Guiding Principles for Engineering ML Tools & Frameworks

Guest Lecture, CS Department, Whitman College

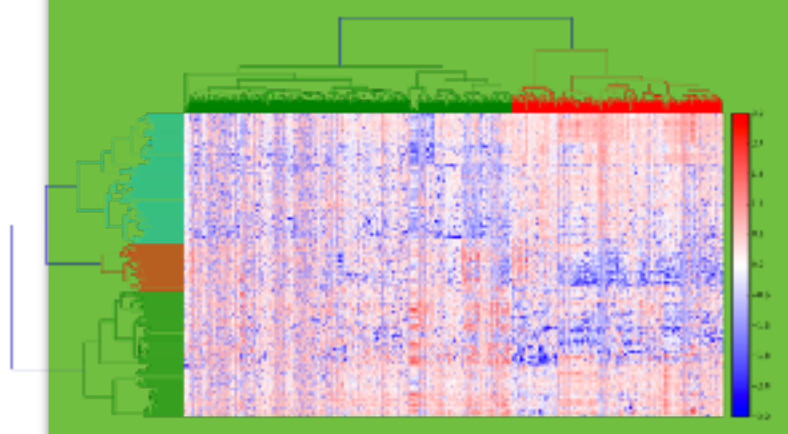
Damian Eads, PhD, Principal Data Engineer - ML Tools and Frameworks
Founder, Wise Intelligent Systems
GE Digital

He/Him/His

March 25, 2019

Public Talk

Open Source



Over dozen projects

Research

Los Alamos
NATIONAL LABORATORY
EST. 1943

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

UNIVERSITY OF CAMBRIDGE

Industry

wise.io

November 2016


NETFLIX
The Netflix Prize

"It may be surprising to the academic community to know that only a fraction of the code … is actually doing 'machine learning'. A mature system might end up being (at most) ***5% machine learning code*** *and (at least)* ***95% glue code****."*

- Complex models erode abstraction boundaries
- Data dependencies cost more than code dependencies: weak contracts
- System-level Spaghetti
- Changing External World

see also, Bottou (Facebook) ICML

Slide courtesy: Joshua Bloom

http://research.google.com/pubs/pub43146.html

Machine Learning: The High-Interest Credit Card of Technical Debt

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young
{dsculley,gholt,dgg,edavydov}@google.com
{toddphillips,ebner,vchaudhary,mwyoung}@google.com
Google, Inc

Abstract

Machine learning offers a fantastically powerful toolkit for building complex systems quickly. This paper argues that it is dangerous to think of these quick wins as coming for free. Using the framework of *technical debt*, we note that it is remarkably easy to incur massive ongoing maintenance costs at the system level when applying machine learning. The goal of this paper is highlight several machine learning specific risk factors and design patterns to be avoided or refactored where possible. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, changes in the external world, and a variety of system-level anti-patterns.

1 Machine Learning and Complex Systems

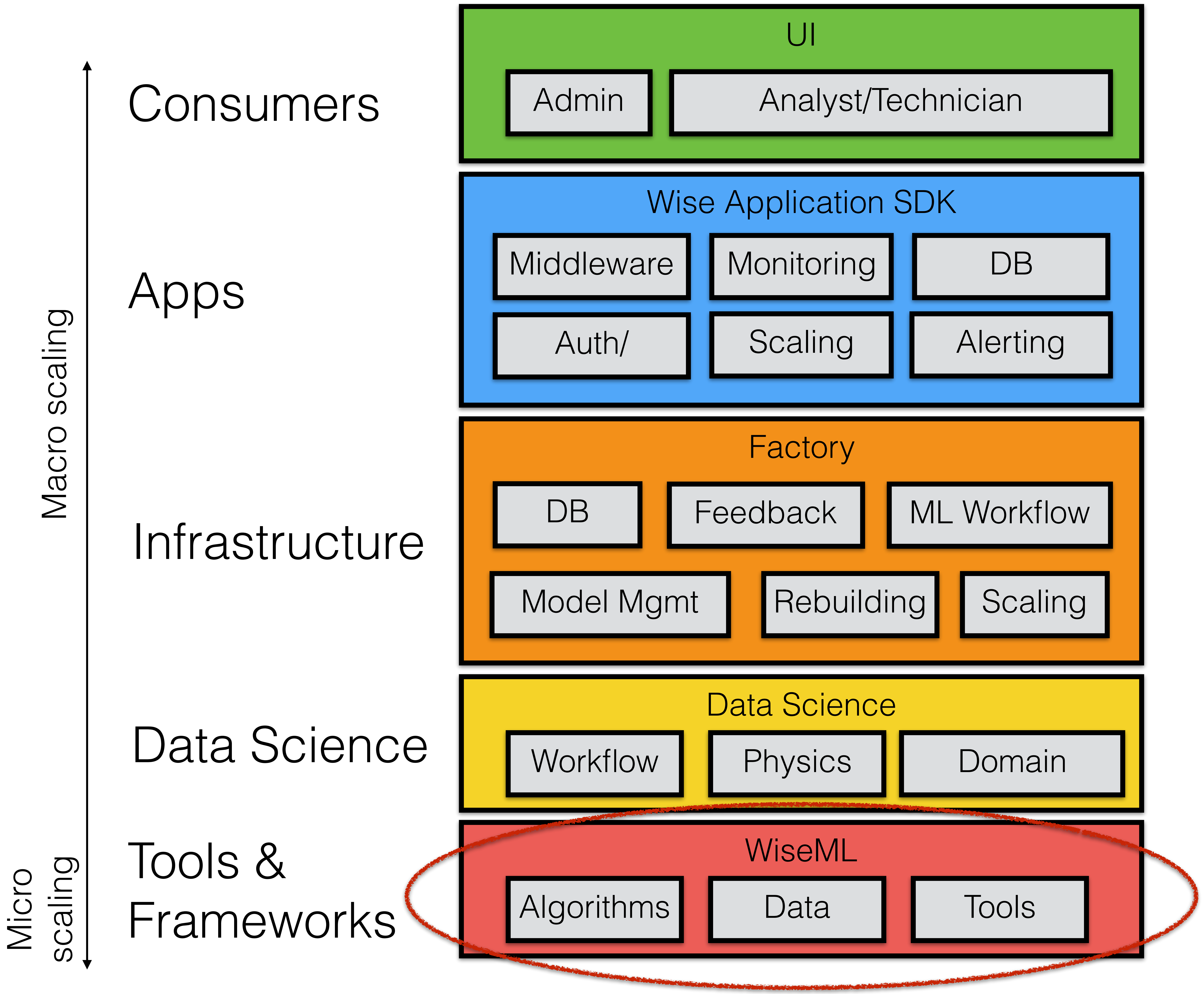
Real world software engineers are often faced with the challenge of moving quickly to ship new products or services, which can lead to a dilemma between speed of execution and quality of en-

- **Automated**: involve minimal person hours
- **Scalable**: size of data
- **Domain-aware:** lean on subject matter experts
- **Adaptable**: model retraining and domain expert feedback
- **Interpretable** by consumers (analysts, technicians, managers)
- **Repeatable**: can launch anywhere and give desired behaviors

Industrial Machine Learning at Wise.io

- Predictive Maintenance
- Operational Efficiency
- Technician/Analyst Workflows

Service-oriented Architecture



Philosophy

Speed

fast training, tuning & prediction
cache-exploitative
algorithmic innovations
template meta-programming

Memory Efficiency

usage ~ data set size
predictable footprint

Heterogeneous Data

mixed feature types
many categorical levels
1000s of classes
dense & sparse
text, time series, images

Flexible Models

fast traversal
compact
backwards compatible
flexible representation
fast parsing
directories

I/O

Minimize serialization
Exploit bandwidth

Diagnostics & Tuning

ILFIs
Validation Trees
Prediction Calibration

What ML algorithm?

	Data	Considerations
Deep Learning	images, time series, sensor data	More accurate, more sensitive to tuning, preprocessing and architecture crucial
Tree Ensembles	logs, meta-data, categorical-heavy, mixed data, high class	Easy to train out-of-the-box, insensitive to parameter changes, requires feature engineering, interpretability

Why another DataFrame?

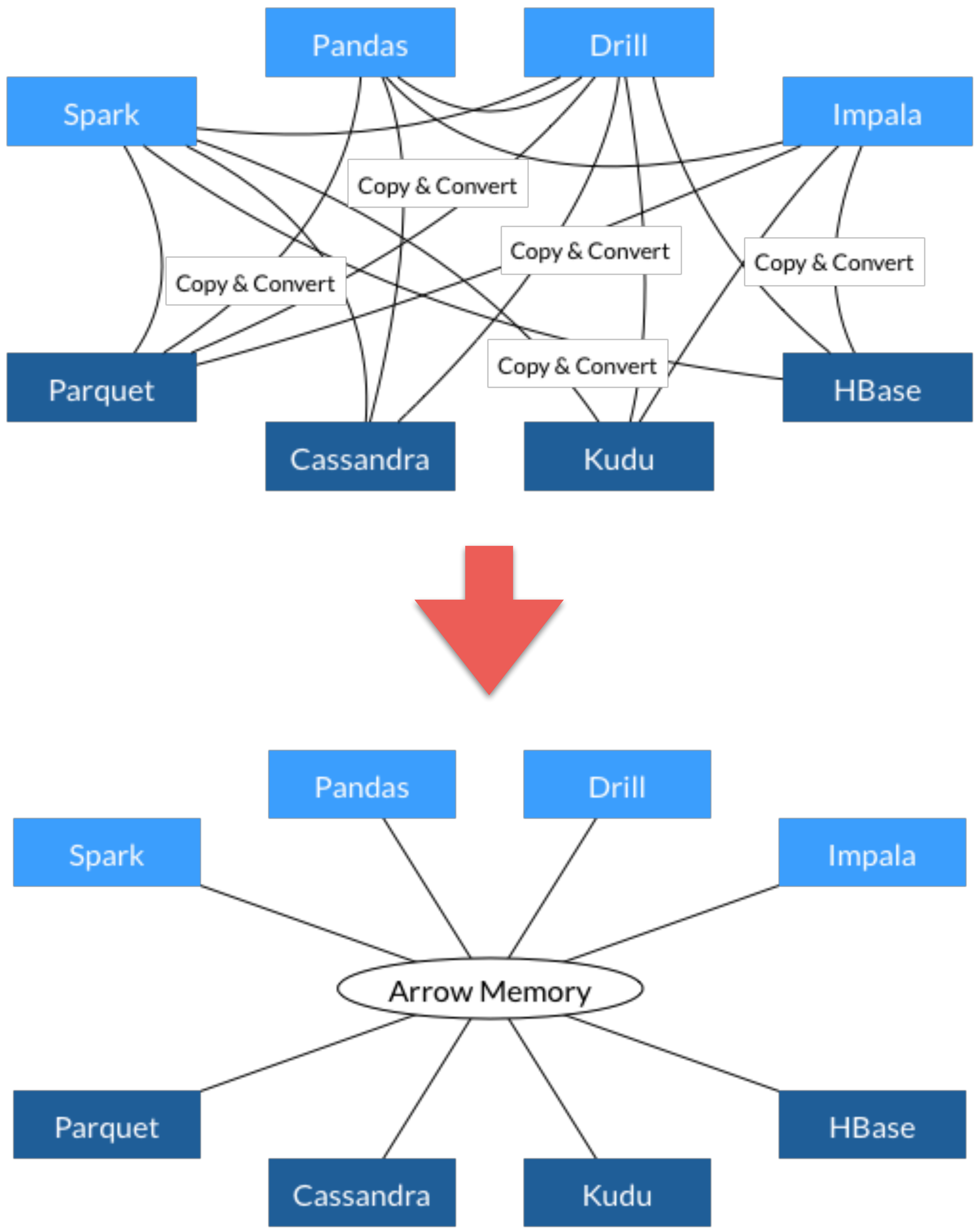
- Pandas DataFrame
  - Single machine
  - Implemented in Python & Cython
  - UDFs, joins, queries, aggregates
  - not optimized for machine learning
  - memory-inefficient

- Spark DataFrame
  - Multi-machine
  - Built on top of RDDs
  - UDFs, joins, queries, aggregates
  - Workers cannot share data
  - Does not handle array-heavy data well: time series, images
  - Row-based: problems
  - Static task graph

- WiseFrame
  - Off-heap
  - Out-of-core
  - Heterogeneous
  - Supports diverse heavy data types: images, time series, volumes, documents.
  - UDFs and Functional Primitives

Apache Arrow

- Serialization is a major bottleneck
- Polygotal columnar and array abstraction
- Flat representation
- Common execution framework
- Plasma: off-heap object store



Images credit: https://arrow.apache.org/

EdgeFrame Features

Off-heap

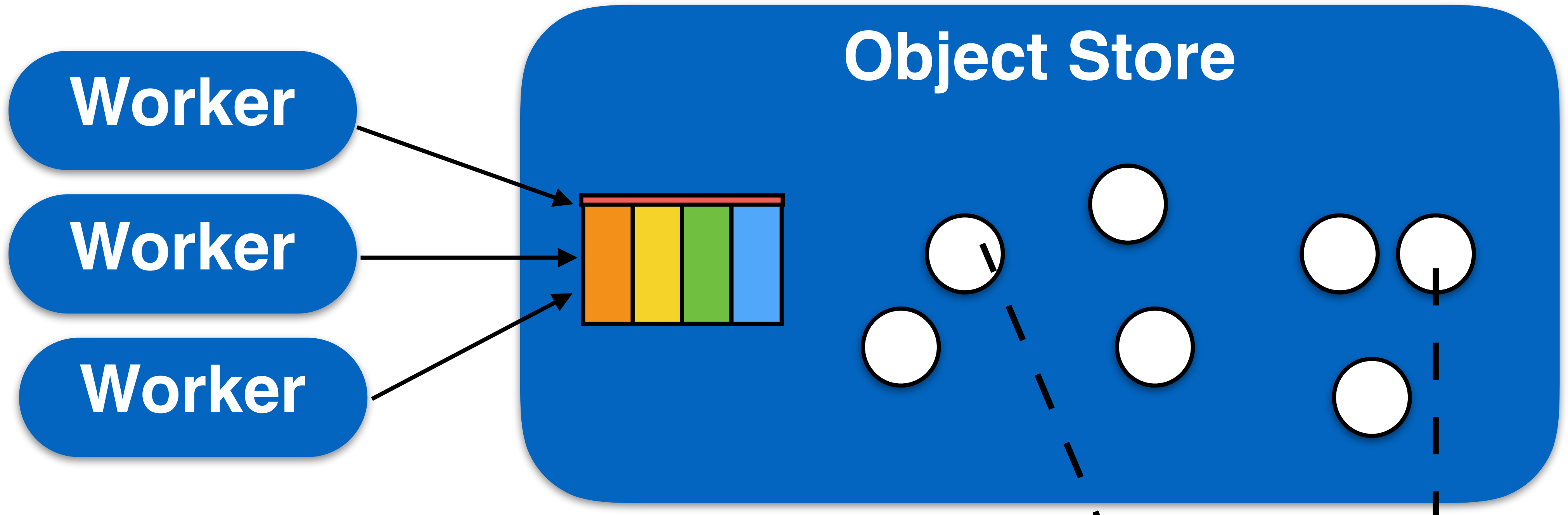
- Centrally managed
- Objects live in shared memory
- Multiple workers share the same data
- Object lifetimes beyond worker lifetimes
- Particularly efficient for array-heavy data

Array-Heavy

- Supports images, 3D volumes, time series
- Structured objects

Heterogeneous

- Text, Categorical, Datetimes, etc.
- Sparse & Dense



Id	Calibration	Created	Img	Report
31412	0.731581	2017/09/01		
94391	0.928414	2017/09/01		
74814	0.131253	2017/09/01		
87151	0.941457	2017/09/01		

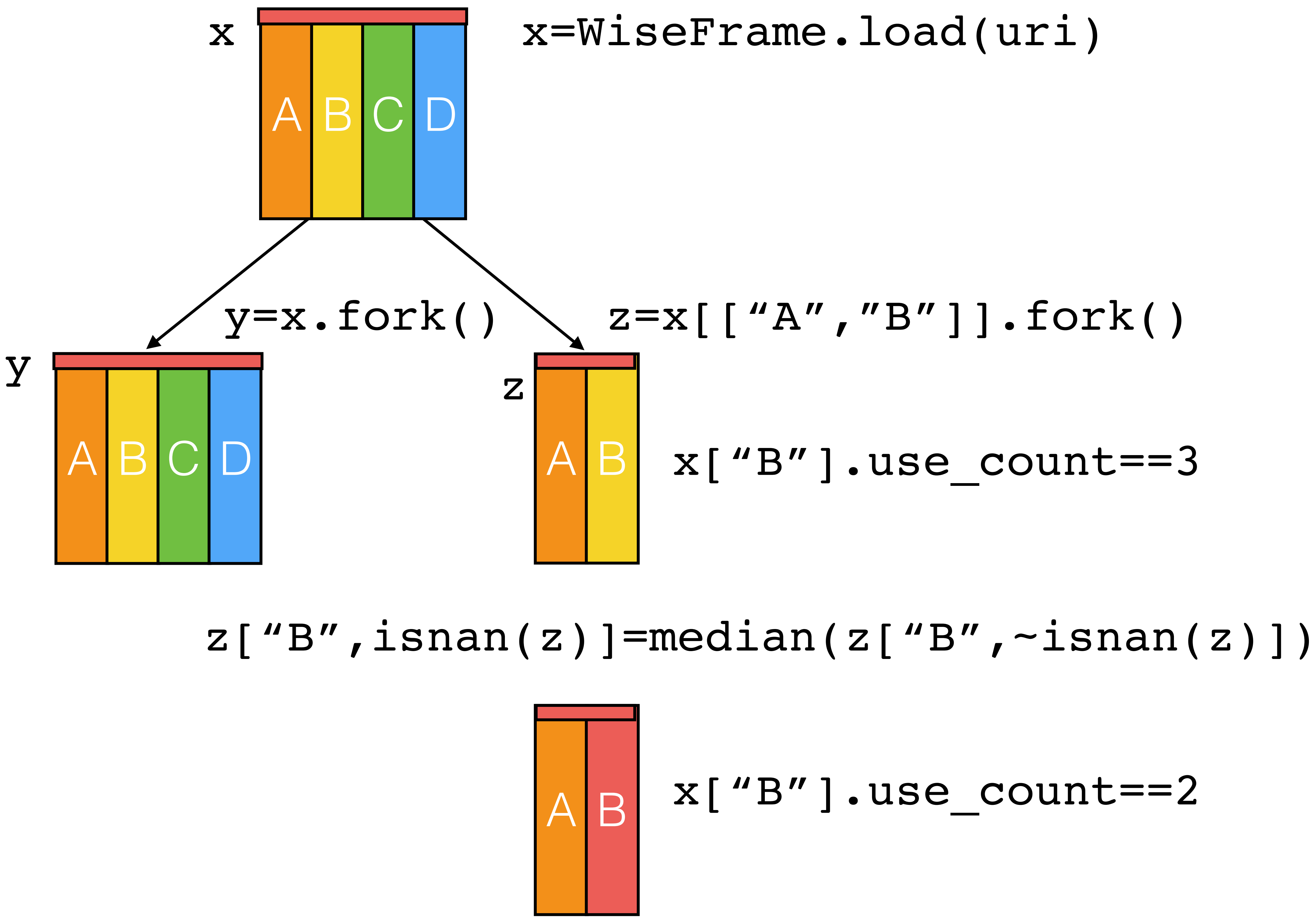
Out-of-core

- Use local SSDs as caching layer
- Support DataSets larger than RAM.

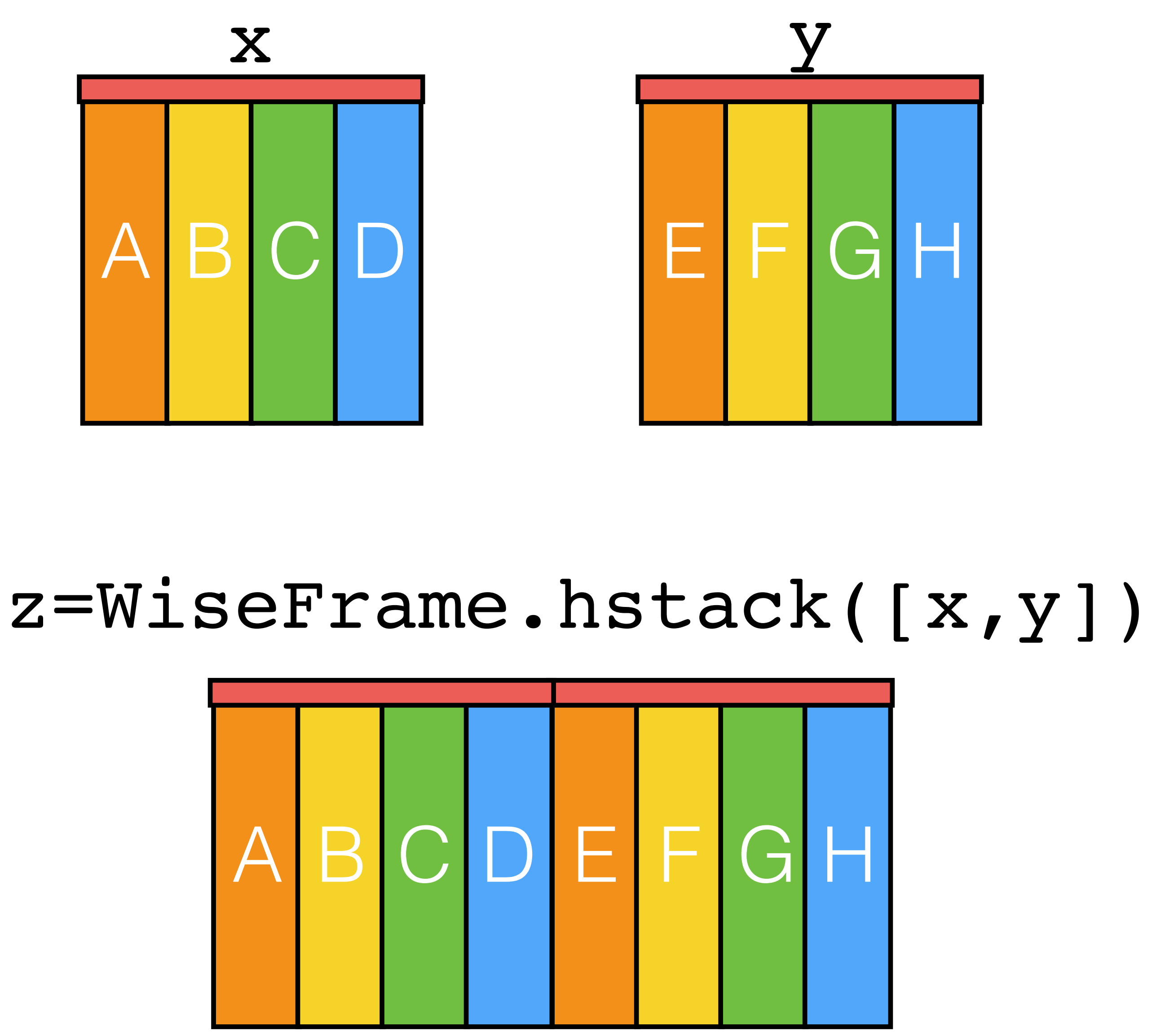
Mutability

- Efficient copying and reuse of building blocks.
- Columns of reference counted-blocks.
- Fork a data frame like a git repo!

Copy-on-write



Data Merging

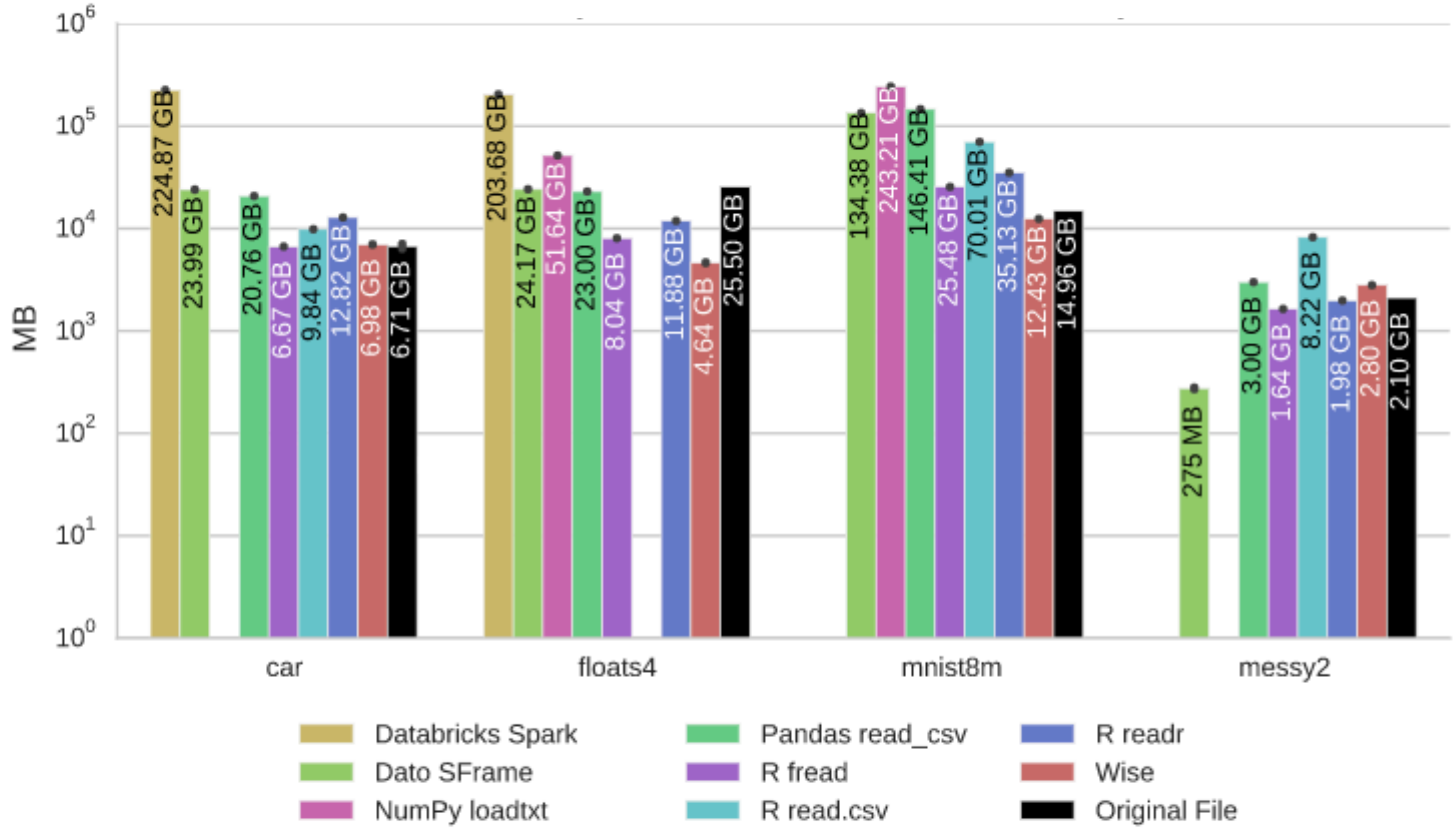


US 9,547,830

Off-heap Persistence

- avoid expensive serialization overhead
- data lifetimes independent of process lifetimes
- example workloads:
  - worker prediction scaling: intra-node and inter-node
- ***data frame sharing:***
  - **Master:** `df.persist_shared(“myframe”)`
  - **Workers:** `df.restore_shared(“myframe”)`
- ***model sharing:***
  - **Persist:** `model.persist_shared(“myframe”)`
  - **Restore:** `model.restore_shared(“myframe”)`

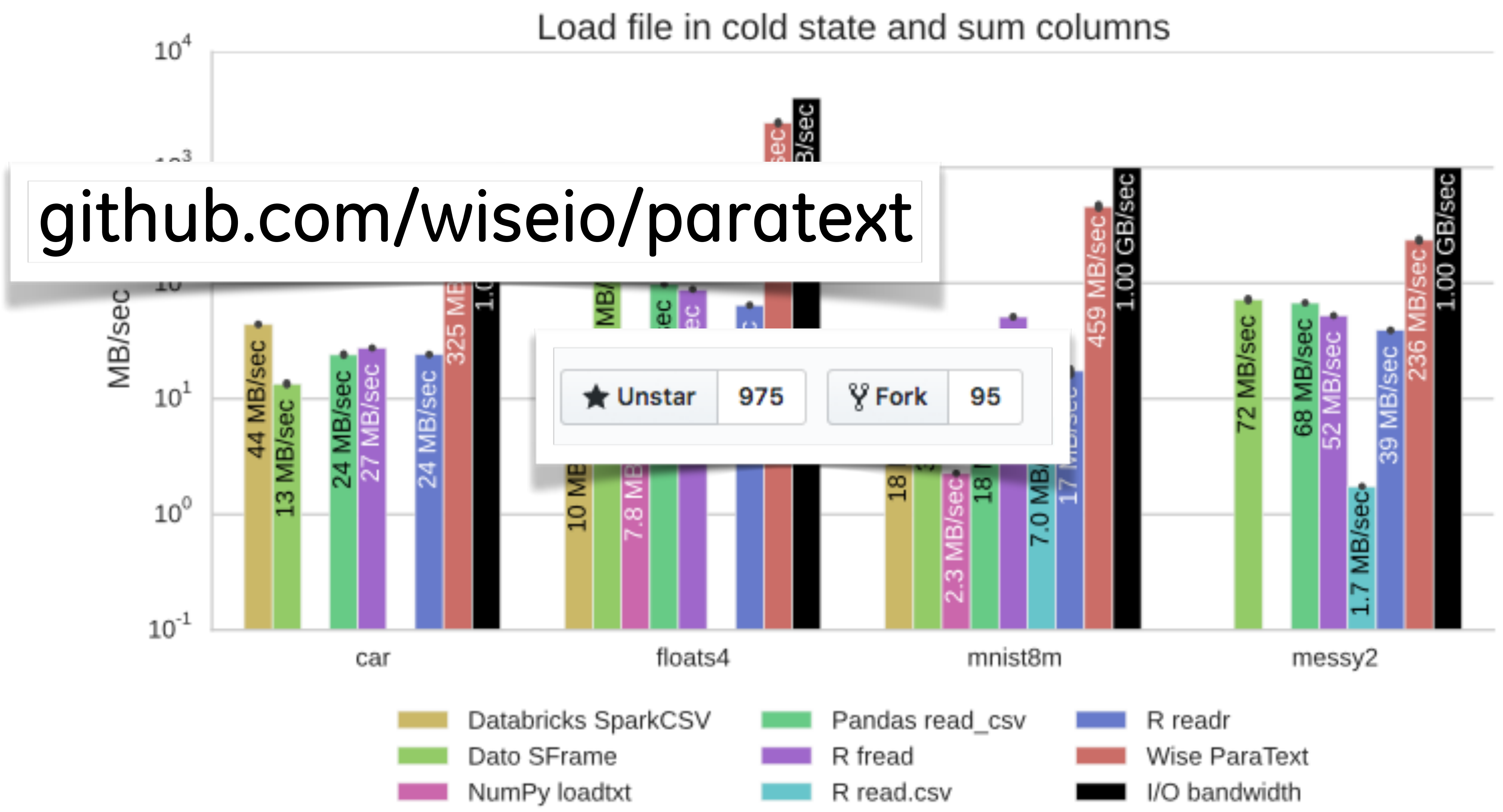
Memory Footprint



Parallel I/O: Accelerating Data Ingest

1 TB File (Small/Medium Data)

	Apache Spark	wise.io
Load Time	~11 hours, 37 mins	**~7 mins**
To Python Time	Out-of-memory	**~6 mins**
Sum Time	~11 hours, 16 mins	**~44 secs**
Cost	$156.14	**$1.57**



wise.io

Out-of-core (Medium Data)

5 TB file in ~26 mins ($2.99)

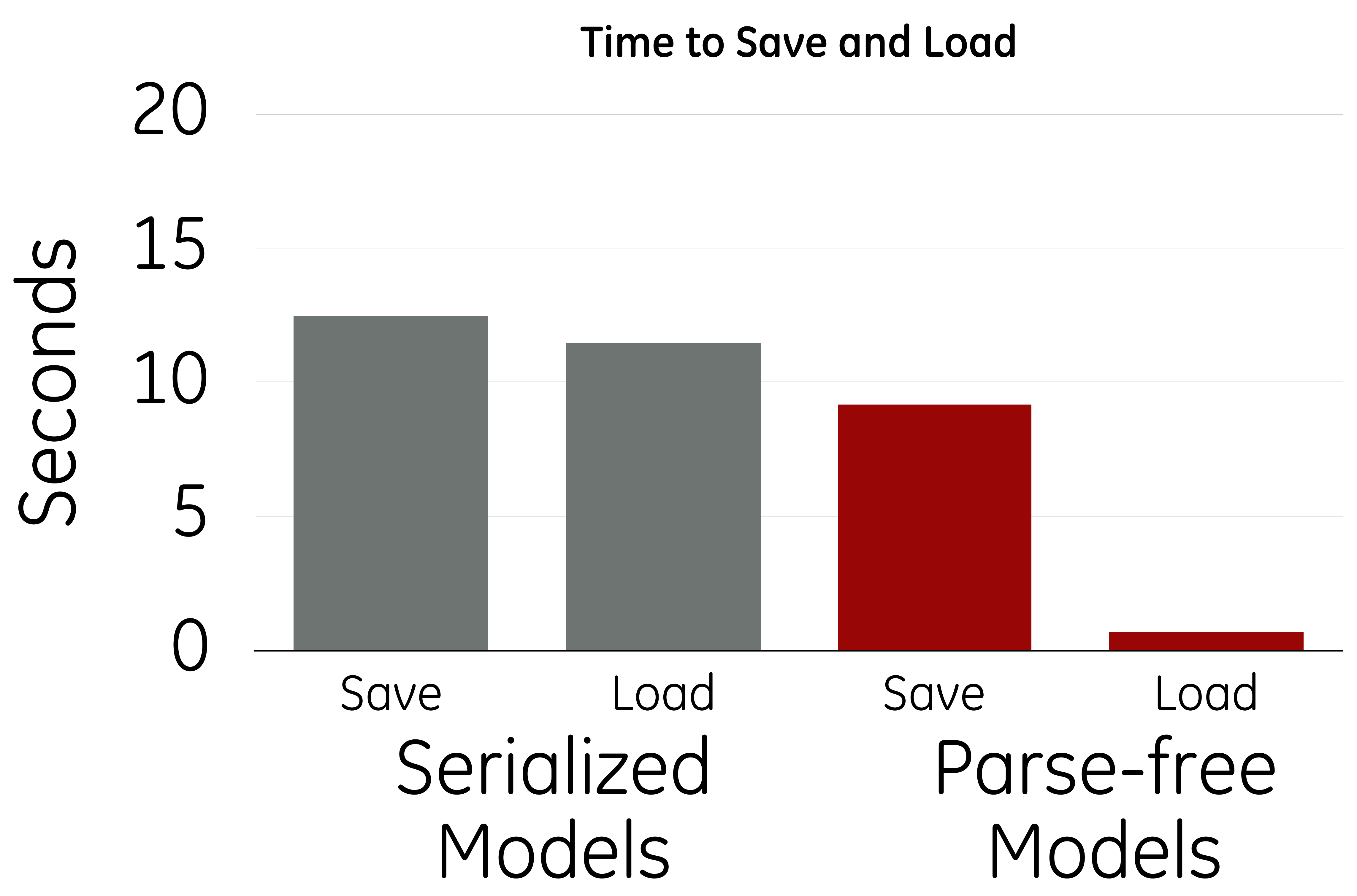
Full benchmark details: https://deads.gitbooks.io/paratext-bench/content/

Scaling Prediction with Parse-free Models

- surge in prediction requests: **deploy new prediction workers**:

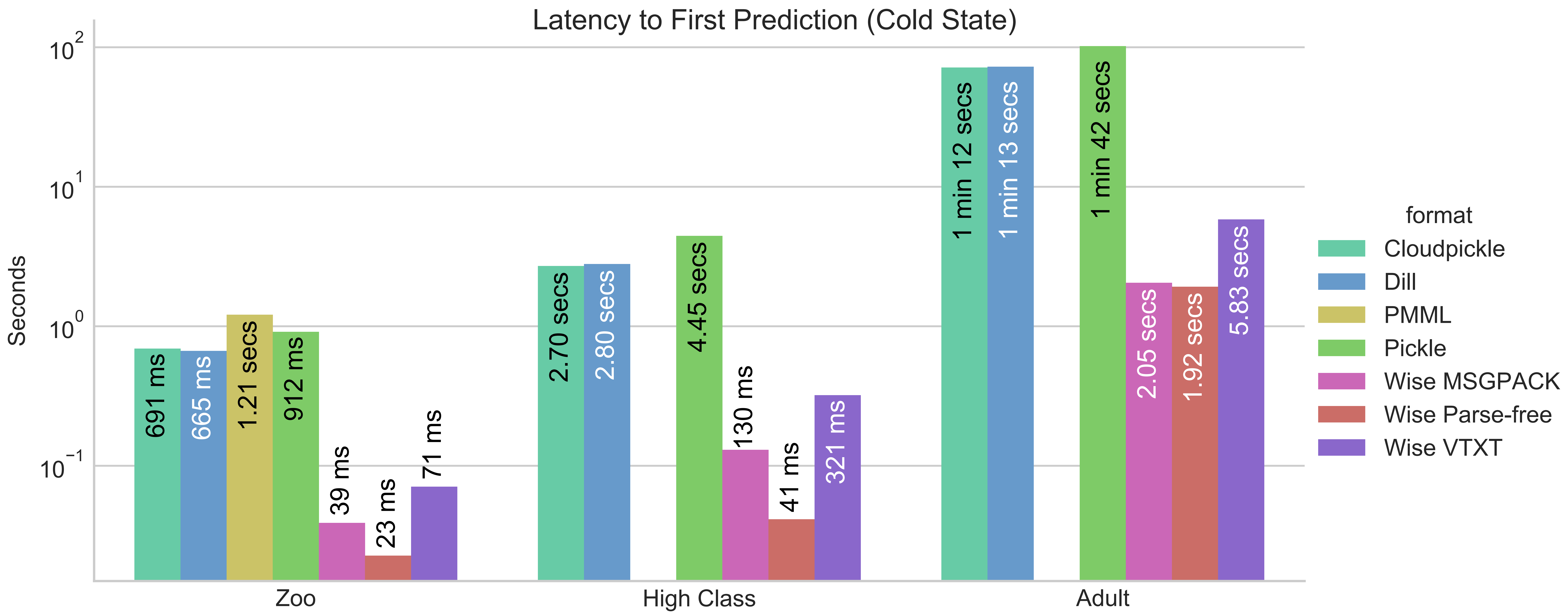
Prediction Worker | **Prediction Worker** | **Prediction Worker**

- **Problem**: Model serialization slows down model deployment
- **Solution**: Parse Free Models
- Loads at the limit of the I/O bandwidth
- Parse-free models: >20x speed-up



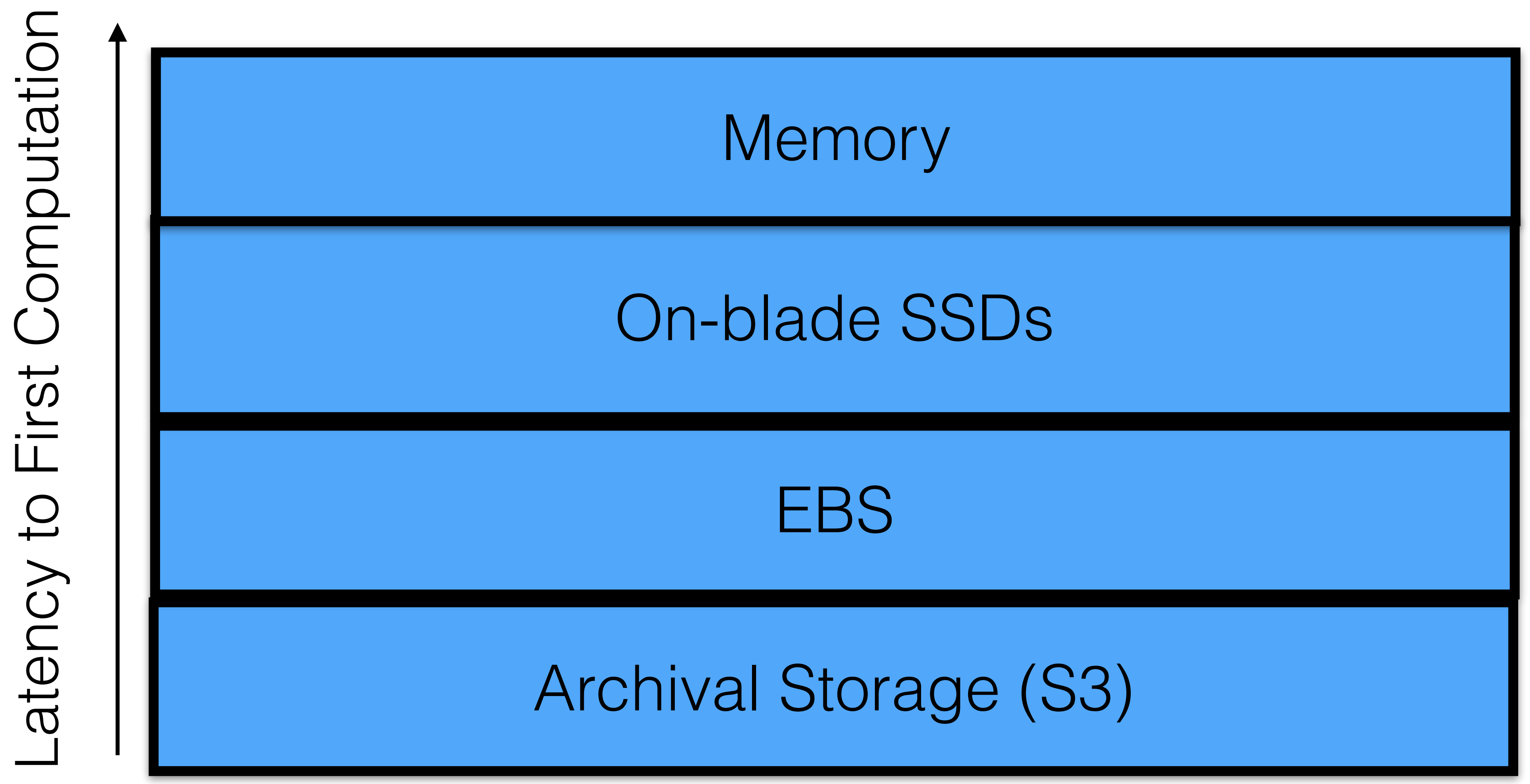
Pending US 62/526,233

Parse-Free Models

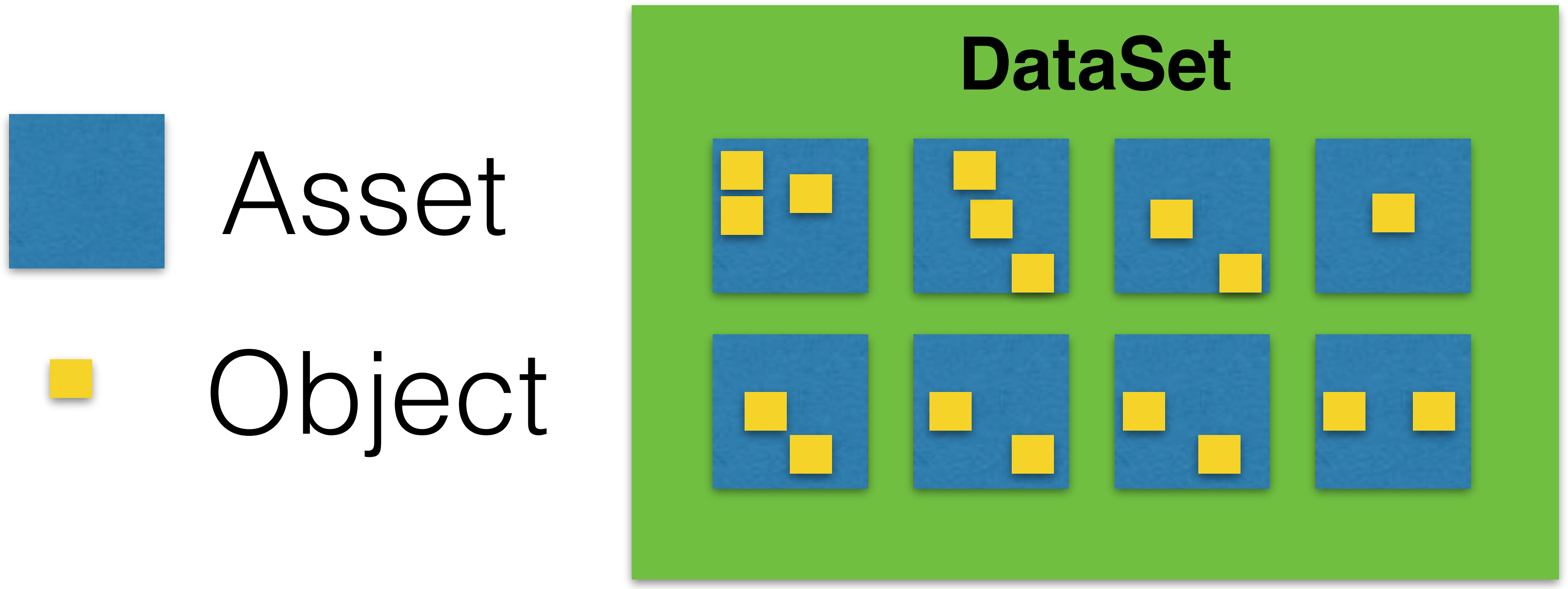


Storage and Fast Retrieval of Industrial Data for Machine Learning (DL & RF)

- Data Set: sensor data, images, multi-channel time series across multiple assets
- Objects identified by URI
- Stored in Multi-tiered Storage Hierarchy
- Lazy Loading
- Parallel Ingest
- Caching

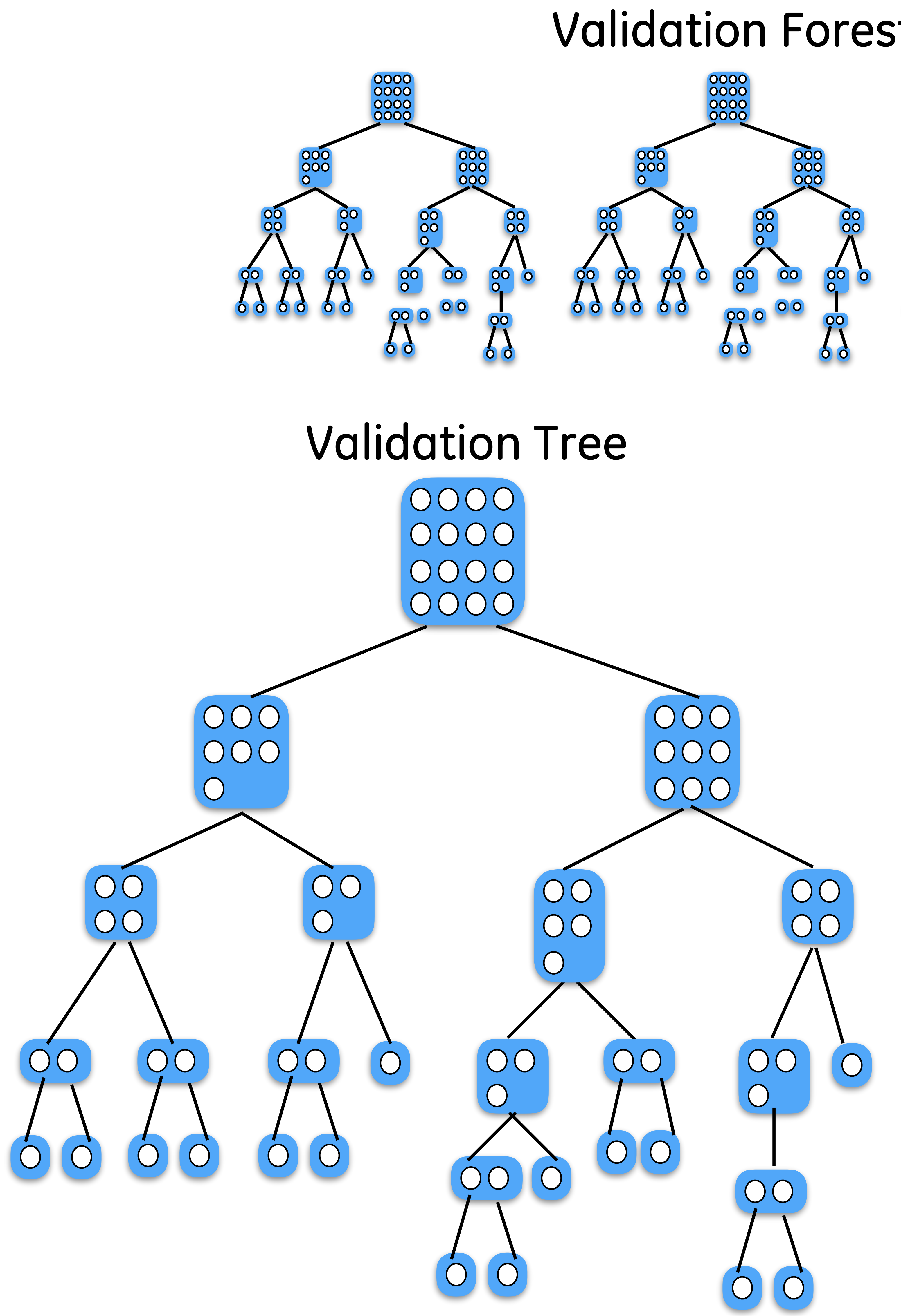



```
for batch, next_batch in batcher(dataset):
     next_batch.prefetch()
     model.update(batch, …)
     next_batch.wait()
     batch.evict()
```



Learning: Hyperparameter Tuning

- model tuning is slow
- **Validation Tree** 5D search into 1D search!
- Finds optimal
  - ***node size***
  - ***maximum depth***
  - ***forest size***
  - ***prediction method***



Validation Tree

Special tree data structure that stores info to compute AUC and Accuracy for every hyper parameter

Pending: US 62/532,221

Learning: High-class and Categorical Heavy

Data

- 1000s of classes
- 1000s of levels

Innovations

- Avoid 1-hot encodings
- Specialized learning algorithms
- Sparse algorithms and data structures
- Compact model representation

US 9,547,830

Same algorithm and languages...

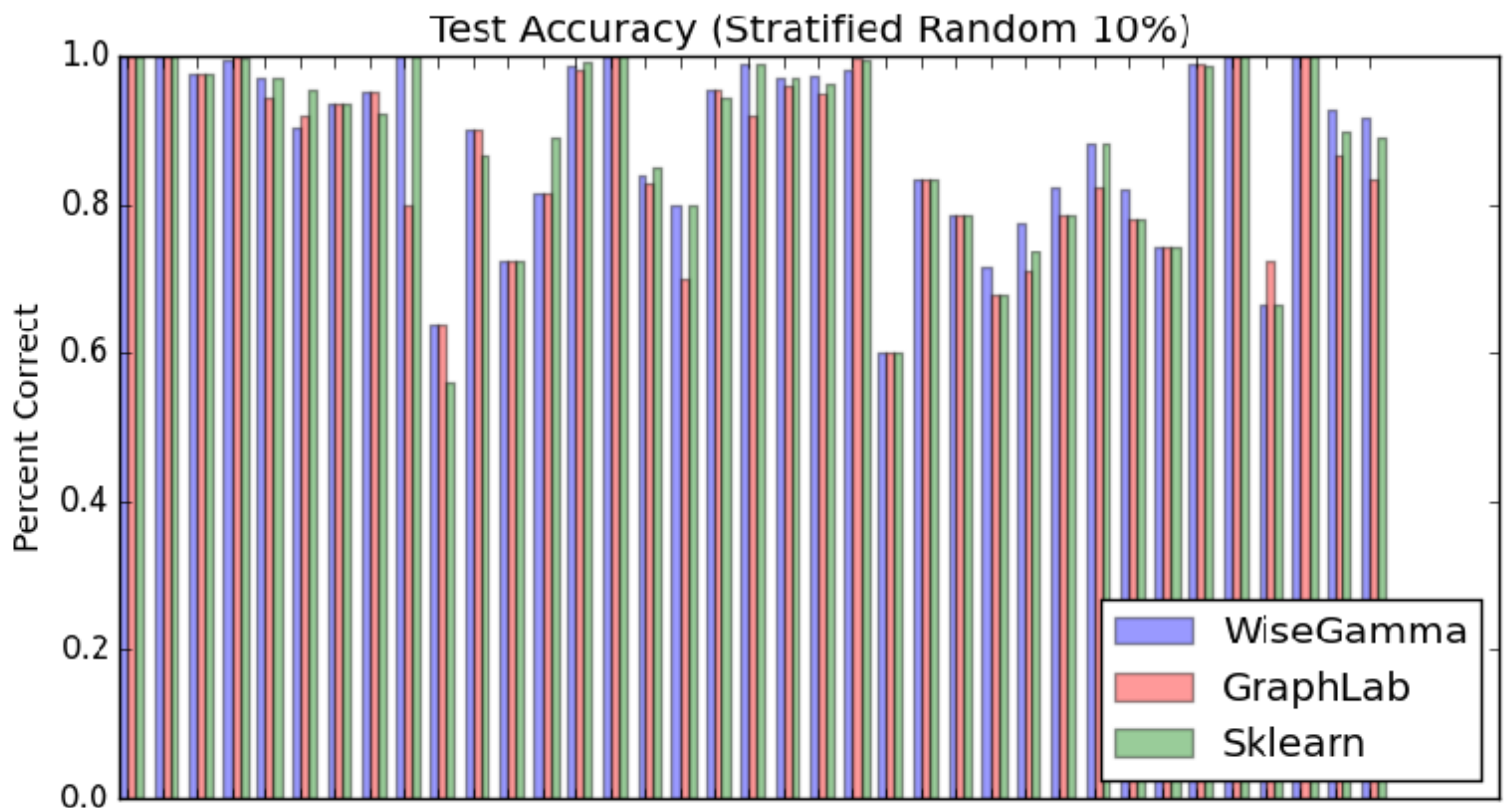
	turi	Wise.io
Test Set Accuracy	15.07%	**38.27%**
Learn Time	50,758 sec (~14 hours, 6 min)	**178.5 sec (~3.5 min)**
Predict Time	55.25 sec (54.10 per sec)	**0.16 sec (20,541 per sec)**
Peak Memory	6,156 MB	**1,231 MB**
Model File	1,178 MB	**65 MB**

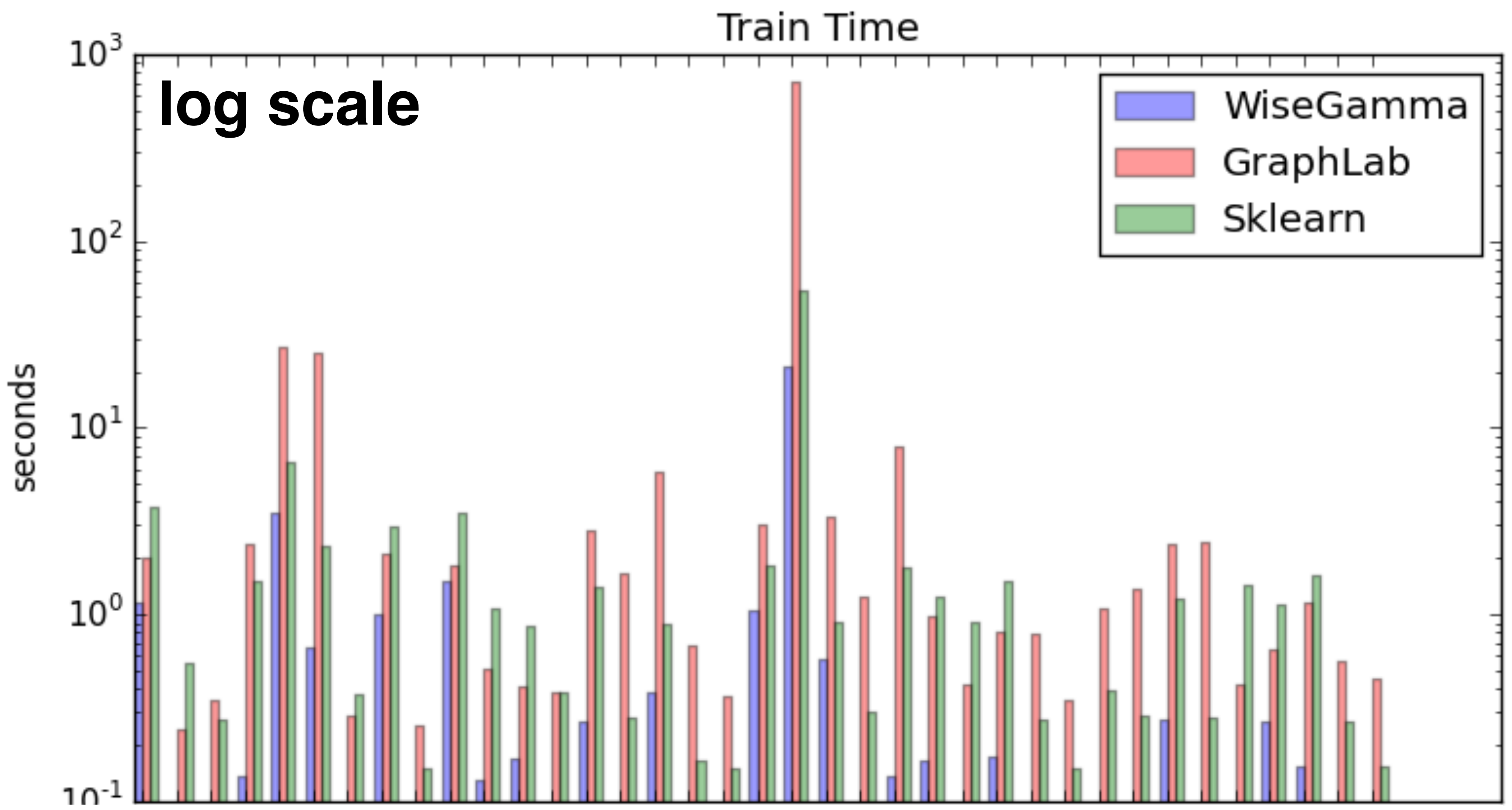
machine: 16 core, 32 GiB RAM

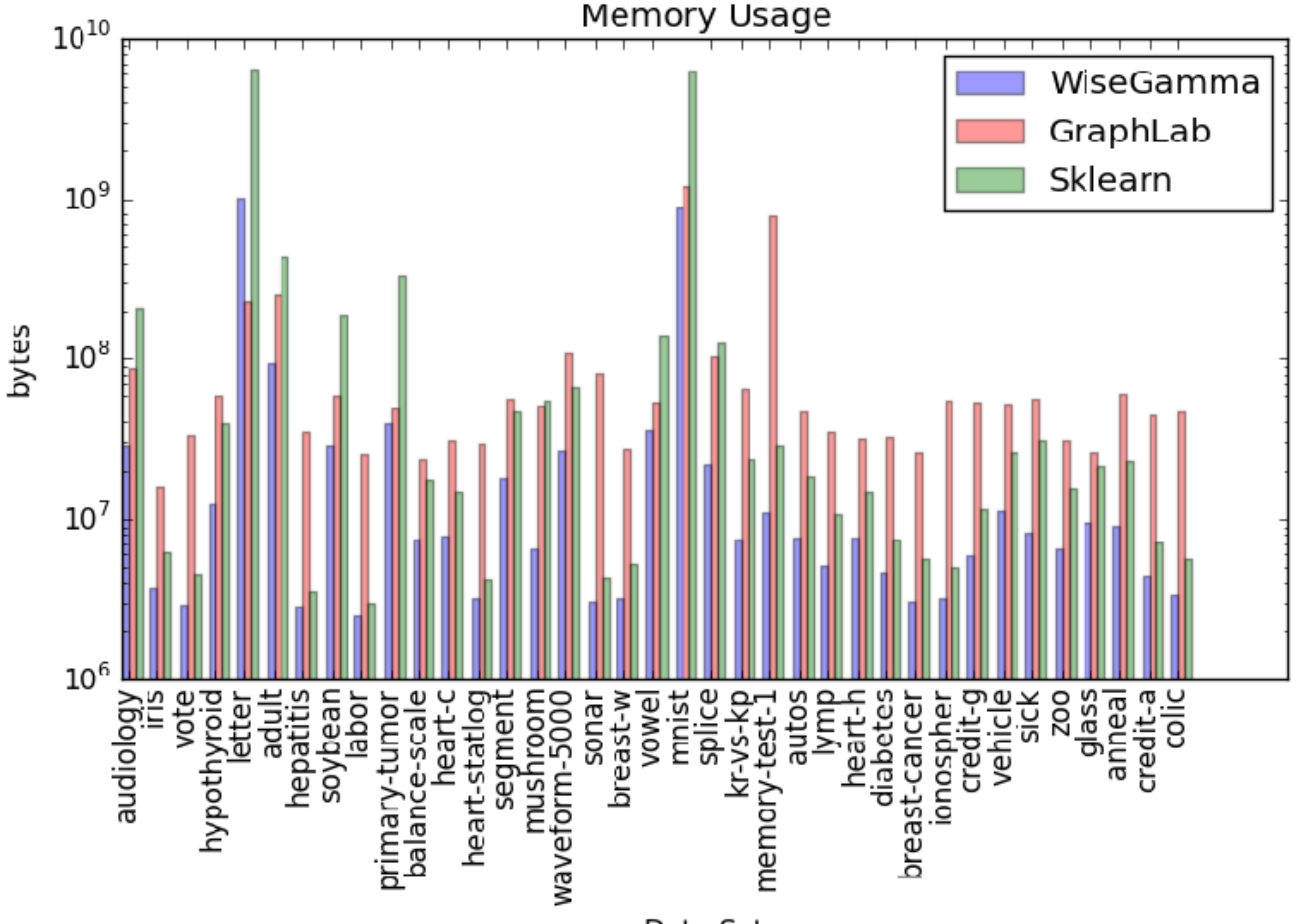
Data Set
> GB
1619 classes
3528 features
1000+ levels

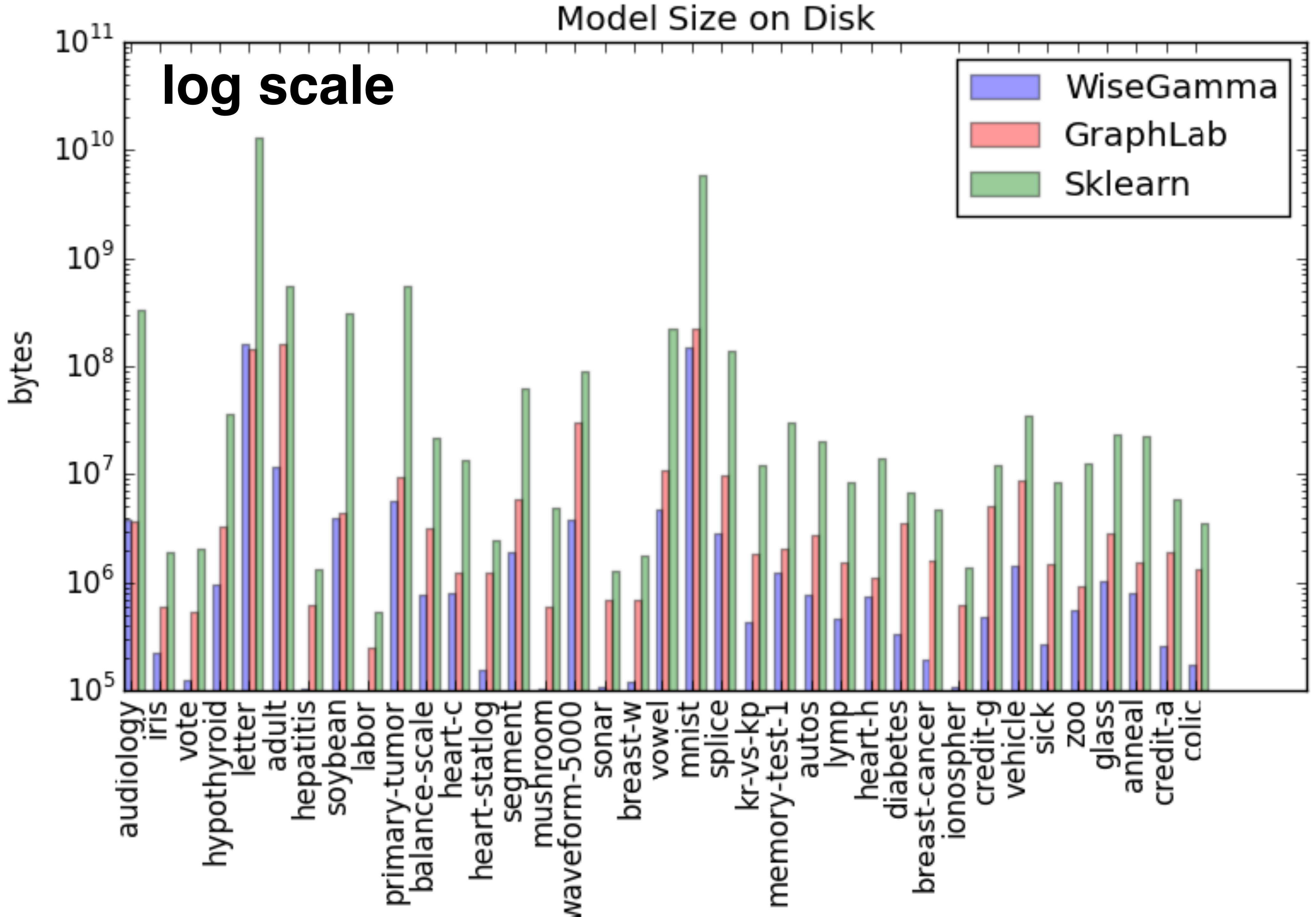
US 9,547,830

Accuracy, Timing, Resource Usage, Cost ($)









Workload: Load Data and Learn

Spark

```
sdf = sqlContext.read.format('com.databricks.spark.csv') \
        .option('header', str(bool(header)).lower()) \
        .option('inferschema', 'true') \
        .load(filename)
str_columns = [field.name for field in df.schema.fields if str(field.dataType)=="StringType"]
str_index_columns = [colname + "_index" for colname in str_columns]
for colname, index_colname in zip(str_columns, str_index_columns):
   stringIndexer = StringIndexer(inputCol=colname, outputCol=index_colname)
   model = stringIndexer.fit(sdf)
   indexed = model.transform(sdf)
   encoder = OneHotEncoder(includeFirst=False, inputCol=colname, outputCol=index_colname)
   encoded = encoder.transform(indexed)
   sdf = encoded
rf = RandomForestClassifier()
pipeline = Pipeline().setStages([sdf, rf])
pipeline.fit(sdf)
```

WiseML

```
df = DataSet.load(filename)
model = Model.learn(df)
```

Summary

Data

- Heterogeneous for Industrial Data (Time Series, Structured Meta-Data, Sensor Data, Images, etc.)
- Out-of-core
- Copy-on-write
- Off-heap
- Distributed
- Well-typed

I/O

- Exploit Storage Hierarchy
- Multi-tiered Cache
- Lazy Loading
- Parallel Data Ingest
- Parse-free Models
- Flat Data Structures
- Compact, Sparse Models

Learning

- Memory-efficient
- Categorical
- High Class
- Fast Parameter Tuning via Validation Trees
- Cache-exploitative