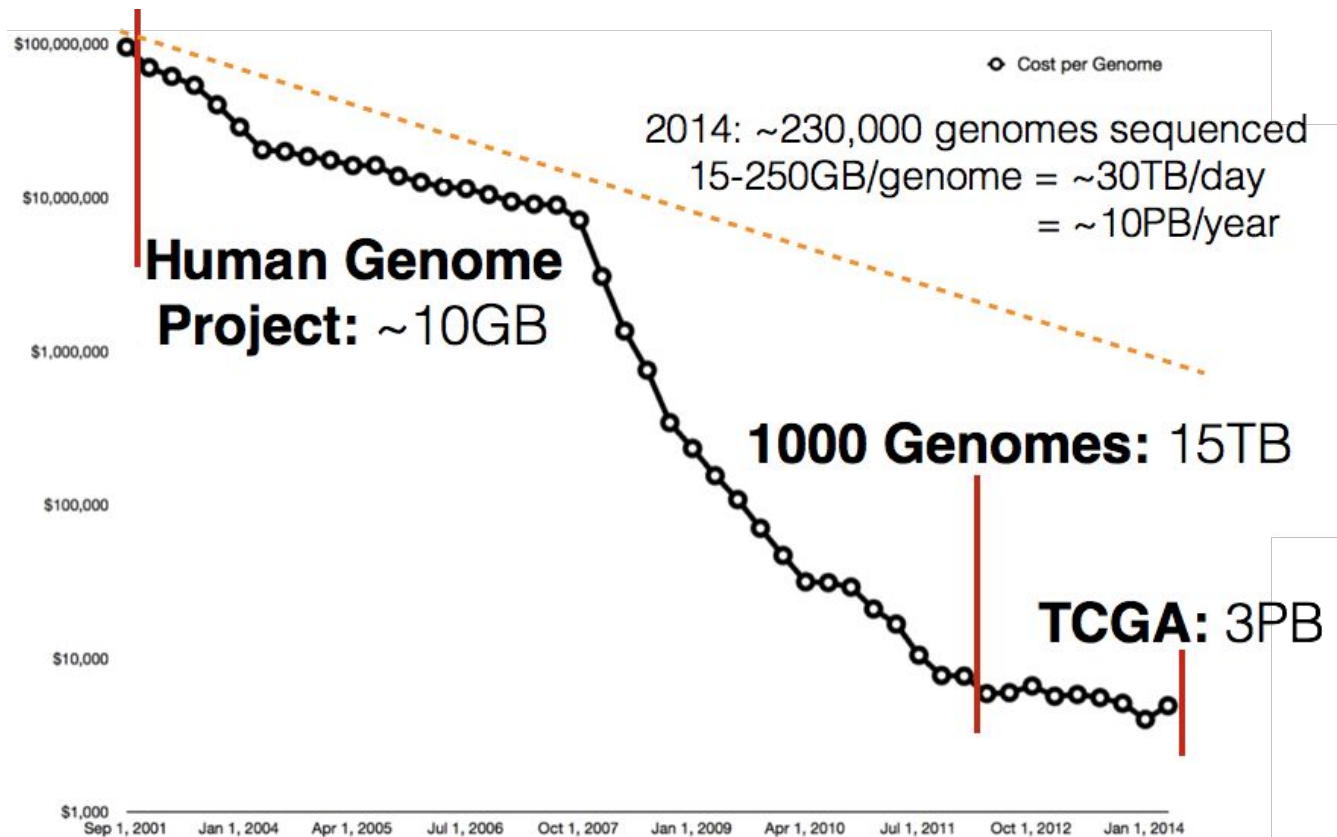# Analyzing 100's of TB of Genomic Data with ADAM and Toil

Frank Austin Nothaft
@fnothaft, fnothaft@berkeley.edu
6/8/2016

Compulsory Moore's Law Slide

The Sequencing Abstraction
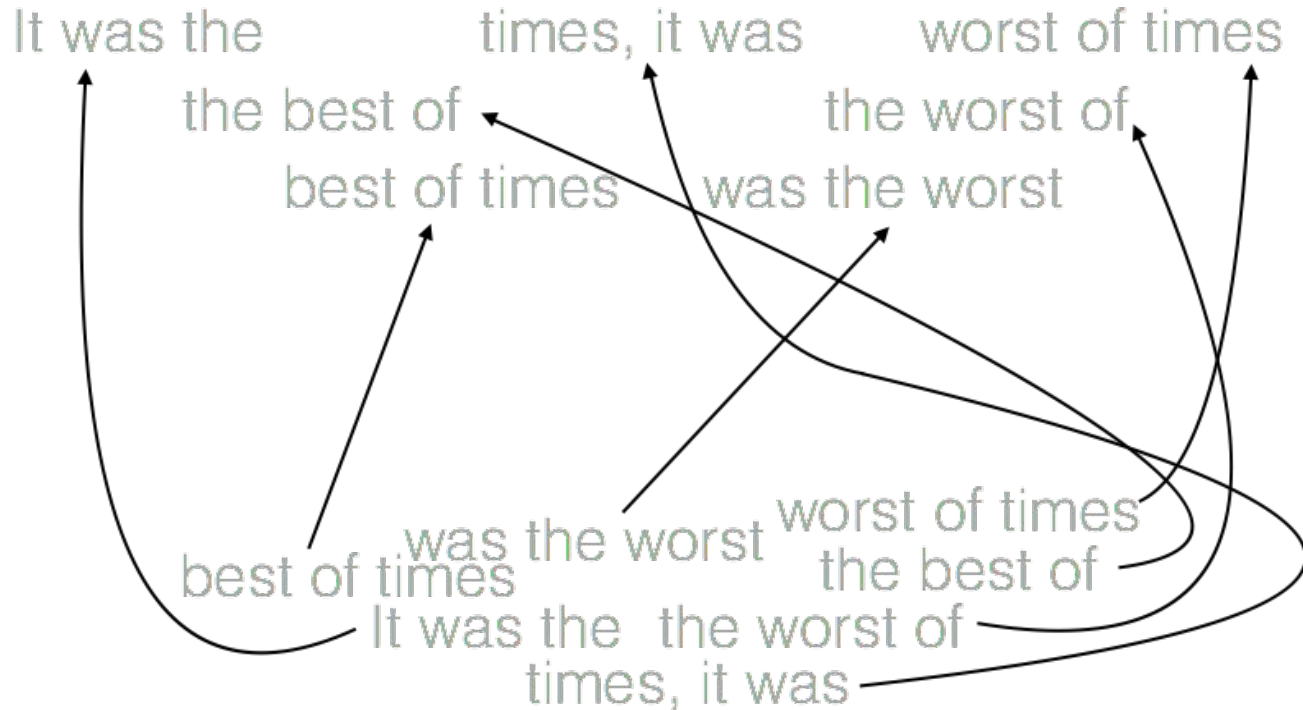
It was the best of times, it was the worst of times…

was the worst worst of times
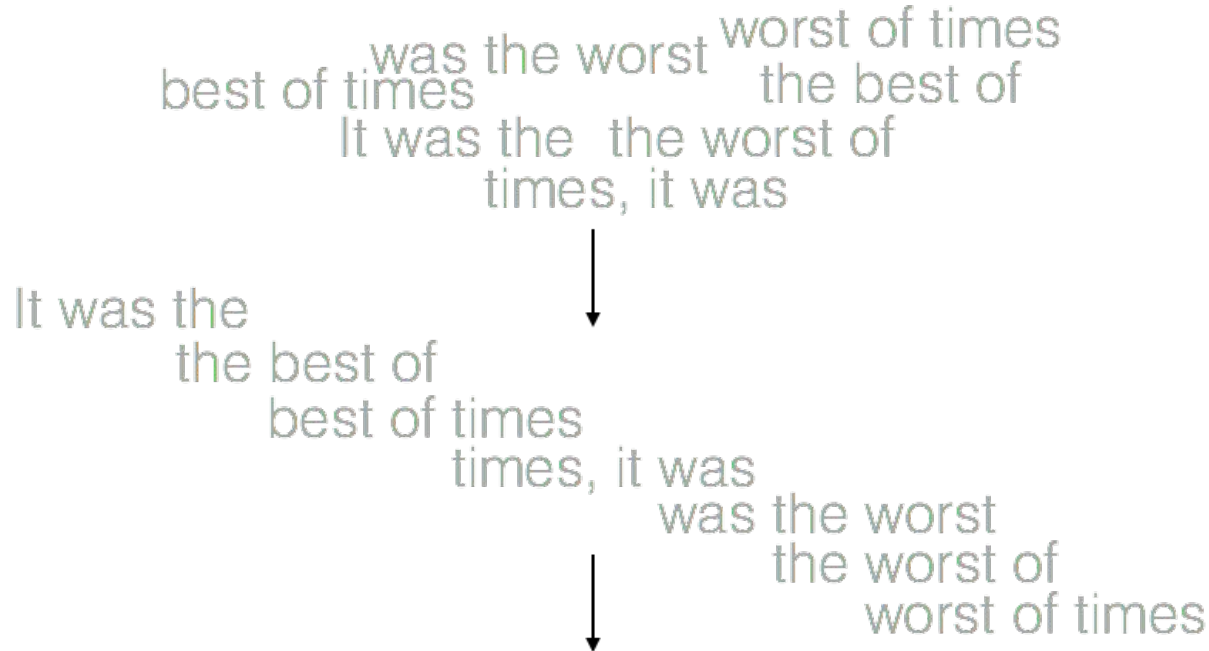best of times the best of
It was the the worst of
times, it was

- Sequencing is a Poisson substring sampling process
- For $1,000, we can sequence a 30x copy of your genome, but what is the analysis cost?
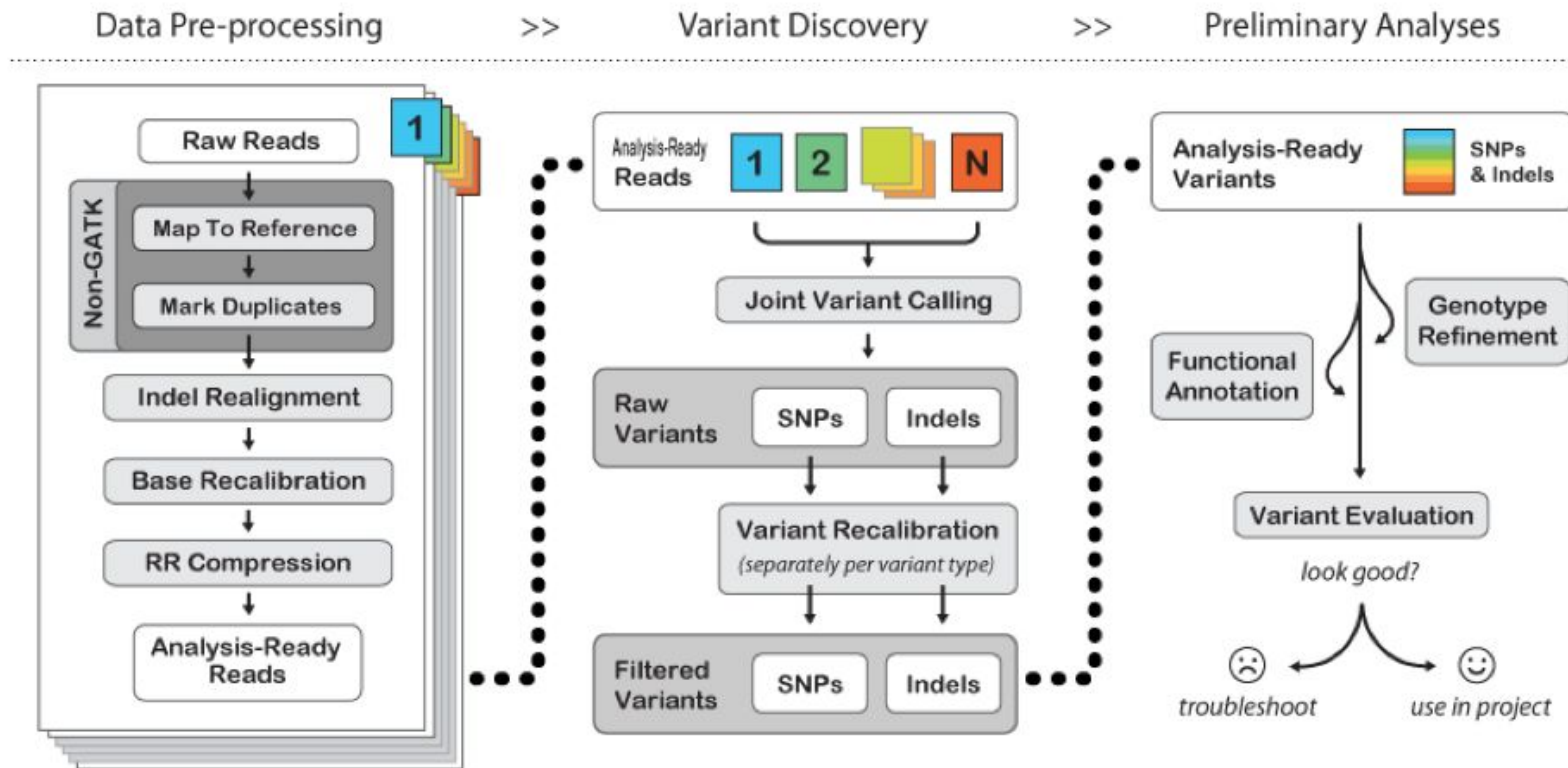
The Alignment Process

It was the best of times, it was the worst of times…

The Sequence Re-assembly Process

It was the best of times, it was the worst of times…

End-to-end variant analysis

Broad Institute of MIT/Harvard

End-to-end variant analysis

Align and clean up
errors in raw reads
150 hrs/sample

End-to-end variant analysis

Generate diffs between samples
1 week/100 samples
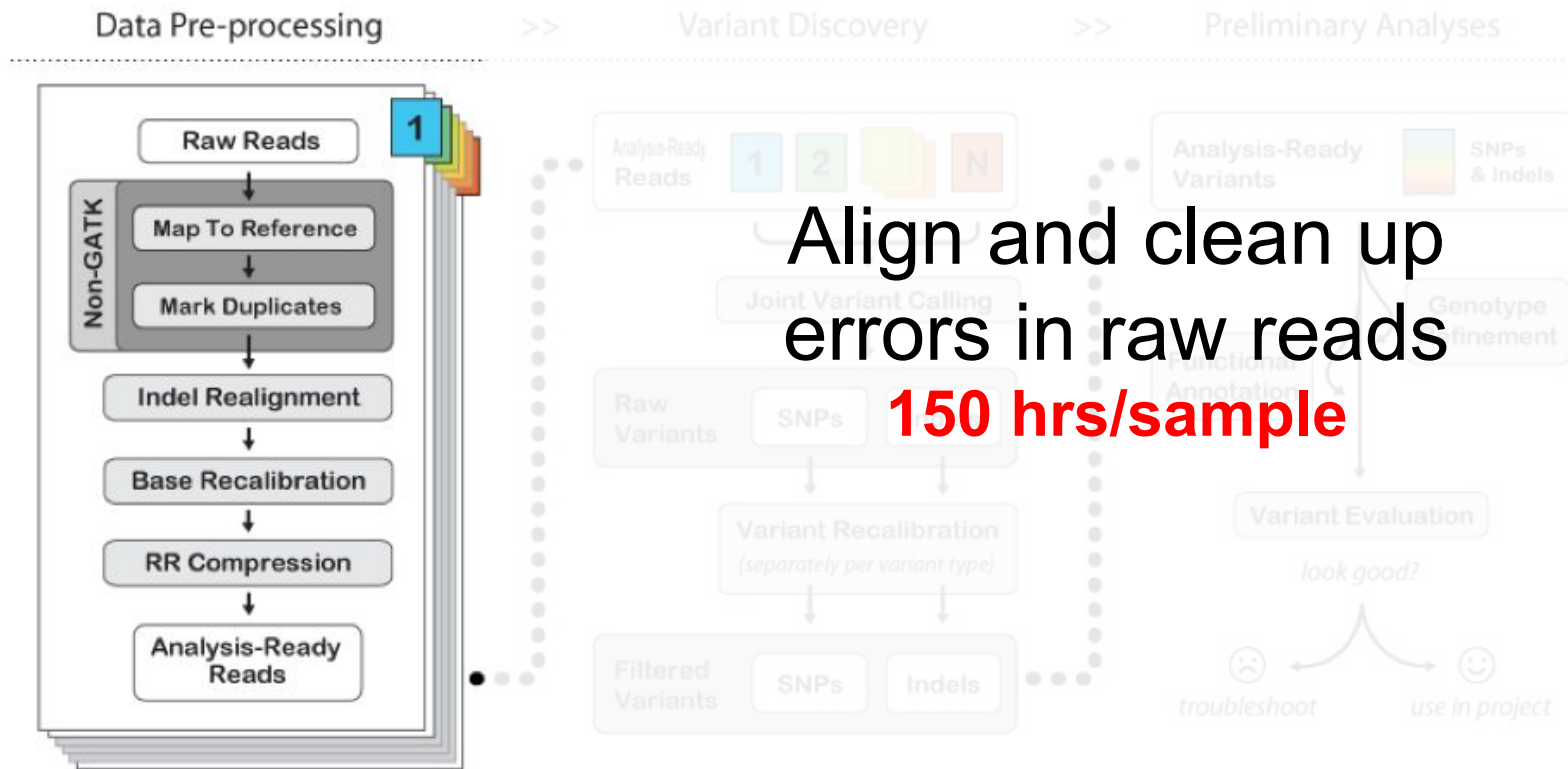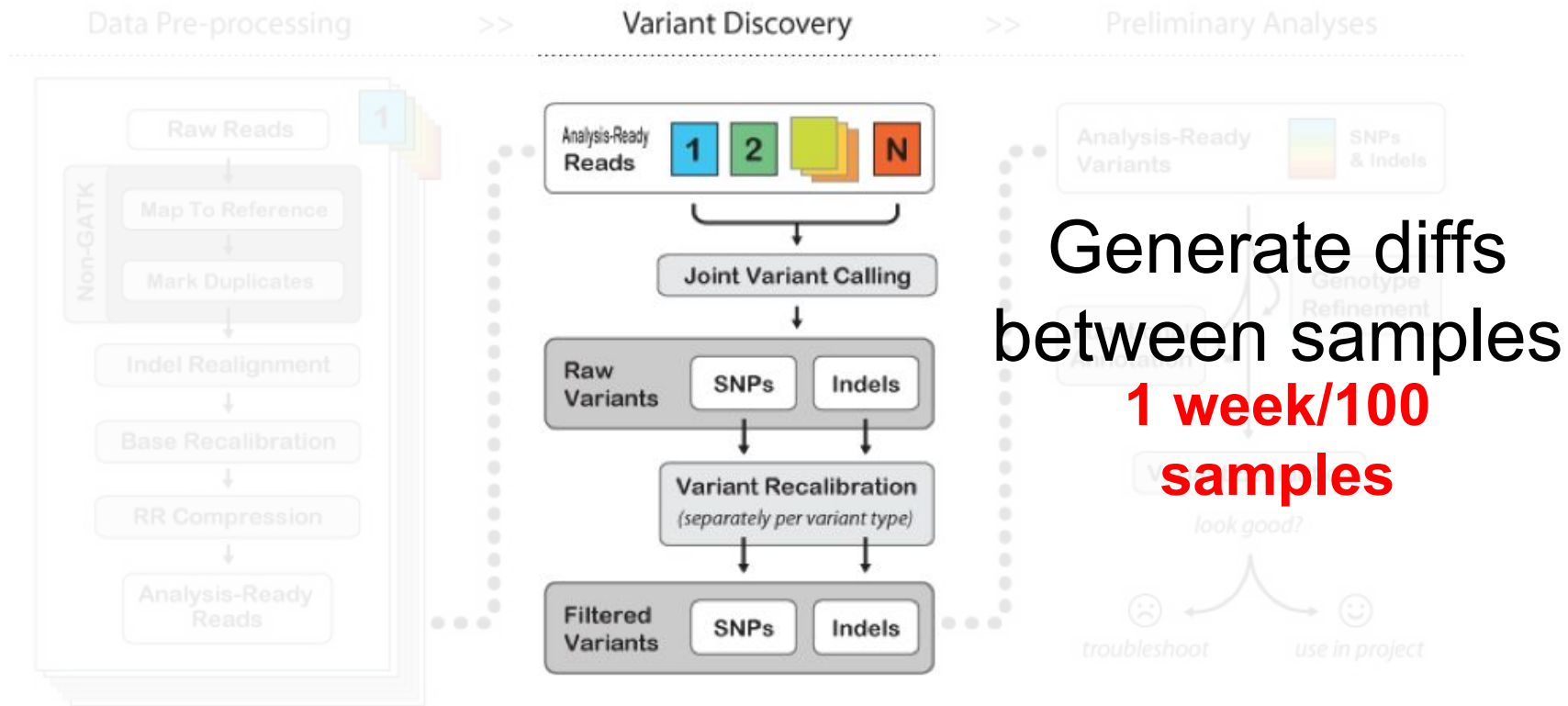
Genomics is built around flattened tools

Genomics is built around legacy file formats:

- E.g., SAM/BAM → alignment, VCF → variants, BED/GTF/etc → features
- Manually curated text/binary flat files

These formats dictate:

- What accesses can be optimized (read a full row)
- What predicates can be evaluated (small number of genomic loci)
- How we write genomic algorithms (sorted iterator over genome)

BAM File Format

```
Header: n = 500
Reference 1
Sample 1
1: c20, TCGA, 4M; 2: c20,
GAAT, 4M1D; 3: c20, CCGAT,
5M; 4: c20, TTGCAC, 6M; 5:
c20, CCGT, 3M1D1M;  ...
```

Why avoid flat architectures?

Flat architectures tend to expose bad programming interfaces:

- What access patterns do our flat files lock us into?
- GATK: *sorted iterator over the genome*

What do flat architectures break?

1. Trivial: low level abstractions are not productive
2. Trivial: flat architectures create technical lock-in
3. Subtle: low level abstractions can introduce bugs

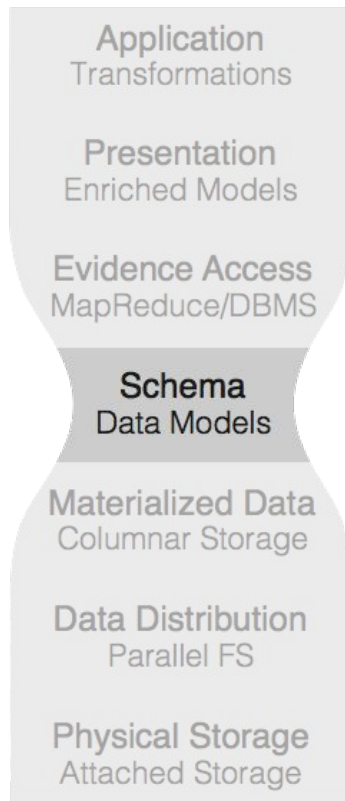# Green field genomics: start with a schema!

```
record AlignmentRecord {
    union { null, Contig } contig = null;
    union { null, long } start = null;
    union { null, long } end = null;
    union { null, int } mapq = null;
    union { null, string } readName = null;
    union { null, string } sequence = null;
    union { null, string } mateReference = null;
    union { null, long } mateAlignmentStart = null;
    union { null, string } cigar = null;
    union { null, string } qual = null;
    union { null, string } recordGroupName = null;
    union { int, null } basesTrimmedFromStart = 0;
    union { int, null } basesTrimmedFromEnd = 0;
    union { boolean, null } readPaired = false;
    union { boolean, null } properPair = false;
    union { boolean, null } readMapped = false;
    union { boolean, null } mateMapped = false;
    union { boolean, null } firstOfPair = false;
    union { boolean, null } secondOfPair = false;
    union { boolean, null } failedVendorQualityChecks = false;
    union { boolean, null } duplicateRead = false;
    union { boolean, null } readNegativeStrand = false;
    union { boolean, null } mateNegativeStrand = false;
    union { boolean, null } primaryAlignment = false;
    union { boolean, null } secondaryAlignment = false;
    union { boolean, null } supplementaryAlignment = false;
    union { null, string } mismatchingPositions = null;
    union { null, string } origQual = null;
    union { null, string } attributes = null;
    union { null, string } recordGroupSequencingCenter = null;
    union { null, string } recordGroupDescription = null;
    union { null, long } recordGroupRunDateEpoch = null;
    union { null, string } recordGroupFlowOrder = null;
    union { null, string } recordGroupKeySequence = null;
    union { null, string } recordGroupLibrary = null;
    union { null, int } recordGroupPredictedMedianInsertSize = null;
    union { null, string } recordGroupPlatform = null;
    union { null, string } recordGroupPlatformUnit = null;
    union { null, string } recordGroupSample = null;
    union { null, Contig } mateContig = null;
}
```

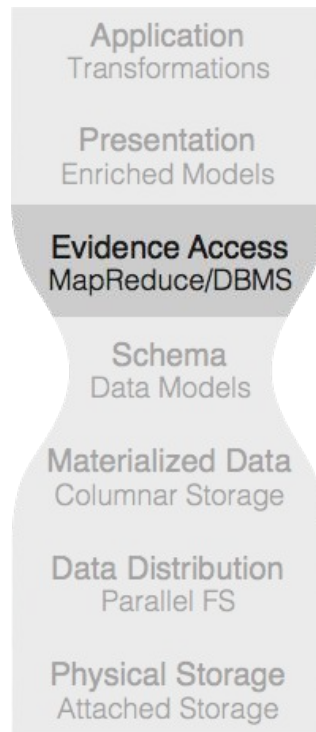Schema
Data Models

A schema provides a narrow waist

```
record AlignmentRecord {
    union { null, Contig } contig = null;
    union { null, long } start = null;
    union { null, long } end = null;
    union { null, int } mapq = null;
    union { null, string } readName = null;
    union { null, string } sequence = null;
    union { null, string } mateReference = null;
    union { null, long } mateAlignmentStart = null;
    union { null, string } cigar = null;
    union { null, string } qual = null;
    union { null, string } recordGroupName = null;
    union { int, null } basesTrimmedFromStart = 0;
    union { int, null } basesTrimmedFromEnd = 0;
    union { boolean, null } readPaired = false;
    union { boolean, null } properPair = false;
    union { boolean, null } readMapped = false;
    union { boolean, null } mateMapped = false;
    union { boolean, null } firstOfPair = false;
    union { boolean, null } secondOfPair = false;
    union { boolean, null } failedVendorQualityChecks = false;
    union { boolean, null } duplicateRead = false;
    union { boolean, null } readNegativeStrand = false;
    union { boolean, null } mateNegativeStrand = false;
    union { boolean, null } primaryAlignment = false;
    union { boolean, null } secondaryAlignment = false;
    union { boolean, null } supplementaryAlignment = false;
    union { null, string } mismatchingPositions = null;
    union { null, string } origQual = null;
    union { null, string } attributes = null;
    union { null, string } recordGroupSequencingCenter = null;
    union { null, string } recordGroupDescription = null;
    union { null, long } recordGroupRunDateEpoch = null;
    union { null, string } recordGroupFlowOrder = null;
    union { null, string } recordGroupKeySequence = null;
    union { null, string } recordGroupLibrary = null;
    union { null, int } recordGroupPredictedMedianInsertSize = null;
    union { null, string } recordGroupPlatform = null;
    union { null, string } recordGroupPlatformUnit = null;
    union { null, string } recordGroupSample = null;
    union { null, Contig } mateContig = null;
}
```


Application
Transformations

Presentation
Enriched Models

Evidence Access
MapReduce/DBMS

Schema
Data Models

Materialized Data
Columnar Storage

Data Distribution
Parallel FS

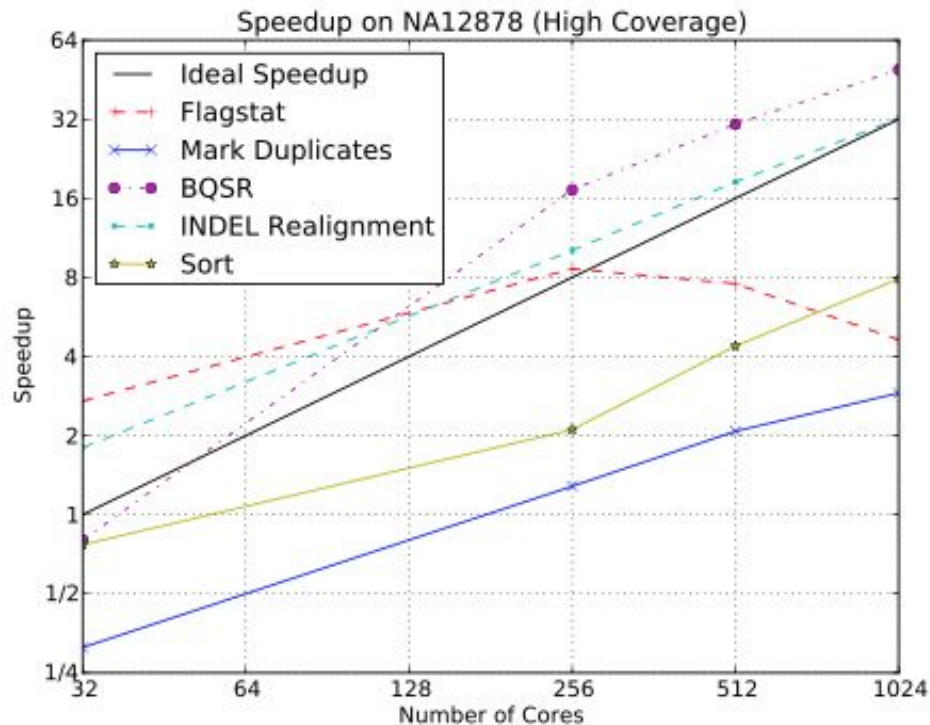Physical Storage
Attached Storage

We can use our stack to accelerate common queries!

- In genomics, we commonly have to find observations that overlap in a coordinate plane
- This coordinate plane is genomics specific, and is known a priori
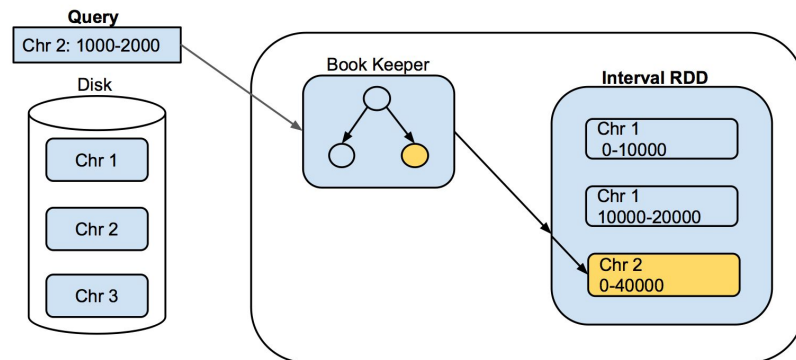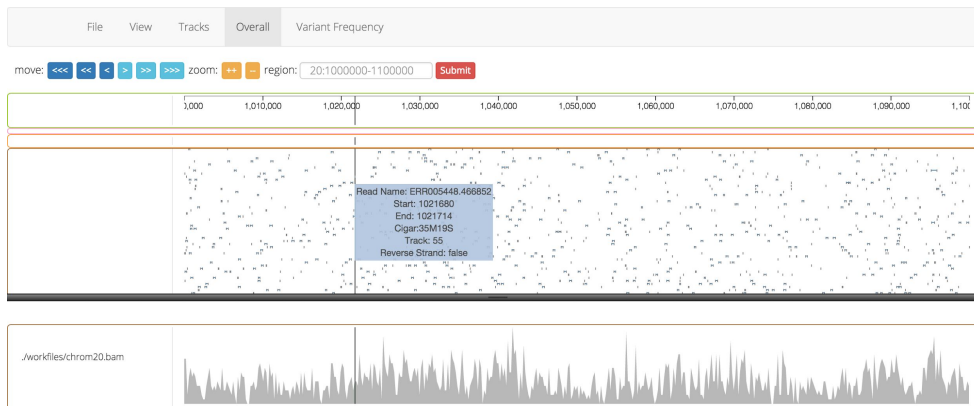- We can use our knowledge of the coordinate plane to implement a fast overlap join

Application
Transformations

Presentation
Enriched Models

Evidence Access
MapReduce/DBMS

Schema
Data Models

Materialized Data
Columnar Storage

Data Distribution
Parallel FS

Physical Storage
Attached Storage

ADAM/Spark Yields Horizontal Scalability

- 30–50x speedup over traditional implementations
- Speedup extends to O (200MB / node)
- 3x improvement in analysis cost

Speedup on NA12878 (High Coverage)

Legend:
- Ideal Speedup
- Flagstat
- Mark Duplicates
- BQSR
- INDEL Realignment
- Sort

X-axis: Number of Cores (32, 64, 128, 256, 512, 1024)
Y-axis: Speedup (1/4, 1/2, 1, 2, 4, 8, 16, 32, 64)

This means we can support genomic EDA as well...

- Narrow waist in stack → can swap in/out stages
- If I want to run interactive queries against genomic loci, swap in an RDD implementation that is optimized for point/range queries

Using ADAM+Spark+Toil
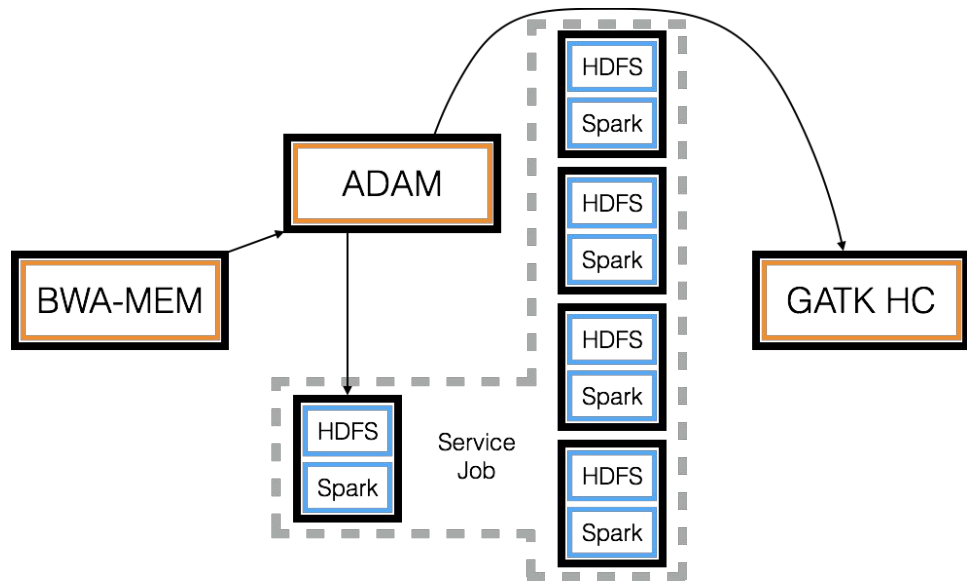
- On-going validation of ADAM on Toil against existing "best practice" tools
- Using Simons Genome Diversity Project:
  - 260 individuals from 127 ethnic groups: more than 100TB of genomic data
- At peak scale, we will be running on ~1,000 r3.8xlarge instances

Toil: Pipeline Architecture for Massive Workflows

- Work led by UCSC/BD2K Center
- Massively Scalable — tested on 32,000 cores
- Resumable after failure of *any* node
- Portable: installed with a single command
  - Amazon, OpenStack, Azure, HPC, and Google (soon)
- Simple: built entirely in Python
- Open-source → github.com/BD2KGenomics/toil

Develop workflows locally…
Deploy at scale without changing source code!

TOIL

Massively Parallel Workflows

Toil is a DAG metascheduler

- Users decompose a workflow into jobs and data:
  - Job → python function or class
  - Toil runs jobs on a single node by delegating to a sub-scheduler
  - Files get written to persistent FileStore for communication between jobs running on different nodes, use local cache to improve performance
- Supports many underlying schedulers/file stores
- When running in the cloud, jobs can be autoscaled
  - Happens transparently to the running jobs

Accommodating "clusters" in a job-oriented workflow

- Hint: a cluster is just a collection of single node jobs!
  - However, the fate of all of the jobs are tied!
- Service jobs persist for the whole lifetime of the task that spawned them
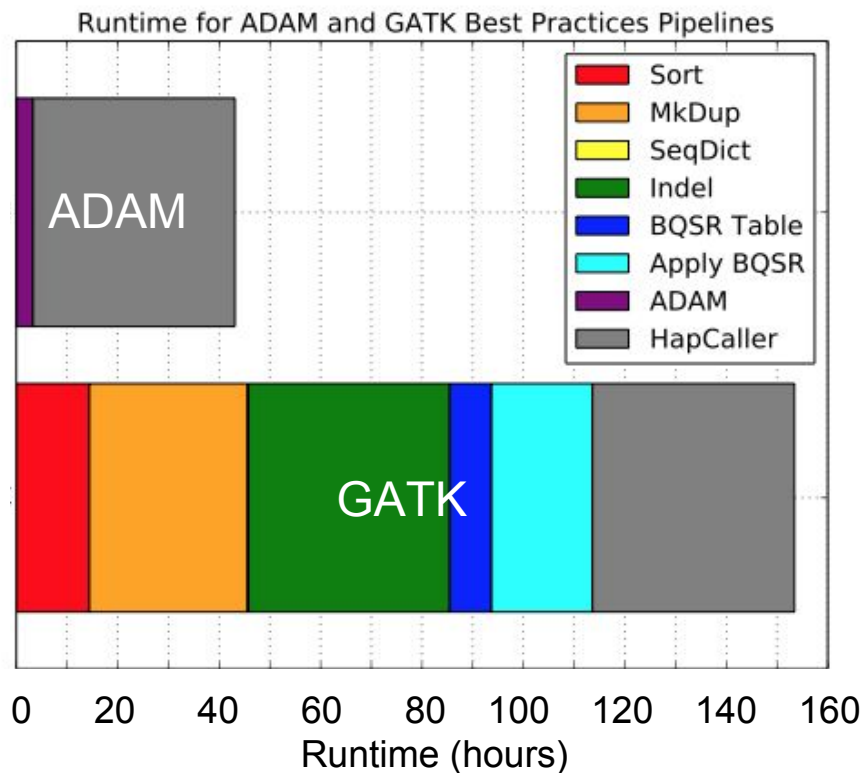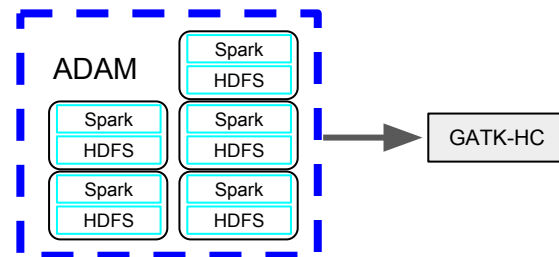- Spark has a clean mapping to a set of services
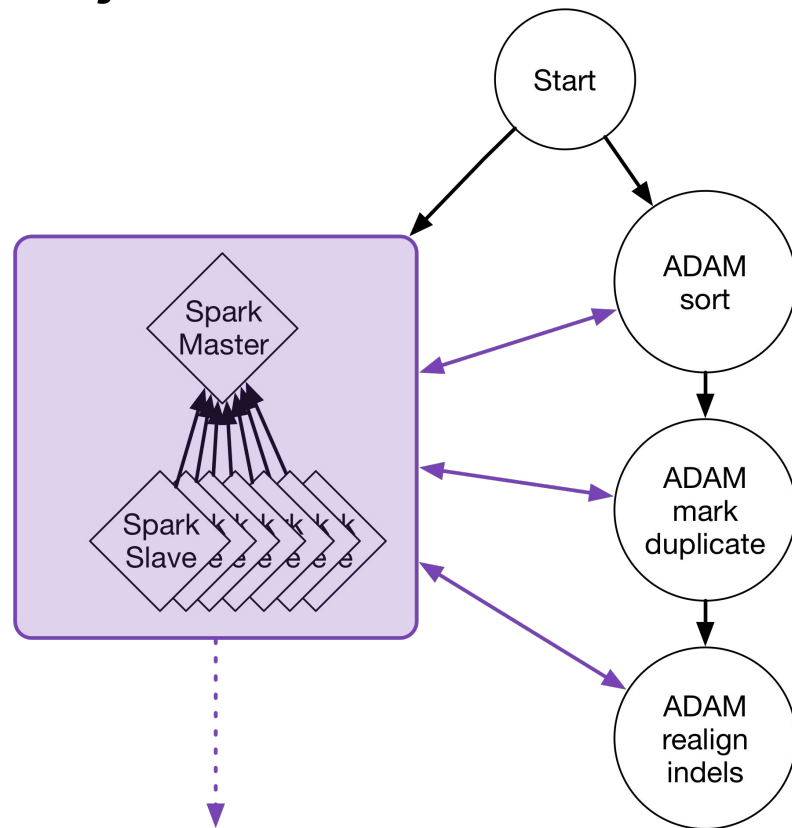
Vivian et al, 2016

ADAM Evaluation
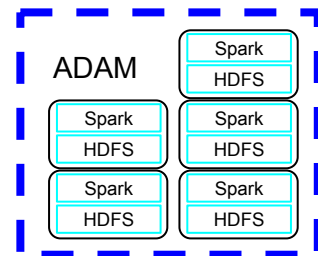
- ADAM produces statistically equivalent results to the GATK best practices pipeline
- Read preprocessing is >30x faster and 3x cheaper
- In the process of recalling the Simons Genome Diversity Project using ADAM
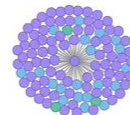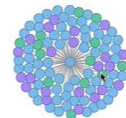- We have a working pipeline using both HG19 and GRCh38

Runtime for ADAM and GATK Best Practices Pipelines

Legend:
- Sort (Red)
- MkDup (Orange)
- SeqDict (Yellow)
- Indel (Green)
- BQSR Table (Blue)
- Apply BQSR (Cyan)
- ADAM (Purple)

ADAM

GATK

Runtime (hours): 0 20 40 60 80 100 120 140 160
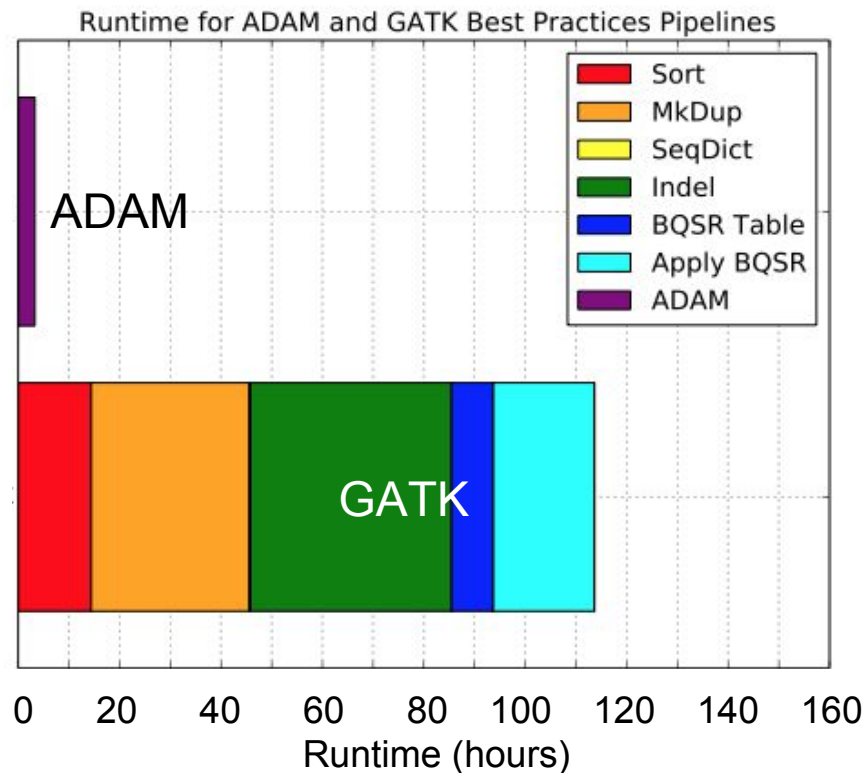
ADAM Evaluation

- ADAM produces statistically equivalent results to the GATK best practices pipeline
- Our end-to-end pipeline is 3.5x faster while also being 4x cheaper
- In the process of recalling the Simons Genome Diversity Project using ADAM
- We have a working pipeline using both HG19 and GRCh38

Runtime for ADAM and GATK Best Practices Pipelines

Legend:
- Sort
- MkDup
- SeqDict
- Indel
- BQSR Table
- Apply BQSR
- ADAM
- HapCaller

Runtime (hours)

Check out the code!

ADAM: https://github.com/bigdatagenomics/adam

Check out a demo!

https://databricks.com/blog/2016/05/24/

genome-sequencing-in-a-nutshell.html

Run ADAM in Databricks CE!

http://goo.gl/xK8x7s

Acknowledgements