SPRING: a next-generation compressor for FASTQ data

Shubham Chandak

Stanford University

ISMB/ECCB 2019

Joint work with

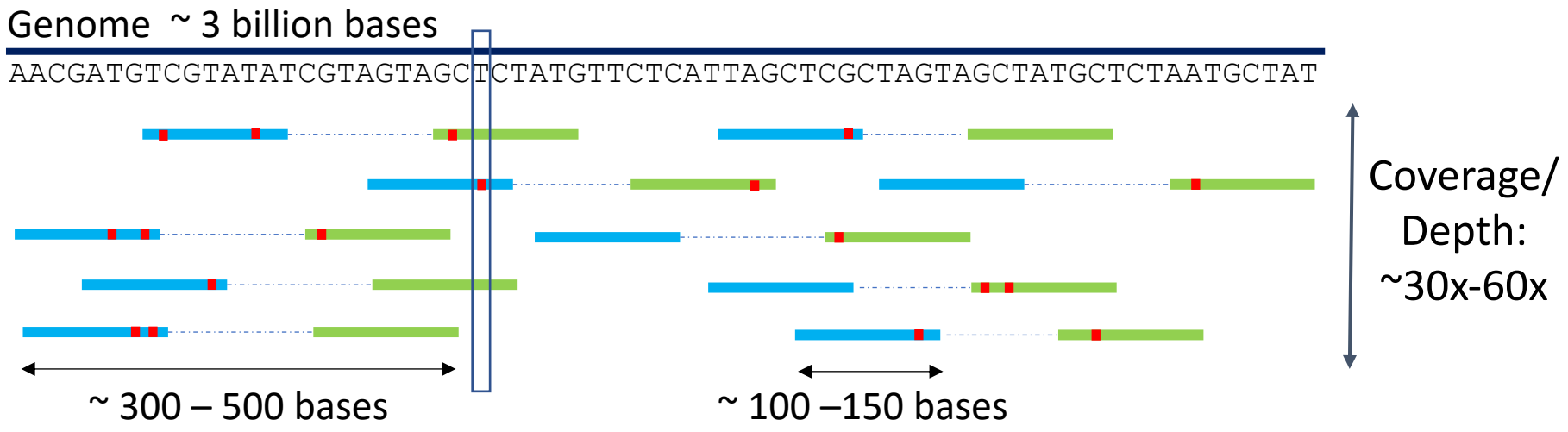
- Kedar Tatwawadi, Stanford University
- Idoia Ochoa, UIUC
- Mikel Hernaez, UIUC
- Tsachy Weissman, Stanford University

Outline

- Introduction and motivation
- FASTQ format and compression results
- Algorithms - SPRING and others
- SPRING as a practical tool
- Next steps

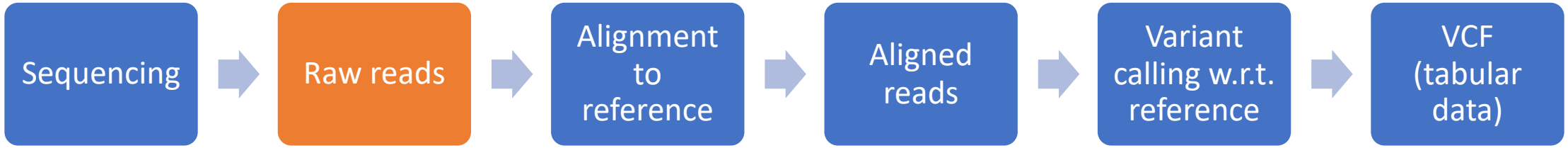
Genome sequencing

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

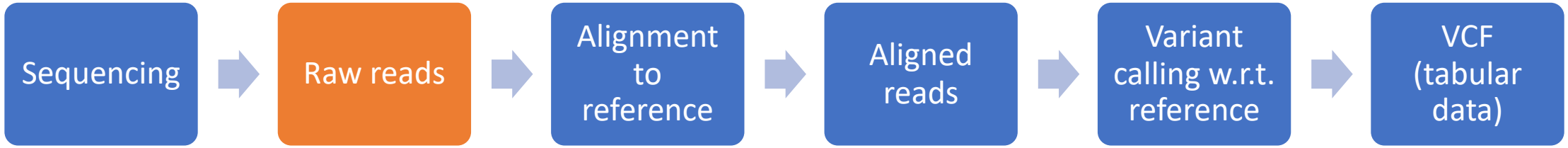


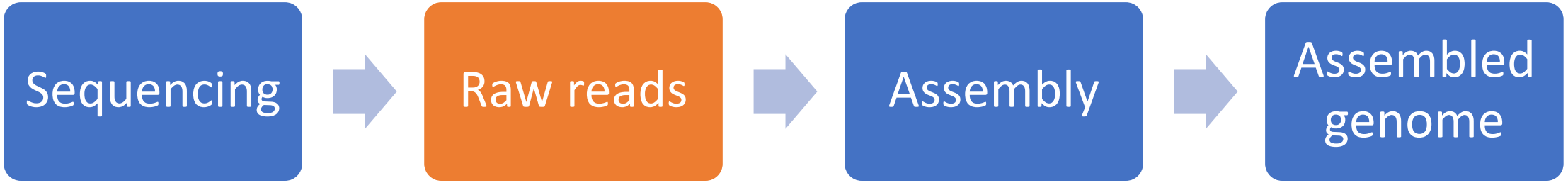
Typical workflows

Typical workflows



Typical workflows





Why store raw reads?

Why store raw reads?

- Pipelines improve with time - need raw data for reanalysis

Why store raw reads?

- Pipelines improve with time - need raw data for reanalysis
- For temporary storage - alignment and assembly time-consuming

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

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

FASTQ format

File 1

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

File 2

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

Read

Quality scores

Read identifier

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

Read compression

Read compression

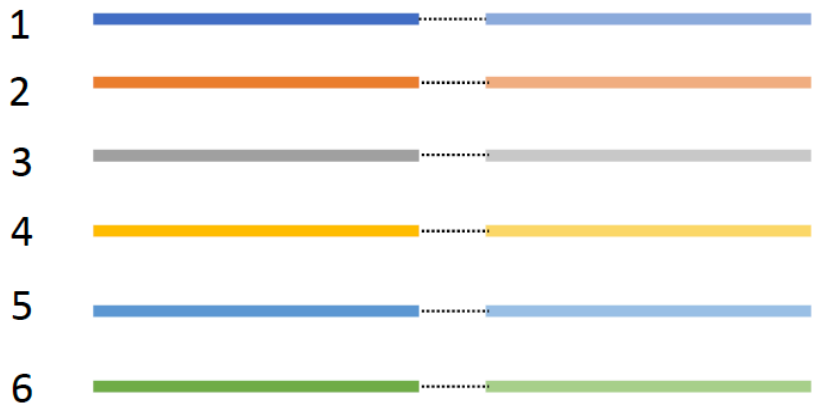
- For a typical 25x human dataset:
  - Uncompressed: 79 GB (1 byte/base)

Read compression

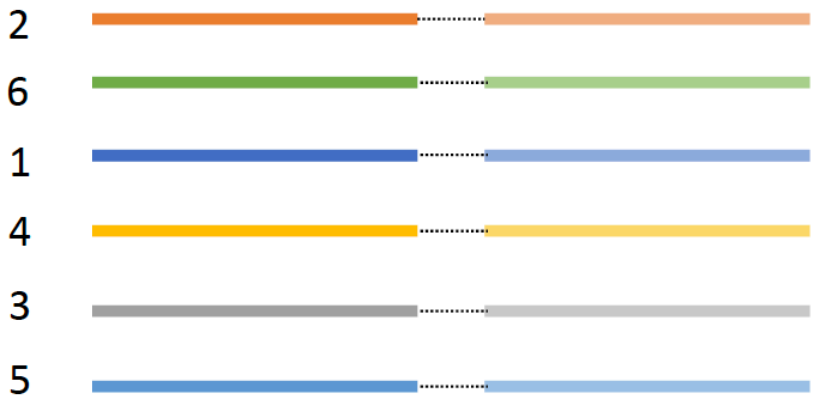
- For a typical 25x human dataset:
  - Uncompressed: 79 GB (1 byte/base)
  - Gzip: ~20 GB (2 bits/base) – still far from optimal

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**

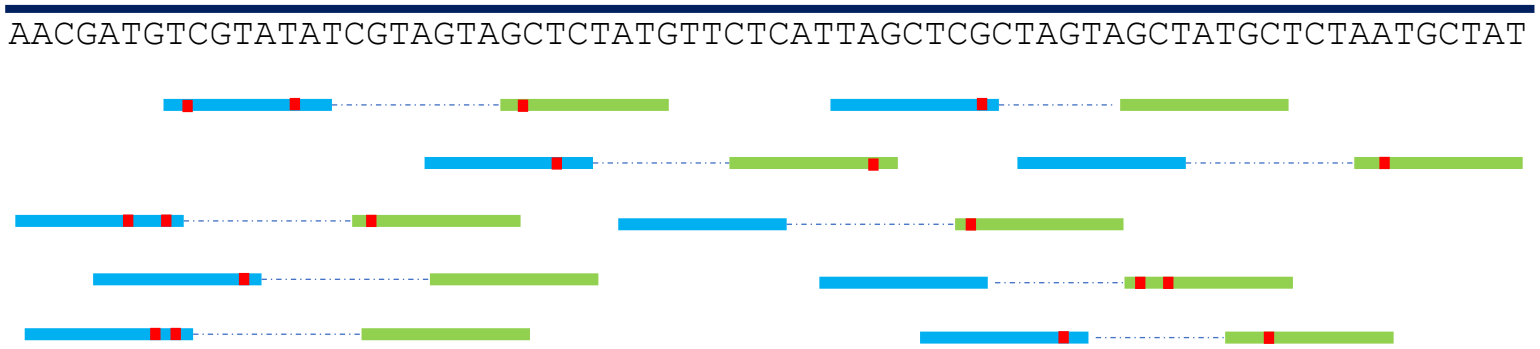
Łukasz Roguski, Idoia Ochoa, Mikel Hernaez, Sebastian Deorowicz; FaStore: a space-saving solution for raw sequencing data, *Bioinformatics*, Volume 34, Issue 16, 15 August 2018, Pages 2748–2756

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**

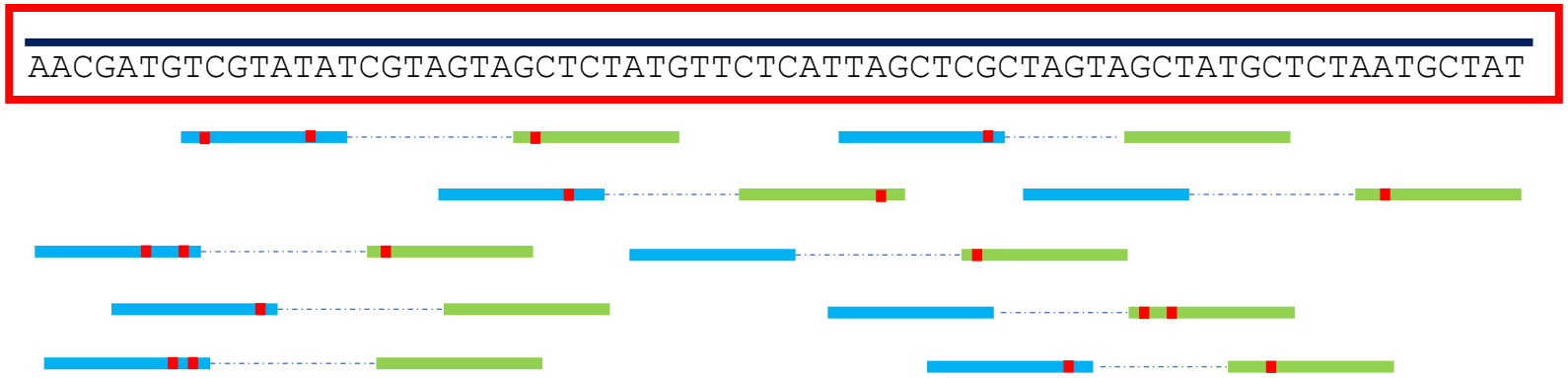
Łukasz Roguski, Idoia Ochoa, Mikel Hernaez, Sebastian Deorowicz; FaStore: a space-saving solution for raw sequencing data, *Bioinformatics*, Volume 34, Issue 16, 15 August 2018, Pages 2748–2756

Key idea



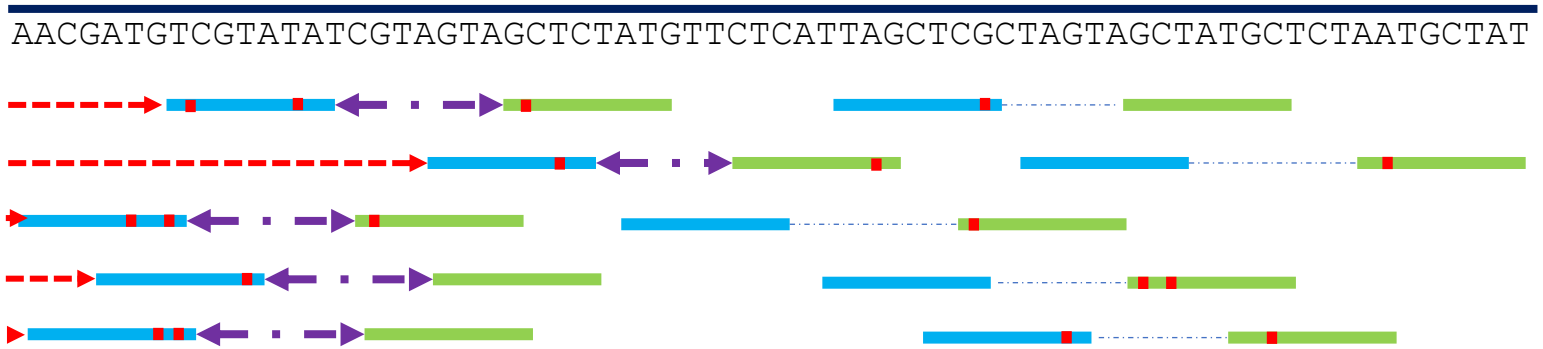
- Storing reads equivalent to

Key idea



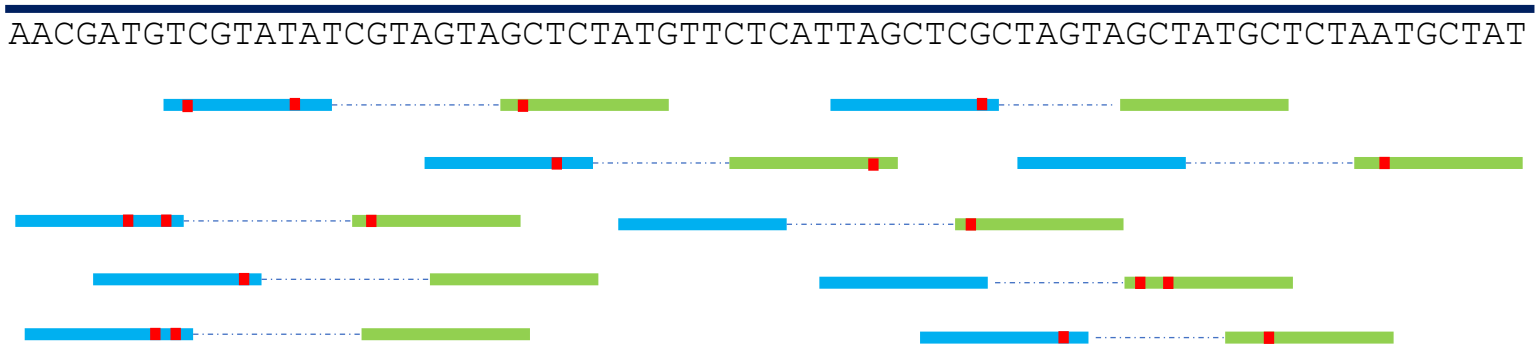
- Storing reads equivalent to
  - Store genome

Key idea



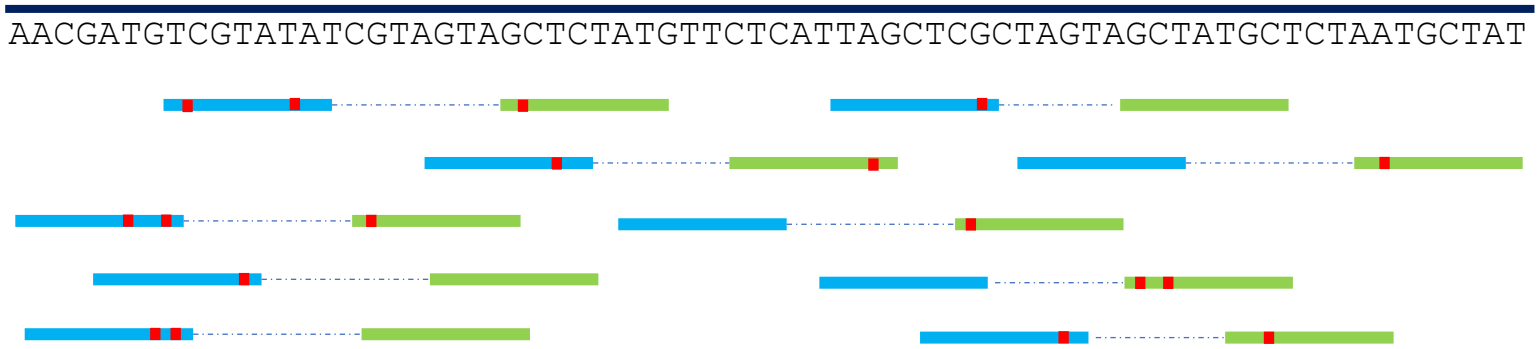
- Storing reads equivalent to
  - Store genome
  - Store read positions in genome (+ gap between paired reads)

Key idea



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

Key idea



- 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?

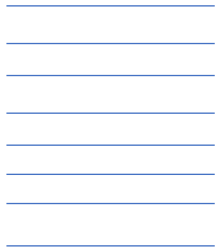
Key idea

- 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

Key idea

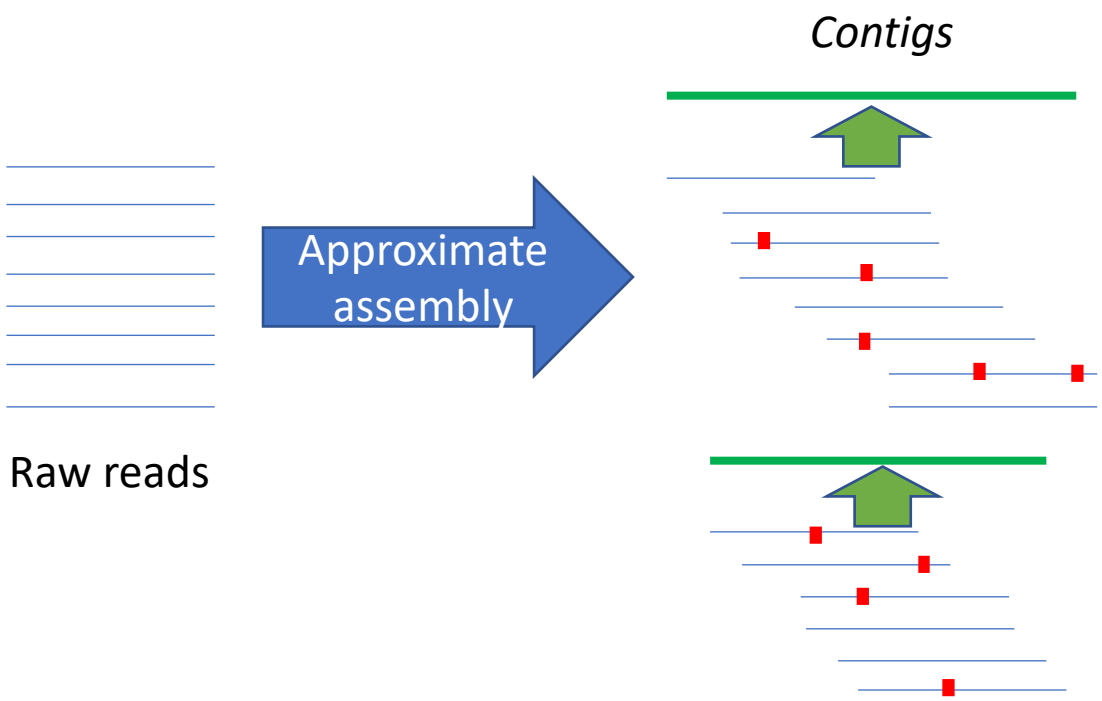
- But... How to get the genome from the reads?
- Genome assembly too expensive - big challenges:
  - resolve repeats
  - get very long pieces of genome from shorter assemblies
- Solution: Don’t need perfect assembly for compression!

SPRING workflow

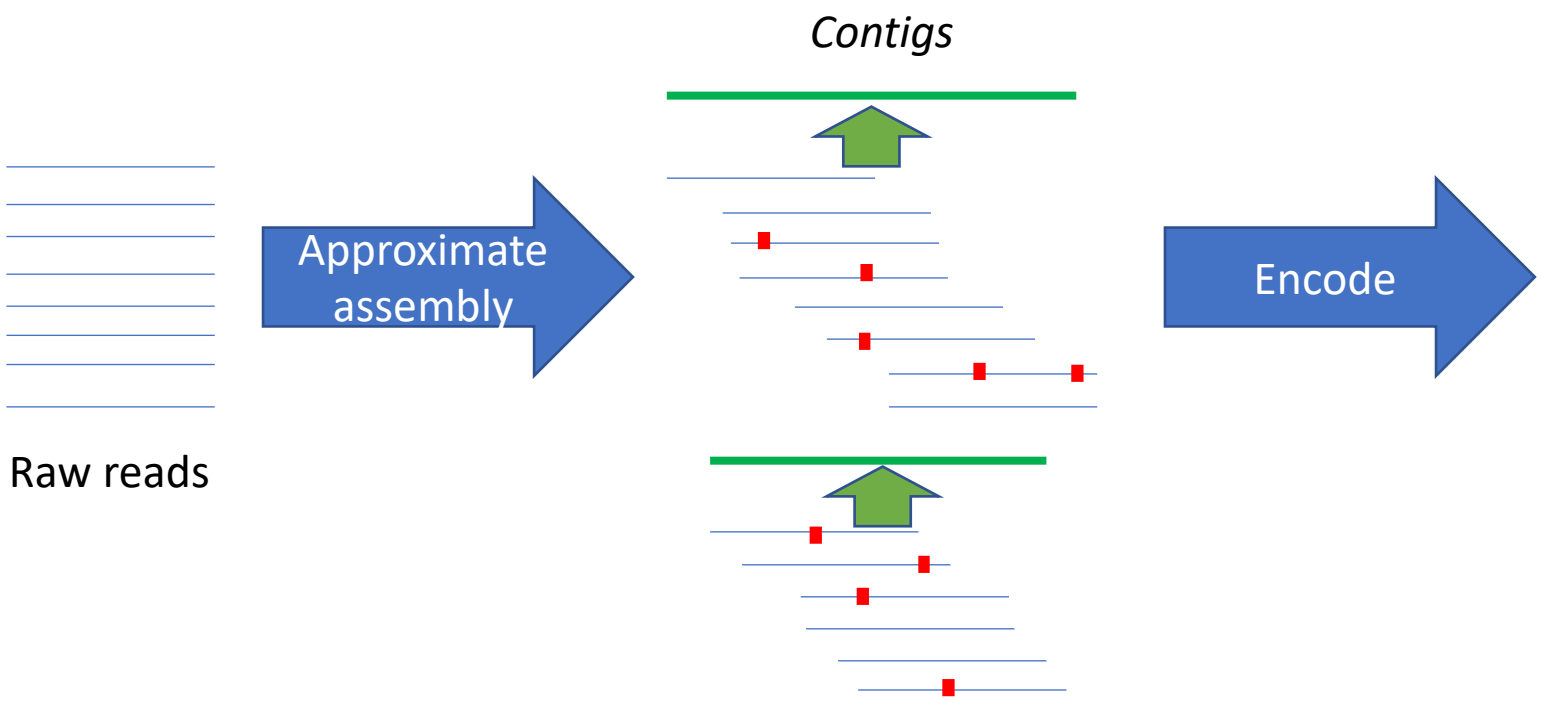


Raw reads

SPRING workflow

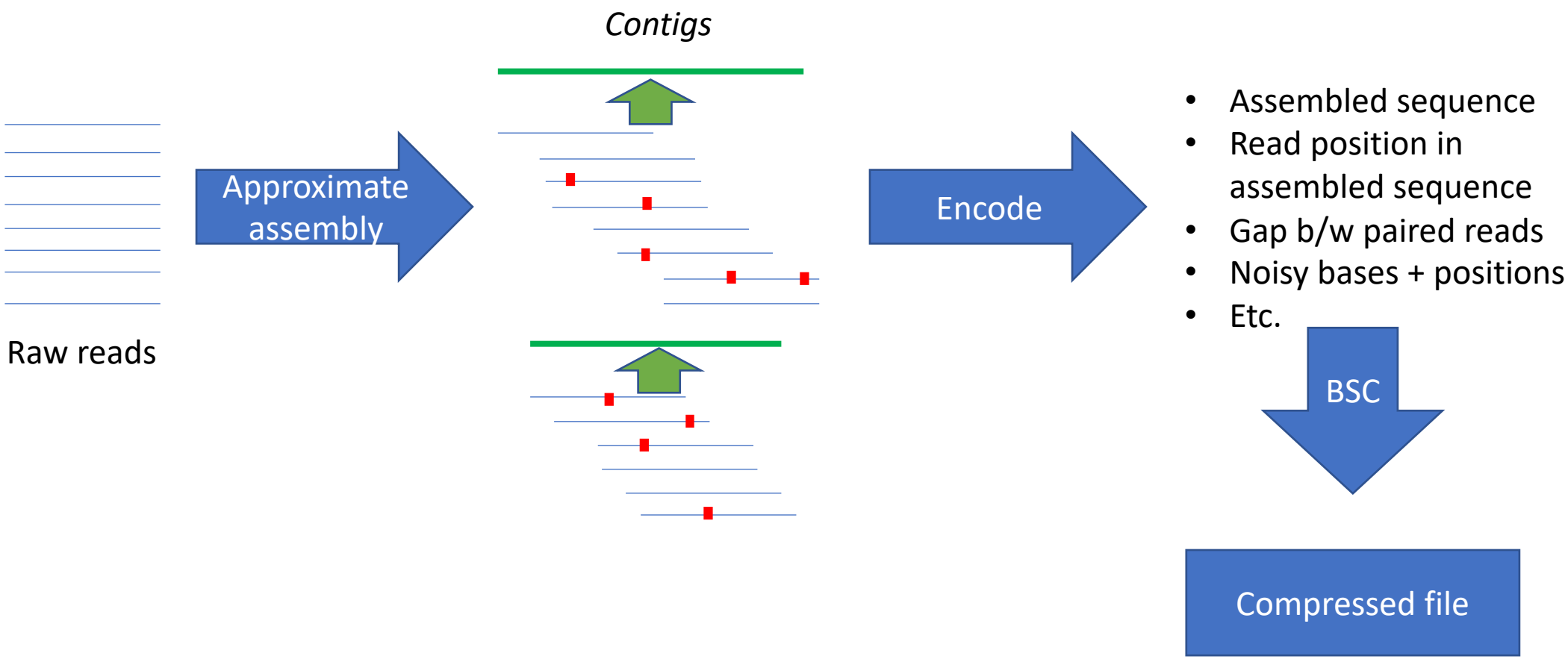


SPRING workflow



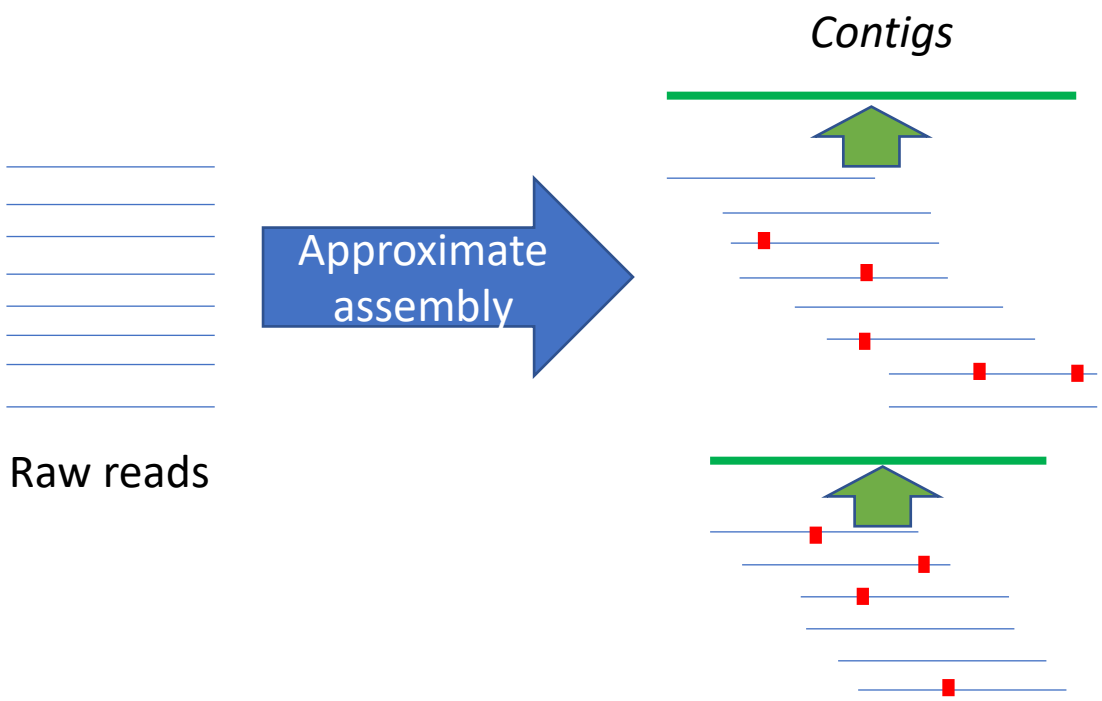
- Assembled sequence
- Read position in assembled sequence
- Gap b/w paired reads
- Noisy bases + positions
- Etc.

SPRING workflow



https://github.com/IlyaGrebnov/libbsc

SPRING workflow



Encode

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

BSC

Compressed file

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*

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.

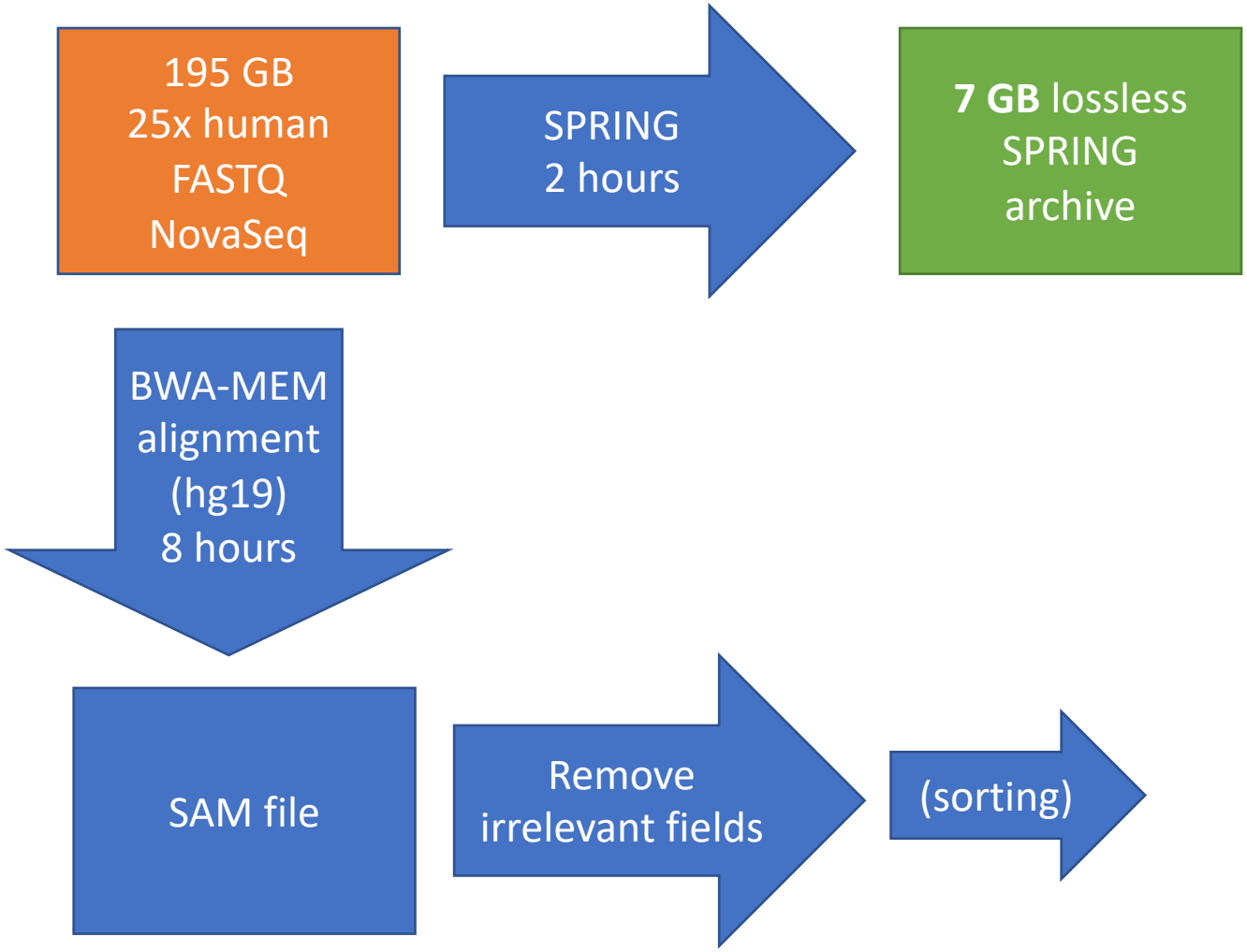
1. Bonfield, James K., and Matthew V. Mahoney. "Compression of FASTQ and SAM format sequencing data." *PloS one* 8.3 (2013): e59190.

SPRING vs. reference-based compression

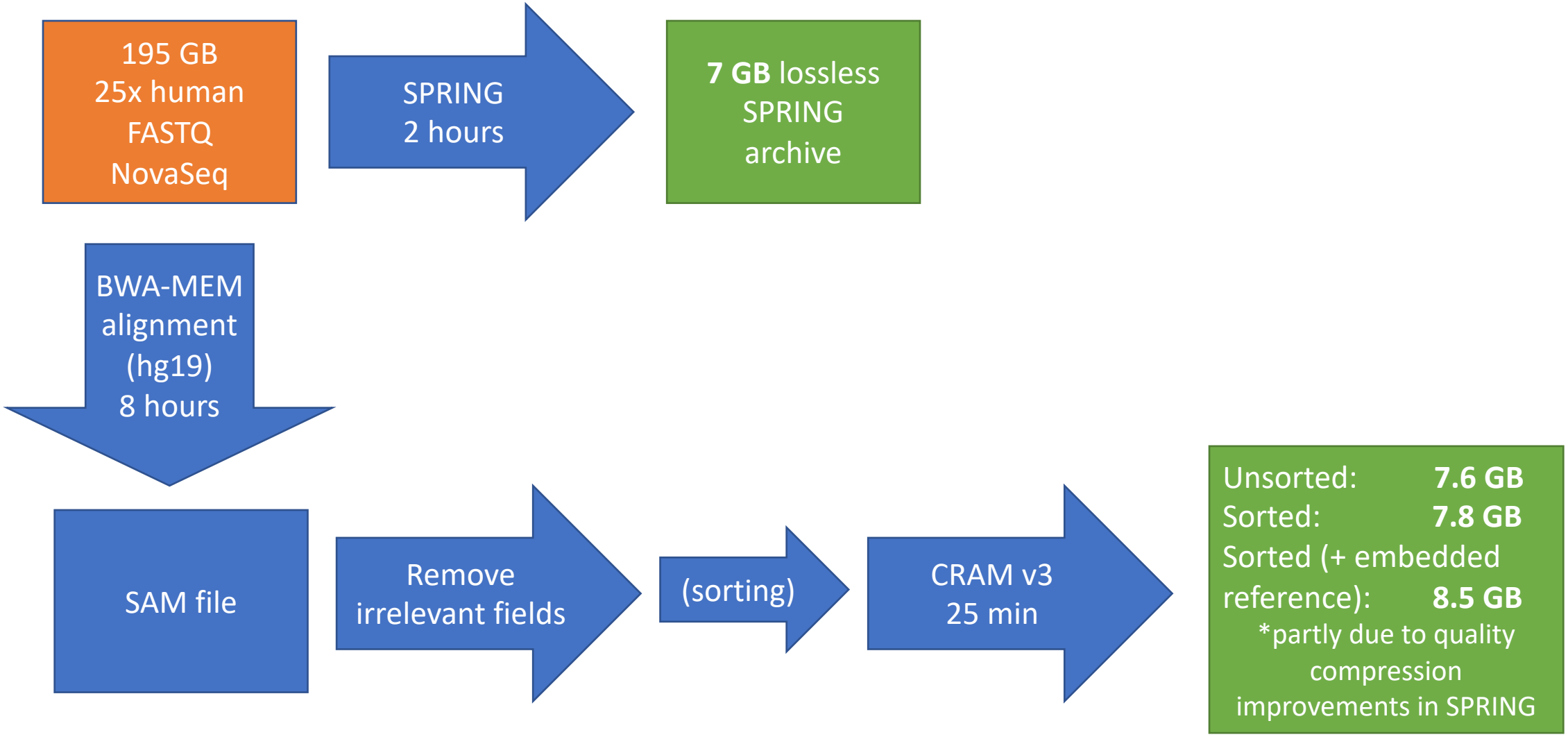
195 GB
25x human
FASTQ
NovaSeq

SPRING vs. reference-based compression

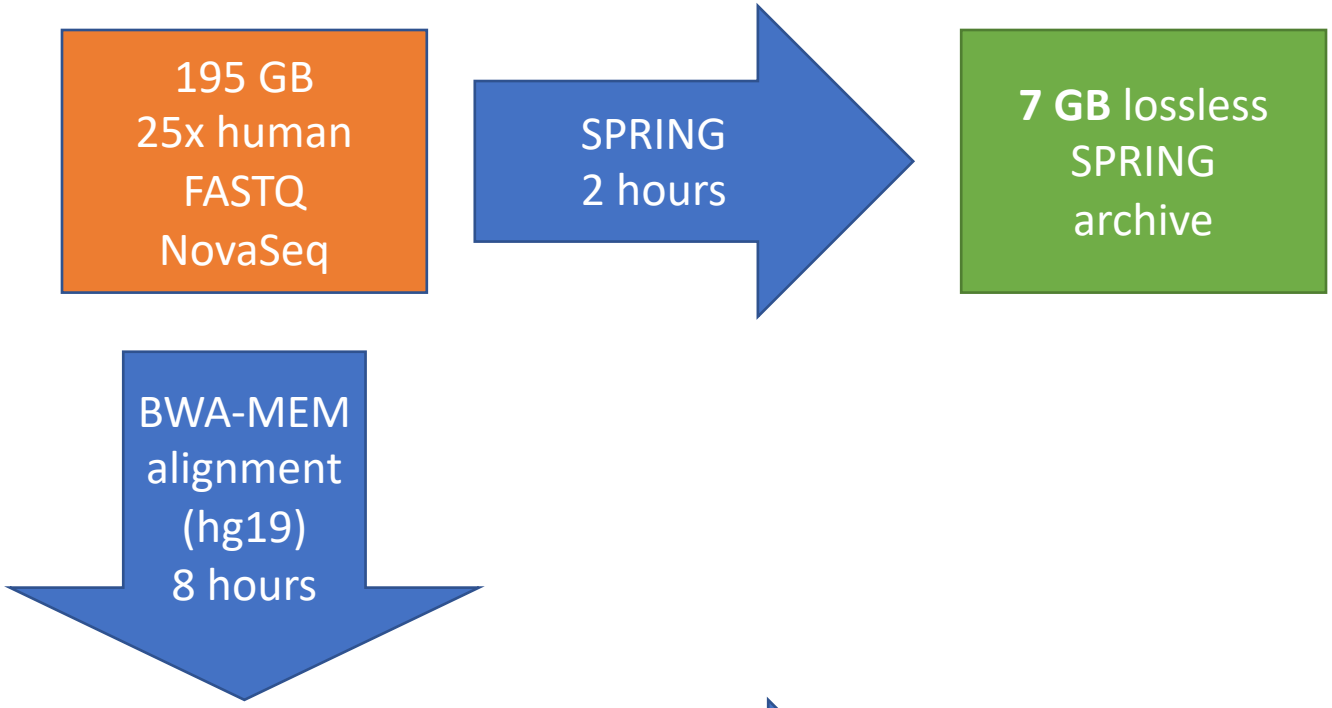
SPRING vs. reference-based compression



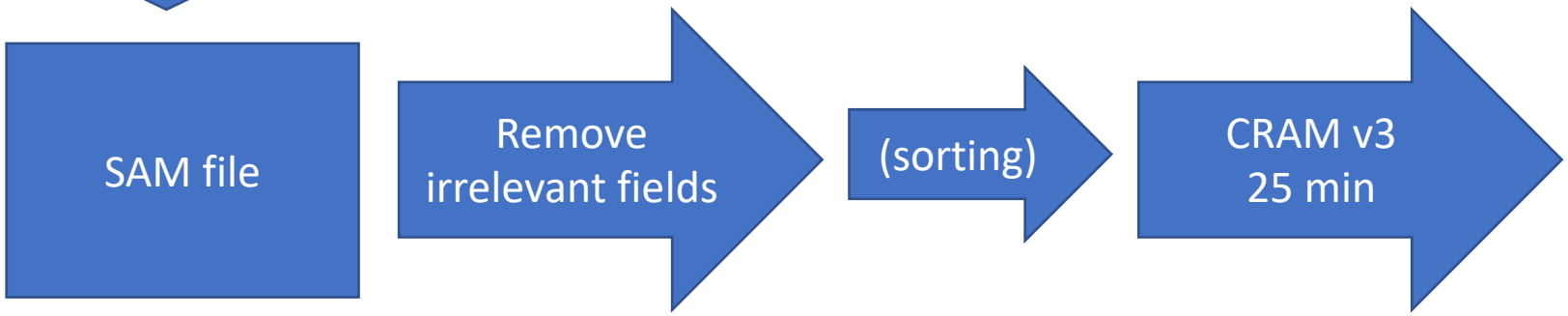
SPRING vs. reference-based compression



SPRING vs. reference-based compression



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



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

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).

Other approaches for FASTQ compression

- gzip/bzip2

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).

Other approaches for FASTQ compression

- gzip/bzip2
- Context-based arithmetic coding: DSRC 2, Fqzcomp, Quip

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).

Other approaches for FASTQ compression

- gzip/bzip2
- Context-based arithmetic coding: DSRC 2, Fqzcomp, Quip
- Assembly based: Leon, Quip, Assembletrie

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).

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

Recent FASTQ compressors

- minicom [1]: Use minimizers to construct large contigs (assemblies)
  - Slight improvement (5-10%) over SPRING on RNA-seq reads

1. Yuansheng Liu, Zuguo Yu, Marcel E Dinger, Jinyan Li, Index suffix–prefix overlaps by (*w*, *k*)-minimizer to generate long contigs for reads compression, *Bioinformatics*, Volume 35, Issue 12, June 2019, Pages 2066–2074.

Recent FASTQ compressors

- minicom [1]: Use minimizers to construct large contigs (assemblies)
  - Slight improvement (5-10%) over SPRING on RNA-seq reads
- 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

1. Yuansheng Liu, Zuguo Yu, Marcel E Dinger, Jinyan Li, Index suffix–prefix overlaps by (*w*, *k*)-minimizer to generate long contigs for reads compression, *Bioinformatics*, Volume 35, Issue 12, June 2019, Pages 2066–2074.
2. Deorowicz, Sebastian. "FQSqueezer: k-mer-based compression of sequencing data." *bioRxiv* (2019): 559807.

Recent FASTQ compressors

- minicom [1]: Use minimizers to construct large contigs (assemblies)
  - Slight improvement (5-10%) over SPRING on RNA-seq reads
- 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
- Both require significantly more time and memory than SPRING
  - Not tested on moderate to high coverage human datasets

1. Yuansheng Liu, Zuguo Yu, Marcel E Dinger, Jinyan Li, Index suffix–prefix overlaps by (*w*, *k*)-minimizer to generate long contigs for reads compression, *Bioinformatics*, Volume 35, Issue 12, June 2019, Pages 2066–2074.
2. Deorowicz, Sebastian. "FQSqueezer: k-mer-based compression of sequencing data." *bioRxiv* (2019): 559807.

SPRING as a practical tool

SPRING as a practical tool

SPRING as a practical tool

- ~1.6x better compression than FaStore with similar time/memory

SPRING as a practical tool

- ~1.6x better compression than FaStore with similar time/memory
- Easy to use with support for:
  - Lossless and lossy modes
  - Variable length reads, long reads, etc.
  - Random access
  - Scalable to large datasets

SPRING as a practical tool

- ~1.6x better compression than FaStore with similar time/memory
- Easy to use with support for:
  - Lossless and lossy modes
  - Variable length reads, long reads, etc.
  - Random access
  - Scalable to large datasets
- Github: https://github.com/shubhamchandak94/SPRING/

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec
- Third generation sequencing technologies (e.g., nanopore):

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec
- Third generation sequencing technologies (e.g., nanopore):
  - Long reads, lots of insertions and deletion errors
    - Hash based approximate assembly doesn’t extend immediately

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec
- Third generation sequencing technologies (e.g., nanopore):
  - Long reads, lots of insertions and deletion errors
    - Hash based approximate assembly doesn’t extend immediately
  - New types of raw data – e.g., raw current signal for nanopore sequencing
    - Need huge amounts of space and typically retained for further analysis

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec
- Third generation sequencing technologies (e.g., nanopore):
  - Long reads, lots of insertions and deletion errors
    - Hash based approximate assembly doesn’t extend immediately
  - New types of raw data – e.g., raw current signal for nanopore sequencing
    - Need huge amounts of space and typically retained for further analysis
- Time and memory efficient tool with compression close to SPRING:

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec
- Third generation sequencing technologies (e.g., nanopore):
  - Long reads, lots of insertions and deletion errors
    - Hash based approximate assembly doesn’t extend immediately
  - New types of raw data – e.g., raw current signal for nanopore sequencing
    - Need huge amounts of space and typically retained for further analysis
- Time and memory efficient tool with compression close to SPRING:
  - Disk based strategies (like Orcom/FaStore)

Next steps

- Currently integrating SPRING with genie, an upcoming open source MPEG-G codec
- Third generation sequencing technologies (e.g., nanopore):
  - Long reads, lots of insertions and deletion errors
    - Hash based approximate assembly doesn't extend immediately
  - New types of raw data – e.g., raw current signal for nanopore sequencing
    - Need huge amounts of space and typically retained for further analysis
- Time and memory efficient tool with compression close to SPRING:
  - Disk based strategies (like Orcom/FaStore)
  - When reference is available, can do fast and approximate alignment

Thank you!

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 download: https://github.com/shubhamchandak94/Spring
- genie (open source MPEG-G codec – *under development*): https://github.com/mitogen/genie