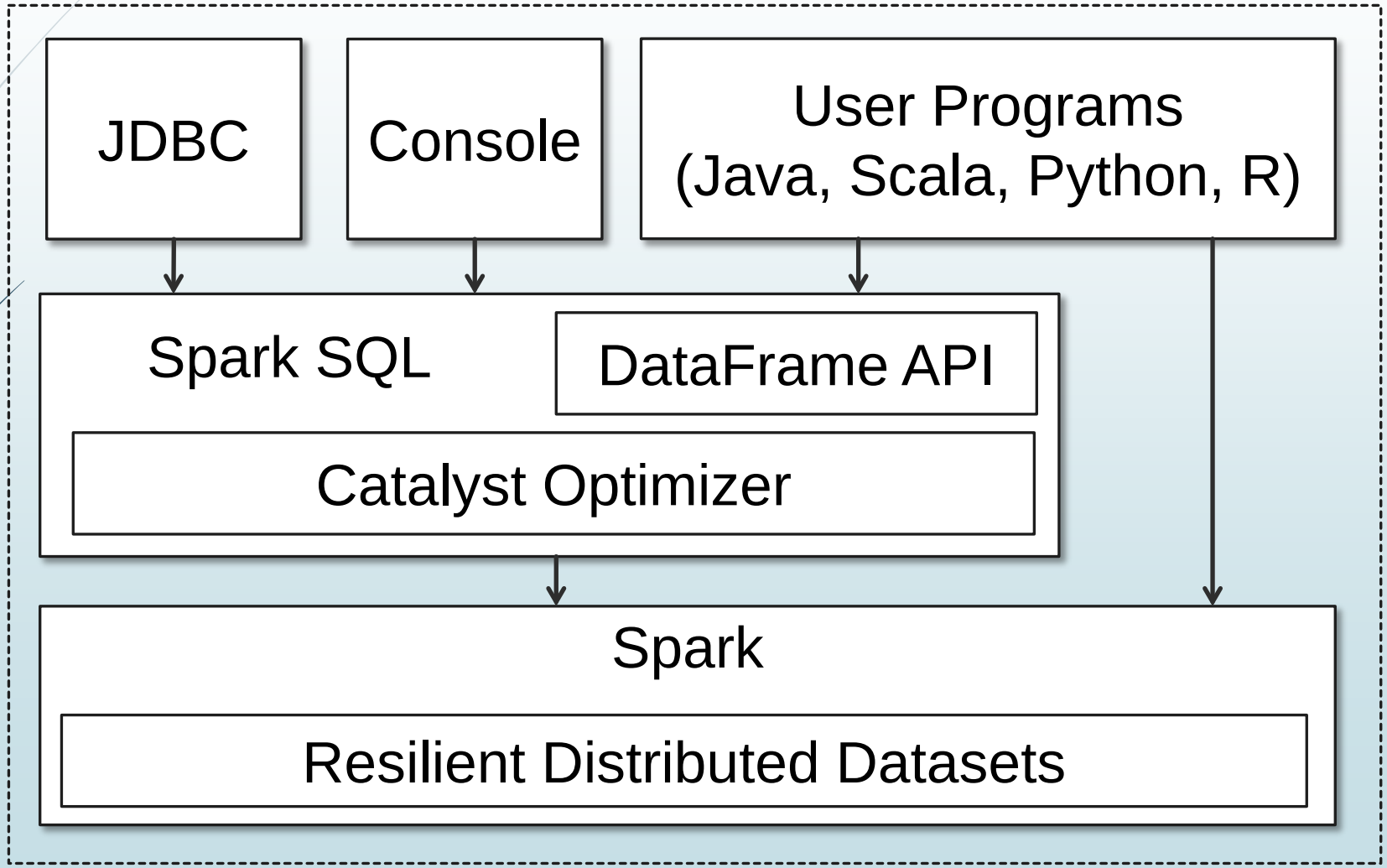
Spark SQL

סמינר במסדי נתונים

Spark SQL


JDBC
Console
User Programs
(Java, Scala, Python, R)
Spark SQL
DataFrame API
Catalyst Optimizer
Spark
Resilient Distributed Datasets

Spark SQL feature support

- Data types:
  - Boolean, Integer, Double, Decimal, String, Date, Timestamp
  - Complex data types: Struct, Array, Map, Union
- In-memory caching
- User-defined functions

Initialization

```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
```

Basic API

```
points = sqlCtx.read.json('example.json')
# show table:
points.show()
# print schema:
points.printSchema()
# filter:
points.filter('x = 1')
# two different select types:
points.select('x', points['y'] + 1)
# both:
points.filter('x = 1').select('x')
# other:
dir(points)
```

example.json:

```
{'x': 0, 'y': 1}
{'x': 1, 'y': 2}
```

Registering tables

```
points.registerTempTable('points') # finally!
sqlCtx.sql("SELECT * FROM points").show() # awesome!
sqlCtx.sql("SELECT y FROM points WHERE x = 1").show()
```

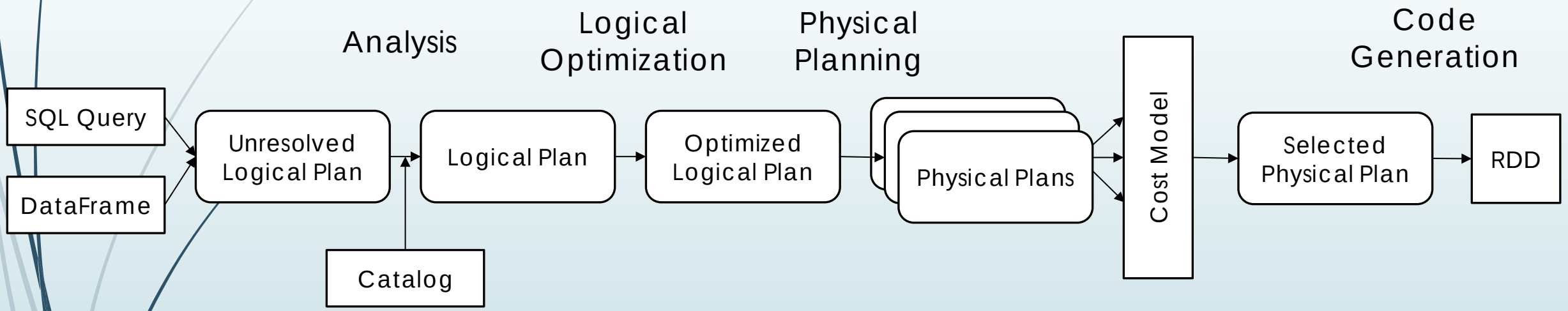
From scratch (from RDD)

```
lines = sc.textFile('example.txt')
points_raw = lines.map(lambda l: l.split(','))
points_raw_int = points_raw.map(lambda p: (int(p[0]), int(p[1])))

from pyspark.sql.types import IntegerType, StructField, StructType

fields = [StructField('x', IntegerType()), StructField('y', IntegerType())]
schema = StructType(fields)

points = sqlCtx.createDataFrame(points_raw_int, schema)
```

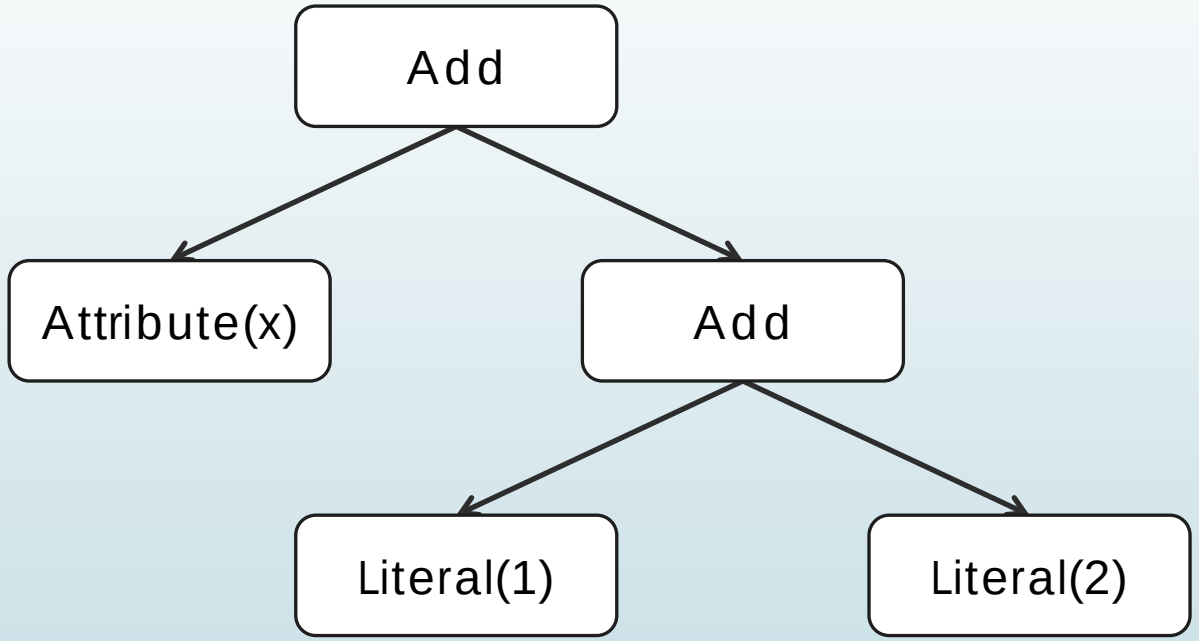
User Defined Functions

```
def foo(x): ...
def bar(x): ...

sqlCtx.udf.register('foo', foo, IntegerType())
sqlCtx.udf.register('bar', bar, BooleanType())
sqlCtx.sql("SELECT foo(x) FROM points WHERE bar(y)").show()
```

Catalyst


Analysis
Logical Optimization
Physical Planning
Code Generation
SQL Query
DataFrame
Unresolved Logical Plan
Logical Plan
Optimized Logical Plan
Physical Plans
Cost Model
Selected Physical Plan
RDD
Catalog

Catalyst tree for the expression x+(1+2)



Pattern matching rules (in Scala)

```
Add(Attribute(x), Add(Literal(1), Literal(2)))
```

+

```
tree.transform {
    case Add(Literal(c1), Literal(c2)) => Literal(c1+c2)
    case Add(left, Literal(0)) => left
    case Add(Literal(0), right) => right
}
```

=

```
Add(Attribute(x), Literal(3))
```

Code generation of Scala AST

```
Add(Literal(1), Attribute(x))
```

+

```
def compile(node: Node): AST = node match {
    case Literal(value) => q"$value"
    case Attribute(name) => q"row.get($name)"
    case Add(left, right) =>
        q"${compile(left)} + ${compile(right)}"
}
```

=

```
1 + row.get("x")
```