PingCAP

When Apache Spark meets TiDB

=> TiSpark

maxiaoyu@pingcap.com

TiDB | COMMUNITY

Who am I

- Shawn Ma@PingCAP
- Tech Lead of OLAP Team
- Working on OLAP related products and features
- Previously tech lead of Big Data infra team@Netease
- Focus on SQL on Hadoop and Big Data related stuff

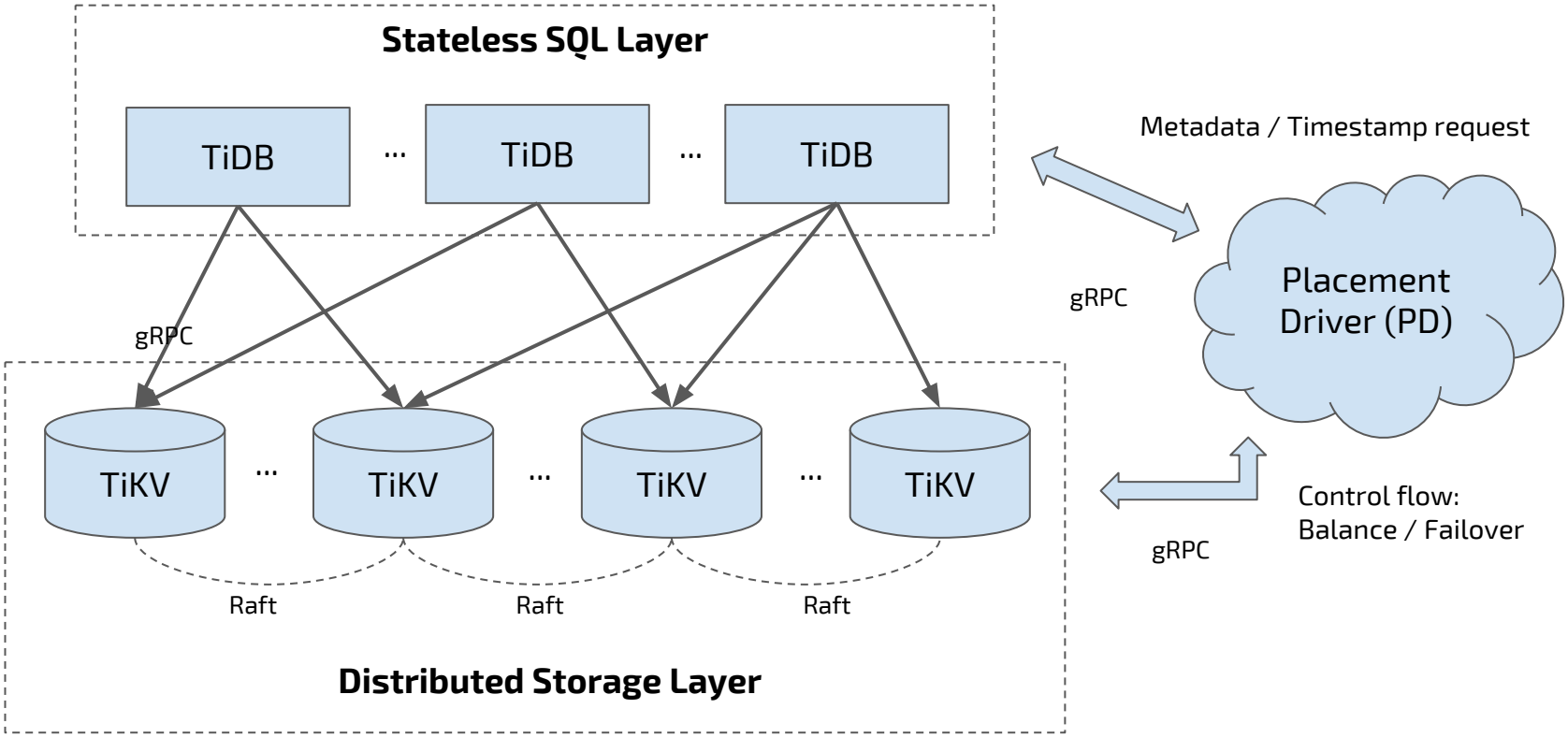
Agenda

- A little bit about TiDB / TiKV
- What is TiSpark
- Architecture
- Benefit
- What's Next

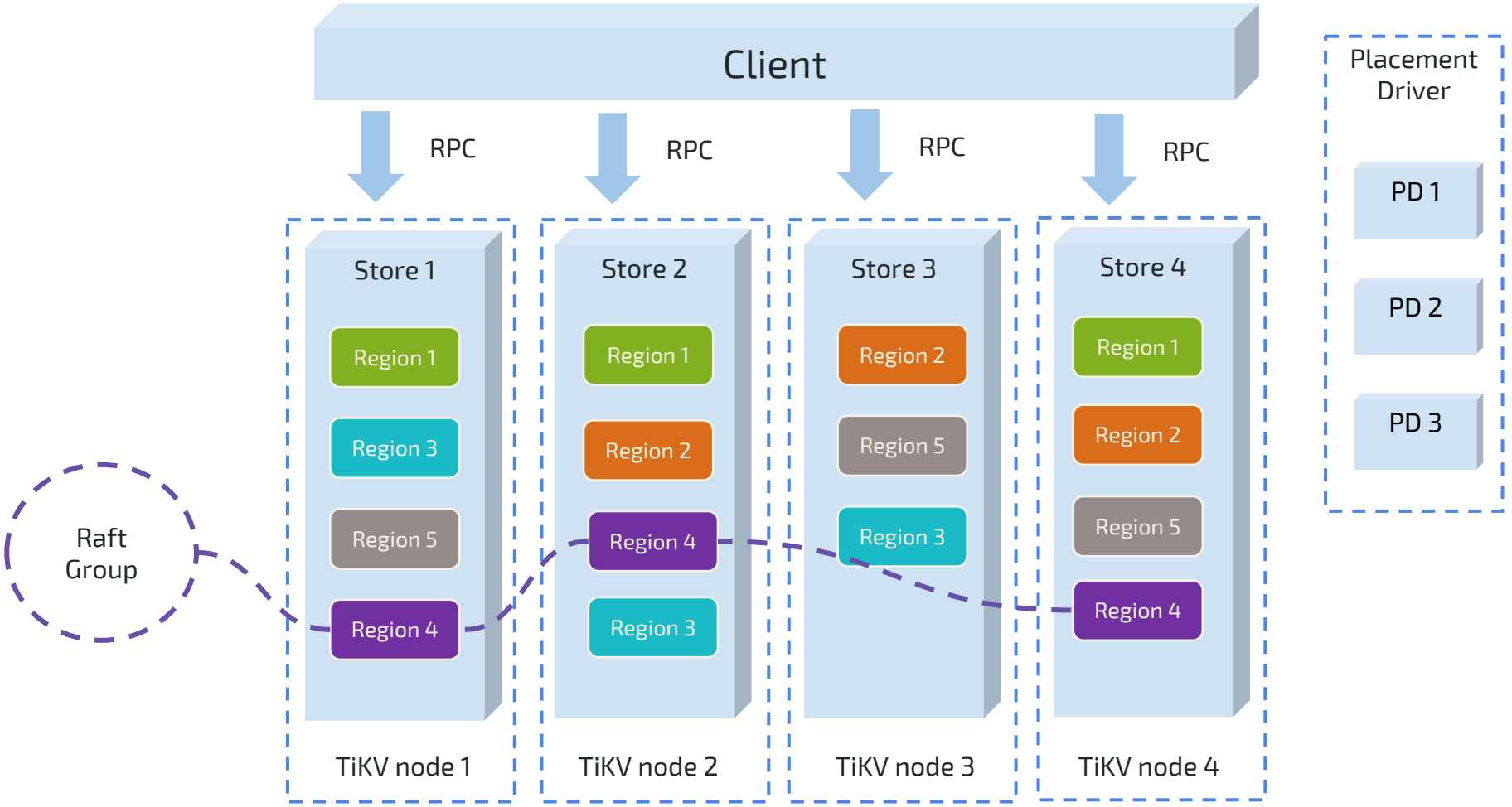
What's TiDB

- Open source distributed RDBMS
- Inspired by Google Spanner
- Horizontal Scalability
- ACID Transaction
- High Availability
- Auto-Failover
- SQL at scale
- Widely used in different industries, including Internet, Gaming, Banking, Finance, Manufacture and so on (200+ users)

A little bit about TiDB and TiKV



TiKV: The whole picture



TiKV is powered by RocksDB

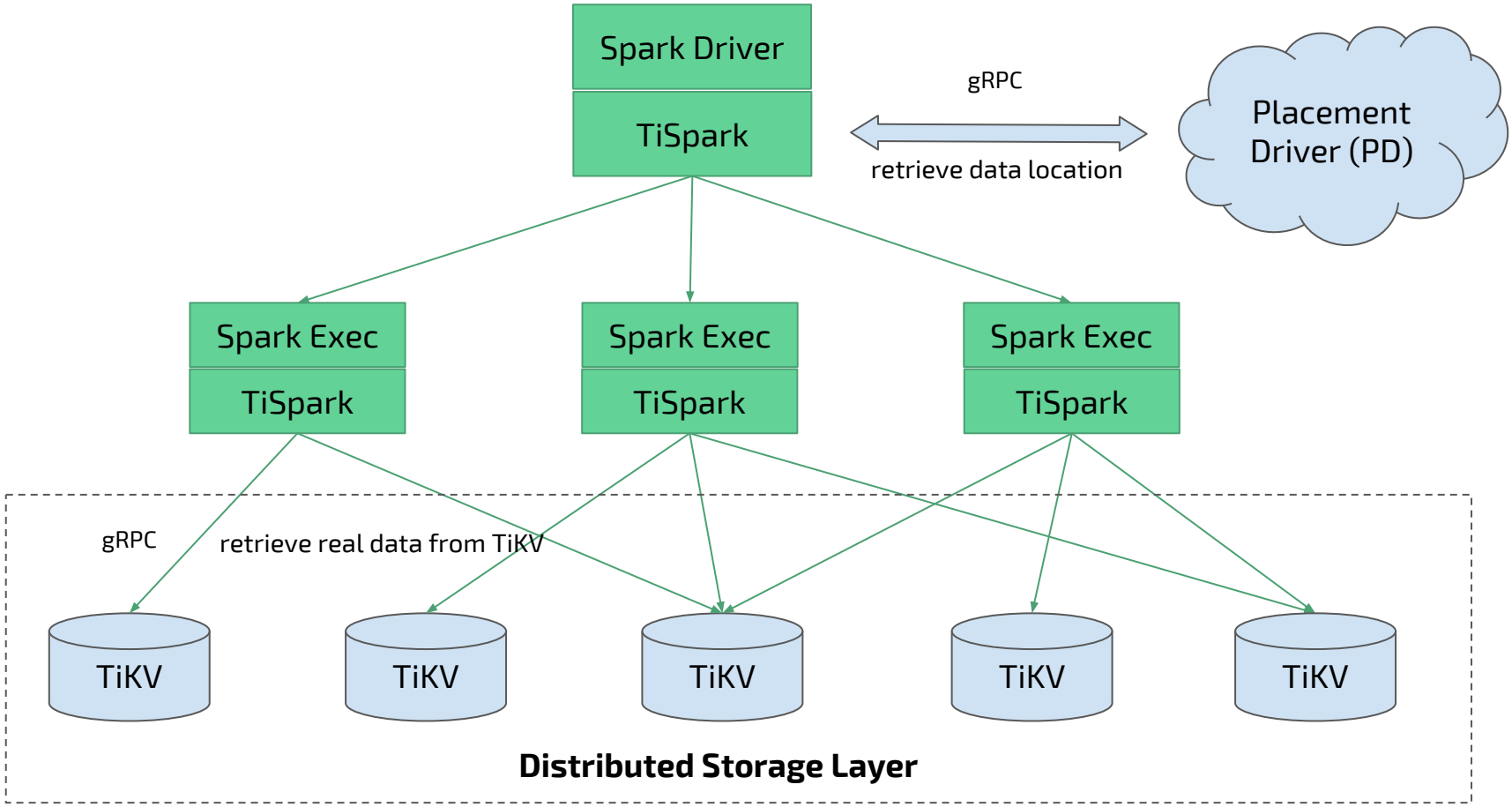
What is TiSpark

- TiSpark = Spark SQL on TiKV
  - Spark SQL directly on top of a distributed Database Storage
- Hybrid Transactional/Analytical Processing (HTAP) rocks
  - Provide strong OLAP capacity together with TiDB

What is TiSpark

- Complex Calculation Pushdown
- Key Range pruning
- Index support
  - Clustered index / Non-Clustered index
  - Index Only Query
- Cost Based Optimization
  - Histogram
  - Pick up right Access Path

Architecture



Architecture

- On Spark Driver
  - Translate metadata from TiDB into Spark meta info
  - Transform Spark SQL logical plan, pick up elements to be leverage by storage (TiKV) and rewrite the plan
  - Locate Data based on Region info from Placement Driver and split partitions;
- On Spark Executor
  - Encode Spark SQL plan into TiKV's coprocessor request
  - Decode TiKV / Coprocessor result and transform result into Spark SQL Rows

How everything made possible

public class **ExperimentalMethods**
extends Object

Holder for experimental methods for the bravest. We make NO guarantee about the stability regarding binary compatibility and source compatibility of methods here.

```
spark.experimental.extraStrategies += ...
```

Since:

1.3.0

Method Summary

Methods

Modifier and Type	Method and Description
scala.collection.Seq<org.apache.spark.sql.catalyst.rules.Rule<org.apache.spark.sql.catalyst.plans.logical.LogicalPlan>>	extraOptimizations()
scala.collection.Seq<org.apache.spark.sql.execution.SparkStrategy>	extraStrategies() Allows extra strategies to be injected into the query planner at runtime.

- Extension points for Spark SQL Internal
- Extra Strategies allow us to inject our own physical executor and that's what we leveraged for TiSpark
- Trying best to keep Spark Internal untouched to avoid compatibility issue

How everything made possible

- A fat java client module, paying the price of bypassing TiDB
  - Parsing Schema, Type system, encoding / decoding, coprocessor
  - Almost full featured TiKV client (without write support for now)
  - Predicates / Index - Key Range related logic
  - Aggregates pushdown related
  - Limit, Order, Stats related
- A thin layer inside Spark SQL
  - TiStrategy for Spark SQL plan transformation
  - And other utilities for mapping things from Spark SQL to TiKV client library
  - Physical Operators like IndexScan
  - Thin enough for not bothering much of compatibility with Spark SQL

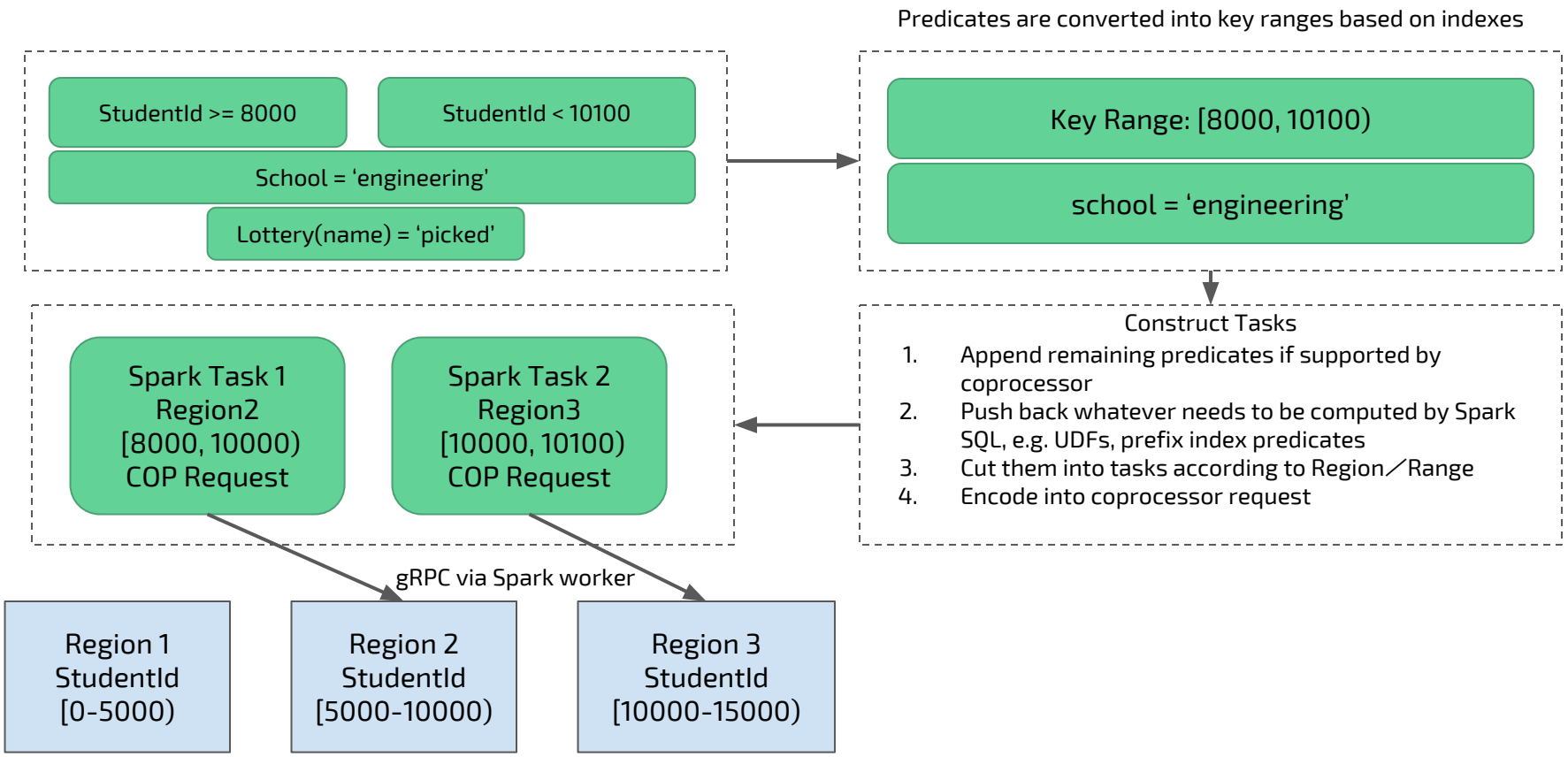
Too Abstract? Let's get concrete

```
select class, avg(score) from student
WHERE school = 'engineering'  and lottery(name) = 'picked'
and studentId >= 8000 and studentId < 10100
 group by class ;
```

- Above is a table on TiDB named student
- Clustered index on StudentId and a secondary index on School column
- Lottery is an Spark SQL UDF which pick up a name and output 'picked' if RNG decided so

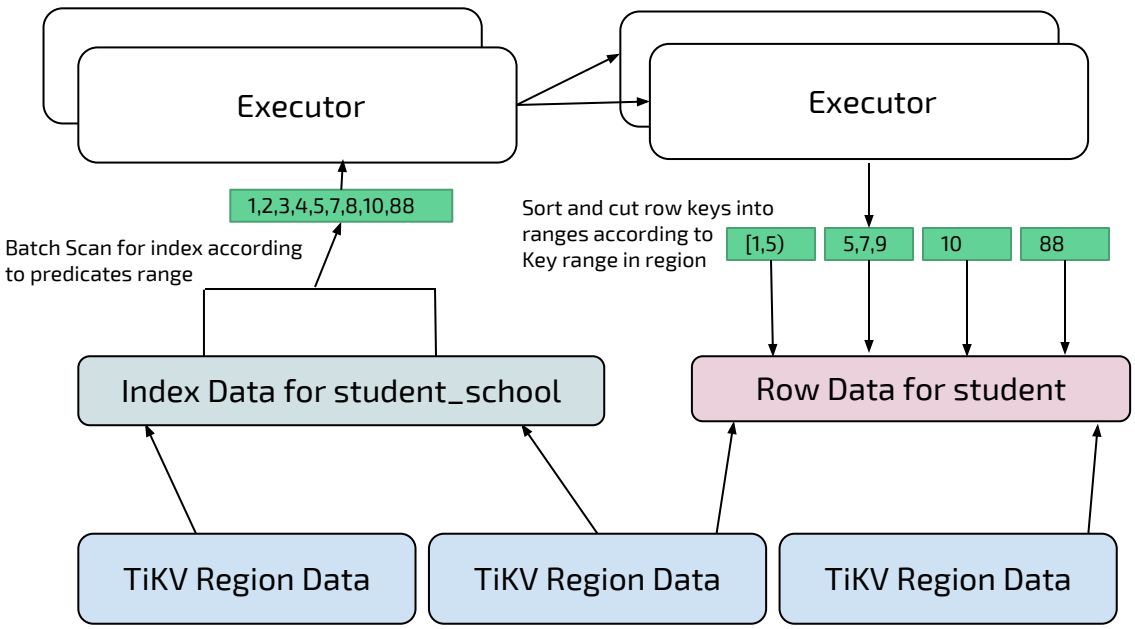
Predicates Processing

WHERE school = 'engineering' and lottery(name) = 'picked'
and studentId >= 8000 and studentId < 10100



Index Scan

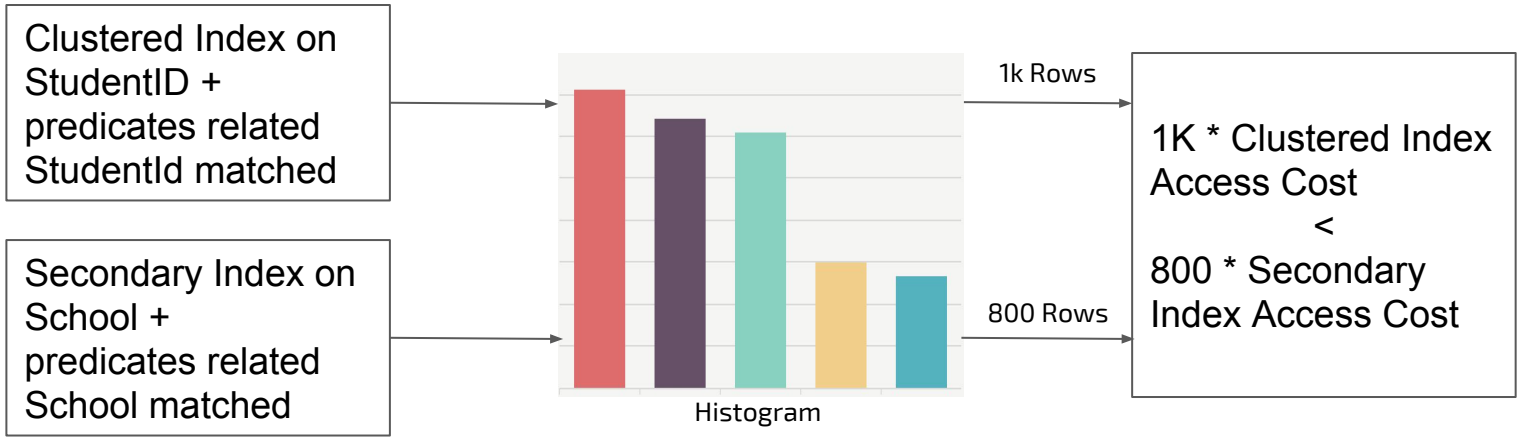
WHERE school = 'engineering' and lottery(name) = 'picked'
and (studentId >= 8000 and studentId < 10100)



- Secondary Index is encode as key-value pair
  - Key is comparable bytes format of all index keys in defined order
  - Value is the row ID pointing to table row data
- Reading data via Secondary Index usually requires a double read.
  - First, read secondary index in range just like reading primary keys in previous slide.
  - Shuffle Row IDs according to region
  - Sort all row IDs retrieved and combine them into ranges if possible
  - Encoding row IDs into row keys for the table
  - Send those mini requests in batch concurrently
- Optimize away second read operation
  - If all required column covered by index itself already

Index Selection

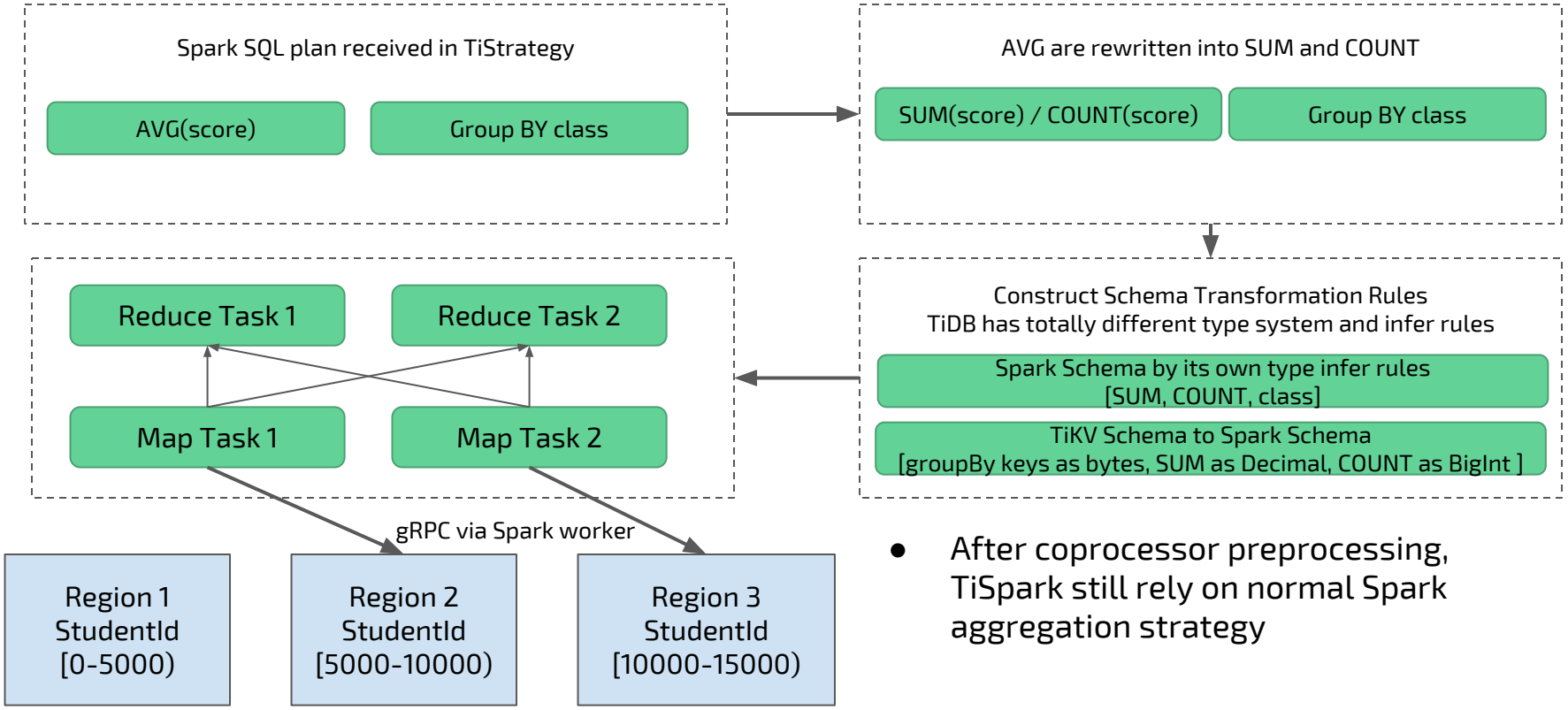
WHERE school = 'engineering' and lottery(name) = 'picked' and (studentId >= 8000 and studentId < 10100) or studentId in (10323, 10327)



- If the columns referred are all covered by index, then instead of retrieving actual rows, we apply index only query and cost function will be different
- If histogram not exists, TiSpark using pseudo selection logic.

Aggregates Processing

```
select class, avg(score) from student
.....
 group by class ;
```



- After coprocessor preprocessing, TiSpark still rely on normal Spark aggregation strategy

Benefit

- Analytical / Transactional support all on one platform
  - No need for ETL and query data in real-time
  - High throughput and consistent snapshot read from database
  - Simplify your platform and reduce maintenance cost
- Embrace Apache Spark and its eco-system
  - Support of complex transformation and analytics beyond SQL
  - Cooperate with other projects in eco-system (like Apache Zeppelin)
  - Apache Spark bridges your data sources

Ease of Use

- Working on your existing Spark Cluster
  - Just a single jar like other Spark connector
- Workable as standalone application, spark-shell, thrift-server, pyspark and R
- Work just like another data source

```
val ti = new org.apache.spark.sql.TiContext(spark)

// Map all TiDB tables from database tpch as Spark SQL tables
ti.tidbMapDatabase("sampleDB")

spark.sql("select count(*) from sampleTable").show
```

What's Next

- Batch Write Support (writing directly as TiKV native format)
- JSON Type support (since TiDB already supported)
- Partition Table support (both Range and Hash)
- Join optimization based on range and partition table
- (Maybe) Join Reorder with TiDB's own Histogram
- Another separate columnar storage project using Spark as its execution engine (not released yet)

Thanks!

Contact me:

maxiaoyu@pingcap.com
www.pingcap.com

https://github.com/pingcap/tispark
https://github.com/pingcap/tidb
https://github.com/pingcap/tikv