SystemML: Declarative Machine Learning on MapReduce

By: Rohit Ranjan

Topics

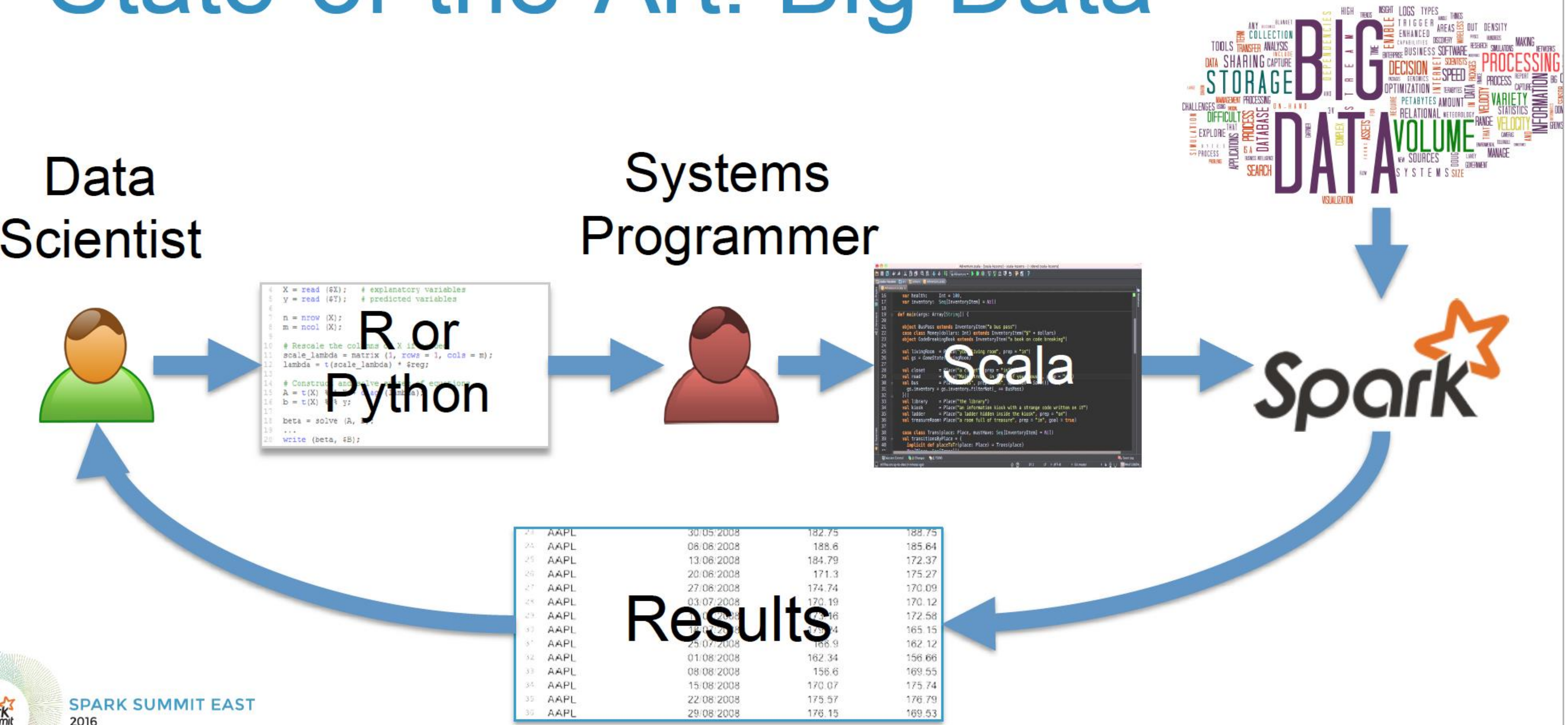
- Declarative Machine Learning Language(DML)
- SystemML
  - Overview
  - Architecture
  - Components
  - Matrix Multiplication Algorithm
  - Experiments

Why Declarative Machine Learning Language?

Why Declarative Machine Learning Language?

- Data scientists can write an algorithm in an expressive language
- 4 major requirements for Algorithms:
  - **High-level semantics**: A data scientist should be able to write an algorithm in a high-level language without focusing on any low-level implementation details
  - **Flexibility**: A data scientist should have flexibility to leverage existing algorithms with or without any customization
  - **Data independence**: A data scientist should not worry about data characteristics while writing the algorithms.
  - **Scale independence**: The size of the data could be small or large

State-of-the-Art: Big Data



State-of-the-Art: Big Data

Data Scientist

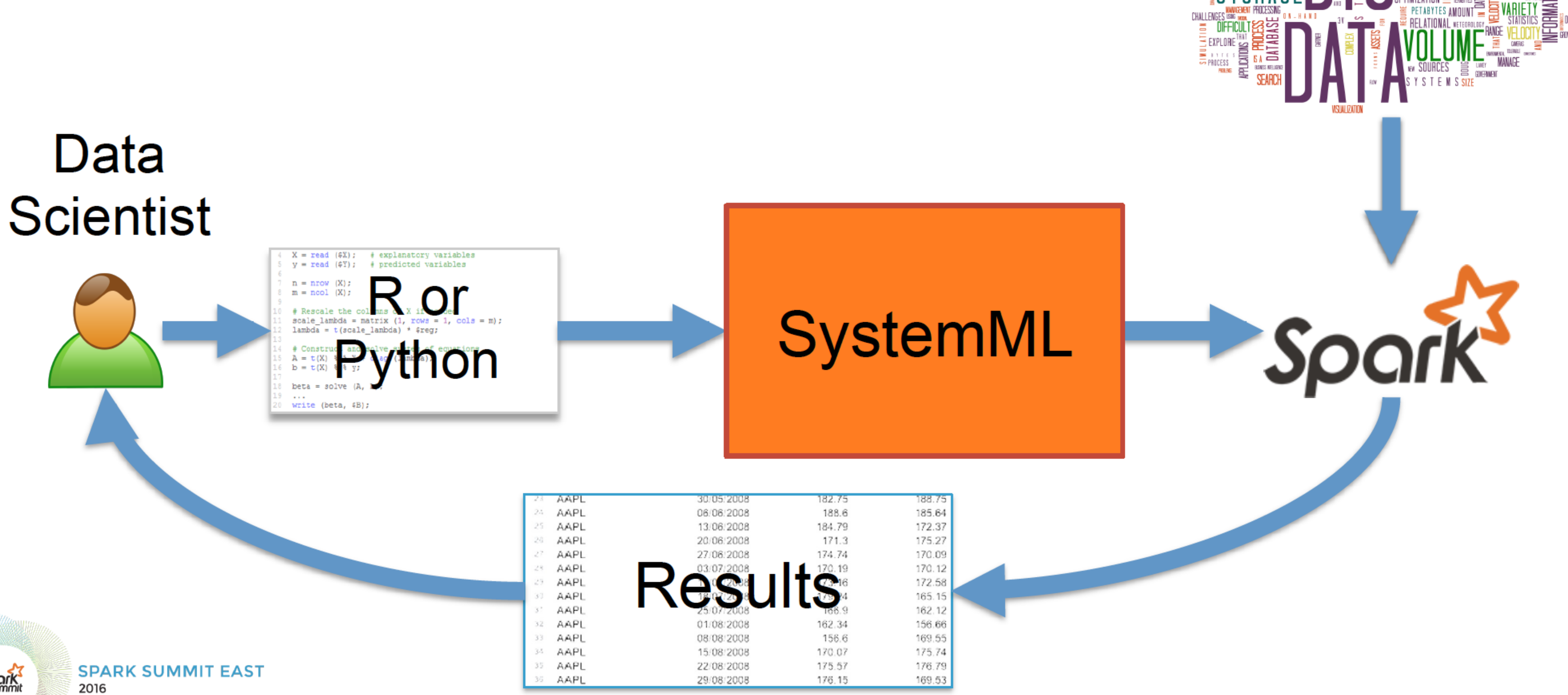
- 😞 Days or weeks per iteration
- 😞 Errors while translating algorithms

Spark

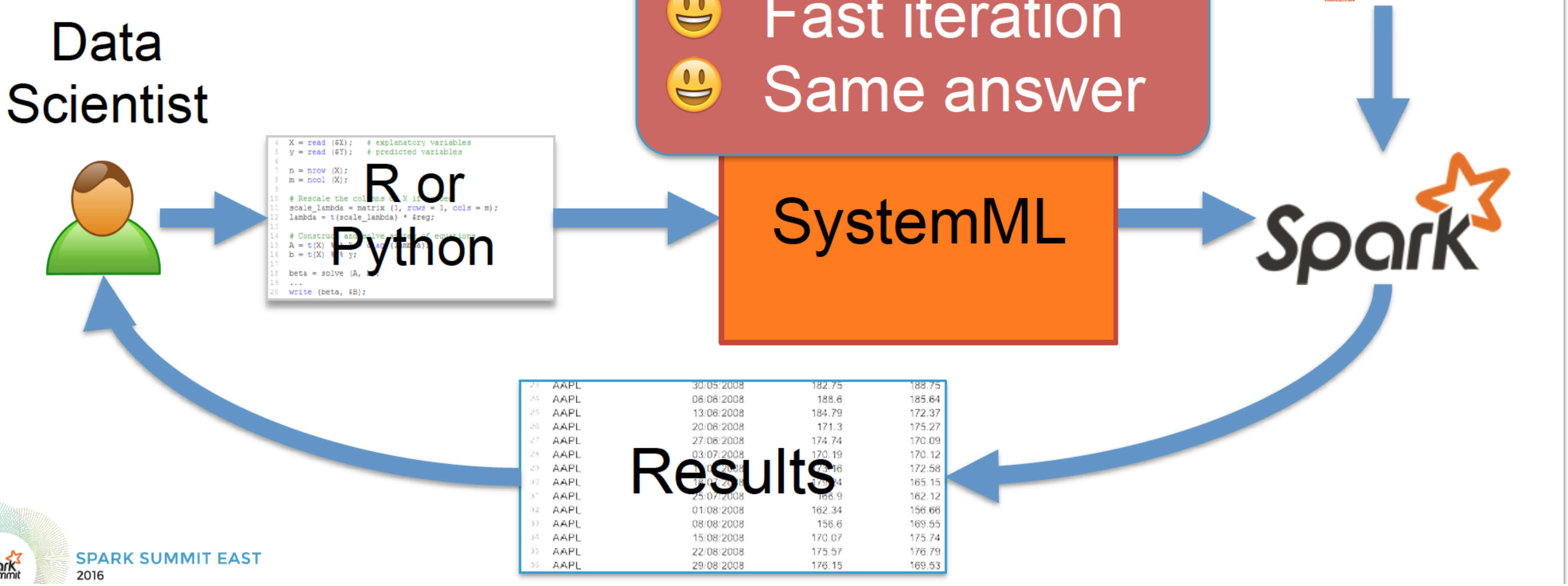
Results

23	AAPL	30/05/2008	182.75	188.75
24	AAPL	06/06/2008	188.6	185.64
25	AAPL	13/06/2008	184.79	172.37
26	AAPL	20/06/2008	171.3	175.27
27	AAPL	27/06/2008	174.74	170.09
28	AAPL	03/07/2008	170.19	170.12
29	AAPL	[illegible]	[illegible]	172.58
30	AAPL	[illegible]	[illegible]	165.15
31	AAPL	25/07/2008	[illegible]	162.12
32	AAPL	01/08/2008	162.34	156.66
33	AAPL	08/08/2008	156.6	169.55
34	AAPL	15/08/2008	170.07	175.74
35	AAPL	22/08/2008	175.57	176.79
36	AAPL	29/08/2008	176.15	169.53

The SystemML Vision



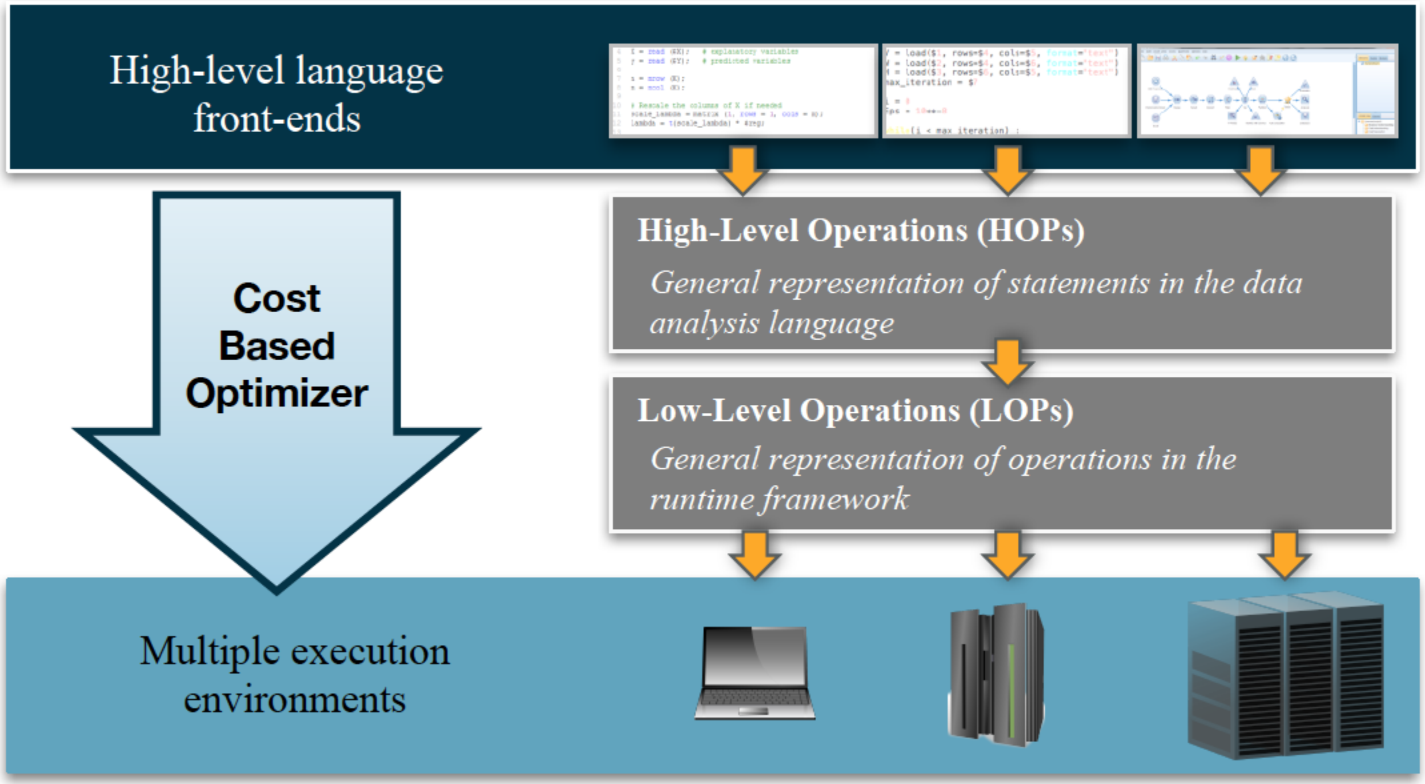
The SystemML Vision

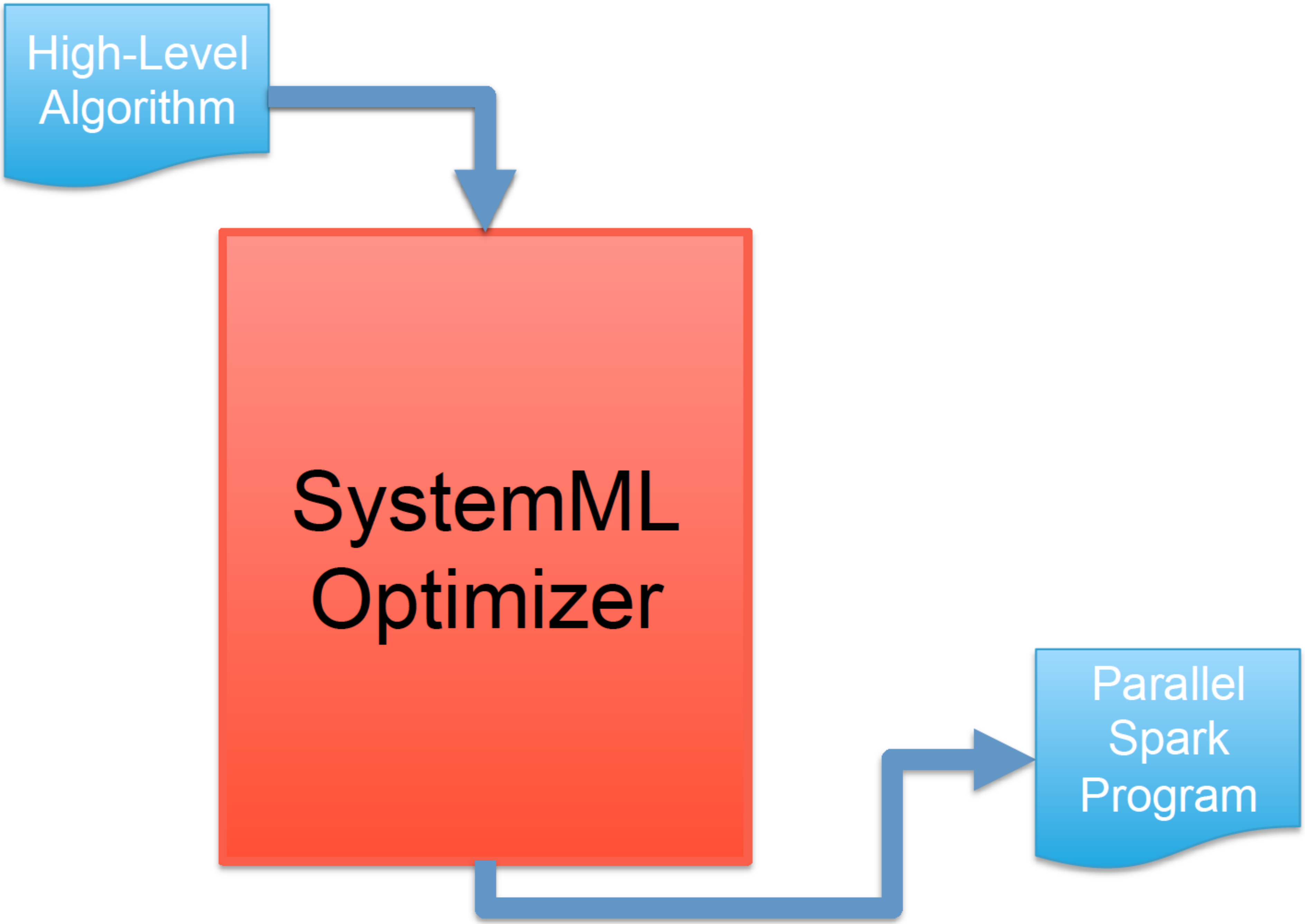


What is Apache SystemML?

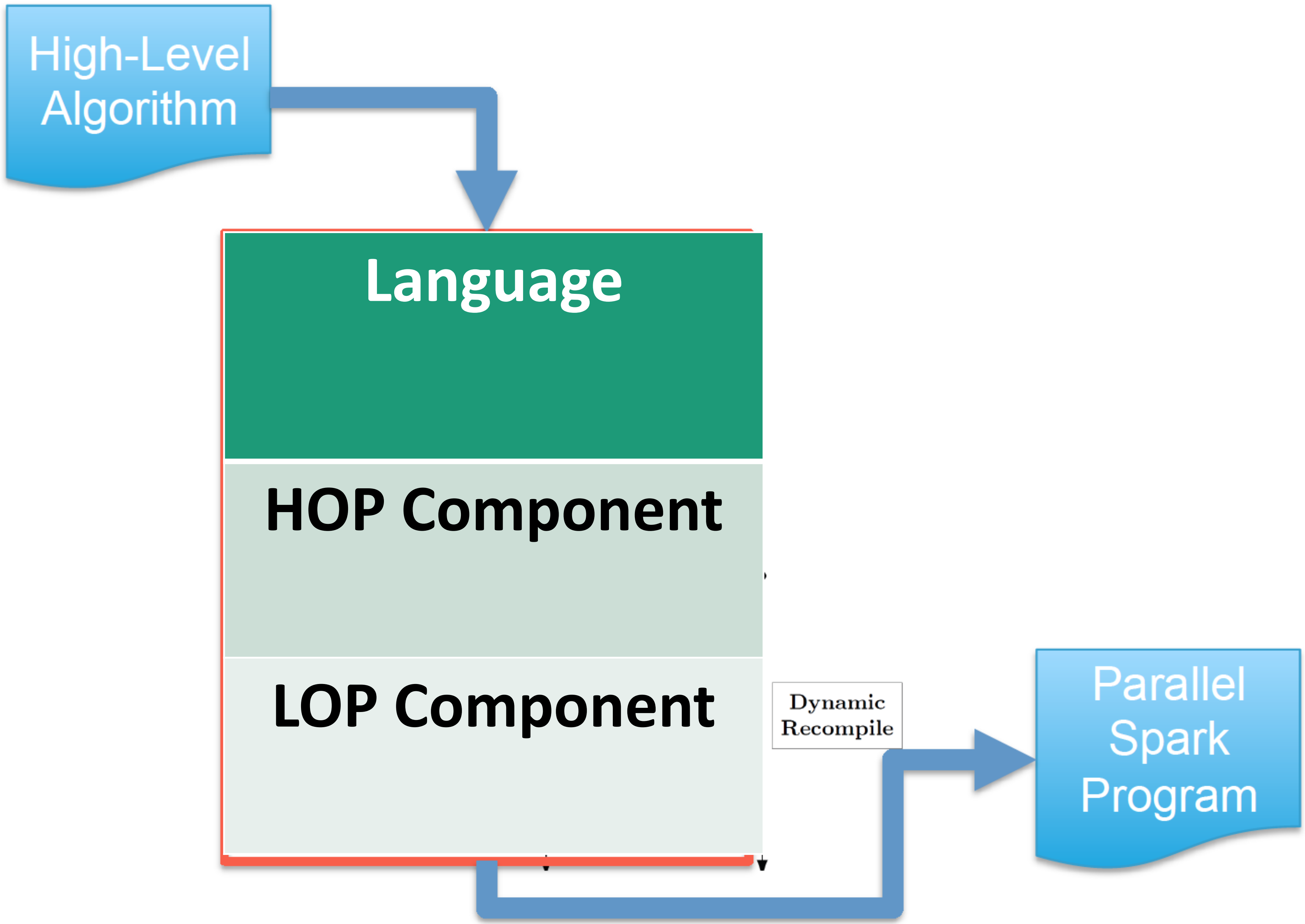
- **In a nutshell**
  - **a language for data scientists to implement scalable ML algorithms**
    - 2 language variants: R-like and Python-like syntax
    - Strong foundation of linear algebra operations and statistical functions
    - Comes with approx. 20+ algorithms pre-implemented
  - **Cost-based optimizer to compile execution plans**
    - Depending on data characteristics (tall/skinny, short/wide; dense/sparse) and cluster characteristics
    - ranging from single node to clusters (MapReduce, Spark); hybrid plans
- **APIs & Tools**
  - **Command line: hadoop jar, spark-submit, standalone Java app**
  - **JMLC: embed as library**
  - **Spark MLContext: Scala, Python, and Java**
  - **Tools**
    - REPL (Scala Spark and pyspark)
    - Spark ML pipeline

High level Architecture of SystemML

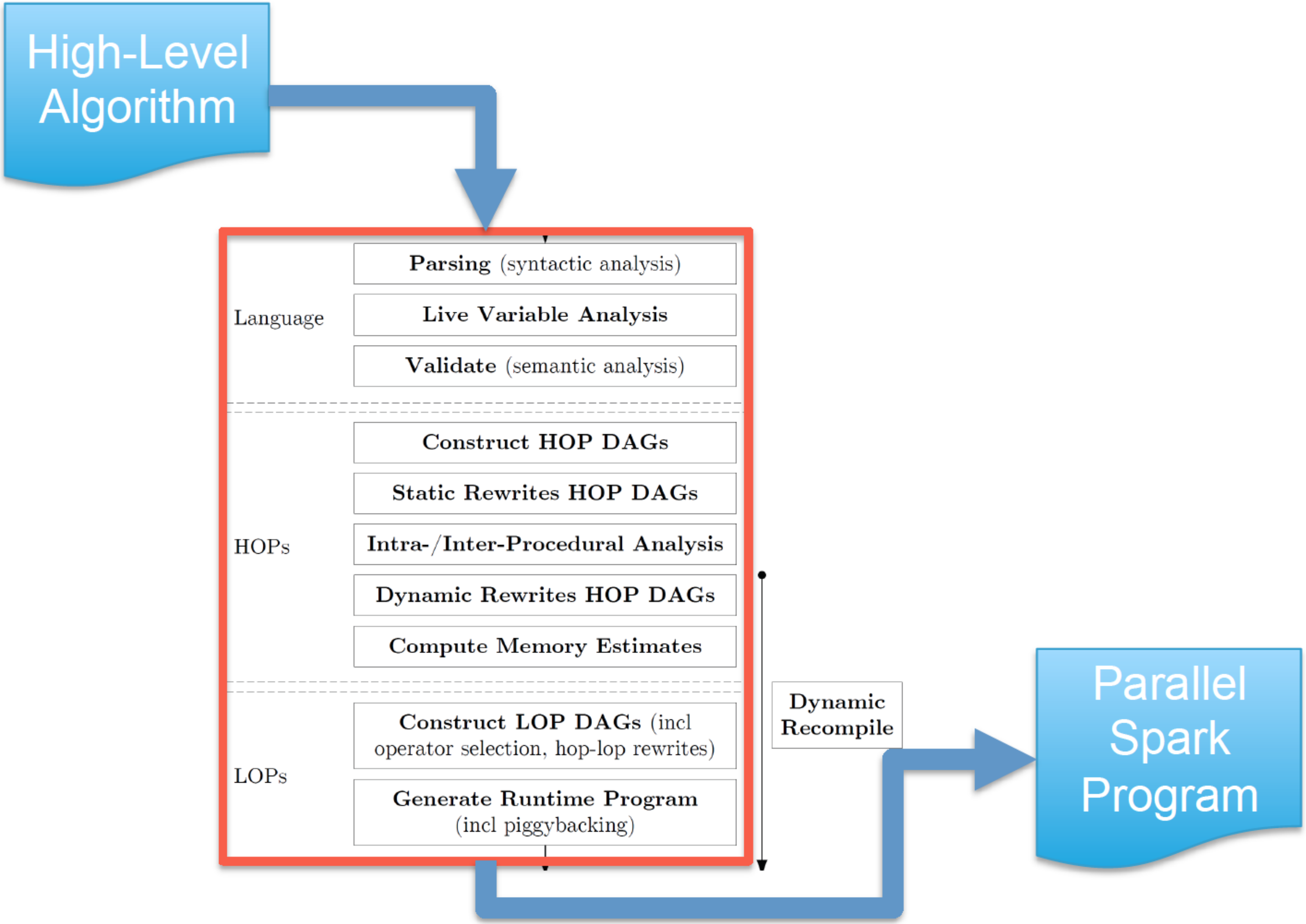



High-Level Algorithm
SystemML Optimizer
Parallel Spark Program

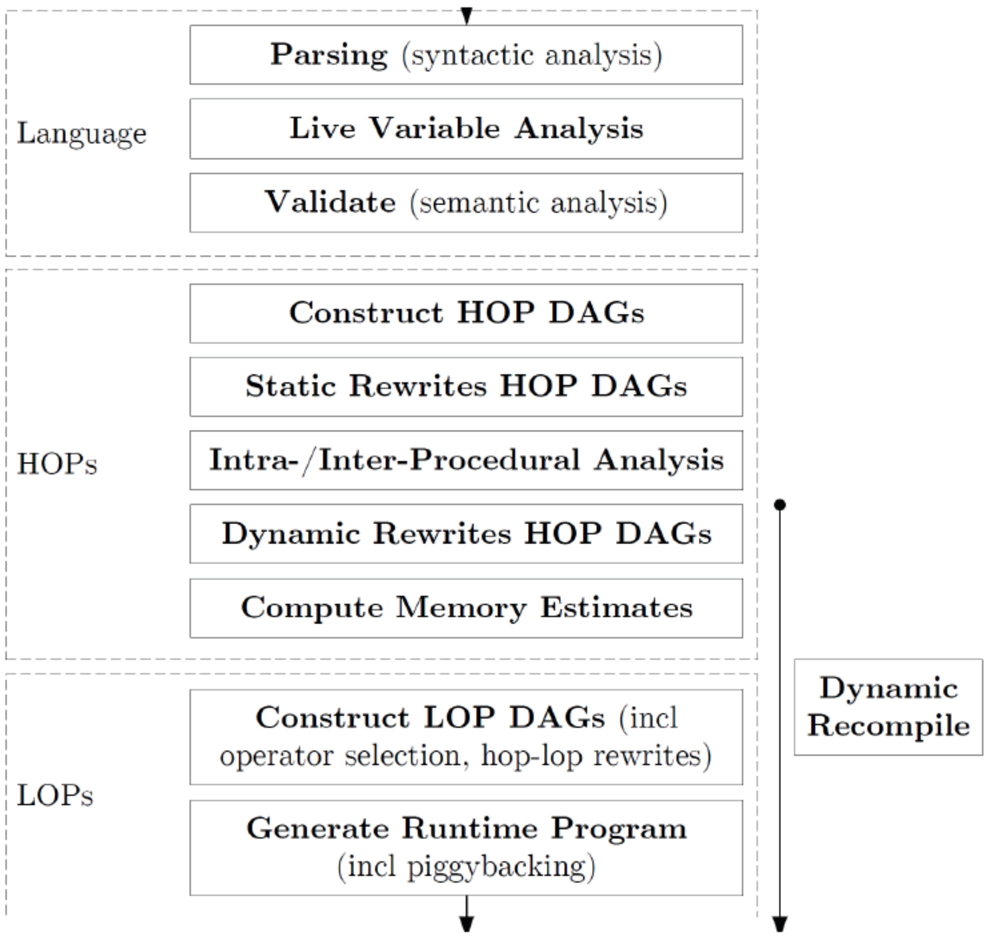
AST



High-Level Algorithm
Language
HOP Component
LOP Component
Dynamic Recompile
Parallel Spark Program


High-Level Algorithm
Language
Parsing (syntactic analysis)
Live Variable Analysis
Validate (semantic analysis)
HOPs
Construct HOP DAGs
Static Rewrites HOP DAGs
Intra-/Inter-Procedural Analysis
Dynamic Rewrites HOP DAGs
Compute Memory Estimates
LOPs
Construct LOP DAGs (incl operator selection, hop-lop rewrites)
Generate Runtime Program (incl piggybacking)
Dynamic Recompile
Parallel Spark Program

The SystemML Optimizer Stack

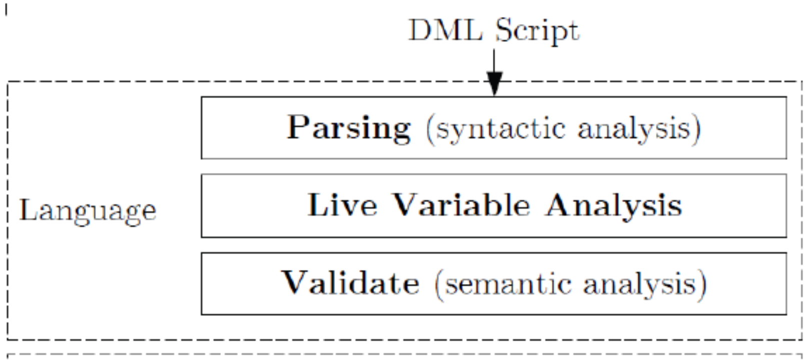


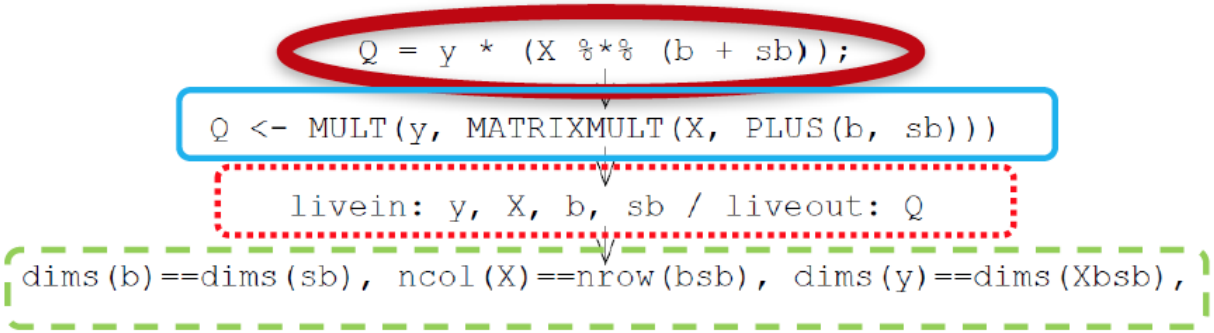
SystemML Compilation Chain

DML Script

```
Q = y * (X %*% (b + sb));
```

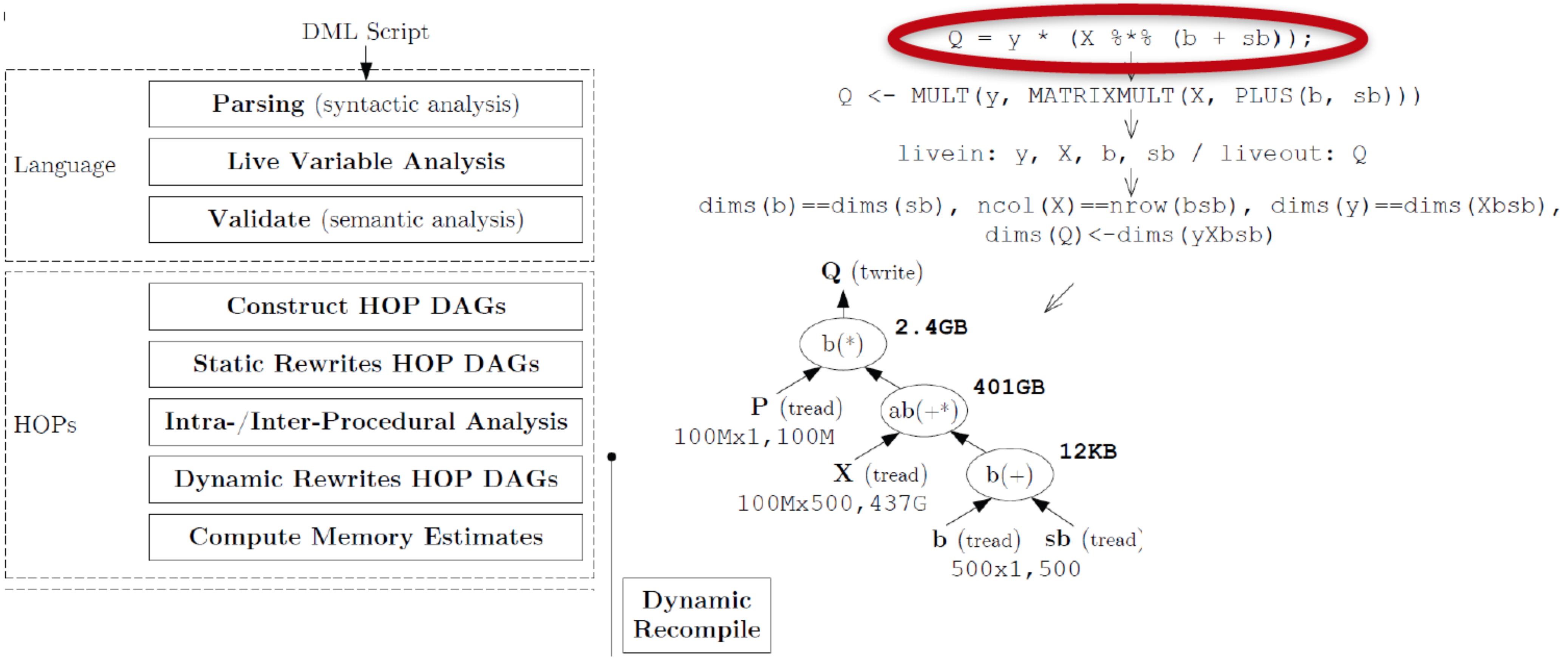
SystemML Compilation Chain





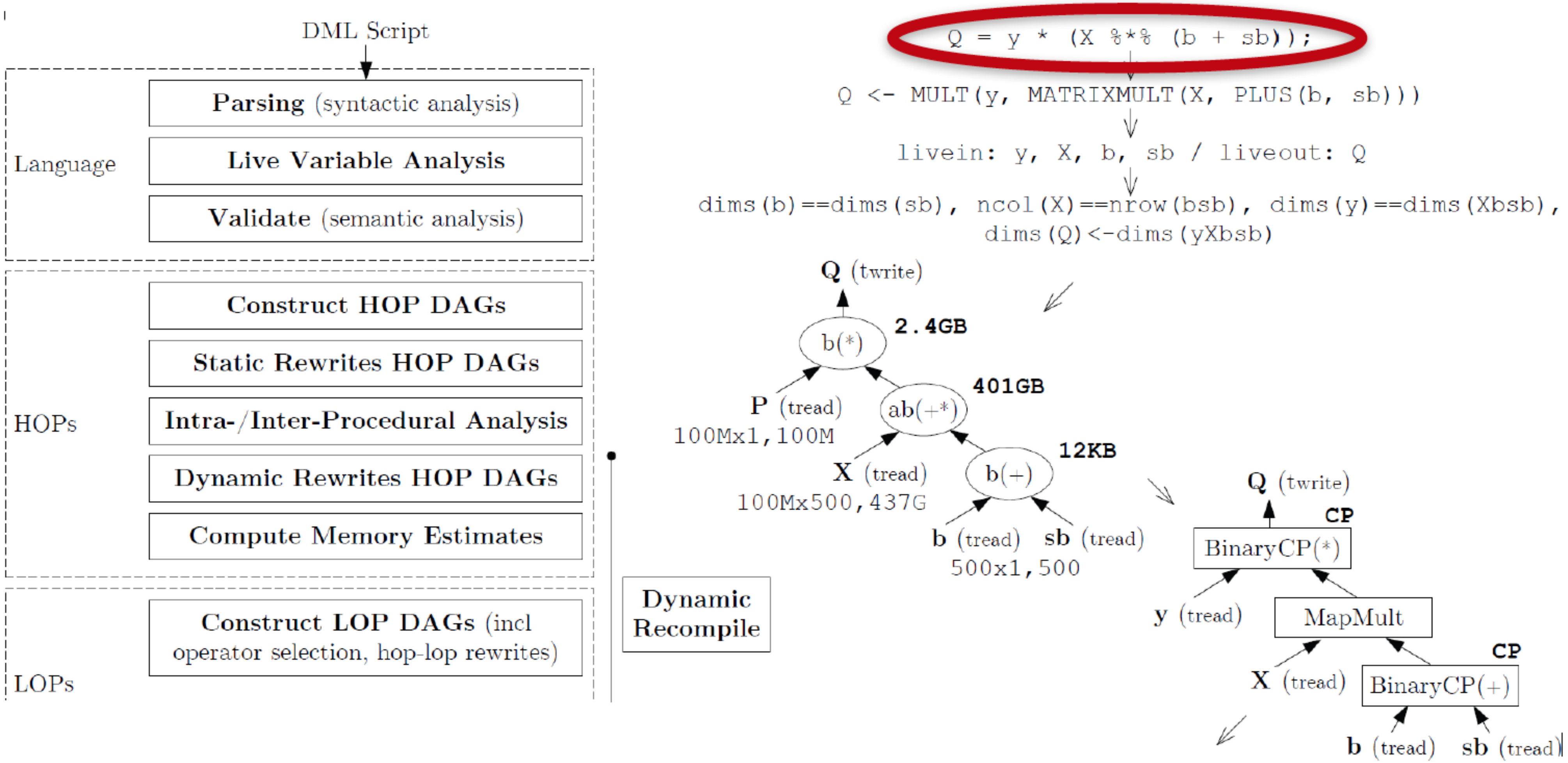
- Parsing
  - Parse input DML/PyDML using Antlr v4 (see Dml.g4 and Pydml.g4)
  - Perform syntactic validation
  - Construct DMLProgram (=> list of Statement and function blocks)
- Live Variable Analysis
  - Classic dataflow analysis
  - A variable is “live” if it holds value that may be needed in future
  - Dead code elimination
- Semantic Validation

SystemML Compilation Chain



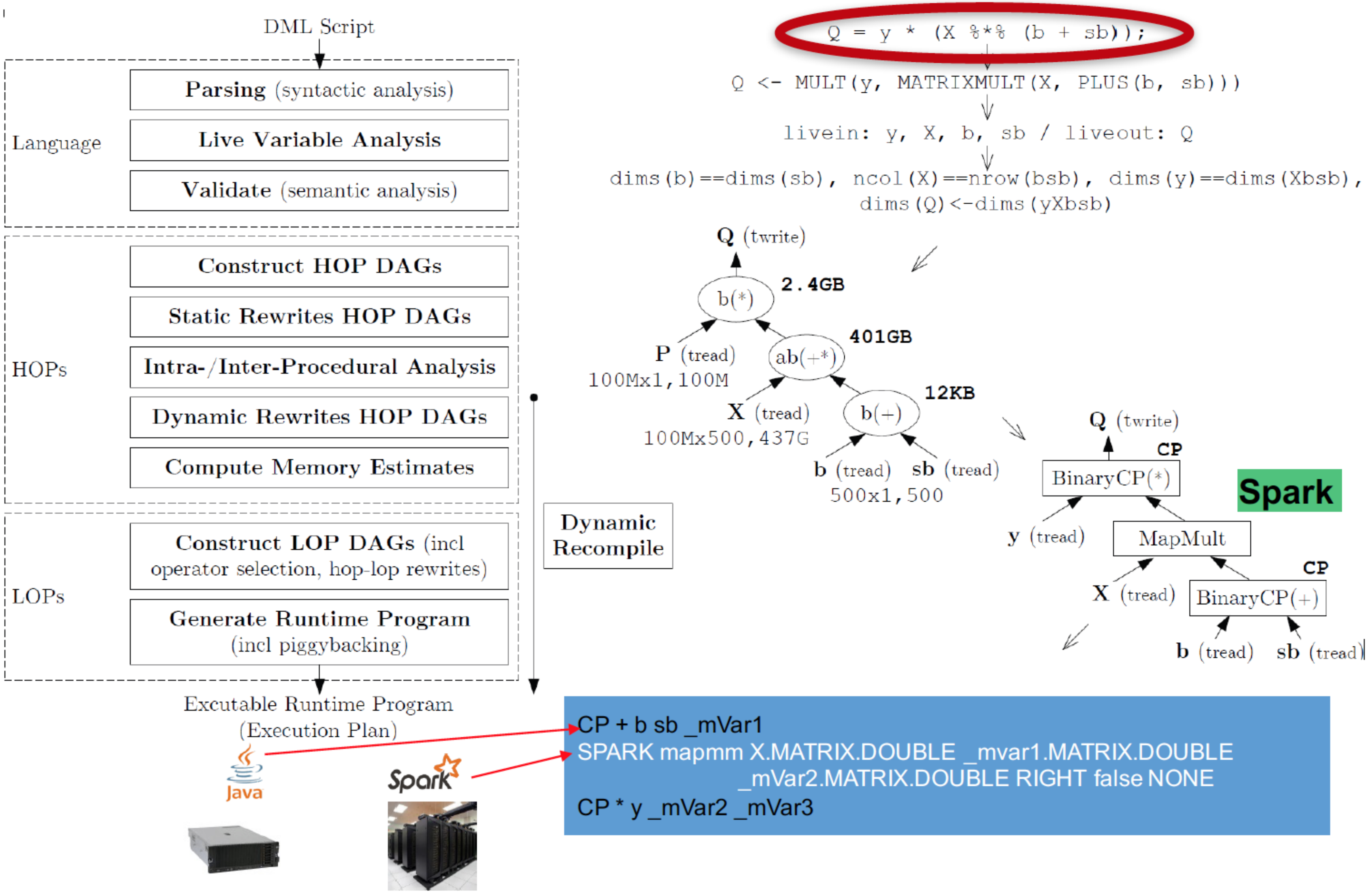
- Dataflow in DAGs of operations on matrices, frames, and scalars
- Choosing from alternative execution plans based on memory and cost estimates
- Operator ordering & selection; hybrid plans

SystemML Compilation Chain



- Low-level physical execution plan (LOPDags)
  - Over key-value pairs for MR
  - Over RDDs for Spark
- “Piggybacking” operations into minimal number Map-Reduce jobs

SystemML Compilation Chain



Problems we are going to Discuss- Matrix Multiplication

- For Matrix Multiplication System ML offers two alternative execution plans
  - RMM-Replication based Matrix Multiplication- Requires only one Map-Reduce Job
  - CPMM: Cross Product based Matrix Multiplication- Requires Two Map-reduce jobs

RMM-Replication based Matrix Multiplication

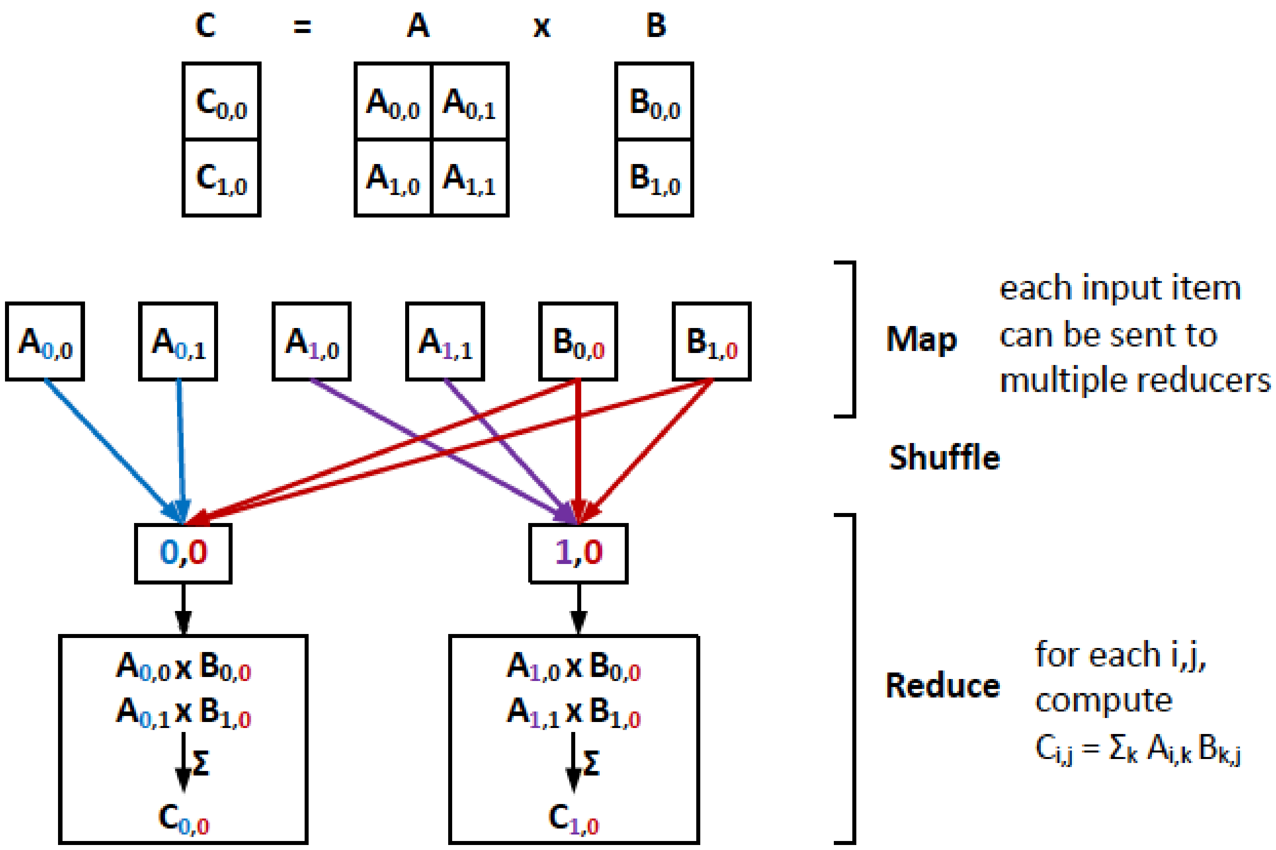
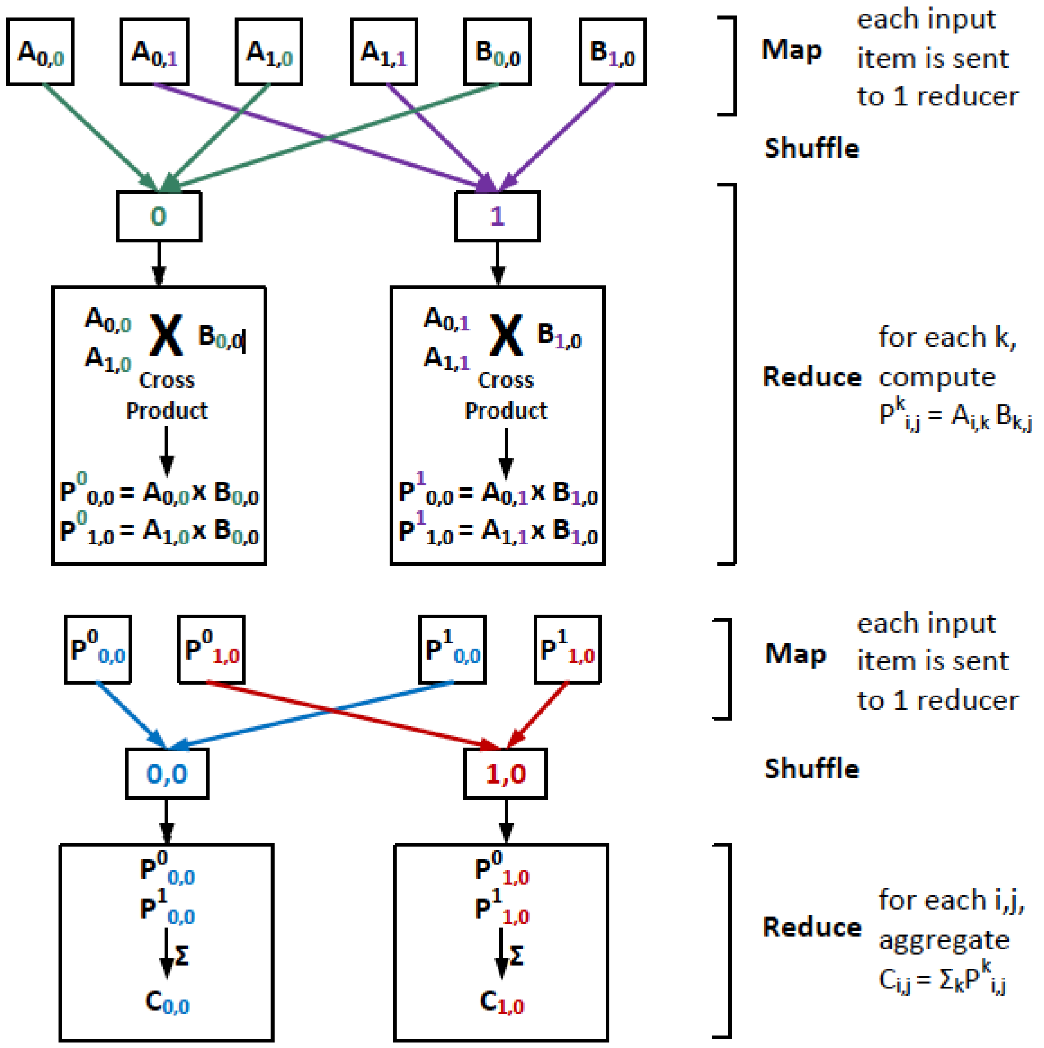


Fig. 1. RMM: Replication based Matrix Multiplication

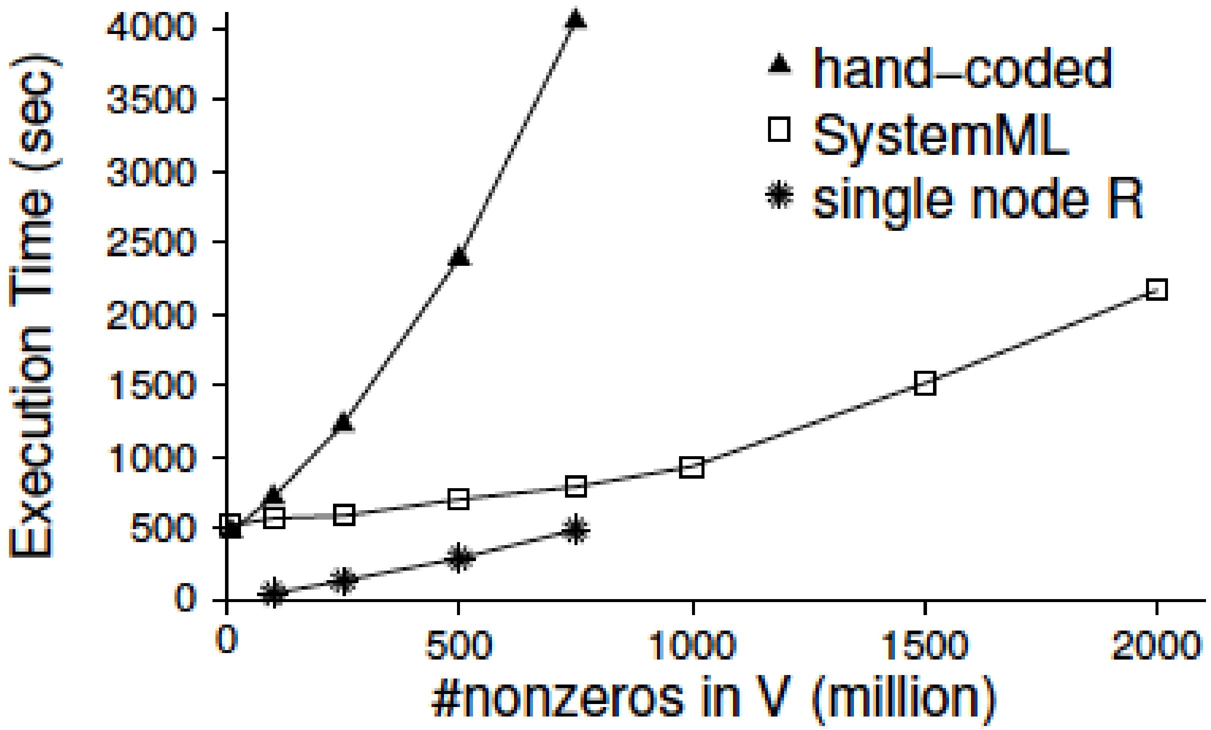
CPMM: Cross Product based Matrix Multiplication



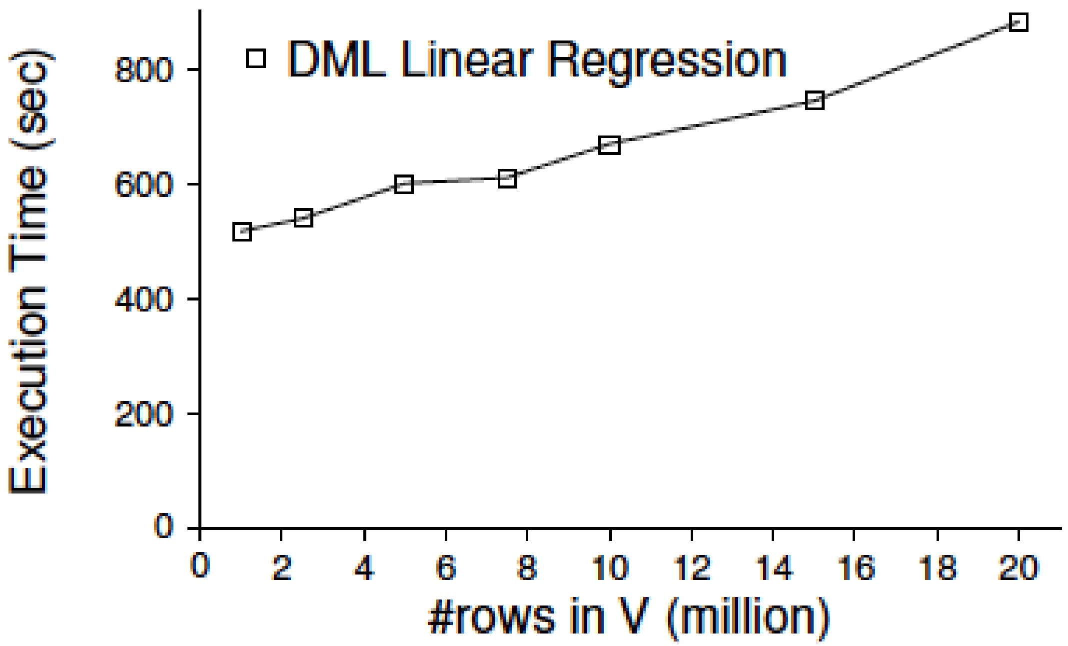
SystemML vs GNMF (GaussianNon-Negative Matrix Factorization)

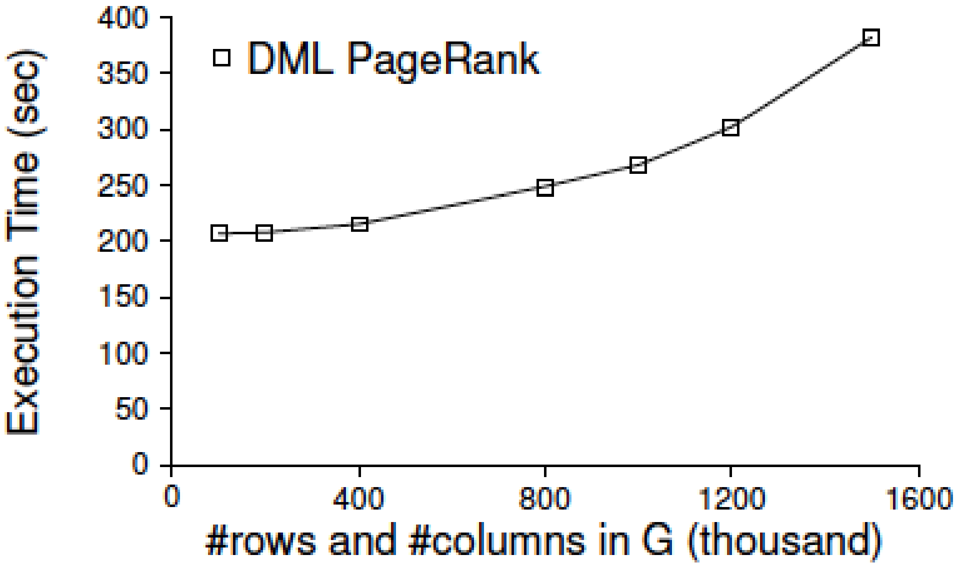
Dataset is Sparse Matrix-
Calculating Time consumed for Matrix Multiplication
Increasing Data Size (V) in 40 Core Cluster
Methods Used:

- Hand-Coded GNMF
- SystemML
- Single Node R

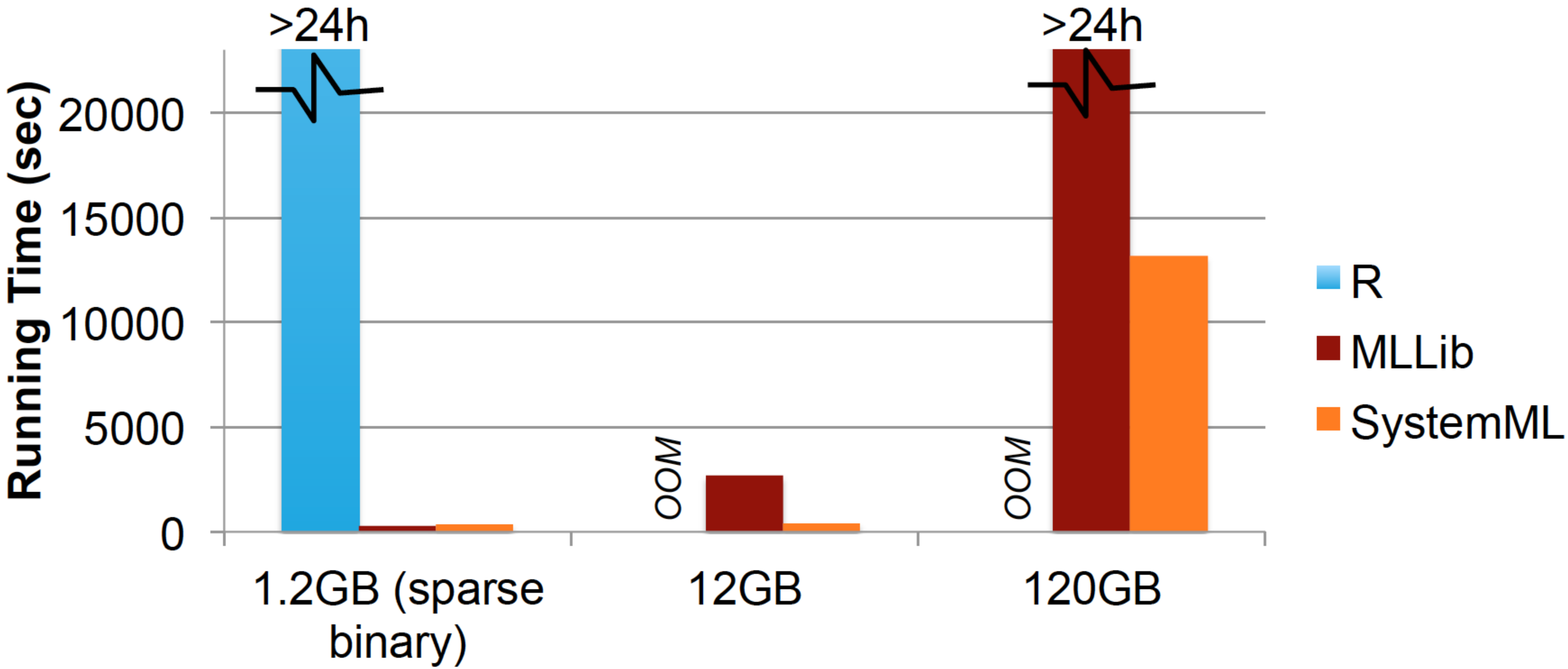


Scaling in SystemML-
Linear Regression and Page Rank



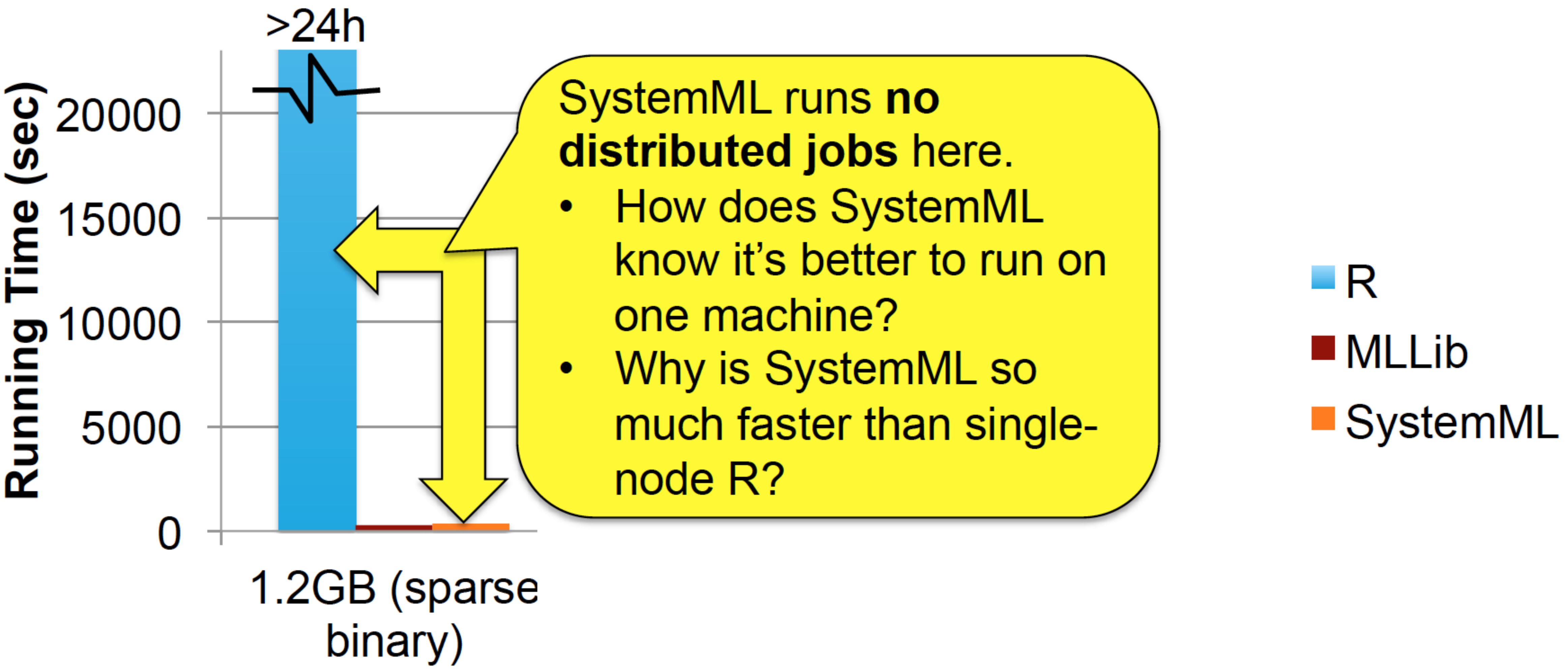


Performance Comparison- Alternating Least Square



Details: Synthetic data, 0.01 sparsity, 10^5 products × {10^5,10^6,10^7} users. Data generated by multiplying two rank-50 matrices of normally-distributed data, sampling from the resulting product, then adding Gaussian noise. Cluster of 6 servers with 12 cores and 96GB of memory per server. Number of iterations tuned so that all algorithms produce comparable result quality.

Performance Comparison- Alternating Least Square



Recap

Questions

- How does SystemML know it's better to run on one machine?
- Why is SystemML so much faster than single-node R?

Answers

- Live variable analysis
- Propagation of statistics

- Advanced rewrites
- Efficient runtime

Benefits of the SystemML Approach

Simplifies algorithm development

It can compile and run algorithm at scale

No additional performance code needed!

Your code gets faster as the system improves

Question?