Data Analysis

We used the following datasets:

1 Nanopore E.coli data. The nanopore data used in the analysis are the MARC sample ERA434488, and can be downloaded from: ftp://dcc_marc@ftp.sra.ebi.ac.uk/data/ERA434/ERA434488/oxfordnanopore_native/UCSC_MARC_Phase_Ia_Run_2.tar.gz. The 2D reads have been extracted from the fast5 files in the 'pass' directory using Poretools (https://github.com/arq5x/poretools), and they have 48X coverage. In order to test the assemblers, we created a subsample with 20X coverage using randomly selected reads.

2 PacificBiosciences E.coli data. We compared the results of the various assemblers when using Nanopore or PacificBiosciences (PacBio) datasets.
For this task, we used the PacBio fasta file ecoli_p6_25x.filtered.fasta provided as test sample by the PBcR assembler authors and availale here: http://www.ncbi.nlm.nih.gov/sra/SRX533603. We run the assemblers on the entire dataset, which has about 25X coverage.For a better comparison with the Nanopore low coverage dataset (20X) we also run the assemblers on a PacBio subsample with 20X coverage.

For the ONT and PacBio data, we provide an alternative place for data downloading:

ftp://ngs.sanger.ac.uk/scratch/project/zn1/ont/

3 MiSeq E.coli data. For the hybrid assembler SPAdes we used a 1263X set of MiSeq paired reads, available at http://schatzlab.cshl.edu/data/nanocorr/.

Assemblers: We tested five pure-nanopore assembler pipelines:

1. LQS (https://github.com/jts/nanopore-paper-analysis) is a Nanopore-only-pipeline as it uses information provided in the original fast5 files.
   We followed the pipeline and used the settings as described in the related paper [19] and as were defined in the makefile provided by the authors: https://github.com/jts/nanopore-paper-analysis, except for the choice of a more recent Nanopolish commit [82]. Running for more than 8600 CPU hours on the 48X Nanopore set, this was the slowest of the pipelines we tried. Due to its lack of speed, and because the LQS authors themselves deprecated Nanocorrect, the LQS assembler (http://simpsonlab.github.io/), we only ran it for one case: the complete Nanopore dataset (48X). Unless it is explicitly requested, the LQS assembly pipeline does not include the Nanopolish step to allow for a fairer comparison with assemblers without a final polishing stage.
2. PBcR (http://wgs-assembler.sourceforge.net/wiki/index.php/PBcR) is a Celera based assembler, and was run with the standard nanopore settings for the 48X Nanopore dataset. For the 25X PacBio dataset, PBcR was run with the standard PacBio settings. For both the Nanopore and the PacBio 20X samples, we chose the low

coverage settings suggested in http://wgs-assembler.sourceforge.net/wiki/index.php/PBcR#Low_Coverage_Assembly, as we saw significant improvement in the assemblies when running with these settings.

3. The Canu (http://canu.readthedocs.org/) assembler is supposed to run without specifying any settings except for the genome expected size, as it should autodetect the available resources and scale the jobs accordingly. For this reason, Canu is the easiest assembler to run out of the one we tried for this analysis, and furthermore it ranks as one of the best in terms of speed and assembly quality. Similarly to PBcR, Canu is a fork of the Celera Assembler. For the smaller datasets (20X nanopore and 20X PacBio) we used the 'low-coverage' settings suggested on the package's webpage.
4. MiniAsm (https://github.com/lh3/miniasm) is a very fast assembler (< 5 min for our E.coli datasets) for long ad error-prone reads, but it does not perform an initial read error correction so the generated draft assembly will have an average identity with the reference genome similar to that of the raw reads. We ran it with the standard settings as suggested by the author.
5. Falcon (https://github.com/PacificBiosciences/FALCON) is an assembler specifically built for the long and error-prone reads from PacificBiosciences platform. As it does not access the raw PacBio h5 files, it can also be run on Nanopore reads. We ran it using the makefile provided in the example https://github.com/PacificBiosciences/FALCON/wiki/Setup%3A-Complete-example, but we changed the assembly settings to match the ones optimized for Nanopore reads in [61].

We also ran the hybrid assembler SPAdes (http://bioinf.spbau.ru/spades) with standard settings on the 1263X MiSeq paired reads and either the long reads from the Nanopore or the PacBio platforms.

Even in the presence of an initial read error correction step, the maximum average identity we observed from pure-Nanopore data assemblies was 99.0% with the tested data. In an attempt to increase this value we ran Nanopolish on a few assemblies from the 48X Nanopore dataset: the LQS assembly (which includes Nanopolish naturally as a step of the pipeline); the assembly from Canu, which provided a good single contig with the second highest average identity after LQS; and on the MiniAsm assembly, which does not include an error correction step at all. We ran Nanopolish with its standard settings in all these cases. After using Nanopolish, assemblies using LQS and Canu had similar average identities of about 99.6%, while MiniAsm's assembly reached 98.5% average identity. MiniAsm's result is remarkable because the assembly was initially constructed without an error correction step. Unfortunately, as the original MiniAsm assembly's average identity was only around 89%, the Nanopolish step was very slow (taking more than 2500 CPU hours), making the combination MiniAsm and Nanopolish highly impractical.