SPARK SUMMIT 2017

Real-time Machine Learning with Redis-ML and Apache Spark

Shay Nativ - Redis Labs

Agenda

- Intro to Redis and Redis Labs - 5 min
- Using Redis-ML for Model Serving - why and how - 10 min
- Building a recommendation system using Spark-ML and Redis-ML - 10 min
- QA

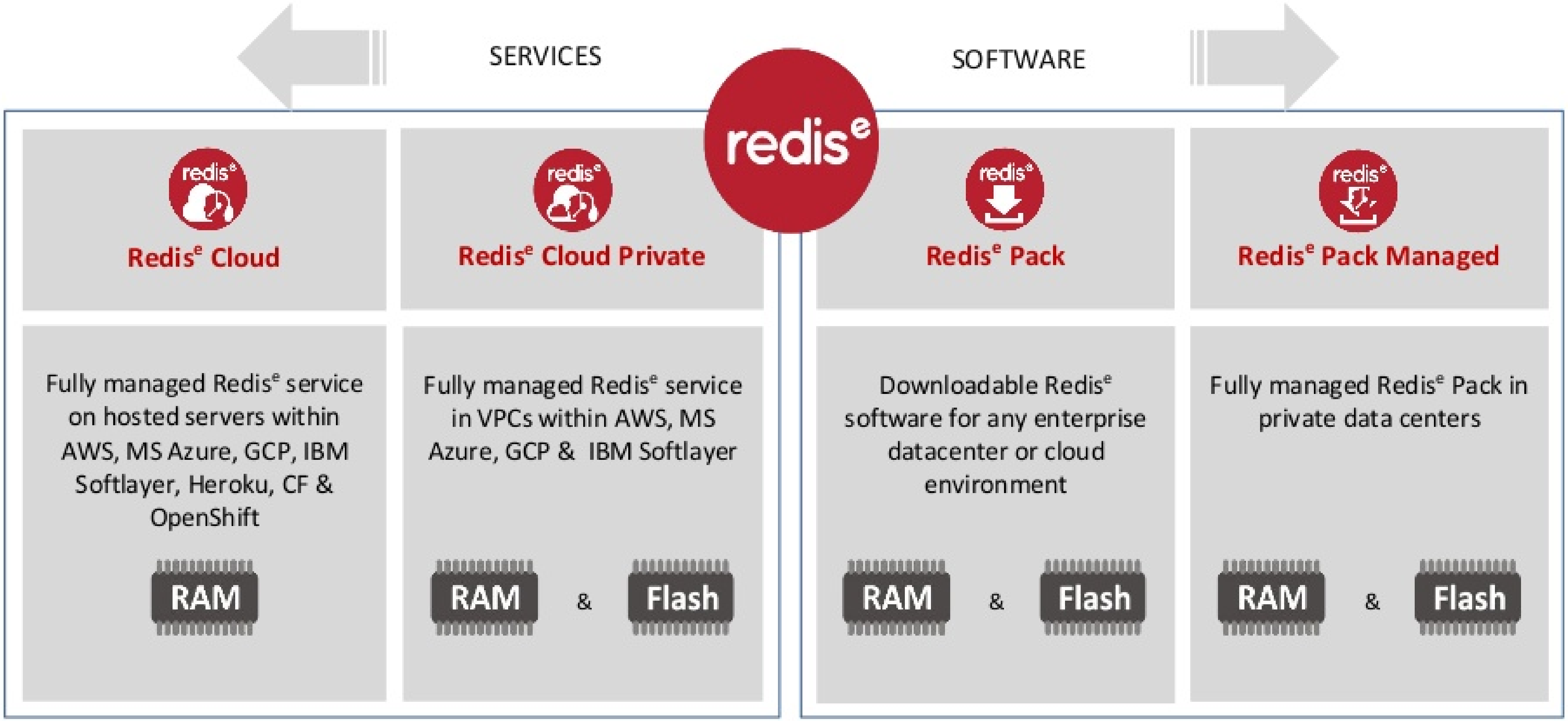
Redis Labs – Home of Redis

The commercial company behind Open Source Redis

Provider of the **Redis Enterprise (Redis^{e})** technology, platform and products

Founded in 2011
HQ in Mountain View CA, R&D center in Tel-Aviv IL

Redis Labs Products



SERVICES

SOFTWARE

redis^e

Redis^e Cloud

Fully managed Redis^e service on hosted servers within AWS, MS Azure, GCP, IBM Softlayer, Heroku, CF & OpenShift

RAM

Redis^e Cloud Private

Fully managed Redis^e service in VPCs within AWS, MS Azure, GCP & IBM Softlayer

RAM & Flash

Redis^e Pack

Downloadable Redis^e software for any enterprise datacenter or cloud environment

RAM & Flash

Redis^e Pack Managed

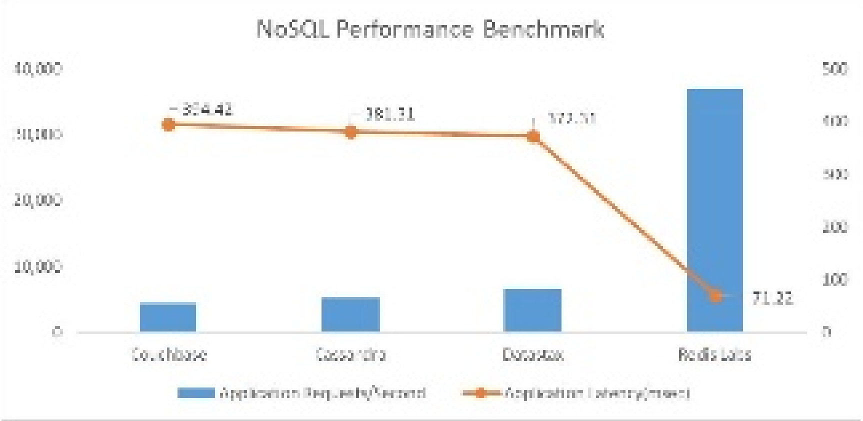
Fully managed Redis^e Pack in private data centers

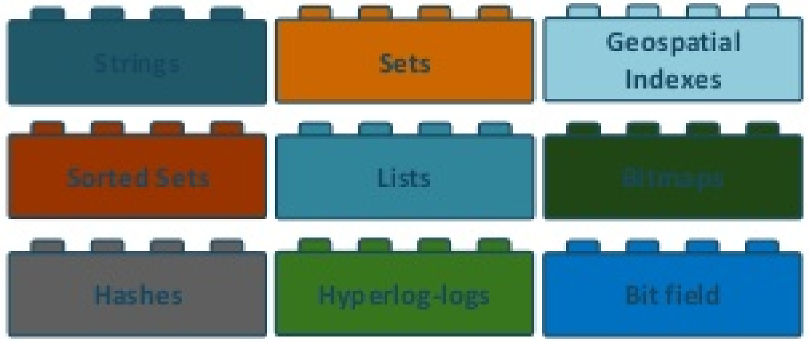
RAM & Flash

A Brief Overview of Redis

- Started in 2009 by Salvatore Sanfilippo
- Most popular KV store
- In memory - disk backed
- Notable Users:
  - Twitter, Netflix, Uber, Groupon, Twitch
  - Many, many more...

Redis Main Differentiations





Strings

Sets

Geospatial Indexes

Sorted Sets

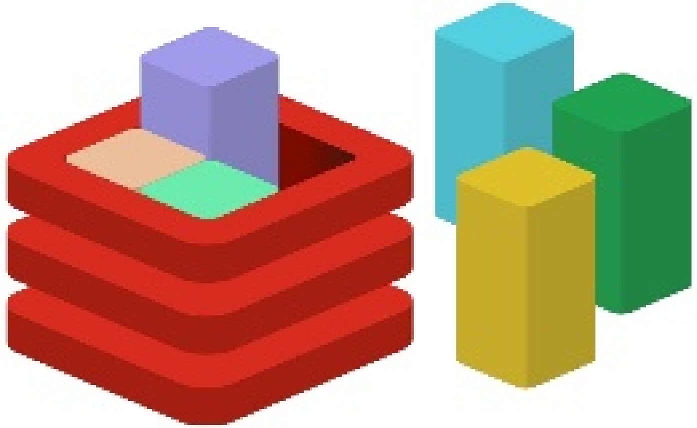
Lists

Bitmaps

Hashes

Hyperlog-logs

Bit field



Performance

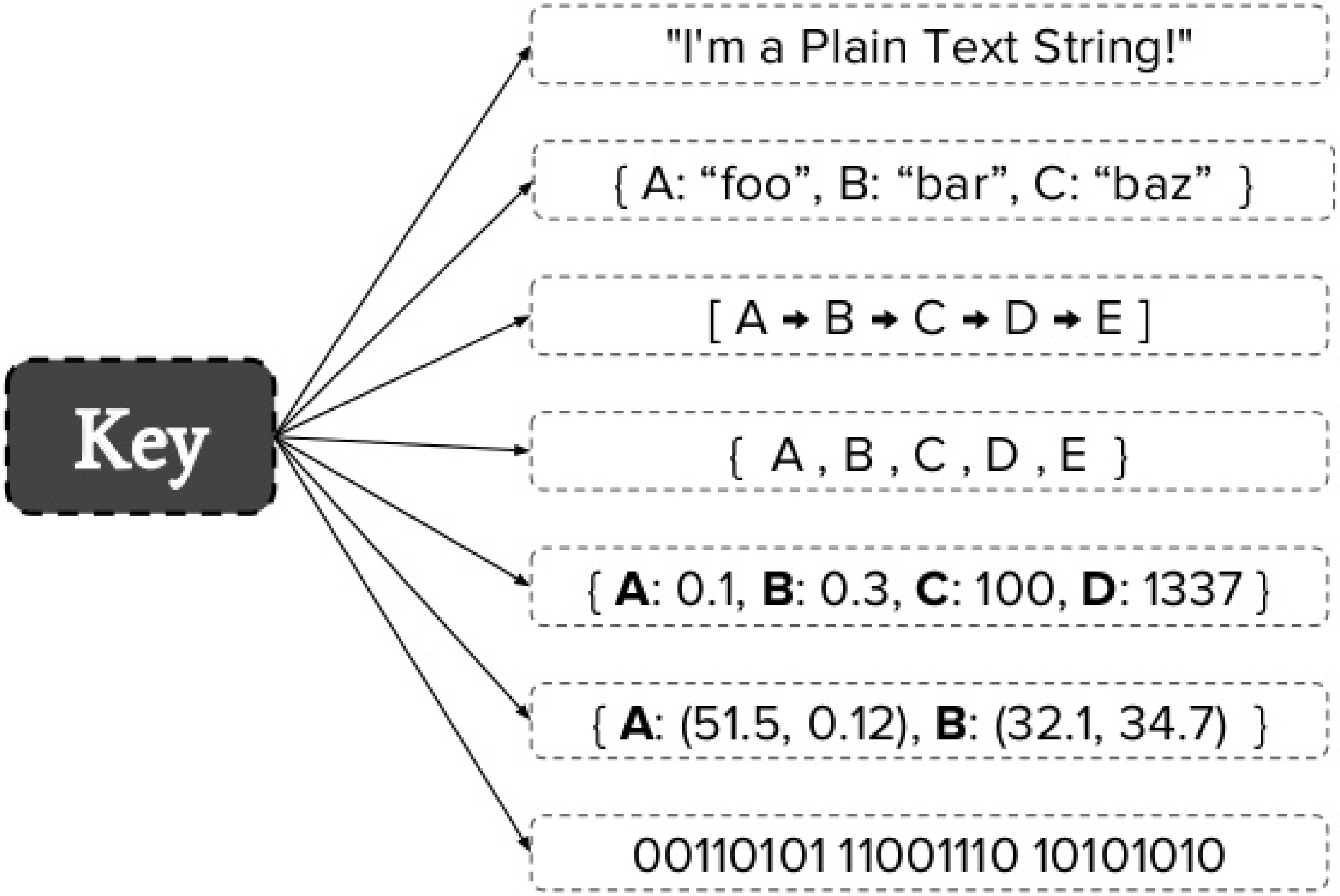
Simplicity

(through Data Structures)

Extensibility

(through Redis Modules)

A Quick Recap of Redis



Key

"I'm a Plain Text String!" — Strings / Bitmaps / BitFields

{ A: “foo”, B: “bar”, C: “baz” } — Hash Tables (objects!)

[A → B → C → D → E] — Linked Lists

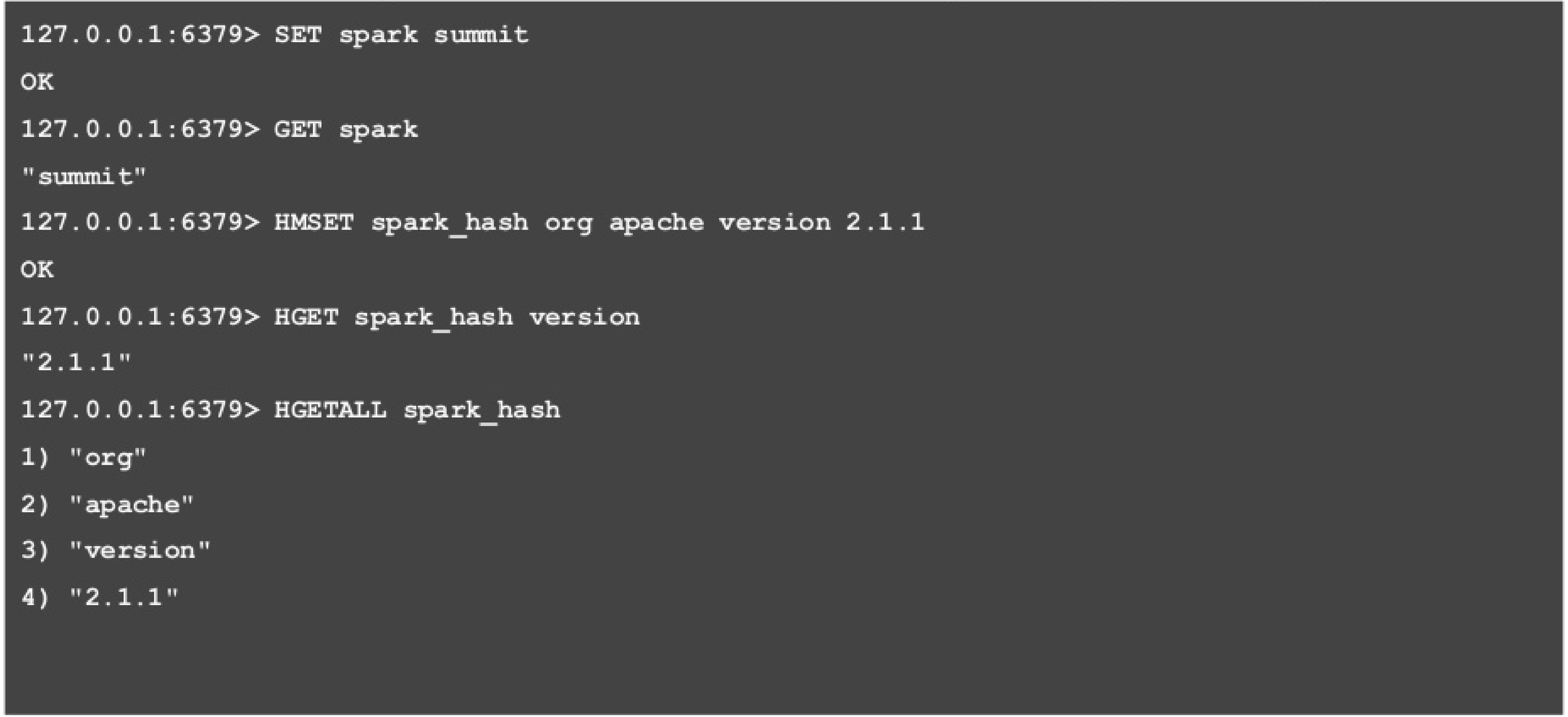
{ A , B , C , D , E } — Sets

{ **A**: 0.1, **B**: 0.3, **C**: 100, **D**: 1337 } — Sorted Sets

{ **A**: (51.5, 0.12), **B**: (32.1, 34.7) } — Geo Sets

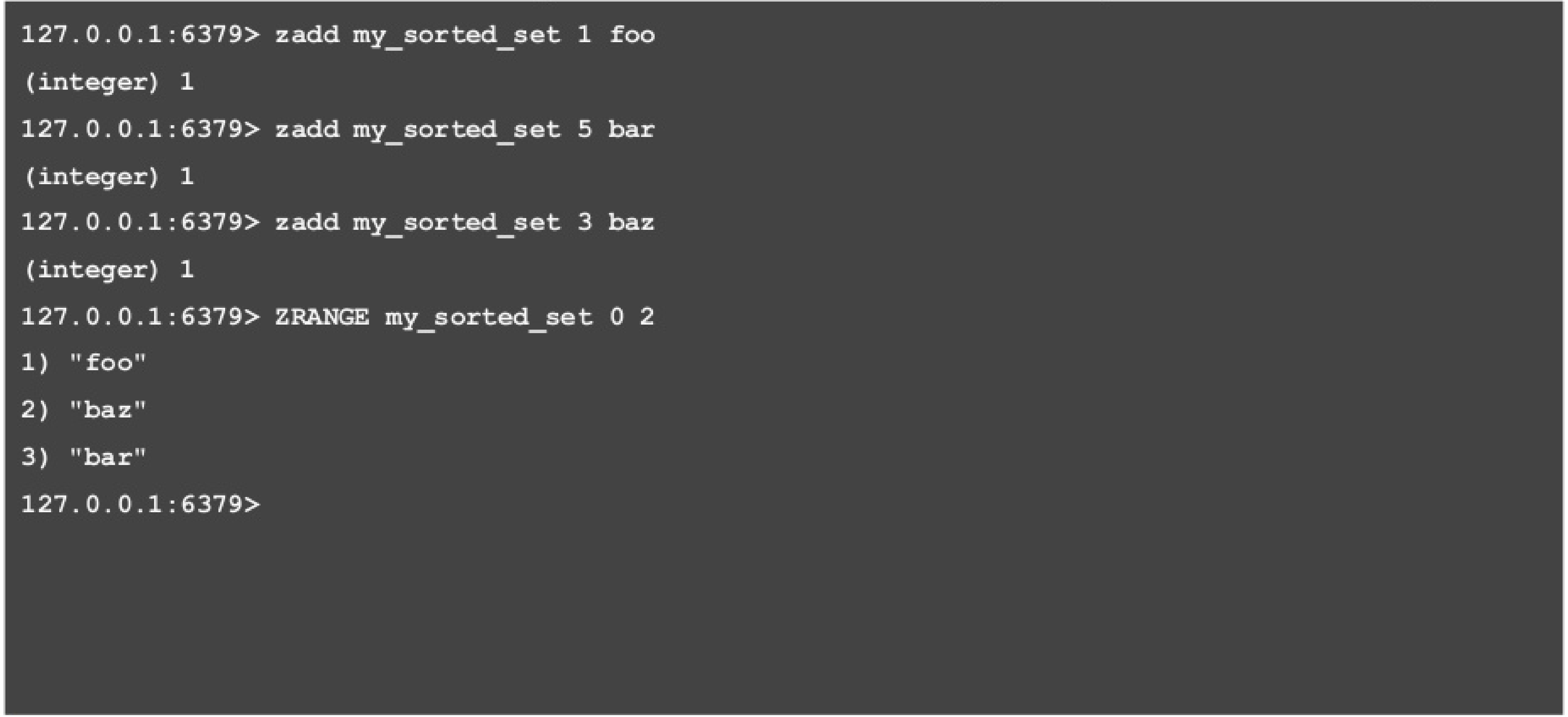
00110101 11001110 10101010 — HyperLogLog

Simple Redis Example (string, hash)



```
127.0.0.1:6379> SET spark summit
OK
127.0.0.1:6379> GET spark
"summit"
127.0.0.1:6379> HMSET spark_hash org apache version 2.1.1
OK
127.0.0.1:6379> HGET spark_hash version
"2.1.1"
127.0.0.1:6379> HGETALL spark_hash
1) "org"
2) "apache"
3) "version"
4) "2.1.1"
```

Another Simple Redis Example (sorted set)



```
127.0.0.1:6379> zadd my_sorted_set 1 foo
(integer) 1
127.0.0.1:6379> zadd my_sorted_set 5 bar
(integer) 1
127.0.0.1:6379> zadd my_sorted_set 3 baz
(integer) 1
127.0.0.1:6379> ZRANGE my_sorted_set 0 2
1) "foo"
2) "baz"
3) "bar"
127.0.0.1:6379>
```

What Modules Actually Are

- Dynamic libraries loaded to redis
- Written in C/C++
- Use a C ABI/API isolating redis internals
- Use existing or add new data-structures
- Near Zero latency access to data

New Data Types

New Commands

New Capabilities

Modules : A Revolutionary Approach

Adapt your database to your data, not the other way around

Neural Redis

Simple Neural Network Native to Redis

Redis-ML

Machine Learning Model Serving

RediSearch

Full Text Search Engine in Redis

ReJSON

JSON Engine on Redis.
Pre-released

Time Series

Time series values aggregation in Redis

Graph

Graph database on Redis based on Cypher language

Rate Limiter

Based on Generic Cell Rate Algorithm (GCRA)

Crypto Engine Wrapper

Secure way to store data in Redis via encrypt/decrypt with various Themis primitives

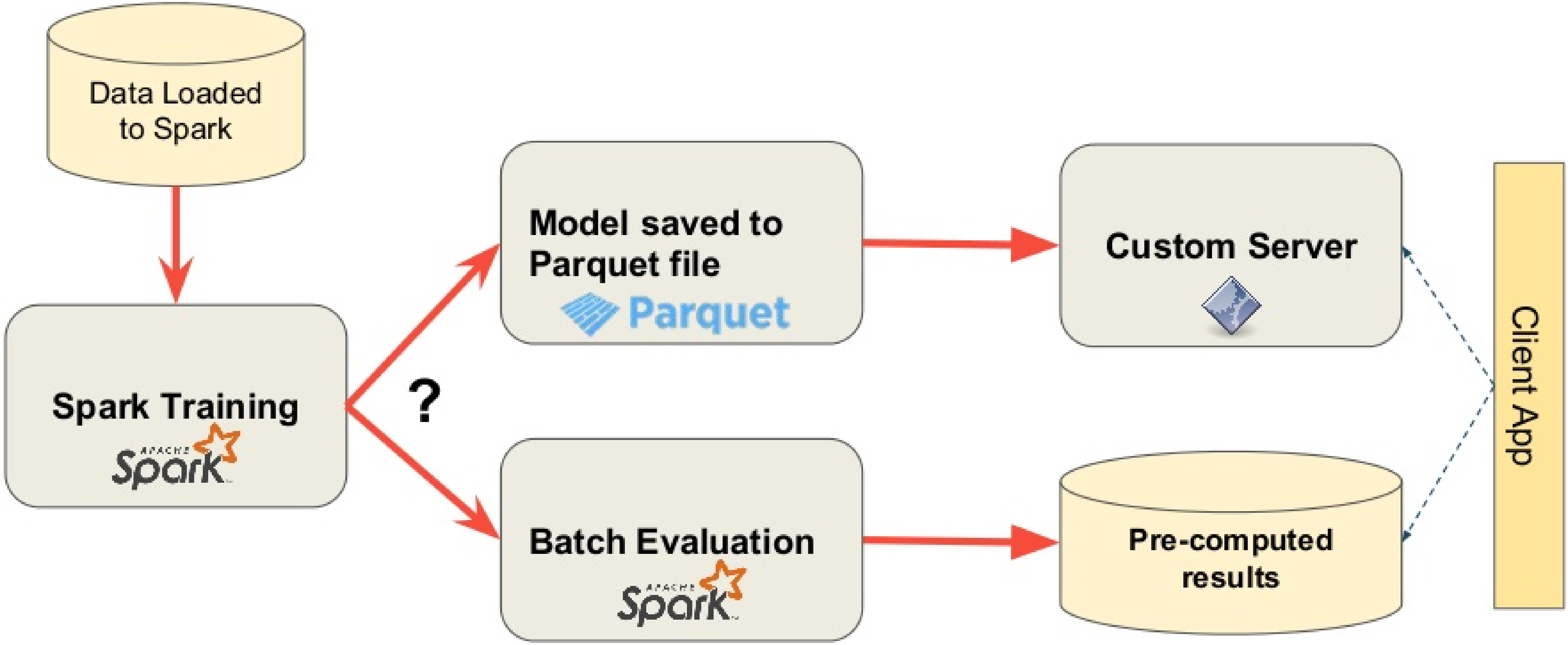
Secondary Index/RQL

Indexing + SQL -like syntax for querying indexes.
Pre-released

Redis ML

Machine Learning Model Server

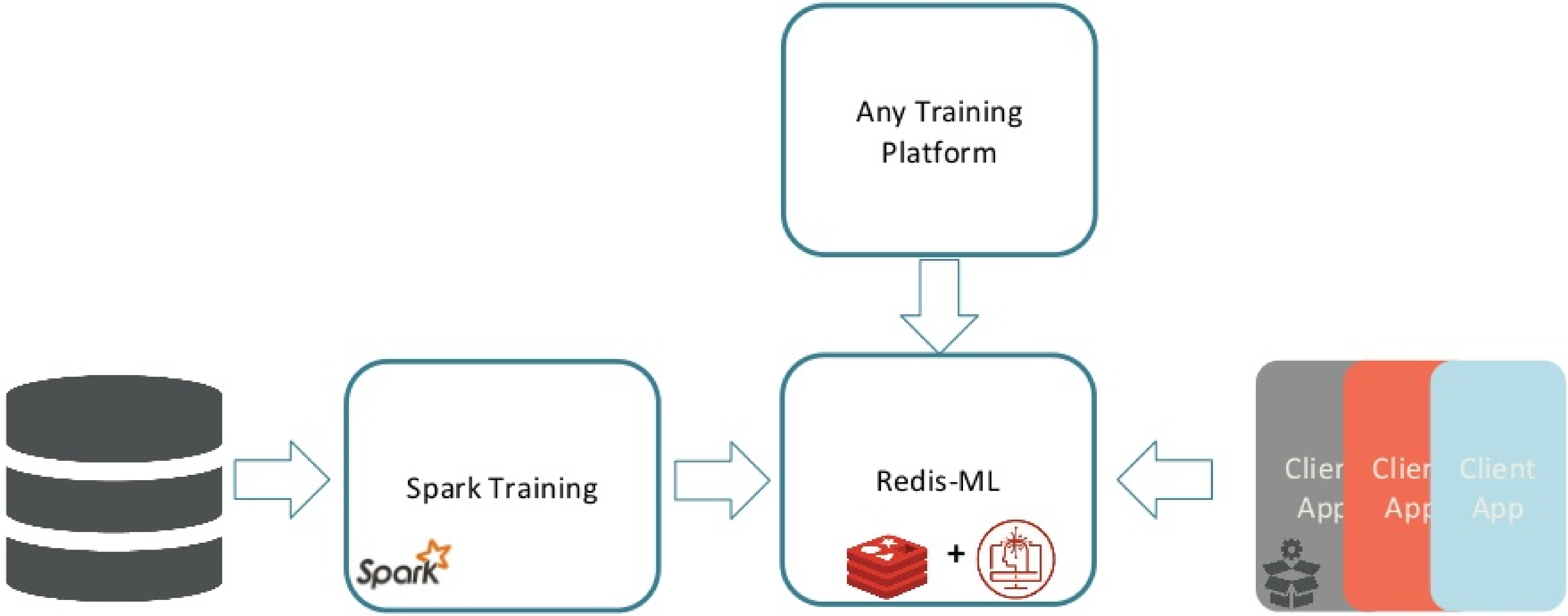
Spark-ML End-to-End Flow



ML Models Serving Challenges

- Models are becoming bigger and more complex
- Can be challenging to deploy & serve
- Do not scale well, speed and size
- Can be very expensive

A Simpler Machine Learning Lifecycle



Data loaded into Spark

Model is saved in Redis-ML

Serving Client

Redis-ML – ML Serving Engine

- Store training output as “hot model”
- Perform evaluation directly in Redis
- Enjoy the performance, scalability and HA of Redis

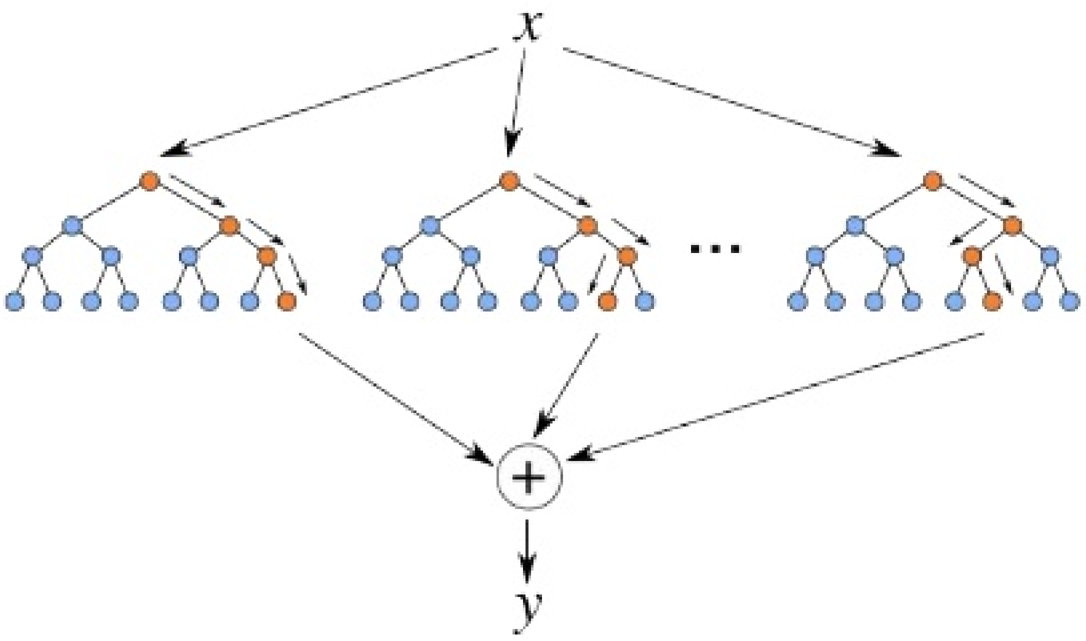
Redis-ML

ML Models

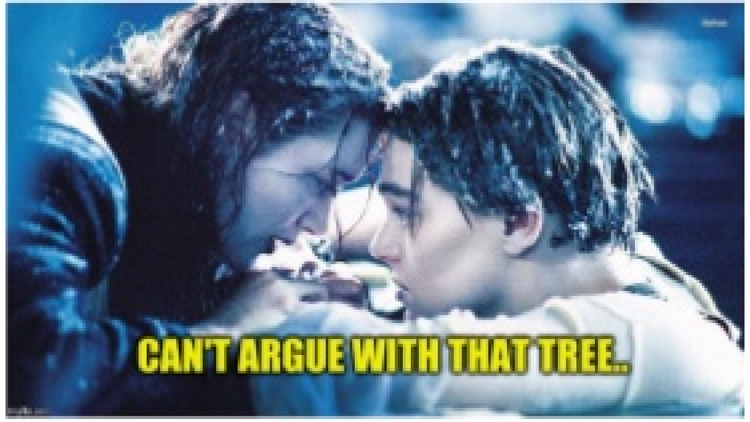
- Tree Ensembles
- Linear Regression
- Logistic Regression
- Matrix + Vector Operations
- More to come…

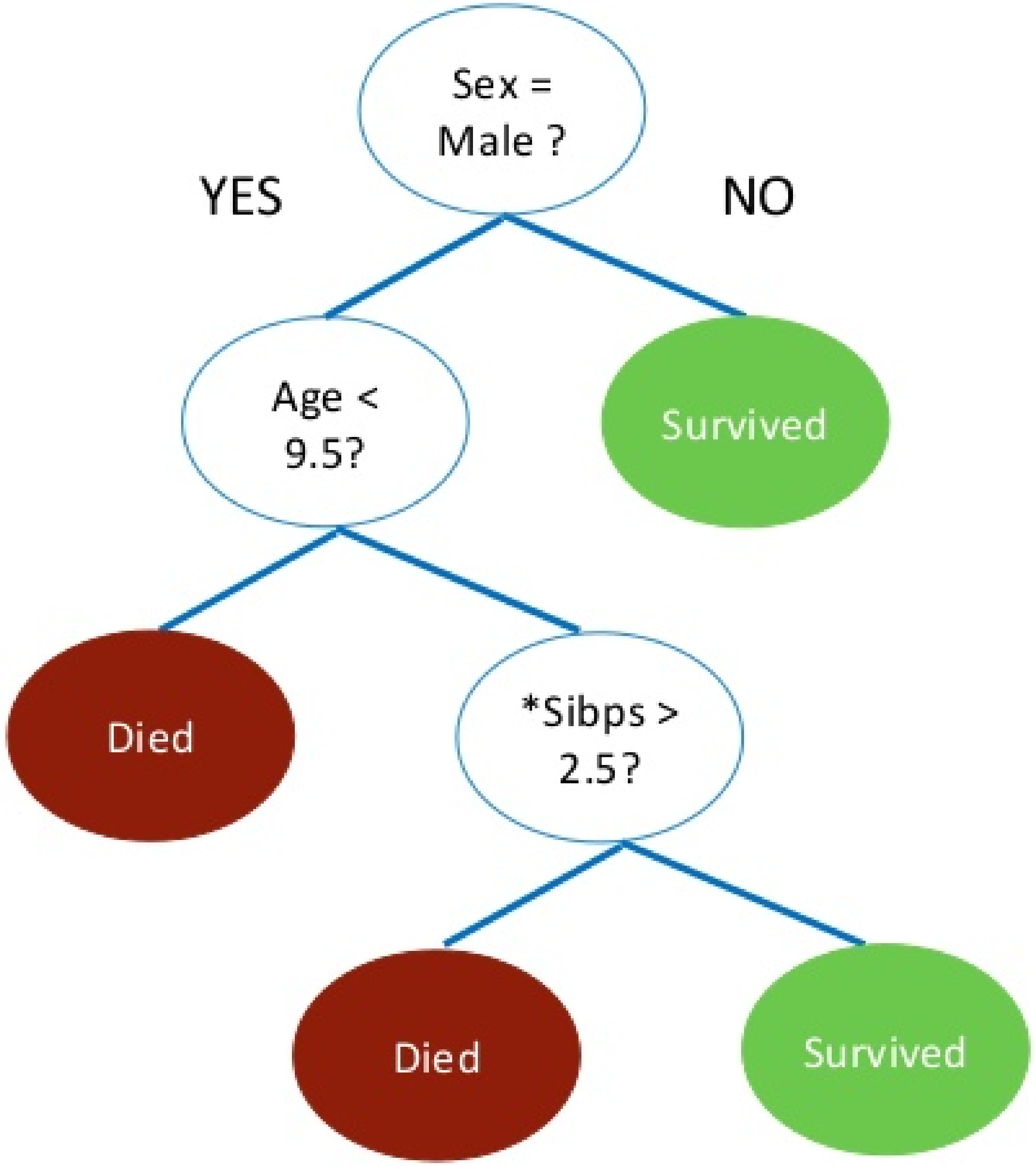
Random Forest Model

- A collection of decision trees
- Supports classification & regression
- Splitter Node can be:
  - Categorical (e.g. day == “Sunday”)
  - Numerical (e.g. age < 43)
- Decision is taken by the majority of decision trees



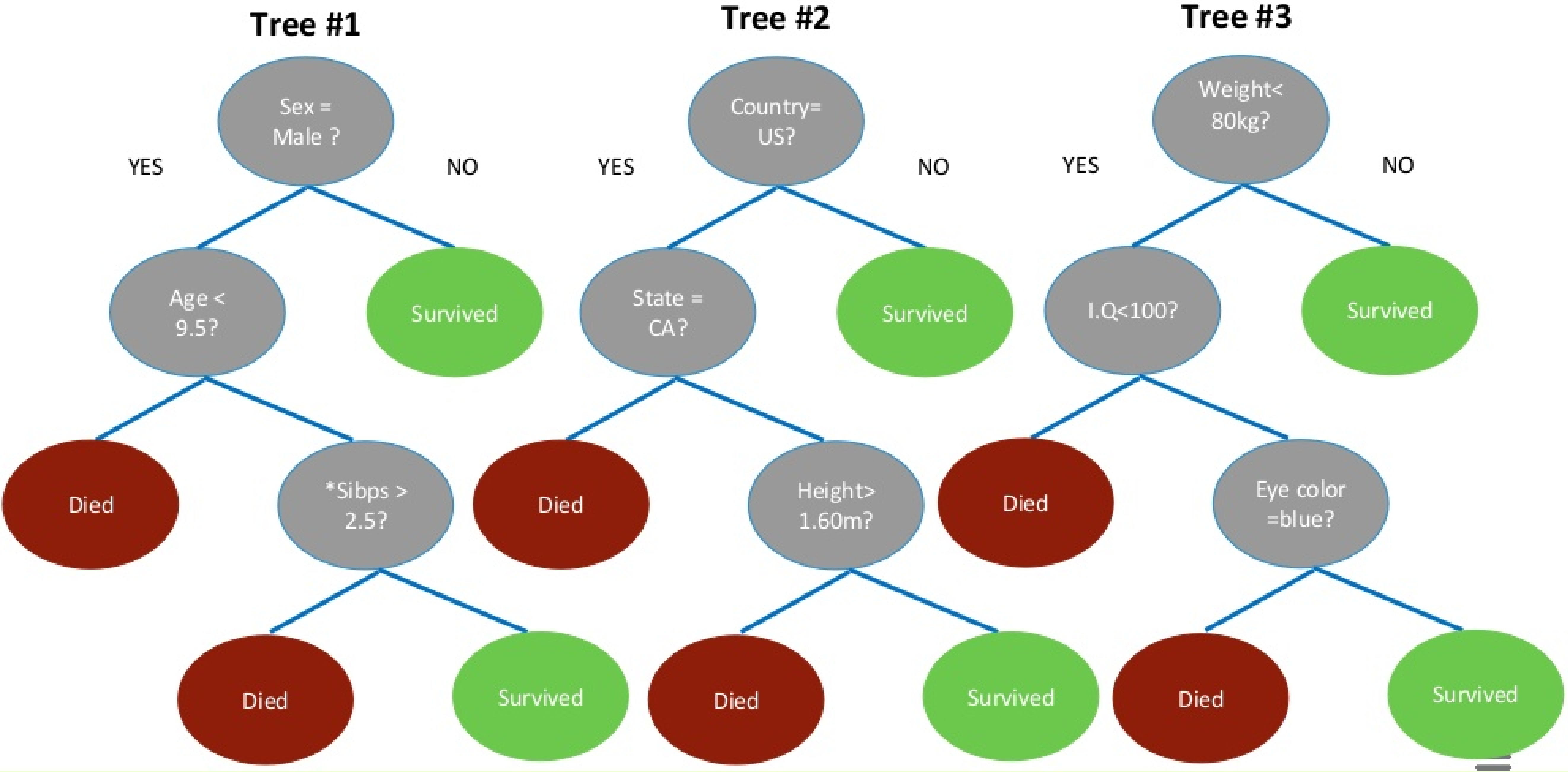
Titanic Survival Predictor on a Decision Tree





*Sibps = siblings + spouses

Titanic Survival Predictor on a Random Forest



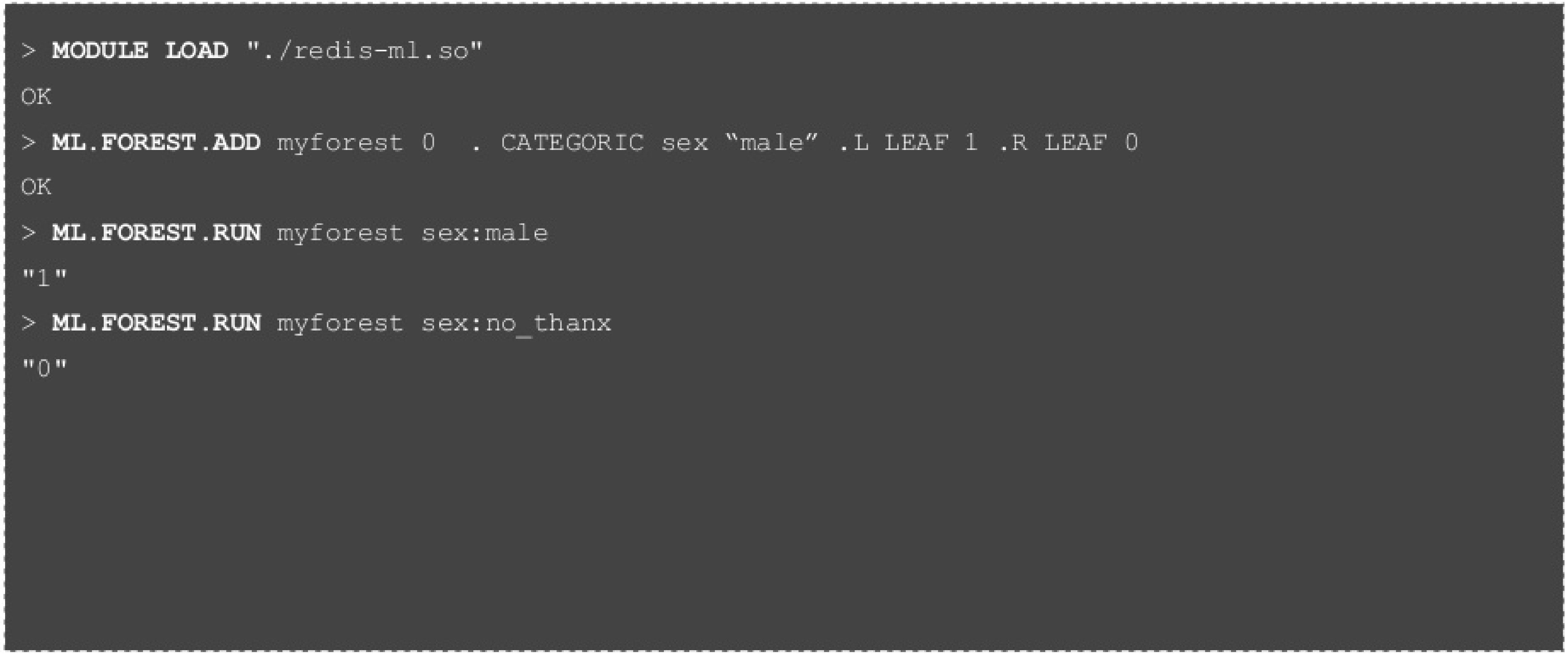
Would John Survive The Titanic

- John's features:

  {male, 34, married + 2, US, CA, 1.78m, 78kg, 110iq, blue eyes}

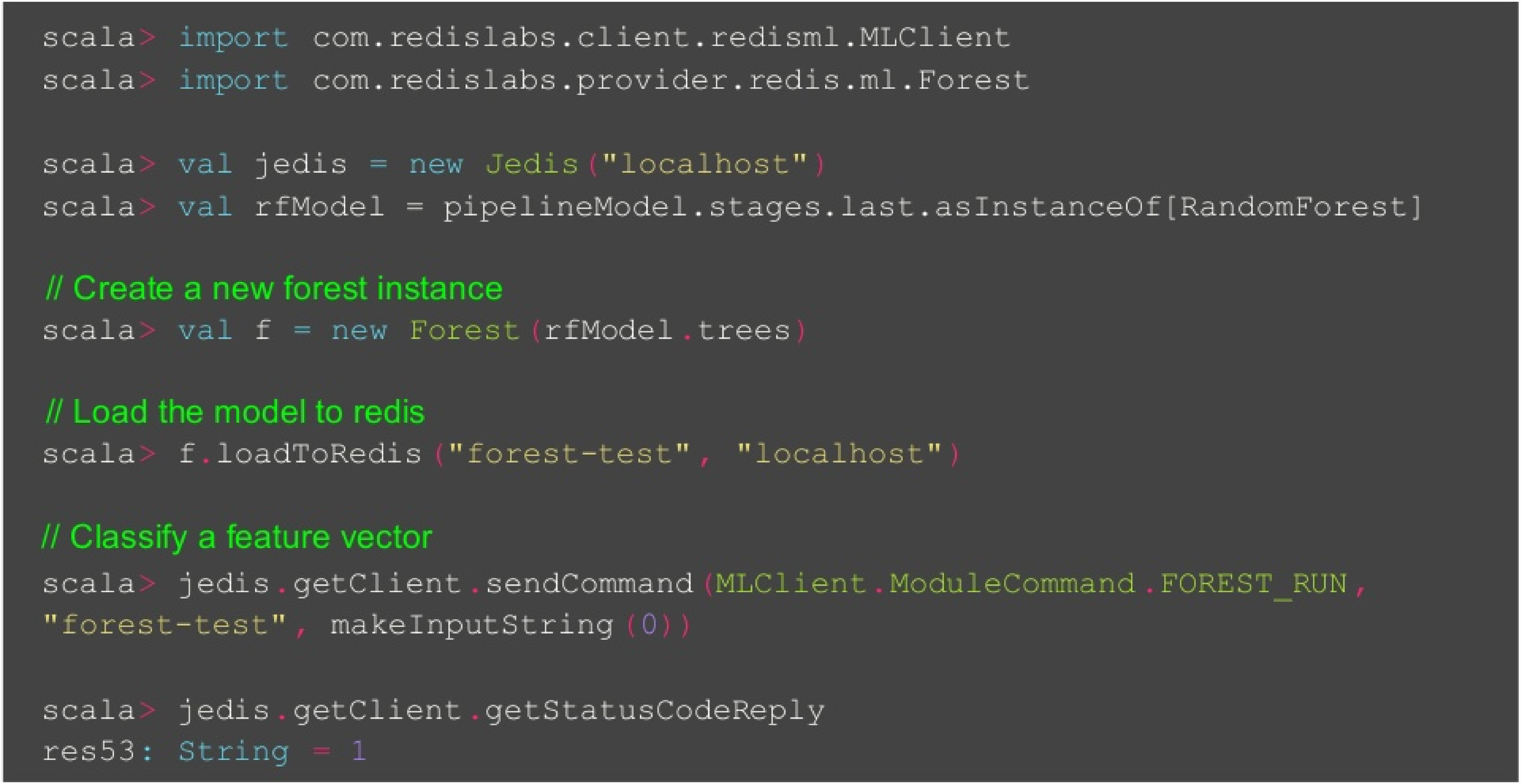
- Tree#1 – Survived
- Tree#2 – Failed
- Tree#3 – Survived
- Random forest decision - Survived

Forest Data Type Example



```
> MODULE LOAD "./redis-ml.so"
OK
> ML.FOREST.ADD myforest 0  . CATEGORIC sex “male” .L LEAF 1 .R LEAF 0
OK
> ML.FOREST.RUN myforest sex:male
"1"
> ML.FOREST.RUN myforest sex:no_thanx
"0"
```

Using Redis-ML With Spark



```
scala> import com.redislabs.client.redisml.MLClient
scala> import com.redislabs.provider.redis.ml.Forest

scala> val jedis = new Jedis("localhost")
scala> val rfModel = pipelineModel.stages.last.asInstanceOf[RandomForest]

// Create a new forest instance
scala> val f = new Forest(rfModel.trees)

// Load the model to redis
scala> f.loadToRedis("forest-test", "localhost")

// Classify a feature vector
scala> jedis.getClient.sendCommand(MLClient.ModuleCommand.FOREST_RUN,
"forest-test", makeInputString(0))

scala> jedis.getClient.getStatusCodeReply
res53: String = 1
```

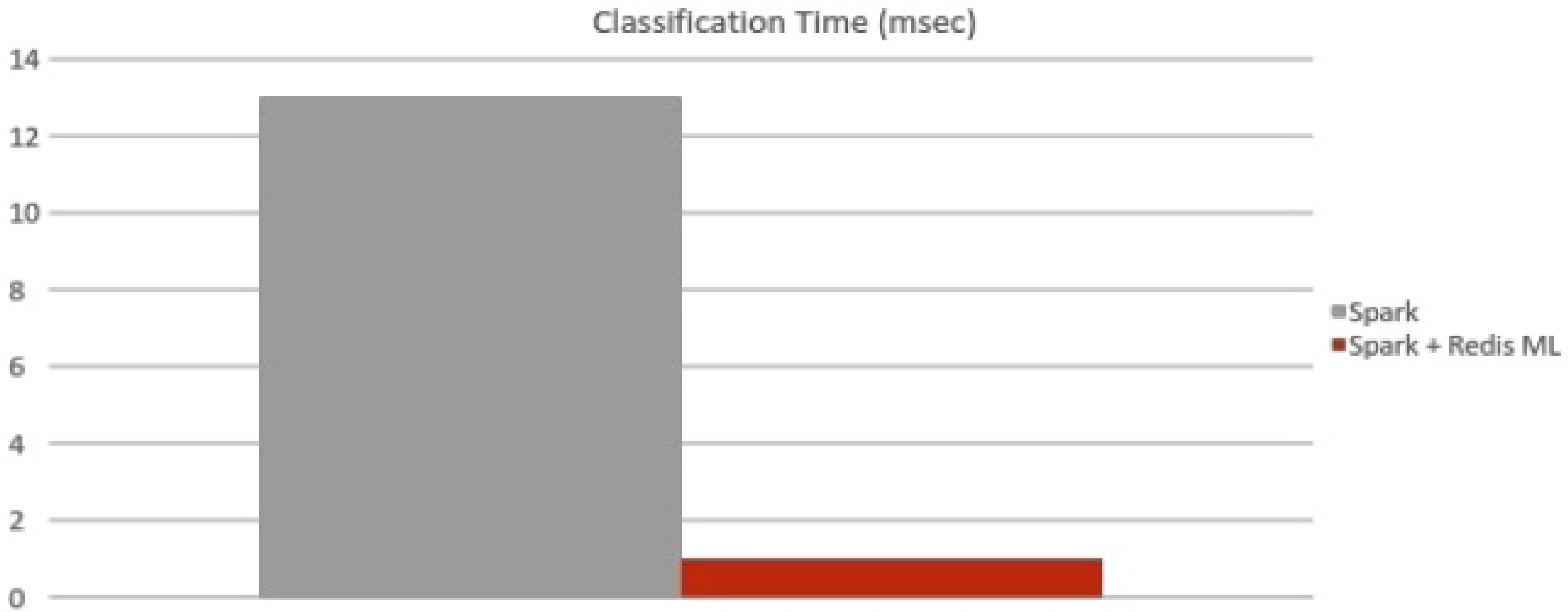
Real World Challenge

- Ad serving company
- Need to serve 20,000 ads/sec @ 50msec data-center latency
- Runs 1k campaigns → 1K random forest
- Each forest has 15K trees
- On average each tree has 7 levels (depth)
- Would require < 1000 x c4.8xlarge

Redis ML with Spark ML

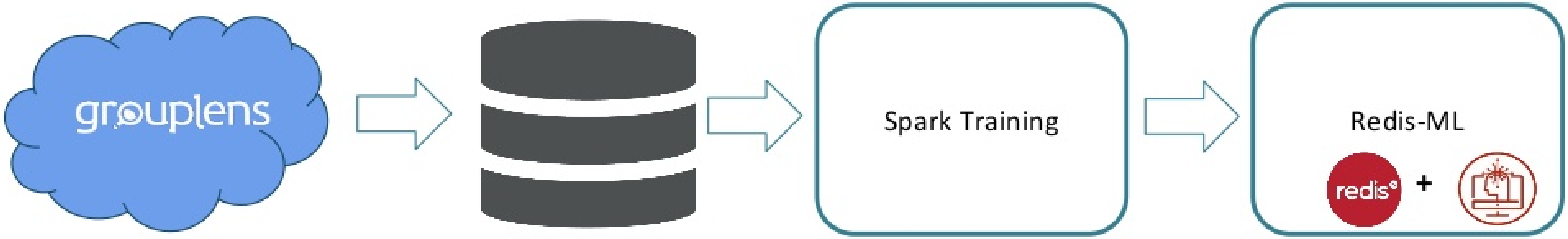
40x Faster

Classification Time Over Spark

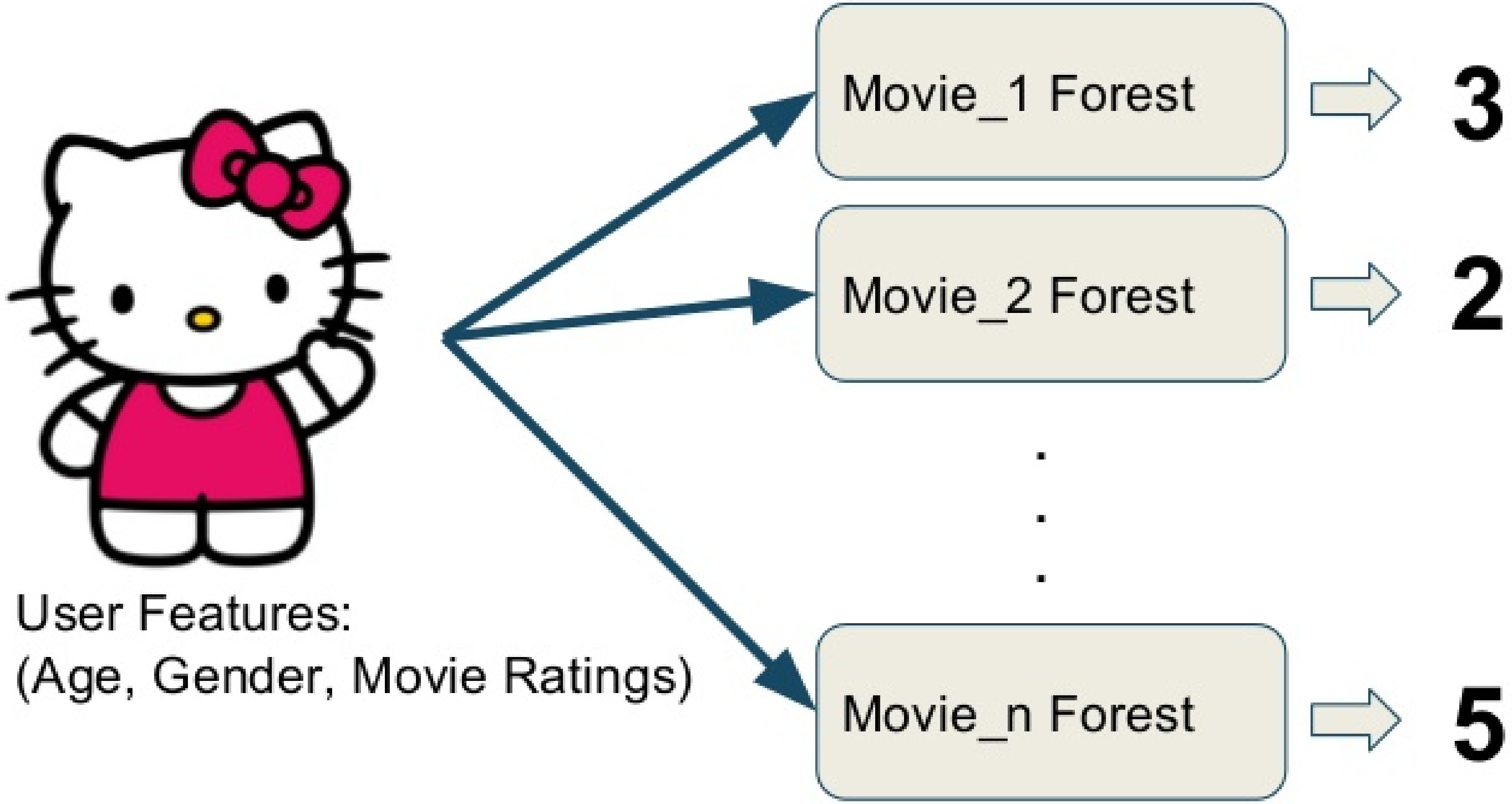


Real World Example:
Movie Recommendation System

Overview



Concept: One Forest For Each Movie



User Features:
(Age, Gender, Movie Ratings)

Movie_1 Forest → 3

Movie_2 Forest → 2

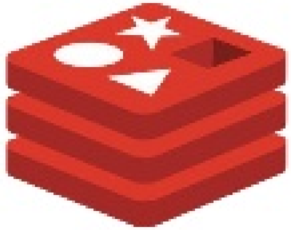
.
.
.

Movie_n Forest → 5

The Tools

Transform:

Train:

Classify:  +

Containers:

Using the Dockers

```
$ docker pull shaynativ/redis-ml
$ docker run  --net=host shaynativ/redis-ml &
$
$ docker pull shaynativ/spark-redis-ml
$ docker run  --net=host shaynativ/spark-redis-ml
```

Step 1: Get The Data

- Download and extract the MovieLens 100K Dataset
- The data is organized in separate files:
    - Ratings: user id | item id | rating (1-5) | timestamp
    - Item (movie) info: movie id | genre info fields (1/0)
    - User info: user id | age | gender | occupation
- Our classifier should return the expected rating (from 1 to 5) a user would give the movie in question

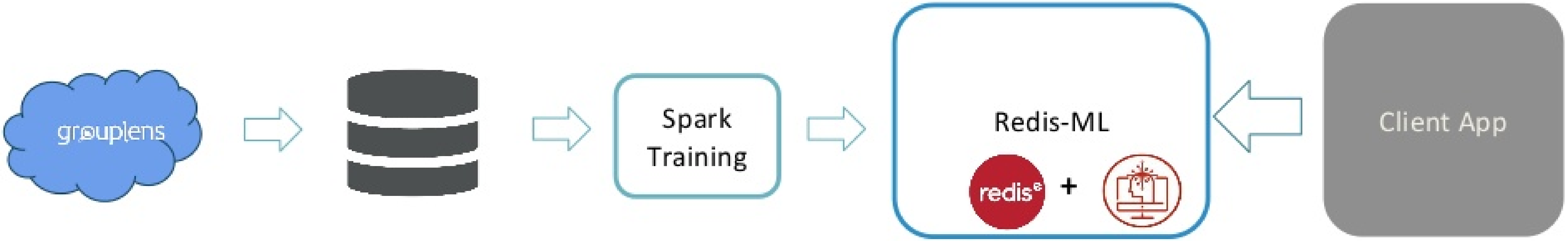
Step 2: Transform

- The training data for each movie should contain 1 line per user:
    - class (rating from 1 to 5 the user gave to this movie)
    - user info (age, gender, occupation)
    - user ratings of other movies (movie_id:rating ...)
    - user genre rating averages (genre:avg_score ...)
- Run gen_data.py to transform the files to the desired format

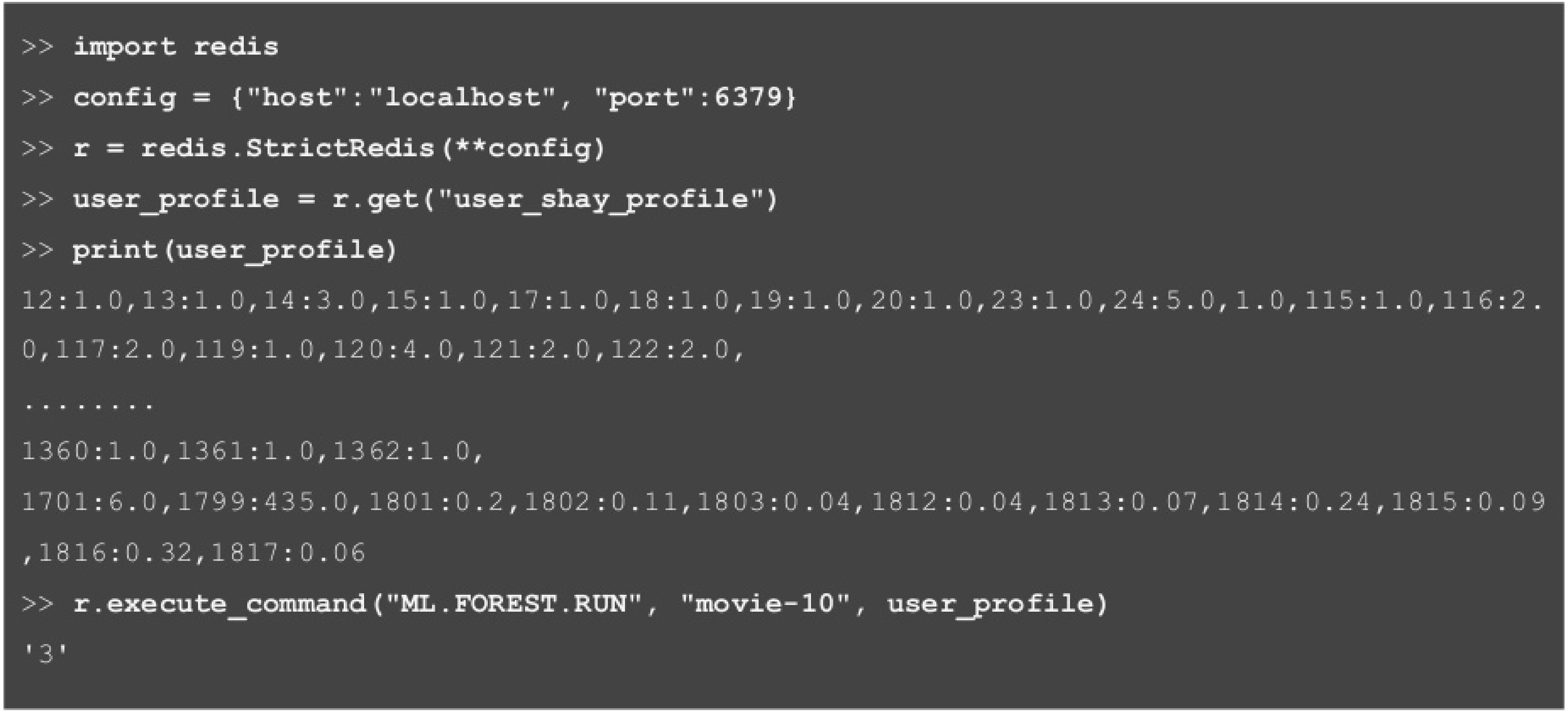
Step3: Train and Load to Redis

```
// Create a new forest instance
val rf = new
RandomForestClassifier().setFeatureSubsetStrategy("auto").setLabelCol("indexedLabel").setFeat
uresCol("indexedFeatures").setNumTrees(500)
…..
// Train model
val model = pipeline.fit(trainingData)
…..
val rfModel = model.stages(2).asInstanceOf[RandomForestClassificationModel]

// Load the model to redis
val f = new Forest(rfModel.trees)
f.loadToRedis(”movie-10", "127.0.0.1")
```

Step 3: Execute in Redis

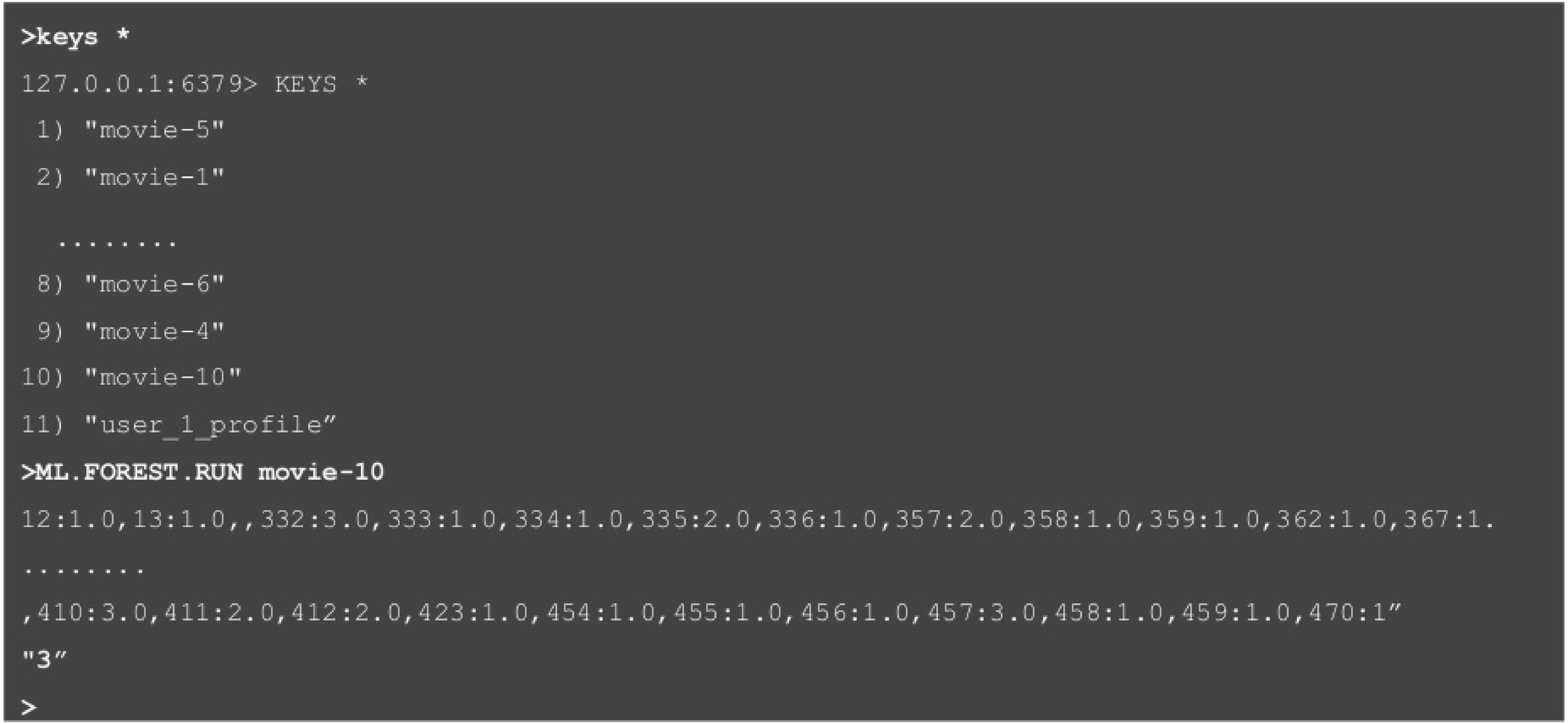


Python Client Example



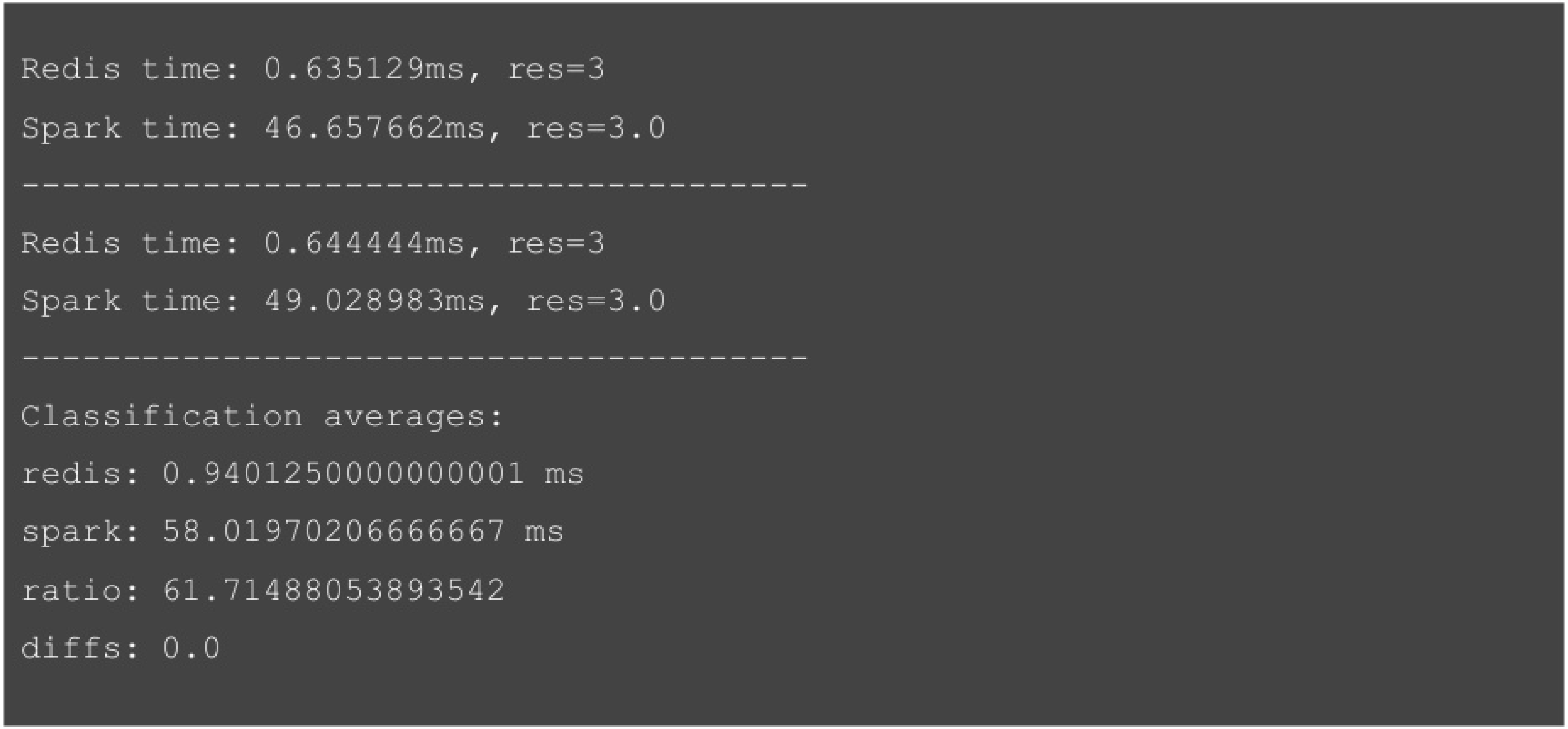
```
>> import redis
>> config = {"host":"localhost", "port":6379}
>> r = redis.StrictRedis(**config)
>> user_profile = r.get("user_shay_profile")
>> print(user_profile)
12:1.0,13:1.0,14:3.0,15:1.0,17:1.0,18:1.0,19:1.0,20:1.0,23:1.0,24:5.0,1.0,115:1.0,116:2.
0,117:2.0,119:1.0,120:4.0,121:2.0,122:2.0,
........
1360:1.0,1361:1.0,1362:1.0,
1701:6.0,1799:435.0,1801:0.2,1802:0.11,1803:0.04,1812:0.04,1813:0.07,1814:0.24,1815:0.09
,1816:0.32,1817:0.06
>> r.execute_command("ML.FOREST.RUN", "movie-10", user_profile)
'3'
```

Redis CLI Example



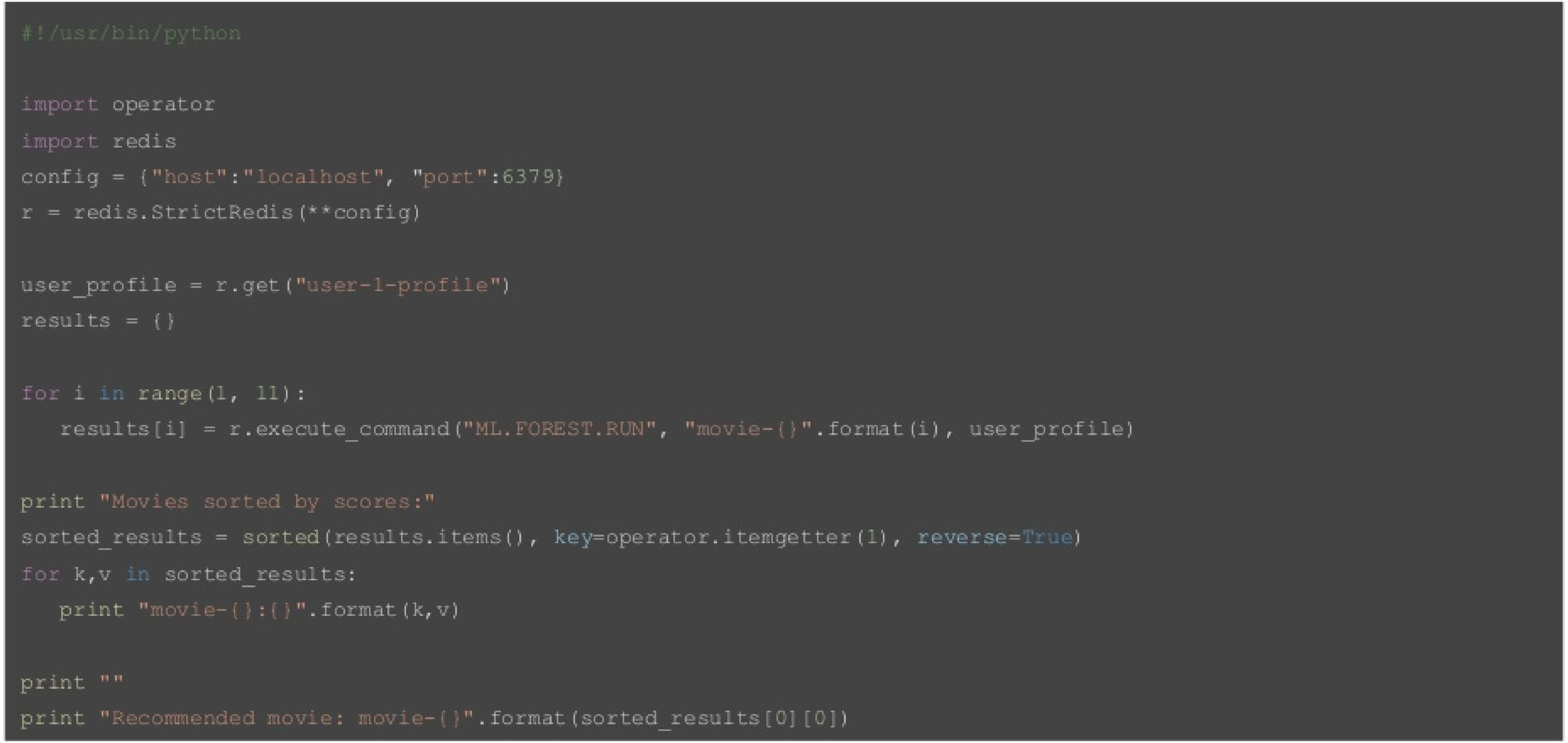
```
>keys *
127.0.0.1:6379> KEYS *
 1) "movie-5"
 2) "movie-1"

  ........
 8) "movie-6"
 9) "movie-4"
10) "movie-10"
11) "user_1_profile”
>ML.FOREST.RUN movie-10
12:1.0,13:1.0,,332:3.0,333:1.0,334:1.0,335:2.0,336:1.0,357:2.0,358:1.0,359:1.0,362:1.0,367:1.
........
,410:3.0,411:2.0,412:2.0,423:1.0,454:1.0,455:1.0,456:1.0,457:3.0,458:1.0,459:1.0,470:1”
"3”
>
```

Performance



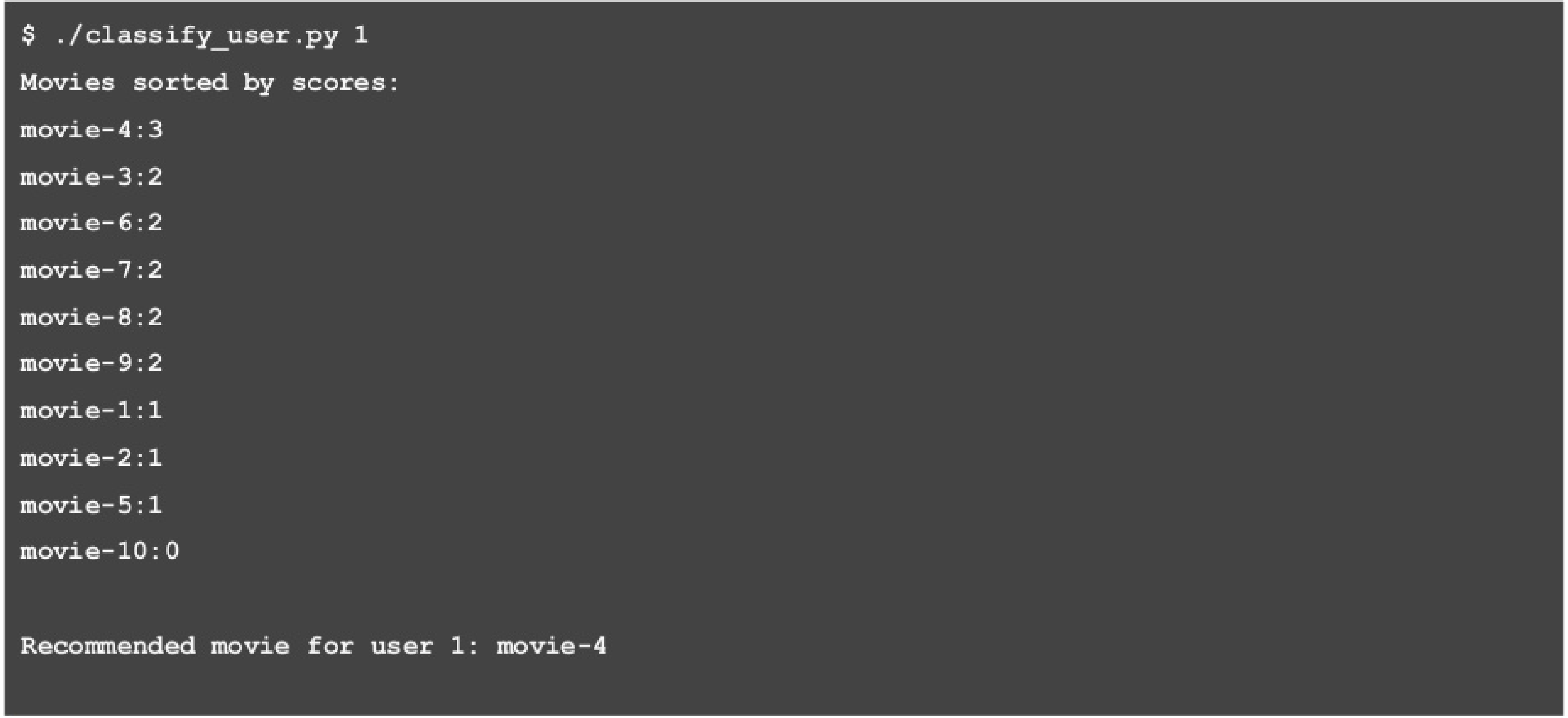
```
Redis time: 0.635129ms, res=3
Spark time: 46.657662ms, res=3.0
---------------------------------------
Redis time: 0.644444ms, res=3
Spark time: 49.028983ms, res=3.0
---------------------------------------
Classification averages:
redis: 0.9401250000000001 ms
spark: 58.01970206666667 ms
ratio: 61.71488053893542
diffs: 0.0
```

Getting Actual Recommendations - Python Script



```python
#!/usr/bin/python

import operator
import redis
config = {"host":"localhost", "port":6379}
r = redis.StrictRedis(**config)

user_profile = r.get("user-1-profile")
results = {}

for i in range(1, 11):
   results[i] = r.execute_command("ML.FOREST.RUN", "movie-{}".format(i), user_profile)

print "Movies sorted by scores:"
sorted_results = sorted(results.items(), key=operator.itemgetter(1), reverse=True)
for k,v in sorted_results:
   print "movie-{}:{}".format(k,v)

print ""
print "Recommended movie: movie-{}".format(sorted_results[0][0])
```

Getting Actual Recommendations - Results



```
$ ./classify_user.py 1
Movies sorted by scores:
movie-4:3
movie-3:2
movie-6:2
movie-7:2
movie-8:2
movie-9:2
movie-1:1
movie-2:1
movie-5:1
movie-10:0


Recommended movie for user 1: movie-4
```

Summary

- Train with Spark, Serve with Redis
- 97% resource cost serving
- Simplify ML lifecycle

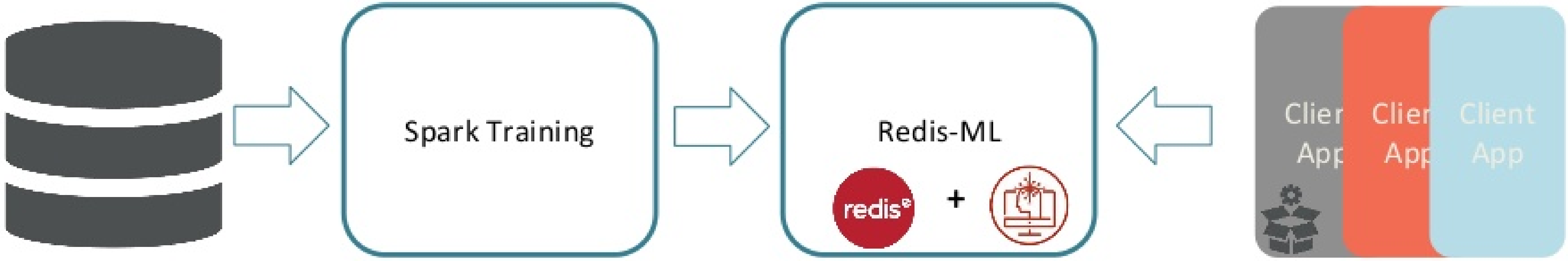
- Redis^e (Cloud or Pack):

–Scaling, HA, Performance

–PAYG – cost optimized

–Ease of use

–Supported by the teams who created Spark and Redis



Data loaded into Spark

Model is saved in Redis-ML

Serving Client

Resources

- Redis-ML: https://github.com/RedisLabsModules/redis-ml
- Spark-Redis-ML: https://github.com/RedisLabs/spark-redis-ml
- Databricks Notebook: http://bit.ly/sparkredisml
- Dockers: https://hub.docker.com/r/shaynativ/redis-ml/
  https://hub.docker.com/r/shaynativ/spark-redis-ml/

Q&A

Thank You.

shay@redislabs.com