Genomic data compression

Shubham Chandak

Stanford University

Roche Seminar - Nov 13, 2020

Outline

- FASTQ compression – SPRING
  - Introduction and motivation
  - FASTQ format and compression results
  - Algorithms - SPRING and others
  - SPRING as a practical tool
  - Next steps: preliminary work on noisy long read compression
- Lossy compression for nanopore raw signal data
  - Background
  - Evaluation pipeline
  - Results

Outline

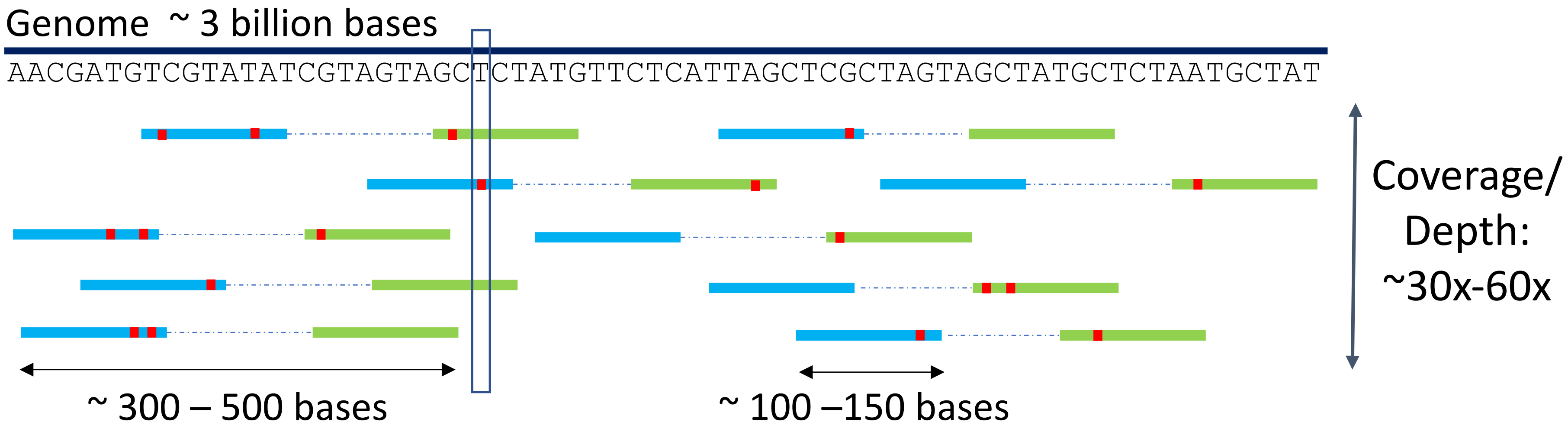
- FASTQ compression – SPRING
    - Introduction and motivation
    - FASTQ format and compression results
    - Algorithms - SPRING and others
    - SPRING as a practical tool
    - Next steps: preliminary work on noisy long read compression
- Lossy compression for nanopore raw signal data
    - Background
    - Evaluation pipeline
    - Results

Joint work with

- Kedar Tatwawadi, Stanford University
- Idoia Ochoa, UIUC
- Mikel Hernaez, UIUC
- Tsachy Weissman, Stanford University

Genome sequencing

- Genome: long string of bases {A, C, G, T}
- Sequenced as noisy paired substrings (*reads*):



Genome ~ 3 billion bases

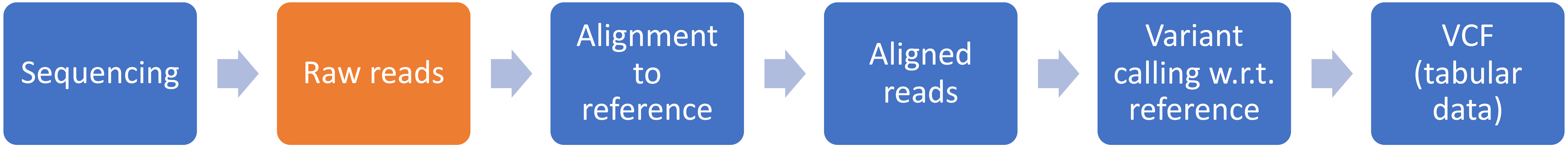
AACGATGTCGTATATCGTAGTAGCTCTATGTTCTCATTAGCTCGCTAGTAGCTATGCTCTAATGCTAT

Coverage/
Depth:
~30x-60x

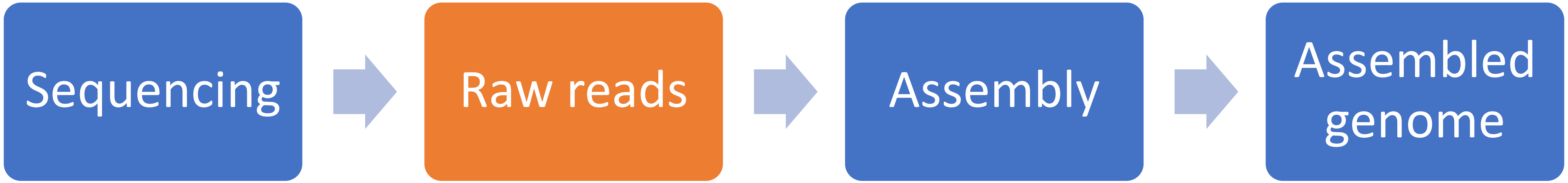
~ 300 – 500 bases

~ 100 –150 bases

Typical workflows



Sequencing → Raw reads → Alignment to reference → Aligned reads → Variant calling w.r.t. reference → VCF (tabular data)

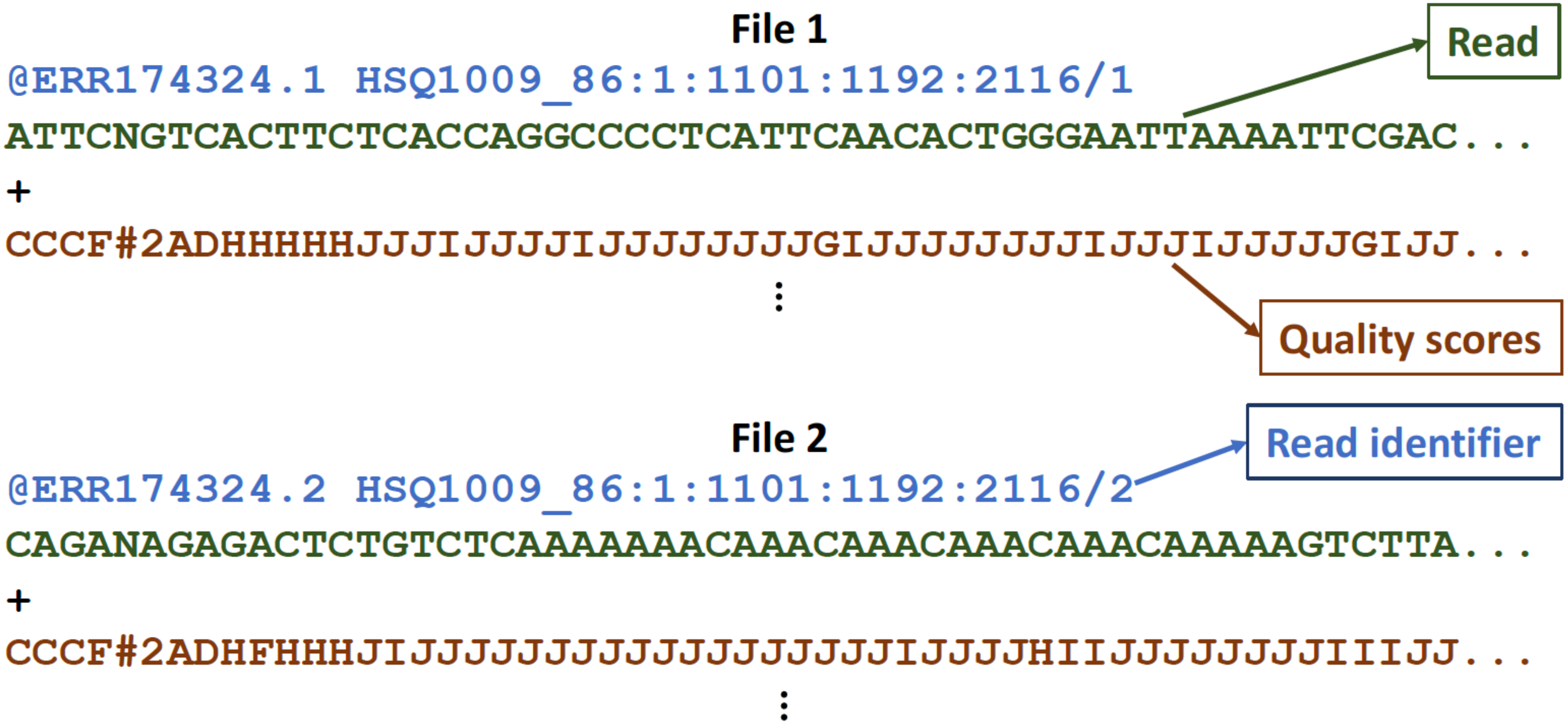


Sequencing → Raw reads → Assembly → Assembled genome

Why store raw reads?

- Pipelines improve with time - need raw data for reanalysis
- For temporary storage - alignment and assembly time-consuming
- Can’t perform alignment when reference genome not available – e.g., de novo assembly or metagenomics
- Can get better compression than aligned data compression if significant variation from reference (more on this later)!

FASTQ format



File 1

```
@ERR174324.1 HSQ1009_86:1:1101:1192:2116/1
ATTCNGTCACTTCTCACCAGGCCCCTCATTCAACACTGGGAATTAAAATTCGAC...
+
CCCF#2ADHHHHHJJJIJJJJIJJJJJJJJGIJJJJJJJJIJJJIJJJJJGIJJ...
```

⋮

Read

Quality scores

File 2

```
@ERR174324.2 HSQ1009_86:1:1101:1192:2116/2
CAGANAGAGACTCTGTCTCAAAAAAACAAACAAACAAACAAACAAAAAGTCTTA...
+
CCCF#2ADHFHHHJIJJJJJJJJJJJJJJJJJJIJJJJHIIJJJJJJJJIIIJJ...
```

⋮

Read identifier

We'll mostly focus on **reads** in this talk.

Read compression

Read compression

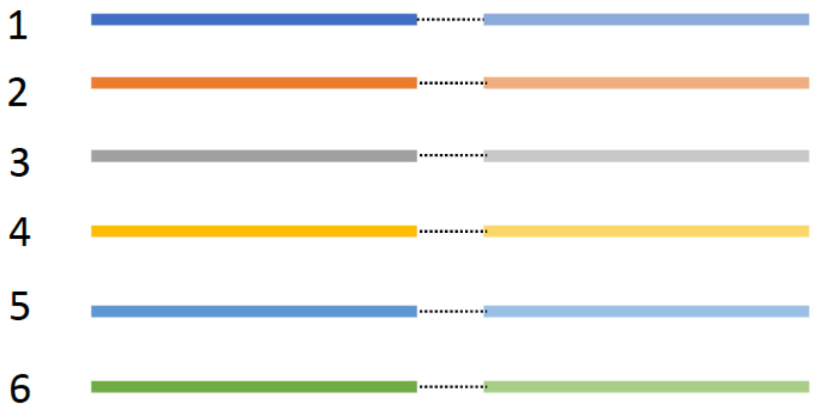
- For a typical 25x human dataset:
  - Uncompressed: 79 GB (1 byte/base)

Read compression

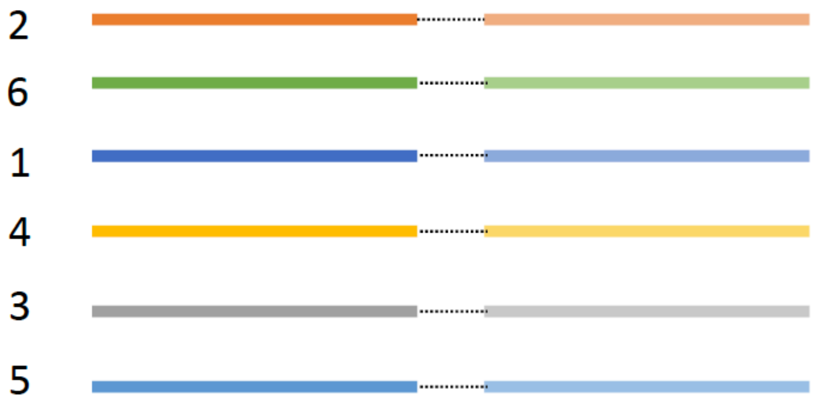
- For a typical 25x human dataset:
  - Uncompressed: 79 GB (1 byte/base)
  - Gzip: ~20 GB (2 bits/base) – still far from optimal

Read compression

- For a typical 25x human dataset:
  - Uncompressed: 79 GB (1 byte/base)
  - Gzip: ~20 GB (2 bits/base) – still far from optimal
- Order of read pairs in FASTQ irrelevant – can this help?



Original order in FASTQ



New order (preserves read pairing but pairs ordered arbitrarily)

Read compression results

Compressor	25x human
Uncompressed	79 GB
Gzip	~20 GB

Read compression results

Compressor	25x human
Uncompressed	79 GB
Gzip	~20 GB
FaStore (allow reordering)	6 GB

Łukasz Roguski, Idoia Ochoa, Mikel Hernaez, Sebastian Deorowicz; FaStore: a space-saving solution for raw sequencing data, *Bioinformatics*, Volume 34, Issue 16, 15 August 2018, Pages 2748–2756

Read compression results

Compressor	25x human
Uncompressed	79 GB
Gzip	~20 GB
FaStore (allow reordering)	6 GB
SPRING (no reordering)	**3 GB**
SPRING (allow reordering)	**2 GB**

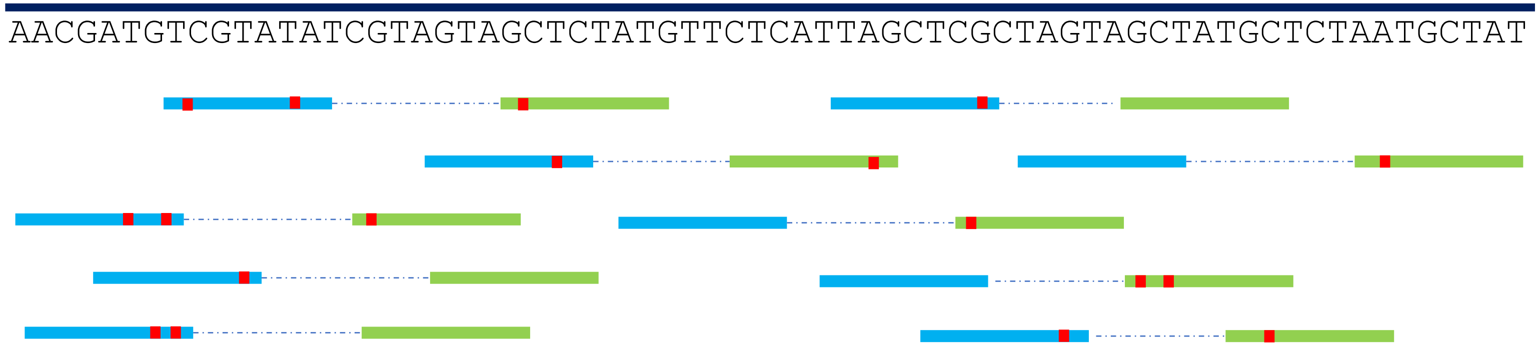
Łukasz Roguski, Idoia Ochoa, Mikel Hernaez, Sebastian Deorowicz; FaStore: a space-saving solution for raw sequencing data, *Bioinformatics*, Volume 34, Issue 16, 15 August 2018, Pages 2748–2756

Read compression results

Compressor	25x human	100x human
Uncompressed	79 GB	319 GB
Gzip	~20 GB	~80 GB
FaStore (allow reordering)	6 GB	13.7 GB
SPRING (no reordering)	**3 GB**	**10 GB**
SPRING (allow reordering)	**2 GB**	**5.7 GB**

Łukasz Roguski, Idoia Ochoa, Mikel Hernaez, Sebastian Deorowicz; FaStore: a space-saving solution for raw sequencing data, *Bioinformatics*, Volume 34, Issue 16, 15 August 2018, Pages 2748–2756

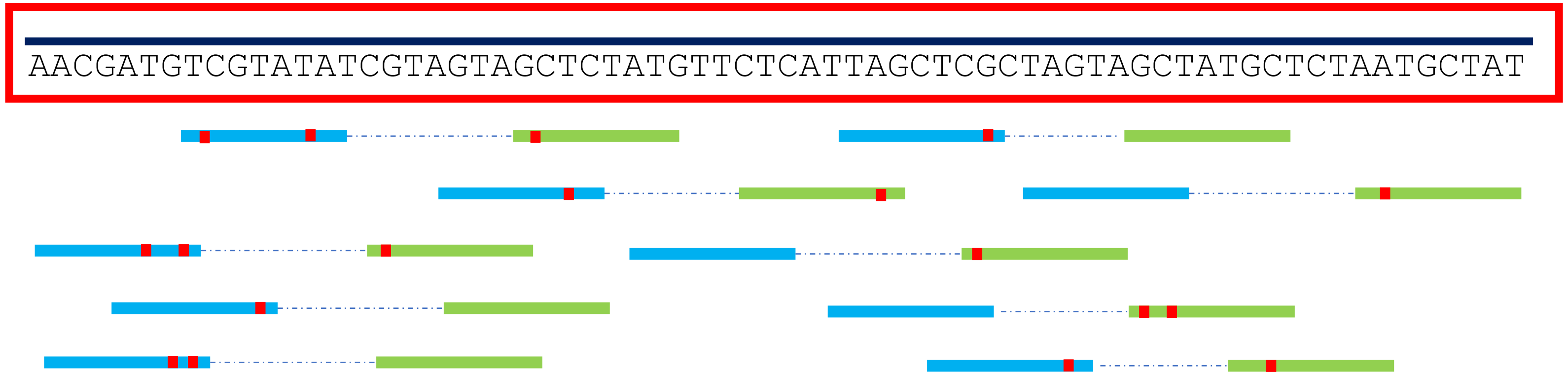
Key idea



AACGATGTCGTATATCGTAGTAGCTCTATGTTCTCATTAGCTCGCTAGTAGCTATGCTCTAATGCTAT

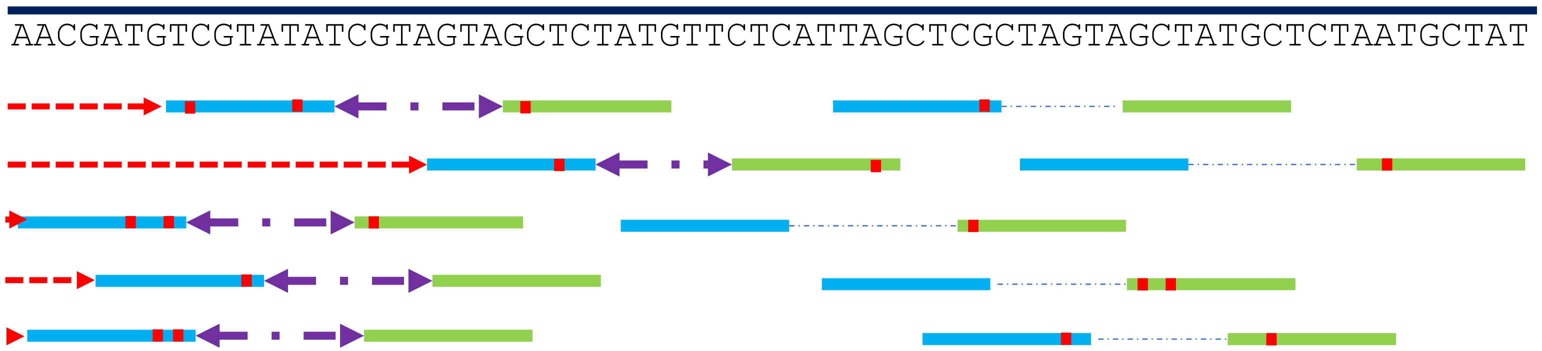
- Storing reads equivalent to

Key idea


AACGATGTCGTATATCGTAGTAGCTCTATGTTCTCATTAGCTCGCTAGTAGCTATGCTCTAATGCTAT

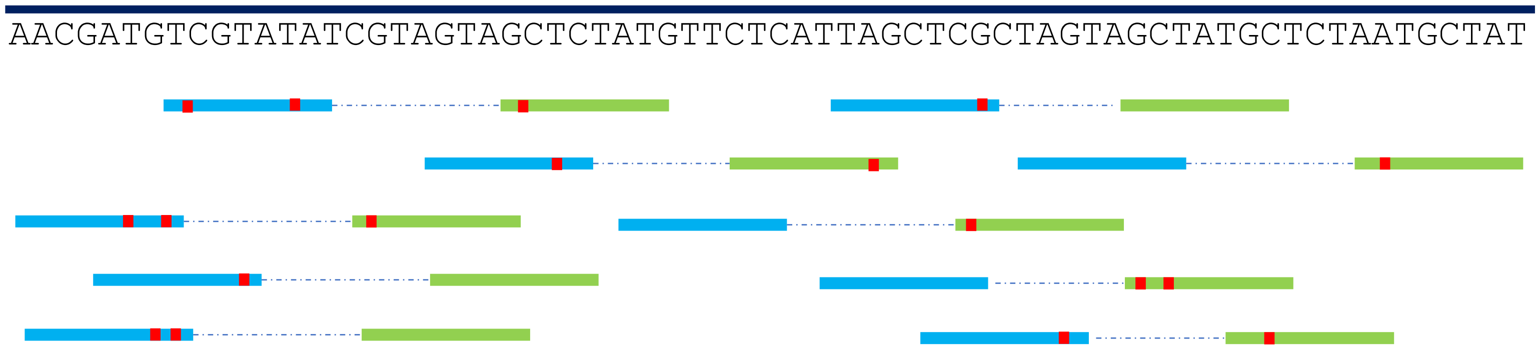
- Storing reads equivalent to
  - Store genome

Key idea


AACGATGTCGTATATCGTAGTAGCTCTATGTTCTCATTAGCTCGCTAGTAGCTATGCTCTAATGCTAT

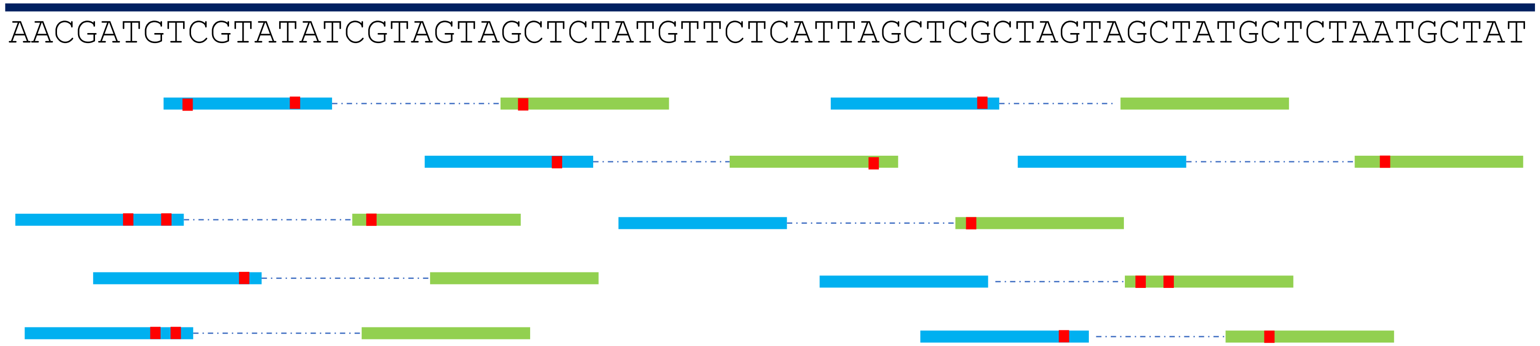
- Storing reads equivalent to
  - Store genome
  - Store read positions in genome (+ gap between paired reads)

Key idea


AACGATGTCGTATATCGTAGTAGCTCTATGTTCTCATTAGCTCGCTAGTAGCTATGCTCTAATGCTAT

- Storing reads equivalent to
  - Store genome
  - Store read positions in genome (+ gap between paired reads)
  - Store noise in reads

Key idea

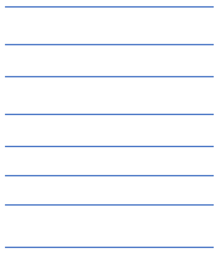

AACGATGTCGTATATCGTAGTAGCTCTATGTTCTCATTAGCTCGCTAGTAGCTATGCTCTAATGCTAT

- Storing reads equivalent to
  - Store genome
  - Store read positions in genome (+ gap between paired reads)
  - Store noise in reads
- Entropy calculations show this outperforms previous compressors

Key idea

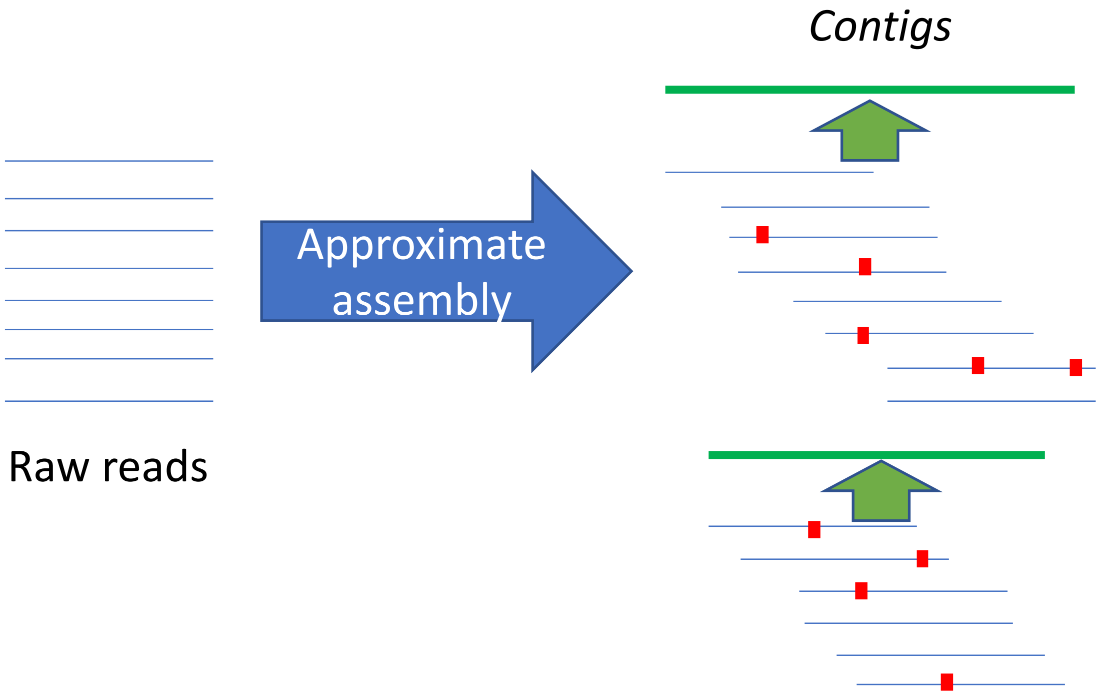
- But... How to get the genome from the reads?
- Genome assembly too expensive - big challenges:
  - resolve repeats
  - get very long pieces of genome from shorter assemblies
- Solution: Don’t need perfect assembly for compression!

SPRING workflow

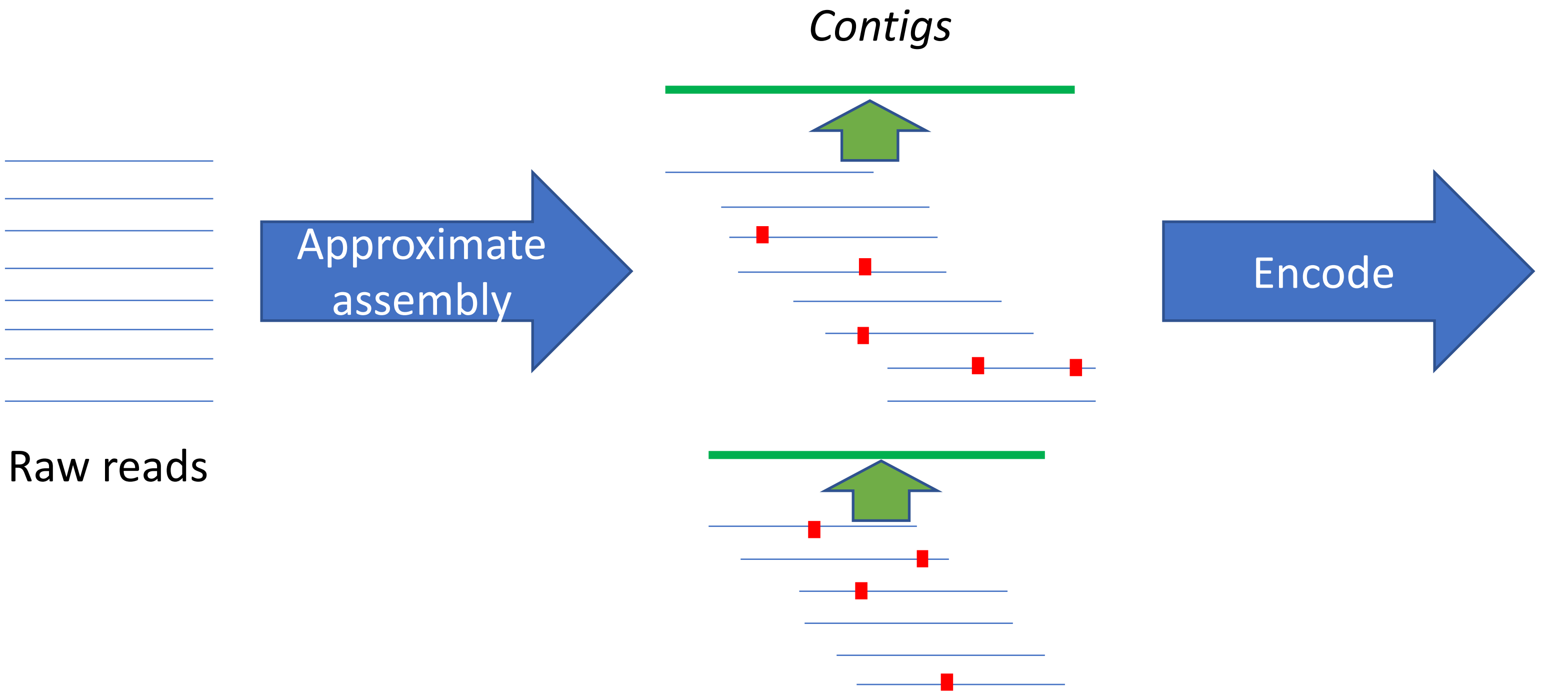


Raw reads

SPRING workflow

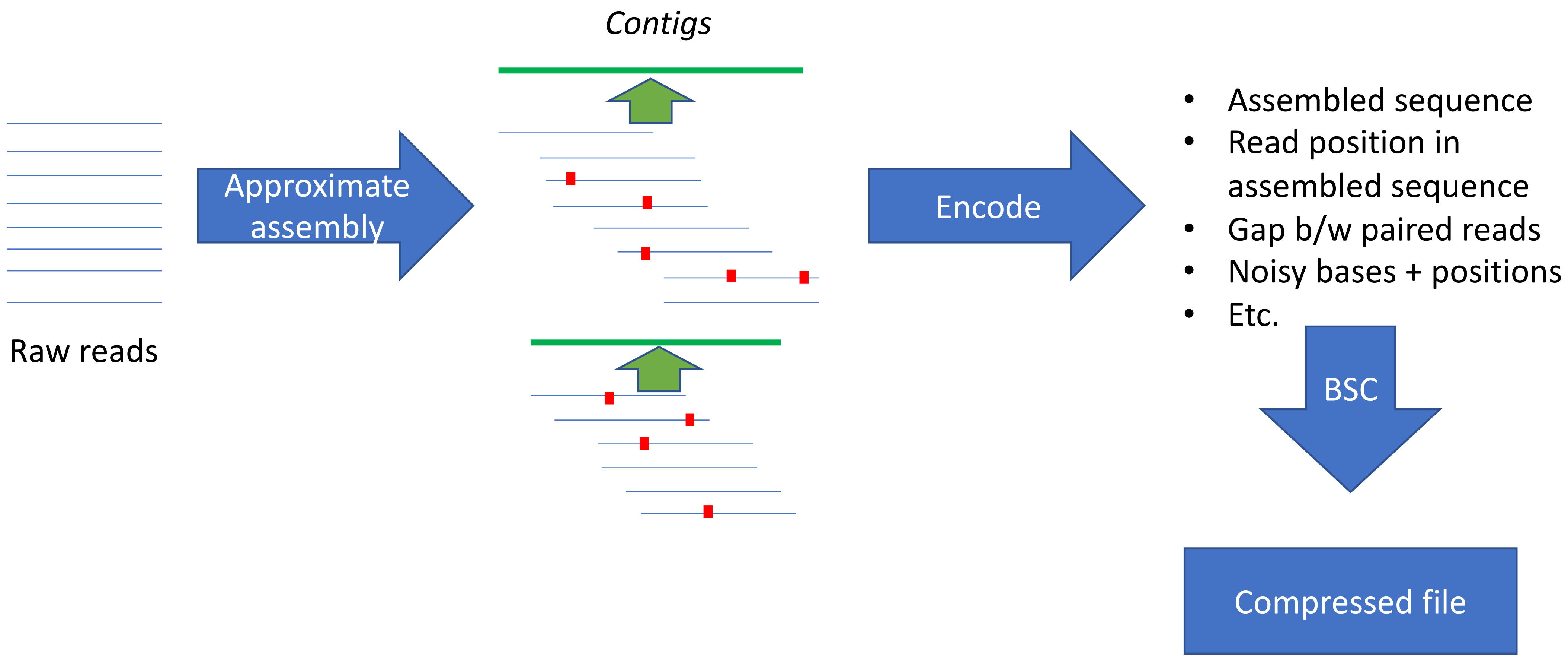


SPRING workflow



- Assembled sequence
- Read position in assembled sequence
- Gap b/w paired reads
- Noisy bases + positions
- Etc.

SPRING workflow



Raw reads

Approximate assembly

Contigs

Encode

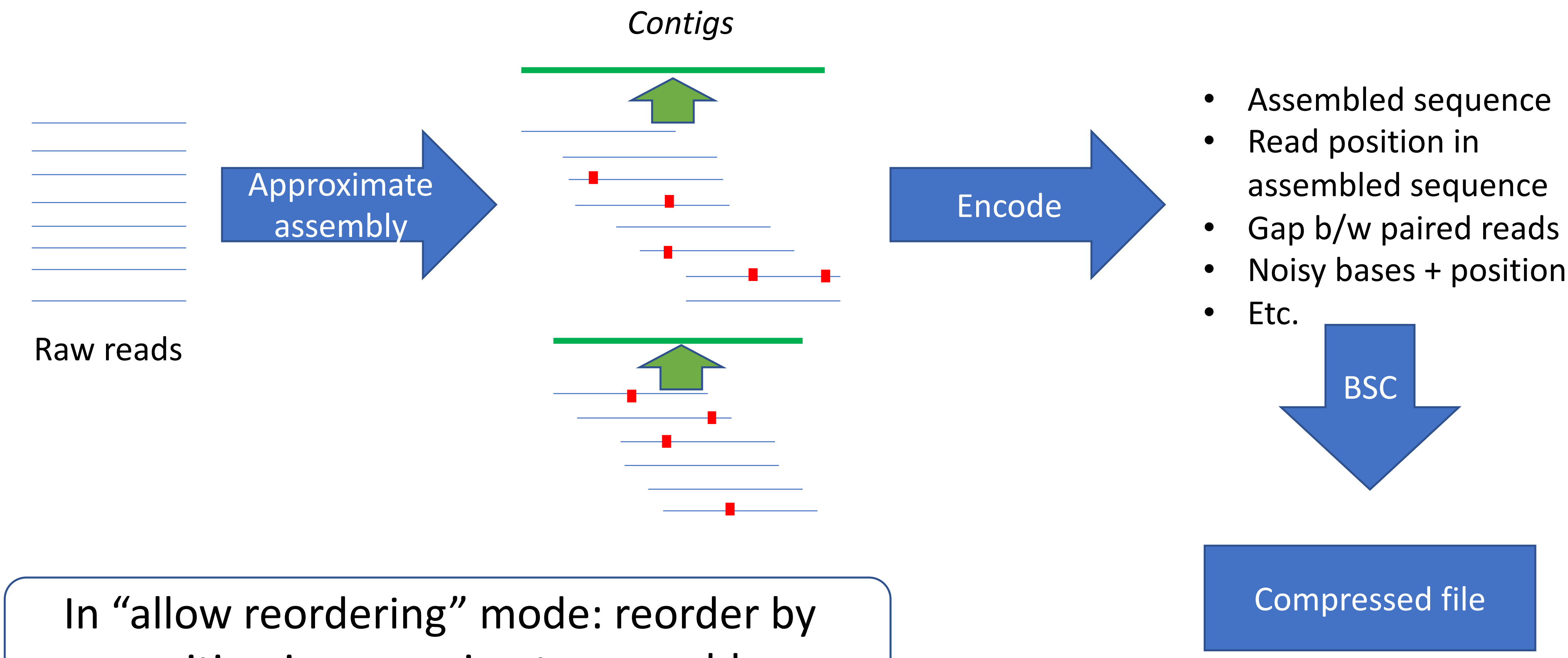
- Assembled sequence
- Read position in assembled sequence
- Gap b/w paired reads
- Noisy bases + positions
- Etc.

BSC

Compressed file

https://github.com/IlyaGrebnov/libbsc

SPRING workflow



- Assembled sequence
- Read position in assembled sequence
- Gap b/w paired reads
- Noisy bases + positions
- Etc.

In “allow reordering” mode: reorder by position in approximate assembly

https://github.com/IlyaGrebnov/libbsc

Approx. assembly/reordering step (simplified)

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
  - **ACGATCGTACGTACGATCGTCAG** *(current read)*

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
  - ACGATCGTACGTACGATCGTCAG *(current read)*
  - ACGATCGTACGTATACGGGTACG *(candidate next read)*

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
  - **ACGATCGTACGTACGATCGTCAG** *(current read)*
  - **ACGATCGTACGTATACGGGTACG** *(candidate next read)*
  - Index match found but Hamming distance too large → shift search substring by one

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
    - **ACGATCGTACGTACGATCGTCAG** *(current read)*

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
    - `ACGATCGTACGTACGATCGTCAG` *(current read)*
    - No index match found → shift search substring by one

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
  - **ACGATCGTACGTACGATCGTCAG** *(current read)*
  - **GATCGTACGTATGATGGTCATTA** *(candidate next read)*

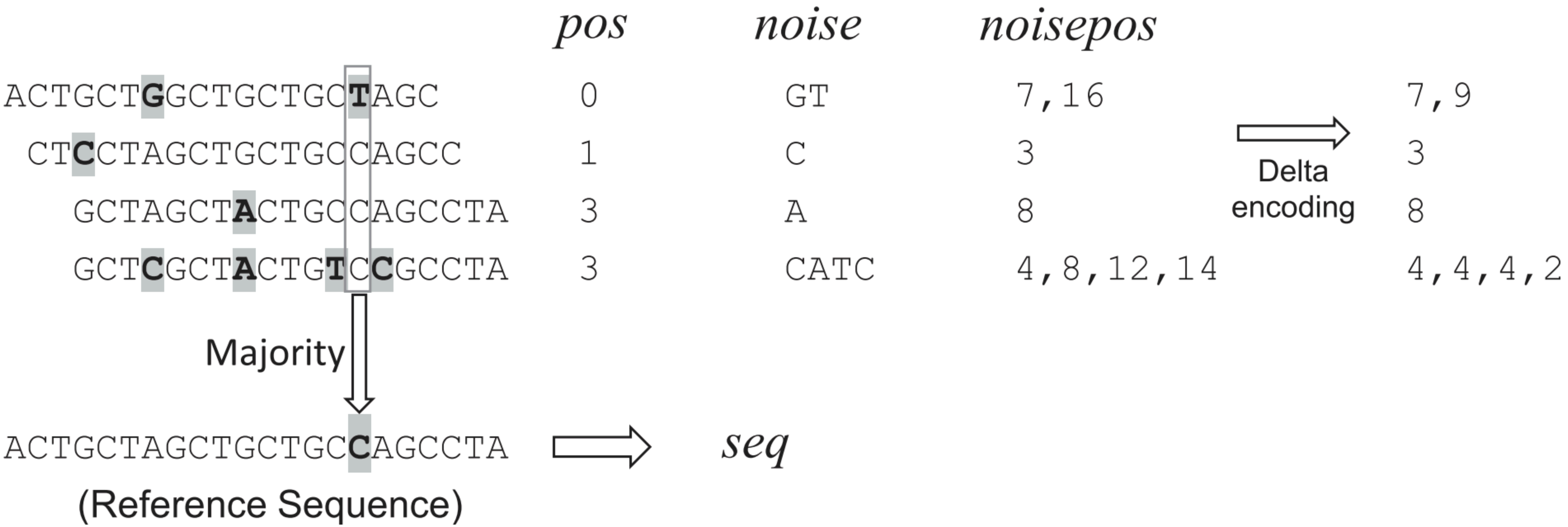
Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
  - **ACGATCGTACGTACGATCGTCAG** *(current read)*
  - **GATCGTACGTATGATGGTCATTA** *(candidate next read)*
  - Next read found!
- Repeat process with the new read

Approx. assembly/reordering step (simplified)

- Index reads by specific substrings using hash tables
- For the current read, try to find an overlapping read within small Hamming distance
- Example (reads indexed by prefix for simplicity):
  - **ACGATCGTACGTACGATCGTCAG** *(current read)*
  - **GATCGTACGTATGATGGTCATTA** *(candidate next read)*
  - Next read found!
- Repeat process with the new read.
- If no match found at any shift, pick arbitrary remaining read & start new *contig*

Consensus + encoding stage (simplified)



	pos	*noise*	*noisepos*	
ACTGCTGGCTGCTGCTAGC	0	GT	7,16	7,9
CTCCTAGCTGCTGCCAGCC	1	C	3	3
GCTAGCTACTGCCAGCCTA	3	A	8	8
GCTCGCTACTGTCCGCCTA	3	CATC	4,8,12,14	4,4,4,2

ACTGCTAGCTGCTGCCAGCCTA
(Reference Sequence)

seq

Some technical details

- Hash 2 substrings per read to improve recall rate
- Handle reverse complement reads by searching both orientations
- Specialized hash table structure (BBHash) to reduce memory usage
    - Utilize fact that all keys are known in advance
- Parallelized – each thread works on a different contig
- For reads that are left out in assembly step – try to realign with less strict threshold after consensus
- Several other heuristics to increase speed without sacrificing compression

Quality and read identifier compression

Quality and read identifier compression

- Quality – use general purpose compressor BSC (optionally apply quantization)
- Read identifier – split into tokens and use arithmetic coding [1]

[1] Bonfield, James K., and Matthew V. Mahoney. "Compression of FASTQ and SAM format sequencing data." *PloS one* 8.3 (2013): e59190.

Quality and read identifier compression

- Quality – use general purpose compressor BSC (optionally apply quantization)
- Read identifier – split into tokens and use arithmetic coding [1]

Dataset	Reads	Quality	Read identifier
Hiseq 2000 28x, 100 bp x 2	4.3	**23.8**	0.9
Novaseq 25x, 150 bp x 2	3.0	**3.6**	0.3

All human datasets. Sizes in GB.

[1] Bonfield, James K., and Matthew V. Mahoney. "Compression of FASTQ and SAM format sequencing data." *PloS one* 8.3 (2013): e59190.

Quality and read identifier compression

- Quality – use general purpose compressor BSC (optionally apply quantization)
- Read identifier – split into tokens and use arithmetic coding [1]

Dataset	Reads	Quality	Read identifier
Hiseq 2000 28x, 100 bp x 2	4.3	23.8	0.9
Novaseq 25x, 150 bp x 2	**3.0**	3.6	**0.3**
Novaseq 25x, 150 bp x 2 (allow reordering)	**2.0**	3.6	**1.4**

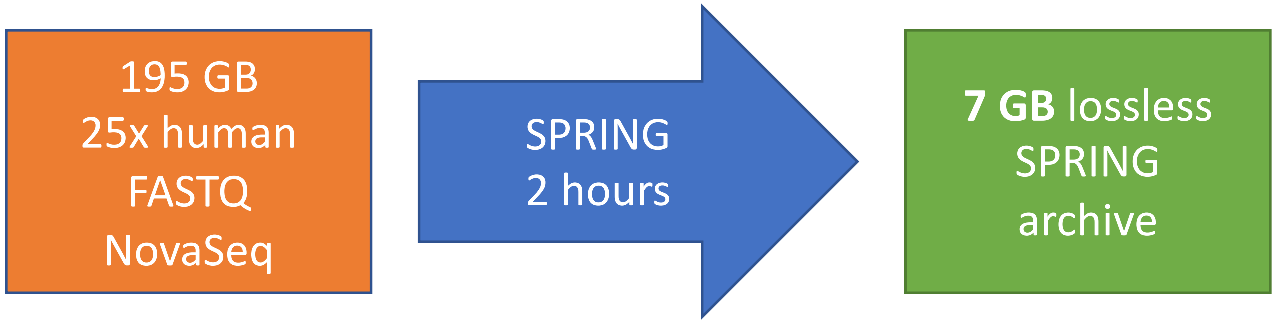
All human datasets. Sizes in GB.

[1] Bonfield, James K., and Matthew V. Mahoney. "Compression of FASTQ and SAM format sequencing data." *PloS one* 8.3 (2013): e59190.

SPRING vs. reference-based compression

195 GB
25x human
FASTQ
NovaSeq

SPRING vs. reference-based compression

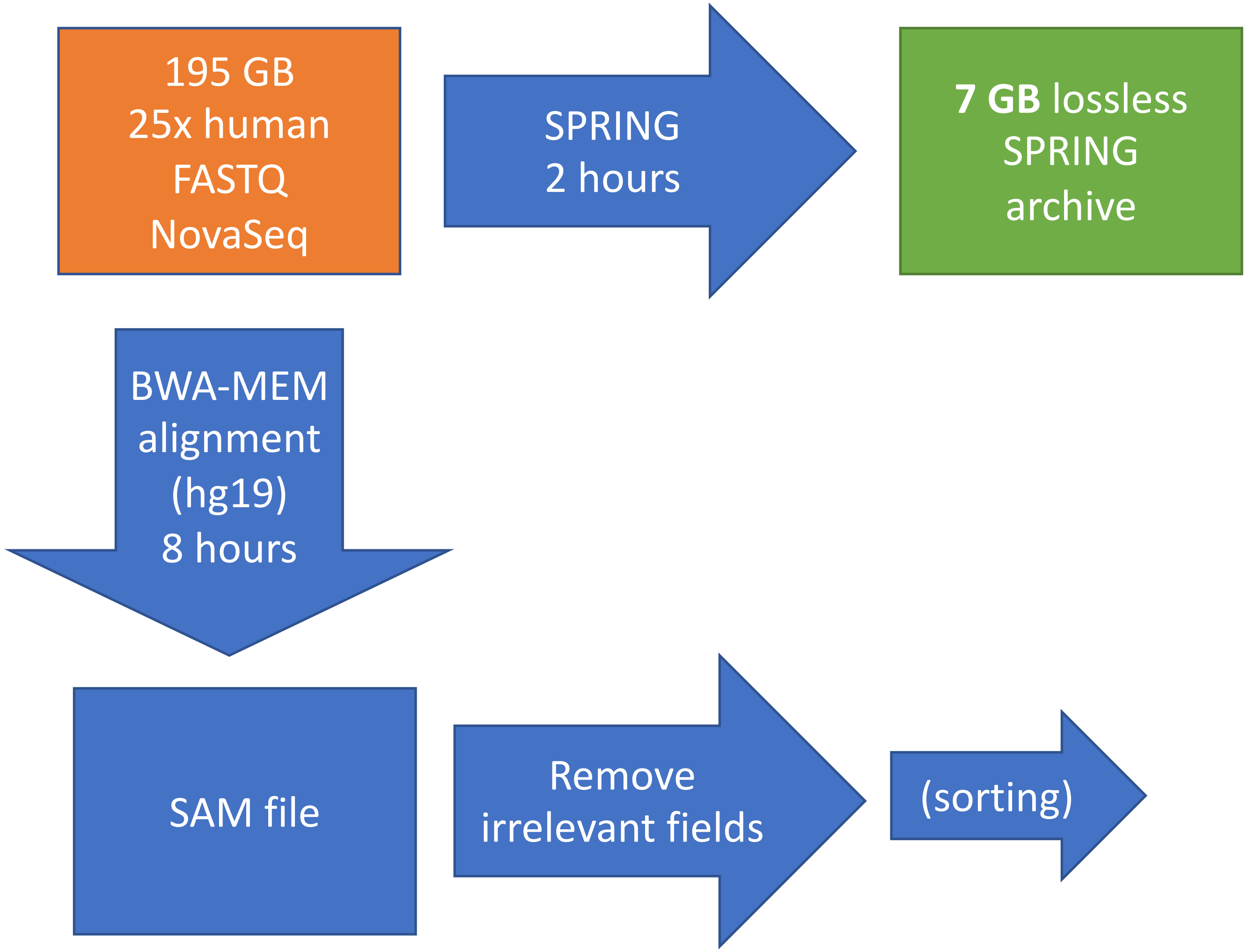


195 GB
25x human
FASTQ
NovaSeq

SPRING
2 hours

7 GB lossless
SPRING
archive

SPRING vs. reference-based compression



195 GB
25x human
FASTQ
NovaSeq

SPRING
2 hours

7 GB lossless
SPRING
archive

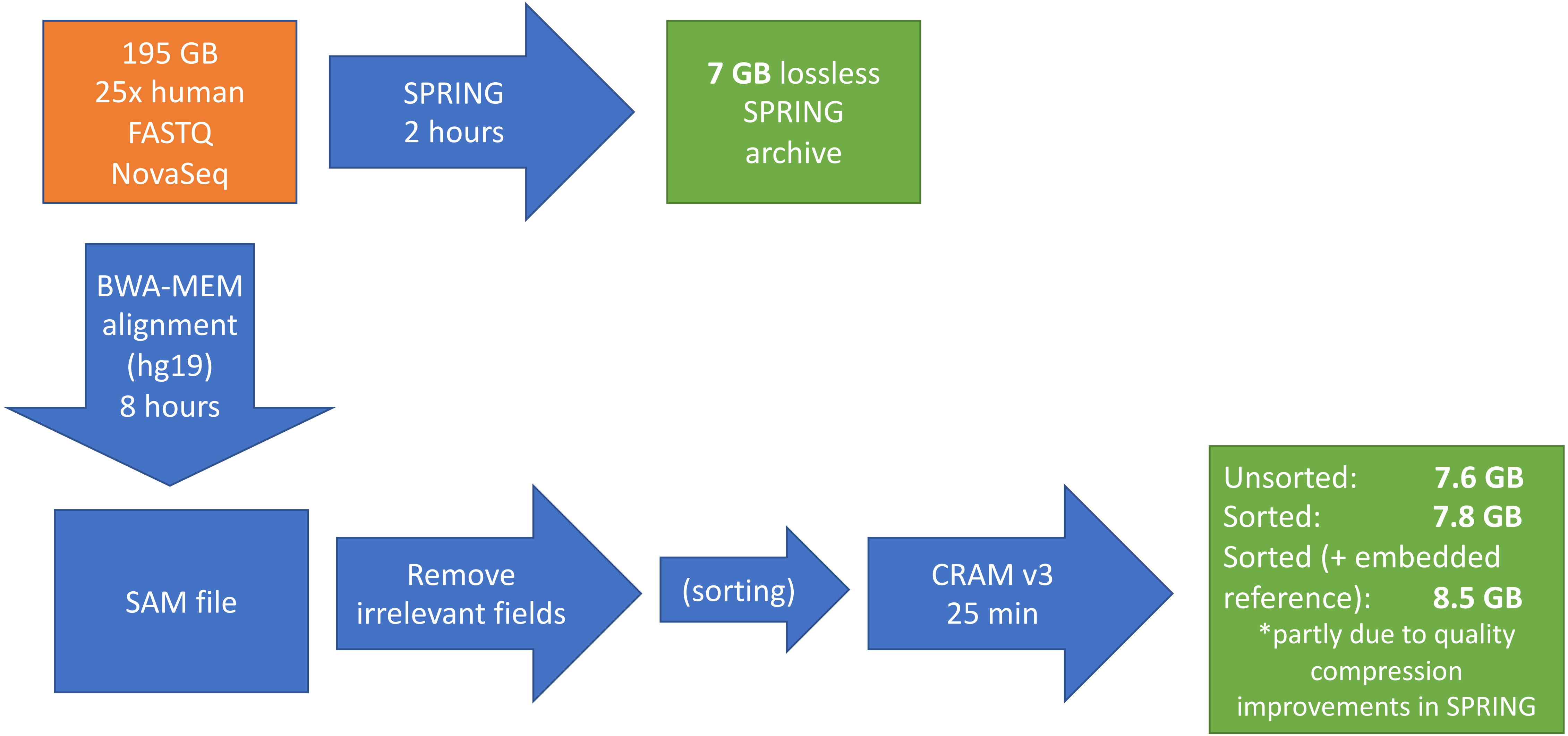
BWA-MEM
alignment
(hg19)
8 hours

SAM file

Remove
irrelevant fields

(sorting)

SPRING vs. reference-based compression



195 GB
25x human
FASTQ
NovaSeq

SPRING
2 hours

7 GB lossless
SPRING
archive

BWA-MEM
alignment
(hg19)
8 hours

SAM file

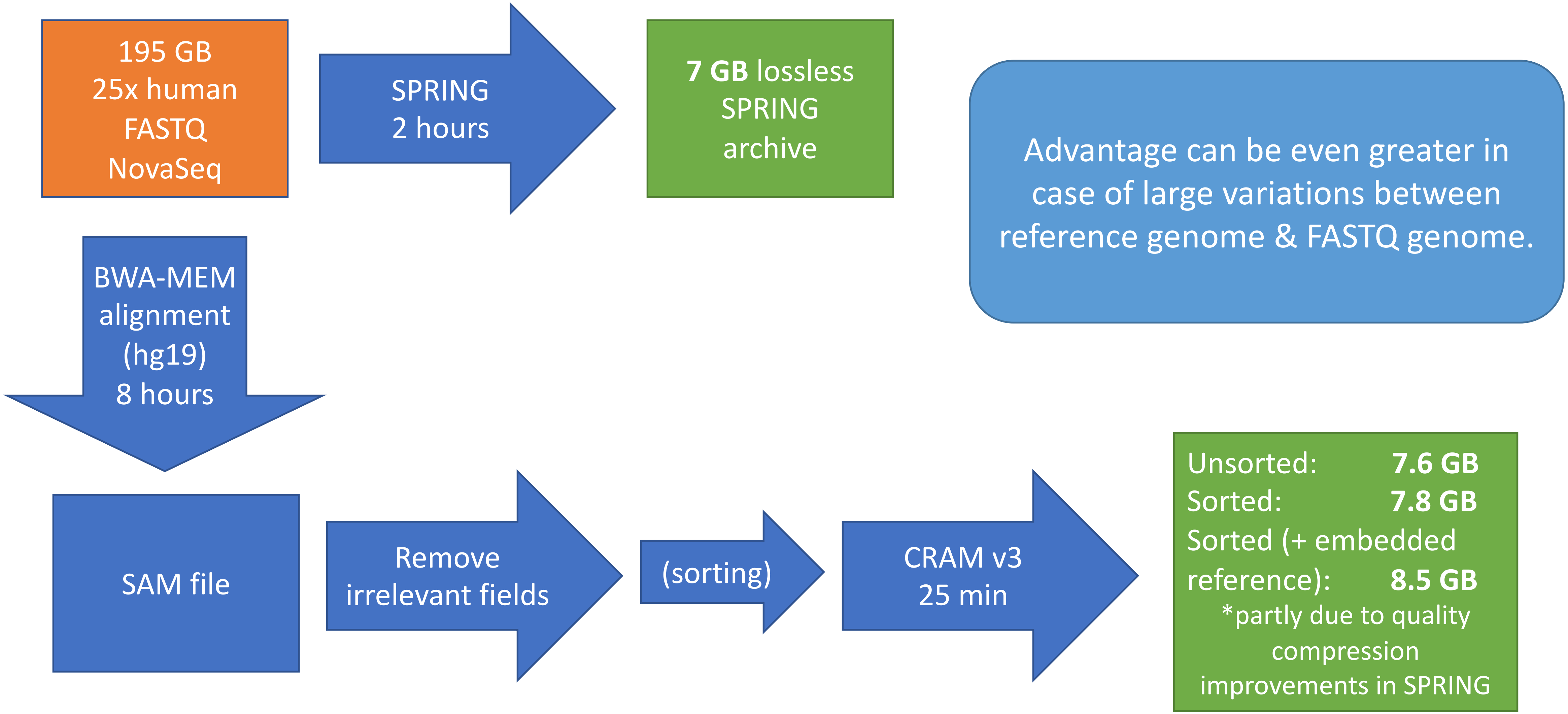
Remove
irrelevant fields

(sorting)

CRAM v3
25 min

Unsorted: **7.6 GB**
Sorted: **7.8 GB**
Sorted (+ embedded
reference): **8.5 GB**
*partly due to quality
compression
improvements in SPRING

SPRING vs. reference-based compression



195 GB
25x human
FASTQ
NovaSeq

SPRING
2 hours

7 GB lossless
SPRING
archive

Advantage can be even greater in case of large variations between reference genome & FASTQ genome.

BWA-MEM
alignment
(hg19)
8 hours

SAM file

Remove
irrelevant fields

(sorting)

CRAM v3
25 min

Unsorted: **7.6 GB**
Sorted: **7.8 GB**
Sorted (+ embedded reference): **8.5 GB**
*partly due to quality compression improvements in SPRING

Other approaches for FASTQ compression

- gzip/bzip2
- Context-based arithmetic coding: DSRC 2, Fqzcomp, Quip
- Assembly based: Leon, Quip, Assembletrie
- Reordering based:
    - Reordering based on substrings/minimizers: Orcom, Mince, FaStore, SCALCE
    - BWT-based reordering: BEETL

Numanagić, Ibrahim, et al. "Comparison of high-throughput sequencing data compression tools." *Nature Methods* 13.12 (2016): 1005.
Hernaez, Mikel, et al. "Genomic Data Compression." *Annual Review of Biomedical Data Science* 2 (2019).

Recent FASTQ compressors: FQSqueezer

- FQSqueezer [2]: Adapt general-purpose compressors such as prediction by partial matchting (PPM) and dynamic Markov coding (DMC) to read compression
  - 10-30% improvement over SPRING for bacterial datasets
- But requires significantly more time and memory than SPRING
  - Not tested on moderate to high coverage human datasets

1. Deorowicz, Sebastian. "FQSqueezer: k-mer-based compression of sequencing data." *bioRxiv* (2019): 559807.

Recent FASTQ compressors: PgRC

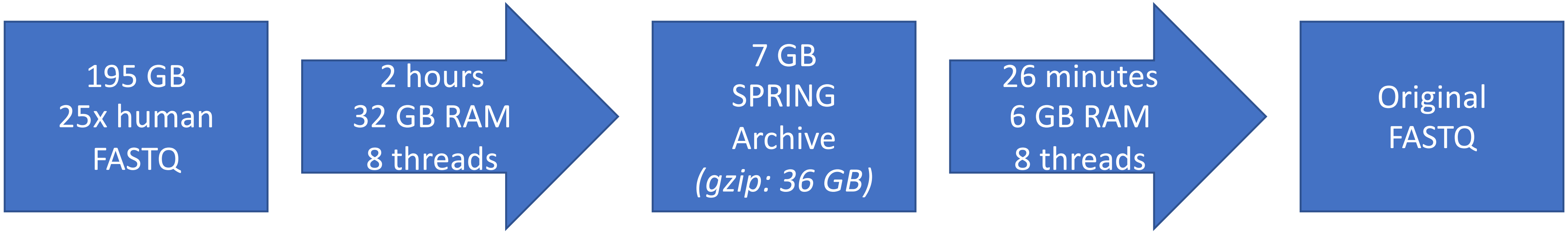
- Pseudogenome-based Read Compressor
- Similar framework as SPRING, but different “assembly” algorithm
- ~10-15% better compression than SPRING
- ~40% slower than SPRING
- Currently only supports read sequences

Kowalski, Tomasz, and Szymon Piotr Grabowski. "Engineering the Compression of Sequencing Reads." bioRxiv (2020).

Recent FASTQ compressors: alignment-based

- Setting: reference of same/related species available
- Approach:
  - Perform quick, inaccurate alignment
    - Much faster than bwa mem or minimap
  - Perform local assembly (optional)
  - Perform reference-based encoding
- Results:
  - Much better computational performance than SPRING
  - Compression generally a bit worse (even worse when reference is included in size)
- References:
  - Jammula, Nagakishore, and Srinivas Aluru. "ParRefCom: Parallel Reference-based Compression of Paired-end Genomics Read Datasets." *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. 2019.
  - Enancio (acquired by Illumina)

SPRING as a practical tool



195 GB
25x human
FASTQ

2 hours
32 GB RAM
8 threads

7 GB
SPRING
Archive
(gzip: 36 GB)

26 minutes
6 GB RAM
8 threads

Original
FASTQ

- Easy to use with support for:
    - Lossless and lossy modes
    - Variable length reads, long reads, etc.
    - Compressed in blocks to allow partial/streaming decompression
    - Scalable to large datasets
    - Gzipped I/O
- Github: https://github.com/shubhamchandak94/SPRING/

References

- Shubham Chandak, Kedar Tatwawadi, Tsachy Weissman; Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis, *Bioinformatics*, Volume 34, Issue 4, 15 February 2018, Pages 558–567
- Shubham Chandak, Kedar Tatwawadi, Idoia Ochoa, Mikel Hernaez, Tsachy Weissman; SPRING: a next-generation compressor for FASTQ data, *Bioinformatics*, bty1015
- SPRING code: https://github.com/shubhamchandak94/Spring
- genie (open-source MPEG-G codec – *under development*): https://github.com/mitogen/genie

Preliminary work: Noisy long read compression

- Joint work with Yifan Zhu
- Building a compressor for noisy long reads (e.g., ONT, PacBio)
- Very similar approach as SPRING
  - Much more challenging due to higher error rates (5-10%), including insertion and deletion errors
- Borrow ideas from assemblers but use approximations/heuristics to achieve >100x speedup
- Multi-stage filtering of reads: kmer-based search -> proper alignment
- Preliminary results encouraging, but need to scale up

Outline

- FASTQ compression – SPRING
    - Introduction and motivation
    - FASTQ format and compression results
    - Algorithms - SPRING and others
    - SPRING as a practical tool
    - Next steps: preliminary work on noisy long read compression
- Lossy compression for nanopore raw signal data
    - Background
    - Evaluation pipeline
    - Results

Impact of lossy compression of nanopore raw signal data on basecalling and consensus accuracy

Shubham Chandak *, Kedar Tatwawadi, Srivatsan Sridhar and Tsachy Weissman *

Background

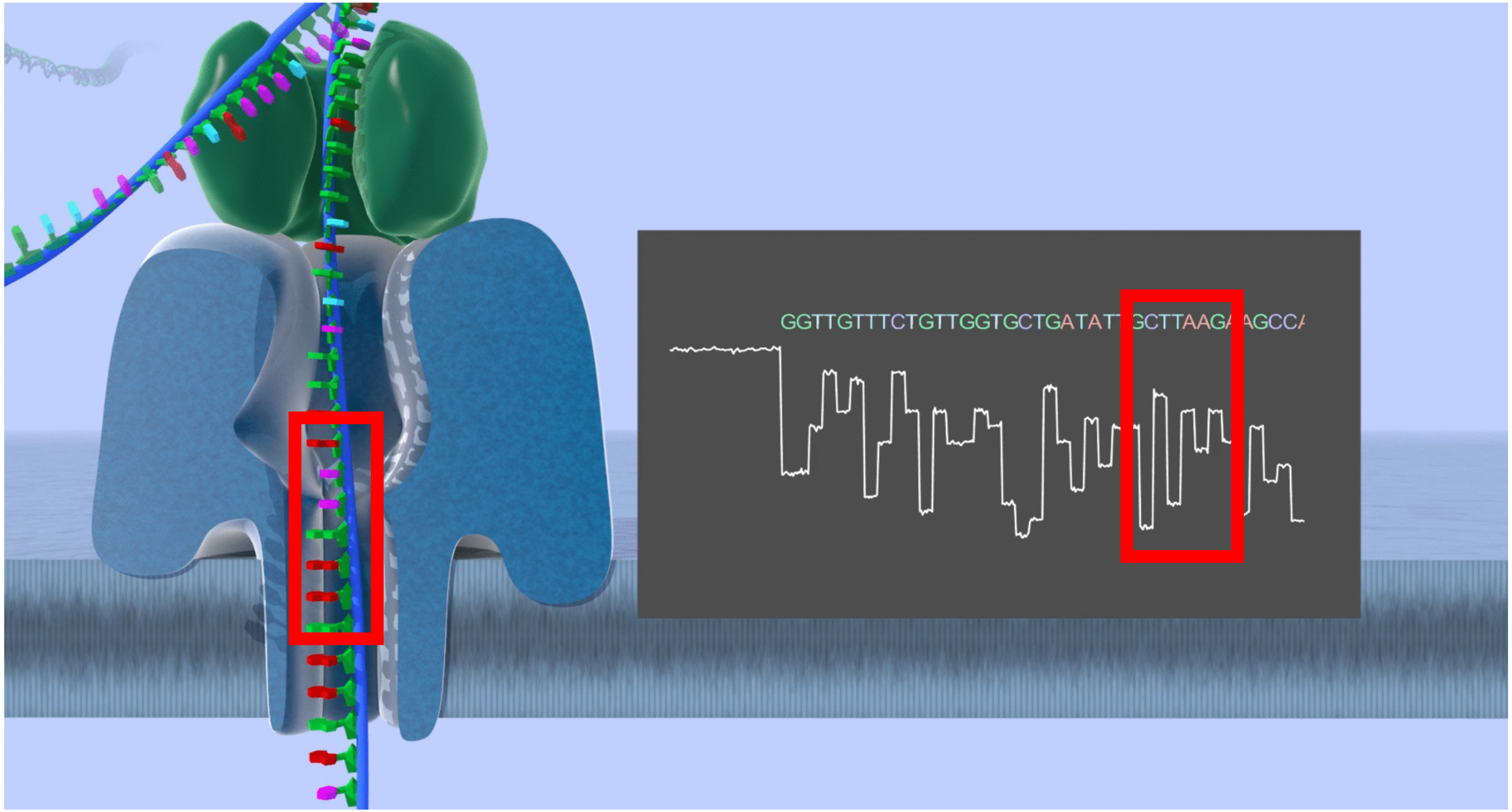
- (Oxford) nanopore sequencing gaining popularity
  - Long reads -> better assembly , structural variant discovery
  - Sequence native DNA and detect modifications
  - Real-time & portable
- Sequencer generates raw current signal that is decoded to base sequence
  - Often need to retain raw intermediate data for (re)analysis
  - Noisy – lossless compression difficult
  - Typical human whole genome exp: terabytes of raw data – 10x more than base sequence

Oxford Nanopore Sequencing

- Nanopore sequencing: portable, real time

https://directorsblog.nih.gov/2018/02/06/sequencing-human-genome-with-pocket-sized-nanopore-device/

Nanopore Sequencing Process

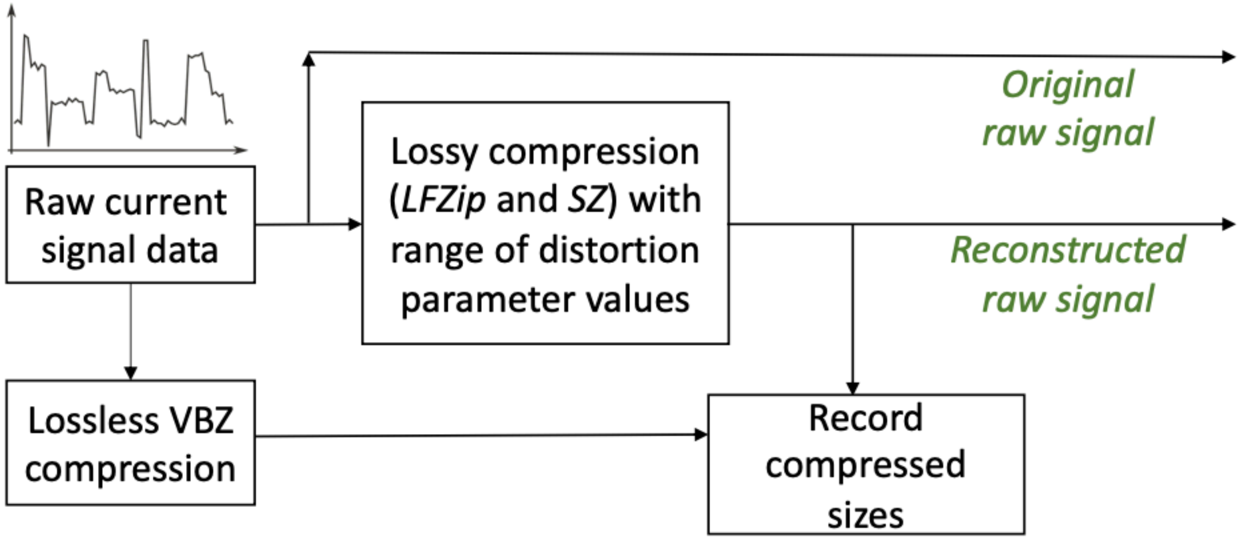


Source: https://youtu.be/E9-Rm5AoZGw

Raw data format

- HDF5 file (“.fast5”) with signal stored as series of 16-bit integers
- 5-15 current samples per base -> ~18 bytes/base (uncompressed)
- VBZ: state-of-the-art lossless compressor
  - Variable byte integer encoding followed by zstd
  - 60% size reduction over uncompressed representation
  - Still require 1 TB for 30x human whole genome data

Evaluation pipeline: part 1



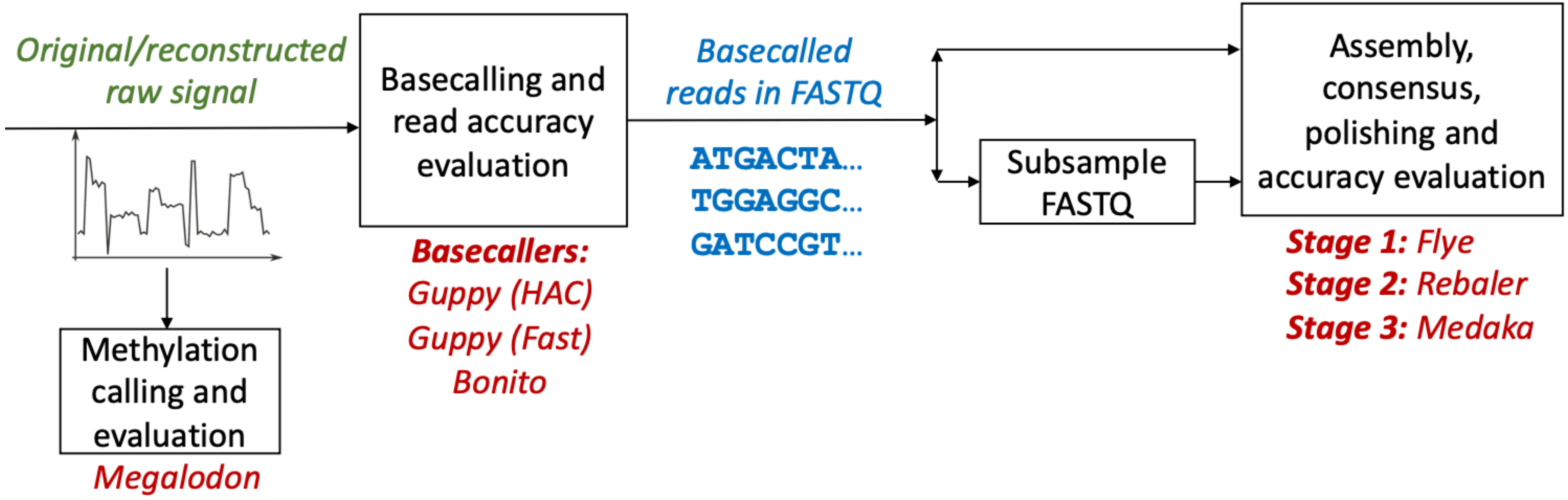
Raw current signal data
Lossy compression (*LFZip* and *SZ*) with range of distortion parameter values
Original raw signal
Reconstructed raw signal
Lossless VBZ compression
Record compressed sizes

(a) Lossless and lossy compression of raw signal data

Note on lossy time-series compressors LFZip and SZ:

- Guarantee reconstruction at each time step is within ϵ of true value (ϵ user defined parameter)
- Rely on simple prediction/quantization followed by entropy coding (gzip/bzip2/…)
- LFZip simply performs uniform scalar quantization (“rounding”) followed by entropy coding

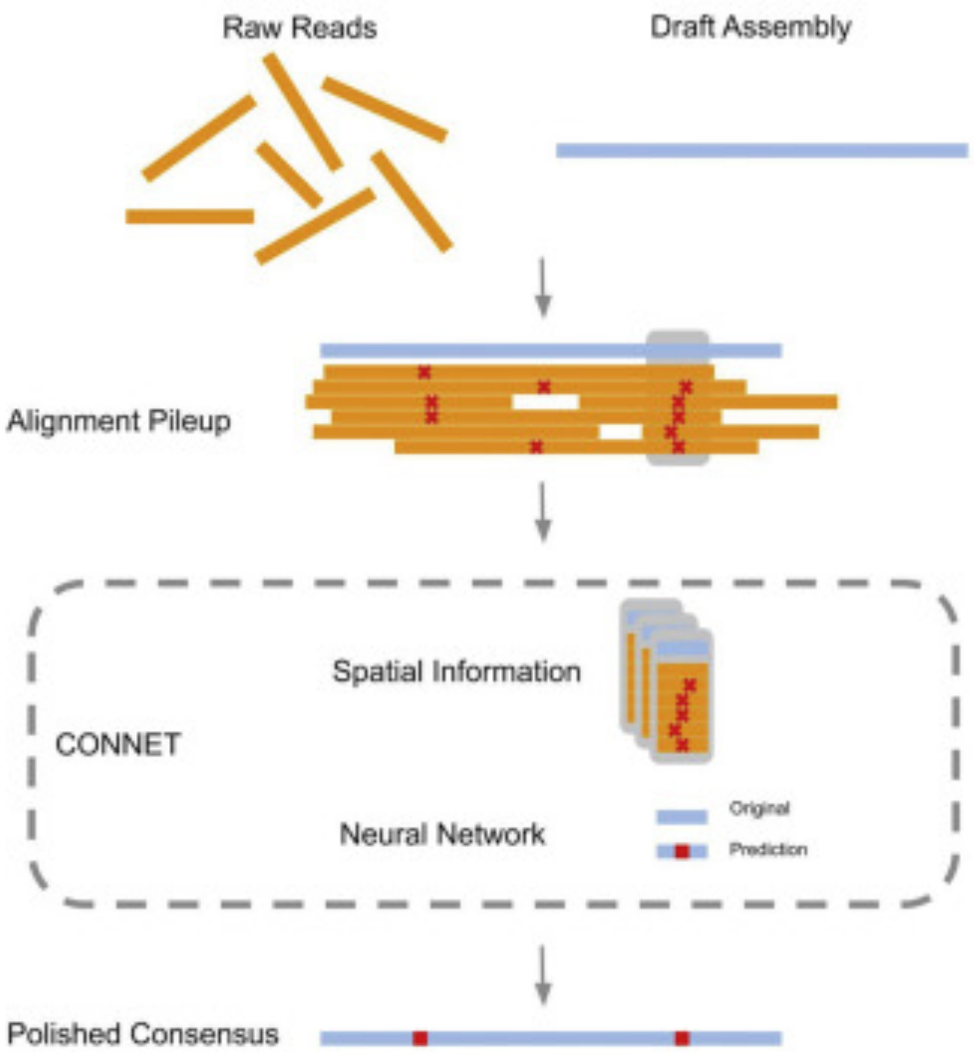
Evaluation pipeline: part 2


Original/reconstructed raw signal
Basecalling and read accuracy evaluation
Basecallers:
Guppy (HAC)
Guppy (Fast)
Bonito
Basecalled reads in FASTQ
ATGACTA...
TGGAGGC...
GATCCGT...
Subsample FASTQ
Assembly, consensus, polishing and accuracy evaluation
Stage 1: Flye
Stage 2: Rebaler
Stage 3: Medaka
Methylation calling and evaluation
Megalodon

(b) Basecalling, consensus and methylation calling accuracy analysis

Note: Attempt to "future-proof" by testing various tools/use cases

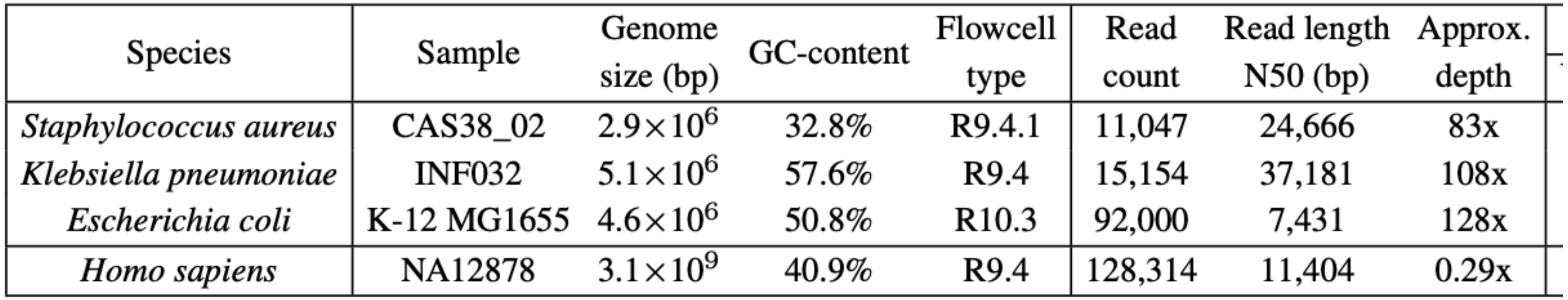
Why consensus accuracy might differ from basecall (read) level accuracy:
systematic vs. random errors



Source: https://www.sciencedirect.com/science/article/pii/S2589004220303138

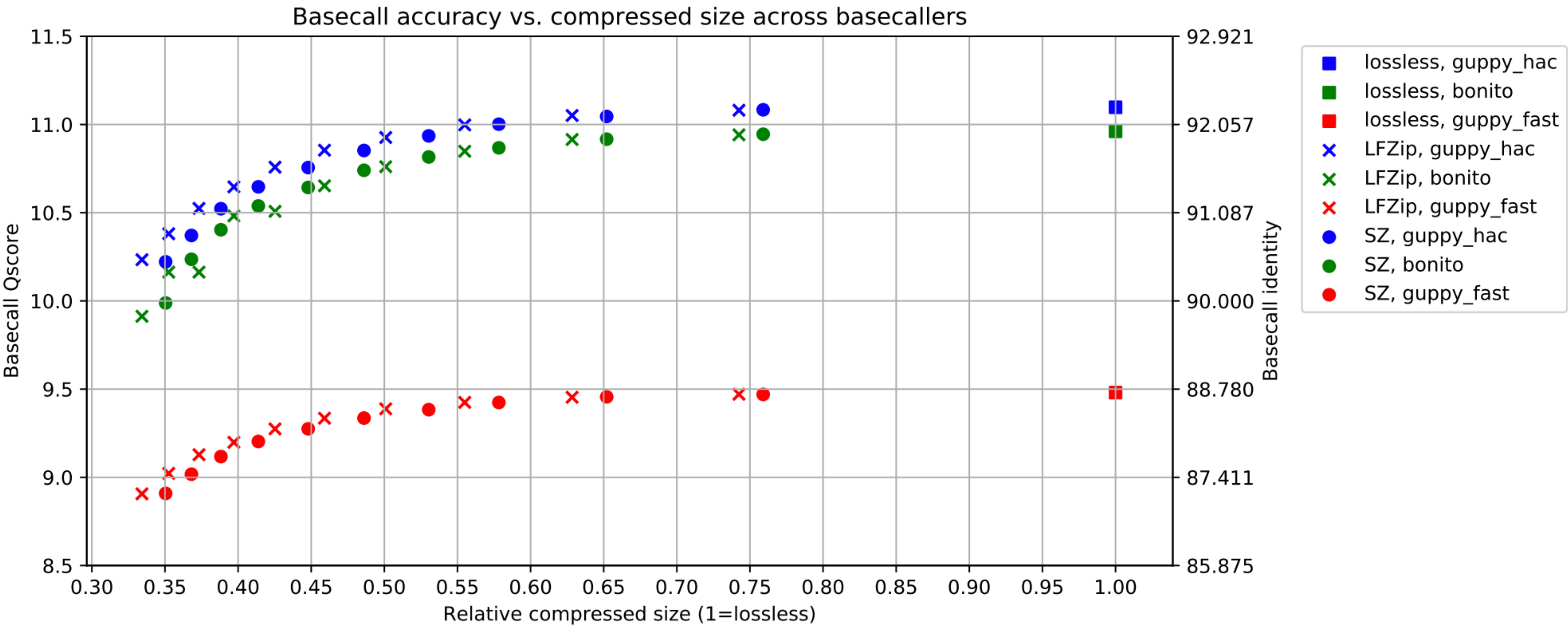
Datasets

- Human and bacterial datasets for basecall accuracy
- Bacterial datasets for consensus accuracy
- Human dataset with bisulfite benchmark for methylation accuracy

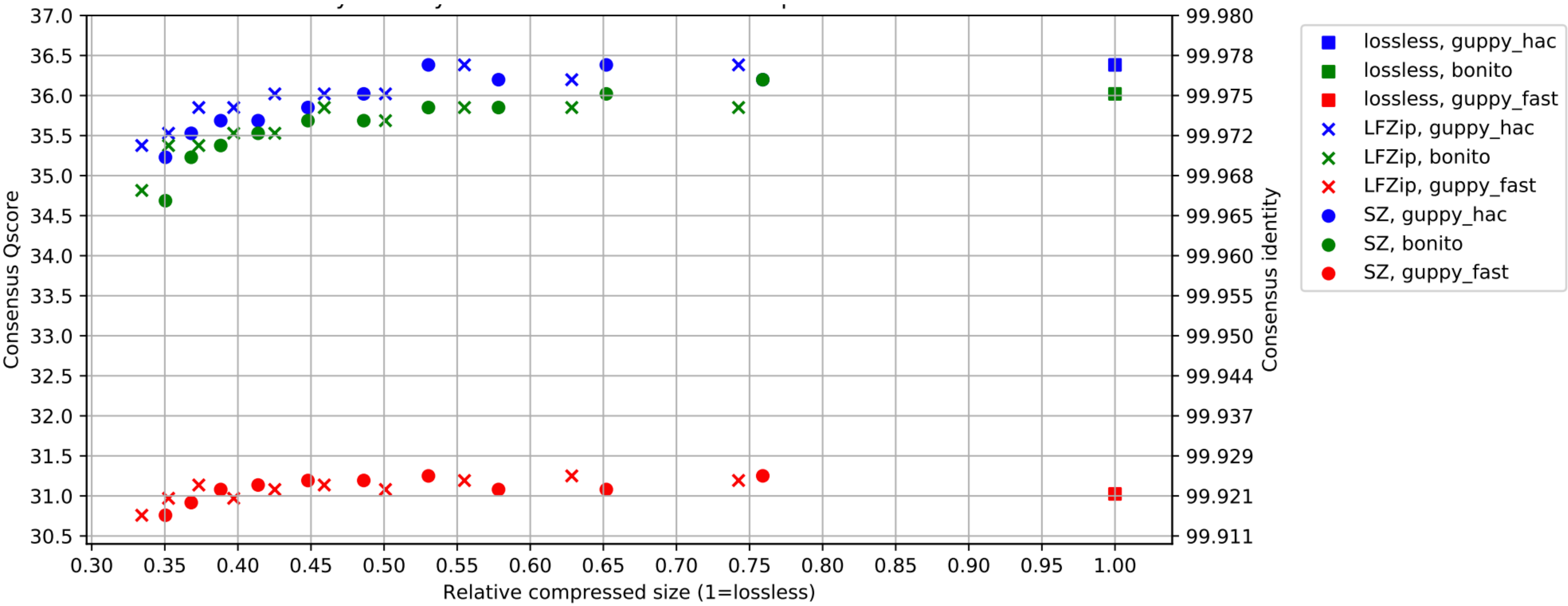


Species	Sample	Genome size (bp)	GC-content	Flowcell type	Read count	Read length N50 (bp)	Approx. depth
Staphylococcus aureus	CAS38_02	2.9×10^6	32.8%	R9.4.1	11,047	24,666	83x
Klebsiella pneumoniae	INF032	5.1×10^6	57.6%	R9.4	15,154	37,181	108x
Escherichia coli	K-12 MG1655	4.6×10^6	50.8%	R10.3	92,000	7,431	128x
Homo sapiens	NA12878	3.1×10^9	40.9%	R9.4	128,314	11,404	0.29x

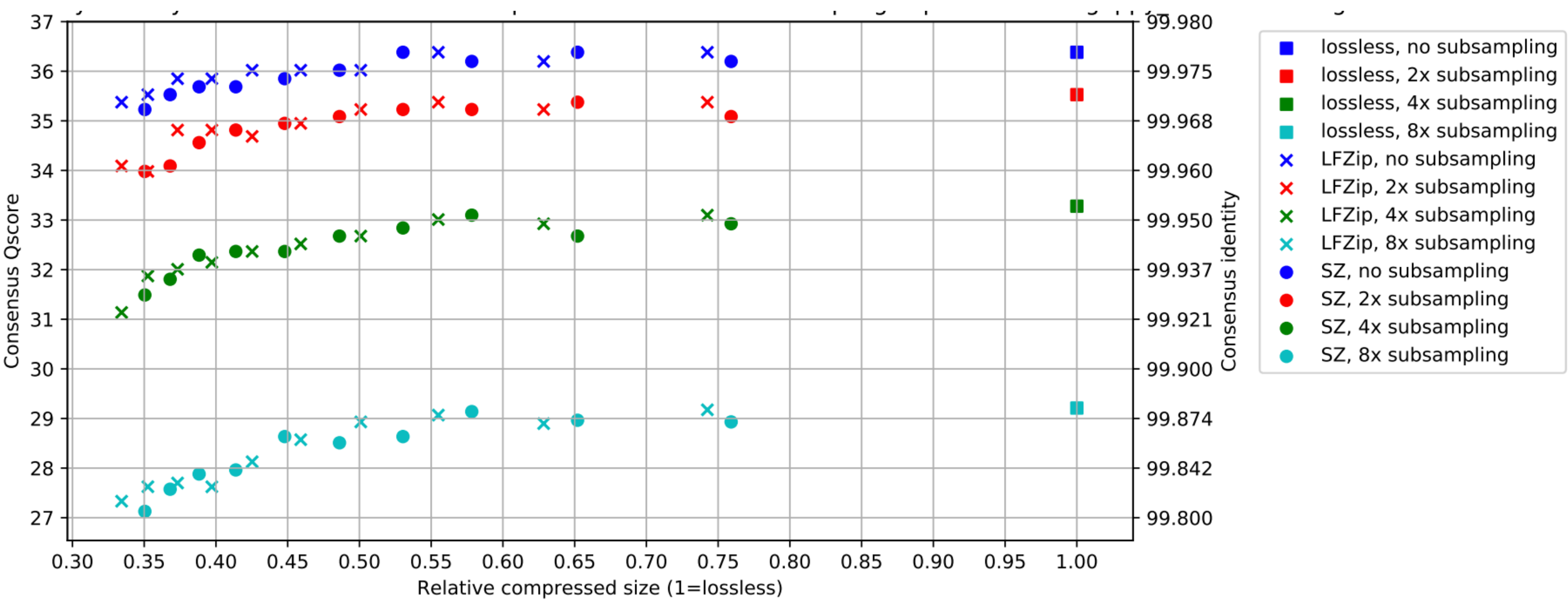
Basecall accuracy



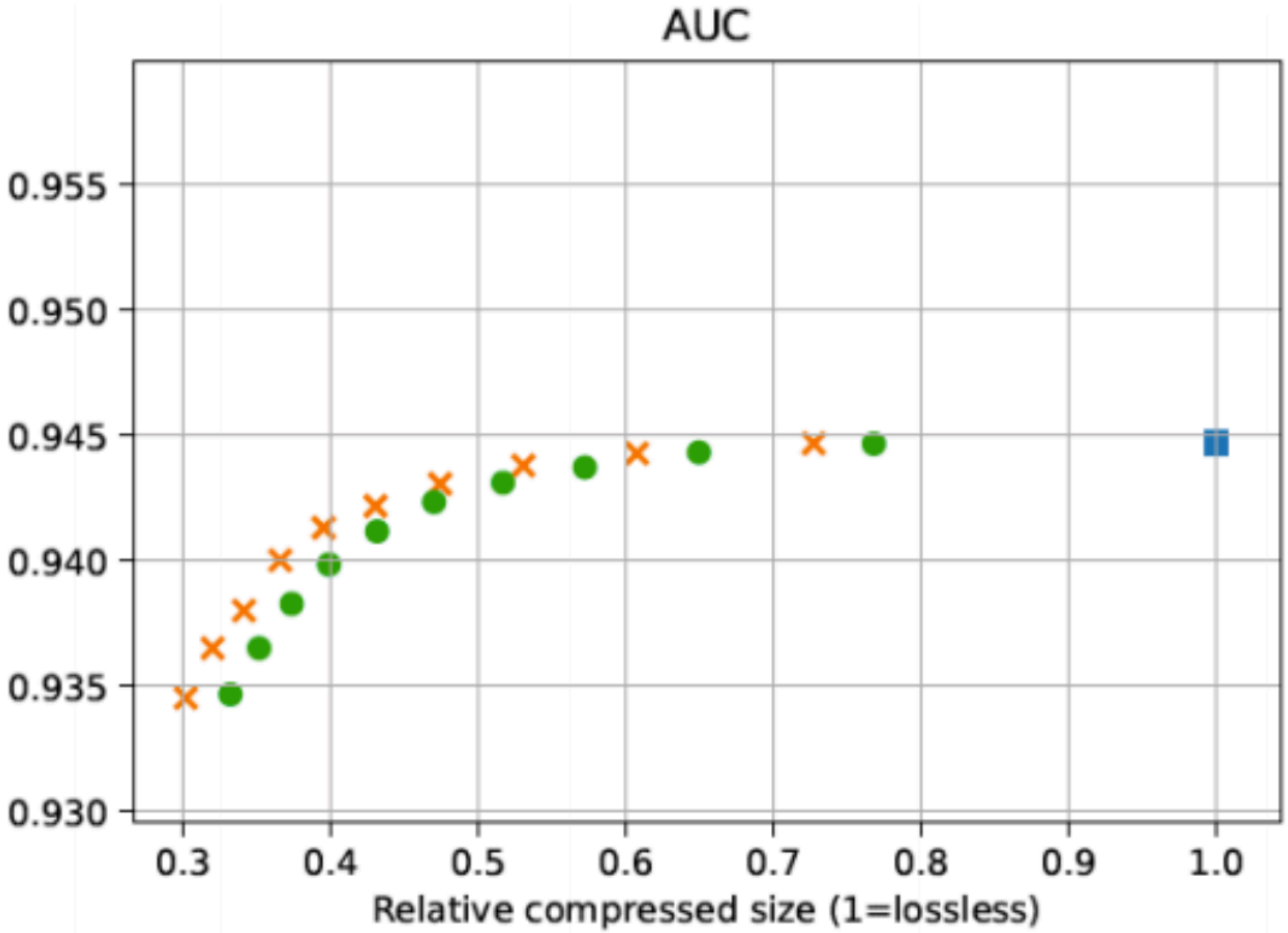
Consensus accuracy



Subsampling experiments



Per-read methylation calling accuracy



Summary

- Lossy compression achieves 35-50% reduction over current best lossless compression:
  - <0.2% reduction in basecall (read) accuracy
  - <0.002% reduction in consensus accuracy (even better for high coverage)
- Highly practical – LFZip simply reduces the data resolution!
- Can be adopted at the nanopore sequencer device itself
  - Similar to Illumina reducing quality score resolution from 40 to 4.
- Future work:
  - Specialized lossy compressors for this data
  - Further evaluation on human data with improved benchmark datasets

Availability

- Biorxiv: https://www.biorxiv.org/content/10.1101/2020.04.19.049262v3
- Evaluation scripts, data, plots: https://github.com/shubhamchandak94/lossy_compression_evaluation

Thank you!