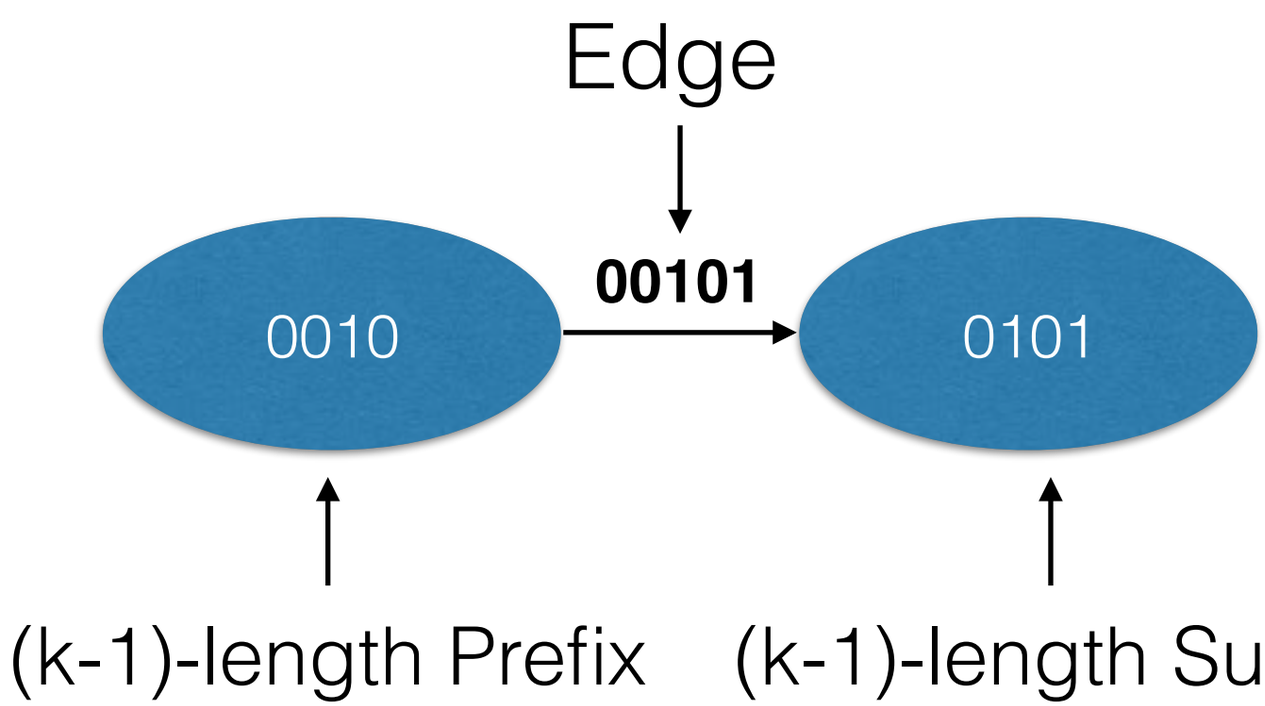
Sequence search

Short/Long reads transcriptome assembly

Long reads error correction

A de Bruijn graph is the data representation at the heart of a lot of sequence analyses.

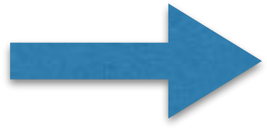
De Bruijn graph (dBG)



An edge is a length-k string connecting its two k-1 substrings.

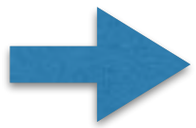
De Bruijn graph (dBG)

A ***read*** is a string of bases over the DNA alphabet A, C, T, and G.



CACTGAACTCACTGACTCA

A **k-mer** is a substring of length k. Here, k is 5.

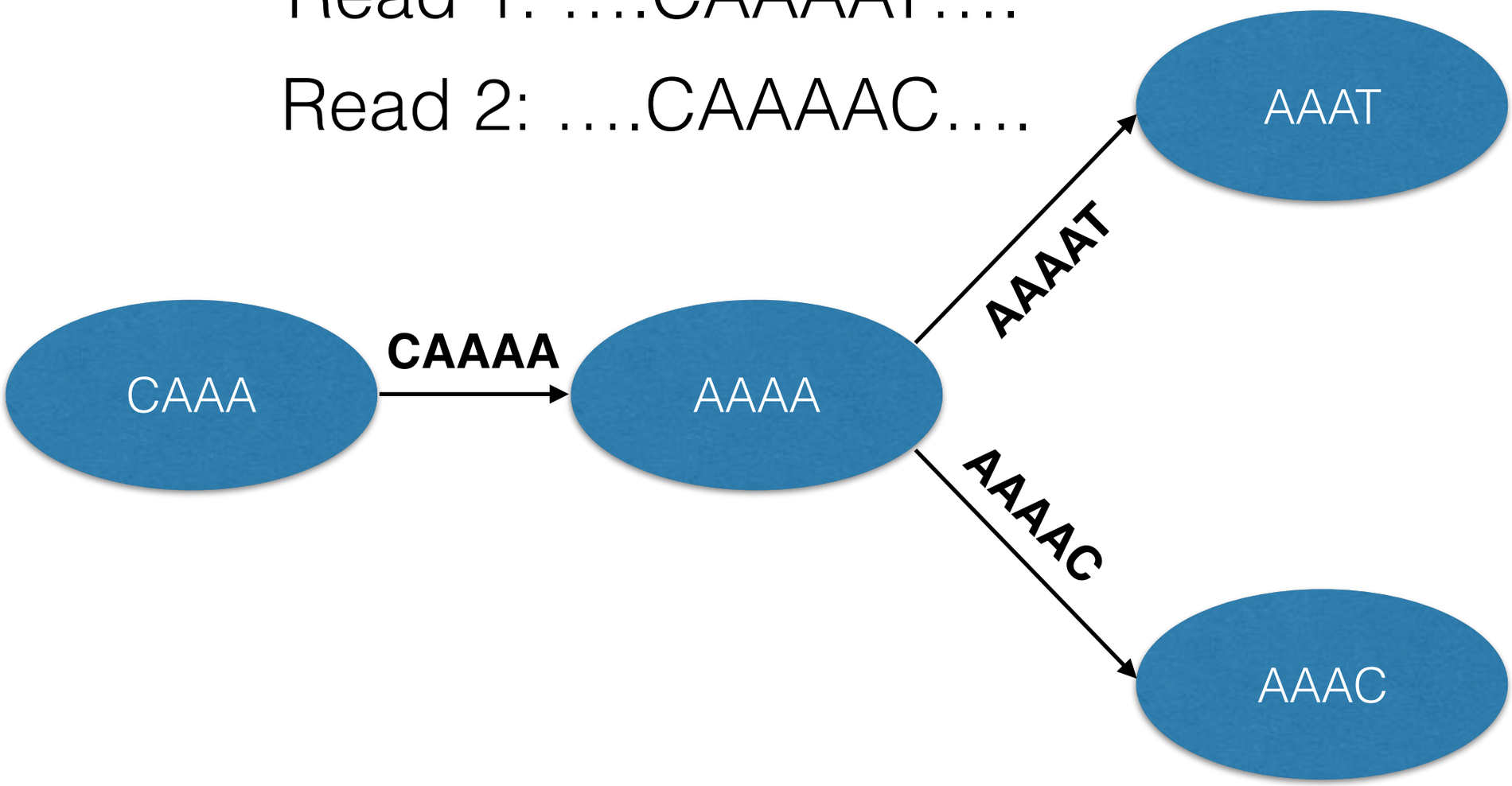


CACTG
ACTGA
CTGAA
TGAAC
GAACT
AACTC
ACTCA
CTCAC
...

De Bruijn graph (dBG)

Read 1: ….CAAAAT….

Read 2: ….CAAAAC….



Annotated De Bruijn graphs

- Topology-only de Bruijn graphs are not adequate for downstream applications.

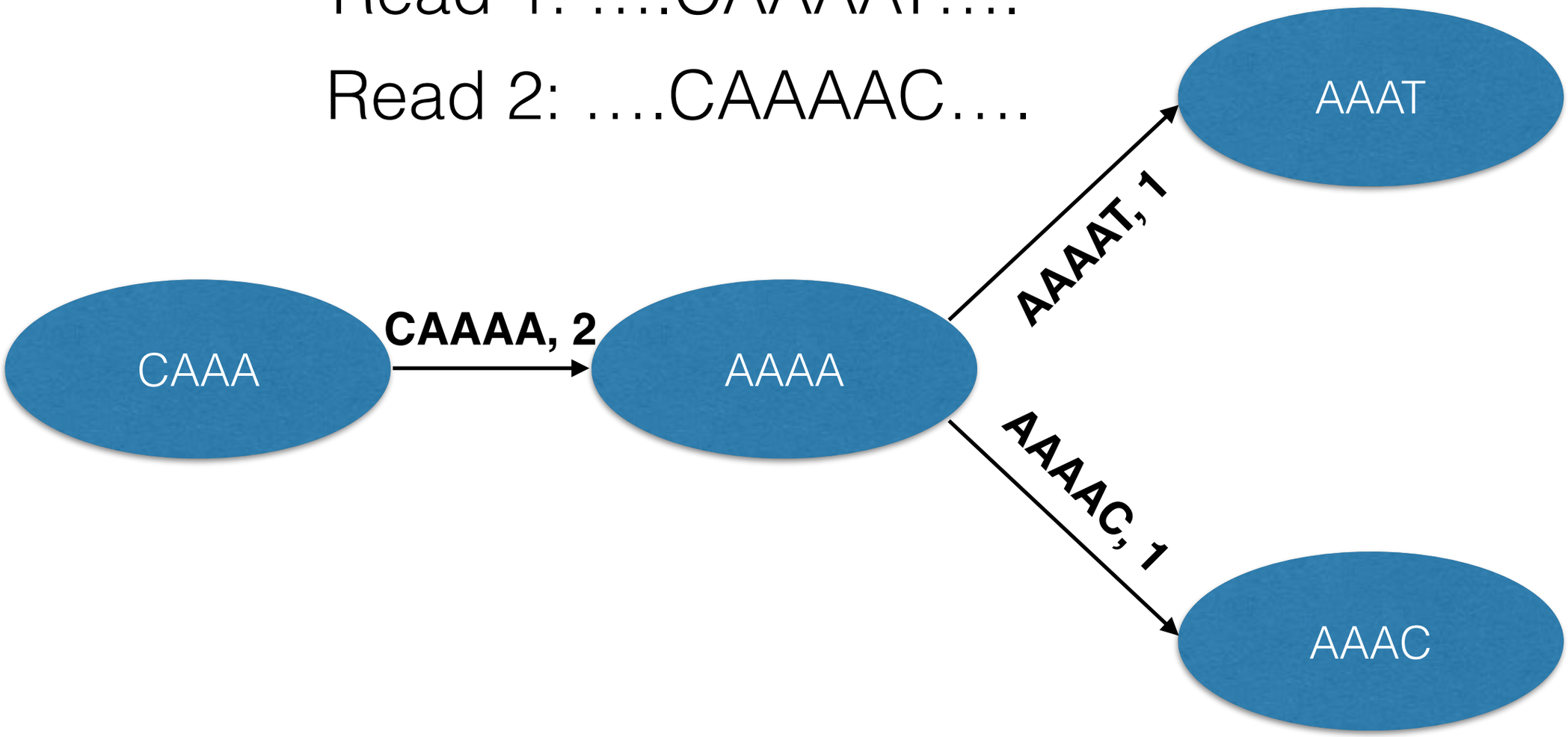
- Abundance information of each k-mer is critical for transcriptome assembly.

- Information about samples in which a k-mer is present in a union de Bruijn graph of multiple samples is critical for variant discovery.

Weighted de Bruijn graph (WdBG)

Read 1: ….CAAAAT….

Read 2: ….CAAAAC….

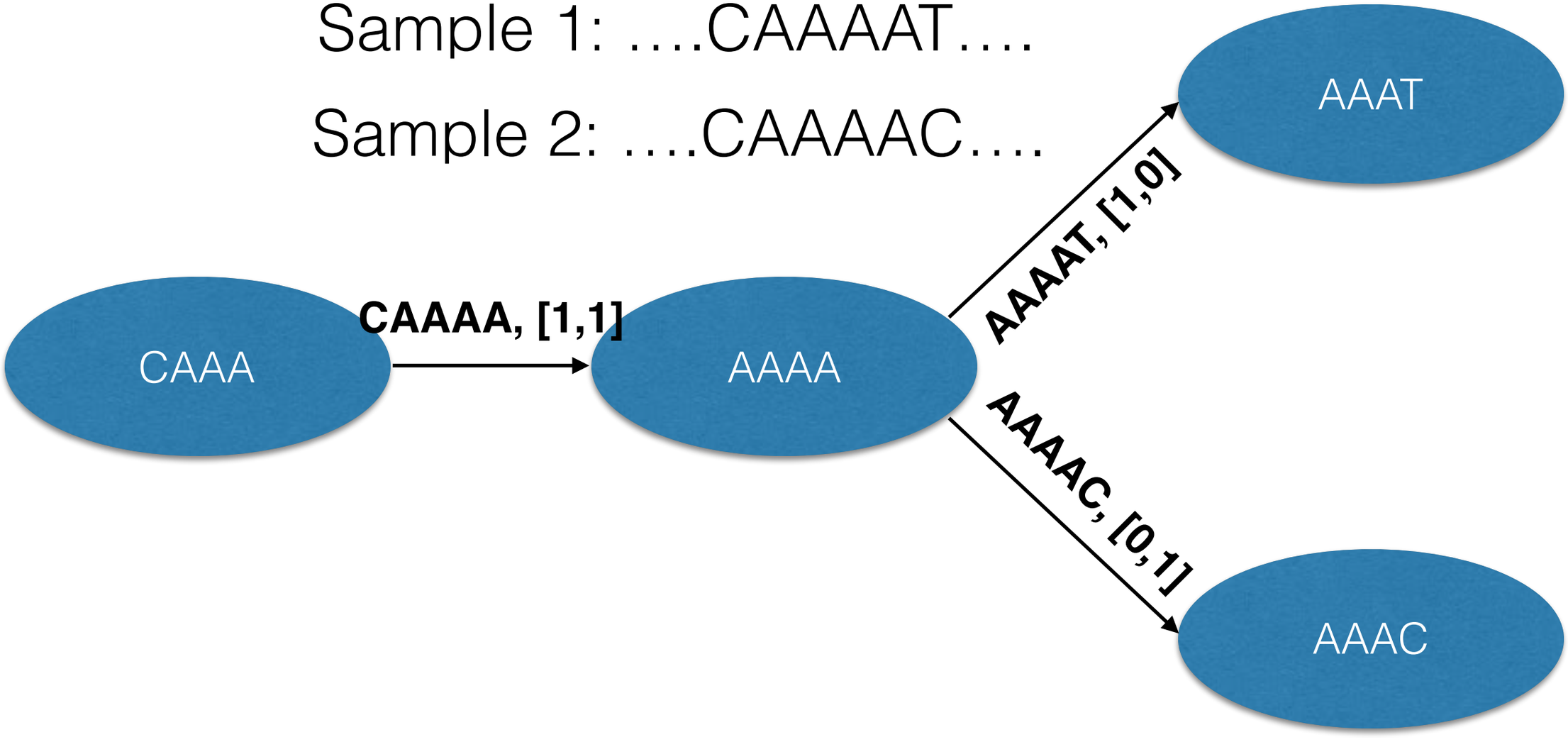


A weighted de Bruijn graph associates each edge (k-mer) its abundance in the underlying dataset.

Colored de Bruijn graph (CdBG)

Sample 1: ….CAAAAT….

Sample 2: ….CAAAAC….



A colored de Bruijn graph is a union graph of multiple samples, where the identity of each sample is retained by coloring those edges present in a sample.

Measuring annotated dBG representation

de Bruijn graphs store only k-mers, memory usage scales with the number of unique k-mers.

Human genome (few Billion k-mers): >100 GB
Soil metagenomes (few Million species): Few TBs

Beefy server machines are needed to perform weighted de Bruijn graph analysis.

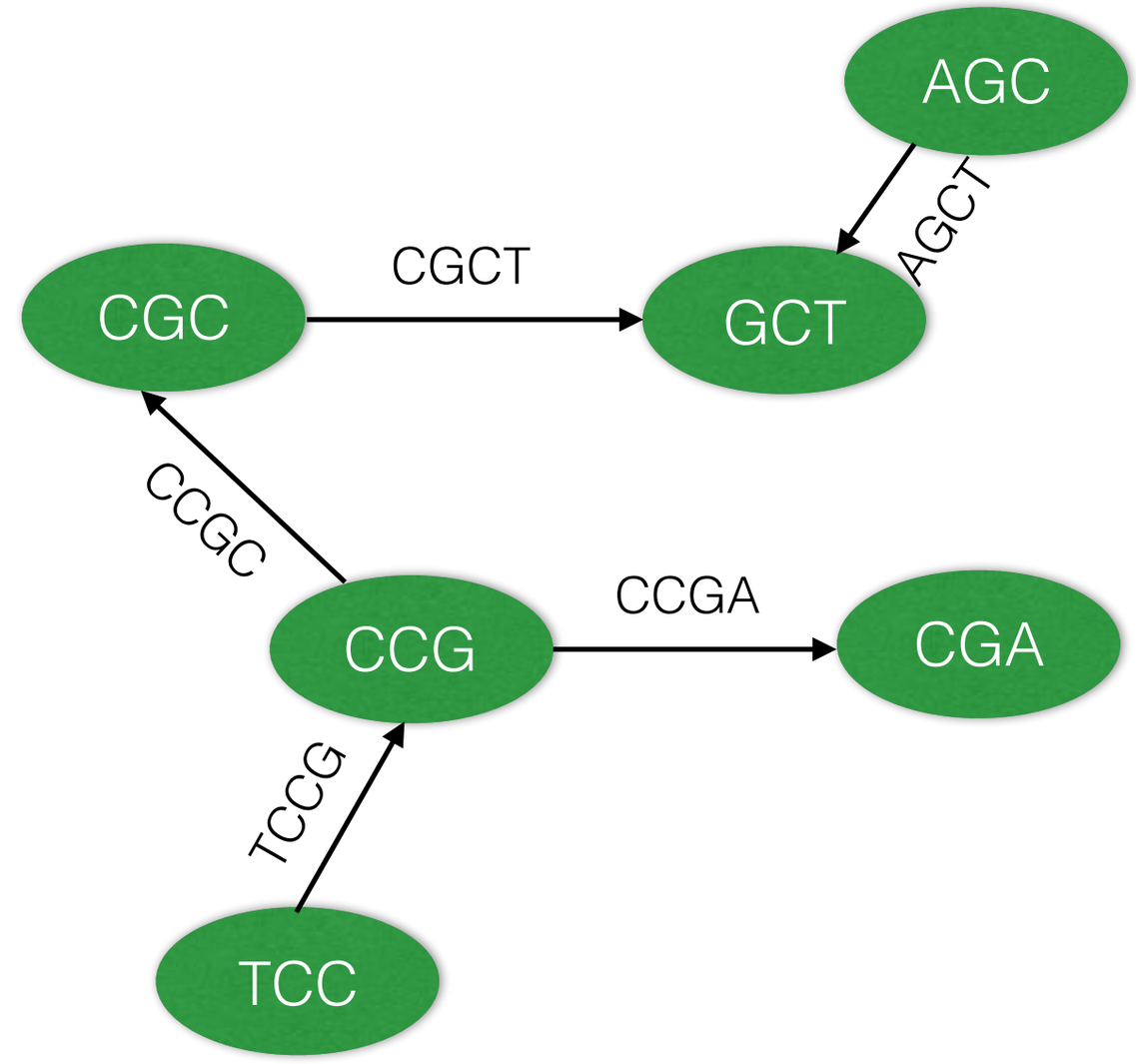
In this talk

- A compact representation of annotated de Bruijn graphs.
    - It would enable transcriptome assembly on machines with less resources.
    - It would also enable assembly of fundamentally large datasets that wasn't possible before on a single machine.
    - It would also enable sequence-level searches and variation detection on all of the available RNA-seq experiments in SRA.

dBG as a set

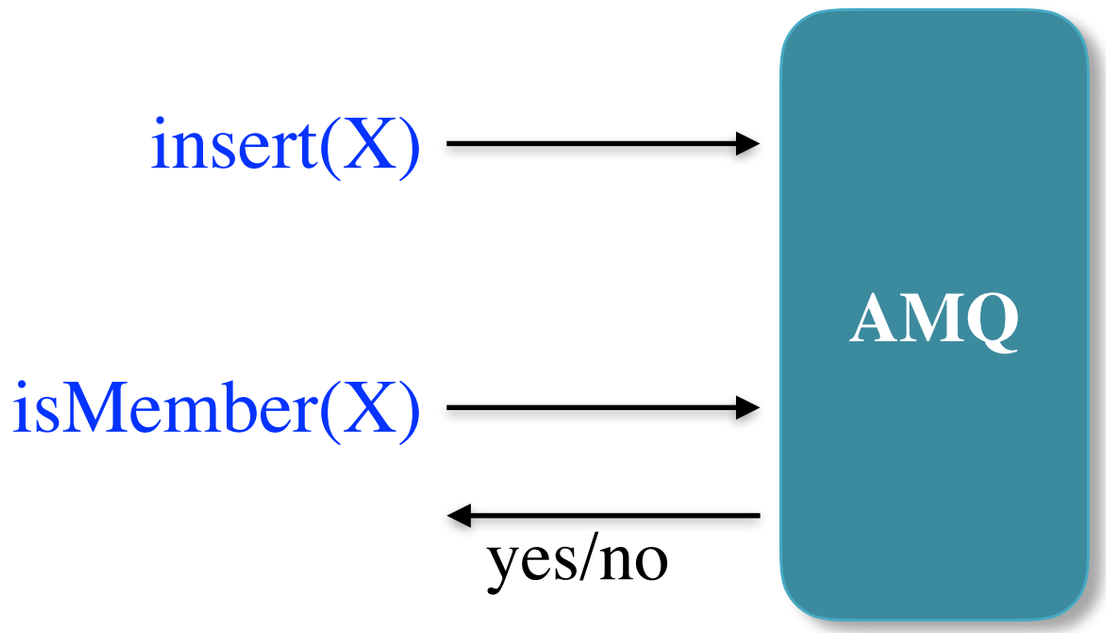
Set
TCCG
CCGC
CCGA
CGCT
AGCT

(Edges)



de Bruijn graph

Approximate Membership Query
(AMQ)

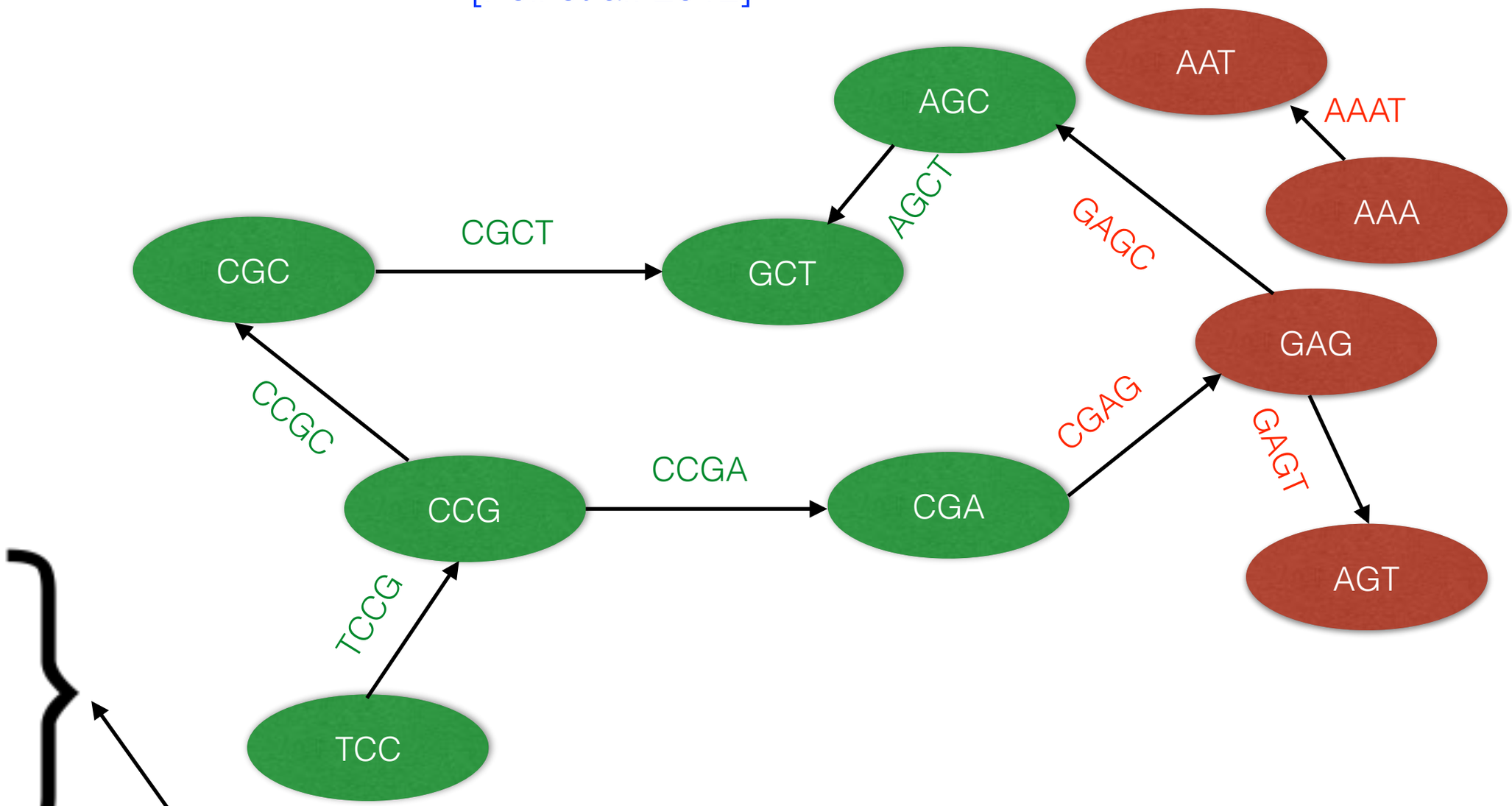


- An AMQ is a lossy representation of a set.
- Operations: inserts and membership queries.
- Compact space:
  - Often taking < 1 byte per item.
  - Comes at the cost of occasional false positives.

Probabilistic de Bruijn graph

[Pell et al. 2012]

Bloom filter
TCCG
CCGC
CCGA
CGCT
AGCT
GAGC
CGAG
GAGT
AAAT



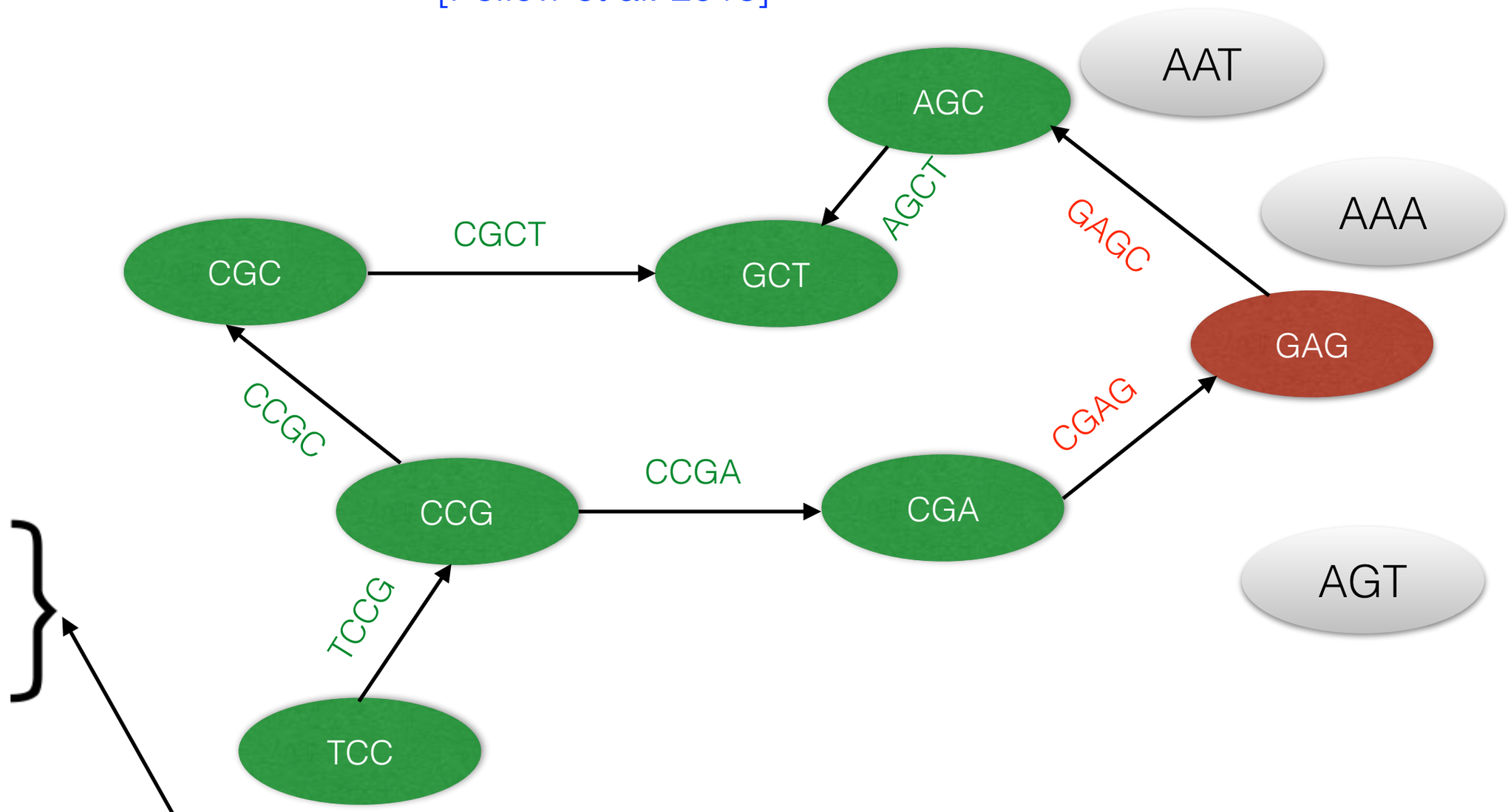
Topological errors

Representing a dBG using a Bloom filter.

Probabilistic de Bruijn graph

[Pellow et al. 2016]

Bloom filter
TCCG
CCGC
CCGA
CGCT
AGCT
GAGC
CGAG
~~GAGT~~
~~AAAT~~



Topological errors

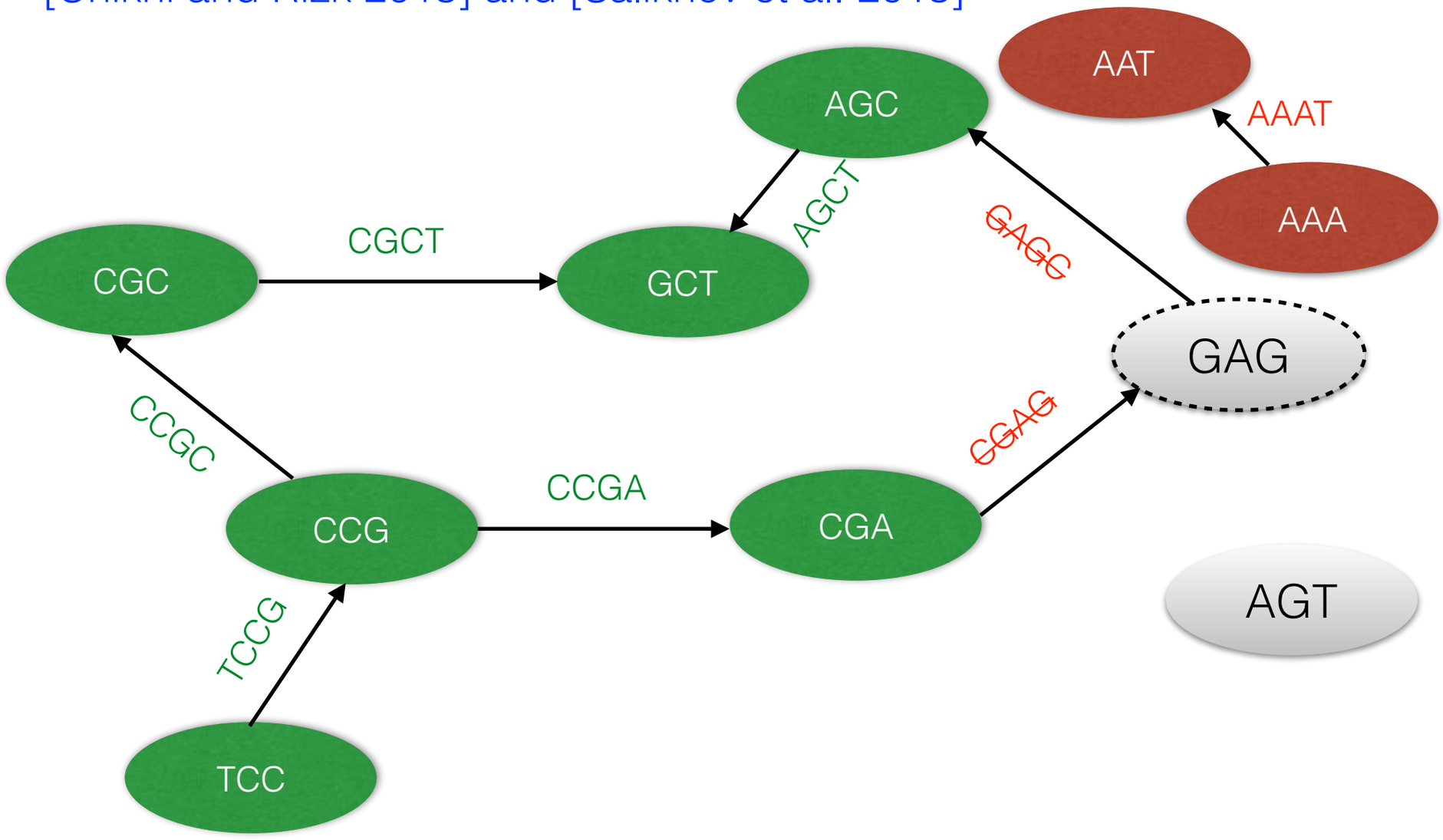
Showed how to exploit redundancy in k-mers to reduce the false-positive rate of the Bloom filter without increasing the space.

Exact de Bruijn graph

[Chikhi and Rizk 2013] and [Salikhov et al. 2013]

Bloom filter
TCCG
CCGC
CCGA
CGCT
AGCT
~~GAGC~~
~~CGAG~~
~~GAGT~~
~~AAAT~~

Hash table
GAGC
CGAG

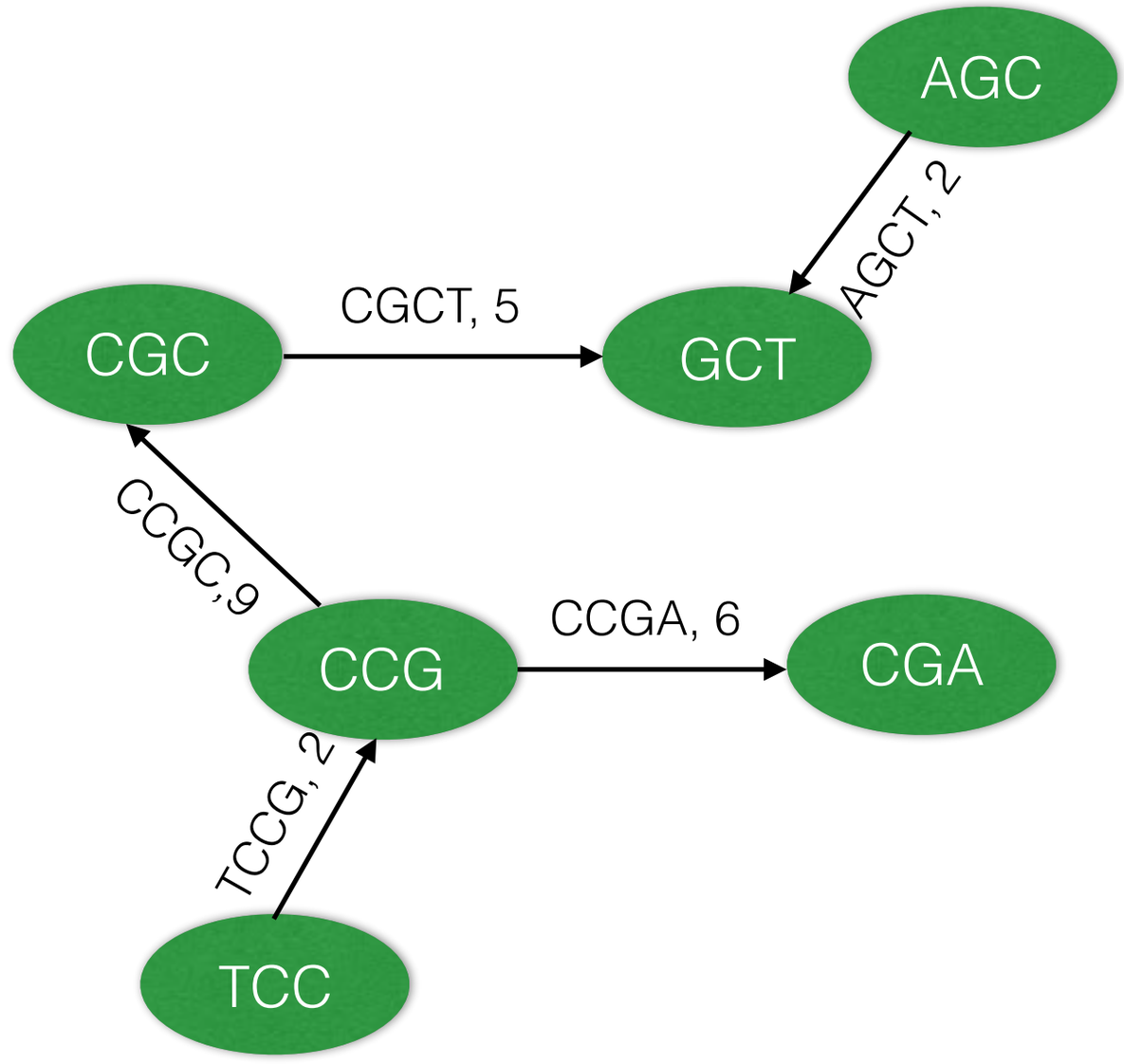


Critical false-positive k-mers

They showed how to convert a probabilistic representation into an exact one using a small and exact auxiliary data structure.

WdBG as a multiset

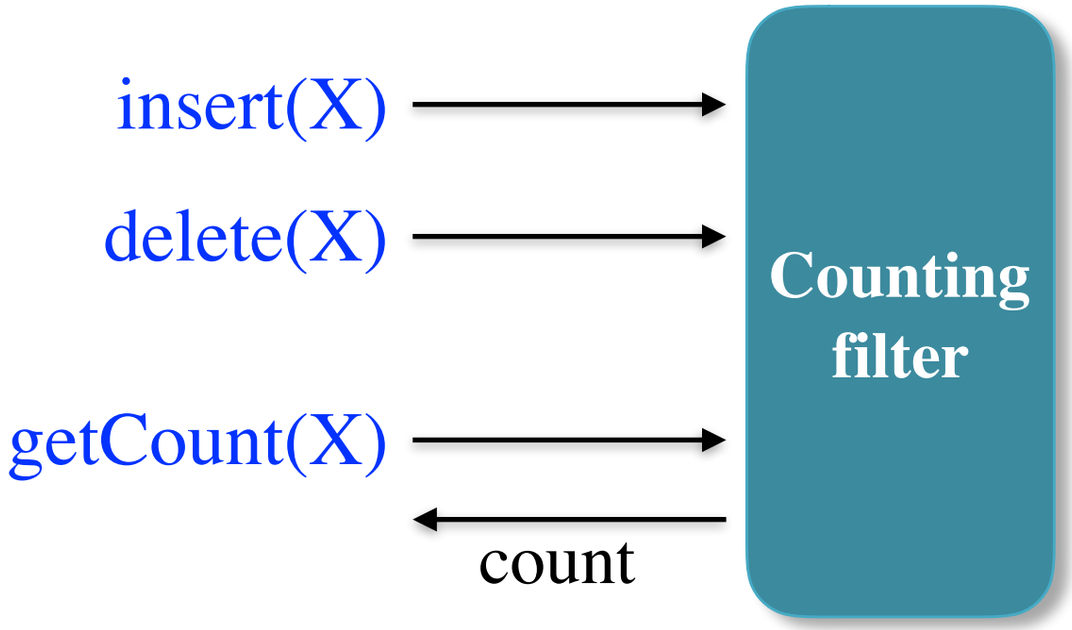
MultiSet
TCCG, 2
CCGC, 9
CCGA, 6
CGCT, 5
AGCT, 2



(Edge, Abundance)

Weighted de Bruijn graph

Counting filters: AMQs for multisets



- A counting filter is a lossy representation of a multiset.
- Operations: inserts, count, and delete.
- Generalizes AMQs
  - False positives ≈ over-counts.
- **Counting quotient filter**

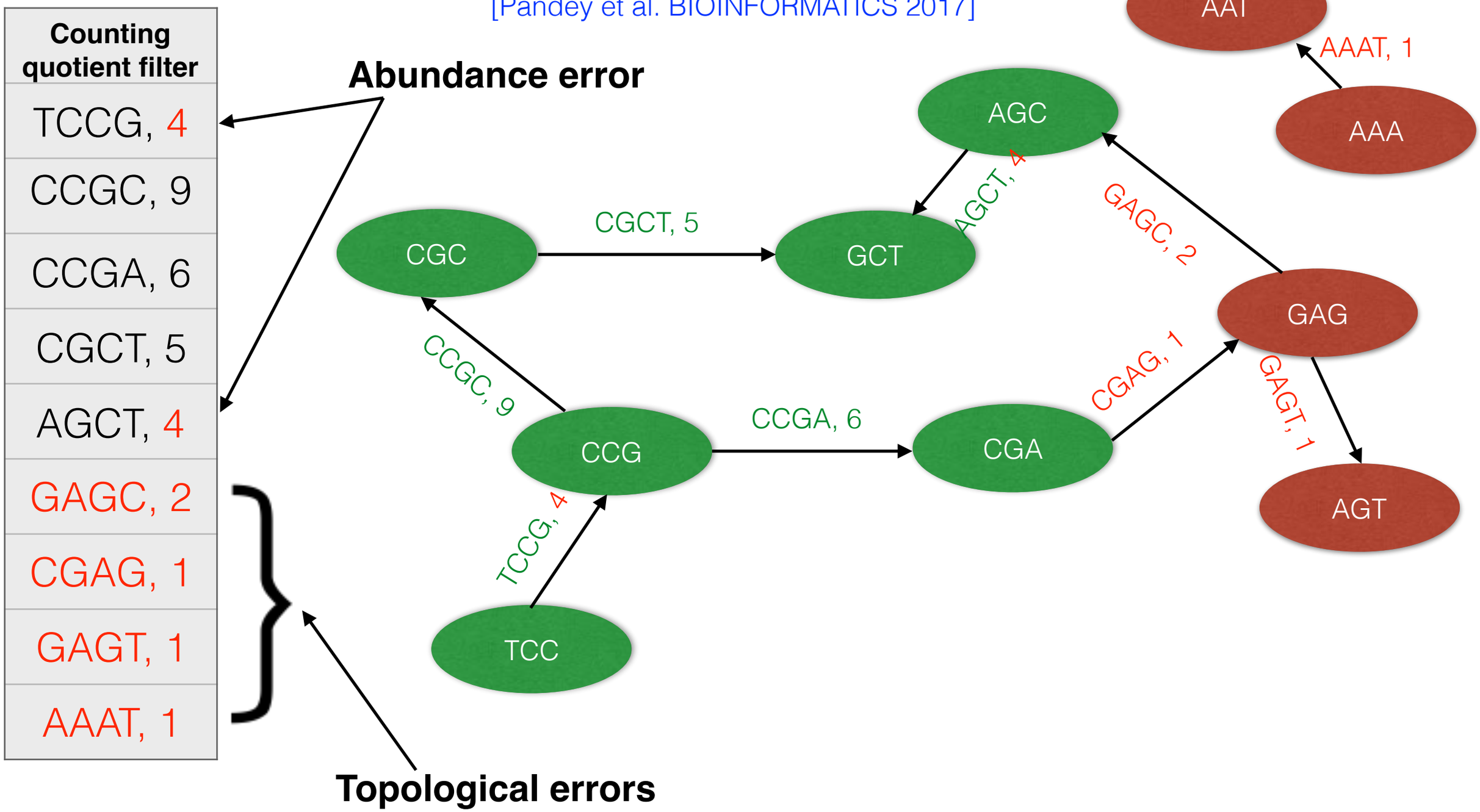
The counting quotient filter

[Pandey et al. SIGMOD 2017]

- Smaller than many non-counting AMQs
  - Bloom, cuckoo [Fan et al., 2014], and quotient [Bender et al., 2012] filters.
- Uses variable-sized counters to handle skewed data sets efficiently.
- Good cache locality
- Deletions
- Dynamically resizable
- Mergeable

Squeakr

[Pandey et al. BIOINFORMATICS 2017]



Approximate weighted de Bruijn graph.

deBGR

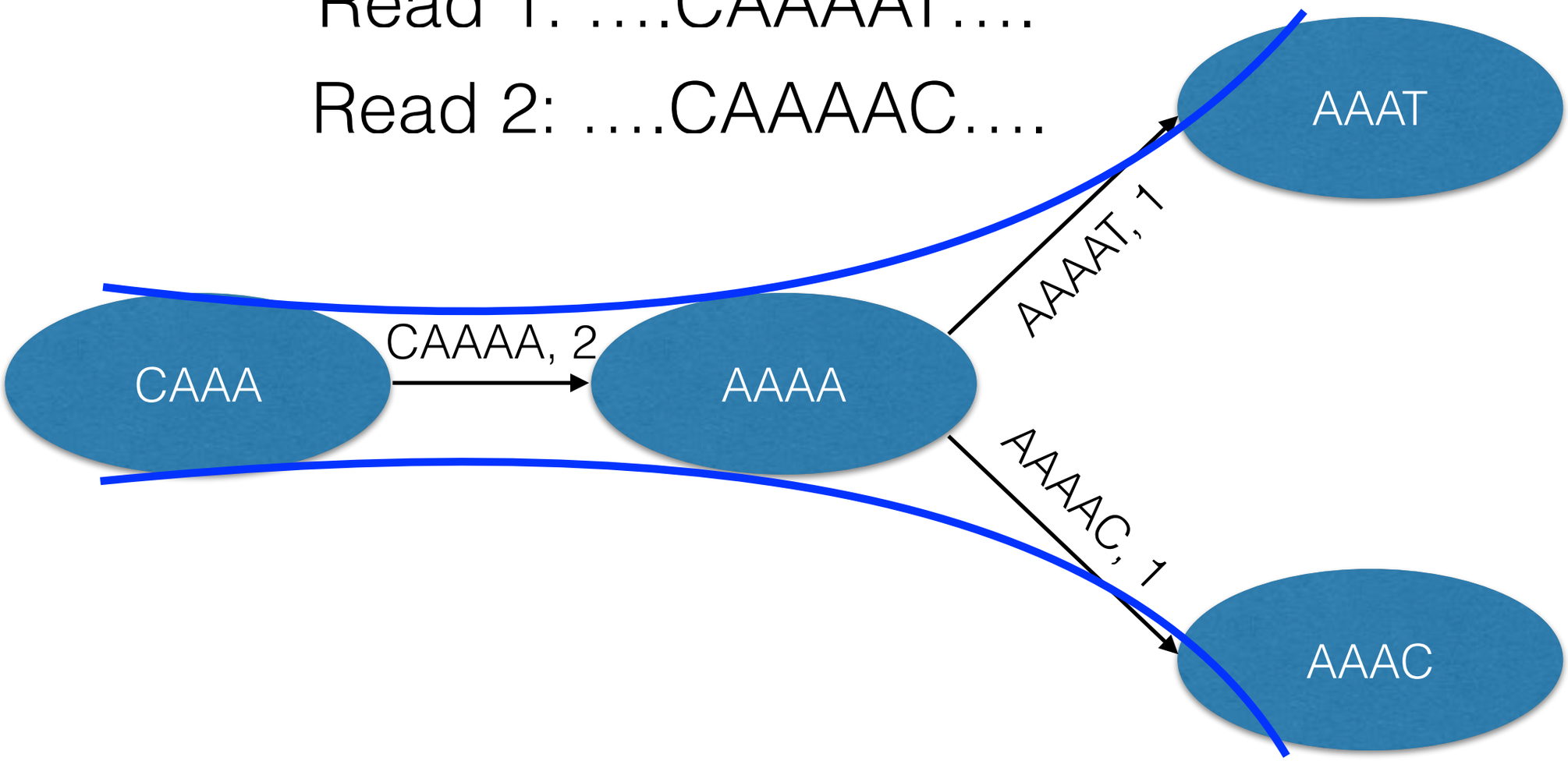
[Pandey et al. ISMB 2017]

- An **exact representation** of the weighted de Bruijn graph.
  - An algorithm that uses counts in the approximate representation in an AMQ to iteratively **self-correct approximation errors**.
  - It corrects both kinds of errors, **abundance and topological errors** and supports **membership queries**.
  - It takes 18-28% more space than the approximate representation and has **no errors**.

A weighted de Bruijn graph invariant

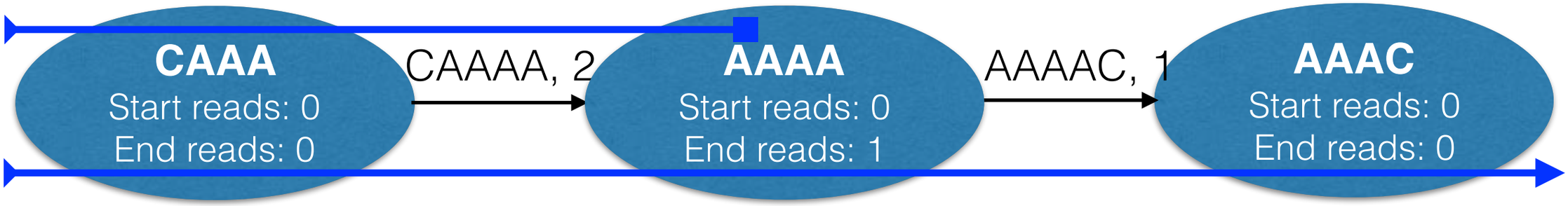
Read 1: ….CAAAAT….

Read 2: ….CAAAAC….



Total incoming abundance = Total outgoing abundance

A weighted de Bruijn graph invariant



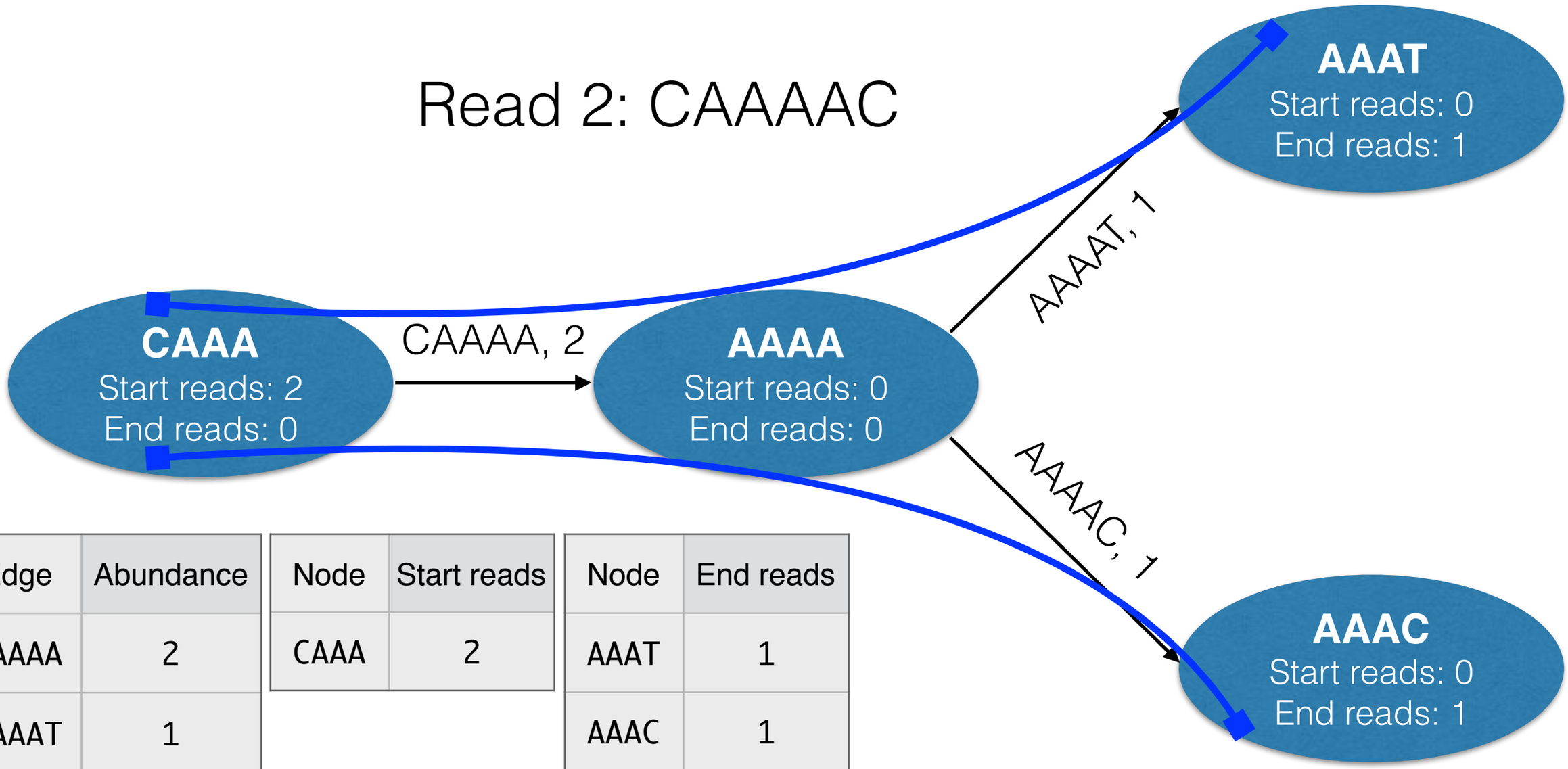
Total incoming abundance = Total outgoing abundance*

***After accounting for read starts and ends.**

WdBG representation in deBGR

Read 1: CAAAAT

Read 2: CAAAAC

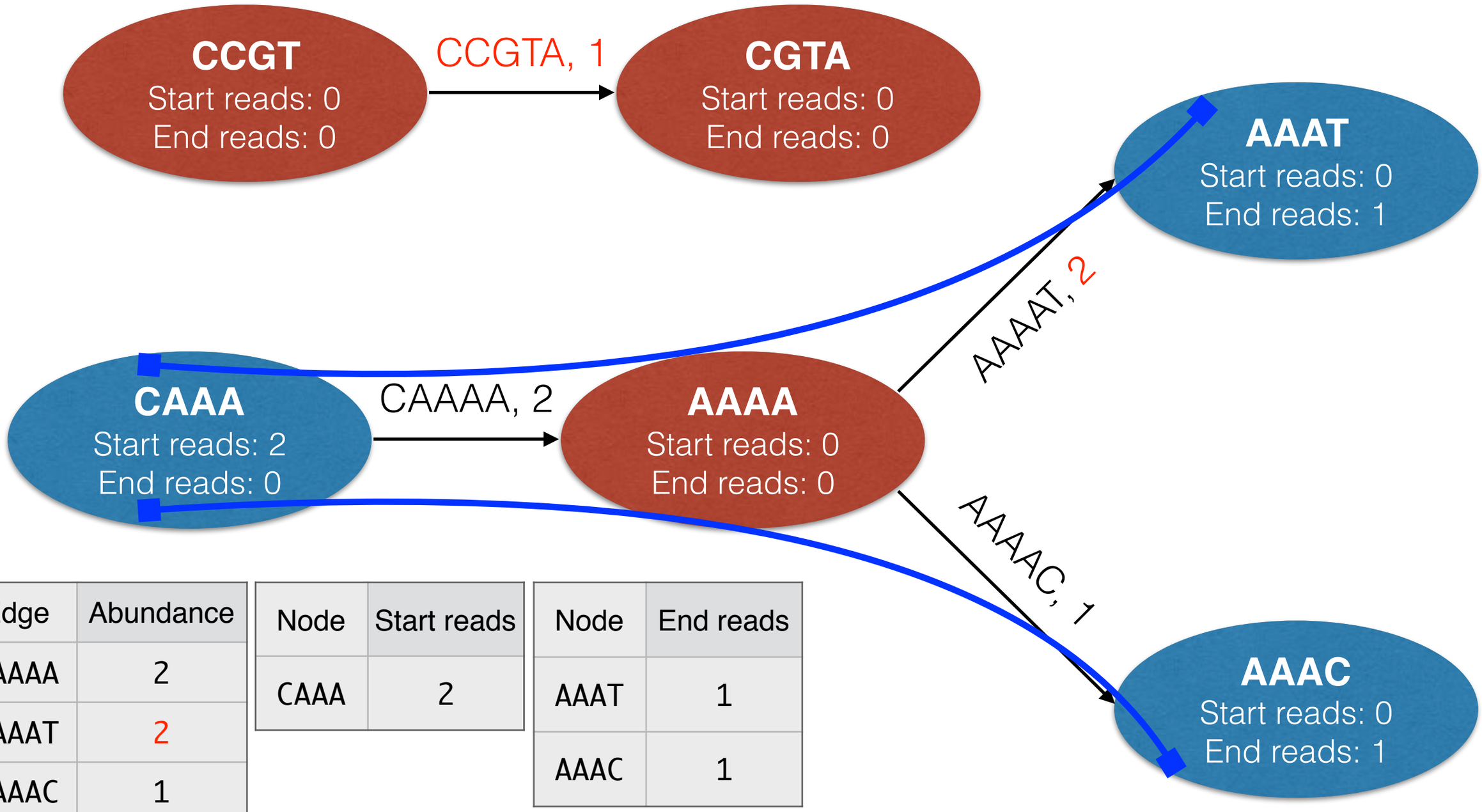


Edge	Abundance
CAAAA	2
AAAAT	1
AAAAC	1

Node	Start reads
CAAA	2

Node	End reads
AAAT	1
AAAC	1

WdBG representation in deBGR

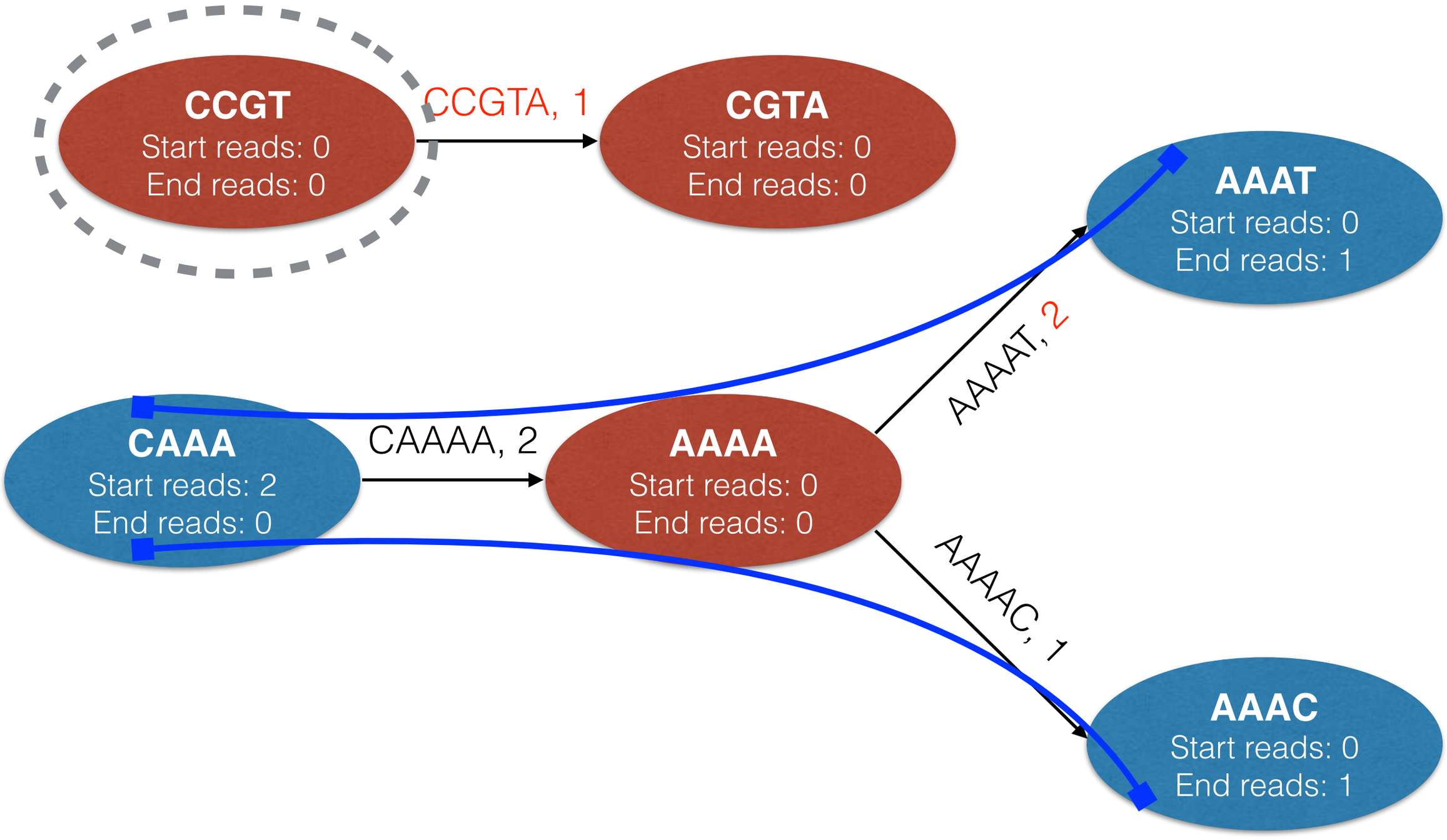


Edge	Abundance
CAAAA	2
AAAAT	2
AAAAC	1
CCGTA	1

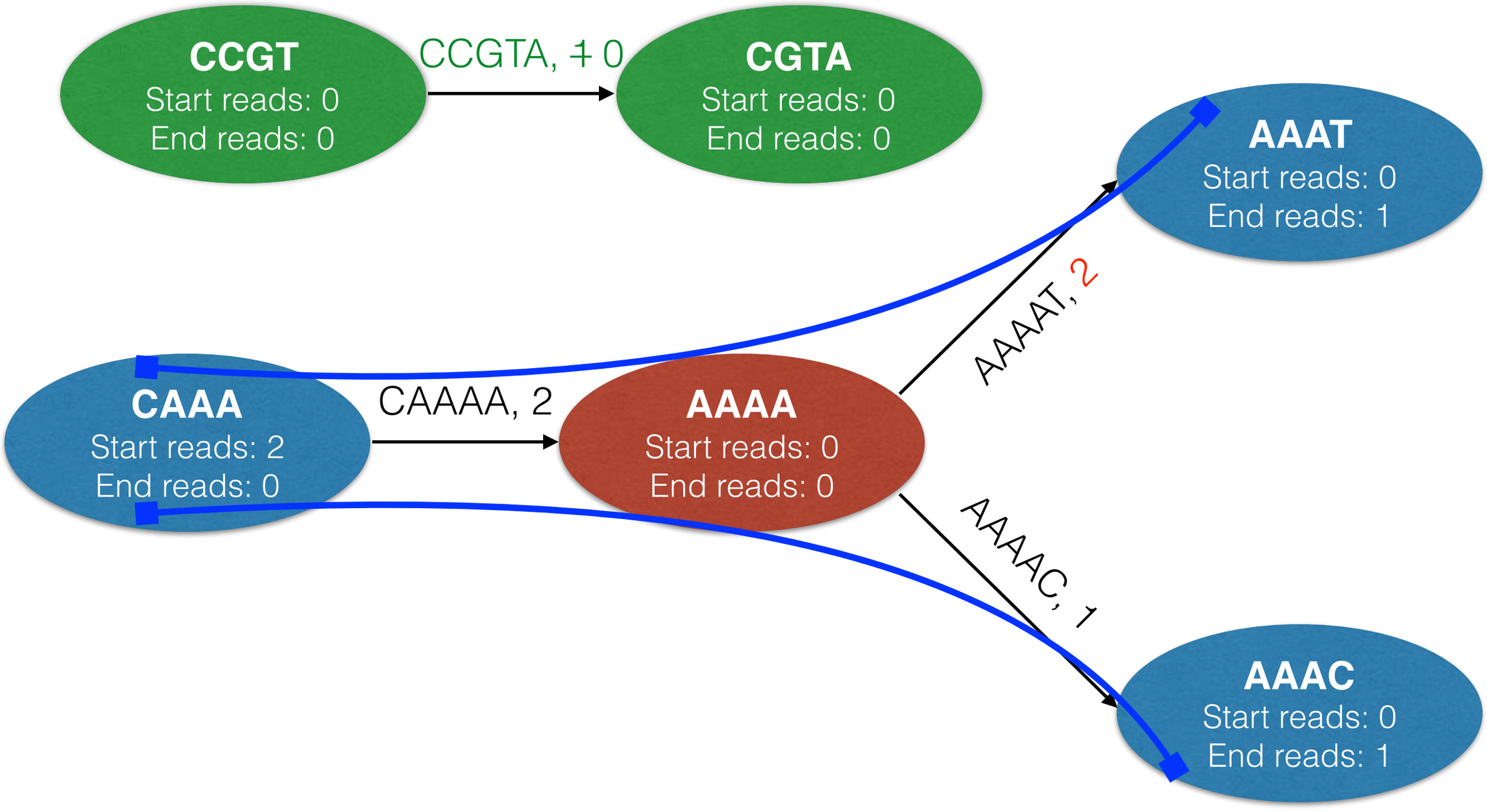
Node	Start reads
CAAA	2

Node	End reads
AAAT	1
AAAC	1

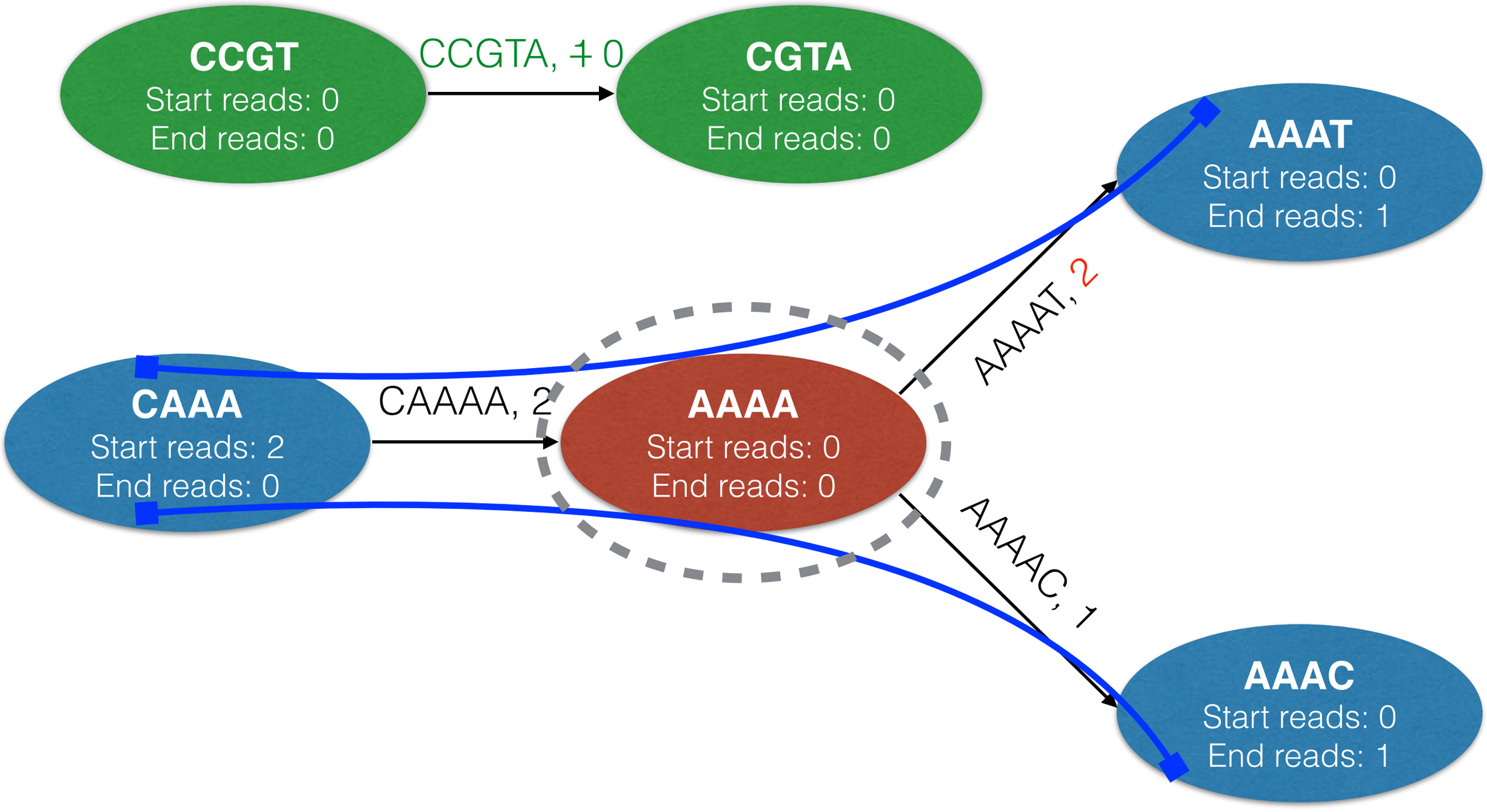
Error correction



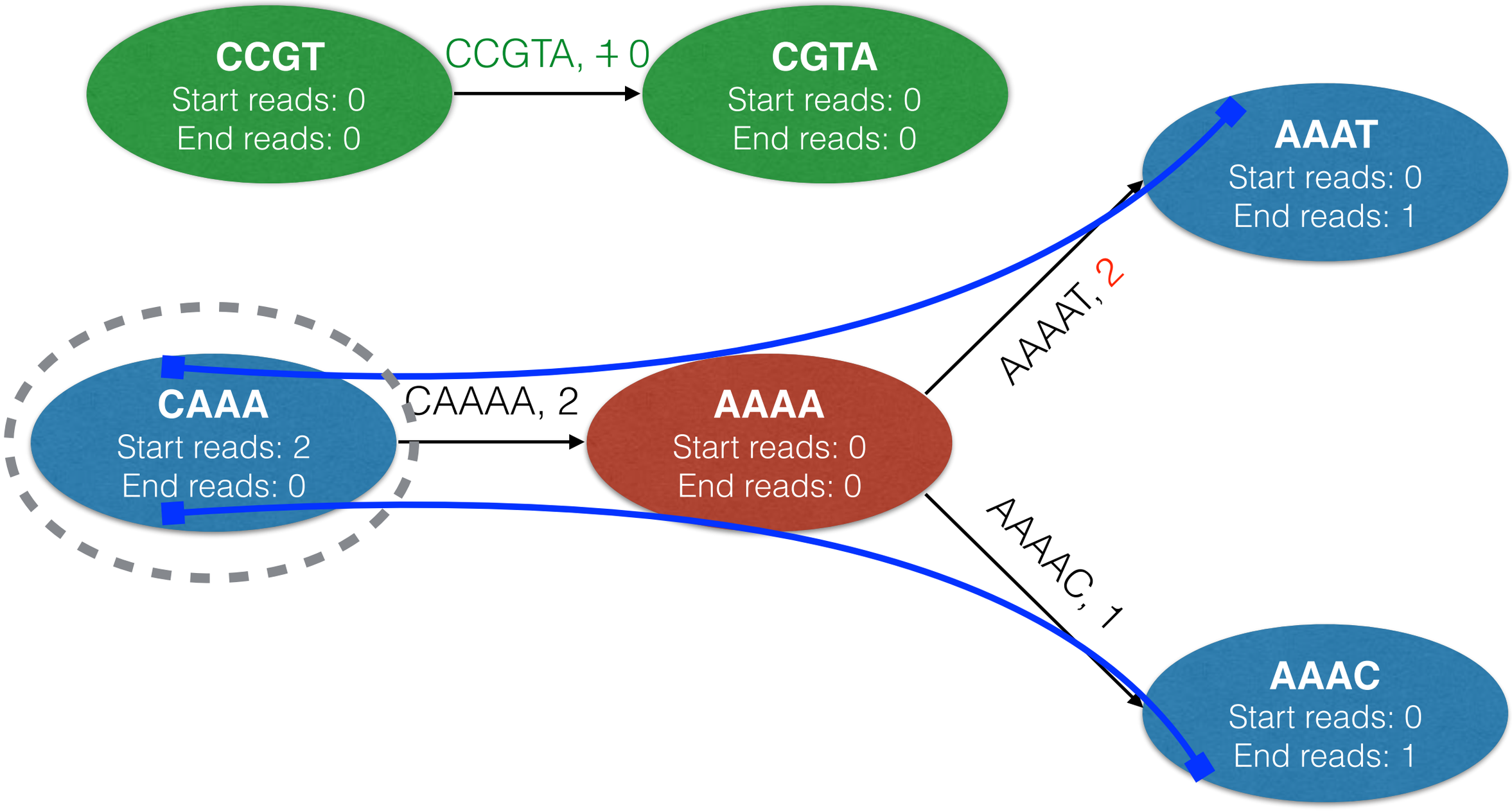
Error correction



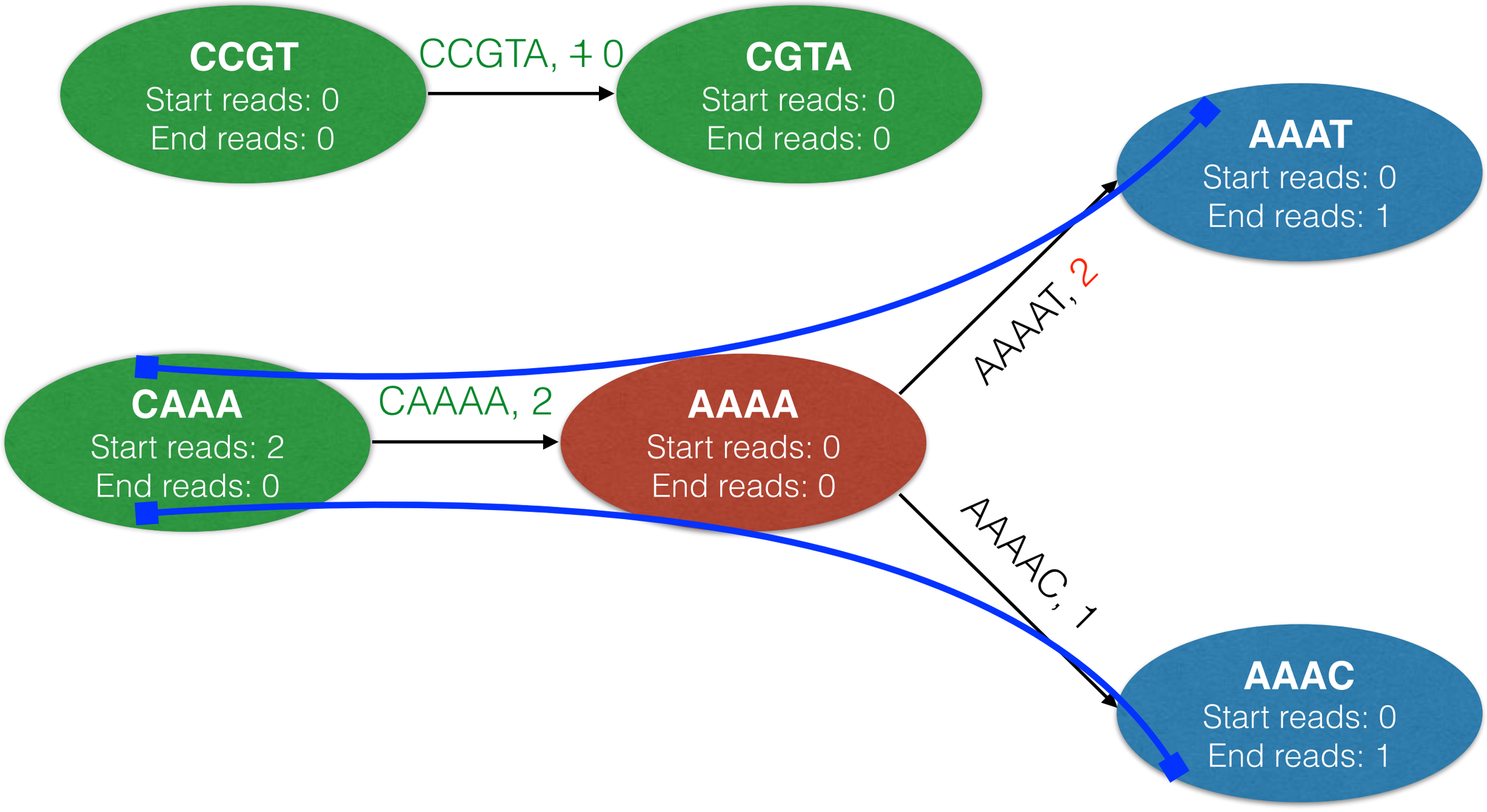
Error correction



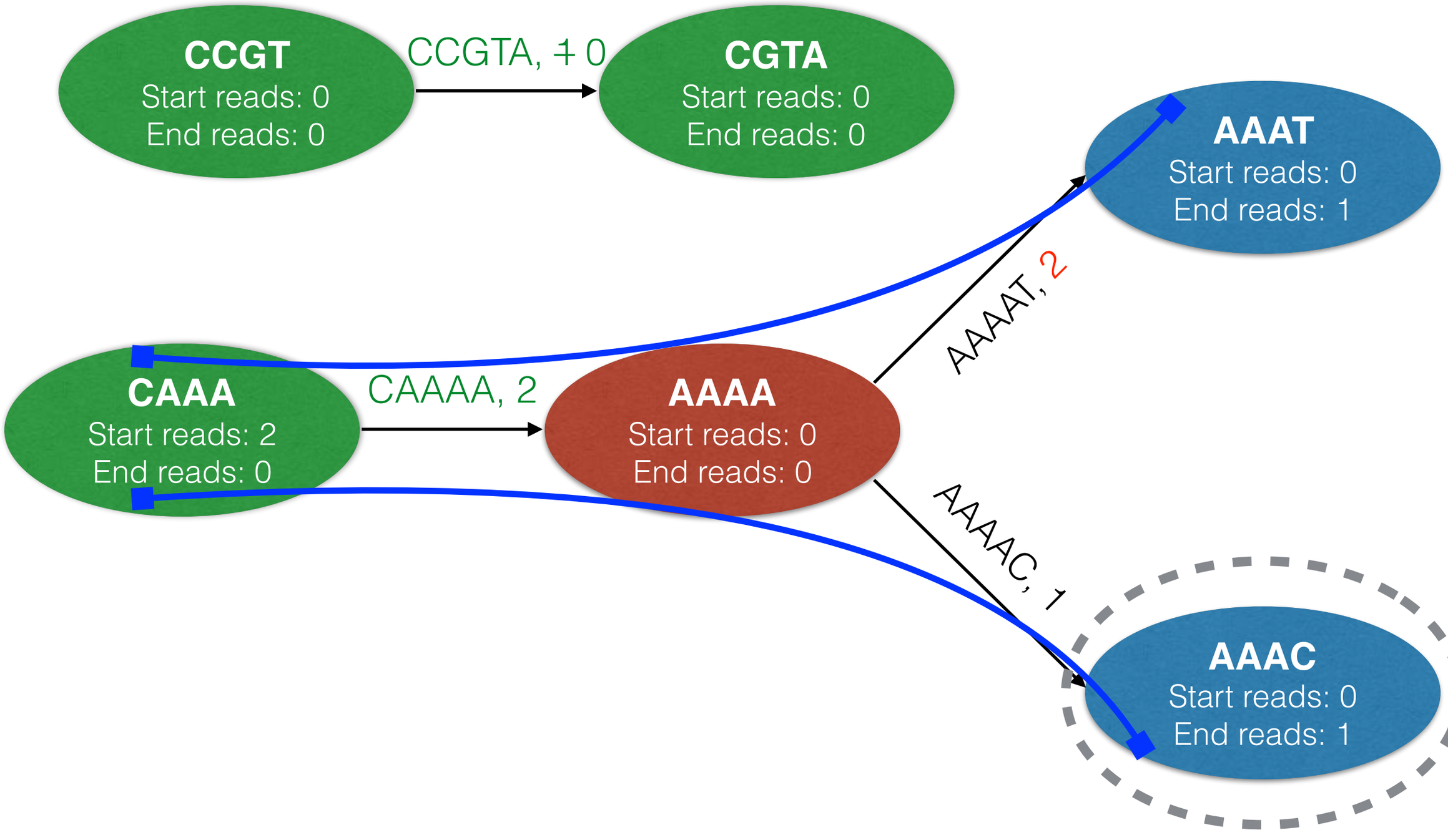
Error correction



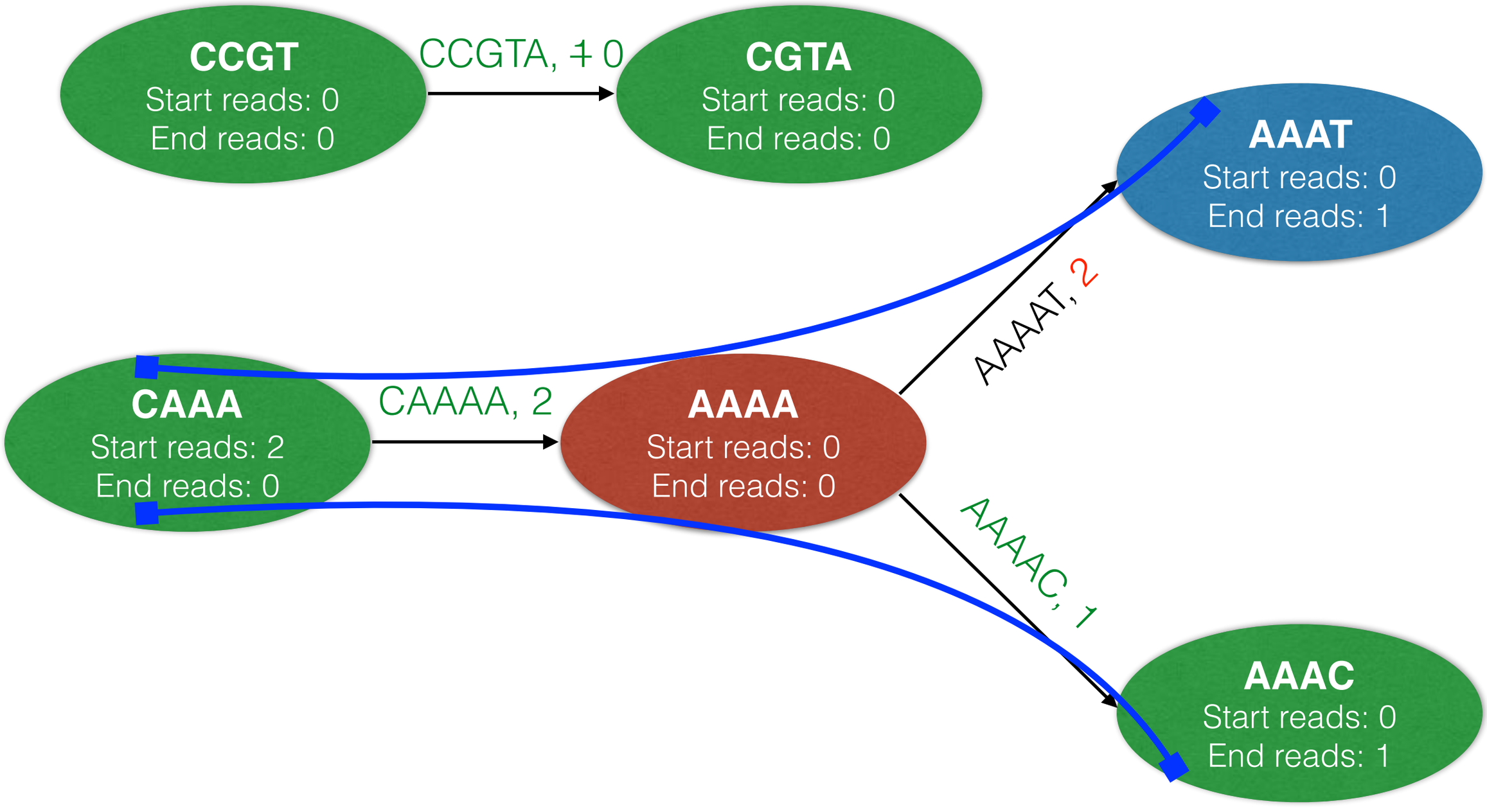
Error correction



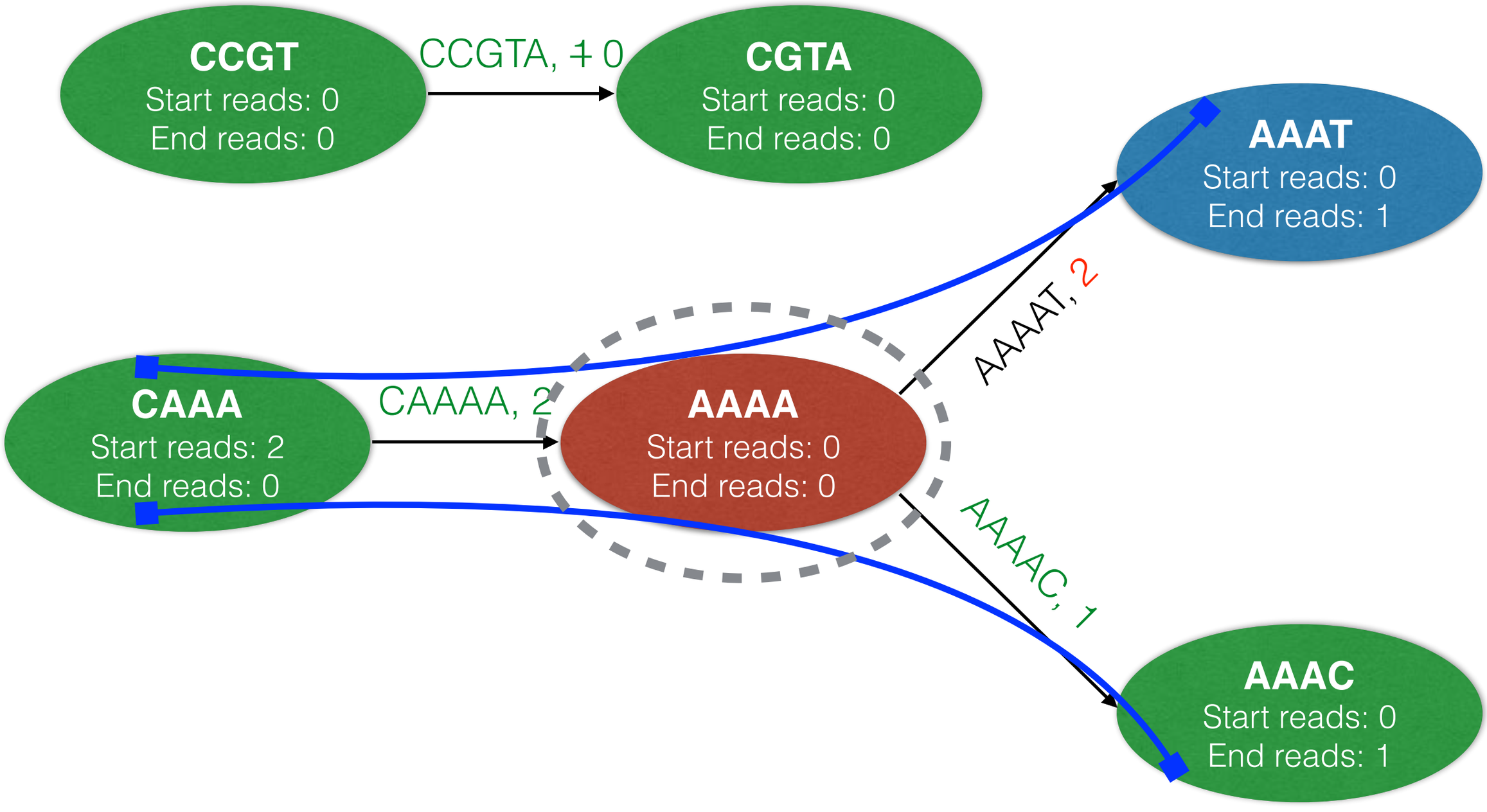
Error correction



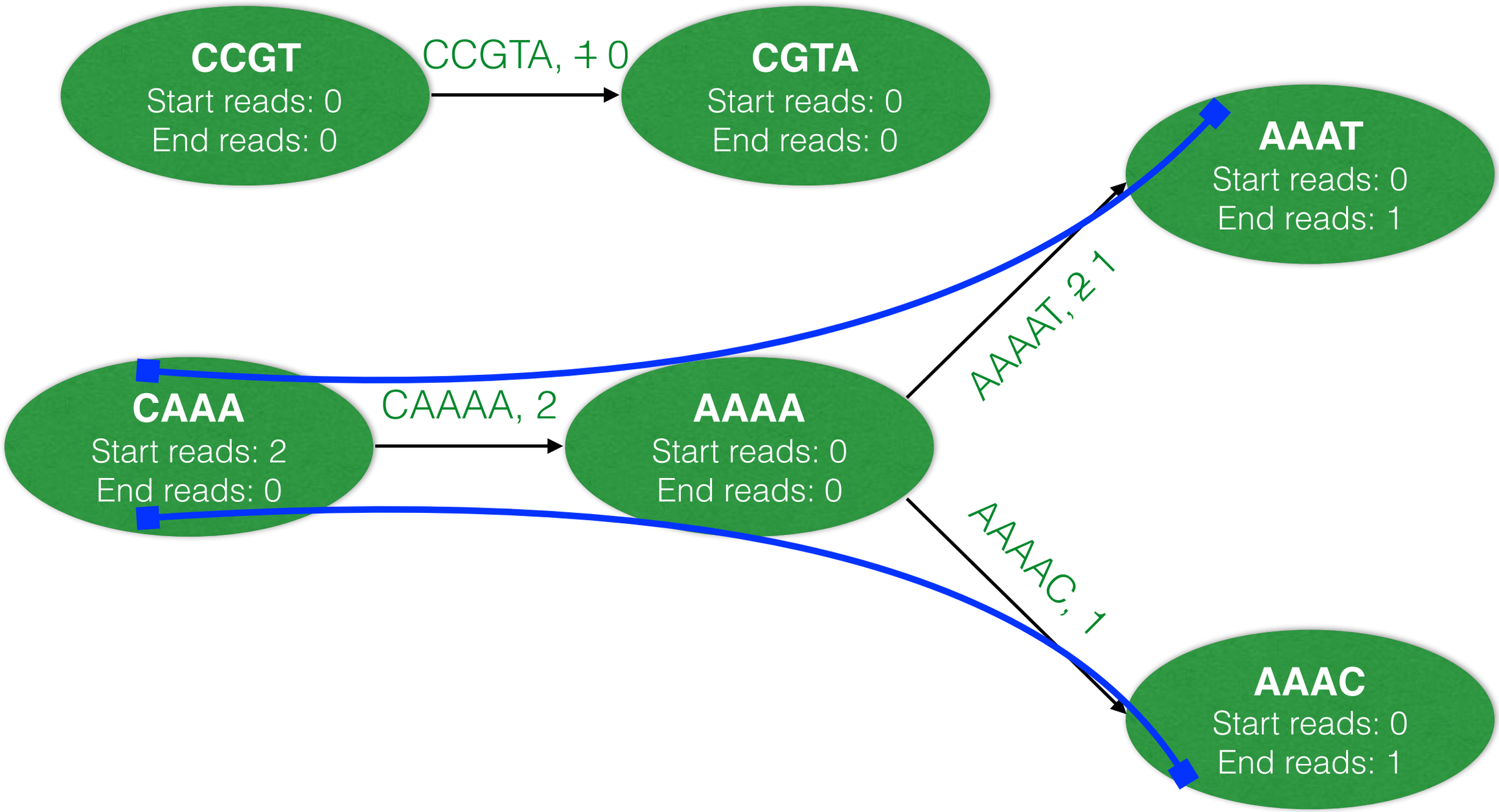
Error correction



Error correction



Error correction



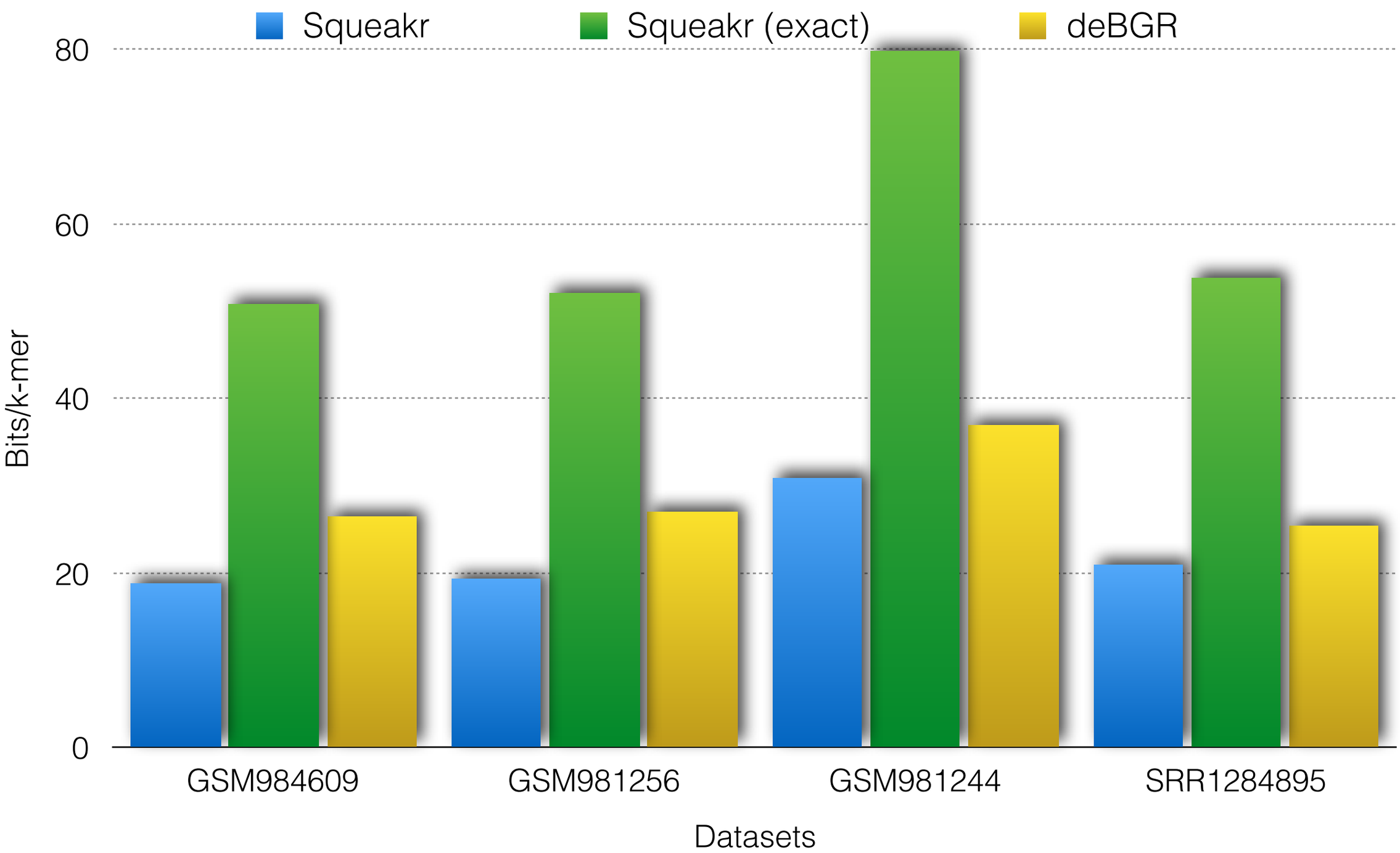
Error correction algorithm

- We use a standard work queue algorithm.
- We bootstrap with a set *C* of edges for which we know the abundance is correct.
- We then expand the set *C* of edges using the weighted de Bruijn graph invariant.
- Please refer to the paper for exact set of rules for error correction.
- Running time: $O(n \cdot \log(n) / \log(1/4\varepsilon))$.

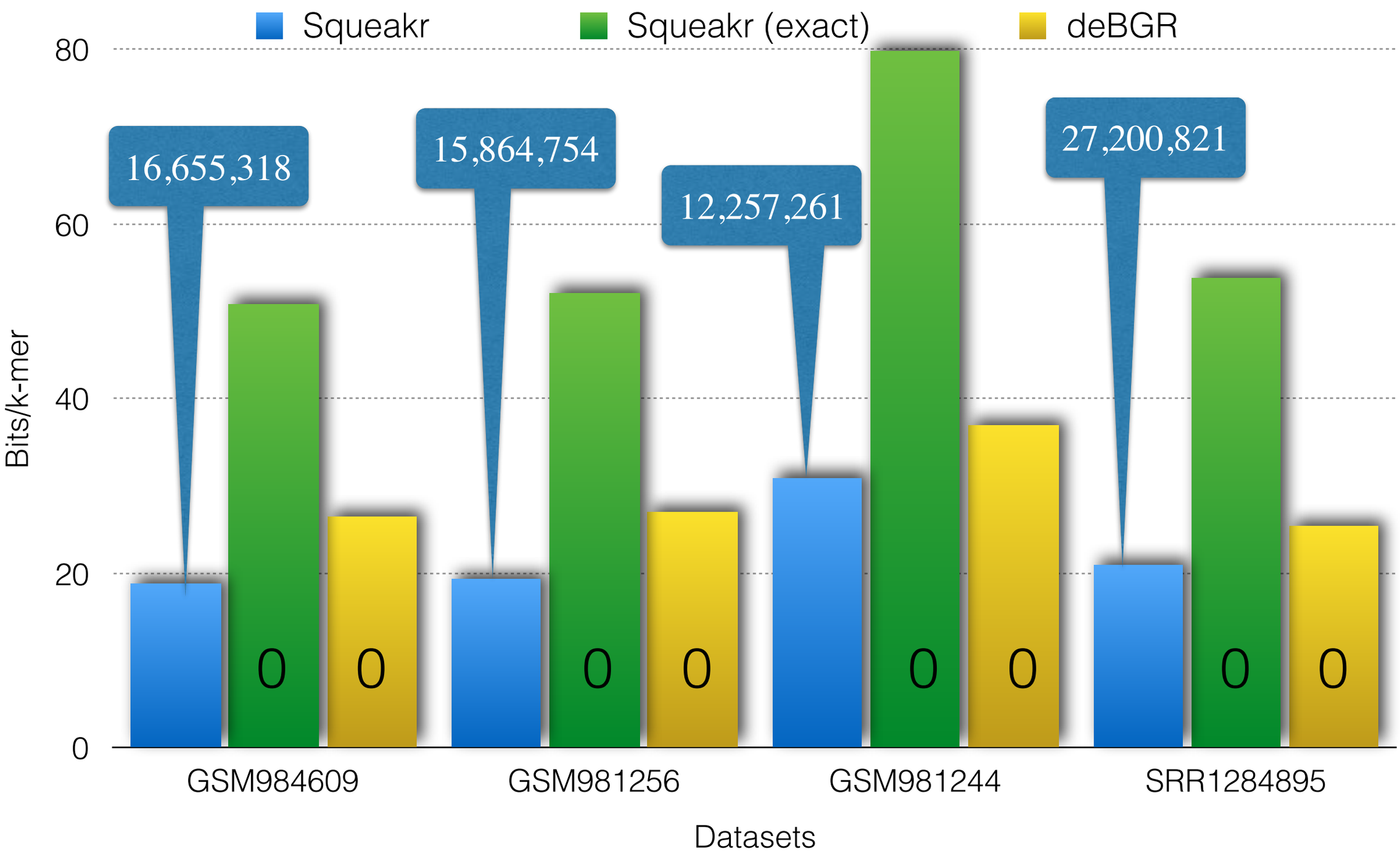
Datasets

Dataset	Size	#k-mer instances	#Distinct k-mers
GSM984609	26 GB	19,662,773,330	1,146,347,598
GSM981256	22 GB	16,470,774,825	1,118,090,824
GSM981244	43 GB	37,897,872,977	1,404,643,983
SRR1284895	33 GB	26,235,129,875	2,079,889,717

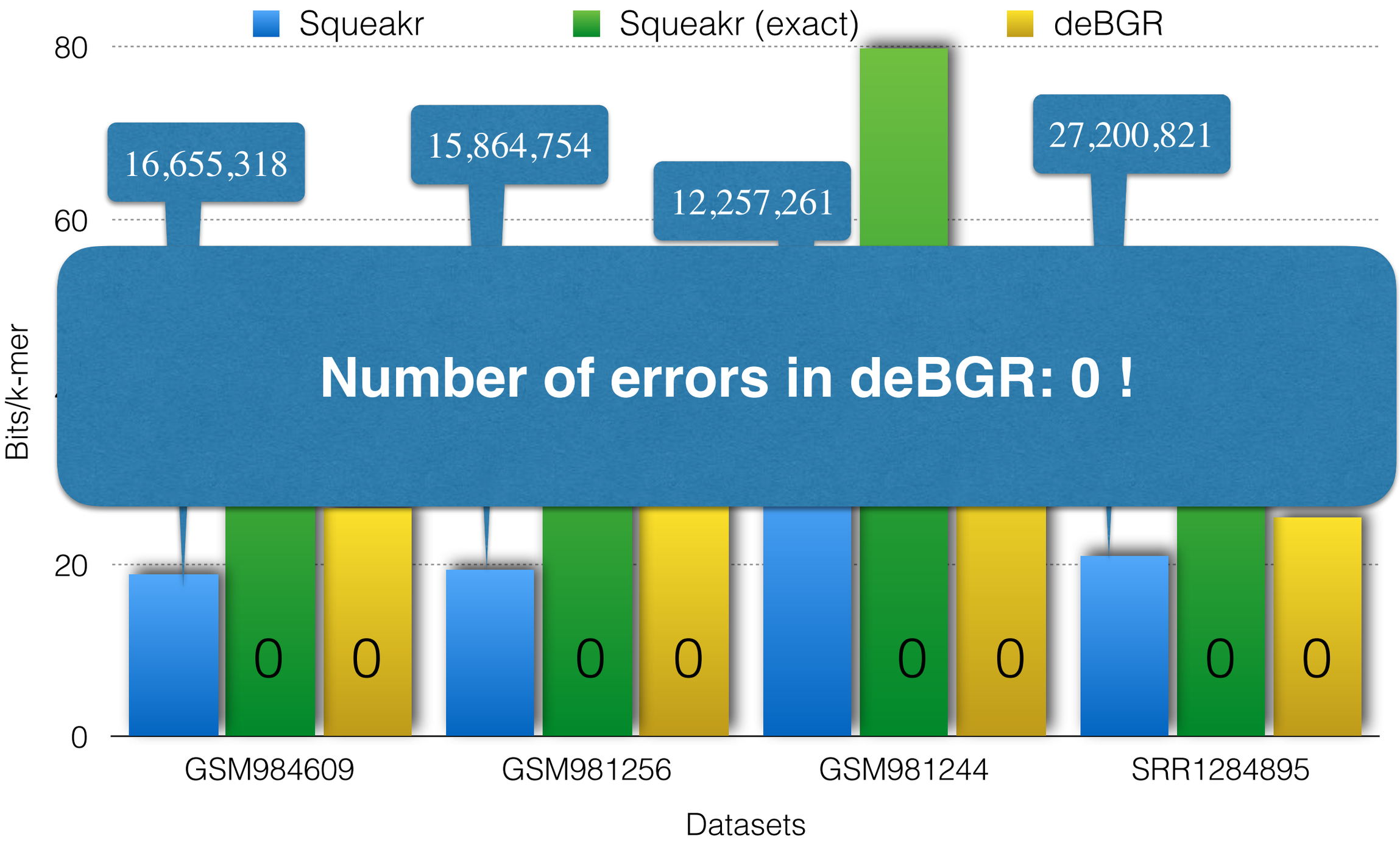
Space vs Accuracy



Space vs Accuracy



Space vs Accuracy



Conclusion

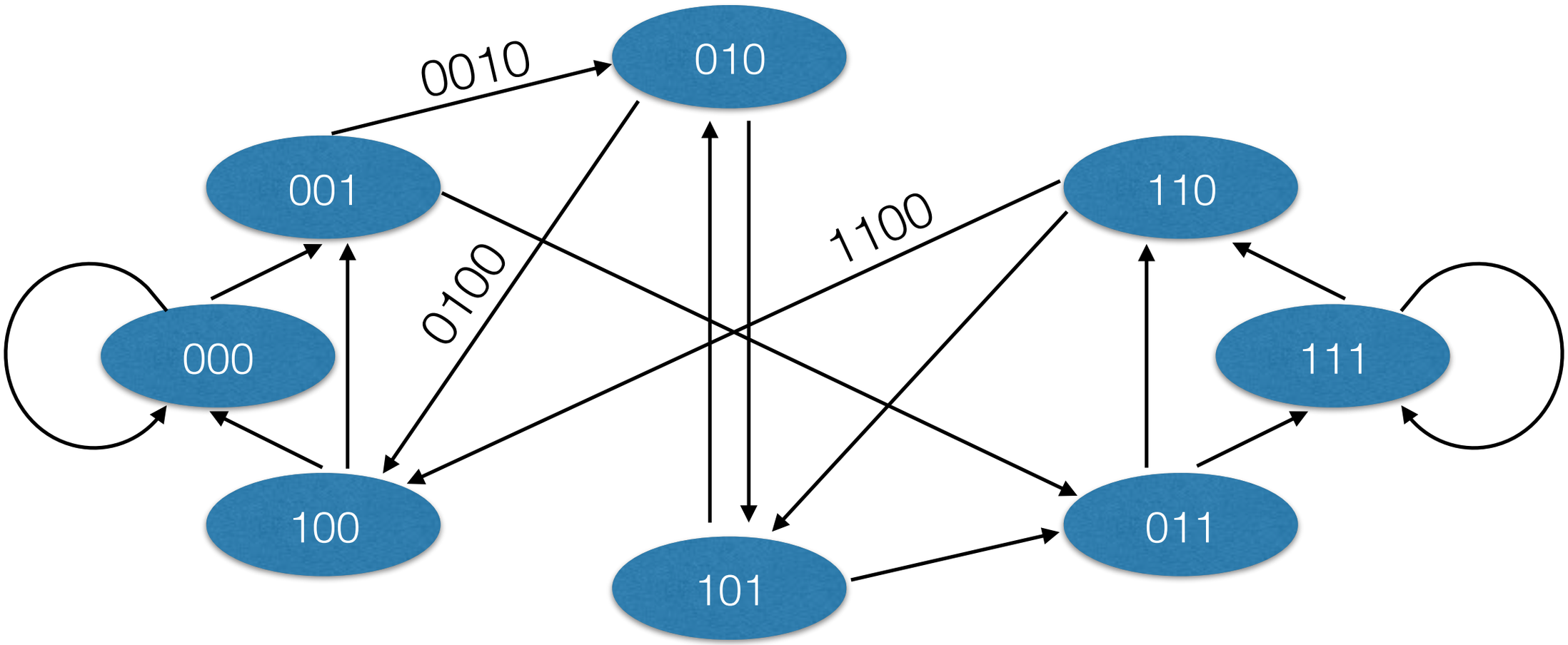
- Abundance information is important for many data analyses.

- But the abundance information can be used to remove effectively all the errors in an approximate weighted de Bruijn graph representation.

- The basic ideas behind our error-correction technique may also be useful for compactly representing weighted graphs by exploiting other domain-specific invariants.

https://github.com/splatlab/

De Bruijn graph (dBG)



In graph theory, an ***n*-dimensional** de Bruijn graph of ***m* symbols** is a directed graph representing overlaps between sequences of symbols.

The counting quotient filter (CQF)

[Pandey et al. SIGMOD 2017]

- A replacement for the (counting) Bloom filter.
- Space and computationally efficient.
- Uses variable-sized counters to handle skewed data sets efficiently.

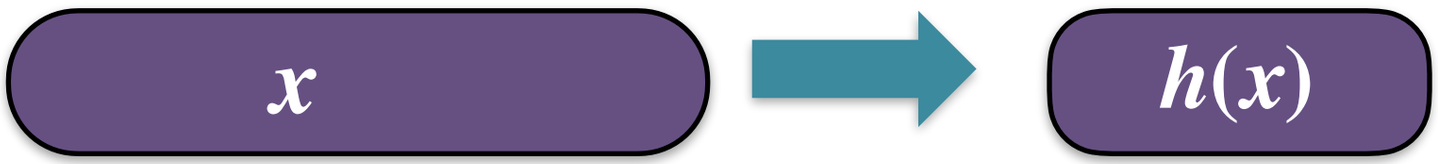
$$\text{CQF space} \leq \text{BF space} + \underbrace{O\left(\sum_{x \in S} \log c(x)\right)}_{\text{Asymptotically optimal}}$$

Counting quotient filter (CQF)

- Smaller than many non-counting AMQs
  - Bloom, cuckoo [Fan et al., 2014], and quotient [Bender et al., 2012] filters.
- Good cache locality
- Deletions
- Dynamically resizable
- Mergeable

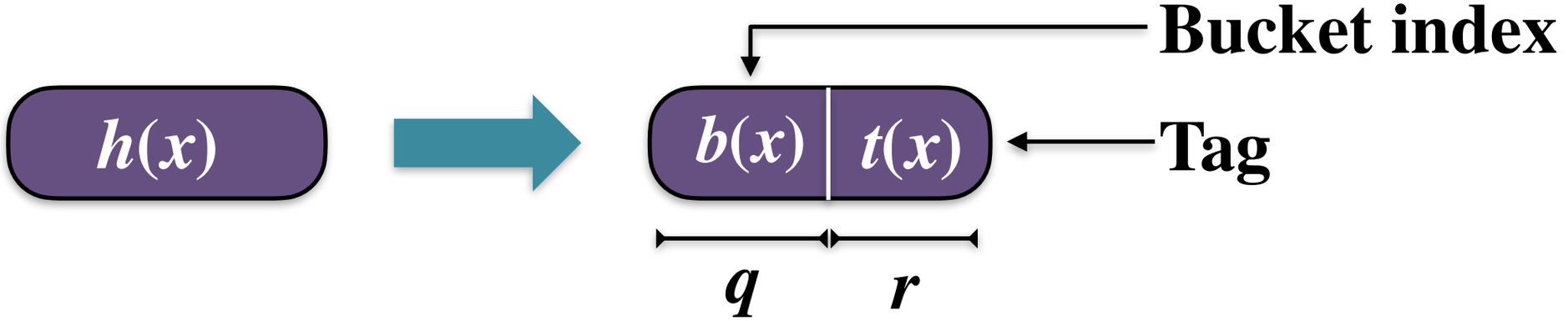
Quotienting: An alternative to Bloom filters

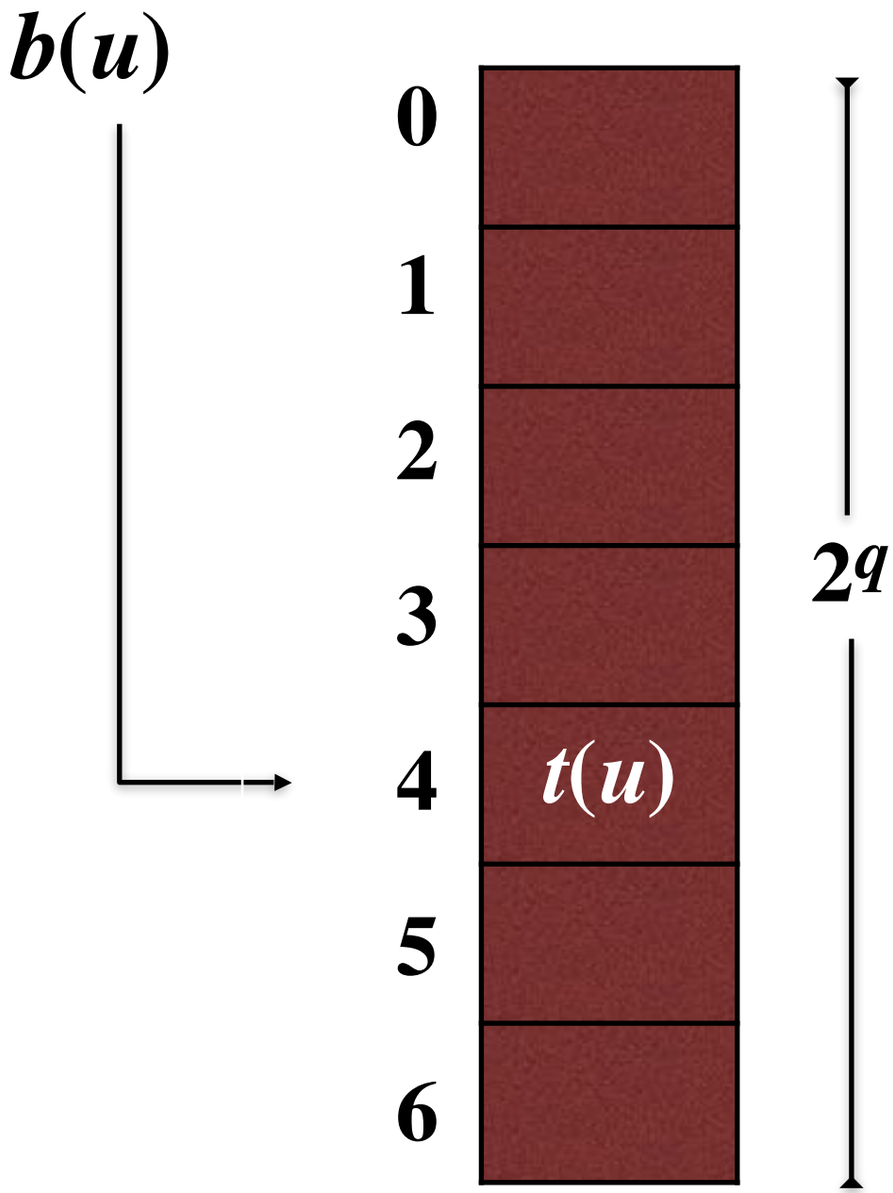
- **Store fingerprint compactly in a hash table.**
  - Take a fingerprint ***h(x)*** for each element ***x***.



- **Only source of false positives:**
  - Two distinct elements ***x*** and ***y***, where ***h(x)*** = ***h(y)***.
  - If ***x*** is stored and ***y*** isn't, ***query(y)*** gives a false positive.

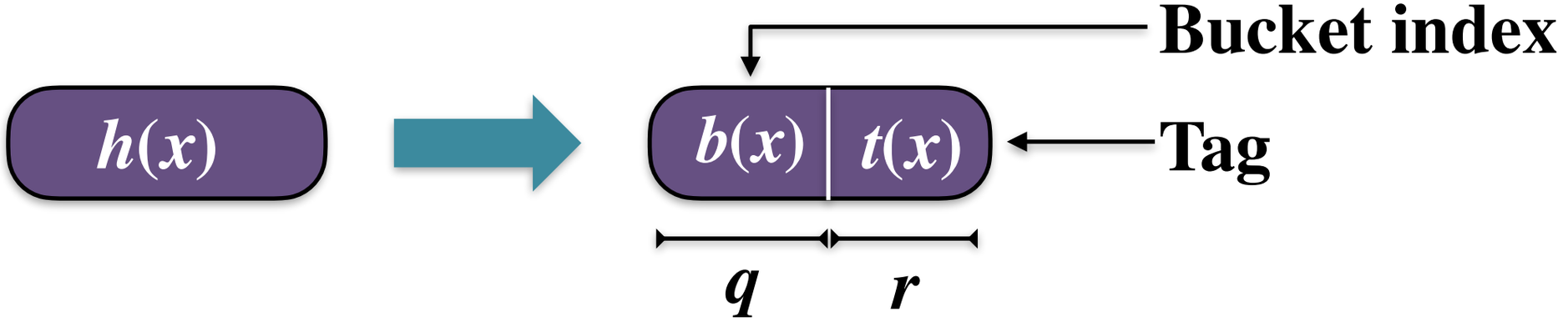
Storing compact fingerprints

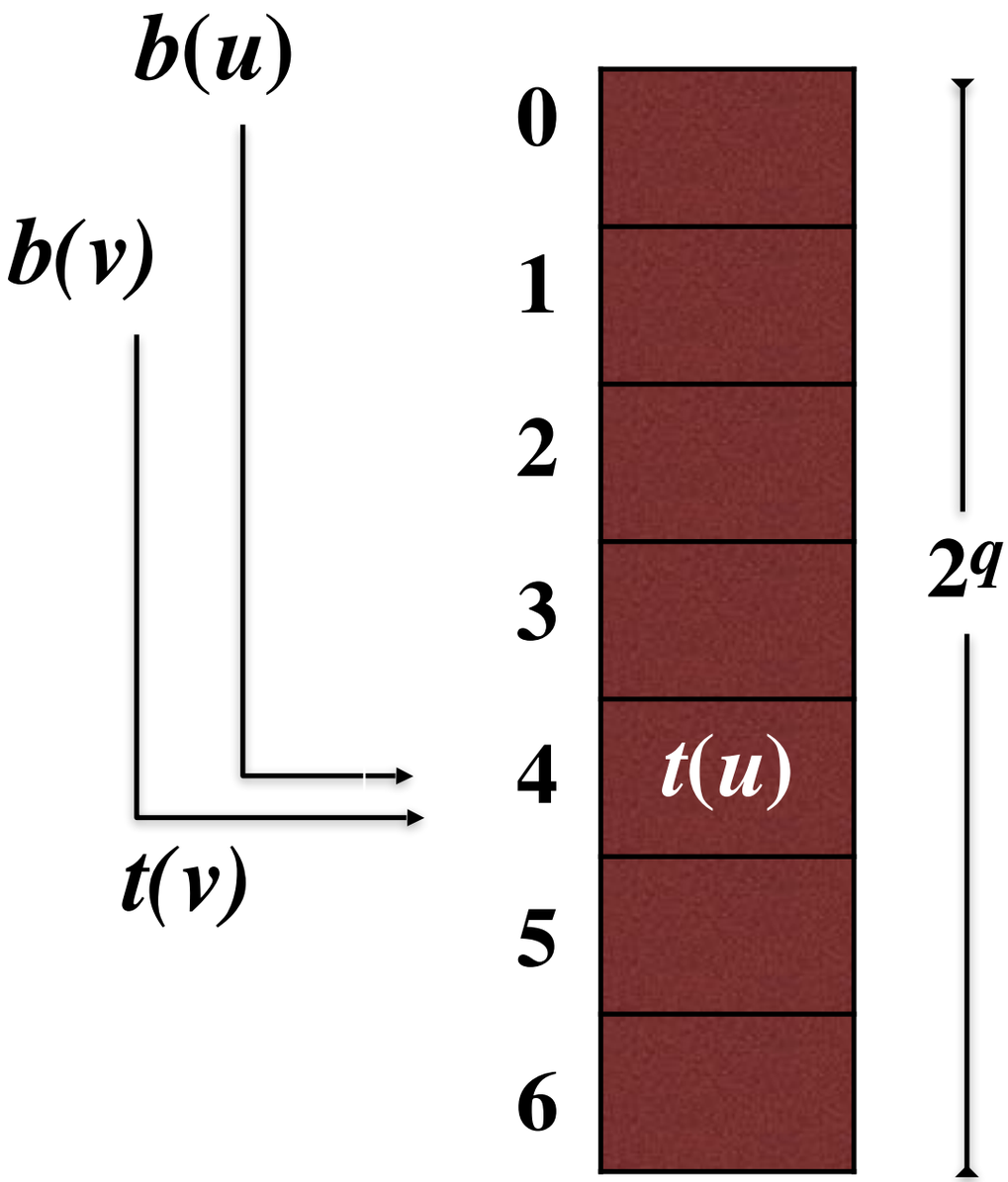




- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Storing compact fingerprints

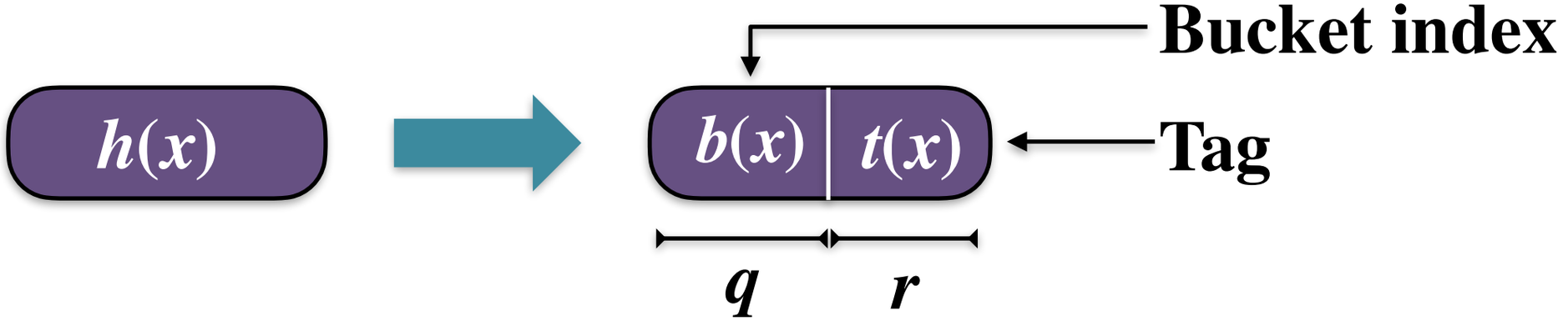


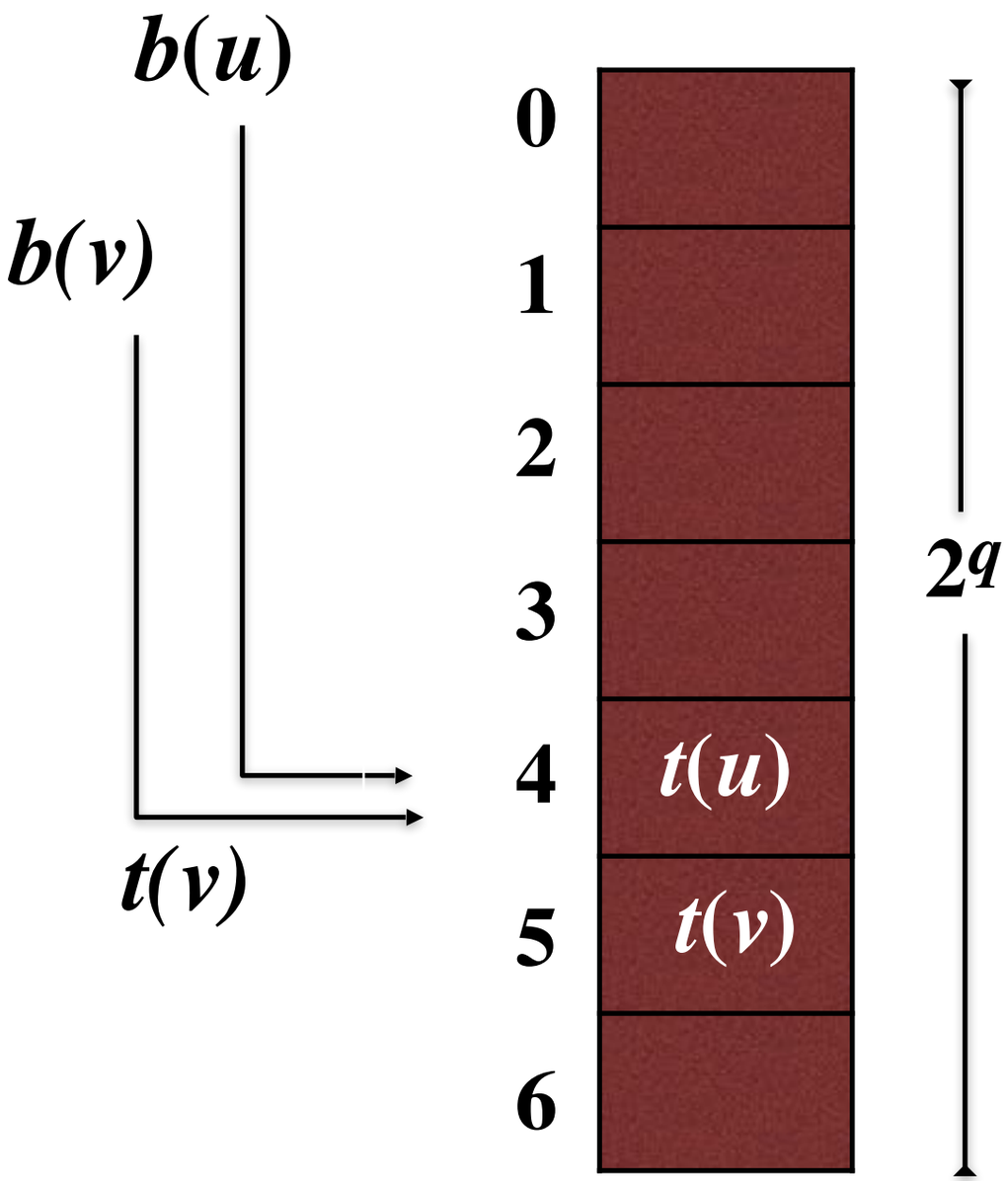


- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?

Storing compact fingerprints

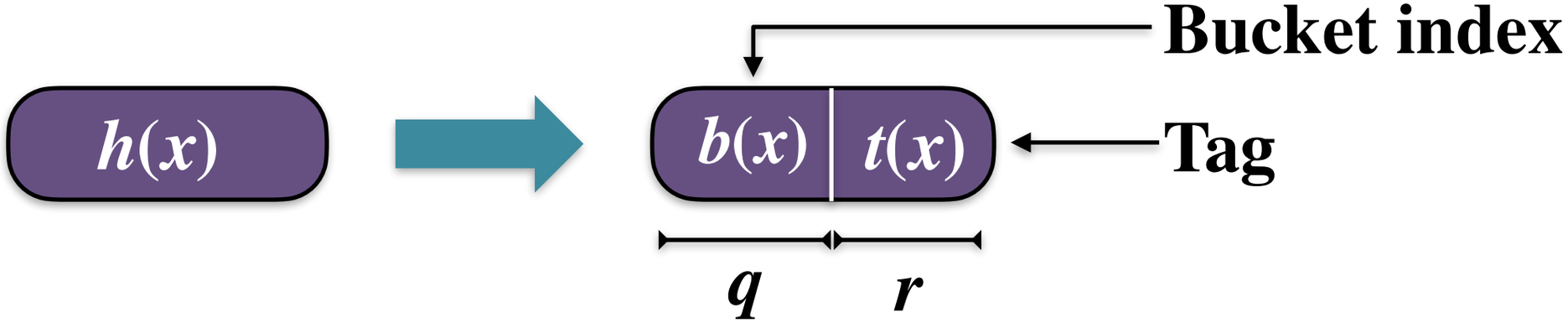


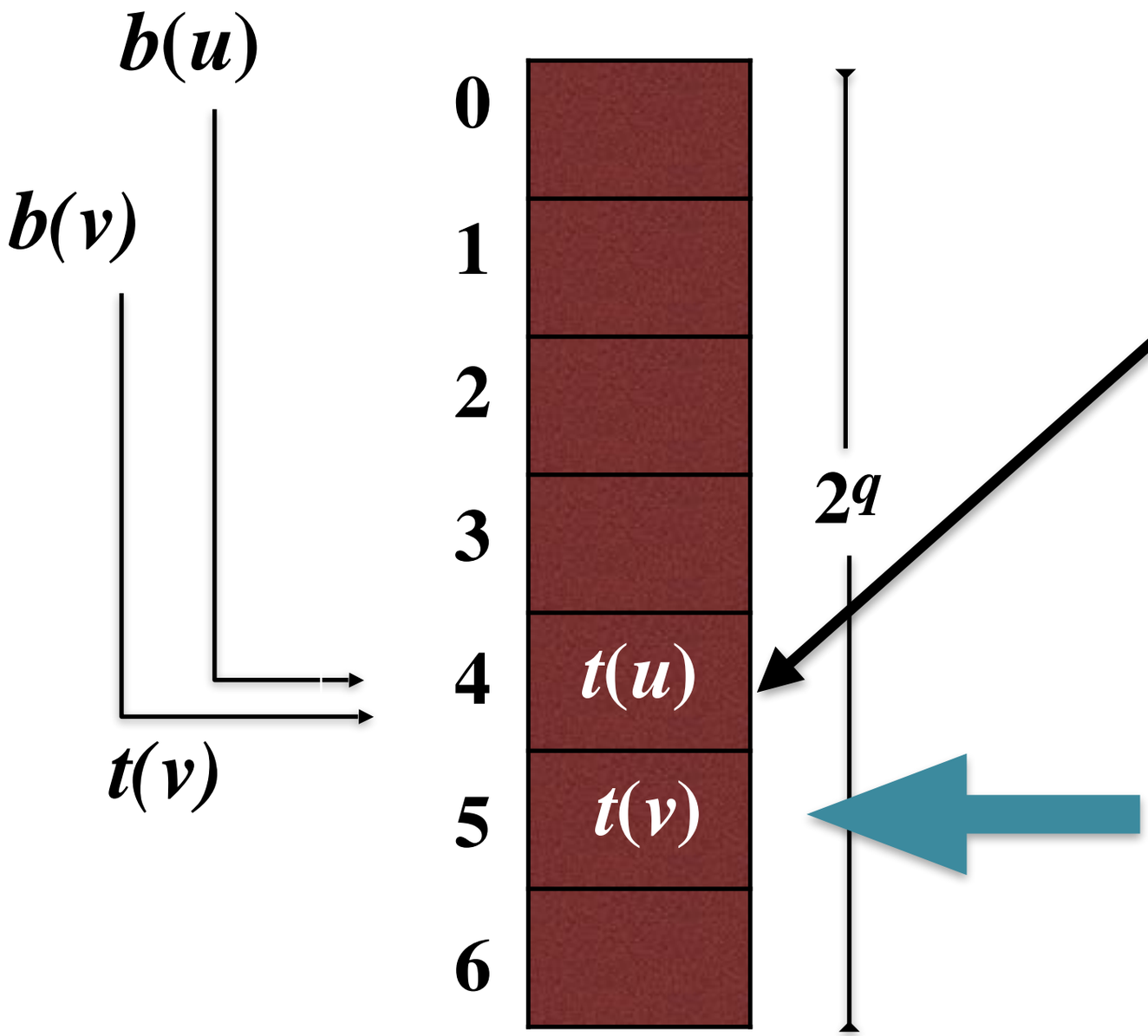


- $b(x)$ = location in the hash table
- $t(x)$ = tag stored in the hash table

Collisions in the hash table?
Linear probing.

Storing compact fingerprints





- The home bucket for $t(u)$ and $t(v)$ is 4.

Does $t(v)$ belongs to bucket 4 or 5 ?

Resolving collisions in the CQF

- CQF uses two metadata bits to resolve collisions and identify the home bucket.

	1		**1**				
	***t*(*u*)**	***t*(*v*)**	***t*(*w*)**	***t*(*x*)**	***t*(*y*)**		

- The metadata bits group tags by their home bucket.

Encoding counts

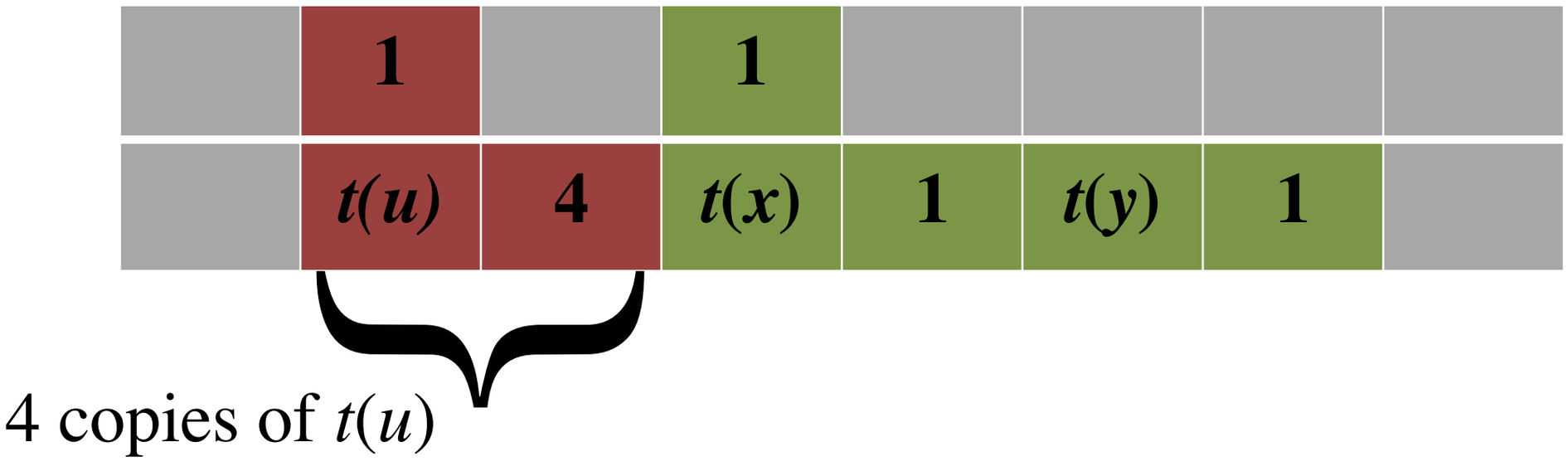
- Metadata scheme tells us the run of slots holding contents of a bucket.
- We can encode contents of buckets however we want.
- ***The original quotient filter used repetition (unary).***

	1		**1**				
	***t*(*u*)**	***t*(*u*)**	***t*(*u*)**	***t*(*u*)**	***t*(*x*)**	***t*(*y*)**	

Encoding counts

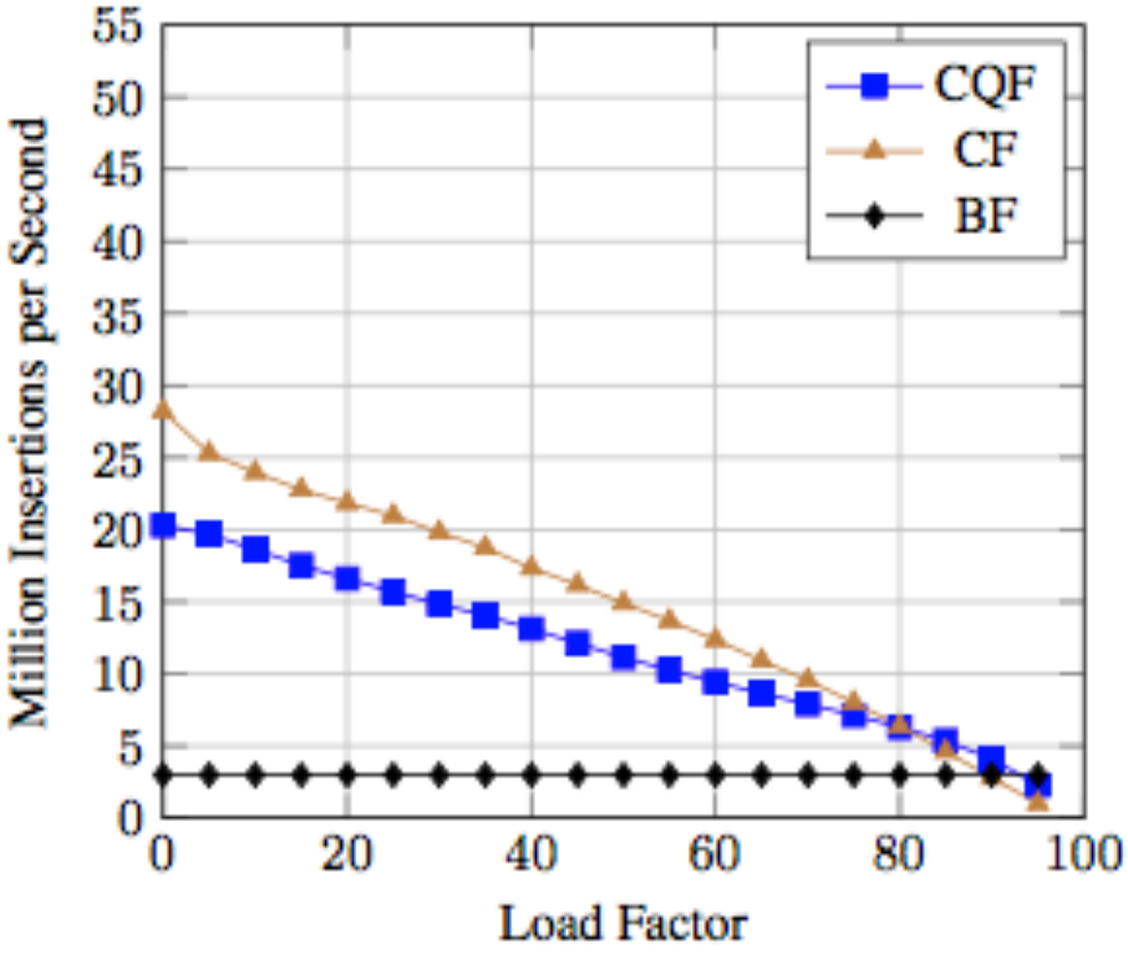
- ***We want to count in binary, not unary.***
- Idea: use some of the space for tags to store counts.
- Issue: determine which are tags and which are counts

without using even one “control” bit.

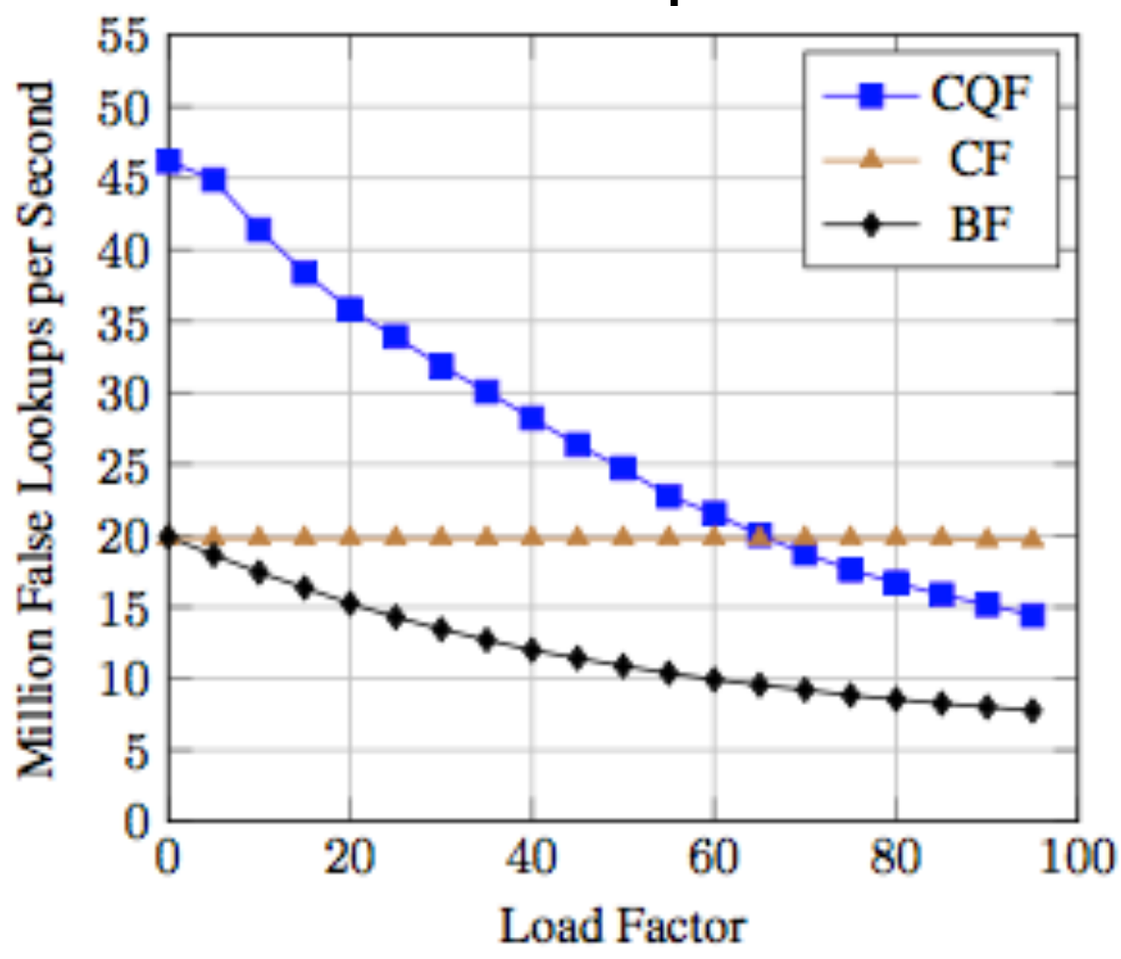


Performance: In memory

Inserts



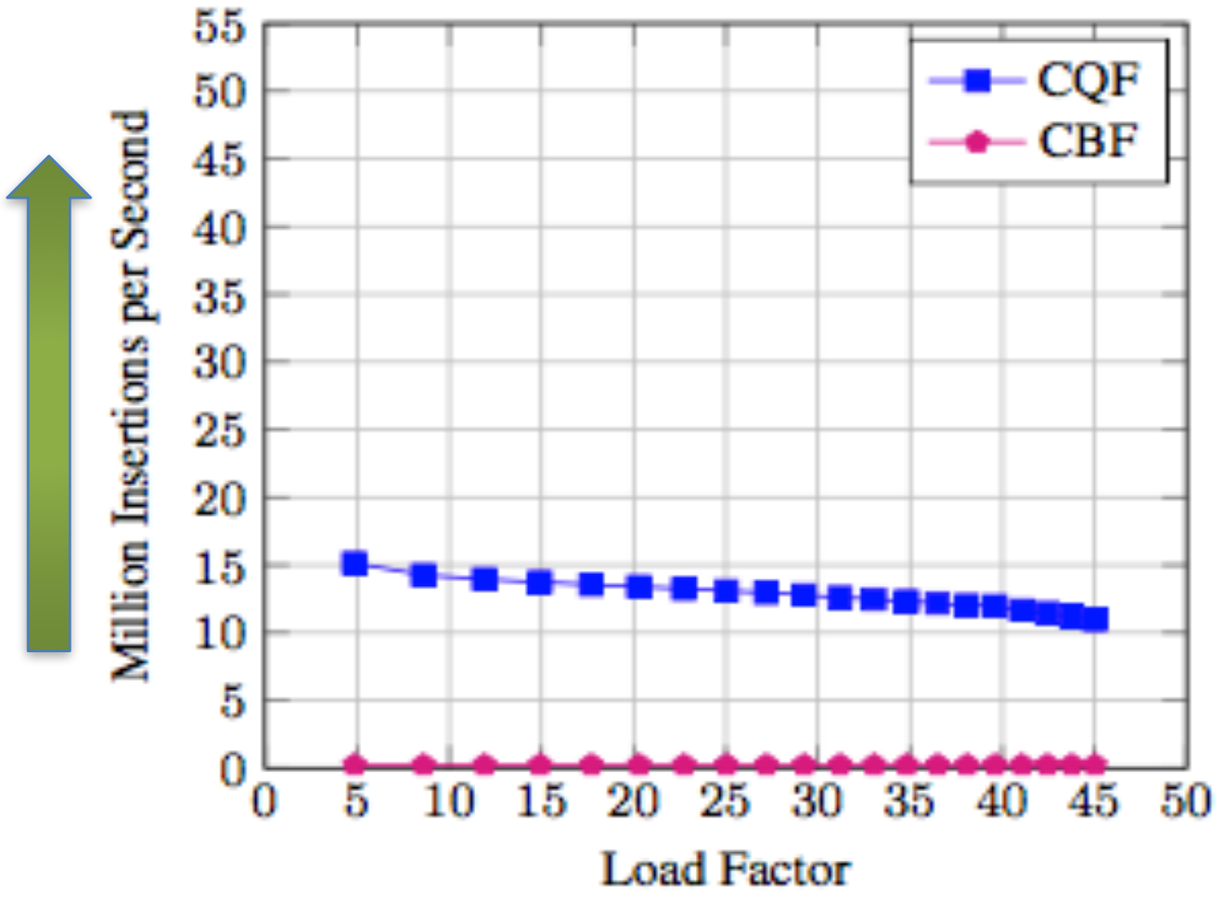
lookups



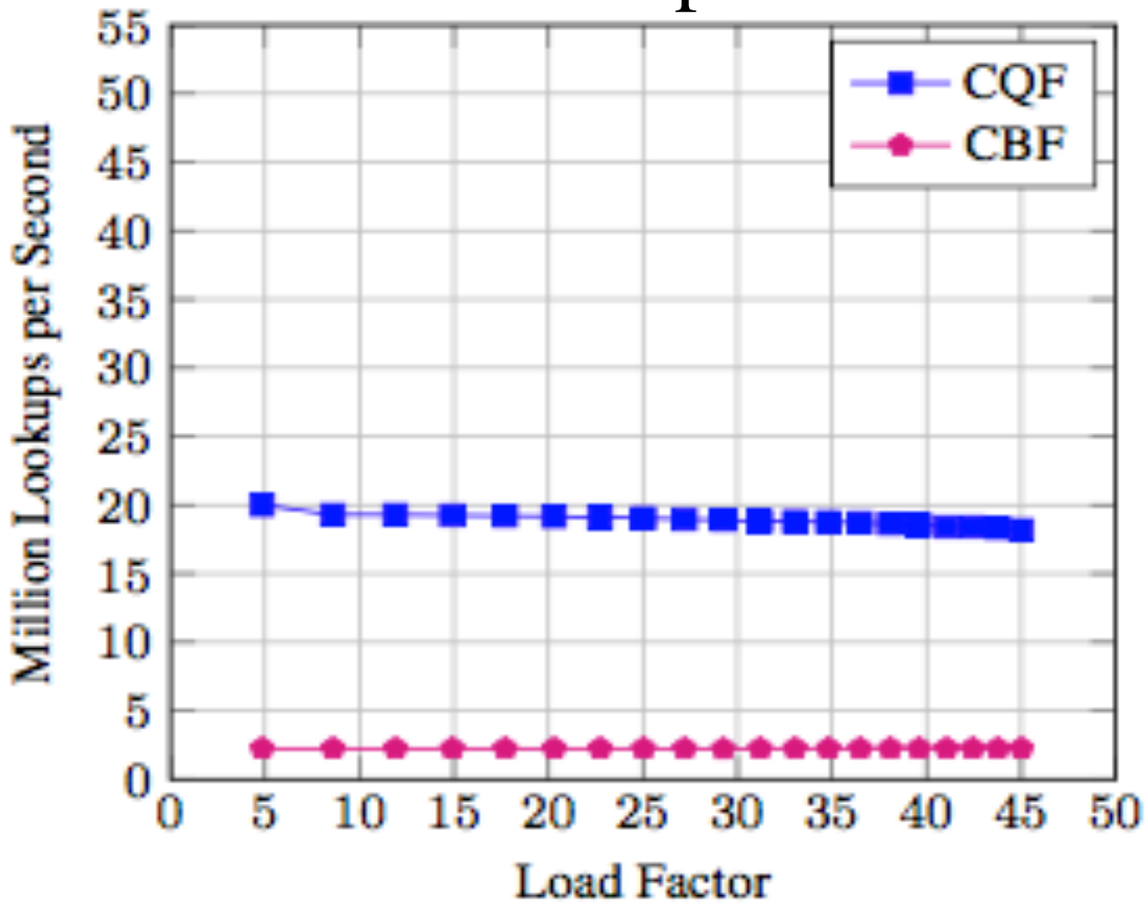
- The CQF insert performance in RAM is similar to that of state-of-the-art **non-counting** AMQs.
- The CQF is significantly faster at low load factors and slightly slower on high load factors.

Performance: Skewed datasets

Inserts



lookups



- The CQF outperforms the CBF by a factor of 6x-10x on both inserts and lookups.