Genome Assembly
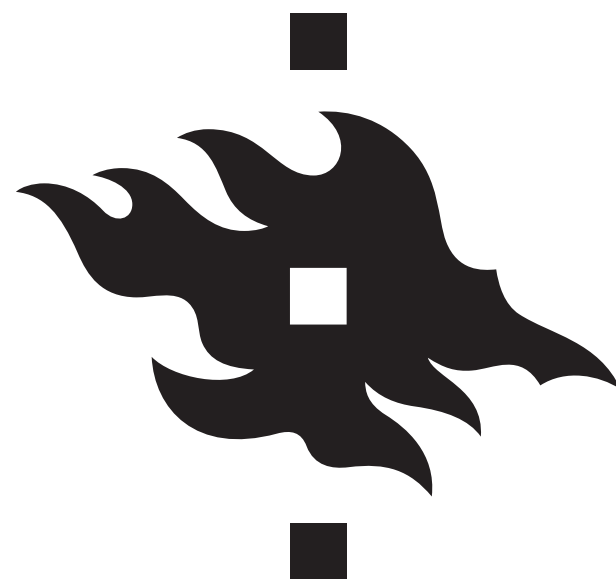
From theory to practice (and back)

UNIVERSITY OF HELSINKI

Alexandru Tomescu
Department of Computer Science

Outline

1. The problem

2. Practical issues

3. Theoretical problem formulations

4. Practical genome assembly
   - Contig assembly
   - Scaffolding
   - Gap filling

5. And back: a more "practical" theoretical formulation

LECTURE
Theory + Abstract view

⬇

ASSIGNMENT
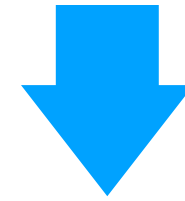**Practical + Hands-on view**

The problem
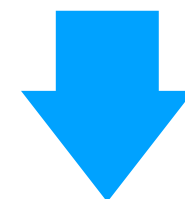
(A general description of the input and output)

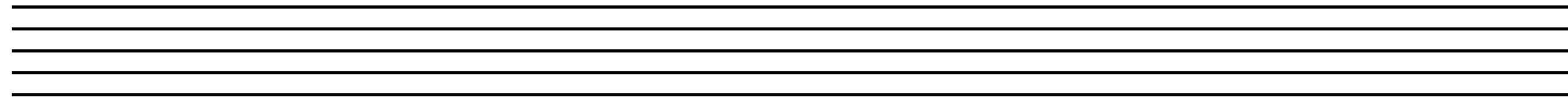# Short-read sequencing

(Third-generation sequencing)

DNA

⬇ **Amplification**

⬇ **Fragmentation**
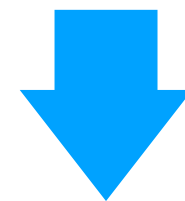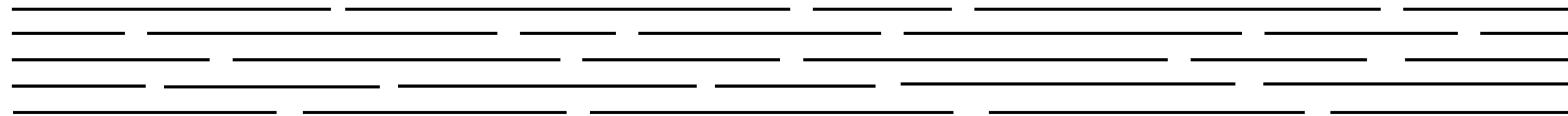
⬇ **Size selection**
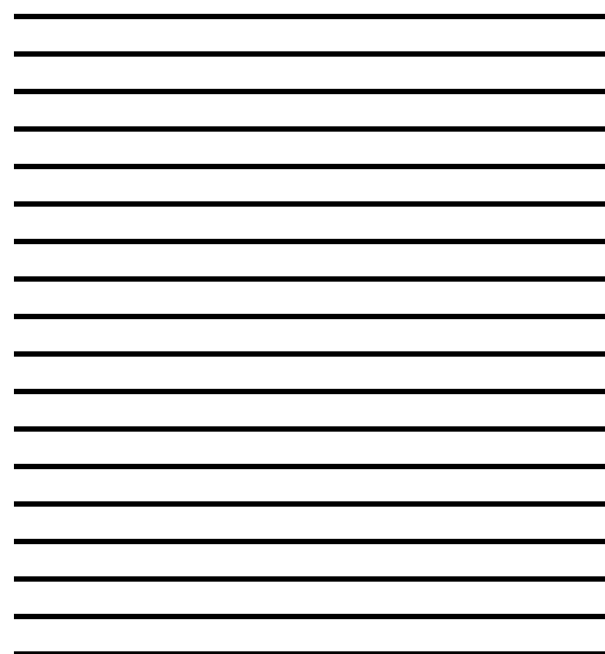
Sequencing ➡

length ~450
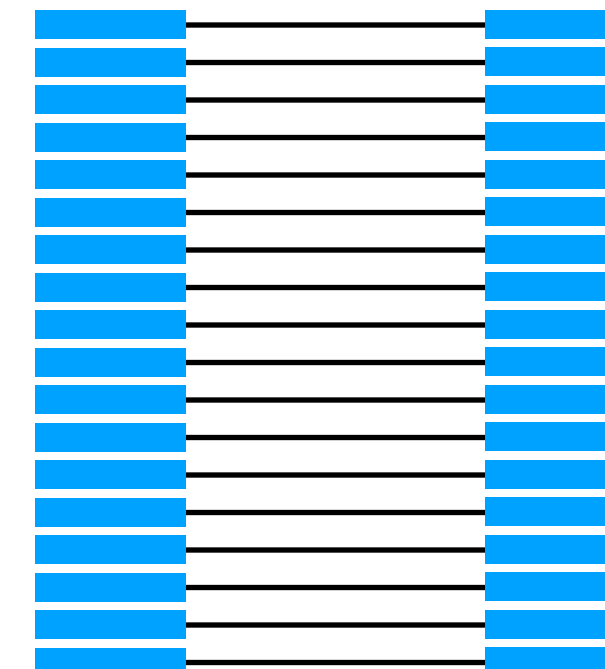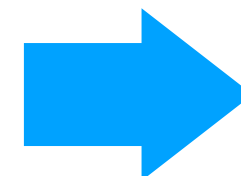
length 100   ~250   100

The assembly problem

INPUT: A collection of paired-end reads

OUTPUT: The genome from which they were sequenced

(we will see precise computational formulations later)

The assembly problem

INPUT: A collection of paired-end reads

OUTPUT: The genome from which they were sequenced

(we will see precise computational formulations later)

ACTAGTGCTAGATGCTCGAGCTAGCTAGCT

The assembly problem

INPUT: A collection of paired-end reads

OUTPUT: The genome from which they were sequenced

(we will see precise computational formulations later)

```
                              GCTCG . . . . . . . . . TAGCT
                              TGCTC . . . . . . . . . CTAGC
                              TGCTC . . . . . . . . . CTAGC
                              TGCTC . . . . . . . . . CTAGC
                              TGCTC . . . . . . . . . CTAGC
                            ATGCT . . . . . . . . . GCTAG
                           GATGC . . . . . . . . . AGCTA
                          AGATG . . . . . . . . . TAGCT
                         TAGAT . . . . . . . . . CTAGC
                  TGCTA . . . . . . GAGCT
                  TGCTA . . . . . . GAGCT
                 GTGCT . . . . . . CGAGC
                AGTGC . . . . . . TCGAG
              CTAGT . . . . . . GCTCG
              CTAGT . . . . . . GCTCG
             ACTAG . . . . . . TGCTA
```

ACTAGTGCTAGATGCTCGAGCTAGCTAGCT

The assembly problem

INPUT: A collection of paired-end reads

OUTPUT: The genome from which they were sequenced

(we will see precise computational formulations later)

```
TGCTC · · · · · · CTAGC
TGCTC · · · · · · CTAGC
TGCTA · · · · · · GAGCT
CTAGT · · · · · · GCTCG
ATGCT · · · · · · GCTAG
GATGC · · · · · · AGCTA
AGTGC · · · · · · TCGAG
AGATG · · · · · · TAGCT
TAGAT · · · · · · CTAGC
TGCTC · · · · · · CTAGC
TGCTA · · · · · · GAGCT
GTGCT · · · · · · CGAGC
TGCTC · · · · · · CTAGC
GCTCG · · · · · · TAGCT
CTAGT · · · · · · GCTCG
ACTAG · · · · · · TGCTA
```

???

Practical issues

(several of which are not covered in this course)

Repeats longer than read length

• If every substring of the genome of length = read length - 1 is unique, then assembly is trivial

AATTGAATTTACACCAC
```
AATTG
 ATTGA
  TGAAT
   TGAAT
    GAATT
     AATTT
      ATTTA
       TTTAC
        TTACA
         TACAC
          ACACC
           CACCA
            ACCAC
```
read length - 1

Repeats longer than read length

- If every substring of the genome of length = read length - 1 is unique, then assembly is trivial

- Repeated substrings at least as long as the read length introduce ambiguity

Repeats longer than read length

- If every substring of the genome of length = read length - 1 is unique, then assembly is trivial

- Repeated substrings at least as long as the read length introduce ambiguity

```
TAGTGCTTAGCGCGTAGATGCTTTAGCGCGTCGAGCTAGCT
TAGTGCTTA              AGATGCTTTA
      TTAGC                  TTAGC
      TAGCG                  TAGCG
       GCGCGT                 GCGCGT
```
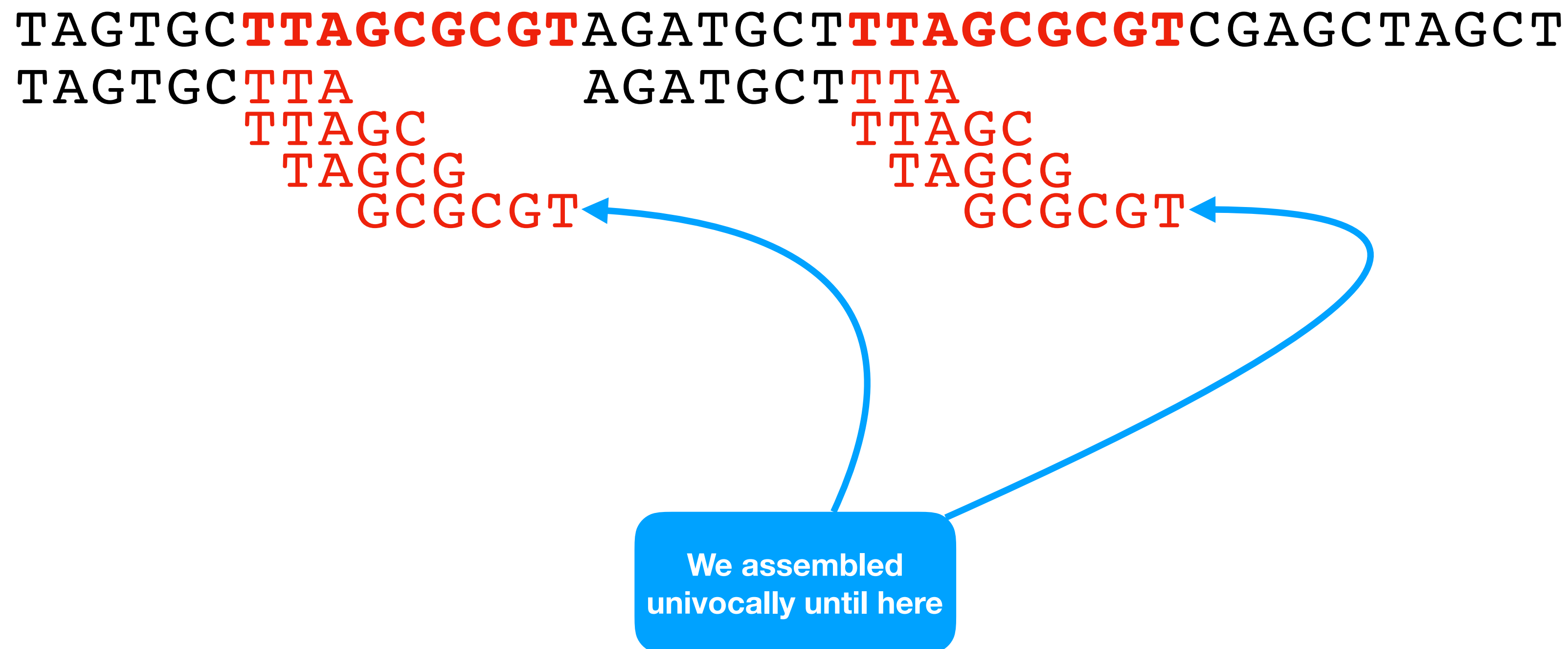
Repeats longer than read length

- If every substring of the genome of length = read length - 1 is unique, then assembly is trivial

- Repeated substrings at least as long as the read length introduce ambiguity

```
TAGTGCTTAGCGCGTAGATGCTTTAGCGCGTCGAGCTAGCT
TAGTGCTTA            AGATGCTTA
      TTAGC               TTAGC
      TAGCG               TAGCG
       GCGCGT               GCGCGT
```

We assembled univocally until here
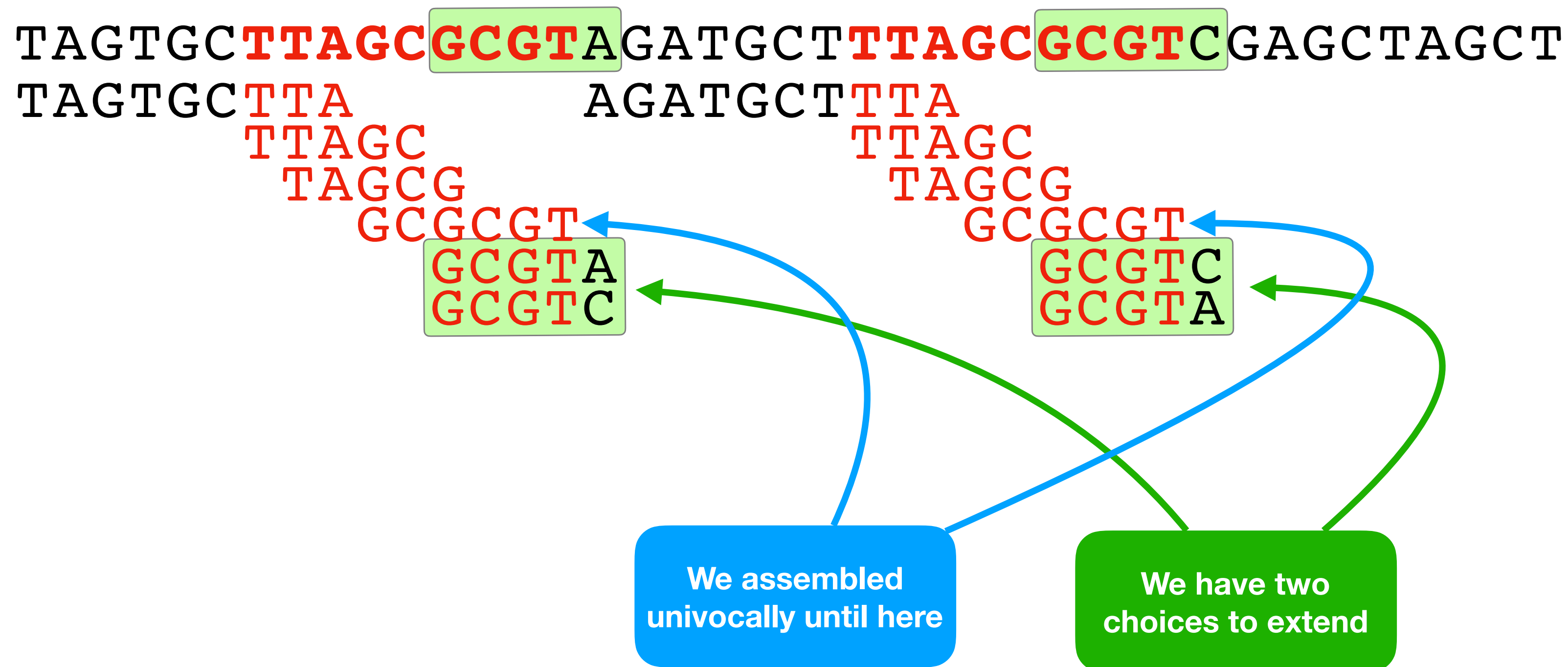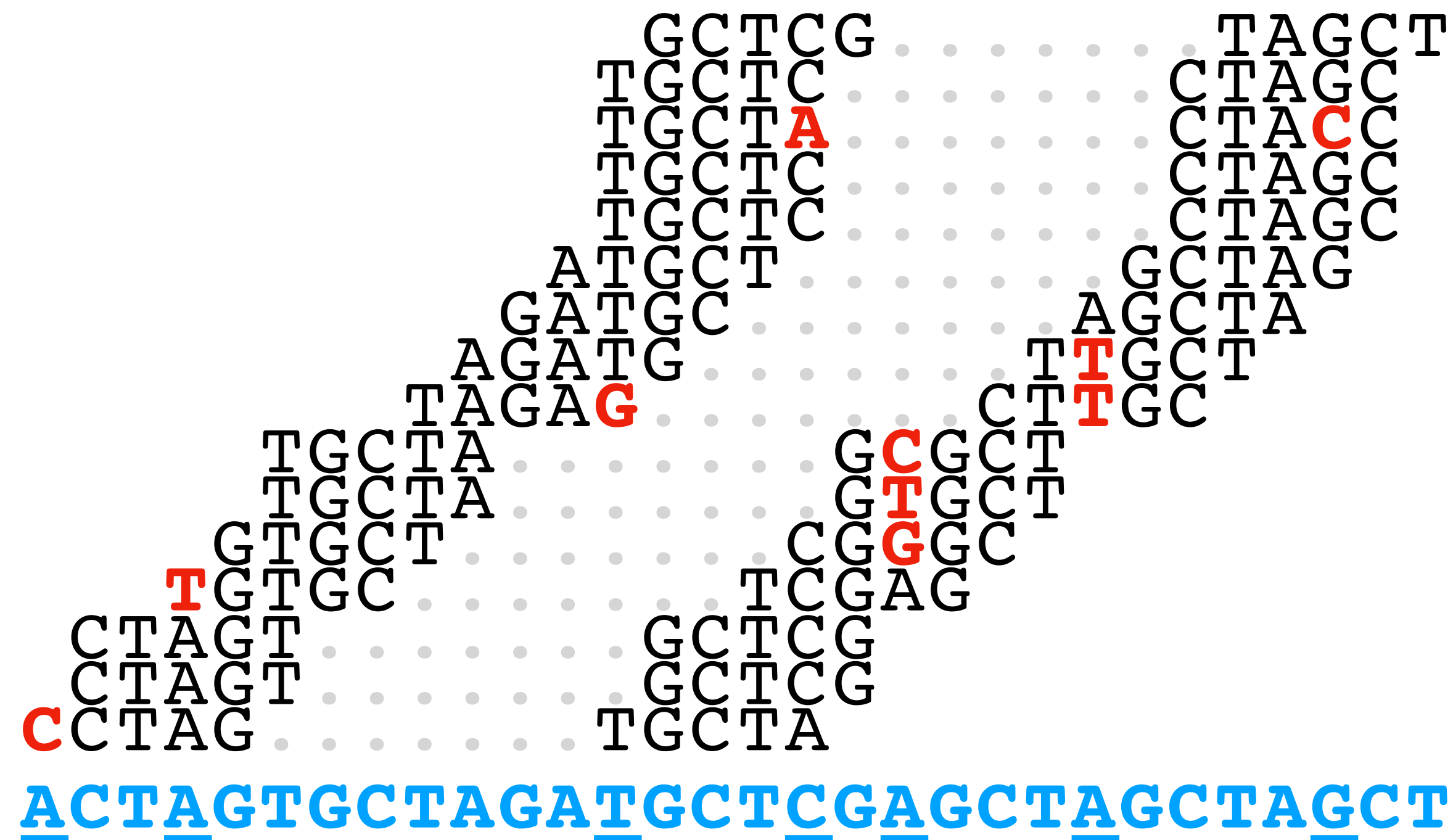
Repeats longer than read length

- If every substring of the genome of length = read length - 1 is unique, then assembly is trivial

- Repeated substrings at least as long as the read length introduce ambiguity
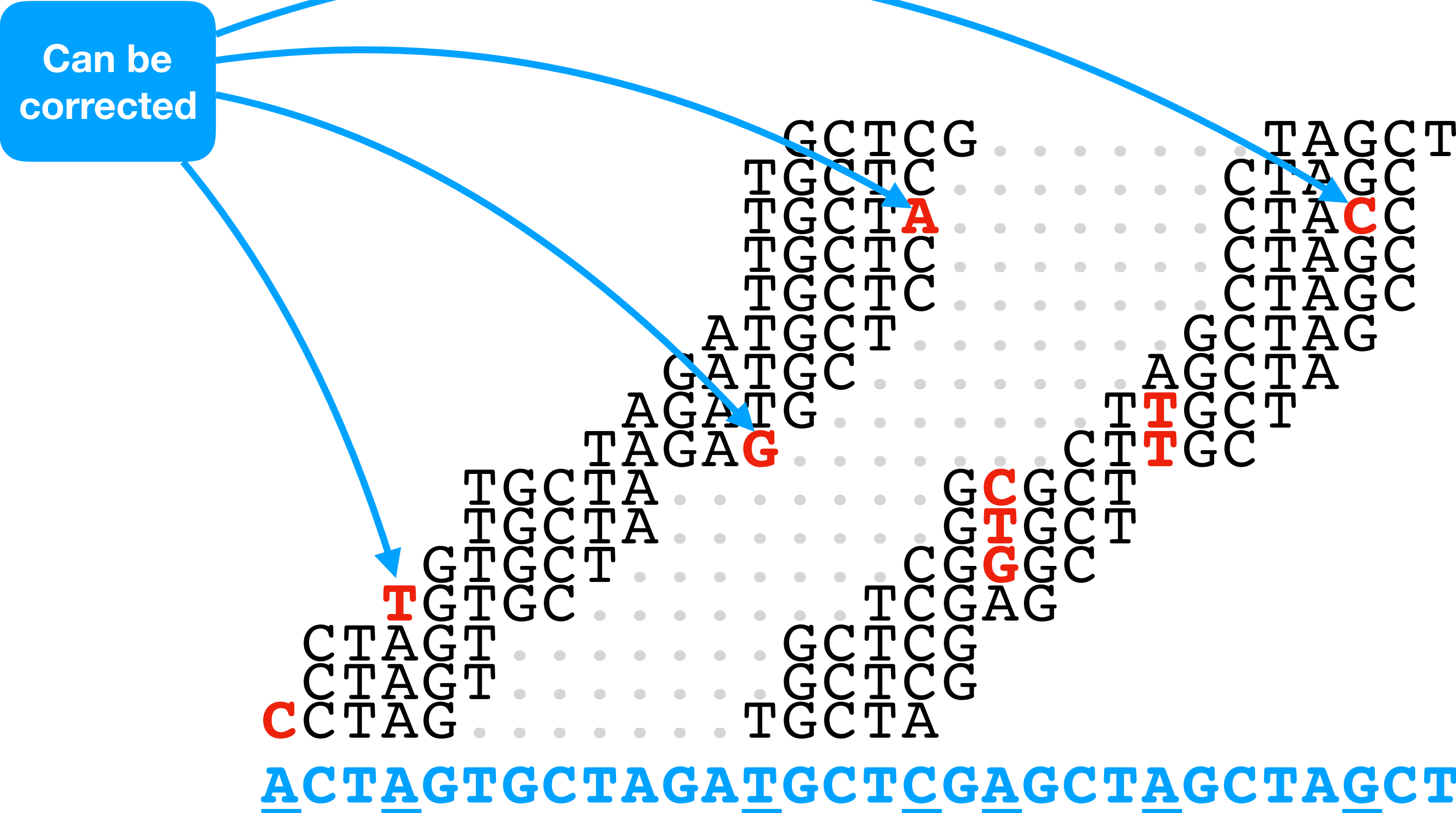
Sequencing errors
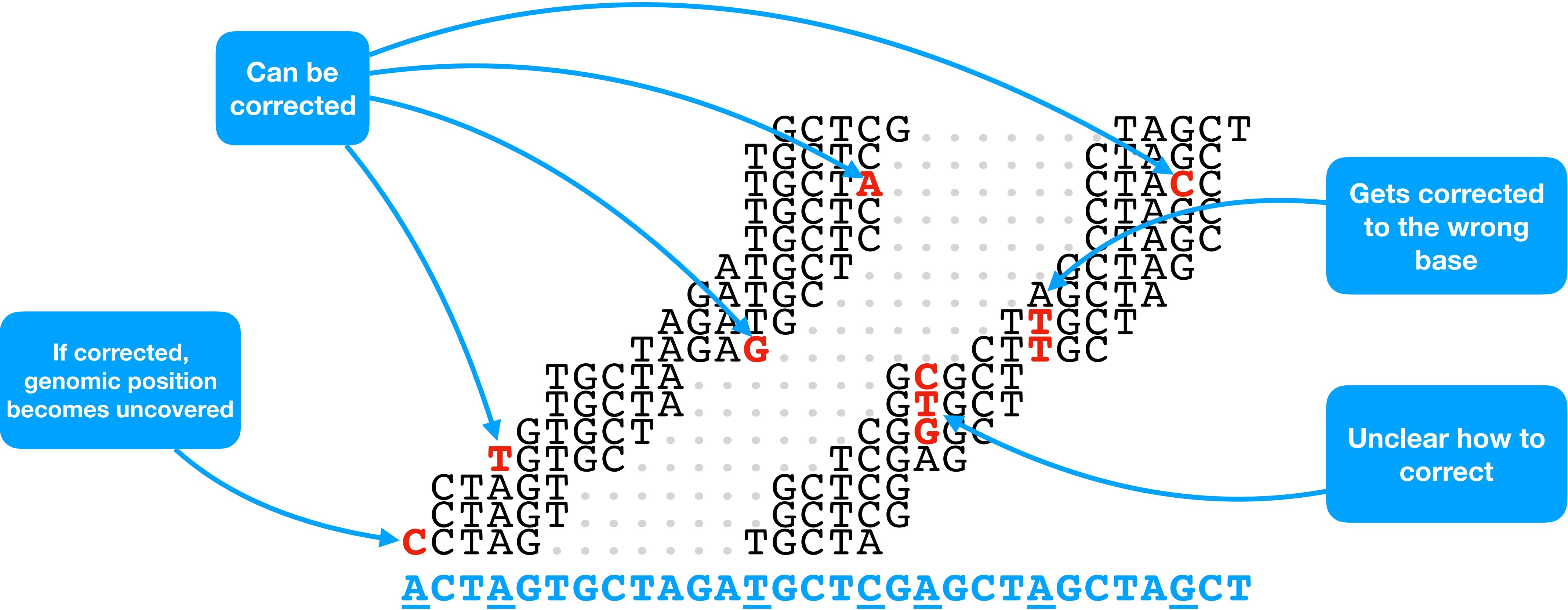
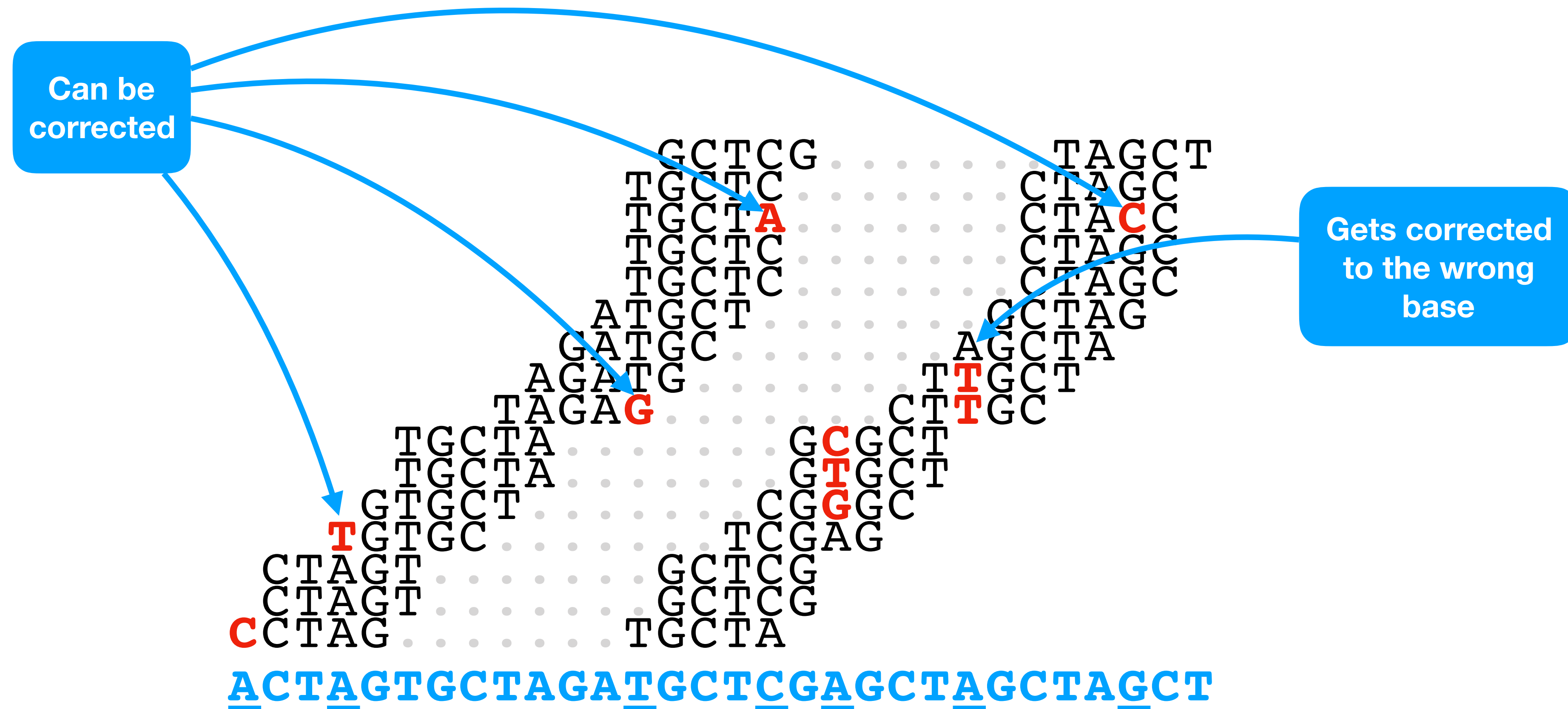# Sequencing errors

Sequencing errors
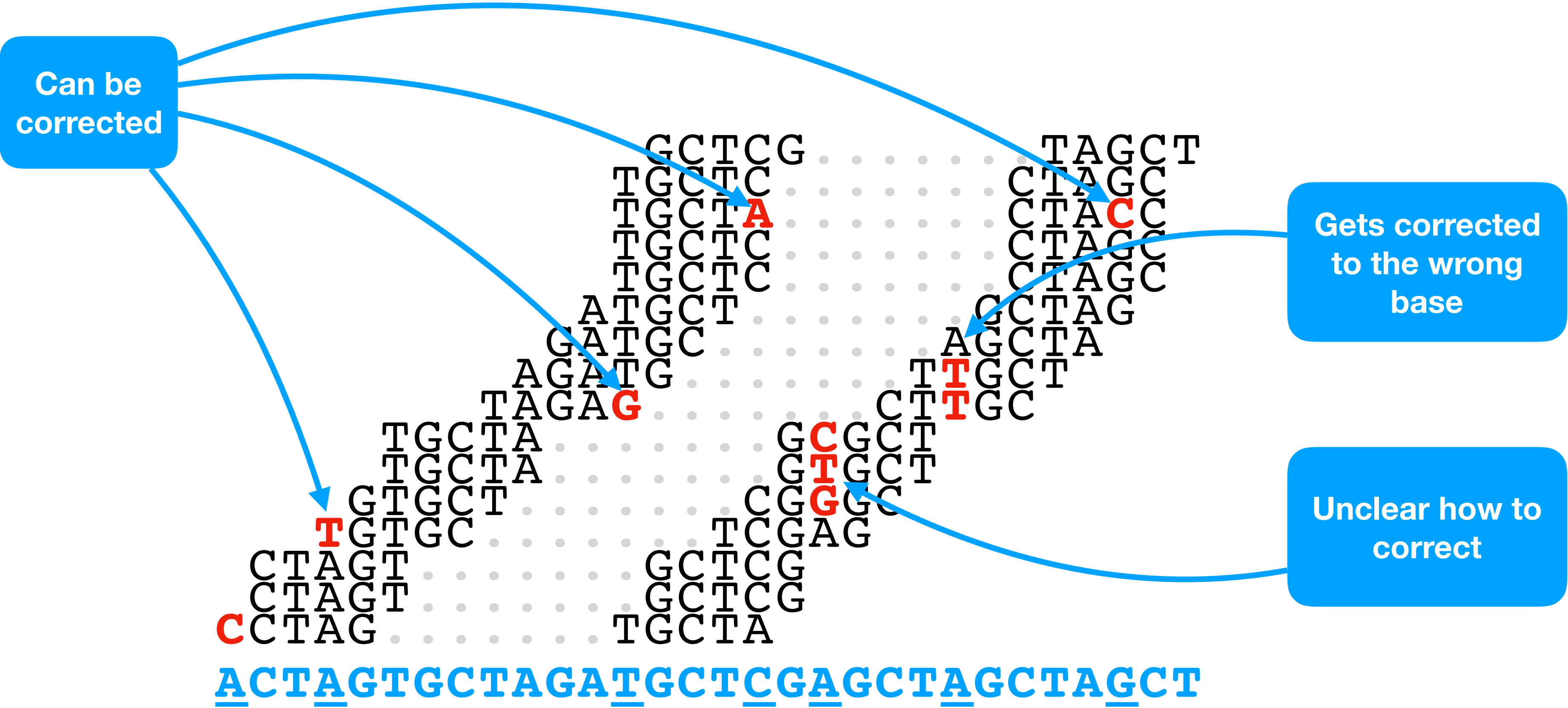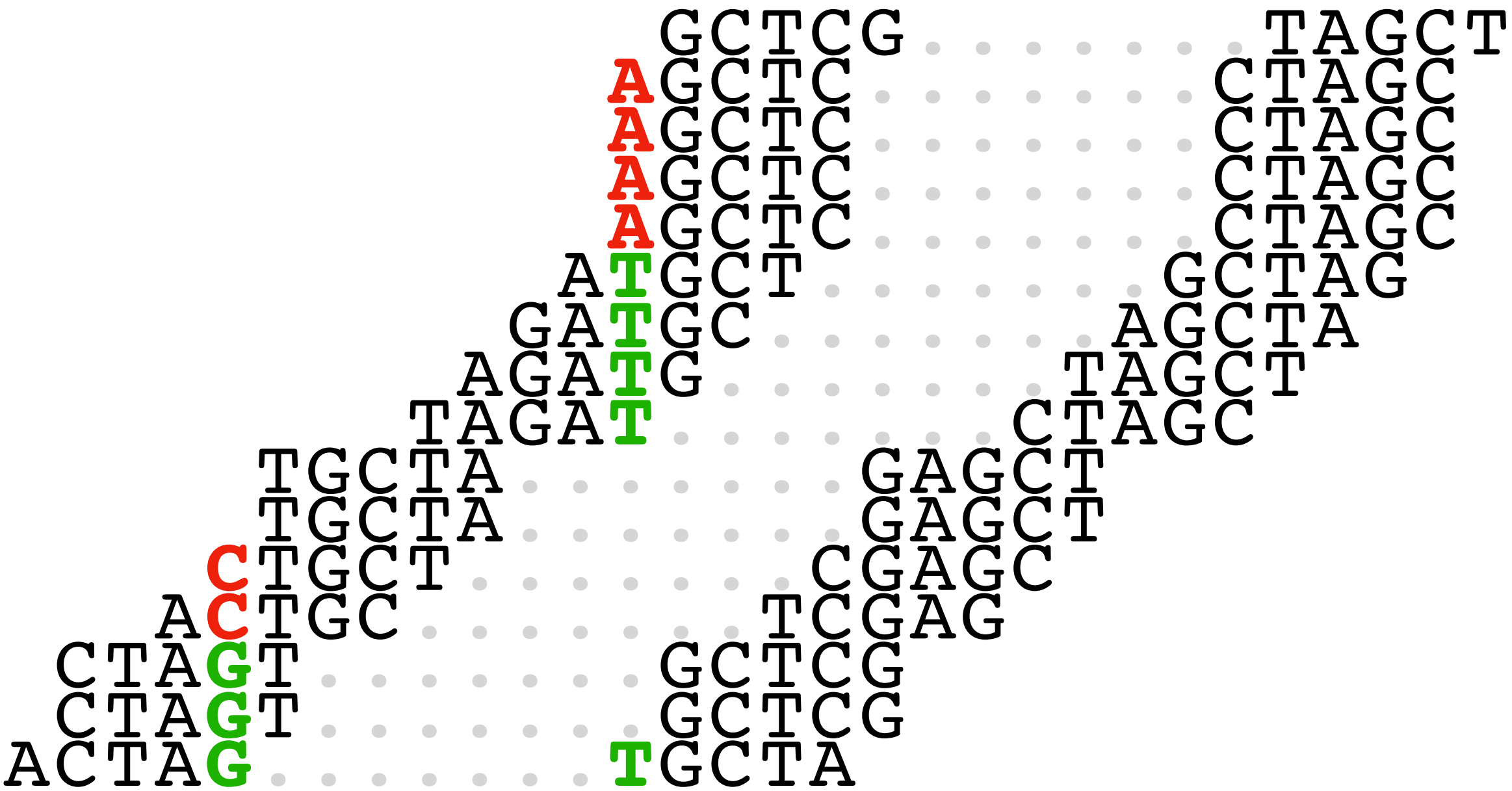
Polyploidy

Mother ACTACTGCTAGAAGCTCGAGCTAGCTAGCT
Father ACTAGTGCTAGATGCTCGAGCTAGCTAGCT

Polyploidy

```
                      GCTCG . . . . . . . . .  TAGCT
                     AGCTC . . . . . . . . .   CTAGC
                     AGCTC . . . . . . . . .   CTAGC
                     AGCTC . . . . . . . . .   CTAGC
                     AGCTC . . . . . . . .     CTAGC
                    ATGCT . . . . . . . .      GCTAG
                   GATGC . . . . . . . .      AGCTA
                  AGATG . . . . . . .        TAGCT
                 TAGAT . . . . . . .        CTAGC
               TGCTA . . . . . . .        GAGCT
               TGCTA . . . . . .          GAGCT
              CTGCT . . . . . .          CGAGC
             ACTGC . . . . . .          TCGAG
            CTAGT . . . . .           GCTCG
            CTAGT . . . . .           GCTCG
           ACTAG . . . . .      TGCTA
```

Mother **ACTACTGCTAGAAGCTCGAGCTAGCTAGCT**

Father **ACTAGTGCTAGATGCTCGAGCTAGCTAGCT**
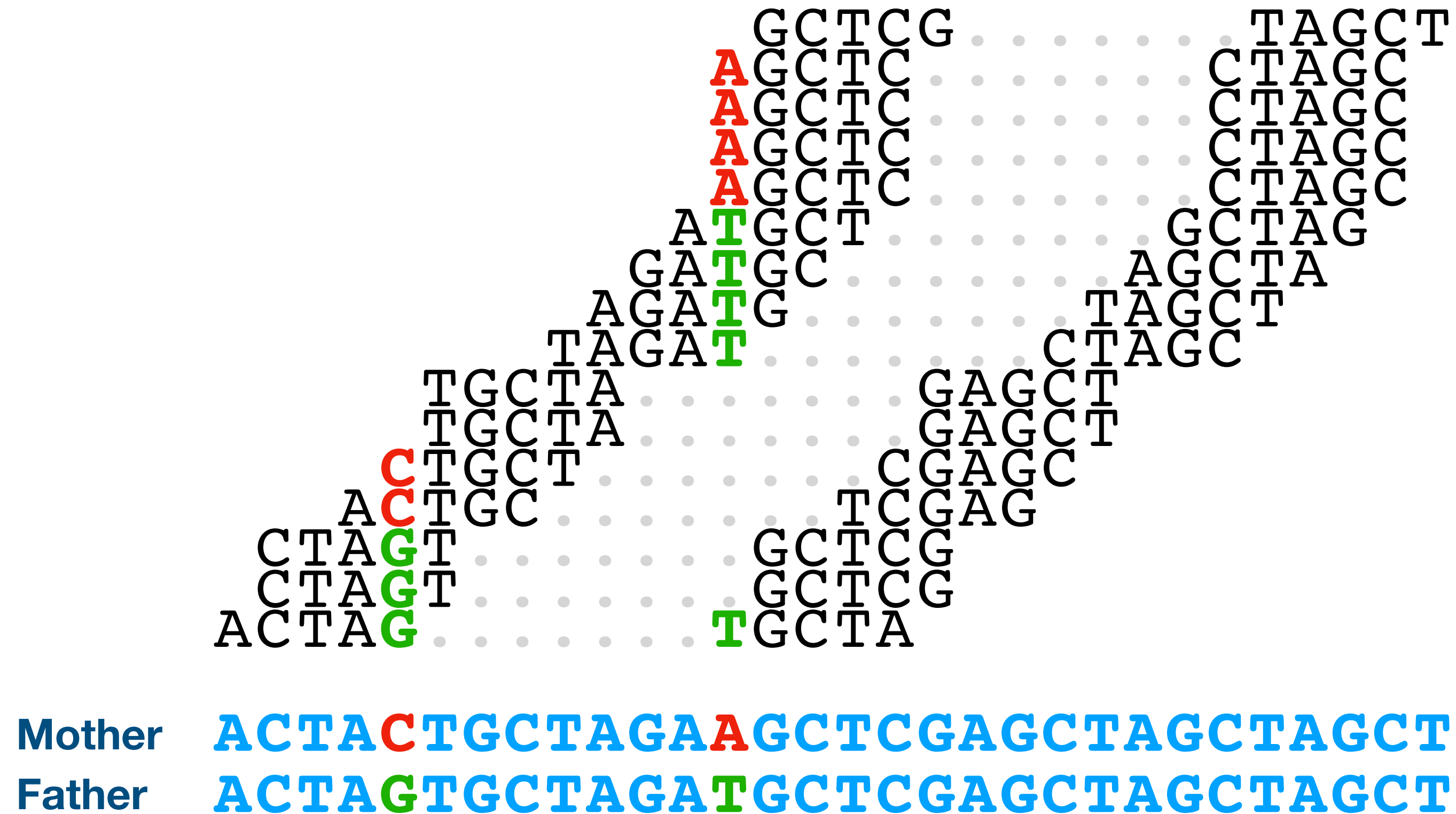
- Sequencing errors + polyploidy at the same time

- Phasing SNPs (**C** and **A** from same haplotype, **NOT** e.g. **C** and **T**) is a separate problem, called *haplotype assembly* or *haplotype phasing*

Unsequenced areas

```
                 GCTCG . . . . . . .       TAGCT
                 TGCTC . . . . . . . .      CTAGC
                 TGCTC . . . . . . . . .    CTAGC
                 TGCTC . . . . . . . . . .  CTAGC
                 TGCTC . . . . . . . . . .  CTAGC
                ATGCT . . . . . . . . . . . GCTAG
               GATGC . . . . . . . . .    AGCTA
              AGATG . . . . . . . .      TAGCT
             TAGAT . . . . . . .       CTAGC
```

```
CTAGT . . . . . . . . .     GCTCG
CTAGT . . . . . . . .       GCTCG
ACTAG . . . . . . .         TGCTA
```

ACTAGT..TAGATGCTCG..CTAGCTAGCT

Non uniform paired-end distance

```
                    GCTCG . . . . TAGCT
                    TGCTC . . . . . . CTAGC
                    TGCTC . . . . . . . CTAGC
                    TGCTC . . . CTAGC . . . CTAGC
                    TGCTC . . . . . . . . CTAGC
                    ATGCT . . . . . . GCTAG
                  GATGC . . . . . . AGCTA
                 AGATG . . . . . . . TAGCT
                TAGAT . . . . CTAGC
              TGCTA . . . . TCGAG
              TGCTA . . . . . GAGCT
             GTGCT . . . . CGAGC
            AGTGC . . . . . . CTAGC
          CTAGT . . . . . ATGCT
          CTAGT . . . . . TCGAG
        ACTAG . . . . AGATG
```
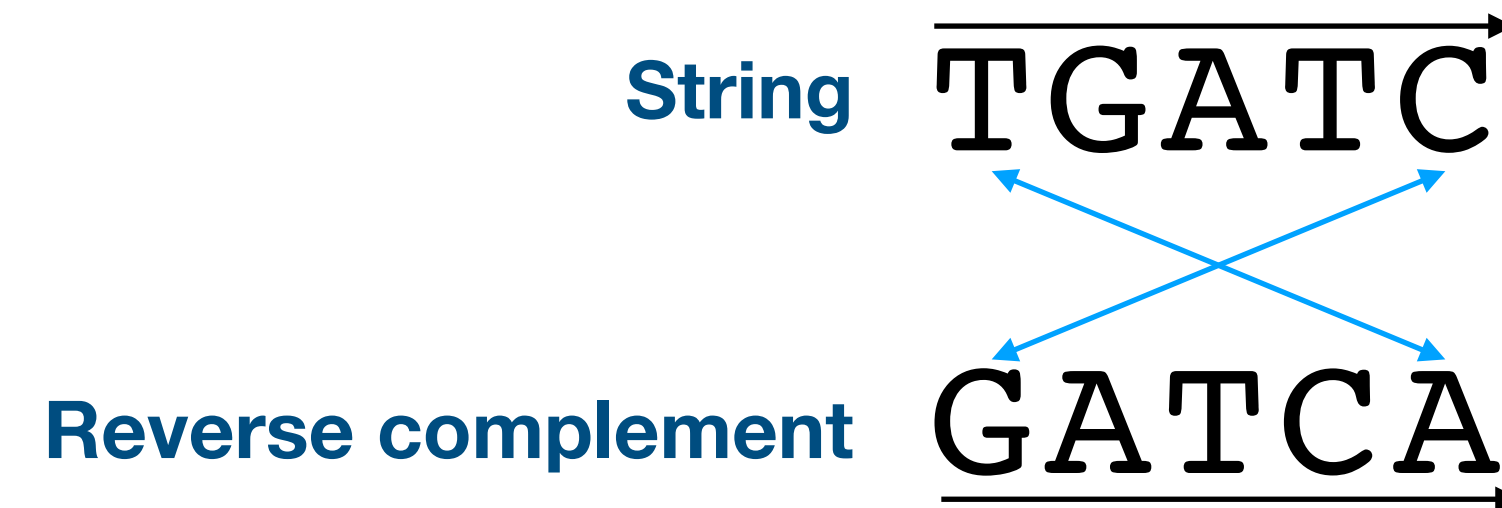
ACTAGTGCTAGATGCTCGAGCTAGCTAGCT

- Distance between each pair not known "exactly" from sequencer

Double-stranded DNA

- Reads consist of strings and their reverse complements:

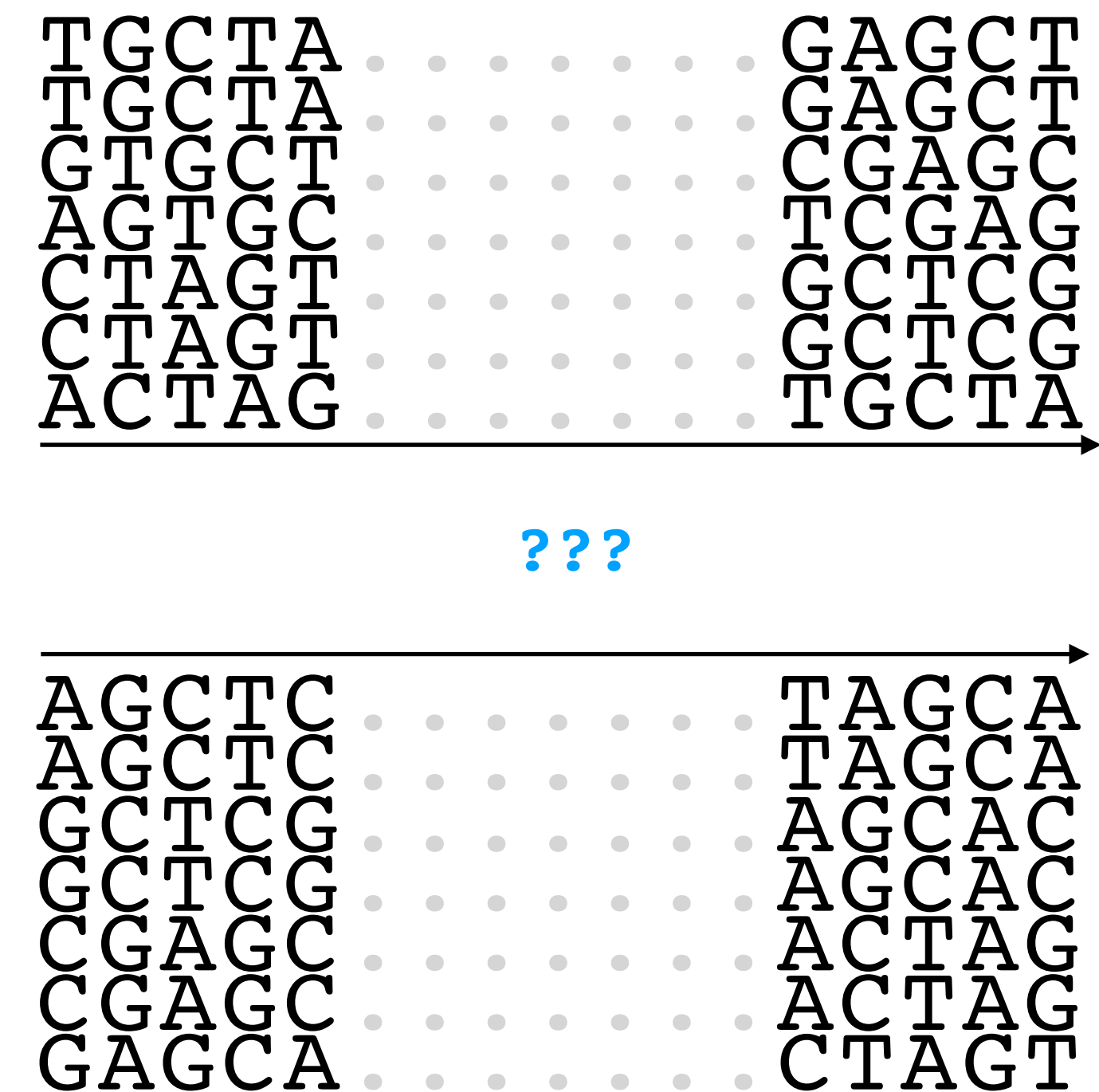**String** $\overrightarrow{\text{TGATC}}$

Reverse complement $\underrightarrow{\text{GATCA}}$

Double-stranded DNA

- Reads consist of strings and their reverse complements:

String TGATC

Reverse complement GATCA

```
TGCTA                GAGCT
TGCTA                GAGCT
GTGCT                CGAGC
AGTGC                TCGAG
CTAGT          GCTCG
CTAGT          GCTCG
ACTAG          TGCTA
```

ACTAGTGCTAGATGCTCGAGCT

TGATCACGATCTACGAGCTCGA

```
ACGAT                CTCGA
ACGAT                CTCGA
CACGA          GCTCG
TCACGA         GCTCG
GATCA          CGAGC
GATCA          CGAGC
TGATC          ACGAG
```

Double-stranded DNA

- Reads consist of strings and their reverse complements:

String $\overrightarrow{\text{TGATC}}$

Reverse complement $\underrightarrow{\text{GATCA}}$

```
TGCTA            GAGCT
 TGCTA           GAGCT
  GTGCT          CGAGC
   AGTGC         TCGAG
CTAGT          GCTCG
 CTAGT         GCTCG
ACTAG        TGCTA
```
ACTAGTGCTAGATGCTCGAGCT

TGATCACGATCTACGAGCTCGA
```
  ACGAT         CTCGA
   ACGAT        CTCGA
    CACGA       GCTCG
     TCACGA     GCTCG
  GATCA       CGAGC
   GATCA      CGAGC
TGATC       ACGAG
```

```
TGCTA            GAGCT
TGCTA            GAGCT
GTGCT            CGAGC
AGTGC            TCGAG
CTAGT            GCTCG
CTAGT            GCTCG
ACTAG            TGCTA
```

???

```
AGCTC            TAGCA
AGCTC            TAGCA
GCTCG            AGCAC
GCTCG            AGCAC
CGAGC            ACTAG
CGAGC            ACTAG
GAGCA            CTAGT
```

Large amount of data

	Genome length	Total bases at 30x coverage	Size if each base takes 2 bits
E. coli	$4.6 \cdot 10^6$	$138 \cdot 10^6$	34 MBytes
Human	$3.2 \cdot 10^9$	$96 \cdot 10^9$	24 GBytes
Spruce	$25 \cdot 10^9$	$750 \cdot 10^9$	187.5 GBytes
Axolotl	$32 \cdot 10^9$	$960 \cdot 10^9$	240 GBytes

Theoretical problem formulations

("Classical" computational formulations of how to
obtain the output from the input)

Shortest superstring

INPUT: A collection of strings (the reads)

OUTPUT: A string S such that every given string is a substring of S (S is a *superstring*), and S is shortest

Shortest superstring

INPUT: A collection of strings (the reads)

OUTPUT: A string S such that every given string is a substring of S (S is a *superstring*),
and S is shortest

```
TAGA
ATAG
CATA
TCAT
```

TCATAGA

Input   **Output** S

Shortest superstring

INPUT: A collection of strings (the reads)

OUTPUT: A string S such that every given string is a substring of S (S is a *superstring*), and S is shortest

```
TAGA              TAGA
ATAG               ATAG
CATA                CATA
TCAT            TCAT
               TCATAGA
```

Input   **Output** S

Shortest superstring

INPUT: A collection of strings (the reads)

OUTPUT: A string S such that every given string is a substring of S (S is a *superstring*),

and S is shortest

```
TAGA              TAGA
ATAG               ATAG
CATA             CATA
TCAT            TCAT
                  TCATAGA
```

Input   **Output** S

- NP-hard to compute (i.e. it cannot be solved efficiently)
- Not practical: it collapses repeats (main drawback)

Shortest superstring

INPUT: A collection of strings (the reads)

OUTPUT: A string S such that every given string is a substring of S (S is a *superstring*), and S is shortest

```
TAGA            TAGA
ATAG             ATAG
CATA            CATA
TCAT           TCAT
               TCATAGA
```

Input   **Output** S

- NP-hard to compute (i.e. it cannot be solved efficiently)

- Not practical: it collapses repeats (main drawback)

The genome [**TCATATATAGA**

The reads
```
TCAT
 CATA
  ATAT
   TATA
    ATAT
     TATA
      ATAG
       TAGA
```

Shortest superstring

INPUT: A collection of strings (the reads)

OUTPUT: A string S such that every given string is a substring of S (S is a *superstring*), and S is shortest

```
TAGA              TAGA
ATAG               ATAG
CATA             CATA
TCAT           TCAT
                 TCATAGA
```
Input   **Output** S

- NP-hard to compute (i.e. it cannot be solved efficiently)

- Not practical: it collapses repeats (main drawback)

The genome [**TCATATATAGA**

The reads [
```
TCAT
 CATA
  ATAT
   TATA
    ATAT
     TATA
      ATAG
       TAGA
```

The shortest superstring S [**TCATATAGA**

The input [
```
TCAT
 CATA
  ATAT
   TATA
    ATAT
     TATA
      ATAG
       TAGA
```

Overlap graphs + Hamiltonian path

ACTAGACTAGACC
ACTA
 CTAG
  TAGA
   AGAC
    GACT
     ACTA
      CTAG
       TAGA
        AGAC
         GACC

Overlap graphs + Hamiltonian path

```
ACTAGACTAGACC
ACTA
 CTAG
  TAGA
   AGAC
    GACT
     ACTA
      CTAG
       TAGA
        AGAC
         GACC
```

INPUT: Overlap graph of order t:

- Every read is a node

- Every suffix-prefix overlap of length $\geq t$ is an edge

OUTPUT: A path going through every node (i.e. read) exactly one (*Hamiltonian*)

Overlap graphs + Hamiltonian path

ACTAGACTAGACC
```
ACTA
 CTAG
  TAGA
   AGAC
    GACT
     ACTA
      CTAG
       TAGA
        AGAC
         GACC
```

INPUT: Overlap graph of order t:

- Every read is a node

- Every suffix-prefix overlap of length $\geq t$ is an edge

OUTPUT: A path going through every node (i.e. read) exactly one (*Hamiltonian*)

Overlap graph of order 2:

(other overlaps of length 2 not drawn)

Overlap graphs + Hamiltonian path

ACTAGA**CTAGA**CC
ACTA
 CTAG
  TAGA
   AGAC
    GACT
     ACTA
      CTAG
       TAGA
        AGAC
         GACC

INPUT: Overlap graph of order t:

- Every read is a node

- Every suffix-prefix overlap of length $\geq t$ is an edge

OUTPUT: A path going through every node (i.e. read) exactly one (*Hamiltonian*)

Overlap graph of order 2:

(other overlaps of length 2 not drawn)

Overlap graphs + Hamiltonian path

ACTAGACTAGACC
ACTA
 CTAG
  TAGA
   AGAC
    GACT
     ACTA
      CTAG
       TAGA
        AGAC
         GACC

INPUT: Overlap graph of order t:

- Every read is a node

- Every suffix-prefix overlap of length $\geq t$ is an edge

OUTPUT: A path going through every node (i.e. read) exactly one (*Hamiltonian*)

- NP-hard to compute

- Not practical: usually graph has no Hamiltonian path (missing coverage, errors)

Overlap graph of order 2:

(other overlaps of length 2 not drawn)

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

ATCATG
  ATGATC
    ATCGCC
      GCCATC
        ATCATCC

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

```
ATCATG
   ATGATC
      ATCGCC
         GCCATC
            ATCATCC
```
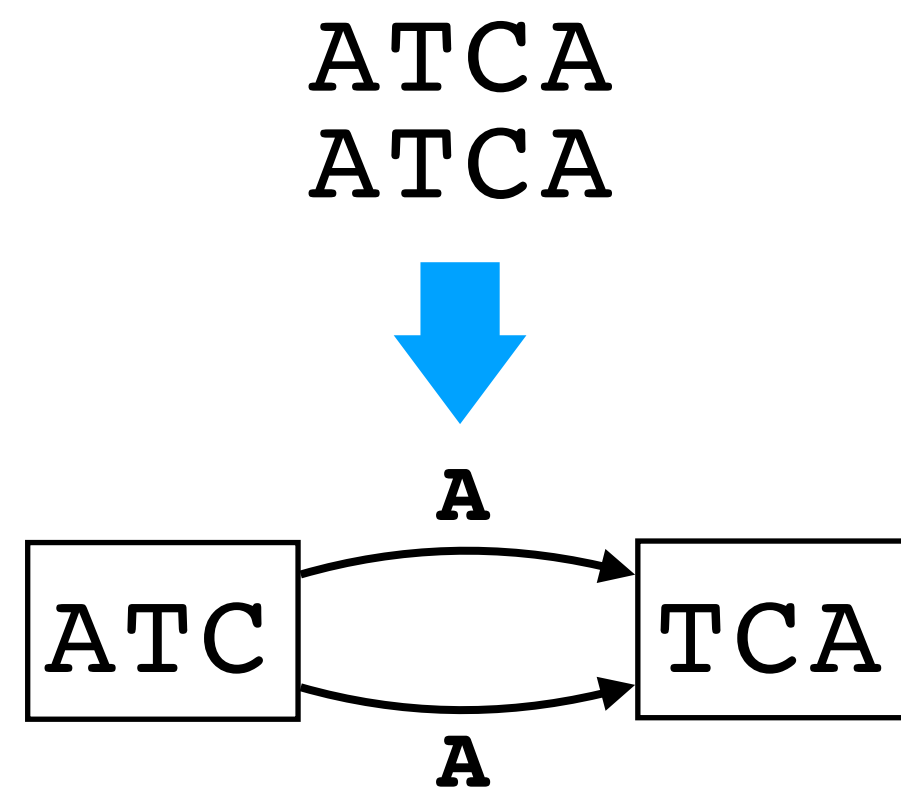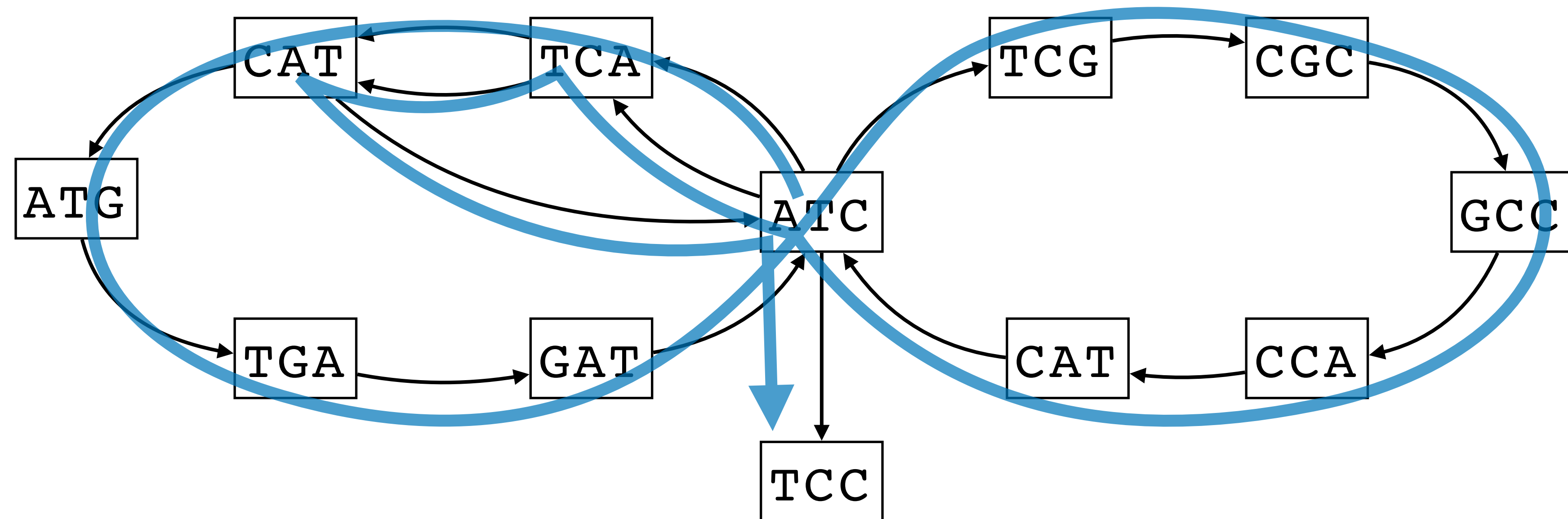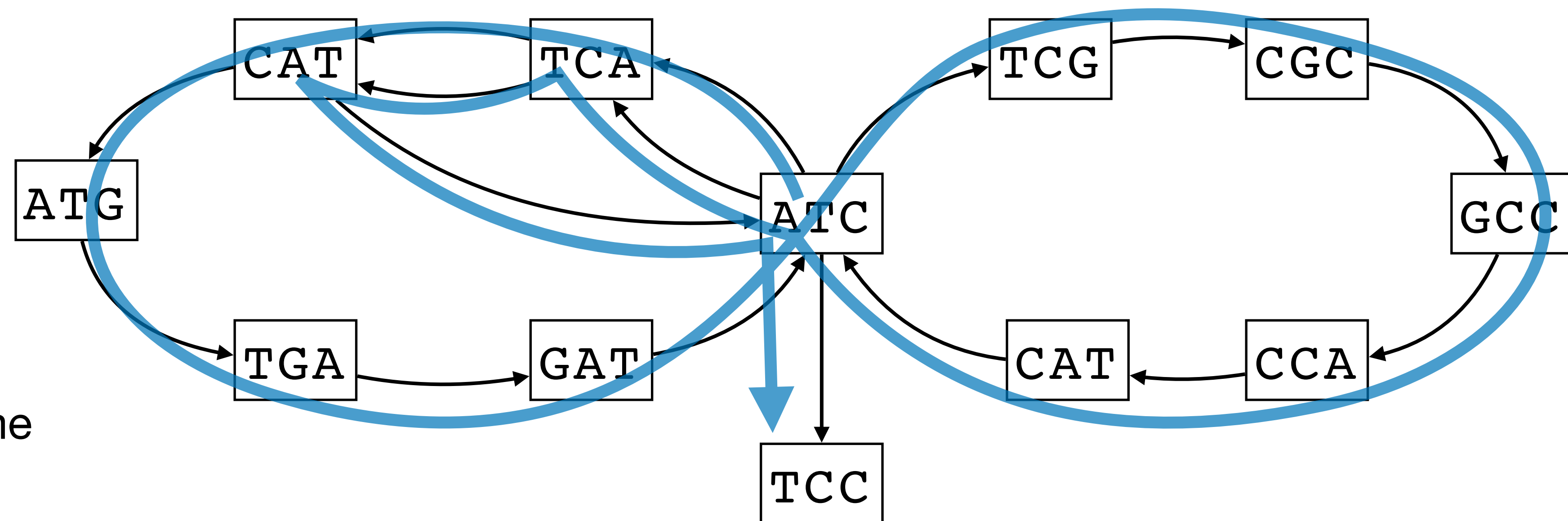
INPUT: De Bruijn graph of order k:

- Every k-mer (substring of length k) in the reads is a **single** node

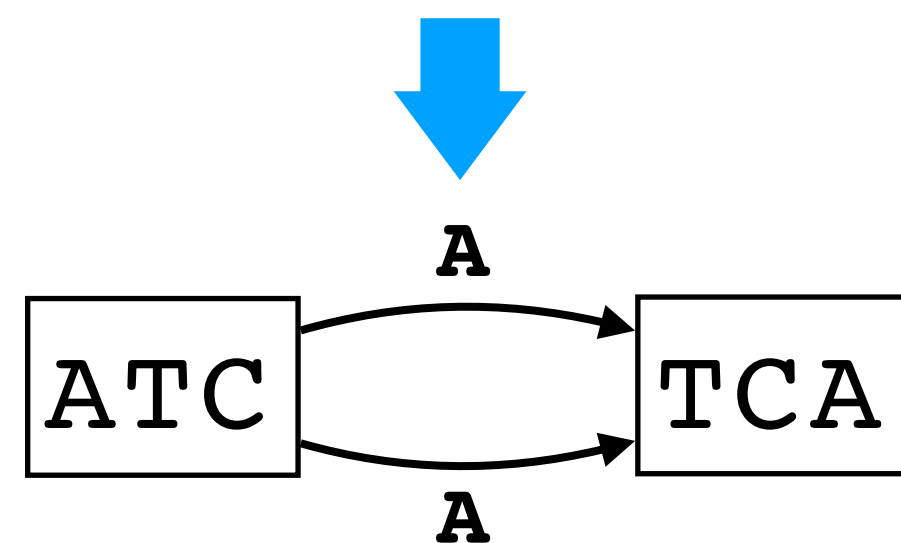- Every $(k+1)$-mer is a **different** arc from its length-k prefix to its length-k suffix

ASSUMPTION: Every length-k interval of the genome appears exactly the same number of times in the reads (*uniform coverage*)

OUTPUT: A path going through every **edge** (i.e. k-mer) exactly one (*Eulerian*)

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

```
ATCATG
   ATGATC
      ATCGCC
         GCCATC
            ATCATCC
```

INPUT: De Bruijn graph of order k:

- Every k-mer (substring of length k) in the reads is a ***single*** node

- Every $(k+1)$-mer is a ***different*** arc from its length-k prefix to its length-k suffix

ASSUMPTION: Every length-k interval of the genome appears exactly the same number of times in the reads (*uniform coverage*)

OUTPUT: A path going through every ***edge*** (i.e. k-mer) exactly one (*Eulerian*)

```
ATCA
ATCA
```

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

```
ATCATG
   ATGATC
      ATCGCC
         GCCATC
            ATCATCC
```

INPUT: De Bruijn graph of order k:

- Every k-mer (substring of length k) in the reads is a **single** node
- Every $(k+1)$-mer is a **different** arc from its length-k prefix to its length-k suffix

ASSUMPTION: Every length-k interval of the genome appears exactly the same number of times in the reads (*uniform coverage*)

OUTPUT: A path going through every **edge** (i.e. k-mer) exactly one (*Eulerian*)

De Bruijn graph of order 3:

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

```
ATCATG
  ATGATC
    ATCGCC
      GCCATC
        ATCATCC
```

INPUT: De Bruijn graph of order k:

- Every k-mer (substring of length k) in the reads is a **single** node

- Every $(k+1)$-mer is a **different** arc from its length-k prefix to its length-k suffix

ASSUMPTION: Every length-k interval of the genome appears exactly the same number of times in the reads (*uniform coverage*)

OUTPUT: A path going through every **edge** (i.e. k-mer) exactly one (*Eulerian*)

```
ATCA
ATCA
```

De Bruijn graph of order 3:

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

```
ATCATG
   ATGATC
      ATCGCC
         GCCATC
            ATCATCC
```

INPUT: De Bruijn graph of order k:

- Every k-mer (substring of length k) in the reads is a **single** node

- Every $(k+1)$-mer is a **different** arc from its length-k prefix to its length-k suffix

ASSUMPTION: Every length-k interval of the genome appears exactly the same number of times in the reads (*uniform coverage*)

OUTPUT: A path going through every **edge** (i.e. k-mer) exactly one (*Eulerian*)

De Bruijn graph of order 3:

```
ATCA
ATCA
```

De Bruijn graphs + Eulerian path

ATCATGATCGCCATCATCC

```
ATCATG
  ATGATC
    ATCGCC
      GCCATC
        ATCATCC
```

INPUT: De Bruijn graph of order k:

- Every k-mer (substring of length k) in the reads is a **single** node

- Every $(k+1)$-mer is a **different** arc from its length-k prefix to its length-k suffix

ASSUMPTION: Every length-k interval of the genome appears exactly the same number of times in the reads (*uniform coverage*)

OUTPUT: A path going through every **edge** (i.e. k-mer) exactly one (*Eulerian*)

De Bruijn graph of order 3:

- Can be solved in *O(|edges|)* time
- Too restrictive assumption

Section summary

- Modeling the genome assembly problem evolved over time (and still does)

Section summary

- Modeling the genome assembly problem evolved over time (and still does)

- Computational complexity ranges from NP-hard to linear

Section summary

- Modeling the genome assembly problem evolved over time (and still does)

- Computational complexity ranges from NP-hard to linear

- Not robust to practical issues, and hard to integrate them into the formulations

Section summary

- Modeling the genome assembly problem evolved over time (and still does)

- Computational complexity ranges from NP-hard to linear

- Not robust to practical issues, and hard to integrate them into the formulations

- <u>Most importantly (even if the above are solved)</u>:

  ‣ Many solutions, which one is the true genome?

  ‣ We will see a different "theoretical" approach, closer to practice **AT END OF LECTURE**

Practical genome assembly

(The sequence of algorithmic steps behind
"real" genome assemblers)

Contig assembly

Focus on single-end reads

Assembling a full genomic sequence in one shot is hopeless

→

Forget about the assembly model (i.e. the problem formulation)

Assemble only parts about which we are sure (contigs → **contig**uous sequences)

Contig assembly

Focus on single-end reads

Assembling a full genomic sequence in one shot is hopeless

→

Forget about the assembly model (i.e. the problem formulation)

Assemble only parts about which we are sure (contigs → **contig**uous sequences)

- Assume an assembly graph (here de Bruijn with parallel edges collapsed into one)

Contig assembly

Focus on single-end reads

Assembling a full genomic sequence in one shot is hopeless

→

Forget about the assembly model (i.e. the problem formulation)

Assemble only parts about which we are sure (contigs → **contig**uous sequences)

- Assume an assembly graph (here de Bruijn with parallel edges collapsed into one)
- Focus on *unitigs*: non-branching paths

Contig assembly

Focus on single-end reads

Assembling a full genomic sequence in one shot is hopeless

→

Forget about the assembly model (i.e. the problem formulation)

Assemble only parts about which we are sure (contigs → _contig_uous sequences)

- Assume an assembly graph (here de Bruijn with parallel edges collapsed into one)
- Focus on **_unitigs_**: non-branching paths
- Unitig $=_{\text{def}}$ "non-branching" path

Contig assembly

Focus on single-end reads

Assembling a full genomic sequence in one shot is hopeless

→

Forget about the assembly model (i.e. the problem formulation)

Assemble only parts about which we are sure (contigs → <u>contig</u>uous sequences)

- Assume an assembly graph (here de Bruijn with parallel edges collapsed into one)
- Focus on *unitigs*: non-branching paths
- Unitig $=_{\text{def}}$ "non-branching" path
- (Usually) Contig $=_{\text{def}}$ unitig in a graph "corrected" for polyploidy

Unitigs

- Let $P = (v_1, v_2, \ldots, v_{t-1}, v_t)$ be a path. We say that P is a **unitig** if either:

Unitigs

- Let $P = (v_1, v_2, \ldots, v_{t-1}, v_t)$ be a path. We say that P is a **unitig** if either:
  - ‣ $t = 2$, that is, $P = (v_1, v_2)$ is a single edge (we "trust" edges), or

Unitigs

- Let $P = (v_1, v_2, \ldots, v_{t-1}, v_t)$ be a path. We say that P is a ***unitig*** if either:

  ‣ $t = 2$, that is, $P = (v_1, v_2)$ is a single edge (we "trust" edges), or

  ‣ for every $i \in \{2, \ldots, t-1\}$, we have $d^-(v_i) = d^+(v_i) = 1$,
    (every internal node has exactly one in-neighbor and one out-neighbor)

Unitigs

- Let $P = (v_1, v_2, \ldots, v_{t-1}, v_t)$ be a path. We say that P is a **unitig** if either:

  ▸ $t = 2$, that is, $P = (v_1, v_2)$ is a single edge (we "trust" edges), or

  ▸ for every $i \in \{2, \ldots, t-1\}$, we have $d^-(v_i) = d^+(v_i) = 1$,
  (every internal node has exactly one in-neighbor and one out-neighbor)

Unitigs

- Let $P = (v_1, v_2, \ldots, v_{t-1}, v_t)$ be a path. We say that P is a **unitig** if either:

  ‣ $t = 2$, that is, $P = (v_1, v_2)$ is a single edge (we "trust" edges), or

  ‣ for every $i \in \{2, \ldots, t-1\}$, we have $d^-(v_i) = d^+(v_i) = 1$,
    (every internal node has exactly one in-neighbor and one out-neighbor)

$$P = (v_1, v_2, v_3, v_4)$$

$d^-(v_1) = 2$ $d^-(v_2) = 1$ $d^-(v_3) = 1$ $d^-(v_4) = 2$
$d^+(v_1) = 1$ $d^+(v_2) = 1$ $d^+(v_3) = 1$ $d^+(v_4) = 1$

Unitigs

- Let $P = (v_1, v_2, \ldots, v_{t-1}, v_t)$ be a path. We say that P is a **unitig** if either:
  - ▸ $t = 2$, that is, $P = (v_1, v_2)$ is a single edge (we "trust" edges), or
  - ▸ for every $i \in \{2, \ldots, t-1\}$, we have $d^-(v_i) = d^+(v_i) = 1$,
    (every internal node has exactly one in-neighbor and one out-neighbor)

$$P = (v_1, v_2, v_3, v_4)$$

- We want maximal (longest) unitigs

Example
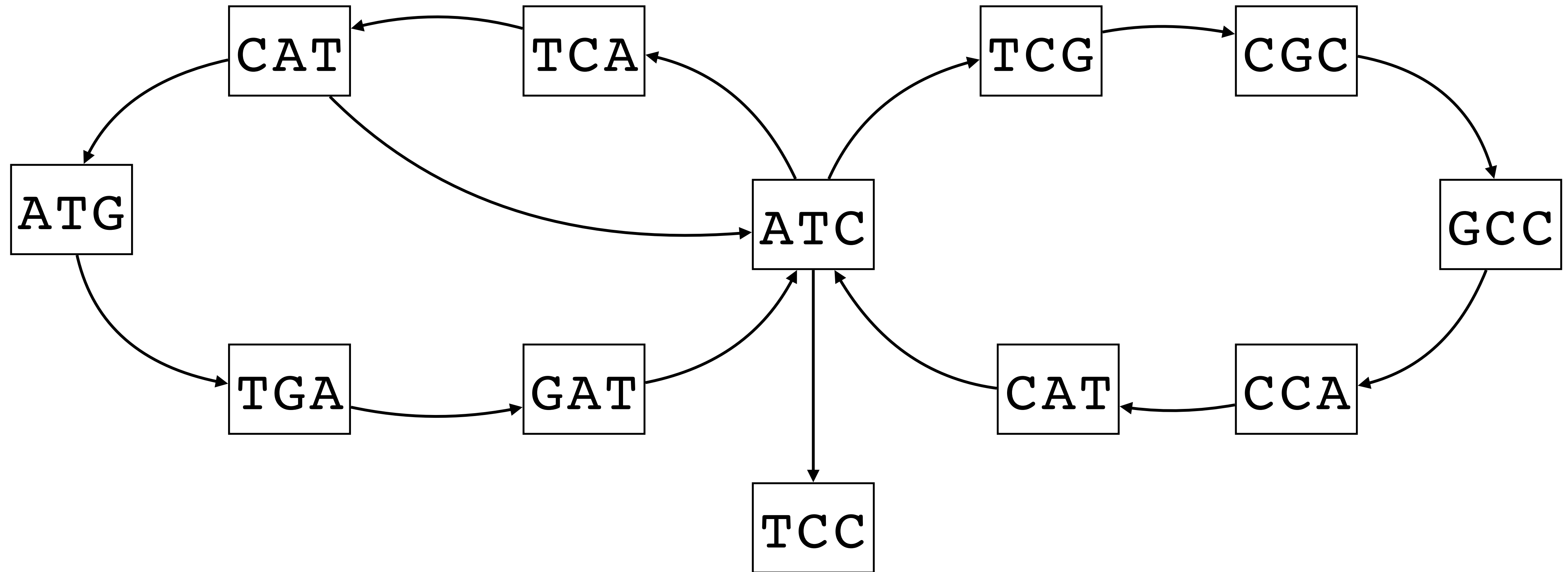
Example
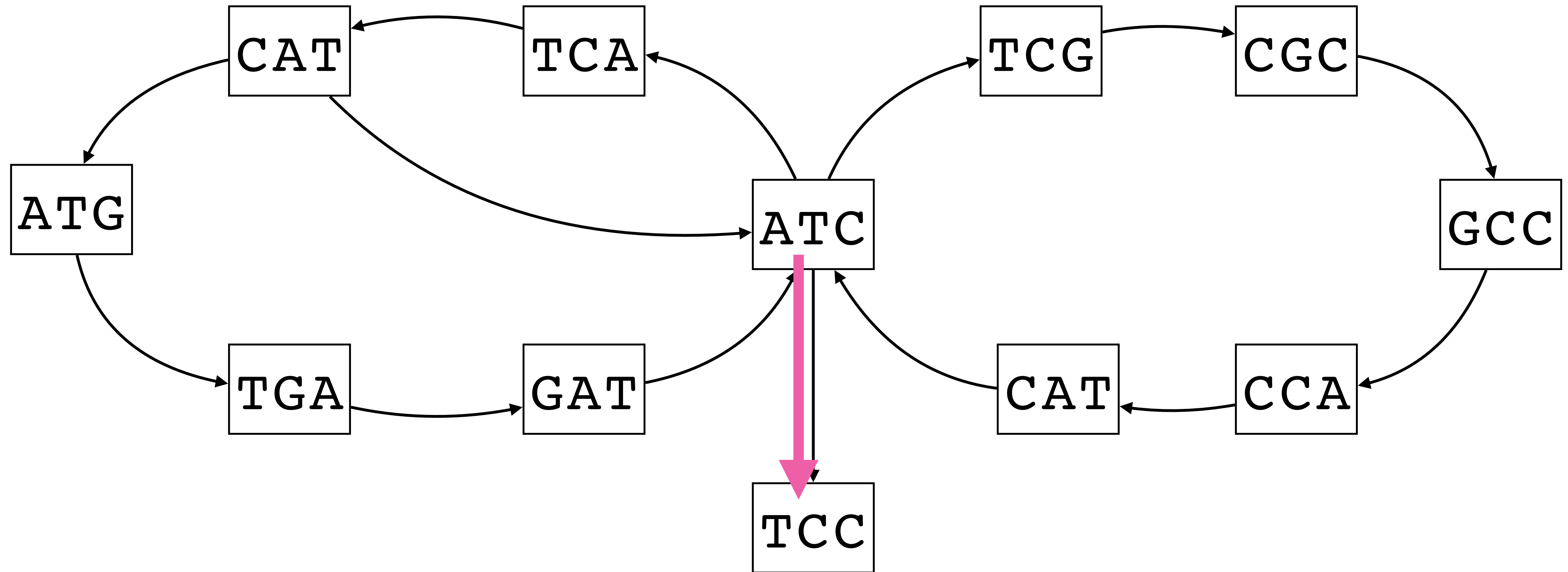
Example
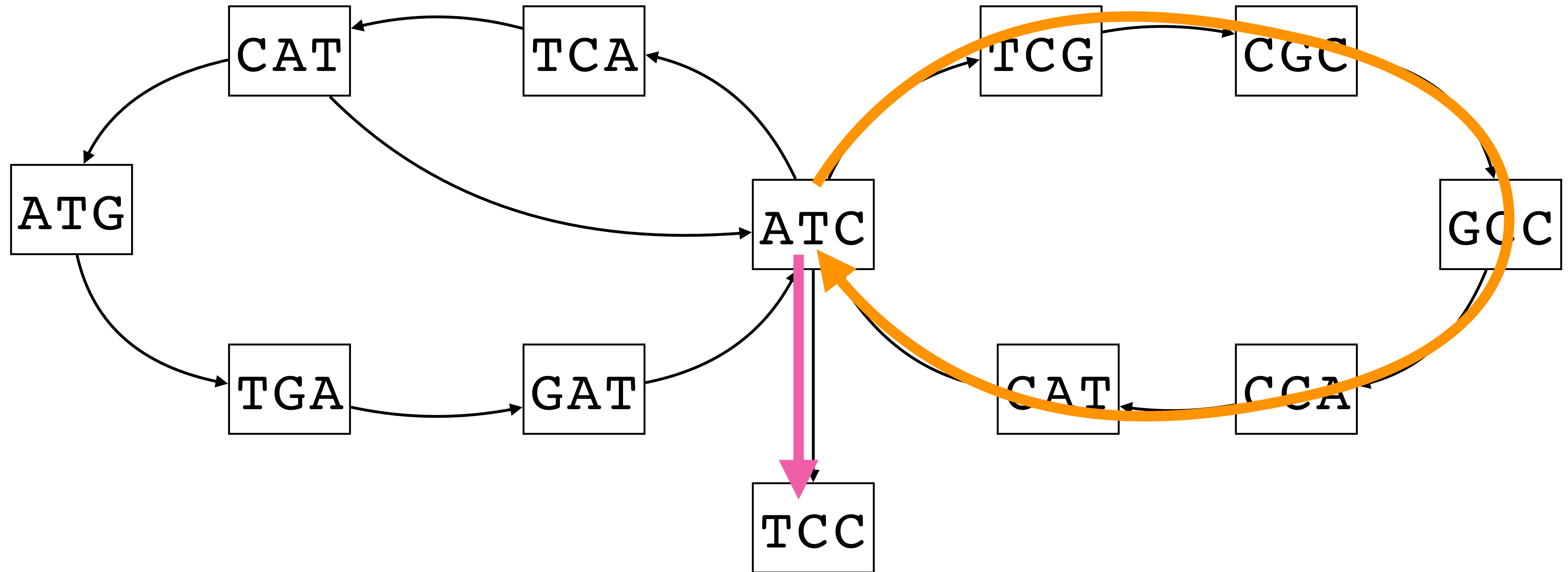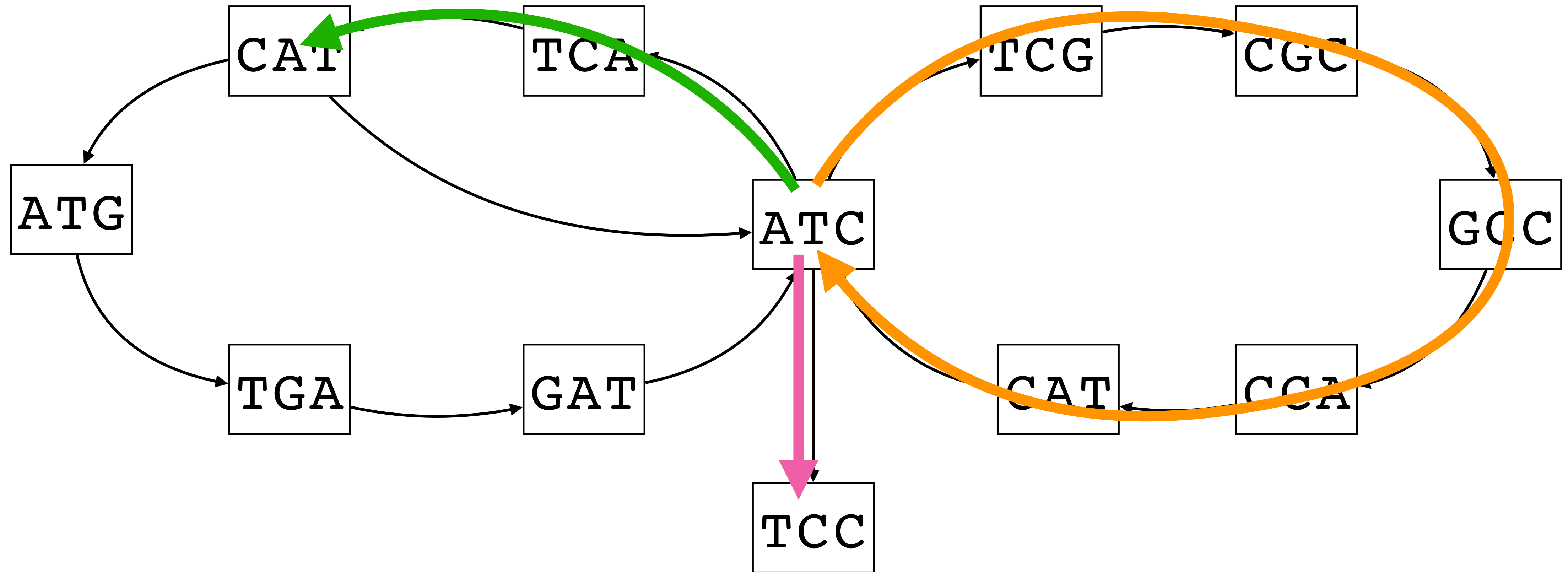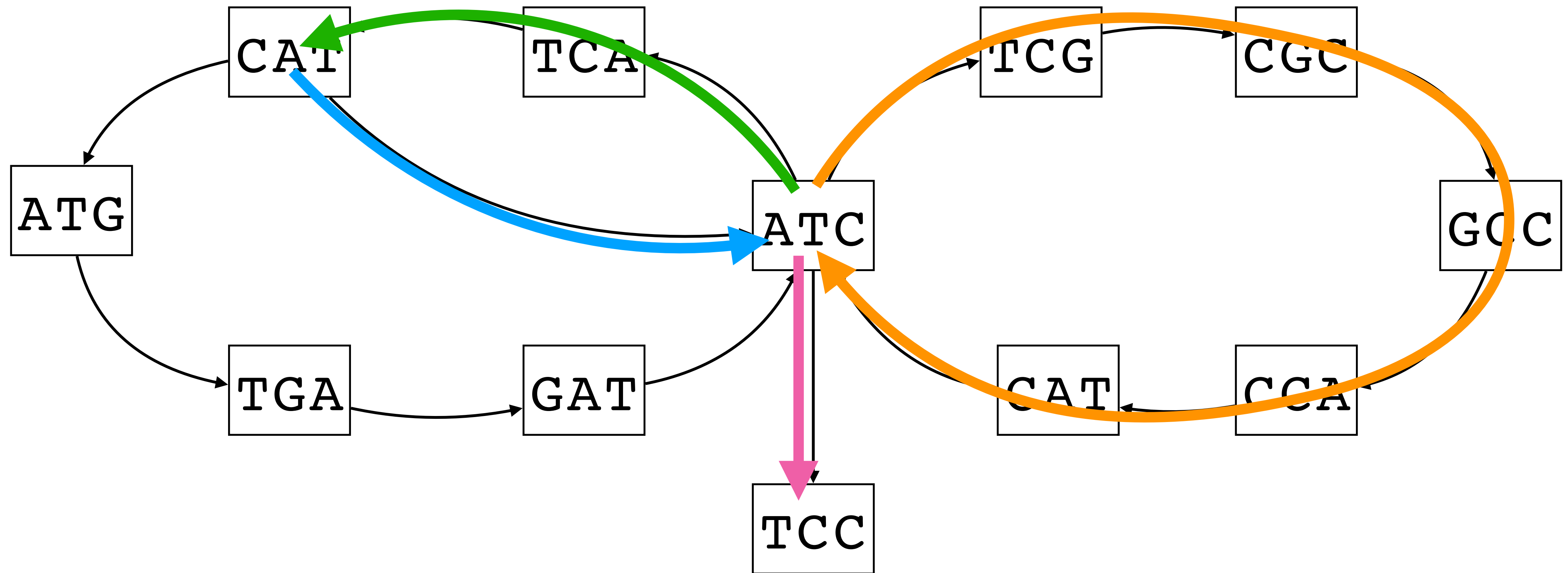
Example
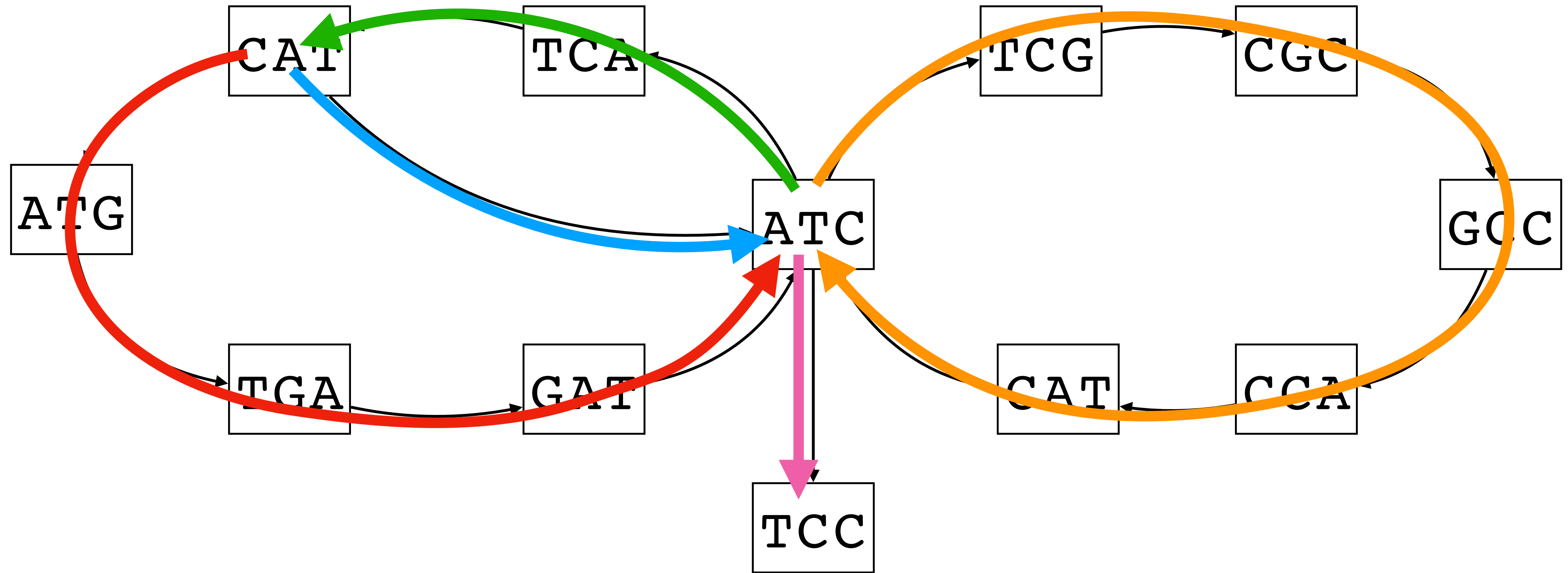
Example

Example

Example
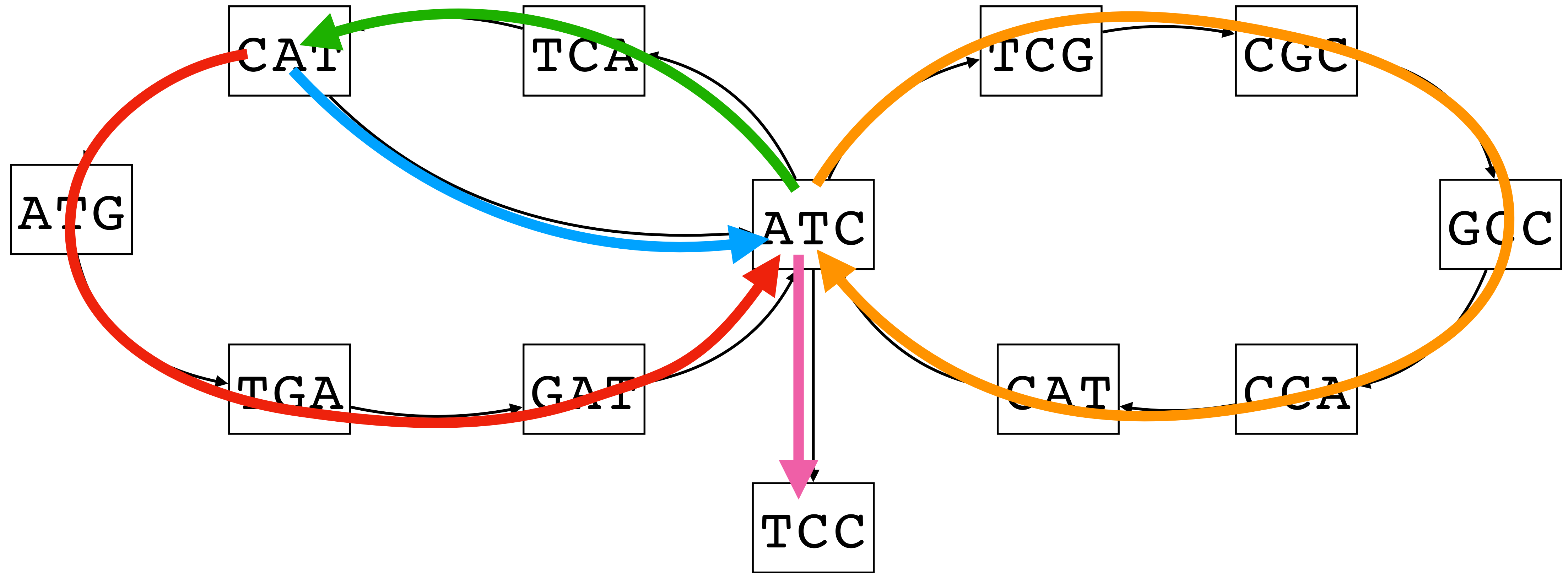
- Unitigs can be found in $O(|\text{edges}|)$ time

Bubble popping

Bubble popping

ATGTA
CATGT
G**C**ATG
TG**C**AT
TTG**C**A
ATTG**C**
AATTG
Mother **AATTGCATGTA**
Father **AATTGAATGTA**
AATTG
ATTG**A**
TTG**A**A
TG**A**AT
G**A**ATG
AATGT
ATGTA

Bubble popping

- Bubble $=_{\text{def}}$ two paths of same length with same start and end node
  - ‣ Caused by a SNP in haplotype not present in the other

Bubble popping

ATGTA
CATGT
G**C**ATG
TG**C**AT
TTG**C**A
ATTG**C**
AATTG
Mother **AATTGCATGTA**
Father **AATTGAATGTA**
AATTG
ATTG**A**
TTG**A**A
TG**A**AT
G**A**ATG
AATGT
ATGTA

- Bubble $=_{def}$ two paths of same length with same start and end node
  - ‣ Caused by a SNP in haplotype not present in the other
- Leads to shorter unitigs (poliploidy can be solved later)

Bubble popping

- Bubble $=_{def}$ two paths of same length with same start and end node
  - ▸ Caused by a SNP in haplotype not present in the other
- Leads to shorter unitigs (poliploidy can be solved later)

Bubble popping

- Bubble $=_{\text{def}}$ two paths of same length with same start and end node

  ‣ Caused by a SNP in haplotype not present in the other

- Leads to shorter unitigs (poliploidy can be solved later)

- Generalizations possible (e.g. *superbubble*), handling more complex scenarios

  ‣ Taku Onodera, Kunihiko Sadakane, Tetsuo Shibuya:
  Detecting Superbubbles in Assembly Graphs. WABI 2013: 338-348

Bubble popping

- Bubble $=_{\text{def}}$ two paths of same length with same start and end node
  - ‣ Caused by a SNP in haplotype not present in the other
- Leads to shorter unitigs (poliploidy can be solved later)

- Generalizations possible (e.g. *superbubble*), handling more complex scenarios
  - ‣ Taku Onodera, Kunihiko Sadakane, Tetsuo Shibuya:
    Detecting Superbubbles in Assembly Graphs. WABI 2013: 338-348

Bubble popping

- Bubble $=_{def}$ two paths of same length with same start and end node

  ‣ Caused by a SNP in haplotype not present in the other

- Leads to shorter unitigs (poliploidy can be solved later)

- Generalizations possible (e.g. *superbubble*), handling more complex scenarios

  ‣ Taku Onodera, Kunihiko Sadakane, Tetsuo Shibuya:
  Detecting Superbubbles in Assembly Graphs. WABI 2013: 338-348

Tip removal

```
       ATGTA
      GATGT
      GCATG
       TGCAT
       TTGCA
     ATTGC
     ATTGA
   AATTG
   AATTGCATGTA
```

Tip removal

ATGTA
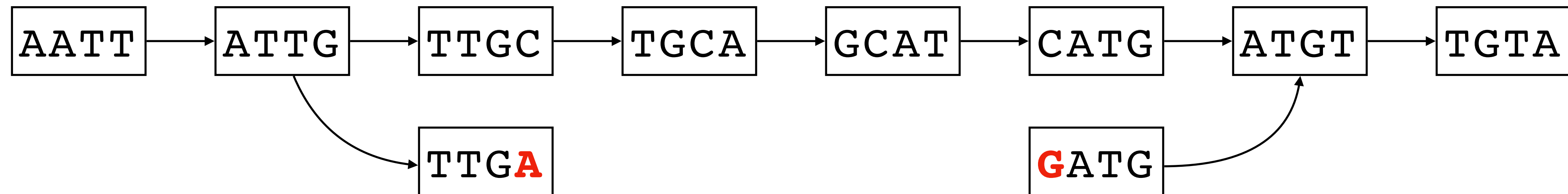**G**ATGT
GCATG
TGCAT
TTGCA
ATTGC
ATTG**A**
AATTG
AATTGCATGTA

Tip removal

```
        ATGTA
       GATGT
      GCATG
     TGCAT
    TTGCA
   ATTGC
   ATTGA
  AATTG
  AATTGCATGTA
```

- Tip $=_{def}$ a "short" path that either:

  ▸ ends in a *sink* v (i.e. $d^+(v) = 0$)

  ▸ starts in a *source* v (i.e. $d^-(v) = 0$)

Tip removal

```
        ATGTA
       GATGT
       GCATG
      TGCAT
     TTGCA
    ATTGC
    ATTGA
   AATTG
   AATTGCATGTA
```

- Tip $=_{\text{def}}$ a "short" path that either:

  ‣ ends in a *sink* v (i.e. $d^+(v) = 0$)

  ‣ starts in a *source* v (i.e. $d^-(v) = 0$)

- Caused by sequencing errors that remain after error correction

Tip removal

```
      ATGTA
     GATGT
    GCATG
   TGCAT
  TTGCA
 ATTGC
 ATTGA
AATTG
AATTGCATGTA
```
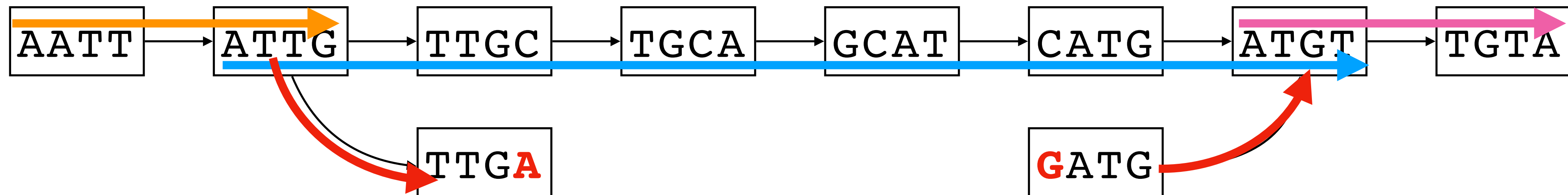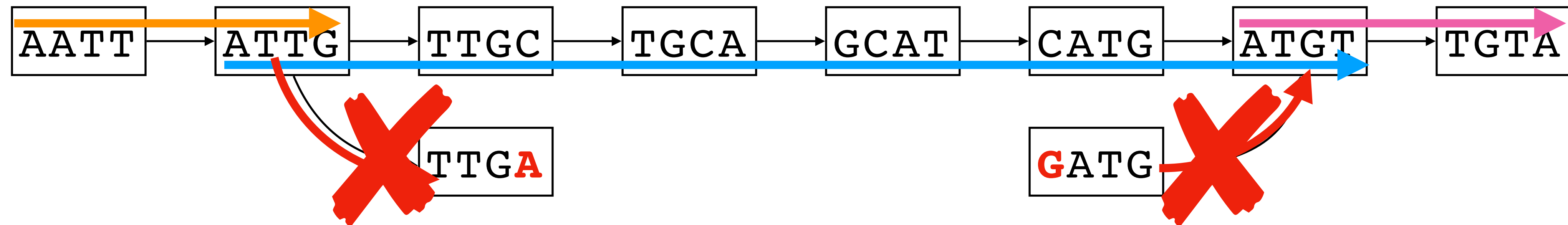
- Tip $=_{\text{def}}$ a "short" path that either:

  ‣ ends in a *sink* v (i.e. $d^+(v) = 0$)

  ‣ starts in a *source* v (i.e. $d^-(v) = 0$)

- Caused by sequencing errors that remain after error correction

- Leads to shorter unitigs

Tip removal

```
        ATGTA
      G ATGT
      GCATG
      TGCAT
      TTGCA
     ATTGC
     ATTG A
    AATTG
    AATTGCATGTA
```
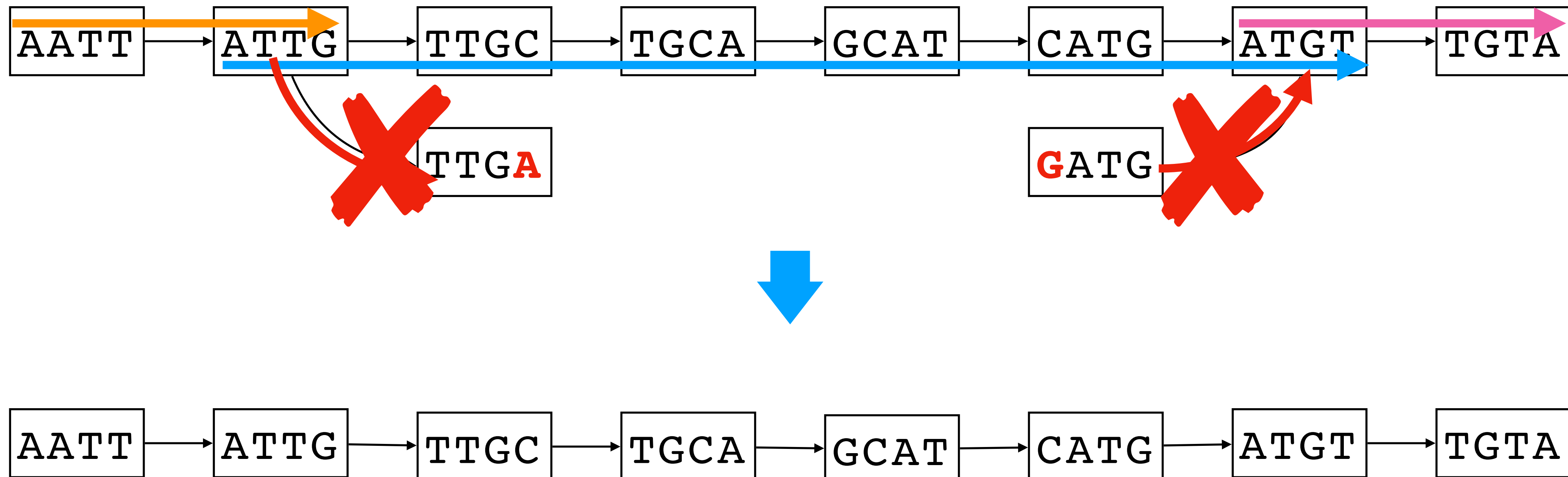
- Tip $=_{\text{def}}$ a "short" path that either:

  ‣ ends in a *sink* v (i.e. $d^+(v) = 0$)

  ‣ starts in a *source* v (i.e. $d^-(v) = 0$)

- Caused by sequencing errors that remain after error correction
- Leads to shorter unitigs

Tip removal

```
      ATGTA
     GATGT
    GCATG
   TGCAT
  TTGCA
 ATTGC
 ATTGA
AATTG
AATTGCATGTA
```
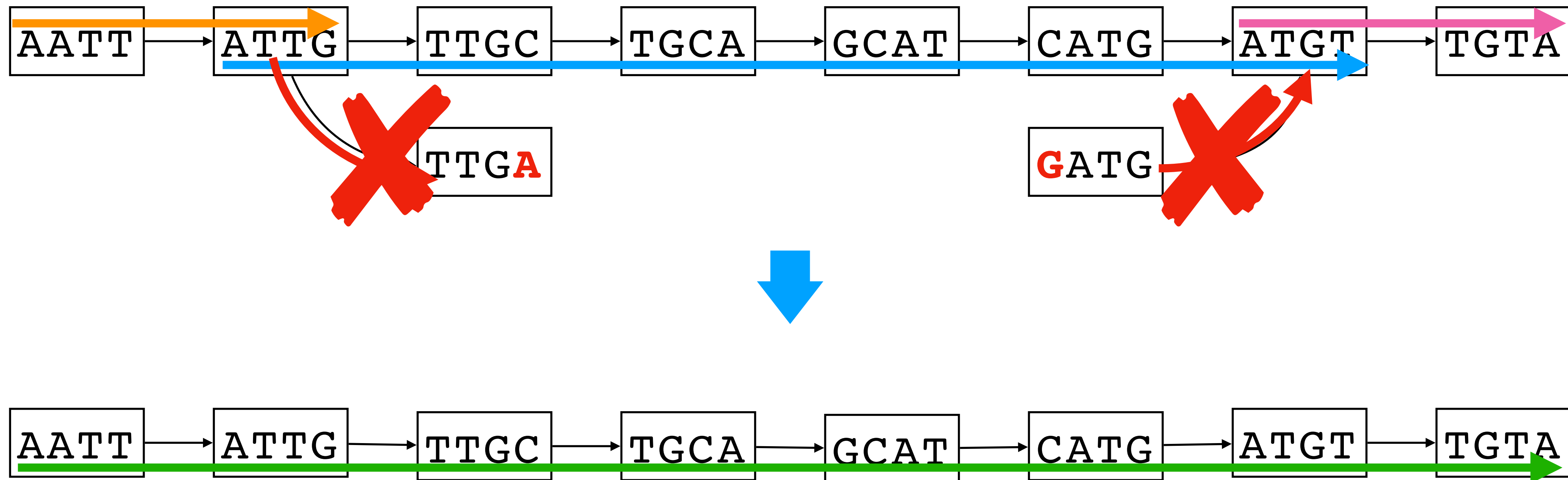
- Tip $=_{\text{def}}$ a "short" path that either:

  ▸ ends in a *sink* v (i.e. $d^+(v) = 0$)

  ▸ starts in a *source* v (i.e. $d^-(v) = 0$)

- Caused by sequencing errors that remain after error correction

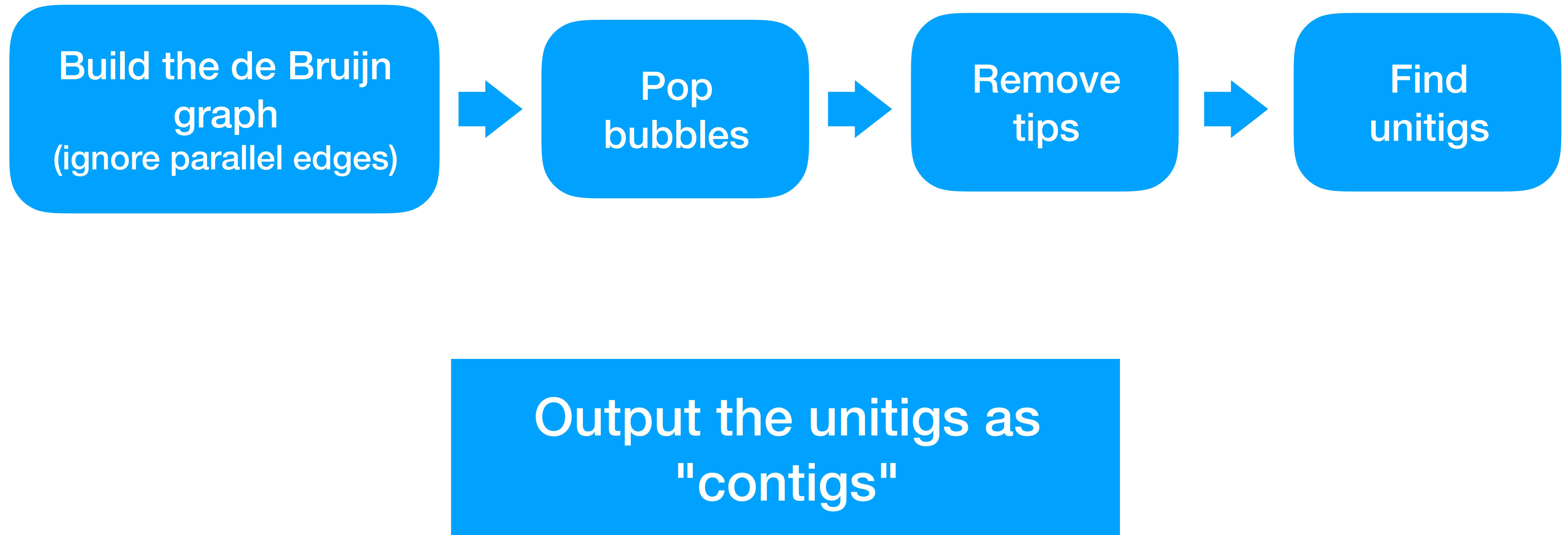- Leads to shorter unitigs

Tip removal

```
        ATGTA
       GATGT
       GCATG
       TGCAT
      TTGCA
     ATTGC
     ATTGA
    AATTG
    AATTGCATGTA
```

- Tip $=_{\text{def}}$ a "short" path that either:

  ‣ ends in a *sink* v (i.e. $d^+(v) = 0$)

  ‣ starts in a *source* v (i.e. $d^-(v) = 0$)

- Caused by sequencing errors that remain after error correction

- Leads to shorter unitigs

Tip removal

```
        ATGTA
      GATGT
      GCATG
       TGCAT
      TTGCA
     ATTGC
     ATTGA
    AATTG
    AATTGCATGTA
```

- Tip $=_{\text{def}}$ a "short" path that either:
  - ‣ ends in a *sink* v (i.e. $d^+(v) = 0$)
  - ‣ starts in a *source* v (i.e. $d^-(v) = 0$)
- Caused by sequencing errors that remain after error correction
- Leads to shorter unitigs

Contigs assembly

(simplified to ignore some practical issues e.g. errors, reverse complements)

Build the de Bruijn graph (ignore parallel edges) → Pop bubbles → Remove tips → Find unitigs

Output the unitigs as "contigs"
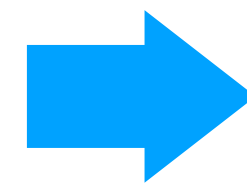
Scaffolding

Bring in paired-end information

Align reads to contigs

→

Chain (order) the contigs

Output chains of contigs with "gaps" (`NNNN...`) between them

Contigs

```
TCGATAGCTAAAA
AATTGT
ATAGAGATATTT
ATATCGCTAGA
```

→

Scaffolds

```
TCGATAGCTAAAANNNNNNNNNNNNAATTGTNNNATAGAGATATTT
ATATCGCTAGA
```

Align paired-end reads to contigs

```
              ATATA......15.......TGCAA
            AGAAT.............24..............GTAAT
          ATACA....11.....CTAAT
         AATAG.......16.......TGCAA
```
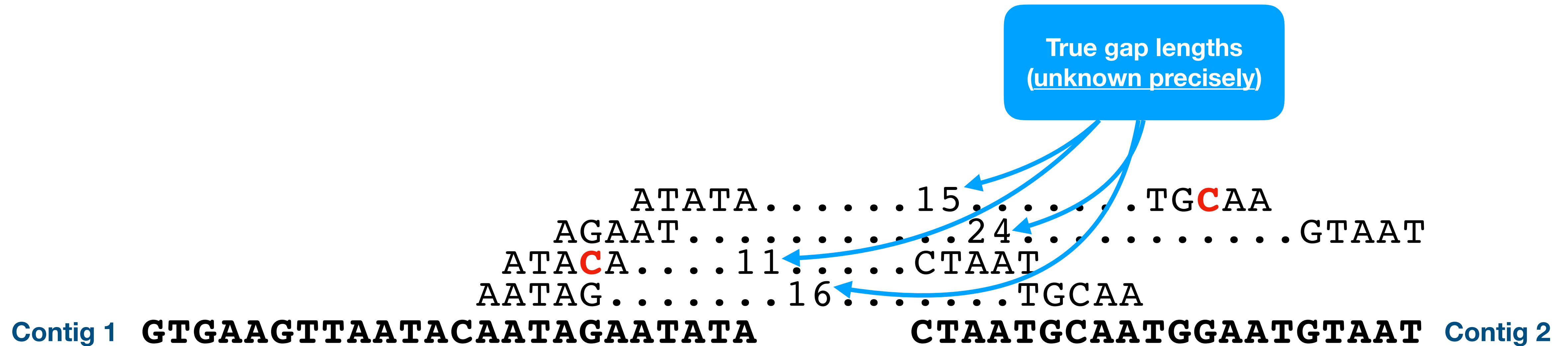**Contig 1** **GTGAAGTTAATACAATAGAATATA**   **CTAATGCAATGGAATGTAAT** **Contig 2**

- Align paired-end reads to contigs, focus on read pairs aligning to different contigs
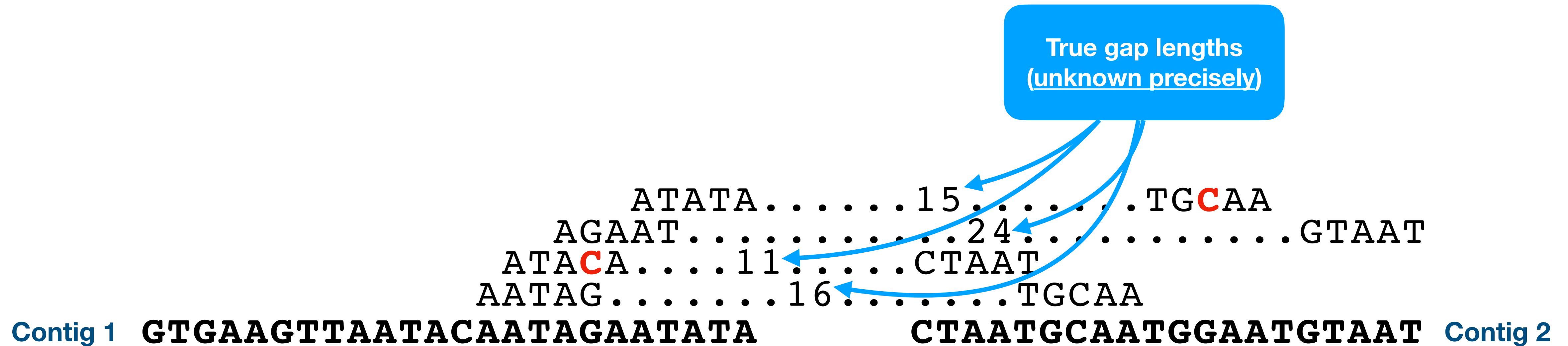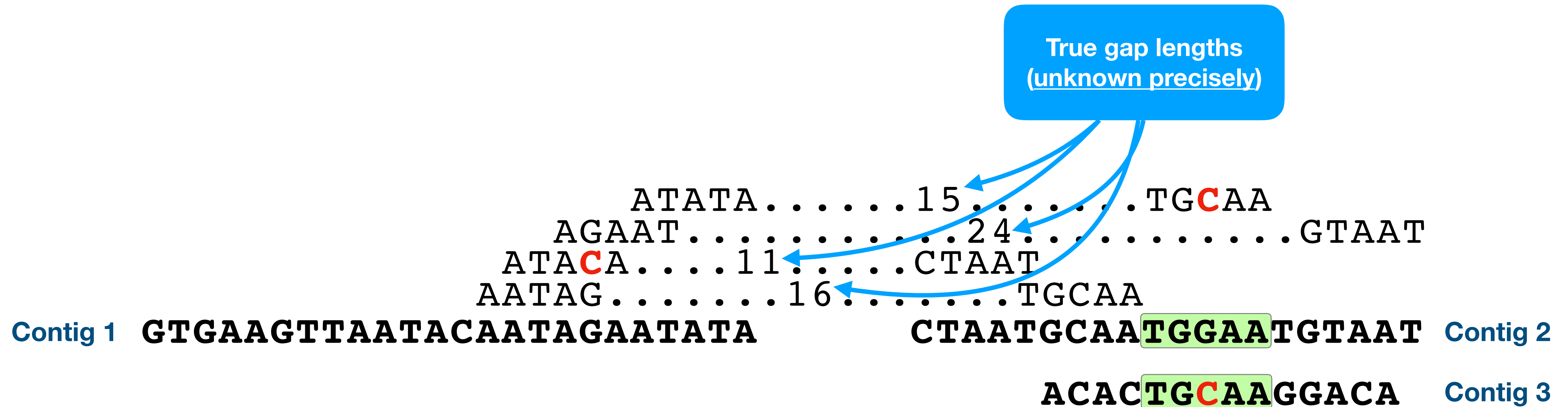
Align paired-end reads to contigs

True gap lengths (unknown precisely)

```
ATATA.......15........TGCAA
AGAAT..............24.................GTAAT
ATACA....11......CTAAT
AATAG........16.........TGCAA
```

Contig 1 **GTGAAGTTAATACAATAGAATATA**   **CTAATGCAATGGAATGTAAT** **Contig 2**

- Align paired-end reads to contigs, focus on read pairs aligning to different contigs

Align paired-end reads to contigs

True gap lengths (unknown precisely)

```
ATATA.......15........TGCAA
AGAAT...............24................GTAAT
ATACA....11......CTAAT
AATAG........16.........TGCAA
```

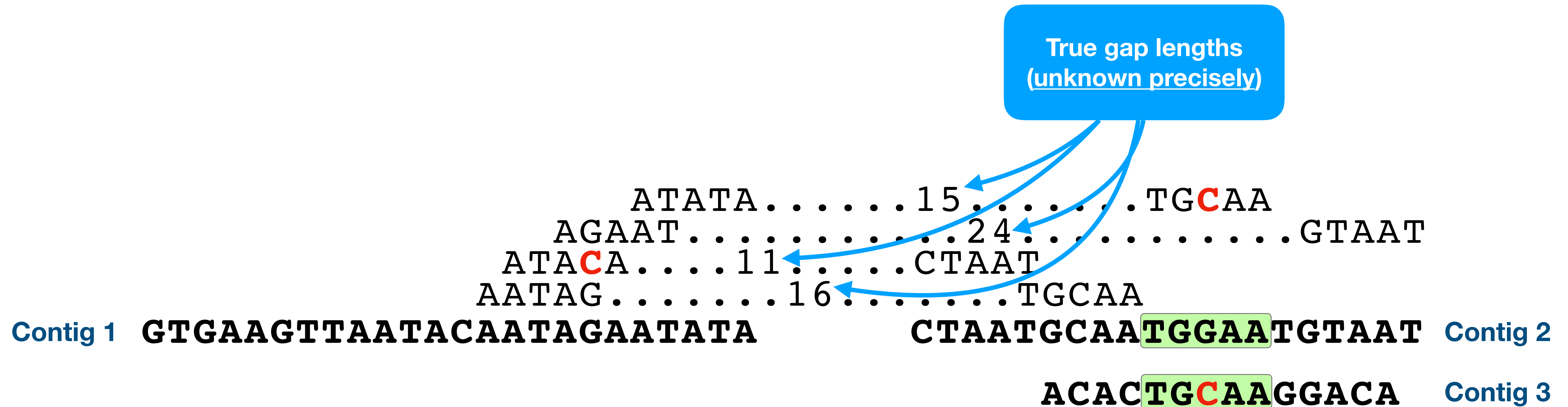**Contig 1** **GTGAAGTTAATACAATAGAATATA** **CTAATGCAATGGAATGTAAT** **Contig 2**
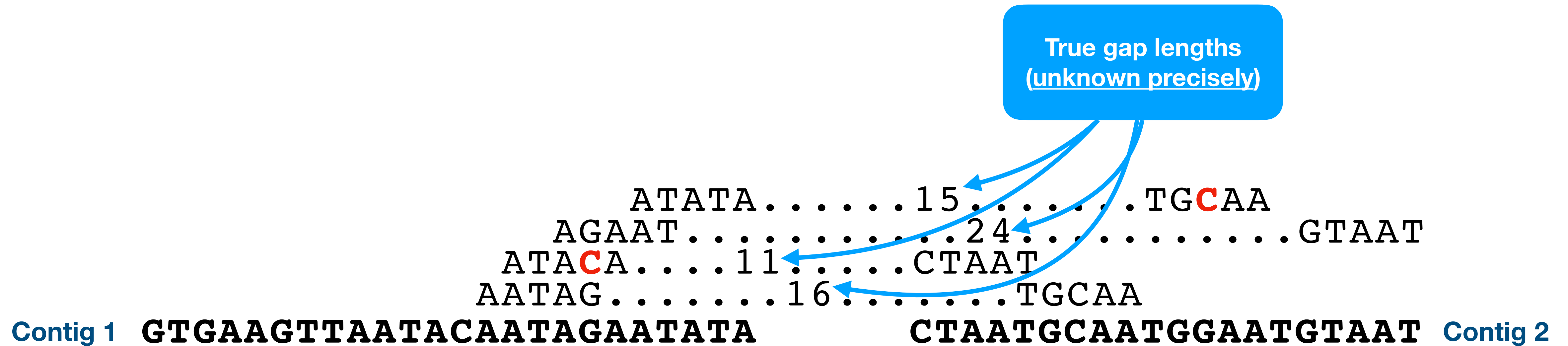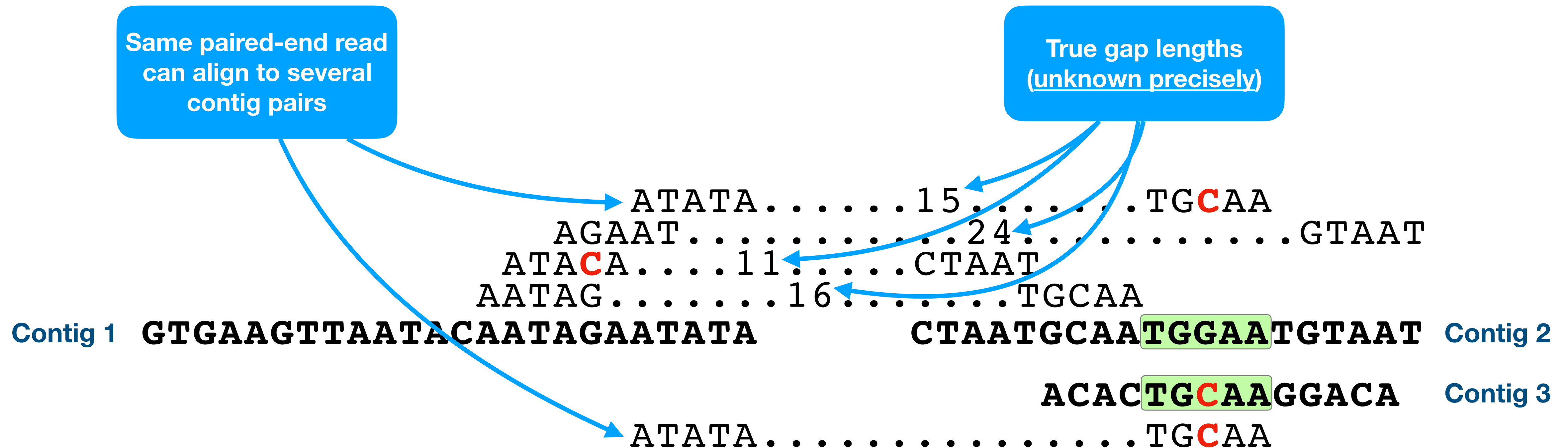
- Align paired-end reads to contigs, focus on read pairs aligning to different contigs
  - ‣ Even if we know two contigs are "consecutive" in a genome, it is not easy to estimate the gap length between them, see e.g.

Align paired-end reads to contigs

True gap lengths
(unknown precisely)

```
ATATA......15......TGCAA
AGAAT.........24..........GTAAT
ATACA....11.....CTAAT
AATAG.......16.......TGCAA
```

Contig 1 **GTGAAGTTAATACAATAGAATATA**   **CTAATGCAATGGAATGTAAT** Contig 2

- Align paired-end reads to contigs, focus on read pairs aligning to different contigs
  - ‣ Even if we know two contigs are "consecutive" in a genome, it is not easy to estimate the gap length between them, see e.g.
  - ‣ Kristoffer Sahlin, Nathaniel Street, Joakim Lundeberg, Lars Arvestad:
    Improved gap size estimation for scaffolding algorithms. *Bioinformatics* 28(17): 2215-2222 (2012)

Align paired-end reads to contigs

True gap lengths (unknown precisely)

```
ATATA......15........TGCAA
AGAAT.............24...............GTAAT
ATACA....11.......CTAAT
AATAG.......16.........TGCAA
```

Contig 1 **GTGAAGTTAATACAATAGAATATA**   **CTAATGCAATGGAATGTAAT** **Contig 2**

**ACACTGCAAGGACA** **Contig 3**

- Align paired-end reads to contigs, focus on read pairs aligning to different contigs
  - ‣ Even if we know two contigs are "consecutive" in a genome, it is not easy to estimate the gap length between them, see e.g.
  - ‣ Kristoffer Sahlin, Nathaniel Street, Joakim Lundeberg, Lars Arvestad:
    Improved gap size estimation for scaffolding algorithms. *Bioinformatics* 28(17): 2215-2222 (2012)

Align paired-end reads to contigs

True gap lengths (unknown precisely)

```
             ATATA......15.......TGCAA
             AGAAT..............24..............GTAAT
         ATACA....11......CTAAT
             AATAG........16........TGCAA
```

Contig 1 **GTGAAGTTAATACAATAGAATATA**   **CTAATGCAATGGAATGTAAT** **Contig 2**

ACACTGCAAGGACA **Contig 3**

- Align paired-end reads to contigs, focus on read pairs aligning to different contigs

  ‣ Even if we know two contigs are "consecutive" in a genome, it is not easy to estimate the gap length between them, see e.g.

  ‣ Kristoffer Sahlin, Nathaniel Street, Joakim Lundeberg, Lars Arvestad: **Improved gap size estimation for scaffolding algorithms**. *Bioinformatics* 28(17): 2215-2222 (2012)

- Read pairs can align to several contig pairs: which pairs are the consecutive ones?

Align paired-end reads to contigs

Same paired-end read can align to several contig pairs

True gap lengths (<u>unknown precisely</u>)

```
              ATATA......15........TGCAA
              AGAAT...........24..............GTAAT
            ATACA....11......CTAAT
            AATAG........16.........TGCAA
Contig 1  GTGAAGTTAATACAATAGAATATA        CTAATGCAATGGAATGTAAT  Contig 2
                                           ACACTGCAAGGACA  Contig 3
              ATATA.................TGCAA
```

- Align paired-end reads to contigs, focus on read pairs aligning to different contigs
  - ‣ Even if we know two contigs are "consecutive" in a genome, it is not easy to estimate the gap length between them, see e.g.
  - ‣ Kristoffer Sahlin, Nathaniel Street, Joakim Lundeberg, Lars Arvestad:
    Improved gap size estimation for scaffolding algorithms. *Bioinformatics* 28(17): 2215-2222 (2012)
- Read pairs can align to several contig pairs: which pairs are the consecutive ones?

Chaining (scaffolding) contigs
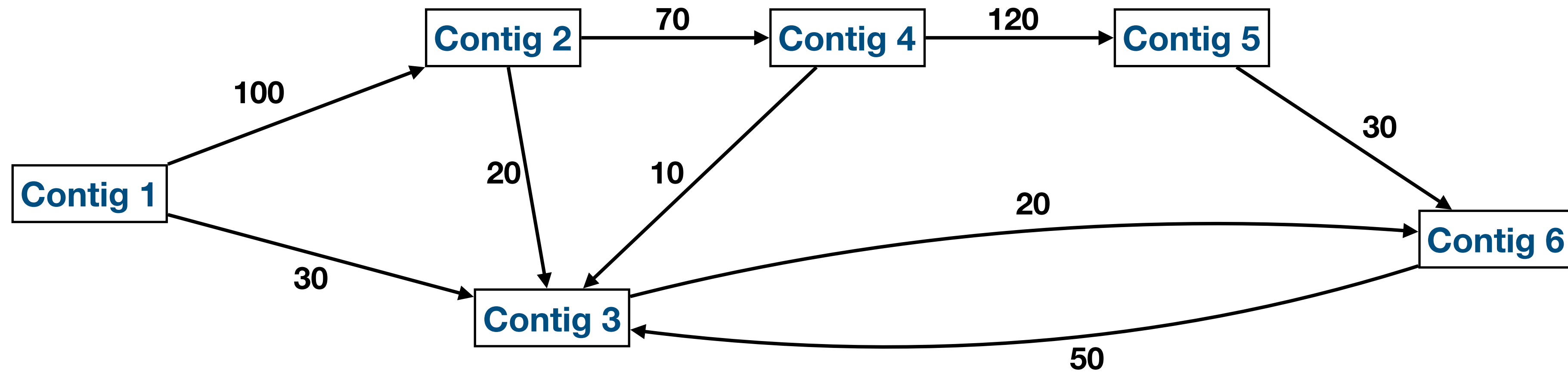
Chaining (scaffolding) contigs
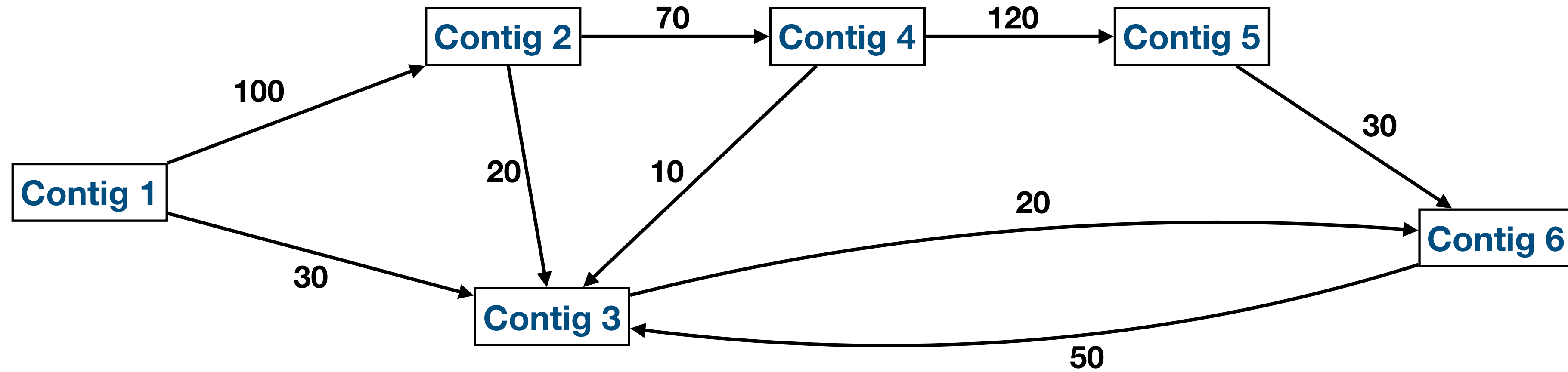
- Another contig assembly-like problem

Chaining (scaffolding) contigs
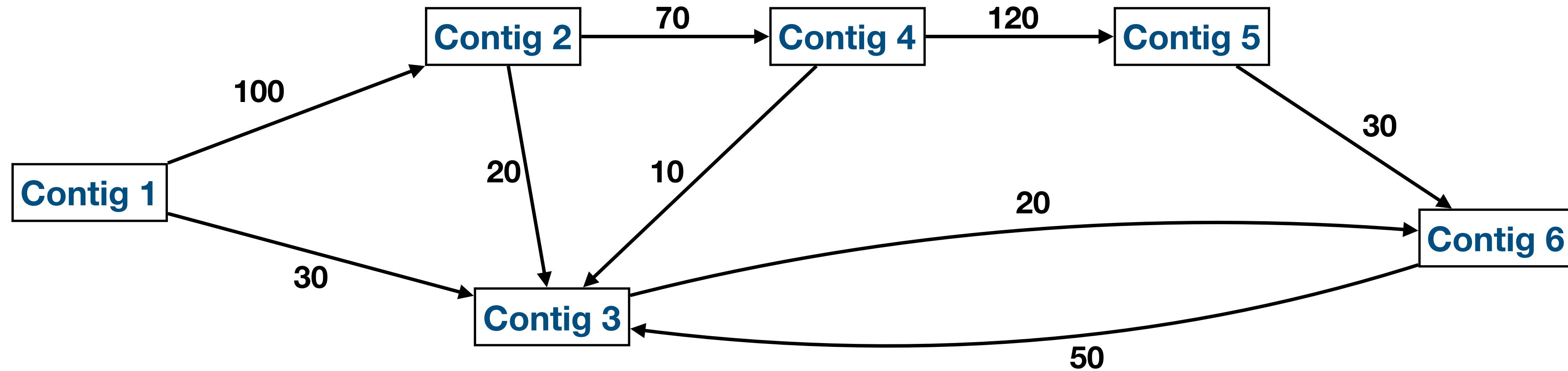
- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations
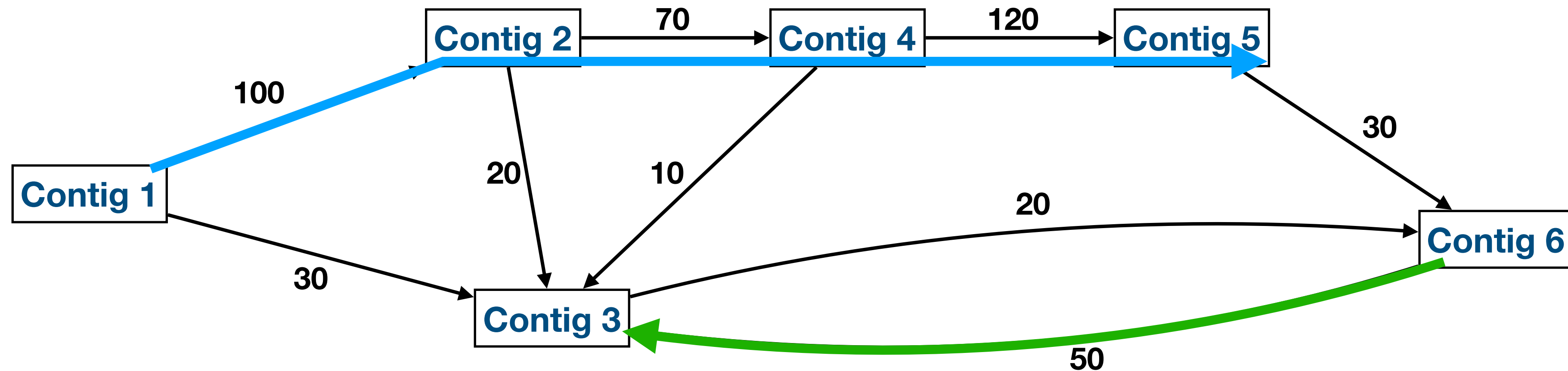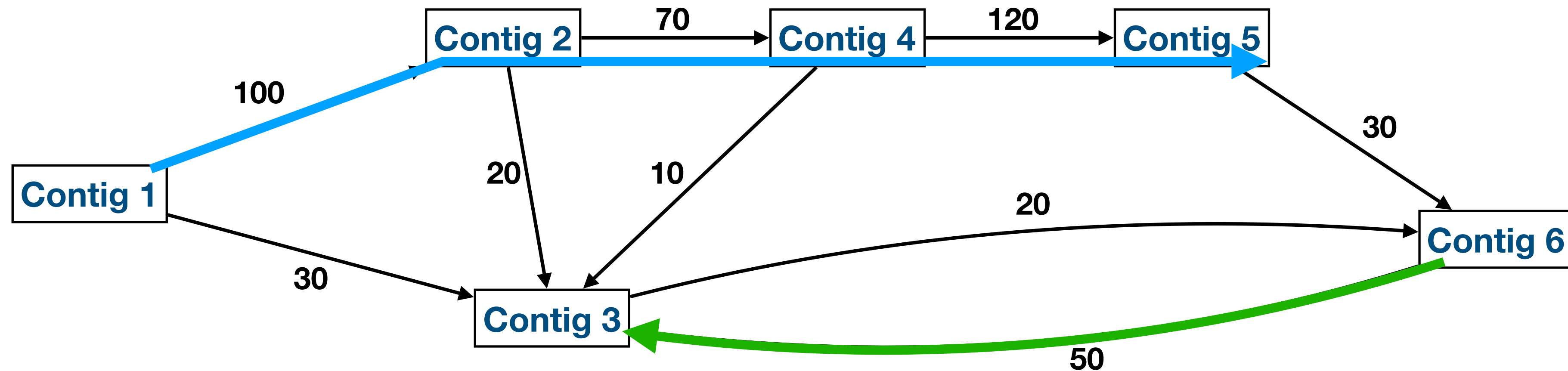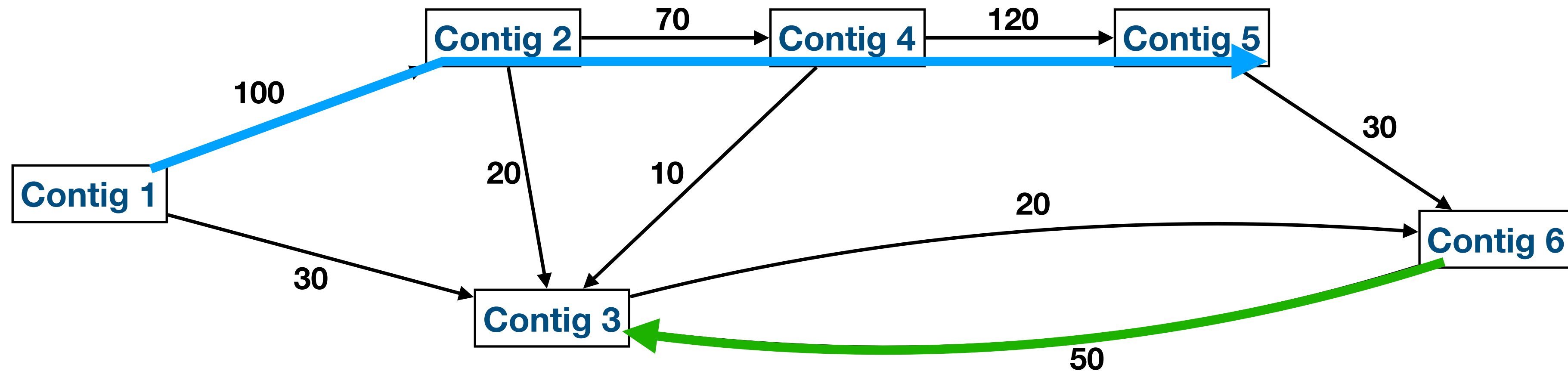
Chaining (scaffolding) contigs
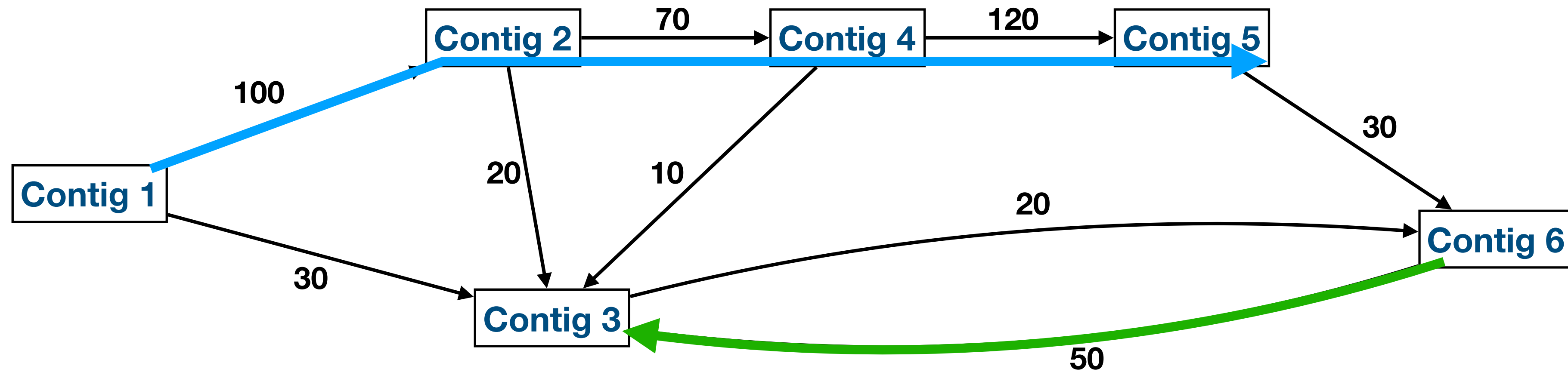
- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations
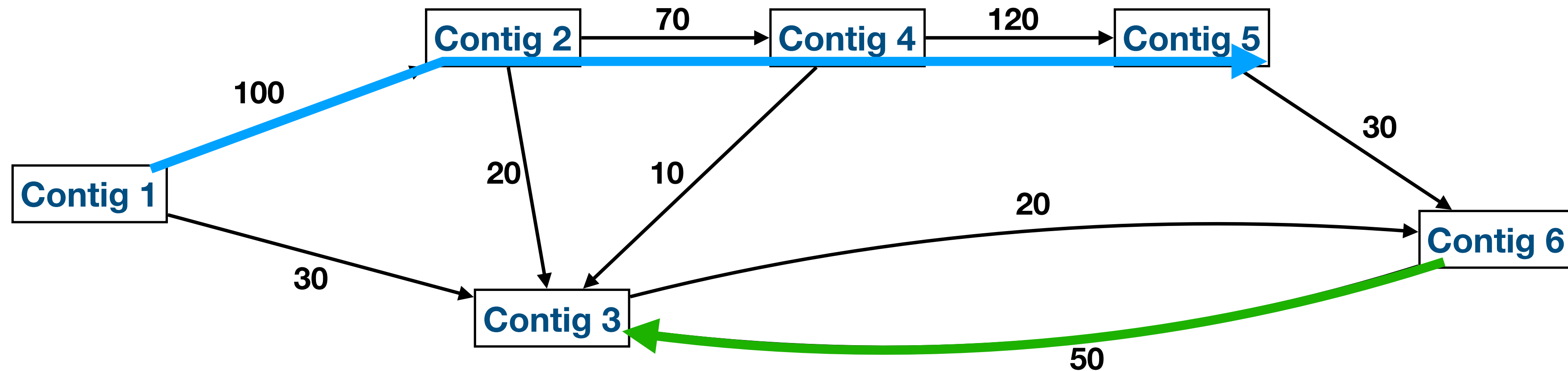
Chaining (scaffolding) contigs
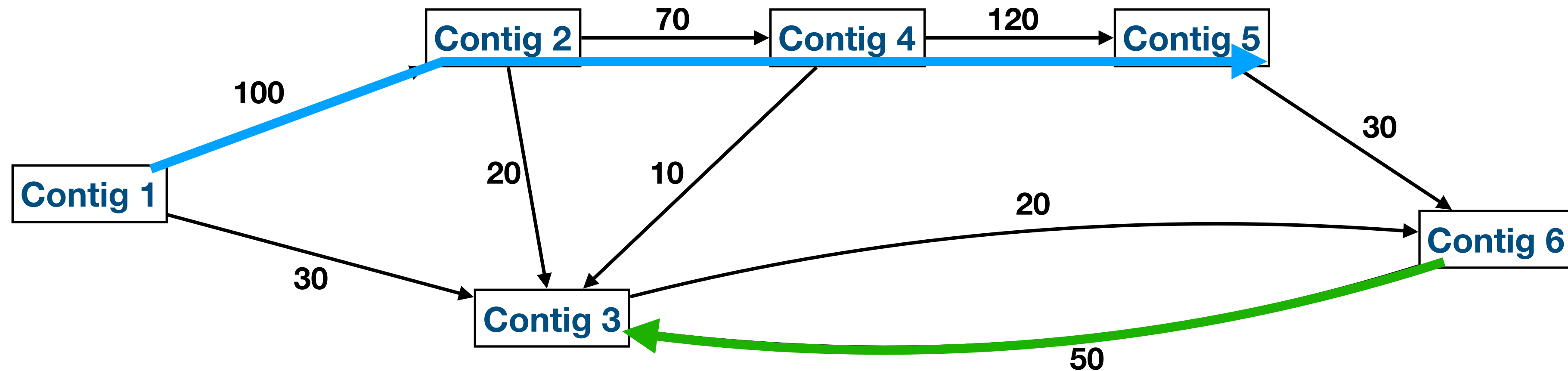
- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations

- Weights not trivial (just #aligned read pairs not enough):

Chaining (scaffolding) contigs
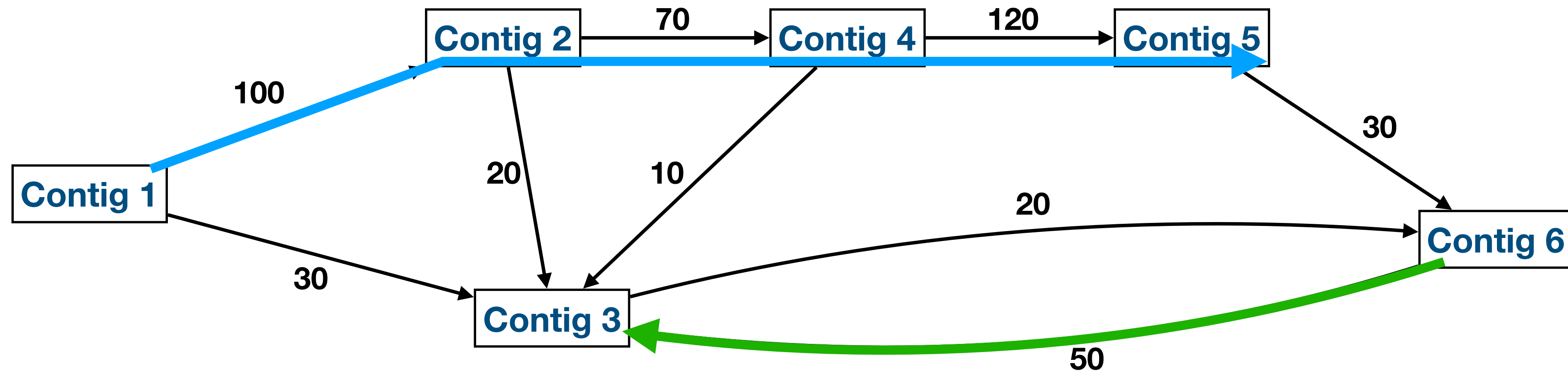
- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations

- Weights not trivial (just #aligned read pairs not enough):
  ‣ If gap is short and contigs are long: many aligned pairs

Chaining (scaffolding) contigs

- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations

- Weights not trivial (just #aligned read pairs not enough):
  - If gap is short and contigs are long: many aligned pairs
  - If gap is long and/or contigs are short: few aligned pairs

Chaining (scaffolding) contigs

- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations

- Weights not trivial (just #aligned read pairs not enough):
  - If gap is short and contigs are long: many aligned pairs
  - If gap is long and/or contigs are short: few aligned pairs

- Some formulations NP-hard, some polynomially-time solvable, see e.g.

Chaining (scaffolding) contigs

- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations

- Weights not trivial (just #aligned read pairs not enough):
  - If gap is short and contigs are long: many aligned pairs
  - If gap is long and/or contigs are short: few aligned pairs

- Some formulations NP-hard, some polynomially-time solvable, see e.g.
  - Leena Salmela, Veli Mäkinen, Niko Välimäki, Johannes Ylinen, Esko Ukkonen:
    Fast scaffolding with small independent mixed integer programs. Bioinformatics 27(23): 3259-3265 (2011)

Chaining (scaffolding) contigs

- Another contig assembly-like problem

- Now weights on edges (how much "evidence" there is): new problem formulations

- Weights not trivial (just #aligned read pairs not enough):
  ‣ If gap is short and contigs are long: many aligned pairs
  ‣ If gap is long and/or contigs are short: few aligned pairs

- Some formulations NP-hard, some polynomially-time solvable, see e.g.
  ‣ Leena Salmela, Veli Mäkinen, Niko Välimäki, Johannes Ylinen, Esko Ukkonen:
    Fast scaffolding with small independent mixed integer programs. Bioinformatics 27(23): 3259-3265 (2011)
  ‣ Igor Mandric, Alex Zelikovsky:
    ScaffMatch: Scaffolding Algorithm Based on Maximum Weight Matching. RECOMB 2015: 222-223

Gap filling

Scaffolds contain gap length estimates (number of Ns)

Bring back all reads

→

Find filling paths from the assembly graph

Output the scaffolds in which some gaps are "filled"

TCGATAGCTAAAA**NNNNNNNNNN**AATTGT**NNN**ATAGAGATATTT

↓

TCGATAGCTAAAA**TGCCGTTCGG**AATTGT**NNN**ATAGAGATATTT
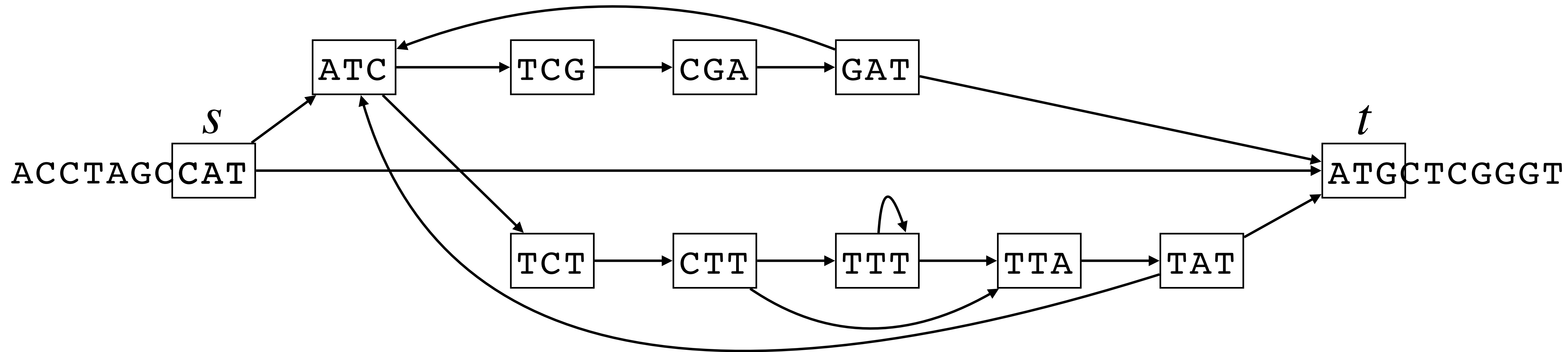
Find path of given length

ACCTAGCCAT

ATGCTCGGGT

Find path of given length
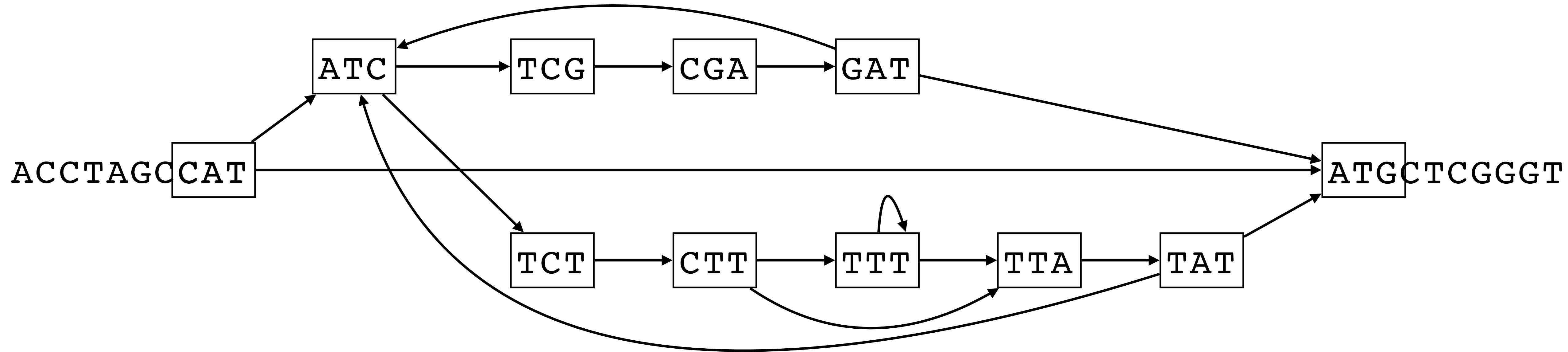
- Build the de Bruijn graph of all reads
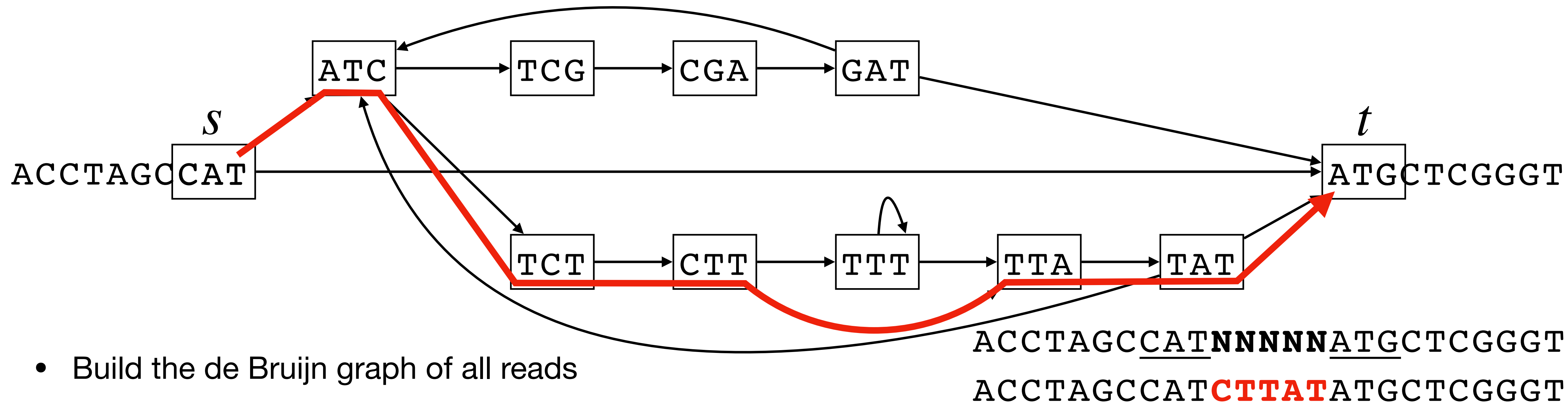
Find path of given length

- Build the de Bruijn graph of all reads

- Take the last k-mer of the 1st contig (node s) and the first k-mer of the second contig (node t)

Find path of given length

- Build the de Bruijn graph of all reads

- Take the last k-mer of the 1st contig (node s) and the first k-mer of the second contig (node t)

- Find an s-t path whose length "matches" or is "close enough" to the gap length estimate
  - ▸ If gap length estimate is d, how long should be the path? **ASSIGNMENT**
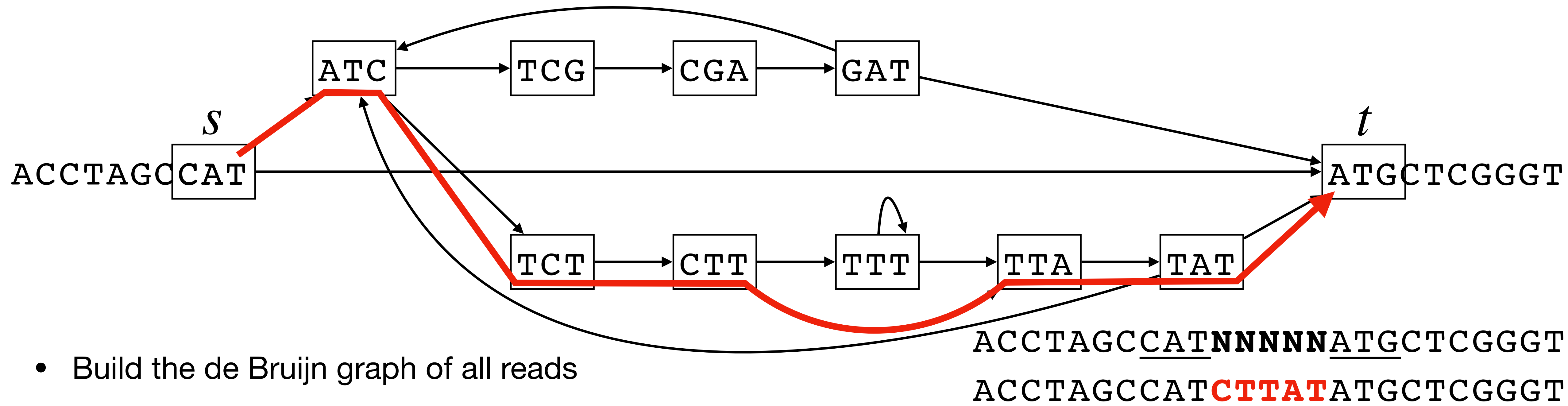
Find path of given length

- Build the de Bruijn graph of all reads

- Take the last k-mer of the 1st contig (node s) and the first k-mer of the second contig (node t)

- Find an s-t path whose length "matches" or is "close enough" to the gap length estimate
  - If gap length estimate is d, how long should be the path? **ASSIGNMENT**

Find path of given length

ACCTAGCCAT**NNNNN**ATGCTCGGGT
ACCTAGCCAT**CTTAT**ATGCTCGGGT

- Build the de Bruijn graph of all reads

- Take the last k-mer of the 1st contig (node s) and the first k-mer of the second contig (node t)

- Find an s-t path whose length "matches" or is "close enough" to the gap length estimate

  ‣ If gap length estimate is d, how long should be the path? ASSIGNMENT

Find path of given length

ACCTAGCCAT**NNNNN**ATGCTCGGGT
ACCTAGCCAT**CTTAT**ATGCTCGGGT

- Build the de Bruijn graph of all reads

- Take the last k-mer of the 1st contig (node s) and the first k-mer of the second contig (node t)

- Find an s-t path whose length "matches" or is "close enough" to the gap length estimate
  - If gap length estimate is d, how long should be the path? **ASSIGNMENT**

- Can be solved by dynamic programming in time $O(d|\text{edges}|)$
  - Leena Salmela, Kristoffer Sahlin, Veli Mäkinen, Alexandru I. Tomescu:
    Gap Filling as Exact Path Length Problem. RECOMB 2015: 281-292

Section summary

Contig assembly

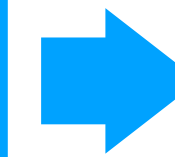Forget about the assembly model (i.e. the problem formulation)

Assemble only parts about which we are sure (contigs → **contig**uous sequences)
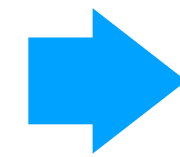
Scaffolding

Chain (order) the contigs

Output chains of contigs with "gaps" (`NNNN...`) between them

**Gap filling**

Find filling paths from the assembly graph

Output the scaffolds in which some gaps are "filled"

- A natural decomposition into subproblems based on the available paired-end information

- One can improve each step individually, thus improving the overall result

Long-read sequencing
(Third-generation sequencing)

- No paired-end reads (focus is on contig assembly)

- Higher error rate: 15% compared to 0.1% for short reads

  ‣ Still developing: accurate PacBio HiFi reads

- No "clear" best strategy

- Short reads still relevant for some scenarios (e.g. metagenomic sequencing)

Picture: Leena Salmela
https://www.cs.helsinki.fi/u/lmsalmel/
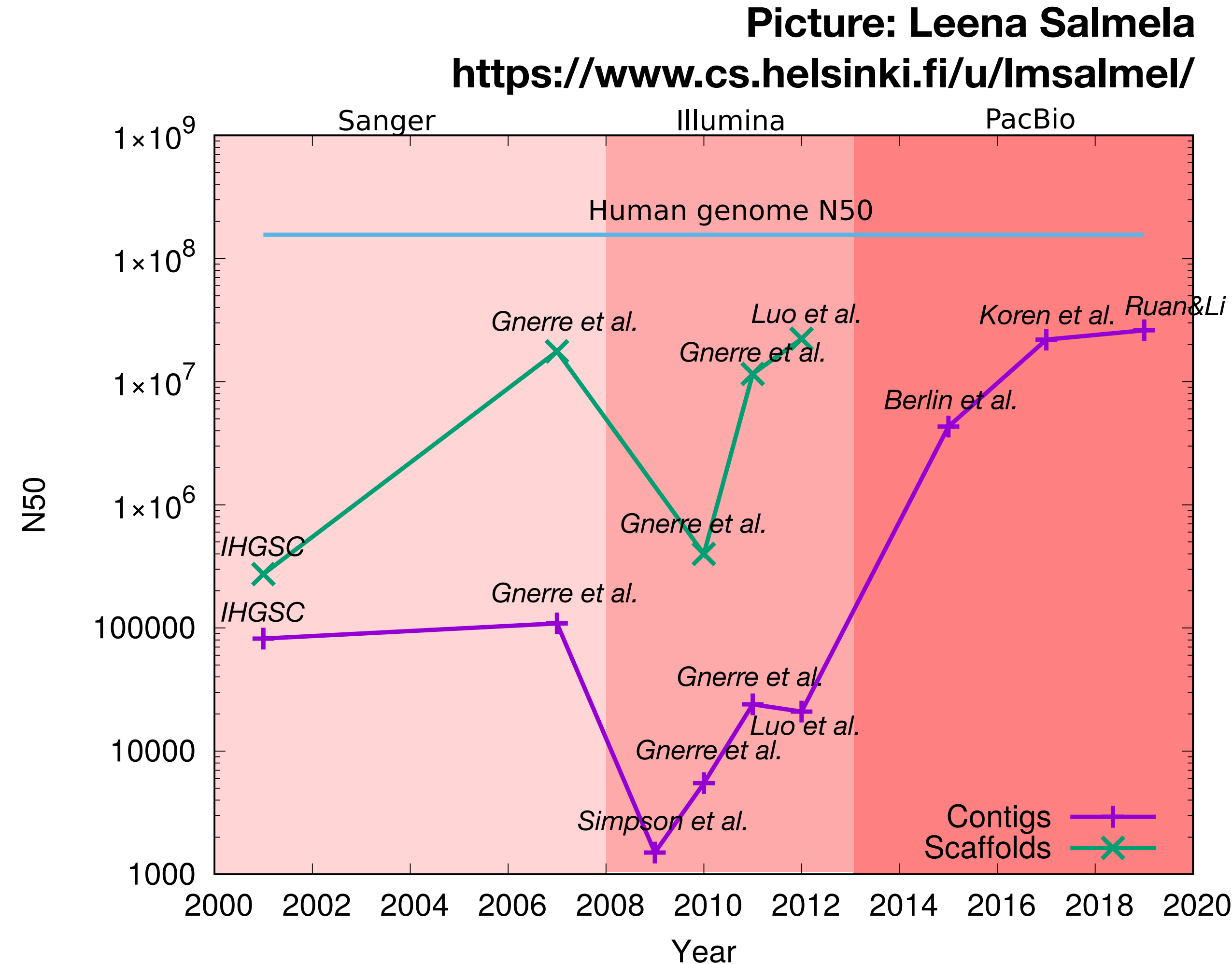
N50 measure → ASSIGNMENT

A more "practical" theoretical formulation

(A principled approach to contig assembly)

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome
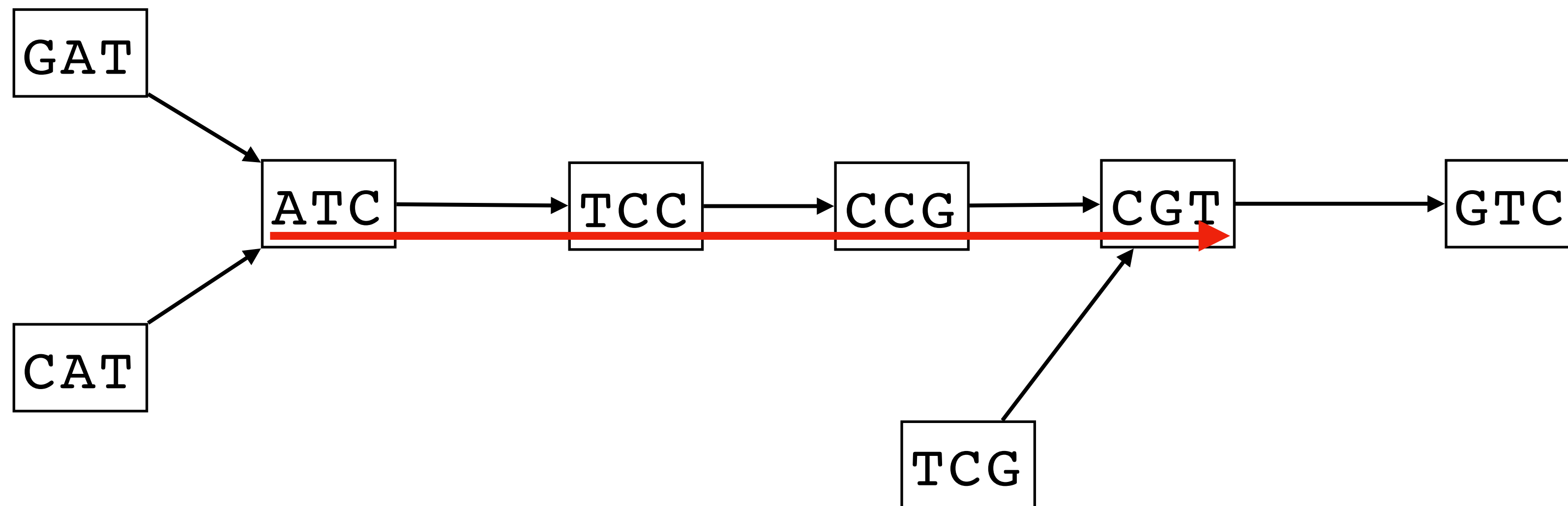
Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome
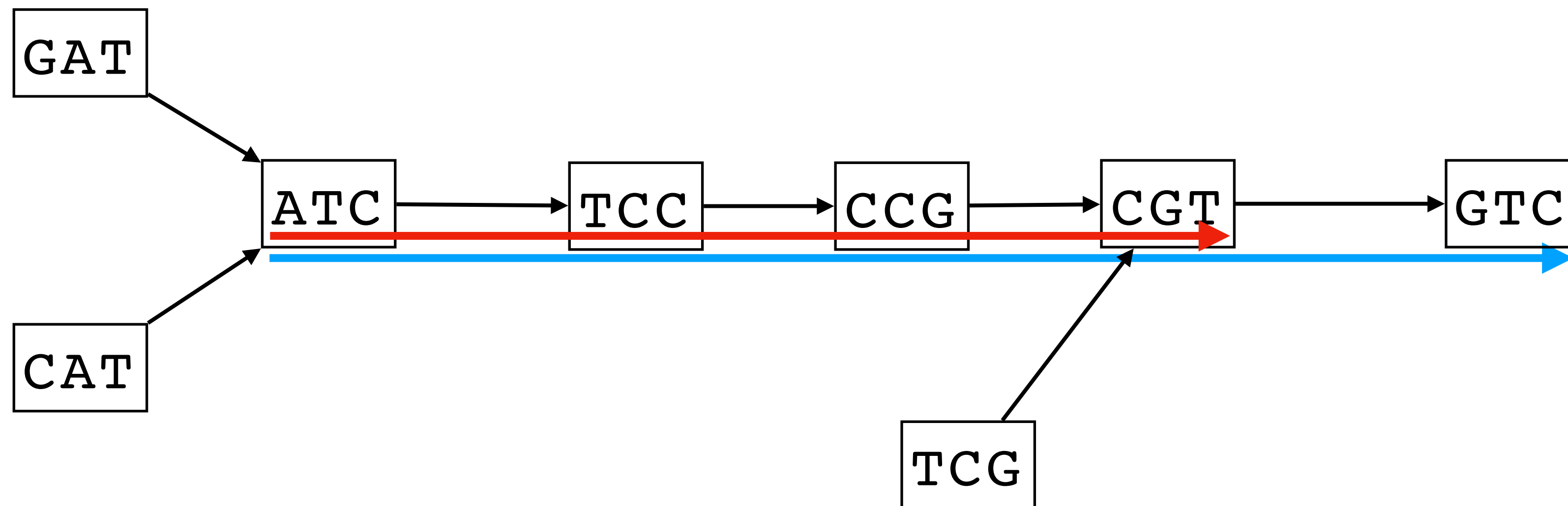
- Why only unitigs?

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

- Retake: assemble paths whose internal nodes have **out-degree = 1**
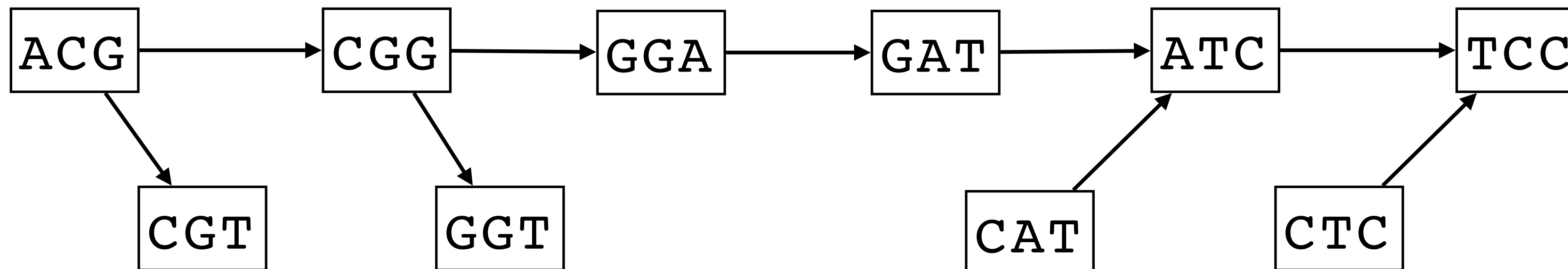
Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

- Retake: assemble paths whose internal nodes have **out-degree = 1**

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

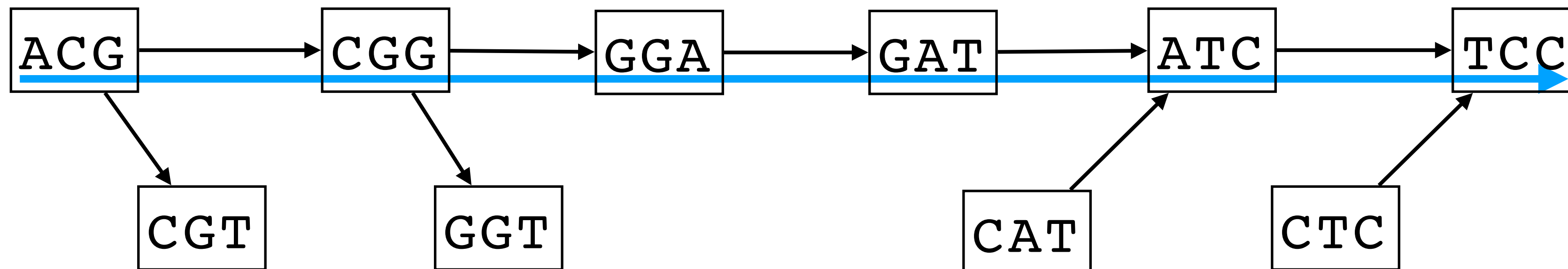- Retake: assemble paths whose internal nodes have **out-degree = 1**

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

- Retake: assemble paths whose internal nodes have **out-degree = 1**

- Is there something more to assemble?

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

- Retake: assemble paths whose internal nodes have **out-degree = 1**

- Is there something more to assemble?

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

- Retake: assemble paths whose internal nodes have **out-degree = 1**

- Is there something more to assemble?

Back to contig assembly

- Goal: obtain sequences that are "guaranteed" to occur in the genome

- Why only unitigs?

- Retake: assemble paths whose internal nodes have **out-degree = 1**

- Is there something more to assemble?

- Is there something more to assemble?
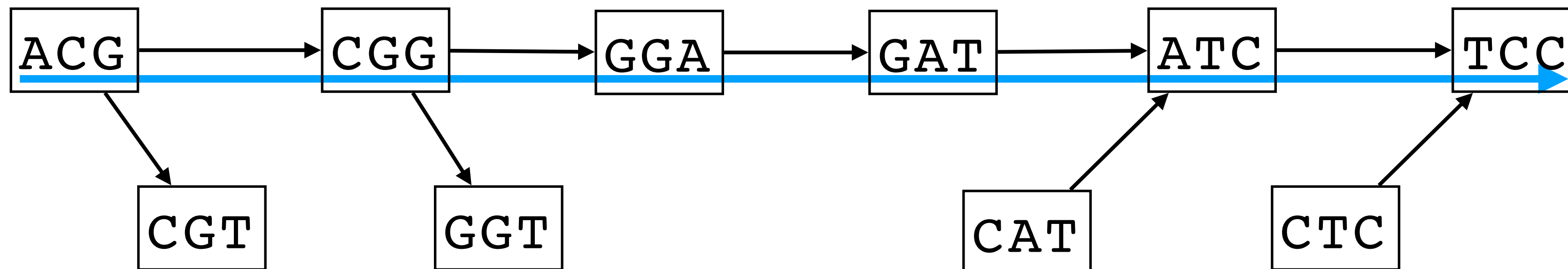
Omnitigs

- Assume that the "genome assembly solution" is a circular walk covering every edge at least once (walk can repeat nodes)
  - ‣ Trivial to find one, exponential to find all
  - ‣ Makes sense for single circular chromosomes (i.e. most bacteria), full coverage, no errors

Omnitigs

- Assume that the "genome assembly solution" is a circular walk covering every edge at least once (walk can repeat nodes)

  ‣ Trivial to find one, exponential to find all

  ‣ Makes sense for single circular chromosomes (i.e. most bacteria), full coverage, no errors

- **Omnitig** $=_{def}$ a walk common to **all** "genome assembly solutions"

 ‣ Omnitigs are **all** that can be correctly assembled

Omnitigs

- Assume that the "genome assembly solution" is a circular walk covering every edge at least once (walk can repeat nodes)

  ‣ Trivial to find one, exponential to find all

  ‣ Makes sense for single circular chromosomes (i.e. most bacteria), full coverage, no errors

- **Omnitig** $=_{def}$ a walk common to **all** "genome assembly solutions"

  ‣ Omnitigs are **all** that can be correctly assembled

- Omnitigs can be found efficiently

  ‣ Alexandru I. Tomescu, Paul Medvedev:
  Safe and Complete Contig Assembly Via Omnitigs. RECOMB 2016: 152-163

  ‣ Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, Alexandru I. Tomescu:
  An Optimal *O(nm)* Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. ACM Trans. Algorithms 15(4): 48:1-48:17 (2019)
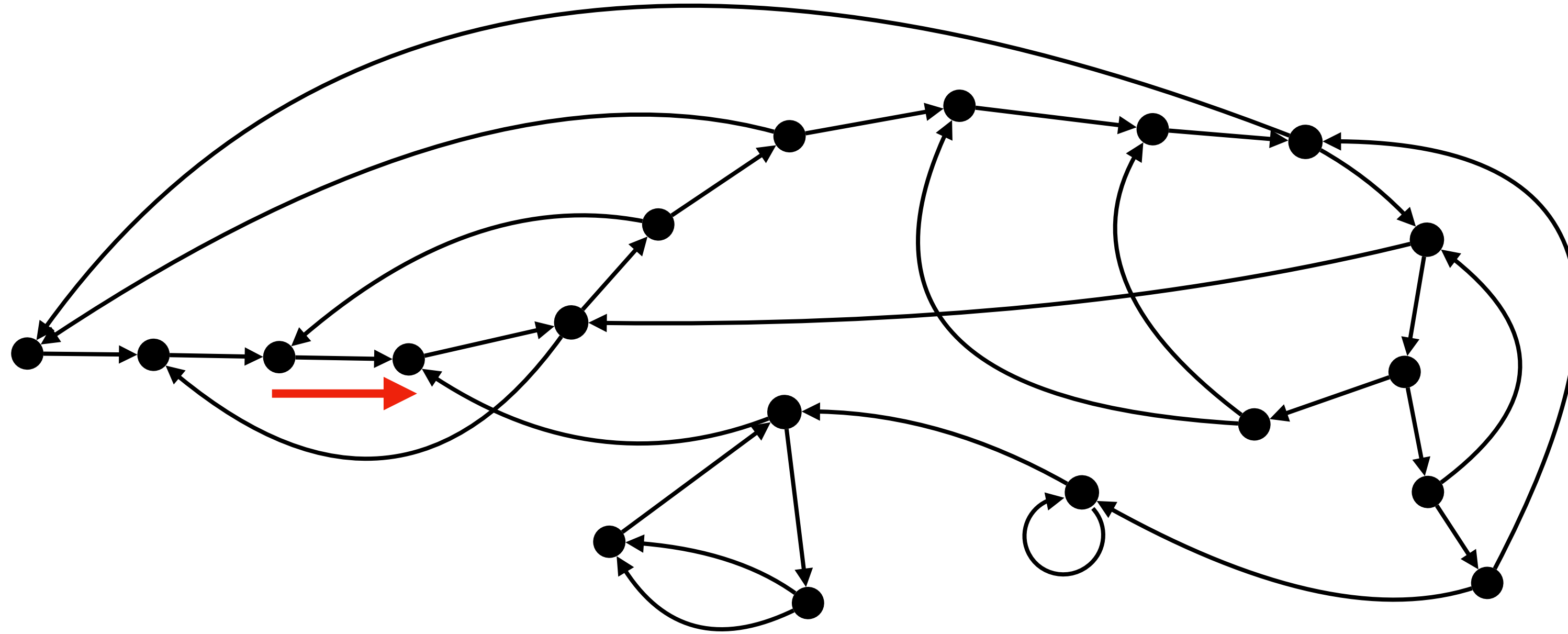
Omnitigs

- Assume that the "genome assembly solution" is a circular walk covering every edge at least once (walk can repeat nodes)

  ‣ Trivial to find one, exponential to find all

  ‣ Makes sense for single circular chromosomes (i.e. most bacteria), full coverage, no errors

- **Omnitig** $=_{\text{def}}$ a walk common to **all** "genome assembly solutions"

  ‣ Omnitigs are **all** that can be correctly assembled

- Omnitigs can be found efficiently

  ‣ Alexandru I. Tomescu, Paul Medvedev:
    Safe and Complete Contig Assembly Via Omnitigs. RECOMB 2016: 152-163

  ‣ Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, Alexandru I. Tomescu:
    An Optimal *O(nm)* Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. ACM Trans. Algorithms 15(4): 48:1-48:17 (2019)

- Can be adapted to deal with practical issues (ongoing work)
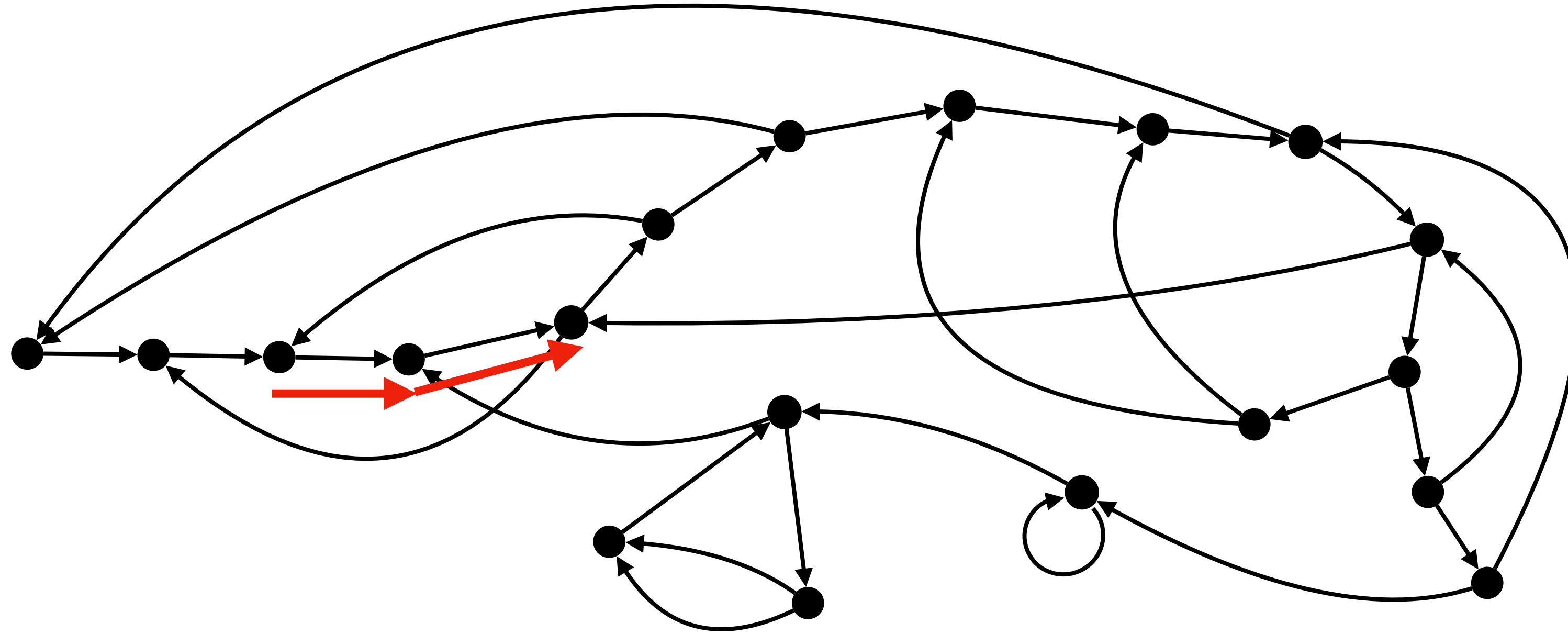
  ‣ **<u>Subprojects available as Master thesis topics</u>**
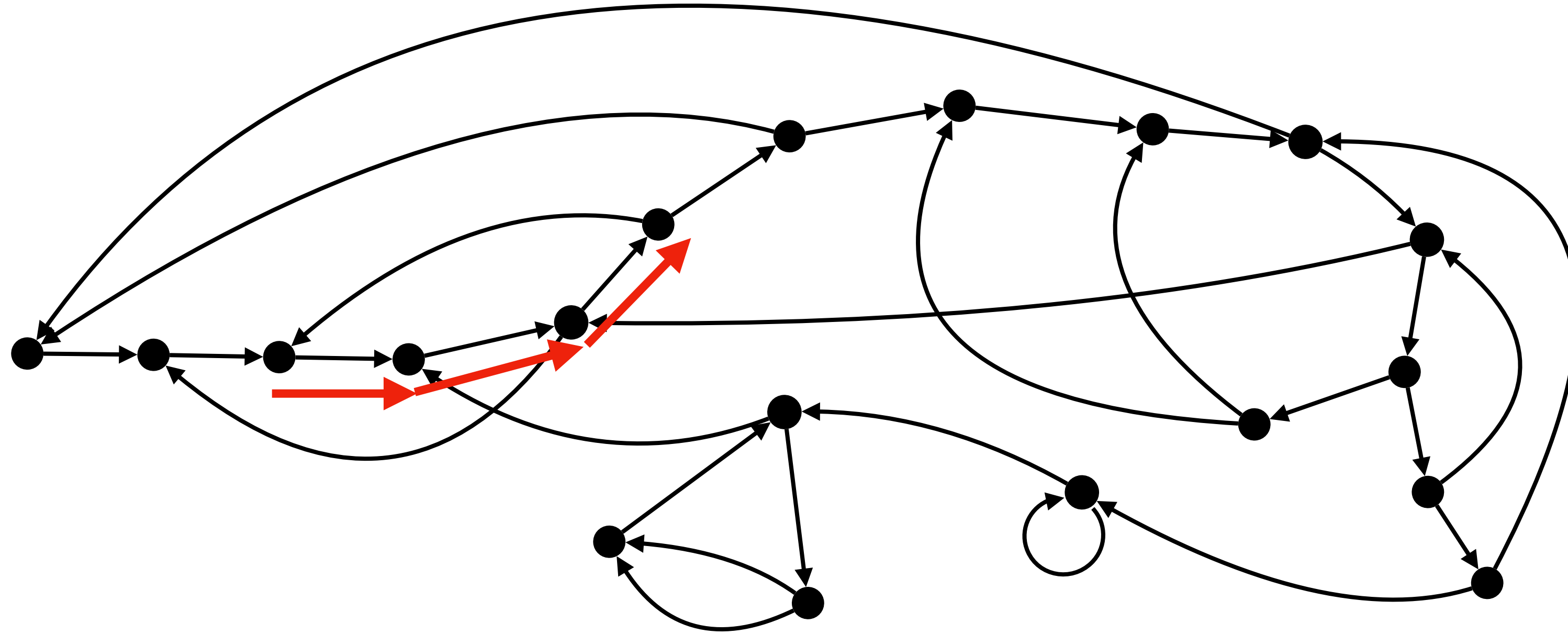
Example + Results on perfect data
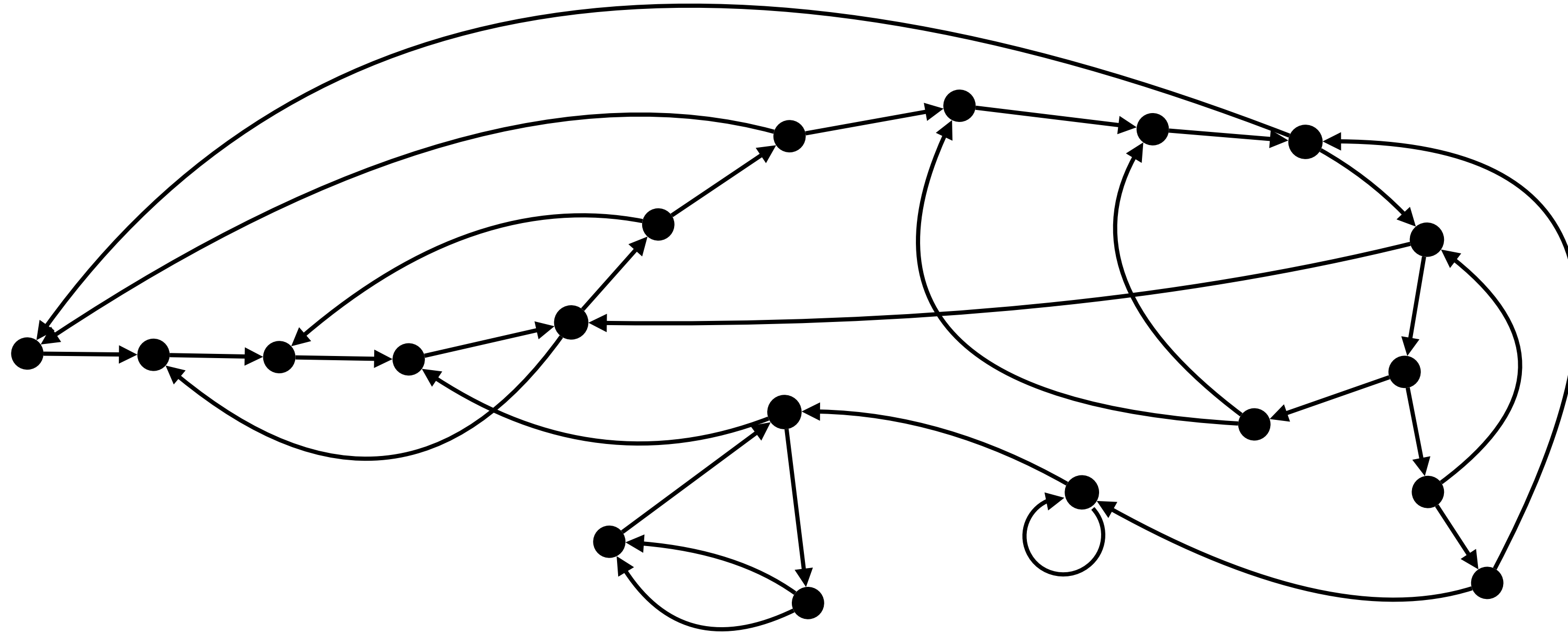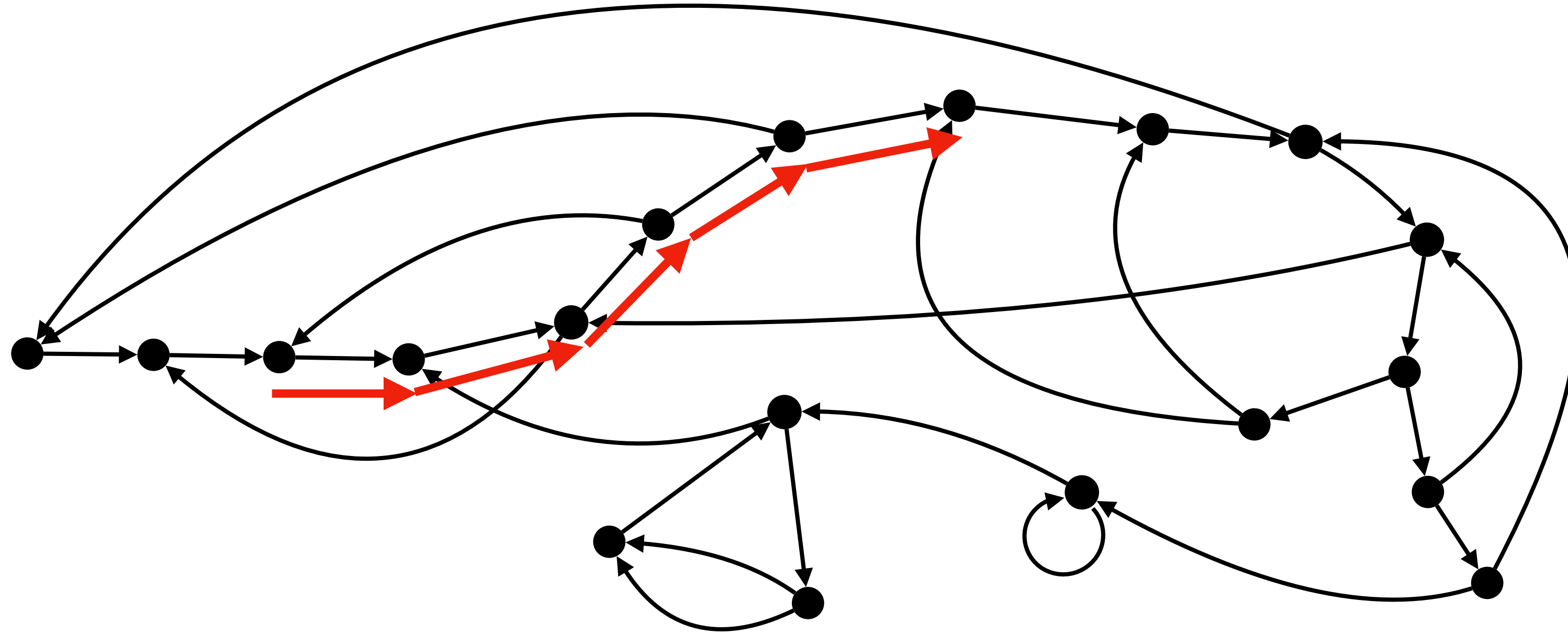
Example + Results on perfect data

Example + Results on perfect data

Example + Results on perfect data

Example + Results on perfect data

Example + Results on perfect data
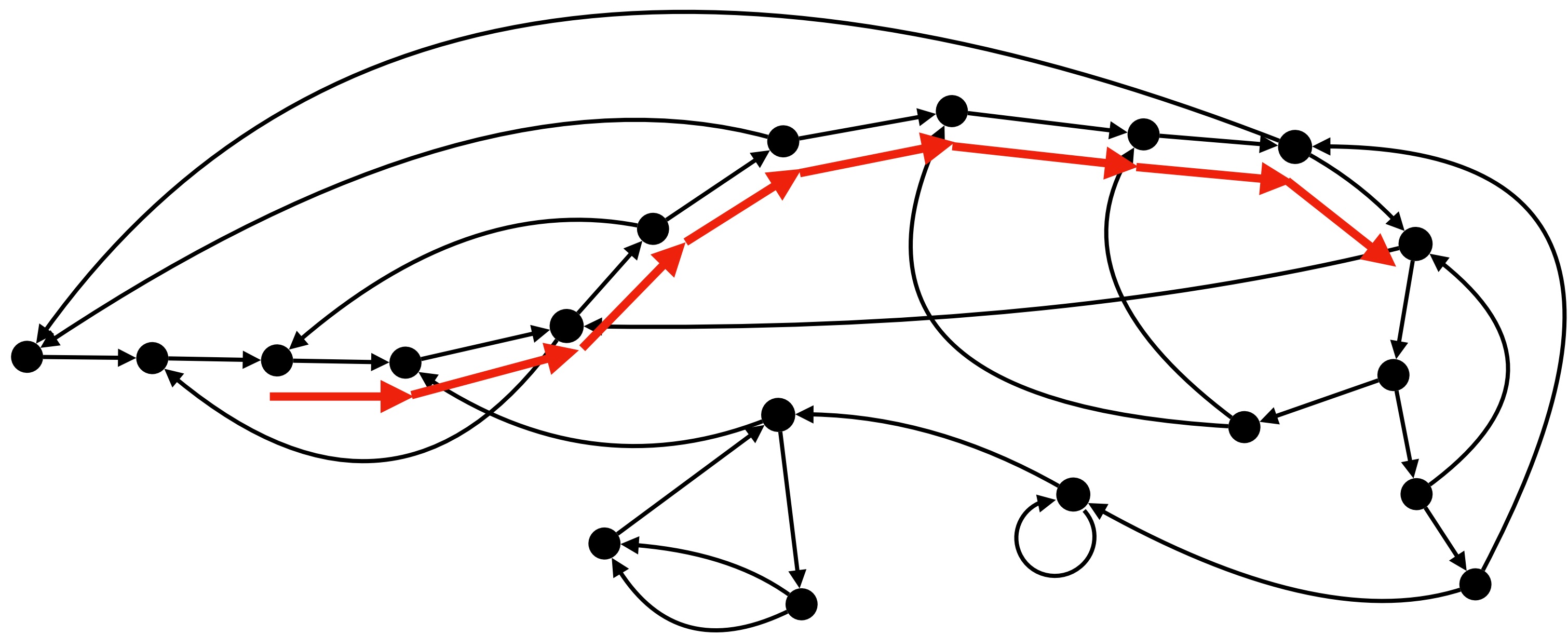
Example + Results on perfect data

Example + Results on perfect data

Example + Results on perfect data

Example + Results on perfect data

chr10, length 135M	#Strings	Avg. length	Avg. #SNPs / string
unitigs	260K	546	26
omnitigs	158K (-40%)	887 (+62%)	41 (+58%)

Section summary

Theory is important,
but more so when
it is motivated by practice