Hashing and Sketching

Comparison can be slow

We know calculating local alignments is $O(n^2)$

- in the case of read overlapping if there are say 10^6 reads
- if reads are 10^2 bases each, thats 10^{10} computations!

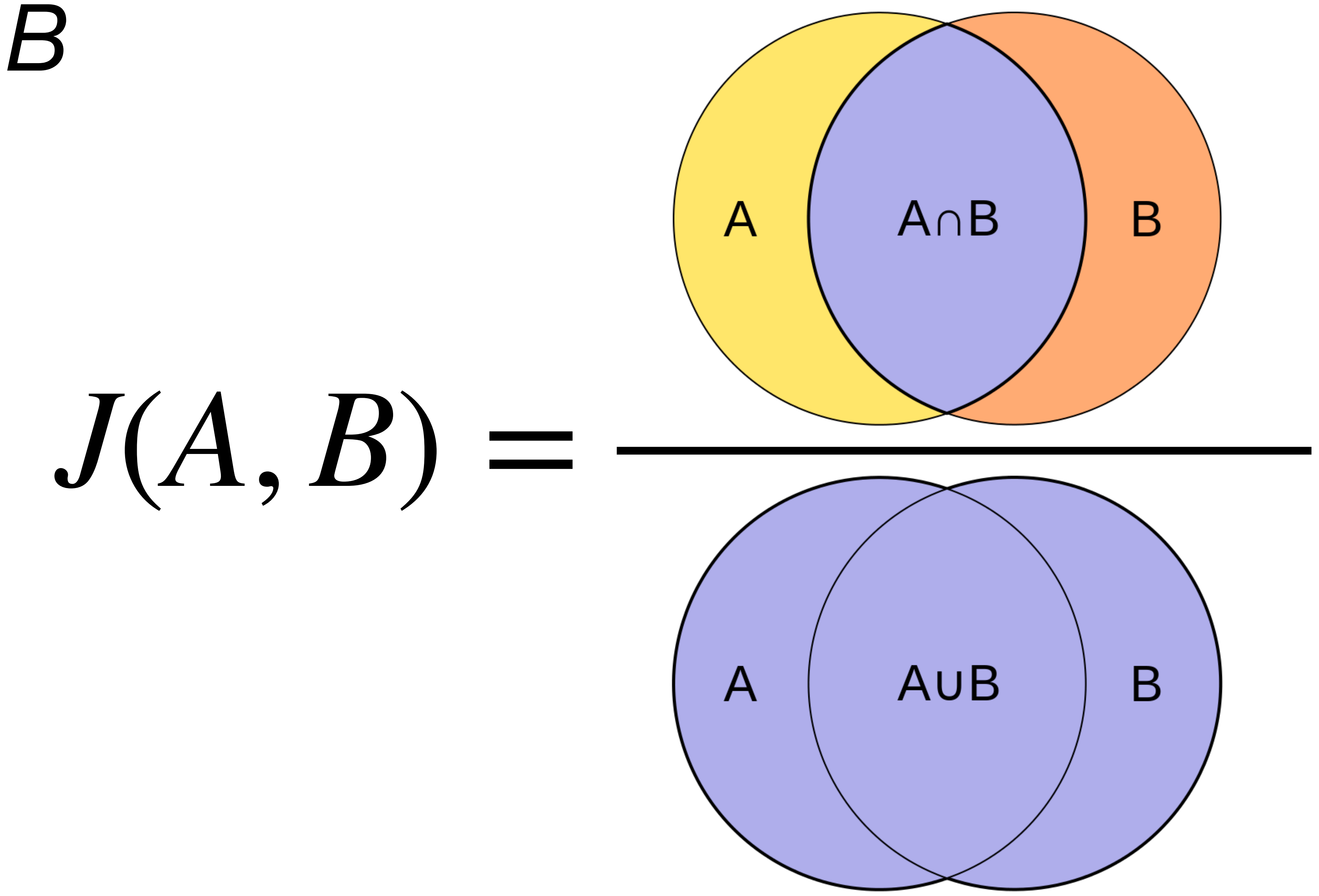
Even hamming distance ($O(n)$) may be too slow.

Remember, finding overlaps is just step 1 of assembly!

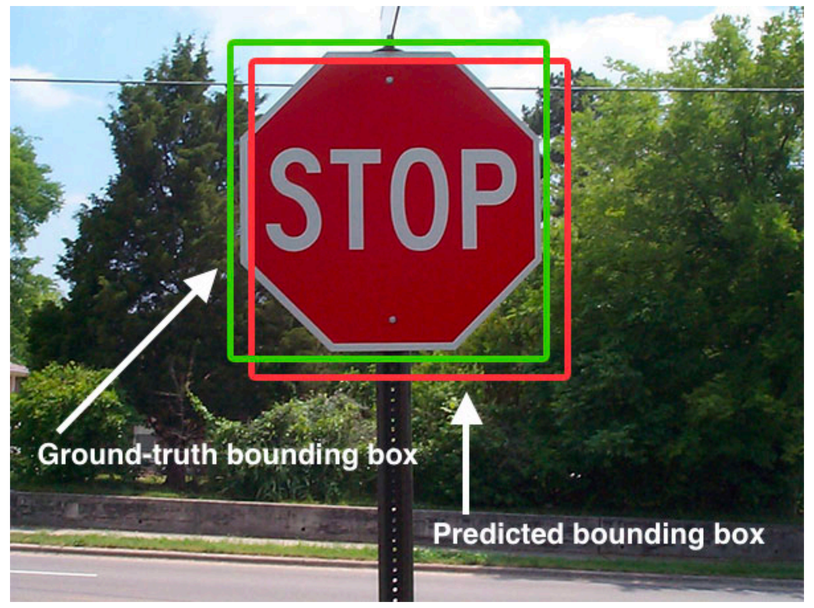
Jaccard Similarity

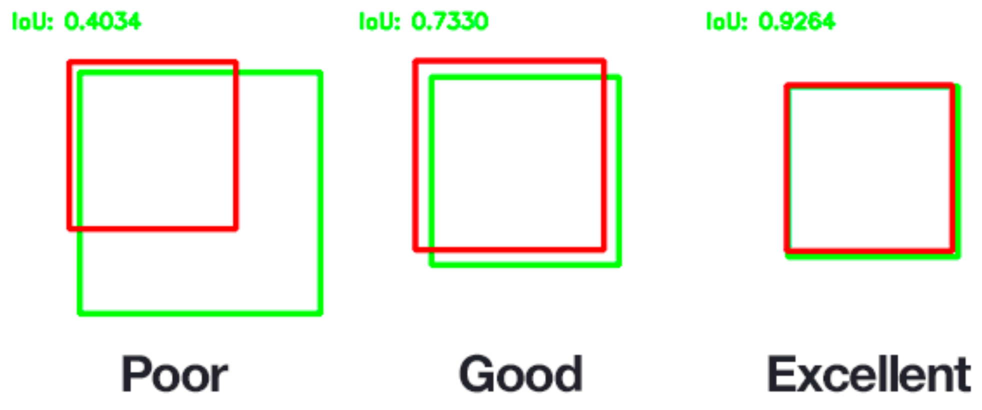
Measures the similarity of two sets of items *A* and *B* as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$



Used also used in computer vision, sometimes called the "Intersection over Union" (IoU) metric



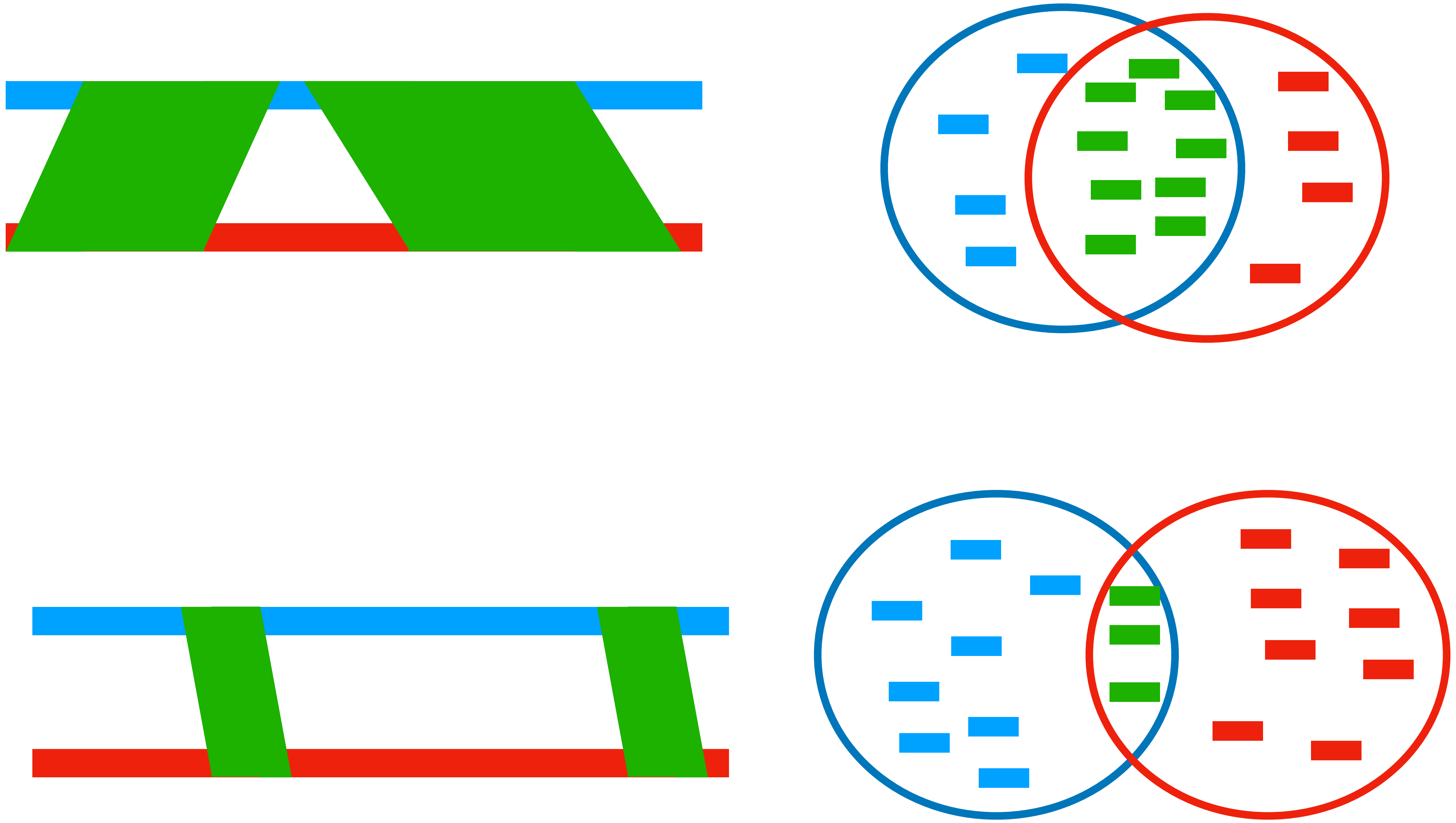


How would we use Jaccard for sequences?

Images courtesy of Wikimedia Commons.

Jaccard Similarity

In sequence analysis we construct a sets of *k*-mers for each of the strings being compared



Min-Hash Sketch

Calculating the union and intersection of a set of anything (in particular k-mers) can be time consuming (*O(n)* time)

Can we calculate it faster?

Consider the following scenario:

- given a hash function on *k*-mers $h: \Sigma^k \rightarrow \mathbf{Z}^+$
- and the sets of *k*-mers for two string *A* and *B,*
- What is the probability that $min_{c \in A}\{h(c)\} = min_{c \in B}\{h(c)\}$?

Turns out that

$$Pr_h\left[min_{c \in A}\{h(c)\} = min_{c \in B}\{h(c)\}\right] = J(A, B)$$

Min-Hash Sketch

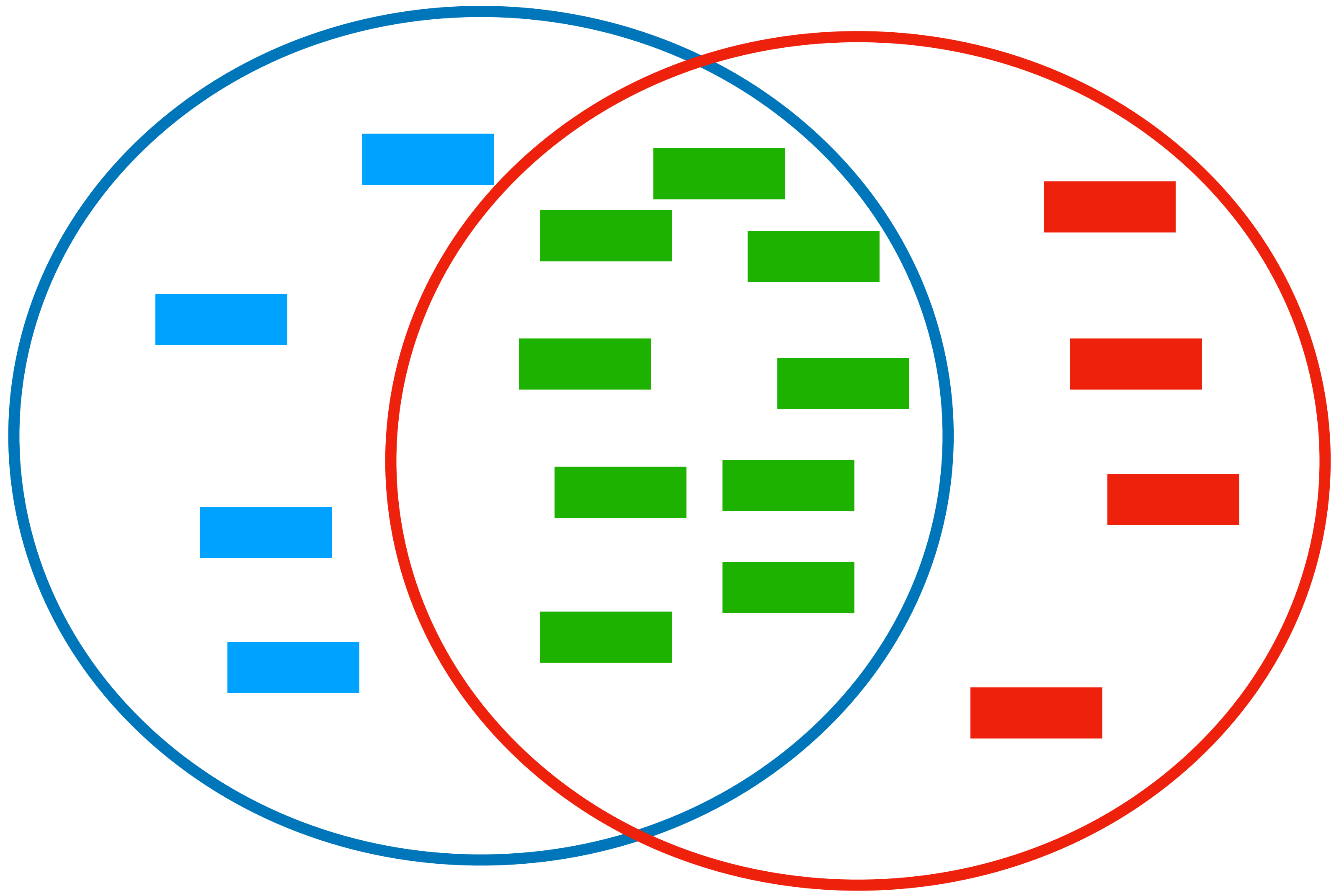
Why is $Pr_h\left[min_{c\in A}\{h(c)\} = min_{c\in B}\{h(c)\}\right] = J(A,B)$?

Think of *h* as applying a randomized ordering on the *k*-mers.

If the minimum *k*-mer from the union is in the intersection, it will be minimum for both *A* and *B.*

How many minimum *k*-mers from the union can we choose?

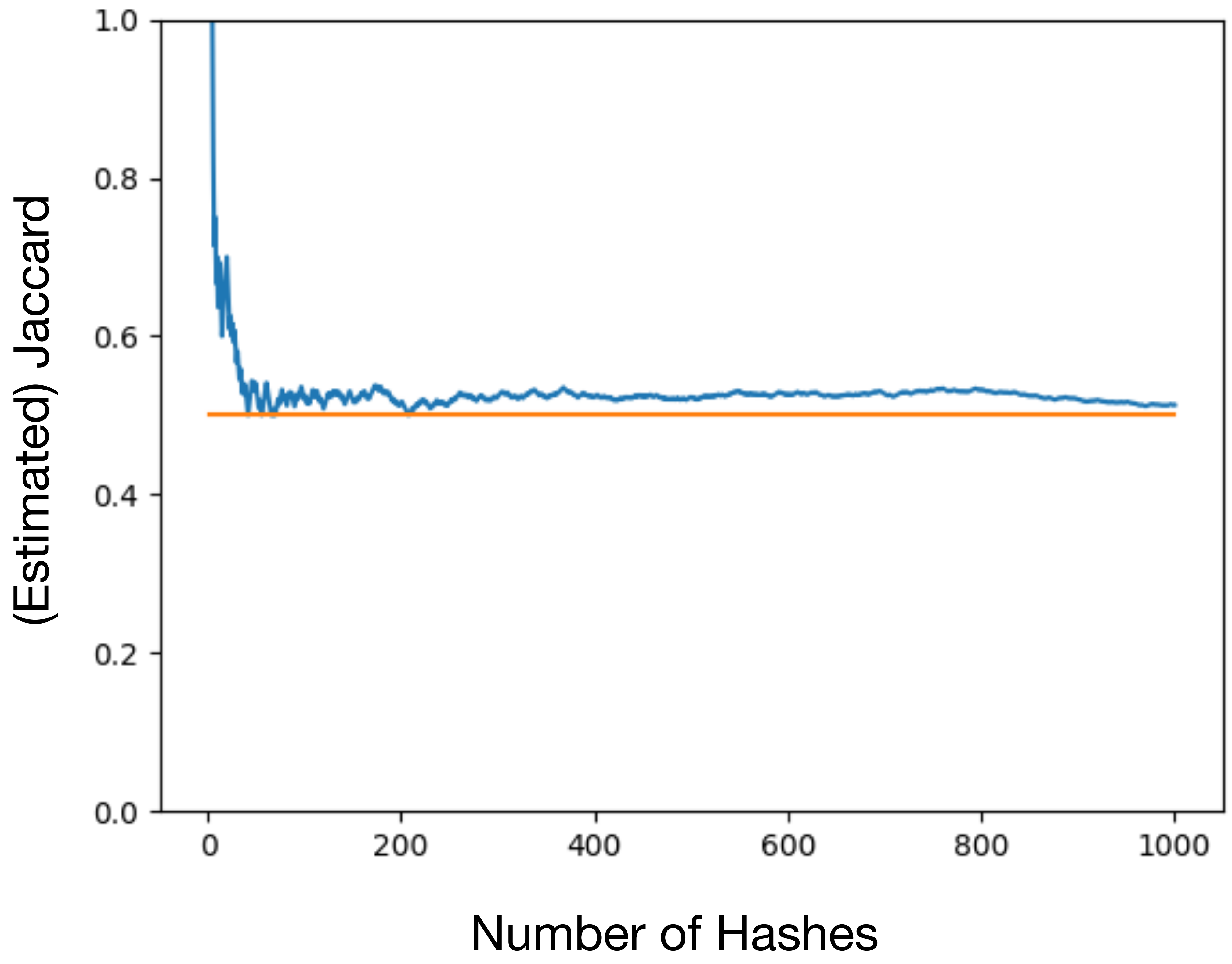
What fraction of those are in the intersection?



Min-Hash Sketch

As you increase the number of hashes, you will get closer to the estimate of the real jaccard value

Finding that many independent hashes may be hard

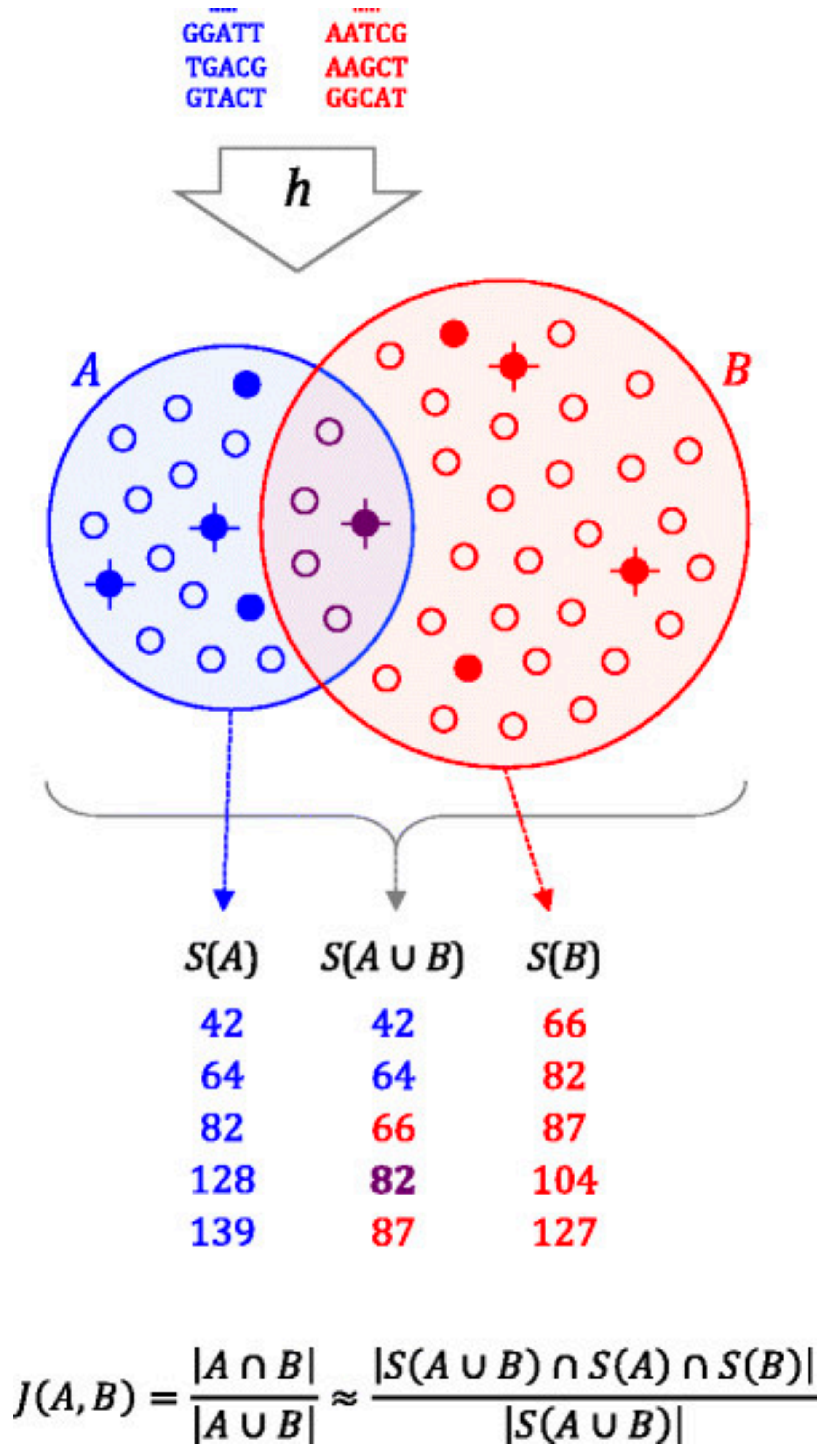


Min Hash Sketch with 1 Hash

The idea is that you choose the minimum *n* elements according to the hash *h*, and compute jaccard on these subsets

This subset of *k*-mers is called a "sketch"

Sometimes called "MinHash bottom sketching"



$S(A)$	$S(A \cup B)$	$S(B)$
42	42	66
64	64	82
82	66	87
128	82	104
139	87	127

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \approx \frac{|S(A \cup B) \cap S(A) \cap S(B)|}{|S(A \cup B)|}$$

Image credit: Ondov, et al. (2016) *Mash: Fast genome and metagenome distance estimation using MinHash.* **Genome Biology.**

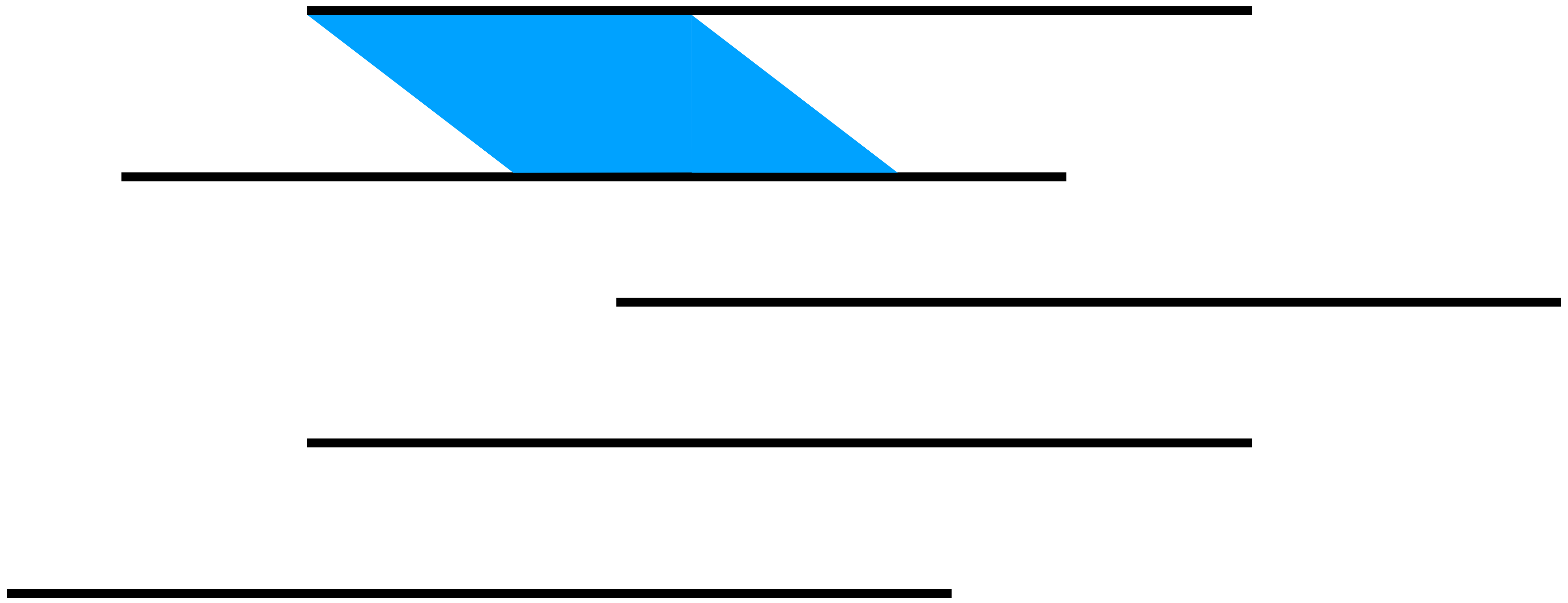
Minimizer Schemes

Another way to *sketch* a sequence is though the use of minimizer schemes

Here a set of *k*-mers for a sequence are selected by finding the minimum *k*-mer in overlapping windows

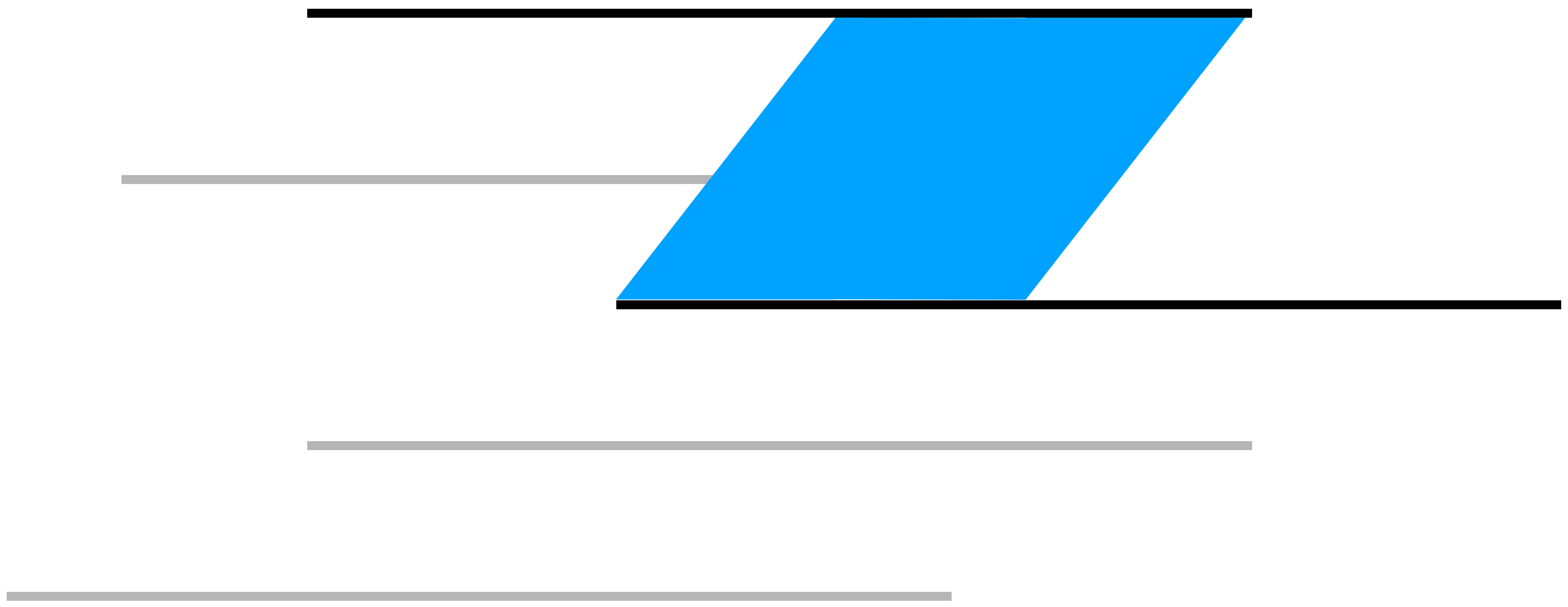
Minimizer Schemes

Roberts, *et al.* (2004) introduced minimizer schemes as a way to decrease the time needed for sequence overlap computation



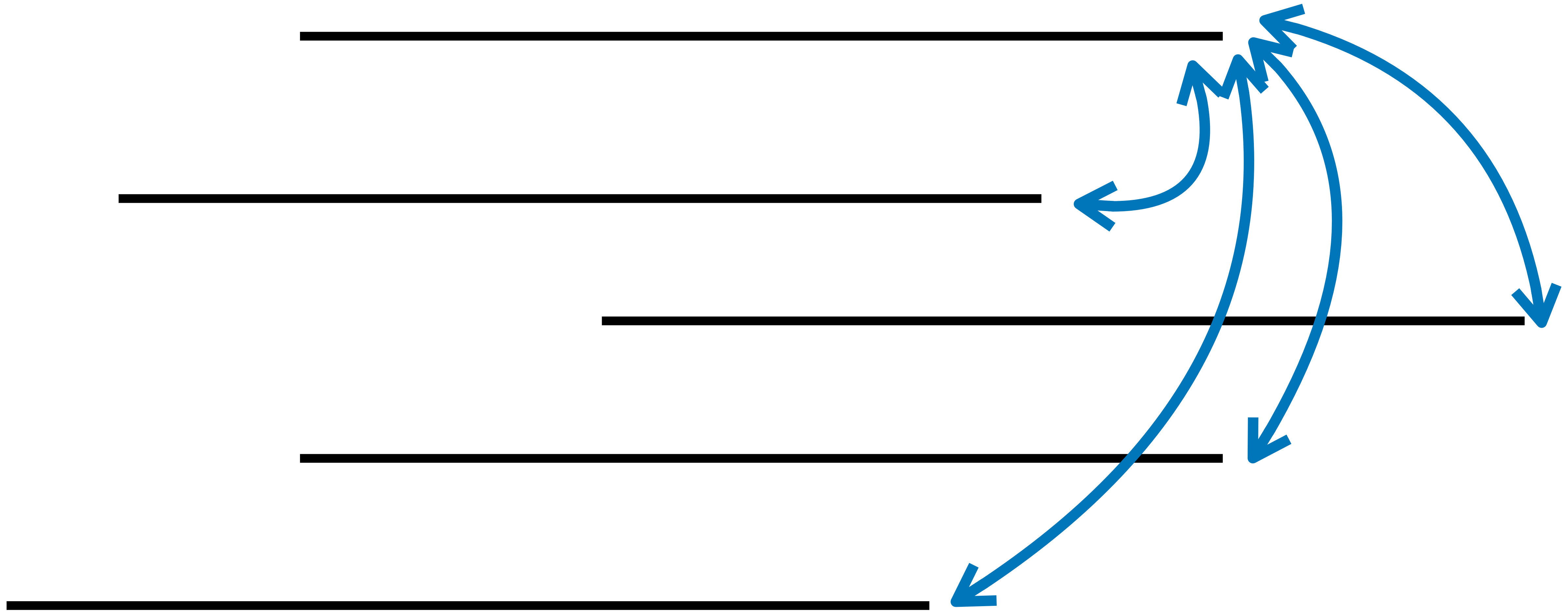
Minimizer Schemes

Roberts, *et al.* (2004) introduced minimizer schemes as a way to decrease the time needed for sequence overlap computation



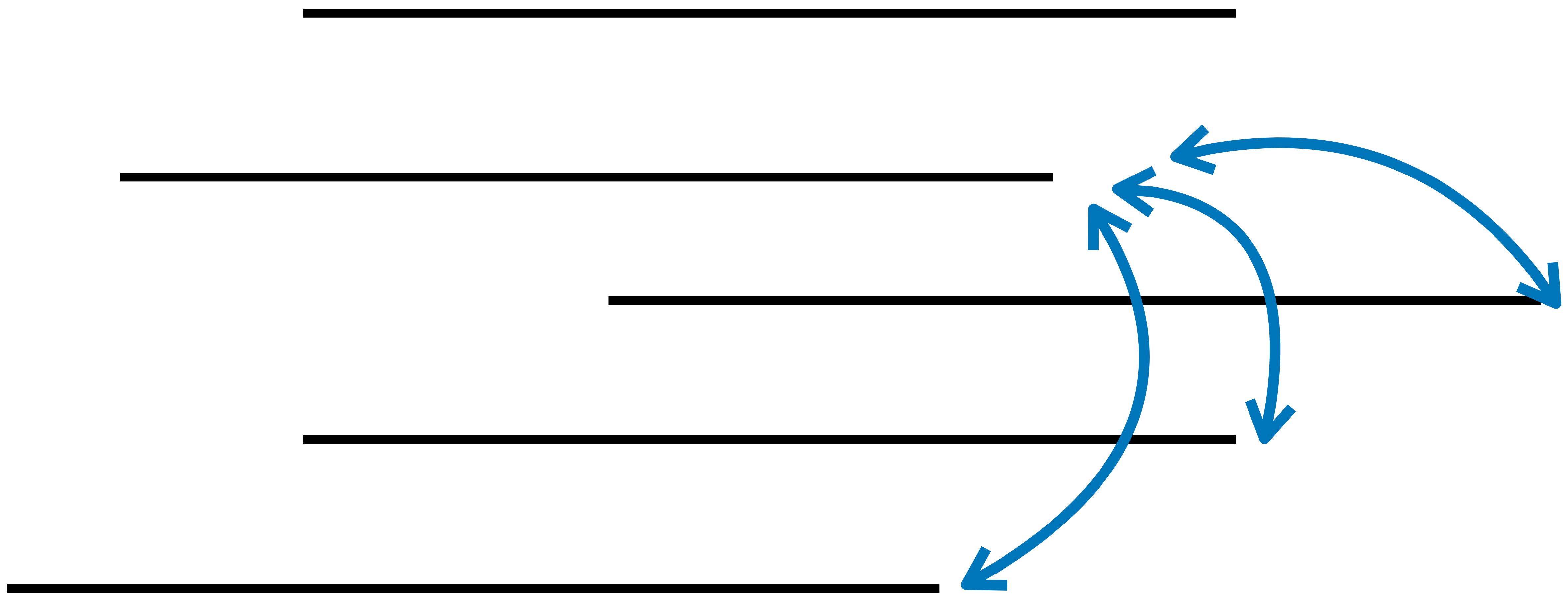
Minimizer Schemes

Roberts, *et al.* (2004) introduced minimizer schemes as a way to decrease the time needed for sequence overlap computation



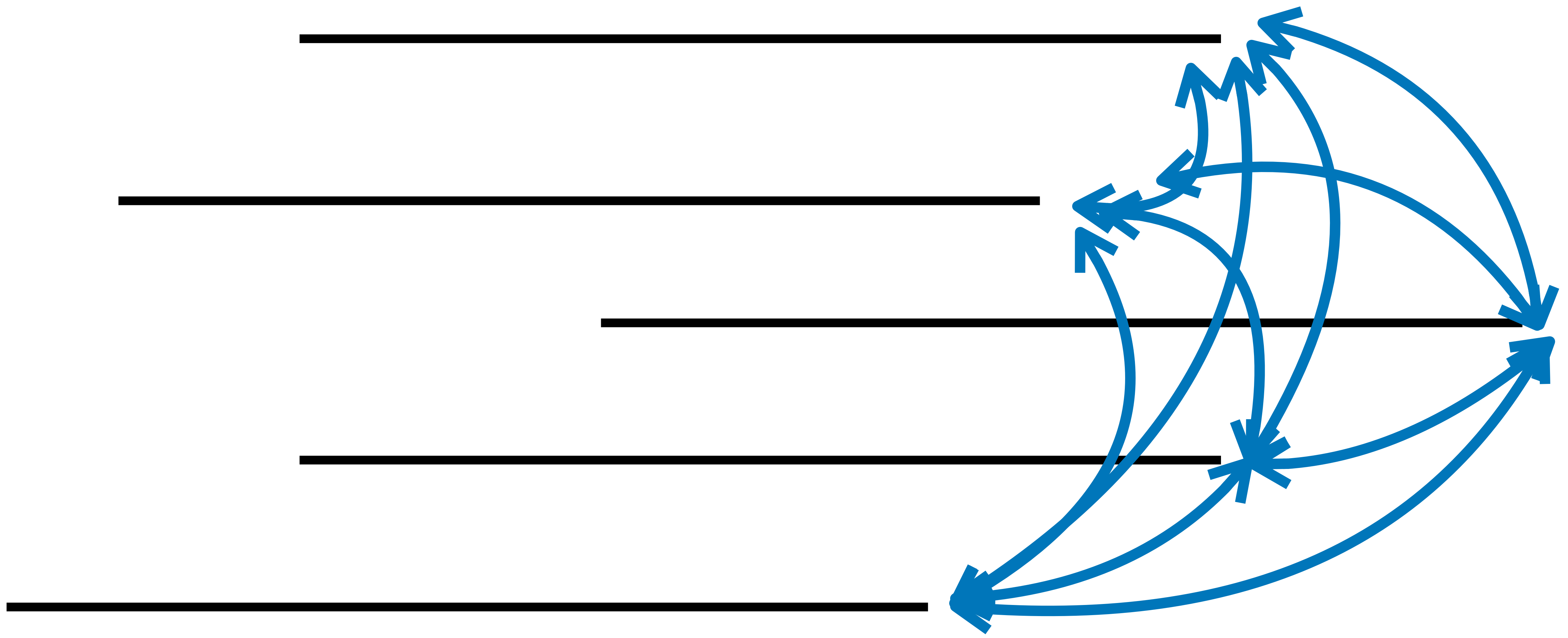
Minimizer Schemes

Roberts, *et al.* (2004) introduced minimizer schemes as a way to decrease the time needed for sequence overlap computation



Minimizer Schemes

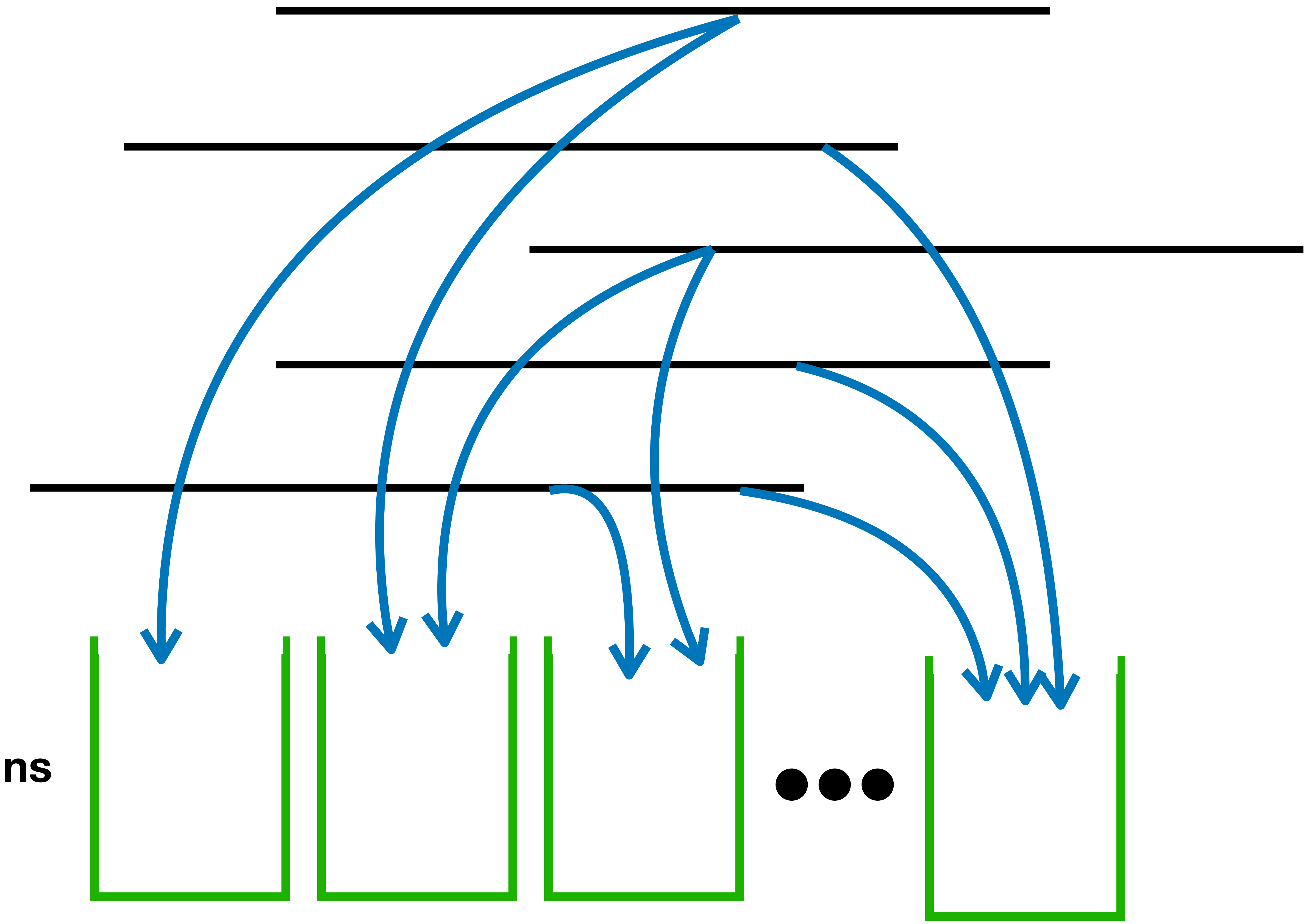
Roberts, *et al.* (2004) introduced minimizer schemes as a way to decrease the time needed for sequence overlap computation



O(n^2) alignments!

Minimizer Schemes

Roberts, *et al.* (2004) introduced minimizer schemes as a way to decrease the time needed for sequence overlap computation



Only compare within bins

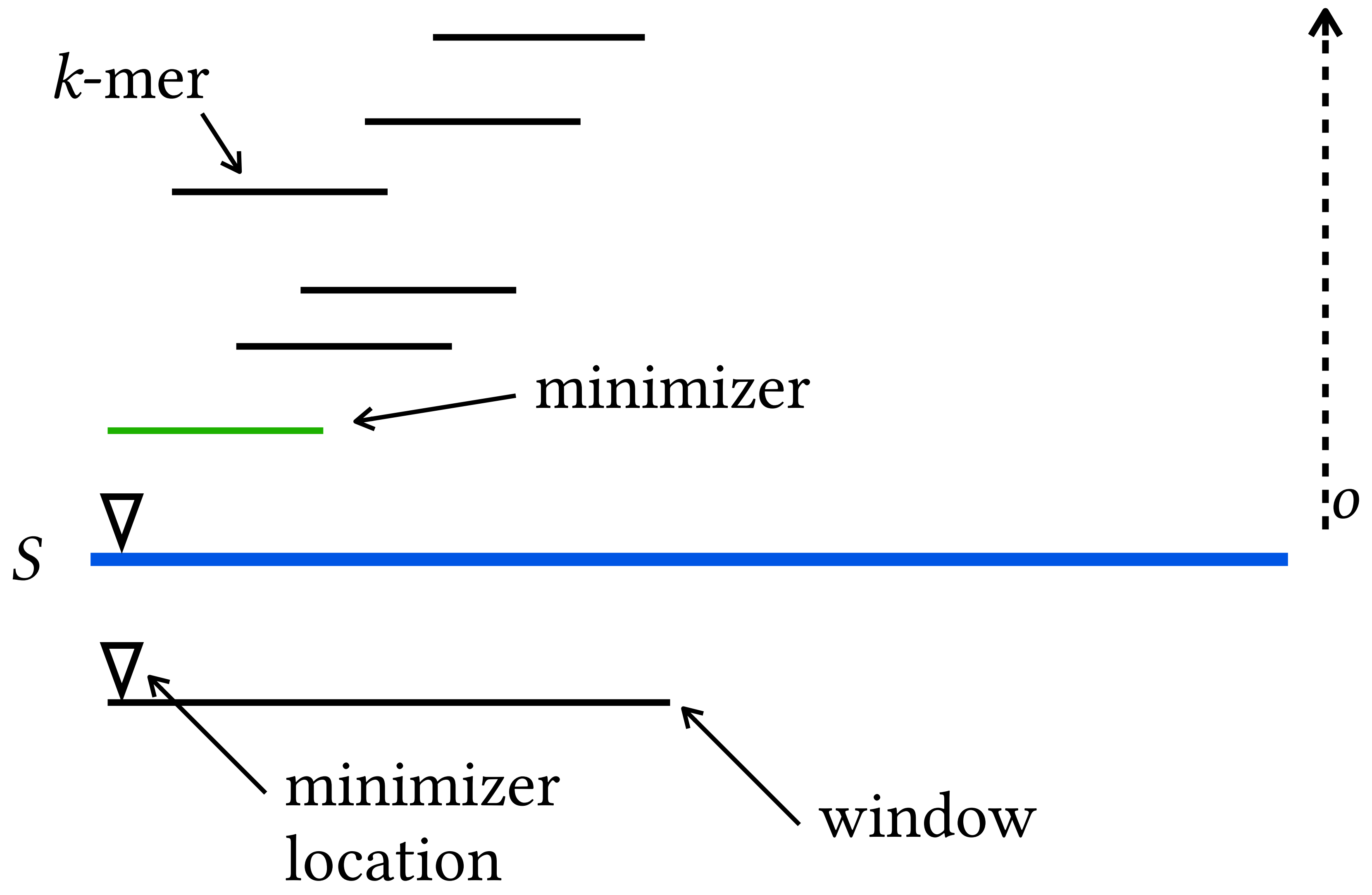
Minimizer Schemes

- Minimizer schemes have two special properties:
  - two sequences with a long exact match must select the same k-mers
  - there are no large gap between selected k-mers

- Use in k-mer counting, *de Brujin* graph construction, data structure sparsification, etc.

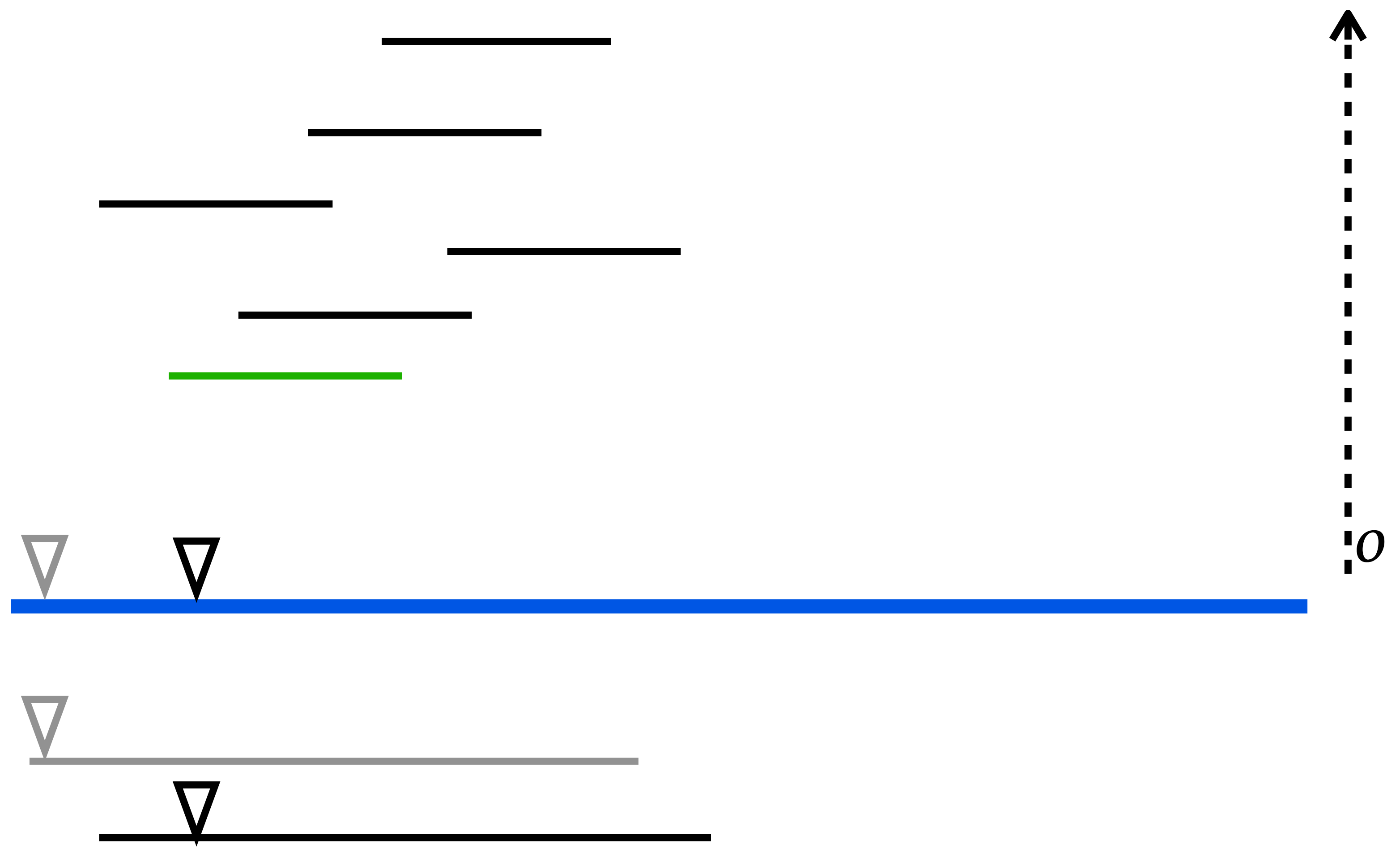
Minimizer Schemes

For a windows of w consecutive k-mers from a sequence S, a minimizer scheme selects the minimum according to an ordering o as a representative



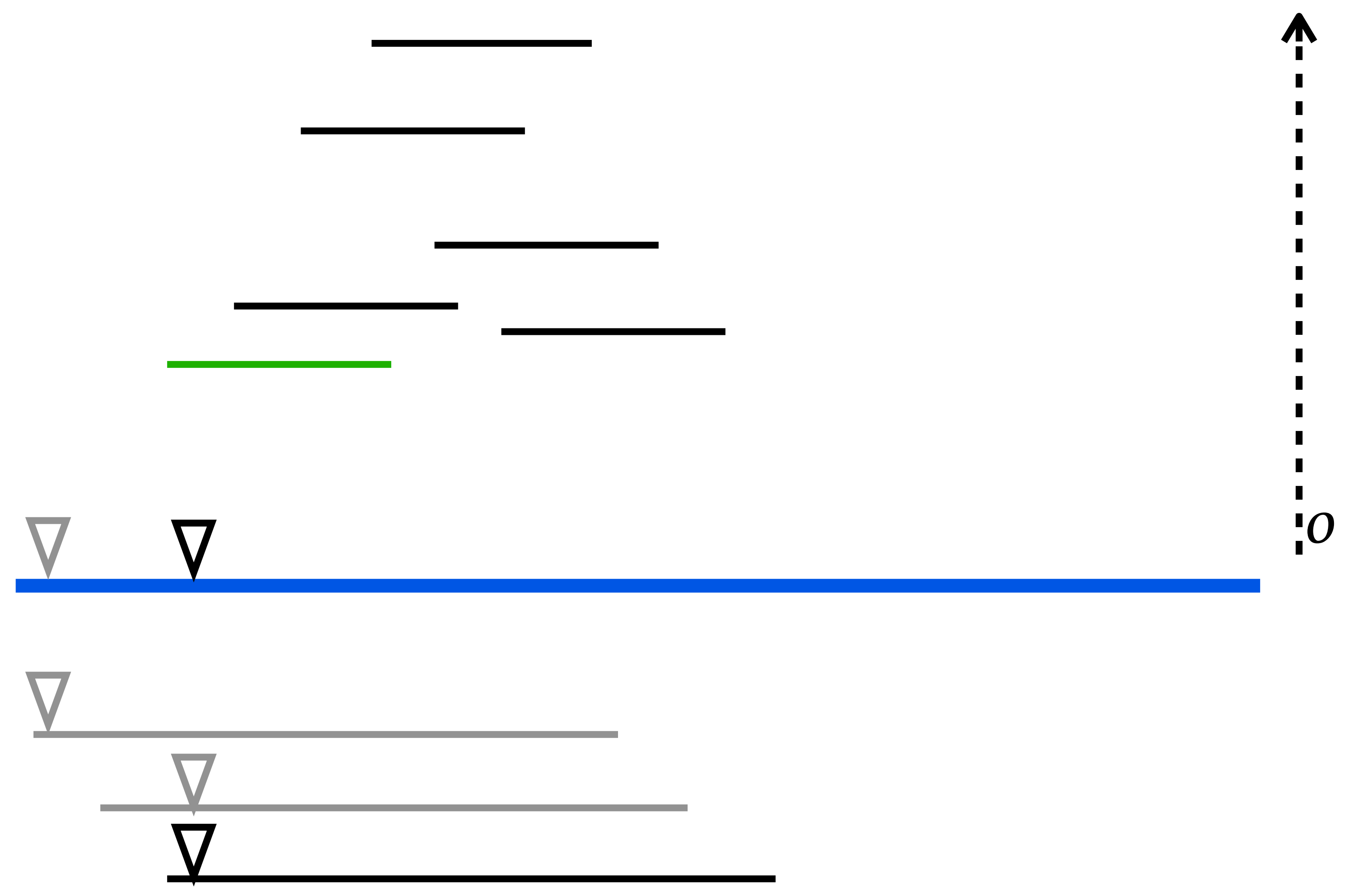
Minimizer Schemes

For a windows of w consecutive k-mers from a sequence S, a minimizer scheme selects the minimum according to an ordering o as a representative



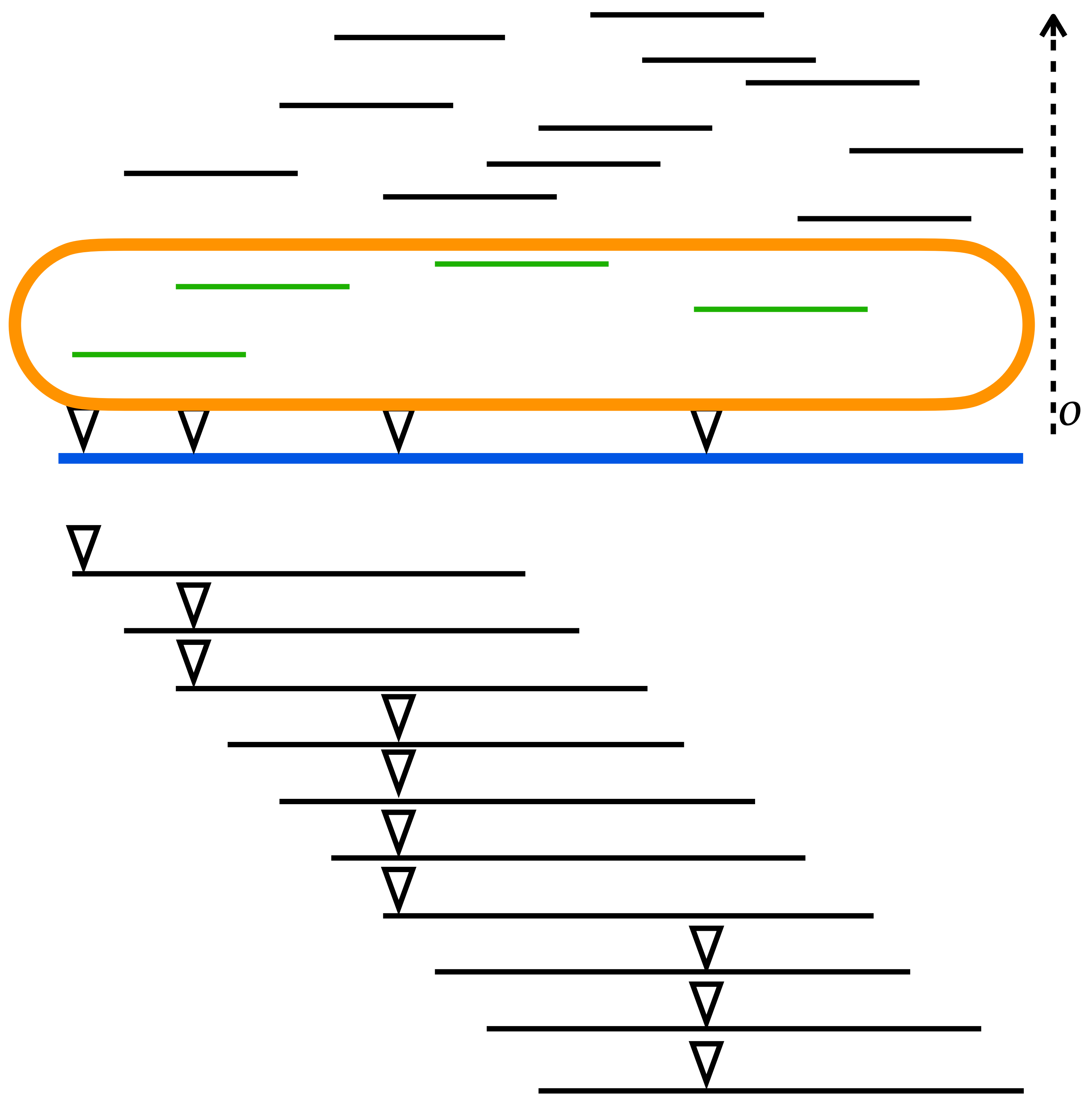
Minimizer Schemes

For a windows of w consecutive k-mers from a sequence S, a minimizer scheme selects the minimum according to an ordering o as a representative



Minimizer Schemes

Only some k-mers are used a minimizers



- Changing the ordering used can greatly impact the number of unique minimizers
- Can we find an order that minimizes the number of minimizer locations

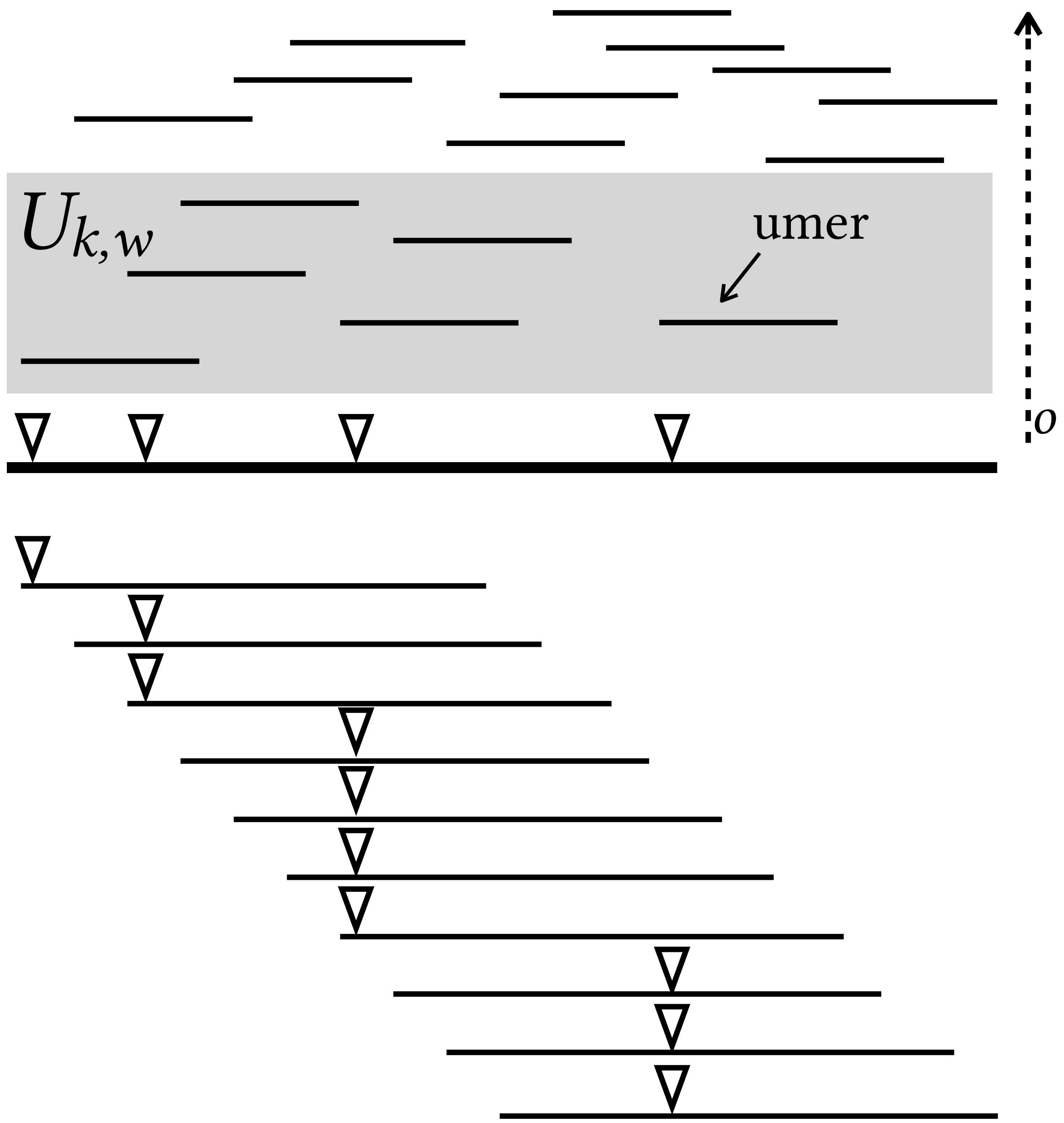
Universal k-mer Set

A universal k-mer set $U_{k,w} \subseteq \Sigma^k$ is a set of k-mers such that any window of w consecutive k-mers must contain at least one element from the set

Universal k-mer Set and Minimizer Ordering

- A universal k-mer set induces a family of compatible orderings

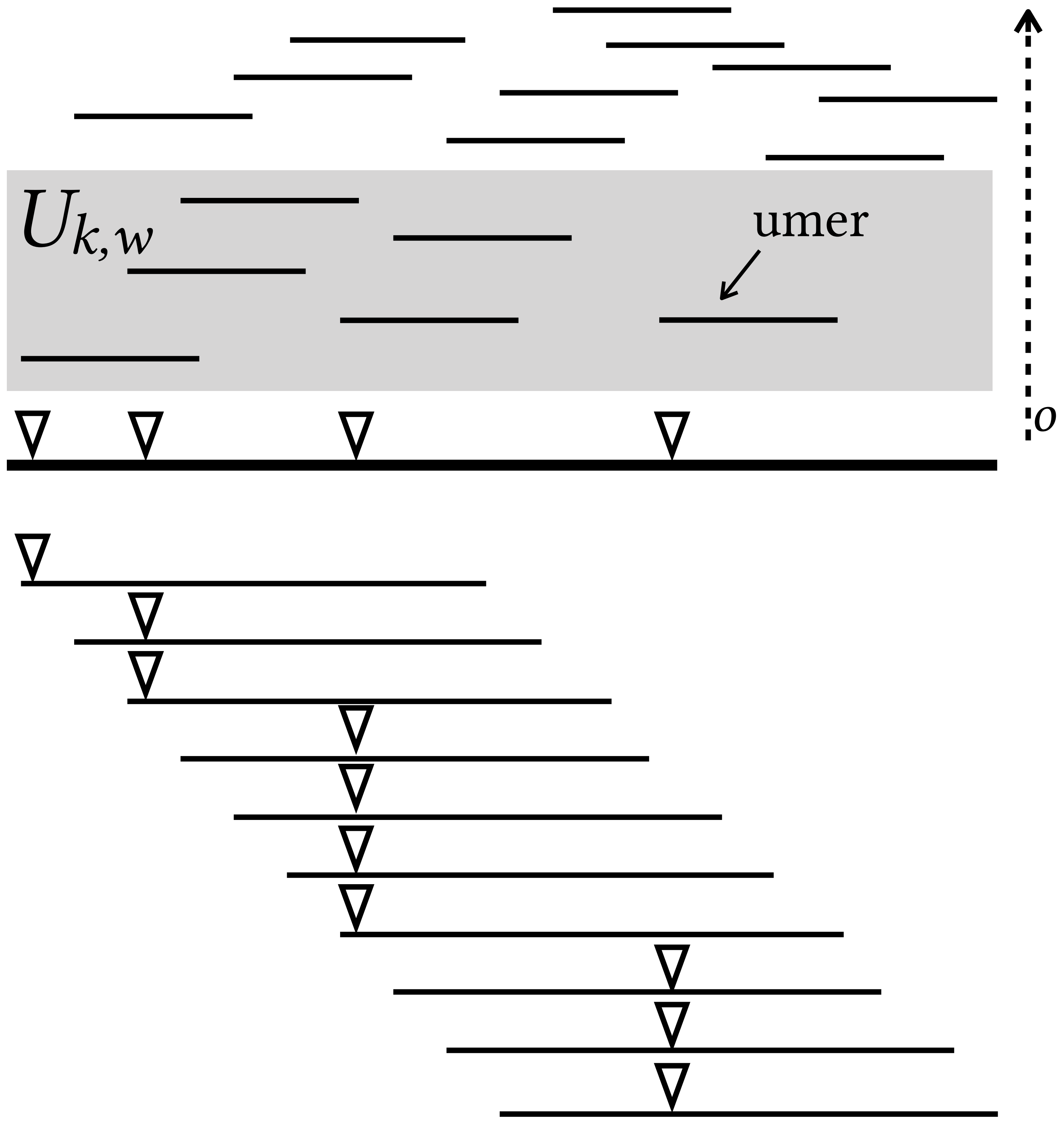
- Orderings based on universal sets have better performance then lexicographic or random orders (Marçais, et al., 2017)



Universal k-mer Set and Minimizer Ordering

- A universal k-mer set induces a family of compatible orderings

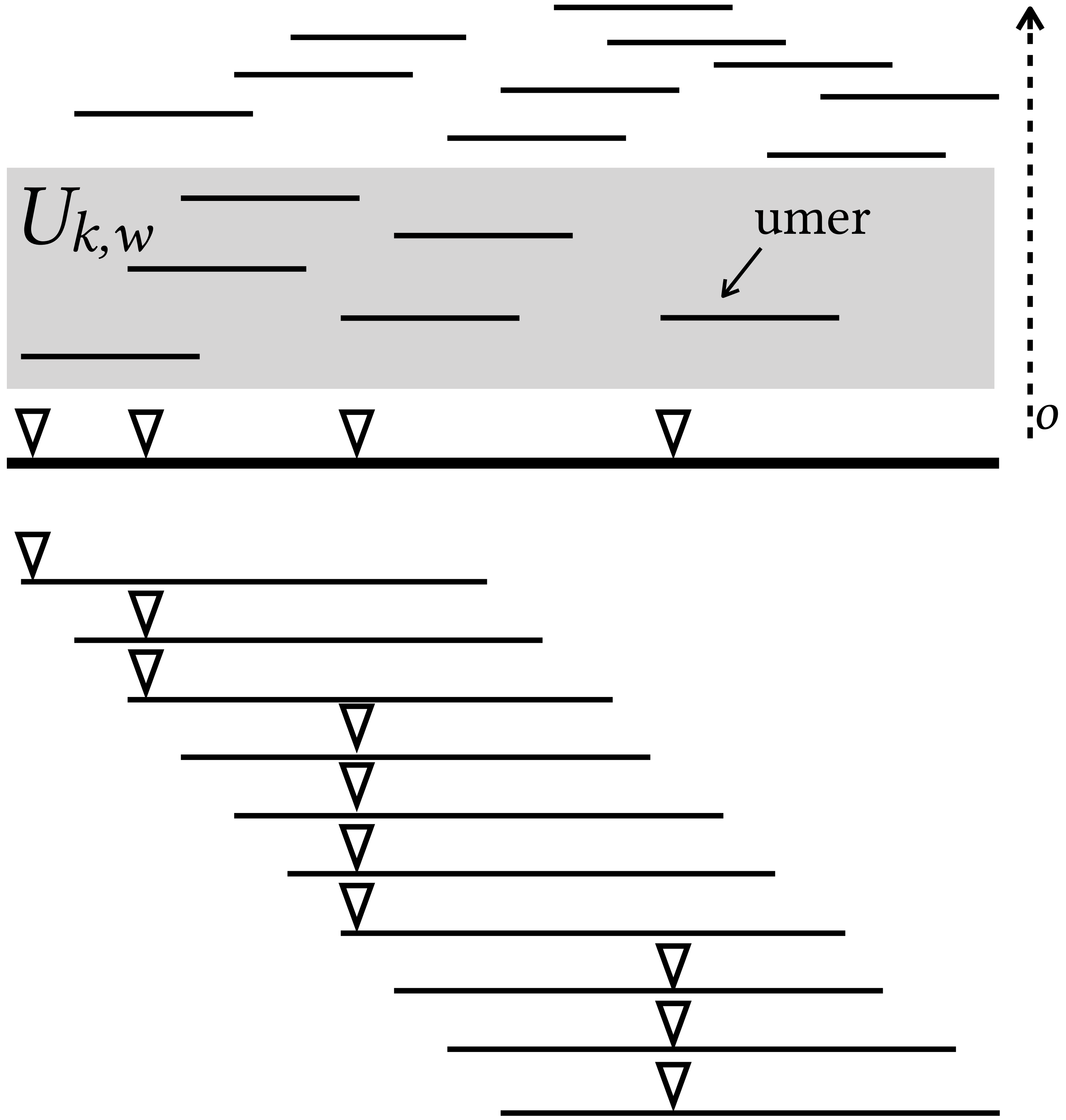
- Orderings based on universal sets have better ***performance*** then lexicographic or random orders (Marçais, et al., 2017)



Universal k-mer Set and Minimizer Ordering

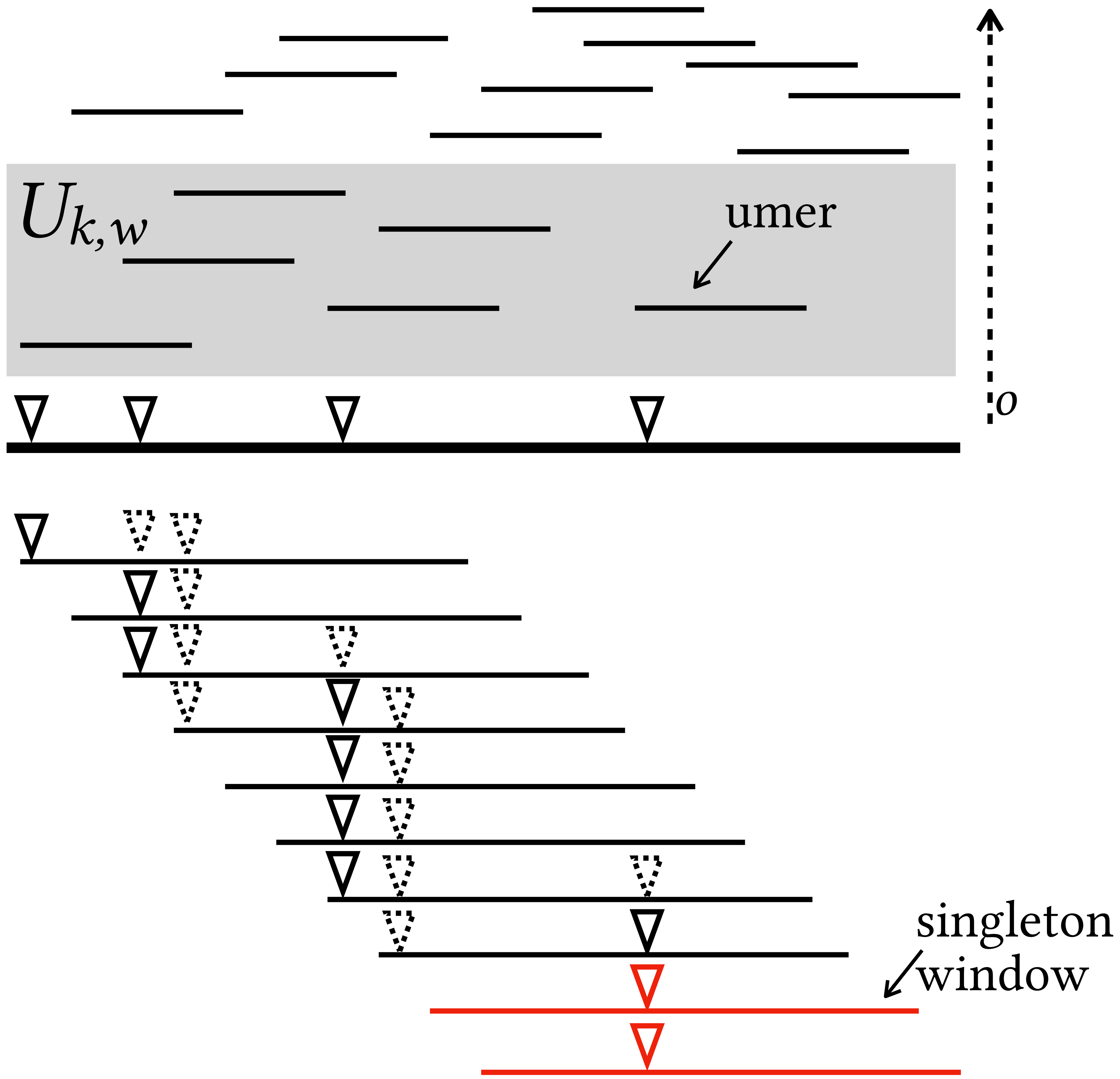
- Set Size
  - Fraction of all k-mers in the universal set

- Density
  - Normalized count of minimizer locations in S



Universal k-mer Set and Minimizer Ordering

- Set Size
  - Fraction of all k-mers in the universal set
- Density
  - Normalized count of minimizer locations in S
- Sparsity
  - Normalized count of windows in S with only one umer (universal k-mer)

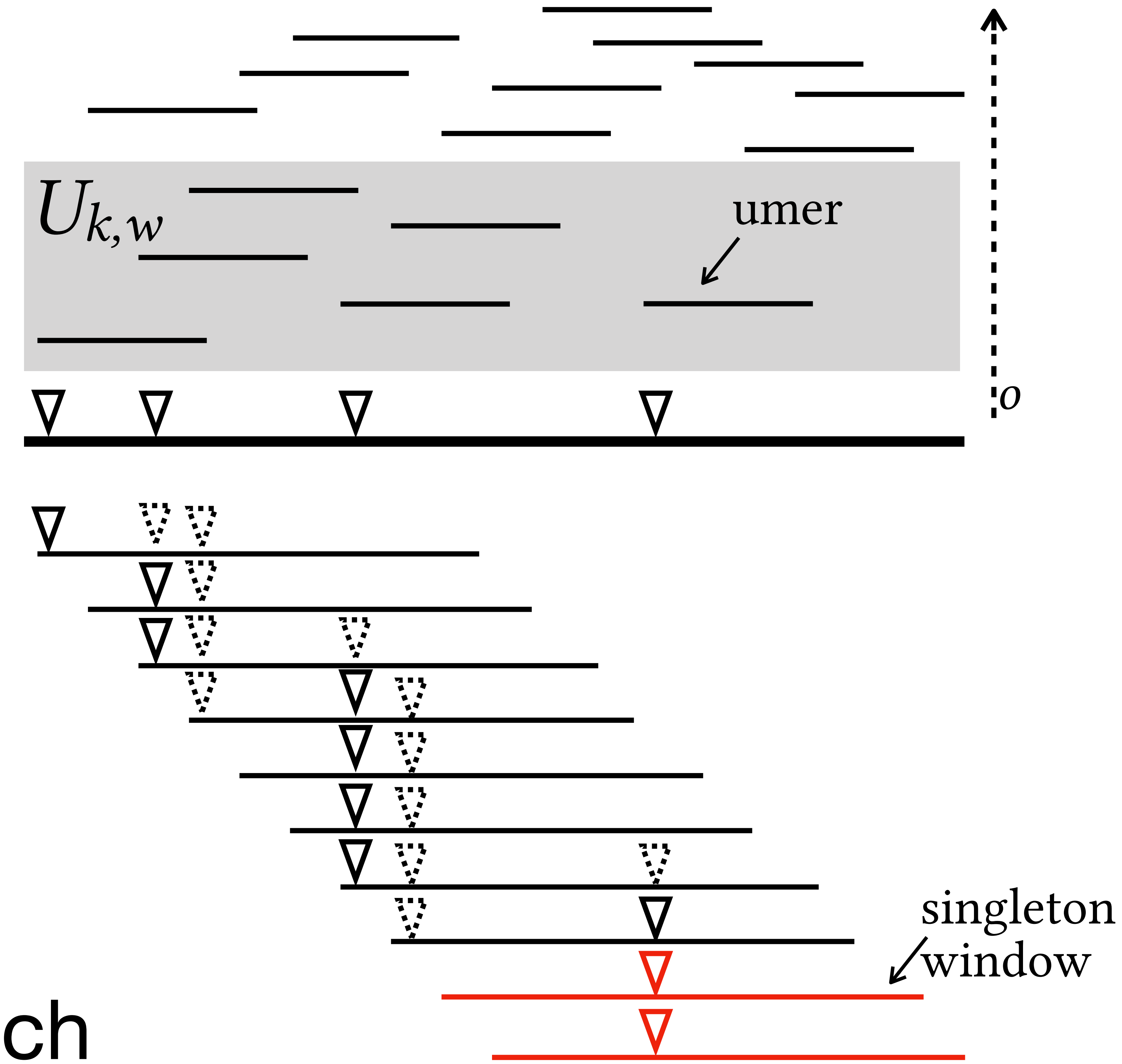


Universal k-mer Set and Minimizer Ordering

- Set Size
  - Fraction of all k-mers in the universal set

- ***Expected*** Density
  - Normalized count of minimizer locations in B_L

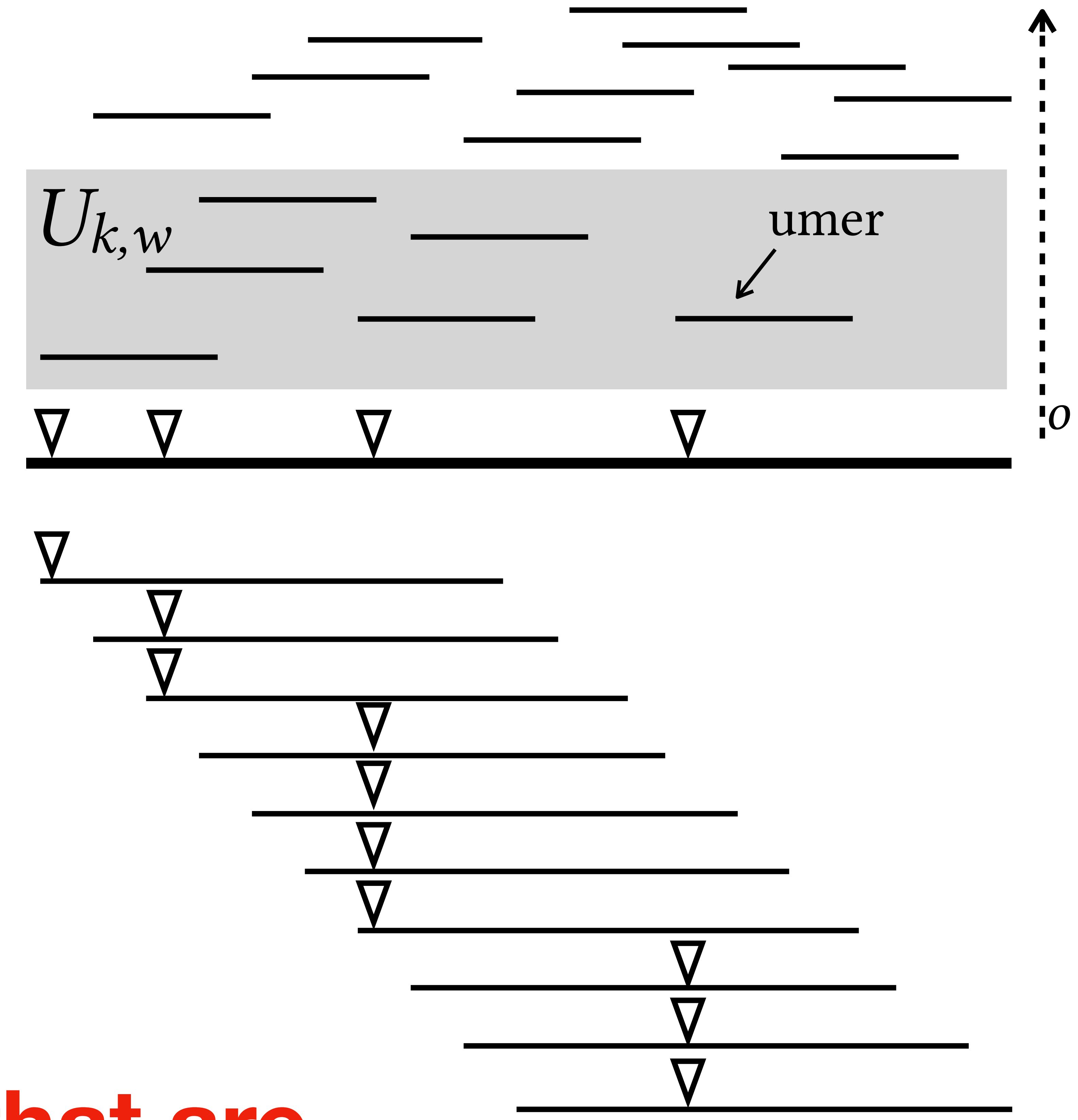
- ***Expected*** Sparsity
  - Normalized count of windows in B_L with only one umer (universal k-mer)



B_L is the **de Brujin** sequence of order L, it contains each window exactly once

Universal k-mer Set and Minimizer Ordering

- A universal k-mer set induces a family of compatible orderings
- Orderings based on universal sets have better performance then lexicographic or random orders (Marçais, et al., 2017)
- Current methods cannot construct sets for values of k and w used in practice



Can we construct universal k-mer sets that are practical for use in minimizer schemes?

Detour: De Brujin graphs

though we call them De Brujin graphs they were independently described by Nicolaas Govert de Bruijn and Irving John Good in 1946

they are used to encode sequence information as paths in a graph

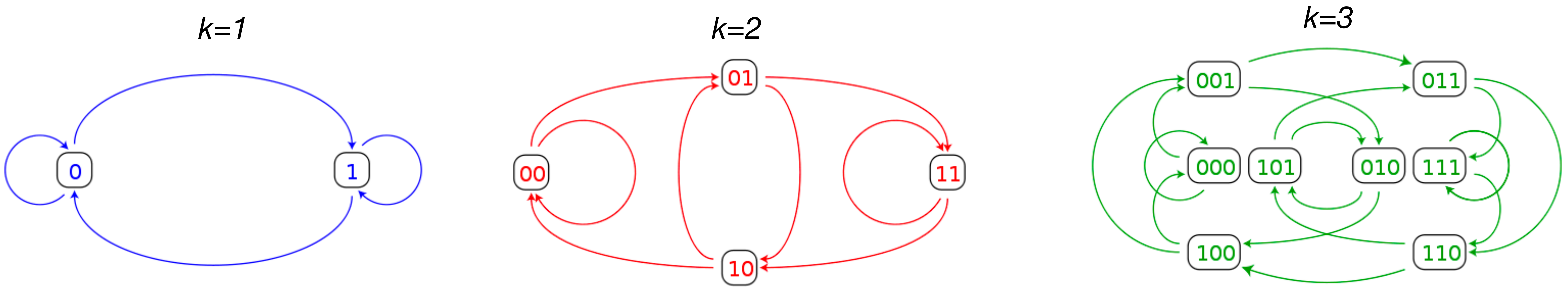
Definition a *k*-order de Brujin Graph (DBG) $D = (V,E)$ has:

- $V = \Sigma^k$ -- there is a vertex for each possible *k*-mer
- $E = \{ax \rightarrow xb \mid a,b \in \Sigma,\ x \in \Sigma^{(k-1)}\}$ -- for each $(k+1)$-mer *axb*, there is an edge from the *k*-mer *ax* to the *k*-mer *xb*

De Brujin Graphs

Definition a *k*-order de Brujin Graph (DBG) $D = (V,E)$ has:

- $V = \Sigma^k$ -- there is a vertex for each possible *k*-mer
- $E = \{ax \rightarrow xb \mid a,b \in \Sigma,\ x \in \Sigma^{(k-1)}\}$ -- for each $(k+1)$-mer *axb*, there is an edge from the *k*-mer *ax* to the *k*-mer *xb*



k=1

k=2

k=3

Image courtesy commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

De Brujin Graphs

Definition a k-order de Brujin Graph (DBG) $D = (V,E)$ has:

- $V = \Sigma^k$ -- there is a vertex for each possible k-mer
- $E = \{ax \rightarrow xb \mid a,b \in \Sigma,\ x \in \Sigma^{(k-1)}\}$ -- for each $(k+1)$-mer axb, there is an edge from the k-mer ax to the k-mer xb

Each node has σ outgoing edges, and σ incoming edges

k=2

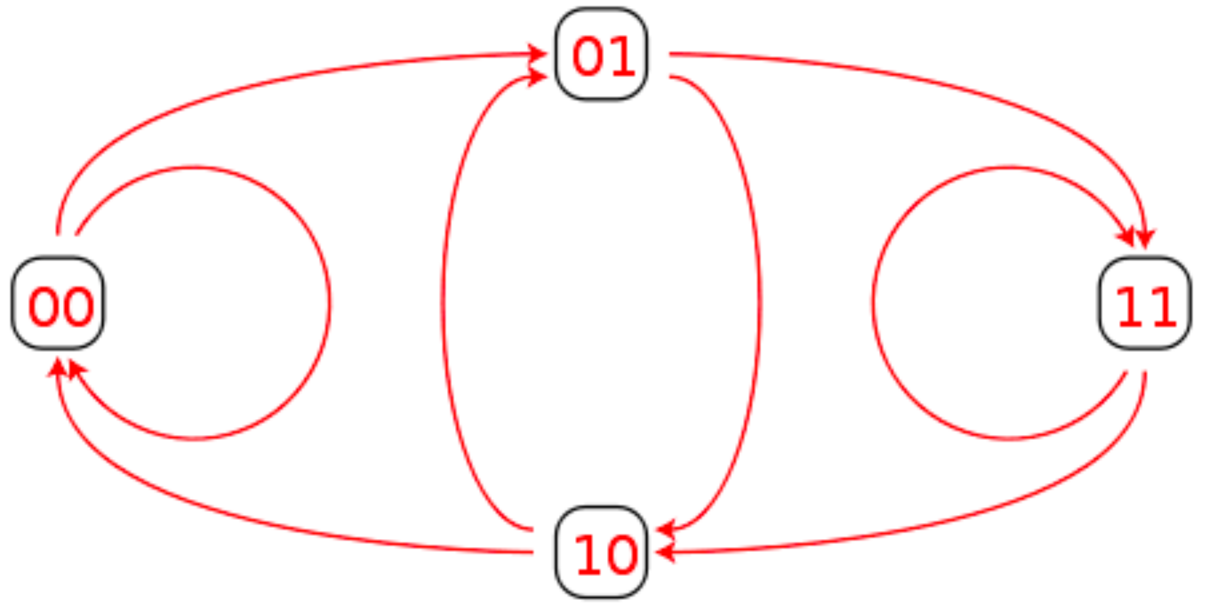


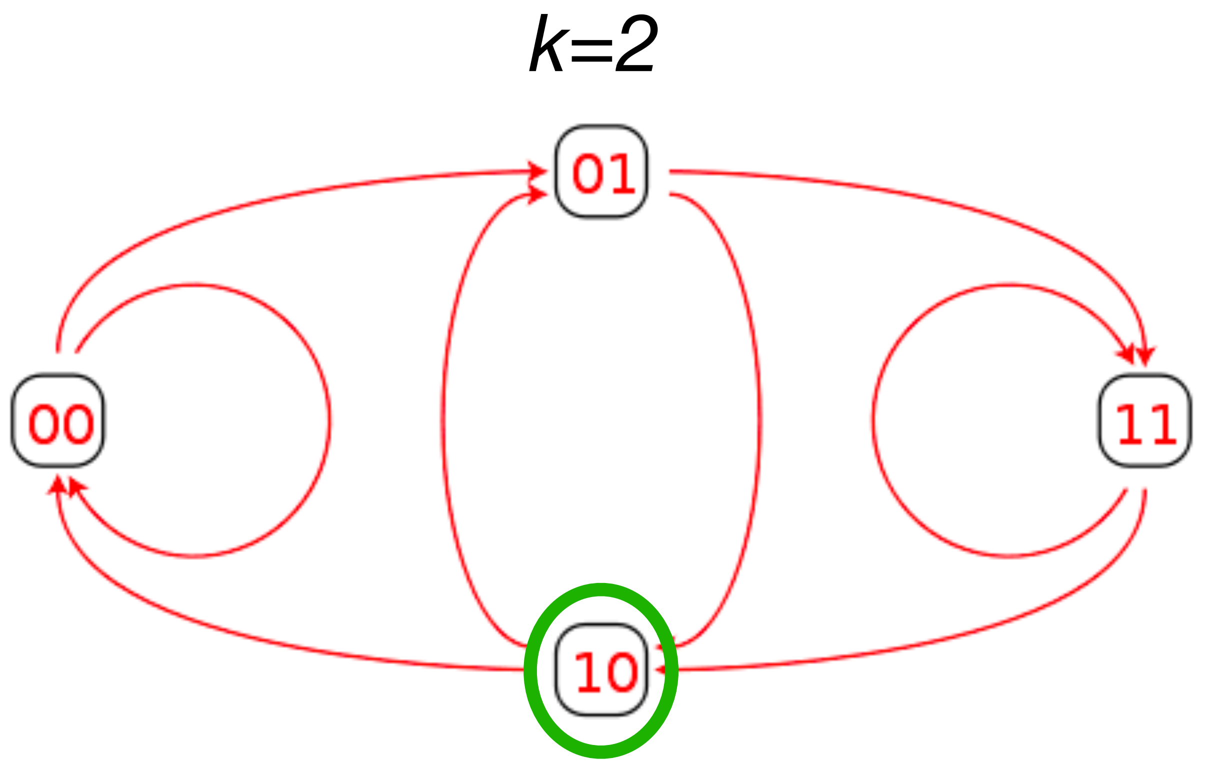
Image courtesy commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

De Brujin Graphs

Definition a *k*-order de Brujin Graph (DBG) $D = (V,E)$ has:

- $V = \Sigma^k$ -- there is a vertex for each possible *k*-mer
- $E = \{ax \rightarrow xb \mid a,b \in \Sigma,\ x \in \Sigma^{(k-1)}\}$ -- for each $(k+1)$-mer *axb*, there is an edge from the *k*-mer *ax* to the *k*-mer *xb*

Each node has σ outgoing edges, and σ incoming edges



Any string over the alphabet can be encoded as a path on the DBG

Example: 1011000

Image courtesy commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

Other properties for DBGs we won't use for assembly

the de Brujin of order *k* is a **line graph** of the debrujin graph of order *k-1*

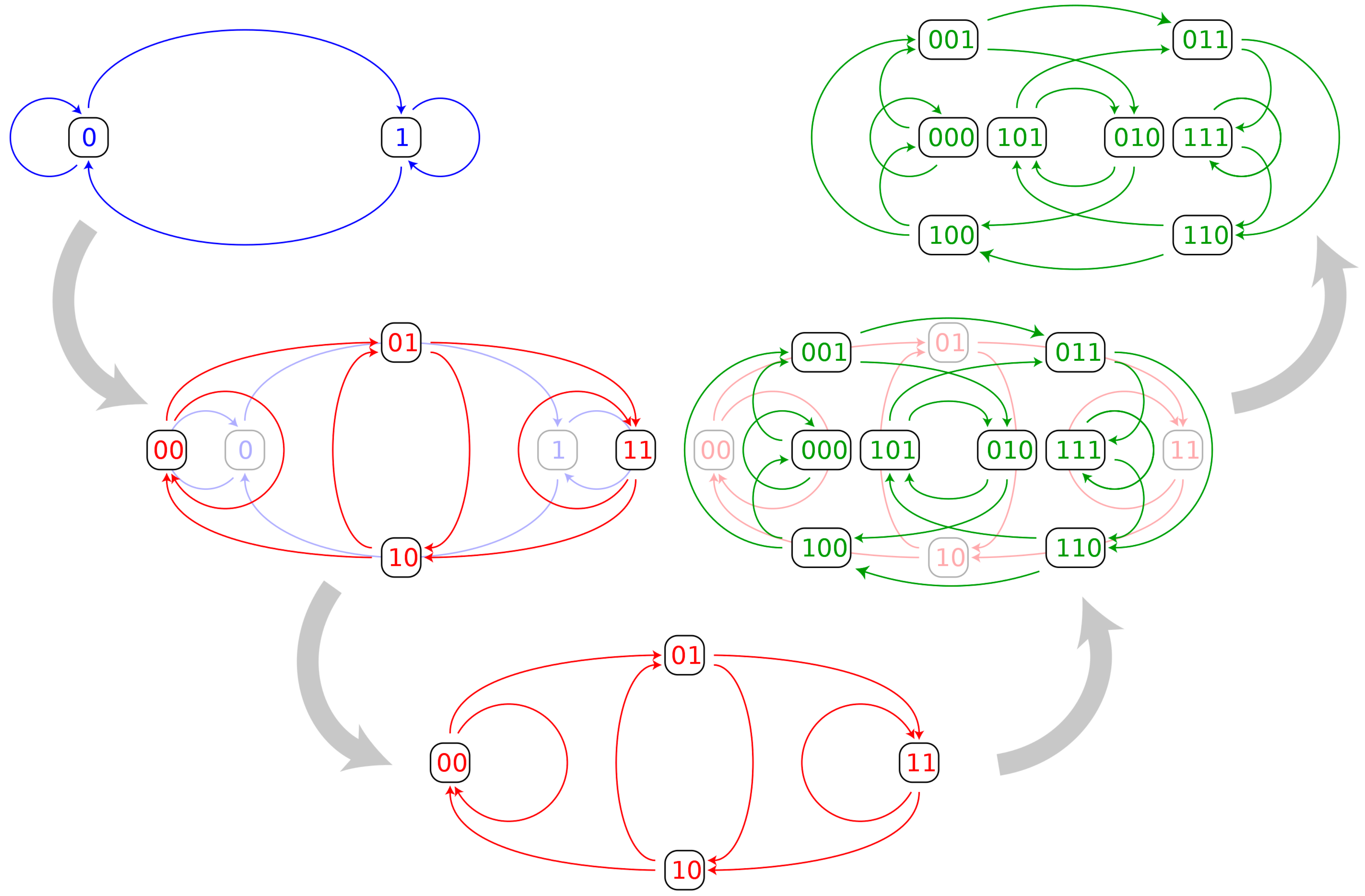


Image courtesy commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

Other properties for DBGs we won't use for assembly

A de Brujin *sequence* is an *Hamiltoninan* path of the graph, meaning it contains all *k*-mers exactly once

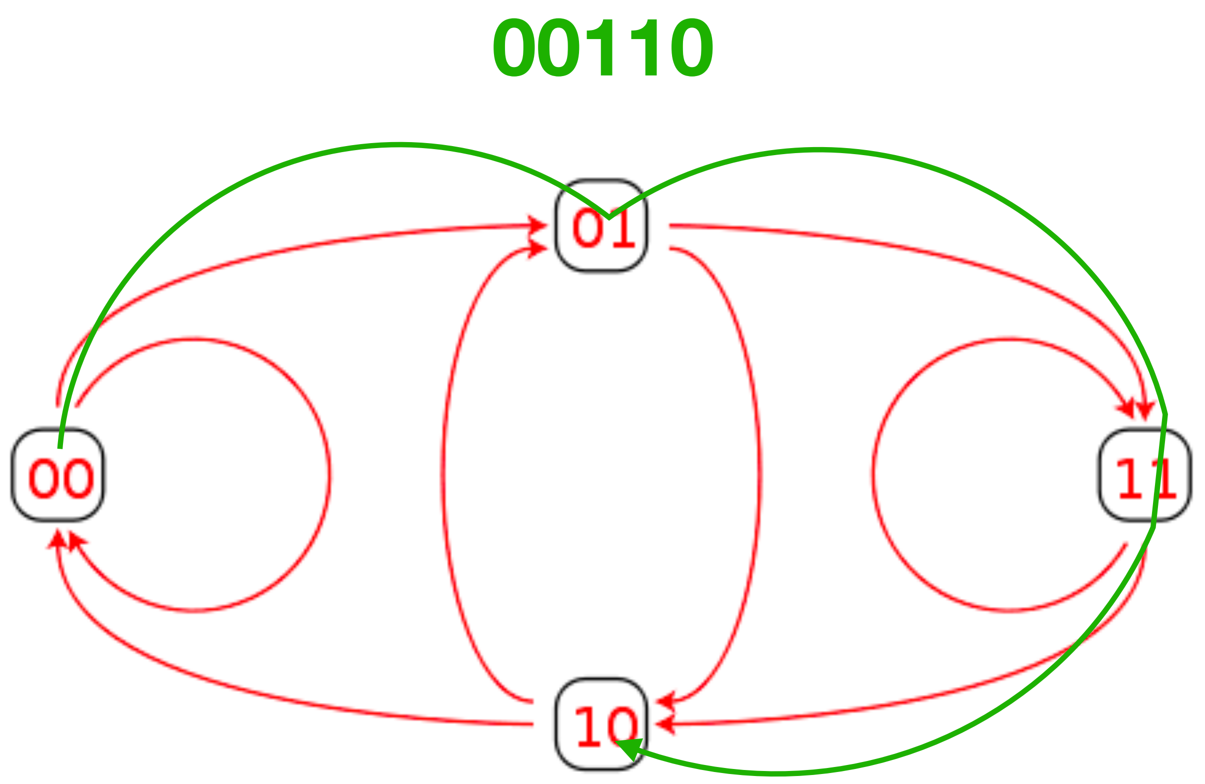
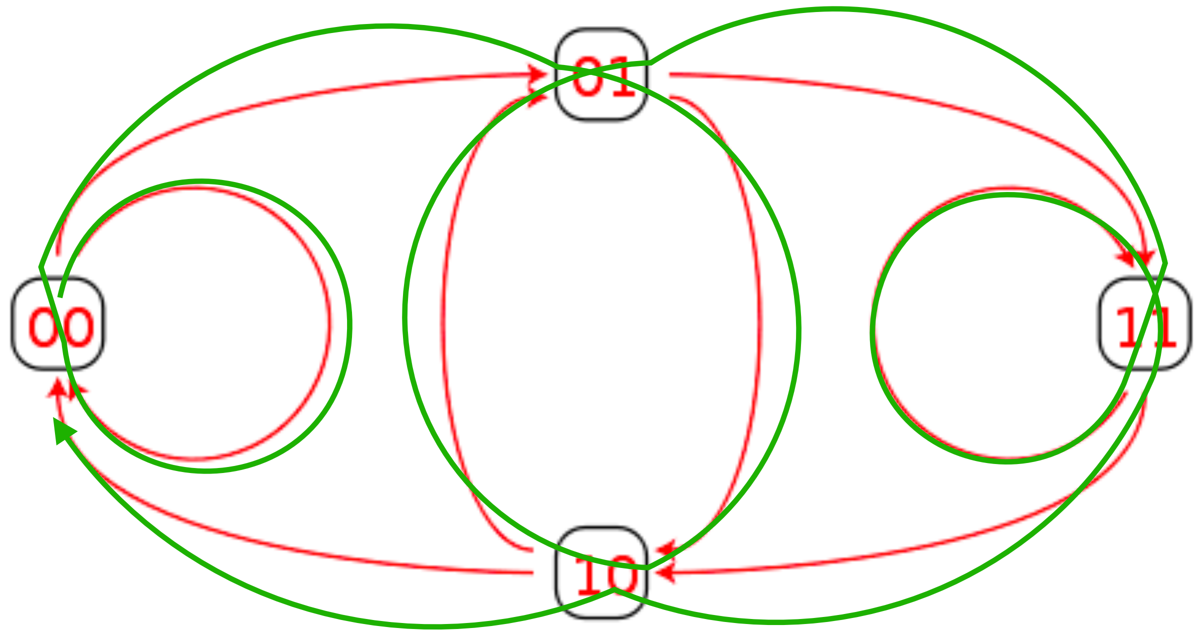


Image courtesy commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

Other properties for DBGs we won't use for assembly

A de Brujin *sequence* is an *Hamiltoninan* path of the graph, meaning it contains all *k*-mers exactly once

- or the *Eulerian* path of the graph of *k-1*



0001011100

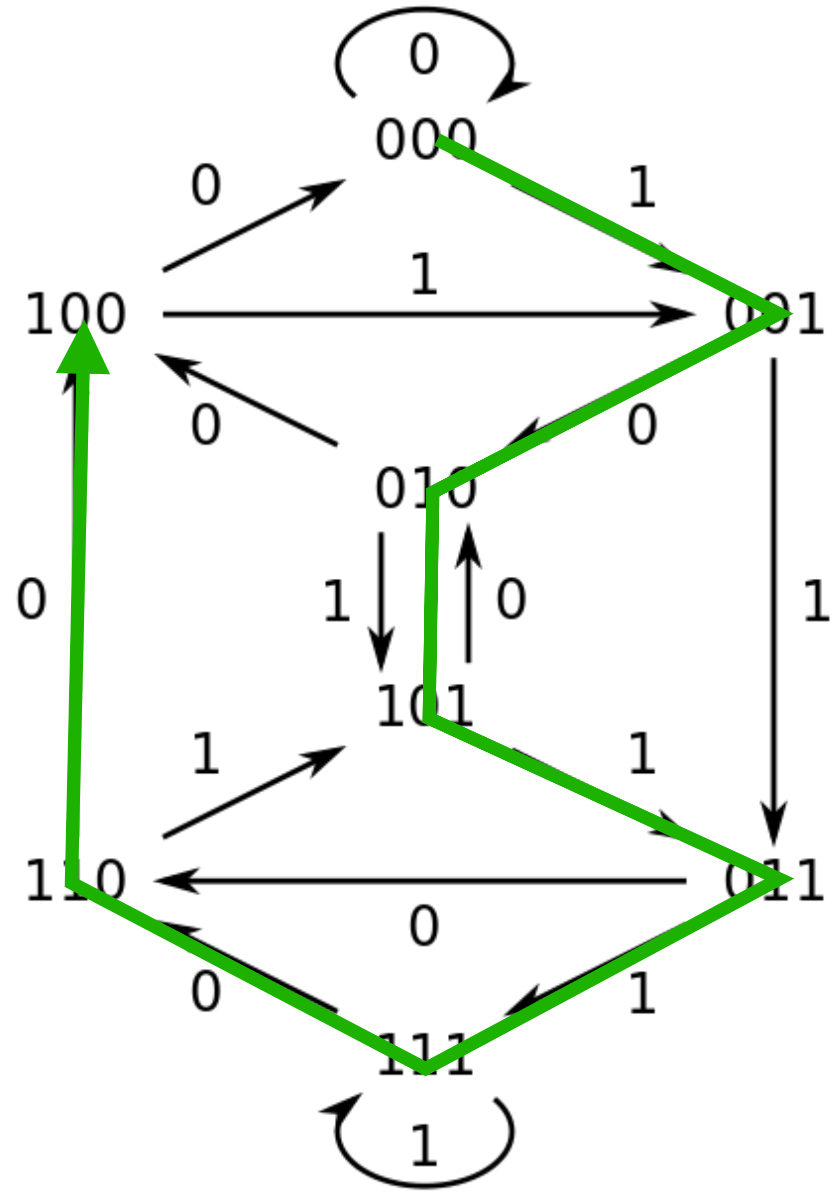


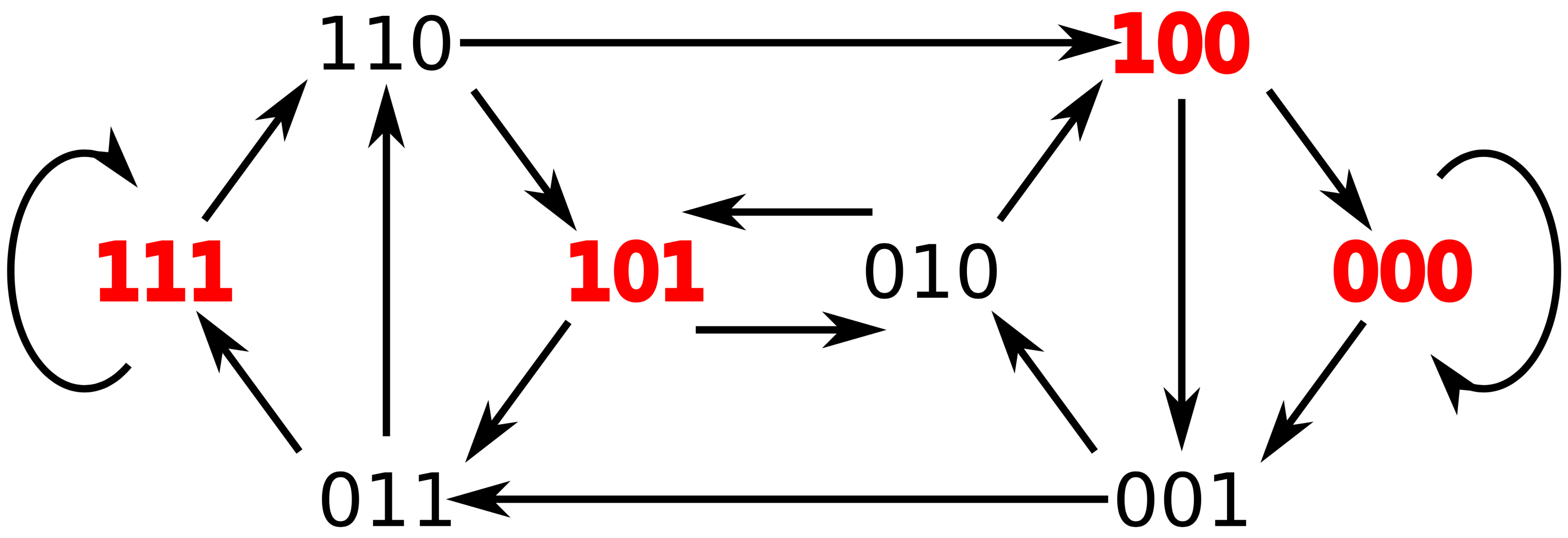
Image courtesy commons.wikimedia.org/wiki/File:Debruijngraph.gif

Image courtesy commons.wikimedia.org/wiki/File:DeBruijn-as-line-digraph.svg

Other properties for DBGs we won't use for assembly

a decycling set of edges a de Brujin graph is a set of nodes that when removed leave a DAG

- this set of *k*-mers is guaranteed to exist in all long enough sequences



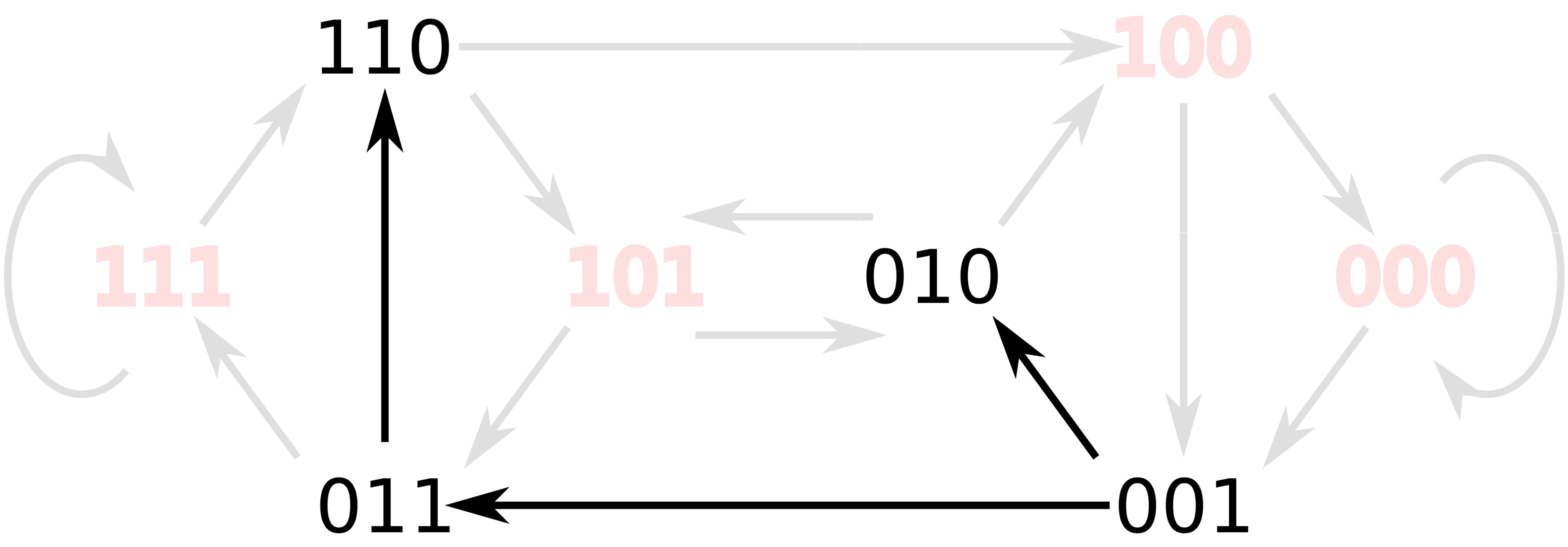
Can you find a length 6 binary sequence, which does not intersect one of the red *k*-mers?

Image courtesy of Guillaume Marçais

Other properties for DBGs we won't use for assembly

a decycling set of edges a de Brujin graph is a set of nodes that when removed leave a DAG

- this set of *k*-mers is guaranteed to exist in all long enough sequences



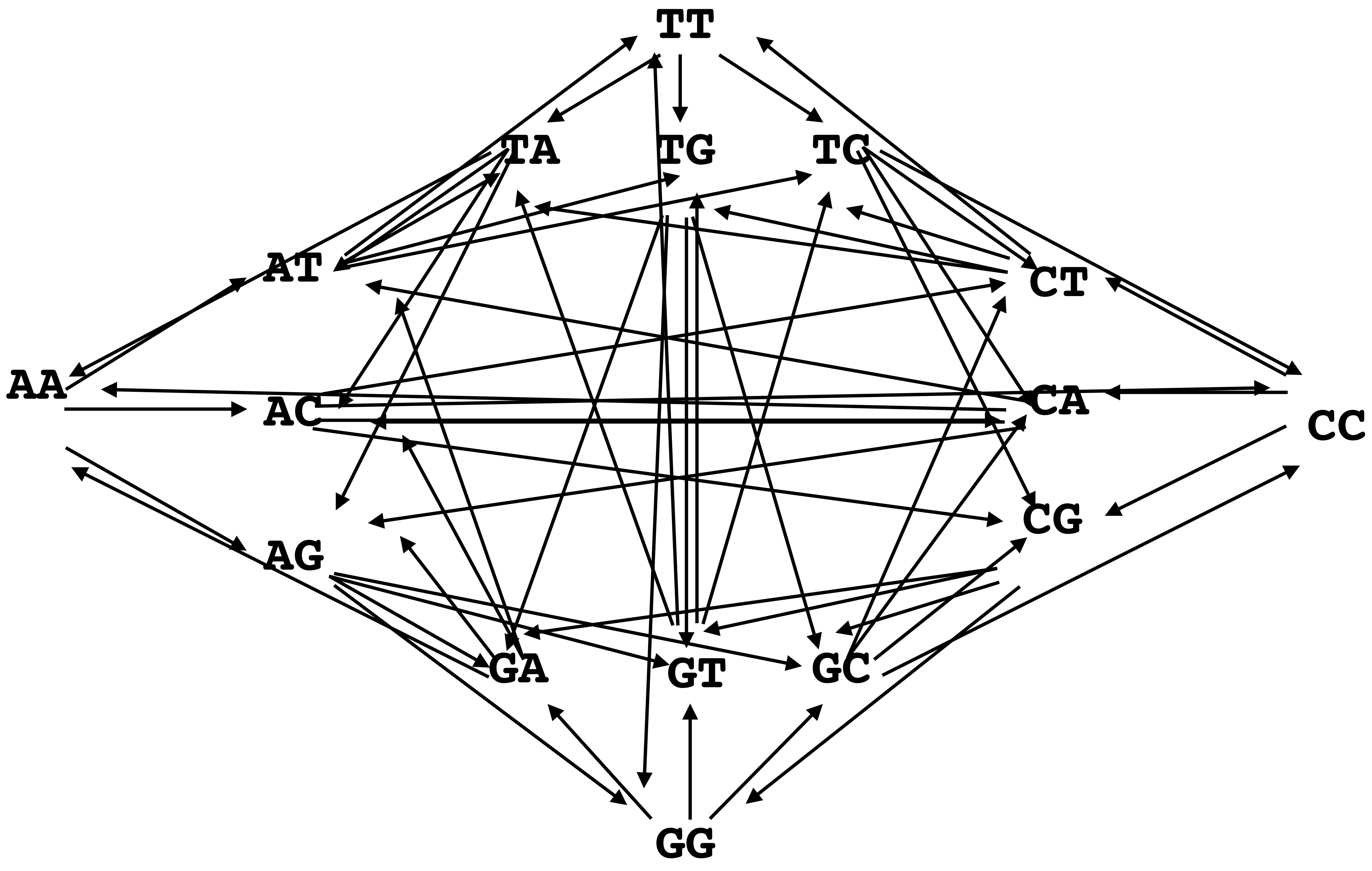
Can you find a length 6 binary sequence, which does not intersect one of the red *k*-mers?

Image courtesy of Guillaume Marçais

DBG for DNA

What we have seen in the previous slides was the DBG for $\Sigma = \{1,0\}$

For DNA ($\Sigma = \{\texttt{A,C,T,G}\}$) the graph is a little more complicated



Remember, the 4 DNA characters can be represented by 2-bit binary numbers.

Universal k-mer Set Extension

The naïve extension $U_{k,w} \cdot \Sigma$ of a universal set $U_{k,w}$ is universal

create $|\Sigma|$ new $(k+1)$-mers from each k-mer by concatenating each character from Σ to the end

Example:

ACCTG $\in U_{k,w} \longrightarrow$

$\{$**ACCTGA, ACCTGC, ACCTGT, ACCTGG**$\} \in U_{k,w} \cdot \Sigma$

Universal k-mer Set Extension

The naïve extension $U_{k,w} \cdot \Sigma$ of a universal set $U_{k,w}$ is universal

The sparsity of $U_{k,w} \cdot \Sigma$ is equal to that of $U_{k,w}$

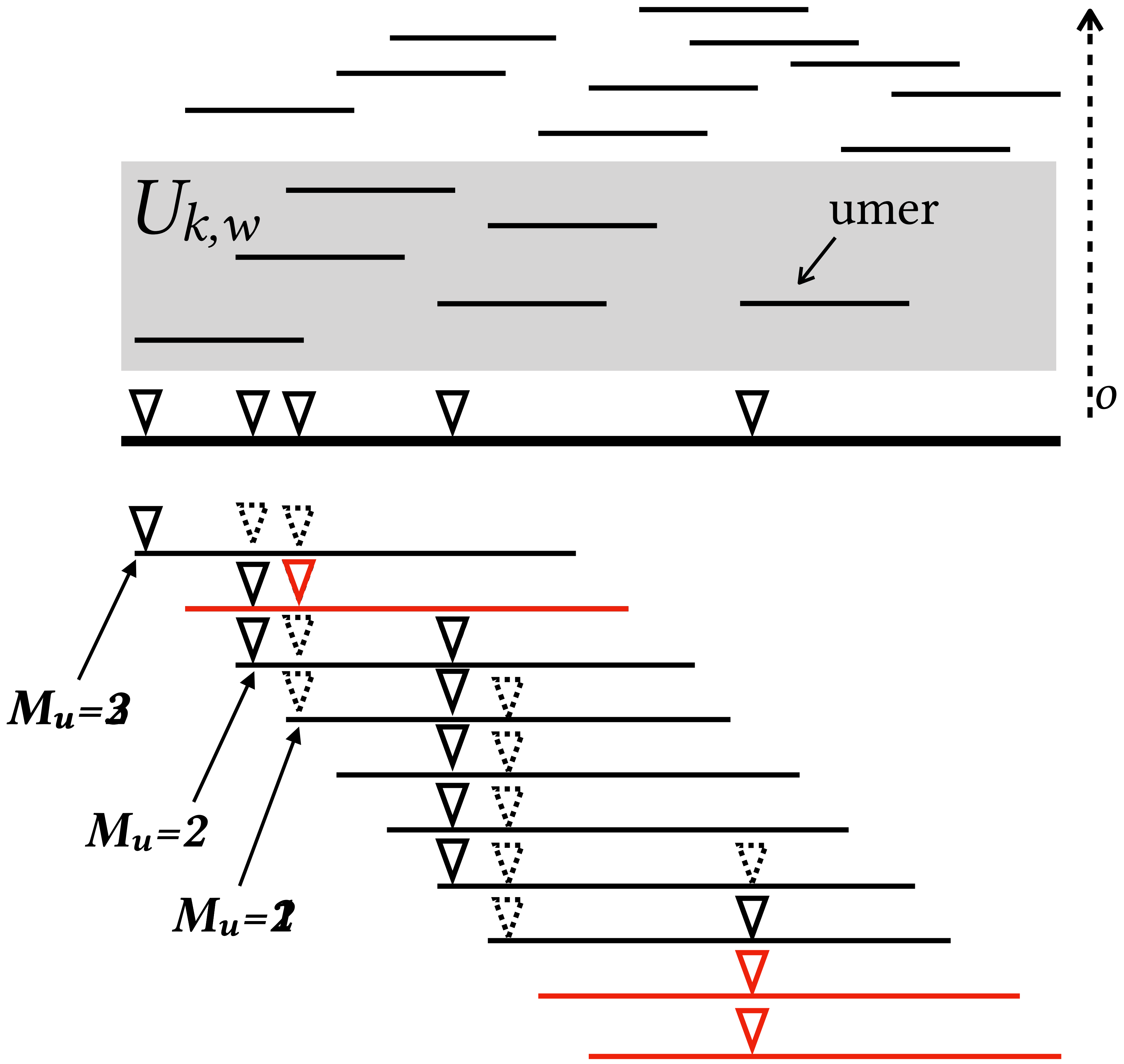
The density of a compatible order for $U_{k,w} \cdot \Sigma$ is less than or equal to
the density of a compatible order for $U_{k,w}$
if the orderings are compatible with each other

M_u and re$M_u$val

- the minimum co-occurance count for $u \in U$
  $M_u = \min_{\omega \in W_u} |\omega \cup U|$

- For any $u \in U$ such that $M_u > 1$, $U \backslash u$ is universal

- The universal set after the removal of u has:
  - smaller size, and
  - higher (possibly equal) sparsity



Optimal re$M_u$val

- Not all umers with $M_u{>}1$ can be removed from U,
- Integer linear programming (ILP) is used to find the minimum number of k-mers to retain
- The ILP is deceptively simple

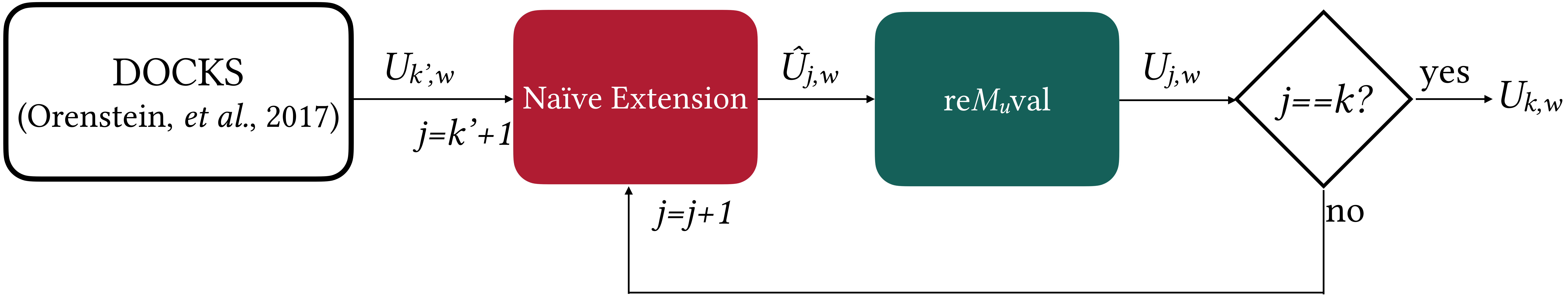
All of the umer co-occurance information is encoded in W

$$\text{minimize} \sum_{u \in U} y_u$$

$$\text{subject to} \sum_{u \,\in\, \omega \cap U} y_u \geq 1 \qquad \forall \omega \in W$$

$$y_u \in \{0,1\} \qquad \forall u \in U$$

Practical Universal k-mer Set Construction

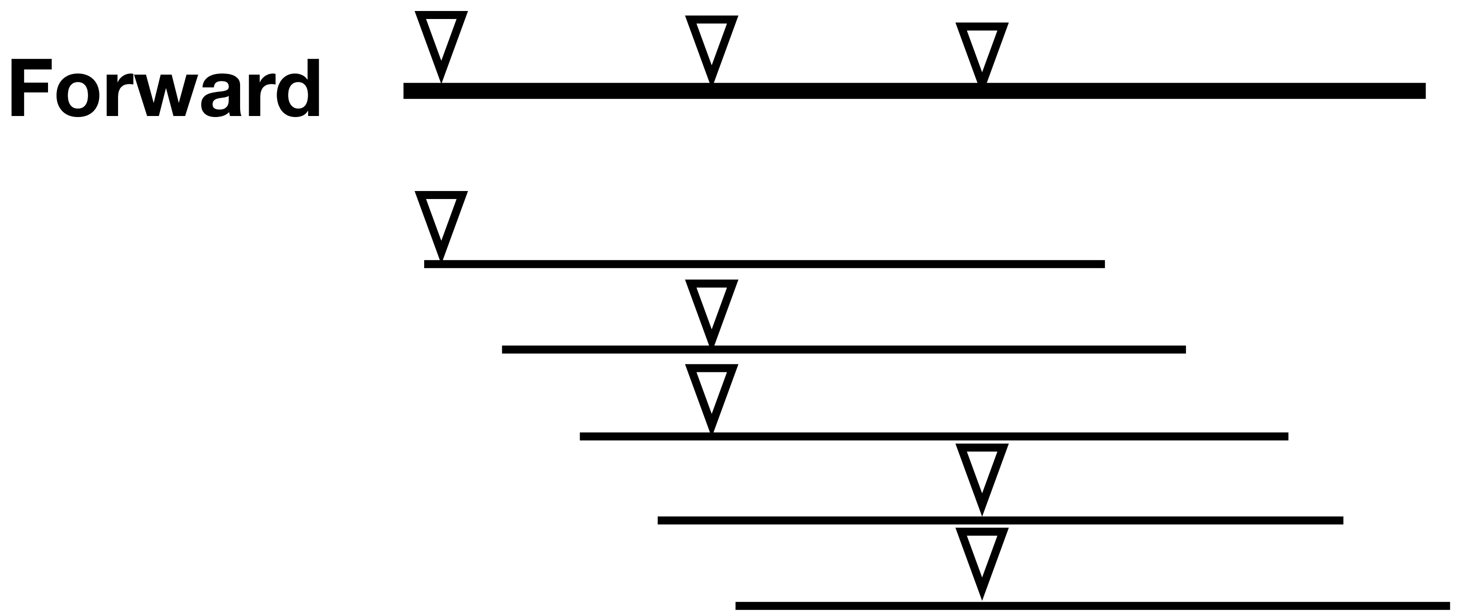


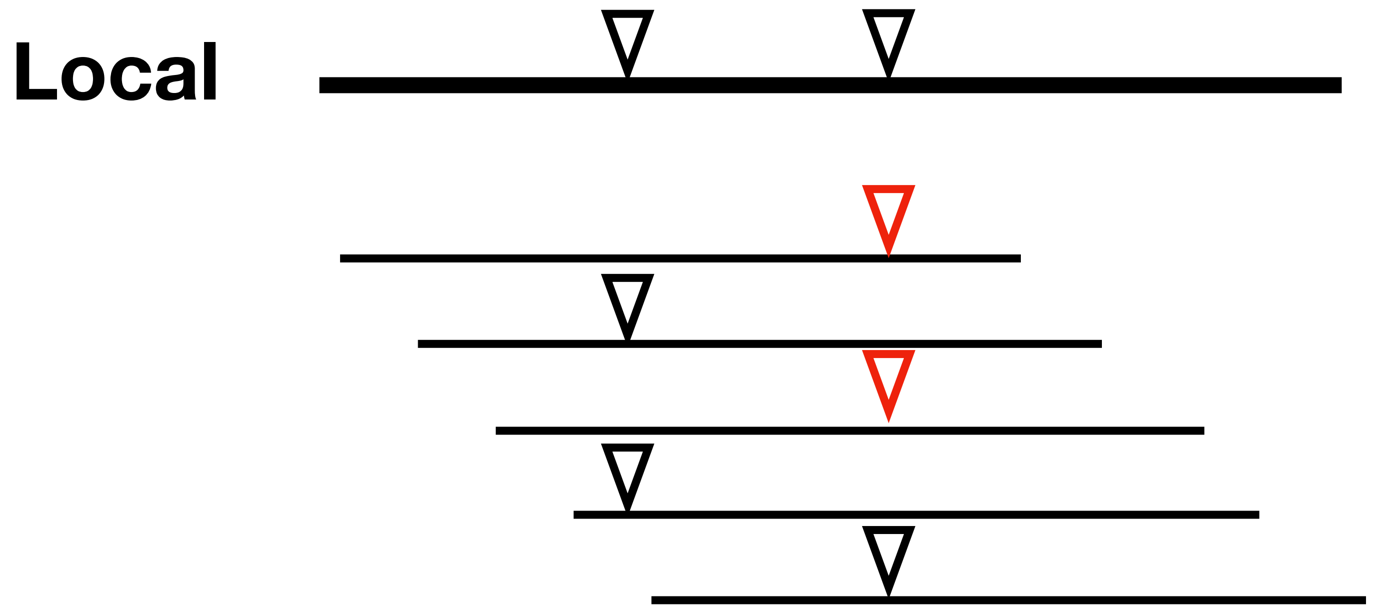
Local vs. Forward vs. Minimizer Schemes

Assume we're going to rewrite it *F(M)*→*m* where *M* is the ordered set of k-mers from the window, and *m* is the returned *k*-mer.

What if we relax the rules a bit:

- **Minimizer Schemes --** choose the $m = arg \min_{m' \in M} \big(O(m')\big)$
- **Forward Schemes** -- choose any *m* such that for all *M*' that can proceed *M* the choice is at the same position or later
- **Local Schemes** -- choose any *m*





Minimizer ⊂ Forward ⊂ Local

Problems with Jaccard