Gamgee: A C++14 library for genomic data processing and analysis

Mauricio Carneiro, PhD
Group Lead, Computational Technology Development
Broad Institute

Talk breakdown

- An overview of genetics data and how complex disease research became a big data problem

- The first C++ example that steered us away from Java.

- Gamgee: the C++14 library memory model and examples

- Performance comparisons with the old Java framework.

- Discussion of C++11/14 features used in the library and how they affected development

To fully understand **one** genome we need **hundreds of thousands** of genomes

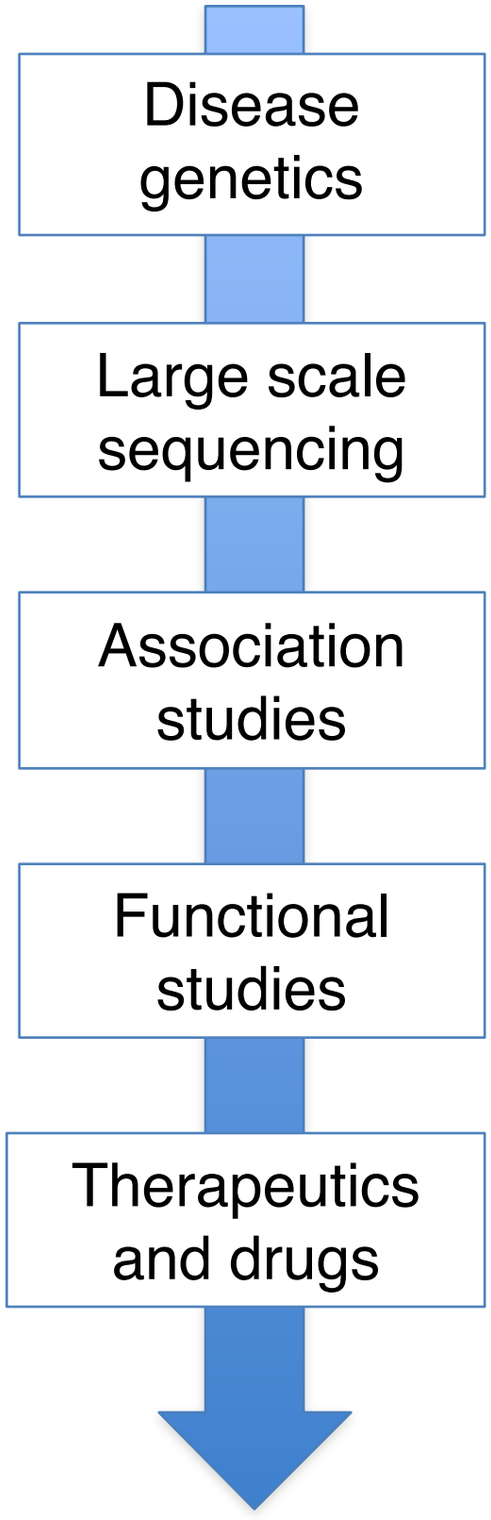
Rare Variant Association Study (RVAS)

vs

Common Variant Association Study (CVAS)

vs

Improving human health in 5 easy steps



Many simple and complex human diseases are heritable. Pick one.

Affected and unaffected individuals differ systematically in their genetic composition

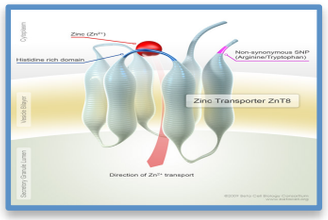
These systematic differences can be identified by comparing affected and unaffected individuals

These associated variants give insight into the biological mechanisms of disease

These insights can be used to intervene in the disease process itself

The Importance of Scale…Early Success Stories (at 1,000s of exomes)

Type 2 Diabetes

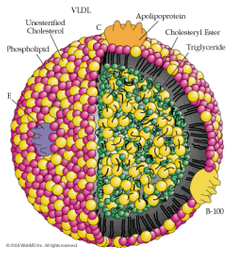


- 13,000 exomes
- SLC30A8
  (Beta-cell-specific Zn^{++} transporter)
- 3-fold protection against T2D!
- ***1 LoF per 1500 people***

Schizophrenia

- 5,000 exomes
- Pathways
  - Activity-regulated cytoskeletal (ARC) of post-synaptic density complex (PSD)
  - Voltage-gated Ca^{++} Channel
- 13-21% risk in carriers
- ***Collection of rare disruptive mutations (~1/10,000 carrier frequency)***

Coronary Heart Disease



- 3,700 exomes
- APOC3
- 2.5-fold protection from CHD
- ***4 rare disruptive mutations (~1 in 200 carrier frequency)***

Early Heart Attack

- 5,000 exomes
- APOA5
- 22% risk in carriers
- ***0.5% Rare disruptive / deleterious alleles***

Broad Institute in 2013

50
HiSeqs

10
MiSeqs

2
NextSeqs

14
HiSeq X

6.5
Pb of data

427
projects

180
people

2.1
Tb/day

* we also own 1 *Pacbio RS* and 4 *Ion Torrent* for experimental use

Broad Institute in 2013

44,130
exomes

2,484
exome express

2,247
genomes

2,247
assemblies

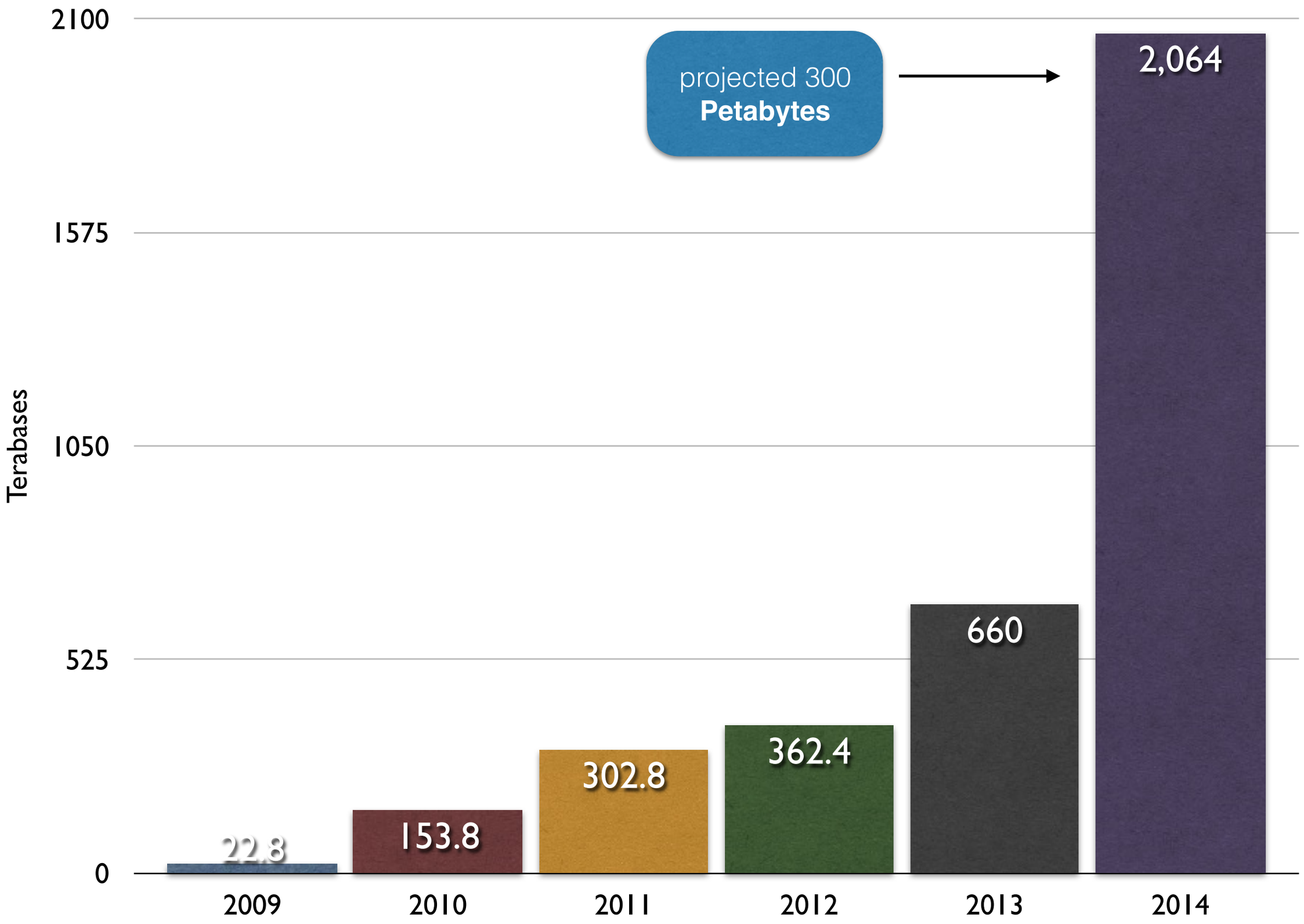
8,189
RNA

9,788
16S

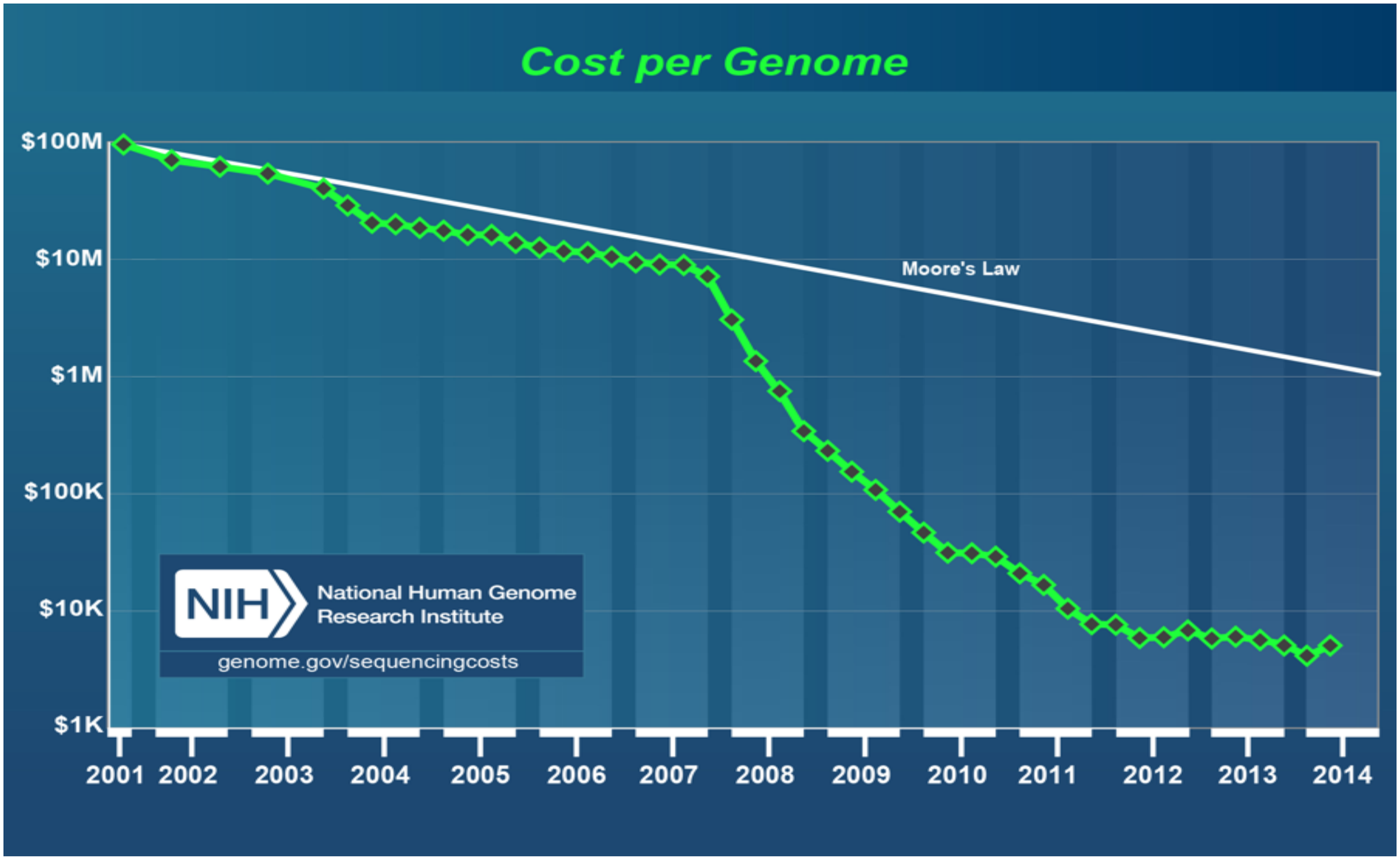
47,764
arrays

228
cell lines

Terabases of Data Produced by Year

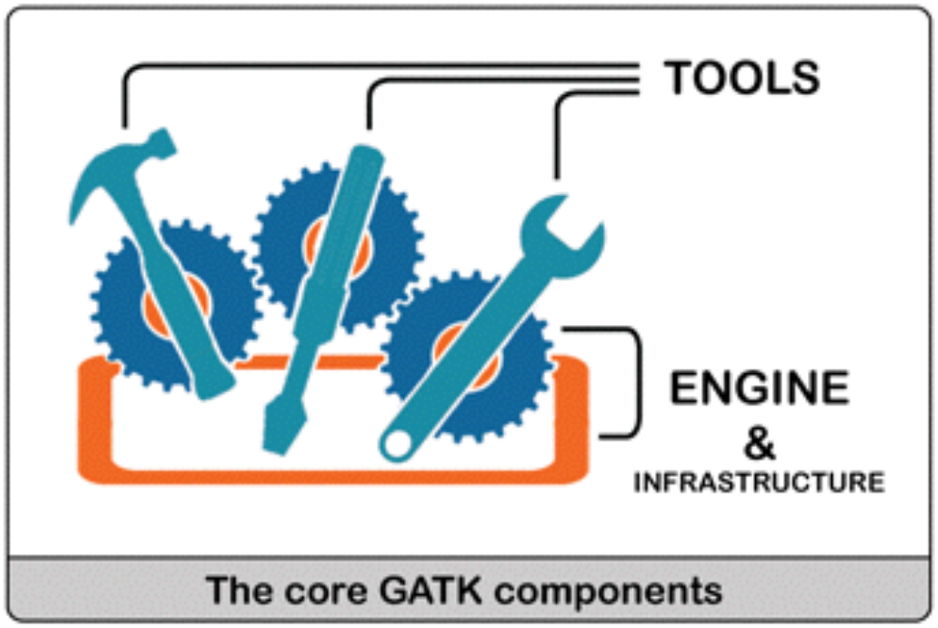


…and these numbers will continue to grow faster than Moore's law



GATK is both a toolkit and a programming framework, enabling NGS analysis by scientists worldwide

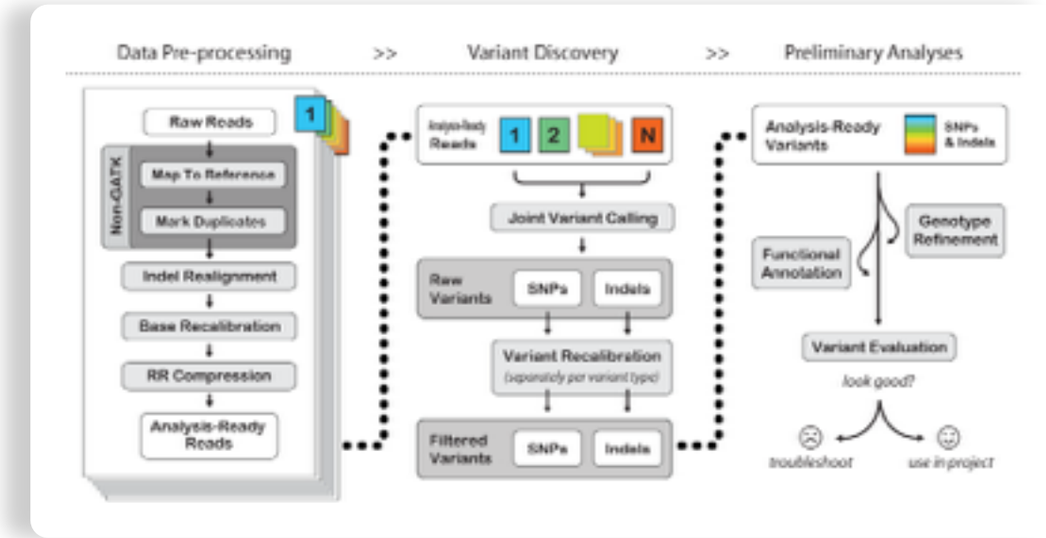
Toolkit & framework packages



The core GATK components

Toolkit

Best practices for variant discovery



Framework

MuTect, XHMM, GenomeSTRiP, ...

Tools developed on top of the GATK framework by other groups

Extensive online documentation & user support forum serving >10K users worldwide

http://www.broadinstitute.org/gatk

About
Overview of the GATK and the people behind it

Guide
Detailed documentation, guidelines and tutorials

Community
Forum for questions and announcements

Events
Materials from live and online events

iSeqTools

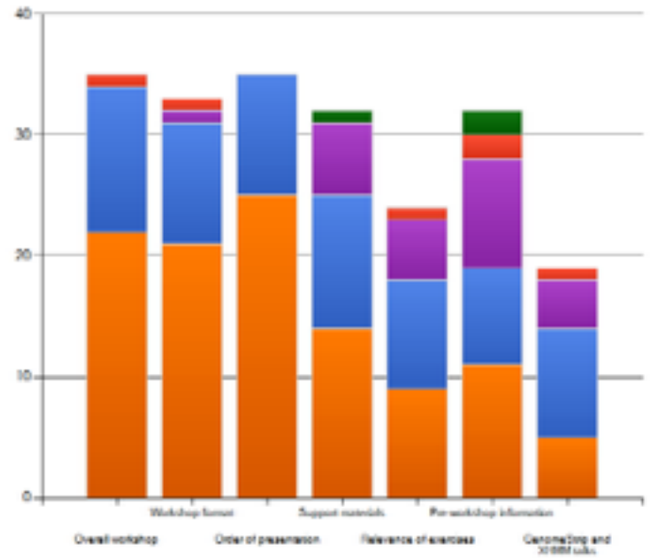
Workshop series educates local and worldwide audiences

Past:
- Dec 4-5 2012, Boston
- July 9-10 2013, Boston
- July 22-23 2013, Israel
- Oct 21-22 2013, Boston
- March 3-5 2014, Thailand
- June 6-9 2014, Belgium

Upcoming:
- Sep 17-18 2014, Philadelphia
- Oct 18-29 2014, San Diego

Format
- Lecture series (general audience)
- Hands-on sessions (for beginners)

Portfolio of workshop modules
- GATK Best Practices for Variant Calling
- Building Analysis Pipelines with Queue
- Third-party Tools:
  - GenomeSTRiP
  - XHMM



- High levels of satisfaction reported by users in polls
- Detailed feedback helps improve further iterations

Tutorial materials, slide decks and videos all available online through the GATK website, YouTube and iTunesU

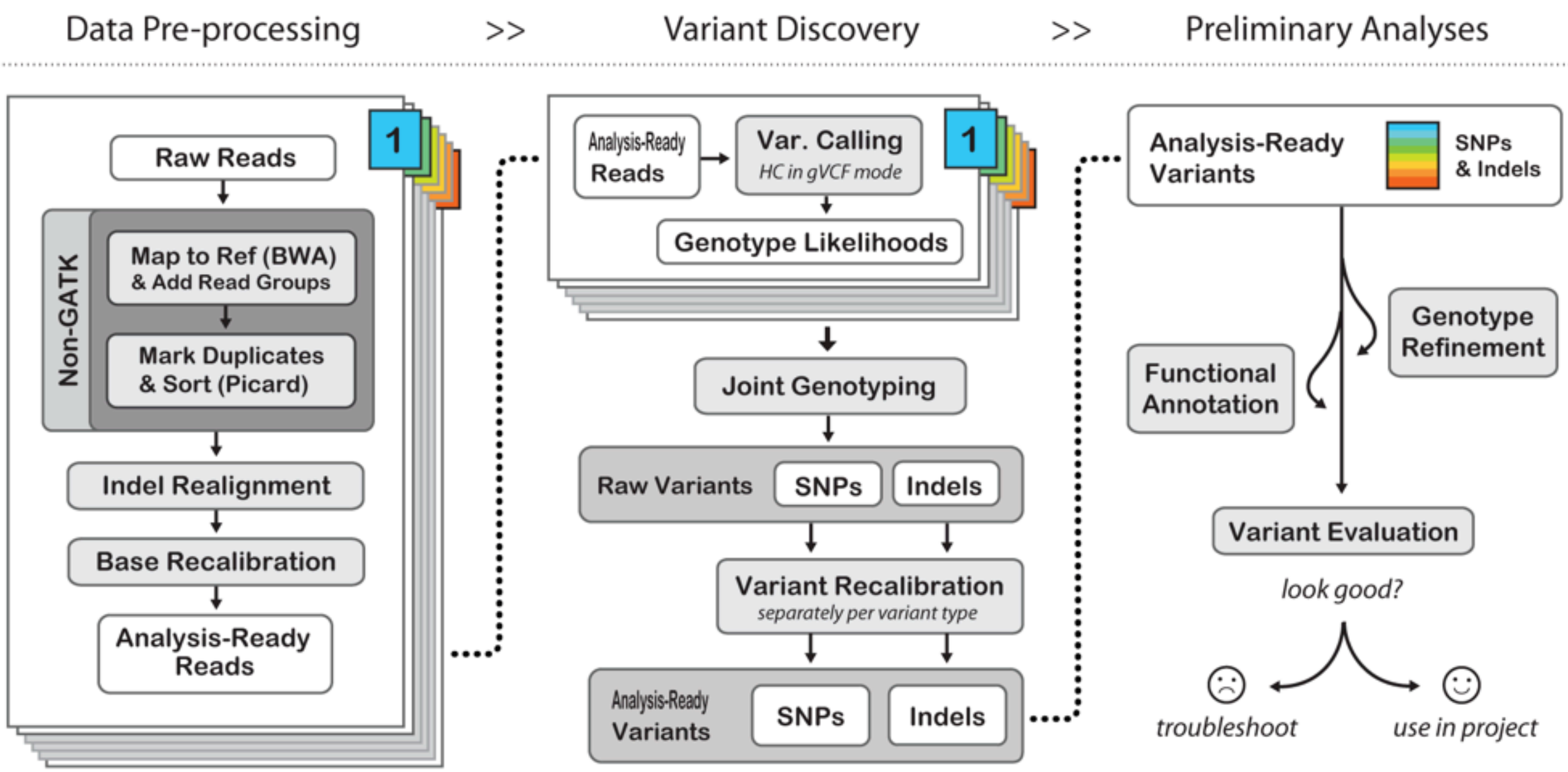
BroadE: Overview of GATK & best practices

by **broadinstitute** • 1 week ago • 1 view

Copyright Broad Institute, 2013. All rights reserved. The presentations below were filmed during the 2013 **GATK** Workshop, part of ...

NEW HD

We have defined the best practices for sequencing data processing



Auwera, GA et al. *Current Protocols in Bioinformatics* (2013)

To fully understand **one** genome we need **hundreds of thousands** of genomes

Rare Variant Association Study (RVAS)

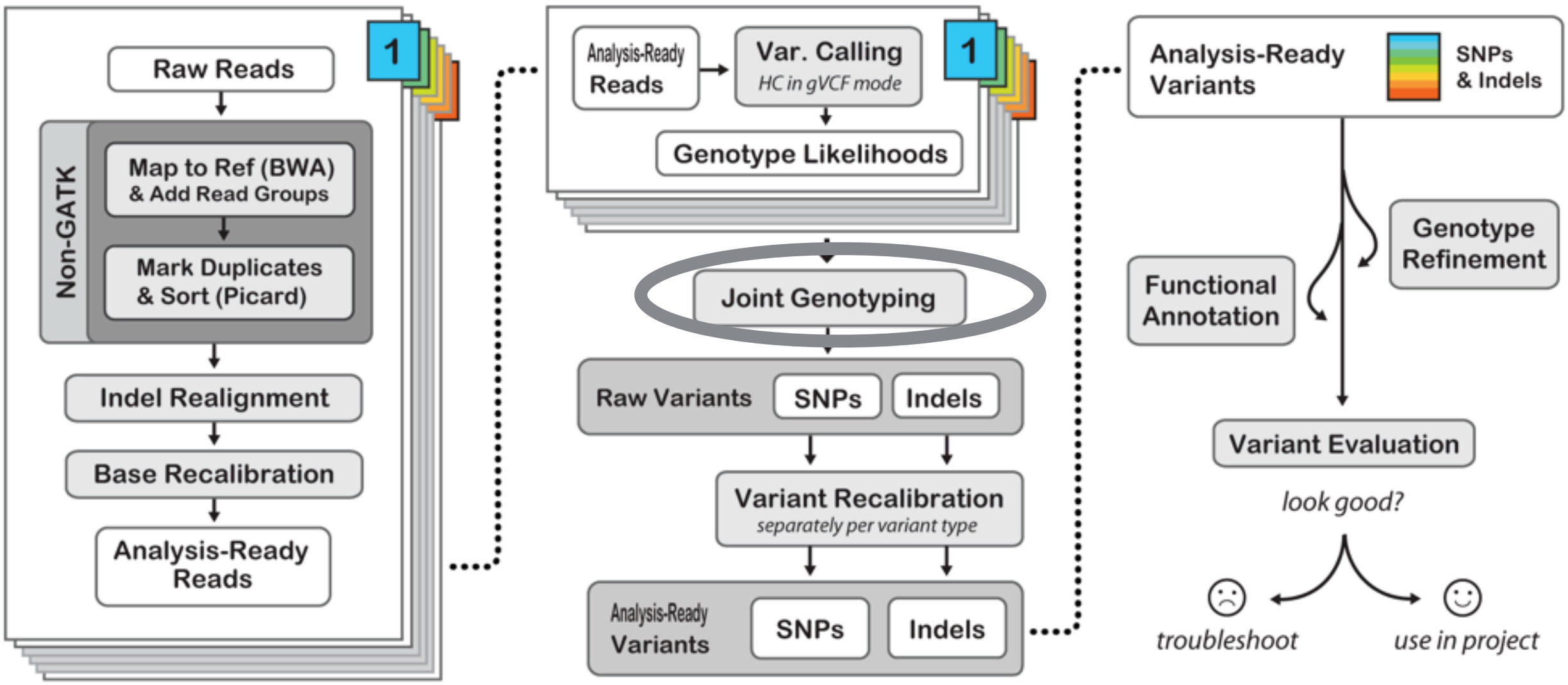
vs

Common Variant Association Study (CVAS)

vs

The motivating example

Joint genotyping is an important step in Variant Discovery



Auwera, GA et al. *Current Protocols in Bioinformatics* (2013)

The ideal database for RVAS and CVAS studies is a complete mutation matrix

All case and control samples

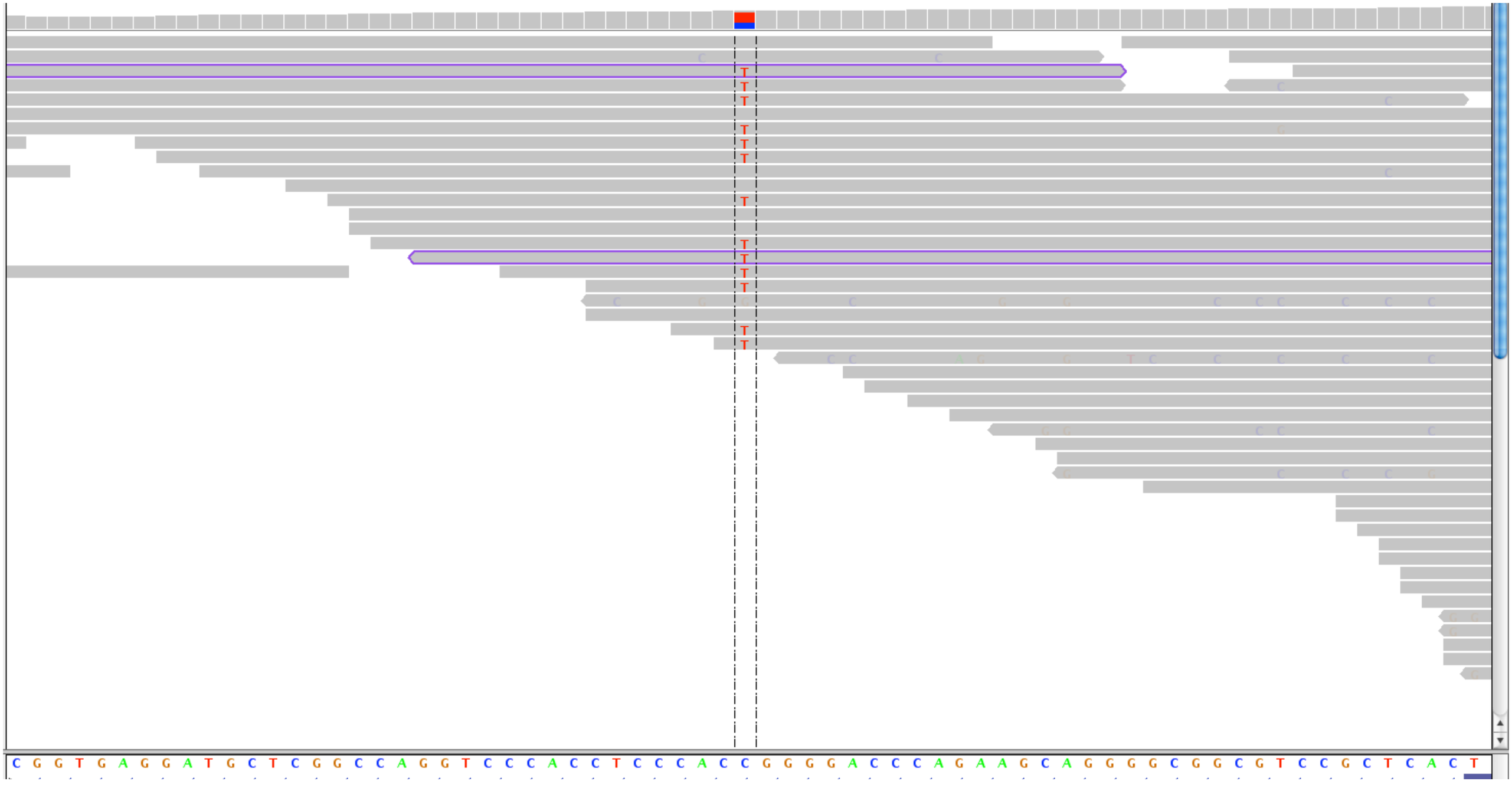
~3M variants

	Site	Variant	Sample 1	Sample 2	…	Sample N
SNP	*1:1000*	*A/C*	*0/0 0,10,100*	*0/1 20,0,200*	…	*0/0 0,100,255*
Indel	1:1050	T/TC	0/0 0,10,100	0/0 0,20,200	…	1/0 255,0,255
SNP	*1:1100*	*T/G*	*0/0 0,10,100*	*0/1 20,0,200*	…	*0/0 0,100,255*
	…	…	…	…	…	…
SNP	*X:1234*	*G/T*	*0/1 10,0,100*	*0/1 20,0,200*	…	*1/1 255,100,0*

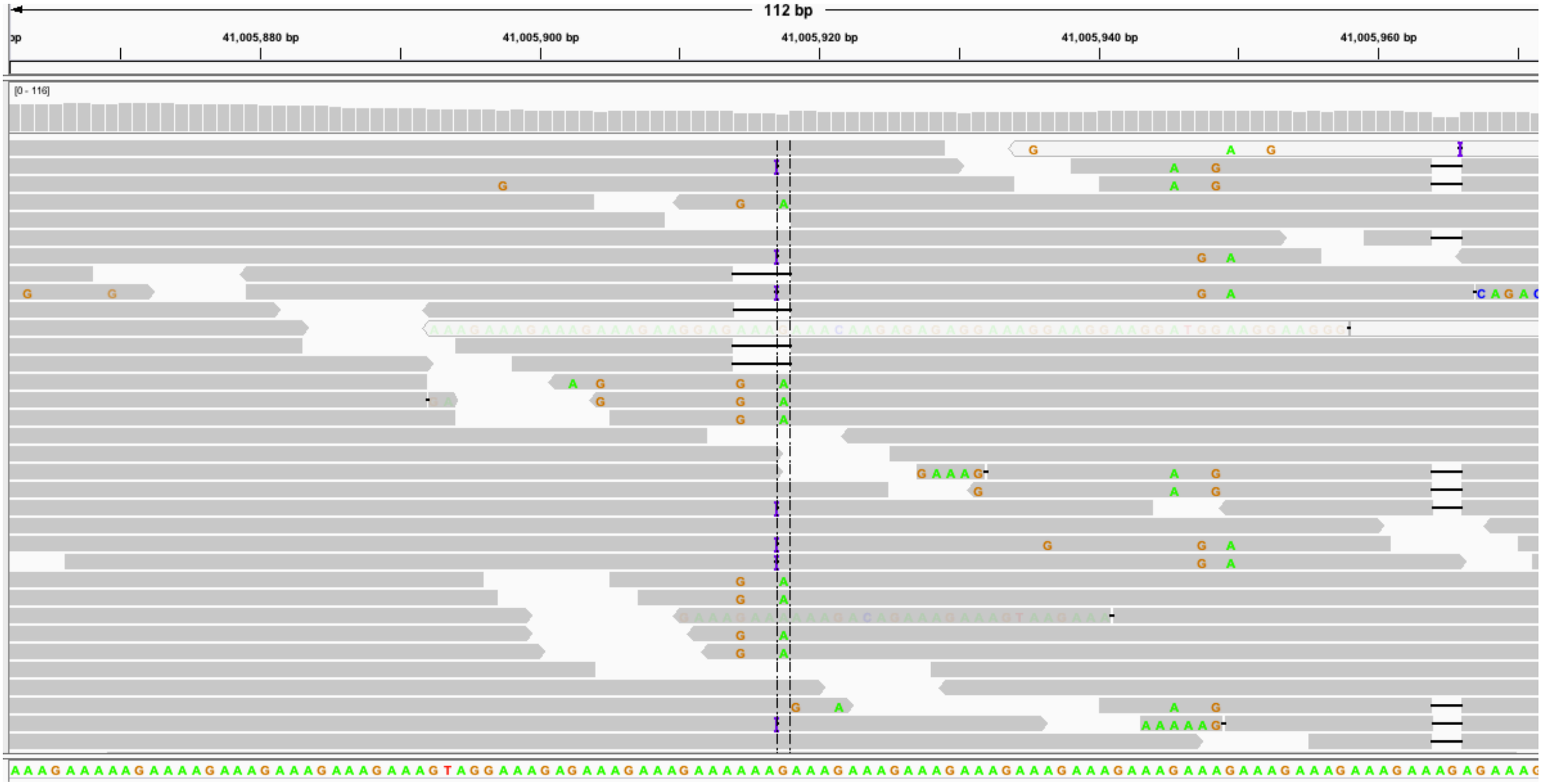
Genotypes:
0/0 ref
0/1 het
1/1 hom-alt

Likelihoods:
A/B/C phred-scaled probability of hom (A), het (B), hom-alt (C) genotypes given NGS data

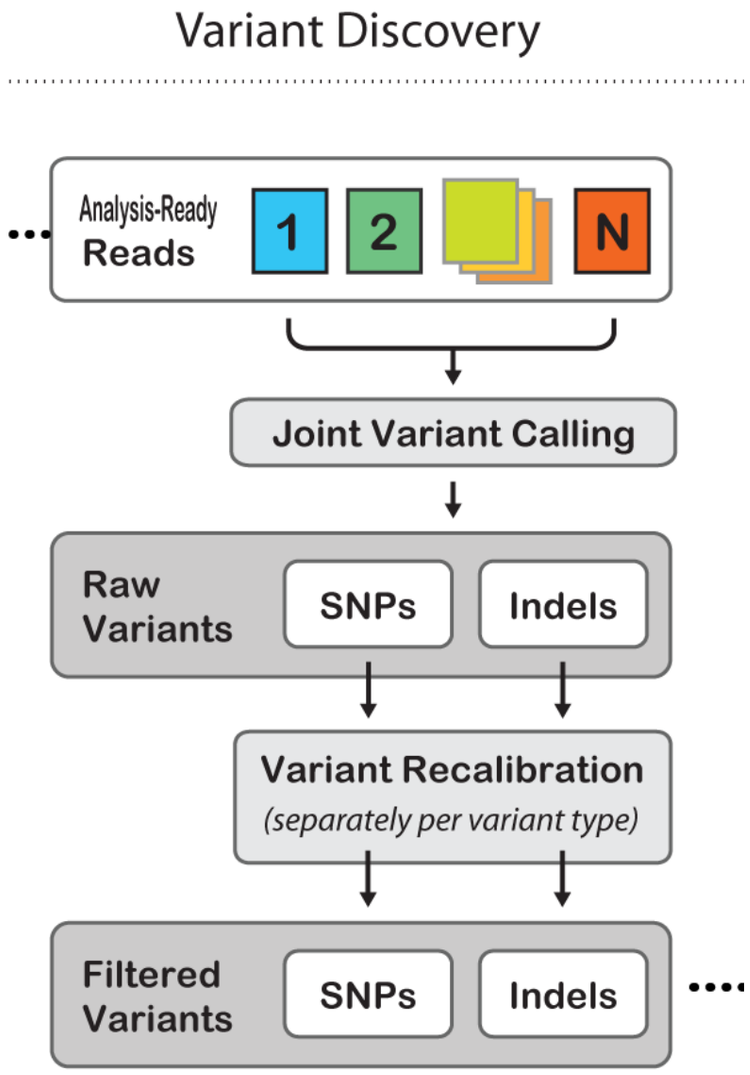
Identifying mutations in a genome is a simple “find the differences” problem



Unfortunately, real data doesn't look that simple



Variant calling is a large-scale bayesian modeling problem



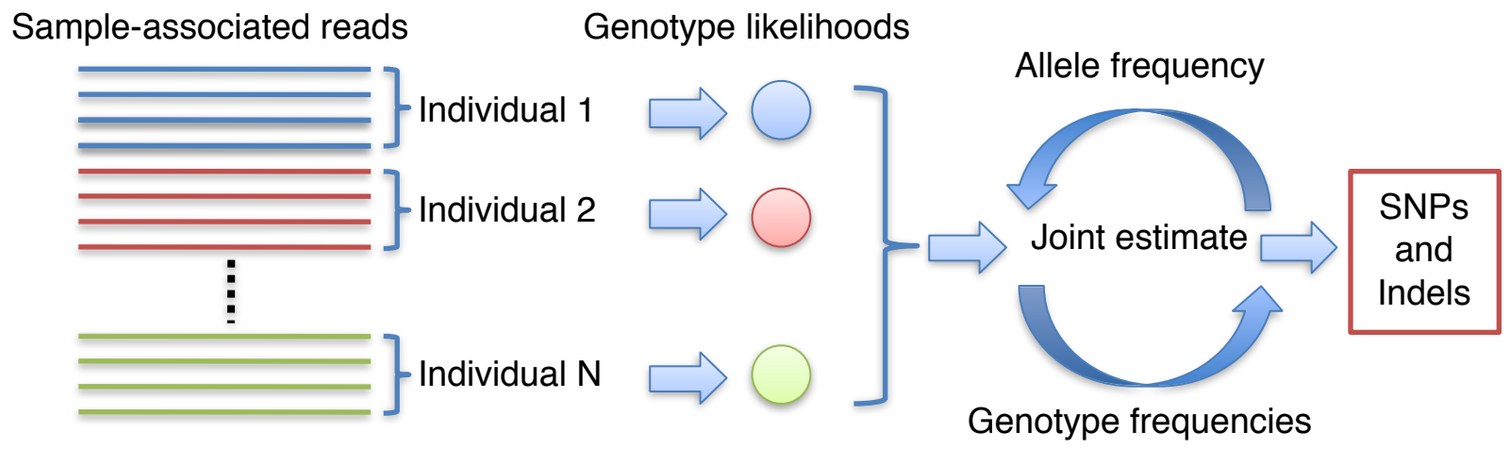
prior Likelihood

$$\Pr\{G|D\} = \frac{\Pr\{G\}\Pr\{D|G\}}{\Sigma_i \Pr\{G_i\}\Pr\{D|G_i\}}, \text{ [Bayes' rule]}$$

Diploid

$$\Pr\{D|G\} = \prod_j \left(\frac{\Pr\{D_j|H_1\}}{2} + \frac{\Pr\{D_j|H_2\}}{2}\right) \text{ where } G = H_1H_2$$

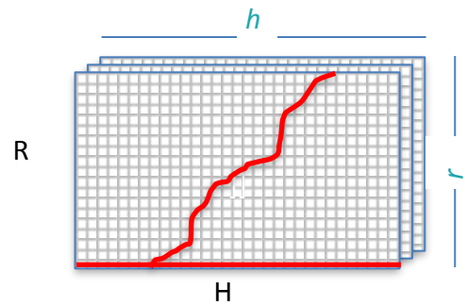
$\Pr\{D|H\}$ is the haploid likelihood function



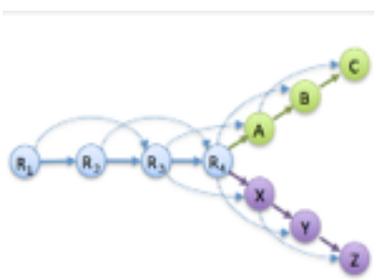
DePristo et al. *Nature Genetics* (2011)

Understanding the Haplotype Caller

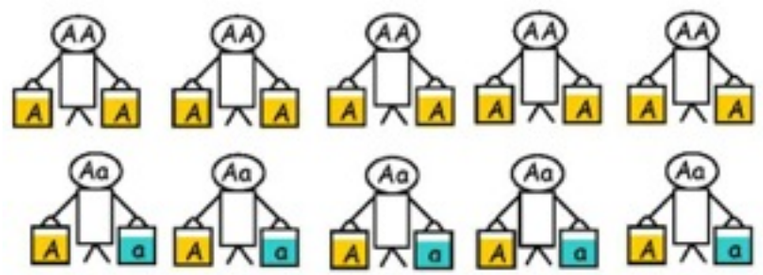
1. **Active region traversal**
identifies the regions that need to be reassembled



3. **Pair-Hmm evaluation** of all reads against all haplotypes
(scales exponentially)



2. **Local de-novo assembly**
builds the most likely haplotypes for evaluation



4. **Genotyping**
using the exact model

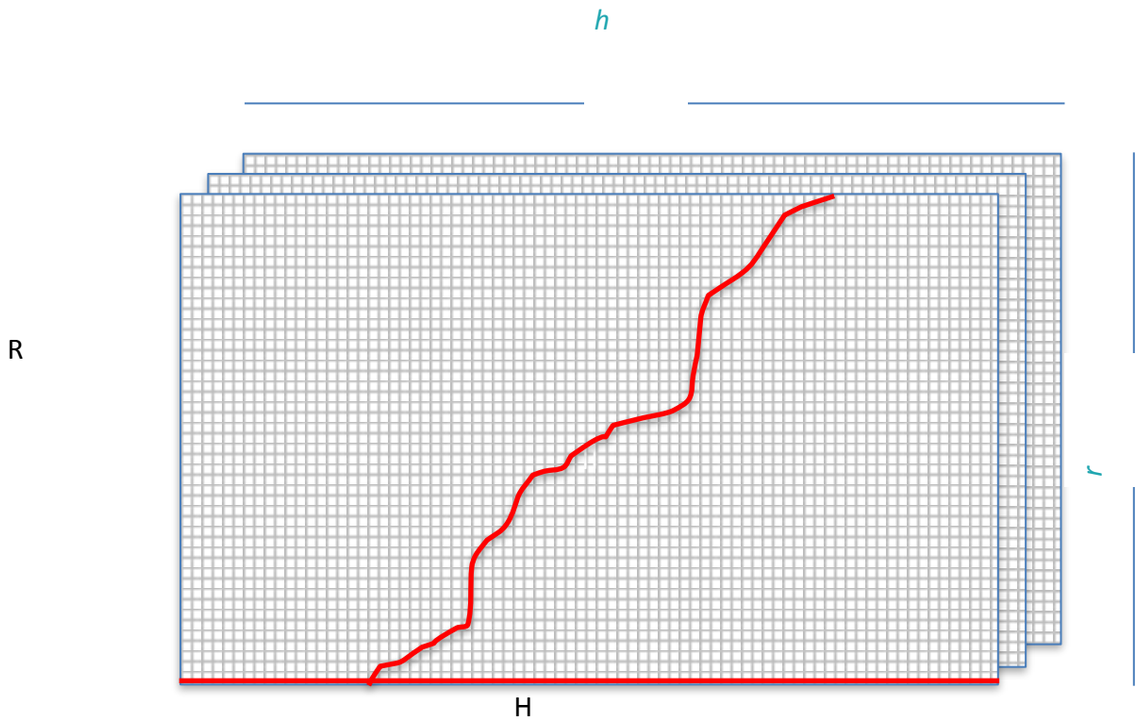
7.6 cpu/days per genome

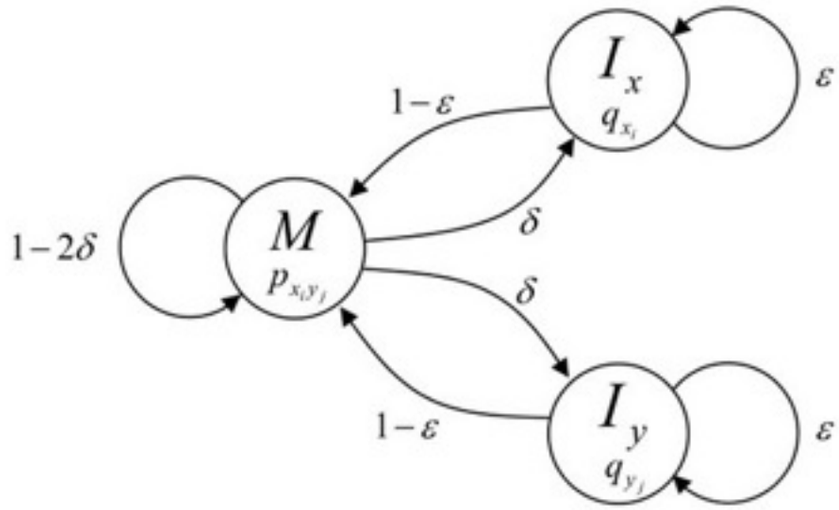
Pair-HMM is the biggest culprit for the low performance of the Haplotype Caller

Stage	Time	Runtime %
Assembly	2,598s	13%
Pair-HMM	14,225s	70%
Traversal + Genotyping	3,379s	17%

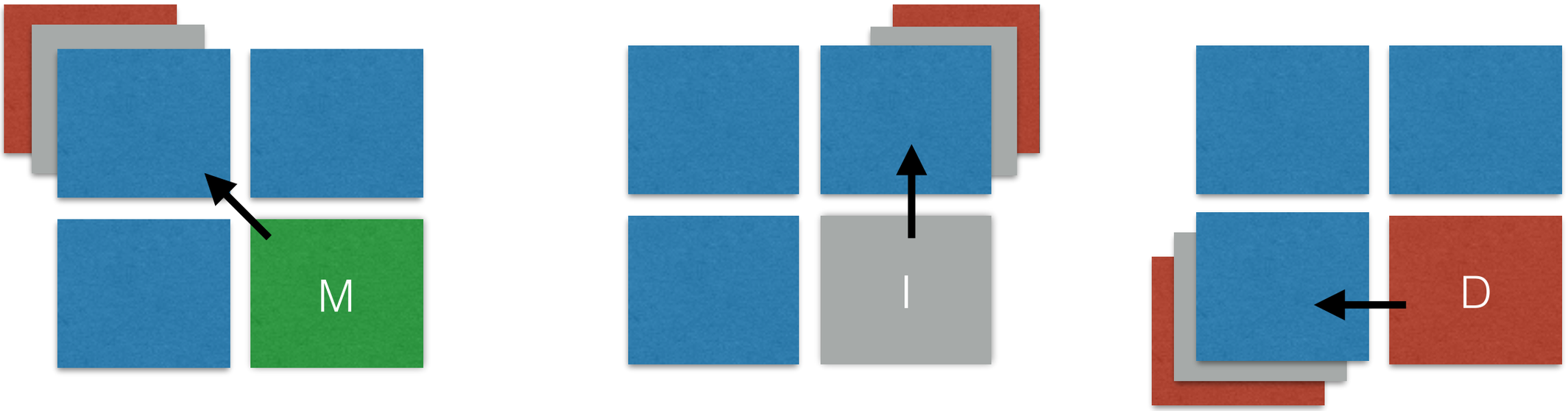
times are for chromosome 20 on a single core

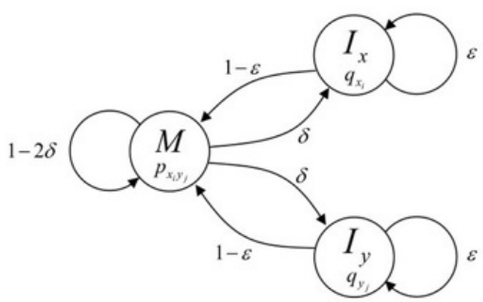
Understanding the Pair-HMM





Data dependencies of each cell in each of the three matrices (states)

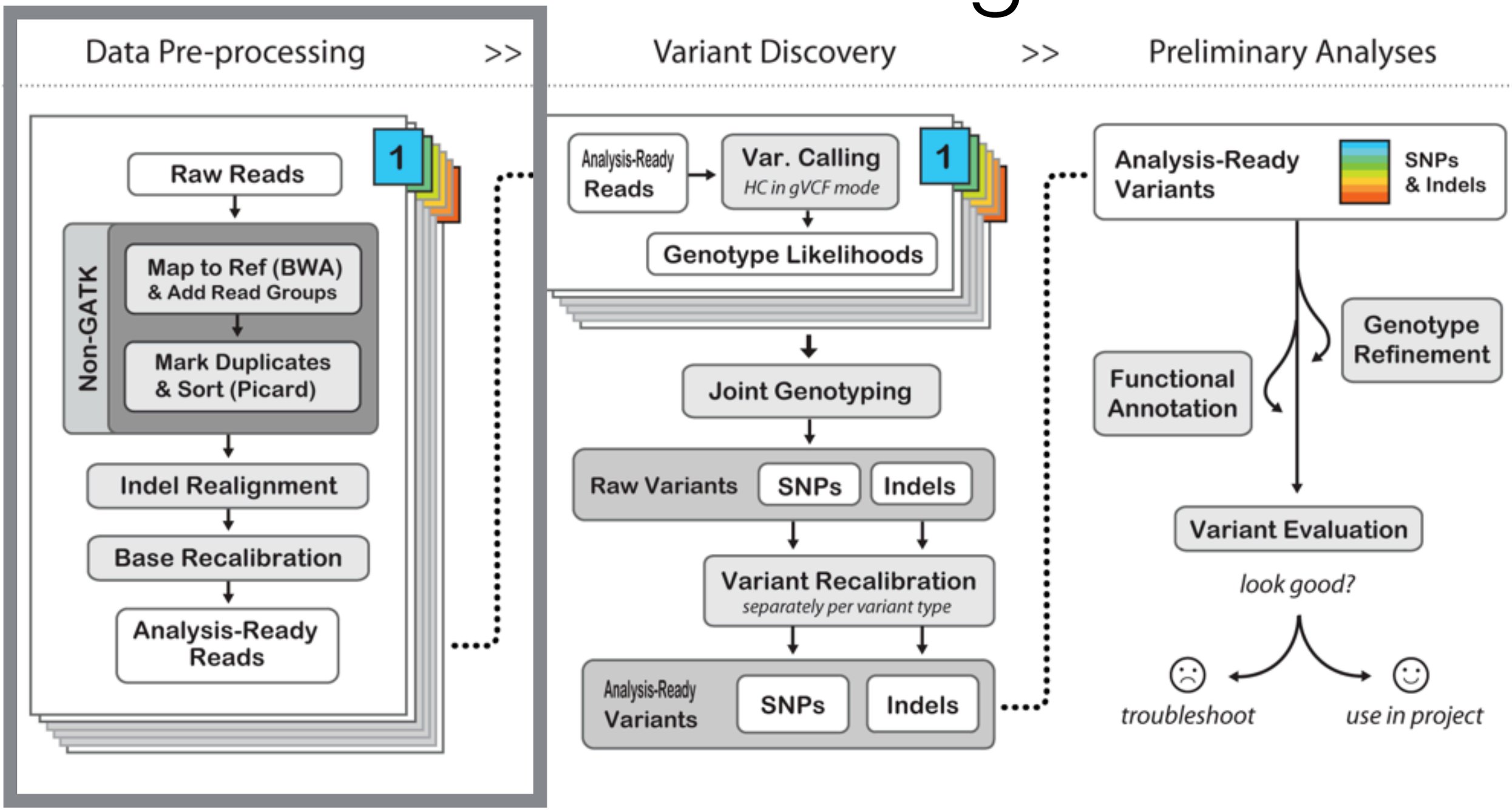




Heterogeneous compute speeds up variant calling significantly

Technology	Hardware	Runtime	Improvement
-	Java (gatk 2.8)	10,800	-
-	C++ (baseline)	1,267	9x
FPGA	Convey Computers HC2	834	13x
AVX	Intel Xeon 1-core	309	35x
GPU	NVidia GeForce GTX 670	288	38x
GPU	NVidia GeForce GTX 680	274	40x
GPU	NVidia GeForce GTX 480	190	56x
GPU	NVidia GeForce GTX Titan	80	135x
GPU	NVidia Tesla K40	70	154x

The rest of the pipeline is also not scaling well

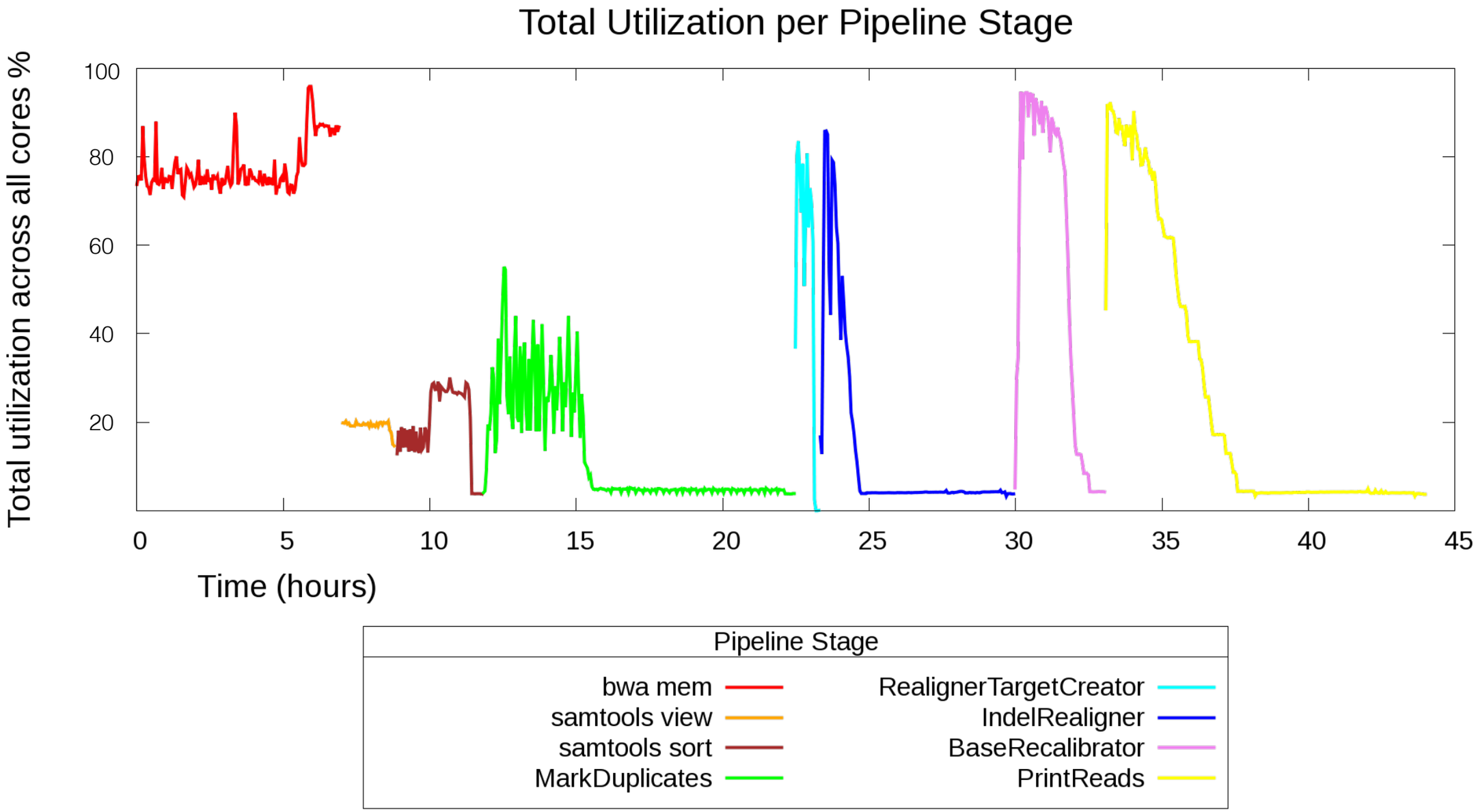


Auwera, GA et al. *Current Protocols in Bioinformatics* (2013)

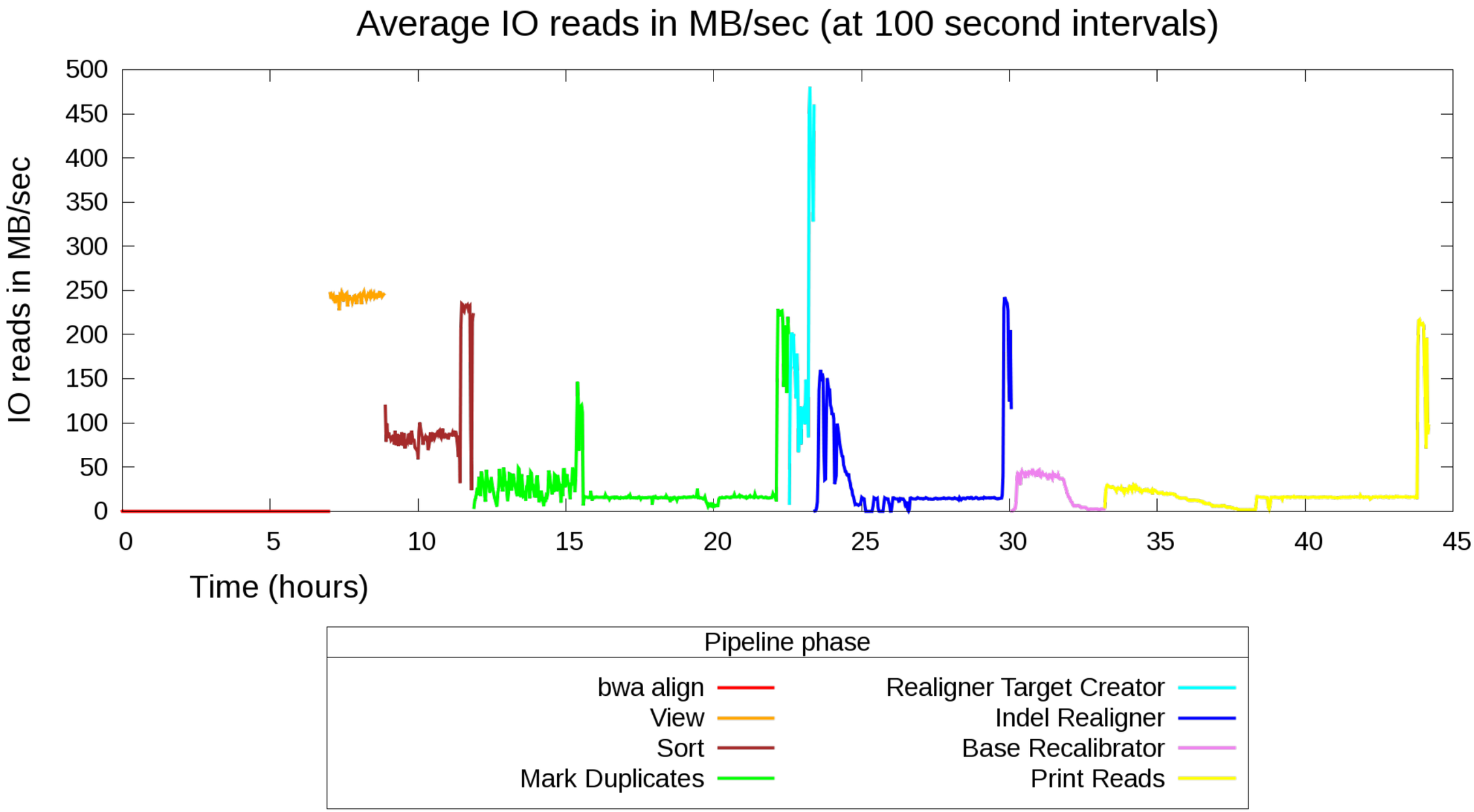
It takes 2 days to process a single genome!

step	threads	time
BWA	24	7
samtools view	1	2
sort + index	1	3
MarkDuplicates	1	11
RealignTargets	24	1
IndelRealigner	24	6.5
BaseRecalibrator	24	1.3
PrintReads + index	24	12.3
Total		**44**

Processing is a big cost on whole genome sequencing



And it is never I/O bound



The GATK java codebase has severe limitations

- More than 70% of the instructions in the current GATK pipeline are memory access — the processor is just waiting.

- Excessive use of strings, maps and sets to handle basic data structures that are frequently used in the codebase.

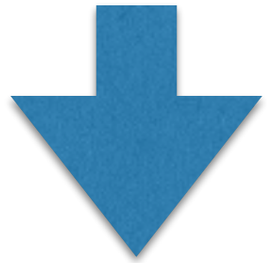
- Java makes it extremely difficult to explore memory contiguity in its data structures.

- Java floating point model is incompatible with modern x86 hardware.

- Java does not offer access to the hardware for optimizations even when desired. As a result, we are forced to underutilize modern hardware.

A typical GATK-Java Data Structure: A Map-of-Maps-of-Maps

```
Map<String,PerReadAlleleLikelihoodMap> map;
```



```
public class PerReadAlleleLikelihoodMap {
    protected Map<GATKSAMRecord,
                  Map<Allele, Double>> likelihoodReadMap
                = new LinkedHashMap<>();
...
```

No data locality – most lookups will consist of a series of cache misses

To fully understand **one** genome we need **hundreds of thousands** of genomes

Rare Variant Association Study (RVAS)

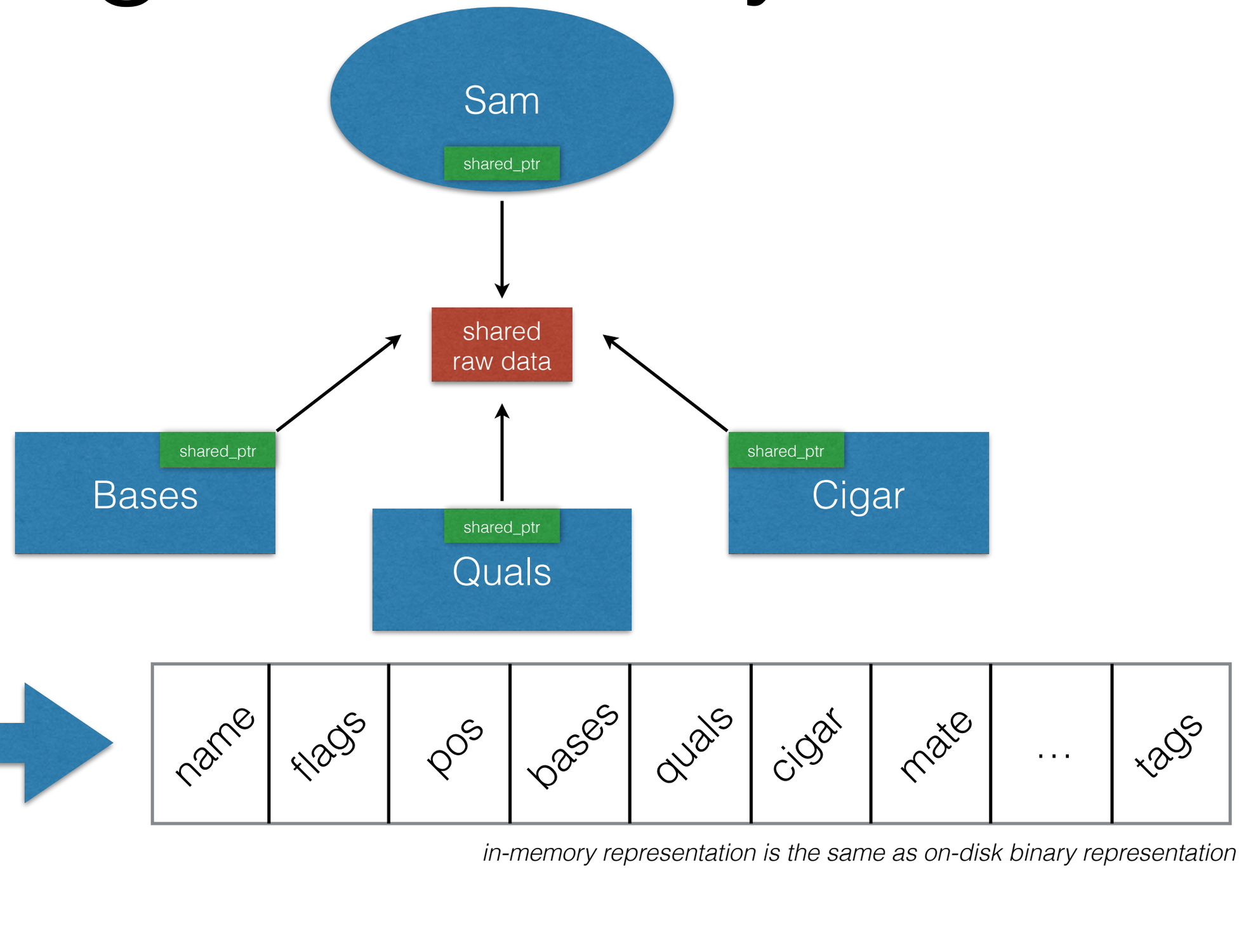
vs

Common Variant Association Study (CVAS)

vs

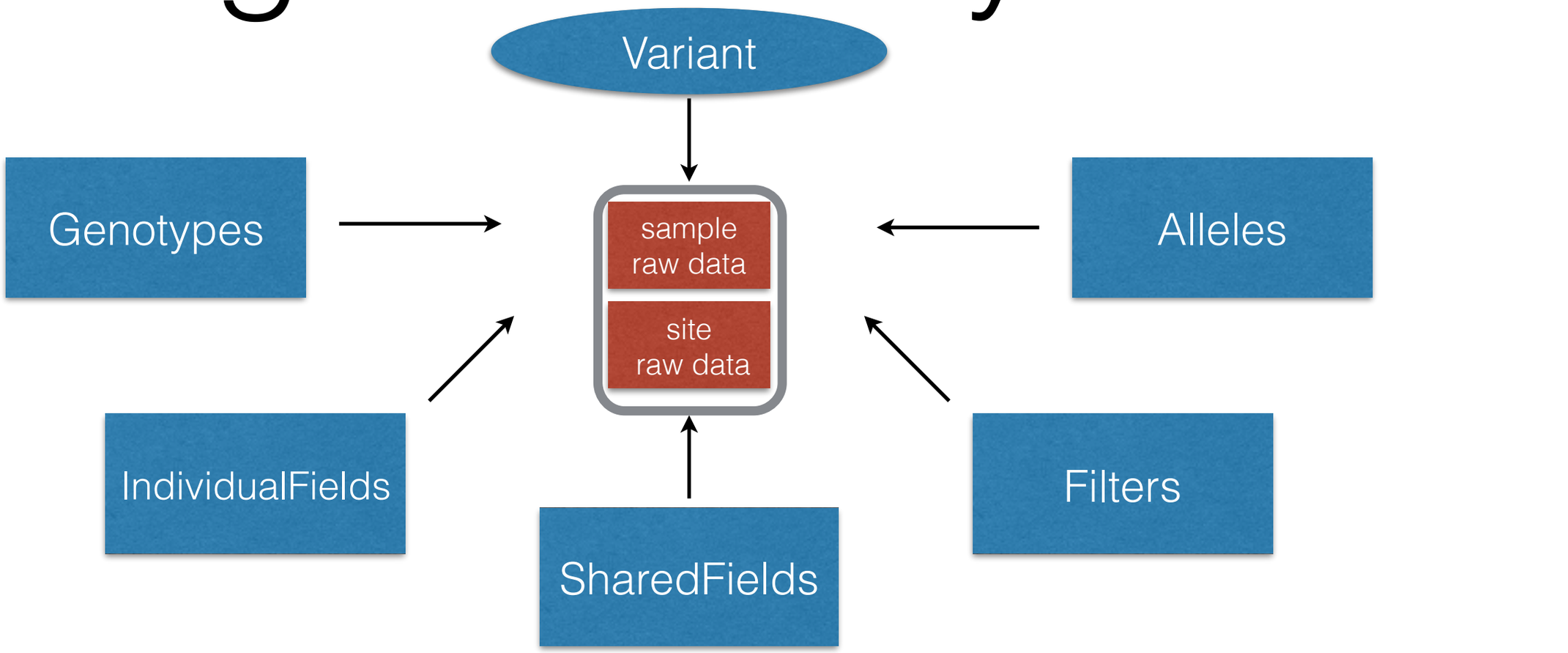
How we are using C++ to address these issues

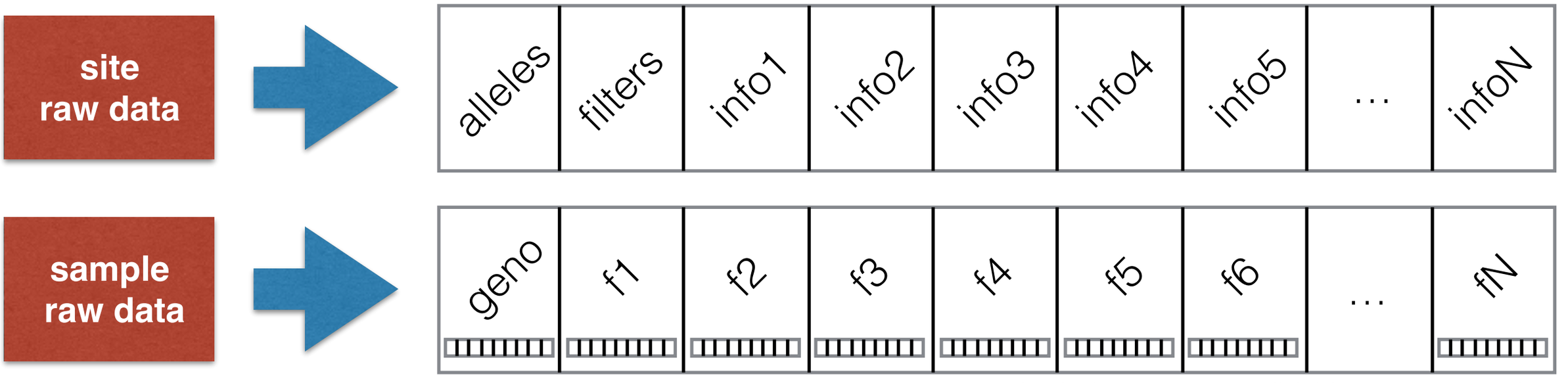
Gamgee memory model



in-memory representation is the same as on-disk binary representation

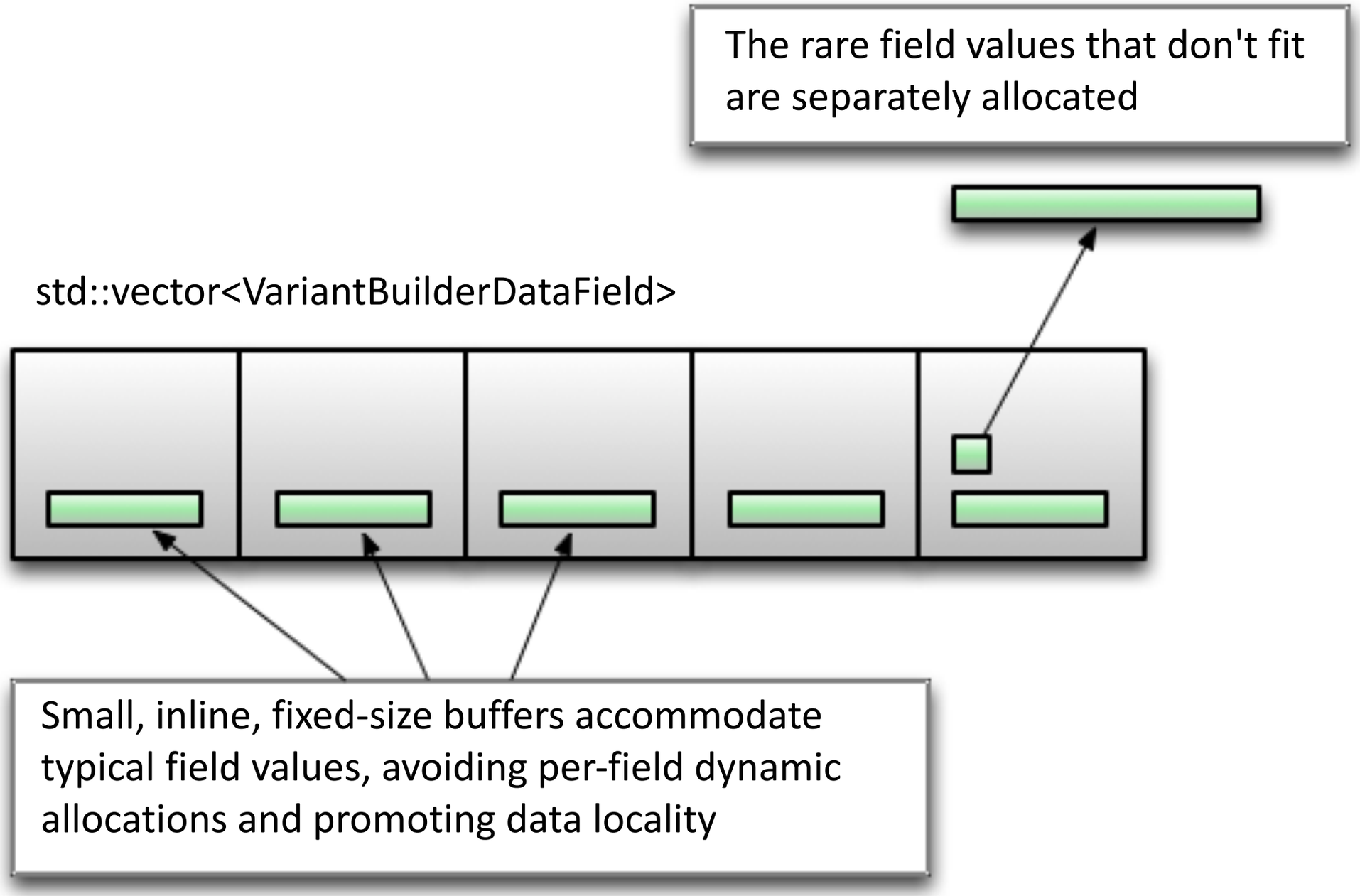
Gamgee memory model





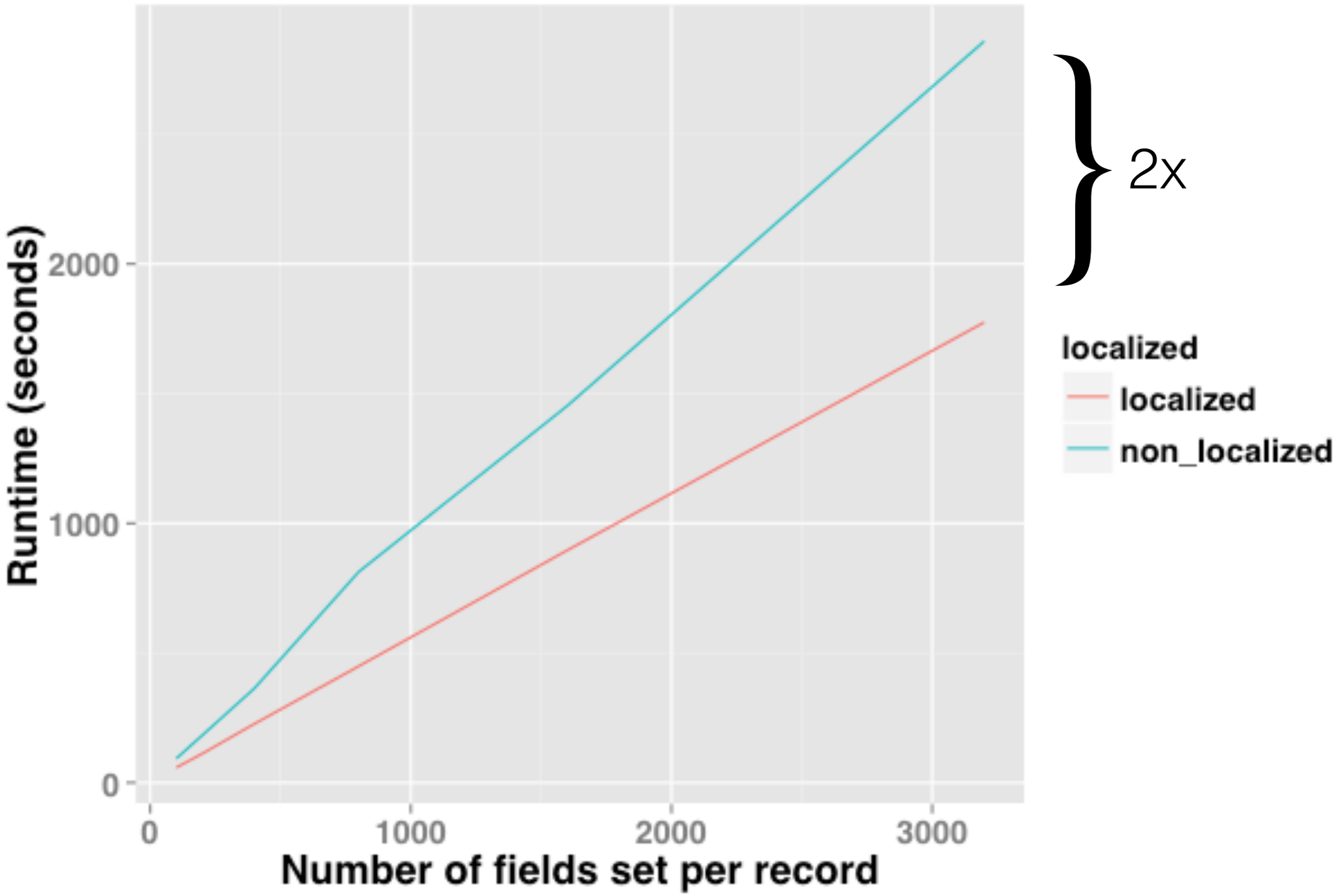
in-memory representation is the same as on-disk binary representation

VariantBuilder is optimized to preserve data locality and avoid dynamic allocation as much as possible when building records

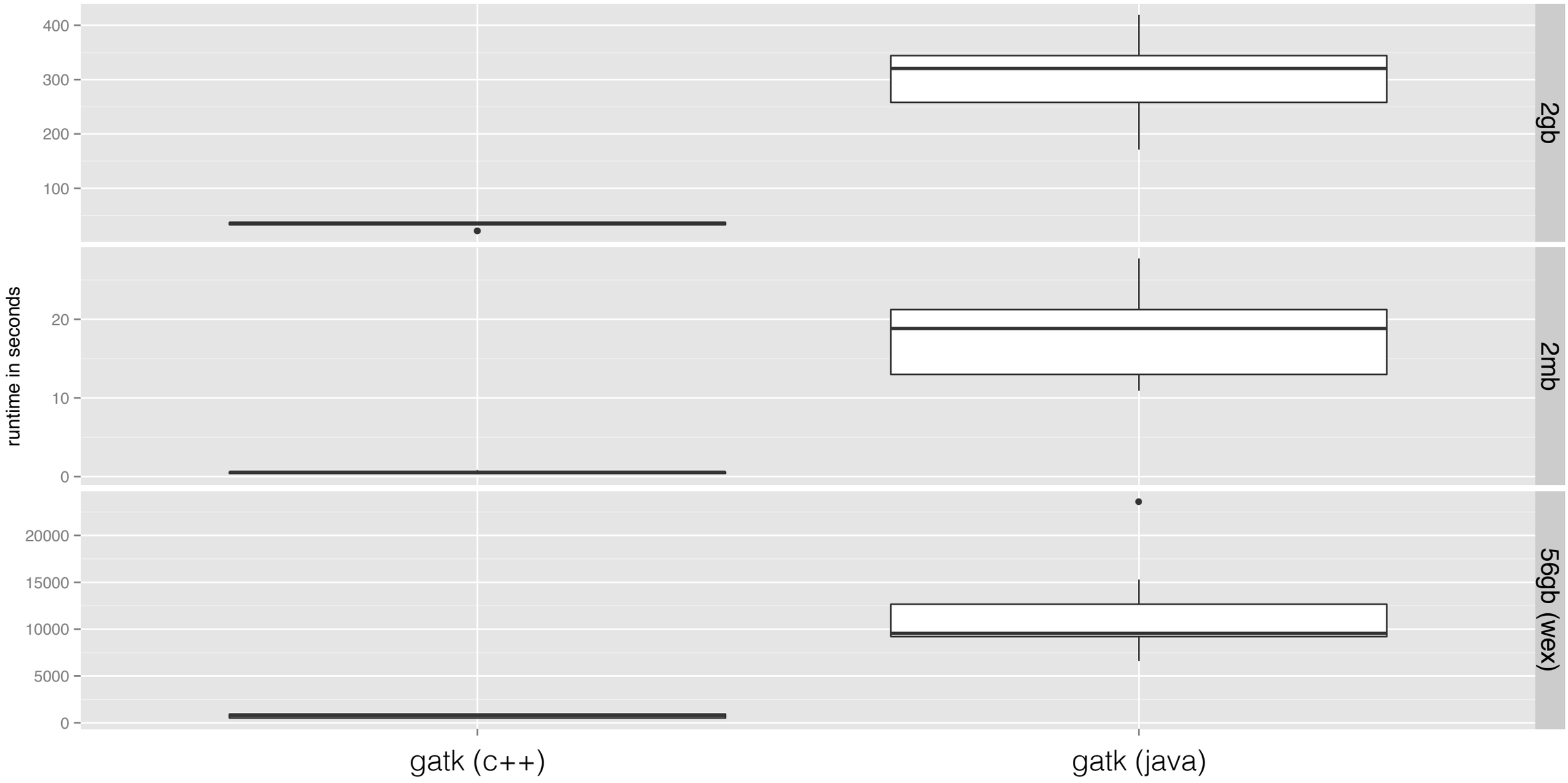


- Same idea as Short String Optimization (SSO) in std::string
- Almost impossible to achieve in Java

Time to create 3,000,000 variant records in VariantBuilder, with and without data locality optimizations



Reading BAM files is 17x faster in gamgee



Reading variant files is much faster in gamgee

2GB (1KG)	GATK C++	GATK Java
Text Variant File (VCF)	32.71s	137.57s
Binary Variant File (BCF)	4.61s	242.33s

the new memory model makes the binary version of the file extremely fast to read and write

MarkDuplicates is 5x faster

	GATK C++	new Picard (java)	old Picard (java)
Exome	4m	20m	2h23m
Genome	1h15m	4h47m	11h06m

exact same implementation in Java after our C++ version was presented

To fully understand **one** genome we need **hundreds of thousands** of genomes

Rare Variant Association Study (RVAS)

vs

Common Variant Association Study (CVAS)

vs

C++11/14

AAA makes it easy to change interfaces

Gamgee library public API code:

```
// first implementation quick and dirty
vector<vector<int32_t>> integer_individual_field(const string& tag) const;
vector<Genotype> genotypes() const;

// after refactor -- avoid unnecessary copies of shared data
IndividualField<IndividualFieldValue<int32_t>> integer_individual_field(const string& tag) const;
IndividualField<Genotype> genotypes() const;
```

Client code written before API change never had to change:

```
// count variants, skip low quality genotypes
for (const auto& record : svr) {
  const auto quals = record.integer_individual_field("GQ");
  const auto genotypes = record.genotypes();
  for (auto i = 0u; i != record.n_samples(); ++i)
    if (!missing(quals[i][0]) && quals[i][0] >= m_min_qual &&
     (genotypes[i].het() || genotypes[i].hom_var()))
    {
      nvar[i]++;
    }
}
```

Diligent use of `auto` has already saved us from modifying client code as the library changes underneath them.
— Thank's Herb!

Smart pointers make interfacing with C libraries manageable

```
class Sam {
 private:
  std::shared_ptr<bam1_t> m_body;

 public:
  Cigar cigar() const { return Cigar{m_body}; }
  ReadBases bases() const { return ReadBases{m_body}; }
  BaseQuals base_quals() const { return BaseQuals{m_body}; }
};
```

Sharing the pointers allocated in the C-library across different objects is taken care of by the `shared_ptr`

Writing tools to perform operations on variants is very simple

percent missing.cpp

```
#include "gamgee/gamgee.h"
#include <iostream>

void main() {
  for (const auto& record : SingleVariantReader{“file.bcf”}) {
    const auto g_quals = record.integer_individual_field("GQ");
    const auto n_bad_gs = count_if(g_quals.begin(), g_quals.end(),
        [&](const auto& x) { return missing(x[0]) ? true : x[0] < m_min_qual; });
    const auto percent_miss = double(n_bad_gs) / g_quals.size() * 100;
    cout << percent_miss << endl;
  }
}
```

see http://broadinstitute.github.io/gamgee/doxygen/ for the full VARIANT API

Writing tools to perform operations on read data is very simple

insert_size_distribution.cpp

```
#include "gamgee/gamgee.h"
#include <iostream>

constexpr auto EXPECTED_MAX_INSERT_SIZE = 5'000u;

void main() {
for (const auto& record : SingleSamReader{"input.bam"}) {
    auto abq = 0.0;
    const auto bqs = record.base_quals();
    accumulate(bqs.begin(), bqs.end(), [&abq](const auto q) {abq += q;}
    cout << abq / bqs.size() << endl;
  }
}
```

see http://broadinstitute.github.io/gamgee/doxygen/ for the full SAM API

`select_if` enables functional style programming across samples

variant.h

```
template <class VALUE, template<class> class ITER>
  static boost::dynamic_bitset<> select_if(
      const ITER<VALUE>& first,
      const ITER<VALUE>& last,
      const std::function<bool (const decltype(*first)& value)> pred)
{
  const auto n_samples = last - first;
  auto selected_samples = boost::dynamic_bitset<>(n_samples);
  auto it = first;
  for (auto i = 0; i != n_samples; ++i)
    selected_samples[i] = pred(*it++);
 return selected_samples;
}
```

applies a predicate over a *Container* and selects those that *pass* in a dynamic bitset

`select_if` statements make it trivial to parallelize batch operations over samples

indel_length.cpp

```
auto select_high_quality_variants(const Variant& var, const int32_t q) {
    const auto quals = var.integer_individual_field("GQ");
    const auto genotypes = var.genotypes();

    const auto pass_qual = select_if(quals.begin(), quals.end(),
        [&q](const auto& gq) { return gq[0] > q; });

    const auto is_var = select_if(genotypes.begin(), genotypes.end(),
        [](const auto& g) { return !g.missing() && !g.hom_ref(); });

    return pass_qual & is_var;
}
```

multiple `select_if` operations can be easily parallelized with `std::async`

A lambda configurable class for locus level operations

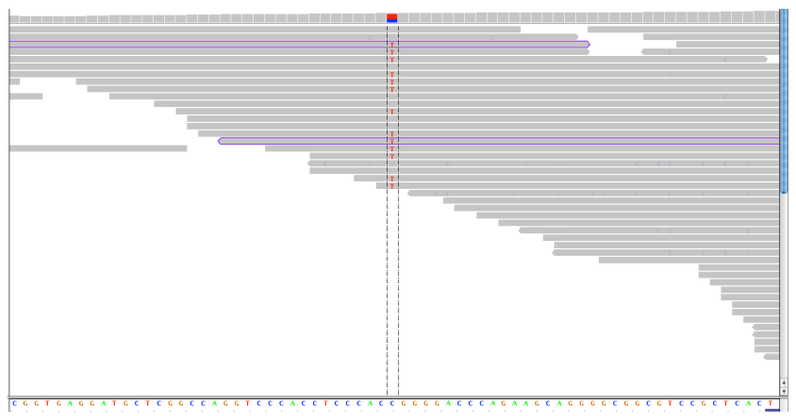
locus_coverage.h

(1)
(2)

```
class LocusCoverage {
 public:
  LocusCoverage(
    const std::function<uint32_t (
          const std::vector<uint32_t>& locus_coverage,
          const uint32_t chr,
          const uint32_t start,
          const uint32_t stop ) >& window_op,

    const std::function<uint32_t (const uint32_t)>& locus_op =
          [](const auto){return 1;}
  );

  void add_read(const Sam& read);
  void flush() const;
  ...
};
```



Coverage distribution tool: functional style

coverage_distribution.cpp

```
using Histogram = std::vector<uint32_t>;
constexpr auto MAX_COV = 50’000u;

void main() {
  auto hist = Histogram(MAX_COV,0u);

  auto window_op = [&hist](const auto& lcov, const auto,
                           const auto start, const auto stop)
  {
    std::for_each(lcov.begin() + start,
                  lcov.begin() + stop + 1,
                  [&hist](const auto& coverage)
                  {
                     ++hist[min(coverage,MAX_COV-1)];
                  }
    );
    return stop;
  };

  auto reader = SingleSamReader{“file.bam”};
  auto state = LocusCoverage{window_op};

  for_each(reader.begin(), reader.end(),
    [&state](const auto& read) { if (!read.unmapped()) state.add_read(read); });

  output_coverage_histogram(hist);
}
```

The future of the GATK

Libraries
frameworks

MIT License

gamgee

+

GATK tool developer framework
c++

GATK tool developer framework
java

Toolkits

GATK License

c++

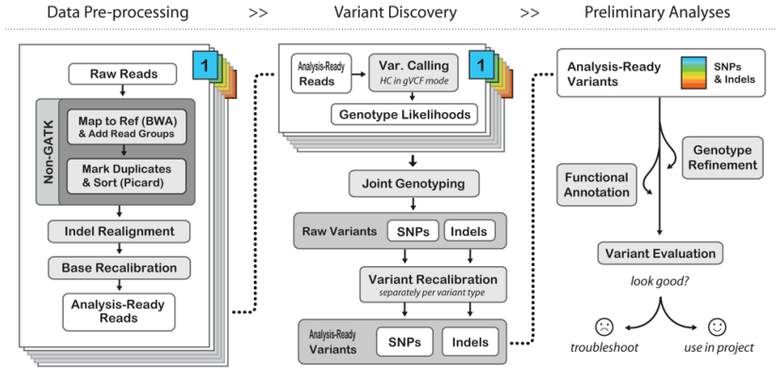
gatk

+

java

gatk

Research tools need this scalability for the next wave of scientific advances



Data Processing from DNA to Variants

ready for ~1 million genomes

(will need more work to reach tens-hundreds of millions)

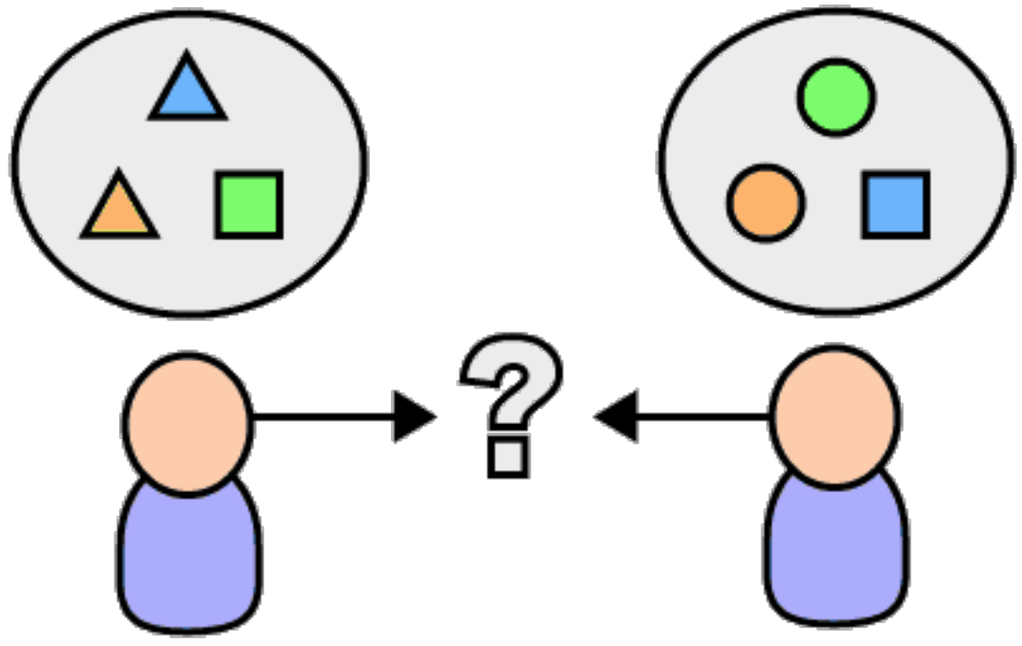
Variant analysis and association studies

fails today at just a few thousand genomes

Post-calling pipeline standardization and scaling is the next big challenge

- Tools are not generalized and performance does not scale. (typically written in matlab, R, PERL and Python…)

- Most code is written by one grad student/postdoc and is no longer maintained.
- Not standardized.
- Analyses are very often unrepeatable.
- Complementary data types are not standardized (e.g. phenotypic data).



This is the work of many…

the team

Eric Banks
Ryan Poplin
Khalid Shakir
David Roazen
Joel Thibault
Geraldine VanDerAuwera
Ami Levy-Moonshine
Valentin Rubio
Bertrand Haas
Laura Gauthier
Christopher Wheelan
Sheila Chandran

Broad colleagues

Heng Li
Daniel MacArthur
Timothy Fennel
Steven McCarrol
Mark Daly
Sheila Fisher
Stacey Gabriel
David Altshuler

collaborators

Menachem Fromer
Paolo Narvaez
Diego Nehab