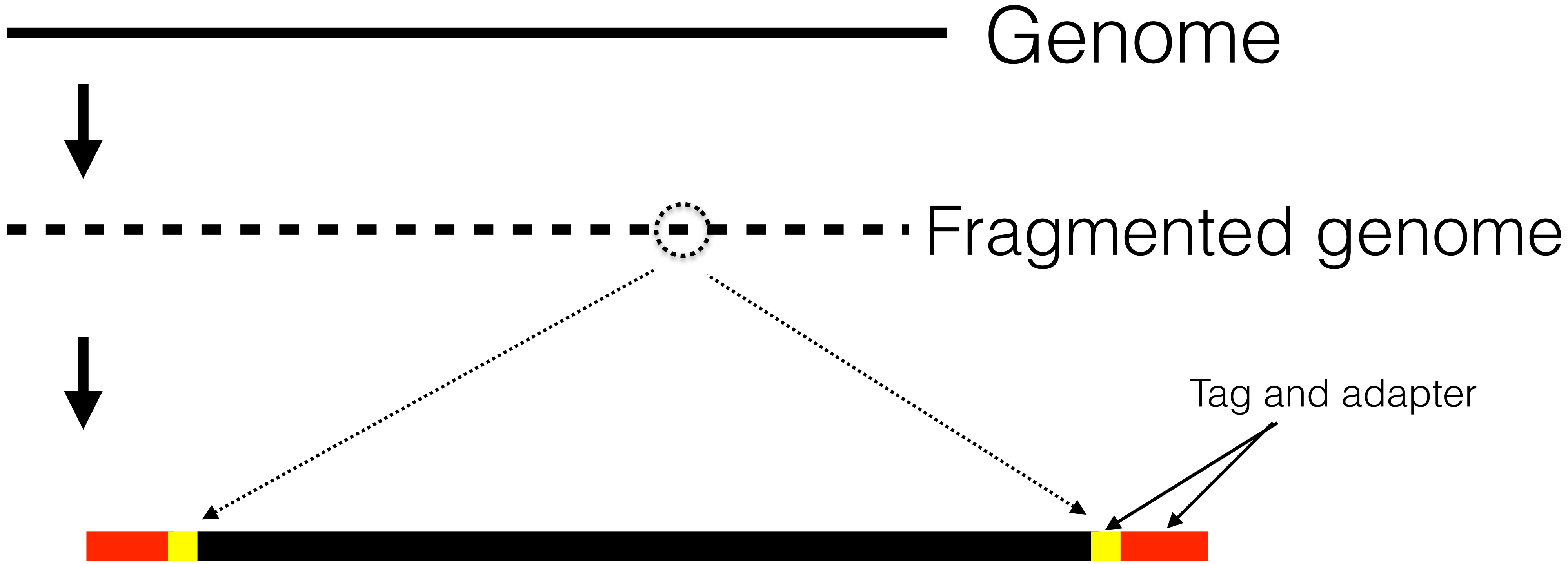
Sequence Assembly

Ben Kulohoma

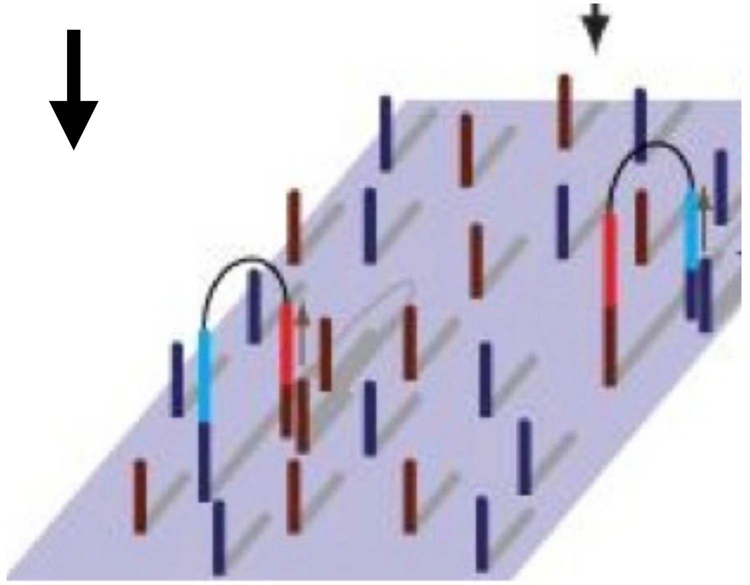
icipe

African Insect Science for Food and Health

Data generation steps



Create sequence libraries & tag fragments



Amplify fragments and identify the bases

Data format

# Remember that you fragmented your genome

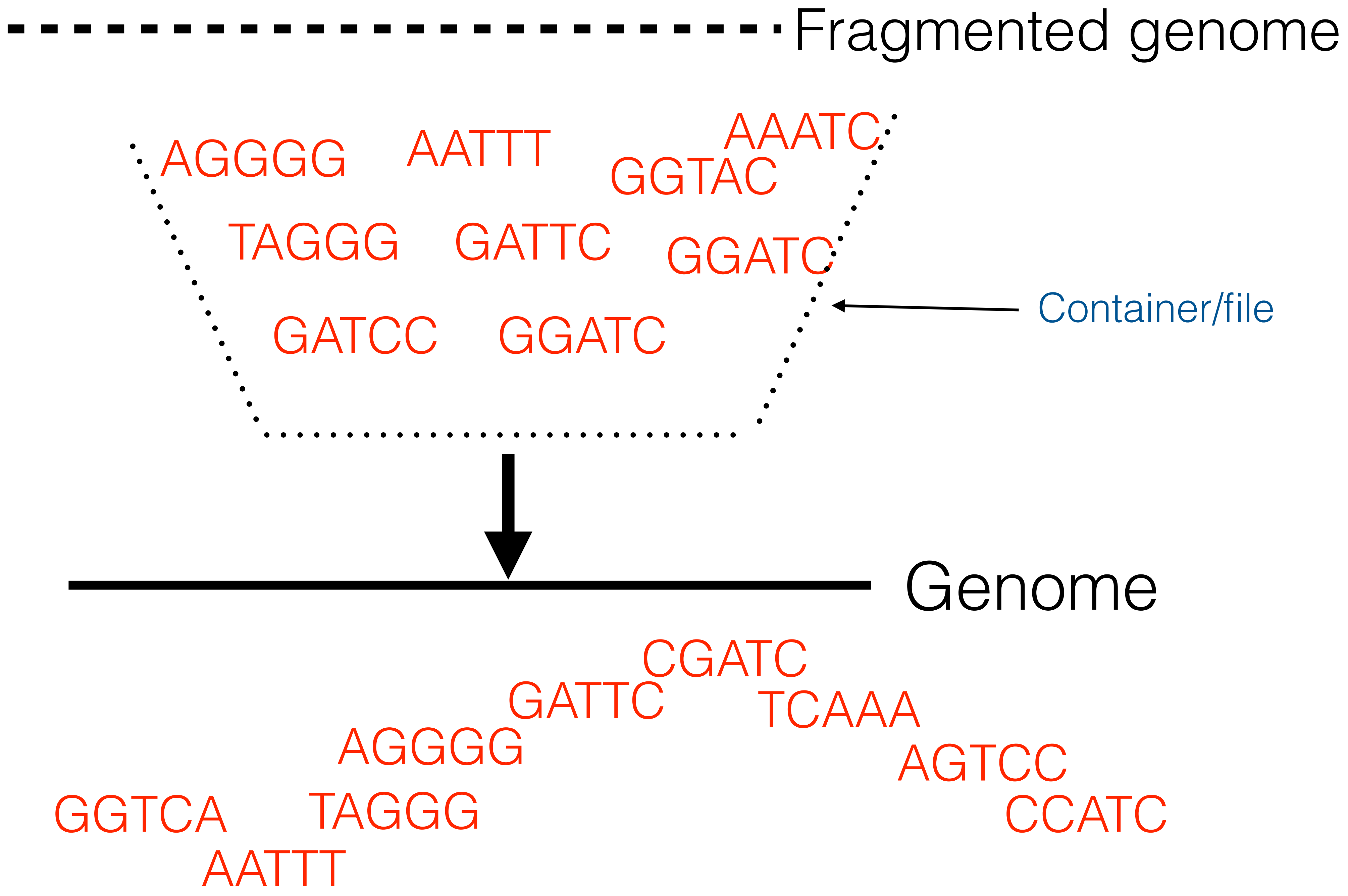
# So you end up with fragmented sequences of the genome (i.e. your NGS data)

# We shall only consider data in **fastq** format

# However, you may have data in a different format

# It is possible to convert between formats

How your data looks



Another example using a broken sentence

are having

We

fun

NGS

so much

t ICIPE

course a

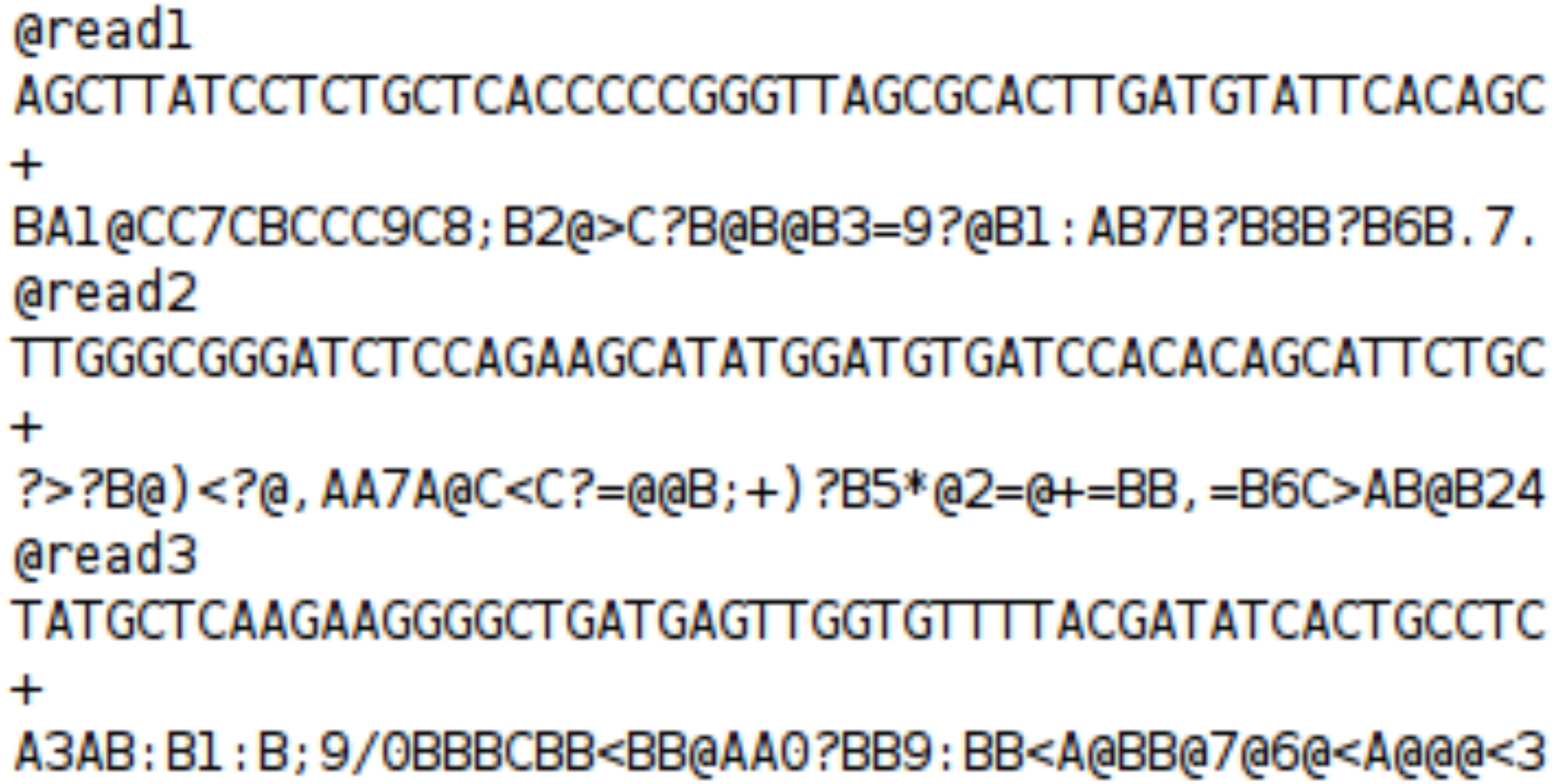
at the

$\downarrow$

“We are having so much fun at the NGS course at ICIPE”

# This is the principle of ***de novo*** assembly

How your data file actually looks



```
@read1
AGCTTATCCTCTGCTCACCCCCGGGTTAGCGCACTTGATGTATTCACAGC
+
BA1@CC7CBCCC9C8;B2@>C?B@B@B3=9?@B1:AB7B?B8B?B6B.7.
@read2
TTGGGCGGGATCTCCAGAAGCATATGGATGTGATCCACACAGCATTCTGC
+
?>?B@)<?@,AA7A@C<C?=@@B;+)?B5*@2=@+=BB,=B6C>AB@B24
@read3
TATGCTCAAGAAGGGGCTGATGAGTTGGTGTTTTACGATATCACTGCCTC
+
A3AB:B1:B;9/0BBBCBB<BB@AA0?BB9:BB<A@BB@7@6@<A@@@<3
```

Assembly methods/approaches…

# **OLC**: Overlap-Layout-Consensus assembly
# Hash tables
# **DBG**: De Bruijn Graph assembly

<u>1. OLC</u>

# They build the overlap graph

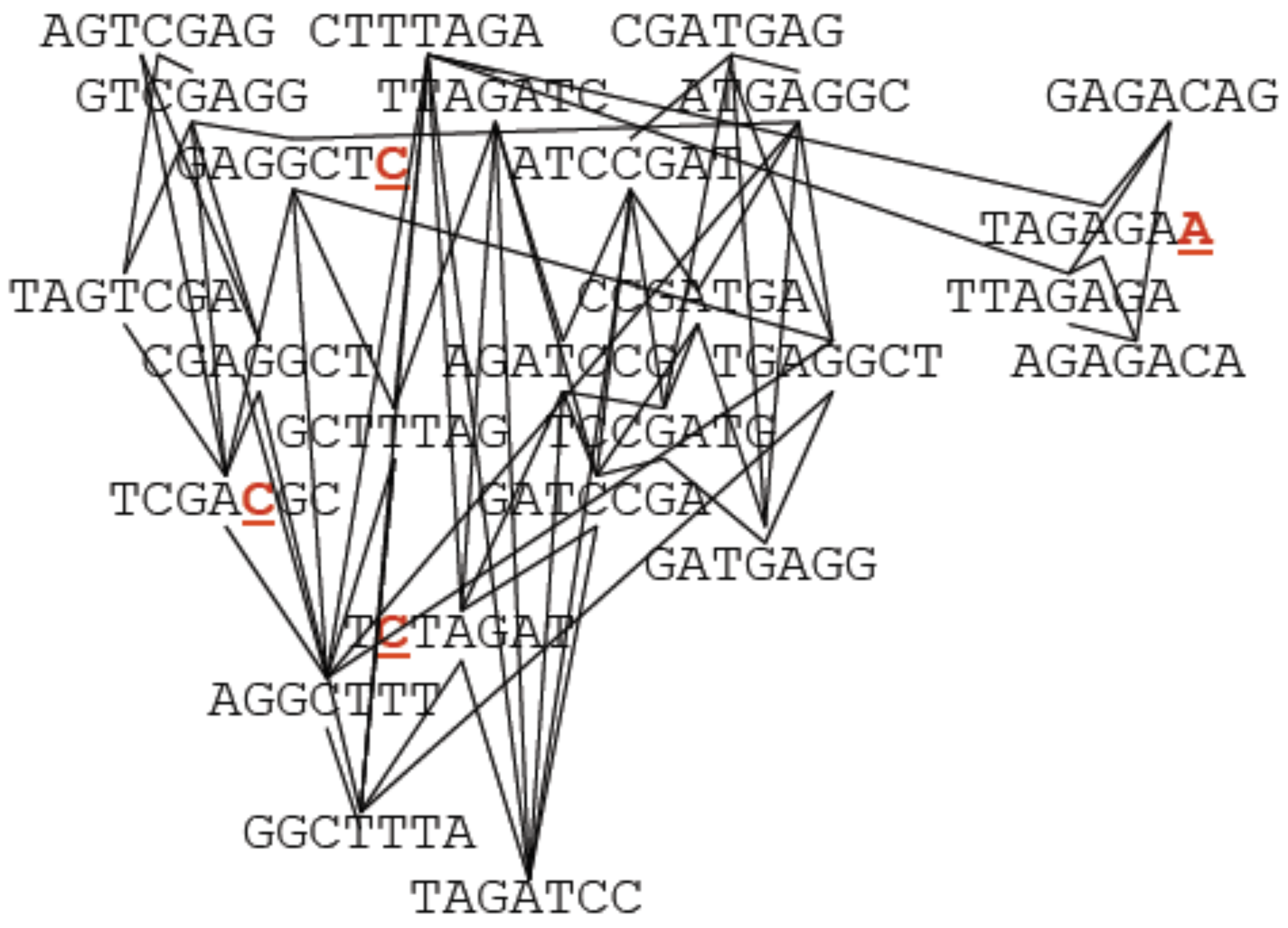
# Bundle stretches of the overlap graphs into contigs

# Pick the most likely nucleotide sequence for each contig

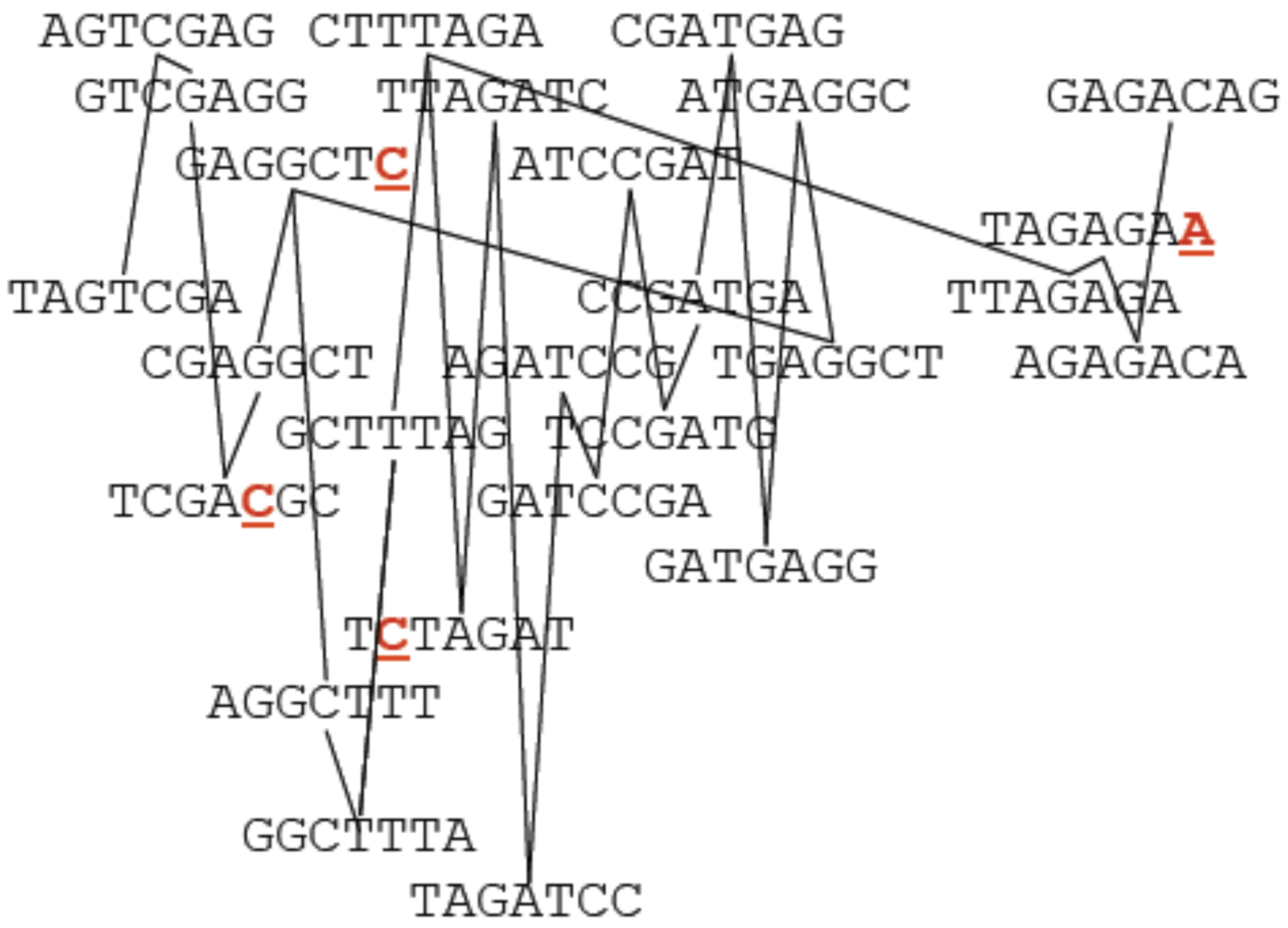
The OLC process…

```
 AGTCGAG CTTTAGA  CGATGAG CTTTAGA
  GTCGAGG  TTAGATC  ATGAGGC    GAGACAG
     GAGGCTC   ATCCGAT AGGCTTT GAGACAG
 AGTCGAG    TAGATCC ATGAGGC  TAGAGAA
TAGTCGA  CTTTAGA CCGATGA    TTAGAGA
    CGAGGCT  AGATCCG TGAGGCT  AGAGACA
TAGTCGA GCTTTAG TCCGATG  GCTCTAG
   TCGACGC    GATCCGA GAGGCTT AGAGACA
TAGTCGA    TTAGATC GATGAGG TTTAGAG
  GTCGAGG TCTAGAT   ATGAGGC  TAGAGAC
      AGGCTTT  ATCCGAT AGGCTTT GAGACAG
 AGTCGAG   TTAGATT  ATGAGGC   AGAGACA
       GGCTTTA  TCCGATG    TTTAGAG
    CGAGGCT TAGATCC  TGAGGCT   GAGACAG
 AGTCGAG  TTTAGATC  ATGAGGC TTAGAGA
     GAGGCTT  GATCCGA GAGGCTT  GAGACAG
```

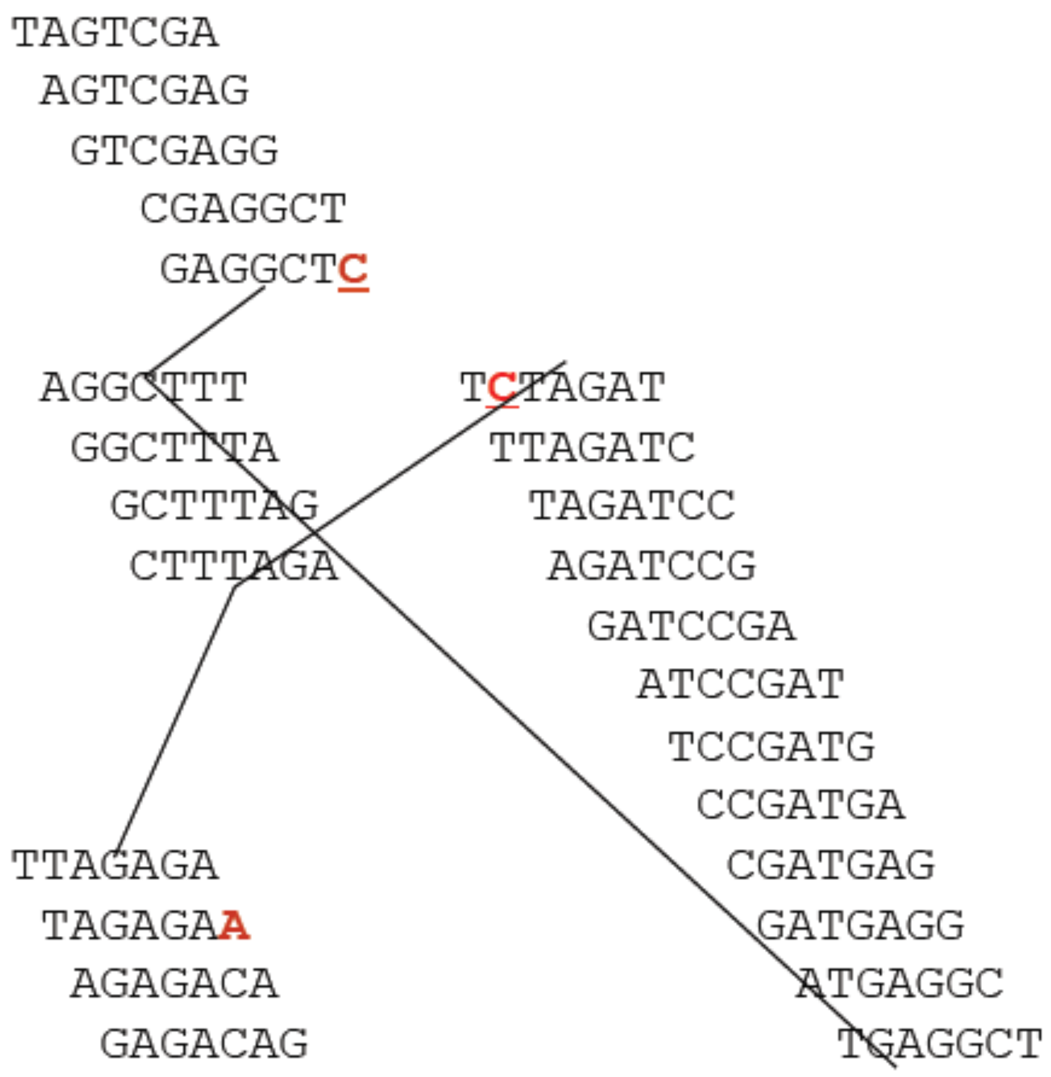
Get all possible OLC graphs…



Refine the OLC graphs…



Get the final OLC graph that best supports the data…



e.g. TTAGAGSCAGATCTAGATCCGAT…..

OLC summary…

# Used in traditional assemblers
(e.g. Phrap, Arachne, Celera…)

# and some short read assemblers
(e.g. Edena)

# Generally more expensive computationally
(i.e. require pairwise global alignments)

# However, as reads get longer (>300bp?) they produce better results through finer alignments
(SGA [String graph assembler] and Fermi)

2. Hash tables

# Transitional assemblers (~2007)
(SSAKE, VCAKE, SHARCGS…)

# New ones are appearing…
(e.g. Pride, Monument, Ray…)

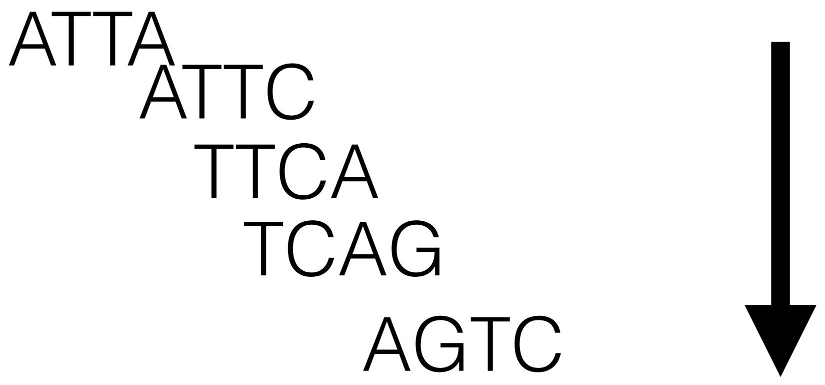
# New ones are much faster and leaner

# Hash table methods are generally less robust to variation and noise (especially the older ones)

# They are essentially simplified de Bruijn graph assemblers

The Hash Table process…

TCCAT
ATTCC
AAGGG
GGAAT
GCGTA
CGTTC
⋮
⋮
⋮



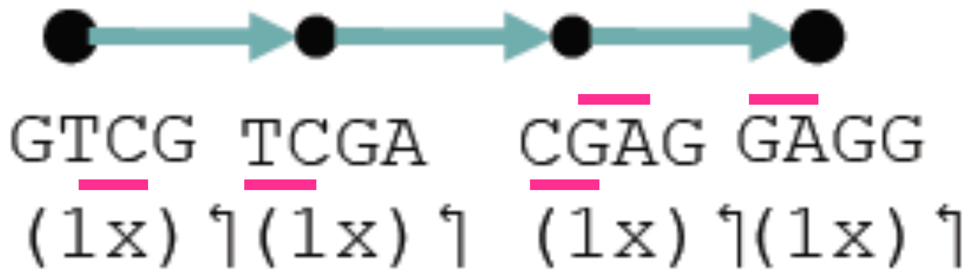
your sequence ATTATTCAGTC….

2. De Bruijn graphs

# Requires that you deal with your sequence reads in small sizes referred to as k-mer

# These k-mers are represented by a unique node on the graph

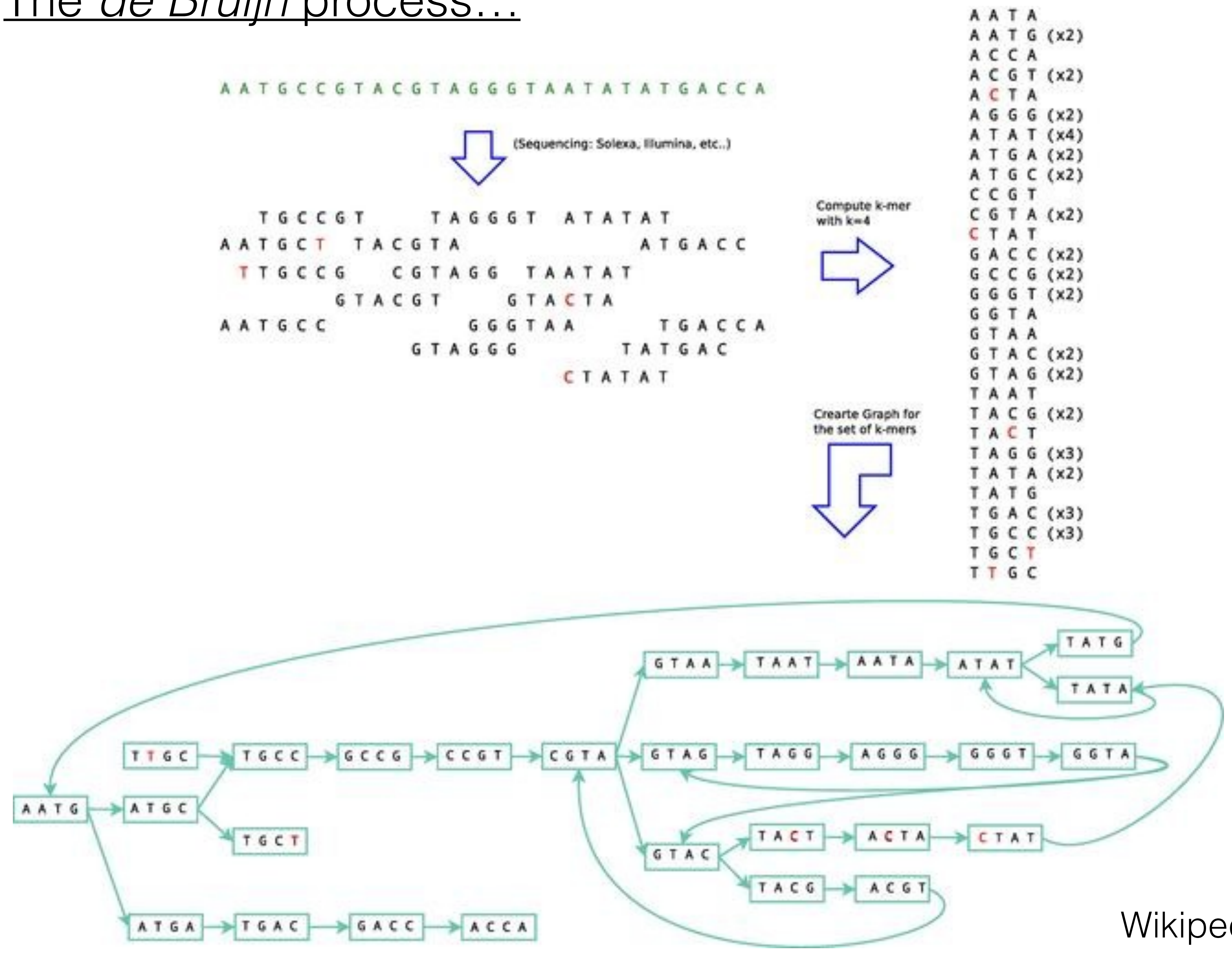
# in the example below the k-mer size = 4



# Two nodes are connected if the k-mers have an overlap

(i.e a connection between nodes is made if characters between nodes match!)

The *de Bruijn* process…



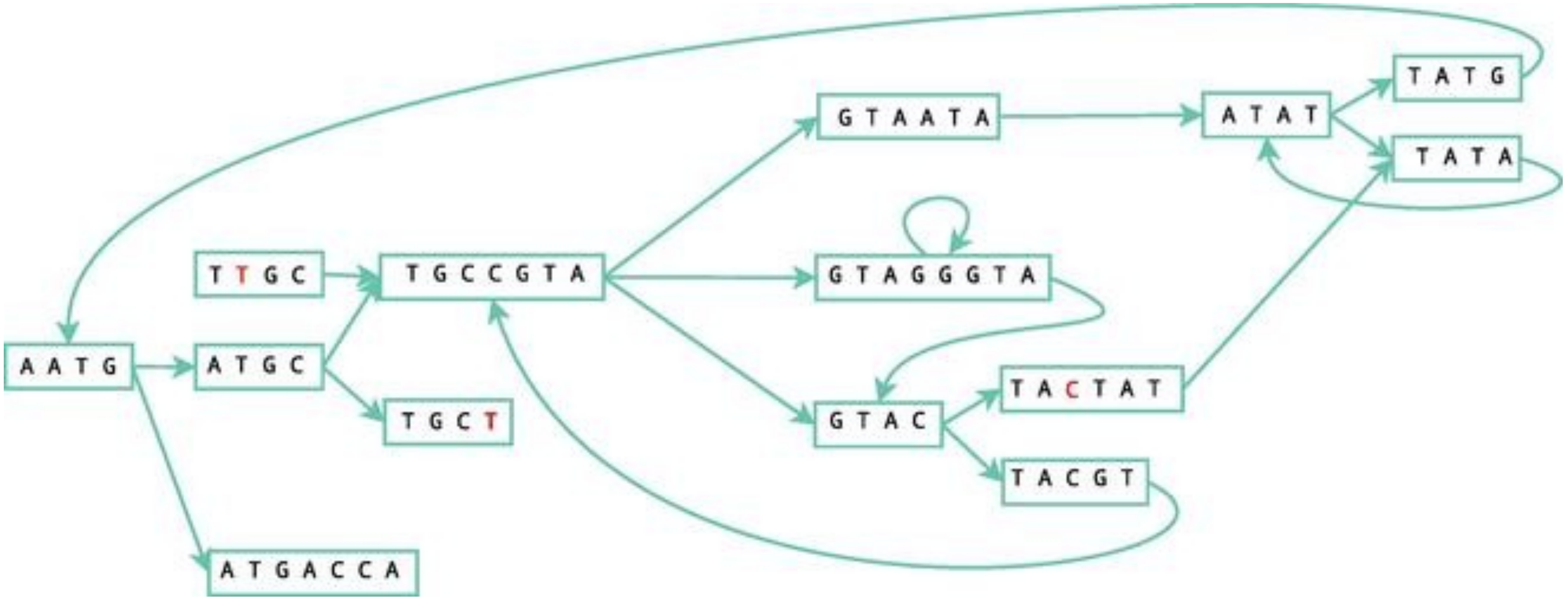
```
AATGCCGTACGTAGGGTAATATATGACCA

  TGCCGT   TAGGGT ATATAT
AATGCT TACGTA         ATGACC
 TTGCCG  CGTAGG TAATAT
      GTACGT   GTACTA
AATGCC       GGGTAA    TGACCA
          GTAGGG     TATGAC
                  CTATAT
```

AATA	
AATG	(x2)
ACCA	
ACGT	(x2)
ACTA	
AGGG	(x2)
ATAT	(x4)
ATGA	(x2)
ATGC	(x2)
CCGT	
CGTA	(x2)
CTAT	
GACC	(x2)
GCCG	(x2)
GGGT	(x2)
GGTA	
GTAA	
GTAC	(x2)
GTAG	(x2)
TAAT	
TACG	(x2)
TACT	
TAGG	(x3)
TATA	(x2)
TATG	
TGAC	(x3)
TGCC	(x3)
TGCT	
TTGC	

Wikipedia

# Memory requirements can be reduced by merging “undisputed” paths on the graph

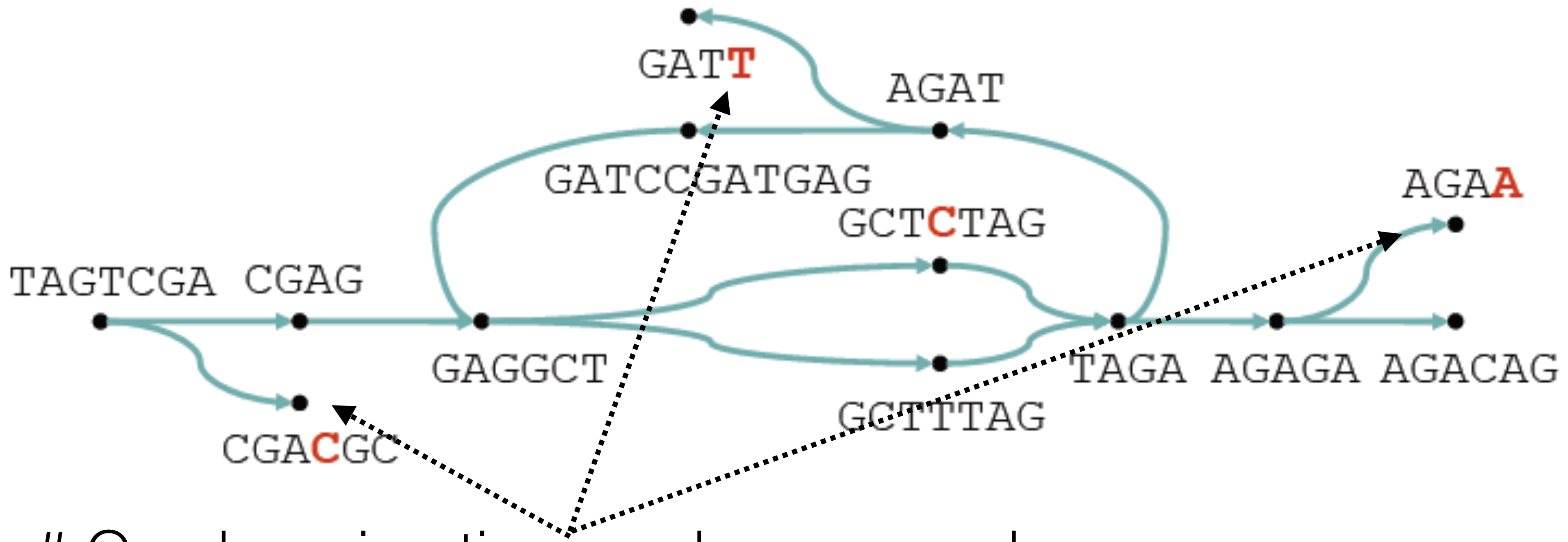


Wikipedia

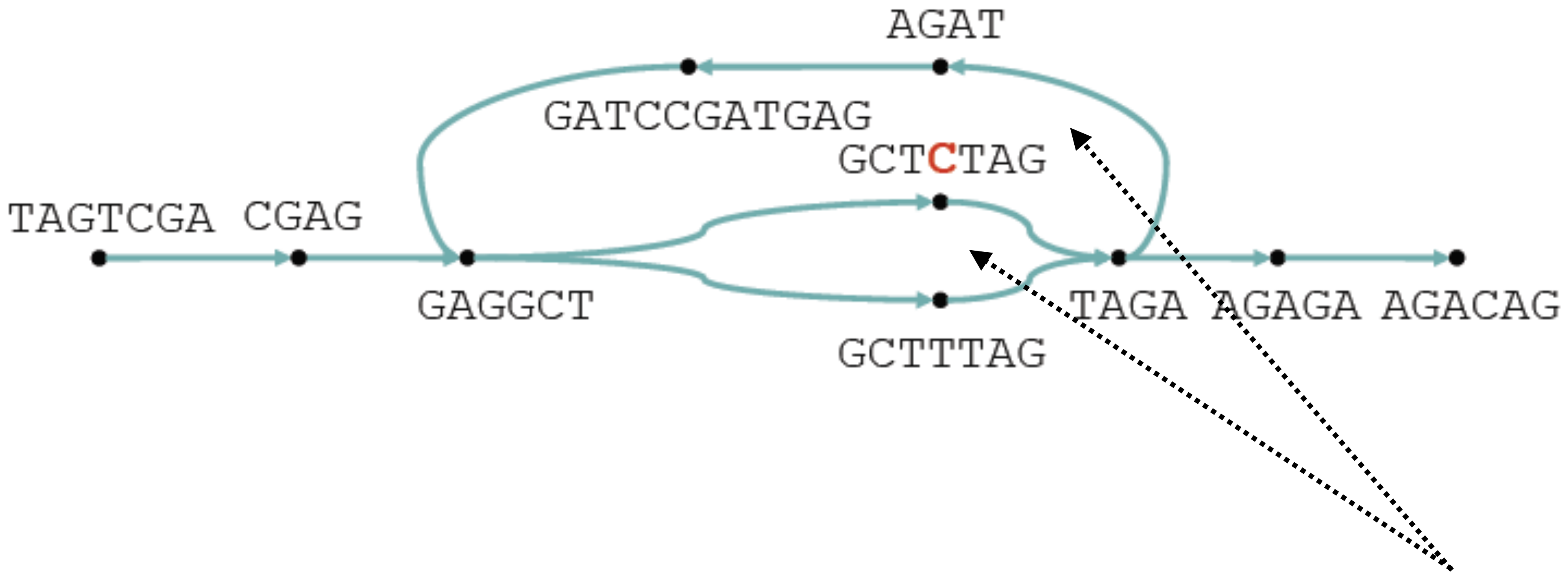
# The assembly can be further improved by removing:

- over-hanging tips
- bubbles
- unlikely/poor node connections

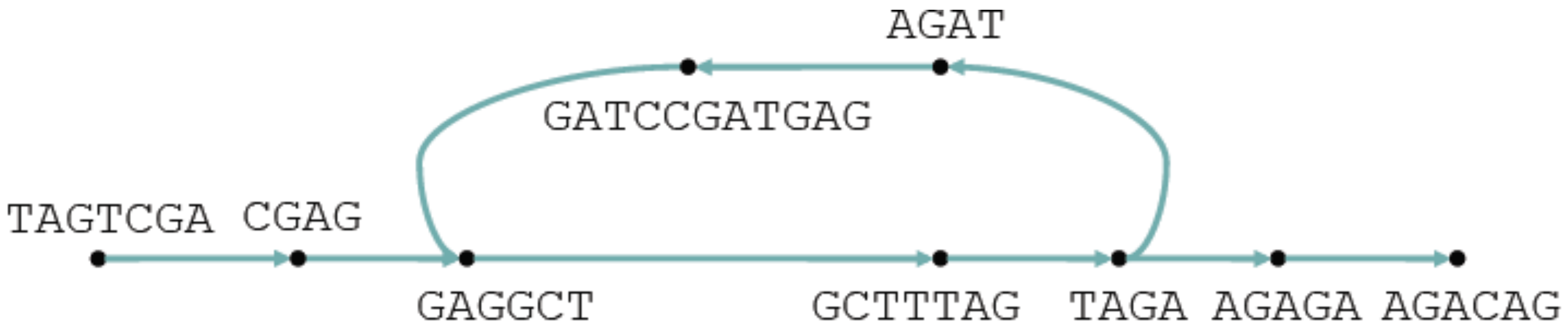
# The graph is then refined to find the optimal path that gives the most likely assembly



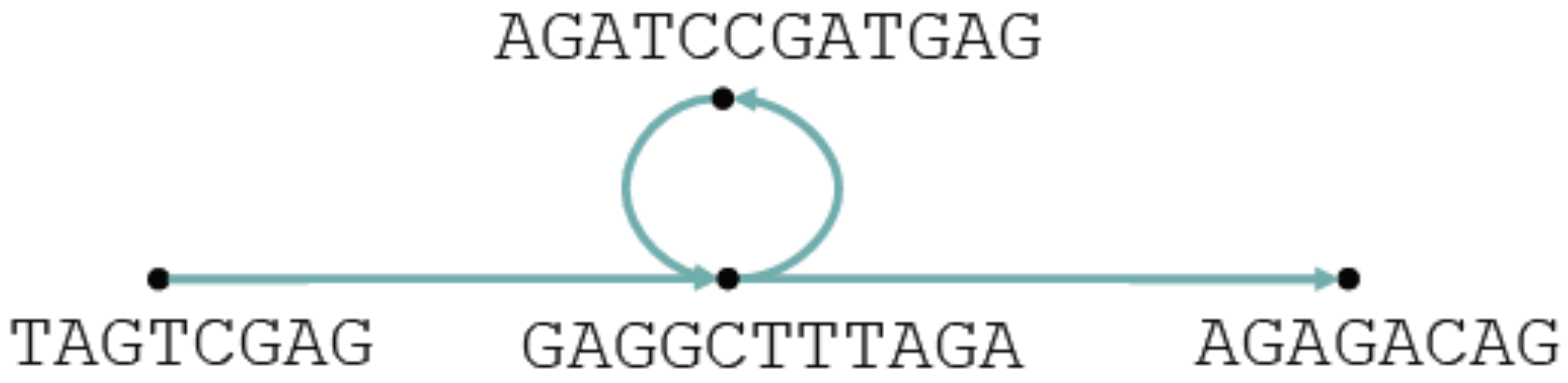
# Overhanging tips can be removed



# Once the tips are removed now we have to remove <u>bubbles</u> and find the most optimal path



# So you end up with this…



# The de Bruijn graph tool that you are using will select the most optimal path for you

# The user only provides a few parameters e.g. the coverage cut-off etc…

# Currently the most common assemblers for NGS are:

Euler, Allpaths, Velvet, ABySS, SOAPdenovo

# They are essentially a compromise between the other two methods (k-mer hash tables and OLC)

# Graph construction and correction is essentially common in all these assemblers - with a few implementation differences

# A problem that’s been discussed during this course is how to handle tricky regions in NGS sequence data (e.g. repeats)

Different approaches to repeat resolution

# Euler: finds all possible paths for each pair of reads; and gives the “best”

# ABySS: same as above - but reads are bundled into node to node connections to reduce calculations

# AllPATHS: same as above - but search is limited to localised clouds around pre-computed scaffolds

# Velvet: uses the variance between read pairs (on a common contig).

Choosing assemblers

# How big is your genome?

# How repetitive?

- Short repeats?
- Long repeats?
- Known repeats?

# Most assemblers are fine tuned for a specific task:

- Big mammalian genomes: ALLPATHS, SOAPdenovo, SGA, ABySS
- Small genomes: Velvet
- Single cell assembly: SPAdes
- Transcripts: Trinity, Oases, TrnasABySS, SOAPtrans

Public benchmarks

# assemblathon.org

# gage.cbcb.umd.edu

# Setting target coverage will help make things clearer (50 -100X is a good target)

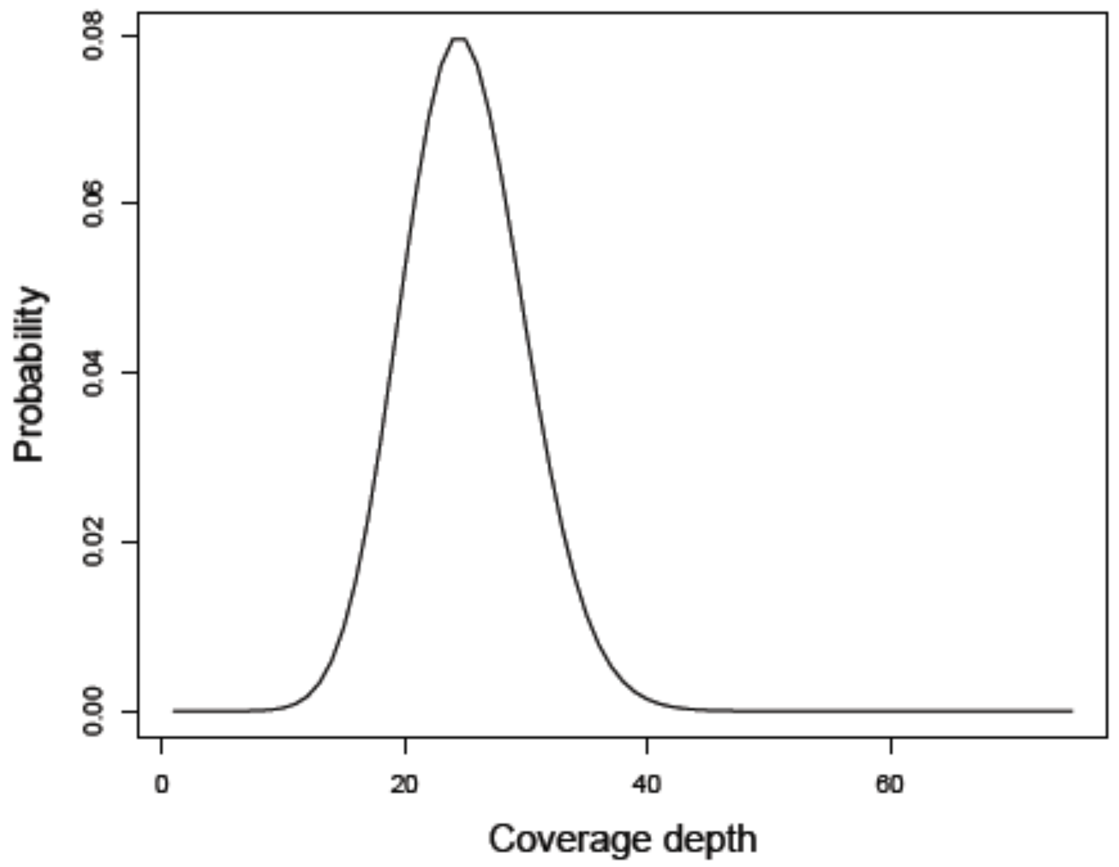
Choosing the computer hardware

# Memory (single machine or distributed?)

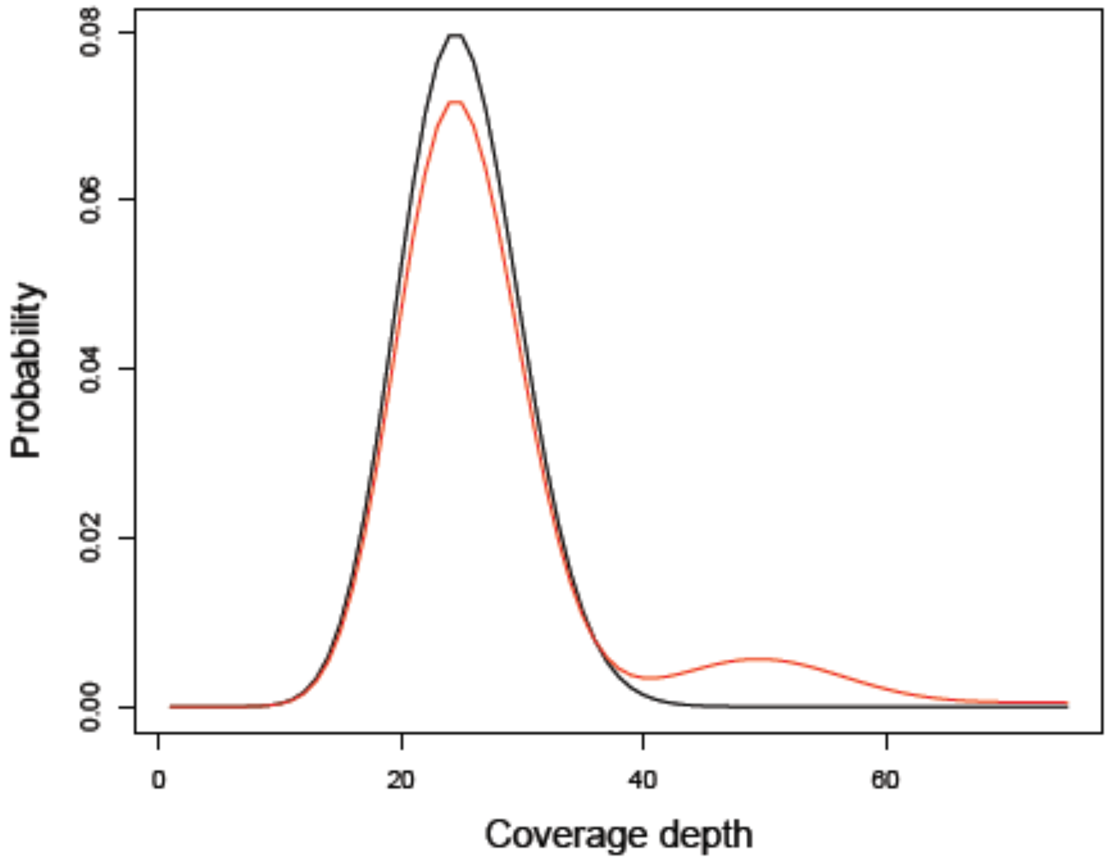
# CPU power

# Rent cloud CPUs

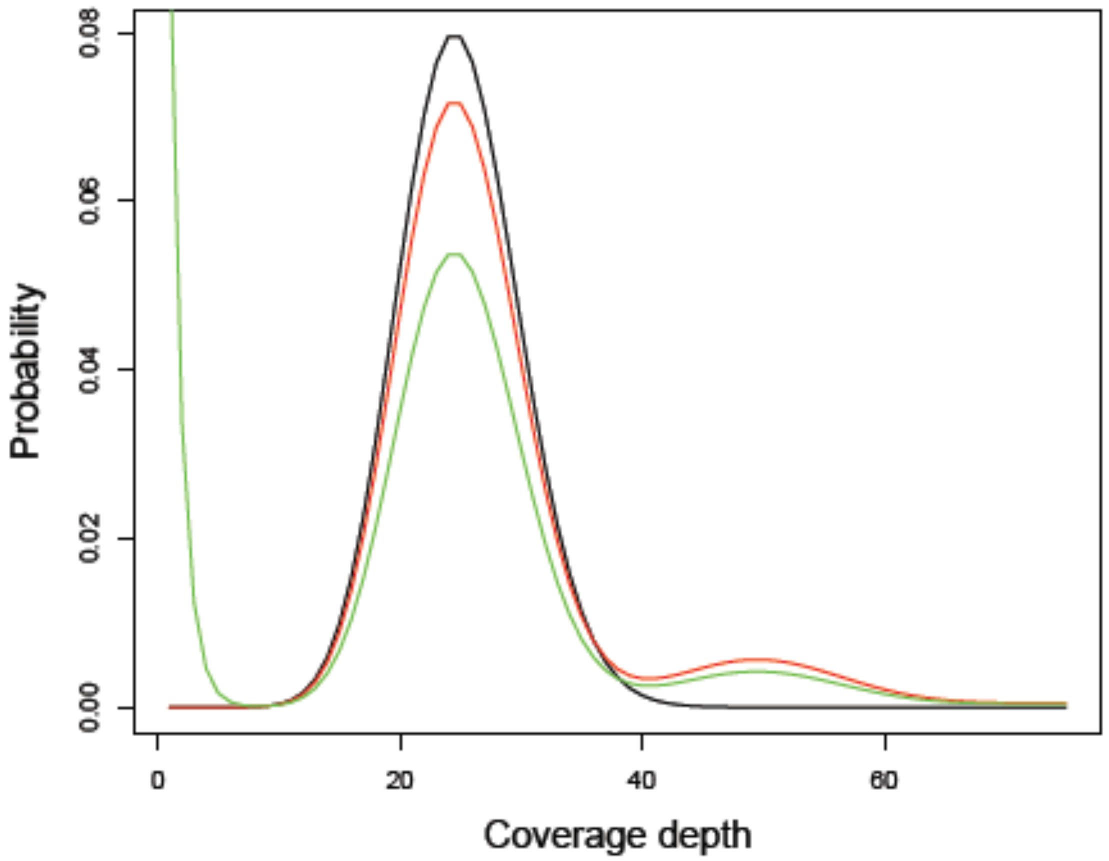
Coverage: in a perfect world



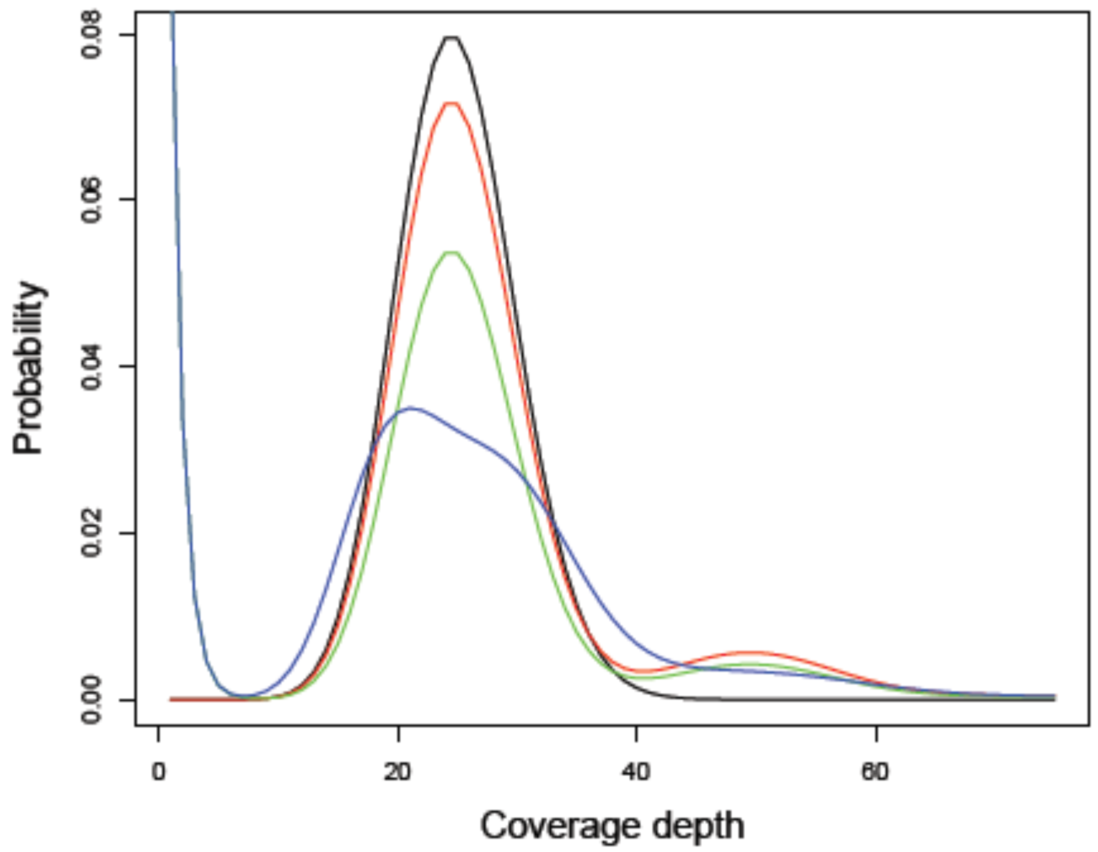
Coverage: handling repeats



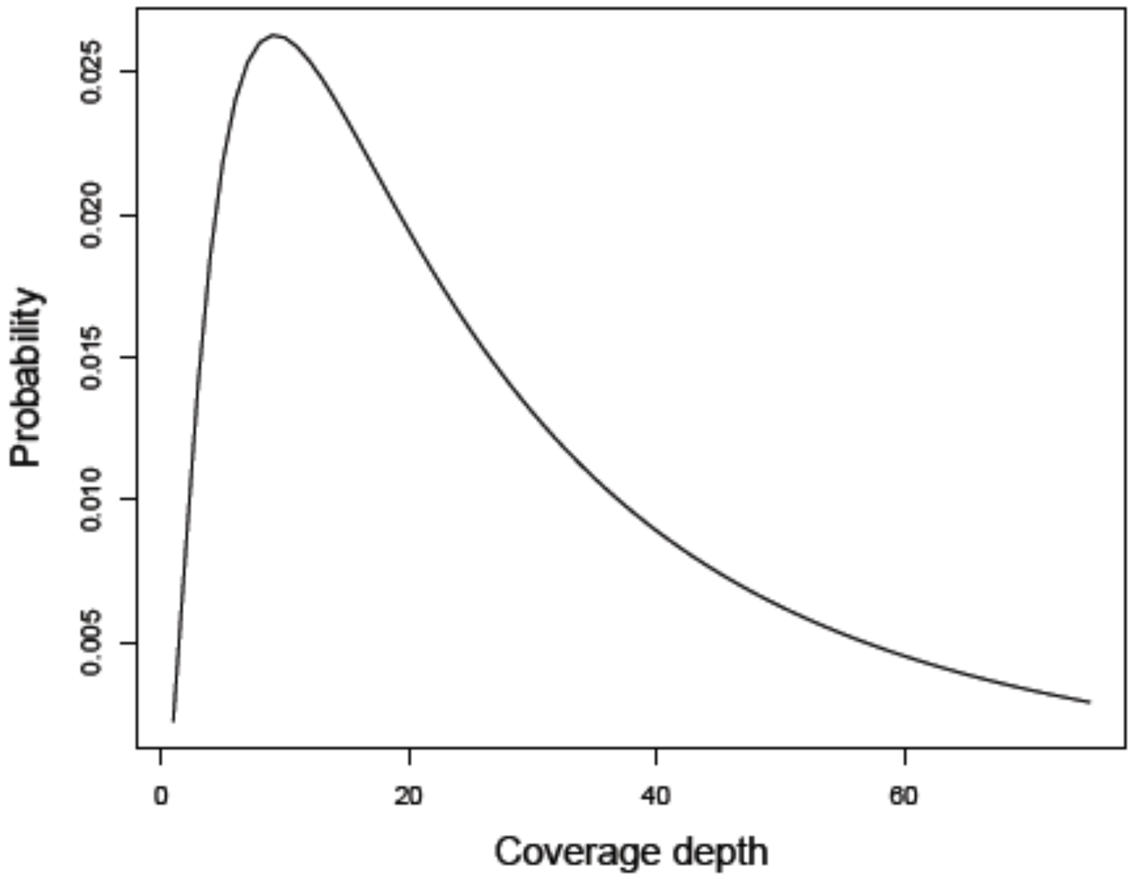
Coverage: with sequencing errors



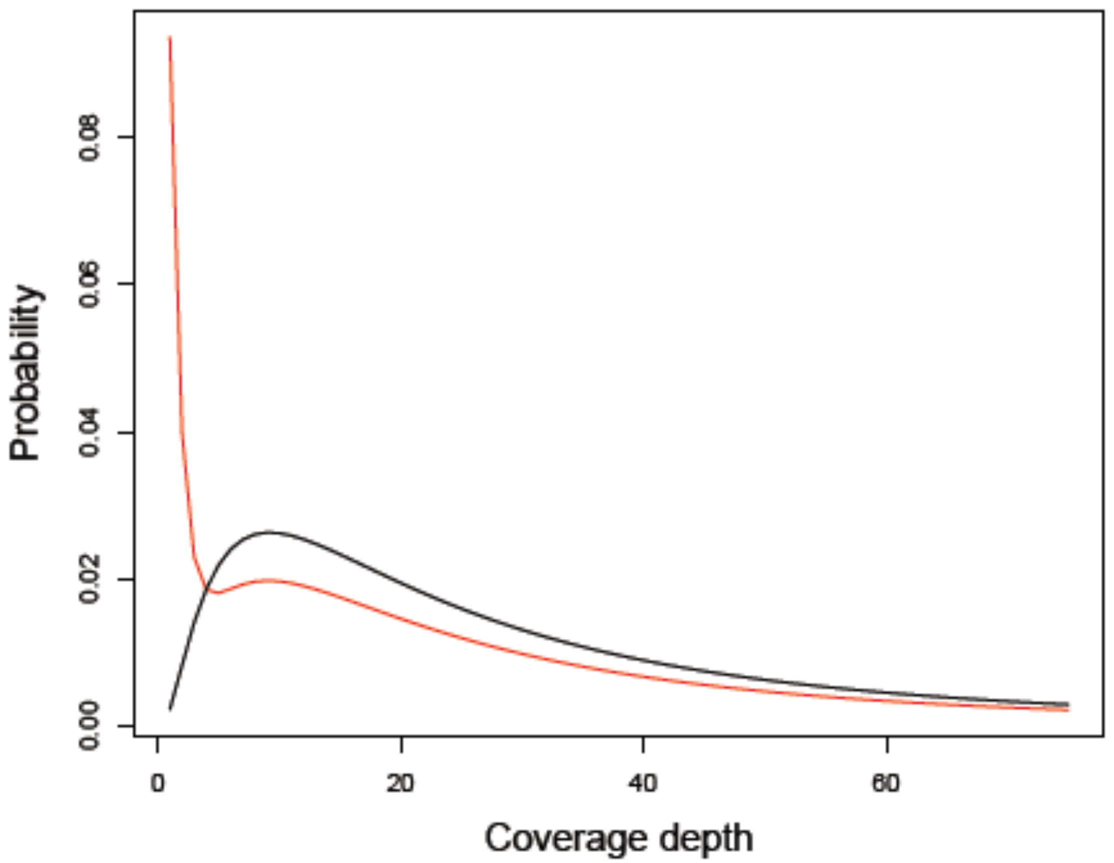
Coverage: with GC-bias



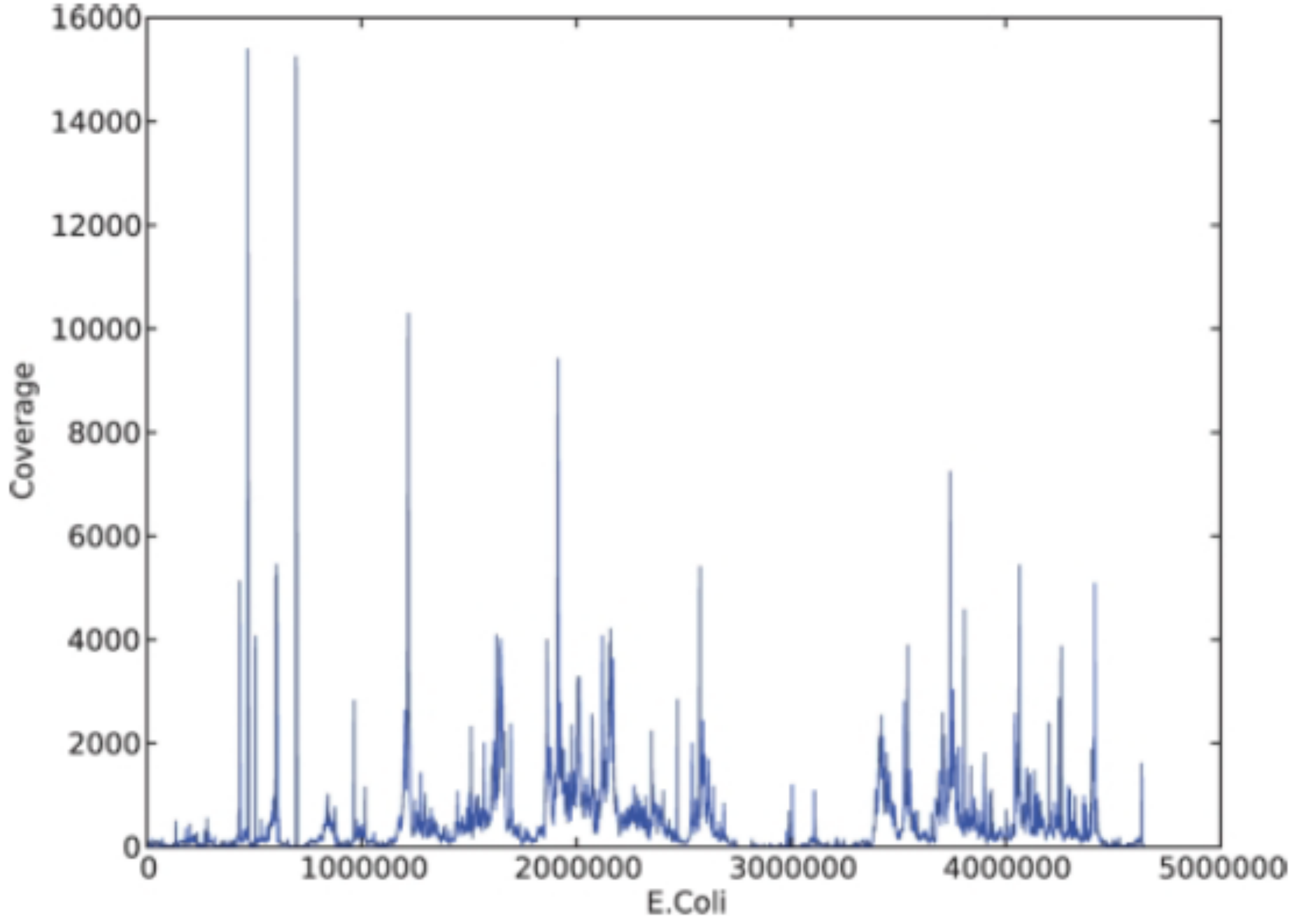
Coverage: RNAseq/ Metagenomics



Coverage: .. with errors



Coverage: Single cell sequencing

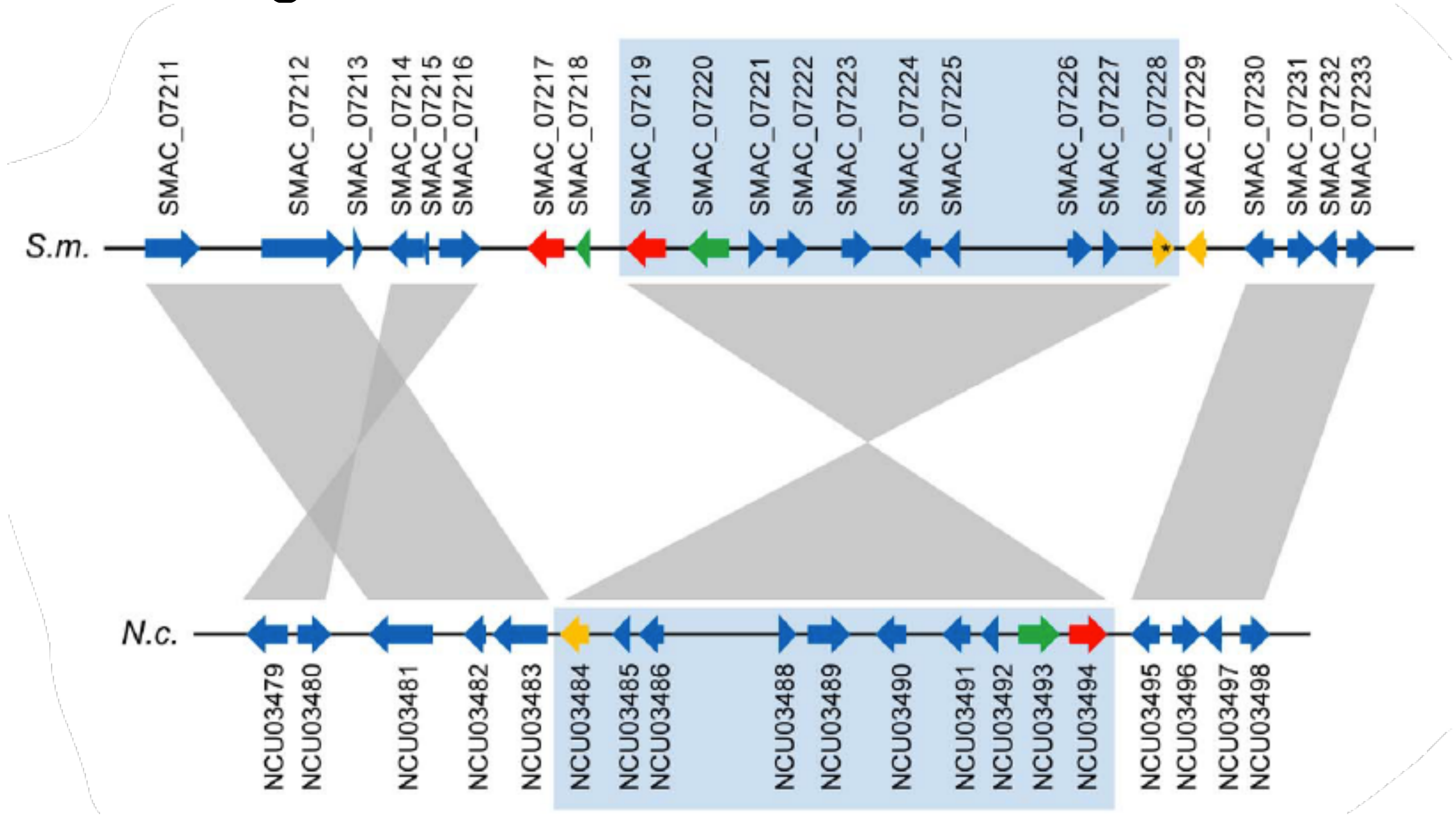


Medvedev *et al.*, *Bioinf.* 2011.

So now you have yoiur assembly…(yeeey!!!)

# so what can we do next…?

# We can order the contigs using as closely related reference genome



After doing that we can then do

a. ab-initio gene prediction - e.g using glimmer3

c. transfer annotations from closely related reference genomes - e.g. using RATT

UNIVERSITY OF CAMBRIDGE

CAMBRIDGE AFRICA

CERNVec
Community of Excellence for Research in Neglected Vector Borne and Zoonotic Diseases

THRIVE
TRAINING HEALTH RESEARCHERS INTO VOCATIONAL EXCELLENCE IN EAST AFRICA

inqaba biotec ™

kentros[biosciences]® UNLIMITED

STEVENAGE BIOSCIENCE CATALYST
Open Innovation Campus

Healx^3

TGAC
The Genome Analysis Centre™

Thanks for listening!

Any questions!