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
MAPPING: ALIGNING READS TO A REFERENCE

AAAGATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGCTGTGGGTTGATTCCACACCCCGGCGGCCGCACCCGCGTCGCGC

TTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGCTGTGGG
CCTGCCCCTCAACAAAGATGTTTTTGCCCAACTGGCCAAGACCT
TGTTTTTGCCCTACTGGCCAAGACCTTGCCCTGTGCAGCTGTG
GCCCTCAACAAAGATGTTTTTGCCCAACTGGCCAAGACCTGC
TGTTTTTGCCCTACTGGCCAAGACCTTGCCCTGTGCAGCTGTG
CTCAACAAAGATGTTTTTGCCCAACTGGCCAAGACCTGCCT
CTGCCCTCAACAAAGATGTTTTTGCCCTACTGGCCAAGACCTG
CCCTGCCCTCAACAAAGATGTTTTTGCCCTACTGGCCAAGAC
ATGTTTTTGCCCTACTGGCCAAGACCTTGCCCTGTGCAGCTGTG
ATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGCTGTG
GGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGCTGTG
AACAAGATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGC
CAACAAAGATGTTTTTGCCCTACTGGCCAAGACCTTGCCCTGTG
CAAGATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAG
CAAGATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAG
GATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGCTG
CAAGATGTTTTTGCCCTACTGGCCAAGACCTTGCCCTGTGCAG
TGCCCTCAACAAAGATGTTTTTGCCCTACTGGCCAAGACCTG
CCTCAACAAAGATGTTTTTGCCCTACTGGCCAAGACCTGCCCT
AAGATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGC
AGATGTTTTTGCCCAACTGGCCAAGACCTTGCCCTGTGCAGCTT
## Variant Calling: Looking for Differences

<table>
<thead>
<tr>
<th></th>
<th>TTTTGCCCAACTGGCCAAGACCTGCGCCTTGAGCTGTGGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CCTGCCCTCAACAAGATGGTTTTGGCCCAACTGGCCAAGACCTG</td>
</tr>
<tr>
<td>3</td>
<td>TTTTTGCCCTACTGGCCAAGACCTGCCCTTGAGCTGTGG</td>
</tr>
<tr>
<td>4</td>
<td>GGCCTCAACAAGATGGTTTTGGCCCAACTGGCCAAGACCTGC</td>
</tr>
<tr>
<td>5</td>
<td>TTTTTGCCCTACTGGCCAAGACCTGCCCTTGAGCTGTGG</td>
</tr>
<tr>
<td>6</td>
<td>CTCAACAAGATGGTTTTGGCCCAACTGGCCAAGACCTGCC</td>
</tr>
<tr>
<td>7</td>
<td>CTGCCCTCAACAAGATGGTTTTGGCCCTACTGGCCAAGACCTGC</td>
</tr>
<tr>
<td>8</td>
<td>CCCCTGCCCTCAACAAGATGGTTTTGGCCCTACTGGCCAAGAC</td>
</tr>
<tr>
<td>9</td>
<td>ATTTTTGCCCTACTGGCCAAGACCTGCCCTTGAGCTGTGG</td>
</tr>
<tr>
<td>10</td>
<td>ATTTTTGCCCTACTGGCCAAGACCTGCCCTTGAGCTGTGG</td>
</tr>
<tr>
<td>11</td>
<td>GTTTTTGCCCTACTGGCCAAGACCTGCCCTTGAGCTGTGG</td>
</tr>
<tr>
<td>12</td>
<td>ACAAGATGGTTTTGGCCCAAGACCTGCCCTTGAGCT</td>
</tr>
<tr>
<td>13</td>
<td>CAACAAGATGGTTTTGGCCCTACTGGCCAAGACCTGCCCTTG</td>
</tr>
<tr>
<td>14</td>
<td>CAAGATGGTTTTGGCCCTACTGGCCAAGACCTGCCCTTGAG</td>
</tr>
<tr>
<td>15</td>
<td>CAAGATGGTTTTGGCCCTACTGGCCAAGACCTGCCCTTGAG</td>
</tr>
<tr>
<td>16</td>
<td>GATTTTTGCCCTACTGGCCAAGACCTGCCCTTGAGCT</td>
</tr>
<tr>
<td>17</td>
<td>CAAGATGGTTTTGGCCCTACTGGCCAAGACCTGCCCTTGAG</td>
</tr>
<tr>
<td>18</td>
<td>TGCCCTCAACAAGATGGTTTTGGCCCTACTGGCCAAGACCTGC</td>
</tr>
<tr>
<td>19</td>
<td>CCTCAACAAGATGGTTTTGGCCCTACTGGCCAAGACCTGCC</td>
</tr>
<tr>
<td>20</td>
<td>AAGATGGTTTTGGCCCAACTGGCCAAGACCTGCCCTTGAGC</td>
</tr>
<tr>
<td>21</td>
<td>AGATGGTTTTGGCCCAACTGGCCAAGACCTGCCCTTGAGC</td>
</tr>
</tbody>
</table>

Coverage depth: 22  
A:11 T:11  
Heterozygous SNP A/T
THE COMPUTATIONAL PHASES OF DNA SEQUENCING

**Computationals phases**

- Mapping
- BAM Processing
- Variant Calling

**Communication via file formats**

- FASTQ
- BAM
- Clean BAM
- VCF

**Software tools and pipelines**

- BWA, Bowtie, …
- Picard, SAMtools, …
- GATK, Platypus, FreeBayes, …

DEC VT100 image by Jason Scott, CC BY 2.0, https://creativecommons.org/licenses/by/2.0/
BAM PROCESSING WITH ELPREP

BAM Processing

Remove Unmapped Reads

Sort Contig Order

Sort Coordinate Order

Mark Duplicates

Add Read Group Information
BAM PROCESSING WITH ELPREP

BAM Processing

elPrep
**BENCHMARKS: JANSSEN PHARMACEUTICA PROTOCOL**

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

- Picard/Samtools
- elPrep
- elPrep (merged)
- elPrep (max RAM)
- elPrep (max RAM + merged)

**Sort by coordinates**: 0, 20m, 40m, 1h, 1h 20m, 1h 40m, 2h

- **Filter unmapped reads**: NA
- **Add read groups**: NA
- **Filter sequence dictionary**: NA
- **Mark duplicates**: NA
- **Merged**: NA

**Benchmarks**:

- Picard/Samtools: 1.5 – 2x faster
- elPrep: 6.5x faster
- elPrep (merged): 2 – 3.5x faster
- elPrep (max RAM): 9.5x faster

**System Configuration**:

- 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.
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 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.
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.
## EXPRESSION 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](https://github.com/exascience/elprep)
- Pargo library for parallel programming in Go
  - [https://github.com/exascience/pargo](https://github.com/exascience/pargo)
embracing a better life