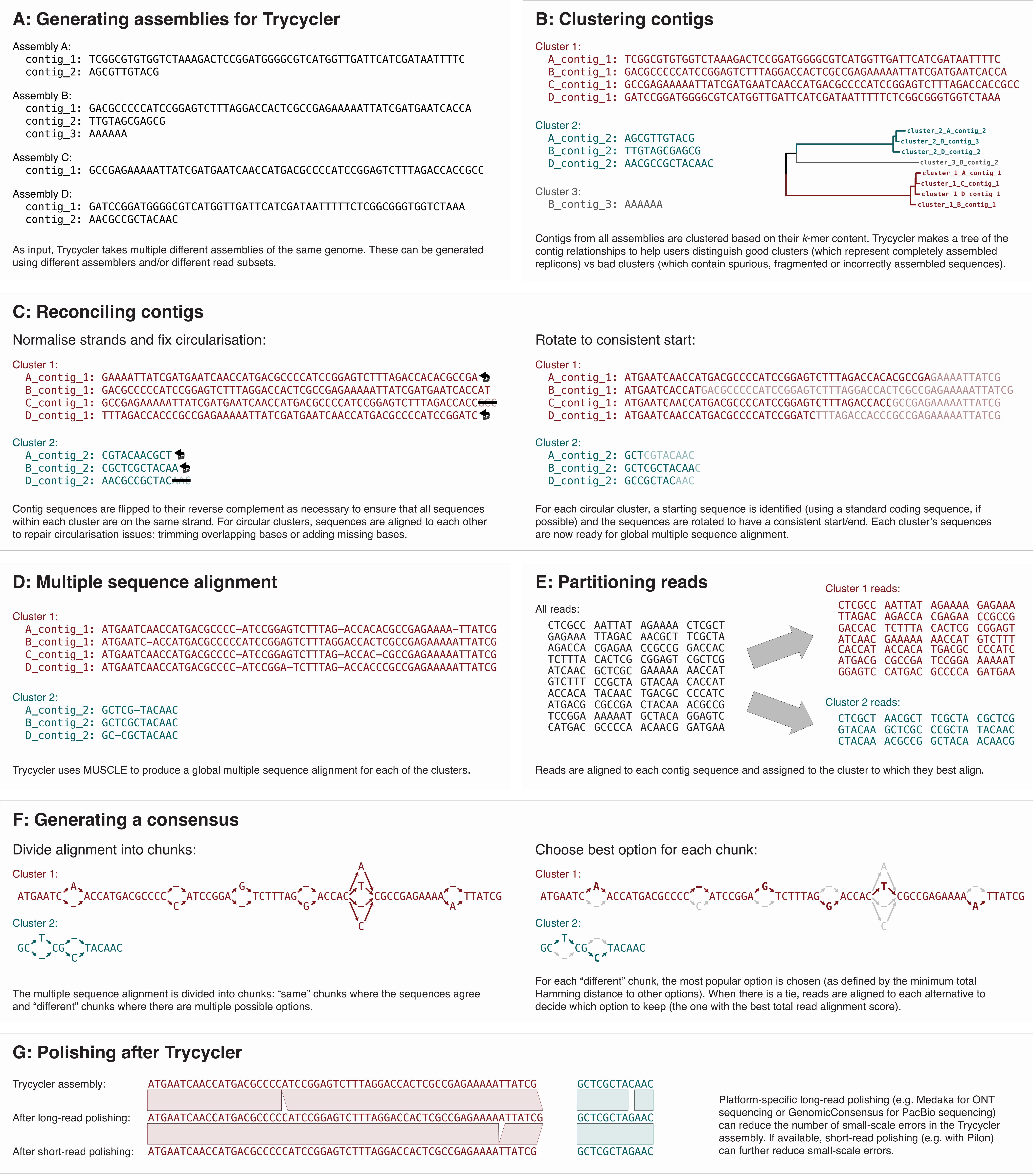

A: Generating assemblies for Trycycler

Assembly A:
```
  contig_1: TCGGCGTGTGGTCTAAAGACTCCGGATGGGGCGTCATGGTTGATTCATCGATAATTTTC
  contig_2: AGCGTTGTACG
```

Assembly B:
```
  contig_1: GACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCGATGAATCACCA
  contig_2: TTGTAGCGAGCG
  contig_3: AAAAAA
```

Assembly C:
```
  contig_1: GCCGAGAAAAATTATCGATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGACCACCGCC
```

Assembly D:
```
  contig_1: GATCCGGATGGGGCGTCATGGTTGATTCATCGATAATTTTTCTCGGCGGGTGGTCTAAA
  contig_2: AACGCCGCTACAAC
```

As input, Trycycler takes multiple different assemblies of the same genome. These can be generated using different assemblers and/or different read subsets.

B: Clustering contigs

Cluster 1:
```
  A_contig_1: TCGGCGTGTGGTCTAAAGACTCCGGATGGGGCGTCATGGTTGATTCATCGATAATTTTC
  B_contig_1: GACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCGATGAATCACCA
  C_contig_1: GCCGAGAAAAATTATCGATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGACCACCGCC
  D_contig_1: GATCCGGATGGGGCGTCATGGTTGATTCATCGATAATTTTTCTCGGCGGGTGGTCTAAA
```

Cluster 2:
```
  A_contig_2: AGCGTTGTACG
  B_contig_2: TTGTAGCGAGCG
  D_contig_2: AACGCCGCTACAAC
```

Cluster 3:
```
  B_contig_3: AAAAAA
```

Contigs from all assemblies are clustered based on their *k*-mer content. Trycycler makes a tree of the contig relationships to help users distinguish good clusters (which represent completely assembled replicons) vs bad clusters (which contain spurious, fragmented or incorrectly assembled sequences).

C: Reconciling contigs

Normalise strands and fix circularisation:

Cluster 1:
```
  A_contig_1: GAAAATTATCGATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGACCACACGCCGA
  B_contig_1: GACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCGATGAATCACCAT
  C_contig_1: GCCGAGAAAAATTATCGATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGACCACC~~GCC~~
  D_contig_1: TTTAGACCACCCGCCGAGAAAAATTATCGATGAATCAACCATGACGCCCCATCCGGATC
```

Cluster 2:
```
  A_contig_2: CGTACAACGCT
  B_contig_2: CGCTCGCTACAA
  D_contig_2: AACGCCGCTAC~~AAC~~
```

Contig sequences are flipped to their reverse complement as necessary to ensure that all sequences within each cluster are on the same strand. For circular clusters, sequences are aligned to each other to repair circularisation issues: trimming overlapping bases or adding missing bases.

Rotate to consistent start:

Cluster 1:
```
  A_contig_1: ATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGACCACACGCCGAGAAAATTATCG
  B_contig_1: ATGAATCACCATGACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCG
  C_contig_1: ATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGACCACCGCCGAGAAAAATTATCG
  D_contig_1: ATGAATCAACCATGACGCCCCATCCGGATCTTTAGACCACCCGCCGAGAAAAATTATCG
```

Cluster 2:
```
  A_contig_2: GCTCGTACAAC
  B_contig_2: GCTCGCTACAAC
  D_contig_2: GCCGCTACAAC
```

For each circular cluster, a starting sequence is identified (using a standard coding sequence, if possible) and the sequences are rotated to have a consistent start/end. Each cluster's sequences are now ready for global multiple sequence alignment.

D: Multiple sequence alignment

Cluster 1:
```
  A_contig_1: ATGAATCAACCATGACGCCCC–ATCCGGAGTCTTTAG–ACCACACGCCGAGAAAA–TTATCG
  B_contig_1: ATGAATC–ACCATGACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCG
  C_contig_1: ATGAATCAACCATGACGCCCC–ATCCGGAGTCTTTAG–ACCAC–CGCCGAGAAAAATTATCG
  D_contig_1: ATGAATCAACCATGACGCCCC–ATCCGGA–TCTTTAG–ACCACCCGCCGAGAAAAATTATCG
```

Cluster 2:
```
  A_contig_2: GCTCG–TACAAC
  B_contig_2: GCTCGCTACAAC
  D_contig_2: GC–CGCTACAAC
```

Trycycler uses MUSCLE to produce a global multiple sequence alignment for each of the clusters.

E: Partitioning reads

All reads:
```
  CTCGCC  AATTAT  AGAAAA  CTCGCT
  GAGAAA  TTAGAC  AACGCT  TCGCTA
  AGACCA  CGAGAA  CCGCCG  GACCAC
  TCTTTA  CACTCG  CGGAGT  CGCTCG
  ATCAAC  GCTCGC  GAAAAA  AACCAT
  GTCTTT  CCGCTA  GTACAA  CACCAT
  ACCACA  TACAAC  TGACGC  CCCATC
  ATGACG  CGCCGA  CTACAA  ACGCCG
  TCCGGA  AAAAAT  GCTACA  GGAGTC
  CATGAC  GCCCCA  ACAACG  GATGAA
```

Cluster 1 reads:
```
  CTCGCC  AATTAT  AGAAAA  GAGAAA
  TTAGAC  AGACCA  CGAGAA  CCGCCG
  GACCAC  TCTTTA  CACTCG  CGGAGT
  ATCAAC  GAAAAA  AACCAT  GTCTTT
  CACCAT  ACCACA  TGACGC  CCCATC
  ATGACG  CGCCGA  TCCGGA  AAAAAT
  GGAGTC  CATGAC  GCCCCA  GATGAA
```

Cluster 2 reads:
```
  CTCGCT  AACGCT  TCGCTA  CGCTCG
  GTACAA  GCTCGC  CCGCTA  TACAAC
  CTACAA  ACGCCG  GCTACA  ACAACG
```

Reads are aligned to each contig sequence and assigned to the cluster to which they best align.

F: Generating a consensus

Divide alignment into chunks:

Cluster 1:

ATGAATC A – ACCATGACGCCCC – C ATCCGGA G – TCTTTAG – G ACCAC A T – C CGCCGAGAAAA – A TTATCG

Cluster 2:

The multiple sequence alignment is divided into chunks: "same" chunks where the sequences agree and "different" chunks where there are multiple possible options.

Choose best option for each chunk:

Cluster 1:

ATGAATC A – ACCATGACGCCCC – C ATCCGGA G – TCTTTAG – G ACCAC A T – C CGCCGAGAAAA – A TTATCG

Cluster 2:

For each "different" chunk, the most popular option is chosen (as defined by the minimum total Hamming distance to other options). When there is a tie, reads are aligned to each alternative to decide which option to keep (the one with the best total read alignment score).

G: Polishing after Trycycler

```
Trycycler assembly:           ATGAATCAACCATGACGCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCG     GCTCGCTACAAC
After long-read polishing:    ATGAATCAACCATGACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAAATTATCG    GCTCGCTAGAAC
After short-read polishing:   ATGAATCAACCATGACGCCCCCATCCGGAGTCTTTAGGACCACTCGCCGAGAAAATTATCG     GCTCGCTAGAAC
```

Platform-specific long-read polishing (e.g. Medaka for ONT sequencing or GenomicConsensus for PacBio sequencing) can reduce the number of small-scale errors in the Trycycler assembly. If available, short-read polishing (e.g. with Pilon) can further reduce small-scale errors.

Figure S1: steps in the Trycycler assembly pipeline.