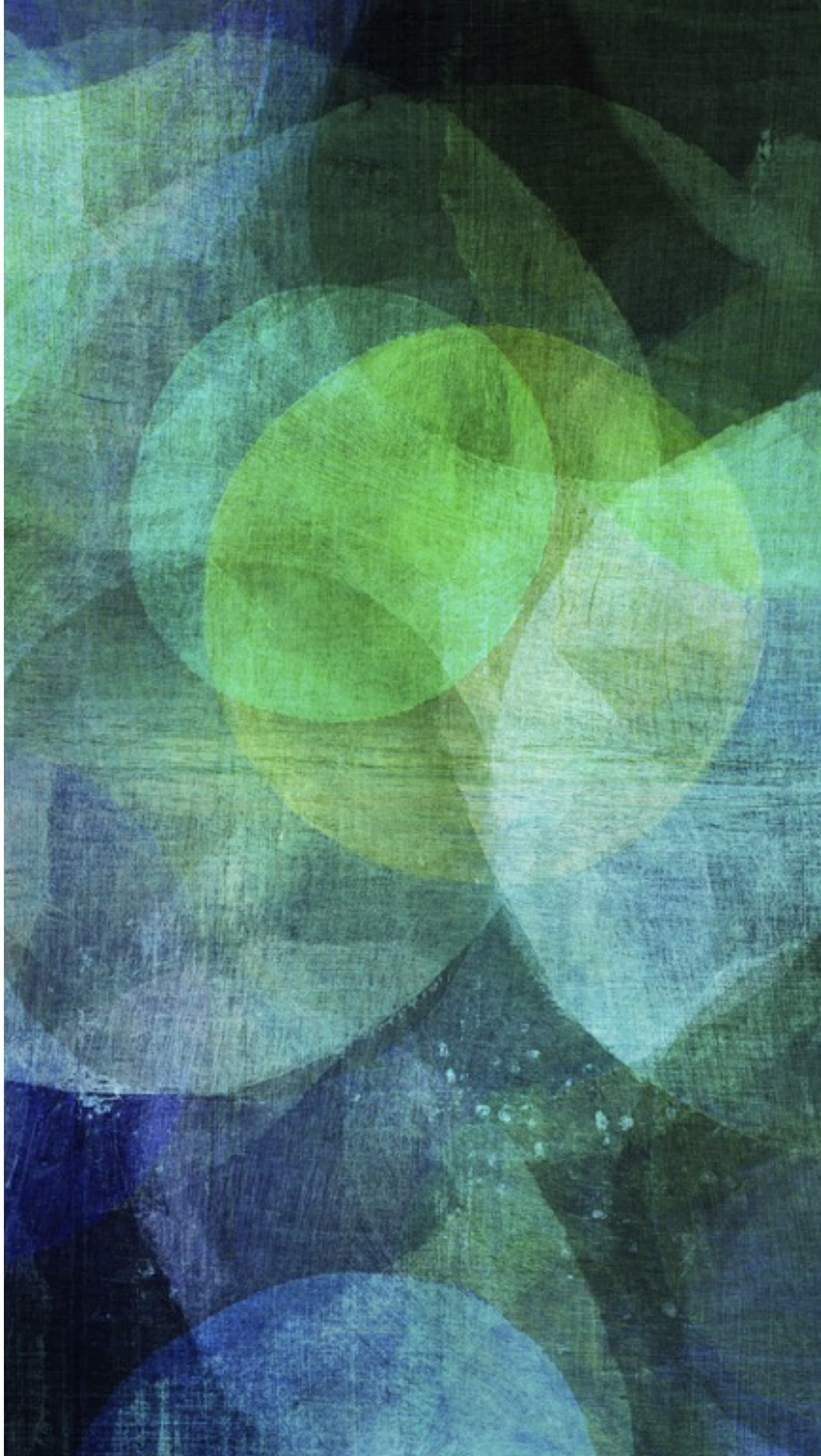
SPARK SQL

Shijie Zhang



OUTLINE

- Background
- Spark SQL overview
- Spark components
  - Data Source API
  - DataFrame API
  - Catalyst optimizer
- SparkSQL future
- Conclusion
- References

WHY BIG DATA WITH SQL

- SQL is compatible with tooling
  - e.g. Connect to existing BI tools via JDBC/ODBC
- Large pool of engineers proficient in SQL
- Compared with MapReduce, SQL is more expressive/succinct

“

Spark SQL is more than SQL

-not so top secret

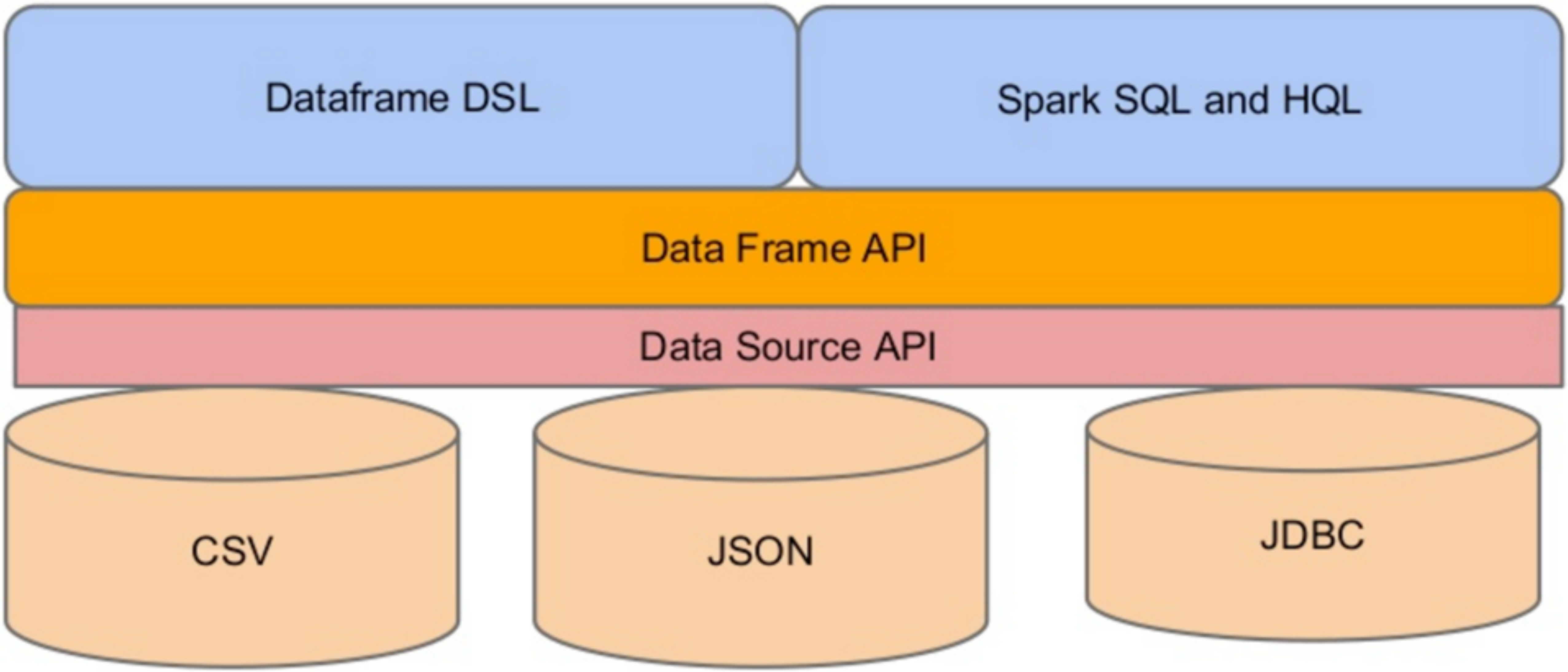
CHALLENGES

- Spark needs to perform ETL on various data source formats
  - Solution: Data source APIs
- Implement SQL for Spark
  - In an extensible way
    - Solution: DataFrame API
  - In an efficient way
    - Solution: Catalyst optimizer

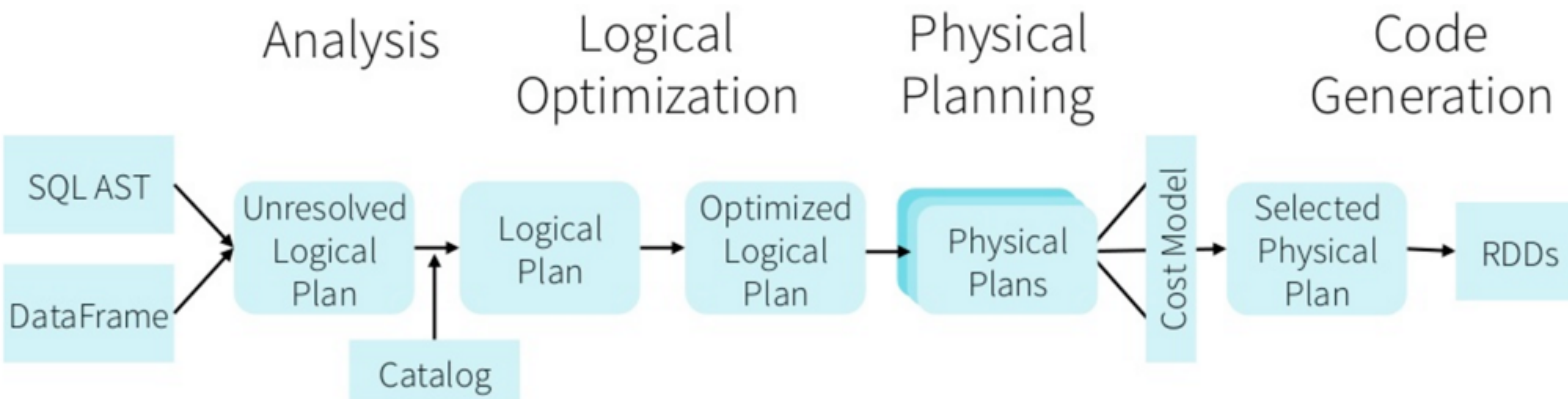
Spark SQL

Major Components

SPARK SQL OVERALL ARCHITECTURE



SPARK SQL WORKFLOW



DATA SOURCE API

- Read and write with a variety of formats

DATA SOURCE API

- Unified interface to reading/writing data in a variety of formats

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

DATA SOURCE API

- Unified interface to reading/writing data in a variety of formats

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

read and write functions create new builders for doing I/O

DATA SOURCE API

➤ Unified interface to reading/writing data in a variety of formats

```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

Builder methods are used to specify:

- Format
- Partitioning
- Handling of existing data
- and more

DATA SOURCE API

- Unified interface to reading/writing data in a variety of formats

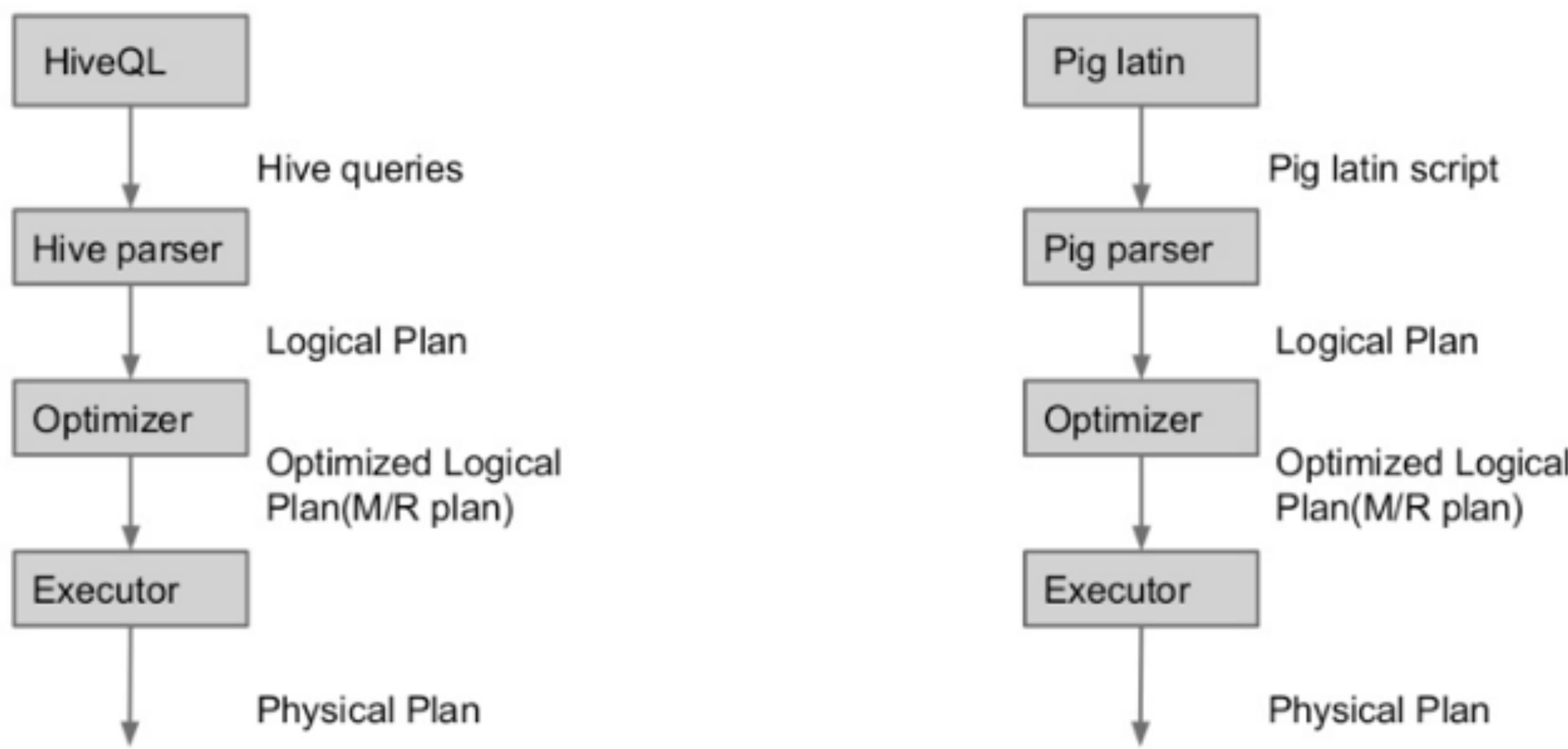
```
df = sqlContext.read \
  .format("json") \
  .option("samplingRatio", "0.1") \
  .load("/home/michael/data.json")

df.write \
  .format("parquet") \
  .mode("append") \
  .partitionBy("year") \
  .saveAsTable("fasterData")
```

load(…), save(…) or saveAsTable(…) functions create new builders for doing I/O

SQL ON HADOOP

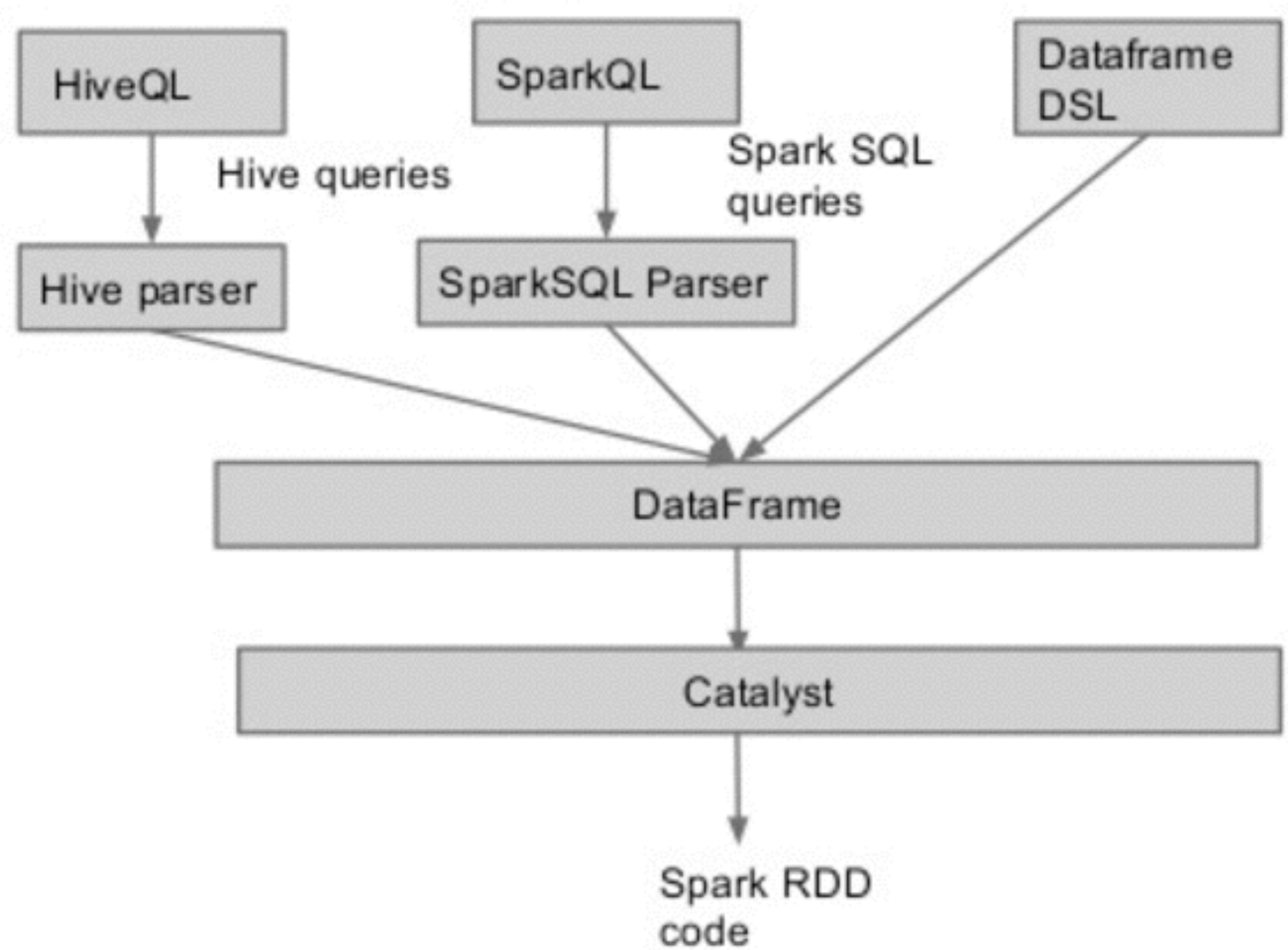
- Hive and Pig pipeline



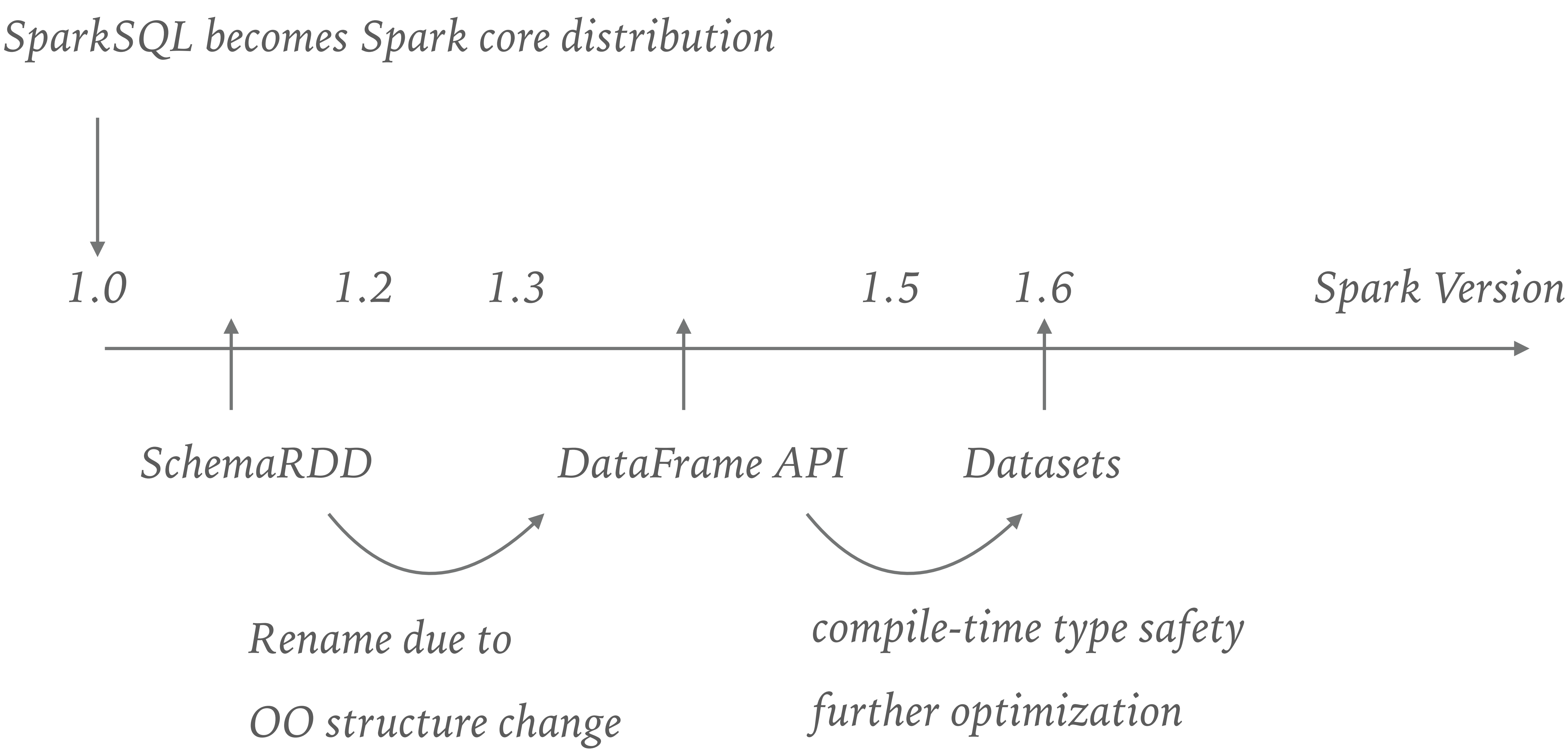
- Problems
  - Quite similar but implements its own optimizer/executor
  - No common data structure to use both Hive/Pig/Other DSL

DATAFRAME API

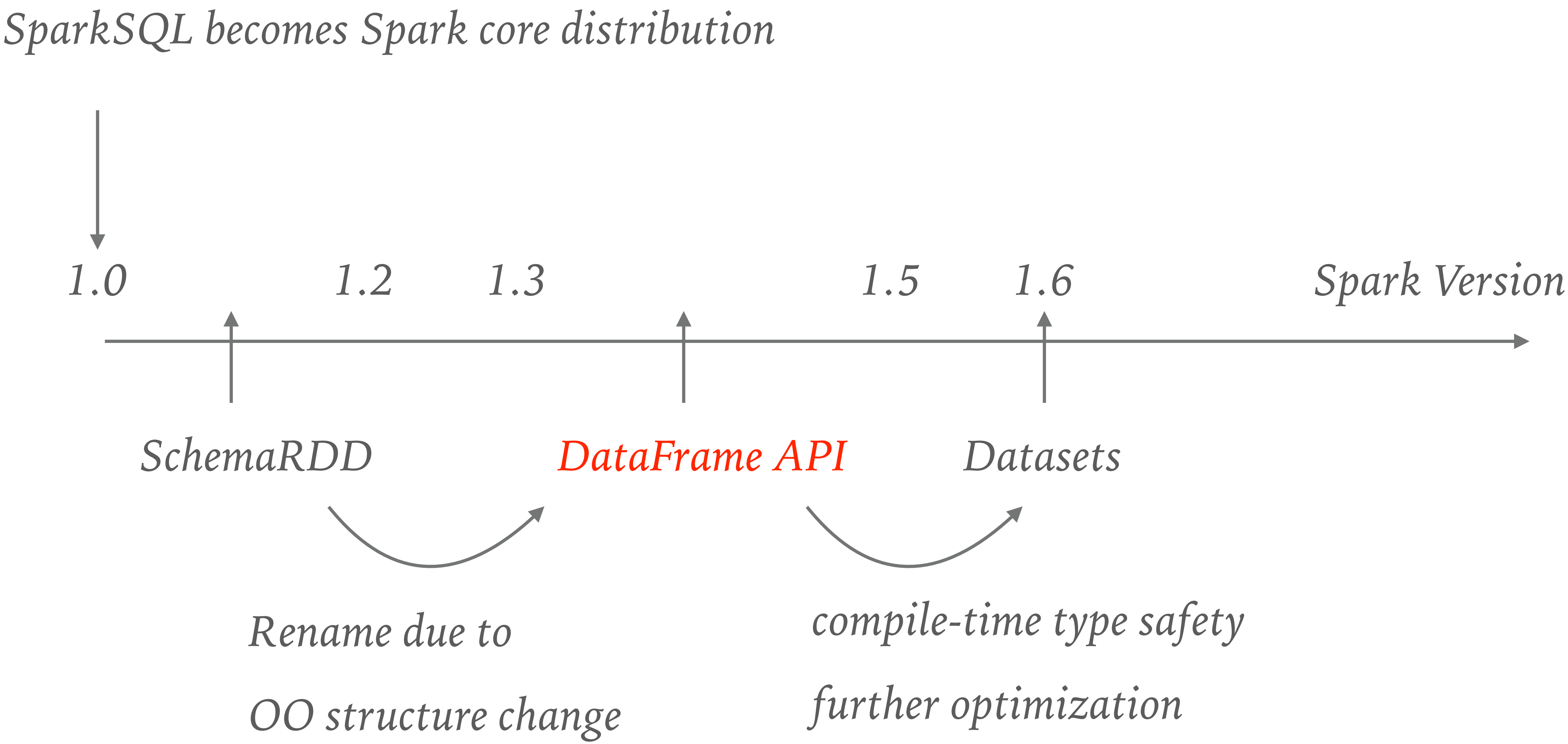
- DataFrame as a front end
- Multiple DSL share same optimizer and executor
- Plug your own DSL too



MAJOR MILESTONE IN SPARK SQL



MAJOR MILESTONE IN SPARK SQL



DATAFRAME API

- Idea borrowed from Python Panda
  - single node tabular data with API for (math, stats, algebra…)
- Def:
  - RDD + Schema
  - RDDs with additional relational operators such as
    - selecting required columns
    - joining different data sources
    - aggregation
    - filtering

DATAFRAME API

- Writing less code

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Using SQL

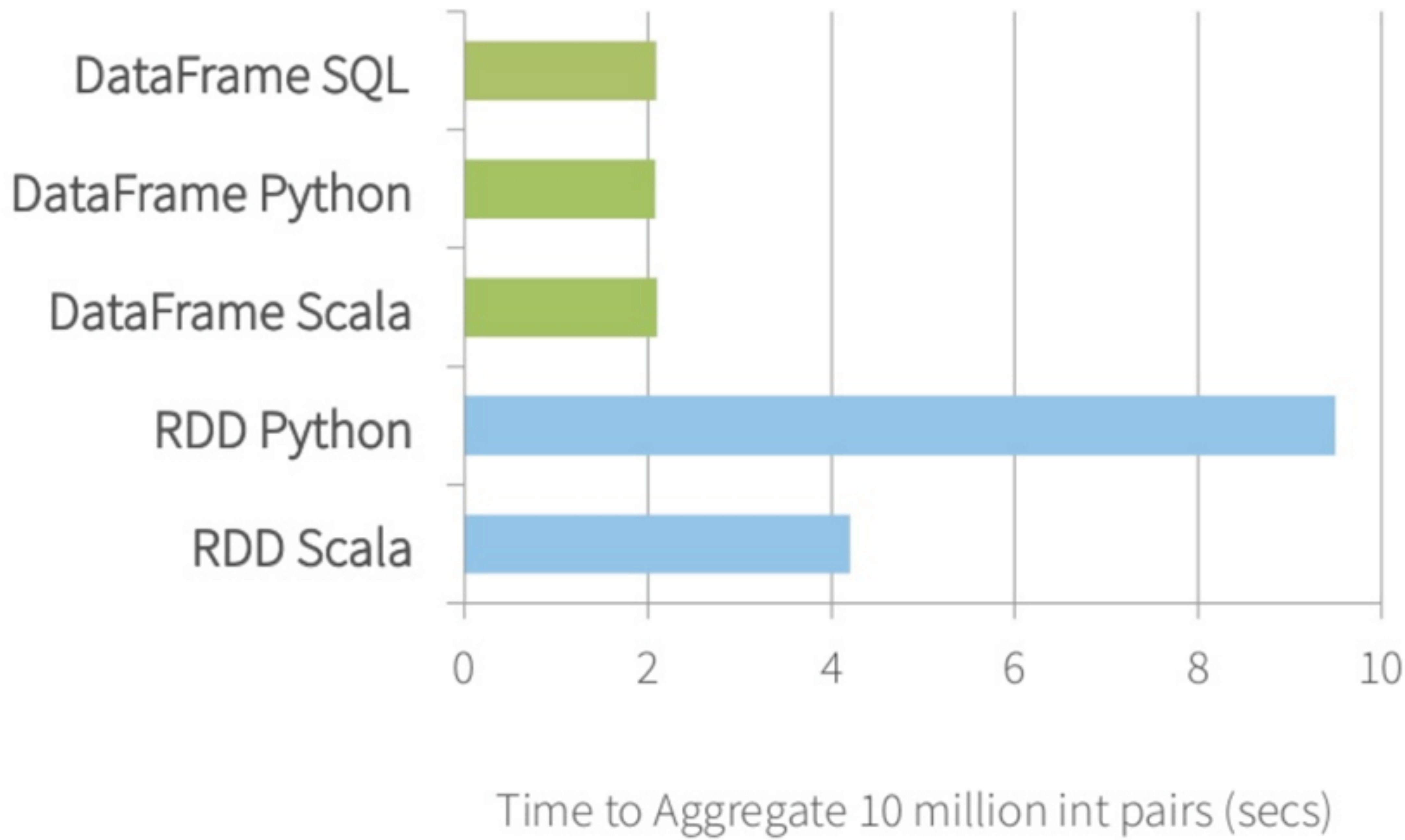
```
SELECT name, avg(age)
FROM people
GROUP BY name
```

Using DataFrames

```
sqlCtx.table("people") \
   .groupBy("name") \
   .agg("name", avg("age")) \
   .collect()
```

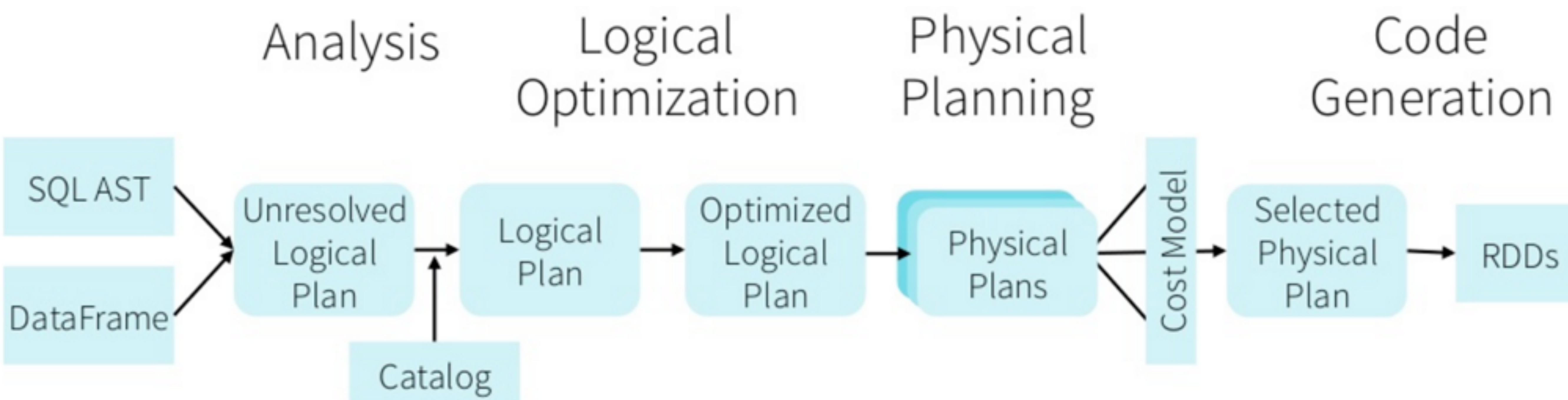
DATAFRAME API

- Faster implementations



CATALYST OPTIMIZER

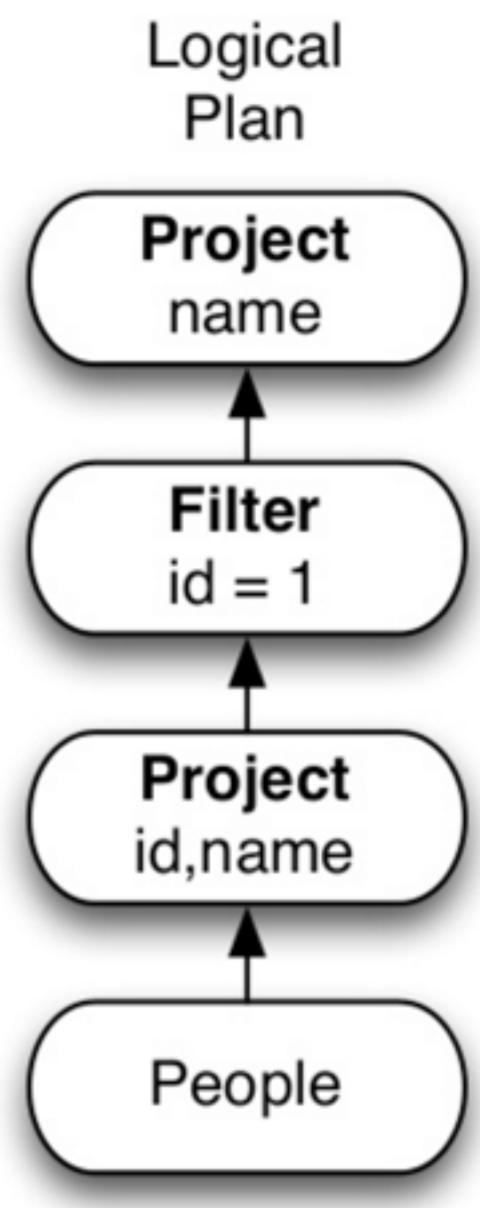
- Goal: convert logical plan to physical plans
- Process:
  - Logical plan is a tree representing data and schema
  - The tree is manipulated and optimized by catalyst rules



CATALYST OPTIMIZER

- An example query

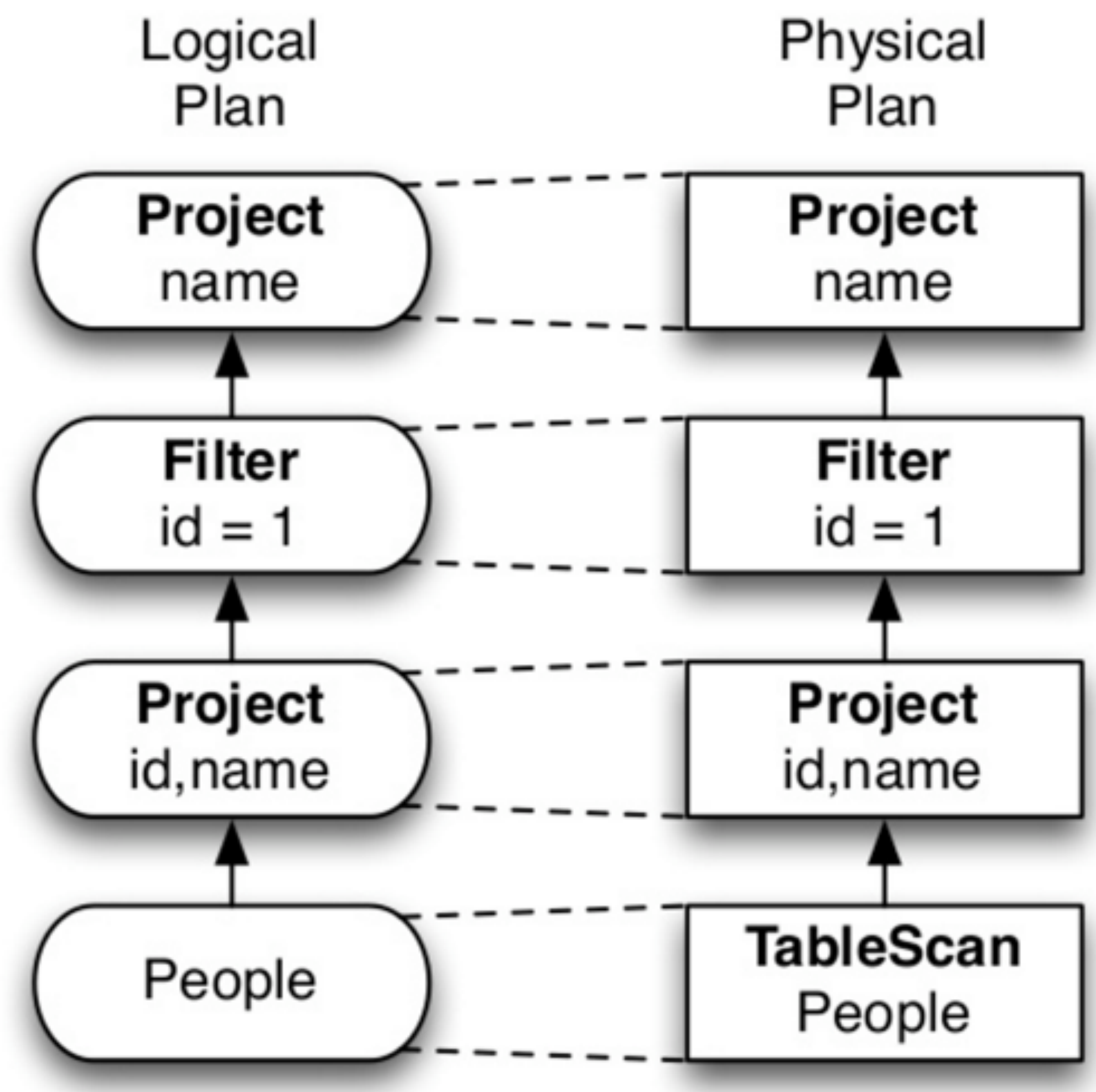
```
SELECT name
FROM (
    SELECT id, name
    FROM People) p
WHERE p.id = 1
```



CATALYST OPTIMIZER

- Native query planning

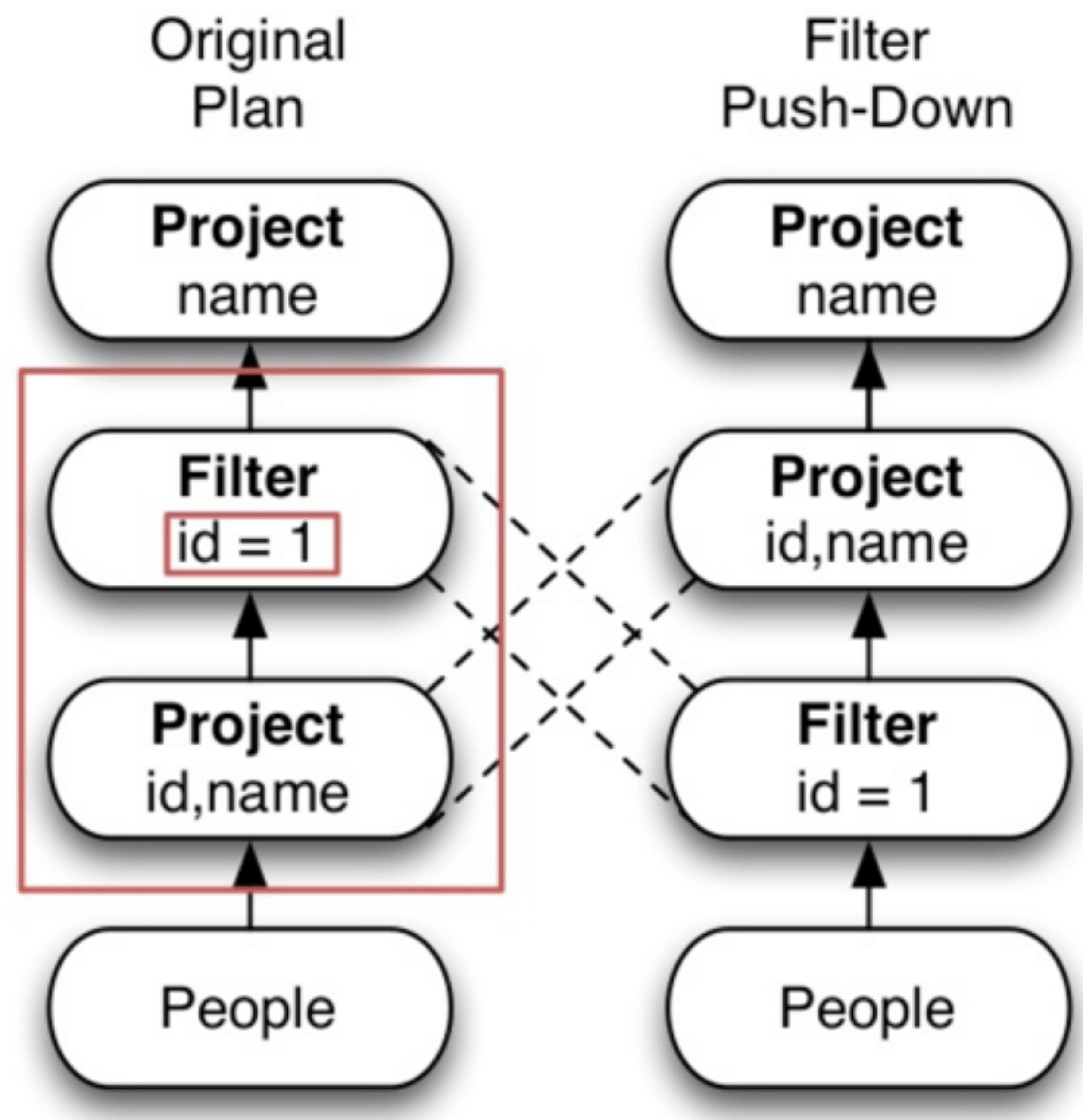
```
SELECT name
FROM (
    SELECT id, name
    FROM People) p
WHERE p.id = 1
```



CATALYST OPTIMIZER

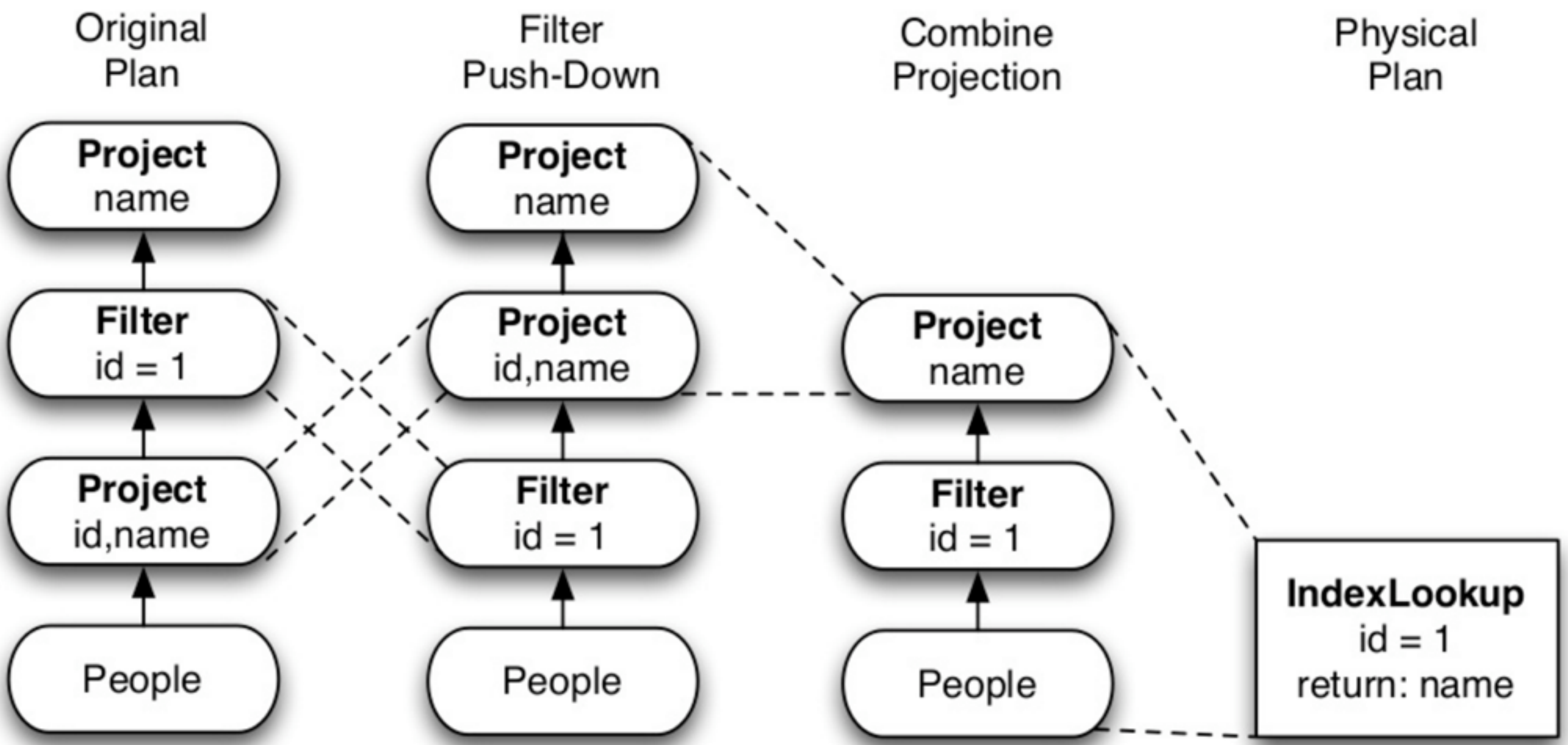
- Optimization Rules example

1. Find filters on top of projections.
2. Check that the filter can be evaluated without the result of the project.
3. If so, switch the operators.



CATALYST OPTIMIZER

- Optimization Rules continued
  - Allow defining customized rules



CATALYST OPTIMIZER

- Optimizing rules
  - Eliminate subqueries
  - Constant folding
  - Simplify filters
  - PushPredicate through filter
  - Project collapsing

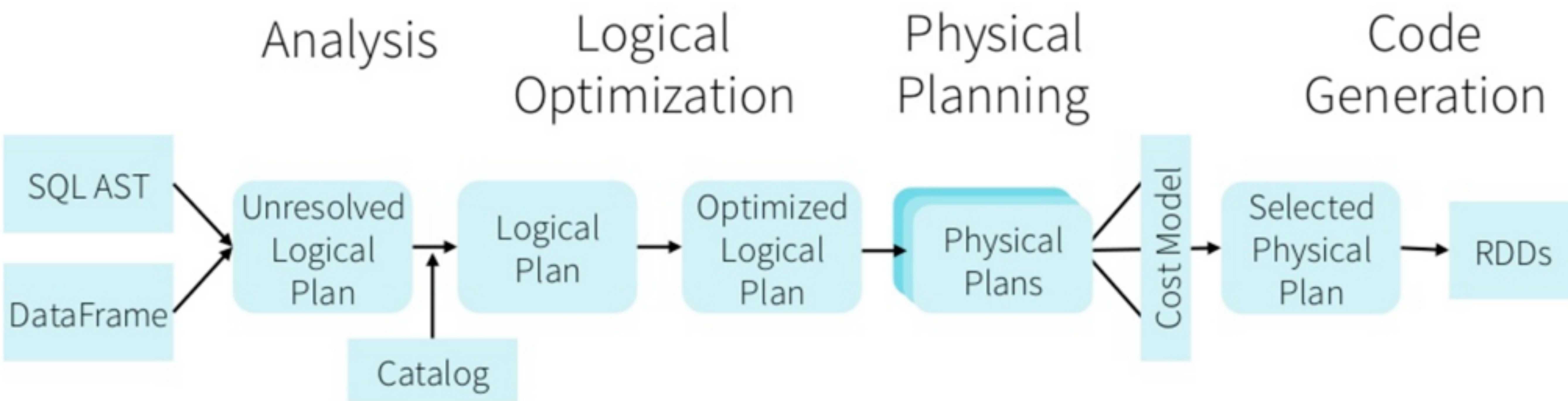
SPARK SQL FUTURE

- Tungsten - Optimization for the next few years
- Begin with hardware trends

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

SPARK SQL FUTURE

- Tungsten - Optimization for the next few years
- Recall our SparkSQL workflow



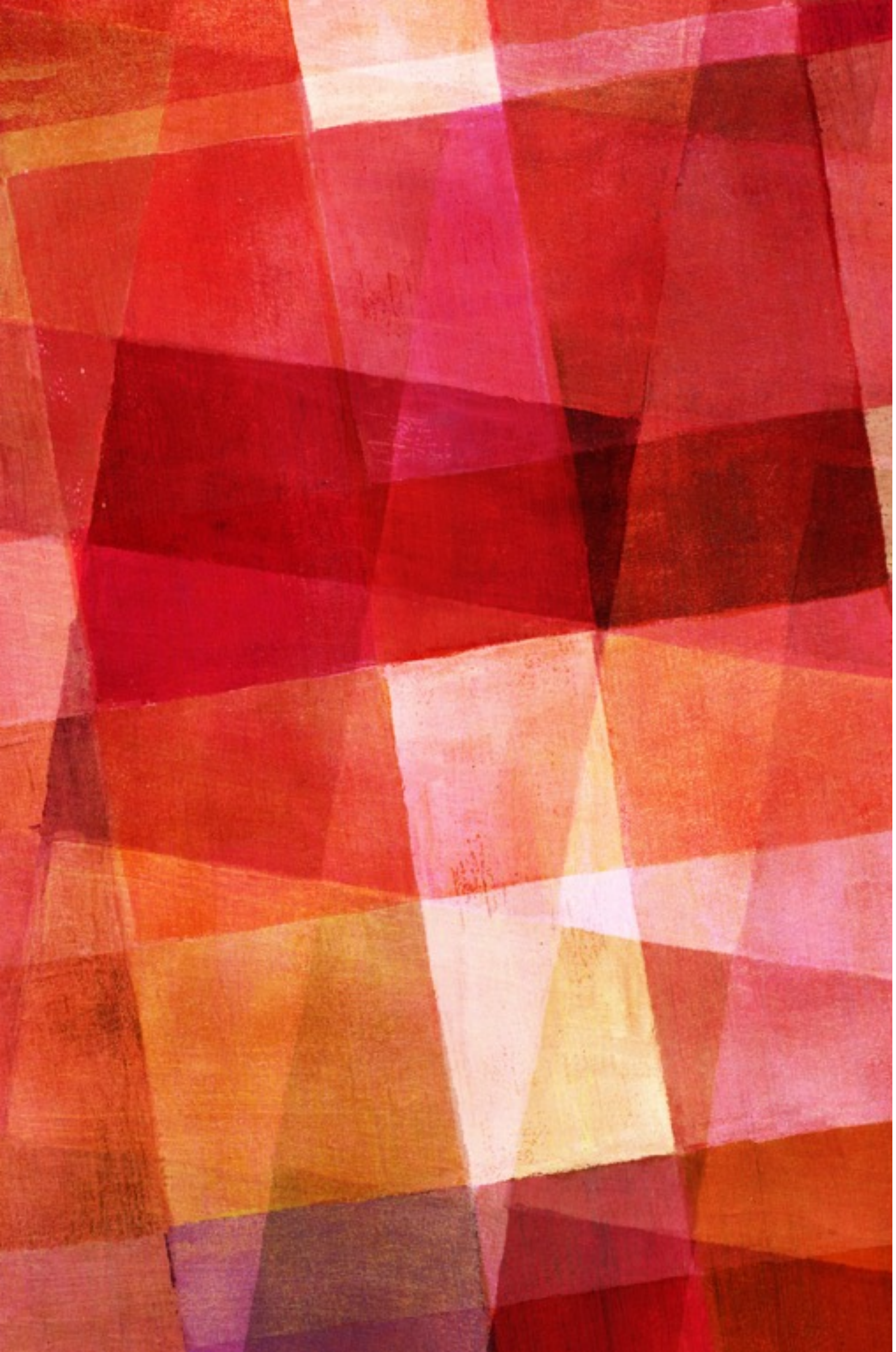
SPARK SQL FUTURE

- Tungsten - Preparing Spark for next 5 years
- Begin with hardware trends

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

SPARK SQL FUTURE

- Tungsten - Preparing Spark for next 5 years
- Substantially speed up execution by optimizing CPU efficiency via
  - Runtime code generation
  - Exploiting cache locality
  - Off-heap memory management



CONCLUSION

- SparkSQL intuition
- SparkSQL pipeline/ architecture

REFERENCES

- http://www.slideshare.net/datamantra/anatomy-of-data-frame-api
- http://www.slideshare.net/databricks/2015-0616-spark-summit
- http://www.slideshare.net/databricks/spark-sql-deep-dive-melbroune
- http://www.slideshare.net/databricks/spark-sqlsse2015public
- http://www.slideshare.net/datamantra/introduction-to-structured-data-in-spark
- Data bricks official blog

MY LEARNING WORKFLOW

- Recently I gradually set up my workflow and reduced learning cycle
- Everyone has different learning workflows
- But let's share and make progress together

Knowledge collecting	*Summarization*	*Sharing*
Evernote web clipper	*MindManager*	*Blog:Evernote+postach.io*
Organize notes by tags	*Evernote markdown doc with Marxico*	*Keynote/PowerPoint*