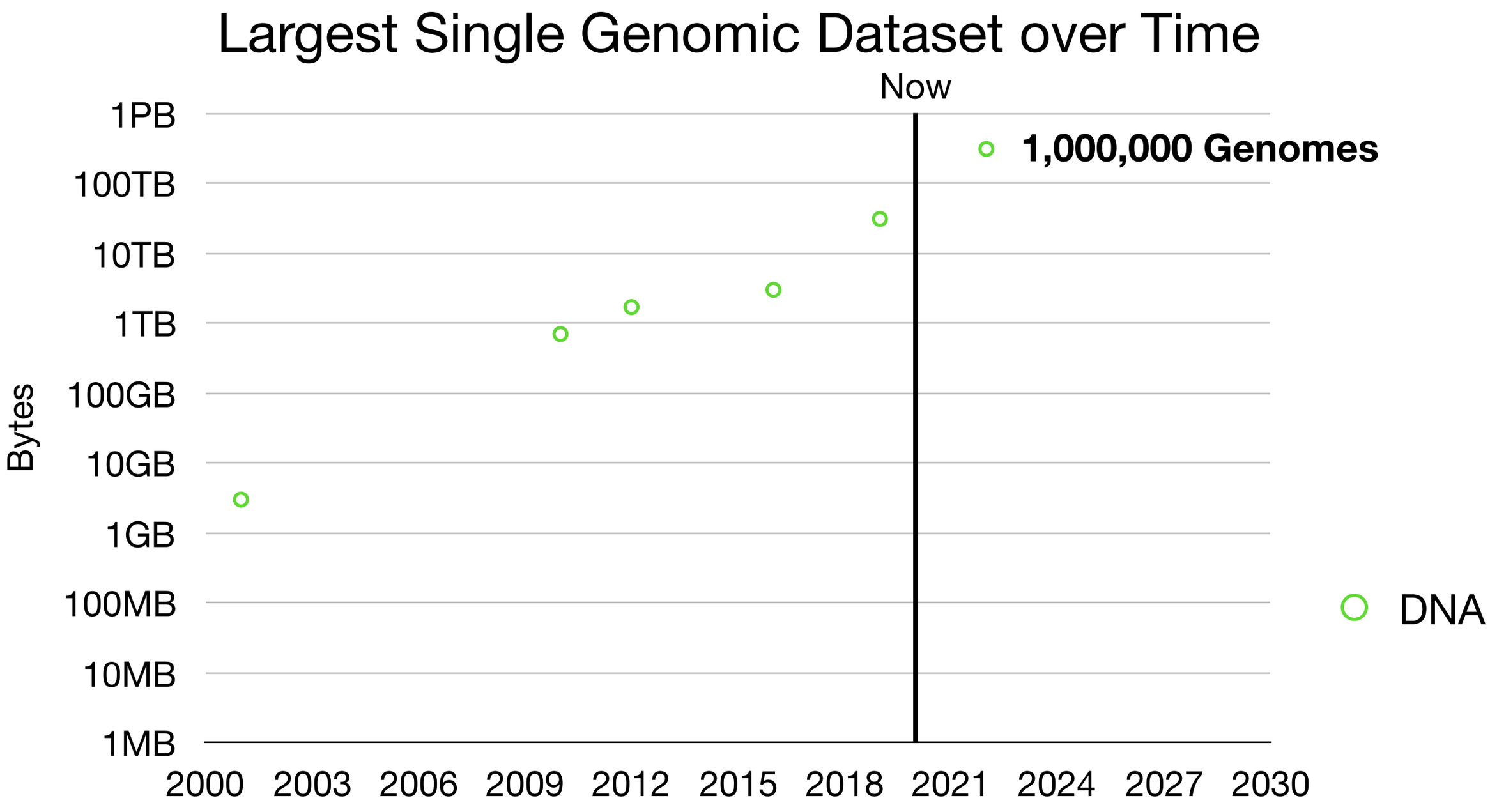
The Design and Implementation of a System and Language for the Analysis of Large Genetics Datasets

Daniel King
@danking00
Hail Team
The Broad Institute

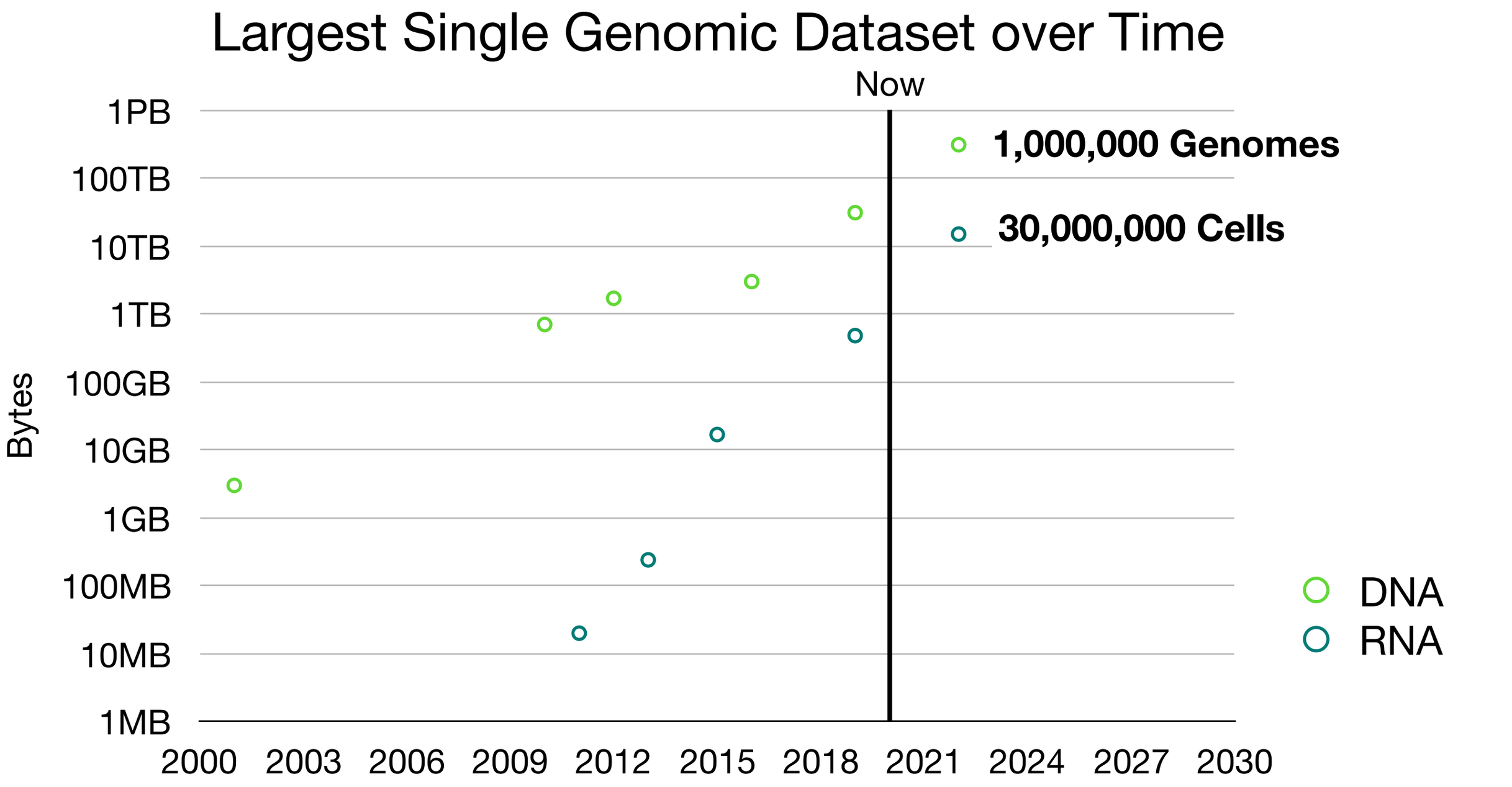

THE ELI AND EDYTHE L.
BROAD
INSTITUTE

Broad Institute was launched in 2004 to improve human health by using genomics to advance our understanding of the biology and treatment of human disease, and to help lay the groundwork for a new generation of therapies.

Data in Biology



Data in Biology

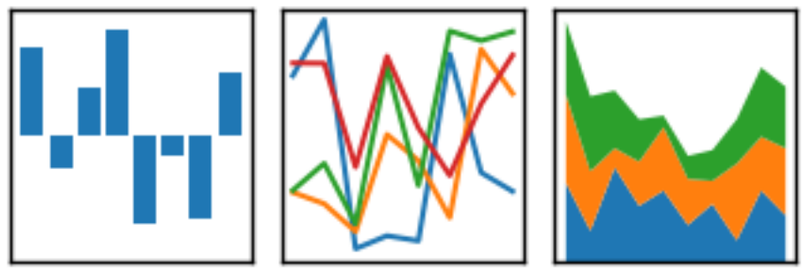


Kumar, et. al. “Understanding development and stem cells using single cell-based analyses of gene expression”

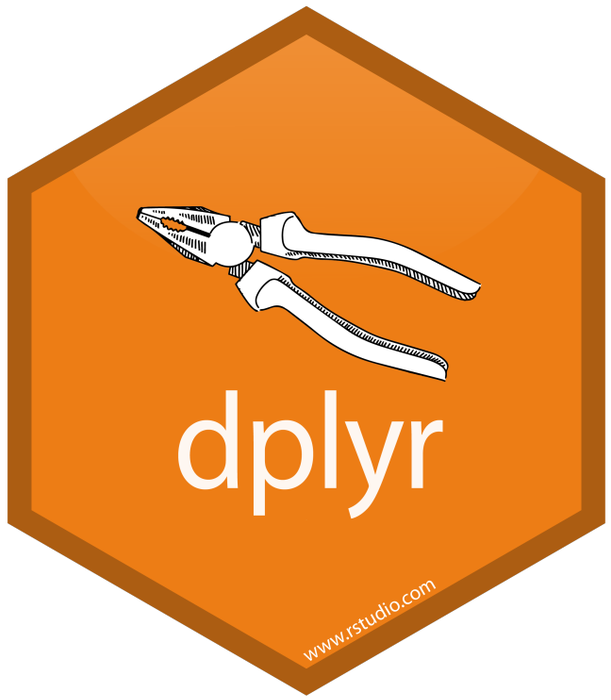
Python API

pandas

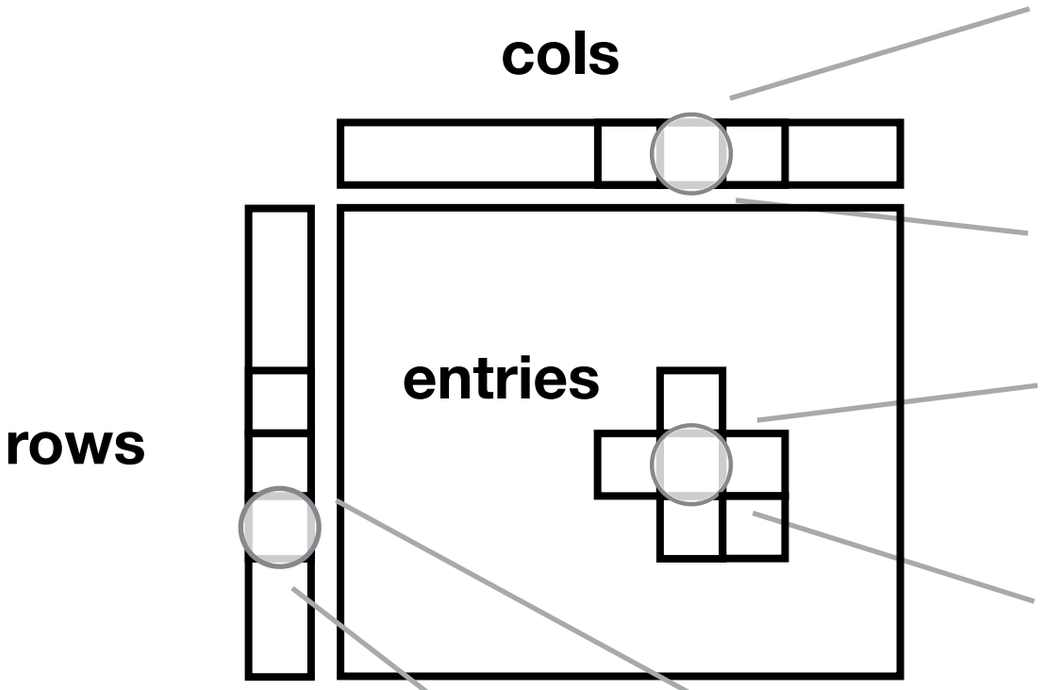
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



TensorFlow



MatrixTable

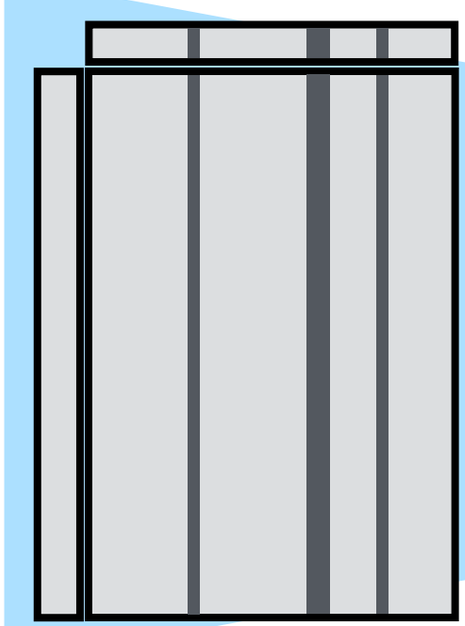


ID	has_disease	age	…
`str`	`bool`	`int32`	
NA12878	True	67	…

genotype	allele_depth	depth	…
`call`	`array<int32>`	`int32`	
0/1	8,11	19	…

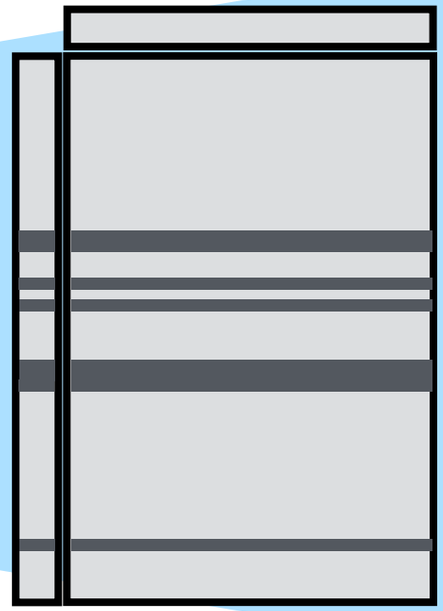
locus	probability_loss_of_func	…
`locus<GRCh37>`	`float64`	
17:37282	0.75334	…

Distributed Relational Ops

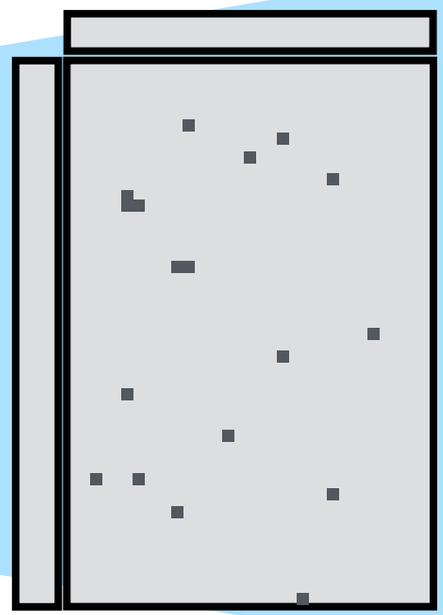


```
mt = mt.filter_cols(mt.age > 45)
```

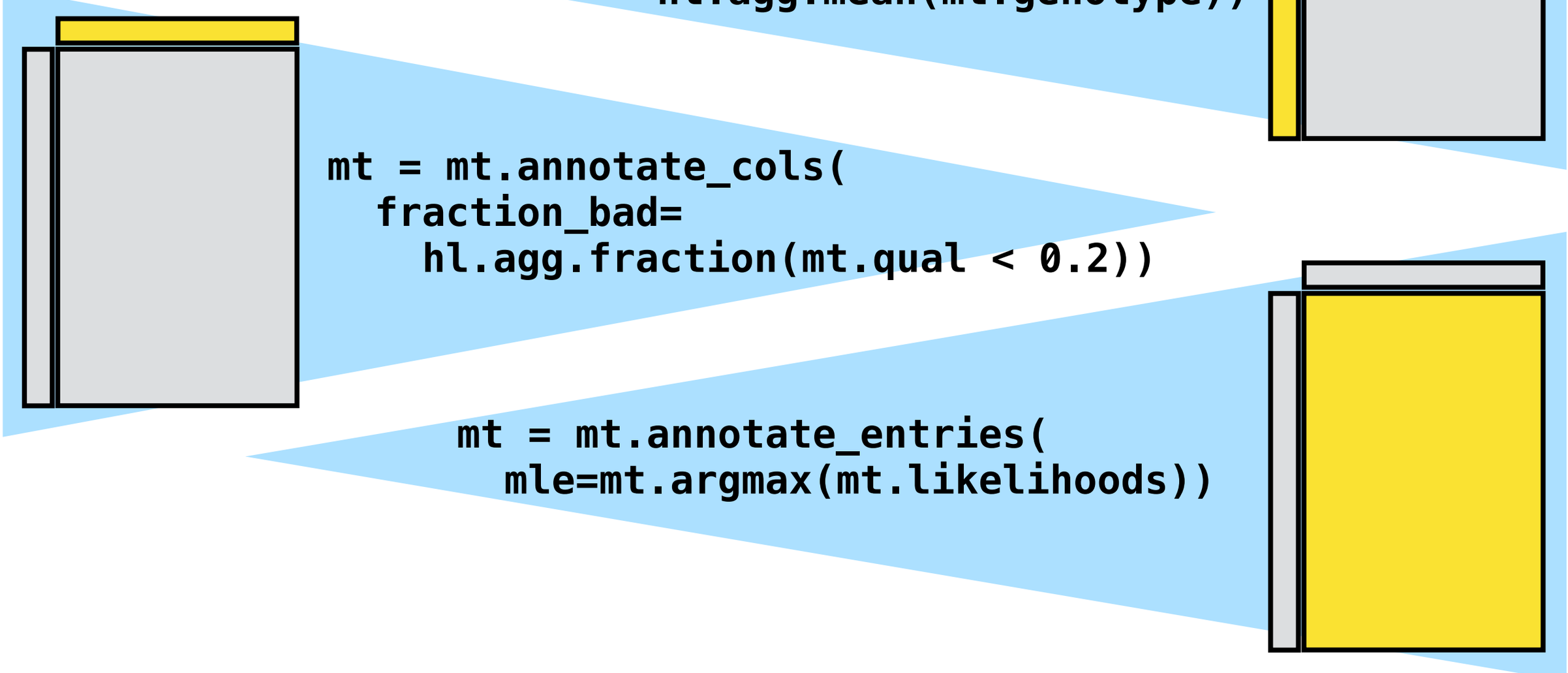
```
mt = mt.filter_rows(
        mt.variant_quality < 0.03)
```



```
mt = mt.filter_entries(
        mt.genotype_quality > 0.8)
```

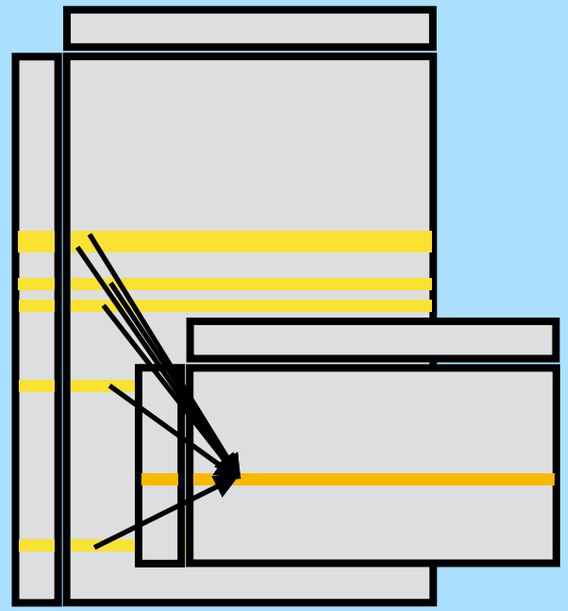


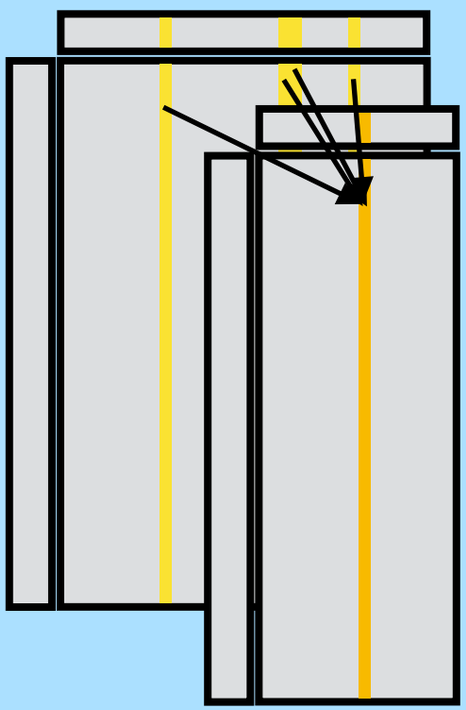
Distributed Relational Ops



Distributed Relational Ops

```
mt = mt.group_rows_by(
  mt.gene
).aggregate(
  hl.agg.sum(mt.n_mutations)
)
```





```
mt = mt.group_cols_by(
  mt.ancestral_pop
).aggregate(
  hl.agg.any(mt.n_mutations > 0)
)
```

Scalar Expressions

- Primitives: `1, 3.141, “str”, True`
- Collections: `[1,2], (1,2), {1,2}, {“a”: 1, “b”}`
- Genetics: `hl.interval(3,4), hl.call(0,1), hl.locus(“X”, 1231)`
- Structured Data: `hl.struct(x=3, y=hl.struct(z=[1,2,3]), z=“f”)`
- ndarrays: `hl.nd.array([[1, 0], [0, 1]])`
- Every type can be missing: `hl.null(hl.tint32)`

Scalar Expressions

- Primitives: `1, 3.141, “str”, True`
- Collections: `[1,2], (1,2), {1,2}, {“a”: 1, “b”}`
- Genetics: `hl.interval(3,4), hl.call(0,1), hl.locus(“X”, 1231)`
- **Structured Data: `hl.struct(x=3, y=hl.struct(z=[1,2,3]), z=“f”)`**
- ndarrays: `hl.nd.array([[1, 0], [0, 1]])`
- Every type can be missing: `hl.null(hl.tint32)`

Scalar Expressions

- Primitives: `1, 3.141, “str”, True`
- Collections: `[1,2], (1,2), {1,2}, {“a”: 1, “b”}`
- Genetics: `hl.interval(3,4), hl.call(0,1), hl.locus(“X”, 1231)`
- Structured Data: `hl.struct(x=3, y=hl.struct(z=[1,2,3]), z=“f”)`
- **ndarrays: `hl.nd.array([[1, 0], [0, 1]])`**
- Every type can be missing: `hl.null(hl.tint32)`

Scalar Expressions

- Primitives: `1, 3.141, “str”, True`
- Collections: `[1,2], (1,2), {1,2}, {“a”: 1, “b”}`
- Genetics: `hl.interval(3,4), hl.call(0,1), hl.locus(“X”, 1231)`
- Structured Data: `hl.struct(x=3, y=hl.struct(z=[1,2,3]), z=“f”)`
- ndarrays: `hl.nd.array([[1, 0], [0, 1]])`
- **Every type can be missing: `hl.null(hl.tint32)`**

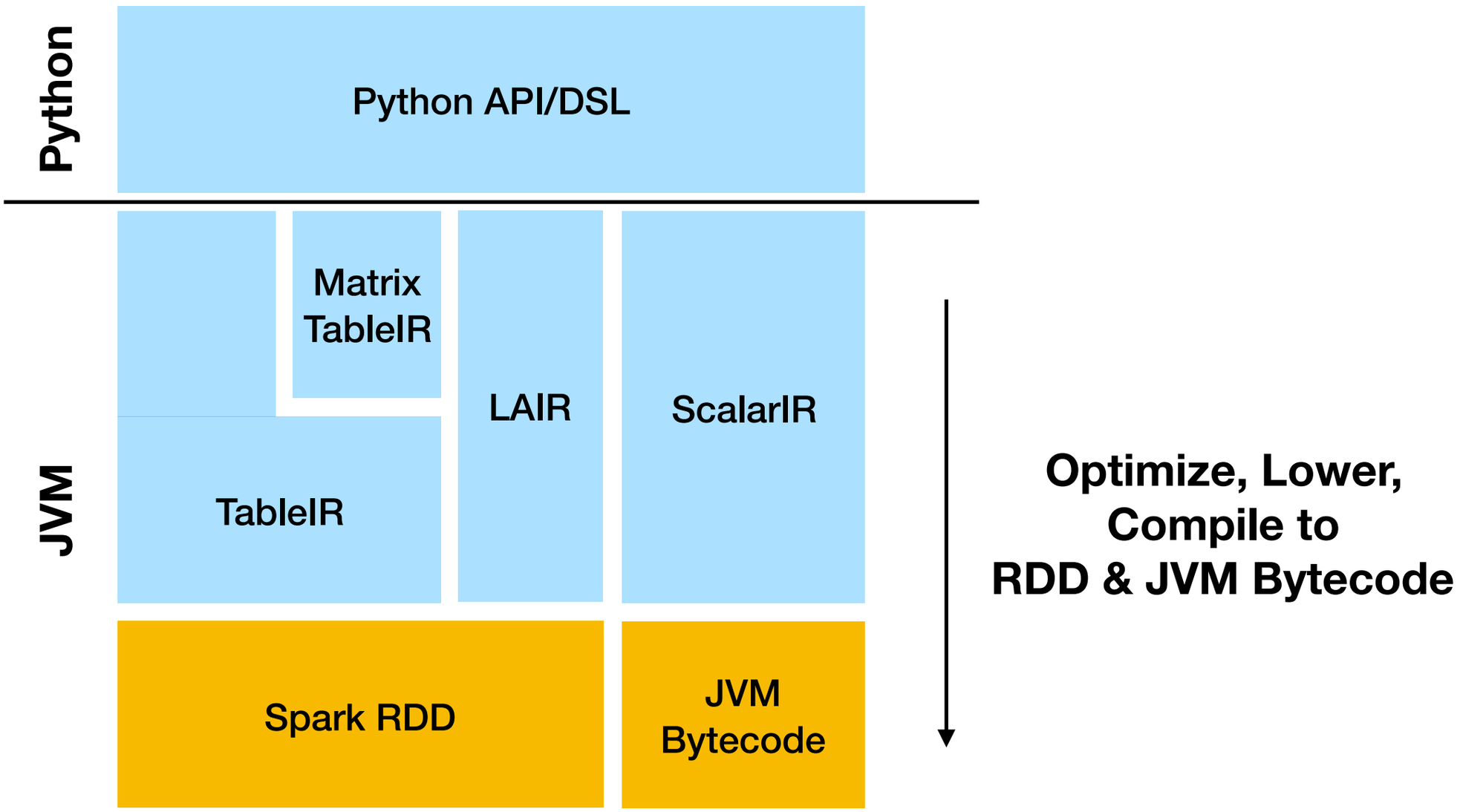
Aggregations and Scans

- Standard: `sum, mean, count, explode, …`
- Approximate: `approx_quantiles, approx_cdf`
- Combinators: `filter, group_by`

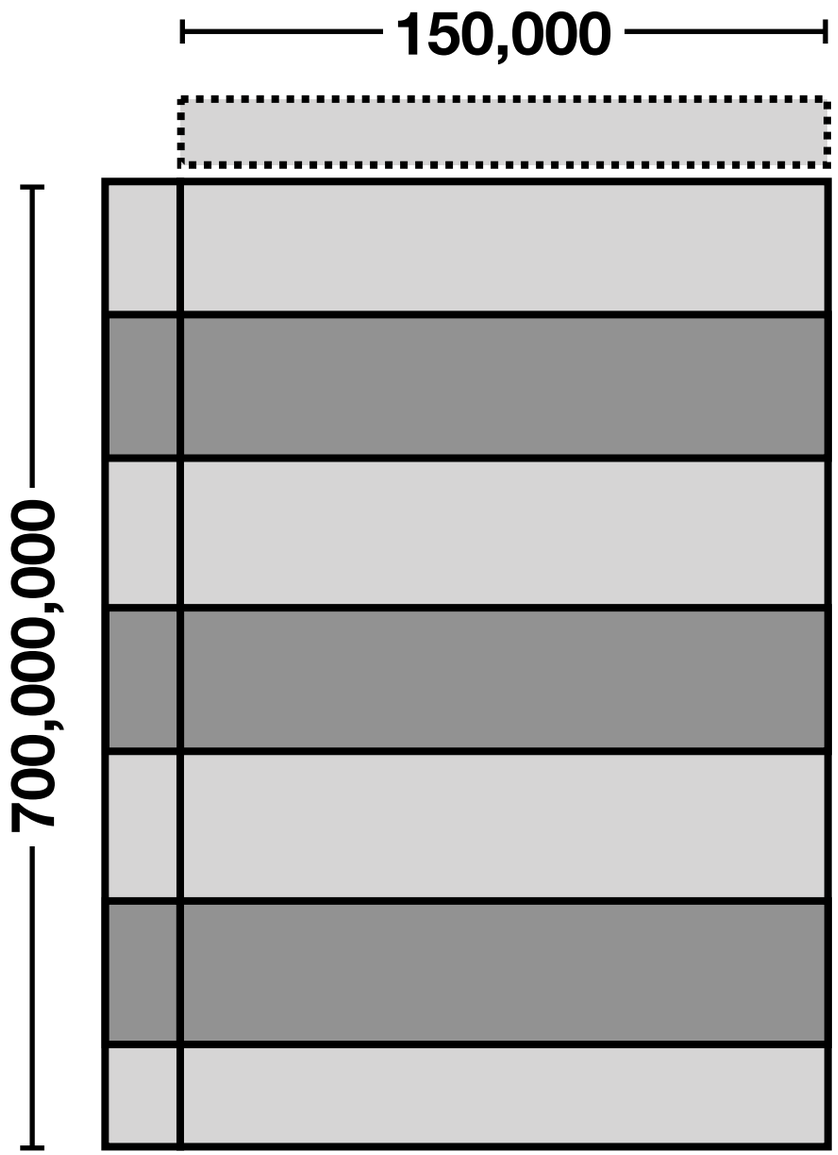
Distributed Linear Algebra

- Used to work with 10M by 10M matrices
- `1 + v, v + A, v @ A, A @ B, …`
- `pca(A)`

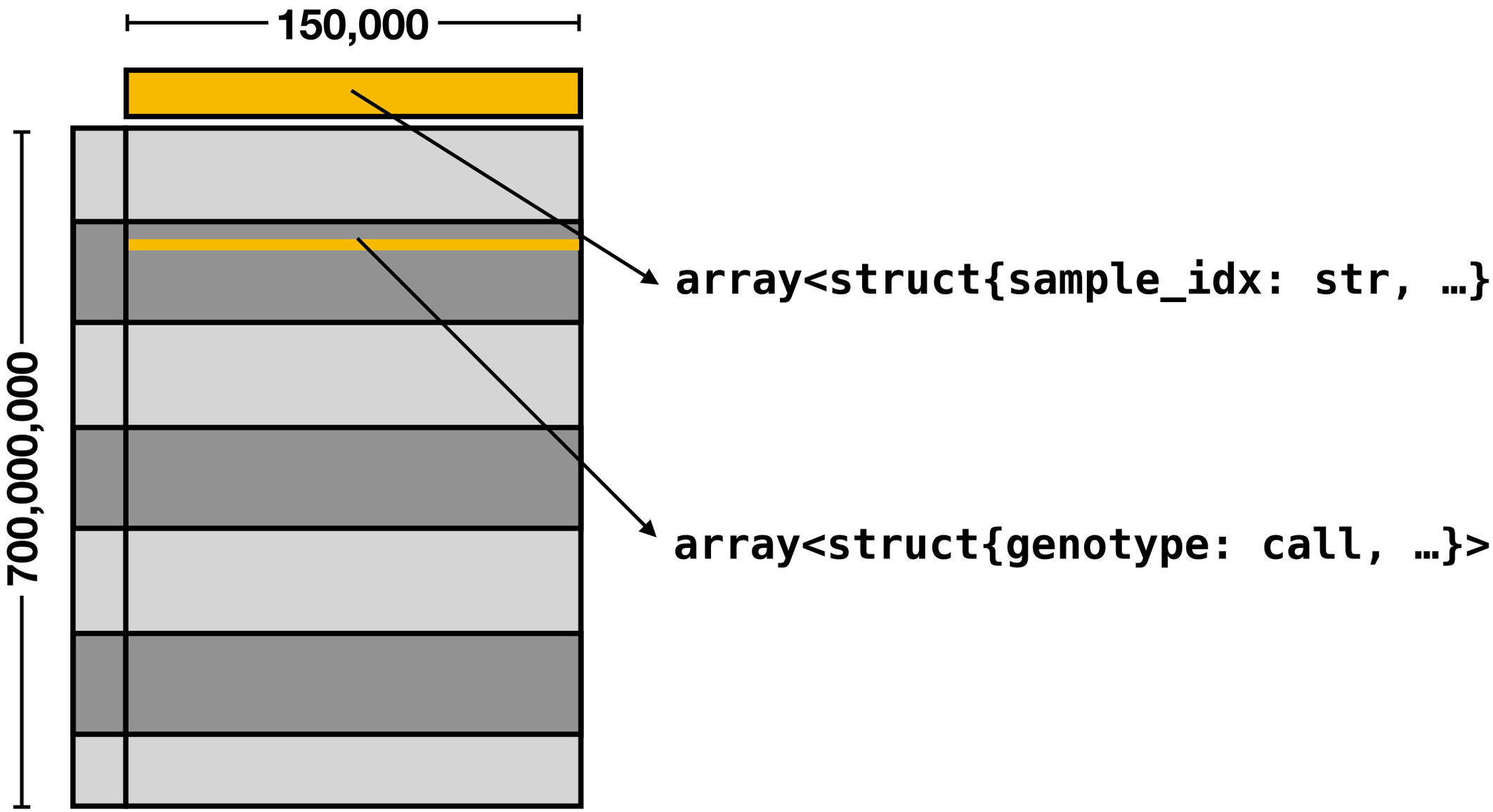
Implementation



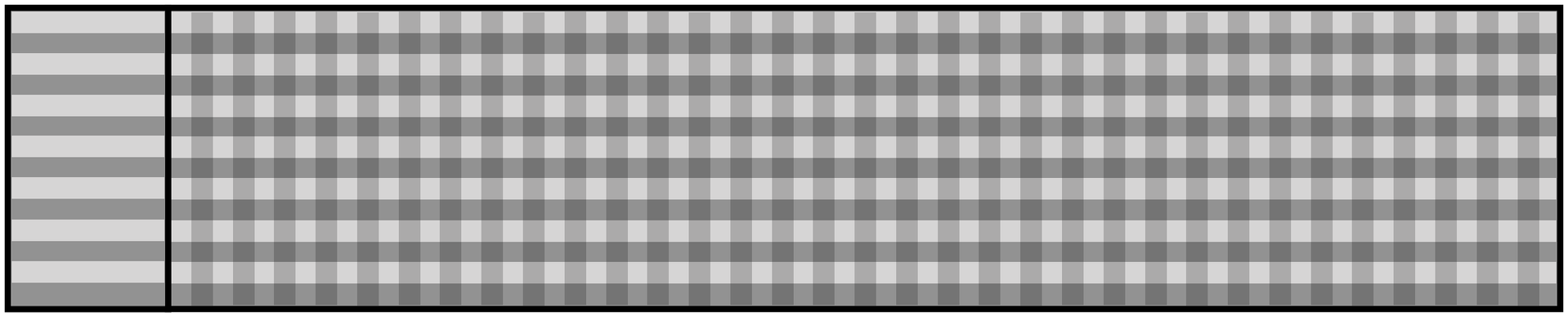
Genetics Matrices



Wide Schemas



Region Allocation



- Two natural regions: partition and row
- Cheap allocations, O(1) deallocation of an entire region
- No GC process using cycles, no pauses
- Related work: “Broom”, Gog, et. al

Ordering & Joins

- Streaming, cache-friendly data access patterns
- Join datasets keyed by `K` with those keyed by `Interval[K]`
- Preserve ordering on-disk

Future: Runtime

- An unmanaged run-time with access to all of x86
- No-cost interoperation with R and Python
- Efficiency and cost-savings from elastic compute
- Dependency graphs beyond map-reduce
- Shared elastic compute => high utilization, low latency

Future: Memory

- Choose memory representations based on cost-estimates
- Mix columnar and row representations
- Mix sparse and dense representations

Future:
Containerized Binaries

- Goal: Schedule tens of millions of containers
  on tens of thousands of cores

- Python API for DAGs of containerized binaries

- Data transfer into/out-of the relational & linear language

- Prototype: schedules 100k, 15 min jobs across 25k cores

Far Future: `TensorTable`

- What's the formal model here?
- How do we efficiently compile and distribute this?
- What's the Python surface syntax?
- Are axes named or ordinal?

The Team

- Unorthodox backgrounds: CS, math, chem, bio, physics, …
- Incentives aligned to the scientists around us
- Eyes on the prize: accelerate the discovery of treatments

Thanks

- **The scientists:**
  Benjamin Neale & Lab
  Daniel MacArthur & Lab
  Konrad Karczewski
  gnomAD team
  ATGU

- **Our funders:**
  Stanley Center for Psychiatric Disease,
  Jeremy Wertheimer,
  Chan-Zuckerberg Initiative,
  National Human Genome Research Institute,
  National Institute of Mental Health,
  National Institute of Diabetes and Digestive and Kidney Diseases

And of course, the IBM PL Day organizers

Let's Talk!

@danking00

dking@broadinstitute.org

@hailgenetics

hail-team@broadinstitute.org

github.com/hail-is/hail

Extra Slides

Related Work

- Off-heap memory, GC issues in big-data row-processing:
  Tungsten (Armbrust, et. al), Broom (Gog, et. al)

- Shared elastic relational engine:
  Dremel/BigQuery (Melnik, et. al)

- DataFrame APIs:
  pandas (McKinney), R (Ihaka, et. al), dplyr (Wickham)

- Matrix Databases:
  SciDB (Stonebreaker, et. al), TileDB (Papadopoulos, et. al)

- Fault-tolerant, elastic linear-algebra:
  numpywren (Shankar, et. al)

- Massively scalable elastic container scheduling:
  AWS Fargate

- Pluggable Numeric Tensor Representations & Numeric Tensor Algebra Compilation
  “TACO” (Kjolstad, et. al), “Halide” (Ragan-Kelley, et. al)