ELPREP PERFORMANCE ACROSS PROGRAMMING LANGUAGES

PASCAL COSTANZA

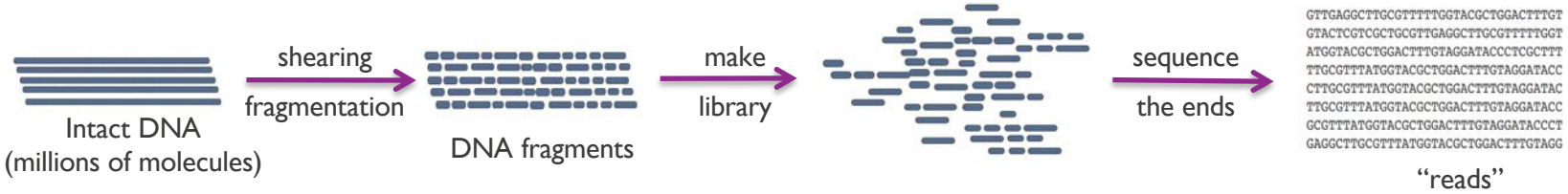
CHARLOTTE HERZEEL

FOSDEM, BRUSSELS, BELGIUM, FEBRUARY 3, 2018

imec is
the world-leading R&D and innovation hub
in nanoelectronics and digital technology.

WHAT IS NEXT-GENERATION SEQUENCING?

- Next-generation sequencing = massively parallel sequencing of short reads



- Sequencing is typically performed at 30-50x coverage, tumor sequencing at 80-100x
- Data generated per sample:
  - Raw data: 50-120GB compressed (WGS), 5-15GB (WES)
  - Variant data: ~1GB, ~200MB compressed

Illumina HiSeq X image courtesy of Illumina, Inc.

SEQUENCING

AAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGGGTTGATTCCACACCCCCGCCCGGCACCCGCGTCCGC

TTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGGG
CCTGCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCT
TGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTG
GCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCC
TGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTG
CTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTG
CTGCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCTG
CCCCTGCCCTCAACAAGATGTTTTGCCAACTGGCCAAGAC
ATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGT
ATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGT
GTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGG
ACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCA
CAACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTG
CAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAG
CAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAG
GATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTG
CAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAG
TGCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGC
CCTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCT
AAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGC
AGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCT

imec

MAPPING: ALIGNING READS TO A REFERENCE

AAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGGGTTGATTCCACACCCCCGCCCGGCACCCGCGTCCGC

TTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGGG
CCTGCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCT
TGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAGCTGTG
GCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCC
TGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAGCTGTG
CTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTG
CTGCCCTCAACAAGATGTTTTGCCTACTGGCCAAGACCTG
CCCCTGCCCTCAACAAGATGTTTTGCCTACTGGCCAAGAC
ATGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAGCTGT
ATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGT
GTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGG
ACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCA
CAACAAGATGTTTTGCCTACTGGCCAAGACCTGCCCTGTG
CAAGATGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAG
CAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAG
GATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTG
CAAGATGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAG
TGCCCTCAACAAGATGTTTTGCCTACTGGCCAAGACCTGC
CCTCAACAAGATGTTTTGCCTACTGGCCAAGACCTGCCCT
AAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGC
AGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCT

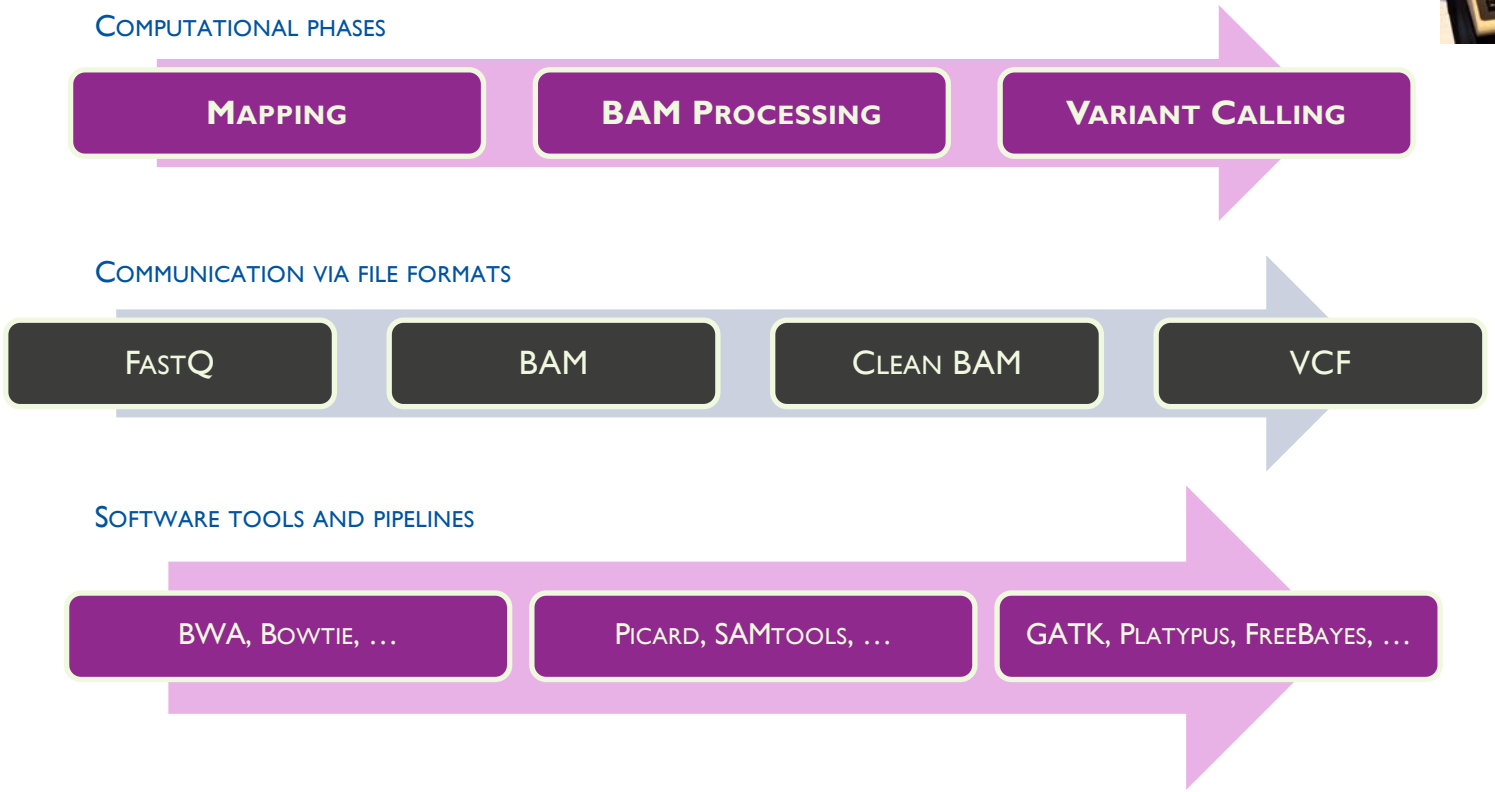
VARIANT CALLING: LOOKING FOR DIFFERENCES

```
AAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGGGTTGATTCCACACCCCCGCCCGGCACCCGCGTCCGC
1       TTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGGG
2             CCTGCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCT
3                                TGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAGCTGTG
4                  GCCCTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCC
5                                TGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAGCTGTG
6                     CTCAACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTG
7                CTGCCCTCAACAAGATGTTTTGCCTACTGGCCAAGACCTG
8              CCCCTGCCCTCAACAAGATGTTTTGCCTACTGGCCAAGAC
9                              ATGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAGCTGT
10                             ATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGT
11                               GTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTGTGG
12                         ACAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCA
13                       CAACAAGATGTTTTGCCTACTGGCCAAGACCTGCCCTGTG
14                         CAAGATGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAG
15                         CAAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAG
16                            GATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGCTG
17                         CAAGATGTTTTGCCTACTGGCCAAGACCTGCCCTGTGCAG
18               TGCCCTCAACAAGATGTTTTGCCTACTGGCCAAGACCTGC
19                  CCTCAACAAGATGTTTTGCCTACTGGCCAAGACCTGCCCT
20                        AAGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGC
21                        AGATGTTTTGCCAACTGGCCAAGACCTGCCCTGTGCAGC
```

Coverage depth: 22
A:11 T:11
Heterozygous SNP A/T

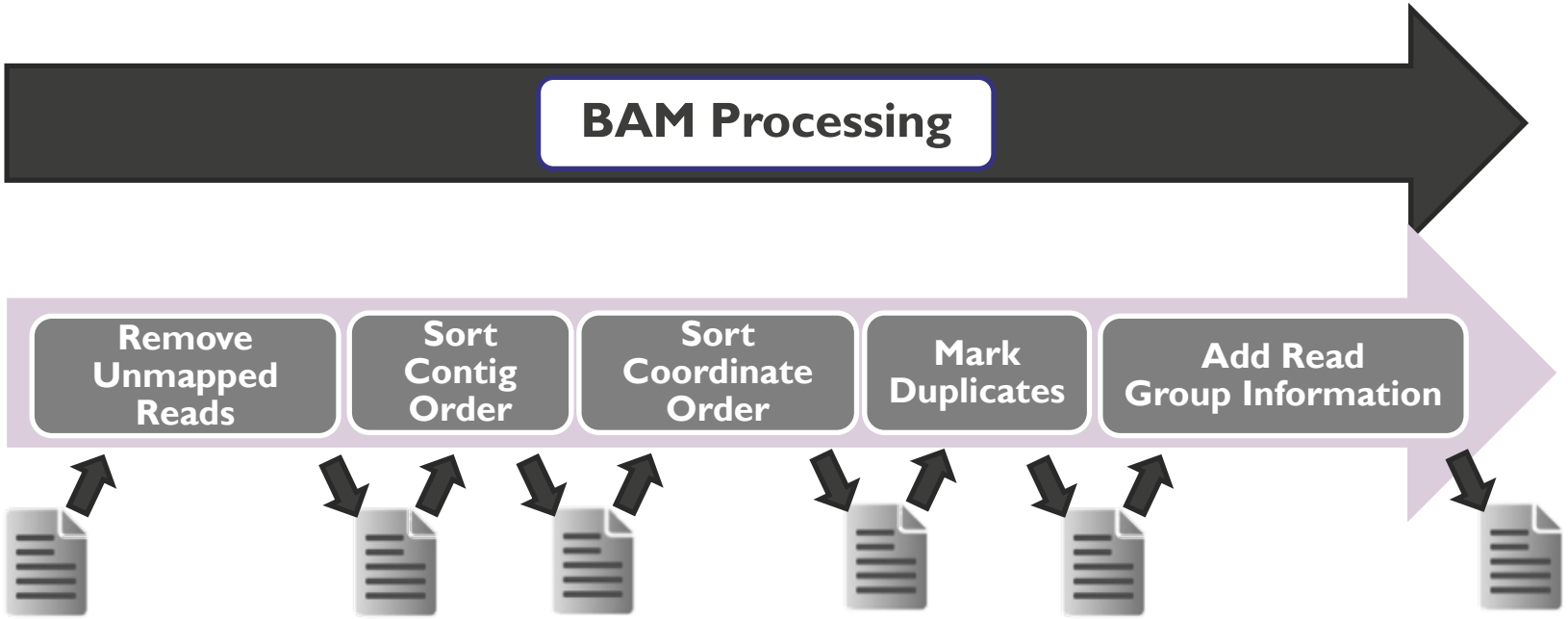
imec

THE COMPUTATIONAL PHASES OF DNA SEQUENCING



imec

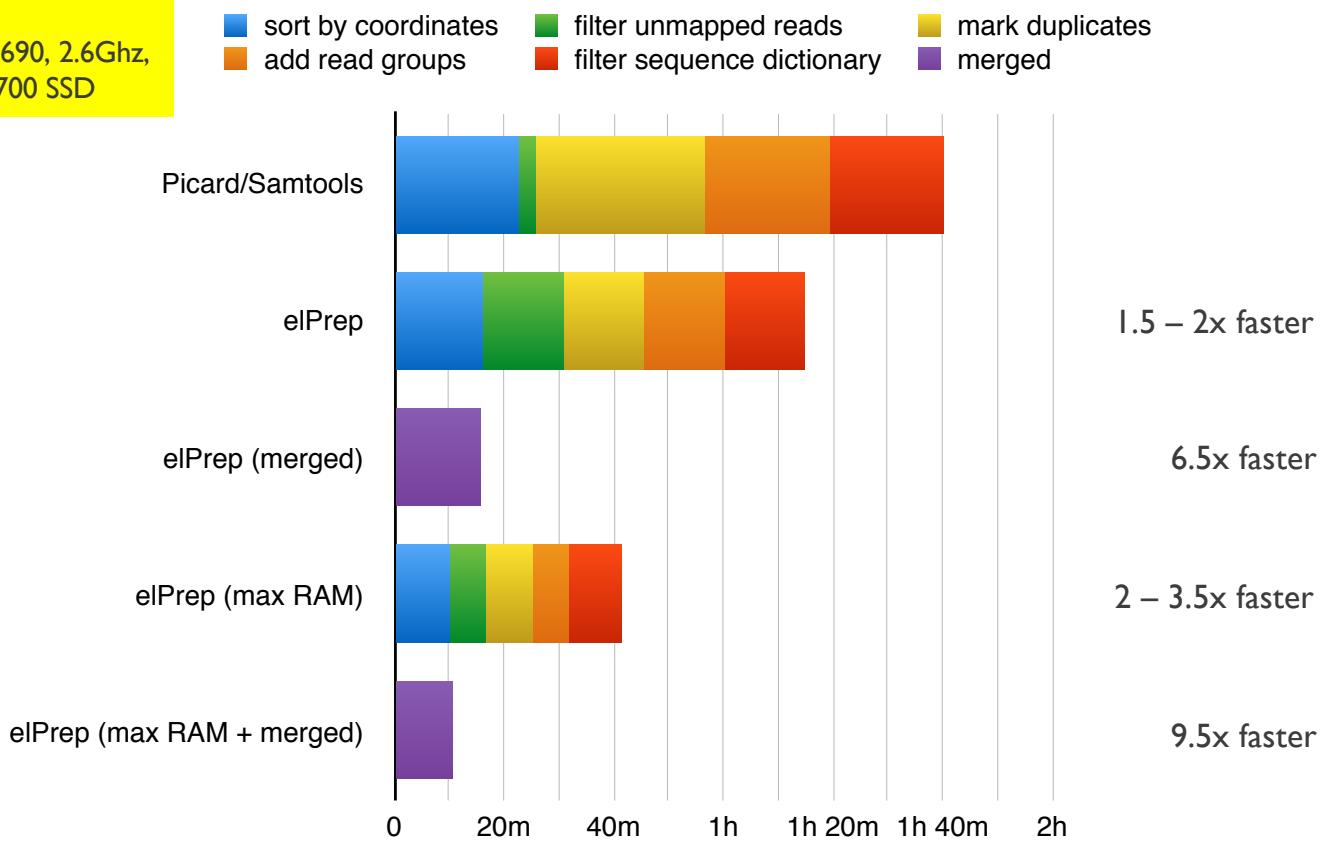
BAM PROCESSING WITH ELPREP



BAM PROCESSING WITH ELPREP

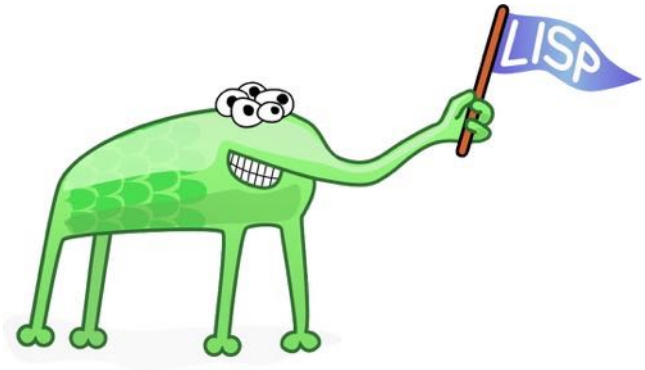
BENCHMARKS: JANSSEN PHARMACEUTICA PROTOCOL

NA12878 WES
2x12-core Intel Xeon E5-2690, 2.6Ghz,
256GB RAM, 2TB Intel P3700 SSD



ELPREP IN COMMON LISP
VERY BRIEF HISTORY

- Originally implemented in Common Lisp by Charlotte Herzeel, with help from Pascal Costanza.
- Initial version developed over the course of 6 months, with several major design changes along the way.



Alien Lisp Mascot by Conrad Barski, M.D.

MEMORY MANAGEMENT IN ELPREP

- Memory management is a key performance issue in elPrep.
  - All Common Lisps known to us use a sequential stop-the-world garbage collector.
    - This is especially bad for multi-threaded programs due to Amdahl's law.
    - Charlotte tricked the garbage collector into not interfering with parallel phases, but the solution is not intuitive and not portable.
  - A lot of effort went into elPrep to:
    - Minimize memory use for representing the data.
    - Manual control of memory management.

MEMORY MANAGEMENT IN ELPREP

- Memory management is a key performance issue in elPrep:
  - Two questions:
    - Did we achieve the best result possible?
    - Is there an easier way to achieve the same or a better result?
  - Unexplored memory management choices:
    - Concurrent garbage collection
    - Reference counting
    - …but they need support from the programming language and its implementation.

ELPREP IN OTHER PROGRAMMING LANGUAGES

- Concurrent parallel garbage collection
  - Garbage collection as much as possible in separate threads, to avoid disruption of the main program.
  - Beneficial because it reduces negative impact of Amdahl's law.
  - Mature languages known to us at the start of experiment:
    - **Java**
    - **Go** (concurrent GC introduced in 2016)

ELPREP IN OTHER PROGRAMMING LANGUAGES

- Reference counting
  - No stopping of the world by design.
  - Synchronization spread over whole program due to atomic operations on reference counts.
  - More advanced implementation schemes known in literature,
    but no mature language known to us.
  - Mature languages with reference counting known to us:
    - **C++11/14/17** (through std::shared_ptr)
    - Objective-C
    - Swift
    - Rust
  - Objective-C and Swift discarded, because they don't synchronize reference counting.
  - Rust allows for atomic compare-and-swap only on unsafe pointers.

ELPREP IN VARIOUS PROGRAMMING LANGUAGES
EXPERIMENTAL SETUP

- Experimental setup based on https://github.com/ExaScience/elprep/tree/master/demo
  - Input data set: SRR1611184, a high-coverage whole-exome sequencing of NA12878.
  - elPrep pipeline consisting of five steps:
    1. Filter unmapped reads
    2. Replace reference sequence dictionary
    3. Replace read group
    4. Mark duplicates
    5. Sort by coordinate order

- Hardware environment:
  - Intel Xeon E5-2699 v4 (Broadwell)
    - 22 cores x 2 sockets = 88 threads
    - 768 GB RAM

RESULTS

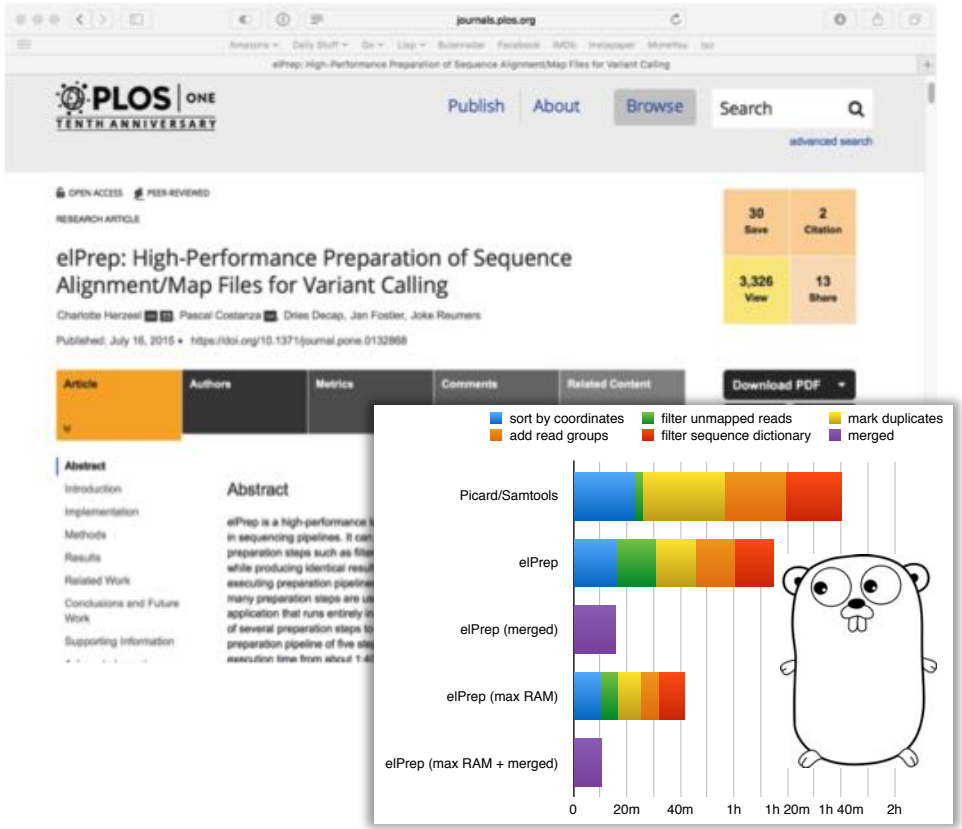
- C++
  - GNU g++ 6.3
  - Intel TBB 4.4
  - gperftools 2.5
  - 13:38 mins @ 227.4 GB RAM
- Java (JDK 1.8)
  - ConcMarkSweepGC
  - G1GC
  - ParallelGC
  - 15:05 mins @ 293.4 GB RAM
  - 11:57 mins @ 358.1 GB RAM
  - 11:07 mins @ 477.3 GB RAM
- Go 1.7
  - default settings
  - 10:20 mins @ 233.7 GB RAM

ELPREP: A HIGH-PERFORMANCE TOOL FOR SEQUENCING

- High-performance tool for preparing SAM files for variant calling.
- Multi-threaded application that runs entirely in RAM and merges multiple steps to avoid repeated file I/O.
- Can improve performance by a factor of up to x10 compared to standard tools.

- elPrep 3.0 implemented in Go
- Open-sourced (BSD) in September 2017
  - https://github.com/exascience/elprep

- Pargo library for parallel programming in Go
  - https://github.com/exascience/pargo



Go Gopher image by Renee French, CC BY 3.0, https://creativecommons.org/licenses/by/3.0/


imec
embracing a better life