Update on Apache Spark

Ion Stoica
UC Berkeley

http://www.istc-cc.cmu.edu/

Intel Science & Technology
Center for Cloud Computing

Apache Spark

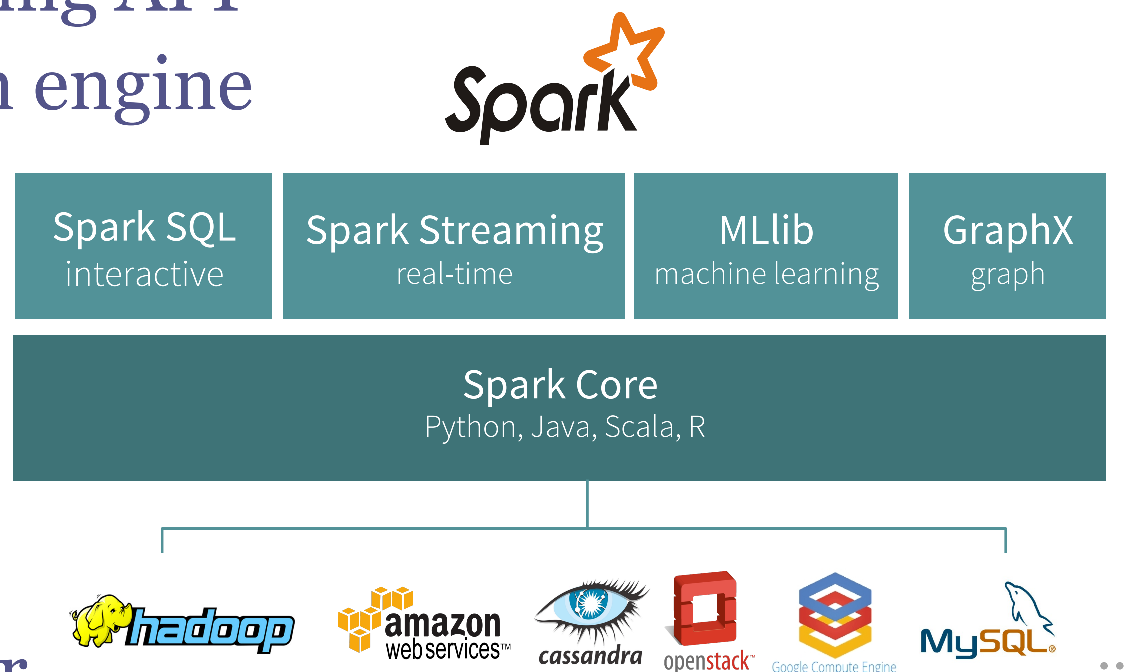
2009: Started at AMPLab, UC Berkeley by Databricks co-founders

- Fault tolerance, in-memory storage
- Powerful programming API
- Unified computation engine

2010: Open Source

2013: Apache Project

Today: widely popular



Adoption

1000+ companies

Goldman Sachs, TOYOTA, Capital One, NOVARTIS
verizon, THOMSON REUTERS, salesforce, Adobe
ebay, UBER, AUTODESK
airbnb, Telefónica, NBCUniversal
Palantir, Microsoft, NETFLIX

. . .

50+ companies

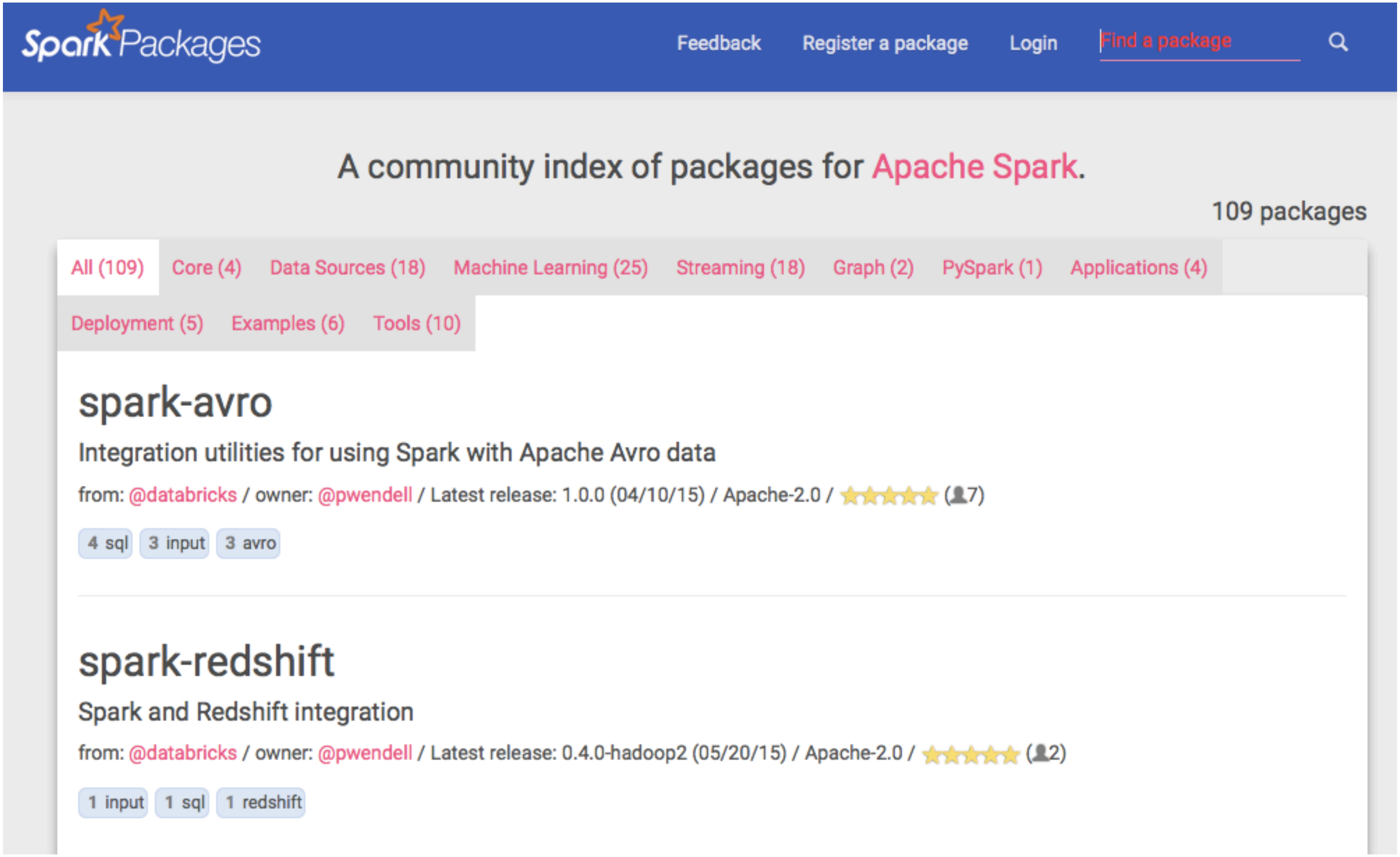
databricks, cloudera, intel
Hortonworks, MAPR, IBM
ORACLE, SAP, Qlik
DATASTAX, TIBCO, amazon web services
tableau SOFTWARE, alteryx, bluedata
informatica, MicroStrategy

. . .

Growth is Accelerating

- Number of total contributors increased 3x since last year
  - Most active Apache project since 2014
  - 255 (June 2014) → 730 (June 2015)
- Number of meetup members increased 2x since beginning of this year
  - 12K (Jan 2015) → 26K (July 2015)
- Number of Spark Summit attendees increased 2x last year; 3x this year
  - 450 (2013) → 1,100 (2014) → 3,500 (2015; New York, San Francisco, Amsterdam)

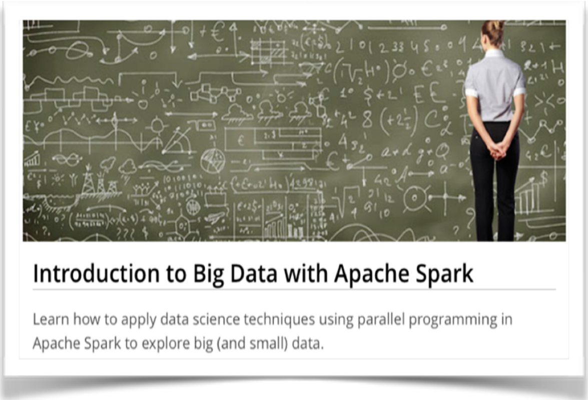
Spark Packages



Spark based MOOCs (EdX)

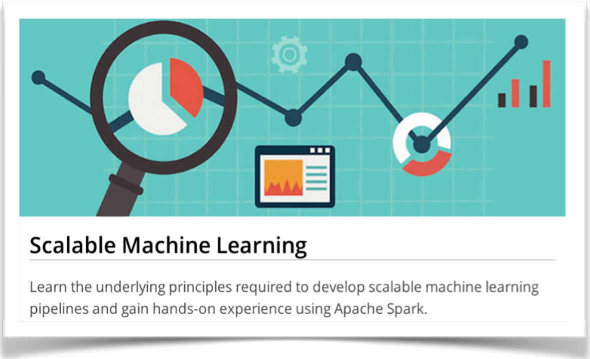
"Intro to Big Data with Apache Spark"

- Anthony Joseph, UC Berkeley
- June 1st, 5 weeks
- 78,000+ registrations, 12% completion



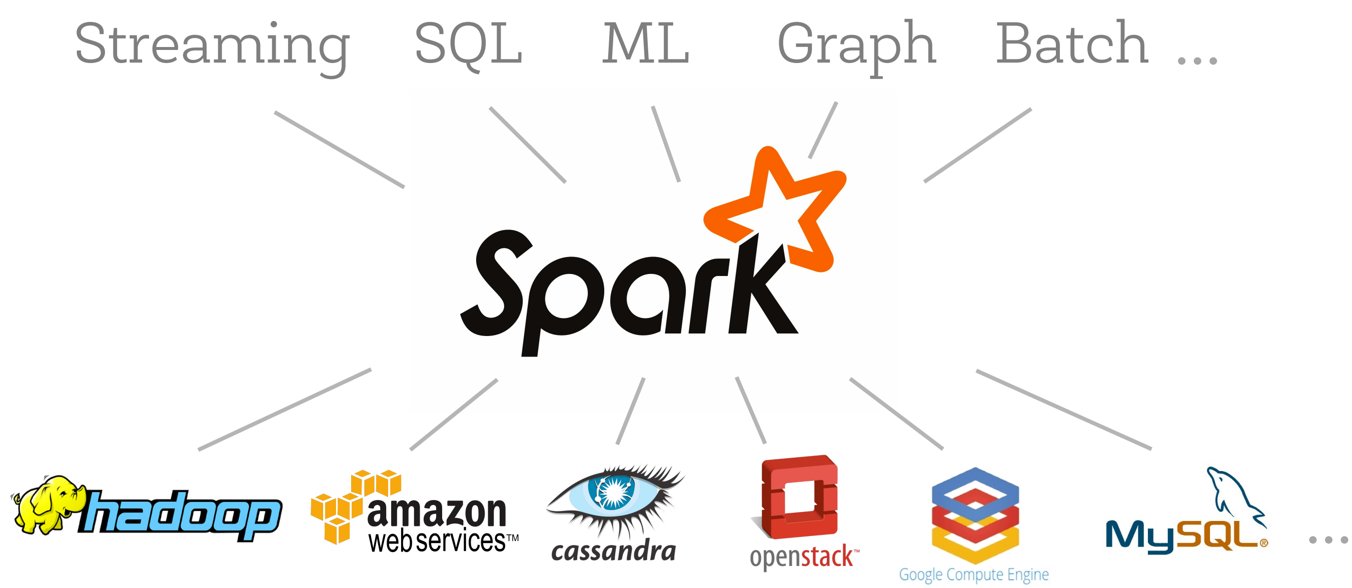
"Scalable Machine Learning with Spark"

- Ameet Talwalkar, UCLA
- June 22nd, 5 weeks
- 50,000+ registrations, 14% completion



Our Goal for Spark

Unified engine across data workloads and platforms



Past 2 Years

Fast growth in libraries and integration points

- 10x growth of ML library
- Pluggable data source API
- R language

Result: very diverse use of Spark

- Only 40% of users on Hadoop YARN
- Most users use at least 2 of Spark's built-in libraries
- 98% of Databricks customers use SQL, 60% use Python

New Directions

Dataframes API: raising the level of abstraction

Project Tungsten: bringing Sparks' performance closer to the bare metal

Some of the largest changes to Spark (Core) since it has been created

Google Trends for "dataframe"

Single-node tabular data structure, with API for

- relational algebra (filter, join, …)
- math and stats
- input/output (CSV, JSON, …)
- …

Data frame: lingua franca for “small data”

```
head(flights)
#> Source: local data frame [6 x 16]
#>
#>    year month day dep_time dep_delay arr_time arr_delay carrier tailnum
#> 1  2013     1   1      517         2      830        11      UA  N14228
#> 2  2013     1   1      533         4      850        20      UA  N24211
#> 3  2013     1   1      542         2      923        33      AA  N619AA
#> 4  2013     1   1      544        -1     1004       -18      B6  N804JB
#> ..  ...   ... ...      ...       ...      ...       ...     ...     ...
```

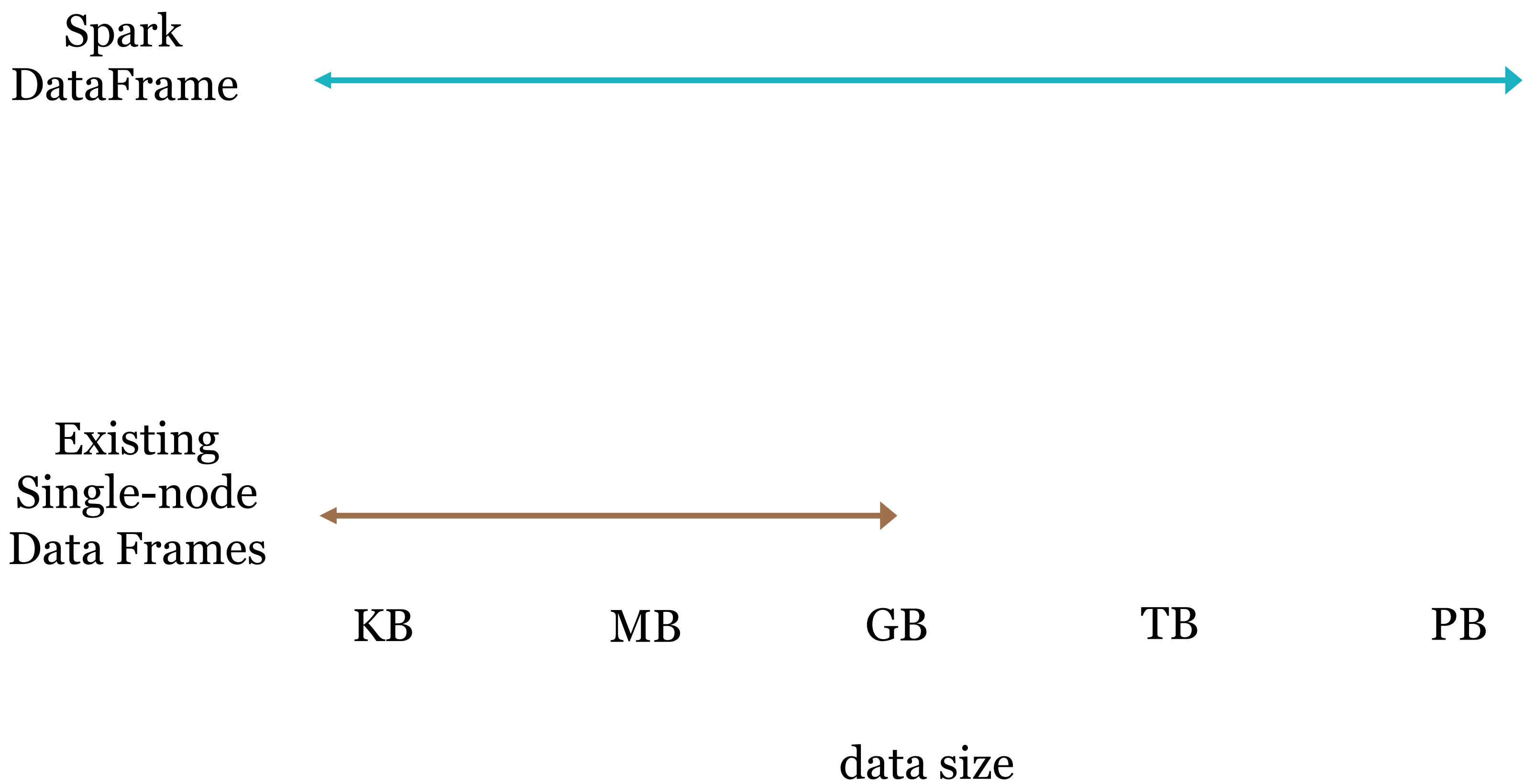
Spark DataFrame

Distributed data frame for Java, Python, R, Scala

Similar APIs as single-node tools (Pandas), i.e. easy to learn

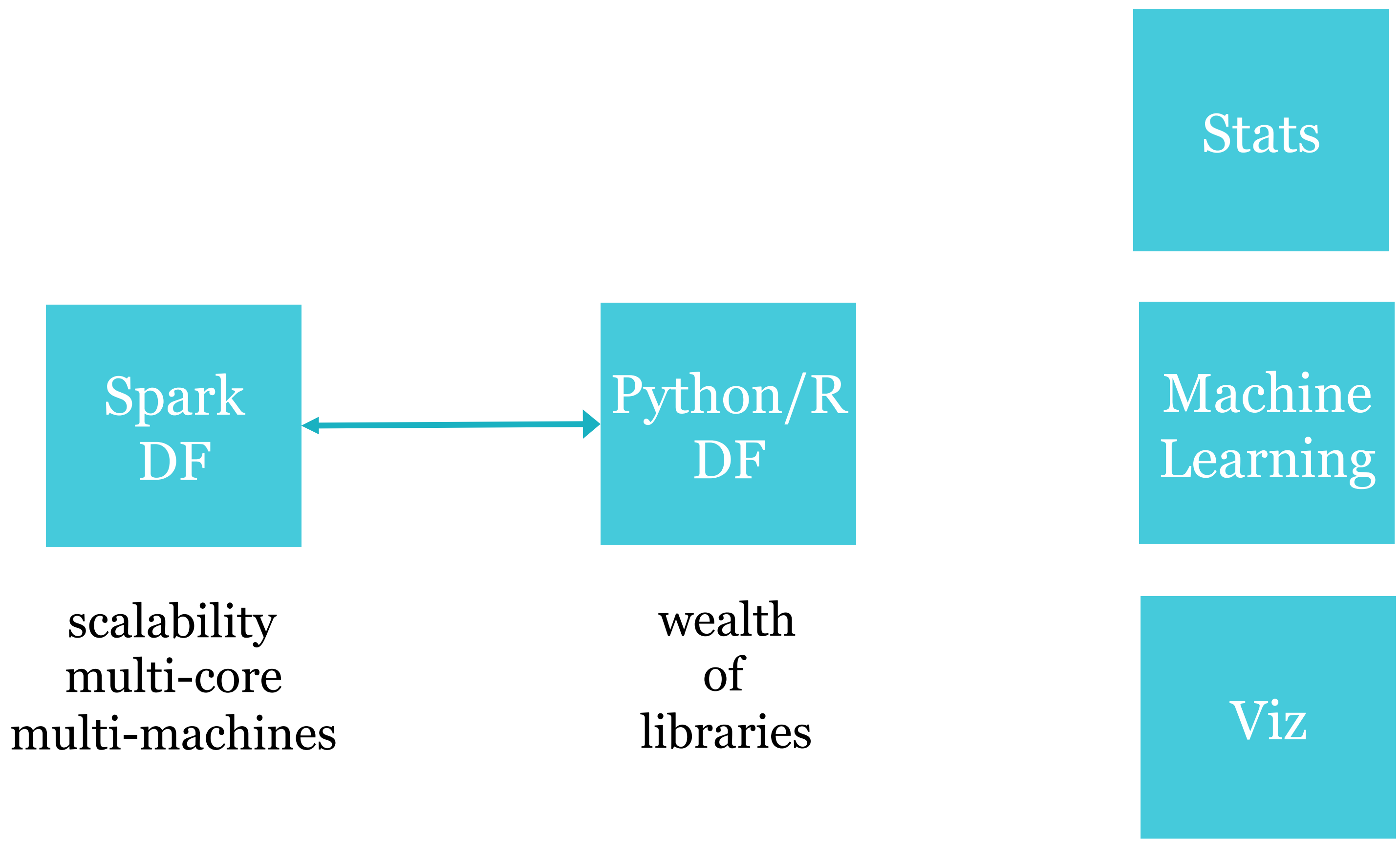
```
> head(filter(df, df$waiting < 50))  # an example in R
##  eruptions waiting
##1     1.750      47
##2     1.750      47
##3     1.867      48
```

Scaling

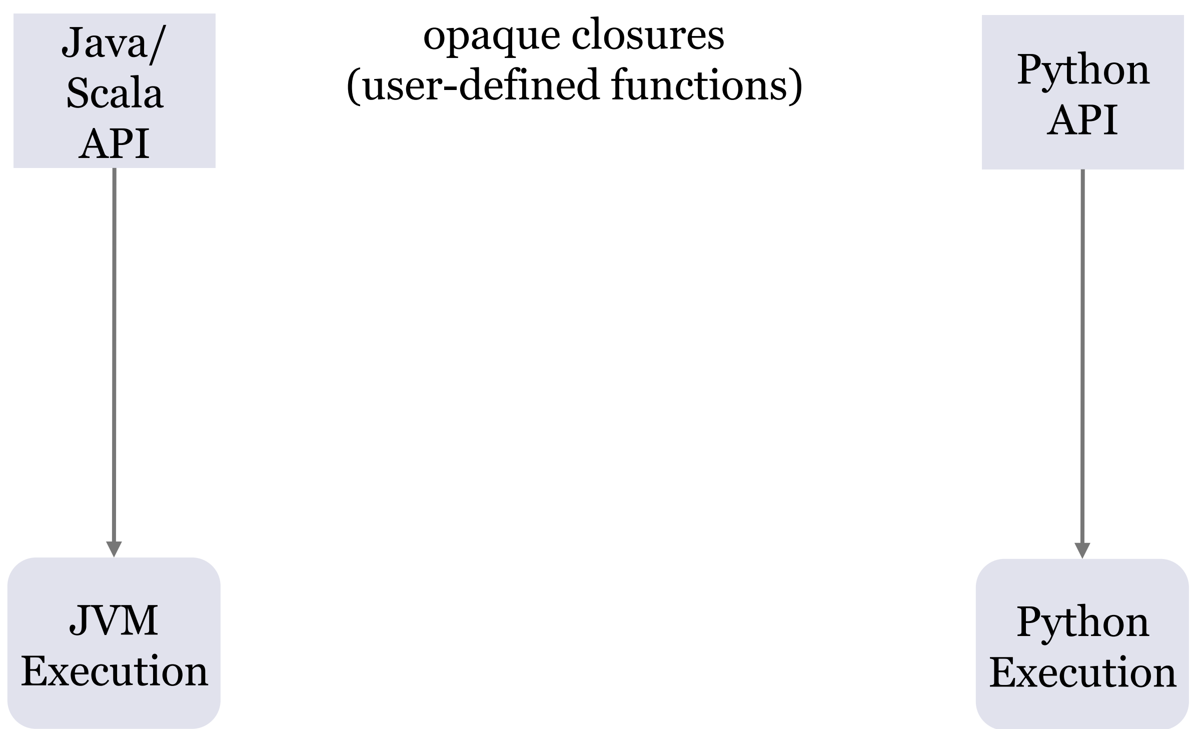


Spark DataFrame
Existing Single-node Data Frames
KB
MB
GB
TB
PB
data size

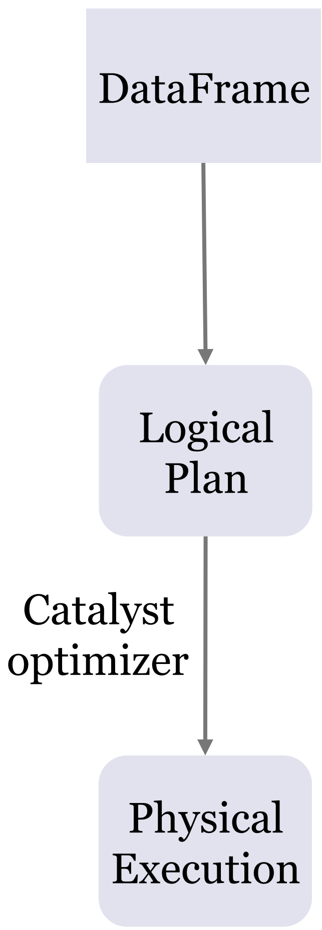
Spark and Python/R



Spark RDD Execution

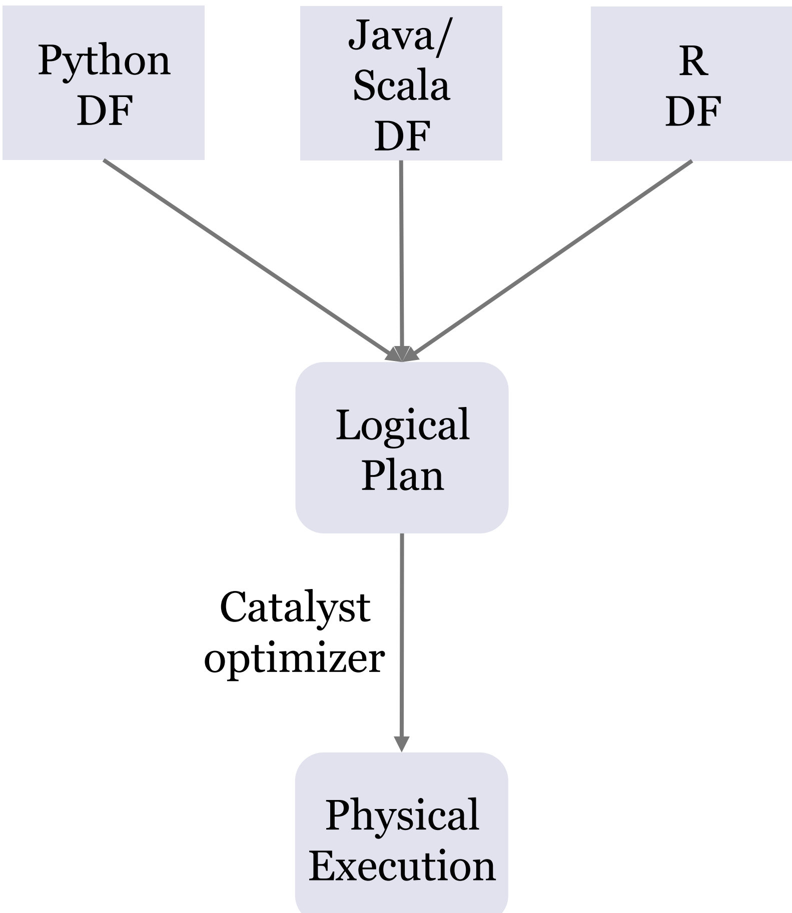


Spark DataFrame Execution



Intermediate representation for computation

Spark DataFrame Execution



Simple wrappers to create logical plan

Intermediate representation for computation

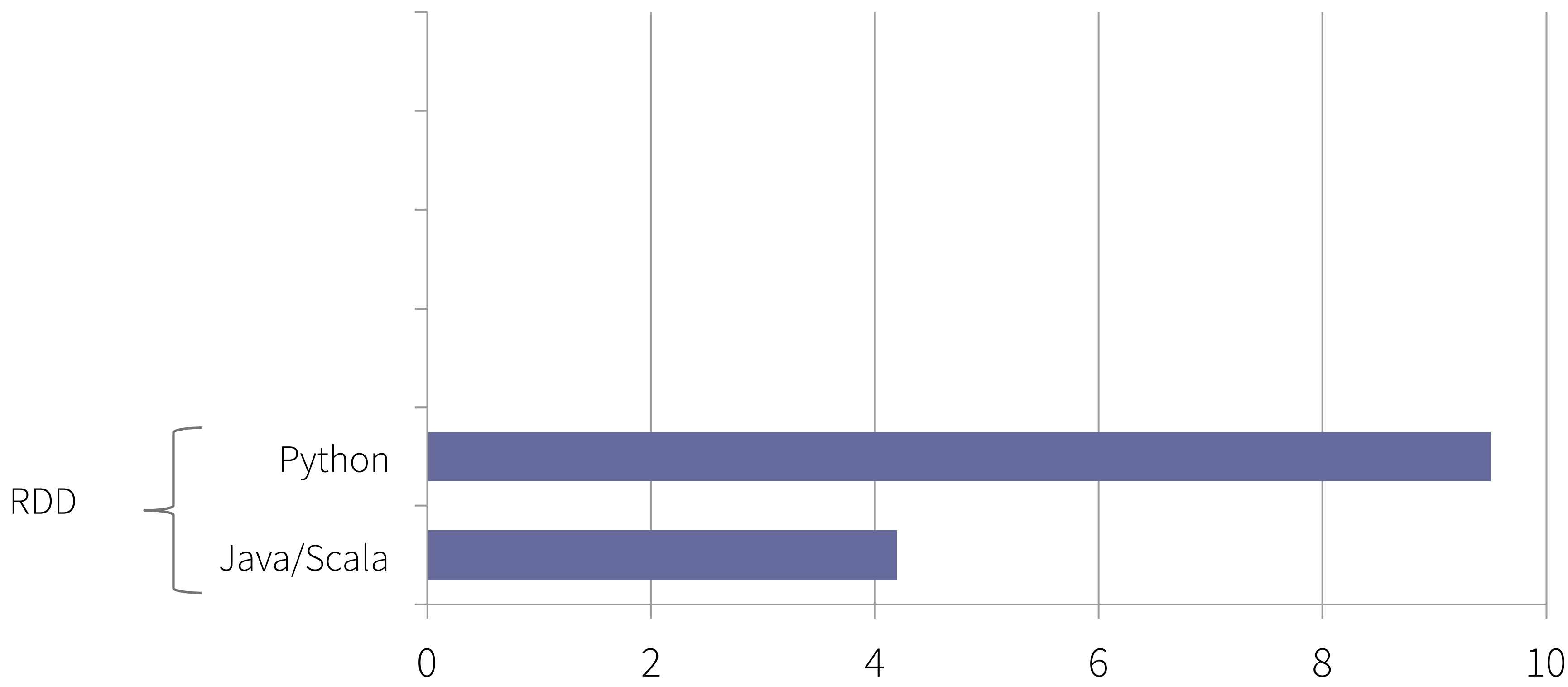
Benefit of Logical Plan: Simpler Frontend

Python : ~2000 line of code (built over a weekend)

R : ~1000 line of code

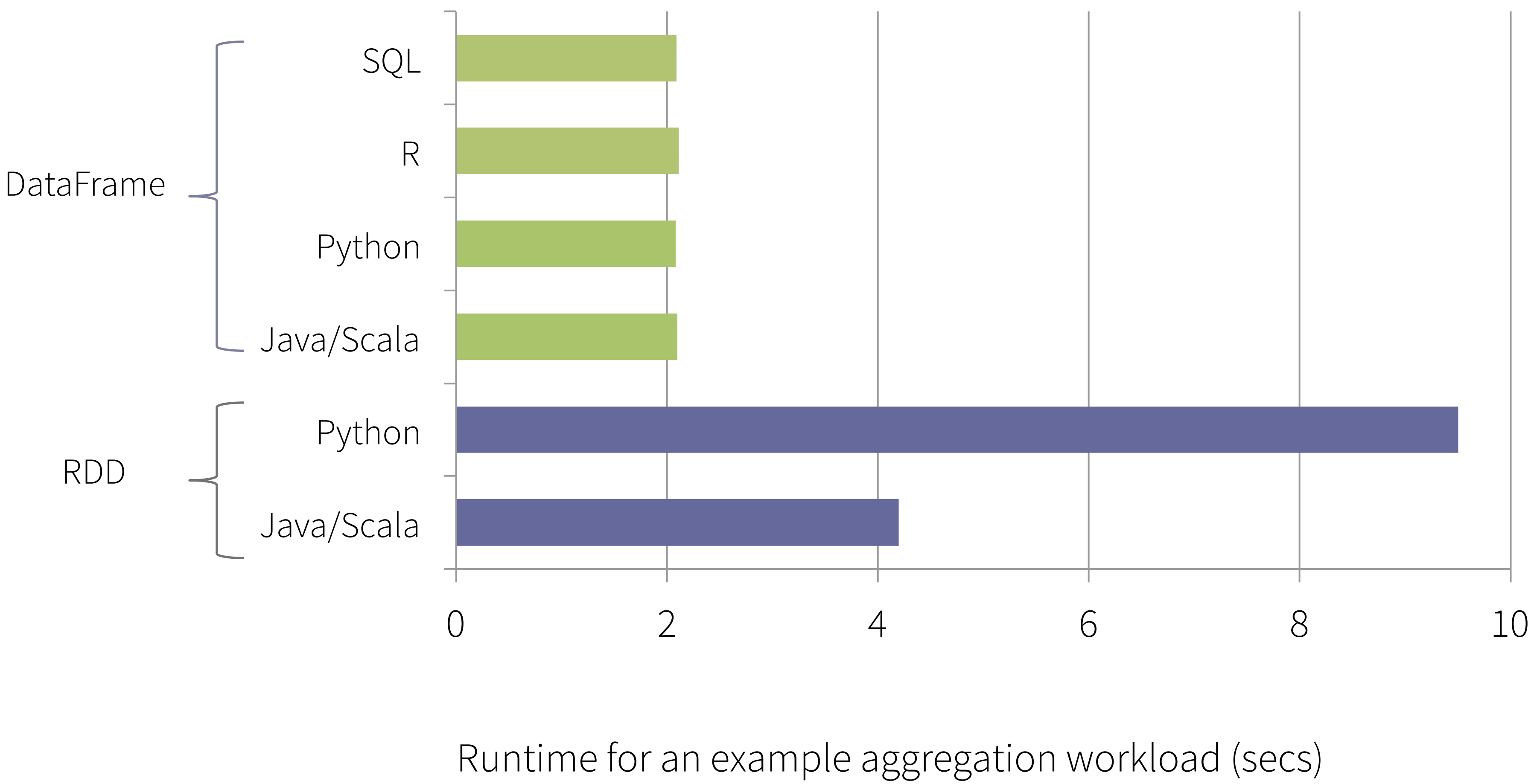
Much easier to add new language bindings (i.e., Julia, Clojure, …)

Performance



Runtime for an example aggregation workload

Benefits: Performance Parity Across Languages



Tungsten

Hardware Trends

Storage

Network

CPU

Hardware Trends

	2010
Storage	50+MB/s (HDD)
Network	1Gbps
CPU	~3GHz

Hardware Trends

	2010	2015
Storage	50+MB/s (HDD)	500+MB/s (SSD)
Network	1Gbps	10Gbps
CPU	~3GHz	~3GHz

Hardware Trends

	2010	2015	
Storage	50+MB/s (HDD)	500+MB/s (SSD)	10x
Network	1Gbps	10Gbps	10x
CPU	~3GHz	~3GHz	☹

Tungsten: Preparing Spark for Next 5 Years

Substantially speed up execution by optimizing CPU efficiency, via:

(1) Off-heap memory management
(2) Runtime code generation
(3) Exploiting cache locality

1. Off-Heap Memory Management

Store data off-heap to avoid object overhead & GC

- For RDDs: fast serialization libraries
- For DataFrames & SQL: binary format we compute on directly

2-10x space saving, especially for strings, nested objects

Can use new RAM-like devices, e.g. flash, 3D XPoint

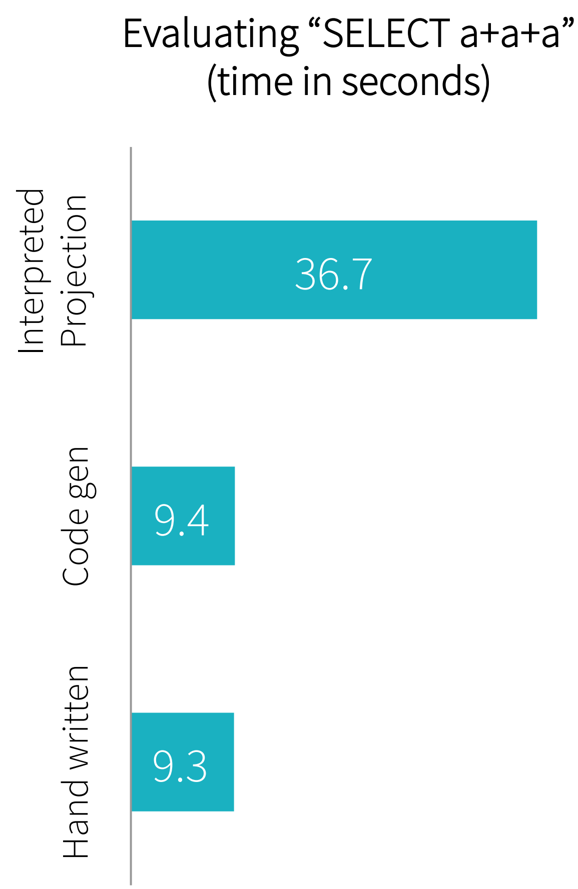
2. Runtime Code Generation

Generate Java code for DataFrame and SQL expressions requested by user

Avoids virtual calls and generics/boxing

Can do same in core, ML and graph

- Code-gen serializers, fused functions, math expressions

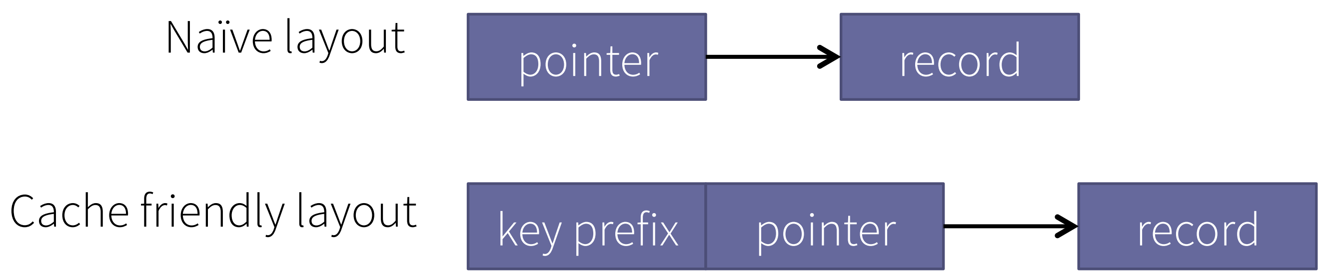


3. Cache-Aware Algorithms

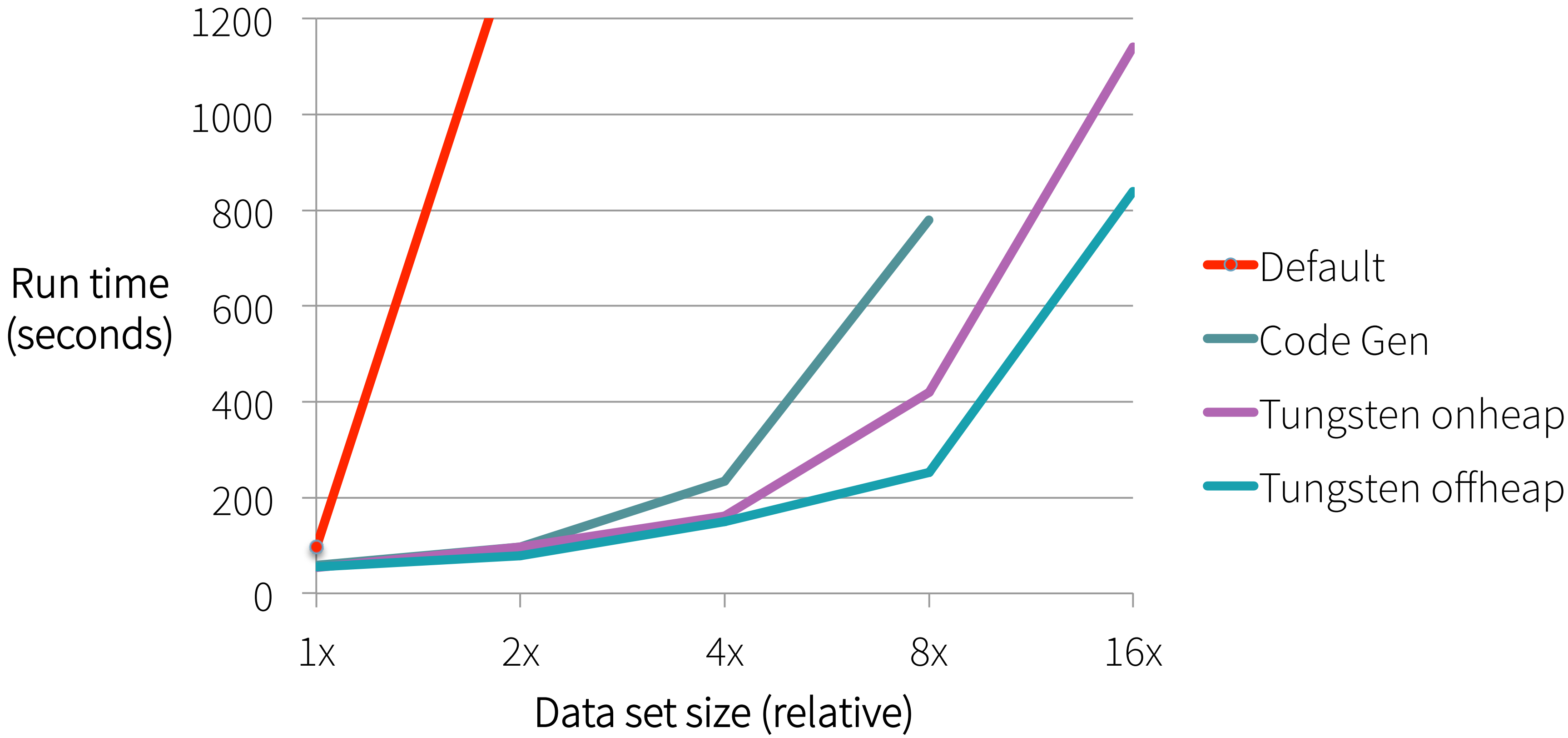
Use custom memory layout to better leverage CPU cache

Example: AlphaSort-style prefix sort

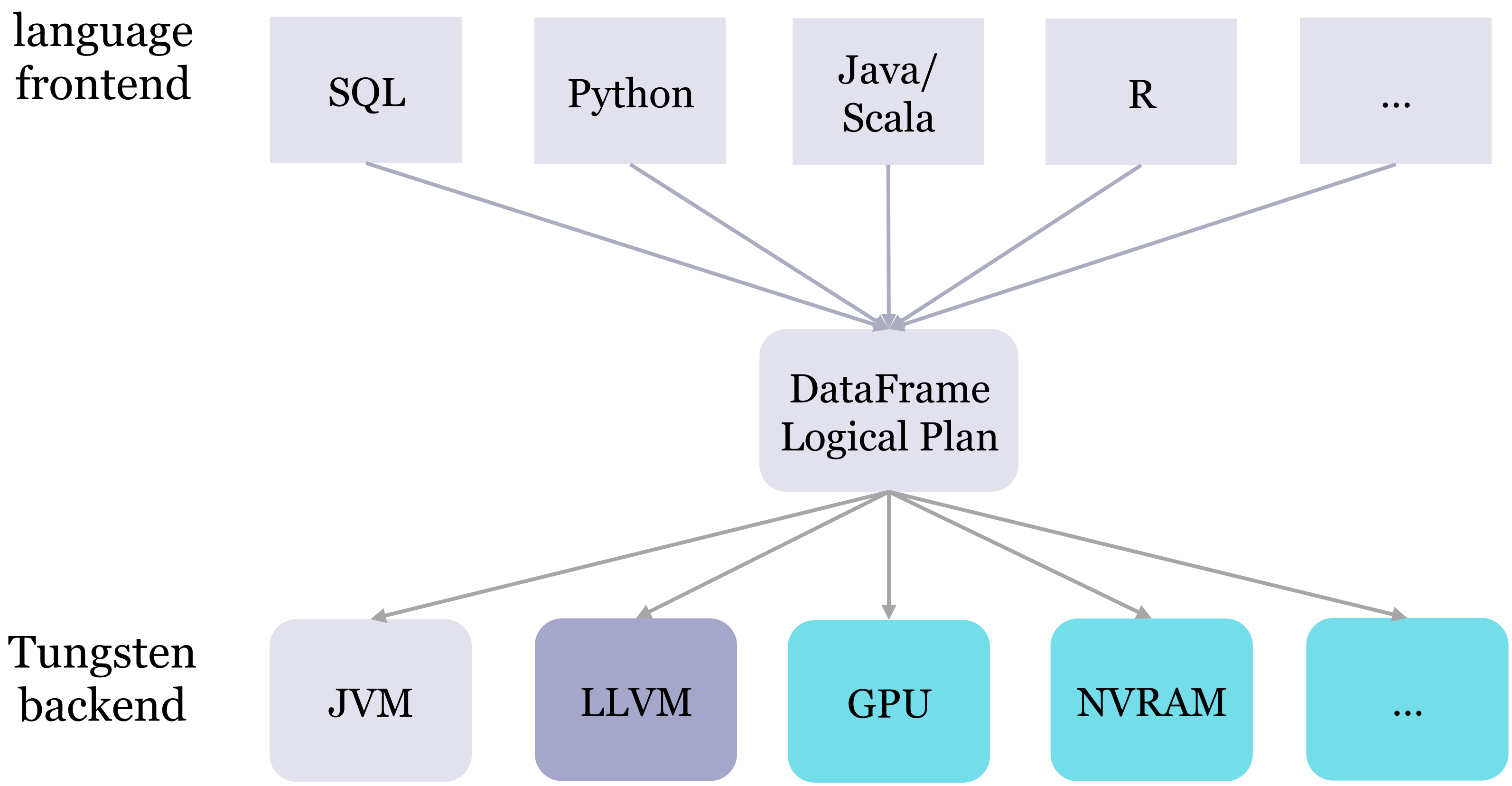
- Store prefixes of sort key inside pointer array
- Compare prefixes to avoid full record fetches + comparisons



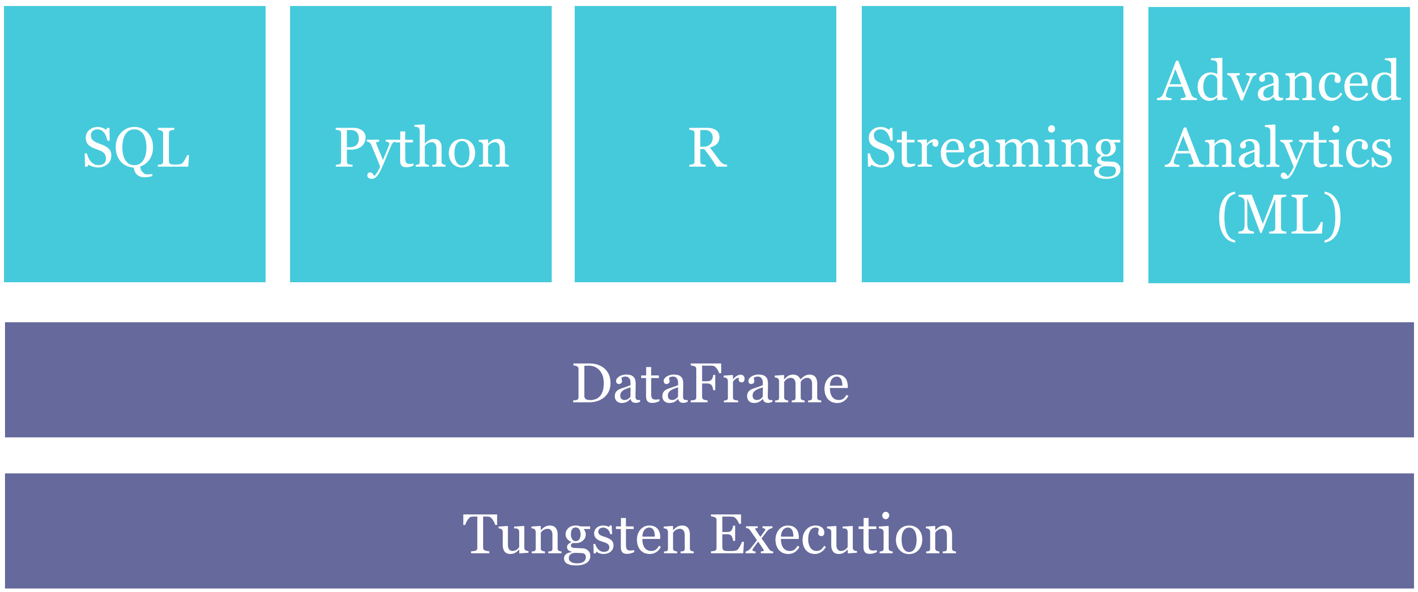
Tungsten Performance Results



Unified API, One Engine, Automatically Optimized



Spark New Architecture



Summary

Growing community

Some of the most exciting times ahead!

- Dataframes
- Project Tunsten: Memory and CPU efficiency
- Others
  - Network and disk I/O optimizations
  - Adaptive query execution

Intel a great collaborator along the years

- Major contributions to Streaming, ML