# Towards Selecting Best Combination of SQL-on-Hadoop Systems and JVMs

**Tatsuhiro Chiba**, Takeshi Yoshimura,
Michihiro Horie and Hiroshi Horii

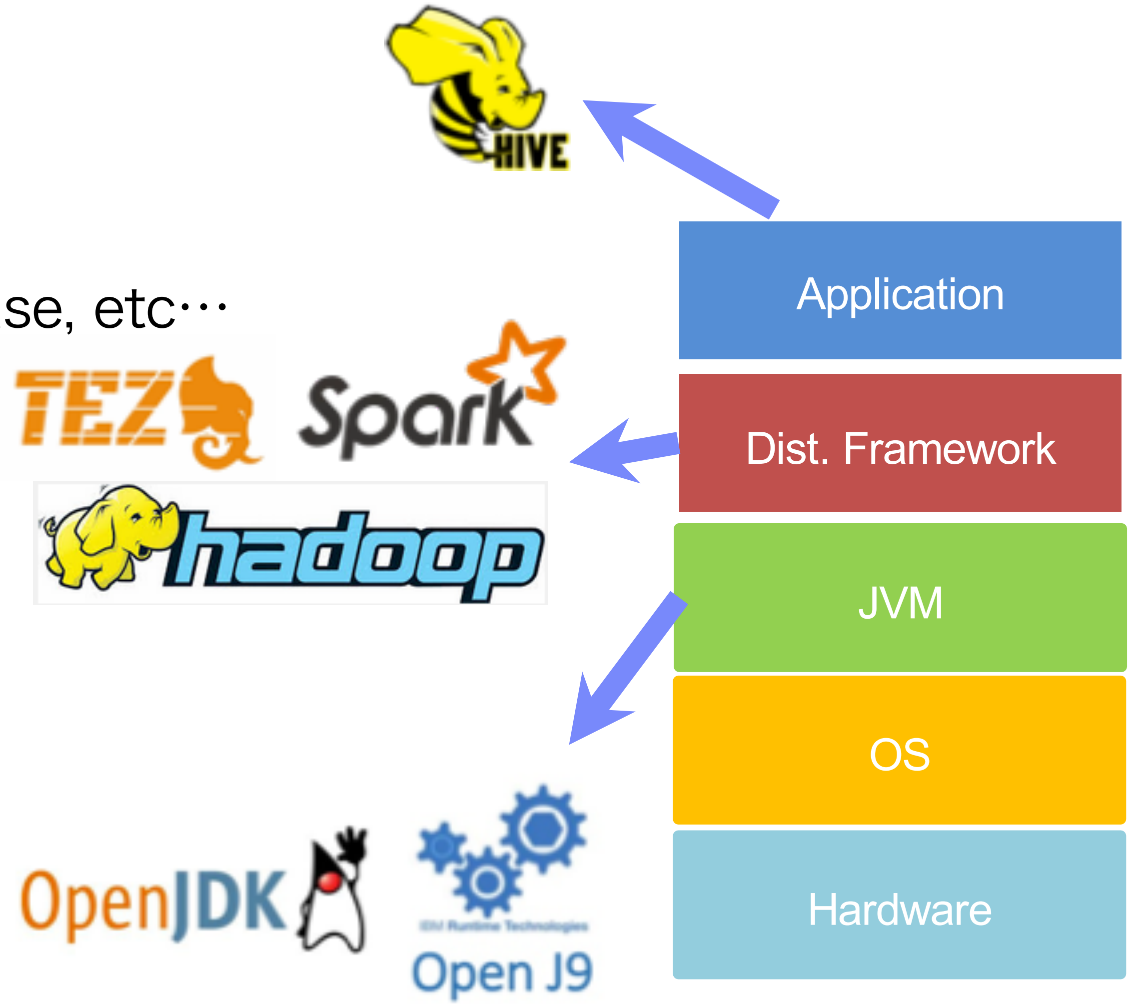
**IBM Research**

# Agenda

- **Motivation, Problems and Challenges**
- **Backgrounds**
  - Backend engines: Spark and Tez
  - Backend runtimes: OpenJDK and J9
- **Empirical Study**
  - Performance evaluation
  - Performance analysis
- **ML model**
  - training classification model
  - evaluating classification model
- **Summary**

# Distributed Processing Framework for Big Data

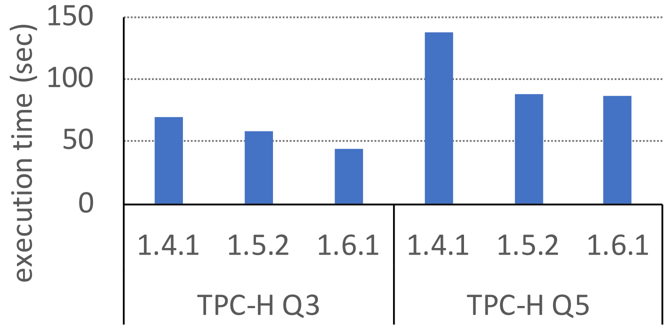
- **Hadoop Eco-Systems**
  - HDFS: the center of data store
  - utilizing data between different frameworks
    - Spark, Tez, Flink, YARN, MR, Hive, Pig, Hbase, etc…
- **Big Data Workload**
  - ETL
  - SQL
  - ML / DL / Streaming
- **JVM as a Hadoop Runtime**
  - disk-oriented → in-memory oriented
  - I/O intensive → CPU-intensive



# Motivation and Problem – Many choices of the systems

- **Rapid Development Cycle**
  - Fast open sources releases
  - marge new feature frequently
  - query performance is also improved

Performance Improvement History




- **Too many SQL-on-Hadoop Systems**
  - Which one is best? (SparkSQL or Hive or Impara or Presto or …)
  - Should we switch a system to another one?
  - No single SQL-on-Hadoop engine is best for ALL queries
  - No single JVM is best for ALL queries as well

# Motivation and Problem – Selecting a system adaptively

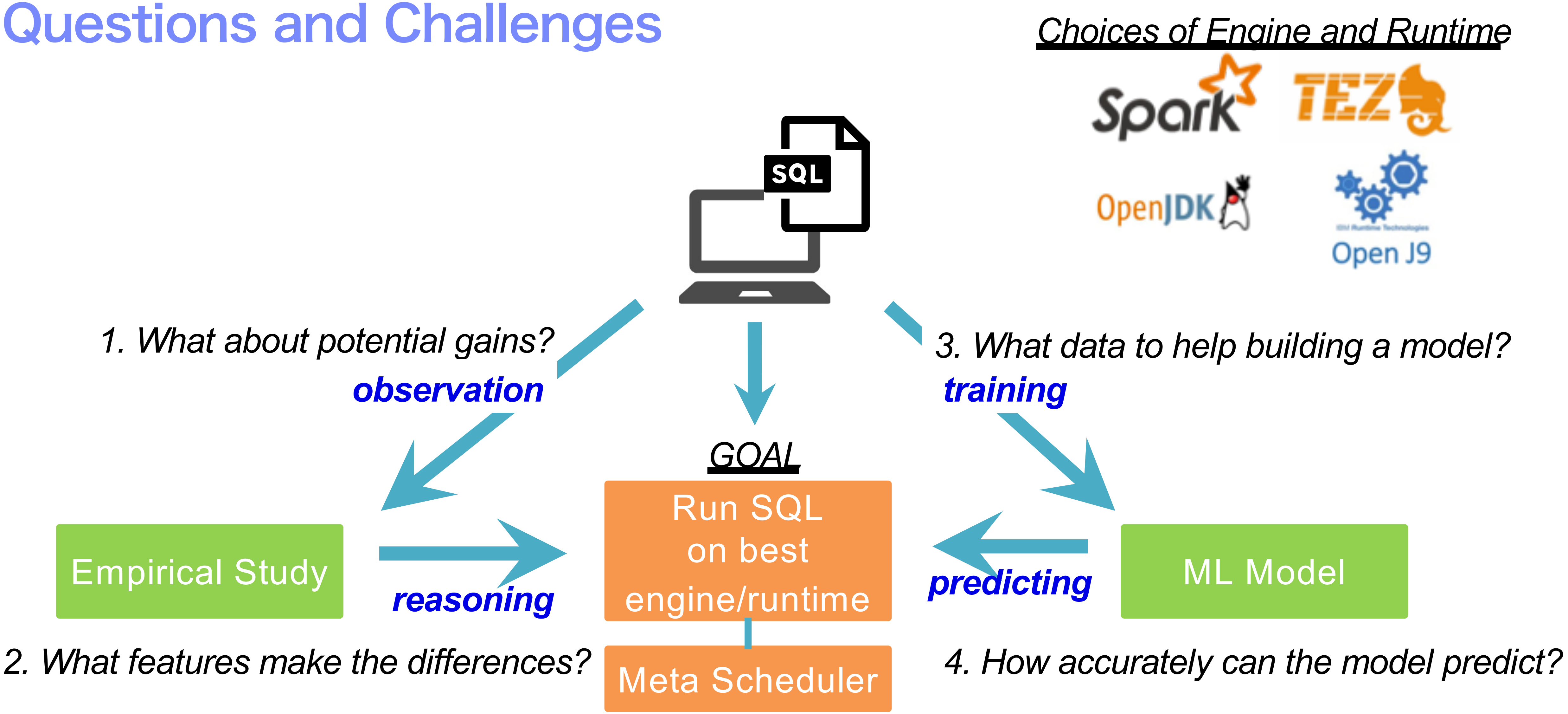
- **Requirements of Query Execution on Cloud**
  – query users: do not care about backend system as long as it returns a result fast
  – cloud providers: wants to minimize resources by using fast processing backend

- **Related work: workload translation**
  – generate suitable code for a best system
  – Musketeer [Eurosys '16], Weld [CIDR '17]

- **Related work: Multi Store / Hybrid Engines**
  – MISO [SIGMOD '14], MuSQLE [BigData '16]
  – using multiple engines/stores based on cost model / heuristics / etc.

No JVM awareness
need to update cost model / heuristics frequently

# Questions and Challenges

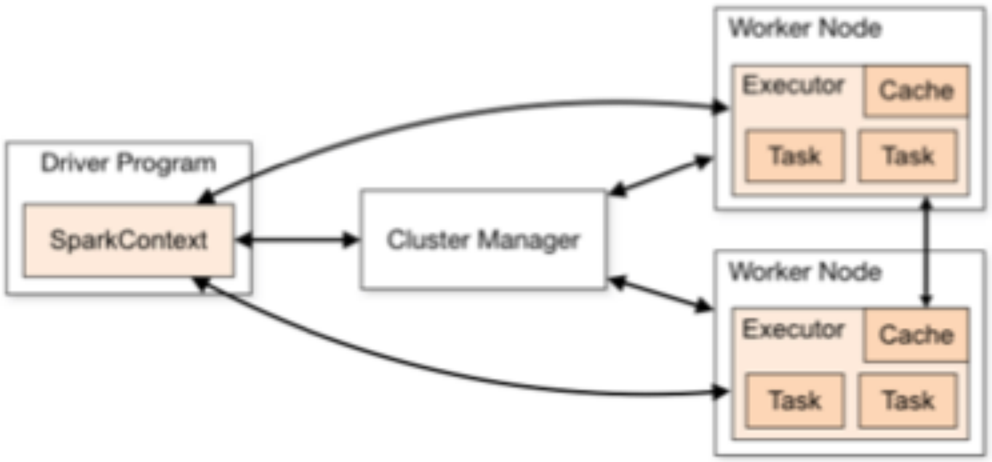


# Agenda

- **Motivation, Problems and Challenges**
- **Backgrounds**
  - Backend engines:  Spark and Tez
  - Backend runtimes: OpenJDK and J9
- **Empirical Study**
  - Performance evaluation
  - Performance analysis
- **ML model**
  - training classification model
  - evaluating classification model
- **Summary**

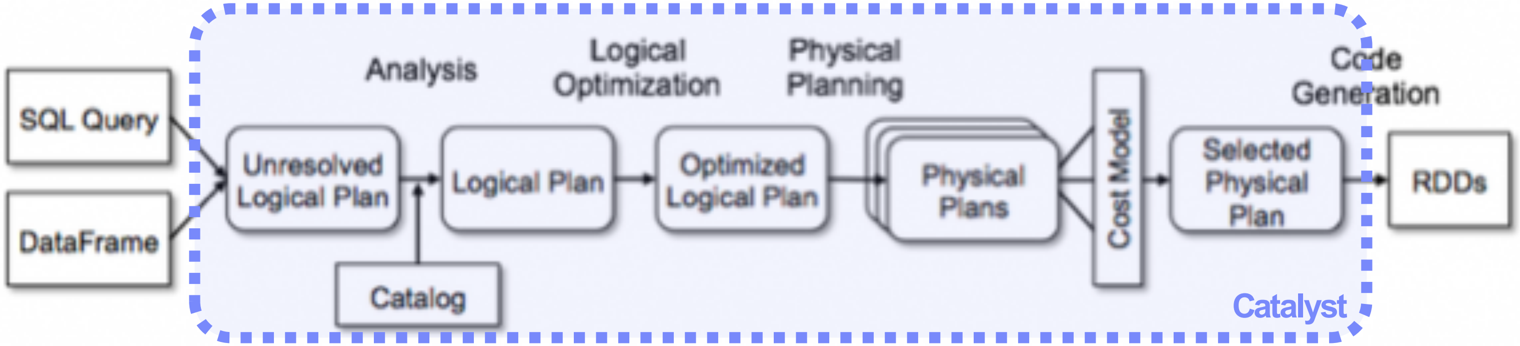
# Spark/Spark SQL

- **Spark**
  - DAG-based distributed framework
  - execute stage by stage




引用: https://spark.apache.org/docs/latest/cluster-overview.html

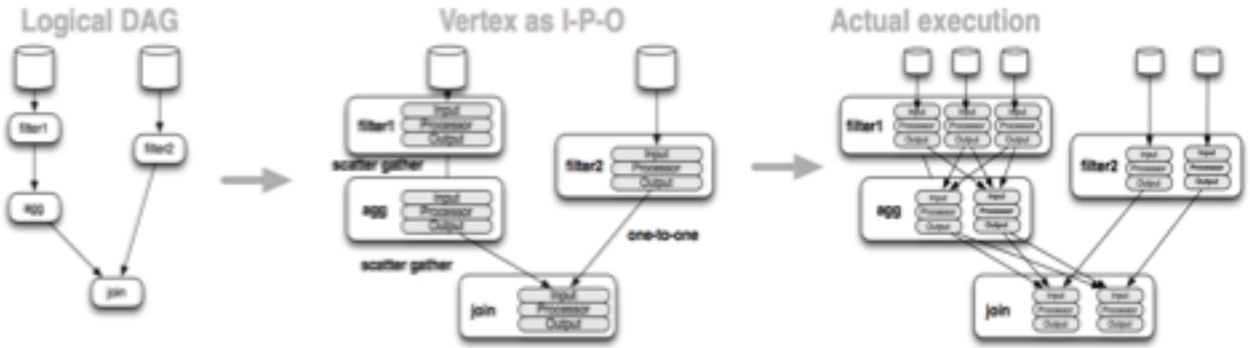
- **Spark SQL**
  - Catalyst – Query Optimizer
  - Parquet Columnar Format
  - code generation (SIMD, loop unrolling)




引用: Michael et al., Spark SQL: Relational Data Processing in Spark , *SIGMOD'15*

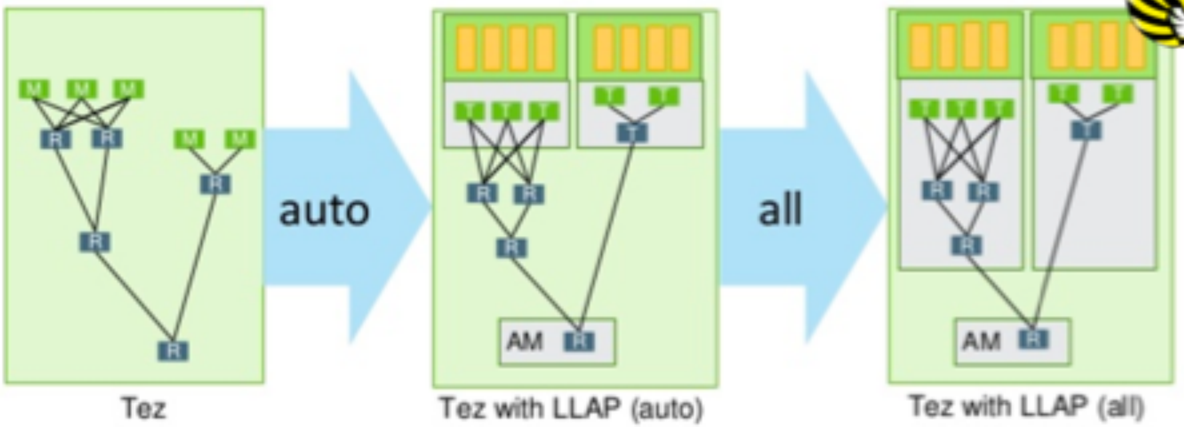
# Tez/Hive

- **Tez**
  - Generalized Map Reduce
  - DAG-based distributed framework




Ref：Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications, *SIGMOD'15*

- **Hive/LLAP**
  - focus on interactive query
  - Vectorization / Pipeline
  - In-Memory Columnar Cache (off-heap)
  - ORC Columnar Format




Ref：https://www.slideshare.net/Hadoop_Summit/llap-longlived-execution-in-hive, Hadoop Summit 2015

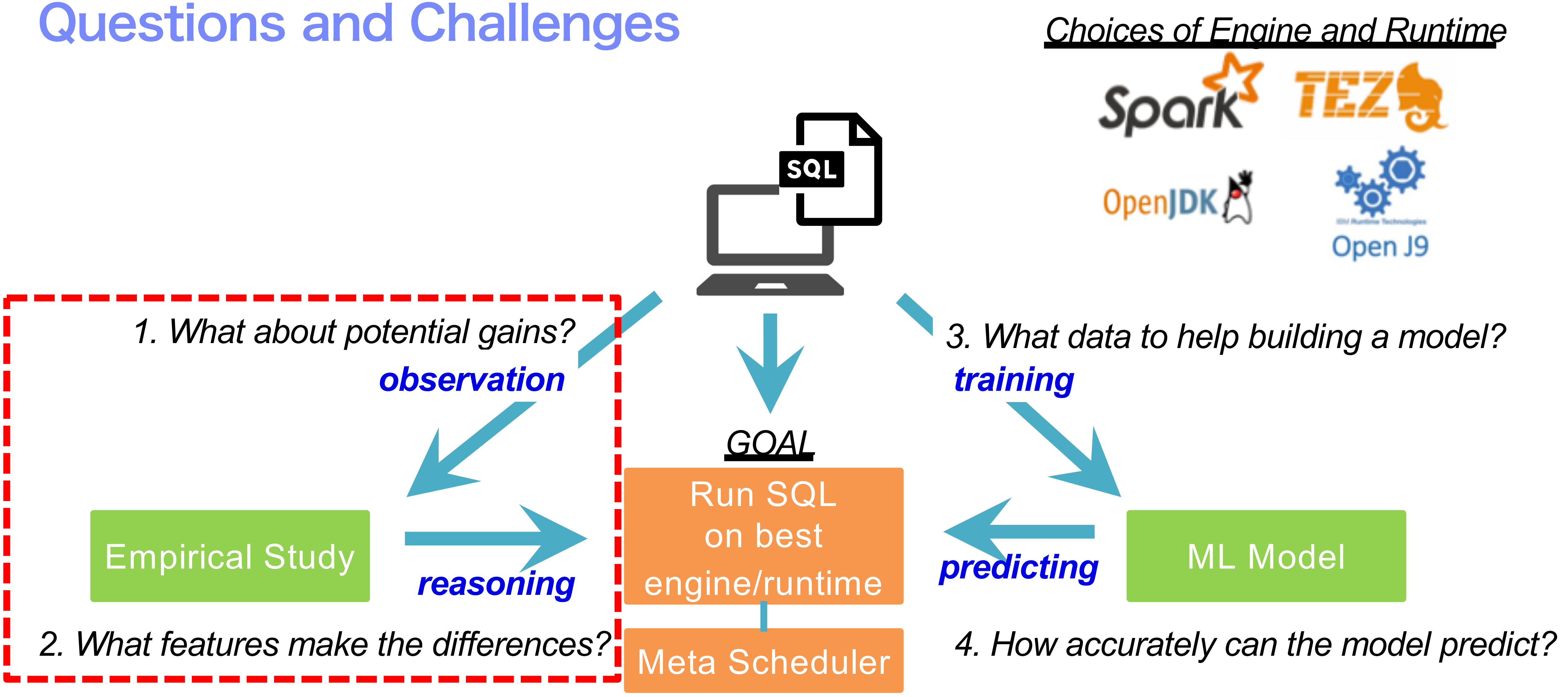
# JVM – OpenJDK & IBM J9

- **JVM**
  - OpenJDK / J9 (Eclipse OMR based)
  - internal optimization / implementation are different
- **JIT**
  - Tiered Compilation Level
  - Intrinsics
  - Inlining Heuristics
  - Vectorization Code
- **Memory Management**
  - GC Algorithm (G1GC / Generational / CMS / Parallel / Copying etc.)
  - Memory Fence
- **Thread**
  - Lock Reservation

# Agenda

- **Motivation, Problems and Challenges**
- **Backgrounds**
  - Backend engines:  Spark and Tez
  - Backend runtimes: OpenJDK and J9
- **Empirical Study**
  - Performance evaluation
  - Performance analysis
- **ML model**
  - training classification model
  - evaluating classification model
- **Summary**

# Questions and Challenges



# Environment – HW/SW Spec & Benchmark

- **Machine**
  - evaluated on a single POWER8 node
  - Use Flash storage for HDFS
- **TPC-DS Benchmark**
  - hive-testbench (*1)
  - 68 queries
- **data set**
  - Scale Factor 500 (500GB)
  - prepared two columnar dataset; Parquet & ORC

| Machine | Description |
|---|---|
| Processor | POWER8<br>3.3 GHz * 2 |
| # Cores | 24 cores<br>(2 Sockets * 12 Cores) |
| SMT | 8 |
| Memory | 1TB |
| Disk | Flash System (9.3TB) |
| OS | Ubuntu 16.04<br>(kernel 4.4.0-31) |

| Software | version |
|---|---|
| Spark | 2.1.0 |
| Hadoop (HDFS) | 2.7.2 |
| Tez | 0.9.0 |
| Hive | 2.2.0 |
| OpenJDK | 1.8.0_u121 |
| IBM J9 JVM | 1.8.0 SR4FP2 |

(*1) https://github.com/hortonworks/hive-testbench

# Environment - Others

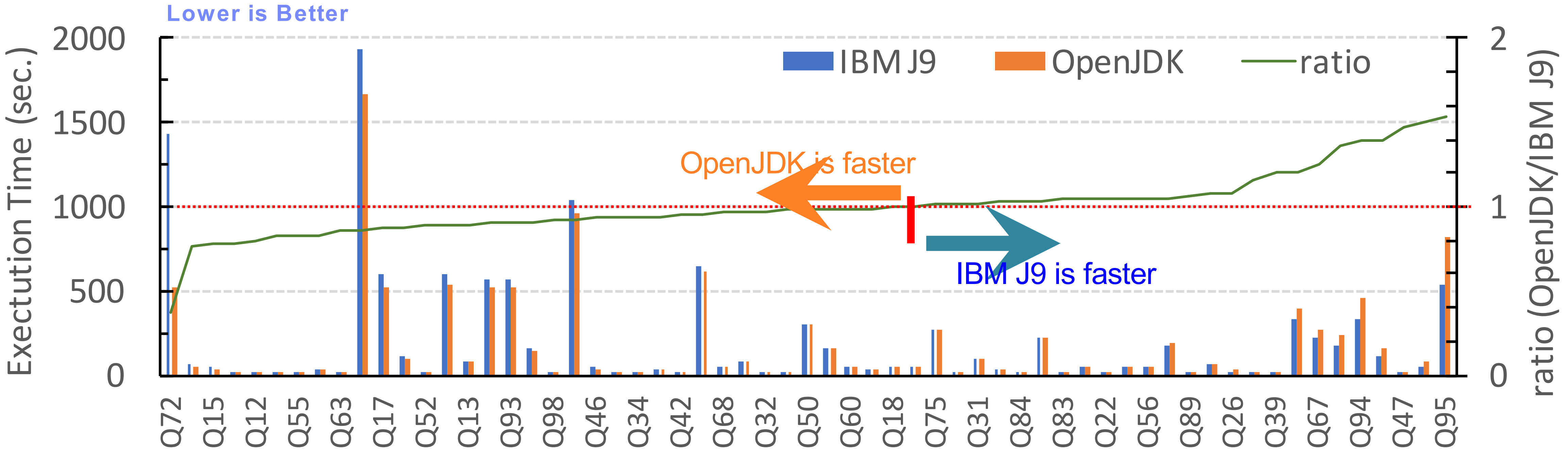
- **Configurations of Spark & Tez**

| Configuration | Spark / Spark SQL | Tez / Hive |
|---|---|---|
| Executor JVM | 1 | 1 |
| Worker Threads | 12 | 12 |
| I/O Threads | - | 12 |
| On Heap Size | 192 GB | 96 GB |
| Off Heap Size | - | 96 GB |
| Execution Mode | Daemon (Thrift Server) | LLAP Daemon |
| Columnar Format | Parquet | ORC |
| Compression Format | gzip (zlib) | gzip (zlib) |
| Other JVM Options (Common) | GC Threads = 12, -agentpath:libjvmti_oprofile.so | |

- **Evaluation Methodology**
  – used Thrift Server
  – picked up fastest result in 5-times test per query
  – reset buffer cache (`echo 3 > /proc/sys/vm/drop_caches`)

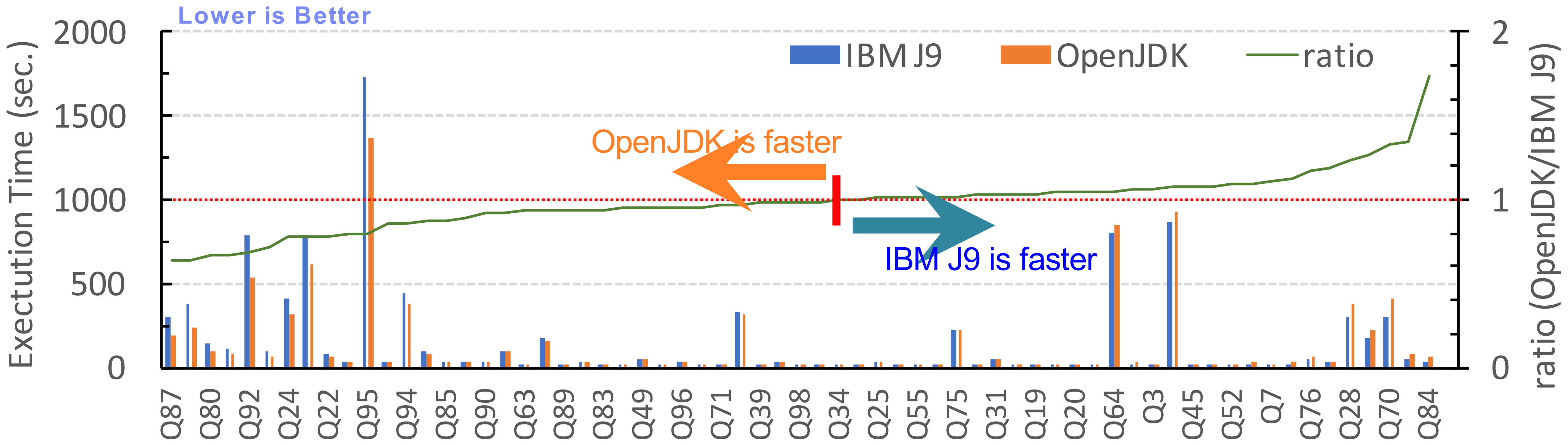
# Performance Comparison of TPC-DS on Spark - Which JVM is better for Spark?

- **Performance Comparison Result**
  - OpenJDK achieved faster than J9 in 35 queries (35/62 = 56.5%)
  - J9 achieved faster than OpenJDK in 27 queries (27/62 = 43.5%)
  - **leads up to 3x drawback**



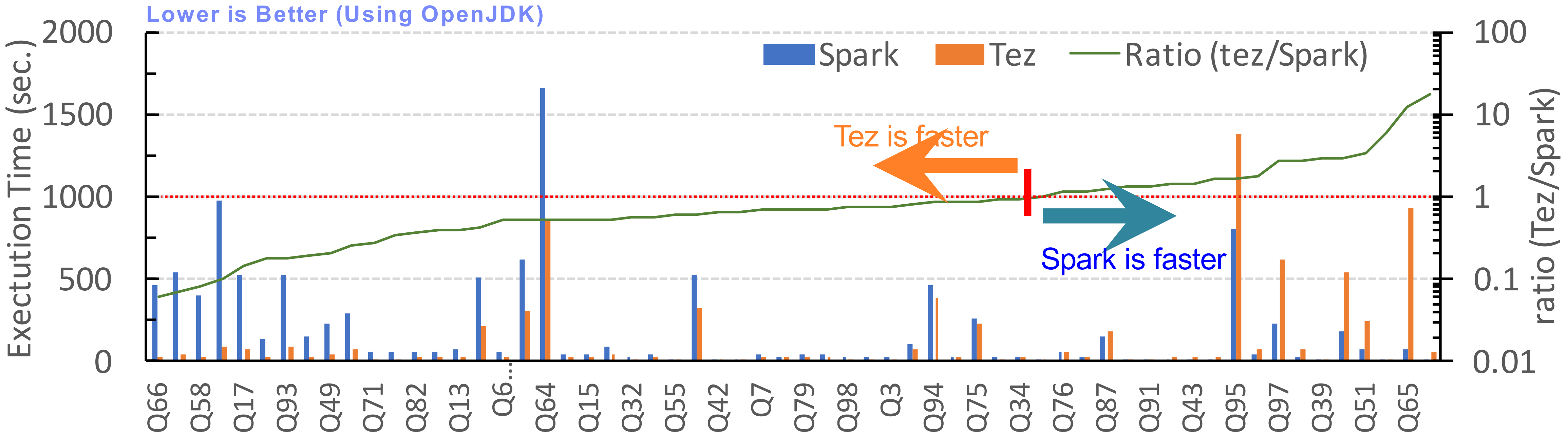
# Performance Comparison of TPC-DS on Tez - Which JVM is better for Tez?

- **Performance Comparison Result**
  - OpenJDK achieved faster than J9 in 35 queries (35/65 = 53.8%)
  - J9 achieved faster than OpenJDK in 30 queries (30/65 = 46.1%)
  - **leads up to 2x drawback**



# Performance Comparison of TPC-DS with OpenJDK - Which query engine is better with OpenJDK?

- **Performance Comparison Result**
  - Tez is faster in two-thirds queries than Spark
  - **leads up to 17x drawback**



# Summary of Motivational Evaluation

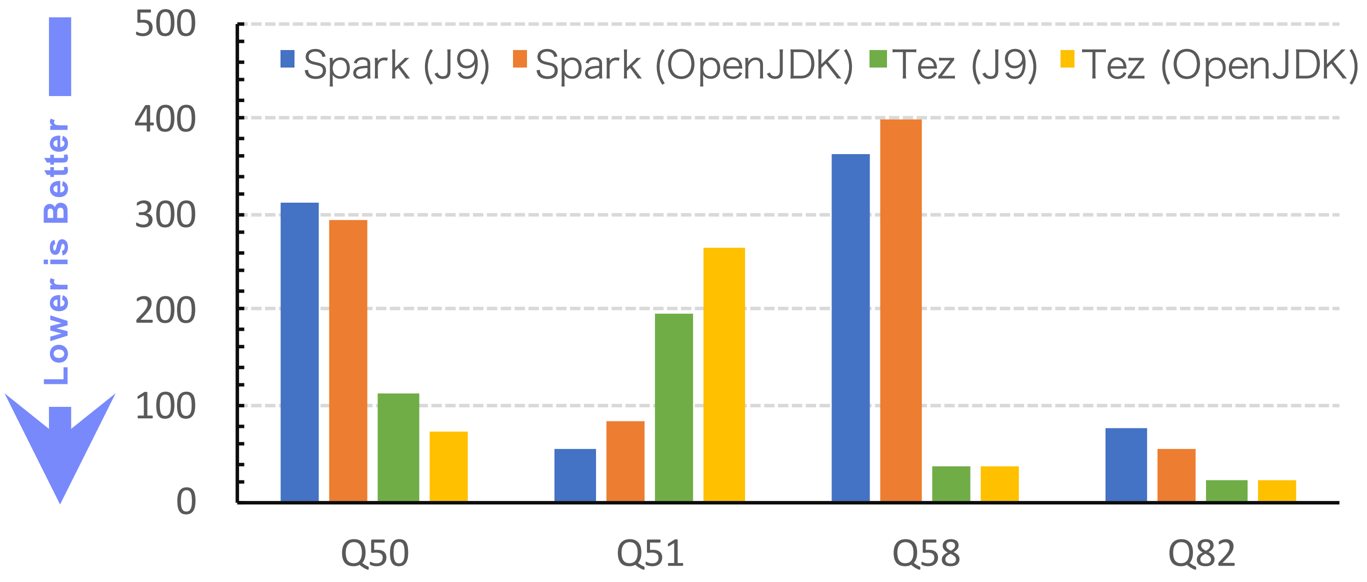
- **Result**
  - 60 queries are successfully run
  - picked up a best combination for all queries

| | IBM J9 | OpenJDK | total |
|---|---|---|---|
| Spark | 13 | 6 | 19 |
| Tez | 22 | 19 | 41 |
| total | 35 | 25 | 60 |

- **Tendency**
  - Tez is better than Spark
  - J9 is better than OpenJDK
  - Combination of Tez & J9 is good at in many cases

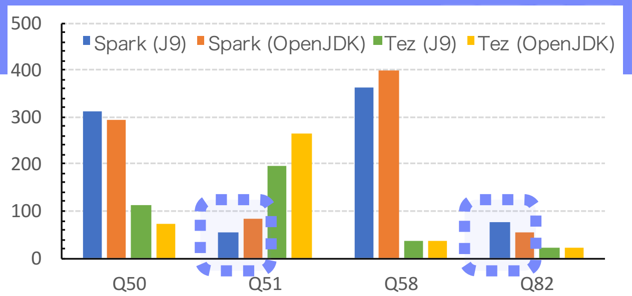
# Comparison of picked up queries

- **System**
  - Spark wins Tez : Q51
  - Tez wins Spark : Q50, Q58, Q82
- **Runtime**
  - J9 wins OpenJDK: Q51, Q58
  - OpenJDK wins J9: Q50, Q82



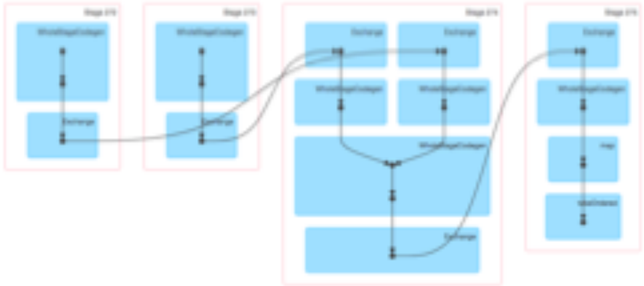

- **analysis**
  - query plan (DAG) / middleware execution stats
  - hot method profiling (oprofile) / system utilization
  - Java method stack trace / GC Log / JIT Log

# Gain comes from JVM difference – Spark case

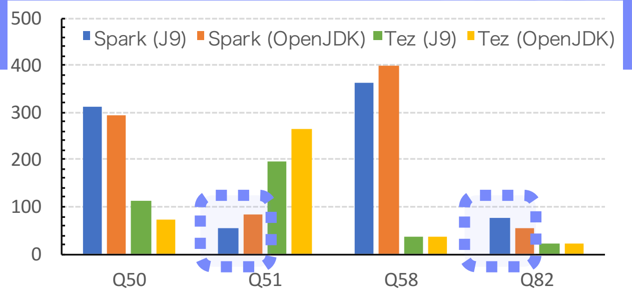



- **Q51**
  - J9 wins
  - many stages
  - less shuffle data
  - gets 2.6x gain in shuffle stage

- **Q82**
  - OpenJDK wins
  - few stages
  - much shuffle data
  - gets 1.4x gain in map stage



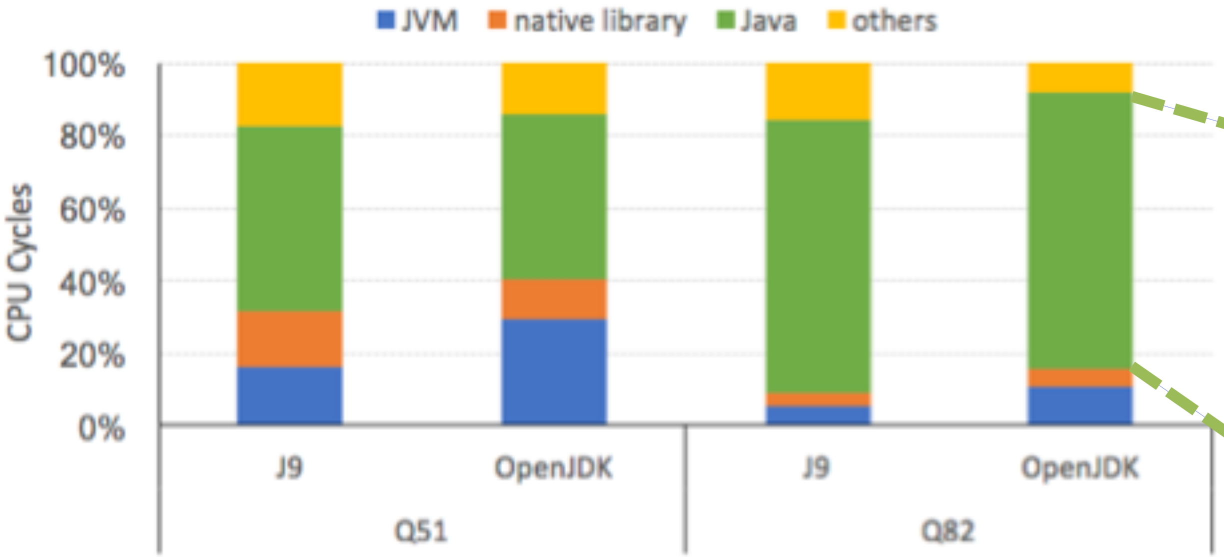
| Query | # Map Stages | # Reduce Stages | Input Read | Shuffle Output | Difference |
|---|---|---|---|---|---|
| Q51 | 2 | 6 | 6.0 GB | 1.0 GB | **Shuffle**<br>**J9: 11s OpenJDK: 29 s** |
| Q82 | 2 | 2 | 2.5 GB | 5.6 GB | **Map**<br>**J9: 66s OpenJDK: 47s** |




# Gain comes from JVM difference – Spark case

- **Method Profiling**
  - J9 is good at Intrinsic for `Sun.misc.Unsafe.copyMemory` (JNI overhead)
  - OpenJDK is good at serialization and sort in data shuffling

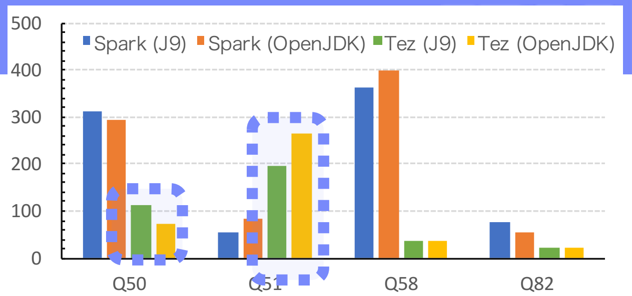
*J9 Advantage: Many Stages, less Shuffling Data*
*OpenJDK Advantage: Few Stages, much Shuffling Data*




J9 is faster

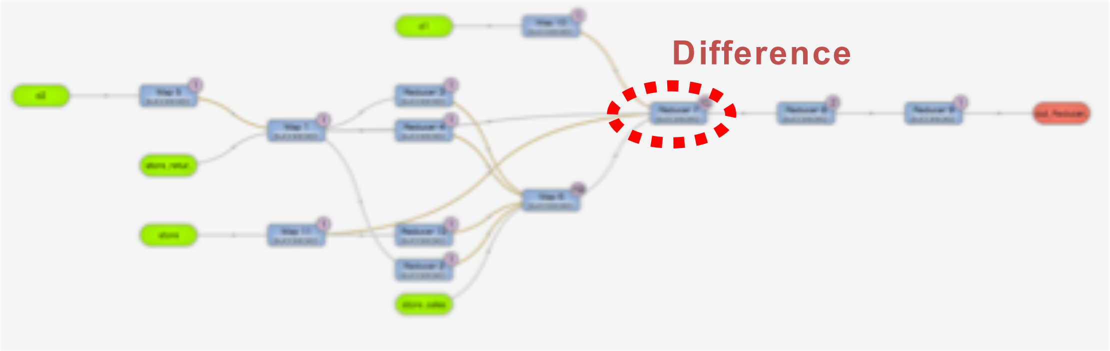
OpenJDK is faster

| | Q51 | | Q82 | |
|---|---|---|---|---|
| | J9 | OpenJDK | J9 | OpenJDK |
| catalyst | 25.3% | 13.2% | 15.7% | 13.0% |
| sql core | 3.49% | 3.65% | 24.4% | 13.5% |
| sql sort | 0.41% | 0.29% | 10.3% | 5.2% |
| crc32c | 2.5% | 2.0% | 0.59% | 0.9% |
| lz4 | 3.0% | 3.6% | 6.9% | 14.5% |
| Unsafe | 0.0001% | 2.0% | 0.0003% | 2.2% |



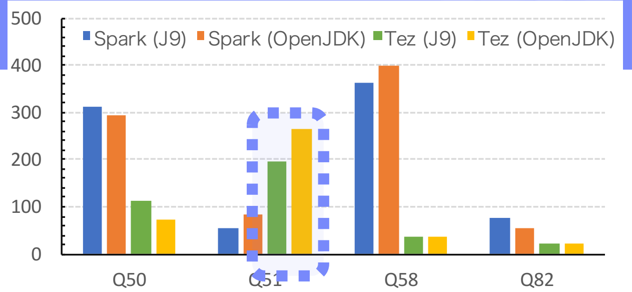

# Gain comes from JVM difference – Tez case

- Q50
  - OpenJDK wins
  - gets 1.7x gain in reduce vertex
- Q51
  - J9 wins
  - gets 3x gain in map vertex





| Query | # Map Stages | # Reduce Stages | Input Records / GB | Shuffle Records / GB | Difference |
|---|---|---|---|---|---|
| Q50 | 5 | 7 | 1.3 * 10^9 (5.4 GB) | 5.0 * 10^7 (1.9 GB) | Reduce J9: 106s, OpenJDK: 60s |
| Q51 | 4 | 5 | 3.5 * 10^8 (1.3 GB) | 3.5 * 10^8 (3.5 GB) | Map J9: 7s, OpenJDK: 21s |




# Gain comes from JVM difference – Tez case

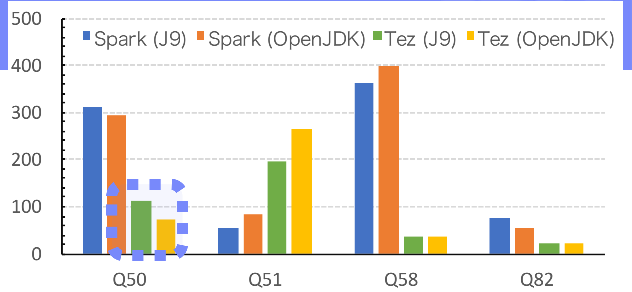
- Q51
  - J9 achieved 3x gain in map vertex
  - writing intermediate data (including in-mem agg. & SerDe) is time-consuming

*J9 Advantage: Few Vertices, Much Shuffling Data*

| | | Q50 | | Q51 | |
|---|---|---|---|---|---|
| | | J9 | OpenJDK | J9 | OpenJDK |
| | kallsyms | 18.4% | 6.5% | 4.1% | 3.3% |
| | jvm | 9.1% | 25.9% | 11.2% | 7.0% |
| | java | 68.6% | 61.0% | 81.5% | 87.0% |
| PipelinedSorter (Reduce Vertex) ・・・ | tez | 3.1% | 2.6% | 28.9% | 12.0% |
| In memory ORC (LLAP) Read ・・・ | orc | 11.9% | 17.0% | 0.8% | 0.3% |
| java.io.DataOutputStream ・・・ | java.io | 1.0% | 4.0% | 6.0% | 18.3% |
| JOIN / Aggregation ・・・ | hive.ql | 38.0% | 30.7% | 16.0% | 9.3% |
| Serialization/Deserialization ・・・ | serDe | 2.3% | 1.3% | 12.4% | 29.0% |

Serialize + Spill

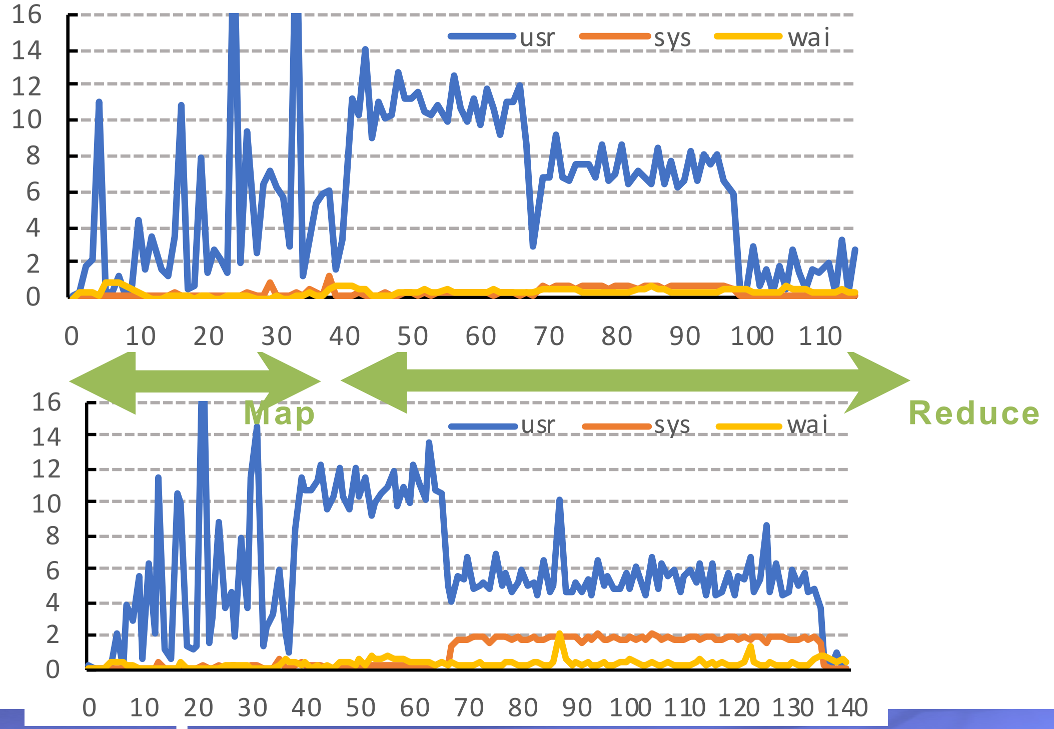
# Gain comes from JVM difference – Tez case




- Q50
  - OpenJDK achieved **1.7x gain** in reduce vertex
  - many shuffle threads / many vertices
  - huge context switch overhead

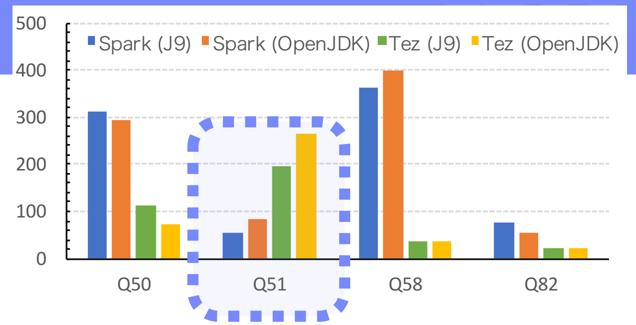
*OpenJDK Advantage: Many Vertices, Less Shuffling Data*

OpenJDK



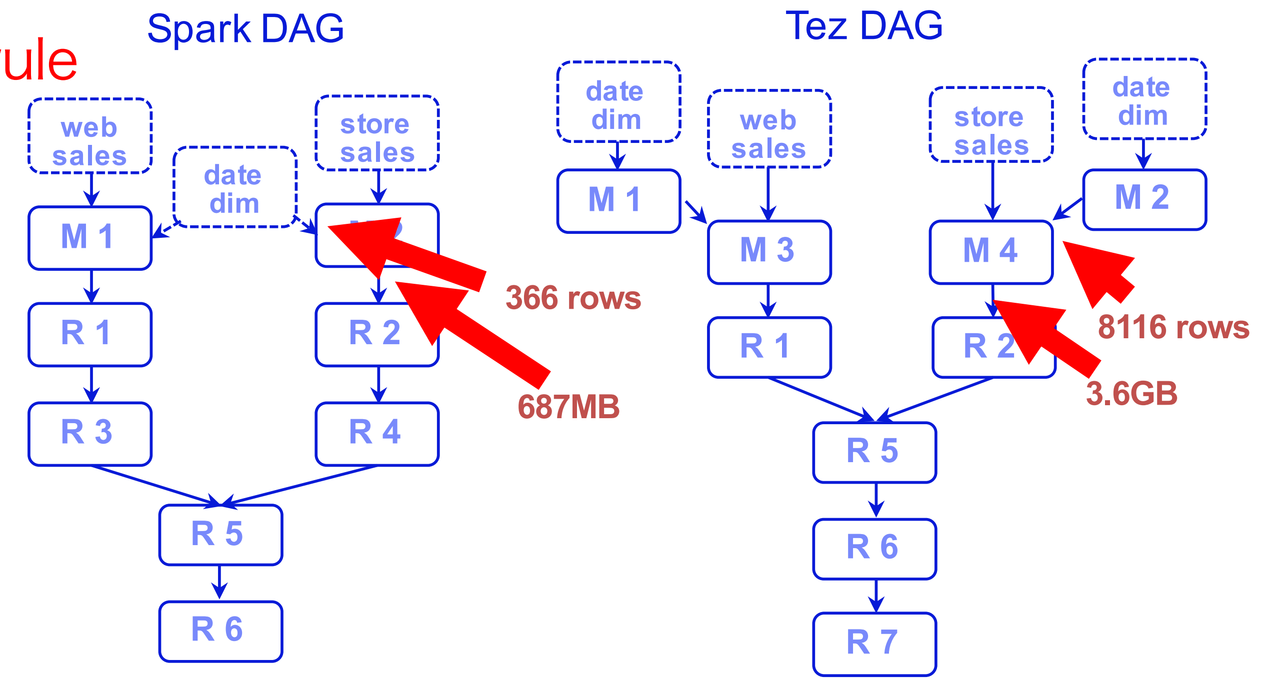

J9

| | Q50 | | Q51 | |
|---|---|---|---|---|
| | J9 | OpenJDK | J9 | OpenJDK |
| kallsyms | 18.4% | 6.5% | 4.1% | 3.3% |
| jvm | 9.1% | 25.9% | 11.2% | 7.0% |
| java | 68.6% | 61.0% | 81.5% | 87.0% |
| tez | 3.1% | 2.6% | 28.9% | 12.0% |
| orc | 11.9% | 17.0% | 0.8% | 0.3% |
| java.io | 1.0% | 4.0% | 6.0% | 18.3% |
| hive.ql | 38.0% | 30.7% | 16.0% | 9.3% |
| serDe | 2.3% | 1.3% | 12.4% | 29.0% |




# Gain comes from query engine difference – Spark or Tez

- **Spark Advantage (Q51)**
  - reduce shuffling data by better filtering rule
  - Spark: 366 rows → shuffling 687MB
  - Tez: 8,116 rows → shuffling 3.6GB




- **Tez Advantage (Q50, Q58, Q82)**
  - reduce shuffling data by Bloom Filter

*Good query optimizer (Cost Based Optimizer) helps to reduce shuffling data*

# Empirical Study Summary - What features affect the performance

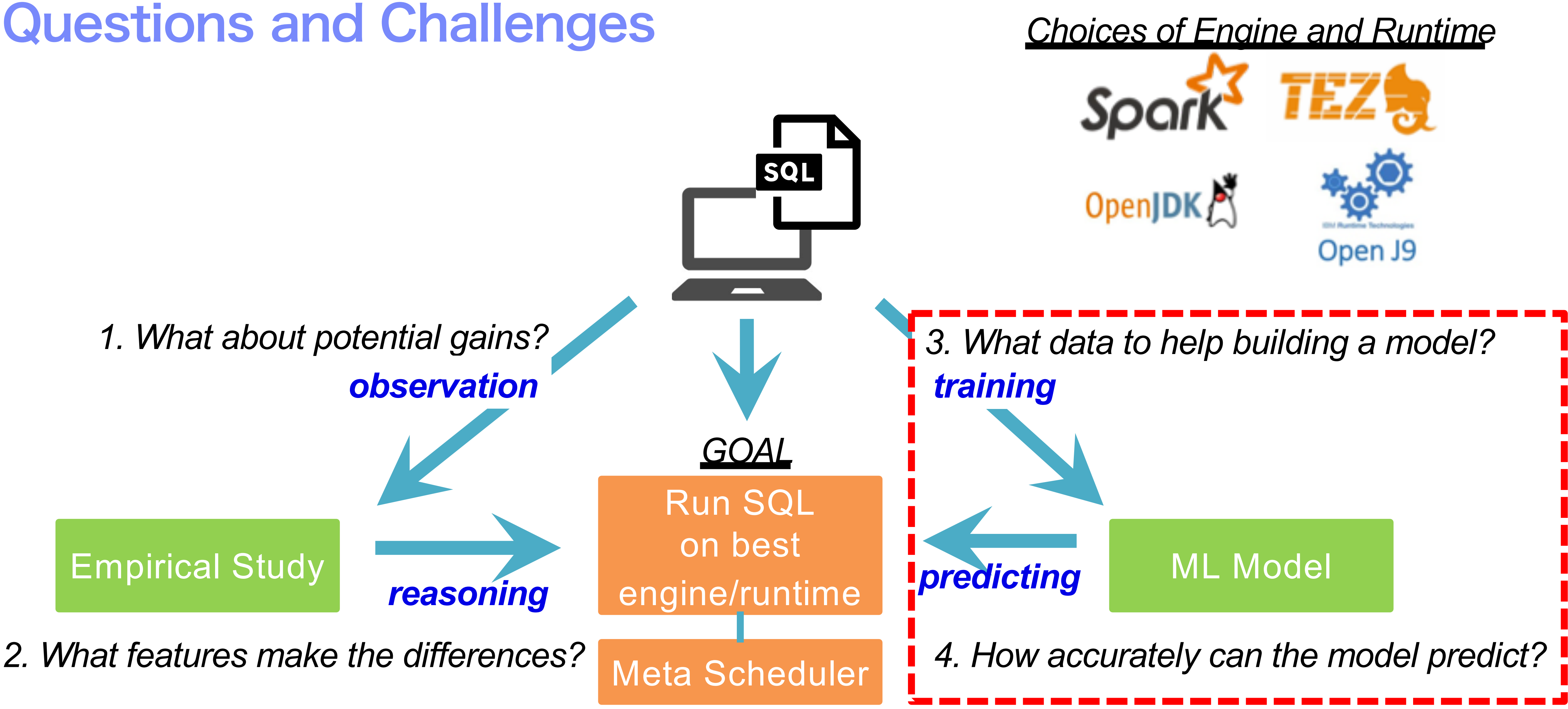
- **Query Engine**
  - DAG
    - # of Vertices / Stages (Map or Reduce)
    - amount of shuffling data (intermediate data)
    - input data size (tables)
  - CBO
    - filtering rule

- **JVM**
  - # of threads
  - Intrinsic
  - SerDe performance
  - I/O performance

# Agenda

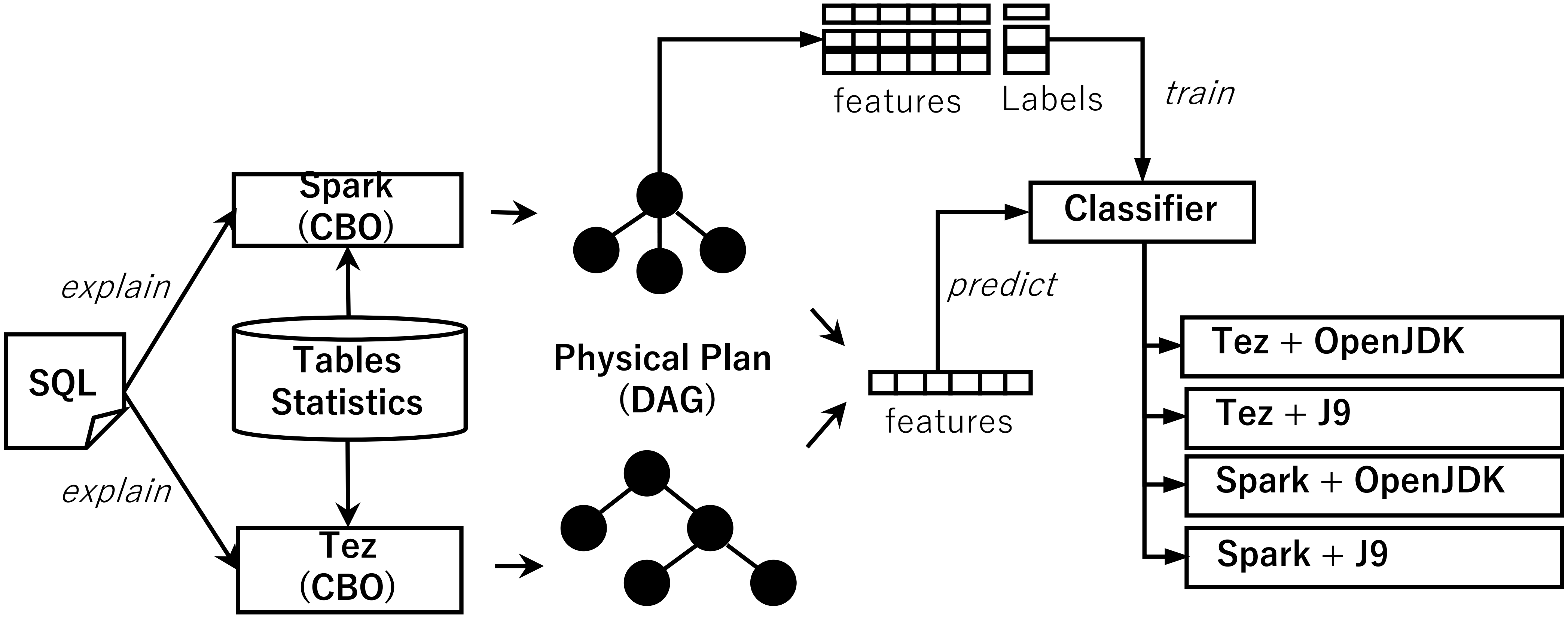
- **Motivation, Problems and Challenges**
- **Backgrounds**
  – Backend engines:  Spark and Tez
  – Backend runtimes: OpenJDK and J9
- **Empirical Study**
  – Performance evaluation
  – Performance analysis
- **ML model**
  – training classification model
  – evaluating classification model
- **Summary**

# Questions and Challenges



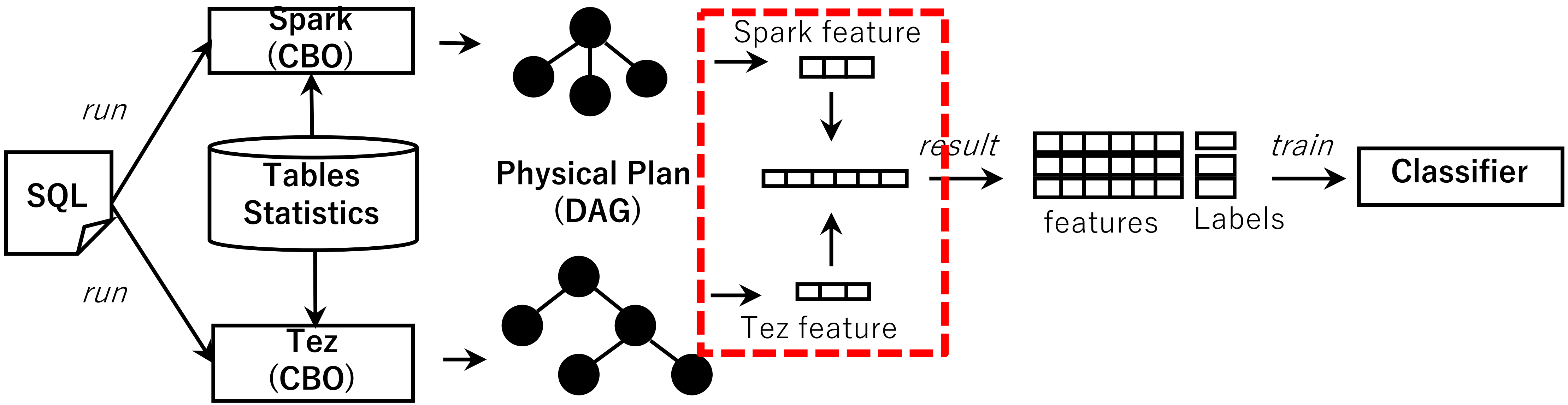
# Proposed Classifier Overview - Training and Prediction

- **Key points**
  - Making classifier model based on the features that come from DAG
  - Selecting a combination of the system based on the model before query execution



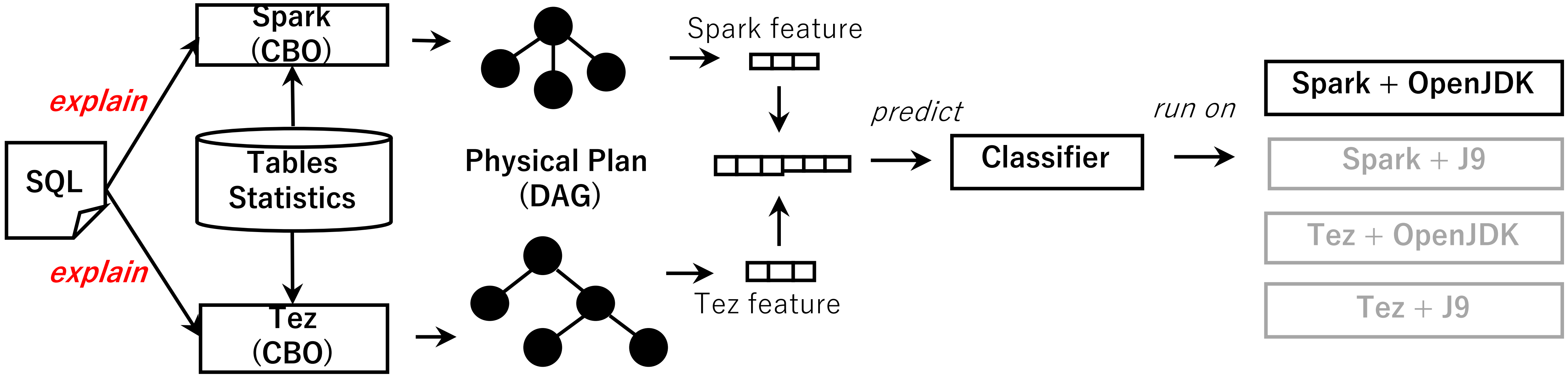
# Training Classifier

- **Why extract features from DAG? Why not SQL?**
  - contains much more info including table stats/actual stages than SQL
- **What features are used**
  - # of stages, # of joins, join types, used tables, etc.
  - 69 features in total



# Predicting best combination using classifier

- **Extract features without actual query run**
  - sql explain generates DAG (compiling it in 2-5 sec)
- **Predict best system for the query**
  - decide a combination based on the classifier



# Evaluation of classifier

- **Training and testing four ML algorithms**
  - kNN, Decision Tree, SVM, Random Forest
  - k-fold cross-validation (split data into 80:20)
- **Models**
  - binary class: Spark or Tez
  - multi class: Spark/OpenJDK or Spark/J9 or Tez/OpenJDK or Tez/J9

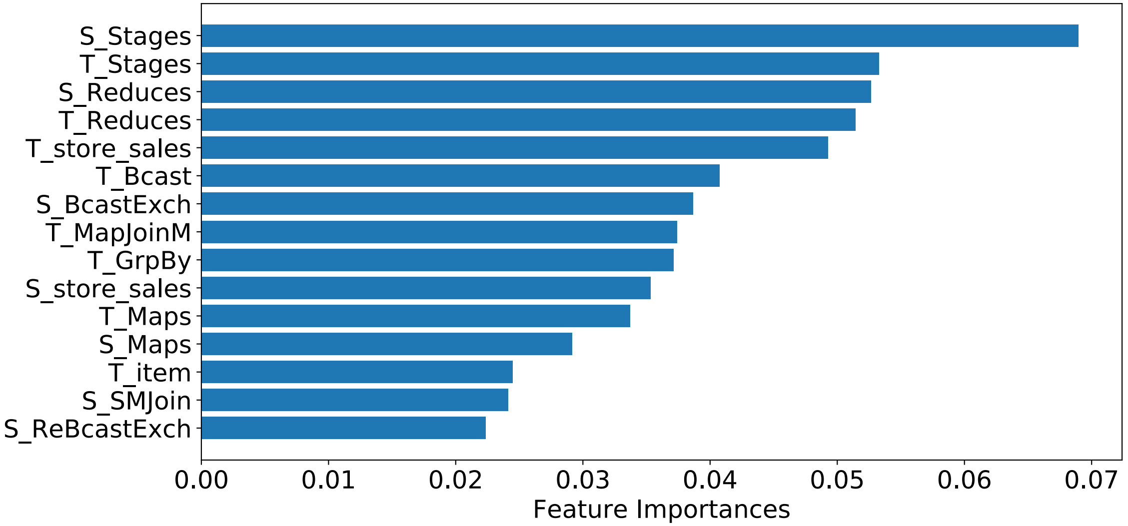
**Accuracy**

| | binary classifier | | multinomial classifier | |
|---|---|---|---|---|
| | mean | stddev (+/-) | mean | stddev (+/-) |
| kNN | 0.65 | 0.08 | 0.43 | 0.04 |
| Decision Tree | 0.69 | 0.17 | 0.34 | 0.16 |
| SVM | 0.72 | 0.10 | 0.39 | 0.08 |
| Random Forest | 0.72 | 0.02 | 0.46 | 0.06 |

*Random Forest is better than others*

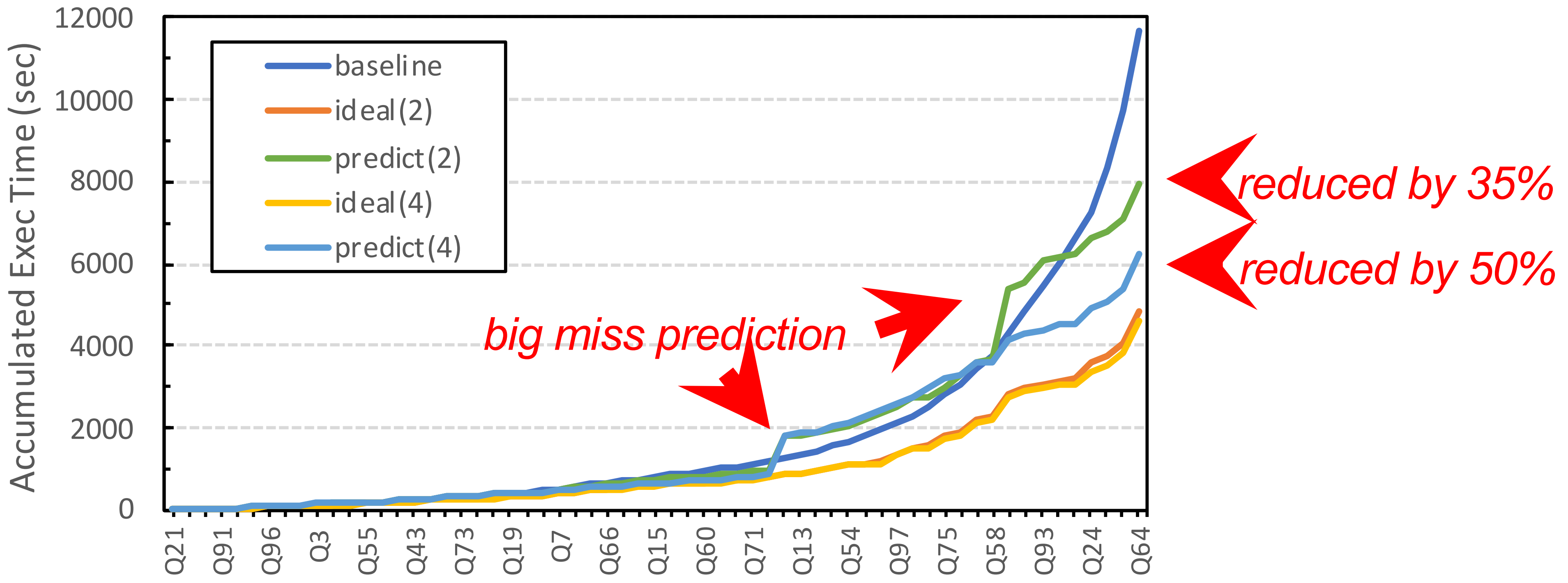
*- # of stages makes impact to the model*
*- Using BF or Join types do not affect*

**Features Impact in Random Forest**



# Evaluation of classifier

- **training and testing model**
  - k-fold cross-validation except test query feature
- **Result**
  - baseline: exec time with Spark/J9 only
  - ascending order



# Summary and Future Works

- **Summary**
  - No single query engine and JVM is best for all queries
  - query engine mismatch leads up to 10x drawback
  - JVM mismatch also leads up to 3x drawback
  - Proposed Random Forest based classifier achieved 50% time reduction in total

- **Future Works**
  - implements meta scheduler
  - applies it on Cloud/Container/Kubernetes environment
  - training data augmentation